# Informatique et Langage C

1re année GPSE

Thibaud Sadé

ENSICAEN

8 septembre 2026

# Table des matières


- Qu'est-ce que l'informatique ?
- Une brève histoire des ordinateurs
- L'architecture matérielle d'un ordinateur
  - Le processeur (CPU)
  - Modèles d'architectures
  - Mémoires et temps d'accès
- Du langage machine au langage évolué
- Système d'exploitation
  - Gestion des processus
  - Gestion de la mémoire
  - Système de fichier
  - Pilotes matériels et abstraction
- Mémoire vive
- Capacité d'adressage et puissance de calcul

# Qu'est-ce que l'informatique ?

# Origine du mot « informatique »

En 1957, l'ingénieur allemand *Karl Steinbuch* crée le terme *« Informatik »* dans son essai « Informatik : Automatische Informationsverarbeitung », que l'on peut traduire par « Informatique : traitement automatique de l'information ».

En 1962, *Philippe Dreyfus*, ingénieur et entrepreneur français, utilise le terme *« informatique »* pour nommer sa société « Société d'Informatique Appliquée ».

C'est un néologisme formé par contraction des mots **information** et **automatique**.

En 1967, l'Académie française adopte ce terme pour désigner la « science du traitement de l'information ».

# Définition et composantes

L'informatique est la science du traitement automatique de l'information par des ordinateurs et des logiciels.

**Trois grands domaines :**

- *Le matériel (hardware) :* ordinateurs et composants (carte mère, processeur, mémoire, disque, carte graphique, écran, clavier, etc.).
- *Les logiciels (software) :* programmes et instructions qui ordonnent au matériel les tâches à exécuter (système d'exploitation, pilotes, navigateur, éditeur de texte, jeux, etc.).
- *Les réseaux (network) :* ensemble des systèmes et moyens de télécommunications permettant aux ordinateurs de communiquer entre eux.

# Une brève histoire des ordinateurs

# De la mécanique à l'électronique

L'histoire de l'informatique commence par des machines à calculer purement mécaniques avant de basculer vers l'électronique programmable au milieu du XX$^{e}$ siècle.

**Trousiales ancêtres mécaniques**

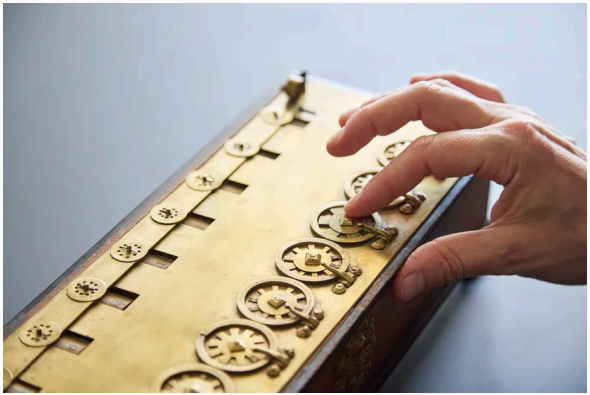
- **1642** — *Blaise Pascal* : la *Pascaline*, première machine à calculer mécanique.
- **1837** — *Charles Babbage* : la *machine analytique*, première conception d'un ordinateur programmable (jamais construite de son vivant).
- *Ada Lovelace* y écrira le premier « programme ».

**Le bascu vers l'électronique**

- **1945** — **ENIAC** : premier grand ordinateur entièrement électronique (tubes à vide, 30 tonnes).
- **1947** — invention du **transistor** (Bell Labs).
- **1958** — premier **circuit intégré** (puce), par *Jack Kilby* (Texas Instruments).

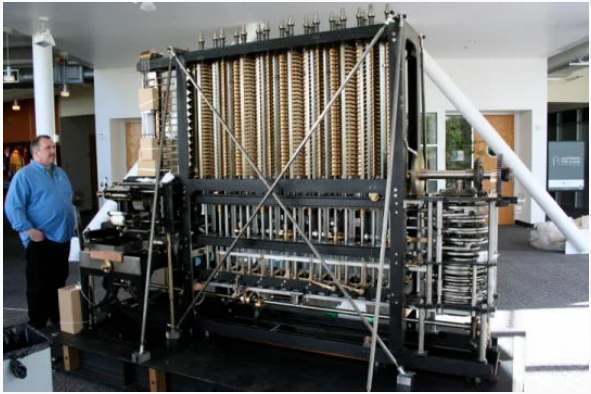
# La pascaline - 1642

La Pascaline est la première calculatrice mécanique de l'histoire, inventée en 1642 par le jeune prodige français Blaise Pascal alors qu'il n'avait que dix-neuf ans pour alléger le travail de son père, Étienne Pascal, qui était percepteur et contrôleur des impôts en Haute-Normandie.



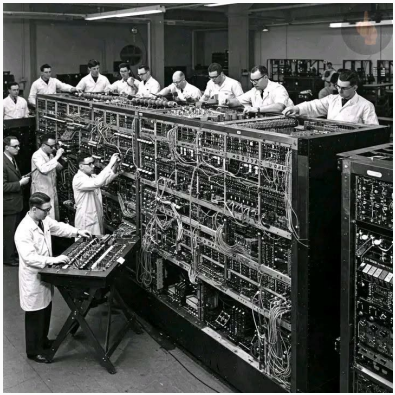
# Machine à différence de Charles Babbage - 1837

La « machine à différences », ordinateur entièrement mécanique, qui calcule sans erreurs des polynômes grâce à la méthode de calcul différentielle. Une simple manivelle permet de faire fonctionner la machine et de lancer le calcul. Un calcul par 6 secondes ! Construite pour la 1ère fois en 1991 seulement par le musée des sciences de Londre.



# ENIAC - 1945

L'ENIAC, conçu en 1945, mesurait 15 mètres de long et pesait 30 tonnes. Pas de disque dur, Les données étaient saisies via des cartes perforées. Il nécessitait 150 kW d'électricité, assez pour alimenter une petite ville de l'époque. Révolutionnaire à l'époque, mais aujourd'hui moins puissant qu'une calculatrice graphique de lycée.



# Vers l'ordinateur personnel

## Les étapes clés

- **1971** — **Intel 4004** : premier microprocesseur complet sur une seule puce de silicium.
- **1976** — création d'Apple, premiers micro-ordinateurs.
- **1981** — **IBM PC** : standardisation de l'ordinateur personnel modulaire moderne.
- **2007** — *iPhone* : généralisation de l'informatique mobile (ARM).

## La miniaturisation

Le passage des tubes à vide aux transistors, puis aux circuits intégrés, a permis de réduire les calculateurs de la taille d'une pièce à celle d'une puce en silicium. C'est le fil conducteur de toute l'histoire du matériel, encore actif aujourd'hui.

# Evolution du transistor dans l'informatique

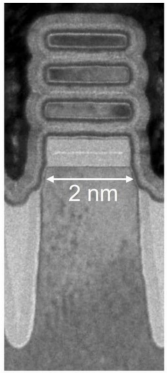
**Un transistor en 1945**


ENIAC
105 tubes à vide / m²
(source : https://computersicence.chemeketa.edu)

**76 ans**

**Densité x 4762 milliards pour la fonction « transistor »**

**Un transistor en 2021**




Prototype IBM
500 000 milliards GAA FET / m²
(source : IBM)

# Évolution du transistor : Le grand saut historique

## 1945 : L'ère des tubes à vide (ENIAC)

Le tout premier calculateur entièrement électronique de l'histoire.

- **Quantité** : 17 468 transistors à tubes.
- **Taille** : $\approx$ 10 cm par tube.
- **Volume** : Remplit un hangar de **167 m²** (30 tonnes).
- **Énergie** : Consommation équivalente à une **petite ville**.
- **Calcul** : $\approx$ 1 000 opérations / sec (une calculette).

