

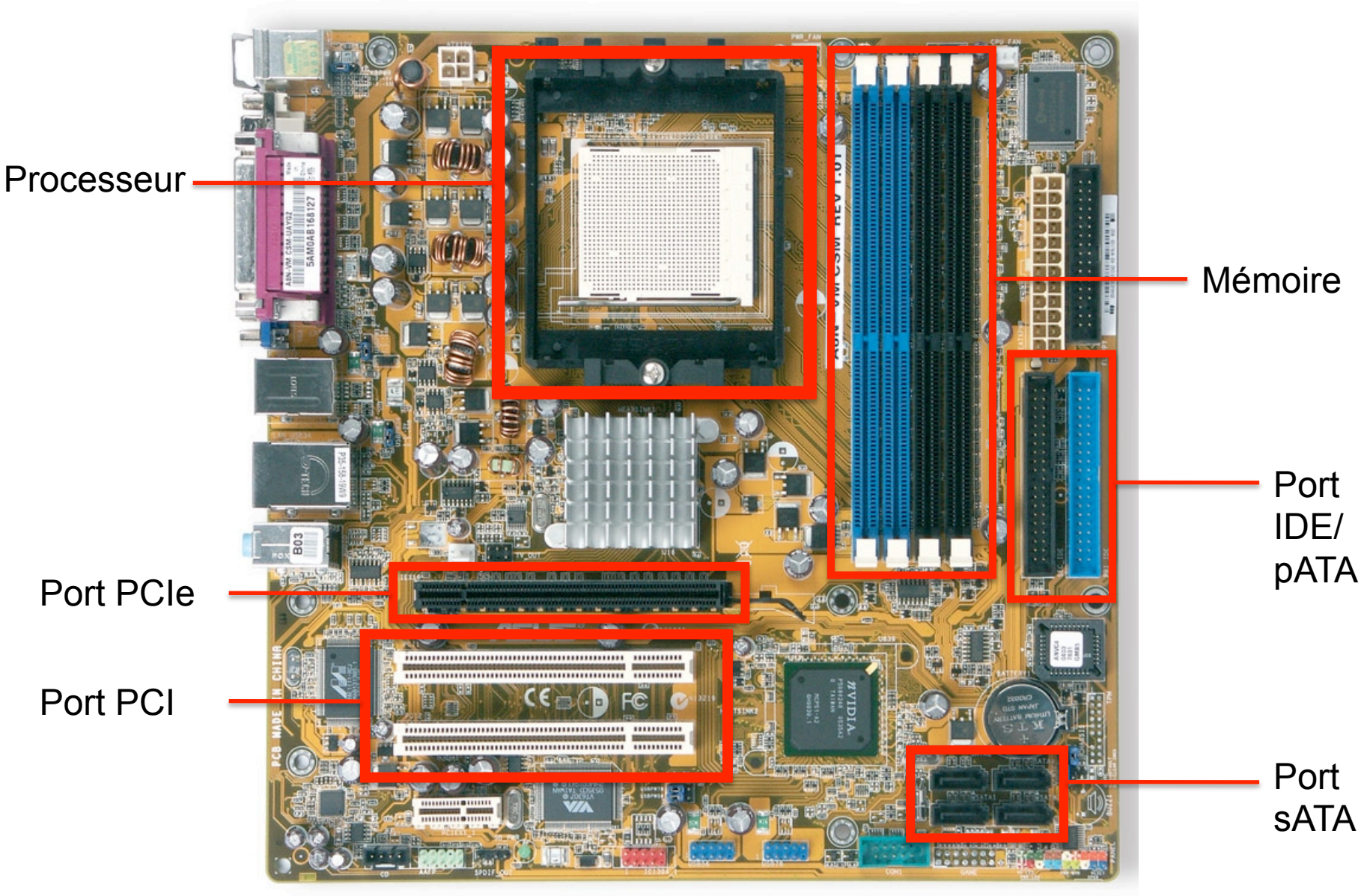
COMMENT FONCTIONNE UN ORDINATEUR

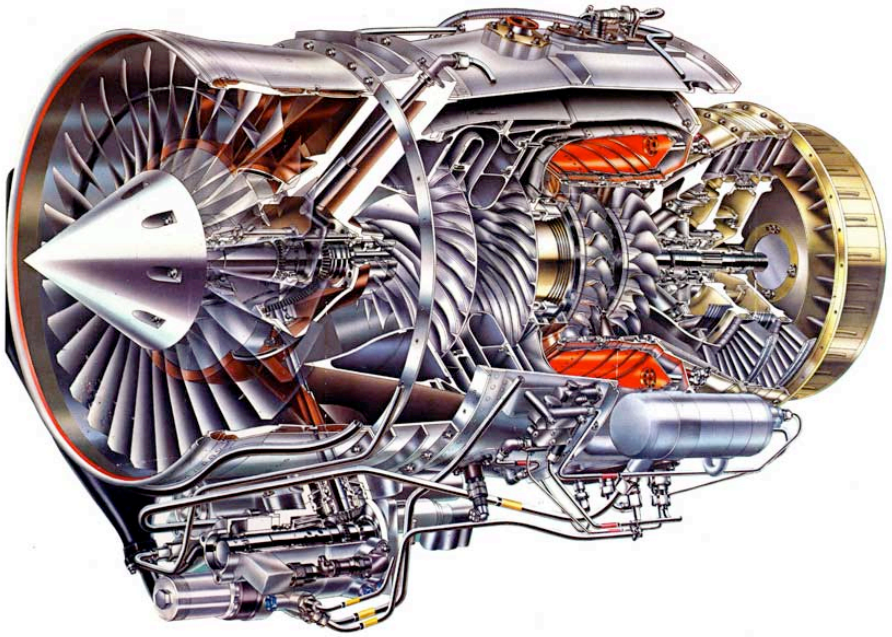
Fabrice Huet

INRIA-CNRS-Université de Nice

Composants d'un ordinateur moderne

- Processeur (CPU)
- Mémoire
- Stockage (Disque dur, CD, carte mémoire…)
- Périphériques de sortie (carte vidéo, écran, carte son…)
- Périphériques d'entrée (clavier, souris, capteur …)
- Tous ces éléments sont placés sur une *carte mère*


Processeur
Mémoire
