EPOCH

Une histoire de l'informatique

STÉPHANE FOSSE

EPOCH

Une histoire de l'informatique

Stéphane FOSSE

Décembre 2025

ISBN : 978-29-59208-94-2
ISBN (PDF) : 978-29-59208-95-9
ISBN (EPUB) : 978-29-59208-96-6

Copyright © 2025 - Stéphane Fosse
Auto-édition - https ://fosse.fr
Deuxième édition, décembre 2025
Dépôt légal : décembre 2025

Cet ouvrage est mis à disposition selon les termes de la Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 4.0 International.

Vous pouvez consulter une copie de cette licence en fin d'ouvrage, ou à l'adresse : https ://creativecommons.org/licenses/by-sa/4.0/deed.fr

Vous êtes libre de partager et d'adapter ce livre à condition de me créditer et de redistribuer vos modifications sous la même licence (CC BY-SA 4.0).

Les humains sont allergiques au changement. Ils adorent dire : « On a toujours fait comme ça. » J'essaie de lutter contre cette tendance. C'est pour ça que j'ai une horloge qui tourne dans le sens inverse des aiguilles d'une montre sur mon mur.

Grace Hopper

Table des matières

Avant-propos **xvii**
Avant 1930 **1**
Système binaire 4
Carte perforée 6
Charles Babbage 8
Algèbre de Boole 9
Câble sous-marin 11
Herman Hollerith 13
1930 **15**
L'analyseur différentiel de Vannevar Bush 17
Le théorème d'incomplétude de Gödel 18
La machine de Turing 20
L'ordinateur mécanique Z1 22
Lambda-calcul 24
Circuit logique de Shannon 25
Le Complex Number Calculator 27
1940 **31**
Ordinateur d'Atanasoff et Berry 34
Réseaux de neurones artificiels 35
Colossus 38
Harvard Mark I 39
ENIAC 41
Mémoire à ligne de retard. 43
Méthode de Monte Carlo 45
Tube Williams 46
Association for Computing Machinery 48

Transistor 50
Cybernétique 52
Tambour magnétique 54
Théorie de l'information de Claude Shannon 56
Short Code 57
EDSAC 59
Application Programming Interface 60
Whirlwind I. 62
1950 **65**
Code de Hamming 68
Mémoire à tores magnétiques 70
Bande magnétique 71
Ferranti Mark I 73
LEO I 75
UNIVAC I 77
Compilateur A-0 79
IBM 701 81
IBM 702 83
Assembleur 85
IBM 650 87
IBM 704 89
Conférence de Dartmouth. 91
IBM 350 RAMAC 93
IPL 95
TX-0 97
FORTRAN 99
Circuit intégré 101
ALGOL 103
Lisp 105
DEC PDP-1 107
IBM 1401 108
COBOL 111
Apprentissage automatique 113
Schéma de Backus-Naur 115

1960 **119**
Quicksort 121
Bull Gamma 60 123
APL 125
Simula. 127
SNOBOL. 129
Spacewar ! 130
ASCII 132
Modem Bell 103 134
Music V 136
Sketchpad 138
GECOS 140
IBM System/360 142
BASIC 144
PL/I 146
IBM CP-40 149
Courrier électronique 151
Multics 152
DEC PDP-8 154
CDC 6600 156
ELIZA 158
Information Management System 159
LOGO. 161
Read Only Memory 163
NLS 165
ARPANET 167
B 169
Premier RFC 171
RAM 172
UNIX 174
1970 **177**
ALOHAnet 180
FORTH 182
Pascal 184
DEC PDP-11 186
B-tree 188

Disquette. 190
ed 192
File Transfer Protocol 194
Intel 4004 196
Intel 8008 198
C 200
Prolog 202
Smalltalk. 205
IBM VM/370 207
Ethernet 209
TCP/IP 210
Telnet 212
Le langage SQL 215
System R. 216
Intel 8080 219
CP/M 221
Diffie-Hellman 224
Altair 8800 226
X.25 228
Cray-1. 229
UUCP. 232
Apple I 234
vi 236
DEC VAX-11 238
RSA 240
Apple II 242
Commodore P.E.T 244
Tandy TRS-80. 246
Berkeley Software Distribution 248
Space Invaders. 250
TEX 252
WordStar 254
dBASE 255
Ada. 258
Usenet. 260
VisiCalc 262

1980 **267**

Apple III 269
Intel 8086/8088 272
Smalltalk-80 274
UDP 276
Commodore VIC-20 278
WordPerfect 280
ICMP 282
Acorn BBC Micro 284
IBM PC-5150 286
MS-DOS 288
Osborne 1 290
LZW 292
AutoCAD 294
CD-ROM 296
Intel 80286 298
SMTP 300
Commodore 64 302
Sinclair ZX Spectrum 304
Thomson TO/MO 307
PostScript 309
SunOS. 311
Agat 312
Apple Lisa 315
Coleco Adam 317
Compaq Portable 320
GNU Project 322
DNS 324
C++ 327
Objective-C 329
IPX/SPX 331
SMB 333
Lotus 1-2-3 335
Microsoft Word 337
MSX 339
Novell Netware 341

Oric-1 343
Amstrad CPC 464 345
Apple Macintosh 347
HP-UX 349
NFS 351
POP3 353
IBM PCjr 355
GNU Emacs 357
Microsoft Paint 359
X Window System 361
Intel 80386 363
AppleTalk 365
NTP 368
Microsoft Excel 370
Microsoft Windows 372
Amstrad PCW. 374
Atari 520 ST 376
Toshiba T1100. 377
Aldus PageMaker. 379
IBM AIX. 381
Eiffel 383
Erlang. 385
IMAP 387
NSFNet 389
SCSI 391
Bash 393
GIF. 395
Perl. 397
Self 399
Tcl 401
Acorn Archimedes 403
Apple Macintosh II 405
Commodore Amiga 500 407
Sharp X68000 409
OS/2 411
Unicode 412

FrameMaker 414
IBM AS/400 416
Internet Relay Chat 418
SNMP 421
ZIP 423
Intel 80486 425
Intel i960 426
World Wide Web 428
Microsoft Office 430
NeXTSTEP 432
1990 **435**
Adobe Photoshop 437
IBM PS/1 439
Haskell 441
HTML 443
Gopher 445
PCMCIA 447
MP3 449
WAVE 452
Python 454
Linux 456
Visual Basic 458
DEC OSF/1 460
Blowfish 462
JPEG 465
IBM ThinkPad 468
OpenGL 470
IBM Simon 472
Adobe PDF 474
Common Gateway Interface 476
FreeBSD 478
Intel Pentium 480
Lua 482
R 484
DHCP 486
NCSA Mosaic 488

Secure Socket Layer 489
Microsoft Windows NT 3.51 491
Apple Power Macintosh 493
CD-RW 495
IBM Aptiva 497
Netscape Navigator 499
World Wide Web Consortium 501
Yahoo ! 503
AltaVista 504
Apache Web Server 506
BeOS 508
DVD 510
GPU 512
Microsoft Internet Explorer 513
Java 515
JavaScript 517
PHP 519
Ruby 521
SSH 522
Lynx 524
MySQL 526
RealPlayer 529
Sun UltraSPARC 530
Wiki 532
Microsoft Windows 95 533
Microsoft DirectX 535
ICQ 537
KDE 538
Cascading Style Sheets 540
Flash 542
OpenBSD 543
PostgreSQL 545
USB 547
VPN 549
Hotmail 551
Internet Archive 553

Wi-Fi 554
Google 556
VMware 558
Microsoft Windows 98 560
XML 562
GNOME 564
Bluetooth 566
MSN Messenger 567
RSS 569
SOAP 571
Transport Layer Security 573
Napster 574
Bug de l'an 2000 576
2000 **579**
C# 581
FLAC 583
JSON 585
BitTorrent 587
Apple Mac OS X 589
SHA-256 591
Microsoft .NET Framework 593
Mozilla Firefox. 595
Apple Xserve 596
Tor 598
Apple iTunes 600
Skype 602
WordPress 603
Xen. 606
Apache Cassandra 608
Google Gmail 610
Scala 612
NGINX 614
OAuth. 616
Git 617
Redis 619
YouTube 621

Amazon Web Services 623
Hadoop 624
jQuery. 626
PowerShell 628
MongoDB 630
Apple iPhone 632
Clojure 634
F# 636
Groovy 638
Android 640
Google Chrome 642
HTML5 644
Memcached 645
OpenStack 647
Bitcoin 649
CouchDB. 651
Node.js 653
Raspberry Pi 655
2010 **657**
Elasticsearch 660
HEIF 662
Rust 664
ChromeOS 666
NVMe 668
Dart 670
Kotlin 672
WebRTC 674
WebSocket 676
Go 678
Julia 680
TypeScript 682
Docker 684
SHA-3 687
Spark 689
Kubernetes 691
Swift 693

Kafka 695
GraphQL. 696
TensorFlow 699
Accelerated Mobile Pages 701
Natural Language Processing 703
WebAssembly 704
WebAuthn 706
2020 **709**
Apple M1 712
Zig 714
Mojo 716
WebGPU. 718
Épilogue **723**
Licence **761**

Avant-propos

Mon premier contact avec un ordinateur remonte à un ORIC-1 en 1983. Suivait de près un Apple IIc, lors des tous premiers cours d'initiation à l'informatique à l'école primaire. J'ai eu cette chance. LOGO et BASIC ont été mes premières fenêtres sur la programmation. À l'époque, on achetait des livres remplis de code source qu'il fallait recopier ligne après ligne, sauvegarder sur cassette, puis déboguer : `syntax error`. Acheter les programmes compilés sur cassette aurait coûté une fortune. Avec tous ces nouveaux mots, je me souviens aussi m'être demandé quelle différence il y avait entre un logiciel et Loriciel, cette société française qui éditait des jeux vidéo créée en 1983. Tout était confus.

Des décennies plus tard, devenu architecte en systèmes d'information, je manipule quotidiennement des technologies dont j'ignorais souvent l'origine. Je savais comment elles fonctionnaient, comment les configurer et les assembler. Mais le contexte originel de leur création restait dans l'ombre. Alors je me suis posé les mêmes types de questions que je pose quotidiennement dans mon travail. À quels problèmes étaient confrontés celles et ceux qui les ont inventées ? Quels besoins ou quelles frustrations ont poussés à chercher des solutions nouvelles ? Ces questions restaient sans réponse.

J'ai donc commencé à lire et compiler des articles, des documents officiels de constructeurs, des scripts de conférences internationales, des normes ISO ou des RFC. Plus de 3750 documents passés au crible. Petit à petit, le volume a pris forme. Assez pour tenir dans un livre. Cette compilation n'était plus seulement une curiosité personnelle mais devenait un projet de transmission.

Sur internet, 38% des pages web qui existaient en 2013 sont désormais inaccessibles. L'information se délite. Fixer ce savoir dans un livre, sous licence libre qui plus est, représente pour moi une façon de préserver ce

qui risque de disparaître. Le *copyleft* me semblait la réponse la plus appropriée : le savoir doit circuler, se partager, s'enrichir au fil des lectures et des réutilisations.

J'ai donc choisi d'aborder chaque technologie sous l'angle du problème à résoudre. En abordant le contexte qui amène à l'innovation, on peut montrer que l'histoire de l'informatique n'est pas une succession de prouesses techniques abstraites. C'est une histoire humaine, faite de contraintes concrètes et de réponses souvent pragmatiques.

Le livre se construit autour de 350 technologies, présentées chronologiquement sous forme de chroniques. Chacune peut se lire indépendamment des autres, gardant son unité pour que le lecteur puisse naviguer à sa guise. L'idée était de rendre l'ouvrage utilisable comme une ressource qu'on consulte selon ses intérêts du moment, ou tout simplement dans l'ordre en suivant la ligne du temps.

Cette deuxième édition s'enrichit de plus de 250 technologies. Les choix reflètent forcément mes propres sensibilités et mon parcours. Tout raconter serait une utopie. Il a fallu trancher, sélectionner, renoncer, et il y a encore tellement de choses à raconter. Les technologies apparues après 2010 sont trop récentes selon moi pour qu'on puisse vraiment juger de leur poids historique. Le recul manque. C'est la raison pour laquelle elles sont moins représentées dans ces pages.

Ce livre s'adresse à plusieurs publics. Tout d'abord, les passionnés d'histoire des sciences et techniques y trouveront matière à explorer. Les professionnels de l'innovation, les consultants, ceux qui travaillent en recherche et développement y verront des patterns d'évolution qui se répètent. Les enseignants et formateurs en informatique pourront y puiser une ressource pour leurs cours. Les étudiants en écoles d'ingénieurs ou à l'université disposeront d'un matériau d'apprentissage structuré. Les curieux intellectuels attirés par les récits de découvertes et d'inventeurs ne seront pas en reste. Et les investisseurs ou entrepreneurs du secteur technologique y découvriront peut-être les cycles d'innovation qui préfigurent les ruptures à venir.

EPOCH, au sens informatique du terme, désigne un point de départ à partir duquel le temps commence à se compter. Ce livre se veut un point de repère dans la compréhension de notre héritage technologique. Un repère parmi d'autres, certes, mais un repère qui fixe une histoire.

I

Avant 1930

L'aube de l'informatique

La période de 1700 à 1930 bouleversa radicalement notre humanité. Ces deux siècles, marqués par l'industrialisation et ses secousses sociales, constituèrent les bases de l'informatique qui allait naître.

Le XVIII^{e} siècle vit l'Europe s'enflammer d'une fièvre intellectuelle sans précédent. Les Lumières imposèrent une vision du monde où science et raison prirent le trône. Des inventeurs, habités par une curiosité frénétique, cherchèrent à dompter l'univers par les nombres et les mécanismes. Les académies des sciences se multiplièrent comme autant de ruches bourdonnantes d'idées nouvelles, tissant leurs réseaux à travers le vieux continent.

Vers 1760, la Grande-Bretagne fut secouée par les premières convulsions de la révolution industrielle. Les usines naissantes avalèrent les travailleurs et les machines remplacèrent peu à peu les mains. Les industriels, confrontés à la complexité grandissante de leurs entreprises, réclamèrent des outils précis pour jongler avec leurs chiffres. Cette soif de précision arithmétique poussa les inventeurs à imaginer des solutions mécaniques aux calculs les plus laborieux.

La fin du siècle assista à l'éclosion des premiers dispositifs mécanisés dans les filatures. Joseph-Marie Jacquard transforma le tissage en 1801 avec ses cartes perforées qui dictaient aux métiers les motifs à reproduire.

Ces rectangles de carton troués, qui semblaient si anodins, portaient en eux le germe des futurs systèmes informatiques. Chaque trou, chaque plein représentait une information binaire que la machine interprétait sans erreur, une préfiguration saisissante des mémoires numériques.

Le XIX^{e} siècle s'ouvrit dans un tourbillon de changements. Les empires étirèrent leurs tentacules aux quatre coins du globe, le commerce s'intensifia, créant un besoin viscéral de communications rapides. Le chemin de fer, ce monstre d'acier qui dévorait l'espace, imposa la synchronisation des horloges et la gestion rigoureuse des horaires. Les États modernes gonflèrent leurs administrations, qui se noyèrent bientôt sous les montagnes de paperasse.

George Boole, mathématicien anglais à l'esprit incisif, publia en 1847 sa théorie de l'algèbre logique. Semblant n'intéresser que quelques savants excentriques, ses travaux contenaient la clé qui déverrouillerait un jour les portes de l'informatique. Son système binaire, réduisant toute proposition à vrai ou faux, attendait patiemment que des esprits visionnaires s'en saisissent.

Le télégraphe électrique, fils de cuivre tendus d'un continent à l'autre, pulvérisa les barrières du temps et de l'espace après 1840. Ce qui prenait des mois ne demandait désormais que des minutes. Samuel Morse imposa son code de points et de traits, première standardisation du langage binaire. Les câbles sous-marins, serpents de cuivre posés au fond des océans, connectèrent l'Europe à l'Amérique dès 1866, créant le premier réseau mondial de communication instantanée.

L'adoption du temps universel en 1884 lors de la Conférence internationale du méridien de Washington illustra parfaitement comment la technologie remodela notre perception du monde. Les 24 fuseaux horaires imposèrent une discipline temporelle inédite à l'humanité. Le monde entier régla ses montres sur Greenwich, soumis à la tyrannie douce des horloges synchronisées.

Les dernières décennies du XIX^{e} siècle virent les entreprises croître jusqu'à devenir des monstres bureaucratiques. Banques, assurances et grands magasins croulèrent sous les registres et les fiches. Le recensement américain de 1890 illustra ce problème : l'administration fédérale mit sept ans à dépouiller les données du recensement précédent. Herman Hollerith, jeune ingénieur surdoué, conçut sa machine à tabuler électromagnétique. Grâce à elle, les résultats du recensement de 1890 furent disponibles en seulement six semaines. Sa Tabulating Machine Company, fondée en 1896, deviendrait plus tard IBM.

La Grande Guerre transforma radicalement le rapport aux technologies de l'information. Dans l'ombre des tranchées, mathématiciens et linguistes

brisèrent les codes ennemis. En France, le lieutenant Georges Painvin déchiffra le code ADFGVX allemand en 1918, déjouant l'offensive du Printemps et sauvant peut-être Paris. Les calculs balistiques, nécessaires pour l'artillerie à longue portée comme le Paris-Geschütz allemand, exigeaient une précision sans précédent. La guerre avait montré que la victoire appartenait à ceux qui maîtrisaient l'information.

L'entre-deux-guerres vit naître de nouvelles méthodes d'organisation du travail. Frederick Taylor disséqua les gestes ouvriers au chronomètre tandis que Henry Ford imposa sa chaîne de montage. Le bureau suivit l'usine dans cette course à la rationalisation. Les machines comptables Burroughs, les caisses enregistreuses National, les calculatrices Monroe envahirent les administrations. Le bruit des touches, le carillon des totaux, le glissement des chariots rythmèrent désormais le quotidien des cols blancs.

Les années 1920 virent l'explosion des télécommunications. La TSF (Télégraphie Sans Fil) porta les voix et la musique dans chaque foyer. La BBC (British Broadcasting Corporation) émit ses premiers programmes réguliers en 1922, suivie par de nombreuses stations à travers le monde. Le téléphone, invention de Graham Bell en 1876, quitta le statut de gadget pour les riches et commença sa conquête des foyers. Les standardistes, majoritairement des femmes, devinrent les premières interfaces humaines entre les machines et leurs utilisateurs, métaphore saisissante des futures relations homme-machine.

Les mutations sociales accompagnèrent ces évolutions technologiques et l'alphabétisation massive changea le rapport aux connaissances. En France, les lois Jules Ferry de 1881-1882 imposèrent l'école obligatoire. Aux États-Unis, le taux d'alphabétisation dépassa 80% dès 1870. Cette population instruite demanda toujours plus de livres, de journaux, d'informations. Le savoir devint une marchandise qui circulait désormais à la vitesse de l'électricité.

L'urbanisation galopante créa des difficultés inédits d'organisation. Paris passa de 500 000 à 3 millions d'habitants entre 1800 et 1930. New York explosa de 60 000 à 7 millions d'âmes pendant cette période. Ces métropoles tentaculaires exigèrent des infrastructures colossales. Le baron Haussmann redessina Paris, imposant ses avenues rectilignes aux vieux quartiers médiévaux. À New York, l'ingénieur John Augustus Roebling jeta son pont suspendu entre Manhattan et Brooklyn en 1883. Ces prouesses techniques imposèrent des calculs d'une précision diabolique et une gestion méthodique des ressources.

Le monde du travail se métamorphosa sous ces influences croisées. Le secteur tertiaire gonfla jusqu'à dominer l'économie américaine dès les années 1920. Les travailleurs de l'information, comptables, sténographes, opé-

rateurs, formèrent une nouvelle classe sociale. Les femmes envahirent ces professions, modifiant les rapports de genre dans la société. La dactylographe, penchée sur sa Remington, devint une figure emblématique de cette modernité bureaucratique.

En 1929, à la veille du krach boursier qui allait ébranler l'économie mondiale, l'informatique restait encore à naître officiellement. Pourtant, tous les ingrédients de sa future explosion étaient réunis. Les besoins en calcul s'intensifiaient dans tous les domaines. Les théories mathématiques nécessaires avaient mûri. L'électronique balbutiante promettait des performances inouïes. Le terrain était prêt pour l'avènement des calculateurs programmables qui allaient bientôt transformer le monde plus radicalement que la machine à vapeur ne l'avait fait un siècle plus tôt.

Système binaire

Le parcours historique du système binaire révèle comment cette idée mathématique s'est muée en fondement de notre informatique. Thomas Harriot, mathématicien anglais, esquissa vers 1600 les premiers traits d'une représentation numérique basée uniquement sur 0 et 1. Ses recherches sur les combinaisons l'amenèrent à constater que tout nombre pouvait s'exprimer comme somme de puissances de 2. Ces travaux, jamais publiés, restèrent dans l'ombre jusque dans les années 1920.

Un siècle plus tard, Gottfried Wilhelm Leibniz redécouvrit ce système par hasard. Dans les années 1670, alors qu'il planchait sur la division des nombres et les nombres premiers, la notation binaire lui apparut comme une solution élégante pour certains calculs mathématiques. Au début, il l'utilisait simplement pour illustrer ses théorèmes sans imaginer son usage pratique. Dans sa lettre de 1697 au Duc de Brunswick, Leibniz propose de créer une médaille commémorative et développe une lecture théologique du binaire où le 1 symbolise l'être, le 0 le néant. Cette médaille ne vit jamais le jour, mais la lettre lança une série de publications sur ce système.

En 1703, dans les *Mémoires de l'Académie Royale des Sciences*, Leibniz publia son « Explication de l'arithmétique binaire ». Ce texte exposait les principes du système et montrait comment réaliser les opérations de base tout en admettant la lourdeur de ces longues séquences de 0 et de 1 pour un usage quotidien.

Le XVIII^{e} siècle vit des mathématiciens comme Jean Bernoulli et Leonard Euler s'emparer du binaire. Euler l'utilisa notamment pour étudier

les propriétés des nombres de type $2^n + 1$. D'autres chercheurs tissèrent des liens entre ce système et d'autres bases numériques comme l'octal ou l'hexadécimal.

L'application pratique du binaire apparut au XIX^{e} siècle avec l'avènement des télécommunications. Le télégraphe de Morse, breveté en 1837, s'appuyait sur deux états électriques, présence ou absence de courant, pour transmettre l'information. Son code associait points et traits, préfigurant l'usage futur du binaire dans les communications numériques. Émile Baudot franchit un pas supplémentaire en 1870 avec son code télégraphique explicitement binaire. En 1901, Charles Bouton démontra l'utilité du binaire pour analyser le jeu de Nim, ouvrant la voie à son application dans la théorie des jeux.

La révolution vint avec l'électronique du XX^{e} siècle. Le circuit *flip-flop* d'Eccles-Jordan, créé en 1919, donna une existence physique aux états 0 et 1. Cette invention capitale rendit possibles les premiers circuits de mémoire binaire. En 1937, Claude Shannon démontra dans sa thèse le lien entre l'algèbre booléenne et le système binaire pour concevoir des circuits de commutation électroniques.

Les premiers ordinateurs comme l'ENIAC (1946) fonctionnaient encore en base décimale, avec dix tubes à vide par chiffre. Le rapport Burks-Goldstine-von Neumann de 1947 changea la donne en montrant les atouts du binaire : circuits plus simples, fiabilité accrue, harmonie naturelle avec les opérations logiques. Cette recommandation influença tous les ordinateurs suivants.

Les années 1950-1960 virent l'essor des mémoires électroniques, renforçant l'hégémonie du binaire. Les mémoires à tores magnétiques puis à semi-conducteurs reposaient par nature sur deux états distincts. Certains constructeurs tentèrent d'explorer des mémoires à quatre ou seize niveaux dans les années 1970-1980, Intel expérimenta des ROM à quatre niveaux dans certains processeurs, mais ces tentatives restèrent marginales face à la robustesse du stockage binaire.

Le binaire s'étendit au-delà du matériel informatique. Les télécommunications numériques l'adoptèrent pour leur résistance au bruit et leur capacité à régénérer les signaux. Les supports optiques comme les CD-ROM et DVD-ROM l'utilisèrent aussi. Sa réussite tient à sa simplicité conceptuelle qui rend les circuits plus fiables, sa compatibilité avec l'algèbre de Boole qui en fait le langage naturel des opérations logiques, et sa robustesse face aux perturbations qui s'explique par la distinction de seulement deux états.

Les recherches sur d'autres systèmes numériques n'ont pas disparu pour autant. Le ternaire (à trois états) fit l'objet d'études poussées en Union soviétique dans les années 1950. Plus récemment, l'informatique quantique

a ouvert des horizons nouveaux avec ses *qubits* capables d'exister dans une superposition d'états. Malgré ces alternatives, le système binaire demeure l'âme de l'informatique classique. Sa capacité à coder toute information avec deux symboles uniquement lui confère une universalité remarquable.

Carte perforée

En 1801, Joseph-Marie Jacquard, tisserand lyonnais, cherchait un moyen d'automatiser ses métiers à tisser. Il inventa des cartes en carton perforé pour encoder les motifs de tissage. Sans le savoir, il venait de poser la première pierre d'une idée majeure. Ces cartons n'étaient pas destinés aux calculs, mais ils matérialisaient déjà le concept puissant de transformer une instruction en trous lisibles par une machine.

C'est presque un siècle plus tard, en 1890, que le Bureau du Recensement américain fait face à un problème colossal. La population croît à une vitesse folle et le précédent recensement s'était étalé sur huit années. Un statisticien nommé Herman Hollerith proposa alors la solution audacieuse d'adapter le principe de Jacquard au comptage des personnes. Son système réduisit le temps de traitement à deux ans et demi seulement. Un gain stupéfiant qui ne passa pas inaperçu à l'époque. Le succès fut tel que Hollerith fonda sa propre entreprise en 1896, la Tabulating Machine Company. Celle-ci fusionnerait plus tard avec d'autres pour donner naissance à un géant que nous connaissons tous : IBM.

Au début du XX^{e} siècle, la carte perforée s'installa dans le paysage administratif des grandes organisations. Finie la comptabilité manuelle ! Les trésoriers et comptables s'émerveillaient devant ces machines capables de trier, compter et additionner en un temps record. La gestion des stocks, les calculs de paie, tout y passait.

Les premières cartes de Hollerith ne comptaient que 45 colonnes, dimension rapidement jugée insuffisante. En 1928 chez IBM, Clair D. Lake remplaça les trous circulaires par des trous rectangulaires, permettant ainsi de passer de 45 à 80 colonnes sur la même surface. Une innovation qui marquera durablement l'informatique.

Cette norme des 80 colonnes a laissé une empreinte surprenante sur notre monde numérique actuel. Quand IBM conçut ses premiers terminaux dans les années 1960, comme le fameux 3270, chaque ligne affichait exactement 80 caractères. Un choix dicté par la compatibilité avec l'existant. Cette contrainte technique s'est propagée comme une onde de choc à travers

les décennies. Aujourd'hui encore, de nombreux éditeurs de code limitent par défaut les lignes à 80 caractères, ou du moins le proposent. Les émulateurs de terminal perpétuent cette tradition. Les guides de bonnes pratiques en programmation recommandent cette limite pour améliorer la lisibilité. L'ombre de la carte perforée plane toujours sur nos écrans.

Entre les deux guerres mondiales, une véritable industrie du traitement mécanographique des données se développa. IBM dominait aux États-Unis, tandis qu'en Europe, des sociétés comme Powers (future Remington Rand) ou Bull en France tentaient de se tailler une part du marché. Leur modèle économique reposait sur la location des machines plutôt que sur leur vente, assurant des revenus stables et un contrôle rigoureux sur cette technologie stratégique.

Cette nouvelle technique créa de nouveaux métiers. Les salles de mécanographie s'animaient sous les doigts agiles des perforatrices, majoritairement des femmes, qui transformaient les données brutes en cartes perforées avec une précision d'orfèvre. D'autres vérifiaient chaque carte, pendant que les opérateurs réglaient les trieuses et tabulatrices. Ces départements devinrent le cœur battant des grandes organisations.

L'apogée de la carte perforée survint dans les années 1950-1960. Après l'arrivée des premiers ordinateurs électroniques, elle demeura le support privilégié pour l'entrée et le stockage des données. Les premiers modèles d'IBM, comme le 650 ou le 1401, furent d'ailleurs conçus pour s'intégrer aux installations existantes de cartes perforées, accompagnant en douceur la transition vers l'ère électronique.

L'histoire de cette technologie comporte aussi ses zones d'ombre. Durant la Seconde Guerre mondiale, les systèmes à cartes perforées servirent au recensement et au contrôle des populations en Europe occupée. La Deutsche Hollerith-Maschinen Gesellschaft, filiale allemande d'IBM, fournit des machines utilisées pour organiser la déportation. Un rappel glaçant que toute technologie peut aussi être utilisée à des fins terribles.

Le déclin s'amorça dans les années 1970. L'arrivée des terminaux interactifs et des supports magnétiques signa progressivement l'arrêt de mort de ces rectangles de carton. La carte perforée a établi le principe du stockage binaire de l'information (trou ou pas trou, 1 ou 0), séparé clairement le support de données de la machine de traitement, et montré l'importance de la standardisation. Les méthodes organisationnelles développées autour d'elle ont influencé la conception des premiers ordinateurs et structuré l'organisation du travail dans les centres de calcul.

Charles Babbage

Le jeune Charles Babbage voit le jour dans un foyer fortuné de Londres en 1791. Son parcours à Cambridge, où il obtient une formation mathématique rigoureuse, lui fait prendre conscience des limites du calcul manuel. Avec quelques camarades, il fonde la Société Analytique pour moderniser l'enseignement des mathématiques à l'université. Cette première expérience traduit déjà son désir de renouveler les méthodes établies.

Un soir de 1821, Charles discute avec son ami John Herschel. Tous deux s'épuisent à vérifier des tables mathématiques truffées d'erreurs. « Imaginez ces calculs réalisés par une machine à vapeur », lance Babbage. Cette réflexion, jetée dans la conversation, marque le début d'une aventure intellectuelle extraordinaire.

Babbage se met au travail. Sa première création, la machine à différences, doit calculer automatiquement des polynômes grâce à la méthode des différences finies. Le prototype de 1822 fonctionne assez bien pour convaincre le gouvernement britannique de financer une version plus ambitieuse. Un vrai tour de force puisque l'appareil ne se contente pas d'effectuer les opérations, il imprime aussi les résultats, évitant toute erreur de transcription.

Les roues dentées s'emboîtent dans un ballet mécanique complexe. Mais les artisans londoniens, malgré leur savoir-faire, ne parviennent pas à fabriquer des pièces avec la précision requise. Le projet s'enlise. Babbage, loin de se décourager, se tourne vers une idée plus audacieuse encore.

En 1834, sa machine analytique prend forme sur le papier. Cette nouvelle machine ne suit plus un programme fixe mais peut être reprogrammée pour différents calculs. Son architecture ressemble étrangement aux ordinateurs du XX^{e} siècle : une unité de calcul (le moulin), une mémoire (le magasin), des dispositifs d'entrée et de sortie de données.

Fasciné par les métiers à tisser Jacquard, Babbage leur emprunte le principe des cartes perforées pour commander sa machine. Ces cartes définissent à la fois les données et les instructions à exécuter. La machine analytique intègre tout ce qui fait l'essence de l'informatique moderne, c'est-à-dire le programme stocké, les boucles conditionnelles et les sous-programmes. Elle compare des nombres, prend des décisions, exécute les quatre opérations arithmétiques de base.

Sa rencontre avec Ada Lovelace donne une nouvelle dimension au projet. Fille de Lord Byron et mathématicienne brillante, Ada traduit un article italien décrivant la machine. Ses notes ajoutées à la traduction détaillent un algorithme pour calculer les nombres de Bernoulli. Ce texte reste gravé dans l'histoire comme le premier programme informatique jamais écrit.

La machine analytique ne dépassera jamais le stade des plans et des schémas. L'Angleterre victorienne, malgré sa puissance industrielle, ne sait pas fabriquer un mécanisme d'une telle complexité et précision. Le coût du projet effraie les financeurs publics. Pourtant, jusqu'à sa mort en 1871, Babbage ne cesse d'affiner ses dessins, obsédé par sa vision.

Cent ans plus tard, Howard Aiken crée le Harvard Mark I en s'inspirant directement des travaux de Babbage. Les concepteurs de l'ENIAC et des premiers ordinateurs électroniques reprennent ses principes architecturaux sans toujours connaître leur origine. Babbage avait tout compris : la séparation entre matériel et logiciel, la notion de débogage, la réutilisation de code. Il avait aussi anticipé les questions de fiabilité et de performance qui hantent toujours l'informatique d'aujourd'hui.

En 1991, le Science Museum de Londres construit enfin une version fonctionnelle de sa machine à différences n°2. Elle fonctionne parfaitement, prouvant la justesse des calculs de Babbage et la cohérence de sa conception mécanique. Ce n'est pas seulement un inventeur que l'on redécouvre, mais un penseur aux multiples facettes.

Ses écrits sur l'économie industrielle, ses critiques du système universitaire britannique, son intérêt pour l'application des mathématiques à l'industrie montrent un esprit en avance sur son temps. Entre les calculatrices mécaniques du XVII^{e} siècle et les ordinateurs électroniques, l'œuvre de Charles Babbage trace un chemin visionnaire, trop souvent méconnu.

Algèbre de Boole

Au début du XIX^{e} siècle, la froideur des mathématiques de nos voisins britanniques contraste avec l'effervescence continentale. Une querelle entre disciples de Newton et de Leibniz avait sclérosé les travaux outre-Manche. Tandis que l'Europe calculait avec la notation différentielle leibnizienne, les Britanniques s'accrochaient aux fluxions newtoniennes, moins pratiques, moins fécondes.

Vers 1810, quelques esprits novateurs fondèrent la Société Analytique à Cambridge. Babbage et ses camarades brisèrent l'isolement intellectuel en important les méthodes du continent. Cela mit en question les fondements mêmes de l'algèbre : que signifiaient réellement ces nombres négatifs, ces quantités imaginaires ? Une réponse émergea dans l'algèbre symbolique. Elle ne tirait plus sa légitimité d'opérations sur des nombres, mais établissait des lois formelles applicables à n'importe quels symboles.

C'est dans cette atmosphère intellectuelle que George Boole forgea son œuvre. Né en 1815 à Lincoln dans un milieu modeste, jamais diplômé d'université, il se forma seul aux mathématiques avec une rigueur stupéfiante. D'abord maître d'école, il finit professeur au Queen's College de Cork en 1849. Une controverse entre Augustus De Morgan et William Hamilton sur la quantification du prédicat l'incita à publier en 1847 *The Mathematical Analysis of Logic*. Son intuition était d'appliquer l'algèbre au raisonnement logique.

Boole transforma la logique en système formel avec sa notation algébrique. Des symboles littéraux comme x ou y représentaient des classes d'objets, combinables par opérateurs ($+$, $\times$). Ces opérations suivaient des règles précises : commutativité ($xy = yx$), distributivité ($x(u + v) = xu + xv$), et cette propriété singulière : $x^2 = x$. Cette dernière règle marquait une rupture avec l'algèbre numérique ordinaire, où seuls 0 et 1 la respectent.

Son système traduisait les propositions logiques en équations. Ainsi, tous les X sont Y devenait $x(1 - y) = 0$. L'usage du symbole 1 pour l'univers du discours et 0 pour la classe vide constituait une autre trouvaille.

Augustus De Morgan, ami et contemporain de Boole, développait parallèlement ses propres théories logiques. Son *Formal Logic* parut le même jour que l'ouvrage de Boole en 1847. De Morgan introduisit la notion d'univers du discours variable, rompant avec l'univers fixe aristotélicien. Il formula aussi les fameuses lois qui portent son nom : la négation d'une conjonction équivaut à la disjonction des négations, et réciproquement.

La notation booléenne souffrait de limitations. Boole exigeait que l'addition ne s'applique qu'à des classes disjointes, compliquant l'expression de certaines relations. Sa définition de la soustraction imposait que la classe soustraite soit incluse dans celle de départ.

Dans les décennies suivantes, des mathématiciens affinèrent ce système. Charles Sanders Peirce apporta des contributions décisives dans les années 1880, démontrant notamment que toutes les opérations booléennes pouvaient se réduire à une seule : le NAND (non-et) ou le NOR (non-ou).

La notation moderne prit forme progressivement. Bertrand Russell introduisit le symbole $\vee$ (ou) en 1906, tandis que Arend Heyting proposa $\wedge$ (et) en 1930. L'expression « algèbre booléenne » fut employée pour la première fois par Henry Maurice Sheffer en 1913.

En 1936, Marshall Harvey Stone franchit un pas vers l'abstraction en unifiant les travaux antérieurs sous le concept d'anneau booléen. Il établit l'isomorphisme entre l'algèbre booléenne et cette structure, créant un pont théorique fondamental.

Les applications pratiques de l'algèbre de Boole explosèrent au XX$^{\text{e}}$ siècle. En 1938, Claude Shannon démontra dans sa thèse de master au

MIT que ces principes permettaient d'analyser et concevoir des circuits de commutation électrique. Cette découverte créa un lien entre logique mathématique et conception électronique, fondant l'informatique moderne.

De nos jours, l'algèbre booléenne traverse toute l'informatique. Elle structure la conception des circuits logiques, la vérification formelle des programmes et l'optimisation des requêtes dans les bases de données. Chaque microprocesseur repose sur des techniques d'optimisation issues de cette théorie.

L'histoire de l'algèbre de Boole montre comment une théorie abstraite, née de questions sur la logique, transforme un domaine technologique entier. Elle souligne l'importance des notations mathématiques en lesquelles un symbolisme adapté révèle des relations invisibles auparavant. George Boole rêvait d'un « calcul de la pensée » mécanisant le raisonnement logique. Si ce projet philosophique n'a pas abouti tel quel, ses outils mathématiques sont devenus le langage secret des machines qui façonnent notre monde.

Câble sous-marin

Au milieu du XIX^{e} siècle, franchir l'Atlantique exigeait au moins dix jours de navigation. C'est en 1840 que Samuel Morse lance une idée qui semble tenir du rêve : relier l'Europe à l'Amérique par un câble télégraphique au fond de l'océan. Cette vision presque délirante allait transformer notre rapport au temps et à l'espace.

Un homme d'affaires nommé Cyrus W. Field s'empare de cette ambition. Après avoir bâti sa fortune dans l'industrie papetière, il s'intéresse à la télégraphie durant les années 1850. Son premier projet, modeste, visait à connecter Saint-Jean de Terre-Neuve à New York, reprenant le travail de l'ingénieur Frederic Newton Gisborne. Les finances manquent. Qu'à cela ne tienne, Field voit plus grand et réoriente tout vers une liaison transatlantique.

L'Atlantic Telegraph Company naît en 1856 sous l'impulsion de Field et des ingénieurs John Watkins Brett et Charles Tilston Bright, deux Britanniques rompus aux télécommunications sous-marines. Londres et Washington participent au financement. La conception du câble déclenche des discussions passionnées entre spécialistes. Morse, soutenu par Michael Faraday, plaide pour un fil fin limitant le retard du signal. William Thomson, futur Lord Kelvin, défend un noyau de cuivre épais réduisant la résistance électrique.

La Gutta Percha Company fabrique finalement ce câble révolutionnaire. Sa structure était finalement composée de sept fils de cuivre torsadés créant un conducteur de 2,1 millimètres, enveloppé dans trois couches de gutta-percha, résine extraite d'un arbre indonésien. Une protection de chanvre goudronné et une armature métallique complètent l'ensemble, pour un diamètre final d'environ 1,6 centimètre.

Aucun navire ne pouvait emporter seul les 3 200 kilomètres de câble nécessaires. L'opération mobilise donc deux vaisseaux : le HMS *Agamemnon* britannique et l'USS *Niagara* américain. Le chargement dure trois semaines, attirant déjà l'œil des journalistes et la curiosité publique.

Le premier essai de 1857 fut un échec. Le câble se rompt près des côtes irlandaises. L'année suivante, trois tentatives se succèdent. Le 29 juillet 1858, les navires se retrouvent au milieu de l'océan, raccordent leurs portions de câble et partent en sens opposés. Le 5 août, les continents sont enfin reliés entre la baie de Bull's Arm à Terre-Neuve et Telegraph Field sur l'île irlandaise de Valentia.

Le 16 août 1858, la reine Victoria et le président James Buchanan échangent des messages. La transmission des 98 mots de la souveraine prend presque seize heures, un délai qui représentait alors une avancée prodigieuse.

La liesse explose de part et d'autre de l'océan. New York organise défilé et feu d'artifice si enthousiaste qu'il embrase partiellement l'hôtel de ville. L'église Trinity célèbre l'événement dans le sud de Manhattan. À Londres, les actions de l'Atlantic Telegraph Company doublent de valeur tandis que Charles Tilston Bright reçoit un titre de noblesse.

Le joaillier Tiffany & Co. flaire l'affaire et rachète les centaines de kilomètres de câble inutilisés rapportés par l'USS *Niagara*. L'entreprise les transforme en souvenirs, des segments de dix centimètres avec embouts en laiton et plaque descriptive vendus cinquante cents pièce. D'autres commerçants suivent avec boutons de manchette, boucles d'oreilles ou coupe-papier.

La magie dure peu et en quelques semaines, le câble cesse de fonctionner, les transmissions sont illisibles. Edward Orange Wildman Whitehouse, responsable technique du terminus irlandais, avait appliqué des tensions démesurées jusqu'à 2000 volts, persuadé qu'il fallait augmenter l'intensité avec la distance. Une analyse de 1985 par l'historien Donard de Cogan révélera aussi des problèmes de fabrication, le conducteur central décentré passait dangereusement près de l'armature métallique extérieure. Le mariage de ces défauts avec l'isolation imparfaite de la gutta-percha et les impuretés des matériaux signait l'arrêt de mort du câble.

Durant sa brève existence, cette première liaison aura transmis 732 messages. Parmi eux, l'ordre britannique d'annuler l'envoi de deux régiments vers l'Inde, la rébellion locale étant maîtrisée. Cette décision économisa

entre 50 000 et 60 000 livres sterling, environ un septième du coût du câble.

L'échec technique provoque rumeurs et accusations d'escroquerie. Cyrus W. Field, loin de se décourager et toujours soutenu par le gouvernement britannique, persévère. L'Atlantic Telegraph Company installe finalement une liaison permanente en 1866. Ce projet, aboutissant à l'époque où l'eau courante et l'électricité restaient inaccessibles pour la majorité des gens, témoigne d'une vision extraordinaire.

Cette première traversée électrique de l'Atlantique marque l'histoire des communications. Elle prouve la possibilité des échanges intercontinentaux instantanés et annonce l'architecture du réseau mondial de câbles sous-marins sur lequel repose aujourd'hui tout Internet.

Herman Hollerith

Au XIX^{e} siècle, les États-Unis croulent sous les données. Le Bureau du recensement américain peine à suivre la cadence face à une population qui explose. Herman Hollerith va créer une machine qui répondra à la problématique du traitement de ces montagnes de formulaires.

En 1879, fraîchement diplômé de l'École des mines de Columbia, il rejoint le Bureau du recensement. Il rencontre le Dr John Shaw Billings, qui dirige la division des statistiques démographiques, et qui lui souffle l'idée de mécaniser le traitement des données. En effet, l'enjeu saute aux yeux quand on regarde le recensement de 1880. Cinquante millions d'Américains à comptabiliser. Des opérateurs qui traitent péniblement vingt caractéristiques par minute avec des outils rudimentaires. Les chiffres ne sont publiés qu'en 1889, soit un an à peine avant le prochain recensement. La situation est intenable.

Hollerith quitte ensuite le Bureau pour suivre le général Francis Walker au MIT. Devenu instructeur en génie mécanique, il commence à développer ses premiers prototypes. Il s'inspire du travail du colonel Charles W. Seaton, qui avait créé une machine utilisant un système d'alimentation par rouleau comparable à celui d'un piano mécanique. Mais Hollerith prendra une autre direction.

Après cette parenthèse académique, il revient à Washington, intègre l'Office des brevets, puis s'établit comme expert indépendant. Cette expérience lui donne les clés pour protéger ses futures inventions. Il travaille d'abord sur des freins électriques pour les chemins de fer, surpassant George

Westinghouse lors d'essais en 1887. Mais les examinateurs doutent de la fiabilité de l'électricité comme source d'énergie. Qu'à cela ne tienne.

Son système de traitement des données repose sur des cartes perforées de format 6,5 × 3,25 pouces. L'opérateur, en lisant les formulaires du recensement, peut y coder jusqu'à 17 caractéristiques différentes parmi 240 positions possibles. Les codes standard, comme ceux des districts, sont préréglés et reproduits sur des piles entières de cartes.

La machine à tabuler révèle l'ingéniosité de Hollerith. Dans la mâchoire supérieure, des rangées de tiges à ressort ; dans la mâchoire inférieure, de minuscules godets de mercure. Les tiges, connectées à un courant électrique, font face aux godets reliés à 40 compteurs. L'employé insère une carte dans la presse, ferme les mâchoires, et les tiges passant par les trous plongent dans le mercure. Le courant circule jusqu'aux compteurs électromagnétiques, faisant avancer chaque aiguille concernée. Un tour complet de l'aiguille des unités active celle des centaines. La carte est ensuite déposée dans une case du trieur avant de passer à la suivante.

Les chiffres parlent d'eux-mêmes : 700 cartes perforées par jour et par opérateur, 7 000 cartes traitées quotidiennement par un utilisateur de la machine, 250 caractéristiques analysées par minute. Grâce à des relais, le système réalise des analyses plus fines, comme repérer les « hommes blancs nés à l'étranger ».

Le grand baptême du feu arrive avec le recensement de 1890. Le Bureau commande cinquante machines pour comptabiliser près de 63 millions d'Américains. Un tel succès que l'administration utilisera des versions améliorées du système pendant six décennies, jusqu'à l'arrivée de l'UNIVAC I.

L'influence de Hollerith se propage au-delà du recensement. Les services statistiques, les compagnies d'assurance, les chemins de fer, tous veulent cette technologie qui réduit drastiquement les délais de traitement des données. Dans une société industrielle en pleine mutation, elle répond à un besoin critique. D'ailleurs, certains choix techniques qu'il a fait ont eu une longévité surprenante. Le format de ses cartes perforées correspond aux dimensions des billets de banque de l'époque, simplement parce qu'il existait déjà des meubles de classement adaptés. Son système de codage positionnel, conçu pour le recensement agricole de 1901, influencera IBM et d'autres entreprises jusqu'aux années 1970.

Les brevets déposés par Hollerith sont les précurseurs de l'industrie du traitement des données. Durant soixante ans, sa machine a comblé le fossé entre traitement manuel et informatique électronique. Elle a transformé la façon dont nous gérons l'information, tout en initiant les techniques de l'industrie informatique moderne. Une invention née d'un problème très concret.

II

1930

Les prémices du calcul automatique

Un monde en transition s'éveille à la mécanisation du calcul. Le terreau des années 1930 se nourrit d'un riche héritage intellectuel : le système binaire de Leibniz (1703), la carte perforée de Jacquard (1801), et l'algèbre de Boole (1854). Ces concepts, restés en sommeil durant des décennies, s'apprêtent à bouleverser notre rapport aux mathématiques.

Le fracas du krach boursier de 1929 résonne encore quand s'ouvre cette décennie charnière. L'administration américaine s'enlise dans ses propres chiffres. Le recensement de 1930 illustre ce bourbier bureaucratique où deux années entières furent nécessaires pour dépouiller manuellement les formulaires. Une éternité.

Face à la Grande Dépression, Roosevelt déploie son New Deal. L'État s'étend, les programmes sociaux naissent. Chaque citoyen devient une fiche, chaque allocation un chiffre à consigner. Les rouages administratifs grincent sous la charge.

Dans le même temps, l'œil du cyclone totalitaire s'abat sur l'Europe. L'Allemagne nazie, avec sa froide obsession classificatoire, utilise les tabulatrices de la Dehomag, filiale d'IBM. Des existences réduites à des trous dans du carton. La bureaucratie de l'horreur prend forme.

À la fin de la décennie 1930, l'air se charge de tensions. Les services de renseignement tentent de percer le mystère d'Enigma. Cette machine allemande de chiffrement, avec ses rotors tournants, déjoue les analyses. La cryptographie accouchera des premiers calculateurs électroniques, un enjeu vital.

La radio transforme l'information. Les ondes transportent désormais voix et musique à travers continents et océans. Dans les studios, les techniciens manipulent un équipement toujours plus sophistiqué. Pourtant, si l'information voyage à la vitesse de la lumière, son traitement reste archaïque.

Les comptoirs des entreprises s'équipent. Le tac-tac caractéristique des machines à calculer mécaniques rythme la journée des employés. IBM, Burroughs et NCR prospèrent en vendant ces précieux auxiliaires du calcul commercial.

Au cœur de cette effervescence, quelques éclairs de génie illuminent l'horizon scientifique. À Cambridge, Alan Turing publie en 1936 « On Computable Numbers ». Ce texte fondateur pose la question des limites théoriques du calcul automatique. Sa machine abstraite universelle établit le concept de l'ordinateur moderne, sans que personne ne mesure sa portée. De son côté au MIT, Claude Shannon, étudiant brillant de 21 ans, relie dans son mémoire de master l'algèbre booléenne aux circuits électriques. Ce mariage fécond engendrera l'électronique numérique du XX^{e} siècle.

L'usine s'automatise progressivement. Les chaînes de montage de Ford ne représentent qu'une première étape. Les industriels recherchent des systèmes capables d'exécuter et de réguler les processus. Les cartes perforées, héritières du métier Jacquard, s'imposent comme support d'instructions.

Tous les domaines scientifiques se heurtent aux limites du calcul manuel. Les physiciens quantiques, les astronomes, les météorologues : tous jonglent avec des équations complexes qui saturent la capacité humaine de traitement. Le crayon et le papier ne suffisent plus.

L'aéronautique exige une précision diabolique. Une erreur minime dans les calculs d'aérodynamisme provoque une catastrophe. Au laboratoire balistique d'Aberdeen, des mathématiciennes, appelées « *computers* », calculent inlassablement les trajectoires pour l'artillerie. Travail minutieux dont dépendent des vies humaines.

Une génération nouvelle émerge des universités. Ces jeunes scientifiques conçoivent le calcul différemment : non plus comme une série d'opérations isolées, mais comme un processus, un flux logique. Ils pensent en algorithmes avant que le mot ne devienne courant. Les Bell Labs d'AT&T forment un creuset d'innovation sans pareil. Dans ses couloirs se croisent mathématiciens, physiciens, linguistes et ingénieurs. Cette interdisciplina-

rité féconde nourrit l'informatique balbutiante.

La course à la mécanisation du calcul s'accélère. Le relais électromagnétique remplace graduellement le levier mécanique. Plus rapide, plus fiable, il constitue une première forme de bouleversement électronique. Les brevets s'amoncellent dans les bureaux des offices nationaux.

Dans cette fièvre d'innovation, quelques standards commencent à s'imposer, comme la carte perforée d'IBM avec ses 80 colonnes et 12 lignes. Ce rectangle de carton s'affirme comme le support universel de l'information codée, l'ancêtre direct de nos fichiers numériques.

Quand les troupes hitlériennes envahissent la Pologne en septembre 1939, nul ne soupçonne que le conflit mondial servira d'accélérateur technologique à l'informatique. L'urgence militaire transformera bientôt les calculateurs mécaniques en géants électroniques.

Ces années 1930, prises en étau entre la Grande Dépression et la Seconde Guerre mondiale, portent en elles les semences de notre ère numérique. Les concepts théoriques attendent leur incarnation matérielle. La terre est prête, la tempête gronde. Paradoxalement, dans ce monde au bord du gouffre germent les racines de la révolution informatique.

L'analyseur différentiel de Vannevar Bush

Plus de deux siècles après les réflexions de Gottfried Wilhelm Leibnitz sur la mécanisation du raisonnement mathématique, un rêve longtemps resté dans le domaine du fantasme prit forme dans les laboratoires du Massachusetts Institute of Technology (MIT). Un homme, Vannevar Bush, né en 1890 à Everett dans le Massachusetts, s'attela à cette tâche colossale durant l'entre-deux-guerres.

Il a créé l'analyseur différentiel, achevé en 1931. Une machine issue d'un travail collectif avec Frank D. Gage, Harold L. Hazen, King E. Gould et Samuel H. Caldwell. L'idée n'était d'ailleurs pas totalement nouvelle. Sir William Thomson avait suggéré cinquante ans plus tôt que les intégrateurs conçus par son frère pourraient, s'ils étaient reliés entre eux, résoudre des équations différentielles. Mais les contraintes techniques de l'époque victorienne rendaient ce rêve inaccessible.

Bush réussit là où d'autres avaient échoué. Son analyseur différentiel résolvait des équations différentielles du sixième ordre ou trois équations du second ordre simultanément, une véritable prouesse technique. La machine reposait sur des amplificateurs de couple qui supportaient des charges

mécaniques considérables, un système de tiges de transmission (*bus shafts*) qui reliait les différentes unités, et des dimensions impressionnantes pour maximiser la précision des tracés. Un concept audacieux.

Le cœur de cette invention visait trois qualités rarement réunies : flexibilité extrême, robustesse mécanique et précision acceptable. Dans des conditions normales, la machine atteignait une précision d'un millième pour chaque unité individuelle. Les erreurs momentanées se compensaient naturellement lors du processus d'intégration, rappelant le comportement d'un planimètre dont les écarts finissent par s'équilibrer.

Pour utiliser cette machine, il fallait quelques heures pour la configurer après avoir réalisé les tracés nécessaires et déterminé le schéma de connexion. La résolution proprement dite prenait généralement une dizaine de minutes pour chaque ensemble de conditions aux limites. Les opérateurs devaient acquérir une certaine expérience, mais gagnaient en retour une compréhension intime des équations différentielles qu'ils manipulaient.

Vannevar Bush et ses collaborateurs durent surmonter d'importants obstacles techniques. Le jeu mécanique et le glissement des intégrateurs étaient complexes. Ils développèrent un système ingénieux appelé *lashlock* pour éliminer le jeu dans les vis sans fin, et conçurent des amplificateurs de couple à deux étages produisant des rapports de couple très élevés avec un couple d'entrée minimal.

Pour valider la fiabilité de leur invention, ils menèrent quarante essais rigoureux sur une unité d'intégration complète. Ces tests, effectués avec des charges variant de zéro à un pied-livre de couple de sortie, à différentes positions et dans les deux sens de rotation, révélèrent un écart moyen de seulement 0,032 % par rapport à la constante de calibration. L'écart maximal, observé lors d'un essai unique, ne dépassait pas 0,12 %.

Cette machine marqua l'histoire du calcul scientifique. En se positionnant à mi-chemin entre les calculateurs mécaniques rudimentaires et les futurs ordinateurs électroniques, l'analyseur différentiel démontra qu'on pouvait mécaniser des calculs mathématiques complexes. Cette avancée ouvrit des perspectives inédites pour résoudre des problèmes ardus en physique et en ingénierie.

Le théorème d'incomplétude de Gödel

Kurt Gödel publia en 1931 un texte qui changea radicalement notre vision des mathématiques. Son article « Sur les propositions formellement

indécidables des Principia Mathematica et des systèmes apparentés » mit en lumière une barrière infranchissable dans les systèmes formels mathématiques. Cette révélation, le fameux théorème d'incomplétude, allait plus tard transformer l'informatique théorique.

Les mathématiciens du début du XX^{e} siècle rêvaient d'une discipline aux bases solides et inébranlables. David Hilbert incarnait cette quête avec son programme visant à prouver la cohérence totale des mathématiques par des méthodes formelles. Bertrand Russell et Alfred North Whitehead avaient écrit leurs *Principia Mathematica* entre 1910 et 1913, tentative ambitieuse de reconstruire toutes les mathématiques à partir d'axiomes logiques élémentaires. C'est dans ce monde intellectuel que Gödel posa une question dérangeante : ces systèmes formels ont-ils des limites intrinsèques ?

Sa démarche fut brillante. Il créa une correspondance entre énoncés mathématiques et nombres, baptisée *numérotation de Gödel*. Chaque symbole, formule et preuve se voyait attribuer un code numérique unique. Cette astuce transformait les propositions mathématiques en objets arithmétiques manipulables. À l'aide de ce système, Gödel façonna une proposition mathématique particulière qui, traduite en langage courant, affirme : « Je ne peux pas être prouvée dans ce système formel ». Cette construction évoque le paradoxe antique du menteur d'Euboulide (V^{e} siècle av. J.-C.), mais sans tomber dans la contradiction.

Le résultat fut dévastateur pour le programme de Hilbert : dans tout système formel cohérent capable de décrire l'arithmétique élémentaire, des vérités mathématiques existent qui ne sont ni démontrables ni réfutables dans ce système. Son second théorème enfonça le clou en prouvant qu'un système formel cohérent ne saurait démontrer sa propre cohérence. L'édifice de certitudes que cherchaient à bâtir les mathématiciens s'effondrait.

Cinq ans plus tard, Alan Turing, inspiré par ces travaux, développa le concept de machine universelle. La technique d'encodage numérique de Gödel lui montra comment représenter des programmes sous forme de nombres, idée qui sous-tend nos ordinateurs actuels. Turing établit l'existence du problème de l'arrêt, question informatique qu'aucun algorithme ne permet de résoudre systématiquement.

La théorie de la calculabilité naquit de ces découvertes et définit les frontières du calcul automatique. Les langages de programmation, leurs compilateurs et interpréteurs portent l'empreinte de ces résultats théoriques. Quand nous tentons de certifier formellement des programmes informatiques, nous nous heurtons aux limitations identifiées par Gödel.

Les chercheurs continuent d'explorer ces territoires complexes. La théorie de la complexité s'intéresse aux ressources nécessaires pour résoudre les problèmes décidables. La logique floue propose des chemins alternatifs face

aux limites des systèmes classiques. Les outils d'aide à la démonstration mathématique intègrent ces contraintes dans leur conception.

Pour la représentation des connaissances, l'intelligence artificielle puise dans les techniques d'encodage gödéliennes. Les systèmes experts s'appuient sur ces fondements tout en reconnaissant les barrières inhérentes au raisonnement formel. Sur un plan philosophique, ces théorèmes nous interrogent sur la nature de la pensée, notamment sur l'existence d'une différence entre l'intuition mathématique humaine et les capacités des systèmes formels.

La cryptographie moderne s'inspire directement des méthodes de numérotation inventées par Gödel. La théorie des nombres, centrale dans ses preuves, constitue maintenant un pilier de la sécurité informatique. Les chercheurs qui travaillent sur les systèmes de types, la vérification de programmes ou les assistants de preuve naviguent dans un espace intellectuel dont Gödel a tracé les contours.

Presque cent ans après leur publication, les théorèmes d'incomplétude demeurent au cœur de l'informatique fondamentale. Ils nous rappellent l'existence de limites inhérentes aux systèmes formels et stimulent notre créativité pour concevoir des approches nouvelles.

La machine de Turing

Alan Mathison Turing, jeune mathématicien britannique de 24 ans tout juste sorti de King's College, publie l'article scientifique *On Computable Numbers, with an Application to the Entscheidungsproblem* en 1936. Un bouleversement intellectuelle qui allait fonder l'informatique théorique.

Il y décrit une machine abstraite d'une étonnante simplicité : un ruban sans fin divisé en cases, une tête qui lit et écrit des symboles, un ensemble d'états et de règles dictant son comportement. Cette construction minimaliste cache une puissance extraordinaire : elle résout n'importe quel problème calculable mécaniquement.

Le génie de Turing ne réside pas tant dans la complexité de son modèle que dans sa façon de repenser le calcul. Il rompt avec les abstractions mathématiques traditionnelles et s'inspire du travail des calculateurs humains, ces personnes qui exécutaient des calculs à la main selon des procédures précises. Sa machine modélise le calcul comme une séquence d'actions élémentaires déterministes.

Cette nouvelle approche règle une question lancinante posée par David Hilbert en 1928, l'Entscheidungsproblem. Turing démontre l'inexistence

d'une méthode universelle décidant si une formule mathématique donnée est démontrable. Sa conclusion rejoint celle d'Alonzo Church, obtenue indépendamment via le lambda-calcul. Mais Turing va plus loin : sa machine universelle, capable de simuler toute autre machine de Turing, esquisse l'idée d'ordinateur programmable qui marquera l'architecture des premiers calculateurs électroniques conçus par John von Neumann.

La Seconde Guerre mondiale transforme ces idées théoriques en enjeu vital. À Bletchley Park, Alan Turing met son intelligence au service du déchiffrement des communications allemandes. Il conçoit les « Bombes », machines électromécaniques automatisant la recherche des clés de chiffrement Enigma. Ces 211 machines décodent jusqu'à 3000 messages par jour. Cette contribution raccourcit le conflit d'environ deux ans selon le général Eisenhower.

La paix revenue, Turing poursuit ses recherches sur les machines pensantes, d'abord au National Physical Laboratory, ensuite à Manchester. Il invente le premier jeu d'échecs informatique et propose son fameux test : si un humain ne distingue pas les réponses d'une machine de celles d'un autre humain lors d'une conversation, alors la machine manifeste une forme d'intelligence. Cette idée reste une référence dans la recherche en intelligence artificielle.

Mais le destin frappe brutalement. En 1952, la police arrête Turing pour homosexualité, crime à l'époque au Royaume-Uni. Condamné à un traitement hormonal censé « guérir » son orientation sexuelle, il subit des effets dévastateurs sur sa santé. Privé d'habilitation sécuritaire et interdit de voyage aux États-Unis, Turing meurt en 1954, vraisemblablement par suicide. Il avait 41 ans.

Sa reconnaissance publique attendra longtemps, ses travaux à Bletchley Park restant classifiés. Les années 1970 commencent à lever le voile sur ses contributions. En 2009, Gordon Brown, Premier ministre britannique, présente des excuses officielles pour le traitement subi par Turing. La reine Elizabeth II lui accorde un pardon posthume en 2013. Quatre ans plus tard, la loi « Turing » étend ce pardon à tous les hommes condamnés pour homosexualité.

L'héritage scientifique de Turing rayonne dans notre monde numérique, car sa machine théorique est l'étalon pour comprendre les limites du calcul. Un système est dit « Turing-complet » s'il égale sa puissance de calcul, critère devenu standard pour évaluer langages de programmation et architectures informatiques.

Ses intuitions sur l'intelligence artificielle nourrissent encore la recherche actuelle. La Médaille Turing, plus haute distinction en informatique, porte son nom. En mathématiques, ses travaux sur la calculabilité et la décida-

bilité ont ouvert des champs entiers. Sa dernière recherche sur la morphogenèse, utilisant les mathématiques pour expliquer les motifs biologiques, témoigne de sa curiosité sans frontières.

Aujourd'hui, son portrait orne le billet de 50 livres depuis 2021. Son nom marque les universités, les instituts et les prix scientifiques. L'histoire de Turing raconte à la fois la naissance de l'informatique moderne et l'évolution des mentalités. Sa vie brisée et son œuvre visionnaire continuent d'inspirer mathématiciens, informaticiens et chercheurs en intelligence artificielle du XXI^{e} siècle.

Tout calcul numérique contemporain porte son empreinte. Notre monde d'algorithmes, de données et d'intelligence artificielle découle directement de sa vision. La prochaine fois que vous utiliserez un ordinateur, souvenez-vous du jeune mathématicien qui, dans les années 1930, imaginait déjà notre avenir numérique.

L'ordinateur mécanique Z1

Berlin, 1936. Konrad Zuse, un jeune homme de 26 ans, s'active dans l'appartement de ses parents. Cet ingénieur civil vient de quitter son poste chez l'avionneur Henschel où il travaillait comme calculateur. Il a une idée fixe : construire une machine capable d'automatiser les calculs mathématiques fastidieux. Sans le savoir, il s'apprête à écrire une page importante de l'histoire des calculateurs.

La naïveté technique joue en sa faveur. Il ignore tout des cartes perforées d'IBM ou des conceptions de Charles Babbage du siècle précédent. Vierge de toute influence, il repense l'architecture des machines calculantes depuis le début. Son premier choix technique est le système binaire, au moment où toutes les machines existantes fonctionnent en décimal. Sa machine séparera la mémoire de l'unité arithmétique, manipulera des nombres en virgule flottante et stockera 64 valeurs de 24 bits.

Le cœur du Z1 repose sur une invention surprenante. Plutôt que d'utiliser des engrenages classiques, Zuse fabrique des plaques métalliques qui coulissent dans un cadre. Un mouvement dans une direction symbolise un 1, l'absence de mouvement représente un 0. Ces plaques, empilées et connectées par des tiges verticales, forment des portes logiques purement mécaniques. L'unité arithmétique se divise en deux sections distinctes : l'une traite les exposants, l'autre les mantisses des nombres à virgule flottante. Quatre phases de calcul rythment l'exécution des micro-instructions, chacune associée à une direction cardinale.

Pour programmer son invention, Zuse réutilise de la pellicule de cinéma 35 mm. Il perfore ce film selon le code précis de huit trous par ligne d'instruction, dont deux pour l'opération et six pour l'adressage mémoire. Ses huit instructions couvrent les opérations arithmétiques élémentaires, les transferts mémoire, la saisie et l'affichage des résultats. Le Z1 sait aussi reconnaître des cas particuliers comme la valeur zéro (exposant -64) ou l'infini (exposant 63). Des circuits dédiés arrêtent la machine en cas d'opération invalide.

Cependant, la belle mécanique se heurte aux contraintes du réel. La coordination parfaite des milliers de pièces mobiles tourne au casse-tête. Terminé en 1938, le Z1 ne fonctionne jamais correctement. Les bombardements alliés de 1943 réduisent la machine en cendres, mais l'architecture survivra dans le Z3, version électromécanique à relais que Zuse construit en 1941.

Quarante ans plus tard, dans les années 1980, Konrad Zuse entame la reconstruction de son premier né pour le Deutsches Technikmuseum de Berlin. Cette réplique, achevée en 1989, profite des techniques de fabrication modernes : plus compacte, ses douze couches mécaniques contiennent environ 6 000 portes logiques. Malgré ces raffinements, la machine conserve les caprices de l'original et nécessite une surveillance constante.

Le Z1 est étonnamment à l'origine de nos ordinateurs actuels : architecture séparant calcul et mémoire (plus tard nommée « architecture von Neumann »), représentation normalisée en virgule flottante, système binaire. Ces caractéristiques impressionnent davantage quand on sait que Zuse n'avait ni formation en électronique ni connaissances approfondies en logique formelle.

La rareté des matériaux dans l'Allemagne d'avant-guerre força Zuse à optimiser chaque composant. Cette approche minimaliste contraste avec les projets américains contemporains comme l'ENIAC ou le Mark I de Harvard, qui bénéficiaient de budgets considérables et d'équipes nombreuses.

Si le Z1 ne constitue pas un ordinateur universel au sens strict défini par Turing – il lui manque notamment les instructions de branchement conditionnel – il marque néanmoins une rupture dans l'évolution des machines calculantes. Son architecture binaire et sa conception modulaire restent les fondations de l'informatique moderne. Il reste une anomalie, l'œuvre presque solitaire d'un visionnaire qui, par son ignorance des conventions établies, réinventa l'art du calcul mécanique au XX^{e} siècle.

Lambda-calcul

Quand Alonzo Church publie ses travaux sur le lambda-calcul dans les années 1930 à Princeton, il ne se doute probablement pas de l'impact monumental qu'aura sa création sur l'informatique future. Ce formalisme mathématique, d'apparence austère, cache une élégance qui fascine encore les chercheurs presque un siècle plus tard.

L'histoire commence véritablement avec une remarque de Frege en 1893 : toute fonction à plusieurs arguments se transforme en une suite de fonctions à un seul argument. Prenons une fonction d'addition : au lieu de calculer directement la somme de deux nombres, on construit d'abord une fonction qui attend un premier nombre, puis renvoie une autre fonction qui attend le second. Cette vision correspond étonnamment à la réalité physique des ordinateurs. Quand la machine charge un nombre en mémoire, elle est prête à l'associer à n'importe quelle autre valeur.

En 1924, Moses Schönfinkel découvre que deux fonctions élémentaires, K et S, suffisent à construire toutes les autres. Church reprend cette idée plus tard en créant une notation plus raffinée : l'application d'une fonction F à un argument A s'écrit simplement FA, les parenthèses n'apparaissant que lorsqu'elles sont nécessaires.

Les années 1930 voient fleurir des résultats majeurs. Church et Rosser prouvent en 1936 la confluence des réductions, une propriété mathématique fondamentale du lambda-calcul, qui énonce que le résultat final ne dépend pas de l'ordre des calculs intermédiaires. Cette année-là, Alan Turing établit un lien entre sa machine abstraite et le formalisme de Church, les deux approches calculent exactement les mêmes fonctions.

Le lambda-calcul réussit l'exploit de représenter les nombres naturels sans utiliser de chiffres. Un nombre n est aussi une fonction qui applique n fois une autre fonction à un argument. Cette représentation, qu'on nomme encodage de Church, rend possibles toutes les opérations arithmétiques. Les fonctions récursives, piliers de la programmation moderne, s'expriment grâce à un mécanisme appelé point fixe.

Church formule alors sa célèbre thèse : les fonctions calculables sont celles définissables dans son formalisme. Cette affirmation relie une notion intuitive à un concept mathématique précis et reste impossible à démontrer rigoureusement. Kleene la renforce en 1936 en prouvant l'équivalence entre les fonctions lambda-définissables et les fonctions récursives générales de Gödel.

L'empreinte du lambda-calcul sur l'informatique moderne s'avère profonde. John McCarthy s'en inspire directement pour créer LISP en 1958, ancêtre de toute une famille de langages fonctionnels comme Haskell ou

ML. Ces langages héritent des concepts fondamentaux du formalisme de Church : fonctions comme valeurs premières, évaluation par réduction, systèmes de types sophistiqués.

Dans les années 1970, Dana Scott construit une sémantique mathématique rigoureuse pour le lambda-calcul. Ses travaux donnent naissance à la théorie des domaines, outil puissant pour comprendre et vérifier les langages de programmation.

L'influence du lambda-calcul touche aussi la conception des compilateurs. La représentation des programmes sous forme d'arbres plutôt que de séquences linéaires d'instructions, inspirée de la structure des termes lambda, optimise l'utilisation de la mémoire – technique toujours pertinente dans les compilateurs actuels. La théorie des types dépendants, extension du système de types simple du lambda-calcul original, sert désormais de fondement aux assistants de preuve comme Coq ou Agda, outils qui vérifient formellement la validité des programmes ou des démonstrations mathématiques.

Les concepts du lambda-calcul nourrissent les architectures parallèles, les langages web comme JavaScript, et les systèmes de types modernes. Sa philosophie minimaliste – tout est fonction – combinée à son expressivité extraordinaire en fait un instrument précieux pour penser les systèmes informatiques.

Circuit logique de Shannon

La conception des circuits électriques dans les années 1930 relevait d'un processus presque artisanal. Les ingénieurs travaillant sur les premiers ordinateurs assemblaient les relais et les interrupteurs selon leur flair technique, sans méthode formalisée. Leurs créations, fruit d'une expérience personnelle, manquaient d'un cadre théorique rigoureux.

C'est au MIT qu'un jeune étudiant nommé Claude Shannon bouleversa cette approche. En 1937, il établit un pont entre deux mondes apparemment distincts : les circuits de commutation et l'algèbre de Boole. Cette discipline mathématique, élaborée au XIX^{e} siècle, n'avait jamais été appliquée à l'électricité. L'intuition de Shannon fut qu'un circuit fermé représente la valeur 1, un circuit ouvert la valeur 0. Deux composants en série fonctionnent comme l'opération logique `AND`, tandis que leur disposition en parallèle correspond à l'opération `OR`.

Sa thèse de master *A Symbolic Analysis of Relay and Switching Cir-*

cuits, publiée en 1938, formalisa cette correspondance. Le document ne se contentait pas d'exposer une théorie abstraite. Shannon y détailla des applications concrètes : un additionneur binaire et une serrure électrique à combinaison. Ces exemples démontraient la puissance d'une approche qui transformait la conception des circuits en discipline mathématique.

Les répercussions furent immédiates. Les ingénieurs disposaient désormais d'outils pour calculer le nombre minimal de composants nécessaires à une fonction donnée. Les coûts diminuèrent, la fiabilité s'améliora. Au lieu de procéder par tâtonnements, ils pouvaient vérifier leurs concepts avant la construction physique.

Shannon n'était pas seul dans cette voie. Au Japon, Akira Nakashima travaillait depuis 1935 sur des concepts similaires pour la compagnie NEC. En URSS, Viktor Shestakov explorait des idées comparables, inspiré par les travaux du physicien Paul Ehrenfest. La convergence de ces recherches montrait que le temps était mûr pour cette rupture conceptuelle.

L'arrivée des ordinateurs électroniques dans les années 1940 et 1950 donna une nouvelle dimension aux travaux de Shannon. Les relais mécaniques cédèrent la place aux tubes à vide, puis aux transistors. L'approche mathématique s'adapta parfaitement à ces nouvelles technologies. La miniaturisation constante des composants rendit indispensable l'utilisation de méthodes formelles.

L'élaboration des circuits intégrés dans les années 1960 posa la question de la conception manuelle des puces contenant des milliers de portes logiques. Les principes établis par Shannon devinrent alors le fondement des outils de conception assistée par ordinateur. Ces logiciels traduisent automatiquement des descriptions abstraites en circuits optimisés.

L'industrie des semi-conducteurs n'a cessé d'évoluer depuis, mais le cadre théorique est resté stable. Les ordinateurs actuels, malgré leur complexité vertigineuse, fonctionnent toujours selon les principes identifiés par Shannon. Sa théorie illustre comment un travail d'abstraction mathématique peut engendrer des progrès technologiques majeurs.

La représentation symbolique des systèmes proposée par Shannon a aussi inspiré le développement des langages de programmation et des méthodes de vérification formelle. Son influence s'étend à l'informatique théorique, notamment la théorie des automates et l'étude de la complexité algorithmique.

Sa thèse reçut en 1940 le prix Alfred Noble de l'American Institute of Electrical Engineers. Herman H. Goldstine la qualifia plus tard d'« une des thèses de master les plus importantes jamais écrites », qui avait transformé la conception des circuits numériques « d'un art en une science ».

Cette réussite scientifique incarne la fusion réussie entre théorie ma-

thématique et pratique d'ingénierie. Sans cette vision, l'électronique moderne aurait suivi un chemin bien différent. Le génie de Shannon fut de comprendre qu'un formalisme abstrait du XIX^{e} siècle pouvait résoudre les problèmes techniques du XX^{e} : le calcul automatique et le traitement de l'information. Les ordinateurs de nos jours, avec leurs milliards de transistors, restent fidèles aux principes qu'il a formulés. Rares sont les idées qui traversent ainsi les décennies sans perdre leur pertinence.

Le Complex Number Calculator

Une planche en contreplaqué, deux relais récupérés dans une benne, des bandes découpées dans une boîte de tabac, des piles et des ampoules. Qui aurait cru que ces matériaux de fortune assemblés par George R. Stibitz dans sa cuisine en 1937 marqueraient notre histoire de l'informatique ? Ce mathématicien des Bell Telephone Laboratories (Bell Labs) venait de remarquer un parallèle entre les positions des relais téléphoniques et la notation binaire. Son bricolage domestique donna naissance à un additionneur binaire à un chiffre.

Les soirées de Stibitz se transformèrent en séances de conception de circuits pour d'autres opérations arithmétiques. En 1938, il présenta ses travaux à Thornton Fry, chef de la section mathématique du laboratoire. Fry souleva une question pratique : ces calculateurs à relais pourraient-ils traiter des nombres complexes ? Cette tâche mobilisait alors une armée de calculateurs humains chez Bell Labs.

Stibitz s'attela au défi. Les plans furent achevés dès février 1938. Il s'associa avec Sam Williams, un ingénieur spécialiste des commutations. Leur collaboration porta ses fruits en 1939 avec l'achèvement du Complex Number Calculator.

La machine brillait par ses innovations techniques. Elle utilisait un système de codage décimal-binaire avec quatre relais par chiffre décimal. Elle traitait des nombres jusqu'à huit chiffres décimaux, avec deux chiffres supplémentaires internes pour limiter les erreurs d'arrondis. Sa structure comportait deux unités de calcul distinctes : une pour la partie réelle des nombres complexes, l'autre pour la partie imaginaire.

L'interface utilisateur constituait une révolution en soi. La machine était cachée dans un placard, uniquement accessible pour la maintenance. Les utilisateurs travaillaient sur trois stations opérateur disséminées dans le bâtiment Bell Labs de West Street à New York. Chaque station disposait

d'un clavier pour la saisie et d'un téléscripteur pour l'affichage des résultats. Cette configuration représente la première utilisation de terminaux distants dans l'histoire informatique.

Les touches de multiplication, ainsi que celle de division, activaient des sous-programmes d'une douzaine d'étapes. Ces derniers exécutaient les opérations complexes en utilisant les deux unités de calcul qui travaillaient exclusivement sur des nombres réels. Cette architecture annonçait déjà la notion moderne de sous-programme.

Le 11 septembre 1940, le Complex Number Calculator fut dévoilé au public lors d'une réunion de l'American Mathematical Society à Dartmouth College. Pour l'occasion, une console fut modifiée pour communiquer avec l'ordinateur via une ligne téléphonique longue distance. Les participants, dont le célèbre Norbert Wiener (père fondateur de la cybernétique), soumettaient des problèmes sur le clavier. Les données voyageaient jusqu'à l'équipement de relais à New York, et les résultats revenaient sur le téléscripteur en moins d'une minute. Cette démonstration inaugura le contrôle à distance d'un ordinateur, annonçant la transmission de données informatiques qui exploserait dans les années 1960.

Le Complex Number Calculator servit jusqu'en 1949, avec une fiabilité remarquable. Durant la Seconde Guerre mondiale, les groupes de conception de réseaux, ses principaux utilisateurs, le firent tourner presque sans interruption de 8h à 21h, six jours par semaine. La machine n'avait pas été conçue pour un tel régime. Construite comme modèle de démonstration avant la guerre, il lui manquait des fonctionnalités d'auto-vérification et de protection des contacts standard dans les centraux téléphoniques. La guerre empêcha la construction d'une seconde machine plus robuste. Vers la fin du conflit, il fallut l'arrêter deux jours pour remplacer les contacts usés des relais.

L'héritage du Complex Number Calculator est immense. Il prouva la faisabilité des calculs automatiques avec des composants électromécaniques fiables. Son succès engendra d'autres calculateurs à relais chez Bell Labs pendant la guerre : le Model II en 1943, capable d'opérations itératives, puis les Model III et IV en 1944-1945, plus puissants avec environ 1400 relais et sept unités de téléscripteur chacun.

Ces machines établirent des principes fondamentaux toujours présents dans l'informatique moderne : l'utilisation du binaire pour les calculs, la séparation entre unité de calcul et interface utilisateur, la notion de sous-programme, et l'accès distant aux ressources de calcul. Les calculateurs à relais de Bell Labs continuèrent à fonctionner efficacement 13 à 15 ans après la guerre, certains restant en service bien après l'arrivée des premiers ordinateurs électroniques commerciaux.

Le Complex Number Calculator, né dans une cuisine, a montré qu'on pouvait automatiser des tâches mathématiques complexes de façon fiable. Son influence s'est étendue jusqu'aux systèmes de comptabilité automatique des messages téléphoniques, où des calculateurs similaires traitaient la facturation détaillée des appels. Cette machine représente un chaînon essentiel entre les calculateurs mécaniques et nos ordinateurs.

III

1940

L'informatique dans la tourmente de la guerre

Quand la guerre éclate en 1939, personne n'imagine que ce conflit transformera radicalement notre rapport au calcul et à l'information. Hitler envahit la Pologne, le monde bascule, et dans l'ombre des états-majors, une bataille tout aussi décisive commence : celle de l'intelligence.

Le manoir de Bletchley Park, à quelques kilomètres de Londres, abrite un secret qui changera l'histoire. Alan Turing y conçoit des machines capables de décoder les messages chiffrés par Enigma, ce système réputé inviolable qu'utilisent les Allemands. La cryptanalyse, science confidentielle jusqu'alors, devient un enjeu vital. Les « bombes » électromécaniques, ces ancêtres primitifs des ordinateurs, tournent jour et nuit. Déchiffrer les communications ennemies sauve des milliers de vies et aurait raccourci la guerre de deux ans, selon certains historiens.

Pendant ce temps, l'armée américaine se heurte à ses calculs balistiques, trop complexes pour être résolus à la main, et qui ralentissent la production d'armement. À l'Université de Pennsylvanie, une équipe s'acharne sur un projet fou : l'ENIAC. Cette machine de 30 tonnes, avec ses 18 000 tubes à vide qui chauffent comme des braises, dévore 200 kilowatts d'électricité pour calculer des trajectoires d'obus en quelques secondes. Quand elle s'allume pour la première fois en 1946, la guerre est finie depuis des mois, mais l'ère du calcul électronique vient de commencer.

Les femmes jouent un rôle étrangement oublié dans cette histoire. Six mathématiciennes programment l'ENIAC en reconfigurant physiquement ses circuits. Elles inventent, sans le savoir, le métier de programmeuse. Elles sont Kay McNulty, Betty Snyder, Marlyn Wescoff, Ruth Lichterman, Betty Jean Jennings et Fran Bilas. Leurs noms méritent de figurer dans les livres d'histoire aux cotés des ingénieurs qui ont assemblé la machine.

L'après-guerre dessine un paysage neuf où les États-Unis, pratiquement intouchés par les combats, possèdent désormais 50% de la richesse mondiale. L'industrie américaine tourne à plein régime pour reconstruire l'Europe dévastée. Les machines à cartes perforées d'IBM équipent les administrations et les grandes entreprises, mais leurs limites apparaissent vite face à l'explosion des besoins de calcul.

Dans le monde scientifique, un jeune ingénieur des Bell Labs, Claude Shannon, publie « Une théorie mathématique de la communication », un article de 1948 qui passerait presque inaperçu. Il y démontre qu'on peut traiter n'importe quelle information comme une séquence de 0 et de 1. Cette idée, d'une simplicité trompeuse, allait bouleverser notre civilisation.

La guerre froide s'installe et pousse les deux superpuissances à investir dans la recherche. À Moscou, Sergueï Lebedev travaille secrètement sur le MESM, premier ordinateur soviétique. Les Américains, eux, multiplient les projets : Whirlwind au MIT, IAS à Princeton, Mark III à Harvard. John von Neumann théorise une architecture qui restera la norme pendant plus d'un demi-siècle. Les circuits mémoire s'améliorent, les tubes à vide cèdent la place aux transistors inventés en 1947.

Dans le monde innovant des télécommunications, AT&T étend son réseau de téléphone automatique. Les câbles sous-marins se multiplient. Dans les laboratoires, on expérimente la transmission numérique de données. Ces technologies finiront par converger avec l'informatique.

Les universités créent leurs premiers départements de « calcul automatique ». Un vocabulaire nouveau apparaît : programmation, algorithme, bit, octet. Des machines commerciales voient le jour. Ferranti sort son Mark I en Grande-Bretagne. Konrad Zuse, en Allemagne, crée la première entreprise d'informatique européenne. Remington Rand livre l'UNIVAC I au Bureau du recensement américain en 1951.

Les bibliothèques et centres de documentation, submergés par l'explosion de la littérature scientifique d'après-guerre, cherchent des solutions nouvelles. La classification décimale universelle montre ses limites. Des travaux théoriques sur les systèmes documentaires automatisés émergent. Vannevar Bush, dans son fameux article de 1945, imagine un bureau équipé d'un « Memex », machine capable de stocker et d'associer des documents par des liens, préfiguration troublante du Web que nous connaissons.

L'informatique n'existe pas encore en tant que discipline, on parle de cybernétique, terme proposé par Norbert Wiener en 1948 pour désigner la science du contrôle et de la communication. En France, IBM traduit « *computer* » par « ordinateur », sur suggestion du professeur Jacques Perret. Le terme s'impose vite dans la francophonie.

Qui aurait pu prédire, en 1940, que dix ans plus tard, des machines électroniques traiteraient l'information à une vitesse vertigineuse ? Ces premières réalisations, malgré leur taille monumentale et leur coût exorbitant, transformeront tous les aspects de notre vie quotidienne dans les décennies suivantes.

Le désir de créer des machines pensantes ne date pas d'hier. Pascal et Leibniz avaient construit des calculateurs mécaniques au XVII^e^ siècle. Babbage avait conçu sa machine analytique au XIX^e^. Mais il a fallu l'urgence de la guerre, l'investissement massif des États et la collaboration entre universités et industries pour que le rêve devienne réalité. L'ordinateur moderne naît de cette alchimie complexe où se mêlent nécessité militaire, recherche mathématique et ingénierie électronique.

En 1950, l'informatique sort à peine de sa préhistoire. Les machines sont énormes, fragiles, coûteuses. Seuls quelques initiés comprennent leur fonctionnement et leurs possibilités. Pourtant, le chemin vers le changement numérique est tracé. Les bases théoriques et techniques sont posées. Les premières applications civiles apparaissent.

L'aventure qui s'amorce va durer plus longtemps que la plupart des révolutions technologiques. Soixante-quinze ans plus tard, nous vivons dans les remous de cette transformation. L'ordinateur, enfant de la guerre devenu outil universel, continue de modifier notre rapport au savoir, au travail, à la communication. Son histoire commence vraiment dans les années 1940, quand des hommes et des femmes, face aux défis d'un monde en flammes, ont inventé des machines à penser.

À l'aube des années 1950, une page se tourne. Les pionniers cèdent la place aux industriels. La recherche fondamentale s'efface devant les applications pratiques. L'informatique entre dans sa phase commerciale. Des noms comme IBM, Remington Rand, Ferranti ou Bull s'imposent. Une nouvelle industrie prend son envol, portée par les besoins croissants d'une économie en pleine expansion. Nul ne soupçonne l'ampleur des bouleversements à venir.

Ordinateur d'Atanasoff et Berry

L'hiver de 1937, John Vincent Atanasoff, physicien et mathématicien à l'université d'Iowa State, rentrait chez lui par une nuit glaciale. Durant ce trajet, son cerveau s'illumina soudain d'une intuition qui transformerait le monde : quatre principes fondamentaux qui formeraient la base conceptuelle du premier ordinateur électronique. Ces principes étaient l'utilisation de l'électronique pour les calculs, l'adoption du système binaire, la séparation du calcul et de la mémoire, et le rafraîchissement périodique des données en mémoire.

Né en 1903 à Hamilton, New York, Atanasoff avait étudié l'ingénierie électrique en Floride avant de se tourner vers les mathématiques à l'Iowa State College. Son doctorat en physique théorique du MIT, achevé en 1930, l'avait confronté aux limites des calculateurs mécaniques existants. Ces machines obsolètes freinaient ses recherches sur la constante diélectrique de l'hélium, problème nécessitant une puissance de calcul alors inexistante.

La rencontre avec Clifford Berry changea la donne. Ce jeune étudiant brillant en génie électrique apporta les compétences techniques nécessaires à la mise en œuvre des idées d'Atanasoff. Leur collaboration débuta en 1939 par un prototype rudimentaire validant les concepts de base. Ce succès initial leur valut une modeste subvention de 850$ pour construire la machine complète, l'Atanasoff Berry Computer (ABC).

Cette machine pesait plus de 300 kg et comportait divers éléments novateurs. Des tambours rotatifs couverts de condensateurs stockaient les données sous forme binaire. Des tubes à vide exécutaient les opérations logiques. Des cartes perforées assuraient l'entrée et la sortie des données. Une horloge centrale cadençait l'ensemble. Contrairement aux autres calculateurs, l'ABC ne traitait pas les chiffres directement mais les convertissait en binaire, prémisse de tous les ordinateurs modernes. Sa finalité restait spécifiquement de résoudre des systèmes d'équations linéaires jusqu'à 29 équations à 29 inconnues, avec une précision remarquable de quinze chiffres décimaux.

Le destin de cette machine prit un tour tragique. Achevée en 1942, l'ABC fut délaissée quand ses créateurs quittèrent l'université pour participer à l'effort de guerre. La machine souffrait de soucis de fiabilité, notamment au niveau du système de cartes perforées. L'université négligea de finaliser la procédure de brevet entamée par Atanasoff. Ces circonstances regrettables conduisirent au démantèlement de l'ABC en 1948, sans que sa contribution à l'informatique ne soit reconnue.

L'injustice s'amplifia avec l'apparition de l'ENIAC en 1945. Cette machine conçue par Presper Eckert et John Mauchly à l'université de Penn-

sylvanie fut longtemps célébrée comme le premier ordinateur électronique. La réalité était tout autre. Mauchly avait visité Atanasoff à Iowa State en juin 1941, examiné l'ABC en détail et lu le manuscrit de 35 pages décrivant son architecture. De nombreux concepts fondamentaux de l'ABC se retrouvèrent dans l'ENIAC, sans attribution à leur véritable créateur. Le talent médiatique de Mauchly et les généreux financements militaires firent le reste : l'ENIAC vola la vedette, Atanasoff tomba dans l'oubli.

La vérité éclata au grand jour lors d'un procès retentissant. Dans les années 1960, Honeywell contesta la validité du brevet de l'ENIAC détenu par Sperry Rand. La bataille juridique, qui dura de 1971 à 1973, mit en lumière l'appropriation des idées d'Atanasoff par Mauchly. Le juge trancha sans ambiguïté : Eckert et Mauchly n'avaient pas inventé l'ordinateur numérique électronique mais s'étaient inspirés des travaux d'Atanasoff. Le brevet fut invalidé, plaçant l'invention de l'ordinateur dans le domaine public.

La reconnaissance officielle arriva enfin pour Atanasoff. Dans les décennies suivantes, il reçut de multiples distinctions, dont la prestigieuse National Medal of Technology remise par le président George H. W. Bush en 1990. Des livres, une biographie par Clark Mollenhoff en 1988, et un documentaire (*Atanasoff : Forgotten Father of the Computer*, 2012) lui furent consacrés. Une réplique exacte de l'ABC fut construite dans les années 1990 et exposée à l'Iowa State University, à côté d'un rare tambour de mémoire original.

L'histoire d'Atanasoff illustre la façon dont le génie solitaire peut transformer une discipline. En questionnant radicalement les approches existantes, il posa les bases conceptuelles de toute l'informatique du XX^{e} siècle. Certains de ses concepts, comme l'usage de l'électronique ou du binaire, s'imposèrent avec l'ENIAC puis l'EDVAC. D'autres, tel le programme enregistré, mirent plus de temps à devenir des standards industriels. L'ABC joua un rôle de catalyseur dans l'évolution du calcul électronique, justifiant pleinement pour Atanasoff le titre de père de l'informatique moderne.

Réseaux de neurones artificiels

Au milieu du XX^{e} siècle, la recherche sur l'intelligence artificielle connut une avancée remarquable avec les travaux de Warren McCulloch et Walter Pitts. Ces deux chercheurs, que tout semblait opposer, unirent leurs talents pour concevoir un modèle mathématique du neurone. Leur proposition fit évoluer notre compréhension du cerveau et fonda la théorie des futures technologies d'apprentissage automatique.

En 1943, McCulloch, neuropsychiatre chevronné, et Pitts, jeune prodige des mathématiques, publièrent un article fondateur. Ce texte novateur *A Logical Calculus of the Ideas Immanent in Nervous Activity* établit une correspondance entre le fonctionnement des neurones biologiques et les opérations logiques. Les auteurs s'appuyèrent sur le caractère « tout ou rien » de l'activité neuronale pour modéliser les cellules nerveuses comme des dispositifs binaires, soit actifs, soit inactifs. Cette simplification leur donna la possibilité de construire une équivalence avec la logique booléenne et ses valeurs « vrai/faux ».

Les parcours de ces deux chercheurs méritent qu'on s'y attarde. McCulloch, formé à la psychiatrie, nourrissait depuis longtemps des questionnements philosophiques sur la nature logique de la pensée et son substrat physiologique. Durant les années 1930, il s'investit dans la recherche en neurophysiologie et collabora notamment avec J. G. Dusser de Barenne à Yale, où ils étudièrent la localisation des fonctions cérébrales. Pitts, quant à lui, se distingua par son génie précoce pour les mathématiques. Sans formation académique formelle, il se passionna pour la logique mathématique dès son plus jeune âge, dévorant les *Principia Mathematica* de Bertrand Russell. Au début des années 1940, il rejoignit l'équipe de Nicolas Rashevsky à l'université de Chicago, groupe pionnier dans l'application des mathématiques à la biologie.

La rencontre entre McCulloch et Pitts survint en 1942, orchestrée par Jerome Lettvin. Malgré leur différence d'âge significative, ils découvrirent très vite leurs intérêts communs. Pitts s'enthousiasma pour le concept de « machine logique », capable d'exécuter des raisonnements, idée qu'il rattachait aux travaux de Leibniz et Turing. McCulloch, de son côté, tentait depuis les années 1920 de traduire l'activité neuronale en calcul logique. Ensemble, ils se demandèrent si le système nerveux ne fonctionnait pas comme une telle machine logique.

Leur article de 1943 marie avec audace des éléments de neurophysiologie, de philosophie et de mathématiques. Ils y présentent un modèle simplifié du neurone comme une entité binaire qui s'active ou non selon la somme de ses entrées excitatrices et inhibitrices, émettant un signal lorsque cette somme dépasse un certain seuil. En reliant ces neurones idéalisés, ils démontrent la construction de réseaux utilisant les fonctions logiques fondamentales (`ET`, `OU`, `NON`). Leur démarche suit une approche axiomatique : à partir d'hypothèses simplificatrices sur l'activité neuronale, ils bâtissent un calcul logique des relations entre neurones.

Les auteurs soulignent eux-mêmes les limites théoriques de leur modèle. Ils ne cherchent pas à représenter fidèlement les neurones biologiques, mais à établir un cadre formel pour raisonner sur des réseaux hypothé-

tiques aux propriétés connues. Ils admettent que l'activité des neurones réels s'avère plus continue que discrète et que des phénomènes comme l'apprentissage modifient durablement la structure des réseaux. Leur ambition réside ailleurs : fournir un outil mathématique pour manipuler rigoureusement des réseaux connus et créer aisément des réseaux aux propriétés désirées.

Cette approche théorique de la biologie, dont Nicolas Rashevsky fut un ardent promoteur, marque l'article de McCulloch et Pitts. Physicien de formation, Rashevsky fonda dans les années 1930 un programme de « biophysique mathématique » à Chicago. Selon lui, la biologie mathématique devait procéder par abstractions et idéalisations, similairement à la physique. Comme le physicien étudie des concepts idéaux tels que le point matériel ou le fluide parfait, le biologiste théoricien doit partir de modèles simplifiés pour saisir progressivement la complexité du vivant. Les mathématiques, d'après Rashevsky, offrent un cadre précieux pour formaliser les phénomènes biologiques et en déduire des propriétés par un raisonnement rigoureux.

C'est dans cet esprit que McCulloch et Pitts façonnent leur modèle de réseaux de neurones logiques. Leur projet ne vise pas tant à décrire le système nerveux réel qu'à montrer comment un réseau d'éléments simples, connectés adéquatement, développe des fonctions logiques et, potentiellement, des raisonnements complexes. Leur travail se rattache aussi aux réflexions contemporaines sur les fondements logiques du calcul, illustrées par la machine de Turing. En 1936, Turing avait montré comment définir le processus de calcul par une machine abstraite manipulant des symboles selon des règles précises. McCulloch et Pitts s'inspirent de cette idée pour concevoir le cerveau comme une machine logique, dont les composants élémentaires seraient les neurones tout-ou-rien.

L'impact de cet article dépassa toutes les attentes. Sur le plan théorique, il posa les fondations des neurosciences computationnelles et des sciences cognitives : l'idée que des fonctions cognitives complexes naissent de réseaux d'unités simples inspira de nombreux travaux en intelligence artificielle et en neurosciences. L'article présente aussi une dimension philosophique en proposant une vision mécaniste de l'esprit, où la pensée se réduirait à des opérations logiques exécutées par des réseaux de neurones. Cette conception marqua durablement les débats sur la nature de l'intelligence et ses rapports au substrat biologique.

Le modèle proposé comporte certes d'importantes limitations, que les auteurs reconnaissent. Il idéalise le neurone comme entité binaire à seuil fixe, alors que les neurones biologiques possèdent des dynamiques continues et adaptatives. Il suppose une structure de réseau figée, quand les

connexions neuronales se reconfigurent constamment par plasticité. Néanmoins, par sa clarté formelle, l'article ouvrit un vaste champ de recherches. Il montra comment une approche mathématique éclaire le fonctionnement du système nerveux et, plus largement, les bases biologiques de l'intelligence.

Ce travail témoigne de la richesse des échanges interdisciplinaires. Il naît au carrefour de questionnements philosophiques sur l'esprit, d'avancées expérimentales en neurophysiologie et de développements formels en logique mathématique. C'est par ce croisement que McCulloch et Pitts aboutissent à une vision nouvelle, où le cerveau apparaît comme système de traitement logique de l'information. Si leur modèle demeure rudimentaire, il esquisse les contours conceptuels des futures recherches sur les réseaux de neurones formels et leurs applications.

Colossus

Au cœur de la guerre secrète que livraient les cryptanalystes britanniques, une machine électronique permit le déchiffrement des messages ennemis et changea le cours de l'histoire : Colossus.

Entre 1943 et 1945, dans le silence des bâtiments de Bletchley Park, dans le Buckinghamshire en Angleterre, Tommy Flowers dirigea la construction d'une série de calculateurs destinés à percer les codes de la machine allemande Lorenz SZ40. Les cryptanalystes britanniques l'avaient surnommée « Tunny » – le thon. Les Allemands, eux, l'appelaient *Schlüsselzusatz* ou SZ. La mission de Colossus était de trouver les réglages initiaux des rotors de Lorenz pour décoder les messages captés.

Techniquement, Colossus incarnait une prouesse. La machine lisait 5 000 caractères par seconde sur des bandes perforées, effectuait des calculs booléens à la volée et imprimait immédiatement les résultats. Son cœur battait au rythme de 1 500 tubes à vide, à une fréquence de 5 000 impulsions par seconde. Elle disposait de circuits arithmétiques binaires, d'une forme primitive de mémoire et de capacités de branchement conditionnel.

Cette réalisation n'aurait jamais vu le jour sans le mariage des mathématiques et de l'ingénierie. Max Newman et Alan Turing, par leur vision théorique, ont aidé à la conception de cette machine, même si Turing n'a pas directement participé au projet. Ses travaux sur la « machine universelle » ont néanmoins infusé l'esprit du projet.

Le premier modèle fonctionna début 1944. Un succès tel qu'une dizaine

d'exemplaires furent construits jusqu'à la fin de la guerre. Chose remarquable pour l'époque, Colossus fonctionnait sans tomber en panne. Une fiabilité rare, héritée du savoir-faire téléphonique de l'équipe de Dollis Hill. Cette machine donna des résultats exploitables presque immédiatement, quand les premiers ordinateurs électroniques exigeaient de longs mois de calibration avant de fonctionner correctement.

L'impact de Colossus sur l'issue du conflit reste difficile à mesurer avec précision. Des témoignages suggèrent un rôle capital dans l'anticipation des mouvements allemands autour du débarquement en Normandie. La machine aurait a priori déchiffré des messages signés par Hitler.

Pendant trois décennies, Colossus resta dans l'ombre du secret militaire. L'ENIAC américain recevait les honneurs quand Colossus dormait dans l'oubli. Il fallut attendre le milieu des années 1970 et le travail de Brian Randell pour que son existence soit révélée.

La qualification exacte de Colossus divise toujours. S'agissait-il d'un véritable ordinateur ? Randell le définit comme un « calculateur électronique programmable à usage spécifique ». Tommy Flowers, lui, préférait parler de « processeur électronique ». Car Colossus ne calculait pas comme les ordinateurs classiques. Il exécutait un programme fixe, sans possibilité réelle de modification. Sa logique, basée sur des flux de bits plutôt que sur des mots, le distinguait des architectures traditionnelles. Cette nature hybride représente sa véritable force, et témoigne de la richesse des approches dans cette période fondatrice. L'histoire des machines n'a pas suivi une ligne droite vers l'ordinateur moderne, mais a exploré différentes voies pour répondre à des problèmes concrets.

Le succès de Colossus tient aussi à sa pertinence pratique. Il répondait parfaitement aux besoins des cryptanalystes dans un contexte d'urgence. Son histoire nous rappelle que le développement technologique se nourrit d'innovations théoriques, d'adéquation aux usages, de fiabilité et d'efficacité opérationnelle.

Harvard Mark I

Dans les années 1930, le paysage scientifique manquait d'outils capables de résoudre des problèmes mathématiques d'une grande complexité. Howard Aiken, physicien en herbe à Harvard, s'attela à combler cette lacune. Durant l'année 1937, il esquissa les plans d'un calculateur qui traiterait des lignes entières au lieu des colonnes qu'affectionnaient les machines de

l'époque. Il y voyait une machine capable de jongler avec les nombres positifs et négatifs, de manier les fonctions transcendantes et d'enchaîner automatiquement les calculs.

Après un parcours semé d'embûches auprès de différents fabricants, Aiken finit par convaincre IBM. L'entreprise baptisa le projet *Automatic Sequence Controlled Calculator* (ASCC). Mars 1939 marqua la signature d'un accord entre IBM qui construirait la machine pour en faire cadeau à Harvard qui fournirait locaux et personnel. La construction démarra deux mois plus tard dans les laboratoires d'IBM à Endicott, New York. Aiken collabora étroitement avec l'ingénieur Clair D. Lake pour mener à bien ce projet ambitieux.

Le prototype vit le jour en janvier 1943 et passa avec succès ses tests initiaux. Transporté démonté vers Harvard, il prit place au sous-sol du *Physics Research Laboratory*. Dès le printemps 1944, alors qu'il devenait opérationnel, son utilisation fut détournée vers l'effort de guerre. Robert Campbell prit les commandes d'une équipe de la marine américaine où figuraient Grace Murray Hopper et Richard Bloch. Aiken, officier de réserve, rejoignit le service actif en 1941 pour travailler sur les mines navales avant de prendre le commandement du Mark I.

Le 7 août 1944, une cérémonie officielle réunit James Bryant Conant, président de Harvard, et Thomas J. Watson Sr., président d'IBM. L'événement tourna au vinaigre suite à un communiqué de presse rédigé sous l'influence d'Aiken qui s'attribuait tout le mérite de l'invention, effaçant presque totalement la contribution d'IBM. Watson, furieux, menaça de boycotter la cérémonie. Aiken dut battre en retraite et reconnaître le rôle joué par Lake, Hamilton et Durfee dans la création de cette machine.

Le monstre mécanique impressionnait par ses dimensions : 15,5 mètres de long, 2,4 mètres de haut pour une masse de 4,5 tonnes. Son ventre abritait des relais électromagnétiques, des compteurs décimaux rotatifs et des rubans perforés pour l'entrée des instructions. Les données entraient par cartes perforées et sortaient via des machines à écrire électriques modifiées. Lent comparé aux futures merveilles électroniques, le Mark I compensait par une fiabilité exemplaire. Il tournait sans interruption, jour et nuit, produisant un flot continu de résultats.

Au-delà de l'exploit technique, cette machine incarna la faisabilité des calculateurs automatiques à grande échelle. Ces machines pouvaient désormais suivre une séquence d'opérations du début à la fin sans intervention humaine. L'écho médiatique mondial généré par ses prouesses balaya les derniers doutes quant à l'avenir de telles machines dans la sphère scientifique. Le Harvard Mark I symbolise la transition entre l'ère du calcul manuel et l'aube de l'informatique moderne.

Aiken ne s'arrêta pas à cette première réussite. Il développa une série de quatre calculateurs toujours plus sophistiqués. Visionnaire, il comprit que ces nouvelles machines appelaient une nouvelle génération de mathématiciens capables de les programmer. Il persuada Harvard de créer un cursus spécifique, d'abord en maîtrise puis en doctorat, dans cette discipline naissante qu'était l'informatique. Premier professeur titulaire dans ce domaine encore vierge, il initia ce qui semble avoir été le premier programme académique du genre au monde. Son enseignement façonna des pionniers comme Gerrit Blaauw, Frederick Brooks Jr., Kenneth Iverson et Anthony Oettinger.

ENIAC

En 1943, l'armée américaine lança un projet ambitieux pour accélérer ses calculs balistiques. La guerre battait son plein et les tables de tir d'artillerie, vitales pour la précision des bombardements, exigeaient des calculs interminables. La Moore School of Electrical Engineering de l'université de Pennsylvanie fut chargée de créer une machine radicalement nouvelle, sous la direction de John Mauchly et J. Presper Eckert.

Baptisée ENIAC (Electronic Numerical Integrator and Computer), cette machine marqua une rupture avec les calculateurs électromécaniques. Son développement mobilisa une énergie colossale : environ 200 000 heures de travail furent nécessaires pour aboutir, en novembre 1945, à un monstre technologique. Trop tard pour servir pendant la guerre, l'ENIAC s'imposa néanmoins comme une prouesse d'ingénierie sans précédent.

Les chiffres donnent le vertige : 30 tonnes d'équipement réparties sur une surface de 150 m^2, 40 grands panneaux abritant 18 000 tubes à vide, 1 500 relais, 10 000 condensateurs et 6 000 commutateurs manuels. Contrairement à l'ABC d'Atanasoff qui utilisait le binaire, l'ENIAC manipulait directement des nombres décimaux de 10 chiffres. Sa mémoire se composait d'accumulateurs stockant chacun un nombre, complétés par des tables de fonctions pour les constantes.

L'ENIAC se programmait d'une façon rudimentaire et fastidieuse. Il fallait régler des milliers de commutateurs et brancher des centaines de câbles, opération qui pouvait prendre quelques jours. Une fois configuré, cependant, ses performances stupéfiaient : 5 000 additions ou 385 multiplications par seconde, pulvérisant les records des machines existantes.

Le 14 février 1946, une conférence de presse soigneusement orchestrée

dévoila l'ENIAC au monde. L'effet fut saisissant. Les journaux s'enflammèrent, qualifiant la machine de « cerveau géant », de « cerveau magique » ou de « calculateur météore ». Pour la première fois, les médias attribuaient des capacités quasi humaines à une machine. Cette anthropomorphisation marqua l'imaginaire collectif, malgré les tentatives de Mauchly pour tempérer l'enthousiasme en rappelant que l'ENIAC exécutait simplement des calculs sans vraiment « penser ». Le mythe était né, la machine capable de résoudre tous les problèmes venait d'apparaître.

Ce que le public ignorait, c'est que l'ENIAC s'inspirait largement des travaux antérieurs de John Atanasoff et Clifford Berry. Comme révélé plus tard lors d'un procès retentissant, Mauchly avait visité Atanasoff en 1941 et étudié en détail l'ABC, y reprenant plusieurs concepts fondamentaux sans jamais en créditer les inventeurs. Cette controverse sur la paternité de l'ordinateur électronique resta enfouie pendant des décennies, alors que l'ENIAC récoltait tous les honneurs.

L'ENIAC fut aussi le premier ordinateur programmé par des femmes, une autre réalité méconnue. Des mathématiciennes et physiciennes, auparavant employées comme « computers » humains pour effectuer des calculs à la main, furent sélectionnées pour leur excellence. Six d'entre elles devinrent les premières programmeuses de l'histoire. Elles développèrent les programmes initiaux et réalisèrent d'innombrables calculs, souvent dans l'ombre, leur contribution restant longtemps ignorée.

La Guerre froide offrit à l'ENIAC une seconde vie. Installé à l'Aberdeen Proving Ground dans le Maryland, il servit pour les tables balistiques initialement prévues, ainsi que pour des calculs liés aux premières bombes thermonucléaires. Sa capacité à résoudre rapidement des équations complexes accéléra les recherches en physique nucléaire et dans d'autres domaines scientifiques.

Bien que retiré du service en 1955, l'ENIAC laissa une empreinte durable sur le développement de l'informatique. Il stimula la recherche aux États-Unis, tant au niveau matériel que théorique. Les leçons tirées de sa conception inspirèrent directement des machines plus avancées comme l'EDVAC et l'UNIVAC. Parallèlement, des figures comme John von Neumann, Alan Turing et Claude Shannon élaborèrent les bases théoriques qui allaient structurer la discipline.

L'influence de l'ENIAC dépassa largement les frontières américaines. Sa médiatisation fit prendre conscience au monde entier du potentiel des ordinateurs électroniques. Des projets similaires émergèrent au Royaume-Uni, en URSS, en Allemagne et au Japon. En une décennie, l'informatique passa d'expérimentale à industrielle. Le concept d'« ordinateur » s'installa dans la culture populaire, mêlant fascination et inquiétude face à ces « cerveaux

électroniques » aux capacités mystérieuses.

L'histoire de l'ENIAC connut un rebondissement dans les années 1970. Un procès retentissant, opposant Honeywell à Sperry Rand (détenteur du brevet de l'ENIAC), révéla l'appropriation des idées d'Atanasoff par Mauchly. Le juge statua en 1973 que l'ENIAC n'était pas le premier ordinateur électronique et invalida son brevet, plaçant ainsi l'invention dans le domaine public. Cette décision juridique eut des conséquences capitales sur l'évolution ultérieure de l'informatique.

L'annulation du brevet de l'ENIAC libéra un potentiel d'innovation considérable. Privés de l'exclusivité sur les concepts fondamentaux, les laboratoires et entreprises durent miser sur des améliorations techniques spécifiques plutôt que sur la protection juridique des idées de base. Cette situation favorisa une éclosion remarquable de machines concurrentes : IBM, DEC, Control Data Corporation et d'autres s'engouffrèrent dans la brèche, chacun proposant sa vision de l'architecture informatique. Sans cette décision, le paysage numérique aurait pu ressembler davantage à une industrie dominée par quelques détenteurs de brevets imposant leurs conditions au marché. L'informatique prit au contraire un chemin unique où les principes fondateurs restèrent accessibles à tous tandis que les innovations spécifiques pouvaient être protégées. Cette dynamique singulière explique en partie la vitesse stupéfiante à laquelle les ordinateurs évoluèrent des monstres de plusieurs tonnes aux machines personnelles, puis aux appareils mobiles que nous connaissons maintenant.

Ce basculement dans la perception des machines calculantes fut peut-être la contribution la plus durable de l'ENIAC, malgré sa paternité controversée. En quelques temps, l'image publique des ordinateurs se transforma radicalement : de simples outils de calcul, ils devinrent des entités potentiellement capables de surpasser l'intelligence humaine. Cette vision, bien qu'exagérée, orienta les recherches en informatique pour les décennies suivantes, notamment avec l'émergence de l'intelligence artificielle dans les années 1950.

Mémoire à ligne de retard

Les premiers calculateurs électroniques se heurtaient à la limite du stockage de leurs données et leurs instructions. L'ENIAC ne possédait qu'une vingtaine de registres à tubes à vide, une solution ruineuse pour toute mémoire substantielle.

C'est alors que J. Presper Eckert imagine un concept novateur de transformation des signaux électriques en ondes sonores circulant dans un tube de mercure. Ces vibrations, captées à l'autre bout, redevenaient électriques avant d'être réinjectées en entrée. La boucle était bouclée. Cette astuce multipliait par cent la capacité des mémoires à prix égal, chaque ligne pouvant contenir jusqu'à 500 bits.

Maurice Wilkes et son équipe de Cambridge donnèrent vie à cette idée dans l'EDSAC en 1949. Leur machine exploitait 32 tubes de mercure stockant 512 mots de 36 bits. Un mot y circulait toutes les 48 microsecondes sur chacune des 128 lignes, tandis qu'une simple addition nécessitait 864 microsecondes.

Cette technologie répondait aux besoins qu'avait identifiés John von Neumann dans sa description de l'EDVAC en 1945. Il y soulignait la nécessité d'une mémoire « interne » distincte des dispositifs externes comme les cartes perforées. Cette mémoire devait conserver résultats intermédiaires, instructions, tables de fonctions, conditions initiales et résultats d'itérations successives.

Malgré son ingéniosité, cette technologie souffrait de fiabilité incertaine et de coûts élevés. Les constructeurs lui préférèrent la mémoire à tambour magnétique. Les IBM 650 s'équipèrent ainsi de tambours capables de stocker jusqu'à 2 000 mots de dix chiffres, avec un temps d'accès moyen de 2,4 millisecondes, limité par la rotation du tambour à 12 500 tours par minute.

L'EDVAC illustra cette transition : en 1954, une mémoire à tambour de 4608 mots vint compléter ses lignes de retard. Les tambours restèrent présents jusque dans les années 1960, principalement comme mémoire secondaire.

La mémoire à ligne de retard a prouvé qu'on pouvait bâtir des mémoires de grande capacité séparées des unités de calcul, concrétisant ainsi l'architecture de von Neumann. Elle a été le concept fondateur de notre hiérarchie mémoire actuelle.

Les années 1960 virent l'avènement des tores magnétiques. Sans pièces mobiles, ils offraient un accès véritablement aléatoire avec des temps de cycle d'environ 6 microsecondes—mille fois plus rapides que les tambours. Les capacités explosèrent : le CDC 6600 de 1965, cadencé à 100 nanosecondes, disposait d'environ 128 000 mots accessibles en 1 microseconde.

La ligne de retard symbolise les équilibres délicats recherchés par les pionniers entre coût et performance, fiabilité et capacité, complexité et faisabilité. Elle témoigne de l'inventivité des premiers architectes d'ordinateurs, confrontés à des défis avec les moyens techniques du XX^{e} siècle.

Le principe de circulation et régénération des données qu'elle mit en œuvre se retrouve, transformé, dans les mémoires DRAM contemporaines.

Sa trajectoire illustre un motif récurrent dans l'histoire informatique : une innovation technique surmonte un obstacle, avant d'être supplantée par des solutions plus performantes.

La mémoire à ligne de retard représente l'instant où les ordinateurs acquirent une véritable mémoire de travail, distincte des unités de calcul et du stockage permanent. Cette séparation, devenue axiomatique dans l'architecture des ordinateurs modernes, trouva sa première expression concrète avec cette technologie.

Méthode de Monte Carlo

L'histoire de la méthode Monte Carlo débute dans le cadre singulier du projet Manhattan durant la Seconde Guerre mondiale. Les chercheurs de Los Alamos butaient sur des calculs de physique nucléaire d'une complexité inédite. Le manque d'outils adaptés stimula leur imagination mathématique.

Dès les années 1930, Enrico Fermi, physicien italien naturalisé américain en 1945, expérimenta quelques techniques d'échantillonnage statistique pour résoudre ses équations de neutronique. Ces balbutiements restèrent confidentiels, faute d'ordinateurs capables de traiter de grands volumes de calculs.

La véritable naissance de Monte Carlo survint en 1946. Stanislaw Ulam, mathématicien polonais, se remettait d'une maladie en jouant au solitaire. Une question le tarauda : comment calculer exactement les chances de gagner une partie ? L'idée lui vint de simuler un nombre colossal de parties pour obtenir une estimation statistique fiable. Cette intuition fulgurante rencontra l'enthousiasme de John von Neumann, qui programma les premières simulations sur l'ENIAC en 1947. Pour ces calculs, il créa une technique mathématique puissante de génération de nombres pseudo-aléatoires, baptisée *middle-square digits*, pour estimer des valeurs numériques et résoudre des problèmes complexes. Le nom « Monte Carlo » naquit d'une plaisanterie de Nicholas Metropolis, en clin d'œil à l'oncle d'Ulam, joueur compulsif au casino monégasque.

Entre 1946 et 1947, pendant une panne prolongée de l'ENIAC, Enrico Fermi conçut le FERMIAC, appareil mécanique étonnant capable de simuler la diffusion des neutrons selon le principe Monte Carlo. Cette invention témoigne de l'engouement immédiat des physiciens pour cette approche radicalement nouvelle.

Un bond conceptuel intervint en 1953 avec l'algorithme imaginé par Metropolis et son équipe, dont les physiciens Marshall et Augusta Mici Rosenbluth, ainsi qu'Edward Teller. Leur méthode exploitait les chaînes de Markov pour générer des échantillons suivant une distribution donnée et explorer des espaces complexes. En 1970, Wilfred Keith Hastings enrichit cette approche, aboutissant à l'algorithme Metropolis-Hastings, pilier actuel de la statistique computationnelle moderne.

La décennie 1980 vit fleurir des variantes inspirées. L'algorithme du recuit simulé, conçu par Kirkpatrick, Gelatt et Vecchi en 1983, adapta Monte Carlo à l'optimisation combinatoire en s'inspirant du refroidissement contrôlé des métaux. Un an plus tard, les frères Geman appliquèrent l'échantillonnage de Gibbs au traitement d'images, ouvrant de nouveaux horizons en vision artificielle.

Le début des années 1990 marqua la consécration des méthodes Monte Carlo par chaînes de Markov grâce aux travaux fondamentaux de Gelfand et Smith. Ces avancées bouleversèrent l'inférence bayésienne et transformèrent la pratique statistique quotidienne. Cette décennie foisonnante vit aussi émerger l'algorithme du saut réversible de Peter Green en 1995, qui autorisait l'exploration d'espaces de dimensions variables, ainsi que l'échantillonnage parfait de Propp et Wilson en 1996, garantissant des échantillons exactement distribués.

L'évolution de Monte Carlo illustre le dialogue fécond entre théorie et technologie. Des premiers calculateurs jusqu'aux supercalculateurs actuels, chaque bond de puissance a décuplé le potentiel de ces méthodes. Leurs champs d'application se sont diversifiés : physique statistique, chimie moléculaire, finance quantitative, apprentissage automatique, et infographie 3D avec le transport de lumière de Metropolis d'Eric Veach en 1997.

Ironie historique, cette technique initialement mise au point pour les armes nucléaires sert désormais la médecine moderne. Monte Carlo sert en radiothérapie pour simuler avec une précision inégalée les interactions entre rayonnements et tissus biologiques. Son impact scientifique colossal lui a valu d'être désigné comme l'un des dix algorithmes les plus décisifs du XX^{e} siècle par la Society for Industrial and Applied Mathematics.

Tube Williams

L'après-guerre fut témoin d'une quête frénétique pour résoudre l'épineux problème du stockage de données dans les premiers calculateurs électroniques. Tandis que Presper Eckert et John Mauchly s'acharnaient sur les

lignes à retard au mercure à Philadelphie et que Jay Forrester entamait ses recherches sur les tores magnétiques au MIT, Frederic Williams traçait un chemin différent au Telecommunications Research Establishment de Great Malvern.

La visite de Williams au projet ENIAC en 1946 lui donna deux clés : l'idée du programme enregistré et le besoin vital d'un stockage électronique rapide. Son passé dans les circuits radar durant le conflit mondial l'amena, avec son doctorant Tom Kilburn, à explorer les tubes cathodiques comme dispositifs mémoire. Les ordinateurs réclamaient environ 32 000 bits binaires organisés en 1 024 mots de 32 bits. Les bascules électroniques auraient exigé plus de 64 000 tubes à vide, rendant toute fiabilité chimérique.

La trouvaille de Williams et Kilburn reposait sur un phénomène physique subtil : le bombardement d'électrons sur l'écran créait un puits de potentiel positif. Un second bombardement à proximité générait des électrons secondaires qui comblaient partiellement le premier puits. La détection de ce comblement lors d'un balayage ultérieur déterminait si un 1 ou un 0 avait été inscrit. Une plaque métallique contre l'écran captait ces variations infimes de charge.

Cinq variantes de ce principe virent le jour. Le système `dot-dash` codait un 0 par un point et un 1 par un tiret. Le `dash-dot` inversait cette logique. Les systèmes `defocus-focus` et `focus-defocus` jouaient sur la netteté du faisceau. Le système `anticipation` exploitait un signal prédictif lors de la destruction d'une charge.

Début 1947, désormais à l'Université de Manchester, Williams et Kilburn stockèrent leur premier bit stable. À l'automne, ils maintinrent 1 024 digits pendant plusieurs heures. Leur éclair de génie fut d'introduire un mécanisme de régénération. La charge ne subsistant que quelques dixièmes de seconde, le contenu était sans cesse lu et réécrit – technique qui survit dans nos mémoires DRAM actuelles.

Juin 1948 vit naître le Manchester Baby, surnom du Small-Scale Experimental Machine. Ce prototype utilisait trois tubes : un pour le stockage principal (32 mots), un pour l'accumulateur (registre de calcul), et un dernier pour le contrôle (compteur ordinal et instruction courante). Son architecture autorisait un accès aléatoire à la mémoire, quand les lignes à retard imposaient un accès séquentiel. Son jeu d'instructions se résumait à sept opérations : `JMP` (saut), *JRP* (saut relatif), `LDN` (chargement du négatif), `STO` (stockage), `SUB` (soustraction), `CMP` (comparaison) et `STP` (arrêt).

Deux nouvelles moutures surgirent en 1949. La première offrait 128 mots de 40 bits avec un multiplicateur électronique. Elle servit à des calculs sophistiqués comme la vérification de l'hypothèse de Riemann et le test de primalité des nombres de Mersenne. La seconde mariait la mémoire Williams à

un tambour magnétique stockant 40 960 bits supplémentaires, créant ainsi une hiérarchie mémoire à deux niveaux qui annonçait l'organisation des ordinateurs futurs.

IBM s'empara de cette technologie en 1951, acquit une licence et lança la fabrication de tubes Williams pour son 701, premier calculateur commercial de la firme. Cette industrialisation exigea des prouesses, l'entreprise créa des environnements ultra-propres avant l'heure. Le personnel portait des blouses en orlon pour éviter les particules textiles. Des tapis collants piégeaient la poussière des chaussures. Ces pratiques devancèrent celles adoptées plus tard pour les semi-conducteurs.

La version finale de l'ordinateur de Manchester, livrée en 1951, alignait huit tubes Williams stockant chacun 1 024 bits (mémoire totale de 10 240 bits), complétés par un tambour magnétique de 150 000 bits. Elle traitait les instructions en 1,2 milliseconde et effectuait les multiplications en 2,16 millisecondes. Ses applications touchaient la biologie, l'optique, la cristallographie, l'ingénierie chimique et même les échecs.

Les contraintes techniques des tubes Williams se révélèrent nombreuses. La qualité de l'écran phosphorescent s'avérait capitale : la moindre imperfection – particule de carbone ou trou microscopique – compromettait le stockage. Le faisceau exigeait une uniformité parfaite sur toute la surface. La vitesse de balayage nécessitait un contrôle d'une précision absolue.

Malgré ces écueils, la technologie équipa quelques machines des années 1950 : le Ferranti Mark I (héritier commercial de la machine de Manchester), l'IBM 701, et le MANIAC I de Los Alamos. Le milieu de la décennie la vit céder la place aux tores magnétiques de Forrester, plus fiables et plus denses. En 1998, pour célébrer le cinquantenaire du Manchester Baby, Chris Burton dirigea la construction d'une réplique fonctionnelle, jetant une lumière nouvelle sur les problèmes qu'avait affrontés l'équipe de 1948.

Association for Computing Machinery

En 1947, un petit groupe de pionniers, comprenant E. C. Berkeley, R. V. D. Campbell, J. H. Curtiss, H. E. Goheen, J. W. Mauchly, T. K. Sharpless, R. Taylor et C. B. Tompkins, décide de créer une organisation informelle rassemblant les personnes intéressées par les nouvelles machines de calcul et de raisonnement. Le 25 juin 1947, ils diffusent une note proposant la création d'une *Eastern Association for Computing Machinery*.

Le 15 septembre 1947, l'université Columbia accueille ces 78 pionniers.

Curtiss prend la présidence, Mauchly la vice-présidence, et Berkeley, cheville ouvrière du groupe, assume le secrétariat. L'ambition grandit vite. Dès janvier 1948, le qualificatif *Eastern* disparaît au profit d'une vision nationale.

L'organisation prend forme autour de quatre pôles géographiques : Boston, New York, Philadelphie et Washington. La structure s'articule selon deux niveaux d'échanges : les rencontres locales et nationales. Les sections, d'abord pensées comme des foyers d'activité restreints, s'étendent rapidement. Les *chapters* véritablement locaux n'apparaîtront que plus tard.

Les années 1950 voient l'ouverture vers d'autres communautés scientifiques. L'ACM tisse des liens avec l'American Institute of Electrical Engineers et l'Institute of Radio Engineers. Ces collaborations donnent naissance, en 1961, à l'American Federation of Information Processing Societies, porte-voix des États-Unis dans le concert international. La croissance rapide bouscule le fonctionnement artisanal des débuts. En 1953, l'évidence s'impose : un secrétaire bénévole ne suffit plus. L'Académie des Sciences de New York prête main-forte pour fournir le personnel et les installations nécessaires, avant que l'association n'établisse son propre quartier général à New York.

Le rayonnement intellectuel se construit par les publications. Le *Journal of the ACM* voit le jour en 1954, suivi du *Communications of the ACM*. Ces revues sculptent le paysage académique, fixent des standards d'excellence et structurent un champ disciplinaire en pleine définition.

Avec les années 1960 surgissent les Special Interest Groups, véritables communautés dans la communauté. Ces groupes spécialisés deviennent des laboratoires d'idées, organisent leurs propres rendez-vous scientifiques, diffusent leurs bulletins et font éclore des sous-disciplines fécondes.

L'ACM s'attaque alors au besoin de la formation à cette science nouvelle. En 1968, sous l'égide de William F. Atchison, le comité des programmes d'études publie ses recommandations, texte fondateur pour l'enseignement informatique.

Les honneurs viennent consacrer les grands bâtisseurs de cette science. Le prix Turing, créé en 1966, s'impose comme la distinction suprême. Alan J. Perlis l'inaugure, avant Wilkes, McCarthy et tant d'autres figures majeures. Par la suite, d'autres récompenses suivent : le Distinguished Service Award (1976) et le prix Grace Murray Hopper (1971).

Face à la multiplication des formations et soucieuse de contribuer aux questions sociales, l'ACM crée un comité d'accréditation en 1968. Elle élabore un code d'éthique et met sur pied un système de médiation pour les problèmes liés à l'informatisation de la société.

La décennie 1970 voit fleurir de nouveaux services comme les cycles

de conférences nationales, les formations professionnelles continues, ou le centre de documentation au siège. L'association répond ainsi aux attentes d'une communauté qui s'étoffe jour après jour.

Dès les années 1960, l'ACM se préoccupe de la préservation de l'histoire de l'informatique. Grace Hopper et Robert Bemer initient une campagne pour préserver les documents historiques de l'informatique. Ces efforts aboutissent à la création d'archives de l'ACM, d'abord hébergées à l'École Moore d'ingénierie électrique de l'Université de Pennsylvanie.

Au fil du temps, l'ACM se mue en une référence mondiale. Son arsenal de publications s'étoffe : des dizaines de revues spécialisées, des milliers d'actes de conférences, une bibliothèque numérique riche de plus de 430 000 articles. Chaque année, plus de 275 événements rassemblent ses 118 000 membres venus de plus de cent pays. La bibliothèque numérique est un trésor accessible, regroupant l'intégralité des publications depuis 1951. Des outils d'analyse bibliométrique, des profils d'auteurs et d'institutions, des statistiques d'usage enrichissent cette mine d'or scientifique.

Durant trois quarts de siècle, l'ACM a façonné l'informatique comme discipline et comme métier. Par ses publications, ses conférences, ses formations et ses distinctions, elle est une boussole dans le monde numérique.

Transistor

L'hiver 1947 voit naître une innovation qui allait métamorphoser notre rapport à la technologie. Dans les Bell Labs, trois hommes : John Bardeen, Walter Brattain et William Shockley conçoivent le transistor, un minuscule dispositif dont personne ne soupçonne encore la portée. Leur travail leur vaudra le prix Nobel de physique en 1956.

Cette découverte ne doit rien au hasard. Dès les années 1930, Mervin Kelly, directeur de recherche aux Bell Labs, avait pressenti les limites des tubes à vide qui équipaient tous les équipements électroniques. Ces tubes, fragiles comme du cristal, engloutissaient l'énergie et occupaient un espace considérable. Kelly cherchait une alternative et misait sur les semiconducteurs, matériaux à mi-chemin entre conducteurs et isolants, alors peu compris par la communauté scientifique.

Pour explorer cette voie, Kelly bâtit patiemment une équipe pluridisciplinaire. Il attire William Shockley et Dean Woolbridge en 1936, puis James Fisk et Charles Townes trois ans plus tard. Walter Brattain, déjà présent aux Bell Labs depuis 1929, rejoint l'aventure. L'équipe tient des séminaires

pour déchiffrer la physique des semi-conducteurs. Leurs premiers essais sur l'oxyde de cuivre tournent court. À l'aube de la guerre, Russell Ohl propose de se concentrer sur le silicium.

Le conflit mondial donne un coup d'accélérateur à ces recherches. Les radars exigent des fréquences trop élevées pour les tubes à vide. Le germanium et le silicium sont les candidats idéaux, d'autant que l'armée finance des techniques de purification sophistiquées pour ces matériaux. En 1945, Kelly restructure ses laboratoires et crée une section dédiée à la physique des états solides sous la houlette de Shockley. L'équipe s'enrichit de Walter Brattain, Gerald Pearson et John Bardeen, fraîchement recruté.

L'automne 1947 voit une accélération fulgurante des travaux, période que Shockley surnommera plus tard le « mois magique ». Le 17 novembre, Brattain et Ralph Gibney découvrent qu'un dispositif à semi-conducteur plongé dans un électrolyte produit un effet d'amplification. Les expériences s'enchaînent avec du germanium. Le 16 décembre 1947, Bardeen et Brattain obtiennent une amplification notable avec un dispositif à deux points de contact rapprochés. Le transistor vient de naître.

L'histoire s'écrit dans les semaines suivantes. Entre décembre 1947 et février 1948, Shockley élabore la théorie du transistor à jonction, qui s'avérera industriellement viable. Il comprend que l'amplification provient d'une injection de porteurs de charges minoritaires et imagine un sandwich de trois couches semi-conductrices. Le 23 février 1948, une expérience de John Shive valide brillamment cette hypothèse.

La fabrication du premier transistor à jonction tel que l'imaginait Shockley exige deux années supplémentaires de mise au point. Gordon Teal et Morgan Sparks y parviennent en avril 1950, grâce à des innovations métallurgiques. En novembre 1950, Shockley publie *Electron and Holes in Semiconductors*, bible instantanée pour les scientifiques du domaine.

Les premières applications du transistor surprennent ses créateurs. Les réseaux téléphoniques d'AT&T, qui sont pourtant à l'origine de l'invention, n'adoptent cette technologie qu'à la fin des années 1950. En revanche, les transistors conquièrent les radios portables, les appareils auditifs et surtout l'armée. Le programme de missile Minuteman (1958-1962) intègre des transistors dans son système de guidage informatique.

Les ordinateurs transistorisés éclosent dès le début des années 1950. Les Bell Labs conçoivent le TRADIC, premier ordinateur embarqué à transistors. IBM lance sa calculatrice transistorisée, le modèle 608, en 1957. Philco commercialise ses ordinateurs TRANSAC basés sur sa technologie de transistors à barrière de surface. Texas Instruments, General Electric et RCA proposent des transistors spécifiquement conçus pour l'informatique.

Cette mutation technologique bouleverse aussi les modes de production.

Fairchild Semiconductor, fondée en 1957 par huit transfuges de l'entreprise de Shockley, dont Gordon Moore et Robert Noyce, invente de nouvelles méthodes de fabrication. Ces innovations façonnent peu à peu ce qui deviendra la Silicon Valley, épicentre mondial de l'industrie des semi-conducteurs.

Cybernétique

Au cœur des turbulences de la Seconde Guerre mondiale, tandis que les grandes puissances mobilisaient leurs scientifiques, naquit une science nouvelle : la cybernétique. Cette discipline, qu'on qualifierait de transversale, vit le jour grâce aux travaux de Norbert Wiener, mathématicien au MIT.

En 1940, face à la menace des bombardiers allemands, Wiener s'attaqua à un problème technique d'une rare complexité : comment abattre des avions filant à 600km/h à des altitudes vertigineuses ? Par un prédicteur de trajectoire baptisé « AA predictor », conçu en collaboration avec Julian Bigelow.

L'intuition géniale de Wiener fut de considérer l'ensemble pilote-avion comme un système unique, dont les comportements obéissaient à certaines lois statistiques prévisibles. Cette approche rompait radicalement avec la vision mécaniste traditionnelle. Pour concrétiser cette idée, Wiener puisa dans trois technologies naissantes : les radars, les servomécanismes et les calculateurs analogiques.

Mais l'apport de Wiener dépassa largement le cadre militaire initial. De ces travaux naquit une réflexion essentielle sur les mécanismes de contrôle et de communication, tant dans les systèmes artificiels que naturels. Le terme « cybernétique », du grec *kubernêtikê* (art du pilotage), fut choisi pour traduire cette vision unificatrice.

Au cœur de la pensée cybernétique se trouvent quelques notions fondatrices : la rétroaction, l'information comme grandeur mesurable, et les systèmes autorégulés. Wiener postula qu'un système cybernétique, qu'il soit vivant ou artificiel, doit nécessairement posséder un modèle interne de son environnement pour interagir efficacement avec lui. Ce modèle guide l'extraction d'informations pertinentes via des capteurs, leur traitement selon des règles internes, puis l'action sur l'environnement via des effecteurs, le tout dans une boucle ininterrompue.

La parution de *Cybernetics : Or Control and Communication in the Animal and the Machine* en 1948 marqua l'avènement officiel de cette discipline. L'ouvrage suscita un enthousiasme immédiat dans les cercles scien-

tifiques internationaux, comme en témoigne sa publication simultanée aux États-Unis et en France chez Hermann.

La cybernétique connut toutefois des destins divergents selon les pays. Aux États-Unis, sous l'égide du MIT et de chercheurs comme Jay Forrester, elle s'inscrivit résolument dans les sciences de l'ingénieur, donnant naissance à des applications concrètes comme le système SAGE de défense antiaérienne. En France, elle prit davantage la forme d'une réflexion théorique et philosophique, parfois au détriment de ses applications pratiques.

Les fameuses conférences Macy, tenues entre 1946 et 1953, jouèrent un rôle déterminant dans l'expansion de cette discipline. Ces rencontres extraordinaires rassemblèrent des esprits brillants venus d'horizons variés : mathématiciens, ingénieurs, neurologues, psychologues et anthropologues s'y côtoyaient pour explorer les ramifications multiples de la pensée cybernétique.

L'après-guerre vit Wiener prendre ses distances avec la recherche militaire, horrifié par les destructions d'Hiroshima et Nagasaki. Il orienta ses travaux vers des applications civiles et médicales, voyant dans la cybernétique un outil de lutte contre l'entropie sociale.

L'informatique moderne porte l'empreinte indélébile de cette pensée. Les concepts de rétroaction, de traitement de l'information et de modélisation des systèmes complexes constituent désormais le socle conceptuel de nos systèmes numériques. L'attention que Wiener portait aux interfaces homme-machine trouve un écho frappant dans nos préoccupations actuelles.

La cybernétique a nourri plusieurs domaines adjacents : la théorie de l'information développée par Claude Shannon, la théorie des jeux formalisée par John von Neumann, et les premières réflexions sur l'intelligence artificielle. Ces courants parallèles ont progressivement construit l'édifice théorique sur lequel repose l'informatique du XXI^{e} siècle.

Dans les années 1960-1970, les limites de la cybernétique devinrent manifestes. Sa prétention à fournir un cadre explicatif universel se heurta à la complexité irréductible de nombreux phénomènes naturels et sociaux. Ses ambitions totalisantes furent revues à la baisse, mais ses concepts fondamentaux gardèrent leur pertinence pour comprendre les systèmes d'information et de contrôle.

Dans notre monde numérique saturé d'informations, où les systèmes complexes s'imbriquent les uns dans les autres, les intuitions de Wiener gardent une étonnante fraîcheur. Si son champ d'application s'est précisé depuis, sa vision d'une science du contrôle et de la communication n'a rien perdu de sa force explicative. Les principes qu'il a dégagés continuent d'éclairer notre compréhension des systèmes informatiques et de leur interaction avec les utilisateurs humains, dans un dialogue incessant entre

l'homme et la machine qu'il avait pressenti dès les années 1940.

Tambour magnétique

On peut le dire avec humilité et sans jugement : le monde informatique a connu d'étranges machines à ses débuts. Parmi elles, un cylindre tournant a dominé trois décennies d'évolution technologique. Gustav Tauschek, inventeur autrichien, créa en 1932 un objet qui allait changer la face du stockage de données : le tambour magnétique.

Un cylindre métallique enduisait sa surface d'une couche ferromagnétique captant les informations binaires. Tout autour du tambour, des têtes de lecture-écriture captaient et inscrivaient des données sous forme de minuscules points magnétisés. La rotation constante du cylindre donnait accès aux informations stockées, dans un ballet mécanique précis et régulier.

Les premières versions des années 1940 montraient déjà les limites techniques de l'époque. La densité d'enregistrement atteignait seulement 50 bits par pouce, avec une vingtaine de pistes par pouce. Le réglage demandait une finesse d'horloger, si bien que les techniciens ajustaient l'écart entre têtes et surface au millième de pouce près, grâce à des vis différentielles et un usinage de haute précision.

L'IBM 650, commercialisé dès 1954, incarna parfaitement cette technologie. Son tambour, fait d'un alliage cobalt-nickel, mesurait 4 pouces de diamètre pour 14 pouces de longueur. Il tournait à la vitesse folle de 12 500 tours par minute sur des roulements à billes ultra-précis. Sa densité d'enregistrement atteignait 50 points magnétiques par pouce, pulsant à 128 kHz. L'organisation des données suivait une logique particulière : stockage parallèle au niveau du bit dans chaque chiffre, mais sériel pour les chiffres d'un même mot. Cinquante mots s'inscrivaient sur une circonférence du tambour, chacun contenant dix chiffres décimaux et un signe. Le système codait le signe moins par le chiffre 8, et le signe plus par le chiffre 9. Un espace équivalent à un chiffre séparait chaque mot, et cinq pistes parallèles encodaient chaque valeur.

David Macklin, programmeur chez Republic Aviation en 1957, raconte les contraintes posées par cette architecture. Les développeurs devaient tenir compte de la position physique réelle des données sur le tambour pour optimiser les temps d'accès. Avec ses 2 000 positions adressables réparties sur des pistes parallèles, la machine imposait une gymnastique intellectuelle constante. Le programmeur calculait l'emplacement de l'instruction

suivante ou laissait l'assembleur SOAP s'en charger. En moyenne, trois ou quatre exécutions par tour complétaient le travail, selon les positions précédentes des données.

La technologie évolua significativement avec l'invention du palier hydrodynamique. Cette avancée, mise au point par IBM pour l'ordinateur SAGE de défense aérienne, permit d'atteindre des densités comparables aux bandes magnétiques sans recourir aux ajustements mécaniques complexes d'antan.

L'héritage du tambour magnétique survit dans certains systèmes UNIX modernes, où `/dev/drum` désigne le périphérique virtuel de swap. Ce nom vient directement de l'utilisation historique du tambour comme support de pagination mémoire.

L'arrivée des mémoires à tores magnétiques sonna le déclin progressif du tambour comme mémoire principale. Toutefois, ces cylindres tournants restèrent utilisés comme extensions mémoire jusque dans les années 1960, grâce à leur fiabilité et leur coût modéré comparé aux alternatives contemporaines.

Le cœur électronique du tambour magnétique battait au rythme d'environ 2 000 tubes, principalement des modèles 5955, 6211, 12AY7, 6AL5, 2D21 et 5687. Les types 6211 et 5965 s'apparentaient au 12AV7 mais répondaient aux tests d'acceptation spécifiques d'IBM. Près de 3 600 diodes à cristal complétaient les circuits logiques. L'alimentation électrique engloutissait 16 à 18 KVA en 208V, 60 cycles, monophasé. Des redresseurs au sélénium fournissaient le courant continu, évitant le recours à une régulation électronique complexe.

Au-delà du simple stockage, le tambour magnétique façonna la conception des premiers ordinateurs et la méthode d'écriture des programmes. Les développeurs plongeaient dans la réalité physique du stockage pour optimiser leurs codes, créant une symbiose matériel-logiciel caractéristique du XX^{e} siècle. Ses limites en termes d'accès et de capacité stimulèrent la recherche d'alternatives. Ces travaux aboutirent aux disques magnétiques, plus rapides et plus volumineux. Le RAMAC d'IBM, premier disque dur commercialisé en 1956, marqua le début de la fin pour le tambour magnétique comme dispositif de stockage principal.

Théorie de l'information de Claude Shannon

Il est difficile d'imaginer notre univers numérique sans les percées théoriques de Claude Shannon, cet américain à l'esprit brillant. Dans son texte fondateur de 1948, *A Mathematical Theory of Communication*, Shannon énonce les bases mathématiques qui sous-tendent aujourd'hui toutes nos communications numériques.

L'histoire commence aux Bell Labs d'AT&T dans les années 1940. Le téléphone et la radio n'y sont plus des curiosités mais des technologies en pleine maturation. Pourtant, ces systèmes manquent d'un cadre théorique solide. Shannon s'attaque à ce vide conceptuel. Son génie réside dans l'intuition d'aborder l'information comme une grandeur mathématique. Il invente le concept lumineux d'entropie informationnelle, mesure qui quantifie l'incertitude d'un message. Au lieu de rester dans le flou, Shannon traduit l'information en équations et formules mathématiques. Il démontre que plus l'entropie monte, plus le message contient d'informations. Voilà l'idée révélatrice qui transforme notre vision du monde.

Son théorème du codage de canal constitue une autre avancée décisive. Il prouve qu'il est possible de transmettre des informations sur un canal bruité avec une probabilité d'erreur presque nulle. La seule condition est de ne pas dépasser la capacité du canal. Cette découverte éclaire d'un jour nouveau les limites théoriques de nos systèmes de communication.

Les concepts de Shannon ont trouvé des applications concrètes dès les premiers temps de l'informatique. Ses travaux ont guidé la création de codes correcteurs d'erreurs robustes. Aucun système de stockage ou de transmission moderne ne fonctionnerait sans eux. Du CD au Wi-Fi, en passant par les communications satellitaires, l'empreinte de Shannon est visible partout.

La compression de données doit tout autant à ce mathématicien visionnaire. Les algorithmes Huffman et Lempel-Ziv, qui vous permettent d'envoyer des photos ou de stocker des films, tirent leurs principes de base des travaux de Shannon. Sans lui, nos disques durs seraient 10 fois plus gros et nos connexions internet 10 fois plus lentes.

Mais l'influence de Shannon ne s'arrête pas aux frontières de l'informatique. Sa théorie a débordé dans de multiples disciplines. En psychologie, elle a inspiré des modèles sur le traitement cérébral de l'information. Les biologistes l'utilisent pour décoder l'ADN et comprendre la transmission génétique. Les économistes y ont puisé des outils pour modéliser les marchés financiers, la physique quantique s'étant approprié certains concepts « shannoniens ».

Avec le temps, sa théorie s'est ramifiée. La théorie de l'information quantique explore les propriétés inédites des systèmes quantiques. L'information

algorithmique étudie les limites fondamentales du calcul et de la complexité. Ces extensions témoignent de la robustesse des idées originales de Shannon.

Plus de 75 ans après sa publication initiale, cette théorie reste une étoile polaire pour l'informatique et les télécommunications du XXI^{e} siècle. Elle guide encore les chercheurs dans des domaines d'avant-garde comme l'apprentissage automatique ou les sciences des données. Qui aurait cru qu'un article publié lorsque les ordinateurs occupaient des salles entières continuerait à éclairer l'ère des puces nanométriques ?

Quand vous envoyez un message qui traverse des milliers de kilomètres sans erreur, quand vous compressez un fichier pour l'envoyer par courriel, quand vous regardez un film en *streaming* sans interruption, vous bénéficiez directement de ses découvertes. Si nos données voyagent à travers le temps et l'espace sans se dégrader, c'est grâce à ses équations. Les mathématiques pures finissent toujours par trouver des applications pratiques, parfois bien au-delà de ce qu'imaginaient leurs créateurs.

Short Code

Le Short Code naquit dans un monde où la programmation s'apparentait davantage à de l'ingénierie électrique qu'à l'écriture de code telle qu'on la conçoit maintenant. En cette fin des années 1940, l'informatique balbutiait avec des machines monumentales remplissant des salles entières pour une puissance de calcul désormais ridicule.

L'histoire débute dans les laboratoires de l'école d'ingénieurs Moore à l'université de Pennsylvanie. C'est là que John Mauchly, physicien reconverti à l'électronique, travailla dès 1941 sur un projet militaire visant à calculer des tables balistiques. Face aux limites des calculateurs, il imagina une machine entièrement électronique : l'ENIAC.

Cette machine colossale, mise en service en 1946, exécutait 5 000 additions par seconde – un exploit technique sans précédent. L'ENIAC souffrait toutefois d'un défaut : l'absence de programme enregistré. Chaque calcul nécessitait de modifier manuellement les connexions des circuits, un travail fastidieux qui pouvait prendre plusieurs jours.

Pour dépasser cette contrainte, Mauchly et son collaborateur J. Presper Eckert conçurent dès 1944 l'EDVAC, intégrant le concept de programme stocké en mémoire. Leur départ de l'université en 1946, suite à des désaccords sur la propriété intellectuelle, retarda cependant le projet qui n'aboutit qu'en 1951. Mauchly proposa alors, en juillet 1949, un langage plus

accessible que le code machine : le Short Code, initialement nommé Brief Code.

Le Short Code se démarquait par sa simplicité conceptuelle. Plutôt que d'écrire directement en binaire, les programmeurs utilisaient des expressions mathématiques. Le langage supportait les nombres à virgule flottante, les variables, les sauts conditionnels et l'appel à des sous-routines stockées séparément. Le manuel de l'UNIVAC illustrait cette syntaxe novatrice par des exemples comme :

```
00 IXABC            I est mis à la valeur X+(A*B*C)
01 BVII             Aller à l'instruction 11 si B est positif
02 HRDW             Imprimer les valeurs de R, D et W
```

Cette approche, bien que rudimentaire selon nos standards actuels, représentait une rupture avec les méthodes antérieures. Les programmes restaient néanmoins complexes et devaient être traduits manuellement en langage machine – le compilateur n'existait pas encore. Les tests montrèrent que le code écrit en Short Code s'exécutait environ 50 fois plus lentement qu'un programme équivalent en langage machine, suscitant le scepticisme de nombreux spécialistes. Malgré ces limites, le Short Code a marqué une étape décisive. Pour la première fois, l'algorithme prenait forme indépendamment de la machine, permettant l'émergence des langages plus évolués comme Fortran (1957) ou Algol (1958). Les concepts qu'il introduisit – expressions mathématiques, variables, branchements, sous-programmes – devinrent les fondements de la programmation moderne.

L'œuvre de Mauchly et Eckert ne se limita pas au Short Code. Après l'ENIAC et l'EDVAC, ils créèrent le BINAC (Binary Automatic Computer, 1949), premier ordinateur à programme enregistré opérationnel, puis l'UNIVAC I (1951), premier ordinateur commercial de l'histoire. William Schmitt développa un prototype de Short Code sur le BINAC, ensuite adapté à l'UNIVAC I, et enfin à l'UNIVAC II. En fondant l'Eckert-Mauchly Computer Corporation dès 1946, ils furent les premiers à tenter de commercialiser ces machines extraordinaires, dont l'utilité restait mystérieuse pour le grand public.

Le Short Code, en libérant partiellement les développeurs de la complexité matérielle, contribua à démocratiser la programmation et accéléra l'essor de l'industrie informatique. Soixante-dix ans après les travaux de Mauchly, la quête de langages expressifs, sûrs et performants, comme Rust par exemple, demeure au cœur des préoccupations des informaticiens du XXI^{e} siècle.

EDSAC

Au lendemain de la Seconde Guerre mondiale, alors que l'Europe pansait ses plaies, un projet prenait forme dans les laboratoires de l'Université de Cambridge. Maurice Wilkes et son équipe y concevaient une machine qui marquerait à jamais l'histoire du calcul automatique : l'EDSAC.

Cette calculatrice automatique à stockage à ligne de délai, construite entre 1947 et 1949, ne ressemblait en rien aux ordinateurs que nous connaissons. Volumineuse, bruyante, gourmande en électricité, l'EDSAC incarnait la première application pratique des théories de von Neumann sur l'architecture des ordinateurs, un cadre conceptuel qui régit encore nos machines actuelles.

Dans le contexte britannique d'après-guerre, où les ressources manquaient cruellement, bâtir un tel engin relevait de la gageure. Le projet s'inscrivait pourtant dans une tradition riche, avec les travaux de déchiffrement à Bletchley Park et l'ACE de Turing au National Physical Laboratory. La singularité de l'EDSAC résidait dans sa vocation généraliste qui, contrairement à ses prédécesseurs dédiés à des calculs spécifiques, se voulait adaptable à tous types de problèmes mathématiques.

Sa mémoire constituait une prouesse technique fascinante. Imaginez des tubes remplis de mercure où des impulsions ultrasonores se propagent, représentant par leur présence ou leur absence les fameux 0 et 1 du langage binaire. Cette « mémoire à ligne de délai » stockait 1024 mots de 17 bits chacun. L'équipe avait aussi conçu des versions miniatures de ces lignes pour les registres de l'unité arithmétique, une solution aussi ingénieuse qu'élégante malgré ses inconvénients pratiques.

L'unité de contrôle microprogrammée, autre innovation majeure, séparait physiquement le décodage des instructions de leur exécution. Cette architecture, codée sur 17 bits avec un format mono-adresse, incluait des sauts conditionnels, un multiplicateur câblé et des opérations de décalage. Les nombres y étaient représentés avec un bit de signe et une virgule fixe, pour manipuler soit des entiers courts, soit des nombres à précision quasi-double en utilisant deux mots mémoire.

En mai 1949, l'EDSAC effectue ses premières démonstrations publiques, devenant ainsi le premier ordinateur opérationnel basé sur l'architecture de von Neumann. Avec ses modestes 650 instructions par seconde et ses 2 kilo-octets de mémoire (un million de fois moins que nos téléphones actuels), la machine trouvait sa place dans la recherche scientifique de Cambridge.

Les années 1950 virent l'EDSAC au cœur d'avancées scientifiques. Rosalind Franklin l'utilisa pour ses travaux sur la structure de l'ADN, tandis que des chercheurs en astronomie, économie ou linguistique se formaient à

son usage. Une véritable communauté informatique naissait autour de cette machine pionnière.

Sur le plan logiciel, David Wheeler créa en 1951 le premier assembleur sur l'EDSAC, remplaçant les fastidieux codes binaires par des instructions mnémoniques. Quelques années plus tard, le langage Autocode d'Alick Glennie préfigurait les langages de haut niveau comme Fortran ou Algol. Des bibliothèques de sous-programmes réutilisables apparurent, initiant les méthodes de programmation structurée du futur.

L'EDSAC a inspiré d'autres universités britanniques pour leurs propres machines, à savoir l'EDSAC 2 à Cambridge, le MOSAIC à Manchester. Outre-Atlantique, l'Université d'Illinois développa l'ILLIAC, directement dérivé des plans partagés généreusement par Wilkes et son équipe.

Le démantèlement de l'EDSAC en 1958, après une décennie de loyaux services, marqua la fin d'une époque mais non celle de son héritage. Sans le savoir, les utilisateurs de l'EDSAC manipulaient déjà les concepts fondamentaux que nous utilisons quotidiennement : programme stocké en mémoire, architecture séquentielle, jeu d'instructions et sous-programmes.

Application Programming Interface

Les Application Programming Interfaces, connues sous l'acronyme API, ont modelé notre environnement numérique. Leur histoire débute dans l'Angleterre d'après-guerre, au sein des laboratoires de Cambridge. Maurice Wilkes et David Wheeler, deux pionniers méconnus, élaborèrent pour l'ordinateur EDSAC une bibliothèque modulaire de programmes. Ces chercheurs britanniques rangèrent leurs créations sur bandes perforées dans une simple armoire métallique, accompagnées d'un catalogue détaillant leur usage. Ce modeste dispositif constitue la première trace documentée d'une API.

Wilkes et Wheeler formalisèrent leurs travaux en 1951 dans l'ouvrage *The Preparation of Programs for an Electronic Digital Computer*. Ce livre marque la naissance conceptuelle des interfaces standardisées entre composants informatiques, bien avant leur dénomination actuelle.

Le terme « API » fit sa première apparition scientifique en 1968 dans les travaux d'Ira Cotton et Frank Greatorex. Leur article « Data structures and techniques for remote computer graphics », présenté lors d'une conférence de l'AFIPS, décrivait des moyens d'abstraire les spécificités des périphériques graphiques. Les programmeurs Fortran utilisaient alors des appels de sous-programmes pour s'affranchir des contraintes matérielles.

Les années 1970 virent l'extension du concept aux systèmes de données. En 1974, CJ Date publia « The Relational and Network Approaches : Comparison of the Application Programming Interface », intégrant ces interfaces dans l'architecture ANSI/SPARC des systèmes de gestion de bases de données. Cette contribution scientifique élargit considérablement leur champ d'application.

L'informatique réseau des années 1980 transforma les besoins des développeurs, désormais confrontés à l'accès aux bibliothèques sur différentes machines. Les API évoluèrent pour prendre en charge les appels de procédure à distance. Cette période vit naître des interfaces favorisant la compatibilité entre plateformes disparates.

La révolution Internet des années 1990 poussa les API vers l'échange de données entre applications via des protocoles normalisés. Carl Malamud proposa en 1990 une définition qui traversa les époques : un ensemble cohérent de services mis à disposition d'un programmeur pour accomplir des tâches spécifiques. Cette décennie prépara le terrain aux services web émergents.

L'an 2000 fut marquant avec la thèse doctorale de Roy Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. Ce travail établit REST comme protocole de référence pour les communications entre systèmes connectés. Toujours en 2000, Salesforce présenta la première API moderne lors de la conférence IDG Demo, inaugurant ainsi une nouvelle ère technologique.

Durant cette décennie, les services en ligne se multiplièrent. Salesforce, eBay et Amazon innovèrent en exploitant HTTP pour diffuser des données lisibles par machine, formatées en JSON ou XML, via des API web. Du petit acteur jusqu'aux géants industriels, les entreprises adoptèrent massivement le modèle fondé sur le cloud et les API. Amazon joua un rôle précurseur en exigeant que toutes les ressources numériques partagées disposent d'une API.

À partir de 2010, l'explosion des réseaux sociaux catalysa l'évolution des API. Les organisations cherchaient des solutions économiques pour créer des applications adaptables. Ces interfaces simplifiaient l'intégration avec des services externes comme les processeurs de paiement ou les systèmes de relation client. L'arrivée de Kubernetes vers 2015 accéléra la transition vers des architectures distribuées composées de microservices autonomes, chacun doté de sa propre API.

La crise sanitaire de 2020 intensifia notre dépendance aux services web, démultipliant l'usage des interfaces de programmation. Leur rôle s'étendit aux objets connectés et à la création d'intelligences artificielles. Les organisations déployèrent leurs applications entre différents fournisseurs cloud,

rendant les API aux centre de la communication entre les services.

Actuellement, l'évolution se concentre sur deux axes majeurs. La sécurité est prioritaire face aux vulnérabilités croissantes des systèmes distribués. Le principe de *zero trust* s'impose, vérifiant systématiquement l'authentification et les autorisations, y compris pour les requêtes internes. Les mécanismes de limitation de débit protègent contre les attaques par déni de service, tandis que l'observabilité détecte et résout les incidents.

La gouvernance des API établit en parallèle des règles pour leur conception, déploiement et maintenance au sein des organisations. L'objectif est de garantir leur cohérence, sécurité et évolutivité en harmonie avec la stratégie d'entreprise.

Les API ont bouleversé la façon dont les développeurs créent des applications, avec des abstractions toujours plus sophistiquées. Ce qui commença comme un simple échange entre deux ordinateurs dans un laboratoire universitaire britannique forme aujourd'hui l'épine dorsale des systèmes numériques du XXI^{e} siècle, particulièrement dans l'univers distribué du cloud computing.

Whirlwind I

Au cœur du MIT, le projet Whirlwind I démarra en 1944, sous l'impulsion de l'US Navy. À l'origine, l'ambition était modeste : fabriquer un simulateur de vol pour l'entraînement des pilotes. Jay Forrester, chercheur au laboratoire de servomécanique, prit les rênes du projet l'année suivante. L'idée initiale d'une machine analogique fut vite écartée au profit d'une approche numérique, que la Navy valida en 1946.

Cette décision transforma radicalement la portée du projet. Au lieu d'un simple simulateur, le MIT s'engageait dans la création d'un calculateur numérique universel, capable de répondre aux exigences du simulateur tout en ouvrant des perspectives vers d'autres applications scientifiques. La machine centrale devint pleinement opérationnelle en 1951.

La vocation première du Whirlwind I marqua sa conception. Dès le début, l'équipe le pensa comme un système temps réel. Cette orientation nécessitait une mémoire ultrarapide. La solution retenue fut une mémoire à tubes cathodiques, la plus véloce disponible à ce moment. Un rapport de 1952 notait toutefois que cette mémoire restait « le facteur le plus important affectant la fiabilité du système Whirlwind I ».

Pour répondre à ce problème, l'équipe élabora des procédures rigoureuses de test systématique, visant à repérer les défaillances matérielles avant qu'elles ne compromettent les calculs. En parallèle, l'usage militaire grandissant de la machine exigeait une fiabilité sans faille. Dans le climat tendu de guerre froide, le Whirlwind, dont le financement passa de l'Office of Naval Research à l'US Air Force, s'intégra au dispositif de défense américain.

Sa version de production, l'AN/FSQ-7, devint un élément du système SAGE de surveillance aérienne. Insatisfaits des limites des tubes cathodiques, les chercheurs cherchèrent des alternatives. Vers la fin des années 1940, Jay Forrester et d'autres scientifiques explorèrent l'utilisation de tores magnétiques. William Papian, membre de l'équipe, mentionna dans ses notes les travaux de Harvard sur les « lignes à retard magnétiques statiques ».

L'été 1953 vit l'installation d'une mémoire à tores sur le Whirlwind. Un rapport du projet soulignait deux avantages : « la mémoire à tores magnétiques a deux grands avantages : (1) une plus grande fiabilité avec en conséquence une réduction du temps de maintenance consacré à la mémoire ; (2) un temps d'accès plus court (le temps d'accès des tores est de 9 microsecondes contre environ 25 microsecondes pour les tubes), ce qui augmente la vitesse de fonctionnement de l'ordinateur ». Cette innovation fit du Whirlwind le premier grand calculateur doté de cette technologie, qui domina le marché jusqu'aux années 1970.

L'architecture du Whirlwind I reposait sur un modèle à programme enregistré avec des mots de 16 bits. Sa puissance atteignait 20 000 instructions par seconde, avec une mémoire vive de 2048 mots. Il utilisait des instructions à adresse unique et des nombres en virgule fixe en complément à 9. Sa structure se divisait en trois parties : une unité de stockage avec 32 tubes électrostatiques, un élément arithmétique pour les calculs, et un organe de contrôle orchestrant l'ensemble. Les composants électroniques reposaient principalement sur des bascules pour mémoriser les états binaires et des tubes de porte pour diriger les impulsions.

La préoccupation centrale était la vitesse. L'objectif initial de 50 000 opérations par seconde ne fut pas atteint, mais les 20 000 opérations effectives représentaient déjà une prouesse. Les temps d'exécution – 3 microsecondes pour une addition, 16 pour une multiplication – surpassaient largement ceux des machines contemporaines. Le jeu d'instructions, réduit à une trentaine de commandes, incluait des sauts conditionnels rendant la machine auto-modifiable selon les résultats. Des instructions spécialisées, comme les transformations automatiques de coordonnées, accéléraient encore les calculs.

L'interface homme-machine du Whirlwind I était audacieuse. Au-delà du lecteur photoélectrique de bande perforée et des machines à écrire classiques, la machine disposait d'écrans cathodiques – alphanumériques ou vectoriels – avec capture photographique des résultats. Ce dispositif avancé donnait aux programmeurs un retour visuel immédiat sur leurs travaux.

Le cycle de programmation préfigurait celui que nous connaissons aujourd'hui. Le programmeur découpait d'abord son problème en étapes élémentaires, les traduisait en instructions, que des opérateurs saisissaient via un équipement spécial. Ce code était converti en langage intermédiaire, puis en binaire, avant d'être stocké sur bande perforée ou magnétique. Le contrôleur chargeait ensuite les instructions séquentiellement depuis la mémoire, les décodait et envoyait les signaux nécessaires à leur exécution.

Les applications du Whirlwind I touchèrent de nombreux domaines : économie, magnétisme, radar, conception mécanique, traitement du signal. Mais son héritage le plus durable est sans doute ses avancées en informatique temps réel appliquée au contrôle industriel et aérien. Cette machine extraordinaire contribué à l'amélioration de l'architecture des ordinateurs et accéléré le développement de l'informatique, tant matérielle que logicielle.

IV

1950

La naissance silencieuse d'un monde calculé

Au lendemain de 1945, tandis que l'Europe pansait ses plaies, un nouvel ordre mondial s'instaurait. Les décombres fumants de la Seconde Guerre mondiale laissaient place à une confrontation idéologique entre deux géants : les États-Unis et l'URSS. Cette rivalité, moins sanglante mais tout aussi féroce, allait façonner le développement technologique durant la décennie 1950-1960.

Washington et Moscou déversaient des fonds colossaux dans leurs programmes militaires et spatiaux. Les bombes devenaient plus destructrices, les missiles plus précis, les avions plus rapides. Chaque avancée technique exigeait des calculs plus complexes. Les vieux bouliers et règles à calcul ne suffisaient plus. Il fallait des machines capables de traiter rapidement des équations à plusieurs variables, simuler des explosions nucléaires ou calculer des trajectoires balistiques en quelques heures au lieu de quelques mois.

La guerre avait prouvé l'utilité des calculateurs. Le chiffre d'Enigma cassé par les Britanniques à Bletchley Park avait montré qu'une machine bien conçue pouvait résoudre des problèmes jugés insolubles. Aux États-Unis, le « National Security Act » de 1947 créait la CIA et la NSA. Ces nouvelles agences se nourrissaient de données comme les messages interceptés, les rapports d'espions ou des photos de reconnaissance aérienne. Le chiffrement et le déchiffrement devenaient des enjeux stratégiques majeurs.

Loin des préoccupations militaires, la société civile traversait une mutation sans précédent. L'économie occidentale connaissait une croissance fulgurante. Les entreprises s'agrandissaient, les administrations s'étoffaient. Partout, les papiers s'accumulaient : fiches de paie, bons de commande, factures, courriers, ou dossiers médicaux. Les méthodes traditionnelles de classement et d'archivage montraient leurs limites. Les cartes perforées d'Herman Hollerith, déjà anciennes, ne répondaient plus aux besoins modernes.

Le *baby-boom* transformait la démographie. Des millions d'enfants naissaient chaque année, exigeant des systèmes de gestion publique plus efficaces. En Europe, les nouvelles politiques de protection sociale généraient des montagnes de formulaires. Aux États-Unis, la Social Security Administration croulait sous les dossiers. Comment suivre efficacement tous ces citoyens, leurs cotisations, leurs droits ?

Jadis réservés aux plus fortunés, les services bancaires se démocratisaient. Le nombre de comptes explosait, les transactions se multipliaient. Les caissiers et les comptables, armés de leurs livres de comptes et de leurs machines à calculer mécaniques, peinaient à tenir le rythme. Le secteur financier réclamait des solutions plus rapides et plus fiables.

Dans les usines, l'automatisation progressait à grands pas. Les chaînes de montage adoptaient des systèmes de contrôle électroniques. La théorie cybernétique de Norbert Wiener trouvait des applications concrètes. Les principes de rétroaction (« *feedback* »), utilisés pour guider les missiles pendant la guerre, servaient désormais à réguler la production industrielle.

Une alliance féconde se formait entre universités, industriels et militaires. Le MIT, Harvard, l'Université de Pennsylvanie, mais aussi les laboratoires privés comme Bell Labs ou IBM, devenaient des creusets d'innovation. Une nouvelle génération de chercheurs et d'ingénieurs s'attaquait aux problèmes du calcul automatique. Ce modèle de coopération, que le président Eisenhower qualifierait plus tard de « complexe militaro-industriel », accélérait considérablement le progrès technique.

Le nouveau métier de programmeur faisait son apparition. Ces spécialistes traduisaient les besoins et les problèmes concrets en instructions compréhensibles par les machines. Fait notable, de nombreuses femmes occupaient ces postes, comme Grace Hopper, qui travaillait sur le premier compilateur. Cette marine mathématicienne ouvrait la voie à des méthodes de programmation plus intuitives, libérant les utilisateurs de la complexité des langages machine.

La multiplication des fabricants de calculateurs créait un problème inattendu d'incompatibilité. Chaque constructeur développait ses propres standards. Les machines ne se comprenaient pas entre elles, les programmes écrits pour un modèle ne fonctionnaient pas sur un autre. Les premiers

efforts de normalisation apparaissaient timidement, prémices d'une problématique qui marquerait durablement l'histoire informatique.

Les médias découvraient ces mystérieuses machines. Les journaux populaires évoquaient avec fascination ou inquiétude ces « cerveaux électroniques ». Hollywood s'emparait du sujet, tantôt présentant l'ordinateur comme un assistant merveilleux, tantôt comme une menace pour l'humanité. Ces représentations contradictoires façonnaient la perception publique d'une technologie encore mal comprise.

L'automatisation soulevait des inquiétudes sociales. Certains métiers semblaient menacés de disparition et les syndicats s'alarmaient des risques de chômage massif. Dans les milieux académiques, les premiers débats sur l'intelligence artificielle prenaient forme, notamment après la conférence de Dartmouth en 1956 où le terme fut officiellement adopté.

Le réseau téléphonique mondial s'étendait et de nouveaux câbles traversaient les océans, reliant les continents. La qualité des communications s'améliorait et les coûts diminuaient. La transmission de données à distance devenait techniquement possible, ouvrant la voie aux futurs réseaux informatiques.

Les travaux théoriques de Claude Shannon sur l'information trouvaient leurs premières applications pratiques. Ses concepts mathématiques de codage et de correction d'erreurs rendaient les communications plus fiables. Le bit s'imposait progressivement comme unité fondamentale de mesure de l'information.

Le transistor, inventé en 1947 aux Bell Labs, remplaçait peu à peu les tubes à vide. Plus petit, plus fiable, moins gourmand en énergie, il permettait de concevoir des appareils plus compacts. Dans la région de San Francisco, de jeunes entreprises spécialisées dans les semi-conducteurs s'établissaient dans ce qui deviendrait la Silicon Valley.

Les sciences naturelles adoptaient ces nouveaux outils de calcul. Astronomes, physiciens, chimistes découvraient les possibilités offertes par l'automatisation. Des disciplines émergentes comme la biologie moléculaire bénéficiaient de ces capacités de traitement accrues. La météorologie commençait à utiliser les calculateurs pour améliorer ses prévisions, avec des résultats modestes mais prometteurs.

Cette décennie 1950-1960 a été le départ d'une transformation sociale. Les innovations techniques se mêlaient inextricablement aux mutations économiques et culturelles. L'impact sur nos sociétés demeure palpable au XXI^{e} siècle, alors que nous naviguons dans un monde toujours plus numérisé et interconnecté.

Code de Hamming

Richard Wesley Hamming travaillait aux Bell Labs sur le Model V, un des premiers ordinateurs à relais destiné à l'Aberdeen Proving Ground, lorsqu'il il observe une réalité contraignante : cette machine composée de 8 900 relais mécaniques connaît quotidiennement deux à trois défaillances. Cela représente approximativement une panne tous les deux à trois millions d'opérations. Le problème ne réside pas tant dans la détection des erreurs, puisque les circuits d'auto-vérification s'en chargent correctement. La difficulté surgit quand la machine fonctionne sans surveillance, notamment la nuit et durant les week-ends. Les calculs s'arrêtent net. Parfois l'ordinateur reprend son travail sur un nouveau problème, sans toutefois garantir la continuité des opérations précédentes.

Cette situation technique très concrète a poussé Hamming vers une réflexion plus profonde. Il cherchait un dépassement de la simple détection d'erreurs pour créer un système capable de corriger automatiquement ces défaillances. Une véritable rupture avec les approches antérieures.

La particularité du contexte technologique donne du relief à sa démarche. Les ordinateurs commençaient à supplanter les centraux téléphoniques comme systèmes de traitement d'information les plus complexes. Ces deux architectures présentaient pourtant une différence clé. Un central téléphonique dispose de nombreux chemins parallèles relativement autonomes les uns des autres. Une erreur sur l'un de ces chemins n'affecte qu'un nombre restreint d'appels. De surcroît, les erreurs échappant à la détection automatique finissent signalées par les clients eux-mêmes. Un ordinateur, lui, suit généralement un chemin unique où l'information passe maintes fois par les mêmes composants avant d'atteindre le résultat final. Une seule erreur non détectée risque d'invalider l'ensemble des calculs ultérieurs.

Face à cette structure problématique, Hamming a élaboré une approche mathématique novatrice. Son point de départ était l'information codée sous forme binaire. Cette représentation s'accordait naturellement avec les relais ouverts ou fermés, les circuits à bascule, les points et traits du morse, ou encore les bandes perforées. Sa véritable trouvaille réside dans l'ajout aux données de bits supplémentaires, nommés bits de parité, qui ne détectent pas seulement les erreurs mais les localisent avec précision.

Le cœur du code de Hamming tient dans un principe d'une simplicité trompeuse : chaque bit de parité vérifie un ensemble spécifique de positions dans le message. Par un choix judicieux de ces ensembles, on identifie exactement quelle position a subi une altération. À titre d'exemple, dans

un code à sept positions dont quatre dédiées à l'information, trois bits de parité suffisent pour corriger n'importe quelle erreur simple. Cette redondance, définie comme le rapport entre le nombre total de bits utilisés et le nombre minimum nécessaire pour transmettre l'information, constitue le prix à payer pour garantir l'intégrité des données.

Pour la première fois, lors de la publication de Hamming dans le *Bell System Technical Journal* en 1950, un cadre théorique rigoureux aborde la question de la fiabilité des systèmes numériques. Hamming introduit notamment une représentation géométrique des codes correcteurs, où les séquences de bits sont des points dans un espace à n dimensions. La distance entre deux points, mesurée par le nombre de positions où leurs coordonnées diffèrent, se transforme en un outil d'analyse puissant pour évaluer les capacités de détection et de correction des codes.

Cette approche géométrique a permis à Hamming de prouver l'optimalité de ses codes dans plusieurs cas significatifs. Il démontre qu'aucun code ne peut détecter les erreurs simples avec moins de redondance que sa construction, et établit des résultats analogues pour la correction d'erreur simple et la détection d'erreur double. Ces démonstrations théoriques confirment que les compromis proposés entre redondance et capacité de correction atteignent le meilleur équilibre possible sous les hypothèses considérées.

Richard Wesley Hamming a entrevu d'autres applications au-delà du domaine des ordinateurs. Trois situations particulières tirent un bénéfice notable de ses codes : le fonctionnement sans surveillance sur de longues périodes avec un équipement de secours minimal, les grands systèmes étroitement interconnectés où une défaillance isolée risque de paralyser l'ensemble de l'installation, et la transmission de signaux en présence de bruit quand sa réduction s'avère impossible ou trop coûteuse.

Sa vision s'est révélée prophétique. L'avènement de l'électronique a certes rendu les composants plus fiables que les relais mécaniques, par opération. Toutefois, l'accroissement considérable de la vitesse et de la complexité des systèmes a maintenu la question des erreurs au premier plan des préoccupations. Les codes de Hamming et leurs descendants peuplent aujourd'hui les mémoires d'ordinateur, les dispositifs de stockage et les systèmes de communication.

L'héritage de Hamming vit dans des techniques de correction d'erreurs sophistiquées intégrées aux technologies modernes. Des codes plus complexes traitent désormais des motifs d'erreurs variés, tous reposant sur les fondements posés par Hamming : l'ajout contrôlé de redondance pour détecter et corriger les erreurs, et l'analyse mathématique rigoureuse des capacités et limites des codes correcteurs. Sa réflexion sur la fiabilité numérique a

fait naître une vision renouvelée de la conception des ordinateurs. La gestion des erreurs n'y apparaît plus comme une question secondaire mais comme partie intégrante de l'architecture système. Une perspective qui garde toute sa pertinence à notre époque.

Mémoire à tores magnétiques

Les ferrites magnétiques ont bouleversé le monde de l'informatique au milieu du XX^{e} siècle. Ces petits anneaux de ferrite, capables de conserver un état magnétique stable, ont marqué l'histoire du stockage informatique pendant près de vingt ans. Leur principe repose sur une élégante application des lois physiques d'hystérésis magnétique : chaque tore pouvait être magnétisé dans l'un de deux états distincts, représentant ainsi parfaitement les valeurs binaires 0 et 1.

An Wang, chercheur au Harvard Computation Laboratory, décrit dès 1949 cette technique qui garde l'information y compris après coupure de l'alimentation électrique. Le véritable changement tient dans la manipulation de cet état magnétique par de simples impulsions électriques traversant le tore.

Jay Forrester transforme cette théorie en réalité concrète. Au MIT, il démontre en 1951 la viabilité d'un système de stockage tridimensionnel basé sur ces minuscules anneaux. William Papian, travaillant sur le projet Whirlwind, et Jan Rajchman chez RCA Laboratory développent dans la même période des architectures qui seront massivement adoptées. Leur travaux débouchent sur l'intégration de cette technologie dans des ordinateurs comme le Mark IV de Harvard.

La construction d'une telle mémoire repose sur un maillage bi ou tridimensionnel d'anneaux, chacun traversé par de nombreux conducteurs : lignes X et Y de sélection, fil de lecture/écriture, et parfois un fil d'inhibition. Pour sélectionner un tore spécifique, il faut appliquer simultanément des courants dans les lignes X et Y correspondantes. Cette astuce de coïncidence de courant résout le problème d'adressage individuel tout en limitant les circuits de commande nécessaires.

Fait notable, la lecture efface le contenu du tore. Il faut donc réécrire immédiatement la donnée si on souhaite la conserver. Malgré cette contrainte, les avantages de cette technologie ont séduit l'industrie par la conservation des données sans alimentation, l'accès rapide en quelques microsecondes, l'excellente fiabilité et la densité de stockage remarquable pour l'époque.

La fabrication exigeait une minutie extrême. L'assemblage restait largement manuel, le tissage des fils à travers les minuscules anneaux demandant patience et précision. Des tests semi-automatiques permettaient de vérifier environ 200 tores par heure avant leur intégration. Une fois la matrice assemblée, remplacer un tore défectueux relevait du casse-tête.

Burroughs Corporation a exploré une variante intéressante avec la mémoire *rope*. Cette configuration particulière des tores et des fils permettait le stockage permanent d'informations, idéal pour conserver programmes et routines système. L'ordinateur de navigation du programme Apollo a notamment tiré parti de cette technologie.

Les performances n'ont cessé de progresser. Les temps d'accès sont passés à moins de 2 microsecondes. Les signaux de sortie, d'abord de quelques millivolts, ont atteint plusieurs centaines de millivolts, simplifiant grandement les circuits de détection. La standardisation des composants a entraîné une baisse des coûts. Les tores magnétiques ont ainsi trouvé leur place dans les grands systèmes et dans les calculateurs industriels et embarqués.

L'avènement des mémoires à semi-conducteurs amorce le déclin de cette technologie à la fin des années 1960. Bénéficiant de l'essor fulgurant de la microélectronique, ces nouvelles mémoires offraient des possibilités de miniaturisation sans précédent. La transition s'est néanmoins étalée dans le temps, la non-volatilité des tores magnétiques restant un atout pour certaines applications critiques.

L'héritage conceptuel des mémoires à tores magnétiques demeure visible dans l'informatique moderne. L'adressage matriciel, la lecture avec rafraîchissement et l'organisation hiérarchique de la mémoire sont autant de principes toujours pertinents. Ces anneaux de ferrite ont rendu possible des ordinateurs plus rapides et plus fiables, contribuant à l'explosion informatique dans de nombreux secteurs d'activité.

Bande magnétique

En 1878, un ingénieur américain du nom d'Oberlin Smith rêve un système d'enregistrement sur fil métallique. Fasciné par le phonographe de Thomas Edison, il publie son idée dans *The Electrical World* dix ans plus tard. Un trait de génie qui ouvre un nouveau chapitre dans l'art de capturer les sons.

Le Danois Valdemar Poulsen transforme cette vision en réalité tangible. Son « Télégraphone » de 1898 grave pour la première fois des signaux sur un

fil d'acier. La preuve est faite que l'enregistrement magnétique fonctionne. Toutefois, sans amplification électronique, l'appareil reste un prototype sans avenir commercial immédiat.

La vraie métamorphose survient dans l'Allemagne des années 1930. Fritz Pfleumer invente en 1928 son fameux « papier sonore », un support recouvert d'oxyde de fer. Cette trouvaille séduit AEG qui s'associe avec BASF pour créer le Magnetophon en 1935. Sur sa bande d'acétate de cellulose enduite d'oxyde de fer s'écrit déjà l'avenir du stockage magnétique.

La Seconde Guerre mondiale accélère l'évolution technique. Walter Weber découvre en 1940 la polarisation par courant alternatif, un procédé qui sublime la qualité sonore. Les radios du Reich diffusent alors des enregistrements d'un tel réalisme que les Alliés les prennent pour des émissions en direct. La guerre terminée, John Mullin, officier américain, rapporte aux États-Unis deux précieux Magnetophons. Ses démonstrations éblouissent l'industrie américaine. Ampex saisit l'occasion et lance en 1948 son modèle 200, premier magnétophone professionnel made in USA.

Les années 1950 structurent la technologie et la standardisent. Dans l'univers informatique, la bande magnétique s'impose comme référence pour l'enregistrement audio et règne en maître pour archiver les données. Sa capacité généreuse et son coût modéré en font la solution idéale pour les premiers *mainframes*. La bande 9 pistes standardise l'échange entre systèmes différents. Le Japon s'éveille à cette technologie, avec Sony en tête de file.

Un bond en avant se produit en 1963. Philips dévoile sa cassette compacte, un format minuscule comparé aux imposantes bobines de l'époque. Le géant néerlandais fait un coup de maître en rendant sa licence accessible gratuitement. Le format explose. Les fabricants japonais perfectionnent la technique avec des matériaux magnétiques novateurs et des têtes de lecture ultrasensibles.

Les décennies 1970-1980 voient jaillir des innovations majeures. Les particules de chrome puis de métal densifient l'enregistrement. Les têtes de lecture s'affinent grâce au Sendust et aux alliages amorphes. La bande magnétique a bouleversé notre rapport au son et à la mémoire. Elle a engendré des pratiques culturelles inédites comme l'écoute nomade, symbolisée par le Walkman de Sony en 1979.

L'ère numérique des années 1980 transforme la bande sans la tuer. Pour l'informatique, DLT (Digital Linear Tape) apparaît en 1984, suivi par DDS (Digital Data Storage) en 1989. Ces formats offrent des capacités toujours plus vastes. Pour les centres de données du XXI^{e} siècle, elle demeure une solution d'archivage privilégiée, avec un rapport coût/téraoctet imbattable.

L'histoire de ce ruban brun démontre une adaptation constante. Né pour

le son, il conquiert les données informatiques, la vidéo, l'archivage professionnel. À l'heure du cloud, la bande conserve un atout maître face aux cybermenaces : sa capacité d'isolation physique (*airgap*). Ce rempart contre les rançongiciels en fait l'ultime protection des données critiques. Cette résistance aux attaques, couplée à sa rentabilité pour stocker d'immenses volumes, lui assure une place durable dans les architectures modernes, où elle complète les disques durs et les solutions cloud.

Ferranti Mark I

Durant l'hiver 1951, l'Université de Manchester reçoit des mains de la société Ferranti une machine qui va changer la donne : le Mark I. Ce calculateur électronique britannique devance de quelques mois l'UNIVAC I américain et s'inscrit dans l'histoire comme le premier ordinateur commercialisé. Cette aventure technique n'aurait jamais vu le jour sans l'intuition de Sir Ben Lockspeiser. Ce haut fonctionnaire du ministère britannique de l'Approvisionnement avait saisi l'importance de relier la recherche universitaire au monde industriel.

En réalité, cette machine prolonge les travaux de F.C. Williams et Tom Kilburn. Ces chercheurs avaient déjà conçu un prototype fonctionnel à l'Université de Manchester. Le gouvernement britannique, désireux de valoriser cette avancée, confie à l'entreprise Ferranti la tâche de transformer l'expérimental en commercial. Si l'architecture générale est fidèle à l'original, les ingénieurs de Ferranti y ajoutent leur touche : un jeu d'instructions revu, des registres d'index supplémentaires (les fameux B-lines) et des fonctions inédites comme un générateur de nombres aléatoires.

La mémoire du Mark I reflète les contraintes technologiques de l'époque. D'un côté, une mémoire électronique construite autour des tubes Williams, rapide mais limitée à 20 480 bits. De l'autre, un tambour magnétique plus lent mais avec 655 360 bits répartis sur 256 pistes de 2 560 bits, double système annonçant l'architecture hiérarchique des mémoires modernes.

Programmer le Mark I relevait presque de l'épreuve initiatique. Alan Turing, figure légendaire du calcul et alors chercheur à Manchester, rédige le premier manuel de cette machine. Son approche consiste en du code machine sans fioritures, sans concessions aux limites cognitives humaines. À Cambridge, pendant ce temps, David Wheeler avait créé pour l'EDSAC un assembleur rudimentaire mais combien plus accessible. Deux visions s'opposaient.

L'interface avec le monde extérieur passait par des bandes perforées à cinq trous. Le flux d'entrée atteignait 400 caractères par seconde grâce au lecteur photoélectrique conçu par Ferranti. En sortie, une perforatrice plus modeste (33 caractères par seconde) et une imprimante télétype traduisaient les résultats des calculs. Un haut-parleur complétait le dispositif, alertant l'opérateur par des signaux sonores quand son attention devenait nécessaire.

Les mathématiciens furent parmi les premiers à s'emparer de cette puissance de calcul inédite. Ils l'utilisèrent pour explorer les nombres de Mersenne ou tester des aspects de l'hypothèse de Riemann. Les physiciens s'en servirent pour des calculs de tracé de rayons en optique. La vitesse de traitement des données ouvrait des possibilités nouvelles pour la recherche, malgré une fiabilité qui laissait parfois à désirer.

L'aventure commerciale du Mark I raconte aussi les difficultés de l'industrie informatique britannique. De 1951 à 1957, neuf exemplaires seulement trouvèrent preneurs : sept Mark I Star et deux Mark I. L'étroitesse du marché britannique limitait les perspectives de croissance. Quand IBM débarqua au Royaume-Uni avec son modèle de location, la concurrence devint féroce pour les constructeurs locaux.

L'histoire du Mark I se lit aussi à travers les relations complexes entre Ferranti et les organismes gouvernementaux, notamment la National Research Development Corporation (NRDC). Le financement public a certes rendu possible cette aventure, mais les négociations souvent tendues entre Ferranti et la NRDC ont parfois ralenti le projet. Sir Vincent de Ferranti défendait âprement les intérêts de son entreprise familiale, cherchant à minimiser les risques financiers dans cette aventure incertaine.

Le contraste entre le potentiel technique britannique et sa concrétisation commerciale saute aux yeux dans les statistiques : en 1960, le Royaume-Uni comptait 240 ordinateurs en fonction, contre 5 400 aux États-Unis. Pourtant, le Ferranti Mark I occupe une place à part dans le panthéon informatique. Sa mémoire à plusieurs niveaux, son système d'index et son architecture générale ont façonné la conception des ordinateurs qui ont suivi. Premier calculateur commercial, il matérialise ce moment où la recherche universitaire rencontre la production industrielle.

Cette expérience a aussi réorienté la trajectoire de Ferranti. Face aux réalités économiques, l'entreprise a progressivement abandonné le marché des grands systèmes pour se concentrer sur les applications militaires, les mini-ordinateurs puis les microprocesseurs. En 1963, Ferranti quittait définitivement le secteur des ordinateurs centraux.

LEO I

Au milieu du XX^{e} siècle, J. Lyons & Co dominait le marché alimentaire au Royaume-Uni. Issue des idées visionnaires de la famille Salmon-Gluckstein à la fin du XIX^{e}, cette entreprise gérait un réseau tentaculaire de salons de thé élégants, restaurants populaires et boulangeries industrielles à travers tout le pays.

Contrairement aux autres sociétés, Lyons cultivait une atmosphère propice à l'innovation technique. La raison en était simple : chaque jour, des milliers de transactions modestes s'accumulaient, chacune générant une marge infime. La survie financière exigeait donc une rigueur presque maniaque dans la gestion administrative. Cette particularité avait poussé les dirigeants à recruter dès 1923 un mathématicien de Cambridge, John Simmons, capable d'apporter une dimension scientifique à leurs méthodes de travail.

Au printemps 1947, deux cadres de l'entreprise, Oliver Standingford et Raymond Thompson, s'envolèrent vers l'Amérique. Leur mission initiale concernait l'étude d'équipements de bureau modernes, mais ils tombèrent par hasard sur des informations concernant l'ENIAC. Ces gigantesques machines de calcul, alors cantonnées aux laboratoires scientifiques, éveillèrent leur curiosité. Ils pressentirent que ces calculs automatisés pourraient transformer radicalement la bureaucratie de Lyons.

Leur intuition les conduisit jusqu'à l'université de Cambridge, où Maurice Wilkes travaillait sur l'EDSAC. À leur retour, ils rédigèrent un rapport audacieux. Non contents de suggérer l'adoption de l'informatique pour leur entreprise, ils proposèrent que Lyons fabrique sa propre machine. Cette idée, qui semblerait aujourd'hui absurde pour une société alimentaire, s'inscrivait parfaitement dans la philosophie d'indépendance technique de l'entreprise, habituée à concevoir ses propres outils industriels.

La direction valida le projet et déboursa 3 000 livres sterling pour soutenir les travaux de Wilkes, s'assurant en échange une assistance technique précieuse. Pour mener à bien cette aventure, l'entreprise embaucha le Dr John Pinkerton, un brillant ingénieur recommandé par Wilkes. David Caminer, responsable des systèmes chez Lyons, reçut la tâche de développer les applications et méthodes de programmation adaptées aux besoins spécifiques de l'entreprise.

La construction du LEO I (Lyons Electronic Office) commença en 1949. Ce colosse occupait plus de 450 mètres carrés et fonctionnait grâce à environ 6 000 tubes à vide répartis dans 21 armoires métalliques imposantes. Bien

que basé sur l'architecture de l'EDSAC, LEO I intégrait des modifications substantielles pour traiter efficacement les données commerciales. Les ingénieurs accordèrent une attention particulière aux systèmes d'entrée-sortie, combinant cartes perforées et bandes de papier pour l'alimentation en données, complétées par des imprimantes et perforateurs pour la restitution des résultats.

Le 29 novembre 1951, LEO I accomplit sa première tâche régulière, l'évaluation hebdomadaire de la production des boulangeries Lyons. Ce programme analysait les chiffres de fabrication de pains, gâteaux et tartes provenant de douze sites différents. Il calculait automatiquement la valorisation en intégrant le coût des matières premières, les salaires et les frais indirects. Aucun ordinateur n'avait jamais effectué pareille tâche de gestion auparavant.

Ce premier succès encouragea l'équipe à développer d'autres applications : gestion des stocks, calcul des meilleurs mélanges de thé, organisation des livraisons aux différents points de vente, facturation automatisée. Un nouveau cap fut franchi le 12 février 1954, lorsque LEO I traita la paie de 1 670 employés des boulangeries, produisant directement les chèques de salaire sans système de secours.

Les prouesses de cette machine attirèrent bientôt l'attention d'organisations extérieures. Le service météorologique britannique l'utilisa pour ses prévisions, l'administration fiscale pour calculer ses barèmes d'imposition, et British Railways pour déterminer les distances entre toutes ses gares. Le succès de ces prestations poussa Lyons à créer en 1954 une filiale spécialisée, LEO Computers Limited, pour commercialiser cette technologie.

Cette histoire raconte un curieux paradoxe : pendant que les fabricants traditionnels d'équipements de bureau et les géants de l'électronique hésitaient, une chaîne de restaurants et de salons de thé créait le premier ordinateur pour une application commerciale viable au monde. LEO I fonctionna jusqu'en janvier 1965, avant de céder sa place aux modèles plus performants LEO II et LEO III qui équipèrent de nombreuses entreprises et administrations britanniques.

La saga LEO se poursuivit à travers une série de fusions industrielles : d'abord avec English Electric en 1963, puis avec d'autres sociétés pour former International Computers Limited en 1968. Le dernier LEO III s'éteignit discrètement en 1981 dans les locaux de la poste britannique.

UNIVAC I

Le premier ordinateur commercial produit cette fois en série a vu le jour sous le nom d'UNIVAC I, acronyme d'*Universal Automatic Computer*. Son histoire démarre avec deux ingénieurs talentueux de l'université de Pennsylvanie : J. Presper Eckert et John Mauchly. Ces visionnaires venaient tout juste d'achever la conception de l'ENIAC durant la Seconde Guerre mondiale quand ils eurent l'intuition du potentiel commercial des ordinateurs.

Fin 1945, alors que l'encre des plans de l'ENIAC séchait à peine, Eckert et Mauchly prirent contact avec le Bureau du Recensement américain. Leur proposition était un ordinateur capable de digérer rapidement les données démographiques grâce à l'utilisation novatrice de bandes magnétiques comme support de stockage.

Au printemps 1946, nos deux ingénieurs quittèrent l'université de Pennsylvanie, décidés à tirer profit de leur savoir-faire unique en passant dans le monde de l'entreprise privée. Ils créèrent d'abord un simple partenariat avant de fonder, en décembre 1947, l'Eckert-Mauchly Computer Corporation. Au-delà du calcul scientifique, ils envisageaient des applications pour la comptabilité, la gestion documentaire et l'analyse statistique.

L'UNIVAC évolua constamment durant sa phase de développement. Conçu initialement comme un outil statistique pour le Bureau du Recensement, le projet s'élargit suite aux discussions avec A.C. Nielsen Company et la compagnie d'assurance Prudential. Ces nouveaux besoins orientèrent la machine vers le traitement de données alphanumériques, indispensables pour la gestion des polices d'assurance, la facturation et les calculs actuariels.

Les soucis d'argent n'épargnèrent pas le développement de cette machine. En 1948, American Totalisator Company injecta 500 000 dollars dans l'entreprise contre 40% des actions. Malgré cette bouffée d'oxygène, les difficultés s'accumulèrent. La situation devint si critique en 1950 qu'Eckert et Mauchly durent se résoudre à vendre leur société à Remington Rand, qui reprit le flambeau du développement et de la commercialisation.

Sur le plan technique, l'UNIVAC I brillait par ses innovations. Sa mémoire à mercure stockait 1 000 mots de 12 caractères chacun. Utilisées pour l'entrée et la sortie des données, les bandes magnétiques traitaient 20 000 caractères par seconde. La machine réalisait une addition en 525 microsecondes et une multiplication en 2 150 microsecondes. Sa fiabilité remarquable reposait sur un système d'auto-vérification : près du tiers des circuits avait pour unique fonction de détecter les erreurs.

La conception modulaire de l'UNIVAC I constituait l'une de ses forces.

Les tâches ne nécessitant pas directement l'unité centrale étaient déléguées à des modules auxiliaires autonomes, chacun disposant de sa propre alimentation. Cette architecture permettait de réaliser en parallèle la saisie sur bande magnétique, la conversion des cartes perforées, et l'impression, sans ralentir le traitement principal.

Le Bureau du Recensement reçut le premier exemplaire en mars 1951, après une batterie de tests draconiens. Ces épreuves, élaborées conjointement par le Bureau des Standards et le Bureau du Recensement, évaluaient autant les capacités de calcul que les performances du système d'entrée-sortie. L'ordinateur devait notamment prouver sa capacité à exécuter 2 500 opérations en 1,26 seconde et à manipuler correctement 1,4 million de caractères sur bande magnétique.

L'arrivée de l'UNIVAC I transforma l'informatisation des organisations. Le Bureau du Recensement l'utilisa pour ses rapports mensuels sur l'économie et le recensement des entreprises. General Electric, à Louisville, s'en servit pour la paie, le suivi des matériaux et l'analyse commerciale. Franklin Life Insurance Company l'adopta pour gérer la facturation des primes, calculer les commissions et tenir la comptabilité des dividendes.

L'US Air Force illustra bien le potentiel de cette machine. Son « Air Materiel Command » gérait un stock pharaonique de 1,2 million d'articles, environ six fois l'inventaire d'un grand magasin. Grâce à l'UNIVAC, l'établissement du budget prévisionnel des pièces détachées pour l'aviation ne prenait plus qu'une journée, contre plusieurs semaines auparavant avec les systèmes à cartes perforées.

La maintenance de ce monstre technologique exigeait une discipline de fer. Le Bureau du Recensement faisait tourner son système 24 heures sur 24, sept jours sur sept, hormis quatre créneaux de huit heures dédiés à la maintenance préventive. Cette dernière mobilisait environ 21% du temps disponible. Les tubes électroniques causaient les pannes les plus fréquentes : en moyenne, deux tubes rendaient l'âme chaque semaine et devaient être remplacés.

La formation des opérateurs s'avéra déterminante pour tirer le meilleur parti du système. Remington Rand comprit très vite qu'un UNIVAC I nécessitait un superviseur du calibre de celui gérant une grande installation de cartes perforées, mais avec une équipe réduite d'opérateurs hautement qualifiés. Cette expertise humaine permettait tant de prendre les bonnes décisions opérationnelles que de repérer précocement les dysfonctionnements.

L'expérience acquise avec le premier UNIVAC I engendra des améliorations continues. Les ingénieurs identifièrent et corrigèrent progressivement divers problèmes : oscillations parasites dans certains circuits, rebonds dans les relais des Uniservo (unités de bande magnétique), et autres soucis méca-

niques. Ces corrections bénéficièrent naturellement aux modèles suivants.

En octobre 1954, Remington Rand avait livré douze UNIVAC I et engrangé des commandes pour quatre systèmes supplémentaires. Ce succès commercial démontra la viabilité du marché des ordinateurs professionnels. L'UNIVAC I fut le premier ordinateur d'une industrie informatique tournée vers les applications commerciales, distincte des calculateurs purement scientifiques qui dominaient jusque-là. Sa réussite donna raison à la vision d'Eckert et Mauchly : celle d'un ordinateur universel capable de servir des applications diverses, du calcul scientifique à la gestion d'entreprise.

Compilateur A-0

Au printemps de l'année 1952, la mathématicienne américaine Grace Murray Hopper crée l'A-0 (Arithmetic Language 0), premier compilateur jamais conçu. Cette création survient à une époque où l'informatique tente de s'extraire des laboratoires militaires pour s'ouvrir à d'autres horizons.

Le chemin vers cette innovation commence durant la Seconde Guerre mondiale. En 1943, cette diplômée de Yale intègre la réserve navale américaine avant d'être affectée à Harvard. Elle y travaille sur le Mark I, alors calculateur numérique le plus imposant, sous la supervision de Howard Aiken. Elle fait partie des trois premiers programmeurs de l'histoire à manipuler cet ancêtre des ordinateurs modernes.

Sur le Mark I, la programmation s'effectue via des rubans perforés : un trou symbolise un 1, l'absence de perforation un 0. Hopper rédige le manuel d'utilisation de l'engin, un *Manuel d'opération pour le calculateur à séquence automatique* de 500 pages. Cette plongée dans les entrailles de la machine lui donne une vision unique des difficultés de la programmation à son niveau le plus élémentaire.

En 1949, elle quitte Harvard pour la Eckert-Mauchly Computer Corporation. Elle participe à la conception de l'UNIVAC I, premier ordinateur commercial électronique. C'est là qu'elle remarque un problème pratique : les programmeurs gaspillent des heures à recopier du code existant pour l'adapter à de nouveaux projets. Cette tâche fastidieuse génère quantité d'erreurs, principalement lors de l'ajustement manuel des adresses mémoire.

Cette observation la pousse en 1951 à créer un système d'automatisation du processus. Elle développe l'A-0, capable d'assembler des sous-routines préexistantes en un programme cohérent. Son système utilise des caractères alphabétiques pour indiquer la nature des opérations (a pour arithmétique,

x pour exponentiel). Les arguments et résultats figurent dans une liste numérotée, référencés par leur position.

La magie de l'A-0 tient à sa capacité de traduction entre langage symbolique et code machine. Pour calculer le carré d'un nombre, un programmeur spécifie simplement l'opération et les références aux variables, sans écrire toutes les instructions binaires correspondantes. Cette abstraction constitue une rupture car pour la première fois, les humains programment dans un langage plus naturel pour eux.

Les gains se révèlent considérables, des semaines de programmation se réduisant à quelques heures. Les erreurs diminuent, la fiabilité augmente. L'A-0 introduit une couche d'abstraction entre l'humain et la machine qui deviendra la pierre angulaire de l'informatique moderne.

Le système évolue avec les versions A-1 puis A-2 en 1953. Cette dernière mouture introduit des mnémoniques courtes pour les opérations, rendant le code plus lisible. Par exemple, `ADD 00X 00Y 00Z` additionne X et Y, stockant le résultat dans Z. L'A-2 est le premier compilateur largement adopté par les utilisateurs d'UNIVAC, validant ainsi le concept.

En 1954, Hopper prend la direction de la programmation automatique pour la division UNIVAC. Cette position lui donne l'opportunité de concrétiser sa vision d'un langage plus accessible. Elle comprend que la majorité des utilisateurs n'apprécient guère les symboles mathématiques et préfèrent quelque chose de plus naturel : l'anglais. Cette réflexion mène au B-0 (Business Language 0) en 1955, rebaptisé FLOW-MATIC, qui autorise la rédaction de programmes avec une syntaxe proche de l'anglais courant.

L'A-0 atteint son apogée dans le développement du COBOL à partir de 1959. Lors de la Conférence sur les Langages de Systèmes de Données (CODASYL), où Hopper est conseillère technique, les fondements établis avec l'A-0 et FLOW-MATIC inspirent directement ce langage. La structure du code en sections distinctes et l'usage d'une syntaxe proche de l'anglais caractérisent principalement du COBOL.

Le compilateur A-0 instaure plusieurs principes structurants pour l'informatique : la réutilisation du code, l'abstraction des opérations machine vers des instructions de haut niveau, et l'accessibilité de la programmation. Ces concepts restent incontournables, au point qu'on peine à imaginer l'informatique contemporaine sans eux.

La contribution de Hopper lui vaut de nombreuses distinctions. En 1969, elle reçoit le *Computer Sciences Man-of-the-Year Award* de la Data Processing Management Association, titre dont l'ironie n'échappe à personne avec notre regard actuel. Elle poursuit sa carrière dans la Marine américaine jusqu'en 1986, atteignant le grade de contre-amiral, tout en défendant l'innovation informatique et la démocratisation de la programmation.

Le compilateur A-0 a métamorphosé un domaine autrefois réservé aux mathématiciens et scientifiques en un outil accessible plus largement. La simplicité d'utilisation d'une part, et l'abstraction des complexités techniques d'autre part, demeurent aujourd'hui au cœur des préoccupations des concepteurs de langages informatiques, héritage direct du travail de Grace Hopper il y a 73 ans.

IBM 701

Au début des années 50, Thomas J. Watson Sr. donna son accord à la proposition de James Birkenstock pour créer une machine de calcul novatrice. Cette dernière utiliserait une mémoire à tubes cathodiques d'environ 20 000 chiffres et disposerait d'un cycle d'horloge capable de multiplier deux nombres en une milliseconde. Baptisée d'abord « Defense Calculator », la machine vit le jour pendant la guerre de Corée à la demande du Département de la Défense américain. Une équipe dirigée par J. A. Haddad et Nathan Rochester fut chargée de son développement.

Le monde ne comptait qu'une vingtaine d'ordinateurs numériques fonctionnels. Birkenstock pensait qu'une trentaine de ces machines trouveraient preneur dans l'industrie de la défense. La création du Defense Calculator fut relativement rapide grâce à l'expérience d'IBM dans ses produits antérieurs. L'équipe adopta une philosophie minimaliste : des équipements simples, un schéma logique épuré et l'absence de dispositifs superflus.

La question de la longueur des mots suscita de nombreux débats chez IBM. Werner Buchholz souligna l'importance de ce choix technique, car une longueur insuffisante obligerait à utiliser plusieurs mots pour un seul nombre. À l'inverse, une longueur excessive ralentirait les opérations et augmenterait les coûts. Après une étude des machines existantes, IBM détermina qu'une longueur entre 10 et 12 chiffres (35 à 41 bits) serait idéale. La technologie des bandes magnétiques suggérait que 6 canaux parallèles représentaient un optimum pour le stockage. IBM fixa donc la taille de mot à 36 bits pour son 701.

Le 29 avril 1952, Thomas J. Watson annonça officiellement l'IBM 701 Electronic Data Processing System lors d'une réunion annuelle. L'appellation « Data Processing » soulignait sa capacité à traiter différents types de données dans des contextes variés. Le premier modèle arriva au siège d'IBM à New York en décembre 1952. Installé dans l'ancienne salle du SSEC, il devint une attraction pour les passants. Cette machine servit aux opérations

de conseil technique pour divers clients, avec une équipe de scientifiques IBM dédiée à la résolution de leurs problèmes.

La deuxième machine (numéro de série 1) fut livrée au Los Alamos Scientific Laboratory le 1er avril 1953. Trois jours suffirent pour la rendre opérationnelle. Elle fonctionna jusqu'à l'automne 1956, avant qu'une seconde machine ne soit installée en 1954. Sa mémoire électrostatique, malgré son manque de fiabilité, contenait 2048 mots de 36 bits. Le système comprenait l'unité centrale, un lecteur de cartes, une imprimante, une unité de stockage sur tambour magnétique, deux lecteurs de bandes magnétiques et une perforatrice de cartes.

Le 701 fut la première machine produite en quantité suffisante pour répondre aux besoins d'une clientèle en expansion. Sa conception modulaire, avec ses composants détachables, marqua le début d'une nouvelle philosophie chez IBM : construire des machines évolutives plutôt que des systèmes monolithiques à remplacer intégralement. La configuration standard comportait onze éléments : deux unités de bande magnétique (chacune avec deux lecteurs), une unité de mémoire sur tambour magnétique, une unité de stockage électrostatique à tube cathodique, une unité arithmétique et de contrôle en forme de L avec panneau opérateur, un lecteur de cartes, une imprimante, une perforatrice de cartes et trois unités d'alimentation.

L'IBM 701 affichait une compacité remarquable. Sa logique tenait dans des modules à 64 broches, chacun portant une rangée de 8 tubes à vide sur la façade. Les opérations logiques s'exécutaient via des diodes au germanium situées à la base des modules. Ces derniers s'intégraient dans un fond de panier conçu pour permettre leur remplacement sans éteindre le système. IBM fut l'un des premiers à utiliser les fonds de panier comme moyen d'accueillir différents types de modules fonctionnels.

L'unité de contrôle analytique (ACU) regroupait dans une grande armoire les composants arithmétiques, les circuits de contrôle d'entrée-sortie et les circuits de contrôle du programme stocké. Une armoire distincte abritait la console opérateur. L'ACU fonctionnait avec une cadence de répétition d'impulsions de 1 mégahertz. Un anneau de 12 étages générait douze impulsions de 1 microseconde formant le cycle machine de base. Cette vitesse répondait principalement aux exigences de régénération de l'unité de stockage électrostatique. Cette dernière contenait 72 tubes cathodiques stockant 10 240 chiffres, soit 2 048 mots de 36 bits, avec un temps d'accès d'environ 12 microsecondes. Les données y étaient accessibles de façon aléatoire. Le coût par bit avoisinait un dollar, bien moins que le stockage à tubes à vide. La location mensuelle de cette unité s'élevait à environ 2 600 dollars.

Le système d'entrées-sorties du 701 assurait cinq fonctions : sélection

d'une unité pour lecture, écriture ou contrôle ; verrouillage de l'opération avec l'exécution d'un programme ; copie des données entre l'unité de stockage électrostatique et les unités d'E/S ; synchronisation des signaux entre l'unité choisie et le CPU ; déconnexion de l'unité d'E/S une fois son travail achevé.

En 1956, l'IBM 737 Magnetic Core Storage Unit vint remplacer l'unité de stockage électrostatique. Dotée d'une capacité de 4 096 mots de 36 bits, elle réduisit le temps d'addition, accès compris, de 60 à 36 microsecondes. Sa location mensuelle atteignait environ 6 100 dollars.

Comme un ordinateur scientifique binaire à adresse unique, l'IBM 701 disposait d'une taille de mot de 36 bits et d'un jeu de 33 instructions. Le CPU du 701 restait vide à 25% pour faciliter la correction d'erreurs ou l'ajout de fonctionnalités. Sa location mensuelle s'élevait à environ 16 000 dollars. Fin 1954, 18 machines étaient installées et pleinement fonctionnelles.

Son successeur, le 704, combla cet espace avec des fonctions supplémentaires. S'ensuivit une lignée de machines toujours plus puissantes et rapides, conservant une compatibilité ascendante et intégrant de nouvelles fonctionnalités. Cette évolution culmina avec le 7094 II et ses versions simplifiées, les 7040 et 7044.

IBM 702

Après le 701, IBM créa le 702, parmi les premiers ordinateurs destinés aux applications de gestion, avec le LEO et l'UNIVAC. Présentée en 1954, cette machine marqua une rupture avec l'ère des équipements mécanographiques à cartes perforées qui dominaient alors le traitement administratif. General Electric figura parmi ses premiers acquéreurs, cette entreprise cherchant à moderniser ses méthodes de gestion dans un contexte social tendu d'après-guerre. L'automatisation des tâches administratives représentait pour GE un levier pour diminuer sa dépendance envers le personnel syndiqué, en plein conflit avec l'organisation United Electrical Workers.

L'IBM 702 se démarquait par sa mémoire électrostatique basée sur des tubes à rayons cathodiques, stockant jusqu'à 10 000 caractères. Cette capacité correspondait à environ 125 cartes perforées, un bond considérable par rapport aux machines antérieures qui ne géraient qu'une ou deux cartes simultanément. Des tambours magnétiques supplémentaires pouvaient étendre cette capacité, chacun équivalant à 750 cartes. La machine

innovait avec son architecture à longueur de mot variable. Cette approche différait des ordinateurs scientifiques contemporains utilisant des mots de taille fixe. L'IBM 702 traitait ainsi des données allant d'un à plusieurs centaines de caractères, une souplesse idéale pour les applications commerciales manipulant des informations comme les noms, adresses et montants. Le traitement numérique s'effectuait en décimal plutôt qu'en binaire, choix révélateur de la volonté d'IBM de simplifier la programmation pour les usages commerciaux. L'ordinateur manipulait aussi bien les lettres que les chiffres, un avantage pour les données d'entreprise. Chaque caractère utilisait un codage sur sept bits, dont un bit de parité pour détecter les erreurs.

Son architecture adoptait un fonctionnement série, traitant les caractères séquentiellement. Cette méthode, bien que théoriquement moins rapide que le traitement parallèle des ordinateurs scientifiques, offrait une grande agilité pour manipuler et réorganiser les données, opérations courantes dans le monde de la gestion.

L'IBM 702 brillait par sa vitesse en entrée-sortie. Tandis que les équipements à cartes perforées classiques traitaient environ 100 cartes par minute, ses bandes magnétiques atteignaient l'équivalent de 3 500 cartes par minute, rendant possible le traitement de volumes de données jusque-là inaccessibles.

La programmation s'exécutait en langage machine. Chaque instruction comportait un code opération d'un caractère suivi d'une adresse de quatre caractères. Les programmes résidaient dans la même mémoire que les données, selon le principe de l'architecture von Neumann. Son jeu d'instructions comprenait des opérations arithmétiques, logiques et de transfert, conçues pour les applications commerciales. Les adresses mémoire étaient absolues, forçant une reprogrammation lors de toute modification dans l'organisation des données. Le programmeur devait tenir compte des délais d'accès à la mémoire électrostatique pour optimiser les performances.

L'installation d'un IBM 702 chez General Electric en 1954 illustre l'informatisation des entreprises à cette période. La première application visée fut un système de paie, projet qui mobilisa une équipe conséquente réunissant personnel de GE, consultants d'Arthur Andersen et ingénieurs d'IBM. En parallèle, une application de contrôle de fabrication fut développée pour le département « Lave-vaisselle et Broyeurs ». Fait notable, cette dernière devint opérationnelle avant le système de paie malgré des ressources plus limitées, préfigurant les observations que Fred Brooks formulerait plus tard dans son essai « The Mythical Man-Month » en 1975 sur la gestion des projets informatiques : l'ajout de personnel à un projet en retard le ralentit davantage, les petites équipes cohérentes surpassent souvent les grandes structures fragmentées, et la complexité de communication croît exponen-

tiellement avec la taille de l'équipe.

La fiabilité constituait un critère primordial. Pour une machine dédiée aux opérations financières, l'IBM 702 intégrait de nombreux mécanismes de contrôle comme, notamment le bit de parité pour repérer les erreurs dans la transmission et le stockage des données. Ces vérifications automatiques s'ajoutaient aux procédures comptables traditionnelles, comme l'utilisation de totaux de contrôle.

L'IBM 702 inaugura une nouvelle page dans l'histoire d'IBM et de l'informatique commerciale. Son succès valida l'intérêt des ordinateurs électroniques pour les applications de gestion, créant un marché qui allait s'imposer. Les concepts introduits, comme le traitement de données à longueur variable et l'orientation vers les applications commerciales, influencèrent durablement l'évolution des systèmes d'entreprise.

La machine eut plusieurs successeurs, notamment l'IBM 705, qui reprit ses principes fondamentaux en y apportant des améliorations techniques. Néanmoins, son véritable héritage se retrouve dans la famille System/360, lancée en 1964, qui unifia les gammes scientifiques et commerciales d'IBM dans une architecture commune. Le System/360 conserva certaines caractéristiques de l'IBM 702, comme l'aptitude à traiter efficacement données numériques et alphabétiques.

L'évolution des ordinateurs transforme des outils de calcul scientifique en machines de traitement polyvalentes. Cette mutation accompagna la modernisation des entreprises américaines durant les années 1950, contribuant à l'émergence de l'informatique de gestion telle que nous la connaissons au XXI^{e} siècle.

Assembleur

Dans les années 1940-1950, les programmeurs étaient contraints de coder en binaire pur. Ce travail fastidieux engendrait maintes erreurs. L'assembleur surgit alors comme une réponse ingénieuse : remplacer les séquences binaires par des mnémoniques plus faciles à retenir. Cette transition marqua le premier pas vers l'abstraction dans l'art de la programmation.

L'assembleur évolue en 1978 avec l'apparition du microprocesseur Intel 8086. Ce composant 16 bits apporta une architecture qui influença durablement le monde de la programmation bas niveau. La saga continua avec les 80186 et 80286 en 1982, qui maintinrent une compatibilité ascendante précieuse pour la continuité des applications existantes.

Le bond vers les 32 bits s'opéra avec le 80386 en 1985, suivi du 80486 quatre ans plus tard. Cette lignée, baptisée x86, établit un standard qui traverse encore les décennies. Le jeu d'instructions s'enrichit sans cesse tout en conservant sa compatibilité historique, gage de pérennité pour les développements logiciels.

Les années 1990 virent le monopole d'Intel s'effriter. AMD lança ses propres processeurs compatibles avec les séries K5, K6 et K7. En 2003, la firme révolutionna le marché avec l'architecture x86-64, extension 64 bits du jeu d'instructions classique. D'autres fabricants comme VIA et Transmeta rejoignirent l'aventure dès 2000. Le jeu d'instructions x86 s'étoffa constamment. Intel déploya diverses extensions : MMX, puis les différentes versions de SSE, tandis qu'AMD ripostait avec 3D Now !. En 2012, ce jeu atteignit plus de 900 instructions, témoignant de la sophistication grandissante de cette architecture.

Les outils modernes tels que `MASM`, `NASM` ou `GNU Assembler` ont transformé le travail du programmeur avec leurs fonctionnalités avancées. Les macros, les outils de débogage et d'optimisation rendent aujourd'hui la programmation en assembleur bien moins rébarbative qu'à ses débuts. Certains domaines valorisent toujours l'assembleur. Les systèmes embarqués, où chaque octet compte, y trouvent un allié de choix. Les créateurs de systèmes d'exploitation l'utilisent pour les sections critiques nécessitant un contrôle total du matériel. Les développeurs de jeux vidéo y font appel pour extraire la quintessence des performances dans les parties gourmandes en calculs.

La segmentation représente une spécificité notable de l'architecture x86. Cette approche divise l'espace mémoire en zones distinctes dédiées au code, aux données et à la pile. Créée initialement pour pallier les limites matérielles des premiers processeurs, elle s'avéra finalement un atout pour sécuriser les systèmes en cloisonnant les accès aux différentes régions mémoire. L'interface entre assembleur et système d'exploitation s'est normalisée au fil du temps. Les appels système, notamment sous DOS via l'interruption 21h, ont donné aux programmes assembleur un accès structuré aux ressources système. Cette normalisation a nettement amélioré la portabilité et simplifié la création d'applications.

Le traitement des nombres à virgule flottante illustre l'évolution de l'assembleur x86. Les premiers processeurs de la gamme nécessitaient un coprocesseur mathématique externe (x87). L'intégration de cette unité directement dans le cœur du processeur, à partir du 486, a simplifié l'écriture du code assembleur tout en décuplant les performances. Les processeurs contemporains transforment en interne les instructions x86 complexes en micro-opérations plus élémentaires. Cette approche marie astucieusement

les bénéfices de l'architecture CISC pour la programmation avec l'efficacité de l'architecture RISC pour l'exécution, préservant ainsi la richesse du jeu d'instructions tout en maximisant les performances.

Bien que souvent oublié, l'enseignement de l'assembleur demeure primordial dans la formation des informaticiens du XXI^{e} siècle. Il offre une compréhension du fonctionnement des ordinateurs et fournit une base solide pour appréhender les concepts de bas niveau. Cette maîtrise s'avère essentielle pour développer des programmes efficaces, notamment dans les secteurs où la performance prime, et peut-être aussi pour réduire la consommation de ressources et d'énergie.

Loin d'être figé, l'assembleur continue d'évoluer. L'émergence d'instructions vectorielles et de registres élargis facilite désormais l'exploitation des capacités des processeurs modernes pour le calcul parallèle et le traitement multimédia. Ces avancées prouvent la remarquable adaptabilité de l'assembleur face aux exigences sans cesse renouvelées de l'informatique.

IBM 650

Le modèle 650 d'IBM, lancé sur le marché en 1954, est la première machine informatique vendue à grande échelle – plus de mille exemplaires – qui ouvrit les portes d'une nouvelle ère numérique. Surnommé « calculateur à tambour magnétique », l'IBM 650 présentait une architecture remarquable pour son temps. Son cœur battait au rythme d'un tambour rotatif nickel-cobalt de 10 centimètres de diamètre et 40 centimètres de longueur. Ce cylindre tournait à la vitesse vertigineuse de 12 500 tours par minute et stockait 2 000 « mots » de dix chiffres. La disposition particulière des données en 40 bandes de 50 positions obligeait les programmeurs à une gymnastique intellectuelle constante pour minimiser les temps d'accès liés à la rotation du tambour.

L'architecture physique du système se divisait en trois blocs distincts : la console centrale abritant le fameux tambour et les circuits arithmétiques, l'unité de lecture-perforation (modèle 533) pour l'entrée et la sortie des données, et l'unité d'alimentation (modèle 655) qui transformait le langage des cartes perforées en signaux électroniques compréhensibles par la machine.

Une particularité technique de l'IBM 650 résidait dans son jeu d'instructions « une plus une adresse ». Chaque instruction contenait l'opération à réaliser ainsi que l'emplacement de l'instruction suivante. Cette approche, parfois décriée par les théoriciens modernes, s'avéra brillante dans le

contexte des contraintes du tambour magnétique. Avec seulement 44 codes opération disponibles, la machine forçait à l'économie et à l'élégance dans l'écriture des programmes. Donald Knuth, qui fit ses premiers pas en programmation sur cette machine durant ses études, garda toute sa vie une tendresse particulière pour cette sobriété qui stimulait l'inventivité.

La machine évolua et le système 653 vint enrichir la configuration initiale avec des registres d'index, une unité de calcul en virgule flottante et 60 mots de mémoire rapide. Des périphériques supplémentaires comme les bandes magnétiques et les imprimantes complétèrent progressivement l'écosystème du 650, étendant ses capacités tout en maintenant la compatibilité avec les programmes existants.

L'aspect logiciel connut un développement tout aussi riche. L'assembleur SOAP (Symbolic Optimal Assembly Program) créé par Stan Poley libéra les programmeurs de la programmation directe en code machine. L'interpréteur des Bell Labs, œuvre de Wolontis, facilita considérablement les calculs scientifiques grâce à la virgule flottante. Une vaste bibliothèque de plus de 200 programmes vit le jour, couvrant des domaines aussi variés que les mathématiques pures, les statistiques, la gestion ou la résolution d'équations différentielles.

À Carnegie Tech (aujourd'hui Carnegie Mellon), Herbert Simon et Allen Newell développèrent sur l'IBM 650 leur langage IPL-V (Information Processing Language), précurseur dans le domaine de l'intelligence artificielle. Ce langage de traitement de listes introduisit des concepts déterminants comme la programmation descendante et les sous-routines fermées – principes qui devinrent plus tard des fondamentaux de la programmation structurée.

La robustesse et l'accessibilité relative de l'IBM 650 le transformèrent en outil pédagogique sans égal. Contrairement à ses contemporains qui exigeaient un personnel hautement spécialisé, cette machine pouvait être utilisée directement par les étudiants et chercheurs. Des milliers de programmeurs firent leurs classes sur ses consoles, formant la première génération massive de professionnels de l'informatique.

Les limitations techniques de la machine – mémoire restreinte et temps d'accès variables au tambour – se révélèrent paradoxalement fécondes. Elles forcèrent les programmeurs à développer des techniques d'optimisation sophistiquées et à réfléchir à la structure de leurs algorithmes. Ces pratiques influencèrent durablement l'enseignement et la conception des programmes informatiques jusqu'à nos jours.

Au-delà des spécifications techniques, l'IBM 650 représente le moment où l'informatique quitta les laboratoires pour entrer dans les bureaux et les universités. Ce n'était plus une curiosité expérimentale mais un outil

pratique, fiable et relativement accessible.

IBM 704

En 1954, IBM lance sur le marché un ordinateur qui changera la donne : l'IBM 704. Premier calculateur produit en série doté d'une unité de calcul en virgule flottante, cette machine s'éloigne radicalement des IBM 701 et 702 qui l'ont précédée.

Qu'apportait de neuf cette machine ? D'abord une mémoire à tores de ferrite remplaçant les tubes Williams utilisés jusqu'alors. L'ajout de trois registres d'index transformait aussi la façon de programmer. Pour tirer parti de ces atouts techniques, les concepteurs ont revu le format des instructions, utilisant désormais le mot complet de 36 bits. Ce jeu d'instructions deviendra l'ADN de toute la lignée scientifique IBM 700/7000 dans les années suivantes.

Sous le capot de l'IBM 704 se cachait un accumulateur de 38 bits, un registre multiplicateur-quotient de 36 bits et trois registres de décrémentation de 15 bits. Ces derniers présentaient la particularité de soustraire leur contenu de l'adresse de base au lieu d'y être ajouté. La machine autorisait l'usage simultané des trois registres dans une instruction, via un champ d'étiquette de 3 bits servant de carte binaire pour indiquer les registres participants. Une astuce technique qui mérite d'être soulignée.

L'architecture de l'IBM 704 reposait sur deux formats d'instruction : le « Type A » et le « Type B ». Le premier combinait un préfixe de trois bits (code instruction), un champ de décrémentation (15 bits), un champ d'étiquette (3 bits) et un champ d'adresse (15 bits). Le second associait un code d'instruction de 12 bits, un champ de drapeaux de 2 bits, quatre bits non utilisés, un champ d'étiquette de 3 bits et un champ d'adresse de 15 bits.

Entre 1955 et 1960, IBM a vendu 123 exemplaires de cette machine. Ses performances étaient d'environ 40 000 instructions par seconde, un nombre qui impressionnait à l'époque. Son prix conséquent la réservait aux grands laboratoires scientifiques et aux groupes industriels.

Mais l'IBM 704 n'a pas seulement brillé par son matériel, il a aussi porté deux créations logicielles importantes. Le FORTRAN, d'abord, créé spécifiquement pour cette machine. Ce langage rendait la programmation scientifique plus naturelle, avec une écriture de code proche des équations mathématiques. Un traducteur automatique convertissait ce code en langage machine, réduisant ainsi les erreurs et le temps de programmation.

LISP, autre enfant prodige né sur l'IBM 704, conserve encore aujourd'hui dans ses primitives `car` et `cdr` la trace de cette filiation. Ces noms, abréviations de « Contents of Address part of Register number » et « Contents of Decrement part of Register number », reflètent directement l'architecture de la machine, où ces champs servaient à stocker respectivement la tête et la queue des listes chaînées.

La musique assistée par ordinateur a fait ses premiers pas sur cette machine. Max Mathews y développa MUSIC, ouvrant un territoire vierge à l'exploration artistique. En 1962, aux Bell Labs, le physicien John Larry Kelly Jr fit chanter l'IBM 704. Son synthétiseur `vocoder` reproduisit « Daisy Bell » avec un accompagnement musical signé Max Mathews. Arthur C. Clarke, témoin émerveillé de cette démonstration, l'immortalisa dans la scène finale de son roman « 2001, l'Odyssée de l'espace », où l'ordinateur HAL 9000 entonne cette mélodie.

L'IBM 704 s'est illustrée dans des domaines scientifiques variés. Ed Thorp l'utilisa pour modéliser ses équations sur les probabilités au blackjack grâce au FORTRAN. La machine participa à la course à l'espace : à l'automne 1957, elle devint le calculateur officiel de l'Operation Moonwatch du Smithsonian Astrophysical Observatory. IBM détacha quatre scientifiques – Giampiero Rossoni, John Greenstadt, Thomas Apple et Richard Hatch – pour aider les chercheurs du Smithsonian à calculer les trajectoires des premiers satellites artificiels.

Le système FORTRAN développé pour l'IBM 704 témoignait d'une maturité logicielle croissante. Son manuel de référence, publié en octobre 1956, décrivait un environnement complet avec gestion des entrées-sorties, conversions automatiques entre formats décimaux et binaires, manipulation de tableaux multidimensionnels et optimisation du code généré. Le système gérait aussi l'intégration de routines externes et l'allocation de la mémoire disponible.

La documentation technique insistait sur l'efficacité du code produit par FORTRAN, comparable à celui écrit par les meilleurs programmeurs en langage machine. Cette performance, couplée à la simplicité de programmation, explique l'adoption rapide du système dans les milieux scientifiques.

L'IBM 704 symbolise cette période où l'ordinateur, d'abord machine à calculer, s'est transformé en outil de création polyvalent. Son influence sur les langages informatiques, la synthèse sonore et le calcul scientifique a marqué les débuts de l'informatique moderne du XX^{e} siècle.

Conférence de Dartmouth

Un document rédigé le 31 août 1955 par quatre scientifiques, John McCarthy, Marvin Minsky, Nathaniel Rochester et Claude Shannon, proposait l'organisation d'une conférence d'été au Dartmouth College. Leur texte, *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*, imaginait un rassemblement de deux mois pour explorer la création de machines douées d'intelligence.

Ces chercheurs défendaient une idée audacieuse : tout aspect de l'apprentissage et de l'intelligence pouvait être décrit avec une précision suffisante pour être reproduit par une machine. Ils voulaient découvrir comment ces machines utiliseraient le langage, formeraient des concepts abstraits, résoudraient des problèmes complexes et s'amélioreraient par elles-mêmes.

La proposition détaillait sept directions de recherche. D'abord, les ordinateurs automatiques : selon eux, la limitation n'était pas tant matérielle que logicielle – nous ne savions pas encore écrire des programmes exploitant pleinement les ressources disponibles. Puis venait la programmation du langage, partant du principe que la pensée humaine manipule essentiellement des mots selon certaines règles. Les réseaux de neurones constituaient le troisième axe, cherchant à comprendre la formation des concepts dans un réseau neuronal. Le quatrième aspect concernait une théorie sur les calculs et leur taille, mesurant l'efficacité des dispositifs informatiques. L'auto-amélioration représentait le cinquième volet, suggérant qu'une machine véritablement intelligente devrait pouvoir accroître ses propres capacités. Le sixième thème abordait les abstractions et leur classification. Le dernier explorait le rôle du hasard et de la créativité, suggérant qu'une dose contrôlée d'aléatoire distinguait la pensée créative de la simple compétence technique.

Chacun des quatre organisateurs apportait une expertise unique. Claude Shannon, mathématicien travaillant aux Bell Labs, avait développé la théorie statistique de l'information et appliqué le calcul propositionnel aux circuits de commutation. Marvin Minsky, jeune chercheur à Harvard spécialisé en mathématiques et neurologie, avait construit une machine simulant l'apprentissage neuronal. Nathaniel Rochester dirigeait la recherche sur l'information chez IBM et avait participé à la conception de l'IBM 701, l'un des premiers ordinateurs commerciaux largement adoptés. John McCarthy, professeur assistant de mathématiques à Dartmouth, étudiait la nature mathématique du processus de pensée.

La Fondation Rockefeller finança l'événement à hauteur de 13 500 dollars, couvrant salaires (1 200 dollars par universitaire), voyages, hébergement et frais administratifs. Les participants venant d'entreprises comme

Bell Labs ou IBM étaient pris en charge par leurs organisations.

Les annexes de la proposition révèlent les visions personnelles des organisateurs. Shannon souhaitait appliquer sa théorie de l'information aux machines calculatrices et aux modèles cérébraux, notamment pour étudier le calcul fiable avec des composants non fiables. Minsky voulait créer des systèmes capables de développer des abstractions sensorielles pour modéliser leur environnement. Rochester s'intéressait à l'originalité dans les performances des machines, cherchant comment introduire une forme maîtrisée de hasard pour favoriser la créativité. McCarthy, quant à lui, explorait la relation entre langage et intelligence, visant à créer un langage artificiel donnant aux machines la capacité de formuler des hypothèses et de s'autoréférencer.

La conférence suscita un vif intérêt. Une liste des chercheurs passionnés par ce que l'on nommait le « problème de l'intelligence artificielle » fut établie. On y trouvait des scientifiques renommés comme John Nash, Herbert Simon ou Warren McCulloch. Cette diversité disciplinaire témoignait de l'attrait du sujet : mathématiques, ingénierie électrique, neurologie, psychologie et informatique s'y rencontraient.

L'expression « intelligence artificielle » fait sa première apparition officielle dans ce document. Ce choix terminologique affirmait clairement l'ambition de créer des machines capables d'une véritable intelligence, bien au-delà de la simple automatisation de tâches spécifiques.

Cette conférence de l'été 1956 établit l'intelligence artificielle comme domaine distinct, avec ses objectifs, méthodes et vision propres. Les questions qu'elle souleva – apprentissage machine, traitement du langage naturel, représentation des connaissances, raisonnement automatique – structurent aujourd'hui la recherche en IA au XXI^{e} siècle.

Son influence, au-delà du cadre technique, créa une communauté scientifique vivante. Les participants retournèrent dans leurs institutions pour y fonder des laboratoires d'IA, former des étudiants et lancer des programmes de recherche qui transformèrent l'informatique durant les décennies suivantes.

L'enthousiasme qui transparaît dans la proposition de Dartmouth caractérise cette époque où l'informatique semblait promise à un avenir sans limites. Certains espoirs se sont heurtés à des obstacles plus complexes que prévu, mais les bases conceptuelles posées lors de cette rencontre ont rendu possibles les applications qui transforment notre quotidien, de la reconnaissance vocale à la traduction automatique, des systèmes experts à l'apprentissage profond.

IBM 350 RAMAC

Le 6 mai 1955, IBM dévoile sans tapage une avancée technique qui changera à jamais notre rapport aux données. Dans un laboratoire de San José, une équipe d'ingénieurs vient de créer le premier disque dur de l'histoire. Cet appareil au nom peu évocateur pour le grand public, l'IBM 350 RAMAC (Random Access Method of Accounting and Control), marque le début d'une nouvelle ère dans le stockage numérique.

Tout commence trois ans plus tôt. En janvier 1952, W. Wallace McDowell contacte Reynold B. Johnson, un ancien professeur reconverti chez IBM. La direction souhaite établir un petit laboratoire sur la côte Ouest américaine, loin des habitudes et structures de l'Est. Johnson reçoit carte blanche pour monter une équipe limitée à 50 personnes. Sa mission consistait à explorer des voies techniques différentes, notamment l'impression sans impact et la réduction des données.

Johnson ne perd pas de temps. En quelques semaines, il loue un bâtiment vide à San José, lance le recrutement et débute les travaux. Louis D. Stevens, envoyé temporairement pour l'aider, sera finalement son bras droit technique. Dès juillet 1952, le laboratoire fonctionne avec une trentaine de collaborateurs. L'atmosphère y est particulière, façonnée par trois règles établies par Johnson : comprendre sa machine et son usage, connaître tous les projets en cours dans le laboratoire, et toujours donner priorité à l'aide entre collègues.