## 2021 : L'ère du silicium (Apple M1)

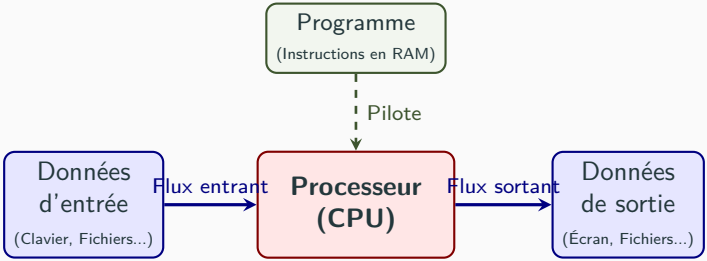
La révolution de la miniaturisation moderne sur une seule puce.

- **Quantité** : **16 milliards** de transistors.
- **Taille** : **5 nanomètres** (échelle atomique).
- **Volume** : Une puce de quelques millimètres.
- **Énergie** : Consommation de seulement **deux ampoules LED**.
- **Calcul** : **2,6 milliards de fois** plus rapide que l'ENIAC.

# Qu'est-ce qu'un ordinateur ?

**Un ordinateur est une machine automatique qui traite des données d'entrée, au moyen d'un programme, exécuté par un processeur, pour produire des données de sortie.**

Un programme est un ensemble d'instructions traitées de manière séquentielle, au rythme de l'horloge du processeur.



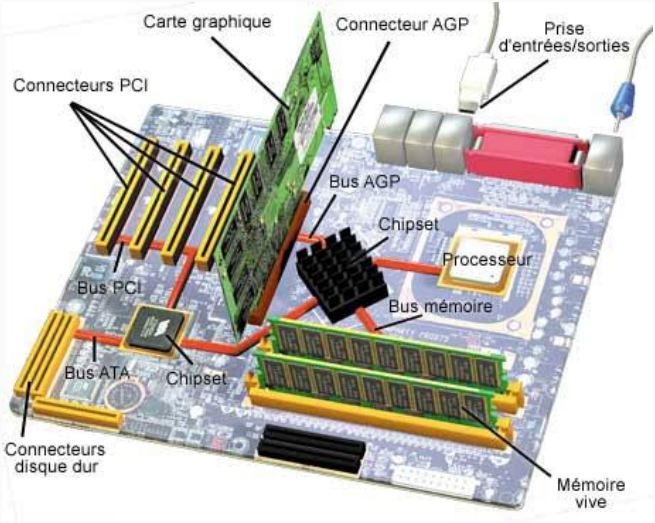
# L'architecture matérielle d'un ordinateur

# Les composants matériels internes d'un PC

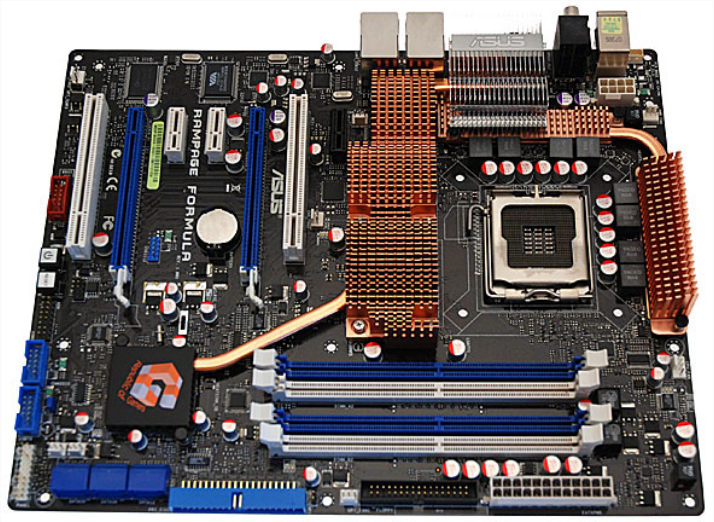
Un ordinateur moderne est un assemblage de composants spécialisés connectés sur un circuit imprimé central appelé la **carte mère**. Chaque composant interne assure un rôle précis de traitement ou de stockage.

- **Processeur (CPU)** : le cerveau, exécute les instructions logiques du code.
- **Mémoire RAM** : mémoire de travail volatile, rapide, connectée directement au CPU.
- **Stockage (SSD/HDD)** : mémoire de masse persistante (fichiers, système d'exploitation).
- **Carte graphique (GPU)** : dédiée aux calculs parallèles et à l'affichage.
- **Carte son** : décodage, conversion numérique / analogique du son, mixage, pré-amplification.
- **Alimentation (PSU)** : convertit et distribue l'énergie électrique.
- **Périphériques d'E/S** : USB, Firewire, Ethernet, Wifi, Bluetooth...