Port IDE/ pATA
Port PCIe
Port PCI
Port sATA



LE PROCESSEUR

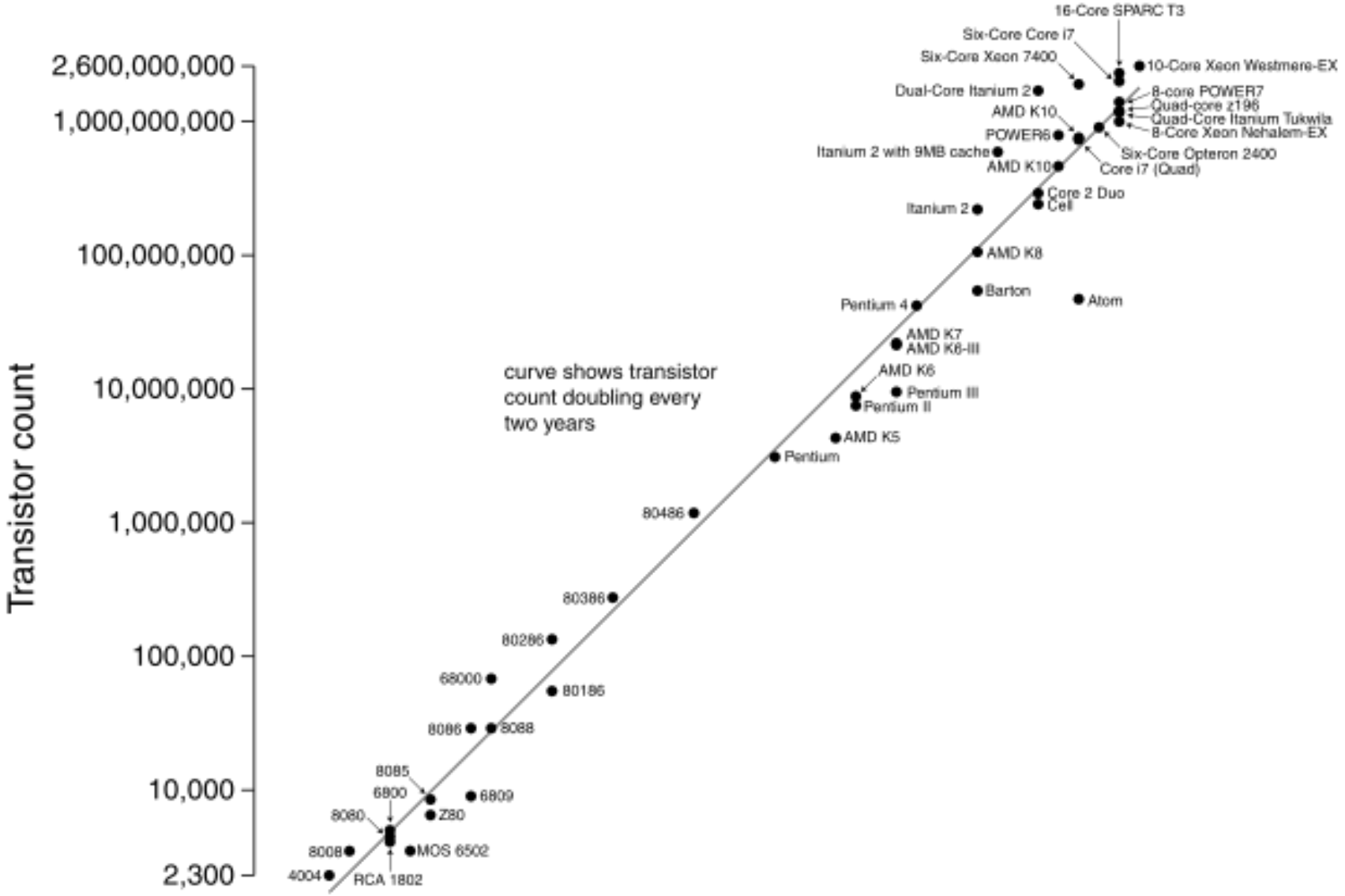
Le processeur

- **C'est le cœur de l'ordinateur**
- **Il applique un programme sur des données**
  - Un programme est une suite d'instructions
  - Les données sont situées en mémoire
- **Principe général**
  - Lecture d'une instruction en mémoire
  - Lecture des paramètres nécessaires en mémoire
  - Exécution de l'instruction
  - Stockage du résultat en mémoire