Le projet qui aboutira au RAMAC s'appelle au départ « Source Recording ». Son objectif est simplement de transformer des données alphanumériques en format lisible par machine, alors que les options de stockage se comptent sur les doigts d'une main : cartes perforées, bandes magnétiques et, plus marginalement, tambours magnétiques. Ces supports présentent tous des limitations, rendant les processeurs souvent inactifs pendant la recherche d'information.

À l'automne 1952, l'équipe s'intéresse aux « fichiers bac » de cartes perforées très répandus dans les entreprises. Ces grands casiers rectangulaires contiennent des cartes classées par numéro de client ou d'article. Les employés y fouillent manuellement pour produire factures et bons de livraison, un processus lent et sujet aux erreurs.

Après avoir exploré diverses pistes, Johnson prend en janvier 1953 une décision audacieuse qui suscite l'incrédulité : concentrer tous les efforts sur les disques magnétiques. Un ingénieur lui prédit un échec cuisant, et le projet hérite du surnom moqueur de « trancheuse à saucisson ».

Les problèmes techniques semblent insurmontables. Maintenir un écart microscopique entre la tête de lecture et la surface tournante, créer des surfaces parfaitement planes, développer un revêtement magnétique fiable... La liste paraît sans fin. L'équipe teste différents matériaux : aluminium, laiton, verre, plastiques, magnésium. Les disques en aluminium se déforment à grande vitesse. La solution vient finalement de stratifiés d'aluminium compressés et chauffés au-delà du point de recuit. Le revêtement magnétique pose un autre casse-tête. Jake Hagopian trouve une méthode artisanale : verser le revêtement sur la face interne d'un disque en rotation rapide. Bill Crooks l'améliore en filtrant la solution à travers du nylon et en utilisant... des gobelets en papier pour les mesures. Cette technique rudimentaire sera utilisée pendant un an avant d'être automatisée. Norman Vogel résout quant à lui le problème de la tête de lecture-écriture. Il conçoit un système qui rétracte la tête pendant le mouvement entre les disques, avec trois minipistons pneumatiques pour contrôler la position.

Le 10 février 1954, le premier transfert d'information entre cartes et disques est réussi. Le prototype ressemble à une machine abracadabrante, mais l'équipe persévère. Une refonte complète aboutit au Modèle II, avec comme innovation le passage à un axe vertical pour les disques, facilitant leur remplacement et laissant plus d'espace pour les mécanismes d'accès.

Le 10 janvier 1955, moins de trois ans après la création du laboratoire, le premier Modèle II du RAMAC 350 est présenté avec succès. Le système est commercialisé en septembre 1956, intégré au système IBM 305 RAMAC qui comprend processeur, lecteur de cartes et imprimante.

Les caractéristiques du RAMAC 350 étaient sans précédent. Cinquante disques de 61 cm tournant à 1 200 tr/min, un débit de 100 000 bits par seconde, une capacité totale de 5 millions de caractères. La densité atteignait 2 000 bits par pouce carré, avec un temps d'accès moyen de 600 millisecondes. Pour mesurer le chemin parcouru, l'IBM 3380 des années 1980 utilisait neuf disques de 35,5 cm tournant à 3 600 tr/min, avec un débit de 24 millions de bits par seconde et une capacité de 1,25 milliard d'octets. Le coût de location mensuel par mégaoctet chuta de 130 dollars pour le RAMAC 350 à environ un dollar pour le 3380.

L'arrivée du RAMAC 350 a bouleversé notre rapport aux données. Pour la première fois, l'information devenait directement accessible sans manipulation physique préalable. Cette rupture technologique a rendu possibles les applications informatiques interactives : réservations aériennes, gestion des stocks, services bancaires automatisés, contrôle des vols spatiaux, traitement de texte...

Cette innovation, née dans un modeste laboratoire californien, a lancé une industrie du stockage générant des milliards de dollars, et stimulé le

développement des processeurs rapides et des services informatiques. En 1984, l'American Society of Mechanical Engineers reconnaît officiellement le RAMAC 350 comme jalon historique international, saluant son rôle dans l'évolution du génie mécanique et son influence sur la société du XX^{e} siècle.

Le RAMAC marque la naissance véritable de l'ère numérique où nous vivons aujourd'hui, où l'accès instantané à l'information est devenu si naturel que nous en oublions parfois le miracle technologique qu'il représente.

IPL

Les ordinateurs, machines d'abord créées pour résoudre des équations mathématiques, allaient bientôt s'attaquer à un domaine différent : celui du traitement symbolique et de la résolution de problèmes complexes. C'est dans cette effervescence qu'a émergé Information Processing Language, famille de langages qui a tracé un sillon dans l'histoire de l'intelligence artificielle.

L'aventure d'IPL débute avec le rêve audacieux de construire un programme capable de prouver des théorèmes logiques. IPL-I, esquissé au début des années 1950, n'a jamais existé ailleurs que sur le papier. Cette ébauche constituait néanmoins une première tentative pour formaliser les mécanismes de manipulation symbolique indispensables à la démonstration automatique.

À la RAND Corporation se développèrent ensuite IPL-II et IPL-III, déployés sur l'ordinateur JOHNNIAC. Ces versions ont donné vie au programme Logic Theorist, une œuvre pionnière dans l'univers de l'intelligence artificielle. Ce programme démontrait des théorèmes par des méthodes heuristiques calquées sur la pensée humaine, rompant avec l'approche purement calculatoire des ordinateurs.

La migration d'IPL vers l'IBM 650 par une équipe du Carnegie Institute of Technology donna naissance à IPL-V, version achevée et influente de cette lignée. Pendant ce temps, IPL-IV tournait sur le JOHNNIAC pour des applications novatrices : un programme d'échecs et un système d'équilibrage de chaînes de production industrielle.

IPL-V marquait une rupture dans la conception des langages informatiques. Il apportait des notions comme la manipulation de listes et de structures de données évolutives. Le langage rendait possible la création et la modification dynamique des données durant l'exécution du programme, caractéristique rare à une époque où la mémoire restait généralement figée.

La grande force d'IPL résidait dans sa capacité à modéliser des tâches que les humains accomplissent sans pouvoir en décrire le cheminement. Jouer aux échecs, prendre des décisions commerciales ou démontrer des théorèmes mathématiques sont autant d'activités où l'expertise humaine repose sur des intuitions plutôt que sur des algorithmes définis. IPL fournissait un cadre pour transcrire ces comportements adaptatifs typiques de l'intelligence naturelle.

L'architecture d'IPL s'articulait autour de quelques principes fondamentaux. Le système employait des cellules pour stocker des mots IPL, formés de deux préfixes (`P` et `Q`) et de deux symboles (`SYMB` et `LINK`). Les symboles, régionaux ou locaux, offraient une représentation souple des données. Le langage a popularisé le concept de « liste », structure fondamentale pour organiser l'information de façon hiérarchique.

Une création remarquable d'IPL fut son système de « liste de description ». Ce mécanisme associait des informations descriptives à une liste, informations modifiables en cours d'exécution. Cette approche annonçait les concepts actuels de métadonnées et d'attributs d'objets que l'on retrouve dans la programmation moderne du XXI^{e} siècle.

IPL intégrait aussi des fonctionnalités avancées de gestion mémoire, notamment avec sa « liste d'espace disponible » qui automatisait l'allocation et la libération des ressources. Cette méthode affranchissait les développeurs des contraintes de gestion manuelle, leur laissant davantage d'énergie pour la logique algorithmique.

Le langage proposait environ 150 processus fondamentaux couvrant un large spectre d'opérations : manipulation de listes, calculs arithmétiques, gestion des entrées-sorties, et traitement de structures complexes. Un dispositif élaboré de « générateurs » rendait possible l'exécution d'opérations répétitives avec élégance.

IPL a servi de socle à diverses applications dans le champ de l'intelligence artificielle naissante. Des programmes d'apprentissage de discrimination, de choix binaires et de démonstration de théorèmes ont vu le jour grâce à ce langage. Ces créations ont prouvé l'aptitude d'IPL à soutenir des applications sophistiquées de traitement symbolique.

IPL-VI, ultime version de cette famille avec une vision ambitieuse, proposait une architecture matérielle conçue spécifiquement pour exécuter directement ce langage, sans passer par une couche d'interprétation logicielle. Cette démarche visait à dépasser les limites de performance inhérentes à la nature interprétée des versions antérieures sur des machines conventionnelles.

IPL a inspiré la création d'autres systèmes comme LISP, le traitement de listes en FORTRAN, ou COMIT. Ces langages ont prolongé l'explora-

tion des voies ouvertes par IPL dans le domaine du traitement symbolique et de l'intelligence artificielle. Malgré ses faiblesses en termes de rapidité, notamment pour les calculs numériques traditionnels, il a montré que les bénéfices d'un système de haut niveau compensaient largement ses coûts en ressources, tout particulièrement pour des applications complexes nécessitant une gestion fine des données et des processus.

TX-0

Le TX-0 (Transistorized Computer Zero) vit le jour au Lincoln Laboratory du MIT en 1956. L'histoire de cette machine exceptionnelle débute par la volonté de créer le premier ordinateur à usage général entièrement transistorisé. Les deux ambitions qui guidaient cette aventure étaient d'expérimenter l'utilisation des transistors comme éléments logiques dans un calculateur numérique rapide, et de mettre à l'épreuve une mémoire à tores de ferrite d'une capacité extraordinaire de 65 536 mots.

L'équipe dirigée par Kenneth Olsen fit le choix d'une architecture minimaliste : un mot de 18 bits, dont 16 bits d'adressage et seulement 2 bits pour le code instruction. Ce jeu d'instructions minimal ne proposait que quatre commandes. La première stockait l'accumulateur, la deuxième y additionnait une valeur, la troisième gérait les transferts conditionnels sur valeur négative. La quatrième, véritable tour de force, autorisait la microprogrammation, avec plusieurs transferts de registres au sein d'un cycle machine. Cette astuce rendit le TX-0 remarquablement souple malgré sa conception épurée.

L'apport technologique du TX-0 résidait dans ses composants. Les transistors Philco L-5122, commercialisés plus tard sous la référence 2N240, subirent une sélection draconienne avant leur montage. Le résultat dépassa toutes les attentes : après 10 000 heures de fonctionnement, ces semiconducteurs ne montraient qu'une infime baisse de gain. Plus stupéfiant, en 1974, après 49 000 heures d'opérations, à peine une douzaine avaient rendu l'âme.

L'architecture électronique du TX-0 reposait sur des circuits RC, association d'une résistance et d'un condensateur, couplés en logique négative, avec -3V pour représenter le 1 et 0V pour le 0. La vraie prouesse résidait dans l'usage exclusif de transistors pour réaliser les fonctions logiques, sans recourir aux diodes usuelles. Les opérations `AND` et `OR` s'obtenaient par manipulation subtile des tensions.

La première mémoire du TX-0, baptisée « S Memory », constituait un monument technologique : 1,25 million de minuscules tores de ferrite, fabriqués artisanalement au Lincoln Laboratory. Cette matrice stockait 65 536 mots de 19 bits avec un temps de cycle de 5 microsecondes. En 1958, cette mémoire fut transférée vers le projet TX-2, cédant la place à une nouvelle mémoire transistorisée de 4 096 mots.

L'année 1958, le TX-0 quitta son berceau pour rejoindre le campus du MIT à Cambridge. Ce déménagement transforma son rôle d'objet de recherche en outil expérimental, accessible jour et nuit aux chercheurs et étudiants. Cette politique de libre accès engendra un foisonnement d'applications novatrices.

Les évolutions matérielles se succédèrent. La mémoire passa à 8 192 mots en 1959, tandis que le jeu d'instructions s'étoffait progressivement. L'ajout d'un registre d'index et l'extension du code opération à 5 bits ouvrirent de nouvelles possibilités. Le système d'entrée-sortie s'enrichit de convertisseurs analogique-numérique et numérique-analogique, préparant le terrain aux applications temps réel.

Le logiciel suivit cette mutation. L'utilitaire initial de Jack Gilmore céda la place à des assembleurs sophistiqués. L'assembleur Macro, œuvre de J.B. Dennis en 1959, apporta des fonctions avancées comme les macro-instructions et la gestion automatisée des constantes. Son successeur, Midas, permit la manipulation de symboles plus longs et introduisit la macro-programmation récursive.

Les applications du TX-0 illustrent l'étendue de son influence. Le laboratoire de biophysique des communications l'adopta pour analyser les données électrophysiologiques du cortex auditif. L'équipe de recherche sur la parole s'en servit pour créer les premières techniques de reconnaissance vocale. D'autres chercheurs explorèrent le traitement d'images et la reconnaissance de caractères, ainsi qu'une main robotique munie de capteurs grâce à cet ordinateur.

Kenneth Olsen, fort de cette expérience du TX-0, fonda Digital Equipment Corporation (DEC). Les concepts développés sur le TX-0 inspirèrent directement le PDP-1, premier ordinateur commercial de DEC, qui reprit nombre des caractéristiques de son ancêtre académique.

Le TX-0 démontra la pertinence d'une approche interactive et personnelle de l'informatique. Son mode de fonctionnement en accès direct bouscula le modèle dominant du traitement par lots. Sa longévité témoigne de l'excellence de sa conception. Après plus de 49 000 heures d'activité, la machine nécessitait toujours très peu d'entretien. En 1975, le TX-0 trouva refuge au Computer Museum de DEC à Marlborough, Massachusetts, préservant pour la postérité un jalon fondateur de l'informatique transistori-

sée.

FORTRAN

IBM lança le projet FORTRAN durant l'été 1954, acronyme de FORmula TRANslating, sous la houlette de John Backus. À cette époque, programmer le nouvel ordinateur d'IBM, le 704, relève du parcours du combattant. Près des deux tiers du budget des projets scientifiques s'envolent dans la simple préparation des programmes. Pire, la planification, l'écriture et le débogage engloutissent plus de 90% du temps total des projets.

Face à ce constat, l'équipe de Backus nourrit l'ambition claire de créer un langage qui se rapproche de l'écriture mathématique tout en générant automatiquement un code machine performant. L'objectif audacieux est de diminuer d'au moins 80% le temps consacré au codage et au débogage.

Le développement du système s'étale sur deux ans et demi. Une équipe de 18 personnes y travaille sans relâche. Leurs efforts aboutissent en 1957 à un système complet comprenant deux éléments fondamentaux : le langage FORTRAN et son traducteur, qu'on nomme *executive routine*.

FORTRAN bouscule les habitudes avec ses innovations. Les expressions arithmétiques s'écrivent désormais naturellement, proches de la notation mathématique classique. L'astérisque (`*`) symbolise la multiplication tandis que le double astérisque (`**`) représente l'exponentiation. Le système génère un code machine optimisé, évitant les calculs redondants et sélectionnant les meilleures instructions pour chaque opération.

Le langage regorge de fonctionnalités inédites. Les instructions `DO` et `IF` contrôlent le flux d'exécution. Les tableaux multidimensionnels se manipulent aisément grâce aux variables indicées. Les instructions `READ`, `PRINT` et `FORMAT` gèrent les entrées-sorties avec souplesse. Les programmeurs définissent leurs propres fonctions, favorisant la réutilisation du code et structurant mieux leurs programmes.

La prouesse technique du traducteur FORTRAN réside dans sa capacité à produire un code machine presque aussi efficace que celui écrit manuellement par des programmeurs chevronnés. Cette performance découle d'une analyse fine du code source et de techniques d'optimisation novatrices. L'architecture du traducteur se divise en six sections distinctes, chacune remplissant une fonction précise dans la chaîne de traduction.

Un exemple concret illustre l'efficacité du système. Après une journée de formation et quelques consultations du manuel, un programmeur réalise une

application en quatre heures seulement, avec 47 instructions FORTRAN. La compilation sur le 704 prend six minutes et génère environ 1000 instructions machine. Le programmeur repère et corrige une erreur dans son code, sans recourir à des techniques de débogage complexes. Ce programme lui aurait demandé trois jours de codage manuel, sans compter le temps de débogage.

Le succès ne se fait pas attendre. FORTRAN s'impose comme la référence en programmation scientifique. Après sa première normalisation en 1966, le langage est largement adopté, facile à enseigner et relativement indépendant des machines. Il a démontré les avantages des sous-programmes et de la compilation indépendante.

Les années 1970 voient poindre de nouveaux langages qui révèlent certaines limites de FORTRAN. Une nouvelle mouture, Fortran 77, est normalisée en 1978. Malgré des améliorations bienvenues, cette mise à jour ne comble pas toutes les attentes d'une communauté avide de changements plus profonds. D'autres langages comme Pascal, Ada, Modula 2, C et C++ commencent à gagner du terrain dans les cercles scientifiques et académiques.

Il faut attendre 1991 pour voir émerger Fortran 90, une évolution majeure. Cette version introduit une pléthore de fonctionnalités modernes : forme source libre autorisant des noms de variables plus longs, structures de contrôle revues, spécification précise de la précision numérique, traitement des tableaux comme entités à part entière, allocation dynamique de mémoire avec pointeurs et récursivité, types de données définis par l'utilisateur, modules avec surcharge d'opérateurs et les procédures génériques.

Fortran 95, finalisé en 1996, apporte des améliorations plus modestes mais significatives comme l'instruction `FORALL` pour le traitement parallèle, les sous-programmes purs et élémentaires définis par l'utilisateur, ainsi que diverses optimisations concernant les pointeurs et les types dérivés.

Un bond en avant s'opère avec Fortran 2003, qui intègre la programmation orientée objet. Cette version comprend l'héritage des types dérivés, le polymorphisme permettant aux variables de changer de type durant l'exécution, les procédures liées aux types, et l'interopérabilité avec le langage C. Les capacités d'entrée-sortie se voient renforcées, et le support des exceptions arithmétiques IEEE fait son apparition.

La dernière évolution, Fortran 2008, enrichit encore le langage avec le support de la programmation parallèle via les tableaux `coarray`, l'augmentation du rang maximal des tableaux à quinze dimensions, de nouvelles fonctionnalités pour améliorer les performances comme la construction `DO CONCURRENT`, et l'ajout de nombreuses fonctions mathématiques intrinsèques pour le calcul scientifique.

D'un langage pionnier conçu pour simplifier la programmation des cal-

culateurs scientifiques, FORTRAN s'est métamorphosé en un outil moderne qui intègre les paradigmes de programmation contemporains tout en restant fidèle à sa vocation première : offrir aux scientifiques et aux ingénieurs un moyen efficace d'exprimer leurs calculs. Sa longévité tient à sa capacité d'évoluer tout en maintenant la compatibilité avec les versions antérieures, préservant ainsi les investissements colossaux réalisés dans le code existant. Le code Fortran 77 demeure un sous-ensemble valide des versions récentes, ce qui rend possible la modernisation progressive des applications des organisations sans tout reconstruire.

Cette robustesse hors norme trouve une illustration saisissante dans les sondes spatiales Voyager. Plus de 45 ans après leur lancement, à 24 milliards de kilomètres de la Terre, elles fonctionnent toujours avec leur code FORTRAN d'origine. En 2023, les équipes de la NASA réalisent un exploit technique en réussissant à déployer des correctifs sur ce code, témoignant de l'extraordinaire pérennité du langage dans des conditions extrêmes.

Circuit intégré

À la fin de la Seconde Guerre mondiale, l'électronique complexe des bombardiers B-29 avait un problème technique de taille : chaque appareil embarquait près d'un millier de tubes à vide et des dizaines de milliers de composants passifs. Cette multiplication des éléments entraînait une hausse exponentielle des coûts de fabrication et une baisse inquiétante de la fiabilité. Les militaires américains, confrontés à cette réalité technique, craignaient que cette complexité ne freine drastiquement l'évolution des systèmes électroniques.

Le National Bureau of Standards lança les premières recherches pour simplifier la fabrication. La division Centralab de Globe-Union proposa une technique novatrice à base de substrats céramiques supportant des interconnexions métalliques et des condensateurs, sur lesquels venaient se fixer des tubes à vide miniaturisés. Cette méthode, fruit du travail de S.J. Rubenstein, Mr Ehlers, Gertrude B. Sherwood et Howard J. White, marqua la première tentative de création de composants *in situ*.

Après 1945, les travaux continuèrent sous la direction de R.L. Wolff et A.S. Khouri, et Centralab mit au point des procédés de sérigraphie haute cadence. La société fabriqua des amplificateurs pour appareils auditifs intégrant des dizaines de composants passifs et quelques supports de tube. L'utilisation d'un substrat à forte constante diélectrique permit la produc-

tion économique de réseaux RC destinés aux postes radio et aux téléviseurs.

Tout changea avec l'arrivée du transistor en 1948. Plus compact, moins énergivore et plus fiable que le tube à vide, il générait aussi moins de chaleur. Son intégration aux circuits existants demanda toutefois une longue adaptation technologique. Jack Kilby, chez Texas Instruments, et Robert Noyce, de Fairchild Semiconductor, conçurent indépendamment le circuit intégré en 1958.

Kilby imagina que tous les composants d'un circuit pouvaient être fabriqués dans un bloc de semi-conducteur. En septembre 1958, sa démonstration d'un oscillateur à déphasage utilisant transistors, résistances et condensateurs en germanium prouva la validité du concept. Cette approche pionnière utilisait des interconnexions par fils d'or soudés qui s'avéraient incompatibles avec une production industrielle, ce qui constituait un défaut important.

Noyce développa une solution différente chez Fairchild, basée sur la technologie planaire de Jean Hoerni. Sa conception utilisait le silicium comme substrat, recouvert d'une couche d'oxyde isolante sur laquelle étaient déposées des pistes d'aluminium pour les interconnexions. Cette méthode, plus adaptée à l'industrialisation, devint le fondement de la technologie moderne des circuits intégrés.

Ces inventions transformèrent radicalement l'électronique. La fabrication de circuits complets sur une seule puce de silicium réduisit drastiquement les coûts tout en améliorant spectaculairement la fiabilité. En 1961, Fairchild commercialisa la famille Micrologic, premier ensemble de circuits logiques compatibles avec résistances diffusées et interconnexions. Cette année-là, Texas Instruments livra à l'US Air Force un petit ordinateur intégrant plusieurs centaines de bits de mémoire semi-conductrice.

L'industrie des circuits intégrés suivit une trajectoire exponentielle, décrite par Gordon Moore en 1965 : le nombre de transistors par circuit double environ tous les deux ans. Cette prédiction s'est vérifiée pendant plus de cinquante ans. Des premiers circuits contenant quelques milliers de transistors, nous sommes passés à des puces renfermant plusieurs milliards de composants.

Les progrès technologiques ont constamment réduit la taille des transistors : de plusieurs microns dans les années 1970 à quelques nanomètres aujourd'hui. Cette miniaturisation s'accompagne d'une hausse des performances et d'une baisse de la consommation électrique. La technologie actuelle approche cependant ses limites physiques, avec des transistors ne mesurant que quelques dizaines d'atomes de large.

Les circuits intégrés ont rendu possible l'ordinateur personnel, le téléphone mobile, Internet et l'intelligence artificielle. L'essor de l'industrie des

semi-conducteurs a bouleversé l'économie mondiale, créant de nouveaux secteurs et modifiant nos modes de vie.

Face aux contraintes physiques de la miniaturisation, les chercheurs explorent désormais l'intégration tridimensionnelle ou l'informatique quantique. Les besoins de puissance de calcul pour l'intelligence artificielle et le *machine learning* du XXI^{e} siècle stimulent encore l'innovation dans ce domaine.

ALGOL

La naissance d'ALGOL (ALGOrithmic Language) remonte à la fin des années 1950. Ce langage résulte d'une vision partagée par des chercheurs européens et américains : créer un outil universel pour exprimer des algorithmes indépendamment des machines. Lors d'un symposium à Darmstadt en 1955, cette idée prend racine dans les discussions entre spécialistes du calcul automatique.

Le contexte technologique de l'époque justifie pleinement la démarche. Aux États-Unis, l'informatique se transforme en secteur industriel avec IBM et d'autres fabricants qui commercialisent leurs premiers ordinateurs. Une multitude de langages spécifiques fleurit au gré des besoins et des équipes. L'Europe, quant à elle, voit naître ses premières grandes machines dédiées principalement aux calculs scientifiques.

La GAMM (Gesellschaft für Angewandte Mathematik und Mechanik) et l'ACM (Association for Computing Machinery) décident en 1957 de former des groupes de travail sur ce langage commun. La rencontre de Zürich en 1958 aboutit à la fusion des propositions européennes et américaines sous le nom d'IAL (International Algebraic Language), rebaptisé ALGOL 58.

En 1959, John Backus fait une avancée en inventant une notation formelle pour décrire la syntaxe des langages de programmation. Cette forme Backus-Naur (BNF) remplace les descriptions imprécises en langage naturel par une définition rigoureuse. Peter Naur perfectionne cette notation et l'intègre à la définition d'ALGOL 60, posant ainsi les bases d'une méthode désormais classique.

ALGOL 60 se distingue par plusieurs innovations : les blocs de code avec leur propre portée lexicale, les procédures récursives, les paramètres transmis par nom ou par valeur. Sa rigueur mathématique transparaît dans l'obligation de déclarer les types de données. Les structures de contrôle comme la boucle `for` atteignent un niveau d'élégance et de clarté inédit.

Cette nouvelle approche stimule la recherche sur les techniques de compilation. Friedrich Bauer et Klaus Samelson publient leur méthode de traduction séquentielle des formules en 1960, tandis qu'Edsger Dijkstra réussit l'exploit de mettre au point un compilateur ALGOL 60 en seulement dix mois.

La théorie informatique s'enrichit grâce à ALGOL. Seymour Ginsburg et Gordon Rice établissent en 1962 un lien entre les langages définis par la BNF et les travaux linguistiques de Chomsky sur les grammaires contextuelles libres. Ce rapprochement entre informatique et linguistique favorise l'émergence d'une informatique théorique.

Le rayonnement d'ALGOL atteint le monde académique. Les revues scientifiques l'adoptent pour la publication d'algorithmes, notamment dans les « Algorithms » du *Communications of the ACM* dès 1960. Les universités européennes en font leur langue d'enseignement privilégiée.

Malgré ces succès, ALGOL peine à s'imposer face à FORTRAN dans le monde commercial. Ce paradoxe s'explique par l'absence d'un acteur industriel majeur pour le soutenir, le manque de bibliothèques standard pour les entrées-sorties, et l'inertie liée aux programmes FORTRAN existants. L'Europe, moins dépendante de cet héritage, l'adopte plus largement, notamment en Allemagne où le gouvernement l'intègre aux systèmes universitaires.

À partir de 1962, la responsabilité du développement d'ALGOL revient au Working Group 2.1 de l'IFIP (International Federation for Information Processing). Les travaux sur un successeur aboutissent à ALGOL 68, langage plus complet mais aussi plus complexe. Sa définition suscite de vives controverses au sein du groupe, au point qu'en 1969, près de la moitié des membres publient un « Minority Report » exprimant leur désaccord.

ALGOL a inspiré Pascal, Simula (précurseur de la programmation orientée objet), et Ada. Ses concepts fondamentaux – blocs structurés, portée lexicale, récursivité – deviennent des références incontournables. La notation BNF reste aujourd'hui la méthode standard pour définir la syntaxe des langages informatiques.

ALGOL témoigne de la maturation de l'informatique comme discipline scientifique. Il transforme la programmation d'une pratique artisanale en une activité formalisée. Sa vision mathématique a marqué la pensée informatique du XX^{e} siècle. Si son succès commercial fut limité, son influence intellectuelle a façonné des générations de langages et de programmeurs.

Lisp

En 1956, le projet Lisp émerge dans le cadre des recherches estivales de Dartmouth sur l'intelligence artificielle. John McCarthy cherche alors un langage algébrique pour traiter des listes sur l'IBM 704. Il s'intéresse aux travaux de Newell, Shaw et Simon sur IPL 2, un langage conçu pour le JOHNNIAC de la RAND Corporation, mais refuse d'en reproduire la forme trop dépendante de son environnement d'origine. L'approche algébrique du Fortran l'attire davantage.

Deux circonstances favorisent ce développement : la création d'un centre de calcul IBM au MIT, accessible depuis Dartmouth, et le lancement par IBM d'un programme de démonstration de théorèmes géométriques basé sur une idée de Marvin Minsky, pour lequel McCarthy travaille en tant que consultant. IBM semble s'engager vers la recherche naissante en IA.

En 1958, McCarthy propose le concept d'Advice Taker, un système devant représenter les informations par des phrases dans un langage formel et prendre des décisions par déductions logiques. La représentation des phrases par structures de listes s'impose naturellement, tout comme l'usage d'un langage de traitement de listes pour programmer les opérations déductives.

Le premier défi technique concerne la mise en œuvre des structures de listes sur l'IBM 704. Cet ordinateur utilise des mots de 36 bits, avec deux sections de 15 bits nommées adresse et décrément. L'adressage limité à 15 bits rend nécessaire l'utilisation de pointeurs de taille identique pour les structures de listes. McCarthy établit initialement des fonctions dont `cwr` (Contents of Word in Register) et quatre fonctions d'extraction. Il note que l'extraction d'une sous-expression exige la composition de l'extraction de l'adresse avec `cwr`, tandis que la progression dans la liste demande la composition de l'extraction du décrément avec `cwr`. Ces observations donnent naissance aux fonctions `car` (Contents of Address Register) et `cdr`, qui deviendront emblématiques du langage.

Durant les années 1957 et 1958, McCarthy conçoit les expressions conditionnelles en programmant des routines d'échecs en Fortran. Les limites de l'instruction `IF` de Fortran le conduisent à créer une fonction `XIF(M,N1,N2)` retournant N1 ou N2 selon la nullité de M. Cette solution simplifie les programmes mais souffre de l'évaluation systématique des trois arguments. Il invente l'expression conditionnelle authentique, n'évaluant que les expressions nécessaires.

À l'été 1958, invité par Nathaniel Rochester chez IBM, McCarthy travaille sur la différentiation d'expressions algébriques. Ce travail introduit plusieurs avancées : l'écriture de fonctions récursives utilisant des expressions conditionnelles, la fonction `maplist` formant une liste d'applications

d'un argument fonctionnel aux éléments d'une liste, et l'usage de la notation λ de Church pour nommer les fonctions. Sa définition récursive de la différentiation ne prévoit pas l'effacement des structures abandonnées, car il refuse d'altérer l'élégance de sa définition.

À l'automne 1958, McCarthy évolue en tant que professeur assistant en sciences de la communication au MIT et lance avec Minsky le projet d'intelligence artificielle du MIT. Le développement de Lisp démarre avec l'objectif initial de créer un compilateur. L'équipe commence par compiler manuellement diverses fonctions en assembleur et écrire des sous-programmes d'environnement Lisp pour lire et écrire les structures de listes.

Les programmes s'écrivent d'abord dans une notation informelle appelée M-expressions, conçue pour ressembler au Fortran. Cette notation utilise des crochets pour les arguments des fonctions, réservant les parenthèses aux constantes de structure de liste. La notation M ne sera jamais complètement formalisée, car la représentation des fonctions Lisp par des listes s'impose comme approche dominante après l'apparition de l'interpréteur.

Les fonctions `READ` et `PRINT` établissent une notation externe standard pour l'information symbolique. Par exemple, l'expression $x+3y+z$ devient `PLUS X (TIMES 3 Y) Z`. Cette notation sera nommée Cambridge Polish, en référence à la notation préfixée de Lukasiewicz et à son usage antérieur par Quine.

La gestion mémoire soulève un problème particulier. L'effacement explicite, utilisé dans IPL, manque d'élégance. Deux options se présentent : effacer l'ancien contenu d'une variable lors de sa mise à jour, ce qui nécessite des compteurs de références impossibles à intégrer dans les six bits disponibles, ou utiliser le ramasse-miettes, qui abandonne le stockage jusqu'à épuisement de la liste libre, marque le stockage accessible depuis les variables et la pile, puis transforme le stockage non marqué en nouvelle liste libre.

Les versions suivantes de Lisp apportent diverses fonctionnalités : listes de propriétés associées aux atomes, insertion et suppression d'éléments dans les listes (via `rplaca` et `rplacd`), application efficace des nombres comme atomes, et le *program feature* pour écrire des programmes séquentiels avec instructions d'affectation et sauts.

Deux raisons expliquent la longévité de Lisp. D'une part, son noyau occupe une forme d'optimum local dans l'espace des langages de programmation, avec l'utilisation récursive des expressions conditionnelles, la représentation de l'information symbolique par listes, et la représentation uniforme des programmes et données. D'autre part, Lisp offre des caractéristiques opérationnelles uniques qui en font un véhicule privilégié pour les systèmes de calcul symbolique et d'intelligence artificielle : son système d'exécution

donnant accès aux fonctionnalités de la machine hôte, sa représentation interne facilitant la compilation depuis des langages de plus haut niveau, et la disponibilité de son interpréteur comme langage de commande.

DEC PDP-1

En 1960 arriva le PDP-1 (Programmed Data Processor), premier ordinateur commercial de Digital Equipment Corporation. Cette machine rompt avec les monstres informatiques de l'époque par sa philosophie radicalement différente de vouloir rendre l'ordinateur accessible et interactif.

Kenneth Olsen et Harlan Anderson, fondateurs de DEC en 1957, ont conçu cet ordinateur en puisant leur inspiration du TX-2 du MIT Lincoln Laboratory, où ils avaient précédemment travaillé. Le TX-2 leur sert de modèle, mais ils en simplifient l'architecture tout en gardant sa dimension interactive.

L'un des traits distinctifs du PDP-1 réside dans ses performances techniques avancées. Sa mémoire à tores de ferrite contient 4 096 mots de 18 bits, avec un cycle de 5 microsecondes. Il traite 100 000 additions par seconde, surpassant de 2,5 fois les grands calculateurs contemporains grâce à son architecture parallèle et ses circuits à 5 MHz. La substitution des tubes à vide par des transistors Micro-Alloy et Micro-Alloy-Diffused diminue son encombrement à seulement 17 mètres carrés.

Son originalité tient surtout dans l'interaction directe qu'il propose avec l'utilisateur. L'écran cathodique de 16 pouces, le clavier Friden Flexowriter et le lecteur de bande perforée lisant 300 lignes par seconde créent une expérience d'utilisation immédiate, alors que le mode *batch* dominait largement les usages informatiques.

D'un point de vue technique, le PDP-1 apporte plusieurs innovations durables. Son jeu d'instructions comprend 32 opcodes avec des opérations arithmétiques, logiques et de contrôle. Son système d'interruption *sequence break* gère plusieurs périphériques simultanément. L'adressage indirect multiple apporte une flexibilité inédite dans le traitement des données. Ces caractéristiques marqueront les futures générations d'ordinateurs. Ce modèle utilise le complément à un pour son arithmétique, choix abandonné dans les versions suivantes au profit du complément à deux, illustrant sa nature expérimentale.

Vendu 120 000 dollars, la commercialisation du PDP-1 montre l'audace de DEC et défie les ordinateurs IBM. Cette machine ne nécessite ni climati-

sation ni faux plancher, fonctionne sur une simple alimentation de 110 volts et intègre des circuits d'autodiagnostic pour sa maintenance. DEC cherche ainsi à « démocratiser » l'accès à l'informatique.

L'environnement logiciel comprend un assembleur avec support des macros, un éditeur de texte appelé Expensive Typewriter, l'un des premiers interpréteurs Lisp, et divers utilitaires. Le système accepte plusieurs périphériques additionnels tels qu'un lecteur de cartes perforées, une imprimante rapide, des bandes magnétiques IBM, ou des convertisseurs analogique-numérique.

En 1962, des étudiants du MIT créent *Spacewar !* sur cette machine. Ce programme, considéré comme le premier jeu vidéo sur ordinateur, illustre les capacités graphiques et interactives du PDP-1 tout en inventant une nouvelle forme de divertissement.

Malgré la production d'une cinquantaine d'exemplaires entre 1960 et 1969, l'influence du PDP-1 établit DEC comme acteur majeur de l'informatique interactive. Son architecture inspire directement les futurs modèles de la série : PDP-4, PDP-7, PDP-9 et PDP-15.

Aujourd'hui, le Computer History Museum de Mountain View conserve trois PDP-1, dont un fonctionnait encore en 1995. Ces machines témoignent d'une période où l'informatique commençait à quitter les centres de calcul pour se rapprocher des utilisateurs. Contrairement à IBM qui privilégiait les systèmes centralisés, DEC misait sur l'interaction homme-machine et l'accessibilité. Le succès du PDP-1 valide la stratégie de DEC basée sur des machines moins onéreuses et plus accessibles. Le PDP-1 marque ainsi les premiers pas d'une informatique où miniaturisation et interactivité sont primordiales.

IBM 1401

L'IBM 1401, annoncé en 1959, est le premier ordinateur largement adopté par les entreprises à travers le monde. Il marque la transition entre l'ère des machines mécanographiques à cartes perforées et celle de l'informatique moderne. À cette époque, le traitement des données dans les entreprises – inventaire, facturation, comptes clients, paie – s'effectuait en faisant passer des paquets de cartes perforées dans des machines électromécaniques qui triaient, calculaient, intercalaient et totalisaient. Ces opérations étaient déterminées par des tableaux de connexion câblés manuellement. Les ordinateurs centraux à tubes offraient certes une programmation stockée flexible

et un stockage sur bande magnétique, mais leur coût les rendait inaccessibles pour la plupart des entreprises. Au milieu des années 1960, la moitié des ordinateurs en service dans le monde étaient des IBM 1401 ou des membres de sa famille.

C'est en 1957 que l'histoire du 1401 commence, lorsqu'IBM France et IBM Allemagne développent un prototype de « machine comptable universelle » transistorisée (WWAM) en réponse à un concurrent européen. Malgré l'utilisation d'un tableau de connexion, la configuration d'entrée de gamme s'avère trop coûteuse. À Endicott, New York, un architecte système audacieux, Fran Underwood, découvre qu'il peut remplacer économiquement le tableau de connexion par un contrôle par programme stocké. Conservant les chemins de données du WWAM, il conçoit un jeu d'instructions adapté aux applications commerciales et nomme le nouvel ordinateur SPACE (Stored-Program Accounting & Calculating Equipment).

Le développement du SPACE est dirigé par Charles Branscomb, qui étend l'équipe à 60 ingénieurs travaillant 24 heures sur 24. Il intègre une imprimante en cours de développement, décourage les fonctionnalités superflues et atteint les objectifs de coût pour l'entrée de gamme. À la mi-1959, avec un prototype fonctionnel et la formation des vendeurs en cours, la machine, officiellement renommée IBM 1401, s'apprête à révolutionner le marché professionnel.

Le 5 octobre 1959, IBM annonce le 1401 par télévision en circuit fermé à 50 000 participants dans 104 lieux différents. En cinq semaines seulement, 5 200 systèmes sont commandés, dépassant les prévisions de ventes sur toute la durée de vie du produit et surpassant le nombre total d'ordinateurs commerciaux alors en service dans le monde. Un an plus tard, le premier système est livré à Time-Life, qui transfère 60 millions de cartes perforées sur quelques centaines de bandes magnétiques. Les systèmes 1401 à bandes libèrent ainsi les utilisateurs d'une pratique vieille de sept décennies : le stockage des données sur cartes perforées.

En 1962, les revenus du 1401 dépassent ceux des machines comptables traditionnelles. En 1965, le 1401 et sa famille (modèles 1410, 1440, 1460, et 7010) représentent la moitié de tous les ordinateurs dans le monde, atteignant un pic de 16 000 systèmes en 1967. La majorité des 1401 étaient loués. La location d'un système d'entrée de gamme coûtait 2 500 dollars par mois, soit l'équivalent de trois machines comptables et une calculatrice. Un système typique se louait 6 500 dollars par mois, ou s'achetait 500 000 dollars – environ un dixième du coût d'un ordinateur central.

Considéré comme un ordinateur « petit système », un 1401 pesait entre 2 et 3 tonnes, consommait jusqu'à 14 000 watts et contenait environ un demi-million de pièces. Entretenus principalement par les ingénieurs clients

d'IBM, ces systèmes étaient réputés pour leur fiabilité et fonctionnaient souvent 24 heures sur 24.

Sur le plan technique, le 1401 reposait sur le traitement série des chiffres et des caractères. Pendant son cycle d'horloge de 11,5 microsecondes, une instruction pouvait additionner deux chiffres décimaux ou traiter un caractère en entrée, en sortie, ou le formater pour l'impression. Les nombres et les chaînes de longueur variable étaient stockés à raison d'un chiffre ou d'un caractère par position mémoire. La taille de la mémoire variait de 1 400 à 16 000 positions, chacune sur 8 bits – 6 bits pour le caractère, 1 bit pour marquer la fin du nombre ou de la chaîne, 1 bit de parité.

Le processeur du 1401 fonctionnait à 87 000 cycles par seconde. Pour aider à la comparaison, un PC actuel à 3 gigahertz permet d'additionner deux nombres de 20 chiffres environ un million de fois plus vite qu'un 1401. La machine utilisait des transistors germanium à jonction alliée et des diodes à pointe de contact peu coûteux, montés sur environ 4 000 cartes SMS (Standard Modular System). La mémoire à tores de ferrite, constituée d'un empilement de grilles de minuscules anneaux magnétiques traversés par des fils fins, coûtait environ 60 cents par bit – 400 millions de fois plus cher que la RAM actuelle.

Le succès du 1401 tenait en grande partie à ses périphériques : l'imprimante à chaîne 1403 dont la vitesse (600 lignes/minute, saut rapide) et l'exceptionnelle qualité d'impression (chaîne interchangeable à rotation horizontale) en firent un standard industriel ; le lecteur-perforateur de cartes 1402 (800 et 250 cartes/minute) ; et les bandes magnétiques 729 de type mainframe (15 millions de caractères par bobine). Le disque RAMAC 1405 (20 millions de caractères) et un lecteur optique de chèques bancaires furent disponibles dès 1961.

Côté logiciel, IBM proposait des assembleurs pour la programmation en langage machine (Autocoder, SPS), des compilateurs pour les langages Fortran, COBOL et RPG (Report Program Generator), ainsi que des utilitaires de tri et d'entrée-sortie. IBM ne développa pas de système d'exploitation pour le 1401, mais en créa un pour le 1410.

Fin 1964, Honeywell conteste la domination d'IBM sur le marché en annonçant le H200, plus rapide et plus performant, accompagné de son logiciel *Liberator* capable d'exécuter les programmes du 1401 sans modification. En 1961, IBM commence à planifier l'unification de ses nombreuses gammes d'ordinateurs. Le System/360, annoncé en avril 1964, consolide logiciels, périphériques et support dans une famille d'ordinateurs compatibles. Une technologie clé était le *control store* flexible qui permettait à son modèle d'entrée de gamme, le Modèle 30, d'exécuter de manière compétitive les programmes du 1401, contrant ainsi le H200. L'émulation du 1401 permit

aussi à IBM de livrer des Modèles 30 quand le logiciel système du S/360 prit du retard.

Si le développement de la famille 1401 ralentit après l'annonce du S/360, ses membres restèrent plus nombreux que les S/360 jusqu'en 1968. IBM continua à proposer le 1401 jusqu'en 1971, et des émulateurs sur PC sont disponibles aujourd'hui.

Le 1401 laisse aussi un héritage inattendu dans les domaines artistiques. La machine pouvait produire de la musique en activant mélodiquement les marteaux d'impression du 1403 ou en modulant ses émissions d'ondes radio captées par un poste AM. Inspiré par un enregistrement de musique radio réalisé lors d'une cérémonie de mise hors service d'un 1401 en 1971, un compositeur islandais écrivit en 2001 une orchestration en 60 morceaux intitulée *IBM 1401 : A User's Manual* (disponible sur Spotify, Apple Music etc.). Enfin, dans le film de 1964 « Dr Folamour », l'acteur Peter Sellers fait sa première apparition devant un 1403.

L'IBM 1401 continue de passionner les férus d'informatique. Des bénévoles du Computer History Museum, dont une douzaine d'anciens ingénieurs clients d'IBM, ont consacré plus de 40 000 heures à la restauration de deux 1401, l'un acquis en 2003 dans un garage allemand et l'autre en 2008 dans une maison du Connecticut (où il avait fonctionné jusqu'en 1995).

COBOL

En mai 1959, le Pentagone accueille à Washington une réunion qui changera l'informatique d'entreprise pour les décennies à venir. Charles A. Phillips, à la tête du Data Systems Research Staff, rassemble une cinquantaine de personnes : militaires, fonctionnaires, consultants et ingénieurs des plus grands fabricants d'ordinateurs. Chaque fabricant de l'époque a son propre langage, ses propres machines. Un programme écrit pour un IBM ne fonctionne pas sur un UNIVAC. Comment mutualiser les efforts dans un secteur où les applications administratives commencent à proliférer ?

De cette rencontre naît le CODASYL, un comité chargé de réfléchir aux langages informatiques. Un groupe plus restreint, le « comité court terme », reçoit la mission d'analyser les points forts et les faiblesses des compilateurs existants, principalement AIMACO (sa variante militaire), FLOW-MATIC (l'œuvre de Grace Hopper chez Remington-Rand), et COMTRAN (un projet d'IBM très théorique).

Personne ne s'attendait à ce que ce comité accouche d'un nouveau lan-

gage. Pourtant, en novembre 1959, six personnes s'enferment pendant deux semaines et en sortent avec les spécifications d'un langage qu'ils nomment COBOL : Common Business Oriented Language. Ce document, finalisé en avril 1960, allait bien au-delà d'une simple synthèse des langages existants.

Le choix technique du COBOL reposait sur sa structure en quatre divisions : `IDENTIFICATION` pour les informations du programme, `ENVIRONMENT` pour décrire l'environnement d'exécution, `DATA` pour structurer les données, et `PROCEDURE` pour les instructions. La syntaxe se rapprochait de phrases en anglais, rendant le code lisible y compris pour des non-spécialistes. Les noms de variables pouvaient compter jusqu'à 30 caractères, une nouveauté qui permettait d'utiliser des noms clairs et compréhensibles plutôt que des abréviations illisibles.

Le moment de vérité survint les 6 et 7 décembre 1960 lors d'une démonstration publique : un même programme COBOL s'exécuta correctement sur deux machines radicalement différentes, un RCA 501 et un UNIVAC II. Pour la première fois, la portabilité du code informatique devenait réalité.

Les premières années de COBOL furent marquées par plusieurs évolutions : COBOL-61, COBOL-61 Extended, puis COBOL-65. La normalisation ANSI arriva en 1968, suivie par d'autres normes en 1974 et 1985. La compatibilité ascendante, garantissant que les anciens programmes continueraient à fonctionner, est une caractéristique remarquable qui fut maintenue.

Le gouvernement américain avait l'exigence que tout ordinateur vendu ou loué aux administrations devait disposer d'un compilateur COBOL, et ce fut un des facteurs de succès de COBOL. Cette décision força les constructeurs à développer leurs propres compilateurs, propulsant COBOL au rang de standard incontournable dans l'informatique de gestion.

On pourrait questionner la longévité extraordinaire de ce langage né comme une solution temporaire, que plusieurs éléments expliquent. D'abord sa relative simplicité le rendait accessible aux programmeurs non académiques qui formaient le gros des équipes informatiques des entreprises. Sa lisibilité facilitait la maintenance des applications. Son orientation vers le traitement des données administratives correspondait parfaitement aux besoins du secteur tertiaire.

Face au bug de l'an 2000, le monde découvrit avec stupeur l'omniprésence de COBOL dans les rouages de l'économie. Les estimations vertigineuses montrèrent que sur les 300 milliards de lignes de code en production mondiale en 1997, environ 240 milliards étaient écrites en COBOL. Plus de 95% des données financières et d'assurance transitaient par ces programmes.

Contre toute logique apparente, COBOL traversa le cap de l'an 2000 sans perdre son importance. En 1999, plus de 50% des nouvelles applications

critiques étaient encore développées dans ce langage, face à l'émergence de Java. Les projections pour 2004-2005 estimaient que 15% des nouvelles applications (5 milliards de lignes) seraient en COBOL, tandis que 80% des applications déployées constitueraient des extensions à des programmes COBOL existants.

Cette résistance inattendue trouve sa source dans les caractéristiques techniques du langage : pas de pointeurs, des types de données élémentaires, une description détaillée des fichiers et des sorties d'impression. Ces limitations, qui pourraient sembler handicapantes pour des applications scientifiques, sont en réalité des atouts pour les applications de gestion en réduisant les risques d'erreurs catastrophiques.

Dans notre monde obsédé par l'innovation perpétuelle, parfois la simplicité et la stabilité l'emportent sur la sophistication. Un langage jugé dépassé par les informaticiens depuis les années 1970 continue de faire tourner les systèmes financiers mondiaux au XXI^{e} siècle, démontrant que l'élégance théorique n'est pas toujours gage de pertinence pratique.

Apprentissage automatique

Le jeu de dames comme laboratoire d'intelligence. Voilà comment Arthur Samuel conçut son approche. Ce pionnier, actif des années 1940 jusqu'aux années 1960, transforma l'informatique en lui insufflant l'apprentissage de l'expérience, une capacité jusqu'alors réservée aux humains.

Le choix des dames n'était nullement fortuit. Dans le paysage de l'intelligence artificielle, les jeux avaient un rôle comparable à celui de la drosophile pour les généticiens, des organismes simples, faciles à manipuler, parfaits pour l'expérimentation. Les dames, moins complexes que les échecs, offraient un terrain idéal où Samuel put observer les mécanismes d'apprentissage sans s'embourber dans des règles trop sophistiquées.

En 1952, le premier programme vit le jour sur l'IBM 701. Trois ans plus tard, la version dotée d'apprentissage autonome captiva le public lors d'une démonstration télévisée. Sans atteindre le niveau des champions, ce programme innovait par deux méthodes : l'apprentissage par cœur (« rote learning ») et l'apprentissage par généralisation.

La première technique, l'apprentissage par cœur, ressemblait davantage à notre propre mémoire qu'on ne pourrait l'imaginer. Samuel programma son ordinateur pour qu'il se souvienne de chaque configuration rencontrée durant ses parties. Concrètement, quand l'ordinateur analysait une

position, il calculait sa valeur grâce à un algorithme appelé minimisation-maximisation qui simule les meilleurs coups possibles pour chaque joueur. La machine stockait ensuite cette position avec son évaluation. Lors des parties suivantes, face à une situation déjà vue, plus besoin de longs calculs, la réponse surgissait instantanément de sa mémoire. Samuel intégra un mécanisme ingénieux qu'il nomma « dépréciation ». Pour comprendre, visualisons un arbre où chaque branche représente un coup possible. Plus on s'enfonce dans cet arbre, plus on avance dans le déroulement hypothétique de la partie. Samuel fit en sorte que les positions lointaines dans cet arbre valent légèrement moins que celles proches du moment présent, même à valeur de jeu égale. Cette subtilité encourageait le programme à privilégier les chemins courts vers la victoire. Sans cette astuce, l'ordinateur aurait pu s'enliser dans des stratégies interminables, préférant gagner en 50 coups plutôt qu'en 10, simplement parce que les deux scénarios aboutissaient mathématiquement à la victoire.

Sa seconde méthode, l'apprentissage par généralisation, préfigurait les techniques modernes d'apprentissage par différences temporelles. Samuel fit jouer son programme contre lui-même des milliers de fois. À chaque coup, par une idée brillante et intuitive, l'ordinateur modifiait subtilement sa façon d'évaluer le plateau. Si une position du jeu mène à d'autres positions avantageuses, alors cette position initiale doit être favorable. Imaginons un coup qui semble banal mais conduit systématiquement à des configurations gagnantes trois mouvements plus tard, le programme apprend graduellement à valoriser ce coup apparemment anodin. L'ordinateur n'attendait pas la fin de la partie pour ajuster ses évaluations, il apprenait en continu, après chaque mouvement. Sans qu'on lui dise explicitement « ceci est bon » ou « cela est mauvais », le programme découvrait par lui-même les stratégies efficaces. Sa boussole restait néanmoins l'avantage matériel – le nombre de pièces possédées par rapport à l'adversaire – avec une préférence marquée pour les dames, ces pièces couronnées aux capacités de déplacement supérieures. Cette approche, étonnamment moderne, constitue l'une des premières manifestations d'apprentissage autonome en informatique au XX^{e} siècle.

Au-delà du cadre de la recherche théorique, les travaux de Samuel sur le calcul non numérique façonnèrent l'architecture des premiers ordinateurs IBM. Les instructions logiques qu'il proposa devinrent un standard industriel tant leur utilité s'imposa pour tout traitement non mathématique. Avant de bouleverser l'intelligence artificielle, il s'était distingué comme ingénieur électricien. Diplômé d'Emporia College en 1923, puis titulaire d'un master du MIT en 1926, il y enseigna brièvement avant de rejoindre les Bell Labs. Ses recherches sur les tubes électroniques, notamment sur la charge d'espace entre électrodes parallèles, marquèrent l'époque. Durant la guerre,

ses travaux sur les dispositifs protégeant les récepteurs radar s'avérèrent primordiaux.

En 1946, devenu professeur à l'Université de l'Illinois, il participa à la création d'un des premiers ordinateurs électroniques. C'est là qu'il rêva d'un programme de dames capable de vaincre un champion du monde. Ce projet resta inachevé jusqu'à son arrivée chez IBM en 1949, où il travailla sur le 701, premier ordinateur à programme enregistré de la firme. Samuel améliora considérablement sa mémoire à tubes Williams, quadruplant sa capacité de stockage et stabilisant son fonctionnement. La première démonstration publique de son programme impressionna Thomas J. Watson Sr., fondateur d'IBM, au point que celui-ci prédit – correctement – une hausse de 15 points de l'action en bourse.

Samuel contribua à l'expansion internationale d'IBM, façonnant l'orientation des laboratoires européens, notamment à Vienne (informatique) et Zurich (physique). Sa discrétion explique sans doute pourquoi l'importance de ses travaux ne fut pleinement reconnue qu'après son départ d'IBM en 1966.

À Stanford, où il devint professeur de recherche, il poursuivit ses travaux sur le jeu de dames jusqu'à ce que d'autres programmes surpassent le sien dans les années 1970. Il s'intéressa à la reconnaissance vocale et dirigea de nombreux doctorants. Ses talents s'étendaient à la rédaction technique. Il excellait à comprendre des documentations confuses et à produire des manuels limpides.

Samuel continua à programmer jusqu'à un âge remarquable. Sa dernière contribution, à 85 ans, fut d'adapter des programmes d'impression multipolices pour le département d'informatique de Stanford. Seule la maladie de Parkinson mit fin à cette activité exceptionnelle. Sa dernière connexion date du 2 février 1990, ce qui fait probablement de lui le plus âgé des programmeurs actifs de son temps.

Schéma de Backus-Naur

Les premiers langages de haut niveau font leur apparition dans les années 1950 et avec eux surgit la problématique de la description claire et sans ambiguïté de leur syntaxe. À cette question, John Backus apporta une réponse qui allait transformer notre manière de concevoir les langages informatiques.

Avant, la description des langages relevait presque de l'art divinatoire.

Les informaticiens s'évertuaient à expliquer en prose les règles syntaxiques, produisant des textes souvent confus. Le Fortran, création de Backus chez IBM, n'échappait pas à cette difficulté. Sa documentation regorgeait d'explications alambiquées du type « Un nombre réel est une séquence quelconque de chiffres décimaux avec un point décimal précédant ou intervenant entre deux chiffres ou suivant une séquence de chiffres, le tout pouvant être précédé d'un signe plus ou moins ». Pas vraiment limpide.

L'année 1959, lors du congrès IFIP à Zurich, Backus présenta une notation formelle pour décrire la syntaxe d'ALGOL. Sa proposition, qu'il avait élaborée sans prétention théorique particulière (il avoua plus tard sa méconnaissance relative de la logique mathématique), aurait pu passer inaperçue. Mais ce fut Peter Naur qui saisit immédiatement la portée de cette idée. Il retoucha la notation et l'appliqua à la définition d'ALGOL 60. Cette collaboration donna naissance au nom *Backus-Naur Form* (BNF), une appellation proposée par Donald Knuth pour remplacer le terme initial « Backus Normal Form » qu'il jugeait inadéquat sur le plan mathématique.

Le génie du BNF tient à sa capacité d'exprimer la récursivité avec une simplicité désarmante. Pour définir un nombre entier par exemple, écrire `<integer> ::= <digit> | <integer><digit>` indique qu'un entier est soit un chiffre unique, soit un entier suivi d'un chiffre. En une ligne, nous voilà capables de décrire des nombres de n'importe quelle longueur.

Backus fit un autre choix déterminant en s'affranchissant des contraintes matérielles. Contrairement au Fortran, qui limitait par exemple les nombres à l'intervalle $[10^{-38}, 10^{38}]$ pour des raisons de mémoire, le BNF se concentre sur la syntaxe pure, sans se soucier des limitations physiques des ordinateurs. Cette rupture conceptuelle a donné aux langages une dimension théorique indépendante de leur mise en œuvre technique.

Le succès du BNF ne tarda pas. Sa concision séduisit immédiatement : quelques lignes remplaçaient des pages entières d'explications. Sa précision balaya les ambiguïtés du langage naturel. Son aspect opérationnel acheva de convaincre : les règles BNF permettaient de construire directement des analyseurs syntaxiques automatiques.

ALGOL 60 devint ainsi le premier langage entièrement défini par une notation formelle. Cette standardisation donna aux développeurs sur différentes machines la possibilité de créer des compilateurs compatibles, concrétisant l'ambition universelle d'ALGOL. La façon de concevoir les langages informatiques s'en trouva bouleversée. Le BNF incita les informaticiens à adopter une démarche plus rigoureuse et systématique. Les étudiants purent saisir plus clairement la structure des langages de programmation grâce à ce formalisme pédagogique.

Au fil du temps, des extensions vinrent enrichir la notation originale. La

EBNF (*Extended BNF*) intégra des opérateurs pour exprimer plus simplement la répétition ou le caractère optionnel d'éléments. Des représentations graphiques apparurent, mais les principes fondamentaux posés par Backus et Naur demeurèrent intacts.

Le BNF illustre l'apport des notations formelles à l'informatique. Il a transformé une approche artisanale en méthode scientifique, et ce, sous l'impulsion de praticiens confrontés à des problèmes concrets. La séparation qu'il opère entre syntaxe et mise en œuvre est devenue un principe cardinal dans la conception des langages.

Plus de soixante ans après son introduction, au XXI^{e} siècle, cette notation demeure fondamentale. Elle sert aujourd'hui à définir la syntaxe des langages de programmation, les formats de données, les protocoles de communication et diverses notations formelles. Le BNF est ainsi l'un de ces outils discrets mais puissants qui transforment toute une discipline sans faire la une des journaux.

V

1960

L'aube d'une nouvelle ère technologique

La décennie 1960 a transformé notre rapport aux machines. Entre deux blocs qui se toisaient dans une guerre jamais déclarée, l'informatique s'est frayé un chemin qui dépassait les simples enjeux militaires. Le choc provoqué par Spoutnik en 1957 résonnait encore dans les couloirs de Washington. La création de la DARPA en 1958 fut la réponse américaine : un laboratoire d'idées qui allait bouleverser notre rapport à la technologie.

L'Amérique investit alors des sommes colossales dans la recherche. Le MIT, Stanford, Berkeley, ces noms devinrent synonymes d'innovation. Dans leurs laboratoires, des jeunes gens en chemise blanche et cravate étroite imaginaient l'avenir. Le gouvernement américain, inquiet de la montée en puissance soviétique, ne comptait plus ses dollars pour garder l'avantage technologique.

Les entreprises commençaient à s'équiper de ces machines énormes aux prix vertigineux. Les banques, les assurances et les industries lourdes cherchaient à dompter le flot grandissant de données qui submergeait leurs services comptables. Ces premiers ordinateurs, véritables cathédrales électroniques, transformèrent l'organisation du travail. Les cartes perforées remplaçaient progressivement les registres manuscrits.

AT&T modernisait son réseau avec la commutation électronique. Ce changement technique, peu spectaculaire pour le grand public, constituait

la première brique des futurs réseaux informatiques, un édifice colossal. Les ingénieurs des Bell Labs menaient les premières expériences de transmission numérique.

La jeunesse des années 60 regardait le monde différemment. Les campus universitaires bouillonnaient d'idées nouvelles. Pour ces étudiants, l'ordinateur ne se réduisait pas à un outil militaire ou comptable. Ils y voyaient un moyen d'expression, un outil de liberté. Cette vision alternative de l'informatique germait dans l'esprit des futurs pionniers de la micro-informatique.

La course à l'espace stimulait l'innovation. Comment envoyer un homme sur la Lune sans calculateurs performants ? La NASA exigeait des ordinateurs fiables, capables de résister aux conditions extrêmes. L'informatique embarquée naquit de cette contrainte. Plus tard, ces avancées furent adaptées aux besoins civils. Le programme Apollo accoucha ainsi d'innovations qui dépassèrent par la suite le cadre spatial.

L'Europe, meurtrie par la guerre, tentait de suivre le mouvement. En France, Charles de Gaulle lança le Plan Calcul en 1966, rêvant d'indépendance technologique. La Compagnie Internationale pour l'Informatique devait incarner cette ambition française. Le Royaume-Uni capitalisait sur l'héritage d'Alan Turing et des calculateurs de Bletchley Park. L'Allemagne, tout à sa reconstruction industrielle, automatisait ses usines.

À l'autre bout du monde, le Japon s'éveillait. Fujitsu et NEC, ces noms inconnus en Occident, développaient leurs premiers ordinateurs. Le pays du Soleil-Levant initiait sa future domination dans l'électronique.

En 1964, IBM changea les règles du jeu avec son System/360. Cette famille d'ordinateurs compatibles entre eux créa un standard de fait. Fini le temps où chaque machine parlait un langage différent. Cette standardisation accéléra l'adoption de l'informatique dans les entreprises moyennes. Le marché du logiciel trouva là un terreau fertile pour se développer.

Le MIT bouillonnait d'idées. Le Compatible Time-Sharing System permettait à plusieurs personnes d'utiliser simultanément le même ordinateur, concept qui annonçait l'informatique interactive. À Xerox PARC, des chercheurs inventaient les interfaces graphiques qui rendraient plus tard l'ordinateur accessible à tous.

Ces transformations technologiques inquiétaient les syndicats qui redoutaient la disparition massive d'emplois. Ces questions se posaient déjà, si actuelles avec l'intelligence artificielle. La société découvrait qu'une révolution technologique entraîne toujours des bouleversements sociaux.

L'ordinateur fascinait le grand public. La télévision montrait ces machines mystérieuses aux lumières clignotantes. L'ordinateur de *2001, l'Odyssée de l'espace* de Stanley Kubrick, HAL 9000, incarnait à la fois la promesse et la menace de l'intelligence artificielle. L'informatique entrait dans

l'imaginaire collectif.

La fin des années 60 vit naître une préoccupation nouvelle : la protection des données personnelles. L'informatisation des administrations soulevait des questions inédites sur la vie privée. Les premiers textes juridiques encadrant ces enjeux commencèrent à apparaître.

Les années 1960 ont donc vu l'ordinateur quitter les laboratoires pour entrer dans le monde réel. Cette décennie a défini les contours de notre relation aux machines. Les choix techniques, économiques et culturels de cette période continuent d'influencer notre présent numérique. Dans les universités américaines des années 60 s'écrivait l'histoire du XXI^{e} siècle. Les ordinateurs pesaient plusieurs tonnes et coûtaient l'équivalent de 4 000 000 euros actuels. De nos jours, chacun porte dans sa poche une machine infiniment plus puissante.

Quicksort

En 1961, Charles Antony Richard Hoare ne se doutait pas, qu'il venait de concevoir l'un des algorithmes les plus durables de l'informatique. Son Quicksort naît dans un contexte particulier où les ordinateurs traitent des volumes de données croissants et les méthodes de tri traditionnelles montrent leurs limites. Alors jeune chercheur britannique, il propose une approche radicalement différente.

L'idée paraît simple au premier regard. On choisit un élément du tableau, le « pivot », puis on réorganise les autres éléments autour de lui : les plus petits d'un côté, les plus grands de l'autre. On répète ensuite l'opération sur chaque moitié jusqu'à obtenir un tableau entièrement trié. Cette méthode de « diviser pour régner » révolutionne l'approche du tri en informatique.

Sa publication de 1962 dans le *Computer Journal* présente les premiers résultats formels. Il y démontre que son algorithme surpasse nettement les méthodes existantes. Mais l'histoire de Quicksort ne fait que commencer.

Dès 1965, R.S. Scowen repère une faiblesse dans le choix aléatoire du pivot. Il développe Quickersort, qui sélectionne plutôt l'élément médian du tableau. Cette modification change la donne pour les tableaux partiellement triés, qui posaient problème à la version originale de Hoare.

R.C. Singleton franchit une nouvelle étape en 1969 avec sa méthode *median of three*. Au lieu de se contenter d'un seul élément, il examine trois valeurs : le premier élément, celui du milieu et le dernier. Il choisit ensuite la

médiane de ces trois comme pivot. Cette astuce améliore les performances moyennes d'environ 5%, un gain substantiel pour l'époque.

Robert Sedgewick transforme véritablement la compréhension de l'algorithme. Soutenue à Stanford en 1975, sa thèse de doctorat dissèque mathématiquement Quicksort sous tous ses aspects. Ses travaux, publiés par la suite en 1977 dans *Acta Informatica*, établissent des formules précises pour calculer le temps d'exécution sur des machines réelles. Il ne s'arrête pas à la théorie et introduit le *loop unwrapping*, une technique qui réduit les coûts de gestion des boucles internes.

L'année suivante, en 1978, Sedgewick propose une nouvelle méthode de partitionnement qui fera office de référence. Deux indices parcourent le tableau en sens inverse, se rapprochant progressivement l'un de l'autre. Cette approche minimise le nombre d'échanges d'éléments, accélérant l'exécution.

Les années 1980 voient naître des variantes plus sophistiquées. Roger L. Wainwright explore en 1985 une approche hybride avec Bsort, qui mélange les techniques de Quicksort et du tri à bulles. Deux ans plus tard, il développe Qsorte, capable de détecter les sous-séquences déjà triées et d'éviter ainsi des partitionnements inutiles.

En 1993, les travaux de Bentley et McIlroy ont pour objectif de créer une version optimisée pour la bibliothèque standard du langage C. Ils conçoivent un algorithme adaptatif qui change de stratégie selon la taille du tableau. Leur innovation la plus remarquable reste le partitionnement *fat* en trois parties, spécialement conçu pour traiter efficacement les tableaux contenant de nombreux éléments identiques.

À la fin du XX^{e} siècle et au début du XXI^{e}, les chercheurs adaptent Quicksort aux chaînes de caractères, aux architectures parallèles et aux grandes bases de données. L'algorithme s'installe dans pratiquement toutes les bibliothèques standard des langages de programmation modernes.

Au-delà de ses performances, Quicksort marque les esprits par sa structure élégante. Il sert de cas d'école pour enseigner les techniques de division-conquête et l'analyse probabiliste des algorithmes. Les étudiants en informatique du monde entier décortiquent son fonctionnement, apprennent à calculer sa complexité et comprennent pourquoi il fonctionne si bien en moyenne.

L'évolution de Quicksort reflète celle de l'algorithmique dans son ensemble. Les premiers travaux se concentraient sur la complexité théorique, cherchant à prouver mathématiquement l'efficacité des méthodes. Progressivement, l'attention s'est déplacée vers les performances pratiques.

Plus de soixante ans après sa naissance, Quicksort garde toute sa pertinence. Sa rapidité moyenne, sa faible consommation mémoire et sa fiabilité en font le choix par défaut de nombreux systèmes informatiques. Les pro-

cesseurs modernes, les nouvelles architectures et les volumes de données toujours croissants continuent d'inspirer les chercheurs qui adaptent encore cet algorithme vénérable.

Bull Gamma 60

En 1960, la Compagnie des Machines Bull, fleuron de l'industrie française, dévoilait une création exceptionnelle : le Gamma 60. Cette machine, conçue pour accomplir aussi bien des tâches commerciales que scientifiques, marqua son époque par des innovations techniques audacieuses.

Le Gamma 60 se démarquait par son approche radicale du traitement de l'information. Sa particularité était sa capacité à gérer des fonctions ou problèmes simultanément sans programmation spécifique de ce parallélisme. Une prouesse technique qui rendait possible une utilisation maximale des ressources matérielles.

L'organisation interne suivait une logique modulaire avant-gardiste. Son cœur battait autour d'une unité principale dotée d'une mémoire à tores magnétiques, couplée à une unité de contrôle supérieure. Cette dernière comportait deux éléments stratégiques, à savoir un distributeur de programmes et un distributeur de données. Autour gravitaient diverses unités spécialisées : arithmétique, logique, comparaison, traduction, stockage sur tambour magnétique, sans oublier les dispositifs d'entrée-sortie tels que lecteurs de cartes et de bandes, perforateurs et imprimantes.

La mémoire centrale exploitait la technologie des tores magnétiques saturés, avec 32 768 « catenæ » – blocs de 24 bits représentant environ 786 432 bits. Cette capacité permettait de stocker 196 608 chiffres décimaux ou 131 072 caractères alphabétiques, chaque catena étant accessible en seulement 11 microsecondes. Le terme latin « catena » signifiant « chaîne » fut délibérément choisi par les ingénieurs pour marquer leur distance avec le vocabulaire informatique anglo-saxon. Ce qui n'a visiblement pas pris.

Le système de codage du Gamma 60 témoignait d'une adaptabilité remarquable. Les données numériques utilisaient un code décimal sur 4 bits, tandis que les données alphanumériques employaient 6 bits. Les instructions, quant à elles, apparaissaient en binaire pur. Cette variété de formats avait pour objectifs d'économiser de l'espace, de coder efficacement, de simplifier la conversion entre supports, et de maximiser les débits de transfert.

L'unité arithmétique faisait les opérations sur des nombres en virgule fixe comme en virgule flottante. La représentation numérique mobilisait

deux catenæ, structurées avec un bit de signe, 40 bits pour coder 10 chiffres décimaux et 7 bits dédiés à l'exposant décimal, couvrant une plage de 0 à 79. Selon leur complexité, les instructions nécessitaient 2 à 4 catenæ. À titre d'exemple, une addition en virgule flottante à trois adresses prenait 150 microsecondes, contre 88 microsecondes pour une version à adresse unique.

Les périphériques du système affichaient des performances solides. Les unités de bande magnétique utilisaient des supports d'un demi-pouce comportant 8 canaux d'information. Avec une densité de 200 bits par pouce et une vitesse de défilement de 50 pouces par seconde, elles atteignaient des débits de 20 000 chiffres décimaux ou 13 333 caractères alphabétiques chaque seconde. Les lecteurs traitaient 300 cartes perforées par minute, tandis que les imprimantes à volant d'inertie produisaient 300 lignes par minute, chacune pouvant contenir 120 caractères parmi une palette de 60.

La véritable avancée du Gamma 60 résidait dans son organisation du contrôle. Chaque élément fonctionnel disposait de son propre programme et de sa propre unité de contrôle. Une fois paramétrés, ces modules fonctionnaient en totale autonomie. Seuls la mémoire principale et les bus de transfert étaient mutualisés entre les différentes unités. Le distributeur de programmes orchestrait la répartition des instructions et établissait les verrouillages nécessaires pour éviter les conflits entre programmes concurrents. En parallèle, le distributeur de données gérait les priorités d'accès aux bus de transfert.

Cette architecture sophistiquée permettait d'exécuter côte à côte des tâches très hétérogènes : un calcul de paie, une inversion de matrice, plusieurs conversions de bandes vers l'imprimante, de cartes vers bande magnétique, ou de rubans perforés vers bande magnétique. Et cela sans presque aucune perte de performance : le temps perdu dans cette exécution simultanée ne dépassait presque jamais quelques pourcents par rapport à un traitement séquentiel.

Issu de l'industrie informatique française du XX^{e} siècle, le Gamma 60 fut commercialisé dès 1960, mais son parcours fut malheureusement écourté par les difficultés financières de Bull qui conduisirent au rachat de l'entreprise par General Electric en 1964. Cette acquisition mit un terme au développement indépendant de la machine, mais ses concepts sur le traitement parallèle et la modularité étaient novateurs.

APL

À Harvard, Kenneth E. Iverson planche sur sa thèse de doctorat sous la direction d'Howard Aiken et Wassily Leontief. Son travail porte sur la résolution automatique d'équations différentielles linéaires dans un modèle économique. Cette première incursion dans la programmation l'interroge : comment mieux représenter les algorithmes et manipuler les données ?

La réponse germe en 1956 quand Iverson enseigne dans le tout nouveau programme d'Automatic Data Processing qu'Aiken vient de créer à Harvard. Il commence à élaborer une notation formelle pour décrire et analyser le traitement des données. Cette notation, qui n'a pas encore de nom, lui sert d'abord comme outil pédagogique. Avec Frederick P. Brooks Jr., alors doctorant, il rédige un livre intitulé « Automatic Data Processing » qui formalise ces premières réflexions.

L'année 1957, Iverson travaille comme consultant chez McKinsey & Company. Sa notation trouve là un terrain d'application concret, agissant tel un langage de communication entre concepteurs et programmeurs sur des systèmes complexes. Les équipes découvrent l'intérêt de cette approche pour clarifier des architectures autrement difficiles à appréhender.

Le hasard des carrières amène Iverson chez IBM en 1960. Il y rencontre Adin Falkoff, qui s'intéresse immédiatement à cette notation pour ses travaux sur les mémoires à recherche parallèle. Cette collaboration va transformer une idée académique en nouveau paradigme informatique.

En 1962, Iverson publie « A Programming Language ». L'ouvrage présente formellement sa notation, mais celle-ci reste cantonnée aux tableaux noirs et au papier. Pour l'utiliser sur ordinateur, il faut la traduire manuellement en Fortran ou dans un autre langage existant. Une étape intermédiaire frustrante mais nécessaire.

Tout change en 1964. Iverson et Falkoff décident de transformer cette notation théorique en langage de programmation opérationnel. Lawrence M. Breed les rejoint en 1965 pour concevoir une version pratique d'APL destinée aux systèmes interactifs à temps partagé. La première contrainte à surmonter concerne les caractères. Le terminal IBM 1050 ne peut afficher que 88 symboles. Cette limitation, d'abord perçue comme un handicap, pousse l'équipe à rationaliser et généraliser plusieurs aspects du langage. Finalement, cette contrainte se révèle bénéfique puisqu'elle force une clarification conceptuelle salutaire.

Dès 1970, APL conquiert ses premiers utilisateurs. Des milliers de personnes l'adoptent, comme les équipes de recherche et de conception d'IBM, le personnel du NASA Goddard Space Flight Center, les planificateurs financiers de grandes entreprises. Le langage séduit par son élégance et son

caractère pratique, malgré les critiques de la communauté informatique établie qui le juge trop ésotérique.

Au Laboratoire de Physique Appliquée Johns Hopkins, le Centre de Calcul F.T. McClure pousse l'innovation plus loin. L'équipe développe des logiciels système APL spéciaux pour l'ordinateur IBM 360/91. Ces développements exploitent les capacités de traitement vectoriel de la machine et transforment radicalement les performances d'exécution. En 1978, le centre franchit une nouvelle étape avec un système APL capable de gérer des applications très volumineuses grâce à la technologie de mémoire virtuelle.

La vraie innovation d'APL réside dans son traitement des tableaux comme objets primitifs. Cette approche bouleverse la façon de concevoir les algorithmes. Les programmeurs pensent naturellement en termes de traitement vectoriel et matriciel plutôt qu'en boucles et conditions. Le langage se distingue par sa simplicité syntaxique remarquable avec seulement trois types d'instructions possibles. L'affectation de nom, le branchement, ou ni l'un ni l'autre. Les règles sémantiques restent peu nombreuses et les définitions des fonctions primitives s'affranchissent des représentations de données.

Les années 1970 voient APL progresser malgré le soutien limité des principaux fournisseurs d'ordinateurs. L'introduction du système de variables partagées en 1973 avec APLSV est une innovation qui résout plusieurs problèmes systèmes d'APL/360, particulièrement la communication entre programmes concurrents et avec des périphériques externes. Les possibilités s'élargissent considérablement.

La décennie suivante confirme cette expansion. APL s'installe sur pratiquement tous les ordinateurs du marché, y compris les ordinateurs personnels comme l'IBM PC et le Tandy TRS-80. De nombreuses sociétés de temps partagé commercial proposent APL comme service. Certaines se spécialisent dans les bases de données financières internationales mises à jour en temps réel, exploitant la capacité du langage à traiter efficacement de gros volumes de données numériques.

Contrairement aux autres langages majeurs de l'époque qui s'éloignent des notations mathématiques traditionnelles, APL poursuit leur développement. Il conserve les symboles établis quand c'est possible et emploie une terminologie mathématique cohérente. Cette fidélité aux conventions mathématiques n'empêche pas quelques écarts dictés par les principes de simplicité et d'uniformité, comme l'adoption d'une forme unique pour les fonctions dyadiques et monadiques.

Le développement d'APL illustre l'importance de principes édictés clairement. D'abord la simplicité, recherchée à travers l'uniformité, la généralité et la concision. Ensuite le pragmatisme, manifesté par l'attention portée aux

applications réelles et aux limitations du matériel. Le fait que le langage ait été développé pendant huit ans sans installation machine a donné une liberté de modification impossible pour les langages contraints par une base d'utilisateurs existante.

Simula

En 1961 au Norwegian Computing Center, Kristen Nygaard initie le développement d'un nouveau langage de programmation destiné à la simulation d'événements discrets. Ce projet, qui deviendra Simula, marque le début d'une collaboration étroite avec Ole-Johan Dahl, qui le rejoint en 1962. Les deux chercheurs partagent une expérience au Norwegian Defence Research Establishment, mais leurs parcours diffèrent : Nygaard s'est concentré sur la recherche opérationnelle et les simulations Monte-Carlo, tandis que Dahl a développé des logiciels système et des langages de programmation.

Le développement de Simula répond au besoin concret de disposer d'outils adaptés pour simuler des systèmes complexes comme les réseaux de neurones, les systèmes de communication ou les flux de trafic. Mais la programmation de telles simulations en langage machine ou en Fortran s'avère ardue. Il manque un ensemble cohérent de concepts pour comprendre et décrire la structure et les interactions dans ces systèmes complexes.

La première version du langage, Simula I, est développée entre 1962 et 1964 sous contrat avec UNIVAC. Les concepteurs choisissent de baser leur langage sur ALGOL 60, notamment pour sa structure en blocs et sa sécurité de programmation. L'innovation majeure de Simula I réside dans l'introduction du concept de processus, qui représente des entités capables d'exécuter des actions en quasi-parallèle tout en portant leurs propres données. Ces processus peuvent interagir et être programmés pour simuler le comportement d'objets du monde réel.

Le développement de Simula I nécessite un nouveau système de gestion de la mémoire basé sur des listes chaînées, développé par Dahl, s'affranchissant de la contrainte « dernier entré, premier sorti » (LIFO) d'ALGOL 60 et autorise une existence des processus indépendante de la structure d'appel des blocs. Le compilateur est opérationnel fin 1964 et le langage connaît un certain succès dans la communauté de la simulation. Il est notamment utilisé sur les ordinateurs UNIVAC 1100, Burroughs B5500 et URAL 16. Cependant, Dahl et Nygaard réalisent que les concepts développés pour

Simula I pourraient être utiles au-delà de la simulation, pour la programmation générale et la conception de systèmes.

Cette réflexion mène au développement de Simula 67, une refonte complète du langage intégrant les concepts de classe et d'héritage. L'inspiration vient en partie des travaux de Tony Hoare sur la manipulation des enregistrements, présentés lors d'une école d'été de l'OTAN en 1966. La percée conceptuelle survient en janvier 1967 avec l'idée du « préfixage de classe », c'est-à-dire qu'une classe peut en préfixer une autre, créant ainsi une relation d'héritage où la seconde hérite des propriétés de la première.

Simula 67 introduit des concepts fondamentaux qui influenceront durablement la programmation : les objets comme structures de données avec leurs opérateurs associés, les classes comme modèles pour créer ces objets, l'héritage pour spécialiser les classes, et les procédures virtuelles permettant de modifier le comportement des objets dans les sous-classes. Le langage distingue la vue interne d'un objet (ses variables et constantes) de sa vue externe (les procédures accessibles à distance).

La standardisation de Simula 67 s'effectue lors de la Simula Common Base Conference en juin 1967, qui établit le Simula Standards Group pour gérer l'évolution du langage. Les premières installations apparaissent dès 1969 sur les ordinateurs Control Data, suivies par des versions pour UNIVAC et IBM. La diffusion du langage est toutefois freinée par les prix élevés demandés par le Norwegian Computing Center, sous la pression du conseil norvégien de la recherche qui considère qu'un langage de programmation n'a qu'une durée de vie de trois à cinq ans.

Alan Kay et son équipe au Xerox PARC s'inspirent de Simula pour créer Smalltalk, qui popularise la programmation orientée objet dans les années 1970. Les concepts de Simula influencent le développement des interfaces graphiques, d'abord sur le Macintosh puis dans Windows. Le langage inspire de nombreux successeurs, comme le C++ par Bjarne Stroustrup, qui diffuse largement la programmation orientée objet, Eiffel par Bertrand Meyer, et plus récemment Java.

Le succès des idées de Simula s'explique en partie par leur origine dans la modélisation de simulation, où la conception en termes d'objets coopérants est naturelle. La généralité de l'approche a permis son application à de nombreux aspects du développement logiciel. Les concepts introduits par Dahl et Nygaard sont devenus si omniprésents.

SNOBOL

L'année 1962 voit naître dans les couloirs des Bell Labs de Whippany, dans le New Jersey, un projet qui changera la façon de concevoir le traitement des chaînes de caractères. David J. Farber, Ralph E. Griswold et Ivan P. Polonsky travaillent au département Programming Research Studies, une unité qui déménagera bientôt à Holmdel. Sous la direction de Chester Lee, ce laboratoire explore des territoires variés allant de la théorie des automates, en passant par l'analyse de graphes ou les processeurs associatifs. Leur quotidien les confronte aux limites frustrantes du langage SCL, créé par Chester Lee lui-même. Les performances décevantes, l'espace de données restreint et une syntaxe tortueuse qui transforme les opérations simples en véritables parcours du combattant finissent par les convaincre qu'une alternative s'impose.

Après quelques tentatives avec COMIT, un langage dédié au traitement du langage naturel, le trio décide de prendre les choses en main. Le développement de ce qui deviendra SNOBOL commence dans le plus grand secret, à l'insu de Chester Lee qui peaufine justement une nouvelle version de son SCL. Cette situation quelque peu cocasse prend une tournure embarrassante en novembre 1962 quand les deux projets sortent au grand jour. Le département ne peut se permettre de financer deux développements concurrents, mais l'équipe de SNOBOL a une longueur d'avance considérable. Chester Lee, bon joueur, accepte finalement l'inévitable.

Le baptême du nouveau langage relève de l'improvisation. D'abord appelé SCL7, puis rebaptisé SEXI pour String EXpression Interpreter, il finit par hériter du nom SNOBOL avec un pseudo-acronyme délibérément fantaisiste : StriNg Oriented symBOlic Language. Cette plaisanterie embarrassera plus tard ses créateurs lors des présentations officielles, n'imaginant pas le retentissement que prendrait leur création.

Au début de 1963, la première version opérationnelle tourne sur IBM 7090. L'équipe fait le choix audacieux de l'interprétation pure, écartant d'emblée l'idée d'un compilateur traditionnel qu'elle juge inadapté aux spécificités voulues. Cette décision s'avère payante car trois semaines suffisent pour obtenir une version utilisable, un exploit dont l'équipe tire une fierté légitime et qui prouve qu'un langage de programmation peut voir le jour en peu de temps sans engloutir des budgets considérables.

SNOBOL bouleverse les habitudes en traitant les chaînes de caractères comme des entités unifiées plutôt que comme de simples tableaux. Cette approche s'accompagne de mécanismes sophistiqués de reconnaissance de motifs avec retour arrière, ouvrant des perspectives inédites pour l'analyse textuelle. Le succès rencontré aux Bell Labs pousse naturellement vers des

versions enrichies. SNOBOL2 arrive dès 1964 avec des fonctions intégrées pour manipuler chaînes et nombres. SNOBOL3 suit en introduisant les fonctions définissables par l'utilisateur.

En 1966, SNOBOL4 révolutionne l'approche. Les motifs deviennent des objets de données à part entière, de multiples types apparaissent incluant tableaux et tables, la compilation dynamique fait son entrée avec les expressions non évaluées. L'utilisation d'un langage machine abstrait baptisé SIL garantit une portabilité remarquable.

La philosophie de distribution de SNOBOL tranche avec les pratiques de l'époque. Contrairement aux usages établis, le langage se répand gratuitement, sans restriction aucune, code source inclus. Cette générosité inhabituelle, accompagnée d'un support technique sans frais de la part des Bell Labs, facilite son adoption massive dans les universités et contribue largement à son rayonnement dans la communauté informatique.

Les applications dépassent les intentions initiales. Conçu pour manipuler des formules et analyser des graphes, SNOBOL trouve sa voie dans le formatage de documents et l'informatique appliquée aux sciences humaines. La compilation, le prototypage rapide et le traitement de données non numériques élargissent encore son champ d'action. Ses concepts de manipulation de chaînes et de reconnaissance de motifs influencent de nombreuses créations postérieures, tandis que des extensions SNOBOL enrichissent Fortran, PL/I et bien d'autres.

Partie d'un besoin pratique de quelques chercheurs, SNOBOL a fini par transformer l'approche du traitement des chaînes de caractères et influencer durablement l'évolution des langages de programmation. Elle démontre aussi que la liberté d'expérimentation et la générosité dans la diffusion, bien avant les logiciels libres, constituent des moteurs puissants pour l'innovation technologique.

Spacewar !

En 1962, le MIT vient de recevoir son nouveau PDP-1 de Digital Equipment Corporation, une machine qui tranche avec les monstres de calcul habituels. Cet ordinateur dispose d'un écran cathodique capable d'afficher des images en temps réel, une prouesse technique qui va changer la donne.

Steve Russell, jeune programmeur passionné de science-fiction, découvre cette machine avec émerveillement. Grand lecteur des aventures spatiales d'Edward Elmer Smith et de sa série *Lensman*, il imagine aussitôt recréer

sur cet écran les batailles cosmiques qui l'ont tant fait rêver. Avec quelques complices, il se lance dans la création de ce qui deviendra *Spacewar !*, sans se douter qu'il vient de donner naissance au premier jeu vidéo interactif de l'histoire.

Le principe qu'ils développent semble d'une simplicité déconcertante : deux vaisseaux s'affrontent dans l'espace, chacun manœuvré par un joueur. Rotation, accélération, tir de torpilles, les commandes de base suffisent à créer des duels acharnés. Mais Russell et son équipe placent une étoile au centre de l'écran, dont l'attraction gravitationnelle perturbe les trajectoires et transforme chaque partie en défi tactique.

Le PDP-1 n'est pas conçu pour ce genre d'exercice. Dépourvu d'instructions de multiplication et de division, il oblige les programmeurs à redoubler d'ingéniosité pour calculer rotations et trajectoires. Cette contrainte technique, loin de décourager l'équipe, stimule leur créativité. Alan Kotok et Bob Saunders fabriquent aussi des boîtiers de contrôle sur mesure, abandonnant les interrupteurs rudimentaires du panneau de l'ordinateur pour des commandes dignes d'un cockpit spatial.

Peter Samson pousse le raffinement plus loin encore. Il programme un fond d'étoiles d'une précision astronomique, reproduisant fidèlement la voûte céleste. Cette addition, baptisée avec ironie *Expensive Planetarium*, transforme l'écran en fenêtre sur le cosmos. L'équipe ajoute bientôt une fonction d'« hyperespace », un bouton de panique qui téléporte le vaisseau en détresse vers un point aléatoire de l'écran, au risque croissant de l'autodétruire à chaque usage.

La diffusion de *Spacewar !* suit un rythme différent de ce que racontent les légendes. Les 55 exemplaires qui sortent des chaînes de production, dont une vingtaine équipés d'écrans, font du PDP-1 une machine rare. Le jeu circule d'abord dans le cercle fermé des universités et centres de recherche qui possèdent ces ordinateurs d'exception. Il faut attendre la fin des années 1960 et l'arrivée de nouveaux modèles d'écrans chez IBM, DEC, HP et Control Data Corporation pour que *Spacewar !* commence vraiment à se répandre.

Chaque installation développe ses propres variantes. À l'Université du Minnesota, Albert Kuhfeld imagine en 1967 le « Minnesota Panic Button », un dispositif de camouflage qui rend les vaisseaux invisibles. À Cambridge, M.S. Peterson et John C. Viner créent DUEL, une version où l'espace se transforme en environnement « visqueux » et ralentit progressivement les combattants. À l'Université de l'Illinois, sur le système PLATO, Richard Blomme adapte le jeu avec des graphismes en caractères ASCII et invente un système de mise en relation des joueurs d'une modernité saisissante.

Ces nuits de programmation au MIT ne se déroulent jamais pendant les

heures officielles. Les étudiants attendent que les machines se libèrent de leurs tâches de recherche pour s'adonner à leurs expérimentations. Cette clandestinité nocturne forge une culture informatique inédite, où le plaisir de programmer l'emporte sur les contraintes institutionnelles. *Spacewar !* est un terrain d'apprentissage inattendu où les étudiants qui le programment assimilent les principes de l'informatique plus vite qu'avec les méthodes académiques traditionnelles.

Le jeu attire bientôt l'attention au-delà du monde universitaire. En 1971, Nolan Bushnell et Ted Dabney, après avoir découvert *Spacewar !* à Stanford, créent *Computer Space*, première tentative de commercialisation du jeu vidéo en salle d'arcade. Bill Pitts et Hugh Tuck suivent une voie identique avec *Galaxy Game*, une adaptation installée à Stanford. Ces initiatives marquent les balbutiements d'une industrie qui va bouleverser les loisirs.

L'approche de Russell et son équipe tranche avec la vision utilitariste de l'informatique. Leurs machines, initialement conçues pour le calcul scientifique et la gestion administrative, révèlent soudain leur potentiel créatif et ludique. *Spacewar !* démontre l'importance de l'interactivité, de l'interface graphique et de la programmation centrée sur l'utilisateur, des concepts qui vont influencer le développement de l'informatique personnelle.

Le code source circulant librement entre les universités représente l'esprit de partage qui caractérisera plus tard le logiciel libre. Cette ouverture encourage les programmeurs à modifier et enrichir le jeu, créant une multitude de versions. DEC adopte *Spacewar !* comme programme de test pour ses ordinateurs : si le jeu fonctionne correctement, la machine est déclarée opérationnelle.

En 1972, Stewart Brand organise au Stanford Artificial Intelligence Laboratory les « Spacewar Olympics », un tournoi qui propulse le jeu sous les projecteurs. Le magazine *Rolling Stone* couvre l'événement, présentant au grand public cette nouvelle forme de compétition électronique. Ralph Gorin a développé pour l'occasion une version sophistiquée acceptant jusqu'à cinq joueurs simultanés, agrémentée de mines spatiales et autres innovations tactiques.

ASCII

Le code ASCII naît dans les années 1960, alors que chaque constructeur informatique invente ses propres règles. À cette époque, échanger des données entre machines relève du parcours du combattant. Un caractère sur un

ordinateur IBM est illisible sur une machine Control Data, et réciproquement. Cette cacophonie technologique freine tout développement sérieux de l'informatique.

En 1961, l'American Standards Association décide de mettre fin à cette anarchie. Son comité X3 reçoit la mission de créer un standard universel. Le projet atterrit entre les mains du sous-comité X3.2, puis du groupe de travail X3.2.4. John L. Little, ingénieur au National Bureau of Standards, prend les rênes de cette entreprise délicate. Il faut convaincre des constructeurs qui utilisent déjà leurs propres systèmes, comme Control Data avec son FIELDATA sur 6 bits ou IBM avec l'EBCDIC sur 8 bits. Mais chacun campe sur ses positions.

Après des mois de débats, le comité tranche pour un code sur 7 bits. Ce compromis facilite la représentation de 128 caractères différents, suffisant pour couvrir l'alphabet latin, les chiffres et la ponctuation. Un code sur 6 bits aurait été trop restrictif, tandis que 8 bits auraient alourdi les coûts de transmission. Les 7 bits laissent aussi la possibilité d'ajouter un huitième bit pour détecter les erreurs, fonction précieuse quand les lignes téléphoniques crachotent.

ASCII voit le jour en 1963, et subit une révision en 1967 avant sa dernière version de 1986. Les concepteurs réservent les 32 premières positions aux caractères de contrôle, hérités des téléscripteurs, comme le retour chariot, le saut de ligne, la tabulation. Les 95 positions suivantes accueillent lettres, chiffres et symboles. Rien n'est laissé au hasard dans cette organisation. Les chiffres s'alignent pour faciliter les conversions, les lettres respectent l'ordre alphabétique, et la ponctuation trouve sa place logique.

En novembre 1968, le National Bureau of Standards adopte ASCII comme premier standard fédéral de traitement de l'information. Le président Lyndon Johnson signe cette décision qui rend ASCII obligatoire pour tous les achats informatiques du gouvernement américain. Comme il constitue le plus gros client du secteur, cette mesure pousse l'ensemble de l'industrie vers ASCII.

Dans la réalité, ASCII sur 7 bits s'intègre généralement dans un code sur 8 bits : soit le bit supplémentaire est à zéro, soit il sert au contrôle de parité. Cette approche s'adapte bien aux ordinateurs qui manipulent l'information par groupes de 8 bits. Le bit de parité détecte les erreurs de transmission : dans un système de parité paire, il s'ajuste pour maintenir un nombre pair de bits à 1. Si une erreur survient pendant le transfert, le décompte change et signale le problème.

Le succès d'ASCII inspire des variantes nationales. Le Canada développe sa version pour les caractères français, l'Inde crée ISCII, le Vietnam VISCII, la Yougoslavie YUSCII. Chaque pays adapte le code à ses besoins

linguistiques. En 1972, ASCII intègre la norme internationale ISO/CEI 646, qui autorise certains réaménagements nationaux. Mais ASCII s'est imposé comme standard de fait, créant parfois des confusions quand d'autres pays redéfinissent certaines positions.

L'arrivée des ordinateurs 16 et 32 bits ouvre la voie aux extensions 8 bits. Ces versions conservent les 128 caractères ASCII originaux et ajoutent 128 nouvelles positions. L'IBM PC introduit la page de codes 437, qui remplace les caractères de contrôle par des symboles graphiques. Digital Equipment Corporation conçoit le jeu multinational pour son terminal VT220, l'une des premières extensions pensées pour l'international plutôt que pour les graphiques. La norme ISO/CEI 8859 finit par s'imposer, dérivée du système de Digital. Microsoft développe ensuite Windows-1252, qui ajoute les signes typographiques nécessaires à l'impression soignée. Ces trois systèmes – ASCII 7 bits, ISO-8859-1 et Windows-1252 – dominent l'informatique jusqu'à la fin des années 2000.

Unicode prend depuis le relais d'ASCII, capable de gérer des dizaines de milliers de caractères. Mais les concepteurs d'Unicode ont eu la sagesse de préserver la compatibilité : les 128 caractères ASCII conservent leurs codes d'origine. ASCII est valide en UTF-8, l'encodage dominant actuel, ce qui en fait un sous-ensemble d'Unicode.

Plus de soixante ans après sa naissance, ASCII continue de structurer nos échanges numériques. Quand tout s'effondre, quand les formats sophistiqués échouent, ASCII se maintient. Internet repose sur lui, les protocoles de communication l'utilisent, et nos smartphones décodent encore ses caractères. Cette longévité exceptionnelle témoigne d'une conception remarquable, fruit d'un consensus industriel rare.

Modem Bell 103

En 1963, AT&T commercialise le Bell 103, un modèle de modem qui va transformer la façon dont les ordinateurs communiquent entre eux. Cette machine peut être qualifiée de technologie simple et inaugurale. Elle rend possible la transmission bidirectionnelle de données numériques sur les lignes téléphoniques ordinaires, à la vitesse respectable de 300 bits par seconde.

Cette année-là, les réseaux informatiques ne sont pas légions. Les ordinateurs, ces mastodontes coûteux, commencent tout juste à se répandre dans quelques grandes entreprises. Chaque machine demeure isolée dans

son centre de calcul, incapable de dialoguer avec ses semblables. Comment faire voyager des informations numériques sur un réseau conçu pour la voix humaine ? Le réseau téléphonique commuté représente pourtant la seule infrastructure de communication véritablement étendue et immédiatement disponible.

AT&T trouve la solution dans une technique appelée modulation par déplacement de fréquence, d'où son nom : MOdulation/DEModulation. Le principe repose sur l'idée astucieuse de transformer chaque bit en son musical. Le Bell 103 traduit les 0 et les 1 en fréquences sonores distinctes que les lignes téléphoniques savent transporter. Pour l'émission, il utilise 1070 Hz pour représenter un bit et 1270 Hz pour l'autre. À la réception, il écoute sur d'autres fréquences : 2025 Hz et 2225 Hz. Cette répartition ingénieuse autorise la communication simultanée dans les deux directions, ce qu'on appelle le *full duplex*.

La connexion entre deux modems Bell 103 ressemble à un rituel musical. L'appareil qui appelle entonne d'abord une note continue à 1270 Hz. Son correspondant lui répond par une tonalité à 2225 Hz. Cette « poignée de main » acoustique synchronise les deux machines et confirme qu'elles parlent le même langage. Une fois cette séquence accomplie, les données peuvent circuler librement.

Le Bell 103 affiche une simplicité déconcertante. Aucun mécanisme ne vérifie l'intégrité des données transmises. Les informations voyagent nues, sans protection contre les parasites ou les erreurs de transmission. Cette approche minimaliste reflète les contraintes techniques de cette période, mais aussi une certaine confiance dans la qualité du réseau téléphonique.

AT&T lance simultanément son service DATA-PHONE, qui ouvre aux entreprises les portes du réseau téléphonique public pour leurs ordinateurs. Les machines jusqu'alors prisonnières de leur salle climatisée découvrent soudain l'existence de leurs congénères. Les fichiers voyagent d'un site à l'autre, les bases de données sont accessibles à distance, le traitement distribué fait ses premiers pas.

La vitesse de 300 bits par seconde nous fait sourire aujourd'hui, mais elle révolutionne les pratiques de l'époque. Trente caractères transmis chaque seconde suffisent largement pour de nombreuses applications professionnelles. L'interface avec l'ordinateur respecte la norme RS-232, ce fameux port série à 9 broches qui équipe tous les systèmes informatiques. Cette standardisation accélère l'adoption du Bell 103 en garantissant sa compatibilité avec différentes marques d'ordinateurs.

Le succès du modem d'AT&T crée un standard de fait dans l'industrie des télécommunications. Ses fréquences de modulation et sa méthode de transmission font école. Les concurrents s'alignent sur ces spécifications

pour en assurer l'interopérabilité. Cette convergence technique culmine en 1981 quand l'Union Internationale des Télécommunications officialise ces paramètres dans sa recommandation V.21, dix-huit ans après la naissance du Bell 103.

Cette longévité exceptionnelle s'explique par la robustesse de la solution technique retenue. La modulation par déplacement de fréquence, bien que rudimentaire, fonctionne aussi sur des lignes téléphoniques défaillantes. Elle tolère les distorsions, supporte les parasites et maintient la communication quand d'autres techniques plus sophistiquées échouent.

L'utilisation du Bell 103 exige parfois des manipulations pittoresques. Pour établir une connexion, l'opérateur compose le numéro de téléphone, attend la tonalité de réponse, puis pose délicatement le combiné sur un coupleur acoustique. Ce dispositif, qui ressemble à un berceau en plastique, assure la conversion entre les signaux sonores et électriques. Les versions ultérieures intègrent heureusement la numérotation automatique, rendant la procédure moins acrobatique.

Contre toute attente, le Bell 103 survit jusqu'aux années 1990, résistant vaillamment à l'arrivée de modems plus rapides. Cette persistance remarquable tient à différents facteurs. Sa simplicité technique en fait un équipement d'une fiabilité à toute épreuve. Quand la qualité de la ligne se dégrade, il continue de fonctionner là où ses successeurs abandonnent. Sa vitesse modeste est alors un atout, avec une solution de repli acceptable. Son omniprésence dans les entreprises crée un effet de réseau puissant.

Après le Bell 103, les fabricants de modems perfectionnent les techniques de modulation pour augmenter les débits. Ils ajoutent des protocoles de correction d'erreurs pour sécuriser les transmissions. La numérotation et la réponse automatiques simplifient l'usage quotidien. Ces améliorations successives conduisent aux modems haute vitesse des années 1990, capables d'atteindre 56 000 bits par seconde sur les mêmes lignes téléphoniques analogiques que leurs ancêtres.

Music V

Les années 1960 voient naître l'idée farfelue de faire chanter les ordinateurs. Dans les Bell Labs, Max Mathews observe les oscilloscopes qui tracent les courbes de la voix humaine et se dit qu'après tout, si la parole est convertible en nombres, pourquoi ne pourrait-on pas faire l'inverse ? En 1963, il publie dans *Science* un article qui va faire grincer des dents : « The

Digital Computer as a Musical Instrument ». L'idée semble absurde. Les ordinateurs calculent, ils ne font pas de musique.

Pourtant, le principe que développe Mathews tient en quelques lignes. L'ordinateur génère une suite de nombres qui, transformés en impulsions électriques et lissés, produisent des sons. Rien de plus simple en théorie. En pratique, c'est une autre histoire. Pour obtenir quelque chose de correct, il faut au minimum 30 000 échantillons par seconde si l'on veut couvrir les fréquences audibles jusqu'à 15 000 Hz. Le hic, c'est que l'IBM 7090 des Bell Labs ne calcule que 5 000 nombres par seconde pour des sons un tant soit peu complexes. Mathews trouve alors la parade en stockant d'abord tout sur bande magnétique, puis en relisant le tout à la bonne vitesse. Cette solution de contournement marche parfaitement, mais ce n'est qu'un début, car la vraie nouveauté arrive avec MUSIC, et MUSIC V, un langage de programmation entièrement dédié à la création sonore.

La puissance de l'innovation tient dans la séparation entre la fabrication des instruments et leur utilisation. Mathews invente les « unit generators », petits blocs de programme qui font chacun une chose précise comme osciller, générer du bruit, additionner des signaux. En les combinant, on crée des « instrument units », véritables instruments virtuels. Cette modularité précède de plusieurs années les synthétiseurs de Moog et Buchla qui ne verront le jour qu'après 1964.

Le compositeur travaille donc en deux temps. Il construit d'abord ses instruments en assemblant ces générateurs comme on monte un circuit électronique. Après il écrit sa partition sous forme de listes de nombres : moment de début, durée, hauteur, amplitude... Chaque note est une ligne de chiffres. Cette approche numérique doit d'abord apprivoiser les musiciens, mais elle offre une précision inouïe sur chaque paramètre du son.

Jean-Claude Risset, qui rejoint l'équipe de Mathews aux Bell Labs entre 1964 et 1969, pousse le système dans ses retranchements. Il analyse des sons de trompette au microscope acoustique et découvre que leur timbre si reconnaissable provient de corrélations subtiles : plus la note est forte, plus elle contient d'harmoniques aiguës. Fort de cette observation, il programme des trompettes virtuelles qui reproduisent ce comportement. Le résultat bluff les auditeurs. Mais il ne s'arrête pas à l'imitation et explore les possibilités inédites qu'offre la synthèse numérique. Il crée des sons impossibles dans le monde physique, comme son fameux « glissando infini » qui semble monter éternellement tout en revenant sur lui-même, explorant ainsi les limites de la perception auditive grâce à l'ordinateur.

Les recherches menées avec MUSIC V transforment la compréhension du son. Elles montrent l'importance des transitoires d'attaque dans la reconnaissance des timbres, le rôle des micro-variations aléatoires dans la

richesse perçue. L'ordinateur est un laboratoire d'acoustique où chaque paramètre est isolable, modifiable, et son effet sur la perception observable. Cependant, MUSIC V souffre de limitations rédhibitoires. Il est impossible de jouer en temps réel, il faut attendre que l'ordinateur termine ses calculs avant d'entendre le résultat. La programmation par listes de nombres rebute les musiciens habitués aux portées, et la qualité des sons dépend d'une compréhension fine de l'acoustique que peu possèdent. Ces obstacles n'empêchent pas certains compositeurs d'adopter l'outil entre 1964 et 1969, utilisant le catalogue de sons synthétisés publié par Jean-Claude Risset comme une référence. MUSIC V démontre qu'un ordinateur peut devenir un instrument de création artistique à part entière, et pas seulement une machine à calculer.

L'architecture modulaire de MUSIC V inspire directement les synthétiseurs analogiques et numériques. Sa séparation entre définition des instruments et contrôle musical annonce la norme MIDI des années 1980. Il établit des principes de programmation – modularité, abstraction, séparation des préoccupations – qui vont bien au-delà du domaine sonore.

MUSIC V transforme la conception de l'informatique en montrant que la machine est aussi un moyen d'expression plutôt qu'un simple automate. Cette intuition de Mathews trouve aujourd'hui son aboutissement dans chaque application audio moderne qui porte en elle un peu de l'ADN de MUSIC V, ce premier cri numérique poussé dans les couloirs feutrés des Bell Labs.

Sketchpad

En 1963, Ivan Sutherland travaille dans les laboratoires du MIT sur la problématique de l'interaction entre un humain et une machine autrement qu'en tapant des lignes de code sur un clavier. Sa réponse s'appelle Sketchpad, et elle va changer la donne. Pour la première fois, dessiner directement sur un écran d'ordinateur avec un crayon lumineux est possible.

L'idée relève de la science-fiction. Les ingénieurs qui veulent dessiner un circuit électrique ou un mécanisme doivent d'abord le décrire par écrit, ligne par ligne, instruction après instruction. Autant dire que concevoir la moindre forme géométrique est un calvaire. Ivan Sutherland comprend qu'il faut casser cette logique.

Il installe son système sur l'ordinateur TX-2 du Lincoln Laboratory, une machine équipée d'un écran cathodique de 9 pouces. L'arsenal se compose d'un crayon lumineux pour tracer et sélectionner, de boutons-poussoirs

pour lancer les commandes, d'interrupteurs pour activer diverses fonctions. Quelques potentiomètres rotatifs permettent de faire pivoter ou d'agrandir ce qui s'affiche à l'écran. Mais la vraie magie opère ailleurs. Contrairement à un dessin classique figé sur le papier, un tracé Sketchpad garde en mémoire les liens entre ses différents éléments. Le système sait qu'une ligne reste parallèle à une autre, qu'un cercle demeure tangent à une droite, qu'un angle conserve sa mesure. Quand on déplace une partie du dessin, tout l'ensemble s'ajuste automatiquement pour respecter ces relations. Cette prouesse technique repose la structure de données en anneaux, une architecture logicielle audacieuse. Chaque trait, chaque courbe se transforme en bloc d'informations qui stocke sa forme, sa position et ses connections avec les autres élémentscomposant le dessin. Les éléments se parlent entre eux, se coordonnent, maintiennent leurs accords géométriques.

Ivan Sutherland pousse le concept plus loin avec ses *master drawings*. On dessine une fois un symbole, une résistance électrique ou un engrenage, et on peut ensuite le reproduire à volonté dans le dessin, chaque copie étant liée à son modèle original. Si on modifie la forme de base, toutes ses instances se transforment instantanément. Les dessinateurs industriels, habitués à redessiner sans cesse les mêmes composants, découvrent un gain de temps inouï.

Les applications se multiplient vite. Les concepteurs de circuits imprimés y trouvent leur compte pour dessiner des motifs répétitifs. Les mécaniciens simulent le fonctionnement d'articulations et de bielles. Les ingénieurs calculent la répartition des forces dans une structure comme un pont, et visualisent immédiatement les résultats. L'interface établit des règles qui traverseront les décennies. Le crayon lumineux fonctionne selon deux modes : il saisit des coordonnées quand on veut positionner un nouvel élément, et il désigne des objets existants quand on souhaite les modifier. Le système devine souvent les intentions de l'utilisateur grâce à sa « pseudo-position » : le curseur se verrouille automatiquement sur les points et lignes proches, facilitant la précision du tracé. Sketchpad transforme l'écran en planche à dessin intelligente.

Les limites de l'époque bridaient néanmoins certaines ambitions. Les calculs de contraintes ralentissaient parfois le système sur les dessins complexes. La résolution d'écran et les capacités de stockage imposaient leur loi. Mais dès 1963, Timothy Johnson étend Sketchpad vers la troisième dimension avec Sketchpad III. Lawrence Roberts s'appuie sur ces structures de données pour ses travaux de reconnaissance d'objets dans les photographies, initiant ainsi la vision par ordinateur.

Ivan Sutherland conçoit l'ordinateur non comme une calculatrice sophistiquée, mais comme un amplificateur d'intelligence humaine. Cette vision

inspire tout le mouvement de l'informatique personnelle qui suivra. En montrant qu'une conversation naturelle entre l'homme et la machine passait par l'image autant que par le texte, Sketchpad a redessiné les contours de l'informatique. Soixante ans plus tard, nos écrans tactiles et nos logiciels de dessin portent encore la trace de cette intuition pionnière, mais aussi les intelligence artificielles génératives avec leur capacité de vision.

GECOS

Bull traverse une période mouvementée quand General Electric rachète l'entreprise en 1964. Cette acquisition déclenche la création de GECOS – General Electric Comprehensive Operating System – destiné à équiper le système GE-635. Le nom porte l'empreinte de ces bouleversements industriels qui caractérisent l'informatique de l'époque. Six années plus tard, Honeywell absorbe Bull General Electric et redessine une fois de plus les contours de l'entreprise.

Ces changements de propriétaire successifs ne relèvent pas du simple jeu financier. Ils sculptent directement l'évolution technique des systèmes d'exploitation et déterminent les orientations stratégiques futures. Quand CII et Honeywell Bull fusionnent en 1976, l'héritage se complexifie davantage. L'offre commerciale présente une particularité déroutante où trois systèmes d'exploitation distincts cohabitent sous la même bannière GCOS.

Cette tripartition reflète les origines géographiques et techniques éclatées du système. GCOS6 naît des laboratoires de recherche Honeywell à Boston, GCOS7 sort des bureaux d'études Bull parisiens, tandis que GCOS8 hérite des développements initiaux de General Electric à Phoenix. Trois villes, trois équipes, trois approches techniques qui donnent naissance à des systèmes portant le même nom mais incapables de communiquer entre eux.

L'incompatibilité mutuelle de ces trois lignes constitue un paradoxe commercial troublant. Malgré leur dénomination commune, aucune application ne peut migrer de l'une vers l'autre. Les seuls échanges possibles transitent par les télécommunications, comme si ces systèmes appartenaient à des univers technologiques distincts. Cette situation illustre parfaitement les difficultés d'intégration qui surgissent quand plusieurs héritages techniques se rencontrent brutalement.

L'architecture Bull des années 1980 est aussi composée de CTOS, système multipostes développé par Convergent Technology, qui rejoint la famille après que CIT Transac en ait acquis la technologie. DNS complète cet

ensemble hétéroclite en gérant les couches basses des télécommunications : protocoles X21 et X25, Ethernet, transport et terminaux divers. Ce système s'exécute sur un frontal Datanet partagé entre GCOS7 et GCOS8, tout en s'intégrant directement sous GCOS6. Cette diversité traduit les dilemmes stratégiques de Bull face aux mutations du marché informatique. Dès 1982, Jacques Stern formule le constat lucide que Bull n'atteindra jamais la taille critique pour rivaliser mondialement avec ses seuls systèmes propriétaires, quelle que soit leur qualité technique. Cette prise de conscience déclenche une réflexion sur l'équilibre entre maintien de l'existant et préparation de l'avenir.

Les ingénieurs Bull imaginent alors des solutions techniques pour faciliter cette transition délicate. Un coprocesseur UNIX voit le jour pour GCOS7 et GCOS8, fonctionnant comme un frontal étroitement couplé au processeur propriétaire. Cette architecture hybride autorise l'ajout de fonctionnalités modernes aux applications héritées comme des bases de données relationnelles, des interfaces web ou des outils de développement contemporains. Pour GCOS6, l'approche diffère avec la création d'un émulateur sous AIX, le systèmes d'exploitation d'IBM, destiné à récupérer le code des applications métier ayant survécu au passage à l'an 2000. La maintenance de ces trois lignes distinctes s'organise selon une géographie héritée de l'histoire mouvementée de Bull. Boston continue de développer GCOS6, Paris s'occupe de GCOS7, Phoenix pilote l'évolution de GCOS8. Cette répartition internationale témoigne de la capacité de l'entreprise à préserver des centres d'expertise distincts malgré les turbulences économiques et les changements de propriétaire.

L'alliance avec IBM en 1992 découle d'une analyse réaliste : Bull ne peut plus suivre seul l'évolution effrénée du marché UNIX. L'adoption d'AIX s'accompagne d'apports techniques spécifiques de Bull, notamment dans les télécommunications et les architectures multiprocesseurs. Face à la domination américaine du secteur, cette coopération illustre une forme de spécialisation européenne. La Commission Européenne encourage d'ailleurs cette démarche en soutenant le développement d'une industrie informatique européenne forte. La création de la « Table ronde des 12 » réunit les principales entreprises du continent, Bull en tête, pour coordonner les efforts face aux géants américains. Cette initiative politique traduit la prise de conscience des enjeux stratégiques liés à l'indépendance technologique. En 2025, on voit que les résultats sont mitigés, et que les mêmes questions se posent.

La persistance de GCOS dans le paysage informatique contemporain surprend par sa longévité. Malgré les transformations répétées de Bull et l'évolution technologique accélérée, ces systèmes continuent de faire tourner des applications critiques dans de nombreuses organisations privées ou

gouvernementales. Cette résilience s'explique par la qualité technique intrinsèque des systèmes, la fidélité des clients et la capacité de Bull à faire évoluer ses plateformes tout en préservant la compatibilité.

L'histoire de GCOS révèle les contradictions et les limites de l'industrie informatique européenne. Entre héritage technique et innovation, entre spécificités locales et standards internationaux, entre indépendance et coopération, Bull a navigué dans un environnement complexe en tentant de préserver sa singularité technologique. Cette expérience éclaire les enjeux contemporains de souveraineté numérique et de diversité technologique dans un monde informatique de plus en plus uniforme.

IBM System/360

En 1964, IBM mise 5 milliards de dollars sur une idée avant-gardiste, soit l'équivalent de 50 milliards de nos jours. Jamais une entreprise privée n'avait investi de telles sommes dans un projet technologique. À cette époque, IBM règne sur le monde des machines de calcul électromécaniques grâce à sa fameuse carte perforée à 80 colonnes. Mais l'entreprise traverse une crise paradoxale. Sa gamme comprend huit ordinateurs totalement incompatibles entre eux. Chaque migration vers un système plus puissant oblige les clients à racheter tous leurs périphériques et réécrire leurs programmes. Cette situation est intenable.

Thomas Watson Jr., qui préside IBM, confie cette épineuse question à Thomas Vincent Learson. Ce dernier désigne Bob O. Evans, responsable du développement des systèmes à Poughkeepsie. Evans arrive très vite à une conclusion radicale et courageuse : il faut abandonner tous les projets en cours et repartir de zéro. La décision suscite des débats houleux, notamment avec Frederick P. Brooks Jr., qui dirige le développement de la série 8000.

L'équipe d'Evans échafaude une vision audacieuse. Le code écrit pour le plus petit modèle de la famille doit tourner sans modification sur les processeurs les plus puissants. Les imprimantes, les dispositifs de communication et les unités de stockage doivent fonctionner avec n'importe quel ordinateur de la gamme. Cette compatibilité universelle exige de repenser entièrement l'informatique. Il faut créer des logiciels d'exploitation pour une famille de processeurs compatibles, concevoir des périphériques inédits, et développer une technologie de circuits intégrés baptisée Solid Logic Technology. IBM mobilise ses ressources mondiales. Des équipes d'ingénieurs travaillent simultanément sur six nouveaux processeurs et des dizaines de périphériques,

en Amérique comme en Europe. Des centaines de programmeurs se lancent dans l'écriture de millions de lignes de code.

Le 7 avril 1964, IBM dévoile le System/360 avec une précision militaire. L'entreprise organise des conférences de presse dans 165 villes américaines et 14 pays. La présentation révèle six nouveaux processeurs et 44 périphériques. L'accueil dépasse toutes les espérances : plus de mille machines sont commandées en quatre semaines. Mais l'euphorie des commandes laisse place à deux années d'enfer. La production des modules SLT pose le premier problème. IBM construit une usine de plus de 9 200 mètres carrés à East Fishkill. La production s'envole de 500 000 modules en 1963 à 12 millions en 1964, puis 28 millions l'année suivante. Les défauts de qualité s'enchaînent, retardant les livraisons de deux à quatre mois. En 1966, l'usine fabrique 90 millions de modules, plus que tous les autres producteurs de semi-conducteurs réunis.

La logistique tourne au cauchemar. Les commandes affluent tandis que les retards s'accumulent dans la fabrication des composants et le développement des logiciels. La tension monte entre les équipes commerciales et les services de production. Un épisode cocasse illustre d'ailleurs cette période : l'entreprise manque de disjoncteurs en cuivre, paralysant temporairement toute la production.

Le développement logiciel représente le troisième écueil. Initialement budgété entre 30 et 40 millions de dollars, le coût final atteint 500 millions. L'équipe doit créer un système d'exploitation fonctionnant sur différents processeurs et périphériques, une prouesse jamais réalisée. Le code passe de 100 000 lignes pour l'IBM 1401 à plus d'un million pour le System/360.

En 1966, sept à huit mille systèmes équipent les entreprises, générant plus de 4 milliards de dollars de nouveaux revenus. IBM embauche 25 000 employés cette année-là et construit plus de 270 000 mètres carrés d'espaces de production supplémentaires. Entre 1964 et 1970, le chiffre d'affaires double, passant de 3,2 à 7,5 milliards de dollars.

Le System/360 standardise l'octet de 8 bits et introduit de nombreuses innovations en matière de stockage et de mémoire. Son architecture inspire immédiatement les concurrents. RCA lance la série Spectra 70, compatible avec le System/360. L'Union soviétique développe le Ryad, une copie presque conforme. Gene Amdahl, qui participe à la conception du système, fonde Amdahl Corporation pour produire des processeurs compatibles. Une industrie entière naît autour des périphériques compatibles avec des sociétés comme Telex, Memorex et Storage Technology.

La Bank of America, la NASA et d'autres grandes organisations choisissent le System/360 pour des applications allant du traitement des transactions bancaires aux calculs scientifiques. Grâce à son architecture uni-

fiée, les entreprises comparent désormais prix et performances sur toute une gamme de produits, simplifiant leurs décisions d'achat et leur planification informatique.

En 1985, le président Ronald Reagan remet la National Medal of Technology à Erich Bloch, Fred Brooks et Bob Evans pour leur contribution au System/360. Cette reconnaissance officielle souligne l'importance historique du projet dans le développement de l'informatique moderne. Son architecture influence toutes les générations suivantes d'ordinateurs IBM, du System/370 au System/390, jusqu'aux serveurs IBM zSeries actuels. Les principes de compatibilité et de standardisation qu'il établit demeurent des piliers de l'industrie informatique contemporaine.

BASIC

En 1964, dans les couloirs du Dartmouth College, New Hampshire, John G. Kemeney et Thomas E. Kurtz achèvent le BASIC, acronyme de Beginners All-Purpose Symbolic Instruction Code. Cette création naît d'une frustration légitime face à l'état de l'informatique d'alors. Les ordinateurs géants et coûteux exigent des compétences que seuls quelques initiés possèdent. Pour programmer, il faut d'abord se rendre dans un centre de perforation où les instructions sont laborieusement traduites en cartes perforées, puis attendre son tour dans la file d'attente du centre de calcul. Une erreur dans le code et il faut tout recommencer.

Ancien collaborateur du projet Manhattan et assistant d'Albert Einstein, Kemeney avait rejoint Kurtz à Dartmouth en 1956. Leur première rencontre avait donné naissance à une collaboration qui allait durer des décennies. Avant le BASIC, ils avaient tenté de créer des langages simplifiés comme le Darsimco ou le DOPE, mais ces expérimentations n'avaient jamais vraiment décollé. L'idée qui les animait restait toujours de rendre l'informatique accessible au plus grand nombre.

Devenu directeur du centre de calcul Kiewit, Kurtz imagine un système de temps partagé novateur. Ce système nécessite un langage à la fois simple et puissant. Les deux hommes fixent des objectifs ambitieux pour un langage polyvalent, facile d'apprentissage, extensible, interactif, et surtout indépendant du matériel. En s'inspirant du FORTRAN et de l'ALGOL, ils conçoivent une structure qui privilégie les mots anglais courants. Les lignes de programme sont numérotées, une innovation qui simplifie grandement l'édition : pour supprimer une ligne, il suffit d'indiquer son numéro, et pour la modifier, on la réécrit avec son numéro.

Le 1er mai 1964, à quatre heures du matin, deux programmes BASIC s'exécutent simultanément sur le General Electric 225 de Dartmouth College. Cette date marque la naissance officielle du langage. Kemeney et Kurtz prennent alors une décision qui influencera toute l'histoire de l'informatique : ils renoncent à breveter leur invention. Cette générosité va transformer le BASIC en phénomène mondial.

General Electric, qui avait fourni l'ordinateur à Dartmouth, figure parmi les premiers utilisateurs du nouveau langage. Vers 1970, l'entreprise commercialise des machines équipées de la cinquième version de BASIC, sans attendre la version suivante en préparation. Cette précipitation marque le début de la fragmentation du langage. Kemeney et Kurtz publient leur *BASIC the Sixth* en 1971, mais d'autres développeurs créent déjà leurs propres dialectes.

Gordon Eubanks, le futur PDG de Symantec, développe en 1970 le BASIC-E, utilisant une approche novatrice dans laquelle les instructions sont d'abord transformées en code intermédiaire avant d'être converties en code machine. Cette technique annonce celle qu'utilisera Java des décennies plus tard avec le *byte code*. Contrairement à ses créateurs, Eubanks protège sa version suivante, le CBASIC, qu'il commercialise via sa société Compiler Systems.

La multiplication des versions de BASIC inquiète l'ANSI, qui reconnaît en 1974 la nécessité d'une standardisation. Un comité se forme pour définir deux normes : le BASIC minimal et le BASIC standard. Le processus s'étale sur des années, bien après que le langage ait conquis la planète. L'ANSI publie finalement ses spécifications en 1978 pour le BASIC minimal et en 1987 pour le BASIC standard. L'ISO reprend ces normes respectivement en 1984 et 1991.

L'année 1975 voit naître le TinyBASIC de Bob Albrecht et Dennis Allison, un prodige de miniaturisation qui fonctionne avec seulement 2 Ko de RAM. William H. Gates III et Paul Allen créent leur version pour le MITS Altair. Contraints par les limitations mémoire, ils choisissent un interpréteur plutôt qu'un compilateur. Cette solution, qui tient dans 4 Ko, présente l'avantage inattendu d'un débogage grandement facilité par l'interactivité qu'elle procure. Kemeney et Kurtz, d'abord sceptiques face à cette approche, finiront par reconnaître son rôle dans la diffusion de leur langage.

L'explosion des ordinateurs personnels à la fin des années 1970 transforme le paysage du BASIC. Chaque constructeur développe sa propre version : Radio Shack avec son Level 1 BASIC pour le TRS 80, Apple avec l'Integer BASIC de l'Apple II en 1977, Commodore avec le PET BASIC, Atari avec son BASIC en 1978. La liste s'allonge avec le Timex Sinclair

1000 BASIC, le TI-BASIC de Texas Instruments, le Commodore BASIC du VIC 20 et du C64.

Au début des années 1980, IBM bouleverse le marché en intégrant un BASIC interprété dans la ROM de son PC. Extensible par le chargement de BASICA, ce BASIC accompagne chaque disquette PC-DOS. Son équivalent MS-DOS, le GW-BASIC, domine très vite le marché des compatibles PC. Microsoft franchit une nouvelle étape en 1984 avec la commercialisation de BASCOM, son premier compilateur BASIC.

L'évolution s'accélère avec QuickBASIC en 1985, qui atteint sa version 4.5 en 1988. Le langage s'enrichit d'une syntaxe structurée, de sous-fonctions, de types de données personnalisables et de la gestion multi-fichiers. En 1990, le Microsoft BASIC Professional Development System 7.1 franchit la barrière symbolique des 64 Ko de mémoire.

L'arrivée de Windows transforme une nouvelle fois le BASIC avec Visual Basic, spécialement conçu pour les applications graphiques. Cette version, dont le caractère « orienté objet » fait débat parmi les programmeurs, remporte un succès phénoménal. À la fin des années 1990, les estimations suggèrent que 90% des logiciels Windows sont développés avec Visual Basic.

L'histoire du BASIC raconte finalement celle d'une démocratisation technologique. Ce langage né dans un laboratoire universitaire a accompagné la transformation de l'informatique, d'une science réservée aux experts vers un outil accessible à tous. Il a formé des générations entières de programmeurs.

PL/I

L'informatique du début des années 1960 ressemblait à un archipel de communautés étanches. D'un côté, les scientifiques exploitaient leurs IBM 7090, codaient en FORTRAN et se retrouvaient dans les réunions de SHARE – un groupe d'utilisateurs des *mainframes* IBM. De l'autre, le monde des affaires travaillait sur IBM 7080, jurait par COBOL et organisait ses propres cercles au sein de G.U.I.D.E – un autre groupe d'utilisateurs des ordinateurs IBM. Quelques machines spécialisées comme l'IBM 7750 disposaient de leurs langages dédiés, à l'image de JOVIAL (Jules' Own Version of the International Algorithmic Language).

Cette partition géographique du territoire informatique commençait à montrer ses limites. Les chercheurs ne voulaient plus attendre des heures devant leur terminal pour récupérer une simple valeur numérique, ils ré-

clamaient des rapports structurés, lisibles, dignes de leurs travaux. Leurs fichiers de données gonflaient et rivalisaient désormais avec les volumes manipulés par les entreprises. Côté commercial, les directions marketing découvraient les joies de l'analyse statistique et réclamaient des calculs en virgule flottante sur leurs données de vente. Les responsables informatiques se retrouvaient avec deux systèmes d'exploitation sur les bras, deux équipes de programmeurs qui ne parlaient pas le même langage, littéralement, et des budgets qui s'envolaient.

IBM avait déjà posé les bases d'une réponse avec le System/360 et OS/360, cette famille de machines et de systèmes d'exploitation pensée pour unifier les besoins de tous. Restait à créer un langage de programmation qui suivrait une philosophie identique. En octobre 1963, IBM et SHARE créent le comité Advanced Language Development. Sa mission était de définir un langage universel qui mettrait fin à cette tour de Babel informatique.

L'équipe rassemblée mélangeait les genres et les compétences. Hans Berg de Lockheed apportait son expertise en formation, Jim Cox d'Union Carbide maîtrisait FORTRAN sur le bout des doigts, Bruce Rosenblatt de Standard Oil dirigeait les débats. Du côté IBM, C. W. Medlock était l'homme des compilateurs FORTRAN optimisés, Bernice Weitzenhoffer naviguait entre FORTRAN et COBOL avec aisance, George Radin pilotait le projet fort de son expérience en programmation scientifique. Les réunions s'enchaînaient tous les quinze jours, trois ou quatre jours d'affilée, principalement entre New York et Los Angeles. Les membres extérieurs à IBM gardaient leurs postes à temps plein, ce nouveau langage restant pour eux une mission annexe. Prévu pour décembre 1963, le gel des spécifications glissa finalement à février 1964. Les délais étaient tendus.

L'idée d'étendre FORTRAN fut abandonnée. Sa syntaxe ne collait pas aux terminaux modernes, ses déclarations organisées par type plutôt que par identificateur heurtaient les habitudes, son stockage des tableaux par colonne déroutait les programmeurs d'applications commerciales. Les extensions nécessaires auraient tellement transformé FORTRAN que maintenir une quelconque compatibilité perdait son sens.

Le langage s'appela d'abord NPL – New Programming Language – dans la grande tradition des acronymes descriptifs. En 1965, un conflit avec le National Physical Laboratory britannique obligea IBM à le rebaptiser PL/I (parfois écrit PL/1). La première présentation officielle eut lieu en mars 1964 à San Francisco, devant l'assemblée de SHARE. Les réactions furent partagées. Certains saluèrent son exhaustivité et son attention portée aux programmeurs de tous niveaux. D'autres pointèrent sa complexité et ses redondances. Un membre de SHARE le compara à « un couteau suisse à cent lames », une métaphore qui en disait long sur les ambitions du projet.

Le développement des compilateurs atterrit dans les laboratoires IBM de Hursley, en Angleterre. L'équipe de John Fairclough se chargea de stabiliser le langage et de le rendre utilisable en pratique. Sans les programmeurs de Hursley, PL/I serait resté une belle théorie. Ils construisirent ses compilateurs, participèrent à sa standardisation et transformèrent une spécification ambitieuse en outil de travail quotidien.

PL/I bouscula les conventions établies. Il introduisit les tâches asynchrones, définit des conditions `ON` pour spécifier des blocs de code à exécuter lors d'événements particuliers, autorisa les procédures récursives avec stockage statique et automatique. Sa palette de types de données impressionnait : arithmétiques ou chaînes de caractères, décimaux ou binaires, fixes ou flottants, réels ou complexes. Les tableaux acceptaient une dimension arbitraire avec des bornes choisies librement. Les entrées-sorties reçurent un soin particulier, elles représentaient plus de 10% de la définition du langage. PL/I mariait les capacités de formatage de FORTRAN avec certaines fonctionnalités COBOL pour le traitement des enregistrements, tout en exploitant les nouvelles possibilités d'OS/360 comme les entrées-sorties asynchrones.

L'une des particularités les plus controversées de PL/I résidait dans sa gestion des déclarations. Le langage n'exigeait pas que tous les attributs soient explicitement déclarés. En cas d'oubli, des types implicites étaient assignés selon l'usage des variables dans le programme. Les attributs manquants recevaient des valeurs par défaut dépendant de ceux explicitement mentionnés. Cette flexibilité répondait aux contraintes de l'époque où chaque soumission de programme impliquait attente et délais.

L'évolution de PL/I témoigne d'un effort constant de formalisation. Les premiers documents de 1964 laissaient de nombreuses zones d'ombre et ambiguïtés. Ray Larner et John Nicholls développèrent une première définition sémantique formelle baptisée Universal Language Definition II. Le laboratoire de Vienne prit ensuite le relais avec une définition encore plus rigoureuse. PL/I rejoignit ainsi le cercle fermé des langages les mieux définis de son temps.

Le succès fut au rendez-vous. Dans les années 1970, IBM vendait plus de licences de compilateurs PL/I que de FORTRAN pour les System/370. COBOL gardait certes la première place pour les applications commerciales, mais PL/I avait trouvé sa place. Le langage et ses variantes servirent à programmer des systèmes d'exploitation entiers comme MULTICS et OS/VS2 Release 2, ainsi que de nombreux compilateurs. Des sous-ensembles pédagogiques facilitèrent son enseignement dans les universités.

PL/I privilégiait l'efficacité du code objet et la commodité d'usage plutôt que l'élégance théorique. Son développement s'appuyait sur l'expérience

concrète des programmeurs plutôt que sur des considérations mathématiques abstraites. Cette philosophie tranche avec les préoccupations actuelles comme l'extensibilité ou la vérification formelle des programmes.

IBM CP-40

L'histoire des machines virtuelles commence dans les laboratoires du MIT, où Fernando Corbató et son équipe travaillent sur le *Compatible Time-Sharing System*. Ce système modifie l'usage des ordinateurs en introduisant le partage temporel, mais sa mise en œuvre nécessite des modifications matérielles délicates. IBM, fournisseur des machines du MIT, maintient sur place un bureau de liaison pour accompagner ces adaptations techniques.

L'année 1964, IBM dévoile sa nouvelle gamme System/360, censée unifier l'informatique d'entreprise. Pourtant, les clients du MIT découvrent avec amertume l'absence de capacités de translation d'adresses dans ces nouvelles machines. Cette fonction leur paraît indispensable pour développer leurs systèmes de partage temporel. Face à cette déception, IBM réagit en créant le Cambridge Scientific Center sous la houlette de Norm Rasmussen. L'objectif était de rattraper le retard en matière de partage temporel.

Robert Creasy, ancien du projet CTSS, prend la tête du nouveau projet CP-40. Avec Les Comeau, il imagine à la fin de 1964 une approche radicalement différente : simuler plusieurs machines System/360 indépendantes sur un seul ordinateur physique. Cette idée séduit par sa simplicité conceptuelle. Chaque utilisateur dispose de sa propre machine virtuelle, totalement isolée des autres. Fini les risques d'interférence entre programmes.

La réalisation de ce concept exige des prouesses techniques. L'équipe du CSC modifie un IBM System/360 modèle 40 en lui greffant un dispositif baptisé Cambridge Address Translator. Bruce Lindquist et Rex Seeber conçoivent ce mécanisme autour d'une mémoire associative de 64 mots. Le défi consiste à traduire les adresses sans ralentir l'exécution des programmes. Pari tenu : la translation s'effectue sans perte de performances notable.

L'architecture du système CP-40 repose sur une séparation nette entre deux composants. D'un côté, le Control Program gère la création et l'administration des machines virtuelles. De l'autre, le Cambridge Monitor System fournit un environnement d'exploitation simple pour ces machines virtuelles. Cette division du travail constitue une innovation architecturale remarquable. John Harmon dirige le développement de CMS avec son équipe

comprenant Lyndalee Korn et Ron Brennan. Ils s'inspirent largement de l'interface de CTSS pour créer un système convivial. Paradoxalement, CMS est mono-utilisateur et toute la complexité du partage des ressources incombe au Control Program.

Janvier 1967 voit la mise en production de CP-40 et CMS. Le système démontre immédiatement sa capacité à faire tourner OS/360 dans une machine virtuelle. Cette compatibilité ascendante se révélera décisive pour l'adoption future de la technologie. L'équipe découvre aussi les écueils de la mémoire virtuelle. Le phénomène de *thrashing* les surprend : quand la pagination est excessive, les performances s'effondrent brutalement.

Le succès de CP-40 ouvre la voie à CP-67, adapté au System/360 modèle 67 qui intègre nativement la translation d'adresses. Dick Bayles, Dick Meyer et Harit Nanavati mènent cette évolution. CP-67 apporte des améliorations substantielles comme la gestion dynamique de la mémoire et la flexibilité dans la configuration des machines virtuelles.

Mai 1968 marque la première distribution de CP-67 vers huit sites pilotes. En juin, le système est disponible comme programme de Type III, statut réservé aux contributions d'employés IBM. Le succès dépasse toutes les attentes. D'anciens employés créent deux entreprises commerciales qui vendent des services de partage temporel basés sur la plateforme CP/CMS : National CSS et Interactive Data Corporation.

La polyvalence du concept de machine virtuelle séduit différentes communautés. Les développeurs de systèmes d'exploitation y trouvent un environnement de test sécurisé. Les utilisateurs finaux apprécient la coexistence de systèmes d'exploitation différents sur une même machine. Les centres informatiques exploitent cette capacité pour faciliter leurs migrations technologiques.

Août 1972 voit naître VM/370 dans le cadre des Advanced Functions du System/370. L'équipe de Burlington, enrichie de Dick Newson, Carl Young et Dave Tuttle, pousse plus loin l'innovation. VM/370 introduit la capacité d'exécuter VM sous lui-même, simplification considérable pour le développement et les tests.

La séparation entre gestion matérielle et services système structure encore nos hyperviseurs contemporains. L'interface matérielle comme frontière d'isolation est un paradigme central. Cette simulation transparente des ressources matérielles s'épanouit aujourd'hui dans le *cloud computing*.

Courrier électronique

Il y a soixante ans, le monde professionnel vivait au rythme du papier. Les cadres dictaient leurs lettres à leurs secrétaires, qui maîtrisaient l'art délicat de la sténographie avant de transcrire sur machine à écrire, avec ce bruit caractéristique des touches qui frappaient le ruban encreur. Chaque message suivait le protocole rigoureux de l'en-tête officiel, des formules de politesse codifiées, et des copies carbone pour les archives. Le courrier partait ensuite dans les méandres du système postal.

Dès 1965, au MIT, Tom Van Vleck et Noel Morris bricolent quelque chose d'inédit. Sur l'ordinateur à temps partagé de leur laboratoire, ils créent une commande baptisée `mail` qui autorise les utilisateurs à s'envoyer des messages stockés dans des boîtes aux lettres virtuelles. L'idée paraît anodine, mais elle bouleverse déjà les habitudes : plus besoin de se déplacer physiquement pour transmettre une information, plus d'attente, plus de papier qui traîne.

Ray Tomlinson accomplit six ans plus tard le geste qui va changer l'histoire de la communication humaine. Cet ingénieur de Bolt Beranek and Newman travaille sur ARPANET quand il décide d'adapter le programme `sndmsg` pour faire communiquer des ordinateurs distants. Il lui faut un moyen de distinguer l'utilisateur de sa machine, et c'est presque par hasard qu'il choisit le symbole `@` sur son clavier. Cette arobase, qui servait aux comptables pour noter « tant d'unités à tel prix », fait le pont entre l'identité numérique et l'adresse physique de l'ordinateur. Premier message envoyé : « QWERTYUIOP » ou quelque chose d'approchant. Tomlinson en a oublié le contenu exact, ce qui montre qu'il ne mesurait pas l'ampleur de son invention.

L'effet de contagion est immédiat sur ARPANET. Les chercheurs et ingénieurs découvrent qu'ils peuvent échanger sans contrainte horaire ni géographique, et instantanément. Fini les appels téléphoniques interrompus, terminées les lettres qui mettent une semaine à parcourir le pays. Larry Roberts, qui dirige le bureau IPTO de l'ARPA, comprend vite l'intérêt de cette technologie. En 1972, il développe `rd`, un programme qui range et classe les messages reçus. Barry Wessler l'améliore avec `nrd`, et John Vittal franchit un cap décisif en 1974 avec MSG, qui introduit les fonctions « répondre » et « transférer ». Le courrier électronique cesse d'être un simple système de transmission pour devenir un véritable outil de conversation.

La technique suit l'usage. Les premiers formats de message ressemblent à du bricolage, chaque système ayant ses propres conventions. La RFC 561 de 1973 tente de mettre de l'ordre en définissant les champs `From:`, `Subject:` et `Date:`. Ce travail de normalisation se poursuit laborieusement jusqu'à la

RFC 822 de 1982, qui définit les bases du format encore utilisé aujourd'hui.

Jon Postel, figure légendaire d'Internet, s'attaque au problème du transport des messages. Les premiers systèmes utilisent FTP, mais ce protocole s'avère inadapté au volume croissant d'échanges. Il conçoit Simple Mail Transfer Protocol en 1982, un protocole d'une élégante simplicité qui va traverser les décennies sans prendre une ride. Cette longévité exceptionnelle dans l'univers informatique témoigne de la qualité de sa conception.

Les années 1980 voient proliférer une mosaïque de réseaux incompatibles. UUCP relie les machines UNIX, CSnet dessert les universités, Bitnet connecte les centres de calcul. Chaque réseau a ses règles d'adressage, ses conventions et ses limites. Les administrateurs système jonglent avec des adresses abracadabrantes du type `ihnp4!ucbvax!user@host`, véritables plans de route informatiques qui spécifient le chemin exact du message. Cette complexité disparaît avec l'arrivée du système de noms de domaine en 1984 et des enregistrements `MX` deux ans plus tard. L'adresse `user@domain.com` s'impose alors comme le standard universel.

En 1981, Eric Allman développe `sendmail`, et Dave Crocker crée MMDF en 1982. Ces logiciels sophistiqués gèrent l'acheminement des messages entre systèmes hétérogènes, mais butent sur une limitation : le courrier électronique ne sait transporter que du texte ASCII. Impossible d'envoyer une photo, un document formaté ou une présentation. Les utilisateurs contournent le problème en encodant les fichiers binaires, mais la manipulation est délicate. La norme MIME de 1992 libère enfin le courrier électronique de ces chaînes en autorisant les contenus multimédias. Cette évolution transforme la messagerie en véritable plateforme d'échange professionnel.

De quelques centaines d'utilisateurs pionniers sur ARPANET, le courrier électronique rassemble aujourd'hui plus de 4,37 milliards de personnes. Cette adoption massive s'explique par des choix techniques remarquablement judicieux : protocoles simples et robustes, standards ouverts et architecture décentralisée. Le système s'est adapté à l'évolution des usages sans jamais rompre sa compatibilité fondamentale. L'émergence des messageries instantanées et des réseaux sociaux n'a pas réussi à détrôner complètement cette vieille dame de soixante ans, qui continue de faire transiter chaque jour plus de 350 milliards de messages à travers la planète en 2024.

Multics

En 1962, Joseph Carl Robnett Licklider, qui dirige alors la DARPA, lance l'idée du système d'exploitation Multics. Le contrat tombe en août

1964. Multics prolonge les travaux du Compatible Time-Sharing System, développé au MIT sous Fernando Corbató. Les ordinateurs fonctionnent encore comme des machines à traitement par lots, les programmes s'exécutant les uns après les autres, et les utilisateurs attendant parfois des jours entiers leurs résultats. Les premiers systèmes à temps partagé peinent à gérer plus de quelques programmes simultanément, contraints de décharger sur bande magnétique ceux qui ne s'exécutent pas.

Le projet réunit une alliance inhabituelle constituée du MIT pour la recherche, de General Electric pour le matériel, et des Bell Labs pour le développement logiciel. GE décroche le contrat après qu'IBM ait boudé les concepts de pagination et de segmentation proposés. Cette convergence de talents universitaires et industriels rassemble quelques-uns des plus grands esprits de l'informatique de cette période.

L'ambition technique dépasse tout ce qui a été tenté jusque-là. Multics intègre mémoire virtuelle, protection par segmentation, système de fichiers hiérarchique, multitraitement à mémoire partagée, sécurité et reconfiguration en ligne. Le système compte parmi les premiers à être développé principalement dans un langage de haut niveau et à supporter plusieurs langages de programmation. MacLisp et `troff` trouvent leurs origines dans Multics, ainsi qu'une large part de la ligne de commande UNIX moderne.

La concrétisation se révèle ardue. Le matériel GE 645 arrive en retard, et Multics n'atteint l'auto-hébergement qu'en 1968. En 1969, la DARPA menace d'arrêter les frais, poussant les Bell Labs vers la sortie. Ce retrait donnera naissance à UNIX, créé par d'anciens de Multics : Kenneth Lane Thompson, Dennis MacAlistair Ritchie, Douglas McIlroy et Joseph Francis Ossana.

Honeywell, qui a racheté l'activité informatique de General Electric en 1970, commercialise Multics de 1973 jusqu'en 1985 au prix initial de 7 millions de dollars. La clientèle se compose de l'US Air Force, d'universités comme celle du Sud-Ouest de la Louisiane et le système universitaire français, de grandes entreprises comme General Motors et Ford. Au total, environ 80 licences trouvent preneur. Le dernier système Multics, exploité par le ministère canadien de la Défense nationale, s'est éteint en 2000.

L'architecture système repose sur deux piliers : les processus et les segments. Les processus fournissent les contextes d'exécution, tandis que les segments stockent code, données et périphériques d'entrée-sortie. Ces segments s'organisent dans une hiérarchie de répertoires, ancêtres des systèmes de fichiers actuels. Le domaine de protection d'un processus délimite les segments accessibles et les opérations autorisées.

Concernant la sécurité, Multics introduit les anneaux de protection, structure hiérarchique allant de l'anneau 0, le plus privilégié, aux anneaux

supérieurs moins puissants. Le GE 645 prévoit 64 anneaux, mais seuls 8 sont réellement mis en œuvre. Le superviseur Multics, confiné aux anneaux 0 et 1, ne peut être modifié que par du code s'exécutant dans ces anneaux privilégiés. Cette approche influence encore les processeurs modernes, qui utilisent généralement deux niveaux : superviseur (*kernel*) et utilisateur.

En 1974, Paul Karger et Roger Schell de l'US Air Force mènent une analyse de vulnérabilité qui révèle plusieurs failles. Une vulnérabilité matérielle contourne les vérifications d'accès dans certains cas d'adressage indirect. Des failles logicielles apparaissent, notamment liées au mode maître d'exécution dans les anneaux utilisateur, modification introduite pour améliorer les performances. Ces découvertes illustrent la difficulté de maintenir la sécurité d'un système complexe face aux pressions d'optimisation.

Plusieurs facteurs expliquent l'échec commercial relatif de Multics. Les circuits intégrés évoluent à un rythme exponentiel, transformant le paysage informatique. Quand Multics arrive sur le marché en 1973, il répond parfaitement aux problèmes de 1964, mais le monde a changé. Le système coûte trop cher, dépend trop d'une architecture propriétaire, et est trop centré sur l'informatique partagée centralisée alors que l'informatique départementale décentralisée prend son envol. En parallèle, Digital Equipment Corporation commercialise des versions d'entrée de gamme du PDP-11, et le microprocesseur 8080 fait tourner les premières versions de CP/M dès 1974.

DEC PDP-8

En 1965, Digital Equipment Corporation surprend le monde informatique en présentant le PDP-8, un ordinateur qui tient dans la moitié d'une armoire. L'époque des monstres d'acier occupant des salles entières semble révolue. Cette machine inaugure l'ère des mini-ordinateurs et transforme radicalement notre vision de ce qu'un système informatique doit être.

L'histoire débute trois ans plus tôt avec le PDP-5. DEC y expérimente le bus d'entrée-sortie, un concept novateur. Fini les connexions radiales complexes qui imposent de prévoir à l'avance chaque équipement et son câblage vers un point central. Le bus autorise l'ajout progressif de périphériques, une souplesse inédite. Mais le PDP-5 est une ébauche et les ingénieurs de DEC voient plus loin.

Le PDP-8 naît de cette ambition. Les technologies logiques évoluent et promettent des gains de vitesse spectaculaires. Les mémoires à tores de ferrite améliorent les performances du stockage en faisant chuter le temps

de cycle de 6 à 1,6 microseconde. Rendu possible par la baisse du coût des composants logiques, le compteur ordinal, jusqu'alors stocké en mémoire, migre vers le matériel, réduisant drastiquement les temps d'exécution.

L'architecture du PDP-8 se distingue par sa simplicité. Cet ordinateur 12 bits à adresse unique ne cherche pas la complexité pour impressionner. Il assume ses limites de calculs arithmétiques réduits et de mémoire primaire modeste. Ces contraintes sont ses forces dans certains domaines, notamment le contrôle de processus industriels et les applications de laboratoire qui trouvent en lui un partenaire idéal. Contrôler un analyseur de hauteur d'impulsion ou un analyseur de spectre ne demande pas la puissance d'un géant, la précision et la fiabilité suffisent.

Le marché répond avec enthousiasme. Environ 50 000 unités se vendent jusqu'en 1979, sans compter les versions CMOS qui suivront. Pour un ordinateur non produit par IBM, ce chiffre relève du miracle. La taille du PDP-8 explique en partie ce succès. Posé sur un banc de laboratoire ou intégré dans un équipement plus large, il s'adapte aux contraintes d'espace. Cette flexibilité donne d'ailleurs naissance au marché OEM, où les constructeurs achètent des ordinateurs pour les incorporer dans leurs systèmes.

L'évolution du PDP-8 raconte l'histoire des technologies qui se succèdent. En 1966, le PDP-8/S tente de conquérir le bas de gamme avec une architecture sérielle où les données sont traitées bit par bit de manière séquentielle. L'économie réalisée se paie cher car les performances s'effondrent et les ventes déçoivent. DEC retient la leçon. En 1968, le PDP-8/I exploite les circuits intégrés de moyenne échelle pour surpasser le modèle original tout en coûtant un tiers de moins. Le PDP-8/L, lui, vise les clients qui se contentent de configurations simples. Vient ensuite le PDP-8/E et son architecture Omnibus. Ce bus unifié de 144 broches révolutionne la modularité. Le processeur, qui occupait 100 cartes dans le PDP-5, se contente désormais de trois cartes de 8 × 10 pouces. La miniaturisation s'accélère et redéfinit les possibilités. Le PDP-8/M compacte encore le concept pour séduire les installations exiguës. En 1975, le PDP-8/A tire parti des cartes hexagonales et des mémoires à semi-conducteurs. Son processeur microprogrammé tient sur une seule carte, un exploit technique remarquable.

L'apothéose survient en 1976 quand Intersil grave le premier processeur PDP-8 sur une puce unique, en technologie CMOS. Cette prouesse s'accompagne d'une consommation électrique dérisoire. Cette miniaturisation autorise des systèmes d'une compacité impensable quelques années plus tôt, comme le terminal vidéo VT78.

Le bus d'entrée-sortie du PDP-8 s'impose comme référence et sa simplicité architecturale inspire des générations d'ingénieurs. Cette machine prouve qu'un ordinateur peut concilier simplicité, puissance et économie.

Elle ouvre un marché nouveau entre les systèmes coûteux et les calculateurs spécialisés.

CDC 6600

En 1957, William C. Norris quitte son poste chez Sperry Rand avec une poignée de collègues pour s'installer dans un entrepôt désaffecté de Minneapolis. Leur projet est de fonder Control Data Corporation pour bâtir les ordinateurs les plus rapides au monde. Cette ambition démesurée prend forme quand Seymour Cray rejoint l'équipe en 1958. Six ans plus tard, leur CDC 6600 bouleverse l'informatique en devenant le premier véritable superordinateur.

IBM règne sur le marché avec son 7094, machine de référence face au 1604 de CDC. Ces ordinateurs représentent le summum de la puissance disponible, mais leurs architectures héritées du passé bridaient toute évolution majeure. Maintenir la compatibilité avec l'existant impose des compromis techniques qui limitent les gains de performance. Les transistors atteignent leurs limites physiques et ne suffisent plus à propulser seuls les machines vers de nouveaux sommets.

Seymour Cray et son équipe comprennent qu'une rupture s'impose. James Thornton, l'un des ingénieurs, assiste à un séminaire à UCLA sur les machines haute performance où une évidence s'impose : toutes exploitent une forme de parallélisme. L'équipe découvre qu'un quart du temps de calcul se perd dans la transmission des signaux entre portes logiques. Cette observation va forger leur approche avec l'arrivée des transistors silicium de Fairchild Semiconductor. Ces nouveaux composants autorisent une densité inédite et raccourcissent les chemins critiques. Les cartes de circuits peuvent désormais se monter dos à dos, tous les composants logés à l'intérieur. Cette organisation résout d'un coup les problèmes de refroidissement que pose la concentration accrue des circuits.

Le CDC 6600 brise les codes avec ses dix unités fonctionnelles spécialisées travaillant en parallèle sous la coordination d'une unité de contrôle centrale. Avec cette architecture, les différentes parties d'un programme s'exécutent simultanément, un exploit technique sans précédent. Le processeur central se concentre sur les calculs scientifiques tandis que dix processeurs périphériques gèrent les entrées-sorties et les tâches système.

L'approche anticipe remarquablement les futurs processeurs RISC. La mémoire s'organise en pages protégées par un système avancé. Un *scoreboard*

matériel sophistiqué gère les dépendances entre instructions et optimise leur exécution parallèle. Ces innovations propulsent la machine vers un million d'instructions par seconde, performance jamais atteinte.

IBM réagit à l'annonce du 6600 en promettant son propre superordinateur. Cette contre-attaque freine temporairement les commandes, les clients préférant attendre la riposte du géant. Problème : la machine d'IBM n'existe que sur le papier. CDC dépose une plainte *antitrust* et lance dans la foulée son 7600, défiant ouvertement IBM qui cette fois maitient son silence. Le 7600 est l'un des ordinateurs les plus performants jamais construits. Cinq ans plus tard, CDC remporte son procès.

Les laboratoires scientifiques, universités et centres de recherche militaire adoptent massivement le CDC 6600. Sa puissance inégalée fait progresser la météorologie, la physique nucléaire et la conception aérodynamique. SCOPE, le système d'exploitation développé spécifiquement pour la machine, apporte des fonctionnalités avancées de gestion des ressources et de multiprogrammation. Sa conception modulaire, son usage novateur du parallélisme et sa gestion sophistiquée des ressources inspirent les développements futurs. Les processeurs modernes reprennent ses principes dans la gestion du parallélisme d'instructions et l'organisation des unités fonctionnelles spécialisées. Control Data s'impose comme leader du calcul scientifique et développe la série CYBER dans la continuité du 6600.

En 1972, Seymour Cray quitte CDC pour fonder Cray Research. L'entreprise investit 300 000 dollars dans cette nouvelle société, maintenant des liens étroits avec son ancien architecte. L'émergence des microprocesseurs et mini-superordinateurs dans les années 1980 érode progressivement la position dominante de CDC. La concurrence de DEC, IBM et Cray Research s'intensifie. CDC accumule les pertes, poussant William Norris vers la sortie en 1986. Malgré plusieurs restructurations, l'entreprise scinde ses activités informatiques en 1992, marquant la fin d'une époque.

Le CDC 6600 demeure une innovation technique majeure qui établit les fondements de l'informatique haute performance. Sa conception élégante démontre qu'une architecture soigneusement pensée dépasse les limites apparentes des composants. Cette machine ouvre l'ère des superordinateurs et fixe des standards de performance qui orientent durablement l'évolution du calcul scientifique.

ELIZA

En 1966, Joseph Weizenbaum publie un article dans les « Communications of the ACM ». Il y décrit ELIZA, un programme capable de converser en langage naturel avec un être humain. Au MIT où travaille le chercheur, cette réalisation fait sensation dans les cercles de l'intelligence artificielle.

Le nom choisi pour ce programme n'est pas le fruit du hasard. Eliza Doolittle, l'héroïne de *Pygmalion* de George Bernard Shaw, incarne cette transformation d'une simple vendeuse de fleurs en dame distinguée grâce aux leçons du professeur Higgins. Joseph Weizenbaum voit dans cette métaphore une machine qui apprend à « parler » sous la supervision d'un enseignant tout en gardant ses limites, une illustration parfaite de son programme.

Sur le système de temps partagé MAC du MIT, ELIZA tourne grâce au langage MAD-SLIP sur un IBM 7094. Les utilisateurs se connectent depuis des terminaux distants et entament ce qui ressemble à une véritable conversation. Ils tapent leurs phrases, le programme répond. L'illusion fonctionne. Pourtant, les mécanismes qui animent ELIZA sont d'une simplicité déconcertante. Le programme scrute chaque phrase à la recherche de mots-clés. Une fois un terme identifié, il applique des règles de transformation prédéfinies. Si aucun mot-clé n'est détecté, il puise dans un répertoire de réponses génériques ou réutilise une transformation antérieure.

Cette approche révèle toute sa subtilité dans la hiérarchisation des mots-clés. Lorsqu'une phrase contient « je » et « tout le monde », le programme privilégie ce dernier terme, parce que les généralisations abusives méritent souvent qu'on s'y attarde. Les règles de transformation découpent les phrases selon des schémas préétablis pour les reconstituer en réponses cohérentes.

L'architecture modulaire d'ELIZA constitue sa véritable innovation. Les règles conversationnelles vivent dans des scripts séparés du cœur du programme. Cette conception autorise des modifications comportementales sans toucher au code source. Des scripts voient ainsi le jour en allemand et en gallois, preuve de la flexibilité du système.

Le script qui va rendre ELIZA célèbre simule un psychothérapeute rogerien. Carl Rogers a développé une thérapie centrée sur la personne où le thérapeute reformule les propos du patient sans prétendre tout comprendre. Cette approche épouse parfaitement les capacités limitées du programme de Joseph Weizenbaum. Face à « Je suis triste », ELIZA répond « Je suis désolé d'entendre que vous êtes triste » ou « Pouvez-vous m'expliquer ce qui vous rend triste ? ». Le programme maintient l'illusion par la reformulation, les questions ouvertes, les demandes de précision.

Joseph Weizenbaum n'anticipe pas l'impact de sa création. Les utilisateurs attribuent à ELIZA une compréhension véritable. Certains réclament des sessions privées avec le programme. Cette réaction trouble le chercheur, qui réalise que les humains projettent leurs propres interprétations sur les réponses mécaniques de la machine. Ils construisent eux-mêmes le sens de l'échange. Cette anthropomorphisation spontanée résonne encore dans nos rapports aux assistants vocaux et autres *chatbots*. ELIZA a révélé comment des mécanismes élémentaires génèrent l'illusion d'intelligence, soulevant des interrogations durables sur la nature de la compréhension.

Les faiblesses techniques du programme sautent aux yeux : aucune mémoire des échanges passés, aucun modèle de l'interlocuteur, aucune connaissance du monde réel. Les réponses naissent de transformations syntaxiques aveugles au sens. Mais ces limitations n'effacent pas l'importance historique d'ELIZA qui établit les fondements du traitement automatique du langage : reconnaissance de mots-clés, identification de contextes minimaux, sélection de transformations adaptées, génération de réponses par défaut.

Au-delà de la technique, ELIZA cristallise notre réflexion sur l'intelligence artificielle. Joseph Weizenbaum observe ce paradoxe où les mécanismes ayant été expliqués, la « magie » s'évapore et le programme se révèle comme une collection de procédures triviales. L'effet sur les utilisateurs perdure pourtant. ELIZA nous enseigne que la perception d'intelligence dans nos échanges avec les machines dépend autant de nos projections que des capacités réelles des systèmes.

Les questions que soulève le programme de Joseph Weizenbaum sur la nature de la compréhension, les frontières de la simulation et les enjeux éthiques de l'intelligence artificielle n'ont rien perdu de leur acuité. ELIZA demeure ce moment fondateur où l'informatique a commencé à explorer les territoires ambigus entre le traitement mécanique du langage et la véritable intelligence.

Information Management System

En 1966, dans les bureaux d'études de la division spatiale de Rockwell, les ingénieurs du programme Apollo jonglent avec des milliers de dessins techniques, chaque modification d'une pièce déclenchant une cascade de vérifications sur l'ensemble de la capsule spatiale. L'erreur n'a pas sa place quand il s'agit d'envoyer des hommes sur la Lune.

Uri Berman, consultant IBM détaché chez Rockwell, observe ce chaos

organisé. Avec Pete Nordyke, ils imaginent une solution audacieuse qui sépare complètement la gestion des données du code des applications. Leur prototype, qu'ils baptisent DATE (Disk Applications in a Teleprocessing Environment), transforme radicalement la manière dont les ingénieurs accèdent aux informations techniques. Fini les allers-retours incessants vers les archives papier, un terminal connecté à un IBM 7010 suffit désormais pour consulter l'état d'un dessin et ses modifications en cours. Cette séparation entre données et traitement constitue la base de Data Language/One (DL/I), une interface qui libère les programmeurs des contraintes physiques de stockage sur disque. L'idée fait son chemin qu'une application ne devrait pas réinventer la roue de la gestion des fichiers. DL/I standardise ces opérations, améliore l'intégrité des données et simplifie la maintenance des programmes.

Le projet prend une nouvelle ampleur en 1966 avec l'arrivée des ordinateurs IBM System/360. Le Dr Robert R. Brown prend la direction technique, épaulé par le consultant Bob Patrick. L'équipe se fixe l'objectif de créer un logiciel respectant les standards IBM Type 2 sans toucher au système d'exploitation OS/360. Cette contrainte, apparemment handicapante, garantira en réalité la pérennité de leur création.

La collaboration s'élargit au-delà de Rockwell. Caterpillar Tractor rejoint l'aventure, apportant ses besoins spécifiques en gestion industrielle. Une vingtaine de programmeurs travaillent sur ce qui devient IMS/360, intégrant DL/I dans une architecture plus ambitieuse. Le 14 août 1968, sur un terminal de Rockwell, apparaît le premier message `READY` d'IMS. La mission Apollo peut compter sur un nouveau compagnon informatique.

L'architecture d'IMS repose sur trois piliers : le gestionnaire de base de données (IMS DB), le gestionnaire de transactions (IMS TM) et les services communs. Cette modularité autorise une utilisation à la carte. Certains clients n'exploitent que la partie base de données (`DBCTL`), d'autres se concentrent sur le traitement transactionnel (`DCCTL`). Chacun y trouve son compte selon ses besoins spécifiques.

Une innovation organisationnelle accompagne le déploiement technique. Dan Gilbert est le premier *Database Manager* de l'histoire chez Rockwell, préfigurant le métier d'administrateur de bases de données. Ce poste centralise la responsabilité de l'organisation et de l'intégrité des données, rompant avec la dispersion antérieure de ces tâches entre différents services. L'approche hiérarchique d'IMS tranche avec l'univers relationnel qui dominera plus tard. Cette structure, née des besoins de gestion des nomenclatures spatiales, excelle dans les accès prédéfinis. Les relations entre données sont établies à l'avance, créant des chemins optimisés. Cette philosophie diffère radicalement des bases relationnelles, où les relations se construisent dyna-

miquement lors de l'exécution des requêtes.

En 1988, vingt ans après sa naissance, IMS équipe 7 000 installations dans le monde. Dans les années 2000, le système traite quotidiennement plus de 50 milliards de transactions et gère plus de 15 millions de gigaoctets de données pour plus de 200 millions d'utilisateurs. Ces volumes témoignent de sa robustesse et de sa capacité d'évolution.

L'évolution technologique n'épargne pas IMS, qui s'adapte sans renier ses principes. L'introduction d'OTMA (Open Transaction Manager Access) ouvre le système aux applications TCP/IP, brisant l'isolement des architectures réseau propriétaires IBM. Java, XML et les services Web trouvent progressivement leur place, inscrivant IMS dans l'ère de l'architecture orientée services. L'intégration avec Microsoft .NET ou SAP élargit encore son écosystème.

Le Parallel Sysplex d'IBM donne une seconde jeunesse à IMS. Le partage de données et de messages entre instances multiples optimise l'utilisation des ressources tout en renforçant la disponibilité. Ces mécanismes sophistiqués de reprise après incident en font un choix privilégié pour les applications où l'arrêt n'est pas une option.

Aujourd'hui, IMS demeure un pilier du traitement transactionnel dans les secteurs financiers, manufacturiers et gouvernementaux. Cette longévité dans un domaine marqué par des cycles d'innovation effrénés valide les choix architecturaux initiaux. La séparation des préoccupations, l'intégrité des données et la performance des traitements restent des préoccupations centrales, plus de cinquante ans après les premiers dessins techniques de la capsule Apollo.

LOGO

En 1966, quatre chercheurs de BBN (Bolt, Beranek and Newman) à Cambridge, Massachusetts, se lancent dans un projet qui va faire évoluer l'apprentissage : Seymour Papert, Wallace Feurzeig, Daniel Bobrow et Cynthia Solomon créent LOGO.

Le parcours de Seymour Papert explique beaucoup de choses. Mathématicien de formation, il a passé cinq années à Genève avec le psychologue suisse Jean Piaget, absorbant ses théories sur le développement cognitif. Ce dernier refuse l'idée que l'apprentissage consiste à remplir un cerveau vide. Pour lui, l'enfant construit activement ses connaissances. En 1964, il rejoint le laboratoire d'Intelligence Artificielle du MIT, dirigé par Marvin Minsky.

Les deux hommes partagent l'obsession commune de comprendre la pensée humaine en observant comment les machines apprennent.

Le nom LOGO provient du grec *logos*, qui signifie « mot ». C'est Wallace Feurzeig qui propose cette appellation, car le langage se concentre d'abord sur la manipulation de mots et de phrases. Les créateurs partent du constat simple que les enfants jouent naturellement avec les mots, alors que les mathématiques les terrorisent souvent. Techniquement, LOGO descend directement de LISP, le langage de l'intelligence artificielle inventé par John McCarthy. Il en hérite l'évaluation interactive, la récursivité et la manipulation de listes, tout en gardant sa syntaxe accessible aux plus jeunes.

La première expérience grandeur nature se déroule en 1969 à l'école Muzzey Junior High School. Douze terminaux connectés à un ordinateur PDP-1 accueillent des élèves de 12 ans qui programment en LOGO. Ils créent des générateurs de phrases, codent des jeux comme le morpion, développent des programmes pédagogiques. Toujours cette année 1969, Seymour Papert et Cynthia Solomon quittent BBN pour fonder le groupe LOGO au laboratoire d'IA du MIT.

L'année suivante apporte l'innovation qui va rendre LOGO célèbre : la « tortue ». Ce petit robot se déplace au sol en traçant une ligne derrière lui. Puis une version graphique apparaît à l'écran. La géométrie de la tortue transforme l'apprentissage des mathématiques : les enfants « jouent la tortue » avec leur corps, établissant un pont direct entre mouvement physique et abstraction mathématique.

Les années 1970 voient naître le concept de « micromondes », ces environnements d'apprentissage conçus autour de domaines spécifiques. Au langage et à la géométrie s'ajoutent de nouveaux territoires d'exploration. Andrea diSessa développe la « dynatortue » pour explorer la physique newtonienne. Les *sprites* sur le TI-99/4 permettent l'animation. Radia Perlman pousse l'innovation plus loin en créant des interfaces pour les très jeunes enfants : les *button boxes* et la *slot machine* rendent la programmation accessible sans savoir lire ni écrire.

En 1980, avec l'avènement des ordinateurs personnels, Seymour Papert publie *Mindstorms*, un livre qui expose sa vision pédagogique, où l'ordinateur se mue en instrument qui aide les enfants à réfléchir sur leur propre pensée. Le succès est immédiat. Les intégrations commerciales se multiplient : Terrapin et LCSI sortent leurs versions pour l'Apple II en 1981, bientôt suivies par des adaptations sur pratiquement tous les micro-ordinateurs de cette génération.

Le langage évolue et se diversifie au cours des décennies suivantes. L'orientation objet s'invite dans LOGO avec Object LOGO et TLC LOGO.

LogoWriter de LCSI marie programmation et traitement de texte. StarLogo introduit la programmation parallèle et la simulation de systèmes complexes. Les variantes prolifèrent et plus de 300 dialectes voient le jour.

La pédagogie LOGO repose sur des principes qui bouleversent l'enseignement traditionnel. Le *debugging* cesse d'être la correction honteuse d'erreurs pour devenir un processus naturel d'apprentissage par essai-erreur. Le développement de procédures apprend aux enfants à décomposer les problèmes en sous-problèmes nommés. L'élève choisit ses projets, le professeur le guide.

Ces innovations pédagogiques suscitent des débats académiques acharnés. Certaines études, notamment celles de Pea et Kurland, ne parviennent pas à démontrer les bénéfices annoncés sur les capacités générales de résolution de problèmes. Ces résultats refroidissent parfois l'enthousiasme institutionnel pour LOGO, mais ces recherches font l'objet de critiques sévères. Leur méthodologie et leur vision restrictive de l'apprentissage limitent leur portée.

LOGO traverse les décennies. Scratch, né au MIT Media Lab, reprend la philosophie basée sur la construction dans un environnement de programmation par blocs colorés. NetLogo poursuit la tradition de la simulation multi-agents. App Inventor transpose les principes de LOGO à la programmation mobile. LOGO propose une vision radicale de l'éducation. L'informatique n'est pas une matière supplémentaire à enseigner mais un outil pour repenser l'apprentissage. En donnant aux enfants le contrôle de la technologie, il offre une alternative à l'enseignement magistral. LOGO est un langage de programmation et une philosophie qui place l'enfant au cœur de son apprentissage.

Read Only Memory

Dans l'univers des composants électroniques, la ROM fait figure d'exception. Là où la plupart des mémoires s'empressent d'oublier leurs données dès qu'on leur coupe l'alimentation, elle garde précieusement ses informations, comme un coffre-fort numérique. Cette particularité lui vaut son nom : Read Only Memory, mémoire en lecture seule.

L'histoire commence dans les années 1950, quand les ingénieurs découvrent qu'ils ont besoin de stocker des données de manière permanente. Les premières ROM naissent de cette nécessité, entièrement câblées à la main dans les usines. Chaque bit se trouve littéralement gravé dans la

matière, figé au moment où le silicium prend forme. Ces ROM masquées, comme les baptisent leurs créateurs, conviennent parfaitement aux productions en série. Fabriquer cent mille exemplaires identiques d'un programme est désormais économiquement viable. Mais gare à celui qui découvre une erreur après la fabrication : impossible de revenir en arrière.

Cette rigidité finit par peser. Les années 1960 voient naître la PROM, une ROM que l'utilisateur programme seul. L'idée d'acheter une puce vierge remplie de uns binaires, puis la personnaliser avec un équipement spécialisé, séduit. L'opération demeure irréversible, ce qui vaut à ces composants le surnom peu flatteur de « programmables une seule fois ». Une faute de frappe, et direction la poubelle.

La décennie suivante apporte un vent de liberté avec l'EPROM. Pour la première fois, les développeurs peuvent corriger leurs erreurs. Le secret tient dans une petite fenêtre de quartz ménagée au sommet de la puce. Exposer le composant à une lumière ultraviolette intense efface complètement son contenu, comme une gomme électronique. Certes, l'opération prend du temps et coûte plus cher, mais quelle révolution pour qui doit tester et retester son code !

L'EEPROM débarque dans les années 1980 avec la promesse d'en finir avec les séances d'ultraviolets. Cette fois, l'effacement se fait électriquement, avec une précision qui modifie chaque octet individuellement. Les concepteurs apprécient cette chirurgie de l'information, peu importe si les temps d'écriture traînent en longueur et si le prix décourage les utilisations intensives.

La mémoire flash, apparue vers la fin des années 1980, réunit enfin tous les avantages recherchés : densité respectable, coût abordable, vitesse de lecture correcte et reprogrammation électrique. L'effacement ne fonctionne que par blocs entiers, généralement de quelques centaines d'octets à plusieurs kilo-octets, ce qui est fait son unique contrainte. Un détail technique qui n'empêche pas son triomphe commercial.

Cette évolution transforme notre rapport aux machines. La ROM peuple nos appareils électroniques. Elle abrite le BIOS des ordinateurs, ces instructions primitives qui réveillent la machine et vérifient son bon fonctionnement. Calculatrices, imprimantes, téléphones portables, tous ces objets familiers dissimulent une ROM qui conserve leurs programmes de base. Sa nature protège naturellement contre les modifications non autorisées, formant un rempart silencieux mais efficace contre les intrusions.

L'industrie trouve dans cette technologie un équilibre entre permanence et souplesse. Cette trajectoire illustre une tendance de l'informatique qu'est la recherche constante d'adaptabilité. Les laboratoires continuent d'explorer de nouvelles voies comme les mémoires résistives (ReRAM), les technolo-

gies à changement de phase, ou les approches quantiques. Ces recherches dessinent les contours de composants futurs qui pourraient marier la robustesse traditionnelle de la ROM avec des performances inédites. Certains chercheurs évoquent aussi une protection intrinsèque contre les rançongiciels. L'aventure de la ROM semble loin d'avoir atteint son épilogue.

NLS

Le 9 décembre 1968, Douglas Engelbart monte sur l'estrade d'un auditorium de San Francisco bondé de 2000 spectateurs. Pendant une heure et demie, cet homme tranquille va bouleverser la vision que chacun se fait de l'informatique. Son système NLS (oN-Line System) dévoile ce que sera l'ordinateur personnel des décennies suivantes. La souris, l'hypertexte, la vidéoconférence, l'édition collaborative en temps réel... autant d'inventions qui semblent surgir de nulle part mais qui, en réalité, mijotent dans son esprit depuis plus de vingt ans.

Au milieu des années 1940, Engelbart, technicien électronicien dans la Marine, tombe sur un article de Vannevar Bush intitulé « As We May Think ». Bush y décrit le Memex, une machine imaginaire capable d'organiser et de relier toutes les connaissances humaines. Cette lecture frappe Engelbart comme une révélation. De retour dans le civil, il rumine cette idée. Face à la complexité croissante du monde et les problèmes de plus en plus difficiles, il est convaincu qu'il faut créer des outils qui démultiplient nos capacités de réflexion et de collaboration.

En 1957, il intègre le Stanford Research Institute comme consultant. Là, il mûrit sa vision d'une informatique qui augmente l'intelligence humaine plutôt que de la remplacer. Contrairement aux partisans de l'intelligence artificielle qui rêvent de machines pensantes, lui mise sur la symbiose homme-machine. En 1962, il couche ses idées dans un rapport qui fera date : « Augmenting Human Intellect : A Conceptual Framework ». Ce manifeste lui vaut le soutien de l'ARPA, de la NASA et d'autres organismes. L'année suivante, il fonde l'Augmentation Research Center au sein du SRI. L'équipe qu'il rassemble va donner naissance au système NLS. Bill English rejoint l'aventure et co-invente avec Engelbart ce petit objet en bois qui révolutionnera l'interface homme-machine : la souris. Deux roues orthogonales dans un boîtier, voilà qui remplace avantageusement le crayon optique de l'époque. D'autres tentatives voient le jour : des dispositifs commandés par le genou, par la tête, mais rien ne vaut la simplicité de la souris. L'équipe

développe aussi un clavier à accords qui transforme l'utilisateur en pianiste de l'informatique, composant ses commandes d'une seule main.

NLS transforme la façon de travailler avec l'information. Fini les programmes qui s'exécutent en mode batch, place à l'interaction directe. Les documents se structurent, se lient entre eux, se modifient en temps réel par plusieurs utilisateurs simultanément. Il est possible de les voir sous différents angles, les rechercher, les annoter. La messagerie s'intègre naturellement au système, la vidéoconférence aussi. L'écran se partage en fenêtres, les objets se manipulent directement.

La démonstration de 1968 orchestre tout cela avec un sens du spectacle remarquable. Engelbart trône devant sa console, relié à un ordinateur distant de 50 kilomètres. Son écran et son image se projettent sur un mur de 6 mètres sur 5. Dans les coulisses, Bill English coordonne une équipe technique qui jongle avec la vidéo, le son et les liaisons à distance. La technologie de d'alors rend l'exploit d'autant plus impressionnant. Sur scène, il édite du texte comme si c'était naturel, crée des listes hiérarchiques, insère des graphiques. À 50 kilomètres de là, Bill Paxton collabore avec lui en direct. Leurs deux curseurs dansent ensemble sur l'écran géant. La conversation vidéo avec l'équipe de Menlo Park stupéfie l'audience. Ces gestes semblent tenir de la magie à cette période.

Cette « Mother of All Demos » sème les graines de notre informatique moderne. Dans l'audience, Alan Kay et Charles Irby observent attentivement. Ils rejoindront plus tard le Xerox PARC où naîtront les premières interfaces graphiques commerciales. Pourtant, le succès immédiat échappe à son créateur. Rebaptisé Augment en 1978, NLS demeure un outil de spécialistes. Le système exige un matériel onéreux et des utilisateurs experts. Pour Engelbart, la puissance prime sur la facilité d'usage. Il compare l'apprentissage de son système à celui d'un instrument de musique. Cette philosophie se heurte à l'émergence des ordinateurs personnels qui privilégient la simplicité. L'architecture centralisée de NLS vieillit mal face à ce changement. L'équipe se disperse, ses membres partant développer ailleurs des interfaces plus accessibles.

NLS n'est qu'un premier pas vers une transformation de la collaboration humaine. Engelbart imagine un processus de *bootstrapping* où chaque génération d'outils engendre la suivante, dans une spirale vertueuse entre progrès technique et évolution des pratiques. Cette démonstration légendaire incarne une certaine idée de l'informatique comme amplificateur de l'intelligence collective. Nos usages multimédias actuels sont issus de cette performance, mêlant télévision, télécommunications et informatique. Elle nous rappelle surtout que le potentiel d'augmentation de nos capacités cognitives reste largement inexploité.

Engelbart continuera jusqu'à sa mort en 2013 à explorer ces questions, notamment au Bootstrap Institute qu'il crée en 1989. Le prix Turing qu'il reçoit en 1997 couronne une œuvre visionnaire.

ARPANET

Dans les laboratoires américains du début des années 1960, les ordinateurs vivaient en autarcie. Ces machines coûteuses fonctionnaient isolément, chacune gardant jalousement ses données et ses programmes. Mais au MIT, Joseph Carl Robnett Licklider voyait plus loin. En 1962, cet homme aux idées visionnaires griffonnait dans ses mémorandums l'esquisse d'un « réseau galactique » qui transformerait radicalement notre rapport aux machines. Il imaginait un monde où n'importe qui pourrait puiser instantanément dans les ressources informatiques dispersées aux quatre coins du pays. Son rêve trouvait enfin un terrain d'expression concret en prenant enfin la tête du programme de recherche informatique de l'ARPA, cette agence du Pentagone spécialisée dans les projets d'avant-garde..

Leonard Kleinrock publiait en juillet 1961 au MIT le premier article sur la transmission par paquets, une idée qu'il développerait trois ans plus tard dans un livre complet. Pendant ce temps, Paul Baran planchait à la RAND Corporation sur un réseau de communication sans point central, où la redondance garantirait la survie des échanges en cas de destruction partielle. Outre-Atlantique, Donald Davies du National Physical Laboratory britannique explorait des voies parallèles.

L'été 1965, Lawrence Roberts et Thomas Merrill parvinrent à relier deux ordinateurs séparés par un continent entier, l'un dans le Massachusetts, l'autre en Californie. Une simple ligne téléphonique suffisait. L'expérience fonctionnait, certes, mais elle révélait aussi les limites criantes du réseau téléphonique traditionnel pour ce type d'usage.

Roberts rejoignit l'ARPA l'année suivante et donna chair au projet ARPANET. Son plan détaillé, présenté en 1967, suscita d'abord la méfiance des chercheurs concernés. L'architecture technique proposée leur semblait trop complexe. Wesley Clark sauva la mise en suggérant une solution d'une simplicité géniale : créer des ordinateurs dédiés au routage des données, baptisés Interface Message Processors. Ces fameux IMP séduisirent immédiatement. En décembre 1968, l'entreprise Bolt Beranek and Newman en décrocha le contrat de construction. Le réseau s'éveilla en septembre 1969. Le premier nœud prit ses quartiers à l'UCLA, où Leonard Kleinrock dirigeait le Network Measurement Center. Le second s'installa au Stanford

Research Institute de Douglas Engelbart, qui abritait le Network Information Center. L'université de Californie à Santa Barbara et celle de l'Utah complétèrent ce quatuor fondateur.

Le 29 octobre 1969 restera dans l'histoire : ce jour-là, les premiers bits voyagèrent entre UCLA et Stanford. Contrairement à une idée répandue, ARPANET ne naquit pas d'une obsession militaire liée à la guerre froide. L'objectif beaucoup plus pragmatique était de rentabiliser des investissements colossaux en partageant ces ordinateurs hors de prix entre centres de recherche. L'ARPA finançait des machines souvent incompatibles qui dormaient une bonne partie du temps. Les connecter relevait du simple bon sens économique.

Steve Crocker prit la tête du Network Working Group qui accoucha en décembre 1970 du Network Control Protocol, premier protocole de communication entre ordinateurs hôtes d'ARPANET. L'adoption se fit graduellement entre 1971 et 1972, permettant aux premières applications véritables de voir le jour. Bob Kahn orchestra en octobre 1972 une démonstration publique qui fit sensation lors de l'International Computer Communication Conference de Washington. Depuis les sous-sols de l'hôtel Hilton, les visiteurs découvraient ce réseau mystérieux en temps réel. Le succès balaya toutes les prévisions. Cette année vit aussi naître le courrier électronique de Ray Tomlinson, première application qui conquit vraiment les utilisateurs.

La croissance devint exponentielle. Quinze nœuds fin 1971. En 1973, premières liaisons internationales vers l'University College de Londres et le Royal Radar Establishment norvégien. ARPANET venait d'inventer le premier réseau international de commutation de paquets.

Vinton Cerf et Bob Kahn franchirent un cap décisif dès 1973 en concevant TCP/IP. Ils voulaient interconnecter des réseaux de natures différentes, comme les futurs réseaux radio et satellite. Leur architecture ouverte permettait à chaque réseau de conserver ses spécificités tout en communiquant avec les autres via TCP/IP. Le 1er janvier 1983, ARPANET basculait vers TCP/IP. Cette migration technique exigeait de modifier simultanément tous les ordinateurs du réseau. Pour forcer la transition, les concepteurs désactivèrent progressivement l'ancien protocole NCP en 1982, un jour après l'autre, avant l'arrêt définitif. L'opération réussit parfaitement, démontrant la maturité de la nouvelle architecture.

ARPANET inventa aussi une méthode de développement qui survit aujourd'hui. Steve Crocker créa dès 1969 les Request for Comments, ces documents de travail informels où les chercheurs partageaient leurs réflexions. Les RFC devinrent l'outil de référence pour documenter et normaliser le réseau. Cette tradition d'ouverture traverse encore les décennies. Le Network Working Group initial évolua vers l'Internet Working Group et diverses

structures comme l'Internet Activities Board. Des chercheurs et ingénieurs y développaient les protocoles ensemble, loin des pressions commerciales ou politiques.

ARPANET transforma la collaboration scientifique. Il fit naître une communauté internationale soudée par une culture commune d'ouverture et de coopération. Les choix techniques et organisationnels des années 1970 ont influencé l'architecture et la gouvernance de l'Internet contemporain. En 1990, ARPANET s'effaça discrètement, remplacé par NSFNET et les premiers réseaux commerciaux qui utilisaient TCP/IP.

B

En 1969, dans les couloirs des Bell Labs, Ken Thompson et Dennis Ritchie travaillent sur UNIX, ce système d'exploitation qui promet de révolutionner l'informatique. Mais comment programmer efficacement sur ces mini-ordinateurs aux ressources limitées sans se perdre dans les méandres de l'assembleur ?

Thompson connaît bien BCPL, ce langage développé par Martin Richards à Cambridge. Puissant, certes, mais trop lourd pour les PDP-7 et PDP-11 dont disposent les chercheurs. L'idée germe alors de créer une version simplifiée, épurée, qui garderait l'esprit de BCPL tout en s'adaptant aux contraintes matérielles. C'est ainsi que naît B, nom choisi par simple contraction alphabétique qui s'avérera prémonitoire avec l'arrivée ultérieure du langage C.

La philosophie de B tient dans sa simplicité radicale. Fini les types complexes et les subtilités syntaxiques qui alourdissent le développement. Dans B, tout est mot machine. Les variables se dimensionnent automatiquement selon l'architecture, généralement 36 bits. Cette approche, qui pourrait sembler limitative, cache en réalité une efficacité redoutable pour la programmation système. Chaque octet compte quand les ressources se mesurent en kilo-octets.

La syntaxe hérite de BCPL mais Thompson la dépouille de ses fioritures. Les structures de contrôle classiques demeurent : `if`, `while`, `for`. Les opérateurs arithmétiques et logiques conservent leur notation familière. Une innovation marque durablement le paysage informatique : l'usage d'accolades pour délimiter les blocs de code. Cette convention, apparemment anodine, traversera les décennies pour se retrouver dans d'innombrables langages modernes.

B supporte les pointeurs et les tableaux, concepts indispensables pour manipuler la mémoire au plus près. Cette capacité se révèle essentielle pour UNIX, où chaque cycle processeur doit être optimisé. Thompson et Ritchie développent un compilateur remarquablement compact, tenant en quelques milliers de lignes. Plus tard, ils y ajoutent un interpréteur pour faciliter le développement interactif. Cette dualité compilation-interprétation offre le meilleur des deux mondes : performance pour la production, flexibilité pour les tests.

Des portions entières du système d'exploitation, initialement codées en assembleur, sont réécrites en B, faisant d'UNIX son terrain d'expérimentation privilégié. L'expérience révèle les forces et faiblesses de B. D'un côté, le développement s'accélère considérablement et facilite la maintenance du code. De l'autre, certaines limitations surgissent, notamment dans la manipulation de chaînes de caractères où l'absence de distinction entre caractères et entiers complique les opérations.

La portabilité guide les choix de conception. Dans un monde informatique fragmenté entre architectures incompatibles, B promet de libérer les développeurs de ces contraintes matérielles. Théoriquement, une application écrite en B peut migrer d'une machine à l'autre par simple recompilation. Si la réalité se montrera plus nuancée, cette vision avant-gardiste qui influencera les langages futurs.

L'évolution des architectures vers des mots de 8 et 16 bits rend le modèle de B moins optimal. Les besoins croissants d'UNIX dépassent progressivement ses capacités. Dennis Ritchie, fort de l'expérience acquise, entreprend de concevoir un successeur. En 1972 naît C, qui conserve l'esprit de B tout en résolvant ses principales limitations grâce à un système de types plus sophistiqué.

Cette évolution illustre parfaitement la nature itérative du progrès informatique. B n'a vécu que quelques années, mais son influence dépasse cette courte existence. Chaque programmeur qui écrit une boucle `for` en Java ou JavaScript utilise une syntaxe héritée de B. Les accolades qui structurent le code, les opérateurs familiers, l'approche procédurale, autant d'éléments que B a légués à la postérité.

Aux Bell Labs et ailleurs, B sert de formation à toute une génération de programmeurs. Sa simplicité relative en fait un excellent véhicule pédagogique pour comprendre la programmation système. Nombreux sont ceux qui contribueront plus tard à l'essor d'UNIX et d'Internet après avoir fait leurs premières armes avec ce langage. Brian Kernighan rédige un tutoriel qui établit des standards de qualité pédagogique durables, démontrant qu'un langage technique peut s'enseigner avec clarté et méthode.

La philosophie de B – privilégier la simplicité sur la complexité, l'effica-

cité sur l'exhaustivité – traverse les époques. Cette approche minimaliste, où chaque élément doit justifier sa présence par son utilité pratique, caractérise encore des langages comme Go, développé par des vétérans de Bell Labs. C'est toute une culture de développement qui se transmet.

Cette histoire rappelle que l'innovation informatique procède rarement par ruptures brutales. Elle avance par accumulation, chaque génération s'appuyant sur les acquis de la précédente tout en cherchant à dépasser ses limites. B s'inspire de BCPL, engendre C, qui donnera naissance à d'innombrables descendants, dont chaque maillon compte.

Premier RFC

En 1969, une poignée d'étudiants et de chercheurs s'attelle à un projet qui changera le monde : l'ARPANET. Cette initiative de l'Advanced Research Projects Agency du Pentagone vise à connecter quatre universités de la côte ouest : UCLA, Stanford Research Institute, UC Santa Barbara et l'université de l'Utah. Chaque site dispose de machines différentes, avec ses propres systèmes et ses particularités techniques. Cette diversité, qui pourrait sembler un atout, se révèle plutôt un casse-tête.

Steve Crocker, étudiant à UCLA, peine à trouver le sommeil par une nuit d'avril. Les discussions techniques entre équipes s'accumulent, mais rien n'est consigné nulle part. Comment faire dialoguer ces machines si différentes et documenter les solutions trouvées ? Dans l'obscurité de sa chambre, il prend une feuille et commence à rédiger ce qui deviendra le RFC 1, intitulé simplement « Host Software ».

Ce document naît au sein du Network Working Group, un collectif informel de jeunes chercheurs qui se retrouvent physiquement pour discuter de ce réseau qu'ils construisent justement pour éviter les déplacements. L'ironie de la situation n'échappe à personne. Ces réunions se déroulent sans hiérarchie particulière, sans mandat officiel, dans une atmosphère décontractée qui tranche avec les pesanteurs administratives habituelles.

Crocker choisit délibérément le terme « Request for Comments » plutôt qu'une formulation plus autoritaire. Il redoute qu'un ton trop doctoral ne froisse des experts plus confirmés qui travaillent peut-être, imagine-t-il, sur les mêmes questions depuis leurs bureaux de la côte est. Cette prudence initiale forge l'esprit des RFC : documenter sans imposer, proposer sans contraindre.

Le 7 avril 1969, ce premier RFC aborde des questions sur la structura-

tion des messages entre ordinateurs qui semblent aujourd'hui rudimentaires. Crocker propose des en-têtes de 16 bits avec des adresses de destination sur 5 bits seulement, ce qui limite le réseau à 32 machines. Il évoque les *links*, ces connexions fixes entre machines, bien éloignées des connexions dynamiques actuelles. La latence du réseau, estimée à une seconde, paraît énorme comparée aux microsecondes actuelles.

Une proposition visionnaire émerge de ce document : le DEL, ou Decode-Encode Language, un langage de description des terminaux destiné à faire communiquer des systèmes hétérogènes. Cette idée a engendré les standards de communication qui structureront plus tard l'internet.

La publication de ce RFC inaugure des principes nouveaux. D'abord, la gratuité totale : les documents sont consultables par n'importe qui sans débourser un centime. Ensuite, l'ouverture : chacun peut rédiger un RFC, qu'il soit professeur titulaire ou simple étudiant. Ces règles démocratiques bousculent les habitudes académiques et industrielles. La distribution se fait d'abord par courrier postal, avant d'évoluer vers le FTP puis l'e-mail. Les premiers RFC sont parfois manuscrits ou tapés à la machine, témoins d'une époque où l'informatique se cherche. Certains documents portent encore les traces de corrections au stylo, vestiges touchants de cette période artisanale.

Jon Postel est le premier éditeur officiel des RFC en 1971. Sous son autorité bienveillante, le processus se structure sans se rigidifier. Les documents gagnent en uniformité tout en préservant leur spontanéité originelle. Il instaure des critères de qualité qui élèvent le niveau général sans décourager les contributions. Cinquante ans plus tard, plus de 8 500 RFC ont vu le jour. Ils couvrent tous les aspects d'internet, des protocoles fondamentaux aux applications les plus sophistiquées. Cette collection constitue la mémoire technique d'internet, un patrimoine documentaire unique par son ampleur et sa cohérence.

Les RFC prouvent qu'il est possible de maintenir des standards rigoureux dans un cadre ouvert et collaboratif. Cette aventure rappelle aussi que les fondations d'internet ont été posées par des jeunes gens animés par l'envie de partager plutôt que de dominer.

RAM

Dans les laboratoires américains des années 1960, une contradiction flagrante tourmente les ingénieurs. D'un côté, ils miniaturisent les composants logiques jusqu'à la taille d'un petit pois. De l'autre, ils conservent

des systèmes de mémoire à tores magnétiques gros comme des barils. Ces dispositifs hérités des années 1950 fonctionnent certes à merveille, mais leur encombrement est grotesque face aux progrès de l'intégration.

Bob Norman de Fairchild Semiconductor cherche une issue dès 1961. Il imagine d'intégrer plusieurs circuits *flip-flop* sur une puce de silicium, chaque élément stockant un bit d'information. L'idée séduit sur le papier, mais la technologie refuse de suivre. Les dirigeants de Fairchild rangent le projet au placard, le jugeant irréalisable. Pourtant, Texas Instruments livre à l'US Air Force un petit ordinateur équipé de quelques centaines de bits de mémoire semi-conductrice. Cette démonstration prouve que le concept tient la route, à petite ou grande échelle. IBM flaire le potentiel et publie en 1965 un rapport ambitieux baptisé *Potential for Monolithic Megabit Memories*. L'entreprise pousse l'expérimentation en développant une puce de 16 bits pour le System/360 Model 95, machine spécialement conçue pour les besoins de la NASA en 1966.

Tom Longo bouleverse la donne l'année suivante. Avec son équipe de Transitron, il met au point la première puce de mémoire commerciale multi-cellules. Ces 16 bits exploitent la technologie TTL et trouvent immédiatement preneur chez Honeywell. Le succès pousse Fairchild et Texas Instruments à développer leurs propres versions.

Depuis les laboratoires d'IBM, Robert Dennard révolutionne l'approche en 1966. Il invente la cellule DRAM à un transistor, prouesse technique qui simplifie radicalement l'architecture des mémoires. Les cellules SRAM existantes nécessitent plusieurs transistors par bit stocké. Cette innovation ouvre la voie aux mémoires DRAM modernes.

Fairchild relance ses ambitions en 1967 avec le programme SAM (Semiconductor Advanced Memory). L'objectife est de produire une puce de 64 bits. L'année suivante, des constructeurs proposent des mémoires TTL de cette capacité, organisées en 16 mots de 4 bits chacun, dont les temps d'accès atteignent 60 nanosecondes.

Radiation Inc., future Harris Semiconductor, introduit en 1969 les mémoires PROM (Programmable Read-Only Memory). Ces composants programmables par l'utilisateur séduisent les développeurs d'ordinateurs. Ils autorisent la modification du microcode lors des phases de débogage, souplesse inédite qui accélère le développement des systèmes.

En 1970 avec l'Intel 1103, William Regitz et Joel Karp, épaulés par Ted Hoff et Ted Rowe, mettent au point la première DRAM commercialement viable. Cette puce de 1 024 bits souffre encore de limitations en vitesse et en simplicité d'usage, mais elle prouve définitivement que les mémoires semi-conductrices peuvent équiper les ordinateurs.

Dov Frohman frappe fort l'année suivante avec l'Intel 1702, première

mémoire EPROM. Cette invention exploite une technique de grille flottante imaginée chez Bell Labs. Le principe : programmer électriquement la mémoire et l'effacer par exposition aux ultraviolets à travers une petite fenêtre en quartz ménagée dans le boîtier.

Bob Proebsting de Mostek change les règles en 1973. Sa MK 4096 propose 4 Kbit dans un boîtier de seulement 16 broches grâce à une technique astucieuse d'adressage multiplexé. Cette approche influence toutes les générations suivantes de DRAM, comme un standard industriel.

La course à la capacité s'emballe. Les puces de 16 Kbit arrivent en 1976, exploitant une technologie à double couche de polysilicium qui optimise l'agencement des cellules mémoire. Les 64 Kbit suivent en 1979, puis les 256 Kbit en 1982. Chaque génération apporte son lot d'innovations technologiques et repousse les limites de la miniaturisation.

En 1983, Intel frappe un grand coup avec la première DRAM CMOS d'1 Mbit. Cette transition technologique vers le CMOS dessine l'avenir du secteur. Paradoxalement, Intel abandonne peu après le marché des DRAM, laissant le champ libre à ses concurrents japonais et coréens qui s'engouffrent dans la faille.

L'innovation continue avec les puces d'1 Mbit en 1986. Ces composants inaugurent l'usage de cellules mémoire non planaires, empilées ou creusées en tranchée pour gagner de la place. Les capacités grimpent inexorablement : 4 Mbit en 1988, 16 Mbit en 1991, 64 Mbit en 1994. Chaque bond technologique s'accompagne de contraintes de fabrication toujours plus redoutables.

L'année 1998 voit naître les DRAM de 256 Mbit qui introduisent les diélectriques à haute permittivité. Cette innovation matérielle autorise la poursuite de la miniaturisation tout en conservant les propriétés électriques nécessaires au bon fonctionnement des cellules mémoire.

Cette course effrénée aux performances suit la loi de Moore avec une régularité saisissante. Le doublement des capacités tous les deux ans environ s'accompagne d'une chute vertigineuse du coût par bit stocké. Cette démocratisation économique propulse les mémoires vives dans tous les appareils électroniques, des ordinateurs personnels aux téléphones portables.

UNIX

Les Bell Labs bouillonnaient d'activité à la fin des années 1960. Une équipe de chercheurs travaillait d'arrache-pied sur Multics, un projet ti-

tanesque mené avec le MIT et General Electric. Cette ambition colossale était de créer un système d'exploitation capable de gérer plusieurs utilisateurs et tâches simultanément. Parmi ces visionnaires se trouvaient Ken Thompson et Dennis Ritchie, deux hommes qui allaient bientôt bousculer l'informatique.

Multics patinait. AT&T, lassée par les coûts astronomiques et les délais qui s'étiraient sans fin, claqua la porte en 1969. Thompson et Ritchie se retrouvèrent avec une expérience solide mais sans machine pour poursuivre leurs recherches. La direction refusa leur demande d'acquisition d'un nouvel ordinateur. Thompson eut l'idée de récupérer un vieux PDP-7 qui prenait la poussière dans les laboratoires. Sur cette machine désuète, il se mit à construire quelque chose d'inédit. Il développa un noyau élémentaire, un assembleur, un shell et quelques outils de base comme `rm`, `cat` et `cp`. Le nom UNIX surgit d'un jeu de mots avec Multics, clin d'œil ironique à la simplicité face à la complexité de son aîné.

L'année 1970 changea la donne. L'équipe décrocha un PDP-11. Cette machine ne disposait d'aucun disque et de seulement 24 Ko de mémoire, mais elle permit l'éclosion de la première édition officielle d'UNIX. Le développement s'emballa. Thompson créa le langage B, version allégée du BCPL, tandis que Ritchie planchait sur ce qui deviendrait le langage C entre 1971 et 1973.

Cette invention du C bouleversa tout. Dès la quatrième édition d'UNIX en 1973, Thompson et Ritchie osèrent réécrire entièrement ce nouveau langage. Ce pari, audacieux à l'époque où les systèmes d'exploitation s'écrivaient en assembleur, rendit UNIX portable d'une architecture à l'autre. Les six premières éditions se bousculèrent entre 1971 et 1975.

La sixième édition, surnommée V6, marqua la première grande diffusion d'UNIX hors des Bell Labs. AT&T distribua le système aux universités avec son code source complet pour une bouchée de pain, décision qui secoua le milieu informatique.

L'Université de Californie à Berkeley fit son entrée dans l'épopée UNIX en 1975 quand Bob Fabry y installa la quatrième édition. Ken Thompson y passa une année sabbatique qui galvanisa deux étudiants brillants : Bill Joy et Chuck Haley. Ces derniers se lancèrent dans l'enrichissement du système, développant les éditeurs `ex` et `vi`, qui demeure aujourd'hui un pilier de l'édition de texte, au point d'inspirer des projets comme neovim. En 1977, Bill Joy assembla la Berkeley Software Distribution (BSD), tirée à seulement une trentaine d'exemplaires mais qui allait faire des émules.

Les années 1980 virent éclore une myriade de versions UNIX. AT&T commercialisa System III puis les différentes moutures de System V via son UNIX System Group. Berkeley poursuivait parallèlement le dévelop-

pement de BSD, apportant des innovations remarquables comme le Fast File System et surtout la pile réseau TCP/IP. Cette dernière, intégrée à la version 4.2BSD en 1983, propulsa BSD vers la gloire.

Bill Joy quitta Berkeley en 1982 pour co-fonder Sun Microsystems, société qui allait peser lourd dans l'évolution d'UNIX. Sun développa le Network File System (NFS) et l'architecture VFS/vnode, rendant le partage de fichiers en réseau invisible pour l'utilisateur. En 1987, une alliance inattendue se noua entre Sun et AT&T pour créer System V Release 4, fusion des meilleures caractéristiques de SunOS et de System V Release 3.2.

Cette prolifération de versions UNIX créa un besoin criant de standardisation. Le groupe `/usr/group`, association pour la promotion d'UNIX constituée en 1980, s'attaqua à l'uniformisation des interfaces utilisateur. Ces efforts débouchèrent en 1985 sur la naissance du standard POSIX (Portable Operating System Interface for Computing Environments), référence désormais incontournable pour la compatibilité entre systèmes UNIX. Le groupe `X/Open`, créé en 1984, élabora le X/Open Portability Guide (XPG), intégrant POSIX et d'autres standards.

Le début des années 1990 vit surgir un rival inattendu : Linux. En 1991, Linus Torvalds, étudiant à l'université d'Helsinki, développa un noyau compatible UNIX capable d'utiliser les outils GNU de la Free Software Foundation. Linux reprenait l'âme d'UNIX en l'adaptant aux architectures modernes. Sa licence libre accéléra sa diffusion et son adoption massive.

UNIX révolutionna l'informatique par ses concepts novateurs. Le système de fichiers hiérarchique, où tout est fichier, simplifia radicalement la gestion des ressources. Les tubes (*pipes*), suggérés par Doug McIlroy en 1972, connectèrent la sortie d'un programme à l'entrée d'un autre, créant des chaînes de traitement d'une puissance inégalée. Le shell offrit un environnement de programmation complet en ligne de commande.

La portabilité d'UNIX, rendue possible par son écriture en C, facilita son adaptation à une multitude d'architectures. Cette caractéristique, alliée à sa conception modulaire et sa documentation exemplaire, explique sa longévité remarquable. Les manuels UNIX, présents dès la première édition, établirent un standard de documentation technique qui fait toujours référence. Sa philosophie de conception imprègne encore le développement logiciel : créer des programmes simples qui excellent dans leur domaine, utiliser le texte comme interface universelle, composer des outils élémentaires pour résoudre des problèmes complexes.

VI

1970

Quand le numérique transforma le monde

Tandis que les bombes pleuvent sur le Vietnam, trois ingénieurs chez Intel achèvent la conception du premier microprocesseur commercial. Leur invention, presque passée inaperçue dans un monde obsédé par la guerre froide, changera tout.

La décennie qui s'ouvre se joue dans un climat particulier. L'Amérique saigne au Vietnam. Les jeunes descendent dans la rue. L'URSS et les États-Unis se toisent. Le pétrole devient une arme. Et au milieu de ce chaos mondial, une poignée d'hommes et de femmes inventent l'avenir numérique.

Les campus américains bouillonnent. La contestation sociale y rejette l'autorité établie, y compris celle des grands ordinateurs centralisés qui symbolisent le pouvoir institutionnel. À Berkeley comme au MIT, des étudiants rêvent d'une informatique différente, plus personnelle, plus accessible. Dans les laboratoires universitaires, on s'affaire autour de machines qui n'ont rien à voir avec les colosses climatisés des entreprises. C'est là, à l'écart des sentiers balisés par IBM, que naît une autre vision de l'ordinateur.

En 1973, le choc pétrolier frappe. Les files s'allongent devant les stations-service. L'inflation s'envole. Face à cette crise, les entreprises cherchent à réduire leurs coûts. L'informatisation apparaît comme une solution. Les banques sont les premières à s'y engouffrer car elles doivent gérer un nombre

croissant de transactions dans un contexte économique instable. Leurs besoins propulsent le développement de bases de données toujours plus sophistiquées. Oracle naît dans ce contexte, sur une idée de Larry Ellison inspirée par un article théorique d'IBM.

À l'Est, derrière le rideau de fer, l'URSS tente de ne pas décrocher. Elle lance son propre plan informatique, copiant parfois les architectures occidentales. Les accords SALT n'y changent rien, la technologie continue d'être un enjeu de puissance. Les ordinateurs soviétiques équipent les usines, les administrations, mais souffrent d'un retard croissant. L'informatique marque visiblement la différence entre les deux blocs.

Le Japon, lui, joue sa propre partition. Sans les contraintes militaires qui pèsent sur l'industrie américaine, les entreprises japonaises se concentrent sur l'électronique grand public. Elles excellent dans la miniaturisation. Sony, Panasonic et Sharp s'imposent partout dans le monde. Leurs composants équipent bientôt les premiers ordinateurs personnels. Le pays, défait en 1945, prend sa revanche technologique.

L'Europe avance en ordre dispersé. La France mise sur son Plan Calcul et la Compagnie Internationale pour l'Informatique pour créer un champion national. L'Allemagne s'appuie sur Siemens. Le Royaume-Uni développe ICL. Mais ces initiatives nationales peinent face aux géants américains. Les universités européennes, en revanche, brillent dans la recherche théorique. Le langage Pascal, créé par Niklaus Wirth à l'École Polytechnique Fédérale de Zurich, marque une avancée dans la conception des langages de programmation.

La télévision règne alors sans partage sur les foyers occidentaux. Les images du premier pas sur la Lune, vues par des centaines de millions de spectateurs, ont démontré la puissance des réseaux de communication. Les ingénieurs télécoms réfléchissent à la transmission de données numériques, notamment avec les premières expériences de vidéotex. En France, les PTT lancent les études qui mèneront au Minitel, premier réseau grand public avant l'heure d'Internet.

Dans les écoles et universités, l'informatique fait ses premiers pas comme discipline d'enseignement. Les langages BASIC et Logo sont conçus avec un objectif pédagogique. Seymour Papert, au MIT, théorise l'apprentissage par la programmation. Ces réflexions ouvrent un nouveau champ : l'informatique est un outil de calcul, mais aussi un moyen d'apprendre autrement.

Le monde du travail se transforme, les écrans à tube cathodique remplacent peu à peu les fiches cartonnées. Dans les bureaux, les secrétaires découvrent le traitement de texte. Les premières imprimantes à marguerite reproduisent la qualité des machines à écrire, mais avec la souplesse de l'électronique, ce qui modifie les métiers administratifs. Les syndicats

s'inquiètent, négocient des accords sur les conditions de travail sur écran, mais la marche vers le tout-numérique est lancée.

La science-fiction s'empare du thème de l'ordinateur. « 2001, l'Odyssée de l'espace » a popularisé l'image de HAL 9000, intelligence artificielle devenue folle. En 1973, le film « Westworld » imagine des robots incontrôlables. Ces œuvres traduisent les espoirs et les craintes suscités par l'informatisation. Simultanément, les premiers jeux vidéo commerciaux comme Pong attirent un public nouveau vers ces machines autrefois réservées aux scientifiques.

La santé n'échappe pas à cette vague. Les hôpitaux s'équipent de systèmes informatiques pour la gestion des patients. Les premiers scanners médicaux, apparus au début de la décennie, produisent des images numériques que seuls des ordinateurs peuvent traiter. La recherche pharmaceutique s'appuie de plus en plus sur la modélisation moléculaire. L'informatique médicale est une nouvelle discipline qui émerge.

Les grandes villes deviennent des laboratoires où s'expérimentent les applications urbaines de l'informatique. La gestion des feux de circulation, l'optimisation des réseaux d'eau ou d'électricité bénéficient des systèmes de contrôle en temps réel. Ces applications exigent des ordinateurs capables de réagir instantanément à leur environnement, stimulant le développement des systèmes embarqués.

Le commerce international se métamorphose avec l'adoption de normes d'échange électronique. Le code-barres fait son apparition dans les supermarchés américains. Les transactions bancaires s'automatisent avec le réseau SWIFT. L'informatisation des chaînes logistiques débute. Ces évolutions accélèrent la mondialisation des décennies suivantes, où les flux d'informations circuleront aussi vite que les marchandises.

Steve Jobs et Steve Wozniak fondent Apple en 1976. Leur Apple I symbolise l'esprit de cette nouvelle informatique : artisanale, personnelle, presque rebelle. Au même moment, Bill Gates et Paul Allen créent Microsoft. Ces jeunes entrepreneurs ne ressemblent en rien aux dirigeants des grandes entreprises informatiques traditionnelles. Ils inaugurent une nouvelle ère où l'ordinateur est un produit de consommation et un objet de passion.

La recherche fondamentale avance à grands pas. Les travaux d'Alan Kay chez Xerox PARC sur les interfaces graphiques permettent une informatique plus intuitive. Les créations de Dennis Ritchie et Ken Thompson aux Bell Labs, le langage C et le système UNIX, initient l'informatique moderne. Ces innovations, d'abord confidentielles, changeront le visage de l'industrie au cours des décennies suivantes.

À la fin des années 1970, les premiers clubs d'informatique amateur ras-

semblent des passionnés autour de machines rudimentaires comme le MITS Altair 8800. Ces communautés développent une culture du partage contrastant avec le monde fermé de l'informatique professionnelle. Les magazines spécialisés publient des programmes que les lecteurs recopient ligne à ligne. Personne n'imagine encore que ces pratiques confidentielles représentent l'informatique du XXI^{e} siècle.

L'ordinateur, autrefois outil réservé aux grandes organisations, amorce sa mue vers un compagnon quotidien durant la décennie 1970. Les idées qui émergeront au cours de cette période façonneront notre monde numérique pour les 50 années suivantes, dessinant une société où le traitement de l'information est aussi essentiel que la production de biens matériels.

ALOHAnet

En septembre 1968 à l'université d'Hawaï, Norman Abramson se trouve face à un problème qui semble presque insurmontable. Comment connecter entre eux des ordinateurs dispersés sur plusieurs îles, séparés par des centaines de kilomètres d'océan ? Les câbles coûtent une fortune, les liaisons téléphoniques sont lentes et peu fiables. Pourtant, l'université a besoin de relier son campus principal de Manoa, près d'Honolulu, aux centres de recherche disséminés sur Oahu, Kauai, Maui et la grande île d'Hawaï.

L'idée qui germe dans son esprit paraît un peu folle : utiliser les ondes radio pour faire communiquer les ordinateurs. En 1968, personne n'avait encore tenté pareille aventure. Les communications informatiques se cantonnaient aux connexions directes par câble, point à point, selon des schémas hérités du télégraphe et du téléphone. Mais Hawaï impose ses propres règles. La géographie volcanique de l'archipel transforme ce qui ailleurs serait une contrainte en laboratoire d'innovation.

Le système qu'imagine Abramson repose sur une architecture d'une simplicité trompeuse. Au cœur du dispositif trône un IBM 360/65 équipé de 750 Ko de mémoire, installé sur le campus principal. Cette machine dialogue avec un ordinateur plus modeste, un HP 2115A baptisé MENEHUNE, du nom des petits êtres de la mythologie hawaïenne réputés pour accomplir des travaux « impossibles » en une nuit. Le MENEHUNE gère les émissions radio grâce à deux canaux UHF de 100 kHz : l'un diffuse à 407,350 MHz vers les terminaux distants, l'autre collecte leurs réponses à 413,475 MHz. Chaque canal fonctionne à 24 000 bauds.

Mais la véritable audace du projet réside ailleurs. Dans le sens descendant, du MENEHUNE vers les terminaux, tout est classique : une seule

source émet, les destinataires écoutent, l'ordonnancement suit des règles de priorité bien établies. C'est dans l'autre sens que l'innovation surgit. Comment gérer les communications quand des dizaines de terminaux veulent tous parler ensemble sur le même canal ? Les techniques traditionnelles de multiplexage imposeraient à chaque terminal une tranche horaire ou fréquentielle fixe, qu'il l'utilise ou non. Un gaspillage inadmissible pour un trafic informatique fait de courtes rafales séparées par de longues pauses.

La solution proposée par l'équipe d'Abramson brise tous les codes établis. Chaque terminal émet quand bon lui semble, sans demander l'autorisation à quiconque. Les données voyagent dans des paquets de taille standardisée : 80 caractères de 8 bits pour l'information utile, plus 64 bits pour l'identification, le contrôle et la détection d'erreurs. Quand le MENEHUNE reçoit un paquet intact, il envoie un accusé de réception. Sinon l'émetteur attend un délai aléatoire et recommence. Cette approche accepte l'inacceptable : les collisions entre paquets. Deux terminaux qui émettent simultanément voient leurs signaux se mélanger, créant un charabia incompréhensible pour le récepteur. Plutôt que de chercher à éviter ces collisions à tout prix, ALOHA les considère comme un mal nécessaire, le prix à payer pour une simplicité de fonctionnement inégalée.

Les calculs mathématiques d'Abramson révèlent les limites de cette philosophie. Le système exploite au mieux que 18,4% de la capacité théorique du canal, performance atteinte quand la charge offerte représente exactement la moitié de cette capacité. Au-delà, les collisions se multiplient, les retransmissions s'accumulent, et le réseau s'effondre dans un chaos de signaux parasites.

ALOHA prouve qu'un système peut fonctionner sans chef d'orchestre, sans coordination centrale, sans planning préétabli. Les terminaux se débrouillent entre eux, s'accommodent des conflits, trouvent leur équilibre dans ce qui ressemble à une anarchie organisée.

En 1973, un jeune ingénieur de Xerox nommé Robert Metcalfe visite l'installation. Il comprend que les principes d'ALOHA peuvent s'adapter à d'autres supports que les ondes radio. Quelques années plus tard naîtra Ethernet, transposition filaire des idées d'Abramson qui révolutionnera les réseaux locaux.

Entre-temps, les chercheurs d'Hawaï affinent leur création. Ils inventent « ALOHA discrétisé », variante où les émissions ne peuvent débuter qu'à des instants prédéfinis, comme les coups d'un métronome invisible. Cette synchronisation double l'efficacité théorique, portant le rendement maximal à 37% de la capacité du canal. L'amélioration ouvre la voie à une famille entière de protocoles dérivés.

Le succès du projet illustre une vérité sur l'innovation technologique.

Souvent, les contraintes les plus sévères engendrent les solutions les plus créatives. Hawaï ne pouvait pas se contenter de recopier les recettes continentales. L'isolement géographique força les chercheurs à repenser de fond en comble les communications informatiques. Ce faisant, ils découvrirent des principes universels qui dépassaient leur situation particulière.

Les réseaux Wi-Fi, les communications satellitaires, les systèmes cellulaires, tous incorporent des mécanismes inspirés du protocole hawaïen. L'idée qu'un réseau puisse fonctionner sans contrôle central strict, en gérant les conflits plutôt qu'en les évitant, est devenue un paradigme de l'informatique distribuée. Quarante ans après sa création, ALOHA conserve sa force d'inspiration. Non pas tant par ses performances brutes, largement dépassées par les standards actuels, que par l'audace conceptuelle dont il témoigne. Parfois, il suffit de laisser les machines se débrouiller entre elles.

FORTH

Charles « Chuck » Moore travaillait au MIT, puis au Stanford Linear Accelerator Center. L'informatique de l'époque le frustrait car FORTRAN et ALGOL monopolisaient l'attention, mais ces langages l'obligeaient à passer un temps fou entre l'écriture du code, sa compilation et son exécution. Il cherchait quelque chose de plus direct, de plus immédiat.

Sa solution prit la forme d'un interpréteur de texte qu'il écrivit en ALGOL. Le principe était que tout ce qu'on lui donnait était soit un nombre, soit un mot représentant une commande. Fini les phases séparées d'édition, compilation et exécution. Tout se passait de manière interactive, dans l'instant. Il inventa des mots spéciaux comme « : » pour dire au système qu'il fallait commencer à compiler le code qui suivait, et « ; » pour lui dire de revenir au mode interprétation normal. Quand le compilateur rencontrait un tel bloc, il stockait ce « mot » – l'équivalent de nos fonctions actuelles – dans un dictionnaire pour pouvoir le réutiliser plus tard.

L'architecture de FORTH repose sur deux piles qui travaillent ensemble. La première stocke les valeurs temporaires et les paramètres, la seconde gère les cadres de pile. Cette approche donne un code compact et rapide, mais surtout extensible à volonté. En 1970, Moore travaillait sur un ordinateur IBM de troisième génération qui limitait les noms à cinq caractères maximum. Il baptisa donc son langage « FORTH », allusion en anglais à une quatrième génération de langage de programmation.

L'Observatoire National de Radioastronomie en Arizona accueillit la

première version publique de FORTH en 1971. Le système servait à récupérer et analyser les données tout en pilotant le radiotélescope en temps réel. La communauté astronomique adopta cette technologie. Moore créa ensuite plusieurs versions : MiniFORTH pour les mini-ordinateurs, MicroFORTH pour les microcontrôleurs. Chaque variante s'adaptait aux contraintes spécifiques de son environnement.

FORTH utilise la notation postfixe, aussi appelée polonaise inverse. Les opérateurs viennent après leurs opérandes. L'expression `1 2 +` signifie qu'on additionne 1 et 2. Cependant le fonctionnement est interactif : quand l'utilisateur tape cette expression, l'interpréteur pousse d'abord 1 sur la pile, ensuite 2, avant d'exécuter « + » qui récupère ces deux valeurs, les additionne et remet le résultat sur la pile.

La vraie force de FORTH vient de sa capacité à grandir. Les programmeurs créent de nouvelles commandes, des « mots », qui deviennent partie intégrante du langage. Cette extensibilité adapte FORTH aux besoins précis de chaque application. Le langage mélange des calculs de bas niveau avec un mécanisme puissant pour déclarer des fonctions. Une fois compilés, les programmes FORTH tournent presque aussi vite que de l'assembleur écrit à la main. Avec un compilateur qui optimise agressivement, FORTH est très performant dans le traitement numérique. Pour les systèmes embarqués, le matériel prototype ou les chargeurs d'amorçage, il suffit d'un noyau compilateur minimal. Une fois ce noyau disponible, on peut construire un compilateur FORTH complet en n'utilisant que FORTH.

La philosophie minimaliste de FORTH se retrouve dans la gestion de la mémoire. Le langage ne propose que le stockage de valeurs simples ou doubles, généralement des entiers. Pour construire un tableau, il faut allouer une variable et incrémenter un pointeur interne dans le dictionnaire afin de réserver plus d'espace. Un tableau en FORTH n'est qu'une région de mémoire sans documentation particulière. Le langage n'offre pas de tas (*heap*) natif non plus, les programmeurs écrivent leurs propres routines de gestion mémoire.

Les applications concrètes de FORTH se multiplient. Sun Microsystems l'intègre dans le firmware OpenBoot de ses stations de travail. WearLogic a développé un portefeuille électronique basé sur un microcontrôleur AVR programmé en FORTH. AEMS (Advanced Energy Monitoring Systems) emploie FORTH pour son Yatesmeter, un instrument sophistiqué qui surveille les pompes en temps réel. Ces réussites montrent que FORTH garde sa pertinence pour les systèmes embarqués et le contrôle matériel.

L'évolution technologique n'a pas affaibli l'intérêt pour FORTH. Son approche interactive et sa capacité à construire des abstractions par étapes en font un outil pédagogique de valeur. Des projets comme pbFORTH pour

les robots LEGO Mindstorms révèlent son potentiel pour enseigner la programmation. Sa syntaxe particulière et son modèle mental différent des langages conventionnels offrent aux étudiants une autre perspective sur les concepts de programmation.

La communauté FORTH est active. Des groupes d'utilisateurs existent encore dans plusieurs pays, des conférences comme euroFORTH se tiennent régulièrement. Les développements récents incluent GNU Gforth, une version portable qui respecte les standards, et VFX Forth, un compilateur optimisant moderne qui accélère les programmes FORTH d'un facteur trois à cinq par rapport aux compilateurs classiques.

Plutôt que d'imposer une abstraction entre l'homme et la machine, FORTH crée un lien direct avec le matériel tout en autorisant la construction d'abstractions utiles.

Pascal

Niklaus Wirth travaille à l'École Polytechnique Fédérale de Zurich. Une frustration le saisit face aux langages de programmation existants. Ces outils lui semblent incohérents, leurs constructions défient toute explication logique. Pour un esprit habitué au raisonnement systématique, méthodique et ordonné, cette situation est insupportable.

Il conçoit Pascal avec deux ambitions distinctes. Il veut d'abord créer un langage vraiment adapté à l'enseignement, où chaque concept se reflète naturellement dans la syntaxe. Mais il refuse de sacrifier l'efficacité sur l'autel de la pédagogie : ses installations devront tourner de manière fiable sur les machines de l'époque.

L'héritage d'Algol 60 pèse lourd dans cette démarche. Ce langage répond mieux que tout autre aux exigences pédagogiques du moment. Wirth reprend ses principes de structuration et la forme de ses expressions. Pourtant, il résiste à la tentation de faire de Pascal un simple sous-ensemble d'Algol 60, car certains mécanismes de déclaration l'empêcheraient d'intégrer les nouvelles fonctionnalités qu'il a en tête. Ces fonctionnalités concernent principalement les structures de données, dont l'absence dans Algol 60 explique largement pourquoi ce langage est cantonné à un domaine d'application restreint. En introduisant les enregistrements et les fichiers, Wirth espère que Pascal pourra s'attaquer aux problèmes de gestion commerciale, ou du moins servir à les démontrer efficacement dans un cours de programmation.

La première version paraît en 1970, accompagnée d'une installation sur

les ordinateurs CDC 6000. Trois années d'utilisation intensive, d'enseignement et d'expérimentation donnent naissance à une version révisée en 1972. Les changements reflètent les leçons apprises sur le terrain : les paramètres constants cèdent la place aux paramètres de valeur, la structure de classe disparaît, la gestion des fichiers se trouve repensée.

La normalisation arrive en 1983 avec le standard ISO 7185, qui définit le Pascal non étendu. Une mise à jour a lieu en 1990, l'année où naît le standard ISO 10206, le Pascal étendu. Cette distinction répond à la tension qui consiste à garder la simplicité du langage original tout en répondant aux besoins grandissants de la programmation moderne.

Le Pascal étendu apporte une séparation entre la modularité et la compilation. Chaque module permet d'exporter des interfaces contenant valeurs, types, schémas, variables, procédures et fonctions. Le système contrôle finement la visibilité, et les chaînes de caractères reçoivent un traitement unifié. Chaînes de longueur fixe, valeurs de caractères et chaînes de longueur variable sont compatibles entre elles. L'opérateur de concaténation les combine librement, tandis que le programmeur spécifie les longueurs maximales des chaînes variables. La liaison de variables constitue une innovation remarquable. Une variable déclarée comme liaison peut se connecter au stockage de fichiers, à l'horloge temps réel ou aux lignes de commande. Cette restriction limite les connexions externes aux seules variables explicitement prévues à cet effet.

L'émergence de la programmation orientée objet pousse certains compilateurs Pascal vers ce nouveau paradigme. En 1993, le comité des standards Pascal publie un rapport technique sur les « Extensions orientées objet de Pascal ». Les membres de ce comité représentent un éventail surprenant d'organisations : de l'université Pace à l'US Air Force, d'Apple Computer à Microsoft et Digital Equipment Corporation.

Pascal marque l'histoire de la programmation par sa syntaxe limpide et sa philosophie du typage fort. Toute une génération de programmeurs découvre à travers lui les principes de la programmation structurée et de la conception rigoureuse. Son usage massif dans l'enseignement durant les années 1970 et 1980 forge des habitudes de programmation disciplinées qui lui survivront longtemps. Le succès commercial accompagne le succès pédagogique, notamment grâce à Turbo Pascal de Borland. Ces versions commerciales enrichissent souvent le langage standard d'extensions pratiques, sans jamais trahir la clarté et la rigueur qui caractérisent l'original.

Le langage Pascal porte l'idée qu'un programme doit être une construction mathématique précise, dont la correction se démontre formellement. Cette vision contraste avec les tendances actuelles qui privilégient la rapidité de développement et la flexibilité, parfois au détriment de la rigueur

formelle, et donc parfois de la sécurité. Wirth avait peut-être vu juste : dans un monde où les logiciels gouvernent nos vies, la discipline qu'il prônait n'a jamais été aussi nécessaire.

DEC PDP-11

En 1970, Digital Equipment Corporation bouleverse le monde des mini-ordinateurs avec le PDP-11. Cette machine marque une étape décisive dans l'évolution informatique, non pas par sa puissance brute, mais par une vision architecturale : l'UNIBUS. Ce bus unifié connecte processeur, mémoire et périphériques sur une interface commune, là où les machines précédentes multipliaient les connexions spécialisées. L'innovation paraît technique, elle transforme pourtant radicalement la conception des systèmes informatiques.

L'histoire du PDP-11 débute vers 1968 dans les laboratoires de DEC. Gordon Bell, Harold McFarland et Roger Cady dirigent une équipe confrontée aux limites des mini-ordinateurs de l'époque. Espace d'adressage étriqué, registres en nombre insuffisant, absence de pile matérielle, gestion approximative des interruptions... Ces défauts handicapent le développement d'applications sophistiquées, alors les ingénieurs explorent différentes pistes. Un prototype appelé PDP-X voit le jour, suivi d'une version DCM (Desk Calculator Machine). Ces expérimentations convergent finalement vers l'architecture définitive du PDP-11.

Le résultat dépasse les attentes. Le PDP-11 propose un jeu d'instructions orthogonal, les mêmes modes d'adressage fonctionnent avec toutes les instructions. Ses huit registres généraux offrent une souplesse inédite, l'un servant de pointeur de pile, l'autre de compteur de programme. L'adressage direct des octets, rarissime, facilite le traitement des caractères et des données de longueur variable. Ces innovations techniques se traduisent par une programmation plus naturelle et plus efficace.

Le marché répond avec enthousiasme. Entre 1970 et 1975, DEC écoule plus de 20 000 exemplaires du PDP-11. Ce succès s'appuie sur la modularité de l'architecture qui autorise une gamme étendue du petit système à quelques milliers de dollars jusqu'aux configurations à plusieurs centaines de milliers de dollars. Les progrès technologiques accompagnent cette montée en puissance. Les premiers modèles s'appuient sur des mémoires à tores magnétiques, bientôt remplacées par les mémoires à semi-conducteurs MOS, puis par les mémoires cache rapides en technologie bipolaire.

La famille s'étoffe. Le modèle initial PDP-11/20 cède la place au 11/45 en 1972, machine plus robuste qui gagne en performances. Le 11/70 arrive en 1975 avec sa mémoire cache et sa capacité d'adressage étendue à 4 mégaoctets. À l'opposé, le LSI-11 de 1975 exploite les circuits intégrés à large échelle pour miniaturiser et réduire les coûts. De cette diversité, le PDP-11 va conquérir des territoires variés tels que le calcul scientifique, la gestion d'entreprise, le contrôle de processus industriels ou les communications.

DEC comprend qu'un ordinateur sans système d'exploitation est une coquille vide. L'entreprise développe plusieurs environnements logiciels adaptés aux besoins spécifiques : RT-11 pour les applications temps réel simples, RSX-11M pour le multitâche temps réel complexe, RSTS/E pour le temps partagé interactif. Ces systèmes matures, accompagnés d'outils de développement de qualité, facilitent l'adoption du PDP-11 par une clientèle diverse.

L'introduction de DECnet en 1975 interconnecte les PDP-11 entre eux et avec d'autres machines DEC selon une approche distribuée qui tranche avec la philosophie hiérarchique d'IBM. DECnet déploie une architecture réseau complète, de la couche physique aux applications. Ses évolutions successives enrichissent ses capacités : la Phase II de 1978 établit des passerelles vers les VAX, la Phase III introduit le routage adaptatif.

L'architecture du PDP-11 inspire directement les processeurs futurs comme le Motorola 68000. La concurrence reprend UNIBUS comme un standard de facto. L'organisation en registres généraux et les modes d'adressage sophistiqués du PDP-11 imprègnent la conception des processeurs modernes. Des mécanismes comme l'adressage relatif au compteur de programme ou l'auto-incrément/auto-décrément se généralisent dans l'industrie.

Le PDP-11 trouve aussi sa place dans l'histoire d'UNIX. Ken Thompson et Dennis Ritchie choisissent cette machine chez Bell Labs pour développer leur système d'exploitation. La clarté et la cohérence de l'architecture du PDP-11 en font un excellent support pédagogique. De nombreux manuels universitaires l'utilisent comme référence, formant ainsi toute une génération d'informaticiens à ses concepts.

Mais le temps révèle progressivement les faiblesses de cette architecture. L'espace d'adressage de 16 bits limite la mémoire à 64 kilo-octets par programme, contrainte qui pèse de plus en plus lourd avec la sophistication croissante des applications. DEC ajoute des mécanismes de gestion mémoire pour contourner cette limite, au prix d'une complexité accrue. La prolifération des variantes du jeu d'instructions, notamment les différentes options de calcul flottant, complique la portabilité des logiciels.

Le déclin s'amorce dans les années 1980. Les micro-ordinateurs 16 bits gagnent en puissance tout en restant beaucoup moins chers. Surtout, DEC

lance le VAX qui offre un espace d'adressage 32 bits et relègue le PDP-11 au second plan. La production se maintient néanmoins jusqu'aux années 1990 pour des applications spécialisées qui n'exigent pas les capacités du VAX.

Le PDP-11 aura marqué son temps. Il établit DEC comme un acteur majeur de l'industrie informatique, et démontre qu'une architecture bien pensée est à même de transformer un marché et influencer l'évolution technologique.

B-tree

Quand on évoque l'informatique moderne, certaines innovations restent dans l'ombre malgré leur omniprésence. L'histoire du B-tree commence à la fin des années 1960. Les ordinateurs jonglent déjà avec des volumes de données considérables, mais l'accès à ces informations stockées sur disques magnétiques pose un véritable casse-tête. Chaque lecture nécessite des mouvements mécaniques coûteux en temps, et les programmeurs cherchent désespérément des moyens d'organiser leurs index pour limiter ces accès. Chez Sperry Univac, Howard Chiat et Meyer Schwartz planchent sur ce défi en collaboration avec l'université Case Western Reserve. Ils ne sont pas seuls : Bruce Cole, Stewart Radcliffe et Michael Kaufman mènent des recherches parallèles chez Control Data Corporation, épaulés par l'université Stanford.

Ces travaux dispersés trouvent leur aboutissement théorique grâce à Rudolf Bayer et Edward McCreight. Aux Boeing Scientific Research Labs, ces deux chercheurs formalisent les concepts émergents et publient en 1972 leur article *Organization and maintenance of large ordered indices*. Ce texte pose les bases mathématiques du B-tree et révolutionne la gestion des index.

Le génie du B-tree réside dans sa simplicité conceptuelle. Là où les arbres binaires classiques limitent chaque nœud à deux fils, le B-tree autorise des nœuds plus larges contenant plusieurs clés. Cette liberté architecturale transforme radicalement l'efficacité des opérations. L'arbre maintient automatiquement son équilibre lors des insertions et suppressions, garantissant une hauteur logarithmique qui minimise les accès disque. Imaginez un index téléphonique intelligent qui se réorganise constamment pour que chaque recherche nécessite un nombre d'étapes identique, quelle que soit la taille de l'annuaire.

D'autres variantes émergent les années suivantes. Donald Knuth propose le B+-tree, une modification astucieuse qui concentre toutes les données

dans les feuilles de l'arbre. Les nœuds internes ne servent plus qu'à l'indexation, simplifiant considérablement le parcours séquentiel des données. Cette variante devient la référence dans les systèmes de gestion de bases de données. En 1977, Bayer s'associe à K. Unterauer pour créer le Prefix B+-tree. Leur innovation consiste à ne stocker que les préfixes nécessaires pour distinguer les entrées, économisant l'espace et augmentant le facteur de branchement.

L'adoption industrielle suit naturellement. IBM intègre les B-trees dans VSAM (Virtual Storage Access Method), démontrant leur viabilité commerciale. Le système exploite une variante du B+-tree et introduit des optimisations comme la réplication des nœuds de séquence sur les cylindres du disque. Cette implémentation concrète prouve que la théorie permet effectivement améliorer les performances en production.

