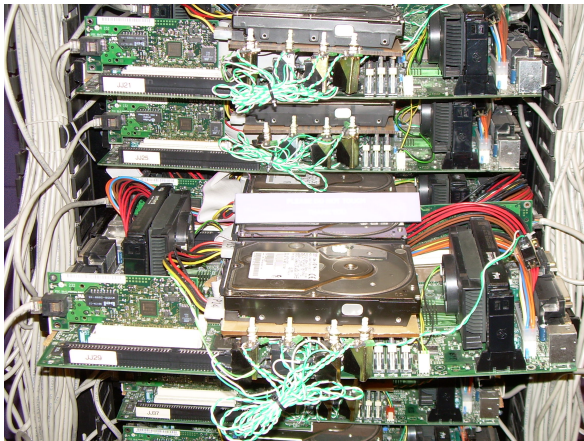


Introduction au développement logiciel

L'interface matériel / logiciel

Georges-André Silber, Centre de recherche en informatique

Mines Paris — PSL, octobre 2026



Deux étudiants de Stanford, un *cluster* fait maison, une application massivement parallèle, dès le premier jour.



Quelques jours après la signature du contrat avec IBM pour écrire le logiciel de l'IBM PC : MS-DOS.

Les compagnies informatiques dominantes se sont construites sur la **maîtrise de l'informatique à bas niveau** :

Google un moteur de recherche *plus rapide* que les autres : cluster et application parallèle construits par ses fondateurs.

Apple d'abord du matériel, puis son propre logiciel : aujourd'hui toute la chaîne, du microprocesseur à l'OS.

Facebook un socle technologique solide et un investissement massif et constant sur l'infrastructure et le logiciel.

Amazon a compris très tôt l'importance stratégique de sa propre infrastructure et de sa propre pile logicielle.

Microsoft est né d'un système d'exploitation pour l'IBM PC : MS-DOS.

Quoi?

Applications
Langages de haut niveau
Langages système : C, Rust
Système d'exploitation
Langage d'assemblage
Langage machine
Matériel

Ce cours s'intéresse aux couches **basses** de la pile :

- comment la machine exécute un programme ;
- comment le logiciel s'interface avec elle ;
- comment on écrit, organise et fait évoluer ce logiciel (**génie logiciel**).

1. Introduction : quoi, pourquoi, comment
2. Matériel : codes, portes logiques, architecture d'un ordinateur
3. Langage machine, langage d'assemblage
4. Système d'exploitation, programmation système
5. Langages « bas niveau » : C, Rust
6. Génie logiciel
7. Optimisation de code et parallélisme

Pourquoi?

1. **Comprendre ce qu'on programme.** Un programme n'est pas un texte abstrait : il consomme des cycles, des octets, des watts.
2. **Aspects stratégiques.** Qui maîtrise les couches basses maîtrise la chaîne (GAFAM, souveraineté, cloud, IA).
3. **Performance.** La fin de la loi de Moore déplace les gains de performance du matériel vers le logiciel.
4. **Responsabilité.** Le numérique a une empreinte croissante ; c'est le logiciel qui décide de l'usage du matériel.

Responsabilité environnementale

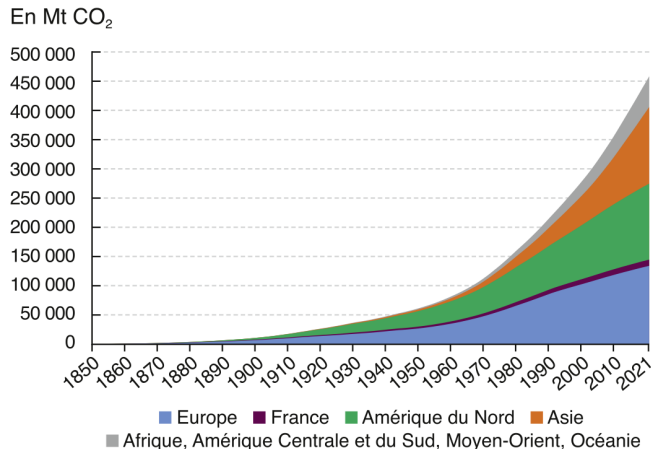
- consommer moins d'énergie lors de l'utilisation des logiciels ;
- utiliser plus efficacement le matériel, plus longtemps → moins de déchets.