Contenu d'un processeur

- **Un processeur est composé de transistors**
  - Sorte d'interrupteurs, ouverts ou fermés
- **Ces transistors sont gravés sur une pastille de silicium**
  - Finesse de gravure == taille d'un transistor
  - Actuellement 14nm (14.10^{-9} m) au mieux
- **Nombre de transistors par CPU**
  - 10-core Xeon Westmere-EX 2.6 milliards sur 512 mm^2

Microprocessor Transistor Counts 1971-2011 & Moore's Law



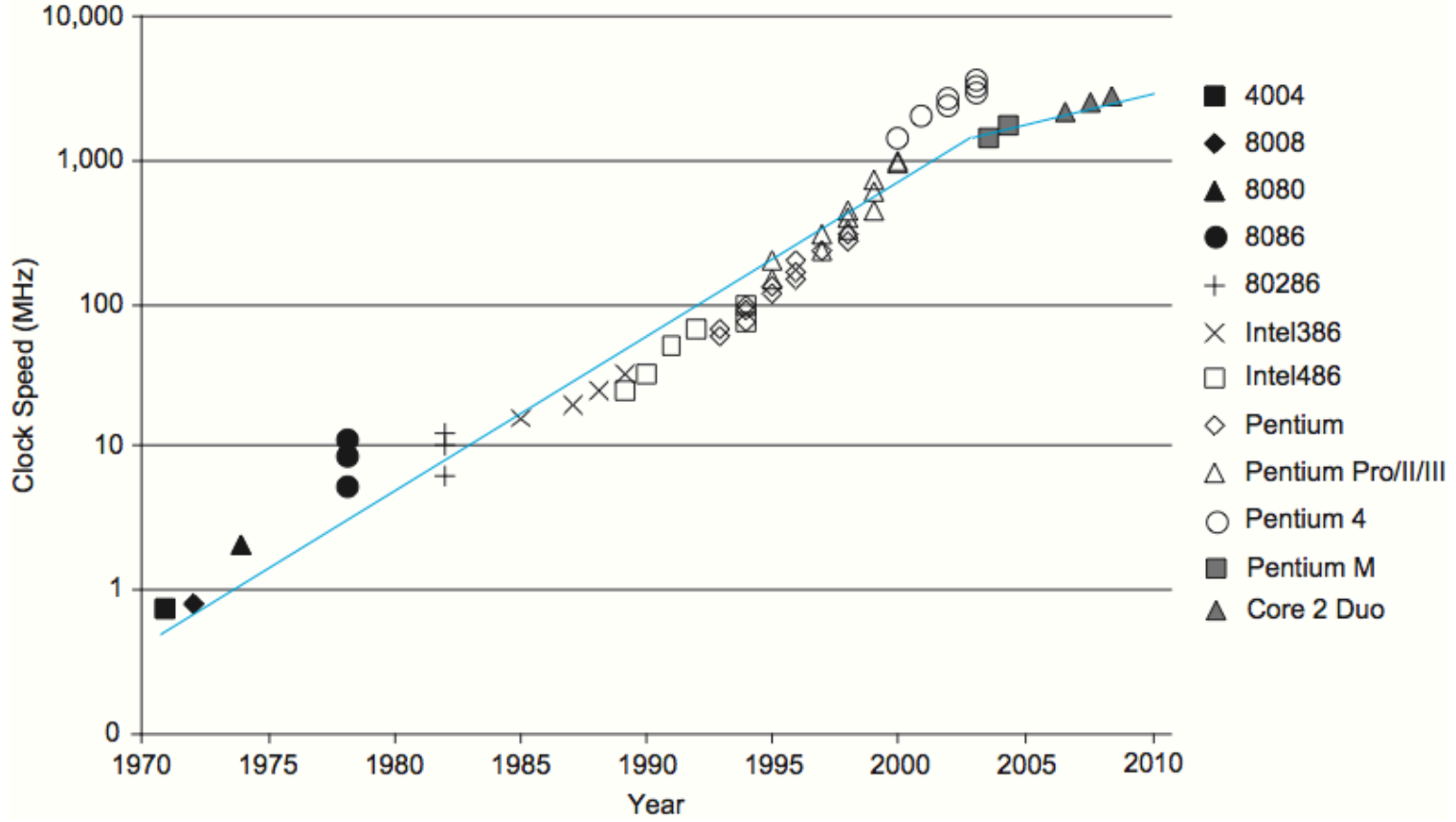
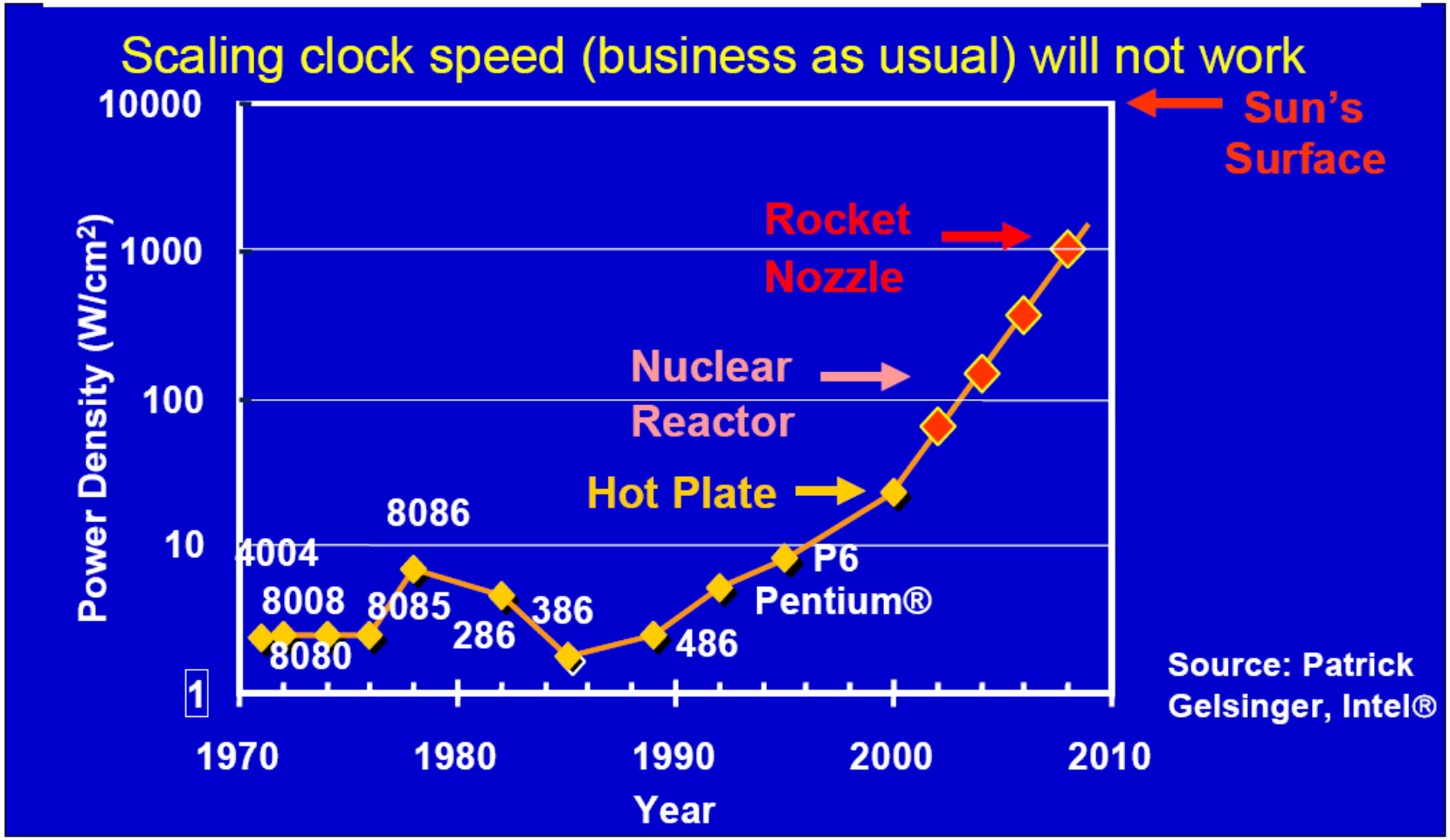


FIGURE 1.5 Clock frequencies of Intel microprocessors

From : **CMOS VLSI Design,** Weste and Harris.