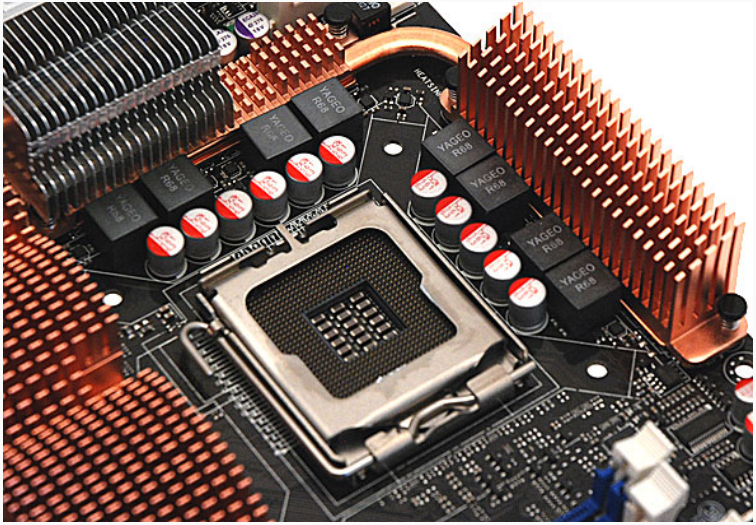
# Eléments notables d'une carte mère



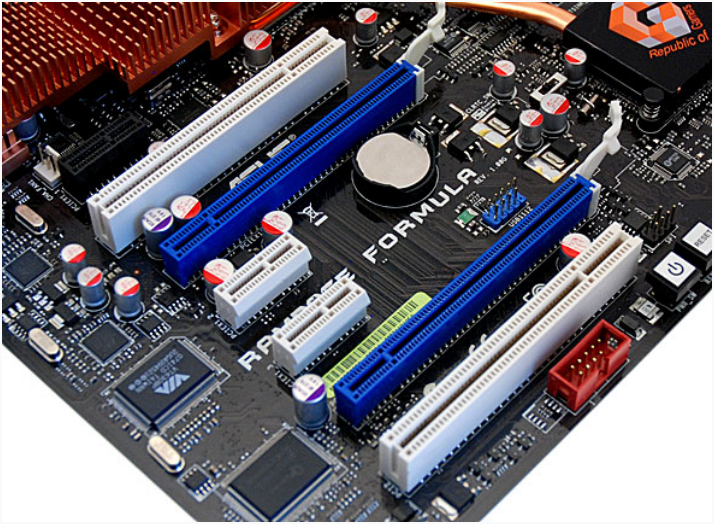
# Carte mère



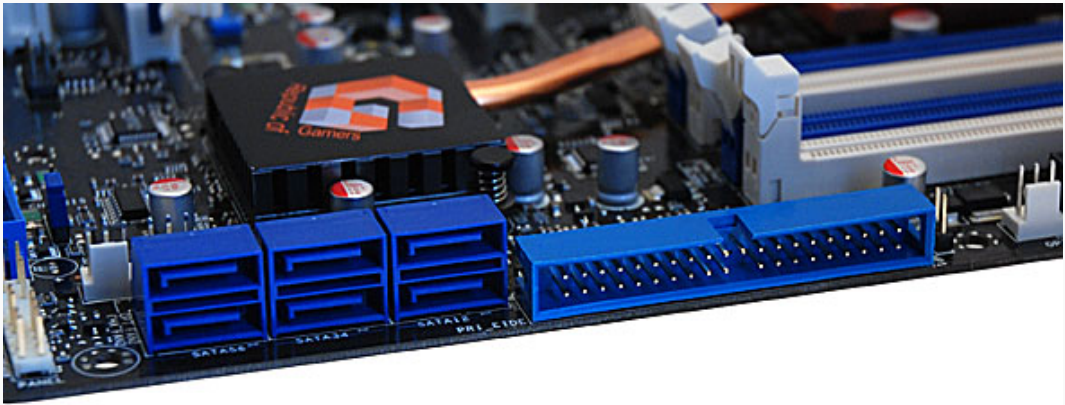
# Socket processeur



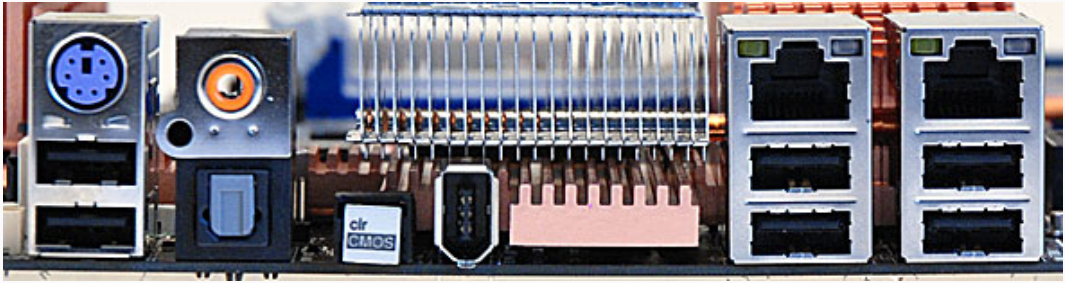
# Ports PCI-Express



# Ports SATA et port IDE



# Connecteurs d'entrées et sorties



# Communication interne : chipset et bus

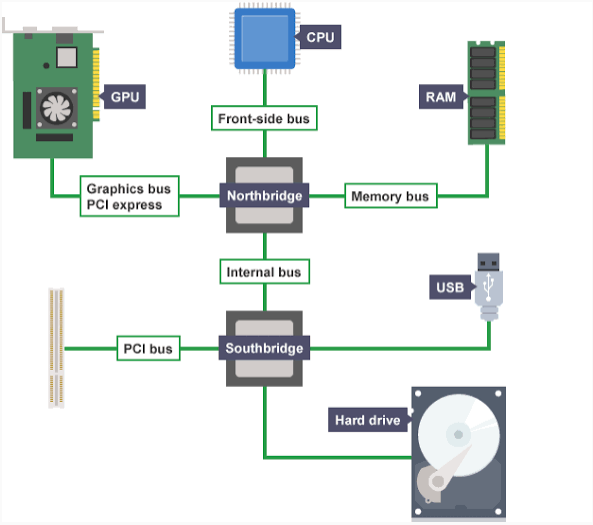
**Le chipset (le chef d'orchestre)**

Circuit(s) intégré(s) à la carte mère qui aiguille, régule et contrôle les flux de données entre le CPU et tous les autres composants (RAM, SSD, Carte graphique, périphériques d'E/S).

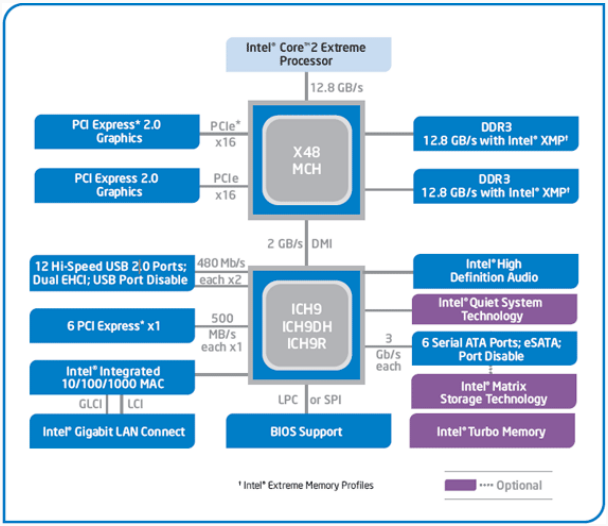
Le processeur communique avec la mémoire et le système via trois bus distincts :

- **Bus de données** : transporte les données d'un point à un autre par des fils.
- **Bus d'adresses** : sélectionne l'adresse mémoire de départ et d'arrivée.
- **Bus de contrôle** : commande la lecture, l'écriture, ou l'interruption.

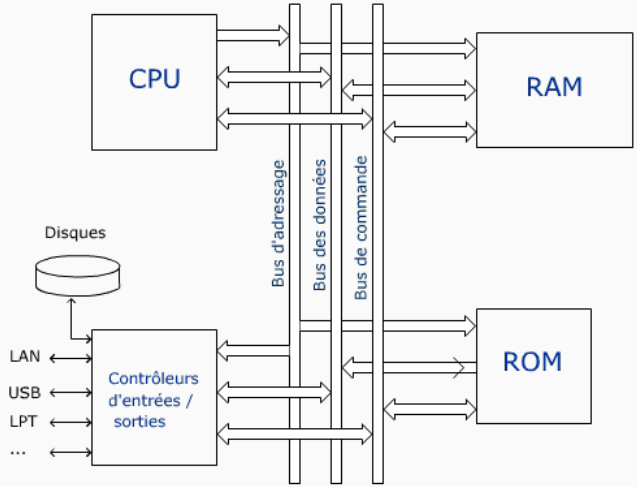
# Chipset carte mère



# Chipset Intel

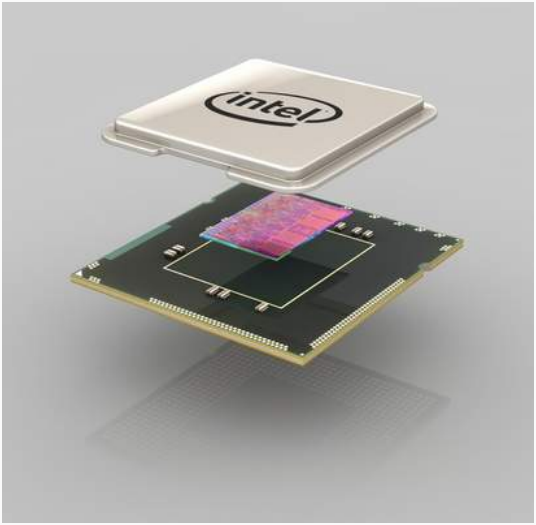


# Vue schématique autour des BUS



# Boitier processeur

# Constitution du boitier



# Constitution physique d'un processeur

### Le silicium et la gravure

Un processeur est un morceau de silicium pur (une *die*) sur lequel sont gravés des milliards de transistors formant des circuits logiques par photolithographie.

### Le transistor (l'atome logique)

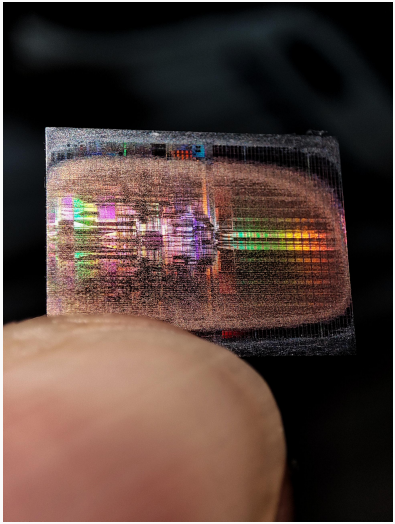
Composant de base du CPU. Il fonctionne comme un interrupteur électronique microscopique : laisser passer ou bloquer le courant, ce qui représente le 1 ou le 0 binaire.

