ISG
BIZERTE
INSTITUT SUPÉRIEUR DE GESTION

Système d'Exploitation

Zaabi Radhouane
PH.student at CESLab. ENIS
zaabiradhouane24@gmail.com

Sujets traités:

***Chapitre 1** : Notions fondamentales des Systèmes d'exploitation*

***Chapitre 2** : La gestion du processeur*

***Chapitre 3** : Synchronisation des processus*

***Chapitre 4** : Allocation de la mémoire*

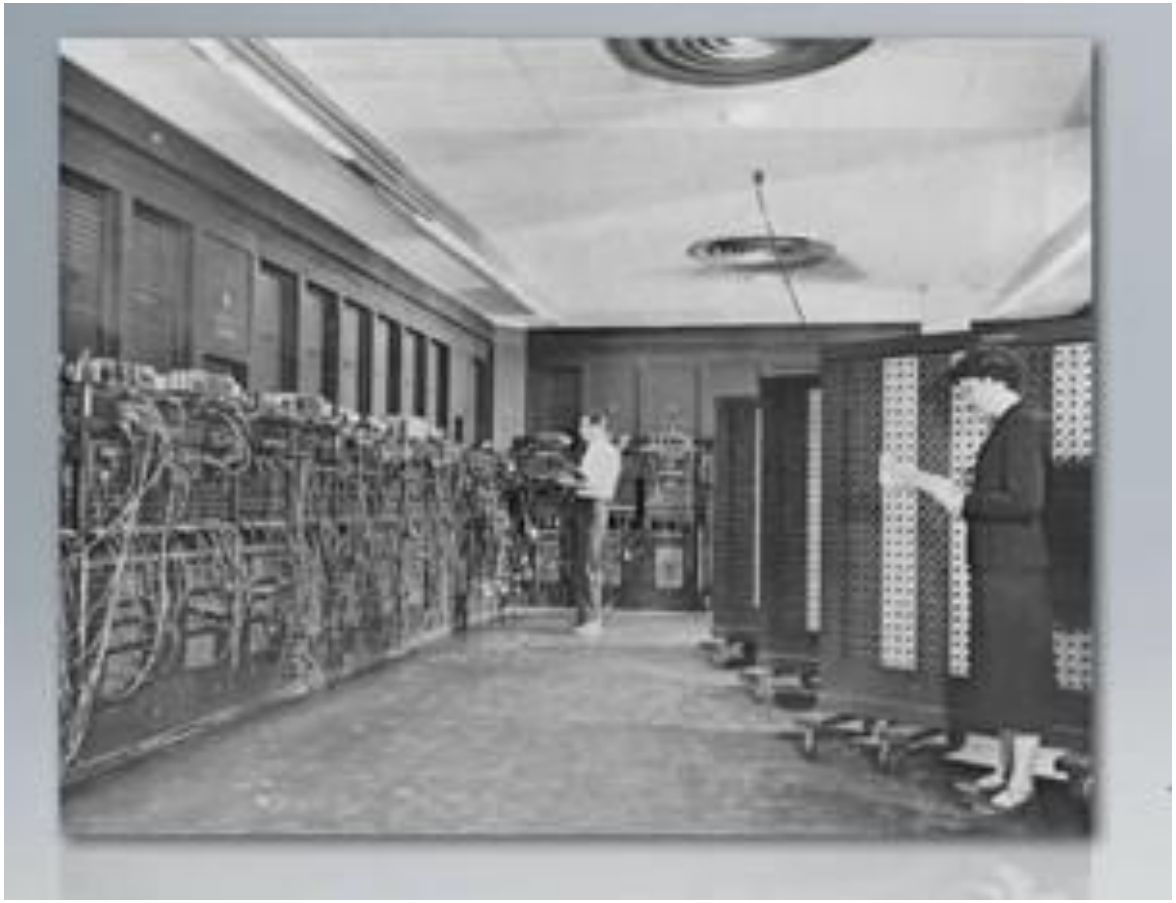
Plan

- Introduction à L'Informatique
- Architectures matérielles des ordinateurs
- La nécessité d'un système d'exploitation
- Les rôles du système d'exploitation
- Les Caractéristiques du système d'exploitation

Traitement automatique de l’information

ENIAC (1946)

Electronic Numerical Integrator Analyser and computer



Poids 30 tonnes

Surface 72 m2

330 multiplication/s

Ordinateur(computer)

- *Qu'est ce qu'un ordinateur ?*

Le mot « ordinateur » fut introduit par IBM France en 1955

Il veut dire : "ordonnateur" : celui qui met en ordre

Ordinateur

caractéristiques fondamentales nécessaires pour être considérées comme un ordinateur sont :

- qu'il soit électronique,
- numérique (au lieu d'analogique),
- qu'il soit programmable,
- qu'il puisse exécuter les quatre opérations élémentaires (addition, soustraction, multiplication, division)
- qu'il puisse exécuter des programmes enregistrés en mémoire.

Exemples d’ordinateur

Supercalculateur

Prévisions Météo
Simulations Physiques
La Recherche Scientifique
...

Serveur

Réseau, et Sites Internet
Les Centres de Données
Les Entreprise, Les Institutions
...

Ordinateur Individuel

Bureautiques
Conceptions, Gestions
Jeux, Loisirs
...

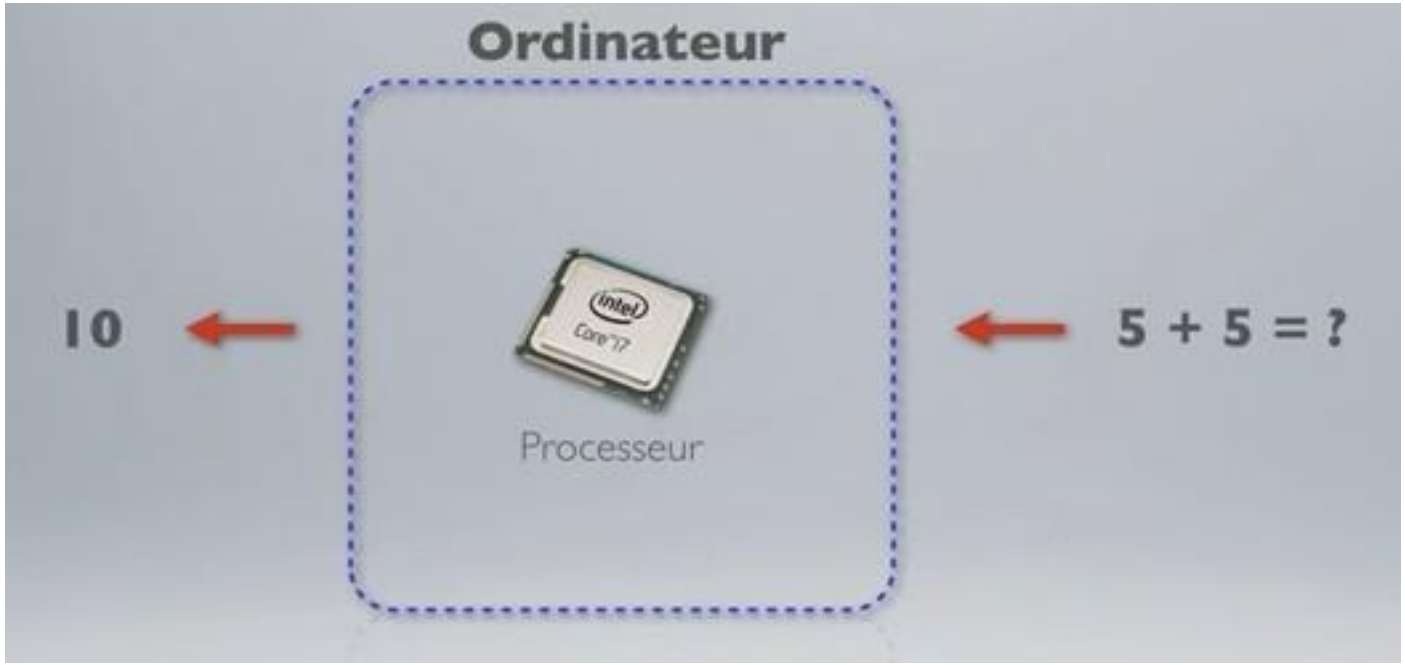
Architecture matérielle de l’ordinateur

Imitation du monde naturel

Imitation du monde naturel

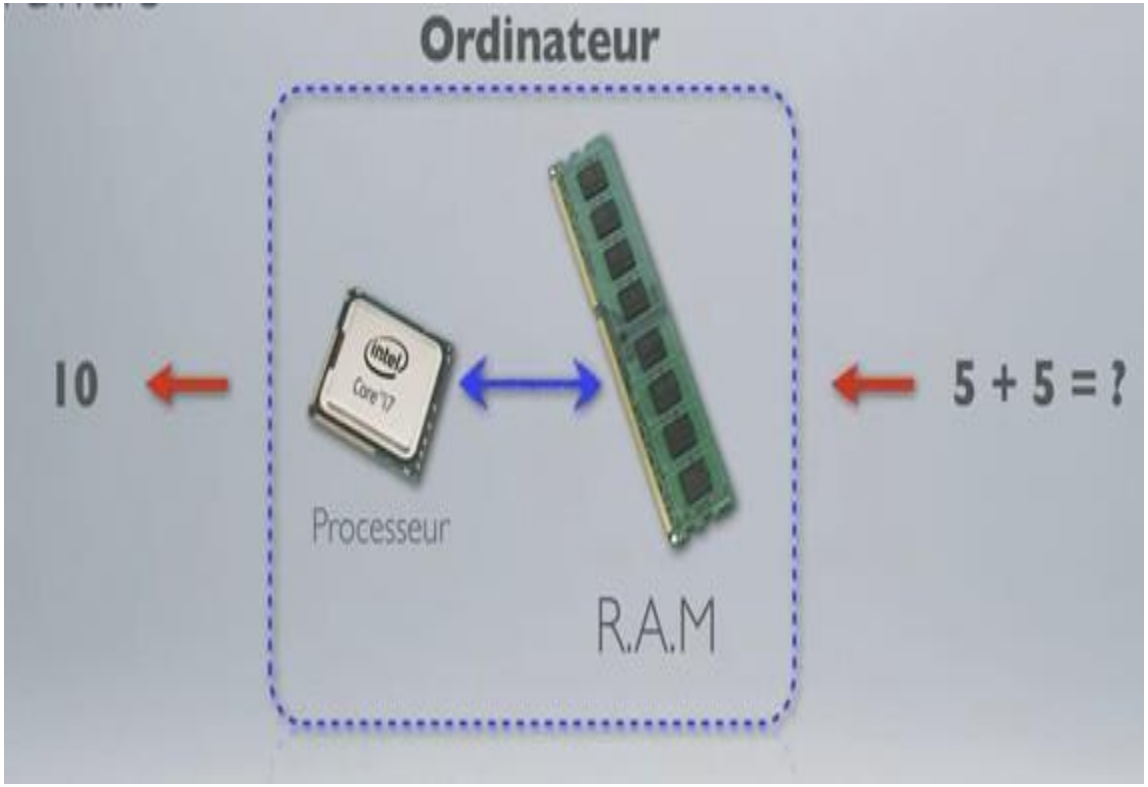
Processeur

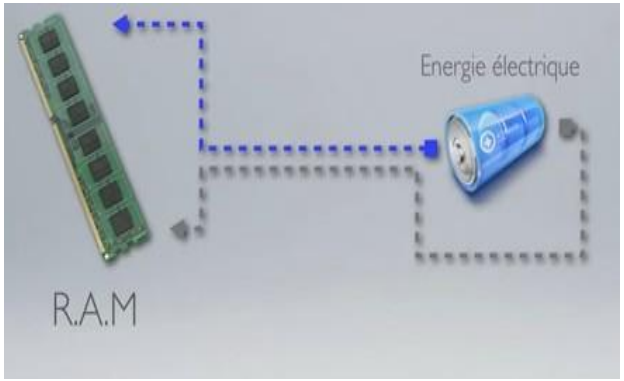
- Le composant qui exécute les programmes
- En anglais « **CPU** »