Les besoins du multi-utilisateur émergent, et la nécessité pour plusieurs programmes de modifier simultanément la même structure sans la corrompre. Bayer et Mario Schkolnick résolvent cette équation en 1977 avec des protocoles de verrouillage sophistiqués. Leur système autorise plusieurs lecteurs simultanés tout en gérant proprement les modifications concurrentes, une prouesse technique dont les bases de données modernes héritent.

La recherche théorique accompagne ces développements pratiques. Raymond A. Miller étudie en 1977 la construction d'arbres optimaux, tandis qu'Andrew Chi-Chih Yao analyse en 1978 le comportement probabiliste des nœuds. Ses calculs révèlent un taux d'occupation moyen de 69%, un chiffre qui aide les concepteurs à dimensionner leurs systèmes.

L'influence des B-trees dépasse leurs origines. MySQL les exploite dans son moteur InnoDB, PostgreSQL s'en sert pour ses index, et de nombreux systèmes de fichiers comme NTFS, HFS+ ou Ext4 organisent leurs données selon ces principes. Cette ubiquité témoigne de leur robustesse conceptuelle.

Les années 1990 et 2000 apportent de nouveaux problèmes. Les architectures multicœurs et les hiérarchies mémoire complexes exigent des adaptations. Les chercheurs développent des variantes optimisées pour les caches processeur et créent le Bε-tree spécifiquement pour les disques SSD. Ces évolutions montrent comment une idée est capable de s'adapter aux contraintes technologiques émergentes.

Le Big Data et le traitement temps réel poussent encore plus loin les limites. De nouvelles versions améliorent le parallélisme et l'extensibilité, tandis que les recherches actuelles explorent l'optimisation pour les mémoires non volatiles et les architectures distribuées.

Plus de cinquante ans après sa formalisation, le B-tree continue de structurer nos données quotidiennes. Des smartphones aux centres de données, cette invention discrète mais essentielle organise silencieusement l'informa-

tion numérique.

Disquette

En 1967 chez IBM, David Noble se voit confier la résolution d'un problème apparemment banal : trouver un moyen de charger les programmes dans les nouveaux ordinateurs System/370 au démarrage. Les mémoires à semi-conducteurs qui équipent ces machines perdent leur contenu dès qu'on les éteint, contrairement aux anciennes mémoires à tores magnétiques. Il faut donc un support externe fiable.

L'équipe de Noble, qui rassemble Alan F. Shugart, Herbert Thompson, Ralph Flores et Warren L. Dalziel, commence par explorer les cartouches de bandes magnétiques. Mais Noble a une autre idée : pourquoi ne pas utiliser un disque souple ? Les premiers essais frisent l'amateurisme. L'équipe achète un lecteur de disques vinyle 45 tours dans un magasin et y bricole une tête de lecture magnétique montée sur le bras. Le disque souple repose sur une platine garnie de mousse.

Ces expérimentations révèlent vite un cauchemar car la moindre poussière, le moindre cheveu qui traîne provoque des pertes de données. Pire, ces impuretés « rampent » littéralement sur le disque, poussées par la tête de lecture. L'idée géniale naît d'une soirée de travail entre Herbert Thompson et Ralph Flores. Frustrés par l'approche du tourne-disque, ils abandonnent cette voie et filent dans une épicerie acheter du papier absorbant rose, celui qu'on utilise pour faire le ménage.

En découpant un dossier de classement à la bonne dimension et en y collant ce papier rose, ils créent la première enveloppe protectrice. Ils y pratiquent des ouvertures pour la tête de lecture et le système de fixation centrale. Les problèmes de poussière disparaissent d'un coup. Ce papier rose cédera plus tard sa place à un matériau blanc spécialisé, mais le principe restera inchangé pendant des décennies.

Le premier lecteur de disquettes IBM, baptisé « Minnow », débarque sur le marché en 1971. Il ne sait que lire et équipe les contrôleurs du disque dur IBM 3330 ainsi que certains modèles du System/370. L'année suivante, Memorex frappe un grand coup avec le 651, premier lecteur OEM capable de lire et d'écrire. Wang Laboratories et DEC se l'arrachent pour remplacer leurs vieillissants lecteurs de bandes perforées.

Le véritable tournant arrive en 1973 quand IBM lance son système de saisie de données 3740 équipé du lecteur 33FD. Une seule disquette est

désormais en mesure de stocker l'équivalent de 3 000 cartes perforées, réduisant drastiquement la consommation de papier. Sa capacité de stockage de 80 kilo-octets révolutionne les habitudes. L'industrie adopte ce format comme standard de facto. Shugart Associates, la société fondée par Alan Shugart, prend la tête du marché des lecteurs compatibles.

En 1975, le marché s'organise autour de créneaux bien définis. La saisie de données représente 12 500 unités vendues, les terminaux intelligents 10 200, les mini-ordinateurs 3 100, le traitement de texte 800 unités, sans compter d'autres applications émergentes. Un lecteur coûte environ 380 dollars, une somme conséquente.

L'année 1976 marque l'arrivée du format 5,25 pouces, né d'une conversation informelle chez Shugart Associates. Les clients se plaignent de l'encombrement des lecteurs 8 pouces qui pèsent 17 kg chacun. L'équipe développe en six semaines un prototype aux dimensions d'une serviette de cocktail. Cette innovation conserve les performances techniques du format 8 pouces dans un boîtier adapté aux ordinateurs de bureau qui commencent à émerger.

IBM riposte en 1977 avec le modèle 53FD, premier lecteur double-face qui double la capacité tout en gardant la compatibilité avec les disques simple face. L'adaptation de cette technologie se révèle un parcours du combattant. Shugart Associates manque de faire faillite avec ses têtes de lecture double face inspirées des disques durs Winchester. C'est finalement Tandon qui trouve la solution viable avec une tête fixe et une tête mobile.

Cette année 1977 voit aussi naître l'Apple II, premier ordinateur personnel produit en masse équipé de lecteurs de disquettes au format 5,25 pouces. Les ordinateurs personnels adoptent cette configuration comme une norme qui contribue à standardiser les supports de stockage. L'industrie du logiciel peut enfin décoller.

La genèse du format 5,25 pouces révèle des anecdotes savoureuses. Wang Laboratories rêvait d'un ordinateur de bureau avec un lecteur à 100 dollars. Steve Jobs, inconnu du grand public, débarquait régulièrement chez Shugart Associates pour obtenir un lecteur abordable destiné à remplacer les cassettes audio qu'Apple utilisait pour stocker ses données. L'équipe de Don Massaro et Jimmy Adkisson choisit délibérément la dimension de 5,25 pouces car elle représentait la plus petite taille qui ne permettait pas de glisser la disquette dans une poche, évitant ainsi les dommages par négligence.

Le format 8 pouces connaît son âge d'or au début des années 1980 avec 1,5 million d'unités produites annuellement. L'industrie rassemble 25 fabricants américains, 11 japonais et 8 européens. Les marges s'effritent face à une concurrence féroce, passant de 55% à 40%. Les fabricants japonais,

obsédés par la qualité de production, s'imposent progressivement sur le marché mondial.

Le déclin du format 8 pouces s'amorce avec le succès du 5,25 pouces, mais la production se maintient jusqu'à la fin des années 1990. IBM garde des contrats avec YE Data au Japon pour livrer 5 000 à 10 000 lecteurs par an, uniquement destinés à la maintenance d'anciens systèmes encore sous contrat.

Les années 1980 voient cohabiter sur la plupart des ordinateurs personnels un lecteur 5,25 pouces et un lecteur 3,5 pouces. Ce dernier format finit par s'imposer comme nouveau standard. Les années 1990 marquent l'apogée de la disquette 3,5 pouces avec plus de cinq milliards d'unités écoulées chaque année. Mais l'arrivée des CD réinscriptibles, des DVD puis des clés USB sonne le glas de cette technologie. En 2011, Sony, dernier fabricant mondial, tire le rideau sur la production de disquettes.

L'épopée de la disquette raconte en filigrane la transformation complète des supports de stockage informatique. D'un simple outil de chargement de programmes, elle se mue en un périphérique polyvalent qui annonce le stockage amovible moderne. Sans elle, ni l'industrie du logiciel ni la démocratisation de l'informatique personnelle n'auraient pris l'ampleur qu'on leur connaît.

ed

Certains outils marquent leur époque par leur simplicité radicale plutôt que par leur sophistication. L'éditeur `ed` appartient à cette catégorie d'innovations qui, nées aux Bell Labs au début des années 1970, ont façonné la manière dont nous concevons l'édition de texte. Son développement répond aux contraintes matérielles des terminaux télétype qui ne savaient qu'imprimer une ligne à la fois sur papier, mais sa conception révèle une philosophie qui dépasse ces limitations techniques.

Le génie d'`ed` réside dans sa capacité à transformer l'édition de texte en un dialogue entre l'utilisateur et la machine. Contrairement aux éditeurs modernes qui exposent l'intégralité du document, `ed` oblige à penser le texte comme une séquence de lignes numérotées, stockées dans un espace mémoire appelé *buffer*. Loin d'être une faiblesse, cette abstraction est une force qui contraint l'utilisateur à développer une représentation mentale précise de la structure de son document.

La syntaxe d'`ed` se distingue par sa concision. Les commandes `a`, `i`, `d`,

`s` et `p` suffisent à couvrir la plupart des opérations d'édition. Derrière cette apparente simplicité se cache un système d'adressage d'une richesse remarquable. L'utilisateur est apte à désigner la ligne courante par un simple point, la dernière ligne par un dollar, ou spécifier des plages entières grâce aux expressions régulières. Cette approche révolutionne la manipulation de texte en introduisant une grammaire formelle pour décrire les motifs textuels.

Les expressions régulières constituent peut-être l'héritage le plus durable d'`ed`. Ces motifs, qui décrivent des structures de texte plutôt que des chaînes littérales, transforment la recherche et la modification en opérations logiques. Le point qui représente n'importe quel caractère, l'astérisque qui quantifie les répétitions, les crochets qui définissent des ensembles, autant de concepts qui inspireront plus tard les langages de programmation et les outils d'analyse textuelle. La commande de substitution `s` illustre cette puissance puisqu'elle peut transformer des documents entiers en une seule instruction, pourvu que l'utilisateur maîtrise le langage des motifs.

Cette philosophie de l'édition trouve son expression la plus pure dans l'interface minimaliste d'`ed`. Face à une erreur, l'éditeur se contente d'afficher un laconique `?`. Cette parcimonie, qui désarçonne les débutants, reflète une conception où l'outil doit disparaître derrière la tâche. Les créateurs d'`ed` partaient du principe que l'efficacité naît de la maîtrise, et que la verbosité nuit à la concentration.

Les limites techniques transparaissent dans certaines contraintes, celle des 512 caractères maximum par ligne, ou des 256 caractères pour les commandes globales. Ces restrictions, qui semblent dérisoires aujourd'hui, témoignent de l'ingéniosité nécessaire pour créer des outils puissants avec des ressources limitées. Les programmeurs devaient penser chaque octet, optimiser chaque algorithme.

L'influence d'`ed` dépasse le domaine de l'édition de texte. L'utilitaire `grep` tire son nom de la commande `g/re/p` (recherche globale d'expression régulière et impression), transformant une fonctionnalité d'`ed` en outil autonome. L'éditeur `sed` étend la logique d'`ed` au traitement de flux, tandis que `vi` conserve ses concepts fondamentaux en les adaptant à l'édition visuelle. Cette généalogie révèle comment les innovations se propagent et évoluent dans l'écosystème UNIX.

`ed` subsiste dans tous les systèmes UNIX et Linux. Sa présence n'est plus anecdotique dans les environnements contraints, lors de connexions réseau limitées, ou dans les scripts d'administration, ses qualités originelles retrouvent leur pertinence. L'éditeur qui semblait destiné aux musées de l'informatique se révèle parfois l'outil le plus adapté à certaines situations.

La documentation originale d'`ed`, rédigée par Brian W. Kernighan, fixe

les standards de la documentation technique UNIX. Elle combine rigueur théorique et pragmatisme, expliquant ce que fait l'outil et pourquoi il le fait ainsi. Cette approche pédagogique, où chaque concept s'appuie sur des exemples concrets, influence encore la manière dont nous documentons les logiciels.

File Transfer Protocol

En 1971, Internet ressemblait à un terrain technologique où chaque opération simple exigeait de maîtriser des commandes hermétiques. L'idée d'échanger des fichiers entre machines distantes relevait du défi technique. C'est alors que naît le File Transfer Protocol, plus connu sous l'acronyme FTP, avec la publication du RFC 114 par l'Internet Engineering Task Force. Ce document décrit la première version d'un protocole qui va traverser les décennies.

Initialement, le RFC 114 d'avril 1971 proposait une architecture simple à une seule connexion full-duplex pour véhiculer à la fois les commandes et les données. Cette approche directe reflétait l'état des réseaux informatiques. Mais les concepteurs du protocole allaient comprendre les limites de cette approche. Au fil des révisions successives dans les années 1970, une innovation majeure émerge avec la séparation complète du canal de contrôle et du canal de données. Cette architecture, formalisée dans le RFC 765 de juin 1980 puis perfectionnée dans le RFC 959 d'octobre 1985, établit deux connexions TCP distinctes. La première, sur le port 21, sert uniquement aux échanges de commandes entre le client et le serveur. La seconde, traditionnellement sur le port 20, se consacre au transfert effectif des fichiers. Cette évolution, en apparence anodine, cache une intelligence remarquable. Si le transfert plante, la connexion de contrôle reste active et les messages d'erreur continuent de circuler.

Le protocole se répand dans les universités américaines. Les étudiants découvrent qu'ils peuvent déposer leurs devoirs sur des serveurs distants, les développeurs distribuent leurs logiciels, et les administrateurs système synchronisent leurs données entre machines. FTP rejoint les piliers d'Internet aux côtés du courrier électronique et de l'accès distant.

Mais FTP cache des subtilités. Le protocole propose deux modes de fonctionnement qui reflètent l'évolution des architectures réseau. Le mode actif, implémenté en premier, fait que le serveur initie la connexion de données vers le client. Cette approche fonctionnait parfaitement dans les

premiers temps d'Internet, quand les machines étaient directement connectées au réseau. L'arrivée des pare-feux et de la translation d'adresses réseau a tout changé. Comment un serveur peut-il établir une connexion vers un client caché derrière un NAT ? Le mode passif inverse la logique : c'est le client qui établit les deux connexions. Cette évolution illustre parfaitement comment les protocoles s'adaptent aux contraintes techniques émergentes.

La distinction entre mode ASCII et mode binaire trahit l'héritage des premiers systèmes informatiques. UNIX terminait ses lignes par un caractère, Windows par deux, et Mac par un autre encore. Le mode ASCII de FTP convertit automatiquement ces marqueurs selon le système destinataire. Le mode binaire, lui, copie le fichier bit par bit sans aucune transformation. Bien qu'archaïque en apparence aujourd'hui, cette différenciation témoigne de la diversité des systèmes que FTP devait relier.

L'authentification anonyme constitue l'une des innovations les plus démocratiques de FTP. Plutôt que d'obliger chaque utilisateur à créer un compte, le protocole accepte le nom d'utilisateur « anonymous » avec une adresse électronique en guise de mot de passe. Cette fonctionnalité a révolutionné la distribution de logiciels libres et de documents publics. Parallèlement, FTP supporte l'authentification classique par nom d'utilisateur et mot de passe pour les accès restreints.

La sécurité s'impose progressivement comme un enjeu critique. FTP, dans sa version originale, transmet tout en clair : identifiants, commandes, et données. Un espion placé sur le réseau est susceptible de capturer les mots de passe ou le contenu des fichiers échangés. Cette faiblesse stimule le développement d'extensions sécurisées. FTPS encapsule FTP dans SSL ou TLS, soit de manière implicite sur le port 990, soit de manière explicite en négociant le chiffrement sur une connexion standard. SFTP, malgré son nom, constitue en réalité un protocole distinct basé sur SSH qui offre une alternative moderne et sécurisée.

L'intégration de FTP dans les systèmes d'exploitation révèle son statut de protocole fondamental. UNIX intègre nativement des clients et serveurs en ligne de commande. Windows incorpore le support FTP dans son explorateur de fichiers, démocratisant l'accès au protocole. Des logiciels comme FileZilla enrichissent l'expérience utilisateur avec des interfaces graphiques intuitives tout en respectant scrupuleusement les standards.

La reprise de transfert après interruption transforme l'expérience utilisateur : plus besoin de recommencer entièrement le téléchargement d'un fichier de plusieurs gigaoctets après une coupure réseau. Le support IPv6 et les extensions pour les fichiers volumineux modernisent le protocole. Chaque évolution préserve la compatibilité avec l'existant, principe cardinal de la conception des protocoles Internet.

Les failles de sécurité de FTP sensibilisent la communauté technique à l'importance du chiffrement. Les complications du mode actif avec les pare-feux alimentent la réflexion sur l'architecture des protocoles réseau. La gestion de deux connexions simultanées enrichit la compréhension des défis de sécurisation des communications.

L'émergence du stockage *cloud* et des protocoles web modernes grignote progressivement les parts de marché de FTP. HTTPS s'impose pour de nombreux usages grand public grâce à sa simplicité et sa sécurité intégrée. Pourtant, FTP conserve sa pertinence dans certains contextes professionnels, particulièrement pour l'administration de serveurs web et la maintenance de sites Internet.

Cette longévité s'explique par certaines qualités, dont la simplicité du protocole, sa robustesse éprouvée, et sa capacité d'évolution sans rupture de compatibilité. La séparation entre canal de contrôle et canal de données, innovation des années 1970, se révèle précieuse pour la gestion d'erreurs et la reprise de transferts. Les codes de réponse numériques standardisés facilitent l'automatisation et la gestion programmatique. Ces évolutions prouvent qu'un protocole quinquagénaire sait s'adapter aux exigences modernes. FTP illustre parfaitement qu'en informatique, la valeur d'une technologie ne se mesure pas seulement à sa nouveauté, mais à sa capacité d'évolution et à la solidité de ses fondations.

Intel 4004

Le 15 novembre 1971, une petite annonce paraît dans *Electronic News*. Intel Corporation y présente le 4004, qu'elle décrit comme un « ordinateur microprogrammable sur une puce ». Derrière cette formulation se cache une invention qui va changer le monde : le premier microprocesseur commercial de l'histoire.

Cette innovation naît d'une rencontre improbable entre une startup américaine spécialisée dans les mémoires et une entreprise japonaise de calculatrices. En 1969, Busicom cherche un partenaire pour développer les circuits intégrés de ses futures calculatrices. Intel, fondée l'année précédente par Bob Noyce et Gordon Moore, accepte ce contrat pour équilibrer ses finances en attendant que son activité principale décolle.

Les spécifications initiales de Busicom prévoient douze circuits intégrés différents, chacun dédié à une fonction précise. Trois ingénieurs japonais, dont Masatoshi Shima, débarquent chez Intel en juin 1969 pour superviser

le projet. Ted Hoff, douzième employé de la société et responsable des applications, doit faire le lien avec cette équipe venue d'Asie. Il examine le cahier des charges et fronce les sourcils. Le système proposé nécessite trop de connexions entre les puces, ce qui rendra le produit final d'un coût prohibitif. Il propose d'abandonner l'idée de circuits spécialisés pour créer un processeur universel programmable, accompagné de quelques puces de mémoire. L'architecture serait plus simple, moins chère, et surtout infiniment plus flexible.

Stanley Mazor rejoint l'aventure en septembre 1969. Son expérience chez Fairchild dans la conception d'ordinateurs apporte la crédibilité technique nécessaire au projet. Ensemble, Hoff et Mazor peaufinent leur vision et parviennent à convaincre les sceptiques de Busicom. Ils développent un interpréteur de 20 octets pour exécuter des macro-instructions, démontrant ainsi la viabilité de leur approche.

En octobre 1969, lors d'une réunion officielle, Busicom donne son feu vert à la nouvelle architecture d'Intel. Shima prolonge son séjour jusqu'en décembre pour écrire les programmes de base de la calculatrice. Le contrat signé accorde à l'entreprise japonaise l'exclusivité sur ce système. Il n'y a plus qu'à transformer cette belle idée en réalité physique. Or ni Hoff ni Mazor ne maîtrisent la conception de circuits intégrés. Intel recrute Federico Faggin en avril 1970. Cet Italien avait mis au point chez Fairchild la technologie MOS à grille de silicium, indispensable pour atteindre les performances et la densité d'intégration souhaitées.

L'équipe finalise l'architecture du système MCS-4. Quatre circuits composent l'ensemble : le processeur 4004, la mémoire ROM 4001, la RAM 4002 et le registre d'entrée-sortie 4003. Le cœur du système, le 4004, rassemble environ 2 300 transistors sur une puce de 3,0 × 4,0 mm. Il exécute 8 000 instructions par seconde et adresse jusqu'à 4 Ko de ROM et 640 octets de RAM.

La fabrication des premiers prototypes démarre en octobre 1970. Les puces périphériques fonctionnent assez vite. Le 4004 résiste davantage mais exige quelques révisions avant d'atteindre sa maturité en mars 1971. Un mois plus tard, Busicom valide l'ensemble dans sa calculatrice. Intel négocie un coup de maître. En mai 1971, l'entreprise récupère les droits de commercialisation du MCS-4 pour toutes les applications autres que les calculatrices de bureau, en échange d'une baisse des prix pour Busicom. Ed Gelbach, nouveau vice-président marketing, orchestre le lancement officiel du 4004 en novembre 1971.

Le succès dépasse les espérances. Intel enchaîne avec le 8008, premier microprocesseur 8 bits, annoncé en avril 1972. Ces deux processeurs ouvrent deux voies distinctes qui structurent encore aujourd'hui l'industrie des semi-

conducteurs : d'un côté les microcontrôleurs intégrés pour l'embarqué, héritiers du 4004, de l'autre les processeurs pour ordinateurs programmables, descendants du 8008.

La postérité du 4004 se mesure à l'aune de chiffres vertigineux. En 1995, vingt-quatre ans après sa naissance, les microcontrôleurs 4 bits représentent la moitié des unités produites dans le monde. L'architecture du 8008 évolue vers le 8080, puis vers les processeurs x86 qui équipent la majorité des ordinateurs personnels actuels.

En 2024, près de 70% des semi-conducteurs fabriqués mondialement sont des microprocesseurs, des microcontrôleurs ou des composants associés. Ce marché pèse plus de 100 milliards de dollars. Des automobiles aux téléphones, des appareils électroménagers aux objets connectés, ces puces issues de l'ADN du 4004 peuplent notre quotidien.

Cette technologie a modifié notre rapport aux machines et transformé notre façon de vivre. Elle illustre parfaitement comment une solution technique élégante à un problème industriel spécifique engendre parfois des bouleversements sociétaux imprévisibles. Le 4004 n'était au départ qu'un composant pour calculatrices japonaises. Il est devenu le fondement de notre civilisation numérique.

Intel 8008

En avril 1972, Intel lança sur le marché l'Intel 8008, premier microprocesseur 8 bits de l'histoire. Cette innovation ne sortait pourtant pas de nulle part. Trois années plus tôt, Computer Terminal Corporation (CTC) cherchait une solution pour moderniser son terminal Datapoint 2200. L'entreprise contacta Intel, société créée seulement un an auparavant par Gordon Moore et Robert Noyce, qui se limitait à la fabrication de mémoires.

CTC avait un problème concret : son terminal utilisait des composants MSI (Medium Scale Integration) dispersés sur plusieurs cartes. L'idée de concentrer toute cette logique sur une seule puce séduisait l'entreprise, qui baptisa ce projet du nom de code 1201. Stan Mazor et Ted Hoff prirent en charge les spécifications fonctionnelles avant de passer le relais à Federico Faggin et Hal Feeney pour la réalisation technique.

L'architecture reproduisait fidèlement celle du processeur CTC existant. Un accumulateur de 8 bits travaillait avec six registres généraux 8 bits (B, C, D, E, H, L). Le processeur gérait un adressage de 14 bits, donnant accès à 16 Ko de mémoire. Son jeu d'instructions restait basique mais couvrait les

opérations arithmétiques et logiques courantes, avec une gestion rudimentaire des interruptions. Techniquement, environ 3 500 transistors PMOS gravés à 10 microns s'entassaient dans un boîtier à 18 broches. La fréquence d'horloge atteignait 200 kHz, permettant d'exécuter 60 000 instructions par seconde.

Cette contrainte des 18 broches faillit compromettre le projet. Intel voulait maîtriser ses coûts de production et refusait d'augmenter le nombre de connexions. L'équipe de conception se retrouva dans une impasse technique, se demandant comment faire passer toutes les informations nécessaires avec si peu de connexions disponibles. La solution vint du multiplexage des bus d'adresses et de données, technique qui compliquait l'utilisation mais rendait la fabrication économiquement viable.

Les premiers prototypes sortirent des chaînes de production fin 1971. Hélas, ils révélèrent des défauts de fonctionnement. Les ingénieurs durent corriger les masques de gravure et recommencer la production. Mars 1972 vit enfin naître les premières puces pleinement fonctionnelles, mais CTC avait changé d'avis entre-temps. L'entreprise jugeait désormais le 8008 trop lent et préférait développer sa propre solution en logique TTL.

Intel se retrouvait avec un produit sur les bras mais décida de le commercialiser. Le pari s'avéra payant. Le 8008 trouva sa place dans des terminaux informatiques, des contrôleurs industriels, des instruments scientifiques. Son prix de lancement – 120 dollars pour la version standard, 180 dollars pour la variante rapide – restait abordable pour de nombreux concepteurs.

La création de ce processeur bouscula les habitudes des dessinateurs qui traçaient les masques à la main sur d'immenses feuilles dépassant 1,5 mètre de largeur, à une échelle 500 fois supérieure à la taille réelle du circuit. Les simulations logiques s'effectuaient laborieusement sur les rares ordinateurs centraux disponibles en temps partagé. L'équipe inventa au passage des techniques comme les charges *bootstrap* pour améliorer les performances électriques du circuit.

Intel ne laissa pas ses clients se débrouiller seuls face à cette technologie inédite. L'entreprise publia des manuels de programmation détaillés, une documentation de référence complète, et proposa des cartes d'évaluation comme la SIM-8. Ces ressources permirent aux ingénieurs de s'approprier les spécificités du microprocesseur.

Kodak intégra le 8008 dans ses photocopieurs, Toledo Scale dans ses balances industrielles. McDonald's l'utilisa pour ses premiers terminaux de point de vente. Certaines applications continuèrent d'exploiter ce processeur plus de dix ans après sa sortie. L'absence de compatibilité binaire avec les générations suivantes comme le 8080 explique cette fidélité. Plutôt que de tout reprogrammer, beaucoup préféraient conserver leurs développements

existants.

Ce petit composant démontra qu'une puce unique pouvait abriter un calculateur complet, produit en série à coût raisonnable. Les concepteurs électroniques commencèrent à voir les choses différemment. L'architecture du 8008, avec ses registres généraux et son multiplexage de bus, influença durablement les générations suivantes de microprocesseurs.

Les ventes restèrent modestes, quelques centaines de milliers d'unités sur toute sa carrière commerciale. Ces chiffres pâlissaient devant ceux du 4004 ou du futur 8080, mais la direction d'Intel hésitait. Elle craignait de devenir concurrente de ses propres clients fabricants d'ordinateurs en se lançant trop franchement sur ce marché. Texas Instruments tenta de créer un rival au 8008 sans y parvenir. Ironiquement, cette société déposa en 1971 le premier brevet couvrant le concept de microprocesseur, alors qu'Intel n'avait protégé que certains aspects techniques spécifiques de sa réalisation. Une bataille juridique qui illustrait la nouveauté radicale de ces technologies.

Les leçons du 8008 nourrirent directement la conception du 8080. Cette fois, les ingénieurs adoptèrent un boîtier à 40 broches qui leur donnait enfin la souplesse nécessaire. Ils conservèrent l'architecture générale mais l'améliorèrent considérablement sur tous les plans.

Toute la famille x86 qui domine le marché des ordinateurs personnels en garde des traces du 8008. Des choix architecturaux comme l'ordre des octets (*little-endian*) ou l'organisation des registres ont survécu cinquante ans et se retrouvent dans les processeurs les plus modernes. Premier microprocesseur 8 bits jamais commercialisé, le 8008 établit des orientations techniques et commerciales qui façonnèrent l'industrie des semi-conducteurs pour les décennies suivantes. Sa création exigea des innovations en méthodologie et en outillage. Malgré ses limites techniques, il révéla le formidable potentiel des microprocesseurs.

C

Aux Bell Labs, au début des années 1970, l'ambiance n'était pas à l'optimisme. L'entreprise venait de claquer la porte de Multics, ce projet pharaonique mené avec le MIT et General Electric qui avait englouti des budgets considérables sans tenir ses promesses. Déçu, Ken Thompson décida de prendre les choses en main pour créer son propre environnement informatique, selon sa vision personnelle.

Il récupéra un vieux DEC PDP-7, une machine déjà dépassée avec ses maigres 8 Ko de mémoire. Là-dessus, il bâtit les premières versions d'UNIX, d'abord en assembleur. Mais programmer directement en assembleur, c'était l'enfer quotidien. Thompson avait testé Fortran, sans succès. Il créa B, une version allégée de BCPL qui tenait dans les contraintes de mémoire du PDP-7.

Dennis Ritchie arriva dans l'équipe et constata que B montrait ses limites. Entre 1969 et 1973, il entreprit de le faire muer vers quelque chose de plus ambitieux. 1972 fut l'année décisive. Le passage au PDP-11 révéla les faiblesses de B : sa gestion des caractères posait problème, et ses pointeurs généraient des conversions absurdes entre indices de mots et adresses. B ignorait complètement les nombres à virgule flottante, alors que le PDP-11 allait bientôt savoir les manipuler.

Ritchie introduisit d'abord les types avec « New B », ajoutant `int` et `char` avec leurs tableaux et pointeurs. Mais c'est avec C que l'idée prit vraiment forme. Le système de types devint sophistiqué, autorisant la composition de structures complexes à partir des types de base. Ritchie eut une intuition géniale pour la syntaxe des déclarations : faire en sorte qu'elle reflète l'usage. Si vous écrivez `int *p`, c'est parce que `*p` vous donne un `int`.

La gestion des tableaux causa des débats. En C, le nom d'un tableau se transforme automatiquement en pointeur vers son premier élément dans les expressions. Cette approche résolvait élégamment le problème des chaînes de caractères tout en préservant le code B existant. Le préprocesseur arriva en 1972-1973, modeste au départ avec juste l'inclusion de fichiers et quelques macros.

En 1973, C était mûr. Ritchie et Thompson s'attaquèrent à la réécriture complète du noyau UNIX. Pari audacieux : personne n'avait encore prouvé qu'un langage de haut niveau pouvait produire du code assez rapide pour un système d'exploitation. Le pari fut gagné. Entre 1973 et 1980, C continua sa croissance : types `unsigned` et `long`, `union`, énumérations, structures devenues citoyens de première classe.

Brian Kernighan et Dennis Ritchie publièrent *The C Programming Language* en 1978. Ce petit livre blanc, que tout le monde appelait « K&R », devint la bible du langage. Sa clarté remarquable contribua largement à populariser C. Mais le langage manquait d'une norme officielle, et chaque compilateur apportait ses petites variations.

La standardisation s'imposa au début des années 1980. C débordait d'UNIX et colonisait d'autres systèmes. Le comité ANSI X3J11 se forma en 1983 sous l'impulsion de M. D. McIlroy. Fin 1989, la norme ANSI C était prête, adoptée comme standard international ISO/IEC 9899-1990. Le

comité resta prudent, n'introduisant qu'une modification majeure : l'intégration des types d'arguments dans les signatures de fonctions. Cette idée, empruntée à C++, renforçait le contrôle de types sans casser l'existant.

L'expansion de C fut fulgurante. Dans les années 1980, des compilateurs apparurent sur pratiquement toutes les architectures et tous les systèmes. Les développeurs de logiciels commerciaux pour micro-ordinateurs l'adoptèrent massivement. Cette réussite tenait à l'équilibre du langage : assez simple pour être maîtrisé, assez proche de la machine pour produire du code efficace, assez abstrait pour la portabilité.

C engendra une descendance prolifique. Bjarne Stroustrup créa C++, et d'autres donnèrent naissance à Objective C et Concurrent C. Par la suite, les langages modernes qui ont repris sa syntaxe et ses concepts sont innombrables. Les pointeurs, le traitement des tableaux, la syntaxe des déclarations de types restent des contributions durables, même si elles font débat.

Le langage a ses défauts. La modularisation des gros systèmes demeure problématique avec sa structure de nommage à deux niveaux seulement. La gestion manuelle de la mémoire dynamique piège régulièrement les programmeurs. Ces faiblesses n'ont pourtant pas empêché le développement de systèmes d'exploitation complets en C.

L'histoire de C montre comment un outil né de nécessités pratiques immédiates devient, par des choix de conception judicieux, un standard qui traverse les époques. Sa longévité dans un univers technologique en perpétuel bouleversement témoigne de la justesse des intuitions de ses créateurs. Plus de cinquante ans après sa naissance, C figure encore parmi les langages les plus utilisés dans la programmation système et embarquée. Go, dont Ken Thompson est le co-créateur, tente de le rattraper.

Prolog

En 1970, Alain Colmerauer ne cherche pas à inventer un nouveau langage de programmation. Ce professeur assistant en informatique à l'Université de Montréal dirige le projet TAUM (Traduction Automatique de l'Université de Montréal) et veut simplement pousser plus loin ses recherches sur le traitement informatique des langues. Deux jeunes enseignants de 25 ans de la toute nouvelle faculté des sciences de Luminy, Robert Pasero et Philippe Roussel, le rejoignent pour deux mois. Ils découvrent les systèmes-Q, un langage que Colmerauer a développé pour son projet de traduction.

Jean Trudel, doctorant canadien sous la direction de Colmerauer, travaille sur la démonstration automatique de théorèmes. Il s'appuie sur l'article fondateur d'Alan Robinson sur le principe de résolution, publié en 1965. Il a l'avantage d'avoir suivi un cours de logique avec Martin Davis à New York et maîtrise déjà l'unification selon une approche moderne où tous les calculs se résument à modifier des pointeurs.

Début 1971, l'équipe se reconstitue à Marseille. Alain Colmerauer obtient un poste de maître de conférences en informatique et Jean Trudel l'accompagne grâce à une bourse de deux ans d'Hydro-Québec. Leur objectif était de réaliser des déductions à partir de textes français. La répartition du travail est telle que Jean Trudel et Philippe Roussel s'attaquent aux déductions, Robert Pasero et Alain Colmerauer au langage naturel. Leur environnement de travail tient du luxe pour la France de l'époque. L'IBM 360-44 de leur département offre près de 900 Ko de mémoire interne, sans mémoire virtuelle. Jean Trudel développe un moniteur interactif et ils monopolisent la machine la nuit.

En juin 1971, l'équipe invite Robert Kowalski, l'un des inventeurs de la *SL-resolution*. Cette rencontre change tout. Pour la première fois, ils échangent avec un spécialiste capable d'expliquer le principe de résolution, ses variantes et ses raffinements. Kowalski découvre des chercheurs passionnés par ses travaux et déterminés à les appliquer au traitement du langage naturel.

En février 1972, l'Institut de Recherche d'Informatique et d'Automatique (IRIA) leur accorde 122 000 FF pour 18 mois. Cette subvention leur permet d'investir dans un terminal télétype, une liaison dédiée à 300 bauds vers l'IBM 360-67 de Grenoble, et le recrutement d'Henry Kanoui.

L'automne 1972 voit naître le premier système Prolog. Philippe Roussel le développe en Algol-W tandis qu'Alain Colmerauer et Robert Pasero créent en parallèle le système de communication homme-machine en français. L'interaction entre Philippe, qui code Prolog, Alain et Robert, qui programment dans un langage en cours de création, se pérennise. C'est l'épouse de Philippe qui suggère le nom définitif : Prolog, abréviation de « PROgrammation en LOGique ».

Le premier grand programme écrit en Prolog est le système de communication homme-machine. Ses 610 clauses se répartissent ainsi : Alain en écrit 334 pour l'analyse, Robert Pasero 162 pour la partie déductive, et Henry Kanoui développe une morphologie française en 104 clauses. Cette morphologie gère les liens entre singulier et pluriel de tous les noms communs et tous les verbes, réguliers et irréguliers, à la troisième personne du singulier du présent.

La reconnaissance officielle arrive en 1973. Le CNRS constitue une

« équipe de recherche associée » baptisée « Dialogue homme-machine en langage naturel » et accorde 39 000 FF pour la première année. Les utilisateurs de la version préliminaire de Prolog ont suffisamment programmé pour que leur expérience nourrisse une deuxième version, orientée vers un véritable langage de programmation et non plus seulement un système déductif automatisé.

Entre juin et décembre 1973, trois étudiants en DEA – Gérard Battani, Henry Méloni et René Bazzoli – écrivent l'interpréteur en FORTRAN et son superviseur en Prolog. Cette version introduit toutes les caractéristiques de base des Prolog actuels. C'est aussi à cette période que disparaît la vérification d'occurrence (*occur check*), jugée trop coûteuse.

La diffusion commence vraiment en 1974-1975. David Warren séjourne à Marseille de janvier à mars 1974 et utilise Prolog pour écrire Warplan, son système de génération de plans. La version interactive de Prolog qui fonctionne à Grenoble via télétype suscite de nombreuses demandes. Gérard Battani et Henry Méloni croulent sous les distributions. Ils expédient Prolog à Budapest, Varsovie, Toronto, Waterloo et se déplacent à Édimbourg pour aider David Warren à l'installer sur un PDP 10. Hélène Le Gloan, ancienne étudiante, l'installe à l'Université de Montréal. Michel Van Caneghem fait de même à l'IRIA à Paris avant de rejoindre l'équipe. Maurice Bruynooghe emporte Prolog à Louvain après trois mois à Marseille.

En 1975, toute l'équipe participe au portage sur un mini-ordinateur 16 bits, le T1600 de Télémécanique. La machine ne dispose que de 64 Ko et nécessite un système de gestion de mémoire virtuelle spécifique. Pierre Basso s'attaque à cette tâche et remporte le concours de la séquence d'instructions la plus courte qui réalise un adressage sur 32 bits tout en testant le défaut de page. Chaque membre du laboratoire traduit alors deux pages de FORTRAN en langage machine. Les fragments réassemblés fonctionnent. Après cinq ans, ils possèdent enfin leur propre machine et leur Prolog tourne, lentement, mais il tourne.

L'histoire de la naissance de Prolog s'achève fin 1975, mais son influence perdure. L'article d'Alan Robinson de janvier 1965, « A machine-oriented logic based on the resolution principle », contenait les germes du langage Prolog. Si cet article a donné naissance à un courant important de travaux sur la démonstration automatique de théorèmes, la contribution spécifique de l'équipe marseillaise a été de transformer ce démonstrateur en langage de programmation. Pour y parvenir, ils n'ont pas hésité à introduire des mécanismes purement calculatoires et des restrictions qui constituaient des hérésies pour le modèle théorique existant. Ces modifications, souvent critiquées, ont assuré la viabilité et donc le succès de Prolog.

Robert Kowalski a isolé le concept de « clause de Horn », légitimant leur

principale hérésie : une stratégie de démonstration linéaire avec retour arrière et unifications uniquement aux têtes de clauses. L'équipe marseillaise disposait des conditions idéales pour cette création. Colmerauer appartenait à la première génération de docteurs en informatique français et sa spécialité était la théorie des langages. Il avait acquis une expérience précieuse en créant les systèmes-Q au sein du projet de traduction automatique de l'Université de Montréal. La rencontre avec Philippe Roussel et les conditions particulières de Marseille ont fait le reste. L'équipe a bénéficié d'une liberté d'action dans un centre scientifique nouvellement créé et, sans pressions extérieures, elle a pu se consacrer pleinement à son projet.

Smalltalk

En 1969, au cœur du Xerox PARC de Palo Alto, Alan Kay commence à rêver d'un ordinateur différent, dans l'effervescence des premières machines interactives, des écrans graphiques émergents et de ces souris que Douglas Engelbart vient d'inventer. Kay imagine un « medium personnel » capable d'amplifier l'intelligence humaine. De cette vision radicale naîtra Smalltalk, un langage qui bouleversera notre façon de concevoir la programmation.

L'idée germe à partir de trois influences marquantes. Simula avait introduit les objets, LISP maîtrisait la manipulation symbolique, et Sketchpad de Sutherland montrait ce que pouvait donner l'interaction graphique. Alan Kay mélange ces ingrédients avec l'audace qui le caractérise, et lance le pari de définir « le langage de programmation le plus puissant du monde » en une seule page de code. Pari tenu avec Smalltalk-71, une première version réalisée en quelques jours qui démontre qu'un système entier est capable de reposer sur l'échange de messages entre objets. Fini les langages où les objets côtoient les types primitifs dans une hiérarchie bancale. Ici, tout est objet : les nombres comme les structures de contrôle, les fenêtres comme les caractères. Cette radicalité conceptuelle n'a rien d'un caprice intellectuel. Elle forge un système d'une cohérence rare, où chaque élément obéit aux mêmes règles fondamentales.

Le concept mûrit au rythme des versions successives. Entre 1972 et 1976, l'équipe du PARC peaufine ce monde d'objets communicants. Dan Ingalls apporte sa contribution décisive avec l'invention des fenêtres superposables. Cette interface graphique deviendra plus tard un standard industriel, notamment quand Apple s'en inspirera pour le Macintosh.

Avec Smalltalk-76, Ingalls consolide les acquis des expérimentations précédentes en proposant un système complet : compilation à la volée, gestion

automatique de la mémoire, interface graphique intégrée. L'environnement de développement autorise la modification du code pendant que le programme s'exécute, quelque chose d'inédit. Cette souplesse transforme la programmation en exploration permanente.

Mais c'est avec Smalltalk-80, rendu public en 1980, que le système atteint sa maturité. Dix années de recherche aboutissent à cette version qui standardise le langage et son environnement. Les concepts fondamentaux se cristallisent : tout est objet, les objets communiquent par messages, les classes sont elles-mêmes des objets, l'héritage rend possible la réutilisation du code. Ces principes influenceront durablement l'évolution de l'informatique par la suite.

La force de Smalltalk tient à sa pureté conceptuelle. Là où d'autres langages comme C++ ou Java greffent tant bien que mal des fonctionnalités orientées objet sur une base procédurale, Smalltalk applique son paradigme de bout en bout. Cette cohérence théorique s'accompagne d'innovations pratiques remarquables : l'environnement de développement intégré, le ramasse-miettes qui libère le programmeur de la gestion mémoire, l'interface moderne avec ses fenêtres, menus et souris.

L'expansion de Smalltalk coïncide avec l'émergence des ordinateurs personnels assez puissants pour l'héberger. Steve Jobs découvre le système lors de sa visite au PARC en 1979. Le choc est tel qu'il oriente directement le développement de l'interface du Macintosh. L'influence se propage ensuite dans toute l'industrie : Objective-C, Ruby, Python, Java empruntent chacun des éléments du modèle objet de Smalltalk.

Alan Kay avait initialement conçu Smalltalk pour l'apprentissage de la programmation par les enfants. Les objets modélisent intuitivement les entités du monde réel, l'interface graphique facilite l'expérimentation. Cette dimension éducative survit dans des projets comme Squeak et Etoys, qui perpétuent l'idée d'une informatique accessible à tous.

Le développement de Smalltalk illustre l'importance du contexte institutionnel dans l'innovation. Le Xerox PARC offrait à ses chercheurs une liberté exceptionnelle, sans pression commerciale immédiate. Cette autonomie a rendu possible l'émergence d'un système novateur. Paradoxalement, Xerox n'a jamais su tirer profit commercialement de cette innovation majeure.

Les années 1990 et 2000 voient fleurir de multiples implémentations : VisualWorks pour les professionnels, Squeak pour l'éducation, GNU Smalltalk pour le monde libre. Cette diversité témoigne de la vitalité persistante du modèle et de sa capacité d'adaptation.

Smalltalk n'est plus un langage dominant de nos jours, mais les principes qu'il a introduits – objets, messages, héritage, environnement interac-

tif – sont devenus des standards industriels. L'évolution vers les interfaces tactiles et la programmation par blocs prolonge sa vision d'une informatique intuitive et accessible. Cette histoire révèle l'importance de la vision à long terme dans le progrès technologique. Smalltalk combine une théorie élégante de la programmation objet avec des innovations pratiques dans l'interface utilisateur. Son parcours montre qu'il faut du temps pour que les idées trouvent leur place, et que l'impact véritable d'une innovation se mesure souvent à l'aune des générations suivantes.

IBM VM/370

En 1972, IBM lança VM/370, pour Virtual Machine Facility/370. Derrière ce nom technique se cachait la possibilité de faire tourner plusieurs systèmes d'exploitation en parallèle sur une seule machine physique. L'idée n'était pas entièrement nouvelle puiqu'elle prolongeait les travaux menés sur CP-67/CMS pour le System/360 Model 67, mais sa mise en œuvre systématique passa à l'échelle de l'informatique d'entreprise.

Il fallait de l'audace pour imaginer ce concept à l'époque. VM/370 créait des « machines virtuelles », des environnements informatiques complets mais factices, chacun croyant disposer de son propre ordinateur. Cette illusion reposait sur quatre composants : le programme de contrôle CP qui orchestrait les ressources physiques, le système conversationnel CMS pour dialoguer avec les utilisateurs, le sous-système RSCS qui gérait les échanges de fichiers, et IPCS qui s'occupait des incidents. L'architecture masquait la complexité au profit de l'efficacité, chaque machine virtuelle évoluait dans sa bulle, ignorant totalement ses voisines. Cette isolation permettait de faire cohabiter différents systèmes IBM – DOS, OS/VS1, OS/VS2 – sans qu'ils se marchent sur les pieds. La prouesse technique tenait dans cette capacité à découper un seul ordinateur en multiples environnements étanches.

Côté mémoire, VM/370 introduisit une traduction d'adresses dynamique qui convertissait les adresses virtuelles en adresses physiques réelles. Le système découpait la mémoire en segments de 64 Ko et en pages de 4 Ko, créant plusieurs niveaux d'abstraction. Cette gymnastique intellectuelle permettait de faire tourner des machines virtuelles dont la somme dépassait la mémoire physique disponible. L'innovation s'étendait aux périphériques avec le « spooling », cette technique qui gérait les entrées-sorties de toutes les machines virtuelles. Un disque ou une imprimante pouvaient servir plusieurs environnements sans créer de conflits. VM/370 jonglait avec les ressources comme un prestidigitateur avec ses balles.

Chaque utilisateur recevait son identifiant et son mot de passe, et l'accès aux ressources suivait des règles de sécurité strictes. L'isolation entre machines virtuelles constituait une barrière naturelle suffisante pour qu'un plantage dans l'une n'affecte pas les autres. Cette robustesse séduisit immédiatement les développeurs qui pouvaient tester leurs programmes sans craindre de faire s'effondrer le système de production.

L'impact sur l'industrie dépassa les espérances d'IBM. Les entreprises découvrirent qu'elles pouvaient exploiter leurs coûteux ordinateurs de manière bien plus intensive. Au lieu de laisser des ressources dormir, elles les partageaient entre diverses applications. Fini les attentes interminables des informaticiens pour tester leur programme, chacun disposait de son environnement virtuel.

VM/370 s'adapta progressivement aux nouveaux matériels IBM, du modeste Model 135 au puissant Model 168. Le système intégra le support des disques 3330, 3340 et 3350, s'enrichit de fonctionnalités réseau et gagna en performances. Cette évolution constante témoignait de la vitalité du concept.

Les principes fondamentaux de VM/370 – isolation, partage des ressources, gestion dynamique de la mémoire – façonnent toujours les technologies modernes de virtualisation. VMware, Xen, KVM et les autres solutions contemporaines puisent dans ce patrimoine conceptuel forgé dans les années 1970. Plus qu'une prouesse technique, VM/370 modifia la façon de concevoir l'informatique d'entreprise. Le système prouva qu'il était possible de découper les ressources informatiques sans sacrifier les performances ni la sécurité. Les architectures à venir seraient donc plus souples et économiques.

L'influence du système sur la formation informatique fut considérable. Les écoles et universités adoptèrent VM/370 pour enseigner les systèmes d'exploitation. Les étudiants pouvaient expérimenter sur leurs propres machines virtuelles sans monopoliser un ordinateur entier. Cette démocratisation de l'accès aux ressources informatiques forma une génération d'informaticiens familiers de la virtualisation. VM/370 illustre comment une innovation technique peut transformer un secteur entier. En démontrant la viabilité de la virtualisation, IBM redéfinissait l'usage des ordinateurs. Cette transformation continue de résonner dans les centres de données modernes, où la virtualisation constitue désormais la norme plutôt que l'exception.

Ethernet

En 1972, dans les laboratoires de Xerox à Palo Alto, Bob Metcalfe et David Boggs travaillent sur la communication des ordinateurs avec des imprimantes laser dans un bâtiment. Leur solution, qu'ils baptisent Ethernet, va bien au-delà de cette demande initiale en atteignant 2,94 mégabits par seconde.

L'idée de base tient du génie dans sa simplicité. Tous les ordinateurs se partagent le même câble, comme des gens qui discutent autour d'une table. Avant de parler, chacun écoute pour s'assurer que personne d'autre n'a pris la parole. Si deux personnes commencent en même temps, elles s'arrêtent, attendent un moment au hasard, puis recommencent. Cette méthode, technique baptisée CSMA/CD, fonctionne remarquablement bien malgré son apparente anarchie.

Le premier Ethernet circule dans un câble coaxial de 75 ohms. Les machines s'y connectent grâce à des boîtiers spéciaux qui détectent les collisions et isolent électriquement chaque station. Il est possible de brancher ou débrancher une machine sans paralyser tout le réseau, ce qui représente un avantage considérable sur les systèmes centralisés. Quatre ans plus tard, Metcalfe et Boggs décrivent leur invention dans un article des « Communications of the ACM ». Ethernet sort du laboratoire pour gagner le monde industriel. Les premières installations montrent la capacité à connecter des centaines de machines sur plus d'un kilomètre.

Xerox comprend alors le potentiel commercial de sa création. En 1980, l'entreprise s'allie avec Intel et Digital Equipment Corporation pour transformer Ethernet en standard ouvert. Cette décision stratégique s'avère payante : libérer la technologie de son créateur accélère son adoption. La nouvelle spécification pousse le débit à 10 mégabits par seconde et adopte le câble coaxial 50 ohms. Les adresses des machines passent de 8 à 48 bits, garantissant leur unicité mondiale.

L'IEEE prend le relais avec son comité 802.3 au début des années 1980. Cette normalisation inscrit Ethernet dans le modèle OSI et le rend totalement indépendant des constructeurs. Le protocole CSMA/CD est la sous-couche MAC de la couche liaison, tandis que les aspects physiques relèvent des normes 802.3.

En 1990, la norme 802.3i abandonne le bus coaxial au profit des paires torsadées et de la topologie en étoile. Fini les câbles qui serpentent dans les faux plafonds, chaque machine a désormais son propre lien vers un concentrateur central, faisant de l'installation et la maintenance un jeu d'enfant. Le Fast Ethernet arrive en 1995 avec ses 100 mégabits par seconde, suivi trois ans plus tard par le Gigabit Ethernet.

Metcalfe et Boggs ont fait des choix techniques qui durent. Ethernet ne gère ni les priorités ni la qualité de service, laissant ces questions aux protocoles de niveau supérieur. Cette apparente limitation se révèle être une force car la simplicité du mécanisme de base autorise toutes les adaptations ultérieures. Le cœur du système est remarquablement stable. Le contrôleur surveille le canal et applique les règles CSMA/CD. L'émetteur-récepteur fait l'interface avec le support physique. Les trames transportent les adresses source et destination, un champ type pour identifier le protocole encapsulé, les données et un code de contrôle CRC. Cette architecture épurée traverse les décennies sans modifications majeures.

Durant les années 1990, les commutateurs (*switches*) transforment radicalement la nature d'Ethernet. Au lieu d'un canal partagé où tout le monde s'écoute, le réseau évolue vers un ensemble de liens point à point. Différentes conversations peuvent avoir lieu simultanément. Le mode full-duplex remplace progressivement le CSMA/CD, devenu moins utile avec la commutation.

Pourquoi Ethernet a-t-il triomphé là où d'autres ont échoué ? D'abord par son coût réduit, conséquence directe de sa simplicité. Ensuite par sa capacité d'évolution : les débits augmentent, les supports changent, mais les principes fondamentaux demeurent. Enfin par son ouverture, aucun constructeur ne pouvant s'approprier le réseau.

Les concepteurs ont validé leurs choix par l'expérimentation. L'Ethernet expérimental a fourni des mesures précises montrant une utilisation du support proche de 98% dans certaines conditions. Le mécanisme de *binary exponential backoff* (BEB), un algorithme permettant de limiter la charge du réseau quand une collision se produit entre deux messages, assure la stabilité sous n'importe quelle charge. Ces données concrètes ont guidé l'élaboration du standard industriel.

Aujourd'hui, Ethernet transporte la voix sur IP, connecte les baies de stockage, irrigue les centres de données. Les débits atteignent 400 gigabits par seconde sur fibre optique. La technologie née pour relier quelques ordinateurs à des imprimantes structure désormais l'internet mondial. Les solutions les plus durables ne sont pas toujours les plus sophistiquées.

TCP/IP

À la fin des années 1960, dans le contexte particulier de la guerre froide, les États-Unis cherchaient à établir un réseau de communication militaire

capable de résister à une attaque nucléaire soviétique. Le Département de la Défense américain, à travers son agence DARPA (Defense Advanced Research Projects Agency), lança en 1969 le projet ARPAnet qui visait à interconnecter les principaux centres informatiques du pays. Les contraintes militaires façonnent d'emblée la conception du réseau. Pas question d'avoir un talon d'Achille, un point central dont la destruction compromettrait tout le système. Les messages doivent pouvoir emprunter des routes alternatives de façon dynamique si certaines portions tombent. Plus complexe, le réseau doit relier des ordinateurs tournant sous des systèmes d'exploitation différents.

ARPAnet démarre avec le protocole NCP, mais ses défauts apparaissent en peu de temps. En 1974, deux chercheurs changent la donne : Vinton Cerf et Robert Kahn publient « A Protocol for Packet Network Interconnection », article fondateur qui présente TCP. Cette publication dessine les contours de ce qui deviendra TCP/IP, standardisant la communication entre les machines. L'astuce de TCP/IP tient dans sa structure en couches séparées. IP s'occupe d'acheminer les paquets de données d'un point à l'autre du réseau, tandis que TCP veille au contrôle de la transmission et reconstitue les messages complets à l'arrivée. Cette division des tâches rend l'ensemble plus solide et plus simple à faire évoluer, un principe d'ingénierie qui fera ses preuves.

En 1983, TCP/IP remplace officiellement NCP sur ARPAnet. Bien que ce point soit très largement discuté et discutable, le concept d'Internet naît véritablement à ce moment, au sens littéral du terme. Le succès de TCP/IP repose sur trois atouts décisifs. Il fonctionne sur n'importe quel matériel, ordinateur ou réseau physique. Il maintient la communication quand une partie du réseau tombe en panne grâce à son routage dynamique. Il découpe les messages en paquets qui peuvent emprunter des chemins différents avant d'être réassemblés. L'adressage IP de première génération utilise des adresses sur 32 bits qui identifient chaque machine de manière unique. Le système classe ces adresses en catégories A, B et C pour simplifier la gestion. Mais la croissance explosive du réseau révèle vite les limites de cette approche. Les informaticiens développent alors des solutions de contournement comme le sous-adressage, en attendant IPv6 et ses adresses sur 128 bits.

L'architecture TCP/IP s'organise en quatre étages : Accès réseau pour la connexion physique, Internet (IP) pour le routage, Transport (TCP/UDP) pour la transmission, et Application pour les programmes utilisateur. Cette hiérarchie facilite l'ajout de nouvelles fonctionnalités sans bouleverser l'existant, un avantage énorme pour l'évolution du système. Plusieurs protocoles auxiliaires viennent compléter TCP/IP. ARP traduit les adresses IP logiques en adresses physiques des cartes réseau. ICMP transporte les mes-

sages de contrôle et d'erreur. Ces mécanismes contribuent à la fiabilité et à l'efficacité du système global.

UNIX Berkeley contribue à la diffusion de TCP/IP durant les années 1980. L'intégration native du protocole dans BSD facilite son adoption par les universités et centres de recherche qui utilisent massivement ce système d'exploitation. La documentation technique est publique et accessible à tous sous forme de RFC, documents ouverts aux contributions de la communauté. Cette démarche décentralisée stimule l'innovation et l'amélioration continue, une approche qui influencera durablement le développement technologique.

L'explosion du World Wide Web dans les années 1990 valide spectaculairement le choix de TCP/IP. Le protocole absorbe sans broncher une croissance exponentielle du nombre d'utilisateurs et l'arrivée d'applications gourmandes en ressources. Sa souplesse autorise l'ajout de fonctionnalités avancées comme le *multicast* pour diffuser vers de multiples destinataires, et qualité de service pour prioriser certains flux. L'histoire de TCP/IP continue d'ailleurs de s'écrire. L'épuisement annoncé des adresses IPv4 et les nouveaux besoins en sécurité et performance poussent au développement d'IPv6 dans les années 1990. Cette version propose un espace d'adressage quasi infini – $3{,}4\times10^{38}$ adresses – et des améliorations substantielles. La migration vers IPv6 progresse très lentement, compliquée par la nécessité de maintenir la compatibilité avec l'existant.

De projet militaire à standard universel, TCP/IP colonise tous les domaines contemporains. Ordinateurs bien sûr, mais aussi smartphones, objets connectés, équipements industriels, la liste s'allonge chaque jour. Cette ubiquité tient à sa remarquable faculté d'adaptation aux évolutions technologiques sans rompre avec l'héritage. La standardisation de TCP/IP a rendu possible l'émergence d'un réseau planétaire où des milliards de machines dialoguent, peu importe leur origine ou leur localisation géographique.

Telnet

Dans l'univers informatique du début des années 1970, l'ARPANET venait de naître, mais ses promesses d'interconnexion universelle se heurtaient à la réalité technique. Chaque fabricant développait ses propres protocoles, ses propres standards, créant autant d'îlots technologiques étanches. Un terminal fonctionnant avec le jeu de caractères EBCDIC ne savait tout simplement pas dialoguer avec un ordinateur PDP-10 configuré pour l'AS-

CII. Il fallait faire communiquer entre elles des machines de constructeurs différents.

Le protocole Telnet émergea pour répondre à ce problème. Ses concepteurs comprirent qu'il fallait inventer une langue commune informatique qui permettrait à n'importe quel terminal de se connecter à n'importe quel ordinateur distant. Cette intuition donna naissance au Network Virtual Terminal (NVT), concept d'une élégance remarquable dans sa simplicité. Il fonctionne comme un traducteur universel. Plutôt que de forcer chaque machine à comprendre tous les dialectes possibles de ses congénères, le système impose un format standardisé unique. Chaque ordinateur n'a qu'une seule traduction à maîtriser : celle entre son langage natif et cette *lingua franca* virtuelle. Un terminal EBCDIC traduit vers le NVT, qui retraduit ensuite vers l'ASCII attendu par la machine de destination. L'efficacité de cette approche frappa immédiatement les ingénieurs de l'ARPANET.

Mais l'histoire de Telnet recèle un épisode savoureux. En 1972, Bernard Cosell et David Walden, tous deux employés de BBN et travaillant sur le Terminal Interface Message Processor, se trouvèrent confrontés à un casse-tête récurrent. Les spécifications de Telnet évoluaient constamment, et chaque modification imposait de mettre à jour simultanément toutes les instances existantes sur le réseau. Une tâche pharaonique qui ralentissait considérablement le développement. C'est dans un avion en route vers UCLA que Cosell eut l'illumination. Pourquoi ne pas laisser les deux extrémités d'une connexion négocier entre elles les fonctionnalités qu'elles souhaitaient utiliser ? Cette idée, qu'il développa avec Walden pendant le vol, révolutionna l'approche des protocoles réseau. De retour au sol, ils conçurent un mécanisme reposant sur quatre commandes lapidaires : `DO`, `DON'T`, `WILL` et `WON'T`. Ces quatre mots magiques transformèrent Telnet en un protocole vivant, capable d'évoluer sans casser l'existant. Quand une machine proposait une nouvelle fonctionnalité via `WILL`, son interlocutrice pouvait l'accepter avec `DO` ou la décliner poliment avec `DON'T`. Si elle ne comprenait pas la proposition, un simple refus suffisait, et la communication reprenait sur les bases communes du NVT. Cette négociation permanente permit au protocole de s'enrichir progressivement, telle une langue qui intègre naturellement de nouveaux termes sans perdre sa cohérence. Ray Tomlinson, déjà célèbre pour avoir inventé l'e-mail, travaillait alors sur l'implémentation Telnet pour le système TENEX. Dès qu'il découvrit le mécanisme imaginé par Cosell et Walden, il en saisit immédiatement la portée. David Walden formalisa l'idée dans le RFC 435, publié en janvier 1973, posant les jalons de ce qui deviendrait un standard durable.

La standardisation définitive intervint dix ans plus tard, avec le RFC 854 de 1983. Telnet y gagna sa forme moderne : connexion TCP bidirectionnelle sur le port 23, encodage ASCII 7 bits, et gestion sophistiquée du flux de

données. Les ingénieurs y ajoutèrent un mécanisme de synchronisation ingénieux, pensé pour les situations d'urgence où un utilisateur doit reprendre le contrôle d'un programme distant devenu fou. Le signal « Synch » utilise les capacités d'urgence de TCP pour court-circuiter le contrôle de flux normal et garantir la transmission immédiate des commandes critiques. Cette architecture élégante fit de Telnet bien plus qu'un simple émulateur de terminal. Le protocole devint l'épine dorsale de l'accès distant aux systèmes informatiques. Dans les années 1980 et 1990, il alimenta l'explosion des premiers services en ligne et des BBS, ces babillards électroniques précurseurs des forums d'aujourd'hui. Sa simplicité séduisait autant les développeurs que les utilisateurs.

Conçu dans les années 1970, quand l'informatique évoluait dans un monde de confiance relative, Telnet ne chiffrait rien. Mots de passe, données sensibles, tout transitait en clair sur les réseaux. Cette faiblesse, anodine à l'origine, devint rédhibitoire avec la démocratisation d'Internet et l'émergence des préoccupations sécuritaires. L'arrivée des interfaces graphiques porta un coup supplémentaire au vénérable protocole. Les utilisateurs délaissèrent progressivement les terminaux en mode texte pour les fenêtres colorées et les icônes cliquables. SSH, apparu dans les années 1990, récupéra les atouts de Telnet en y ajoutant le chiffrement qui lui manquait tant.

S'il est chassé par les experts en sécurité, Telnet n'a pas disparu pour autant. Dans les salles serveurs, il est l'outil de diagnostic par excellence. Les administrateurs réseau l'ont utilisé longtemps, peut-être encore d'ailleurs, pour tester la connectivité sur un port donné ou configurer de vieux équipements récalcitrants. Cette persistance témoigne de la robustesse de sa conception originelle.

Le concept de terminal virtuel a essaimé dans d'innombrables protocoles ultérieurs. Le mécanisme de négociation d'options, né dans cet avion vers UCLA, inspira les ingénieurs confrontés au défi de l'extensibilité. Comment faire évoluer un protocole sans casser l'existant ? Cosell et Walden avaient trouvé une réponse d'une simplicité déconcertante, qui continue de faire école. Face à la complexité croissante des systèmes, la tentation était grande de développer des solutions sophistiquées, aux fonctionnalités pléthoriques. Telnet choisit de privilégier l'interopérabilité et la flexibilité sur la sophistication.

Le langage SQL

En 1969, un chercheur d'IBM nommé Edgar Frank Codd publie un article aux allures anodines : « Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks ». Personne ne se doute que ce texte va bouleverser la façon dont nous stockons et manipulons les données informatiques. L'année suivante, Codd récidive avec un papier plus ambitieux dans Communications of l'ACM : « A Relational Model of Data for Large Shared Data Banks ». Cette fois, l'industrie prête attention.

Le modèle relationnel de Codd rompt avec les approches hiérarchiques et réseau dominantes à l'époque. Il propose d'organiser les données dans des tables simples, liées entre elles par des relations mathématiques. L'idée paraît presque triviale aujourd'hui, mais elle révolutionne la pensée informatique. Les communautés universitaire et industrielle s'emparent de ces concepts théoriques.

Chez IBM, Donald Chamberlin et son équipe ne perdent pas de temps. En 1974, ils développent System R, le premier prototype de base de données relationnelle digne de ce nom. Le langage de requête associé reçoit le nom de SEQUEL, pour Structured English QUEry Language. Ce choix trahit une ambition : créer un langage proche de l'anglais naturel, accessible aux utilisateurs non spécialistes. Fini les commandes cryptiques, place à des instructions compréhensibles par le plus grand nombre.

Le projet avance par étapes. SEQUEL-XRM voit le jour entre 1974 et 1975, suivi d'une réécriture complète en 1976-1977. Cette nouvelle mouture, baptisée SEQUEL/2, introduit des fonctionnalités indispensables : requêtes multi-tables, support multi-utilisateurs, gestion avancée des données. Mais un obstacle imprévu surgit : la société britannique Hawker Siddeley Aircraft Company possède déjà la marque SEQUEL. IBM doit rebaptiser son langage avec l'acronyme SQL.

L'histoire prend une tournure inattendue en 1979. Alors qu'IBM peaufine ses prototypes, une petite société californienne nommée Relational Software commercialise le premier système de gestion de base de données relationnelle. Dans un trait de génie marketing, cette entreprise numérote son produit « Version 2 », évitant ainsi les réticences habituelles face aux premières versions. Relational Software deviendra Oracle Corporation, prenant de vitesse le géant IBM sur son propre terrain.

IBM rattrape son retard avec SQL/DS en 1981, et DB2 en 1983. Entretemps, les tests clients menés dès 1978 ont validé l'approche : SQL séduit par sa simplicité d'usage et sa puissance, permettant de décrire ce que l'on veut obtenir sans préciser comment l'obtenir. Cette approche déclarative contraste avec les langages procéduraux traditionnels et démocratise l'accès

aux bases de données.

La standardisation s'impose naturellement. En 1986, l'American National Standards Institute certifie SQL comme norme officielle. L'Organisation internationale de normalisation suit en 1987. Ces premières normes, d'environ 150 pages, standardisent le dialecte SQL d'IBM DB2. Mais elles souffrent de lacunes et d'imprécisions, conséquences des intérêts commerciaux divergents des éditeurs.

La norme SQL2 de 1992 corrige ces défauts. Ce document de 500 pages élimine quelques unes des faiblesses de la version précédente et standardise des fonctionnalités conceptuelles avancées, parfois au-delà des capacités techniques admissibles. Trois niveaux de conformité voient le jour : *Entry* pour les modifications mineures, *Intermediate* pour au moins 50% de la norme, et *Full* pour la conformité totale.

SQL99, ou SQL3, marque une nouvelle évolution à la fin des années 1990. Cette norme de 2000 pages intègre les modèles objet-relationnels, les interfaces d'appel et la gestion avancée de l'intégrité. Elle remplace les anciens niveaux de conformité par deux catégories : Core SQL99 et Enhanced SQL99. Le document se structure désormais en cinq parties spécialisées, témoignant de la complexité croissante du langage.

Le XXI^{e} siècle voit naître SQL :2003, SQL :2008 et SQL :2011, chaque version enrichissant le langage de nouvelles capacités. Mais au-delà de cette chronologie technique, SQL doit son succès à des qualités plus profondes. Sa syntaxe inspirée de l'anglais naturel le rend abordable aux non-spécialistes. Sa nature déclarative libère l'utilisateur des contraintes procédurales. Le modèle relationnel sous-jacent structure les données de façon claire et cohérente.

Les concepts de clés primaires et étrangères, de normalisation et d'intégrité référentielle garantissent la cohérence des données stockées. Cette rigueur mathématique, alliée à la flexibilité d'usage, explique l'adoption massive de SQL par l'industrie. Oracle, MySQL, PostgreSQL, Microsoft SQL Server et tant d'autres systèmes perpétuent cet héritage, étendant les capacités du langage tout en préservant la compatibilité.

System R

Durant l'année 1974, une équipe de chercheurs du laboratoire IBM de San José voulait démontrer que le modèle relationnel théorisé par Edgar F. Codd quatre ans plus tôt pouvait fonctionner dans la réalité, avec de

vraies données et de vrais utilisateurs. Ce projet, baptisé System R, n'était pas qu'une simple expérimentation technique. Il s'agissait de construire un pont entre la beauté mathématique du modèle relationnel et les contraintes brutales du monde industriel.

L'équipe dirigée par W. F. King débuta par ce qu'elle appela la « Phase Zéro », une période d'exploration qui dura de 1974 à 1975. Le premier prototype, limité à un seul utilisateur, implémentait déjà les bases de ce qui deviendrait SQL. Cette version initiale s'appuyait sur le gestionnaire d'accès relationnel XRM développé par Raymond Lorie au Centre scientifique IBM de Cambridge. Les chercheurs découvrirent que transformer une théorie élégante en code fonctionnel soulevait des questions inattendues. Comment stocker efficacement les données ? Comment garantir l'intégrité des transactions ? Comment faire en sorte que les requêtes s'exécutent dans des temps raisonnables ?

La vraie révolution eut lieu pendant la « Phase Un », entre 1976 et 1977. L'équipe conçut une architecture à deux niveaux qui séparait clairement les responsabilités. Le RSS (Research Storage System) gérait tout ce qui touchait au stockage physique, aux verrous et à la journalisation des modifications. Au-dessus, le RDS (Relational Data System) s'occupait des autorisations et de l'optimisation des requêtes. Cette séparation n'était pas qu'une coquetterie d'ingénieur : elle rendait possible la maintenance et l'évolution du système.

Raymond Lorie proposa début 1976 l'idée de compiler les requêtes SQL au lieu de les interpréter. Traditionnellement, les systèmes de bases de données analysaient et exécutaient les requêtes à la volée, ce qui limitait leurs performances. L'approche de System R transformait chaque requête en code machine optimisé. Pour les applications qui répétaient les mêmes opérations des milliers de fois, le gain était spectaculaire. Pour les requêtes ponctuelles, le surcoût de compilation était largement compensé par la vitesse d'exécution.

L'optimiseur automatique de requêtes constituait le cerveau du système. Il analysait chaque demande, consultait les statistiques stockées dans le catalogue système, et choisissait la stratégie d'accès aux données la plus efficace. Son algorithme calculait le coût des entrées-sorties et du temps processeur pour différentes approches. Les index, sous forme de B-trees, garantissaient des accès rapides sur de gros volumes de données.

Le verrouillage hiérarchique réglait les problèmes d'accès concurrent avec une granularité variable. Un utilisateur pouvait verrouiller un seul enregistrement pour une modification ponctuelle, ou une table entière pour une opération de maintenance. Trois niveaux d'isolation permettaient aux développeurs de choisir entre cohérence stricte et performances. Le niveau

le plus exigeant garantissait que relire un enregistrement plusieurs fois dans une transaction donnait toujours un résultat identique.

La récupération après incident s'appuyait sur un mécanisme ingénieux de « pages fantômes ». Lors d'une modification, le système créait une nouvelle version de la page à un autre emplacement du disque, préservant l'ancienne jusqu'à la validation de la transaction. Des points de contrôle périodiques purgeaient les versions obsolètes. Si elle compliquait l'organisation physique des données, cette approche simplifiait considérablement la gestion des pannes.

La « Phase Deux », de 1978 à 1979, sortit System R des laboratoires pour l'installer chez de vrais utilisateurs. Des sites pilotes IBM et trois clients sélectionnés testèrent le système dans des conditions réelles. Ces expérimentations confirmèrent l'intuition initiale : les développeurs gagnaient en productivité grâce à l'uniformité du langage SQL. Qu'ils écrivent du code applicatif, lancent des requêtes ponctuelles ou définissent la structure des données, ils utilisaient la même syntaxe.

Les tests révélèrent aussi les limites du système. Pour des transactions simples manipulant peu de données, l'utilisation de tables de hachage ou de pointeurs directs aurait donné de meilleures performances. Le mécanisme des pages fantômes, malgré ses avantages pour la récupération, empêchait de regrouper efficacement les données liées sur le disque.

Un phénomène inattendu émergea lors des tests de montée en charge. L'équipe le baptisa « convoi » : quand un processus tenant un verrou très demandé était mis en attente longue par l'ordonnanceur, tous les autres processus se retrouvaient bloqués en chaîne derrière lui. La solution nécessita de repenser le protocole de libération des verrous pour éviter ces embouteillages.

System R transforma durablement l'industrie. Ses innovations architecturales – compilation des requêtes, optimisation automatique basée sur des statistiques, verrouillage hiérarchique – furent reprises par pratiquement tous les systèmes relationnels qui suivirent. En 1981, IBM commercialisa SQL/DS, son premier produit de base de données relationnelle, suivi deux ans plus tard par DB2 pour les mainframes. Oracle, Ingres et d'autres éditeurs développèrent leurs propres systèmes, démocratisant cette technologie.

L'impact dépassa largement la technique. System R prouva qu'un système informatique pouvait combiner simplicité d'usage et sophistication interne. Les utilisateurs manipulaient des concepts intuitifs – tables, lignes, colonnes – sans se soucier des mécanismes complexes qui garantissaient cohérence et performance. Cette séparation entre vue logique et physique devint un principe de l'informatique de gestion.

Le projet valida la vision d'Edgar F. Codd sur l'importance de l'abstraction dans la gestion des données. Il établit aussi l'importance d'une démarche expérimentale rigoureuse dans le développement technologique. Plutôt que de se contenter d'élégantes démonstrations théoriques, l'équipe de San José construisit un système complet et le testa dans des conditions réelles.

Face à l'émergence des systèmes NoSQL qui remettent en question certains aspects du modèle relationnel, les principes architecturaux de System R gardent leur pertinence. L'optimisation déclarative des requêtes, la gestion transactionnelle et la séparation des niveaux logique et physique restent des références.

Intel 8080

Dans l'effervescence technologique du début des années 1970, Intel tentait de s'imposer sur le marché des microprocesseurs. La société de Robert Noyce et Gordon Moore venait de lancer le 4004 en 1971, et l'année suivante le 8008, son premier processeur 8 bits. Mais ces premières réalisations, malgré leur caractère novateur, montraient leurs limites face aux attentes grandissantes du marché.

L'arrivée du physicien italien Federico Faggin dans les rangs d'Intel avait donné une impulsion décisive à ces développements. Son expertise en conception logique et en technologies MOS à grille silicium permit d'intégrer tous les circuits sur une seule puce, là où d'autres constructeurs peinaient à maîtriser cette complexité. Pourtant, dès 1972, les critiques s'accumulaient contre le 8008. Plessey et Nixdorf, deux entreprises informatiques européennes, pointaient du doigt une structure d'interruptions primitive, une multiplexion des adresses et des données pénalisante pour les performances, et une vitesse de fonctionnement famélique à 0,5 MHz.

Ces reproches résonnaient d'autant plus fort qu'ils émanaient de clients potentiels aux besoins réels. Le projet Sac State 8008, conçu pour gérer des dossiers médicaux, illustrait bien les possibilités du traitement 8 bits, mais révélait aussi les insuffisances du processeur face aux applications professionnelles. Intel comprit qu'une refonte complète s'imposait.

Le développement du 8080 débuta en novembre 1972 sous la direction de Masatoshi Shima, supervisé par Federico Faggin. Shima apportait au projet son expérience acquise chez Ricoh dans la conception d'interfaces entre périphériques et mini-ordinateurs. Sa méthode de travail tranchait

avec les pratiques habituelles : il créa des tables détaillées pour valider chaque instruction, appliquant une approche systématique de vérification logique qui préfigurait les méthodes modernes de conception.

Seize mois s'écoulèrent entre les premières esquisses et la commercialisation en janvier 1974. La répartition du temps révèle la méticulosité du processus : un mois pour définir le produit, huit mois pour concevoir la puce, et cinq mois pour fabriquer les masques, produire les wafers et déboguer l'ensemble. Cette rigueur porta ses fruits : le 8080 intégrait environ 5 000 transistors dans une architecture cohérente et performante.

La technologie NMOS retenue pour le 8080 provenait directement des développements d'Intel sur les mémoires dynamiques 4K. Cette filiation technique explique les trois tensions d'alimentation nécessaires : +5V, +12V et -5V, contrainte qui disparaîtra avec les générations suivantes. Le boîtier 40 broches, plus de deux fois supérieur aux 18 broches du 8008, témoignait de l'ambition du projet et libérait enfin le processeur des limitations de connectivité.

L'architecture interne révolutionnait cette approche. Shima abandonna la pile interne limitée du 8008 au profit d'un pointeur de pile en mémoire principale, donnant une flexibilité jusqu'alors inconnue. Les registres furent organisés selon une hiérarchie réfléchie : trois paires (BC, DE, HL) aux capacités différenciées, HL étant la plus puissante, BC la plus basique. Cette asymétrie, parfois critiquée par les programmeurs habitués aux architectures uniformes, résultait d'un choix délibéré d'optimisation des ressources matérielles.

Le jeu d'instructions s'enrichit considérablement tout en préservant une compatibilité au niveau assembleur avec le 8008. Innovation remarquable, le 8080 introduisit des opérations 16 bits : incrémentation, décrémentation, addition et soustraction. Shima privilégia cette approche générique plutôt que l'ajout d'un registre d'index spécialisé, choix débattu mais qui offrait une souplesse de programmation appréciable.

Les premiers tests de performance révélèrent l'ampleur du bond technologique. Deux programmes témoins – un transfert de 256 octets en mémoire et des additions décimales sur 16 chiffres – démontraient des gains de 10 à 20 fois par rapport au 8008 à fréquence identique. Porté à 2 MHz, le 8080 creusait l'écart avec son prédécesseur.

Intel accompagna le lancement d'une stratégie commerciale audacieuse. Le prix initial de 360 dollars l'unité pouvait sembler dissuasif, mais la société pariait sur la valeur ajoutée de son processeur. Ce pari se révéla gagnant : les constructeurs acceptaient cette prime pour bénéficier des capacités accrues. L'écosystème développé autour du 8080 renforça cette position : circuits spécialisés comme le 8224 (générateur d'horloge) et le 8228 (contrôleur

de bus), systèmes de développement Intellec, tout concourait à simplifier l'adoption du processeur.

Le succès de l'Altair 8800 de MITS, basé sur le 8080, confirma les intuitions d'Intel. Ce micro-ordinateur personnel, vendu en kit, ouvrit la voie à une nouvelle génération de machines accessibles aux particuliers passionnés. L'informatique personnelle trouvait là son premier véritable support technique.

Un défaut de conception faillit pourtant ternir ce succès. La ligne de masse principale, trop étroite, limitait l'utilisation aux circuits TTL basse consommation. Intel réagit en proposant la version corrigée 8080A, qui devint la référence du marché. Cette réactivité face aux problèmes techniques illustrait la maturité nouvelle de l'entreprise.

L'influence du 8080 dépassa largement les frontières d'Intel. Federico Faggin, parti fonder Zilog, conçut le Z80 en s'inspirant directement de son travail sur le 8080. Motorola répondit avec le 6800, architecture concurrente mais aux principes similaires. Cette émulation stimula l'innovation dans tout le secteur des microprocesseurs.

Au-delà de ses performances, le 8080 établit des standards durables : bus de données 8 bits séparé du bus d'adresses 16 bits, boîtier 40 broches, architecture de registres généralistes. Ces caractéristiques se retrouvèrent dans de nombreux processeurs ultérieurs, témoignage de la justesse des choix techniques de Shima et Faggin.

L'héritage du 8080 se mesure aussi à l'aune de l'industrie qu'il contribua à créer. En démontrant la viabilité commerciale des microprocesseurs 8 bits, il permettra à l'informatique personnelle de prendre toute sa place et confirma la position dominante d'Intel sur ce marché stratégique. Son architecture équilibrée et ses performances établirent des références qui guidèrent l'évolution technologique pendant des années.

CP/M

En 1972, Gary Kildall, alors consultant pour Intel via sa société Microcomputer Applications Associates, développe PL/M, un langage de programmation destiné aux microprocesseurs Intel. Ce langage, dérivé du langage XPL utilisé pour l'écriture de compilateurs, constitue le premier langage de haut niveau spécifiquement conçu pour la programmation des microprocesseurs. Le premier programme substantiel écrit en PL/M fut un éditeur de texte pour le microprocesseur Intel 8008, qui deviendra plus

tard l'éditeur ED de CP/M (Control Program for Microcomputers).

En 1974, Kildall perçoit le besoin d'un système d'exploitation pour exploiter les nouveaux lecteurs de disquettes 8 pouces de Shugart Associates. Cette année-là, le support de stockage dominant était la bande papier perforée, qui coûtait environ 100 fois plus cher que les disquettes à quantité de données équivalente. Kildall propose à Intel de développer un système d'exploitation complet incluant un éditeur, un assembleur et un chargeur, pour un montant de 20 000 dollars. Intel décline l'offre, mais fournit à Kildall le matériel nécessaire pour poursuivre son développement.

Le développement de CP/M s'effectue dans l'atelier situé derrière la maison de Kildall au 781 Bayview Avenue à Pacific Grove, Californie. Ne parvenant pas à réaliser seul l'interface électronique complexe entre le microprocesseur et le lecteur de disquettes, il fait appel à John Torode, un ami de l'université enseignant à Berkeley, qui conçoit le contrôleur nécessaire. C'est à l'automne 1974 que Kildall réussit à faire fonctionner CP/M pour la première fois, chargeant le système depuis une bande papier vers une disquette puis démarrant l'ordinateur depuis celle-ci.

L'une des innovations majeures de CP/M est sa conception modulaire séparant les fonctions dépendantes du matériel dans un composant distinct appelé BIOS (Basic Input/Output System). CP/M s'adapte ainsi facilement à différentes configurations matérielles en ne modifiant que le BIOS. Le reste du système, incluant le gestionnaire de fichiers et l'interpréteur de commandes, est identique quelle que soit la plateforme. Cette portabilité s'avère déterminante pour le succès commercial de CP/M.

En 1976, Gary et Dorothy Kildall fondent Digital Research Inc. (initialement nommée Intergalactic Digital Research) pour commercialiser CP/M. Les premières ventes s'effectuent depuis un bureau loué au 716 Lighthouse Avenue à Pacific Grove. Les disquettes CP/M sont vendues 70 dollars pièce, et le couple se rend quotidiennement à la poste pour récupérer les chèques résultant des publicités placées dans des magazines comme Byte et Dr. Dobb's Journal.

Un accord de licence avec IMS Associates, Inc. en 1977 accroît considérablement la crédibilité de CP/M dans l'industrie, faisant de ce système un standard adopté par la plupart des fabricants d'ordinateurs personnels de l'époque, dont Altair, Amstrad, Kaypro et Osborne. En 1978, le chiffre d'affaires atteint 100 000 dollars par mois et Digital Research s'installe dans une maison victorienne au 801 Lighthouse Avenue. En 1980, l'entreprise emploie plus de 20 personnes et génère un chiffre d'affaires de 3,5 millions de dollars, soit cinq fois celui de Microsoft à cette période.

En 1980, Bill Gates, qui connaissait Kildall pour avoir discuté d'une possible fusion entre leurs entreprises, recommande Digital Research à IBM

pour le système d'exploitation de son PC en cours de développement. Les négociations échouent sur des questions de confidentialité et de modèle commercial : IBM souhaite acheter CP/M sans redevances, tandis que Digital Research préfère un modèle par copie vendue. Microsoft saisit cette opportunité en achetant un clone de CP/M à Seattle Computer Products, renommé PC-DOS pour IBM et MS-DOS pour les autres constructeurs.

Face à cette situation, Kildall menace IBM d'un procès pour copie illégale de CP/M. IBM accepte de proposer CP/M-86 en option sur le PC, mais à 240 dollars contre 40 dollars pour PC-DOS. Cette différence de prix oriente massivement le marché vers PC-DOS/MS-DOS. Digital Research continue néanmoins à prospérer pendant quelques années, développant des systèmes d'exploitation multitâches pour l'ordinateur IBM PC-XT et introduisant des interfaces graphiques avant Apple et Microsoft.

Au milieu des années 1980, la domination de MS-DOS sur le marché des PC compatibles IBM affecte sérieusement l'activité de Digital Research. Gary Kildall, qui n'appréciait guère la gestion d'une grande entreprise, vend Digital Research à Novell en 1991. Les actifs sont ensuite revendus à Caldera en 1996, qui utilisera la propriété intellectuelle de DRI dans un procès contre Microsoft.

CP/M a connu des évolutions majeures. MP/M (Multi-Programming Monitor) ajoute des capacités multitâches et multi-utilisateurs. CP/NET, introduit en 1980, connecte des ordinateurs CP/M à des serveurs MP/M via divers protocoles réseau. Une version 16 bits, CP/M-86, est développée pour les processeurs Intel 8086/8088. Mais c'est la version 8 bits qui connaît le plus grand succès, avec environ 200 000 installations sur plus de 3000 configurations matérielles différentes.

CP/M établit des concepts fondamentaux comme la séparation entre le système d'exploitation et le matériel via le BIOS, l'utilisation de commandes standardisées, et la gestion de fichiers sur disque. Ces principes influencent directement la conception de MS-DOS et, par extension, l'évolution des systèmes d'exploitation personnels. En 1995, la Software and Information Industry Association décerne à Gary Kildall un prix pour l'ensemble de sa carrière, citant huit contributions majeures à l'industrie des micro-ordinateurs. En 2014, l'IEEE installe une plaque commémorative devant l'ancien siège de Digital Research, reconnaissant CP/M comme une étape importante dans l'histoire de l'informatique personnelle.

Le succès de CP/M illustre l'importance des standards ouverts et de la compatibilité logicielle dans le développement de l'industrie informatique. En permettant le développement d'un écosystème de logiciels indépendants du matériel, CP/M contribue à transformer les micro-ordinateurs d'outils spécialisés en plateformes généralistes accessibles à tous. Cette standardisa-

tion favorise l'émergence d'un marché du logiciel indépendant, préfigurant le modèle économique dominant de l'industrie informatique personnelle.

Diffie-Hellman

Dans les années 1970, la cryptographie vivait dans un monde clos. Les militaires et les gouvernements détenaient les clés de cette science secrète, laissant le reste de la société se débrouiller avec des communications peu sécurisées. Les systèmes d'alors imposaient une contrainte de taille : deux personnes souhaitant échanger des messages secrets devaient d'abord se rencontrer physiquement pour partager une clé commune. Imaginez devoir traverser l'Atlantique pour remettre en main propre un code secret avant de pouvoir envoyer un simple télégramme chiffré !

Whitfield Diffie et Martin Hellman, deux chercheurs de Stanford, ont bouleversé cette logique en 1976. Leur article *New Directions in Cryptography*, paru dans les IEEE Transactions on Information Theory, proposait l'impensable : créer un secret partagé entre deux inconnus sans qu'ils aient jamais échangé la moindre information confidentielle. Cette idée paraissait aussi absurde que de demander à deux personnes de choisir la même carte dans un jeu mélangé sans se concerter.

Le tour de force reposait sur l'arithmétique modulaire et les propriétés fascinantes des fonctions à sens unique. Diffie et Hellman ont exploité le fait qu'il est facile de calculer $g^x \bmod p$ mais extraordinairement difficile de retrouver x en connaissant seulement le résultat. Chaque participant génère son nombre secret, en calcule une version publique qu'il transmet ouvertement, puis combine cette information avec le nombre public de son correspondant. Par une sorte de magie mathématique, ils obtiennent tous deux un résultat secret identique.

Cette découverte a cassé les codes de la cryptographie traditionnelle. AT&T a vite saisi l'intérêt commercial de cette innovation en développant le système Common Channel Interoffice Signaling pour protéger leurs communications téléphoniques. Mais c'est avec l'explosion d'Internet dans les années 1990 que le protocole a trouvé sa véritable destinée. SSL, et son successeur TLS, ont intégré Diffie-Hellman au cœur de leurs mécanismes, transformant chaque connexion HTTPS en une démonstration vivante de cette prouesse mathématique.

La sécurité du protocole repose entièrement sur la difficulté du calcul des logarithmes discrets. Aucun ordinateur classique ne sait résoudre efficacement ce problème mathématique, ce qui explique pourquoi le protocole

résiste depuis près de cinquante ans. Toutefois, l'ordinateur quantique menace cet équilibre. L'algorithme de Shor pourrait un jour transformer ce problème réputé insoluble en un simple exercice de calcul.

Les mathématiciens n'ont pas chômé pour adapter et améliorer le protocole original. Victor Miller et Neal Koblitz ont proposé en 1985 une variante utilisant les courbes elliptiques, baptisée ECDH. Cette version réduit drastiquement la taille des clés nécessaires tout en conservant un niveau de protection constant. Un véritable gain d'efficacité qui a séduit de nombreux développeurs soucieux d'optimiser leurs applications.

Le protocole Diffie-Hellman a fait émerger RSA, développé par Rivest, Shamir et Adleman en 1978, et inspiré toute une génération de cryptographes. Cette innovation a aussi démocratisé la recherche en cryptographie, brisant le monopole des agences secrètes et donnant naissance à une communauté académique mondiale.

La reconnaissance officielle a pris son temps. Diffie et Hellman ont reçu le prix Turing en 2015 seulement, près de quarante ans après leur découverte. Cette attente s'explique probablement par le climat de guerre froide qui entourait toute innovation cryptographique. D'ailleurs, des chercheurs britanniques du GCHQ avaient développé des idées similaires quelques années plus tôt, mais leurs travaux sont restés classifiés jusqu'en 1997. Une belle illustration de la manière dont le secret d'État peut parfois retarder le progrès scientifique.

L'arrivée annoncée de l'informatique quantique mobilise aujourd'hui les cryptographes du monde entier. Ils travaillent d'arrache-pied sur des variantes post-quantiques du protocole, explorant de nouveaux problèmes mathématiques résistants aux futures machines quantiques. Ces recherches prolongent l'esprit pionnier de Diffie et Hellman, poursuivant leur vision d'une cryptographie accessible à tous. Ils ont prouvé qu'une approche ouverte et académique de la cryptographie pouvait rivaliser avec les laboratoires gouvernementaux les plus secrets. Ce nouveau paradigme intellectuel a libéré la créativité des chercheurs et donné naissance à une industrie florissante de la sécurité informatique.

Chaque fois que vous vous connectez à votre banque en ligne ou que vous achetez quelque chose sur Internet, vous bénéficiez de l'héritage de ces deux visionnaires. Leur protocole continue de protéger des milliards de communications quotidiennes, témoignage vivant d'une époque où deux chercheurs ont su imaginer l'impossible et le transformer en réalité.

Altair 8800

En janvier 1975, quand MITS (Micro Instrumentation and Telemetry Systems) lance l'Altair 8800, personne ne soupçonne que cette petite boîte métallique vendue moins de 400 dollars va bouleverser le monde informatique. Pourtant, ce premier ordinateur personnel à connaître un vrai succès commercial vient de naître dans les locaux d'une entreprise au bord de la faillite.

L'histoire débute avec H. Edward Roberts, un diplômé en génie électrique de l'Oklahoma State University qui a servi comme officier dans l'US Air Force. Au laboratoire d'armement d'Albuquerque, il rencontre Forrest Mims III. Ensemble, ils travaillent sur des projets de lasers et de fusées pour modèles réduits. Cette collaboration donne naissance à MITS en 1969, une société spécialisée dans les instruments de télémétrie pour l'aéromodélisme.

Puis l'entreprise se tourne vers les calculatrices électroniques et connaît quelques belles années. Mais en 1974, la concurrence des gros fabricants fait s'effondrer les prix. Roberts voit son chiffre d'affaires chuter brutalement. Il lui faut trouver autre chose, vite. C'est alors qu'il imagine un ordinateur personnel abordable, une idée qui paraît folle quand l'informatique représente l'apanage des grandes entreprises et des universités.

Le microprocesseur Intel 8080 vient d'arriver sur le marché. Ce composant 8 bits surpasse largement son prédécesseur, l'Intel 8008. Roberts y voit l'occasion de construire sa machine. L'architecture de l'Altair tire parti des capacités du processeur : un bus d'adressage 16 bits accédant à 65 000 mots mémoire, un jeu de 78 instructions de base, et surtout un système de bus extensible. La configuration de base ne propose que 256 mots de mémoire vive, extensible par l'utilisateur selon ses besoins et son budget.

Le panneau avant de l'Altair frappe par sa complexité apparente. Des rangées d'interrupteurs et de LED permettent la programmation directe en langage machine. Une approche spartiate qui décourage les non-initiés, mais qui correspond aux contraintes de coût de l'époque.

Roberts mise tout sur un article dans Popular Electronics de janvier 1975. Avec William Yates, il rédige un papier au titre accrocheur : « World's First Minicomputer Kit to Rival Commercial Models ». Le magazine sort en décembre 1974, et l'effet dépasse toutes les espérances. MITS espérait écouler quelques centaines d'exemplaires ; les commandes arrivent par milliers dès les premiers jours.

Cette déferlante révèle l'existence d'un marché que personne n'avait vraiment imaginé. L'Altair attire deux jeunes étudiants de Harvard, Paul Allen et Bill Gates, qui développent un interpréteur BASIC pour la machine. C'est la naissance de Microsoft. En mars 1975, le Homebrew Compu-

ter Club se forme autour de l'Altair dans la Silicon Valley. Steve Wozniak y fait ses premières armes et y présente les travaux qui mèneront à la création d'Apple Computer.

L'influence technique de l'Altair va jusqu'à rendre son bus système une norme connue sous le nom de S-100. De nombreux fabricants adoptent cette architecture pour leurs propres machines. L'Altair utilise le système d'exploitation CP/M, qui dominera le marché des micro-ordinateurs jusqu'à l'arrivée du PC IBM et de MS-DOS. L'approche modulaire, avec ses cartes d'extension, inspire toute une génération d'ordinateurs.

Les limites de la machine reflètent l'état de la technologie en 1975. Pas d'écran, pas de clavier dans la version standard. La programmation passe par les interrupteurs du panneau avant, exercice laborieux qui rebute les néophytes. La mémoire est peu abondante et il faut attendre les extensions pour disposer de vrais moyens de stockage. Mais ces contraintes stimulent la créativité : une multitude de fabricants développent des cartes et des périphériques compatibles, créant un véritable écosystème.

MITS découvre les joies et les peines du succès inattendu. L'entreprise peine à honorer les commandes et à maintenir un niveau de qualité acceptable sur des kits assemblés à la main. Le support technique, assuré notamment par le bulletin Computer Notes, ne suffit pas toujours face aux questions d'utilisateurs parfois dépassés par la complexité de leur acquisition. Ces difficultés de jeunesse n'empêchent pas le succès, mais elles illustrent les défis de la création d'un nouveau marché.

L'aventure de l'Altair s'achève en 1977 avec la vente de MITS à Pertec Computer Corporation. La production s'arrête, mais l'héritage perdure. La machine a prouvé qu'un ordinateur personnel pouvait être commercialement viable et techniquement crédible. Elle a donné naissance à une communauté de passionnés qui préfigure la culture informatique moderne.

L'Altair 8800 marque la frontière entre deux mondes : celui des ordinateurs institutionnels, réservés aux initiés, et celui de l'informatique personnelle, promise à tous. Son architecture ouverte et modulaire pose les bases de principes qui gouvernent encore l'industrie. L'enthousiasme qu'il déclenche parmi les amateurs transforme une curiosité technique en mouvement culturel. Plus qu'une prouesse d'ingénierie, l'Altair démontre qu'un autre rapport à l'informatique est possible.

X.25

En 1976, quand le Comité consultatif international téléphonique et télégraphique adopte X.25, personne ne se doute que ce protocole survivra plus de trente ans. Pourtant, cette norme technique née des débats acharnés entre ingénieurs français et britanniques va façonner durablement l'architecture des réseaux de données.

Les premières interrogations sur l'avenir des communications numériques émergent. Arpanet fait ses premiers pas outre-Atlantique depuis 1969, mais en Europe, la situation diffère radicalement. Les administrations des télécommunications règnent sans partage sur les infrastructures. En France, la Direction générale des télécommunications tient les rênes, tout comme le British Post Office au Royaume-Uni. Ces monopoles publics ne voient pas d'un bon œil l'anarchie relative qui caractérise les développements américains.

La commutation par paquets cristallise les passions techniques. Cette méthode, qui découpe les messages en petites portions pour optimiser leur circulation, divise la communauté scientifique. D'un côté, Rémi Després du CNET défend mordicus les circuits virtuels : chaque communication emprunte un chemin prédéterminé dans le réseau, ce qui garantit un certain ordre. De l'autre, Louis Pouzin et son équipe de l'IRIA militent pour les datagrammes : les paquets naviguent librement, trouvant leur route au gré des disponibilités du réseau.

Cette querelle technique cache des enjeux bien plus vastes. Les circuits virtuels séduisent les opérateurs téléphoniques traditionnels par leur prévisibilité. Ils rappellent l'univers familier de la commutation téléphonique classique, où chaque appel établit un circuit dédié. Les datagrammes, eux, incarnent une philosophie différente, plus proche de l'esprit libertaire qui souffle sur les campus américains.

Le processus d'adoption au CCITT ressemble à une partie d'échecs diplomatique. Larry Roberts, pourtant père d'Arpanet, soutient paradoxalement les circuits virtuels depuis sa société privée. Mais sans droit de vote, son influence est limitée. Situation similaire pour les Canadiens de Bell Canada, qui développent pourtant Datapac sur cette base technique.

Le tournant survient avec le ralliement inattendu du British Post Office aux thèses françaises. Halvor Bothner, le rapporteur norvégien, abandonne alors sa position initiale. Lui qui avait forgé le terme « datagramme » se range finalement du côté des circuits virtuels. Cette volte-face scelle le sort de X.25, adopté en mars 1976 puis confirmé solennellement à l'automne.

L'ironie veut que cette victoire française profite immédiatement à l'international. L'Espagne et les Pays-Bas bâtissent leurs réseaux nationaux sur

X.25. Bell Canada l'intègre dans Datapac, validant ainsi les choix techniques européens. IBM, toujours pragmatique, adapte dès 1975 ses équipements à cette norme.

En France, X.25 trouve son apothéose avec Transpac. Ce réseau national supportera pendant plus de trois décennies le trafic du Minitel, de 1980 à 2012. Performance remarquable pour une technologie que d'aucuns jugeaient dépassée dès les années 1990. Les banques, en particulier, lui restent fidèles : sa fiabilité et sa sécurité intrinsèques correspondent parfaitement à leurs exigences strictes.

L'émergence d'Internet aurait pu sonner le glas de X.25. TCP/IP, héritier direct des datagrammes de Pouzin, s'impose progressivement comme la norme de fait. Mais X.25 résiste dans ses créneaux spécialisés. Le Costa Rica l'utilisait encore en 1999, la France jusqu'en 2012. Cette longévité contraste avec l'échec retentissant de l'OSI, ce projet pharaonique de l'ISO qui prétendait normaliser l'ensemble des communications informatiques.

X.25 illustre parfaitement les caprices de l'innovation technique. Une norme peut naître d'un compromis politique, s'épanouir dans des usages inattendus et survivre bien au-delà de ses prévisions initiales. Elle témoigne aussi de la capacité des ingénieurs français à peser dans les débats techniques internationaux, quand la qualité technique rencontre l'habileté diplomatique.

Cette histoire révèle surtout la complexité des écosystèmes technologiques. Contrairement aux idées reçues, la « meilleure » solution technique ne s'impose pas toujours. X.25 et TCP/IP ont cohabité pendant des décennies, chacun répondant à des besoins spécifiques. Cette coexistence paisible tranche avec les guerres de standards qui agitent régulièrement le monde informatique.

Cray-1

L'année 1976 voit naître une machine qui va redéfinir les limites du calcul scientifique. Le Cray-1, fruit de l'obsession technique de Seymour Cray, émerge d'un parcours déjà remarquable dans l'univers des supercalculateurs. Cray avait signé auparavant les Control Data 6600 et 7600, deux références qui avaient installé sa réputation d'architecte visionnaire. Mais cette fois, il s'agit d'autre chose : construire la machine la plus rapide jamais conçue.

En 1972, Cray quitte Control Data pour créer sa propre société, Cray

Research. L'objectif affiché tient en quelques mots : bâtir l'ordinateur le plus puissant au monde. Quatre années de développement intensif aboutissent à une réalisation spectaculaire. Le Cray-1 frappe d'abord par son allure : un cylindre de 2,75 mètres de diamètre, haut de près de 2 mètres, qui tranche avec l'esthétique rectangulaire habituelle des ordinateurs. Cette forme circulaire n'a rien d'un caprice esthétique. Elle répond à une contrainte technique implacable : raccourcir au maximum les liaisons entre composants pour gagner quelques nanosecondes précieuses.

Les chiffres donnent le vertige. La machine atteint 80 millions d'opérations en virgule flottante par seconde en fonctionnement normal, avec des pointes à 250 millions. Une performance qui pulvérise tous les records établis. Cette puissance brute s'appuie sur une innovation décisive : l'architecture vectorielle. Le Cray-1 traite simultanément des séries entières de données grâce à une technique de « chaînage » qui orchestre huit registres vectoriels de 64 éléments chacun.

Sous son habillage futuriste, la machine cache une philosophie de conception d'une simplicité déconcertante. Seymour Cray n'utilise que quatre types de circuits intégrés différents : des portes NAND 5/4, des puces mémoire et des registres, tous basés sur la technologie ECL. Cette approche minimaliste dissimule pourtant une complexité vertigineuse : 1 662 modules répartis sur 113 types différents, chaque module pouvant embarquer jusqu'à 288 circuits intégrés. L'ensemble mobilise l'équivalent de 2,5 millions de transistors.

La densité électronique génère un défi thermique redoutable. Concentrer autant de composants dans un espace aussi réduit produit une chaleur que les systèmes de refroidissement traditionnels ne peuvent évacuer. Cray développe alors une solution au fréon d'une sophistication remarquable. Des barres de refroidissement en aluminium et acier inoxydable serpentent dans chaque colonne de la machine, maintenant une température stable de 21°C. Cette prouesse technique conditionne le fonctionnement fiable de l'ensemble.

La mémoire atteint un million de mots de 64 bits, organisée en 16 bancs indépendants. Cette architecture offre la possibilité d'accéder simultanément aux données depuis plusieurs processus, réduisant les temps d'attente qui pénalisent les performances. Un système SECDED protège l'intégrité des calculs en corrigeant automatiquement les erreurs simples et en détectant les erreurs doubles.

Le succès commercial dépasse toutes les prévisions. Entre 1976 et 1982, Cray Research écoule environ 80 machines au prix unitaire de 19 millions de dollars. Les premiers clients comptent des centres de recherche gouvernementaux américains, mais aussi des institutions européennes comme

le Centre Européen pour les Prévisions Météorologiques à Moyen Terme. Cette dimension internationale confirme l'avance technologique prise par Cray.

Les domaines d'application révèlent la polyvalence de la machine. Météorologie, aéronautique, recherche nucléaire : partout où la simulation numérique exige des calculs intensifs en virgule flottante, le Cray-1 excelle. Sa capacité à résoudre rapidement des systèmes d'équations différentielles complexes ouvre de nouvelles perspectives à la modélisation scientifique.

L'influence technique va bien au-delà des performances brutes. Le système d'exploitation et le compilateur FORTRAN, spécialement optimisés pour exploiter l'architecture vectorielle, établissent de nouveaux standards. La programmation s'avère plus accessible que sur des concurrents comme l'ILLIAC IV, favorisant l'adoption par la communauté scientifique.

Le développement s'accompagne d'enjeux considérables. La mise au point du système de refroidissement mobilise dix-huit mois de recherche intensive. La fabrication des cartes à cinq couches, avec leurs interconnexions d'une précision extrême, repousse les limites des techniques. Ces contraintes stimulent des innovations dont bénéficie l'ensemble de l'industrie.

En 1990, quatorze ans après sa mise en service, le Lawrence Livermore National Laboratory retire son Cray-1, dépassé par les nouvelles générations. La machine est vendue aux enchères en 1993 pour 10 000 dollars, soit 0,05% de son prix initial. Cette chute vertigineuse illustre l'accélération du progrès technologique dans les supercalculateurs.

Le Cray-1 cristallise une époque où la recherche de la performance absolue guidait l'innovation. Son influence perdure dans les principes de conception actuels : optimisation des communications inter-composants, gestion thermique sophistiquée, architecture simplifiée malgré la complexité des défis techniques. La machine incarne la vision de Seymour Cray, qui privilégiait l'élégance et la cohérence dans la quête de l'excellence technique.

Cette réalisation démontre qu'une architecture audacieuse, soutenue par une ingénierie méticuleuse, a les moyens de repousser spectaculairement les frontières du calcul scientifique. Les principes établis par le Cray-1 confirment sa place d'étape fondatrice dans l'évolution de l'informatique de haute performance.

UUCP

En 1976, Mike Lesk travaille sur un problème concret aux Bell Labs. L'ARPANET existe déjà, certes, mais son accès demeure l'apanage d'un cercle restreint : en janvier de cette année, seuls 63 hôtes y sont connectés. Pour les utilisateurs d'UNIX dispersés aux quatre coins des États-Unis, échanger des fichiers ou exécuter des commandes à distance relève du parcours du combattant. Lesk conçoit UNIX-to-UNIX Copy Protocol (UUCP), une solution pragmatique qui transforme les lignes téléphoniques ordinaires en artères numériques.

La première mouture d'UUCP intègre UNIX en février 1978. Le protocole fonctionne sur des liaisons à 300 bauds, une vitesse qui paraît dérisoire mais qui représentait un progrès notable. Quelques mois plus tard, en octobre, la Seventh Edition d'UNIX embarque une version remaniée du protocole. Lesk s'associe cette fois à Dave Nowitz, avec l'aide de Greg Chesson, pour corriger les défauts de jeunesse du système initial.

Fin 1979 à l'Université de Caroline du Nord, Steve Bellovin, étudiant cherchant à maîtriser les subtilités du système, développe un modeste programme de diffusion de nouvelles. Parallèlement, Tom Truscott et Bellovin expérimentent une liaison UUCP entre l'UNC et Duke University. De cette expérimentation naît une idée qui va transformer l'usage d'UUCP : pourquoi ne pas distribuer des informations vers d'autres sites en utilisant Duke comme plaque tournante ? Les sites éloignés remboursaient Duke pour les frais de téléphone, un système rudimentaire mais efficace.

Au début de 1980, le réseau compte trois participants : UNC, Duke University et le département de physiologie du centre médical de Duke. Jim Ellis présente le concept lors d'une réunion USENIX à Boulder en janvier. L'accueil est favorable. Steve Daniels produit une implémentation du logiciel baptisé A News, distribuée sur la bande USENIX de l'été 1980 à Newark. Le réseau s'étend à 15 sites.

Armando Stettner et Bill Shannon, travaillant chez Digital Equipment Corporation, proposent un marché séduisant à l'Université de Californie à Berkeley : ils prennent en charge les coûts de connexion de Berkeley en échange d'un accès au réseau via leur machine decvax, située dans le New Hampshire. Stettner ne s'arrête pas là et finance les premières connexions internationales vers l'Europe, le Japon et l'Australie.

Le réseau reçoit le nom d'Usenet, en hommage à USENIX. La croissance dépasse toutes les prévisions : en un an, plus de 100 sites échangent environ 25 articles quotidiens. Cette expansion révèle les faiblesses du système initial. Mark Horton, étudiant à Berkeley, s'associe à Matt Glickman, alors lycéen, pour réécrire intégralement A News. Leur création, B News, voit le

jour en 1981. Horton continue d'affiner son œuvre jusqu'en 1984, avant de passer le relais à Rick Adams du Center for Seismic Studies.

En juin 1984, une carte géographique d'USENET révèle l'ampleur du phénomène. Les connexions s'étendent désormais jusqu'en Australie, à Hawaï, au Canada et en Europe. La version 2.11 de B News, parue en 1986, intègre les contributions de Rick Adams, Spencer Thomas, Ray Essick et Rob Kolstad. Bien que de nouvelles versions paraissent jusqu'en 1994, le système montre ses limites dès 1989.

Le protocole NNTP (Network News Transfer Protocol), publié en février 1986 par Brian Kantor et Phil Lapsley, annonce une nouvelle ère. Un an plus tard, Geoff Collyer et Henry Spencer de l'Université de Toronto lancent C News, une alternative plus performante qui enterre définitivement les anciens systèmes.

La question financière obsède les acteurs du réseau. Lauren Weinstein présente en 1984 lors de la conférence USENIX d'été le projet Stargate, une proposition audacieuse qui consiste à utiliser les portions inutilisées du signal vidéo télévisé pour transporter des données ASCII à 65 kbps. Malgré le soutien de plusieurs entreprises et des tests concluants, le projet s'enlise et finit par être abandonné.

Face aux insuffisances des réseaux commerciaux existants, Rick Adams propose à USENIX en 1985 de créer un site central accessible via Tymnet. UUNET naît en 1987 avec un budget initial de 35 000 dollars. Le service répond à un besoin réel : nombreux sont ceux qui perdent leur accès e-mail et Usenet en changeant d'employeur. Le succès dépasse les espérances. Installé sur un ordinateur Sequent B21 à 16 processeurs, UUNET compte plus de 50 abonnés dès juin 1987 et quelques milliers cinq ans plus tard.

UUNET participe en 1991 à la création de la Commercial Internet Exchange Association. L'année suivante, l'entreprise co-fonde MAE-East avec Metropolitan Fiber Systems, point d'échange Internet le plus important au monde l'espace de quelques temps. L'entrée en bourse intervient en mai 1995, consacrant le succès commercial du concept.

La réussite d'UUCP tient à ses caractéristiques techniques novatrices. Le système fonctionne de manière asynchrone, avec une mise en file d'attente des tâches. Les transmissions s'effectuent en arrière-plan, s'accommodant des contraintes des lignes téléphoniques souvent occupées. Le protocole ne nécessite aucune modification du système d'exploitation et s'adapte aussi bien aux connexions directes qu'aux modems.

L'aspect sécuritaire n'est pas négligé. Chaque site configure les fichiers et commandes accessibles aux systèmes distants. L'authentification repose sur des mots de passe, et certains sites utilisent le rappel automatique pour vérifier l'identité des appelants. Des compteurs de séquence peuvent être

activés pour contrer les tentatives d'usurpation d'identité.

UUCP perdure jusqu'aux années 1990, notamment dans les situations où une connexion Internet permanente est inaccessible ou trop onéreuse. Sur les lignes séries lentes, il se montre plus efficace que TCP/IP. Son architecture par lots convient parfaitement au transport du courrier électronique et des news Usenet.

L'héritage d'UUCP traverse les décennies. La syntaxe des adresses e-mail (user@host) lui doit son existence, tout comme certains mécanismes de routage du courrier électronique. Le protocole démontre qu'il est possible de bâtir des réseaux informatiques décentralisés à grande échelle en s'appuyant sur les infrastructures de télécommunication existantes. Son influence se retrouve dans le développement ultérieur d'Internet et des technologies de communication asynchrone.

Apple I

C'était en 1975, dans ce bouillonnement intellectuel qu'était le Homebrew Computer Club de la Silicon Valley. Steve Wozniak, figure discrète mais passionnée de ces réunions d'amateurs d'électronique, avait bricolé quelque chose d'inhabituel : un ordinateur qu'il voulait présenter aux autres membres du club. Son ami Steve Jobs, rencontré quatre ans plus tôt, flaire immédiatement autre chose qu'un simple projet de passionné.

Le cœur de cette machine repose sur un microprocesseur MOS Technology 6502 qui tourne à 1,023 MHz. Wozniak a eu l'idée d'intégrer une interface vidéo directement dans son système, capable d'afficher 24 lignes de 40 caractères sur n'importe quel téléviseur standard. C'était loin d'être évident. La machine embarque 4 Ko de mémoire vive, extensible jusqu'à 8 Ko, avec des registres à décalage dynamiques qui gèrent l'affichage.

Tout bascule quand Paul Terrell débarque. Ce propriétaire du Byte Shop, une boutique informatique de Mountain View, commande carrément 50 machines d'un coup. Jobs et Wozniak se regardent : ils n'ont ni l'argent ni l'infrastructure pour produire quoi que ce soit à cette échelle. Wozniak sacrifie sa précieuse calculatrice HP-65, Jobs sa camionnette Volkswagen. Avec cet argent et un crédit de 30 jours négocié auprès des fournisseurs, ils se lancent dans l'aventure.

Contrairement à la légende du garage familial, la réalité de la production se révèle plus prosaïque. Les cartes sont manufacturées et soudées par vague dans une véritable usine, probablement Santa Clara Circuits. Le fameux ga-

rage des parents Jobs sert surtout aux tests finaux et à l'assemblage. Daniel Kottke, copain de Jobs, passe ses soirées à tester les machines pendant que Wozniak intervient pour dépanner les cas les plus tordus.

L'Apple I arrive sur le marché sous une forme qui surprend. Pas de boîtier, pas de clavier, pas d'écran : juste une carte mère nue que l'acheteur doit compléter selon ses moyens et ses envies. Paul Terrell tombe des nues quand il reçoit sa commande. Il s'attendait à des ordinateurs complets, pas à des cartes électroniques. Le prix affiché fait 666,66 dollars, non par provocation mais parce que Wozniak adore les chiffres qui se répètent.

Côté logiciel, l'Apple I reste spartiate. Un petit moniteur en mémoire morte autorise la saisie de programmes directement en langage machine. Wozniak développe ensuite son propre interpréteur BASIC, inspiré de son expérience chez HP avec les calculatrices. Mais ce BASIC traîne quelques casseroles : impossible de manipuler des décimales ou des nombres au-delà de 32 767, une limitation qui embarrassera aussi la première mouture de l'Apple II.

L'interface cassette audio, ajoutée plus tard, tourne au cauchemar. Liza Loop, première acheteuse officielle, galère comme beaucoup d'autres utilisateurs avec son Apple I qui refuse de coopérer. Wozniak, génial sur le numérique, patauge un peu quand il faut s'attaquer à l'électronique analogique. L'Apple II bénéficiera heureusement du renfort d'autres ingénieurs pour corriger le tir.

Pourtant, Wozniak glisse ses trouvailles techniques remarquables dans sa conception. Il détourne astucieusement les compteurs du circuit vidéo pour rafraîchir la mémoire vive, une technique qu'il affinera sur l'Apple II. Le circuit vidéo garde ses mystères : certaines parties restent obscures, y compris pour Wozniak, qui s'est peut-être inspiré du « TV Typewriter » de Don Lancaster, publié dans *Radio-Electronics* en 1973.

Au final, moins de 200 Apple I sortent des chaînes de production. Un score modeste qui ne présage en rien du raz-de-marée qui suivra. Aujourd'hui, entre 30 et 50 exemplaires survivent dans les collections privées ou les musées. Certains proviennent d'employés d'Apple qui les ont récupérés dans les années 1970 parmi les invendus ou les retours clients.

L'Apple I n'a jamais été un succès commercial, mais il forge quelque chose d'infiniment plus précieux : l'expérience. Jobs et Wozniak apprennent sur le tas les ficelles de la conception, de la production et de la vente d'ordinateurs. Chaque défaut de l'Apple I est une leçon pour l'Apple II, qui explosera tous les compteurs avec plus de 6 millions d'exemplaires écoulés sur seize ans.

Cette première machine, au-delà d'une anecdote technique, matérialise l'idée qu'un ordinateur est en mesure de s'adresser au grand public, sor-

tir des laboratoires et des universités pour atterrir chez monsieur « Tout-le-monde ». L'Apple I ouvre une brèche dans laquelle s'engouffreront des millions d'autres machines.

Il témoigne aussi de cette époque bénie où quelques passionnés d'électronique, armés de leur seule ingéniosité et de quelques milliers de dollars, pouvaient chambouler un secteur entier. L'histoire de sa création cristallise parfaitement cette alchimie entre le génie technique de Wozniak et l'instinct commercial de Jobs, une recette qui propulsera Apple vers des sommets inimaginables.

vi

En 1976, à l'université de Berkeley, Bill Joy se retrouve face à un problème que connaissent tous les programmeurs de cette période : comment éditer du texte de manière décente sur un système UNIX ? Les outils disponibles, comme *ed*, relèvent plus de la torture intellectuelle que de l'aide à la programmation. Peter H. Salus n'hésite d'ailleurs pas à les qualifier d'« éditeurs les plus hostiles jamais créés ». Imaginez-vous travailler ligne par ligne, sans jamais voir l'ensemble de votre texte !

Joy, étudiant diplômé, ne se contente pas de subir cette situation. Il travaille déjà sur un interpréteur Pascal et collabore avec Chuck Haley pour améliorer *ed*. L'université dispose d'un PDP-11, machine bien modeste comparée aux PDP-10 que s'offrent MIT ou Stanford. Cette contrainte matérielle les pousse à innover différemment, à chercher l'efficacité plutôt que la puissance brute.

L'été 1976 apporte un déclic inattendu. Joy et Haley découvrent *em* (Editor for Mortals), création de George Coulouris à Queen Mary College. Voilà un éditeur qui ose afficher deux messages d'erreur au lieu d'un seul et propose une approche plus visuelle de l'édition. Rien de fulgurant en soi, mais suffisant pour déclencher l'inspiration. Les deux complices développent alors *ex*, dont le mode « visual » donnera naissance au nom « `vi` ».

La genèse de `vi` tient du hasard autant que du génie. Joy se souvient avoir conçu l'interface un samedi de 1976, la radio en fond sonore diffusant les questions-réponses de Jimmy Carter fraîchement élu. Cette anecdote révèle l'atmosphère détendue dans laquelle naît un outil destiné à marquer l'histoire de l'informatique. L'éditeur intègre ses innovations tout en s'adaptant aux terminaux existants, notamment les ADM-3A. Le fameux système de modes – normal, insertion, commande – découle naturellement de ces contraintes techniques et de l'art de tirer le meilleur parti de chaque touche.

Joy ne s'embarrasse pas de circuits commerciaux sophistiqués pour diffuser sa création. Il propose des copies sur bandes magnétiques 9 pistes pour 50 dollars, dont 40 servent à financer le développement. Cette distribution artisanale rencontre un succès inattendu : des centaines de copies partent ainsi vers des universités et laboratoires. La supériorité technique de `vi` sur ses concurrents explique cet engouement, renforcé par sa distribution avec BSD, la version Berkeley d'UNIX.

L'arrivée des VAX en 1977-1978 donne un second souffle au projet. Joy adapte `vi` aux capacités de ces nouvelles machines, exploitant notamment la mémoire virtuelle. Le développement progresse bien jusqu'à un incident technique qui change tout : une panne de lecteur de bande fait perdre le code source. Les fonctionnalités avancées prévues, comme le support multi-fenêtres, sont abandonnées. Joy préfère stabiliser l'existant et produire une documentation correcte.

L'histoire prend une tournure surprenante quand AT&T incorpore `vi` dans UNIX System V. Joy apprend cette nouvelle après coup, découvrant son éditeur comme standard dans tout l'univers UNIX. Cette décision s'explique par l'adoption massive de `vi` dans les Bell Labs eux-mêmes. Paradoxalement, cette popularité créait des problèmes : chaque équipe maintenait sa propre copie, gaspillant la mémoire des machines.

`vi` établit une philosophie d'édition qui divise encore. Son système modal demande un apprentissage initial conséquent, mais offre ensuite une puissance de manipulation textuelle inégalée. Les uns le trouvent trop complexe, les autres ne jurent que par son efficacité une fois maîtrisé. Cette dualité traverse les décennies sans perdre de sa pertinence.

Les années 1990 voient naître `vim` (Vi IMproved), qui prolonge l'héritage de l'éditeur original tout en ajoutant les fonctionnalités attendues par les utilisateurs modernes. `vi` est présent sur tous les systèmes UNIX et leurs descendants, Linux et macOS inclus. Son influence dépasse ces frontières et nombre d'éditeurs modernes proposent un « mode `vi` » pour séduire ses adeptes.

Cette pérennité étonne dans un domaine où l'obsolescence est rapide. Un outil conçu dans les années 1970 continue de servir quotidiennement des millions d'utilisateurs. Cette longévité tient à son efficacité redoutable pour certaines tâches, sa présence universelle, et sa fiabilité éprouvée. Les administrateurs système et programmeurs apprécient sa disponibilité garantie et sa stabilité à toute épreuve.

L'aventure de `vi` illustre l'importance des choix initiaux dans la conception logicielle. Les contraintes qui ont guidé Joy – économie de mémoire, exploitation optimale du clavier, compatibilité avec les terminaux limités – ont produit un outil dont l'utilité traverse les époques technologiques.

Cette réussite rappelle qu'un bon logiciel ne se juge pas seulement à ses fonctionnalités, mais à sa cohérence conceptuelle et à sa réponse aux besoins fondamentaux des utilisateurs. Cinquante ans après sa création, `vi` continue de prouver cette vérité tous les jours.

DEC VAX-11

Au milieu des années 1970, les ordinateurs centraux règnent encore sur les grandes organisations, tandis que les mini-ordinateurs 16 bits de Digital Equipment Corporation dominent un marché en pleine expansion. Mais chez DEC, quelques ingénieurs visionnaires comprennent qu'ils touchent aux limites de cette architecture. Les programmes grossissent, les données se multiplient, et les 16 bits montrent leurs faiblesses.

En mars 1975, une équipe se forme discrètement dans les bureaux de Maynard, Massachusetts. Leur mission : concevoir une architecture 32 bits innovante. Le projet porte des noms de code poétiques : « Star » pour le matériel, « Starlet » pour le système d'exploitation. Cette approche inhabituelle frappe d'emblée : contrairement aux habitudes où matériel et logiciel évoluent séparément, les équipes travaillent main dans la main depuis le premier jour.

Trois cents années-homme de développement intensif s'écoulent avant que DEC ne dévoile son chef-d'œuvre. Le 25 octobre 1977, lors de l'assemblée annuelle des actionnaires, la présentation du VAX-11/780 et de son système VMS marque les esprits. La démonstration choisie frôle l'audace : l'ordinateur affronte un humain au Scrabble. Le mot « sensibly » lui rapporte 127 points et la victoire. Ce clin d'œil ludique cache une réalité technique impressionnante : le VAX-11/780 combine la puissance d'un ordinateur central, l'interactivité d'un mini-ordinateur et un prix enfin abordable.

L'architecture du VAX-11/780 bouscule les conventions. La mémoire virtuelle sur 32 bits libère les programmeurs des contraintes d'adressage qui bridaient leurs ambitions. La correction d'erreur intégrée dans la mémoire, une première mondiale, transforme la fiabilité des systèmes. DEC assume une philosophie audacieuse : concevoir une machine destinée à durer quinze à vingt ans, tournant le dos à l'obsolescence programmée que pratiquent déjà ses concurrents.

Le marché répond avec enthousiasme. Dès 1979, les ventes de DEC franchissent le cap des 2 milliards de dollars. Les universités adoptent massivement ces machines pour leurs laboratoires de recherche. Les entreprises

découvrent un outil capable de traiter leurs applications scientifiques les plus exigeantes. VMS, le système d'exploitation, impressionne par sa robustesse et ses fonctionnalités avancées.

La famille VAX s'étend progressivement. Le VAX-11/750 arrive en 1980 avec sa technologie de portes logiques semi-personnalisées. Deux ans plus tard, le VAX-11/730 démocratise l'accès à cette architecture. En 1984, le VAX 8600 établit de nouveaux records de performance avec une puissance quadruplée par rapport au modèle original.

Une innovation remarquable surgit en 1983 : les VAXclusters. Pour la première fois dans l'industrie, plusieurs ordinateurs VAX peuvent se connecter en réseau et former un système unique. Cette prouesse technique révolutionne la haute disponibilité. Les bourses, les systèmes de contrôle industriel, tous les environnements critiques s'emparent de cette technologie qui tolère les pannes comme aucune autre.

En 1985, le MicroVAX concentre toute l'architecture VAX sur une seule puce, rendant possible des systèmes plus compacts et moins onéreux. Le succès dépasse toutes les prévisions : 20 000 MicroVAX II trouvent preneur durant la première année de commercialisation.

L'évolution technique se poursuit à un rythme soutenu. Le CVAX de 1987 adopte la technologie CMOS, plus économe en énergie. Le processeur Rigel en 1989, puis le NVAX en 1991, repoussent sans cesse les limites de performance. Ce dernier atteint une vitesse trente fois supérieure à celle du VAX-11/780 qui avait tout commencé quatorze ans plus tôt.

VMS façonne l'évolution des systèmes d'exploitation. Conçu sur mesure pour le VAX, il établit des standards inédits de fiabilité et de sophistication fonctionnelle. L'intégration native de DECnet anticipe l'importance croissante des communications entre machines, préfigurant l'ère des réseaux informatiques.

Le début des années 1990 voit émerger les processeurs RISC 64 bits. DEC prépare sa transition vers l'architecture Alpha sans abandonner son héritage. VMS, rebaptisé OpenVMS, migre vers cette nouvelle plateforme tout en conservant le support des anciens systèmes. L'entreprise maintient sa production VAX, sortant le modèle 4000 en 1996.

Le succès du VAX repose sur un savant mélange d'ingrédients : puissance de calcul, fiabilité légendaire, système d'exploitation sophistiqué et support technique exceptionnel de DEC. Les anecdotes abondent sur cette robustesse hors norme. Ce VAX-11/780 qui continue de fonctionner après sa chute d'un chariot élévateur. Ces systèmes qui tournent des années sans interruption, défiant les lois de l'usure électronique.

L'architecture 32 bits du VAX influence la conception de nombreux processeurs modernes. Les concepts de VMS, gestion avancée de la mémoire

virtuelle ou *clustering*, se retrouvent dans nos systèmes d'exploitation actuels. Cette philosophie de conception intégrée matériel-logiciel s'impose désormais comme une évidence dans toute l'industrie.

Le VAX témoigne d'une époque révolue où l'innovation informatique privilégiait la vision long terme et l'excellence technique plutôt que la course aux nouveautés. Sa longévité exceptionnelle – certains systèmes VAX fonctionnent encore de nos jours – valide cette approche. Au-delà de l'objet technique, le VAX incarne une étape fondatrice de l'histoire informatique, ce moment charnière entre l'ère des ordinateurs centraux et celle des systèmes distribués qui façonnent notre monde numérique.

RSA

En 1976, personne n'imaginait qu'un article de recherche publié par Whitfield Diffie et Martin Hellman allait bouleverser le monde de la cryptographie. Leur publication « New Directions in Cryptography » introduit une idée qui semblait relever de la pure théorie : la cryptographie asymétrique. Jusqu'à cette date, tous les systèmes cryptographiques fonctionnaient selon un principe unique : deux personnes qui souhaitaient communiquer de façon sécurisée devaient d'abord se rencontrer pour échanger une clé secrète. Cette contrainte paralysait le développement des communications numériques.

Le trio de chercheurs – Diffie, Hellman et Ralph Merkle – propose trois concepts qui vont révolutionner la discipline : le chiffrement à clé publique, la signature numérique et l'échange de clés. Leur article ne contient toutefois qu'une seule implémentation concrète, celle de l'échange de clés qui portera leurs noms. Ce protocole Diffie-Hellman constitue la première méthode pratique pour établir un secret partagé sans rencontre préalable.

L'année suivante, trois chercheurs du MIT vont franchir le pas décisif. Ron Rivest, Adi Shamir et Len Adleman forment une équipe aux talents complémentaires. Rivest excelle dans l'art d'appliquer des concepts théoriques à des problèmes concrets. Il dévore la littérature scientifique et génère un flux constant d'idées nouvelles. Shamir possède cette rare capacité à percer l'essence d'un problème au-delà de sa complexité apparente. Adleman, mathématicien rigoureux, évalue chaque proposition avec la précision d'un horloger suisse.

Pendant des mois, le trio explore diverses pistes sans succès. Puis arrive cette soirée d'avril 1977. Rivest a passé la soirée de Pessah chez un étudiant,

où ils ont partagé du vin Manischewitz. De retour chez lui, incapable de dormir, il s'installe sur son canapé avec un manuel de mathématiques. La question qui l'obsède depuis un an tourne dans sa tête : existe-t-il une fonction mathématique facile à calculer dans un sens, mais impossible à inverser sans une information particulière ? Au lever du soleil, Rivest a rédigé l'intégralité de l'article décrivant le système RSA.

Le génie de cette découverte réside dans sa simplicité conceptuelle. Le système RSA s'appuie sur une propriété arithmétique fondamentale : multiplier deux nombres premiers est un jeu d'enfant, mais retrouver ces nombres à partir de leur produit est un cauchemar quand ils atteignent une taille respectable. Le système génère deux clés mathématiquement liées : une clé publique que chacun peut connaître, et une clé privée que seul son propriétaire détient. Ce qui est chiffré avec la première ne peut être déchiffré qu'avec la seconde.

Pour démontrer leur confiance dans ce nouveau système, les trois inventeurs lancent un défi audacieux dans les pages de *Scientific American*. Ils publient un nombre de 129 chiffres, fruit de la multiplication de deux nombres premiers secrets, accompagné d'un message chiffré. Cent dollars attendent celui qui parviendra à le déchiffrer avant le 1er avril 1982. Ce pari témoigne de leur conviction dans la robustesse mathématique de leur création.

RSA-129 résistera bien au-delà de l'échéance fixée. Il faudra attendre 1994 pour voir ce géant numérique s'effondrer sous l'assaut coordonné de 600 personnes réparties dans 24 pays. Arjen Lenstra, Paul Leyland, Michael Graff et Derek Atkins orchestrent cette entreprise collaborative qui mobilise 1 600 ordinateurs pendant sept mois. Le message secret révélé – « THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE » – récompense cette prouesse technique d'une poésie toute relative.

Cette victoire collective ne signe pas la mort du RSA mais illustre l'évolution permanente du rapport de force entre cryptographes et cryptanalystes. Au fil des décennies, de nombreuses attaques ont vu le jour, ciblant souvent les faiblesses d'implémentation plutôt que les fondements mathématiques du système. Michael Wiener démontre en 1990 qu'un exposant privé trop petit compromet la sécurité. Paul Kocher révèle en 1996 qu'une carte à puce peut trahir sa clé secrète par le simple temps qu'elle met à calculer. Daniel Bleichenbacher découvre en 1998 que certains messages d'erreur divulguent des informations précieuses à un attaquant patient.

Ces découvertes n'ont pas entamé la confiance placée dans le RSA, devenu l'un des piliers de l'internet moderne. Chaque fois que vous vous connectez à votre banque en ligne, que vous effectuez un achat ou que vous consultez votre messagerie, le RSA travaille en silence pour proté-

ger vos données. Sa présence discrète mais omniprésente a rendu possible l'explosion du commerce électronique et la dématérialisation des services financiers.

Pourtant, une épée de Damoclès plane sur ce géant de la cryptographie. En 1994, Peter Shor publie un algorithme qui pourrait, sur un ordinateur quantique suffisamment puissant, factoriser efficacement de très grands nombres. Cette menace théorique stimule la recherche en cryptographie post-quantique, une course contre la montre pour développer de nouveaux systèmes avant que les ordinateurs quantiques ne soient réalité.

Quarante-cinq ans après sa naissance, le RSA conserve sa pertinence et sa fiabilité. Son parcours démontre qu'une innovation est en mesure de transformer la société bien au-delà des intentions initiales de ses créateurs, alors que de nouveaux enjeux cryptographiques émergent avec l'avènement de l'ère quantique.

Apple II

Avec la sortie de l'Apple II en 1977, Steve Wozniak et Steve Jobs ne se contentent plus de bricoler dans un « garage », ils veulent conquérir le grand public. Leur première machine grand public ne ressemble en rien aux kits électroniques qui encombrent alors les clubs d'informatique. Fini les cartes mères nues et les boîtiers de fortune : l'Apple II se présente dans un élégant coffrage en plastique beige, clavier intégré, prêt à l'emploi.

Wozniak a conçu l'architecture autour du microprocesseur 6502, moins cher que l'Intel 8080 qui équipe ses concurrents. Son génie technique transparaît dans la sobriété du circuit imprimé, où chaque composant trouve sa place avec une économie de moyens remarquable. La machine démarre avec 4 Ko de mémoire vive, extensible jusqu'à 48 Ko, et embarque l'Integer BASIC directement en mémoire morte. Mais ce qui frappe d'emblée, c'est la gestion native des couleurs et du son. Wozniak avait d'ailleurs développé ces fonctionnalités dans un but précis : programmer *Breakout* en BASIC.

Jobs, lui, pense marketing et design. Il refuse catégoriquement les boîtiers métalliques qui donnent aux ordinateurs des airs d'équipement industriel. Jerry Manock dessine un coffrage en plastique moulé aux angles arrondis, avec cette forme caractéristique en biseau qui deviendra la signature visuelle d'Apple. L'appareil devra trouver sa place dans un salon comme n'importe quel téléviseur.

La commercialisation démarre en juin 1977 à 1 298 dollars pour la confi-

guration de base. Le succès dépasse les espérances. Particuliers, écoles et petites entreprises se disputent la machine. L'architecture ouverte avec ses huit slots d'extension facilite les améliorations : mémoire supplémentaire, interfaces diverses, contrôleurs de disques. Un écosystème complet de fabricants tiers se développe.

Avec l'arrivée du lecteur de disquettes Disk II en 1978, Wozniak a frappé fort : pour 495 dollars, sa création offre 113 Ko de stockage sur des disquettes 5 pouces 1/4. Le système d'exploitation DOS, dans ses versions successives 3.1, 3.2 puis 3.3, simplifie la gestion des fichiers. Cette solution fiable et abordable ouvre définitivement les portes du marché professionnel à l'Apple II.

L'année 1979 restera dans les mémoires. Apple commercialise l'Apple II Plus qui remplace l'Integer BASIC par l'Applesoft BASIC en ROM, plus sophistiqué avec ses nombres à virgule flottante. La mémoire standard grimpe à 48 Ko. Mais surtout, VisiCalc fait son apparition. Ce premier tableur de l'histoire révolutionne l'usage de l'ordinateur en entreprise. Beaucoup achètent un Apple II uniquement pour faire tourner VisiCalc.

L'Apple IIe de 1983, véritable vedette de la gamme, possède 64 Ko de RAM extensibles à 128 Ko et un affichage 80 colonnes, le tout en faisant un outil de bureautique crédible. Le clavier complet avec minuscules facilite la saisie de texte. La compatibilité avec l'existant est totale, gage de pérennité des investissements logiciels.

Apple tente le pari de la compacité avec l'Apple IIc en 1984. Ce modèle intégré, lecteur de disquette incorporé, vise les foyers et les écoles. L'abandon des slots d'extension au profit de ports standardisés simplifie l'usage mais limite l'évolutivité. Son design soigné lui vaut plusieurs prix esthétiques, une première pour Apple.

Le chant du cygne arrive avec l'Apple IIGS en 1986. Son processeur 16 bits est compatible avec le 6502, ses capacités graphiques et sonores impressionnent, son interface s'inspire du Macintosh. Hélas, Apple mise déjà tout sur le Macintosh. L'IIGS, malgré ses qualités indéniables, reste dans l'ombre de son grand frère.

La production de l'Apple II s'arrête définitivement en 1993 après seize années d'une carrière exceptionnelle. Plus de six millions d'exemplaires vendus témoignent d'un succès qui dépasse le cadre informatique. Cette machine a démocratisé l'ordinateur personnel, prouvé qu'il pouvait être à la fois puissant et accessible.

Commodore P.E.T

En 1977, quand Commodore International dévoile son Personal Electronic Transactor, l'informatique domestique n'existe pratiquement pas. Les ordinateurs demeurent l'apanage des universités, des entreprises et de quelques passionnés capables d'assembler des kits complexes. Jack Tramiel, le patron de Commodore, flaire pourtant une opportunité. Les microprocesseurs voient leurs prix s'effondrer, et il imagine un appareil différent : un ordinateur complet, prêt à l'emploi, dans un seul boîtier.

Le P.E.T tranche avec tout ce qui existe alors. Fini les composants éparpillés sur plusieurs cartes, fini l'écran de télévision bricolé et le magnétophone détourné de sa fonction première. Commodore intègre tout : processeur, mémoire, écran monochrome de 9 pouces, clavier et lecteur de cassettes. Cette approche révèle une vision commerciale astucieuse. Là où d'autres fabricants s'adressent aux bricoleurs électroniques, Tramiel vise directement les futurs utilisateurs ordinaires.

Le cœur de la machine bat au rythme d'un processeur MOS Technology 6502 cadencé à 1 MHz. Ce choix technique, loin d'être anodin, s'appuie sur les qualités reconnues de ce composant : architecture simple, coût réduit et performances honorables. La mémoire RAM de 4 Ko représente un compromis raisonnable entre fonctionnalité et prix de revient. D'ailleurs, Commodore propose des versions étoffées à 8, 16 et 32 Ko.

Microsoft BASIC trône en mémoire morte, transformant instantanément le P.E.T en machine programmable dès sa mise sous tension. Cette version du langage, adaptée aux spécificités de l'ordinateur, intègre des commandes dédiées aux périphériques et aux capacités graphiques rudimentaires de l'écran. L'utilisateur a la possibilité de taper immédiatement ses premières lignes de code ou charger un programme depuis une cassette.

Car le stockage pose problème. Le Datasette de Commodore, dérivé d'un magnétophone classique, se révèle d'une lenteur exaspérante. Charger un programme de quelques kilo-octets nécessite de longues minutes d'attente, ponctuées de bips et de couinements caractéristiques. Pourtant, cette solution demeure la seule économiquement viable pour démocratiser l'accès à l'informatique. Les lecteurs de disquettes coûtent encore trop cher pour équiper massivement les foyers.

Les premiers utilisateurs découvrent les limites du clavier d'origine. Commodore a repris le principe des calculatrices de poche, avec des touches plates et rapprochées qui rendent la frappe pénible. Les protestations fusent, et la firme américaine rectifie le tir en proposant un clavier complet sur les modèles suivants. Cette réactivité face aux critiques témoigne d'une écoute attentive du marché.

L'écran du P.E.T affiche 40 colonnes sur 25 lignes en caractères verts sur fond noir. Pas de couleurs, pas de sprites animés comme sur les futures machines de jeu. Mais cette sobriété cache des possibilités insoupçonnées. Les programmeurs détournent ingénieusement les caractères semi-graphiques pour dessiner des formes, créer des interfaces rudimentaires ou concevoir des jeux d'action. Le P.E.T dispose aussi d'un mode haute résolution optionnel, le Visible Memory de MTU, qui autorise le contrôle individuel de chaque point à l'écran.

Dans les écoles, le P.E.T trouve sa place. Sa robustesse, son fonctionnement fiable et son prix abordable en font l'outil idéal pour initier les élèves à l'informatique. Des générations d'étudiants apprennent leurs premiers rudiments de programmation sur ces machines vertes et blanches. L'interface IEEE-488, héritée du monde de l'instrumentation scientifique, permet de connecter des appareils de mesure professionnels, étendant l'usage du P.E.T aux laboratoires et centres de recherche.

La communauté des utilisateurs s'organise spontanément. Le P.E.T User Group voit le jour en 1978 et fédère des milliers de membres à travers le monde. Publications spécialisées, échanges de programmes sur cassette, rencontres régionales : un écosystème vivant se développe autour de la machine. Cette dynamique collective compense les limitations techniques par une créativité débordante.

Des développeurs indépendants créent des applications remarquables. VisiCalc, le premier tableur grand public, trouve dans le P.E.T une plateforme de choix pour séduire les petites entreprises et les professions libérales. Des jeux sophistiqués, des logiciels de gestion, des outils éducatifs enrichissent progressivement le catalogue disponible. Cette diversité logicielle transforme le P.E.T d'ordinateur de programmation en véritable outil de productivité.

Commodore tire les leçons de cette expérience pionnière. Les succès et les échecs du P.E.T nourrissent directement la conception du VIC-20 puis du légendaire Commodore 64. L'approche tout-en-un validée, l'importance du facteur prix confirmée, la nécessité d'un écosystème logiciel étoffé comprise : autant d'enseignements qui guideront les futures créations de l'entreprise californienne.

Le P.E.T s'éteint progressivement au début des années 1980, supplanté par des machines plus puissantes et moins chères. Cette machine prouve qu'un ordinateur personnel peut séduire au-delà du cercle des initiés. Elle trace les contours d'un marché nouveau où la technique cède le pas à l'usage, où la complexité s'efface derrière la simplicité d'emploi. Le P.E.T fascine les collectionneurs et les historiens de l'informatique. Non pas pour ses performances, dépassées depuis longtemps, mais pour ce qu'il représente :

un des premiers pas vers la démocratisation de l'ordinateur personnel.

Tandy TRS-80

En 1976, dans les bureaux de Tandy Radio Shack à Fort Worth au Texas, Don French nourrit une idée audacieuse. Cet homme, passionné d'informatique et cadre dans cette entreprise qui compte près de 3 500 magasins d'électronique aux États-Unis, veut convaincre son président Lew Kornfield de se lancer dans la vente d'ordinateurs en kit. L'idée paraît folle : vendre un produit à 199$ dans des magasins habitués à écouler des marchandises autour de 30$. Les dirigeants se montrent sceptiques. Mais les temps changent. Les ventes de CB, ces postes radio Citizen Band qui faisaient la fortune de l'enseigne, commencent à s'essouffler. L'entreprise a besoin d'un nouveau souffle, d'un produit capable de maintenir ses marges. Le projet de French obtient finalement le feu vert.

Le printemps 1976 voit French et ses collègues se rendre chez National Semiconductor. Là, ils rencontrent Steve Leininger, concepteur de puces et membre du célèbre Homebrew Computer Club. Sollicité d'abord comme simple consultant, Leininger finit par accepter un poste chez TRS. L'homme va bouleverser le projet initial. Alors que les kits dominent le marché face aux systèmes assemblés beaucoup plus chers, il pousse TRS vers la seconde option. Son pari : proposer un ordinateur entièrement monté pour 399,95$. Avec le moniteur, l'ensemble atteindrait 599,95$. Le prix se maintient suffisamment bas pour séduire les dirigeants.

La prudence est de mise chez Tandy. Don French l'a confirmé : seulement 3 500 unités sont produites initialement, soit exactement le nombre de magasins Radio Shack. La logique semble évidente : si l'aventure tourne mal, les machines serviront dans les points de vente. Le 3 août 1977, l'hôtel Warwick de New York accueille la présentation officielle du TRS-80. La conférence de presse passe presque inaperçue. Deux jours plus tard, la chance sourit à l'entreprise : la présence du TRS-80 au Personal Computer Faire de Boston décroche un article en première page de l'Associated Press.

French découvre à son retour que son bébé a pris des proportions inattendues. Les chiffres parlent d'eux-mêmes : environ 250 000 unités vendues pour ce premier modèle. Le prix attractif face à la concurrence d'Apple ou de Commodore n'explique pas tout. Radio Shack bénéficie d'un atout : sa notoriété et ses quelque 5 000 magasins et franchises en 1977. N'importe qui peut désormais voir et tester un TRS-80 dans un centre commercial

ou une rue commerçante. L'ordinateur sort du cercle des passionnés pour toucher le grand public.

Les entreprises commencent à s'intéresser au TRS-80, mais la première génération montre vite ses limites. Face à des machines comme l'IBM 5110, elle manque de puissance et d'espace de stockage. Tandy riposte en juin 1979 lors de la National Computer Conference à New York avec le TRS-80 Model II. Attention, ce n'est pas un successeur du Model I, qui prend alors ce nom. La publicité de Radio Shack le martèle : il s'agit d'une machine différente, autrement plus chère avec un prix de base à 3 490$ incluant un lecteur de disque. Le processeur Z80 passe de 1,77 à 4 MHz, la mémoire grimpe de 4K extensible à 16K vers 32K extensible à 48K, les graphiques gagnent en résolution. Fini le lecteur de cassettes, place à un TRSDOS spécifique et un BASIC renforcé.

Juillet 1980 marque l'arrivée du TRS-80 Model III, véritable héritier du Model I. Les interférences électromagnétiques des premiers ordinateurs personnels poussent la FCC à durcir ses directives. Le Model I tire sa révérence en 1981. Comme son grand frère Model II, le Model III intègre tout dans un seul boîtier : moniteur, clavier, lecteur de disque. Cette conception réduit les parasites radio et séduit les établissements scolaires, moins exposés au vol qu'avec des éléments séparés. Le processeur Z80 atteint 2,0 MHz et embarque un ROM BASIC 2 plus performant. L'architecture préserve une large compatibilité avec le Model I, mais TRSDOS 1.3 est spécifique au Model III. Un modèle complet coûte 2 495$ en 1980, la version d'entrée de gamme s'affichant à 699$.

Avril 1983 voit naître le TRS-80 Model 4. Échaudés par les problèmes de compatibilité entre Model I et Model III, les développeurs misent sur une compatibilité totale avec le prédécesseur. Le monde informatique salue cette décision, mais déplore l'absence de logiciels natifs. Le Model 4 franchit un cap : horloge à 4 MHz, 64K de RAM avec possibilité d'ajouter 64K supplémentaires, écran 80x24 avec vidéo inverse, TRSDOS 6 avec Microsoft BASIC et compatibilité CP/M. La configuration de base avec lecteur de cassette 16K démarre à 999$, le modèle complet avec 64K et deux lecteurs de disquettes culmine à 1 999$. Deux versions limitées suivront, mais les configurations originales tiendront jusqu'en 1991.

Le triomphe du TRS-80 s'explique de différentes façons. D'abord, cette disponibilité immédiate dans le vaste réseau Radio Shack. Ensuite, une approche pédagogique remarquable avec le manuel du Level I BASIC, d'une clarté exemplaire. La conception modulaire séduit aussi : chacun peut faire évoluer sa machine selon ses moyens et ses besoins. Radio Shack assure le service après-vente, gage de confiance. L'entreprise développe tout un écosystème : périphériques variés, logiciels du jeu aux applications profession-

nelles (comptabilité, gestion des stocks). Plus de 100 000 systèmes trouvent preneur, hissant le TRS-80 parmi les premiers ordinateurs personnels largement diffusés.

L'impact sur l'informatisation des petites entreprises, des écoles et des foyers américains se révèle considérable. Le TRS-80 démocratise l'accès aux ordinateurs bien au-delà des cercles de spécialistes et de professionnels. Il incarne cette mutation du marché de la micro-informatique entre la fin des années 1970 et le début des années 1980, ce passage d'un marché de niche vers le grand public et le monde professionnel.

Berkeley Software Distribution

Quand Ken Thompson, Dennis Ritchie et Rudd Canaday créent UNICS à l'université Purdue en 1969, ils ignorent qu'ils viennent de lancer une lignée qui donnera naissance à l'une des familles de systèmes d'exploitation les plus influentes de l'histoire informatique. UNICS est renommé UNIX deux ans plus tard, et dès 1973-1974, le système connaît ses premières ramifications avec PWB/UNIX et MERT. Tandis que PWB/UNIX évoluera vers System III, MERT disparaît des écrans radar.

La véritable aventure BSD débute le 9 mars 1978 dans les locaux du Computer Systems Research Group de l'université de Berkeley. Cette première mouture de BSD est modeste, mais les versions 2 et 3 marquent un tournant. Les machines DEC VAX 32 bits remplacent peu à peu les vieillissantes PDP-11, et BSD s'adapte à cette transition technologique. Les développeurs intègrent le compilateur Pascal retravaillé par Thompson, l'éditeur *vi* qui deviendra mythique, et le shell C qui change la donne pour les administrateurs système.

L'histoire aurait pu s'arrêter là sans l'intervention providentielle de la DARPA. Bob Fabry décroche un financement de 18 mois qui transforme BSD 3 en laboratoire d'innovations. La mémoire virtuelle fait son apparition, accompagnée de la pagination à la demande et d'algorithmes sophistiqués de remplacement de pages. Ces améliorations préparent le terrain pour BSD 4, qui introduit le protocole ARPANET — ancêtre de TCP/IP — et une gestion mémoire repensée. La base de données `termcap` fait évoluer la gestion des terminaux, tandis que le support des périphériques gagne en robustesse.

Jim Kulp apporte sa contribution à BSD 4.1 avec le contrôle des tâches, mais c'est surtout la détection automatique du matériel qui change la vie

des utilisateurs. Fini les recompilations fastidieuses du noyau à chaque changement de configuration : le système reconnaît et configure seul les périphériques au démarrage. Un noyau est désormais capable de tourner sur des machines différentes sans intervention humaine.

L'année 1983 marque l'apothéose avec BSD 4.2. Bill Joy et Kirk McKusick signent leur chef-d'œuvre avec le Berkeley Fast File System, qui surclasse largement l'UFS traditionnel en termes de performances et d'utilisation de l'espace disque. Les signaux révolutionnent la communication entre processus, l'IPC structure les échanges complexes, et TCP/IP s'impose comme la référence réseau. Ces innovations inspirent une génération de systèmes : SunOS 1.2, Ultrix-II, Mach et MIPS OS puisent directement dans cette mine d'or technologique.

BSD 4.3 sort en 1986 avec pour mission de consolider les acquis et peaufiner TCP/IP. Le système atteint une maturité technique remarquable, mais l'aspect commercial entre en scène. En 1990, BSDi lance BSD/386 en s'appuyant sur le code de BSD Net/2. Cette version commerciale se positionne comme une alternative crédible à System V d'AT&T, avec un argument de poids : un prix plus abordable. Cependant, l'épée de Damoclès de la licence AT&T pend toujours au-dessus de BSD.

Le coup de tonnerre arrive en 1992 quand AT&T traîne BSDi devant les tribunaux pour violation de marque et publicité trompeuse. Après un premier échec, AT&T revient à la charge en 1993, cette fois contre BSDi et l'université de Californie. L'accusation : distribution de code propriétaire sans autorisation. La situation se complique quand AT&T vend USL et les droits UNIX à Novell. Il faudra attendre le 4 février 1994 pour voir naître 4.4BSD Lite, épuré de tous les éléments sous licence AT&T.

Pendant ce temps, Bill Jolitz travaille dans l'ombre sur 386BSD. Son approche diffère : plutôt que de négocier avec les juristes, il développe des alternatives aux composants problématiques. Cette démarche débouche sur une version de BSD entièrement libre, débarrassée des contraintes légales qui entravaient son développement. Les distributions ralentissent temporairement, mais cette libération technique constitue une victoire stratégique pour le logiciel libre.

Les années suivantes voient éclore plusieurs branches : NetBSD naît du code BSD Net/2, FreeBSD prolonge l'héritage de 386BSD, DragonFlyBSD se sépare plus tard de FreeBSD, et Apple intègre discrètement des éléments FreeBSD dans le noyau hybride Mach/BSD de macOS X.

L'héritage technique de BSD traverse les décennies. Le système de fichiers FFS inspire encore les développeurs contemporains. La pile TCP/IP de Berkeley est une référence pour comprendre les protocoles réseau. Le contrôle des tâches et la détection automatique du matériel font mainte-

nant partie du paysage UNIX standard. Mais au-delà de ces contributions techniques, BSD inaugure un modèle de développement inédit : la collaboration entre université et industrie, l'amélioration collective du code, le partage des innovations.

Sa licence permissive tranche avec les approches plus restrictives. Cette ouverture favorise l'adoption du code BSD dans des projets commerciaux, créant un écosystème où recherche académique et développement industriel se nourrissent mutuellement. macOS illustre parfaitement cette symbiose : un système commercial qui intègre et valorise des composants libres.

La fragmentation de BSD en multiples variantes — FreeBSD, NetBSD, OpenBSD — reflète paradoxalement sa vitalité. Chaque branche explore un territoire spécifique : NetBSD mise sur la portabilité universelle, OpenBSD privilégie la sécurité absolue, DragonFlyBSD repense les architectures multiprocesseurs. Cette diversité enrichit l'écosystème plutôt qu'elle ne l'affaiblit.

Dans la grande histoire de l'informatique, BSD incarne la transmission du savoir technique à travers les générations de développeurs. Le système voyage de l'université vers l'industrie, de la recherche vers les applications critiques, sans perdre son âme ni ses qualités intrinsèques.

Space Invaders

En 1978, dans les bureaux de Taito Corporation à Tokyo, Tomohiro Nishikado travaille sur un projet qui va bientôt bouleverser l'industrie naissante du jeu vidéo. L'ingénieur japonais ignore qu'il s'apprête à créer l'un des phénomènes culturels les plus durables de son temps. Son *Space Invaders* voit le jour dans un contexte où les constructeurs nippons tentent de rattraper leur retard sur les Américains, qui dominent le secteur depuis le succès retentissant de *PONG* six ans plus tôt.

Là où *PONG* s'appuyait sur des circuits TTL rudimentaires, *Space Invaders* introduit une architecture dont le microprocesseur est couplé à une mémoire vidéo bitmap. Cette combinaison autorise l'affichage simultané de dizaines d'objets graphiques animés, une performance inégalée pour l'époque. L'astuce visuelle du jeu réside dans sa projection monochrome sur un fond coloré par filtres, solution économique mais diablement efficace.

Le génie de Nishikado ne s'arrête pas aux aspects purement techniques. Il forge les codes du jeu vidéo moderne : système de score, vies multiples, montée en difficulté, boucle infinie. Le concept paraît d'une simplicité désar-

mante : un canon laser face à des hordes d'aliens qui descendent inexorablement. Plus le joueur excelle, plus les envahisseurs accélèrent leur descente mortelle. Cette mécanique crée une tension qui ne se relâche jamais.

L'accueil du public japonais confine au délire collectif. Les salles d'arcade se transforment en temples dédiés au nouveau phénomène. Certains établissements abandonnent tous leurs autres jeux pour installer exclusivement des bornes *Space Invaders*. La légende raconte qu'une pénurie de pièces de 100 yens aurait frappé l'archipel – histoire inventée mais symptomatique de l'engouement national. Les chiffres parlent d'eux-mêmes : des centaines de millions de dollars de recettes en quelques mois.

Cette réussite foudroyante redessine la géographie industrielle du secteur. Les constructeurs japonais, galvanisés par ce succès inattendu, déversent des capitaux dans la recherche et le développement. Namco, Sega et Nintendo émergent de cette effervescence créatrice qui va dominer la décennie suivante. L'industrie du jeu d'arcade trouve sa maturité économique.

Les contraintes techniques du système original – incapable de remplir plus d'un quart de l'écran en temps réel – stimulent paradoxalement l'innovation. Namco répond dès 1979 avec *Galaxian* et sa technologie sprite, qui gère plus finement les objets mobiles. Cette course à l'armement technologique ne cessera plus : processeurs de signal numérique pour la 3D dans les années 1990, puis processeurs graphiques spécialisés au tournant du millénaire.

Au-delà du divertissement, *Space Invaders* cristallise les tensions géopolitiques de son temps. L'Amérique des années 1970 voit d'un œil inquiet l'ascension économique du Japon. Les envahisseurs pixellisés font écho aux craintes d'une « invasion » des produits nippons sur le marché américain. Ironie du sort, Atari acquiert les droits du jeu pour les États-Unis, illustrant la complexité des échanges entre les deux puissances.

Cette mutation culturelle s'accompagne d'une professionnalisation du secteur. Fini le temps des bricoleurs solitaires : développer un jeu nécessite désormais une équipe structurée. Programmeurs, ingénieurs hardware, concepteurs, graphistes et producteurs forment les premiers studios dignes de ce nom. Cette organisation marque le début l'industrie tentaculaire que nous connaissons.

L'effet social du jeu surprend ses créateurs. Les spectateurs se rassemblement autour des bornes d'arcade pour admirer les performances des virtuoses. Ces attroupements spontanés portent en germe la culture du spectacle vidéoludique qui explosera avec l'e-sport contemporain. *Space Invaders* révèle la dimension collective d'un médium que l'on croyait solitaire.

Son influence perdure dans les créations actuelles, notamment le principe défensif de protéger son territoire contre des vagues d'assaillants. Score,

progression, tension crescendo : autant d'éléments devenus indissociables de l'expérience vidéoludique. Le jeu de Nishikado démontre comment une intuition créative, soutenue par l'innovation technique, peut transformer un secteur entier.

Space Invaders, ce petit programme de quelques kilo-octets, a redéfini notre rapport au divertissement numérique. Il marque le moment où le jeu vidéo s'émancipe de ses origines artisanales pour s'imposer comme forme d'expression autonome, capable de générer ses propres codes esthétiques et sociaux.

TEX

Donald Knuth reçoit un jour les épreuves du second volume de *The Art of Computer Programming*. Nous sommes en 1977, et le résultat le déçoit au point de l'amener à tout arrêter. La composition typographique de ses formules mathématiques lui paraît si médiocre qu'il décide de créer son propre système plutôt que de continuer l'écriture de ses livres. L'industrie de l'imprimerie traverse un bouleversement technique : elle abandonne la composition au plomb, où chaque caractère était coulé dans le métal puis assemblé à la main, pour adopter la photocomposition. Cette nouvelle méthode coûte moins cher et va plus vite, certes, mais Knuth n'accepte pas la baisse de qualité qui l'accompagne.

L'informaticien de Stanford consacre son année sabbatique 1977-1978 à développer TEX. Il ne se contente pas d'écrire un simple programme de mise en page : il conçoit un système complet avec son langage de description de documents et son format de sortie DVI, indépendant de l'imprimante utilisée. Ses premiers essais ont lieu sur un ordinateur DEC PDP-10, en langage SAIL. Frank Liang et Michael Plass, ses étudiants, testent les prototypes successifs et participent activement au développement.

TEX redéfinit l'art de la composition par ses algorithmes sophistiqués. Frank Liang met au point un système de césure qui s'adapte aux différentes langues, tandis que Michael Plass aide Knuth à concevoir la justification des paragraphes. L'innovation technique réside dans le concept de « boîtes et ressorts » : TEX gère les espaces entre les éléments comme des ressorts virtuels qui se compriment ou s'étirent selon les besoins. Cette approche produit une justification d'une qualité inégalée.

Knuth accompagne TEX d'un second système : METAFONT. Plutôt que de dessiner le contour des lettres comme le feront plus tard PostScript

et TrueType, METAFONT programme les caractères en simulant les mouvements d'une plume. Cette méthode paramétrique génère des familles entières de polices en modifiant quelques variables. Les polices Computer Modern, créées avec cet outil, marquent visuellement tous les documents TEX depuis quarante ans.

En 1982, la réécriture en Pascal donne naissance à TEX82. Ce changement de langage améliore la portabilité et sert de base à toutes les versions ultérieures. Knuth adopte alors une gestion des versions originale : les numéros tendent vers π pour TEX et vers e pour METAFONT. Cette approche mathématique reflète sa volonté de stabiliser définitivement le système. Il s'engage à ne corriger que les bogues graves, garantissant qu'un document produira toujours un résultat identique, même des décennies plus tard.

L'American Mathematical Society fait confiance à TEX pour ses publications. Cette adoption par une institution de référence contribue à sa diffusion dans les universités du monde entier. Leslie Lamport comprend que le système de Knuth est trop complexe pour beaucoup d'utilisateurs. En 1984, il crée LATEX, une couche de commandes simplifiées qui cache la complexité technique derrière une interface logique. L'auteur se concentrer désormais sur la structure de son document plutôt que sur sa mise en forme, et le succès dépasse celui du TEX « plain ».

Knuth réinvente aussi la documentation logicielle. Il développe spécialement pour TEX les principes de la programmation littéraire, mêlant code et explications dans un seul document. Cette approche produit une documentation exceptionnellement complète qui aide d'autres programmeurs à comprendre et modifier le système. Une communauté d'utilisateurs naît autour de TEX. Le TEX Users Group voit le jour en 1980 et publie TUGboat, une revue technique qui accompagne l'évolution du système.

TEX reste l'outil de référence des scientifiques. Les mathématiciens, physiciens et informaticiens l'utilisent massivement pour rédiger leurs articles. Des extensions comme pdfTEX, XeTEX et LuaTEX ont ajouté de nouvelles fonctions sans altérer la qualité typographique originale. Paradoxalement, les innovations de Knuth n'ont jamais été reprises par les logiciels grand public. Microsoft Word ou LibreOffice Writer proposent une composition bien inférieure malgré la puissance des ordinateurs actuels.

L'histoire de TEX montre à quel point l'exigence de qualité d'un seul homme permet de changer une discipline entière. Elle démontre l'importance de l'ouverture du code source et d'une documentation exemplaire pour la survie d'un logiciel. Dans un monde où les technologies se succèdent à un rythme effréné, TEX traverse les décennies en gardant toute sa pertinence. Knuth avait prévu de passer quelques mois sur ce projet personnel. Près d'un demi-siècle plus tard, son œuvre continue de composer

les plus beaux textes mathématiques ou littéraires de la planète.

WordStar

En mai 1978, Seymour Rubinstein claque la porte d'IMSAI Manufacturing Corporation. Un désaccord avec Bill Millard vient de briser sa carrière de directeur marketing. Deux semaines plus tôt, Rob Barnaby, programmeur de talent, avait déjà quitté le navire. Rubinstein le retrouve et lui lance une proposition simple : créer ensemble des logiciels pour ces micro-ordinateurs qui commencent à sortir des garages californiens.

Juin 1978 voit naître MicroPro International Corporation. Le nom traduit l'ambition de Rubinstein : développer des programmes « PROfessionnels » pour les « MICRO-ordinateurs ». Les premiers fruits arrivent dès septembre avec WordMaster et SuperSort. WordMaster reprend les idées d'un éditeur de texte que Barnaby avait développé chez IMSAI, entièrement réécrit pour tourner sur CP/M. SuperSort s'inspire des utilitaires de tri d'IBM 360. Ces deux programmes rapportent 10 700 dollars dès septembre 1978, de quoi voir l'avenir avec confiance.

Mais les revendeurs réclament autre chose : un vrai traitement de texte qui gère directement l'impression. Début 1979, Barnaby se lance dans un tour de force technique. En six mois, il écrit plus de 137 000 lignes de code assembleur. IBM aurait estimé ce travail à 42 années-homme. La première version de WordStar sort en juin 1979.

Le logiciel bouscule les habitudes. Pour la première fois, l'écran montre ce qui sortira sur papier. WordStar intègre une aide qui s'adapte aux gestes de l'utilisateur, guide ses pas sans l'encombrer. Les dactylographes retrouvent leurs marques : toutes les commandes restent accessibles au clavier, et les sauts de page s'affichent à l'écran, un luxe inouï.

Le succès dépasse toutes les espérances. MicroPro grimpe de 500 000 dollars en 1979 à 1,8 million en 1980, puis 5,2 millions en 1981. L'arrivée du capital-risqueur Fred Adler en 1981 transforme l'entreprise. Une vraie force de vente se met en place. Le chiffre d'affaires bondit à près de 25 millions de dollars, avec une perte d'un million sur l'exercice. En 1982, les revenus atteignent 45 millions.

Rubinstein voit grand et vise l'international. Il installe un siège européen à Zoug, en Suisse, histoire de profiter d'une fiscalité clémente. Des bureaux s'ouvrent en Allemagne, au Royaume-Uni, en France, au Japon. WordStar se décline en 42 langues. Le monde entier découvre ce traitement de texte

futuriste.

WordStar hérite de la philosophie des éditeurs de texte plutôt que de celle des machines à écrire. Le texte est désormais un flux continu, non plus une succession de pages. Les commandes de formatage s'insèrent directement dans le document via des caractères spéciaux, méthode venue tout droit de la programmation. Cette approche séduit universitaires et écrivains qui manipulent de longs manuscrits.

En janvier 1984, Rubinstein s'effondre, victime d'une crise cardiaque, alors que l'entreprise prépare son introduction en bourse. Les avocats de la société le persuadent de signer des documents qui le dépossèdent du contrôle. H. Glen Haney prend les rênes et écarte progressivement le fondateur.

La nouvelle direction accumule les maladresses. WordStar 2000 sort en totale incompatibilité avec la version originale, semant la confusion chez les utilisateurs fidèles. La publicité pour WordStar disparaît pendant trois ans. Le réseau de 1 500 revendeurs se voit sacrifié au profit de quelques gros distributeurs. Les ventes dégringolent de 72 à 40 millions de dollars.

WordPerfect débarque en 1980 en combinant édition et formatage dans un affichage unique. Microsoft Word arrive en 1983 avec l'approche inédite que chaque caractère est une entité indépendante dotée de ses propres attributs. Cette méthode, d'abord plus lente, prendra tout son sens avec l'émergence des interfaces graphiques.

Les années 1990 sonnent le glas de WordStar. Le passage à Windows se fait mal. Le logiciel passe de main en main, racheté par des sociétés spécialisées dans la distribution de programmes à bas prix, et enfin par The Learning Company. Corel obtient une licence mais préfère WordPerfect qu'il vient d'acquérir. WordStar survit grâce à quelques groupes d'utilisateurs nostalgiques.

Pourtant, l'héritage demeure immense. WordStar a prouvé qu'un micro-ordinateur pouvait rivaliser avec les machines à écrire professionnelles. Ses innovations d'interface ont inspiré toute une génération de logiciels. Il a contribué à structurer l'industrie du logiciel en créant les contrats de licence utilisateur et en développant la distribution internationale.

dBASE

Au Jet Propulsion Laboratory de la NASA, dans les années 1970, Wayne Ratliff ne se doutait probablement pas qu'il allait écrire une page de l'his-

toire des bases de données personnelles. Son système JPLDIS, conçu pour traiter les informations du laboratoire spatial, allait devenir l'ancêtre de l'un des logiciels les plus marquants de l'informatique personnelle.

Ratliff transforme son travail initial en un produit qu'il baptise Vulcan et qu'il vend par petites annonces dans la presse spécialisée. L'aventure commerciale prend une autre dimension quand Ashton-Tate décide de distribuer le logiciel sous un nouveau nom : dBASE II. Cette dénomination, qui évite astucieusement la version I, suggère d'emblée la maturité d'un produit déjà éprouvé. Le stratagème fonctionne parfaitement. Sur les micro-ordinateurs 8 bits tournant sous CP/M, dBASE II s'impose comme la référence incontournable pour gérer les données.

Ce qui distingue dBASE des autres outils, c'est son approche globale. Là où ses concurrents proposent des fonctions isolées, dBASE offre un véritable environnement de travail intégré. Une fois lancé, l'utilisateur accède à un univers complet : création de tables, manipulation de données, recherche d'informations, exportation vers d'autres formats. Le logiciel va plus loin en proposant un langage de programmation pour automatiser les tâches répétitives ou de construire des interfaces sur mesure.

Cette intégration pousse le concept jusqu'à inclure un éditeur de texte, des outils de gestion de fichiers et la possibilité d'interagir directement avec le système d'exploitation. Tout se passe dans un seul environnement, sans avoir à jongler entre différents programmes. L'interface en ligne de commande, reconnaissable à son fameux point d'invite, est de plus en plus familière aux utilisateurs qui apprennent à dialoguer directement avec leurs données.

L'architecture technique repose sur une organisation logique des informations. Les tables de données, stockées dans des fichiers .dbf, structurent les informations en colonnes et lignes selon le modèle relationnel classique. Le système s'appuie sur différents types de fichiers : index pour accélérer les recherches, mémoire pour les calculs temporaires, formats de rapport pour la présentation, formats d'écran pour l'interaction, programmes pour l'automatisation.

dBASE III marque une rupture technique importante. La maintenance et l'évolution du code sont facilitées à la suite de l'abandon de l'assembleur au profit du langage C par Ashton-Tate. Cette version introduit la gestion des dates et augmente sensiblement les capacités de traitement. dBASE III Plus, commercialisé en 1986, franchit un nouveau palier avec des menus déroulants inspirés de Framework, des performances doublées et le support natif du multi-utilisateurs.

Les chiffres parlent d'eux-mêmes : un nombre illimité de fichiers, jusqu'à un milliard d'enregistrements par fichier, 128 champs maximum par enregis-

trement, 4 Ko par enregistrement. Les tests montrent que le tri s'effectue deux fois plus vite qu'avec la version précédente, tandis que l'indexation gagne un facteur dix. La compatibilité s'étend à PFS :FILE, Lotus 1-2-3, VisiCalc DIF, MultiPlan et aux fichiers ASCII standard.

L'aspect réseau prend une importance particulière. dBASE III Plus intègre directement les fonctionnalités multi-utilisateurs sans surcoût. Le verrouillage automatique des enregistrements et fichiers évite la corruption des données lors d'accès simultanés. Huit niveaux de protection par mot de passe sécurisent l'accès aux informations sensibles. Le chiffrement et déchiffrement des données s'effectuent automatiquement au niveau de chaque poste de travail.

Pourtant, vers la fin des années 1980, la machine commence à gripper. Le départ de Wayne Ratliff d'Ashton-Tate en 1988 prive l'entreprise de son créateur originel, alors que la concurrence s'intensifie avec l'arrivée de produits comme FoxBASE et Clipper, souvent plus performants et moins chers. Les problèmes judiciaires s'accumulent : Ashton-Tate perd plusieurs procès pour violation de droits d'auteur, n'ayant jamais divulgué que dBASE descendait directement de JPLDIS.

Contrairement à WordStar qui avait su rappeler ses fondateurs lors de difficultés similaires, Ashton-Tate persiste dans une stratégie qui s'avère perdante. Le marché se réorganise autour de nouveaux acteurs : Borland absorbe Ashton-Tate, Microsoft acquiert Fox (l'éditeur de FoxBASE), Computer Associates rachète Nantucket (créateur de Clipper).

Après son départ, Wayne Ratliff développe Emerald Bay, un nouveau produit qui présente des ressemblances troublantes avec Microsoft Access. Dans une interview de 2007 accordée à Visual Pro Magazine, Ratliff souligne ces similitudes sans porter d'accusations explicites, laissant planer un doute sur d'éventuels « emprunts » de concepts.

Le concept d'environnement de développement intégré de dBase, l'interaction directe avec les données via une interface utilisateur intuitive, la programmation orientée données : autant d'innovations qui marquent durablement l'industrie. Microsoft Access, entre autres, hérite directement de ces concepts pionniers.

Jerry Pournelle capture parfaitement l'ambiguïté du produit dans les colonnes du magazine BYTE en le qualifiant d'« exaspérant mais excellent ». Cette formule résume l'état d'esprit où l'industrie du logiciel personnel hésite encore entre innovation technique pure et simplicité d'usage. dBASE incarne cette tension : innovant dans ses concepts, parfois déconcertant dans son utilisation quotidienne. dBASE montre surtout comment les systèmes de gestion de bases de données sont devenus des outils indispensables, tant dans l'informatique personnelle que professionnelle.

Ada

Imaginez un instant le cauchemar administratif que représentaient les projets informatiques du Pentagone au milieu des années 1970. Plus de 400 langages et dialectes de programmation pullulaient dans les systèmes du département de la Défense américain, générant chaque année trois milliards de dollars de coûts de maintenance. Cette situation ubuesque trouvait souvent son origine dans l'initiative malheureuse d'un programmeur qui, persuadé d'améliorer la productivité, bricolait un langage existant pour son application particulière. Vingt années plus tard, des générations entières de développeurs devaient encore apprendre ce dialecte pour maintenir ce programme devenu *legacy*.

David Fisher dirigeait alors l'initiative lancée par le DoD pour sortir de cette impasse. L'idée n'était pas neuve : dans les années 1960, le département avait déjà imposé COBOL dans ses contrats de défense. Mais cette fois, l'approche serait différente. Pour la première fois dans l'histoire des langages de programmation, les exigences seraient définies par une équipe complètement distincte de celle qui concevrait le langage.

Le processus de consultation mobilisa des experts militaires, industriels et universitaires du monde entier. Entre 1975 et 1977, plusieurs documents d'exigences virent le jour avec des noms de code évocateurs : Strawman, Woodenman, Tinman et finalement Ironman. Ces spécifications plaçaient la fiabilité, la lisibilité et la maintenabilité au cœur des préoccupations, bien au-delà des objectifs classiques de portabilité et d'efficacité.

Vingt équipes répondirent à l'appel d'offres lancé en 1977. Quatre furent sélectionnées et baptisées par des couleurs : Vert, Rouge, Bleu et Jaune. Après six mois de travail acharné, seules les équipes Verte et Rouge restaient en lice. En mai 1979, la proposition de l'équipe Verte l'emportait. Son leader, Jean Ichbiah de CII Honeywell Bull, avait un atout dans sa manche : son expérience avec LIS, le « Langage d'Implémentation de Systèmes » développé depuis 1972 dans son entreprise.

LIS avait été conçu pour améliorer la fiabilité des systèmes d'exploitation et leur maintenabilité. Cette philosophie collait parfaitement aux exigences du DoD. Ichbiah et son équipe puisèrent dans cette expérience pour façonner leur proposition.

La phase suivante, baptisée « Test and Evaluation », transforma le projet en véritable laboratoire mondial. Une centaine d'équipes réparties sur tous les continents testèrent le langage en recodant des applications existantes. Leurs retours alimentèrent les raffinements successifs qui aboutirent, en

1980, à une proposition de standard.

Le processus de standardisation prit une dimension pharaonique. Plus de mille personnes à travers le monde participèrent à cette entreprise. L'équipe d'Ichbiah dut traiter environ 7 000 commentaires sur le standard proposé, s'appuyant sur une base de données informatisée pour gérer ce déluge de retours. Une prouesse technique.

En février 1983, la standardisation ANSI d'Ada était finalisée. Le langage se distinguait par des innovations remarquables. Sa structure en paquetages permettait une séparation nette entre l'interface utilisateur et l'implémentation. Le concept de lecture linéaire métamorphosait la compréhension du code : un programmeur pouvait lire un programme Ada ligne par ligne, sa compréhension à une ligne donnée ne dépendant que des lignes précédentes.

Robert Dewar développa le premier compilateur Ada à l'Université de New York, en fait un interpréteur destiné à l'enseignement. Le deuxième compilateur, créé par Rolm et Data General pour le mini-ordinateur Eclipse, fut validé en juin 1983. Western Digital proposa le troisième avec leur MicroEngine, marquant l'entrée d'Ada dans l'univers des micro-ordinateurs.

En janvier 1984, le DoD frappait un grand coup : une directive imposait l'utilisation d'Ada pour toutes les applications critiques. Cette décision intervenait après la validation du compilateur Data General, qui prouvait la viabilité pratique du langage. Mais Ada ne resta pas prisonnier des applications militaires. Le secteur civil l'adopta, notamment dans l'aviation civile, le ferroviaire et les systèmes embarqués temps réel où la fiabilité prime sur tout.

Le langage évolua au fil des décennies. Ada 95 introduisit la programmation orientée objet. Ada 2005 et Ada 2012 apportèrent des améliorations substantielles dans la programmation par contrats. Aujourd'hui, Ada équipe des applications critiques où la moindre défaillance a des conséquences potentiellement dramatiques.

La conception d'Ada a introduit des pratiques qui font désormais école : séparation rigoureuse entre spécification et implémentation, validation formelle des compilateurs, processus de standardisation méticuleux. Ces innovations ont influencé la conception d'autres langages et transformé les méthodes de développement logiciel.

Jean Ichbiah avait une vision claire : « Le développement d'un grand programme peut prendre moins de deux ans, mais sa maintenance s'étendra sur plus de vingt ans. » Cette philosophie imprègne chaque aspect du langage. Ada privilégie la clarté et la compréhension du code plutôt que la facilité d'écriture. Une approche a priori contre-intuitive mais qui s'avère payante sur le long terme.

Le nom du langage rend hommage à Augusta Ada Lovelace, considérée comme la première programmeuse de l'histoire. Ce choix n'était pas anodin : il traduisait l'ambition du projet de marquer l'histoire de l'informatique. Quarante ans plus tard, Ada est considéré comme un modèle de conception rigoureuse et méthodique. Il démontre qu'investir dans la qualité du code génère des économies substantielles sur l'ensemble du cycle de vie d'un logiciel. Une leçon que bien des projets informatiques auraient intérêt à méditer.

Usenet

L'automne 1979 à Duke University ressemblait à n'importe quel autre : les feuilles jaunissaient, les étudiants reprenaient leurs cours, et dans le département informatique, Tom Truscott et Jim Ellis se creusaient la tête sur un problème apparemment banal. Comment faire communiquer efficacement les différents ordinateurs du campus qui tournaient sous UNIX ? Une question technique qui allait bouleverser la manière dont les gens échangent des informations.

L'informatique vivait encore sous la domination des gros systèmes centralisés. Les PDP-11 de DEC commençaient tout juste à coloniser les départements universitaires, tandis que les micro-ordinateurs personnels relevaient de la curiosité technologique. Pour communiquer à distance, on utilisait des modems poussifs à 300 bauds qui crachotaient sur les lignes téléphoniques. ARPANET existait, certes, mais son accès restait jalousement gardé par les militaires et quelques laboratoires privilégiés.

Truscott et Ellis trouvèrent leur solution en s'associant avec Steve Bellovin, étudiant à l'université voisine de Caroline du Nord. Ce dernier développa la première version d'Usenet en seulement 150 lignes de script shell, s'appuyant sur UUCP, un protocole UNIX qui venait d'apparaître et permettait de copier des fichiers entre machines distantes. L'idée était simple : créer des forums thématiques appelés « newsgroups » où chacun pourrait poster des messages, ces derniers se propageant automatiquement d'un site à l'autre.

L'architecture choisie reflétait les contraintes de l'époque autant que l'ingéniosité de ses créateurs. Plutôt que de centraliser les données, chaque site stockait localement tous les messages et les transmettait à ses voisins. Duke servait de hub central pour limiter les frais téléphoniques, une préoccupation bien réelle quand chaque appel longue distance se comptait en dollars. Le protocole A News adoptait un format rudimentaire mais efficace :

chaque message débutait par la lettre A suivie d'un identifiant unique, et les en-têtes indiquaient le chemin parcouru pour éviter que les messages tournent en rond.

En janvier 1980, lors d'une conférence Usenix à Boulder, l'équipe présenta publiquement son système. Ils le qualifièrent de « poor man's ARPANET », une alternative économique au réseau des élites. Le code source fut distribué gratuitement sur bande magnétique, dans l'esprit de partage qui caractérisait la communauté UNIX. Cette générosité n'était pas que philosophique : sans moyens financiers, les créateurs misaient sur l'effet de réseau pour faire prospérer leur invention.

Les débuts furent pourtant modestes. En avril 1981, quinze sites seulement composaient le réseau, gravitant autour de Duke, Berkeley et San Diego. Le décollage vint de Mary Ann Horton, qui eut l'idée de créer une passerelle entre Usenet et les listes de diffusion populaires d'ARPANET. Soudain, les discussions de SF-LOVERS et autres groupes renommés devinrent accessibles aux utilisateurs d'Usenet. Horton développa aussi les outils uuencode et uudecode, qui résolvaient l'épineux problème du transfert de fichiers binaires sur un réseau conçu pour le texte.

La version B News corrigea les défauts de jeunesse du système original. Fini les messages qu'il fallait lire dans l'ordre d'arrivée : on pouvait désormais naviguer librement dans les discussions, suivre des fils de conversation, organiser sa lecture. Ces améliorations tombaient à pic car le réseau explosait littéralement. De 150 sites en 1981, Usenet bondit à 11 000 sites en 1988, brassant 1 800 articles quotidiens.

Cette croissance reflétait l'évolution plus large de l'informatique. UNIX se répandait dans les universités et les entreprises, les modems devenaient plus rapides, les lignes téléphoniques plus fiables. Surtout, Usenet répondait au besoin de l'entraide technique et du partage de connaissances. Les groupes proliféraient, couvrant tous les sujets imaginables, des plus pointus aux plus futiles.

Mais cette expansion révélait les failles d'une architecture pensée pour quelques dizaines de sites. Les nœuds centraux peinaient à relayer le flot croissant de messages. La gouvernance posait question : si la création d'un groupe officiel nécessitait un vote communautaire, la hiérarchie « alt. » laissait chacun libre de créer ses propres forums. Cette anarchie organisée favorisait l'innovation mais compliquait le contrôle des dérives.

Les années 1990 marquèrent paradoxalement l'apogée et le début du déclin d'Usenet. Le système servit de tribune à des annonces qui allaient changer le monde : Linus Torvalds y présenta Linux, Tim Berners-Lee y décrivit le World Wide Web. Ironie de l'histoire, c'est justement le Web qui allait supplanter Usenet en proposant une interface plus accessible au

grand public. L'arrivée d'AOL, puis des réseaux sociaux, détourna progressivement les utilisateurs vers des solutions plus conviviales.

Usenet ne disparut pas pour autant. Le protocole NNTP remplaça UUCP pour s'adapter à Internet, et le volume de données continua de croître, principalement pour le partage de fichiers binaires. Les groupes techniques gardent leur utilité pour les développeurs qui apprécient la qualité des discussions et l'absence de publicité.

Usenet a prouvé qu'on pouvait créer des réseaux sociaux décentralisés et participatifs bien avant que ces concepts ne soient à la mode. Il a enrichi le vocabulaire d'Internet avec des mots comme « spam » ou « troll », introduit l'idée du fil de discussion, expérimenté la modération collaborative. Ses choix techniques ont durablement influencé notre façon de communiquer en ligne.

Les révolutions technologiques naissent parfois de besoins très concrets, et de solutions bricolées par quelques étudiants dans un laboratoire universitaire. Entre l'ouverture totale et le contrôle nécessaire, entre la liberté d'expression et la modération des abus, Usenet a incarné les tensions qui traversent encore nos communautés numériques.

VisiCalc

En 1978, Dan Bricklin suit ses cours à la Harvard Business School quand une situation banale lui donne une idée qui va transformer l'informatique personnelle. Dans le cadre d'une étude de cas sur Pepsi-Cola, il regarde son professeur remplir un tableau financier au tableau noir. À chaque modification d'une donnée, l'enseignant doit recalculer péniblement l'ensemble de ses chiffres, effacer, réécrire. Cette scène évoque à Bricklin ses propres séances de calcul avec sa calculatrice : les feuilles de papier griffonnées, les erreurs qui se glissent dans les reports, les recommencements fastidieux.

L'étudiant a derrière lui une solide expérience technique acquise chez Digital Equipment Corporation, où il travaillait sur des systèmes de typographie numérique. Il connaît les systèmes informatiques qui cherchent à réduire le nombre de manipulations des opérateurs. Cette expérience le pousse à imaginer ce qu'il appellera plus tard un « tableau noir et une craie électroniques ». Un système où les calculs s'affichent, se modifient, se propagent automatiquement d'une cellule à l'autre.

De retour dans sa chambre d'étudiant, Bricklin bricolé un premier prototype en BASIC sur un Apple II emprunté. Son programme gère un petit

tableau de vingt lignes sur cinq colonnes, mais le principe fonctionne. Il retrouve Bob Frankston, un ami du MIT avec qui il a travaillé sur Multics, le système d'exploitation expérimental qui les a familiarisés avec les interfaces utilisateur sophistiquées. Frankston possède les compétences techniques nécessaires pour transformer l'idée en produit commercial.

Leur rencontre avec Dan Fylstra change la donne. Ce dernier dirige Personal Software et rédige dans *Byte Magazine*. Il saisit immédiatement l'intérêt du projet et oriente les deux hommes vers l'Apple II, machine qui dispose alors de l'avantage décisif du lecteur de disquettes. Sans support de stockage externe, un tableur reste inutilisable. Fylstra suggère aussi d'adapter le logiciel aux calculatrices programmables HP85 et HP87, mais cette piste ne débouchera sur rien.

En janvier 1979, Bricklin et Frankston fondent Software Arts Corporation. Pour développer leur tableur, ils acquièrent un mini-ordinateur Prime 350, choisi pour sa parenté avec Multics et son support du langage PL/I. Cette approche de développement croisé, inhabituelle pour l'époque, leur donne accès à des outils bien plus puissants que ceux disponibles sur micro-ordinateur. Ils écrivent et compilent le code sur le Prime, et le transfèrent vers l'Apple II pour les tests.

Les contraintes matérielles dictent leurs choix techniques. L'Apple II met à leur disposition 48 Ko de mémoire vive, dont une partie sert au système d'exploitation. Chaque octet compte. Plutôt que d'utiliser l'arithmétique binaire, plus rapide, ils optent pour le calcul décimal. Cette décision sacrifie les performances mais évite les erreurs d'arrondi qui déstabiliseraient des utilisateurs habitués à leurs calculatrices de bureau.

L'interface qu'ils conçoivent surprend par sa simplicité apparente. Une grille de cellules identifiées par des coordonnées : A1, B2, C3. Chaque case peut contenir un nombre, du texte ou une formule. L'utilisateur pointe vers d'autres cellules plutôt que de saisir manuellement leurs valeurs. Quand une donnée change, tous les calculs se mettent à jour instantanément. Cette fonctionnalité de simulation – le fameux « what if » – simplifie l'exploration de différents scénarios d'un geste.

Cette conception découle de leur expérience des gros systèmes informatiques et de leur volonté de créer un outil accessible au grand public. La notation par coordonnées résulte en réalité d'une réflexion approfondie sur l'ergonomie. Comment référencer une cellule de manière intuitive ? Comment rendre visible la logique des calculs ?

Personal Software commercialise VisiCalc en mai 1979 au prix de 100 dollars. Le succès ne vient pas immédiatement, mais il arrive vite. En juillet, Ben Rosen publie dans le *Morgan Stanley Electronics Letter* une analyse qui se révélera prophétique. VisiCalc est la première « killer app » de l'in-

formatique personnelle, le premier logiciel avec lequel on peut utiliser un ordinateur sans savoir programmer. Des acheteurs acquièrent un Apple II uniquement pour faire tourner VisiCalc.

Le programme trouve sa place sur d'autres machines. La version IBM PC de 1981 reprend les mêmes techniques d'optimisation que l'original, démontrant la justesse des choix initiaux. Mais le succès attire la concurrence. Microsoft lance Multiplan en 1982. Le logiciel peine aux États-Unis mais rencontre du succès ailleurs dans le monde.

L'arrivée de Lotus 1-2-3 en 1983 redistribue les cartes. Mitch Kapor, ancien employé de Personal Software, a développé un tableur spécialement optimisé pour l'IBM PC. Son programme intègre des graphiques, gère des bases de données, nomme les cellules et automatise les tâches avec des macros. Lotus 1-2-3 s'impose dans le monde professionnel.

VisiCalc ne résiste pas à cette concurrence renouvelée. Les relations entre Software Arts et Personal Software se dégradent. Chaque société développe ses propres projets : VisiOn pour Personal Software, TK !Solver pour Software Arts. Les conflits dégénèrent en procès croisés en 1984, paralysant l'évolution du produit. La version avancée de VisiCalc arrive trop tard sur un marché déjà conquis.

L'héritage du premier tableur électronique dépasse son succès commercial. VisiCalc prouve qu'un micro-ordinateur est apte à servir d'outil professionnel sérieux. Il accélère l'informatisation des entreprises et influence durablement la conception des interfaces utilisateur. Microsoft Excel, lancé en 1985 sur Macintosh et en 1987 sur Windows, reprend ses principes fondamentaux tout en ajoutant la programmation par macro-instructions.

Sur le plan technique, VisiCalc établit des pratiques qui deviennent courantes : le développement croisé entre machine de développement et plateforme cible, l'optimisation du code pour des ressources limitées, l'importance d'une interface intuitive. Mais cette facilité d'usage cache parfois la complexité sous-jacente. Les études révèlent que la majorité des tableurs contiennent des erreurs, avec environ 3 % de cellules erronées en moyenne. Ces fautes peuvent coûter cher en termes financiers ou de décisions stratégiques.

L'aventure VisiCalc illustre aussi l'évolution de l'industrie logicielle. Le modèle de collaboration entre développeur technique et éditeur commercial, d'abord perçu comme un atout, tend les relations quand le marché évolue. La séparation entre création et commercialisation, qui semblait naturelle, révèle ses limites face aux mutations technologiques.

Quarante ans après sa création, VisiCalc continue d'inspirer. Ses principes – interface utilisateur soignée, optimisation des ressources, équilibre entre innovation et besoins concrets – restent d'actualité. Son histoire rap-

pelle qu'un logiciel réussit autant par sa conception technique que par sa capacité à résoudre un problème réel.

VII

1980

L'informatique entre dans les foyers

La décennie 1980 s'ouvre dans un monde fracturé. La guerre froide n'a rien perdu de sa vigueur et l'opposition Est-Ouest continue de structurer les relations internationales. Pourtant, un autre terrain d'affrontement se dessine : la maîtrise de l'information. Les États-Unis et l'URSS poursuivent leur duel spatial et militaire tandis que le Japon, nouvelle puissance économique, conquiert marchés après marchés grâce à son industrie électronique.

L'information est un enjeu stratégique. Le démantèlement d'AT&T aux États-Unis en janvier 1984 bouleverse le secteur des télécommunications. Cette rupture avec le modèle monopolistique stimule l'innovation. Du côté européen, on commence à questionner la mainmise des PTT sur les communications. Un vent de changement souffle sur le paysage économique.

Le secteur tertiaire, en pleine expansion dans les économies occidentales, réclame des outils adaptés dans les bureaux. Les dactylos rangent leurs machines à écrire, remplacées par des systèmes de traitement de texte. Les comptables délaissent leurs calculatrices pour des tableurs électroniques. La productivité est le maître-mot face à une concurrence internationale toujours plus vive.

Le pouvoir d'achat des ménages occidentaux atteint un niveau sans précédent. Les téléviseurs couleur trônent désormais dans la plupart des

salons. Cette familiarité avec l'électronique grand public prépare le terrain pour l'arrivée d'autres appareils dont les consoles de jeux, les magnétoscopes et, surtout, les micro-ordinateurs. L'informatique, jusque-là cantonnée aux entreprises, s'invite chez les particuliers.

L'école n'échappe pas à cette déferlante technologique. Les responsables politiques saisissent l'importance de former la jeunesse à ces nouveaux outils. Le plan français « Informatique pour tous » lancé en 1985 illustre cette prise de conscience. Des milliers d'ordinateurs Thomson TO7 et MO5 envahissent les établissements scolaires. D'autres pays suivent des voies similaires, créant un marché éducatif substantiel.

Dans les entreprises, deux mondes cohabitent. Les grands systèmes centralisés restent présents, mais voient leur hégémonie contestée par les micro-ordinateurs. Les utilisateurs gagnent en indépendance, au grand dam des services informatiques traditionnels. Cette décentralisation soulève des questions inédites sur la compatibilité entre tous ces systèmes et le partage des données. La standardisation est la solution.

Le monde de la finance bascule dans l'ère électronique. Les bourses abandonnent la cotation manuelle pour des systèmes informatisés. Les transactions s'accélèrent, leur volume explose. Cette automatisation montre pourtant ses limites lors du « lundi noir » du 19 octobre 1987. Les programmes de vente automatique s'emballent, amplifiant le krach en cours. L'incident révèle la fragilité de systèmes trop autonomes.

Les réseaux de communication se modernisent à grande vitesse. La fibre optique remplace progressivement le cuivre. Les satellites de télécommunication se multiplient en orbite. Dans les entreprises, on ne se contente plus d'avoir des ordinateurs isolés, on les connecte. Les premières messageries électroniques commerciales apparaissent, prémices d'un monde interconnecté.

La culture populaire s'empare du phénomène informatique. Hollywood met en scène *hackers* et intelligences artificielles. *Tron* (1982) plonge ses spectateurs à l'intérieur d'un ordinateur. *WarGames* (1983) évoque le risque d'une guerre nucléaire déclenchée par une erreur informatique. La littérature n'est pas en reste avec l'émergence du cyberpunk, dont William Gibson et son *Neuromancien* (1984) constituent la figure de proue.

L'édition de logiciels s'affirme comme un secteur économique majeur. La multiplication des plateformes crée un marché fragmenté où chaque machine possède son écosystème propre. Les éditeurs rivalisent d'ingéniosité pour séduire le grand public. L'ergonomie est un argument de vente décisif. Dans l'ombre de cette effervescence commerciale, les premiers virus informatiques font leur apparition, annonciateurs de futures batailles numériques.

Un nouveau rapport à la technique se développe. L'informatique n'est

plus l'apanage des spécialistes en blouse blanche. Des clubs d'amateurs fleurissent, des revues comme *Hebdogiciel* ou *Byte* diffusent savoir-faire et programmes à taper soi-même. Cette démocratisation s'accompagne d'une philosophie particulière quand Richard Stallman lance en 1983 le projet GNU, première pierre d'un mouvement du logiciel libre qui conteste la mainmise des éditeurs commerciaux.

La Silicon Valley devient le symbole d'une nouvelle génération d'entrepreneurs. Des jeunes gens comme Steve Jobs, Bill Gates ou Michael Dell bousculent les géants établis. L'image du génie informatique travaillant dans un garage avant de conquérir le monde s'inscrit dans l'imaginaire collectif. Les investisseurs affluent, flairant le potentiel de croissance du secteur.

Les États prennent peu à peu conscience des enjeux de souveraineté liés à l'informatique. Contrôler les technologies numériques, c'est assurer son indépendance. La maîtrise des composants électroniques, des systèmes d'exploitation, des réseaux est un objectif stratégique. Les restrictions à l'exportation de matériel sensible vers le bloc soviétique témoignent de cette dimension géopolitique.

Cette décennie entame une transformation de la société. L'ordinateur est un outil de calcul scientifique ou de gestion, et un moyen de création, de communication et de divertissement. Son emprise s'étend à tous les domaines de l'activité humaine.

La fin des années 1980 laisse entrevoir un monde où l'information circule sans entraves. Les innovations techniques répondent aux aspirations sociales de l'époque, entre désir d'autonomie individuelle, besoin d'échanges accrus et soif de modernité. L'informatique personnelle s'inscrit dans cette dynamique, traduisant en silicium les rêves d'une génération.

Apple III

En mai 1980, lors de la National Computer Conference d'Anaheim, Apple dévoile sa nouvelle machine avec un faste inhabituel. L'entreprise privatise Disneyland cinq heures durant, débourse 42 000 dollars et affrète des bus à impériale britanniques pour transporter 7 000 participants de la conférence. Cette mise en scène grandiose annonce l'Apple III, premier ordinateur d'Apple destiné aux professionnels.

L'époque semble pourtant au beau fixe. Fin 1978, l'Apple II cartonne au-delà de toute espérance. Les ingénieurs de la Silicon Valley se disputent les postes chez Apple, certains acceptant des baisses de salaire contre des

stock-options. Richard Jordan, transfuge d'Hewlett-Packard durant l'été 1978, raconte cette période d'euphorie collective. À chaque division par deux du cours de l'action, les équipes se sentent invincibles. L'échec leur semble impossible.

Le projet débute dans ce contexte grisant. L'Apple III embarque un microprocesseur Synertek 6502A de 8 bits à 2 MHz, soit deux fois la vitesse de l'Apple II. Accueillant jusqu'à 128 Ko de mémoire vive, il dispose d'un clavier avec pavé numérique intégré et d'un lecteur de disquette 5,25 pouces de 143 Ko signé Shugart. Quatre slots d'extension internes restent compatibles avec les cartes Apple II, auxquels s'ajoutent deux ports série. La machine fonctionne certes en émulation Apple II, mais révèle sa vraie nature avec son système d'exploitation propriétaire SOS (*Sophisticated Operating System*), que les initiés surnomment « applesauce ». Une horloge temps réel intégrée complète l'équipement, tandis que l'affichage gère 24 lignes de 80 colonnes en texte et 560 par 192 pixels en graphique monochrome.

Steve Jobs dirige le projet et impose ses vues. Pas de ventilateur : le châssis en aluminium suffira à évacuer la chaleur. La taille et la forme du boîtier se décident sans consulter les ingénieurs, qui doivent ensuite entasser les composants dans un espace trop étroit avec une aération défaillante. Le « feature creep » aggrave la situation : marketing, ingénierie, design industriel, fabrication, chaque service ajoute ses exigences. Le produit enfle au-delà de sa conception initiale.

Jerry Manock et Dean Hovey, les designers industriels, conçoivent le châssis avant que la carte mère soit terminée. Manock anticipe des normes FCC strictes sur les interférences électromagnétiques et choisit un châssis en aluminium plutôt massif. Il passe contrat avec Doher-Jarvis, fabricant de pièces automobiles de Toledo. L'esthétique extérieure – chanfreins à 45 degrés, clavier incliné, couleur brune – doit créer une « identité maison » pour les futurs produits Apple.

Les retards menacent l'introduction en bourse prévue en décembre 1980. Les managers ignorent les alertes des ingénieurs et lancent la production. Dès novembre, lors des premières livraisons, le cauchemar commence. En février 1981, Apple abandonne l'horloge temps réel : la puce National Semiconductor ne tient pas ses promesses. Le prix chute à 4 190 dollars, avec un remboursement de 50 dollars pour les premiers clients.

Mars 1981 marque le début des livraisons en volume et l'ampleur du désastre. Vingt pour cent des machines arrivent mortes chez les clients, les puces s'étant délogées pendant le transport. Celles qui fonctionnent d'abord tombent vite en panne : la chaleur dilate les composants qui sortent de leurs supports. Apple préconise alors une solution qui restera dans les annales : soulever l'avant de l'ordinateur de quinze centimètres et le laisser retomber

pour réinsérer les puces.

D'autres défauts s'accumulent. Connecteurs défaillants, vis du boîtier qui percent les câbles internes, carte mère si dense qu'elle provoque des courts-circuits. La précipitation frappe aussi les logiciels : les programmeurs découvrent la machine neuf semaines seulement avant l'expédition. Les manuels sont relus le jour de leur envoi à l'impression, laissant passer tant d'erreurs qu'un addendum s'impose.

Apple tente un sauvetage en novembre 1981 avec une version révisée à 3 495 dollars. L'entreprise maintient que les problèmes venaient de la fabrication et du contrôle qualité, non de la conception. Pourtant, la nouvelle mouture intègre des supports de puces différents, un logiciel mis à jour, une mémoire extensible à 256 Ko et propose un disque dur de 5 Mo en option. Sur les 7 200 Apple III d'origine, 2 000 sont remplacés gratuitement.

Les ventes restent décevantes. Les analystes estiment qu'Apple écoule 3 000 à 3 500 unités mensuelles, soit un dixième des ventes Apple II. En décembre 1983, InfoCorp recense 75 000 unités installées contre 1,3 million d'Apple II. La mauvaise réputation et le manque de logiciels exploitant SOS rebutent les acheteurs.

Un dernier baroud d'honneur survient en décembre 1983 avec l'Apple III Plus à 2 995 dollars. Cette version inclut 256 Ko de RAM standard, une horloge fonctionnelle, une nouvelle carte mère, SOS version 1.3, des ports améliorés avec connecteurs DB-25 et un boîtier facilitant l'installation des cartes. Trop tard : le 24 avril 1984, Apple arrête le développement. Après 60 millions de dollars de pertes, la gamme disparaît définitivement du catalogue en septembre 1985.

L'ironie veut que cet échec n'entame pas la réussite d'Apple. Le 12 décembre 1980, l'introduction en bourse bat tous les records. Les actions émises à 22 dollars grimpent à 29 dollars et se vendent en minutes. À la clôture du premier jour, Apple vaut 1,2 milliard de dollars, réalisant la plus grosse introduction depuis Ford en 1956. Les 15% détenus par Jobs dépasseront bientôt 250 millions de dollars.

L'expérience Apple III brise le mythe d'invincibilité né du succès Apple II. Ce revers force une restructuration. En janvier 1981, le président Mike Scott divise le développement en trois groupes : systèmes de bureau personnels (Apple II/III), périphériques (lecteurs, imprimantes, modems) et systèmes professionnels (division Lisa). Le budget R&D triple à 21 millions de dollars. Ces tensions coûtent son poste à Scott, remplacé par Mike Markkula, tandis que Jobs prends la place de président du conseil.

L'arrivée d'IBM en 1981 sur le marché des ordinateurs personnels servira paradoxalement Apple. Le géant légitime ce secteur émergent. Apple le reconnaît avec humour dans une publicité pleine page du *Wall Street*

Journal : « Bienvenue IBM, vraiment. »

Intel 8086/8088

Lorsque Intel décide de franchir le cap des processeurs 16 bits à la fin des années 1970, l'entreprise ne sait pas encore qu'elle va devenir un empire technologique. Le 8086, lancé en 1978, arrive dans un marché dominé par les puces 8 bits d'Intel, les 8080 et 8085. Cette transition vers les 16 bits représente un défi technique considérable, mais Intel mise sur une stratégie audacieuse : conserver la compatibilité avec l'existant tout en multipliant les performances.

Le nouveau processeur repousse les limites de l'époque. Là où ses prédécesseurs plafonnaient à 64 kilo-octets de mémoire, le 8086 adresse jusqu'à 1 mégaoctet grâce à son bus d'adresses de 20 bits. Ses 16 bits de bus de données traitent des mots doubles en un seul cycle. Intel a conçu une architecture à deux têtes : l'unité d'interface bus gère les échanges avec l'extérieur pendant que l'unité d'exécution se concentre sur les calculs. Cette division du travail autorise un fonctionnement en pipeline, où le processeur mâche l'instruction suivante avant d'avoir fini la précédente.

Gravé en technologie HMOS avec ses 29 000 transistors, le 8086 tourne à 5 MHz dans sa version de base. Intel pousse ensuite la cadence à 8 puis 10 MHz. Ses quatorze registres 16 bits se répartissent astucieusement entre données, pointeurs, drapeaux et segments. Ces derniers constituent l'innovation la plus marquante : ils découpent la mémoire en zones distinctes et facilitent l'organisation des programmes.

Mais voilà le hic. Migrer du 8 bits vers le 16 bits coûte cher aux constructeurs. Il faut redessiner les cartes mères, adapter tous les composants périphériques, revoir l'architecture des machines. Face à ces réticences du marché, Intel sort en 1979 une version tronquée : le 8088. Cette puce garde l'architecture interne 16 bits du 8086 mais rabote son bus de données externe à 8 bits.

Le compromis paraît bancal au premier regard. Les transferts 16 bits du 8088 nécessitent deux cycles au lieu d'un, bridant les performances. Pourtant, cette limitation technique se transforme un atout commercial car les fabricants peuvent réutiliser leurs composants 8 bits existants : contrôleurs, mémoires, circuits de support. Le coût de développement chute drastiquement.

IBM saisit l'occasion en 1981. Pour son premier ordinateur personnel, la

firme cherche une solution rapide et économique. Le 8088 s'impose naturellement. Il se marie parfaitement avec le contrôleur de bus 8288, le gestionnaire d'interruptions 8259A et toute la panoplie de circuits déjà éprouvés. Sa mémoire cache de 6 octets compense partiellement le goulot d'étranglement du bus réduit.

En étalon du marché, l'IBM PC cartonne en quelques années. Microsoft développe MS-DOS sur mesure pour cette architecture, créant un écosystème logiciel qui verrouille la concurrence. Les clones IBM pullulent, tous équipés du 8088. La puce d'Intel est le processeur le plus vendu, détrônant largement son grand frère 8086 pourtant plus performant.

Cette réussite commerciale masque le changement technique. L'architecture du 8088/8086 invente des concepts qui traverseront les décennies. La segmentation mémoire survivra dans toutes les générations x86 suivantes. Le jeu d'instructions étendu, avec ses opérations sur chaînes et sa multiplication matérielle, servira de socle aux extensions futures. Le mode protégé du 80286 s'appuiera sur ces fondations.

Véritable marque de fabrique de la famille x86, Intel prévoit dès l'origine la cohabitation des deux modèles. Même jeu d'instructions, même programmation, seul le bus externe diffère. Chaque nouveau processeur exécutera les logiciels de ses prédécesseurs tout en ajoutant ses propres innovations.

Pour les calculs sophistiqués, Intel propose le coprocesseur mathématique 8087. Ce circuit optionnel se branche directement sur le processeur principal et prend en charge la virgule flottante. Son jeu d'instructions spécialisé sera plus tard intégré nativement dans les puces à partir du 80486.

L'architecture modulaire du 8086/8088 autorise aussi le multiprocesseur. En mode maximum, plusieurs puces se partagent le bus système via le contrôleur 8288. Cette possibilité, ignorée dans les PC grand public, trouve sa place dans l'industrie et les serveurs.

La production s'étale sur plus de dix ans. Supplantés par des processeurs plus musclés, les 8086 et 8088 équipent encore longtemps les systèmes embarqués et les automates industriels. Leur simplicité et leur frugalité énergétique conviennent parfaitement à ces usages discrets.

Quarante ans après, leur influence perdure. En imposant x86 comme standard de facto des ordinateurs personnels, ils ont tracé une route technologique qui se prolonge. La nécessité de préserver la compatibilité avec leur jeu d'instructions pèse toujours sur les processeurs contemporains.

Cette compatibilité ascendante, d'abord perçue comme un boulet, s'est muée en arme absolue. Elle garantit aux développeurs et utilisateurs que leurs programmes survivront aux évolutions matérielles. Cette stabilité a dopé l'expansion du marché des PC compatibles IBM.

Le triomphe du 8088 dans l'IBM PC illustre une vérité éternelle de l'industrie technologique : la solution optimale sur le papier ne l'emporte pas toujours. Le pragmatisme économique et les contraintes pratiques dictent souvent les choix. Intel l'a compris et a bâti sa stratégie sur cette leçon.

Smalltalk-80

À la fin des années 1970, le Xerox PARC abritait déjà plusieurs versions de Smalltalk, mais ces créations demeuraient prisonnières des murs du laboratoire. Elles ne fonctionnaient que sur le matériel spécialisé de Xerox : l'Alto et le Dorado, machines coûteuses et inaccessibles au commun des informaticiens. Cette situation frustrante pousse Adele Goldberg à prendre les rênes d'un projet ambitieux : faire sortir Smalltalk de sa bulle dorée pour qu'il puisse s'épanouir sur d'autres architectures.

L'entreprise s'annonce délicate. Il faut d'abord formaliser avec une précision chirurgicale le fonctionnement d'une « machine virtuelle » Smalltalk qui servirait de socle universel, tout comme celle de Java bien des années plus tard. Cette abstraction technique cache un nouveau concept : tout est objet dans ce nouvel univers. Les nombres, les caractères et les classes se comportent comme des entités autonomes capables de recevoir et d'envoyer des messages. Cette uniformité radicale bouscule les habitudes des programmeurs habitués à séparer soigneusement données et procédures.

Dans ce monde unifié, l'héritage tisse des liens invisibles entre les classes. Une hiérarchie se dessine naturellement, où chaque descendant enrichit ou spécialise les comportements de ses ancêtres. Les méthodes de classe orchestrent les comportements partagés tandis que les méthodes d'instance donnent à chaque objet sa personnalité propre. Cette architecture favorise une forme de réutilisation du code inconnue jusqu'alors.

L'environnement de développement rompt avec le cycle traditionnel qui imposait d'éditer, compiler puis exécuter. Smalltalk-80 propose une expérience continue où le programmeur modifie son code pendant que le programme s'exécute. Cette fluidité transforme l'acte de programmer en une conversation directe avec la machine. Le navigateur de classes et le débogueur, eux-mêmes écrits en Smalltalk, sont des outils vivants que l'utilisateur façonne selon ses besoins.

L'interface graphique introduit des concepts innovants pour l'époque : les fenêtres se superposent, les menus apparaissent au clic droit, les objets se manipulent directement à l'écran. Ces innovations établissent un langage

visuel qui influence encore nos interactions quotidiennes avec les ordinateurs. Steve Jobs ne s'y trompe pas lorsqu'il visite le PARC en 1979 ; il comprend immédiatement le potentiel de ces idées pour le grand public.

Sous le capot, la gestion de la mémoire automatique libère enfin les programmeurs d'une corvée technique fastidieuse. Le ramasse-miettes traque les objets abandonnés pour recycler leur mémoire. Peter Deutsch et Allan Schiffman peaufinent ces mécanismes jusqu'à obtenir des performances acceptables sur du matériel standard, prouvant que l'élégance conceptuelle peut cohabiter avec l'efficacité pratique.

L'implémentation sur différentes plateformes exige des prouesses d'ingénierie. L'équipe du PARC dessine une machine virtuelle avec son propre jeu d'instructions, laissant aux intégrateurs le soin de traduire ces commandes abstraites en code natif. Cette stratégie garantit la portabilité tout en préservant la possibilité d'optimisations spécifiques. Les caches de méthodes accélèrent la recherche dans la hiérarchie des classes : quand un objet reçoit un message, le système mémorise l'association entre le sélecteur et la méthode correspondante pour les appels ultérieurs.

La gestion des contextes d'exécution révèle une autre facette de l'approche objet. Ces contextes, objets manipulables par le programme, permettront à des mécanismes sophistiqués comme la gestion d'exceptions et la réflexion de voir le jour. Cette uniformité conceptuelle pousse la logique objet jusque dans les rouages les plus intimes du système.

Avec la publication de Smalltalk-80 dans un numéro spécial du magazine Byte en 1980, les livres d'Adele Goldberg et David Robson diffusent ces idées bien au-delà des cercles académiques. Objective-C, Ruby, Python et tant d'autres puisent directement dans cette source. L'influence dépasse le domaine des langages : les environnements de développement modernes gardent la trace de cette vision où tout est accessible et modifiable en temps réel.

L'interface de Smalltalk-80 essaime dans le monde commercial grâce à la perspicacité de Jobs. Le Lisa et le Macintosh popularisent ces concepts auprès du grand public, créant un standard *de facto* qui perdure. Les techniques d'optimisation mises au point pour Smalltalk-80 sont désormais des pratiques courantes dans l'intégration des langages orientés objet.

La documentation exhaustive du système, incluant les spécifications de la machine virtuelle et les détails d'implémentation, constitue un cadeau inestimable à la communauté informatique. D'autres équipes vont ainsi réinventer le système selon leurs besoins, accélérant la propagation de ces idées pionnières.

UDP

En 1980, David P. Reed travaillait sur un problème simple en apparence : comment transmettre des données entre ordinateurs sans toute la complexité que TCP imposait déjà ? L'architecture d'Internet prenait forme autour de la suite TCP/IP, mais certaines applications réclamaient quelque chose de plus direct, de plus rapide. Reed imagina alors UDP, le User Datagram Protocol, qui allait devenir l'un des protocoles les plus durables d'Internet.

Là où TCP construisait des connexions solides, vérifiait chaque paquet, garantissait l'ordre d'arrivée des données, UDP choisissait délibérément l'inverse. Pas de connexion persistante, aucune garantie de livraison, aucun contrôle de l'ordre des paquets. Cette approche minimaliste se reflète d'ailleurs dans le RFC 768 qui définit le protocole : trois petites pages suffisent là où d'autres spécifications s'étendent sur des centaines de pages.

Cette brièveté n'était pas de la négligence. Reed et ses collègues avaient compris qu'Internet aurait besoin de deux types d'outils : des protocoles sophistiqués comme TCP pour les données critiques, et des mécanismes légers comme UDP pour d'autres usages. Le premier service à vraiment exploiter cette philosophie fut l'Internet Name Service, l'ancêtre du DNS. Quand un ordinateur demande l'adresse IP d'un site web, il n'a pas besoin d'établir une connexion complexe ; il envoie sa question et attend la réponse. Si celle-ci se perd, il redemande. Simple, basique.

Le DNS utilise encore cette approche, et pour cause, un serveur traite des milliers de requêtes par seconde sans maintenir d'état pour chaque client. Cette économie de ressources s'avère précieuse quand on sait qu'un serveur DNS racine peut recevoir des millions de requêtes par heure.

L'histoire d'UDP prit une tournure inattendue avec l'émergence des communications multimédias. Quand la voix sur IP fit ses premiers pas dans les années 1990, les développeurs découvrirent qu'UDP correspondait parfaitement à leurs besoins. Dans une conversation téléphonique, mieux vaut perdre quelques millisecondes d'audio que d'attendre qu'un paquet manquant soit retransmis. L'expérience utilisateur prime sur la perfection technique. Cette logique s'étendit naturellement à la vidéoconférence, puis aux jeux en ligne. Dans un jeu de tir en ligne, si l'information indiquant qu'un joueur a tiré arrive en retard, elle est inutile. Les développeurs gèrent eux-mêmes ces problématiques temporelles avec UDP, selon les spécificités de leur application.

L'élégance d'UDP réside dans cette flexibilité. Le protocole ne dicte

pas comment gérer les erreurs ou l'ordre des paquets ; il laisse cette responsabilité aux applications. Certains jeux retransmettent les informations critiques plusieurs fois pour s'assurer de leur réception. D'autres ignorent simplement les paquets perdus et se concentrent sur les données les plus récentes.

Cette approche a inspiré de nouvelles innovations. DCCP ajoute le contrôle de congestion à UDP pour éviter de surcharger le réseau, et DTLS apporte la sécurité cryptographique. Ces protocoles montrent qu'UDP sert de fondation stable pour construire des solutions spécialisées.

Les mesures de performance confirment les intuitions initiales de Reed. Une étude menée entre 2004 et 2014 sur un lien transpacifique entre le Japon et les États-Unis révéla des variations dans la taille moyenne des paquets IP. UDP, générant moins de trafic de contrôle que TCP, contribue à une utilisation plus efficace de la bande passante.

L'émergence des réseaux de capteurs a donné une nouvelle jeunesse à UDP. Ces dispositifs, souvent alimentés par batterie, transmettent périodiquement des mesures de température, d'humidité ou de position. Ils n'ont pas besoin de la robustesse de TCP ; ils préfèrent économiser leur énergie en transmettant simplement leurs données sans établir de connexion complexe.

Le cloud computing et les microservices ont redécouvert les vertus de UDP. Quand une application décomposée en dizaines de services doit communiquer rapidement, UDP offre une latence minimale. Netflix utilise ainsi UDP pour certaines communications internes, Spotify pour synchroniser les playlists entre appareils.

L'évolution vers le variant UDP-Lite témoigne de cette adaptabilité, tolèrant certaines erreurs dans les en-têtes de paquets, utile dans les communications sans fil où quelques bits corrompus ne justifient pas de rejeter l'ensemble d'un paquet.

Quarante-trois ans après sa création, UDP continue de prospérer sur Internet. Sa longévité tient à un principe simple : plutôt que d'essayer de résoudre tous les problèmes, il en résout quelques-uns parfaitement bien. Cette spécialisation lui a permis de traverser les époques, des premiers réseaux universitaires aux centres de données modernes. Cette constance en fait l'un des piliers discrets mais indispensables d'Internet.

Commodore VIC-20

En 1980 près de Londres, Jack Tramiel convoque ses équipes pour une réunion qui changera la donne. Le patron de Commodore exprime une préoccupation qui le hante : « Les Japonais arrivent, alors nous allons devenir les Japonais ». Cette phrase lapidaire résume toute la philosophie qui va présider à la naissance du VIC-20.

Tramiel voit loin. Son empire Commodore règne certes en Europe, mais aux États-Unis, Apple et Radio Shack gardent une longueur d'avance. Plus inquiétant, Texas Instruments grignote ses positions sur le marché des calculatrices. Le dirigeant redoute par-dessus tout que les constructeurs japonais débarquent avec des machines subventionnées qui lamineraient la concurrence. Sa réponse ? Créer un ordinateur personnel couleur accessible au grand public, une machine qui viendrait compléter la gamme des PET déjà établie.

Cette vision audacieuse s'appuie sur un élément technique oublié depuis deux ans dans les tiroirs de MOS Technology. En 1978, Alan Charpentier avait conçu le VIC (Video Interface Chip) pour séduire les fabricants de bornes d'arcade. L'accueil avait été froid, et la puce dormait depuis. Mais voilà qu'elle trouve enfin sa vocation dans ce projet d'ordinateur grand public.

Mike Tomczyk débarque en avril 1980 comme assistant de Tramiel. En quelques semaines, cet homme énergique incarne l'âme du projet VIC-20, au point de mériter le surnom de « VIC Czar ». En visitant l'Allemagne et le Japon en un mois, il fonde son programme pour restructurer le marketing.

Pendant ce temps, deux équipes rivalisent d'ingéniosité. Robert Yannes, qui créera plus tard la fameuse puce sonore SID, bricole chez MOS Technology un premier prototype avec des composants de PET et un boîtier de calculatrice. Sa vision pencherait plutôt vers une console de jeux sophistiquée. De leur côté, Bill Seiler et John Feagans assemblent un autre prototype en mélangeant différents éléments. Ils ajoutent un port joystick à 9 broches et un connecteur cartouche inspiré de l'Atari 2600, mais insistent surtout pour intégrer le BASIC. Leur conviction : un ordinateur doit permettre de programmer, pas seulement de jouer.

Le prototype final marie ces deux visions. La machine qui en résulte tient ses promesses techniques : un processeur MOS 6502A tournant à environ 1 MHz, 5 Ko de RAM dont 3,5 Ko exploitables, des connecteurs pour cassette et disque, un port joystick, un port utilisateur polyvalent et quatre canaux audio mono générés par la puce VIC. L'ensemble tient dans un boîtier compact aux lignes épurées.

Commodore teste d'abord ses armes au Japon. En octobre 1980, le VIC-

1001 sort à 69 800 yens. Tout en rodant la production, cette stratégie d'observation de la réaction des constructeurs nippons s'avère payante. En mai 1981, le VIC-20 débarque aux États-Unis à 299,95 dollars, et au Royaume-Uni en septembre à 199,99 livres sterling.

Tomczyk déploie alors sa stratégie marketing. Il dépose l'expression « the friendly computer » et contourne les revendeurs spécialisés pour investir les grandes surfaces. Cette approche place délibérément le VIC-20 en concurrence avec les consoles de jeux, mettant en avant ses capacités d'ordinateur personnel. La documentation reflète cette philosophie : le manuel privilégie la simplicité et laisse les subtilités techniques au Guide de Référence du Programmeur.

William Shatner, l'inoubliable capitaine Kirk de *Star Trek*, prête son visage charismatique à la campagne publicitaire. Ses apparitions télévisées marquent les esprits et donnent au VIC-20 une notoriété inespérée. Les résultats suivent. En 1982, les usines Commodore produisent jusqu'à 9 000 unités par jour pour satisfaire une demande qui explose. Le chiffre d'affaires atteint 300 millions de dollars.

L'écosystème logiciel s'étoffe à vitesse grand V. Les jeux dominent naturellement, avec des créations marquantes comme *Jelly Monsters*, adaptation non autorisée mais réussie de *Pac-Man*, ou *Sword of Fargoal* qui exploite habilement les capacités de la machine. Jeff Minter, futur gourou du jeu indépendant, signe ses premières œuvres sur VIC-20 avec *Matrix* et *Laser Zone*. Le VICModem, premier modem vendu sous la barre des 100 dollars, ouvre les portes du réseau CompuServe via le Commodore Information Network. Les cartouches de 3, 8 ou 16 Ko étendent la mémoire jusqu'à 32 Ko et multiplient les possibilités.

Le succès dépasse toutes les espérances. Le VIC-20 est le premier ordinateur personnel à franchir le cap du million d'unités vendues. Cette performance historique témoigne de la justesse de la vision de Tramiel : l'ordinateur domestique peut séduire bien au-delà du cercle des passionnés.

Pourtant, ce triomphe porte le germe de sa propre fin. En août 1982, Commodore lance le Commodore 64, machine plus puissante qui cannibalise les ventes de son petit frère. La production du VIC-20 s'arrête fin 1984, et les derniers exemplaires disparaissent des rayons début 1985. Environ 2,5 millions d'unités auront trouvé preneur en quatre années d'existence commerciale, performance remarquable mais éclipsée par la longévité du C64 ou du ZX Spectrum.

Cette disparition prématurée ne diminue en rien l'héritage du VIC-20. La machine finance le développement du Commodore 64, prépare l'acquisition de l'Amiga et établit les codes de l'informatique domestique des années 1980. Sa conception influence durablement l'industrie : compatibilité des

périphériques, importance de l'ergonomie, démocratisation assumée. Toute une génération de programmeurs fait ses armes sur cette machine, notamment au Royaume-Uni où elle catalyse la créativité.

Une communauté passionnée maintient la flamme et de nouveaux programmes voient régulièrement le jour, comme *Astro Nell* ou *Game Theory*, qui révèlent des capacités insoupçonnées malgré les contraintes matérielles. Ces créations récentes témoignent de la justesse du concept initial : un ordinateur simple et accessible qui libère la créativité plutôt qu'il ne l'entrave.

WordPerfect

L'été 1979, Alan Ashton, professeur d'informatique à l'université Brigham Young, décide de consacrer ses vacances à un projet qui lui tient à cœur : créer un logiciel qui affiche le texte exactement comme il apparaîtra sur le papier. À l'époque, les systèmes existants fonctionnaient avec des codes de mise en forme disgracieux qui polluaient l'affichage. Dans son laboratoire d'Orem, Utah, Ashton pose les premières pierres de ce qui deviendra l'un des succès les plus retentissants de l'informatique personnelle.

Bruce Bastian, ancien chef d'orchestre de l'université reconverti dans l'informatique, apporte son expertise technique au projet. Ensemble, ils développent SSI*WP, l'acronyme de Satellite Software International Word Processor. Le choix du nom révèle leurs ambitions : ils veulent dépasser les frontières géographiques traditionnelles du logiciel.

La commercialisation débute en 1980 avec un prix, qui fait réfléchir, de 5 500 dollars pour une licence destinée aux ordinateurs Data General. Le marché réagit avec prudence. Les ventes restent modestes mais suffisantes pour que la petite équipe garde le contrôle de son destin. Don Owens rejoint l'aventure comme responsable commercial et commence à tisser un réseau de revendeurs. Pete Peterson, ancien comptable, prend en charge les aspects financiers en 1981. Cette équipe fondatrice va se diviser sur une question : faut-il accepter des investissements externes pour accélérer la croissance ?

Owens plaide pour une levée de fonds qui donnerait les moyens d'une expansion rapide. Ashton et Bastian préfèrent une approche plus prudente, quitte à progresser plus lentement. Ces visions incompatibles créent des tensions qui explosent en 1982 avec le départ d'Owens. Rétrospectivement, ce choix de l'indépendance financière s'avérera déterminant pour la suite.

L'arrivée du PC IBM bouleverse la donne. Cette machine démocratise l'informatique personnelle et ouvre des perspectives inouïes. WordPerfect

sort sa première version pour PC à la fin 1982. Le logiciel tranche avec la concurrence par sa stabilité remarquable et son ergonomie soignée. Les plantages sont rares, les fonctionnalités nombreuses. La correction orthographique intégrée fait sensation, tout comme la gestion native des notes de bas de page.

La version 4.2, lancée en 1986, propulse WordPerfect vers des sommets inattendus. Plus de la moitié des utilisateurs de traitement de texte choisissent désormais cette solution. Ce succès ne doit rien au hasard. L'entreprise, rebaptisée WordPerfect Corporation, révolutionne les codes du secteur en proposant un support technique gratuit par téléphone. Cette innovation commerciale bouleverse les habitudes : jusqu'alors, obtenir de l'aide coûtait cher et prenait du temps.

L'interface utilisateur de WordPerfect cultive sa différence. Les touches de fonction remplacent les menus déroulants qui équipent déjà d'autres logiciels. Un petit gabarit en carton, posé sur le clavier, rappelle les raccourcis disponibles. Cette approche déroute au début mais révèle vite ses avantages. Grâce à la fonction « Reveal Codes », qui dévoile tous les codes de formatage cachés dans le document, l'efficacité des utilisateurs aguerris redouble. Les avocats et notaires s'entichent de cette transparence qui leur garantit un contrôle total sur la mise en forme de leurs actes.

La croissance de l'entreprise défie l'entendement. De 11 employés en 1981, elle passe à plus de 4 500 en 1992. Le chiffre d'affaires atteint 533 millions de dollars en 1991. Cette expansion fulgurante génère des débats internes sur l'organisation. Peterson défend une structure plate, avec peu de niveaux hiérarchiques, tandis que d'autres dirigeants réclament un management plus classique. Ces désaccords provoquent le départ de Peterson en 1992, privant l'entreprise d'une vision organisationnelle cohérente.

Microsoft Windows 3.0 arrive en 1990. Cette interface graphique moderne séduit les utilisateurs par sa convivialité. WordPerfect commet alors une erreur stratégique majeure : l'entreprise tarde à adapter son logiciel à cet environnement, restant attachée au mode texte de MS-DOS. Microsoft Word for Windows profite de ce retard pour s'imposer. Intégré dans la suite Office, il bénéficie d'une cohérence visuelle et fonctionnelle que WordPerfect peine à rattraper.

Quand WordPerfect for Windows sort enfin en 1991, le mal est fait. Le logiciel souffre de problèmes de performance et d'ergonomie. Les utilisateurs, séduits par la modernité de l'interface graphique, basculent massivement vers la solution de Microsoft. Cette transition marque le début du déclin pour WordPerfect.

Novell rachète WordPerfect Corporation en 1994 pour 1,4 milliard de dollars. Cette acquisition pharaonique tourne au désastre. Le géant des ré-

seaux informatiques ne comprend rien au marché du traitement de texte. Les ventes s'effondrent face à l'hégémonie grandissante de Microsoft Office. Deux ans plus tard, Corel récupère WordPerfect pour seulement 158 millions de dollars, soit près de dix fois moins que le prix d'achat initial.

Cette chute spectaculaire illustre la brutalité des retournements dans l'industrie logicielle. WordPerfect dominait son marché grâce à l'excellence technique et un service client irréprochable. Mais l'entreprise n'a pas su anticiper l'importance de l'interface graphique. Elle a sous-estimé la force de frappe de Microsoft, qui contrôlait à la fois le système d'exploitation et pouvait imposer sa suite bureautique comme standard de fait.

WordPerfect conserve une base d'utilisateurs fidèles, concentrée dans le monde juridique. Les professionnels du droit apprécient toujours sa précision dans le formatage des documents complexes. La fonction « Reveal Codes » est unique en son genre et justifie à elle seule la fidélité de certains utilisateurs.

Avec WordPerfect, l'entreprise a imposé le support client gratuit comme norme industrielle. Ses innovations en matière d'affichage WYSIWYG et de correction orthographique ont inspiré tous ses successeurs. En informatique, le leadership technique ne suffit pas : il faut aussi savoir s'adapter aux ruptures technologiques et comprendre les attentes changeantes des utilisateurs. L'industrie du logiciel est impitoyable. Une entreprise peut dominer un marché pendant des années puis sombrer en quelques mois si elle manque un virage technologique.

ICMP

En 1981, les ingénieurs travaillant sur ARPANET se heurtent à un problème récurrent pour faire savoir à un expéditeur que son paquet s'est perdu en route. Le protocole IP, dans sa conception initiale, ne prévoit aucun mécanisme pour signaler les erreurs. Les paquets disparaissent dans les méandres du réseau sans laisser de trace, et l'expéditeur reste dans l'ignorance du sort de ses données.

Cette lacune, criante avec l'expansion d'ARPANET, Jon Postel et Steve Crocker l'observent depuis UCLA. L'absence de retour d'information paralyse littéralement le diagnostic des pannes. Un routeur surchargé rejette un paquet mais l'expéditeur l'ignore. Une destination est injoignable mais il n'y a aucun moyen de le savoir. Cette situation pousse à la création d'ICMP, l'Internet Control Message Protocol.

Le RFC 792 formalise cette solution en définissant deux grandes familles de messages : ceux qui signalent les erreurs et ceux qui interrogent l'état du réseau. La première catégorie informe l'expéditeur des problèmes rencontrés : destination inaccessible, durée de vie du paquet expirée, paramètres incorrects. La seconde famille sert à sonder le réseau, vérifier qu'un hôte répond ou mesurer le temps de trajet des paquets.

L'architecture technique d'ICMP est particulièrement surprenante. Plutôt que de se positionner au-dessus d'IP comme un protocole de niveau supérieur, ICMP s'y intègre directement. Les messages de contrôle voyagent dans des datagrammes IP ordinaires, empruntant les mêmes chemins que les données qu'ils concernent. Cette approche garantit que les messages d'erreur suivent une route identique à celle des paquets défaillants.

Mike Muuss bouleverse l'usage d'ICMP en 1983 avec son utilitaire `ping`. Cette petite application exploite les messages « echo request » et « echo reply » pour créer le test de connectivité le plus populaire de l'Internet. Le principe séduit par sa simplicité : envoyer un message à une machine et attendre sa réponse. Si elle répond, la connexion fonctionne. Sinon, un problème existe quelque part sur le chemin.

Le succès de `ping` inspire d'autres outils. `Traceroute` détourne astucieusement les messages « time exceeded » pour cartographier les routes réseau. En envoyant des paquets avec des durées de vie progressivement croissantes, l'outil force chaque routeur intermédiaire à renvoyer un message d'erreur, révélant ainsi son identité et sa position dans le chemin.

La découverte du MTU de chemin illustre l'adaptabilité d'ICMP face aux besoins émergents. Cette technique utilise les messages « fragmentation needed » pour déterminer la taille maximale des paquets traversant un chemin réseau sans fragmentation. Le procédé améliore sensiblement les performances en évitant la division des paquets en fragments plus petits.

Mais ICMP attire aussi l'attention des pirates informatiques. Les attaques « ping of death » exploitent des failles dans le traitement des paquets ICMP surdimensionnés, provoquant le plantage de systèmes vulnérables. D'autres attaques utilisent ICMP pour saturer les réseaux de messages de contrôle. Ces détournements malveillants poussent de nombreux administrateurs à filtrer ou bloquer certains messages ICMP, créant parfois des effets de bord indésirables sur le fonctionnement normal des réseaux.

L'arrivée d'IPv6 redéfinit le rôle d'ICMP. La nouvelle version, ICMPv6, absorbe des fonctions auparavant assurées par d'autres protocoles. La découverte des voisins, par exemple, remplace ARP et s'intègre directement dans ICMPv6. Cette consolidation simplifie l'architecture réseau tout en renforçant l'importance d'ICMP dans les infrastructures modernes.