Le CPU, l'objet physique que l'on manipule dans nos mains comporte 3 éléments :

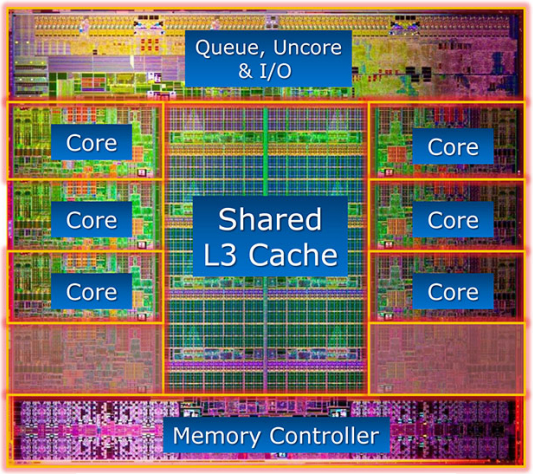
- **Le Die** : la puce de silicium contenant les circuits logiques.
- **L'IHS** (*Integrated Heat Spreader*) : capsule métallique qui protège la puce et dissipe la chaleur vers le refroidissement.
- **Le PCB et les broches** : plaque de support inférieure munie de connecteurs dorés (pins ou pastilles) pour s'insérer dans le socket.

# Le Die du processeur

# Focus sur le Die



# Blocs fonctionnels du Die



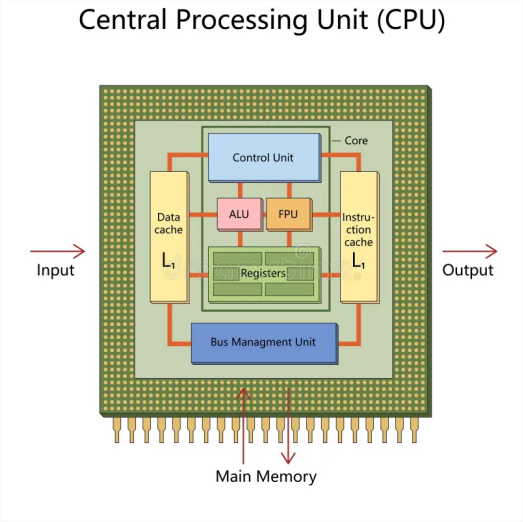

Intel® Core™ i7-3960X Processor Die Detail

# Les blocs fonctionnels d'un processeur

Pour traiter l'information, l'intérieur d'un cœur de processeur est divisé en sous-systèmes spécialisés qui collaborent à chaque cycle d'horloge :

- **Control unit (UC)** : le chef d'orchestre. Charge les instructions depuis la RAM, les décode et pilote les autres blocs.
- **Arithmétique logic unit (ALU)** : Opère les nombres entiers.
- **Floating point unit (FPU)** : Opère les nombres réels.
- **Les registres** : mémoire ultra-rapide interne (quelques octets). Stockent les données immédiates sur lesquelles l'ALU travaille.
- **La mémoire cache (L1, L2, L3)** : mémoire intermédiaire rapide qui stocke les données fréquemment utilisées pour éviter d'attendre la RAM.

# Schéma fonctionnel du CPU



# Le cycle d'instruction : Fetch – Decode – Execute

Le processeur travaille de manière cyclique, cadencé par une horloge interne. Chaque instruction d'un programme compilé passe par trois étapes matérielles :

- **1. Chargement (*Fetch*)** : l'unité de contrôle récupère l'instruction machine à l'adresse indiquée par le pointeur d'instruction (registre *Program Counter*).
- **2. Décodage (*Decode*)** : l'unité de contrôle traduit l'instruction binaire brute pour comprendre l'action à mener (addition, saut de boucle, lecture RAM, etc.).
- **3. Exécution (*Execute*)** : l'ALU/FPU effectue le calcul, met à jour les registres internes ou écrit le résultat en RAM.
- **3. Ecriture (*Write back*)** : le résultat de l'instruction, est enregistré en mémoire.

# Qui fait quoi dans l'industrie du silicium ?

Il est crucial de distinguer les entreprises qui **conçoivent** l'architecture logique des puces de celles qui possèdent les usines pour les **fabriquer physiquement**.

### Les concepteurs (*fabless*)

Ils dessinent les plans mais n'ont pas d'usines :

- **AMD** (PC, serveurs, consoles)
- **Apple** (puces A et M pour Mac/iPhone)
- **Nvidia** (GPU, intelligence artificielle)
- **Qualcomm** / **MediaTek** (mobile)

### Les fabricants (*fondeurs*)

Ils possèdent les usines de gravure nanométrique :

- **TSMC** (Taïwan) : leader mondial, grave pour Apple, AMD, Nvidia.
- **Intel** : modèle mixte, conçoit ses puces x86 et possède ses usines.
- **Samsung Foundry** (Corée).

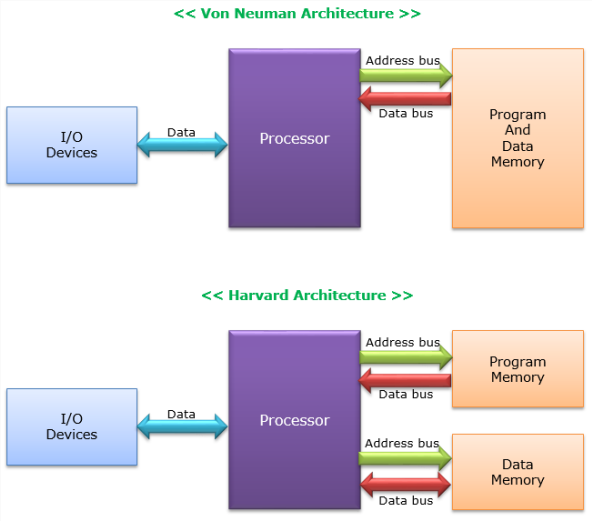
# Von Neumann vs Harvard

L'architecture définit la façon dont le processeur et la mémoire sont agencés pour exécuter un programme. Le monde informatique s'est structuré autour de grands concepts fondamentaux :

**1. Architecture de Von Neumann**
La mémoire des instructions et la mémoire des données sont considérées comme unifiées et partagent **un même bus d'adresses** et **un même bus de données**.
*Conséquence :* le processeur ne peut pas lire une instruction et accéder à une variable simultanément (*goulot d'étranglement de Von Neumann*).

**2. Architecture de Harvard**
La mémoire des instructions et la mémoire des données sont considérées comme séparées, **physiquement distinctes** et possèdent chacune leurs propres bus dédiés.
*Conséquence :* le CPU peut lire une instruction et manipuler une variable simultanément.

# Structures de Von Neumann et Harvard



# L'architecture des PC modernes : Un modèle hybride

**Vu par le programmeur (Logique) :**

À l'échelle globale, les PC d'aujourd'hui suivent le modèle de **Von Neumann**.

Le système d'exploitation charge le code et les variables dans une seule et unique mémoire partagée : la **RAM principale**.

**Vu par le processeur (Physique) :**

Au cœur même du CPU, les PC fonctionnent selon le modèle de **Harvard** pour maximiser les performances.

Les mémoires caches ultra-rapides intégrées au processeur sont physiquement séparées.

**La solution : L'architecture Harvard modifiée**

Pour briser le goulot d'étranglement historique, les processeurs modernes intègrent :

- Un cache **L1 Instructions** (dédié uniquement au code machine à exécuter).
- Un cache **L1 Données** (dédié uniquement aux variables).

*Bilan :* Le processeur possède des bus internes distincts et accède simultanément au code et aux données au niveau des caches (Harvard), tout en conservant une mémoire globale unifiée en RAM (Von Neumann).