R.A.M: La mémoire Vive

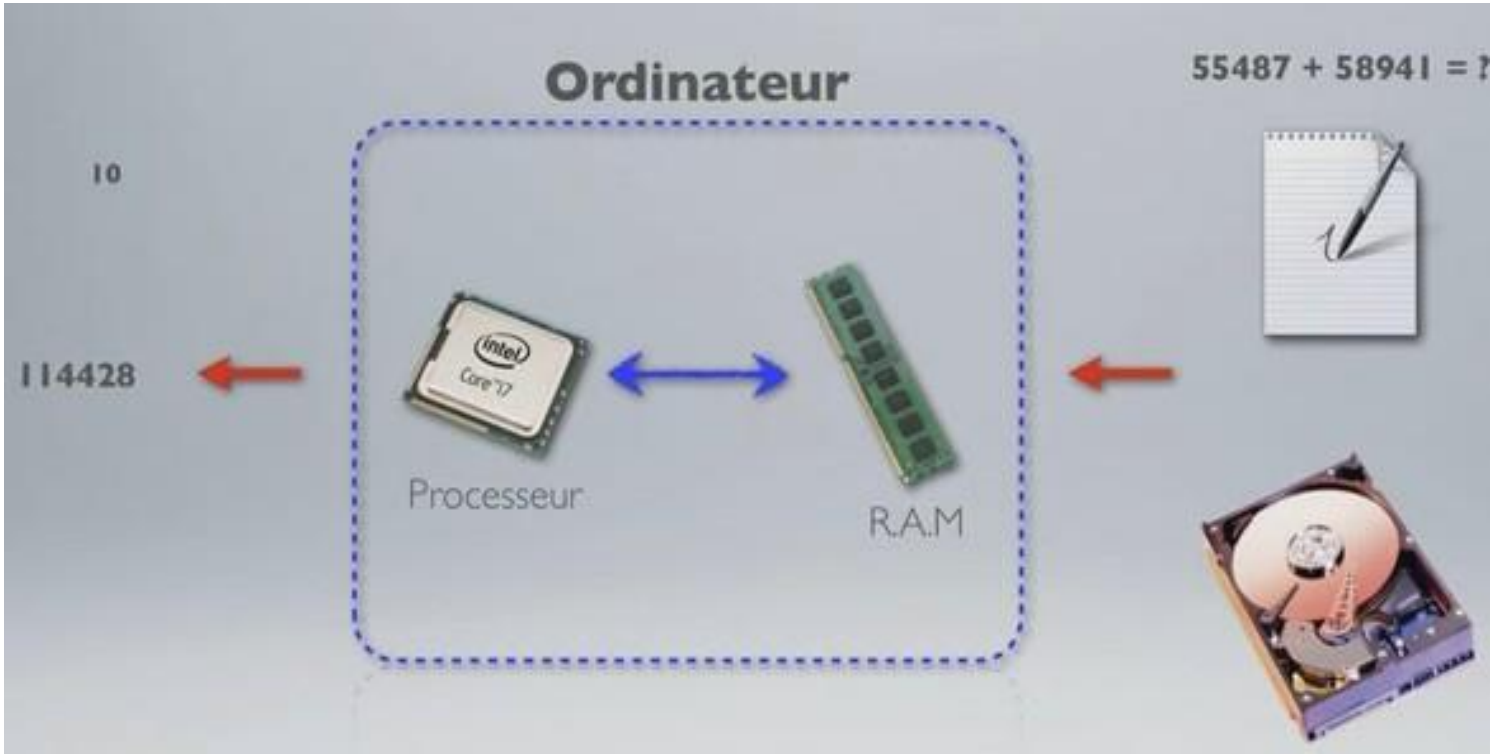
- **Random Access Memory**
- **R.A.M:Mémoire Volatile**



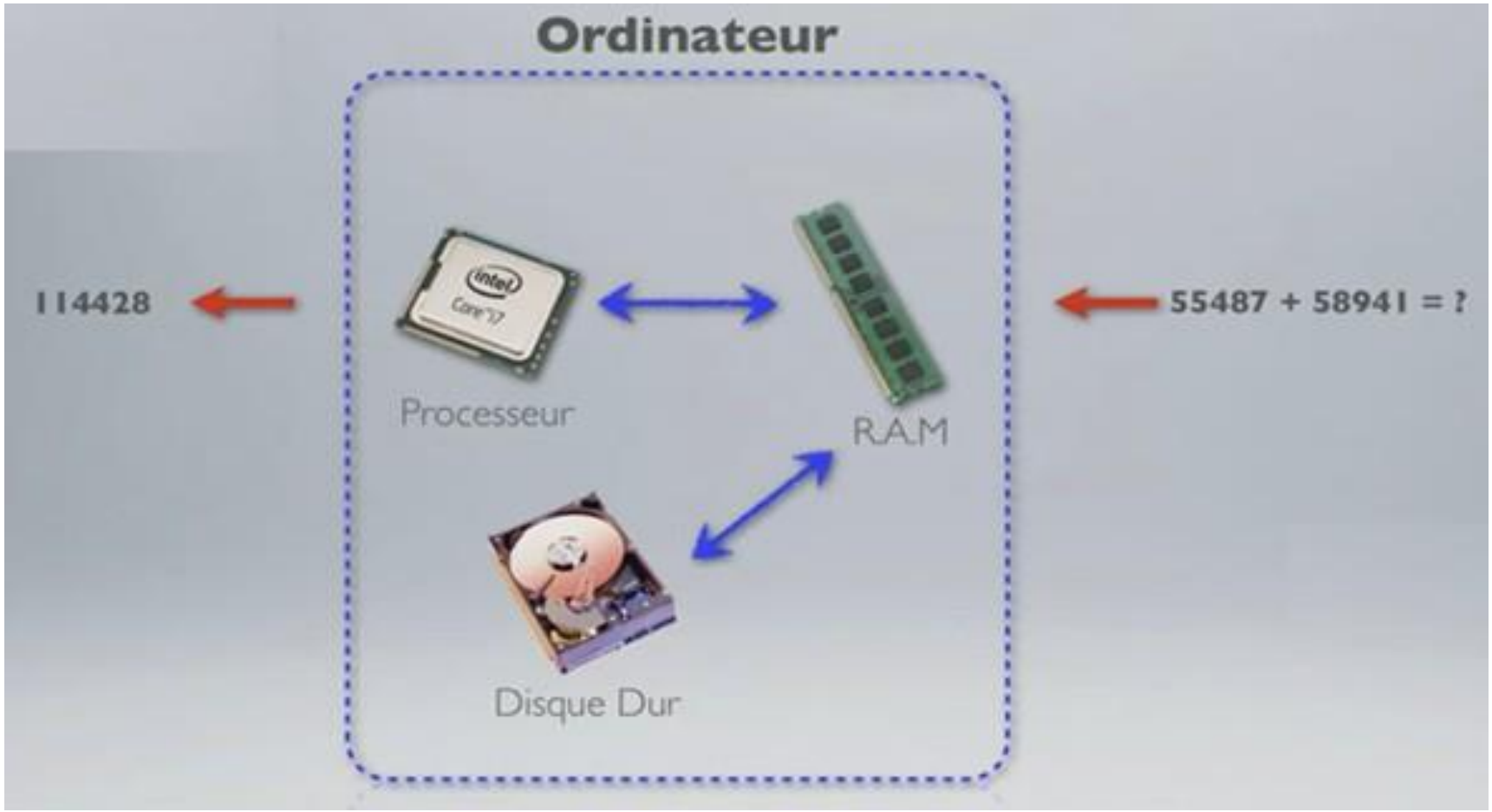


Disque Dur

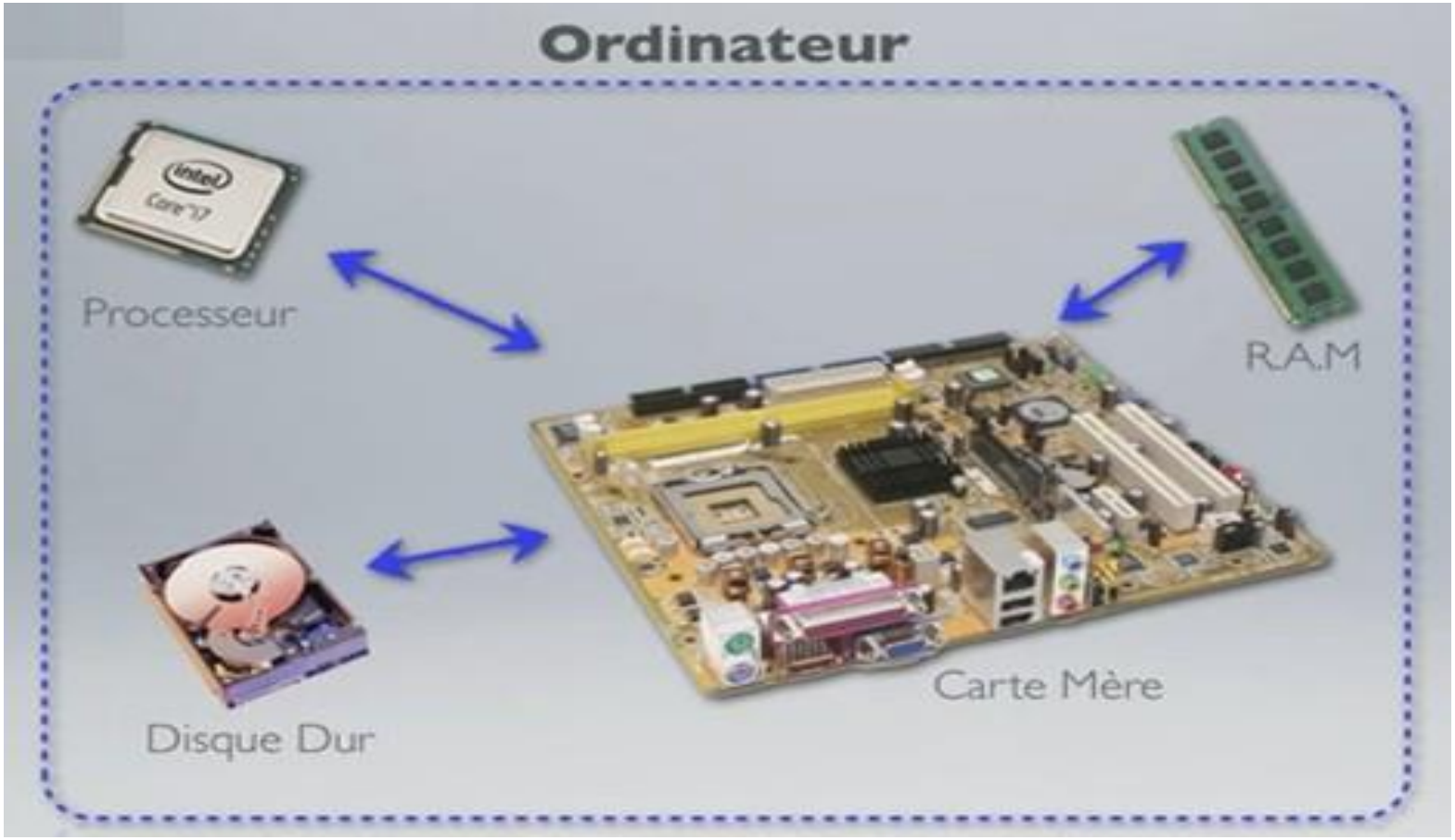
- Mémoire de stockage permanente (magnétique)
- Grande Capacité