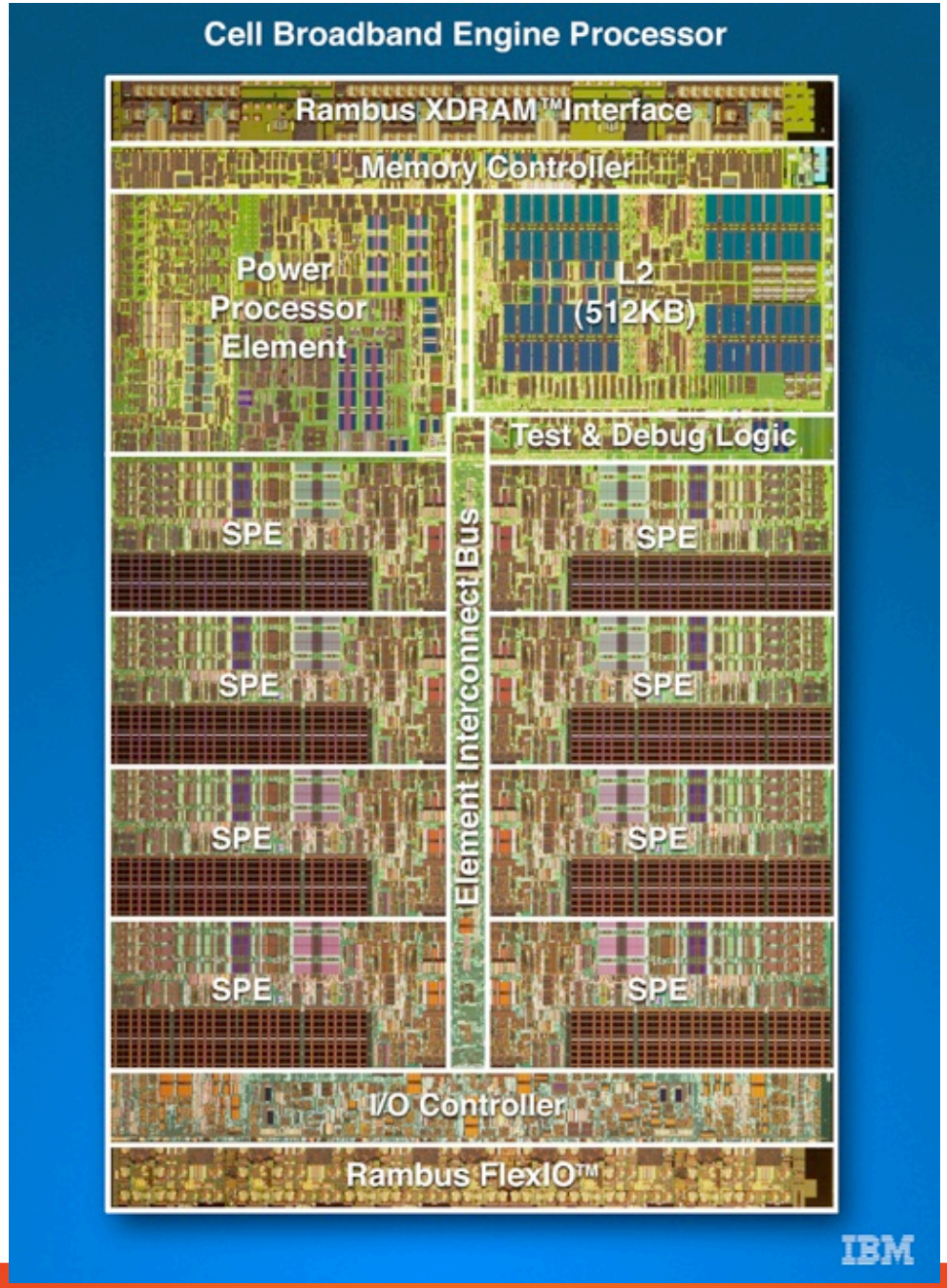
Moore's Law limitations



Source, cours CS194 U.Berkeley

Contenu d'un processeur

- **Tous les transistors ne servent pas à la même chose**
- **Organisation d'un processeur par fonctionnalité**
  - Calcul sur entier, sur flottant
  - Gestion mémoire
- **A chaque fonctionnalité est dédié un certain nombre de transistors**


Cell Broadband Engine Processor
Rambus XDRAM™ Interface
Memory Controller
Power Processor Element
L2 (512KB)
Test & Debug Logic
SPE
SPE
SPE
SPE
SPE
SPE
SPE
SPE
Element Interconnect Bus
I/O Controller
Rambus FlexIO™
IBM

Fréquence du processeur

- **Un processeur fonctionne à une certaine fréquence**
  - Guidée par une horloge extérieure
  - Exprimée en MHz ou GHz
  - Nombre de basculements de transistors par seconde
  - Core 2 Duo E6850 : 3Ghz
  - Phenom 9950 : 2,6Ghz
- **A chaque tic d'horloge le CPU peut effectuer du travail**
  - Cycle CPU
- **Chaque instruction du CPU nécessite un certain nombre de cycles**
  - Dépend de la complexité de l'instruction

Fréquence du processeur

- **Fréquence élevée => beaucoup d'instructions/seconde**
- **Mais une fréquence élevée augmente la température**
  - Facteur limitant
- **La fréquence n'est pas une bonne indication des performances**
  - Une opération (addition, division…) peut nécessiter un nombre différentes instructions suivant les processeurs
- **Utilisation de benchmarks**

Langage Machine

- **Donc l'instruction se trouve en mémoire**
- **À quoi ressemble une instruction?**
  - C'est un chiffre en binaire
- **Exemple (source: wikipedia)**
  - 10110000 01100001 : place la valeur 97 dans le registre AL
- **C'est compliqué**