Responsabilité sociétale

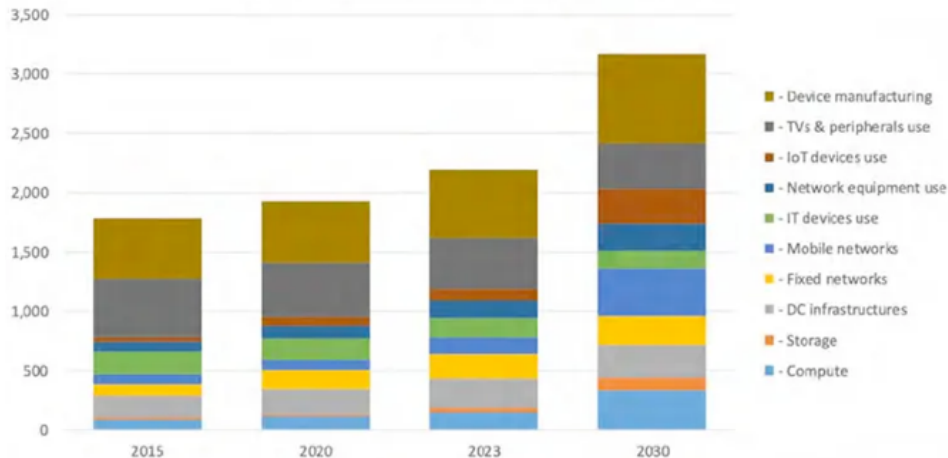
- contribuer aux logiciels libres, biens communs numériques ;
- produire des logiciels plus sûrs.

Faire mieux avec moins

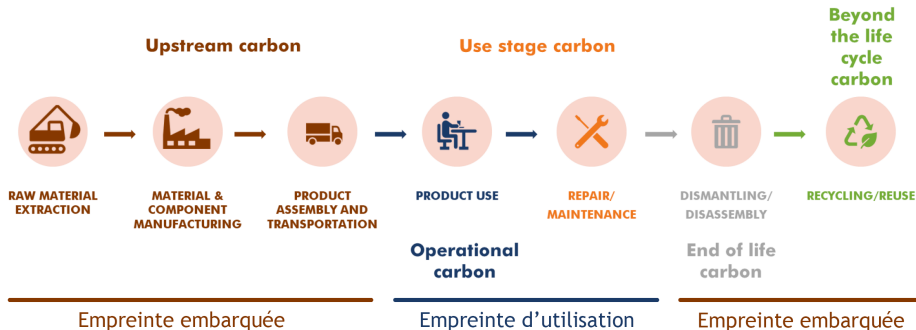
Cumuls des émissions de CO₂ de 1850 à 2021



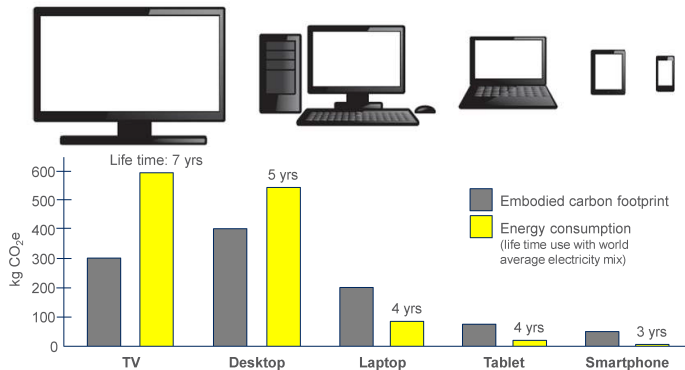
Source : Global Carbon Project, via <https://www.statistiques.developpement-durable.gouv.fr/edition-numerique/la-france-face-aux-neuf-limites-planetaires/4-changement-climatique>.



Source : L. Eeckhout, « Sustainable computer systems », HiPEAC 2024 Keynote 2, 23 février 2024.
<https://www.youtube.com/watch?v=jUw7QQ-a00Q>



Source : L. Belkhir et al., « Assessing ICT global emissions footprint : Trends to 2040 & recommendations », J. of Cleaner Production, 3/2018.