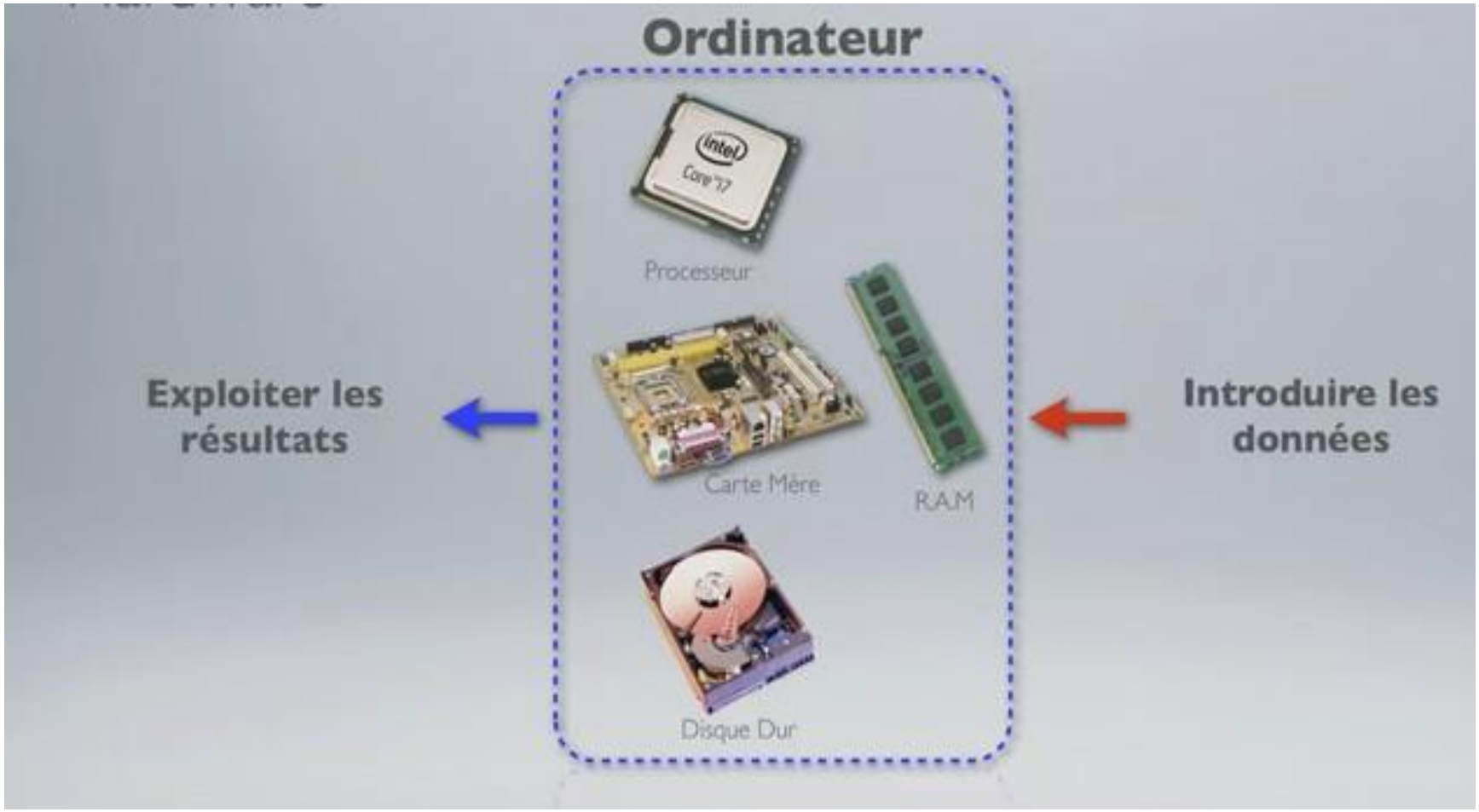
Chargement des données



Support: Carte mère

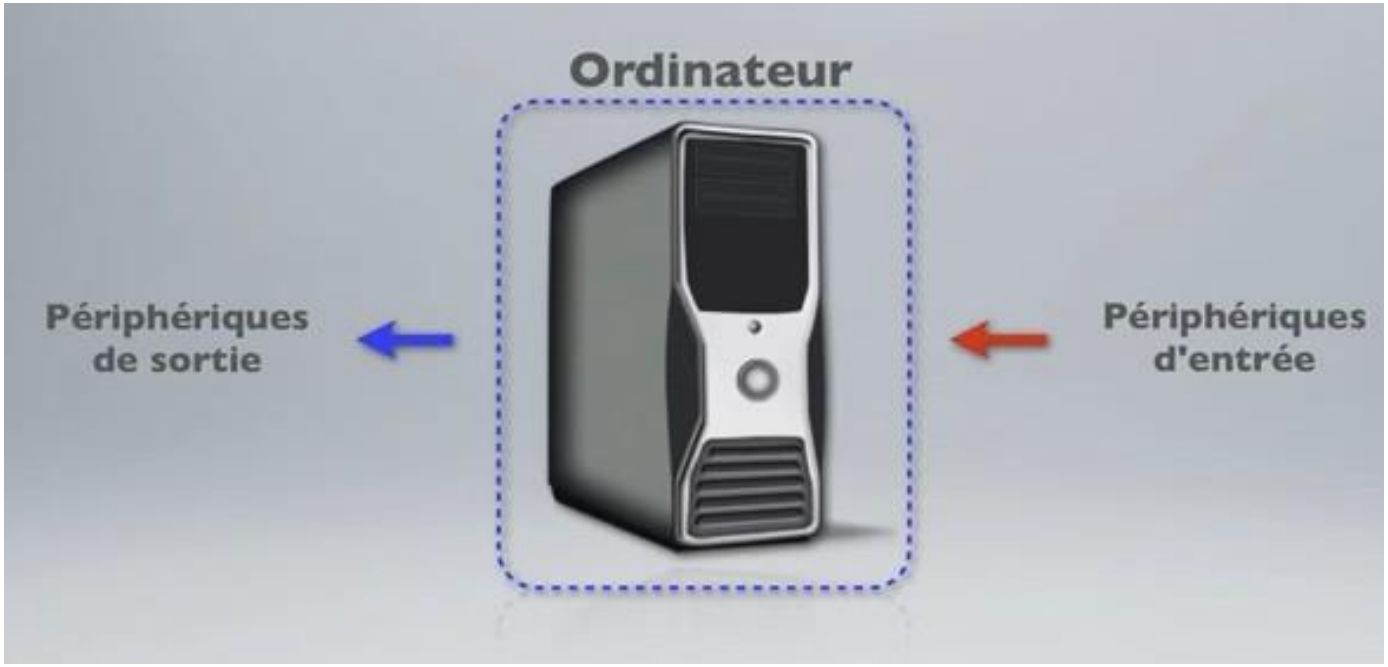


Comment introduire et exploiter les résultats

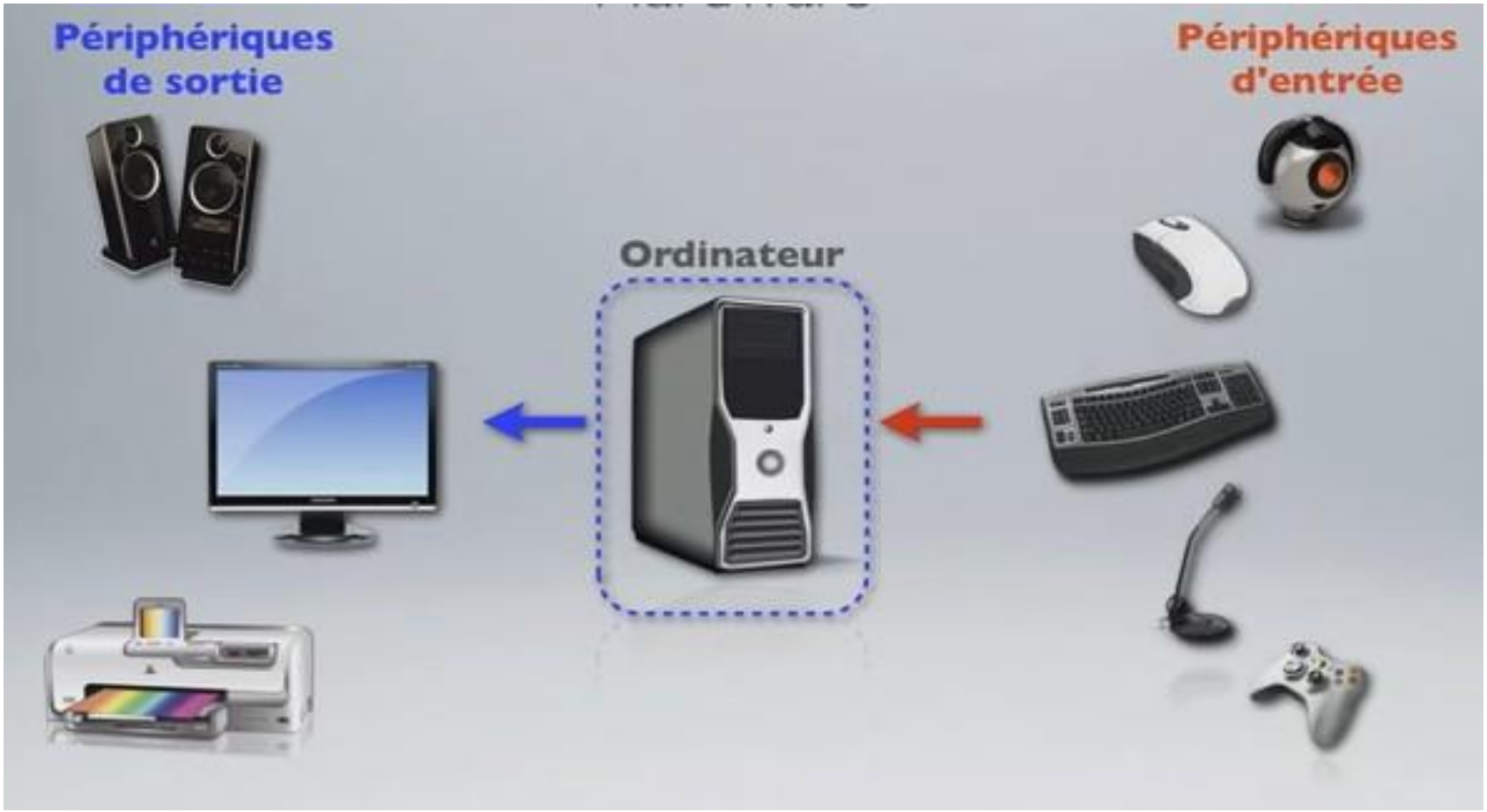


Périphériques

- périphérie est la surface externe

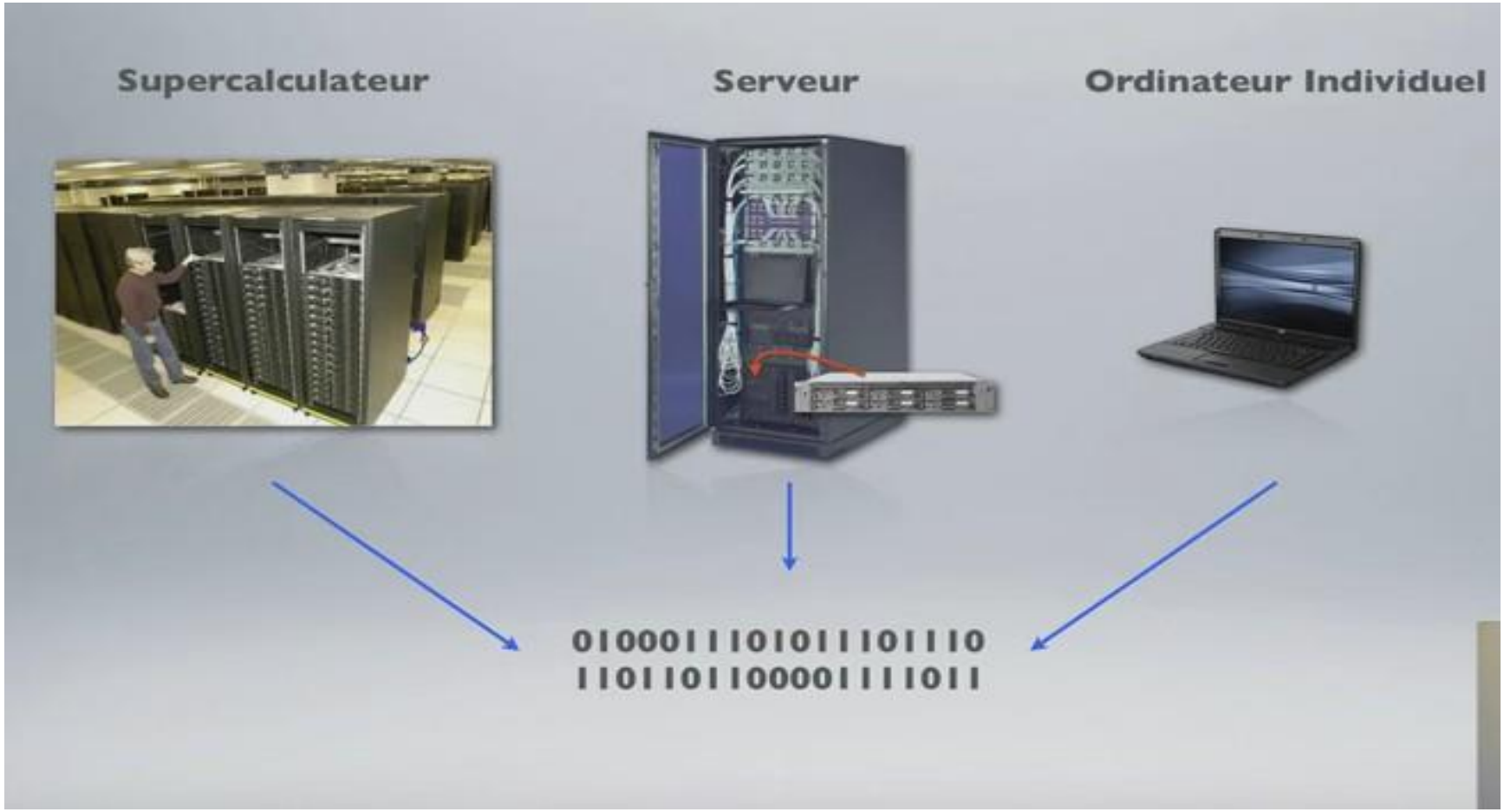