Langage Assembleur

- **Résumons :**
  - Le CPU exécute des instructions trouvées en mémoire
  - Ces instructions sont des chiffres
  - Elles forment un programme
- **Qui écrit un programme ?**
  - Un programmeur ☺
- **Aucun humain n'est capable d'écrire un programme composé de suites de chiffres**
- **Il faut une représentation de plus haut niveau**
  - Le langage Assembleur

Langage Assembleur

- **Un langage assembleur est un langage de *bas niveau***
  - Syntaxe spécifique à la famille de processeurs
  - Supporte directement les instructions du processeur
- **Il introduit une représentation symbolique des instructions**
  - Remplace les chiffres par des mots
  - Beaucoup plus facile pour un humain

Exemple : (extrait de) petit programme x86

```
    mov eax, 1
    mov ebx, 10
    xor ecx, ecx
our_loop:
    add ecx eax
    inc eax
    cmp eax, ebx
    jle our_loop
```

```
int sum = 0;
for(int i = 1; i <= 10; i++)
   sum += i;
```

équivalent C

Et plus lisible ?

- **Peut-on avoir des langages**
  - qui ressemblent plus à des langues naturelles?
  - qui ne nécessitent pas de gérer tous les détails…
  - font tout tout seul...
- **Oui!**
  - Langages de haut niveau
- **Le problème c'est qu'il y en a énormément**
  - 2000+

Mother Tongues

Tracing the roots of computer languages through the ages

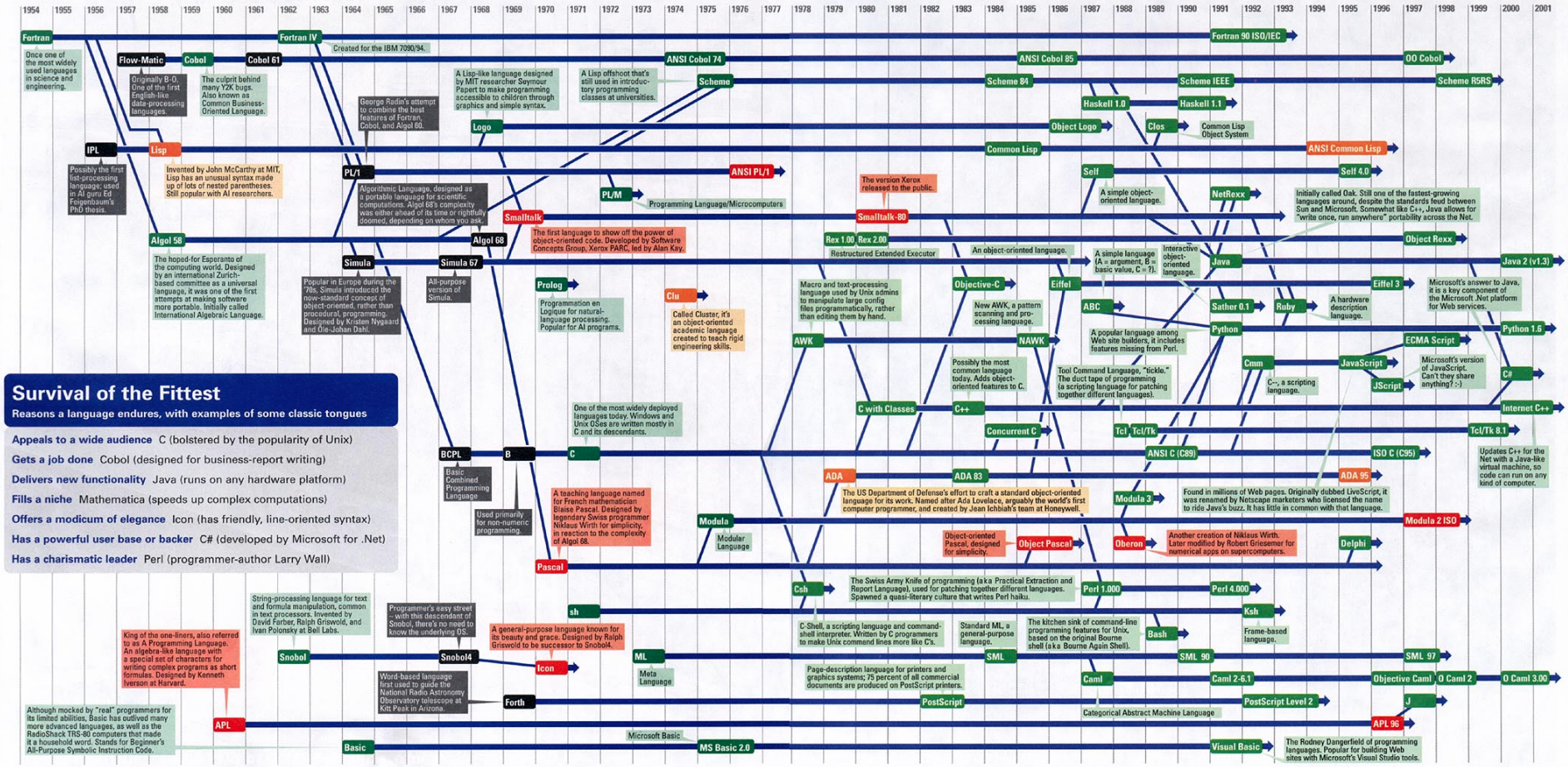
Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java, and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers – electronic lexicographers, if you will – aim to save, or at least document, the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua francas. Among the most endangered are Ada, APL, B (the predecessor of C), Lisp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *www.informatik.uni-freiburg.de/Java/misc/lang_list.html*. **– Michael Menduno**