Le protocole ICMP influence la conception d'outils de surveillance ré-

seau sophistiqués qui analysent les messages de contrôle pour détecter les anomalies et mesurer les performances. Les administrateurs réseau apprennent à interpréter ces signaux pour diagnostiquer les pannes et anticiper les problèmes.

Cette transformation d'un simple protocole de contrôle en pierre angulaire de l'infrastructure Internet témoigne de sa conception judicieuse. Sa flexibilité lui a permis de s'adapter aux besoins changeants des réseaux informatiques pendant plus de quarante ans. Aujourd'hui, chaque `ping` lancé sur Internet perpétue l'héritage de ces premiers ingénieurs d'ARPANET qui voulaient simplement savoir pourquoi leurs paquets se perdaient.

Acorn BBC Micro

En 1980, la BBC traverse une période de questionnements sur l'avenir numérique du Royaume-Uni. Les responsables de la chaîne publique britannique observent avec inquiétude la domination américaine sur le marché de la micro-informatique. Les machines disponibles coûtent une fortune ou se limitent à des capacités dérisoires. Face à ce constat, la BBC décide de prendre les choses en main et lance un projet d'envergure : créer un ordinateur personnel qui accompagnerait ses émissions éducatives.

Le cahier des charges établi par la BBC reflète cette ambition pédagogique. La machine devra intégrer le langage BASIC, disposer d'un véritable clavier digne de ce nom, afficher des graphiques en couleur et accepter des extensions comme le télétexte. Surtout, son prix devra rester à la portée des établissements scolaires et des particuliers motivés. Un défi technique et commercial considérable.

L'appel d'offres lancé fin 1980 attire plusieurs candidats. Parmi eux, Acorn Computers, une petite entreprise de Cambridge qui travaille sur un prototype baptisé Proton. Cette machine impressionne les évaluateurs de la BBC, qui lui préfèrent notamment le Sinclair ZX-81 encore en gestation. Acorn décroche le contrat et renomme officiellement son prototype le « BBC Microcomputer ».

La collaboration entre les ingénieurs d'Acorn et les équipes de la BBC se révèle fructueuse. Le prototype initial subit de nombreuses améliorations qui dépassent largement les spécifications d'origine. ICL et Cleartone se chargent de la fabrication. L'architecture s'appuie sur le processeur 6502A cadencé à 2 MHz, une puce fiable et éprouvée. Deux circuits intégrés spécialisés complètent l'ensemble : l'un gère les graphiques, l'autre les interfaces série.

Deux versions sortent simultanément début 1982. Le modèle A se vend 235 £ avec 16 Ko de mémoire vive, tandis que le modèle B affiche 335 £ pour 32 Ko. Ce dernier embarque davantage de connectivité : interfaces série et parallèle, port utilisateur 8 bits, entrées analogiques et bus d'extension. Une originalité technique mérite d'être soulignée : l'interface « Tube » qui autorise la connexion d'un second processeur.

Le BASIC intégré surprend par sa richesse. Proche du Microsoft BASIC dans sa philosophie, il se distingue par de nombreuses extensions spécifiques. Le système d'exploitation et le BASIC monopolisent 32 Ko de mémoire morte, une allocation généreuse. Le clavier professionnel compte 64 touches disposées selon le standard QWERTY, complété par 10 touches de fonction programmables.

Les performances graphiques constituent un atout majeur. Huit modes différents permettent de mélanger texte et graphiques haute résolution selon les besoins. Le mode 7, compatible télétexte, produit des graphiques couleur en ne consommant qu'un seul Ko de mémoire. Une puce sonore génère trois notes simultanées et divers effets, avec un contrôle logiciel sophistiqué des enveloppes sonores.

L'extensibilité distingue le BBC Micro de ses concurrents. Les interfaces pour lecteur de disquettes et réseau Econet sont directement intégrées sur la carte mère. Une option de synthèse vocale, développée avec Richard Baker, présentateur vedette de la BBC, peut prononcer environ 150 mots préprogrammés. Des adaptateurs Prestel et télétexte, prévus pour le printemps 1982, ouvrent la voie au téléchargement de programmes.

Les possibilités d'extension séduisent les utilisateurs avancés. Un second processeur 6502, un Z-80 pour faire tourner CP/M, ou un processeur 16 bits National 16032 capable d'adresser 16 Mo de mémoire peuvent être connectés via l'interface Tube. Des moniteurs dédiés complètent la gamme : un modèle monochrome à 105 £ et un modèle couleur à 288 £.

La BBC commande des applications professionnelles : traitement de texte, tableur, base de données domestique, conception assistée par ordinateur. Les éditeurs tiers développent des logiciels éducatifs et ludiques. Un manuel utilisateur fouillé accompagne chaque machine, ainsi qu'une cassette de 16 programmes de démonstration.

Quelques défauts ternissent le tableau. Le boîtier plastique manque de solidité. La mémoire utilisateur de 32 Ko s'avère parfois juste, le système d'exploitation pouvant consommer jusqu'à 8 Ko. En mode graphique haute résolution, il ne reste que 2 Ko disponibles. Le mode télétexte affiche parfois des caractères erronés à cause de son jeu de caractères spécifique. Aucun câble magnétophone n'est livré avec la machine.

Ces défauts mineurs ne gâchent pas l'excellence générale du produit. La

qualité d'affichage impressionne : 80 caractères par ligne restent parfaitement lisibles sur un téléviseur noir et blanc standard grâce à une police spécialement étudiée. Le mode télétexte offre un des meilleurs rendus visuels de sa génération.

Le BBC Micro surpasse la plupart des productions japonaises et américaines de l'époque par sa richesse technique et sa modularité. Le succès commercial suit : 100 000 unités sont prévues pour la première année. Cette réussite découle de la rencontre entre l'expertise technique d'Acorn et la vision éducative de la BBC.

La machine est l'outil pédagogique de référence dans les écoles britanniques des années 1980. Elle initie toute une génération à l'informatique ainsi que la culture numérique nationale. L'expérience acquise par Acorn dans la conception de processeurs débouchera plus tard sur l'architecture ARM, présente dans la quasi-totalité des appareils mobiles modernes de la planète.

IBM PC-5150

Quand IBM annonce son Personal Computer 5150 en août 1981, l'entreprise débarque avec six ans de retard sur un marché occupé par Apple, Atari, Commodore et Radio Shack. Cette arrivée tardive force Big Blue à bouleverser ses habitudes : fini le développement entièrement interne, place aux partenariats externes. Une équipe de douze personnes seulement va concevoir cette machine en un an, temps record pour IBM.

L'approche surprend. Intel fournit le processeur, Microsoft livre PC-DOS et BASIC. La distribution aussi rompt avec les traditions : plutôt que ses propres circuits, IBM choisit Computerland et Sears. Pour la publicité, l'entreprise abandonne son image austère et mise sur le Vagabond de Charlie Chaplin dans une série de spots télévisés décalés.

À 1 565 dollars pour le modèle de base, sans lecteur de disquettes ni écran, le PC 5150 vise d'abord les particuliers. Ils peuvent brancher leur téléviseur comme moniteur et utiliser un magnétophone pour sauvegarder leurs données. Les professionnels, eux, déboursent 4 500 dollars pour une configuration complète avec 64 Ko de mémoire, deux lecteurs de disquettes, écran couleur et imprimante matricielle. Ajustés à l'inflation et convertis en francs suisses actuels, ces montants atteindraient 6 000 et 14 000 CHF.

Techniquement, rien de spectaculaire. Le processeur Intel 8088 tourne à 4,77 MHz, soit mille fois moins que nos puces actuelles. Ce dérivé écono-

mique du 8086 se contente d'un bus 8 bits pour les données en adressant jusqu'à 1 Mo de mémoire. La carte mère accueille entre 16 et 64 Ko de RAM, deux lecteurs 5,25 pouces de 160 Ko chacun.

Quelques détails marquent pourtant une rupture. Le clavier détaché fait figure d'exception à une époque où la plupart des ordinateurs l'intègrent dans le boîtier principal. Ses 83 touches pèsent 2,7 kg et produisent ce claquement caractéristique qui ravira les nostalgiques. L'unité centrale avec ses lecteurs affiche 12,7 kg sur la balance, le moniteur 7,7 kg. Total : 23 kg d'informatique domestique.

Pas de disque dur en série, il faudra attendre l'extension 5161. Pas de souris non plus, elle ne se démocratisera que plus tard. Le stockage passe par cassette audio via un connecteur spécialisé ou par disquettes. Les premières versions utilisent des supports simple face de 160 Ko, puis double face de 360 Ko pour le modèle B commercialisé dès 1983.

IBM surprend par sa transparence technique. La documentation fournie avec le PC 5150 dévoile tout : schémas électroniques, diagrammes d'architecture, spécifications d'interfaces, code source du BIOS. Cette ouverture inhabituelle dope la création de cartes d'extension et de logiciels compatibles. Mais elle ouvre aussi la porte aux clones moins chers qui prolifèreront.

Cinq slots internes acceptent les extensions via ce qu'IBM baptise « I/O Channel », ancêtre du standard ISA. Ces connecteurs 62 broches accueillent contrôleurs de disquettes, adaptateurs graphiques, interfaces parallèles pour imprimantes ou ports série RS-232 pour modems. Les PC contemporains ont abandonné ces bus parallèles pour des connexions série comme PCI Express ou USB, plus véloces.

PC-DOS gère les fichiers avec le système FAT, sans sous-répertoires dans sa première mouture. Tous les fichiers s'étalent dans une structure plate : 64 maximum sur disquette simple face, 112 sur double face. Les virus informatiques et le piratage, même s'ils existent, n'ont pas contaminé la planète. Le manuel évoque uniquement la sécurité physique : ranger ses disquettes à l'abri, sauvegarder régulièrement, contrôler l'accès à la machine. Le premier virus pour PC compatible IBM, Brain, n'apparaîtra qu'en 1986.

La production du 5150 s'arrête en 1987, remplacé par le modèle 5160. L'internationalisation démarre en 1983. Son architecture influence toujours nos machines actuelles, des objets connectés aux serveurs cloud d'entreprise. Les choix d'IBM – composants standardisés, documentation exhaustive – créent un écosystème ouvert qui dominera le marché.

MS-DOS

Jusqu'alors centré sur les gros systèmes, IBM a donc jeté son dévolu sur le marché des ordinateurs personnels en 1980. La firme d'Armonk cherche un système d'exploitation pour sa future machine. Elle se tourne vers Microsoft, une petite société que Bill Gates et Paul Allen ont créée quelques années plus tôt. Problème : Microsoft ne possède aucun système d'exploitation digne de ce nom.

La solution vient de Seattle Computer Products. Cette entreprise a développé QDOS – Quick and Dirty Operating System – sous la houlette de Tim Paterson. L'histoire de QDOS mérite qu'on s'y arrête. En mai 1979, Seattle Computer Products conçoit une carte processeur 8086 pour le bus S-100. L'entreprise compte sur Digital Research pour livrer CP/M-86, mais les promesses s'éternisent. Lassés d'attendre, les ingénieurs décident de créer leur propre système. Paterson s'attelle à la tâche en avril 1980. Deux mois plus tard, QDOS voit le jour, et le système fonctionne remarquablement bien pour un développement si rapide.

Microsoft flaire la bonne affaire. La société fait l'acquisition d'une licence non-exclusive de QDOS pour 25 000 dollars en décembre 1980. Quelques mois plus tard, elle rachète l'intégralité des droits pour 50 000 dollars supplémentaires. Tim Paterson rejoint l'équipe de Microsoft pour transformer QDOS en MS-DOS.

Le premier IBM PC sort en 1981, équipé de MS-DOS 1.0. Ce système d'exploitation rudimentaire fonctionne entièrement en mode texte. L'utilisateur tape des commandes : DIR pour lister les fichiers, COPY pour les dupliquer, ERASE pour les supprimer. Pas de répertoires, juste des disquettes de 160 Ko. Austère, certes, mais fonctionnel.

Les améliorations arrivent. MS-DOS 2.0 débarque en 1983 avec le support des disques durs de 10 Mo et une structure de fichiers hiérarchique. Cette version emprunte à UNIX quelques idées, notamment un processeur de commandes par lots plus sophistiqué. MS-DOS 3.0 suit en 1984, taillé sur mesure pour l'IBM PC AT. Le système de fichiers FAT16 fait son apparition, gérant des partitions jusqu'à 32 Mo.

L'architecture technique repose sur trois composants principaux. Le fichier caché MSDOS.SYS contient le cœur du système et traite les requêtes de haut niveau des programmes. IO.SYS s'occupe des aspects matériels, cette couche d'abstraction entre le logiciel et le hardware. COMMAND.COM fournit l'interface utilisateur, cette invite de commande que des millions d'utilisateurs apprendront à connaître.

Le système de fichiers FAT révolutionne la gestion de l'espace disque. Une table d'allocation suit l'usage de chaque unité, composée de plusieurs

secteurs consécutifs. Cette approche évite la fragmentation externe mais génère une fragmentation interne : la dernière unité d'un fichier n'est jamais complètement remplie. Un compromis technique qui fera ses preuves.

Microsoft joue finement sa partition commerciale. L'accord avec IBM autorise la vente de MS-DOS à d'autres fabricants. Cette stratégie établit MS-DOS comme standard de facto pour tous les PC compatibles IBM. Quand Windows 3.0 débarque en 1990, l'interface graphique s'appuie sur MS-DOS. Le succès est au rendez-vous.

MS-DOS 6.0 arrive en 1993 avec son lot de nouveautés : DoubleSpace pour compresser les disques, un antivirus intégré, un système de sauvegarde. Mais DoubleSpace pose problème. Des corruptions de données incitent Microsoft à publier la version 6.2, complétée de ScanDisk pour réparer les disques endommagés. L'année ne se termine pas en beauté : Stac Electronics attaque Microsoft pour violation de brevet sur l'algorithme de compression. Microsoft perd et doit retirer DoubleSpace. La version 6.22 sort en 1994 avec DriveSpace, utilisant un nouvel algorithme pour éviter tout litige.

Windows 95 sonne le glas de l'âge d'or de MS-DOS. Le système intègre MS-DOS 7.0 comme simple composant de démarrage et de compatibilité. Windows 98 et Me gardent encore MS-DOS, mais son influence s'amenuise. Windows XP, en 2001, ferme définitivement le chapitre, ne conservant qu'une compatibilité minimale pour faire tourner d'anciennes applications récalcitrantes.

Le système MS-DOS standardise l'univers PC en accompagnant l'éclosion d'un écosystème logiciel d'une richesse inouïe. Des traitements de texte aux tableurs, des bases de données aux jeux, tout un monde d'applications naît. Des entreprises comme Lotus, Borland ou WordPerfect bâtissent leur fortune sur cette plateforme.

MS-DOS démocratise l'informatique à sa manière. Son interface en ligne de commande, austère et exigeante, forme toute une génération d'utilisateurs. Apprendre MS-DOS, c'est comprendre la logique interne de la machine. Cette école de rigueur forge des compétences qui se révéleront précieuses avec l'arrivée des interfaces graphiques, car l'invite de commande Windows conserve nombre de commandes héritées de MS-DOS : DIR, COPY, DEL fonctionnent encore. Les concepts de gestion des fichiers, d'allocation mémoire et d'exécution des programmes établis à cette époque constituent toujours les fondements des systèmes actuels. DOSBox et d'autres émulateurs préservent cette histoire, permettant aux nostalgiques de retrouver leurs anciens logiciels.

MS-DOS est un jalon important de l'informatique personnelle. Ce système humble et efficace a rendu possible l'explosion du PC. Sans lui (et

QDOS), l'histoire de Microsoft aurait pris un cours différent, et l'informatique grand public aurait peut-être emprunté d'autres chemins.

Osborne 1

Adam Osborne avait compris quelque chose que peu d'entrepreneurs saisissaient en 1980. Auteur d'ouvrages informatiques et patron d'une maison d'édition spécialisée, il observait un marché en pleine ébullition où les micro-ordinateurs restaient cantonnés aux bureaux et aux laboratoires. Son idée paraissait simple : créer un ordinateur que l'on puisse emporter partout.

En avril 1981, dans les colonnes de *Creative Computing Magazine*, Osborne exposait sa vision avec une lucidité remarquable. Il identifiait cinq besoins fondamentaux que devait satisfaire un ordinateur personnel : un prix abordable, une simplicité d'usage sans sacrifier la polyvalence, la capacité de transport, une compatibilité logicielle grâce à des systèmes standardisés, et une adaptabilité aux exigences des développeurs. Cette analyse découlait directement de son expérience dans l'industrie et de sa frustration face aux limitations des machines existantes.

L'Osborne 1 matérialisait cette réflexion. Avec ses 10,9 kg, la machine adoptait un design tout-en-un audacieux : écran de 5 pouces, deux lecteurs de disquettes, clavier détachable et alimentation intégrés dans un boîtier unique transportable. Cette approche tranchait radicalement avec les systèmes traditionnels qui nécessitaient de multiples composants reliés par un enchevêtrement de câbles.

Techniquement, l'ordinateur se montrait cohérent. Son processeur Zilog Z80A cadencé à 4 MHz, ses 64 Ko de RAM et son système d'exploitation CP/M offraient des performances correctes pour l'époque. L'écran affichait 52 caractères sur 24 lignes, avec possibilité de défilement horizontal jusqu'à 128 caractères. Chaque lecteur de disquettes 5,25 pouces proposait 102 Ko de stockage. Cette configuration représentait en 1981 un compromis judicieux entre performances et mobilité.

Osborne avait surtout compris l'importance du logiciel. Pour 1 795 dollars, l'acheteur recevait la machine accompagnée d'une suite complète incluant WordStar pour le traitement de texte, SuperCalc comme tableur, ainsi que BASIC et CBASIC pour la programmation. Cette stratégie de regroupement logiciel était inédite et transformait l'achat d'un simple matériel en acquisition d'un environnement de travail complet.

La cible commerciale s'articulait autour de trois segments distincts. Les

professionnels itinérants y trouvaient un moyen de gérer leurs documents et tableurs en déplacement, avant de synchroniser leur travail au bureau via modem. Le secteur éducatif découvrait une solution économique pour équiper les établissements en matériel informatique. Les particuliers accédaient enfin à une alternative abordable aux ordinateurs de bureau traditionnels.

Le succès dépassa toutes les prévisions. La société Osborne Computer Corporation connut une croissance fulgurante, devenant l'une des entreprises à la progression la plus rapide de l'histoire américaine. Les ventes atteignirent 10 000 unités mensuelles, générant plus de 70 millions de dollars de chiffre d'affaires dès la première année.

Les défauts de la machine se révélèrent. L'écran 5 pouces, constamment critiqué, contraignait les utilisateurs à un défilement horizontal permanent pour consulter des documents standard. Le stockage sur disquette se révélait insuffisant pour certaines applications professionnelles. Osborne réagit en développant une option double densité portant la capacité à 200 Ko par disque.

En 1982, l'entreprise annonça une amélioration majeure : le SCREENPAC, option d'affichage 80/104 colonnes. Cette mise à niveau matérielle permettait d'afficher jusqu'à 104 caractères par ligne, rapprochant l'Osborne 1 des standards industriels. L'annonce prématurée de cette amélioration non disponible provoqua cependant un effet pervers : les ventes du modèle existant chutèrent brutalement. Ce phénomène, baptisé plus tard « effet Osborne », illustrait les dangers d'une communication produit mal maîtrisée.

L'influence de l'Osborne 1 sur l'industrie informatique dépassa largement son succès commercial. Il établit plusieurs standards durables : l'intégration complète des composants, l'inclusion systématique de logiciels avec le matériel, et un positionnement prix accessible. Son impact se manifesta directement dans le développement des premiers portables d'IBM, Compaq et autres constructeurs qui dominèrent ensuite le marché.

La machine introduisait des innovations techniques remarquables. Son clavier utilisait une technologie *full travel membrane* offrant une qualité de frappe excellente tout en résistant aux infiltrations de poussière et de liquides. Sa conception modulaire simplifiait les réparations. L'adoption du bus S-100 et du système CP/M garantissait une compatibilité avec un vaste écosystème logiciel existant.

La robustesse de l'appareil impressionnait. Des tests documentés dans *The Portable Companion* rapportaient sa survie à des chutes dans des escaliers et à des conditions d'utilisation extrêmes. Cette solidité, couplée à sa portabilité, en faisait un outil parfaitement adapté aux professionnels mobiles.

L'Osborne 1 transforma l'organisation du travail en entreprise. Il permit l'émergence du concept de « bureau mobile », autorisant les employés à emporter leur environnement numérique en déplacement. Cette évolution annonçait les mutations que connaîtrait l'organisation du travail avec la généralisation de l'informatique portable.

La chute fut aussi brutale que l'ascension avait été fulgurante. En septembre 1983, Osborne Computer Corporation déposait le bilan, victime de la concurrence accrue, de difficultés de production et d'erreurs stratégiques. L'entreprise avait sous-estimé la vitesse d'évolution du marché et la nécessité d'innovation constante. En deux ans, l'Osborne 1 passa du statut d'innovation révolutionnaire à celui de technologie dépassée. Son impact sur la démocratisation de l'informatique portable et l'établissement de standards industriels est néanmoins indéniable.

L'expérience Osborne rappelle l'importance de la portabilité, la nécessité d'un écosystème logiciel complet, et l'équilibre délicat entre innovation technologique et attentes du marché. L'Osborne 1 a écrit malgré tout un chapitre incontournable de l'histoire de l'informatique personnelle.

LZW

L'algorithme de compression LZW raconte une histoire singulière, celle d'une découverte théorique qui a mis des années avant de trouver sa forme pratique. Tout commence en 1977, quand Abraham Lempel et Jacob Ziv publient dans l'IEEE Transactions on Information Theory un article aux allures hermétiques intitulé « A Universal Algorithm for Sequential Data Compression ». Leur travail expose une méthode utilisant des dictionnaires de phrases qui glissent sur un texte déjà lu, construisant progressivement de nouveaux motifs en ajoutant des symboles aux séquences existantes.

L'année suivante, les deux chercheurs récidivaient dans la même revue avec une version affinée de leur algorithme. Ces publications restaient néanmoins très abstraites, posant certes les fondations théoriques mais laissant le lecteur sur sa faim quant aux applications concrètes. Il fallut attendre 1984 pour qu'un troisième larron entre en scène et transforme ces travaux académiques en une technologie utilisable.

Terry Welch travaillait au Sperry Research Center quand il publia dans Computer magazine son article « A technique for High-Performance Data Compression ». Son apport révolutionna pourtant l'approche de Lempel et Ziv. Welch eut l'idée de pré-remplir le dictionnaire avec les 256 caractères

ASCII standards plutôt que de partir d'un dictionnaire vide. Cette modification permit de traiter immédiatement tous les caractères simples, y compris lors de leur première apparition. Les codes restants, de 256 à 4095 dans un dictionnaire sur 12 bits, pouvaient ainsi servir exclusivement aux nouvelles séquences découvertes.

L'algorithme LZW était né, portant les initiales de ses trois créateurs. Sa force résidait dans sa capacité à dénicher et éliminer les redondances tout en garantissant une restitution parfaite des données originales. Contrairement à d'autres méthodes, LZW n'envoyait jamais les caractères bruts mais uniquement des codes, ce qui généraient des gains substantiels sur les fichiers contenant des motifs répétitifs.

Une caractéristique remarquable de cet algorithme tenait à son autonomie lors de la décompression. Le décodeur n'avait pas besoin de recevoir le dictionnaire utilisé pour la compression ; il pouvait le reconstituer à l'identique en appliquant les mêmes règles que le compresseur. Cette propriété rendait LZW économe en mémoire et rapide à l'exécution.

L'industrie informatique s'empara de cette innovation. En 1987, CompuServe lança le format GIF (Graphics Interchange Format), qui exploitait LZW pour compresser les images. Ce format devint vite incontournable sur les premiers réseaux puis sur Internet, notamment grâce à sa capacité à stocker de nombreuses images dans un fichier unique et à gérer la transparence. UNIX intégra aussi l'algorithme dans sa commande `compress`, tandis que les formats TIFF et PDF l'adoptaient pour leurs besoins spécifiques.

Les performances de LZW variaient considérablement selon le type de données traitées. Les textes en langue naturelle, riches en redondances, lui convenaient parfaitement, tout comme les images contenant de vastes zones uniformes. Sur ces contenus, l'algorithme atteignait couramment des taux de compression de 50% ou plus. En revanche, les fichiers binaires donnaient des résultats plus imprévisibles : certains se compressaient mieux que du texte ordinaire, d'autres résistaient obstinément à tout traitement.

L'implémentation pratique de LZW nécessitait de résoudre quelques casse-têtes techniques. Le plus délicat survenait lors de la décompression, quand le décodeur rencontrait un code avant d'avoir pu le définir. Cette situation baroque se produisait avec des séquences du type (chaîne, caractère, chaîne, caractère, chaîne), où l'algorithme émettait un code que le décompresseur n'avait pas encore eu l'occasion de créer. Il fallait donc prévoir un mécanisme d'exception spécifique pour gérer ces cas particuliers.

La belle histoire de LZW connut un tournant inattendu dans les années 1990. Unisys, détenteur d'un brevet sur l'algorithme, commença à réclamer des redevances pour toute utilisation commerciale. Cette revendication soudaine provoqua un tollé dans la communauté informatique et poussa au

développement du format PNG (Portable Network Graphics) comme alternative libre au GIF. Le brevet finit par expirer en 2003 aux États-Unis et en 2004 ailleurs, libérant enfin l'algorithme de ces contraintes juridiques.

La taille du dictionnaire influençait directement l'efficacité de la compression. Un dictionnaire de 12 bits, capable de stocker 4096 entrées, représentait souvent un équilibre satisfaisant entre taux de compression et consommation mémoire. L'algorithme gagnait en performance avec des fichiers volumineux, car le dictionnaire avait davantage d'occasions de repérer et réutiliser des motifs récurrents.

Sur les petits fichiers, LZW montrait ses faiblesses. Le coût de stockage du dictionnaire pouvait dépasser le bénéfice de la compression, rendant le fichier compressé plus volumineux que l'original. Des variantes comme LZWS (Lempel-Ziv-Welch-Setia) tentèrent d'adapter dynamiquement la stratégie selon la taille des données, avec un succès mitigé.

L'algorithme LZW établit des principes durables dans le domaine de la compression : l'usage de dictionnaires adaptatifs, la compression sans perte, l'importance de l'autonomie lors de la décompression. Il occupe une place particulière dans l'écosystème informatique. Des méthodes plus récentes offrent certes de meilleurs taux de compression, mais sa simplicité relative en fait un excellent outil pédagogique pour comprendre les mécanismes fondamentaux de la compression de données. Il reste utilisé dans certaines applications spécifiques où sa robustesse et sa rapidité d'exécution compensent ses limitations. Dans l'histoire des technologies de l'information, LZW demeure l'un des premiers algorithmes de compression à avoir connu un succès commercial durable et une adoption véritablement universelle.

AutoCAD

En 1982, la conception assistée par ordinateur est l'apanage de quelques privilégiés. Les systèmes de CAO coûtent une fortune – plusieurs centaines de milliers de dollars – et nécessitent des machines spécialisées que seules les grandes entreprises peuvent s'offrir. Une poignée de programmeurs décide de changer la donne.

John Walker réunit autour de lui une quinzaine de développeurs, la plupart issus d'Information Systems Design, une société travaillant sur les systèmes UNIVAC. Ils créent Autodesk avec une idée simple mais audacieuse : développer des logiciels de CAO pour les micro-ordinateurs personnels qui commencent à envahir les bureaux. L'organisation qu'ils mettent

en place détonne pour l'époque : chacun travaille depuis son domicile, communiquant par téléphone et courrier. Avec 59 000 dollars d'investissement initial, ils lancent une poignée de projets en parallèle, convaincus que l'un d'eux finira par percer.

Le déclic vient de Mike Riddle et de son programme Interact, écrit dans un langage de programmation qu'il a lui-même conçu. L'équipe d'Autodesk rachète le code source et entreprend de le réécrire entièrement en C. Dan Drake et Greg Lutz se chargent de cette conversion délicate, transformant progressivement ce logiciel de dessin en ce qui deviendra AutoCAD. Le choix du langage C, encore peu répandu dans le monde commercial, se révèle judicieux pour la portabilité future du logiciel.

Fin 1982, au salon Comdex de Las Vegas, AutoCAD fait ses premiers pas publics. Le logiciel tourne sur un IBM PC équipé de 64 Ko de mémoire et de deux lecteurs de disquettes. Pour frapper les esprits, l'équipe organise une démonstration avec une table traçante HP empruntée pour l'occasion. Le prix annoncé fait sensation : 1 000 dollars, certes considérable pour un logiciel PC, mais dérisoire comparé aux solutions existantes.

Mike Ford élabore une stratégie commerciale originale qui repose sur un réseau de revendeurs formés. Ces derniers assurent le support technique de première ligne, libérant ainsi Autodesk des contraintes du service client direct. Cette approche décentralisée correspond bien à l'ADN de l'entreprise et à sa philosophie du travail à distance.

AutoCAD séduit sa clientèle cible : les petits cabinets d'architecture et d'ingénierie. Sa capacité à fonctionner sur différentes plateformes matérielles et à échanger des fichiers de manière transparente représente un atout majeur dans un monde informatique très cloisonné. Les utilisateurs apprécient cette souplesse qui leur évite d'être prisonniers d'un constructeur particulier.

L'équipe technique maintient un rythme effréné d'innovations. Le support du coprocesseur mathématique 8087 d'Intel accélère considérablement les calculs. L'intégration d'un langage LISP offre aux utilisateurs avancés la possibilité d'étendre les fonctionnalités du logiciel selon leurs besoins spécifiques. Les outils de cotation deviennent de plus en plus sophistiqués. Chaque semaine, lors des réunions techniques collégiales, l'équipe tranche sur les orientations futures du produit.

Trois ans après sa création, Autodesk entre en bourse. La société n'a jamais fait appel au capital-risque, se développant uniquement grâce à ses revenus propres. Business Week la classe deux années consécutives comme l'entreprise à plus forte croissance. John Walker, peu attiré par les aspects managériaux, transmet les rênes à Al Green, puis à Carol Bartz en 1992.

L'arrivée de Windows 3.0 en 1990 bouleverse la donne technique. Jus-

qu'alors, AutoCAD fonctionnait principalement sous DOS et était porté sur de nombreuses variantes UNIX. La direction prend le parti de concentrer ses efforts sur Windows, anticipant la domination future de ce système. Cette décision coïncide avec l'émergence de la modélisation 3D comme fonctionnalité incontournable, développée sous l'impulsion de Scott Heath.

AutoCAD a réussi là où d'autres ont échoué grâce à différents ingrédients : une équipe technique exceptionnelle obsédée par la qualité et la compatibilité, une stratégie commerciale efficace via les revendeurs, un positionnement tarifaire accessible, et une capacité remarquable à suivre l'évolution rapide du matériel informatique. En démocratisant la CAO et en la rendant accessible aux petites structures, le logiciel a transformé les pratiques de conception dans l'industrie et l'architecture.

CD-ROM

Au tournant des années 1980, l'informatique personnelle vivait à l'heure des disquettes. Ces petits carrés de plastique, fragiles et limités à quelques centaines de kilo-octets, constituaient le principal moyen d'échanger programmes et données. L'idée qu'un seul disque puisse contenir l'équivalent de plusieurs centaines de disquettes relevait de la science-fiction. Pourtant, cette révolution se préparait déjà dans les laboratoires de Philips.

L'histoire commence par les recherches sur le stockage optique menées par l'entreprise néerlandaise durant les années 1970. Les ingénieurs de Philips travaillaient sur le système VLP, une technologie qui gravait des signaux vidéo analogiques sur un disque en plastique lu par laser. Ces expérimentations, d'abord destinées à remplacer les cassettes vidéo, allaient ouvrir une voie totalement inattendue. Car de ces travaux naîtrait d'abord le Compact Disc audio, puis son descendant informatique : le CD-ROM.

La collaboration entre Philips et Sony débouche en 1983 sur l'annonce du CD-ROM, acronyme de Compact Disc Read-Only Memory. Les deux géants de l'électronique définissent ensemble les spécifications techniques dans le fameux « Yellow Book », document de référence qui établit les règles du nouveau format. Le concept reprend les dimensions physiques du CD audio : un disque de polycarbonate de 12 cm de diamètre sur 1,2 mm d'épaisseur, parcouru par un laser infrarouge qui déchiffre les informations gravées sous forme de microsillons.

Mais là où le CD audio stocke de la musique, le CD-ROM organise les données différemment. Il propose deux modes de fonctionnement. Le Mode

1 privilégie la fiabilité pour les données informatiques grâce à un système de correction d'erreurs renforcé. Le Mode 2 se destine aux contenus audio et vidéo compressés, acceptant quelques erreurs mineures imperceptibles à l'oreille ou à l'œil humain.

La capacité de stockage révolutionne les habitudes : 650 Mo d'un coup, soit l'équivalent de 450 disquettes standard. Avec cette abondance subite, les éditeurs de logiciels peuvent enfin concevoir des applications ambitieuses sans se soucier des contraintes de place. Fini le temps où il fallait rogner sur les fonctionnalités pour tenir sur quelques disquettes.

Les premiers lecteurs de CD-ROM arrivent sur le marché en 1985. Ces machines coûteuses et relativement poussives affichent une vitesse de transfert de 150 Ko/s, baptisée « simple vitesse » ou « 1X » dans le jargon technique. L'adoption est d'abord confidentielle, réservée aux professionnels et aux passionnés fortunés. Il faudra attendre le début des années 1990 pour que la technologie trouve vraiment son public.

L'essor du multimédia change tout. Les développeurs de jeux vidéo et d'applications éducatives découvrent les possibilités offertes par cette nouvelle capacité de stockage. Là où Windows 95 nécessitait pas moins de 13 disquettes dans sa version de base, la version CD-ROM tient sur un seul disque et simplifie radicalement l'installation. Plus besoin de jongler avec une pile de disquettes, de surveiller l'ordre d'insertion, de redémarrer la procédure quand l'une d'elles se révèle défectueuse.

L'industrie du jeu vidéo saisit l'opportunité. Le CD-ROM favorise l'intégration de séquences vidéo de qualité cinématographique, de la musique orchestrale et de graphismes détaillés jusqu'alors impensables. Sony construit le succès de sa PlayStation sur cette technologie, abandonnant les cartouches coûteuses pour miser sur le CD-ROM. Les jeux prennent une ampleur nouvelle, les studios peuvent raconter des histoires complexes avec des moyens de production décuplés.

La course à la vitesse s'engage. Les constructeurs rivalisent d'ingéniosité pour accélérer la lecture des données. De 1X, on passe à 2X, 4X, 8X et bien au-delà, jusqu'à atteindre 52X au début des années 2000. Cette montée en puissance s'accompagne d'une chute spectaculaire des prix. Les lecteurs, d'abord vendus plusieurs milliers de francs, sont maintenant accessibles au grand public. Les disques vierges suivent une trajectoire identique, transformant le CD-ROM en support d'archivage abordable.

L'évolution technique ne s'arrête pas là. En 1988 apparaît le CD-R, permettant enfin aux utilisateurs de graver leurs propres disques. Cette innovation démocratise la production de CD personnalisés, ouvrant la voie aux sauvegardes individuelles et à l'échange de données entre particuliers. Le support de stockage CD-RW, introduit en 1997, franchit un pas supplé-

mentaire vers la polyvalence en autorisant l'effacement et la réécriture des données.

La standardisation est la clé de ce succès. Les spécifications techniques font l'objet de normes internationales strictes, l'ISO 9660 définissant notamment le système de fichiers universel. Cette approche, héritée du format « High Sierra » de 1985, garantit qu'un CD-ROM gravé sur une machine pourra être lu sur n'importe quelle autre, quel que soit le fabricant du lecteur.

Le CD-ROM engendre une famille nombreuse de formats dérivés. Le CD-i vise les applications multimédias interactives, le Video CD s'attaque au marché de la vidéo domestique, le Photo CD révolutionne le stockage d'images numériques. Ces déclinaisons rencontrent des fortunes diverses, mais témoignent de la richesse conceptuelle du support originel.

Le CD-ROM rend possible la distribution d'encyclopédies multimédias comme Encarta, de collections documentaires exhaustives, de bases de données professionnelles volumineuses. Il contribue directement à l'explosion du multimédia dans les années 1990, période où l'ordinateur personnel cesse d'être une simple machine à traitement de texte pour devenir un centre de divertissement familial.

Cette hégémonie commence à vaciller au début des années 2000. Le DVD, avec sa capacité de stockage supérieure, relègue progressivement le CD-ROM au second plan. L'arrivée d'Internet à haut débit et la généralisation des clés USB accélèrent le processus. La dématérialisation des logiciels, téléchargés directement depuis les serveurs des éditeurs, porte un coup supplémentaire au support physique.

Intel 80286

Le 1er février 1982 marque l'entrée en scène de l'Intel 80286, baptisé iAPX 286 dans la terminologie officielle d'Intel. Cette puce de 134 000 transistors s'impose comme un tournant dans l'évolution des microprocesseurs 16 bits. Son destin se lie intimement à celui de l'IBM PC/AT dès 1984, puis à l'ensemble de l'écosystème des compatibles PC/AT qui domineront le paysage informatique jusqu'aux premières années 1990.

L'histoire des performances du 80286 raconte celle d'une montée en puissance progressive. Intel lance ses premiers modèles à 6 MHz et 8 MHz, avant de proposer des versions plus véloces culminant à 12,5 MHz. AMD et Harris pousseront plus tard cette architecture vers des sommets inattendus :

20 MHz pour le premier, 25 MHz pour le second. Ces chiffres masquent une réalité plus nuancée : le processeur n'exécute en moyenne que 0,21 instructions par cycle d'horloge. Traduite en puissance brute, cette cadence donne 0,9 MIPS pour le modèle 6 MHz, 1,5 MIPS à 10 MHz, et 1,8 MIPS à 12 MHz.

Sous le capot, Intel a repensé l'architecture de fond en comble. Le 80286 double pratiquement les performances par cycle par rapport à ses aînés 8086 et 8088. Cette prouesse technique ne relève pas de la magie : elle résulte d'optimisations ciblées, comme le traitement des modes d'adressage complexes base+index par un circuit dédié plutôt que par l'unité arithmétique générale. Les calculs mathématiques exigeants, multiplication et division en tête, réclament désormais moins de cycles.

Le bond le plus spectaculaire concerne l'adressage mémoire. Avec ses 24 bits d'adresses, le 80286 gère théoriquement jusqu'à 16 Mo de RAM, pulvérisant la barrière du mégaoctet qui bridait le 8086. Cette capacité est toutefois largement théorique dans la pratique quotidienne. Le prix prohibitif de la mémoire vive et la rareté des logiciels compatibles cantonnent la plupart des machines à un seul mégaoctet. L'accès à cette mémoire étendue depuis le mode réel traditionnel impose par ailleurs une pénalité de performance non négligeable.

L'apport majeur du 80286 réside dans son mode protégé. Cette innovation le hisse au niveau des processeurs professionnels de l'époque. Au-delà de l'exploitation des 16 Mo de mémoire physique grâce à son unité de gestion mémoire intégrée, ce mode ouvre un espace d'adressage logique d'un gigaoctet. La fin des plantages causés par des applications indisciplinées qui écrivent n'importe où est arrivée. La protection mémoire est désormais réelle. Le système organise la mémoire en segments distincts pour les données, le code et la pile, avec une hiérarchie de privilèges qui interdit aux programmes de bas niveau d'interférer avec ceux de niveau supérieur.

Cette belle mécanique cache néanmoins un défaut rédhibitoire. Une fois basculé en mode protégé, le 80286 ne peut revenir au mode réel compatible 8086 qu'au prix d'une réinitialisation complète. IBM déploie des trésors d'ingéniosité dans le PC/AT pour contourner cette limitation : circuits externes, code spécialisé dans le BIOS ROM, séquence d'instructions alambiquée qui provoque la réinitialisation tout en préservant mémoire et contrôle. La solution fonctionne, mais la performance en pâtit lourdement.

Ces contraintes techniques influencent directement l'écosystème logiciel. En janvier 1985, Digital Research s'associe à Intel pour présenter Concurrent DOS 286, un système d'exploitation pensé pour exploiter nativement le mode protégé dans un environnement multi-utilisateur et multitâche. Le projet se heurte aux réalités du terrain : l'émulation du 8086 sur les puces de

production révèle des dysfonctionnements. Intel réagit en corrigeant les erreurs documentées dans le pas E-1, et en modifiant carrément le microcode dans le pas E-2 pour accélérer l'émulation. Ces améliorations permettent à IBM d'adopter DR Concurrent DOS 286 comme fondation de son système IBM 4680 OS en 1986, destiné aux produits IBM Plant System et aux terminaux de vente.

Les critiques pleuvent sur le 80286. Bill Gates, jamais avare en formules choc, le qualifie de « puce au cerveau mort », anticipant l'impossibilité pour Windows d'exécuter plusieurs applications MS-DOS en parallèle. Cette position accentue la rupture entre Microsoft et IBM, cette dernière s'obstinant à développer OS/2 pour le 286 en mode texte, projet initialement commun aux deux géants.

Malgré ses imperfections criantes, le 80286 démocratise des mécanismes de protection mémoire jusque-là réservés aux ordinateurs centraux et mini-ordinateurs. Là où ses concurrents NS320xx et M68000 nécessitent des composants externes pour gérer la MMU, le 80286 intègre ces fonctions directement sur sa puce. Cette intégration, couplée aux gains de performance substantiels, propulse l'architecture x86 et les PC IBM depuis l'entrée de gamme jusqu'aux stations de travail et serveurs haut de gamme.

Le processeur enrichit aussi les capacités de calcul en gérant différents types de nombres : décimal condensé non signé, binaire non signé, décimal non condensé non signé, binaire signé et virgule flottante. Ces fonctionnalités, alliées à sa capacité multitâche, l'orientent vers les applications de communication, le contrôle de processus temps réel et les systèmes multi-utilisateurs.

L'épopée du 80286 illustre parfaitement la complexité inhérente au développement des microprocesseurs, où chaque choix architectural résonne sur des décennies d'évolution logicielle. Son empreinte perdure à travers les générations suivantes de processeurs x86, qui conservent ses concepts fondamentaux tout en gommant ses défauts de jeunesse.

SMTP

En août 1982, l'IETF publie la RFC 821. Cette spécification technique, qui pourrait paraître anodine dans le flux constant des documents normatifs, va transformer la communication électronique mondiale. Elle donne naissance au Simple Mail Transfer Protocol, l'architecture invisible qui transporte aujourd'hui des milliards de messages quotidiens.

L'idée paraît simple : créer un langage commun entre machines pour échanger des courriers électroniques. Avant SMTP, chaque système de messagerie parlait son propre dialecte. Un message envoyé depuis un serveur UNIX ne pouvait pas atteindre un destinataire sur une machine Windows. L'interconnexion des réseaux informatiques exigeait un *esperanto* numérique.

La RFC 821 établit les règles du jeu. Un client SMTP entre en conversation avec un serveur distant, s'identifie, précise l'expéditeur et les destinataires, puis transmet le contenu. Chaque étape génère des codes de réponse numériques standardisés. Cette chorégraphie technique, répétée des millions de fois chaque seconde à travers la planète, orchestre l'échange mondial des messages.

Le protocole traverse les années 1980 et 1990 sans modification majeure. Sa robustesse impressionne car pendant près de vingt ans, la même spécification technique supporte l'explosion d'Internet. Mais cette longévité révèle aussi ses failles. Les pirates découvrent comment exploiter les relais ouverts pour inonder le réseau de courriers indésirables. Le spam naît et prolifère.

En avril 2001 arrive la RFC 2821. L'IETF introduit ESMTP (Extended SMTP), une version enrichie du protocole original. Les nouveautés changent la donne : notifications de réception automatiques, messages d'erreur détaillés, support de caractères non standard dans les en-têtes. Surtout, ESMTP autorise le chiffrement des communications entre serveurs. La sécurité entre enfin dans l'équation.

Cette évolution préserve cependant un principe cardinal : la rétrocompatibilité. Tous les serveurs ESMTP acceptent les connexions SMTP classiques. Cette stratégie assure une transition progressive, évitant le chaos d'une rupture brutale avec l'existant.

La RFC 5321, publiée en octobre 2008, consolide ces acquis. Elle affine les règles d'utilisation du TLS pour sécuriser les échanges et clarifie certains aspects techniques ambigus des versions précédentes. Mais le protocole affronte désormais des défis autrement complexes que ses spécifications originales.

Le spam transforme Internet en champ de bataille. Des automates malveillants testent systématiquement des combinaisons d'adresses pour collecter des cibles valides. Les attaques de type « Directory Harvest Attack » automatisent cette prospection. Les logiciels malveillants se propagent via les pièces jointes. Les techniques de « spoofing » permettent de falsifier l'origine des messages, semant la confusion sur l'identité réelle des expéditeurs.

Face à cette escalade, l'écosystème technique réagit. Avec le framework SPF, les domaines déclarent officiellement quels serveurs peuvent envoyer des e-mails en leur nom. DKIM ajoute des signatures cryptographiques aux

messages pour authentifier leur origine. Les filtres anti-spam analysent le contenu, consultent des listes noires, appliquent des heuristiques sophistiquées pour séparer le bon grain de l'ivraie.

Pourtant, le protocole SMTP porte en lui certaines vulnérabilités structurelles. Il ne vérifie pas nativement si l'adresse de l'expéditeur correspond à son domaine déclaré. Les notifications automatiques peuvent confirmer l'existence d'adresses e-mail aux spammeurs. Conçu initialement pour enrichir les messages avec différents types de contenu, le format MIME sert aussi à dissimuler des contenus malveillants. Une nouvelle technologie à double usage.

Malgré ces faiblesses, SMTP domine toujours la messagerie électronique mondiale. Sa pérennité tient d'abord à sa simplicité conceptuelle : les développeurs comprennent très vite son fonctionnement. Ensuite, sa flexibilité : le protocole accepte des extensions sans rompre avec les implémentations existantes. Enfin, l'inertie de l'immense base installée de serveurs et clients qui l'utilisent.

Cette histoire technique révèle une vérité plus large sur l'évolution des standards informatiques. Un protocole survit rarement par ses qualités intrinsèques seules. Il doit s'adapter aux nouveaux usages tout en intégrant les contraintes de sécurité émergentes et en préservant la compatibilité avec son écosystème. SMTP illustre cet équilibre délicat entre innovation et continuité.

Les préoccupations actuelles portent sur le chiffrement systématique des communications, l'authentification renforcée des expéditeurs, l'adaptation à la messagerie mobile. Le protocole continue d'évoluer, quarante ans après sa naissance. Cette longévité exceptionnelle dans l'univers informatique témoigne de sa conception initiale robuste et de sa capacité d'adaptation continue.

L'architecture complexe qui entoure désormais SMTP – filtres anti-spam, systèmes d'authentification, protocoles de sécurité – forme un écosystème technique sophistiqué. Cette complexité croissante contraste avec la simplicité originelle du protocole.

Commodore 64

En 1982, Jack Tramiel lance un pari audacieux avec le Commodore 64. Connu pour sa philosophie « les ordinateurs pour les masses, pas les classes », l'homme d'affaires fixe le prix à 595 dollars : moitié moins que la

concurrence. Cette machine tire son nom de ses 64 ko de mémoire vive, une quantité impressionnante qui laisse présager des possibilités inédites.

Le secret de cette performance économique réside dans une décision prise six ans plus tôt. En rachetant MOS Technology en 1976, Commodore s'assure la maîtrise de ses composants : le processeur MOS 6510, dérivé du célèbre 6502, mais aussi deux puces novatrices. Le VIC-II gère l'affichage avec ses 16 couleurs, sa résolution de 320x200 pixels et surtout ses huit sprites, ces petits objets graphiques qui donneront vie aux jeux d'arcade. Quant au SID, ce synthétiseur intégré propose trois voix de synthèse avec un contrôle précis des enveloppes sonores ADSR. Peu d'ordinateurs de l'époque rivalisent avec de telles capacités audio.

L'architecture du C64 révèle une ingéniosité particulière. Le système jongle entre ROM et RAM selon les besoins, libérant de l'espace mémoire quand nécessaire. Le jeu de caractères PETSCII inclut des symboles semi-graphiques qui transforment l'écran en véritable toile de création. Un programmeur débutant a la possibilité de dessiner des interfaces rudimentaires sans connaître une ligne de code machine. Le BASIC intégré et le système KERNAL offrent cette accessibilité recherchée par Tramiel, sans brider les experts qui explorent les recoins les plus techniques de la bête.

L'innovation commerciale accompagne l'innovation technique. Commodore abandonne les magasins spécialisés pour investir les grandes surfaces comme K-Mart. Cette stratégie fracassante porte le C64 dans les foyers américains, puis européens. Les parents découvrent un ordinateur « familial » à prix abordable. Les enfants y trouvent une console de jeu sophistiquée. Cette double nature séduit un public bien plus large que les seuls passionnés d'informatique.

Les chiffres parlent d'eux-mêmes : entre 12 et 17 millions d'exemplaires vendus jusqu'en 1994 font du C64 l'ordinateur personnel le plus diffusé de l'histoire. Cette longévité exceptionnelle de douze ans témoigne d'une conception robuste et évolutive. Quand les PC compatibles dominent le marché professionnel, le petit Commodore continue de régner sur le divertissement domestique.

La bibliothèque logicielle explose littéralement. Des milliers de titres voient le jour, des classiques comme *Impossible Mission* aux créations amateurs distribuées via les réseaux BBS. Ces bulletins électroniques, ancêtres de nos forums, connectent les utilisateurs par modem. On y échange programmes, astuces et passions communes. Des outils comme le *Shoot'em Up Construction Kit* démocratisent la création vidéoludique. Plus besoin d'être programmeur pour concevoir son propre jeu de tir !

Le traitement de texte EasyScript ou le tableur CalcResult prouvent que le C64 ne se limite pas au jeu. L'arrivée de GEOS en 1986 bouleverse

l'interface utilisateur. Ce système graphique propose fenêtres, icônes et menus déroulants, rivalisant avec le Macintosh d'Apple dans un segment de marché pourtant différent.

Mais c'est la communauté qui forge véritablement l'âme du C64. Des sociétés comme HAL Laboratory repoussent continuellement les limites techniques. La « demoscene » naît de cette émulation créative. Ces programmeurs artistes créent des démonstrations visuelles époustouflantes, mélangeant effets graphiques impossibles, animations fluides et compositions musicales raffinées. Leurs œuvres prouvent qu'une machine modeste est capable de rivaliser avec des systèmes bien plus coûteux, à condition de maîtriser chaque registre, chaque cycle processeur.

Avec cette puce, un véritable instrument de musique électronique, des compositeurs comme Rob Hubbard ou Martin Galway créent un style musical spécifique : la « SID music ». Leurs mélodies marquent durablement les bandes-sons de jeux vidéo et influencent la musique électronique naissante. Qui aurait imaginé qu'un ordinateur domestique engendrerait un nouveau genre musical ?

Sinclair ZX Spectrum

En 1980, l'ordinateur personnel restait un objet mystérieux pour la plupart des gens. Les machines coûtaient une fortune et semblaient réservées aux initiés ou aux entreprises qui avaient les moyens. Clive Sinclair nourrissait une ambition différente. Cet entrepreneur britannique, qui avait déjà marqué son temps avec ses calculatrices de poche, voulait créer un ordinateur que chacun pourrait s'offrir.

En avril 1982, Sinclair Research Ltd lançait le ZX Spectrum, une petite boîte noire qui allait bousculer bien des certitudes. Avec son prix de 125 livres sterling pour 16 Ko de mémoire ou 175 livres pour la version 48 Ko, cette machine cassait littéralement les codes du marché. Le design était d'une simplicité déconcertante : un boîtier compact avec un clavier à membrane qui se branchait directement sur n'importe quel téléviseur familial.

Sous ce capot minimaliste battait le cœur d'un Zilog Z80A à 3,5 MHz. Sinclair avait fait des choix techniques astucieux. La vraie trouvaille résidait dans la gestion des couleurs : huit teintes différentes avec deux niveaux de luminosité chacune, d'où ce nom de « Spectrum » qui sonnait comme une promesse. L'écran affichait 256 × 192 pixels, une définition correcte

pour l'époque. Le son se limitait à un simple buzzer, mais qu'importe ! Les programmes se chargeaient depuis des cassettes audio ordinaires, celles-là mêmes qu'on utilisait pour écouter de la musique.

Le système embarquait une version du langage BASIC adaptée par Nine Tiles Ltd, conçue pour tirer parti des capacités graphiques et sonores de la machine. Pas de fioritures dans le système d'exploitation, juste l'essentiel pour que tout fonctionne. Cette approche dépouillée cachait en réalité une intelligence remarquable : rendre l'informatique accessible sans la simplifier à l'excès.

La commercialisation débuta par vente directe depuis les bureaux de Sinclair, avant de gagner les rayons des magasins britanniques. L'accueil dépassait toutes les prévisions. Les gens découvraient qu'ils pouvaient enfin s'offrir un « vrai » ordinateur, pas un gadget. L'engouement fut tel qu'une industrie du jeu vidéo émergea presque instantanément autour de cette petite machine noire.

L'usine Timex de Dundee, en Écosse, tournait à plein régime. La production grimpa jusqu'à 50 000 unités par mois en 1983 pour répondre à une demande qui ne faiblissait pas. Le cap du million d'exemplaires vendus fut franchi dès 1983, puis celui des deux millions l'année suivante. Le Spectrum devenait l'ordinateur personnel le plus vendu d'Europe.

Cette réussite commerciale permit à Sinclair de développer tout un univers autour de sa machine. L'imprimante thermique ZX Printer, les interfaces joystick, et surtout les lecteurs Microdrive lancés en 1983 vinrent compléter l'offre. Si leur prix élevé en limitait l'adoption, ces petites cartouches offraient un stockage plus rapide et fiable que les cassettes.

La presse spécialisée fleurissait avec des magazines entièrement dédiés au Spectrum. Les lecteurs y trouvaient des programmes à recopier ligne par ligne et des cours de programmation accessibles. Une véritable industrie du logiciel se structurait autour d'éditeurs comme Ultimate Play The Game, Ocean, ou Imagine. Les jeux gagnaient en sophistication, exploitant chaque octet de mémoire disponible.

L'éducation nationale britannique ne resta pas indifférente au phénomène. Voyant dans cette machine un moyen de préparer une génération entière aux enjeux de l'informatique, le gouvernement subventionna l'acquisition de Spectrum dans les écoles. Des milliers d'écoliers découvrirent ainsi la programmation sur ces petits claviers à membrane.

Face à une concurrence qui s'intensifiait, Sinclair lança en octobre 1984 le Spectrum+. Même électronique, mais dans un boîtier repensé avec un vrai clavier mécanique inspiré de celui du QL, l'ordinateur haut de gamme de la marque. Vendu 179,95 livres, il séduisait ceux qui avaient été rebutés par le clavier à membrane de l'original. Un kit de mise à niveau était proposé

aux propriétaires des premiers modèles.

Le catalogue de logiciels s'étoffait de jour en jour, dépassant les 10 000 titres en 1985. Si les jeux dominaient largement, on trouvait aussi des applications professionnelles et éducatives. La communauté des développeurs partageait ses trouvailles et ses astuces, créant une émulation créative remarquable autour des limites techniques de la machine.

Mais le marché évoluait vite. L'arrivée du Commodore 64 et de l'Amstrad CPC changeait la donne. Ces concurrents proposaient de meilleurs claviers et des capacités multimédia supérieures pour des prix comparables. Les ventes du Spectrum commencèrent à s'essouffler dès 1985.

Les difficultés financières de Sinclair Research s'aggravaient. L'échec cuisant d'autres projets comme l'ordinateur QL ou le véhicule électrique C5 plombaient les comptes. En 1986, Amstrad rachetait la division ordinateurs. La production du Spectrum continua sous ce nouveau propriétaire jusqu'en 1992, totalisant plus de 5 millions d'exemplaires vendus toutes versions confondues.

Le Spectrum a littéralement démocratisé l'informatique personnelle en Europe, la sortant des laboratoires pour l'installer dans les salons familiaux. Des générations entières de programmeurs et de créateurs de jeux vidéo ont fait leurs premières armes sur ce petit clavier à membrane. Sa simplicité encourageait l'expérimentation et la compréhension des mécanismes informatiques.

Une communauté passionnée continue de créer pour le Spectrum. Des émulateurs permettent de revivre l'expérience sur nos machines modernes, tandis que des développeurs s'amusent à repousser les limites de cette architecture vieille de plus de quarante ans. Le Spectrum est devenu le symbole d'une ère bénie où programmer restait à la portée de tous.

Son impact culturel se mesure moins en parts de marché qu'en vocations suscitées. Cette petite boîte noire a démystifié l'informatique en montrant qu'un ordinateur pouvait être simple, abordable et puissant à la fois. La vision de Clive Sinclair, rendre la technologie accessible au plus grand nombre, trouve ici sa plus belle illustration. D'autres ordinateurs lui ont bien succédé, plus puissants et plus confortables, mais aucun n'aura eu un impact social comparable.

Thomson TO/MO

En 1979, dans les ateliers de l'usine Thomson de Moulins, José Henrard manipule des prototypes de cartes électroniques. Cet économiste et sociologue, chercheur au CNRS, découvre l'informatique grâce à l'ordinateur de son frère formateur. Thomson le recrute avec une mission ambitieuse : concevoir un micro-ordinateur familial français. Sa vision dépasse le simple gadget technologique. Il veut créer une machine qui démystifie l'informatique, qui enseigne et qui communique.

Son prototype intègre un microprocesseur 8 bits Motorola 6809, réputé parmi les plus performants de l'époque. Avec ses 8 Ko de mémoire vive extensible à 32 Ko et sa résolution graphique de 320×200 points en 8 couleurs, le TO7 se distingue par son crayon optique livré en standard. Désormais l'utilisateur interagit directement avec l'écran, là où la plupart des ordinateurs se contentent encore d'un simple clavier.

En septembre 1980, Henrard présente son prototype aux dirigeants de Thomson. Leur perplexité est manifeste face à ce qu'ils perçoivent comme un gadget. Le projet obtient néanmoins le feu vert. La production démarre à Moulins. Deux ans plus tard, Thomson, fraîchement nationalisée sous la direction d'Alain Gomez, dévoile les 100 premiers TO7 au SICOB. La presse spécialisée applaudit : qualité graphique remarquable, dimension ludique assumée, Basic Microsoft robuste. Seul bémol : le prix de 7 000 francs.

L'année suivante est stratégique. Thomson crée la SIMIV (Société Internationale de Micro-Informatique et Vidéo) pour orchestrer sa politique micro-informatique. Jean Gerothwohl, ancien publicitaire et condisciple d'Alain Gomez à Sciences Po, prend les rênes. Sa stratégie privilégie l'éducatif et s'inscrit dans la durée. Les ventes décollent grâce à des campagnes marketing audacieuses : publicité grand public, promotions ciblées, dossiers de presse fouillés.

1984 voit naître deux nouveaux modèles. Le MO5, proposé à 2 390 francs, défie les machines britanniques comme l'Oric ou le Spectrum. Le TO7-70, lui, embarque 64 Ko de mémoire vive. Thomson décroche un contrat décisif avec l'UGAP : 40 000 machines sur cinq ans pour l'Éducation nationale. Un succès qui ouvre la voie à une ambition plus vaste.

En janvier 1985, Laurent Fabius, alors Premier ministre, annonce le plan « Informatique Pour Tous ». En voulant initier tous les élèves français à l'informatique et former 110 000 enseignants en une année, l'objectif frôle l'utopie. Thomson est le fournisseur principal avec une commande pharaonique de 108 400 machines destinées à équiper 24 000 établissements scolaires. L'entreprise lilloise Léanord développe le « Nanoréseau », système ingénieux permettant de connecter jusqu'à 31 ordinateurs Thomson à une

machine maître plus puissante.

Le TO7-70 révèle ses innovations techniques. Sa mémoire vive atteint 48 Ko, extensible à 64 Ko. Il gère désormais 16 couleurs contre 8 précédemment. Son système d'exploitation propriétaire dialogue avec le Basic et le Logo. L'ordinateur multiplie les interfaces : lecteur de cassettes pour stocker les programmes, port cartouche pour les logiciels, prise Péritel pour la télévision, interface sonore, manettes de jeu.

Une bibliothèque logicielle riche se développe autour de l'éducation, notamment pour l'apprentissage du français et des mathématiques. Les applications bureautiques arrivent avec le tableur Colorcalc et la base de données Pique-Fiche. Colorpeint démocratise le dessin assisté par ordinateur. Les jeux complètent cette offre diversifiée.

Pourtant, Thomson peine face à la concurrence internationale. Les tentatives d'exportation échouent, hormis quelques ventes en Italie, et un contrat avec l'Algérie. En 1986, l'entreprise lance le TO9, modèle plus puissant facturé 8 990 francs, mais la compatibilité avec les modèles antérieurs pose problème. Les utilisateurs s'interrogent, les développeurs hésitent.

L'arrivée des compatibles PC à bas prix, menés par Amstrad, bouleverse brutalement le marché. Thomson abandonne sa stratégie de standard propriétaire en 1986 pour se lancer dans les compatibles PC avec la gamme TO16. Trop tard. Les ventes s'effondrent : 160 000 machines en 1986, 100 000 en 1987, 60 000 en 1988. L'usine de Saint-Pierre-Montlimart ferme ses portes, supprimant 450 emplois.

Le 27 janvier 1989, Thomson annonce l'arrêt définitif de son activité micro-informatique. Cette décision, prise sans concertation avec les utilisateurs ni l'Éducation nationale, clôt près d'une décennie d'aventure industrielle française. Le désengagement définitif intervient le 1er janvier 1990. Thomson préfère se recentrer sur l'électronique grand public et la défense.

L'héritage des ordinateurs Thomson marque l'informatisation de la société française. Le plan « Informatique Pour Tous » a permis la première introduction massive de l'informatique dans les écoles. Leur conception privilégiant l'interaction directe via le crayon optique et la dimension éducative témoigne d'une vision distinctive de l'informatique personnelle. Les choix techniques audacieux, du microprocesseur 6809 au Nanoréseau, illustrent la capacité d'innovation de l'industrie française des années 1980.

L'échec commercial résulte d'un marché national trop étroit, de l'absence de percée internationale, de tarifs élevés et surtout de l'émergence implacable du standard PC. Cette aventure représente néanmoins une étape significative de l'industrie informatique française, mêlant ambition technologique, volonté politique d'indépendance nationale et démocratisation de l'accès à l'informatique.

PostScript

En 1982, deux chercheurs de Xerox PARC décidèrent de claquer la porte. John Warnock et Charles Geschke venaient de passer des mois à développer une technologie d'impression baptisée Interpress, capable de décrire une page à imprimer sans se soucier du type d'imprimante utilisée. Le concept était novateur, mais Xerox ne voyait pas l'intérêt commercial de la chose. Les deux hommes fondèrent alors Adobe Systems et se mirent au travail sur leur propre version : PostScript.

L'époque était pourtant peu propice à ce genre d'aventure. Les années 1970 baignaient dans une conception purement calculatoire de l'informatique. IBM vantait dans ses publicités le coût dérisoire des multiplications sur ses machines. Mais au Xerox PARC, une autre idée germait : l'ordinateur deviendrait un outil de communication. Chacun aurait bientôt sa machine personnelle, reliée à des imprimantes partagées par le réseau.

Le plan initial d'Adobe prévoyait de fabriquer des imprimantes et d'ouvrir des centres de services d'impression. Bill Hambrecht, leur investisseur, les persuada de changer de cap. Pourquoi ne pas plutôt concevoir des systèmes complets associant stations de travail et imprimantes laser ? C'est là que Steve Jobs fit irruption dans l'histoire.

Jobs découvrit PostScript et saisit immédiatement son potentiel. Son Macintosh révolutionnait l'interface utilisateur, mais côté impression c'était dramatique. L'ImageWriter ne dépassait pas 72 points par pouce, une résolution dérisoire pour un usage sérieux. En 1985, Apple sortit la LaserWriter, première imprimante à embarquer PostScript. Cette machine contenait plus de puissance de calcul que le Macintosh : 512 Ko de mémoire morte rien que pour le code PostScript, une véritable fortune.

Le prix annoncé de 7 000 dollars affolait les commerciaux d'Apple. Trois fois le prix du Macintosh pour une imprimante ! Jobs tint bon, et la chute opportune du prix des puces mémoire finit par donner raison à son pari.

PostScript résolvait un problème technique complexe avec une approche d'une simplicité déconcertante. Le langage considérait le texte comme un cas particulier des graphiques. Cette unification permettait de manipuler les caractères avec une liberté totale : on pouvait les tourner, les redimensionner, les déformer à volonté. L'idée séduisait d'autant plus qu'elle s'affranchissait de la résolution du périphérique de sortie. Une description de page pouvait s'imprimer sur différents types d'imprimantes en conservant sa qualité.

Adobe ne se contenta pas de créer un standard d'impression. En 1987,

l'entreprise lança Illustrator, un logiciel de dessin vectoriel. L'époque était audacieuse car aucun graphiste n'utilisait d'ordinateur. L'interface d'Illustrator, fondée sur les courbes de Bézier et leurs points de contrôle, naquit d'une anecdote familiale. La femme de Warnock, graphiste de métier, lui demandait régulièrement de programmer à la main en PostScript les logos qu'elle devait réaliser. Cette expérience domestique inspira directement l'interface du logiciel.

L'année suivante, Adobe racheta un petit programme développé par les frères Knoll. Ce logiciel allait devenir Photoshop. Le pari paraissait fou : les disques durs les plus volumineux ne dépassaient pas 20 mégaoctets, juste de quoi stocker une photo haute résolution. Adobe misait sur l'évolution du matériel, et l'histoire lui donna raison.

En 1989, la position dominante de PostScript vacilla. Apple et Microsoft s'associèrent pour créer TrueType, un format concurrent pour les polices de caractères. Adobe riposta en développant à marche forcée ATM (Adobe Type Manager) pour l'affichage des polices à l'écran. La qualité technique de la solution et la confiance qu'inspirait Adobe lui permirent de conserver son avance.

Deux ans plus tard, Warnock eut une illumination. Pourquoi ne pas utiliser PostScript pour créer des documents portables ? Il suffisait de capturer le flux PostScript généré par n'importe quelle application pour obtenir un fichier lisible sur toute plateforme. Cette idée donna naissance à Acrobat et au format PDF. Pour que le succès arrive, il fallut qu'Internet se développe pour que l'échange de documents formatés soit une nécessité évidente.

Le triomphe d'Adobe repose sur des intuitions remarquables. L'entreprise sut identifier des besoins dans le marché. Elle maintint une exigence technique sans faille dans le rendu typographique. Sa stratégie commerciale, qui consistait à licencier PostScript aux fabricants plutôt qu'à produire ses propres machines, s'avéra judicieuse.

Le langage PostScript établit des principes toujours actuels comme la séparation entre la description du document de son rendu, l'affranchissement des contraintes matérielles, le traitement uniforme du texte et des graphiques. Le PDF, héritier direct de PostScript, règne sur l'échange documentaire mondial. Au-delà de ces aspects techniques, PostScript transforma l'ordinateur en outil de création graphique et de communication, l'arrachant définitivement à son statut de simple calculatrice.

SunOS

En 1982, quand Bill Joy quitte les couloirs de l'université de Berkeley. Il emporte avec lui bien plus qu'un diplôme de master en génie électrique. Cet homme a passé des années à triturer BSD, cette version d'UNIX née dans les laboratoires universitaires californiens. Il a créé l'éditeur `vi`, développé le shell C, et connaît les entrailles du système mieux que quiconque. Il cofonde une nouvelle société : Stanford University Network, raccourci en Sun. Sun Microsystems. Une époque où AT&T commence tout juste à commercialiser UNIX, et où l'informatique vit une période d'effervescence rare.

La première mouture de SunOS voit le jour en 1983. Joy et ses collègues prennent BSD comme base, mais ils ne se contentent pas de reproduire l'existant. Ils y greffent des outils réseau remarquablement efficaces et mettent au point le Network File System, ce NFS qui deviendra plus tard incontournable dans l'industrie. Parallèlement, Sun travaille sur des solutions de fenêtrage pour UNIX. Leur objectif ? Rendre ce système, réputé austère, accessible au plus grand nombre.

Les années 1980 consacrent SunOS comme une référence, surtout dans les domaines techniques pointus. Les bureaux d'études en conception assistée par ordinateur s'arrachent les stations de travail Sun. Ces machines, animées par SunOS, écrasent littéralement les ordinateurs personnels de cette période. Elles offrent un environnement multi-utilisateurs stable, ce qui change tout pour les entreprises habituées aux limites des systèmes mono-utilisateur.

Mais en 1988, un coup de tonnerre secoue le petit monde d'UNIX. AT&T entre au capital de Sun Microsystems. Les concurrents voient rouge, persuadés que Sun va bénéficier d'informations privilégiées sur l'évolution d'UNIX. Ils ripostent en créant l'Open Software Foundation, tandis qu'AT&T, Sun, Data General et Unisys mettent sur pied UNIX International. Ces manœuvres révèlent les tensions qui traversent l'industrie informatique, tiraillée entre standardisation technique et guerre commerciale.

Le paysage change encore en 1993. Sun annonce que SunOS 4.1.4 sera la dernière version construite sur BSD. L'entreprise bascule vers System V Release 4, fruit d'une collaboration avec AT&T qui marie les atouts de System V et de BSD. Cette nouvelle mouture hérite du nom commercial Solaris, SunOS devenant la désignation technique du noyau. Un virage stratégique qui s'inscrit dans une démarche de standardisation plus large du monde UNIX.

L'interface graphique constitue un autre terrain d'innovation et de bataille. Sun développe avec AT&T l'interface OPEN LOOK, dans l'espoir de créer une expérience utilisateur unifiée pour toutes les variantes d'UNIX.

Mais cette interface se heurte à OSF/Motif, bâtie sur X Window, ce système développé au MIT qui présente un avantage de taille, à savoir d'exécuter des programmes sur une machine distante en affichant leur sortie localement, quelle que soit l'architecture matérielle ou le système d'exploitation.

La confrontation tourne à l'avantage de Motif. Sun encaisse le coup et s'adapte : à partir de Solaris 2.5.1, l'entreprise propose un package regroupant les deux interfaces sous l'appellation Common Desktop Environment. Cette capacité d'adaptation témoigne de l'agilité de Sun face aux évolutions du marché et à l'émergence de nouveaux standards.

Sur le plan technique, SunOS apporte des innovations qui marquent durablement l'informatique. Le système de fichiers rapide, introduit dans BSD 4.2 grâce au financement de la DARPA, révolutionne les performances et combat la fragmentation des fichiers. Le NFS, conception purement Sun, s'impose comme la solution de référence pour le partage de fichiers en réseau. Ces avancées, couplées à l'intégration poussée avec le matériel Sun, expliquent le succès commercial du système.

L'histoire de SunOS reflète celle de l'informatique dans son ensemble. Né dans l'univers universitaire comme une variante de BSD, héritier de cette tradition de recherche qui caractérise UNIX, il évolue vers une solution commerciale complète. Il intègre les standards qui émergent sans renier ses spécificités techniques. Cette métamorphose accompagne celle de Sun Microsystems, qui passe du statut de start-up audacieuse à celui de poids lourd de l'industrie.

Quand SunOS cède la place à Solaris, une page se tourne, mais l'héritage demeure. Les innovations qu'il a portées, notamment dans le réseau et les systèmes de fichiers distribués, continuent d'influencer le développement des systèmes d'exploitation modernes. Son parcours illustre comment les choix techniques, les stratégies commerciales et les standards industriels s'entremêlent pour façonner l'évolution technologique.

SunOS a contribué à établir UNIX comme une plateforme crédible pour les applications professionnelles, particulièrement dans les secteurs techniques et scientifiques. Son succès dans les universités et centres de recherche a formé toute une génération d'informaticiens aux concepts UNIX.

Agat

En 1980, tandis que l'Apple II connaît un succès retentissant dans les écoles américaines, l'Union soviétique se lance dans une aventure techno-

logique ambitieuse mais semée d'embûches. Les dirigeants du Kremlin ont pris conscience d'un retard inquiétant : leurs enfants grandissent sans jamais toucher un clavier d'ordinateur. Face à cette réalité, ils décident de créer l'Agat, copie presque conforme de la machine de Steve Wozniak.

L'histoire de l'Agat commence par un constat douloureux pour les responsables soviétiques. Dans une société qui se voulait à la pointe du progrès scientifique, l'informatique personnelle reste un terrain largement inexploré. Le ministère de l'Éducation comprend qu'il faut agir vite et former une génération d'utilisateurs d'ordinateurs prioritairement. Plutôt que de partir de zéro, les ingénieurs soviétiques font le choix pragmatique de reproduire ce qui fonctionne ailleurs.

Cette décision révèle une stratégie plus large pratiquée par l'URSS depuis des décennies. Si cette approche pose des questions sur l'autonomie technologique du pays, copier les innovations occidentales fait gagner du temps sur la recherche. L'Agat s'inscrit dans cette logique de rattrapage accéléré, avec tout ce que cela suppose.

La production démarre donc dans des usines qui découvrent les subtilités de l'électronique grand public. Les premières machines sortent des chaînes d'assemblage avec un design qui rappelle furieusement l'Apple II, mais les similitudes s'arrêtent souvent à l'apparence. Le cœur de la machine bat au rythme d'un processeur K588, version soviétique du célèbre 6502. La mémoire vive oscille entre 16 et 32 kilo-octets selon les modèles, des chiffres respectables pour l'époque.

Sorti en 1983, les problèmes s'accumulent dès les premiers mois. Les composants électroniques produits localement ne tiennent pas leurs promesses de fiabilité. Les taux de panne grimpent, transformant chaque livraison d'ordinateurs en loterie technologique. Les écoles qui reçoivent ces machines découvrent qu'une partie d'entre elles ne fonctionnent tout simplement pas, ou tombent en panne après quelques semaines d'utilisation.

Le défi logiciel s'avère encore plus redoutable. Reproduire le matériel est possible avec de bons ingénieurs et de la persévérance, mais créer un écosystème logiciel complet demande des ressources et une expertise qui font défaut. Les programmes conçus pour l'Apple II refusent obstinément de tourner correctement sur l'Agat. Chaque adaptation nécessite un travail considérable, ralentissant d'autant le développement d'une bibliothèque logicielle digne de ce nom.

Les chiffres de production parlent d'eux-mêmes. En trois ou quatre ans, à peine 6 000 unités sortent des usines soviétiques. Pour un pays qui compte des dizaines de milliers d'écoles, ces volumes restent dérisoires. L'objectif d'informatiser massivement l'éducation se transforme en vœu pieux. Seuls quelques établissements pilotes bénéficient de ces machines rares, créant

une inégalité criante entre les élèves soviétiques.

Cette pénurie trouve ses racines dans les limites structurelles de l'industrie soviétique. La planification centralisée, si efficace pour produire de l'acier ou des tracteurs, peine à s'adapter aux exigences de l'électronique de précision. Les délais de livraison s'étirent, les spécifications changent en cours de route, et la coordination entre les différents fournisseurs de composants tourne au cauchemar logistique.

L'isolement technologique aggrave la situation. Les embargos occidentaux sur les technologies sensibles privent l'URSS de composants avancés et de machines-outils de précision. Les ingénieurs soviétiques doivent réinventer des solutions éprouvées ailleurs, perdant un temps précieux dans une course où chaque mois compte.

L'État soviétique nourrit des sentiments ambigus envers l'informatisation qu'il prône officiellement. D'un côté, les dirigeants comprennent l'enjeu stratégique de ces technologies. De l'autre, ils redoutent la circulation incontrôlée d'informations que pourrait favoriser la démocratisation des ordinateurs. Cette méfiance transparaît dans les restrictions d'usage imposées aux Agat : pas question de les laisser aux mains d'utilisateurs non encadrés.

Les responsables éducatifs découvrent aussi que posséder des ordinateurs ne suffit pas. Former les enseignants, créer des programmes pédagogiques adaptés, maintenir le parc informatique : autant de questions annexes qui n'avaient pas été anticipés. Beaucoup d'Agat finissent dans des placards faute de personnel compétent pour les utiliser.

Vers le milieu des années 1980, l'enthousiasme initial cède la place au réalisme. Les responsables soviétiques commencent à regarder du côté des clones d'IBM PC, plus faciles à produire et bénéficiant d'un écosystème logiciel plus riche. L'Agat, qui devait symboliser l'indépendance technologique soviétique, symbolise progressivement l'échec.

La perestroïka accélère cette remise en cause. L'ouverture progressive aux technologies occidentales rend caduc le projet d'autarcie informatique. Pourquoi s'obstiner à réinventer la roue quand on peut acheter ou copier plus efficacement ce qui existe ? Cette question, longtemps occultée par l'idéologie, refait surface avec force.

L'abandon de l'Agat marque la fin d'une illusion. L'URSS découvre que rattraper son retard technologique demande plus que de la volonté politique et des investissements massifs. Les écosystèmes technologiques modernes reposent sur des réseaux complexes d'innovations, de compétences et de savoir-faire qui ne se décrètent pas depuis Moscou.

Cette expérience laisse des traces dans le paysage technologique post-soviétique. Le retard accumulé dans les années 1980 pèse encore aujourd'hui sur les pays de l'ex-URSS. La Russie est largement tributaire des technolo-

gies informatiques occidentales, situation qui trouve ses origines dans l'échec de projets comme l'Agat.

Au-delà de ses aspects purement techniques, l'aventure de l'Agat raconte l'histoire d'un système confronté à ses propres contradictions. L'URSS voulait moderniser son éducation sans perdre le contrôle de l'information, innover sans remettre en cause ses structures rigides, rattraper l'Occident sans s'inspirer de ses méthodes. Ces tensions, longtemps masquées par les succès spatiaux et militaires, explosent au grand jour avec l'émergence de l'informatique personnelle.

L'Agat témoigne d'une époque où l'innovation technologique est un enjeu géopolitique majeur. Son échec n'est que le début des difficultés plus larges que rencontrera l'URSS face au numérique. Dans cette course technologique, les règles du jeu ont changé : la créativité individuelle, la flexibilité organisationnelle et la liberté d'information prennent le pas sur la planification centralisée et le secret d'État.

L'histoire de cet ordinateur soviétique nous rappelle que les technologies ne se développent jamais dans le vide. L'Agat, dans ses réussites comme dans ses échecs, est un miroir d'une URSS en mutation, tiraillée entre ses ambitions de puissance et les limites de son modèle politique et économique.

Apple Lisa

Dans les couloirs d'Apple, fin 1979, l'effervescence est palpable. Une délégation vient de rentrer du Xerox PARC à Palo Alto, et les visages trahissent une excitation contenue. Ce qu'ils ont vu là-bas va bouleverser un projet qui semblait pourtant bien engagé. Le Lisa, dont le développement avait débuté l'année précédente comme un simple ordinateur professionnel doté d'un processeur sur mesure, va subir une métamorphose radicale.

Larry Tesler leur avait montré Smalltalk. Cette interface graphique, avec ses fenêtres qui se chevauchent et sa souris pour naviguer, représentait tout ce qu'Apple cherchait sans le savoir. Tesler, séduit par la vivacité d'esprit des visiteurs, les rejoindra en juillet 1980. Du projet initial, il ne restera que le nom de code, quelques composants et une poignée d'ingénieurs. Tout le reste sera repensé.

Apple mise gros sur cette aventure. Cinquante millions de dollars disparaissent dans les méandres du développement, l'équivalent de 200 années de travail humain. L'équipe gonfle jusqu'à atteindre 300 personnes, dont une centaine se consacrent au cœur du projet : faire dialoguer le matériel et le

logiciel dans une harmonie parfaite. Le processeur maison cède sa place au Motorola 68000, plus adapté aux exigences du multitâche et des graphiques.

Les ingénieurs d'Apple inventent ce qui deviendra notre quotidien informatique. La barre de menus apparaît pour la première fois, accompagnée d'une souris à un seul bouton. Le concept de presse-papiers naît dans ces laboratoires, tout comme cette fameuse corbeille qui avalera nos fichiers indésirables. Certes, le Xerox Star utilisait déjà des icônes, mais le Lisa va plus loin en permettant de les saisir à la souris, les déplacer, les ouvrir d'un double-clic. Les fenêtres se superposent naturellement, créant cet effet de bureau électronique.

L'obsession du détail pousse l'équipe vers des sommets de perfectionnisme. Des utilisateurs novices défilent dans les laboratoires pour tester chaque élément d'interface. Les débats sur la terminologie s'éternisent : comment nommer tel bouton, telle fonction ? Les traducteurs passent des mois à chercher les équivalents dans d'autres langues, conscients que chaque mot compte dans l'expérience utilisateur.

Sept applications accompagnent le système d'exploitation : LisaWrite pour le texte, LisaDraw pour le dessin, LisaCalc pour les tableaux ordonnés. LisaGraph traduit les données en courbes parlantes, LisaProject orchestre les projets complexes, LisaList organise l'information en bases structurées, et LisaTerminal ouvre les portes du monde extérieur. Toutes partagent une philosophie commune : la cohérence prime sur l'originalité.

L'écran de 12 pouces affiche ses pixels noir et blanc avec une précision inédite. Le processeur Motorola 68000 pulse à travers 1 Mo de mémoire, extensible au double. Les lecteurs « Twiggy », baptisés d'après le mannequin longiligne de l'époque, se révèlent capricieux. Sony les remplacera par ses lecteurs 3,5 pouces, plus fiables. Le disque dur ProFile de 5 Mo complète l'ensemble, un stockage pharaonique.

Le système d'exploitation multitâche fait figure de prouesse technique. La mémoire virtuelle et la protection contre les erreurs témoignent d'une maturité rare dans l'univers des ordinateurs personnels. Bill Atkinson grave son nom dans l'histoire avec QuickDraw, ce moteur graphique capable d'afficher 4 000 caractères et 800 lignes par seconde. Les mécanismes de redondance du système de fichiers promettent de limiter les catastrophes en cas de panne brutale.

Janvier 1983 : Apple dévoile sa création au monde. Le prix : 9 995 dollars. Une somme astronomique qui fait sourciller les entreprises les mieux nanties. Les ventes démarrent correctement, environ 80 000 unités s'écoulent en dix-huit mois. Mais l'enthousiasme retombe vite. Le tarif décourage, les applications tierces se font attendre, et surtout, un petit frère arrive : le Macintosh, plus abordable et plus compact, vole la vedette dès 1984.

Avril 1985 sonne le glas de l'aventure Lisa. La production s'arrête, laissant Apple avec un stock embarrassant. L'ordinateur renaît brièvement sous le nom de Macintosh XL, modifié pour exécuter les applications de son petit frère grâce au programme MacWorks. Sun Remarketing rachète les invendus et prolonge artificiellement la carrière de la machine. Quant aux derniers exemplaires, ils finissent dans une décharge, victimes collatérales d'un procès d'actionnaires.

Pourtant, le Lisa a semé des graines qui germeront partout. Son ADN se retrouve dans le Macintosh, puis dans Windows, GEM ou Framework. Les menus déroulants, les boîtes de dialogue, le glisser-déposer deviennent des évidences. QuickDraw migre intact sur le Macintosh, MacProject remplace LisaProject, LisaDraw se métamorphose en MacDraw. Le gestionnaire de bureau inspire le Finder. Le Pascal Lisa continue sa carrière comme langage de référence.

Au-delà des aspects purement techniques, le Lisa redéfinit notre rapport à l'ordinateur. Il prouve qu'une machine complexe peut rester accessible, que la puissance n'exclut pas la simplicité. L'informatique sort de ses cercles d'initiés pour s'adresser au plus grand nombre. Les principes qu'il établit – manipulation directe, cohérence visuelle, réaction immédiate – traversent les décennies sans prendre une ride. Mais l'innovation ne suffit pas : encore faut-il trouver le bon moment, le bon prix, le bon public. Le Lisa échoue commercialement mais triomphe conceptuellement.

Coleco Adam

Coleco Industries décide de quitter l'univers familier des consoles de jeux vidéo en 1983 pour tenter l'aventure de l'ordinateur personnel. Cette société américaine, qui avait connu le succès avec sa ColecoVision, lance l'ADAM, une machine qui se voulait révolutionnaire dans sa conception.

L'idée de départ avait de quoi séduire : pourquoi obliger les familles à acheter séparément un ordinateur, une imprimante, un clavier et des supports de stockage quand on peut tout intégrer dans un seul produit ? L'ADAM arrivait donc sur le marché avec un package complet à 750 dollars. Dans la boîte, l'acheteur trouvait l'unité centrale équipée de 80 Ko de mémoire vive (que l'on pouvait porter à 144 Ko), un clavier détachable de 75 touches au toucher professionnel, une imprimante à marguerite capable de produire du courrier de qualité, et deux manettes de jeu pour ne pas oublier les origines ludiques de la marque.

Le système de stockage constituait l'une des originalités les plus marquantes de la machine. Coleco avait développé un format propriétaire baptisé « digital data pack », sorte de grosse cassette magnétique haute densité capable de stocker jusqu'à 500 Ko de données. Ce format promettait des capacités de stockage supérieures aux disquettes 5,25 pouces de l'époque, mais cette promesse allait se transformer en cauchemar.

D'un point de vue technique, l'ADAM adoptait une architecture pour le moins originale. Plutôt que de concentrer toute l'intelligence dans l'unité centrale, Coleco avait distribué plusieurs microprocesseurs à travers le système. L'ordinateur en contenait deux, le clavier disposait du sien, et l'imprimante avait également son propre cerveau électronique. Cette approche réseau assez visionnaire devait théoriquement permettre des opérations multitâches qu'aucun autre ordinateur familial ne proposait alors.

SmartWriter, le logiciel phare de l'ADAM, était directement gravé dans la mémoire morte de la machine. Ce traitement de texte intégré représentait un avantage concurrentiel indéniable : là où les propriétaires d'Apple II ou de Commodore 64 devaient débourser des centaines de dollars supplémentaires pour WordStar ou d'autres logiciels similaires, l'ADAM offrait cette fonctionnalité d'emblée. SmartBASIC complétait l'offre logicielle de base, avec une compatibilité annoncée avec l'AppleSoft BASIC d'Apple. Coleco promettait une future prise en charge du système d'exploitation CP/M, ce sésame vers le monde professionnel.

Le catalogue logiciel de l'ADAM témoignait des ambitions éducatives de Coleco. SmartLOGO, développé en collaboration avec Seymour Papert du MIT, proposait une initiation à la programmation pour les plus jeunes. Des jeux sophistiqués comme Buck Rogers Planet of Zoom côtoyaient des utilitaires de gestion personnelle. La rétrocompatibilité avec les cartouches ColecoVision ajoutait une dimension ludique bienvenue à cet ensemble.

Malheureusement, la réalité du terrain allait doucher l'enthousiasme initial. Dès les premiers mois de commercialisation, les retours des utilisateurs se multipliaient pour signaler des dysfonctionnements répétés. L'imprimante, pourtant présentée comme un atout majeur, tombait en panne régulièrement. Les lecteurs de bandes magnétiques se révélaient capricieux, et la documentation technique fournie était largement insuffisante pour comprendre le fonctionnement de cette machine complexe.

Le système de stockage sur bande cristallisait à lui seul tous les problèmes de conception de l'ADAM. Les utilisateurs découvrirent à leurs dépens qu'il fallait impérativement retirer les cassettes avant d'allumer ou d'éteindre la machine, sous peine de voir leurs données définitivement corrompues. Cette contrainte, jamais clairement expliquée dans le manuel, provoqua la perte d'innombrables heures de travail et alimenta la mauvaise

réputation de la machine.

L'architecture réseau, si séduisante sur le papier, engendrait des limitations inattendues. L'imprimante servait d'alimentation générale pour tout le système, ce qui rendait impossible son remplacement par un modèle plus rapide ou plus fiable. Le multitâche tant vanté se limitait en réalité à bien peu de chose : impossible de faire autre chose pendant qu'une impression était en cours.

La politique de Coleco envers les développeurs tiers ne favorisait pas l'émergence d'un écosystème logiciel riche. Contrairement à Apple qui encourageait activement les développeurs externes, Coleco imposait des accords de licence contraignants et ne fournissait que peu d'informations techniques sur son système. Cette fermeture limitait d'autant les possibilités d'enrichir le catalogue de programmes disponibles.

Les tests de performance menés par le magazine BYTE en 1984 confirmaient les impressions mitigées des premiers utilisateurs. Si SmartBASIC se montrait effectivement plus véloce que l'AppleSoft BASIC d'Apple dans certains calculs, les opérations de lecture et d'écriture sur bande s'avéraient dramatiquement lentes comparées aux lecteurs de disquettes. L'absence d'un système de sauvegarde automatique et la difficulté à copier des données d'un support à l'autre compliquaient singulièrement l'usage quotidien de la machine.

Face à ce constat d'échec, Coleco tenta de rattraper le coup. Un lecteur de disquettes 5,25 pouces fut annoncé, ainsi qu'un modem 300/1200 bauds et une extension mémoire de 64 Ko. La société promit une refonte complète du manuel technique et des corrections pour les bogues logiciels les plus gênants. Mais ces améliorations tardives ne purent sauver une machine dont la réputation était déjà durablement ternie.

L'ADAM disparut des rayons en 1985, après moins de deux années d'une existence commerciale chaotique. Les pertes financières considérables liées à ce projet précipitèrent Coleco vers des difficultés qui aboutiront à sa faillite en 1988. Cette histoire illustre cruellement les écueils qui guettaient les nouveaux entrants sur le marché de l'informatique personnelle des années 1980. La complexité technique, l'exigence de fiabilité absolue et la nécessité d'un écosystème logiciel solide constituaient des barrières à l'entrée.

Avec le recul, l'ADAM avait pourtant introduit des concepts avant-gardistes dans l'univers de l'informatique familiale. L'intégration d'un traitement de texte directement en mémoire morte préfigurait nos systèmes d'exploitation modernes avec leurs applications préinstallées. Le concept du package « tout-en-un » influencerait durablement la conception des ordinateurs personnels.

L'aventure Coleco Adam est un témoignage précieux sur l'importance

de la qualité d'exécution dans le succès d'une innovation technologique. Elle rappelle aussi combien une communauté de développeurs active s'avère déterminante pour établir une plateforme informatique viable. Ces enseignements gardent toute leur pertinence pour comprendre les dynamiques actuelles du marché technologique.

Compaq Portable

Tandis qu'IBM domine le marché avec son PC, une jeune entreprise texane, Compaq Computer Corporation, s'apprête à bousculer les règles du jeu en 1983. Son pari : créer le premier ordinateur portable entièrement compatible avec l'architecture IBM PC.

L'idée n'est pas complètement nouvelle. Deux ans plus tôt, Adam Osborne avait déjà commercialisé son Osborne 1, présenté comme le premier ordinateur portable à connaître un véritable succès commercial. Mais la machine d'Osborne souffrait d'un défaut rédhibitoire : elle ne pouvait faire tourner les logiciels conçus pour les IBM PC, qui s'imposaient comme la référence du marché. C'est exactement ce problème que les ingénieurs de Compaq vont résoudre.

Le Compaq Portable se présente sous l'aspect d'une mallette de 12 kg, équipée d'une poignée robuste. Son design annonce ce qu'on baptisera bientôt les « luggables », ces machines que l'on traîne plus qu'on ne les porte. L'écran CRT de 9 pouces trône au centre du boîtier, le clavier se détache pour libérer l'accès aux lecteurs de disquettes 5,25 pouces. Une « brique d'alimentation » externe complète l'ensemble, ajoutant son poids considérable à celui de la machine.

Sous le capot, on trouve un processeur Intel 8088, le même que celui qui équipe l'IBM PC. Cette compatibilité technique totale permet pour la première fois à un ordinateur portable de faire tourner exactement les mêmes programmes qu'une machine de bureau. L'écran monochrome, la mémoire vive extensible et le lecteur de disquettes complètent un ensemble vendu 3 590 dollars, un prix qui le positionne clairement sur le segment haut de gamme.

Le succès ne se fait pas attendre. Les professionnels découvrent enfin une machine qui leur donne la liberté de travailler en déplacement sans renoncer à leurs habitudes logicielles. La qualité de fabrication et la fiabilité du système séduisent une clientèle exigeante, habituée aux standards IBM. Compaq mise aussi sur un réseau de distribution soigneusement sélectionné et un support technique irréprochable.

En 1986, la firme texane frappe un grand coup avec le Portable 386. Cette nouvelle génération intègre le processeur Intel 80386 cadencé à 20 MHz, une première mondiale qui propulse Compaq au rang de pionnier technologique. Les performances s'envolent : 25% plus rapide que les machines à 16 MHz, et jusqu'à 3,5 fois plus véloce que les ordinateurs basés sur le 80286 à 8 MHz.

La machine se décline en deux versions : le modèle 40 avec son disque dur de 40 mégaoctets, et le modèle 100 qui double la capacité de stockage. L'innovation la plus spectaculaire réside dans l'adoption d'un écran plasma à double mode, avec une résolution de 640 x 400 pixels inédite sur ce type de machine. Un mégaoctet de mémoire vive équipe la configuration de base, extensible jusqu'à 10 mégaoctets.

L'ergonomie fait l'objet de tous les soins. Le clavier détachable s'enrichit de 12 touches de fonction programmables, la ventilation redouble d'efficacité, et des supports antichocs protègent les disques durs. Les dimensions se réduisent : 24,8 x 40,6 x 19,8 centimètres pour un poids ramené à 9 kg, soit une baisse sensible par rapport au modèle initial.

La connectivité s'étoffe avec des ports série et parallèle, une sortie RGBI pour moniteur externe, et la possibilité d'installer un modem interne 1 200 ou 2 400 bauds. Un module d'extension optionnel ajoute deux slots 8/16 bits. Pour la sauvegarde, une unité de bande magnétique de 40 mégaoctets complète l'offre d'accessoires.

Ces succès propulsent Compaq dans les sommets de l'industrie américaine. Entre 1983 et 1986, l'entreprise bat tous les records en devenant la société ayant atteint le plus rapidement le classement Fortune 500. Cette ascension fulgurante valide l'existence d'un marché pour des ordinateurs portables professionnels haut de gamme et incite la concurrence à se lancer dans la course.

Le concept du « luggable » évolue naturellement vers plus de légèreté. En 1988, Compaq présente le SLT, son premier portable adoptant le format « clamshell » en forme de coquillage qui caractérise nos ordinateurs portables modernes. Avec ses 6 kg, le SLT marque une étape vers la véritable mobilité, même si les standards actuels restent encore lointains.

L'aventure du Compaq Portable révèle l'importance de la compatibilité dans l'économie informatique. En proposant des machines 100% compatibles avec le standard IBM PC, Compaq établit un modèle économique que toute l'industrie va adopter. Cette stratégie accélère la standardisation du marché des PC et fait émerger une véritable industrie de l'informatique personnelle.

Le Compaq Portable égale les performances d'une machine de bureau, une véritable révélation à l'époque. Sur le plan commercial, elle valide le

modèle de la distribution spécialisée et établit des standards de qualité qui influencent durablement le secteur.

Cette période d'innovation intense voit les constructeurs chercher l'équilibre entre puissance, autonomie et portabilité. Les compromis du Compaq Portable, avec sa « brique » d'alimentation et ses 12 kg, peuvent paraître archaïques selon nos critères actuels. Pourtant, cette machine trace le chemin vers l'informatique mobile moderne et prouve qu'une innovation technologique réussie résulte toujours de la rencontre entre des avancées techniques, une vision juste du marché et une stratégie commerciale cohérente.

Grâce à cette combinaison gagnante, Compaq s'impose comme un acteur majeur de l'industrie informatique, position qu'elle conservera jusqu'à son absorption par Hewlett-Packard en 2002. L'histoire du Compaq Portable est celle d'une entreprise qui a su transformer une contrainte technique – la compatibilité IBM – en avantage concurrentiel décisif.

GNU Project

L'histoire commence en 1983 dans les couloirs du laboratoire d'intelligence artificielle du MIT. Richard Stallman, programmeur chevronné, vit alors une expérience qui va changer sa vie et, par ricochet, l'histoire de l'informatique. Face à une imprimante Xerox défaillante, il se heurte à un refus catégorique : impossible d'accéder au code source pour corriger le problème. Cette frustration cristallise un malaise plus profond : l'industrie logicielle bascule vers un modèle fermé qui rompt avec la tradition de partage qui caractérisait jusque-là les laboratoires universitaires.

Cette prise de conscience pousse Stallman à annoncer, cette année 1983, le lancement d'un projet d'une ambition rare : GNU. L'acronyme récursif « GNU's Not UNIX » cache un objectif révélateur : construire un système d'exploitation complet, compatible UNIX certes, mais libéré de toute contrainte propriétaire. Chaque utilisateur devra pouvoir étudier, modifier et partager les programmes qu'il utilise au quotidien.

Pour donner corps à cette vision, Stallman franchit un cap décisif en 1984 : il démissionne du MIT. Cette rupture lui garantit la propriété intellectuelle de son travail futur. Il s'attaque d'abord à un éditeur de texte, GNU Emacs, qui rencontre un succès inattendu dans la communauté informatique. Le momentum créé encourage la suite : GCC, le compilateur, voit le jour en 1987, suivi du débogueur GDB et d'une panoplie d'utilitaires système.

En 1985, un virage institutionnel s'opère avec la création de la Free Software Foundation. Cette organisation dépasse le simple soutien technique au projet GNU : elle forge les bases philosophiques et juridiques d'un mouvement naissant. Stallman y définit quatre libertés qu'il juge inaliénables : utiliser, étudier, modifier et redistribuer les logiciels. Ces principes trouvent leur traduction juridique en 1989 avec la General Public License, une licence qui retourne astucieusement le droit d'auteur contre lui-même pour garantir la pérennité des libertés accordées.

Le développement s'organise selon un modèle inédit pour l'époque. Petites équipes de volontaires dispersées géographiquement, coordination par listes de diffusion, échanges de fichiers par FTP : Internet rend possible cette collaboration décentralisée qui n'est que que le début de nos méthodes modernes du développement logiciel. Chaque équipe travaille sur des composants spécifiques selon un plan méticuleux, une approche qui contraste avec l'improvisation souvent reprochée aux projets communautaires.

Vers 1990, l'édifice GNU approche de sa complétude. Compilateur, débogueur, éditeurs, utilitaires système : presque tous les éléments d'un système d'exploitation sont en place. Il manque pourtant la pièce maîtresse : le nœud du système, son kernel. HURD, le noyau GNU, s'enlise dans une complexité technique inattendue. C'est là qu'intervient un jeune étudiant finlandais, Linus Torvalds, qui lance en 1991 le développement de Linux. La rencontre entre les outils GNU et ce nouveau noyau donne naissance à un système complet et libre que Stallman baptise GNU/Linux, soulignant ainsi la contribution de son projet.

La GPL, progressivement un standard de facto, est adoptée par des milliers de développeurs à travers le monde. Des entreprises comme Cygnus Solutions, créée en 1989, prouvent qu'un modèle économique viable peut s'articuler autour du logiciel libre en vendant support et services. La méthodologie collaborative initiée par GNU inspire une génération entière de projets.

Les questions soulevées par Stallman et la FSF alimentent des débats plus larges sur la propriété intellectuelle à l'ère numérique. Les concepts de liberté logicielle essaiment vers d'autres domaines : culture libre, données ouvertes, science ouverte. Cette expansion conceptuelle révèle la portée philosophique d'une initiative née d'une frustration technique.

Les années 1990 et 2000 confrontent le projet GNU à de nouveaux défis. Brevets logiciels, gestion numérique des droits, émergence d'un mouvement « open source » plus pragmatique : autant d'évolutions qui testent la cohérence doctrinale du mouvement. La FSF maintient sa ligne intransigeante sur les questions de liberté, parfois au prix d'une marginalisation relative face à des approches moins radicales.

Pourtant, les faits donnent raison à cette persévérance. GNU/Linux s'impose massivement dans les serveurs web, conquiert les supercalculateurs et, via Android, infiltre des milliards d'appareils mobiles. GCC demeure une référence technique incontournable. Les outils GNU continuent d'équiper la majorité des environnements de développement modernes.

Le projet GNU a démontré au-delà de ses réalisations techniques qu'une initiative portée par des convictions éthiques fortes pouvait transformer durablement tout un secteur industriel. En créant simultanément des outils performants et un cadre juridico-philosophique cohérent, Stallman et ses collaborateurs ont fondé un écosystème alternatif viable.

Cette contribution historique éclaire nos enjeux contemporains. Au moment où la technologie façonne nos sociétés avec une intensité croissante, les questions soulevées par le projet GNU sur la liberté et le contrôle dans le monde numérique résonnent avec une acuité particulière. L'utopie technique de 1983 s'est muée en réalité économique et sociale, prouvant que les idées, même les plus radicales, peuvent transformer le monde quand elles rencontrent les outils techniques appropriés.

DNS

L'histoire du Domain Name System commence par un fichier. Un simple fichier texte baptisé HOSTS.TXT que le SRI Network Information Center distribuait péniblement à travers l'ARPANET du début des années 1980. Cette solution artisanale fonctionnait tant que le réseau se limitait à quelques dizaines de machines, mais l'explosion du nombre d'ordinateurs connectés transforme ce système en cauchemar administratif.

Paul Mockapetris et son équipe de l'USC Information Sciences Institute observent cette dérive avec inquiétude en 1982. Le fichier HOSTS.TXT enfle chaque jour davantage, sa distribution coûte une fortune, et personne n'arrive plus à suivre le rythme des modifications. L'évolution technologique aggrave la situation : là où l'ARPANET initial connectait surtout des systèmes centraux partagés, l'arrivée des stations de travail individuelles multiplie exponentiellement le nombre de machines à référencer. On passe du nombre d'organisations au nombre d'utilisateurs, une perspective vertigineuse pour l'époque.

Face à cette impasse, Mockapetris explore les alternatives existantes. Il étudie IEN116 de DARPA Internet, examine Grapevine de XEROX, mais aucun système ne répond vraiment aux besoins spécifiques d'Internet.

L'équipe décide alors de concevoir une architecture entièrement nouvelle, distribuée par nature : le Domain Name System. Les premières spécifications paraissent en 1983 dans les RFC 882 et 883, documents techniques qui vont transformer l'Internet.

L'idée centrale du DNS tient dans sa structure arborescente. Chaque nœud de l'arbre porte une étiquette, et le nom complet d'un ordinateur se construit en remontant de nœud en nœud jusqu'à la racine. Cette hiérarchie naturelle rend élégante la délégation d'autorité car chaque organisation gère sa portion de l'espace des noms sans dépendre d'un organisme central pour les modifications courantes. Les premiers domaines de premier niveau reflètent cette philosophie pragmatique : .edu pour l'éducation, .com pour les entreprises, .fr et .uk pour les pays.

L'architecture technique repose sur deux piliers. D'un côté, les serveurs de noms stockent et distribuent les informations de leur zone de responsabilité. De l'autre, les résolveurs intégrés aux systèmes clients interrogent ces serveurs pour localiser l'information recherchée. Entre les deux, un système de cache sophistiqué mémorise temporairement les réponses selon une durée de vie définie par chaque administrateur. Cette mise en cache constitue le secret de performance du DNS : elle évite de répéter sans cesse les mêmes requêtes vers les serveurs distants.

La transition depuis HOSTS.TXT ne se fait pas du jour au lendemain. L'université de Berkeley prend les devants en 1985, devenant la première organisation à abandonner complètement l'ancien système au profit du DNS. Cette migration pionnière révèle immédiatement les problèmes pratiques : les utilisateurs peinent à s'habituer aux nouvelles formes d'adressage, les applications doivent être reprogrammées pour gérer les échecs temporaires inhérents à tout système distribué.

En 1987, environ 300 domaines ont été délégués par le SRI-NIC. Un an plus tard, ce nombre dépasse déjà 650, dont 400 pour des espaces de noms normaux et 250 pour des espaces d'adresses réseau. Sept serveurs racine assurent la redondance mondiale du système, judicieusement répartis sur les principaux réseaux dorsaux de l'Internet. Cette croissance exponentielle valide les intuitions de Mockapetris : le DNS absorbe sans difficulté une charge que l'ancien système n'aurait jamais pu supporter.

Les années 1990 voient exploser l'utilisation du DNS avec l'arrivée du Web et la démocratisation d'Internet. Le système prouve sa robustesse face à cette montée en charge inattendue. De nouveaux types d'enregistrements s'ajoutent progressivement pour gérer de nouvelles fonctionnalités, comme les enregistrements MX qui révolutionnent le routage du courrier électronique. L'une des forces majeures du protocole réside dans cette capacité d'évolution sans rupture de compatibilité.

Mais le succès attire aussi les convoitises malveillantes. Au début des années 2000, les attaques par empoisonnement de cache (*DNS spoofing*) se multiplient, les détournements de domaines font la une des journaux spécialisés. La communauté technique répond par la cryptographie avec DNSSEC, un protocole de sécurisation des échanges DNS. Son déploiement démarre symboliquement en 2010 pour la racine du DNS, avant de s'étendre progressivement aux domaines de premier niveau puis aux organisations.

L'internationalisation soulève d'autres enjeux, qu'ils soient techniques ou politiques. Comment permettre aux utilisateurs chinois, arabes ou russes d'utiliser leurs propres caractères dans les noms de domaine ? Les IDN (Internationalized Domain Names) apportent une réponse élégante dans les années 2000, mais leur mise en œuvre nécessite des modifications profondes pour garantir la compatibilité avec les milliards de systèmes existants.

La gouvernance du DNS connaît une évolution majeure en 1998 avec la création de l'ICANN. Cette organisation à but non lucratif prend le relais du gouvernement américain pour coordiner le système mondial des noms de domaine. L'ICANN supervise l'attribution des nouveaux domaines de premier niveau, établit les politiques de nommage, et tente tant bien que mal de concilier les intérêts techniques et commerciaux.

Entre 2012 et 2024, le DNS traverse une nouvelle phase d'expansion. Le programme des nouveaux gTLD de l'ICANN libère des centaines d'extensions inédites : .shop, .paris, .blog et des milliers d'autres. Parallèlement, les préoccupations de confidentialité poussent au développement de DNS-over-HTTPS et DNS-over-TLS. Ces protocoles chiffrent les requêtes DNS, limitant les possibilités d'espionnage et de censure par les intermédiaires.

Quarante ans après sa conception, le DNS continue de fonctionner sur ses principes originaux. Cette longévité exceptionnelle s'explique par les choix architecturaux de Mockapetris : privilégier la simplicité sur la sophistication, la flexibilité sur l'optimisation prématurée. Les mécanismes de mise en cache et la distribution hiérarchique de l'autorité restent pertinents malgré l'évolution technologique. Le DNS illustre comment une infrastructure grandit tout en préservant sa compatibilité avec l'existant.

En 2024, le DNS gère des milliards de noms de domaine et répond à des centaines de milliards de requêtes quotidiennes. Cette infrastructure invisible supporte l'intégralité du trafic Internet mondial. Ses défis actuels concernent la sécurité face aux attaques par déni de service, la confidentialité des requêtes, et la résilience des serveurs racine.

L'aventure du DNS raconte comment une innovation technique peut restructurer l'architecture d'Internet, transformer un simple annuaire en infrastructure critique mondiale. Sa conception modulaire et évolutive est un modèle pour tous les protocoles qui aspirent à la pérennité sur Internet.

C++

Dans les années 1970, Bjarne Stroustrup découvre à Cambridge les joies et les frustrations de Simula. Ce langage lui plaît par sa capacité à structurer le code avec ses classes, mais l'exaspère par sa lenteur d'exécution. Pour sa thèse sur les systèmes distribués, il doit abandonner l'élégance de Simula et se remettre dans BCPL, plus rapide mais moins organisé. Cette contradiction le marque : pourquoi faudrait-il choisir entre performances et clarté du code ?

Quand Stroustrup rejoint les Bell Labs d'AT&T en 1979, il garde cette interrogation en tête. Les projets sur lesquels il travaille – analyse du noyau UNIX, distribution sur réseau – demandent à la fois efficacité et structure. Il se lance le pari audacieux de greffer les classes de Simula sur le langage C. Naît ainsi « C with Classes », un hybride qui conserve la rapidité du C tout en y ajoutant l'organisation orientée objet.

Les premiers pas sont modestes. Stroustrup ajoute les classes, puis les constructeurs et destructeurs pour gérer automatiquement l'initialisation et la destruction des objets. Le contrôle d'accès public-privé apparaît ensuite, suivi des fonctions virtuelles qui autorisent le polymorphisme. Chaque ajout répond à un besoin précis rencontré sur le terrain.

En 1983, Rick Mascitti suggère un nouveau nom : C++. L'opérateur d'incrémentation du C symbolise parfaitement cette évolution du langage. Le nom colle et l'histoire retiendra cette appellation. Deux ans plus tard, en 1985, la première version commerciale sort des Bell Labs, accompagnée du livre fondateur de Stroustrup, *The C++ Programming Language*. Le compilateur Cfront traduit le code C++ en C, solution élégante pour assurer une portabilité maximale.

Le succès arrive vite. Les programmeurs découvrent qu'ils retrouvent leurs habitudes C tout en accédant aux bénéfices de l'orienté objet. Pas de changement brutal, mais une transition en douceur. Le langage ne ralentit pas l'exécution par rapport au C, ne demande pas d'environnement spécialisé comme Smalltalk. Cette approche pragmatique séduit une communauté grandissante.

Entre 1985 et 1989, C++ s'enrichit. L'héritage multiple fait son apparition en 1989, les templates ouvrent la voie à la programmation générique, la gestion des exceptions apporte plus de robustesse. Chaque fonctionnalité mûrit d'abord dans l'usage avant d'intégrer le langage. Cette prudence forge la réputation de stabilité de C++.

La croissance est impressionnante : de quelques centaines d'utilisateurs

en 1985, on passe à 400 000 en 1991. Les domaines d'application se multiplient : systèmes d'exploitation, jeux vidéo, interfaces graphiques, calcul scientifique. Des environnements de développement apparaissent, des bibliothèques spécialisées enrichissent l'écosystème.

En 1989 débute le processus de standardisation sous l'égide de l'ANSI et de l'ISO. Le travail aboutit en 1998 avec la publication du premier standard international (ISO/IEC 14882 :1998). Cette normalisation libère C++ de son créateur et assure sa pérennité au-delà des Bell Labs.

La Standard Template Library mérite une mention particulière. Alexander Stepanov y développe une approche transformatrice : des conteneurs génériques (vector, list, map) et des algorithmes réutilisables qui fonctionnent avec tous ces conteneurs. Cette bibliothèque transforme la manière d'écrire du C++ en privilégiant la réutilisation de code éprouvé.

Le système d'entrées-sorties `iostream` remplace avantageusement les `printf` et `scanf` hérités du C. Plus sûr au niveau des types, plus extensible, il s'intègre naturellement dans la philosophie C++. Ces ajouts à la bibliothèque standard rendent le langage plus riche sans alourdir le cœur du compilateur.

Stroustrup défend des principes constants tout au long de l'évolution : aucune dégradation des performances par rapport au C, compatibilité ascendante préservée, support de styles de programmation variés. C++ accepte la programmation procédurale classique, l'orienté objet avec classes et héritage, la programmation générique avec les templates. Cette flexibilité explique son adoption dans des contextes très divers.

Le C++ popularise l'orienté objet dans des domaines traditionnellement réservés aux langages procéduraux. Java et C# puiseront largement dans ses concepts, y compris s'ils font d'autres choix d'architecture. La programmation générique avec les templates inspire de nombreux langages ultérieurs.

C++ poursuit son évolution à travers les standards successifs. C++11 modernise le langage avec les expressions lambda et les smart pointers. C++14, C++17 et C++20 continuent d'enrichir les possibilités : modules, coroutines, concepts. Le langage s'adapte aux architectures multicœurs, aux besoins du calcul parallèle, aux contraintes de la programmation moderne.

Cette longévité tient à l'approche pragmatique de ses concepteurs. Plutôt que de révolutionner pour révolutionner, ils ont toujours privilégié l'utilité pratique et la préservation de l'existant. C++ est fidèle à ses origines : un outil puissant pour qui veut contrôler finement les ressources sans renoncer à l'expressivité du code. L'histoire de ce langage illustre comment une bonne idée, nourrie par l'expérience du terrain, traverse les décennies en s'adaptant sans perdre son identité.

Objective-C

Brad Cox et Tom Love n'imaginaient sans doute pas que leur décision d'hybrider le langage C avec les concepts de Smalltalk donnerait naissance à l'une des technologies les plus durables de l'informatique moderne. Objective-C, créé en 1983, allait traverser les décennies en s'adaptant en permanence.

Le point de départ était pourtant simple. Dennis Ritchie et Ken Thompson avaient conçu le C dans les Bell Labs à la fin des années 1960 pour développer UNIX. Ce langage procédural offrait un compromis intéressant entre performance et lisibilité, mais ses limites se révélaient dans la construction de systèmes complexes. De l'autre côté, au Xerox PARC, Alan Kay et Dan Ingalls révolutionnaient l'approche de la programmation avec Smalltalk et ses objets qui encapsulaient données et comportements.

Cox et Love ont imaginé marier ces deux mondes. Plutôt que de créer un nouveau langage de toutes pièces, ils ont étendu le C existant en y greffant la programmation orientée objet de Smalltalk. Cette approche pragmatique préservait l'investissement en code C tout en ouvrant de nouvelles possibilités de conception logicielle. La Free Software Foundation a reconnu l'intérêt de cette innovation et l'a intégrée dans sa boîte à outils.

Objective-C devint un langage de niche apprécié des informaticiens curieux. Quand Steve Jobs quitte Apple en 1985 pour lancer NeXT, il mise sur Objective-C pour développer NeXTSTEP. Ce choix technique transforme le destin du langage. NeXT pousse la collaboration jusqu'à travailler avec Sun Microsystems sur OpenStep, une version standardisée de leur système. La Free Software Foundation emboîte le pas avec GNUstep.

Apple, enlisé dans les années 1990, cherche désespérément à moderniser Mac OS. Après plusieurs tentatives ratées, deux options s'offrent à l'entreprise : acquérir Be, Inc. de Jean-Louis Gassée ou racheter NeXT. Le choix de NeXT en 1996 scelle le retour de Jobs et fait d'Objective-C le cœur technique de Mac OS X. Un simple langage expérimental se mue soudain le socle du futur d'Apple.

L'iPhone arrive en 2007. iOS, dérivé de Mac OS X, propulse Objective-C sur des millions d'appareils mobiles. L'iPad trois ans plus tard amplifie encore le phénomène. Les développeurs du monde entier découvrent ce langage hybride pour créer des applications touchant des centaines de millions d'utilisateurs. Cette explosion inattendue révèle la robustesse de l'architecture imaginée par Cox et Love vingt-cinq ans plus tôt.

Face à cette adoption massive, Apple introduit des améliorations sub-

stantielles, notamment le comptage automatique des références (ARC) qui simplifie drastiquement la gestion mémoire. Finies les fuites mémoire et les plantages liés aux pointeurs perdus : ARC automatise ces tâches fastidieuses et sources d'erreurs.

La syntaxe d'Objective-C porte les traces de cette histoire mouvementée. Le C traditionnel côtoie les crochets de Smalltalk dans une cohabitation qui surprend au premier regard mais révèle sa cohérence à l'usage. Le système de messages, hérité de Smalltalk, remplace l'appel direct de méthodes par un mécanisme plus flexible où les objets se parlent. Cette approche autorise une liaison dynamique des méthodes et facilite l'extension des systèmes.

Les frameworks accompagnent cette richesse syntaxique. Foundation fournit les briques de base : manipulation de données, collections, opérations sur fichiers. Ces bibliothèques forment un ensemble cohérent qui accélère le développement et standardise les pratiques. Le type `id` illustre parfaitement la philosophie du langage : il autorise la manipulation générique d'objets tout en préservant la sécurité du typage quand nécessaire.

L'écosystème se développe au-delà des frontières d'Apple. GNUstep maintient la portabilité *cross-platform* d'Objective-C, démontrant que les concepts dépassent l'implémentation propriétaire. Xcode enrichit l'expérience développeur avec Interface Builder et ses outils visuels. La communauté crée bibliothèques, frameworks et bonnes pratiques qui nourrissent un cercle vertueux.

Swift arrive en 2014 comme le nouveau poulain d'Apple. Cette annonce ne sonne pas le glas d'Objective-C mais témoigne de sa maturité. Apple organise une coexistence progressive plutôt qu'une rupture brutale, reconnaissant implicitement la valeur de son langage historique. Les millions de lignes de code Objective-C continueront de fonctionner pendant des années.

L'Objective-C a démontré qu'étendre un langage existant pouvait réussir là où créer *ex nihilo* échoue souvent. Son modèle de messages a inspiré de nombreux frameworks orientés objet. Sa capacité à mélanger paradigmes procédural et objet a ouvert des pistes explorées par d'autres langages.

Quarante ans après sa naissance, Objective-C reste une leçon d'ingénierie pragmatique. Cox et Love ont su identifier les forces de deux mondes différents et les fusionner intelligemment. Leur création a survécu aux modes, aux rachats, aux appareils mobiles et continue d'alimenter des applications utilisées quotidiennement par des centaines de millions de personnes. Cette longévité témoigne d'une conception solide qui a su s'adapter sans perdre son identité.

IPX/SPX

Au milieu des années 1980, quand Novell travaillait sur NetWare, ses ingénieurs se trouvaient face à un problème : créer un protocole réseau capable de faire dialoguer des ordinateurs dispersés dans l'entreprise. Ils ne partirent pas de zéro. Le système XNS que Xerox avait mis au point quelques années plus tôt leur servit de modèle. Cette approche pragmatique donna naissance à IPX/SPX, une suite de protocoles qui allait marquer une décennie entière de l'informatique réseau.

L'époque était aux premiers pas de la mise en réseau. Les entreprises découvraient l'intérêt de partager des fichiers et des imprimantes entre plusieurs postes de travail. NetWare proposait une réponse à ces besoins avec son architecture client-serveur, où IPX/SPX jouait le rôle de messager entre les machines. Le protocole s'articulait en deux couches distinctes : IPX pour acheminer les paquets d'un point à l'autre, SPX pour garantir leur bonne réception.

Novell avait trouvé une astuce élégante pour simplifier la configuration réseau. Là où d'autres protocoles nécessitaient de complexes tables de correspondance, IPX utilisait directement l'adresse MAC de la carte réseau comme identifiant de machine. Chaque adresse IPX combinait un numéro de réseau sur 32 bits, défini par l'administrateur, et cette fameuse adresse MAC sur 48 bits. Finalement, il n'y avait plus besoin de protocole supplémentaire pour faire le lien entre l'identité logique et l'identité physique de la machine.

Le routage s'appuyait sur RIP (Routing Information Protocol), mais pas le RIP d'IP. Les deux protocoles partageaient le nom sans être compatibles. Toutes les minutes, les routeurs IPX s'échangeaient leurs tables de routage en utilisant une métrique particulière appelée « tick ». Un tick correspondait à un dix-huitième de seconde, donnant une estimation du délai sur chaque liaison. Si deux routes affichaient un nombre de ticks identique, c'est le nombre de sauts qui tranchait.

SAP (Session Announcement Protocol) apportait une dimension supplémentaire à l'écosystème IPX/SPX. Ce protocole d'annonce de services transformait le réseau en un vaste tableau d'affichage où chaque serveur proclamait ses talents : partage de fichiers, impression, messagerie. Comme pour RIP, les annonces se répétaient toutes les minutes. Les routeurs tenaient à jour une liste de tous les services disponibles, prêts à renseigner n'importe quel client en quête d'une ressource particulière.

L'adaptabilité d'IPX/SPX se manifestait dans sa capacité à cohabiter avec différents standards Ethernet. Le protocole acceptait plusieurs types d'encapsulation : le format propriétaire de Novell, les standards IEEE 802.2

et 802.3, Ethernet II, et SNAP. Cette souplesse facilitait l'intégration dans des environnements hétérogènes, à condition de maintenir une cohérence dans la configuration de tous les équipements.

Au sommet de la pile protocolaire, NCP gérait les aspects métier : accès aux fichiers, contrôle de l'impression, sécurité. NetBIOS trouvait sa place grâce à une couche d'émulation qui permettait aux applications existantes de continuer à fonctionner. Les modules NLM étendaient les possibilités du système avec de nouveaux protocoles, des services de communication ou l'accès à des bases de données.

La décennie 1990 vit IPX/SPX s'épanouir dans les entreprises. Sa facilité de mise en œuvre et ses bonnes performances en faisaient un choix naturel pour les administrateurs réseau. Mais Internet grandissait et TCP/IP s'imposait comme le langage universel des communications numériques. Face à cette montée en puissance, Novell dut réviser sa stratégie.

L'entreprise intégra progressivement TCP/IP dans NetWare. Les versions ultérieures proposèrent d'encapsuler IPX dans des paquets UDP/IP pour traverser les réseaux basés sur le protocole d'Internet. L'arrivée de NDS dans NetWare 4.0 réduisit la dépendance au protocole SAP en permettant aux clients de localiser les services via un annuaire centralisé.

La transition s'accéléra dans les années 2000. Les nouvelles applications naissaient avec TCP/IP dans leurs gènes, reléguant IPX/SPX au rang de technologie héritée. Novell finit par abandonner le développement de sa suite protocolaire pour se tourner vers les solutions IP.

Cette évolution ne doit pas masquer l'influence durable d'IPX/SPX. L'utilisation directe des adresses MAC, la découverte automatique des services, ces concepts resurgiront dans des technologies plus récentes. Les protocoles modernes comme mDNS ou UPnP reprennent des idées que SAP avait déjà explorées.

L'histoire d'IPX/SPX raconte celle d'une époque où les protocoles réseau se battaient pour imposer leur vision de la communication. Face à la standardisation universelle de TCP/IP, les solutions propriétaires, techniquement abouties ou non, ne pouvaient que s'effacer. Cette leçon dépasse le simple cadre technique : dans un monde interconnecté, l'ouverture et l'interopérabilité l'emportent souvent sur l'excellence locale.

Le destin d'IPX/SPX illustre finalement une vérité immuable de l'informatique : aucune technologie, aussi brillante soit-elle, ne survit sans s'adapter aux évolutions du marché. TCP/IP n'a pas triomphé par ses seules qualités intrinsèques, mais parce qu'il est devenu la *lingua franca* d'Internet. Dans cette bataille des protocoles, c'est l'universalité qui a eu le dernier mot.

SMB

En 1984, IBM crée un protocole qui va transformer la manière dont les ordinateurs partagent leurs ressources : SMB, pour Server Message Block. À cette époque, personne n'imagine que cette technologie deviendra l'épine dorsale des réseaux d'entreprise modernes. L'idée paraît pourtant simple : permettre à des machines distantes de partager fichiers et imprimantes via NetBIOS.

Microsoft s'empare de cette innovation. Dès 1987, la firme de Redmond intègre SMB dans son programme LAN Manager, suivi l'année suivante par IBM avec son serveur OS/2 LAN. Ces premières implémentations marquent le début d'un système qui va bientôt coloniser les bureaux du monde entier. Le partage de ressources en réseau local trouve enfin sa voie.

Neuf ans plus tard, Microsoft franchit un cap décisif. Windows NT 4.0 arrive en 1996 avec CIFS, Common Internet File System. Cette version dialectale de SMB s'adapte aux réseaux d'entreprise qui gagnent en complexité. L'IETF publie un projet de norme l'année suivante, tandis que la SNIA propose une spécification technique en 1999. Le protocole sort progressivement de son carcan propriétaire.

Le millénaire change, et avec lui l'appellation du protocole. Windows 2000 renoue avec le nom SMB tout en apportant des extensions considérables. Les ports TCP dédiés (445) viennent s'ajouter aux traditionnels ports NetBIOS (137-139). Cette évolution accompagne la migration de NetBIOS vers TCP/IP, reflétant la maturité croissante des infrastructures réseau.

L'année 2006 marque une rupture. Windows Vista introduit SMB 2.0, version qui bouleverse l'architecture du protocole. Microsoft fait le ménage : les 75 commandes de l'ancienne version se réduisent à 19 instructions. Les opérations asynchrones font leur apparition, les lectures et écritures gagnent en efficacité. Le hachage HMAC SHA-256 remplace l'antique MD5, renforçant la sécurité d'un protocole devenu incontournable.

Windows 7 peaufine cette base avec SMB 2.1 en 2009. La gestion du cache s'affine, les réseaux à large bande passante trouvent leur compte. Le système de bail de fichiers remplace le mécanisme de verrouillage opportuniste, réduisant le bavardage réseau tout en améliorant la gestion des métadonnées.

Mais c'est en 2012 que SMB connaît sa plus belle métamorphose. Windows 8 et Windows Server 2012 accouchent de SMB 3.0, version qui répond enfin aux besoins des entreprises modernes. La haute disponibilité est ren-

due possible grâce au basculement transparent. Le *multichannel* agrège plusieurs connexions réseau, multipliant les performances. Le protocole RDMA s'intègre naturellement, avec des débits jusqu'alors inimaginables. Le chiffrement AES-CCM sécurise les échanges les plus sensibles.

L'évolution ne s'arrête pas là. SMB 3.02 débarque avec Windows 8.1 l'année suivante, apportant le support des clusters asymétriques et perfectionnant RDMA. Windows 10 introduit la version 3.1.1, durcissant encore la sécurité face aux menaces grandissantes.

Pendant ce temps, une autre histoire s'écrit dans le monde du logiciel libre. Andrew Tridgell développe Samba, implémentation alternative qui réconcilie les univers Windows et UNIX. Ce projet de rétro-ingénierie titanesque force Microsoft à publier ses spécifications, démocratisant un protocole longtemps gardé secret. Samba transforme SMB en standard de facto, dépassant les frontières des systèmes d'exploitation.

Cette évolution sur quatre décennies raconte l'histoire des réseaux d'entreprise. D'un simple mécanisme de partage local, SMB s'est mué en système complexe gérant performance, sécurité et haute disponibilité. Chaque version répond aux mutations de l'informatique d'entreprise : virtualisation, cloud, protection des données.

La sécurité reste le talon d'Achille du protocole. Les premiers mécanismes d'authentification LM et NTLM montrent leurs faiblesses. NTLMv2 puis les signatures de paquets, introduites avec Windows NT 4.0 SP3, colmatent les brèches. Les versions récentes intègrent chiffrement robuste et protections anti-attaques, répondant aux menaces contemporaines.

Devenu standard universel du partage de fichiers, SMB équipe tous les systèmes d'exploitation modernes. Son adaptation constante accompagne les architectures distribuées, le stockage cloud, et les nouvelles menaces cybernétiques.

La vulnérabilité découverte par le Cult of the Dead Cow en 2001 (CVE-2008-4037) impose des correctifs majeurs. Les attaques par relais SMB nécessitent l'ajout de protections spécifiques. Chaque incident forge un protocole plus résistant.

Les environnements informatiques modernes exigent toujours plus : performances accrues, sécurité renforcée, adaptation aux nouvelles architectures de stockage. Cette capacité d'évolution perpétuelle de SMB explique la longévité d'un protocole né il y a quarante ans, toujours indispensable aux entreprises du XXI^{e} siècle.

Lotus 1-2-3

VisiCalc régnait en maître sur le marché des tableurs quand Mitch Kapor et Jonathan Sachs décidèrent de créer leur propre logiciel. Nous sommes en 1982, et ces deux hommes connaissent bien l'univers qu'ils s'apprêtent à bouleverser. Kapor a travaillé chez Personal Software, l'éditeur de VisiCalc, où il a conçu VisiPlot et VisiTrend pour la visualisation graphique. Sachs a déjà programmé des tableurs, notamment pour Data General. Cette connaissance du terrain leur donne une longueur d'avance : ils savent exactement ce qui manque aux solutions existantes.

L'IBM PC, lancé l'année précédente, apporte une crédibilité nouvelle à la micro-informatique et offre des capacités techniques sans commune mesure avec les ordinateurs 8 bits. Ses 640 Ko de mémoire et son processeur 16 bits ouvrent des perspectives inédites. Kapor et Sachs décident de parier sur cette plateforme et de concevoir un tableur qui exploite pleinement ses ressources.

Leur approche technique tranche avec celle de leurs concurrents. Là où d'autres utilisent des langages de haut niveau, ils choisissent l'assembleur pour maximiser les performances. Ils programment l'interface directe avec le matériel vidéo, contournant le BIOS pour accélérer l'affichage. Ils repensent l'ordre de recalcul des cellules pour optimiser la mise à jour des formules interdépendantes, résultant en une vitesse d'exécution cinq fois supérieure à celle des autres tableurs.

L'interface utilisateur bénéficie d'innovations remarquables. Les menus déroulants avec curseur mobile et invites contextuelles rendent le logiciel accessible aux novices. L'intégration des graphiques transforme instantanément les données en diagrammes, créant des démonstrations spectaculaires qui suscitent l'enthousiasme des prospects. Les macros clavier, ajoutées malgré les réticences initiales de Sachs, donnent aux utilisateurs expérimentés la possibilité d'automatiser leurs tâches et de développer de véritables applications.

Lotus Development Corporation ne se contente pas d'un bon produit : l'entreprise innove aussi dans sa stratégie commerciale. Elle forme ses revendeurs, soigne son support client, et accompagne le logiciel d'un disque de démonstration avec tutoriel interactif. La documentation est travaillée. Surtout, la société investit massivement en publicité dans la presse économique, une démarche inhabituelle pour un éditeur de logiciels à cette époque.

Le lancement au Comdex de l'automne 1982 dépasse toutes les espérances : 900 000 dollars de commandes dès les premiers jours. En 1983, première année complète de commercialisation, le chiffre d'affaires atteint

53 millions de dollars. L'année suivante, il bondit à 156 millions, et les effectifs passent de 20 à 750 personnes en deux ans seulement, faisant de Lotus le plus gros éditeur mondial de logiciels.

Cette réussite s'explique par un alignement parfait des planètes. Le produit répond exactement aux besoins des entreprises qui découvrent la micro-informatique. Sa compatibilité avec les fichiers VisiCalc facilite la migration. La base installée d'IBM PC connaît une croissance explosive. La qualité du produit et l'exécution commerciale ne laissent aucune place aux concurrents.

Mais ce succès foudroyant crée ses propres difficultés rendant la croissance de l'entreprise ingérable. Les équipes originelles partent vers d'autres projets. Le développement de nouveaux produits s'avère complexe. Symphony, tentative de créer une suite bureautique intégrée, déçoit. Jazz, la version pour Macintosh, échoue face à Excel de Microsoft.

Les tentatives de diversification se multiplient avec Metro, Express ou Manuscript, sans jamais retrouver la magie de 1-2-3. Seul Notes, développé par Ray Ozzie, rencontre un véritable succès après le rachat de Lotus par IBM en 1995.

L'évolution technologique finit par rattraper le leader. Le passage à Windows s'avère tardif et laborieux. Le code en assembleur, initialement un atout, est finalement un boulet pour faire évoluer le produit. Microsoft Excel, conçu d'emblée pour les interfaces graphiques, grignote progressivement des parts de marché et s'impose comme le nouveau standard.

Lotus 1-2-3 a démontré le potentiel commercial gigantesque du marché des logiciels professionnels. Il établit les standards de l'interface utilisateur moderne avec ses menus déroulants et raccourcis clavier. Les macros permettent la programmation par les utilisateurs finaux. Le format DIF (Data Interchange Format) facilite l'échange de données entre applications.

Lotus 1-2-3 transforme la perception de l'ordinateur personnel en entreprise, d'outil expérimental à instrument indispensable à la productivité individuelle. Cette mutation accélère l'informatisation des organisations et fonde une véritable industrie du logiciel. Le tableur de Kapor et Sachs n'était pas seulement un produit : c'était un catalyseur qui a précipité l'entrée du monde des affaires dans l'ère numérique.

Microsoft Word

Novembre 1983. Les lecteurs de *PC World* découvrent avec surprise une disquette glissée dans leur magazine. Sur cette disquette : Multi-Tool Word, un traitement de texte inconnu signé Microsoft. Cette distribution audacieuse marque les débuts commerciaux d'un logiciel qui transformera radicalement notre rapport à l'écriture numérique.

L'histoire commence pourtant dix ans plus tôt, dans les laboratoires feutrés du Xerox PARC. Charles Simonyi, jeune informaticien hongrois fraîchement diplômé de Stanford, rejoint en 1972 cette pépinière d'innovations. Il travaille sur Bravo, un projet qui bouscule les conventions : pour la première fois, un traitement de texte affiche le texte exactement comme il apparaîtra sur papier. Fini les codes de formatage cryptiques, fini les surprises à l'impression. Le concept porte un nom qui fera fortune : WYSIWYG, « What You See Is What You Get ».

Bravo reste confiné aux laboratoires. Xerox, géant de la photocopie, ne saisit pas l'enjeu commercial de ces recherches. Bill Gates, lui, a l'œil. En 1981, il débauche Simonyi avec pour mission précise de créer les applications bureautiques de Microsoft. Le Hongrois rédige un mémorandum de trois pages et demie qui définit toute la stratégie, une suite logicielle intégrée pour particuliers et entreprises. Ce document, aujourd'hui conservé dans un coffre-fort, dessine les contours de ce qui deviendra Office.

Word naît dans ce contexte. Sa première version MS-DOS tranche avec les usages de l'époque. Là où WordStar, le leader du marché, s'accommode du clavier, Word mise sur la souris Microsoft. Les publicités vantent cette interface graphique rudimentaire qui affiche du texte en gras, en italique ou souligné. Nouveau pour MS-DOS, mais limité car il est impossible de changer de police. Les ventes peinent et WordStar garde ses fidèles, habitués à ses raccourcis clavier.

Tout bascule avec l'arrivée du Macintosh en 1984. Apple impose son interface graphique, et Microsoft saisit l'opportunité. L'équipe de Simonyi, une vingtaine de développeurs tout au plus, adapte Word au Mac. Cette fois, les concepts de Bravo trouvent leur terrain d'expression : polices multiples, mise en page sophistiquée, WYSIWYG intégral. Le logiciel révèle enfin son potentiel.

Cette philosophie de développement illustre l'approche Microsoft des années 1980. Repérer les innovations prometteuses, les adapter au marché de masse, itérer sans relâche. Simonyi cultive cette organisation volontairement réduite pour garder sa cohérence et sa vélocité, ce qui lui rappelle ses années au PARC.

Avec Windows, Microsoft transpose sur sa plateforme les acquis de la

version Mac. Windows 3.0, lancé en 1990, propulse Word au sommet. La stratégie commerciale fait mouche avec sa vente groupée à Windows, et les prix cassés. WordPerfect et WordStar, pourtant bien implantés, perdent pied. Word est désormais incontournable.

Microsoft ne se contente pas de dominer le marché du traitement de texte. L'entreprise orchestre l'intégration entre ses applications : Word, Excel, PowerPoint forment désormais Office. Cette cohérence répond aux attentes des entreprises qui cherchent des outils compatibles. Le format .doc s'impose naturellement comme référence pour échanger des documents.

Les années 1990 voient Word s'enrichir sans cesse : correcteur orthographique, dictionnaire des synonymes, publipostage, suivi des modifications. Cette accumulation de fonctions divise. Les utilisateurs occasionnels se perdent dans des menus interminables. Les professionnels apprécient cette richesse fonctionnelle. Simonyi s'interroge sur cette dérive de Word vers une complexité de plus en plus grande.

Au-delà de l'outil, Word transforme nos pratiques d'écriture. La mise en forme sophistiquée, jadis apanage des typographes et des éditeurs, se démocratise. Chacun peut créer des documents aux allures professionnelles. Cette accessibilité a son revers : les modèles intégrés standardisent la présentation. Word influence notre façon de structurer la pensée, de hiérarchiser les idées.

Cette critique accompagne Word depuis ses débuts. Le logiciel formate-t-il notre créativité ? Uniformise-t-il notre expression ? Ces questions traversent quarante années d'évolution technologique. De MS-DOS au cloud, de la disquette au smartphone, Word accompagne nos mutations numériques.

Office 365 propulse Word dans l'ère collaborative. Les documents vivent dans le cloud, se partagent en temps réel, suivent l'utilisateur sur tous ses appareils. Cette omniprésence couronne quatre décennies d'évolution continue.

Charles Simonyi a quitté Microsoft en 2002. Sa création lui survit, transformée mais fidèle à ses principes fondateurs. Traitement de texte de référence, Word est utilisé par des centaines de millions de personnes comme logiciel de traitement de texte de référence. Son histoire résume celle de l'informatique grand public, d'innovation de laboratoire à outil universel, une technologie qui façonne nos gestes quotidiens.

MSX

L'informatique personnelle des années 1980 ressemblait à un *far-west* technologique. Chaque constructeur développait ses propres machines, incompatibles les unes avec les autres, transformant l'achat d'un ordinateur familial en véritable parcours du combattant. Un logiciel écrit pour un Commodore ne fonctionnait pas sur un Amstrad, et vice versa. Cette cacophonie technologique frustra Kazuhiko Nishi, fondateur d'ASCII Corporation au Japon (aucun lien avec American Standard Code for Information Interchange) et vice-président des nouvelles technologies chez Microsoft.

L'homme avait observé avec attention le succès du standard VHS dans l'industrie vidéo. Là où Betamax et VHS s'affrontaient, la victoire était revenue au format qui avait su fédérer le plus grand nombre de fabricants. Fort de cette leçon, Nishi imagina le MSX en 1983. Son idée tenait en quelques mots : créer un standard technique que différents constructeurs pourraient adopter, garantissant ainsi la compatibilité totale entre toutes les machines estampillées du logo MSX.

Le nom choisi révélait l'ambition du projet. « Machines with Software eXchangeability », selon son créateur, traduisait cette volonté de permettre l'échange de logiciels entre ordinateurs de marques différentes. D'autres interprétations circulèrent : « MicroSoft eXtended BASIC » ou « Matsushita Sony X-machine », témoignant de l'engagement des géants technologiques derrière l'initiative.

L'architecture technique du MSX reposait sur des composants déjà éprouvés. Un processeur Zilog Z80A cadencé à 3,58 MHz constituait le cœur du système, épaulé par une puce vidéo TMS9918 capable d'afficher 16 couleurs en résolution 256x192, et un générateur de son AY-3-8910. La mémoire vive oscillait entre 8 et 128 ko, avec généralement 32 ou 64 ko selon les modèles. Cette configuration permettait l'utilisation de trois supports différents : cassettes, disquettes et cartouches.

Microsoft développa MSX-DOS, un système d'exploitation inspiré du MS-DOS des PC IBM. Il intégrait un BASIC étendu qui simplifiait la programmation et garantissait la compatibilité entre machines. Dans un marché où chaque constructeur bricolait habituellement ses propres solutions, cette standardisation matérielle et logicielle constituait une véritable innovation.

Les géants de l'électronique japonaise mordirent à l'hameçon. Sony, Toshiba, Panasonic, Canon, Yamaha : tous adoptèrent le standard. Sony lança le bal avec son HB-10 en 1983, d'abord habillé de rouge et blanc pour séduire le marché japonais, puis de noir pour l'exportation. Plus de vingt fabricants produisirent leurs propres ordinateurs MSX, certains comme Sony

et Philips proposant jusqu'à une dizaine de modèles dans leur gamme.

Au Japon, le succès fut immédiat. Les consommateurs appréciaient cette garantie de compatibilité qui leur évitait les mauvaises surprises. Les éditeurs de jeux vidéo flairèrent l'opportunité : Konami y développa sa série Metal Gear, tandis que Castlevania et Gradius trouvèrent sur MSX un terrain d'expression privilégié. C'est d'ailleurs sur cette plateforme que naquit le fameux « Konami Code », cette combinaison de touches devenue légendaire dans l'univers du jeu vidéo.

Le phénomène déborda largement des frontières nippones. La Corée du Sud, le Brésil, le Chili adoptèrent massivement le MSX. En Europe, les Pays-Bas constituèrent son bastion, Philips en faisant son cheval de bataille. L'Union soviétique succomba : le ministère de l'Éducation acquit des centaines de MSX1 et MSX2 pour équiper les salles informatiques des écoles, formant toute une génération de programmeurs derrière le rideau de fer.

Pourtant, l'Amérique résista. Seuls le SpectraVideo et le Yamaha CX-5M (commercialisé comme instrument de musique) parvinrent à franchir l'Atlantique. En 1984, le Commodore 64 régnait en maître sur le territoire américain, et le marché ne voyait guère l'intérêt d'adopter ce qu'il considérait comme un produit techniquement inférieur. Outre-Manche, la concurrence du ZX Spectrum et le prix prohibitif des jeux sur cartouche face aux cassettes bon marché limitèrent sévèrement son adoption.

Face à ces résistances, le standard évolua. Le MSX2, lancé en 1986, apporta des améliorations substantielles : davantage de mémoire, un processeur plus véloce, une puce graphique plus performante. Il conservait la compatibilité avec son prédécesseur et intégrait un lecteur de disquettes, alternative économique aux cartouches. En 1988, alors que la distribution européenne cessait, le MSX2+ apparut au Japon et en Corée, suivi par le MSX TurboR, ultime évolution du standard.

Le TurboR marqua une rupture en adoptant un processeur 16 bits tout en préservant la compatibilité Z80. Hélas, son prix élevé et des problèmes de compatibilité avec certains logiciels antérieurs brisèrent son élan. Il ne fut distribué qu'au Japon, avec un BIOS exclusivement japonais qui acheva de limiter son rayonnement.

L'émergence du PC IBM et la concentration de Microsoft sur Windows sonnèrent finalement le glas du MSX. ASCII Corporation tenta de maintenir le standard en vie, développant l'idée visionnaire de connecter les MSX en réseau pour créer un « internet primitif ». Mais le refus d'adopter le format CD-ROM enterra définitivement les espoirs du MSX3.

L'héritage du MSX perdure dans la communauté des passionnés d'informatique rétro. Sa philosophie de standardisation et sa riche bibliothèque

logicielle continuent d'inspirer développeurs et amateurs. On retrouve son influence dans la musique *chiptune* et la culture du *retrogaming*. Des projets comme le 1chipMSX, qui concentre toute la logique MSX dans une puce FPGA, perpétuent son esprit d'innovation.

L'aventure MSX illustre une tentative audacieuse de standardisation anticipant des problématiques toujours actuelles. Si le standard n'atteignit pas son objectif de domination mondiale, il démontra la viabilité d'une approche collaborative dans le développement informatique.

Novell Netware

Dans l'Utah de 1979, quatre hommes se lancent dans l'aventure informatique sans imaginer qu'ils vont révolutionner la façon dont les machines communiquent entre elles. Dennis Fairclough, George Canova, Darin Field et Jack Davis créent Novell Data Systems Inc. à Orem, loin des laboratoires de Silicon Valley. Leur bailleur de fonds, Safeguard Scientifics, mise sur cette équipe pour développer des systèmes basés sur CP/M. Les premiers pas sont laborieux : la concurrence est rude et les ventes peinent à décoller.

L'idée qui va tout changer surgit presque par nécessité. Pourquoi ne pas connecter ces micro-ordinateurs qui prolifèrent dans les bureaux, plutôt que de s'acharner sur un marché saturé ? Cette intuition ne sort pas de nulle part. Plusieurs membres de l'équipe, dont les nouveaux arrivants Drew Major, Dale Neibaur et Kyle Powell, ont travaillé sur des contrats gouvernementaux qui les ont familiarisés avec ARPANET. Ils comprennent la puissance des réseaux avant que le grand public ne découvre Internet.

La route reste semée d'embûches. Le conseil d'administration de Safeguard manque de fermer la boutique, mais l'entreprise décroche de nouveaux financements in extremis. Raymond Noorda arrive à la direction en 1983. Cet homme d'affaires expérimenté rebaptise la société Novell Inc. et mise tout sur NetWare, leur système d'exploitation réseau.

L'approche de NetWare tranche avec ce qui existe alors. Une machine proclamée serveur dédié du réseau, orchestrant l'accès aux disques durs et imprimantes que se partagent les autres postes. La topologie en étoile s'appuie sur du câblage en paire torsadée, une technologie plus fiable que les solutions concurrentes. NetWare, d'abord baptisé ShareNet, décline en deux versions : l'une pour les processeurs Intel 8086, l'autre pour les Motorola.

Les années 1980 voient l'informatique personnelle s'installer massive-

ment dans les entreprises. NetWare surfe sur cette vague et est la référence des réseaux locaux départementaux. En 1986, Novell étend son territoire avec GroupWise, une solution de messagerie et d'outils collaboratifs bien avant Microsoft Office 365.

Le début des années 1990 consacre la domination de NetWare : près de 70% du marché des systèmes d'exploitation réseau lui appartient. La version 4 transforme radicalement les possibilités du système. Novell Directory Services révolutionne l'administration en regroupant plusieurs serveurs sous une gestion unifiée. L'interface NetWare Administrator simplifie le quotidien des administrateurs réseau. Les services de fichiers et d'impression de NetWare n'ont alors pas d'équivalent.

Tout bascule avec l'explosion d'Internet à la fin des années 1990. Les réseaux locaux s'interconnectent, dépassent les frontières des départements pour embrasser des bâtiments entiers, puis des continents via les liaisons WAN. Novell réplique avec intraNetWare, qui embarque une version améliorée de NetWare 4.11. Cette mouture facilite l'installation, booste les performances et introduit le premier client 32 bits pour Windows.

IntraNetWare s'adapte aux nouveaux usages avec une passerelle IPX/IP pour connecter les postes aux réseaux IP, un serveur web intégré, le support natif de DHCP et DNS. NetWare excelle toujours dans son domaine de prédilection, mais les entreprises lui reprochent sa faiblesse pour héberger des applications métier.

Microsoft choisit ce moment pour frapper fort. La firme de Redmond rompt son partenariat avec 3Com et développe Windows NT avec des fonctionnalités réseau intégrées. Cette stratégie d'intégration verticale menace directement le modèle de Novell. Face à cette offensive, la société utahienne commet une erreur stratégique : elle rachète WordPerfect et Quattro Pro pour affronter Microsoft sur ses propres terres. Cette diversification disperse les efforts et les ressources sur des marchés que Novell maîtrise mal.

Microsoft joue la guerre d'usure. Ses ressources financières considérables lui donnent les moyens de persévérer jusqu'à ce que Windows NT s'impose face à NetWare. Novell, engagé sur trop de fronts, doit céder ses acquisitions et constamment réviser sa stratégie sans retrouver sa superbe.

Le tournant des années 2000 voit Novell épouser la philosophie du logiciel libre. NetWare renaît sous le nom d'Open Enterprise Server (OES), décliné en deux versions : l'une conserve le noyau NetWare traditionnel pour assurer la compatibilité, l'autre s'appuie sur SUSE Linux Enterprise Server. NetWare 6.5 marque la fin d'une époque : elle correspond à OES-NetWare Kernel avec le Support Pack 2. Désormais, Novell recommande d'exécuter NetWare dans une machine virtuelle Xen hébergée sur SUSE Linux Enterprise Server.

Cette trajectoire sur trente ans raconte l'évolution des réseaux d'entreprise. NetWare est né d'une intuition : connecter des machines isolées pour démultiplier leur utilité. Il a accompagné l'informatisation des organisations, transformant des postes autonomes en réseaux collaboratifs. Sa chute face à Microsoft illustre la brutalité de l'industrie informatique, où la position dominante ne garantit rien face à un concurrent déterminé et bien financé. La conversion finale au logiciel libre témoigne des mutations du secteur, où l'ouverture et l'interopérabilité sont aussi devenues des impératifs de survie.

Oric-1

Cambridge, octobre 1979. Le Dr Paul Johnson et Barry Muncaster viennent de fonder Tangerine Computer Systems dans cette ambiance effervescente qui caractérise l'industrie de la micro-informatique. Le nom de leur société s'inscrit dans cette mode un peu fantasque : après Apple, voici venue l'ère des fruits dans l'informatique. Leur premier né, le Microtan 65, ouvre la voie à ce qui deviendra bientôt une aventure bien plus ambitieuse.

Paul Kaufman rejoint l'équipe pendant l'été 1981 et prend les rênes de la *Tansoft Gazette*. L'entreprise grandit, se restructure. Début 1982, après s'être séparée de sa division Tandata Prestel, Tangerine s'installe au Cambridge Science Park et lance Tansoft, sa branche dédiée au logiciel. C'est à ce moment-là que naît Oric Products International, en avril 1982, avec Tangerine dans le rôle du laboratoire de recherche.

Au départ, l'idée semblait pourtant différente : concevoir un ordinateur de bureau pour cadres, capable de dialoguer avec le réseau Prestel. Kaufman couche sur le papier les spécifications d'un hypothétique Microtan 2, enrichi de capacités sonores et graphiques. Ces réflexions donnent naissance fin 1981 au projet Tangerine Tiger, une machine ambitieuse dotée de trois processeurs : un Z80 pour CP/M, un 6809 pour gérer les périphériques, et une puce graphique dédiée. Mais ce Tiger ne rugira jamais : H.H. Electronics rachète le projet qui sombrera dans l'oubli.

Le 27 janvier 1983, c'est dans le cadre somptueux du manoir de Coworth Park, ancienne résidence de Lord et Lady Derby, qu'Oric Products International dévoile officiellement l'Oric-1. Peter Harding, directeur commercial de 34 ans seulement, annonce la couleur avec une assurance déconcertante : six contrats majeurs signés avec des chaînes de magasins pour plus de 200 000 unités. Son objectif était de battre Clive Sinclair en offrant beaucoup plus pour beaucoup moins d'argent.

La stratégie tarifaire semble audacieuse. Il fallait débourser 129 £ pour la version 16k, 169,95 £ pour celle de 48k. Derrière cette machine, une équipe de développeurs talentueux travaille d'arrache-pied. Andy Brown et Chris Shaw, deux programmeurs chevronnés, s'attellent au ROM principal. Peter Halford développe les routines cassette et Oric Mon, tandis que Kaufman code les parties sonores sur un Microtan en Forth avant qu'Andy Brown ne les transpose en langage machine.

Les premiers échos de la presse spécialisée restent prudents. Le port imprimante Centronics fait sensation : « inhabituel, voire unique pour une machine de ce prix », note un critique. Mais les bugs du ROM et les ratés du chargement des cassettes ternissent l'image. La réaction d'Oric ne se fait pas attendre : Cosma Sales, le duplicateur de cassettes, se voit montrer la porte, accusé d'être responsable des dizaines de milliers de cassettes défectueuses qui circulent.

Outre-Manche, le succès tarde. La France se montrant plus accueillante, A.S.N. signe le 29 juin 1983 un contrat de distribution exclusive sur la base de 4 000 machines mensuelles. Le marché français développe un écosystème logiciel de qualité, faisant de l'Oric l'équivalent hexagonal du Spectrum britannique. Une réussite inattendue dans un pays pourtant réputé difficile à conquérir.

Cette embellie ne dure qu'un temps. Février 1985 : Edenspring place Oric en redressement judiciaire. Les chiffres font froid dans le dos : 5,5 millions de livres de dettes contre 3 millions d'actifs seulement. La chute semble inéluctable quand, en juin 1985, S.P.I.D. surgit de France et rachète la société pour « plusieurs centaines de milliers de livres ». L'Atmos, successeur de l'Oric-1, reprend vie dans une usine normande.

Les nouveaux propriétaires français tentent de relancer la machine avec le Telestrat, spécialement conçu pour le réseau Minitel. L'idée paraît séduisante sur le papier, mais le prix élevé et la spécificité du marché limitent les ambitions. Décembre 1987 : Oric International retombe en redressement judiciaire, cette fois avec d'importantes dettes fiscales au compteur. Un an plus tard, en décembre 1988, les chaînes de production s'arrêtent définitivement.

L'Oric-1 repose sur le populaire processeur 6502, choix dicté autant par sa réputation que par son coût modéré. Sa mémoire gère l'affichage texte et haute résolution avec des capacités sonores remarquables. Ses attributs graphiques série se révèlent adaptés aux jeux, un atout non négligeable. La machine embarque un BASIC Microsoft modifié et un modulateur UHF pour la connexion télévisée.

L'histoire d'Oric révèle les pièges de ce marché des années 1980. Malgré des caractéristiques techniques honorables et des prix attractifs, l'entreprise

a échoué à maîtriser sa croissance et à alimenter régulièrement le marché en logiciels de qualité. Les promesses non tenues se sont accumulées, notamment concernant le lecteur de disquettes Microdisc, annoncé dès 1983 mais disponible seulement en 1984, quand il était presque trop tard.

Pourtant, l'Oric a survécu à sa propre disparition grâce à une communauté d'utilisateurs passionnés. Des publications comme *Oric User Monthly* ont continué à paraître bien après la fin de la société. En France comme au Royaume-Uni, des groupes d'utilisateurs entretiennent encore la mémoire de cette machine qui, malgré ses défauts, a initié toute une génération à l'informatique personnelle.

L'aventure Oric incarne cette époque où l'audace technique côtoyait l'improvisation commerciale. Si l'échec final était peut-être inévitable, il témoigne de la vitalité créatrice de l'industrie informatique britannique des années 1980, capable de proposer des alternatives crédibles aux géants du secteur dans un marché en perpétuelle mutation.

Amstrad CPC 464

En 1984, Alan Sugar gribouillait sur une serviette en papier durant un vol transatlantique. Ce croquis allait donner naissance à l'une des machines les plus marquantes de l'informatique familiale britannique : l'Amstrad CPC 464. L'idée paraissait simple sur le papier, mais sa réalisation technique nécessitait bien plus que quelques traits au stylo-bille. Roland Perry, directeur technique d'Amstrad, hérita de cette esquisse et la transforma en réalité.

Le défi était de taille. Sugar voulait un ordinateur qui fonctionnerait dès la sortie de sa boîte, sans le casse-tête habituel des branchements multiples qui rebutait tant de familles. Le CPC 464 intégrait tout dans un seul boîtier : clavier, lecteur de cassettes, et liaison directe vers son moniteur dédié. Un seul câble d'alimentation suffisait. Cette simplicité tranchait avec les habitudes de l'époque, où monter un système informatique relevait souvent du parcours du combattant.

Perry avait initialement jeté son dévolu sur le processeur 6502, mais les développeurs britanniques connaissaient mieux le Z80. Ce processeur de Zilog, cadencé à 4 MHz, équipait déjà de nombreuses machines outre-Manche. La décision se révéla judicieuse : les 64 Ko de mémoire vive s'accompagnaient de capacités graphiques remarquables. Vingt-sept couleurs disponibles, plusieurs modes d'affichage, de quoi séduire les amateurs de jeux comme les apprentis programmeurs.

La stratégie commerciale d'Amstrad mérite qu'on s'y arrête. Sugar proposa deux configurations : 200 livres sterling pour la version écran monochrome, 300 pour la couleur. Ces tarifs se glissaient habilement entre le Sinclair Spectrum, moins cher mais limité, et le BBC Micro, plus puissant mais hors de portée de beaucoup de bourses. Surtout, Sugar anticipa l'écueil qui avait tué bien des machines prometteuses : le manque de logiciels au lancement.

Quarante prototypes partirent chez les développeurs. Des sessions de formation furent organisées, et une bibliothèque logicielle conséquente attendait donc les premiers acheteurs. Cette préparation minutieuse paya. Les utilisateurs découvrirent une machine fiable, facile à installer, qui tenait ses promesses. Si la patience restait de mise avec les temps d'accès interminables du support magnétique, le lecteur de cassettes intégré simplifiait le chargement des programmes.

Cette limitation poussa Amstrad à réagir vite. 1985 vit naître le CPC 664, équipé d'un lecteur de disquettes 3 pouces. Mais ce modèle intermédiaire céda sa place au CPC 6128, doté de 128 Ko de mémoire et d'ambitions plus professionnelles. L'évolution était lancée, mais le 464 original conservait ses atouts pour le grand public.

Les jeux firent la renommée du CPC. La série des « Roland » rendait hommage à Perry, tandis que les capacités graphiques et sonores permettaient des adaptations de qualité. Le Basic intégré ouvrait les portes de la programmation aux débutants, épaulé par une documentation soignée. Certains osèrent s'aventurer vers des applications sérieuses : Protext démontrait qu'un traitement de texte digne de ce nom pouvait tourner sur la machine.

L'international ne fut pas oublié. En France, Amstrad soigna la localisation : claviers adaptés, manuels traduits, distribution organisée. Cette attention aux spécificités nationales porta ses fruits dans quelques pays européens. Le CPC s'installait durablement dans les foyers, ouvrant à Amstrad les portes du marché informatique. L'entreprise enchaîna avec le PCW 8256 en 1985, puis se lança dans les compatibles PC dès 1986.

Les années passant, Amstrad tenta de relancer la flamme. 1990 vit arriver les modèles Plus, 464 Plus et 6128 Plus, aux graphismes améliorés. Une console dérivée, la GX4000, échoua face aux mastodontes japonais. La production s'essouffla au début des années 1990, mais l'histoire ne s'arrêtait pas là.

Une communauté active préserve l'héritage du CPC. Les émulateurs ressuscitent les logiciels d'antan, tandis que des développeurs amateurs créent de nouveaux programmes. Cette fidélité témoigne des qualités durables de la machine : conception cohérente, documentation complète, robustesse

à toute épreuve. Le pragmatisme d'Amstrad avait visé juste : simplicité d'usage plutôt que course aux performances.

Apple Macintosh

L'histoire commence en 1979 dans les couloirs d'Apple, là où Jef Raskin, ancien de l'équipe Apple II, ruminait une idée qui paraissait folle à l'époque. Il voulait créer un ordinateur aussi simple qu'un grille-pain, vendu mille dollars et produit en masse. La direction d'Apple, absorbée par les déboires de l'Apple III et le développement coûteux de la station Lisa, accueillit cette proposition avec un haussement d'épaules poli.

Tout bascula quand Burrell Smith débarqua dans l'équipe. Ce réparateur d'Apple II du service maintenance cachait un génie technique. Après une visite dans la Silicon Valley qui l'avait électrisé, il s'était mis à bidouiller avec les microprocesseurs. Son prototype artisanal — un Motorola 6809 bricolé sur un Apple II avec un écran de télé — impressionna Raskin au point qu'il l'intégra immédiatement au projet. Steve Jobs, alors vice-président mais écarté du projet Lisa pour manque d'expérience, flaira l'opportunité et prit les rênes du Macintosh.

Jobs installa son équipe loin du siège social, dans des locaux surnommés « Texaco Towers » parce qu'ils se trouvaient derrière une station-service. L'endroit ressemblait plus à une colocation étudiante qu'à des bureaux d'entreprise. Cette proximité physique créa une dynamique particulière : Andy Hertzfeld, chargé du système d'exploitation, travaillait coude à coude avec Smith sur le matériel. Cette collaboration étroite leur permit d'exploiter chaque bit de mémoire disponible.

L'équipe s'était fixé un objectif ambitieux : 300 000 unités par an. Smith, obsédé par la réduction des coûts, entreprit de révolutionner l'architecture. Là où la concurrence multipliait connecteurs et slots d'extension, le Macintosh n'aurait que deux cartes électroniques. Pas de tampons, pas d'emplacements superflus. Les fonctions vitales seraient gravées en mémoire morte pour gagner en fiabilité.

Le parti pris était radical : architecture spécialisée contre flexibilité généraliste. L'équipe disséqua méthodiquement les produits concurrents et constata que leurs connecteurs multiples ralentissaient les performances tout en gonflant les prix. Le Macintosh miserait sur un port série rapide pour les extensions futures.

L'interface utilisateur s'inspira du Lisa, mais en version allégée. Les

fenêtres graphiques permettaient d'afficher plusieurs programmes, les icônes remplaçaient les commandes obscures, la souris devenait reine. Seulement, tout devait tourner avec moins de mémoire que le Lisa, et plus vite. L'équipe réécrivit le logiciel de fond en comble, abandonna le multitâche du Lisa au profit d'une illusion créée par de petits programmes système comme la calculatrice.

Bill Atkinson leur légua QuickDraw, son logiciel de gestion graphique créé pour le Lisa. Cette boîte à outils, stockée en ROM, devint l'âme du Macintosh. Les développeurs pouvaient enrichir le système en chargeant des définitions supplémentaires depuis les disques, préservant ainsi la mémoire limitée.

Le choix du Motorola 68000 divisa l'équipe. Ce processeur coûteux dépassait largement les besoins immédiats, mais Jobs paria sur une baisse des prix. L'équipe optimisa tout : un seul mode du processeur utilisé sur les quatre disponibles, trois niveaux d'interruption sur sept, système de fichiers avec un index unique.

La fabrication nécessita une usine révolutionnaire à Fremont, en Californie. Inspirée des méthodes japonaises, elle appliquait la production en juste-à-temps et la tolérance zéro défaut. Les fournisseurs livraient de petits lots fréquents, les erreurs étaient analysées immédiatement. L'usine était conçue en trois phases d'évolution, anticipant la montée en cadence.

En janvier 1984, le Macintosh débarqua à 2 495 dollars. Le prix était loin des mille dollars rêvés par Raskin, mais l'ordinateur marqua les esprits. La publicité du Super Bowl, pastiche d'*1984* d'Orwell, présenta Apple en libérateur face à l'uniformisation IBM. L'interface graphique conviviale et le design compact créèrent un nouveau standard.

Les critiques ne manquèrent pas : prix élevé, limitations réseau, mémoire insuffisante. Mais le Macintosh avait planté une graine. Il démontra qu'un ordinateur pouvait être autre chose qu'une machine à calculer : un objet de design, une extension de la pensée humaine.

L'équipe originale avait établi des principes qui survivront à toutes les modes : design avant tout, intégration matériel-logiciel poussée, simplicité obsessionnelle. Jobs quitta Apple en 1985 après un conflit avec le conseil d'administration. L'entreprise sombra dans l'incohérence, multipliant les modèles sans fil conducteur.

Le retour de Jobs en 1997 remit ces fondamentaux au centre. Gamme épurée, design soigné, intégration totale : ces principes, appliqués à l'iPod puis à l'iPhone, permirent à Apple de conquérir le monde. Cette machine beige de 1984, née de la rencontre entre un réparateur génial et un visionnaire obsessionnel, avait changé notre rapport à la technologie.

HP-UX

C'est au début des années 1980 que Hewlett-Packard décide de se lancer dans l'aventure UNIX. Le marché bouillonne alors d'activité, les constructeurs rivalisent d'ingéniosité pour proposer leur propre variante du système. En 1984, HP sort la première mouture de HP-UX, un UNIX System V retravaillé qui tourne exclusivement sur les serveurs HP 9000 des séries 200, 300 et 400. Ces machines embarquent le processeur Motorola 68000, déjà éprouvé dans d'autres contextes.

HP ne se contente pas d'une simple adaptation d'UNIX. Les ingénieurs introduisent d'emblée les listes de contrôle d'accès, une approche plus fine que le traditionnel système de permissions UNIX. Un gestionnaire de volumes logiques fait aussi partie de l'équipement de base, simplifiant la vie des administrateurs confrontés aux problèmes de stockage. Ces ajouts ne relèvent pas du gadget : ils répondent aux besoins concrets des utilisateurs professionnels.

L'évolution suit le rythme du matériel HP. La version 5.0 de 1985 s'étend à l'Integral PC, puis aux séries 200/300 et 500. HP commence à unifier son approche, cherchant une cohérence entre ses différentes gammes. Cette stratégie porte ses fruits quand arrive l'architecture PA-RISC vers la fin de la décennie.

PA-RISC, ces processeurs RISC développés en interne par HP, vont orienter durablement l'évolution technique du système. En 1988, HP jongle avec deux lignes de développement : les versions 3.x pour les séries 600/800 et 6.x pour la série 300. Cette situation reflète la transition délicate entre les anciennes architectures et la nouvelle génération PA-RISC.

L'unification se concrétise en 1990 avec HP-UX 7.x, qui couvre les séries 300/400 et 600/700/800. Les utilisateurs y gagnent en simplicité, les administrateurs en cohérence. Un an plus tard, la version 8.x apporte les bibliothèques partagées, améliorant sensiblement les performances et l'usage de la mémoire. La version 9.x de 1992 enrichit l'arsenal avec le System Administration Manager et le Logical Volume Manager, deux outils qui modernisent l'administration système.

En 1995 sort HP-UX 10.0. Cette version unifie les séries 700 et 800 tout en adoptant la structure de répertoires SVR4, se rapprochant des standards UNIX du moment. L'année suivante, HP-UX 10.20 franchit le cap du 64 bits avec les processeurs PA-RISC 2.0. Une prouesse technique qui ouvre de nouveaux horizons.

HP-UX 11.00 réussit en 1997 un tour de force remarquable : supporter

l'adressage 64 bits sans abandonner les applications 32 bits. Cette rétrocompatibilité évite aux entreprises la rupture brutale d'une migration. Trois ans plus tard, HP-UX 11i v1 (11.11) introduit les environnements d'exploitation, personnalisant le système selon les besoins spécifiques de chaque type d'usage.

L'architecture connaît un nouveau bouleversement en 2001 avec HP-UX 11i v1.5 (11.20), qui intègre le support Itanium d'Intel. Cette décision stratégique accompagne la transition progressive de PA-RISC vers les processeurs Intel. HP-UX 11i v2 (11.23) de 2003 renforce la virtualisation, tandis que la version 11i v3 (11.31) de 2007 ajoute le *multipathing* natif et peaufine encore les capacités virtuelles.

Au fil des versions, HP-UX se taille une solide réputation dans les services informatiques, l'édition logicielle et la finance. Le système de fichiers VxFS et le gestionnaire LVM sont des références incontournables. Les administrateurs système apprécient leur souplesse et leur efficacité pour gérer le stockage, surtout dans les environnements complexes.

Point fort constant, HP intègre progressivement la sécurité et la détection d'intrusion directement dans le noyau, développe le partitionnement sécurisé et met en place le contrôle d'accès par rôles. Ces éléments hissent HP-UX parmi les UNIX les plus sûrs du marché, un atout décisif pour les applications sensibles.

La virtualisation occupe une place croissante dans l'évolution du système. Les nPartitions arrivent d'abord, suivies des vPars (Virtual Partitions), et enfin des Integrity Virtual Machines. Ces technologies permettent aux entreprises d'optimiser leurs ressources matérielles et de consolider leurs serveurs. Le passage de PA-RISC à Itanium demande un effort technique considérable, mais HP préserve habilement la compatibilité applicative.

Le clustering avec HP Serviceguard, l'amélioration continue des outils d'administration maintiennent HP-UX dans la course face aux exigences changeantes des organisations. Mais Linux gagne du terrain et le cloud computing modifie les règles du jeu. La position de HP-UX sur le marché se trouve progressivement remise en question.

En 2017, HP annonce sa feuille de route jusqu'en 2025, rassurant sa base d'utilisateurs fidèles. Cette annonce répond aux inquiétudes des entreprises qui s'appuient sur HP-UX pour leurs opérations critiques. Le vieillissement du matériel compatible pose toutefois problème : les serveurs HP 9000 et Integrity sont difficiles à maintenir, les pièces détachées se raréfient.

Face à cela, de nouvelles approches émergent. L'émulation matérielle, notamment avec Charon-PAR de Stromasys, offre une alternative intéressante. Les entreprises peuvent continuer à faire tourner leurs applications HP-UX sur du matériel x86 moderne ou dans le cloud, échappant ainsi aux

contraintes du matériel propriétaire vieillissant.

L'histoire de HP-UX témoigne d'une adaptation constante aux besoins des entreprises. Ses innovations en gestion des volumes logiques, sécurité et virtualisation ont inspiré l'évolution des systèmes d'exploitation professionnels. Sa réputation de fiabilité et de robustesse en a fait l'outil de choix pour de nombreuses applications critiques, de la finance à la santé en passant par l'industrie.

NFS

En 1985, Sun Microsystems publie une technologie qui va transformer la manière dont les ordinateurs partagent leurs fichiers à travers les réseaux : le Network File System, plus connu sous l'acronyme NFS. À cette époque, l'idée de faire dialoguer des machines distantes pour accéder à des données communes relevait du défi technique. Pourtant, cette innovation allait s'imposer comme l'une des pierres angulaires des systèmes distribués.

L'histoire commence avec SunOS 2.0, qui intègre la première version publique du protocole, NFSv2. Mais derrière cette sortie se cache une vision stratégique audacieuse de Sun. Là où d'autres entreprises auraient gardé leur technologie sous clé, Sun fait le pari de l'ouverture. Le protocole NFS ne définit que les formats des messages échangés entre clients et serveurs, laissant chacun libre de mettre en œuvre sa propre solution. Cette décision engendre un écosystème foisonnant : NetApp, EMC, IBM et bien d'autres se lancent dans la course, créant leurs propres serveurs NFS tout en respectant l'interopérabilité.

Cette stratégie tranche avec les approches précédentes. Le Remote File System de SVR3, par exemple, établissait une correspondance directe entre les appels système du client et ceux du serveur. L'approche fonctionnait, certes, mais révélait ses limites dès qu'un serveur tombait en panne : les clients se retrouvaient souvent bloqués, contraints de redémarrer pour reprendre leur activité.

Les ingénieurs de Sun prennent une direction radicalement différente. Ils conçoivent NFS comme un système « sans état », où le serveur ne conserve aucune trace des opérations en cours. Chaque requête arrive avec toutes les informations nécessaires à son traitement. Cette architecture, contre-intuitive au premier regard, recèle une élégance redoutable : si le serveur s'arrête brutalement, les clients n'ont qu'à renvoyer leurs requêtes une fois qu'il a redémarré. Pas de reconstruction d'état complexe, pas de synchronisation délicate.

Au cœur de cette mécanique se trouve le « file handle », véritable carte d'identité de chaque fichier sur le serveur. Dans NFSv2, ce descripteur rassemble un identifiant de volume, un numéro d'inode et un numéro de génération. Ce dernier élément évite les confusions quand le système réutilise des numéros d'inode : chaque fichier conserve ainsi son identité unique, même après suppression et recréation.

Bien sûr, interroger le serveur à chaque accès fichier aurait généré un trafic réseau insoutenable. NFS intègre donc des mécanismes de cache côté client, stockant temporairement les données et métadonnées consultées. Cette optimisation soulève immédiatement une question épineuse : comment garantir que tous les clients voient les mêmes informations ? La réponse tient dans le modèle « close-to-open ». Les modifications remontent vers le serveur à la fermeture des fichiers, tandis que les clients vérifient l'état des données à chaque ouverture. Cette approche privilégie la performance tout en maintenant une cohérence raisonnable.

Le protocole va connaître plusieurs métamorphoses au fil des années. En 1995, NFSv3 arrive avec des améliorations de performance, notamment une meilleure gestion des écritures asynchrones. Mais c'est NFSv4, standardisé en 2000, qui bouleverse véritablement l'architecture. Paradoxalement, cette version abandonne le principe sans état au profit d'un protocole orienté état, introduit des délégations de cache et améliore considérablement la compatibilité Internet.

La sécurité constituait le talon d'Achille des premières versions. Le système d'authentification reposait sur une confiance aveugle entre clients et serveur, utilisant simplement les identifiants numériques UNIX. Cette naïveté fut progressivement corrigée par l'intégration de Kerberos, puis par les mécanismes plus sophistiqués de NFSv4.

Mais le développement de NFS a nécessité la création de l'interface Virtual File System, une couche d'abstraction permettant la coexistence de différents systèmes de fichiers dans le noyau UNIX. Cette innovation, toujours présente dans les systèmes modernes, a grandement simplifié l'intégration de nouveaux systèmes de fichiers.

Les années 1990 voient naître une concurrence féroce. Microsoft pousse son Server Message Block, rebaptisé plus tard Common Internet File System, qui s'impose naturellement dans l'univers Windows. Des systèmes plus ambitieux comme Andrew File System ou, plus récemment, Lustre pour les supercalculateurs, proposent des fonctionnalités avancées pour des niches spécifiques.

Pourtant, NFS résiste à ces assauts. Sa simplicité relative, sa robustesse éprouvée et sa disponibilité quasi-universelle en font un choix de raison pour nombre d'organisations. Les constructeurs de stockage continuent d'inno-

ver autour du protocole. NetApp, notamment, développe des optimisations matérielles pour accélérer les écritures synchrones, historiquement le point faible de NFS.

L'héritage technique de NFS alimente encore la conception des systèmes distribués. Le principe architectural sans état, initialement controversé, inspire désormais de nombreux protocoles réseau. Les leçons apprises sur la cohérence de cache alimentent toujours la recherche académique et industrielle.

Alors que le stockage cloud redessine les contours de l'informatique, les défis que NFS a relevés demeurent d'actualité. Si les solutions évoluent, la distribution, la cohérence et la tolérance aux pannes sont des problématiques qui traversent le temps. L'aventure NFS démontre qu'une approche pragmatique privilégiant la robustesse sur la sophistication peut durer. Dans un domaine où les changements technologiques se succèdent à un rythme effréné, cette longévité force le respect.

POP3

En 1984, les communications électroniques connaissaient un problème que nous avons du mal à imaginer. Les messages arrivaient bien sur les serveurs grâce au protocole SMTP, mais comment les récupérer quand on ne disposait pas d'une connexion permanente à Internet ? Cette question préoccupait les ingénieurs de l'époque, qui cherchaient une solution pratique pour des utilisateurs encore habitués aux connexions téléphoniques intermittentes.

Le protocole POP (Post Office Protocol) apparut pour répondre à ce besoin. La première version vit le jour dès 1984. L'idée était simple : créer un système de « boîte aux lettres » électronique où les messages patientent jusqu'à ce que l'utilisateur vienne les chercher. POP2 suivit en 1985 avec quelques améliorations, et enfin POP3 arriva en 1988 en introduisant des mécanismes d'extension et des systèmes d'authentification plus solides.

Le fonctionnement de POP3 témoigne d'une simplicité qui primait sur la sophistication. Le client se connecte au serveur via le port TCP 110, s'identifie et utilise une poignée de commandes basiques. STAT pour connaître le nombre de messages, LIST pour voir leur taille, RETR pour en récupérer un, QUIT pour partir. Rien de plus, rien de moins. Cette économie de moyens a d'ailleurs largement contribué à son succès auprès des développeurs.

L'intégration de POP3 dans Microsoft Outlook et d'autres clients de messagerie grand public a scellé son destin. Le protocole s'est répandu comme une traînée de poudre, devenant la solution de référence pour les particuliers et les petites organisations. Sa logique correspondait parfaitement aux habitudes de l'époque : on relevait son courrier sur son ordinateur personnel, comme on vidait sa boîte aux lettres physique.

Mais cette approche cache un piège qui allait se révéler problématique. Par défaut, POP3 supprime les messages du serveur après les avoir téléchargés. Une fois rapatriés sur l'ordinateur du salon, impossible de les consulter depuis le bureau ou, plus tard, depuis un téléphone portable. Cette limitation, anodine dans les années 1990, est devenue un véritable handicap avec l'explosion de l'informatique mobile.

Face à ces contraintes, le protocole IMAP s'est imposé comme l'alternative moderne. Là où POP3 vide la boîte aux lettres du serveur, IMAP la conserve intacte. Les messages restent accessibles depuis n'importe quel appareil, organisés en dossiers, consultables hors ligne si nécessaire. IMAP propose aussi des fonctions de recherche côté serveur et le téléchargement partiel des gros messages.

Pourtant, POP3 refuse de disparaître. Sa légèreté technique reste un atout indéniable : il consomme peu de ressources serveur et s'utilise sans difficulté. Pour certains usages spécifiques, notamment quand l'utilisateur travaille principalement sur un seul appareil, POP3 conserve ses avantages. Les messages stockés localement se consultent sans connexion Internet et n'encombrent pas les serveurs.

Cette résistance illustre un phénomène intéressant dans l'évolution technologique. Deux philosophies s'affrontent ici : POP3 fait de l'ordinateur personnel le centre de l'univers numérique, tandis qu'IMAP privilégie une vision centralisée où le serveur est le référentiel unique. Chaque approche répond à des besoins différents et influence directement l'expérience utilisateur.

L'histoire récente a plutôt donné raison à IMAP. L'augmentation des capacités de stockage, la démocratisation des connexions haut débit permanentes et surtout l'usage simultané de multiples appareils ont rendu la synchronisation indispensable. Qui accepterait aujourd'hui de ne pouvoir consulter ses mails que depuis son ordinateur portable ?

Le protocole n'est pas resté figé pour autant. Des améliorations sécuritaires ont vu le jour, comme le mécanisme d'authentification APOP qui renforce la protection des échanges. Ces évolutions témoignent d'une capacité d'adaptation aux exigences contemporaines.

La standardisation par l'IETF a joué un rôle décisif dans la pérennité de POP3. Cette normalisation a garanti son interopérabilité et facilité son

intégration dans une multitude de logiciels. Paradoxalement, c'est peut-être cette ubiquité qui explique sa survie : remplacer un standard si largement répandu demande des efforts considérables.

L'aventure de POP3 révèle qu'une solution technique n'a pas besoin d'être la plus avancée pour durer, elle doit surtout répondre efficacement à un besoin concret. POP3 a brillamment résolu le problème de la messagerie électronique pour les connexions intermittentes des années 1980. Que ce problème ait depuis largement disparu n'efface pas cette réussite initiale.

Cette persistance s'explique aussi par des facteurs très humains. La familiarité des utilisateurs avec un outil, la résistance au changement, la préférence pour des solutions éprouvées constituent autant de freins à l'adoption de nouvelles technologies. Les administrateurs système le savent bien : migrer vers un nouveau protocole demande du temps, de l'argent et de la formation.

Le parcours de POP3 s'inscrit dans une dynamique plus large de l'évolution d'Internet. Les protocoles naissent, évoluent, coexistent et parfois disparaissent selon des logiques qui dépassent les seules considérations techniques. L'histoire de la messagerie électronique montre comment différentes approches peuvent survivre côte à côte, chacune conservant sa niche d'utilisation.

Cette coexistence n'est pas un accident mais une caractéristique des écosystèmes technologiques matures. POP3 et IMAP répondent à des philosophies différentes de la gestion des données personnelles. L'un privilégie l'autonomie et le contrôle local, l'autre mise sur la centralisation et l'accessibilité universelle. Les deux visions restent légitimes selon le contexte d'usage.

Comprendre l'histoire de POP3, c'est saisir comment les technologies s'enracinent dans leur temps tout en gardant une capacité d'adaptation limitée. C'est aussi mesurer l'importance des choix de conception initiaux, qui peuvent déterminer le succès ou l'obsolescence d'un standard sur plusieurs décennies.

IBM PCjr

En 1983, IBM trône sur le marché professionnel avec son PC. Fort de ce succès, le géant américain lorgne vers l'informatique domestique, territoire jusque-là dominé par des machines comme l'Apple II ou le Commodore 64. L'idée germe d'adapter le PC pour les familles, en créant une version

allégée baptisée « Peanut » en interne. Les rumeurs se multiplient dans la presse spécialisée pendant des mois, jusqu'à l'annonce officielle du PCjr le 1er novembre 1983.

L'attente est énorme. IBM dispose alors d'un budget de recherche qui dépasse celui de certains États, et son nom seul suffit à déclencher l'engouement du public. La machine reprend le processeur Intel 8088 du PC et conserve une compatibilité logicielle substantielle avec son grand frère. Cette filiation assumée vise deux objectifs : rassurer les acheteurs sur la pérennité de leur investissement et préserver les ventes du PC professionnel.

Deux versions voient le jour. Le modèle de base, affiché à 669 dollars, embarque 64 Ko de mémoire. Le modèle étendu grimpe à 1 269 dollars avec ses 128 Ko et son lecteur de disquette intégré. Ces prix placent d'emblée la machine dans le haut du panier, bien au-dessus de la concurrence directe.

La grande originalité du PCjr réside dans son clavier sans fil, une première mondiale pour un ordinateur personnel. La liaison infrarouge rend la machine utilisable jusqu'à 6 mètres de distance, transformant le salon en bureau improvisé. Mais cette innovation s'accompagne d'un choix discutable : des touches plates inspirées des calculatrices, dépourvues d'inscriptions. Les symboles et lettres apparaissent sur le châssis, entre les touches, compliquant la frappe et fatiguant les yeux.

Côté performances, IBM a doté sa machine de capacités graphiques remarquables. Plusieurs modes d'affichage coexistent, du 320×200 pixels en 4 couleurs au 640×200 en deux teintes. Le modèle étendu grimpe à 16 couleurs simultanées. Le son n'est pas en reste avec un circuit sophistiqué produisant trois voies de synthèse sur 7 octaves, complétées par un générateur de bruit blanc.

La commercialisation débute en janvier 1984, mais les livraisons restent confidentielles jusqu'en mars. IBM déploie une artillerie marketing sans précédent. Charlie Chaplin, déjà égérie du PC, revient dans les publicités télévisées. Plus de cent ouvrages paraissent avant la sortie effective, et quatre magazines spécialisés voient le jour. Rarement une machine aura bénéficié d'un tel battage médiatique.

Pourtant, les ventes peinent à décoller. Les premiers tests révèlent des failles béantes. Avec sa vitesse d'exécution divisée par deux par rapport au PC, le PCjr déçoit les utilisateurs habitués aux performances de son aîné. La mémoire, plafonnée à 128 Ko, limite sévèrement les possibilités d'évolution. L'architecture interne rend impossible l'ajout d'un second lecteur de disquette ou un disque dur. Les connecteurs propriétaires obligent à acheter des adaptateurs hors de prix pour raccorder des périphériques standards.

Face à ces critiques, IBM tente de redresser la barre en juillet 1984. Un nouveau clavier, plus conventionnel, remplace le modèle à touches plates.

Les prix baissent sensiblement. Une campagne promotionnelle d'envergure accompagne les fêtes de fin d'année 1984, propulsant temporairement les ventes à 17% du marché domestique. Mais l'embellie est de courte durée : dès janvier 1985, ce chiffre rechute à 4%.

Le couperet tombe le 19 mars 1985. IBM annonce l'arrêt de la production, quinze mois seulement après le lancement commercial. Entre 100 000 et 350 000 machines invendues s'entassent dans les entrepôts. La firme brade les stocks, allant jusqu'à les offrir aux écoles achetant des PC. La facture s'élève à 45 millions de dollars de pertes.

Paradoxalement, cet échec commercial masque quelques réussites durables. Sierra On-Line développe spécialement pour le PCjr le jeu *King's Quest*, qui révolutionne le genre de l'aventure graphique et lance une franchise légendaire. Tandy Computer s'inspire directement de l'expérience IBM pour créer son modèle 1000, corrigeant au passage les défauts majeurs : clavier professionnel, performances équivalentes au PC, possibilités d'extension. Ce Tandy 1000 rencontre un succès considérable et prolonge l'héritage technique du PCjr.

Une communauté d'amateurs entretient la mémoire de cette machine singulière. Des extensions comme la carte jrIDE ajoutent un disque dur et 640 Ko de mémoire aux exemplaires survivants. L'émulateur DOSBox ressuscite son catalogue logiciel. Au Japon, IBM avait tenté une variante appelée JX, développée avec Matsushita, mais ce projet concurrent a sombré encore plus avec seulement 8 000 exemplaires écoulés.

L'histoire du PCjr illustre les pièges d'une stratégie de dérivation mal maîtrisée. En bridant volontairement les capacités de leur machine domestique pour protéger les ventes du PC professionnel, les ingénieurs d'IBM ont créé un produit bancal, trop cher et trop limité face à une concurrence agile et inventive. La distribution exclusive via les réseaux professionnels, inadaptée au marché familial, a achevé de condamner une aventure qui marqua la fin des ambitions d'IBM sur l'informatique domestique.

GNU Emacs

En 1974, dans les couloirs du laboratoire d'intelligence artificielle du MIT, Richard Stallman s'attaque à un problème qui agace tous les programmeurs de l'époque : l'éditeur de texte TECO, pourtant puissant, reste d'un maniement laborieux. L'idée lui vient d'améliorer cet outil en s'inspirant de l'éditeur `E` développé à Stanford. Il ajoute un affichage en temps réel et repense le système de commandes.

Au début, cette amélioration ne change pas grand-chose à TECO. Les utilisateurs lancent le mode d'édition visuelle avec Control-R, modifient leur texte, puis en sortent pour d'autres tâches. Mais une fonctionnalité va tout transformer : la possibilité de redéfinir les commandes pour appeler des programmes écrits directement en TECO. Cette liberté déchaîne la créativité des utilisateurs qui développent leurs propres systèmes de commandes. TECMAC, MACROS, RMODE, TMACS, Russ-mode ou DOC fleurissent dans la communauté. L'usage évolue pour que la sortie du mode d'édition soit superflue, les nouveaux arrivants n'apprennent plus à manipuler TECO directement.

Pourtant, TECO montre ses limites. Il manque de structures de programmation décentes : pas de fonctions nommées, pas de variables propres. Les premiers éditeurs construits sur cette base sont vite ingérables. En 1976, le système TMACS tente d'ajouter ces fonctionnalités manquantes. L'expérience fonctionne, mais les performances souffrent. Stallman observe, analyse et décide de créer EMACS.

Son approche combine deux défis : enrichir TECO des bibliothèques et de l'auto-documentation nécessaires à une programmation lisible, tout en concevant un ensemble de commandes d'édition totalement nouveau. Il étudie les nombreux éditeurs basés sur TECO et cherche à minimiser le nombre de touches pour les opérations courantes. Fin 1976, le premier EMACS fonctionne.

Le développement se poursuit sur le PDP-10 de Digital Equipment Corporation, sous le système ITS du MIT. La plupart du nouveau code s'écrit en TECO, Stallman n'ajoutant au langage que le strict minimum : des optimisations pour les boucles de recherche, l'analyse d'expressions symboliques, et quelques nouvelles interfaces d'entrée-sortie. En 1977, l'intérêt grandit au-delà des murs du MIT. Mike McMahon de SRI International adapte EMACS au système Twenex de Digital. Une centaine de sites l'utilisent déjà.

Le terme « Emacs » se généralise alors, désignant toute une famille d'éditeurs extensibles. L'une des versions les plus marquantes naît sous UNIX, développée par James Gosling. Cette implémentation apporte deux nouveautés remarquables. D'abord, elle exploite les capacités de gestion des processus d'UNIX : filtrer du texte à travers des programmes externes, exécuter différents outils dans des fenêtres séparées. Ensuite, elle introduit MLisp (Mock Lisp), un langage d'extension indépendant du C utilisé pour l'implémentation.

Cette séparation entre langage d'implémentation et langage d'extension révolutionne l'usage. Avec MLisp, simple et élégant, les utilisateurs créent des extensions sans compilation ni édition de liens. Le cycle de dévelop-

pement s'accélère, encourageant les petites améliorations qui font toute la différence dans une interface utilisateur. Le langage offre une vision abstraite du texte, des chaînes, des processus et des fenêtres.

En 1984, Stallman lance GNU Emacs dans le cadre de son projet GNU. Cette fois, il choisit le vrai Lisp comme langage d'extension, plus puissant et cohérent que MLisp. GNU Emacs s'impose. Une communauté active développe des extensions pour tout : courrier électronique, forums de discussion, édition spécialisée selon les langages de programmation, interfaces de débogage.

Emacs prouve qu'un logiciel extensible et personnalisable peut s'adapter aux besoins de chacun. Ce modèle inspire nombre de développements ultérieurs : un cœur stable combiné à un langage d'extension puissant. Les environnements de développement modernes reprennent cette approche, intégrant des langages de programmation, la personnalisation d'interface et l'automatisation de tâches.

L'héritage d'Emacs se retrouve dans son auto-documentation, qui donne accès aux explications des commandes et fonctions depuis l'éditeur, se généralise pour l'aide en ligne des logiciels interactifs.

GNU Emacs continue d'évoluer, fidèle à sa philosophie d'origine. Dans un monde où les technologies sont obsolètes en quelques années, sa longévité témoigne de la justesse de ses principes : extensibilité, puissance et flexibilité. Il s'adapte aux besoins contemporains sans renier ce qui a fait son succès.

Microsoft Paint

Certains logiciels marquent les esprits par leur sophistication technique ou leurs prouesses innovantes. Microsoft Paint ne fait pas partie de cette catégorie. Pourtant, depuis 1985, cette modeste application de dessin accompagne des millions d'utilisateurs Windows et a façonné, presque malgré elle, toute une esthétique numérique.

L'aventure commence dans les années 1970, quand deux philosophies s'affrontent dans le domaine du graphisme informatique : les programmes vectoriels qui travaillent avec des objets géométriques précis, et les logiciels de peinture numérique qui manipulent directement les pixels. Dick Shoup, chez Xerox PARC, développe SuperPaint en 1973, un système qui capture les mouvements de la main sur une grille de pixels. Cette approche bitmap, moins rigoureuse que le vectoriel, offre une spontanéité qui rapproche

l'ordinateur du geste artistique traditionnel.

Microsoft découvre cette voie quand Apple triomphe avec MacPaint sur ses premiers Macintosh. En 1985, Windows 1.0 inclut Windows Paint, codé par Dan McCabe. Le programme propose 24 outils dans une interface spartiate comme le crayon, le pinceau, les formes géométriques, et aussi quelques fonctions avancées comme les courbes de Bézier ou le dessin isométrique. Mais ces ambitions se heurtent aux réalités techniques de l'époque. Les ordinateurs peinent, les écrans manquent de résolution, et Windows 1.0 ne convainc pas vraiment.

Les années suivantes, Microsoft se détourne de Paint. L'entreprise collabore avec IBM sur le Presentation Manager, un projet de système d'exploitation graphique qui finalement avorte. Paint stagne dans Windows 2.0, victime de cette réorientation stratégique qui ne mène nulle part.

Le réveil arrive avec Windows 3.0 en 1990. Microsoft abandonne l'idée de développer toutes ses applications en interne et fait appel aux éditeurs existants. Microsoft Paintbrush remplace Windows Paint, basé sur PC Paintbrush de ZSoft, créé par Mark Zachmann. La métamorphose est spectaculaire : la couleur fait son apparition avec 256 teintes disponibles, la palette d'outils migre sur le côté gauche, et les couleurs s'alignent en bas de l'écran. L'interface gagne en cohérence et en lisibilité.

Windows 95 transforme définitivement Paint. Rebaptisé Microsoft Paint (ou MS Paint pour les intimes), il est un composant standard de chaque installation Windows. Cette omniprésence coïncide avec l'explosion d'Internet, qui mute d'un média textuel vers un univers visuel. Entre 1995 et 2000, près d'un Américain sur deux découvre Internet, et Paint est leur premier contact avec la création graphique numérique.

L'outil développe alors sa signature esthétique particulière. L'utilisation exclusive de la souris, combinée à l'approche bitmap, produit des lignes tremblotantes, des contours approximatifs, des zones de couleur aux bordures franches. Ces « défauts » techniques créent un langage visuel reconnaissable entre mille. Là où d'autres logiciels visent la perfection, Paint assume ses imperfections et les transforme en caractéristiques expressives.

Curieusement, Microsoft laisse Paint dans cet état pendant plus de dix ans. De 1995 à 2007, les fonctionnalités évoluent peu. Cette inertie s'explique par un changement de paradigme : Windows n'a plus besoin de séduire par ses applications intégrées. Paint remplit son rôle d'outil basique pour les tâches simples, sans prétendre concurrencer Photoshop ou Illustrator.

Dans les années 2000, Internet voit naître les premiers mèmes visuels. Les communautés de Something Awful ou 4chan adoptent le côté esthétique « amateur » de Paint comme un étendard. Les Rage Comics de 2008

incarnent parfaitement ce mouvement : des bandes dessinées rudimentaires aux personnages expressifs, entièrement réalisées avec Paint. L'imprécision est volontaire, l'amateurisme revendiqué.

Microsoft tente bien de moderniser l'expérience en 2012 avec Fresh Paint, une application tactile qui simule la peinture traditionnelle. Mais cette version lisse et sophistiquée masque sa nature numérique au lieu de l'assumer. Pendant ce temps, des outils comme Rage Maker prolifèrent pour faciliter la création de mèmes dans le style Paint authentique.

L'histoire de Paint révèle comment les contraintes façonnent la créativité. Ses limitations techniques, qui auraient pu constituer des handicaps, sont presque devenues des atouts distinctifs. Paint n'a jamais figuré dans les palmarès d'innovation, mais il a sa propre empreinte sur la culture visuelle.

Cette trajectoire éclaire l'importance des technologies « ordinaires » dans l'évolution informatique. Les innovations spectaculaires accaparent l'attention, pendant que des outils modestes structurent discrètement nos pratiques quotidiennes. Paint survit dans Windows, témoin d'une période où créer numériquement devenait accessible au plus grand nombre, avec ses maladresses et sa poésie particulière.

X Window System

En 1984, au MIT, deux projets apparemment sans rapport allaient converger vers une transformation majeure dans l'affichage graphique des systèmes UNIX. D'un côté, le système Argus du Laboratory for Computer Science cherchait un environnement de débogage pour ses applications distribuées. De l'autre, le projet Athena devait gérer des milliers de stations de travail équipées d'écrans bitmap. Face à la diversité du matériel – machines Digital VSlOO, postes IBM et autres architectures hétérogènes – les ingénieurs comprirent qu'il fallait repenser entièrement l'approche de l'affichage graphique.

L'inspiration vint de Stanford, où Paul Asente et Brian Reid avaient créé le système W comme alternative à VGTS. W autorisait déjà l'accès transparent au réseau pour l'affichage, grâce au mécanisme de communication synchrone du système V. Ce système proposait des fenêtres textuelles pour l'émulation de terminaux ASCII et des fenêtres graphiques construites sur un mécanisme simple de listes d'affichage. Asente et Chris Kent adaptèrent W pour les machines VSlOO aux laboratoires de recherche Digital, mais quelques jours d'expérimentation révélèrent les limites de cette approche :

un système de fenêtres hiérarchique accessible via réseau serait bien plus efficace, à condition de ne pas le limiter à des modes d'application spécifiques.

X Window System naît de cette réflexion. Son architecture client-serveur inverse la logique habituelle : le serveur contrôle l'affichage physique tandis que les applications clientes dialoguent avec lui via un protocole asynchrone transporté sur un flux d'octets fiable. Cette inversion conceptuelle change tout. Il est désormais possible d'exécuter un programme sur n'importe quelle machine du réseau et utiliser n'importe quel écran, sans se préoccuper de l'architecture matérielle ou du système d'exploitation sous-jacent. Les performances restent remarquables : une VAXStation-II/GPX affiche 19 500 caractères par seconde et trace 3 500 vecteurs courts par seconde, que ce soit en local ou via le réseau.

Parmi les innovations introduites par le système à l'époque, les fenêtres s'organisent en hiérarchie arbitraire, se redimensionnent et se chevauchent librement. Le serveur encapsule toutes les dépendances matérielles, rendant le protocole de communication totalement indépendant des périphériques. Une application fonctionne sans modification sur un terminal monochrome ou un écran couleur sophistiqué. Le protocole minimise la latence réseau grâce aux requêtes asynchrones et à une gestion fine des événements.

La gestion des couleurs révèle la sophistication technique de X. Le modèle abstrait proposé s'adapte aux écrans monochromes comme aux affichages pseudo-couleur les plus avancés. Plusieurs applications peuvent partager une table de couleurs commune tout en conservant leurs propres espaces colorimétriques. Cette approche autorise la coexistence harmonieuse d'applications graphiques multiples sans conflit de ressources.

En 1987, DEC intègre X dans ses produits commerciaux, validant l'approche technique. Le MIT distribue le code source complet sans restriction, accélérant l'adoption par les universités et les entreprises. Cette stratégie open source avant la lettre transforme X en standard de fait. Chercheurs académiques et ingénieurs commerciaux collaborent au développement, créant un écosystème riche et diversifié. Le système se retrouve porté sur plus de sept architectures matérielles différentes et seize types d'affichages.

Les années 1990 voient naître GNOME et KDE, environnements de bureau qui démontrent la robustesse de X11. Mais les limites apparaissent : le système de polices s'avère insuffisant pour les besoins modernes, les capacités graphiques 2D demandent des améliorations, l'accessibilité pour les malvoyants reste difficile à mettre en œuvre.

Xft résout les problèmes typographiques en autorisant l'accès direct aux fichiers de polices et en introduisant l'anticrénelage. Fontconfig sépare la

gestion des caractères du système de fenêtrage, simplifiant l'internationalisation et l'installation de nouvelles polices. Cairo apporte des fonctionnalités graphiques avancées : courbes de Bézier, composition d'images translucides, rendu vectoriel haute qualité. Ces évolutions s'intègrent naturellement grâce à l'architecture modulaire de X.

Le protocole Composite transforme la gestion visuelle en autorisant fenêtres translucides et effets sophistiqués. XEvIE améliore l'accessibilité en interceptant et transformant les événements d'entrée. Ces extensions maintiennent X pertinent face aux exigences croissantes des interfaces modernes, sans rompre la compatibilité avec l'existant.

La sécurité ne fut jamais négligée. X implémente des mécanismes d'authentification et d'autorisation sophistiqués dès sa conception. Les terminaux autonomes demandent des services de connexion à des hôtes distants de manière sécurisée avec X Display Manager Control Protocol. Les administrateurs peuvent centraliser la gestion des accès tout en préservant l'expérience utilisateur des terminaux traditionnels.

Le modèle client-serveur de X et sa conception modulaire inspirent les architectures logicielles contemporaines. En établissant un standard pour les systèmes distribués graphiques, X façonne l'évolution des interfaces utilisateur sur Linux et les systèmes apparentés.

En 2004, la fondation X.org reprend la gouvernance du projet et poursuit le développement. Les innovations continuent : support des écrans haute résolution, accélération matérielle via OpenGL, adaptation aux technologies d'affichage émergentes. Wayland apparaît comme alternative pour certains cas d'usage, mais X Window System demeure en 2024 le système de fenêtrage de référence sur de nombreuses plateformes UNIX et Linux.

Les principes architecturaux de X – modularité, extensibilité, indépendance matérielle – conservent leur pertinence dans le développement des interfaces contemporaines. Plus qu'une technologie, X illustre l'efficacité de l'approche collaborative où universités, entreprises et communautés coordonnent leurs efforts pour créer une infrastructure durable.

Intel 80386

Quand Intel lance le 80386 en octobre 1985, personne ne s'attend vraiment à ce que ce processeur révolutionne l'informatique. Dans les couloirs de l'entreprise, on le considère plutôt comme un produit de transition, une solution d'attente jusqu'à l'arrivée du véritable projet phare : l'iAPX 432.

Ce processeur ambitieux accumule pourtant les retards, et le 386 doit combler le vide. L'histoire retiendra que c'est finalement ce « bouche-trou » qui transformera durablement le paysage informatique.

Le projet démarre début 1982 sous la direction de John Crawford, l'architecte en chef. Son équipe d'une dizaine d'ingénieurs fait face à un dilemme cornélien : faut-il rompre avec le passé pour créer un processeur avant-gardiste ou maintenir la compatibilité avec l'existant au risque de brider les performances ? Les discussions se succèdent. Les clients d'Intel sont clairs : ils ne veulent pas réécrire leurs logiciels. La compatibilité totale au niveau du code objet s'impose donc naturellement.

Cette contrainte n'empêche pas l'équipe de Crawford de voir grand. Le 386 abandonne définitivement l'architecture 16 bits de ses prédécesseurs pour passer au 32 bits. Registres, unité arithmétique et logique, bus internes : tout passe à la nouvelle dimension. Le processeur peut désormais adresser directement 4 gigaoctets de mémoire physique, une capacité qui paraît alors gigantesque. Les systèmes d'exploitation modernes comme UNIX ou OS/2 feront de la gestion de la mémoire virtuelle avec pagination leur réalité.

La fabrication du 386 nécessite des innovations techniques de premier plan. Intel abandonne la technologie NMOS au profit du CMOS, réduisant la consommation électrique et améliorant la qualité des signaux. Mais c'est surtout le processus de fabrication à deux niveaux de métal qui pose des problèmes. Cette technologie nouvelle génère des défauts de production importants. Les ingénieurs découvrent que la « zone interdite » entre les pistes métalliques complique énormément la production. Les rendements chutent, les coûts explosent, et l'équipe doit sans cesse ajuster les paramètres.

Les moyens de développement restent modestes. Une dizaine d'ingénieurs travaillent avec des outils souvent artisanaux. L'équipe prend une décision audacieuse en adoptant UNIX comme système d'exploitation de développement, malgré l'absence d'autorisation officielle. Cette transgression se révèle payante. L'automatisation du placement et du routage des composants, réalisée grâce à un logiciel développé par un étudiant de Berkeley, accélère considérablement la conception.

Le lancement commercial réserve une surprise. Compaq, et non IBM, sort le premier ordinateur personnel équipé du 386. IBM ne contrôle plus seul l'évolution technologique du marché, c'est une rupture dans cette industrie. Le Compaq Deskpro 386 est une référence qui force IBM à réagir. Intel découvre que la demande dépasse ses prévisions initiales.

Pour répondre à cette demande tout en maîtrisant les coûts, Intel développe le 386SX. Cette version simplifiée conserve l'architecture 32 bits interne mais utilise un bus externe 16 bits. Le compromis séduit les construc-

teurs d'ordinateurs bon marché qui peuvent proposer une architecture moderne à prix réduit. Le 386SX rencontre un succès commercial important, démocratisant l'accès aux capacités 32 bits.

Les performances du 386 évoluent avec la première version cadencée à 16 MHz qui atteint 5 MIPS. Les versions ultérieures sont capables de monter jusqu'à 33 MHz pour 9,9 MIPS. Ces chiffres représentent un bond considérable par rapport aux processeurs précédents. Le 386 transforme l'informatique personnelle en permettant l'exécution fluide d'applications graphiques et de systèmes multitâches.

Le succès du processeur s'étend bien au-delà du marché des PC. Les stations de travail UNIX l'adoptent massivement. L'industrie l'intègre dans des systèmes de contrôle automatisé. La recherche scientifique en fait un composant de référence pour ses équipements. Plus surprenant, le 386 trouve sa place dans l'espace : le télescope Hubble et la sonde SAMPEX emportent des versions durcies du processeur.

Les premiers appareils mobiles sophistiqués adoptent le 386. Le BlackBerry 950 et le Nokia 9000 Communicator intègrent des versions basse consommation du processeur. Cette longévité exceptionnelle témoigne de la robustesse de l'architecture conçue par Crawford et son équipe. Le 386 reste en production pendant des années, bien après l'arrivée de ses successeurs.

L'influence du 386 dépasse son seul succès commercial. Intel adopte suite à cette expérience une stratégie de développement alternant innovations architecturales et optimisations de fabrication. Cette approche, baptisée plus tard « tick-tock », guide encore la stratégie de l'entreprise. Chez Intel, le principe de compatibilité ascendante établi avec le 386 garantit la pérennité des investissements logiciels, un véritable dogme.

Le 386 illustre parfaitement comment l'histoire de l'informatique procède parfois par accidents heureux. Conçu comme une solution temporaire, ce processeur finit par établir les bases de l'informatique moderne. Son héritage perdure dans chaque ordinateur personnel, rappelant qu'en technologie, les solutions pragmatiques l'emportent souvent sur les projets les plus ambitieux.

AppleTalk

Au début des années 1980, Apple Computer se lance dans un défi ambitieux : créer un protocole réseau en parallèle du développement du Ma-

cintosh. L'idée paraît audacieuse, mais elle répond à un besoin concret. Les utilisateurs veulent partager leurs fichiers et leurs imprimantes sans s'embarrasser de configurations techniques complexes. AppleTalk naît de cette volonté de simplifier l'informatique en réseau.

Le protocole cache sa sophistication derrière une interface d'une transparence remarquable. Les ordinateurs communiquent entre eux sans que l'utilisateur ait besoin de comprendre les mécanismes sous-jacents. Cette philosophie tranche avec les systèmes de l'époque, souvent rébarbatifs et réservés aux experts. Apple mise sur l'invisibilité des opérations techniques : seul le résultat compte.

Deux versions principales voient le jour. La Phase 1 s'adresse aux petits groupes de travail mais souffre de limitations contraignantes. Impossible de dépasser 135 machines sur un segment, et les réseaux étendus restent hors de portée. Ces restrictions freinent l'expansion d'AppleTalk dans les entreprises. La Phase 2 corrige ces défauts en autorisant jusqu'à 253 machines par segment et en gérant les réseaux étendus. Cette évolution ouvre la voie à des déploiements plus ambitieux.

La flexibilité d'AppleTalk s'exprime dans ses déclinaisons physiques. LocalTalk exploite les paires torsadées propriétaires d'Apple, tandis que TokenTalk s'appuie sur les réseaux Token Ring. EtherTalk tire parti de l'Ethernet, et FDDITalk exploite les fibres optiques FDDI. Cette modularité autorise les transitions technologiques sans refonte complète du système.

L'architecture protocolaire s'organise avec méthode. Le DDP (Datagram Delivery Protocol) transporte les paquets d'un point à un autre. L'AARP (AppleTalk Address Resolution Protocol) établit la correspondance entre adresses logiques et physiques. Le RTMP (Routing Table Maintenance Protocol) maintient les informations de routage à jour. Chaque protocole assume sa fonction sans empiéter sur les autres.

La gestion des noms repose sur des mécanismes ingénieux. Le NBP (Name Binding Protocol) associe des noms compréhensibles aux adresses techniques, tandis que le ZIP (Zone Information Protocol) organise les réseaux en zones logiques. L'AEP (AppleTalk Echo Protocol) vérifie que les machines restent accessibles. L'ATP (AppleTalk Transaction Protocol) garantit la fiabilité des échanges pour les applications critiques comme le partage de fichiers ou l'impression.

L'adressage dynamique constitue l'une des innovations d'AppleTalk. Un identifiant est automatiquement attribué à chaque machine à sa connexion. Cette approche évite les conflits d'adresses tout en simplifiant l'administration. L'adresse combine trois éléments : un numéro de réseau sur 16 bits, un identifiant de nœud sur 8 bits, et un numéro de socket sur 8 bits pour désigner le service concerné.

Le concept de zone révolutionne l'organisation des réseaux. Il est possible de regrouper des machines selon des critères fonctionnels plutôt que géographiques. Le service comptabilité peut ainsi rassembler ses ressources dans une zone unique, même si les ordinateurs sont dispersés dans différents bâtiments. Cette flexibilité séduira de nombreux administrateurs réseau.

La transmission s'effectue sans établissement de connexion préalable. Les paquets voyagent indépendamment vers leur destination. Les protocoles de haut niveau ajoutent ensuite les mécanismes de fiabilité nécessaires selon les besoins. Cette architecture modulaire facilite l'évolution et la maintenance du système.

Cisco enrichit progressivement son intégration d'AppleTalk. Il ajoute le support des différentes versions d'EtherTalk, la compatibilité avec les VLAN, l'intégration aux protocoles WAN comme Frame Relay. Ces extensions repoussent les limites du protocole bien au-delà des intentions initiales d'Apple. L'encapsulation des paquets RTMP sur IP fait aussi transiter AppleTalk sur Internet.

Parent pauvre d'AppleTalk, le protocole délègue la sécurité aux applications, choix discutable à l'heure actuelle mais cohérent avec la philosophie des années 1980. Cisco tente de combler cette lacune avec ses listes de distribution, mécanisme rudimentaire mais utile pour contrôler la propagation des informations de routage.

L'implémentation Cisco présente une particularité surprenante : elle refuse de transmettre certains paquets avec des adresses locales identiques. Cette restriction, contraire aux spécifications d'Apple, vise à préserver l'intégrité des tables d'adresses. Le constructeur privilégie la stabilité du réseau à la conformité stricte aux standards.

Le déclin d'AppleTalk s'amorce dans les années 1990. TCP/IP s'impose progressivement comme le standard universel des communications réseau. Sa nature ouverte et son adoption par Internet scellent le sort des protocoles propriétaires. En 2009, Mac OS X v10.6 Snow Leopard abandonne définitivement AppleTalk. Apple boucle ainsi la boucle en se tournant vers les standards ouverts qu'elle avait initialement boudés.

Avec le protocole AppleTalk, un réseau sophistiqué reste simple d'usage. L'auto-configuration et la découverte automatique des services, concepts centraux d'AppleTalk, inspirent encore les développements actuels. Bonjour, le successeur spirituel d'AppleTalk chez Apple, perpétue cette philosophie de la simplicité dans un monde TCP/IP.

NTP

Dans l'univers des systèmes distribués, synchroniser l'heure entre machines distantes relève du défi technique permanent. Le Network Time Protocol a résolu cette équation avec une élégance qui traverse les décennies. Son origine remonte à 1979, quand une démonstration publique révèle Internet au grand jour via une liaison satellite transatlantique. Cette première apparition marque déjà l'importance que prendra la coordination temporelle dans l'architecture réseau.

Deux ans plus tard, en 1981, les Internet Engineering Notes consacrent leur référence IEN-173 à cette technologie de synchronisation. Le RFC-778 suit avec la première spécification publique du protocole. La conception initiale prend une forme inattendue : elle s'intègre directement dans le protocole de routage Hello, documenté sous RFC-891. Cette approche hybride perdure plusieurs années au sein de Fuzzball, un système d'exploitation expérimental qui servait de banc d'essai.

L'année 1985 marque la naissance officielle de NTP version 0. David L. Mills développe sa version dans Fuzzball tandis que Louis Mamakos et Michael Petry travaillent simultanément sur une implémentation UNIX à l'Université du Maryland. Leurs codes respectifs laissent des traces durables dans les logiciels actuels, témoignage de la solidité de leurs choix architecturaux. Le RFC-958 documente cette première version formelle, se concentrant sur l'en-tête des paquets NTP et les calculs de décalage et de délai qui demeurent inchangés aujourd'hui.

À cette époque, les contraintes matérielles dictent les performances. Les réseaux et ordinateurs modestes limitent la précision à une dizaine de millisecondes sur Ethernet. À titre de comparaison, les liaisons transatlantiques maintiennent généralement une précision sous les 100 millisecondes.

La version 1 émerge trois ans plus tard avec le RFC-1059. Ce document constitue le premier exposé complet du protocole et de ses algorithmes, incluant les premières versions des mécanismes de filtrage, sélection et discipline d'horloge. Ces développements s'appuient sur une série d'expérimentations documentées dans le RFC-956, où la théorie de l'algorithme de filtrage prend forme. Cette version introduit les modes client/serveur et symétrique, ainsi que l'utilisation du champ version dans l'en-tête des paquets.

En 1991 paraît le premier article de transaction académique consacré à NTP Version 1. Ce document présente pour la première fois le modèle complet à la communauté technique : architecture, protocole et algorithmes forment un ensemble cohérent dont les principes restent d'actualité. La reconnaissance scientifique du protocole est atteinte avec cette publication.

NTP Version 2 arrive en 1989 avec le RFC-1119. Dennis Fergusson à l'Université de Toronto le reconstruit intégralement pour respecter fidèlement la spécification. Cette version apporte deux innovations majeures : le protocole de message de contrôle NTP pour administrer serveurs et clients, et le système d'authentification cryptographique basé sur la cryptographie symétrique.

La spécification de NTP Version 3 paraît en 1992. Ses 113 pages incluent une annexe remarquable proposant une analyse formelle d'erreur accompagnée d'un budget détaillé couvrant toutes les contributions d'erreur depuis la source primaire jusqu'au client final. Ces travaux établissent les fondements des statistiques d'erreur maximale et estimée, caractérisant rigoureusement la qualité de synchronisation.

Lars Mathiesen à l'Université de Copenhague entreprend alors une révision minutieuse de la version 2 pour la conformer à la spécification version 3. Plus d'une année d'échanges intensifs entre spécification et implémentation aboutit à un modèle formel cohérent, processus exemplaire de convergence entre théorie et pratique.

Les années suivantes voient NTP continuer d'évoluer. Nouvelles fonctionnalités et révisions algorithmiques s'accumulent tout en préservant l'interopérabilité avec les versions antérieures. Cette période voit naître le Simple Network Time Protocol (SNTP) version 4, protocole simplifié compatible avec NTP sur IPv4, IPv6 et OSI, mais dépourvu des algorithmes sophistiqués d'atténuation et de discipline de son aîné.

Le déploiement mondial de NTP évoque l'esprit radio amateur. Chaque nouveau pays découvert utilisant NTP représente une victoire symbolique pour les développeurs. La satisfaction atteint son comble quand le laboratoire national des normes d'un pays établit un serveur NTP primaire connecté directement à son ensemble national de temps et de fréquence.

Maintenir scripts de configuration et bibliothèques de correctifs constitue une tâche ingrate mais nécessaire. Harlan Stenn, bénévole dévoué, gère actuellement ce processus grâce aux outils modernes d'autoconfiguration. Les nouvelles versions subissent d'abord les tests du réseau de recherche DCnet, puis ceux d'environnements plus larges comme CAIRN avant leur publication sur www.ntp.org.

Les progrès technologiques transforment radicalement les performances de NTP. La précision passe de centaines de millisecondes dans l'Internet tumultueux des années 1980 à des dizaines de nanosecondes dans l'Internet du nouveau millénaire. Cette évolution mêle leçons historiques et reprises technologiques, conservant ces résonances radio amateur à chaque nouveau pays apparaissant sur Internet avec NTP actif.

L'arrivée des horloges atomiques abordables révolutionne le domaine.

Les modèles au rubidium, avec leur dérive d'une microseconde par jour, démocratisent l'accès à une référence temporelle de haute qualité. Les oscillateurs OCXO offrent la meilleure alternative pour obtenir des horloges précises en boîtier compact. La température exerçant l'influence principale sur le taux d'oscillation du cristal, la thermostatisation contrôle et compense ces variations, limitant la dérive horlogère quotidienne.

NTP conserve le titre d'application distribuée la plus ancienne en fonctionnement continu sur Internet. Son développement se poursuit face aux nouveaux besoins de synchronisation temporelle. Centres de données et cloud computing exigent une précision temporelle toujours plus fine, où chaque nanoseconde compte pour les applications modernes. Cette quête perpétuelle de précision temporelle illustre parfaitement l'évolution d'Internet, d'un réseau expérimental à l'infrastructure critique de notre société numérique.

Microsoft Excel

En 1978, Dan Bricklin et Bob Frankston développent VisiCalc, vendu à près d'un million d'exemplaires et révolutionne la micro-informatique. Microsoft flaire le potentiel du marché, et en 1982 la firme de Redmond lance MultiPlan, qui adopte le système d'adressage R1C1 plutôt que le format A1 de VisiCalc. Le logiciel trouve son public sur les systèmes CP/M mais échoue face à Lotus 1-2-3 sur MS-DOS. Cette défaite pousse Microsoft à repenser sa stratégie.

Excel voit le jour en 1985, mais uniquement sur Macintosh. Le choix paraît audacieux : Microsoft mise sur l'interface graphique, les menus déroulants et la souris quand la plupart des programmes fonctionnent encore en mode texte. Cette vision se révèle payante. La version Windows débarque en 1987, accompagnée d'une version allégée du système d'exploitation pour les machines peu puissantes.

Les années suivantes transforment Excel en véritable couteau suisse du calcul. En 1990, la version 3.0 ajoute les barres d'outils, le dessin, les graphiques en 3D et les fonctions de plan. Excel 4.0, en 1992, améliore l'ergonomie et introduit la poignée de recopie, ce petit carré qui duplique des formules dans d'autres cellules d'un simple glissement. Un geste devenu si naturel que les utilisateurs peinent à imaginer s'en passer.

Les classeurs multi-feuilles changent la donne avec Excel 5.0 en 1993, mais c'est surtout l'arrivée du langage VBA qui révolutionne l'usage. Les

utilisateurs découvrent qu'ils peuvent enregistrer leurs actions répétitives sous forme de macros. Du jour au lendemain, Excel dépasse le simple calcul pour devenir un outil de programmation accessible.

Le passage à l'architecture 32 bits avec Excel 95 apporte stabilité et rapidité. Excel 97 nous gratifie de l'assistant Office, ce trombone animé qui agace autant qu'il aide, et perfectionne la validation des données. Plus sophistiquée, l'interface VBA attire une communauté grandissante de développeurs.

Excel 2000 améliore le presse-papiers multiple et ajoute la fonction d'auto-réparation. Excel 2002 introduit la récupération de fichiers après plantage, feature salvatrice pour quiconque a perdu des heures de travail. Excel 2003 mise sur le support XML et les plages « liste ».

2007 bouleverse les habitudes. Le « Ruban » remplace les menus traditionnels, déconcertant les utilisateurs aguerris mais séduisant les novices. Les formats XLSX et XLSM, basés sur XML, modernisent le stockage des données. Excel 2010 ajoute les graphiques sparkline, ces mini-graphiques intégrés dans les cellules, et des filtres plus puissants.

Cette évolution technique masque une réalité plus complexe. Excel s'impose certes dans les entreprises pour l'analyse financière, la gestion de projet et la modélisation statistique, mais sa facilité d'usage cache des pièges redoutables. Les erreurs se glissent facilement dans les formules complexes, et leurs conséquences peuvent être dramatiques.

JPMorgan Chase en fait l'amère expérience en 2012. Une erreur dans un modèle Excel calculant les risques coûte 6,2 milliards de dollars à la banque. Fidelity Magellan connaît un incident similaire en 1995 : un comptable oublie un signe moins devant une perte de 1,3 milliard de dollars, faussant complètement les estimations de dividendes. Ces accidents rappellent qu'Excel, malgré sa simplicité apparente, manipule souvent des enjeux considérables.

Cette double nature fait la richesse et la faiblesse d'Excel. D'un côté, il démocratise l'analyse de données et rend accessible le calcul numérique à des millions d'utilisateurs non spécialistes. De l'autre, il génère une dépendance parfois dangereuse dans des domaines où la précision compte. Les secteurs de la finance structurée et de l'analyse quantitative lui doivent leur essor, mais aussi quelques-unes de leurs plus cuisantes déconvenues.

Les versions récentes tentent de concilier ces exigences contradictoires. L'intelligence artificielle s'invite dans les formules, les outils de visualisation se sophistiquent, les capacités de traitement s'étoffent pour gérer des volumes de données toujours plus importants. Excel s'adapte au cloud computing et au travail collaboratif, domaines où il accusait un retard certain.

Microsoft ne lâche pas prise. Les développements récents montrent une

volonté de maintenir Excel au cœur des pratiques professionnelles modernes. L'intégration de langages de programmation avancés comme Python, l'amélioration des analyses et l'adaptation aux nouveaux formats témoignent de cette ambition.

Plus de quarante ans après VisiCalc, Excel incarne l'évolution des outils informatiques professionnels. Son parcours reflète les transformations du travail de bureau et l'informatisation progressive de la société. Sa persistance dans un monde technologique en perpétuel mouvement démontre sa capacité unique à répondre aux besoins fondamentaux de traitement de l'information, malgré ses imperfections et les risques qu'il génère.

Microsoft Windows

En 1983, Bill Gates annonce que Microsoft travaille sur une interface graphique. L'idée n'a rien de vraiment nouveau, Xerox a déjà exploré cette voie au PARC, Apple vient de sortir Lisa et prépare le Macintosh. Mais Microsoft veut sa part du gâteau et compte bien transformer l'expérience utilisateur sur PC.

Novembre 1985 voit naître la première version de Windows. Le résultat déçoit. Cette couche graphique posée sur MS-DOS propose des fenêtres, des menus déroulants et quelques icônes. Les utilisateurs, habitués à taper leurs commandes, peinent à adopter cette nouveauté. Les ventes restent confidentielles.

Deux ans plus tard, Windows 2.0 corrige plusieurs défauts. Les fenêtres peuvent désormais se chevaucher au lieu de s'aligner sagement en mosaïque. Microsoft collabore alors avec IBM sur OS/2 et cherche une cohérence visuelle entre ses différents projets. Cette version passe largement inaperçue.

Le décollage véritable survient en 1990 avec Windows 3.0. Cette fois, Microsoft tient un produit qui fonctionne. L'interface affiche 16 couleurs, exploite les capacités des processeurs 386 et propose trois composants qui vont marquer l'histoire : le Gestionnaire de Programmes, le Gestionnaire de Fichiers et le Gestionnaire d'Impression. Ces outils simplifient la vie des utilisateurs et rendent l'ordinateur accessible au grand public. Les ventes s'envolent et rattrapent celles d'Apple.

Windows 3.1 arrive en 1992 avec une stabilité renforcée. La technologie OLE fait ses débuts en apportant l'intégration des objets d'une application dans une autre. Les premiers éléments multimédias font leur apparition sur

PC. Windows for Workgroups 3.11 ajoute les fonctions réseau indispensables aux entreprises qui commencent à connecter leurs machines.

Microsoft développe en parallèle Windows NT, un système pensé pour les professionnels. Sorti en 1993, NT rompt avec MS-DOS et propose une architecture entièrement nouvelle. L'entreprise vise le marché des serveurs, chasse gardée d'UNIX. La robustesse de NT lui vaut une certification C2 du gouvernement américain, gage de sérieux dans le monde de l'entreprise.

Août 1995 marque une date dans l'histoire de l'informatique personnelle. Windows 95 débarque avec son menu Démarrer et sa barre des tâches, deux éléments qui vont devenir familiers à des millions d'utilisateurs. Le « Plug and Play » simplifie l'installation de matériel. L'architecture 32 bits améliore les performances. Microsoft dépense 300 millions de dollars en publicité et utilise « Start Me Up » des Rolling Stones pour accompagner le lancement. L'informatique entre définitivement dans les foyers.

Windows 98 peaufine la formule l'année suivante. Internet prend de l'importance et Microsoft facilite les connexions. L'USB fait ses débuts, le format FAT32 augmente la capacité des disques durs. Le système gagne en stabilité et propose des outils de maintenance automatique.

En février 2000, Windows 2000 unifie enfin les branches grand public et professionnelle. Basé sur NT, ce système apporte la fiabilité tant attendue. Active Directory révolutionne la gestion des réseaux d'entreprise. Le multiprocesseur est une réalité sur les postes de travail.

Windows XP, sorti en 2001, parachève cette évolution. L'interface graphique « Luna » modernise l'apparence du système. La compatibilité matérielle s'améliore, le multimédia trouve sa place. XP combine la convivialité des versions domestiques avec la solidité des éditions professionnelles. Cette version va rester populaire pendant une décennie.

Les années 2000 voient naître de nouvelles préoccupations. Internet se démocratise mais apporte son lot de virus, de chevaux de Troie et autres maliciels. Microsoft réagit avec des correctifs réguliers et renforce la sécurité. Le Service Pack 2 de XP introduit un pare-feu intégré et un centre de sécurité. La guerre contre les menaces informatiques commence.

Cette histoire technique s'accompagne de controverses commerciales. L'intégration d'Internet Explorer dans Windows suscite des poursuites antitrust aux États-Unis et en Europe. Les concurrents dénoncent les pratiques de Microsoft et sa position dominante. Ces batailles juridiques marquent les années 1990 et 2000.

L'évolution de Windows reflète celle de l'informatique personnelle tout entière. Du mode texte austère de MS-DOS aux interfaces colorées modernes, Microsoft a su maintenir la compatibilité avec les applications existantes tout en innovant. Cette continuité explique en partie la domination

persistante du système.

Windows équipe 72% des ordinateurs personnels dans le monde en 2025. Quatre décennies après sa création, le système de Microsoft résiste aux assauts de ses concurrents. Les smartphones et tablettes ont bousculé le marché, mais Windows garde sa place sur les bureaux et dans les entreprises.

Amstrad PCW

En 1985, l'informatique personnelle européenne connut un tournant inattendu avec l'arrivée de l'Amstrad PCW. Loin des machines de jeux colorées qui dominaient alors le marché domestique, cet ordinateur britannique adoptait une philosophie radicalement différente : faire du traitement de texte son unique raison d'être.

Alan Sugar, patron d'Amstrad, imagine un ensemble intégré comprenant clavier, écran, unité centrale, et imprimante matricielle. Son cahier des charges tient en quelques mots : simple, abordable, efficace. Le Z80 cadencé à 4 MHz constituait le cœur technique du système. Ce processeur 8 bits permettait aux développeurs de s'appuyer sur leur expertise existante plutôt que de partir de zéro avec le 6502 à la mode. La mémoire de 256 Ko du PCW 8256 pouvait être doublée sur le modèle supérieur, une capacité respectable pour l'époque. Mais c'est le choix du format de disquette 3 pouces qui marqua le plus l'histoire de cette machine : propriétaire, il la protégeait temporairement de la concurrence tout en hypothéquant son avenir face au 3,5 pouces qui s'imposait ailleurs.

L'écran monochrome de 720×256 pixels, d'un vert caractéristique, affichait exclusivement du texte. Cette limitation assumée correspondait parfaitement à l'usage prévu : LocScript, le logiciel de traitement de texte intégré, transformait chaque PCW en machine à écrire électronique perfectionnée. Le système d'exploitation CP/M ouvrait l'accès à une bibliothèque logicielle existante, héritée des machines professionnelles précédentes.

Le succès commercial dépassa toutes les prévisions. À 400 livres sterling avec l'imprimante matricielle incluse, le PCW séduisit un public que personne n'avait vraiment ciblé auparavant : secrétaires indépendantes, petites entreprises, écrivains, administrations locales. En France notamment, ces machines blanches et vertes colonisèrent les bureaux des professions libérales et des services publics. Plus d'un million d'exemplaires trouvèrent preneur entre 1985 et 1990, un chiffre remarquable pour un produit aussi spécialisé.

Les évolutions successives affinèrent le concept initial sans le trahir. Le PCW 9512 de 1987 troqua l'imprimante matricielle contre une imprimante à marguerite, produisant enfin une qualité « courrier » aux documents produits. Sa robe entièrement blanche modernisait l'esthétique tout en conservant l'ergonomie éprouvée. Le PCW 9256, ultime version de 1991, apporta quelques améliorations techniques mineures sans révolutionner l'usage.

LocScript évoluait lui aussi, intégrant progressivement les accents français, une correction orthographique rudimentaire, la gestion de tableaux simples. Des logiciels tiers enrichirent l'écosystème : comptabilité, gestion de fichiers clients, avec quelques jeux pour les moments de détente. L'interface LocoLink permettait d'échanger des documents avec les PC sous MS-DOS, une ouverture précieuse quand ces derniers commencèrent à se démocratiser.

Pourtant, les faiblesses du concept apparurent au fil du temps. Pas de couleur, pas de polices proportionnelles à l'écran, une lenteur perceptible face aux machines 16 bits. Les compatibles PC, devenus abordables au début des années 1990, offraient plus de polyvalence pour un prix comparable. Le PCW, prisonnier de sa spécialisation, perdait progressivement sa pertinence commerciale.

Cette machine prouva qu'un ordinateur personnel pouvait choisir l'efficacité contre la sophistication, la simplicité contre la puissance brute. De nos jours, de nombreux PCW fonctionnent parfaitement, témoignage de leur robustesse constructive exceptionnelle.

L'Amstrad PCW incarne une vision particulière de l'informatisation : plutôt qu'imposer un système complexe et généraliste, mieux vaut concevoir l'outil adapté à un besoin précis. Cette approche permit à des milliers d'utilisateurs de découvrir le traitement de texte numérique sans intimidation technique. Secrétaires, artisans, associations : tous trouvèrent dans cette machine blanche leur première porte d'entrée vers l'informatique professionnelle.

Cette réussite européenne, face à la standardisation américaine du PC, montre qu'il existait plusieurs voies possibles pour démocratiser l'informatique. Amstrad explora l'une d'elles avec succès, prouvant qu'une demande existait pour des solutions simples et abordables.

Atari 520 ST

Après avoir mené Commodore International vers le succès du Commodore 64, Jack Tramiel quitte l'entreprise en 1983 suite à une guerre des prix acharnée contre Texas Instruments qui l'a épuisé. Cette rupture le conduit à fonder Tramel Technology Ltd avec ses fils et quelques fidèles, notamment Shiraz Shivij, l'un des cerveaux derrière le C64.

La petite équipe se lance dans l'aventure d'un ordinateur personnel 16 bits à prix abordable. Le choix du processeur s'avère délicat : le National Semiconductor NS32000 ne tient pas ses promesses de performances. C'est finalement le Motorola 68000 qui emporte la décision, malgré son coût plus élevé. Pendant ce temps, Atari Inc. sombre. Le krach du jeu vidéo de 1983 fait perdre 10 000 dollars par jour à Warner Communications, sa maison mère. Jack Tramiel flaire l'aubaine et rachète la division grand public d'Atari en juillet 1984 pour 240 millions de dollars en actions.

L'équipe technique accomplit le développement du 520ST en cinq mois seulement, un petit miracle qui paraît irréalisable aujourd'hui. Le nom raconte l'histoire de la machine : 520 Ko de mémoire vive et cette architecture « Sixteen/Thirty-two » qui mélange les mondes 16 et 32 bits. Microsoft propose bien d'adapter Windows, mais les calendriers ne collent pas. Digital Research sauve la mise avec une licence de GEM, qui donnera à l'Atari son interface graphique si caractéristique.

Au Consumer Electronics Show de janvier 1985, les spécifications du 520ST font sensation. Le Motorola 68000 à 8 MHz cache une architecture interne 32 bits derrière un bus externe 16 bits. Trois modes graphiques cohabitent : du 320 x 200 en 16 couleurs pour les jeux, du 640 x 200 en 4 couleurs pour un compromis équilibré, et surtout ce mode monochrome haute résolution 640 x 400 qui va séduire les professionnels.

Les ingénieurs d'Atari ont glissé quelques trouvailles dans leur machine. Ces ports MIDI intégrés pour la modique somme de 75 cents, grâce à des puces Motorola déclassées, vont transformer le 520ST en instrument de musique électronique. Le lecteur de disquettes 3,5 pouces stocke 360 Ko, une capacité respectable quand la plupart des concurrents s'obstinent encore avec le 5,25 pouces. Le système d'exploitation TOS charge d'abord depuis une disquette avant de migrer définitivement en ROM.