Exemples de Périphériques

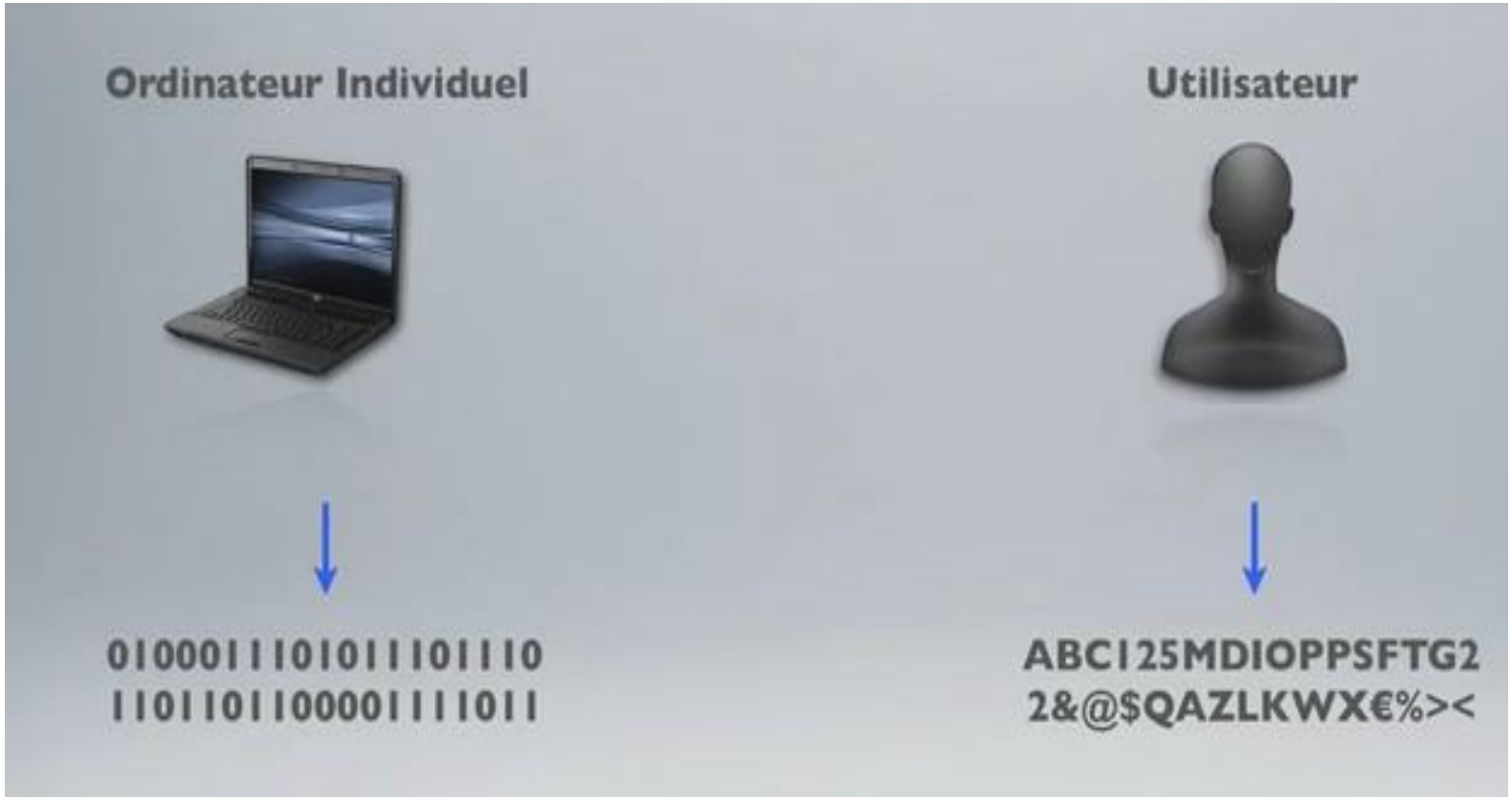


Le langage de l'ordinateur:

Langage binaire

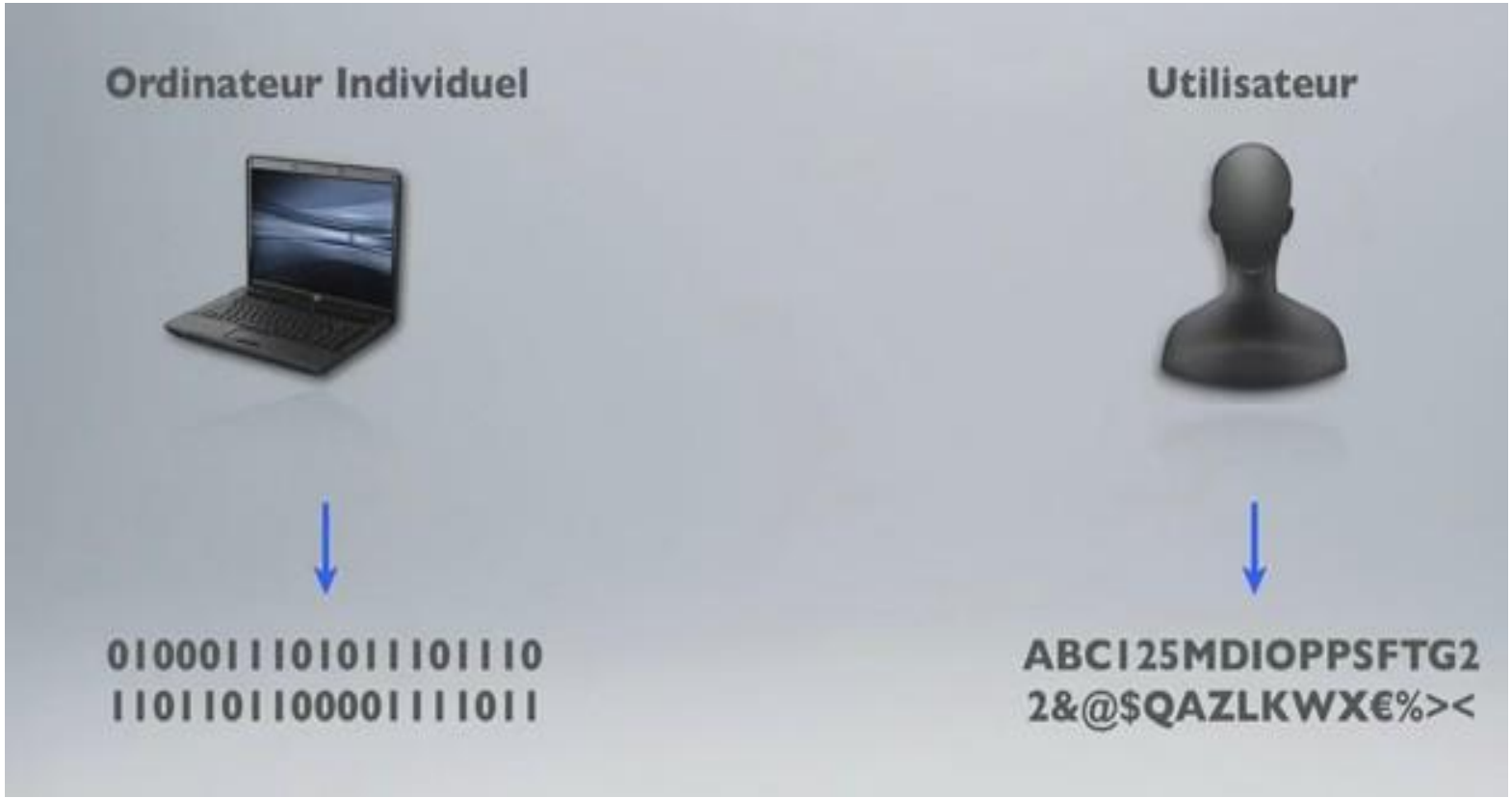


Langage de l'ordinateur et langage de l'utilisateur

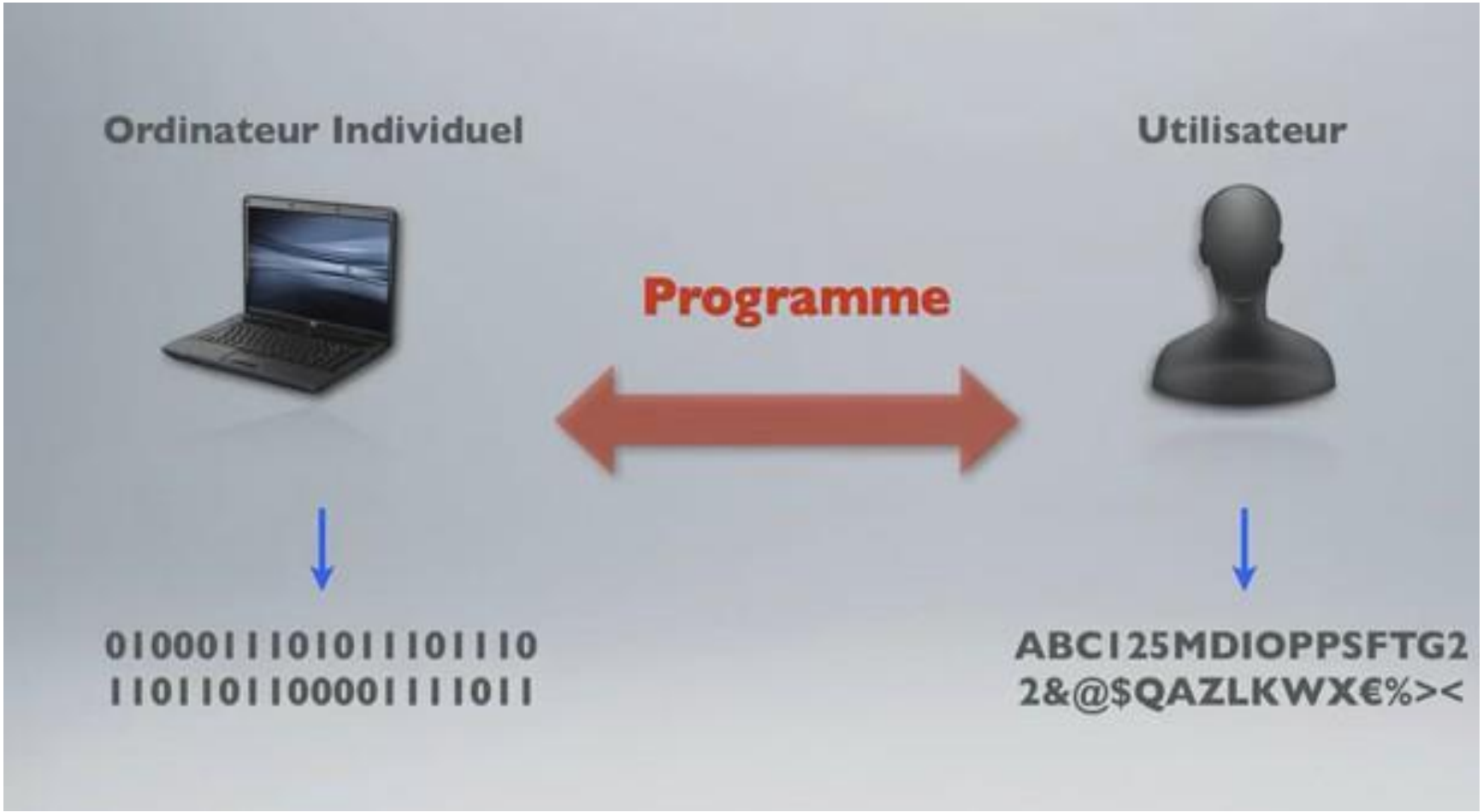


Comment peut-on assurer une bonne exploitation des performances des ordinateurs ?