# La hiérarchie des mémoires caches : L1, L2 et L3

**Le problème de la RAM :**

Le processeur travaille à une vitesse extrême (nanoseconde), tandis que la mémoire RAM principale est beaucoup plus lente. Aller chercher une donnée en RAM bride le CPU.

**Le principe des caches :**

Des mémoires intermédiaires ultra-rapides, intégrées au processeur, stockent les données les plus fréquemment utilisées pour anticiper les besoins du CPU.

| Niveau | Taille | Vitesse | Emplacement | Partage |
|---|---|---|---|---|
| **Cache L1** | $\approx$ 32 - 96 Ko | Ultra-rapide | Au plus près des circuits | Propre à chaque |
| **Cache L2** | $\approx$ 512 Ko - 2 Mo | Très rapide | Interne au cœur | Propre à chaque |
| **Cache L3** | $\approx$ 16 - 96 Mo | Rapide | Au centre de la puce | **Partagé** par tous |
| *Mémoire RAM* | *Plusieurs Go* | *Lente (relative)* | *Barrettes externes* | *Global (Système)* |

**Le fonctionnement (Harvard) :** Le cache L1 est le seul à être physiquement divisé en deux sections autonomes : le cache **L1-I** (Instructions) et le cache **L1-D** (Données).

# La hiérarchie de la mémoire : Temps d'accès

Plus une mémoire est proche du processeur, plus elle est rapide, coûteuse et de faible capacité. L'écart de vitesse entre le haut et le bas est **gigantesque**.

## Dans le Processeur (Ultra-rapide)

- **Registres** $\rightarrow$ **$<$ 1 ns**
  *Équivalent humain :* 1 seconde.
- **Cache L1 (I/D)** $\rightarrow \sim$ **1 ns**
  *Équivalent humain :* 1 seconde.
- **Cache L2** $\rightarrow \sim$ **4 ns**
  *Équivalent humain :* 4 secondes.
- **Cache L3** $\rightarrow \sim$ **12 ns**
  *Équivalent humain :* 12 secondes.

## Hors du Processeur (Plus lent)

- **Mémoire RAM** $\rightarrow \sim$ **60 - 100 ns**
  *Équivalent humain :* 1,5 à 2 minutes.
- **SSD (NVMe)** $\rightarrow \sim$ **10 000 ns** (10 $\mu$s)
  *Équivalent humain :* 3 heures !
- **Disque dur (HDD)** $\rightarrow \sim$ **10 ms**
  *Équivalent humain :* 4 mois !!!

**Bilan : Si le CPU met 1s à lire un registre, aller chercher une info sur un SSD équivaut pour lui à attendre 3 heures !**

# Du langage machine au langage évolué

# Le jeu d'instructions (ISA) : l'alphabet du CPU

## Qu'est-ce qu'une ISA ?

L'*Instruction Set Architecture* est la frontière entre le logiciel et le matériel.

- Ensemble des opérations binaires élémentaires.
- Ce que le processeur sait exécuter physiquement.

## Une frontière matérielle

Chaque famille de CPU possède sa propre interface, câblée électroniquement.

- Dictée par l'agencement des transistors.
- Un CPU ne comprend **que** son ISA.

**Les grands standards actuels :**

- **x86 / x86-64 (CISC)** → Standard historique d'Intel & AMD (PC, serveurs lourds).
- **ARM (RISC)** → Ultra-économe. Standard des smartphones d'Apple Silicon.
- **RISC-V (RISC)** → Architecture moderne open-source et libre de droits en plein essor.

# CISC vs RISC : Deux philosophies d'instructions

## CISC (*Complex*)

**Philosophie :** Le matériel fait le travail lourd avec des instructions denses.

**Fiche technique :**

- **Instructions :** Rares mais complexes
- **Cycles/Inst. :** Variables (1 à plusieurs)
- **Énergie :** Consommation plus élevée
- **Code machine :** Très compact

*Standard :* **x86 / x64** (Intel, AMD)
→ PC, Serveurs

## RISC (*Reduced*)

**Philosophie :** Le compilateur assemble des instructions simples et rapides.

**Fiche technique :**

- **Instructions :** Multiples mais simples
- **Cycles/Inst. :** Fixes (1 cycle par inst.)
- **Énergie :** Très faible consommation
- **Code machine :** Plus volumineux

*Standard :* **ARM**, **RISC-V**
→ Smartphones, Apple M, Embarqué

# L'architecture ARM : efficacité et révolution

### Le modèle RISC d'ARM

ARM repose sur une architecture **RISC**. Contrairement au x86/x64 (**CISC**) qui cherche la puissance brute via des instructions complexes, ARM exécute des tâches simples très rapidement.

### Le succès récent

Historiquement réservé aux smartphones pour sa **très faible consommation**, ARM a conquis les PC et serveurs (Apple Silicon M, AWS Graviton) avec un rapport puissance/consommation inégalé.

### Une particularité économique majeure : la vente de licences

L'entreprise ARM ne fabrique **aucun processeur**. Elle conçoit l'architecture et vend les plans (licences) à d'autres entreprises (Apple, Qualcomm, Samsung) qui **personnalisent** cette architecture pour leurs propres puces avant de les faires fabriquer par des fondeurs.

# Le langage assembleur : traduction humaine de l'ISA

### Le code machine brut

Pour le processeur, une instruction n'est qu'une suite de 0 et de 1 (ex : `00000101`). Ce code machine est illisible pour un humain.

### Le langage assembleur

L'assembleur n'étant que la copie textuelle de l'ISA, le code est **propre à chaque processeur**. Un programme en assembleur x64 (AMD, Intel) est totalement incompréhensible pour un processeur ARM (Apple M).

### Le rôle de l'assembleur

Le langage assembleur associe un mot-clé textuel (un *mnémonique*) à chaque instruction binaire de l'ISA.

Exemple Assembleur (x64)

```
1 mov eax, 5  ; Charge 5 dans EAX
2 add eax, 3  ; Ajoute 3 dans EAX
```

# Les langages évolués et la portabilité du code

## 1. L'abstraction matérielle

Les langages évolués (comme le **C**) masquent la complexité physique.

- Logique humaine
- Indépendant du matériel et de l'ISA

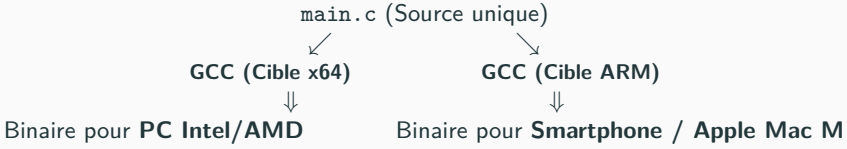
## 2. Le rôle du compilateur

Le **compilateur** (ex : GCC) traduit cette logique vers la machine cible.

- Génère l'ISA spécifique
- Adapte le code au matériel

## La Portabilité : Le grand intérêt du C

Un **unique fichier source** (`main.c`) peut être traduit pour différents ordinateurs et processeurs sans modifier le code écrit (c'est d'ailleurs la raison de son invention).



# De l'humain à la machine : Les niveaux d'abstraction

### 1. Langage de haut niveau (Code source en C)

```c
int a = 5;
int b = 10;
int result = a + b;
```

↓ *Compilation (génère l'assembleur)*

### 2. Langage de bas niveau (Assembleur x86-64)

```
movl $5, -4(%rbp)  ; Stocker 5 dans la variable a
movl $10, -8(%rbp) ; Stocker 10 dans la variable b
addl -8(%rbp), %eax ; Additionner a et b
```

↓ *Assemblage (génère les opcodes machine)*

### 3. Code machine (Binaire exécuté par le processeur)

```
C7 45 FC 05 00 00 00 -> 11000111 01000101 11111100 00000101
C7 45 F8 0A 00 00 00 -> 11000111 01000101 11111000 00001010
```

# Système d'exploitation

# Le Système d'Exploitation : Shell et Kernel

Un système d'exploitation (OS) agit comme un intermédiaire entre l'utilisateur et le matériel. Il est structurellement divisé en deux grandes couches logicielles complémentaires.