Key

- 1954 **Year Introduced**
- **Active:** thousands of users
- **Protected:** taught at universities; compilers available
- **Endangered:** usage dropping off
- **Extinct:** no known active users or up-to-date compilers
- **Lineage continues**



Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

STOCKAGE – DISQUE DUR

Description

- **Un disque dur (DD/HD) est un stockage non volatile**
- **Composé de plateaux en rotation**
  - Nombre de plateaux et vitesse variable suivant les modèles
  - Actuellement 5400,7200, 10000 ou 15000 tr/min
  - Vitesse de rotation constante
- **Plateaux double face**
  - À chaque face est associé une tête de lecture
  - L'ensemble des tête est posé sur un bras rigide

Description - 2

- **Stockage magnétique de l'information**
  - Une tête de lecture « flotte » au dessus des disques
  - 8-12 nanomètres
- **La tête est un aimant qui arrange des particules sur le disque**
- **Nécessite une bonne pression atmosphérique dans le disque**
  - Aucune impureté


SAMSUNG
WU32543A
WARNING
DELICATE PRODUCT
www.letheonline.net
LeThe
Joshua Marius

Head crash

LE SYSTÈME D'EXPLOITATION

Définition

- **Il n'existe pas une unique définition pour un Système d'Exploitation (OS)**
  - Pourtant plein sont connus : Windows, Linux, OSX, Solaris, FreeBSD, ….
- **On peut dégager des éléments**
  - C'est un **programme**
  - Il fournit des **services** aux autres programmes
  - Il **masque** la diversité matérielle
  - Il **gère** l'accès aux ressources matérielles

Langage de programmation

- **L'OS est un programme**
- **Donc écrit dans un langage**
  - Premiers OS écrits en Assembleur
  - Maintenant en C et assembleur quand nécessaire
- **Exemple : Linux 2.6.27**
  - 6 399 191 lignes de code source
  - 96.39% de code ANSI C, 3.32% d'assembleur

Programme privilégié

- **L'OS est différent des autres programmes**
  - C'est le premier à s'exécuter
  - Il bénéficie d'un accès privilégié au processeur
- **L'OS peut exécuter des instructions interdites aux autres programmes**
- **Notion de niveau de privilège (ring en x86)**
  - Niveau 0 : OS, tous les droits
  - Niveau 3 : programme classique

Fournisseur de service

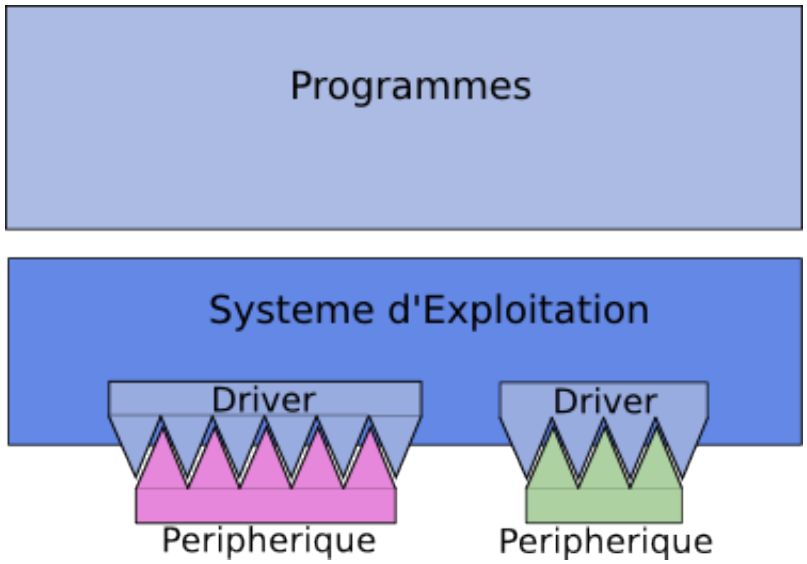
- **L'OS fournit des services aux autres programmes**
- **Il leur permet**
  - de s'exécuter
  - d'accéder aux ressources (mémoire, disque, carte d'extension…)
  - de communiquer entre eux

OS et diversité

- **Un OS doit s'exécuter sur un maximum de matériel différent**
  - Impossible d'avoir une version différente pour chaque processeur, mémoire, carte vidéo …
- **Comment faire ?**
- **Limiter la diversité en ayant du matériel normalisé**
  - Norme USB 1.0, 2.0, …
- **Spécialiser l'OS en lui ajoutant des extensions pour communiquer avec du matériel spécialisé**