Langage de l'ordinateur et langage de l'utilisateur



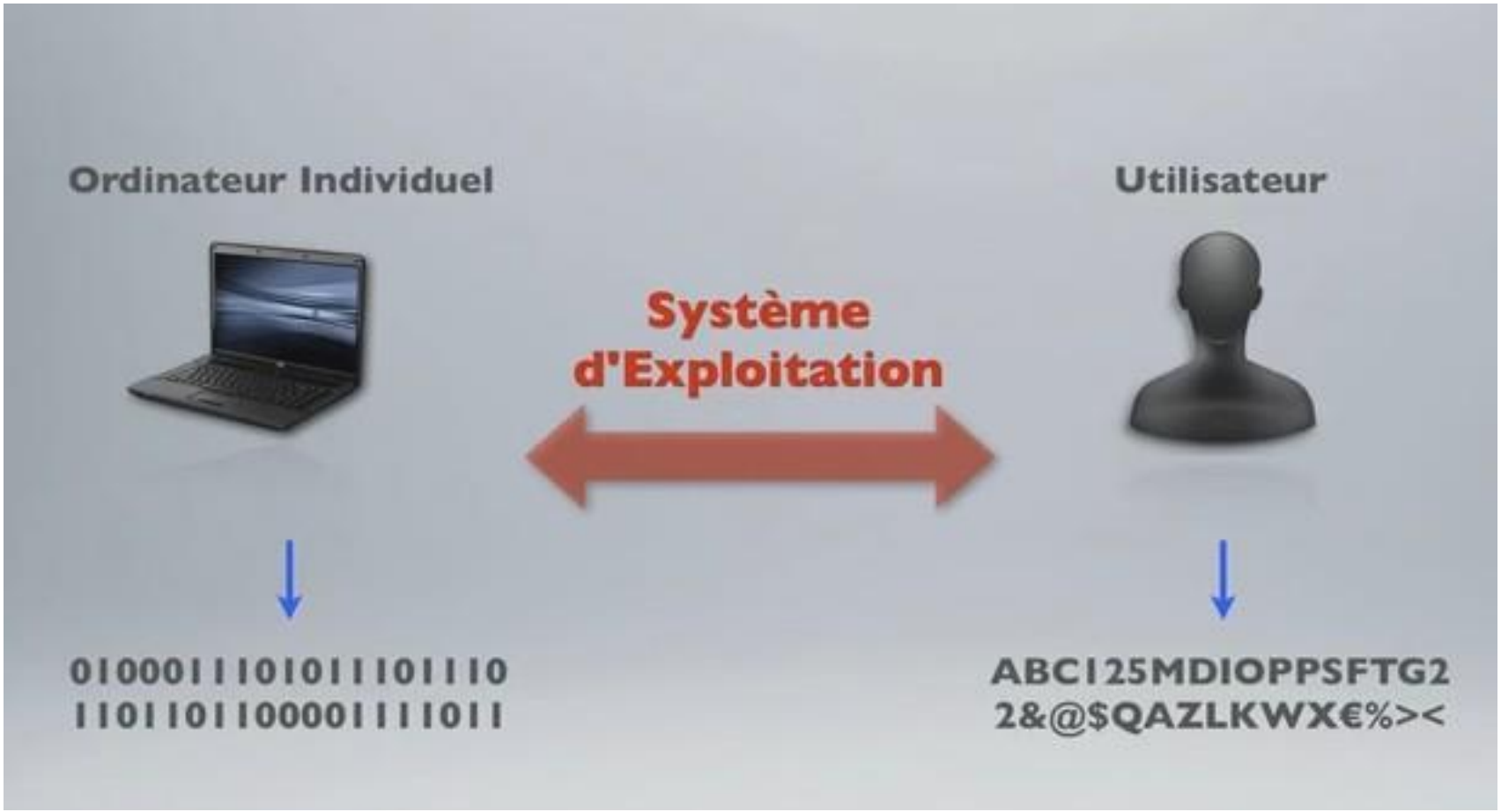
Intermédiaire entre ordinateur et utilisateur



Programmes

- une suite d’opérations prédéterminées destinées à être exécutées de manière automatique par un appareil informatique en vue d’effectuer des travaux, des calculs arithmétiques ou logiques.

Intermédiaire entre ordinateur et utilisateur



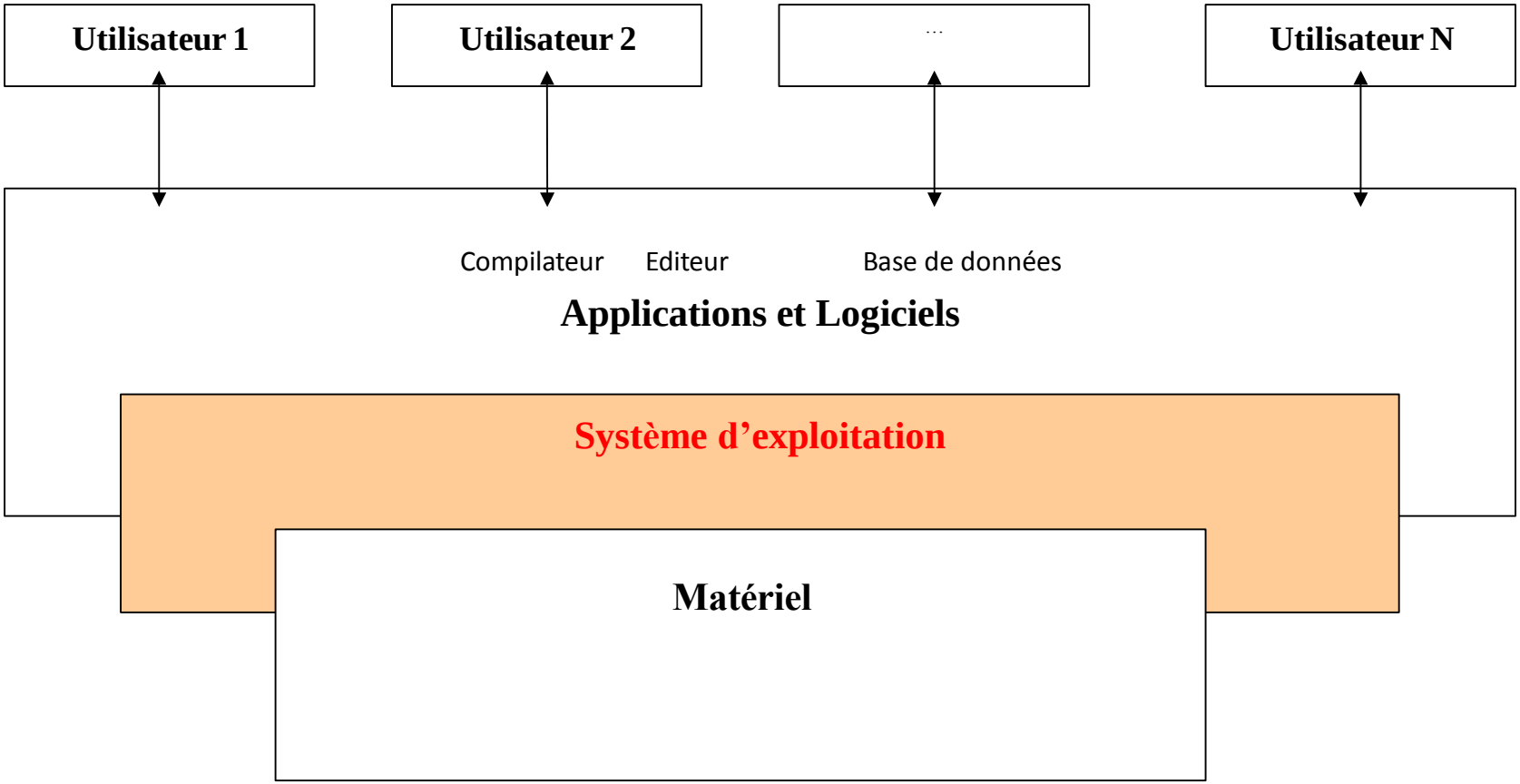
La nécessité d'un SE

Deux **besoins** majeurs :

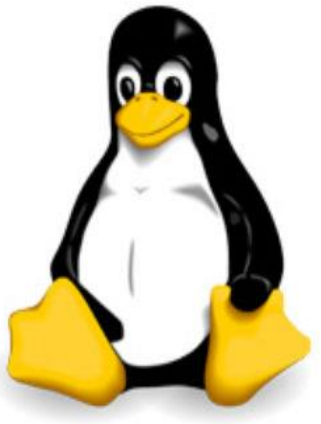
- **Point de vue utilisateur**
  - Comment un utilisateur (ou plusieurs) peut exploiter les ressources matérielles.
  - Trouver des mécanismes pour optimiser l'utilisation du matériels (facilité, rapidité, partage,…)

- **Point de vue système**
  -Comment exploiter d'une manière optimale les ressources matérielles pour améliorer leur fonctionnement
  - Trouver des mécanismes pour gérer efficacement les ressources matérielles (mémoires centrales, disque dur, temps processeur, …)

Système d'exploitation:
un élément du système informatique



Exemples de Systèmes d’exploitation



iOS 

Différents systèmes

Monde PC

MS-DOS (MicroSoft-Disk Operating System)
Windows 95 et 98
Windows NT, 2000, XP, Vista,7,8,10

Monde MAC (Multi-utilisateurs)

MAC OS 9, OS 10, OS 10.2

Monde UNIX

SUN,... **LINUX** (Multi-utilisateurs)

Multi-tâches

Historique de WINDOWS

- 81 : Système d ’exploitation MS-DOS
  - Lié au PC (IBM)
- 83 : Création de Windows
- 85 : Windows 1.0
- 90 : Windows 3.0
- 92 : Windows 3.1
- 95 : Windows 95

Interface graphique pour DOS

- 98 : Windows 98
- 2000 : Windows 2000
- 2006 : vista
- 2009 : 7 ; 2012: 8 ;2015 : 10

Un « vrai » SE

Historique Linux

1969 : création d'Unix - Ken Tompson (Laboratoires Bell)
1970 : adaptation au DEC PDP-11/20 par Thompson&Ritchie et naissance du premier langage portable : le langage C
1974-77 : les sources d'Unix sont distribuées gratuitement aux Universités
1978 : Unix devient la propriété d'ATT et les sources deviennent payantes
1979 : création de BSD Unix pour l'Université de Californie à Berkeley
1987 : diffusion de X Window, interface graphique pour Unix développée par le MIT
1987 : AIX d'IBM et HP-UX d'HP naissent
1991 : émergence de Linux