## Le Kernel (Le Noyau)

C'est le cœur invisible du système d'exploitation, connecté directement au matériel (*Hardware*).

- **Rôle** : Gère les ressources physiques (RAM, temps CPU, accès au disque).
- **Sécurité** : Protège le matériel en contrôlant strictement les droits d'accès des programmes.
- *Exemple* : Le noyau Linux, le noyau NT (Windows).

## Le User Space (Espace utilisateur)

C'est la couche externe qui entoure le noyau. Le Shell et les applications s'y exécutent au même niveau.

- **Les Shells (CLI / GUI)** : Applications d'interface (Terminal ou Bureau) permetant à l'utilisateur de lancer d'autres applications.
- **Les Applications** : Les programmes lancés (navigateurs, jeux, etc.). Elles communiquent en direct avec le noyau, **sans passer par le Shell**.

# Architecture : Terminal, Shell textuel et Shell graphique

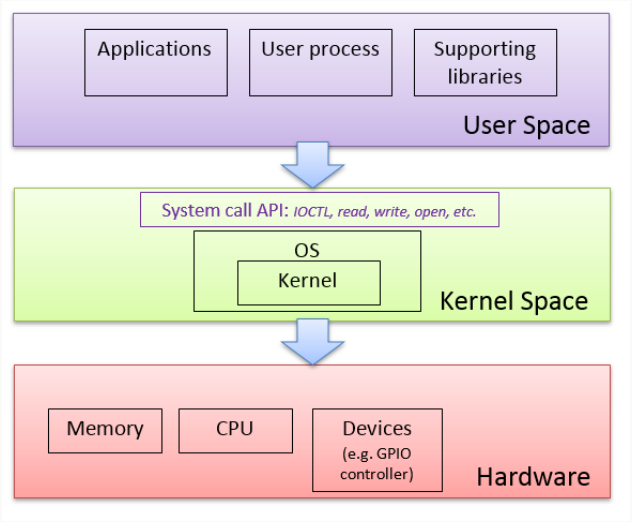
**Le concept de Shell (Coquille) :**

En informatique, un **Shell** désigne de manière générale la couche logicielle externe qui entoure le cœur du système (le noyau) afin de servir d'interface avec l'être humain.

**Les trois visages de l'interface utilisateur :**

- **Le Terminal** : C'est uniquement l'enveloppe visuelle graphique (la fenêtre). Son seul rôle est de capturer les touches du clavier et d'afficher du texte à l'écran.
- **Le Shell textuel (CLI)** : Le moteur logique qui tourne *à l'intérieur* de la fenêtre (ex : `Bash`, `Zsh`). Il interprète les commandes tapées au clavier.
- **Le Shell graphique (GUI)** : L'environnement de bureau complet (ex : `GNOME Shell` sous Linux, `explorer.exe` sous Windows, `Finder` sous Mac). Il traduit les clics de souris et les fenêtres en ordres pour le système.

# Représentation en couches de l'OS



# En résumé

Le système d'exploitation sert de plateforme aux applications en leur mettant à disposition les ressources nécessaires (via des appels systèmes) et en contrôlant leur exécution.

**Le noyau (Kernel)**

Le noyau (Kernel) a un niveau de privilège matériel où toutes les instructions CPU et tout l'espace mémoire sont accessibles.

**L'espace utilisateur (User Space)**

Les applications, que ce soit le shell (terminal ou bureau graphique) ou les applications utilisateurs (navigateur, jeux, éditeurs de textes...) s'exécutent avec des droits restreints. Pour accéder au matériel (ex : accéder à la RAM, écrire dans un fichier etc.), elles doivent obligatoirement passer par des fonctions du noyau nommées "Appels Systèmes" (Syscalls).

# Pilier 1 : La Gestion du Processeur et des Processus

Un processeur possède un nombre limité de cœurs, mais l'OS doit faire tourner des centaines de programmes en même temps. Il crée l'illusion de la simultanéité.

## Le concept de Processus

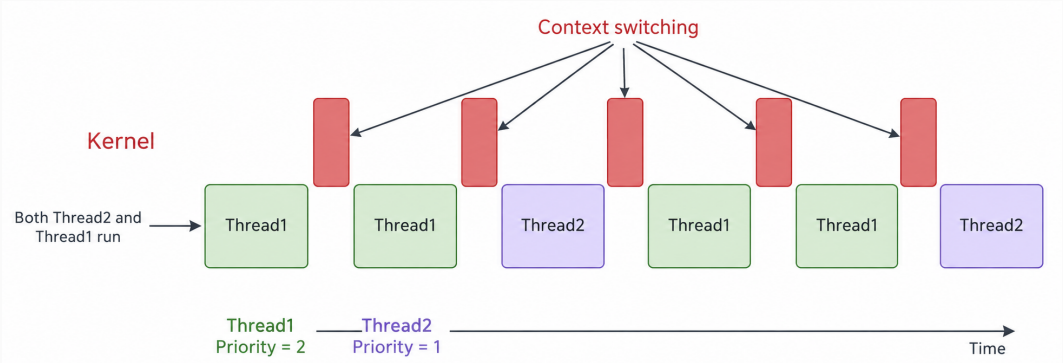
Un processus est l'instance d'un programme en cours d'exécution. L'OS lui associe un espace mémoire isolé, des descripteurs de fichiers et un contexte CPU.

## L'Ordonnanceur (*Scheduler*) et le Temps Partagé

L'OS distribue le temps de calcul du CPU milliseconde par milliseconde :

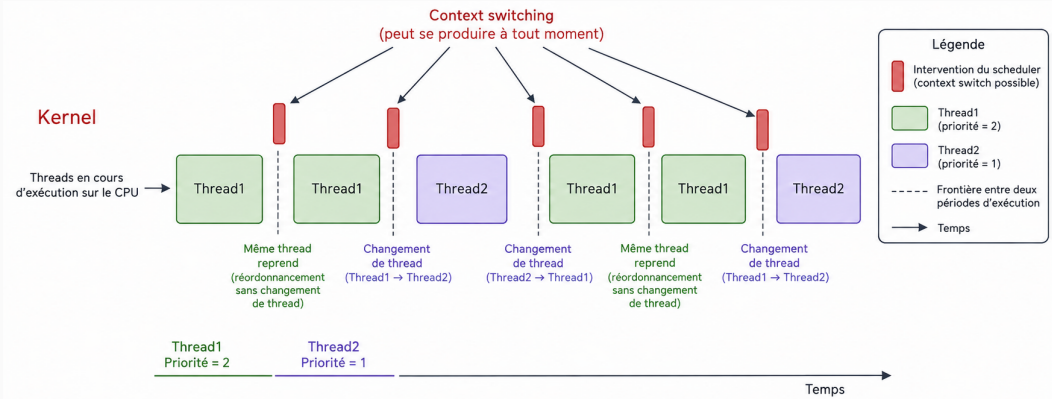
- **Commutation de contexte (*Context Switch*)** → L'OS sauvegarde l'état exact des registres du CPU pour un processus, l'interrompt, et charge les registres d'un autre processus.
- **Ordonnancement préemptif** → Le noyau a le pouvoir d'interrompre de force un programme qui monopolise le processeur pour donner la main aux autres.

# Ordonnanceur, préemption et sauvegarde de contexte




**The Normal Linux Kernel Scheduler**

# Ordonnanceur, préemption et sauvegarde de contexte




## The Normal Linux Kernel Scheduler

Le scheduler peut intervenir (préempter) à tout moment :

- il peut choisir de relancer le même thread (réordonnancement sans changement de thread), ou
- il peut choisir un **autre thread** (changement de thread).