Driver/Pilote

Un *driver* (*module* sous Linux) est une extension du système d'exploitation qui lui permet de gérer un périphérique



Accès aux ressources

- **Un programme passe par l'OS pour accéder à une ressource, par exemple**
  - demande de mémoire
  - demande de lecture/écriture sur un périphérique de stockage
  - demande d'affichage
- **L'OS vérifie**
  - que le programme a le droit (sécurité)
  - que la ressource est libre (graveur par exemple) ou partageable (mémoire)
  - que l'accès est équitable

Résumons

- **L'OS est un programme critique**
  - qui sert aux autres programmes
  - qui a le contrôle sur les ressources du système
  - qui applique des règles de contrôle d'accès
- **Si un programme ne respecte pas les règles….**

- **Mais si c'est un programme, il s'exécute sur le CPU…**
  - Comment d'autres programmes peuvent-ils s'exécuter?

CPU partagé

- **Quand il peut y avoir plusieurs programmes en cours d'exécution, on parle de multitâche**
- **Il n'y a souvent qu'un CPU par machine**
  - mais tous les programmes en ont besoin
  - et l'OS aussi en a besoin
- **Considérons le CPU comme une ressource**
  - chaque programme devrait avoir un peu de cette ressource
  - si chaque programme utilise un peu le CPU, on aura l'impression que tout le monde l'a (effet dessin animé)
- **Le passage d'un processus à un autre est appelé changement de contexte**

Multitâche collaboratif

- **Comment partager le CPU?**
- **Solution simple:**
  - L'OS donne le CPU a un processus
  - le processus utilise le CPU
  - quand il a fini, il rend le CPU à l'OS ou à un autre processus
- **Facile à mettre en œuvre mais**
  - Problème si un processus ne rend pas le CPU (volontairement ou bug)

Multitâche préemptif

- **Il faudrait pouvoir limiter le temps que passe un processus sur le CPU**
  - chaque a le cpu pour une durée *x*
  - si il n'en a plus besoin avant, il donne le cpu à l'OS
  - sinon l'OS est remis sur le CPU
- **Ok, mais remis comment ?**
- **On utilise un mécanisme du CPU appelé interruption**

Interruption

- **Une interruption est un signal indiquant un évenement nécessitant une action**
- **Interruption logicielle**
  - Signal envoyé par un logiciel
- **Interruption matérielle**
  - Signal envoyé par du matériel (clavier, souris, carte réseau, horloge, …)

Interrupt handler

- **Quand une interruption est levée, elle arrive au CPU**
- **Le CPU doit la traiter**
  - Il exécute un programme (fonction) appelé *interrupt handler*
- **C'est l'OS qui enregistre son propre code comme *interrupt handler***

Multitâche préemptif - 2

- **Une horloge génère des interruptions régulières**
  - 50-60Hz
- **L'OS place un *handler* pour les interruptions de l'horloge**
  - toutes les 15-20ms, le CPU exécute ce *handler*
  - donc l'OS est remis sur le CPU même si un processus ne l'avait pas rendu
- **L'OS peut ensuite décider quel processus aura le CPU**
  - c'est le *scheduling* (ordonnancement)

Résumons

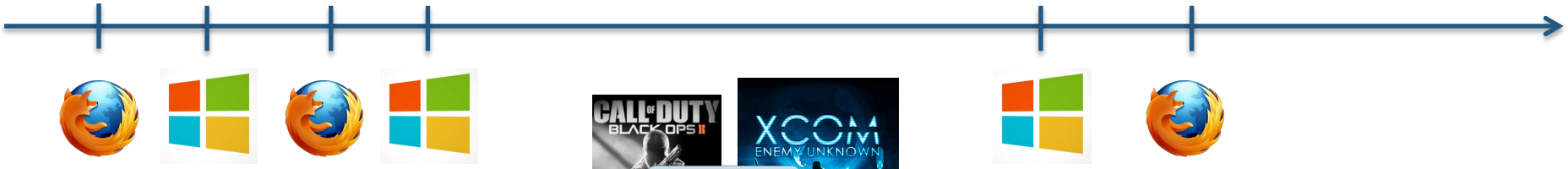
- **L'OS laisse sa place à un programme**
- **Si le programme n'a pas fini après 15ms**
  - L'horloge génère une interruption
  - L'OS reprend le CPU
  - Il décide ce qu'il doit faire
- **Et c'est pareil pour n'importe quelle interruption**
  - Souris déplacée par exemple…

Exemple complet

- **Regardons l'exécution ce qui se passe lors du click sur une page web**

L'OS comprend que le click est pour FF

En attendant la page, l'OS continue a donner le CPU aux autres programmes



Interruption matérielle

FF demande la page web en utilisant la carte réseau

La carte réseau a reçu les données, interruption matérielle

L'OS donne les données à FF qui affiche la page