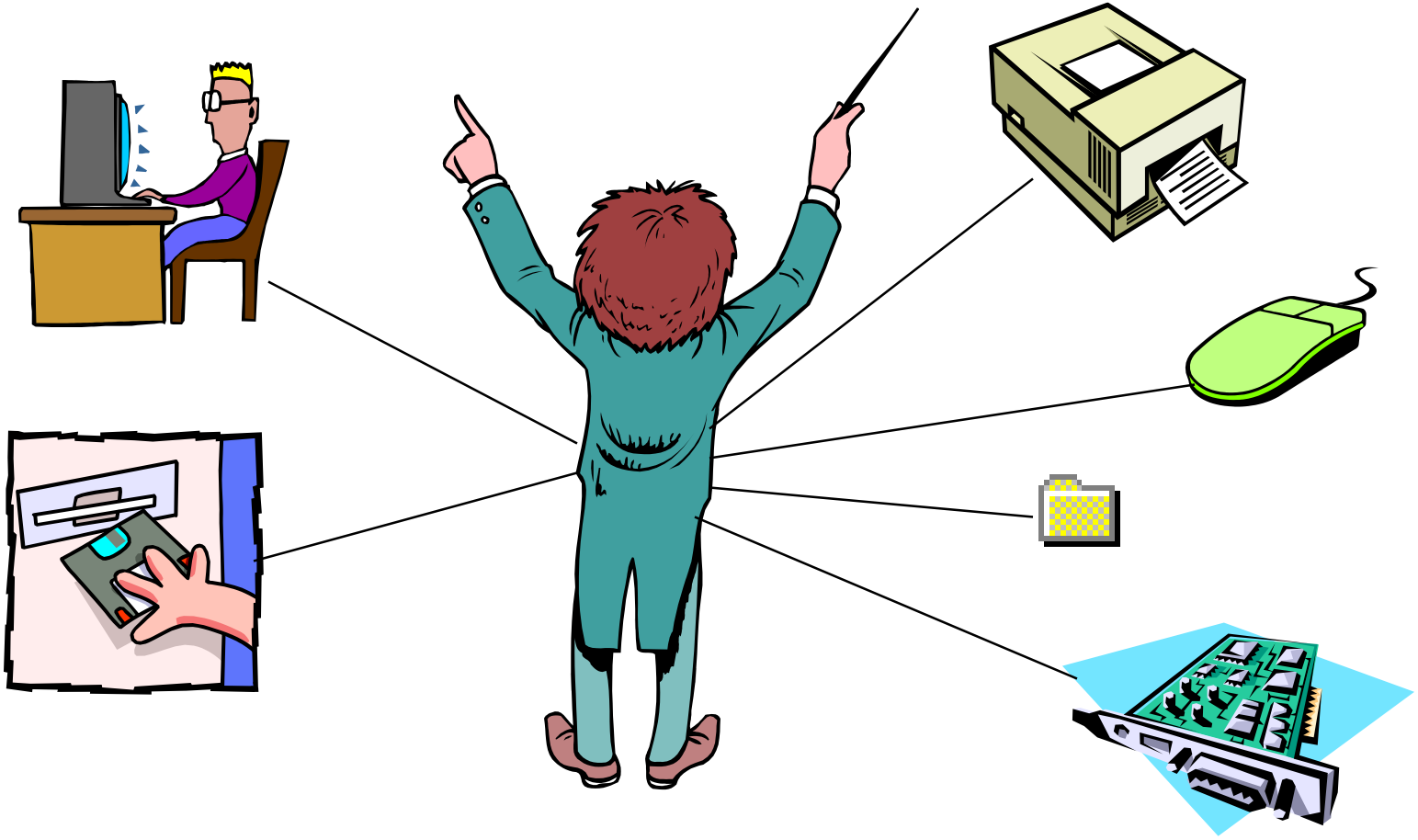
Historique Linux

1992 : développement de Sun OS par Sun
Linux a été écrit par Linus Torvalds, jeune étudiant finlandais, et a été amélioré par de nombreux développeurs dans le monde entier.
1991 : Linux 0.1 et diffusion du code source sur Internet
1993 : Linux 0.99
1994 : FreeBSD 1.0 basé sur BSD Unix
1995 : première distribution « commerciale » RedHat
2002: GNOME 2, KDE 3.0, GTK+ 2, Arch, OpenOffice.org, Mozilla 1.0
2003 :Xfce 4.0, Fedora, Phoenix, Firebird
2004 : Firefox 1.0, X.Org, Ubuntu
2018 : Ubuntu avec X.org par défaut
2019 : Firefox 65.0

Système d’Exploitation:

Définitions, rôles, Tâches…

Définition d'un **SE** ? ? ?

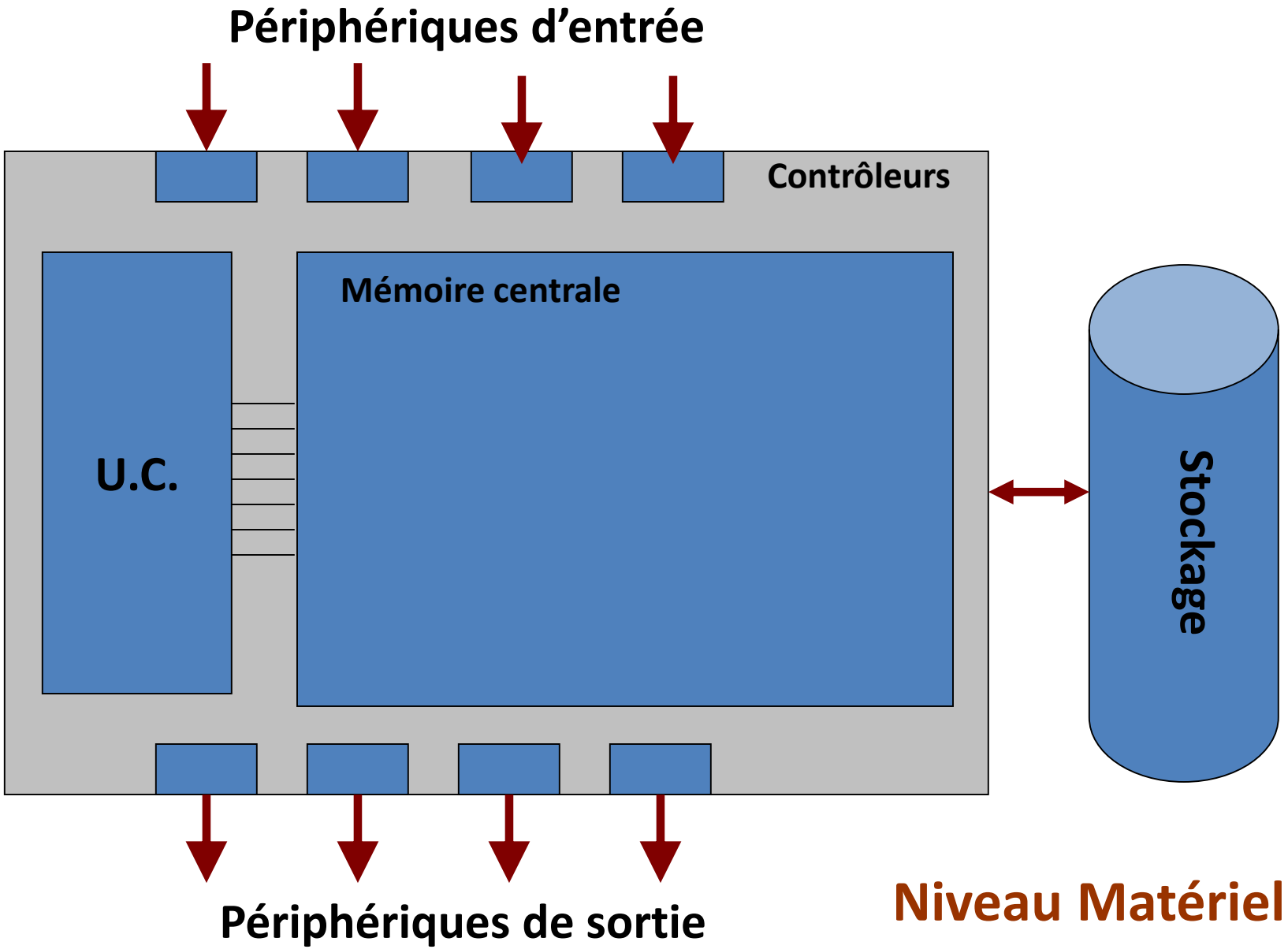


Un chef d'orchestre

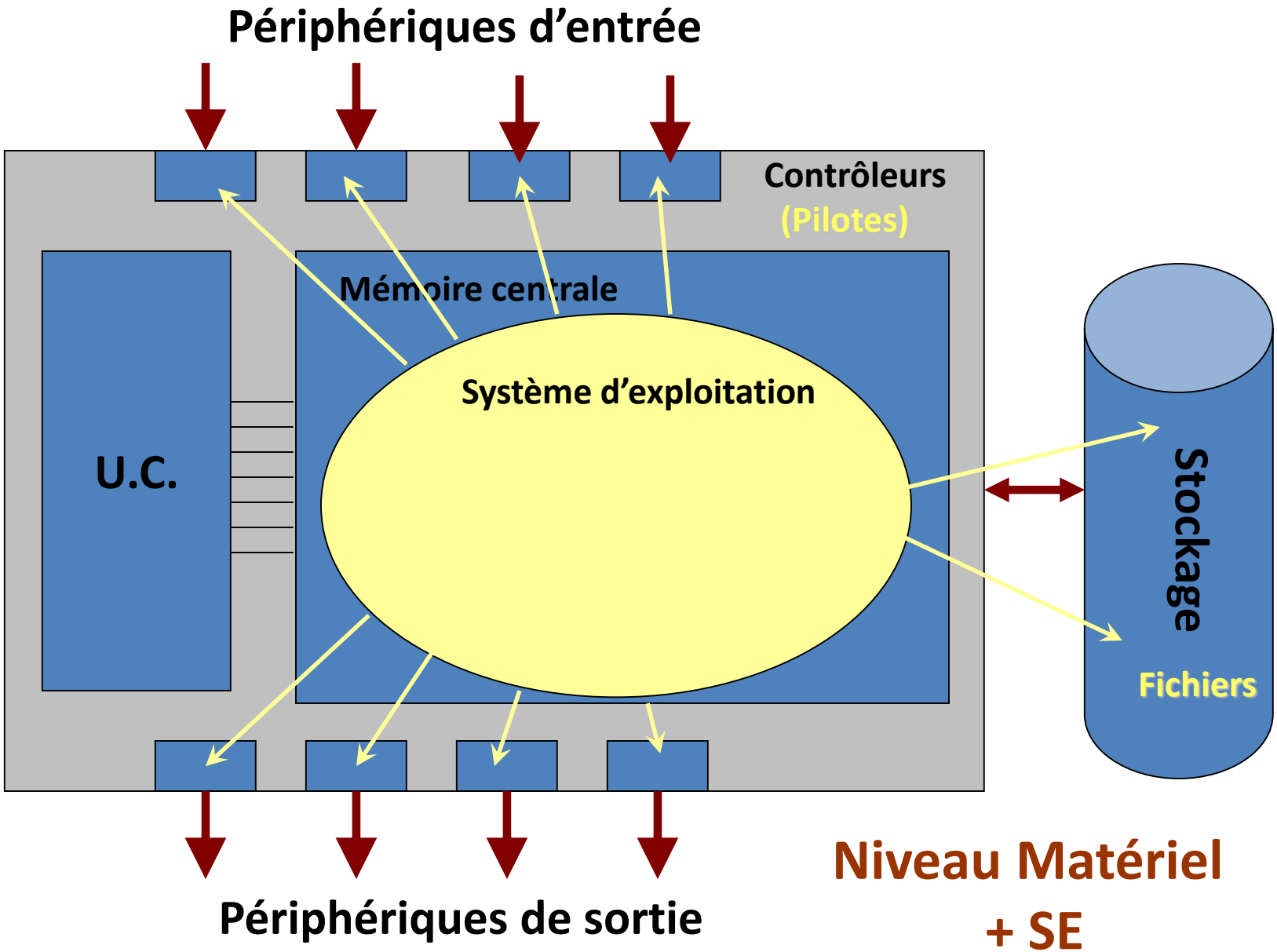
Définition « formelle »

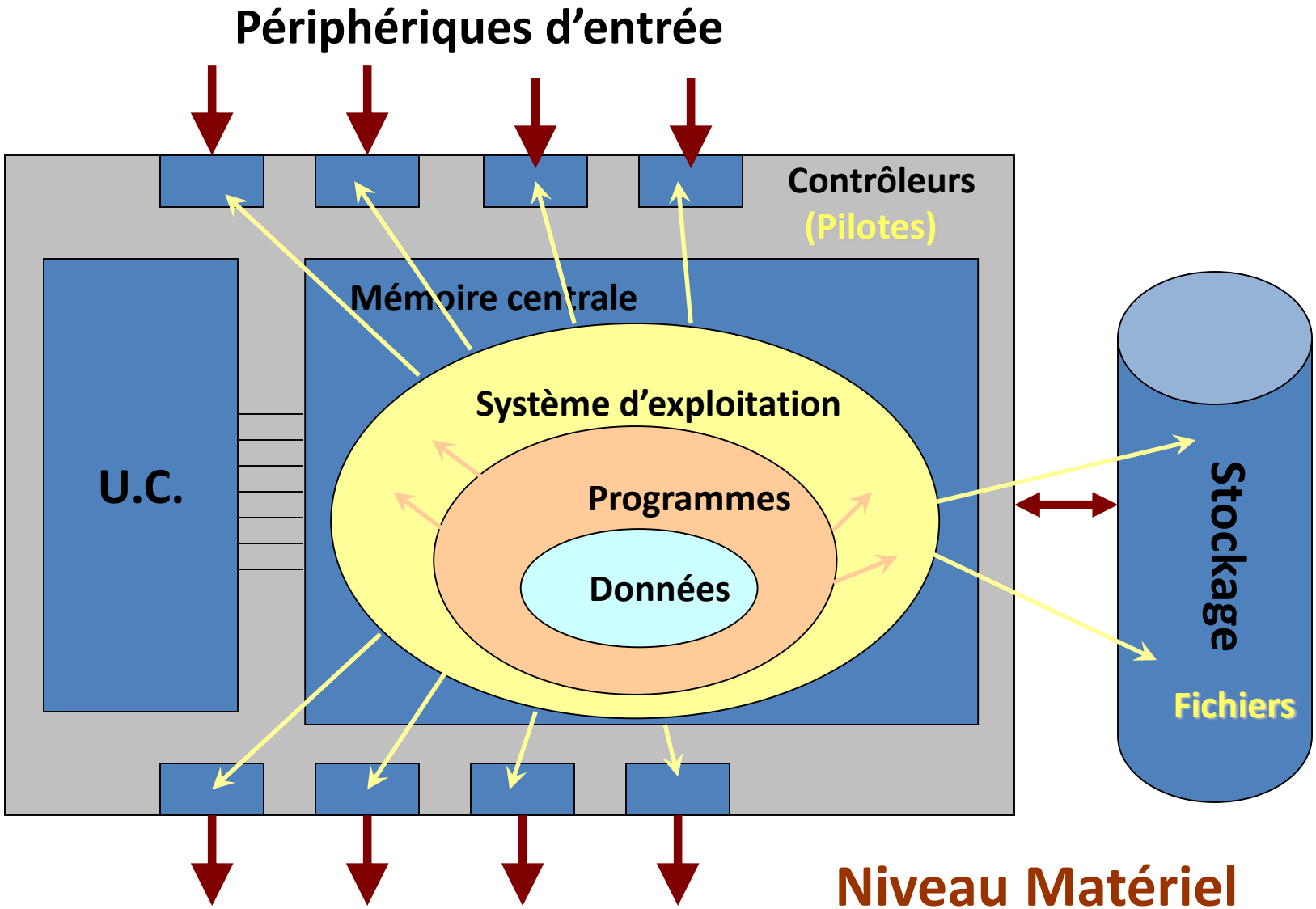
Définition : Un Système d'Exploitation (S.E.) est une machine abstraite conçue pour faciliter l'exploitation du matériel (pilotes de périphériques) ou pour arbitrer l'accès au matériel par les utilisateurs.