# Pilier 2 : La Gestion de la Mémoire RAM

L'OS est le gardien de la mémoire vive. Il doit s'assurer qu'aucun programme ne puisse lire ou corrompre les données d'un autre programme ou du noyau lui-même.

## Le mécanisme de Mémoire Virtuelle

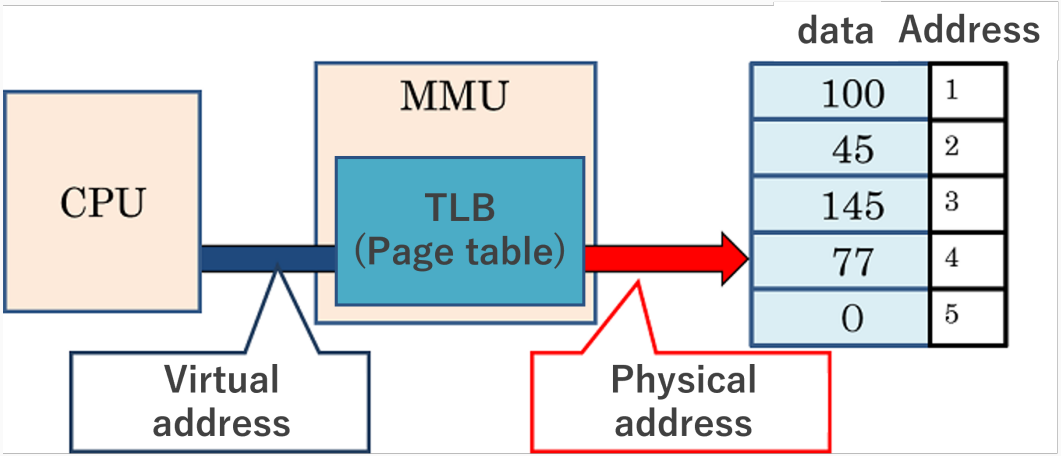
Les applications ne manipulent jamais les adresses physiques réelles de la RAM.

- Chaque processus croit disposer d'un espace mémoire continu et gigantesque.
- L'OS travaille avec l'unité matérielle du CPU (**MMU** - *Memory Management Unit*) pour traduire à la volée les adresses virtuelles en adresses physiques.

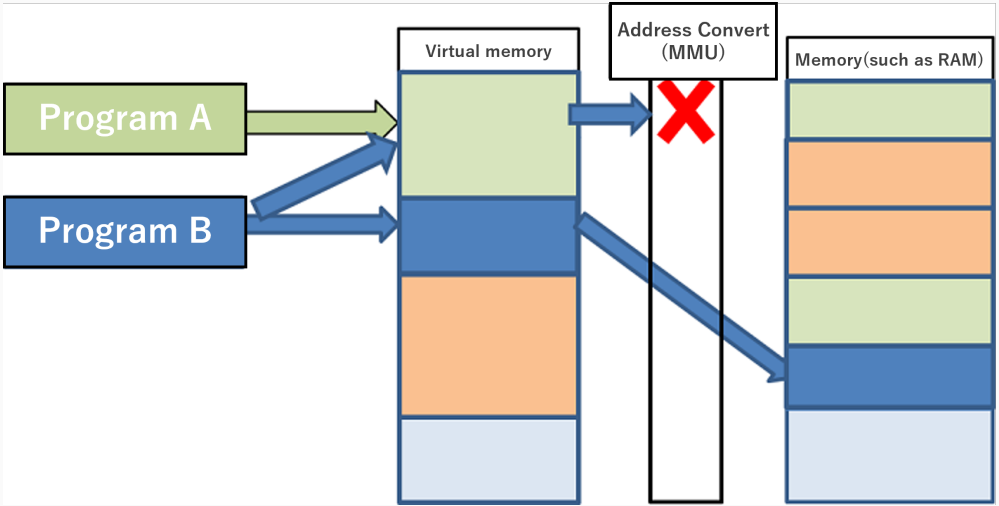
## Le Partitionnement et la Protection

- **Pagination** → Découpage de la mémoire en blocs appelés "pages".
- **Segmentation fault** → Si un processus tente d'accéder à une adresse en dehors de sa table de pages autorisée, la MMU lève une exception et l'OS tue immédiatement le programme fautif.

# Abstraction de l'arborescence de fichiers



# Protection contre les sorties de pages de mémoire



# Pilier 3 : Système de Fichiers

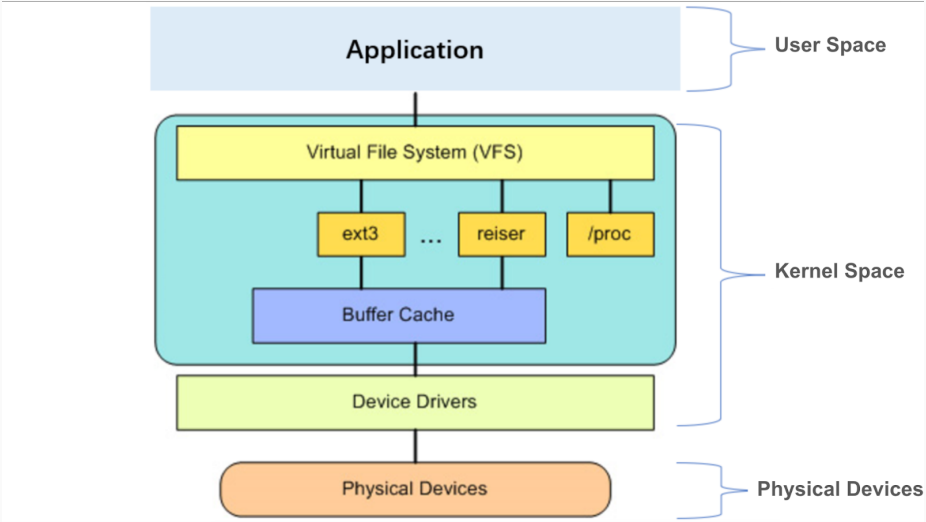
L'OS uniformise le fonctionnement du système de fichiers pour offrir un environnement standardisé.

### Le Système de Fichiers (*File System* - ext4, NTFS...)

Un disque (SSD/HDD) n'est physiquement qu'une immense suite de blocs de secteurs.

- L'OS crée l'abstraction de l'arborescence (dossiers, fichiers, noms).
- Il gère les métadonnées (droits de lecture/écriture, dates de modification) et la table d'allocation de l'espace pour éviter les chevauchements.

# Abstraction de l'arborescence de fichiers



# Pilier 4 : Pilotes matériels

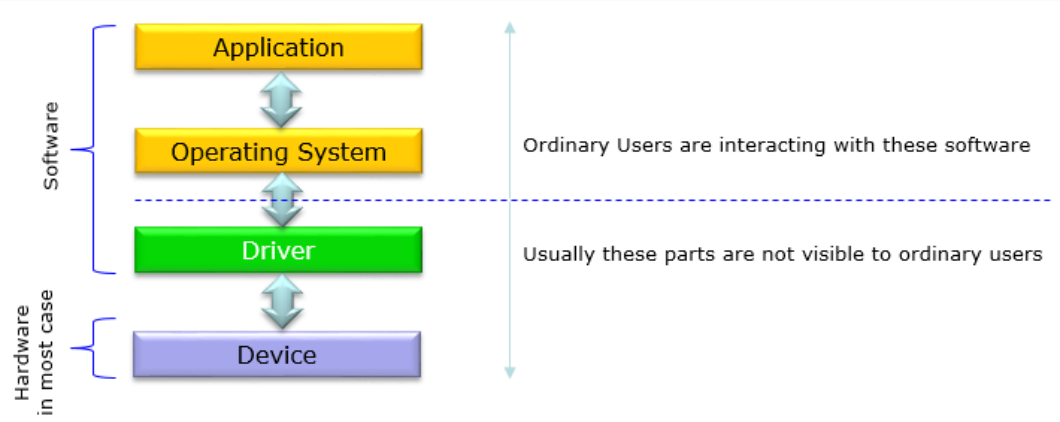
L'OS uniformise le fonctionnement des périphériques pour offrir un environnement de développement standardisé.