Mai 1985 voit arriver les premiers 520ST dans les magasins, devançant de justesse l'Amiga 1000 de Commodore. Jack Tramiel joue la carte du prix agressif : 799 dollars avec un moniteur monochrome, 999 dollars en couleur. Face aux machines professionnelles qui dépassent allègrement les 2 000 dollars, l'argument porte. Plus de 50 000 unités trouvent preneur en six mois, principalement en Europe où l'accueil est chaleureux.

La machine séduit des publics inattendus. Les musiciens découvrent les joies du MIDI avec des logiciels comme Cubase et Logic Pro qui feront carrière. Les programmeurs amateurs s'amusent avec GFA BASIC et STOS. Quant aux professionnels de la PAO, ils apprécient ce mode haute résolution, d'autant que l'émulation Macintosh tourne plus vite sur ST que sur un vrai Mac !

Le succès européen ne doit rien au hasard. Ce design compact et intégré correspond aux goûts du Vieux Continent. Le slogan « Power Without the Price » fait mouche au Royaume-Uni, conquis par les ordinateurs domestiques abordables. Aux États-Unis, c'est plus compliqué : Jack Tramiel entretient des relations exécrables avec les revendeurs, qui le lui rendent bien.

L'influence du 520ST dépasse ses simples performances techniques. Sa capacité à lire les disquettes MS-DOS facilite les échanges avec l'univers PC. Ces ports MIDI standardisés créent tout un écosystème qui survit encore aujourd'hui. Jean-Michel Jarre, Fatboy Slim ou Madonna l'utilisent en studio et sur scène, créant une légende qui colle à la machine.

Le catalogue logiciel s'étoffe. Dungeon Master débarque en 1987 et révolutionne le jeu de rôle en vue subjective. Des éditeurs comme Batteries Included explorent les possibilités graphiques avec des programmes comme DEGAS. La « demo scene » naissante pousse la machine dans ses retranchements, créant des œuvres d'art numérique qui défient toute logique technique.

La gamme évolue au fil des versions. Le 520STM ajoute un modulateur RF pour se brancher sur une télévision. Le 520STFM intègre alimentation et lecteur dans un boîtier repensé, plus familial. TOS migre progressivement de la disquette vers la ROM, simplifiant l'utilisation.

En 1993, la production du 520ST s'arrête, emportée par la vague des PC compatibles IBM. Pourtant, la machine garde ses fidèles, surtout chez les musiciens qui apprécient sa faible latence MIDI. Elle incarne cette époque où l'innovation technique rimait avec démocratisation de la technologie, quand une poignée d'ingénieurs pouvait encore bousculer l'industrie informatique.

Toshiba T1100

En avril 1985, Toshiba bouleverse l'informatique personnelle avec le T1100. Pour la première fois, un ordinateur portable offre une compati-

bilité totale avec l'IBM PC. Cette machine de 4 kg marque la naissance de l'informatique nomade moderne.

L'époque paraît pourtant peu propice à pareille innovation. Les ordinateurs personnels, Apple II depuis 1977 et IBM PC depuis 1981, restent cloués aux bureaux. Quelques constructeurs tentent bien de créer des machines transportables : le Portable I de Compaq pèse 12,5 kg en 1982, tandis que Seiko sort son HX-20/HC-20 de 1,6 kg la même année, suivi par le PC8201A de NEC en 1983. Mais ces tentatives se heurtent à un dilemme technique : soit miniaturiser au maximum avec des architectures propriétaires incompatibles, soit viser la compatibilité IBM PC en acceptant l'encombrement et les limitations.

Les équipes de Toshiba refusent ce compromis. Leur devise « anywhere, anytime, anyone » exprime une ambition claire : transformer l'ordinateur de bureau en compagnon mobile. L'entreprise théorise cette approche comme une « révolution sur le bureau », un passage d'une informatique centrée sur la machine vers une utilisation centrée sur l'humain. Trois objectifs guident le développement : abandonner les architectures fermées, rendre l'ordinateur vraiment portable et permettre l'usage de logiciels standard sans programmation spécifique.

La réalisation technique tient du prodige d'ingénierie. Les circuits intégrés CMOS remplacent la technologie NMOS gourmande en énergie. Là où l'IBM PC mobilise près de 100 circuits pour l'affichage, le T1100 s'en contente de cinq. Cette optimisation radicale produit un appareil compact doté d'une autonomie inédite. L'énergie est un terrain d'innovation. Toshiba crée de toutes pièces un système de charge-décharge et un logiciel de contrôle énergétique. Ces développements jetteront les bases de la fonction *resume* introduite en 1989 sur le modèle J3100SS001, puis du standard ACPI élaboré avec Microsoft et Intel.

L'écran LCD à fort contraste, monté sur support inclinable, résulte d'études poussées sur l'ergonomie visuelle. Mais le vrai défi commercial surgit ailleurs : le choix des disquettes 3,5 pouces. Alors que l'IBM PC fonctionne avec du 5,25 pouces, Toshiba mise sur le format compact. Seulement les éditeurs rechignent à distribuer leurs logiciels sur ce support encore marginal, malgré le travail des équipes commerciales qui mènent une campagne de persuasion intensive pour les convaincre d'adopter le 3,5 pouces.

La stratégie marketing rompt avec les habitudes du secteur. Fini les argumentaires techniques ! Toshiba montre concrètement comment travailler en mobilité avec les mêmes logiciels qu'au bureau fixe. Cette approche pragmatique séduit les professionnels non-techniciens et élargit considérablement le marché.

Le succès commercial confirme cette intuition. Une étude révèle deux

attentes distinctes : la portabilité d'un côté, plus de puissance et de capacité de l'autre. Toshiba répond en janvier 1986 avec deux nouveaux modèles. Le T2100 embarque un processeur i8086 et deux lecteurs 3,5 pouces. Le T3100 monte en gamme avec un i80286 et un disque dur de 10 mégaoctets. Ces machines nécessitent une alimentation secteur, mais *PC Week* valide ce choix en reconnaissant l'existence d'un marché pour les portables branchés. Le magazine *Byte* qualifie le T3100 de « Roi des portables ». En 1987, Toshiba atteint environ 40% de parts sur les 130 000 ordinateurs portables vendus en Europe.

L'impact économique dépasse toutes les prévisions. En 2008, le marché des écrans LCD de moins de 17 pouces représente 15 milliards de dollars. Celui des disques durs 2,5 pouces et moins atteint 19 milliards. Les batteries lithium-ion pour équipements mobiles génèrent 12 milliards de chiffre d'affaires. Un an plus tard, les ventes de portables dépassent celles des machines de bureau sur un marché estimé à 150 milliards de dollars.

Le T1100 démontre qu'une approche centrée sur l'utilisateur transforme les secteurs technologiques. En privilégiant compatibilité logicielle, portabilité et simplicité plutôt que les performances brutes, Toshiba fait sortir l'ordinateur des services informatiques pour l'installer dans les mallettes des cadres. Cette démocratisation de l'informatique mobile annonce l'omniprésence actuelle des appareils portables dans nos vies.

Aldus PageMaker

En 1984 à Seattle, Paul Brainerd quitte son poste chez Atex pour fonder Aldus Corporation. Il n'est pas un inconnu dans le monde de l'édition. Journaliste de formation, il a supervisé la modernisation des systèmes du Minneapolis Star and Tribune avant de rejoindre ce fabricant de systèmes informatiques destinés aux professionnels de l'édition. Cette expérience lui fait toucher du doigt un problème : les équipements de composition coûtent une fortune, souvent plusieurs centaines de milliers de dollars, et restent inaccessibles aux petites structures.

Avec son interface graphique, le Macintosh d'Apple ouvre des perspectives inédites. Brainerd pressent qu'il est possible de démocratiser les outils de mise en page. Il rassemble une petite équipe d'ingénieurs et développe un prototype en six mois. Trouver des financements s'avère autrement plus compliqué : sur cinquante sociétés de capital-risque contactées, une seule, Vanguard, accepte de parier sur ce projet. À l'époque, les investisseurs se

méfient des entreprises de logiciels, jugées trop risquées dans un secteur où Microsoft n'a pas encore fait son entrée en bourse.

Le 13 juillet 1985, PageMaker 1.0 voit le jour. Cette date n'a rien d'un hasard : elle suit de sept mois le lancement de l'imprimante LaserWriter d'Apple. Brainerd a compris que son logiciel doit s'appuyer sur les capacités du langage PostScript d'Adobe, intégré dans cette imprimante, pour produire des documents de qualité professionnelle. Il parle du « tabouret à trois pieds » de la publication assistée par ordinateur, expression qu'il invente d'ailleurs lors d'une réunion avec ses investisseurs fin 1984.

Ce terme de « publication assistée par ordinateur » fait mouche. Simple, efficace, il résume parfaitement l'ambition du projet : mettre les outils de mise en page professionnelle à la portée de tous. L'interface de PageMaker traduit cette philosophie. Le logiciel simule une vraie table de montage où l'utilisateur place et déplace textes et images à sa guise. La palette flottante d'outils forme une innovation marquante, donnant accès aux fonctions principales. Le principe WYSIWYG garantit que ce qui apparaît à l'écran correspond exactement au résultat imprimé.

Le succès dépasse toutes les prévisions. Les utilisateurs affluent de secteurs inattendus. Des églises se mettent à produire des bulletins paroissiaux parfois tirés à plus de 600 000 exemplaires. Steve Jobs avait poussé pour maintenir le prix sous les 200 dollars, mais les 495 dollars finalement retenus s'avèrent judicieux : cette tarification dégage les marges nécessaires au développement continu et au support technique.

Aldus distribue aussi FreeHand, un logiciel de dessin vectoriel créé par Altsys, puis rachète Silicon Beach Software avec sa gamme d'applications grand public. Ces acquisitions construisent progressivement une offre complète d'outils de création graphique.

La concurrence réagit. Ventura Publisher débarque en 1986 sur PC avec une approche différente, fondée sur les feuilles de style et adaptée aux documents longs. QuarkXPress gagne du terrain dans le secteur professionnel. Microsoft tente sa chance mais doit retirer son produit en raison de défauts techniques rédhibitoires.

Au-delà du logiciel, Aldus façonne les standards du secteur. Steve Carlson développe chez Aldus le format TIFF (Tagged Image File Format), qui est alors la référence pour l'échange d'images numériques. L'entreprise crée plus tard l'OPI (Open Prepress Interface) pour fluidifier l'intégration avec les systèmes d'impression professionnelle.

En 1990, les marges s'érodent sous la pression concurrentielle et l'évolution des circuits de distribution. PageMaker se complexifie, tiraillé entre les besoins des utilisateurs occasionnels et les exigences des professionnels. Cette complexité croissante complique le développement et alourdit

les coûts.

En 1994, Aldus fusionne avec Adobe Systems. Cette opération, préparée avec un soin méticuleux, unit les technologies des deux entreprises. L'absorption se déroule remarquablement bien, créant un géant de la création graphique numérique.

PageMaker a prouvé qu'un outil informatique accessible pouvait supplanter des équipements spécialisés hors de prix. Il a établi des conventions d'interface utilisateur, mais il a surtout ouvert l'accès à la publication de qualité professionnelle à des individus et des organisations qui n'avaient jamais pu s'offrir ce luxe auparavant.

Par la suite, le code de PageMaker nourrit Adobe InDesign, son héritier spirituel qui règne désormais sur le marché de la PAO. Paul Brainerd, lui, s'est tourné vers la philanthropie après la fusion avec Adobe. Sa fondation Brainerd œuvre pour la protection de l'environnement dans le Nord-Ouest américain.

Cette histoire montre comment une vision nette, soutenue par une exécution technique rigoureuse et un calendrier favorable, peut bouleverser tout un secteur. Elle souligne aussi l'importance des standards ouverts et de la coopération entre entreprises pour faire émerger de nouveaux marchés.

IBM AIX

Pendant les années 1970, UNIX se répand dans les universités, mais sans version commerciale standardisée. Chaque installation demandait d'adapter le code source au matériel local. Cette contrainte technique pousse à la création de multiples variantes, chacune reflétant les spécificités de son environnement.

IBM s'intéresse au phénomène UNIX et lance AIX (Advanced Interactive eXecutive) en 1986, développé avec l'aide d'Interactive Systems Corporation. Le nom provient d'ailleurs d'ISC, qui avait déjà créé le système PC-IX pour les ordinateurs personnels d'IBM.

IBM jonglait alors avec deux architectures Power distinctes. D'un côté, OS/400 équipait une gamme de machines, de l'autre, AIX prenait ses quartiers sur la station de travail IBM 6150 RT en 1986. L'AS/400 suivait l'année d'après avec OS/400.

La véritable montée en puissance d'AIX arrive en 1990 avec la Version 3, rebaptisée AIX/6000. Elle accompagne la plateforme RS/6000 qui en-

terre définitivement l'IBM RT. Cette génération RS/6000 inaugure l'ère des microprocesseurs POWER et PowerPC chez IBM. AIX trouve enfin sa machine de prédilection.

1994 marque une étape technique importante avec AIX 4, qui introduit le multitraitement symétrique sur les premiers serveurs RS/6000 SMP. Le système mûrit jusqu'à la version 4.3.3 en 1999, pendant qu'IBM repense sa stratégie produits.

L'an 2000 bouleverse l'organisation d'IBM. La société unifie ses gammes de serveurs sous la bannière eServer : AS/400 se renomme iSeries, RS/6000 se transforme en pSeries. Au-delà du marketing, cette restructuration cache une réalité technique. L'arrivée du processeur POWER4 en 2001-2002 rapproche les plateformes matérielles. Les différences entre systèmes « p » et « i » se limitent désormais aux couches logicielles.

POWER5 franchit un pas supplémentaire en 2004 en rendant le System i5 570 rigoureusement identique au System p5 570. AIX 5.3 débarque en août, tirant parti de cette standardisation. POWER6 suit en mai 2007, accompagné d'AIX 6.1 en novembre. IBM pousse la logique jusqu'au bout en avril 2008 en fusionnant ses deux lignes sous l'appellation Power Systems. Un seul matériel, trois choix de système : IBM i, AIX ou Linux.

Les années 2010 s'enchaînent à un rythme soutenu. POWER7 arrive en février 2010, AIX 7.1 en septembre. POWER7+ débarque en août 2012, puis POWER8 en juin 2014. Ce dernier processeur rompt avec les habitudes : chaque cœur gère huit threads matériels en parallèle, une architecture massivement multithreadée qui exploite des caches volumineux pour maximiser les débits mémoire et d'entrées-sorties.

AIX 7.2, annoncé en octobre 2015, apporte une fonction remarquable : Live Kernel Update. Les correctifs du noyau s'appliquent désormais sans perturber les applications en cours d'exécution. Cette version nettoie aussi les composants obsolètes et restructure le réseau avec bos.net.tcp.client, mais elle exige impérativement POWER7 ou une génération ultérieure.

POWER9 débarque en 2016 avec sa gravure 14 nm et ses 3,9 GHz de fréquence. Les gains de performance atteignent 50% par rapport à POWER8, et doublent face à POWER7+. Sur AIX 7.2 TL 3, le mode SMT8 s'active automatiquement, exploitant au maximum les capacités du processeur.

Aujourd'hui, AIX équipe les infrastructures les plus critiques. Finance, industrie, distribution, télécommunications, santé, administration et secteur public : partout où la fiabilité ne se négocie pas. Le système a prouvé sa capacité à évoluer avec les nouvelles architectures matérielles sans sacrifier la compatibilité avec l'existant.

Cette longévité technique illustre parfaitement la stratégie d'IBM : faire converger ses gammes de serveurs vers une plateforme unifiée tout en pré-

servant l'investissement logiciel des clients. AIX incarne cette continuité dans l'innovation, héritière d'UNIX mais résolument tournée vers l'avenir de l'informatique d'entreprise.

Eiffel

23 septembre 1985. Dans les bureaux d'Interactive Software Engineering, une petite société californienne de Santa Barbara, Bertrand Meyer travaille sur un projet interne qui va bousculer bien des habitudes dans l'univers de la programmation orientée objet. Ce jour-là naît Eiffel, mais il faudra attendre le milieu de l'année 1986 pour voir fonctionner la première version opérationnelle.

Meyer ne part pas de rien. Depuis 1973, il manipule les concepts orientés objet avec Simula 67. Les travaux de Jean-Raymond Abrial sur la spécification formelle le fascinent, particulièrement sa version originale du langage Z. Liskov, Zilles et Guttag ont défriché le terrain des types de données abstraits, un domaine sur lequel Meyer a planché. La lignée prestigieuse Algol 60, Algol W, Pascal et Ada forge l'architecture générale du nouveau langage, tandis que Floyd, Hoare et Dijkstra apportent leur expertise sur la preuve de programmes et la sémantique axiomatique.

Cette alchimie particulière produit quelque chose d'inhabituel. En septembre 1986, à la première conférence « Object-Oriented Programming, Systems, Languages & Applications » de Portland, la présentation d'Eiffel fait sensation. Les concepts développés surpassent ce qui existe alors, tant dans l'industrie que dans les laboratoires de recherche. La transformation suit avec la commercialisation du compilateur interne dès décembre 1986. Les entreprises et universités peuvent enfin s'en emparer.

La version 2 arrive en 1988, année qui voit paraître « Object-Oriented Software Construction », le livre qui va propulser Eiffel sur le devant de la scène. L'édition de 1997, considérablement étoffée, confirme l'influence grandissante du langage. La version 2.3 de 1990 marque l'apogée de la technologie originale d'ISE.

Meyer prend alors une décision audacieuse : en 1990, ISE libère la définition du langage dans le domaine public. Cette ouverture déclenche une floraison de projets, compilateurs et bibliothèques éclosent un peu partout. Le langage traverse une période de refonte générale, se simplifie par endroits, s'enrichit par d'autres, mais conserve son âme originelle. Cette stabilité remarquable fait d'Eiffel un langage qui n'a guère changé depuis sa conception de 1985.

Entre 1990 et 1993, l'équipe d'ISE repense entièrement sa technologie. La version 2.3 sert d'amorce pour cette reconstruction complète. ISE Eiffel 3 débarque en 1993 avec son environnement de développement graphique, suivi par ISE Eiffel 4 en 1997 et Eiffel 5 en 2001. Ces versions apportent leur lot de nouveautés : les agents empruntent aux langages fonctionnels, l'introspection scrute les programmes, le mécanisme Precursor simplifie l'héritage répété, la création d'objets gagne en souplesse et les mécanismes de conversion s'affinent.

La notation BON surgit en 1995 avec l'ouvrage « Seamless Object-Oriented Software Construction » de Waldén et Nerson. Cette Business Object Notation étend Eiffel vers l'analyse et la conception, dans un langage que comprennent gestionnaires, analystes et architectes système.

L'écosystème grandit. Object Tools en Allemagne développe Visual Eiffel, héritier d'Eiffel/S. L'Université de Nancy propose Small Eiffel en version libre, Halstenbach GmbH commercialise sa propre solution allemande. Les bibliothèques foisonnent : graphisme 3D, analyse lexicale avec Gobo, interfaces DirectX, calcul à précision variable. Chaque domaine trouve sa solution.

Les secteurs d'activité découvrent les vertus d'Eiffel. La banque l'adopte, la finance s'y intéresse, la comptabilité l'expérimente. Les télécommunications, la santé, la CAO-FAO, la simulation et le calcul scientifique lui font confiance. Le système Rainbow de la banque CALFP raconte cette success story : conçu initialement pour le trading de produits dérivés, il finit par superviser la majorité des opérations bancaires.

L'université s'empare du langage avec enthousiasme. Nombreuses sont les facultés qui en font le premier langage enseigné aux étudiants. D'autres l'intègrent à différents niveaux du cursus, encouragées par les packages attractifs des éditeurs.

Pourquoi ce nom, Eiffel ? Meyer rend hommage à Gustave Eiffel, l'ingénieur qui ne s'est pas contenté d'ériger la tour parisienne. L'armature métallique de la Statue de la Liberté à New York, une gare ferroviaire à Budapest, autant de constructions durables sorties de ses ateliers. La Tour Eiffel, entamée en 1887 pour l'Exposition universelle de 1889, respecte délais et budget. Elle traverse les hostilités politiques, résiste aux tentatives de destruction, trouve de nouveaux usages avec la radio et la télévision. Construite à partir de motifs robustes et élégants, combinés et variés pour produire un résultat puissant, elle incarne parfaitement ce qu'Eiffel accomplit dans l'univers logiciel.

Le « Design by Contract » forge l'identité d'Eiffel. Cette approche fait évoluer la construction logicielle en la fondant sur des contrats entre clients et fournisseurs. Obligations et bénéfices mutuels s'explicitent par des asser-

tions, établissant clairement les responsabilités de chaque composant. Cette philosophie produit des logiciels plus fiables et maintenables, où chaque élément connaît son rôle et ses limites.

Erlang

En 1986, dans les bureaux du laboratoire d'informatique d'Ericsson Telecom AB, Joe Armstrong se trouve confronté à un casse-tête technique : comment programmer des applications de téléphonie qui défient toutes les conventions de l'époque ? Les commutateurs téléphoniques traitent des milliers d'appels simultanés, ces opérations se distribuent naturellement sur plusieurs machines, et la moindre panne fait les gros titres. Rien à voir avec une application de bureau qui plante sans conséquence.

Armstrong explore d'abord différentes pistes, développe des prototypes en Smalltalk, invente une notation graphique pour décrire les opérations téléphoniques. C'est Roger Skagervall qui fait la remarque décisive : cette notation ressemble à du Prolog. Armstrong bascule alors vers ce langage, et c'est ainsi que naît véritablement Erlang.

Le contexte technique d'Ericsson influence la conception du nouveau langage. PLEX, créé par Göran Hemdahl pour programmer les commutateurs AXE, impose ses contraintes : modifier le code sans arrêter le système, éviter à tout prix les erreurs de pointeurs qui avaient handicapé les générations précédentes. Armstrong et son équipe intègrent ces leçons dans l'ADN d'Erlang.

Robert Virding rejoint le projet, et ensemble ils développent les premières versions du langage, toujours en Prolog. Le test de vérité arrive en 1989 avec le projet ACS/Dunder. L'équipe développe 25 fonctionnalités téléphoniques en Erlang, soit environ un dixième de celles du MD110. Les résultats dépassent leurs espérances : selon les fonctionnalités, le développement s'accélère de 3 à 25 fois par rapport à PLEX.

Ces succès attirent l'attention, et Erlang gagne en maturité. Bogumil Hausman développe en 1993 le système Turbo Erlang, rebaptisé BEAM (Bogdan's Erlang Abstract Machine), qui améliore drastiquement les performances. Claes Wikström ajoute le support de la distribution, permettant l'exécution sur plusieurs machines.

Mais c'est un événement inattendu qui va propulser Erlang hors des murs d'Ericsson. En 1998, Ericsson Radio Systems interdit l'utilisation du langage pour les nouveaux développements. Cette décision, qui aurait pu si-

gner l'arrêt de mort d'Erlang, produit l'effet inverse. En décembre, Ericsson libère le code source, et une partie de l'équipe originale quitte l'entreprise pour fonder Bluetail AB, société qui utilisera Erlang comme technologie de base.

L'architecture technique d'Erlang reflète sa mission initiale. Le langage structure les programmes autour de processus concurrents qui ne partagent aucune mémoire et communiquent par messages asynchrones. Ces processus appartiennent au langage, pas au système d'exploitation, ce qui les rend remarquablement légers. Si l'un d'eux échoue, les autres continuent leur travail comme si de rien n'était.

Le système de réception de messages constitue l'une des innovations les plus remarquables d'Erlang. Un processus peut attendre sélectivement certains types de messages tout en laissant les autres en attente. Cette approche simplifie énormément la programmation de protocoles de communication complexes, un atout considérable dans le monde des télécommunications.

L'AXD301, commutateur ATM développé par Ericsson, est la vitrine technologique d'Erlang. En 2001, ce système compte 1,13 million de lignes de code réparties dans 2 248 modules. Il atteint la disponibilité rêvée de 99,9999999%, démontrant que le langage est en capacité de gérer des projets industriels d'envergure avec une fiabilité exceptionnelle.

Le système OTP (Open Telecom Platform), développé à partir de 1996, enrichit l'écosystème Erlang. Cette collection de bibliothèques et de modèles de conception propose des « behaviours », abstractions qui encapsulent des motifs courants comme le modèle client-serveur ou la gestion d'événements. OTP transforme Erlang d'un langage expérimental en plateforme industrielle.

L'arrivée des processeurs multicœurs confirme rétrospectivement la pertinence des choix initiaux. Le modèle de concurrence sans partage de données s'adapte naturellement au parallélisme matériel. Les programmes Erlang exploitent les architectures multicœurs sans modification, un avantage considérable à une époque où les autres langages peinent à tirer parti de cette évolution.

Au-delà des télécommunications, Erlang trouve sa place dans des domaines variés. Le serveur de messagerie ejabberd (Erlang Jabber Daemon) gère des millions de connexions simultanées, CouchDB stocke et distribue des données à grande échelle, RabbitMQ route des messages entre applications distribuées. Ces succès valident l'approche d'origine : les contraintes des systèmes téléphoniques se retrouvent dans de nombreuses applications contemporaines.

Le langage répond initialement aux exigences particulières d'Ericsson,

mais ses solutions se révèlent pertinentes pour de nombreux problèmes actuels. Les choix architecturaux initiaux, le modèle de concurrence par passage de messages notamment, semblaient radicaux mais permettent aujourd'hui d'exploiter naturellement les architectures parallèles modernes.

IMAP

En 1970, qui aurait imaginé que l'échange de messages électroniques deviendrait l'une des applications les plus universelles d'Internet ? À cette époque, les systèmes de messagerie ressemblaient à des carnets d'adresses internes, confinés aux utilisateurs partageant une machine.

Les années 1980 transformèrent radicalement le paysage informatique. L'arrivée des stations de travail individuelles bouscula les habitudes établies. Les utilisateurs réclamaient désormais un accès nomade à leurs messages, depuis n'importe quel poste, tout en gardant une vision unifiée de leur courrier. Le protocole POP, lancé en 1984, répondait partiellement à cette attente en téléchargeant les messages localement. Mais cette solution montrait vite ses limites dès qu'il s'agissait de gérer les courriels stockés sur le serveur.

Mark Crispin observait ces évolutions depuis l'Université de Stanford. En 1986, il développa IMAP, une approche radicalement différente qui plaçait le serveur au cœur de la gestion des messages. La première version publique, IMAP2, vit le jour en 1988. L'innovation majeure tenait dans la capacité de manipuler directement les messages sur le serveur, sans téléchargement préalable. Cette architecture autorisait la création de multiples dossiers et proposait des fonctions de recherche sophistiquées dans le contenu des courriels.

L'architecture technique d'IMAP reposait sur une séparation claire entre deux composants : le système de gestion des messages (MHS) qui orchestrait les transferts, and l'agent utilisateur (UA) qui dialoguait avec l'interface. Cette modularité ouvrait un champ immense aux développeurs de clients de messagerie, libres d'innover sur l'expérience utilisateur sans se préoccuper des mécanismes de transport.

La décennie 1990 marqua l'adoption progressive d'IMAP par la communauté Internet. Le protocole s'enrichit au fil des versions : support des pièces jointes via MIME, mécanismes d'authentification renforcés, gestion des quotas d'espace disque. IMAP4 obtint le statut de standard Internet en 1994, consacrant sa maturité technique.

L'Université de Pittsburgh offre un exemple saisissant de cette transition. En 1997, l'établissement lança un projet pharaonique : migrer 25 000 utilisateurs vers IMAP. L'opération, bouclée en 2000, nécessita le développement d'outils sur mesure pour convertir les anciennes messageries, former les équipes et déployer une infrastructure serveur à la hauteur des enjeux. Cette migration illustre parfaitement les défis organisationnels que représentait l'adoption d'IMAP à grande échelle.

La sophistication technique d'IMAP transparaît dans sa gestion des états de connexion. Le protocole jongle entre trois modes : connecté, déconnecté, et synchronisation. Cette flexibilité s'avéra prophétique avec l'explosion des usages mobiles. IMAP savait déjà gérer les connexions intermittentes, optimiser les transferts de données et maintenir la cohérence entre client et serveur, des préoccupations qui deviendraient centrales avec l'avènement des smartphones.

Rétrospectivement, IMAP anticipait des technologies du cloud computing : centralisation du stockage, accès multi-périphériques, synchronisation transparente. Ces caractéristiques expliquent sa longévité remarquable et son adoption par la quasi-totalité des fournisseurs de messagerie, des géants du web aux hébergeurs locaux.

Les clients de messagerie ont largement contribué au succès d'IMAP. Pine, avec son interface spartiate mais efficace, Mulberry et ses fonctions avancées, puis Thunderbird et sa démocratisation ont tour à tour popularisé le protocole. L'intégration native dans les systèmes d'exploitation et les navigateurs web paracheva sa diffusion universelle.

L'impact d'IMAP sur les pratiques professionnelles dépasse le simple cadre technique. Le partage de boîtes aux lettres, l'accès simultané aux messages par plusieurs collaborateurs, la consultation nomade du courrier ont transformé l'organisation du travail. Bien avant l'ère du smartphone, IMAP rendait possible la messagerie mobile, condition sine qua non de la flexibilité professionnelle actuelle.

Les innovations techniques portées par IMAP ont essaimé vers d'autres domaines. La gestion fine des connexions réseau, la synchronisation robuste entre client et serveur, l'optimisation des flux de données ont inspiré de nombreux autres protocoles. Cette fécondité technique témoigne de la qualité de conception d'IMAP.

Malgré la concurrence des webmails et l'explosion des messageries instantanées, IMAP conserve une place de choix dans l'écosystème Internet. Sa capacité d'évolution tout en préservant la compatibilité, sa robustesse éprouvée et sa standardisation ouverte en font une technologie pérenne. Les développements actuels portent sur l'amélioration des performances, le renforcement de la sécurité et l'adaptation aux nouveaux usages comme

l'Internet des objets.

NSFNet

Vers la fin des années 1970, une frustration grandissante s'empare de la communauté scientifique américaine. ARPANET relie une douzaine d'universités privilégiées, mais le Département de la Défense refuse d'étendre ce réseau à d'autres institutions. Cette situation crée un fossé technologique entre les établissements connectés et les autres, privés des bénéfices de la communication numérique.

Larry Landweber saisit l'ampleur du problème. En mai 1979, il convoque une réunion à l'Université du Wisconsin pour chercher des solutions. L'assemblée découvre que les services de messagerie électronique et de transfert de fichiers transforment la façon de travailler des chercheurs. Déjà, quelques réseaux spécialisés montrent la voie : THEORYNET regroupe 200 informaticiens théoriciens, SAMNET unit 50 spécialistes de l'analyse de performances. Ces expériences limitées révèlent le potentiel des échanges électroniques pour la recherche.

Six mois plus tard, un consortium d'universités frappe à la porte de la National Science Foundation. Le projet paraît séduisant : créer un réseau national accessible à tous les départements d'informatique, avec des coûts raisonnables et une facturation à l'usage. L'infrastructure s'appuierait sur les réseaux commerciaux comme Telenet. Mais les évaluateurs restent sceptiques. N'y a-t-il pas redondance avec ARPANET ? Comment gérer un tel projet ? La NSF préfère commander une étude détaillée.

L'été 1980 voit naître un comité de planification rassemblant dix-neuf experts en réseaux. Deux découvertes changent la donne. Le logiciel MMDF, conçu à l'Université du Delaware, transporte les messages sur différents supports, incluant ARPANET et les lignes téléphoniques. En parallèle, la DARPA adopte les protocoles internet qui autorisent la communication entre réseaux distincts. Ces innovations techniques ouvrent des perspectives inédites : plusieurs réseaux physiques peuvent former une organisation logique unique.

Une nouvelle proposition germe à l'automne 1980. Le consortium s'enrichit des universités du Wisconsin, Purdue, Utah, Delaware et de la société Rand Corporation. Le National Science Board approuve le projet début 1981, mais impose une condition : la NSF dirigera l'opération pendant deux ans avant de la confier à une structure indépendante. Les contrats sont signés au printemps.

En 1985, la NSF lance un défi plus audacieux : connecter ses centres de supercalculateurs dispersés sur le territoire. NSFNET voit le jour en 1986, avec des liaisons à 56 kbps reliant six sites stratégiques. L'innovation technique repose sur des mini-ordinateurs LSI-11/73 équipés du logiciel Fuzzball. Ce système intègre les protocoles internet avec des mécanismes sophistiqués de routage et de gestion des embouteillages.

Le succès dépasse les espérances. Dès 1988, la saturation des liens impose une montée en débit vers 1,5 Mbps. Le réseau s'étend aux réseaux régionaux académiques, créant une toile d'interconnexions complexe. Les nœuds Fuzzball orchestrent le routage adaptatif, synchronisent les horloges avec précision et répartissent équitablement les ressources. Des agents logiciels filtrent les informations de routage entre le cœur du réseau et sa périphérie.

La montée en puissance d'internet transforme le paysage. Entre 1993 et 1998, NSFNET mue vers une architecture commerciale. Les fournisseurs privés se multiplient, poussant la NSF à repenser l'organisation du réseau en 1993. Cette nouvelle structure perdure aujourd'hui. Des contrats signés en 1995 établissent des points d'interconnexion entre réseaux commerciaux. En avril 1995, le service public NSFNET ferme ses portes, remplacé par un maillage de réseaux privés.

Cette transition s'accompagne d'un transfert de responsabilités inattendu. Depuis les origines, le Département de la Défense gère les noms de domaine pour ses utilisateurs militaires. Au début des années 1990, les institutions académiques représentent la majorité des nouvelles inscriptions. Le Federal Networking Council confie alors cette mission à la NSF. Face à l'explosion de la demande, l'enregistrement est rendu payant en septembre 1995. Les chiffres donnent le vertige : 7 500 domaines en 1993, 2 millions en 1998.

L'année 1998 marque la privatisation complète des fonctions critiques d'internet. L'ICANN (Internet Corporation for Assigned Names and Numbers) naît pour superviser le système des noms de domaine. La NSF se recentre sur son cœur de métier : soutenir la recherche sur les protocoles et applications réseaux. Elle continue notamment d'aider les institutions isolées à se connecter.

NSFNET a métamorphosé internet. D'un réseau militaire et universitaire confidentiel, il est devenu l'épine dorsale d'une infrastructure planétaire. Son architecture distribuée et ses mécanismes de collaboration public-privé ont modelé l'organisation actuelle du réseau mondial. Les innovations techniques développées pour ce projet – routage adaptatif, synchronisation temporelle précise – restent des piliers d'internet aujourd'hui.

Au-delà de la technique, NSFNET a inventé un modèle de gouvernance

unique. Universités, entreprises et organismes publics collaborent pour préserver l'ouverture du réseau tout en assurant sa croissance. Cette réussite démontre comment une initiative publique peut déclencher le développement d'une infrastructure devenue indispensable à l'économie numérique moderne.

SCSI

Dans les années 1960, IBM travaillait sur son ordinateur central 360 et développa un bus d'entrées-sorties innovant pour l'époque : il pouvait communiquer simultanément avec plusieurs périphériques. Ce bus évolua pour devenir le Canal OEM, mais quand IBM le présenta à l'ANSI pour en faire une norme, l'institut refusa. La raison ? Son caractère trop propriétaire. L'ANSI préférait voir naître un bus parallèle d'entrées-sorties qui répondrait aux besoins commerciaux sans dépendre d'un seul constructeur.

L'histoire prit un tournant inattendu au début des années 1980. Chez Shugart Associates, fabricant de disques durs, quelques ingénieurs qui allaient plus tard fonder Adaptec mirent au point une interface parallèle qu'ils baptisèrent SASI (Shugart Associates System Interface). Cette spécification circula parmi les constructeurs et rencontra un franc succès commercial. En 1982, forte de cette adoption, elle fut présentée à l'ANSI comme base pour une nouvelle norme. L'institut saisit l'opportunité, formalisa et étendit les spécifications du SASI, mais changea son nom en SCSI pour éviter toute référence à un constructeur particulier. Le 18 juin 1986, le SCSI devenait officiellement une norme ANSI.

Le SCSI se distinguait par sa polyvalence : il pouvait contrôler une multitude de périphériques, des disques durs aux imprimantes en passant par les lecteurs de CD-ROM et les scanners. Son architecture interconnectait les petits ordinateurs avec leurs périphériques intelligents, notamment les systèmes de stockage. Les performances étaient remarquables : jusqu'à 4 Mo/s selon l'implémentation, avec une portée qui atteignait 25 mètres grâce aux pilotes et récepteurs différentiels.

Ce qui rendait le SCSI astucieux, c'était son protocole d'adressage logique plutôt que physique pour tous les blocs de données. Chaque unité logique pouvait être interrogée pour connaître sa capacité en blocs. Cette abstraction simplifiait considérablement la gestion et le remplacement des périphériques. Le protocole prévoyait la connexion de multiples initiateurs et cibles, avec un système d'arbitrage distribué intégré directement dans son architecture.

Les années 1990 révélèrent une fracture intéressante dans le monde informatique. Les Macintosh adoptèrent massivement le SCSI pour connecter leurs périphériques, tandis que les PC restaient fidèles à l'interface IDE/ATA. Cette différence n'était pas anodine : presque tous les composants des Mac se raccordaient via SCSI, alors que l'IDE ne gérait que les disques durs. Sur PC, il fallait jongler avec des contrôleurs spécifiques pour les lecteurs de CD-ROM (souvent intégrés aux cartes son) et brancher les lecteurs de bandes sur le contrôleur de disquettes. Le SCSI évitait cette multiplication des interfaces grâce à son bus unique.

La course aux performances poussa constamment la technologie vers de nouveaux sommets. Le SCSI original fonctionnait avec un bus 8 bits cadencé à 5 MHz, ce qui donnait environ 5 Mo/s. Pour aller plus vite, deux voies s'offraient : élargir le bus ou accélérer l'horloge. Ces améliorations successives donnèrent naissance au SCSI-2, qui introduisit des variantes comme le Wide SCSI (bus 16 bits) et le Fast SCSI (horloge 10 MHz).

Le HVD (High Voltage Differential) fit son apparition dans la norme SCSI-2 pour repousser les limites de distance. Cette technique utilisait une broche supplémentaire qui garantissait une transmission précise, permettant d'atteindre 25 mètres de câblage. L'Ultra SCSI, qui relevait de la norme SCSI-3, perfectionna les concepts du SCSI-2. Les débits continuèrent leur ascension avec le Wide Ultra SCSI, l'Ultra2 SCSI et le Wide Ultra2 SCSI. L'Ultra3 SCSI doubla carrément la fréquence d'horloge tout en conservant sa compatibilité avec le SCSI-3.

La vitesse du SCSI tenait beaucoup à son protocole sous-jacent, le SCSI Block Commands (SBC). Ce système autorisait la transmission continue de données entre périphériques avec une seule commande. Prenons l'exemple d'un CD audio : le contrôleur SCSI envoyait un signal SBC au lecteur, qui diffusait ensuite les données vers le contrôleur hôte jusqu'à la fin du disque ou jusqu'à recevoir un signal d'arrêt. Le lecteur pouvait rediriger directement ses données vers un disque dur, court-circuitant le contrôleur et le bus système pour économiser la bande passante.

Pourtant, malgré ses qualités indéniables, le SCSI finit par perdre la bataille du marché grand public face à l'IDE/ATA. L'arrivée de l'ATAPI (AT Attachment Packet Interface) changea la donne : les périphériques IDE pouvaient désormais gérer d'autres appareils que les simples disques durs. Cette extension du standard IDE, couplée au coût élevé des contrôleurs SCSI, précipita l'abandon progressif de cette interface dans les ordinateurs personnels. Mais le SCSI ne disparut pas pour autant : il conserva sa place dans les systèmes professionnels et les serveurs, où ses caractéristiques techniques restaient incontournables.

L'innovation ne s'arrêta pas là. L'iSCSI permit d'envoyer des com-

mandes SCSI sur les réseaux locaux, étendus ou Internet, révolutionnant l'accès et la gestion à distance des périphériques SCSI. L'influence de cette interface se retrouva aussi dans d'autres technologies : l'IEEE-1394 (FireWire) s'appuya sur un sous-ensemble des spécifications SCSI-3 pour contrôler ses périphériques, et les concentrateurs FireWire ressemblaient étrangement à des adaptateurs SCSI simplifiés.

Aujourd'hui, l'héritage du SCSI imprègne encore le stockage informatique moderne. Son approche d'adressage logique et son architecture de commandes ont inspiré les interfaces qui lui ont succédé. Sa capacité à gérer efficacement des périphériques et sa robustesse dans les environnements professionnels en ont fait une technologie fondatrice.

Bash

Dans l'univers des shells UNIX, peu d'histoires rivalisent avec celle du Bourne-Again Shell. Bash tire son nom d'un jeu de mots avec Steve Bourne, l'homme derrière le shell original d'UNIX (/bin/sh) de la septième édition de Bell Labs. Brian Fox, travaillant pour la Free Software Foundation, lance ce projet avec une ambition claire : créer un interpréteur de commandes libre qui deviendrait le cœur du système GNU.

L'exercice s'avère plus délicat qu'il n'y paraît. Recréer un shell compatible avec celui de Bourne ne se limite pas à développer les fonctionnalités documentées. Fox et son équipe découvrent que les utilisateurs Unix exploitent tous les recoins du shell original, y compris ses comportements les plus obscurs et ses subtilités non documentées. La grammaire du Bourne shell manquait de précision, transformant chaque détail en défi technique.

Malgré ces difficultés, Bash s'enrichit. Les développeurs puisent dans le meilleur des autres shells : les fonctionnalités avancées du Korn shell (ksh) et les innovations du C shell (csh) viennent compléter la base compatible POSIX. Cette approche hybride donne naissance à un outil qui respecte les standards tout en offrant une expérience utilisateur moderne.

Quand Chet Ramey de l'université Case Western Reserve reprend les rênes du projet, Bash entre dans une phase de maturation. Ramey apporte une vision long terme et une méthode de développement rigoureuse. Sous sa direction, le shell gagne en stabilité et en fonctionnalités. La version 1.13 adopte le fonctionnement « eight-bit clean ». Jusque-là, Bash utilisait le huitième bit des caractères pour marquer leur statut lors des expansions, ce qui limitait la manipulation de fichiers contenant des caractères accentués

ou des symboles spéciaux. Cette correction ouvre la voie à une meilleure internationalisation.

L'interface de Bash révèle l'attention portée à l'expérience utilisateur. La bibliothèque readline transforme la saisie de commandes en véritable éditeur de texte intégré. Les utilisateurs d'emacs ou de vi retrouvent leurs raccourcis clavier familiers directement dans le shell. La complétion automatique devine les noms de fichiers, les commandes disponibles et les variables d'environnement. L'historique des commandes, héritage du C shell, transforme chaque session en une base de données personnelle d'instructions réutilisables.

L'une des forces de Bash réside dans la personnalisation. Chaque utilisateur peut façonner son environnement selon ses préférences grâce aux fichiers `.bash_profile` et `.bashrc`. L'invite de commande se pare de couleurs et d'informations contextuelles grâce aux séquences d'échappement. Certains affichent l'heure, d'autres le répertoire courant ou le niveau de charge du système. Cette flexibilité transforme le terminal de simple interface en véritable tableau de bord personnalisé.

La gestion des tâches héritée du Korn shell apporte une dimension multitâche au shell. Les utilisateurs jonglent entre plusieurs processus, les suspendent, les reprennent ou les relancent en arrière-plan. Les alias raccourcissent les commandes fréquentes, tandis que l'expansion des accolades multiplie les possibilités de génération de chaînes de caractères. Taper « file{1,2,3}.txt » génère automatiquement file1.txt, file2.txt et file3.txt.

La distribution de Bash illustre les méthodes de diffusion du logiciel libre des années 1990. Le code source circule via FTP anonyme depuis les serveurs du MIT et ses miroirs dispersés dans le monde entier. La Free Software Foundation commercialise des bandes magnétiques et des CD-ROM pour ceux qui préfèrent un support physique. Les premières distributions Linux adoptent une stratégie double : une version complète riche en fonctionnalités cohabite avec une version minimale destinée à remplacer directement /bin/sh.

L'évolution technique ne s'arrête jamais. Bash acquiert le support des tableaux unidimensionnels, copiant cette fonctionnalité du Korn shell. La complétion programmable se sophistique, permettant aux développeurs de spécifier des comportements sur mesure selon le contexte. Le chargement dynamique de commandes internes étend les capacités du shell pendant son exécution, transformant l'interpréteur en plateforme d'extension.

Grâce à sa portabilité remarquable, Bash tourne sur OS/2, DOS et Windows NT. QNX et Minix l'intègrent dans leurs distributions standard. Cette ubiquité renforce sa position de shell de référence et familiarise une génération d'utilisateurs à ses spécificités.

La communauté d'utilisateurs façonne Bash autant que ses développeurs officiels. Des milliers de retours d'expérience alimentent les cycles de développement. Les corrections de bogues arrivent du monde entier, transformant chaque version en effort collectif. Cette dynamique collaborative donne naissance à un outil exceptionnellement robuste et adapté aux besoins réels.

Bash finit par devenir bien plus qu'un simple shell : il incarne l'esprit UNIX dans sa version libre et démocratisée. Sa documentation complète, ses pages de manuel détaillées et sa maintenance active en font un compagnon fiable pour des générations de programmeurs et d'administrateurs système. Dans les serveurs web, les scripts de sauvegarde ou les chaînes d'intégration continue, Bash continue de traiter silencieusement des millions de tâches quotidiennes, héritier direct d'une tradition vieille de plusieurs décennies.

GIF

En 1987, CompuServe lance le format GIF (Graphics Interchange Format) avec une ambition simple : créer un système d'échange d'images qui fonctionne sur toutes les plateformes. À cette époque, partager un fichier graphique entre différents systèmes relève souvent du parcours du combattant. CompuServe mise sur l'ouverture : les spécifications du format sont publiées librement, ce qui accélère son adoption dans la communauté informatique.

La première mouture, baptisée GIF87a, intègre l'algorithme de compression LZW développé par Terry Welch trois ans plus tôt. Ce procédé comprime les images sans perdre d'informations. Le format accepte jusqu'à 256 couleurs indexées, une limitation qui paraît contraignante mais qui révélera plus tard son potentiel créatif inattendu. Deux ans après, CompuServe perfectionne son format avec la version GIF89a, qui ajoute la transparence et surtout la possibilité de créer des animations par succession d'images.

L'idylle tourne court en 1994. Unisys, propriétaire du brevet sur l'algorithme LZW, décide brutalement d'exiger des redevances pour toute utilisation commerciale. La nouvelle fait l'effet d'une bombe dans l'industrie du logiciel. De nombreux développeurs se retrouvent pris au piège, contraints de revoir leurs modèles économiques ou d'abandonner purement et simplement le support du format. Cette crise stimule paradoxalement l'innovation : elle pousse à la création du format PNG en 1996, conçu comme une alternative libre au GIF.

Mais le GIF résiste et se métamorphose. Avec l'explosion du Web, ses prétendues faiblesses se muent des forces. Cette palette de 256 couleurs, jugée archaïque par les puristes, inspire les artistes qui y trouvent une esthétique minimaliste et expressive. L'animation en boucle, pensée au départ comme une simple fonctionnalité technique, est depuis la signature du format.

Durant les années 2000, les outils de création se démocratisent tandis que les premières plateformes sociales encouragent le partage de contenus visuels. Le GIF trouve sa vocation : devenir un langage universel de l'émotion instantanée. Les « reaction GIFs », ces extraits de films ou séries transformés en boucles expressives, révolutionnent la communication en ligne. Quelques images suffisent désormais à exprimer ce que des paragraphes peinent parfois à dire.

Tumblr cristallise ce phénomène en encourageant la création et la réappropriation des contenus. Sur cette plateforme, le GIF dépasse son statut d'outil pour devenir un medium artistique à part entière. Les créateurs explorent ses possibilités, produisant des œuvres qui oscillent entre art cinétique et commentaire social. Une génération numérique grandit avec ce format, privilégiant l'impact visuel immédiat à la complexité narrative.

L'expiration des brevets LZW entre 2003 et 2004 libère définitivement le format de ses entraves juridiques. Cette émancipation coïncide avec l'avènement des smartphones et des réseaux sociaux mobiles. Le GIF s'avère parfaitement adapté aux contraintes de la communication nomade : compact, universel, instantané. Sa compatibilité avec tous les appareils en fait un dénominateur commun dans un écosystème technologique fragmenté.

Le monde de l'art finit par reconnaître l'importance culturelle du format. Le MoMA expose des GIFs dans ses collections, légitimant ce medium longtemps considéré comme mineur. Cette reconnaissance institutionnelle accompagne l'émergence de nouvelles pratiques artistiques comme le *cinemagraph*, qui mélange photographie statique et animation subtile pour créer des images troublantes de réalité suspendue.

Malgré l'apparition de formats techniquement supérieurs comme WebM ou MP4, le GIF maintient sa position. Sa simplicité constitue paradoxalement sa force : pas besoin de codecs complexes, pas de problèmes de compatibilité, pas de licences restrictives. Cette accessibilité technique se double d'une accessibilité culturelle qui en fait un outil d'expression démocratique.

L'histoire du GIF illustre un phénomène rare dans l'univers technologique : l'appropriation sociale d'un format technique au-delà de ses intentions premières. Ce qui devait servir uniquement à échanger des images statiques est devenu un élément central de la culture numérique contemporaine. Le format porte en lui des pratiques créatives spécifiques, une

esthétique reconnaissable, tout un vocabulaire visuel partagé.

Cette longévité exceptionnelle dans un environnement en mutation perpétuelle souligne l'importance des standards ouverts et simples. Le GIF démontre que les innovations durables ne sont pas forcément les plus sophistiquées, mais celles qui laissent aux utilisateurs la liberté de se les approprier et de les détourner selon leurs besoins.

Le format occupe une position singulière à la croisée de l'art, de la communication et de la culture populaire. Son influence déborde largement le cadre technique pour se répandre dans les pratiques sociales et médiatiques contemporaines. Trente-cinq ans après sa création, le GIF continue d'évoluer, porté par les usages créatifs d'une communauté mondiale qui y trouve un moyen d'expression à la fois accessible et sophistiqué.

Perl

En 1987, Larry Wall travaille au Jet Propulsion Laboratory de la NASA. Linguiste de formation, il passe ses journées face aux écrans UNIX de l'époque, jonglant entre `sed`, `awk` et les scripts shell pour traiter des données. Un jour, las de cette gymnastique quotidienne entre outils disparates, il décide de créer son propre langage. Perl naît de cette frustration pratique : faire dialoguer les programmes UNIX sans passer par mille contorsions.

Il n'invente rien de nouveau, mais puise dans ce qui existe déjà, comme les expressions régulières de sed, la puissance de traitement d'`awk`, la syntaxe familière du C. Mais il assemble ces éléments avec un œil neuf et fait de Perl ce langage « colle » qui transforme la corvée de conversion de données en jeu d'enfant. Wall avait montré ses talents de bricoleur génial avec `rn`, l'un des premiers lecteurs de nouvelles Usenet, et patch, cet utilitaire qui révolutionne la distribution de correctifs logiciels.

La philosophie de Perl tient en quelques mots : l'efficacité prime sur l'élégance. Là où d'autres langages imposent leur vision rigide, Perl s'adapte au programmeur. Son credo, « Il existe plus d'une façon de le faire », est une souplesse qui déconcerte parfois, mais qui séduit une communauté grandissante d'administrateurs systèmes et de bidouilleurs en tout genre.

Au début des années 1990, tandis que Netscape popularise la navigation graphique, Perl s'impose comme l'outil indispensable pour traiter les formulaires HTML, bien avant PHP. Sa capacité à décortiquer le texte, ses performances correctes sur de petits scripts, sa disponibilité sur toutes les plateformes en font le compagnon idéal des premiers développeurs web.

Amazon, Yahoo !, Slashdot : tous ces sites qui façonnent l'internet s'appuient sur Perl.

L'évolution du langage suit un rythme tranquille mais sûr. La version 2 améliore les expressions régulières, la version 3 gère les données binaires. Puis vient 1994 et la version 5. Wall repense la syntaxe, ajoute un support objet convenable, introduit des modules. Cette version de référence est celle que les développeurs utilisent, des décennies plus tard.

Mais Perl ne se limite pas aux caprices de son créateur. Une communauté se forme autour du langage, organisée de façon originale. Les « porteurs » se partagent la maintenance, coordonnés par la liste de diffusion perl5-porters. Pour chaque version, l'un d'eux hérite de la « citrouille des correctifs », supervisant les modifications et les publications. Cette gouvernance informelle fonctionne remarquablement bien.

Le vrai trésor de Perl, c'est le CPAN. Cette archive de modules réutilisables transforme le langage en couteau suisse numérique. Besoin de traiter des images ? De se connecter à une base de données ? De parser du XML ? Il y a forcément un module CPAN pour ça. Cette richesse compense largement les défauts du langage de base.

Wall, avec sa formation de linguiste, insuffle à Perl une dose d'humanité. Le langage tolère l'ambiguïté, s'adapte au contexte, pardonne les approximations. Un programme Perl ressemble parfois à de la prose anglaise mal ponctuée. Cette expressivité séduit, mais elle rebute aussi : lire du Perl écrit par un autre développeur relève parfois du déchiffrage cryptographique.

La question des licences agite alors la communauté du logiciel libre. Wall tranche en proposant une double licence : GPL d'un côté, Artistic License de l'autre, chacun choisit selon ses besoins. Cette souplesse juridique annonce l'approche open source qui émerge à la fin des années 1990.

Les entreprises flairent le filon. O'Reilly & Associates publie des manuels qui seront des références. ActiveState adapte Perl pour Windows, brisant le monopole UNIX. Ces initiatives commerciales n'entament pas l'esprit communautaire du projet. Au contraire, elles le renforcent en apportant ressources et visibilité.

Le nouveau millénaire apporte son lot de concurrents. Python séduit par sa clarté, Ruby par son élégance objet. Ces langages s'inspirent de Perl tout en corrigeant ses défauts supposés. La communauté Perl ne reste pas les bras croisés. En 2000, elle lance le projet Perl 6, refonte ambitieuse du langage. Trop ambitieuse peut-être. Le projet s'enlise, change de nom pour devenir Raku, tandis que Perl 5 continue sa route, imperturbable.

Car Perl 5 a trouvé sa niche. Les biologistes l'utilisent pour analyser l'ADN, les administrateurs pour leurs scripts quotidiens, certains sites web continuent de s'en satisfaire. Le langage a perdu son statut de star, mais il

conserve une communauté fidèle et compétente. Ses conférences rassemblent encore des centaines de passionnés, ses modules continuent d'enrichir le CPAN.

Perl a montré qu'un projet libre pouvait rivaliser avec les solutions commerciales, qu'une communauté dispersée pouvait maintenir un logiciel complexe, qu'un langage imparfait pouvait rendre d'immenses services. Dans les serveurs qui font tourner l'internet d'aujourd'hui, quelque part, du code Perl continue sûrement de traiter vos données, discrètement, efficacement.

Self

David Ungar et Randall Smith lancent Self en 1987, un projet qui va bousculer les idées reçues. Ce nouveau langage refuse délibérément tout ce qui semblait acquis à l'époque. Fini les classes, oubliées les variables. Seuls restent les objets et leurs conversations par messages.

Cette radicalité n'est pas gratuite. Elle naît d'une frustration face à la complexité croissante des systèmes orientés objet. Smalltalk-80, pourtant admiré pour son élégance, traîne le poids de ses méta-classes et de leurs imbrications vertigineuses. Self propose une vision différente, celle de ne pas s'embarrasser de moules quand on peut directement sculpter dans la matière.

Dans cet univers épuré, chaque objet est un conteneur de « slots » nommés. Ces emplacements stockent indifféremment données et comportements. L'héritage s'appuie sur de simples pointeurs « parent » qui créent des chaînes de délégation entre objets. Un système si simple qu'il en est déroutant.

La création d'objets suit une logique naturelle. Au lieu de consulter le plan d'une classe, Self clone directement un prototype existant. Cette approche résout d'un coup la fameuse régression infinie des méta-classes qui donnait des sueurs froides aux concepteurs de Smalltalk. Un objet sert de modèle à d'autres objets, point final.

L'uniformité constitue l'autre pilier de cette architecture. Qu'un objet demande une information stockée ou qu'il déclenche un calcul complexe, la syntaxe est identique : l'envoi d'un message. Cette cohérence gomme la distinction traditionnelle entre propriétés et méthodes. Le programmeur navigue dans un monde homogène où chaque interaction obéit aux mêmes règles.

Les premières implémentations de Self surprennent par leurs performances. Dès 1989, le langage rattrape puis dépasse Smalltalk en vitesse d'exécution. Ce succès cache une innovation technique discrète mais décisive. L'équipe invente les « maps », structures internes qui optimisent l'organisation des objets en mémoire. Ces cartes cachées sont ce qu'on appellera plus tard les « hidden classes ».

La compilation personnalisée pousse l'optimisation encore plus loin. Au lieu de produire du code générique, Self génère du code machine taillé sur mesure pour chaque type de destinataire. Cette spécialisation dynamique transforme les performances du langage.

Les caches d'appels polymorphes voient le jour en 1991. Ces PIC mémorisent plusieurs résolutions de méthodes pour chaque site d'appel. Concrètement, quand un message peut aboutir à différents traitements selon le contexte, le système garde en réserve les solutions déjà trouvées. Les appels polymorphes, traditionnellement coûteux, sont soudain abordables.

La version de 1996 franchit un nouveau cap avec le « type feedback » et l'optimisation adaptative. Le premier mécanisme intègre directement les appels dynamiques dans le code compilé, réduisant de 70% les invocations de méthodes. Le second observe le programme en marche et retravaille à la volée les passages les plus sollicités.

Ces avancées techniques attirent l'attention de Sun Microsystems. En 1991, la société recrute Ungar et Smith pour poursuivre leurs recherches. Mais l'histoire prend une tournure inattendue quand deux figures clés du projet, Urs Hölzle et Lars Bak, quittent Sun en 1994 pour créer Animorphic Systems. Leur mission : développer Strongtalk, un Smalltalk haute performance.

Le rachat d'Animorphic par Sun en 1997 scelle le destin des innovations de Self. Elles migrent vers HotSpot, la machine virtuelle qui va propulser Java vers les sommets. Lars Bak récidive en 2008 avec V8, le moteur JavaScript de Google, qui puise lui aussi dans l'héritage de Self.

Cette filiation technique ne constitue qu'une partie de l'histoire. Self influence aussi la conception des langages. JavaScript adopte les prototypes comme mécanisme d'héritage principal. Lua, Io et d'autres langages modernes s'inspirent de cette approche directe où les objets engendrent d'autres objets sans intermédiaire.

La philosophie du concret de Self se reflète dans son environnement de développement. Les objets y apparaissent comme des entités manipulables, visibles et modifiables en temps réel. Cette interface graphique novatrice transforme la programmation en activité tangible. On touche, on déplace, on transforme. L'abstraction cède le pas à la manipulation directe.

Simplicité, uniformité, caractère concret sont les trois principes guident

cette vision. Réduire le nombre de concepts, traiter toutes les opérations de façon identique, utiliser des métaphores compréhensibles. Ces idées simples produisent un environnement d'une richesse surprenante.

Self n'a jamais conquis les masses de programmeurs. Trop radical pour certains, trop expérimental pour d'autres, le langage est confiné aux laboratoires de recherche. Pourtant, son influence traverse les décennies. Les techniques d'optimisation qu'il a introduites permettent aujourd'hui aux langages dynamiques de rivaliser avec leurs cousins compilés. Python, Ruby, PHP bénéficient de ses innovations.

Cette histoire singulière montre comment une quête de simplicité peut déclencher une cascade d'innovations durables. Self a démontré qu'il fallait parfois tout remettre à plat pour avancer. Sa conception minimaliste anticipe certaines tendances actuelles où l'épurement des concepts prime sur l'accumulation de fonctionnalités.

L'héritage technique de Self alimente les machines virtuelles les plus performantes du marché. Sa philosophie inspire les concepteurs en quête d'élégance. Quarante ans après sa naissance, Self reste une source d'inspiration pour qui cherche à repenser la programmation orientée objet.

Tcl

John Ousterhout et ses étudiants de l'Université de Californie à Berkeley se heurtaient à un problème récurrent. Ils concevaient des outils interactifs pour la création de circuits intégrés – Magic, Crystal et d'autres – et chacun réclamait son propre langage de commandes. L'équipe consacrait peu d'efforts à ces langages. Le résultat se révélait prévisible avec des syntaxes limitées, incohérentes entre elles, qui finissaient par devenir gênantes.

La solution germa durant l'automne 1987, pendant qu'Ousterhout passait son congé sabbatique au laboratoire de recherche DEC. Pourquoi ne pas créer un interpréteur unique, pensé dès le départ comme une bibliothèque que n'importe quelle application pourrait intégrer ? Ce langage fournirait les bases – variables, boucles, procédures – et chaque programme y ajouterait ses propres commandes spécialisées. Cette extensibilité deviendrait la marque de fabrique de Tcl, le « Tool Command Language ».

Les premiers développements démarrèrent début 1988. Quelques mois plus tard, un éditeur de texte graphique utilisait déjà Tcl. Ousterhout ne s'attendait pas à ce que d'autres développeurs s'y intéressent – erreur de jugement qui l'amuserait plus tard.

Pendant ce temps, les interfaces graphiques compliquaient drastiquement la vie des programmeurs. Les années 1980 voyaient naître des environnements visuels de plus en plus sophistiqués, mais leur développement demandait des investissements colossaux. Ousterhout s'inquiétait : les petites équipes de recherche allaient-elles encore pouvoir innover ? La seule issue consistait à assembler des systèmes complexes à partir de briques réutilisables.

Cette réflexion déboucha sur Tk, l'extension graphique de Tcl. L'idée était simple : construire des interfaces en combinant des composants prêts à l'emploi, avec Tcl comme ciment entre les éléments. Fin 1988, le développement de Tk commençait. Il faudrait deux années pour obtenir quelque chose de vraiment utilisable.

En 1989, Tcl sortait de l'ombre. Ousterhout le présenta en janvier 1990 à la conférence USENIX, où l'accueil fut enthousiaste. Il décida alors de mettre le code source sur le serveur FTP de Berkeley. Internet fit le reste : le bouche-à-oreille propagea Tcl à une vitesse surprenante.

Don Libes du National Institute of Standards and Technology créa la première application Tcl à connaître un succès retentissant. Expect automatisait les programmes UNIX interactifs – une bénédiction pour les administrateurs systèmes qui passaient leurs journées à jongler entre terminaux et scripts. Le succès d'Expect fit connaître Tcl bien au-delà des cercles universitaires.

La croissance devint exponentielle au début des années 1990. D'une poignée d'utilisateurs en 1989, on passa à plusieurs dizaines de milliers en 1993. Deux raisons expliquent cet engouement. Tk représentait le moyen le plus direct de créer des interfaces graphiques sous UNIX – cinq à dix fois plus rapide qu'avec Motif. Et Tcl répondait aux besoins de développeurs qui voulaient rendre leurs applications scriptables sans s'embarrasser de créer leur propre langage.

En 1994, Ousterhout quittait Berkeley pour Sun Microsystems. L'entreprise lui donna les moyens de ses ambitions : une équipe dédiée qui atteignit douze personnes en trois ans. Cette période fut prolifique. Tcl s'émancipait d'UNIX pour conquérir Windows et Macintosh. Le système d'entrées-sorties fut refondu, les sockets réseau intégrés. Un compilateur transformait désormais les scripts en *bytecode*. Safe-Tcl apportait un modèle de sécurité, tandis qu'un plugin permettait d'exécuter des scripts directement dans les navigateurs web.

1998 marqua la reconnaissance officielle. Tcl reçut le prix ACM du meilleur logiciel système, distinction réservée aux programmes ayant marqué durablement l'informatique. La même année, le prix USENIX STUG venait saluer l'excellence technique de l'outil.

Fin 1997, Ousterhout avait créé Scriptics pour se consacrer entièrement à Tcl. La société développait TclPro, une suite d'outils commerciaux, tout en continuant de distribuer gratuitement le cœur de Tcl. La version 8.1 d'avril 1999 introduisit Unicode, la gestion des threads et un nouveau moteur d'expressions régulières.

Le nom Scriptics disparut en mai 2000, remplacé par Ajuba Solutions pour refléter l'orientation vers XML. Interwoven racheta l'entreprise en octobre. Pour garantir l'indépendance du développement, une équipe centrale autonome – la Tcl Core Team – prit le relais durant l'été 2000.

L'histoire de Tcl témoigne de la trajectoire imprévisible des innovations techniques. Un projet universitaire modeste peut devenir un outil majeur si sa conception répond à des besoins réels et si une communauté s'en empare. Tcl prouva l'importance de rendre les applications scriptables et démontra l'intérêt des composants réutilisables pour les interfaces graphiques.

Acorn Archimedes

À l'automne 1987, Acorn dévoile l'Archimedes, un ordinateur qui va bouleverser les codes de l'informatique personnelle. Derrière ce nom emprunté au célèbre mathématicien grec se cache une machine équipée d'un processeur ARM qui inaugure l'ère du RISC grand public.

L'histoire commence quelques années plus tôt, quand les ingénieurs d'Acorn cherchent un successeur 32 bits au vénérable 6502. Après avoir examiné les processeurs existants sur le marché, ils concluent qu'aucun ne répond à leurs exigences. La décision tombe : ils vont concevoir leur propre puce. Cette audace technique va donner naissance à l'ARM, un processeur d'une simplicité déconcertante mais d'une efficacité redoutable.

L'architecture RISC de l'ARM rompt avec la complexité croissante des processeurs de l'époque. Ses 44 instructions se contentent, pour la plupart, de s'exécuter en un cycle d'horloge. Cette économie de moyens produit des résultats spectaculaires : avec seulement 4 MHz, l'ARM délivre environ quatre MIPS, surpassant bien des processeurs plus rapides mais moins optimisés. Les 27 registres 32 bits complètent cette architecture élégante, conçue pour fonctionner avec de la mémoire dynamique standard.

Acorn décline l'Archimedes en deux gammes distinctes. La série A300 vise le marché éducatif, héritier naturel du BBC Micro qui équipe déjà de nombreuses écoles britanniques. Ces machines privilégient la simplicité : pas de contrôleur de disque dur intégré, mais l'essentiel pour apprendre et

découvrir. La série A400 s'adresse aux utilisateurs plus exigeants avec son contrôleur de disque dur et ses capacités d'extension étoffées.

L'A305 ouvre le bal avec 512 Ko de RAM, tandis que l'A310 double la mise avec 1 Mo. Au sommet, l'A440 impressionne avec ses 4 Mo de mémoire et son disque dur de 20 Mo, une configuration qui ferait pâlir bien des machines concurrentes. Ces chiffres reflètent l'ambition d'Acorn : créer un ordinateur personnel capable de rivaliser avec les stations de travail professionnelles.

Le génie de l'Archimedes réside dans son intégration. Quatre circuits spécialisés orchestrent le ballet électronique : l'ARM pour les calculs, le VIDC pour l'affichage, le MEMC pour la mémoire et l'IOC pour les entrées-sorties. Cette approche réduit drastiquement le nombre de composants tout en optimisant les performances. Le résultat se mesure à l'écran : 256 couleurs simultanées, des résolutions jusqu'à 640x512 pixels en couleur ou 1152x976 en monochrome, et un système sonore stéréo huit canaux qui fait de l'Archimedes une machine multimédia avant la lettre.

En 1989, Acorn renouvelle sa stratégie avec l'A3000, une machine compacte qui intègre le clavier dans le boîtier principal. Ce format rappelle l'Amiga 500 ou l'Atari 520ST, mais les performances de l'ARM font la différence. La série A400/1 corrige les défauts de jeunesse des premiers modèles et améliore la gestion mémoire. L'A540, lancé l'année suivante, franchit un nouveau cap avec son ARM 3 plus véloce et ses capacités étendues jusqu'à 16 Mo de RAM.

Le système d'exploitation suit cette évolution matérielle ambitieuse. Arthur, le premier OS de l'Archimedes, hérite de l'approche spartiate du BBC Micro. Son interface en ligne de commande rebute les néophytes mais séduit les programmeurs par sa puissance. Acorn comprend que l'avenir appartient aux interfaces graphiques et lance RISC OS en 1988.

RISC OS transforme l'expérience utilisateur. Son interface multitâche coopérative privilégie la fluidité à la robustesse absolue. Les applications se partagent civilement le processeur, une approche qui fonctionne remarquablement bien grâce aux performances de l'ARM. Le menu contextuel activé par le bouton central de la souris en fait une signature ergonomique, imité plus tard par d'autres systèmes.

L'architecture modulaire de RISC OS autorise une extensibilité remarquable. Les modules, qu'ils résident en ROM ou se chargent en mémoire, enrichissent les fonctionnalités sans toucher au noyau. Cette souplesse facilite l'intégration de nouveaux périphériques et l'évolution du système. Le système de fichiers unifié gomme les différences entre disquettes, disques durs, CD-ROM et réseau Econet, une approche moderne qui anticipe les besoins futurs.

BBC BASIC V, considérablement enrichi par rapport à ses prédécesseurs, tire parti de l'architecture 32 bits pour des performances inédites. Les développeurs découvrent un environnement de programmation réactif qui transforme l'apprentissage du code en plaisir.

L'Archimedes conquiert les écoles britanniques, perpétuant la tradition éducative du BBC Micro. Sa puissance de calcul attire aussi les scientifiques et les professionnels qui y trouvent une alternative crédible aux stations de travail UNIX. Acorn exploite d'ailleurs cette polyvalence en déclinant l'architecture ARM vers le monde professionnel avec les machines R140 et R260.

L'ARM, né dans les laboratoires d'Acorn pour équiper l'Archimedes, va conquérir le monde. La société ARM, issue de cette aventure, est depuis l'un des acteurs majeurs de l'industrie des semi-conducteurs. Aujourd'hui, les processeurs ARM équipent la quasi-totalité de nos smartphones et tablettes, validant rétrospectivement les choix audacieux des ingénieurs britanniques.

L'héritage de l'Archimedes dépasse ses ventes modestes. Cette machine démontre qu'une approche originale de la conception peut produire des résultats exceptionnels. Son architecture RISC influence durablement l'industrie, ses circuits spécialisés anticipent l'intégration moderne, et son système d'exploitation à l'origine les interfaces graphiques contemporaines.

L'Archimedes reste un modèle de cohérence technique. Chaque élément — processeur, mémoire, graphisme, son, système d'exploitation — participe à une vision d'ensemble harmonieuse. Cette approche globale, rare dans l'industrie informatique, produit une machine aux performances remarquables et à l'ergonomie soignée.

Apple Macintosh II

Mars 1987 : Apple bouleverse ses habitudes avec le Macintosh II. Trois ans après le premier Mac, cette machine brise les codes établis par Steve Jobs. Fini le boîtier compact et fermé, place à une architecture qui s'ouvre aux extensions. Cette volte-face répond à une nécessité : IBM gagne du terrain avec ses PC, et Apple doit séduire les professionnels.

Le cœur de la machine bat grâce au Motorola 68020, cadencé à 16 MHz. Apple l'associe au coprocesseur mathématique 68881, une combinaison qui quadruple les performances par rapport aux précédents Mac équipés du 68000. La mémoire vive de base atteint 8 mégaoctets, extensible jusqu'à 20 Mo avec un kit spécialisé. Pour le stockage, on trouve un lecteur de

disquettes de 800 Ko et, en option, un disque dur de 40 à 80 Mo.

Mais la véritable révolution du Macintosh II tient dans ses capacités graphiques. L'affichage en couleurs fait son apparition dans la gamme Mac avec une palette théorique de 16,7 millions de teintes. Seules 256 s'affichent simultanément, certes, mais c'est déjà un bond considérable. Plus audacieux, la machine accepte plusieurs moniteurs. Les utilisateurs choisissent entre l'écran monochrome 12 pouces et le modèle couleur 13 pouces, ce dernier exploitant la technologie Trinitron de Sony et sa réputation d'excellence.

L'aspect extérieur tranche avec la tradition Mac. Le boîtier horizontal évoque les PC IBM plutôt que l'élégance verticale des premiers Macintosh. Cette forme abrite une carte mère dotée de six slots d'extension NuBus. Ces emplacements accueillent cartes graphiques, contrôleurs SCSI ou émulateurs CP/M et Pascal. Apple maintient sa simplicité légendaire : aucun cavalier à positionner, aucun commutateur DIP à régler. Les cartes se configurent automatiquement.

La connectique ne lésine pas sur les possibilités. Deux ports série, deux connecteurs Apple Desktop Bus pour périphériques, une interface SCSI et une sortie audio stéréo équipent la machine. Le son mérite une attention particulière grâce à sa puce personnalisée. Quatre voix stéréo avec échantillonnage à 44,1 kHz : une qualité remarquable pour l'époque.

MultiFinder accompagne cette puissance matérielle. Ce système d'exploitation autorise l'exécution simultanée de plusieurs applications. Imprimer tout en téléchargeant, calculer en arrière-plan pendant la rédaction, toutes ces possibilités représentaient une innovation en 1987. L'interface graphique conserve la clarté Mac tout en exploitant la couleur.

Le prix reflète l'ambition du produit : 5 498 dollars avec disque dur, soit plus de 14 000 dollars actuels. Cette tarification vise clairement les professionnels et les utilisateurs fortunés. Les résultats suivent : les secteurs de la publication assistée par ordinateur, du graphisme et de l'ingénierie adoptent volontiers cette machine.

Au-delà des chiffres de vente, le Macintosh II instaure des standards durables. L'importance de la qualité d'affichage, la modularité maîtrisée, l'automatisation des configurations : ces principes guident encore la conception informatique contemporaine. La preuve qu'un ordinateur peut conjuguer puissance et simplicité d'usage, une philosophie qui traverse les décennies chez Apple.

La production s'achève en janvier 1990, remplacée par les Macintosh IIx et IIfx, plus performants. L'héritage perdure dans les stations de travail professionnelles qui suivront. Cette attention portée à l'affichage et au son annonce l'ère multimédia de l'informatique personnelle.

Le Macintosh II cristallise un moment particulier dans l'histoire d'Apple.

L'entreprise démontre sa capacité d'adaptation aux contraintes du marché professionnel sans abandonner ses convictions. Cette recherche d'équilibre entre ouverture commerciale et identité propre caractérise toujours la stratégie de développement d'Apple.

Commodore Amiga 500

Jay Miner n'avait qu'une idée en tête : construire l'ordinateur de ses rêves autour du processeur Motorola 68000. Cet ingénieur d'Atari se heurtait aux refus de sa hiérarchie, qui trouvait son projet trop onéreux. La rupture était inévitable. En 1982, Miner claque la porte et crée Hi-Toro, bientôt rebaptisée Amiga. Le pari ? Concevoir une machine aux performances graphiques et sonores jamais vues, dotée d'un vrai multitâche, sans pour autant ruiner l'acheteur.

L'équipe d'Amiga travaille dans un esprit bohème qui tranche avec la rigidité des grandes firmes. Les ingénieurs se réunissent armés de battes en mousse pour « voter » contre les mauvaises idées lors de leurs brainstormings déjantés. Cette atmosphère décontractée libère la créativité et pousse chacun à repenser l'informatique personnelle. Les prototypes s'empilent, faits de circuits imprimés gigantesques qui simulent les futurs composants sur mesure. L'innovation bouillonne, mais l'argent manque cruellement.

1984 sonne l'alerte : Amiga frôle la faillite. Atari propose un prêt de 500 000 dollars, mais le piège se cache dans les conditions. Si la somme n'est pas remboursée avant fin juin, Atari récupère toute la technologie. Un scénario cauchemardesque pour Miner et son équipe. Au dernier moment, Commodore sort de l'ombre avec une offre de rachat à 24 millions de dollars. Le sauvetage a un prix : l'indépendance d'Amiga s'évapore, mais la technologie survit.

Commodore accélère brutalement le développement. L'Amiga 1000 débarque en 1985, machine éblouissante techniquement mais commercialement ratée. Son prix décourage les familles, tandis que son positionnement embrouille les professionnels. Thomas Rattigan prend les commandes de Commodore en 1986 et tranche dans le vif : la gamme se divise en deux branches distinctes. D'un côté, l'Amiga 500 pour séduire le grand public. De l'autre, l'Amiga 2000 pour les utilisateurs exigeants.

L'Amiga 500 naît en 1987 de cette stratégie de différenciation. Les ingénieurs reprennent l'architecture du modèle 1000 en taillant dans les coûts sans pitié. Le clavier s'intègre directement dans le boîtier, l'alimentation

migre vers l'extérieur, la sortie TV directe cède la place à un adaptateur vendu séparément. Mais le cœur technologique demeure intact : trois processeurs spécialisés baptisés Agnes, Denise et Paula orchestrent un spectacle multimédia sans précédent.

Agnes gère la mémoire et les effets visuels, Denise s'occupe de l'affichage, Paula supervise le son et les entrées-sorties. Cette trinité électronique délivre 4 096 couleurs simultanées quand la concurrence peine à en afficher une poignée. Les sprites matériels animent les personnages de jeux avec une fluidité saisissante. Le son stéréo sur quatre canaux échantillonnés fait oublier les bips électroniques des autres machines. Le processeur Motorola 68000 cadencé à 7,14 MHz exécute un multitâche préemptif digne des stations de travail professionnelles.

AmigaOS repense entièrement l'interaction homme-machine. L'interface Workbench étonne par sa modernité, tandis que le noyau Exec jongle avec les tâches concurrentes sans broncher. RJ Mical signe avec l'API Intuition un petit chef-d'œuvre de programmation qui simplifie radicalement la création d'interfaces. Le système manipule plusieurs écrans de résolutions différentes, prouesse technique unique à l'époque.

Les développeurs de jeux se ruent sur cette puissance inédite. Avec ses douze plans de scrolling parallaxe qui glissent à des vitesses différentes, *Shadow of the Beast* hypnotise. *Defender of the Crown* éblouit par ses illustrations digitalisées d'un réalisme stupéfiant. Les musiciens inventent le format MOD, transformant l'Amiga en studio d'enregistrement domestique. *Deluxe Paint* ouvre aux artistes amateurs les portes de la création graphique numérique.

Une communauté passionnée se cristallise autour de la machine. Les magazines spécialisés comme *Amiga World* témoignent de cet engouement délirant. Les programmeurs échangent trucs et astuces pour extraire jusqu'à la dernière goutte de performance du matériel. Certains désactivent carrément le système d'exploitation pour un contrôle total, utilisant le processeur graphique comme débogueur en modifiant la couleur de fond à des points stratégiques du code.

L'Europe adopte massivement l'Amiga 500. Au Royaume-Uni, c'est la référence absolue du jeu vidéo familial. Son tarif abordable et ses capacités multimédia séduisent créateurs graphiques et musiciens en herbe. Andy Warhol s'empare de cette nouvelle toile électronique pour explorer les possibilités de l'art numérique.

Commodore gâche pourtant ce succès par une gestion désastreuse. L'entreprise accumule les erreurs stratégiques face à la montée des PC compatibles IBM. Le licenciement de Thomas Rattigan en 1987 prive la société de toute vision claire. L'Amiga 600 qui suit déçoit par son manque d'ambition,

simple toilettage cosmétique d'une architecture qui vieillit.

L'Amiga 500 reste dans l'histoire comme le symbole d'une innovation technique qui primait sur les considérations marketing. Multitâche préemptif, accélération graphique matérielle, son multicanal échantillonné : autant de concepts qui ne s'imposeront dans le grand public qu'une décennie plus tard. Son héritage perdure dans l'industrie vidéoludique et les outils de création multimédia.

La fidélité de sa communauté défie le temps. Des équipes comme les développeurs irakiens de *Babylonian Twins* ressortent leurs vieux projets Amiga des années plus tard pour les adapter aux smartphones. Cette obstination témoigne de l'empreinte laissée par une machine qui a repoussé les frontières de l'informatique personnelle, au point de rendre obsolètes certains ordinateurs professionnels.

Sharp X68000

En 1987, alors que le paysage informatique japonais grouillait de machines diverses – MSX, NEC PC-88, PC-98, ordinateurs Fujitsu –, Sharp prit une direction différente. L'entreprise décida de créer un ordinateur sans compromis, quitte à le vendre à prix d'or.

Le Sharp X68000 affichait ses ambitions dès le premier regard. Son boîtier en double tour, équipé d'une poignée rétractable, évoquait davantage un équipement Hi-Fi haut de gamme qu'un ordinateur traditionnel. Cette esthétique futuriste s'inspirait du Sharp X1 Twin, qui combinait déjà un ordinateur X1 et une console PC Engine. Le design industriel franchissait ici un cap : l'ordinateur devenait un objet de désir.

Sous ce capot distinctif se cachait une mécanique de précision. Le processeur Motorola 68000 tournait à 10 MHz, soutenu par 1 Mo de mémoire vive extensible jusqu'à 12 Mo et 1 Mo de mémoire vidéo dédiée. Ces spécifications plaçaient la machine dans une catégorie à part. Le X68000 affichait 65 536 couleurs simultanées et gérait jusqu'à 128 sprites matériels – des chiffres qui faisaient rêver les développeurs de jeux vidéo.

Le prix reflétait ces performances : 369 000 yens, soit environ 6 000 dollars actuels. Sharp ne visait pas le grand public mais une clientèle d'amateurs éclairés et de professionnels. Cette stratégie se matérialisa par le développement d'outils et de bibliothèques spécifiquement destinés aux créateurs de jeux. Le pari porta ses fruits : le X68000 accueillit des versions souvent supérieures aux originaux arcade de *Final Fight*, *Ghouls'n Ghosts*

ou *Street Fighter II*.

Le système d'exploitation Human68k, conçu par Hudson Soft, empruntait à MS-DOS tout en apportant ses propres innovations. Les premières machines utilisaient l'interface VS, remplacée plus tard par SX-Window sur les modèles ultérieurs. Une troisième interface, Ko-Window, rappelait Motif et fut développée par des tierces parties.

L'histoire du X68000 s'écrivit au fil des années à travers une succession de modèles. En 1988 apparut l'ACE, suivi l'année suivante par l'EXPERT qui intégrait d'office 2 Mo de RAM. Le modèle PRO adopta un boîtier PC classique pour accueillir davantage de slots d'extension. 1990 marqua l'arrivée des versions SUPER, qui troquèrent l'interface SASI contre le SCSI. Le XVI de 1991 poussa le processeur à 16 MHz, tandis que le Compact XVI de 1992 proposait un format plus compact avec des lecteurs 3,5 pouces. La série culmina en 1993 avec le X68030, propulsé par un processeur 68030 à 25 MHz.

La richesse de l'écosystème X68000 impressionne encore. Des cartes d'extension MIDI permettaient de connecter des synthétiseurs comme le Roland MT-32. La carte graphique TS-6BGA combinait accélération et son PCM. Certains accessoires atteignent désormais des sommets en termes de rareté, comme la carte POLYPHON qui intégrait FPU, MIDI, PCM et 8 Mo de RAM dans un seul module. Côté contrôle, les manettes incluaient le Cyber Stick pour les simulations et l'étonnant XE-1 au design non conventionnel.

La créativité de la communauté homebrew produisit des œuvres remarquables. *Cho Ren Sha 68k*, développé par deux personnes en 1995, reste un exemple saisissant. Ce jeu de tir affichait 512 sprites simultanés grâce à des techniques avancées de multiplexage – un exploit technique d'autant plus remarquable qu'il fut programmé principalement en C plutôt qu'en assembleur. Cette réalisation témoigne de la qualité des outils de développement mis à disposition.

Pourtant, l'aventure X68000 s'acheva en 1993. Sharp n'avait pas fait évoluer significativement le matériel vidéo et audio pendant six ans. Face à la montée en puissance des PC équipés de cartes graphiques et sonores performantes, la machine perdait de son attrait malgré ses qualités intrinsèques. Un projet de version PowerPC fut aussi abandonné. Mais un geste inattendu changea la donne. En 2000, Sharp, Hudson et d'autres entreprises impliquées libérèrent le système d'exploitation Human68k, le BIOS et divers outils en domaine public. Cette décision exceptionnelle rend ce système toujours accessible grâce à une émulation légale et fidèle de la machine.

Le X68000 incarne une vision japonaise particulière de l'informatique personnelle des années 1980. Là où d'autres privilégiaient le compromis

entre prix et performances, Sharp choisit l'excellence technique et le design audacieux. Sa capacité à reproduire l'expérience arcade, ses performances graphiques exceptionnelles et son architecture évoluée en font une machine à part. Mais les problèmes de fiabilité de son alimentation et son prix prohibitif limitèrent sa diffusion.

OS/2

En 1987, IBM et Microsoft scellent une alliance improbable pour créer OS/2, le système d'exploitation qui devait équiper les nouveaux ordinateurs IBM PS/2. Cette collaboration entre deux mastodontes partait d'un constat simple : le DOS montrait ses limites face aux processeurs Intel 80286. Il fallait faire mieux, faire plus moderne. L'idée séduisante sur le papier cachait déjà des tensions latentes entre deux visions diamétralement opposées de l'avenir informatique.

OS/2 apportait ce qui manquait aux utilisateurs de l'époque : une interface graphique digne de ce nom et des fonctionnalités jusque-là réservées aux gros systèmes. Il partageait certains traits avec Windows, UNIX ou Xenix, mais ses ambitions dépassaient celles de ses concurrents. Cette première mouture ne laissait rien présager du divorce retentissant qui allait secouer l'industrie.

Car les deux partenaires regardaient dans des directions opposées. Microsoft lorgnait vers Windows, qu'elle considérait comme l'avenir. IBM voulait poursuivre sur OS/2. Cette opposition grandissante a fini par exploser en 1992, provoquant ce que les observateurs ont appelé un « divorce ». Microsoft est repartie avec ses billes et une partie du code, qu'elle a recyclé dans Windows NT et Windows 95. IBM s'est retrouvée seule avec son système sur les bras.

Pour rebondir, IBM s'est tournée vers Commodore Business Machines et a puisé son inspiration dans AmigaOS. La version 2.0 d'OS/2, lancée en mars 1992, a introduit le Workplace Shell, une transformation dans l'interface utilisateur. Cette innovation marquait l'entrée de la programmation orientée objet dans l'univers grand public des systèmes d'exploitation. Une première qui ne passait pas inaperçue.

Le Workplace Shell bousculait les habitudes. Glisser-déposer des icônes dans des dossiers, accéder aux menus par clic droit, déclencher une impression en déplaçant simplement une icône : autant de gestes qui changeaient alors l'interaction homme-machine. Le hic venait des limitations

matérielles, car les ordinateurs manquaient de mémoire vive, contraignant les utilisateurs à des mises à niveau coûteuses pour exploiter pleinement ces nouveautés.

1994 voit naître OS/2 Warp, la version qui marquera l'apogée du système. Cette mouture séduisait par sa simplicité d'usage et son architecture 32 bits, tout en gardant la compatibilité DOS. Un atout technique non négligeable face aux Windows de l'époque, englués dans leur base DOS 16 bits. Pourtant, malgré ces qualités indéniables, OS/2 n'arrivait pas à percer sur le marché grand public.

Le système a trouvé refuge dans des créneaux spécialisés, particulièrement les guichets automatiques bancaires où sa stabilité et sa fiabilité faisaient merveille. Microsoft, de son côté, déployait une stratégie commerciale agressive pour imposer Windows, reléguant OS/2 dans ces niches techniques. Une situation frustrante pour IBM, qui voyait son système cantonné à des usages spécialisés.

IBM finit par jeter l'éponge en 2005, arrêtant la production d'OS/2, suivi de son support en 2006. Les inconditionnels du système ont réclamé sa libération sous licence libre, mais cette démarche s'est heurtée aux droits de propriété intellectuelle de Microsoft sur certaines parties du code.

L'histoire d'OS/2 ne s'arrête pas là. En 2001, Serenity Systems obtient une licence d'IBM et commercialise le système sous le nom d'eComStation. Plus tard, en 2015, Arca Noae récupère le flambeau avec ArcaOS.

OS/2 a introduit des innovations qui font partie de notre quotidien numérique. Sa conception rigoureuse et sa recherche de stabilité ont établi de nouvelles références. Pourtant, son échec commercial rappelle que les qualités techniques ne suffisent pas : stratégies commerciales, alliances industrielles et capacité à fédérer un écosystème déterminent largement le succès d'une technologie.

Unicode

En 1968, ASCII est une norme aux États-Unis. Le principe est d'attribuer une valeur numérique entre 0 et 127 à chaque caractère. La lettre « a » prend le code 97, « Z » le code 90. Simple, efficace... mais terriblement limité. Impossible d'écrire correctement « naïve » ou « café » avec ce système pensé pour l'anglais.

Durant les années 1980, l'informatique personnelle sur machines 8 bits prend son envol. Les valeurs possibles s'étendent jusqu'à 255. Constructeurs

et éditeurs se ruent sur ces 128 positions supplémentaires pour y loger les caractères accentués. Chacun y va de sa solution. Commodore place les lettres accentuées françaises à sa façon, IBM aussi avec son code page 437. Un fichier créé sur une machine est illisible sur une autre. Ce qui ressemblait à du français sur un Amstrad CPC se transforme en hiéroglyphes sur un PC compatible.

Des conventions émergent petit à petit. L'Organisation internationale de normalisation standardise certains codages, d'autres s'imposent par la force de l'usage commercial. Mais le problème de fond demeure : 255 caractères, c'est dérisoire face aux besoins réels. Impossible d'intégrer simultanément les accents français et l'alphabet cyrillique. Les utilisateurs jonglent entre codages selon la langue du moment. KOI8 pour taper en russe, Latin1 pour le français. Écrire un texte citant du Tolstoï en version originale relève de l'exploit technique.

À la fin des années 1980, quelques visionnaires se lancent dans un projet fou : créer un système capable de représenter tous les caractères de toutes les langues humaines. Unicode naît de cette ambition, avec comme idée de départ passer aux caractères 16 bits. Au lieu de 255 positions, on dispose de 65 536 emplacements. De quoi loger chinois, arabe, hébreu, cyrillique et alphabets latins dans le même système.

L'ISO travaille en parallèle sur sa norme 10646. Deux projets, un seul objectif. Heureusement, les équipes finissent par fusionner leurs efforts avec la version 1.1 d'Unicode. Les 65 536 positions se révèlent insuffisantes et la spécification moderne s'étend jusqu'à 1 114 111 caractères possibles (0x10FFFF en notation hexadécimale).

Unicode va bien au-delà du simple catalogue de caractères. La norme définit des propriétés et règles d'usage complexes. Le consortium responsable publie un manuel de plus de 1 000 pages. S'y ajoutent 14 annexes techniques, 7 standards, 6 rapports techniques et 4 rapports stabilisés. Césure, rendu bidirectionnel, disposition verticale : tout y passe. L'ampleur de cette documentation révèle la complexité réelle du traitement de texte multilingue.

Mais en pratique, comment stocker ces caractères en mémoire ? La solution brutale consiste à réserver 32 bits par caractère. Mais elle multiplie par quatre l'espace disque et la bande passante par rapport à ASCII. Sans compter les problèmes de portabilité entre processeurs organisant différemment les octets.

UTF-8 apporte la réponse élégante. Ce format utilise un nombre variable d'octets selon le caractère. Un seul octet pour ASCII (valeurs inférieures à 128), deux octets pour les valeurs jusqu'à 2 047, trois ou quatre octets au-delà. Compatibilité ASCII préservée, pas d'octets nuls gênants, possibilité

de resynchronisation en cas de corruption : les avantages s'accumulent.

UTF-16 existe aussi mais reste moins populaire. Il encode les caractères sur deux ou quatre octets avec un mécanisme spécial appelé « surrogate pairs » pour les caractères hors du plan multilingue de base. Un marqueur d'ordre des octets indique l'organisation des données en mémoire.

Les mises à jour de la norme créent parfois des turbulences. L'amendement 5 à ISO 10646 déplace et étend le bloc coréen Hangul, invalidant les données existantes. Cet épisode, baptisé « le bazar coréen », pousse les comités de normalisation à s'engager contre ce type de modification incompatible.

Unicode gère des mécanismes sophistiqués. Les caractères se combinent, comme un accent venant modifier sa lettre de base. La norme définit des règles de rendu bidirectionnel pour mélanger textes arabes (droite vers gauche) et occidentaux (gauche vers droite) dans le même document.

L'adoption d'Unicode transforme l'informatique. Systèmes d'exploitation, protocoles Internet, formats de fichiers : tout bascule vers cette représentation unifiée. L'internationalisation des logiciels s'en trouve grandement facilitée. Échanger un document multilingue est enfin naturel.

Le consortium Unicode poursuit son travail d'enrichissement. Nouveaux caractères pour couvrir des écritures historiques, support de besoins émergents : la norme évolue constamment. L'arrivée des émojis illustre cette capacité d'adaptation aux nouveaux usages de la communication numérique. Chaque ajout nécessite un soin minutieux pour préserver la cohérence d'ensemble.

Unicode marque une rupture dans l'histoire de l'informatique textuelle. Cette norme établit une base technique solide pour le traitement multilingue. Son adoption universelle prouve qu'il est possible de créer des standards internationaux complexes servant efficacement tous les utilisateurs de la planète.

FrameMaker

En 1986, un étudiant en astrophysique de l'université Columbia abandonnait ses études pour se lancer dans l'aventure de l'édition technique. Charles Corfield venait de développer un prototype baptisé /etc/publisher et avait la conviction qu'il pouvait faire mieux que ce qui existait alors. Insatisfait des outils disponibles et désireux de s'assurer un avenir financier

stable, il quitta les bancs de l'université pour se consacrer entièrement à la programmation.

Cette décision marqua la naissance de Frame Technology, une société fondée par quatre personnalités aux profils complémentaires. Aux côtés de Corfield et de ses compétences en mathématiques et développement, Steven Kirsch apportait son expérience entrepreneuriale forgée chez Mouse Systems. David Murray maîtrisait les subtilités des interfaces utilisateur et de l'édition, tandis que Vickie Blakeslee excellait dans l'organisation et la gestion opérationnelle. Cette association d'expertises diverses créa un terreau fertile pour l'innovation.

Le marché de l'édition technique était alors dominé par Interleaf, qui imposait sa loi sur les stations de travail UNIX. Son système, vendu autour de 30 000 dollars, exigeait une installation complexe et imposait ses propres standards d'interface. Face à ce monopole, Frame Technology choisit une approche radicalement différente. FrameMaker fut proposé à 2 500 dollars avec une interface native aux systèmes d'exploitation et une installation d'une simplicité déconcertante.

Cette stratégie de rupture s'appuyait sur des innovations techniques remarquables. FrameMaker réussissait à marier les fonctionnalités des traitements de texte et des logiciels de publication assistée par ordinateur. Le programme gérait la numérotation automatique des chapitres et sections, les références croisées en temps réel, proposait une gestion sophistiquée des tableaux et un support multilingue avancé. Sa capacité à gérer des documents volumineux répartis sur plusieurs fichiers séduisait les rédacteurs techniques. L'architecture basée sur le concept de « frames » (cadres) offrait une flexibilité inédite dans l'intégration du texte et des graphiques.

Le succès ne se fit pas attendre. Dès la version bêta 0.6, commercialisée en 1986, l'entreprise vendit quelques centaines de licences. Le premier client, un groupe d'ingénierie de John Deere, ouvrit la voie à une adoption rapide. Les administrations fédérales américaines furent séduites par cette approche non liée à du matériel spécifique. Les partenariats avec les fabricants de stations de travail UNIX comme Sun Microsystems accélérèrent cette expansion.

Initialement développé pour UNIX, FrameMaker fut adapté pour le Macintosh puis Windows. Cette portabilité reposait sur une architecture logicielle innovante, le Device Independent Maker, qui séparait le code central des interfaces spécifiques à chaque système. Le format d'échange MIF (Maker Interchange Format) garantissait l'interopérabilité entre plateformes, une préoccupation croissante dans un monde informatique de plus en plus hétérogène.

Frame Technology innova aussi sur le plan commercial. Le système de

licences partagées autorisait plusieurs utilisateurs à se partager un nombre limité d'accès simultanés, réduisant considérablement le coût pour les organisations. L'entreprise développa des partenariats stratégiques, notamment avec Toshiba, qui généra plus de 5 millions de dollars de redevances. Cette approche flexible du modèle économique contribua largement à l'adoption du logiciel.

La croissance de la société fut spectaculaire. De 5 employés en 1986, Frame Technology passa à 297 personnes en 1991. Le chiffre d'affaires suivit une progression similaire : 3,4 millions de dollars en 1987, 41,7 millions en 1991. L'introduction en bourse de février 1992 valorisa l'entreprise à 146 millions de dollars. Adobe Systems racheta Frame Technology en 1995 pour environ 500 millions de dollars, consacrant ainsi le succès de cette aventure entrepreneuriale.

La culture d'entreprise se distinguait par son ouverture et sa diversité. Les équipes incluaient une proportion remarquable de femmes, de minorités et de personnes LGBT, y compris aux postes de direction. Cette diversité nourrissait la créativité et stimulait l'innovation. La gestion des projets s'inspirait de l'approche décrite par Richard Feynman dans *Surely You're Joking, Mr. Feynman !*, privilégiant la discussion ouverte suivie de décisions claires et assumées.

L'impact de FrameMaker sur l'industrie dépassa largement les espérances de ses créateurs. Le logiciel démocratisa l'accès à l'édition technique professionnelle, auparavant réservée aux grandes organisations disposant de budgets conséquents. Il établit de nouvelles normes en matière d'interface utilisateur et d'intégration aux systèmes d'exploitation. Son architecture modulaire et sa gestion avancée des documents structurés influencèrent durablement le développement des futurs outils d'édition.

Trois décennies plus tard, FrameMaker demeure un outil de référence dans l'édition technique, utilisé par plus de 700 000 personnes dans 30 000 entreprises. Adobe continue à faire évoluer le logiciel, l'adaptant aux besoins contemporains avec le support de formats comme XML et DITA, ainsi que la publication multicanal vers HTML5 et les formats numériques.

IBM AS/400

Alors que l'informatique d'entreprise se fragmentait entre une multitude de systèmes incompatibles, IBM préparait en secret une machine qui allait bousculer les codes établis. L'AS/400 naîtrait en 1988 dans les laboratoires

de Rochester, Minnesota, loin des grands centres de recherche de la côte Est américaine. Cette gestation particulière explique en partie pourquoi cette architecture développa des caractéristiques si singulières.

L'équipe de Frank Soltis travaillait depuis 1978 sur le System/38, un ordinateur aux concepts précurseurs mais au succès commercial mitigé. Les ingénieurs de Rochester avaient choisi une voie radicalement différente de celle explorée par les créateurs d'UNIX, de VMS ou des futurs concepteurs de Windows NT. Leur approche resta d'ailleurs si confidentielle qu'IBM ne partageait pas tous les secrets de cette architecture avec ses autres divisions.

Quand l'AS/400 fut dévoilé le 21 juin 1988, l'annonce surprit par son ampleur. Six modèles de processeurs étaient proposés simultanément, accompagnés de plus de 1 000 applications prêtes à l'emploi. Cette disponibilité logicielle immédiate constituait un record dans l'industrie informatique. Les performances promises impressionnaient : la mémoire était multipliée par 24, le stockage par 48, et la puissance de calcul décuplée par rapport aux générations antérieures.

La véritable innovation résidait dans l'architecture interne. L'AS/400 reposait sur une interface machine indépendante de la technologie, baptisée TIMI. Les programmes ne communiquaient plus directement avec le processeur mais avec une machine virtuelle. Cette abstraction permit en 1995 un tour de force technique : IBM remplaça les processeurs CISC 48 bits par des puces RISC PowerPC 64 bits sans qu'aucune application existante ne nécessite la moindre modification. Tous les programmes continuèrent de fonctionner comme si de rien n'était.

Le système organisait tout sous forme d'objets. Fichiers, programmes, imprimantes, utilisateurs : chaque élément possédait une description précise de ses usages autorisés et de sa fonction. Cette conception orientée objet, inhabituelle pour l'époque, renforçait la sécurité et simplifiait la gestion. Un administrateur pouvait manipuler n'importe quel composant du système avec les mêmes commandes standardisées.

L'intégration constituait l'autre pilier de cette architecture. Là où des systèmes assemblaient péniblement base de données, sécurité, communications et outils de sauvegarde provenant de fournisseurs différents, l'AS/400 incorporait nativement toutes ces fonctions dans son système d'exploitation. DB2/400 gérait les données, la sécurité contrôlait chaque accès, les protocoles de communication étaient intégrés. Cette unité simpliait drastiquement l'administration quotidienne.

La gestion mémoire révolutionnait les pratiques habituelles. Le « single-level store » créait un espace d'adressage unique de 64 bits, soit environ 18 quintillions d'octets. Les programmes et leurs données recevaient des adresses permanentes dans cet espace gigantesque. Plus besoin de jongler

entre mémoire principale et stockage secondaire : le système se chargeait automatiquement de placer les objets au bon endroit selon leur utilisation.

Les années 1990 virent l'AS/400 évoluer pour s'adapter aux nouveaux besoins. En 1994, l'Advanced Series intégrait Lotus Notes et l'accès Internet. Des modèles portables apparurent, ainsi que des versions destinées aux petites structures. La fiabilité s'améliora d'un facteur 20 entre 1988 et 1992, tandis que les performances progressaient de 30% chaque année.

Le succès commercial accompagna l'excellence technique. 250 000 systèmes des gammes précédentes (System/34, /36 et /38) étaient déjà installés lors du lancement. En 1992, IBM livrait le 200 000e AS/400 à la brasserie Heineken. Ces chiffres témoignaient d'une adoption massive dans les entreprises du monde entier.

L'identité de la machine évolua au fil des rachats et des stratégies marketing d'IBM. AS/400e, puis eServer iSeries, eServer i5, System i5, System i, et finalement IBM i en 2008. Ce dernier nom soulignait les nouvelles capacités du système, désormais capable d'exécuter simultanément IBM i, AIX et Linux sur le même matériel POWER.

L'environnement de développement se modernisa progressivement. L'Integrated Language Environment remplaça l'ancien modèle de programmation en améliorant les performances des programmes modulaires. RPG et COBOL, langages historiques de la plateforme, cohabitèrent avec C/400, Pascal et Java. L'interface en ligne de commande, reconnaissable à ses mnémoniques comme WRKOBJ (« work with object »), survécut à tous les changements tout en s'enrichissant d'interfaces graphiques modernes.

Cette longévité exceptionnelle révèle la pertinence d'une vision différente de l'informatique d'entreprise. Pendant que la plupart des constructeurs développaient des systèmes modulaires assemblant des composants hétérogènes, IBM Rochester privilégiait l'intégration, la cohérence et la simplicité d'usage. L'AS/400 prouve qu'une approche alternative aux standards dominants peut survivre et prospérer pendant plus de trois décennies.

Internet Relay Chat

En 1988, dans les locaux de l'université d'Oulu, un étudiant finlandais du nom de Jarkko Oikarinen travaille sur un problème qui l'agace : comment faire discuter plusieurs personnes simultanément sur les BBS, ces tableaux d'affichage électroniques qui précèdent le Web ? Les outils existants comme « talk » ou « rmsg » sous UNIX se limitent à des conversations à deux.

Oikarinen imagine alors quelque chose de différent, des salons de discussion où n'importe qui pourrait entrer et participer à la conversation générale.

Cette idée donne naissance à l'Internet Relay Chat, plus connu sous l'acronyme IRC. Pour la première fois, des dizaines de personnes peuvent échanger en temps réel dans un espace virtuel. Oikarinen baptise ces espaces des « canaux », identifiés par le symbole dièse qui deviendra plus tard célèbre sur d'autres plateformes.

Les premiers mois d'IRC restent confidentiels. Le réseau se cantonne à la Scandinavie, bridé par les connexions internationales balbutiantes de l'époque. La situation change quand Mike Jacobs, depuis le MIT, parvient à se connecter au système d'Oikarinen. Ce contact transatlantique ouvre les vannes : le code circule, de nouveaux serveurs s'ajoutent, le réseau grandit.

En janvier 1991, lors de la guerre du Golfe, IRC connaît sa première utilisation massive lors d'un événement mondial. Des utilisateurs se rassemblent pour relayer en temps réel des informations de guerre, atteignant pour la première fois plus de 300 utilisateurs simultanés. Les logs de ces échanges sont conservés dans les archives ibiblio.

En août 1991, lors de la tentative de coup d'État contre Mikhaïl Gorbatchev, IRC joue son rôle. Pendant le blocus médiatique imposé par les putschistes (19-21 août), des utilisateurs à Moscou utilisent IRC pour transmettre des informations en temps réel vers le reste du monde, contournant ainsi la censure. Les logs de ces échanges sont aussi conservés dans les archives. Des sources mentionnent parfois l'utilisation d'IRC lors de la crise constitutionnelle russe de septembre-octobre 1993, mais cette information reste à confirmer par des sources primaires.

Le succès attire les convoitises et les désaccords. Des querelles éclatent sur la gestion du réseau unique d'IRC. Qui peut créer de nouveaux serveurs ? Selon quels critères ? Les tensions montent jusqu'à la rupture. Une partie des utilisateurs fait sécession pour créer Undernet, laissant le serveur original d'Oikarinen former la base d'EFnet. Ces deux réseaux rivaux continuent d'exister, rejoints par des centaines d'autres nés des mêmes divisions.

Car IRC se fragmente. Chaque réseau développe ses propres règles, ses propres serveurs, son propre code. Les programmeurs expérimentent, modifient, adaptent. De cette effervescence émergent trois grandes familles techniques : le protocole P10 d'Undernet, hybrid pour EFnet, bahamut pour Dalnet. La diversité technique reflète la philosophie décentralisée d'IRC, où personne ne contrôle l'ensemble.

Cette décentralisation a un prix. Les réseaux IRC souffrent régulièrement de *netsplits*, ces coupures temporaires qui divisent un réseau en plusieurs morceaux. Quand deux serveurs perdent contact, leurs utilisateurs se retrouvent séparés jusqu'à ce que la connexion se rétablisse. Ces incidents,

d'abord perçus comme des pannes gênantes, sont peu à peu une caractéristique acceptée du système.

L'architecture d'IRC reste d'une simplicité déconcertante. Les messages transitent en texte brut, sans chiffrement ni stockage persistant. Cette légèreté technique explique en partie la robustesse du protocole, qui fonctionne aussi sur des connexions poussives, avec des serveurs aux ressources limitées. Là où d'autres systèmes exigent des serveurs puissants et des connexions rapides, IRC se contente de peu.

Les utilisateurs compensent les limitations du protocole par leur créativité. Ils développent des robots de discussion, des systèmes d'enregistrement de conversations, des interfaces graphiques sophistiquées. L'écosystème logiciel autour d'IRC foisonne. Chacun programme ses propres outils, partage ses scripts, améliore ceux des autres.

Cette culture du bricolage marque IRC. Le protocole influence d'ailleurs les plateformes qui lui succéderont. Le @ des opérateurs de canaux, le # des noms de salons, ces conventions nées dans les années 1980 réapparaîtront bien plus tard sur Twitter et d'autres réseaux sociaux. IRC invente un langage symbolique que le Web grand public finira par adopter.

Conçu pour quelques centaines d'utilisateurs maximum, IRC dépasse ses limites théoriques. Au plus fort de son succès, les quatre plus gros réseaux rassemblent plus de 500 000 personnes connectées simultanément. Les serveurs tiennent bon, la technique s'adapte, les communautés prospèrent.

Bien plus qu'un simple outil de discussion, des communautés entières se développent avec IRC, avec leurs codes, leurs habitudes, leurs rituels. Les développeurs de logiciels libres y trouvent un lieu d'échange technique. Les passionnés d'informatique s'y retrouvent pour débattre des dernières innovations. Des amitiés naissent, des projets se montent, des collaborations s'organisent.

IRC survit discrètement dans l'ombre des géants du Web social. Les jeunes générations l'ignorent souvent, préférant des interfaces plus colorées et des fonctionnalités plus riches. Mais les communautés techniques lui restent fidèles. Sur des milliers de canaux, des conversations continuent, des projets s'élaborent, des savoirs se transmettent. IRC incarne une certaine idée d'Internet : décentralisé, ouvert, maîtrisable par ses utilisateurs.

SNMP

À la fin des années 1980, les réseaux informatiques connaissaient une croissance sans précédent. Cette expansion révélait un problème épineux : chaque constructeur développait ses propres outils de gestion, créant un véritable casse-tête pour les administrateurs. IBM avait ses solutions, Cisco les siennes, et aucune ne parlait le même langage. Jongler entre ces multiples interfaces relevait de l'acrobatie, d'autant que certains outils se limitaient à de simples commandes cryptiques.

L'Internet Architecture Board comprit l'ampleur du défi. En 1988, l'organisme publia la recommandation RFC 1052, qui esquissait les contours d'un standard unifié pour la gestion des réseaux Internet. La mission fut confiée à l'IETF avec un délai serré : 90 jours pour concevoir ce qui allait devenir SNMP. Les contraintes étaient claires : simplicité d'implémentation et inspiration du protocole CMIP de l'ISO.

Trois documents RFC virent le jour simultanément cette année-là. Le premier introduisait la Structure of Management Information, un langage abstrait fondé sur ASN.1 pour décrire formellement les données de gestion. Le deuxième définissait la Management Information Base, cette base de données arborescente où résideraient tous les objets surveillés. Le troisième détaillait les mécanismes opérationnels du protocole proprement dit.

L'architecture reposait sur une logique manager-agent d'une simplicité désarmante. D'un côté, des agents dispersés sur chaque équipement collectaient inlassablement les informations locales. De l'autre, un manager central interrogeait ces agents à la demande ou recevait leurs alertes spontanées, appelées traps. Cette asymétrie fonctionnait remarquablement bien dans les premières installations.

Le succès rapide de SNMPv1 révéla toutefois ses faiblesses congénitales. La sécurité n'était qu'une façade : les community strings voyageaient en clair sur le réseau, accessibles à quiconque interceptait les paquets. Cette vulnérabilité n'était pas un oubli mais un choix délibéré. Les concepteurs avaient parié sur la bénignité des informations échangées et tablé sur des réseaux fermés où la confiance régnait.

Cette naïveté poussa l'IETF à développer SNMPsec au début des années 1990. Cette extension promettait authentification, chiffrement et contrôle d'accès sophistiqué. Mais sa complexité effraya les développeurs, et elle fut éclipsée par SNMPv2 en 1993.

La seconde version du protocole corrigeait des irritants de la première mouture. La MIB s'enrichissait de capacités de regroupement, rendant la description d'équipements complets plus naturelle. L'opération GetBulk permettait de récupérer massivement des données, économisant de précieux

allers-retours réseau. Les notifications gagnaient en fiabilité grâce au mécanisme Inform, qui exigeait un accusé de réception.

Malheureusement, les querelles intestines au sein de la communauté de développement firent voler en éclats l'unité du standard. SNMPv2 se fragmenta en variantes incompatibles : SNMPv2, SNMPv2c, SNMPv2u, SNMPv2*. Cette cacophonie technique découragea les adoptions et maintint SNMPv1 dans une position dominante paradoxale.

L'IETF tira les leçons de ces échecs pour concevoir SNMPv3, achevé en 2002. Cette troisième version préservait les acquis de SNMPv2 tout en proposant une architecture sécuritaire modulaire. Le User-based Security Model autorisait différents algorithmes de chiffrement comme DES ou AES. Le View-based Access Control Model offrait une granularité remarquable dans la définition des permissions d'accès.

Parallèlement, des extensions spécialisées enrichissaient l'écosystème. En 1991, RMON débarqua avec des fonctions de surveillance réseau avancées adaptées aux réseaux Ethernet. RMON2 étendit six ans plus tard cette surveillance aux couches application. SMON compléta en 1999 l'arsenal avec des outils dédiés aux commutateurs et réseaux virtuels.

Pourtant, contre toute logique, SNMPv1 conserve aujourd'hui la majorité des déploiements. Sa rusticité explique cette longévité surprenante. Les versions plus récentes, techniquement supérieures, n'ont jamais réussi à convaincre massivement les praticiens, attachés à la simplicité éprouvée de la version originale.

Le protocole SNMP a validé l'approche pragmatique face aux solutions sophistiquées mais lourdes. Il a établi des principes architecturaux durables comme la séparation entre format de données et contenu, et le modèle manager-agent devenu référence universelle.

SNMP témoigne aussi de la capacité d'extension des standards Internet. Initialement limité aux équipements réseau, il supervise serveurs, onduleurs, systèmes de climatisation et une multitude d'autres dispositifs. Cette polyvalence a consolidé sa position dans l'écosystème informatique.

Les limites du protocole affleurent néanmoins avec l'évolution des infrastructures. SNMP collecte des données élémentaires mais délègue leur traitement intelligent au manager. Cette philosophie montre ses faiblesses face aux réseaux contemporains, générateurs de flots informationnels torrentiels. Des technologies comme NETCONF émergent pour répondre à ces nouvelles contraintes, sans remettre en cause l'omniprésence de SNMP dans l'existant.

ZIP

En 1989, Phil Katz et Gary Conway imaginaient-ils que leur format de compression révolutionnerait durablement l'informatique ? L'histoire du ZIP commence pourtant dans la controverse. PKWARE, leur entreprise, venait d'être attaquée en justice par Systems Enhancement Associates qui l'accusait d'avoir copié le système d'archivage ARC. Plutôt que de se défendre, Katz et Conway décidèrent de créer quelque chose de radicalement nouveau.

Le nom ZIP traduit cette ambition. Robert Mahoney, ami de Katz, suggéra ce terme pour évoquer la vitesse de leur nouvelle solution, bien supérieure aux formats concurrents. Le 14 février 1989, PKWARE et Infinity Design Concepts annoncent la mise dans le domaine public de leur format. Cette décision stratégique, véritable coup de génie, transforme instantanément ZIP en standard ouvert. La spécification technique, documentée dans le fichier `APPNOTE.TXT`, est accessible à tous. Les développeurs s'emparent de cette opportunité, multipliant outils et bibliothèques compatibles.

Les premières versions du format reflètent les contraintes matérielles de l'époque. PKZIP 2.04g ne peut contenir que 16 383 fichiers par archive, avec une taille maximale de 2 Go. Ces limitations correspondaient alors aux capacités des machines disponibles. L'évolution rapide du matériel informatique pousse néanmoins les concepteurs à repenser constamment leur format.

En 1993 apparaît la version 2.0 qui introduit la compression DEFLATE et le chiffrement traditionnel PKWARE. Cette mouture pose les bases de l'architecture moderne des fichiers ZIP. Trois ans plus tard, Deflate64 améliore les performances de compression. Mais c'est en 2001 que survient une transformation majeure avec la version 4.5 et l'introduction du format ZIP64. Fini la limite des 4 Go, terminée la restriction à 65 535 fichiers par archive.

L'année 2002 marque l'ouverture vers de nouvelles méthodes de chiffrement : DES, Triple DES, RC2, RC4 rejoignent l'arsenal disponible. En 2003, l'AES fait son apparition. La version 6.3.0 de 2006 franchit encore un cap en intégrant l'Unicode pour les noms de fichiers et de nouveaux algorithmes comme LZMA et PPMd. Chaque évolution témoigne d'une capacité d'adaptation remarquable aux besoins changeants des utilisateurs.

La structure interne du format ZIP révèle une conception astucieuse. Le répertoire central, placé en fin de fichier, catalogue tous les éléments de l'archive. Cette organisation autorise l'ajout ou la suppression de fichiers sans retraiter l'ensemble de l'archive. À l'époque des disquettes, où chaque opération d'écriture prenait un temps considérable, cette caractéristique

représentait un avantage décisif.

Le succès du format pousse les éditeurs de systèmes d'exploitation à l'adopter nativement. Microsoft intègre le support ZIP dans Windows dès 1998 via le pack Windows Plus ! pour Windows 98, baptisant cette fonctionnalité « dossiers compressés ». Apple suit en 2003 avec Mac OS X 10.3. Les systèmes libres n'ont pas tardé à emboîter le pas.

Depuis 2006, Microsoft utilise le format compressé ZIP comme base de son Office Open XML. Les fichiers .docx, .xlsx et .pptx ne sont rien d'autre que des archives ZIP contenant des fichiers XML et diverses ressources. Cette utilisation détournée illustre la polyvalence d'un format devenu infrastructure invisible de l'informatique moderne.

La reconnaissance officielle arrive en 2015 avec la publication de la norme ISO/IEC 21320-1. Cette standardisation définit un format d'archive compressé minimal, compatible avec différents standards comme OpenDocument, Office Open XML et EPUB. La norme impose toutefois des restrictions : seule la compression DEFLATE est autorisée, le chiffrement est interdit.

Les questions de sécurité ont longtemps taraude le format ZIP. Le système de chiffrement original, ZipCrypto, s'est révélé vulnérable aux attaques par texte en clair connu. Ces faiblesses ont motivé l'introduction de méthodes plus robustes. Une controverse éclate en 2003 quand WinZip intègre son propre système de chiffrement AES tandis que PKWARE maintient sa spécification concurrente. L'accord trouvé en 2004 entre les deux entreprises facilite le support mutuel de leurs formats respectifs.

L'héritage du ZIP se lit dans l'adoption de l'algorithme DEFLATE par d'autres technologies comme gzip et zlib. Le format continue sa mue : la version 6.3.8 de 2020 intègre des méthodes de compression modernes comme Zstandard, MP3 et XZ. Cette capacité d'évolution, rendue possible par une conception modulaire acceptant de nouveaux champs, assure la pérennité du format.

Trente-cinq ans après sa création, ZIP reste omniprésent. Les implémentations contemporaines supportent ZIP64 et ses archives théoriques de 16 exaoctets. Cette longévité exceptionnelle s'explique par une combinaison rare entre simplicité d'usage, compatibilité ascendante et adaptabilité technique. Phil Katz, décédé en 2000, n'aura pas vu son format traverser le nouveau millénaire et s'imposer comme l'un des piliers silencieux de l'informatique moderne.

Intel 80486

L'année 1989 voit naître le processeur Intel 80486, une puce qui marque une rupture dans l'univers des microprocesseurs. Pour la première fois dans l'histoire de la famille x86, un million de transistors se retrouvent gravés sur un seul composant. Cette prouesse technique s'accompagne d'innovations qui transforment radicalement les performances des ordinateurs personnels.

La véritable rupture du 80486 tient dans son architecture repensée. Intel intègre directement sur la puce une unité de calcul en virgule flottante, jusqu'alors cantonnée à un coprocesseur séparé, le 80387. Cette fusion simplifie la conception des cartes mères et accélère considérablement les calculs mathématiques complexes. À cette innovation s'ajoute une mémoire cache unifiée de 8 Ko, positionnée au cœur du processeur. Fini les accès lents au cache externe qui pénalisaient le 80386.

L'introduction d'un pipeline d'instructions ajoute au processeur la possibilité de traiter simultanément des étapes d'instructions différentes. Les opérations arithmétiques simples s'exécutent désormais en un seul cycle d'horloge, divisant par deux le temps nécessaire par rapport au 80386. Cette efficacité nouvelle propulse les performances bien au-delà des gains apportés par la simple augmentation de fréquence.

Les premières versions du 80486 cadencent entre 16 et 33 MHz. En 1991, Intel tente de pousser la fréquence jusqu'à 50 MHz, mais se heurte à des problèmes de dissipation thermique. La solution passe par une finesse de gravure réduite à 0,8 micromètre. Malgré ces ajustements, ce modèle rapide peine à séduire, notamment à cause de sa compatibilité limitée avec les bus locaux indispensables aux cartes graphiques de l'époque.

La gamme 80486 se diversifie. Le 486SX, version allégée dépourvue d'unité de calcul flottant, cible le marché grand public soucieux d'économies. Plus astucieux, le 486DX2 inaugure une technique qui fera école : le doublement de la fréquence interne par rapport au bus système. Cela augmente les performances sans imposer de refonte complète des cartes mères existantes. Le DX4, malgré son nom trompeur, ne quadruple pas mais triple la fréquence du bus.

Un élément juridique vient perturber la stratégie d'Intel. L'entreprise n'a plus le droit de déposer de marque sur des dénominations purement numériques commençant par « 80 ». Cette contrainte pousse Intel à repenser sa communication produit et annonce l'ère des noms de marque comme Pentium.

Le succès commercial du 80486 attire la concurrence. AMD, IBM, Texas Instruments, Cyrix, UMC et STMicroelectronics développent leurs propres versions. AMD se distingue en proposant des fréquences absentes du cata-

logue Intel, comme le bus à 40 MHz. La firme commercialise des modèles originaux : 486DX-40, 486DX/2-80 et 486DX/4-120. En 1995, l'Am5x86 d'AMD est le 486 le plus rapide jamais conçu, cadencé à 133 MHz, tandis que des versions expérimentales atteignent 150 et 160 MHz.

Cyrix adopte une stratégie différente en développant ses puces par rétro-ingénierie complète, sans s'appuyer sur les plans d'Intel. Les premiers modèles, 486DLC et 486SLC, constituent des solutions hybrides compatibles avec les supports 386. Handicapés par leur cache de seulement 1 Ko, ils peinent face aux 8 Ko des modèles Intel et AMD. Les versions ultérieures de Cyrix, dotées d'un cache plus généreux, arrivent hélas trop tard pour bouleverser les parts de marché.

L'âge d'or des jeux MS-DOS coïncide avec l'apogée du 80486. Le modèle DX2-66 MHz est la référence des passionnés de jeux vidéo au début des années 1990. Cette hégémonie vacille avec l'émergence de la 3D temps réel. Ces nouveaux graphismes sollicitent intensément l'unité flottante et exigent une bande passante mémoire considérable. Les développeurs commencent à optimiser leurs créations pour l'architecture P5 du Pentium, condamnant progressivement les processeurs 486.

Dans l'univers des ordinateurs personnels, le 80486 survit jusqu'aux années 2000 dans des configurations économiques. L'abandon du support de Windows 95 et les exigences croissantes des systèmes d'exploitation suivants accélèrent son obsolescence. Paradoxalement, Intel maintient la production jusqu'en septembre 2007 pour alimenter le marché des systèmes embarqués.

Cette puce incarne une période où les performances informatiques progressent à un rythme effréné, permettant de nouvelles applications et accélérant la démocratisation de l'outil informatique.

Intel i960

Intel frappe un grand coup en septembre 1989 avec le lancement du i960CA, son premier processeur superscalaire 32 bits dédié aux systèmes embarqués. Cette puce marque une rupture dans l'univers du calcul embarqué en exécutant des instructions simultanément à chaque cycle d'horloge. Le résultat ne se fait pas attendre avec 66 MIPS, un chiffre qui fait sensation à l'époque.

La philosophie derrière le i960 surprend par son audace. Intel marie le cœur RISC traditionnel avec ses 32 registres à un jeu d'instructions enrichi qui emprunte au monde CISC. Ce mélange des genres, loin d'être un

compromis bancal, répond avec précision aux contraintes particulières des applications embarquées. Les ingénieurs d'Intel ont su doser cette hybridation pour créer un processeur qui tire le meilleur des deux mondes.

La famille s'étoffe, le i960CF succède au CA et double ses performances grâce à une mémoire cache repensée. Intel décline ensuite sa création selon une logique commerciale bien rodée : les SA/SB pour les budgets serrés, les KA/KB pour le milieu de gamme, et les versions MC durcies pour les applications militaires. Cette stratégie de gamme témoigne d'une approche industrielle mûre.

Sous le capot, les innovations foisonnent. Le contrôleur d'interruptions ne se contente pas de gérer les priorités, il le fait de manière autonome. Les registres locaux se sauvegardent sans intervention lors des appels de sous-programmes, éliminant une source traditionnelle de latence. Les caches d'instructions équipent tous les modèles, tandis que certaines versions bénéficient de caches de données qui accélèrent les traitements répétitifs.

Les transferts de données atteignent 160 Mo par seconde sur les bus haute performance, une vitesse remarquable. Cette capacité transforme la gestion des flux d'informations dans les applications gourmandes. La compatibilité du code objet entre les différents modèles constitue un atout majeur : les développeurs peuvent migrer leurs applications sans repartir de zéro, un gage de pérennité apprécié.

L'intégration pousse l'innovation plus loin. Le modèle VH rassemble sur une seule puce une interface PCI v2.1 32 bits à 33 MHz, un contrôleur mémoire et diverses fonctionnalités périphériques. Cette concentration réduit les coûts, l'encombrement et la consommation, trois paramètres critiques dans l'embarqué.

Les domaines d'application du i960 dessinent une cartographie impressionnante. Les réseaux l'adoptent pour leurs contrôleurs de communication, leurs ponts et leurs routeurs. L'imagerie médicale, particulièrement l'échographie, exploite ses capacités de traitement. L'automatisation industrielle, la robotique et la vision artificielle trouvent dans ce processeur un allié de choix. Le secteur aérospatial l'intègre dans ses équipements de contrôle de vol et ses systèmes de navigation par satellite.

Intel ne se limite pas au silicium. Le programme Solutions960 fédère plus de 200 outils développés par 70 entreprises partenaires. Compilateurs optimisés, systèmes d'exploitation, outils de débogage, cartes d'évaluation : l'écosystème prend forme autour du processeur. Cette richesse logicielle transforme l'adoption du i960 en expérience fluide pour les développeurs.

Le contrôleur de DMA intégré impressionne par ses performances : 59 Mo/s en transferts « fly-by », 32 Mo/s en transferts deux cycles. Le contrôleur d'interruptions gère jusqu'à 248 sources externes avec 32 niveaux de

priorité programmables. La mémoire cache verrouillable garantit l'exécution optimale des algorithmes critiques, une fonctionnalité précieuse dans les systèmes temps réel.

La spécialisation guide le développement des différentes versions. Les modèles réseau traitent les paquets sans saturer, tandis que ceux dédiés à l'imagerie s'adaptent aux différentes largeurs de bus et conventions de données. Cette approche ciblée renforce l'attractivité des processeurs selon leur domaine d'application.

L'exécution superscalaire s'est démocratisée, l'intégration de périphériques sur la puce principale est devenue la norme, l'importance d'un écosystème de développement complet reste d'actualité. La fin des années 1990 voit naître de nouvelles architectures concurrentes qui bousculent le paysage. Le i960 aura néanmoins tracé la voie vers des solutions toujours plus intégrées et performantes.

World Wide Web

En cette fin d'année 1989, Tim Berners-Lee travaille au CERN à Genève. L'informaticien britannique se heurte quotidiennement au même problème : comment retrouver une information dans cette tour de Babel qu'est devenue l'organisation européenne de recherche nucléaire ? Les chercheurs accumulent des documents sur leurs expériences, leurs équipements, leurs découvertes, mais tout cela reste dispersé, inaccessible. Berners-Lee imagine alors un système pour relier ces informations éparses.

Depuis des décennies, des visionnaires rêvent de machines à organiser le savoir humain. Paul Otlet avait esquissé son Mundaneum dès 1934, une bibliothèque universelle connectée. Vannevar Bush théorise en 1945 le Memex dans « As We May Think », une machine capable de naviguer entre des documents liés par associations d'idées. Ted Nelson baptise « hypertexte » cette navigation non-linéaire en 1965 avec son projet Xanadu, qui ne verra jamais le jour malgré des décennies de développement. Apple concrétise une partie de ces rêves en 1987 avec HyperCard, premier succès commercial de l'hypertexte.

Berners-Lee tranche dans le vif. Plutôt que de concevoir un système complexe, il mise sur la simplicité. En 1990, avec l'aide de Robert Cailliau, il jette les bases de son architecture : l'URL identifie chaque ressource de manière unique, le protocole HTTP organise les échanges entre machines, le langage HTML structure les documents et leurs liens. Dans la foulée, il

développe le premier navigateur Web qu'il appelle WorldWideWeb, avant de le renommer Nexus.

Le premier site Web s'allume le 6 août 1991 à l'adresse info.cern.ch. Tim Berners-Lee y explique ce qu'est le World Wide Web, comment créer des pages, comment installer un navigateur. L'information se propage sur les forums Usenet. L'année suivante, les Cernettes, groupe de musique formé par des employées du CERN, voient leur photo devenir la première image non scientifique publiée sur le Web. Un clin d'œil qui annonce la démocratisation à venir.

Le tournant survient le 30 avril 1993. Ce jour-là, le CERN renonce à ses droits d'auteur sur les technologies du Web et les place dans le domaine public. Cette décision, en apparence anodine rétrospectivement, change tout. N'importe qui peut désormais utiliser, modifier, distribuer le code sans contrainte. Les universités s'emparent de la technologie, les entreprises suivront.

Marc Andreessen saisit l'opportunité. En 1994, son navigateur Mosaic révolutionne l'expérience utilisateur avec son interface graphique intuitive. Exit les lignes de commande et les interfaces austères : le Web devient accessible au grand public. Cette année naît aussi le World Wide Web Consortium sous la direction de Berners-Lee. Cette organisation fixe les standards techniques du Web.

1995 marque l'entrée fracassante du commerce sur la toile. Yahoo ! lance son annuaire, classant méticuleusement les sites par catégories. Jeff Bezos ouvre sa librairie Amazon depuis son garage de Seattle. Microsoft intègre Internet Explorer dans Windows 95, déclenchant la première guerre des navigateurs contre Netscape. Le Web n'est plus seulement un outil de recherche, c'est aussi un marché.

Trois ans plus tard, deux étudiants de Stanford bouleversent la donne. Larry Page et Sergey Brin créent Google avec leur algorithme PageRank, qui classe les pages selon leur popularité mesurée par les liens entrants. Cette approche mathématique de la pertinence transforme radicalement la recherche d'information. Le Web est le point d'entrée universel vers le savoir numérique.

L'évolution se poursuit par vagues successives. Le Web des premiers temps, celui de 1989 à 2005, ressemble à une immense bibliothèque. Les sites restent statiques, les utilisateurs se contentent de lire. HTML, HTTP et les protocoles de base suffisent à faire fonctionner cet univers encore simple.

Entre 2004 et 2016, tout bascule vers l'interaction avec ce qu'on appellera plus tard le Web 2.0. Les utilisateurs ne se contentent plus de consulter ; ils créent, partagent, commentent. Facebook ouvre ses portes en 2004, You-

Tube suit en 2005. JavaScript, XML et Ajax enrichissent les interfaces, les rendent dynamiques et réactives.

À partir de 2015 émerge une troisième génération, le Web sémantique. L'ambition change d'échelle : il s'agit de rendre l'information compréhensible par les machines elles-mêmes. RDF, RDFS et OWL structurent les données pour que les ordinateurs puissent en saisir le sens, pas seulement l'afficher.

Le protocole HTTP accompagne ces mutations. Sa version 1.1, standardisée en 1997, introduit les connexions persistantes qui évitent de renégocier chaque échange. HTTP/2 arrive en 2015 avec le multiplexage des flux, qui autorise plusieurs requêtes simultanées sur une seule connexion. HTTP/3, finalisé en 2022, abandonne TCP au profit de QUIC basé sur UDP, gagnant en vitesse.

L'apparition de l'AppStore d'Apple en 2008 redéfinit l'accès au Web. Les applications mobiles complètent le navigateur traditionnel, parfois le remplacent. Le Web sort des ordinateurs pour s'installer dans nos poches, nos voitures, nos montres. Cette ubiquité était impensable en 1989.

L'architecture imaginée par Berners-Lee a résisté à toutes ces transformations. La séparation entre contenu et présentation, les identifiants uniques pour chaque ressource, la standardisation des protocoles : ces choix techniques ont permis une extension continue sans rupture majeure. Le modèle client-serveur et l'architecture REST offrent un cadre assez souple pour accueillir tous les usages.

Bien sûr, le Web actuel ressemble peu au système documentaire pensé au CERN. Il a modifié nos façons de communiquer, de nous informer, de commercer, de travailler. Si certaines dérives inquiètent (concentration des services entre quelques acteurs, exploitation des données personnelles, désinformation) la vision de Berners-Lee d'un espace d'information universel et décentralisé s'est largement réalisée.

Microsoft Office

Le 1er août 1988, dans l'effervescence de Las Vegas, Bill Gates prend la parole pour annoncer une suite logicielle destinée au travail de bureau. À cette époque, personne ne mesure vraiment l'ampleur de ce qui se joue. Pourtant, cette annonce lance une aventure qui bouleversera les habitudes de travail partout dans le monde.

Deux ans plus tard, en 1990, la première version de Microsoft Office

voit le jour. Elle rassemble trois programmes que l'on peut qualifier désormais de basiques : Word pour écrire, Excel pour calculer, PowerPoint pour présenter. Cette combinaison n'a rien d'anodin. Microsoft a flairé le besoin d'avoir sous la main tous les outils nécessaires au quotidien professionnel, sans jongler entre différents éditeurs et leurs logiques parfois contradictoires.

L'histoire de Word commence en réalité bien avant, dès 1981, quand Microsoft embauche Charles Simonyi pour développer un traitement de texte. La première version sort en 1983, mais elle déroute. WordPerfect règne en maître et les utilisateurs ne comprennent pas cette interface si différente. Microsoft ne baisse pas les bras. Une version Macintosh arrive en 1985, puis en 1987 le format RTF fait son apparition, simplifiant enfin l'échange de documents entre systèmes différents.

Excel, de son côté, s'impose comme la référence des tableurs. Le logiciel sait gérer des calculs complexes, dessiner des graphiques et analyser des données grâce aux tableaux croisés dynamiques. Avec l'arrivée de Visual Basic pour Applications, les utilisateurs peuvent programmer des automatisations et se libérer des tâches répétitives.

PowerPoint a une genèse particulière. Initialement baptisé Presenter par la société Forethought, Microsoft le rachète en 1987 pour 14 millions de dollars. Robert Gaskins suggère alors le nouveau nom. Ce logiciel transforme radicalement la façon de concevoir et de présenter. Dans sa première mouture de 1990, impossible de revenir en arrière dans les diapositives, et les options de personnalisation restent limitées.

Microsoft sort Windows 95 et Office 95 simultanément... en 1995. Cette synchronisation n'est pas fortuite : l'intégration poussée entre le système d'exploitation et la suite bureautique crée un environnement de travail uniforme qui séduit massivement les entreprises. Les ventes explosent et Microsoft établit son hégémonie.

La suite s'enrichit progressivement. Access arrive pour gérer les bases de données, Publisher pour créer des supports marketing, OneNote pour prendre des notes numériquement. Outlook complète l'ensemble en gérant e-mails et calendriers. Chaque ajout renforce l'attractivité de l'écosystème.

Microsoft doit s'adapter aux évolutions technologiques. Des versions voient le jour pour Windows et macOS. L'entreprise développe des applications mobiles comme Office Lens et Office Remote pour suivre les nouveaux usages. SharePoint et Skype for Business Server étendent les capacités collaboratives vers les serveurs d'entreprise.

L'avènement du cloud computing redistribue les cartes. Office Web Apps propose une version allégée accessible par navigateur. Office 365 révolutionne ensuite le modèle économique en passant à l'abonnement. Cette mutation répond aux attentes de mobilité et de collaboration temps réel des

utilisateurs contemporains.

La cohérence de la suite se renforce au fil des versions. Le correcteur orthographique se partage, les données circulent entre applications, l'interface s'harmonise. Apprendre un logiciel de la suite facilite la maîtrise des autres. Le format .docx, introduit avec Office 2007, améliore compatibilité et sécurité des documents.

L'éducation est un terrain de conquête stratégique. Des générations d'élèves et d'étudiants apprennent sur Office, créant une base d'utilisateurs familiers de ces outils. La disponibilité en 35 langues accélère l'adoption mondiale. Elle déroute initialement les habitués, mais l'interface ribbon de 2007 modernise l'accès aux fonctionnalités.

Office façonne les pratiques professionnelles contemporaines. Word impose ses standards de mise en forme documentaire. Excel révolutionne l'analyse de données dans les entreprises et la gestion financière. PowerPoint redéfinit les codes de la communication d'entreprise, et ces pratiques perdurent dans l'univers numérique.

Cette histoire reflète les mutations du travail de bureau depuis les années 1990. D'une collection de trois logiciels, Office se mue en écosystème complet mêlant services cloud, applications mobiles et outils collaboratifs. Cette capacité d'adaptation continue explique sa longévité dans un secteur pourtant mouvant.

NeXTSTEP

Quand Steve Jobs quitte Apple en 1985, il emporte avec lui une idée fixe : créer l'ordinateur parfait. Cette obsession donnera naissance à l'une des aventures les plus fascinantes de l'histoire informatique, celle de NeXTSTEP. Voilà un système d'exploitation qui va bouleverser les codes établis.

Dans l'informatique du milieu des années 1980, les interfaces graphiques se cherchent, les systèmes plantent au moindre faux pas, la programmation relève du parcours du combattant. Jobs et son équipe de NeXT Computer décident de repartir de zéro. Leur pari était de marier la robustesse d'UNIX à une interface nouvelle. Ils choisissent BSD comme système de base, y greffent le noyau Mach, et voilà leur machine de guerre prête à défier l'industrie.

Mais la vraie audace de NeXTSTEP tient dans son architecture entièrement orientée objet. Là où ses concurrents bricolent avec du code procédural, NeXT mise tout sur Objective-C. Cette décision paraît folle à l'époque

pour qui veut comprendre cette approche. Les développeurs vont-ils suivre ? L'équipe persiste et transfore chaque composant du système en objet : les fenêtres, les boutons, les fichiers, les processus. Cette cohérence architecturale transforme l'expérience utilisateur.

L'interface de NeXTSTEP frappe par son élégance. Display PostScript assure un rendu graphique d'une précision chirurgicale. Ce que vous voyez à l'écran correspond exactement à ce que l'imprimante produira. Le fameux Dock fait sa première apparition, cette barre d'icônes qui simplifie l'accès aux applications. Innovation plus discrète mais tout aussi remarquable : le Workspace Manager propose une navigation en colonnes qui révolutionne la gestion des fichiers.

NeXTMail pousse plus loin l'intégration multimédia. Envoyer une image dans un message électronique ? Un fichier son ? Un document formaté ? Rien de plus simple. Cette vision moderne de la messagerie devance son temps de plusieurs années. Les autres systèmes peinent à afficher correctement du texte simple quand NeXT propose déjà l'équivalent des e-mails enrichis qu'on connaît.

Interface Builder révolutionne le développement logiciel. Fini le codage laborieux des interfaces : les programmeurs assemblent désormais leurs fenêtres visuellement, par glisser-déposer. Cette approche inspire encore les outils actuels. Les Services introduisent une interopérabilité inédite entre applications. Retoucher une image depuis un traitement de texte devient un jeu d'enfant.

Le système de fichiers de NeXTSTEP anticipe nos besoins contemporains. Les métadonnées extensibles permettent d'enrichir chaque fichier d'informations contextuelles. Le Digital Librarian indexe l'ensemble des documents et propose des recherches sophistiquées dans toute la machine. Google Desktop et Spotlight reprendront cette idée vingt ans plus tard.

Tim Berners-Lee choisit une station NeXT pour inventer le Web au CERN. Ce choix n'est pas anodin puisque les outils de développement de NeXTSTEP permettent de prototyper des applications complexes. Le premier navigateur web et le premier serveur web naissent sur cette plateforme. Wall Street s'intéresse aussi au système. Les banques d'investissement découvrent sa stabilité et adoptent les stations NeXT pour leurs applications critiques.

Mais l'excellence technique ne suffit pas. Le prix prohibitif des machines NeXT freine leur adoption. Seuls les professionnels fortunés et quelques universités peuvent se les procurer. Le grand public reste inaccessible. En 1994, NeXT abandonne le matériel et se recentre sur le logiciel. OPENSTEP, la version portable de NeXTSTEP, fonctionne sur différentes architectures. Cette stratégie élargit l'audience mais arrive peut-être trop tard.

Le rachat par Apple en 1996 relance tout. NeXTSTEP renaît sous les traits de Mac OS X. L'ADN du système de Jobs façonne le nouveau système d'Apple : architecture orientée objet, frameworks Cocoa, philosophie de l'interface utilisateur. iOS hérite directement de cette lignée. L'iPhone et l'iPad doivent beaucoup aux innovations de NeXTSTEP.

Cette histoire prouve qu'en informatique, l'avance technologique finit toujours par payer. Les meilleures idées survivent aux échecs commerciaux et trouvent leur chemin vers le succès.

VIII

1990

L'avènement d'un monde connecté

La destruction du mur de Berlin en novembre 1989 a bouleversé l'équilibre mondial. Cette fracture dans l'histoire a créé un terrain fertile pour les technologies numériques qui ne demandaient qu'à se propager. L'échiquier géopolitique redessiné a changé la donne pour l'informatique et sa diffusion.

Quand les républiques soviétiques ont ouvert leurs portes aux technologies occidentales, elles ont déverrouillé des marchés insoupçonnés pour les firmes informatiques. Les mathématiciens de l'Est, réputés pour leur excellence, ont partagé leurs approches avec leurs collègues occidentaux. Ce métissage intellectuel a enrichi la pensée informatique des deux côtés.

Les années 1990 ont vu l'économie mondiale s'affranchir des barrières traditionnelles. Les grandes entreprises ont dû s'adapter à cette nouvelle réalité : gérer des opérations planétaires exigeait des systèmes d'information capables de traiter des montagnes de données et de synchroniser des activités à travers les fuseaux horaires.

La finance a subi une métamorphose radicale. Les transactions se sont dématérialisées, les algorithmes ont investi les parquets. Les bourses ont multiplié la cadence et le volume des échanges, ce qui réclamait des machines toujours plus véloces et des réseaux toujours plus robustes.

Internet a quitté le cercle fermé des universitaires et des militaires pour envahir le quotidien des particuliers. Les fournisseurs d'accès ont fleuri, d'abord avec de modestes connexions par modems qui sifflaient et grésillaient, ensuite avec des débits de plus en plus généreux. Cette ouverture a fait naître de nouveaux services et modes de communication.

Les citoyens ont créé leurs espaces d'expression numérique. Les forums, les messageries électroniques et les sites personnels ont tissé des liens invisibles entre les individus. Des communautés virtuelles se sont formées, réunissant des passionnés par-delà les distances. Cette nouvelle socialisation a transformé les interfaces utilisateur, qui devaient maintenant séduire un public non initié aux arcanes informatiques.

L'ordinateur s'est imposé sur chaque bureau professionnel. Les entreprises ont massivement investi dans le matériel et la formation. Le traitement de texte a enterré la machine à écrire. Les tableurs ont conquis la gestion. Les diaporamas ont chamboulé les réunions en remplaçant les films transparents et leurs rétroprojecteurs. Cette informatisation à marche forcée a créé un appétit vorace pour des logiciels adaptés aux réalités professionnelles.

Dans les écoles et universités, l'informatique a trouvé sa place dans les programmes. Les salles d'ordinateurs sont devenues des lieux familiers pour les étudiants. Cette évolution répondait aux exigences d'un marché du travail en quête de compétences techniques.

Le jeu vidéo s'est mué en industrie majeure durant cette période. Les consoles ont repoussé les limites techniques, avec des graphismes toujours plus saisissants. Ce secteur a dynamisé la recherche sur les processeurs graphiques et a fait entrer l'informatique dans les foyers par la porte du divertissement.

L'œuvre numérique s'est enrichie avec l'avènement du multimédia. Le CD-ROM a permis de marier texte, images, sons et vidéos dans un format accessible. Les encyclopédies numériques ont ranimé le savoir. Les applications ludiques ont fait souffler un vent nouveau sur l'informatique personnelle.

Les composants électroniques ont continué à rétrécir, donnant naissance aux premiers ordinateurs portables grand public et aux assistants numériques personnels. La mobilité devenait une exigence, forçant l'adaptation des interfaces et des systèmes d'exploitation.

La téléphonie mobile a connu un essor fulgurant. Les réseaux numériques se sont déployés à travers villes et campagnes. Les SMS ont inventé un nouveau langage, concis et immédiat. On devinait la fusion future entre téléphone et ordinateur.

Les administrations ont modernisé leurs infrastructures informatiques,

et numérisé leurs archives poussiéreuses et développé les premiers services en ligne pour les citoyens. Ces projets d'envergure ont stimulé le marché des solutions professionnelles.

Avec la multiplication des connexions, la sécurité est devenue une préoccupation centrale. Les premiers virus ciblant l'informatique personnelle ont semé la panique, suivis par le développement des antivirus et des pare-feu. La protection des données personnelles a émergé comme un enjeu du XXI^{e} siècle naissant.

Le logiciel libre a trouvé sa voie avec GNU et Linux. Cette philosophie alternative au logiciel propriétaire a bousculé les modèles établis en proposant une approche collaborative et ouverte du développement informatique.

Les dernières années de la décennie ont vu fleurir les premières boutiques en ligne et gonfler la bulle Internet. Les investisseurs se sont rués sur tout ce qui touchait au numérique, parfois avec une fièvre plus passionnelle que rationnelle. L'éclatement de cette bulle attendait au tournant du millénaire.

Cette décennie a changé notre rapport à l'informatique. L'ordinateur s'installait dans notre quotidien, transformait nos métiers, modifiait nos loisirs. Il n'était plus cet être étrange confiné dans des laboratoires climatisés. Le taux d'équipement des ménages a grimpé en flèche, atteignant 23% en France en 1999, contre à peine 8,5% en 1990.

Le monde se découvrait connecté, sans bien mesurer encore les conséquences de cette mutation.

L'informatique des années 1990 ne se résume pas à une simple évolution technique, elle constitue une rupture anthropologique dont nous commençons à peine à percevoir l'ampleur.

Adobe Photoshop

En 1987, Thomas Knoll prépare sa thèse de vision par ordinateur à l'Université du Michigan. Sur son Macintosh Plus, lancé en janvier 1986, il bricole un petit programme qu'il baptise Display, pour afficher des images en niveaux de gris sur son écran bitmap noir et blanc. Une distraction académique, rien de plus. Thomas ne se doute pas qu'il vient de planter la première graine de ce qui deviendra l'outil de référence mondial pour traiter les images.

John Knoll, le frère de Thomas, découvre ce programme. John travaille chez Industrial Light and Magic, la division effets spéciaux de Lucasfilm. Chez ILM, on cherche des solutions pour traiter les images numériquement.

John voit immédiatement l'intérêt du petit logiciel de son frère et lui propose de s'associer. Cette collaboration fraternelle va changer la donne : Thomas adapte son code pour gérer la couleur sur le nouveau Macintosh II, John développe des routines de traitement qui font penser aux futurs filtres du logiciel.

John pressent le potentiel commercial de leur création en 1988. Thomas hésite, conscient du travail titanesque que représente la transformation d'un programme personnel en véritable application commerciale. L'optimisme de John l'emporte. Les deux frères se mettent au travail, enrichissent les fonctionnalités, ajoutent la prise en charge de formats de fichiers variés, développent des outils de sélection sophistiqués. Après de nombreuses tentatives de baptême, le programme hérite de son nom définitif : Photoshop.

Trouver un éditeur s'avère plus compliqué que prévu. Les frères Knoll frappent à plusieurs portes. Barneyscan, fabricant de scanners, accepte de commercialiser une première version sous l'appellation Barneyscan XP. Environ 200 exemplaires de cette version 0.87 accompagnent leurs appareils. Le logiciel circule dans le petit monde des professionnels de l'image. Chez Apple, les ingénieurs s'enthousiasment et partagent des copies avec leurs collègues.

En septembre 1988, Adobe Systems entre dans la partie. Russell Brown, directeur artistique de l'entreprise, tombe sous le charme du logiciel. Les négociations aboutissent à un accord de licence de distribution plutôt qu'à un rachat pur et simple. Décision judicieuse pour les Knoll, qui toucheront des royalties sur chaque vente. La signature formelle intervient en avril 1989.

Février 1990, Photoshop 1.0 débarque sur le marché. Malgré quelques bugs de jeunesse, l'accueil est enthousiaste. Face à la concurrence, le logiciel tire son épingle du jeu grâce à son code optimisé et son interface intuitive. Russell Brown multiplie les démonstrations spectaculaires, révélant les possibilités créatives du programme et sa facilité d'usage.

La version 2.0 de 1991 marque une étape décisive. Le support CMYK bouleverse l'industrie de l'impression en démocratisant les techniques de séparation des couleurs, jusque-là réservées aux professionnels équipés de matériels coûteux. La version 2.5 franchit un nouveau cap en arrivant sur Windows, multipliant l'audience potentielle.

1994 voit naître Photoshop 3.0, et avec lui, les calques. Cette fonctionnalité transforme radicalement la manipulation d'images numériques. Les utilisateurs peuvent désormais superposer et modifier des éléments indépendamment. Contrairement aux rumeurs de l'époque, cette innovation ne copie pas le concurrent Live Picture mais découle des recherches antérieures de Thomas Knoll.

La version 4.0 de 1996 bouscule les habitudes. Andrei Herasimchuk repense l'interface utilisateur pour l'harmoniser avec les autres produits Adobe. Ces changements provoquent d'abord des remous dans la communauté des utilisateurs, puis finissent par convaincre par leur cohérence.

Avec la version 5.0 de 1998, Mark Hamburg révolutionne le processus créatif en introduisant la palette Historique. Cette innovation rompt avec la logique linéaire traditionnelle des annulations, autorisant un retour en arrière non séquentiel sur les modifications. Simultanément, l'intégration directe des outils de gestion des couleurs simplifie la production d'images destinées à différents supports.

Le succès de Photoshop dépasse toutes les prévisions. Le logiciel s'impose dans la photographie, la publicité, l'édition, la création web. Adobe voit son chiffre d'affaires bondir de 16 millions de dollars en 1986 à plus d'un milliard en 1999. Un écosystème florissant se développe autour du programme : plugins, formations, publications spécialisées.

En démocratisant la manipulation d'images, Photoshop nourrit les arts visuels contemporains et transforme notre rapport à la photographie. Le verbe « photoshoper » intègre le vocabulaire courant, preuve de l'omniprésence culturelle du logiciel.

Cette réussite exceptionnelle résulte du génie technique de Thomas Knoll, l'intuition commerciale de John, l'efficacité marketing d'Adobe, le tout porté par l'essor de l'informatique personnelle et du design numérique. Thomas n'achèvera jamais sa thèse, mais sa « distraction » révolutionne l'univers de l'image numérique. Il continue de collaborer avec Adobe comme consultant, tandis que John poursuit sa carrière dans les effets spéciaux chez Industrial Light & Magic.

IBM PS/1

L'histoire du PS/1 commence en 1987, quand IBM observe avec inquiétude la montée des clones PC. Ces machines compatibles grignotent ses parts de marché, vendues bien moins cher que les ordinateurs de Big Blue. Les particuliers et les petites entreprises se détournent d'IBM, plus sensibles au prix qu'au prestige de la marque.

IBM n'ignorait pas ce segment. Trois ans plus tôt, l'entreprise avait tenté le coup avec le PCjr, un ordinateur censé séduire les foyers américains. Cette machine proposait des innovations intéressantes : des cartouches pour les jeux, une sortie télévision et un clavier sans fil baptisé « Freeboard ». Mais

le PCjr s'écrasait lamentablement en 1985, laissant IBM les mains vides sur le marché grand public.

Entre-temps, la société mise tout sur sa gamme PS/2, destinée aux entreprises. Ces ordinateurs rivalisent en performances avec la concurrence, offrent de nombreuses possibilités d'extension, mais leur tarif les rend inaccessibles aux particuliers. Avec plus de vingt modèles au catalogue, l'offre est illisible pour qui cherche simplement un ordinateur familial.

En 1990, IBM lance le Personal System/1. Cette fois, l'approche change du tout au tout. Les premiers modèles adoptent un design tout-en-un audacieux : l'écran couleur et l'unité centrale forment un ensemble compact, reliés par d'épais câbles. L'alimentation électrique loge dans le moniteur, créant une dépendance totale entre les deux éléments. Cette configuration évoque les Macintosh compacts d'Apple, à ceci près qu'Apple intégrait tout dans un seul boîtier.

Les versions suivantes abandonneront ce parti pris esthétique pour revenir à des éléments séparés, annonçant la future ligne Aptiva. Le PS/1 fonctionne sous PC-DOS, la version IBM du système MS-DOS. Pour simplifier la vie des utilisateurs novices, IBM développe une interface graphique surnommée « 4-quad ». Windows ou OS/2 restent techniquement installables, mais IBM déconseille leur usage à cause des problèmes de performances et de stabilité que cela engendre.

Une particularité technique distingue ces premières machines : contrairement aux installations habituelles sur disque dur ou disquettes, le système d'exploitation et l'interface résident directement dans la mémoire morte. Tandy avait expérimenté cette approche sur certains modèles de sa gamme 1000. L'avantage se révèle immédiat : démarrage rapide et fiabilité accrue.

L'évolution technique s'accélère au fil des années. Les modèles de lancement embarquent un processeur Intel 80286 à 10 MHz, performance modeste mais suffisante pour les applications bureautiques de l'époque. Quatre ans plus tard, quand IBM cessera la production, les derniers PS/1 intègrent un Intel 486 DX2 cadencé à 66 MHz, multipliant les capacités de calcul.

Cette disposition particulière de l'alimentation dans l'écran ne constitue pas une première. L'Amstrad PC1512 avait adopté cette solution, tout comme le Coleco Adam qui logeait son bloc d'alimentation dans l'imprimante. Mais cette configuration impose l'usage d'un connecteur propriétaire entre moniteur et unité centrale, limitant les possibilités de remplacement ou d'évolution.

IBM ne se contente pas de livrer une machine. L'entreprise déploie une stratégie marketing ambitieuse pour reconquérir le grand public. Un service d'assistance téléphonique fonctionne sept jours sur sept, week-ends inclus. Les propriétaires de PS/1 peuvent échanger entre eux via un service en

ligne dédié, le « Users' Club », préfigurant les communautés d'utilisateurs qui fleuriront avec Internet.

L'aventure PS/1 s'achève en 1994. IBM arrête la production, marquant un point d'arrêt dans sa stratégie. L'année suivante, les déclinaisons PS/VP et PS/2e disparaissent aussi du catalogue. Ces décisions traduisent le retrait progressif d'IBM du marché des ordinateurs personnels grand public, un secteur où la standardisation des composants et la guerre des prix réduisent drastiquement les marges.

Haskell

Une douzaine de langages fonctionnels coexistaient à la fin des années 1980, chacun avec ses particularités mais tous partageant des fondements sémantiques similaires. Cette dispersion irritait les chercheurs qui peinaient à faire adopter leurs idées au-delà de cercles restreints. L'idée d'unifier ces efforts germa lors d'une réunion en septembre 1987, pendant la conférence Functional Programming Languages and Computer Architecture à Portland. Les participants décidèrent de créer un comité pour concevoir un langage commun qui servirait de référence stable.

Cette démarche collective surprend de nos jours. On imagine difficilement qu'un comité puisse accoucher d'un langage élégant, tant la conception par consensus semble vouée aux compromis et aux incohérences. Pourtant, Haskell prouve le contraire. Le secret tient dans l'alignement des objectifs individuels et l'importance accordée à la beauté mathématique.

John Backus avait ouvert la voie en 1978 avec sa conférence Turing. Le créateur de Fortran y présentait la programmation fonctionnelle comme une alternative crédible au modèle von Neumann. Venant d'une telle figure, cette caution transforma la perception du paradigme fonctionnel, qui cessa d'être perçu comme une curiosité académique.

L'évaluation paresseuse fascine les concepteurs de Haskell. Cette technique, découverte indépendamment par plusieurs équipes dans les années 1970, change radicalement la manière de concevoir les programmes. Dan Friedman et David Wise à Indiana, Peter Henderson et James H. Morris Jr. chez Xerox PARC, David Turner à St Andrews et Kent : tous explorent cette voie prometteuse. Turner démontre son élégance dans SASL et KRC, utilisant les listes paresseuses pour simuler des comportements complexes avec une simplicité déconcertante.

Le 1er avril 1990 marque la publication du premier rapport Haskell.

Cette date, choisie par hasard, alimentera de nombreuses plaisanteries. Qui aurait cru qu'un langage né un jour de blague deviendrait si influent ? Le développement se poursuit pendant quinze années, principalement par courrier électronique. Une époque où les échanges techniques passaient encore par des messages réfléchis plutôt que par des notifications incessantes.

Miranda influence la conception de Haskell. David Turner commercialise ce langage fonctionnel non strict via sa société Research Software Limited, avec un succès notable : 250 universités et 50 entreprises l'adoptent. Haskell hérite de nombreuses caractéristiques syntaxiques de Miranda mais s'en distingue par des innovations marquantes.

Le système de classes de types constitue la première de ces innovations. Philip Wadler l'introduit en février 1988, résolvant élégamment les problèmes de surcharge des opérateurs numériques. Plus systématique que les solutions adoptées par Miranda ou SML, cette approche modulaire est depuis un modèle pour d'autres langages.

La pureté représente le second pilier de Haskell. En tant que langage fonctionnel pur, il garantit qu'une fonction retourne toujours le même résultat pour des arguments identiques, sans aucun effet de bord. Cette contrainte, directement liée à l'évaluation paresseuse, complique initialement la gestion des entrées-sorties. Mais cette difficulté apparente mène à l'invention des entrées-sorties monadiques, reconnues comme une contribution majeure à l'informatique.

Les monades transforment un problème technique en solution conceptuelle. Elles encapsulent les effets de bord dans un cadre mathématique rigoureux, préservant la pureté du langage tout en permettant les interactions avec le monde extérieur. Cette élégance théorique cache une complexité pratique qui rebute parfois les nouveaux venus.

En 1999, « Haskell 98 » stabilise le langage. La communauté réclamait cette standardisation, fatiguée des évolutions permanentes qui compromettaient la portabilité du code. Le comité se dissout alors, laissant le langage évoluer librement. Cette approche à deux vitesses fonctionne remarquablement bien : Haskell sert à la fois de laboratoire pour explorer des idées avancées et d'outil pratique pour des applications réelles.

Les versions successives enrichissent progressivement le langage. La 1.1 en 1991, la 1.2 en 1992, la 1.3 en 1996, la 1.4 en 1997 : chaque itération apporte son lot d'améliorations. Ce rythme soutenu témoigne de la vitalité d'une communauté engagée dans l'exploration de nouveaux territoires conceptuels.

Les innovations du Haskell inspirent de nombreux langages et frameworks. Les systèmes de types avancés, la gestion des effets via les monades, l'évaluation paresseuse : autant de concepts qui rayonnent aujourd'hui dans

l'écosystème informatique. Les langages impératifs intègrent aussi des éléments fonctionnels, témoignant de l'influence durable de cette approche.

Le nom « Haskell » honore Haskell B. Curry, mathématicien et logicien dont les travaux sur le calcul combinatoire influencent la théorie des langages de programmation. Ce choix n'est pas anodin car il ancre le langage dans une tradition mathématique rigoureuse, rappelant que la programmation fonctionnelle puise ses racines dans la logique formelle.

Haskell occupe une position singulière. Ni langage de masse ni curiosité académique, il maintient un équilibre subtil entre rigueur théorique et utilité pratique. Son influence se mesure moins à son adoption directe qu'à sa capacité à faire évoluer les pratiques de programmation. Dans un monde où l'élégance cède souvent le pas à l'efficacité, Haskell rappelle qu'il existe d'autres voies, plus exigeantes mais infiniment plus satisfaisantes.

HTML

Le langage HTML naît en 1991 au CERN, dans le bureau de Tim Berners-Lee. Ce physicien britannique travaille sur un problème concret : comment partager des documents entre chercheurs dispersés dans le monde entier ? Sa solution tient en quelques lignes de code qui vont bouleverser notre rapport à l'information. L'HyperText Markup Language, c'est son nom complet, s'inspire d'un standard existant, le Standard Generalized Markup Language (SGML), normalisé par l'ISO en 1986. Berners-Lee y ajoute un ingrédient la possibilité de créer des liens entre documents.

L'idée représente un saut conceptuel immense à l'époque. Imaginez un texte capable de vous emmener vers un autre texte d'un simple clic, puis vers un troisième, et ainsi de suite, créant une toile d'informations interconnectées. C'est exactement ce qui est rendu possible avec ses balises d'ancrage de HTML. Le langage fonctionne par marqueurs encadrés de chevrons : `<p>` pour les paragraphes, `<h1>` pour les titres principaux. Ces balises remplissent un double rôle, elles structurent le contenu et déterminent son apparence à l'écran.

Dès septembre 1991, une liste de diffusion baptisée www-talk rassemble les premiers passionnés du projet. Dave Raggett, ingénieur chez Hewlett-Packard, Marc Andreessen du National Center for Supercomputing Applications, et bien d'autres contribuent à enrichir le langage. Leurs échanges donnent naissance à HTML+, une version étoffée des développements futurs.

Cette effervescence créative cache un piège. Chaque navigateur – Lynx pour les terminaux texte, Mosaic pour les interfaces graphiques, Arena comme alternative européenne – développe ses propres extensions. Les « balises propriétaires » se multiplient, menaçant l'universalité chère à Berners-Lee. Un document lisible sur Mosaic est illisible sur Lynx. Le web risque l'éclatement avant d'avoir vraiment existé.

Pour éviter cette balkanisation, Berners-Lee fonde en 1994 le World Wide Web Consortium, le fameux W3C. L'initiative bénéficie du soutien financier du CERN, de la DARPA américaine et de la Commission européenne. Trois laboratoires hébergent cette nouvelle institution : le MIT aux États-Unis, l'INRIA en France et l'université Keio au Japon. Cette géographie reflète la volonté de faire du web un projet véritablement mondial.

Le premier chantier du W3C consiste à normaliser l'existant. HTML 2.0 sort en novembre 1995, recensant et harmonisant les balises utilisées par les différents navigateurs. Cette version coïncide avec un moment charnière : le prix des ordinateurs personnels chute sous les 5 000 dollars, les rendant accessibles au grand public. Aux États-Unis, des initiatives politiques encouragent le développement d'internet, pressentant son potentiel économique.

Mais la standardisation doit composer avec des intérêts contradictoires. Netscape et Microsoft, qui dominent le marché des navigateurs, cherchent à imposer leurs propres innovations pour se démarquer de la concurrence. Leurs ingénieurs inventent des balises spectaculaires : `<blink>` fait clignoter le texte chez Netscape, `<marquee>` le fait défiler chez Microsoft. Ces ajouts provoquent l'ire des puristes du W3C, qui y voient une confusion dangereuse entre structure documentaire et présentation visuelle.

Le débat dépasse la technique pure. Il oppose deux visions du web : d'un côté, celle des créateurs de contenu réclamant plus de contrôle sur l'apparence de leurs pages ; de l'autre, celle des pionniers défendant une séparation stricte entre fond et forme. Cette tension traverse encore les discussions sur l'évolution des standards web.

La solution émerge fin 1996 avec les feuilles de style CSS (Cascading Style Sheets). Cette nouvelle norme délègue les aspects visuels à un langage spécialisé, libérant HTML de cette responsabilité. La division du travail se formalise avec HTML 4.0 en décembre 1997 : au HTML la structure, au CSS l'habillage.

Le W3C se professionnalise. Un « Process Working Group » incluant Netscape, HP, IBM et Microsoft formalise les procédures de standardisation. Leur travail aboutit en 1999 à un « Process Document » qui établit les règles de gouvernance du consortium. Ces dispositifs bureaucratiques peuvent paraître fastidieux, mais ils garantissent la transparence et l'équité

du processus de normalisation.

L'histoire du HTML révèle les enjeux politiques cachés derrière l'apparente neutralité technique. L'interopérabilité prônée par le W3C entre en collision avec les stratégies commerciales des éditeurs de navigateurs. L'idéal d'un web décentralisé se heurte aux velléités de contrôle de certains acteurs dominants. Ces questions résonnent encore face à l'influence croissante de Google, Apple ou Meta sur l'évolution des standards.

Pourtant, HTML a tenu sa promesse originelle. Il a donné naissance à un web ouvert où chacun peut publier sans autorisation préalable. Cette vision s'inscrit dans la lignée des utopies cybernétiques des années 1960, qui rêvaient d'espaces numériques horizontaux et collaboratifs. Si le web contemporain s'éloigne parfois de ces idéaux fondateurs, HTML reste un pilier de l'architecture internet aux côtés du protocole HTTP et du système des URLs.

Le W3C poursuit sa mission. L'arrivée d'acteurs comme la Motion Picture Association of America ou les éditeurs scientifiques élargit le spectre des intérêts à concilier. L'industrie du divertissement réclame des dispositifs anti-piratage, les éditeurs veulent protéger leurs contenus, tandis que les défenseurs des libertés numériques craignent une restriction de l'ouverture du web.

Cette tension permanente entre innovation et préservation des principes fondateurs constitue l'ADN du HTML. Trente ans après sa création, ce langage continue d'évoluer tout en gardant sa simplicité originelle. Une balise est une balise, un lien est un lien. Dans un monde numérique en perpétuelle mutation, cette stabilité rassure. Elle rappelle qu'internet n'est pas qu'un marché ou un terrain de jeu technologique, mais aussi un espace public qu'il faut préserver et transmettre.

Gopher

À l'Université du Minnesota en 1990, Mark McCahill et Farhad Anklesaria dirigent une équipe confrontée à un problème concret, et semblable à celui de Tim Berners-Lee : comment organiser l'information sur le campus ? Le comité officiel propose une solution que l'équipe juge trop lourde et inadaptée. Ils décident de créer leur propre système.

Au printemps 1991, la première version de Gopher voit le jour. Le nom fait référence à deux choses : la mascotte de l'université (un gaufre) et l'idée d'« aller chercher » l'information. L'équipe conçoit une architecture

hiérarchique, semblable à un système de fichiers. Les utilisateurs naviguent dans des menus pour accéder aux documents, comme ils le feraient dans une bibliothèque traditionnelle où les ouvrages sont classés par thèmes.

L'interface est volontairement simple. Des menus textuels, conçus pour fonctionner y compris sur des connexions lentes. Les créateurs intègrent un moteur de recherche en texte intégral grâce aux ordinateurs NeXT, permettant de retrouver les contenus recherchés. Cette combinaison entre navigation structurée et recherche libre séduit immédiatement.

La stratégie de diffusion suit le modèle du logiciel libre : le code source du serveur circule gratuitement, mais l'équipe garde le contrôle des clients pour maintenir la cohérence du système. Les administrateurs installent leurs serveurs sans difficulté et créent des liens vers d'autres ressources Gopher.

Le succès dépasse toutes les attentes. En novembre 1994, plus de 8 000 serveurs Gopher tournent sur Internet. Les bibliothécaires adoptent massivement le protocole, appréciant son organisation logique et sa simplicité d'usage. L'équipe développe des extensions comme Gopher+ pour gérer les métadonnées et différents formats de documents.

Cette histoire s'inscrit dans une période de transition majeure. L'informatique abandonne progressivement les mainframes au profit de systèmes distribués. L'équipe de McCahill, issue du support micro-informatique, défend une vision décentralisée qui s'oppose à celle des administrateurs des ordinateurs centraux. Cette tension reflète l'évolution des paradigmes de l'époque.

Gopher démocratise l'accès à l'information numérique avant l'arrivée du Web. Le « Gopherspace » est un espace d'information que les utilisateurs parcourent intuitivement. Le protocole résout élégamment le problème de l'organisation des contenus numériques en mariant navigation hiérarchique et recherche textuelle.

En 1993, l'université fait une erreur stratégique. Elle tente de monétiser Gopher en imposant des frais de licence aux utilisateurs commerciaux. Cette décision, dictée par des contraintes budgétaires, provoque des remous. Mais le véritable défi vient d'ailleurs : Tim Berners-Lee propose à l'équipe de fusionner Gopher et le World Wide Web. La complexité apparente du Web les fait hésiter.

Cette hésitation coûte cher. Mosaic popularise le Web et relègue progressivement Gopher au second plan. Le support limité du protocole dans ce navigateur, combiné à la richesse croissante des pages Web, détourne les utilisateurs vers le nouveau standard. L'équipe de McCahill perd le contrôle de l'expérience utilisateur, élément déterminant dans l'évolution d'un protocole.

Les idées développées pour Gopher survivent néanmoins. L'organisation

hiérarchique des contenus influence encore de nombreux systèmes actuels. Les contraintes qui ont guidé sa conception — bande passante limitée, simplicité d'utilisation — retrouvent une actualité avec les applications mobiles et les environnements à ressources contraintes.

L'expérience enseigne plusieurs leçons importantes. D'abord, l'importance du contrôle de l'interface utilisateur dans le succès d'un protocole réseau. Ensuite, les difficultés de la monétisation des protocoles Internet. Enfin, l'impact des choix architecturaux sur la pérennité d'un système. Avec ce projet, l'Université du Minnesota développe une expertise reconnue dans les systèmes d'information distribués.

Les questions soulevées par Gopher restent d'actualité : comment organiser l'accès à l'information ? Comment naviguer dans les espaces numériques ? Comment concilier simplicité et fonctionnalités ? Les concepteurs de systèmes informatiques continuent de chercher des réponses élégantes à ces problématiques.

PCMCIA

En 1985, la JEIDA (Japan Electronic Industries Development Association) découvre un problème qui va marquer l'industrie informatique. Les ordinateurs portables commencent à se démocratiser, mais ils souffrent d'une limitation frustrante : impossible d'y ajouter facilement des périphériques comme on le ferait sur une machine de bureau. Chaque fabricant développe ses propres formats de cartes mémoire, créant une cacophonie technologique où rien n'est compatible.

C'est à San Jose, en Californie, que tout bascule en 1989. Une vingtaine d'entreprises américaines se retrouvent autour d'une table pour créer la PCMCIA. L'acronyme « Personal Computer Memory Card International Association » cache l'ambition de faire en sorte que toutes les cartes mémoire parlent le même langage. La société Poqet Computer, pionnière dans ce domaine, avait imaginé un ordinateur fonctionnant uniquement avec des cartes mémoire amovibles. Mais convaincre les éditeurs de logiciels de distribuer leurs programmes sur ces supports relevait du parcours du combattant sans standard unifié.

Le premier standard PCMCIA voit le jour en juin 1990. Fruit d'une collaboration avec la JEIDA, il définit des cartes au format d'une carte de crédit, équipées d'un connecteur à 68 broches. Cette version 1.0 se contente de régir les cartes mémoire, mais les ingénieurs des comités techniques com-

prennent vite le potentiel plus large de ce format. Pourquoi ne pas y greffer des modems ou des cartes réseau ?

Un an plus tard, en septembre 1991, la version 2.0 franchit le cap. Les cartes PCMCIA transforment les ordinateurs portables en plateformes modulaires telles de véritables extensions. Le standard décline alors trois épaisseurs différentes : les cartes Type I de 3,3 mm pour la mémoire, les Type II de 5 mm pour les périphériques de communication, et les imposantes Type III de 10,5 mm destinées aux disques durs miniatures.

L'architecture logicielle imaginée par les concepteurs du PCMCIA témoigne d'une vision remarquable. Trois couches s'articulent harmonieusement : les Socket Services gèrent le matériel au plus bas niveau, les Card Services s'occupent de l'allocation des ressources et de la configuration automatique, tandis que les pilotes spécifiques à chaque carte coiffent l'ensemble. L'insertion et le retrait des cartes sans extinction du système est désormais possible.

PCMCIA transforme l'informatique mobile. Les fabricants rivalisent d'ingéniosité pour proposer modems, cartes réseau, adaptateurs sans fil et multiples périphériques dans ce format compact. Les ordinateurs portables gagnent enfin la flexibilité qui leur manquait, devenant de véritables couteaux suisses électroniques.

Les évolutions techniques s'enchaînent au rythme des besoins du marché. 1995 marque l'arrivée de la version 5.0 avec le bus CardBus 32 bits, capable de transférer jusqu'à 132 Mo/s. Les cartes basse consommation 3,3V apparaissent, la gestion d'alimentation se raffine, et les cartes multifonctions voient le jour. L'année suivante, le Zoomed Video crée une liaison directe entre carte PCMCIA et contrôleur graphique, libérant le processeur des tâches de décodage vidéo.

Ce qui impressionne dans le PCMCIA, c'est sa sophistication technique précoce. Chaque carte embarque son propre système d'information CIS, une sorte de carte d'identité électronique qui décrit ses capacités au système d'exploitation. Cette approche plug-and-play devance de plusieurs années l'arrivée de l'USB. L'ordinateur reconnaît automatiquement la carte insérée et configure les ressources nécessaires, sans intervention de l'utilisateur.

La robustesse physique des cartes PCMCIA établit des références durables. Les spécifications exigent 10 000 cycles d'insertion en bureau et 5 000 en conditions difficiles. La conception électrique protège les composants : les broches d'alimentation se connectent en premier et se déconnectent en dernier, évitant les surtensions destructrices.

Au tournant des années 2000, CardBay modernise l'interface en introduisant une liaison série haute vitesse basée sur l'USB, tout en préservant la compatibilité avec l'existant. Cette approche illustre la maturité du stan-

dard, capable d'évoluer sans casser l'écosystème construit autour de lui.

Les concepts d'insertion à chaud et de configuration automatique du PCMCIA inspirent directement l'USB et le PCI Express. La coopération entre industriels américains et japonais démontre qu'un standard mondial peut émerger de la collaboration plutôt que de la guerre commerciale.

Au-delà de l'informatique portable, le PCMCIA essaime dans l'électronique grand public. Appareils photo numériques, décodeurs télévision, systèmes embarqués automobiles adoptent ce format pour sa fiabilité et sa souplesse d'usage. Cette polyvalence témoigne de la qualité de conception originelle.

Pourtant, le PCMCIA n'échappe pas aux limites de son époque. Son bus simple, hérité d'une interface mémoire directe, manque de signaux de synchronisation élaborés. La largeur de 16 bits restreint les performances des applications gourmandes. Ces contraintes techniques préparent son remplacement progressif par des standards plus récents comme l'ExpressCard.

Le PCMCIA appartient à l'histoire de l'informatique, mais il a prouvé qu'un consortium industriel pouvait créer un standard durable, capable de stimuler l'innovation tout en résolvant des problèmes d'interopérabilité complexes.

MP3

En 1987, dans les laboratoires de l'Institut Fraunhofer en Allemagne, des chercheurs travaillaient sur le Digital Audio Broadcasting, un projet qui paraissait anodin. Personne ne se doutait alors qu'ils étaient en train de poser les fondements d'une révolution qui allait secouer l'industrie musicale mondiale. Le CD venait juste de faire ses premiers pas, cinq ans plus tôt, et se posait la question de la compression audio numérique.

Karlheinz Brandenburg, diplômé en génie électrique et mathématiques de l'université d'Erlangen, dirigeait ces recherches avec une obsession : comment réduire drastiquement la taille des fichiers audio sans massacrer leur qualité ? La réponse résidait dans les bizarreries de notre système auditif. L'oreille humaine cache des failles que Brandenburg comptait bien exploiter. Certains sons en masquent d'autres, certaines fréquences disparaissent dans l'ombre d'autres plus dominantes. Ces phénomènes psychoacoustiques allaient devenir les piliers du futur MP3.

L'année 1988, l'ISO crée le Moving Pictures Experts Group. Ce consortium international avait pour mission d'établir des standards de compres-

sion audio et vidéo. Les travaux de Fraunhofer trouvaient enfin leur cadre institutionnel. Quatre années de développement s'écoulèrent avant que la norme MPEG-1 Layer 3 ne voie officiellement le jour en 1992. Ce nom barbare allait bientôt se muer en trois lettres qui changeraient le monde : MP3.

La magie opère grâce à un processus d'une complexité redoutable. Le signal audio subit d'abord un découpage temporel minutieux, puis passe sous le crible d'un banc de filtres hybrides mêlant transformation en cosinus discrète modifiée et filtrage polyphase. Un modèle psychoacoustique calcule ensuite les seuils de masquage pour chaque bande fréquentielle. Tout ce que l'oreille ne perçoit pas naturellement disparaît purement et simplement du fichier final. Seules subsistent les informations sonores réellement audibles, encodées avec le soin nécessaire.

Cette approche chirurgicale produit des résultats spectaculaires. Un morceau de musique occupe environ 10 mégaoctets par minute sur un CD. Avec une compression MP3 à 128 kilobits par seconde, ce même morceau tient dans 1 mégaoctet. Dix fois moins d'espace pour une qualité que la plupart des auditeurs jugent convenable. Cette prouesse technique allait bientôt rencontrer Internet.

Le milieu des années 1990 voit naître une convergence explosive. Internet s'étend, les connexions s'améliorent, et le MP3 trouve sa voie. En 1995, le format circule sur les réseaux informatiques. WinAmp, petit logiciel de lecture audio, accompagne cette diffusion et transforme l'ordinateur personnel en chaîne hi-fi. Mais cette démocratisation s'accompagne d'un phénomène imprévu : le piratage musical explose.

Napster fait trembler l'industrie du disque. Cette plateforme d'échange peer-to-peer met en relation des millions d'internautes désireux de partager leur collection musicale numérisée. Les maisons de disques découvrent avec effroi que leur modèle économique vacille. Les procès pleuvent, les débats sur le droit d'auteur s'enflamment. Le MP3 est le symbole d'une bataille juridique et économique sans précédent.

L'industrie riposte en créant l'offre légale. Saehan Information Systems commercialise le premier baladeur MP3 portable en 1998. Trois ans plus tard, Apple frappe un grand coup avec l'iPod et iTunes. Steve Jobs réussit là où beaucoup ont échoué : créer un écosystème viable alliant matériel et distribution légale de musique numérique. L'iPod est l'objet technologique incontournable du début du XXI^{e} siècle.

La souplesse technique du MP3 explique en partie son succès. Le format propose une gamme de débits allant de 32 à 320 kilobits par seconde. Chaque utilisateur adapte le compromis taille-qualité selon ses besoins et contraintes. Trois modes de compression coexistent : CBR maintient un

débit constant, VBR fait varier le débit selon la complexité du passage musical, ABR trouve un compromis entre les deux approches. Cette diversité satisfait aussi bien l'audiophile exigeant que l'utilisateur pressé.

Sous le capot, la technologie impressionne par sa sophistication. L'analyse spectrale repose sur la transformée de Fourier rapide qui décompose le signal en ses composantes fréquentielles. Un mécanisme de « réservoir de bits » redistribue dynamiquement le débit entre les trames selon les besoins de chaque segment audio. Cette gestion intelligente optimise chaque octet disponible.

Pourtant, le MP3 n'échappe pas aux compromis inhérents à toute compression. Des artefacts peuvent apparaître, comme ces pré-échos qui précèdent les attaques instrumentales brutales. Les hautes fréquences perdent parfois de leur netteté. Ces défauts, imperceptibles pour beaucoup, agacent les puristes et motivent le développement de successeurs plus performants comme l'AAC ou l'Opus.

L'aventure juridique du MP3 trouve son épilogue en 2017. Les brevets de l'Institut Fraunhofer et de Thompson Licensing expirent cette année-là. Leurs créateurs annoncent officiellement l'arrêt du support du format, encourageant l'adoption de technologies plus récentes. Une page se tourne après vingt-cinq années de domination.

Le MP3 a métamorphosé notre rapport à la musique. Fini les rayonnages de CD, terminés les walkman et leurs cassettes fragiles. La musique est nomade, accessible partout, copiable à l'infini. Cette liberté nouvelle bouleverse les habitudes d'écoute mais interroge sur la valorisation du travail artistique. Comment rémunérer équitablement les créateurs dans un monde où leur œuvre se dématérialise ?

Au-delà de la musique, le MP3 illustre parfaitement comment la recherche fondamentale peut transformer la société. Les travaux sur la psychoacoustique et le traitement du signal ont révolutionné une industrie entière. Cette technologie démontre aussi la puissance des standards ouverts et de la recherche publique dans l'innovation technologique.

L'héritage du MP3 se retrouve dans tous les codecs audio modernes. Spotify, Deezer, Apple Music s'appuient sur les principes qu'il a établis. Supplanté techniquement, le MP3 reste le format de référence dans l'inconscient collectif.

WAVE

En 1991, quand Microsoft et IBM unissent leurs efforts, ils créent bien plus qu'un simple format de fichier audio. Le WAVE naît dans cette collaboration, à une époque où l'ordinateur personnel découvre vraiment le multimédia. Les machines commencent à produire autre chose que des bips électroniques, et il faut bien stocker cette nouvelle richesse sonore quelque part.

Le choix technique s'oriente vers RIFF, cette architecture de fichier pensée comme un grand conteneur modulaire. Chaque élément trouve sa place dans des « chunks », ces petits segments identifiés par quatre caractères. Une approche qui fait alors évoluer la gestion des données multimédias. Les développeurs apprécient cette souplesse de pouvoir ajouter, retirer, réorganiser les informations sans casser la structure générale.

La philosophie du WAVE tient en quelques mots : préserver l'intégrité du son. Pas de compression, pas d'artifice, juste la représentation numérique brute du signal analogique. Le format se contente de deux éléments essentiels : le segment « fmt » qui décrit les caractéristiques techniques, et le « data » qui contient les échantillons sonores. Cette simplicité apparente cache en réalité une robustesse remarquable.

Les années 1990 voient le WAVE s'imposer progressivement. Sa compatibilité naturelle avec Windows lui donne un avantage certain, mais les systèmes Macintosh et Linux l'adoptent aussi. Les professionnels du son découvrent un outil fiable, capable de restituer exactement ce qu'ils ont enregistré. Fini les dégradations dues aux algorithmes de compression : ce qu'on entend à la sortie correspond fidèlement à ce qu'on a capturé à l'entrée.

L'Union européenne de radio-télévision comprend vite l'intérêt du format. En 1997, elle lance le Broadcast Wave Format, une extension qui enrichit le WAVE de métadonnées spécialisées. Soudain, un fichier audio raconte son histoire : qui l'a créé, quand, dans quelles conditions, avec quel matériel. Cette traçabilité devient vite indispensable dans l'industrie audiovisuelle, où chaque seconde de programme doit être documentée et vérifiable.

Mais le WAVE révèle aussi ses limites. Cette frontière des 4 Go, héritée de la structure RIFF originale, se trouve problématique quand les studios s'orientent vers la haute résolution. Un enregistrement de concert en 96 kHz peut facilement dépasser cette limite. L'UER réagit en 2009 avec RF64, une extension qui repousse ces contraintes techniques. Les archivistes et les producteurs peuvent enfin travailler sans se soucier de la durée de leurs enregistrements.

La représentation des données dans le WAVE suit une logique précise. Les échantillons 8 bits utilisent des octets non signés, de 0 à 255, tandis que le 16 bits adopte le complément à deux, entre -32 768 et 32 767. Cette cohérence technique facilite grandement le travail des programmeurs et garantit une conversion fidèle vers le signal analogique.

Le monde archivistique adopte massivement le format. L'Association Internationale des Archives Sonores et Audiovisuelles en fait une recommandation officielle. Sa logique ? Un format simple, documenté, sans compression, c'est la garantie d'une lecture possible dans cinquante ans. La British Library numérise ses collections audio en WAVE, tout comme la Bibliothèque du Congrès américain. Ces choix institutionnels ancrent définitivement le format dans la culture de la conservation numérique.

Dans les années 2000, FLAC arrive et compresse sans perte, divisant par deux la taille des fichiers tout en préservant la qualité audio. Pourtant, le WAVE ne faiblit pas. Dans les studios d'enregistrement, les salles de montage, les archives, il reste la référence. Sa simplicité constitue finalement sa force : pas d'algorithme complexe à décoder, pas de brevet à contourner, juste des données brutes et lisibles.

Le format évolue discrètement. Les métadonnées s'enrichissent avec les segments INFO, les balises XMP ou ID3. Ces ajouts transforment progressivement le WAVE d'un simple conteneur audio en véritable document numérique, capable de porter des informations contextuelles riches. Les workflows professionnels modernes tirent parti de cette flexibilité pour automatiser la gestion des contenus.

Des outils comme JHOVE accompagnent cette évolution en proposant validation et analyse technique. Vérifier la conformité d'un fichier WAVE, extraire ses métadonnées, détecter d'éventuelles corruptions sont des fonctions indispensables quand on gère des milliers d'heures d'archives audio. La communauté développe ainsi tout un écosystème autour du format.

Plus de trente ans après sa création, le WAVE traverse les époques sans prendre une ride. Sa longévité s'explique par un équilibre subtil entre simplicité technique et polyvalence d'usage. Là où d'autres formats disparaissent avec leurs technologies d'origine, le WAVE persiste parce qu'il répond au besoin de conserver le son dans sa forme la plus pure, sans artifice ni compromis.

Le format audio de Microsoft et IBM de 1991 continue de capturer nos sons les plus précieux, preuve que certaines innovations traversent les générations sans perdre leur pertinence.

Python

C'est une histoire qui commence par un week-end de décembre 1989. Guido van Rossum, chercheur au Centre de Mathématiques et d'Informatique des Pays-Bas, profite de ses congés pour bricoler un nouveau langage de programmation. Rien de très ambitieux au départ : il veut créer quelque chose de plus pratique qu'ABC, un langage sur lequel il a travaillé, quelque chose qui fonctionne mieux avec UNIX sans pour autant y être enchaîné.

Le nom qu'il choisit trahit son goût pour l'humour britannique. Python, c'est un clin d'œil aux *Monty Python's Flying Circus*, cette troupe qui a marqué la comédie anglaise. Ce choix annonce l'état d'esprit qui va habiter la communauté des développeurs : un mélange de sérieux technique et de second degré bienvenu dans un monde parfois trop austère.

Il publie sa première version en février 1991 sur alt.sources, ce forum où les programmeurs échangent leurs créations. La version 0.9.0 contient ce qui va faire la singularité de Python, à savoir une syntaxe qui privilégie la lisibilité, une approche qui tranche avec les langages de l'époque. L'indentation, élément de syntaxe à part entière, est un héritage direct d'ABC. Cette idée fait grincer des dents au début. Les programmeurs habitués aux accolades et aux points-virgules trouvent cela étrange. Pourtant, cette contrainte va se révéler libératrice : elle force à écrire du code propre et lisible.

La philosophie se cristallise autour d'un principe simple : « Il devrait y avoir une façon unique et préférable de faire les choses ». Cette maxime guide toutes les décisions de conception. Là où d'autres langages multiplient les syntaxes pour une même opération, Python fait le choix de la simplicité. Le code est plus prévisible, donc plus facile à maintenir et à comprendre.

Python 1.0 arrive en janvier 1994 avec un arsenal fonctionnel solide. Que ce soit lambda, map, filter ou reduce, ces outils empruntés à la programmation fonctionnelle enrichissent les possibilités d'expression. Le langage trouve ses marques, attire ses premiers adeptes. La version 1.5 de décembre 1997 consolide ces fondations et prépare le terrain pour la suite.

L'an 2000 apporte Python 2.0 et ses nouveautés. Le ramasse-miettes automatise la gestion mémoire, libérant le programmeur de cette corvée. Les compréhensions de liste introduisent une syntaxe élégante pour manipuler les collections : `[x**2 for x in range(10)]` remplace avantageusement plusieurs lignes de boucles traditionnelles. Ces ajouts ne trahissent pas l'esprit originel mais l'affinent.

Le succès grandissant de Python révèle une approche différente du développement. Le langage encourage la modularité sans l'imposer brutalement. Les modules se combinent naturellement, le code se réutilise sans

acrobaties. Cette fluidité séduit des programmeurs lassés par la complexité d'autres outils.

Puis vient 2008 et Python 3.0, une version qui bouleverse les habitudes. Van Rossum et son équipe font le choix de la rupture pour corriger des défauts de conception. Cette décision courageuse fracture temporairement la communauté : les anciens programmes ne fonctionnent plus directement avec la nouvelle version. Mais cette audace paie à long terme. Python 3 pose des bases plus saines, plus cohérentes.

Le typage dynamique de Python facilite l'écriture rapide de prototypes. Pas besoin de déclarer le type d'une variable avant de l'utiliser, le langage s'en charge. Cette souplesse accélère le développement initial tout en demandant parfois plus de rigueur dans les tests. La gestion des erreurs, inspirée de Modula-3, structure les programmes sans les alourdir.

L'orientation objet s'intègre naturellement dans Python dès ses premières versions. Pas de choc conceptuel, juste une approche pragmatique qui laisse le choix au programmeur, comme écrire du code procédural simple ou organiser ses programmes en classes sophistiquées selon les besoins.

La richesse de la bibliothèque standard distingue Python de ses concurrents. « Batteries incluses », dit-on dans la communauté. E-mail, web, cryptographie, compression : la plupart des besoins courants trouvent une réponse dans la distribution de base. Cette abondance évite de chercher des bibliothèques tierces pour des tâches standard.

L'accessibilité fait de Python un langage de choix pour l'apprentissage. Sa syntaxe, proche du langage naturel, démystifie la programmation. Des livres comme *Python for Kids* popularisent son usage pédagogique. Les universités l'adoptent massivement pour initier leurs étudiants.

PyPI, l'index des paquets Python, transforme le partage de code en jeu d'enfant. Une simple commande `pip install` suffit pour ajouter des fonctionnalités à un projet. Cette facilité nourrit un écosystème dynamique où chacun contribue et bénéficie du travail des autres.

Les PEP (Python Enhancement Proposals) organisent l'évolution du langage de manière démocratique. Ces propositions d'amélioration structurent les débats, documentent les choix techniques. Cette gouvernance ouverte maintient la cohérence tout en accueillant les contributions externes.

Google adopte Python dès ses débuts pour certaines parties de son moteur de recherche. Cette légitimation par une entreprise technologique majeure ouvre des portes. D'autres organisations suivent, attirées par la productivité du langage et la qualité de son écosystème.

L'explosion de l'intelligence artificielle propulse Python au premier plan. TensorFlow, PyTorch, scikit-learn : les bibliothèques spécialisées se multiplient. Les data scientists adoptent massivement le langage pour ses capaci-

tés d'analyse et de visualisation. NumPy transforme Python en concurrent sérieux de MATLAB pour le calcul scientifique.

Django révolutionne le développement web en Python. Ce framework propose une approche complète : ORM intégré, interface d'administration automatique, système de templates sophistiqué. Instagram, Pinterest, Mozilla utilisent Django pour leurs sites à fort trafic, prouvant sa robustesse industrielle.

Les versions récentes peaufinent l'expérience utilisateur. Python 3.8 introduit l'opérateur morse (:=) qui assigne une valeur tout en l'utilisant dans une expression. Python 3.9 améliore les performances du dictionnaire, structure de données centrale du langage. Ces évolutions traduisent une maturité technique croissante.

La portabilité reste un atout majeur : un programme Python fonctionne identiquement sous Windows, macOS ou Linux. Cette universalité, combinée à la gratuité et au modèle open source, élimine bien des barrières à l'adoption. Les entreprises apprécient cette liberté qui les protège du verrouillage technologique.

L'intégration avec d'autres langages ouvre des perspectives. Cython compile du Python vers du C pour gagner en performance. Jython transpile (convertit du code dans un autre langage) vers Java, PyPy recompile à la volée. Ces passerelles permettent d'optimiser les parties critiques sans abandonner la productivité de Python.

Python s'impose dans des domaines variés : automatisation système, développement web, analyse de données, intelligence artificielle, calcul scientifique. Cette polyvalence reflète les choix de conception initiaux de van Rossum. En privilégiant la simplicité et la lisibilité, il a créé un outil qui s'adapte aux besoins changeants de l'informatique. Trente ans après ce week-end de décembre 1989, le langage continue d'évoluer tout en gardant son âme : rendre la programmation accessible sans sacrifier la puissance.

Linux

UNIX était un système d'exploitation écrit en langage C qui pouvait s'adapter à différents matériel. C'était un moyen de contourner le casse-tête des ordinateurs de qui ne se parlaient pas entre constructeurs, comme ceux d'IBM et de Burroughs par exemple. Seul le noyau nécessitait des ajustements spécifiques à chaque architecture. Malin, mais les licences UNIX coûtaient une fortune : les revendeurs gonflaient les prix jusqu'à dix fois le

tarif initial.

Andrew Tanenbaum, professeur américain installé aux Pays-Bas, eut alors une autre approche. Il voulait montrer à ses étudiants comment fonctionnait réellement un système d'exploitation. Son MINIX, taillé pour les processeurs Intel 8086 qui envahissaient le marché, n'égalait pas les performances des systèmes commerciaux. Sa force résidait ailleurs : Tanenbaum publia les 12 000 lignes de code dans son livre « Operating Systems : Design and Implementation ». On voyait déjà cette pratique pour des programmes en BASIC sur des ordinateurs personnels, comme des jeux, mais pas pour des systèmes d'exploitation entiers. Les programmeurs curieux pouvaient enfin regarder sous le capot.

Linus Torvalds, étudiant en deuxième année d'informatique à Helsinki, faisait partie de ces lecteurs passionnés. Ce Finlandais autodidacte découvrait simultanément les idées de Richard Stallman. Stallman avait démarré sa carrière au laboratoire d'IA du MIT, où il créa l'éditeur Emacs. Au début des années 1980, il observa avec amertume que les entreprises logicielles débauchaient les meilleurs programmeurs en leur imposant des clauses de confidentialité draconiennes. Selon lui, les logiciels devaient rester libres, copiables et modifiables par tous.

Le projet du système GNU (acronyme récursif pour « GNU is Not UNIX ») vit le jour en 1983. Stallman commença par le compilateur GCC en 1984, un chef-d'œuvre technique qui surpassait les réalisations d'équipes entières de développeurs commerciaux. Mais il manquait encore le noyau du système.

Torvalds, inspiré par MINIX, s'en chargea et sa première version Linux (0.01) apparut en septembre 1991, suivie par la 0.02 en octobre. L'accueil fut immédiat : des programmeurs du monde entier téléchargeaient le code, l'essayaient, l'amélioraient et renvoyaient leurs contributions à Torvalds. Une dynamique collaborative inédite se mettait en marche.

Tanenbaum ne fut pas tendre avec ce petit nouveau. En 1991, il critiqua violemment l'architecture monolithique du noyau Linux, la qualifiant d'erreur de conception majeure. « Linux est obsolète », asséna-t-il. Torvalds encaissa les critiques et poursuivit son travail. L'histoire lui donnera raison.

L'association avec les programmes GNU transforma Linux en système complet. La licence GPL garantissait que le code source resterait accessible, créant un cercle vertueux : plus de contributeurs, plus d'améliorations, plus d'utilisateurs. Les étudiants et programmeurs affluaient vers ce terrain de jeu libre.

Red Hat et Debian lancèrent les premières distributions commerciales, assemblant des logiciels précompilés pour simplifier l'installation. Les interfaces graphiques KDE et GNOME rendirent le système accessible aux

non-initiés. Linux sortait des laboratoires pour conquérir les bureaux.

Le manchot Tux, mascotte du projet, raconte une anecdote savoureuse. Lors de vacances dans l'hémisphère sud, Torvalds rencontra un pingouin qui lui mordit la main. Cet incident cocasse inspira plus tard le choix du symbole. L'humour n'était jamais loin dans cette aventure.

Linux développa sa propre philosophie. Chaque outil accomplit une seule tâche, mais la réalise parfaitement. L'interface traite fichiers et périphériques d'entrée-sortie de façon uniforme. Les utilisateurs combinent ces outils simples pour créer des fonctionnalités sophistiquées, personnalisant leur environnement selon leurs goûts.

La robustesse impressionne : un programme défaillant ne fait pas s'effondrer le système entier. La mémoire de chaque processus reste cloisonnée, empêchant les interférences non autorisées. L'environnement de développement intègre nativement une panoplie d'outils de programmation.

Linux est la transformation d'un projet étudiant en phénomène planétaire. Il fait tourner des serveurs web, des supercalculateurs et nos appareils mobiles Android. Une communauté internationale de développeurs prouve qu'elle peut créer et maintenir un système d'exploitation sophistiqué, bousculant les modèles propriétaires établis. Le petit noyau de Linus a grandi jusqu'à défier les géants de l'industrie, portant les couleurs du développement collaboratif et de l'ouverture du code source.

Visual Basic

Visual Basic commence avec une idée simple : rendre accessible la programmation Windows à un public plus large que les seuls experts en C ou en assembleur. En mai 1991, Microsoft frappe un grand coup en lançant Visual Basic 1.0. Pour la première fois, créer une interface graphique ne nécessite plus d'écrire des centaines de lignes de code obscur. Il suffit de glisser-déposer des boutons, des zones de texte et des menus directement sur l'écran.

Cette approche visuelle bouleverse les habitudes. Là où il fallait auparavant maîtriser les subtilités de l'API Windows et manipuler des structures complexes, Visual Basic propose une logique intuitive : ce que vous voyez à l'écran correspond exactement à ce que verra l'utilisateur final. Un bouton placé dans un coin de la fenêtre apparaîtra au même endroit dans l'application terminée.

Microsoft ne s'arrête pas là. En septembre 1992, une version DOS voit

le jour, tentant de transposer cette facilité d'usage dans l'environnement texte. Les caractères ASCII étendus simulent tant bien que mal des éléments graphiques, mais l'expérience est rudimentaire comparée à la version Windows.

La version 2.0 de novembre 1992 corrige les premiers défauts et accélère l'exécution des programmes. L'environnement de développement gagne en fluidité, ce qui encourage encore plus de programmeurs à franchir le pas. Avec la 3.0 de l'été 1993, Microsoft introduit une nouveauté de taille : l'accès direct aux bases de données. L'intégration avec Access simplifie la création d'applications de gestion, un marché en pleine expansion.

Visual Basic 4.0 sort en 1995. Windows 95 démocratise les systèmes 32 bits, et Microsoft adapte son langage en conséquence. Cette version navigue entre deux mondes : elle génère du code 16 bits pour les anciens systèmes tout en exploitant les capacités des nouveaux processeurs 32 bits. La programmation orientée objet fait son apparition, timidement certes, mais elle ouvre de nouvelles perspectives architecturales.

Deux ans plus tard, Visual Basic 5.0 abandonne définitivement le 16 bits. Cette décision audacieuse témoigne de la confiance de Microsoft dans l'avenir du 32 bits. La compilation native remplace l'interprétation traditionnelle du BASIC, accélérant significativement l'exécution des programmes. Les développeurs découvrent la possibilité de créer leurs propres contrôles réutilisables, ouvrant la voie à un écosystème de composants tiers.

Visual Basic 6.0, sorti mi-1998, s'aventure sur le terrain du web. Internet Explorer évolue vers une plateforme de développement à part entière, permettant de créer des applications hybrides mêlant desktop et web. Cette version connaît un succès phénoménal et semble toujours utilisée dans certaines entreprises, malgré l'arrêt officiel du support.

L'arrivée du nouveau millénaire nous apporte Visual Basic .NET en 2002. Microsoft opère une rupture totale avec le passé. Le code géré, la gestion automatique de la mémoire et l'intégration au Framework .NET transforment radicalement la nature du langage. Cette transition brutale déstabilise une partie de la communauté, habituée à la simplicité des versions précédentes.

Visual Basic .NET 2003 consolide cette nouvelle orientation. Le support du .NET Compact Framework ouvre les portes du développement mobile, confidentiel à cette époque. Microsoft propose des outils de migration automatique, mais la transformation est souvent laborieuse.

En 2005, Microsoft simplifie la nomenclature et abandonne le suffixe .NET. Visual Basic 2005 récupère une partie de son identité perdue. La fonctionnalité « Edit and Continue » révolutionne le débogage en autorisant les modifications de code en temps réel. L'espace de noms « My »

facilite l'accès aux ressources système, retrouvant l'esprit de simplicité des premières versions.

Les versions suivantes accompagnent l'évolution du Framework .NET : 2008 exploite la version 3.5, 2010 s'appuie sur la 4.0, 2012 utilise la 4.5. Chaque itération apporte son lot d'améliorations techniques, mais l'essence du langage reste fidèle à sa mission initiale : démocratiser le développement logiciel.

La programmation événementielle constitue l'âme de Visual Basic. À l'inverse des programmes traditionnels qui suivent un cheminement linéaire du début à la fin, les applications Visual Basic vivent au rythme des interactions utilisateur. Un clic sur un bouton déclenche une procédure, une frappe au clavier active une autre fonction. Cette approche réactive transforme la logique de la programmation.

Cette philosophie événementielle reflète l'évolution de l'informatique personnelle. Les ordinateurs des années 1990 sont de plus en plus interactifs, et les utilisateurs attendent des interfaces réactives et intuitives. Visual Basic répond parfaitement à cette attente en proposant un modèle de développement aligné sur cette nouvelle réalité.

L'adoption massive de Visual Basic témoigne de sa pertinence : à la fin des années 1990, neuf applications Windows sur dix sortent de cet environnement. Cette domination s'explique par l'explosion du marché informatique, la standardisation de Windows et surtout l'accessibilité du langage.

Visual Basic a métamorphosé le paysage du développement logiciel en rendant la création d'applications Windows accessible à un public bien plus large que les seuls informaticiens chevronnés. Des comptables créent leurs propres outils de gestion, des ingénieurs développent des applications de calcul, des enseignants programment des logiciels éducatifs. Cette démocratisation redistribue les cartes de l'industrie logicielle.

DEC OSF/1

En 1992, Digital Equipment Corporation traversait une période de mutation. L'entreprise, autrefois géant de la mini-informatique, cherchait à s'imposer sur le marché des serveurs d'entreprise face à une concurrence de plus en plus féroce. Elle lance alors le développement d'une version particulière d'OSF/1, adaptée à sa nouvelle architecture Alpha et destinée aux serveurs AlphaServer.

Le projet naît d'une nécessité : créer un système capable de rivaliser avec

les solutions UNIX existantes tout en exploitant pleinement les capacités des processeurs Alpha. Les équipes d'ingénieurs se fixent des objectifs ambitieux : gérer au minimum deux processeurs, atteindre 120 transactions par seconde sur les tests TPC-A et supporter au moins 1 gigaoctet de mémoire. Ces spécifications représentaient un travail technique considérable.

L'architecture retenue s'appuie sur un bus système à huit nœuds maximum. Sept emplacements accueillent jusqu'à quatre processeurs chacun, le huitième étant réservé au pont système-vers-PCI. Cette conception modulaire répond à une philosophie claire : offrir une évolutivité sans compromettre les performances. Les ingénieurs intègrent des technologies émergentes comme les bus PCI et EISA, un pari risqué qui s'avérera payant pour la compatibilité avec l'écosystème PC.

La gestion de la mémoire pose des contraintes particulières. Le système doit jongler avec des processeurs de vitesses différentes tout en maintenant une cohérence parfaite des caches. La solution adoptée fait appel à un cache secondaire de 1 Mo utilisant des composants SRAM de 15 nanosecondes. Ce choix résulte d'un compromis délicat entre performance et coût, les mémoires plus rapides étant prohibitives pour un produit commercial.

Les tampons de flux constituent l'une des innovations les plus remarquables du système. Ces dispositifs surveillent les adresses des transactions de lecture et anticipent les besoins en préchargeant les données fréquemment utilisées, faisant chuter les temps d'accès de 9 à 7 cycles, un gain significatif qui se répercute sur les performances globales.

Côté entrées-sorties, DEC adopte une approche hybride entre PCI et EISA. Cette décision conservatrice s'explique par des contraintes économiques. Bien que plus ancien, l'EISA reste moins cher que le PCI tout en offrant des performances acceptables pour certaines applications. Le système supporte ainsi jusqu'à 18 ports réseau.

Le démarrage du système intègre des fonctionnalités de haute disponibilité soignées. Les dispositifs de stockage « hot-swap » permettent le remplacement à chaud des composants défaillants. Plus impressionnant encore, le système gère automatiquement les ROM flash corrompues sans intervention humaine, une prouesse technique qui évitera bien des cauchemars aux administrateurs système.

La compatibilité multi-OS représente un autre atout important. DEC OSF/1 cohabite avec Windows NT et OpenVMS sur la même machine, répondant ainsi aux besoins hétérogènes des entreprises. Cette polyvalence, rare à l'époque, séduira de nombreux clients soucieux de préserver leurs investissements logiciels existants.

La version 3.0, dévoilée en 1994, marque l'aboutissement de ces efforts. Le support complet du multiprocesseur symétrique (SMP) place le système

dans la cour des grands. Les développeurs ont repensé les mécanismes de synchronisation et les algorithmes du noyau pour tirer parti du parallélisme. Le résultat dépasse les attentes : les performances mesurées surpassent largement les objectifs initiaux.

L'accueil du marché est enthousiaste. L'AlphaServer 2100, équipé de DEC OSF/1, est la référence en termes de rapport prix-performances. Les tests indépendants confirment la supériorité du système, consolidant la position de DEC sur le marché des serveurs d'entreprise.

Cette réussite technique influence durablement l'industrie. Les concepts développés pour DEC OSF/1 – gestion avancée de la mémoire, support multiprocesseur optimisé, architecture modulaire – essaiment dans d'autres systèmes. L'approche équilibrée entre innovation et pragmatisme sera un modèle pour les développements ultérieurs.

En 1995, DEC rebaptise son système Digital UNIX, puis Tru64 UNIX après le rachat par Compaq en 1998. HP, qui absorbe Compaq en 2002, poursuit le développement jusqu'en 2012. Cette longévité témoigne de la solidité des fondations posées par les équipes de DEC au début des années 1990. Rétrospectivement, DEC OSF/1 illustre parfaitement les enjeux de cette décennie charnière. Face à la montée en puissance des architectures RISC et à l'explosion des besoins en calcul distribué, DEC a su créer un système à la hauteur des ambitions.

Blowfish

En 1993, alors que le Data Encryption Standard montrait ses premiers signes de faiblesse avec sa clé de 56 bits devenue vulnérable aux attaques par force brute, Bruce Schneier présentait au monde Blowfish, un algorithme de chiffrement symétrique qui allait révolutionner l'approche de la sécurité informatique.

Il n'avait pas choisi le hasard pour développer cet algorithme. Les solutions propriétaires dominaient le marché, souvent assorties de brevets contraignants et de coûts prohibitifs. L'idée d'un algorithme libre, gratuit et performant germait dans l'esprit de ce cryptographe américain qui souhaitait démocratiser l'accès au chiffrement fort. Blowfish naissait de cette vision : proposer une alternative crédible aux standards établis, sans les entraves juridiques ni financières.

Les spécifications techniques de Blowfish tranchaient avec les pratiques de l'époque. Contrairement à DES et sa clé figée, ce nouvel algorithme

acceptait des clés de longueur variable, s'étendant de 32 à 448 bits. Cette flexibilité répondait aux besoins diversifiés des utilisateurs, qu'ils soient particuliers cherchant une protection basique ou gouvernements exigeant une sécurité maximale. Le cœur de l'algorithme reposait sur un réseau de Feistel à 16 tours, architecture éprouvée mais rehaussée par une innovation remarquable : les S-boxes dépendantes de la clé.

Ces boîtes de substitution constituaient la signature de Blowfish. Plutôt que d'utiliser des tables fixes comme ses prédécesseurs, l'algorithme générait ses propres S-boxes à partir de la clé fournie. L'initialisation démarrait avec les décimales de π, garantissant l'absence de trappes secrètes, puis les modifiait selon un processus complexe impliquant la clé utilisée. Cette approche rendait chaque implémentation unique tout en conservant les propriétés cryptographiques recherchées.

La présentation officielle eut lieu en 1994 lors du workshop Fast Software Encryption à Cambridge. L'accueil fut mitigé : si l'innovation technique impressionnait, la communauté cryptographique restait prudente face à ce nouveau venu. Les premiers tests de sécurité se succédèrent. Serge Vaudenay identifia des classes de clés faibles dans les versions réduites à moins de 14 tours, mais ces vulnérabilités n'affectaient pas la version standard. Vincent Rijmen développa une attaque différentielle du second ordre contre une variante à 4 tours, démonstration technique brillante qui ne remettait nullement en cause la robustesse de l'algorithme complet.

L'adoption commerciale de Blowfish dépassa les espérances de son créateur. Access Manager de Citi-Software Ltd l'intégra dans son gestionnaire de mots de passe, exploitant sa rapidité d'exécution pour sécuriser les données sensibles. Le traitement de texte AEdit en fit son moteur de chiffrement de documents, tandis que Blowfish Advanced CS de Markus Hahn l'utilisait pour l'effacement sécurisé de fichiers. Cette multiplication des implémentations témoignait de la confiance accordée par l'industrie à cet algorithme libre.

Les performances constituaient l'un des atouts majeurs de Blowfish. Sur les processeurs 32 bits, il ne nécessitait que 18 cycles d'horloge par octet chiffré, performance remarquable face aux 45 cycles de DES ou aux 50 cycles d'IDEA. Cette efficacité provenait du choix judicieux d'opérations simples : XOR, addition sur des mots de 32 bits, accès à des tables pré-calculées. Les tableaux de sous-clés tenaient dans le cache des processeurs comme le 80486 ou le 68040, optimisant ainsi les accès mémoire.

Cependant, cette rapidité d'exécution s'accompagnait d'un revers : l'initialisation. La génération des sous-clés requérait l'équivalent de 521 itérations de chiffrement, processus long qui pénalisait les applications nécessitant des changements fréquents de clé. Cette caractéristique orientait na-

turellement Blowfish vers les usages où la clé restait stable : chiffrement de fichiers, communications sécurisées établies, stockage de données.

La sécurité théorique de Blowfish impressionnait par sa solidité mathématique. Schneier avait calculé qu'une clé de 197 bits résisterait même si l'on convertissait toute l'énergie produite par le Soleil en opérations de calcul. Ces projections, bien qu'hypothétiques, illustraient la marge de sécurité considérable offerte par l'algorithme. Une clé de 128 bits nécessitait des milliards d'ordinateurs puissants pendant des millions d'années pour être brisée par force brute.

Les S-boxes dépendantes de la clé renforçaient cette sécurité en compliquant la cryptanalyse différentielle. Chaque clé générait ses propres tables de substitution, rendant difficile l'établissement de patterns exploitables. Le réseau de Feistel assurait une diffusion optimale : après quelques tours, chaque bit de sortie dépendait de l'ensemble des bits d'entrée, propriété centrale pour résister aux attaques statistiques.

Malgré ses qualités indéniables, Blowfish révélait certaines limites avec le temps. Sa taille de bloc de 64 bits, standard dans les années 1990, paraissait insuffisante face aux exigences croissantes de sécurité qui privilégiaient désormais les blocs de 128 bits. L'empreinte mémoire de l'algorithme, avec ses multiples tables, posait problème sur les systèmes embarqués aux ressources limitées comme les cartes à puce.

Ces contraintes n'empêchèrent pas Blowfish de s'imposer durablement. Trente ans après sa création, l'algorithme équipe encore de nombreux systèmes, témoignage de sa conception solide et équilibrée. Son successeur Twofish, finaliste du concours AES en 1998, reprit plusieurs de ses innovations tout en corrigeant les limitations identifiées. Mais Blowfish conservait ses adeptes, séduits par sa simplicité d'intégration et ses performances éprouvées.

L'impact de Blowfish dépassa largement le domaine technique. Il démontra qu'un algorithme cryptographique libre pouvait rivaliser avec les solutions commerciales les plus sophistiquées. Cette réussite inspira des développeurs et des chercheurs, contribuant à l'essor du mouvement de la cryptographie libre. OpenSSL et d'autres projets comme GnuPG s'appuyèrent sur cette démonstration pour légitimer leur approche collaborative.

L'analyse publique dont bénéficia Blowfish valida également le principe de Kerckhoffs, selon lequel la sécurité d'un système cryptographique ne doit reposer que sur le secret de la clé, non sur celui de l'algorithme. Cette transparence, loin d'affaiblir la sécurité, la renforçait en soumettant l'algorithme au regard critique de la communauté scientifique internationale.

Blowfish demeure une référence pédagogique incontournable pour com-

prendre les mécanismes de la cryptographie symétrique moderne. Son histoire illustre la transition du chiffrement matériel vers le chiffrement logiciel, démocratisation des outils cryptographiques, importance croissante des performances sur les architectures grand public. Cette synthèse réussie entre sécurité théorique et efficacité pratique en fit un modèle pour de nombreux algorithmes ultérieurs.

JPEG

Dans les années 1980, une situation paradoxale caractérisait le monde numérique : les images proliféraient mais aucun standard n'existait pour les compresser et les échanger. Les télécommunications faisaient leurs premiers pas vers le multimédia tandis que l'informatique découvrait les joies de la couleur. Face à ce vide normatif, une initiative internationale voit le jour en 1986 : le Joint Photographic Experts Group, plus connu sous l'acronyme JPEG.

Cette collaboration réunit des chercheurs venus de l'ISO et du CCITT (Comité consultatif international télégraphique et téléphonique, l'ancêtre de l'UIT-T). Leur mission semble claire sur le papier : inventer une méthode de compression universelle. Dans les faits, le défi se révèle autrement plus complexe. Comment concilier qualité visuelle et économie d'espace de stockage ? Comment créer un format qui fonctionne aussi bien pour la télécopie couleur que pour les futures applications qu'on ne soupçonne pas encore ?

Les travaux démarrent par une phase d'exploration tous azimuts. En juin 1987, douze techniques de compression différentes se disputent la vedette lors de tests comparatifs. L'ambiance doit être électrique : chaque équipe défend sa méthode avec la conviction que la sienne changera le monde. Après cette première confrontation, trois approches sortent du lot et méritent un approfondissement.

Les trois finalistes subissent une nouvelle batterie d'évaluations en janvier 1998 à Copenhague. C'est là qu'émerge la technique qui dominera les décennies suivantes : la Transformée en Cosinus Discrète, plus connue sous le sigle DCT. Cette approche mathématique, développée dans les années 1970, trouve enfin son terrain d'application idéal.

Le principe du JPEG ressemble à une recette de cuisine sophistiquée. L'image se découpe d'abord en petits carrés de 8×8 pixels, comme si on la taillait en mosaïque. Chaque morceau subit la fameuse transformation

DCT qui convertit les pixels en coefficients de fréquences spatiales. Cette étape révèle l'ingéniosité du procédé : plutôt que de stocker chaque pixel individuellement, on encode les variations et les répétitions dans l'image.

Vient ensuite la quantification, l'étape la plus délicate du processus. Ici, l'algorithme fait des choix : il élimine les informations que l'œil humain perçoit mal ou pas du tout. Cette sélection impitoyable constitue le cœur de la compression avec perte. L'encodage de Huffman complète la chaîne en compactant les données résultantes selon leur fréquence d'apparition.

Les résultats dépassent les espérances. Des taux de compression de 10 à 1, voire 20 à 1, sont monnaie courante sans altération visible de la qualité. Pour la première fois, on peut stocker des centaines d'images sur l'équivalent numérique de quelques disquettes.

La normalisation s'étire sur des années, révélant la complexité de l'exercice. Le comité JPEG opte pour une architecture modulaire qui autorise différents modes de fonctionnement. Le mode « baseline » constitue le socle minimal : tout décodeur digne de ce nom doit le supporter. Cette approche pragmatique évite l'écueil des standards trop rigides ou trop permissifs.

Les premiers brouillons circulent dès 1990 dans les laboratoires et les entreprises. L'UIT-T valide la spécification en 1992 sous la référence T.81, suivie par l'ISO en 1994 avec la norme ISO/IEC 10918-1. Ces dates officielles masquent une réalité plus nuancée : le format s'impose dans les faits bien avant sa validation administrative.

Le comité prend une décision qui se révélera déterminante pour l'avenir : les composants du mode baseline restent libres de droits. Seules certaines fonctionnalités optionnelles peuvent faire l'objet de licences RAND. Cette politique tranche avec l'époque où les brevets constituaient souvent des barrières à l'adoption.

L'Independent JPEG Group publie en 1991 une implémentation complète sous licence libre. Ce code source, distribué selon les termes d'une licence proche de BSD, devient la référence technique. Les développeurs du monde entier peuvent enfin intégrer le JPEG dans leurs applications sans négociation de licences complexes.

L'IJG ne s'arrête pas là. Ses développeurs affinent régulièrement leur code, corrigent les bugs, ajoutent les fonctionnalités avancées de la norme. Cette démarche collaborative annonce l'esprit open source qui triomphera quelques années plus tard.

Les premières applications restent fidèles aux objectifs initiaux : télécopie et vidéotex exploitent les capacités du nouveau format. Mais c'est l'explosion du Web au milieu des années 1990 qui propulse le JPEG vers un succès planétaire. Les pages web s'enrichissent d'images photographiques sans pour autant devenir impraticables sur les modems 56K.

L'arrivée des appareils photo numériques amplifie le phénomène. Kodak, Canon, Nikon adoptent massivement le format pour leurs boîtiers. Les cartes mémoires, chères et limitées en capacité, bénéficient directement de la compression JPEG. Un appareil photo de 2000 stocke plusieurs centaines d'images sur une carte CompactFlash de 64 Mo.

À cette époque, le nombre d'images JPEG en circulation se compte déjà en milliards. Chaque jour apporte son lot de nouvelles photographies numériques. Le format est invisible tant il s'impose naturellement. Les utilisateurs manipulent des fichiers .jpg sans connaître l'existence du comité qui les a rendus possibles.

Pourtant, des limites apparaissent progressivement. À fort taux de compression, des artefacts caractéristiques se manifestent : effet de blocs, oscillations autour des contours nets, dégradation des zones uniformes. Ces défauts gênent quand on pousse la compression à l'extrême, alors qu'ils sont imperceptibles dans les usages modérés.

Les images HDR révèlent d'autres insuffisances du format original. Le JPEG traditionnel est confiné à 8 bits par canal de couleur, un plafond qui paraît dérisoire face aux capteurs modernes capables de 12 ou 14 bits.

Ces limitations motivent le développement de JPEG 2000, une refonte complète basée sur les ondelettes plutôt que sur la DCT. Cette nouvelle mouture promet une meilleure qualité à taux de compression égal et supporte nativement les images HDR. Elle intègre aussi des fonctionnalités avancées comme la montée en résolution progressive.

Paradoxalement, JPEG 2000 ne parvient jamais à détrôner son prédécesseur. L'inertie technologique joue son rôle : pourquoi changer un format qui fonctionne ? Les gains de qualité, réels mais subtils, ne justifient pas le coût de migration pour la plupart des applications.

Le JPEG classique domine toujours la photographie numérique. Les smartphones génèrent chaque année des milliers de milliards d'images au format .jpg. Instagram, Facebook, Twitter s'appuient massivement sur cette technologie vieille de plus de trente ans.

Ce succès durable s'explique par des facteurs favorables. La compression efficace en est l'argument principal, mais d'autres éléments comptent tout autant. L'architecture flexible du standard a permis son adaptation à des usages très variés. La politique ouverte en matière de brevets a facilité l'adoption industrielle. Le caractère libre de l'IJG a démocratisé l'accès à la technologie.

Trente-cinq ans après sa création, le comité JPEG continue ses travaux. L'approche modulaire, les tests comparatifs rigoureux, la gestion équilibrée de la propriété intellectuelle constituent autant de leçons pour les normalisateurs d'aujourd'hui. Il explore maintenant de nouvelles pistes comme

l'intelligence artificielle appliquée à la compression. Mais son premier-né reste son plus bel héritage : un format devenu tellement universel qu'on ne le remarque plus.

IBM ThinkPad

Dans les couloirs d'IBM des années 1920, un petit carnet noir accompagnait chaque employé. Sur sa couverture, un mot simple : « THINK ». Ce bloc-notes de poche incarnait la philosophie de l'entreprise et son exhortation permanente à la réflexion. Soixante-dix ans plus tard, cet esprit donnerait son nom à l'une des gammes d'ordinateurs portables les plus influentes de l'histoire informatique.

L'éveil d'IBM aux réalités du marché portable remonte à 1980. Une équipe d'analystes internes livre alors une prédiction audacieuse : les ordinateurs portables surpasseront les machines de bureau en volume de ventes avant 1996. Cette révélation pousse John Akers, PDG de l'époque, à restructurer l'organisation. En 1992, il crée la Personal Computer Company, une division indépendante confiée à Robert J. Corrigan pour rattraper le retard accumulé face aux constructeurs japonais et californiens.

Les premiers pas se révèlent laborieux, mais le ThinkPad 700T est dévoilé en avril 1992. Muni d'un processeur Intel 386SX/20 et d'un écran monochrome STN de 10 pouces, il peine à convaincre. Ses 4 ou 8 Mo de mémoire et ses deux unités de stockage de 10 Mo semblent dérisoires face aux attentes du marché. Mais cette expérience forge les bases d'un projet plus ambitieux.

Au laboratoire Yamato d'IBM, près de Tokyo, Arimasa Naitoh orchestre une équipe passionnée. Cet ingénieur japonais jouit d'une réputation exceptionnelle pour dénouer les problèmes techniques les plus épineux. Aux côtés de Ken Yonemochi et Koichi Higuchi, il exploite l'art nippon de la miniaturisation pour repenser chaque composant. Pendant ce temps, Richard Sapper, designer allemand consultant pour IBM depuis 1980, prend une décision radicale. Exit le beige omniprésent dans l'informatique : le futur ThinkPad sera noir.

Cette couleur inhabituelle traduit une ambition esthétique qui dépasse les conventions. Sapper s'inspire de l'élégance japonaise et de la fonctionnalité allemande pour dessiner un ordinateur différent. Le ThinkPad 700C bouleverse les codes établis.

L'écran TFT couleur de 10,4 pouces constitue sa première avancée ma-

jeure. Quand la concurrence se contente encore d'affichages monochromes, IBM parie sur la couleur. Cette prise de risque technologique s'accompagne d'une innovation plus polémique : le TrackPoint. Ted Selker, scientifique chez IBM, conçoit ce petit bouton rouge situé au cœur du clavier. Sensible à la pression, il remplace la souris traditionnelle. L'idée divise. Certains utilisateurs s'en entichent immédiatement, d'autres le détestent. Mais cette originalité forge l'identité ThinkPad.

Sous le capot, les spécifications impressionnent. Le processeur IBM 486 SLC cadencé à 25 MHz, la mémoire extensible jusqu'à 16 Mo et le disque dur de 120 Mo placent la machine au sommet des performances. L'autonomie de 3,8 heures défie les standards. Commercialisé à 4 350 dollars, le 700C affiche un prix inférieur de 15% à celui du Toshiba 4400, son rival direct. Cette agressivité tarifaire porte ses fruits : 100 000 commandes affluent dans les deux premiers mois.

Le succès encourage IBM à multiplier les expérimentations. Le ThinkPad 500 de 1993 explore l'ultra-portabilité avec son écran de 7,24 pouces. Au Japon exclusivement, le modèle 550BJ intègre une imprimante Canon, anticipant les besoins de mobilité totale. Le 750P ose les jalons du tactile avec son écran sensible à la pression, concept que perfectionne le 360P grâce à un mécanisme pivotant. Le 755CD marque une autre étape en incorporant un lecteur de CD-ROM, équipement qui demeurera standard pendant deux décennies.

Mais c'est le ThinkPad 701C qui cristallise le génie d'IBM. John Karidis, ingénieur chez Big Blue, imagine l'impensable : un clavier qui se déploie à l'ouverture de l'écran. Deux parties mécaniques s'articulent pour former un clavier complet dans un châssis compact. Cette prouesse vaut au 701C, surnommé « Butterfly », une place dans les collections permanentes du Museum of Modern Art de New York. Paradoxe cruel : malgré cette consécration artistique, le modèle disparaît vite du catalogue, victime de sa complexité mécanique.

Les années 1990 voient la gamme se structurer. La série T s'impose comme référence professionnelle, alliant robustesse et performances. La ligne X cible les nomades exigeants, privilégiant la légèreté sans sacrifier la fiabilité. Les modèles A répondent aux besoins de puissance brute des stations de travail mobiles. Cette segmentation claire transforme le ThinkPad en véritable phénomène culturel. Posséder un ThinkPad noir est un marqueur social dans les entreprises. Dès la première année, les ventes dépassent le milliard de dollars.

L'ironie frappe en 2004. IBM, inventeur du PC et créateur du ThinkPad, enregistre une perte d'un milliard de dollars dans sa division informatique personnelle. Les marchés évoluent trop vite, la concurrence asiatique presse.

En mai 2005, Lenovo rachète l'activité PC d'IBM pour 1,75 milliard de dollars. Le ThinkPad change de propriétaire mais conserve son âme.

Le T60, premier ThinkPad estampillé Lenovo, rassure les inquiets. Son identité visuelle perdure, le TrackPoint résiste, la robustesse demeure. Les processeurs double cœur Intel font leur apparition, la cage en alliage de magnésium renforce la structure. Lenovo comprend qu'elle a hérité d'un patrimoine précieux.

Les transformations s'amorcent néanmoins. Le passage aux écrans 16 :10 avec le T400 en 2008 suscite des débats passionnés sur les forums spécialisés. L'adoption du clavier *chiclet* sur le T430 en 2011 divise la communauté d'utilisateurs. Mais c'est la suppression des boutons physiques du pavé tactile sur le T440 qui provoque la levée de boucliers la plus virulente. Lenovo recule partiellement sur le T450, preuve que l'héritage ThinkPad ne se manipule pas impunément.

La série X1 Carbon, lancée en 2012, incarne la modernité assumée. Son châssis en fibre de carbone repousse les limites de la légèreté sans compromettre la solidité légendaire de la marque. Ces innovations techniques s'accompagnent d'une fidélité aux codes esthétiques originels : le noir mat, le TrackPoint rouge, la sobriété des lignes.

Plus de 150 millions de ThinkPad circulent dans le monde. Cette longévité exceptionnelle traverse les modes. Peter Hortensius, cadre ayant navigué entre IBM et Lenovo, livre une explication éclairante : le ThinkPad incarne « moins un objet qu'un système de valeurs ». Fiabilité, sobriété, efficacité : ces principes traversent les décennies et résistent aux sirènes du marketing.

OpenGL

Dans l'informatique graphique des années 1990, chaque constructeur avait sa propre interface de programmation, ses propres règles, sa propre vision du rendu 3D. Silicon Graphics utilisait IrisGL, d'autres avaient leurs solutions maison. Les développeurs qui voulaient faire tourner leur application sur plusieurs plateformes devaient réécrire leur code autant de fois qu'il y avait de systèmes différents.

C'est dans ce chaos que Kurt Akeley et Mark Segal, tous deux chez SGI, ont eu une idée qui paraissait presque utopique : créer un standard ouvert et universel pour la programmation graphique 3D. Ils ne se contentaient pas de rêver, ils sont passés à l'acte. En juin 1992 naissait OpenGL, la première

spécification vraiment multiplateforme pour la 3D.

Pour éviter qu'OpenGL devienne un nouveau produit propriétaire de SGI, l'entreprise a immédiatement constitué un conseil de surveillance : l'Architecture Review Board. Digital Equipment, Evans & Sutherland, Intel, IBM, Hewlett Packard, Intergraph, Microsoft, Silicon Graphics et Sun Microsystems se sont tous assis autour d'une table. Imaginez la scène : des concurrents directs qui acceptent de travailler ensemble sur un projet commun.

Le pari d'OpenGL tenait en quelques lignes directrices simples. D'abord, standardiser l'accès au matériel graphique sans se mêler de ce qui ne le regardait pas. La gestion des fenêtres ? Laissée au système d'exploitation. L'interface utilisateur ? Pas notre problème. Cette approche minimaliste a fait sa force.

Les créateurs ont fait des choix tranchés. Ils ont refusé d'inclure des fonctions qui ne pouvaient pas être accélérées par le matériel. Tout ce qui relevait de la commodité ou de la gestion de données était renvoyé vers des bibliothèques de niveau supérieur. GLU, la bibliothèque utilitaire qui accompagnait OpenGL, prenait en charge les matrices et les surfaces NURBS que les développeurs réclamaient.

En janvier 1996, OpenGL 1.1 franchit une étape importante. Les textures étendues, les opérations logiques en mode RGB et les tableaux de vertices changent la donne. Ces améliorations répondent aux demandes pressantes d'une industrie qui commence à comprendre le potentiel de la 3D temps réel.

L'adoption ne s'est pas faite du jour au lendemain. D'abord les applications scientifiques, suivies par l'industrie. Les logiciels de CAO ont compris l'intérêt d'OpenGL : culling de faces avant, lignes pointillées, tampons de stencil, toutes ces fonctionnalités étaient là, prêtes à l'emploi.

Le monde du jeu vidéo restait sceptique jusqu'à ce que John Carmack d'id Software prenne une décision qui allait tout changer. En 1996, il décide d'utiliser OpenGL pour une nouvelle version de Quake, alors que les PC équipés d'accélération 3D abordable se comptaient sur les doigts de la main. Pourtant, cette version de Quake a démontré qu'un standard ouvert pouvait rivaliser avec n'importe quelle solution propriétaire. Les constructeurs de cartes graphiques ont pris note.

OpenGL a grandi avec son époque. Quand les GPU sont devenus programmables, la version 2.0 de 2004 a introduit GLSL, le langage de shaders qui donnait aux développeurs un contrôle direct sur le traitement des vertices et des pixels. Cette évolution majeure reflétait l'émergence d'une nouvelle génération de processeurs graphiques.

L'explosion des appareils mobiles a donné naissance à OpenGL ES.

Cette version allégée, taillée sur mesure pour les systèmes embarqués, est devenue l'âme graphique des smartphones et tablettes. Son succès a renforcé l'écosystème OpenGL dans son ensemble, créant un cercle vertueux entre desktop et mobile.

En 2006, le groupe Khronos reprend les rênes d'OpenGL. Cette organisation à but non lucratif garantit la continuité du standard tout en l'adaptant aux nouvelles technologies. Le changement de gouvernance marque une nouvelle étape dans la maturité d'OpenGL.

Trente ans après sa naissance, les domaines d'application d'OpenGL donnent le vertige. L'imagerie médicale s'appuie dessus pour visualiser des scanners en 3D. Les simulateurs de vol l'utilisent pour recréer des environnements réalistes. Hollywood s'en sert pour ses effets spéciaux au cinéma et à la télévision. Adobe l'a intégré dans After Effects, Premiere Pro et Photoshop. Les grandes chaînes de télévision comme CBS, NBC, CNN et BBC ont fait appel à OpenGL pour leurs couvertures d'élections.