Généralement composé d'un **noyau** et d'un **ensemble d'outils** système, le S.E. permet de développer des applications portables, qui ne sont pas spécifiques à un ordinateur ou un système donné.



Niveau Matériel


Périphériques d'entrée
Contrôleurs
(Pilotes)
Mémoire centrale
Système d'exploitation
U.C.
Stockage
Fichiers
Périphériques de sortie
Niveau Matériel
+ SE


Périphériques d'entrée
Contrôleurs
(Pilotes)
Mémoire centrale
U.C.
Système d'exploitation
Programmes
Données
Stockage
Fichiers
Périphériques de sortie
Niveau Matériel
+ SE + Programmes

Les rôles du SE

- Le SE a deux rôles principaux:

1. le rôle du système d'exploitation en tant que **machine étendue** (ou « **machine virtuelle** »),

2. Le rôle du système d'exploitation en tant que **gestionnaire de ressources**.

Le rôle du SE en tant que Machine virtuelle

- Le système d'exploitation correspond à « *l'interface* » entre les applications et le matériel.

- De ce point de vue le système d'exploitation est assimilé à une machine étendue ou virtuelle plus facile à programmer ou à utiliser que le matériel :

- Un **programmeur** doit se concentrer sur les fonctionnalités de son logiciel et non pas sur le fonctionnement du matériel.
- Un **utilisateur** peut aussi manipuler un système d'exploitation, sans pour autant avoir à créer un programme ou connaître le matériel. (masquer la complexité du matériel).

Le rôle du SE en tant que gestionnaire de ressources

- Les différents composants d'un ordinateur doivent coopérer et partager des ressources.
- Dans cette optique, le travail du système d'exploitation consiste à :
  - ordonnancer,
  - contrôler l'allocation des ressources :
    - processeurs,
    - mémoires,
    - périphériques d'E/S,
    - ...

  entre les différents programmes qui y font appel.

Le rôle du SE en tant que gestionnaire de ressources

- Pour chacune des ressources d'un ordinateur, le système d'exploitation doit :
  - connaître à tout moment l'utilisateur de la ressource,
  - en accorder l'usage de manière équitable,
  - éviter les conflits d'accès entre les différents programmes ou utilisateurs.

- Le Système d'Exploitation doit assurer
  - Le **partage** des ressources.
  - La **protection** de l'accès aux ressources.

Exemple du rôle du SE comme étant gestionnaires de ressources

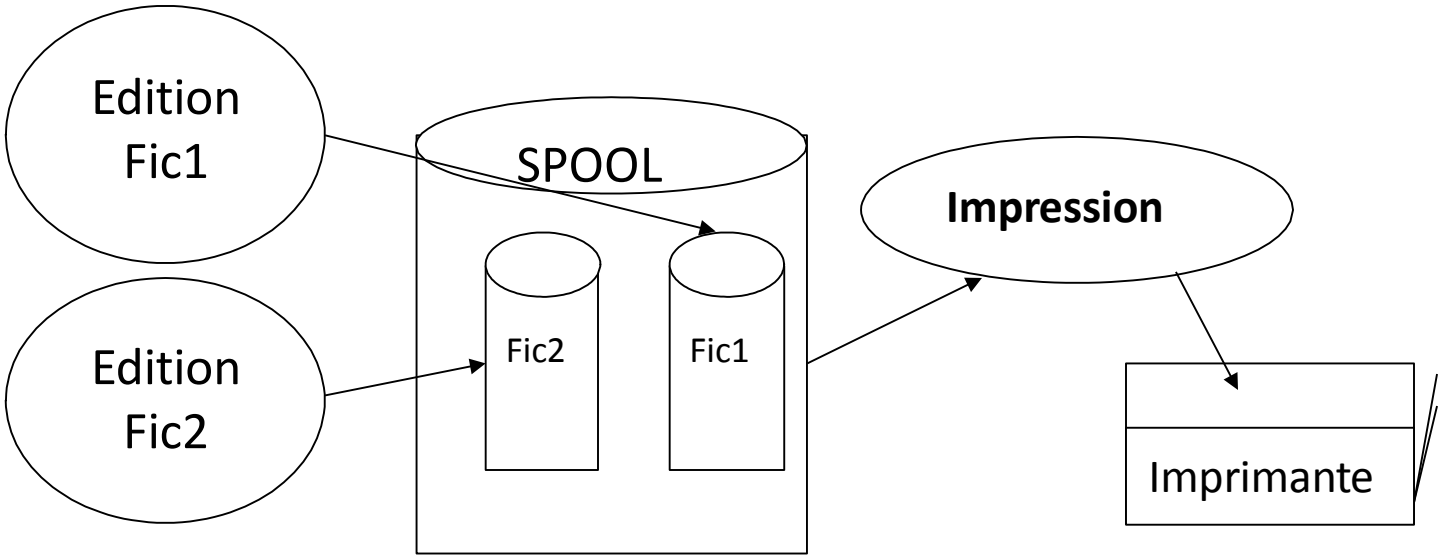
- Trois programmes d'impression sur la même imprimante.

  - Désordre total: Les premières lignes imprimées pourraient provenir du programme 1, les suivantes du programme 2, puis du programme 3 et ainsi de suite.

- Comment éviter ce problème ?

  - Le système d'exploitation peut éviter ce désordre grave en transférant les résultats à imprimer dans un fichier tampon sur le disque. Lorsqu'une impression se termine, le système d'exploitation peut alors imprimer un des fichiers se trouvant dans le tampon.

Exemple du rôle du SE comme étant gestionnaires de ressources



Les taches(de base) d'un OS

1. La gestion des processus
   - qui correspondent à l'exécution des programmes.
2. La gestion de la mémoire
   - qui permet de gérer les transferts entre les mémoires principales et secondaires.
3. Le système de fichiers
   - qui offre à l'utilisateur une vision homogène et structurée des données et des ressources : disques, périphériques.
4. Les entrées-sorties
   - qui correspondent aux mécanismes qu'utilisent les processus pour communiquer avec l'extérieur.

Autres Tâches du OS

- Les réseaux d'ordinateurs

  avec les protocoles de communication, d'interconnexion et d'application.

- Les systèmes répartis
  - avec les protocoles d'appels de procédures à distance (RPC)
  - ou les objets distribués.

- Les systèmes de fenêtrage graphiques, … etc.

Caractéristiques

des Systèmes d'Exploitation

Mono-tâche	Multi-tâches
A tout instant, un seul programme est exécuté; un autre programme ne démarrera, sauf conditions exceptionnelles que lorsque le premier sera terminé.	Plusieurs processus (un programme en cours d'exécution peuvent s'exécuter simultanément (systèmes multiprocesseurs ou systèmes à temps partagé).
mono session	**multisessions**
Au plus un utilisateur à la fois sur une machine. Les systèmes réseaux permettent de différencier plusieurs utilisateurs, mais chacun d'eux utilise de manière exclusive la machine (Multi-Utilisateurs, mono session)	Plusieurs utilisateurs peuvent travailler simultanément sur la même machine.

comparaison des systèmes d'exploitation

Système	Codage	Mono-/multi-utilisateur	Mono-/multi-tâche
DOS	16	mono	mono
Windows 3.1	16/32	mono	non préemptif
Windows 95	32	mono	coopératif
Win. NT/2000	32	multi	préemptif
Windows XP, vista,7,8,10	32/64	multi	préemptif
Unix/Linux	32/64	multi	préemptif
Mac OS X	32/64	multi	préemptif

SYSTÈMES D'EXPLOITATION I

Chapitre 1 (suite)

Dr. Mohamed Barkaoui
Département Informatique FSB

Résumé cours précédent

- Abstraction haut-niveau de la machine
- 4 grandes tâches
  - processus
  - mémoire
  - fichiers
  - entrées/sorties
- 2 niveaux privilèges CPU : **noyau (0)** et **utilisateur (3)**
- Multiprogrammation :
  - plus d'un processus chargé en même temps
  - partage des ressources
  - maximiser l'utilisation de la machine : **performance!**

Suite du Cours

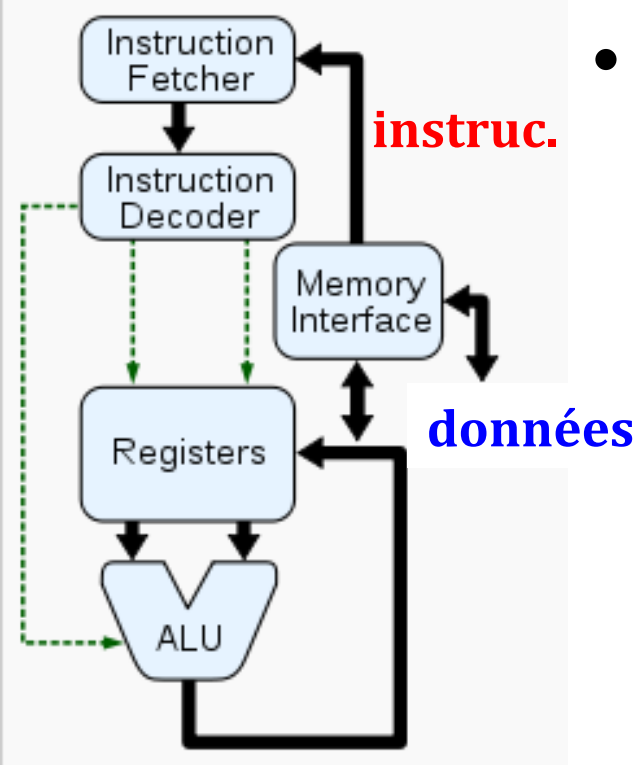
- Structure matérielle d'un ordinateur
  - CPU,
  - Mémoire,
  - Disques,
  - Périphériques,
  - Bus
- Différents types de systèmes d'exploitation
- Bref survol de la structure

Structure matérielle d’un ordinateur

Composantes de base d'un ordinateur

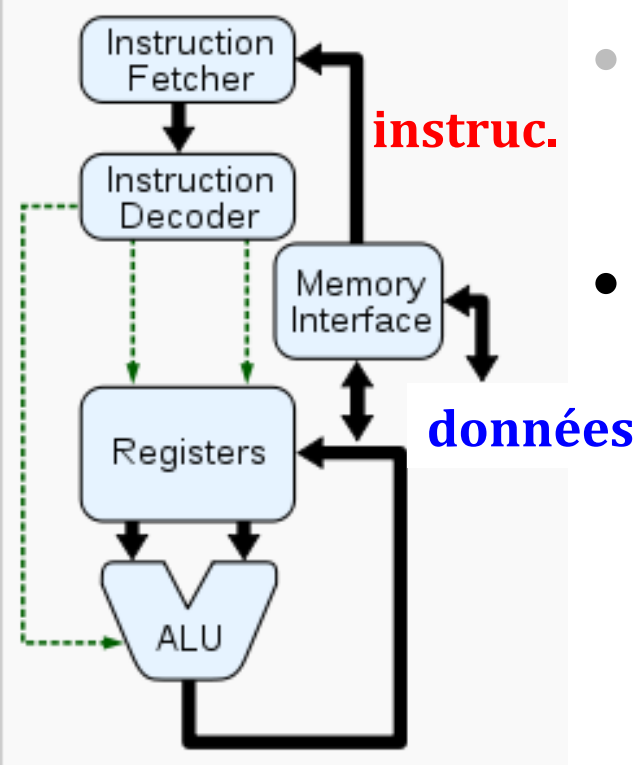
- Processeur(s) ou CPU
- Mémoire
- Disques
- Périphériques d'entrées et sorties (E/S)
- Les bus (ISA, PCI, etc…)

Processeur (CPU)



- Base même de tous les ordinateurs
- Extrait les instructions de la mémoire et les exécute.