Source : J. Malmelin et D. Lundén, « *The Energy and Carbon Footprint of the Global ICT and E&M Sectors 2010–2015* », Sustainability, 9/2018.



Crédit : Arnout Fierens (<https://arnulf.be>).

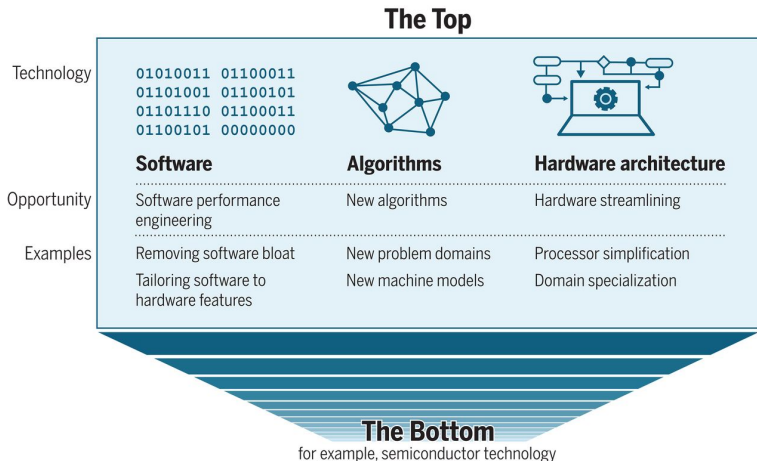


- Laboratoire expérimental du CRI
- 100 machines
- 169 CPU, 1 094 cœurs
- 2,234 To RAM / 65 To stockage
- En état de marche, considérées dépassées
- Prolonger leur vie ? Énergie ? Performance ?
- → ce cours : comprendre la machine pour mieux l'utiliser

« *Software expands to fill the available memory.* » — Parkinson

« *Software is getting slower more rapidly than hardware becomes faster.* » — Reiser

— Niklaus Wirth, *A Plea for Lean Software*, IEEE Computer, 1995.

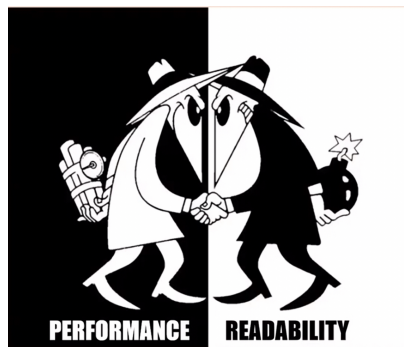


Pendant 50 ans, les gains sont venus du *bottom* (loi de Moore). Ils viendront désormais du *top* : logiciel, algorithmes, architecture.

```
for i in range(4096):
    for j in range(4096):
        for k in range(4096):
            C[i][j] += A[i][k] * B[k][j]
```

#	Version	Temps (s)	Accélération
1	Python	25 552,48	1
2	Java	2 372,68	11
3	C	542,67	47
4	C, boucles parallèles	69,80	366
5	C, diviser pour régner parallèle	3,80	6 727
6	+ vectorisation	1,10	23 224
7	+ instructions AVX	0,41	62 806

Leiserson et al., 2020, table 1. Chaque ligne correspond à un chapitre de ce cours.



D'où le chapitre « génie logiciel » : on ne gagne rien à écrire du code rapide que personne ne peut maintenir.

Comment?

Cours

- transparents et notes de cours (polycopié) par chapitre;
- lectures : Feynman (1959), Leiserson (2020), Wirth (1995).

Travaux pratiques

- microcontrôleur STM32 (Arm Cortex-M7) sur carte Nucleo;
- assembleur x86-64;
- programmation système : processus, ordonnanceur, secteur de démarrage;
- Git : histoire et entrailles;
- Rust;
- optimisation et parallélisme.

- R. Feynman, *There's Plenty of Room at the Bottom*, 1959.
- C. E. Leiserson et al., *There's plenty of room at the Top : What will drive computer performance after Moore's law ?*, Science, 2020. Code des expérimentations : <https://github.com/neboat/Moore>.
- N. Wirth, *A Plea for Lean Software*, IEEE Computer, 1995.
- Site du cours : transparents, polycopiés, sujets de TP.