Le mécanisme d'extensions d'OpenGL explique en partie cette longévité exceptionnelle. Les fabricants peuvent ajouter de nouvelles fonctionnalités sans casser l'existant. Ces extensions, une fois adoptées massivement, finissent par rejoindre le standard officiel. Cette souplesse a permis à OpenGL d'évoluer sans jamais perdre sa compatibilité descendante.

OpenGL reste la seule API graphique vraiment universelle. Windows, macOS, Linux, systèmes embarqués : partout où il y a un écran et un processeur graphique, OpenGL peut s'installer. WebGL apporte ses capacités aux navigateurs web, Vulkan prend le relais pour les applications haute performance. L'héritage continue.

IBM Simon

Quand IBM dévoile en 1992 le Simon Personal Communicator, personne ne mesure vraiment la portée de cette invention. L'idée paraît presque saugrenue : fusionner un téléphone portable avec un assistant personnel numérique dans le même boîtier. Les ingénieurs d'IBM imaginent un terminal mobile qui ferait tout à la fois : téléphoner, organiser ses rendez-vous, envoyer des messages électroniques. Une vision qui dépasse ce que le marché attend.

BellSouth Cellular lance la commercialisation en 1994. Prix affiché : 900 dollars, une somme rondelette qui équivaut à environ 1 960 dollars actuels. L'appareil impressionne par ses dimensions : 20,3 sur 6,4 sur 3,8 centi-

mètres. Certains le comparent à un *talkie-walkie* militaire tant il paraît massif. Sous ce châssis noir se cache pourtant une technologie d'avant-garde : processeur compatible x86, modem fax intégré, slot pour carte PCMCIA Type II et surtout un écran LCD tactile de 11,4 sur 3,8 centimètres.

Car c'est bien là la révolution du Simon : son écran tactile. Fini le clavier physique traditionnel, l'utilisateur interagit directement avec ses doigts ou un stylet. Cette approche bouleverse les codes établis. L'interface graphique présente onze applications préinstallées qui couvrent les besoins quotidiens. Agenda, carnet d'adresses, calculatrice, bloc-notes, messagerie électronique, fax, téléphonie bien sûr. Le Simon intègre un navigateur web, fonctionnalité quasi-inexistante en 1994 sur les terminaux mobiles.

Le système d'exploitation, développé sur mesure, propose une interface intuitive avec des icônes claires. Passer d'une fonction à l'autre se fait d'une simple pression sur l'écran. On peut griffonner des notes à la main, dessiner des croquis, rédiger et expédier des courriers électroniques. La mémoire embarquée stocke toutes ces données tandis que l'emplacement PCMCIA ouvre des perspectives d'extension intéressantes.

Hélas, le succès ne suit pas. À peine 2 000 exemplaires sortent des chaînes de production. La plupart reviennent chez BellSouth et finissent détruits. Les raisons de cet échec commercial ? Le prix d'abord, prohibitif pour beaucoup. L'autonomie ensuite, décevante. Le poids aussi, rédhibitoire pour un usage quotidien. L'ergonomie enfin, perfectible malgré les innovations.

Mais le vrai problème vient d'ailleurs. En 1994, les réseaux ne suivent pas. L'infrastructure mobile reste balbutiante, les débits dérisoires, la couverture inégale. Comment exploiter pleinement un navigateur web quand la connexion peine à afficher une page de texte ? Les utilisateurs cherchent avant tout la mobilité et la simplicité. Un téléphone qui téléphone, point final. Le Simon arrive dix ans trop tôt.

Cette expérience malheureuse cache pourtant une réussite conceptuelle remarquable. IBM anticipe des usages qui ne se généraliseront qu'une quinzaine d'années plus tard. L'iPhone d'Apple en 2007 reprend nombre d'idées du Simon : écran tactile, interface graphique intuitive, convergence des fonctions. Les premiers smartphones Android s'inscrivent aussi dans cette lignée.

L'histoire du Simon rappelle une vérité de l'innovation technologique : la technique ne suffit pas. Un produit peut être techniquement parfait et commercialement désastreux si l'écosystème n'est pas mûr. Les réseaux, les usages, les mentalités doivent évoluer de concert. IBM l'apprend à ses dépens mais cette leçon enrichit la compréhension des marchés mobiles.

Trente ans après, le Simon fascine encore les passionnés d'informatique. Premier terminal à marier communication et organisation personnelle, il

pose les jalons conceptuels des smartphones actuels. Son échec commercial n'efface pas sa valeur historique. Il témoigne d'une époque où les constructeurs osaient des paris technologiques audacieux sans besoin précis, quitte à essuyer des revers cuisants.

IBM avait imaginé l'informatique de poche avant l'heure. Les ingénieurs rêvaient d'un appareil unique capable de gérer tous les besoins numériques de son propriétaire. Le Simon n'a pas conquis les consommateurs de son époque, mais cette vision prématurée de 1994 domine aujourd'hui le marché mobile.

Adobe PDF

En 1990, John Warnock fixait un objectif ambitieux à Adobe Systems : créer un format de fichier qui préserverait la mise en forme exacte des documents, peu importe où ils seraient consultés. Ce projet interne, baptisé « The Camelot Project », naissait d'une frustration quotidienne : échanger des documents numériques relevait du parcours du combattant. Les polices disparaissaient, les mises en page se désorganisaient, les systèmes refusaient de coopérer.

Warnock et son équipe disposaient d'un atout de taille : PostScript, le langage qu'Adobe avait développé dans les années 1980. Mais transformer ce langage de programmation en format de fichier pratique demandait une approche différente. Trois ans plus tard, en 1993, PDF voyait le jour sous une forme binaire structurée, abandonnant la complexité de PostScript au profit de performances accrues pour l'affichage interactif.

Les débuts furent modestes. Adobe lança deux produits complémentaires : Acrobat pour créer les fichiers PDF, et Acrobat Reader distribué gratuitement pour les lire. Cette stratégie double – payant d'un côté, gratuit de l'autre – finit par porter ses fruits, mais l'adoption resta progressive pendant les premières années.

Internet changea la donne. Vers le milieu des années 1990, le Web explosait et PDF trouvait sa place naturelle : un document pouvait désormais traverser le réseau en gardant son apparence originale. Les entreprises, les administrations, puis les particuliers adoptèrent le format pour leurs communications numériques. PDF répondait à une promesse simple : ce que vous voyez est ce que votre correspondant verra.

Adobe ne se contenta pas de ce succès initial. Les versions successives enrichirent le format de fonctionnalités qui le transformèrent d'un simple

conteneur de documents statiques en véritable plateforme interactive. Les annotations permirent d'annoter les textes, les liens hypertextes de naviguer entre les sections, les formulaires de collecter des données, les signatures numériques d'authentifier les documents. Le multimédia fit son apparition, intégrant vidéos et sons dans les fichiers.

La technique suivait cette évolution fonctionnelle, comme ça devrait l'être à chaque fois. PDF développa une approche sophistiquée de la compression, adaptant ses méthodes au contenu : JPEG pour les photographies, CCITT Group 4 pour les images en noir et blanc, LZW pour les textes et graphiques. Cette flexibilité technique maintenait la qualité tout en contrôlant la taille des fichiers, équilibre délicat mais indispensable.

La gestion des polices illustrait bien l'ingéniosité du format. Plutôt que de subir les caprices des systèmes d'exploitation et leurs collections de fontes variables, PDF proposait deux solutions : incorporer directement les polices dans le document ou utiliser des mécanismes de substitution intelligents. Le document conservait ainsi son apparence sur un ordinateur dépourvu des polices d'origine.

L'année 2008 marqua un tournant. Adobe confia la spécification PDF à l'Organisation internationale de normalisation, transformant son format propriétaire en standard ouvert ISO 32000-1 :2008. Cette décision stratégique confirmait la maturité du format et ouvrait la voie à une adoption encore plus large.

Des versions spécialisées émergèrent pour des usages particuliers. Le format PDF/X conquit l'industrie graphique et l'impression professionnelle, PDF/A répondit aux besoins d'archivage à long terme, PDF/E s'adapta aux documents d'ingénierie, PDF/UA intégra les exigences d'accessibilité. Chaque déclinaison témoignait de la capacité du format à se plier aux contraintes spécifiques de différents secteurs.

PDF 2.0 arriva en 2017 avec la norme ISO 32000-2, apportant des améliorations en matière de signatures numériques, de métadonnées et de support multimédia. Cette version renforçait la sécurité et améliorait l'interopérabilité avec les technologies contemporaines, preuve que le format continuait d'évoluer.

Pendant ces décennies d'évolution, l'écosystème PDF s'enrichit bien au-delà des solutions Adobe. Des bibliothèques open source, des services en ligne, des applications tierces proposèrent leurs propres approches pour créer, modifier ou consulter les fichiers PDF. Cette diversification consolida le statut de standard universel du format.

Le PDF intégra progressivement Unicode pour le texte international, les métadonnées XMP pour décrire les documents, les mécanismes de protection des droits numériques pour sécuriser les contenus. Il s'adapta aux

écrans mobiles et aux nouveaux modes de consultation, prouvant sa capacité d'adaptation aux transformations technologiques.

PDF s'impose dans pratiquement tous les échanges numériques professionnels et personnels. Son succès tient à une promesse : garantir que les documents conservent leur apparence exacte, où qu'ils soient consultés. Cette fidélité visuelle, associée à une compatibilité universelle et à une capacité d'évolution constante, a fait du PDF bien plus qu'un format de fichier, c'est un véritable langage commun de la communication numérique.

Common Gateway Interface

Le Web des premières années ressemblait à une immense bibliothèque figée. Consulter une page revenait à feuilleter un livre : on pouvait lire, mais impossible d'interagir. Cette situation changea radicalement quelques années plus tard quand les utilisateurs commencèrent à réclamer davantage. Ils voulaient remplir des formulaires, rechercher des informations, personnaliser leur expérience. Les pages statiques atteignaient déjà leurs limites.

Rob McCool travaillait au National Center for Supercomputing Applications sur le serveur web NCSA HTTPd. En 1993, il imagina une solution élégante pour résoudre ce problème : créer un pont entre le serveur web et des programmes externes. Cette idée donna naissance au Common Gateway Interface, plus connu sous l'acronyme CGI. Le terme *gateway* (passerelle) reflétait parfaitement sa vocation : relier deux mondes jusqu'alors séparés.

Le génie de CGI tenait dans sa simplicité. Quand un visiteur cliquait sur un lien ou envoyait un formulaire, le serveur ne se contentait plus de retourner un fichier existant. Il lançait un programme sur le serveur qui générait une réponse sur mesure, puis transmettait cette réponse au navigateur. Soudain, le Web devenait vivant. Les développeurs pouvaient écrire leurs scripts dans le langage de leur choix, pourvu qu'il respecte quelques règles de base.

Ces règles définissaient un protocole de communication standardisé. Le serveur transmettait les informations de la requête via des variables d'environnement baptisées méta-variables. Le script récupérait ces données, les traitait, et générait une réponse conforme au protocole HTTP. Cette approche universelle fonctionnait avec n'importe quel langage de programmation, de C à Python en passant par Perl.

Perl justement devint le champion de CGI. Sa capacité à manipuler le texte, sa syntaxe permissive et sa disponibilité sur toutes les plateformes

séduisirent les développeurs web. Les premiers scripts traitaient les formulaires HTML, interrogeaient des bases de données et assemblaient des pages personnalisées. L'époque était aux expérimentations.

Ces expérimentations donnèrent naissance à des collections de scripts partagés. Matt Wright, lycéen dans le Colorado, créa en 1995 le « Matt's Script Archive », qui devint une référence incontournable. Son script FormMail permettait d'envoyer le contenu des formulaires par e-mail et fut téléchargé des milliers de fois. Problème : Wright était jeune, inexpérimenté, et ses scripts comportaient des failles de sécurité béantes. La communauté Perl réagit en créant « Not Matt's Scripts », proposant des alternatives plus robustes.

Car la sécurité posait un défi constant. Chaque script CGI représentait une porte d'entrée potentielle vers le système. Les développeurs devaient valider scrupuleusement les données d'entrée, éviter les débordements de mémoire et se prémunir contre l'injection de code malveillant. Les administrateurs confinaient généralement ces scripts dans un répertoire spécial, le fameux cgi-bin, pour limiter les dégâts en cas de problème.

L'architecture de CGI souffrait d'un défaut congénital : chaque requête déclenchait le lancement d'un nouveau processus. Sur un serveur peu fréquenté, cela passait inaperçu. Mais dès que le trafic augmentait, les performances s'effondraient. La machine passait plus de temps à créer et détruire des processus qu'à traiter les requêtes elles-mêmes.

Des solutions émergèrent pour contourner cette limitation. FastCGI gardait les processus en vie entre les requêtes, éliminant le surcoût de lancement. Mod_perl intégrait l'interpréteur Perl directement dans Apache, transformant les scripts en modules persistants. Ces optimisations redonnèrent une seconde jeunesse à CGI.

Pendant ce temps, d'autres approches gagnaient du terrain. PHP, créé en 1994 par Rasmus Lerdorf, proposait une intégration native au serveur web. Plus besoin de processus externes : le code s'exécutait directement dans Apache. Microsoft développait ASP pour ses serveurs IIS, tandis que Sun travaillait sur JSP pour l'univers Java. Chacun cherchait sa voie vers la programmation web dynamique.

Ces nouvelles technologies n'effaçaient pas l'héritage de CGI. Elles en reprenaient les concepts fondamentaux : séparation entre serveur et logique applicative, transmission d'informations via l'environnement, modèle requête-réponse. CGI avait tracé la route, d'autres l'élargissaient.

L'évolution s'accéléra au tournant des années 2000. Les frameworks web modernes apportaient des abstractions de plus haut niveau, une gestion d'état sophistiquée, des architectures orientées services. Ruby on Rails révolutionna le développement web en 2004, Django fit de même pour Python.

Ces outils rendaient CGI obsolète pour la plupart des usages.

Pourtant, CGI refusait de disparaître complètement. Sa standardisation officielle dans la RFC 3875 en 2004 reconnaissait son importance historique. Cette spécification codifiait dix ans d'expérience pratique, détaillant chaque aspect technique de l'interface. Apache et d'autres serveurs continuaient de la supporter, préservant la compatibilité avec d'anciennes applications.

Cette longévité s'explique par la robustesse du modèle conceptuel de CGI. L'idée d'une interface standard entre serveur web et programmes externes reste pertinente aujourd'hui. Les architectures de microservices modernes et d'API REST reprennent ce principe à plus grande échelle, dont la philosophie *stateless*.

FreeBSD

Pour comprendre l'histoire de FreeBSD, il faut remonter aux années 1970, quand l'Université de Californie à Berkeley reçut le code source d'UNIX de Bell Labs. Les chercheurs se mirent alors à développer leurs propres améliorations sur ce système, créant ce qu'on appela BSD, pour Berkeley Software Distribution. Ces travaux universitaires aboutirent à plusieurs versions majeures qui marquèrent l'histoire, la dernière étant la 4.4BSD-Lite.

Cette version 4.4BSD-Lite concentrait une décennie d'innovations techniques. L'interface socket pour les communications réseau y fit son apparition, accompagnée de l'implémentation de référence du protocole TCP/IP. Le système de fichiers rapide améliora les performances, tandis que le support NFS et le modèle de mémoire virtuelle `mmap` posaient des bases qui restent d'actualité. Ces éléments techniques ne relevaient pas du simple exercice académique : ils transformèrent la manière dont les ordinateurs communiquaient et géraient leurs ressources.

Au début des années 1990, la situation était particulière. La communauté informatique cherchait un système d'exploitation libre et surtout fonctionnel. Un groupe de développeurs décida de créer une distribution complète à partir de 4.4BSD-Lite. Mais ils refusèrent le modèle habituel du leader unique. Sept personnes formèrent ce qu'ils nommèrent le *Core Team*, une direction collégiale qui allait définir la culture du projet pour les décennies suivantes.

Le choix technique initial porta sur l'architecture Intel 386. Les développeurs visaient la stabilité et la performance, notamment pour le réseau

et le stockage. Le noyau intégra des mécanismes sophistiqués hérités de la technologie Mach pour la gestion de la mémoire virtuelle. L'interface de programmation resta fidèle aux standards UNIX, facilitant la migration des applications existantes.

La gouvernance évolua en 2000. Les neuf membres du Core Team devinrent élus tous les deux ans par les contributeurs actifs. Ce système démocratique permit un renouvellement régulier et l'émergence de nouvelles idées. Le projet se dota d'outils centralisés de développement : contrôle de versions, suivi des bogues. Ces infrastructures rendirent la collaboration à distance bien plus fluide qu'auparavant.

Le système de ports représenta une innovation majeure. Cette collection de logiciels préparés pour une installation automatisée simplifia radicalement l'ajout d'applications tierces. Les utilisateurs n'eurent plus à se soucier des dépendances ou des problèmes de compilation. Le système pkg vint plus tard améliorer encore la gestion des paquets logiciels, rendant l'expérience utilisateur comparable aux distributions Linux les plus avancées.

La documentation reçut une attention inhabituelle. Les développeurs ont constitué une équipe dédiée à la rédaction et la maintenance des manuels. Ces contributeurs obtinrent les mêmes droits que les programmeurs, une reconnaissance rare qui soulignait l'importance accordée à la documentation. Cette approche porta ses fruits : FreeBSD devint réputé pour la qualité de sa documentation technique.

En 2000, la FreeBSD Foundation apporta un soutien institutionnel au projet. Cette organisation à but non lucratif fournit l'infrastructure technique et finança des développements spécifiques. Elle emploie aujourd'hui une vingtaine de personnes qui travaillent au développement, à la documentation et à la promotion du système.

Le choix de la licence Berkeley fut stratégique. Moins restrictive que la GPL de Linux, elle permit aux entreprises d'intégrer le code sans obligation de publier leurs modifications. Cette flexibilité attira de nombreuses sociétés dans les domaines des systèmes embarqués et des appareils réseau. Apple y trouva d'ailleurs la base de Darwin, le cœur d'iOS et macOS.

Les échanges avec d'autres projets BSD enrichirent le développement. Tandis que NetBSD apporta son expertise multi-architecture et ses méthodes de tests automatisés, OpenBSD contribua significativement à la sécurité avec le programme SSH et les composants de chiffrement pour HTTPS. Cette collaboration entre projets cousins créa un écosystème technique cohérent.

L'infrastructure d'Internet adopta FreeBSD. Sa stabilité ainsi que ses performances réseau en firent le choix privilégié des fournisseurs d'accès dans les années 1990. Yahoo ! et Hotmail l'utilisèrent pour leurs serveurs,

démontrant sa capacité à gérer des charges massives. Les hébergeurs web suivirent, trouvant dans ce système une base fiable pour leurs services.

La culture communautaire joua un rôle déterminant. Les développeurs établirent des règles de communication respectueuses, évitant les conflits qui paralysaient d'autres projets. Cette approche inclusive encouragea la participation de contributeurs du monde entier. La communauté devint diverse et productive, loin des guerres d'ego qui nuisaient à certains projets concurrents.

Trente ans plus tard, FreeBSD reste pertinent. Le système continue d'intégrer les technologies modernes sans sacrifier sa stabilité. Son code se retrouve dans des produits commerciaux variés, des routeurs aux consoles de jeux. Les systèmes de stockage en réseau l'utilisent pour sa robustesse. Cette présence discrète mais omniprésente témoigne de la qualité du travail accompli.

Cette longévité s'explique par des facteurs qui se renforcent mutuellement. La base technique héritée de BSD était solide. L'organisation communautaire s'est révélée efficace. La documentation complète a facilité l'adoption. La licence permissive a séduit les entreprises. Mais surtout, une attention constante à la qualité technique et une gouvernance équilibrée ont permis de traverser les tempêtes du monde informatique sans perdre le cap.

Intel Pentium

En 1993, Intel abandonne sa nomenclature numérique. Le processeur qui devait s'appeler 586 prend finalement le nom de Pentium. Cette rupture n'a rien d'anodin puisqu'elle traduit une stratégie défensive face à AMD, qui avait commercialisé son Am486 en exploitant la similitude des appellations. Intel tente d'enregistrer « 586 » ou « i586 » comme marque déposée, mais essuie un refus. Une simple séquence de chiffres manque de caractère distinctif aux yeux des autorités compétentes.

Le premier Pentium repose sur l'architecture P5. Ce processeur superscalaire fonctionne entre 60 et 66 MHz, dispose d'un cache L1 de 16 Ko et utilise un bus système cadencé sur la même plage de fréquences. Sa fabrication en technologie 800 nanomètres représente une prouesse technique pour l'époque. Deux ans plus tard, le Pentium MMX intègre de nouvelles instructions dédiées au traitement multimédia. Intel développe parallèlement l'architecture P6, commercialisée sous l'appellation Pentium Pro en 1995. Cette version introduit l'exécution dans le désordre et embarque un cache de niveau 2 dans un boîtier multi-puces.

La famille s'enrichit en 1997 avec le Pentium II, qui combine les avancées du Pentium Pro et les instructions MMX. Ce modèle adopte un nouveau format physique, la cartouche SECC (Single Edge Contact Cartridge), qui simplifie l'assemblage et les tests. Le Pentium III arrive en 1999 avec le jeu d'instructions SSE (Streaming SIMD Extensions) et des registres 128 bits autorisant le traitement simultané de quatre nombres à virgule flottante.

Intel bascule vers l'architecture NetBurst avec le Pentium 4 en l'an 2000. Cette nouvelle conception privilégie les hautes fréquences d'horloge grâce à un pipeline allongé. Des versions mobiles voient le jour pour les ordinateurs portables, avec des optimisations sur la consommation énergétique. En 2005, le Pentium D inaugure l'ère multicœur en intégrant deux processeurs Pentium 4 dans un unique boîtier.

Intel modifie sa stratégie en 2006. La marque Pentium se positionne désormais entre les Celeron d'entrée de gamme et la nouvelle série Core haut de gamme. Les Pentium récents utilisent les mêmes puces que les processeurs Core, mais bridés : fréquences réduites, cache L3 partiellement désactivé, technologies avancées retirées. Les processeurs sont moins coûteux tout en conservant la compatibilité avec l'architecture x86.

Les évolutions techniques ont nécessité des adaptations matérielles. Les premiers modèles s'insèrent dans des supports Socket, puis les formats Slot apparaissent pour accueillir les cartouches SECC. L'augmentation du nombre de connexions requises conduit au développement du LGA 775, où les broches se trouvent sur le support plutôt que sur le processeur.

La miniaturisation progresse, la finesse de gravure passe de 800 nanomètres en 1993 à 32 nanomètres pour les versions récentes. Cette évolution autorise l'augmentation du nombre de transistors, améliore les performances et réduit la consommation électrique. En 2011, l'architecture Sandy Bridge intègre des fonctionnalités graphiques directement dans le processeur. Cette intégration répond aux besoins croissants des applications courantes en matière de rendu graphique.

Les processeurs Pentium ont contribué à démocratiser l'informatique personnelle. Leur architecture x86 s'impose comme standard de fait et crée un vaste écosystème logiciel. La rétrocompatibilité maintenue au fil des générations fait conserver leurs applications aux utilisateurs lors du renouvellement de leur matériel, même si nous savons avec le recul que cela a bien une limite. L'influence du Pentium dépasse le cadre des ordinateurs personnels. Ces processeurs servent de base au développement d'autres gammes : Celeron pour l'entrée de gamme, Xeon pour les serveurs et stations de travail, versions embarquées comme l'EP80579 pour les systèmes sur puce.

Les mutations de l'industrie se reflètent dans l'évolution des Pentium. La montée en puissance des applications multimédias motive l'ajout d'instruc-

tions spécialisées. L'importance croissante de l'efficacité énergétique conduit au développement de versions mobiles optimisées. L'émergence du calcul parallèle entraîne l'adoption d'architectures multicœurs.

Le Pentium illustre une forme de longévité remarquable dans l'industrie informatique. Cette marque créée en 1993 existe toujours en 2025, s'adaptant aux mutations technologiques et aux besoins du marché.

Lua

En 1993, trois chercheurs brésiliens du groupe de technologie graphique de l'Université Pontificale Catholique de Rio de Janeiro, Roberto Ierusalimschy, Luiz Henrique de Figueiredo et Waldemar Celes, travaillaient sur des projets informatiques pour Petrobras, la compagnie pétrolière nationale. Ils ne se doutaient pas qu'ils allaient créer l'un des langages de script les plus influents de l'histoire du jeu vidéo.

Entre 1977 et 1992, une politique protectionniste stricte rendait l'importation de logiciels étrangers difficile au Brésil. Les entreprises locales n'avaient d'autre choix que de développer leurs propres outils. C'est ainsi que Tecgraf avait conçu deux langages spécialisés : DEL pour la saisie de données, et SOL pour générer des rapports lithologiques destinés à Petrobras. Ces deux précurseurs contenaient les bases de ce qui allait devenir Lua.

La première version du langage fusionne les capacités de DEL et SOL dans une approche plus générale. Le nom « Lua », qui signifie « lune » en portugais, répond à SOL (« soleil »), créant ainsi une continuité poétique avec le travail antérieur. Les créateurs choisissent d'abord une licence restrictive, limitant l'usage commercial, avant d'adopter progressivement une philosophie plus ouverte à partir de la version 2.1, et la licence MIT en 2002.

La simplicité constitue le cœur de Lua. Le langage s'appuie sur quelques concepts fondamentaux : les tables (des tableaux associatifs), les fonctions et les coroutines. Cette économie de moyens se traduit par une implémentation remarquablement compacte d'environ 17 000 lignes de code C, capable de tourner sur presque toutes les plateformes, du microcontrôleur au supercalculateur. Mais c'est surtout sa capacité d'intégration avec d'autres langages, spécifiquement le C, qui fait sa force. L'interface entre Lua et C, baptisée API C et élément central du langage, transforme Lua en excellent outil d'extension pour les applications existantes.

Les versions successives marquent des étapes techniques significatives. La version 2.1 de 1995 introduit les mécanismes d'extensibilité sémantique, donnant aux programmeurs la possibilité d'adapter le comportement du langage. La version 3.0 de 1997 unifie les fonctions C et Lua en un seul type. La version 4.0 de 2000 refond complètement l'API C pour la rendre plus cohérente et réentrante. En 2003, Lua 5.0 bouleverse l'architecture interne avec l'introduction de la portée lexicale complète, des coroutines et une machine virtuelle basée sur les registres, une innovation remarquable pour un langage de script à cette époque.

L'industrie du jeu vidéo découvre Lua en 1998 quand LucasArts l'utilise pour *Grim Fandango*. Cette adoption pionnière déclenche un mouvement qui ne s'arrêtera plus. *World of Warcraft*, *Angry Birds*, *The Sims* et des centaines d'autres titres intègrent Lua pour gérer leur logique de jeu. Les raisons de ce succès ? Une intégration facile, une légèreté exceptionnelle, une vitesse d'exécution satisfaisante et une courbe d'apprentissage accessible.

Au-delà des jeux, Lua s'impose dans des domaines inattendus. Adobe l'intègre dans Photoshop Lightroom, où plus de 40% du code repose sur ce langage. On le retrouve dans des routeurs réseau, des téléviseurs numériques, des instruments scientifiques et toutes sortes d'applications embarquées. Cette polyvalence découle d'une philosophie de conception particulière : fournir des mécanismes plutôt qu'imposer des politiques. Au lieu de dicter une façon de programmer, Lua offre des outils généraux permettant aux développeurs de construire leurs propres solutions, que ce soit en programmation procédurale, fonctionnelle, orientée objet ou orientée données.

Le développement du langage reflète une vision cohérente maintenue par une équipe stable depuis l'origine. Les créateurs privilégient systématiquement la simplicité et la cohérence plutôt que l'accumulation de fonctionnalités. Chaque nouvelle version fait l'objet de tests rigoureux, les versions alpha affichant déjà une grande stabilité et les versions bêta étant quasiment finales. Cette prudence garantit la fiabilité du langage.

La communauté Lua s'organise principalement autour d'une liste de diffusion créée en 1997. Ce forum technique maintient un niveau de discussion élevé tout en restant ouvert aux débutants. La publication en 2003 du livre *Programming in Lua* contribue largement à la diffusion du langage et à la formation de nouveaux utilisateurs.

L'architecture interne de Lua témoigne de cette recherche de minimalisme. Le compilateur fonctionne en une seule passe sans représentation intermédiaire, traitant ainsi efficacement de gros fichiers de données. Cette caractéristique s'avère précieuse dans le domaine du jeu vidéo, où Lua sert souvent à décrire les ressources du jeu.

L'histoire de Lua démontre qu'un projet né de contraintes locales spé-

cifiques peut devenir un outil universel. Son succès repose sur des principes de conception clairs, une mise en œuvre rigoureuse et une équipe de développement stable qui n'a jamais perdu de vue les qualités originelles du langage : simplicité et efficacité. Ce qui devait être une solution temporaire à un problème brésilien est devenu un standard mondial, prouvant que les meilleures idées naissent parfois des situations les plus contraignantes.

R

Deux statisticiens de l'Université d'Auckland, Ross Ihaka et Robert Gentleman, cherchaient un environnement statistique pour leur laboratoire d'enseignement sur Macintosh. Leur lecture du livre d'Abelson et Sussman, *The Structure and Interpretation of Computer Programs*, et leur intérêt pour le langage Scheme les conduisirent à développer un interpréteur minimaliste en C d'environ mille lignes de code. Ce qui n'était au départ qu'une expérimentation allait devenir l'un des outils les plus utilisés en statistique et analyse de données.

Le choix d'adopter une syntaxe proche du langage S, créé aux Bell Labs par John Chambers et son équipe, s'imposa. Cette décision garantissait une certaine familiarité pour les statisticiens habitués à S, tout en laissant la liberté d'innover sur le plan technique. R conserva des spécificités héritées de Scheme, notamment dans sa gestion de la mémoire avec un ramassage de miettes à allocation fixe qui limitait les problèmes de pagination. Les règles de portée lexicale permettaient aux fonctions d'accéder aux variables définies lors de leur création, une caractéristique qui distinguait R de ses contemporains.

En août 1993, Ihaka et Gentleman déposèrent leurs premières versions binaires sur StatLib et annoncèrent leur travail sur la liste de diffusion s-news. Martin Mächler de l'ETH Zurich, intrigué par le projet, les incita à publier le code source sous licence GNU GPL. Cette suggestion fut d'abord accueillie avec prudence. Les deux chercheurs hésitaient à ouvrir complètement leur code. Ils franchirent pourtant le pas en juin 1995. Cette décision transforma radicalement la nature du projet : de développement fermé entre deux collaborateurs, R devint une initiative collaborative internationale.

La création des listes de diffusion automatisées à l'ETH Zurich en 1996 accéléra les contributions externes. Les rapports d'anomalies, les suggestions et les correctifs affluèrent. Le langage s'enrichit progressivement de nouvelles fonctionnalités. Face à l'ampleur des contributions, un groupe

plus large de développeurs principaux se forma en 1997, établissant une structure organisationnelle qui perdure.

L'année 2000 marqua une étape importante avec la création de la R Foundation for Statistical Computing, organisation à but non lucratif basée à Vienne. Cette fondation, constituée par les membres de l'équipe de développement, s'assigna trois objectifs : soutenir le développement continu de R, fournir un point de référence pour les interactions avec la communauté, et gérer les droits d'auteur du logiciel et de sa documentation.

R s'imposa comme un outil statistique complet, aux capacités étendues de manipulation de données, de calcul et de visualisation graphique. Son architecture modulaire autorisa l'ajout de fonctionnalités via des packages, enrichissant sans cesse ses possibilités d'application. Le système intégra des opérateurs performants pour les calculs matriciels, une collection d'outils d'analyse statistique, et des fonctionnalités graphiques sophistiquées. La documentation adoptait un format proche de LaTeX, garantissant une documentation exhaustive accessible en ligne et en version imprimée. Cette approche, combinée à la disponibilité du code source, donnait aux utilisateurs les moyens de comprendre le comportement des fonctions.

Le modèle de développement de R illustre les avantages du logiciel libre dans le domaine scientifique. La collaboration internationale des développeurs, la revue par les pairs du code source, et les tests approfondis dans diverses situations réelles contribuèrent à la robustesse du logiciel. La taille de la communauté d'utilisateurs, estimée en dizaines voire centaines de milliers de personnes, multipliant les vérifications et améliorations du code, renforça cette solidité.

Le cycle de développement suit un rythme régulier avec des versions majeures publiées annuellement depuis 2013. Chaque version fait l'objet de tests rigoureux, incluant des phases alpha, bêta et release candidate, concernant autant le code source que les versions binaires précompilées pour différentes plateformes. L'équipe maintient un système de gestion de versions basé sur Subversion, avec des branches distinctes pour la version stable et la version de développement. Les corrections d'erreurs sont intégrées dans la branche stable, tandis que les nouvelles fonctionnalités importantes sont développées dans la branche de développement.

L'impact de R sur la recherche statistique et l'analyse de données dépasse le cadre universitaire. Son adoption dans l'industrie, notamment dans les secteurs de la finance, de la recherche pharmaceutique et de l'analyse de données massives, témoigne de sa maturité technique. La disponibilité d'interfaces avec d'autres langages comme C, C++ et Fortran étend ses capacités aux calculs intensifs.

L'héritage de R réside dans sa combinaison d'un langage de programma-

tion véritablement fonctionnel avec un environnement statistique complet. Cette association donne aux statisticiens les moyens de développer et tester de nouvelles méthodes d'analyse tout en offrant aux utilisateurs finaux un outil pratique et extensible. La pérennité du projet repose sur une infrastructure distribuée, incluant un réseau mondial de miroirs CRAN (Comprehensive R Archive Network) qui assure la disponibilité des ressources. Ce réseau distribue le logiciel principal ainsi que des milliers de packages complémentaires développés par la communauté.

DHCP

Les réseaux TCP/IP des années 1980 restaient modestes et leur configuration était entièrement statique. Les administrateurs attribuaient manuellement une adresse IP à chaque machine, qui conservait cette information dans sa mémoire secondaire. Toute modification impliquait une intervention directe sur la console, généralement suivie d'un redémarrage du système. Cette approche artisanale convenait aux infrastructures de l'époque, mais elle allait bientôt montrer ses limites.

La croissance des réseaux et l'arrivée des stations de travail économiques sans mémoire secondaire ont bouleversé cette organisation. Il devenait urgent de centraliser l'administration des liaisons entre adresses IP et matériel informatique. Le protocole RARP (Reverse Address Resolution Protocol) émergea comme première réponse : une machine connectée à un segment réseau pouvait désormais découvrir son adresse IP et initier normalement ses communications TCP/IP. Entre temps, BOOTP (Bootstrap Protocol) facilitait la configuration des stations sans disque en récupérant l'ensemble des paramètres TCP/IP et des données système nécessaires au démarrage. L'introduction des agents relais BOOTP permit de franchir les frontières d'un unique segment réseau. BOOTP intégrait déjà un mécanisme d'extension utilisant le dernier champ de la trame pour des données spécifiques, une idée que DHCP reprendrait.

Le RFC 1531 définit DHCP comme standard en octobre 1993. Cette extension de BOOTP corrigeait deux faiblesses majeures : l'obligation d'intervenir manuellement pour ajouter les informations de configuration de chaque client, et l'impossibilité de réutiliser les adresses IP. Le protocole gagna en popularité, ce qui conduisit à des clarifications successives. En 1997, le RFC 2131 devint la référence pour les réseaux IPv4, statut qu'il conserve encore aujourd'hui.

L'arrivée d'IPv6 imposa le développement de DHCPv6, documenté dans le RFC 3315. Il ne s'agissait pas d'une simple transposition de DHCPv4 aux adresses IPv6, mais d'un protocole substantiellement différent. Le changement le plus notable concernait l'identification des clients : DHCPv4 repose sur l'adresse MAC, tandis que DHCPv6 introduit le DUID (DHCP Unique Identifier). La conception de DHCPv6 abandonne le principe d'une adresse unique par périphérique, autorisant les appareils à solliciter plusieurs adresses. Le RFC 3633 ajouta la délégation de préfixe, une fonctionnalité inédite sans équivalent en DHCPv4. Le RFC 3736 étendit les capacités du protocole pour prendre en charge la configuration des clients utilisant l'auto-configuration sans état des adresses.

L'Internet Systems Consortium (ISC) a joué un rôle déterminant dans le développement de DHCP. L'organisation maintient deux systèmes majeurs : ISC DHCP et Kea. Ted Lemon et Vixie Enterprises écrivirent la première implémentation, ISC DHCP, comme référence pour le nouveau protocole. La version 1.0 sortit en juin 1998, suivie un an plus tard par la version 2.0. La version 3.0, publiée en 2001, intégra le support de la norme de basculement IETF et les mises à jour DDNS asynchrones. La version 4.0, en 2007, apporta le support IPv6.

Une équipe d'ingénierie dédiée travaille sur ISC DHCP depuis 2004. Ted Lemon et Shawn Routhier, anciens employés d'ISC, ont contribué au projet pendant de nombreuses années. Thomas Markwalder assure la maintenance principale depuis 2016, tandis que Francis Dupont participe activement au maintien du logiciel depuis 2007. La communauté a enrichi le projet avec un système de stockage des baux LDAP et un script d'affichage des baux.

Kea représente une implémentation entièrement nouvelle, destinée à remplacer ISC DHCP vieillissant. Conçu initialement dans le cadre applicatif BIND 10 pour supporter plusieurs applications DNS et DHCP, le projet s'est recentré sur DHCP après l'arrêt du développement DNS en 2014. Tomek Mrugalski et Marcin Siodelski ont dirigé le développement initial de Kea, qui se distingue par son interface de gestion REST moderne et son architecture modulaire. Contrairement à ISC DHCP, Kea sépare les démons DHCPv4, DHCPv6 et DNS dynamique, et propose des bibliothèques optionnelles pour étendre les fonctionnalités du serveur DHCP principal.

La communauté DHCP s'est structurée autour d'outils complémentaires. Le logiciel `perfdhcp`, distribué avec Kea, évalue les performances des serveurs DHCP en générant un trafic important depuis plusieurs clients simulés. Il teste les serveurs IPv4 et IPv6, fournissant des statistiques sur les temps de réponse et les requêtes perdues. Le projet Anterius, issu du programme « Google Summer of Code d'ISC », a démontré la possibilité de développer un tableau de bord de gestion léger pour Kea. Plus récemment,

le projet Stork, lancé en 2020, propose une interface web robuste et extensible, s'intégrant avec la base de données temporelle Prometheus et l'outil de visualisation Grafana.

DHCP illustre bien la capacité d'adaptation des protocoles réseau aux besoins changeants de l'informatique. D'un simple outil de configuration automatique, il s'est transformé en un système sophistiqué gérant l'attribution dynamique des adresses dans les réseaux modernes. Son développement continu, porté par une communauté active, lui garantit une pertinence durable face aux défis des infrastructures réseaux contemporaines.

NCSA Mosaic

NCSA Mosaic débarque en 1993 et chamboule durablement la façon dont les gens accèdent aux ressources sur Internet. L'histoire commence un an plus tôt, quand Marc Andreessen, étudiant à l'Université de l'Illinois, traîne plus souvent au National Center for Supercomputing Applications qu'en amphi. Le timing n'a rien d'anodin : les centres de super calcul traversent une mutation majeure. Les ordinateurs Cray, devenus hors de prix face aux microprocesseurs qui gagnent en puissance, laissent place aux réseaux informatiques reliant chercheurs et enseignants.

Internet à cette époque ressemble à un patchwork de protocoles et de services disparates. FTP transfère les fichiers, Gopher organise des menus, WAIS fouille l'information. Tim Berners-Lee vient de créer HTTP et HTML au CERN, mais les outils disponibles ne tirent pas vraiment parti de ces innovations. Quelques navigateurs pionniers comme Erwise ou ViolaWWW apportent des fonctionnalités intéressantes, sans pour autant dépasser le stade de prototypes cantonnés à des plateformes précises.

Andreessen et son collègue Eric Bina développent la version alpha de Mosaic pendant les vacances de Noël. Leur navigateur se démarque d'entrée : il affiche les images directement dans le texte, ce qui change radicalement l'expérience. Intuitive, l'interface graphique permet une navigation par simples clics sur les liens hypertextes. Le décollage est fulgurant. De douze utilisateurs au lancement début 1993, Mosaic en compte des centaines de milliers dès le milieu de l'année.

Cette adoption massive tient à des raisons techniques concrètes. Le navigateur tourne sur UNIX, Windows et Macintosh, digère les différents protocoles Internet et s'installe facilement grâce à un exécutable unique. Les internautes apprécient les signets qui mémorisent leurs pages favorites et

l'historique de navigation. Le chargement des images reste tributaire des débits modestes de cet période, mais la vitesse d'affichage convient.

Le succès pousse le NCSA à accorder des licences d'exploitation du code source. Spyglass devient le distributeur principal et fournit une version à Microsoft, qui servira de départ à Internet Explorer. Andreessen quitte le NCSA fin 1993 pour monter Mosaic Communications avec Jim Clark, société rebaptisée ensuite Netscape. Leur stratégie commerciale détonne : distribuer gratuitement le navigateur aux particuliers et aux établissements d'enseignement, facturer les entreprises.

L'impact sur Internet est spectaculaire. Le trafic web bondit de 1,5% à 23,9% du volume total sur NSFNet en deux ans. Le nombre de serveurs accessibles explose : une cinquantaine en janvier 1993, plus de 1 500 en juin 1994. Cette croissance soulève des questions de gouvernance. La NSF confie alors la gestion des noms de domaine à Network Solutions Inc., créant un monopole commercial qui fera couler beaucoup d'encre.

Le navigateur Mosaic fixe les standards d'interface graphique repris par tous ses successeurs : barre d'adresse, boutons de navigation, affichage intégré des images. Il popularise la navigation hypertexte auprès du grand public et inspire le modèle économique du « freemium » adopté par tant de services en ligne. Les problématiques qu'il révèle en matière de sécurité, avec l'ajout ultérieur du protocole SSL, ou de respect de la vie privée, restent brûlantes d'actualité.

Dans l'histoire de l'informatique, Mosaic marque le moment où Internet passe d'outil académique à média grand public. Sa simplicité d'usage et ses innovations techniques ont rendu possible le Web moderne.

Secure Socket Layer

Quand le World Wide Web s'ouvre au grand public au début 1990, personne ne mesure vraiment les enjeux de sécurité qui vont en découler. Pourtant, dès 1994, Netscape Communications comprend qu'il faut protéger les échanges sur Internet. L'entreprise lance alors SSL (Secure Socket Layer) dans sa version 1.0, un protocole destiné à sécuriser les communications entre navigateurs et serveurs web.

L'idée de départ ne date pas d'hier : en 1978 déjà, Loren Kohnfelder proposait d'utiliser des certificats numériques pour garantir l'authenticité des clés publiques. Cette intuition théorique servira de fondation à SSL,

qui combine cryptographie asymétrique et symétrique pour assurer confidentialité et intégrité des données transmises.

La première version publique, SSL 2.0, sort en 1995. Mais elle traîne avec elle un cortège de failles : l'authentification des messages repose uniquement sur MD5, les clés servant à l'authentification et au chiffrement sont identiques, et la fermeture des connexions TCP expose le protocole à des attaques par troncature. Pire encore, rien ne protège vraiment la négociation initiale contre les attaques de l'homme du milieu.

Netscape réagit vite et publie SSL 3.0 dès 1996. Cette nouvelle mouture améliore sensiblement la génération des clés à partir du secret maître. Le protocole adopte une version préliminaire de HMAC pour authentifier les messages et impose désormais la prise en charge des algorithmes DH/DSS et Triple-DES.

Trois ans plus tard, l'IETF prend les rênes et standardise le protocole sous le nom de TLS (Transport Layer Security) 1.0. La technologie propriétaire de Netscape devient un standard ouvert. TLS 1.0 est compatible avec SSL 3.0 mais renforce la sécurité grâce à l'utilisation complète de HMAC.

Les versions se succèdent ensuite à un rythme régulier : TLS 1.1 en 2006, TLS 1.2 en 2008, puis TLS 1.3 en 2018. Chaque itération abandonne des mécanismes obsolètes et corrige les vulnérabilités découvertes entretemps. SSL 2.0 est officiellement abandonné, SSL 3.0 suit le même chemin. À partir de 2015, TLS 1.0 ne suffit plus pour les systèmes traitant des données bancaires.

Derrière SSL et TLS se cache une infrastructure complexe de clés publiques (PKI). Les autorités de certification (CA) émettent des certificats X.509 aux entités qui opèrent des serveurs web avec des noms DNS spécifiques. Ces certificats sont généralement signés par des CA intermédiaires, créant une chaîne de confiance qui remonte jusqu'à une racine présente dans les magasins de certificats des systèmes d'exploitation ou des navigateurs.

La validation des certificats par les CA a évolué au fil du temps. La validation de domaine (DV) vérifie simplement l'autorité sur un nom de domaine. La validation d'organisation (OV) authentifie l'entité qui demande le certificat. La validation étendue (EV), introduite plus tard, impose aux CA de suivre un protocole strict pour vérifier l'identité du demandeur.

L'adoption de SSL et TLS a longtemps été timide, jusqu'à l'arrivée de Let's Encrypt en 2015. Cette autorité de certification automatisée et gratuite fait tomber les barrières financières qui freinaient l'obtention de certificats. Google et d'autres géants du web commencent à favoriser les sites sécurisés dans leurs résultats de recherche, ce qui accélère considérablement l'adoption du protocole.

Le fonctionnement de SSL et TLS suit quatre étapes principales. La

session démarre par une négociation où client et serveur échangent leurs capacités cryptographiques. Vient ensuite l'authentification, généralement du serveur vers le client uniquement. La troisième phase établit une clé partagée via la cryptographie asymétrique. Enfin, la session utilise cette clé pour chiffrer les données échangées avec un algorithme symétrique.

Les navigateurs web ont progressivement intégré des indicateurs visuels pour signaler une connexion sécurisée. Le cadenas, devenu symbole universel de la sécurité SSL et TLS, apparaît dès les premières versions de Netscape Navigator. Les navigateurs modernes affichent ces indicateurs dans la barre d'adresse, avec des codes couleur et des messages explicites qui renseignent l'utilisateur sur le niveau de sécurité.

Les premières applications de SSL concernent surtout le commerce électronique et les services bancaires en ligne. Le protocole s'étend ensuite à la messagerie électronique sécurisée, aux réseaux privés virtuels et à bien d'autres services nécessitant des communications confidentielles.

L'évolution des menaces informatiques a conduit à enrichir TLS de nouvelles fonctionnalités. Le mécanisme de politique de sécurité HSTS (HTTP Strict Transport Security) force désormais l'utilisation du protocole sécurisé. La transparence des certificats (Certificate Transparency) favorise la détection d'émission frauduleuse de certificats.

L'histoire de SSL et TLS montre bien qu'en matière de sécurité, rien n'est jamais acquis. Les versions successives du protocole ont dû sans cesse renforcer la protection des communications tout en préservant la compatibilité avec les systèmes existants. Cette technologie s'est imposée comme le standard de facto pour sécuriser les échanges sur Internet, contribuant à l'essor du commerce électronique et des services en ligne qui exigent de la confidentialité.

Microsoft Windows NT 3.51

Microsoft lance Windows NT 3.1 (pour « New Technology » en juillet 1993. Cette première version inaugure une famille de systèmes d'exploitation qui va redessiner le paysage informatique professionnel. La plateforme vise les processeurs Intel x86 et RISC. Les analystes d'IDC définissent alors un système d'exploitation avancé par ses API 32 bits, son multitâche préemptif gérant processus et threads, son réseau intégré et sa mémoire virtuelle paginée à la demande.

Les retours sur Windows NT 3.1 poussent Microsoft à développer la ver-

sion 3.51, qui sort en 1995. Deux déclinaisons apparaissent : Windows NT Workstation et Windows NT Server. La première cherche la réactivité maximale des applications interactives, la seconde optimise les performances réseau. Ce choix reflète une stratégie claire de segmentation entre postes de travail et serveurs.

Windows NT Workstation 3.51 réduit son empreinte mémoire de 4 à 8 Mo comparé à son prédécesseur. Les applications 16 bits gagnent entre 25 et 50% de performances sur les configurations de bureau classiques. Sur les systèmes RISC (Alpha AXP et MIPS), l'émulation Intel pour les applications Windows 16 bits et MS-DOS reçoit des améliorations substantielles.

Les applications 16 bits Windows peuvent désormais tourner dans des machines virtuelles DOS (VDM) séparées. Une application qui plante ne compromet plus la stabilité des autres programmes. Les mécanismes d'intégration comme DDE (Dynamic Data Exchange) et OLE (Object Linking and Embedding) fonctionnent toujours entre les VDM.

Microsoft enrichit Windows NT 3.51 avec OpenGL, cette bibliothèque de fonctions graphiques 3D développée initialement par Silicon Graphics. Un comité réunissant Digital Equipment Corporation, IBM, Intel et Microsoft valide cette technologie. Les applications de CAO, design industriel et analyse scientifique exploitent ces nouvelles capacités 3D.

La connectivité réseau progresse nettement. Le système intègre un redirecteur NetWare compatible qui facilite l'accès aux fichiers et imprimantes des serveurs Novell via IPX/SPX. La pile TCP/IP double ses performances. PPP et SLIP permettent les connexions TCP/IP sur lignes asynchrones, ce qui attire les communautés UNIX et Internet.

La sécurité se renforce avec le verrouillage des comptes après plusieurs tentatives de connexion ratées. Cette protection bloque les attaques par force brute sur les mots de passe. NTFS gagne la compression par fichier et répertoire, qui réduit la taille des données de 40 à 50% selon leur nature.

Microsoft améliore la fiabilité. Quand une erreur système fatale survient, Windows NT 3.51 sauvegarde automatiquement l'état de la mémoire dans un fichier de débogage et redémarre. Cette approche, empruntée aux systèmes UNIX, maximise la disponibilité des machines connectées au réseau.

Le système vise quatre catégories d'utilisateurs. D'abord les entreprises recherchent sa stabilité pour limiter les coûts des redémarrages fréquents. Avec 2 000 postes redémarrant 4 fois par mois pendant 5 minutes, une organisation perd 8 000 heures de productivité par an. Les développeurs apprécient sa robustesse pour créer des applications Win16 et Win32. Les utilisateurs techniques (ingénieurs, scientifiques, statisticiens) exploitent ses performances pour leurs calculs intensifs, souvent portés depuis UNIX. Les passionnés d'informatique adoptent ses innovations sans perdre en produc-

tivité.

La gestion des périphériques s'étend aux cartes PCMCIA pour les portables haut de gamme. Le système gère maintenant les scanners SCSI et les tables traçantes HP. Le mode d'entrée absolu facilite l'usage des numériseurs et écrans tactiles. Windows NT 3.51 intègre aussi les pilotes des formats vidéo Cinepak et Indeo.

L'interface utilisateur adopte les contrôles communs de Windows 95. Le Gestionnaire de fichiers, d'impression et autres applications système héritent d'une présentation modernisée avec des info-bulles. La séquence de connexion est personnalisable, autorisant par exemple les cartes à puce bancaires.

Microsoft propose différentes formules de support technique, incluant une assistance téléphonique disponible 24 heures sur 24 pour les incidents critiques. Les entreprises peuvent choisir différents niveaux de service, du support basique aux contrats Premium avec garantie de temps de réponse. Un réseau de centres autorisés complète le dispositif.

L'administration se simplifie avec cinq modes de sauvegarde : normal (copie complète), copie (sans marquage), incrémental (fichiers modifiés), différentiel (changements depuis la dernière sauvegarde complète) et quotidien (modifications du jour). Le système documente chaque sauvegarde avec le nom de bande, la date, l'identité de l'opérateur et le numéro de séquence.

Windows NT 3.51 marque une étape importante dans l'histoire des systèmes Microsoft. Son architecture solide, ses performances et ses fonctionnalités avancées séduisent les entreprises et utilisateurs exigeants. Le système établit des bases techniques qui influenceront les futures versions de Windows, démontrant la capacité de Microsoft à concevoir enfin des systèmes professionnels fiables.

Apple Power Macintosh

L'annonce d'une alliance entre Apple, IBM et Motorola en mai 1991 aurait fait sourire n'importe quel observateur de l'industrie informatique. Ces trois entreprises, dont deux étaient des adversaires acharnés sur le marché des ordinateurs personnels, décidaient pourtant de collaborer. IBM incarnait l'ennemi historique d'Apple, celui contre lequel la marque à la pomme s'était construite depuis ses débuts.

Cette union improbable trouvait sa source dans les difficultés croissantes

d'Apple avec ses processeurs Motorola 68000. Intel grignotait inexorablement des parts de marché avec ses puces x86, moins élégantes techniquement aux yeux des puristes, mais toujours plus performantes. Motorola ne parvenait plus à tenir la cadence face à son concurrent américain. Les nouveaux Macintosh accusaient un retard préoccupant, et leur compétitivité s'érodait.

Apple devait réagir. L'entreprise se tourna vers l'architecture RISC, acronyme de Reduced Instruction Set Computing. Cette technologie venait d'IBM, qui l'avait expérimentée dès les années 1970 dans son projet 801, mené depuis un bâtiment du même nom à Yorktown Heights, dans l'État de New York. John Cocke dirigeait une équipe qui développait une architecture processeur modulaire, capable d'alimenter aussi bien des ordinateurs personnels que des machines plus imposantes.

Le 3 juillet 1991, Apple et IBM scellaient officiellement leur partenariat. Motorola les rejoignait comme troisième larron. Cette alliance AIM devait accoucher d'une nouvelle génération de processeurs RISC destinés aux futurs Macintosh. IBM et Motorola concevraient les puces ensemble. Apple et IBM collaboreraient sur un système d'exploitation orienté objet. Le PowerPC 601 surgit de cette collaboration. Cette première puce dérivait du processeur RSC d'IBM. Apple voulait des ordinateurs plus véloces que les PC à base d'Intel, sans sacrifier la compétitivité tarifaire. Le Power Macintosh 6100/60 inaugurait la gamme avec ce processeur cadencé à 60 MHz.

L'architecture RISC apportait des bénéfices tangibles. Les processeurs simplifiaient leur jeu d'instructions pour exécuter les programmes plus efficacement. Ils embarquaient 32 registres généraux, contre huit seulement pour les puces Intel x86. Cette richesse réduisait les accès à la mémoire externe et dopait les performances.

La migration vers le PowerPC représentait un casse-tête technique pour Apple. Il fallait préserver la compatibilité avec les applications existantes, écrites pour l'architecture 68000. L'entreprise y parvint grâce à un émulateur logiciel qui permettait aux utilisateurs de continuer à faire tourner leurs programmes habituels sur les nouveaux Power Macintosh, avec une performance acceptable.

Apple rationalisait sa gamme autour de trois modèles : les séries 6100, 7100 et 8100. Cette simplification contrastait avec la prolifération antérieure des références qui compliquait le choix des clients. Les machines se distinguaient par leur fréquence, indiquée après le numéro de série : 60 MHz pour le 6100/60, 66 MHz pour le 7100/66, 80 MHz pour le 8100/80.

Les Power Macintosh proposaient des fonctionnalités avancées : audio stéréo 16 bits, ports série compatibles LocalTalk et GeoPort, connectivité

Ethernet intégrée. La mémoire vive grimpait jusqu'à 72 Mo pour le modèle 6100/60, 136 Mo pour le 7100/66, 264 Mo pour le 8100/80. Ces caractéristiques les plaçaient en bonne position face aux PC concurrents.

Un détail curieux mérite l'attention dans cette histoire. Motorola n'a jamais fabriqué le moindre processeur PowerPC 601, malgré sa communication qui laissait croire le contraire. IBM Microelectronics assurait seule la production. Motorola participa néanmoins au développement des générations suivantes comme les 603, 604 et 620.

Cet arrangement satisfaisait tout le monde. IBM pénétrait le marché des ordinateurs personnels avec sa technologie RISC, initialement pensée pour ses stations de travail RS/6000. Apple obtenait des processeurs performants à bon prix. Motorola gardait un pied dans l'écosystème Macintosh, bien qu'elle perdît son statut de fournisseur exclusif.

Les Power Macintosh rencontrèrent un succès commercial notable. Ils donnaient à Apple la possibilité de vendre des machines plus rapides que les PC équivalents dans certaines applications, notamment graphiques. Un Macintosh Quadra équipé d'un processeur 68040 à 40 MHz surclassait déjà un PC avec un 486 à 66 MHz. Les nouveaux Power Macintosh creusaient l'écart.

Cette réussite technique ne bouleversa pas la position d'Apple sur le marché. L'entreprise conservait une part d'environ 10 à 15%, face à la domination écrasante des PC compatibles IBM. Les utilisateurs professionnels, guidés avant tout par des considérations budgétaires, restaient fidèles à la plate-forme Intel/Windows.

L'architecture PowerPC poursuivit son évolution pendant des années. Les processeurs gagnaient en puissance et en sobriété énergétique. Apple l'exploita jusqu'en 2006, avant de basculer vers les processeurs Intel. Cette transition fermait un chapitre technologique singulier, où des concurrents historiques s'étaient associés pour créer une alternative aux architectures établies.

CD-RW

L'idée de stocker des informations sur un disque optique remonte aux années 1950. Des Américains comme David Paul Gregg et James Russell imaginent d'écrire avec des faisceaux d'électrons et de lire avec des rayons laser. Le principe du disque rotatif et d'une surface réfléchissante transforme ces intuitions en possibilités concrètes.

Vers 1970, Hollywood s'intéresse aux disques optiques pour diffuser des films. MCA et Philips s'associent et lancent fin 1978 le premier vidéodisque laser grand public, le LaserVision. Des lasers hélium-néon lisent les creux moulés sur un disque de 30 cm, dont la taille rappelle celle des vinyles. L'information vidéo se cache dans l'espacement variable entre les bords de ces creux, disposés en spirale.

En 1974, les laboratoires Philips se lancent dans le développement d'un système audio sur disque optique. Leurs ingénieurs misent sur les technologies existantes tout en pariant sur les progrès à venir des circuits intégrés et des lasers à semi-conducteurs. Le projet prend de l'ampleur et l'équipe conclut qu'une technique numérique surpasserait l'enregistrement analogique. L'envergure du chantier pousse Philips à chercher un partenaire : Sony rejoint l'aventure en 1979.

Le compact disc adopte un diamètre de 120 mm, bien plus compact que le LaserVision. Les concepteurs savent que les lasers à semi-conducteurs du moment délivrent environ 1 mW à 800 nm. Ils ajustent l'optique en conséquence. Le faisceau laser traverse un substrat transparent de 1,2 mm avant d'atteindre les données gravées sur la couche d'aluminium du disque.

Les premiers lecteurs CD audio arrivent dans les magasins en 1982. La technologie trouve sa place dans l'univers informatique. Philips et Sony annoncent le CD-ROM en 1984, et les premiers lecteurs sont livrés l'année suivante comme périphériques pour gros systèmes. Les organisations internationales valident le standard en 1985. Mais l'organisation des données reste propriétaire jusqu'en 1988, quand l'ISO 9660 devient la référence.

La recherche sur les disques inscriptibles et réinscriptibles s'accélère dans les années 1970 aux États-Unis, en Europe et au Japon. La puissance limitée des lasers freine les avancées. En France, Thomson-CSF puis Alcatel Thomson Gigadisc expérimentent des disques en verre recouverts de couches contenant de l'or malléable. L'écriture crée des bosses microscopiques, mais les lectures laser répétées déforment ces reliefs.

Une autre piste s'avère plus féconde pour les supports WORM (Write Once Read Many) : enduire du verre ou du plastique de mélanges polymères colorants. L'optique est celle des disques en lecture seule, à condition de disposer d'une puissance crête de 50 à 100 mW. Philips et Sony définissent le CD inscriptible (CD-R) dans leur « Orange Book » de 1988. À la fin des années 1990, les lasers requis se démocratisent et la gravure de CD-R se répand dans l'informatique personnelle.

Deux technologies se disputent le marché des disques réinscriptibles : l'enregistrement magnéto-optique et le changement de phase. La première démarre fort au début des années 1970. Elle repose sur la synchronisation d'un chauffage laser avec la modulation d'un champ magnétique. Les têtes

de lecture/écriture sont complexes, mais les supports tolèrent un nombre quasi illimité de cycles.

Les supports à changement de phase utilisent une fine couche d'alliage de chalcogénures, comme l'AgInSbTe ou le GeSbTe. Cette couche est stable dans deux états : amorphe et microcristallin, chacun présentant une réflectivité distincte. Une impulsion laser brève et intense fait fondre la couche qui refroidit en état amorphe. Une impulsion plus longue mais moins énergétique chauffe le film sans le faire fondre, déclenchant la cristallisation. Des années 1970 aux années 1990, les recherches affinent les compositions d'alliages et les procédés de dépôt.

Au milieu des années 1990, des producteurs coréens et taïwanais entrent sur le marché. La première production de CD-ROM à Taiwan et en Corée démarre en 1994. Deux ans plus tard, les fabricants taïwanais représentent 12% de la production mondiale. En 1996, LG seul pèse près de 10% des ventes mondiales. Cette concurrence fait chuter les prix des lecteurs.

Le nombre de fabricants de lecteurs optiques passe de 2 en 1983 à 16 en 1985, grimpe à 65 vers 1995 pour redescendre à 44 en 1999. Les entreprises spécialisées nouvellement créées stimulent d'abord les progrès techniques. Avant 1988, année de l'ISO 9660, les lecteurs les plus rapides s'appuient sur des formats propriétaires. Après la standardisation, les géants de l'électronique et de l'informatique dominent la course à la performance.

Les prix des graveurs optiques s'effondrent : 15 000 dollars en 1991, 5 000 dollars en 1993, et moins de 1 000 dollars en 1995. Le CD-RW s'installe durablement dans le paysage informatique jusqu'à l'arrivée du DVD réinscriptible des années 2000. La technologie marque l'histoire du stockage numérique.

IBM Aptiva

IBM lance l'Aptiva en septembre 1994. L'ordinateur prend la succession de la gamme PS/1 sur le marché grand public. Les constructeurs cherchent à se démarquer dans un univers standardisé autour d'Intel et de Windows, le fameux « Wintel ». Mais la tâche s'avérera rude.

Les premiers Aptiva reçoivent des processeurs Intel 80486, avant d'évoluer vers les Pentium et les puces AMD. IBM fabrique la plupart des machines en interne, à l'exception de la série E confiée à Acer. Les ordinateurs sont vendus en solutions complètes : unité centrale, écran, enceintes, clavier et souris. La première génération tourne sous IBM PC DOS 6.3 et Windows

3.1. Les versions Pentium proposent Windows 95, et sur certains modèles une option « select-a-system » qui laisse choisir entre PC DOS 7/Windows 3.1 et OS/2 Warp.

Sur les modèles M, A, C et S, IBM intègre une carte Mwave maison pour gérer le son et le modem. Cette solution propriétaire accumule les problèmes de compatibilité et de performances. IBM finit par l'abandonner au profit de composants standards. L'entreprise doit régler un contentieux en dédommageant financièrement les acheteurs pour l'acquisition de périphériques compatibles.

En 1996 arrive la série S « Stealth », qui change sur le plan esthétique. IBM abandonne le beige traditionnel des ordinateurs pour un design noir qui tranche avec les habitudes. L'électronique grand public influence désormais l'apparence des machines. La série S sépare les lecteurs de disquettes et de CD-ROM du boîtier principal contenant la carte mère. Ces éléments trouvent place dans une console média fine qui sert de socle à l'écran. Un câble de 1,8 mètre relie le tout à l'unité centrale. Cette architecture libère l'espace de travail : le boîtier principal peut se glisser sous le bureau ou dans un meuble, tout en gardant les lecteurs à portée de main.

IBM réduit le nombre de configurations à cinq modèles. L'entrée de gamme embarque un Pentium 166 MHz pour 2 499 dollars. Le haut de gamme utilise un Pentium 200 MHz vendu 3 099 dollars, écran non inclus (compter entre 499 et 799 dollars selon la taille). Toutes les machines disposent d'au moins 16 mégaoctets de mémoire vive, de disques durs de 2,5 à 3,2 gigaoctets et de modems 28 800 bits par seconde.

Entre 1994 et 2001, IBM décline l'Aptiva en plusieurs séries identifiables par une lettre : M (Magic), A, C (Courageous), E, L et S (Stealth). Les premiers modèles adoptent un format bureau (référence 2144) ou tour (2168). La production bascule ensuite vers le format tour exclusivement. De nombreux types de machines se succèdent : 2134-2138, 2140-2144, 2151-2159, 2161-2168, 2170-2178, 2193-2198, 2255, 2270-2274, 6832 et 6864.

IBM arrête la commercialisation de l'Aptiva en 2001 sans proposer de remplaçant direct. Cette décision s'inscrit dans le retrait de l'entreprise du marché grand public. Les clients sont redirigés vers la gamme NetVista, plutôt destinée aux professionnels. Ce virage stratégique témoigne des difficultés des constructeurs traditionnels face à la standardisation du marché et à l'arrivée de nouveaux acteurs.

Netscape Navigator

En 1994, Jim Clark et Marc Andreessen fondent Netscape Communications avec une vision claire : créer une interface universelle donnant accès au Web depuis n'importe quel appareil. Clark avait fait fortune avec Silicon Graphics, tandis qu'Andreessen sortait tout juste de l'Université de l'Illinois où il avait dirigé l'équipe de développement de Mosaic, premier navigateur conçu pour le grand public. Leur association allait bouleverser la manière dont le monde accède à Internet.

La première version de Navigator sort en décembre 1994 et connaît un succès immédiat. Les internautes l'adoptent massivement. Un an plus tard, la valeur boursière de Netscape atteint 7 milliards de dollars. L'entreprise ne se limite pas au navigateur : elle développe des solutions serveur qui s'appuient sur les protocoles Internet pour faire fonctionner intranets, extranets et autres applications professionnelles.

Navigator se démarque par ses avancées techniques. Son moteur de rendu charge textes et images en parallèle, là où les navigateurs concurrents affichent d'abord le texte avant de faire apparaître les images. Un système de cache accélère le chargement des pages consultées régulièrement. La sécurité bénéficie d'une attention particulière avec l'intégration du protocole SSL (Secure Socket Layer) pour protéger les données sensibles. Des mécanismes anti-logiciels espions et anti-publicitaires viennent renforcer cette protection.

L'interface de Navigator 1.1 offre neuf façons de naviguer. On peut saisir directement une URL, suivre des liens hypertextes, utiliser un fil d'Ariane pour retracer son parcours, cliquer sur des boutons de raccourcis, revenir en arrière, consulter l'historique, gérer des signets avec annotations, prévisualiser les liens ou s'appuyer sur des repères visuels. Cette richesse fonctionnelle rend le Web accessible à ceux qui n'ont aucune formation technique.

Microsoft comprend vite que Navigator menace son empire. En 1995, Bill Gates lance Internet Explorer et l'intègre gratuitement à Windows 95. La stratégie porte ses fruits. La part de marché de Navigator, qui frôlait les 90% début 1996, commence à fondre. Fin 1997, elle passe sous la barre des 50%.

Les pertes financières s'accumulent. En novembre 1998, Netscape accepte d'être racheté par America Online (AOL) pour 4,3 milliards de dollars. AOL conclut simultanément un accord avec Sun Microsystems à hauteur de 1,25 milliard de dollars pour commercialiser les logiciels Netscape et prendre en charge ses divisions.

Netscape libère le code source de Navigator sous le nom de projet Mozilla en mars 1998. C'est la première fois qu'une entreprise cotée en bourse

rejoint le mouvement du logiciel libre. Des milliers de personnes téléchargent le code dès sa mise en ligne. Des développeurs du monde entier contribuent bénévolement au projet, comme le jeune Pavlov depuis la Géorgie. Netscape mise sur cette communauté pour améliorer le navigateur et contrer Microsoft. La licence adoptée s'inspire de la GPL (GNU Public License) tout en autorisant les exploitations commerciales.

La base de code de Navigator explose, passant de quelques centaines de milliers de lignes à plus de 2 millions. Les ingénieurs enchaînent les journées, certains dorment au bureau pour tenir les délais. La compatibilité multi-plateformes complique encore les choses : le code doit tourner sur Windows, Mac OS et diverses versions d'UNIX.

En 1997, Netscape lance une refonte ambitieuse avec le projet Communicator 6.0. L'objectif : restructurer le code en modules plus faciles à maintenir et réécrire certaines parties en Java pour simplifier le développement multi-plateformes. Début 1998, le projet est abandonné. Java n'offre pas les performances attendues. La modularisation se poursuit de façon plus progressive dans les versions 4.5 et 5.0.

La fusion avec AOL en 1999 signe la fin de l'indépendance de Netscape. De nombreux cadres vendent leurs actions durant la dernière année, mais le PDG Jim Barksdale augmente sa participation et convertit ses titres en actions AOL d'une valeur dépassant le demi-milliard de dollars lors de la fusion. Plusieurs ingénieurs quittent le navire, déçus par l'orientation marketing prise par AOL.

L'héritage de Netscape reste immense. L'entreprise a rendu le Web accessible au plus grand nombre en simplifiant la navigation. Ses innovations en matière de sécurité et de performances ont établi des standards qui perdurent. Le projet Mozilla a inspiré de nombreuses initiatives open source et donné naissance à Firefox.

L'aventure Netscape illustre la vitesse à laquelle les choses basculent dans l'industrie du logiciel. En moins de cinq ans, la start-up est devenue une simple division d'un grand groupe. Cette histoire montre aussi l'enjeu stratégique des navigateurs web, devenus l'interface universelle pour accéder aux services en ligne.

En 2003, la part de marché de Navigator est dérisoire. AOL maintient le développement jusqu'en 2008, date à laquelle le projet s'arrête définitivement. Les développeurs rejoignent l'équipe de Firefox, qui hérite des brevets de Navigator. Cette succession témoigne de la pérennité des innovations de Netscape dans le paysage du Web contemporain.

World Wide Web Consortium

En 1994, quatre ans après que Tim Berners-Lee a créé le Web, le World Wide Web Consortium (couramment appelé W3C) voit le jour. L'organisation internationale se donne une mission : conduire le World Wide Web vers son plein développement par l'élaboration de protocoles et de directives qui garantissent sa croissance sur le long terme. Dans les années 1990, la multiplication rapide des usages du Web rend cette structure indispensable.

Trois éléments techniques forment les fondations du Web. Le protocole HTTP assure les communications, les URI permettent l'identification universelle des ressources, et HTML structure le balisage des documents. Dès 1993, la combinaison de ces technologies provoque une expansion fulgurante. Le caractère libre et ouvert de ces technologies explique en grande partie ce succès : elles restent accessibles sans droits d'utilisation, là où d'autres systèmes comme Gopher échouent après que l'université du Minnesota a tenté d'imposer une licence payante.

Le W3C adopte une structure originale. Ce consortium international réunit des entreprises, des universités et des institutions publiques. Cette configuration associe les différents acteurs du Web dans un forum neutre, à l'écart des intérêts commerciaux directs. Trois institutions hébergent l'organisation : le MIT aux États-Unis, l'ERCIM en France et l'université Keio au Japon. Dix-huit bureaux régionaux complètent le dispositif et assurent une présence mondiale.

L'élaboration des standards forme le cœur de l'activité. Les recommandations techniques suivent un cheminement rigoureux. Des groupes de travail, constitués d'experts issus des organisations membres, élaborent des propositions. Celles-ci sont ensuite soumises à des périodes de commentaires publics. Cette méthode garantit la qualité technique et l'adéquation aux besoins réels. Les premières années produisent des standards fondamentaux : XML, les feuilles de style CSS, le modèle de document DOM.

Le W3C propose alors d'autoriser l'inclusion de technologies brevetées dans ses standards en 2001, moyennant des licences RAND (Reasonable And Non-Discriminatory). La communauté des développeurs, attachée aux principes du logiciel libre, s'oppose vivement à cette proposition. Après des mois de débats intenses, l'organisation adopte finalement une politique qui privilégie les technologies libres de droits. L'épisode illustre la capacité du W3C à intégrer les retours de la communauté.

Les domaines d'intervention s'étendent au fil des années. L'accessibilité est une préoccupation centrale, des recommandations voient le jour pour rendre le Web utilisable par tous, quelles que soient les capacités physiques ou mentales. L'internationalisation constitue un autre axe de travail, no-

tamment l'usage du Web dans différentes langues et systèmes d'écriture. Le consortium développe parallèlement des standards pour l'adaptation aux terminaux mobiles, anticipant la diversification des modes d'accès.

Le Web sémantique représente une direction centrale des travaux depuis le début des années 2000. Cette évolution vise à enrichir les contenus web de métadonnées compréhensibles par les machines. Les technologies RDF et OWL, normalisées par le consortium, décrivent les relations entre les informations et en simplifient le traitement automatique. Des groupes spécialisés, comme celui dédié aux sciences de la vie et à la santé (HCLS), explorent les applications pratiques de ces technologies dans différents domaines.

L'organisation adapte ses méthodes de travail en continu. Les groupes de travail, qui réunissaient initialement des représentants des membres uniquement, s'ouvrent progressivement à des experts invités. Les périodes de commentaires publics prennent une importance croissante et permettent d'intégrer les retours d'une communauté élargie de développeurs et d'utilisateurs. Cette évolution reflète la volonté de maintenir un équilibre entre expertise technique et besoins du terrain.

Les années 2010 voient l'émergence de nouveaux chantiers techniques. Des capacités multimédia natives introduites par HTML5, et réduction de la dépendance aux technologies propriétaires. Les standards pour les applications web progressives permettent la création d'applications sophistiquées directement dans le navigateur. Le W3C développe simultanément des recommandations pour la protection de la vie privée et la sécurité, en réponse aux préoccupations croissantes dans ces domaines.

La gouvernance du consortium suit un modèle original qui combine direction technique et consultation large. Tim Berners-Lee, en tant que directeur, conserve un rôle d'orientation stratégique tout en favorisant l'émergence de consensus. Une équipe permanente d'environ cinquante personnes assure la coordination des activités, tandis que plus de six cents experts participent aux différents groupes de travail.

La production de standards techniques s'accompagne d'un travail important de documentation et d'éducation. Le W3C publie des guides d'utilisation, organise des ateliers et maintient des ressources pédagogiques. Cette action contribue à la diffusion des bonnes pratiques et aide à l'adoption des nouvelles technologies. Des traductions des documents principaux sont réalisées dans de nombreuses langues, ce qui renforce l'accessibilité internationale des standards.

L'histoire du W3C montre qu'il est possible de gérer collectivement une infrastructure technique mondiale. Par son mode de fonctionnement qui associe expertise technique, consultation publique et recherche du consen-

sus, l'organisation maintient un développement cohérent du Web tout en préservant son caractère ouvert et universel.

Yahoo !

En 1994, deux étudiants en génie électrique de Stanford, David Filo et Jerry Yang, commencent à compiler une liste de sites web qu'ils apprécient. Leur petit projet personnel, qu'ils baptisent d'abord « Jerry and David's Guide to the World Wide Web », répond à un besoin simple : s'y retrouver dans un Web débutant. Le nom Yahoo ! leur vient ensuite à l'esprit, acronyme de « Yet Another Hierarchical Officious Oracle », formule qui traduit bien l'humour et la décontraction des deux fondateurs.

Le bouche-à-oreille fait le reste. En avril de la même année, l'annuaire recense une centaine de sites et attire un millier de visiteurs par semaine. Cinq mois plus tard, les chiffres s'emballent : 2 000 sites référencés, 50 000 consultations par jour. Netscape, qui vient de lancer son navigateur, décide d'afficher un lien vers Yahoo ! sur sa page d'accueil, et le trafic explose.

Les serveurs de Stanford ne suivent plus. Marc Andreessen, de Netscape, propose d'héberger gratuitement le service. En mars 1995, Yahoo ! se constitue en société et reçoit un million de dollars de Sequoia Capital. Les deux étudiants quittent l'université, s'installent dans des bureaux à Mountain View et recrutent leurs premiers employés.

La publicité en ligne démarre en août 1995. General Motors et Visa figurent parmi les premiers annonceurs. Yahoo ! expérimente le ciblage publicitaire selon les centres d'intérêt des internautes, une démarche neuve pour l'époque. Le site s'enrichit parallèlement d'actualités, de météo, de cours boursiers et d'autres contenus gratuits.

L'introduction en bourse d'avril 1996 tourne à l'euphorie. L'action augmente de 154% le premier jour, valorisant l'entreprise à 848 millions de dollars. Une performance qui surpasse celle de Netscape quelques mois auparavant. Yahoo ! utilise ces fonds pour se déployer à l'international, d'abord au Japon avec Softbank.

Les acquisitions s'enchaînent : Four11 pour la messagerie électronique, GeoCities pour l'hébergement de pages personnelles, Broadcast.com pour le multimédia. Le groupe diversifie son offre tout en gardant sa ligne directrice : un portail web généraliste et gratuit. La bulle Internet le propulse à des sommets, sa capitalisation dépassant 125 milliards de dollars en 2000. Puis la bulle éclate, les revenus publicitaires s'effondrent. Surtout, Yahoo !

peine à s'adapter. Google s'impose grâce à son algorithme PageRank. Facebook redéfinit les interactions sociales. Les tentatives de redressement, notamment sous la direction de Marissa Mayer entre 2012 et 2017, n'inversent pas la tendance.

Verizon rachète les activités Internet de Yahoo ! en 2017 pour 4,5 milliards de dollars, une fraction de sa valeur passée. Yahoo ! demeure pourtant un cas d'école des premiers temps du Web commercial. L'entreprise a rendu Internet accessible au grand public avec un portail simple. Elle a établi des modèles économiques viables, notamment via la publicité ciblée.

Les erreurs stratégiques parlent d'elles-mêmes. Yahoo ! a sous-traité sa recherche web à Google et Microsoft par la suite, renonçant à investir massivement dans ce domaine. La culture d'innovation s'est étiolée, laissant place à une bureaucratie peu réactive. Les acquisitions coûteuses se sont souvent soldées par des échecs d'intégration.

Quelques marques Yahoo ! survivent chez Verizon. Yahoo ! Finance a toujours une audience notable. L'histoire de cette entreprise montre bien la vitesse à laquelle les positions dominantes s'effritent dans la technologie. Elle souligne l'impératif d'innovation continue et d'adaptation aux usages. Le parcours de Yahoo ! incarne les promesses et les écueils du Web commercial, les difficultés à bâtir des entreprises technologiques durables.

Yahoo ! est lié à l'histoire d'Internet. En démocratisant le Web dans les années 1990, l'entreprise a participé à la diffusion de cette technologie. Son modèle de portail gratuit financé par la publicité a inspiré de nombreux services, même si Yahoo ! n'a pas conservé sa domination.

AltaVista

En 1995, Paul Flaherty, chercheur chez DEC, eut une idée pendant ses vacances. Il voulait démontrer la puissance du processeur Alpha que son entreprise venait de développer. De cette intuition allait naître AltaVista, le premier véritable moteur de recherche du Web. Deux ingénieurs de DEC, Michael Burrows et Louis Monier, prirent en charge le développement du projet.

L'infrastructure reposait sur deux machines aux noms évocateurs : Scooter et Turbo Vista. La première, équipée d'un disque dur de 20 gigaoctets et d'un gigaoctet de mémoire vive, parcourait les pages web. La seconde, avec 250 gigaoctets de stockage et deux gigaoctets de RAM, stockait les données et renvoyait les résultats. Cette configuration représentait alors

une puissance de calcul impressionnante.

AltaVista apportait une rupture technique majeure. Pour la première fois, un moteur indexait le texte intégral des pages web. Ses prédécesseurs se contentaient des titres et des en-têtes. L'interface, volontairement minimaliste, offrait pourtant des fonctions avancées : recherche par mots-clés, recherche de phrases exactes, restriction des résultats à un domaine particulier.

Le succès fut immédiat. Dès le lancement, 300 000 requêtes affluèrent chaque jour. Deux ans après, ce chiffre atteignait 80 millions. En 1998, une étude menée auprès de chercheurs professionnels révéla qu'AltaVista était leur moteur préféré à 45%, loin devant HotBot qui récoltait 20% des suffrages.

La rapidité de traitement des requêtes, rendue possible par l'architecture du processeur Alpha, faisait la différence. Le moteur proposait des fonctions innovantes comme la recherche multilingue et la traduction automatique via son service Babel Fish. Sa base de données contenait plus de 16 millions de pages web en 1996, un chiffre qui grossissait sans cesse.

Mais l'histoire d'AltaVista illustre comment une avance technologique peut fondre faute de vision stratégique. En 1998, Compaq racheta DEC. L'année suivante, sous la direction de Rod Schrock, AltaVista abandonna son interface épurée pour se transformer en portail web. L'entreprise voulait concurrencer Yahoo !. Cette décision diluait pourtant l'expertise principale d'AltaVista dans la recherche d'information.

En 1999, Compaq vendit 83% des parts d'AltaVista à CMGI, propriétaire du moteur de recherche Lycos. CMGI prépara une introduction en bourse, mais l'éclatement de la bulle internet contraignit l'entreprise à y renoncer. Pendant ce temps, Google, fondé en 1998, gagnait du terrain grâce à son focus exclusif sur la recherche et son algorithme PageRank.

En 2003, Overture Services acquit AltaVista, avant d'être racheté par Yahoo !. Le moteur de recherche pionnier se retrouva intégré à la plateforme Yahoo !, perdant son identité propre. Le 8 juillet 2013, Yahoo ! mit fin à l'existence d'AltaVista, redirigeant son domaine vers son propre moteur de recherche.

L'héritage d'AltaVista est pourtant significatif dans l'histoire du Web. Son architecture technique influença les moteurs de recherche ultérieurs. Il établit le standard de la recherche en texte intégral à grande échelle. Son interface simple et efficace inspira de nombreux concurrents, notamment Google.

L'échec d'AltaVista s'explique par plusieurs éléments. DEC considérait initialement le moteur comme un simple démonstrateur technologique plutôt qu'une opportunité commerciale. Les changements fréquents de pro-

priétaire empêchèrent toute stratégie cohérente sur le long terme. La transformation en portail web trahissait une incompréhension des attentes des utilisateurs, qui privilégiaient la simplicité et l'efficacité de la recherche.

Sur le plan technique, AltaVista avait pourtant tout bon. Son crawler, nommé Scooter, indexait des millions de pages. Le moteur proposait des opérateurs de recherche avancés comme NEAR pour mesurer la proximité des termes. Il fut le premier à analyser les métadonnées des pages web et à recourir à l'analyse linguistique pour améliorer les résultats.

Le déclin d'AltaVista coïncida avec la montée de Google, qui perfectionna des concepts introduits par son prédécesseur. Google ajouta la notion de popularité des pages dans son algorithme de classement, là où AltaVista se concentrait principalement sur la pertinence textuelle. Cette évolution répondait mieux aux besoins des utilisateurs confrontés à un web en expansion rapide.

La disparition d'AltaVista marqua la fin d'une époque pionnière du Web, où l'innovation technique pure ne suffisait plus à garantir le succès face à des concurrents maîtrisant mieux les aspects commerciaux et marketing.

Apache Web Server

Le serveur web Apache naît en 1995, mais son histoire commence deux ans plus tôt avec le NCSA HTTPd. En 1993, Rob McCool développe au National Center for Supercomputing Applications de l'Université de l'Illinois ce premier serveur HTTP portable. Distribué gratuitement avec son code source, le logiciel connaît un succès immédiat. Les administrateurs de sites web l'adoptent massivement, préférant cette solution gratuite aux serveurs commerciaux de Netscape qui coûtent plusieurs milliers de dollars. Le NCSA HTTPd est alors le serveur le plus déployé sur Internet.

Mais en 1994, l'équipe originale quitte le NCSA pour rejoindre Netscape, McCool compris. Le développement s'arrête net. Les utilisateurs se retrouvent avec un logiciel figé, certes fonctionnel mais perfectible. Ils avaient pris l'habitude de corriger les bogues par eux-mêmes, d'ajouter des fonctions selon leurs besoins, puis de partager ces modifications sur les listes de discussion sous forme de *patches*. Brian Behlendorf fait partie de ces contributeurs actifs ; il a notamment créé un système d'authentification par mot de passe pour le site Hotwired.

Face à l'abandon du projet, un petit groupe de *webmasters* décide de reprendre le flambeau. En février 1995, huit programmeurs bénévoles dont

Brian Behlendorf et Roy Fielding créent la liste de diffusion « new-httpd » pour coordonner leurs travaux. Deux mois plus tard, en avril, ils publient leur première version : la 0.6.2, construite sur NCSA HTTPd 1.3 et enrichie des correctifs développés par la communauté.

L'origine du nom Apache divise encore aujourd'hui. La fondation a longtemps expliqué qu'il rendait hommage à la tribu amérindienne des Apaches, connue pour son endurance et ses talents guerriers. Pourtant, entre 1996 et 2001, la FAQ du projet avançait une autre explication : « a patchy server », un serveur rapiécé, allusion directe aux nombreux correctifs appliqués au code original. Brian Behlendorf a successivement validé les deux versions, suggérant qu'elles coexistent peut-être toutes deux.

En décembre 1995 arrive Apache 1.0. Robert Thau pilote cette réécriture complète du code, baptisée en interne « Shambhala ». Cette version introduit une architecture modulaire qui change tout et simplifie l'ajout de nouvelles fonctions. Le développement se décentralise, l'innovation vient de partout dans la communauté, et les résultats ne se font pas attendre. Dès avril 1996, Apache est le serveur web le plus utilisé sur Internet. Il n'a jamais quitté cette première place depuis.

Le projet s'inspire du fonctionnement de l'IETF et de sa devise : « rough consensus and running code ». Les règles restent simples. Chacun peut contribuer, mais seuls les développeurs principaux votent pour les versions officielles. L'entrée de nouveaux membres votants exige une nomination et l'unanimité des membres existants. Ce système préserve la cohérence du projet tout en restant ouvert.

Apache s'impose grâce à ses innovations techniques. L'hébergement virtuel, disponible dès l'été 1995, répond à un besoin criant des fournisseurs d'accès Internet. Cette fonction pouvant gérer plusieurs sites web sur un seul serveur, jusqu'à 10 000 sites sur une seule machine. Les solutions commerciales ne proposent rien de tel. Le code source accessible offre un autre avantage décisif : les utilisateurs modifient le logiciel selon leurs besoins, sans attendre qu'un éditeur daigne implanter une nouvelle fonction ou corriger un bogue.

IBM s'intéresse à Apache en 1998. Le géant informatique cherche un serveur web pour sa gamme Websphere. Plutôt que ses propres solutions, il choisit Apache, reconnaissant sa supériorité technique. IBM comprend où réside la valeur : non dans le serveur, mais dans les services et les extensions propriétaires comme les systèmes de commerce électronique. Cette collaboration marque le début d'une relation durable. IBM contribue en retour au développement d'Apache, enrichissant le projet de son expertise.

La structuration du projet franchit une nouvelle étape en 1999 avec la création de l'Apache Software Foundation, une organisation à but non

lucratif. Cette évolution règle les aspects légaux et administratifs tout en préservant l'indépendance et la culture collaborative du projet. L'ASF développe progressivement un portefeuille de projets, dont Tomcat, l'implémentation de référence de l'architecture Java J2EE.

L'équipe de développement principale compte une quinzaine de personnes, réparties entre les États-Unis, la Grande-Bretagne, le Canada, l'Allemagne et l'Italie. Contrairement aux clichés, ces développeurs ne sont pas des adolescents passionnés mais des professionnels aguerris : doctorants, docteurs en informatique, développeurs chevronnés, chefs d'entreprise. Les développeurs principaux produisent environ 80% des nouvelles fonctions, mais plus de 400 personnes ont contribué au code et plus de 3 000 ont signalé des problèmes.

Apache domine le marché malgré une concurrence féroce. Microsoft intègre son Internet Information Server à Windows NT dès 1996, sans jamais dépasser 30% de parts de marché. D'autres acteurs se positionnent sur des niches, Zeus pour les sites à très forte fréquentation par exemple. Certains disparaissent, comme le serveur de Netscape qui avait pourtant atteint 10% du marché.

En septembre 2009, Apache sert plus de 54% des sites web mondiaux et plus de 66% des sites les plus fréquentés. Cette domination s'auto-entretient : la large adoption génère des retours d'expérience et des contributions qui renforcent la qualité et la stabilité du logiciel, attirant à leur tour de nouveaux utilisateurs.

Cette réussite a inspiré d'innombrables autres projets et façonné les pratiques du développement open source.

BeOS

En 1990, Jean-Louis Gassée et Steve Sakoman quittent Apple pour fonder Be Inc. Leur projet est de créer un système d'exploitation totalement neuf, capable de tirer parti des machines modernes sans traîner le boulet de la compatibilité avec l'existant. Apple traverse alors une période difficile, son vieux système montrant ses limites face aux progrès du matériel.

BeOS naît de cette volonté de repartir d'une page blanche. Les ingénieurs de Be construisent un système pensé d'abord pour le multimédia, sans se soucier des compromis habituels. Le multithreading traverse toutes les couches du système, chaque tâche se divise en fils d'exécution multiples pour exploiter au mieux les ressources. Le système de fichiers BFS en 64

bits intègre des capacités de base de données et un journal pour protéger les données. La gestion du multiprocesseur symétrique s'avère soignée ce qui fait que BeOS tourne plus vite sur un Pentium 233 avec 64 Mo de mémoire que Mac OS X sur des machines huit fois plus musclées.

Les démonstrations de 1995-1996 impressionnent. BeOS fait tourner simultanément plusieurs fenêtres de rendu 3D temps réel, lit de la vidéo et navigue sur le web sans broncher, là où les systèmes concurrents peinent. Apple, cherchant désespérément un remplaçant moderne pour son système vieillissant, engage des discussions pour racheter Be Inc. en 1996. Les négociations échouent sur le prix. Apple se tourne vers NeXT, ramenant Steve Jobs dans la maison qu'il avait créée.

Be Inc. poursuit sa route et vise le marché PC Intel à partir de 1998, tout en gardant sa version PowerPC. Le système séduit par son interface élégante, sa réactivité et ses trouvailles techniques comme les requêtes dynamiques dans le système de fichiers. Les utilisateurs apprécient la stabilité, la rapidité d'exécution, la gestion fine des attributs de fichiers.

Mais BeOS ne parvient pas à percer. Les applications manquent cruellement. Les développeurs hésitent à investir du temps sur une plateforme qui compte peu d'utilisateurs, et les utilisateurs attendent des applications avant d'adopter le système. Le serpent se mord la queue. Windows écrase le marché PC, les fidèles du Mac restent sur leur plateforme, l'espace se fait étroit pour un nouveau venu.

Be Inc. tente un virage en 1999 vers les appareils Internet émergents. Trop tard. Les investisseurs se lassent de financer un projet qui ne trouve pas son public. L'entreprise ferme en 2001 et cède ses brevets à Palm.

L'histoire ne s'arrête pas complètement là. Haiku, un projet open source, reprend le flambeau et reconstitue BeOS dans l'esprit. Certaines idées du système, la gestion avancée des métadonnées ou l'architecture orientée multimédia, infusent ailleurs.

BeOS raconte les difficultés d'un marché où l'excellence technique ne suffit pas. Le timing joue son rôle : BeOS arrive probablement trop tard pour bousculer les systèmes installés. Reste l'empreinte d'une architecture moderne, d'une attention aux performances qui continue d'inspirer. Le système montre qu'en se libérant des contraintes historiques, on peut atteindre des résultats remarquables.

La leçon porte sur l'écosystème logiciel. Sans applications populaires, un système si brillant soit-il s'étiole. Cette réalité pèse encore sur les stratégies actuelles, où bâtir une communauté de développeurs compte autant que les prouesses techniques.

L'échec commercial de BeOS n'efface pas l'apport technique ni les questions soulevées sur les dynamiques du marché.

DVD

En 1994, l'industrie cherche un successeur au CD-ROM. Les besoins de stockage augmentent, la vidéo numérique progresse, et le CD montre ses limites. Deux camps se forment autour de technologies concurrentes : le Super Disc d'un côté, le Multimedia CD de l'autre. Personne ne veut revivre la même guerre fratricide qui avait laissé des traces dans les années 1980 entre VHS contre Betamax. Les industriels finissent par s'entendre sur une norme commune.

Cette convergence réunit des acteurs que tout aurait pu séparer : fabricants d'électronique grand public japonais, studios hollywoodiens, géants de l'informatique américains. Ils créent ensemble un disque aux dimensions du CD mais capable de stocker sept fois plus de données. Les Japonais découvrent les premiers lecteurs en novembre 1996. Les Américains doivent patienter jusqu'en août 1997.

Le DVD repose sur une astuce technique simple mais efficace. Là où le CD empile 1,2 mm de polycarbonate en une seule couche, le DVD superpose deux substrats de 0,6 mm. Cette architecture autorise l'usage d'un laser à longueur d'onde réduite : 650 ou 635 nanomètres contre 780 pour le CD. Les pistes se resserrent, les alvéoles rapetissent. Un DVD simple couche contient 4,7 Go. La double couche atteint 8,5 Go par face. Un disque double face stocke 17 Go en théorie, mais ce format reste marginal.

Hollywood adopte le DVD avec enthousiasme. La compression MPEG-2 fait tenir un long métrage de deux heures sur une seule couche, avec une image qui surpasse celle des cassettes VHS et des LaserDisc. Les studios apprécient les menus interactifs, les pistes audio multiples et les bonus. Le spectateur choisit sa langue, active les sous-titres, découvre des scènes coupées. Ce n'est plus un simple film qu'on regarde : c'est une expérience qu'on explore.

Les fabricants tirent les leçons du passé. Tout lecteur DVD est capable de lire les CD existants. Cette rétrocompatibilité rassure les consommateurs qui gardent leurs collections de CD audio et leurs CD-ROM de jeux. La transition se fait sans douleur, sans obligation de tout racheter. Microsoft intègre le support DVD dans Windows 98, normalisant son usage dans l'univers PC.

L'informatique trouve son compte dans cette histoire. Les lecteurs DVD-ROM remplacent progressivement leurs prédécesseurs dans les tours d'ordinateurs. Les développeurs de jeux vidéo et de logiciels multimédia disposent soudain d'espaces généreux pour leurs créations. Myst et Riven avaient dû

se contenter de plusieurs CD ; leurs successeurs tiendront sur un seul DVD.

Le marché se complique avec l'arrivée des formats enregistrables. Le DVD-R débarque en 1997 pour une écriture unique. Puis viennent DVD-RAM, DVD-RW, DVD+RW, chacun avec ses partisans et ses spécificités techniques. Cette profusion crée de la confusion mais stimule aussi l'innovation et tire les prix vers le bas. Les consommateurs s'y perdent parfois, mais les graveurs multistandards finissent par s'imposer.

La question de la protection des contenus obsède les studios. IBM et Disney collaborent pour mettre au point le Content Scrambling System, un cryptage censé empêcher les copies pirates. Le système CSS ne bouleverse pas les performances de lecture mais donne aux producteurs de films la garantie qu'ils réclament. Sans cette protection, nombre d'entre eux auraient sans doute rechigné à adopter le format.

Les chiffres de vente parlent d'eux-mêmes. Entre 1997 et 2002, plus de 27 millions de lecteurs trouvent acquéreur aux États-Unis. Le prix d'entrée fond comme neige au soleil : plus de 1 000 dollars en 1997, moins de 200 en 2001. Cette démocratisation foudroyante signe l'arrêt de mort du VHS. Les vidéoclubs transforment leurs rayonnages, les particuliers renouvellent leurs équipements.

Le DVD tente quelques escapades hors de son territoire naturel. Le DVD Audio, au succès mitigé, veut détrôner le CD avec une qualité sonore supérieure et des fonctions multimédia. Le DVDPlus, appelé aussi DualDisc, propose une face DVD et une face CD sur le même disque. Ces variations enrichissent l'écosystème sans vraiment s'imposer, le DVD étant avant tout associé à la vidéo domestique.

Au-delà du divertissement, le format trouve sa place dans l'éducation et la formation professionnelle. Sa capacité à combiner vidéo haute qualité, son multicanal et interactivité en fait un outil pédagogique apprécié. Les bibliothèques l'utilisent pour numériser et diffuser leurs fonds audiovisuels. Les entreprises distribuent leurs catalogues produits sur DVD interactifs.

Dès le début des années 2000, les laboratoires explorent l'avenir. Le DVD holographique promet 100 Go par disque. Ces recherches préparent l'arrivée du Blu-ray et de la haute définition. Le DVD a accompli sa mission : démocratiser la vidéo numérique, standardiser les pratiques, préparer le terrain pour les générations suivantes.

GPU

Dans les années 1980, le traitement graphique restait l'affaire de contrôleurs VGA rudimentaires. Ces composants ne faisaient que recevoir des données d'image, les organiser et les transmettre à un moniteur. Rien de plus.

L'IBM Professional Graphics Controller sort en 1984. Cette carte intègre un microprocesseur Intel 8088 spécialement dédié aux tâches graphiques, libérant ainsi le processeur principal. À 5 500 dollars et avec une compatibilité limitée, elle ne connaît qu'une diffusion confidentielle. Mais l'idée germe : confier les calculs graphiques à un processeur spécialisé.

La standardisation OpenGL par SGI stimule tout un écosystème de solutions matérielles dédiées au rendu graphique. La décennie suivante accélère le mouvement. SGI présente en 1993 le RealityEngine, dont l'architecture établit les bases du pipeline graphique moderne. Ce principe du pipeline, qui transforme des coordonnées 3D en pixels 2D par étapes successives, structure encore aujourd'hui le fonctionnement des GPU.

Puis arrive la 3dfx Voodoo en 1996. Cette carte grand public dédiée à l'accélération 3D bouleverse le jeu vidéo sur ordinateur personnel. Un million de transistors, 4 Mo de mémoire DRAM 64 bits, une fréquence de 50 MHz : des chiffres offraient alors des performances jamais vues.

NVIDIA frappe fort en 1999 avec la GeForce 256. C'est la première carte commercialisée sous le nom de GPU. Ce choix terminologique n'a rien d'anodin : il souligne l'intégration complète des fonctions de traitement graphique sur une seule puce, transformation géométrique et éclairage compris. Ses 23 millions de transistors et ses 32 Mo de mémoire inaugurent l'ère moderne du processeur graphique.

Le début des années 2000 apporte la programmabilité. En 2001, sur la GeForce 3 de NVIDIA, les développeurs programment certaines étapes du pipeline via des shaders. Cette souplesse nouvelle ouvre des horizons pour les effets visuels et le rendu 3D.

L'année 2002 voit débarquer une génération de GPU entièrement programmables avec la GeForce FX et la Radeon 9700. Les développeurs peuvent désormais programmer les opérations par pixel et par vertex, en contrôlant finement le rendu graphique. La GeForce FX, avec ses 80 millions de transistors et sa mémoire DDR de 128 Mo, témoigne de la complexité grandissante de ces composants.

La GeForce 8800 de NVIDIA, en 2006, opère un virage. Son architecture unifiée remplace les unités de traitement spécialisées par des processeurs polyvalents capables d'exécuter différents types de calculs graphiques. CUDA accompagne cette évolution : cet environnement de programmation

autorise l'usage du GPU pour des calculs généraux, bien au-delà du simple graphisme.

L'architecture Fermi, que NVIDIA dévoile en 2009, confirme cette orientation vers le calcul généraliste. Première architecture GPU pensée pour le calcul scientifique, elle intègre une hiérarchie de mémoire cache, la correction d'erreurs ECC et améliore les performances en double précision. Les GPU se rapprochent des processeurs classiques tout en conservant leur puissance de traitement massivement parallèle.

Cette tendance à la fusion s'observe partout. AMD lance en 2011 sa ligne Fusion, qui intègre GPU et CPU sur une seule puce. Intel développe de son côté Larrabee, une architecture combinant des cœurs x86 avec des unités vectorielles larges. La convergence entre processeurs graphiques et généralistes s'accélère.

Les GPU modernes incarnent la puissance du traitement parallèle. Leur architecture, fruit d'une évolution continue depuis les simples contrôleurs graphiques des années 1980, fait tourner aujourd'hui une gamme étendue d'applications : rendu 3D, calcul scientifique, intelligence artificielle. Cette polyvalence, alliée à une puissance de calcul considérable, fait des GPU des composants indispensables des systèmes informatiques contemporains.

Microsoft Internet Explorer

Le milieu des années 1990 voit le Web prendre son envol. Les premiers navigateurs grand public apparaissent, et Microsoft décide d'entrer dans la danse. Le 16 août 1995, l'entreprise de Redmond lance Internet Explorer 1.0. C'est le début d'une aventure qui durera plus de vingt ans et qui marquera l'histoire d'Internet.

Internet Explorer n'est pas né dans les laboratoires de Microsoft. L'entreprise a pris un raccourci en obtenant une licence du code source de Spyglass Mosaic, un navigateur développé par la société Spyglass. D'autres entreprises ont fait un choix identique à l'époque. Spyglass Mosaic s'appuyait sur les travaux du National Center for Supercomputing Applications, qui avait créé NCSA Mosaic, le premier navigateur graphique à connaître un succès populaire.

Au départ, l'équipe chargée du projet est minuscule : une demi-douzaine de développeurs tout au plus. Microsoft distribue son navigateur de différentes façons. Il est inclus dans le pack Microsoft Plus ! pour Windows 95, un ensemble d'extensions et d'utilitaires vendus à part. Les fabricants d'or-

dinateurs reçoivent une version OEM à préinstaller sur leurs machines. Les utilisateurs peuvent l'installer via l'Internet Jumpstart Kit fourni avec le pack Plus !. Un point intéressant : il fonctionne sur la version originale de Windows 95, sans mise à jour du système.

Quelques mois plus tard, Microsoft sort Internet Explorer 1.5 pour Windows NT. Cette version apporte le support basique des tableaux HTML, un élément qui devenait indispensable pour construire des pages Web.

La stratégie de Microsoft évolue vite. L'entreprise comprend qu'Internet n'est pas une mode passagère et change son fusil d'épaule. Dans les versions suivantes de Windows 95, le navigateur s'intègre de plus en plus au système d'exploitation. Cette intégration croissante va déclencher des polémiques qui marqueront l'histoire du navigateur.

Les documents internes de Microsoft révèlent que l'entreprise n'avait pas prévu d'inclure un navigateur avec Windows 95. Début 1994, les plans mentionnent seulement l'ajout d'éléments de base pour se connecter à Internet : la pile TCP/IP et le support des protocoles PPP et SLIP. Steven Sinofsky écrit en juin 1994 qu'aucun logiciel client comme Mosaic ou Cello n'est prévu dans Windows 95.

Le changement s'opère par étapes. En 1995, Microsoft hésite encore. Faut-il commercialiser le navigateur dans le pack « Plus ! » ou l'intégrer directement à Windows ? Les discussions internes montrent des interrogations sur la pertinence de cette intégration, notamment à cause des contraintes d'espace sur les disquettes d'installation.

Sur le plan technique, Internet Explorer 1.0 reflète son époque. L'interface graphique reste simple, les fonctionnalités limitées. On navigue page par page, sans onglets ni blocage de publicités. Mais cela correspond aux besoins des utilisateurs de 1995, qui découvrent le Web.

Les fabricants d'ordinateurs contribuent à la diffusion du navigateur. Microsoft signe des accords avec eux pour préinstaller Internet Explorer sur les machines neuves. Cette approche assure une distribution massive du logiciel.

L'architecture logicielle pose les bases des versions futures. Le code hérité de Spyglass Mosaic est remanié pour s'adapter à Windows. Les développeurs modifient l'interface et optimisent les performances pour le système d'exploitation de Microsoft. La documentation montre pourtant que Microsoft considère Internet Explorer comme un produit distinct de Windows. Les communications internes et externes présentent le navigateur comme une application autonome, sans lien avec le système d'exploitation.

Cette première version constitue une étape dans l'histoire du Web. Microsoft entre dans un domaine en pleine expansion avec une approche pragmatique : acquérir une technologie existante plutôt que tout développer en

interne.

Java

En 1991, une équipe d'ingénieurs de Sun Microsystems se lance dans le projet « Green » sous la direction de James Gosling. Leur ambition est de créer un système distribué qui permettrait aux appareils électroniques de dialoguer entre eux dans un réseau hétérogène. L'idée était de proposer aux fabricants d'électronique grand public une technologie adaptée à leurs besoins.

Les ingénieurs démarrent leurs travaux avec C++. Mais la nécessité d'adapter le code à différents processeurs les pousse à envisager des modifications du compilateur. Malgré les extensions possibles, C++ se révèle trop complexe pour répondre à leurs exigences. Cette impasse les conduit à concevoir un nouveau langage : Oak, baptisé ainsi d'après l'arbre que Gosling aperçoit depuis la fenêtre de son bureau.

Oak voit le jour pour programmer des dispositifs de contrôle de télévision interactive et des systèmes domotiques. Après plusieurs mois d'un travail intense, l'équipe accouche d'un système d'exploitation complet, d'une boîte à outils, d'une interface, d'une plateforme matérielle inédite et de trois puces personnalisées. Le tout aboutit au « *7 » (Star 7), que Gosling décrit comme une « télécommande portable ».

L'équipe Green, devenue FirstPerson Inc., répond à un appel d'offres en 1993 pour un système d'exploitation de décodeur et une technologie de vidéo à la demande. Leur solution brille par son excellence technique, mais le marché leur échappe au profit de SGI pour des raisons que Gosling juge non techniques. FirstPerson persiste dans le secteur des décodeurs jusqu'en 1994, date à laquelle la réalité s'impose : ce marché n'est pas viable. L'entreprise disparaît, et la moitié de ses effectifs rejoint Sun Interactive pour travailler sur des serveurs de données vidéo numériques.

Tout bascule vers la mi-1994. Internet et le World Wide Web explosent. L'équipe décide alors de réorienter Oak vers les applications web. Les caractéristiques du langage, notamment son indépendance architecturale et sa neutralité vis-à-vis des plateformes, se prêtent remarquablement à cet environnement.

En janvier 1995, Oak devient Java, un nom disponible à l'enregistrement. Le langage se transforme en outil de création d'applications web. Pour en démontrer le potentiel, l'équipe conçoit HotJava, un navigateur

web entièrement écrit en Java, capable d'exécuter des applications Java (les Applets) directement intégrées aux pages web. Netscape et Microsoft s'empareront de cette fonctionnalité. Sun publie ensuite le kit de développement initial (JDK) accompagné de HotJava.

La simplicité de Java frappe d'emblée. Le code se lit et s'écrit aisément, ce qui réduit les risques d'erreurs. L'environnement d'exécution prend en charge automatiquement l'allocation de mémoire et le ramassage des miettes, éliminant ainsi une source majeure de bogues qui empoisonne C ou C++.

Java impose une approche orientée objet rigoureuse, qui structure et modularise la programmation. Son interprétation autorise l'exécution des programmes sur diverses plateformes sans recompilation, grâce à la Machine Virtuelle Java (JVM). Les programmes se compilent en *bytecode*, un format intermédiaire indépendant du matériel. Cette caractéristique garantit la portabilité des applications Java sur tout système doté d'un interpréteur Java.

La robustesse du langage tient à son système de typage fort et ses vérifications poussées lors de la compilation. L'absence de pointeurs, ces sources récurrentes de problèmes dans d'autres langages, renforce la fiabilité des programmes.

Java se décline ensuite en plusieurs éditions. Java 2 Standard Edition (J2SE) vise les ordinateurs de bureau classiques. Java 2 Enterprise Edition (J2EE) s'adresse aux applications d'entreprise. Java 2 Micro Edition (J2ME) couvre les appareils mobiles et embarqués.

Les améliorations s'enchaînent. Java 1.02, sorti en 1996, souffre de limitations importantes, comme l'impossibilité d'imprimer. Les versions suivantes enrichissent progressivement les fonctionnalités. Java 1.2, publié en décembre 1998 et rebaptisé Java 2 trois jours plus tard, introduit notamment de nouvelles bibliothèques graphiques plus élaborées.

La communauté des développeurs adopte Java pour des raisons variées. Le langage simplifie la création d'applications réseau, héritage de sa conception initiale pour les systèmes distribués. Son modèle de sécurité intégré protège contre les programmes malveillants. Ses capacités multitâches facilitent le développement d'applications performantes et réactives.

L'architecture de la Machine Virtuelle Java témoigne de la sophistication technique du système. Le chargeur de classes, la zone de méthodes, le tas et la pile Java composent cet environnement d'exécution. Le moteur d'exécution combine un interpréteur et un compilateur Just-In-Time qui optimise les performances en traduisant le *bytecode* en code natif.

Java a transformé la programmation web en introduisant les applets, qui exécutent du code dans les navigateurs. Cette innovation a enrichi

l'expérience utilisateur sur Internet et inspiré de nouvelles approches du développement web.

La philosophie « Write Once, Run Anywhere » (écrire une fois, exécuter partout) a influencé la conception des langages et plateformes ultérieurs. Le modèle de sécurité de Java, son ramasse-miettes automatique et son approche de la programmation orientée objet ont établi des standards qui perdurent dans le développement logiciel contemporain.

JavaScript

En 1995, le Web traverse une phase d'expansion rapide. Netscape et Microsoft se livrent une bataille commerciale intense autour de leurs navigateurs respectifs. Brendan Eich, ingénieur chez Netscape, reçoit une mission pour le moins inhabituelle : concevoir un langage de programmation en dix jours, un outil qui s'exécutera directement dans le navigateur Netscape. Mais cette demande s'inscrit dans une vision stratégique plus large. Netscape ne voit plus le navigateur comme une simple application ; l'entreprise le considère, avec le serveur, comme une nouvelle forme de système d'exploitation distribué.

À cette époque, Java de Sun Microsystems s'impose comme la solution de référence pour les applications Web complexes. Ce langage compilé génère du *bytecode* pour sa machine virtuelle, reprend les principes orientés objet de C++ et promet des performances comparables aux langages natifs. Pourtant, Netscape identifie un manque : il faut un second langage, plus léger, interprété, qui viendrait compléter Java. Un outil accessible aux programmeurs amateurs, qui s'intégrerait sans friction dans les pages Web.

Brendan Eich conçoit alors JavaScript comme un langage aux influences multiples. La syntaxe emprunte au C ses accolades et ses points-virgules, ainsi que les structures de données, tandis que les modèles s'inspirent de Smalltalk. La symétrie entre données et code, caractéristique de LISP, trouve sa place dans la conception. Pour la gestion des événements, Eich se tourne vers HyperCard. Quant à l'approche orientée objet, elle s'appuie sur une sémantique d'exécution utilisant des prototypes, à la manière du langage Self, plutôt que sur une syntaxe de classes comme Java.

Les premières années du langage restent modestes. JavaScript sert surtout à faire défiler des messages dans la barre d'état du navigateur ou à animer quelques images. Rien de bien spectaculaire. Mais le langage contient suffisamment d'éléments fondamentaux pour perdurer. Son adoption initiale limitée lui laisse le temps d'évoluer progressivement, notamment grâce

au processus de standardisation ECMA qui améliore ses performances et sa robustesse.

Le véritable tournant survient entre 2004 et 2005 avec l'arrivée d'Ajax. JavaScript peut désormais récupérer des données depuis les serveurs et mettre à jour les documents HTML sans recharger toute la page. Microsoft introduit cette fonctionnalité dans Internet Explorer via l'objet XMLHttpRequest, repris par les autres navigateurs. L'interface utilisateur bascule en partie vers le navigateur, créant des expériences beaucoup plus riches. Gmail et Google Maps illustrent cette transformation.

Ce changement révèle vite les faiblesses des moteurs JavaScript des navigateurs. Les pages Web ne redémarrent plus toutes les minutes. Elles maintiennent des sessions prolongées, manipulent des données dynamiques volumineuses, communiquent en continu avec les serveurs en arrière-plan. Google développe Chrome et son interpréteur V8 pour répondre à ces nouveaux besoins de performance. Le marché suit le mouvement. Les performances des interpréteurs JavaScript progressent chez tous les éditeurs.

En 2012, Mozilla lance asm.js, un sous-ensemble strict de JavaScript optimisé pour exécuter du code compilé depuis C ou C++. Grâce à cette innovation, des jeux complexes sont portés sur le Web. Epic Games en fait la démonstration spectaculaire en adaptant son moteur Unreal Engine 3 en JavaScript en quelques jours seulement. Le succès d'asm.js influencera directement la création de WebAssembly, un format binaire plus efficace pour exécuter du code natif dans les navigateurs.

Node.js élargit ensuite le terrain de jeu de JavaScript au-delà du navigateur. Le langage investit le développement d'applications serveur. Sa nature événementielle, présente depuis l'origine, facilite la création d'applications Web hautement évolutives sans la complexité de la programmation « multithreadée ».

La standardisation continue via ECMAScript apporte régulièrement de nouvelles fonctionnalités. Les modules ES6 en sont un exemple. Tout cela en préservant une compatibilité ascendante indispensable à la pérennité du Web. JavaScript se transforme progressivement d'un simple outil d'animation en un langage polyvalent capable de gérer des applications complexes, que ce soit côté client ou serveur.

L'écosystème qui l'entoure s'enrichit constamment. Les transpileurs permettent d'utiliser les dernières fonctionnalités tout en restant compatible avec les anciennes versions des navigateurs. Les gestionnaires de paquets comme npm facilitent le partage de code entre développeurs. React, Angular, Vue.js standardisent le développement d'interfaces utilisateur complexes.

Le succès de JavaScript tient beaucoup à son accessibilité et sa capacité

d'adaptation. Malgré ses imperfections initiales, liées à sa création express, le langage a su répondre aux besoins changeants du Web. Son interprétation directe, sa flexibilité, son intégration native dans les navigateurs en font un élément central du développement Web moderne.

La position dominante de JavaScript ne signifie pas pour autant la fin de son évolution. WebAssembly ouvre de nouvelles perspectives en tant que format de compilation universel pour le Web. D'autres langages peuvent désormais s'exécuter dans le navigateur. JavaScript conserve néanmoins son rôle d'interface privilégiée avec le DOM et les APIs Web. Il démontre sa capacité à coexister avec les nouvelles technologies.

PHP

En 1994, Rasmus Lerdorf, programmeur canadien d'origine groenlandaise, développe quelques scripts Perl pour gérer les visites sur son CV en ligne. Ces outils baptisés « Personal Home Page Tools » servent d'abord à afficher son curriculum vitæ et à enregistrer le trafic. Rien de bien spectaculaire au départ, juste une solution pratique à un besoin personnel.

Consultant en développement web à Toronto, Lerdorf constate qu'il réécrit sans cesse les mêmes scripts CGI en C pour ses différents clients. Cette répétition le pousse à rassembler son code dans une bibliothèque C qu'il intègre au serveur web NCSA. Il y ajoute un système de templates pour simplifier les appels. L'idée reste pragmatique : gagner du temps et travailler plus efficacement.

Le 8 juin 1995, il publie la première version publique sous le nom « Personal Home Page Tools version 1.0 ». Il a réécrit ses scripts en binaires CGI, ajoutant des fonctionnalités pour traiter les formulaires web et dialoguer avec les bases de données. Le tout prend alors le nom de « Personal Home Page/Forms Interpreter », ou PHP/FI. Lerdorf cherche surtout à détecter les bogues plus vite et à améliorer son code grâce aux retours d'autres développeurs.

Cette première mouture contient déjà des variables inspirées de Perl, la gestion des formulaires et la possibilité de s'intégrer directement dans le HTML. La syntaxe emprunte à Perl tout en restant plus accessible, la cohérence n'étant pas toujours au rendez-vous. Les développeurs web commencent à s'y intéresser, séduits par cette manière plus simple de créer des pages dynamiques.

Lerdorf ne considère pas PHP comme sa propriété. Il voit plutôt un outil

pour résoudre des problèmes. Quand d'autres développeurs lui demandent d'utiliser ses scripts, il les partage volontiers. Sa valeur ajoutée réside dans sa capacité à résoudre les problèmes de ses clients, pas dans la possession exclusive du code.

Le vrai tournant arrive en 1997. Zeev Suraski et Andi Gutmans, deux développeurs israéliens du Technion IIT, réécrivent l'analyseur syntaxique. Cette refonte donne naissance à PHP 3, où le nom devient un acronyme récursif : « PHP : Hypertext Preprocessor ». Les tests publics commencent, et la version officielle sort en juin 1998. Les deux Israéliens poursuivent leur travail en réécrivant le cœur de PHP, créant ainsi le moteur Zend en 1999. Ils fondent Zend Technologies à Ramat Gan.

PHP 4 sort le 22 mai 2000, propulsé par le moteur Zend 1.0. Cette version améliore nettement le support de la programmation orientée objet et les performances générales. Le langage gagne en maturité et s'affirme comme un vrai outil pour bâtir des sites web dynamiques.

La version 5 arrive le 13 juillet 2004 avec le moteur Zend Engine II et un nouveau modèle objet. La programmation orientée objet est vraiment exploitable, et les PHP Data Objects (PDO) offrent une interface unifiée pour accéder aux bases de données. En 2008, PHP 5 est la seule version stable activement développée.

L'histoire de PHP montre combien la communauté compte dans son évolution. Le langage s'enrichit grâce aux contributions de nombreux développeurs. La PHP Extensions Community Library (PECL) sert d'incubateur où des groupes de volontaires expérimentent de nouvelles idées avant qu'elles n'intègrent éventuellement la distribution principale.

La gestion du projet suit une logique méritocratique. Le code prime sur les débats théoriques. Si un développeur propose une implémentation qui fonctionne, elle sera probablement adoptée, quel que soit son degré de perfection. Cette philosophie accélère l'innovation et l'ajout de fonctionnalités, parfois au détriment d'une certaine cohérence d'ensemble.

Le succès de PHP tient beaucoup à sa simplicité d'usage et sa proximité avec le HTML. Les développeurs peuvent insérer directement du code PHP dans leurs pages web, ce qui rend le langage accessible aux débutants. Cette facilité d'apprentissage explique en grande partie sa popularité grandissante.

Le développement se poursuit avec PHP 7.0 en 2015. Le moteur Zend Engine 3 booste considérablement les performances. La vitesse d'exécution s'améliore de manière spectaculaire, et de nouvelles fonctionnalités apparaissent comme les déclarations de type scalaire ou l'opérateur combiné de comparaison.

PHP 8.0 sort en 2020, marquant une nouvelle étape avec le compila-

teur JIT (Just-In-Time), les types d'union et les expressions match. Ces évolutions renforcent la robustesse du langage sans sacrifier sa simplicité d'utilisation.

Au fil des années, PHP est devenu incontournable dans le développement web. Son évolution reflète les changements dans les pratiques de programmation et les besoins du web moderne. D'un simple outil personnel, il s'est transformé en langage complet supportant la programmation orientée objet, la gestion des exceptions et de nombreuses fonctionnalités avancées.

Ruby

Le 24 février 1993, deux informaticiens japonais, Yukihiro Matsumoto et Keiju Ishitsuka, discutent en ligne du nom à donner à leur nouveau projet de langage de programmation. Ils veulent une pierre précieuse. Après avoir hésité avec « Coral », ils choisissent « Ruby ». Plus court d'un caractère, donc plus simple à taper.

Yukihiro Matsumoto, que tout le monde appelle Matz, veut créer un langage de script orienté objet vraiment agréable à utiliser. Il emprunte à Perl sa souplesse, à Smalltalk sa vision orientée objet, à Eiffel sa rigueur, à Ada sa fiabilité, à Lisp son expressivité. De ce mélange naît un langage qui se distingue par son élégance.

La première version publique, la 0.95, sort le 21 décembre 1995 sur les forums japonais. Le succès au Japon est immédiat. En 1997, netlab.jp, une entreprise japonaise spécialisée dans l'open source, embauche Matz pour qu'il travaille à plein temps sur Ruby. La version 1.0 paraît en 1996, la 1.1 en août 1997. Le développement s'accélère.

Matsumoto conçoit Ruby selon le principe de moindre surprise. Les méthodes portent des noms tirés du vocabulaire anglais courant qui décrivent naturellement leur fonction. Pour les chaînes de caractères, on trouve des verbes comme `strip`, `split`, `delete` ou `upcase`. L'apprentissage est intuitif, la maintenance du code plus aisée.

Dans Ruby, tout est objet. Vraiment tout, y compris les nombres et les types primitifs. Cette approche, héritée de Smalltalk, unifie la syntaxe. Les développeurs appliquent les mêmes principes à tous les éléments qu'ils manipulent. Le langage gagne en cohérence.

Ruby autorise la modification de ses composants essentiels, les programmeurs pouvant ainsi redéfinir, enrichir les classes existantes, et ajouter des méthodes aux classes intégrées. On peut par exemple enrichir la classe des

nombres d'une méthode `plus` pour rendre le code plus expressif. Cette flexibilité séduit.

En 2001, Ryan Leavengood commence à développer RubyGems, un outil pour simplifier la distribution des bibliothèques de code. Le projet s'arrête quand Leavengood le quitte à la version 0.4.0, mais une nouvelle équipe le reprend en 2003 et le réécrit entièrement. RubyGems se mue alors un élément central de l'écosystème.

L'année 2004 change tout. David Heinemeier Hansson crée Ruby on Rails, un framework web qui propulse Ruby sur la scène internationale. Cette plateforme séduit par son approche pragmatique et sa productivité. Apple intègre Ruby on Rails à Mac OS X v10.5 « Leopard ». Cette reconnaissance par un géant de l'industrie accélère l'adoption du langage.

La popularité de Ruby culmine en 2006, année où il reçoit le titre de « Langage de l'année ». En 2016, il atteint la huitième place de l'index TIOBE. Entre 2013 et 2020, Ruby évolue de la version 2.0.0 à la version 3.0.0. Le langage atteint sa maturité technique.

La communauté reste dynamique. En 2022, on compte 2,4 millions de développeurs Ruby dans le monde. Airbnb, Shopify, Coinbase et GitLab utilisent Ruby et contribuent à son développement. La Fondation Rails, créée récemment, travaille sur la documentation, la formation et la promotion du langage.

La version 3.0, sortie en 2020, apporte des améliorations significatives en termes de performances. Le projet « Ruby 3x3 » vise à tripler les performances de la version 3.x par rapport à Ruby 3.0. Les développeurs optimisent le compilateur MJIT, perfectionnent le ramasse-miettes et améliorent le support de la concurrence via les Ractors.

Plusieurs implémentations alternatives enrichissent l'écosystème. JRuby s'exécute sur la machine virtuelle Java, TruffleRuby offre des performances élevées grâce à GraalVM. Matz dirige le développement de mruby, une version légère conçue pour l'embarqué. Ces variations élargissent les possibilités d'utilisation du langage.

SSH

En 1995, à l'Université d'Helsinki, Tatu Ylönen venait de subir une attaque sur le réseau de son établissement. Des mots de passe circulaient en clair, interceptés par des attaquants qui exploitaient les failles béantes de protocoles comme Telnet, rlogin ou rsh. Face à cette menace concrète,

le chercheur finlandais décida de concevoir une solution radicalement différente : un protocole capable de chiffrer les communications et d'authentifier solidement les parties en présence. SSH, pour Secure Shell, naissait de cette nécessité immédiate de protéger les échanges sur Internet.

La première version sortit en juillet 1995 sous forme gratuite. Le succès fut immédiat dans les cercles universitaires et techniques. Chacun mesurait l'urgence de sécuriser les connexions distantes, et SSH répondait à ce besoin. Devant cet engouement, Ylönen fonda SSH Communications Security en décembre 1995. L'entreprise commercialisa des versions professionnelles tout en maintenant une version libre pour les usages non commerciaux, un équilibre délicat entre logique économique et diffusion large du protocole.

L'année suivante marqua un tournant avec SSH-2, refonte complète de la version initiale. Cette nouvelle mouture adopta une architecture à trois couches : le protocole de transport gérait le chiffrement et l'authentification du serveur, celui d'authentification vérifiait l'identité de l'utilisateur, et le protocole de connexion multipliait les canaux de communication dans un tunnel sécurisé unique. Cette modularité apporta flexibilité et robustesse accrues.

L'IETF lança en 1997 un groupe de travail dédié à la standardisation de SSH. Entre 2004 et 2006, plusieurs RFC précisèrent les spécifications de SSH-2. Cette normalisation favorisa l'interopérabilité entre implémentations et consacra SSH comme standard de fait pour l'administration système distante. Mais un événement allait bouleverser l'écosystème en 1999 : l'apparition d'OpenSSH, implémentation libre développée dans le projet OpenBSD. Cette alternative gratuite et ouverte accéléra l'adoption universelle du protocole. OpenSSH devint vite la référence, intégrée par défaut dans la plupart des systèmes UNIX et Linux.

Sur le plan technique, SSH repose sur une approche hybride mêlant cryptographie asymétrique et symétrique. Au moment d'établir la connexion, des mécanismes à clé publique authentifient et échangent les clés. Une fois la liaison établie, le chiffrement symétrique prend le relais pour protéger les données avec de meilleures performances. Le protocole introduisit le mécanisme des « known hosts » : les clients mémorisent les clés publiques des serveurs contactés, détectant ainsi toute tentative d'attaque par homme du milieu. SSH permit aussi de tunneliser d'autres protocoles TCP/IP via le transfert de port, étendant considérablement son champ d'application.

La souplesse de SSH-2 autorisa le support de méthodes d'authentification variées : mots de passe, clés publiques, authentification par hôte. Cette adaptabilité permit de calibrer la sécurité selon les contextes. Compression des données et multiplexage de canaux vinrent optimiser l'usage de la bande passante, rendant le protocole encore plus efficient.

Dans les années 2000, SSH s'imposa comme brique clé de l'infrastructure Internet. Son usage déborda largement l'accès distant pour englober le transfert sécurisé de fichiers avec SFTP, la gestion de systèmes entiers, le déploiement d'applications. Les entreprises l'adoptèrent massivement pour sécuriser leurs opérations d'administration. Le protocole continua d'évoluer, intégrant des algorithmes cryptographiques plus récents comme AES ou les courbes elliptiques. La communauté de développeurs resta mobilisée pour adapter SSH aux menaces nouvelles et aux cas d'usage émergents.

L'empreinte de SSH sur l'informatique moderne reste considérable. En tant que solution pratique et sûre pour l'administration distante, le protocole a accompagné la croissance d'Internet et l'explosion du cloud computing. Sa conception modulaire a inspiré d'autres protocoles de sécurité. SSH demeure omniprésent dans les environnements de développement, les centres de données, les services cloud. Il illustre comment une réponse technique née d'un incident précis peut devenir un standard international incontournable.

Le succès durable de SSH tient à sa capacité à allier sécurité robuste et simplicité d'usage. En automatisant la complexité cryptographique tout en restant accessible, le protocole a démocratisé les bonnes pratiques de sécurité dans l'administration système.

Lynx

Quand on évoque les navigateurs web, l'image qui vient spontanément à l'esprit est celle d'une interface graphique colorée, animée, interactive. Pourtant, il existe depuis 1992 un programme qui défie cette représentation : Lynx, un navigateur entièrement textuel né à l'Université du Kansas. Son histoire révèle comment l'accessibilité et la sobriété peuvent traverser trois décennies sans prendre une ride.

Au départ, Lynx n'avait rien d'un navigateur web. Les développeurs du Kansas cherchaient à construire une interface pour consulter leur système d'information local. L'hypertexte vint s'y greffer progressivement, puis le protocole Gopher fit son apparition. Ce n'est qu'après ces ajouts successifs que Lynx apprit à parler le langage du Web. Cette genèse par strates explique d'ailleurs la structure du code : une mosaïque de morceaux assemblés au fil du temps, sans plan d'ensemble initial. Certains y verraient un défaut ; d'autres, la marque d'une adaptation organique aux besoins réels.

Le nom du programme renvoie au lynx, ce félin réputé pour sa vision

perçante. L'analogie n'est pas anodine : là où les autres navigateurs multipliaient les ornements visuels, Lynx allait droit au but. Les premiers utilisateurs découvraient un Internet brut, dépouillé, mais fonctionnel. Cette nudité assumée se révéla un atout majeur pour les personnes aveugles ou malvoyantes, qui pouvaient enfin accéder au web via des lecteurs d'écran.

La navigation dans Lynx repose sur le clavier. Les flèches déplacent le curseur entre les liens, la touche Entrée active celui qui est sélectionné, quelques raccourcis donnent accès aux fonctions avancées. Rien de plus. Cette économie de moyens cache néanmoins une vraie sophistication technique. Le programme sait gérer les formulaires HTML, comprend les cookies, et offre une configuration poussée via le fichier .lynxrc. On peut personnaliser les couleurs (si, si, même en mode texte), définir ses préférences, ou brancher son éditeur de texte préféré pour modifier une page directement.

Le traitement des images illustre bien la philosophie du projet. Plutôt que de les ignorer purement et simplement, Lynx affiche leur texte alternatif – cet attribut « alt » que les développeurs web oublient trop souvent. Cette approche, née d'une contrainte technique, est devenue une référence en matière d'accessibilité. Elle force à se demander : que voit quelqu'un qui ne voit pas ?

Dans le web actuel, saturé de JavaScript et de feuilles de style complexes, Lynx garde son utilité. Les administrateurs système qui travaillent en ligne de commande l'utilisent régulièrement. Les connexions à faible débit y trouvent leur compte. Et puis il y a ceux qui préfèrent lire sans être distraits par des animations ou des publicités qui clignotent. Il faut dire que le mode lecture des navigateurs d'aujourd'hui joue parfaitement ce rôle aussi.

Les développeurs web se servent encore de Lynx comme d'un révélateur. Afficher une page dans ce navigateur, c'est voir son squelette HTML à nu. Si le contenu est compréhensible dans ces conditions, c'est bon signe. Cette méthode s'inscrit dans ce qu'on appelle l'amélioration progressive : partir d'une base fonctionnelle simple, et enrichir l'expérience pour ceux qui disposent de moyens techniques plus avancés.

Que Lynx ait survécu aussi longtemps n'est pas un hasard. Son code source ouvert et sa communauté de développeurs ont su le faire évoluer sans trahir ses principes fondateurs. Les mises à jour maintiennent la compatibilité avec les standards modernes, tandis que l'interface reste fidèle à l'esprit d'origine. Peu de logiciels peuvent se targuer d'une telle constance sur trois décennies.

MySQL

En 1979, Michael « Monty » Widenius travaille chez TcX, une petite entreprise, quand il développe un outil de reporting en BASIC. Les contraintes matérielles de l'époque sont sévères : 16 Ko de RAM, un processeur à 4 MHz. Dans cet environnement spartiate, Widenius apprend à écrire du code léger et rapide. Cette expérience forge son approche du développement logiciel. Sa position de copropriétaire chez TcX lui donne une liberté rare pour un programmeur : celle de contrôler son code. Cette conjonction entre compétence technique, propriété intellectuelle et autonomie va se révéler déterminante.

Durant les années 1990, les clients de TcX réclament une interface SQL pour manipuler leurs données. Widenius explore plusieurs pistes : acheter une licence de base de données commerciale, intégrer le code de mSQL... Aucune ne le satisfait vraiment. Il décide de créer sa propre solution. En mai 1996, MySQL version 1.0 sort, mais reste confinée à un cercle restreint. La première version publique (3.11.1) arrive en octobre 1996, d'abord uniquement pour Solaris. Un mois plus tard, Linux est supporté.

MySQL s'étend progressivement à d'autres systèmes d'exploitation. Ses fonctionnalités s'enrichissent au fil des versions. Le modèle de distribution repose sur une licence particulière : l'usage commercial est gratuit tant qu'on ne redistribue pas le logiciel avec ses propres produits. TcX vend aussi du support technique. Ces revenus financent le développement.

La version 3.22 de MySQL propose déjà une part substantielle du langage SQL. Son optimiseur impressionne, surtout pour un projet piloté majoritairement par un seul homme. Le système brille par sa rapidité et sa stabilité. Les API se multiplient, le rendant utilisable depuis presque tous les langages de programmation. Mais des lacunes demeurent : pas de transactions, pas de sous-requêtes, pas de clés étrangères, pas de procédures stockées ni de vues. Le verrouillage au niveau des tables ralentit parfois considérablement les opérations.

Vers 1999-2000, MySQL AB voit le jour en tant qu'entreprise distincte. L'équipe s'étoffe avec le recrutement de développeurs. Un partenariat avec Sleepycat se noue pour créer une interface SQL vers les fichiers Berkeley DB. L'objectif est de doter MySQL de capacités transactionnelles. La version 3.23 résulte de ces travaux.

L'intégration de Berkeley DB ne se déroule pas comme prévu. La stabilité n'est jamais vraiment au rendez-vous. Mais cet effort n'est pas vain : le code source de MySQL est maintenant équipé pour accueillir différents types de moteurs de stockage. En avril 2000, avec l'appui de Slashdot, la

réplication maître-esclave fait son entrée. L'ancien moteur ISAM est remanié pour devenir MyISAM, qui apporte des améliorations dont la recherche en texte intégral.

Heikki Tuuri propose à la même période d'intégrer InnoDB, son propre moteur de stockage aux fonctionnalités similaires. L'interface de gestion des tables, née des travaux sur Berkeley DB, facilite cette intégration. MySQL version 4.0, combiné à InnoDB, est déclarée stable en mars 2003.

Mais la vraie innovation de la 4.0 est le cache des requêtes. Cette fonctionnalité améliore drastiquement les performances de nombreuses applications. Le code de réplication sur l'esclave est réécrit avec deux threads : un pour les entrées-sorties réseau depuis le maître, l'autre pour traiter les mises à jour. L'optimiseur gagne en efficacité. Le protocole client/serveur devient compatible SSL.

Pendant le développement de la 4.1, les travaux sur la branche 5.0 avancent. Cette version apporte les procédures stockées, les curseurs côté serveur, les déclencheurs, les vues, les transactions XA et des améliorations majeures de l'optimiseur. Pourquoi créer une branche séparée ? Les développeurs veulent éviter que l'ajout des procédures stockées ne retarde la stabilisation de la 4.1. La version 5.0 sort en alpha en décembre 2003. Deux branches en alpha coexistent temporairement, créant une certaine confusion.

Entre 2005 et 2006, Oracle rachète successivement Innobase puis Sleepycat, les deux fournisseurs de moteurs de stockage transactionnels de MySQL. Ces acquisitions suscitent des inquiétudes dans la communauté. Larry Ellison, le patron d'Oracle, commente avec désinvolture qu'il a parlé à presque toutes les entreprises du secteur. Il qualifie MySQL de société minuscule dont le chiffre d'affaires oscille entre 30 et 40 millions de dollars, à comparer aux 15 milliards de revenus annuels d'Oracle.

La trajectoire commerciale de MySQL AB suit pourtant une courbe ascendante. En 2001, Mårten Mickos prend la direction de l'entreprise et transforme une startup technique en une entreprise viable. Les revenus grimpent de 6,5 millions de dollars en 2002 à 75 millions en 2007. Le modèle économique repose d'abord sur la double licence pour les intégrateurs OEM, puis bascule vers les abonnements de support pour les utilisateurs finaux avec le lancement du MySQL Network en 2005. L'entreprise compte 8 millions d'installations actives en 2006, emploie 320 personnes réparties dans 25 pays dont 70% travaillent depuis chez elles. Mickos annonce cette année-là préparer une entrée en bourse pour 2008 avec un objectif de 100 millions de revenus.

Mais c'est Sun Microsystems qui frappe le premier. En janvier 2008, l'entreprise californienne annonce l'acquisition de MySQL AB pour envi-

ron 1 milliard de dollars. Jonathan Schwartz, le directeur général de Sun, qualifie cette opération de la plus importante de l'histoire de sa société. Il y voit une occasion de se repositionner au cœur de l'économie du web. Les chiffres parlent d'eux-mêmes : 75% des installations MySQL tournent sur du matériel d'autres fabricants que Sun, 80% utilisent Linux plutôt que Solaris. L'acquisition donne à Sun accès à une immense base installée auprès de laquelle vendre serveurs et logiciels. Avec ce milliard de dollars, MySQL établit un nouveau standard de valorisation pour les sociétés de logiciels libres.

L'intégration chez Sun se révèle chaotique. Widenius et Axmark, deux des trois fondateurs, quittent l'entreprise peu après le rachat. En 2009, Mickos démissionne à son tour pour devenir entrepreneur en résidence chez Benchmark Capital. Sun perd en quelques mois les leaders techniques et commerciaux qui avaient bâti le succès de MySQL.

La même année, Oracle annonce l'acquisition de Sun pour 7,4 milliards de dollars. MySQL devient ainsi propriété du principal concurrent commercial dans le domaine des bases de données. Cette perspective soulève de nombreuses interrogations. Oracle va-t-il continuer à développer MySQL ou chercher à l'étouffer pour protéger sa propre base de données ? Les entreprises qui dépendent de MySQL peuvent-elles encore lui faire confiance ? La Commission européenne examine l'opération de près avant de donner son feu vert, non sans demander des garanties sur le maintien du développement de MySQL.

Widenius lui-même ne reste pas inactif. Dès 2009, il démarre le développement de MariaDB, un *fork* de MySQL. Cette nouvelle base de données conserve une compatibilité totale avec MySQL tout en ajoutant ses propres améliorations. La fondation MariaDB à but non lucratif est créée en 2012. Le nom vient cette fois de Maria, l'autre fille du fondateur. Ce fork devient progressivement l'alternative de référence pour ceux qui se méfient d'Oracle. Wikipedia bascule vers MariaDB, Google aussi. Les distributions Linux commencent à remplacer MySQL par MariaDB dans leurs dépôts par défaut.

MySQL continue néanmoins son évolution sous la gouvernance d'Oracle. L'équipe reste en grande partie la même qu'avant le rachat. Des fonctionnalités NoSQL font leur apparition. Oracle maintient MySQL sous double licence GPL et commerciale.

RealPlayer

En avril 1995, Internet perce dans les foyers du monde entier. Les modems 56k se déploient et peinent à transmettre des fichiers .wav, et personne n'imagine encore qu'on puisse écouter de la musique ou regarder une vidéo sans attendre qu'elle se télécharge entièrement. Rob Glaser, qui vient de quitter Microsoft après dix ans passés sur les applications et le multimédia, voit les choses autrement. Il fonde RealNetworks avec une idée qui paraît presque folle à l'époque : diffuser du son directement depuis Internet.

Le premier RealAudio Player chamboule les usages dès sa sortie. Pour la première fois, on peut écouter des contenus audio en streaming, un mot qui n'appartient pas encore au vocabulaire courant. Le succès arrive vite : trois ans plus tard, Microsoft intègre le lecteur dans Windows 98. On mesure mieux cette prouesse quand on réalise que YouTube ne verra le jour que dix ans après.

Les équipes de RealNetworks travaillent sur deux fronts. D'un côté, elles développent des techniques de compression qui permettent d'obtenir une qualité décente malgré les débits ridicules. De l'autre, elles inventent une façon originale de combiner les protocoles Internet avec et sans connexion pour fluidifier la diffusion. C'est du bricolage savant, mais ça marche.

L'entreprise choisit une stratégie qui deviendra courante plus tard : un lecteur gratuit pour les utilisateurs, des serveurs payants pour ceux qui veulent diffuser. Jim Breyer, du fonds Accel Partners, rejoint le conseil d'administration et les investisseurs affluent. En 1996, contre l'avis d'une partie des équipes, RealNetworks lance une version premium du lecteur. Le pari rapporte gros : plus de trois millions d'exemplaires partent, la majorité vendus en ligne.

La vidéo arrive en 1997 avec le RealSystem. Les connexions restent lentes, mais Glaser anticipe leur amélioration. Pour prouver que sa technologie tient la route, il s'associe avec Spike Lee qui réalise les premiers courts-métrages pensés pour Internet. Plus de 500 000 personnes les regardent en un mois. Le message passe.

Le logiciel change plusieurs fois de nom au fil des ans – RealAudio, RealOne Player, RealPlayer G2 – chaque version apportant son lot de nouveautés. Conversion de formats, téléchargement de vidéos, playlists, extraction audio... Les fonctionnalités s'empilent. En 2015, l'entreprise tente autre chose avec RealTimes, une application mobile pour créer des montages vidéo à partir de photos. C'est une manière de suivre les nouveaux usages, tout en ayant le cœur de métier ailleurs.

Glaser garde de son passage chez Microsoft une certaine vision de la convergence entre technologie et médias. Il pousse sa société vers l'inno-

vation permanente, quitte à prendre des risques. En 2002, RealNetworks lance Helix, une plateforme capable de diffuser n'importe quel format audio ou vidéo. La société va jusqu'à ouvrir une partie de son code source aux développeurs, histoire de favoriser l'amélioration collective des solutions.

RealPlayer a transformé notre rapport aux médias en ligne, nous faisant passer du téléchargement fastidieux au streaming instantané. Les plateformes actuelles lui doivent beaucoup, bien que peu de gens s'en souviennent. L'entreprise a tenu vingt-cinq ans en s'adaptant sans cesse.

Sun UltraSPARC

Les recherches menées à Berkeley entre 1980 et 1984 sur les architectures RISC, conduites en parallèle de celles de Stanford sur MIPS, débouchèrent sur la création de Scalable Processor ARChitecture (SPARC) par Sun Microsystems en 1985. Quatre ans plus tard, Sun fonda SPARC International pour ouvrir cette architecture à d'autres constructeurs.

La première version SPARC V7 sortit en 1986. La V8 arriva en 1990 avec le support matériel des multiplications et divisions, une MMU et des opérations en virgule flottante sur 128 bits. La V9 franchit un cap : elle gérait les registres et opérations 64 bits, repensait la gestion des exceptions et intégrait le préchargement.

C'est en 1995 que Sun lança le premier UltraSPARC, premier processeur à intégrer SPARC V9. Marc Tremblay participa à sa conception. Ce microprocesseur superscalaire exécutait les instructions dans le désordre et en traitait quatre simultanément. Son pipeline comptait huit étages. Les ingénieurs avaient simplifié l'unité d'exécution par rapport au SuperSPARC pour gagner en fréquence, notamment en modifiant le branchement de l'UAL.

Le processeur comportait 32 registres 64 bits répartis en huit fenêtres, totalisant 144 registres. Cette technique évitait aux fonctions appelées de perdre du temps à sauvegarder puis restaurer les registres. Sept registres d'entrée et trois de sortie donnaient accès à deux UAL et à l'unité de gestion mémoire. Une seule UAL gérait multiplications et divisions.

L'unité virgule flottante se découpait en cinq blocs : additions et soustractions, multiplications, divisions et racines carrées, suivis par deux blocs dédiés aux instructions SIMD du Visual Instruction Set. Trente-deux registres 64 bits servaient ces opérations, dont cinq pour les entrées et trois pour les sorties.

Le cache primaire se divisait en deux parties de 16 Ko, l'une pour les instructions, l'autre pour les données. Un cache secondaire externe unifié devait compléter l'ensemble, avec une capacité entre 512 Ko et 4 Mo, accessible en un cycle. Ce cache utilisait de la SRAM synchrone cadencée à la vitesse du processeur.

Texas Instruments fabriqua UltraSPARC avec le procédé CMOS EPIC-3 en 0,5 µm sur quatre niveaux de métal. Le constructeur renonça au BiCMOS qui n'apportait guère d'avantages à cette finesse de gravure. La puce intégrait 3,8 millions de transistors dans un boîtier PBGA de 521 contacts.

Dix ans plus tard, l'UltraSPARC T1 rompit avec cette logique. Sun y matérialisa son concept de *throughput computing*, qui préférait multiplier cœurs et threads plutôt qu'augmenter les fréquences ou complexifier les pipelines. Dans sa version complète, le T1 comptait huit cœurs exécutant chacun quatre threads, soit 32 threads au total. Sun sacrifia performances en virgule flottante et taille des caches au profit du parallélisme.

Le T1 utilisait un pipeline court à exécution dans l'ordre : fetch, sélection de thread, décodage, exécution, mémoire et write-back. Ses UAL fonctionnaient avec une latence d'un cycle, les multiplicateurs et diviseurs sur plusieurs cycles. Une unique unité virgule flottante se partageait entre les huit cœurs. Chaque thread disposait de son propre buffer d'instructions.

Le processeur basculait entre threads selon plusieurs règles. Par défaut, il alternait entre threads disponibles selon une politique LRU. Les instructions longues déclenchaient un changement de thread pour maintenir le pipeline actif. Le cache L1 était traversant, le L2 incluait le L1. Le T1 se contentait volontairement d'un cache réduit : le grand nombre de threads masquait les latences mémoire.

L'UltraSPARC T1 embarquait un hyperviseur qui ajoutait un niveau de privilège au-dessus des modes utilisateur et superviseur. Cette couche logicielle légère offrait une interface de virtualisation complète aux systèmes invités. Sun documenta intégralement cette API et publia les sources VHDL du T1 sous licence libre, rebaptisé OpenSPARC T1.

Le T1 ne consommait pas plus de 70 watts, deux fois moins qu'un Xeon ou un Itanium. Cette sobriété comptait pour les centres de données où la climatisation pouvait coûter plus cher que le matériel lui-même.

Cette architecture convenait aux serveurs web, aux applications client-serveur et à certaines bases de données qui exploitaient son parallélisme élevé. Sun admettait que les charges scientifiques, surtout celles utilisant massivement la virgule flottante, n'étaient pas le terrain de prédilection de ces processeurs.

UltraSPARC reflète l'évolution des architectures processeurs, des premiers RISC aux designs massivement multicœurs. Il témoigne des stratégies

explorées pour accroître les performances : complexification des pipelines d'un côté, parallélisation massive de l'autre.

Wiki

En 1994, Ward Cunningham a inventé quelque chose qui allait changer la manière dont nous créons du contenu sur Internet. Son « Wiki Wiki Web » n'était au départ qu'un outil pour faciliter le travail d'équipe chez Tektronix, mais il allait devenir bien plus que ça.

Cunningham avait grandi dans le Midwest américain pendant les années Spoutnik, bricolant des circuits électroniques dès son enfance. Après des études d'ingénierie électrique à Purdue et un master en informatique, il s'était passionné pour la programmation orientée objet. En travaillant avec Smalltalk et Lisp, il avait remarqué que ces langages produisaient du code qu'on pouvait comprendre en le lisant n'importe où, sans avoir besoin de tout le contexte. Cette idée de lisibilité immédiate allait l'influencer.

Le nom wiki vient du hawaïen et veut dire « rapide ». Cunningham l'avait découvert en prenant les navettes Wiki Wiki à l'aéroport d'Honolulu. C'était exactement ce qu'il cherchait : un mot qui traduisait la rapidité avec laquelle on pouvait modifier des pages web. Fini les processus lourds où il fallait écrire, faire relire, corriger, et enfin publier. Avec le wiki, on écrivait, on publiait tout de suite, et les autres pouvaient relire et corriger après. Un vrai bouleversement.

La simplicité technique était au cœur du système. Un simple navigateur web suffisait. Pas besoin de logiciels compliqués ni de connaître des langages de balisage obscurs. La syntaxe restait basique : pour créer un lien vers une autre page, il suffisait d'écrire un mot en CamelCase, avec une majuscule au milieu. Le système créait automatiquement le lien.

Au lancement, Cunningham a rédigé une centaine de pages sur la programmation informatique. Cette astuce donnait l'impression qu'une communauté existait déjà, que d'autres contribuaient. Les visiteurs voyaient comment ça marchait et se sentaient encouragés à participer. Il a invité les développeurs à créer leur page personnelle, les familiarisant petit à petit avec l'édition collaborative.

Les développeurs ont adopté le concept. Dès les premières semaines, des clones du système sont apparus. Le wiki tranchait avec les systèmes de gestion de contenu classiques qui imposaient des validations à tous les étages. Ici, la liberté était totale.

Beaucoup craignaient le vandalisme, les abus inévitables. Mais la communauté s'est autorégulée naturellement. Cunningham l'expliquait simplement : supprimer du contenu inapproprié demandait moins d'efforts que d'en créer. Cette asymétrie favorisait l'émergence d'informations pertinentes. Le système reposait sur une confiance mutuelle entre utilisateurs, construite par la transparence totale des modifications et la possibilité pour chacun de contribuer.

L'impact du wiki a explosé en 2001 avec Wikipédia. Jimmy Wales a repris les principes du Wiki Wiki Web en y ajoutant une licence adaptée et une vision mondiale multilingue. Le wiki devenait un phénomène culturel planétaire.

Des recherches menées à l'Université d'Ulster ont montré l'efficacité pédagogique des wikis. Plus de 80% des étudiants trouvaient la navigation intuitive et appréciaient la liberté d'édition. Sans barrières techniques, ils se concentraient sur le contenu plutôt que sur les outils.

Le wiki a introduit des innovations techniques durables : versionnage des pages, création automatique de liens, recherche plein texte. Ces fonctionnalités sont devenues des standards du travail collaboratif moderne. Il a prouvé qu'une communauté en ligne pouvait s'organiser efficacement sans hiérarchie rigide. Cette philosophie d'auto-organisation a inspiré d'innombrables projets collaboratifs sur Internet.

Trente ans après, le Wiki Wiki Web original tourne toujours. La simplicité et l'ouverture restent des principes durables dans les technologies collaboratives.

Microsoft Windows 95

Le 24 août 1995, à minuit, des files d'attente s'étiraient devant les magasins CompUSA et Best Buy. On n'avait jamais vu ça pour un logiciel. Microsoft orchestrait le lancement de Windows 95 avec une démesure qui fit date : 300 millions de dollars investis dans une campagne publicitaire mondiale, les Rolling Stones et leur « Start Me Up » en bande-son, Jay Leno animant l'événement aux côtés de Bill Gates. L'entreprise de Redmond alla jusqu'à produire une émission d'une heure avec Jennifer Aniston et Matthew Perry, alors au sommet de leur gloire grâce à Friends. Résultat ? Plus de 40 millions d'exemplaires vendus en cinq semaines.

Derrière ce battage médiatique se cachait une refonte radicale de l'expérience informatique. Windows 95 tournait définitivement la page de MS-

DOS et de Windows 3.1. Le bureau affichait désormais des icônes, une barre des tâches parcourait le bas de l'écran, et un bouton Démarrer concentrait l'accès aux programmes. Cette interface transformait l'ordinateur en un outil que n'importe qui pouvait apprivoiser, sans pour autant renoncer aux fonctions avancées qui intéressaient les utilisateurs chevronnés.

L'architecture 32 bits du système imposait un processeur compatible mais libérait des possibilités nouvelles. La mémoire se gérait mieux, les ressources s'allouaient avec plus de finesse. Le multitâche préemptif entrait en scène pour les applications 32 bits, autorisant l'exécution parallèle de plusieurs programmes sans que le système ne s'enlise. Microsoft devait néanmoins résoudre une équation délicate : faire cohabiter les vieux logiciels DOS et Windows 16 bits avec les nouveaux programmes 32 bits. L'API Win32 apportait une réponse en donnant aux développeurs un ensemble cohérent de fonctions, compatible aussi bien avec Windows NT qu'avec Windows CE. Les programmeurs pouvaient ainsi porter leurs applications d'une plateforme à l'autre sans trop de douleur.

La technologie Plug and Play visait à simplifier l'installation des périphériques. Dans les faits, ses débuts chaotiques valurent au système le surnom ironique de « Plug and Pray ». Malgré ces ratés initiaux, le principe d'une détection automatique du matériel s'imposa par la suite comme une évidence. La gestion des fichiers gagnait elle aussi en souplesse : les noms longs, jusqu'à 250 caractères, remplaçaient la contrainte archaïque des 8 caractères héritée du DOS. MacOS bénéficiait de cette liberté, et Windows 95 comblait enfin son retard. Le système de fichiers VFAT préservait la compatibilité avec les anciennes applications, un compromis typique de l'approche Microsoft.

L'interface réseau intégrée et le support natif de TCP/IP arrivaient au moment où Internet décollait vraiment. Microsoft lançait dans la foulée le Microsoft Network (MSN), service en ligne propriétaire censé concurrencer AOL, accessible moyennant un abonnement mensuel. Internet Explorer ne faisait pas partie du système de base mais fut ajouté dans les versions suivantes via le pack Microsoft Plus ! La fonction « Briefcase » permettait de synchroniser des fichiers entre différents ordinateurs, préfigurant les services cloud qui fleuriraient une décennie plus tard. Les profils utilisateurs offraient à chaque membre d'une famille ou d'une entreprise la possibilité de personnaliser son environnement de travail.

Le système embarquait des fonctionnalités réseau évoluées : partage de fichiers *peer-to-peer*, support des protocoles NetBEUI et IPX/SPX, clients pour NetWare et réseaux Microsoft. Ces caractéristiques facilitèrent son adoption en entreprise, où il supplanta peu à peu Windows 3.1 et Windows for Workgroups. La concurrence n'avait pas dit son dernier mot.

Mac OS séduisait par son élégance et ses capacités multimédias, mais restait prisonnier du matériel Apple. OS/2 d'IBM impressionnait techniquement mais souffrait d'une communication défaillante. Linux, système libre encore émergent, attirait les utilisateurs techniques par sa stabilité et sa modularité.

Un ordinateur type de cette époque montait un processeur 486 vou Pentium, un disque dur IDE ou SCSI, un lecteur CD-ROM double vitesse et une carte son Sound Blaster. Ces machines devaient composer avec les exigences de Windows 95 : 4 Mo de mémoire vive au minimum, 8 Mo recommandés, et un espace disque conséquent. Microsoft fit évoluer le système via plusieurs versions (OSR1, OSR2) qui intégrèrent FAT32, l'USB et l'AGP. Windows 95 resta disponible jusqu'en 2001, cédant progressivement du terrain à Windows 98 puis Windows ME.

L'impact du système sur l'industrie informatique fut considérable. Il standardisa l'interface graphique moderne et contribua à faire de l'ordinateur un objet familier dans les foyers. Cette période marqua l'apogée de la domination Microsoft sur le marché grand public. Une interface repensée, une architecture modernisée, un marketing agressif : Windows 95 établit un standard qui influença le développement des systèmes d'exploitation pendant les vingt années qui suivirent. Le système reste dans les mémoires comme celui qui démocratisa l'informatique personnelle.

Microsoft DirectX

Windows 95 posait un problème de taille aux développeurs de jeux vidéo. La plupart des titres tournaient encore sous DOS, qui laissait les applications accéder directement au matériel. Avec Windows et sa gestion sécurisée de la mémoire, les performances graphiques s'effondraient. Microsoft risquait de perdre les développeurs si rien ne changeait.

La solution vint d'une petite entreprise, RenderMorphics, dont Microsoft racheta le code source. Ce fut le point de départ de DirectX. Les premières versions n'impressionnèrent personne : bugs à répétition, interface de programmation d'une complexité rebutante. Le système s'appuyait sur des *execute buffers*, sortes de paquets contenant les listes de sommets à afficher et leurs instructions. Les développeurs détestaient cette approche.

Tout bascula avec DirectX 5. L'interface DrawPrimitive simplifia radicalement la programmation sans sacrifier les performances. Microsoft venait de créer une option crédible pour développer des jeux sous Windows.

Le marché vidéoludique explosait, et la bataille entre DirectX et OpenGL commença pour de bon.

John Carmack, l'architecte principal d'id Software, ne mâcha pas ses mots en décembre 1996. Il jugea DirectX « terriblement défectueux », une API qui imposait des contraintes absurdes aux programmeurs sans rien apporter en retour. D'autres développeurs reconnus bombardèrent Microsoft de lettres, tous préconisaient l'adoption d'OpenGL.

Microsoft organisa des démonstrations comparatives pour prouver la supériorité de DirectX. Ces tests furent contestés : les méthodes de comparaison semblaient douteuses. Les débats tournaient souvent autour des performances en rendu logiciel, un aspect qui perdrait vite de son importance avec l'arrivée de cartes graphiques puissantes et abordables.

DirectX s'imposa malgré les polémiques. L'API intégrait des outils complets pour gérer le son et les périphériques d'entrée. Le marché Apple pesait peu, ce qui réduisait l'intérêt de la compatibilité multiplateforme. DirectX savait aussi interroger les capacités du matériel graphique, un atout appréciable.

L'architecture technique s'appuyait sur COM (Component Object Model), ce qui permettait de l'utiliser depuis C++, C# ou Visual Basic. Le C pur restait possible, au prix d'une complexité accrue. DirectX progressa de concert avec OpenGL dans le support des *shaders* personnalisés, permettant aux programmeurs un contrôle fin sur le pipeline graphique.

DirectX 8 abandonna DirectDraw, l'API 2D qui avait longtemps constitué un pilier du système. Trois ans plus tard, DirectX 10 débarqua avec Windows Vista après un développement mené en collaboration avec les créateurs d'applications et les fabricants de matériel.

L'apport majeur de DirectX 10 fut le *geometry shader*, une nouvelle étape programmable dans le pipeline graphique. Placé entre le traitement des sommets et la rastérisation, il pouvait générer jusqu'à 1 024 primitives à partir d'une seule entrée. Les concepteurs maintinrent certaines étapes en fonctionnalité fixe, préférant les performances à une flexibilité totale.

DirectX 10 standardisa le traitement des différentes étapes programmables via un « cœur commun ». Cette machine virtuelle unifia la gestion des registres d'entrée-sortie, des registres temporaires et des points de liaison des ressources. Elle utilisait un langage assembleur 32 bits, mais Microsoft encourageait l'emploi du langage de haut niveau HLSL (High Level Shading Language) pour programmer les shaders.

La technologie n'a cessé d'évoluer depuis. Microsoft voulait garder Windows attractif pour les créateurs de jeux, quitte à essuyer des critiques sur ses pratiques commerciales et sa stratégie de verrouillage. DirectX est devenu un standard incontournable de l'industrie vidéoludique.

Une API contestée à ses débuts s'est muée en solution de référence grâce à des améliorations techniques bien pensées et une compréhension aiguë des besoins des développeurs. Les choix techniques et stratégiques pèsent dans l'évolution des technologies, se mêlant aux enjeux commerciaux et aux dynamiques communautaires.

ICQ

En 1996, quatre développeurs israéliens bricolent un système de messagerie qui va changer la façon dont les gens communiquent en ligne. Yair Goldfinger, Sefi Vigiser, Amnon Amir et Arik Vardi, avec Yossi Vardi, le père de ce dernier, ne cherchaient pas à créer un tel outil. Leur projet initial visait simplement à rendre plus pratique la consultation des messages sur les beepers. ICQ est né presque par hasard, comme un outil annexe pour faciliter leur propre travail collaboratif.

À cette époque, rester connecté en permanence relevait du luxe. Les modems occupaient la ligne téléphonique, et maintenir une présence en ligne sur IRC pour dialoguer n'était pas toujours envisageable. L'équipe se heurtait aux difficultés de communication lors du travail à distance. D'où l'idée d'ICQ, dont le nom dit tout : *I Seek You*, je te cherche. La fonction première du logiciel était de permettre aux utilisateurs de se retrouver en ligne.

Deux mois ont suffi pour développer le programme. Chaque utilisateur recevait un numéro d'identification unique, l'UIN, comparable à un numéro de téléphone. Les numéros courts signalaient une inscription ancienne, créant une sorte de prestige social chez les premiers utilisateurs. On pouvait voir qui était connecté, échanger des messages en temps réel, rechercher d'autres utilisateurs via un annuaire intégré.

Le logiciel s'est répandu tout seul, sans campagne marketing. Les communautés de jeux en ligne ont été les premières conquises. Les joueurs de MUD, de Meridian 59 ou de Diablo sur Battle.net adoptaient massivement ICQ pour coordonner leurs parties et garder le contact entre deux sessions. Cette adoption spontanée annonçait le rôle que tiendrait Discord bien plus tard dans l'univers vidéoludique.

Le succès attire America Online, qui rachète Mirabilis en juin 1998 pour 407 millions de dollars. Sous AOL, ICQ continue son expansion en gardant son identité et son équipe basée à Tel Aviv. En 2001, le service comptait 100 millions d'inscrits, en majorité des jeunes entre 13 et 29 ans qui passaient cinq heures par jour connectés.

Les versions se succèdent avec leur lot de nouveautés. ICQ intègre le chiffrement à partir de la version 99b, alors que les premières moutures en étaient dépourvues. Suivent les appels vocaux, la vidéoconférence, le transfert de fichiers, les messages vers les mobiles, l'ICQphone pour appeler des fixes. En 2013, ICQ est le premier service à proposer des appels vidéo chiffrés de bout en bout avec le protocole ZRTP.

Les réseaux sociaux et la messagerie mobile rebattent les cartes. En 2010, AOL cède ICQ au groupe russe Mail.ru pour 188 millions de dollars. Le service est populaire en Europe de l'Est et en Russie, avec 42 millions d'utilisateurs quotidiens. En 2021, Hong Kong connaît un regain d'intérêt pour ICQ : les téléchargements sont multipliés par 35 en une semaine après les polémiques sur la confidentialité de WhatsApp.

ICQ s'adapte à l'ère mobile avec des applications pour Android, iOS, Windows et macOS. L'intelligence artificielle fait son apparition pour suggérer des réponses rapides et transcrire les messages vocaux. Durant la pandémie de COVID-19, le service déploie des fonctionnalités de visioconférence jusqu'à 30 participants, répondant aux besoins du télétravail.

Ces efforts ne suffisent pas. Face à WhatsApp, Messenger ou Telegram, ICQ ne reconquiert pas sa place. En mai 2024, VK annonce l'arrêt définitif pour le 26 juin, invitant les utilisateurs à migrer vers VK Messenger et VK WorkSpace. Un pionnier disparaît.

ICQ a posé les bases de la messagerie instantanée moderne : statuts de présence, messages en temps réel, listes de contacts, recherche d'utilisateurs. Le service a montré l'importance de la communication asynchrone et personnelle sur Internet, avant l'avènement des réseaux sociaux et leurs messageries. Son influence perdure dans les fonctionnalités de nos applications actuelles.

Par son caractère novateur, ICQ a transformé Internet en espace de communication personnelle et immédiate. Rencontres amoureuses, discussions entre joueurs, conversations professionnelles, échanges familiaux : pendant près de trois décennies, le service a tissé une toile de relations humaines à travers le monde numérique.

KDE

Le 14 octobre 1996, un message apparaît sur Usenet. Matthias Ettrich y annonce le lancement d'un projet baptisé KDE, pour Kool Desktop Environment. Son ambition ? Bâtir un environnement de bureau graphique

complet pour UNIX et Linux. À cette époque, les interfaces sous ces systèmes se résument surtout à des gestionnaires de fenêtres rudimentaires comme Feeble Virtual Window Manager pour le système X Window. L'utilisateur confronté à ces outils spartiate peine à retrouver le confort des bureaux Windows ou Mac.

Ettrich fait un pari technique : s'appuyer sur la bibliothèque Qt. Ce choix déclenche aussitôt des controverses, la licence de Qt heurtant certains puristes du libre. Mais le projet avance. Une première version bêta sort en 1997, puis KDE 1.0 débarque neuf mois plus tard. Une communauté de développeurs se fédère autour de cette initiative.

L'architecture repose sur plusieurs briques. KWin gère l'affichage des fenêtres. Konqueror, qui succède à KFM, sert à la fois de gestionnaire de fichiers et de navigateur web. DCOP, remplacé plus tard par D-Bus, fait communiquer les applications entre elles. Le moteur de rendu KHTML mérite qu'on s'y attarde : Apple s'en inspirera pour créer WebKit, qui équipera Safari et Chrome à ses débuts.

Chaque mouture KDE 2 et KDE 3 enrichissent les fonctionnalités pendant les années 2000. En 2008, KDE 4.0 bouleverse la donne. L'interface et l'architecture technique sont refondues de fond en comble. Plasma, le nouvel environnement de bureau, transforme l'expérience utilisateur. Les applications adoptent Qt 4 et exploitent les possibilités qu'offre cette nouvelle version.

Sur le plan organisationnel, le projet se structure. KDE e.V. voit le jour, association de droit allemand qui gère les aspects financiers et juridiques. Elle collecte les dons, protège la propriété intellectuelle, organise Akademy, la conférence annuelle qui rassemble la communauté. Cette professionnalisation n'entame pas l'esprit collaboratif du projet.

La décennie suivante apporte KDE Frameworks 5, qui découpe les bibliothèques en modules réutilisables. Plasma 5 modernise le bureau tout en préservant sa flexibilité légendaire. Le projet s'adapte aux écrans tactiles avec Plasma Mobile, preuve que KDE sait suivre l'évolution des usages.

Le modèle de développement est résolument ouvert. Les décisions sont prises sans pyramide hiérarchique, par consensus. Des contributeurs du monde entier participent : développeurs bien sûr, mais aussi traducteurs, designers, rédacteurs de documentation. Le Google Summer of Code et d'autres programmes de mentorat attirent régulièrement de nouveaux talents.

Les applications couvrent tous les besoins quotidiens. Calligra pour la bureautique, Amarok pour la musique, KMail pour les courriels, Kate pour l'édition de texte... Ces logiciels rivalisent avec les solutions propriétaires. Certaines trouvailles de KDE, comme la détection automatique des pièces

jointes oubliées dans les e-mails, ont d'ailleurs été reprises ailleurs.

L'influence de KDE dépasse la technique. Le projet prouve qu'on peut créer des applications complexes et innovantes en logiciel libre. Il facilite l'adoption de Linux sur les ordinateurs personnels en proposant une interface accessible aux habitués de Windows. Des distributions comme openSUSE ou Kubuntu l'ont choisi comme environnement par défaut, ce qui dit beaucoup sur sa maturité.

Trente après sa naissance, KDE tient toujours sa ligne : redonner aux utilisateurs le contrôle de leur environnement numérique, dans le respect de leur liberté et de leur vie privée. Le projet innove dans l'accessibilité, la sécurité. Sa longévité tient à sa capacité d'attirer de nouveaux contributeurs sans renier ses valeurs.

Cascading Style Sheets

En 1994, au CERN, berceau du Web, Håkon Wium Lie identifie un problème : le Web manque d'un moyen de styliser les documents. Lui qui sortait du MIT Media Laboratory, où il avait travaillé sur des présentations de journaux personnalisés, voyait bien que cette lacune freinait l'édition électronique.

L'idée de séparer structure et mise en page existait déjà. Tim Berners-Lee avait conçu son navigateur NeXT en 1990 avec une feuille de style rudimentaire. Mais il n'avait rien publié, estimant que chaque navigateur devait trouver sa propre solution d'affichage. D'autres expérimentaient de leur côté : Pei Wei avec son navigateur ViolaWWW en 1992, qui proposait son propre langage de style.

Puis arrive NCSA Mosaic en 1993. Ce navigateur qui va populariser le Web représente paradoxalement un retour en arrière : les utilisateurs ne peuvent modifier que quelques couleurs et polices. Les créateurs de pages web s'en plaignent. Marc Andreessen, l'un des programmeurs de Mosaic, refuse d'abord tout changement avant de changer son fusil d'épaule en cofondant Netscape.

Le 13 octobre 1994, trois jours avant l'annonce du navigateur Netscape, Håkon publie la première version des Cascading HTML Style Sheets. Dave Raggett, architecte principal de HTML 3.0, l'avait poussé à publier avant la conférence « Mosaic and the Web » à Chicago. Raggett avait compris que HTML ne deviendrait jamais un langage de description de page.

Bert Bos rejoint Håkon. Il développait alors Argo, un navigateur hautement personnalisable avec des feuilles de style. Leurs deux propositions initiales diffèrent du CSS actuel mais contiennent déjà les concepts originaux. Une particularité du langage : il reconnaît que le style d'un document web ne peut venir uniquement de l'auteur ou du lecteur. Leurs souhaits doivent se combiner en cascade, en tenant compte des capacités du dispositif d'affichage et du navigateur.

Le World Wide Web Consortium (W3C) est opérationnel en 1995. Un atelier sur les feuilles de style se tient à Paris. Thomas Reardon de Microsoft s'engage à intégrer CSS dans Internet Explorer. Fin 1995, le W3C établit le HTML Editorial Review Board pour valider les futures spécifications HTML. La spécification CSS devient un élément de travail en vue d'une Recommandation W3C.

Internet Explorer 3 sort en août 1996. C'est le premier navigateur commercial à prendre en charge CSS. L'équipe de Chris Wilson chez Microsoft implémente de façon fiable la plupart des propriétés de couleur, d'arrière-plan, de police et de texte. Netscape suit avec Navigator 4.0, mais son implémentation reste limitée. Opera entre dans la course en novembre 1998 avec sa version 3.5, qui offre un support plus complet de CSS1.

CSS obtient son propre groupe de travail au W3C en février 1997. Chris Lilley le préside. Le groupe développe CSS niveau 2, qui sera une recommandation en mai 1998. Les réunions se tiennent généralement par téléphone une heure par semaine, avec environ quatre rencontres annuelles dans différentes villes du monde.

Eric Meyer améliore le processus de test en développant une suite de tests CSS1 avec l'aide de nombreux volontaires. Cette suite facilite considérablement le travail des développeurs. Le Web Standards Project (WaSP) surveille la conformité des navigateurs aux recommandations du W3C.

CSS se développe avec plus de 60 modules définissant différentes capacités. Le langage s'adapte à de nouveaux usages : création de livres électroniques au format EPUB, conception d'interfaces utilisateur graphiques. Des langages dérivés apparaissent. Qt Style Sheets pour les widgets Qt, JavaFX Style Sheets pour JavaFX, MapCSS pour décrire le style des cartes.

En 2008, les polices web connaissent une évolution majeure. CSS niveau 2 intégrait des polices dans un document web, mais seul Internet Explorer implémentait cette fonction. Microsoft et Monotype développent le format propriétaire Embedded OpenType. La situation change avec l'émergence de polices libres distribuables sur le Web, le format WOFF étant la solution adoptée par la communauté.

CSS s'est imposé comme un standard incontournable du Web : moins de cinq pourcent des pages HTML n'utilisent pas de feuilles de style. Des

conférences spécialisées et de nombreux ouvrages témoignent de la vitalité de cette technologie qui continue d'évoluer pour répondre aux besoins du Web moderne.

Flash

Jonathan Gay est un lycéen qui programme ses premières applications graphiques sur Apple II en 1985. Ses expérimentations avec le langage Pascal l'amènent à collaborer avec Charlie Jackson, fondateur de Silicon Beach Software. Ensemble, ils développent Airborne !, un jeu pour Macintosh qui combine animation et son numérique.

Il poursuit sa route en créant SmartSketch, un logiciel de dessin fonctionnant aussi bien sur Macintosh que sous Windows. L'utilisateur peut dessiner à l'écran avec un stylet électronique. Cette innovation donne naissance à FutureWave Software au milieu des années 1990. Quand Internet se développe, l'entreprise transforme SmartSketch en outil d'animation bidimensionnelle capable d'afficher des graphismes animés sur le Web. FutureSplash Animator naît en 1995.

Microsoft adopte le logiciel pour son site MSN. Le succès est immédiat. En 1996, Macromedia rachète FutureWave et rebaptise le programme Flash. S'ouvre alors une période d'innovations intenses : audio, vidéo, interactivité transforment la création et la diffusion de contenus sur Internet.

La fin des années 1990 et le début des années 2000 marquent l'apogée de Flash. La technologie s'impose pour les jeux en ligne, les sites web interactifs et les animations. En 2000, ActionScript fait son apparition. Ce langage de programmation orienté objet offre aux développeurs la possibilité de créer des applications web sophistiquées. Des millions de sites intègrent des éléments Flash, du simple bouton animé aux environnements immersifs complets.

Les atouts techniques expliquent ce succès. Le format vectoriel garantit une qualité graphique constante quelle que soit la taille d'affichage. Le plug-in, léger (moins d'1 Mo), s'installe sans difficulté. La distribution atteint 98% des ordinateurs connectés à Internet. Le modèle de sécurité « sandbox » protège les utilisateurs en isolant les contenus Flash, et des éditeurs tiers peuvent créer des outils compatibles à la suite de la publication du format SWF.

Adobe Systems acquiert Macromedia en 2005 et poursuit le développement. Mais arrivent les smartphones et les tablettes. En 2010, Steve Jobs

publie une lettre qui critique vertement Flash : sécurité insuffisante, instabilité, consommation excessive de batterie. Apple refuse d'intégrer la technologie dans iOS. Le déclin s'amorce.

HTML5 accélère le mouvement. Ce nouveau standard propose nativement dans les navigateurs web ce qui relevait auparavant de Flash : animation, audio, vidéo, interactions avancées. Les développeurs migrent vers ces technologies ouvertes, standardisées et plus performantes.

Adobe s'adapte. L'entreprise réoriente Flash vers la création d'animations HTML5 et le rebaptise Adobe Animate. En 2017, Adobe annonce l'arrêt du support pour fin 2020. Vingt-cinq années d'une technologie qui a façonné l'expérience web d'une génération s'achèvent ainsi.

Flash laisse pourtant des traces dans le développement web moderne. L'interactivité et l'expérience utilisateur qu'il a popularisées restent d'actualité. L'animation fluide, la synchronisation audio-vidéo, les interfaces dynamiques : ces concepts sont désormais intégrés aux standards du Web. Ils témoignent de l'impact durable de Flash sur l'évolution d'Internet.

Cette trajectoire illustre les mutations rapides des technologies numériques. Elle montre l'importance de s'adapter aux nouveaux usages et standards. Le passage de Flash au HTML5 symbolise la transition vers un Web plus ouvert, sécurisé et économe en ressources. Une technologie ne perdure que si elle répond aux besoins changeants des utilisateurs et aux exigences techniques de son époque.

OpenBSD

L'histoire d'OpenBSD prend racine dans l'épopée d'UNIX et de l'Internet. Quand IBM et Digital dominaient l'informatique d'entreprise dans les années 1970, le département de la Défense américain finançait des recherches sur les réseaux distribués. À l'université de Berkeley, un groupe de développeurs coordonné par des chercheurs créait BSD UNIX, qui allait devenir la référence pour la mise en œuvre des protocoles TCP/IP de l'Internet.

Au début des années 1990, le Berkeley Computer Science Research Group, qui coordonnait le développement de BSD, s'apprêtait à être dissous. BSD avait commencé comme une collection de logiciels pour UNIX d'AT&T, et s'était transformé au fil des ans en un système d'exploitation complet et des groupes souhaitaient poursuivre son développement. Lynne et Bill Jolitz adaptèrent le système aux processeurs Intel x86 avec 386BSD.

Des développeurs commencèrent à partager des améliorations sous forme de « patchkits », donnant naissance à deux projets distincts : FreeBSD, orienté vers l'optimisation pour le matériel PC, et NetBSD, visant la portabilité maximale.

En octobre 1995, Theo de Raadt scinda NetBSD pour créer OpenBSD. La première version fut publiée en juillet 1996. Cette nouvelle distribution se distinguait par son engagement radical envers la sécurité et l'élégance de sa conception. L'équipe lança un audit complet du code source, recherchant et corrigeant systématiquement les failles de sécurité. Cette démarche produisit un système réputé pour sa fiabilité exceptionnelle, avec seulement deux vulnérabilités exploitables à distance en installation par défaut depuis sa création.

OpenBSD innova en intégrant des mécanismes de sécurité préventifs. Le système adopta le principe `W^X` (Write XOR Execute) où la mémoire ne peut être simultanément inscriptible et exécutable. Il introduisit la randomisation de l'espace d'adressage (ASLR) pour varier les cibles de saut et les écarts entre les zones mémoire à chaque exécution. Des pages de garde illisibles et non inscriptibles furent placées en fin des blocs mémoire alloués pour détecter les dépassements. Les développeurs mirent en œuvre la séparation des privilèges, où les démons s'exécutent avec des droits minimaux dans un environnement restreint.

Le projet se distingua en publiant son code source en temps réel via CVS anonyme, une pratique innovante à l'époque où la plupart des projets travaillaient en vase clos. Cette transparence permit à la communauté de vérifier et d'améliorer continuellement la sécurité du système. OpenBSD fut précurseur dans l'utilisation de la cryptographie forte, devenant en 1997 le premier système libre à intégrer IPSec par défaut. Cette décision entraîna des complications avec les restrictions américaines sur l'exportation des technologies cryptographiques, contournées grâce à la coordination du projet depuis le Canada.

Au-delà de son système d'exploitation, OpenBSD devint la source de nombreux composants largement utilisés. OpenSSH, créé pour remplacer les solutions non libres de connexion sécurisée, s'imposa comme la référence mondiale. Le pare-feu PF (Packet Filter), LibreSSL, le serveur de messagerie OpenSMTPd et d'autres outils issus du projet furent adoptés par de nombreux systèmes. Ces logiciels se retrouvent aujourd'hui dans la plupart des appareils connectés, des routeurs Cisco aux produits Apple, en passant par les systèmes Android.

La philosophie d'OpenBSD privilégie la correction et la sécurité du code plutôt que la simplicité d'utilisation immédiate. Le système n'active par défaut qu'un minimum de services, obligeant l'administrateur à configurer

explicitement les fonctionnalités souhaitées. Cette approche, parfois critiquée pour sa rigidité, a démontré son efficacité en prévenant de nombreuses compromissions. Les développeurs s'attachent à maintenir la compatibilité avec le modèle BSD originel tout en renforçant la sécurité dans les limites de ce cadre.

La licence BSD, plus permissive que la GPL, autorise l'utilisation du code dans des produits commerciaux fermés. Cette liberté a favorisé l'adoption d'OpenBSD comme base pour diverses solutions de sécurité réseau. Le système s'est imposé dans les infrastructures critiques nécessitant une sécurité maximale : pare-feux, serveurs en périphérie de réseau, systèmes de détection d'intrusion. Si son utilisation sur les postes de travail est limitée par le support restreint des applications grand public, OpenBSD continue d'influencer l'ensemble de l'industrie par ses innovations en matière de sécurité.

PostgreSQL

En 1986, à l'Université de Californie à Berkeley, le professeur Michael Stonebraker lance le projet POSTGRES. La DARPA, l'Army Research Office et la National Science Foundation financent cette initiative qui fait suite à INGRES, un système relationnel développé quelques années auparavant par la même équipe. L'ambition était de repousser les limites du modèle relationnel dans un contexte où les critiques fusent de toutes parts.

Le modèle relationnel, introduit par Edgar F. Codd, déchaîne les passions dans la communauté informatique. Les traditionalistes martèlent que ce type de système ne peut pas fonctionner correctement et que les langages de requêtes proposés restent trop complexes pour les utilisateurs. L'équipe de Berkeley décide de leur clouer le bec en créant un prototype opérationnel.

La première version de démonstration apparaît en 1987. Les choix techniques sont audacieux : UNIX comme système d'exploitation et le langage C pour la programmation. Ces décisions se révèlent judicieuses, influencées par les échanges avec Ken Thompson, un diplômé de Berkeley. L'équipe n'hésite pas à réécrire des composants entiers du système pour en améliorer la qualité, appliquant une méthode de développement itérative qui s'avère payante.

Les premières années du projet voient une intense activité de recherche et développement. En 1989, la version 1 est distribuée à quelques utilisateurs externes. La version 2 sort en 1990 avec un nouveau système de règles. La

version 3, publiée en 1991, apporte son lot d'améliorations : support de plusieurs gestionnaires de stockage, optimisation du moteur d'exécution des requêtes, refonte du système de règles.

POSTGRES trouve sa place dans des applications variées. On le retrouve dans l'analyse de données financières, le suivi des performances de moteurs d'avion, une base de données de suivi d'astéroïdes, des systèmes d'information médicale et des applications géographiques. Des universités l'adoptent comme outil pédagogique. En 1992, il est le gestionnaire de données principal pour le projet scientifique Sequoia 2000.

La communauté d'utilisateurs double en 1993. Cette croissance génère une charge de support considérable pour l'équipe universitaire. En 1994, deux étudiants, Andrew Yu et Jolly Chen, ajoutent un interpréteur SQL au système. Cette évolution donne naissance à Postgres95, une version plus compacte et performante. Le code se réduit de 25% et les performances s'améliorent de 30 à 50% sur les tests de référence Wisconsin.

En 1996, le projet prend le nom de PostgreSQL. Ce nouveau nom souligne la continuité avec POSTGRES et l'intégration des fonctionnalités SQL. Cette nouvelle incarnation marque le début d'une approche collaborative et open source. Le système se distingue par son extensibilité, qui autorise l'ajout de nouveaux types de données et d'opérateurs sans toucher au noyau.

Le développement de PostgreSQL repose sur une philosophie minimaliste. Le système privilégie l'utilisation d'outils externes pour des fonctions comme la gestion des connexions, la réplication ou la sauvegarde. Cette approche favorise la création d'un écosystème riche de projets complémentaires et l'adaptation aux évolutions technologiques comme le stockage sur disques SSD ou le déploiement dans le cloud.

La structure organisationnelle du projet reflète sa nature collaborative. L'équipe centrale fonctionne avec une charte minimale et aucune entreprise n'est en capacité d'employer la majorité de ses membres. Cette gouvernance distribuée garantit que PostgreSQL reste sous le contrôle de sa communauté d'utilisateurs et de développeurs.

L'émergence du World Wide Web et des applications mobiles apporte de nouvelles contraintes pour les bases de données relationnelles traditionnelles. PostgreSQL s'adapte en intégrant naturellement des types de données non relationnels : documents complexes, données géographiques, informations GPS, contenus pour réseaux sociaux. Cette capacité d'évolution sans refonte constitue un avantage technique de premier ordre.

Les années 2000 voient l'adoption croissante de PostgreSQL par des entreprises et organisations de toutes tailles. Le système est une alternative sérieuse aux bases de données propriétaires d'Oracle, IBM et Microsoft. Son

modèle de développement ouvert autorise une innovation continue et une réactivité aux besoins des utilisateurs que les éditeurs traditionnels peinent à égaler.

La longévité et le succès de PostgreSQL résultent de choix architecturaux initiaux pertinents et d'une gouvernance stable. Le système maintient un équilibre entre innovation technique et stabilité, tout en préservant son indépendance vis-à-vis des intérêts commerciaux. Cette combinaison unique explique sa position actuelle comme système de gestion de bases de données open source le plus avancé.

Les contributions à PostgreSQL proviennent d'horizons variés : chercheurs universitaires, entreprises utilisatrices, fournisseurs de services, développeurs indépendants. Cette diversité nourrit une dynamique d'amélioration continue du code, de la documentation et des outils associés. Le projet maintient des standards élevés de qualité tout en restant accessible aux nouveaux contributeurs.

PostgreSQL continue d'évoluer pour répondre aux exigences modernes : performances à grande échelle, analyse de données complexes, intelligence artificielle. Sa conception extensible lui autorise l'intégration de ces nouvelles fonctionnalités sans compromettre sa cohérence ni sa fiabilité. Cette capacité d'adaptation, associée à sa maturité technique, en fait une technologie structurante de l'informatique contemporaine capable de rivaliser avec les solutions propriétaires historiques.

USB

En 1990, brancher un périphérique sur un ordinateur relevait du parcours du combattant. L'arrière des machines ressemblait à un enchevêtrement de connecteurs hétéroclites et poussiéreux : ports série et parallèle pour les imprimantes, prises PS/2 pour clavier et souris, connecteurs DIN à 5 broches pour les claviers les plus anciens, sans oublier les ports SCSI et les prises de jeu pour les passionnés. Chaque accessoire réclamait son propre type de connexion, avec ses particularités matérielles et ses pilotes dédiés. Un vrai casse-tête pour les utilisateurs.

C'est Intel qui décida de mettre de l'ordre dans ce chaos en lançant un projet dès 1992. L'idée ? Créer un connecteur universel qui remplacerait progressivement tous les autres. Deux ans plus tard, six entreprises s'associèrent pour donner naissance à l'Universal Serial Bus : Intel, Compaq, IBM, Microsoft, LSI et Hewlett-Packard. Apple rejoignit brièvement

le groupe avant de se retirer pour développer FireWire, sa propre technologie. L'ambition était claire : unifier les ports, les prises et les signaux électroniques qui reliaient ordinateurs et périphériques.

La première version vit le jour en 1996, avec un débit de 1,56 Mbps. Modeste, l'innovation tenait surtout dans le principe : un seul type de câble pour transférer données et énergie. Les concepteurs voulaient un connecteur compact et bon marché, capable de faire office de traducteur entre les différentes façons dont les périphériques communiquaient. Les ordinateurs pourraient ainsi gérer simultanément plusieurs accessoires sans que ces derniers n'entrent en conflit.

Intel déposa un brevet pour l'USB en 1997. Mais voilà, les brevets peuvent tuer une norme dans l'œuf. Quand une seule entreprise détient tous les droits, elle peut bloquer les concurrents ou réclamer des royalties prohibitives. Intel l'avait bien compris. La société créa un « pool de brevets », où les droits sont partagés entre plusieurs détenteurs qui se répartissent les redevances. Cette astuce permit aux fabricants de produire librement des appareils compatibles USB sans craindre de poursuites.

Les promoteurs fondèrent l'USB Implementers Forum (USB-IF), qui rassembla plus de 700 entreprises, pour faire adopter la norme par le plus grand nombre, bien au-delà du cercle des fabricants d'ordinateurs. Cette diversité joua un rôle déterminant dans le succès de l'USB.

L'adoption ne fut pas fulgurante. Les entreprises informatiques commencèrent à intégrer sérieusement l'USB vers la fin des années 1990. La technologie progressa vite : en 2000, l'USB 2.0 débarqua pour contrer FireWire, qui affichait des débits de 400 Mbps et permettait des transferts bidirectionnels simultanés. Sur le papier, l'USB 2.0 grimpait à 480 Mbps, mais les tests montraient parfois des performances inférieures à celles de son rival. Pourtant, sa compatibilité étendue et ses frais de licence raisonnables limitèrent l'essor de FireWire. Apple finit par intégrer l'USB à ses machines en 2003, avant d'abandonner complètement FireWire 400 en 2005.

En 2011, sept milliards d'appareils équipés d'USB étaient fabriqués chaque année dans le monde. La norme continua de progresser, atteignant des débits de 40 000 Mbps en 2019. Le port USB devint la référence pour à peu près tout : souris, claviers, imprimantes, disques durs externes. Les variantes Micro-USB et Mini-USB élargirent encore son territoire. En 2019, le marché mondial des périphériques USB pesait 31 milliards de dollars.

La standardisation transforma l'expérience utilisateur. N'importe quel accessoire USB fonctionne sur n'importe quel ordinateur, parmi des milliers de références disponibles. Les fabricants peuvent concevoir leurs produits en sachant exactement comment ils se connecteront aux machines, ce qui réduit les risques lors du développement de nouveaux appareils. Cette nor-

malisation a libéré l'innovation : adaptateurs Wi-Fi, lecteurs optiques, ports Ethernet, dongles de réseau mobile, tous ont pu se développer grâce à cette base commune.

L'aspect environnemental mérite d'être souligné. En rendant les accessoires compatibles avec n'importe quel nouveau matériel, l'USB limite la quantité d'appareils électroniques jetés. Moins de déchets, moins de matières premières extraites, moins d'émissions de CO^2 liées à la fabrication de câbles et d'accessoires propriétaires.

Dans l'univers de l'Internet des Objets, la consommation énergétique est très importante. Le protocole USB prévoit qu'en mode actif, un périphérique dispose d'une intensité limitée et doit répondre immédiatement aux requêtes de l'ordinateur. Si aucune communication n'a lieu pendant quelques millisecondes, l'appareil bascule en suspension et sa consommation chute. Des innovations récentes, comme le mode basse énergie et les oscillateurs sans cristal, ont amélioré l'efficacité énergétique pour les appareils sur batterie.

Le pool de brevets a permis aux fabricants de converger vers une solution commune, y compris entre concurrents. La présence de technologies alternatives, comme les connexions sans fil ou la charge par induction, continue de pousser l'USB à évoluer pour rester pertinent dans un monde toujours plus connecté.

VPN

Les circuits virtuels des années 1970 ont posé les premières pierres de ce qui deviendrait plus tard les réseaux privés virtuels. Ces connexions logiques traversaient les réseaux pour acheminer les messages longs, créant un chemin entre le port source et sa destination. Le concept fonctionnait bien sur les réseaux fortement maillés, mais personne n'imaginait alors qu'il serait un pilier de la sécurité des communications.

L'expansion d'Internet des années 1990 a bouleversé la donne. Les entreprises se sont retrouvées face à un dilemme : comment relier leurs sites distants et leurs employés nomades tout en préservant la confidentialité des échanges ? Les liaisons spécialisées coûtaient une fortune, et Internet, bien que séduisant par son ubiquité, exposait les données à tous les regards. La solution est venue du chiffrement, cette vieille technique remise au goût du jour pour créer des tunnels sécurisés au cœur du réseau public.

En 1996, Gurdeep Singh Pall et son équipe chez Microsoft ont conçu le

PPTP, le Point-to-Point Tunneling Protocol. Le magazine PC Magazine lui a décerné le prix de l'innovation de l'année, et pour cause : il standardisait enfin ce que l'industrie cherchait depuis des années. Le PPTP creusait des tunnels dans Internet, des passages secrets où les données circulaient à l'abri des curieux. C'était la première vraie solution VPN industrielle.

D'autres protocoles ont suivi. L2TP et IPSec ont enrichi la panoplie disponible. IPSec mérite qu'on s'y attarde : en travaillant directement au niveau de la couche réseau, il protégeait les communications de bout en bout avec une robustesse qui en a fait un standard incontournable. Les entreprises tenaient là leurs outils pour bâtir des réseaux privés virtuels dignes de ce nom.

Les années 2000 ont vu les VPN gagner en sophistication. Pare-feu, authentification forte, gestion fine des accès : les solutions se sont étoffées. Le télétravail prenait son essor, et avec lui la nécessité de connecter toujours plus d'utilisateurs distants. Les entreprises ont commencé à abandonner leurs coûteuses liaisons dédiées au profit de connexions VPN sur Internet. Les économies étaient substantielles, la sécurité restait au rendez-vous.

SSL et TLS ont apporté une simplification bienvenue. Fini les logiciels clients complexes à déployer : un simple navigateur web suffisait désormais pour accéder aux applications d'entreprise à distance. OpenVPN est arrivé dans ce contexte comme une alternative libre et flexible aux solutions propriétaires. La technologie SSL/TLS qu'il utilisait a séduit par sa simplicité d'usage et sa compatibilité avec l'existant.

Le cloud et les smartphones ont rebattu les cartes dans les années 2010. Les infrastructures VPN ont dû s'adapter à des volumes de connexions simultanées sans précédent. Les utilisateurs nomades se multipliaient, jonglant entre leurs différents appareils. Les solutions ont intégré de la qualité de service, de l'optimisation des performances, tout ce qu'il fallait pour maintenir l'expérience utilisateur malgré la complexité croissante.

De nos jours, les VPN combinent plusieurs technologies : IPSec pour les sites distants, SSL/TLS pour l'accès web, pare-feu et authentification forte pour la sécurité. Ils interconnectent les bureaux dispersés, protègent les accès aux ressources internes, sécurisent les communications des travailleurs mobiles. Cette convergence technologique répond aux besoins actuels, mais elle n'était pas évidente il y a vingt ans.

L'histoire des VPN reflète finalement celle des usages réseau. Partis de circuits virtuels rudimentaires, ils se sont transformés en véritables forteresses numériques. La mobilité, le cloud, la cybersécurité : autant d'enjeux que la technologie VPN a su accompagner. Elle continue d'ailleurs d'évoluer, intégrant les dernières avancées cryptographiques pour rester pertinente.

Les particuliers se sont eux aussi emparés des VPN, dépassant largement

le cadre professionnel initial. La protection de la vie privée sur Internet est devenue une préoccupation majeure, et les services VPN grand public ont fleuri. Certains les utilisent pour contourner les restrictions géographiques pour des raisons politiques, d'autres pour masquer leur navigation. Cette démocratisation a changé la perception de la technologie, autrefois réservée aux experts réseau.

L'Internet des objets et la 5G vont générer de nouveaux besoins de connectivité sécurisée. Les menaces évoluent, se sophistiquent, et les VPN devront suivre le mouvement. La cryptographie post-quantique pointe à l'horizon, promettant de nouveaux algorithmes pour résister aux ordinateurs de demain. Le VPN n'a pas fini sa mutation.

Hotmail

En 1996, consulter ses e-mails relevait d'une certaine logistique. Il fallait installer un logiciel sur son ordinateur, configurer les paramètres du serveur, et surtout rester lié à cette machine pour accéder à ses messages. Jack Smith et Sabeer Bhatia, deux anciens d'Apple passés par Stanford, trouvaient cette situation absurde. Ils voulaient lire leurs e-mails depuis n'importe où, sans se heurter aux pare-feux tatillons de leur entreprise. De cette frustration banale allait naître Hotmail, le premier service de messagerie utilisable directement dans un navigateur web.

Les deux compères réunissent 4 000 dollars en 1995 pour bâtir un prototype. La démonstration séduit le fonds Draper Fisher Jurvetson, qui injecte 300 000 dollars contre 15% du capital. Mais l'argent file vite. Un mois avant le lancement, les caisses sont vides. Doug Carlisle leur tend une perche : 100 000 dollars d'investissement supplémentaire. Smith et Bhatia refusent, craignant de perdre le contrôle de leur création. Ils choisissent plutôt l'emprunt.

Le 4 juillet 1996, jour de l'indépendance américaine, Hotmail ouvre ses portes au public. Le nom joue sur les lettres HTML, ces quatre caractères qui structurent les pages web, en les glissant en majuscules dans « HoTMaiL ». Le service offre 2 mégaoctets de stockage gratuit. C'est peu, mais personne ne s'en plaint à l'époque. La vraie trouvaille de Bhatia tient dans un détail apparemment anodin : chaque message expédié depuis Hotmail porte en signature « PS : I Love You » suivi d'un lien vers le service. Cette astuce, qu'on baptisera plus tard marketing viral, fait exploser les compteurs. Cent mille inscriptions en moins d'un mois, un million d'utilisateurs en moins de six mois.

Microsoft observe cette montée en puissance. Fin 1997, Hotmail revendique 10 millions d'utilisateurs et s'arroge un quart du marché des webmails. Bill Gates ne laisse pas passer l'occasion. Le 30 décembre 1997, l'affaire se conclut pour 400 millions de dollars. Bhatia hésite d'abord, peu enchanté par l'image de monopole qui colle à Microsoft, mais finit par reconnaître le flair de Gates.

L'intégration dans la galaxie MSN accélère la croissance. Début 1999, MSN Hotmail franchit les 30 millions d'utilisateurs et enregistre 150 000 nouvelles inscriptions chaque jour. Mais cette expansion rapide s'accompagne de déconvenues embarrassantes. En 1999, n'importe qui peut se connecter à n'importe quel compte en tapant simplement le mot de passe « eh ». Microsoft parle d'un « problème de sécurité inconnu » et écarte l'hypothèse d'une porte dérobée délibérée. Deux ans plus tard, nouvelle faille : une simple manipulation d'URL suffit pour lire les messages d'autrui.

Google lance son service Gmail en 2004 avec un gigaoctet de stockage, soit cinq cents fois plus que les 2 mégaoctets de Hotmail. Microsoft réagit en montant à 250 mégaoctets et en autorisant des pièces jointes de 10 mégaoctets. Une refonte complète du service démarre, mais la version finale n'émerge de sa phase bêta qu'en 2007 sous l'appellation Windows Live Hotmail. Ces tergiversations laissent le champ libre à Gmail, tandis que Hotmail commence à faire figure de dinosaure.

Microsoft multiplie ensuite les ajustements : intégration de Bing, compatibilité avec Firefox et Chrome, filtres rapides, nettoyage automatique de la boîte de réception. En 2010, le protocole Exchange ActiveSync débarque, suivi l'année suivante par les alias et les actions instantanées. Malgré ces efforts, le service traîne une réputation écornée. Les utilisateurs avertis le boudent, les spammeurs en font leur terrain de jeu favori.

Juillet 2012 marque un nouveau départ avec Outlook.com. Microsoft abandonne le nom Hotmail et propose une interface épurée qui tranche avec l'ancienne mouture. Le succès est immédiat : plus de 10 millions d'inscriptions volontaires en deux semaines. La migration des comptes Hotmail s'achève en mai 2013. Microsoft annonce alors 400 millions de comptes actifs. Les utilisateurs peuvent garder leur adresse en @hotmail.com ou basculer vers @outlook.com.

En 2019, un pirate exploite une faille de sécurité pour utiliser les identifiants d'un agent du support pour accéder à certains comptes. Le mode sombre fait son apparition cette année aussi, histoire d'économiser les batteries et de ménager les yeux.

Des millions d'adresses @hotmail.com restent en circulation, côtoyant les @live.com, @msn.com, @passport.com et @outlook.com. L'histoire de Hotmail raconte surtout celle d'un service pionnier, avalé par un géant, qui

dut sans cesse se réinventer pour ne pas disparaître dans l'oubli numérique.

Internet Archive

Brewster Kahle crée Internet Archive en 1996. Il veut bâtir une bibliothèque numérique universelle qui garderait trace de l'histoire d'Internet. Pas une archive fermée dans un coffre-fort numérique, mais un espace ouvert où chacun pourrait retrouver les pages disparues du web. La Wayback Machine, lancée en octobre 2001, donne accès 24 ans plus tard à plus de mille milliards de pages Web et plus de 99 pétaoctets de données. Un vertige.

Kahle fixe dès le départ des règles simples. N'utiliser que du matériel standard. Bannir les logiciels commerciaux. Garder une architecture assez simple pour qu'on n'ait pas besoin d'un doctorat pour la maintenir. Ces choix pragmatiques expliquent probablement pourquoi le service tourne depuis plus de vingt ans avec moins de cinq personnes pour s'en occuper. Une longévité presque incroyable dans l'univers du web.

L'infrastructure repose sur des nœuds de stockage dispersés et des serveurs web en façade. Les données se répartissent sur plus de 2 500 nœuds, soit plus de 6 000 disques durs. La capacité totale frôle le pétaoctet. Chaque jour, le système traite des dizaines de millions de requêtes et transfère plus de 40 téraoctets de données.

Contrairement aux sites classiques, Internet Archive n'utilise pas de cache. Cette absence surprend au premier abord. Mais les concepteurs ont vite compris que les requêtes sur des archives se dispersent trop dans le temps et l'espace. Un cache traditionnel ne servirait à rien. Ils ont préféré optimiser directement l'accès aux données stockées.

Le format ARC structure le stockage des pages archivées. Chaque fichier regroupe environ 100 mégaoctets de pages sans lien entre elles, accompagnées de leurs en-têtes. L'organisation rend le stockage efficace mais impose de lire séquentiellement pour retrouver une page précise. Chaque fichier existe sur au moins deux nœuds différents, histoire de ne pas tout perdre en cas de panne.

Pour localiser les données, le système adopte une méthode originale. Pas d'index centralisé. Les requêtes partent en UDP vers tous les nœuds de stockage. Chaque nœud garde en mémoire la liste de ses fichiers et répond s'il a ce qu'on cherche. Cette approche distribuée résiste bien aux pannes et simplifie l'ajout ou le retrait de nœuds.

Les statistiques montrent que l'anglais écrase les autres langues dans les

contenus consultés. Les langues européennes suivent, loin derrière. Plus de 82% des sessions humaines arrivent par des liens externes. Wikipédia arrive en tête des sites qui envoient du trafic vers Internet Archive. La plupart des pages consultées n'existent plus sur le web actif. La preuve que l'archive joue bien son rôle de mémoire.

Deux types d'utilisateurs se partagent les accès : les humains et les robots. Les sessions robotiques sont dix fois plus nombreuses mais génèrent autant de données que les sessions humaines. Les robots ne viennent jamais avec un référent dans leurs requêtes, alors que les utilisateurs humains arrivent presque toujours via des liens.

La croissance du web impose des défis constants. L'index des URL archivées dépasse 2 téraoctets. Au début, sa mise à jour posait problème. Les disques chauffaient trop sous l'activité intense. Le déménagement vers un centre de données mieux refroidi et une méthode de mise à jour incrémentale ont résolu le problème. De plus, les disques SSD ouvrent de nouvelles pistes. Ils surpassent les disques mécaniques sur les accès aléatoires, fréquents dans l'usage des archives. Ces technologies pourraient rendre efficaces des systèmes de cache, jusqu'ici impossibles avec les supports traditionnels.

Internet Archive travaille avec d'autres institutions comme la Bibliotheca Alexandrina en Égypte ou les Archives européennes. Ces collaborations servent à répliquer les données géographiquement. La disponibilité s'améliore et la préservation aussi. Le service propose aussi des collections spécialisées : le Million Book Project, les Archives Prelinger. La mission s'élargit au-delà de l'archivage du web.

En 2024, Google Search facilite plus que jamais l'accès au passé. Les versions archivées de pages Web sont directement accessibles avec un simple lien vers la Wayback Machine de l'Internet Archive.

Internet Archive démontre qu'une approche minimaliste peut aboutir à un service durable. Son architecture privilégie la simplicité et la solidité plutôt que la sophistication. Un système d'archivage à grande échelle est capable de fonctionner avec une équipe réduite. Cette réussite inspire aujourd'hui d'autres projets d'archivage numérique dans le monde.

Wi-Fi

Quand Norman Abramson lança le projet ALOHAnet à l'Université d'Hawaï dans les années 1970, il ne se doutait probablement qu'il allait transformer notre rapport quotidien à l'information. Cette expérience, qui

consistait à relier des ordinateurs par ondes radio à travers les îles hawaïennes, démontrait qu'on pouvait se passer de câbles pour faire circuler des données.

Les choses se sont accélérées en 1985 quand la Federal Communications Commission américaine, sous l'impulsion de Michael Marcus, a autorisé l'usage libre de certaines bandes de fréquences : 902-928 MHz, 2,4-2,4835 GHz et 5,725-5,850 GHz. Ces bandes ISM (Industrial, Scientific and Medical), initialement réservées à des usages industriels et médicaux, sont devenues le terrain de jeu des technologies sans fil grand public. D'autres pays ont suivi le mouvement.

La première norme IEEE 802.11 est apparue en 1997, avec un débit modeste de 2 Mbit/s sur la bande des 2,4 GHz. Mais c'est Apple qui a véritablement popularisé la technologie auprès du grand public en 1999, en intégrant le Wi-Fi dans sa borne AirPort et son ordinateur portable iBook. Ces appareils exploitaient la norme IEEE 802.11b, qui grimpait à 11 Mbit/s. Le nom « Wi-Fi », création d'une agence de marketing, n'a d'ailleurs aucun sens technique particulier, contrairement à ce que laisse supposer sa consonance avec « Hi-Fi ».

L'architecture technique du Wi-Fi s'inspire directement des travaux sur le protocole ALOHA. Le principe CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) autorise le partage d'un canal radio entre plusieurs appareils sans se marcher dessus en permanence. Les recherches de Kleinrock, Tobagi et Lam à UCLA ont contribué à affiner ces mécanismes d'accès au médium, adaptés aux contraintes particulières des communications radio.

Les normes se sont ensuite succédé à un rythme soutenu. IEEE 802.11a a introduit en 1999 la modulation OFDM et la bande des 5 GHz. En 2003, IEEE 802.11g apportait 54 Mbit/s dans la bande bien encombrée des 2,4 GHz. Le vrai bond en avant est venu en 2009 avec IEEE 802.11n et la technologie MIMO, qui multiplie les antennes pour atteindre 600 Mbit/s. IEEE 802.11ac a franchi en 2013 la barre symbolique du gigabit par seconde, avec un débit théorique de 3,5 Gbit/s.

Cette course aux performances répond aux besoins croissants de connectivité. Les premiers réseaux Wi-Fi peinaient à diffuser une simple page web, quand aujourd'hui on diffuse des films en 4K sur différents appareils simultanément. IEEE 802.11ax, baptisé Wi-Fi 6 et publié en 2021, pousse le curseur à 9,6 Gbit/s tout en gérant mieux les environnements denses comme les stades ou les aéroports. La norme intègre des mécanismes sophistiqués de partage du spectre et réduit la consommation énergétique des appareils.

Le Wi-Fi a modifié bien plus que notre façon de nous connecter à internet. La technologie a redessiné les espaces de travail, rendant possible

le nomadisme professionnel. Les cafés sont devenus des bureaux d'appoint, les aéroports des salles de réunion. Dans nos logements, le Wi-Fi a permis l'explosion des objets connectés, des thermostats aux ampoules en passant par les enceintes. Les écoles et universités ont repensé leurs méthodes pédagogiques autour de cette connectivité permanente.

La sécurité des communications radio a longtemps été un talon d'Achille, les premières protections comme le WEP se révélant vite obsolètes. Le chiffrement WPA3 offre maintenant un niveau de protection satisfaisant. Les interférences entre appareils, la cohabitation avec d'autres technologies sans fil comme le Bluetooth, la portée limitée des signaux : autant de problèmes qui ont trouvé des réponses techniques au fil des versions.

L'avenir s'annonce plus ambitieux. Le Wi-Fi 7 (IEEE 802.11be), finalisé en 2024, vise 40 Gbit/s en débit théorique. Ces performances ouvrent des perspectives pour la réalité virtuelle ou augmentée, qui exigent une latence minimale et des débits considérables. L'intégration d'intelligence artificielle dans la gestion des réseaux devrait améliorer l'allocation dynamique du spectre et l'expérience utilisateur.

De l'expérience ALOHAnet aux réseaux Wi-Fi 6, le parcours illustre cette capacité qu'a l'informatique à transformer une idée de laboratoire en technologie du quotidien. Le Wi-Fi est devenu invisible tant il est omniprésent, présent dans nos poches, nos maisons, nos voitures. Cette discrétion témoigne paradoxalement de sa réussite : les meilleures technologies sont celles qu'on ne remarque plus.

Google

Internet passe dans une autre dimension à la fin des années 1990. Les pages web se multiplient à un rythme qui dépasse l'entendement, et les moteurs de recherche de l'époque peinent à suivre. AltaVista, Lycos, Excite : tous butent sur le même problème. Comment classer ces millions de pages en distinguant l'utile du futile ?

Sergey Brin et Larry Page se rencontrent en 1995 à Stanford. Ils sont doctorants, plutôt brillants, et commencent à travailler sur BackRub, un projet qui scrute les liens entre les pages web. L'idée leur vient assez naturellement : une page importante doit être celle vers laquelle pointent beaucoup d'autres pages. Plus les références affluent, plus la page gagne en légitimité. C'est simple, presque évident après coup.

Un an plus tard, BackRub se transforme. Le projet prend un nouveau

nom, inspiré du terme mathématique « googol » qui désigne 10 puissance 100. Cette référence traduit leur ambition : indexer une masse colossale de documents. L'algorithme PageRank naît de cette réflexion, en attribuant une note à chaque page selon la quantité et la qualité des liens entrants, créant une sorte de hiérarchie spontanée du contenu web.

En septembre 1998, les deux étudiants fondent Google Inc. dans un garage de Menlo Park. Andy Bechtolsheim, cofondateur de Sun Microsystems, leur avance 100 000 dollars. Quelques mois avant, ils avaient tenté de vendre leur technologie à Excite pour un million de dollars. George Bell, le PDG, refuse. Cette décision compte parmi les erreurs les plus monumentales de l'histoire d'Internet.

La technique fait la différence. En 1999, Google traite déjà 3,6 millions de requêtes quotidiennes. L'interface est dépouillée, la vitesse impressionne, et surtout les résultats tombent juste. Les internautes s'en rendent compte vite. L'entreprise développe des méthodes d'indexation efficaces et bâtit une infrastructure distribuée capable d'absorber des volumes de données jamais vus.

Google ne se contente pas de lire le contenu des pages. L'algorithme examine leur position dans le maillage du web. Il analyse le texte des liens qui pointent vers une page, l'indexation des contenus étant rendue possible sans les avoir visités. Cette approche fonctionne remarquablement bien pour les images ou les PDF.

Le modèle économique se dessine progressivement. En octobre 2000, le système publicitaire AdWords voit le jour, les annonceurs achètent des mots-clés pour apparaître dans les résultats. L'innovation tient au système d'enchères et à la séparation claire entre liens sponsorisés et résultats naturels. La confiance des utilisateurs reste intacte.

La croissance s'emballe au tournant des années 2000. Le nombre de pages indexées passe de 1 milliard en 2000 à plus de 4 milliards en 2004. Google construit ses propres centres de données, équipés de serveurs conçus sur mesure. Cette maîtrise technique maintient les performances malgré l'explosion du trafic.

La culture d'entreprise cultive l'innovation. La devise « Don't be evil » fixe un cap. Les ingénieurs disposent de 20% de leur temps pour leurs projets personnels. Gmail et Google News naîtront de cette liberté. L'entreprise attire les meilleurs talents en leur offrant un environnement stimulant et des conditions généreuses.

L'introduction en bourse en 2004 se fait par un système d'enchères original. Le succès dépasse les espérances : la valorisation atteint 23 milliards de dollars dès le premier jour. La domination de Google dans l'économie numérique ne fait plus de doute.

Google transforme notre rapport au savoir. Le moteur devient un réflexe quotidien pour des centaines de millions d'utilisateurs. Nos façons de chercher, d'accéder à l'information changent durablement. L'influence déborde le cadre de la recherche : les techniques développées par l'entreprise, notamment dans le traitement de masses de données, inspirent d'innombrables innovations.

PageRank a trouvé des applications dans l'analyse des réseaux, bien au-delà de la recherche web. Les avancées dans le traitement distribué des données, la gestion d'infrastructure, l'apprentissage automatique ont établi des références qui structurent encore l'informatique du XXI^{e} siècle.

VMware

IBM avait déjà inventé la virtualisation dans les années 1960. Les ordinateurs centraux de l'époque permettaient à des utilisateurs de partager les ressources d'une même machine physique à plusieurs systèmes virtuels. Mais cette technologie resta confinée aux mainframes pendant des décennies, jusqu'à ce qu'une petite équipe de l'université Stanford décide de la transposer sur les processeurs x86.

Au milieu des années 1990, le professeur Mendel Rosenblum et ses étudiants tentaient de construire un supercalculateur. Pour faire tourner différents systèmes d'exploitation sur leur prototype, ils expérimentèrent la virtualisation. Les résultats dépassèrent leurs attentes. Rosenblum comprit alors qu'il tenait quelque chose de plus grand qu'un simple projet universitaire. Il fonda VMware en 1998 avec son épouse Diane Greene, Edouard Bugnion, Scott Devine et Edward Wang.

L'idée de départ était simple : les centres de données gaspillaient leurs ressources. Les serveurs tournaient à 10 ou 15% de leur capacité car ils étaient dimensionnés pour absorber les pics d'activité. Le reste du temps, processeurs et mémoire restaient inutilisés. La virtualisation permettrait de faire tourner plusieurs charges de travail sur une seule machine physique.

VMware Workstation 1.0 sortit en 1999. Il demandait un Pentium II à 266 MHz et 64 Mo de RAM. Cette première version supportait MS-DOS 6, Windows 95/98/NT, Red Hat Linux 5.0 et quelques autres systèmes. Le défi technique était considérable : l'architecture x86 n'avait jamais été pensée pour la virtualisation. Les ingénieurs durent ruser.

Leur solution combinait deux approches. L'exécution directe pour le code applicatif non privilégié, la traduction binaire dynamique pour le code

système. Cette méthode hybride donnait des performances correctes tout en isolant les machines virtuelles les unes des autres. C'était une prouesse : personne n'avait réussi à virtualiser proprement l'architecture x86 jusque-là.

ESX Server 1.5 débarqua en 2002. Cet hyperviseur pouvait gérer 64 machines virtuelles simultanées avec 3,6 Go de RAM chacune. Contrairement à Workstation qui s'installait sur un système d'exploitation existant, ESX tournait directement sur le matériel nu. Les performances s'en trouvaient améliorées. Cette année-là, VMware décrocha son premier brevet (US 6397242) pour ses techniques de virtualisation x86.

L'année suivante marqua l'arrivée de vMotion. Cette technologie changeait la donne : on pouvait déplacer une machine virtuelle d'un serveur physique vers un autre sans coupure de service. Plus besoin d'arrêter les applications pour faire de la maintenance. VMware lança aussi vCenter Server, une console centralisée pour piloter tout l'environnement virtuel.

EMC Corporation racheta VMware en 2004 pour 635 millions de dollars. L'entreprise ouvrit des bureaux au Royaume-Uni, en Chine, en Inde et en Irlande. Trois ans plus tard, EMC décida d'introduire VMware en bourse tout en conservant 90% du capital. L'action était proposée à 29 dollars. Elle termina sa première journée à 51 dollars, valorisant la société à 19,1 milliards de dollars.

VMware vSphere sortit en 2009. Cette plateforme intégrait de nombreuses fonctionnalités avancées de gestion des ressources virtuelles. Les choses s'accélérèrent en 2013 avec NSX pour virtualiser les réseaux et vSAN pour le stockage. La virtualisation ne se limitait plus aux serveurs : infrastructure réseau et stockage passaient eux aussi par la couche logicielle.

Entre-temps, les processeurs avaient évolué. AMD et Intel ajoutèrent des extensions matérielles dédiées à la virtualisation (AMD-V et VT-x). VMware adapta ses produits pour en tirer parti. Ces fonctionnalités matérielles facilitaient la virtualisation des systèmes 64 bits et boostaient les performances.

Dell racheta EMC en 2015 et hérita ainsi de VMware. L'entreprise garda son autonomie opérationnelle et poursuivit ses développements. Elle s'investit dans des projets environnementaux comme Carbon Neutrality, visant à diminuer les émissions de CO_2 des centres de données. En 2023, Broadcom a finalisé son acquisition de VMware.

Un chiffre résume l'impact de VMware au-delà de son propre hyperviseur : dès 2009, le nombre de serveurs virtuels dépassait celui des serveurs physiques. La virtualisation x86 avait transformé l'industrie. Elle réduisait les coûts d'infrastructure, optimisait l'utilisation des ressources et apportait de la souplesse aux systèmes d'information. Elle a aussi été fondatrice des

techniques du cloud computing : automatisation, élasticité des ressources, tout reposait sur ces mécanismes de virtualisation.

En rendant la virtualisation accessible sur l'architecture x86, VMware a modernisé la gestion des infrastructures. Son succès a poussé d'autres acteurs à développer leurs propres solutions, comme AWS Nitro, créant une émulation qui profite à tous.

Microsoft Windows 98

Après le triomphe de Windows 95, Microsoft dominait le marché de l'informatique personnelle en 1998. L'entreprise avait rompu avec IBM, désormais en déclin sur le segment des ordinateurs grand public, qui abandonnait OS/2 au profit de Windows et NT. Le système d'exploitation que Bill Gates avait failli voir disparaître à plusieurs reprises régnait maintenant sur l'univers des ordinateurs de bureau.

Windows 98, nom de code Memphis, sortit le 25 juin 1998. La version OSR2 de Windows 95 avait déjà intégré Internet Explorer 3 puis 4, le système de fichiers FAT32 et le support USB. Cette nouvelle mouture devait consolider ces avancées et apporter des améliorations importantes. Jim Allchin, vice-président senior de la gestion des systèmes commerciaux américains, fixa le cap : améliorer les fonctionnalités réseau, simplifier le Panneau de configuration et automatiser davantage le système. L'équipe privilégia l'optimisation qualitative plutôt que l'ajout de nouveautés tape-à-l'œil. La presse spécialisée ne manqua pas de critiquer cette approche, regrettant l'absence d'innovations spectaculaires.

Le développement débuta en janvier 1997 avec une première version pour développeurs. La version alpha initiale (build 1387) sortit le 7 février 1997. Le 3 octobre, la bêta 2.1 (build 1602) permit enfin la mise à niveau depuis Windows 3.x. La Release Candidate 0 arriva le 16 février 1998, suivie de la version bêta finale le 9 mai. Les passionnés de technologie eurent ainsi le temps de tester le système avant sa sortie officielle.

Sur le plan technique, Windows 98 introduisit le Windows Driver Model (WDM) qui remplaça progressivement le modèle VxD hérité de Windows 2.0 pour 386. Cette évolution permit l'arrivée de l'interface ACPI (Advanced Configuration and Power Interface). Sans WDM, l'hibernation système aurait été impossible, les performances des périphériques Plug and Play auraient plafonné, et le traitement audio aurait souffert de limitations importantes. Les fabricants de matériel adoptèrent ce nouveau modèle pro-

gressivement, mais cette modernisation facilita grandement les transitions ultérieures entre versions de Windows.

Le support matériel progressa sensiblement. La gestion USB gagna en maturité avec l'ajout des concentrateurs, des périphériques d'imagerie et audio. La compatibilité avec les contrôleurs IDE et SCSI s'étendit, le support AGP fit son apparition. DirectX 5.2 enrichit les capacités multimédias du système, particulièrement pour les jeux. La gestion multi-écrans et la lecture de DVD complétèrent ces nouveautés.

Les fonctionnalités réseau connurent une refonte importante. L'intégration de Winsock 2, la signature SMB, l'optimisation DHCP et NDIS 5.0 modernisèrent la connectivité. Le partage de connexion Internet simplifia la mise en réseau domestique. Ces perfectionnements techniques ne se déroulèrent pas sans accrocs. Lors d'une présentation publique au COMDEX de Chicago le 20 avril 1998, Chris Capossela et Bill Gates subirent un écran bleu embarrassant en tentant de démontrer les capacités Plug and Play du système.

Les exigences matérielles comprenaient, au minimum, un processeur 486DX2 à 66 MHz, 16 Mo de RAM et 500 Mo d'espace disque disponible. Les recommandations officielles préconisaient un Pentium avec 24 Mo de RAM, certains services informatiques universitaires suggérant 32 Mo. Des utilisateurs découvrirent qu'en utilisant le paramètre /nm lors de l'installation, il était possible de contourner ces prérequis et d'installer le système sur un 80386 avec seulement 4 Mo de RAM. Une installation minimale n'occupait que 120 Mo d'espace disque. Les temps ont bien changé.

Les premiers jours de commercialisation virent s'écouler 530 000 copies en magasin. Ce chiffre, inférieur aux ventes record de Windows 95, s'explique par une promotion plus modeste et des changements moins spectaculaires. Certains revendeurs et clients tentèrent néanmoins de recréer l'atmosphère des lancements de Windows 95, avec des files d'attente et des événements spéciaux à minuit.

Windows 98 Second Edition, lancé le 5 mai 1999, apporta principalement des corrections de bugs mais inclut quelques améliorations. Internet Explorer 5 remplaça la version 4, le support FireWire fit son apparition, la technologie SBP-2 améliora la gestion du stockage USB. DirectX passa en version 6.1, tandis que le support WDM pour l'audio et les modems progressa significativement. Le support de l'API WinG fut abandonné au profit de DirectX, et Windows Media Player remplaça RealPlayer 4.

L'organisation du développement chez Microsoft étaient en avance sur les méthodes actuelles de l'industrie logicielle. Une équipe se concentrait sur le produit en cours pendant qu'une autre préparait la version suivante. Cette séparation maintenait un cap clair tout en préservant la compatibilité

avec les systèmes antérieurs. L'introduction progressive du WDM illustre cette approche : les anciens pilotes restaient utilisables pendant que les fabricants adoptaient le nouveau modèle.

Windows 98 marqua une étape dans l'évolution des systèmes d'exploitation, malgré un accueil mitigé de la presse technique. Son héritage réside dans sa capacité à avoir consolidé les innovations de son époque tout en préparant les évolutions futures. Le système fut progressivement supplanté par Windows 2000 et Windows XP ensuite, qui offraient une stabilité et une sécurité accrues grâce à leur architecture NT. La fin du support officiel par Microsoft au milieu des années 2000 marqua la conclusion de cette période.

XML

En 1996, le W3C confie à Jon Bosak, ingénieur chez Sun Microsystems, la direction d'un groupe de travail ambitieux. La mission est de concevoir un langage de balisage qui conjuguerait la puissance structurante du SGML avec une simplicité d'usage que ce dernier n'avait jamais atteinte. HTML dominait le web, mais ses limites devenaient criantes : conçu pour la mise en forme visuelle, il se révélait inapte à organiser logiquement l'information.

Deux ans plus tard, XML naît officiellement sous forme de recommandation du W3C. Le langage introduit une syntaxe stricte basée sur des balises, impose la notion de document bien formé et offre aux développeurs la liberté de créer leurs propres vocabulaires de balises via les DTD. Chaque secteur peut désormais façonner ses propres structures de données avec flexibilité sans attendre qu'un organisme ne les standardise.

L'innovation majeure tient dans la séparation radicale entre contenu et présentation. Un document XML peut se transformer en page web, en PDF, en fichier texte, selon les besoins du moment. XSLT, normalisé l'année suivante, fournit les outils de transformation nécessaires à cette modularité. Les données acquièrent une autonomie inédite, libérées de leur format de destination.

Les services web émergent en 2000, le commerce électronique s'installe, et les entreprises découvrent l'intérêt d'échanger des informations entre systèmes disparates sans recourir à des conversions hasardeuses. XML s'impose naturellement. Son écosystème s'enrichit : XPath nous fait naviguer dans les documents, XQuery interroge les données, XML Schema supplante progressivement les DTD en mettant à disposition des mécanismes de validation plus fins.

Cette histoire trouve ses racines bien plus tôt, en 1969, dans les laboratoires d'IBM. Charles Goldfarb, Edward Mosher et Raymond Lorie y développent le GML, ancêtre direct du SGML. Leur intuition : la structure d'un document doit être indépendante de son contenu. Ce principe, avant-gardiste à l'époque, étaye encore notre façon de gérer l'information numérique.

Les bases de données s'adaptent. Les systèmes relationnels traditionnels intègrent le stockage natif de documents XML, tandis que des solutions spécialisées apparaissent pour répondre à des besoins spécifiques. Des secteurs entiers bâtissent leurs standards sur ce format, pivot des architectures orientées services qu'on appelle les SOA : l'édition numérique réorganise ses chaînes de production, la finance structure ses échanges de données.

La simplicité relative de XML face au SGML explique en partie son succès auprès des développeurs. Mais c'est surtout sa nature auto-descriptive qui séduit : un document XML porte en lui les clés de sa compréhension. Cette caractéristique garantit une certaine pérennité : les fichiers restent lisibles et exploitables des années après leur création, la documentation externe restant la plupart du temps accessoire.

L'arrivée de JSON au milieu des années 2000 bouleverse quelque peu le paysage. Plus léger, plus direct, il conquiert le développement web. Pourtant, XML ne cède pas le terrain. Il garde sa pertinence là où la rigueur structurelle compte, où la validation formelle s'impose, où les transformations complexes sont nécessaires. Les deux formats coexistent, chacun excellant dans son domaine.

Le langage XML a transformé notre approche de la documentation, encouragé une pensée structurée de l'information, changé la manière dont les organisations conçoivent leurs systèmes. Les principes de balisage hiérarchique qu'il a popularisés se retrouvent dans des formats bien plus récents. Le web sémantique, les architectures de microservices modernes portent son empreinte.

Le W3C a orchestré tout ce processus de normalisation avec une méthode qui fait référence. XML et ses technologies satellites forment aujourd'hui un ensemble cohérent de spécifications, fruit d'un travail collectif remarquable. Ces standards continuent d'évoluer, s'adaptent aux besoins émergents, témoignent de la vitalité d'un format qui approche ses trente ans d'existence sans avoir pris une ride.

GNOME

En 1996, Miguel de Icaza se lance dans une aventure qui va transformer le paysage des systèmes libres : créer un environnement de bureau graphique complet pour GNU/Linux. Le projet GNOME (GNU Network Object Model Environment) répond alors à une frustration bien réelle. Les systèmes UNIX tournaient sans problème sur les serveurs, mais leur interface utilisateur restait rudimentaire. Comment espérer séduire le grand public avec des outils aussi austères ?

La situation se complique avec l'émergence de KDE, qui repose sur Qt, une bibliothèque dont la licence heurte les principes du logiciel libre. La communauté se trouve face à un dilemme : faut-il recréer Qt en version libre ou partir sur autre chose ? L'équipe de GNOME tranche pour la seconde option. Émuler une API existante leur semble une impasse, comme le montrent les difficultés de projets tels que GNUstep, Wine ou LessTif.

Le choix se porte sur Gtk+, une bibliothèque graphique née pour les besoins de GIMP, le logiciel de retouche d'images développé par Peter Mattis et Spencer Kimball. Écrite en C mais pensée objet, Gtk+ offre des bases solides. Autour d'elle viennent se greffer des composants comme Gdk Imlib pour gérer les images ou VFS pour manipuler les fichiers.

Les débuts du projet reflètent l'esprit du libre : discussions ouvertes sur des listes de diffusion, code source partagé via CVS, accès direct au dépôt pour les contributeurs qui ont fait leurs preuves. La version 0.0 sort en août 1997. Mars 1999 voit arriver la 1.0, aussitôt adoptée par Red Hat Linux comme environnement par défaut.

GNOME se construit de manière modulaire. Des bibliothèques comme libgnome ou libgnomeui côtoient des applications de base et des outils de développement. Cette organisation donne de la souplesse car chacun travaille sur son composant sans marcher sur les pieds du voisin. L'interface suit des règles de style strictes pour garantir une cohérence d'ensemble.

Dès le départ, l'internationalisation compte parmi les priorités. Le système GNU `gettext` traduit les interfaces dans de nombreuses langues. GNOME devient accessible aux utilisateurs du monde entier, bien au-delà de la sphère anglophone.

L'année 1999 voit naître deux entreprises autour du projet : Eazel et Ximian, cette dernière fondée par Miguel de Icaza. Eazel développe Nautilus, le gestionnaire de fichiers, mais ferme boutique en 2001. Ximian connaît un meilleur sort en proposant une version enrichie de GNOME et des outils comme Evolution, avant que Novell ne la rachète en 2003.

Un cap se franchit en 2000 avec la création de la GNOME Foundation. Cette structure vient coordonner les efforts des différents acteurs et préser-

ver les objectifs initiaux. La fondation associe un conseil d'administration élu démocratiquement et un conseil consultatif qui réunit entreprises et organisations à but non lucratif.

Cette année, Sun Microsystems fait un choix remarqué : GNOME remplace CDE sur Solaris. Le projet gagne ses lettres de noblesse dans le monde professionnel. Sun va plus loin en créant un laboratoire dédié à l'accessibilité, ouvrant l'environnement aux personnes en situation de handicap.

L'architecture technique s'appuie sur CORBA (Common Object Request Broker Architecture) pour faire communiquer les composants entre eux. Cette technologie autorise l'interaction entre applications écrites dans des langages différents. Les développeurs créent ORBit, une version allégée de CORBA taillée pour les besoins de GNOME.

Le projet encourage l'utilisation de langages de script comme Scheme et Perl. Cette ouverture simplifie la personnalisation des applications et leur automatisation. Le jeu de cartes GNOME exploite Scheme pour définir les règles des différentes variantes de solitaire.

Red Hat s'impose comme contributeur de poids, notamment via son laboratoire de recherche avancée. L'entreprise met en place une compilation quotidienne qui améliore la qualité du code en détectant vite les problèmes. Novell, Collabora et Intel apportent aussi leur pierre à l'édifice.

GNOME instaure un cycle de développement rigoureux avec des versions majeures tous les six mois. Ce rythme soutenu et innovant pour l'époque prévoit des phases de gel progressif : d'abord les fonctionnalités, puis l'interface, enfin les traductions. Cette méthode stabilise les versions sans précipitation.

La version 2.0 de 2002 consolide l'architecture du projet. Elle améliore l'intégration des composants et harmonise davantage l'interface. Les versions suivantes poursuivent ce travail de cohérence tout en ajoutant de nouvelles possibilités.

GNOME montre qu'un projet libre sert à rassembler des bénévoles et des entreprises autour d'objectifs communs. Sa gouvernance équilibrée, qui mêle démocratie communautaire et participation des acteurs économiques, inspire d'autres initiatives. Le projet prouve qu'une alternative libre aux environnements propriétaires peut atteindre un niveau professionnel, tant en qualité qu'en fonctionnalités.

Bluetooth

Dans les laboratoires d'Ericsson, en 1994, des ingénieurs cherchent à résoudre un problème simple : remplacer les câbles qui relient les appareils électroniques par une liaison radio bon marché et peu gourmande en énergie. Ce projet prendra un nom surprenant, emprunté à l'histoire viking. Harald Bluetooth, roi du Danemark au X^{e} siècle, avait unifié le Danemark et la Norvège. Les concepteurs y voient un symbole parfait pour une technologie destinée à connecter des équipements différents.

Quatre ans plus tard, en février 1998, cinq géants de l'électronique – Ericsson, IBM, Intel, Nokia et Toshiba – créent le Bluetooth Special Interest Group. Cette association a pour mission de définir des spécifications techniques communes. Le mouvement prend de l'ampleur : 3COM, Microsoft, Lucent et Motorola rejoignent le groupe, qui compte déjà plus de 1 900 membres en 2000.

La première version commerciale sort en 1999. Les appareils de différents fabricants peinent à communiquer entre eux. La version 1.1, en 2002, corrige ces défauts de jeunesse et stabilise le débit à 1 Mbit/s. Deux ans après, la version 2.0 triple la vitesse de transmission grâce à l'Enhanced Data Rate.

Le fonctionnement repose sur des piconets, ces petits réseaux où un appareil maître coordonne jusqu'à sept esclaves actifs. La bande de fréquence des 2,4 GHz sert de terrain de jeu, avec une astuce intéressante : la fréquence d'émission change 1 600 fois par seconde parmi 79 canaux disponibles. Cette gymnastique limite les interférences.

La sécurité n'a pas été négligée. Les concepteurs ont prévu trois niveaux de protection, du mode ouvert à la liaison chiffrée, avec des mécanismes d'authentification et d'autorisation. Mais c'est en 2010 que tout bascule. La version 4.0 introduit le Bluetooth Low Energy, une variante qui consomme bien moins d'énergie. Les montres connectées, les capteurs de santé et les objets domotiques trouvent là leur voie d'accès au monde sans fil.