## Les Pilotes (*Drivers*) : L'unification du matériel

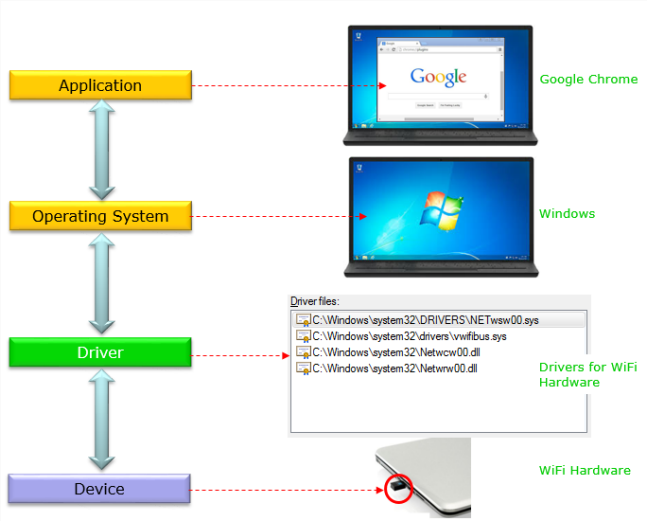
Un pilote est un module logiciel qui traduit les commandes génériques de l'OS en instructions électriques spécifiques au protocole d'un constructeur.

- *Exemple :* Pour l'OS, toutes les souris envoient des coordonnées $(X, Y)$. Le pilote se charge de traduire le signal brut du port USB.

# Abstraction de l'arborescence de fichiers



# Abstraction de l'arborescence de fichiers



# Mémoire vive

# Structure logique de la RAM : La matrice de cases

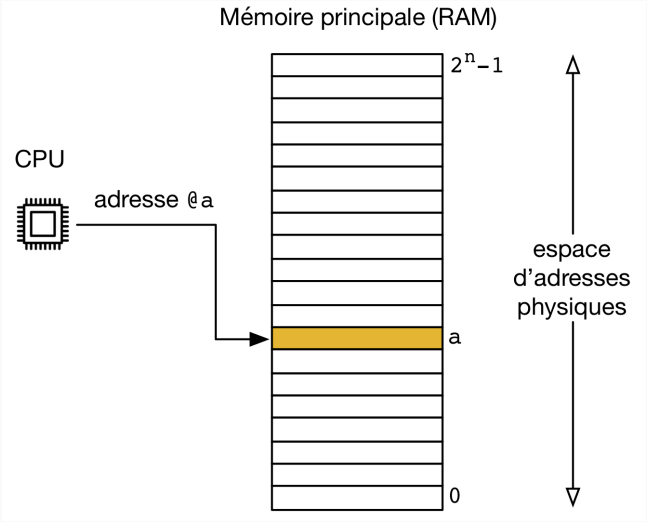
La RAM (*Random Access Memory*) est organisée logiquement comme un immense tableau excel à deux colonnes. Une colonne pour les **cases mémoires** contenant les données, une colonne pour l'**adresse** de chaque case.

| Bus d'adresse | Décodeur physique | Cases Mémoires (1 octet = 8 bits) |
|---|---|---|
| 0x0000 | $\rightarrow$ | [ 01101001 ] |
| 0x0001 | $\rightarrow$ | [ 11001110 ] |
| 0x0002 | $\rightarrow$ | [ 00001010 ] |
| 0x0003 | $\rightarrow$ | [ 11110001 ] |

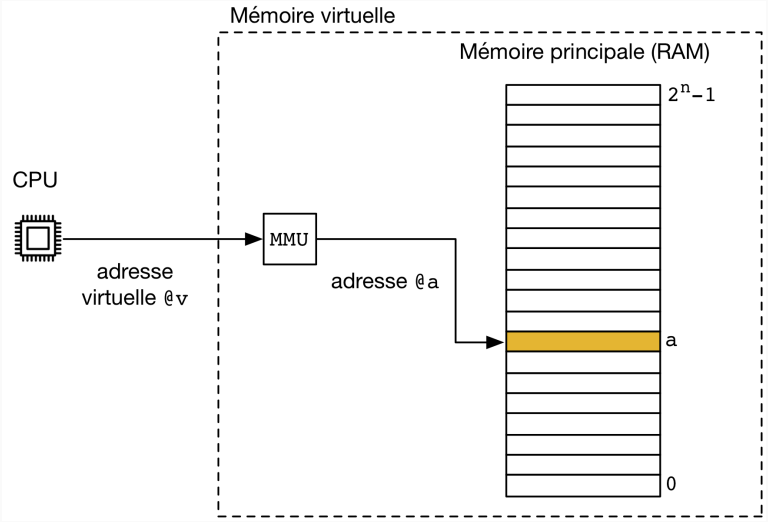
## Le mécanisme d'accès

1. **Indexation :** Le CPU dépose l'adresse de la case visée sur le décodeur du *Bus d'adresse*.
2. **Sélection :** Le décodeur du bus active la ligne correspondante.
3. **Transfert :** Les bits de données transitent via le *Bus de données* (lecture/écriture).

# Accès du CPU à la RAM



# Accès du CPU à la RAM avec un MMU



# La RAM : La taille des mots

## 1. La case mémoire et la taille des "mots"

La RAM est découpée en milliards de **cases mémoires** de taille identique.

- Chaque case stocke un groupe fixe de bits appelé un **"mot"** (*word*).
- Sur presque tous les ordinateurs, la taille standard d'une case est de **8 bits**, soit **1 octet**.

## 2. L'adresse mémoire

Pour pouvoir lire ou écrire dans une case, le processeur doit pouvoir la désigner. Chaque case possède donc un numéro unique et séquentiel : son **adresse** (la case 0, la case 1, la case 2, etc.).

# Capacité d'adressage et puissance de calcul

# Le lien avec les bits du processeur

Le nombre de bits d'un processeur (8, 16, 32 ou 64 bits) détermine la **longueur maximale du numéro de case (l'adresse)** qu'il est capable d'adresser.

## Le nombre de cases dépend du nombre de bits

Si un processeur écrit des numéros de cases sur $N$ bits, il ne peut générer que $2^N$ combinaisons. Il est limité à un **nombre maximal de cases** :

- **Processeur 8 bits** $\rightarrow$ Peut écrire des numéros de 8 chiffres binaires.
  Limite : $2^8 = \mathbf{256}$ **cases** maximum.
- **Processeur 32 bits** $\rightarrow$ Peut écrire des numéros de 32 chiffres binaires.
  Limite : $2^{32} = \mathbf{4\,294\,967\,296}$ **cases** $\rightarrow$ **4 Go** maximum.
- **Processeur 64 bits** $\rightarrow$ Peut écrire des numéros de 64 chiffres binaires.
  Limite : $2^{64} =$ **16 milliards de milliards de cases**.

**En résumé : Un processeur 32 bits ne peut pas utiliser plus de 4 Go de RAM, c'est la raison pour laquelle les processeurs sont passés à 64 bits au cours des années 2000.**

# Le lien avec la puissance de calcul

Le nombre de bits d'un processeur détermine aussi la taille des nombres que les unité arithmétiques et logiques (**ALU**) peuvent traiter **en une seule fois**.

### La taille des registres de calcul

Plus un processeur a de bits, plus ses "cases de travail" internes (**les registres**) sont larges. Il peut ainsi manipuler des nombres beaucoup plus grand en un seul coup d'horloge.

# Merci !

## Questions ?