Six ans plus tard, la version 5.0 multiplie par quatre la portée et double la vitesse. L'Internet des Objets explose, et le Bluetooth s'adapte. Les couches protocolaires s'empilent : la couche radio gère les ondes, la bande de base contrôle les paquets, le gestionnaire de liaison établit les connexions. Au-dessus, L2CAP découpe et répartit les données. Des protocoles comme RFCOMM émulent aussi des ports série pour garder la compatibilité avec l'existant.

Les profils définissent les usages : transfert de fichiers, synchronisation, téléphonie, audio. Le profil d'accès générique sert de socle à tous les autres. Cette architecture en couches a permis au Bluetooth de dépasser son rôle

initial de simple remplaçant de câble. Les prévisions tablent sur 5,4 milliards d'appareils expédiés en 2023.

En 2017, l'ajout du maillage ouvre de nouvelles perspectives. Des réseaux étendus sont rendus possibles, utiles pour l'éclairage connecté et l'automatisation des bâtiments. Dans les hôpitaux, des capteurs suivent les patients à distance. Les usines collectent des données pour anticiper les pannes. Les balises Bluetooth guident les clients dans les magasins et révolutionnent le marketing de proximité.

Cette technologie illustre bien comment une liaison radio simple peut évoluer vers un système complexe. Du point à point aux réseaux maillés, du câble de remplacement à l'épine dorsale de l'IoT, le Bluetooth a su se transformer sans rompre avec ses versions précédentes. Les débits augmentent, la portée s'étend, l'efficacité énergétique progresse. Et pourtant, un appareil récent dialogue encore avec un vieux périphérique des années 2000.

MSN Messenger

Quand MSN Messenger débarque en 1999, la messagerie instantanée n'a rien d'une nouveauté. L'IRC existe depuis 1988, né de l'imagination du Finlandais Jarkko Oikarinen. Ce système de discussion en temps réel a déjà prouvé son utilité en 1991 lors d'événements comme la guerre du Golfe en janvier ou la tentative de coup d'État contre Mikhaïl Gorbatchev en août. Des milliers d'internautes du monde entier s'y retrouvent pour échanger des informations en direct, créant une forme primitive de réseau social mondial.

Microsoft arrive dans un contexte particulier. L'informatique domestique se démocratise, et la vision de Bill Gates d'un ordinateur connecté dans chaque foyer, reprise par le président Clinton, commence à se matérialiser. Les interfaces graphiques ont rendu les machines accessibles aux novices. Le terrain est prêt pour des applications grand public qui ne nécessitent plus de compétences techniques avancées.

Mais MSN Messenger ne se contente pas de reproduire l'IRC. Là où ce dernier privilégie les canaux publics et les discussions de groupe, Microsoft mise sur l'intimité des conversations privées entre contacts autorisés. L'inspiration vient d'ICQ, créé en 1996 par quatre développeurs israéliens de Mirabilis. Ce petit programme au nom astucieux (« I seek you ») a séduit 100 000 utilisateurs simultanés dès juin 1997. AOL l'achète en 1998, preuve que le marché de la messagerie instantanée attire les convoitises.

L'architecture technique repose sur un modèle client-serveur. Contrai-

rement à certains concurrents qui optent pour le pair-à-pair, Microsoft centralise les échanges sur ses serveurs avec des protocoles propriétaires. Cette approche donne à l'entreprise un contrôle total sur sa plateforme, mais crée aussi des failles de sécurité spécifiques. Les communications passent par le port TCP 5190, mais les utilisateurs débrouillards apprennent vite à contourner les restrictions en redirigeant le trafic vers d'autres ports.

Les fonctionnalités s'enrichissent au fil des versions. Liste de contacts avec indicateurs de présence, conversations textuelles instantanées, transfert de fichiers, avatars personnalisables, messages d'humeur, et bien-sûr les mémorables Wizz. La possibilité de modifier son statut (en ligne, occupé, absent, invisible) représente une innovation sociale notable. Pour la première fois, les utilisateurs contrôlent leur disponibilité de façon explicite, chose impossible avec le téléphone traditionnel.

L'arrivée des mobiles au courant des années 2000 bouleverse les usages. Des versions pour téléphones portables apparaissent, mais leur adoption est mitigée. Les téléphones de l'époque souffrent de limitations rédhibitoires : autonomie faible, écrans minuscules, connexions coûteuses. Une étude menée en Suède entre 2007 et 2008 montre que les utilisateurs considèrent la version mobile comme un simple complément. Ils l'utilisent surtout pour « tuer le temps », pas vraiment pour communiquer sérieusement.

La sécurité pose problème dès le départ. Les utilisateurs s'exposent à toutes sortes de menaces : virus, chevaux de Troie, écoute des conversations, usurpation d'identité. Le CERT documente des attaques exploitant l'ingénierie sociale. Les pirates jouent sur la confiance des utilisateurs pour diffuser des logiciels malveillants déguisés en améliorations légitimes. Le fait que beaucoup d'adolescents utilisent MSN Messenger sans supervision parentale aggrave le problème.

Dans les entreprises, l'outil suscite des débats animés. Certaines organisations y voient un moyen de fluidifier la communication interne et la collaboration informelle. D'autres s'inquiètent des risques de fuite d'informations confidentielles et de la baisse de productivité. Beaucoup finissent par déployer des versions sécurisées et contrôlées, quand elles ne bloquent pas purement et simplement l'accès.

L'interopérabilité reste un sujet épineux. Les utilisateurs aimeraient pouvoir discuter avec leurs contacts quel que soit leur service de messagerie. Mais Microsoft, AOL et Yahoo maintiennent des écosystèmes fermés malgré les demandes répétées et la pression de la FCC. Des solutions comme Trillian permettent de contourner ces barrières en agrégeant plusieurs services, mais sans garantir toutes les fonctionnalités natives.

Internet évolue. Les réseaux sociaux émergent et redéfinissent la communication en ligne. Facebook propose sa propre messagerie intégrée. Les

smartphones modernes rendent obsolètes les applications de bureau. Microsoft finit par remplacer MSN Messenger par Skype en 2013, actant la fin d'une ère qui aura duré quatorze ans.

Les applications de messagerie contemporaines reprennent les concepts inventés ou popularisés par MSN Messenger : listes de contacts, indicateurs de présence, conversations instantanées, partage de fichiers, émojis. WhatsApp, Telegram, Signal ou Discord doivent beaucoup à ce pionnier.

MSN Messenger a modifié durablement nos façons de communiquer. En rendant la messagerie instantanée accessible au grand public, il a transformé les relations interpersonnelles à l'ère numérique. Son histoire reflète les mutations technologiques et sociales de la charnière entre XX^{e} et XXI^{e} siècles, période où l'informatique connectée est devenue partie intégrante du quotidien.

RSS

Dave Winer lança scriptingNews en 1997, alors qu'il travaillait chez UserLand. Ce format cherchait à rendre plus simple le partage de contenus sur Internet. Deux ans plus tard, Netscape reprit l'idée et créa RSS 0.90 (Rich Site Summary), un format XML avec un en-tête RDF qu'il déploya aussitôt sur my.netscape.com.

UserLand réagit avec scriptingNews 2.0b1, qui intégrait les fonctionnalités de RSS 0.90. Netscape répondit avec RSS 0.91, une version allégée sans RDF mais gardant l'essentiel de scriptingNews. UserLand abandonna alors son propre format pour rallier RSS 0.91. Netscape stoppa ensuite le développement de la technologie.

L'année 2000 marqua un tournant. Un groupe mené par Rael Dornfest chez O'Reilly conçut RSS 1.0, une refonte complète basée sur RDF et les espaces de noms XML. Cette version rompait avec la lignée 0.9x. Dave Winer continua de son côté avec RSS 0.92, ajoutant des éléments facultatifs à la version 0.91.

En 2002, après avoir quitté UserLand, Winer créa RSS 2.0. Cette mouture enrichissait RSS 0.92 tout en gardant la compatibilité avec les versions précédentes. L'université Harvard publia la spécification sous licence Creative Commons en 2003, ouvrant le format à tous.

RSS permit aux sites web d'annoncer leurs mises à jour dans un format standardisé. Les utilisateurs s'abonnaient à ces flux via des agrégateurs et

recevaient les nouveaux contenus sans visiter chaque site. Les blogs profitèrent largement de cette automatisation.

Les bibliothèques et centres de documentation adoptèrent vite la technologie pour leurs alertes et leur veille. Le format diffusait les sommaires de revues scientifiques ou les nouvelles acquisitions. BioMed Central et Nature l'intégrèrent à leurs plateformes.

La diversité des outils de lecture explosa. FeedReader et NetNewsWire côtoyaient des solutions en ligne comme Bloglines. Firefox et Opera ajoutèrent des fonctions natives de gestion des flux. Google Reader devint une référence jusqu'à sa fermeture en 2013.

La structure XML de RSS garantissait l'interopérabilité. Chaque flux contenait des éléments obligatoires : titre, lien, description. Des métadonnées facultatives enrichissaient le tout. La hiérarchie organisait les contenus par catégories et acceptait les pièces jointes multimédias.

Les médias en ligne transformèrent grâce à RSS la façon dont on consommait l'information. Les lecteurs centralisaient le suivi de leurs sources préférées et filtraient selon leurs intérêts. Cette autonomie réduisait la dépendance aux portails traditionnels.

Le marketing numérique s'empara du format pour diffuser offres commerciales et newsletters. La syndication touchait les audiences via les agrégateurs. Google créa AdSense pour monétiser les flux RSS, ouvrant des opportunités publicitaires.

Les réseaux sociaux et Twitter changèrent progressivement les habitudes. RSS resta néanmoins utile aux professionnels de l'information et aux utilisateurs soucieux de maîtriser leurs sources.

Le format inspira d'autres standards comme Atom, normalisé en 2005 par l'IETF. Ces spécifications complémentaires enrichirent l'écosystème du web en facilitant l'agrégation de contenus structurés.

L'architecture distribuée de RSS influença le web social. La publication automatisée et les abonnements inspirèrent les notifications des réseaux sociaux. Le principe de flux d'activité, central dans les interfaces modernes, vient en partie de là.

La simplicité de RSS en fit un pilier du web participatif des années 2000. La technologie montra l'importance des standards ouverts pour l'interopérabilité des services. Son héritage perdure dans les protocoles actuels de distribution de contenu.

En 2024, des millions de sites et d'applications utilisent encore RSS. Les flux alimentent des systèmes de veille, des agrégateurs d'actualités et des outils d'automatisation. Cette longévité témoigne de la pertinence d'un modèle décentralisé de distribution de l'information.

SOAP

Fin 1997, Microsoft commence à explorer une idée qui va transformer la communication entre applications : utiliser XML pour des appels de procédures distantes via HTTP. L'ambition est simple : faire dialoguer des machines à travers le réseau avec des types de données standard, sans les complications des protocoles propriétaires. DevelopMentor, habitué à collaborer avec Microsoft, et Userland, qui voit le Web comme un terrain de publication, rejoignent l'aventure. Le nom SOAP émerge au début de 1998.

Mais les choses se compliquent vite. L'équipe DCOM chez Microsoft freine des quatre fers. Plutôt que d'adopter cette nouvelle approche, elle préfère exploiter la position dominante de l'entreprise pour imposer DCOM via un *tunneling* HTTP. Les experts XML maison trouvent l'idée séduisante mais prématurée : ils attendent les fonctionnalités avancées que promettent XML Schema et les espaces de noms. Face à ce blocage, Userland prend les devants et publie sa propre version des spécifications sous le nom XML-RPC durant l'été 1998.

En 1999, Microsoft avance sur XML Schema et intègre les espaces de noms dans ses produits. SOAP regagne du terrain, l'équipe BizTalk restant réticente car leur modèle repose sur la messagerie, pas sur les appels de procédures distantes. Le 13 septembre 1999, SOAP 0.9 sort pour examen public et se retrouve soumis à l'IETF. Trois mois plus tard, SOAP 1.0 voit le jour avec peu de changements.

En mars 2000, le W3C annonce qu'il envisage une activité autour des protocoles XML. À la conférence XTech, une session animée rassemble plusieurs visionnaires qui débattent des orientations futures sans parvenir à un consensus. Mais le 8 mai 2000 marque un vrai tournant : SOAP 1.1 arrive au W3C avec IBM comme co-auteur. Ce soutien inattendu change tout. La nouvelle version se révèle bien plus modulaire et extensible, dissipant les craintes que SOAP n'impose des technologies propriétaires Microsoft. IBM publie aussitôt une implémentation Java qu'il confie au projet Apache XML pour un développement open source. Les sceptiques commencent à s'intéresser sérieusement au protocole. Sun manifeste son intérêt et travaille à intégrer les services Web dans J2EE. D'autres fournisseurs et projets libres emboîtent le pas.

En septembre 2000, le W3C forme un groupe de travail dédié au protocole XML, qui prend SOAP 1.1 comme point de départ. Après des mois de modifications, d'améliorations et de décisions difficiles sur ce qu'il fallait conserver ou abandonner, SOAP 1.2 devient une recommandation officielle

en juin 2003.

SOAP s'impose parce qu'il représente le meilleur compromis industriel pour standardiser l'informatique distribuée multi-plateformes basée sur XML. Sa simplicité est son atout majeur : historiquement, les architectures adoptées massivement l'ont été grâce à cette qualité.

Le protocole définit l'unité de communication via une enveloppe qui encadre toutes les informations. Un message contient un corps où du XML arbitraire peut prendre place, accompagné d'en-têtes qui transportent des données en dehors du corps principal. Le modèle de traitement fixe des règles précises pour gérer les messages quand des extensions entrent en jeu. Les SOAP faults gèrent les erreurs en identifiant leur source et leur cause, tout en autorisant l'échange d'informations de diagnostic entre participants.

L'extensibilité passe par les en-têtes SOAP qui transportent des données d'extension avec le message et peuvent cibler des nœuds spécifiques le long de son parcours. SOAP propose un mécanisme flexible de représentation des données : il accepte des données déjà sérialisées dans divers formats (texte, XML) et fournit une convention pour représenter des structures abstraites comme les types des langages de programmation en XML.

Les appels de procédures distantes et leurs réponses se « mappent » naturellement en messages SOAP. C'est un type d'interaction courant en informatique distribuée qui correspond bien aux constructions des langages procéduraux. Le cadre de liaison définit une architecture pour construire des liaisons qui envoient et reçoivent des messages SOAP via des transports arbitraires. Ce cadre sert notamment à déplacer les messages SOAP à travers HTTP, le protocole omniprésent sur Internet.

SOAP reste largement utilisé dans les entreprises pour l'intégration d'applications et de services, surtout avec les systèmes existants. Les secteurs bancaires et financiers comptent parmi ses utilisateurs fidèles. Google l'emploie pour la plupart de ses applications, tout comme PayPal, Amazon et eBay. Le protocole conserve sa pertinence grâce à sa robustesse technique et sa capacité à gérer des opérations complexes qui nécessitent de maintenir des états conversationnels et des informations contextuelles. Son évolution a permis le développement des services Web et l'essor de l'informatique distribuée.

Transport Layer Security

La sécurité des communications sur Internet repose aujourd'hui sur un socle technique discret mais indispensable. En 1994, Netscape cherche à protéger les échanges sur son navigateur Web et invente SSL (Secure Sockets Layer). La première version publique, SSL 2.0, sort l'année suivante. Mais ce premier jet contient trop de failles pour être vraiment fiable. SSL 3.0 arrive en 1996 pour corriger le tir.

Trois ans plus tard, l'IETF récupère le projet et le baptise TLS 1.0. Ce changement de nom marque aussi une prise en main par la communauté technique internationale. Le protocole crée un tunnel chiffré entre un client et un serveur, garantissant que personne ne peut lire ou modifier les données qui transitent.

Les années 2000 voient émerger des acteurs comme Comodo, spécialisés dans l'émission de certificats numériques. Différentes formules apparaissent : les certificats EV (Extended Validation) en 2007, qui nécessitent des vérifications poussées, puis les certificats DV (Domain Validation), plus simples à obtenir. TLS 1.1 sort en 2006 avec quelques protections supplémentaires contre certaines attaques, sans vraiment convaincre les administrateurs systèmes de migrer massivement.

En 2008, TLS 1.2 apporte son lot de nouveautés : support des extensions, mécanismes de sécurité renforcés. Pourtant l'adoption traîne. Il faudra attendre plusieurs incidents retentissants pour que le secteur comprenne l'urgence de moderniser ses infrastructures.

Entre 2011 et 2014, les alertes se multiplient. L'attaque contre DigiNotar en 2011 révèle qu'un pirate a réussi à pénétrer ce certificateur néerlandais et à émettre des centaines de certificats frauduleux. L'attaque BEAST exploite une vulnérabilité dans TLS 1.0. Le réveil est brutal.

En 2014 la faille Heartbleed touche plus de 300 000 serveurs web publics. Trois ans après sa découverte, près de 180 000 équipements restent vulnérables. POODLE en 2014, FREAK et DROWN en 2015-2016 confirment que la vigilance doit être permanente. Chaque incident accélère la migration vers TLS 1.2 et pousse au développement d'une version vraiment nouvelle.

TLS 1.3 débarque en 2018 après plusieurs années de gestation. Cette mouture ne se contente pas de rafistoler l'existant : elle nettoie le protocole de fond en comble. Les fonctionnalités obsolètes ou dangereuses disparaissent. La poignée de main, ce ballet initial entre client et serveur, passe de deux à un seul aller-retour dans la plupart des cas. Le gain de rapidité se fait sentir immédiatement.

Cette version impose le Perfect Forward Secrecy, une propriété qui empêche de déchiffrer rétrospectivement les communications y compris si les

clés long terme tombent entre de mauvaises mains plus tard. Le nombre de suites cryptographiques passe de plus de 35 à seulement cinq. Ce grand ménage améliore la sécurité mais demande des ajustements dans les infrastructures existantes.

L'histoire de TLS reflète celle des menaces sur Internet. Le protocole a dû s'adapter sans cesse, jongler entre innovation et compatibilité avec les systèmes déjà déployés. Les versions successives ont durci les mécanismes cryptographiques et simplifié leur utilisation. Parfois, l'inertie a freiné les évolutions. Mais les incidents de sécurité ont régulièrement rappelé qu'on ne pouvait pas se reposer sur ses acquis.

Désormais, TLS protège tout un pan du trafic web. Le protocole ne se limite plus aux transactions bancaires ou aux données sensibles. Il est la norme pour toute communication en ligne, surtout depuis que Google a indiqué en 2015 que le chiffrement influençait l'indexation, même si cette exigence est plus une conséquence de l'affaire PRISM et la nécessité d'élever le niveau de sécurité du web globalement. Par son architecture modulaire, de nouveaux algorithmes sont ajoutés au fur et à mesure des avancées en cryptographie.

Les leçons du développement de TLS ont essaimé vers d'autres protocoles de sécurité. L'importance de la validation formelle, la transition progressive vers de nouvelles versions, l'équilibre délicat entre sécurité et performances sont devenus des principes directeurs dans le domaine. L'expérience accumulée montre que la sécurité des communications numériques exige une évolution continue.

Napster

Shawn Fanning avait 18 ans quand il a créé Napster en 1999. L'histoire commence simplement : cet étudiant voulait partager des fichiers musicaux avec ses amis. Il ne se doutait pas qu'il allait déclencher l'une des plus grandes révolutions de l'histoire de la musique.

Le principe technique reposait sur le *peer-to-peer*. Les utilisateurs échangeaient directement des fichiers MP3 entre leurs ordinateurs, sans serveur central pour stocker les données. Napster ne conservait qu'un répertoire des fichiers disponibles sur les machines connectées. Quand quelqu'un cherchait une chanson, le logiciel établissait une connexion directe avec l'ordinateur d'un autre membre qui possédait le fichier.

En moins de deux ans, plus de 26 millions d'utilisateurs ont rejoint le

service et échangé plus de 80 millions de chansons. Cette croissance explosive tenait à la simplicité d'utilisation et à la gratuité. Il suffisait de taper le titre d'une chanson pour la télécharger. Le format MP3, qui compressait les fichiers audio sans trop dégrader la qualité, rendait ces transferts rapides avec les connexions Internet de l'époque.

Les maisons de disques ont très vite compris la menace. Elles contrôlaient la diffusion musicale depuis toujours grâce à la vente de supports physiques. Napster bousculait ce modèle en permettant la circulation libre des œuvres sur Internet. La Recording Industry Association of America (RIAA) a estimé ses pertes à 55 milliards de dollars sur une décennie.

Le procès a commencé en décembre 1999. La RIAA poursuivait Napster pour violation contributive des droits d'auteur. L'accusation ne portait pas sur une violation directe, mais sur le fait de faciliter ces violations par les utilisateurs. David Boies, l'avocat de Napster, s'est appuyé sur l'Audio Home Recording Act de 1992 qui autorisait les copies privées à usage personnel non commercial.

Les tribunaux devaient trancher une question inédite : le partage de fichiers constituait-il un « usage loyal » selon la loi américaine ? Quatre critères entraient en jeu : le but de l'utilisation, la nature de l'œuvre, la proportion utilisée, et l'impact sur le marché. En juillet 2000, la juge Marilyn Patel a ordonné l'arrêt de la distribution de fichiers protégés. Une cour d'appel a suspendu temporairement cette décision, mais Napster a fermé en 2001.

Cette fermeture n'a rien arrêté. D'autres services ont pris le relais avec des architectures plus décentralisées : Gnutella, Kazaa, puis BitTorrent. Chacun apprenait des failles juridiques du précédent pour mieux se protéger des poursuites. L'industrie musicale se trouvait face à un problème technique qu'elle ne pouvait résoudre par les seuls moyens juridiques.

Apple a saisi l'opportunité en lançant iTunes en 2003. Steve Jobs a compris qu'il fallait proposer une alternative légale qui soit aussi simple que Napster. Le pari a fonctionné : les gens acceptaient de payer si le service restait pratique. Cette approche a montré qu'un nouveau modèle économique était possible.

L'histoire de Napster révèle surtout l'inadaptation des lois sur le droit d'auteur face aux possibilités ouvertes par Internet. Ces lois avaient été conçues pour un monde où la copie d'une œuvre nécessitait des moyens industriels. Le numérique permettait à chacun de reproduire et distribuer du contenu sans aucune dégradation. Les cadres juridiques du XX^{e} siècle ne tenaient plus.

Le service a transformé les habitudes de consommation musicale. Une génération entière s'est habituée à la gratuité et à l'immédiateté. Cette at-

tente n'a pas disparu avec la fermeture de Napster. Elle a forcé l'industrie à repenser ses modèles, conduisant finalement aux services de streaming comme Spotify qui tentent de concilier accès libre et rémunération des artistes.

La dimension internationale a compliqué les choses. Napster touchait des utilisateurs partout dans le monde, mais les lois sur le droit d'auteur restaient nationales. Comment appliquer le cadre juridique américain à des échanges transfrontaliers ? Cette question reste d'actualité pour toutes les plateformes numériques.

Les procès ont créé une jurisprudence importante. Ils ont montré qu'Internet ne pouvait fonctionner comme une zone de non-droit. Les législateurs du monde entier s'en sont inspirés pour développer leurs propres cadres réglementaires. L'affaire Napster a établi des précédents qui influencent encore les débats sur la régulation du numérique. L'équilibre entre innovation, droits des créateurs et attentes des utilisateurs demeure fragile.

Bug de l'an 2000

L'histoire du bug de l'an 2000 commence bien avant les années de panique qui précédèrent le passage au nouveau millénaire. Dans les années 1960, quand les programmeurs travaillaient avec des cartes perforées limitées à 80 colonnes et que chaque octet de mémoire coûtait une fortune, coder les dates sur deux chiffres semblait une décision parfaitement raisonnable. Qui aurait imaginé que ces programmes tourneraient encore quarante ans plus tard ? Cette économie d'espace, anodine sur le moment, allait se transformer en l'un des plus grands casse-têtes techniques de la fin du XX^{e} siècle.

Le mécanisme du problème était d'une simplicité désarmante. Les systèmes ne stockant que les deux derniers chiffres de l'année ne pouvaient distinguer 1900 de 2000. Un calcul de durée entre deux dates, un tri chronologique, une vérification de validité : autant d'opérations qui risquaient de produire des résultats aberrants. Les premiers signes apparurent dès les années 1980, bien avant que l'expression « bug de l'an 2000 » ne fasse la une des journaux. En 1988, la chaîne britannique Marks & Spencer refusa une livraison de conserves dont le système avait interprété la date de péremption en 2000 comme 1900, considérant les produits périmés depuis près de quatre-vingt-dix ans. L'anecdote fit sourire, mais elle annonçait des ennuis autrement plus sérieux.

La prise de conscience s'amorça vraiment courant 1990. Les experts

commencèrent à multiplier les alertes : dysfonctionnements bancaires, réseaux électriques à l'arrêt, contrôle aérien paralysé, équipements médicaux défaillants. En 1992, Mary Bandar, une dame de 104 ans, reçut une invitation à rejoindre une classe de maternelle — son année de naissance « 88 » ayant été lue comme 1988 plutôt que 1888. L'incident aurait pu prêter à rire s'il n'avait révélé l'ampleur du problème qui se profilait.

Les gouvernements finirent par réagir. Le Royaume-Uni créa la « Taskforce 2000 » en 1996, suivie l'année suivante par « Action 2000 », dotée d'un budget initial d'un million de livres sterling qui grimpa à 17 millions. Les Nations Unies établirent en février 1999 le Centre international de coopération Y2K, financé par la Banque mondiale. Les entreprises lancèrent des programmes de mise en conformité massifs. La Bourse de New York y consacra 30 millions de dollars sur sept ans.

Mais le problème ne se cantonnait pas aux grands systèmes centraux. Les contrôleurs logiques programmables, omniprésents dans l'industrie pour commander machines et processus, utilisaient souvent des dates à deux chiffres. Les PC présentaient leurs propres complications, liées à leur horloge temps réel et à leur BIOS. Certaines versions, comme l'Award v4.50 de 1994-1995, ne géraient tout simplement pas les dates au-delà de 1999, tout comme Windows 95 n'était pas compatible non plus avec le passage à l'an 2000.

Face à cette situation, quelques stratégies techniques émergèrent. L'expansion des années à quatre chiffres représentait la solution la plus solide, mais elle exigeait de revoir en profondeur les programmes et les bases de données. La technique dite du « windowing » conservait le format à deux chiffres tout en ajoutant une logique d'interprétation : « 00 » à « 19 » correspondait à 2000-2019, « 20 » à « 99 » à 1920-1999. Cette méthode, moins coûteuse, avait le défaut de simplement reporter le problème à une date ultérieure.

Les audits se multiplièrent à partir de 1997. Les cabinets commencèrent à exiger de leurs clients qu'ils certifient la conformité de leurs systèmes critiques pour garantir la continuité de leur activité. Des sociétés spécialisées développèrent des outils automatisés pour détecter et corriger les problèmes dans le code source. Des programmeurs COBOL sortirent de leur retraite, réclamant des salaires confortables pour intervenir sur les systèmes qu'ils avaient bâtis des décennies plus tôt.

Les tests révélèrent des failles inquiétantes. Le système antimissile Rapier britannique se révéla inopérable après 2000. Une centrale nucléaire suédoise s'arrêta automatiquement lors d'un test de passage au nouveau millénaire. Chez Chrysler, un essai en 1997 paralysa le système de sécurité d'une usine, bloquant les accès et empêchant la gestion de la paie.

Le coût global des corrections oscilla entre 300 et 500 milliards de dollars. Les États-Unis dépensèrent 34 milliards, l'Italie 2,5 milliards, le Venezuela 100 millions. Les pays industrialisés, dont l'économie reposait davantage sur les systèmes numériques, investirent proportionnellement plus que les autres.

Finalement, le passage à l'an 2000 provoqua moins de chaos que redouté. Des incidents survinrent : quinze réacteurs nucléaires s'arrêtèrent, des systèmes de paiement par carte dysfonctionnèrent, des pannes de courant touchèrent Hawaï. Mais la catastrophe annoncée n'eut pas lieu, résultat direct des efforts déployés pendant la décennie précédente.

Cette expérience laissa des traces. Elle démontra que les principes d'ingénierie logicielle comme l'abstraction et l'encapsulation n'étaient pas que des concepts théoriques. Elle souligna les dangers des points de défaillance uniques et l'intérêt d'un couplage faible entre systèmes. Elle prouva qu'une mobilisation internationale face à une menace informatique identifiée restait possible.

Paradoxalement, le succès de la gestion du « Y2K » freina l'adoption de certaines leçons. L'industrie continua de privilégier la rapidité de mise sur le marché au détriment de la robustesse. Les chaînes d'approvisionnement se resserrèrent, réduisant leur capacité à absorber les chocs. La crise actuelle de la cybersécurité témoigne de cette difficulté persistante à concevoir des systèmes informatiques à la fois robustes et sécurisés. Le bug de l'an 2000 aura été, au fond, une occasion manquée d'apprendre vraiment de nos erreurs.

Ceci illustre bien deux principes d'architecture à garder en tête dans tout travail de conception, informatique ou pas : « Les problèmes d'aujourd'hui viennent des solutions d'hier » et « La cause et l'effet peuvent intervenir à des temps et des lieux lointains ».

IX

2000

Quand l'humanité s'est mise en réseau

Le XXIe siècle a débuté dans le tumulte. Les attentats du 11 septembre 2001 ont secoué notre monde jusque dans ses fondements. Du jour au lendemain, la vision du numérique s'est transformée. Les gouvernements ont durci leur approche de la sécurité informatique, les infrastructures vitales se sont blindées, et les communications ont été placées sous surveillance. À Washington, le Patriot Act a remodelé notre rapport à la vie privée numérique.

Internet s'est démocratisé comme jamais auparavant. Nous avons vu naître ce qu'on a baptisé le Web 2.0, une mutation où l'internaute agit. MySpace et Facebook ont créé de nouveaux territoires sociaux virtuels. Twitter a changé notre rapport à l'actualité. Les blogs personnels ont fleuri partout grâce à WordPress, donnant voix à qui le souhaitait. Ce bouillonnement a redéfini notre manière d'exister en ligne.

L'économie du numérique a traversé une tempête salvatrice. L'éclatement de la bulle Internet a balayé les projets les plus fragiles. Les survivants ont dû prouver leur valeur réelle. Google s'est imposé comme le cartographe de ce nouvel univers informationnel grâce à son algorithme PageRank. Amazon a révolutionné le commerce en créant une expérience d'achat entièrement repensée.

La mobilité a pris une place centrale dans notre quotidien. Les premiers smartphones ont commencé à transformer nos habitudes. L'iPhone en 2007 provoqua une rupture totale. La façon dont nous touchions, manipulions les appareils ne serait plus jamais la même. Les réseaux sans fil se sont multipliés, le Wi-Fi est devenu omniprésent, et la 3G nous a offert Internet dans notre poche.

Le commerce en ligne a mûri et les systèmes de paiement se sont sécurisés, inspirant confiance aux consommateurs les plus réticents. L'analyse des données a permis aux sites marchands de nous connaître parfois mieux que nous-mêmes, suggérant des produits avec une précision troublante. Pour répondre à cette explosion des usages, les centres de données ont dû se réinventer, devenir plus efficaces, moins énergivores.

L'expérience nouvelle du partage s'est produite avec l'émergence des technologies *peer-to-peer*. Napster a ébranlé l'industrie musicale, alors que BitTorrent a transformé nos modes de partage. Les majors du disque ont d'abord résisté avant de s'adapter. Last.fm et Spotify ont inventé de nouvelles façons d'écouter de la musique, annonçant la révolution du streaming vidéo à venir.

Dans les entreprises, le travail s'est métamorphosé. Le haut débit et les outils collaboratifs ont rendu possible le télétravail. Les organisations ont massivement investi dans leurs systèmes d'information, déployant des solutions ERP et CRM pour mieux gérer leurs activités et leurs relations clients.

L'open source a gagné ses lettres de noblesse. Linux, autrefois considéré comme un projet marginal, s'est installé dans les salles serveurs des grandes entreprises. Firefox a prouvé qu'un navigateur libre pouvait tenir tête aux géants. Des communautés passionnées se sont formées autour de ces projets, montrant la puissance du développement collectif.

La fracture numérique a préoccupé les gouvernements du monde entier. Des programmes d'équipement des écoles ont été lancés. L'accès à Internet est devenu un enjeu d'égalité sociale. En Asie, la Chine et l'Inde ont bâti des industries technologiques puissantes en un temps record, transformant l'équilibre mondial du secteur.

Dans les *datacenters*, la virtualisation a explosé et changé la donne. Un seul serveur physique pouvait héberger plusieurs machines virtuelles. Cette évolution a fait émerger le *cloud computing*. Les DSI ont commencé à externaliser leurs infrastructures, gagnant en agilité.

La cybersécurité est devenue un sujet majeur. Les attaques informatiques se sont multipliées et sophistiquées. Les virus ont évolué en *ransomwares* et *malwares* complexes. Face à cette menace grandissante, les solutions de protection ont dû s'adapter en permanence. La criminalité s'est

organisée en ligne, ciblant les données bancaires et personnelles.

Dans les écoles, le numérique a fait son entrée. Les tableaux blancs interactifs ont remplacé la craie. Les Environnements Numériques de Travail ont créé de nouveaux liens entre enseignants, élèves et parents. Les ressources pédagogiques se sont enrichies, diversifiées, numérisées.

La recherche scientifique a franchi des paliers décisifs grâce à la puissance de calcul croissante. Les supercalculateurs ont rendu possibles des simulations climatiques précises, des avancées en génomique, des découvertes en physique fondamentale. Les scientifiques ont appris à travailler avec des systèmes de calcul distribués pour résoudre des problèmes toujours plus complexes.

L'automatisation s'est intensifiée dans tous les secteurs. Les systèmes embarqués sont devenus plus intelligents, plus communicants. La robotique industrielle a franchi un cap, rendant les usines plus productives mais posant aussi des questions sociales importantes.

La conscience environnementale a fait son apparition dans le monde informatique. La consommation électrique des datacenters est devenue un sujet de préoccupation. Le *Green IT* a émergé comme une nécessité, non plus seulement comme un argument marketing.

De nouvelles formes de collaboration ont vu le jour. Wikipédia, lancée en 2001, a montré la voie d'une intelligence véritablement collective. Les wikis et autres outils collaboratifs ont transformé la manière dont les équipes travaillaient ensemble, partageaient leurs connaissances.

Nos appareils électroniques ont convergé vers un écosystème connecté. Téléviseurs, appareils photo, chaînes Hi-Fi : tout est devenu « intelligent » et interconnecté. Cette convergence a nécessité de nouveaux standards pour que nos équipements puissent communiquer entre eux.

Cette décennie a façonné notre monde numérique actuel. Ce n'était pas une simple évolution technologique, mais une transformation de notre société. Nous avons changé notre manière de communiquer, de travailler, de nous divertir. La période 2000-2010 restera dans l'histoire comme celle où l'humanité s'est véritablement mise en réseau, où notre monde est devenu irrémédiablement connecté.

C#

Anders Hejlsberg débarque chez Microsoft en 1996 avec une solide réputation : il a développé le compilateur Turbo Pascal et dirigé la création

de Delphi chez Borland. Microsoft lui confie alors une mission ambitieuse : concevoir un nouveau langage de programmation pour sa future plateforme .NET. Ce langage, d'abord baptisé « Cool » durant sa conception, deviendra C#. L'objectif était clair : offrir une alternative moderne à Java, en évitant ses écueils tout en reprenant ses meilleures idées.

Le C# se différencie d'emblée par son système de types unifié. Tous les types, même les plus basiques comme `int` ou `double`, héritent d'un type racine unique appelé `object`. Cette architecture simplifie la manipulation des données, quelle que soit leur nature. Le langage intègre aussi la collecte automatique de la mémoire, qui récupère l'espace occupé par les objets devenus inutiles, et une gestion structurée des erreurs par exceptions.

Chaque nouvelle version a enrichi le langage de fonctionnalités substantielles. La version 2.0 introduit les génériques, qui renforcent la réutilisabilité du code et la sécurité des types. Les délégués anonymes élargissent les possibilités en programmation événementielle. Avec la version 3.0 arrivent les fonctionnalités LINQ (Language Integrated Query), qui transforment radicalement la façon d'interroger et de manipuler les données.

Hejlsberg et son équipe ont accordé une attention particulière à la compatibilité des versions. Les applications existantes devaient continuer à fonctionner avec les nouvelles versions des bibliothèques. Cette préoccupation transparaît dans plusieurs choix de conception : les modificateurs `virtual` et `override` restent distincts, les règles de résolution des surcharges de méthodes sont clairement définies. Cette rigueur garantit la pérennité des programmes.

La recherche chez Microsoft a joué un rôle déterminant dans l'évolution du langage. Don Syme et Andrew Kennedy, chercheurs à Microsoft Research Cambridge, ont développé un prototype baptisé Gyro pour expérimenter les génériques avant leur intégration dans C#. Cette collaboration entre recherche et développement illustre bien l'approche pragmatique retenue : tester les idées nouvelles dans des projets exploratoires avant de les incorporer dans le langage de production.

Le C# s'est progressivement ouvert à la programmation fonctionnelle. L'inférence de types, les expressions lambda, tous ces concepts issus de langages comme Haskell ou ML ont été intégrés au fil du temps. Cette évolution suit une tendance générale dans l'industrie : les paradigmes de programmation se mélangent pour offrir aux développeurs plus de souplesse. Le langage reste néanmoins accessible aux débutants, une caractéristique pensée dès sa conception.

La question de l'intégration avec les bases de données revient constamment. Les développeurs doivent souvent jongler entre C# et SQL, ce qu'Hejlsberg appelle une « disparité d'impédance » entre ces deux univers. LINQ

répond à cette problématique en unifiant l'interrogation de diverses sources de données directement dans le code C#, avec une syntaxe cohérente.

Microsoft a fait évoluer en parallèle C# et Visual Basic .NET, deux langages qui partagent de nombreuses fonctionnalités tout en conservant leurs spécificités. Cette stratégie visait à répondre aux préférences distinctes de leurs communautés respectives. L'équipe de développement maintient aussi des échanges réguliers avec les chercheurs travaillant sur F#, le langage fonctionnel de la plateforme .NET, pour explorer de nouvelles pistes.

La normalisation ECMA-334 et ISO/IEC 23270 a établi des standards techniques garantissant la cohérence des implémentations. Cette standardisation a renforcé la portabilité des applications C# et leur indépendance vis-à-vis de Microsoft.

Le C# adopte résolument une orientation composant. Le langage propose des constructions syntaxiques spécifiques pour créer et utiliser des modules logiciels autonomes et autodocumentés. Cette approche modulaire facilite le développement d'applications complexes et leur maintenance à long terme.

La communauté des développeurs C# s'est considérablement élargie, créant un écosystème riche en bibliothèques, outils et bonnes pratiques. Microsoft a su maintenir le langage à la pointe tout en préservant sa stabilité et sa compatibilité ascendante. Le passage vers des styles de programmation plus déclaratifs se poursuit : les développeurs décrivent leurs objectifs plutôt que les détails d'implémentation, laissant au compilateur le soin d'optimiser l'exécution.

Le C# s'est adapté aux besoins changeants de l'industrie sans renier ses principes fondateurs de simplicité et de productivité. Guidé par une vision claire et nourri par la recherche, son développement continu en fait un outil incontournable du développement logiciel moderne.

FLAC

Josh Coalson lance le développement du FLAC en 2000. Son ambition ? Créer un format libre pour la compression audio sans perte, face aux solutions propriétaires qui dominent alors le marché. Les mélomanes comme les professionnels du son cherchent des alternatives aux formats destructifs tel le MP3, et Coalson va répondre à cette attente.

La compression sans perte du FLAC ramène un fichier audio à 50 ou 60% de sa taille d'origine, sans perdre le moindre bit d'information sonore.

Un fichier Broadcast Wave Format de 300 Mo mincit en un fichier de 100 Mo après compression. Une fois décodé, le résultat est strictement identique à l'original sur le plan acoustique. Cette capacité distingue radicalement le FLAC des formats à perte qui sacrifient des informations pour gagner de l'espace.

Le projet rejoint la fondation Xiph.Org en 2003. Cette organisation à but non lucratif héberge déjà des formats multimédia libres comme Vorbis, Opus ou Theora. L'intégration du FLAC dans cet écosystème de standards ouverts renforce sa légitimité. La version 1.2.1 sort en 2007 et marque une forme de maturité : les spécifications n'ont depuis connu aucune modification majeure. Les outils de référence continuent leur évolution, la version 1.3.2 paraît en janvier 2017.

Les concepteurs du FLAC ont privilégié une approche pragmatique. Le format consomme peu de mémoire au décodage et repose uniquement sur des opérations en nombres entiers. Ces choix techniques autorisent son déploiement sur des systèmes aux ressources modestes. Le flux de données s'organise en métadonnées suivies de trames audio. Chaque trame contient un en-tête avec un code de synchronisation de 14 bits, des sous-trames, des octets de bourrage et un pied de trame.

Le format, flexible, accepte jusqu'à 8 canaux audio et des échantillons codés entre 4 et 32 bits. Pour la stéréo, quatre modes d'encodage coexistent : indépendant, gauche/côté, droite/côté et milieu/côté. La compression s'adapte donc à la nature du signal sonore traité.

L'industrie musicale adopte le FLAC par étapes successives. D'abord cantonné aux cercles d'audiophiles et plateformes alternatives, le format gagne du terrain avec l'émergence du streaming haute qualité. TIDAL et Deezer Elite lancent leurs services en 2014, Qobuz suit le mouvement. La distribution en ligne évolue parallèlement : 7digital et Bandcamp proposent désormais le téléchargement en FLAC.

Le support natif par les systèmes d'exploitation représente une étape décisive. Android l'intègre dès la version 3.1, Windows 10 emboîte le pas. Apple maintient sa stratégie propre avec l'ALAC, d'abord propriétaire puis libéré en 2011. Les navigateurs web récents, Chrome 56 et Firefox 51, lisent directement les fichiers FLAC.

La préservation numérique trouve dans le FLAC des atouts considérables. Chaque fichier embarque dans son en-tête une empreinte MD5 des données audio originales pour vérifier leur intégrité. Des sommes de contrôle CRC identifient les trames corrompues lors de la transmission en flux. Ces mécanismes de validation intégrés confèrent au FLAC un avantage sur le format WAV qui nécessite des vérifications externes.

Les institutions patrimoniales s'intéressent au potentiel du format. La

division audio et vidéo de la Bibliothèque du Congrès américain envisage son usage pour la consultation des enregistrements sonores dès 2005. L'Internet Archive l'emploie dans ses Archives de Musique Live, où concerts et sessions figurent souvent en FLAC et WAV. L'Union Européenne de Radio-Télévision l'utilise pour distribuer des enregistrements de concerts via Musipop : les fichiers WAV originaux sont convertis en FLAC avant leur transmission par satellite.

Des outils spécialisés accompagnent l'essor du format. VLC, Clementine ou Foobar2000 assurent la lecture sur différentes plateformes. Des utilitaires de conversion comme dbPowerAMP ou FFmpeg facilitent les échanges avec d'autres formats. Mp3tag ou Kid3 gèrent les métadonnées.

L'Internet Engineering Steering Group approuve en 2016 la création du groupe de travail CELLAR pour formaliser les spécifications du FLAC. Cette démarche vise à établir un standard officiel et renforcer la pérennité du format dans le paysage audio numérique.

Un projet libre comme FLAC peut s'imposer comme standard de fait dans son domaine. L'absence de brevets ou de redevances sur son utilisation favorise son adoption par l'industrie musicale et les institutions culturelles. Ce format reste un choix pertinent pour l'archivage numérique et la distribution de musique haute qualité.

JSON

Douglas Crockford travaillait chez State Software et cherchait un moyen simple de faire communiquer un serveur avec un navigateur web. Il remarqua que JavaScript possédait une syntaxe pratique pour représenter les objets : des paires clé-valeur entre accolades. L'idée lui vint d'extraire cette notation du langage pour en faire un format d'échange de données à part entière.

En 2001, Crockford formalisa ce sous-ensemble de JavaScript sous le nom de JSON, pour JavaScript Object Notation. Il publia la spécification sur json.org et proposa des implémentations dans plusieurs langages. Le format tenait en quelques règles : six types de données suffisaient (chaînes de caractères, nombres, booléens, tableaux, objets et la valeur nulle) pour représenter des structures complexes. Cette simplicité brutale tranchait avec la verbosité de XML, omniprésent dans les échanges de données.

La standardisation officielle arriva en 2006 avec la RFC 4627 de l'IETF. Le document établissait le type MIME `application/json` et définissait

la syntaxe précise. D'autres versions suivirent : RFC 7158 en 2013, RFC 7159 en 2014, puis RFC 8259 en 2017. Chaque révision clarifiait des points techniques et ajustait la spécification aux besoins qui émergeaient sur le terrain.

JSON s'est imposé parce qu'il répondait à des besoins concrets. Sa syntaxe minimaliste restait lisible pour un humain tout en étant facile à analyser par une machine. L'absence de balises réduisait la taille des messages échangés. Et surtout, il se mariait naturellement avec JavaScript, le langage qui dominait déjà le développement web côté client. Les développeurs n'avaient rien de nouveau à apprendre car ils utilisaient des structures qu'ils manipulaient quotidiennement.

Les géants du web l'adoptèrent les uns après les autres. Yahoo ! figura parmi les premiers à l'intégrer dans ses services, suivi par Google, Facebook et Twitter. Cette adoption massive transforma JSON en standard de fait pour les API web. Les architectures REST, qui devenaient la norme pour concevoir des services web, s'accordaient parfaitement avec ce format léger.

L'écosystème autour de JSON se développa à une vitesse étonnante. Des bibliothèques apparurent dans tous les langages majeurs. Des variantes spécialisées émergèrent : GeoJSON pour les données géographiques, JSON Schema pour valider la structure des documents. Les bases de données NoSQL comme MongoDB ou CouchDB l'adoptèrent comme format de stockage natif, étendant son influence au-delà du simple transport de données.

Les frameworks JavaScript modernes comme Angular ou React manipulaient JSON de manière native. Le développement mobile tira parti de sa légèreté pour économiser la bande passante. JSON devint progressivement un composant invisible mais omniprésent de l'infrastructure web, présent dans presque chaque échange entre applications.

Cette simplicité révéla aussi ses limites. L'impossibilité d'ajouter des commentaires gênait la documentation. La représentation des nombres posait des questions de précision liées aux calculs en virgule flottante. L'absence de format standard pour les dates obligeait les développeurs à établir leurs propres conventions. Des variantes comme JSON5 ou JSONC apparurent pour combler ces manques, au risque de fragmenter l'écosystème.

Une analyse récente des fichiers JSON publics montre que les développeurs exploitent intensivement la capacité du format à représenter des structures imbriquées. Les objets contenus dans des tableaux eux-mêmes nichés dans d'autres objets sont devenus courants. Cette flexibilité autorise une grande expressivité tout en maintenant une certaine régularité qui facilite le traitement automatisé.

JSON a modifié les pratiques de développement. Il a favorisé l'essor d'architectures découplées où les composants communiquent par des interfaces

bien définies. Sa facilité de manipulation a encouragé l'adoption de paradigmes fonctionnels et réactifs. Les outils de développement ont intégré des fonctionnalités spécifiques pour travailler avec ce format, de la coloration syntaxique à la validation automatique.

La spécification de JSON est volontairement minimaliste. Ses concepteurs ont choisi de préserver la simplicité originelle plutôt que d'ajouter des fonctionnalités. Les besoins spécifiques sont couverts par des extensions et des outils complémentaires qui n'affectent pas la compatibilité de base. Cette stabilité assure la pérennité du format dans un environnement technologique qui évolue constamment.

Le format n'a pas été imposé par un organisme de standardisation mais adopté naturellement parce qu'il répondait à un besoin réel. Sa réussite tient à son adéquation avec les pratiques existantes et à sa capacité à rester simple quand tant d'autres solutions cherchaient la complexité. JSON est devenu un pilier de l'infrastructure web moderne, un de ces éléments qu'on ne remarque plus tant ils paraissent évidents.

BitTorrent

Début 2000, Internet montrait certaines limites. Transférer un fichier volumineux relevait du parcours du combattant : le courrier électronique saturait, les serveurs FTP s'effondraient sous la charge. Bram Cohen avait eu son compte des startups de la bulle Internet. Ces boîtes promettaient monts et merveilles, brûlaient l'argent des investisseurs, puis disparaissaient avant d'avoir publié quoi que ce soit. Il décida de prendre les choses en main.

Son expérience professionnelle lui avait appris une chose : fragmenter les fichiers pour les stocker de manière sécurisée marchait bien. Cette idée le taraudait. Et si on appliquait ce principe au partage de données ? Au lieu qu'un serveur unique distribue un fichier à des centaines d'utilisateurs, pourquoi ne pas découper ce fichier en morceaux et laisser chacun partager ce qu'il possède localement ?

C'est ainsi que naquit BitTorrent en 2001. Le concept tenait en quelques lignes : un fichier se découpe en segments. Vous téléchargez ces segments depuis plusieurs sources à la fois. Pendant ce temps, vous partagez déjà les morceaux que vous avez reçus avec d'autres. Plus il y a de monde, mieux ça marche. Cohen baptisa ce mécanisme de réciprocité *tit-for-tat*, un nom qui résume bien l'esprit : donnant-donnant.

L'été 2001 vit sortir la première version bêta. Cohen présenta son pro-

tocole l'année suivante lors d'une conférence. Son ambition initiale était de distribuer des distributions Linux. Pourtant, l'architecture technique cachait une vraie finesse. Les fichiers .torrent contiennent juste les métadonnées : nom du fichier, taille, découpage, adresse du *tracker*. Ce tracker coordonne les échanges entre participants sans jamais stocker le moindre octet de contenu.

Deux algorithmes firent la différence. Le premier, dit *rarest first*, privilégie les segments les moins répandus dans le réseau. Le second, le *choking*, régule les transferts en favorisant ceux qui partagent activement. Ces mécanismes garantissent qu'un fichier reste disponible quand des milliers d'utilisateurs le téléchargent simultanément.

Le succès fut au rendez-vous. Dès 2005, BitTorrent représentait une part conséquente du trafic Internet mondial. Certains l'utilisèrent pour des contenus piratés, comme avec tous les protocoles de partage avant lui. D'autres y virent une aubaine. Opera intégra BitTorrent dans son navigateur cette année-là et s'en servit pour diffuser ses mises à jour. Blizzard Entertainment l'adopta pour distribuer les correctifs de World of Warcraft. L'Internet Archive suivit en 2012 pour rendre accessibles ses collections numériques.

Côté mécanique interne du protocole, les fichiers se découpent en blocs de 256 Ko. Chaque bloc reçoit un identifiant unique via un hachage SHA-1 qui garantit son intégrité. Le système maintient une liste dynamique des pairs connectés et ajuste en permanence les transferts selon les performances de chaque connexion. Cette décentralisation assure robustesse et capacité à passer à l'échelle.

Cohen avait visé juste : un système pair-à-pair pouvait résoudre les problèmes de distribution massive. Les économies de bande passante séduisirent les entreprises. Distribuer une mise à jour logicielle de plusieurs gigaoctets à des millions d'utilisateurs coûtait une fortune avec l'approche traditionnelle. BitTorrent changeait la donne.

Le code source ouvert favorisa l'émergence d'une communauté de développeurs. La simplicité relative du protocole accéléra son adoption. Les mécanismes d'incitation au partage garantirent son efficacité pratique. Ces trois éléments expliquent pourquoi BitTorrent s'imposa aussi largement.

Plus de vingt ans après, le protocole tourne toujours. Les usages ont évolué, les technologies concurrentes se sont multipliées, mais BitTorrent est encore pertinent pour distribuer des fichiers lourds. Une idée née de la frustration face aux limites techniques peut transformer durablement l'usage d'Internet.

Apple Mac OS X

Apple traversait une période difficile dans les années 1990. Son système d'exploitation Mac OS, né en 1984 avec le premier Macintosh, accusait un retard technique face à la concurrence. Conçu à l'origine pour un ordinateur de 128 Ko de mémoire, il ignorait le multitâche préemptif et la protection mémoire. Le MultiFinder, arrivé en 1988, n'apportait qu'une forme rudimentaire de multitâche coopératif. L'architecture vieillissait mal. Vers le milieu des années 1990, Apple s'appuyait sur un code vieux de dix ans, initialement écrit pour les processeurs Motorola 68000, tant bien que mal adapté aux PowerPC. Une partie du noyau tournait même dans un émulateur 68K, ce qui n'arrangeait rien aux performances.

Apple lance alors en 1994 le projet Copland, avec l'ambition de créer un système moderne qui garderait l'interface et la compatibilité Mac OS. La société multiplie les présentations aux développeurs, publie des documentations, mais Copland ne parvient jamais à une stabilité acceptable. Le projet s'arrête en 1996. Apple se retrouve dans l'urgence.

La solution vient d'ailleurs, par le rachat de NeXT, la société que Steve Jobs avait fondée après son départ d'Apple en 1985. NeXT avait bâti NEXTSTEP, un système remarquable basé sur le micronoyau Mach de l'université Carnegie Mellon. NEXTSTEP mariait ce noyau moderne avec une couche BSD et proposait dès 1989 le multitâche préemptif, la protection mémoire, le support de plusieurs architectures processeur et un environnement de développement orienté objet en Objective-C.

Transformer NEXTSTEP en Mac OS X demande un travail considérable. Apple actualise le code Mach 2.5 vers la version 3.0, met à jour la partie BSD avec du code de 4.4BSD et FreeBSD, ajoute le support des systèmes de fichiers HFS et des protocoles réseau Apple. L'équipe développe I/O Kit, un nouveau système de pilotes qui remplace l'ancien DriverKit. Cette couche, écrite dans un sous-ensemble du C++, autorise une approche orientée objet pour gérer les périphériques.

L'architecture de Mac OS X repose sur la modularité. Le noyau XNU (*X is not UNIX*) combine trois composants : Mach gère les ressources système, BSD assure la compatibilité UNIX et les services réseau, I/O Kit s'occupe des pilotes. Cette organisation facilite la maintenance et les évolutions. Les pilotes se chargent dynamiquement sous forme de KEXTs (*Kernel Extensions*), des modules qu'on ajoute sans recompiler le noyau.

Pour la transition des applications, Mac OS X propose plusieurs environnements. Cocoa, héritier des bibliothèques OpenStep de NeXT pour créer

des applications natives. Carbon offre une version modernisée des API Mac OS classiques et facilite le portage des logiciels existants. Un environnement Classic exécute les anciennes applications Mac OS sans modification.

Les innovations vont au-delà du noyau. Le système introduit Mach-O, un format binaire universel capable de contenir du code pour différentes architectures dans un fichier. Cette caractéristique simplifie la distribution des logiciels et prépare les transitions matérielles futures. Le démarrage s'appuie sur OpenFirmware pour les PowerPC, puis sur EFI pour les processeurs Intel, apportant des capacités étendues par rapport au BIOS traditionnel.

La gestion mémoire témoigne du soin apporté aux performances. Sur les systèmes x86, Mac OS X adopte une approche 4/4 où l'espace utilisateur et le noyau ne sont pas mappés simultanément, contrairement à la répartition classique 3/1 ou 2/2 Go. Cette conception tire mieux parti de la mémoire disponible.

Le passage aux processeurs Intel en 2005-2006 démontre la souplesse de l'architecture. La technologie Rosetta, basée sur QuickTransit de Transitive, exécute les applications PowerPC sur les nouveaux processeurs x86 par recompilation dynamique. Le noyau 32 bits continue de fonctionner sur les machines 64 bits en mode de compatibilité, préservant la compatibilité avec les extensions existantes tout en donnant accès aux fonctionnalités 64 bits.

L'influence de Mac OS X dépasse les ordinateurs de bureau. Le système sert de base à iOS pour l'iPhone et l'iPod touch, en adaptant le noyau XNU et certains composants à l'architecture ARM. Cette réutilisation du code donne à Apple une cohérence technologique entre ses plateformes.

La publication du code source de Darwin, le cœur de Mac OS X, sous licence Apple Public Source License en 1999 marque un tournant dans la stratégie d'Apple. Cette ouverture autorise les développeurs à étudier l'implémentation du système et à y contribuer. Des projets comme PureDarwin montrent qu'on peut créer des systèmes dérivés.

Mac OS X réalise une synthèse entre l'héritage UNIX, les innovations de NeXT et le savoir-faire d'Apple en matière d'interfaces. Le système a permis à Apple de moderniser sa plateforme en préservant la compatibilité avec les applications existantes. Cette base technique solide a soutenu l'expansion d'Apple vers de nouveaux marchés et continue d'évoluer avec les besoins des utilisateurs et les avancées technologiques.

SHA-256

La fin des années 1970 marque les débuts des fonctions de hachage cryptographique. Ces algorithmes, qui transforment n'importe quelle donnée en une empreinte numérique de taille fixe, allaient devenir des briques majeures de la sécurité informatique.

En 1976, Whitfield Diffie et Martin Hellman publient leur article fondateur sur la cryptographie à clé publique. Ils y expliquent qu'une fonction de hachage à sens unique est nécessaire pour construire des signatures numériques. Les premiers travaux concrets arrivent peu après : Michael Rabin propose une conception basée sur le chiffrement DES qui produit un résultat de 64 bits. Gideon Yuval démontre qu'on peut trouver des collisions en exploitant le paradoxe des anniversaires avec une complexité de `2^(n/2)`. Ralph Merkle établit quant à lui les exigences de base : résistance aux collisions, aux pré-images secondaires et aux pré-images.

Les années 1980 voient fleurir de nombreuses propositions. La communauté cryptographique réalise l'importance de ces primitives pour sécuriser les communications numériques. Ivan Damgård formalise en 1987 la définition de la résistance aux collisions. Deux ans plus tard, Moni Naor et Moti Yung introduisent une variante appelée Universal One Way Hash Functions.

Au tournant des années 1990, Ronald Rivest crée MD5, une évolution de MD4 optimisée pour le logiciel. Cette fonction connaît un succès rapide : elle s'avère environ dix fois plus rapide que DES en implémentation logicielle. Surtout, MD5 échappe aux restrictions d'exportation qui pèsent sur les algorithmes de chiffrement et peut être utilisée librement.

La National Security Agency développe alors la famille SHA (Secure Hash Algorithm). Le NIST publie SHA-0 en 1993. Mais l'agence découvre une vulnérabilité deux ans plus tard et sort une version corrigée baptisée SHA-1. En 2001, face aux progrès de la cryptanalyse et à la montée en puissance des ordinateurs, la NSA conçoit la famille SHA-2. SHA-256 en fait partie.

SHA-256 produit une empreinte de 256 bits et travaille sur des mots de 32 bits. L'algorithme traite les messages par blocs de 512 bits après un *padding* spécifique. Le processus comprend une expansion de message et une compression itérative basée sur une fonction de Merkle-Damgård. Cette construction garantit que la sécurité de la fonction de compression s'étend à celle de la fonction de hachage complète.

Les premières fissures dans MD5 apparaissent en 1992. Den Boer et Bosselaers trouvent des collisions pour la fonction de compression. En 1996, Hans Dobbertin découvre des collisions pour MD5 avec une valeur initiale aléatoire. Ces résultats n'inquiètent pas encore vraiment la communauté.

Tout bascule en 2004. L'équipe de Xiaoyun Wang réalise une percée en perfectionnant la cryptanalyse différentielle. Leurs travaux permettent de trouver des collisions pour MD5 en quelques millisecondes. Les techniques développées par Wang réduisent considérablement la marge de sécurité de SHA-1. La communauté cryptographique se lance dans une vague de recherches.

Le 31 décembre 2008, Alexander Sotirov et son équipe frappent un grand coup. Ils créent un certificat CA malveillant qui exploite les collisions de MD5. Cette attaque permet théoriquement d'usurper l'identité de n'importe quel site web. L'industrie comprend qu'il faut abandonner MD5, malgré sa présence massive dans les systèmes existants.

Le NIST lance en novembre 2007 une compétition pour sélectionner SHA-3, un nouvel algorithme de hachage standard. L'objectif : diversifier les options disponibles et préparer le remplacement de SHA-2 si besoin. La compétition attire 64 soumissions dont 51 sont retenues pour le premier tour. En juillet 2009, 14 candidats accèdent au second tour.

Pendant ce temps, SHA-256 tient bon. Les meilleures attaques connues ne compromettent qu'un nombre limité d'étapes de l'algorithme. SHA-256 résiste aux techniques qui ont cassé MD5 et affaibli SHA-1. Cette solidité explique son adoption croissante dans de nombreux protocoles et applications de sécurité.

L'arrivée de Bitcoin en 2009 donne une dimension inattendue à SHA-256. La preuve de travail de la cryptomonnaie repose sur la recherche de collisions partielles, exploitant la résistance aux préimages de la fonction. Cette application démontre la polyvalence de SHA-256.

On retrouve aussi SHA-256 dans l'Internet des objets, les systèmes embarqués distribués, la génération de nombres aléatoires et le chiffrement de données. Cette diversification témoigne de sa maturité et de la confiance qu'il inspire.

Cette histoire montre qu'il faut toujours disposer d'alternatives fiables quand des faiblesses apparaissent dans les standards existants. Elle rappelle l'intérêt de maintenir des options standardisées. La longévité de SHA-256 prouve qu'une conception rigoureuse anticipe les évolutions de la cryptanalyse et de la puissance de calcul disponible.

Microsoft .NET Framework

Lors de la Professional Developers Conference de Microsoft en 2000, la société dévoila alors .NET, un nouveau *framework* pour Windows qui allait redéfinir la manière dont on construisait les logiciels sur sa plateforme. Ce projet, qui avait démarré sous l'appellation peu évocatrice de « Next Generation Windows Services », représentait plus qu'une simple mise à jour des outils existants. Les développeurs reçurent dès cette année-là les premières versions bêta de .NET 1.0.

Microsoft voulait résoudre plusieurs problèmes à la fois. D'une part, Java gagnait du terrain et la société de Redmond avait besoin d'une réponse crédible. D'autre part, le développement sur Windows restait fragmenté entre différentes technologies qui ne communiquaient pas toujours bien entre elles. L'idée était de créer quelque chose de plus cohérent, qui simplifierait vraiment le travail des programmeurs. Au cœur de cette nouvelle plateforme se trouvait le Common Language Runtime, un environnement d'exécution qui prenait en charge automatiquement des tâches auparavant complexes comme la gestion de la mémoire ou les aspects de sécurité.

La première mouture commerciale arriva en 2002 avec Visual Studio .NET et le framework 1.0. Cette version posait les bases d'une architecture qui allait traverser les années suivantes : une bibliothèque de classes commune, un système de types unifié, et la possibilité d'écrire du code dans plusieurs langages comme C# ou Visual Basic .NET. La version 1.1 suivit en 2003, apportant son lot d'améliorations de performances et de nouvelles fonctionnalités.

Avec Visual Studio 2005 et le framework 2.0, les choses prirent une autre dimension. Les génériques firent leur apparition, autorisant les développeurs à écrire du code plus robuste et réutilisable. Cette version consolida .NET comme une plateforme incontournable. L'année 2008 marqua un nouveau tournant avec le « multi-targeting », qui donnait la liberté de cibler simultanément différentes versions du framework – la 2.0, la 3.0 et la 3.5.

Les versions se succédèrent ensuite à un rythme régulier. Visual Studio 2010 intégra .NET 4.0, puis la version 2012 accueillit la 4.5. Cette progression permit à la plateforme de s'adapter aux exigences nouvelles du développement logiciel, entre autres dans le web et le cloud.

L'architecture technique de .NET repose sur un empilement de couches bien pensées. À la base, le CLR exécute le code et gère automatiquement la mémoire grâce à son ramasse-miettes. Au-dessus viennent les bibliothèques de classes, qui fournissent tout le nécessaire pour bâtir des applications. Selon les besoins, le framework propose différents environnements : applications console pour les traitements sans interface, Windows Forms pour

les logiciels de bureau classiques, ASP.NET pour le web.

Le modèle de compilation de .NET se distingue par une approche originale. Le code source ne se transforme pas directement en langage machine. Il passe d'abord par une étape intermédiaire, le MSIL (Microsoft Intermediate Language). Cette stratégie offre quelques bénéfices : le code est portable entre différentes architectures, on peut l'optimiser au moment de l'exécution, et les langages de programmation sont interopérables.

Cette interopérabilité entre langages constitue d'ailleurs l'un des atouts de .NET. Au-delà de C# et Visual Basic .NET, la plateforme accueille F#, Iron Python ou encore Iron Ruby. Le Common Type System définit comment déclarer et utiliser les types de données, tandis que le Common Language Specification établit les règles garantissant que tous ces langages peuvent travailler ensemble.

Anders Hejlsberg a joué un rôle déterminant dans cette histoire. Architecte en chef de C#, il avait auparavant travaillé chez Borland sur Turbo Pascal et Delphi. Cette expérience nourrit la conception du framework. Sous son impulsion, C# devint le langage phare de .NET, intégrant régulièrement des innovations comme LINQ qui permettait d'écrire des requêtes directement dans le code.

L'influence de .NET sur l'industrie du logiciel fut massive. La plateforme transforma la façon dont les entreprises développaient leurs applications Windows, en leur donnant un environnement réellement productif et moderne. La transition vers le web devint plus simple avec ASP.NET, qui rendait moins ardue la création d'applications web solides.

Le framework introduisit des concepts avancés que peu de plateformes offraient à l'époque. La programmation orientée aspect, la réflexion, les attributs : autant d'outils qui autorisaient la construction d'applications plus sophistiquées. La gestion automatique de la mémoire libéra les programmeurs de tâches ingrates et sources d'erreurs. Le déploiement devint moins problématique grâce à l'installation automatique des composants nécessaires.

Microsoft enrichit progressivement .NET pour suivre les évolutions technologiques. Le support du cloud computing, le développement mobile, les applications universelles Windows : la plateforme s'adapta continuellement aux nouveaux paradigmes et aux attentes des développeurs.

La création de .NET représenta un investissement colossal. Des centaines d'ingénieurs y travaillèrent pendant des années. Cet effort paya : le framework devint un standard de fait pour le développement sur Windows, adopté par des millions de développeurs à travers le monde. Son influence perdure avec .NET Core, sa version moderne et multiplateforme.

Mozilla Firefox

En 1993, le navigateur Mosaic voit le jour au National Center for Supercomputing Applications. Son succès ne tarde pas. Marc Andreessen, l'un des créateurs, fonde l'année suivante Netscape Communications Corporation. Le nouveau navigateur, Netscape Navigator, s'empare du marché du Web.

Microsoft réplique en développant Internet Explorer qu'il intègre directement à Windows 95. La stratégie fonctionne. Netscape perd progressivement du terrain. Face à ce recul, l'entreprise prend en 1998 une décision audacieuse : elle rend public le code source de son navigateur à travers le projet Mozilla. L'organisation mozilla.org naît pour coordonner ce développement communautaire.

La situation se complique en 1999 quand AOL rachète Netscape. L'intégration se passe mal. Mitchell Baker, juriste chez Netscape puis chez AOL, dirige mozilla.org avec le titre plutôt original de « Chief Lizard Wrangler ». Son équipe continue le développement tout en gardant son autonomie face à AOL.

Après quatre ans de travail, Mozilla sort en 2002 dans sa première version stable. Le navigateur regorge de fonctionnalités : onglets, blocage des pop-ups, gestionnaire de téléchargements. Mais son interface trop chargée rebute le grand public. Une petite équipe décide alors de repartir sur des bases plus simples.

Le nouveau projet passe par plusieurs noms : Phoenix, Firebird, avant de devenir Firefox pour éviter des conflits de marques. La version 1.0 débarque en novembre 2004. Le succès est immédiat. Firefox apporte des innovations qui vont faire école : les onglets intégrés nativement, un système d'extensions modulaire, une interface épurée. L'approche tranche avec ce qui existe alors.

AOL abandonne Netscape Navigator en 2003. Mozilla se transforme en fondation indépendante à but non lucratif, avec 2 millions de dollars d'AOL comme capital de départ. La Mozilla Foundation peut ainsi développer Firefox en toute liberté, fidèle aux principes du logiciel libre et d'un Web ouvert.

Le navigateur arrive au bon moment. Internet Explorer stagne techniquement. Firefox propose une vraie alternative, plus sûre, qui respecte mieux la vie privée. Son développement ouvert attire une communauté mondiale de contributeurs. La fondation signe des accords avec des moteurs de

recherche, notamment Google, ce qui génère des revenus sans compromettre l'indépendance du projet.

De 2004 à 2008, Firefox grignote des parts de marché à Internet Explorer. Il atteint 20% du marché mondial en 2008. Un projet open source concurrence frontalement un logiciel propriétaire dominant. Firefox pousse les standards du Web et l'interopérabilité, là où Microsoft favorisait son écosystème fermé.

Google Chrome apparaît en 2008. Ce navigateur, basé d'abord sur WebKit puis sur Blink, s'impose peu à peu. Firefox conserve toutefois ses utilisateurs fidèles, attachés à sa vision d'un Web ouvert et respectueux des libertés individuelles.

L'explosion du mobile amène Mozilla à lancer Firefox OS en 2013. Ce système d'exploitation mobile, construit sur les technologies Web, veut offrir une alternative ouverte à iOS et Android. Malgré des partenariats avec des fabricants, Firefox OS ne décolle pas. Le projet s'arrête en 2015.

En 2024, Firefox compte environ 200 millions d'utilisateurs. Le navigateur innove toujours sur la protection de la vie privée, les performances, le respect des standards. La Mozilla Foundation poursuit son combat pour un Internet ouvert et accessible à tous.

Firefox montre qu'un projet open source peut bousculer un marché dominé par des géants commerciaux. Le navigateur a prouvé l'importance d'une gouvernance transparente et d'une communauté engagée. Le modèle économique de Mozilla fait cependant figure d'exception. La fondation tire des revenus substantiels de ses partenariats avec les moteurs de recherche sans renier sa mission non lucrative. Cet équilibre reste rare dans le monde du logiciel.

Apple Xserve

En mai 2002, Apple revient sur le marché des serveurs avec le Xserve, un ordinateur pensé pour les baies de serveurs. L'entreprise n'avait plus tenté l'aventure depuis l'échec cuisant de ses Network Servers dans les années 1990. À cette époque, les Mac équipaient bon nombre d'entreprises du Fortune 500, mais ils restaient aux portes de leurs départements informatiques, sans jamais vraiment y pénétrer.

Le Xserve adopte un format 1U compact de 4,4 cm de hauteur. La première génération embarque des processeurs PowerPC G4 à 1 GHz et de la mémoire DDR, une première chez Apple. L'architecture du système repose

sur quatre baies de disques durs extractibles à chaud, chacune disposant de son propre bus Ultra ATA-100. Ces modules propriétaires existent en 60 ou 120 Go, soit atteindre 480 Go en tout au maximum.

Côté connectique, l'appareil propose deux ports Ethernet Gigabit, trois ports FireWire, deux USB et un port série pour les connexions en terminal. Le panneau avant affiche en couleur l'activité des disques, la charge des processeurs et l'état du réseau. Apple lance deux versions : 2 999 dollars pour le monoprocesseur, 3 999 dollars pour le biprocesseur, licence illimitée de Mac OS X Server comprise.

La machine évolue vite. Dès 2004, le Xserve G5 prend le relais avec des PowerPC G5 montant jusqu'à 2 GHz. Cette montée en puissance impose une refonte complète du châssis pour évacuer la chaleur, ce qui réduit le nombre de baies. Cette version introduit la mémoire ECC, une nouveauté dans le catalogue Apple, et passe aux disques SATA.

Apple complète son offre avec le Xserve RAID, un système de stockage en rack 3U. Ce périphérique accueille 14 modules de 180 Go et dialogue avec le serveur via des canaux Fibre Channel à 2 Gb/s, pour des débits de 400 Mo/s. Son prix défie la concurrence : en 2004, le coût par gigaoctet s'établit à 3,14 dollars, loin derrière Dell (9,05 dollars) et HP (11,39 dollars).

La bascule vers Intel intervient en 2006. Le Xserve adopte des Xeon 64 bits grimpant jusqu'à 3 GHz, multipliant par cinq les performances du G5. Ces processeurs dégagent moins de chaleur, ce qui autorise l'ajout d'alimentations redondantes et fait grimper la capacité de stockage à 2,5 To. Les options se multiplient : graveur DVD double couche, 32 Go de RAM, carte RAID intégrée.

Les dernières déclinaisons sortent entre 2008 et 2009 avec des Xeon quadricœurs de série 5400. Ces machines atteignent des sommets grâce à leur conception aboutie : bus frontaux indépendants à 1 600 MHz, mémoire DDR2 ECC à 800 MHz, connecteurs PCI Express 2.0. Le stockage franchit un cap avec des disques SAS à 15 000 tr/min avec un débit de 125 Mo/s et des temps d'accès de 3,5 ms.

Mac OS X Server constitue l'atout majeur du Xserve. Ce système certifié UNIX intègre partage de fichiers et d'imprimantes, hébergement web, messagerie, streaming multimédia. La version 10.5 Leopard Server enrichit les fonctions collaboratives avec iCal Server pour les calendriers, Wiki Server pour le travail d'équipe et Podcast Producer pour publier automatiquement des contenus multimédias.

Le support technique s'adapte aux besoins des entreprises : assistance téléphonique et e-mail disponible jour et nuit. Des kits de pièces détachées permettent aux techniciens d'intervenir sur site. L'application Server Monitor surveille en continu les composants critiques : températures, ventila-

teurs, alimentations, disques durs, interfaces réseau.

Apple cesse la commercialisation du Xserve RAID en février 2008, lui préférant des solutions Promise Technology certifiées pour Xsan 2. Le serveur Xserve disparaît en janvier 2011 mais il reste un bel exemple de savoir-faire technique dans le domaine des serveurs d'entreprise. La société recommande alors d'utiliser des Mac mini ou Mac Pro configurés en serveurs. Ce retrait traduit un changement de priorités chez Apple, qui se tourne davantage vers le grand public et les professionnels que vers les départements informatiques. Durant neuf années, Xserve a proposé une alternative élégante aux solutions classiques, mariant performances, simplicité d'usage et encombrement réduit.

Tor

Dans les années 1990, le Naval Research Laboratory américain travaillait sur un problème qui semblait assez éloigné des préoccupations du grand public : comment protéger les communications de ses agents à travers le monde ? La solution trouvée s'appelait « Onion Routing », le routage en oignon. Personne n'imaginait alors que cette technologie militaire deviendrait l'un des outils les plus importants pour défendre la vie privée sur Internet.

Le principe technique est ingénieux. Chaque message traverse plusieurs serveurs intermédiaires, chiffré en couches successives comme les pelures d'un oignon. À chaque étape, un serveur retire une couche et transmet le paquet au relais suivant, sans jamais connaître ni l'origine ni la destination finale. Un observateur qui intercepte le trafic ne voit qu'un nœud parmi d'autres, incapable de remonter la chaîne complète.

Cette époque voyait aussi l'émergence des cypherpunks, ces technologues libertariens convaincus que le chiffrement représentait un rempart contre la surveillance étatique. La rencontre entre ces militants et les chercheurs militaires, lors d'un séminaire universitaire, a créé une alliance improbable. Les militaires avaient besoin d'un vaste réseau d'utilisateurs civils pour noyer leurs communications sensibles dans un trafic abondant. Les cypherpunks voulaient démocratiser les technologies de protection de la vie privée. Chacun y trouvait son compte.

En 2002, le protocole a été réimplémenté pour le grand public et publié sous licence libre. Cette transparence du code source comptait beaucoup : comment faire confiance à un outil de sécurité financé par le département d'État américain sans pouvoir vérifier son fonctionnement ? La communauté pouvait désormais l'auditer librement.

Les débuts ont soulevé des questions techniques épineuses. Fallait-il ajouter ces données factices qui masquent les véritables communications ? La sécurité théorique y gagnait, mais les connexions devenaient d'une lenteur insupportable. Les développeurs ont tranché : un réseau trop lent ne séduirait jamais assez d'utilisateurs. Or la force de Tor repose justement sur le nombre : plus les profils sont variés, mieux les communications sensibles se fondent dans la masse du trafic quotidien.

Cette philosophie a guidé toute l'évolution du projet. Tor devait rester accessible au plus grand nombre tout en maintenant un haut niveau de sécurité. Le pari a fonctionné. Journalistes protégeant leurs sources, militants des droits humains dans des régimes autoritaires, entreprises soucieuses de leurs secrets industriels, simples citoyens refusant le traçage publicitaire : tous ont trouvé dans ce réseau une réponse à leurs besoins.

Le réseau permet aussi d'héberger des services cachés, reconnaissables à leurs adresses en « .onion ». L'emplacement physique de ces sites est masqué, ce qui soulève des controverses. Silk Road, cette place de marché illégale fermée par le FBI, utilisait Tor. Mais réduire le réseau à ses usages criminels serait malhonnête. Les révélations d'Edward Snowden en 2013 ont montré que la NSA échouait à compromettre Tor à grande échelle, confirmant sa robustesse technique face à l'adversaire le plus puissant du monde.

L'architecture repose sur des milliers de serveurs volontaires dispersés à travers le monde. Des individus, des associations, parfois des universités, font tourner ces relais par conviction : préserver un Internet libre. Cette infrastructure décentralisée constitue un rempart technique contre la surveillance généralisée. Elle illustre aussi qu'une gouvernance alternative des réseaux est possible, en dehors des logiques commerciales ou étatiques.

L'innovation majeure de Tor ne concerne pas tant le chiffrement des contenus que la protection des métadonnées. Qui parle à qui, quand, combien de fois ? Ces informations révèlent souvent plus que le contenu des messages eux-mêmes. En s'attaquant à ce problème structurel, le réseau a inspiré de nombreux autres projets de protection de la vie privée.

La pérennité de Tor tient à un paradoxe fécond. Le département d'État américain continue de financer son développement pour promouvoir la liberté d'expression dans les pays censurant Internet. Des militants l'utilisent parfois pour échapper à la surveillance occidentale. Cette dualité, loin d'affaiblir le projet, renforce sa légitimité et sa résilience. Servir simultanément des intérêts étatiques et citoyens n'est pas une contradiction mais une condition de survie.

Face aux limites des modèles mathématiques abstraits, les développeurs ont privilégié l'observation empirique des menaces réelles et des usages

concrets. Cette approche pragmatique a produit un système imparfait mais efficace contre la surveillance de masse. Les défis techniques initiaux ont forgé cette méthode : mieux vaut un outil utilisable qu'une solution théoriquement parfaite mais inutilisable en pratique.

Le projet montre qu'une infrastructure alternative peut émerger des marges d'Internet pour devenir un pilier de la liberté d'expression en ligne. Des communautés techniques peuvent créer des outils servant l'intérêt général, en dehors des circuits habituels. Le routage en oignon né dans un laboratoire militaire est devenu un bien commun numérique.

Apple iTunes

En 2000, l'industrie musicale traverse une crise sans précédent. Le partage illégal de fichiers MP3 sur Internet explose, les grandes maisons de disques s'accrochent à leurs vieux modèles, et les utilisateurs désertent les magasins de disques. Steve Jobs, de retour chez Apple depuis 1997, y voit une opportunité : celle de créer un écosystème cohérent autour de la musique numérique.

L'histoire commence véritablement en 2001 avec le premier iPod. Ce baladeur élégant avait besoin d'un logiciel pour que les gens puissent y transférer leur musique. iTunes naît de cette nécessité pratique : offrir aux possesseurs de Mac un moyen simple de convertir leurs CD en fichiers numériques et de les synchroniser avec leur appareil. Rien de nouveau en soi, mais une solution qui fonctionnait vraiment.

Deux ans plus tard, en 2003, tout bascule. Apple lance l'iTunes Store, une boutique en ligne où l'on achète des morceaux à l'unité pour 99 cents. Le concept était audacieux à l'époque. Les maisons de disques vendaient des albums complets, pas des chansons isolées. Jobs doit négocier ferme avec les cinq majors du disque. Son argument ? La simplicité d'utilisation et le prix attractif détourneront les gens du piratage. Les patrons des labels, sceptiques mais dos au mur, finissent par accepter.

Le succès dépasse toutes les attentes. Un million de morceaux vendus en une semaine. Les utilisateurs apprécient de pouvoir écouter un extrait avant d'acheter, de ne payer que pour les chansons qu'ils veulent vraiment. L'interface est intuitive, sans fioritures inutiles. iTunes introduit des listes de lecture intelligentes, des classements, des suggestions personnalisées qui changent la façon dont les gens découvrent de la musique.

En octobre 2003, Apple fait un choix surprenant : adapter iTunes pour

Windows. La firme de Cupertino abandonne son exclusivité habituelle et ouvre ses portes à des millions de nouveaux utilisateurs. Cette décision stratégique élargit l'audience du service bien au-delà des aficionados de Mac. N'importe qui possédant un PC peut désormais acheter de la musique en ligne et synchroniser son iPod.

Entre 2004 et 2006, le catalogue musical s'enrichit constamment. Les accords avec des labels indépendants multiplient l'offre disponible. iTunes ne se contente plus de vendre de la musique : la plateforme accueille des podcasts, puis des séries télévisées, enfin des films. Le logiciel devient peu à peu un hub multimédia complet, loin de sa vocation première de simple gestionnaire de bibliothèque musicale.

L'App Store voit le jour en 2008, destiné aux applications pour iPhone et iPod Touch. iTunes se transforme en carrefour numérique où convergent musique, vidéos, applications, livres numériques. Le système de paiement unifié et la synchronisation automatique entre appareils séduisent toujours plus d'utilisateurs. Apple a réussi son pari : créer un écosystème dont on a du mal à sortir une fois qu'on y est entré.

L'impact sur l'industrie musicale est difficile à mesurer, mais quelques chiffres parlent d'eux-mêmes. En 2010, iTunes est le plus grand vendeur de musique aux États-Unis, devant Walmart. Les ventes physiques continuent de s'effondrer, certes, mais les revenus de la distribution numérique compensent une partie des pertes. Le service a légitimé l'achat de musique en ligne et établi un modèle économique viable pour l'ère numérique.

La technologie évolue en même temps que les usages. L'utilisateur accède à sa bibliothèque depuis n'importe quel appareil grâce au stockage dans le cloud, introduit avec iTunes Match en 2011. Des fonctionnalités de streaming apparaissent progressivement, répondant aux nouveaux modes de consommation musicale. Le format AAC choisi par Apple offre une meilleure qualité sonore que le MP3 pour une taille de fichier comparable. La protection numérique FairPlay, moins contraignante que celle des concurrents, autorise la lecture sur plusieurs appareils tout en rassurant les maisons de disques.

En 2019, Apple annonce la fin d'iTunes. Le logiciel sera séparé en applications distinctes pour la musique, les podcasts et la vidéo sur macOS. iTunes a prouvé qu'une interface soignée, associée à un modèle commercial adapté, satisfait les consommateurs sans sacrifier les intérêts des créateurs de contenus. La plateforme a redéfini la distribution de contenu numérique et inspiré d'innombrables services qui ont suivi.

Skype

Nous sommes en 1998. Niklas Zennström travaille pour Tele2, un opérateur téléphonique suédois, lorsqu'il propose à sa direction un projet de téléphonie par Internet. L'idée ne trouve pas preneur sur le moment, mais elle continue de travailler l'esprit de Zennström. Avec le Danois Janus Friis, il lance d'abord Kazaa, ce logiciel de partage de fichiers en *peer-to-peer* qui fera parler de lui, avant de revenir à son projet initial : la téléphonie sur Internet.

En 2000, le portail Everyday.com rassemble une équipe au sein de laquelle Zennström et Friis y retrouvent trois développeurs estoniens : Jaan Tallinn, Ahti Heinla et Priit Kasesalu. Toivo Annus les rejoint bientôt. Ces développeurs viennent de Bluemoon, une société estonienne où ils ont acquis leur savoir-faire technique en créant des jeux vidéo à l'époque soviétique. Le groupe forme le noyau qui donnera naissance à Skype.

La technologie peer-to-peer, celle que l'équipe a rodée avec Kazaa, devient le cœur du système. Les utilisateurs communiquent directement entre eux, sans transiter par des serveurs centraux. Les coûts d'infrastructure fondent, la qualité des communications grimpe. Le nom Skype vient de *Sky peer-to-peer*, d'abord pensé comme « Skyper », puis raccourci quand ils constatent que le nom de domaine n'est pas disponible.

Trouver des financements s'avère ardu. Les controverses juridiques qui entourent Kazaa ont écorné la réputation des fondateurs. Il faut attendre juillet 2002 pour que le fonds Draper Richards accepte d'investir 250 000 dollars contre 5% du capital. Une valorisation modeste qui se révélera, par la suite, un placement spectaculaire.

Le 29 août 2003, Skype ouvre ses portes au public. Une vingtaine de personnes ont développé ce logiciel qui propose des appels gratuits d'ordinateur à ordinateur. La simplicité d'usage et la qualité sonore séduisent d'emblée. Le premier jour, 10 000 personnes téléchargent l'application. Quelques mois plus tard, elles sont un million.

L'entreprise cultive un style bien à elle. Les bureaux n'affichent aucune enseigne, héritage des déboires juridiques de Kazaa qui ont appris aux fondateurs la discrétion. À Tallinn, une piscine gonflable trône dans la salle de réunion. L'équipe code au fil de l'eau : dès qu'une fonctionnalité est conçue, elle est intégrée au logiciel.

En 2004, Skype lance SkypeOut. Le service s'ouvre aux appels vers les téléphones fixes et mobiles, à prix cassés. Les appels entre utilisateurs Skype restent gratuits, mais c'est le début de la monétisation. Les investisseurs se pressent. L'entreprise lève 18 millions de dollars auprès d'Index Ventures et Bessemer Venture Partners, entre autres.

La croissance fulgurante attire les géants. En septembre 2005, eBay met la main sur Skype pour 2,6 milliards de dollars. Les développeurs estoniens sont millionnaires du jour au lendemain, mais la culture d'entreprise commence à se fissurer. Les tensions montent entre les équipes de Tallinn et celles de Londres. L'intégration d'une startup dans une grande entreprise montre ses limites et les relations se dégradent vite. Les fondateurs ont conservé la propriété intellectuelle de la technologie peer-to-peer via leur société Joltid, ce qui crée une situation inextricable. Microsoft rachète finalement Skype en 2011 pour 8,5 milliards de dollars. Une nouvelle page se tourne.

Sous la houlette de Microsoft, Skype évolue. Les révélations d'Edward Snowden en 2013 montrent que le service, autrefois réputé pour sa confidentialité, collabore désormais avec les agences de renseignement. La startup rebelle est devenue un acteur institutionnel du monde numérique.

L'impact de Skype sur les télécommunications reste massif. Le service a rendu les appels internationaux gratuits accessibles à tous, popularisé la vidéoconférence. En 2012, Skype gère 167 milliards de minutes d'appels internationaux, plus que tous les opérateurs télécoms traditionnels réunis.

Les fondateurs ont pris des chemins différents. Jaan Tallinn investit dans des projets liés à la survie de l'humanité ; il considère désormais le temps plus précieux que l'argent. L'équipe estonienne, malgré sa fortune, garde une certaine modestie. Pas de dépenses ostentatoires, des investissements réfléchis.

Skype a transformé un projet technique ambitieux en outil de communication universel. Le service a montré la puissance du peer-to-peer et l'importance d'une interface simple. Une technologie peut évoluer, passant de l'innovation de rupture à l'institutionnalisation, tout en restant utile à des millions de personnes chaque jour.

WordPress

En 2003, un logiciel baptisé b2/cafelog connaît une fin brutale. Son créateur abandonne le projet, laissant derrière lui un code source libre que deux développeurs, Matt Mullenweg et Mike Little, décident de reprendre. Mullenweg n'a alors que 19 ans et ses compétences en programmation viennent d'un apprentissage autodidacte, loin des cursus universitaires traditionnels. De cette initiative naît WordPress, qui deviendra l'un des systèmes de gestion de contenu les plus utilisés au monde.

Les premières versions du logiciel affichent deux priorités : le respect des standards du web et une installation simple. L'équipe met au point le *Famous Five Minute Installation*, un processus d'installation qui ne demande que cinq minutes là où les concurrents exigent souvent plus d'une demi-heure. Cette philosophie d'accessibilité trouve ses racines dans les valeurs du logiciel libre : offrir à chacun les libertés d'utiliser, d'étudier, de modifier et de distribuer le programme. À l'opposé des logiciels propriétaires qui verrouillent leur code, WordPress s'ouvre complètement.

L'évolution technique suit un rythme soutenu. En 2004, la version 1.2 introduit les plugins, ces extensions qui permettent d'ajouter des fonctionnalités sans toucher au cœur du système. L'année suivante, les thèmes et les pages statiques font leur apparition dans la version 1.5, avant que la 2.0 n'apporte la mise en cache et une interface d'administration repensée.

L'année 2007 voit s'enchaîner les améliorations. La version 2.1 intègre l'auto-sauvegarde et la vérification orthographique. Les widgets arrivent avec la 2.2, facilitant la personnalisation des barres latérales. La 2.3 introduit les tags et refond le système de taxonomie, rendant la classification des contenus plus souple.

En 2008, l'agence Happy Cog conçoit une nouvelle interface d'administration pour la version 2.5. Le tableau de bord se dote de widgets modulables et une API de shortcodes voit le jour. Quelques mois plus tard, la version 2.7 améliore l'ergonomie générale, ajoute les mises à jour automatiques et simplifie l'installation des plugins. La gestion des commentaires en fils de discussion enrichit les interactions entre auteurs et lecteurs.

Parallèlement au développement technique, Matt Mullenweg fonde Automattic, l'entreprise commerciale qui héberge WordPress.com, la version en ligne du système. L'organisation adopte un modèle de travail inhabituel : les équipes sont entièrement distribuées, sans bureau central. En 2014, Automattic dépasse le milliard de dollars de valorisation avec plus de 300 employés répartis dans 37 pays.

Le succès de WordPress repose sur plusieurs piliers. Le système de plugins compte aujourd'hui plus de 35 000 extensions, couvrant des besoins aussi variés que le commerce en ligne, la sécurité ou l'optimisation des performances. L'architecture du logiciel séduit les développeurs qui peuvent s'appuyer sur ses composants pour créer des applications web complexes, bien au-delà du simple blog.

La communauté s'organise autour des WordCamps, des conférences locales qui essaiment dans le monde entier. Ces rencontres créent des liens entre utilisateurs, développeurs et entreprises, nourrissant un écosystème dynamique où s'échangent connaissances et bonnes pratiques.

Les grandes organisations finissent par adopter la plateforme. Le New

York Times, le Wall Street Journal ou le Washington Post l'utilisent pour leurs sites web. WordPress propulse environ 23% des sites mondiaux, prouvant sa capacité à répondre aux exigences les plus diverses, de l'artisan local au média international.

Les versions suivantes continuent d'enrichir le système. La 3.0 en 2010 apporte les types de contenus personnalisés et la gestion multi-sites. Le responsive design s'intègre naturellement, s'adaptant aux smartphones et tablettes qui envahissent le marché. L'éditeur visuel s'améliore, le support des emojis arrive, les icônes de site s'intègrent directement dans l'interface.

WordPress évolue vers une plateforme d'applications complète. Les développeurs exploitent ses API pour transformer un simple blog en système de réservation, en boutique en ligne ou en portail d'apprentissage.

Le modèle économique joue sur deux tableaux : le logiciel reste libre et gratuit, mais des services comme l'hébergement, le support technique ou les fonctionnalités premium génèrent des revenus. Cette approche garantit la pérennité du projet sans sacrifier son accessibilité.

Le recrutement chez Automattic reflète cette culture particulière. Les candidats passent par des tests pratiques et des périodes d'essai, les compétences réelles comptant plus que les diplômes. Cette méthode s'inscrit dans les valeurs du logiciel libre, où la contribution effective l'emporte sur les qualifications formelles.

En démocratisant la création de sites web, WordPress a permis à des millions de personnes d'exister et de s'exprimer en ligne. Son modèle de développement collaboratif inspire d'autres projets libres, démontrant qu'une communauté mondiale peut créer et maintenir des outils professionnels de grande envergure.

La gestion du projet suit une ligne pragmatique. Les décisions techniques répondent aux besoins réels des utilisateurs plutôt qu'aux modes passagères. Cette stabilité rassure les entreprises qui choisissent WordPress pour leurs projets stratégiques, sachant que la plateforme évoluera sans rupture brutale.

WordPress est devenu une plateforme web universelle. Cette transformation s'est faite en préservant les principes fondateurs de simplicité et d'ouverture, tout en intégrant les innovations nécessaires à sa croissance. Le logiciel a su grandir sans renier ses origines, restant fidèle à l'esprit qui animait ses deux créateurs en 2003.

Xen

En 1999, au laboratoire d'informatique de l'Université de Cambridge, le Dr. Ian Pratt et son équipe lancent le projet XenoServers. Leur vision : bâtir une infrastructure de serveurs répartis sur Internet où des individus ou des organisations, une fois authentifiés, pourraient acheter des ressources informatiques à la demande. L'idée n'était pas nouvelle dans son principe, mais sa mise en œuvre technique s'annonçait ambitieuse. Le nom XenoServers fait référence au grec *xenos*, l'étranger auquel on offre l'hospitalité sans pour autant lui accorder une confiance aveugle.

Les premiers pas du projet s'orientent vers l'utilisation de machines virtuelles Java pour exécuter le code des clients. Cette solution se heurte à une contrainte rédhibitoire qui oblige à réécrire toutes les applications en Java. L'équipe change alors de cap et vise plus haut : virtualiser directement le matériel x86 pour faire tourner des systèmes d'exploitation complets. À l'époque, virtualiser l'architecture x86 relève du défi technique. Les approches disponibles reposent sur l'émulation d'instructions et la réécriture binaire, des méthodes qui dégradent les performances de manière notable.

Face à ces limitations, les chercheurs inventent la paravirtualisation. Le principe consiste à modifier légèrement les systèmes d'exploitation invités pour qu'ils collaborent avec un hyperviseur, tout en laissant les applications s'exécuter sans retouche. Keir Fraser, doctorant dans l'équipe, se charge de concrétiser cette idée. Il développe le cœur de l'hyperviseur et adapte Linux 2.2 à cette nouvelle architecture. L'hyperviseur reçoit le nom de Xen, qui condense celui du projet initial.

L'un des paris techniques de Xen tient dans sa capacité à héberger des machines virtuelles appartenant à différents clients sur le même serveur physique. L'isolation doit être totale : un client ne peut ni accéder aux données d'un autre, ni perturber son fonctionnement, et chacun reçoit exactement les ressources qu'il paie. Les concepteurs aiment illustrer cette exigence par un exemple parlant : vendre simultanément des services à Coca-Cola et Pepsi en leur garantissant une confidentialité absolue.

Les années qui suivent voient le projet gagner en ampleur. Des laboratoires comme Intel Research et HP Labs apportent leur soutien financier et technique. En 2003, la première version libre de Xen sort sous double licence : GPLv2 pour l'hyperviseur, BSD pour les composants intégrés aux systèmes d'exploitation. Ce choix de licence BSD n'est pas anodin : il simplifie le portage vers des systèmes propriétaires. Microsoft Research s'intéresse au projet, Paul Barham dirigeant un prototype de portage de Windows XP.

La publication de l'article « Xen and the Art of Virtualization » lors de la conférence ACM SOSP marque la consécration académique du projet. Des

fournisseurs d'accès Internet commencent à déployer Xen commercialement, exploitant ses capacités de virtualisation du calcul, du réseau et du stockage.

Christian Limpach contribue en 2004 au portage de NetBSD sur Xen avant de rejoindre l'équipe de Cambridge. Sous l'impulsion de Pratt, la communauté open source autour de Xen s'étoffe. RedHat, SuSE, Sun, IBM, AMD : tous adaptent leurs systèmes d'exploitation ou portent Xen sur leur matériel.

Les chercheurs réalisent qu'étendre l'architecture x86 rendrait la virtualisation plus simple et plus performante, tout en réduisant la quantité de code privilégié nécessaire. Une collaboration avec Intel aboutit au développement du mode HVM de Xen, qui tire parti des extensions VTx d'Intel. Ce mode évite certaines modifications paravirtualisées des systèmes d'exploitation, tout en conservant la paravirtualisation là où elle apporte un gain réel. Le support d'AMD-V arrive ensuite.

L'année 2004 voit aussi les principaux contributeurs créer XenSource face aux demandes croissantes de services de conseil. Financer une entreprise basée sur l'open source au Royaume-Uni et en Europe reste un parcours semé d'embûches, ces modèles économiques y étant peu établis. Ian Pratt recontacte Simon Crosby, ancien collègue universitaire, et part dans la Silicon Valley où XenSource décroche un premier tour de financement de 8 millions de dollars auprès de KPCB et Sevin Rosen.

La première année de XenSource se concentre sur le développement open source. La stratégie initiale ? Fournir des outils de gestion et collaborer avec les distributions Linux. Mais quand ces dernières annoncent leurs propres solutions, cette approche montre ses limites. Fin 2005, Peter Levine et Frank Artale rejoignent l'entreprise via Accel Ventures. Leur vision commerciale réoriente XenSource vers une confrontation directe avec VMware. Un partenariat stratégique avec Microsoft en 2006 renforce la crédibilité de l'entreprise.

En 2007, Citrix rachète XenSource. Déçu par l'entrée de VMware sur le marché de la virtualisation de postes de travail, qui concurrence directement son activité de virtualisation d'applications, Citrix voit dans cette acquisition l'opportunité de disposer d'une plateforme compatible avec Microsoft.

Pendant ce temps, Amazon bâtit son infrastructure cloud sur Xen à partir de fin 2005. Avec des millions de machines virtuelles déployées, Amazon est le plus grand utilisateur de Xen, et probablement celui qui en tire le plus grand bénéfice financier. La capacité d'Amazon à virtualiser Windows repose sur l'adoption de plateformes Intel dotées d'extensions de virtualisation. Depuis, chez AWS, Xen a laissé progressivement sa place à Nitro, la solution maison.

En septembre 2022, après l'acquisition de Citrix par Vista Equity Partners et Evergreen Coast Capital, XenServer devient une unité commerciale indépendante au sein du Cloud Software Group. Cette réorganisation ouvre un nouveau chapitre pour une technologie née dans un laboratoire universitaire britannique.

Le succès de Xen témoigne du rôle de l'open source dans l'innovation technologique et de la capacité des projets universitaires à produire des solutions qui marquent durablement l'industrie.

Apache Cassandra

Les géants du web se heurtent à un mur. Facebook, qui compte des centaines de millions d'utilisateurs, doit gérer une fonctionnalité apparemment simple : la recherche dans la messagerie. Mais derrière cette simplicité se cache un défi technique redoutable. Chaque jour, des milliards d'écritures affluent dans les systèmes, et la croissance ne montre aucun signe de ralentissement. Les bases de données traditionnelles, conçues dans les années 1970, montrent leurs limites face à ces volumes.

Avinash Lakshman et Prashant Malik s'attellent à la création d'une nouvelle solution pour palier au problème. Lakshman n'arrive pas les mains vides, il a participé chez Amazon à la conception de Dynamo, un système de stockage distribué qui fait référence. Leur intuition consiste à croiser deux approches jusqu'alors distinctes. D'un côté, les principes architecturaux de Dynamo qui ont fait leurs preuves chez Amazon. De l'autre, le modèle de données de BigTable, la solution maison de Google. De cette rencontre naît Cassandra, baptisée d'après la prophétesse de la mythologie grecque.

En juillet 2008, Facebook prend une décision qui va changer la donne. Plutôt que de garder jalousement son code, comme le font Google et Amazon qui se contentent de publier des articles décrivant leurs architectures, le réseau social libère l'intégralité du code source de Cassandra sous licence Apache. Le geste marque une rupture dans les pratiques des entreprises technologiques. L'année suivante, le projet intègre l'incubateur de la fondation Apache, avant d'accéder en février 2010 au statut convoité de projet de premier niveau.

L'architecture de Cassandra rompt avec les schémas établis. Là où les bases de données classiques s'organisent selon une hiérarchie maître-esclave, Cassandra adopte un modèle où tous les nœuds jouent le même rôle. Aucun chef d'orchestre, aucun point de défaillance unique. Les données se répartissent automatiquement entre les machines participantes, et le système

grandit horizontalement sans qu'un administrateur ait besoin d'intervenir. Cette décentralisation totale garantit une disponibilité continue, y compris quand certains nœuds tombent en panne.

La redondance des données s'inscrit dans l'ADN du système. Cassandra duplique automatiquement les informations sur plusieurs nœuds. L'administrateur se contente d'indiquer le nombre de copies souhaitées, le reste se fait tout seul. Si une machine lâche, les données restent accessibles ailleurs. Cette approche simple dans son principe s'avère efficace dans la pratique.

Netflix offre un exemple parlant de ce que Cassandra rend possible. En 2011, le service de streaming déploie un cluster de 288 instances dans le cloud. Le système encaisse 1,1 million d'écritures par seconde venues directement des clients. Avec la réplication sur trois zones de disponibilité différentes, ce chiffre grimpe à 3,3 millions d'écritures par seconde. Ces performances auraient été inimaginables quelques années plus tôt avec les technologies conventionnelles.

L'arrivée du langage CQL facilite grandement l'adoption. Sa syntaxe rappelle celle du SQL que connaissent des millions de développeurs dans le monde. Un programmeur habitué aux bases relationnelles est capable de prendre en main Cassandra sans tout réapprendre. Certes, des différences subsistent : pas de jointures par exemple, ce qui reflète le modèle de données dénormalisé du système. Mais le seuil d'entrée baisse considérablement.

La communauté grossit à vue d'œil. En 2012, plus de 1 000 déploiements en production tournent déjà, de eBay à Disney en passant par Netflix. Les outils se multiplient, les intégrations avec Hadoop, Spark ou Solr enrichissent l'écosystème. Le monde du big data trouve en Cassandra un pilier fiable.

DataStax voit le jour en 2010 et apporte une dimension commerciale au projet. L'entreprise embauche les principaux contributeurs et propose une version entreprise avec des fonctionnalités supplémentaires, du support technique et des outils d'administration comme OpsCenter. Le modèle open source trouve là son pendant économique viable.

Les chiffres parlent d'eux-mêmes. Une étude comparative présentée à la conférence Very Large Database en 2012 met Cassandra face à HBase. Les temps de latence en lecture s'avèrent jusqu'à 100 fois plus rapides, le débit huit fois supérieur. Ces résultats ne sortent pas de nulle part : ils découlent directement des choix architecturaux initiaux.

Les versions se succèdent et apportent leur lot d'améliorations. La 1.0 en 2011 optimise les performances et compresse les données. La 2.0 en 2013 introduit des transactions légères via le protocole Paxos. La 3.0 en 2015 ajoute les vues matérialisées. La 4.0 en 2021 déploie le streaming sans copie et supporte Java 11. Chaque itération pousse un peu plus loin les capacités

du système.

Instagram, racheté par Meta, s'appuie sur Cassandra pour gérer les données de plus d'un milliard d'utilisateurs actifs chaque mois. Les cas d'usage se diversifient : séries chronologiques, analyse temps réel, gestion de contenus multimédias, commerce électronique. Partout où les volumes explosent et où l'indisponibilité coûte cher, Cassandra trouve sa place.

L'impact dépasse la seule dimension technique. Cassandra démontre qu'une base de données peut conjuguer haute disponibilité, montée en charge linéaire et performances élevées. Ces trois piliers répondent exactement aux besoins des applications web contemporaines, où chaque minute d'arrêt se compte en millions et où les données ne cessent de croître.

Cassandra trouve un terrain favorable dans le déploiement hybride et multi-cloud. Le système fonctionne de façon transparente sur plusieurs centres de données ou fournisseurs cloud différents. Tout en gardant leurs données cohérentes, les entreprises échappent ainsi à la dépendance vis-à-vis d'un unique prestataire.

D'autres projets s'inspirent de Cassandra. ScyllaDB intègre ses concepts en C++ pour gagner encore en performances. L'architecture décentralisée et le modèle de cohérence ajustable servent désormais de référence dans le domaine du stockage distribué.

Vingt ans après sa naissance, Apache Cassandra demeure une pièce maîtresse de l'écosystème NoSQL. Son histoire raconte la maturation des architectures distribuées et leur capacité à s'adapter aux exigences toujours plus élevées des applications modernes. Ce qui commença comme une solution interne chez Facebook est devenu un standard de facto pour quiconque doit gérer des données massives avec des garanties de disponibilité strictes.

Google Gmail

En 2001, le marché de la messagerie électronique gratuite se résumait à trois acteurs installés : Microsoft Hotmail, Yahoo ! Mail et AOL. Chacun proposait à peu près la même chose, quelques mégaoctets d'espace où stocker ses messages. Les utilisateurs passaient leur temps à faire le tri, supprimaient régulièrement leurs anciens courriers pour ne pas dépasser la limite imposée. C'était la norme, personne ne s'en plaignait vraiment, probablement à cause de la gratuité.

Chez Google, une employée finit par exprimer sa frustration. Elle en avait assez de cette corvée permanente, ces 4 mégaoctets qui l'obligeaient

sans cesse à archiver, classer, effacer. Paul Buchheit, ingénieur dans l'entreprise, entendit cette remarque. À cette époque, Google ne faisait que de la recherche web. Développer un service de messagerie représentait un virage complet pour l'entreprise, une sortie de son territoire habituel.

Buchheit n'en était pas à sa première tentative dans le domaine. Avant de rejoindre Google, il avait travaillé sur un prototype de messagerie web qui n'avait jamais vu le jour. Fort de cet échec, il décida cette fois d'aller vite. En une seule journée, il créa une première version en recyclant du code existant, celui de Google Groups qui servait à fouiller dans les archives Usenet. Ce premier jet ne faisait qu'une chose : chercher dans ses propres e-mails. Rien de plus, mais c'était un début.

La vraie rupture technique résidait dans l'utilisation de JavaScript et de l'objet XMLHttpRequest. JavaScript avait mauvaise réputation chez Google, associé aux publicités envahissantes qui polluaient le web. Pourtant, cette technologie s'avérait indispensable pour bâtir une interface réactive, sans ces rechargements de page qui cassaient l'expérience utilisateur. L'association de JavaScript et de XMLHttpRequest, qui prendrait plus tard le nom d'AJAX, permit de créer quelque chose qui ressemblait vraiment à un logiciel installé sur l'ordinateur.

Le développement se fit pas à pas, chaque nouvelle fonction testée d'abord par les employés de Google. Les retours arrivaient directement, les améliorations suivaient immédiatement. L'une des idées marquantes fut d'organiser les messages en conversations plutôt qu'en courriers isolés. Cette présentation reprenait la logique des groupes de discussion Usenet, où suivre un fil d'échanges se faisait naturellement.

La question de l'argent se posa. Marissa Mayer, qui supervisait le développement des produits, penchait pour le modèle classique : un espace gratuit limité, une formule payante pour ceux qui voulaient plus. Buchheit proposa autre chose : afficher des publicités liées au contenu des messages. L'idée fit débat, on parla de respect de la vie privée, de confidentialité. Mais cette approche donna naissance à AdSense, qui devint l'une des principales sources de revenus de Google.

Le 1er avril 2004, Google annonça Gmail avec une promesse incroyable : 1 gigaoctet de stockage gratuit. Deux cent cinquante fois plus que la concurrence. La date de l'annonce et l'ampleur de l'offre semèrent le doute. Google adorait les canulars du 1er avril, et proposer autant d'espace paraissait techniquement impossible, économiquement absurde. Beaucoup crurent à une blague.

Le service démarra en envoyant des invitations. Les premiers utilisateurs reçurent quelques invitations à distribuer, créant un système de diffusion virale. Cette rareté artificielle permit une montée en charge progressive des

serveurs tout en construisant une image de service élitiste, désirable. Les invitations Gmail se négociaient même sur eBay.

Gmail transforma Google. L'entreprise prouva qu'elle savait faire autre chose que de la recherche web, qu'elle pouvait créer des services web sophistiqués. Les techniques mises au point pour Gmail, notamment cette fameuse utilisation d'AJAX, devinrent des standards du développement web. Google publia d'ailleurs en 2005 le Google Web Toolkit, un framework qui permettait à d'autres développeurs d'utiliser ces technologies.

La phase bêta dura cinq ans. Gmail en sortit en 2009, après avoir intégré progressivement de nouvelles fonctions : le chat, la vérification orthographique, la détection des pourriels, les connexions avec Calendar et Drive. L'interface évolua mais garda ses principes d'origine, cette simplicité qui faisait sa force.

L'impact dépassa largement le cadre de la messagerie. Gmail redéfinit ce qu'on pouvait attendre d'une application web. Microsoft et Yahoo ! durent revoir leurs copies, augmenter drastiquement leur espace de stockage. Plus largement, le service montra qu'une application web pouvait rivaliser avec un logiciel installé sur le disque dur.

Gmail reflète la façon dont Google aborde l'innovation. Une approche par essais et erreurs, des itérations rapides, une attention portée aux besoins réels. Cette méthode mêle réflexion technique et observation du terrain. Le service continue sa mue en intégrant de l'intelligence artificielle pour trier automatiquement les messages, suggérer des réponses, détecter les tentatives d'hameçonnage.

Scala

Martin Odersky lance le développement de Scala en 2001 au sein du laboratoire des méthodes de programmation de l'École Polytechnique Fédérale de Lausanne. Sa démarche répond à une question précise : comment mieux supporter les logiciels à base de composants dans les langages de programmation ?

Trois ans plus tard, en janvier 2004, la première version publique de Scala voit le jour sur la plateforme Java Virtual Machine. Une mouture pour .NET suit en juin. Le nom retenu, « Scala », contracte *scalable language* : un langage extensible qui s'adapte aux besoins de ses utilisateurs. Cette idée d'adaptabilité traverse toute la conception : des petits scripts jusqu'aux grands systèmes, l'outil doit pouvoir suivre.

L'originalité de Scala tient dans sa manière de fusionner la programmation orientée objet et la programmation fonctionnelle sous un typage statique. Là où d'autres langages maintiennent ces paradigmes séparés, Scala les unit. Son système de types intègre des concepts avancés comme les types abstraits et les types dépendants de chemins, hérités du calcul υObj. La composition modulaire repose sur des *mixins* et des traits. Les vues offrent quant à elles une adaptation modulaire des composants.

La compatibilité avec Java structure fortement la conception. Scala reprend une bonne part de la syntaxe et du système de types de Java. Un développeur Java retrouve ses repères. Les bibliothèques Java s'utilisent directement dans Scala, et inversement. Les classes Scala héritent des classes Java et implémentent leurs interfaces. Cette interopérabilité autorise l'insertion du code Scala dans des projets Java existants sans tout réécrire.

Mars 2006 marque la sortie d'une seconde version majeure. Le système de types gagne en robustesse, les mécanismes de composition de classes se précisent. La syntaxe reste volontairement classique, mais cache une sophistication technique qui autorise l'expression de concepts complexes en peu de lignes.

Les influences de Scala sont multiples. Du côté objet, Simula et Smalltalk inspirent le modèle objet uniforme. L'imbrication universelle — cette capacité à imbriquer presque toutes les constructions les unes dans les autres — vient d'Algol, de Simula, puis de Beta. L'approche fonctionnelle rappelle la famille ML avec SML, OCaml ou F#. Les paramètres implicites trouvent leur racine dans les classes de types de Haskell, adaptées ici au monde objet classique. La bibliothèque de concurrence basée sur les acteurs doit beaucoup à Erlang.

Plutôt que d'imposer un ensemble figé de constructions, Scala mise sur l'extensibilité. Les programmeurs créent leurs propres abstractions. Cette philosophie évoque le bazar plutôt que la cathédrale, selon la métaphore d'Eric Raymond : le langage grandit par l'ajout de constructions inventées par ses utilisateurs, pas par des modifications de son noyau.

Les apports techniques de Scala se remarquent dans plusieurs domaines. Le traitement uniforme des types génériques et des types abstraits sort des sentiers battus. La composition de classes via les traits propose une voie originale. Le mécanisme d'extraction autorise la correspondance de motifs indépendamment de la représentation. Ces innovations ont fait l'objet de présentations dans diverses conférences spécialisées.

L'industrie a saisi l'intérêt de Scala, surtout pour les applications distribuées et le traitement de données massives. Apache Spark, écrit en Scala, montre les capacités du langage sur des systèmes complexes de traitement de données. La syntaxe concise réduit la taille du code comparé à Java —

jusqu'à 50% dans des cas courants.

Le Fonds National Suisse, Microsoft Research, le Centre de Compétences en Recherche MICS, le projet européen PalCom et la Fondation Hasler ont soutenu le développement. La communauté des développeurs participe activement avec des retours d'expérience et des contributions au code.

L'évolution de Scala traduit une approche pragmatique. Pas question de construire un système parfait et rigide. L'adaptabilité et l'extensibilité priment. Cette stratégie donne aux développeurs les moyens d'enrichir le langage selon leurs besoins, créant un écosystème vivant de bibliothèques et de frameworks.

En 2025, Scala poursuit sa route en gardant ses principes fondateurs : l'unification de la programmation orientée objet et fonctionnelle, l'extensibilité, l'interopérabilité avec les plateformes établies. Le langage nourrit les réflexions sur la conception des langages de programmation et l'architecture des systèmes logiciels.

NGINX

Igor Sysoev avait un problème. En 2002, cet ingénieur russe travaillait pour Rambler, un portail internet qui voyait affluer des centaines de milliers de connexions simultanées. Les serveurs peinaient. Apache, la solution dominante de l'époque, montrait ses failles : chaque connexion réclamait ses propres ressources système, créant une lourdeur qui devenait insoutenable quand le trafic grimpait. Sysoev cherchait autre chose, une approche différente pour gérer cette avalanche de requêtes.

Il se lance alors dans l'écriture d'un nouveau serveur web, qu'il baptise NGINX (prononcé engine-x). Son architecture rompt avec les conventions : au lieu d'allouer un processus ou un thread par connexion, le logiciel adopte un modèle asynchrone et événementiel. Un processus maître supervise l'ensemble, quelques processus enfants se répartissent le travail, et cette poignée de composants suffit à traiter des milliers de requêtes en parallèle, pour une économie de ressources spectaculaire.

En octobre 2004, Sysoev publie la version 0.1.0. Il choisit cette date en référence au lancement de Spoutnik, clin d'œil au premier satellite artificiel. Le code est distribué sous licence BSD, libre d'accès. Les premiers à tenter l'aventure sont des sites estoniens et russes comme rate.ee ou zvuki.ru, qui utilisent NGINX pour servir leurs fichiers MP3. La nouvelle recrue fait ses preuves.

Le logiciel gagne du terrain, d'abord dans la sphère russophone, puis au-delà. Les administrateurs système découvrent ses talents : servir du contenu statique avec une vélocité remarquable, jouer le rôle de proxy inverse, répartir la charge entre plusieurs serveurs, gérer le cache ou encore servir de passerelle FastCGI. NGINX se montre polyvalent, fiable, rapide. Sa réputation traverse les frontières.

Sept ans après la première version publique, en 2011, Sysoev et Maxim Konovalov fondent NGINX, Inc. L'idée ? Proposer du support commercial et des fonctionnalités avancées tout en préservant la version open source. Les investisseurs réagissent vite : BV Capital et Runa Capital injectent 3 millions de dollars dès octobre 2011. D'autres levées suivent : 10 millions en 2013, 20 millions en 2014, 43 millions en 2018. L'entreprise développe NGINX Plus, une offre payante qui complète la version communautaire avec des capacités étendues de monitoring et d'analyse.

F5 Networks rachète NGINX en mars 2019 pour 670 millions de dollars, validation éclatante de l'importance du logiciel dans l'infrastructure web contemporaine. Mais l'histoire prend une tournure inattendue en décembre de la même année : une perquisition frappe les bureaux moscovites de NGINX. Rambler, l'ancien employeur de Sysoev, revendique la propriété intellectuelle du code. L'affaire se dénoue grâce à l'intervention de Sberbank, actionnaire de Rambler, mais l'épisode laisse des traces.

Sur le terrain technique, NGINX continue d'évoluer. Il intègre HTTP/2, affine ses mécanismes de compression et de cache, manipule les en-têtes avec souplesse. La mémoire partagée sert à stocker le cache, gérer la persistance des sessions, contrôler le débit. Cette efficacité séduit : en 2022, plus de 33% des sites web mondiaux tournent sur NGINX, dépassant Apache dans différentes « catégories. Des plateformes comme WordPress.com lui font confiance. Le logiciel s'adapte aux architectures modernes, microservices et cloud compris.

Janvier 2022 : Sysoev annonce son départ de NGINX et F5. Il laisse derrière lui un projet qui a redessiné l'infrastructure du web. Son dernier né, NGINX Unit, lancé en 2017, vise les environnements multilingues et les architectures en microservices. L'héritage demeure.

De Rambler aux serveurs du monde entier, le logiciel a grandi pendant que le web changeait de visage. Son succès tient à des choix architecturaux astucieux, une approche pragmatique des questions de performance, et un équilibre réussi entre code ouvert et modèle économique viable.

OAuth

En 2006, Blaine Cook travaillait sur l'API de Twitter et se heurtait à un problème qui devenait récurrent chez les développeurs web. Il cherchait un moyen de connecter Twitter à Flickr, le service de partage de photos. La seule solution disponible était de demander aux utilisateurs de confier leurs identifiants Flickr à Twitter. Une aberration en matière de sécurité, mais c'était la norme à l'époque.

Cette impasse technique a poussé Cook et d'autres développeurs de grandes plateformes web à se réunir. Google avait AuthSub, Yahoo utilisait BBAuth, AOL proposait OpenAuth, et Flickr disposait de sa propre API. Chacun bricolait sa solution dans son coin. Il fallait sortir de ce morcellement. En octobre 2007, leur collaboration aboutit à OAuth 1.0, un protocole ouvert destiné à résoudre ce casse-tête : comment autoriser une application tierce à accéder aux données d'un utilisateur sans jamais manipuler son mot de passe ?

Le mécanisme repose sur un principe simple mais efficace. Imaginons un service d'impression en ligne qui veut accéder aux photos stockées sur un autre site. Au lieu de réclamer les identifiants du compte photos, il obtient un jeton d'accès spécifique, limité aux seules opérations nécessaires. L'utilisateur garde le contrôle : il autorise explicitement l'accès depuis une interface sécurisée du site hébergeant ses photos. Le service d'impression ne connaît jamais le mot de passe.

Les premières failles de sécurité se sont manifestées, et en juin 2009, OAuth 1.0a corrigeait des vulnérabilités permettant des usurpations d'identité. Cette révision a clarifié des zones d'ombre du protocole initial et renforcé les protections contre les attaques. L'année suivante, en avril 2010, l'IETF publiait OAuth 1.0 comme RFC 5849, lui conférant un statut de spécification technique officielle.

Le protocole définit trois acteurs : le propriétaire de la ressource (l'utilisateur), le client (l'application qui demande l'accès) et le serveur d'autorisation. L'application obtient d'abord des identifiants temporaires. Elle redirige ensuite l'utilisateur vers une page d'autorisation où celui-ci examine les permissions demandées. Après validation, l'application reçoit un jeton qu'elle utilisera pour ses requêtes futures. À aucun moment elle ne touche aux identifiants de l'utilisateur.

La sécurité s'appuie sur des garde-fous. Chaque requête doit être signée avec une clé secrète. Le protocole intègre des jetons uniques (nonces) et des timestamps pour bloquer toute tentative de réutilisation malveillante. Les concepteurs ont anticipé les attaques par détournement de session, par rejeu, ou par usurpation.

Facebook, Google, Microsoft et Twitter ont adopté OAuth. Cette adhésion massive des géants du web a créé un écosystème d'applications interconnectées. Des bibliothèques sont apparues dans tous les langages de programmation courants, rendant l'intégration plus accessible.

Mais OAuth 1.0 montrait ses limites. La complexité de l'implémentation, notamment pour la signature cryptographique des requêtes, décourageait les développeurs. Les applications mobiles natives, de plus en plus nombreuses, posaient des problèmes non prévus au départ. Ces contraintes ont motivé une refonte complète du protocole avec OAuth 2.0.

Le protocole OAuth a établi le standard de l'autorisation déléguée sur le web. Il a introduit des concepts désormais centraux : le consentement explicite de l'utilisateur, la limitation stricte des privilèges accordés, la séparation nette entre authentification et autorisation.

OAuth a aussi joué un rôle éducatif. Développeurs et utilisateurs ont pris conscience des enjeux autour de la protection des données personnelles. Le protocole a démontré qu'il était possible de créer des services interconnectés sans sacrifier la sécurité. Cette prise de conscience a accompagné l'évolution des réglementations sur la vie privée.

Par ailleurs, les architectures modernes portent la marque d'OAuth. OpenID Connect s'appuie sur ses fondations pour ajouter une couche d'authentification standardisée. Les API REST et les microservices reprennent ses principes architecturaux. Les objets connectés et les systèmes d'identités décentralisées explorent de nouvelles pistes en partant des bases posées par OAuth.

La communauté continue d'adapter le protocole aux nouveaux usages. Les retours d'expérience des déploiements à grande échelle alimentent les réflexions sur ses évolutions futures. OAuth reste un pilier de la sécurité web, près de vingt ans après sa création.

Git

En 2005, Linus Torvalds se retrouve dans une situation embarrassante. BitKeeper, le système de gestion de versions que la communauté Linux utilise depuis des années, n'est plus accessible gratuitement aux développeurs du noyau. L'histoire commence avec Andrew Tridgell, un développeur respecté, qui décide d'étudier le protocole de BitKeeper par rétro-ingénierie. Larry McVoy, le créateur de BitKeeper, considère cette démarche comme

une violation des conditions d'utilisation et révoque la licence gratuite accordée aux développeurs Linux.

Torvalds tente une médiation pendant quelques semaines, sans résultat. Face à ce blocage, il doit trouver une alternative. Il examine les systèmes existants mais aucun ne le satisfait. Lui qui considérait les gestionnaires de versions comme l'un des domaines les moins intéressants de l'informatique, juste après les bases de données, se lance dans la création de son propre outil. Son expérience avec BitKeeper lui a au moins appris une chose : les avantages d'un système distribué où chaque développeur possède une copie locale du dépôt.

Torvalds disparaît alors pendant une semaine. Quand il refait surface, il présente la première version de Git. Le développement initial progresse à une vitesse remarquable. En dix jours, Git est capable de gérer son propre code source. Cette rapidité ne vient pas d'une quantité massive de code écrit, mais d'une réflexion approfondie sur l'organisation des données. Torvalds a conçu une architecture qui privilégie la simplicité conceptuelle et la performance.

Le système repose sur plusieurs choix techniques. Chaque développeur dispose d'une copie complète du dépôt, ce qui élimine la dépendance à un serveur central et résout les problèmes d'accès en écriture. Git utilise un hachage cryptographique SHA-1 pour identifier les objets dans le dépôt. Cette approche garantit l'intégrité du code et rend impossible toute modification de l'historique sans que cela soit détecté.

La gestion des branches et des fusions constitue l'une des grandes innovations de Git. Ces opérations, réputées complexes dans les systèmes précédents, sont simples et rapides. Torvalds réalise parfois des dizaines de fusions par jour durant les périodes intenses de développement du noyau Linux. Cette fluidité transforme la manière de travailler des développeurs.

Quelques mois après sa création, Torvalds confie la maintenance de Git à Junio Hamano. Il reconnaît en lui un développeur doté d'un excellent jugement technique. Sous sa direction, Git évolue pour devenir plus accessible tout en conservant ses qualités de rapidité, de fiabilité et de flexibilité.

L'adoption de Git se fait progressivement. Le projet Android de Google le choisit comme système de gestion de versions en 2009, ce qui apporte une validation importante de la part d'une grande entreprise. La création de GitHub en 2008 par Tom Preston-Werner, Chris Wanstrath et P.J. Hyett accélère la démocratisation de l'outil. La plateforme offre une interface web conviviale et des fonctionnalités de réseau social pour les développeurs.

Microsoft, longtemps réticent aux logiciels libres, change progressivement de position. En 2012, l'entreprise commence à contribuer à libgit2, une bibliothèque de développement Git. L'année suivante, elle intègre le

support de Git dans Visual Studio et propose l'hébergement via Azure DevOps. En 2017, Microsoft migre le développement de Windows vers Git, créant le plus grand dépôt Git au monde avec 300 Go de code source.

L'acquisition de GitHub par Microsoft en 2018 pour 7,5 milliards de dollars marque un moment symbolique. Cette transaction illustre la transformation de Microsoft et confirme la position dominante de Git dans l'industrie du logiciel. Les statistiques de Stack Overflow en 2018 révèlent que 88,4% des développeurs utilisent Git, loin devant Subversion (16,6%) ou Mercurial (3,7%).

La conception de Git reflète les besoins spécifiques du développement du noyau Linux. La capacité à gérer un grand nombre de contributions simultanées, la rapidité des opérations, la robustesse face aux erreurs : ces caractéristiques, combinées à sa nature distribuée, en font un outil adapté aux projets open source de grande envergure. Les développeurs peuvent travailler de manière autonome sur leurs copies locales, puis synchroniser leurs modifications avec le reste de l'équipe quand ils le souhaitent.

En 2024, Git demeure le standard de fait pour la gestion de versions dans l'industrie du logiciel. Il a transformé la manière dont les développeurs collaborent et partagent leur code. Les principes de son architecture distribuée ont inspiré de nombreux autres outils et pratiques dans le domaine du développement logiciel.

Redis

En 2009, Salvatore Sanfilippo, développeur italien connu sous le pseudonyme « antirez », bute sur un problème irritant. L'analyseur de logs web qu'il développe pour sa startup manque cruellement de réactivité avec MySQL. Plutôt que de bricoler une énième optimisation, il prend le parti de créer sa propre base de données. Ce qui commence comme un prototype en Tcl se transforme en un projet C baptisé Redis, contraction de REmote DIctionary Server.

Quelques semaines après avoir écrit les premières lignes de code, Sanfilippo fait un choix qui scellera l'avenir de son projet : il le publie en open source. La décision s'avère judicieuse. Les développeurs adoptent Redis massivement, attirés par sa vélocité d'exécution et sa simplicité apparente. Le projet rencontre un succès qui dépasse les espérances de son créateur.

Redis rompt avec les conventions des bases de données traditionnelles. Là où la plupart stockent leurs données sur disque, Redis les conserve inté-

gralement en mémoire vive. Cette architecture procure une rapidité d'accès sans commune mesure, mais elle nécessite des mécanismes de persistance pour parer aux arrêts intempestifs du système. Mais Redis ne se contente pas d'être rapide : il manipule nativement des structures de données variées comme les listes, les ensembles, les tables de hachage ou les ensembles triés. Une richesse qui le distingue des simples magasins clé-valeur.

VMware recrute Sanfilippo en 2010 pour qu'il se consacre à plein temps à Redis, tout en préservant la licence BSD du projet. Pieter Noordhuis, contributeur de premier plan, rejoint l'équipe peu après. Cette professionnalisation accélère le développement et Redis s'impose progressivement dans les architectures modernes.

Les géants de la tech ne s'y trompent pas. GitHub et Instagram figurent parmi les premiers adoptants notables. Twitter, Uber, Stripe, Slack et Alibaba suivront le mouvement. Cette adoption massive tient à la polyvalence de l'outil : Redis fonctionne aussi bien comme base de données que comme système de cache ou courtier de messages. Trois rôles pour le prix d'un seul logiciel.

Sanfilippo a pensé Redis comme un jeu de construction. Plutôt qu'un produit monolithique aux fonctionnalités figées, il a conçu des briques élémentaires que les développeurs peuvent assembler selon leurs besoins. Le scripting Lua, le système de publication-souscription ou les flux de données se sont ajoutés progressivement, suivant cette philosophie modulaire. Redis ressemble à un Lego pour programmeurs, où chacun bâtit sa propre solution.

La création de Redis Labs en 2015 (devenue simplement Redis) professionnalise l'écosystème. Cette structure offre des services commerciaux autour du projet tout en maintenant son caractère libre. Après onze ans à piloter Redis, Sanfilippo s'éloigne en 2020 pour se consacrer à l'écriture et à sa famille. Un retrait du quotidien qui n'est pas une rupture.

L'architecture technique de Redis fait des choix audacieux. Son modèle mono-thread évite les casse-têtes liés aux verrous et simplifie la gestion de la concurrence. Pour persister les données, deux mécanismes cohabitent : le *snapshot* périodique et le journal des opérations (AOF). La réplication maître-esclave distribue la charge et garantit la disponibilité. Des solutions pragmatiques qui ont fait leurs preuves.

Redis introduit une manière singulière de penser les données. Au lieu de se limiter aux paires clé-valeur classiques, il manipule directement des structures complexes via des commandes atomiques. Cette approche réduit la complexité du code applicatif et limite les échanges réseau entre client et serveur. Un gain de temps appréciable quand on travaille à grande échelle.

L'histoire pourrait s'arrêter là, mais Sanfilippo revient en 2024 comme

évangéliste de Redis. Son nouveau combat : adapter l'outil à l'ère de l'intelligence artificielle. Outre la nouvelle licence qui n'est plus open source au sens de l'OSI, il travaille sur les *vector sets*, une extension des ensembles triés taillée pour les *embeddings* et les modèles de langage. Redis évolue avec son temps.

YouTube

Le 14 février 2005, trois anciens employés de PayPal enregistrent la marque, le logo et le nom de domaine d'un projet qui va changer la manière dont nous consommons la vidéo sur Internet. Chad Hurley, Steve Chen et Jawed Karim ont l'intuition qu'il manque quelque chose sur le web : un moyen simple de partager des vidéos. L'idée germe lors d'un dîner à San Francisco, nourrie par deux événements de 2004 qui ont mis en lumière ce vide technologique. D'abord l'incident vestimentaire de Janet Jackson au Super Bowl, dont les vidéos sont restées introuvables en ligne. Puis le tsunami en Asie, dont les images peinent à circuler sur la toile.

En mai 2005, le site ouvre en version bêta et attire immédiatement 30 000 visiteurs par jour. La première vidéo mise en ligne ne laisse présager en rien le phénomène à venir. Dans cette séquence de 19 secondes intitulée « Me at the Zoo », Jawed Karim filme les éléphants du zoo de San Diego et commente la taille de leurs trompes. Rien de spectaculaire, et pourtant ce fragment banal totalise aujourd'hui plus de 375 millions de vues. Quatre mois plus tard, une publicité Nike avec Ronaldinho franchit le cap du million de vues, révélant le potentiel commercial de la plateforme.

Le lancement officiel a lieu le 15 décembre 2005, rendu possible par un investissement de 3,5 millions de dollars de Sequoia Capital. Cet apport financier est investi dans l'amélioration des serveurs et l'augmentation de la bande passante, car la croissance est vertigineuse. En janvier 2006, le site enregistre 25 millions de vues quotidiennes. Mars voit le nombre de vidéos disponibles dépasser les 25 millions, avec 20 000 nouvelles publications chaque jour. L'été suivant, le cap des 100 millions de vues quotidiennes est franchi.

Cette expansion fulgurante génère des coûts considérables en infrastructure informatique et en connexions Internet. La question des droits d'auteur surgit lorsque de nombreuses entreprises médiatiques découvrent leurs contenus protégés dans les vidéos téléchargées par les utilisateurs. En octobre 2006, Google tranche et acquiert YouTube pour 1,65 milliard de dollars en actions. Plutôt que de fusionner la plateforme avec Google Video,

son propre service qui peine à décoller, le géant de Mountain View préfère maintenir YouTube en fonctionnement autonome.

Google négocie alors des accords avec des sociétés de divertissement pour réduire les risques juridiques liés aux droits d'auteur. Ces arrangements autorisent l'utilisation de contenus protégés et permettent aux utilisateurs d'intégrer certaines musiques sous copyright dans leurs vidéos. En 2008, un partenariat avec MGM va plus loin en autorisant la diffusion gratuite de films et séries, financée par la publicité.

Grâce au Programme Partenaire en mai 2007, les créateurs de contenus peuvent désormais monétiser leurs vidéos virales, transformant leurs passions en activités lucratives. Certains utilisateurs génèrent des revenus à six chiffres. L'adoption institutionnelle suit en 2009, quand le Congrès américain et le Vatican ouvrent leurs chaînes officielles.

YouTube Live débarque en avril 2011, diffusant concerts, événements sportifs et mariages royaux en direct. En décembre 2012, le clip « Gangnam Style » est la première vidéo à dépasser le milliard de vues. L'événement force la plateforme à modifier son compteur pour gérer des nombres supérieurs aux limites d'un entier 32 bits.

Le modèle économique se diversifie avec des services payants. Music Key apparaît en 2014, remplacé par YouTube Red en 2015, et YouTube Premium en 2018. Ces offres donnent accès à un visionnage sans publicité et des fonctionnalités supplémentaires. En 2016, YouTube Go cible les marchés émergents avec une application optimisée pour les connexions limitées.

L'intelligence artificielle s'invite dans la plateforme. En 2019, le service Video Reach utilise l'apprentissage automatique pour optimiser les campagnes publicitaires. L'année suivante, YouTube abandonne son système de catégories manuel au profit d'algorithmes de classification plus sophistiqués.

En 2023, la plateforme revendique environ 2,5 milliards d'utilisateurs mensuels actifs, se hissant au rang de second réseau social mondial derrière Facebook. Le format YouTube Shorts, réponse directe aux vidéos courtes popularisées par TikTok, génère plus de 50 milliards de vues quotidiennes. YouTube Premium atteint 125 millions d'abonnés en mars 2025.

En vingt ans, YouTube a bouleversé la consommation de contenus vidéo en ligne. La plateforme a instauré un nouveau modèle de diffusion démocratisé, où chacun peut devenir créateur. Son influence sur la culture numérique, l'éducation, le divertissement et la publicité continue de s'étendre, faisant de ce service un pilier de l'Internet moderne.

Amazon Web Services

En 2003, Amazon était un site de commerce en ligne qui grandissait vite. Les serveurs peinaient à suivre la cadence des commandes, et chaque pic d'activité révélait les limites d'une infrastructure conçue à la va-vite en 1994. Jeff Bezos et ses équipes cherchaient comment tenir la charge sans multiplier indéfiniment les investissements matériels.

L'entreprise avait lancé un service censé permettre à des détaillants comme Marks & Spencer de bâtir leurs propres boutiques sur la plateforme Amazon. Sauf que personne n'avait anticipé la pagaille architecturale qui attendait les développeurs. Le code s'était empilé au fil des ans sans véritable plan d'ensemble. Impossible d'extraire proprement les briques logicielles pour les proposer à des tiers. Il fallait tout reprendre depuis la base.

Amazon se mit donc à découper son infrastructure en services distincts, chacun accessible par des interfaces de programmation bien documentées. Cette réorganisation permit de rendre le service fonctionnel, mais elle eut un effet secondaire inattendu : les équipes internes commencèrent à travailler différemment. Chaque service devenait autonome, réutilisable, standardisé. Une discipline nouvelle s'installait dans le développement logiciel.

Pendant ce temps, Andy Jassy, directeur de cabinet de Bezos, constatait un paradoxe troublant. Amazon recrutait des ingénieurs à tour de bras, pourtant la vitesse de développement stagnait. En creusant, il découvrit que les équipes perdaient trois mois à monter leur environnement technique avant de commencer à coder. Base de données, serveurs de calcul, stockage : chacun reconstruisait les mêmes fondations dans son coin. Un gâchis monumental de temps et d'énergie.

L'été 2003, Bezos réunit ses dirigeants dans sa maison pour une retraite stratégique. Une session anodine sur les compétences d'Amazon s'éternisa bien au-delà des trente minutes prévues. Au-delà du commerce électronique et de la logistique, une évidence émergea : Amazon savait gérer des infrastructures informatiques complexes. La contrainte des marges réduites avait forcé l'entreprise à optimiser ses centres de données jusqu'à l'excellence. Cette expertise valait de l'or.

L'idée germa alors de transformer cette infrastructure en un système d'exploitation pour Internet. Tous les composants existaient déjà chez Amazon, mais personne d'autre n'y avait accès. Les développeurs du monde entier réinventaient les mêmes briques, perdaient les mêmes trois mois, butaient sur les mêmes problèmes. Amazon pouvait leur vendre ces éléments tous faits.

La vision mit du temps à se matérialiser. Amazon Simple Queue Service sortit en 2006, suivi de près par Simple Storage Service et Elastic Com-

pute Cloud. Ces trois services formèrent le socle d'AWS. Les développeurs pouvaient désormais louer de la puissance de calcul, du stockage, des files d'attente, sans acheter un seul serveur. Le tout facturé à l'usage, comme l'eau ou l'électricité.

Le modèle économique bousculait les habitudes. Fini les investissements massifs dans des salles machines qui prenaient la poussière la moitié du temps. Les ressources s'ajustaient à la demande, montaient en charge lors des pics, redescendaient pendant les creux. La complexité technique disparaissait derrière des interfaces simples. Les entreprises pouvaient enfin se concentrer sur leur métier plutôt que sur la gestion de serveurs.

AWS débarqua sur un marché vierge. Étonnamment, les concurrents mirent des années à réagir. Microsoft, IBM, Google regardèrent passer le train avant de se lancer à leur tour. Trop tard : AWS avait posé les standards du *cloud computing* et raflé plus de 30% du marché en 2016.

Le chiffre d'affaires grimpa jusqu'à 10 milliards de dollars annuels. Des startups aux géants comme Netflix ou Dropbox, tout le monde adoptait les services Amazon. L'obsession de l'efficacité opérationnelle expliquait en partie ce succès. Amazon construisit des centres de données gigantesques dans des zones stratégiques, là où l'électricité et le refroidissement coûtaient peu. Les économies d'échelle lui permirent de baisser régulièrement les prix tout en restant rentable.

AWS marqua le passage d'une informatique de possession à une informatique de consommation. Comme les services publics au moment de la révolution industrielle, le cloud devint une commodité disponible à la demande.

Hadoop

En 2006, Yahoo ! se trouve face à un problème de taille. Son infrastructure WebMap, cette technologie qui construit le graphe du web pour alimenter son moteur de recherche, ne suit plus. Le graphe compte plus de 100 milliards de nœuds et 1 000 milliards d'arêtes. L'infrastructure précédente, baptisée Dreadnaught, a atteint sa limite à 800 machines. Il faut repenser l'architecture de fond en comble pour suivre la croissance du web. Yahoo ! fait alors le choix d'Apache Hadoop.

L'histoire débute deux ans plus tôt. En 2004, des chercheurs de Google publient deux articles qui vont tout changer. Ils y décrivent leur système de fichiers et l'architecture MapReduce. Ces publications établissent le socle

technique : un système de fichiers distribué capable de stocker d'immenses volumes de données, associé à un modèle pour les traiter en parallèle à grande échelle.

La première version de Hadoop naît au sein du projet Apache Nutch, avant de prendre son indépendance en 2006. Quatre ans plus tard, Google obtient un brevet sur l'algorithme MapReduce, mais accorde une licence à la Fondation Apache Software. Cette décision ouvre la voie à une adoption massive dans l'industrie.

L'architecture initiale repose sur deux piliers. HDFS (Hadoop Distributed File System) gère le stockage distribué des données. MapReduce coordonne l'exécution des tâches de traitement. Un service central, le JobTracker, supervise l'ensemble des opérations, tandis que des TaskTrackers exécutent les tâches sur chaque nœud du cluster.

Cette architecture monolithique révèle vite ses faiblesses. Les développeurs tentent de détourner MapReduce pour des usages auxquels il n'est pas destiné. Certains lancent des tâches map uniquement pour exécuter des serveurs web ou des calculs itératifs. Ces contournements, malgré leur ingéniosité, compromettent la stabilité et l'efficacité du système.

La communauté Apache engage une refonte majeure. YARN (Yet Another Resource Negotiator) voit le jour. Cette nouvelle version sépare la gestion des ressources du modèle de programmation. Un ResourceManager central attribue les ressources du cluster aux applications. Chaque application dispose de son propre ApplicationMaster pour coordonner son exécution.

Cette refonte architecturale transforme l'écosystème. De nouveaux frameworks émergent : Apache Tez pour l'exécution de graphes de tâches, Spark pour le traitement en mémoire, Storm pour le temps réel, Giraph pour le calcul sur les graphes. Les utilisateurs ne sont plus enfermés dans les limitations de MapReduce.

L'adoption de Hadoop dans l'industrie prend une ampleur considérable. Yahoo ! déploie YARN sur l'ensemble de ses clusters de production. Le système traite quotidiennement environ 500 000 jobs, représentant plus de 230 années de calcul. Sur un cluster de 2 500 machines, le passage à YARN double l'utilisation CPU moyenne. L'entreprise exécute deux fois plus de tâches qu'auparavant.

Les performances progressent. En 2013, l'implémentation MapReduce sur YARN établit des records dans les benchmarks Daytona et Indy GraySort. Le système trie 1,42 téraoctets de données par minute sur un cluster de 2 100 nœuds. Ces résultats démontrent sa capacité à traiter efficacement des volumes massifs.

L'architecture de Hadoop influence la conception des systèmes distribués

modernes. Sa capacité à fonctionner sur du matériel standard, sa tolérance aux pannes et son modèle de programmation simple en font une référence. Le concept de localité des données, où le calcul se déplace vers les données plutôt que l'inverse, sera un principe déterminant de l'informatique distribuée.

Le succès de Hadoop inspire le développement de nombreuses plateformes concurrentes. Twitter crée Mesos, Facebook conçoit Corona, Google élabore Omega. Chaque système apporte ses innovations, mais tous partagent l'héritage architectural de Hadoop : la séparation entre la gestion des ressources et les modèles de programmation.

L'évolution de Hadoop reflète la transformation des besoins en traitement de données. D'un outil spécialisé pour l'indexation du web, il se transforme en plateforme générique supportant une multitude d'applications. Cette transformation s'accompagne d'améliorations continues en matière de performances, de flexibilité et de facilité d'utilisation.

En démontrant la viabilité du traitement distribué à grande échelle sur du matériel standard, Hadoop démocratise l'analyse des données massives. Des entreprises de toutes tailles peuvent désormais construire des infrastructures de traitement de données performantes sans investir dans du matériel spécialisé coûteux. La collaboration entre entreprises, développeurs et chercheurs au sein de la Fondation Apache a permis des avancées rapides et une large diffusion des connaissances.

jQuery

En 2005, John Resig était étudiant en informatique au Rochester Institute of Technology. Il passait ses journées à coder des sites web et butait sans cesse sur le même problème : faire fonctionner du JavaScript sur différents navigateurs relevait du parcours du combattant. Chaque navigateur avait ses propres règles, ses propres caprices. Il fallait écrire plusieurs versions du même code, tester, corriger, recommencer. Cette situation épuisait les développeurs et ralentissait tout le monde.

Il décida de créer sa propre solution. L'année suivante, lors d'un événement technologique appelé BarCampNYC, il présenta la première version de jQuery. Son ambition tenait en une phrase : rendre la programmation JavaScript agréable. Il s'était inspiré du travail de Charles Willison, un développeur britannique qui avait créé `css_selector.js`, une bibliothèque permettant de sélectionner des éléments CSS. Resig reprit cette idée et la poussa beaucoup plus loin.

jQuery proposait une interface simple pour manipuler les pages web. Les développeurs pouvaient sélectionner des éléments HTML avec des expressions complexes, les modifier, les animer. Le moteur de sélection fonctionnait remarquablement bien. Plus tard, ce moteur fut extrait dans une bibliothèque à part, baptisée Sizzle, tant il représentait une avancée technique.

Mais la vraie force de jQuery ne résidait pas que dans son code. Resig avait compris quelque chose que beaucoup d'autres développeurs négligeaient : une bonne documentation valait de l'or. Entre 2006 et 2007, jQuery fut la seule bibliothèque JavaScript open source à proposer une documentation complète et claire. Les autres projets laissaient les développeurs se débrouiller avec le code source. Cette attention aux détails changea tout.

Resig recruta d'abord un responsable communautaire. Pas un développeur supplémentaire, non : quelqu'un dont le travail consistait à répondre aux questions, à accompagner les utilisateurs. Il voyait loin et savait qu'un outil technique ne se diffuse pas tout seul.

Au début, jQuery resta un projet parallèle. Resig travaillait sur d'autres choses. Son embauche chez Mozilla changea la donne. L'entreprise lui permit de consacrer du temps de travail à jQuery. Il put alors bâtir l'infrastructure nécessaire, préparer le projet à durer sans lui. Il quitta ensuite Mozilla pour Khan Academy, mais jQuery était lancé.

La bibliothèque connut un succès foudroyant à la fin des années 2000. Elle rendait accessibles des techniques sophistiquées comme les animations ou les requêtes Ajax. Les développeurs l'adoptèrent massivement. D'après W3Techs, environ 74% des sites web utilisèrent jQuery à son apogée. Un chiffre vertigineux.

Le projet grandit, sa gouvernance aussi. Un jQuery Board vit le jour en 2011, suivi d'une jQuery Foundation en 2012. Puis vint une fusion avec la Dojo Foundation en 2015, créant la JS Foundation. En 2019, nouvelle fusion avec la Node.js Foundation : l'OpenJS Foundation naquit, et jQuery y conserva sa place de projet majeur.

Mais le web évoluait. Les éditeurs de navigateurs—Apple, Google, Microsoft et Mozilla—commencèrent à travailler ensemble au sein du Web Hypertext Application Technology Working Group. Les incompatibilités entre navigateurs diminuèrent. Les navigateurs eux-mêmes intégrèrent des fonctionnalités qui rendaient jQuery moins indispensable. L'API Fetch remplaça les fonctions Ajax. Les méthodes querySelector et querySelectorAll permirent de sélectionner des éléments comme le faisait jQuery. L'interface classList simplifia la manipulation des classes CSS.

Les navigateurs adoptèrent des mises à jour automatiques et continues. Les corrections et nouveautés atteignaient les utilisateurs automatique-

ment, sans action manuelle. Les développeurs purent exploiter plus vite les nouvelles fonctionnalités du web. Ils eurent moins besoin de solutions de compatibilité.

Internet Explorer mourut progressivement. Microsoft abandonna le support des versions 10 et antérieures en 2016, ne conservant qu'IE 11. Cette disparition libéra les développeurs des contraintes qui avaient justifié jQuery au départ.

De nouveaux frameworks apparurent : React, Angular, Vue. Ils proposaient une approche différente, par composants, facilitant la construction d'interfaces complexes. Ils encourageaient une programmation déclarative où le développeur décrit l'état souhaité plutôt que les étapes pour y parvenir. jQuery, lui, restait impératif.

Certains acteurs majeurs abandonnèrent jQuery. GitHub le retira de son interface. Bootstrap annonça qu'il s'en affranchirait dans sa version 5, jQuery constituant sa principale dépendance JavaScript côté client.

Pourtant, jQuery ne disparut pas. Pour les sites simples nécessitant peu d'interactivité, la bibliothèque reste une solution éprouvée et facile à mettre en œuvre. Son approche impérative demeure plus accessible aux débutants que les paradigmes déclaratifs modernes. Dans certains contextes professionnels où d'anciennes versions d'Internet Explorer tournent encore, jQuery garde son utilité. La bibliothèque continue d'évoluer, maintenue par une communauté fidèle. Beaucoup de développeurs apprécient sa syntaxe concise et l'utilisent par préférence, tandis que des alternatives natives existent. La progression des standards web et l'émergence de nouvelles approches témoignent d'une évolution continue des technologies, un processus auquel jQuery a largement participé en établissant des pratiques qui perdurent.

PowerShell

En 2002, Jeffrey Snover rédige chez Microsoft un document qui va changer l'administration système sous Windows. Le « Manifeste Monad » expose une vision alors peu commune : créer un shell qui manipule des objets plutôt que du texte. Le nom du projet renvoie à Leibniz et son concept d'unités fondamentales capables, par agrégation, de former des systèmes plus élaborés.

L'administration de Windows traversait à l'époque une crise d'identité. D'un côté, des outils graphiques simples pour les opérations basiques. De

l'autre, des langages de programmation complexes pour l'automatisation avancée. Entre les deux, un vide. Les administrateurs peinaient à composer des solutions d'automatisation sans basculer dans la programmation lourde. Microsoft avait passé des années à simplifier Windows pour les novices, rendant paradoxalement certaines tâches plus ardues pour les utilisateurs expérimentés qui privilégiaient l'efficacité des lignes de commande.

Snover s'inspire des shells UNIX tout en cherchant à dépasser leurs limites. Dans ces environnements, les commandes produisent du texte non structuré que les commandes suivantes doivent analyser, un processus fragile et source d'erreurs. L'idée novatrice de PowerShell tient dans son approche orientée objet : le shell transmet des objets .NET structurés entre les commandes, éliminant cette fragilité.

Le projet faillit ne jamais voir le jour. Entre 2004 et 2006, plusieurs initiatives majeures basées sur du code managé chez Microsoft connaissent des échecs retentissants, provoquant les retards de Windows Vista. Promouvoir un langage de script reposant sur .NET est risqué. L'équipe Exchange Server, première à adopter PowerShell, développe d'ailleurs une couche d'API intermédiaire qui lui permettrait de basculer vers une autre solution en cas de problème.

PowerShell arrive finalement en 2006. Il introduit les « cmdlets », des commandes légères implémentées comme des classes .NET. Cette architecture garantit une cohérence syntaxique et sémantique rare dans les shells traditionnels, où chaque commande définit ses propres conventions. La version 2.0, livrée avec Windows 7, enrichit l'outil avec l'exécution de scripts à distance et les tâches en arrière-plan.

La standardisation des verbes dans les noms de commandes (Get-, Set-, New-) et l'introduction des *providers* marquent une rupture. Ces derniers traitent différentes sources de données – système de fichiers, registre Windows, Active Directory – avec une syntaxe uniforme. Les administrateurs n'ont plus à réapprendre un jeu de commandes pour chaque source, ce qui réduit considérablement la courbe d'apprentissage.

L'adoption s'accélère quand Microsoft intègre PowerShell dans ses produits serveur. Exchange Server 2007 rend PowerShell obligatoire pour certaines tâches. Cette décision force les administrateurs à franchir le pas, se rendant vite à l'évidence des avantages en termes d'efficacité et d'automatisation.

L'architecture du shell rompt avec l'approche traditionnelle. Au lieu de créer des exécutables autonomes, les développeurs écrivent des classes .NET héritant d'une classe de base commune. Celle-ci fournit gratuitement l'analyse des paramètres, la validation des données, la journalisation ou la gestion des erreurs. Cette mutualisation du code réduit les coûts de déve-

loppement et améliore l'expérience utilisateur.

L'environnement de développement intégré (ISE), ajouté plus tard, offre aux administrateurs un outil moderne pour écrire et déboguer des scripts. Coloration syntaxique, auto-complétion et débogueur graphique accompagnent la transition des interfaces graphiques vers l'automatisation par script.

Une communauté active se forme autour de PowerShell. Les administrateurs partagent scripts et bonnes pratiques, tandis que les éditeurs de logiciels adoptent l'outil pour administrer leurs produits. Cette adoption valide la vision initiale de Snover d'une plateforme unifiée d'automatisation.

La sécurité occupe une place centrale dans PowerShell. Le shell intègre des mécanismes de signature de code et de journalisation détaillée des activités. Les organisations peuvent contrôler l'exécution des scripts et tracer leur utilisation, répondant aux exigences de sécurité et de conformité.

L'impact sur l'administration Windows se révèle considérable. PowerShell transforme l'interaction avec le système, remplaçant les actions manuelles répétitives par des solutions automatisées reproductibles. Cette évolution améliore l'efficacité et réduit les erreurs humaines dans les opérations système.

En 2016, Microsoft rend PowerShell open source sous licence MIT et multiplateforme. Cette décision reflète l'évolution de l'entreprise vers plus d'ouverture et reconnaît la réalité des environnements informatiques modernes, où Windows coexiste avec Linux et macOS.

MongoDB

En 2007, trois entrepreneurs américains, Dwight Merriman, Eliot Horowitz et Kevin Ryan, se lancent dans un projet de plateforme en tant que service. Ils constatent que les bases de données relationnelles traditionnelles peinent à répondre aux exigences d'extensibilité et d'agilité de leur infrastructure. Cette contrainte les pousse à imaginer leur propre solution.

Le contexte de l'époque voit s'effriter le mythe de la base de données universelle. Les architectures informatiques se diversifient, chacun cherche des réponses à des problèmes spécifiques. Les bases NoSQL émergent, les bases temporelles aussi. Cette fragmentation du paysage technologique oblige à repenser les choix d'outillage, car chaque type de système répond à des besoins particuliers.

MongoDB se construit autour d'un modèle orienté document qui utilise BSON, une variante binaire de JSON. Cette architecture autorise une représentation souple des données tout en maintenant des performances élevées. Le système gère finement le cache et la localisation des informations, avec une limite fixée à 16 Mo par document pour garantir une utilisation optimale de la mémoire.

L'évolution technique apporte son lot d'innovations. WiredTiger, le moteur de stockage adopté par la suite, introduit la conformité ACID dans les transactions. MongoDB parvient ainsi à marier la souplesse du NoSQL avec la rigueur transactionnelle des systèmes relationnels classiques.

La montée en charge horizontale figure parmi les atouts majeurs de MongoDB. Deux mécanismes de fragmentation coexistent : par plage et par hachage. Cette architecture distribue les données sur plusieurs serveurs, rendant l'extension transparente pour les applications. Le routeur de requêtes exploite cette répartition géographique pour accélérer les traitements, le code écrit pas les développeurs peut donc être identique quelle que soit la distribution physique des données.

La disponibilité atteint 99,99% sans intervention humaine grâce à la réplication automatique et aux capacités d'auto-réparation. Les jeux de réplicas basculent automatiquement lors d'une panne, un nœud secondaire prenant la place du primaire selon un processus d'élection bien rodé.

La sécurité, souvent le talon d'Achille des bases NoSQL, a bénéficié d'améliorations continues. MongoDB intègre désormais le chiffrement au repos via TDE, l'authentification LDAP et Kerberos, le chiffrement TLS/SSL, ainsi que des protections contre les injections SQL. Ces ajouts le placent en bonne position face à ses concurrents NoSQL.

Les cas d'usage se dessinent naturellement. Les applications d'analyse en temps réel sur de gros volumes, comme celles des marchés financiers ou du traitement de flux Twitter, trouvent en MongoDB un allié de choix. Le commerce électronique exploite ses capacités d'analyse comportementale pour affiner l'expérience client. L'Internet des Objets constitue un autre terrain de prédilection, la flexibilité du modèle de données gérant sans difficulté l'hétérogénéité des informations remontées par les capteurs.

La commercialisation se décline en deux versions : entreprise et open source. Cette double approche crée une communauté active de développeurs tout en assurant des revenus réguliers. L'intégration aux plateformes cloud majeures comme Amazon Web Services et Azure renforce sa présence sur le marché des bases de données modernes.

Quelques limites apparaissent néanmoins. MongoDB s'adapte mal aux données fortement relationnelles, aux environnements réclamant une intégrité référentielle stricte, ou aux opérations multipliant les jointures entre

documents. L'absence de procédures stockées et de déclencheurs contraint la logique métier hors de la base. Les entrepôts de données et les représentations graphiques complexes ne correspondent pas à ses points forts.

L'influence de MongoDB sur l'écosystème des bases de données reste tangible. Son succès a validé l'approche NoSQL pour les applications d'entreprise et façonné la conception des systèmes modernes de gestion de données. Sa capacité à traiter dans un même environnement des données non structurées, semi-structurées et structurées a redéfini certains standards industriels.

Le développement continu reflète les mutations des besoins en gestion de données. La progression vers les architectures distribuées, l'importance croissante de l'analyse instantanée et l'apparition de nouveaux modèles d'application ont tracé son évolution. Cette adaptation permanente au marché explique sa position actuelle.

Apple iPhone

En janvier 2007, Steve Jobs monte sur la scène du Macworld à San Francisco pour présenter un appareil dont presque personne ne connaissait l'existence. Deux années de secret absolu ont précédé ce moment. Seuls trois dirigeants d'AT&T avaient pu voir l'iPhone avant sa révélation au grand public. Jobs avait négocié directement avec l'opérateur américain un partenariat exclusif qui allait redessiner les rapports de force dans le monde des télécommunications.

Le marché des téléphones en 2007 ressemblait à ce qu'il était quelques années auparavant. BlackBerry dominait avec ses claviers physiques, Palm proposait ses Treo, et les utilisateurs s'accommodaient d'interfaces complexes qui demandaient un vrai temps d'apprentissage. Internet sur mobile restait une expérience frustrante, la messagerie fonctionnait tant bien que mal, et l'idée d'un téléphone vraiment intuitif paraissait hors de portée.

L'iPhone arrive avec son écran tactile de 3,5 pouces et une idée simple : on touche directement ce qu'on veut faire. Pas de stylet, pas de clavier caché sous l'écran. Les doigts suffisent, et la surface réagit à plusieurs points de contact simultanés. Apple a breveté cette technologie multipoint et l'a placée au cœur d'un appareil qui prétend être trois choses à la fois : un téléphone, un iPod, et un moyen de naviguer sur Internet. Cette convergence n'avait jamais vraiment fonctionné auparavant.

Jobs applique à l'iPhone sa maniaquerie habituelle. Il scrute chaque

vis, examine les courbes du boîtier sous tous les angles, et va jusqu'à faire refaire l'emballage parce que la bande adhésive ne se trouve pas exactement où elle devrait être. Chez Apple, le déballage du produit compte autant que le produit. Cette obsession du détail traduit une vision particulière de ce que doit être un objet technologique : quelque chose qui fonctionne avant qu'on ait lu le mode d'emploi.

Quand l'iPhone arrive dans les magasins en juin 2007, les files d'attente se forment spontanément. Apple et AT&T vendent deux versions : 499 dollars pour 4 Go, 599 dollars pour 8 Go. Ces prix n'effraient personne. L'entreprise vise modestement 1% du marché mondial pour 2008, soit 10 millions d'unités. Elle dépassera largement cet objectif.

La distribution rompt avec les habitudes du secteur. Pas question de voir l'iPhone chez Best Buy ou RadioShack. Apple impose ses propres magasins et ceux d'AT&T comme seuls points de vente. Cette restriction donne entièrement à Apple le contrôle de la présentation de l'appareil et de l'expérience d'achat. Les revendeurs traditionnels rongeront leur frein pendant des mois.

Techniquement, l'iPhone accumule les trouvailles comme le détecteur de proximité qui éteint l'écran quand on approche le téléphone de l'oreille. Un capteur de luminosité ajuste la brillance selon la lumière ambiante. Un accéléromètre fait pivoter l'affichage quand on tourne l'appareil. Le système d'exploitation dérive d'OS X et gère le multitâche avec une efficacité qui étonne les observateurs. Apple a construit matériel et logiciel en même temps, les faisant dialoguer dès la conception. Ses concurrents séparent ces deux mondes.

Le succès transforme Apple. L'entreprise abandonne le mot « Computer » dans sa raison sociale au profit de Apple Inc., tout simplement. Ce changement reflète la réalité d'une société qui ne vend plus seulement des ordinateurs depuis longtemps. L'iPhone accélère cette mutation et fait basculer le centre de gravité vers les appareils mobiles.

La culture du secret règne dans l'entreprise. Les équipes travaillent sur des fragments de projet sans voir l'ensemble. Seule une poignée de responsables connaît la vision complète. Cette compartimentation frustre certains ingénieurs mais garantit que rien ne fuite avant les annonces officielles. Jobs dirige l'ensemble depuis une structure très centralisée, refusant de créer des divisions autonomes qui pourraient diluer le contrôle.

L'iPhone débarque au Royaume-Uni et en France fin 2007, puis en Asie l'année suivante. Dans chaque pays, Apple reproduit le schéma américain : un seul opérateur, un accord exclusif, une distribution maîtrisée. Ces pratiques soulèvent des questions juridiques en Europe, où la vente liée d'un téléphone à un opérateur unique bute sur certaines réglementations. Apple

négocie au cas par cas.

Les concurrents réagissent comme ils peuvent. Nokia lance son N95 avec des capacités multimédia renforcées. D'autres fabricants se précipitent sur les écrans tactiles, souvent sans comprendre que la réussite d'Apple tient autant à l'interface logicielle qu'au matériel. Le marché bascule en quelques mois : ce que l'iPhone propose est la nouvelle référence.

En 2008, Apple ouvre l'App Store et change encore les règles. Les développeurs externes peuvent créer des applications et les vendre directement aux utilisateurs. L'iPhone gagne ainsi des milliers de fonctions que personne n'avait imaginées au départ. Un écosystème économique naît presque instantanément, attirant programmeurs indépendants et grandes entreprises.

L'université Harvard estime la valeur de la publicité gratuite générée par le lancement de l'iPhone à 400 millions de dollars. Apple n'a presque rien dépensé en campagnes traditionnelles. Les médias ont fait le travail gratuitement, relayant chaque annonce, chaque file d'attente, chaque première impression. Cette réussite marketing repose sur une préparation minutieuse et un sens aigu du spectacle.

La valeur boursière d'Apple double dans les douze mois qui suivent. Les analystes comprennent que l'iPhone n'est pas un produit parmi d'autres mais le nouveau pilier de l'entreprise. Jobs avait vu juste : un appareil qui combine téléphone, lecteur multimédia et navigateur Internet dans une interface fluide peut changer les usages. C'est une rupture dans l'histoire de l'informatique mobile et des télécommunications.

Clojure

En 2005, Rich Hickey se lance dans l'aventure Clojure. Il finance cette entreprise avec ses économies de retraite, prenant une année sabbatique pour concevoir un langage qui résoudrait les problèmes qu'il rencontre quotidiennement depuis des années. Son ambition ? Créer un outil aussi acceptable en entreprise que Java ou C#, mais débarrassé des complexités inutiles qui encombrent le développement des systèmes d'information.

Hickey n'est pas un novice. Depuis 1987, il travaille comme architecte logiciel, naviguant entre C++, Java et C#. Cette longue expérience lui a appris une chose : les langages orientés objet traditionnels imposent un prix élevé. La programmation impérative avec état mutable exige des efforts démesurés dès qu'il faut modifier un programme sans rompre la cohérence des

données. Ces systèmes rigides résistent au changement malgré l'encapsulation et les abstractions.

Parallèlement à son travail, Hickey explore Common Lisp dans ses projets personnels. Cette pratique lui ouvre les yeux : toute cette complexité qu'il combat au bureau n'est pas une fatalité. Avec la flexibilité de Lisp, on attaque un problème exactement au bon niveau, sans couches superflues. Mais Common Lisp reste cantonné aux expérimentations. Les entreprises refusent de déployer des applications écrites dans ce langage. À deux reprises, Hickey doit réécrire en C++ des systèmes qu'il avait développés en Common Lisp, ou les transformer en procédures stockées SQL pour satisfaire ses clients.

L'idée germe alors de marier la puissance de Lisp avec l'acceptabilité de Java. Hickey choisit la machine virtuelle Java comme fondation. Cette décision tactique élimine d'emblée les obstacles à l'adoption : Clojure est un citoyen de première classe de l'écosystème Java, avec son interopérabilité native et sa capacité à réutiliser les bibliothèques existantes.

Le premier défi technique se posa pour rendre les structures de données immuables suffisamment performantes pour un usage pratique. Durant l'hiver 2006-2007, Hickey implémente une version persistante des tables de hachage à essai mappé avec un facteur de branchement de 32. Les primitives de copie de tableau de Java se révèlent efficaces. Cette percée délivre des performances quasi-constantes pour les opérations courantes, rendant l'immutabilité enfin viable en production.

En octobre 2007, la première version alpha de Clojure apparaît sur la liste de diffusion jFli. Le site clojure.org ouvre ses portes, accompagné d'un groupe Google. La réaction dépasse toutes les prévisions : 30 000 visites en trois jours. Les dix-huit mois qui suivent voient le langage s'étoffer considérablement. La bibliothèque standard quadruple, passant de 100 à plus de 400 fonctions.

Clojure se distingue par sa gestion de l'état. Le langage propose plusieurs types de références selon les besoins : les refs utilisent un système transactionnel inspiré des bases de données, les agents gèrent des mises à jour asynchrones, les atoms offrent des modifications atomiques simples. Cette approche contraste avec le modèle uniforme mais limité des langages impératifs traditionnels.

Mai 2009 marque la sortie de Clojure 1.0, accompagnée du premier ouvrage consacré au langage. La version 1.1 arrive en décembre avec les séquences chunked et les transients, apportant des gains de performance substantiels. La version 1.2 en 2010 introduit protocoles et types définis par l'utilisateur, complétant l'arsenal polymorphique du langage.

L'année 2011 ouvre un nouveau chapitre avec ClojureScript. Cette im-

plémentation s'exécute sur les moteurs JavaScript, étendant Clojure au développement web côté client. Le projet démontre la robustesse des abstractions du langage : elles fonctionnent sur une plateforme radicalement différente de la JVM sans modification conceptuelle.

En 2012, Hickey lance Datomic, une base de données qui transpose les principes de Clojure au stockage : immutabilité, temps explicite, sémantique fonctionnelle. Le format EDN (Extensible Data Notation) formalise la syntaxe de sérialisation utilisée par Clojure, offrant une alternative aux formats traditionnels.

La communauté grandit à son rythme. Les conférences Clojure/conj débutent en 2010. ClojureBridge naît en 2014 pour diversifier l'écosystème à travers des ateliers gratuits. Le langage trouve sa place dans des secteurs variés : finance, climatologie, commerce de détail, analyse de données, édition, santé, publicité, génomique.

Des acteurs majeurs l'adoptent pour des systèmes critiques. Walmart traite les reçus électroniques de ses 5 000 magasins avec Clojure. Netflix analyse 2 000 milliards d'événements quotidiens grâce au langage. Nubank, banque numérique comptant plus de 12 millions de clients, bâtit son infrastructure sur Clojure et Datomic.

Le succès repose sur une programmation fonctionnelle avec structures immuables performantes, une interopérabilité native avec la plateforme hôte, des abstractions puissantes pour la concurrence, et un traitement générique des données. Ces atouts font de Clojure un outil adapté aux systèmes d'information contemporains. En 2025, le développement continue son cours mesuré, améliorant outillage et expérience développeur plutôt que d'accumuler des fonctionnalités.

F#

Microsoft cherche à consolider sa plateforme .NET face à la montée en puissance de Java. Le langage F# naît alors d'une rencontre singulière : celle entre les travaux académiques sur la programmation fonctionnelle et les besoins pragmatiques de l'industrie logicielle.

Les racines de F# plongent dans les années 1970, avec les travaux de Robin Milner sur ML (Meta Language). Ce langage introduisait l'inférence de types et le *pattern matching*, des concepts pensés initialement pour manipuler des preuves mathématiques. La famille ML, qui compte Standard ML et OCaml, forme le terreau théorique dont F# héritera les principes

fondamentaux.

En 1997, Microsoft lance le « Projet 7 », une initiative ambitieuse visant à intégrer quatorze langages, académiques et commerciaux, dans l'écosystème .NET. Don Syme, chercheur chez Microsoft Research à Cambridge, commence en 2002 le développement de F#. Son objectif ? Marier l'élégance des langages fonctionnels avec la richesse de la plateforme .NET.

La première version publique arrive en 2005. F# se distingue par sa capacité à fusionner programmation fonctionnelle et orientée objet. Mais surtout, il offre une interopérabilité naturelle avec les autres langages .NET. Les développeurs peuvent utiliser les bibliothèques existantes sans friction, un atout déterminant pour l'adoption.

Grâce au système de types inféré, hérité de ML, le compilateur déduit automatiquement les types des expressions. Le code gagne en concision sans sacrifier la sûreté. Les « active patterns » facilitent le pattern matching extensible, tandis que les « computation expressions » simplifient l'écriture de code asynchrone. Ces mécanismes transforment la manipulation de données complexes en opérations fluides.

Microsoft intègre F# à sa gamme officielle en 2007. Le langage trouve sa place dans les domaines financier et scientifique, où la rigueur du typage et la concision du code fonctionnel répondent à des exigences critiques. F# 2.0 sort en 2010, première mouture pleinement supportée par Visual Studio.

En 2012 F# 3.0 introduit des « type providers ». Ce mécanisme génère automatiquement des types correspondant aux schémas de données externes, qu'elles proviennent de bases SQL, de services web ou de fichiers structurés. L'intégration de données rendue presque triviale, des institutions comme Credit Suisse et Morgan Stanley adoptent F# pour leurs systèmes d'analyse et de trading, où chaque erreur peut coûter des millions.

En 2010, Microsoft publie le code source de F# sous licence Apache 2.0. Cette ouverture catalyse la formation d'une communauté de contributeurs. La F# Software Foundation voit le jour en 2014, donnant un cadre organisationnel au développement du langage. F# sort progressivement de l'ombre de Microsoft pour devenir un projet communautaire vivant.

Le langage s'adapte aux mutations du paysage informatique. Des projets comme WebSharper et Fable permettent d'écrire des applications web en F#. Le support de .NET Core ouvre la voie au développement multiplateforme. F# n'est plus cantonné à Windows ; il fonctionne sur Linux et macOS, s'inscrivant dans l'ère du cloud computing.

C# emprunte quelques unes des fonctionnalités de F# : le pattern matching, les types non *nullables*, certaines constructions fonctionnelles. L'approche de la programmation asynchrone via les *computation expressions* inspire aussi d'autres langages. F# démontre qu'un langage fonctionnel

peut cohabiter avec la programmation orientée objet, mais aussi l'enrichir. Il reste un langage de niche, certes, mais une niche où la qualité du code et la sûreté des applications comptent plus que tout.

Groovy

En 2003, Java trônait sur le monde informatique avec sa bibliothèque de composants considérable. Pourtant, son architecture rigide décourageait ceux qui voulaient simplement scripter une tâche, prototyper ou écrire du code minimal. Ruby, Python et Smalltalk paraissaient plus agiles, mais leur syntaxe déroutait les programmeurs habitués à Java. James Strachan créa alors Groovy : un langage qui resterait fidèle à l'esprit Java tout en éliminant ses lourdeurs.

L'idée était simple : garder l'apparence de Java pour ne pas désorienter les développeurs, mais alléger la syntaxe et réduire les contraintes. Strachan voulait un « Python à saveur Java », où Groovy servirait de colle entre les bibliothèques Java existantes. Le langage devait s'intégrer naturellement à l'écosystème de la JVM sans bouleverser les habitudes.

Mars 2004 marqua une étape décisive : Groovy fut soumis comme Java Specification Request (JSR 241) et accepté par vote. Ce processus exigeait une spécification rigoureuse, une implémentation de référence et des tests de compatibilité. La charge de travail s'avéra écrasante. Des tensions émergèrent au sein de la communauté sur les directions à prendre. Strachan finit par se retirer, laissant Guillaume Laforge reprendre le flambeau.

Le JSR ne fut jamais voté dans sa forme finale, mais ce passage obligé força l'équipe à structurer le langage. Sans cette contrainte, les fonctionnalités accumulées pendant le développement initial auraient sans doute formé un ensemble disparate plutôt qu'un tout cohérent.

Janvier 2007 vit naître la version 1.0. Elle apportait les fermetures lexicales, les scripts, les constructeurs, les chaînes interpolées (GStrings) et les paramètres nommés. Le langage intégrait nativement les expressions régulières, la surcharge d'opérateurs et une syntaxe littérale pour les listes et les cartes. Le Groovy Development Kit enrichissait environ 430 classes Java standard avec de nouvelles méthodes.

Deux ans plus tard, Groovy dépassait son rôle initial de langage de script. On l'utilisait désormais pour développer des applications complètes. Ses capacités de métaprogrammation à l'exécution, déjà robustes, furent complétées par des fonctionnalités à la compilation. Cette évolution permit

de réduire la complexité du code sans sacrifier les performances.

En 2012, le langage gagnait du terrain sur des projets traditionnellement réservés à Java. Les utilisateurs réclamaient plus de vérifications de types à la compilation et de meilleures performances. L'équipe ajouta une nature statique au langage. Groovy ne renonçait pas à son dynamisme : il offrait simplement le choix entre typage statique et dynamique selon les besoins du projet.

Les transformations AST (Abstract Syntax Tree), introduites avec la version 1.6, changèrent la donne. Elles permettaient d'étendre le langage sans toucher à sa grammaire. Cette approche déclarative, basée sur des annotations, facilitait l'application de motifs de conception Java. L'annotation @Immutable, par exemple, générait automatiquement tout le code nécessaire pour créer une classe immuable conforme aux bonnes pratiques.

La version 2.4 apporta le support d'Android, ce qui entraîna de nombreuses optimisations dans la génération de *bytecode*. La quantité de code produit diminua, la consommation mémoire des structures internes chuta, et les performances s'améliorèrent. Février 2020 vit arriver la version 3.0 avec son nouveau parseur basé sur Antlr4, offrant une meilleure compatibilité avec la syntaxe Java moderne.

L'histoire organisationnelle de Groovy ressemble à un parcours du combattant. D'abord hébergé sur Codehaus, le projet migra vers la Apache Software Foundation en 2015 après la fermeture de la plateforme. G2One, SpringSource, VMware, Pivotal et OCI sponsorisèrent tour à tour son développement. Malgré ces changements, la communauté open source maintint le cap et assura la progression du langage.

Groovy laissa son empreinte sur d'autres technologies. Kotlin s'inspira de ses fermetures lexicales, de son concept de constructeur, du paramètre par défaut *it* pour les fermetures, des *templates* et chaînes interpolées, de l'opérateur Elvis et de la navigation sécurisée. Swift et C# adoptèrent des fonctionnalités similaires comme l'opérateur de navigation sécurisée.

La philosophie de Groovy repose sur l'extensibilité pragmatique. Plutôt que d'intégrer toutes les fonctionnalités possibles dans le cœur du langage, il fournit des mécanismes pour que les utilisateurs enrichissent le langage selon leurs besoins. Les transformations AST, les extensions de vérification de types et la création de langages dédiés (DSL) illustrent cette approche.

Le langage reste vivant et sa communauté continue d'innover tout en préservant la compatibilité avec Java, qui demeure au cœur de sa conception. Groovy tient son équilibre entre productivité accrue pour les développeurs Java et fonctionnalités dynamiques pour des cas d'usage spécifiques.

Android

En 2003, Andy Rubin fonde Android Inc. à Palo Alto, avec l'ambition de créer un système d'exploitation pour appareils mobiles. Ce nom lui va bien : chez Apple puis chez MSN, ses collègues l'avaient surnommé « Android » à cause de sa passion pour la robotique. Quelques années plus tôt, il avait déjà monté Danger Inc., la société qui donna naissance au Danger Hiptop, ce téléphone que T-Mobile commercialisa sous le nom de Sidekick en 2002.

Au départ, l'équipe d'Android Inc. mise sur les appareils photo numériques. Mais ce marché s'effondre tandis que les téléphones mobiles prennent leur essor. Rubin et ses cofondateurs Rich Miner, Nick Sears et Chris White décident alors de pivoter vers un système d'exploitation mobile bâti sur le noyau Linux. L'argent manque terriblement. Steve Perlman, un ami investisseur, doit même apporter 10 000 dollars en liquide pour éviter la faillite.

Google rachète Android Inc. en 2005 pour 50 millions de dollars. Larry Page et Sergey Brin voient loin : ils veulent étendre leur empire au-delà de la recherche sur ordinateur. Le 11 juillet 2005, toute l'équipe d'Android rejoint les bureaux de Mountain View.

À cette époque, le paysage mobile appartient à d'autres. Symbian de Nokia écrase tout avec 63,5% du marché mondial en 2007. Windows Mobile de Microsoft grappille 12%, BlackBerry de RIM se taille 9,6%. Et Apple débarque en janvier 2007 avec l'iPhone, redéfinissant ce qu'est un smartphone. Face à ces géants, Google choisit une voie différente : proposer un système ouvert et gratuit.

L'Open Handset Alliance naît en 2007. Ce consortium rassemble 34 entreprises, des fabricants de téléphones aux opérateurs en passant par diverses sociétés technologiques. Leur mission est de développer des standards ouverts pour les appareils mobiles. Android est le système de référence de cette alliance, distribué en open source sous licence Apache.

La première version publique sort en 2008. L'architecture repose sur le noyau Linux, qui gère le matériel. Au-dessus tourne Dalvik, une machine virtuelle spécialement conçue par Google pour optimiser l'exécution sur mobile. Chaque application fonctionne dans sa propre instance de cette machine virtuelle, plusieurs programmes peuvent donc s'exécuter simultanément.

Les versions se succèdent, baptisées de noms de desserts. Cupcake (1.5) arrive en 2009 avec les widgets et la navigation GPS. Donut (1.6) améliore la recherche et accepte différentes résolutions d'écran. Éclair (2.0-2.1) gère différents comptes e-mail et peaufine l'interface.

Froyo (2.2) en 2010 accélère les performances et renforce la sécurité. Gingerbread (2.3) intègre le NFC et économise mieux la batterie. Honeycomb (3.0) en 2011 adapte Android aux tablettes. Ice Cream Sandwich (4.0) unifie l'expérience entre smartphones et tablettes, avec des nouveautés comme le déverrouillage facial.

Jelly Bean (4.1-4.3) en 2012 rend l'interface plus fluide grâce au « Project Butter ». KitKat (4.4) allège le système pour qu'il tourne sur des machines modestes. Lollipop (5.0) en 2014 introduit le « Material Design », nouvelle esthétique voulue par Google. Marshmallow (6.0) repense les permissions des applications et prolonge l'autonomie.

Le modèle économique tranche avec celui des concurrents. Google offre le système aux fabricants, se rémunérant via la publicité et les services comme le Play Store. Apple garde son écosystème fermé, Microsoft facture des licences : Android prend une troisième voie.

L'adoption explose. En 2010, plus de 100 millions d'appareils tournent sous Android. En 2021, on en compte 3 milliards dans le monde. Cette domination attire l'attention des autorités de la concurrence. La Commission européenne inflige 4,34 milliards d'euros d'amende à Google en 2018 pour abus de position dominante.

Le système déborde vite de son cadre initial. Il équipe des tablettes, des montres connectées, des télévisions, des voitures. Android Wear s'occupe des objets connectés, Android Auto de l'automobile, faisant de ce système une plateforme universelle pour l'informatique mobile.

L'architecture technique évolue avec les besoins. ART remplace progressivement Dalvik pour améliorer les performances et l'efficacité énergétique. Le système intègre reconnaissance faciale, paiement sans contact, réalité augmentée. La sécurité se renforce continuellement : chiffrement des données, isolation des applications dans des *sandbox*, mises à jour mensuelles. Le modèle de permissions plus transparent donne à l'utilisateur un contrôle accru.

Android a transformé l'industrie mobile. Le système démocratise l'accès aux smartphones en permettant aux constructeurs de proposer des appareils à tous les prix. Il stimule l'innovation grâce à une plateforme ouverte aux développeurs. Son modèle open source inspire d'autres projets et influence l'évolution des systèmes mobiles.

Cette réussite tient à un équilibre : la puissance du noyau Linux, l'efficacité d'une machine virtuelle pensée pour le mobile, une interface intuitive. Android satisfait les constructeurs, les développeurs et les utilisateurs, tout en s'adaptant aux mutations technologiques et aux nouveaux usages.

Google Chrome

En septembre 2008, Google lance la version bêta de Chrome, son propre navigateur web. À cette époque, Internet Explorer domine le marché et Firefox s'est imposé comme l'alternative incontournable. Mais les applications web gagnent en complexité, HTML5 est en préparation et les navigateurs existants montrent leurs limites. Google constate que les navigateurs gèrent mal ces nouvelles applications, provoquant régulièrement des plantages qui affectent toute la navigation.

Chrome arrive avec une architecture multi-processus. Chaque onglet tourne dans son propre processus, isolé des autres. Quand une page plante, seul l'onglet concerné se ferme. Les autres composants du navigateur continuent de fonctionner normalement. Cette rupture avec les navigateurs monolithiques traditionnels transforme l'expérience utilisateur.

La sécurité occupe une place centrale dès la conception. Le *sandboxing* isole les processus de rendu en limitant drastiquement leurs privilèges et leur accès au système d'exploitation. Un processus compromis ne peut pas infecter le reste du système. Chrome s'appuie sur quatre mécanismes de sécurité Windows pour renforcer cette protection contre les logiciels malveillants.

Le moteur JavaScript V8 marque une avancée technique majeure. Développé par une équipe Google au Danemark, il compile directement en code machine sans passer par un interpréteur. Les classes cachées optimisent le traitement des objets similaires. Le ramasse-miettes est précis tout en demeurant léger, rendant l'exécution de JavaScript bien plus rapide que sur les moteurs concurrents.

L'interface de Chrome frappe par sa simplicité. Les développeurs ont suivi le principe « interface simple, cœur puissant ». Ils ont maximisé l'espace d'affichage des pages en réduisant les éléments d'interface au strict nécessaire. La barre d'adresse et le champ de recherche fusionnent dans l'*omnibox*. Cette approche minimaliste inspirera ensuite de nombreux autres navigateurs.

Google publie simultanément Chromium, qui est la version open source (et la base) de Chrome. La majorité du code source sort sous licence BSD, permettant aux développeurs de l'utiliser librement. Ce choix stratégique favorise l'adoption des technologies Chrome et stimule l'innovation dans tout l'écosystème des navigateurs.

Les performances restent au cœur des préoccupations. En 2015, Chrome introduit un cache de *bytecode* JavaScript. Les scripts fréquemment utilisés

voient leur code compilé conservé sur le disque. Les pages se chargent plus vite, même après un redémarrage du navigateur. Cette optimisation illustre l'attention constante portée à la vitesse d'exécution.

Chrome intègre WebKit comme moteur de rendu, initialement créé par Apple. Mais Google ne retient que le composant WebCore et développe sa propre couche d'adaptation. Cette personnalisation lui donne un contrôle total sur les performances du navigateur.

Le succès arrive vite. Un an après son lancement, en août 2009, Chrome est le troisième navigateur le plus utilisé au monde, dépassant Safari et Opera. En novembre 2010, il franchit la barre des 10% de parts de marché aux États-Unis. La progression s'accélère d'année en année.

En 2020, Microsoft abandonne EdgeHTML et reconstruit Edge sur la base de Chromium. Brave, Opera et bien d'autres navigateurs s'appuient sur cette technologie. Un écosystème riche se développe autour du projet.

Le Chrome Web Store, lancé en 2010, centralise la distribution des extensions. Cette plateforme structurée facilite la découverte et l'installation d'extensions, un atout dans l'adoption de Chrome, enrichissant les fonctionnalités du navigateur.

Les mises à jour automatiques changent la façon de distribuer un navigateur. Chrome s'actualise silencieusement en arrière-plan, sans intervention de l'utilisateur. Cette approche améliore la sécurité en garantissant que chacun dispose des dernières corrections. Plus besoin d'attendre qu'un utilisateur décide de mettre à jour manuellement son navigateur.

Chrome évolue avec le Web. Le navigateur intègre progressivement des technologies avancées comme le CSS Paint API pour personnaliser le rendu graphique, ou les transformations CSS 3D pour des effets visuels sophistiqués. Ces fonctionnalités répondent aux besoins des applications web modernes qui rivalisent désormais avec les applications natives.

L'architecture de Chrome inspire la conception de systèmes d'exploitation. Chrome OS, lancé en 2011, étend les principes du navigateur à l'échelle d'un OS complet. Google y voit l'avenir de l'informatique, un monde centré sur le web où le navigateur est la plateforme principale d'exécution des applications. Ce qui se vérifie avec l'explosion des solutions SaaS en entreprise.

La recherche de vitesse ne s'arrête jamais. Les optimisations touchent le chargement des pages, l'exécution du JavaScript, le rendu des images. Chrome introduit le préchargement des ressources et la compression des données pour réduire encore les temps de chargement. Chaque milliseconde compte.

En 2023, quinze ans après son lancement, Chrome domine le marché des navigateurs web. Son influence sur l'évolution des standards web et

des technologies de navigation est immense. Les innovations techniques de Chrome se sont généralisées. Le web s'est transformé en une véritable plateforme d'applications sophistiquées, et Chrome y a joué un rôle décisif.

HTML5

Au tournant des années 2010, le Web traverse une période où les smartphones se multiplient, les réseaux sociaux explosent, et les développeurs se heurtent à une limitation frustrante : comment écrire du code qui fonctionne partout sans devoir tout réécrire pour chaque plateforme ? HTML5 arrive dans ce contexte, non comme une simple mise à jour technique, mais comme une refonte de l'architecture du Web.

L'histoire commence en 2004, quand Ian Hickson, alors chez Apple, cofonde le WHATWG (Web Hypertext Application Technology Working Group). Ce groupe rassemble des ingénieurs d'Apple, de Mozilla Foundation et d'Opera Software, tous convaincus qu'il faut repenser les fondations du Web. Le W3C finit par intégrer leurs travaux dans une vision plus large d'une plateforme ouverte.

HTML5 ne se résume pas à un nouveau langage de balisage. C'est un ensemble cohérent qui englobe le Document Object Model, les feuilles de style CSS, JavaScript, et toute une série d'interfaces standardisées pour la vidéo, la géolocalisation, le stockage local ou les graphiques. Cette approche globale est une aubaine pour les développeurs.

Matthew McVickar, développeur web chez Ocupop, l'exprime clairement : HTML5 officialise des pratiques que les développeurs utilisaient déjà, mais de manière bricolée. Avec l'interface JavaScript pour la géolocalisation, par exemple, les navigateurs mobiles accèdent au GPS sans passer par des solutions propriétaires. Fini les bidouilles, place aux standards.

CSS3 apporte sa pierre à l'édifice en transformant la gestion de l'apparence des pages. Les développeurs créent maintenant des effets visuels directement dans le navigateur, sans passer par Photoshop ou d'autres outils externes. Le gain de temps est considérable, la flexibilité accrue.

Le stockage local marque une rupture avec les cookies traditionnels. Cette fonction autorise le stockage de volumes importants de données structurées côté client, permettant ainsi aux applications Web de tourner sans connexion internet. Les capacités graphiques s'enrichissent aussi avec le support natif des graphiques vectoriels SVG et l'élément Canvas, qui ouvre la porte aux animations 2D et 3D via JavaScript.

Ian Hickson, passé entre-temps chez Google, insiste sur la rigueur technique apportée par HTML5. Les anciennes versions d'HTML laissaient place à l'interprétation : deux navigateurs pouvaient intégrer différemment les mêmes spécifications tout en prétendant respecter les normes. HTML5 fixe des règles précises qui garantissent une vraie compatibilité.

Mais le déploiement se heurte à des obstacles. La vidéo cristallise les tensions : aucun standard unique pour la compression, le streaming ou la gestion des droits numériques. En 2010, Apple refuse Flash sur l'iPhone et l'iPad, lui préférant sa propre version d'HTML5. Microsoft et Google développent leurs approches, obligeant les développeurs à jongler avec plusieurs versions de leur code.

Hui Zhang, professeur à Carnegie Mellon et cofondateur de Conviva Inc., connaît bien le problème. Pour lui, la vidéo reste l'aspect le plus épineux du contenu en ligne. Faire converger codecs, protocoles de streaming et gestion des droits demande du temps et des compromis entre industriels.

Le W3C prévoit en 2014 de finaliser la spécification HTML5. Philippe Le Hégaret, responsable du domaine interaction, tempère pourtant les attentes : le développement continue. L'organisation reçoit régulièrement des demandes d'ajout de fonctionnalités, comme la conversion voix-texte directement dans le navigateur.

Le WHATWG prend en 2011 un virage radical en abandonnant la numérotation des versions. HTML est un « standard vivant » en développement permanent, une vision qui colle à la réalité d'un Web en évolution constante. Ian Hickson le dit sans détour : le développement d'HTML se poursuivra tant que la technologie sera pertinente.

Cette technologie montre comment la standardisation peut stimuler l'innovation sans sacrifier la compatibilité. Elle illustre aussi l'importance de la collaboration entre acteurs de l'industrie pour faire avancer le Web.

Memcached

En 2003, Brad Fitzpatrick se retrouve face à un casse-tête de taille. LiveJournal, la plateforme de blogs qu'il a créée, explose littéralement : plus de 2,5 millions de comptes et une infrastructure qui commence à montrer ses limites. Quelque 70 machines tournent jour et nuit, mais les bases de données craquent sous la pression des requêtes. Le site affiche une particularité qui complique tout : chaque contenu possède différents niveaux de sécurité et s'insère dans de multiples vues. Impossible de générer des pages

statiques quand les éléments qui les composent vivent chacun à leur rythme.

Fitzpatrick observe alors une réalité technique qui va orienter toute sa démarche. Les processeurs gagnent en vitesse, année après année, tandis que les disques durs traînent la patte. Pourquoi ne pas exploiter cette puissance de calcul plutôt que de s'épuiser en accès disques interminables ? L'idée germe d'utiliser la RAM qui sommeille dans les serveurs web, cette mémoire vive qui ne demande qu'à servir.

Un premier prototype voit le jour en Perl. L'essai se révèle décevant : trop lent, trop gourmand en mémoire. Fitzpatrick reprend tout en C et bâtit un démon mono-processus, mono-thread, qui s'appuie sur des entrées-sorties asynchrones. Pour garantir que son système fonctionne partout, il intègre libevent, une bibliothèque qui choisit automatiquement la meilleure stratégie de gestion des descripteurs de fichiers selon l'environnement d'exécution.

Le principe de Memcached tient en quelques mots : transformer la mémoire disponible des serveurs en un vaste réservoir partagé. Une table de hachage distribuée où les données circulent entre les machines grâce à un système de hachage cohérent. Rien de superflu dans l'architecture : le serveur ne bavarde pas avec ses congénères, ne conserve rien sur disque, propose trois opérations basiques – set, get, delete – et c'est tout. Cette austérité voulue procure des performances redoutables, avec des algorithmes en O(1) qui répondent du tac au tac. L'allocateur de mémoire par tranches évite la fragmentation qui pourrissait la vie des premières versions utilisant malloc.

Facebook adopte Memcached en 2007 et développe `mcrouter`, une variante enrichie de fonctions de *sharding*. Twitter suit avec `twemcache`, taillé pour ses besoins propres. Ces géants du web valident l'approche de Fitzpatrick et font de Memcached un standard de fait.

Les chiffres donnent le vertige. Chez Facebook, le système traite des milliards de requêtes par seconde et stocke des milliers de milliards d'objets. Le taux de réussite du cache atteint régulièrement 92% : seule une poignée de requêtes doit vraiment interroger la base de données. La charge s'allège d'autant. Le système introduit les « leases » pour gérer les conflits d'écriture et éviter que des hordes de clients tentent tous en même temps de régénérer une entrée expirée.

L'architecture distribuée autorise une souplesse bienvenue. On ajoute ou retire des serveurs du pool à la volée, le système s'ajuste sans broncher. Cette capacité de montée en charge horizontale devient vite une pratique courante dans la conception des infrastructures web modernes.

La communauté s'empare du projet. Des bibliothèques clientes fleurissent pour PHP, Python, Ruby, Java, C#, et ajoutent sérialisation d'ob-

jets et compression transparente, rendant l'outil plus maniable dans des contextes variés. Le code source continue d'évoluer grâce aux contributions d'une communauté active qui ne faiblit pas.

En 2024, Memcached tient toujours son rang dans les infrastructures web. Le projet conserve sa philosophie initiale : rester simple, rapide, fiable. Cette approche minimaliste tranche avec des alternatives plus baroques comme Redis, mais répond parfaitement aux besoins de mise en cache pure. Memcached démontre qu'une solution élégante et dépouillée résout parfois mieux les problèmes complexes qu'une machinerie sophistiquée.

OpenStack

À la fin des années 2000, les ressources matérielles et logicielles basculaient progressivement vers le *cloud computing*, promettant sécurité accrue et gestion centralisée. En 2009, la NASA cherchait une plateforme pour ses services Web internes. Le marché proposait uniquement des solutions propriétaires, verrouillées par quelques géants du secteur.

Les développeurs de l'agence spatiale américaine pensaient qu'une alternative libre servirait mieux une communauté large : chercheurs universitaires, institutions gouvernementales, entreprises diverses. Rien de tel n'existait. Ils ont donc décidé de créer cette solution eux-mêmes. L'initiative a attiré Rackspace, dont les ingénieurs travaillaient sur un projet similaire pour soutenir leur activité de vente de matériel et de services.

Début 2010, les représentants de Rackspace ont contacté la NASA. Les deux équipes se sont rencontrées dans un restaurant thaïlandais, et cette collaboration modeste a donné naissance à OpenStack. Le lancement officiel a eu lieu en juillet 2010, avec l'ambition d'aider les organisations à déployer des services cloud sur du matériel standard. Quatre mois plus tard, la première version, baptisée Austin, voyait le jour. Elle apportait calcul en cloud et stockage d'objets pour l'Infrastructure as a Service.

Les projets initiaux permettaient de contrôler des pools de ressources : calcul, stockage, réseau, le tout via un tableau de bord centralisé dans les centres de données. Les administrateurs disposaient d'un contrôle fin, tandis que les utilisateurs obtenaient leurs ressources instantanément. La communauté s'est alors engagée sur un rythme de nouvelles versions tous les six mois.

La Fondation OpenStack a vu le jour en 2012. Cette organisation indépendante à but non lucratif avait pour mission de promouvoir le déve-

loppement et l'adoption du logiciel. La structure garantissait une gestion neutre de la propriété intellectuelle et le partage des ressources, créant des conditions équitables pour tous les participants.

Le modèle de gouvernance reposait sur trois piliers : un conseil d'administration fixait vision et gouvernance, un comité technique gérait versions et besoins inter-projets, un comité d'utilisateurs remontait les retours terrain et défendait leurs intérêts. La communauté fonctionnait selon quatre principes : code source ouvert sous licence Apache 2.0, communauté ouverte produisant un écosystème actif, développement transparent avec revues de code publiques, conception ouverte avec des sommets semestriels accessibles à tous.

Les contributeurs venaient d'horizons variés. Ingénieurs logiciels participant au code, rédacteurs fournissant documentation et traductions, développeurs d'applications créant des solutions sur l'infrastructure, opérateurs et utilisateurs déployant et gérant leurs clouds. Des groupes de travail rassemblaient les exigences de segments particuliers comme les télécommunications ou le monde de l'entreprise. L'écosystème commercial regroupait fournisseurs cloud et prestataires ajoutant leur valeur au logiciel de base.

L'adoption s'est accélérée dans différents secteurs. BMW, Disney et Walmart ont prouvé la viabilité de la solution en production. PayPal l'a adoptée pour l'agilité et la disponibilité qu'exigeaient ses produits phares. Time Warner Cable a misé sur cette flexibilité pour gérer d'importants volumes de données dans ses centres.

La technologie s'est étendue bien au-delà des usages prévus au départ. Le CERN l'a employée pour analyser les données de son Grand collisionneur de hadrons, gérant plus de 115 000 cœurs de calcul. Adobe Digital Marketing s'en est servi pour transformer son environnement de virtualisation en informatique en libre-service. L'université australienne NeCTAR l'a adoptée pour sa recherche collaborative nationale, supportant plus de 6 000 chercheurs.

En 2024, Rackspace a réaffirmé son engagement avec OpenStack Enterprise, solution cloud entièrement gérée pour charges de travail critiques. La société avait contribué plus de 5,6 millions de lignes de code et figurait parmi les plus grands fournisseurs cloud OpenStack mondiaux. Son expertise comprenait plus d'un milliard d'heures-serveur d'expérience en production.

L'impact sur l'industrie informatique a été considérable. La plateforme a permis aux entreprises de toutes tailles de déployer leurs propres infrastructures cloud, échappant ainsi à la domination des services propriétaires. Sa nature libre a favorisé l'innovation collaborative et réduit les coûts d'infrastructure. Les API ouvertes et le support de pratiquement tous les grands

fournisseurs informatiques donnaient aux utilisateurs la liberté de déplacer leurs charges de travail entre clouds privés et publics selon leurs besoins.

Les bénéfices incluaient une accélération de la mise sur le marché grâce au portail libre-service, une plateforme basée sur des API pour développer des applications natives du cloud, et des temps de provisionnement drastiquement réduits. L'interopérabilité et la capacité de cloud hybride permettaient aux entreprises de choisir le meilleur environnement pour leurs applications sans dépendre d'un fournisseur unique.

L'évolution s'est poursuivie avec l'intégration de nouvelles technologies comme les conteneurs, les machines virtuelles et les serveurs *bare metal*. La communauté mondiale compte plus de 56 000 membres individuels dans plus de 180 pays et plus de 600 entreprises. Cette croissance témoigne de la pertinence continue de la plateforme dans un paysage technologique en constante mutation, où la demande de solutions cloud flexibles et ouvertes ne cesse de croître.

Bitcoin

En octobre 2008, alors que la crise financière secoue le monde entier et que la confiance envers les banques s'effondre, un mystérieux document circule sur internet. Son auteur, Satoshi Nakamoto, dont on ne saura jamais s'il s'agit d'une personne seule ou d'un groupe, propose ni plus ni moins qu'une révolution : créer une monnaie numérique qui échappe au contrôle des institutions financières. Le titre du document, « Bitcoin : A Peer-to-Peer Electronic Cash System », annonce clairement l'ambition. Il s'agit de bâtir un système de paiement où les transactions s'effectuent directement entre utilisateurs, sans intermédiaire.

Trois mois plus tard, le 3 janvier 2009, Nakamoto mine le premier bloc de cette chaîne qu'on appellera blockchain. Ce bloc genèse contient un message qui en dit long sur les motivations du projet : « The Times 03/Jan/2009 Chancellor on brink of second bailout for banks ». Cette référence au quotidien britannique n'est pas anodine. Elle rappelle que les banques s'apprêtent à recevoir un énième plan de sauvetage, financé par l'argent public. Le Bitcoin naît donc d'une contestation, d'un refus de ce système bancaire jugé défaillant.

Les débuts restent modestes. Le premier bloc suivant n'apparaît que cinq jours après, le 8 janvier. Ce délai inhabituel intrigue encore aujourd'hui. Certains y voient une période de tests, d'autres une référence biblique aux

six jours de la création. Quoi qu'il en soit, le réseau démarre lentement. À l'époque, n'importe qui peut miner des bitcoins avec un ordinateur domestique. La difficulté est fixée au minimum, à 1, un niveau dérisoire comparé aux standards actuels qui nécessitent des installations industrielles.

Actif jusqu'en décembre 2010, Nakamoto disparaît sans laisser de traces. Son départ volontaire laisse la communauté des développeurs poursuivre seule. Cette absence alimente les spéculations sur son identité, mais garantit aussi que personne ne contrôle le projet. Le Bitcoin est véritablement un bien commun, sans chef ni propriétaire.

Des plateformes d'échange voient le jour en 2011, permettant d'acheter et de vendre des bitcoins contre des monnaies classiques. Mt. Gox s'impose et gère bientôt 70% des transactions mondiales. Mais en 2014, le piratage de cette plateforme japonaise fait perdre 850 000 bitcoins. Le choc ébranle l'écosystème et rappelle brutalement les risques liés à ces nouveaux actifs.

Le protocole Bitcoin repose sur des mécanismes ingénieux. La preuve de travail oblige les mineurs à résoudre des calculs complexes pour valider les transactions. En contrepartie, ils reçoivent une récompense en bitcoins. Cette récompense, initialement de 50 unités par bloc, diminue de moitié tous les quatre ans environ. Ce processus, baptisé « halving », limite l'émission totale à 21 millions de bitcoins. Cette rareté programmée rappelle celle de l'or et participe à la valorisation de la cryptomonnaie.

Progressivement, des entreprises acceptent le Bitcoin comme moyen de paiement. En 2012, quelques pionniers se lancent. Des acteurs majeurs comme Microsoft, Dell ou PayPal rejoignent le mouvement. Cette adoption confère une légitimité nouvelle à ce qui n'était au départ qu'une expérimentation cryptographique.

Le cours du Bitcoin traverse des cycles spectaculaires. En 2013, il grimpe de moins de 20 dollars à plus de 1 000 dollars, avant de retomber. En 2017, nouvelle envolée jusqu'à près de 20 000 dollars, suivie d'un effondrement l'année suivante. Ces variations extrêmes trahissent la dimension spéculative de l'actif. Personne ne s'accorde vraiment sur sa valeur réelle, si tant est qu'elle existe.

Le succès du Bitcoin inspire d'autres projets. Des centaines de cryptomonnaies alternatives, les « altcoins », émergent avec des propositions techniques variées. La blockchain trouve des applications insoupçonnées : contrats intelligents, finance décentralisée, traçabilité de produits. Cette technologie, conçue pour une monnaie numérique, s'étend bien au-delà de son usage initial.

Certains États réagissent en interdisant purement et simplement le Bitcoin, d'autres tentant de l'encadrer. Les préoccupations portent sur le blanchiment d'argent, la protection des investisseurs, l'évasion fiscale. L'impact

environnemental du minage, gourmand en électricité, soulève aussi des critiques croissantes. Le débat est toujours vif entre les défenseurs d'une liberté financière absolue et ceux qui réclament une régulation stricte.

Plus de quinze ans après sa création, le Bitcoin s'est imposé comme une classe d'actifs à part entière, reconnue par les investisseurs institutionnels. Comme quoi une idée radicale, portée par quelques lignes de code, peut bousculer les certitudes d'un système financier millénaire. La technologie continue d'évoluer : le Lightning Network tente de résoudre les problèmes de lenteur des transactions, tandis que de nouvelles améliorations sont régulièrement proposées.

Cette expérience inédite a ouvert un champ immense dans les domaines de la cryptographie et des systèmes distribués. Le Bitcoin est un objet de fascination, symbole d'une contestation qui refuse les intermédiaires, les autorités centrales, les contrôles. Finalement, qu'est-ce que la monnaie ? Qui doit la contrôler ? Comment organiser la confiance dans un monde numérique ? Ces interrogations continuent d'alimenter les réflexions sur l'avenir de nos sociétés connectées.

CouchDB

En 2005, Damien Katz quitte IBM avec une idée en tête. Ses années passées sur Lotus Notes lui ont appris ce qu'était une base de données documentaire, mais aussi ce qu'elle pourrait devenir. À cette époque, le monde informatique tourne presque exclusivement autour des bases relationnelles. Pourtant, quelque chose cloche. Les applications web modernes et les outils collaboratifs réclament autre chose, une souplesse que SQL ne sait pas vraiment offrir.

Il commence par écrire CouchDB en C++. Les premiers prototypes fonctionnent, mais la gestion de la concurrence le fait buter. Il cherche une solution, parcourt les forums techniques, et tombe sur le blog « Lambda the Ultimate » où quelqu'un parle d'Erlang. Ce langage, conçu à l'origine pour les télécoms chez Ericsson, possède exactement ce dont il a besoin. La tolérance aux pannes, la distribution native, le traitement concurrent : tout y est. Katz prend alors une décision radicale. Il jette son code C++ et recommence tout en Erlang.

Ce choix technique modèle CouchDB. Erlang apporte avec lui une philosophie : les systèmes doivent survivre aux défaillances, les processus peuvent tomber sans faire chuter l'ensemble, les données peuvent être distribuées naturellement. Ces propriétés, pensées pour des centraux téléphoniques devant

fonctionner sans interruption, collent parfaitement aux besoins d'une base de données moderne. Le parallèle entre télécoms et stockage de données n'était pas évident au premier regard, mais il s'avère pertinent.

CouchDB rompt avec plusieurs conventions établies. Pas de schéma rigide : les documents JSON peuvent évoluer librement, se transformer au gré des besoins sans nécessiter de migration de base. Cette liberté répond aux réalités du développement web où les structures changent constamment. L'interface parle HTTP, utilise GET pour lire, PUT pour écrire, DELETE pour supprimer. Rien de plus, rien de moins. Un développeur web se retrouve en terrain connu.

JavaScript devient le langage des requêtes. Là où d'autres bases imposent leur propre dialecte, CouchDB mise sur un langage que tout développeur web connaît. Les vues, ces index personnalisés qui permettent d'interroger les données, s'écrivent en JavaScript. Map-reduce aussi.

La réplication bidirectionnelle figure parmi les atouts les plus singuliers de CouchDB. Deux instances peuvent se synchroniser, échanger leurs modifications après avoir fonctionné séparément pendant des jours. Par exemple, un smartphone perd sa connexion, continue de travailler en local, puis synchronise tout une fois reconnecté. Peu de temps après la sortie de l'iPhone par Apple, cette capacité anticipe un monde où les appareils mobiles et les connexions intermittentes seront la norme plutôt que l'exception.

Pour gérer les écritures concurrentes, CouchDB adopte une approche optimiste. Pas de verrous qui bloquent, pas d'attentes forcées. Chaque document possède un numéro de révision, un peu comme Git avec ses commits. Quand deux modifications entrent en conflit, CouchDB garde les deux versions et laisse l'application décider. Cette gestion des conflits, inspirée des systèmes de contrôle de version, donne une souplesse inhabituelle pour une base de données.

L'histoire bifurque en 2011. CouchDB fusionne avec Membase, une technologie de cache distribué aux performances remarquables. De cette union naît Couchbase, qui combine la richesse documentaire de CouchDB avec la vitesse brute de Membase. La créature hybride hérite des deux mondes : persistance et flexibilité d'un côté, vélocité de l'autre.

Techniquement, CouchDB repose sur quelques briques bien pensées. Le moteur de stockage écrit sur disque tout en maintenant des index pour l'accès rapide. Le système de vues transforme les documents en index interrogeables. Le protocole de réplication assure la cohérence entre instances distantes. Ces composants interagissent en s'appuyant sur les forces d'Erlang, qui orchestre le tout avec une certaine élégance.

CouchDB participe à l'émergence du mouvement NoSQL, et prouve qu'on peut penser autrement le stockage des données. Son influence se

lit dans l'architecture de nombreuses bases modernes qui reprennent ses idées : documents sans schéma, API web-friendly, réplication intelligente. PouchDB, son dérivé en JavaScript pour navigateurs, étend la synchronisation jusqu'au client web.

Les cas d'usage se multiplient là où la flexibilité compte. Applications mobiles qui doivent fonctionner hors ligne, systèmes collaboratifs où des utilisateurs modifient les mêmes données, outils qui doivent se répliquer entre sites distants. CouchDB excelle dans ces situations où la connexion n'est pas garantie, où les structures évoluent vite, où la distribution des données est une nécessité.

Node.js

En 2009, Ryan Dahl présente Node.js lors d'une conférence européenne sur le JavaScript. Cette technologie allait changer la donne en permettant d'utiliser JavaScript en dehors des navigateurs web. Jusqu'alors, ce langage restait cantonné au front-end, depuis sa création par Brendan Eich en 1995 pour Netscape. À l'origine, JavaScript servait surtout à animer les pages web et manipuler le DOM. Netscape avait bien tenté une version serveur avec Netscape Enterprise Server, mais l'essai s'était soldé par un échec.

Le nom de JavaScript témoigne d'une stratégie marketing : surfer sur la vague de popularité de Java. Pourtant, les deux langages n'ont rien à voir. JavaScript doit bien plus à Scheme et Self qu'à Java, dont il reprend juste quelques éléments de syntaxe. Pendant des années, les programmeurs sérieux regardaient JavaScript de haut, le jugeant trop lent et inadapté aux vraies applications. Cette réputation a volé en éclats avec l'arrivée de moteurs plus puissants, notamment le V8 de Google Chrome. Grâce à la compilation à la volée, l'insertion de code et l'optimisation dynamique, JavaScript a pu rivaliser avec C++ sur certains benchmarks et dépasser Python dans la plupart des cas.

Node.js s'appuie sur le moteur V8 pour bâtir une plateforme serveur robuste. L'architecture tranche avec les approches traditionnelles. Là où Apache crée un nouveau thread ou processus à chaque connexion, Node.js mise sur une boucle événementielle mono-thread. Cette stratégie évite les coûts de création et gestion des threads, qui pèsent lourd quand le serveur croule sous les requêtes.

La gestion des entrées-sorties fait la différence. Au lieu de bloquer l'exécution en attendant qu'une opération se termine, Node.js utilise des callbacks qui traitent les résultats de façon asynchrone. Le serveur peut ainsi

jongler avec plusieurs requêtes pendant qu'une opération tourne en arrière-plan. Les ressources sont mieux exploitées, les temps morts disparaissent.

La communauté s'est vite emparée du projet. Dès 2010, Isaac Schlueter lance NPM (Node Package Manager), véritable la pierre angulaire de l'écosystème. Les développeurs peuvent partager et réutiliser du code sans friction, ce qui accélère l'adoption. L'année suivante, une version Windows native voit le jour et élargit considérablement l'audience.

Joyent, société de logiciels et de services basée à San Francisco, pilote le développement pendant quelques années avant que le projet ne rejoigne la Fondation Node.js en 2015. Pendant cette transition vers une gouvernance plus ouverte, sous l'égide d'un comité technique de pilotage, Node.js devient un projet collaboratif de la Linux Foundation. En 2019, la fusion avec la Fondation JS donne naissance à la Fondation OpenJS, qui coordonne mieux l'écosystème JavaScript.

Le registre NPM héberge aujourd'hui plus de deux millions de paquets, ce qui en fait le plus gros référentiel logiciel au monde. On y trouve des frameworks comme Express, des outils de développement comme browserify et gulp, et plus de 8 000 clients de bases de données, de redis à mongoose.

La boucle événementielle mono-thread pourrait sembler un frein sur les processeurs multicœurs modernes. Node.js contourne l'obstacle avec le module cluster, qui crée des processus workers partageant les ports du serveur parent. Un algorithme round-robin sensible à la charge distribue les requêtes entre les cœurs disponibles.

Le succès de Node.js a popularisé la programmation asynchrone basée sur les événements. Cette approche convient bien aux applications gourmandes en concurrence, comme la messagerie instantanée ou les services temps réel. La plateforme a aussi favorisé le JavaScript isomorphe, où le même code tourne côté client et serveur.

Node.js continue d'évoluer. Le support des modules ECMAScript, l'amélioration des performances et le renforcement de la sécurité figurent parmi les priorités actuelles. La plateforme reste pertinente pour les microservices, les applications serverless, les outils de développement et les applications de bureau via Electron. En proposant une autre façon de faire du serveur, Ryan Dahl a redéfini la construction d'applications web performantes et scalables.

Raspberry Pi

En 2006, Eben Upton et ses collègues du laboratoire d'informatique de Cambridge se heurtent à un problème inattendu : les candidats qui postulent dans leur cursus maîtrisent de moins en moins bien l'informatique. Pire, leur nombre diminue d'année en année. Le diagnostic ne tarde pas à s'imposer. Les ordinateurs des années 1980, avec leurs entrailles visibles et leur programmation directe, ont disparu au profit de machines hermétiques, soigneusement dissimulées derrière des coques en plastique et des interfaces léchées. L'ordinateur n'invite plus à bricoler, il se contente d'être utilisé.

Upton rêve d'une machine bon marché qui rendrait aux jeunes ce plaisir de l'expérimentation, celui qu'il a connu à leurs âges. Son prototype initial, baptisé ABC Micro en hommage au BBC Micro, se limite à une carte bricolée autour d'un microcontrôleur Atmel et de quelques puces de mémoire. Les premiers tests auprès d'enfants révèlent toutefois que cette approche minimaliste ne déclenche pas l'étincelle attendue.

Toujours en 2006, Upton rejoint Broadcom. Cette opportunité lui ouvre l'accès à des composants autrement plus puissants, notamment le BCM2835 qui intègre un processeur ARM11. Cette puce concentre sur quelques centimètres carrés tout ce qu'il faut pour faire fonctionner un vrai ordinateur : sortie HDMI, accélération 3D, décodage vidéo, contrôleur USB. L'ajout d'un processeur ARM généraliste change la donne, notamment en y faisant tourner Linux.

La fondation Raspberry Pi voit le jour en 2009. Deux ans plus tard, la carte définitive prend forme. Upton et son équipe imaginent une production confidentielle, quelques centaines d'exemplaires par an tout au plus. Mais l'engouement dépasse leurs prévisions les plus optimistes. La fondation réoriente sa stratégie : elle conserve la conception, la marque et les prototypes, mais délègue la fabrication à RS Components et Premier Farnell.

Début 2012, les 10 000 premiers Raspberry Pi partent en quelques heures. Le nom circule dans les cercles techniques et les communautés maker. La fondation saisit cette dynamique pour relancer l'industrie électronique britannique : après un démarrage en Chine, la production migre progressivement au Pays de Galles jusqu'à atteindre une localisation complète.

Les utilisateurs détournent aussitôt la carte de sa vocation pédagogique initiale. Les 32 broches d'entrée-sortie, qu'Upton avait ajoutées presque par acquit de conscience, sont l'élément le plus prisé. On voit apparaître des drones pilotés par Raspberry Pi, des ballons stratosphériques, des installations domotiques. Un bricoleur transforme son four à micro-ondes en y greffant un écran tactile, des commandes vocales, une interface web et un lecteur de codes-barres connecté à une base de recettes.

Ces réalisations sophistiquées, qui demandent des compétences pointues, renforcent paradoxalement la dimension éducative du projet. Elles montrent aux jeunes ce qu'on peut accomplir quand on sait programmer. Les ateliers en milieu scolaire confirment cette intuition : il suffit parfois de modifier la couleur d'un serpent dans un jeu pour qu'un élève refuse d'arrêter. Cette première sensation de contrôle sur la machine produit un effet comparable à ce qu'Upton avait vécu dans sa jeunesse.

Le Raspberry Pi combine des caractéristiques techniques et économiques qui expliquent son succès. Son processeur ARM1176JZF-S à 700 MHz, couplé à un GPU VideoCore IV, délivre des performances multimédia étonnantes pour un appareil de cette taille. La mémoire SDRAM, passée de 256 Mo sur le modèle A à 512 Mo sur le modèle B, suffit à faire tourner confortablement Linux. L'absence de disque dur, remplacé par une simple carte SD, maintient le prix bas et simplifie le changement de système d'exploitation.

Les mises à jour matérielles se succèdent sans rupture. Le Raspberry Pi 4 embarque un Broadcom BCM2711 quadricœur ARM Cortex-A72 64 bits à 1,5 GHz, jusqu'à 8 Go de mémoire LPDDR4, le Bluetooth 5.0, le Wi-Fi bibande et des ports USB 3.0. Ces évolutions préservent la compatibilité avec les versions antérieures tout en élargissant le champ des possibles.

Le Raspberry Pi a prouvé qu'il existait un marché pour les ordinateurs monocartes accessibles. Avec 19 millions d'unités vendues, il se classe au troisième rang des ordinateurs généralistes. Il a ranimé l'intérêt pour l'apprentissage pratique de l'informatique et inspiré toute une génération d'ingénieurs et de créateurs. Son histoire montre comment une initiative éducative locale peut, en touchant juste, déclencher un mouvement international qui redéfinit notre rapport à la technologie.

X

2010

L'ère de la connexion permanente et de la donnée massive

La décennie 2010-2020 a marqué une cassure dans notre relation au numérique. Plus qu'un simple tournant technologique, ces années ont vu naître une société où le digital s'est mêlé au réel jusqu'à en brouiller les frontières. Cette mondialisation a pris un visage concret dans nos vies quotidiennes. Les populations, majoritairement urbaines et mobiles, réclamaient un accès instantané à l'information et aux services en ligne. Cette soif d'immédiateté a façonné les innovations de toute une époque.

Les premiers temps de cette décennie ont révélé la puissance insoupçonnée des réseaux sociaux, avec Facebook et Twitter en fer de lance de véritables révolutions. Le printemps arabe de 2011 a montré comment ces plateformes pouvaient contourner les médias traditionnels contrôlés par les régimes autoritaires. Un simple smartphone suffisait désormais à diffuser des images de manifestations, à coordonner des rassemblements, à faire entendre une voix jusque-là étouffée. Ce bond en avant médiatique a changé notre perception du rôle des technologies dans les mouvements sociaux.

La généralisation des *smartphones*, couplée au déploiement des réseaux 4G, a transformé chaque citoyen en créateur de contenu. Nous avons commencé à produire et consommer des données en quantités astronomiques. Cette avalanche d'informations a engendré ce phénomène qu'on a nommé

« Big Data ». Nos infrastructures traditionnelles craquaient sous le poids de ces flux massifs. Il a fallu imaginer de nouvelles architectures, de nouvelles méthodes d'analyse pour donner du sens à ce déluge informationnel.

Au cœur de cette transformation, quelques entreprises américaines ont bâti des empires d'une envergure inédite. Google, Apple, Facebook, Amazon et Microsoft ont renforcé leur mainmise sur l'écosystème numérique mondial. Leur stratégie repose sur le rachat méthodique de toute startup innovante susceptible de menacer leur domination. Cette concentration du pouvoir numérique a fait naître des inquiétudes légitimes concernant la vie privée des utilisateurs et le contrôle de l'information. Qui détient nos données détient un pouvoir considérable sur nos vies.

Face à cette hégémonie occidentale, la Chine a développé son propre univers numérique. Des acteurs comme Alibaba, Tencent et Baidu ont créé un écosystème parallèle, sous la supervision étroite du gouvernement chinois et du Parti Communiste Chinois. Pékin a mis sur pied une stratégie nationale ambitieuse visant à dominer les technologies d'avenir, notamment l'intelligence artificielle et les réseaux 5G. Cette montée en puissance a transformé l'innovation technologique en véritable enjeu géopolitique. La rivalité sino-américaine s'est déplacée des océans vers les laboratoires et les centres de données.

L'année 2013 a marqué un séisme dans notre rapport à la sécurité numérique. Edward Snowden a dévoilé l'ampleur stupéfiante de la surveillance exercée par les services de renseignement américains. Le monde a découvert que nos communications étaient systématiquement interceptées et analysées. Ce choc a provoqué une prise de conscience globale sur la fragilité de notre vie privée à l'ère numérique. L'Europe a réagi en élaborant le Règlement Général sur la Protection des Données, devenu depuis une référence mondiale. La cybersécurité, jusque-là préoccupation technique marginale, s'est imposée comme priorité stratégique pour les États et les organisations.

Les réseaux sociaux ont progressivement colonisé notre espace informationnel. Leur influence sur la formation de l'opinion publique s'est révélée lors de l'élection présidentielle américaine de 2016 avec le scandale Cambridge Analytica qui a révélé comment Facebook avait permis l'exploitation des données de millions d'utilisateurs à des fins politiques. Ces plateformes ont montré leur vulnérabilité face aux tentatives de manipulation. Plus inquiétant encore, leurs algorithmes de recommandation nous enferment dans des « bulles de filtres », des univers informationnels confortables mais déformés. Cette fragmentation du débat public questionne le fonctionnement de nos démocraties à l'ère numérique.

Le cloud computing a transformé les infrastructures informatiques des entreprises et des particuliers. Pourquoi investir dans des serveurs coûteux

et sous-exploités quand on peut louer de la puissance de calcul à la demande ? Cette flexibilité a accéléré la transformation numérique des organisations traditionnelles. Mais cette migration vers le cloud a aussi créé de nouvelles dépendances. La question de la souveraineté numérique s'est à nouveau posée avec acuité, surtout en Europe où la domination des fournisseurs américains suscite des inquiétudes légitimes.

Notre quotidien s'est peuplé d'objets connectés. Montres intelligentes, enceintes à commande vocale, thermostats automatiques, l'Internet des Objets a tissé un maillage invisible mais omniprésent autour de nous. Ces appareils captent en permanence des informations sur nos habitudes, nos déplacements, nos préférences. Ils ont introduit de nouvelles façons d'interagir avec les machines. La voix et le geste remplacent progressivement le clavier et la souris. Cette collecte constante de données personnelles alimente des systèmes d'intelligence artificielle toujours plus sophistiqués.

Bitcoin, né dans l'ombre de la crise financière de 2008, a véritablement émergé durant cette décennie. Cette monnaie numérique, indépendante des banques centrales, a remis en cause nos systèmes financiers. Les technologies financières ont transformé des secteurs entiers : paiements mobiles, prêts entre particuliers, gestion automatisée des investissements. Les institutions traditionnelles ont dû s'adapter face à ces nouveaux concurrents agiles. Les régulateurs, souvent pris de court par l'innovation galopante, ont oscillé entre méfiance et fascination pour ces technologies qui promettaient de démocratiser l'accès aux services financiers.

Les progrès fulgurants de l'intelligence artificielle et de la robotique ont accéléré l'automatisation de nombreuses tâches. Des métiers qu'on croyait réservés aux humains se sont retrouvés menacés par les machines apprenantes. Cette évolution a soulevé des questions sur l'avenir du travail, la préparation des jeunes générations à un monde où les compétences seront obsolètes en peu de temps, et une transition juste pour ceux dont les emplois disparaissent. Ces interrogations ont pris une dimension politique croissante, alors que les inégalités numériques se creusaient entre les différentes catégories de population.

L'économie des plateformes a bouleversé des secteurs entiers. Uber, Airbnb, Deliveroo, ces noms sont devenus emblématiques d'un nouveau modèle économique fondé sur la mise en relation directe entre offre et demande. Cette transformation a créé des formes d'emploi hybrides, ni salariat ni entrepreneuriat classique. Si ces plateformes ont ouvert de nouvelles opportunités et fluidifié les échanges, elles ont aussi généré des tensions sociales considérables. Précarisation du travail, contournement des réglementations, évasion fiscale, les critiques se sont multipliées, forçant les législateurs à repenser des cadres juridiques dépassés par l'innovation.

La conscience de l'impact environnemental du numérique s'est progressivement imposée. L'empreinte carbone des centres de données, la consommation énergétique du minage de Bitcoin, l'obsolescence programmée des appareils électroniques sont des réalités qui ont commencé à influencer les choix technologiques des organisations soucieuses de leur responsabilité écologique. Des initiatives pour un numérique plus sobre ont émergé, cherchant à concilier innovation et durabilité. Cette réflexion sur l'éthique environnementale des technologies numériques deviendra centrale dans les années suivantes.

La fin de la décennie a connu une accélération brutale et inattendue. La pandémie mondiale de 2020 a précipité la transformation numérique de secteurs entiers. Du jour au lendemain, le télétravail s'est généralisé, l'enseignement s'est transporté en ligne, le commerce électronique est devenu vital. Cette crise sanitaire a servi de test grandeur nature pour nos infrastructures numériques, révélant à la fois leur résilience et leurs limites. Elle a aussi mis en lumière l'urgence de réduire la fracture numérique. L'accès à une connexion de qualité et aux compétences digitales est apparu comme une condition essentielle de participation à la vie sociale et économique.

Au terme de ces dix années, le numérique n'apparaît plus comme un secteur distinct mais comme le tissu de notre organisation sociale. Les innovations de cette période ont répondu à notre soif de connexion et d'immédiateté. Mais elles ont aussi engendré de nouveaux problèmes éthiques, sociaux et environnementaux qui continueront de façonner l'évolution de l'informatique. Les années 2010 resteront comme la période où notre monde a basculé définitivement dans l'ère numérique, avec ses promesses éblouissantes et ses zones d'ombre inquiétantes. L'informatique façonne désormais notre perception du monde et nos interactions sociales.

Elasticsearch

L'histoire d'Elasticsearch commence par une histoire d'amour. En 2004, Shay Banon, jeune développeur, déménage à Londres pour accompagner son épouse qui rêve de devenir chef cuisinière. Sans emploi, il décide de mettre ce temps à profit en explorant les technologies émergentes. Son projet : créer une application où sa femme pourrait stocker et retrouver toutes les connaissances culinaires acquises durant sa formation.

Au cœur de cette application, Banon imagine une simple boîte de recherche. Il suffirait d'y saisir un concept, une réflexion ou un ingrédient pour explorer aussitôt les possibilités gastronomiques. Cette vision le conduit vers

Apache Lucene, la bibliothèque de référence en matière de recherche pour Java. Son immersion dans Lucene donne naissance à Compass, un projet visant à simplifier l'utilisation de Lucene dans les applications Java conventionnelles. Conceptuellement, Compass se présente comme un « Hibernate pour Lucene », facilitant le mappage du modèle de domaine vers l'index de recherche.

Le projet captive Banon, qui finit par y consacrer plus de temps qu'à l'application culinaire initiale. Quelques mois après sa création, il ouvre le code source de Compass. Le succès est immédiat. Les développeurs apprécient la simplicité avec laquelle ils peuvent désormais indexer leurs modèles de domaine dans Lucene et effectuer des recherches. Cette liberté leur donne accès à des usages inattendus et novateurs.

En 2010, fort de cette expérience et des retours des utilisateurs, Banon lance Elasticsearch. Cette nouvelle création conserve la philosophie de simplicité de Compass tout en répondant aux besoins modernes de distribution et d'évolutivité. Il quitte son poste chez Gigaspaces pour se consacrer entièrement au développement d'Elasticsearch pendant près de deux ans.

La communauté open source s'approprie le projet. Les développeurs enrichissent les fonctionnalités, créent une console d'administration, développent des outils d'analyse graphique. Ces contributions collectives génèrent un élan considérable. En 2012, Elasticsearch attire l'attention des directeurs techniques de nombreuses entreprises qui vantent ses qualités. Avec 200 000 téléchargements mensuels, un chiffre sans précédent pour un logiciel d'entreprise, Elasticsearch s'impose comme une solution incontournable.

L'équipe fondatrice se constitue autour de Banon avec Steven Schuurman, Uri Boness et Simon Willnauer, tous experts reconnus dans le domaine de la recherche. Ensemble, ils créent une entreprise pour soutenir le développement du projet. La société attire les meilleurs talents de la communauté Lucene et devient le lieu de prédilection pour l'open source dans le domaine de la recherche.

L'architecture d'Elasticsearch repose sur une approche novatrice. Le système utilise JSON et une interface RESTful, ce qui le rend accessible depuis n'importe quel langage de programmation. Sa nature distribuée lui donne la capacité de traiter des volumes de données importants en répartissant automatiquement la charge sur plusieurs nœuds. La flexibilité du schéma autorise le stockage de documents complexes sans définition préalable de structure.

Au fil des années, l'écosystème s'enrichit. Logstash vient faciliter l'ingestion des données, tandis que Kibana offre des capacités de visualisation avancées. Ces outils forment la pile ELK (Elasticsearch, Logstash, Kibana),

devenue une référence pour l'analyse de logs et la surveillance d'infrastructures. Dans les années 2020, des alternatives plus légères comme Fluentd ou FluentBit remplacent souvent Logstash pour l'ingestion des données.

Les cas d'utilisation d'Elasticsearch se multiplient : recherche sur les sites web, analyse des interactions utilisateurs, détection d'anomalies dans les systèmes de sécurité. Des organisations comme GitHub, Wikimedia, Foursquare et le CERN l'adoptent pour gérer leurs besoins en matière de recherche et d'analyse de données.

En 2018, la société réalise une introduction en bourse, confirmant la maturité du projet. L'année suivante marque un changement significatif. Elastic, l'entreprise derrière Elasticsearch, modifie sa licence. En janvier 2021, elle abandonne la licence Apache 2.0 pour une double licence propriétaire, incluant la Server Side Public License (SSPL). Cette décision provoque une réaction d'Amazon Web Services, qui crée un fork du projet nommé OpenSearch, maintenant la tradition open source sous licence Apache 2.0.

L'impact d'Elasticsearch sur l'industrie du logiciel dépasse son objectif initial de moteur de recherche. La technologie a redéfini la manière dont les organisations gèrent et explorent leurs données. Son architecture distribuée et son approche orientée API ont influencé la conception de nombreux systèmes modernes. La simplicité d'utilisation, associée à des fonctionnalités puissantes, donne aux développeurs la possibilité d'intégrer des capacités de recherche sophistiquées sans expertise approfondie en recherche d'information. L'adoption massive d'Elasticsearch illustre l'importance croissante de l'analyse de données en temps réel dans les systèmes d'information contemporains.

Un projet personnel évolue parfois en une infrastructure majeure, utilisée par des milliers d'organisations à travers le monde.

HEIF

Le format JPEG règne depuis 1992 sur le monde de l'image numérique. Les appareils mobiles, les ordinateurs, les serveurs, tous utilisent ce format qui s'est imposé comme une norme de fait. Mais cette hégémonie cache des faiblesses techniques qui se sont révélées au fil du temps : pas de support pour la haute profondeur de couleur, pas de gestion de la haute gamme dynamique, pas de transparence. Et puis il y a ces *artefacts* visuels caractéristiques, cette compression qui montre ses limites face aux besoins actuels.

C'est en 2013 que l'équipe technique de Nokia Technologies commence à réfléchir à une alternative. Ils définissent les exigences, identifient les cas d'utilisation. Le développement des spécifications prend un an et demi, jusqu'à l'été 2015. Mais le destin du projet bascule avec le rachat de Nokia par Microsoft. L'entreprise cède ses droits et ses informations à Apple, qui reprend le flambeau.

HEIF (High Efficiency Image File Format) rompt avec la conception traditionnelle des formats d'images. Là où JPEG stocke une seule image, HEIF peut contenir plusieurs images, des séquences entières, leurs métadonnées. La structure repose sur les technologies de codage vidéo HEVC (H.265), dont les techniques avancées de compression sont appliquées aux images fixes. C'est un changement de paradigme.

En 2017, Apple franchit le pas. L'iPhone 7, la sixième génération d'iPad, les Mac sous OS X High Sierra adoptent le format. L'entreprise introduit l'extension .heic pour les images encodées en HEVC, réservant .heif aux fichiers utilisant d'autres codecs. Des millions d'utilisateurs découvrent le format sans vraiment s'en apercevoir.

Les gains sont immédiats : une réduction de 50% de la taille des fichiers par rapport à JPEG, pour une qualité visuelle équivalente voire supérieure. Le format gère la capture en rafale, l'enregistrement simultané de vidéos et d'images fixes, les animations, les images focales et d'exposition dans un seul fichier. Les opérations d'édition sur des images dérivées prédéfinies sont rendues possibles.

L'architecture de HEIF hérite du format de base ISO/IEC 14496-12. Le système de « boîtes » (*boxes*) donne la possibilité d'imbriquer les données, créant une hiérarchie de relations. La première boîte, nommée « ftyp », contient les métadonnées générales d'encodage. Une boîte « meta » renferme le reste des informations. Le format intègre le tuilage d'image, qui autorise le décodage parallèle ou le chargement partiel des zones d'intérêt dans les images haute résolution.

Apple a adapté les spécifications à sa manière. L'entreprise utilise systématiquement des tuiles de 512×512 pixels, générant des entrées « infe » pour chaque tuile. Ces tuiles sont assemblées en une seule image via une image dérivée de type « grid ». Apple exploite des cartes de profondeur auxiliaires, accompagnées de métadonnées XMP.

L'adoption du format s'étend progressivement. Android l'intègre depuis la version 9 (Pie), Samsung le propose sur certains modèles comme le Galaxy S10. Windows 10 l'a introduit dans sa version 1809, mais il faut installer des extensions spécifiques depuis le Microsoft Store. La transition demande des ajustements techniques et pratiques.

Pour la photographie mobile, l'impact est considérable. Les utilisateurs

conservent plus de photos sur leurs appareils sans sacrifier la qualité visuelle. L'édition non destructive est plus accessible, la gestion des métadonnées plus flexible. Les développeurs d'applications disposent de nouvelles possibilités.

Du côté de l'analyse numérique légale, généralement pour des enquêtes judiciaires, HEIF complique le travail. Avec la capacité de sa structure à dissimuler des données dans différentes boîtes, de manipuler les relations entre les images, les enquêteurs doivent développer de nouvelles méthodologies, tenir compte des multiples couches de données, des implémentations variables selon les fabricants.

La norme ISO/IEC 23008-12 définit aujourd'hui les spécifications techniques du format. Cette standardisation garantit l'interopérabilité entre les différentes implémentations. Les mises à jour successives ont introduit le support pour le codage prédictif des images, les rafales, l'annotation des régions. Le développement continue.

Le format HEIF s'adapte aux contraintes de performance des appareils modernes. Mais la transition depuis JPEG est progressive. Les questions de compatibilité freinent l'adoption, mais les avantages techniques sont indéniables. Le changement prend du temps, et le match ne semble pas gagné d'avance.

Rust

En 2006, Graydon Hoare travaillait chez Mozilla quand il lança un projet personnel dans la programmation système. Ce projet, baptisé Rust en référence aux champignons de la rouille et au mot anglais *robust*, resta discret jusqu'en 2010. Cette année-là, Mozilla comprit le potentiel du langage et décida de soutenir son développement.

La première version pré-alpha du compilateur sortit en janvier 2012. Dès le départ, Rust s'attaquait à une contradiction qui empoisonnait les développeurs depuis des décennies : comment obtenir un contrôle fin sur les ressources matérielles tout en garantissant la sécurité mémoire ? Les langages existants forçaient à choisir entre performance et sûreté. Rust refusait ce compromis.

Le système de types de Rust repose sur deux concepts originaux : le *ownership* (propriété) et le *borrowing* (emprunt). Le principe est simple : une seule entité peut posséder une ressource à un instant donné. Cette règle, vérifiée à la compilation, élimine d'un coup toute une catégorie de

bogues. Les problèmes de concurrence, les fuites mémoire, les accès invalides disparaissent avant l'exécution du programme.

Microsoft a révélé un chiffre édifiant : 70% des failles de sécurité corrigées dans ses produits provenaient d'erreurs de gestion mémoire. Mozilla constatait la même chose avec Firefox. Ces statistiques expliquent pourquoi Rust a trouvé son public. Le langage obtient d'ailleurs le titre de « langage le plus apprécié » dans le sondage Stack Overflow chaque année depuis 2016.

L'écosystème s'est structuré autour de Cargo, le gestionnaire de paquets officiel. Cet outil simplifie radicalement la gestion des dépendances et la construction des projets. La communauté a développé des milliers de bibliothèques, appelées *crates*, qui couvrent à peu près tous les besoins imaginables. Cette richesse a facilité l'adoption du langage dans l'industrie.

Dropbox fut parmi les premiers à parier sur Rust pour ses infrastructures critiques. Facebook, Amazon et Cloudflare ont suivi. Microsoft explore son usage pour renforcer la sécurité de ses composants système. Mozilla a réécrit certaines parties de Firefox en Rust, démontrant qu'on pouvait intégrer le langage dans des projets monumentaux déjà en production.

La gestion de la concurrence constitue une autre force de Rust. Le compilateur détecte les data races avant la première exécution. Impossible de lancer un programme où plusieurs threads accèdent simultanément à une donnée partagée sans synchronisation appropriée. Cette vérification statique épargne des heures de débogage cauchemardesque.

Le langage évolue avec une discipline remarquable. Les concepteurs introduisent des fonctionnalités avancées sans jamais rompre la compatibilité avec le code existant. Le processus de développement s'appuie sur des groupes de travail spécialisés qui débattent longuement de chaque proposition. Cette gouvernance produit des décisions mûrement réfléchies.

Rust s'est imposé dans des domaines variés. On le trouve dans les systèmes d'exploitation, les navigateurs web, les outils de développement, les applications distribuées. L'Internet des objets adopte le langage pour ses garanties de sécurité et son efficacité sur matériel limité. Les systèmes embarqués critiques bénéficient d'un langage qui combine les performances du C avec des protections absentes des langages bas niveau traditionnels.

Le langage Rust prouve que concilier sécurité et performance se fait sans sacrifier la productivité. D'autres projets de langages s'inspirent de ses innovations. Les pratiques de développement sécurisé évoluent sous son impulsion. Des entreprises proposent formation, conseil et services spécialisés, signalant la maturité commerciale atteinte.

Le système de types sophistiqué de Rust représente une avancée conceptuelle. Il démontre qu'un compilateur peut vérifier des propriétés complexes sur le code et garantir leur respect à l'exécution. Cette approche ouvre des

perspectives nouvelles dans la conception des langages. La sécurité n'est plus un objectif contradictoire avec la performance.

De nouvelles fonctionnalités apparaissent régulièrement, toujours dans le respect des principes fondateurs. L'adoption industrielle s'accélère. La communauté reste dynamique et le langage s'inscrit durablement dans le paysage informatique.

Pendant des décennies, les développeurs acceptaient comme inévitables les bugs mémoire et les failles de sécurité associées. Mais Rust a cassé cette fatalité. Il apporte des solutions concrètes à des problèmes anciens, tout en maintenant les performances des langages établis. Cette double victoire technique et communautaire montre qu'on peut repenser les outils de développement pour répondre aux exigences contemporaines de sécurité et de qualité.

ChromeOS

Google lance en 2009 le développement d'un système d'exploitation qui rompt avec les conventions établies : ChromeOS, pensé pour ne fonctionner qu'avec des applications web. Le contexte explique cette audace. Les *netbooks*, ces petits ordinateurs portables bon marché tournés vers Internet, séduisent de plus en plus d'utilisateurs. L'entreprise choisit une méthode inhabituelle pour comprendre leurs besoins : plutôt que de commander des études classiques, les ingénieurs distribuent 200 machines ChromeOS aux employés de Google. Matthew Papakipos, qui dirige techniquement le projet, installe trois appareils chez lui. Il observe que les sessions sont brèves : une recherche rapide, un court message envoyé, puis l'appareil est refermé.

Le 7 juillet 2009, Google annonce officiellement ChromeOS. En novembre, le code source est accessible sous le nom de Chromium OS. Cette dualité perdure : ChromeOS désigne le produit commercial distribué uniquement sur du matériel spécifique via des partenaires, tandis que Chromium OS reste le projet open source que tout développeur peut compiler et modifier. L'interface épurée s'inspire du navigateur Chrome, reflétant la vocation première du système : accéder au web, vite et simplement.

Le 19 novembre 2009, Sundar Pichai, vice-président de Google supervisant Chrome, présente une version préliminaire lors d'une conférence de presse. La démonstration montre un bureau qui ressemble au navigateur Chrome, enrichi d'onglets d'applications occupant moins de place et pouvant être épinglés pour y accéder plus facilement. Le temps de démarrage

atteint alors sept secondes. Les équipes s'attèlent à le réduire encore. Chris Kenyon, vice-président des services OEM chez Canonical, qui édite Ubuntu, annonce une contribution technique sous contrat. Google veut s'appuyer sur les composants existants de la communauté open source plutôt que tout réinventer.

Une interface adaptée aux tablettes tactiles est présentée en janvier 2010. Le design intègre un clavier virtuel et des icônes plus grandes, placées au-dessus des onglets. En mars 2010, Google envisage deux versions : une édition grand public et une version entreprise.

Le prototype Cr-48 est dévoilé en décembre 2010. Cette machine de test, produite à 60 000 exemplaires, introduit une innovation curieuse : la touche « Verrouillage majuscule » disparaît au profit d'une touche de recherche dédiée. Les critiques s'élèvent contre les limitations en mode hors-ligne. Google réagit en démontrant une version d'applications fonctionnant sans connexion, comme Google Docs, et annonce un forfait 3G incluant 100 Mo de données gratuites mensuelles via Verizon.

L'architecture de ChromeOS se déploie sur trois niveaux : le micrologiciel, le navigateur avec son gestionnaire de fenêtres, et les services système. Le micrologiciel accélère le démarrage en évitant la détection de périphériques obsolètes. La sécurité constitue une préoccupation constante, avec une vérification de chaque étape du processus de démarrage et des capacités de restauration intégrées.

La gestion de l'impression illustre l'approche de Google. Le service Google Cloud Print autorise l'impression depuis n'importe quel appareil sur n'importe quelle imprimante, échappant à la complexité habituelle des pilotes d'impression. Le système intègre un lecteur multimédia capable de lire les fichiers MP3 et JPEG hors connexion.

Les premiers utilisateurs du Cr-48 livrent des retours contrastés. Kurt Bakke de Conceivably Tech souligne son adoption rapide comme appareil familial pour les recherches rapides et la consultation des réseaux sociaux. Mais les performances avec Flash déçoivent, et l'absence de fonctionnalités hors-ligne suscite des critiques.

L'intégration matérielle s'effectue via des partenariats avec Acer, Adobe, Asus, Intel, Samsung et Dell. En juillet 2010, Eric Schmidt, PDG de Google, écarte l'idée d'un netbook ChromeOS de marque Google. Il préfère s'appuyer sur l'expérience des constructeurs PC habitués à travailler avec Microsoft.

Le modèle économique de ChromeOS se distingue des systèmes traditionnels. Schmidt reconnaît que la compatibilité logicielle sera plus limitée que Windows, mais la gratuité de la licence autorise les constructeurs à proposer des appareils compétitifs, entre 300 et 400 dollars. Glyn Moody

du *Linux Journal* anticipe même une distribution gratuite du matériel pour développer le modèle publicitaire de Google.

L'adoption de ChromeOS s'accélère à partir de 2015, notamment dans les écoles américaines. IDC rapporte 4,6 millions de Chromebooks vendus en 2014, soit deux fois plus qu'en 2013. Gartner table sur 14,4 millions d'unités en 2017. L'intégration progressive des applications Android et la possibilité d'exécuter des applications Windows via des solutions de virtualisation étendent les capacités du système.

Les performances des Chromebooks basés sur les processeurs Intel démontrent l'importance de l'optimisation matérielle. Les tests comparatifs révèlent des gains de 50% et une autonomie supérieure de 58% par rapport aux solutions ARM concurrentes. Ces résultats témoignent de l'investissement continu des ingénieurs pour améliorer l'efficacité du système.

ChromeOS propose une approche centrée sur le web qui utilise le cloud computing. Sa simplicité, sa sécurité et son modèle de mise à jour automatique influencent aujourd'hui la conception des systèmes d'exploitation modernes. Ce système d'exploitation est loin d'avoir remplacé les systèmes en place, mais il a tracé une voie différente, prouvant qu'une alternative légère et connectée pouvait trouver sa place dans un marché dominé par des géants établis.

NVMe

Les disques SSD commencent à se répandre au début des années 2010, mais se heurtent à un problème inattendu. Ces nouveaux supports dépassent en performances les disques durs mécaniques, pourtant ils restent freinés par les protocoles de communication qui n'ont pas été pensés pour eux. Les interfaces SATA et SAS, conçues pour dialoguer avec des disques à plateaux tournants, imposent leurs contraintes à des mémoires capables de bien mieux faire.

Un groupe de travail regroupant les grands noms de l'industrie décide en 2011 de prendre le problème à bras-le-corps. Plutôt que d'essayer d'adapter l'existant, ces ingénieurs repartent d'une page blanche. Ils publient la première spécification du NVMe (Non-Volatile Memory Express), version 1.0. L'idée directrice est simple : créer une interface native pour les mémoires non volatiles, qui exploite directement le bus PCI Express et dialogue naturellement avec les processeurs modernes.

La proposition bouscule les habitudes. Là où SATA ne gère que 32 com-

mandes dans une file unique, et SAS 256, le NVMe crée jusqu'à 64 000 files d'attente, chacune capable de contenir 64 000 commandes. Cette architecture multi-files s'accorde avec les processeurs multicœurs qui équipent désormais les machines. Chaque application peut disposer de ses propres files, sans interférer avec les autres.

La version 1.1 arrive en octobre 2012 avec son lot d'améliorations. Le protocole intègre la gestion du multi-chemin d'accès et le partage des ressources, des fonctions indispensables dans les environnements professionnels. Les mécanismes de gestion autonome de l'alimentation font aussi leur apparition, anticipant les besoins des ordinateurs portables et des appareils mobiles.

En 2014, la création de NVM Express, Inc. structure la gouvernance du standard. Cette organisation rassemble plus d'une centaine d'entreprises et accélère le développement. La version 1.2 sort en novembre avec des perfectionnements notables : gestion avancée des espaces de nommage, optimisation des tampons mémoire, possibilité de mettre à jour le firmware sans interruption.

Les formats physiques se multiplient pour s'adapter aux usages. Les cartes d'extension PCIe équipent les serveurs et les stations de travail. Les modules M.2 trouvent leur place dans les ordinateurs portables et les machines compactes. Les disques U.2, au format 2,5 pouces, s'installent dans les baies de stockage traditionnelles. Cette diversité facilite l'adoption du NVMe dans tous les segments du marché.

Le protocole intègre des mécanismes de sécurité conformes aux spécifications du Trusted Computing Group. Le chiffrement des données au repos et l'effacement sécurisé répondent aux exigences croissantes en matière de confidentialité. Ces fonctions sont incontournables dans un contexte où la protection des données s'impose comme une priorité.

L'année 2015 apporte la spécification NVMe-MI (Management Interface). Cette extension standardise la gestion hors bande des périphériques, unifiant les commandes de configuration, de surveillance et de maintenance. Les administrateurs système gagnent en simplicité et en cohérence dans la gestion de leurs infrastructures de stockage.

Le tournant de 2016 voit naître NVMe over Fabrics. Cette évolution étend le protocole aux réseaux de stockage, qu'ils utilisent Ethernet, Fibre Channel ou InfiniBand. Le NVMe sort ainsi du cadre des connexions directes pour s'intégrer dans les architectures réseau complexes, sans perdre ses avantages en termes de performances et de latence.

La version 1.3, publiée la même année, enrichit le standard de fonctions destinées aux appareils mobiles. L'optimisation de la gestion de l'énergie et l'intégration de mécanismes adaptés aux systèmes embarqués font du NVMe

la première interface de stockage véritablement universelle. Du téléphone au centre de données, un seul protocole couvre désormais tous les besoins.

Les chiffres parlent d'eux-mêmes. Le NVMe divise par deux le nombre d'instructions processeur nécessaires au traitement des entrées-sorties par rapport à SATA ou SAS. Cette efficacité se traduit par une latence réduite et un débit supérieur. Les périphériques NVMe dépassent le million d'opérations par seconde, là où les interfaces précédentes plafonnaient bien en deçà.

Le support logiciel suit. Windows, Linux et MacOS intègrent des pilotes natifs. Les environnements de virtualisation comme VMware ESXi adoptent le protocole, permettant son utilisation dans les infrastructures virtualisées qui constituent le cœur des centres de données modernes.

Le succès du NVMe tient à une approche globale. En repensant entièrement l'interface entre processeurs et mémoires non volatiles, le protocole supprime les goulots d'étranglement hérités des anciennes architectures. Cette refonte exploite pleinement les performances des nouvelles technologies de mémoire, sans compromis lié à des contraintes obsolètes.

En quelques années, le NVMe s'impose comme un standard incontournable, soutenu par un écosystème riche en produits et solutions. Son évolution continue garantit sa pertinence face aux futures technologies de stockage, confirmant sa place centrale dans l'architecture des systèmes informatiques contemporains.

Dart

En octobre 2011, lors de la conférence GOTO à Aarhus au Danemark, Lars Bak et Kaspar Lund présentent au public un nouveau langage de programmation développé chez Google : Dart. L'ambition affichée est claire : offrir une alternative crédible à JavaScript pour le développement web, un langage qui échapperait aux défauts et aux limites que les développeurs reprochent alors au vénérable JavaScript.

Les deux ingénieurs de Google ont conçu Dart avec une idée en tête : unifier le développement logiciel sur différentes plateformes sans sacrifier les performances. La syntaxe du langage emprunte volontairement aux langages établis comme Java, C ou JavaScript, histoire de ne pas trop désorienter les développeurs qui franchiraient le pas. Cette familiarité calculée vise à faciliter l'adoption : pas besoin de tout réapprendre depuis zéro.

Le 14 novembre 2013, Google publie la version 1.0 de Dart. C'est la première mouture stable, celle qui doit convaincre la communauté que Dart peut bel et bien remplacer JavaScript dans les applications web. L'année suivante, en juillet 2014, Ecma International valide le langage lors de sa 107e Assemblée Générale et crée le standard ECMA-408. Cette reconnaissance officielle confère à Dart une légitimité qu'un simple projet d'entreprise n'aurait pas eue.

Mais les débuts ne sont pas aussi glorieux qu'espéré. Google nourrit l'ambition d'intégrer directement dans Chrome une machine virtuelle Dart pour booster les performances des applications. L'idée fait grincer des dents. Les développeurs web craignent une fragmentation du web, où chaque navigateur imposerait sa propre machine virtuelle. Face à cette résistance, Google fait marche arrière. En 2015, avec la version 1.9, le projet d'intégration native dans les navigateurs est abandonné.

L'histoire de Dart aurait pu s'arrêter là, cantonnée à un énième langage web sans grand relief. Mais 2017, Google lance Flutter, un framework pour créer des interfaces utilisateur qui fonctionnent sur mobile, web et desktop à partir d'un même code source. Et Flutter choisit Dart comme langage principal. Cette alliance redonne un souffle au langage, qui trouve enfin sa vraie raison d'être : servir de socle au développement multiplateforme.

Août 2018 marque une rupture avec l'arrivée de Dart 2.0. Cette version restructure le langage, notamment par l'introduction d'un système de typage bien plus strict. Le code gagne en sécurité et en lisibilité, deux qualités indispensables pour les applications complexes. Le compilateur est également remanié pour produire du code natif performant, les applications Flutter pouvant ainsi atteindre des performances comparables aux solutions développées spécifiquement pour iOS ou Android.

Dart propose deux modes de compilation distincts. La compilation anticipée (Ahead-of-Time ou AOT) transforme le code en JavaScript optimisé pour le web, ou en code machine natif pour les applications mobiles. La compilation à la volée (Just-in-Time ou JIT) facilite un développement plus fluide grâce au rechargement à chaud du code : on modifie le code, et l'application se met à jour instantanément sans redémarrage complet. Cette souplesse accélère considérablement le cycle de développement.

Le langage adopte une approche résolument orientée objet : tout est objet, y compris les nombres et les fonctions. Classes, interfaces, mixins, génériques... Dart reprend les concepts familiers des langages modernes. La gestion automatique de la mémoire libère le développeur des tâches fastidieuses d'allocation et de désallocation manuelle des ressources.

Le succès de Flutter tire Dart vers le haut. La version 2.7 introduit les méthodes d'extension, qui permettent d'ajouter de nouvelles fonctionnalités

aux types existants sans modifier leur code source. Cette astuce s'avère précieuse pour enrichir les interfaces utilisateur développées avec Flutter.

L'écosystème s'étoffe progressivement. Le SDK Dart intègre un compilateur dart2js pour générer du JavaScript, un analyseur statique qui détecte les erreurs avant l'exécution, un formateur de code et un gestionnaire de paquets. Ces outils accompagnent le développeur tout au long du processus de création.

L'approche technique de Flutter distingue Dart des autres solutions multiplateformes. Au lieu de compiler vers du code natif iOS ou Android, Flutter dessine directement chaque pixel à l'écran grâce à son propre moteur de rendu. Cette stratégie garantit une expérience utilisateur identique sur toutes les plateformes, sans dépendre des composants graphiques natifs de chaque système.

La machine virtuelle de Dart a été conçue pour tirer le meilleur parti du langage. Le *garbage collector* gère la mémoire avec efficacité, tandis que le compilateur produit du code optimisé pour chaque plateforme cible, grâce à quoi les applications Dart atteignent des performances élevées tout en conservant une base de code unique.

Des entreprises de toutes tailles utilisent Dart et Flutter pour développer leurs applications. La possibilité de réutiliser le code entre mobile, web et desktop représente un gain de temps et d'argent considérable. On écrit une fois, on déploie partout, avec des performances au rendez-vous.

L'avenir de Dart reste intimement lié à celui de Flutter. Google continue d'investir dans les deux technologies, publiant régulièrement des mises à jour qui enrichissent les fonctionnalités et améliorent les performances. Le langage évolue pour répondre aux besoins du développement moderne, avec un objectif constant : rendre les développeurs plus productifs.

Parti comme une alternative à JavaScript, Dart a finalement trouvé sa voie dans le développement multiplateforme. Son association avec Flutter lui a donné une seconde jeunesse et un véritable terrain de jeu.

Kotlin

En 2010, JetBrains, une entreprise pragoise spécialisée dans les outils de développement, se lance dans la création d'un nouveau langage de programmation. L'objectif est de concevoir une alternative à Java, plus concise et plus sûre, sans pour autant renoncer à la compatibilité avec la machine virtuelle Java. Cette compatibilité n'est pas un détail puisqu'elle garantit

que les développeurs pourront adopter progressivement le nouveau langage sans réécrire leur code existant.

Le projet sort de l'ombre en juillet 2011 sous le nom de « Project Kotlin ». Quelques mois plus tard, en février 2012, JetBrains ouvre le code source sous licence Apache 2.0. Ce choix d'ouverture n'est pas anodin : il traduit une volonté d'impliquer la communauté des développeurs dès les premières heures du projet. Quatre ans plus tard, le 15 février 2016, la version 1.0 débarque. Cette première version stable marque le véritable départ de Kotlin, qui commence à séduire par sa syntaxe épurée et son approche pragmatique. Un chiffre frappe d'emblée : Kotlin réduit en moyenne de 40% le nombre de lignes de code nécessaires comparé à Java pour accomplir une même tâche.

L'année 2017 bouleverse la donne. En mai, Google annonce son support officiel pour le développement Android, faisant de Kotlin une alternative crédible à Java. Cette consécration propulse le langage au cœur du développement mobile. Mais Kotlin ne s'arrête pas là. En avril, il franchit un nouveau cap en devenant multiplateforme grâce à Kotlin/Native et au support stabilisé pour JavaScript. Six mois plus tard, la version 1.2 introduit la possibilité de partager du code entre backend, frontend web et applications mobiles Android. Cette capacité répond à un besoin grandissant : réutiliser son code sur différentes plateformes sans tout réécrire.

En octobre 2018, Google et JetBrains créent ensemble la Kotlin Foundation. Cette structure garantit l'indépendance du langage et assure sa gouvernance de façon transparente. Un an plus tard, en 2019, Google va plus loin en déclarant Kotlin comme langage privilégié pour Android. Concrètement, cela signifie que les nouvelles fonctionnalités et API Android arrivent d'abord en Kotlin. Le langage reçoit le prix « Breakout Project of the Year » lors de la convention O'Reilly Open Source Software, reconnaissance de son ascension fulgurante.

Ce qui séduit dans Kotlin, c'est d'abord sa façon de traiter la nullabilité. Le système de types intègre directement cette gestion, réduisant drastiquement les erreurs liées aux références nulles qui empoisonnent la vie des développeurs Java. Les expressions lambda et les fonctions d'extension enrichissent les possibilités de programmation fonctionnelle. Quant aux classes de données, elles simplifient la création d'objets destinés au stockage d'informations.

Le langage évolue vite. La version 1.1 apporte les coroutines qui facilitent la programmation asynchrone. La version 1.3 introduit les contrats et les classes *inline*, améliorant performances et sécurité du code. Le compilateur transforme le code source en *bytecode* JVM, rendant son exécution possible sur toutes les plateformes supportant Java.

Les chiffres parlent d'eux-mêmes. En 2020, plus de 4,7 millions de développeurs utilisent Kotlin, dont 65% en production. Pour 56% d'entre eux, c'est devenu leur langage principal. Des géants comme Google, Uber, Atlassian, Pinterest et Square l'ont intégré dans leur code de production. Kotlin est le 13e langage de programmation le plus populaire, dépassant Scala et Swift. Le rapport GitHub de 2018 le classait déjà comme le langage connaissant la croissance la plus rapide. Stack Overflow le place quatrième dans le classement des langages préférés des développeurs professionnels.

La communauté grossit. En 2020, plus de 100 personnes travaillent sur le langage chez JetBrains, épaulées par plus de 350 contributeurs indépendants qui participent à son évolution. Cette effervescence collective nourrit le développement continu du langage et de son écosystème.

Kotlin réduit le code passe-partout qui alourdit Java, prévient les erreurs courantes et simplifie des tâches complexes. Son approche multiplateforme répond aux attentes actuelles du développement logiciel.

WebRTC

WebRTC est une technologie est née de la rencontre entre les navigateurs web et la communication temps réel. Ses racines plongent dans plusieurs décennies d'évolution des protocoles internet.

Retour en 1996. La publication du Real-time Transport Protocol (RTP) dans le RFC 1889 jette les bases de ce qui deviendra WebRTC. Ron Frederick, l'un des auteurs de ce protocole, avait commencé dès octobre 1992 des expérimentations avec la carte de capture vidéo Sun VideoPix. Son but était de créer un outil de visioconférence réseau basé sur la multidiffusion IP. Le projet s'appelle « nv » (Network Video tool) et s'inspire de « vat », un outil de conférence audio développé par LBL qui utilisait un protocole de session léger pour gérer les participants.

La compression vidéo représentait un enjeu technique considérable. Frederick voulait obtenir un flux acceptable dans une bande passante limitée à 128 kbps, soit la capacité d'une ligne ISDN standard. Cela exigeait un facteur de compression d'environ 20 pour la taille d'image et la fréquence visées. Frederick parvint à développer une technique innovante qui fit l'objet d'un brevet (US5485212A) sur la compression vidéo logicielle pour la téléconférence.

La première version de nv sort en novembre 1992. Elle retransmet certaines sessions de l'IETF dans le monde entier. L'impact est immédiat :

environ 200 sous-réseaux dans 15 pays peuvent recevoir cette diffusion, et entre 50 et 100 personnes utilisent nv durant cette période. Le succès se confirme lors de différents ateliers internationaux, notamment l'Australian NetWorkshop et le MultiG workshop sur les réalités virtuelles distribuées en Suède.

Frederick rend le code source public en février 1993. Il introduit ensuite un nouveau système de compression basé sur les ondelettes, et ajoute le support de la vidéo couleur. Le protocole réseau utilisé par nv et d'autres outils de conférence Internet sert de base au protocole RTP que l'IETF normalise. RTP est un standard incontournable, d'abord publié dans les RFC 1889-1890, puis révisé dans les RFC 3550-3551.

L'évolution de nv se poursuit avec des portages sur différentes plateformes matérielles et périphériques de capture vidéo. La NASA choisit cet outil pour diffuser en direct la couverture des missions de la navette spatiale. En 1994, Frederick enrichit nv avec des algorithmes de compression développés par d'autres, incluant des systèmes matériels comme le format CellB de la carte de capture SunVideo.

Cette histoire précoce des communications temps réel sur Internet esquisse WebRTC. Les défis rencontrés par les pionniers comme Frederick ont établi les bases des protocoles modernes. La multidiffusion IP permettait une distribution efficace des données vers de multiples destinataires, une approche différente de l'architecture unicast adoptée plus tard par WebRTC.

Le passage de la multidiffusion à l'unicast s'explique par des raisons pratiques. La complexité de mise en œuvre de la multidiffusion dans les réseaux existants a conduit à privilégier des solutions plus simples à déployer. Cette évolution a nécessité l'introduction de nouvelles architectures, comme les SFU (Selective Forwarding Units), pour gérer la distribution des flux médias entre participants.

L'histoire de WebRTC prend un tournant avec l'arrivée de Serge Lachapelle. Sa passion pour les logiciels de communication remonte aux années 1990, où il développe un projet pour rejoindre des appels vidéo directement depuis un navigateur. Cette expérience le conduit à cofonder Marratech, une entreprise spécialisée dans la visioconférence de groupe. Google acquiert Marratech en 2007 et intègre une expertise précieuse dans son équipe.

Le développement de la messagerie vocale et vidéo de Gmail constitue une étape intermédiaire significative. Cette réalisation implique l'intégration complexe de différents composants : l'audio sous licence GIPs, la vidéo de Vidyo et le réseau via libjingle. Justin Uberti relève le défi technique de faire fonctionner ensemble ces systèmes aux API disparates, ce qui nécessite des connaissances approfondies en réseaux, cryptographie et médias.

Le lancement du projet Chrome chez Google ouvre un nouveau chapitre.

L'ambition d'enrichir les capacités du navigateur, combinée à une volonté d'abandonner l'API NPAPI pour des raisons de sécurité, crée un contexte favorable à l'émergence de WebRTC. L'architecture sandbox de Chrome, qui isole les opérations potentiellement dangereuses dans des processus distincts, influence la conception sécurisée de WebRTC.

La naissance de WebRTC répond à trois objectifs principaux. D'abord, simplifier le développement d'expériences de communication temps réel. Ensuite, promouvoir une communication humaine ouverte et non entravée. Enfin, garantir la sécurité par défaut. Pour concrétiser cette vision, Google acquiert et rend open source quelques technologies clés, notamment celles d'On2 pour la vidéo et de Global IP Solutions pour les communications temps réel.

La standardisation de WebRTC représente un effort sans précédent. Elle nécessite la collaboration de deux organismes de normalisation (IETF et W3C) et de centaines d'experts internationaux. Harald Alvestrand, fort de son expérience à l'IETF, initie le processus. Une réunion informelle à Maastricht en 2010 rassemble des développeurs de Google, Cisco, Ericsson, Skype, Mozilla et Linden Labs. Les bases de la spécification sont posées.

WebRTC s'appuie sur l'héritage des technologies existantes. Il évite de réinventer des solutions déjà éprouvées. La décision de ne pas standardiser la signalisation, traitée par SIP et d'autres protocoles, illustre cette approche pragmatique. Cette stratégie concentre les efforts sur l'innovation plutôt que sur la redéfinition de concepts établis.

L'utilisation de WebRTC dans le cloud gaming démontre sa polyvalence. Les avancées en intelligence artificielle améliorent la qualité des communications, notamment par la suppression du bruit de fond. Cette extension des usages témoigne de la robustesse et de la flexibilité de l'architecture WebRTC. La collaboration internationale et l'engagement en faveur de standards ouverts ont créé une plateforme qui continue d'innover et de s'adapter aux nouveaux usages.

WebSocket

Au début des années 2000, les développeurs web se heurtent à un problème récurrent. Les applications qu'ils conçoivent – messageries instantanées, jeux en ligne, tableaux de bord collaboratifs – exigent des échanges constants entre navigateurs et serveurs. Or le Web, tel qu'il existe alors, ne sait pas vraiment gérer ce type de dialogue. HTTP, le protocole qui régit les

communications sur Internet, fonctionne selon un modèle rigide : le client demande, le serveur répond. Pas de place pour une conversation fluide dans les deux sens.

Les ingénieurs trouvent des parades. La plus répandue, le « HTTP polling », consiste à bombarder le serveur de requêtes régulières pour vérifier s'il a du nouveau à dire. Une variante, le « long polling », maintient la connexion ouverte jusqu'à ce qu'une information arrive. Ces bidouilles fonctionnent, certes, mais génèrent un trafic réseau considérable. Chaque requête HTTP trimballe des en-têtes volumineux, des centaines d'octets à chaque fois, pour transporter parfois quelques bytes d'information utile.

La RFC 6455, publiée par l'IETF en décembre 2011, vient mettre fin à cette époque d'improvisation. WebSocket propose une approche radicalement différente. Le protocole démarre par une poignée de main HTTP classique, puis bascule vers une connexion permanente. Une fois établi, ce canal envoie des messages dans les deux sens avec un *overhead* minimal – quelques octets de contrôle au lieu des centaines précédemment nécessaires. Le protocole gère indifféremment du texte ou du binaire, et intègre des mécanismes de surveillance de la connexion via des messages « ping » et « pong ».

Les navigateurs adoptent cette innovation. Chrome, Firefox et Safari déploient des implémentations expérimentales dès 2010. Internet Explorer rejoint le mouvement en 2012. Cette convergence rapide des éditeurs témoigne d'un besoin réel du marché. Les développeurs peuvent enfin créer des applications web réactives sans passer par des contournements complexes et fragiles.

Les chiffres parlent d'eux-mêmes. Sur un site de trading affichant des cours boursiers en temps réel, WebSocket réduit le trafic de 417 octets par mise à jour et par client. Avec 4 835 visiteurs simultanés, l'économie de bande passante grimpe à 16 mégabits par seconde. L'impact se mesure aussi dans les statistiques d'adoption : 1,6% des sites du top million en 2018, 6,3% en 2020. Cette progression témoigne d'une technologie arrivée à maturité.

L'écosystème se structure autour du protocole. Socket.io et SockJS proposent des API de haut niveau qui simplifient son utilisation. Les services de chat, les plateformes collaboratives et les applications financières en font leur cheval de bataille. Les outils d'analyse web l'utilisent pour collecter des données sur le comportement des utilisateurs en temps réel. WebSocket trouve sa place dans l'Internet des objets et les architectures microservices, où sa légèreté et sa bidirectionnalité offrent des avantages tangibles.

La technologie n'échappe pas aux dérives. Certains développeurs négligent les bonnes pratiques de sécurité – vérification des en-têtes Origin,

chiffrement TLS systématique. Des acteurs malveillants détournent le protocole pour du pistage ou la diffusion de logiciels indésirables. Ces zones d'ombre rappellent qu'aucune innovation technique n'est neutre.

HTTP/2, standardisé en 2015, introduit la fonctionnalité Server Push. Les serveurs peuvent maintenant envoyer des ressources aux clients sans attendre de requête. Cette capacité ne remplace pourtant pas WebSocket : elle ne crée pas de véritable canal bidirectionnel. Google l'admet implicitement en abandonnant le support de Server Push dans Chrome en 2020, confirmant la position unique de WebSocket dans le paysage web.

Des technologies concurrentes émergent, comme les Server-Sent Events. WebSocket conserve néanmoins sa pertinence. La stabilisation de son taux d'adoption depuis 2019 suggère qu'il a trouvé sa place dans la palette des outils disponibles. Les frameworks modernes l'intègrent par défaut, les développeurs le maîtrisent, les cas d'usage se multiplient.

Le protocole n'a rien révolutionné : il a simplement rendu possible ce qui auparavant nécessitait des acrobaties. Cette simplicité conceptuelle, associée à une efficacité opérationnelle mesurable, explique son succès.

Go

En 2007, chez Google, des ingénieurs se trouvaient face à un problème devenu courant dans l'entreprise. Les langages existants ne répondaient plus aux besoins des applications modernes : réseautage intensif, processeurs multicœurs, environnements distribués. Rob Pike, l'un des concepteurs du futur langage Go, résumait cette impasse technique avec une certaine frustration. L'industrie disposait de nombreux outils, certes, mais aucun ne convenait vraiment.

Le cloud computing sortait tout juste de l'enfance, l'architecture orientée services trouvait ses marques dans les grandes organisations, et l'intégration continue commençait à transformer les pratiques de développement. Le paysage informatique ressemblait déjà à celui d'aujourd'hui. Go n'est pas né dans un laboratoire coupé du monde réel ; il a émergé au cœur des contraintes que rencontraient les équipes de développement.

Un détail frappe dans cette histoire : parmi les créateurs de Go se trouve Ken Thompson, celui qui avait conçu UNIX et le langage C des décennies plus tôt. Cette filiation n'a rien d'anecdotique. Elle trace une ligne directe entre deux époques de l'informatique, Go incarnant une réinterprétation contemporaine des principes qui guidaient la création du C.

Java, apparu en 1991, s'était imposé grâce à sa portabilité. Mais son statut propriétaire jusqu'en 2006 créait des complications : le procès Oracle contre Google en 2012 en témoigne. Quant au C, né en 1972 pour réécrire UNIX, il s'était répandu faute d'alternative ; il était l'un des rares langages rapides et multiplateformes accessibles aux premiers programmeurs. Go, lui, a été pensé dès l'origine pour les besoins actuels.

Le langage se distingue par sa conception réfléchie. Les temps de compilation restent courts, le code machine se génère directement, le typage fort impose une discipline qui rend les programmes lisibles et fiables. La documentation, exhaustive et libre, accompagne le tout. Dans une équipe de développement, souvent nombreuse et aux compétences variées, ces caractéristiques prennent une autre dimension : la collaboration rendue plus fluide, les délais de construction se réduisent, le code existant se déchiffre plus aisément.

Avec la gestion automatique de la mémoire, les développeurs n'ont plus à programmer explicitement l'allocation et la libération mémoire, contrairement au C ou au C++. Cette automatisation améliore la productivité, certes, mais surtout elle renforce la sécurité en diminuant les vulnérabilités liées aux erreurs de manipulation mémoire.

L'intégration native de la concurrence représente sans doute l'innovation la plus significative. Les applications peuvent exécuter plusieurs tâches simultanément plutôt que séquentiellement, exploitant ainsi les processeurs multicœurs. Pour les applications distribuées sur de multiples serveurs ou les architectures en microservices, cette fonctionnalité se révèle précieuse.

Le déploiement des applications Go reste simple. Contrairement à Java qui nécessite des interpréteurs locaux ou des machines virtuelles, les programmes Go se compilent en binaires autonomes. L'outil Get télécharge et installe une application via une commande unique. Cette approche correspond parfaitement aux pratiques DevOps qui privilégient la cohérence des chaînes de développement et la simplicité des mises à jour.

Dans le domaine de la science des données, Go démontre sa polyvalence. Il intègre aisément des algorithmes écrits en Python dans des applications en microservices. Les performances obtenues évitent souvent le recours à l'optimisation en C ou C++, langages bas niveau réputés plus complexes.

La communauté de développeurs autour de Go n'a cessé de croître. En 2016, plus de 700 programmeurs contribuaient à son développement. Le classement TIOBE des langages de programmation illustre cette ascension : Go passe de la 55^{e} à la 10^{e} place entre 2016 et 2017.

De nombreuses plateformes open source ont adopté Go. Cloud Foundry l'a choisi pour des composants comme `gorouter`, qui gère la connectivité des applications en microservices. Docker, qui a transformé le déploiement

applicatif, est entièrement écrit dans ce langage. Les développeurs Docker ont opté pour Go en 2013, séduits par sa facilité d'installation, de test et d'adoption. InfluxDB, système de stockage haute performance, tire parti de l'efficacité et de la fiabilité du langage.

Des entreprises technologiques de premier plan ont intégré Go dans leurs infrastructures. Dropbox a migré une partie importante de son infrastructure de Python vers Go pour améliorer son extensibilité. Netflix utilise Go pour créer des microservices nécessitant performance et scalabilité. Uber a développé en Go son service de géolocalisation en temps réel, qui affiche les meilleures performances parmi tous ses services.

Le succès de Go s'explique par son adéquation avec les besoins du développement logiciel moderne. Sa conception prend en compte les environnements cloud, le traitement distribué et les architectures en microservices. La simplicité d'utilisation, associée à des performances élevées et une maintenance facilitée, en fait un choix pertinent pour les entreprises qui modernisent leurs applications.

L'émergence de Go marque l'évolution des langages de programmation vers des solutions adaptées au cloud computing et aux architectures distribuées. Les besoins contemporains en matière de développement logiciel ont conduit à repenser les outils de programmation, créant un langage qui répond aux exigences de l'informatique actuelle tout en conservant une approche élégante et pragmatique.

Julia

Depuis une vingtaine d'années, les chercheurs utilisant du calcul scientifique vivent avec une contrainte qui ralentit leur travail : il faut choisir entre un langage facile à utiliser pour tester des idées et un langage qui tourne vite pour produire des résultats. Quand on veut prototyper, on utilise Python ou MATLAB. Mais dès qu'on veut des calculs sérieux, on reprend tout en C++ ou en Fortran. Cette gymnastique permanente entre deux langages fait perdre du temps et complique les projets sans raison valable.

C'est ce problème que Jeff Bezanson, Alan Edelman, Stefan Karpinski et Viral Shah ont décidé d'attaquer en 2009 au MIT. Ils veulent créer un langage qui réunirait la simplicité des langages dynamiques et la vitesse des langages compilés. Un pari audacieux, car ces deux qualités semblent contradictoires par nature.

La première version de Julia sort en 2013. Dès le début, le langage vise

le calcul scientifique avec une architecture pensée pour les opérations sur les tableaux multidimensionnels, à la manière de Fortran. Le moteur repose sur LLVM, qui génère du code machine adapté aux différentes architectures matérielles. Julia compile le code à la volée, au moment où il s'exécute. Cette compilation *just-in-time* réconcilie deux mondes qu'on croyait séparés : la réactivité d'un langage interprété et la rapidité d'un programme compilé.

L'originalité de Julia tient surtout à son système de dispatch multiple. Contrairement aux langages objets classiques où une méthode dépend du type d'un seul objet, Julia choisit la bonne fonction selon les types de tous les arguments. Ce mécanisme rend le code beaucoup plus flexible et facilite la composition de bibliothèques qui n'ont pas été conçues ensemble. Le langage intègre aussi des macros puissantes, du calcul parallèle natif, et dialogue sans friction avec du code C ou Fortran existant.

La communauté se structure. En 2014 a lieu la première JuliaCon, qui se transforme vite un rendez-vous annuel. Trois ans plus tard, le registre compte déjà plus de 1 000 paquets. Des entreprises de poids commencent à s'y intéresser : Aviva et la Federal Reserve Bank of New York côté finance, Pfizer, Moderna et AstraZeneca dans la pharmacologie, ASML pour les semi-conducteurs.

Les performances surprennent. Des benchmarks montrent que certaines fonctions mathématiques écrites en Julia pur tournent plus vite que leurs équivalents optimisés en C ou Fortran. L'implémentation de la fonction inverse de la fonction d'erreur bat celle de MATLAB d'un facteur 3 ou 4, et celle de SciPy de 2 à 3. Cette vitesse vient des techniques de métaprogrammation et de génération de code que Julia exploite naturellement, là où un langage de bas niveau demanderait des contorsions.

Le nom de l'environnement Jupyter rappelle le lien avec Julia (avec Python et R). Ces notebooks, qui mélangent code, résultats, graphiques et texte, collent bien aux besoins de l'enseignement et de la recherche reproductible. Le MIT et d'autres universités l'utilisent maintenant pour leurs cours de calcul numérique.

Julia fonctionne sur diverses architectures : processeurs ARM et GPU. Le gestionnaire de paquets intégré simplifie la gestion des dépendances et garantit la reproductibilité des environnements. Des outils automatisent la création de passerelles vers les bibliothèques C++, ce qui donne accès à un patrimoine logiciel considérable.

En 2022, Julia s'attaque à des projets scientifiques de grande ampleur. La Climate Modeling Alliance s'en sert pour la modélisation climatique, le projet Celeste pour le traitement d'images astronomiques, des équipes de physique des hautes énergies pour l'analyse de données. Ces applications montrent que le langage tient la distance sur des calculs complexes et

massifs, sans sacrifier la lisibilité du code.

Le développement suit une logique open source classique, avec le code sur GitHub. Entre 2009 et 2017, plus de 40 000 commits s'accumulent. Les contributions passent par une revue stricte qui maintient la qualité. La documentation fournie et les tutoriels en ligne aident les nouveaux venus à entrer dans le langage sans trop galérer.

Les choix de départ se révèlent judicieux pour suivre l'évolution du calcul scientifique. L'équipe continue d'améliorer les temps de compilation, d'optimiser automatiquement le code, de supporter les nouvelles architectures. Le fil rouge reste le même : un langage unique qui combine facilité et performance.

Julia montre qu'on peut repenser les langages de programmation pour transformer les pratiques scientifiques. En résolvant élégamment le problème des deux langages, il a changé ce qu'on attend d'un outil de calcul : la productivité sans compromis sur la vitesse.

TypeScript

Lorsque Microsoft lance TypeScript en 2010, le paysage du développement web traverse une période de mutation rapide. JavaScript, jadis relégué aux interactions simples dans les navigateurs, s'impose désormais comme un langage incontournable. Les moteurs d'exécution gagnent en performance, les applications web monopage se multiplient, et Node.js ouvre la voie à JavaScript côté serveur. Cette expansion s'accompagne d'un problème : maintenir de vastes bases de code JavaScript est un véritable casse-tête.

Anders Hejlsberg, l'architecte derrière C#, prend la tête du projet avec une ambition claire. Il ne s'agit pas de remplacer JavaScript, mais de l'enrichir. TypeScript se présente comme un sur-ensemble syntaxique de JavaScript : tout code JavaScript valide reste un code TypeScript valide. L'idée maîtresse consiste à ajouter un système de types statiques facultatifs. Les développeurs peuvent annoter leur code, et ces annotations disparaissent à la compilation. Le JavaScript généré ne garde aucune trace du typage.

Cette décision technique reflète une compréhension fine des besoins réels. Le système de types adopte une approche structurelle plutôt que nominative, ce qui correspond mieux aux habitudes des développeurs JavaScript. Dans cet écosystème, les objets naissent souvent directement, sans passer par des classes formelles. TypeScript épouse cette pratique au lieu de la contrarier.

Un choix de conception mérite qu'on s'y attarde. Les créateurs de TypeScript ont délibérément renoncé à garantir une sûreté totale du typage. Ils ont préféré faciliter l'adoption progressive et maintenir la compatibilité avec les bibliothèques existantes. Cette souplesse autorise les équipes à introduire TypeScript dans leurs projets sans tout réécrire d'un coup.

Steve Ballmer, l'ancien PDG de Microsoft, avait comparé les licences open source à « un cancer qui s'attache à tout ce qu'il touche ». Il dénonçait le manque de responsabilité dans les logiciels libres. Pourtant, l'équipe de TypeScript savait qu'elle ne séduirait jamais la communauté JavaScript sans embrasser l'open source. En 2012, Microsoft publie TypeScript sous licence Apache-2.0, d'abord sur CodePlex, ensuite sur GitHub en 2014.

L'adoption croît progressivement. En 2025, le dépôt GitHub récolte plus de 107 000 étoiles et figure parmi les vingt projets les plus appréciés de la plateforme. Le package NPM enregistre plus de 100 millions de téléchargements hebdomadaires. Plus de 61 000 autres packages en dépendent. Ces chiffres traduisent une confiance établie.

Le langage introduit des concepts structurants : interfaces, classes, modules. Son système d'inférence de type réduit les annotations nécessaires tout en préservant les bénéfices du typage statique. Les outils de développement s'enrichissent d'autocomplétion et de navigation dans le code. Le compilateur, écrit en TypeScript, prouve la capacité du langage à gérer des projets complexes. Il analyse le code source, vérifie les types, et génère du JavaScript standard en préservant la sémantique du programme.

La communauté a bâti un écosystème riche autour de TypeScript. Le référentiel DefinitelyTyped rassemble des milliers de fichiers de définition de types pour les bibliothèques JavaScript traditionnelles. Ces déclarations permettent d'utiliser l'existant avec les avantages du typage statique. Les versions successives apportent des fonctionnalités avancées : types génériques, types conditionnels, types utilitaires. Ces ajouts autorisent l'expression de contraintes complexes et renforcent la sécurité du code. Une nouvelle version sort tous les deux mois, assurant une évolution constante.

TypeScript a instauré des pratiques plus rigoureuses dans l'écosystème JavaScript. Certains concepts ont enrichi JavaScript lui-même, comme le montrent les nouvelles versions d'ECMAScript. Cette réussite témoigne de la transformation de Microsoft vers une approche moins fermée du développement logiciel.

Docker

Il y a une quinzaine d'années, une jeune pousse du nom de dotCloud prenait une décision qui allait chambouler l'industrie informatique. En rendant public le code de son moteur de conteneurisation, l'entreprise offrait au monde Docker. La façon dont on construit, distribue et fait tourner des applications ne serait plus jamais la même. Pourtant, l'histoire des conteneurs ne commence pas avec Docker, loin de là.

Remontons à 1979. UNIX introduisait alors `chroot`, un mécanisme qui permettait de changer le répertoire racine d'un processus. Une brique élémentaire modeste qui posait les bases d'une idée : isoler le stockage des applications les unes des autres. FreeBSD franchit un cap en 2000 avec ses *jails*, fournissant une vraie virtualisation au niveau système. Linux-VServer emboîte le pas l'année suivante, suivi d'OpenVZ en 2005. Toutes ces technologies exploraient une piste similaire : partitionner les ressources système pour cloisonner les applications.

Mais aucune ne décollait vraiment. Trop complexes, trop rigides, elles restaient cantonnées à des usages de niche. LXC débarque en 2008 et pour la première fois, on dispose d'un gestionnaire complet de conteneurs qui s'appuie sur le noyau Linux standard, sans bidouilles ni modifications hasardeuses. La virtualisation légère est enfin accessible, du moins en théorie.

dotCloud utilisait LXC pour faire tourner sa plateforme cloud. Ses ingénieurs avaient bricolé un outil maison en Python, baptisé « dc », qui servait à manipuler les conteneurs LXC. Créer, démarrer, arrêter, exposer des ports réseau : les fonctions de base étaient là. Sauf que l'architecture posait problème. Les accès concurrents aux métadonnées des conteneurs créaient des conflits de verrous. Le système tenait debout, mais la fragilité guettait.

En 2013, Solomon Hykes et son équipe font le grand saut : ils libèrent leur moteur sous le nom de Docker. Docker ne se contente pas de rendre les conteneurs utilisables, il les rend simples. Le format d'image standardisé emballe une application avec tout ce dont elle a besoin. Les couches superposées en lecture seule permettent de partager les parties communes entre images. Le registre Docker Hub transforme le partage d'images en un jeu d'enfant.

L'architecture évolue vite. Docker s'émancipe de LXC dès 2014 en développant `libcontainer`, sa propre bibliothèque pour dialoguer avec le noyau Linux. Plus de dépendance externe, plus de contrôle, plus de stabilité. Les fondations techniques reposent sur les `cgroups` qui limitent les ressources, les `namespaces` qui isolent les processus, et des pilotes de stockage comme AUFS ou OverlayFS qui gèrent les couches d'images avec finesse.

La conception modulaire se précise. Le démon `dockerd` expose l'API et

orchestre le tout. `containerd` prend en charge l'exécution des conteneurs. `runc` standardise leur création. Cette architecture en briques remplace ce qui ne convient pas, et adapte l'outil à des besoins spécifiques.

Le succès de Docker pose une question : comment garantir l'interopérabilité ? La Linux Foundation lance l'Open Container Initiative en 2015 pour établir des standards ouverts. Docker y contribue en donnant les spécifications de son format d'image et `runc` comme référence. Cette année naît aussi la Cloud Native Computing Foundation, qui hébergera Kubernetes, Prometheus et `containerd`. L'écosystème se structure, les standards s'imposent.

En avril 2015, Docker lève 95 millions de dollars lors d'un tour de table Series D mené par Insight Venture Partners. Des noms prestigieux rejoignent l'aventure : Goldman Sachs, Coatue, Northern Trust. Six mois plus tard, 18 millions supplémentaires s'ajoutent. La valorisation dépasse le milliard de dollars. Docker devient une « licorne », tous les poids lourds de la Silicon Valley misent sur cette technologie qui promet de révolutionner le déploiement logiciel. L'entreprise enchaîne les tours de financement jusqu'à atteindre près de 280 millions de dollars levés. En 2017, un nouveau tour rapporte 75 millions de plus, portant la valorisation à 1,3 milliard.

Docker bouleverse les pratiques. Une application mise en conteneur se comporte de la même manière partout : sur la machine du développeur, sur les serveurs de test, en production. Les ressources consommées restent raisonnables, bien loin de la lourdeur des machines virtuelles traditionnelles. La densité sur les serveurs s'améliore mécaniquement.

L'entreprise développe aussi Docker Swarm, son orchestrateur natif. L'idée semble tenir la route : gérer des flottes entières de conteneurs répartis sur plusieurs machines en conservant la simplicité de l'API Docker. Quelques commandes suffisent pour déployer un cluster. Pas de courbe d'apprentissage abrupte, juste une extension naturelle de ce que les développeurs connaissent déjà. En 2016, Docker intègre Swarm directement dans le moteur avec le « Swarm Mode ». C'est dans la boîte, prêt à l'emploi.

Sauf que Google avait libéré Kubernetes en 2014. Fruit d'une décennie d'expérience avec Borg, son système interne d'orchestration, Kubernetes grandit à une vitesse folle. La communauté open source s'enflamme. Les trois grands fournisseurs cloud adoptent Kubernetes : Google bien sûr, mais aussi Amazon avec EKS et Microsoft avec AKS. VMware et Red Hat s'y rallient. En 2017, la bataille est déjà perdue pour Docker Swarm. La direction de Docker l'admet en annonçant que la plateforme prendra en charge Kubernetes. Un aveu d'échec à peine déguisé.

Les tensions montent en interne. Trois directeurs généraux en une seule année. L'entreprise navigue à vue. Elle a deux activités distinctes qui fonc-

tionnent selon des logiques incompatibles : d'un côté Docker Desktop et Docker Hub, prisés des développeurs mais difficiles à monétiser ; de l'autre Docker Enterprise, une suite d'outils pour les grandes entreprises avec orchestration intégrée, contrats juteux, mais une complexité croissante.

En novembre 2019, Mirantis, spécialiste du cloud qui cherchait sa voie après l'essoufflement d'OpenStack, rachète toute l'activité entreprise de Docker. La transaction englobe Docker Enterprise Engine, Docker Trusted Registry, Docker Unified Control Plane, le CLI, et 750 clients grands comptes dont un tiers du Fortune 100. Trois cents employés rejoignent Mirantis. Le montant n'est pas divulgué, mais il est dérisoire comparé aux 280 millions levés. Les observateurs parlent d'une vente pour une bouchée de pain.

Le même jour, Docker annonce 35 millions de dollars de financement de la part de Benchmark Capital et Insight Partners. Scott Johnston, directeur des produits depuis 2014, devient le troisième directeur général de l'année. La société se restructure, se recentre sur les développeurs, garde Docker Desktop et Docker Hub. L'équipe passe de plusieurs centaines de personnes à une soixantaine. Beaucoup y voient la fin d'une époque. Les commentateurs déclarent Docker mort.

Docker et Mirantis conviennent de collaborer sur le développement open source. Mirantis s'engage à supporter Docker Swarm au moins deux ans, mais le message est clair : Kubernetes sera l'orchestrateur principal. L'acquisition des actifs Docker Enterprise permet à Mirantis de se positionner face aux géants comme VMware et Red Hat. Deux ans après le rachat, Mirantis annonce un chiffre d'affaires annuel dépassant 100 millions de dollars, dont la moitié provient des actifs rachetés à Docker. La greffe a pris.

De son côté, Docker renaît de ses cendres. Concentré sur son cœur de métier, l'entreprise retrouve une trajectoire de croissance. Le revenu annuel récurrent grimpe de 6 millions en 2020 à 50 millions en 2021, puis à plus de 135 millions en 2023. En mars 2022, Docker lève 105 millions de dollars lors d'un tour Series C mené par Bain Capital Ventures, avec une valorisation de 2,1 milliards. L'effectif double pour atteindre 150 personnes. En 2021, Docker Business voit le jour, une offre pour les grandes entreprises, et les conditions de licence de Docker Desktop changent. Le pari du recentrement paie.

Ce qui frappe dans l'histoire de Docker, c'est moins l'innovation technique pure que la démocratisation d'une idée. Les conteneurs existaient, mais personne ne les utilisait vraiment. Docker les a rendus accessibles, presque évidents. Cette simplification a ouvert la voie aux architectures microservices, accéléré le mouvement DevOps, transformé la façon dont on pense le déploiement logiciel.

Les composants se sont séparés, donnant aux utilisateurs le choix des

éléments qui leur conviennent. De nouveaux outils apparaissent sans cesse pour optimiser la construction d'images, renforcer la sécurité, orchestrer les conteneurs. L'héritage de Docker tient dans les standards ouverts qu'il a établis, dans les fondations qu'il a posées pour l'industrie, et dans sa capacité à se réinventer après avoir frôlé la disparition. Docker Inc. existe toujours, plus petit mais viable, tandis que la technologie Docker irrigue l'ensemble de l'écosystème cloud moderne.

SHA-3

Fin des années 1970, les fonctions de hachage cryptographique commencent à prendre forme. Ces algorithmes transforment n'importe quelle donnée en une empreinte numérique de taille fixe, pour vérifier l'intégrité des données et créer des signatures électroniques. En 1976, Diffie et Hellman publient leur article fondateur sur la cryptographie à clé publique ; ils y identifient le besoin d'une fonction de hachage à sens unique pour construire des schémas de signature numérique.

Entre 1978 et 1979, Rabin, Yuval et Merkle posent les bases théoriques. Rabin propose une conception basée sur le chiffrement DES avec une sortie de 64 bits. Yuval démontre qu'on peut trouver des collisions pour une fonction de hachage de n bits en temps $2^{n/2}$ grâce au paradoxe des anniversaires. Merkle introduit les exigences de résistance aux collisions, aux pré-images et aux secondes pré-images, qui restent les propriétés fondamentales attendues d'une fonction de hachage cryptographique.

Les années 1990 voient l'émergence de MD4 et MD5, conçus par Ron Rivest, qui seront des standards très utilisés. SHA-1, publié par la NSA en 1995, s'impose comme le successeur de MD5 avec une sortie de 160 bits. SHA-2 arrive en 2001 avec des variantes générant des condensats de 224 à 512 bits. Ces algorithmes reposent sur la construction Merkle-Damgård : le message d'entrée est découpé en blocs de taille fixe auxquels on applique itérativement une fonction de compression.

En 2004, l'équipe de Xiaoyun Wang bouleverse le domaine en perfectionnant la cryptanalyse différentielle jusqu'à rendre triviale la recherche de collisions pour MD5. Ces travaux réduisent aussi considérablement la sécurité de SHA-1. En 2008, une équipe parvient à créer un certificat CA malveillant en exploitant les faiblesses de MD5. La communauté cryptographique s'inquiète de la robustesse des fonctions de hachage existantes. SHA-2 utilise des principes de construction similaires à SHA-1, ce qui pose question sur sa résistance future.

Le NIST lance en 2007 un concours public pour sélectionner un nouvel algorithme qui deviendra SHA-3. Le cahier des charges impose que les candidats supportent des sorties de 224, 256, 384 et 512 bits pour assurer la compatibilité avec SHA-2. Les soumissions doivent être libres de droits et utilisables mondialement sans restriction.

64 candidatures arrivent en 2008, dont 51 sont retenues pour le premier tour. À la mi-2009, environ la moitié des algorithmes sont cassés. Cette hécatombe démontre qu'il est difficile de concevoir une fonction de hachage sûre et efficace. En juillet 2009, le NIST sélectionne 14 candidats pour le second tour : Blake, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD et Skein.

L'algorithme Keccak, développé par Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche, se distingue par son architecture novatrice basée sur la construction éponge. Contrairement à la construction Merkle-Damgård, cette approche absorbe le message bloc par bloc avec une partie appelée « taux » de 1 152 bits, tandis que les 448 bits restants servent de paramètre de sécurité. L'état interne forme un cube de $5\times5\times64$ bits totalisant 1 600 bits.

En octobre 2012, le NIST désigne Keccak comme vainqueur du concours SHA-3. La décision s'appuie sur ses excellentes performances matérielles, sa marge de sécurité confortable et son architecture différente des autres SHA. La standardisation aboutit en août 2015 avec la publication du FIPS 202. SHA-3 comprend quatre fonctions de hachage classiques (SHA3-224, SHA3-256, SHA3-384 et SHA3-512) ainsi que deux fonctions à sortie extensible SHAKE128 et SHAKE256.

Les premières mises en œuvre de SHA-3 révèlent des caractéristiques intéressantes. Sur FPGA, l'algorithme nécessite plus de ressources matérielles que SHA-1 mais offre une meilleure résistance cryptographique. Les tests de performance indiquent que SHA-3 est environ 5,5 fois plus lent que SHA-1 du fait de sa complexité accrue. Pour le cassage de mots de passe par force brute, un mot de 6 caractères requiert quelques minutes avec SHA-1 contre plus d'une heure avec SHA-3.

L'introduction de SHA-3 marque une évolution dans l'histoire des fonctions de hachage cryptographique. Sa construction éponge rompt avec l'approche traditionnelle et inspire de nouveaux modes d'utilisation. Les fonctions à sortie extensible SHAKE apportent une innovation : on peut adapter la taille du condensat aux besoins spécifiques des applications. SHA-3 complète la famille SHA-2 en apportant une diversité architecturale qui renforce la résilience globale face aux futures avancées cryptanalytiques.

À ce jour, SHA-3 n'a pas subi d'attaque significative remettant en cause sa sécurité. Les recherches continuent pour explorer les possibilités offertes

par la construction éponge, notamment pour la conception de primitives cryptographiques authentifiées. Le succès du concours SHA-3 démontre l'efficacité d'une approche ouverte et collaborative pour développer des standards cryptographiques robustes. Cette méthodologie sert désormais de modèle pour d'autres initiatives de standardisation en cryptographie.

Spark

En 2009, un groupe de chercheurs de l'université de Californie à Berkeley se lance dans le développement de Spark. Leur objectif est de créer un système de calcul distribué capable de dépasser les limites de MapReduce. Le problème était devenu évident : les applications modernes de traitement de données devaient sans cesse réutiliser les mêmes ensembles à travers plusieurs opérations parallèles, que ce soit pour l'apprentissage automatique ou l'analyse interactive.

L'idée centrale du projet tient en une abstraction baptisée RDD, pour *Resilient Distributed Datasets*. Il s'agit de collections d'objets répartis sur plusieurs machines, accessibles uniquement en lecture, mais capables de se reconstruire si une partition venait à disparaître. Cette approche apporte une tolérance aux pannes sans dupliquer les données, ce qui résout élégamment les questions de fiabilité dans les environnements distribués.

La première version publique arrive en 2010. Spark se démarque par sa capacité à garder les données en mémoire entre les opérations, là où les systèmes traditionnels réécrivaient systématiquement tout sur disque. Pour les algorithmes itératifs, ceux qui traitent les mêmes données encore et encore, le gain de performance est spectaculaire.

Le choix de Scala comme langage de programmation s'avère déterminant. Ce langage statiquement typé, qui s'exécute sur la machine virtuelle Java, offre une expressivité qui simplifie l'écriture d'applications distribuées. Les développeurs manipulent les RDD avec des constructions fonctionnelles, rendant le code plus lisible.

Le projet rejoint l'écosystème de l'Apache Software Foundation en 2013, renforçant ainsi son adoption. Les fonctionnalités se multiplient. SparkSQL voit le jour pour exécuter des requêtes relationnelles sur les données distribuées. Spark Streaming arrive pour traiter les flux en temps réel.

La conception modulaire du système autorise l'ajout progressif de bibliothèques spécialisées. MLlib propose des algorithmes d'apprentissage automatique adaptés aux environnements distribués. GraphX apporte des outils

pour traiter les graphes à grande échelle. Ces extensions enrichissent l'écosystème tout en préservant la cohérence de l'interface.

L'architecture intègre un optimiseur nommé Catalyst, introduit en 2014. Ce composant analyse et améliore les requêtes SQL ainsi que les transformations de données. Il exploite les caractéristiques de Scala pour générer du code performant, fusionner les opérations et réduire les transferts entre nœuds du cluster.

L'interface de programmation unifiée constitue une avancée. Les développeurs combinent différents types de traitements dans une même application : SQL, streaming, apprentissage automatique. Plus besoin de jongler entre différents systèmes spécialisés. Cette approche allège la complexité des pipelines de données.

La communauté connaît une croissance exceptionnelle. En 2015, plus de 1 000 contributeurs participent au développement, faisant de Spark l'une des initiatives open source les plus dynamiques dans le traitement de données. Des entreprises comme Databricks, fondée par les créateurs originaux, bâtissent des services commerciaux sur cette technologie.

Les performances impressionnent. Pour les algorithmes itératifs, Spark peut être jusqu'à 100 fois plus rapide que Hadoop MapReduce grâce au maintien des données en mémoire. Dans le domaine des requêtes SQL, il rivalise avec des systèmes spécialisés comme Impala, tout en donnant davantage de flexibilité.

Les banques l'utilisent pour l'analyse de risques, les sites de commerce électronique pour leurs systèmes de recommandation, les centres de recherche pour traiter leurs données expérimentales. Cette diversité d'usages témoigne de la polyvalence du système.

En 2016, Spark devient le projet le plus actif de la fondation Apache. Certaines organisations déploient des clusters de plus de 8 000 nœuds. L'introduction des DataFrames, inspirés de R et Python, simplifie l'utilisation pour les data scientists qui manipulent des données tabulaires.

La conception de Spark influence l'évolution des systèmes distribués. Le modèle de programmation basé sur les transformations fonctionnelles et la gestion efficace de la mémoire inspirent de nombreux projets ultérieurs. Son approche unifiée pour différents types de traitements est une référence dans l'industrie.

Le développement se poursuit en intégrant les avancées en matière d'optimisation et d'accessibilité. Le projet maintient son orientation vers la simplicité d'usage tout en améliorant ses capacités techniques. La documentation abondante et les outils d'apprentissage forment une nouvelle génération de développeurs aux principes du traitement distribué.

En proposant une solution aux limites des approches existantes, tout en gardant un équilibre entre puissance et facilité d'usage, ce projet transforme la manière dont les organisations traitent leurs données à grande échelle. Son évolution continue reflète la nature mouvante du domaine du traitement de données distribuées.

Kubernetes

En 2006, Google se trouvait face à une équation difficile. L'entreprise proposait gratuitement Gmail, YouTube et son moteur de recherche à des millions d'utilisateurs. Comment gérer une infrastructure colossale sans que les coûts s'envolent ? Les ingénieurs ont alors développé Borg, un système interne capable de faire tourner des milliers d'applications sur leur parc de machines. Le nom rendait hommage à une race d'extraterrestres de Star Trek, ces créatures reliées par une conscience collective. L'analogie collait bien au projet. Borg optimisait l'usage des ressources matérielles en orchestrant l'exécution d'applications dans des conteneurs, une technologie qui commençait à peine à émerger. Google garda ce trésor secret pendant des années, conscient de l'avantage technique qu'il représentait.

En 2013, trois ingénieurs de l'entreprise – Craig McLuckie, Joe Beda et Brendan Burns – ont eu une intuition. Docker venait de rendre les conteneurs populaires auprès des développeurs, mais la technologie restait limitée. Elle ne savait gérer des conteneurs que sur une seule machine. L'expérience acquise avec Borg ouvrait la voie à quelque chose de plus ambitieux. Pourquoi ne pas créer un système capable d'orchestrer des conteneurs à l'échelle d'un centre de données entier ?

Ils ont baptisé leur idée « Project Seven », référence au personnage Seven of Nine de Star Trek, une ancienne drone Borg. Ce clin d'œil établissait un lien avec le système originel tout en marquant une rupture vers plus d'ouverture. Car le vrai pari résidait là. Rendre ce projet open source représentait un risque. Urs Hölzle, directeur technique de Google, voyait dans cette expertise en gestion de conteneurs un avantage concurrentiel. Mais l'entreprise a compris qu'une adoption massive nécessitait une communauté active de développeurs externes.

Le projet prit finalement le nom de Kubernetes, du grec ancien kubernētēs qui désigne le pilote ou le timonier d'un navire. Ce choix s'inscrivait dans la continuité maritime de Docker. Le système devait piloter des conteneurs, comme un timonier guide son navire. Le logo arbore sept branches, trace discrète du nom initial « Project Seven ».

Les concepteurs ont bâti Kubernetes en s'appuyant sur les leçons tirées de Borg et de son successeur Omega. L'architecture repose sur des clusters, avec d'un côté des nœuds de travail qui exécutent les applications conteneurisées, de l'autre des nœuds de contrôle qui gèrent l'orchestration. Le système utilise des étiquettes (labels) pour organiser et gérer les conteneurs de manière souple, abandonnant la numérotation séquentielle rigide de Borg.

La version 1.0 fut dévoilée en 2015 lors de la conférence OSCON. Dès le lancement, Microsoft, Red Hat et IBM ont apporté leur soutien. Google a confié le projet à la Cloud Native Computing Foundation (CNCF), une organisation à but non lucratif hébergée par la Linux Foundation. Ce transfert garantissait la neutralité du projet et son développement collaboratif.

Les fonctionnalités techniques de Kubernetes répondent aux exigences des applications modernes. Réplication automatique des conteneurs, équilibrage de charge, découverte de services, surveillance de la santé des applications, réparation automatique en cas de panne. Le système regroupe plusieurs machines pour distribuer efficacement la charge de travail.

L'approche de Kubernetes en matière d'API marque une évolution notable par rapport à Borg. Chaque objet possède trois champs fondamentaux : les métadonnées, la spécification de l'état désiré et le statut actuel. Cette standardisation a simplifié le développement d'outils et d'extensions par la communauté.

La gestion des « pods » apporte une dimension nouvelle. Un pod peut contenir plusieurs conteneurs partageant des ressources. Cette conception aide à décomposer les applications complexes en composants plus petits et mieux isolés, sans perdre la coordination entre eux.

L'ascension de Kubernetes s'inscrit dans une transformation plus vaste du développement logiciel. Le système a accompagné l'essor du cloud computing et des architectures en microservices. Les entreprises peuvent désormais déployer leurs applications de manière cohérente, sur leurs propres serveurs comme sur AWS, Microsoft Azure ou Google Cloud.

L'impact sur l'industrie informatique s'est révélé massif. En 2020, Kubernetes figurait parmi les dix projets open source les plus populaires sur GitHub. Selon la CNCF, il est devenu le deuxième plus grand projet open source après Linux. 71% des entreprises du Fortune 100 l'utilisent.

Cette histoire montre comment une technologie développée pour résoudre des problèmes internes à Google s'est muée en standard industriel grâce à l'open source. Elle souligne le rôle des communautés de développeurs dans l'adoption des technologies à grande échelle. Kubernetes a transformé la manière dont les entreprises développent, déploient et gèrent leurs applications, leur donnant l'agilité nécessaire pour innover vite sans sacrifier la

stabilité de leurs services.

Swift

En 2014, le marché des applications mobiles connaît une croissance spectaculaire. Les ventes d'appareils mobiles approchent le milliard d'unités, le double des ordinateurs personnels. Apple saisit ce moment pour annoncer, en juin, la sortie de Swift, un nouveau langage de programmation qui mêle les approches impérative, orientée objet et fonctionnelle.

Depuis 1996, l'entreprise s'appuie sur Objective-C pour développer ses applications. Mais le langage a vieilli. Swift arrive comme une réponse à ce besoin de renouvellement : simplifier le code sans sacrifier la puissance, rendre la programmation plus fluide, plus intuitive. Les concepteurs ont cherché à créer quelque chose d'expressif, où l'interaction avec le code est presque naturelle.

Le succès est immédiat. En moins d'un an et demi, Swift rejoint le club des vingt langages les plus utilisés dans le monde. Cette ascension s'explique d'abord par l'immense base installée d'iOS : plus de 700 millions d'appareils vendus. La domination d'Apple sur son segment et l'annonce du remplacement progressif d'Objective-C ont fait le reste. Les développeurs n'avaient guère le choix, mais le langage s'est avéré suffisamment séduisant pour emporter l'adhésion.

Swift apporte son lot d'innovations. Les types optionnels, par exemple, constituent une vraie avancée. Cette fonctionnalité, habituelle dans les langages fonctionnels mais rare ailleurs, force le développeur à gérer explicitement les valeurs nulles. Moins d'erreurs, plus de clarté. La version 2.0 introduit un nouveau système pour gérer les erreurs avec do-try-catch, abandonnant l'ancienne approche par NSError héritée d'Objective-C.

Les développeurs chevronnés s'adaptent sans trop de peine. Certaines caractéristiques rappellent Objective-C, comme ces paramètres nommés qui rendent le code lisible presque comme une phrase en anglais. Pourtant, une analyse des questions posées sur StackOverflow révèle des zones d'ombre. Près de 17,5% des interrogations portent sur des notions fondamentales : manipulation des types, gestion des tableaux. Le langage n'est pas si simple qu'il n'y paraît.

Les pratiques de développement montrent des usages intéressants. Le chaînage optionnel se popularise. Cette technique élégante évite les erreurs de déréférencement tout en gardant le code compact. Au lieu d'empiler

les conditions, on enchaîne les opérations avec un point d'interrogation. Le programme est plus sûr et plus agréable à lire.

Les premières années ne sont pas exemptes de critiques. Le compilateur, en particulier, agace. Il manque de stabilité, ses messages d'erreur restent obscurs. Xcode, l'environnement de développement, connaît des bugs qui ralentissent les équipes. Ces problèmes techniques contrastent avec les promesses d'une expérience de développement améliorée.

L'intégration avec Objective-C est un enjeu majeur. Apple a prévu des mécanismes pour faire cohabiter les deux langages dans une même application. Cette compatibilité est vitale : impossible de tout réécrire d'un coup, et les bibliothèques existantes sont trop nombreuses pour s'en passer. La transition se fait par étapes, projet après projet.

Face aux limites du langage, les développeurs bricolent leurs solutions. Le système de gestion des erreurs, par exemple, ne convient pas toujours au code asynchrone. Certains adoptent alors des énumérations de résultats, contournant les restrictions. Cette créativité témoigne d'une communauté qui s'approprie l'outil, qui le façonne à ses besoins.

L'App Store génère des revenus considérables. En janvier 2015, les développeurs ont gagné 25 milliards de dollars, davantage que sur les autres plateformes. Cette manne financière attire les entreprises comme les indépendants. Swift est un investissement rentable, pas seulement une question de préférence technique.

Le langage évolue au fil des retours d'expérience. Chaque nouvelle version corrige des défauts, optimise des fonctionnalités. La documentation s'étoffe, les tutoriels se multiplient et l'apprentissage est moins ardu pour les nouveaux arrivants. Apple écoute, ajuste, améliore.

Swift représente un virage dans la stratégie d'Apple. L'entreprise ne se contente pas de suivre les tendances ; elle forge un langage moderne qui intègre les meilleures pratiques actuelles. Cette démarche préserve la stabilité de l'écosystème tout en le propulsant vers l'avenir. Pas de rupture brutale, mais une transformation maîtrisée.

Il est délicat d'introduire une nouvelle technologie de programmation. Il faut innover sans perdre les acquis, séduire sans imposer, renouveler sans casser. Swift y est parvenu, s'imposant comme un acteur incontournable du développement mobile. Les années suivantes confirmeront cette position, faisant du langage bien plus qu'un simple remplaçant d'Objective-C.

Kafka

En 2008, chez LinkedIn, Jay Kreps se retrouve face à un défi que personne n'avait vraiment anticipé. Le réseau social professionnel génère des millions de messages chaque jour, et les systèmes traditionnels peinent à suivre le rythme. Les données transactionnelles classiques – profils, relations entre utilisateurs – restent gérables. Mais c'est surtout la masse d'informations comportementales qui pose problème : consultations de profils, recherches par mots-clés, interactions diverses. Ces traces numériques, indispensables pour comprendre les attentes des membres, submergent littéralement l'infrastructure existante.

Kreps décide alors de développer une solution nouvelle. Il s'associe en 2010 avec Neha Narkhede et Jun Rao pour finaliser ce qui deviendra Apache Kafka. Le nom fait référence à l'écrivain Franz Kafka, un clin d'œil à la vocation première du logiciel : gérer efficacement l'écriture de flux de messages. L'équipe conçoit une plateforme distribuée capable de diffuser des événements en temps réel, avec une architecture pensée pour encaisser des volumes considérables.

L'idée qui fait toute la différence tient en une approche contre-intuitive du stockage. Là où les solutions classiques gardent les messages en mémoire vive, Kafka les inscrit directement sur disque dur, de manière séquentielle. Cette méthode tire parti de l'optimisation naturelle des opérations d'entrée-sortie linéaires sur les disques durs, donnant des débits impressionnants sans sacrifier la fiabilité.

LinkedIn publie Kafka en open source en 2011, sous l'égide de la fondation Apache. Les géants de la Silicon Valley s'y intéressent : Twitter, Netflix et Yahoo adoptent la technologie, bientôt suivis par des entreprises plus traditionnelles comme Goldman Sachs, Walmart ou Cerner. Chez LinkedIn, les chiffres donnent le vertige : plus de 800 milliards de messages traités quotidiennement, soit plus de 175 téraoctets de données qui transitent sans faillir.

En 2014, les trois créateurs fondent Confluent. Leur objectif : développer l'écosystème autour de Kafka et proposer des solutions d'entreprise basées sur cette technologie. L'impact est immédiat : le taux d'utilisation est multiplié par sept dans les onze mois qui suivent le lancement de la société.

L'architecture repose sur des concepts simples mais efficaces. Les messages s'organisent en « topics », des catégories qui structurent les flux de données. Chaque topic se divise en partitions, répliquées sur différents serveurs pour garantir disponibilité et tolérance aux pannes. Cette organisation distribue la charge et fait évoluer le système horizontalement, sans

limite théorique.

Le développement se poursuit avec des fonctionnalités majeures. Kafka Connect, disponible depuis la version 0.9, simplifie l'intégration avec les autres systèmes grâce à un cadre standardisé pour les connecteurs. Kafka Streams offre une bibliothèque de traitement des flux directement intégrée aux applications, sans infrastructure supplémentaire à déployer.

En 2022 sort KRaft (Kafka Raft), un mécanisme de consensus natif qui remplace la dépendance historique à ZooKeeper. L'architecture se simplifie et se fiabilise. Cette année voit aussi l'introduction du stockage hiérarchisé : une partie des données peut désormais être déportée vers des systèmes de stockage objet comme S3, ce qui réduit sensiblement les coûts d'exploitation.

Kafka a transformé la manière dont les entreprises pensent leurs architectures de traitement de données en temps réel. Le modèle a inspiré de nombreuses alternatives : RedPanda propose une implémentation compatible avec le protocole Kafka, tandis que WarpStream développe une architecture radicalement différente privilégiant le stockage sur S3.

En 2024, Kafka s'est imposé comme une technologie mature. Son adoption continue de croître, portée par l'augmentation constante des besoins en traitement temps réel et l'essor de l'Internet des objets. La communauté qui s'est formée autour du projet contribue régulièrement à son amélioration, assurant une évolution continue.

Le succès tient à une architecture technique solide, une réponse précise à un besoin réel, et à une communauté dynamique soutenue par une entreprise stable. Ce qui n'était au départ qu'un projet interne chez LinkedIn est devenu une pierre angulaire des architectures modernes.

GraphQL

En 2012, les équipes de Facebook butent sur un obstacle technique qui menace les performances de leurs applications mobiles. L'API News Feed, conçue selon les standards REST de l'époque, multiplie les allers-retours entre le client et le serveur. Chaque écran nécessite plusieurs requêtes vers différents points d'accès, et cette fragmentation ralentit l'expérience utilisateur. Sur les réseaux mobiles, où chaque milliseconde compte, la situation est critique. GraphQL naît comme une réponse pragmatique à un problème bien réel.

L'idée de départ tient en une observation simple : pourquoi le client ne

pourrait-il pas demander exactement ce dont il a besoin, ni plus ni moins, en une seule fois ? Les architectures REST imposent une structure rigide où chaque *endpoint* retourne un ensemble prédéfini de données. Le client récupère souvent trop d'informations inutiles, ou pas assez, l'obligeant à enchaîner les appels. L'équipe de Facebook inverse la logique : au lieu que le serveur dicte ce qu'il envoie, c'est le client qui spécifie ce qu'il veut recevoir.

Pendant trois ans, GraphQL reste confiné aux serveurs de Facebook. Les ingénieurs affinent le concept, testent différentes approches, corrigent les erreurs. Le langage prend forme autour d'un schéma qui décrit les relations entre les données comme un graphe. La syntaxe s'inspire de JSON, familière aux développeurs web. En 2015, Facebook décide de publier la spécification et une implémentation de référence. La communauté découvre alors une manière radicalement différente d'interroger les API.

Le schéma constitue la pierre angulaire de GraphQL. Il définit les types d'objets disponibles, leurs champs, et comment ils se connectent entre eux. Avec ce contrat explicite entre le client et le serveur, les développeurs frontend savent exactement quelles données ils peuvent demander, les développeurs backend comprennent ce qu'ils doivent fournir. Le système de types fort attrape les erreurs avant l'exécution des requêtes, un filet de sécurité appréciable en production.

GitHub figure parmi les premiers adoptants majeurs. L'entreprise reconstruit son API publique avec GraphQL, suivie par Pinterest et Coursera. Twitter emprunte un chemin progressif : TweetDeck et Twitter Lite servent de terrain d'expérimentation avant le déploiement sur les applications Android et iOS. Chaque organisation adapte la technologie à ses contraintes, forge ses propres pratiques, contribue de nouveaux outils.

Le cas de Twitter illustre bien les ajustements nécessaires. L'infrastructure repose sur des microservices communiquant via Thrift, un système de sérialisation développé par Facebook. Intégrer GraphQL demande une refonte partielle : une couche frontale intercepte les requêtes GraphQL, les valide, gère l'authentification. Les services backend conservent leur logique métier, mais exposent leurs capacités d'une nouvelle manière. Cette architecture hybride préserve les investissements existants tout en exploitant les avantages de GraphQL.

Coursera rencontre d'autres difficultés. La première tentative de migration échoue face aux problèmes de synchronisation entre le schéma GraphQL et les données réelles. L'équipe trouve une solution astucieuse : un middleware traduit à la volée les API REST existantes en GraphQL. Les clients profitent de la nouvelle interface pendant que le backend évolue progressivement. Cette approche pragmatique démontre qu'il n'existe pas une seule bonne manière d'adopter GraphQL.

Les bénéfices techniques ne tardent pas. Fini le sous-chargement de données qui oblige à multiplier les requêtes. Fini le sur-chargement qui gaspille de la bande passante en transférant des informations superflues. Une seule requête bien formulée récupère exactement ce qui est nécessaire, en impliquant plusieurs ressources liées si nécessaire. Les applications mobiles respirent mieux, les pages web se chargent plus vite.

Le système de types introspection ouvre des possibilités inattendues. Les outils de développement interrogent le schéma pour fournir de l'autocomplétion, de la documentation dynamique, de la validation en temps réel. Les IDE se transforment en assistants efficaces qui guident le développeur au lieu de le laisser naviguer dans une documentation PDF obsolète.

Progressivement, la communauté enrichit le langage. Pour réutiliser des morceaux de requêtes, les fragments émergent comme une solution élégante. Les directives offrent un mécanisme souple pour adapter le comportement sans modifier le schéma. Les abonnements répondent au besoin croissant de mises à jour en temps réel, transformant GraphQL d'un système de requête-réponse en un canal de communication bidirectionnel.

La sécurité représente un défi particulier. La flexibilité des requêtes est une arme à double tranchant : un client malveillant pourrait formuler des demandes extrêmement complexes pour surcharger le serveur. La communauté développe des garde-fous : limitation de la profondeur des requêtes, analyse de la complexité avant exécution, contrôle granulaire des permissions. Ces mécanismes permettent d'exploiter la puissance de GraphQL sans exposer l'infrastructure à des abus.

Gartner prévoit que plus de 60% des entreprises utiliseront GraphQL en production d'ici 2027, contre moins de 30% en 2024. Cette progression témoigne de la maturité atteinte par la technologie. Apollo, Hasura, StepZen et d'autres acteurs proposent des solutions qui simplifient le déploiement et la gestion des serveurs GraphQL. L'écosystème s'étoffe, les bonnes pratiques se cristallisent, les pièges sont mieux identifiés.

Au-delà du simple remplacement de REST, GraphQL modifie la façon de penser les interactions client-serveur. Le client ne subit plus les décisions du backend, il exprime ses besoins. Cette inversion change la dynamique entre les équipes frontend et backend, encourage une meilleure collaboration, clarifie les responsabilités. Le schéma partagé est le langage commun qui aligne les efforts de chacun.

Facebook ne cherchait pas à révolutionner le web, juste à accélérer son application mobile. La solution trouvée s'est révélée applicable bien au-delà du contexte initial. En publiant la spécification et en animant une communauté active, Facebook a transformé une solution interne en un standard de l'industrie. GraphQL rejoint ainsi la liste des technologies nées d'un besoin

particulier qui ont fini par redéfinir les pratiques de toute une profession.

TensorFlow

Google met en route le projet Brain en 2011, une initiative destinée à explorer les réseaux de neurones profonds dans un cadre industriel. L'objectif tient en quelques mots : bâtir des systèmes d'intelligence artificielle exploitables, d'abord pour la recherche, puis dans les produits de la société. Ce travail débouche sur DistBelief, un premier système d'apprentissage automatique qui offre aux chercheurs un terrain d'expérimentation pour tester des architectures neuronales complexes.

De 2011 à 2015, DistBelief est le socle d'une multitude d'avancées. L'apprentissage non supervisé, la représentation du langage, la classification d'images, la détection d'objets, la reconnaissance vidéo et vocale : les domaines d'application se multiplient. Plus d'une cinquantaine d'équipes chez Google et dans d'autres filiales d'Alphabet déploient des réseaux neuronaux fondés sur cette technologie. Les services qui en bénéficient couvrent un large spectre : moteur de recherche, publicité en ligne, reconnaissance vocale, Google Photos, Maps, StreetView, Translate, YouTube.

Fort de cet apprentissage, Google conçoit TensorFlow comme héritier de DistBelief. La plateforme sort en novembre 2015 sous licence Apache 2.0 avec une architecture entièrement repensée qui change la donne dans la conception des systèmes d'apprentissage. Sa souplesse autorise l'expression d'algorithmes très variés sous forme de graphes de calcul, le tout avec des performances accrues tant pour l'entraînement que pour le déploiement des modèles.

L'originalité de TensorFlow tient à sa capacité de fonctionner sur des systèmes hétérogènes distribués. Un même modèle s'exécute indifféremment sur un téléphone mobile ou dans un centre de calcul équipé de milliers de processeurs graphiques. Cette adaptabilité simplifie drastiquement l'usage pratique des systèmes d'apprentissage, en supprimant la nécessité de créer des versions spécialisées pour chaque environnement.

TensorFlow repose sur une représentation des calculs en graphes orientés. Les nœuds incarnent des opérations mathématiques, les arêtes transportent des tenseurs, ces tableaux multidimensionnels typés. Cette abstraction convient naturellement aux algorithmes d'apprentissage, qui manipulent matrices et vecteurs en permanence. Le système gère automatiquement la répartition des calculs sur les différentes unités de traitement disponibles.

L'architecture s'articule autour de plusieurs composants. Le cœur du système définit des opérations de base : calculs mathématiques élémentaires, manipulation de tableaux, algèbre linéaire, briques pour réseaux de neurones. Ces opérations bénéficient d'implémentations optimisées selon les types de processeurs. Les variables maintiennent l'état du modèle d'une exécution à l'autre. Des files d'attente assurent synchronisation et transfert efficace des données.

Les développeurs accèdent à TensorFlow via des interfaces Python et C++. Ils construisent leurs modèles en assemblant les opérations de base dans un graphe de calcul. Le système prend alors en charge l'exécution distribuée, la gestion mémoire, les optimisations. Des outils comme TensorBoard aident à comprendre et déboguer les modèles en visualisant leur structure et l'évolution des métriques d'apprentissage.

TensorFlow intègre le calcul symbolique des gradients. Cette fonctionnalité automatise une tâche délicate : le calcul des dérivées qu'exigent les algorithmes d'optimisation tels la descente de gradient stochastique. Le système analyse le graphe de calcul et génère automatiquement les opérations correspondant aux dérivées partielles, en appliquant la règle de dérivation en chaîne.

De nombreuses optimisations améliorent performances et utilisation des ressources. L'élimination des sous-expressions communes supprime les calculs redondants. Le placement judicieux des opérations sur les différents processeurs limite les communications. La compression des données échangées entre machines réduit la bande passante nécessaire. Ces optimisations sont critiques lors de l'entraînement de modèles à grande échelle.

Le déploiement dans les produits Google démontre l'efficacité de TensorFlow. Le système de reconnaissance vocale voit son taux d'erreur chuter de 13,25% à 2,5% en cinq ans. Les performances en classification d'images sur ImageNet passent de 63% à 91% de précision. Ces bonds spectaculaires résultent de la combinaison d'architectures plus élaborées et de la capacité à traiter efficacement d'énormes volumes de données.

Google développe parallèlement une gamme de processeurs spécialisés : les TPU (Tensor Processing Unit). Ces circuits intégrés, taillés pour les opérations d'algèbre linéaire à précision réduite, offrent des gains substantiels en performance et efficacité énergétique face aux CPU et GPU classiques. Les générations successives de TPU forment des systèmes massivement parallèles, atteignant plusieurs exaflops pour l'apprentissage automatique.

La publication en open source crée une communauté active. Chercheurs et développeurs s'emparent de TensorFlow dans des domaines très divers : vision par ordinateur, traitement du langage naturel, robotique, recherche d'information, extraction d'informations géographiques, découverte de mé-

dicaments. Les contributeurs enrichissent sans cesse la plateforme.

L'architecture flexible et les performances élevées font de TensorFlow une référence pour la recherche et le déploiement de modèles complexes. La disponibilité d'outils matures et d'une documentation fournie démocratise l'accès aux techniques d'apprentissage profond auprès d'un large public de développeurs.

L'évolution de TensorFlow reflète la progression rapide de l'intelligence artificielle. Des premiers prototypes de recherche aux systèmes de production à grande échelle, la plateforme accompagne le développement d'applications toujours plus sophistiquées. Son architecture distribuée et ses optimisations tirent pleinement parti des ressources de calcul modernes pour entraîner des modèles de plus en plus puissants.

Au fil des années, TensorFlow s'enrichit d'extensions variées. Support de nouveaux types de modèles, amélioration des performances, ajout d'outils de développement et de débogage renforcent progressivement ses capacités. La plateforme continue d'évoluer pour répondre aux exigences croissantes de l'apprentissage automatique et de l'intelligence artificielle.

Accelerated Mobile Pages

En octobre 2015, Google lance Accelerated Mobile Pages, une technologie conçue pour accélérer la navigation sur mobile. Le constat est sans appel : 53% des internautes abandonnent une page qui tarde plus de trois secondes à s'afficher. Face à ce problème, Google cherche une solution radicale.

Le projet naît des discussions entre Google et les éditeurs européens dans le cadre de la Digital News Initiative. L'idée initiale vise les sites d'information, premiers concernés par la frustration des lecteurs sur smartphone. La solution technique repose sur trois piliers : une version épurée du HTML, une bibliothèque JavaScript qui optimise le chargement des ressources, et un système de cache pour stocker les contenus. Mais cette architecture impose des contraintes sévères. Les feuilles de style ne peuvent excéder 50 Ko, le JavaScript personnalisé est interdit, et les images doivent utiliser des balises spécifiques comme `<amp-img>` ou `<amp-video>`.

Les premiers résultats dépassent les espérances. Le Washington Post enregistre 23% d'utilisateurs supplémentaires qui reviennent dans la semaine suivant leur première visite. En Europe, la plateforme Plista observe une hausse moyenne de 220% du taux de clics sur les contenus AMP, avec des

pics à 600% pour certains éditeurs. Ces chiffres attirent l'attention. Pinterest, Reddit, Bing et eBay adoptent la technologie. Google l'intègre à sa section Actualités en février 2016, puis l'étend à tous les résultats mobiles six mois plus tard. Un petit éclair vient alors signaler les pages AMP dans les résultats de recherche.

Sur le plan technique, les performances sont spectaculaires. Les pages se chargent en moins d'une seconde et consomment dix fois moins de données que leurs équivalents classiques. Le cache AMP vérifie automatiquement la conformité des pages et adapte les images aux écrans mobiles. Cette infrastructure atteint des niveaux de rapidité inaccessibles aux sites web traditionnels.

L'usage d'AMP s'élargit vite au-delà des médias. En 2018, la technologie investit le commerce électronique, les e-mails interactifs et la création de contenus narratifs avec AMP Stories. WordPress, qui fait tourner plus d'un tiers du web, propose une extension officielle pour simplifier son déploiement.

Mais le succès d'AMP s'accompagne de critiques acerbes. Malgré son code ouvert, beaucoup reprochent à Google de contrôler étroitement cette technologie. Le fait que les pages AMP soient servies depuis les serveurs de Google pose question : à qui appartient vraiment le contenu ? La préférence accordée aux pages AMP dans les résultats de recherche alimente les accusations de pratiques anticoncurrentielles. Ces reproches conduisent Google à annoncer, en 2018, la standardisation des principes d'AMP à travers le Web Platform Incubator Community Group. L'objectif consiste à permettre à n'importe quel site d'obtenir un chargement instantané sans passer par AMP, en respectant simplement certains critères de performance avec des standards plus ouverts comme Web Packaging et Feature Policy.

En 2020, Google poursuit le développement d'AMP tout en ouvrant progressivement le format. AMP E-mail apporte l'interactivité aux messages électroniques, tandis qu'AMP Stories crée des contenus visuels immersifs. Le projet compte désormais plus de 600 millions de pages réparties dans 232 pays et 104 langues.

La technologie a indiscutablement accéléré la navigation sur smartphone, mais elle a aussi renforcé la centralisation autour des grandes plateformes. Son évolution vers des standards plus ouverts témoigne de la recherche d'un équilibre entre performance et décentralisation. Au-delà des aspects techniques, avec AMP la vitesse est devenue un critère déterminant de l'expérience mobile.

Natural Language Processing

Depuis les années 1950, les chercheurs tentent d'apprendre aux ordinateurs à comprendre notre langue. Le traitement automatique du langage naturel, qu'on appelle plus volontiers NLP dans les milieux spécialisés, a longtemps buté sur les mêmes obstacles : comment saisir les subtilités, les doubles sens, le contexte changeant d'une conversation ? Les premières machines fonctionnaient avec des dictionnaires figés et des règles grammaticales rigides. Puis sont venues les approches statistiques dans les années 2000, qui ont apporté leur lot d'améliorations sans vraiment résoudre le problème de fond.

L'année 2017 marque un basculement. Sur le campus de Mountain View, Ashish Vaswani et Jakob Uszkoreit discutent de traduction automatique dans un couloir. Une conversation anodine en apparence, qui va déboucher sur quelque chose d'inattendu. Leur équipe compte huit chercheurs de Google, dont Illia Polosukhin, amateur de science-fiction. C'est justement un film, *Premier Contact*, qui lui donne une idée. Dans cette histoire, les extraterrestres communiquent avec des symboles qui expriment des concepts entiers d'un seul coup, sans suivre notre logique séquentielle habituelle. Pourquoi les machines ne pourraient-elles pas faire pareil avec les phrases ?

Cette réflexion conduit au concept d'auto-attention : au lieu d'analyser les mots les uns après les autres comme on lit un livre, le système embrasse la phrase dans sa globalité. Tous les mots dialoguent entre eux simultanément. Noam Shazeer, qui travaille chez Google depuis 2000 et a créé la fameuse fonction « Vouliez-vous dire ? », entend parler du projet dans les bureaux. Il rejoint l'aventure. Les premiers essais sur des traductions anglais-allemand donnent des résultats encourageants.

L'architecture Transformer qui en résulte est plus rapide et plus précise que les réseaux neuronaux récurrents d'avant, et elle ne se limite pas au texte. Le modèle traite des images, du code informatique, des séquences d'ADN. Cette polyvalence surprend ses créateurs.

L'année suivante, une autre équipe de Google AI publie BERT. L'acronyme cache une idée simple : lire dans les deux sens. Le contexte d'un mot dépend de ce qui vient avant, mais aussi de ce qui suit. BERT performe sur onze tâches différentes de traitement du langage, battant les records précédents.

Curieusement, les huit chercheurs à l'origine des Transformers quittent tous Google dans les années qui suivent. Chacun fonde sa société, exploite la technologie dans sa direction. Aidan Gomez lance Cohere pour les entreprises, Jakob Uszkoreit crée Inceptive et applique les Transformers aux vaccins, Noam Shazeer développe Character.ai où les utilisateurs conçoivent

leurs propres agents conversationnels. Vaswani et Parmar montent Essential.ai. On retrouve là un schéma classique : les grandes structures peinent à transformer leurs découvertes en produits commerciaux. Les chercheurs partent créer ailleurs.

Cette base architecturale nourrit depuis tous les modèles de langage modernes. La capacité à traiter d'énormes volumes de données, à tisser des liens complexes entre les éléments d'une séquence, fait des Transformers un outil incontournable. Les applications se multiplient : traduction plus naturelle, génération de texte cohérent, analyse documentaire, systèmes de questions-réponses performants. Les machines s'approchent des nuances du langage humain, comprennent mieux les relations entre les idées.

Du traitement mot à mot des débuts aux systèmes d'attention contemporains, chaque étape a rapproché les ordinateurs de notre complexité linguistique. Les Transformers ne sont pas qu'une prouesse technique. Ils servent de socle à une cascade d'innovations dans des domaines variés : médecine, recherche scientifique, éducation. La communauté scientifique développe sans cesse de nouvelles variantes.

À mesure que ces modèles gagnent en sophistication, ils modifient notre rapport à la technologie. La frontière entre communication humaine et interaction machine est plus floue. Les développements futurs promettent de nouvelles possibilités, tout en soulevant des questions d'éthique et de responsabilité. Jusqu'où ira cette capacité des systèmes à comprendre et produire du langage naturel ? La réponse s'écrit jour après jour, dans les laboratoires comme dans nos usages quotidiens.

WebAssembly

En 1995, JavaScript voyait le jour comme un simple langage de script. Personne n'imaginait alors qu'il deviendrait le moteur d'applications Web sophistiquées. À l'origine, il servait surtout à valider des formulaires ou animer quelques éléments d'une page. Le gros du travail se faisait sur les serveurs, les navigateurs ne gérant que l'affichage et quelques interactions basiques.

Tout a changé avec l'arrivée de moteurs performants comme V8. JavaScript s'est mis à gérer des programmes de plusieurs milliers de lignes, des applications qui auraient été impensables quelques années plus tôt. Le langage se retrouvait utilisé bien au-delà de ce pour quoi on l'avait conçu.

Cette transformation a poussé les acteurs du Web à chercher des voies

alternatives. Microsoft a tenté sa chance avec ActiveX, qui reposait sur la signature de binaires x86 – une approche fondée sur la confiance plutôt que sur de vraies garanties techniques. Google a ensuite développé Native Client, qui introduisait le *sandboxing* pour du code machine sur le Web. Les performances s'approchaient du natif, mais les développeurs devaient suivre des contraintes strictes dans leur code, comme utiliser des masques binaires avant chaque accès mémoire.

Emscripten a ensuite ouvert une nouvelle possibilité : compiler du C ou du C++ vers JavaScript. Le système créait un environnement d'exécution en JavaScript et traduisait le code vers un sous-ensemble spécialisé, qui a donné naissance à asm.js. On pouvait enfin exécuter du code bas niveau dans un navigateur, mais JavaScript restait un intermédiaire contraignant.

C'est en 2015 que l'histoire prend un tournant inattendu. Des ingénieurs de Google Chrome, Mozilla Firefox, Microsoft Edge et Apple Safari se sont mis autour d'une table pour travailler ensemble. Ces concurrents historiques, qui s'étaient livrés à des « guerres des navigateurs » pendant des années, ont décidé de collaborer. Leur but ? Créer un format de *bytecode* portable qu'ils ont appelé WebAssembly.

Le projet avait des ambitions claires. Il fallait une représentation compacte, une validation rapide, une compilation efficace. Et surtout, une exécution sûre sans pénalité de performance. Les concepteurs ont fait un choix audacieux : formaliser la sémantique dès le départ. WebAssembly devenait ainsi le premier langage industriel conçu avec une sémantique formelle dès sa création.

L'architecture qui en résulte présente quelques particularités. WebAssembly s'appuie sur une machine à pile pour représenter les calculs. Mais grâce à son système de types, les compilateurs peuvent analyser le flux de données entre instructions sans avoir à matérialiser physiquement cette pile. Le contrôle de flux est structuré : pas de `goto` sauvage, mais des constructions classiques via des blocs imbriqués.

La mémoire fonctionne comme un grand tableau d'octets, séparé du code et de la pile d'exécution. Un programme défaillant ou compromis ne peut corrompre que ses propres données. Cette isolation garantit la sécurité : le code s'exécute dans un bac à sable qui l'empêche d'accéder au système hôte sans autorisation.

Le format binaire a été pensé pour le réseau. Les instructions tiennent sur un octet, les nombres entiers utilisent le format LEB128. Les navigateurs peuvent commencer la compilation dès les premiers octets reçus, sans attendre le téléchargement complet du module.

En 2017, les quatre principaux navigateurs intégraient le support de WebAssembly. Chaque équipe a choisi sa propre stratégie d'implémentation.

V8 et SpiderMonkey ont réutilisé leurs compilateurs JIT pour compiler les modules avant leur instanciation. JavaScriptCore a suivi une voie similaire. Chakra a préféré traduire à la volée vers un *bytecode* interne, puis compiler les fonctions les plus sollicitées.

WebAssembly atteint souvent 80 à 90% des performances du code natif, tout en gardant la portabilité et la sécurité du Web. Le code généré est compact : 62,5% de la taille d'asm.js en moyenne, 85,3% de celle du x86-64 natif.

L'influence de WebAssembly déborde du navigateur. Sa conception le rend utilisable dans d'autres contextes. Des implémentations autonomes existent, qui transforment WebAssembly en format portable pour des applications natives. On retrouve là l'ambition originelle de Java avec son « écrire une fois, exécuter partout ».

Le développement continue. Les exceptions à coût nul sont arrivées, les threads aussi, ainsi que les instructions SIMD. L'intégration des ramasse-miettes des navigateurs est en cours, ce qui facilitera la compilation de langages comme Java, C#, Swift ou OCaml vers WebAssembly.

Au-delà de la technique, WebAssembly témoigne de l'accomplissement de la collaboration dans l'écosystème Web. Le projet montre qu'une approche rigoureuse, presque académique, peut cohabiter avec les réalités du développement industriel. Cette technologie a redéfini ce qu'il est possible de faire tourner dans un navigateur, sans sacrifier la sécurité qui est au cœur du Web.

WebAuthn

Depuis que les premiers systèmes informatiques ont vu le jour, la question de l'authentification des utilisateurs n'a cessé de se poser. Le mot de passe s'est imposé très tôt comme la réponse évidente : une chaîne de caractères connue de l'utilisateur, simple à mettre en place, facile à comprendre. Sauf que cette simplicité cache mal ses faiblesses. On vole les mots de passe, on les devine, on les intercepte. En 2024, plus de 80% des piratages tirent encore parti de mots de passe trop faibles ou volés.

Au fil du temps, d'autres méthodes sont apparues. La vérification en deux étapes, les certificats numériques, la biométrie. Chacune apporte sa pierre à l'édifice, mais elles restent parfois lourdes à déployer, ou manquent d'une vraie uniformité. C'est là que le W3C et l'Alliance FIDO ont décidé de changer la donne avec Web Authentication, qu'on appelle plus simplement

WebAuthn.

Google, Mozilla, Microsoft et Yubico ont mis la main à la pâte pour aboutir à cette spécification publiée en 2019. L'idée rompt avec ce qu'on faisait depuis des décennies. Au lieu de partager un secret entre l'utilisateur et le serveur, WebAuthn s'appuie sur la cryptographie asymétrique. Quand vous vous inscrivez sur un service, une paire de clés se crée. La clé privée reste bien au chaud sur votre appareil, tandis que la clé publique part vers le serveur avec un identifiant généré au hasard.

Dans cette architecture, la clé publique stockée côté serveur ne sert à rien pour pirater votre compte. Les attaquants se désintéressent de plus en plus des bases de données. L'authentification est liée au domaine exact du site : une clé créée pour example.com ne marchera jamais sur evil-example.com. Les tentatives d'hameçonnage perdent beaucoup de leur efficacité.

WebAuthn repose sur trois piliers. La robustesse d'abord : l'authentification s'appuie si possible sur un module matériel de sécurité qui garde les clés privées et effectue les calculs cryptographiques. Le périmètre ensuite : chaque paire de clés ne vaut que pour une origine précise, comme les cookies qu'on connaît bien. L'attestation enfin : les authentificateurs peuvent fournir un certificat qui prouve au serveur que la clé publique vient d'une source fiable.

Les navigateurs ont vite adopté cette technologie. Chrome, Firefox, Edge, Safari : tous prennent en charge le standard. Les développeurs ont donc la capacité de l'intégrer de façon homogène. Les systèmes d'exploitation modernes ont suivi avec leurs propres systèmes d'authentification : Windows Hello chez Microsoft, Touch ID chez Apple.

Le processus lui-même se déroule en plusieurs temps. À l'inscription, le serveur transmet des données qui lient l'utilisateur à ses identifiants, avec des informations sur l'organisation et une chaîne aléatoire pour contrer les attaques par rejeu. Le navigateur utilise l'API Web Authentication pour demander la création d'une nouvelle paire de clés.

La réponse contient l'identifiant de l'authentificateur, les données client en JSON, un objet d'attestation. Le serveur vérifie tout ça selon une procédure en 19 points décrite dans la spécification. Une fois validé, il stocke la clé publique et l'identifiant dans sa base.

Pour les connexions suivantes, vous prouvez que vous possédez la clé privée en créant une assertion. Le serveur envoie une nouvelle chaîne aléatoire, l'authentificateur génère une signature numérique que le serveur vérifie avec la clé publique qu'il a gardée. Plus de risque de vol de mot de passe, et l'expérience utilisateur s'en trouve améliorée.

WebAuthn s'inscrit dans FIDO2, un ensemble de technologies qui vise à généraliser l'authentification sans mot de passe. Le besoin se fait sentir :

en 2024, seulement 28% des utilisateurs activent la double authentification sur leurs comptes en ligne.

La spécification continue d'évoluer avec de nouvelles fonctionnalités et des implémentations qui s'affinent. Les développeurs trouvent des exemples de code et des bibliothèques dans différents langages, Python et Go notamment, pour intégrer WebAuthn dans leurs applications.

L'histoire de WebAuthn montre comment les méthodes d'authentification sur Internet se transforment. En remplaçant les mots de passe par des clés cryptographiques, cette norme répond aux exigences de sécurité actuelles tout en simplifiant la vie des utilisateurs. Les grands acteurs du web l'adoptent de plus en plus, et l'on peut parier qu'à l'avenir, s'authentifier deviendra à la fois plus sûr et plus pratique.

Cette évolution s'inscrit dans un mouvement plus large où la cryptographie asymétrique remplace peu à peu les secrets partagés d'autrefois. WebAuthn représente une étape importante dans l'histoire de la sécurité informatique, en apportant une solution standardisée à ce problème d'authentification qui nous poursuit depuis les débuts.

XI

2020

L'ère numérique face aux bouleversements mondiaux

La Covid-19 a frappé le monde sans avertissement au début des années 2020, bouleversant nos habitudes les plus ancrées. Du jour au lendemain, nos écrans sont devenus nos seules fenêtres sur le monde. Nous avons assisté à une mutation accélérée de la société : les salles de réunion ont cédé la place aux visioconférences, les salles de classe aux plateformes d'apprentissage en ligne, les cafés aux discussions virtuelles. Cette bascule brutale vers le tout-numérique a révélé notre dépendance technologique autant que notre capacité d'adaptation.

Les infrastructures numériques, jadis invisibles aux yeux du grand public, se sont soudain retrouvées sous les projecteurs. La question de qui contrôlait ces ressources vitales s'est imposée dans le débat politique. L'Europe a pris conscience avec une acuité nouvelle de sa fragilité face aux géants américains du cloud : Amazon Web Services, Microsoft Azure, Google Cloud. Le vieux continent a réagi en lançant GAIA-X, tentative de riposte à cette hégémonie. Un projet ambitieux, sans doute tardif, mais symbolique d'une volonté d'autonomie stratégique.

Sur l'échiquier mondial, la guerre technologique sino-américaine a pris un tour dramatique. Washington a utilisé l'arme des semi-conducteurs pour freiner l'ascension chinoise, interdisant l'exportation de puces avancées vers

Pékin. En réaction, la Chine a redoublé d'efforts pour créer sa propre filière de production.

L'apparition publique de ChatGPT fin 2022 a marqué l'imaginaire collectif. L'intelligence artificielle et les modèles de langage ont démontré qu'ils savaient écrire, coder, inventer, dialoguer. Cette déferlante a suscité fascination et inquiétude. Les parlements du monde entier se sont emparés du sujet, l'Europe en tête avec son « AI Act ». La question n'était plus de savoir si l'IA allait transformer nos métiers, mais à quelle vitesse et dans quelle mesure.

La cybersécurité est passée du statut de préoccupation technique à celui d'enjeu existentiel. Les attaques par rançongiciel ont ciblé hôpitaux, administrations et industries vitales. L'invasion russe en Ukraine a ajouté une dimension nouvelle au conflit : la guerre se jouait aussi dans le cyberespace, chaque offensive terrestre s'accompagnant de son pendant numérique. Cette réalité a forcé États et entreprises à renforcer leurs défenses invisibles.

Les montagnes russes des cryptomonnaies ont captivé l'attention médiatique. L'effondrement spectaculaire de FTX en 2022 a rappelé la volatilité de ce secteur encore jeune. Les régulateurs, longtemps hésitants, ont commencé à poser des garde-fous. Les banques centrales, initialement sceptiques, ont travaillé sur leurs propres versions numériques de la monnaie. Au-delà de la spéculation, la *blockchain* a trouvé sa place dans la traçabilité industrielle et la certification de documents.

L'empreinte écologique du numérique est apparue au grand jour. Les serveurs qui alimentent notre vie connectée consomment l'équivalent énergétique de pays entiers. Le minage des cryptomonnaies a été pointé du doigt pour son impact environnemental démesuré. Face à cette prise de conscience, les entreprises technologiques ont multiplié les promesses vertes comme les centres de données alimentés aux énergies renouvelables, un matériel plus durable, des logiciels moins gourmands en ressources.

Nos maisons se sont peuplées d'objets connectés : assistants vocaux à l'écoute permanente, montres qui suivent nos constantes vitales, caméras qui surveillent nos allées et venues. Cette intrusion technologique a soulevé des questions sur notre vie privée. Le RGPD européen est devenu une référence mondiale, inspirant des législations similaires aux quatre coins du globe, du Brésil à la Californie.

La galaxie des réseaux sociaux a traversé une période tumultueuse. Le rachat de Twitter par Elon Musk a transformé cette plateforme en laboratoire controversé de la liberté d'expression sans limites. TikTok a conquis la jeunesse mondiale, sous l'œil inquiet des gouvernements occidentaux, soucieux de l'influence chinoise qu'il véhicule. De nouveaux réseaux, plus intimistes, ont émergé en réaction aux géants omniscients.

La 5G a tissé sa toile planétaire, malgré les controverses autour des équipementiers chinois, Huawei en tête. Cette technologie a permis des applications industrielles inédites dont les usines hyperconnectées, les véhicules autonomes et communicants, ou la chirurgie à distance. Pendant ce temps, les laboratoires planchent sur la 6G, promettant des débits vertigineux et des latences infinitésimales.

L'espace, nouvelle frontière du numérique, s'est transformé en terrain de jeu pour milliardaires visionnaires. Les constellations de satellites en orbite basse, comme Starlink de SpaceX, ont commencé à tisser un réseau internet global, portant la connectivité dans les régions les plus isolées. Cette démocratisation spatiale s'est toutefois heurtée aux préoccupations des astronomes, inquiets de voir le ciel nocturne constellé de points lumineux artificiels.

La réalité virtuelle et augmentée a cherché sa voie au-delà du simple gadget. Le concept de « métavers », ressuscité par Mark Zuckerberg, a cristallisé espoirs et scepticisme. Si le grand public est resté tiède, ces technologies ont trouvé leur utilité dans la formation professionnelle, la maintenance industrielle et la conception architecturale. L'immersion numérique a progressé pas à pas, loin des promesses futuristes.

L'informatique quantique a franchi des étapes importantes. Les processeurs expérimentaux ont atteint des stabilités impensables quelques années plus tôt. Si l'ordinateur quantique universel reste un horizon lointain, des applications spécifiques ont vu le jour dans la simulation moléculaire et l'optimisation logistique. Cette technologie de rupture a suscité un intérêt stratégique, chaque puissance craignant de prendre du retard dans cette course au XXI^{e} siècle.

Les robots ont quitté les cages des usines pour travailler aux côtés des humains. Ces « cobots », plus souples et intelligents que leurs prédécesseurs, se sont adaptés aux tâches complexes et variables. Les pénuries de main-d'œuvre post-pandémie ont accéléré cette tendance. Dans les entrepôts, les hôpitaux, les champs, ces nouvelles machines ont modifié la nature du travail, soulevant autant d'espoirs que d'appréhensions.

L'État s'est numérisé à marche forcée. Les administrations ont proposé des services en ligne plus fluides et personnalisés. Les villes se sont dotées de capteurs pour optimiser circulation, éclairage et collecte des déchets. L'identité numérique a progressé, entre commodité administrative et craintes liberticides. Le chemin vers la citoyenneté numérique s'est dessiné, sinueux et semé d'embûches.

L'éducation a gardé les traces durables de la période Covid. Les salles de classe ont intégré de façon durable les logiciels éducatifs. Les cours en ligne ont gagné en maturité pédagogique. L'intelligence artificielle a personnalisé

les parcours d'apprentissage. Cette numérisation accélérée a toutefois creusé les inégalités, entre élèves connectés et déconnectés, entre établissements innovants et délaissés.

La médecine a embrassé les outils numériques. La téléconsultation est entrée dans les habitudes. Les algorithmes d'aide au diagnostic ont fait leurs preuves pour détecter cancers et maladies rares. Les objets connectés médicaux ont permis de suivre à distance les patients chroniques. Cette rupture numérique s'est heurtée aux questions de confidentialité et d'humanité du soin, tout en ouvrant de nouvelles perspectives thérapeutiques.

Une conscience collective des enjeux éthiques du numérique s'est développée progressivement. Les biais algorithmiques, la désinformation organisée, l'économie de l'attention, la surveillance généralisée : ces sujets ont quitté les cercles spécialisés pour entrer dans le débat public. La tech-éthique s'est imposée comme discipline qui influence la conception des services numériques.

La première partie de cette décennie 2020 restera celle où le numérique est devenu à la fois notre plus grande force et notre vulnérabilité majeure. Les crises en cascade ont révélé sa place centrale dans l'organisation de nos sociétés. Plus qu'un simple secteur économique, l'informatique s'est affirmée comme le système nerveux de notre civilisation, portant nos espoirs collectifs autant que nos angoisses. Cette prise de conscience appelle une gouvernance renouvelée, où innovation rime avec responsabilité, où progrès technique s'accorde avec justice sociale et durabilité environnementale.

Apple M1

Quand Apple dévoile la puce M1 en novembre 2020, peu d'observateurs mesurent l'ampleur du bouleversement qui se prépare. Pourtant, cette idée trouve ses racines dans une frustration ancienne de Steve Jobs : la dépendance aux fournisseurs de processeurs. Depuis l'Apple I et son modeste MOS Technology 6502, l'entreprise subissait les choix techniques et les calendriers imposés par d'autres. Cette situation ne convenait guère à une société qui cultivait le contrôle absolu sur ses produits.

Les premières tentatives d'émancipation remontent aux années 1990 avec le Newton, cette tablette avant l'heure qui collaborait déjà avec ARM. L'échec commercial du Newton masque un apprentissage précieux : Apple découvre les subtilités de la conception de processeurs. L'iPod marque ensuite une étape intermédiaire avec son système sur puce Portplayer PP5502,

doté de deux cœurs ARM. Cette architecture se retrouve dans l'iPhone original, Samsung fabriquant encore les composants.

En 2008, Apple rachète P.A. Semiconductor pour 278 millions de dollars, une acquisition qui passe relativement inaperçue à l'époque mais qui s'avère déterminante. Cette société texane apporte l'expertise manquante pour concevoir des processeurs maison. Le premier fruit de cette union naît deux ans plus tard : l'A4 qui équipe l'iPad de première génération, puis l'iPhone 4.

Samsung fabrique ces puces A4, mais Apple bascule vers Taiwan Semiconductor Manufacturing Company (TSMC). Cette migration révèle une stratégie mûrement réfléchie : maîtriser la conception tout en s'appuyant sur le savoir-faire manufacturier taïwanais. Les processeurs série A qui suivent confirment la pertinence de cette approche. Chaque génération surpasse les attentes, délivrant des performances remarquables avec une consommation électrique maîtrisée.

Le secret réside dans l'optimisation. Contrairement aux processeurs généralistes d'Intel ou AMD qui doivent satisfaire mille cas d'usage différents, les puces Apple se concentrent sur un écosystème fermé. Cette spécialisation porte ses fruits : un iPhone avec moins de mémoire vive qu'un smartphone Android rival affiche souvent des performances supérieures. L'harmonie entre le silicium et le système d'exploitation fait la différence.

Cette réussite dans le mobile pousse naturellement Apple vers les ordinateurs. En juin 2020, lors de la WWDC, Tim Cook annonce la transition des Mac vers l'architecture ARM. Six mois plus tard, les premiers MacBook Air et MacBook Pro 13 pouces avec la puce M1 débarquent sur le marché.

La M1 marque une rupture technologique. Gravée en 5 nanomètres chez TSMC, elle rassemble 16 milliards de transistors sur une surface minuscule. Son architecture brise les conventions : CPU, GPU, Neural Engine et mémoire unifiée cohabitent sur le même substrat. Cette intégration élimine les goulets d'étranglement traditionnels entre composants séparés. Les données circulent plus vite, la latence diminue, l'efficacité énergétique s'améliore.

L'architecture ARM de la M1 privilégie la simplicité. Ses instructions RISC (Reduced Instruction Set Computing) contrastent avec la complexité croissante des processeurs x86 d'Intel. Cette philosophie minimaliste, héritée des processeurs mobiles, s'adapte parfaitement aux besoins de l'informatique moderne. Huit cœurs de calcul se répartissent les tâches : quatre optimisés pour la performance pure, quatre autres pour l'efficacité énergétique. Cette répartition intelligente module automatiquement la consommation selon la charge de travail.

Les premiers tests sidèrent l'industrie. Un MacBook Air M1 surpasse un MacBook Pro 16 pouces équipé d'un processeur Intel Core i9 dans de

nombreux benchmarks, tout en fonctionnant sans ventilateur. L'autonomie double, parfois triple par rapport aux modèles précédents. Ces gains spectaculaires redistribuent les cartes du marché des ordinateurs portables.

Apple ne s'arrête pas là. La M1 Pro, avec ses 33,7 milliards de transistors et ses dix cœurs de calcul, cible les professionnels de la création. La M1 Max pousse l'exercice à l'extrême avec 57 milliards de transistors et un processeur graphique 32 cœurs qui rivalise avec des cartes graphiques dédiées. Ces déclinaisons établissent Apple comme un acteur incontournable des processeurs hautes performances.

Cette montée en puissance bouleverse l'écosystème informatique. Intel, habitué à dicter ses lois depuis des décennies, découvre qu'un concurrent peut concevoir des processeurs plus efficaces en partant d'une architecture différente. L'approche verticale d'Apple, qui contrôle l'ensemble de la chaîne depuis le silicium jusqu'aux applications, démontre sa supériorité sur le modèle traditionnel d'assemblage de composants génériques.

L'onde de choc dépasse Apple. Qualcomm accélère le développement de processeurs ARM pour ordinateurs portables Windows. Microsoft adapte son système d'exploitation pour mieux exploiter cette architecture. AMD et Intel repensent leurs stratégies face à cette nouvelle donne concurrentielle.

Au-delà des performances brutes, la M1 incarne une vision différente de l'informatique. Elle privilégie l'intégration harmonieuse des composants plutôt que la course aux spécifications. Cette philosophie s'étend à l'ensemble de l'écosystème Apple, où chaque élément se conçoit en symbiose avec les autres.

Une stratégie patiente peut révolutionner un secteur entier. Partie d'une frustration légitime, Apple a construit méthodiquement son expertise jusqu'à surpasser les leaders historiques. Cette transformation anticipe l'avenir d'une industrie où la frontière entre matériel et logiciel s'estompe au profit d'une approche globale de l'innovation.

Zig

En 2015, Andrew Kelley se lance dans une aventure folle qu'est la création d'un nouveau langage de programmation système. À cette époque, les développeurs disposent d'un arsenal conséquent avec C, C++, Rust ou Go. Pourtant, Kelley identifie un problème récurrent : chaque langage traîne ses propres limitations qui compliquent l'écriture de logiciels vraiment fiables.

Son raisonnement part d'une comparaison frappante. Dans l'aéronautique ou l'industrie des ascenseurs, les systèmes de sécurité se superposent pour rendre les accidents quasi impossibles. Le logiciel, lui, souffre encore d'une réputation de fragilité et d'imprévisibilité. Kelley veut changer cette donne avec Zig.

Le nouveau langage adopte une philosophie radicale : faire moins, mais mieux. Zig supprime le préprocesseur du C, cette mécanique jugée trop complexe et source d'erreurs. Cette décision pourrait paraître régressive, mais Kelley introduit d'autres mécanismes qui résolvent les mêmes problèmes de façon plus élégante.

La gestion de la mémoire dans Zig rompt avec les tendances actuelles. Pas de ramasse-miettes ici, contrairement à Java ou Python. Les interruptions imprévisibles du programme pour libérer la mémoire disparaissent. Zig mise sur un système d'« allocateurs » qui donne aux développeurs un contrôle fin sur la mémoire, sans la lourdeur du C traditionnel. Le développeur sait exactement quand et comment son programme utilise la mémoire.

Les erreurs, ce cauchemar de tout programmeur, bénéficient d'un traitement particulier. Fini les exceptions qui peuvent surgir n'importe où dans le code. Zig intègre les erreurs directement dans le type de retour des fonctions. Le compilateur force littéralement le développeur à traiter chaque erreur possible. Cette contrainte, pesante au premier abord, améliore considérablement la fiabilité du code final.

Le compilateur Zig cache la capacité d'exécution du code pendant la compilation, ce qui ouvre des possibilités de méta-programmation sans recourir aux macros complexes du C++. De nombreuses erreurs sont détectables par le développeur avant l'exécution et optimiser les performances en amont.

L'interopérabilité avec C représente un atout stratégique de Zig. Le langage ne se contente pas d'utiliser les bibliothèques C existantes, il fonctionne lui-même comme un compilateur C performant. Cette dualité facilite l'adoption progressive dans les projets existants. Un développeura la possibilité de commencer par utiliser Zig comme simple compilateur C, puis migrer progressivement vers ses fonctionnalités avancées.

Le système de build intégré remplace des outils comme Make ou CMake par une solution unifiée. Plus besoin de jongler avec différents systèmes selon la plateforme : Zig compile de façon identique partout. Cette uniformisation simplifie drastiquement la vie des développeurs qui travaillent sur plusieurs systèmes.

Quelques projets d'envergure ont adopté Zig. Bun.js, cette alternative à Node.js qui fait parler d'elle, est développé avec ce langage. Son créateur, Jarred Sumner, explique ce choix par la simplicité d'apprentissage de Zig

comparé à C++ ou Rust, tout en conservant des fonctionnalités modernes et une excellente sécurité en développement.

La communauté Zig grandit autour d'une documentation soignée et d'un écosystème qui s'étoffe sur GitHub. La fondation Zig Software, avec Loris Cro comme VP communauté, structure le développement et la promotion du langage. Cette organisation donne une légitimité au projet face aux géants établis.

Les différents modes de compilation de Zig méritent qu'on s'y arrête. En mode debug, le compilateur détecte automatiquement de nombreuses erreurs classiques, aidant le développeur à identifier les problèmes. En mode release, ces vérifications s'effacent pour libérer toute la puissance du processeur. Cette flexibilité répond aux besoins contradictoires du développement : sécurité pendant la création, performance en production.

Le chemin vers la version 1.0 est encore long. Cette échéance volontairement lointaine témoigne de la prudence des créateurs qui préfèrent stabiliser le langage plutôt que de précipiter une sortie. Malgré ce statut de développement, Zig tourne en production dans certains projets, preuve de sa maturité technique.

Zig ne prétend pas remplacer C du jour au lendemain. L'ambition est plus subtile : proposer une évolution moderne qui corrige les défauts historiques du C sans perdre ses qualités. Cette approche progressive rend l'adoption de Zig progressive par les développeurs, en commençant par l'utiliser comme compilateur C avant d'explorer ses innovations.

L'aventure Zig illustre une quête constante dans l'informatique : trouver l'équilibre parfait entre simplicité, sécurité et performance. En éliminant la complexité accidentelle tout en préservant le contrôle sur le matériel, ce langage trace une voie originale dans le paysage encombré des outils de développement système.

Mojo

En 2022, Chris Lattner quitte Apple après avoir conçu Swift et fonde Modular avec Tim Davis, ancien de Google. Ils veulent créer un langage qui réconcilie enfin Python et la performance. Une ambition banale tant les tentatives ont été nombreuses, mais leur approche tranche avec les solutions précédentes.

L'intelligence artificielle explose littéralement. Les GPU tournent à plein régime, les TPU (Tensor Processing Units) se multiplient, et pourtant les

développeurs restent coincés entre deux mondes : d'un côté Python, lisible mais d'une lenteur parfois désespérante, de l'autre C++ ou Rust, rapides mais d'une complexité qui décourage. Cette fracture coûte cher. Les équipes de recherche écrivent leurs prototypes en Python, puis les équipes de production les réécrivent entièrement dans un autre langage. Un gaspillage de temps et d'énergie considérable.

Mojo naît de cette frustration. Mais contrairement aux tentatives passées, Lattner et Davis ne cherchent pas à remplacer Python. Ils veulent l'étendre, le sublimer. Le pari est audacieux : conserver la syntaxe familière de Python tout en intégrant des concepts avancés empruntés à Rust pour la gestion mémoire et à LLVM pour la compilation.

Le langage incorpore dès sa conception des outils d'optimisation sophistiqués. L'optimisation du tuilage, par exemple, réorganise automatiquement les calculs pour exploiter au mieux les caches processeur. Le module d'accord automatique ajuste les paramètres d'exécution selon le matériel disponible. Ces innovations techniques restent transparentes pour le programmeur qui écrit du code d'apparence simple.

La compatibilité avec Python est l'atout maître. NumPy ou Matplotlib sont des bibliothèques qui fonctionnent sans modification. Cette interopérabilité évite de partir de zéro, écueil classique des nouveaux langages. Un développeur peut migrer progressivement, remplaçant seulement les parties critiques par du Mojo optimisé.

Les chiffres parlent d'eux-mêmes. Certains benchmarks montrent des accélérations de 68 000 fois par rapport à Python standard. Ces performances spectaculaires s'expliquent par l'utilisation de MLIR, une technologie développée par Google pour optimiser le code sur différents types de processeurs. Là où Python interprète chaque instruction, Mojo compile et optimise l'ensemble.

La société Modular accompagne le langage d'un écosystème complet. Basalt pour l'apprentissage automatique, Endia pour le calcul scientifique, Lightbug HTTP pour le web. Ces bibliothèques, entièrement écrites en Mojo, démontrent les capacités du langage tout en servant d'exemples aux développeurs.

Le 2 mai 2023, le Mojo Playground ouvre ses portes. Cette plateforme en ligne attire immédiatement l'attention : 120 000 inscriptions en quelques mois, une communauté de 19 000 membres sur Discord et GitHub. L'enthousiasme surprend les créateurs, pendant que les développeurs partagent leurs expérimentations, créent des bibliothèques, proposent des améliorations.

L'architecture technique de Mojo rompt avec certaines conventions Python. Le typage statique inféré améliore les performances sans alourdir

l'écriture. La sémantique par valeur, où les fonctions reçoivent des copies plutôt que des références, évite de nombreux bugs classiques. Ces choix de conception reflètent l'expérience de Lattner avec Swift et sa compréhension des pièges du développement moderne.

L'outillage suit le développement du langage. Une extension Visual Studio Code, un noyau Jupyter pour les notebooks, des fonctionnalités de débogage avancées. Cette suite logicielle facilite l'adoption par les équipes déjà familières avec l'écosystème Python.

L'impact sur l'industrie commence à se dessiner. Des équipes de recherche utilisent Mojo pour leurs projets d'intelligence artificielle, éliminant la rupture entre prototype et production. Les algorithmes optimisés réduisent la consommation énergétique des centres de données, un enjeu devenu critique avec l'explosion des modèles de langage.

La communauté produit ses premiers projets remarquables. Maxim Zaks implémente des structures de données sophistiquées, d'autres développent des bibliothèques pour le calcul matriciel. Cette créativité témoigne d'une appropriation rapide du langage par ses utilisateurs.

Modular poursuit le développement avec ambition. L'objectif affiché : faire de Mojo un sur-ensemble complet de Python. L'ouverture progressive du code source devrait accélérer cette évolution. Le support de macOS et Windows s'étend, élargissant la base d'utilisateurs potentiels.

En 2024, Mojo continue sa croissance. Les performances s'améliorent, de nouvelles fonctionnalités apparaissent, l'écosystème s'enrichit. La démocratisation de l'intelligence artificielle et l'explosion des besoins en calcul haute performance créent un terrain favorable pour ce type d'innovation.

Les solutions émergent souvent de la combinaison créative d'approches existantes plutôt que de révolutions conceptuelles. Lattner et Davis ont su identifier une frustration largement partagée et proposer une réponse pragmatique. Leur réussite tient autant à la qualité technique qu'à la compréhension fine des besoins des développeurs modernes.

WebGPU

L'Université de l'Illinois à Urbana-Champaign se trouve confrontée à un défi inattendu en 2013. Les cours en ligne massifs connaissent un essor fulgurant – entre 2002 et 2011, les formations numériques dans l'enseignement supérieur ont grimpé de 17,3% chaque année, faisant passer le nombre

d'étudiants de 1,6 à 6,7 millions. Mais comment enseigner la programmation GPU à distance quand la plupart des apprenants n'ont pas accès à ces processeurs spécialisés ?

Les solutions classiques montrent vite leurs limites. Prêter des ordinateurs équipés de GPU ? Impraticable à grande échelle. Réserver des salles informatiques ? L'idée semble désuète face à l'ampleur des MOOC. Quant aux clusters de calcul traditionnels, ils cumulent les inconvénients : coûts prohibitifs, maintenance complexe, et une courbe d'apprentissage qui décourage les débutants.

WebGPU naît alors, fruit d'une intuition simple mais audacieuse : transformer n'importe quel navigateur web en portail d'accès à la programmation GPU. L'idée révolutionne l'approche pédagogique à l'époque. Plus besoin d'installations laborieuses ni de configurations spécifiques – il suffit d'une connexion internet pour programmer en CUDA, OpenCL ou OpenACC.

L'architecture imaginée par l'équipe de l'Illinois s'articule autour de trois éléments complémentaires. D'un côté, des serveurs web hébergent une interface utilisateur épurée. Au centre, une base de données orchestre la gestion des travaux soumis par les étudiants. Enfin, des nœuds de calcul équipés de GPU exécutent le code dans des conditions sécurisées. Cette triangulation technique cache une réelle sophistication : chaque fragment de code est d'abord analysé pour détecter des appels système interdits, puis confiné dans un environnement aux permissions drastiquement limitées.

Le test grandeur nature arrive avec le cours « Heterogeneous Parallel Programming » sur Coursera. Plus de 100 000 participants s'inscrivent, validant l'intuition initiale mais révélant aussi les spécificités des formations massives. Car si l'affluence impressionne, elle s'accompagne d'un phénomène bien connu : 85% des inscrits abandonnent en cours de route. WebGPU doit donc gérer des fluctuations dramatiques, avec des pics d'activité concentrés sur les veilles de remise de travaux.

L'équipe technique ajuste constamment le nombre de GPU disponibles, s'appuyant sur la flexibilité des services cloud d'Amazon AWS. Cette gymnastique permanente forge une expertise précieuse dans la gestion élastique des ressources de calcul. L'expérience accumule suffisamment de retours pour envisager une refonte majeure.

WebGPU 2.0 voit le jour en 2015 avec des ambitions renforcées. L'interface utilisateur migre vers OpenEdX, simplifiant l'intégration du contenu pédagogique. Mais l'innovation la plus marquante réside dans l'adoption de Docker pour isoler les environnements d'exécution. Cette technologie, encore jeune à l'époque, apporte une granularité inédite dans la gestion des ressources et facilite le déploiement de configurations sur mesure.

Les conteneurs Docker s'associent dynamiquement aux GPU physiques

selon les besoins spécifiques de chaque travail pratique. Cette souplesse architectural adapte finement les environnements d'apprentissage tout en préservant la sécurité du système global. L'université de l'Illinois déploie cette nouvelle version pour ses cours traditionnels ECE 408 et ECE 598HK.

Le succès dépasse les frontières de l'Illinois. L'Université d'État de Caroline du Nord adopte WebGPU, suivie par l'Université du Tennessee. L'école d'été PUMPS à Barcelone intègre la plateforme dans ses sessions intensives de formation. Cette diffusion géographique confirme la pertinence de l'approche au-delà du contexte initial.

L'interface volontairement simplifiée de WebGPU, limitée à six opérations principales, illustre une philosophie pédagogique particulière. Contrairement aux environnements de développement traditionnels qui multiplient les fonctionnalités, cette restriction assumée guide l'apprentissage en évitant la dispersion. L'expérience montre qu'un cadre contraint favorise parfois mieux la compréhension que la liberté totale.

Certaines expérimentations échouent néanmoins. L'évaluation par les pairs, tentée lors des premières sessions, doit être abandonnée face aux difficultés de mise en œuvre. Ces échecs nourrissent la réflexion sur les limites de l'automatisation dans l'enseignement technique.

L'équipe de conception documente minutieusement son travail dans des publications académiques. Cette transparence contribue à l'avancement des connaissances sur les plateformes éducatives en ligne. Les problématiques de scalabilité, de sécurité et d'évaluation automatisée trouvent dans WebGPU un terrain d'expérimentation riche en enseignements.

L'histoire de cette plateforme révèle les transformations de l'enseignement supérieur à l'ère numérique. WebGPU répond aux exigences spécifiques de la programmation parallèle tout en s'adaptant aux contraintes des formations massives. Son développement témoigne de l'importance croissante des infrastructures cloud dans l'éducation moderne.

En 2024, alors que l'intelligence artificielle stimule une demande sans précédent pour la formation GPU, l'héritage de WebGPU persiste. Les plateformes actuelles s'inspirent de ses innovations dans la gestion élastique des ressources et l'automatisation des évaluations. Cette histoire démontre comment une solution technique née d'un besoin pédagogique précis peut engendrer des avancées durables dans l'éducation numérique.

La conception modulaire de WebGPU reste d'actualité. Ses principes fondateurs – accessibilité, évolutivité, sécurité – continuent d'inspirer les créateurs de nouvelles plateformes d'apprentissage. L'expérience acquise avec ce système enrichit notre compréhension des enjeux liés à la démocratisation des technologies de calcul avancées, révélant qu'une innovation pédagogique peut parfois devancer les évolutions technologiques qu'elle ac-

compagne.

Épilogue

Ce livre s'achève là où commence peut-être la plus grande rupture technologique de notre histoire. Les technologies racontées ici appartiennent déjà au passé. Certaines survivent, d'autres ont disparu. Mais ce qui se profile maintenant dépasse en ampleur tout ce que nous avons connu depuis l'invention du transistor.

L'informatique quantique n'est plus une promesse lointaine. Google a démontré en 2024 qu'un de ses processeurs quantiques était 13 000 fois plus rapide que le supercalculateur le plus puissant du monde. IBM prévoit de livrer des systèmes capables d'exécuter 15 000 opérations quantiques d'ici 2028. Les investissements publics ont atteint 10 milliards de dollars en 2025, contre 1,8 milliard l'année précédente. Ces chiffres traduisent une accélération brutale. Le seuil de correction d'erreur, qui a longtemps bloqué toute application pratique, commence à être franchi. La simulation de molécules complexes pour la découverte de médicaments, l'optimisation de processus industriels ou la création de nouveaux matériaux deviennent envisageables à court terme. Mais le quantique menace les systèmes de chiffrement actuels, qui tomberont comme des châteaux de cartes face à ces machines. La course est déjà lancée pour développer des algorithmes post-quantiques avant que les premiers ordinateurs vraiment opérationnels ne sortent des laboratoires.

L'intelligence artificielle a franchi en quelques années le mur qui séparait l'expérimentation de l'usage massif. Elle s'invite partout, et c'est bien l'héritaged es matériels, des systèmes, des langages et des logiciels présentés dans ce livre (mais pas que) qui ont rendu son développement et son exécution possibles. Dans les systèmes de recommandation, les assistants vocaux, la génération de textes et d'images, la conduite autonome. Mais cette omniprésence révèle des failles. Les modèles d'IA peuvent être

manipulés, biaisés, détournés. Leur consommation énergétique explose. Un seul modèle de langage de dernière génération nécessite autant d'électricité qu'une petite ville pendant son entraînement. Les centres de données se multiplient pour absorber cette demande. Et personne ne sait vraiment où cette trajectoire nous mène. Les questions éthiques s'accumulent plus vite que les réponses. Qui contrôle ces systèmes ? Qui en est responsable quand ils se trompent ? Comment garantir qu'ils ne reproduisent pas les discriminations présentes dans leurs données d'apprentissage ? L'Europe tente de réguler avec l'AI Act, mais la technologie avance plus vite que les textes législatifs, très statiques.

Les cyberattaques ont changé d'échelle. En 2024, le coût moyen d'une violation de données a atteint 4,88 millions de dollars, en hausse de 10% sur un an. Les rançongiciels représentent 59% des attaques subies par les organisations. Le montant moyen d'une rançon payée dépasse les 2 millions de dollars. Les hôpitaux, les infrastructures critiques, les systèmes de transport sont devenus des cibles privilégiées. En 2024, l'attaque contre Change Healthcare a perturbé le traitement des réclamations médicales pour plus de 100 millions d'Américains. Snowflake a vu les données de plus de 100 de ses clients exfiltrées, dont AT&T, Ticketmaster et Santander. Les failles non corrigées restent la principale porte d'entrée : 20% des violations commencent par l'exploitation d'une vulnérabilité connue mais non patchée. Le délai moyen pour identifier et contenir une violation est de 241 jours. Plus de neuf mois pendant lesquels les attaquants peuvent se déplacer librement dans les systèmes. Face à cela, les moyens restent dérisoires. Les entreprises recrutent des équipes de sécurité qu'elles peinent à former et à conserver. L'automatisation par l'IA des défenses commence à montrer son efficacité, avec une réduction de 108 jours du temps de détection pour ceux qui l'utilisent. Mais l'IA sert aussi les attaquants, qui automatisent la reconnaissance, l'exploitation et l'exfiltration. La bataille se joue désormais à vitesse machine.

Les interfaces cerveau-machine sortent des laboratoires. Neuralink a implanté ses premiers volontaires en 2024. Un patient tétraplégique contrôle désormais un curseur d'ordinateur par la pensée pour jouer aux échecs. D'autres entreprises comme Synchron ou Precision Neuroscience ont des approches moins invasives. En 2025, une vingtaine d'essais cliniques sont en cours. Les applications médicales visent d'abord les personnes paralysées, celles atteintes de la maladie de Charcot ou d'autres maladies neurodégénératives. Mais l'ambition affichée est de restaurer la vision chez les aveugles avec des prothèses visuelles, augmenter les capacités cognitives, fusionner l'intelligence humaine et artificielle. Ces projets posent des questions vertigineuses. Où commence l'humain, où finit la machine quand une puce lit et modifie l'activité cérébrale ? Les risques d'usage détourné sont immenses.

Qui garantit qu'un État ou une entreprise ne cherchera pas à surveiller ou manipuler les pensées via ces dispositifs ? Les garde-fous éthiques existent sur le papier. Leur application dans un contexte de compétition internationale acharnée reste à prouver.

L'expérience utilisateur bascule vers de nouveaux paradigmes. Les lunettes de réalité augmentée émergent comme le prochain grand écran après le smartphone. Meta, Apple, Google, Xreal, tous investissent massivement. Les lunettes Ray-Ban Meta intègrent déjà l'IA pour identifier des objets en temps réel. Les modèles de 2025 affichent des informations directement dans le champ de vision avec un angle de 70 degrés. Le marché devrait passer de 678 600 unités en 2023 à 13 millions en 2030. Ces dispositifs changent la façon dont nous interagissons avec l'information. Plus besoin de sortir son téléphone : les directions flottent sur le trottoir, les traductions apparaissent en surimpression, les notifications se superposent au monde réel. Mais cette fusion du physique et du numérique soulève des inquiétudes. La surveillance devient permanente avec des caméras toujours actives. La vie privée s'efface devant la collecte incessante de données. Les tablettes ont échoué à remplacer le PC parce qu'elles n'offraient pas assez de puissance pour les tâches complexes. Les lunettes AR pourraient réussir là où elles ont échoué en libérant les mains tout en donnant accès à un espace de travail virtuel illimité. L'ordinateur portable et le smartphone survivront-ils à cette transition ? Difficile à dire. Certains prédisent que d'ici cinq ans, les lunettes AR remplaceront le smartphone comme dispositif principal. D'autres estiment qu'il faudra une décennie de plus pour franchir les obstacles techniques et culturels.

Porterons-nous tous une puce sous la peau ou dans le cerveau connectée en Wi-Fi à internet ? L'idée ne relève plus de la science-fiction. Des implants RFID sont déjà utilisés pour des paiements sans contact ou l'accès à des bâtiments sécurisés. Des milliers de personnes ont franchi le pas. Mais l'implant cérébral généralisé reste un horizon lointain. Les obstacles techniques sont immenses. La biocompatibilité sur le long terme n'est pas garantie. Les risques d'infection ou de rejet existent. Et surtout, l'acceptabilité sociale reste très faible. Peu de gens accepteraient aujourd'hui de se faire percer le crâne pour améliorer leurs performances cognitives. Les lunettes ou les lentilles de contact augmentées représentent une voie moins invasive et probablement plus acceptable. Plusieurs entreprises travaillent sur des lentilles capables d'afficher des informations ou de mesurer des paramètres physiologiques. Mais la miniaturisation des composants et la gestion de l'énergie posent encore problème. Il faudra sans doute attendre les années 2030 pour voir ces dispositifs sortir des prototypes.

Les infrastructures informatiques des entreprises se complexifient. Le cloud hybride, les architectures microservices, les conteneurs, l'*edge com-*

puting créent des environnements d'une sophistication inédite. Gérer ces systèmes demande des compétences coûteuses. L'automatisation par l'IA promet de simplifier cette gestion, mais elle introduit de nouvelles dépendances. Que se passe-t-il quand un système critique repose sur un modèle d'IA dont personne ne comprend vraiment le fonctionnement interne ? Les défaillances deviennent plus difficiles à diagnostiquer. Les réglementations se durcissent. Le RGPD en Europe, les lois sur la protection des données en Californie et ailleurs imposent des contraintes croissantes. Les entreprises doivent tracer l'origine des données, documenter leur traitement, garantir leur sécurité. Les amendes pour non-conformité se chiffrent en centaines de millions. Cette pression réglementaire pousse vers plus de transparence, mais elle ralentit aussi l'innovation. Trouver l'équilibre entre protection des utilisateurs et agilité des acteurs économiques reste un défi non résolu.

Le retour des conflits géopolitiques accélère la militarisation de l'informatique. L'IA embarquée dans des drones, des robots terrestres ou des systèmes d'armes autonomes devient une réalité opérationnelle. Boston Dynamics a beau promettre de ne pas armer ses robots, d'autres n'ont pas ces scrupules. Le robot Spot est testé par plusieurs armées pour des missions de reconnaissance en zone hostile. Des variantes équipées de capteurs CBRN détectent les menaces chimiques, biologiques ou radiologiques. Des prototypes armés circulent déjà dans certains pays. La Chine, les États-Unis, la Russie, Israël développent tous des systèmes d'armes dirigés par l'IA. La décision de tirer peut déjà être prise par un algorithme dans certaines configurations. Les traités internationaux peinent à suivre. L'interdiction des armes autonomes létales est discutée depuis des années sans aboutir. Chaque État craint de prendre du retard face à ses adversaires. Cette course épuise les ressources, mais personne n'ose ralentir. L'informatique devient un enjeu de souveraineté au même titre que l'énergie ou l'accès aux matières premières. Les semi-conducteurs, les données, les algorithmes d'IA sont désormais des armes stratégiques. Les sanctions et les restrictions d'exportation se multiplient. La fragmentation du monde numérique en blocs géopolitiques rivaux s'accentue.

Quelles technologies faudra-t-il inventer pour faire face à tout cela ? Le chiffrement post-quantique est en cours de développement, mais son déploiement à grande échelle prendra des années. Les architectures zero-trust, où aucun acteur n'est considéré comme fiable par défaut, se déploient péniblement. L'informatique confidentielle, qui protège les données même pendant leur traitement, progresse. Mais chaque nouvelle couche de sécurité ajoute de la complexité et ralentit les systèmes. Trouver le bon compromis entre protection et performance reste un art plus qu'une science. Les technologies de vérification formelle, qui prouvent mathématiquement qu'un programme fait exactement ce qu'on attend de lui, pourraient se démocra-

tiser. Aujourd'hui cantonnées aux systèmes critiques comme l'aéronautique ou le nucléaire, elles pourraient s'étendre à d'autres domaines si les outils deviennent plus accessibles. L'informatique quantique elle-même pourrait apporter des solutions en matière de cryptographie ou de simulation de systèmes complexes. Mais elle ne sera pas une baguette magique. Chaque avancée ouvre de nouveaux problèmes.

L'histoire de l'informatique nous a appris une chose : les ruptures ne sont jamais celles qu'on attend. Personne n'avait prévu l'ampleur du web dans les années 1990. Personne n'avait anticipé l'explosion des smartphones après 2007. Personne ne sait vraiment quelle technologie va bouleverser la prochaine décennie. Les paris sont ouverts. Mais une certitude demeure : l'informatique continuera de se réinventer, de nous surprendre, de nous obliger à repenser nos façons de travailler, de communiquer, de penser. Ce livre raconte un morceau d'une histoire qui ne s'arrête jamais. Chaque point final est un nouveau point de départ. Les technologies que nous utiliserons dans dix ans existent peut-être déjà, quelque part dans un laboratoire, sous la forme d'un prototype que personne ne prend encore au sérieux. Ou peut-être n'existent-elles pas encore, attendant qu'une idée fulgurante traverse l'esprit d'un chercheur ou d'un ingénieur quelque part dans le monde.

Mais quelles libertés sommes-nous prêts à céder pour plus de sécurité ou de confort ?

Bibliographie

[1] Olivier Cogis, Jérôme Palaysi, and Richard Terrat. Les machines : architecture des ordinateurs - d'une introduction historique à la définition d'une machine virtuelle universelle -. working paper or preprint, January 2020.

[2] Lloyd Strickland and Harry R. Lewis. *Leibniz on Binary : The Invention of Computer Arithmetic*. The MIT Press, Cambridge, MA, USA, 2022.

[3] A. Glaser. *History of Binary and Other Nondecimal Numeration*. Tomash, 1981.

[4] Kenneth C. Smith. A multiple valued logic : A tutorial and appreciation. *Computer*, 21(4) :17–27, April 1988.

[5] Saul Engelbourg. Herman Hollerith : Forgotten Giant of Information Processing. By Geoffrey D. Austrian. (New York : Columbia University Press, 1982. xvii + 418 pp. Illustrations, notes, selected bibliography, and index. $19.95.). *Journal of American History*, 69(4) :1008–1009, 03 1983.

[6] Lars Heide. *Punched-cards Systems and the Early Information Explosion, 1880-1945*. Studies in Industry and Society. Johns Hopkins University Press, United States, 2009.

[7] Maurice V. Wilkes. Charles babbage—the great uncle of computing ? *Commun. ACM*, 35(3) :15–ff., March 1992.

[8] George Boole. *The mathematical analysis of logic : being an essay towards a calculus of deductive reasoning*. Macmillan, Barclay, & Macmillan, Cambridge, 1847.

[9] Janet Heine Barnett. Origins of boolean algebra in the logic of classes : George boole, john venn and c. s. peirce. 2011.

[10] Seidner Florian. History of the boolean algebra. 2012.

[11] Michael Schroeder. A brief history of the notation of boole's algebra. 1997.

[12] George Strawn and George Strawn. Masterminds of punched card data processing : Herman hollerith and john billings. *IT Professional*, 25(6) :90–93, November 2023.

[13] William F. Luebbert and Geoffrey Austrian. *Hollerith, Herman*, page 785–786. John Wiley and Sons Ltd., GBR, 2003.

[14] Zachary S. Bischof, Romain Fontugne, and Fabián E. Bustamante. Untangling the world-wide mesh of undersea cables. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets '18, page 78–84, New York, NY, USA, 2018. Association for Computing Machinery.

[15] Maurice V. Wilkes. *Differential analyzer*, page 537–539. John Wiley and Sons Ltd., GBR, 2003.

[16] Larry Owens. *Vannevar Bush and the differential analyzer : the text and context of an early computer*, page 3–38. Academic Press Professional, Inc., USA, 1991.

[17] V. Bush. The differential analyzer. a new machine for solving differential equations. *Journal of the Franklin Institute*, 212(4) :447–488, 1931.

[18] Vannevar Bush. As we may think. *Interactions*, 3(2) :35–46, March 1996.

[19] Frank B. Cannonito. The gödel incompleteness theorem and intelligent machines. In *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference*, AIEE-IRE '62 (Spring), page 71–77, New York, NY, USA, 1962. Association for Computing Machinery.

[20] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42) :230–265, 1936.

[21] Charles Petzold. *The Annotated Turing : A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine*. Wiley Publishing, 2008.

[22] Arthur Evans. The lambda calculus and its relation to programming languages. In *Proceedings of the ACM Annual Conference - Volume 2*, ACM '72, page 714–716, New York, NY, USA, 1972. Association for Computing Machinery.

[23] Claude E. Shannon and Warren Weaver. *A Mathematical Theory of Communication*. University of Illinois Press, USA, 1963.

[24] Henry S. Tropp. *Stibitz, George Robert*, page 1687–1689. John Wiley and Sons Ltd., GBR, 2003.

[25] Raúl Rojas. The Z1 : architecture and algorithms of konrad zuse's first computer. *CoRR*, abs/1406.1886, 2014.

[26] AARON STRAUSS. Eniac : The first computer. *The Mathematics Teacher*, 69(1) :66–72, 1976.

[27] Scott McCartney, J. Ross Macdonald, and Harvey G. Cragon. ENIAC : The Triumphs and Tragedies of the World's First Computer. *Physics Today*, 53(7) :58–59, 07 2000.

[28] M. Riordan and L. Hoddeson. *Crystal Fire : The Birth of the Information Age*. The Sloan technology series. Norton, 1997.

[29] W. Shockley. *Electrons and Holes in Semiconductors : With Applications to Transistor Electronics*. Bell Telephone Laboratories series. Van Nostrand, 1950.

[30] John Gilbey. Biography : The abc of computing. *Nature*, 468 :760–761, 2010.

[31] Kiril Boyanov. John vincent atanasoff : the inventor of the first electronic digital computing. 2003.

[32] Tara H. Abraham. (physio)logical circuits : The intellectual origins of the mcculloch–pitts neural networks. *Journal of the History of the Behavioral Sciences*, 38(1) :3–25, 2002.

[33] Jack Cowan. Neural networks : The early days. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[34] Thomas Haigh and Mark Priestley. Colossus and programmability. *IEEE Ann. Hist. Comput.*, 40(4) :5–27, 2018.

[35] B. Randell. *Colossus : Godfather of the Computer*, pages 349–354. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982.

[36] Christian Robert and George Casella. A short history of markov chain monte carlo : Subjective recollections from incomplete data. *Statistical Science*, 26(1), February 2011.

[37] Christian Robert and George Casella. A History of Markov Chain Monte Carlo–Subjective Recollections from Incomplete Data–. working paper or preprint, August 2008.

[38] F.C. Williams and T. Kilburn. A storage system for use with binary-digital computing machines. *Proceedings of the IEE - Part II : Power Engineering*, 96 :183–200, 1949.

[39] Andrew V. Haeff. The memory tube and its application to electronic computation. 3(24) :281–286, October 1948.

[40] Mathieu Triclot. Norbert Wiener's politics and the history of cybernetics. In *2nd International Conference of the European Society for the History of Science : "The Global and the local : The history of science and the cultural integration of Europe"*, pages 763–766, Cracovie, Poland, September 2006.

[41] N. Wiener and R.R. Kline. *Norbert Wiener-A Life in Cybernetics : Ex-Prodigy : My Childhood and Youth and I Am a Mathematician : The Later Life of a Prodigy.* MIT Press, 2018.

[42] David Sharp. Manchester baby simulator user guide. 2001.

[43] S. H. Lavington. *The Manchester Mark 1.* Springer Berlin Heidelberg, Manchester, England, 1975.

[44] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27 :379–423, 1948.

[45] James V. Stone. Information theory : A tutorial introduction. *CoRR*, abs/1802.05968, 2018.

[46] William F. Schmitt. The UNIVAC Short Code. *Annals of the History of Computing*, 10(1) :7–18, 1988.

[47] M V Wilkes and W Renwick. The edsac - an electronic calculating machine. *Journal of Scientific Instruments*, 26(12) :385, dec 1949.

[48] Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. A systematic review of api evolution literature. *ACM Comput. Surv.*, 54(8), October 2021.

[49] Maurice Vincent Wilkes, David J. Wheeler, Stanley Gill, and Fernando J. Corbató. The preparation of programs for an electronic digital computer. 1958.

[50] Ira W. Cotton and Frank S. Greatorex. Data structures and techniques for remote computer graphics. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), page 533–544, New York, NY, USA, 1968. Association for Computing Machinery.

[51] C. J. Date and E. F. Codd. The relational and network approaches : Comparison of the application programming interfaces. In *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control : Data Models : Data-Structure-Set versus Relational*, SIGFIDET '74, page 83–113, New York, NY, USA, 1975. Association for Computing Machinery.

[52] Roy Thomas Fielding and Richard N. Taylor. *Architectural styles and the design of network-based software architectures.* PhD thesis, University of California, Irvine, 2000. AAI9980887.

[53] R. R. Everett. The whirlwind i computer. *Electrical Engineering*, 71(8) :681–686, 1952.

[54] F. E. Hamilton and E. C. Kubie. The ibm magnetic drum calculator type 650. *J. ACM*, 1(1) :13–20, January 1954.

[55] D. D. Majumder. A magnetic drum digital storage system. *IETE Journal of Research*, 7, 1961.

[56] Ravi Nair. Evolution of memory architecture. *Proceedings of the IEEE*, 103(8) :1331–1345, 2015.

[57] Lee Revens. The first 25 year : Acm 1947-1962. *Commun. ACM*, 30(10) :860–865, October 1987.

[58] Joel E. Cohen. *A Model of Simple Competition*. Harvard University Press, Cambridge, MA and London, England, 1966.

[59] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2) :147–160, 1950.

[60] P. Kuttner. The rope memory : a permanent storage device. In *Proceedings of the November 12-14, 1963, Fall Joint Computer Conference*, AFIPS '63 (Fall), page 45–57, New York, NY, USA, 1963. Association for Computing Machinery.

[61] Lars Heide. Making Business of a Revolutionary New Technology : The Eckert-Mauchly Company, 1945–1951. In John Impagliazzo, Per Lundin, and Benkt Wangler, editors, *3rd History of Nordic Computing (HiNC)*, volume AICT-350 of *History of Nordic Computing 3*, pages 207–214, Stockholm, Sweden, October 2010. Springer. Part 6 : The Making of a Nordic Computing Industry.

[62] Nancy Stern. The binac : A case study in the history of technology. *IEEE Ann. Hist. Comput.*, 1(1) :9–20, July 1979.

[63] Kazuo Goda and Masaru Kitsuregawa. The history of storage systems. *Proceedings of the IEEE*, 100(Special Centennial Issue) :1433–1440, 2012.

[64] *The Electrical World*. Number vol. 11 à 12. W.J. Johnston, 1888.

[65] Frank Land. Remembering LEO. In Arthur Tatnall, editor, *Reflections on the History of Computing : Preserving Memories and Sharing Stories*, volume AICT-387 of *IFIP Advances in Information and Communication Technology (SURVEY)*, pages 22–42. Springer, 2012.

[66] David T. Caminer, John Aris, Peter Hermon, and Frank Land. LEO : the incredible story of the world's first business computer. *J. Database Manag.*, 10(1) :44, 1999.

[67] Kurt W. Beyer. *Grace Hopper and the Invention of the Information Age*. The MIT Press, 2009.

[68] Harvard University. *A Manual of Operation for the Automatic Sequence Controlled Calculator*. MIT Press, Cambridge, MA, USA, 1985.

[69] C. J. Bashe, W. Buchholz, G. V. Hawkins, J. J. Ingram, and N. Rochester. The architecture of ibm's early computers. *IBM Journal of Research and Development*, 25(5) :363–376, 1981.

[70] Stephen Kaisler. *First Generation Mainframes : The IBM 700 Series*. Cambridge Scholars Publishing, March 2018.

[71] Donald E. Knuth. The ibm 650 : An appreciation from the field. *Annals of the History of Computing*, 8(1) :50–55, 1986.

[72] Donald E. Knuth. *The Art of Computer Programming Volumes 1-3 Boxed Set*. Addison-Wesley Longman Publishing Co., Inc., USA, 2nd edition, 1998.

[73] Smruti R. Sarangi. *Basic Computer Architecture*. White Falcon Publishing, 1st edition edition.

[74] J. W. Backus. *The Fortran Automatic Coding System for the IBM 704 EDPM*. January 1956.

[75] Albert Hoagland. History of magnetic disk storage based on perpendicular magnetic recording. *Magnetics, IEEE Transactions on*, 39 :1871 – 1875, 08 2003.

[76] Marvin L. Minsky, Nathaniel Rochester, Claude E. Shannon, and John McCarthy. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI Magazine*, 27(4) :12, 2006.

[77] John A. McKenzie. *TX-O Computer History*. MIT, October 1974.

[78] V. A. Newell and F. M. Tonge. *An Introduction to Information Processing Language V*. RAND Corporation, Santa Monica, CA, 1960.

[79] Phillippe Dreyfuss. System design of the gamma 60. In *Proceedings of the May 6-8, 1958, Western Joint Computer Conference : Contrasts in Computers*, IRE-ACM-AIEE '58 (Western), page 130–133, New York, NY, USA, 1958. Association for Computing Machinery.

[80] R. J. Creasy. The origin of the vm/370 time-sharing system. *IBM J. Res. Dev.*, 25(5) :483–490, September 1981.

[81] John Backus. *The history of Fortran I, II, and III*, page 25–74. Association for Computing Machinery, New York, NY, USA, 1978.

[82] Peter Naur. Successes and failures of the algol effort. *ALGOL Bull.*, (28) :58–62, July 1968.

[83] John McCarthy. *History of LISP*, page 173–185. Association for Computing Machinery, New York, NY, USA, 1978.

[84] J.S. Kilby. Invention of the integrated circuit. *IEEE Transactions on Electron Devices*, 23(7) :648–654, 1976.

[85] Stuart Bennett. Control and the digital computer : The early years. *IFAC Proceedings Volumes*, 35(1) :237–242, 2002. 15th IFAC World Congress.

[86] Charlie Bachman, Dan Fogal, Jack Jones, Dick Kerr, Chuck Greenberger, David Black, Mary Hollis, Greg Dillon, Jim Sweeney, Steve Wright, Bob Bemer, Grace Murray Hopper, Bill Randall, Brian Reynolds, Danny, Bill Olle, John Young, Warren Simmons, Bob Grace, Howard Bramberg, Marty Greenville, Stan Epwood, Anaheim Bushed, Tax Metaxides, Bill McGee, John Gosden, Marty Greenfield, Jonas Raven, George Mann, Goodrich Hubert, Ron Hamm, Jim Manner, Herb Beta, Bill Keating, Peg Harper, Dick Schubert, and Herb Manative. Conference on data systems languages (codasyl) interview : May 27-28, 1969. In *Computer Oral History Collection*, page 26–es, USA, 1999. Smithsonian Institution Press.

[87] Gio Wiederhold and John McCarthy. Arthur samuel : Pioneer in machine learning. *IBM Journal of Research and Development*, 36(3) :329–331, 1992.

[88] John W. Backus. The syntax and semantics of the proposed international algebraic language of the zurich ACM-GAMM conference. In *Information Processing, Proceedings of the 1st International Conference on Information Processing, UNESCO, Paris 15-20 June 1959*, pages 125–131. UNESCO (Paris), 1959.

[89] Robert Garner and Rick Dill. The legendary ibm 1401 data processing system. *IEEE Solid-State Circuits Magazine*, 2(1) :28–39, 2010.

[90] American Standards Association. Sectional Committee on Computers, X3 Information Processing, and Business Equipment Manufacturers Association (U.S.). *American Standard Code for Information Interchange : Sponsor : Business Equipment Manufacturers Association. Approved June 17, 1963.* American Standards Association Incorporated, 1963.

[91] Chuck Boyer. The 360 revolution. *IBM*, 2004.

[92] Craig Partridge. The technical development of internet e-mail. *IEEE Annals of the History of Computing*, 30(2) :3–29, 2008.

[93] B.W. Kernighan. *Unix : A History and a Memoir*. Independently Published, 2020.

[94] Robert Sedgewick. The analysis of quicksort programs. *Acta Inf.*, 7(4) :327–355, December 1977.

[95] Laila Khreisat. Quicksort a historical perspective and empirical study. *International Journal of Computer Science and Network Security*, December 2007.

[96] Roger L. Wainwright. A class of sorting algorithms based on quicksort. *Commun. ACM*, 28(4) :396–402, April 1985.

[97] Jared Reyes. A system of communication between two computers using novel frequency shift keying techniques. *University of South Dakota*, 2024.

[98] Stewart Brand. SPACEWAR ! - by Stewart Brand - Fanatic Life and Symbolic Death Among the Computer Bums. *Rolling stone*, 1972.

[99] Michel Élie and Philippe Picard. Unix et les systèmes ouverts dans Bull, avant l'Internet. *Cahiers d'histoire du Cnam*, vol.07 - 08(2) :pp. 163–171, 2017.

[100] Ralph E. Griswold and Madge T. Griswold. *History of the Icon programming language*, page 599–624. Association for Computing Machinery, New York, NY, USA, 1996.

[101] Ralph E. Griswold. *A history of the SNOBOL programming languages*, page 601–645. Association for Computing Machinery, New York, NY, USA, 1978.

[102] Kenneth E. Iverson. *A programming language*. John Wiley & Sons, Inc., 1962.

[103] Ole-Johan Dahl and Kristen Nygaard. Simula : an algol-based simulation language. *Commun. ACM*, 9(9) :671–678, September 1966.

[104] Max V. Mathews, Joan E. Miller, F. R. Moore, John R. Pierce, and J. C. Risset. *The Technology of Computer Music*. The MIT Press, 1969.

[105] M. V. Mathews. The digital computer as a musical instrument. *Science*, 142(3592) :553–557, 1963.

[106] Ivan E. Sutherland. Sketchpad : a man-machine graphical communication system. In *Proceedings of the May 21-23, 1963,*

Spring Joint Computer Conference, AFIPS '63 (Spring), page 329–346, New York, NY, USA, 1963. Association for Computing Machinery.

[107] George Radin. The early history and characteristics of pl/i. *SIGPLAN Not.*, 13(8) :227–241, August 1978.

[108] American National Standards Institute. *American National Standard for information systems : programming languages, full BASIC : ANSI X3.113-1987.* Number 68-2 in FIPS PUB. American National Standards Institute, 1430 Broadway, New York, NY 10018, USA, 1987.

[109] Allison Randal. The ideal versus the real : Revisiting the history of virtual machines and containers. *ACM Comput. Surv.*, 53(1), February 2020.

[110] James E. Thornton. The cdc 6600 project. *Annals of the History of Computing*, 2(4) :338–348, 1980.

[111] Elliott I. Organick. *The multics system : an examination of its structure.* MIT Press, Cambridge, MA, USA, 1972.

[112] J.W. Cooley, P.A.W. Lewis, and P.D. Welch. Historical notes on the fast fourier transform. *Proceedings of the IEEE*, 55(10) :1675–1677, 1967.

[113] Cynthia Solomon, Brian Harvey, Ken Kahn, Henry Lieberman, Mark L. Miller, Margaret Minsky, Artemis Papert, and Brian Silverman. History of logo. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020.

[114] Barbara Klein, Richard Alan Long, Kenneth Ray Blackman, Diane Lynne Goff, Stephen Paul Nathan, Moira McFadden Lanyi, Margaret M. Wilson, John Butterweck, and Sandra L. Sherrill. *An Introduction to IMS : Your Complete Guide to IBM Information Management System.* IBM Press, 2nd edition, 2012.

[115] Joseph Weizenbaum. Eliza — a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1) :36–45, January 1966.

[116] L. H. Seawright and J. A. Kelch. An introduction to CP-67/CMS. *IBM Data Processing Division*, September 1968.

[117] V. Rajaraman. *Fundamentals of Computers.* Prentice Hall India Pvt., Limited, 2004.

[118] Manabu Hirano and Ryotaro Kobayashi. Ransmap : Open dataset of ransomware storage and memory access patterns for creating deep learning based ransomware detectors. *Computers & Security*, 150 :104202, 2025.

[119] Douglas Engelbart. Interview, 2003.

[120] Douglas C. Engelbart and William K. English. A research center for augmenting human intellect. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), page 395–410, New York, NY, USA, 1968. Association for Computing Machinery.

[121] Vint Cerf. How the internet came to be. January 1993.

[122] Steven Pemberton. The B programming language and environment. *CWI Newsletter*, (3) :2–14, June 1984.

[123] Gerald. Luecke and Jack P. Mize. *Semiconductor memory design and application*. Texas Instruments Electronics Series. McGraw-Hill, New York, 1973.

[124] Steve Crocker. Host Software. RFC 1, April 1969.

[125] F. Baccelli, B. Blaszczyszyn, and P. Muhlethaler. An aloha protocol for multihop mobile wireless networks. *IEEE Transactions on Information Theory*, 52(2) :421–436, 2006.

[126] Norman Abramson. The aloha system : another alternative for computer communications. In *Proceedings of the November 17-19, 1970, Fall Joint Computer Conference*, AFIPS '70 (Fall), page 281–285, New York, NY, USA, 1970. Association for Computing Machinery.

[127] Leo B. Brodie. *Thinking FORTH : a language and philosophy for solving problems*. Prentice-Hall, Inc., USA, 1984.

[128] Niklaus Wirth. 50 years of pascal. *Commun. ACM*, 64(3) :39–41, February 2021.

[129] Gordon Bell and William D. Strecker. Computer structures : What have we learned from the pdp-11 ? *SIGARCH Comput. Archit. News*, 4(4) :1–14, January 1976.

[130] Digital Equipment Corporation. *PDP-11/40 Processor Handbook*. Digital Equipment Corporation, 1972.

[131] Douglas Comer. The ubiquitous b-tree. *ACM Comput. Surv.*, 11(2) :121–137, 1979.

[132] R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '70, page 107–141, New York, NY, USA, 1970. Association for Computing Machinery.

[133] James Adkinson, Warren Dalziel, James Porter, Yan Rosenshteyn, and Herbert Thompson. International Business Machines

corporation (IBM) 8 inch floppy disk drives oral history panel. *Computer History Museum*, page 38, 2005.

[134] B.W. Kernighan. *A Tutorial Introduction to the UNIX Text Editor [Ed]*. 1997.

[135] Dogan Ibrahim. *Microcontroller Projects in C for the 8051*. Newnes, London, England, April 2000.

[136] *Intel 8008 8-Bit Parallel Central Processing Unit*. Intel, 3065 Bowers Avenue, Santa Clara, California 95051, April 1972.

[137] Michael Betker, John Fernando, and Shaun Whalen. The history of the microprocessor. *Bell Labs Technical Journal*, 2 :29 – 56, 09 2002.

[138] Federico Faggin, Marcian E. Hoff Jr., Stanley Mazor, and Masatoshi Shima. The history of the 4004. *IEEE Micro*, 16(6) :10–20, December 1996.

[139] File Transfer Protocol. RFC 959, October 1985.

[140] Dennis M. Ritchie. *The development of the C programming language*, page 671–698. Association for Computing Machinery, New York, NY, USA, 1996.

[141] Alain Colmerauer and Philippe Roussel. *The birth of Prolog*, page 331–367. Association for Computing Machinery, New York, NY, USA, 1992.

[142] Alan C. Kay. *The early history of Smalltalk*, page 511–598. Association for Computing Machinery, New York, NY, USA, 1996.

[143] L. H. Seawright and R. A. MacKinnon. Vm/370 : a study of multiplicity and usefulness. *IBM Syst. J.*, 18(1) :4–17, March 1979.

[144] IEEE standard for ethernet. *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, pages 1–7025, 2022.

[145] Brad A. Myers. A brief history of human-computer interaction technology. *Interactions*, 5(2) :44–54, March 1998.

[146] D. Cooke, J. Urban, and S. Hamilton. Unix and beyond : An interview with ken thompson. *Computer*, 32(5) :58–64, 1999.

[147] Diomidis Spinellis. A repository of unix history and evolution. *Empirical Software Engineering*, 22(3) :1372–1404, Jun 2017.

[148] V. Cerf and R. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5) :637–648, 1974.

[149] B. Cosell and D. Walden. Developing telnet's negotiated options. *IEEE Annals of the History of Computing*, 25(2) :80–82, 2003.

[150] Donald D. Chamberlin, Morton M. Astrahan, Michael W. Blasgen, James N. Gray, W. Frank King, Bruce G. Lindsay, Raymond Lorie, James W. Mehl, Thomas G. Price, Franco Putzolu, Patricia Griffiths Selinger, Mario Schkolnick, Donald R. Slutz, Irving L. Traiger, Bradford W. Wade, and Robert A. Yost. A history and evaluation of system r. *Commun. ACM*, 24(10) :632–646, October 1981.

[151] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6) :377–387, 1970.

[152] Gary Kildall. CP/M : A family of 8 and 16-bit operating systems. *BYTE*, June 1981.

[153] Donald D. Chamberlin and Raymond F. Boyce. Sequel : A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '74, page 249–264, New York, NY, USA, 1974. Association for Computing Machinery.

[154] Stanley Mazor. Intel 8080 cpu chip development. *IEEE Ann. Hist. Comput.*, 29(2) :70–73, April 2007.

[155] P J Horne. Implementation of an INTEL 8080 microprocessing development system using existing minicomputers. *IEEE Trans. Nucl. Sci.*, 24(3) :1772–3, 1977.

[156] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6) :644–654, 1976.

[157] H. Edward Roberts and William Yates. Altair 8800 The most powerful minicomputer project ever presented. *Popular Electronics*, pages 33–38, January 1975.

[158] Forrest M. Mims III. The tenth anniversary of the altair 8800. *Computers & Electronics*, pages 58–82, January 1985.

[159] Richard M. Russell. The cray-1 computer system. *Commun. ACM*, 21(1) :63–72, January 1978.

[160] CERN Bulletin Issue No. 17/1988. April 1988.

[161] Valérie Schafer. X.25, une norme pour le transport de données. *Encyclopédie d'histoire numérique de l'Europe*, May 2021.

[162] Joseph Amankwah-Amoah. Competing technologies, competing forces : The rise and fall of the floppy disk, 1971–2010. *Technological Forecasting and Social Change*, 107 :121–129, May 2016.

[163] D. A. Nowitz and M. E. Lesk. A dial-up network of UNIX systems. August 1978.

[164] Peter H. Salus. Distributing the news. *Usenix.org*, 40(4), August 2015.

[165] Mike Willegal. Steve Wozniak and the Apple I. December 2019.

[166] Apple Computer Company. Apple I Operation Manual. 1976.

[167] Jim Joyce. Interview with Bill Joy. *Unix Review*, August 1984.

[168] Christopher Espinosa. *APPLE II reference manual*. Apple Computer, 1979.

[169] PET User Group. *PET User Notes*, 2(1), 1979.

[170] Digital. 1977–1997... and beyond, Nothing Stops It ! 1997.

[171] Steven Levy. *Crypto : How the Code Rebels Beat the Government–Saving Privacy in the Digital Age*. Penguin USA, 2001.

[172] Lewis Thompson. A short, BSD-specific, UNIX history. November 2003.

[173] Yukiharu SAMBE. Brief history of arcade video game display technologies. *Synthesiology*, 6(2) :93–102, 2013.

[174] Barbara Beeton, Karl Berry, and David Walden. Tex : A branch of desktop publishing evolution, part 1. *IEEE Annals of the History of Computing*, 40 :78–93, 07 2018.

[175] Barbara Beeton, Karl Berry, and David Walden. Tex : A branch in desktop publishing evolution, part 2. *IEEE Annals of the History of Computing*, 41(2) :29–41, 2019.

[176] Tim Bergin. The origins of word processing software for personal computers : 1976-1985. *Annals of the History of Computing, IEEE*, 28 :32–47, 11 2006.

[177] Jeffrey R. Yost. An interview with Seymour Rubinstein OH 391. *Charles Babbage Institute, Center for the History of Information Processing*, May 2004.

[178] John R. Hudson. Some notes on dBASE. June 2023.

[179] Jean Ichbiah. Ada : past, present, future. *Commun. ACM*, 27(10) :990–997, October 1984.

[180] Michael Hauben and Ronda Hauben. *Netizens : on the history and impact of Usenet and the Internet*. IEEE Computer Society Press, Washington, DC, USA, 1997.

[181] Steven Bellovin. Netnews : The origin story. *IEEE Annals of the History of Computing*, PP :1–13, January 2024.

[182] Lazar Raković, Marton Sakal, and Veselin Pavlićević. Spreadsheets -how it started. *International Scientific Journal of Management Information Systems*, 9 :9–14, December 2014.

[183] Martin Campbell-Kelly and Paul Ceruzzi. An interview with dan bricklin and bob frankston OH 402. *Charles Babbage Institute, Center for the History of Information Processing*, May 2004.

[184] Barry B. Brey. *The Intel microprocessors (4th ed.) : 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Pro processor : architecture, programming, and interfacing.* Prentice-Hall, Inc., USA, 1997.

[185] L. Peter Deutsch and Allan M. Schiffman. Efficient implementation of the smalltalk-80 system. In *Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '84, page 297–302, New York, NY, USA, 1984. Association for Computing Machinery.

[186] Nuno Garcia, Fabio Gil, Barbara Matos, Coulibaly Yahaya, Nuno Pombo, and Rossitza Goleva. Keyed user datagram protocol : Concepts and operation of an almost reliable connectionless transport protocol. *IEEE Access*, PP :1–1, February 2019.

[187] Gregg Williams. The Commodore VIC-20 Microcomputer : A low-cost, high-performance consumer computer. *BYTE*, pages 46–64, May 1981.

[188] Microsoft Corporation. *Microsoft XENIX Operating System OEM Directory.* Microsoft Corporation, Bellevue, Washington, May 1983. Document référence OEM00918.

[189] Microsoft Corporation. *Microsoft XENIX Operating System Version 3.0 Release Summary.* Microsoft Corporation, Bellevue, Washington, 1983.

[190] Robert B. Greenberg. The unix operating system and the xenix standard operating environment. *BYTE*, 6(6) :248–264, June 1981.

[191] W. E. Pete Peterson. *Almost perfect : How a Bunch of Regular Guys Built WordPerfect Corporation.* Prima Lifestyles, USA, November 1993.

[192] Caralie Heinrichs and Emilie Vandal. Digital preservation guide : 3.5-inch floppy disks. *Preservation guide*, December 2019.

[193] Internet Control Message Protocol. RFC 792, September 1981.

[194] Sol Libes. The origins of MS-DOS. *Micro/Systems Journal*, July 1986.

[195] Tim Paterson. An inside look at MS-DOS. *BYTE*, June 1983.

[196] Review BBC Micro. *Practical Computing*, January 1982.

[197] The IBM Personal Computer. *BYTE*, 7(1), January 1982.

[198] T.B. Henderson. *The Osborne Portable Computer*. Que on systems. Que Corporation, 1982.

[199] Mukesh Sahu and Jeebananda Panda. A time efficient approach to data compression for lzw algorithm. In *2023 Annual International Conference on Emerging Research Areas : International Conference on Intelligent Systems (AICERA/ICIS)*, pages 1–3, 2023.

[200] David E. Weisberg. *The Engineering Design Revolution - CAD History*. 2008.

[201] Devon Shapiro, Merritt Patridge, Erin McCafferty, Hillary Herstin, Smita Gupta, and Crystal Leveillee. *The Invention of Compact Discs*. EIS Main Project, November 2012.

[202] Al Hartmann. iAPX286 compiler writer's guide. *Intel Corporation*, May 1983.

[203] Roberto Dillon. *Ready : A Commodore 64 Retrospective*. Springer, January 2015.

[204] Nick Montfort, Patsy Baudoin, John Bell, Ian Bogost, Jeremy Douglass, Mark C. Marino, Michael Mateas, Casey Reas, Mark Sample, and Noah Vawter. *10 PRINT CHR$(205.5+RND(1)) ; : GOTO 10*. The MIT Press, 11 2012.

[205] Rodney Dale. *The Sinclair Story*. Duckworth, 1985.

[206] Jean-Noël Lafargue. Archéologie d'un carton de l'Informatique pour tous. *Cahiers d'histoire du Cnam*, vol.15(1) :pp. 97–131, 2021.

[207] Brigitte Morin. La saga de Thomson. *Science & Vie Micro*, (61), May 1989.

[208] Simson Garfinkel and Eugene H. Spafford. In memoriam : Charles m. geschke (1939–2021). *Commun. ACM*, 64(7) :22, June 2021.

[209] Bill Calkins. *Solaris 8 : System Administrator Certification : Exams 310-011 and 310-012*. Que Corp., USA, 2001.

[210] *Soviet bloc computers direct descendant of western technologies*. CIA, June 1989.

[211] Jules H. Gilder. The Coleco Adam : system review. *BYTE*, pages 206–220, April 1984.

[212] William Arnold. *COMPAQ Portable Computer : Use, Applications and Basic*. Harcourt Brace College Publishers, 1984.

[213] Sam Williams. *Free as in freedom*. O'Reilly, Beijing [u.a.], 1. ed. edition, 2002.

[214] Bjarne Stroustrup. *A history of C++ : 1979–1991*. AT&T Bell Laboratories, 1996.

[215] Neil Smyth. *Objective-C 2.0 Essentials*. 3rd edition, 2012.

[216] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, page 123–133, New York, NY, USA, 1988. Association for Computing Machinery.

[217] P. Mockapetris and K. J. Dunlap. Development of the domain name system. *SIGCOMM Comput. Commun. Rev.*, 18(4) :123–133, August 1988.

[218] Cisco. *Internetworking Technologies Handbook, Fourth Edition*. Cisco Press, 2003.

[219] Jonathan Brossard. SMB : Sharing more than your files... In *Blackhat USA*, Las Vegas, United States, July 2015. Blackhat.

[220] William Aspray. Oral History of Mitch Kapor. *Computer History Museum*, November 2004.

[221] Grady Booch. Oral History of Charles Simonyi. *Computer History Museum*, February 2008.

[222] O.W. Linzmayer. *Apple Confidential 2.0 : The Definitive History of the World's Most Colorful Company*. No Starch Press Series. No Starch Press, 2004.

[223] Bruce Daniels. The architecture of the lisa- personal computer. *Proceedings of the IEEE*, 72 :331 – 341, April 1984.

[224] Jonathan Pearce and Graham Bland. *MSX an introduction*. Century Communications Ltd, London, 1984.

[225] Jonathan Haworth. *Oric - The story so far*. Cambridge, November 1992.

[226] Paul Weissmann. *OpenPA, The book of PA-RISC*. Bonn, Germany, March 2024.

[227] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems : Three Easy Pieces*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA, 2018.

[228] Post Office Protocol : Version 3 : Extended service offerings. RFC 1082, November 1988.

[229] Microsoft Corporation. Post Office Protocol Version 3 (POP3) Extensions. June 2016.

[230] Ivor Spital. Sir Alan calls it a day at Amstrad. *The Amstrad Times*, June 2008.

[231] Fred Guterl. Design case history : Apple's macintosh : A small team of little-known designers, challenged to produce a low-cost,

exceptionally easy-to-use personal computer, turns out a technical milestone. *IEEE Spectrum*, 21 :34–43, 1984.

[232] David Kummer and Tom Moorman. Case study : Pcjr : Misreading the market. *IEEE Spectrum*, 24(5) :30–31, 1987.

[233] Stefan Monnier and Michael Sperber. Evolution of emacs lisp. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020.

[234] Patrick Davison. Because of the pixels : On the history, form, and influence of ms paint. *Journal of Visual Culture*, 13 :275–297, 12 2014.

[235] Brigitte Morin. La saga de Thomson. *Science & Vie Micro*, 61, May 1989.

[236] Robert W. Scheifler and Jim Gettys. The X Window System. *ACM Trans. Graph.*, 5(2) :79–109, April 1986.

[237] Hamira Shaikh, Mosina Shaikh, Shahid Shaikh, Aatif Qureshi, Farman Khan, Viren Patel, Shakila Shaikh, and Shiburaj Pappu. Closer look at the new processor : 80386. *International Journal of Computer Sciences and Engineering*, 7 :48–51, 10 2019.

[238] D.L. Mills. Internet time synchronization : the network time protocol. *IEEE Transactions on Communications*, 39(10) :1482–1493, 1991.

[239] P. Sestoft. *Spreadsheet Technology*. IT University technical report series. IT University of Copenhagen, 2011.

[240] Lykourgos G. Zacharogeorgas. *Microsoft Windows : The evolution of a revolutionary product*. 2005.

[241] John Cook, Martin Croft, Christina Erskine, Alan Fitch, Mike Gerrard, John Lettice, and Max Phillips. *The Complete Guide to Amstrad PCW 8256 & 8512*, volume 1. Roger Munford, 1986.

[242] Commodore Amiga Inc. *Amiga Hardware Reference Manual*. Addison-Wesley Publishing Company, 1989.

[243] Suzanne Crocker. Oral History of Paul Brainerd. *Computer History Museum*, May 2006.

[244] IBM. *Additional aix security tools on ibm eserver pseries, ibm rs/6000, and sp/cluster*. IBM Corp., USA, 2000.

[245] Bertrand Meyer. *Eiffel : the language*. Prentice-Hall, Inc., USA, 1992.

[246] Joe Armstrong. A history of erlang. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, HOPL III, page 6–1–6–26, New York, NY, USA, 2007. Association for Computing Machinery.

[247] D. L. Mills and H. Braun. The nsfnet backbone network. In *Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology*, SIGCOMM '87, page 191–196, New York, NY, USA, 1987. Association for Computing Machinery.

[248] D. L. Mills and H. Braun. The nsfnet backbone network. *SIGCOMM Comput. Commun. Rev.*, 17(5) :191–196, August 1987.

[249] Robert Elliott and Mallikarjun Chadalapaka. Small Computer Systems Interface (SCSI) Command Ordering Considerations with iSCSI. RFC 3783, May 2004.

[250] Chet Ramey. Bash – The GNU shell. *The Linux Journal*, August 1994.

[251] Mike Battilana. The GIF Controversy : A Software Developer's Perspective. January 1995.

[252] Michael Stevenson. Having it both ways : Larry Wall, Perl and the technology and culture of the early web. *Internet Histories*, 2(3-4) :264–280, 2018.

[253] David Ungar and Randall B. Smith. Self : The power of simplicity. In *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications*, OOPSLA '87, page 227–242, New York, NY, USA, 1987. Association for Computing Machinery.

[254] David Ungar and Randall B. Smith. Self : The power of simplicity. *SIGPLAN Not.*, 22(12) :227–242, December 1987.

[255] John Ousterhout. History of tcl. Tcl Website.

[256] Acorn Archimedes. *Personal Computer World*, pages 98–104, August 1987.

[257] Archie RISC OS. *Personal Computer World*, pages 152–155, January 1989.

[258] Benoît Georges. 1984-2024 : la folle épopée du Mac d'Apple. *Les Echos*, January 2024.

[259] Jimmy Maher. *The Future Was Here : The Commodore Amiga*. The MIT Press, 2012.

[260] R. L. Cook, F. L. Rawson, III, J. A. Tunkel, and R. L. Williams. Writing an operating system/2 application. *IBM Systems Journal*, 27(2) :134–157, 1988.

[261] Unicode Consortium, editor. *The Unicode Standard, Version 4.0*. Addison-Wesley Longman Publishing Co., Inc., USA, 2003.

[262] Jacques Morleghem. *Windows 2 et 386 facile*. 1989.

[263] David J. Murray. Frame technology and framemaker. *IEEE Ann. Hist. Comput.*, 41(3) :62–74, July 2019.

[264] Q.G. Schmierer and A.H. Wottreng. Ibm as/400 processor architecture and design methodology. In *[1991 Proceedings] IEEE International Conference on Computer Design : VLSI in Computers and Processors*, pages 440–443, 1991.

[265] Stijn Peeters. *The IRC Paradigm : An Alternative Approach for Alternative Social Networks.* May 2013.

[266] Daniel Stenberg. History of IRC.

[267] ibiblio. Gulf War IRC logs, January 1991.

[268] 2meta.com. Historic Chat Logs, January 1991.

[269] David Bozak. Soviet Coup Archive, August 1991.

[270] Rene Brogatzki. SNMP - Simple Network Management Protocol. Seminar Innovative Internettechnologien und Mobilkommunikation SS2009.

[271] PKWARE Inc. Technical Note – ZIP File Capacities.

[272] Barry B. Brey. *The Intel 32-Bit Microprocessors : 80386, 80486, and Pentium Microprocessors.* Prentice Hall PTR, USA, 1st edition, 1994.

[273] Tim Berners-Lee. Information management : A proposal. May 1989.

[274] Dr.S.Narayanan, Dr.C.Sabarigirinathan, Dr.K.Vinayagavel, and Dr.D.Deepiha. MS Office – Computer Application. *International Journal of Scientific Engineering and Applied Science*, 4, September 2018.

[275] Apple Computer Inc. Apple Computer, Inc. agrees to acquire NeXT Software Inc. *Press release*, December 1996.

[276] Jeffe Schewe. 10 years of Photoshop. *PEI Mag*, February 2000.

[277] IBM. *User's reference for PS/1 computer.* International Business Machines Corporation, April 1990.

[278] Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler. A history of haskell : being lazy with class. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, HOPL III, page 12–1–12–55, New York, NY, USA, 2007. Association for Computing Machinery.

[279] Michael Stevenson and Anne Helmond. The historical trajectories of algorithmic techniques : an interview with bernhard rieder. *Internet Histories*, 4 :1–10, February 2020.

[280] Mark P. McCahill and Farhad X. Anklesaria. *Evolution of Internet Gopher*, pages 235–246. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[281] W. J. Buchanan. *PCMCIA (PC Card)*, pages 563–570. Springer US, Boston, MA, 2004.

[282] Rassol Raissi. *The Theory Behind Mp3*. December 2002.

[283] Simon Whibley. WAV Format Preservation Assessment. *British Library*, January 2016.

[284] Charles Severance. Guido van rossum : The early years of python. *Computer*, 48(2) :7–9, 2015.

[285] S.N. Bokhari. The linux operating system. *Computer*, 28(8) :74–79, 1995.

[286] D. Clark. Visual basic and the future of web-application development. *Computer*, 34(8) :16–18, 2001.

[287] E. McLellan. The alpha axp architecture and 21064 processor. *IEEE Micro*, 13(3) :36–47, 1993.

[288] R. Neelamani, R. de Queiroz, Zhigang Fan, S. Dash, and R.G. Baraniuk. Jpeg compression history estimation for color images. *IEEE Transactions on Image Processing*, 15(6) :1365–1378, 2006.

[289] Site web IBM. IBM ThinkPad history.

[290] Dave Shreiner and The Khronos OpenGL ARB Working Group. *OpenGL Programming Guide : The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*. Addison-Wesley Professional, 7th edition, 2009.

[291] Adobe Systems. *Portable Document Format Reference Manual*. APL Series. Addison-Wesley Publishing Company, 1993.

[292] Sugandha Sharma, Krishna Nand Patel, and Aashish Siddhath Jha. Cryptography using blowfish algorithm. In *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, pages 1375–1377, 2021.

[293] Chenyuan Kou and F. Springsteel. The security mechanism in the world wide web (www) and the common gateway interface (cgi). example of central police university entrance examination system. In *Proceedings IEEE 31st Annual 1997 International Carnahan Conference on Security Technology*, pages 114–119, 1997.

[294] Annelise Anderson. *FreeBSD : an open-source operating system for your personal computer*. The Bit Tree Press, Portola Valley, CA, USA, 2nd edition, 2001.

[295] B.B. Brey. *The Intel Microprocessors : 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4, and Core2 with 64-bit Extensions : Architecture, Programming, and Interfacing.* Pearson Prentice Hall, 2009.

[296] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes. The evolution of lua. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, HOPL III, page 2–1–2–26, New York, NY, USA, 2007. Association for Computing Machinery.

[297] John M. Chambers. S, r, and data science. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020.

[298] Timothy Rooney. *Dynamic Host Configuration Protocol (DHCP)*, pages 53–68. Wiley, 2010.

[299] R.J. Vetter, C. Spell, and C. Ward. Mosaic and the world wide web. *Computer*, 27(10) :49–57, 1994.

[300] Stephen A. Thomas. *SSL and TLS Essentials : Securing the Web with CD-ROM.* John Wiley & Sons, Inc., USA, 2000.

[301] Robert Cowart. *Windows NT 3.51, with CD-ROM (Unleashed).* Macmillan Publishing Co., Inc., USA, 3rd edition, 1996.

[302] Jacob R. Lorch and Alan Jay Smith. Apple macintosh's energy consumption. *IEEE Micro*, 18(6) :54–63, November 1998.

[303] H. Iwasaki. Cd-rewritable and future disc technology. In *1997 Optical Data Storage Topical Meeting ODS Conference Digest*, pages 9–10, 1997.

[304] David E. Kalish. IBM makes a fashion statement with new aptiva s series. *The Washington Post*, September 1996.

[305] J. Charles. Open source : Netscape pops the hood [in the news]. *IEEE Software*, 15(4) :79–82, 1998.

[306] Oreste Signore. W3c technologies : a key for interoperability. 01 2003.

[307] Karen Angel. *Inside Yahoo ! : Reinvention and the Road Ahead.* John Wiley & Sons, 2002.

[308] Bernard J. Jansen, Amanda Spink, and Jan Pedersen. A temporal comparison of Altavista Web Searching : Research Articles. *J. Am. Soc. Inf. Sci. Technol.*, 56(6) :559–570, April 2005.

[309] Charles Severance. The apache software foundation : Brian behlendorf. *Computer*, 45(10) :8–9, 2012.

[310] Martin C. Brown and Cherie Plumlee. *Beos : Porting UNIX Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1998.

[311] Y. Tsunoda, R. Imura, M. Takahashi, and T. Arai. Present and future of dvd technology. In *1996. Digest of Technical Papers., International Conference on Consumer Electronics*, pages 356–, 1996.

[312] Zeqian Shen and Neel Sundaresan. ebay : an e-commerce marketplace as a complex network. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, page 655–664, New York, NY, USA, 2011. Association for Computing Machinery.

[313] John Nickolls and William J. Dally. The gpu computing era. *IEEE Micro*, 30(2) :56–69, 2010.

[314] Prashanta Das and Ganesh Deka. *History and Evolution of GPU Architecture*, pages 109–135. 01 2016.

[315] A. Van Hoff. The case for java as a programming language. *IEEE Internet Computing*, 1(1) :51–56, 1997.

[316] Allen Wirfs-Brock and Brendan Eich. Javascript : the first 20 years. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020.

[317] Mark Hills. Evolution of dynamic feature usage in php. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 525–529, 2015.

[318] Robert J. Sheehan. Teaching operating systems with ruby. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '07, page 38–42, New York, NY, USA, 2007. Association for Computing Machinery.

[319] Robert J. Sheehan. Teaching operating systems with ruby. *SIGCSE Bull.*, 39(3) :38–42, June 2007.

[320] Daniel J. Barrett, Richard E. Silverman, and Robert G. Byrnes. *SSH, the Secure Shell : The Definitive Guide*. O'Reilly Media, Inc., 2005.

[321] Heike Jurzik. Surfing the shell. *Linux Magazine*, (54) :84–87, May 2005.

[322] Sasha Pachev. *Understanding MySQL Internals*. O'Reilly Media, Inc., 2007.

[323] Dries Buytaert. The history of MySQL AB. *Planet MySQL*, March 2010.

[324] Elliot Jaffe and Scott Kirkpatrick. Architecture of the internet archive. In *Proceedings of SYSTOR 2009 : The Israeli Experimental Systems Conference*, SYSTOR '09, New York, NY, USA, 2009. Association for Computing Machinery.

[325] Jessah Foulk. Oral history interview with Rob Glaser, August 13, 2002 – Museum of History & Industry. *Speaking of Seattle Oral History Collection*, August 2002.

[326] R.P. Garg. An overview of sun ultrasparc/sup tm/ based high performance computing systems. In *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*, pages 1 pp.–, 2001.

[327] David Walden. Ward Cunningham . *IEEE Annals of the History of Computing*, 33(04) :62–67, October 2011.

[328] Bradley Bargen and Peter Donnelly. *Inside DirectX : in-depth techniques for developing high-performance multimedia applications*. Microsoft Press, USA, 1998.

[329] ICQ. *ICQ 99 User's Guide : Official Edition (with CD-ROM)*. ICQ Incorporated, 1999.

[330] Lydia Pintscher. *20 Years of KDE : Past, Present and Future*. August 2016.

[331] Robert Reinhardt, Joey Lott, and Peter Santangeli. *Macromedia Flash MX Actionscript Bible*. John Wiley & Sons, Inc., USA, 1st edition, 2002.

[332] Michael Stonebraker and Greg Kemnitz. The postgres next generation database management system. *Commun. ACM*, 34(10) :78–92, October 1991.

[333] Dawoud Shenouda Dawoud and Peter Dawoud. *7 Universal Serial Bus (USB)*, pages 245–282. 2020.

[334] Sanjay Hardikar, Sameer Rajan, and Saraswati Parmar. Virtual private network : A study of its various aspects. In *2023 International Conference on IoT, Communication and Automation Technology (ICICAT)*, pages 1–7, 2023.

[335] Robert Garner and Marc Weber. Interview of the ALOHAnet Principal Designers. *Computer History Museum*, February 2023.

[336] Jiyu He. The google pagerank algorithm. In *Proceedings of the 2nd International Academic Conference on Blockchain, Information Technology and Smart Finance (ICBIS 2023)*, pages 417–422. Atlantis Press, 2023.

[337] Jean-Pierre Teyssier, Fabien de Groote, Markus Mayer, Peter Schlemko, Andrej Rumantsev, Viktar Khutko, Roland Gesche, and Ralf Doerner. The xml file format as a general solution for measurement data storage and exchange. In *2007 69th ARFTG Conference*, pages 1–5, 2007.

[338] Daniel M. German. The gnome project : a case study of open source, global software development. *Software Process : Improvement and Practice*, 8(4) :201–215, 2003.

[339] Jaap Haartsen, Mahmoud Naghshineh, Jon Inouye, Olaf J. Joeressen, and Warren Allen. Bluetooth : vision, goals, and architecture. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(4) :38–45, October 1998.

[340] Wen-Chih Chang, Te-Hua Wang, and Yen-Ching Huang. Exploring the distribution and relationship between msn and the physical world. In *The 6th International Conference on Networked Computing and Advanced Information Management*, pages 716–721, 2010.

[341] Konstantinos Barlas, Eleni Berki, Iulia Adomnita, Thrushna Nalam, Golnaz S. Nejad, and Jari Veijalainen. Formal specification of open standards and the case of rss v2.0. In *Proceedings of the 18th Panhellenic Conference on Informatics*, PCI '14, page 1–6, New York, NY, USA, 2014. Association for Computing Machinery.

[342] Kenn Scribner, Kennard Scribner, and Mark C. Stiver. *Understanding Soap : Simple Object Access Protocol.* Sams, USA, 2000.

[343] T. Dierks and E. Rescorla. Rfc 5246 : The transport layer security (tls) protocol version 1.2. *RFC Editor*, 2008.

[344] Bengt Carlsson and Rune Gustavsson. The rise and fall of napster - an evolutionary approach. In *Proceedings of the 6th International Computer Science Conference on Active Media Technology*, AMT '01, page 347–354, Berlin, Heidelberg, 2001. Springer-Verlag.

[345] Rob Thomsett. The year 2000 bug : A forgotten lesson. *IEEE Softw.*, 15(4) :91–93,95, July 1998.

[346] Lutz Prechelt. The surprising dynamics of a simple year 2000 bug. *SIGSOFT Softw. Eng. Notes*, 24(3) :56–57, May 1999.

[347] Anders Hejlsberg, Mads Torgersen, Scott Wiltamuth, and Peter Golde. *C# Programming Language.* Addison-Wesley Professional, 4th edition, 2010.

[348] M.Q.C. van Beurden and A. Weaver. RFC 9639 : Free lossless audio codec (FLAC). Technical report, USA, 2024.

[349] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, page 263–273, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee.

[350] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system : measurements and analysis. In *Proceedings of the 4th International Conference on Peer-to-Peer Systems*, IPTPS'05, page 205–216, Berlin, Heidelberg, 2005. Springer-Verlag.

[351] Amit Singh. *Mac OS X Internals.* Addison-Wesley Professional, 2006.

[352] Andrew W. Appel. Verification of a cryptographic primitive : Sha-256. *ACM Trans. Program. Lang. Syst.*, 37(2), April 2015.

[353] Kristina Chodorow. *MongoDB : The Definitive Guide.* O'Reilly Media, Inc., 2013.

[354] Jason Swarts. The collaborative construction of "fact" on wikipedia. In *Proceedings of the 27th ACM International Conference on Design of Communication*, SIGDOC '09, page 281–288, New York, NY, USA, 2009. Association for Computing Machinery.

[355] Thuan L. Thai and Hoang Lam. *.NET Framework Essentials (2nd Edition).* O' Reilly & Associates, Inc., 2002.

[356] Charles Severance. Mitchell baker : The mozilla foundation. *Computer*, 46(2) :7–9, February 2013.

[357] Kelley Misata. The tor project : An inside view. *XRDS*, 20(1) :45–47, September 2013.

[358] Contel Bradford. *The iPod & iTunes Handbook : The Complete Guide to the Portable Multimedia Revolution.* Atlantic Publishing Company (FL), 2008.

[359] Nils Hammar. *Skype – Reasons for Growth.* Thèse de doctorat, Stockholm School of Economics, 2007.

[360] Tim Ferris. Matt Mullenweg on Characteristics and Practices of Successful Entrepreneurs, June 2018.

[361] Yuriko Isoda and Noriko Taji. Linkedin : The Growth Path of the World Largest Professional Network. *The Research Institute for Innovation Management, HOSEI UNIVERSITY*, February 2017.

[362] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the Nineteenth*

ACM Symposium on Operating Systems Principles, SOSP '03, page 164–177, New York, NY, USA, 2003. Association for Computing Machinery.

[363] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5) :164–177, October 2003.

[364] Avinash Lakshman and Prashant Malik. Cassandra : a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2) :35–40, April 2010.

[365] Xuan Zhao, Niloufar Salehi, Sasha Naranjit, Sara Alwaalan, Stephen Voida, and Dan Cosley. The many faces of facebook : experiencing social media as performance, exhibition, and personal archive. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 1–10, New York, NY, USA, 2013. Association for Computing Machinery.

[366] Martin Odersky, Philippe Altherr, Vincent Cremet, Iulian Dragos, Gilles Dubochet, Burak Emir, Sean McDirmid, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Lex Spoon, Erik Stenman, and Matthias Zenger. An overview of the scala programming language (2. edition). 03 2015.

[367] Raghu Garud and Arvind Karunakaran. Process-based ideology of participative experimentation to foster identity-challenging innovations : The case of gmail and adsense. *Strategic Organization*, 16, 06 2017.

[368] Will Reese. Nginx : the high-performance web server and reverse proxy. *Linux J.*, 2008(173), September 2008.

[369] Daniel Fett, Ralf Küsters, and Guido Schmitz. A comprehensive formal security analysis of oauth 2.0. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1204–1215, New York, NY, USA, 2016. Association for Computing Machinery.

[370] Santiago Perez De Rosso and Daniel Jackson. Purposes, concepts, misfits, and a redesign of git. *SIGPLAN Not.*, 51(10) :292–310, October 2016.

[371] Santiago Perez De Rosso and Daniel Jackson. Purposes, concepts, misfits, and a redesign of git. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA

2016, page 292–310, New York, NY, USA, 2016. Association for Computing Machinery.

[372] Vinoo Das. *Learning Redis*. Packt Publishing, 2015.

[373] Gloria Chatzopoulou, Cheng Sheng, and Michalis Faloutsos. A first step towards understanding popularity in youtube. In *2010 INFOCOM IEEE Conference on Computer Communications Workshops*, pages 1–6, 2010.

[374] J. Yates Monteith, John Mcgregor, and John Ingram. Hadoop and its evolving ecosystem. volume 987, June 2013.

[375] Andy Oakley and Jeffrey Snover. *Monad : Introducing the New MSH Command Shell and Language for Windows*. O'Reilly Media, Inc., 2005.

[376] Daylin Vliert. Apple iphone : A market case study. *MacEwan University Student eJournal*, 5, 04 2021.

[377] Rich Hickey. A history of clojure. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020.

[378] Don Syme. The early history of f#. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020.

[379] Paul King. A history of the groovy programming language. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020.

[380] Muhammad Haris, Basit Jadoon, and Farhan Khan. Evolution of android operating system : A review. *Asia Pacific Journal of Contemporary Education and Communication Technology*, November 2018.

[381] Md Tajmilur Rahman, Peter C. Rigby, and Emad Shihab. The modular and feature toggle architectures of google chrome. *Empirical Softw. Engg.*, 24(2) :826–853, April 2019.

[382] Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside dropbox : understanding personal cloud storage services. In *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, page 481–494, New York, NY, USA, 2012. Association for Computing Machinery.

[383] Gary Anthes. Html5 leads a web revolution. *Commun. ACM*, 55(7) :16–17, July 2012.

[384] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling memcache at facebook. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and*

Implementation, nsdi'13, page 385–398, USA, 2013. USENIX Association.

[385] Jose Teixeira, Gregorio Robles, and Jesús M. González-Barahona. Lessons learned from applying social network analysis on an industrial free/libre/open source software ecosystem. *CoRR*, abs/1507.04587, 2015.

[386] Aviv Zohar. Bitcoin : under the hood. *Commun. ACM*, 58(9) :104–113, August 2015.

[387] J. Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB : The Definitive Guide Time to Relax*. O'Reilly Media, Inc., 1st edition, 2010.

[388] Seikyung Jung. Web development with node.js. *J. Comput. Sci. Coll.*, 33(6) :154–156, June 2018.

[389] Eben Upton and Gareth Halfacree. *Raspberry Pi User Guide*. Wiley Publishing, 3rd edition, 2014.

[390] Clinton Gormley and Zachary Tong. *Elasticsearch : The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2015.

[391] Miska Hannuksela, Jani Lainema, and Vinod Malamal Vadakital. The high efficiency image file format standard [standards in a nutshell]. *Signal Processing Magazine, IEEE*, 32 :150–156, 07 2015.

[392] William Bugden and Ayman Alahmar. Rust : The programming language for safety and performance, 2022.

[393] Gunjan Bansal, Madhulika Sharma, Sarita Yadav, and Usha Yadav. Google Chrome Operating System. *International Journal of Innovative Research in Technology*, 2(5), October 2015.

[394] Gabriel Haas and Viktor Leis. What modern nvme storage can do, and how to exploit it : High-performance i/o for high-performance storage engines. *Proc. VLDB Endow.*, 16(9) :2090–2102, May 2023.

[395] Gilad Bracha. *The Dart Programming Language*. Addison-Wesley Professional, 1st edition, 2015.

[396] Bruno Gois Mateus and Matias Martinez. On the adoption, usage and evolution of kotlin features in android development. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '20, New York, NY, USA, 2020. Association for Computing Machinery.

[397] Sean Dubois, Claes Morgen, and Alex Zhukov. *WebRTC for the curious*.

[398] Paul Murley, Zane Ma, Joshua Mason, Michael Bailey, and Amin Kharraz. Websocket adoption and the landscape of the real-time web. In *Proceedings of the Web Conference 2021*, WWW '21, page 1192–1203, New York, NY, USA, 2021. Association for Computing Machinery.

[399] K. L. Clark and F. G. McCabe. Go !—a multi-paradigm programming language for implementing multi-threaded agents. *Annals of Mathematics and Artificial Intelligence*, 41(2–4) :171–206, August 2004.

[400] Valentin Churavy, William F Godoy, Carsten Bauer, Hendrik Ranocha, Michael Schlottke-Lakemper, Ludovic Räss, Johannes Blaschke, Mosè Giordano, Erik Schnetter, Samuel Omlin, Jeffrey S. Vetter, and Alan Edelman. Bridging hpc communities through the julia programming language, 2022.

[401] Philip Wadler and Gavin Bierman. Typescript : The Next Generation. September 2013.

[402] Dirk Merkel. Docker : lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.

[403] Ron Miller. Mirantis acquires docker enterprise, 11 2019.

[404] Adrian Ionel. What we announced today and why it matters, 11 2019.

[405] Ron Miller. After selling enterprise biz, docker lands $35m investment and new ceo, 11 2019.

[406] Docker Inc. Docker restructures and secures $35 million to advance developer workflows for modern applications, 11 2019.

[407] Ron Miller. Mirantis on run rate over $100m two years after buying docker enterprise assets, 2 2022.

[408] Ron Miller. As docker gains momentum, it hauls in $105m series c on $2b valuation, 3 2022.

[409] Frederic Lardinois. Docker raises $95m series d round for its container platform, 4 2015.

[410] Docker Inc. Docker secures $95m in series d funding to meet the unrivaled demand for its open source platform, 4 2015.

[411] Mike Wheatley. Container shakeup : Docker sells enterprise business to mirantis, appoints new ceo, 11 2019.

[412] Mike Melanson. Mirantis acquires docker enterprise, 11 2019.

[413] Bart Preneel. The first 30 years of cryptographic hash functions and the nist sha-3 competition. In *Proceedings of the 2010 International*

Conference on Topics in Cryptology, CT-RSA'10, page 1–14, Berlin, Heidelberg, 2010. Springer-Verlag.

[414] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark : a unified engine for big data processing. *Commun. ACM*, 59(11) :56–65, October 2016.

[415] James G. Shanahan and Laing Dai. Large scale distributed data science using apache spark. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, page 2323–2324, New York, NY, USA, 2015. Association for Computing Machinery.

[416] Nishant Garg. *Apache Kafka*. Packt Publishing, 2013.

[417] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes : Lessons learned from three container-management systems over a decade. *Queue*, 14(1) :70–93, January 2016.

[418] Marcel Rebouças, Gustavo Pinto, Felipe Ebert, Weslley Torres, Alexander Serebrenik, and Fernando Castor. An empirical study on the usage of the swift programming language. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 634–638, 2016.

[419] Vlatko Spasev, Ivica Dimitrovski, and Ivan Kitanovski. An overview of graphql : Core features and architecture. *ICT Innovations 2020, Web Proceedings ISSN null*, pages 110–123, 2020.

[420] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow : a system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 265–283, USA, 2016. USENIX Association.

[421] Byungjin Jun, Fabián E. Bustamante, Sung Yoon Whang, and Zachary S. Bischof. Amp up your mobile web experience : Characterizing the impact of google's accelerated mobile project. In *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery.

[422] Roberto Therón, Alicia García-Holgado, and Samuel Marcos-Pablos. An experience with microsoft teams to improve the interaction with the students. In *2021 XI International Conference on Virtual Campus (JICV)*, pages 1–3, 2021.

[423] Richard E. Turner. An introduction to transformers, 2024.

[424] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers, 2021.

[425] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with webassembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, page 185–200, New York, NY, USA, 2017. Association for Computing Machinery.

[426] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with webassembly. *SIGPLAN Not.*, 52(6) :185–200, June 2017.

[427] Anders Møller. Technical perspective : Webassembly : a quiet revolution of the web. *Commun. ACM*, 61(12) :106, November 2018.

[428] Jeff Hodges, Giridhar Mandyam, and Michael B. Jones. Registries for Web Authentication (WebAuthn). RFC 8809, August 2020.

[429] Abdul Dakkak, Carl Pearson, and Wen-Mei Hwu. Webgpu : A scalable online development platform for gpu programming courses. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 942–949, 2016.

Licence

Attribution - Partage dans les mêmes conditions 4.0 International

==========

Creative Commons Corporation (« Creative Commons ») n'est pas un cabinet d'avocats et ne donne ni services ni conseils juridiques. La mise à disposition des licences publiques Creative Commons ne crée pas de rapport analogue à celui d'un client avec son conseil ni aucun autre type de relation juridique. Creative Commons propose ses licences et les informations qui y sont associées telles quelles, sans aucune garantie relative à ses licences, aux œuvres mises à disposition conformément aux termes et conditions d'utilisation de ses licences, ou à toute autre information afférente. Creative Commons décline formellement toute responsabilité quant aux préjudices pouvant résulter de leur utilisation.

Les licences publiques Creative Commons proposent des termes et conditions d'utilisation standardisés que les auteurs et autres titulaires de droits peuvent utiliser pour partager une œuvre originale ou toute autre œuvre protégée par le droit d'auteur et certains autres droits précisés dans la licence publique ci-dessous. Les

avertissements suivants sont indiqués à titre informatif uniquement ; ils ne sont pas exhaustifs et ne font pas partie de nos licences.

Nos licences publiques sont conçues pour être utilisées par les auteurs et titulaires de droits dans la limite des lois et règlements en vigueur. Nos licences sont irrévocables. Les donneurs de licence doivent lire et comprendre les termes et conditions de la licence qu'ils choisissent avant de l'utiliser. Les donneurs de licence doivent également obtenir tous les droits nécessaires avant d'utiliser nos licences de façon à ce que le public puisse utiliser l'œuvre comme prévu. Les donneurs de licence doivent clairement indiquer quelle œuvre n'est pas soumise à la licence. Cela comprend les œuvres soumises à d'autres licences Creative Commons et les œuvres utilisées aux termes d'une exception ou d'une limitation du droit d'auteur.

Le donneur de licence qui utilise l'une de nos licences publiques accorde au public l'autorisation d'utiliser l'œuvre aux termes et conditions précisés dans la licence. Si l'autorisation du donneur de licence n'est pas nécessaire pour quelque raison que ce soit (en raison, par exemple, d'une exception ou d'une limitation applicable au droit d'auteur), cette utilisation n'est pas soumise aux termes et conditions d'utilisation de la licence. Nos licences accordent uniquement des autorisations en vertu du droit d'auteur et de certains autres droits qu'un donneur de licence a le droit d'accorder. L'utilisation de l'œuvre peut néanmoins être restreint pour d'autres raisons, par exemple, si d'autres personnes détiennent un droit d'auteur ou d'autres droits sur l'œuvre. Un donneur de licence peut formuler des demandes particulières, comme notamment que toute modification soit indiquée ou décrite. Même si cela n'est pas rendu obligatoire par nos licences, nous vous invitons à honorer ces demandes dans la mesure du possible.

==========

Attribution - Partage dans les mêmes conditions 4.0 International

Lorsque Vous exercez les Droits accordés par la licence (définis ci-dessous), Vous acceptez d'être lié par les termes et conditions de la présente Licence publique Creative Commons Attribution - Partage dans les mêmes conditions 4.0 International (la « Licence publique »). Dans la mesure où la présente Licence publique peut être interprétée comme un contrat, Vous bénéficiez des Droits accordés par la licence en contrepartie de Votre acceptation des présents termes et conditions, et le Donneur de licence Vous accorde ces droits en contrepartie des avantages que lui procure le fait de mettre à disposition l'Œuvre sous licence en vertu des présents termes et conditions.

Article 1 - Définitions.

a. Œuvre dérivée signifie œuvre protégée par les Droit d'auteur et droits connexes, dérivée ou adaptée de l'Œuvre sous licence et dans laquelle l'Œuvre sous licence est traduite, retouchée, arrangée, transformée, ou modifiée de telle façon que l'autorisation du Donneur de licence est nécessaire, conformément aux dispositions des Droit d'auteur et droits connexes. Dans le cas de la présente Licence publique, lorsque l'Œuvre sous licence est une œuvre musicale, une représentation publique ou un enregistrement sonore, la synchronisation de l'Œuvre sous licence avec une image animée sera considérée comme une Œuvre dérivée aux fins de la présente Licence publique.

b. Licence d'Œuvre dérivée signifie licence par laquelle Vous accordez Vos Droit d'auteur et droits connexes portant sur Vos contributions à l'Œuvre dérivée, selon les termes et conditions de la présente Licence publique.

c. Licence compatible BY-SA signifie licence figurant à

l'adresse suivante creativecommons.org/compatiblelicenses, approuvée par Creative Commons comme étant essentiellement équivalente à la présente Licence publique.

d. Droit d'auteur et droits connexes signifie droit d'auteur et/ou droits connexes incluant, notamment, la représentation, la radio et télédiffusion, l'enregistrement sonore et le Droit sui generis des producteurs de bases de données, quelle que soit la classification ou qualification juridique de ces droits. Dans le cadre de la présente Licence publique, les droits visés à l'Article 2(b)(1)-(2) ne relèvent ni du Droit d'auteur ni de droits connexes.

e. Mesures techniques efficaces signifie mesures techniques qui, en l'absence d'autorisation expresse, ne peuvent être contournées dans le cadre de lois conformes aux dispositions de l'Article 11 du Traité de l'OMPI sur le droit d'auteur adopté le 20 Décembre 1996 et/ou d'accords internationaux de même objet.

f. Exceptions et limitations signifie utilisation loyale et équitable (fair use et fair dealing) et/ou toute autre exception ou limitation applicable à Votre utilisation de l'Œuvre sous licence.

g. Eléments de licence signifie les composantes de la licence figurant dans l'intitulé de la Licence publique Creative Commons. Les éléments de la présente Licence publique sont : Attribution et Partage dans les mêmes conditions.

h. Œuvre sous licence signifie œuvre littéraire ou artistique, base de données ou toute autre œuvre pour laquelle le Donneur de licence a recours à la présente Licence publique.

i. Droits accordés par la licence signifie droits qui Vous sont accordés selon les termes et conditions

d'utilisation définis par la présente Licence publique, limités aux Droit d'auteur et droits connexes applicables à Votre utilisation de l'Œuvre sous licence et que le Donneur de licence a le droit d'accorder.

j. Donneur de licence signifie un individu ou une entité octroyant la présente Licence publique et les droits accordés par elle.

k. Partager signifie mettre une œuvre à la disposition du public par tout moyen ou procédé qui requiert l'autorisation découlant des Droits accordés par la licence, tels que les droits de reproduction, de représentation au public, de distribution, de diffusion, de communication ou d'importation, y compris de manière à ce que chacun puisse y avoir accès de l'endroit et au moment qu'il choisit individuellement.

l. Droit sui generis des producteurs de bases de données signifie droits distincts du droit d'auteur résultant de la Directive 96/9/CE du Parlement européen et du Conseil du 11 mars 1996 sur la protection juridique des bases de données, ainsi que tout autre droit de nature équivalente dans le monde.

m. Vous (preneur de licence) se rapporte à tout individu ou entité exerçant les Droits accordés par la licence. Votre et Vos renvoient également au preneur de licence.

Article 2 - Champ d'application de la présente Licence publique.

a. Octroi de la licence.

1. Sous réserve du respect des termes et conditions d'utilisation de la présente Licence publique, le Donneur de licence Vous autorise à exercer

pour le monde entier, à titre gratuit, non sous-licenciable, non exclusif, irrévocable, les Droits accordés par la licence afin de :

a. reproduire et Partager l'Œuvre sous licence, en tout ou partie ; et

b. produire, reproduire et Partager l'Œuvre dérivée.

2. Exceptions et limitations. Afin de lever toute ambiguïté, lorsque les Exceptions et limitations s'appliquent à Votre utilisation, la présente Licence publique ne s'applique pas et Vous n'avez pas à Vous conformer à ses termes et conditions.

3. Durée. La durée de la présente Licence publique est définie à l'Article 6(a).

4. Supports et formats : modifications techniques autorisées. Le Donneur de licence Vous autorise à exercer les Droits accordés par la licence sur tous les supports et formats connus ou encore inconnus à ce jour, et à apporter toutes les modifications techniques que ceux-ci requièrent. Le Donneur de licence renonce et/ou accepte de ne pas exercer ses droits qui pourraient être susceptibles de Vous empêcher d'apporter les modifications techniques nécessaires pour exercer les Droits accordés par la licence, y compris celles nécessaires au contournement des Mesures techniques efficaces. Dans le cadre de la présente Licence publique, le fait de ne procéder qu'à de simples modifications techniques autorisées selon les termes du présent Article 2(a)(4) n'est jamais de nature à créer une Œuvre dérivée.

5. Utilisateurs en aval.

a. Offre du Donneur de licence - Œuvre sous

licence. Chaque utilisateur de l'Œuvre sous licence reçoit automatiquement une offre de la part du Donneur de licence lui permettant d'exercer les Droits accordés par la licence selon les termes et conditions de la présente Licence publique.

b. Offre additionnelle du Donneur de licence - Œuvre dérivée. Chaque utilisateur d'une Œuvre dérivée reçoit automatiquement une offre du Donneur de licence lui permettant d'exercer les Droits accordés par la licence sur l'Œuvre dérivée selon les termes et conditions de la Licence d'Œuvre dérivée que Vous appliquez.

c. Pas de restrictions en aval pour les utilisateurs suivants. Vous ne pouvez proposer ou imposer des termes et conditions supplémentaires ou différents, ou appliquer quelque Mesure technique efficace que ce soit à l'Œuvre sous licence si ceux(celles)-ci sont de nature à restreindre l'exercice des Droits accordés par la licence aux utilisateurs de l'Œuvre sous licence.

6. Non approbation. Aucun élément de la présente Licence publique ne peut être interprété comme laissant supposer que le preneur de licence ou que l'utilisation qu'il fait de l'Œuvre sous licence est lié à, parrainé, approuvé, ou doté d'un statut officiel par le Donneur de licence ou par toute autre personne à qui revient l'attribution de l'Œuvre sous licence, comme indiqué à l'Article 3(a)(1)(A)(i).

b. Autres droits.

1. Les droits moraux, tel que le droit à l'intégrité de l'œuvre, ne sont pas accordés par la présente Licence publique, ni le droit à

l'image, ni le droit au respect de la vie privée, ni aucun autre droit de la personnalité ou apparenté ; cependant, dans la mesure du possible, le Donneur de licence renonce et/ou accepte de ne pas faire valoir les droits qu'il détient de manière à Vous permettre d'exercer les Droits accordés par la licence.

2. Le droit des brevets et le droit des marques ne sont pas concernés par la présente Licence publique.

3. Dans la mesure du possible, le Donneur de licence renonce au droit de collecter des redevances auprès de Vous pour l'exercice des Droits accordés par la licence, directement ou indirectement dans le cadre d'un régime de gestion collective facultative ou obligatoire assorti de possibilités de renonciation quel que soit le type d'accord ou de licence. Dans tous les autres cas, le Donneur de licence se réserve expressément le droit de collecter de telles redevances.

Article 3 - Conditions d'utilisation de la présente Licence publique.

L'exercice des Droits accordés par la licence est expressément soumis aux conditions suivantes.

a. Attribution.

1. Si Vous partagez l'Œuvre sous licence (y compris sous une forme modifiée), Vous devez :

a. conserver les informations suivantes lorsqu'elles sont fournies par le Donneur de licence avec l'Œuvre sous licence :

i. identification du(des) auteur(s) de l'Œuvre sous licence et de toute

personne à qui revient l'attribution de l'Œuvre sous licence, dans la mesure du possible, conformément à la demande du Donneur de licence (y compris sous la forme d'un pseudonyme s'il est indiqué) ;

ii. l'indication de l'existence d'un droit d'auteur ;

iii. une notice faisant référence à la présente Licence publique ;

iv. une notice faisant référence aux limitations de garantie et exclusions de responsabilité ;

v. un URI ou un hyperlien vers l'Œuvre sous licence dans la mesure du possible ;

b. Indiquer si Vous avez modifié l'Œuvre sous licence et conserver un suivi des modifications précédentes ; et

c. Indiquer si l'Œuvre sous licence est mise à disposition en vertu de la présente Licence publique en incluant le texte, l'URI ou l'hyperlien correspondant à la présente Licence publique.

2. Vous pouvez satisfaire aux conditions de l'Article 3(a)(1) dans toute la mesure du possible, en fonction des supports, moyens et contextes dans lesquels Vous Partagez l'Œuvre sous licence. Par exemple, Vous pouvez satisfaire aux conditions susmentionnées en fournissant l'URI ou l'hyperlien vers la ressource incluant les informations requises.

3. Bien que requises aux termes de l'Article 3(a)(1)(A), certaines informations devront être

retirées, dans la mesure du possible, si le Donneur de licence en fait la demande.

b. Partage dans les mêmes conditions.

Outre les conditions indiquées à l'Article 3(a), si Vous Partagez une Œuvre dérivée que Vous avez réalisée, les conditions suivantes s'appliquent aussi.

1. La Licence d'Œuvre dérivée que Vous appliquez doit être une licence Creative Commons avec les mêmes Eléments de licence, qu'il s'agisse de cette version ou d'une version ultérieure, ou une Licence compatible BY-SA.

2. Vous devez inclure le texte, l'URI ou l'hyperlien correspondant à la Licence d'Œuvre dérivée que Vous appliquez. Ces conditions peuvent être satisfaites dans la mesure du raisonnable suivant les supports, moyens et contextes via lesquels Vous Partagez l'Œuvre dérivée.

3. Vous ne pouvez pas proposer ou imposer des termes ou des conditions supplémentaires ou différents ou appliquer des Mesures techniques efficaces à l'Œuvre dérivée qui seraient de nature à restreindre l'exercice des Droits accordés par la Licence d'Œuvre dérivée que Vous appliquez.

Article 4 - Le Droit sui generis des producteurs de bases de données.

Lorsque les Droits accordés par la licence incluent le Droit sui generis des producteurs de bases de données applicable à Votre utilisation de l'Œuvre sous licence :

a. afin de lever toute ambiguïté, l'Article 2(a)(1) Vous accorde le droit d'extraire, réutiliser,

reproduire et Partager la totalité ou une partie substantielle du contenu de la base de données ;

b. si Vous incluez la totalité ou une partie substantielle du contenu de la base de données dans une base de données pour laquelle Vous détenez un Droit sui generis de producteur de bases de données, la base de données sur laquelle Vous détenez un tel droit (mais pas ses contenus individuels) sera alors considérée comme une Œuvre dérivée, y compris pour l'application de l'Article 3(b) ; et

c. Vous devez respecter les conditions de l'Article 3(a) si Vous Partagez la totalité ou une partie substantielle du contenu des bases de données.

Afin de lever toute ambiguïté, le présent Article 4 complète mais ne remplace pas Vos obligations découlant des termes de la présente Licence publique lorsque les Droits accordés par la licence incluent d'autres Droit d'auteur et droits connexes.

Article 5 - Limitations de garantie et exclusions de responsabilité.

a. Sauf indication contraire et dans la mesure du possible, le Donneur de licence met à disposition l'Œuvre sous licence telle quelle, et n'offre aucune garantie de quelque sorte que ce soit, notamment expresse, implicite, statutaire ou autre la concernant. Cela inclut, notamment, les garanties liées au titre, à la valeur marchande, à la compatibilité de certaines utilisations particulières, à l'absence de violation, à l'absence de vices cachés ou autres défauts, à l'exactitude, à la présence ou à l'absence d'erreurs connues ou non ou susceptibles d'être découvertes dans l'Œuvre sous licence. Lorsqu'une limitation de garantie n'est pas autorisée en tout ou partie, cette clause peut ne pas Vous être applicable.

b. Dans la mesure du possible, le Donneur de licence ne saurait voir sa responsabilité engagée vis-à-vis de Vous, quel qu'en soit le fondement juridique (y compris, notamment, la négligence), pour tout préjudice direct, spécial, indirect, incident, conséquentiel, punitif, exemplaire, ou pour toutes pertes, coûts, dépenses ou tout dommage découlant de l'utilisation de la présente Licence publique ou de l'utilisation de l'Œuvre sous licence, même si le Donneur de licence avait connaissance de l'éventualité de telles pertes, coûts, dépenses ou dommages. Lorsqu'une exclusion de responsabilité n'est pas autorisée en tout ou partie, cette clause peut ne pas Vous être applicable.

c. Les limitations de garantie et exclusions de responsabilité ci-dessus doivent être interprétées, dans la mesure du possible, comme des limitations et renonciations totales de toute responsabilité.

Article 6 - Durée et fin.

a. La présente Licence publique s'applique pendant toute la durée de validité des Droits accordés par la licence. Cependant, si Vous manquez à Vos obligations prévues par la présente Licence publique, Vos droits accordés par la présente Licence publique seront automatiquement révoqués.

b. Lorsque les Droits accordés par la licence ont été révoqués selon les termes de l'Article 6(a), ils seront rétablis :

    1. automatiquement, à compter du jour où la violation aura cessé, à condition que Vous y remédiiez dans les 30 jours suivant la date à laquelle Vous aurez eu connaissance de la violation ; ou

    2. à condition que le Donneur de licence l'autorise expressément.

Afin de lever toute ambiguïté, le présent Article 6(b) n'affecte pas le droit du Donneur de licence de demander réparation dans les cas de violation de la présente Licence publique.

c. Afin de lever toute ambiguïté, le Donneur de licence peut également proposer l'Œuvre sous licence selon d'autres termes et conditions et peut cesser la mise à disposition de l'Œuvre sous licence à tout moment ; une telle cessation n'entraîne pas la fin de la présente Licence publique.

d. Les Articles 1, 5, 6, 7, et 8 continueront à s'appliquer même après la résiliation de la présente Licence publique.

Article 7 - Autres termes et conditions.

a. Sauf accord exprès, le Donneur de licence n'est lié par aucune modification des termes de Votre part.

b. Tous arrangements, ententes ou accords relatifs à l'Œuvre sous licence non mentionnés dans la présente Licence publique sont séparés et indépendants des termes et conditions de la présente Licence publique.

Article 8 - Interprétation.

a. Afin de lever toute ambiguïté, la présente Licence publique ne doit en aucun cas être interprétée comme ayant pour effet de réduire, limiter, restreindre ou imposer des conditions plus contraignantes que celles qui sont prévues par les dispositions légales applicables.

b. Dans la mesure du possible, si une clause de la présente Licence publique est déclarée inapplicable, elle sera automatiquement modifiée a minima afin de la rendre applicable. Dans le cas où la clause ne

peut être modifiée, elle sera écartée de la présente Licence publique sans préjudice de l'applicabilité des termes et conditions restants.

c. Aucun terme ni aucune condition de la présente Licence publique ne sera écarté(e) et aucune violation ne sera admise sans l'accord exprès du Donneur de licence.

d. Aucun terme ni aucune condition de la présente Licence publique ne constitue ou ne peut être interprété(e) comme une limitation ou une renonciation à un quelconque privilège ou à une immunité s'appliquant au Donneur de licence ou à Vous, y compris lorsque celles-ci émanent d'une procédure légale, quel(le) qu'en soit le système juridique concerné ou l'autorité compétente.

==========

Creative Commons n'est pas partie prenante de ses licences publiques. Néanmoins, Creative Commons se réserve le droit d'utiliser une de ses licences publiques pour les œuvres qu'elle publie, et dans ce cas sera considérée comme « Donneur de licence ». Le texte des licences publiques Creative Commons est versé au domaine public en vertu de CC0 Domaine Public. A l'exception des seuls cas où il est indiqué que l'œuvre est mise à disposition sous licence publique Creative Commons et ceux autorisés par les statuts de Creative Commons disponibles sur creativecommons.org/policies, Creative Commons n'autorise l'utilisation par aucune partie de la marque "Creative Commons" ou de toute autre marque ou logo de Creative Commons sans le consentement écrit préalable de Creative Commons. Cette restriction relative à l'utilisation des marques ne constitue pas une partie de nos licences publiques.

Vous pouvez joindre Creative Commons via creativecommons.org.