Processeur (CPU)



- Base même de tous les ordinateurs
- Extrait les instructions de la mémoire et les exécute.
- Chaque modèle possède
  - jeu d'instructions
  - registres
  - architecture
  - mémoire (cache)
  - autres (gestion mémoire, etc…)

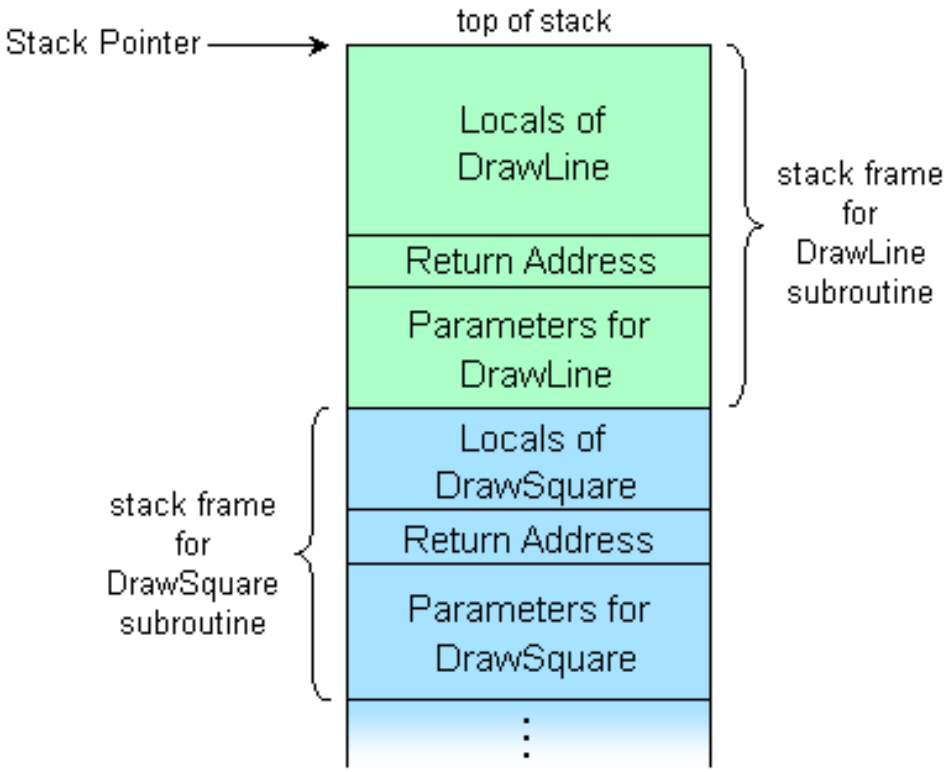
Principaux registres CPU

- Compteur ordinal / *Program Counter* (PC)
  - adresse de l'instruction en cours
- Pointeur de pile / *Stack pointer*
  - adresse courante du sommet de la pile
- Mot d'état / *Program status word*
  - état du processeur +

    bit pour comparaison (zéro, overflow, etc)

Révision : pile (*stack*)

- Sert pour :
  - passer les paramètres lors d'un appel de fonction
  - adresse de retour
  - variables locales de cette fonction

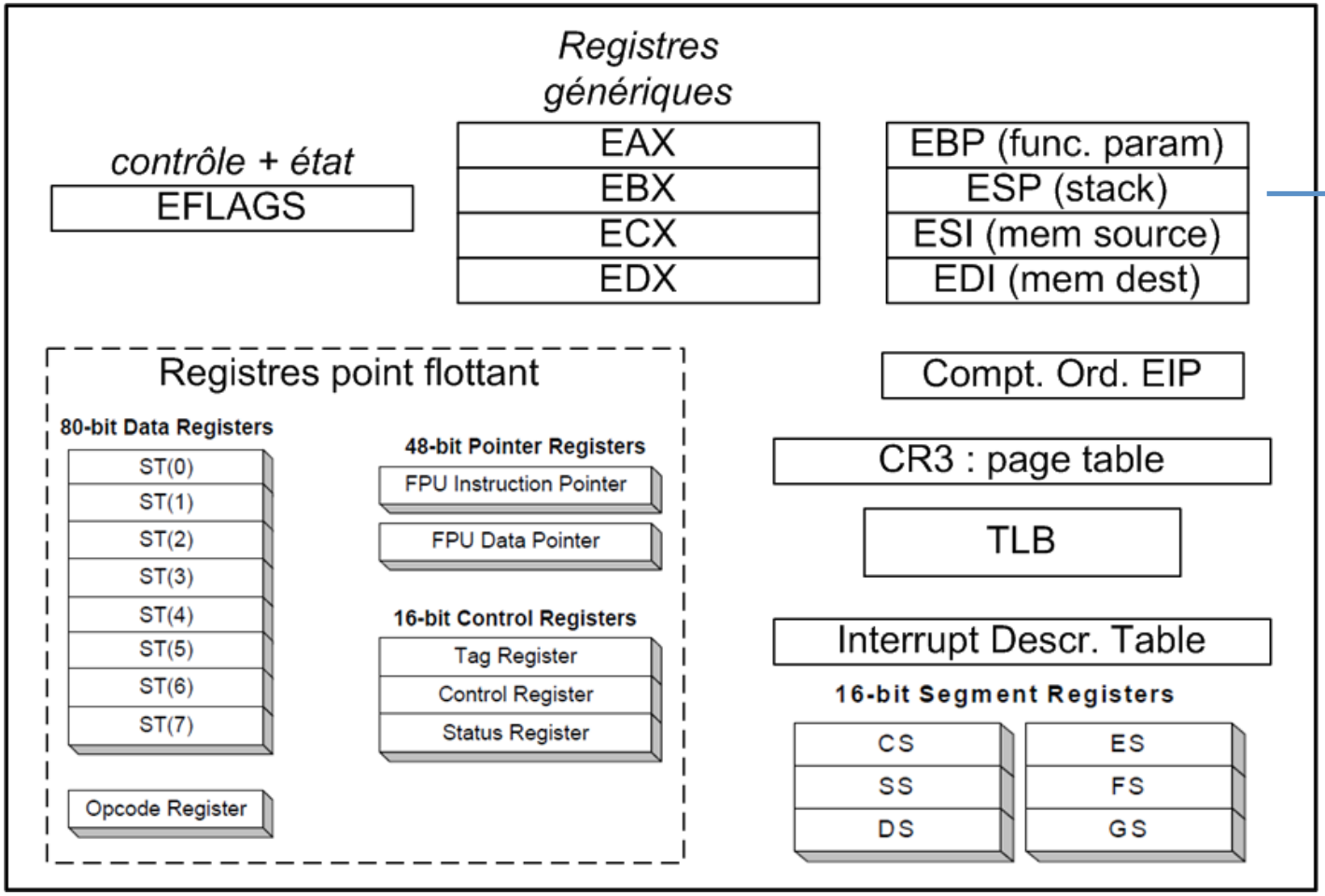


tiré de wikipédia
auteur : R. S. Shaw

DrawLine( ) est appelé de DrawSquare( )

Principaux registres CPU

- Architecture Intel x86



Principaux registres CPU

- Exemple avec addition

mémoire

A=?

Registre **EIP**

Registre **EAX**

```
int A;
A = 3;

A = A + 4;
```

Principaux registres CPU

- Exemple avec addition

mémoire

A=?

Registre **EIP**

Registre **EAX**

```
            int A;
            A = 3;
001F138E    mov   dword ptr [A],3
            A = A + 4;
001F1395    mov   eax,dword ptr [A]
001F1398    add   eax,4
001F139B    mov   dword ptr [A],eax
```

(Sortie du compilateur Visual C++ 2008 Express Edition)

Principaux registres CPU

- Exemple avec addition

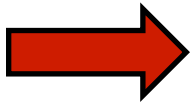
mémoire

A=?

Registre **EIP**

0x001F138E

Registre **EAX**



```
            int A;
            A = 3;
001F138E    mov   dword ptr [A],3
            A = A + 4;
001F1395    mov   eax,dword ptr [A]
001F1398    add   eax,4
001F139B    mov   dword ptr [A],eax
```

Principaux registres CPU

- Exemple avec addition

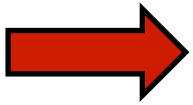
mémoire

A=3

Registre **EIP**

0x001F1395

Registre **EAX**



```
            int A;
            A = 3;
001F138E    mov   dword ptr [A],3
            A = A + 4;
001F1395    mov   eax,dword ptr [A]
001F1398    add   eax,4
001F139B    mov   dword ptr [A],eax
```

Principaux registres CPU

- Exemple avec addition

mémoire

A=3

Registre **EIP**

0x001F1398

Registre **EAX**

3

```
            int A;
            A = 3;
001F138E    mov   dword ptr [A],3
            A = A + 4;
001F1395    mov   eax,dword ptr [A]
001F1398    add   eax,4
001F139B    mov   dword ptr [A],eax
```

Principaux registres CPU

- Exemple avec addition

mémoire

A=3

Registre **EIP**

0x001F139B

Registre **EAX**

7

```
            int A;
            A = 3;
001F138E    mov   dword ptr [A],3
            A = A + 4;
001F1395    mov   eax,dword ptr [A]
001F1398    add   eax,4
001F139B    mov   dword ptr [A],eax
```

Principaux registres CPU

- Exemple avec addition

mémoire

A=7

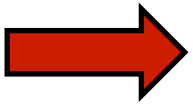
Registre **EIP**

0x001F139E

Registre **EAX**

7

```
            int A;
            A = 3;
001F138E    mov   dword ptr [A],3
            A = A + 4;
001F1395    mov   eax,dword ptr [A]
001F1398    add   eax,4
001F139B    mov   dword ptr [A],eax
001F139E    ...
```



Principaux registres CPU

- Exemple avec addition

mémoire

A=7

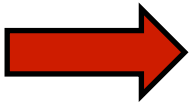
Registre **EIP**

0x001F139E

Registre **EAX**

7

```
            int A;
            A = 3;
001F138E    mov   dword ptr [A],3
            A = A + 4;
001F1395    mov   eax,dword ptr [A]
001F1398    add   eax,4
001F139B    mov   dword ptr [A],eax
001F139E    ...
```

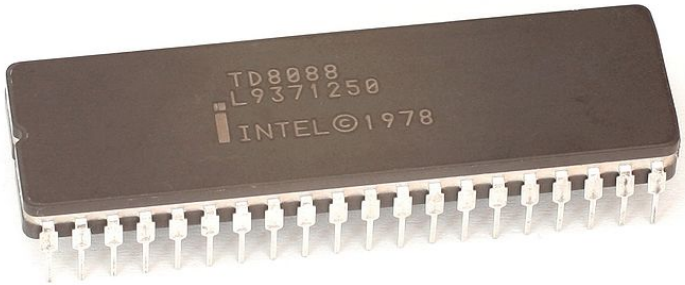


est en fait 3 instructions assembleur

Évolution : monotâche

- 8088 : Accès à tous les registres, adresses mémoires

Intel 8088
16 bits interne, 5-10 MHz

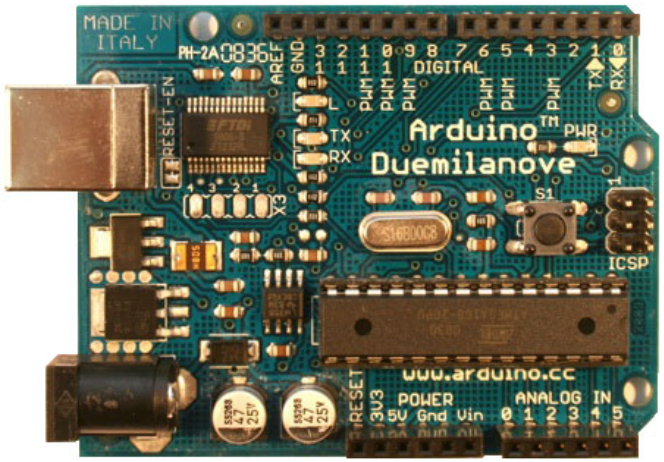


- Si désire faire tourner + d'un processus, il faut :
  - protéger les autres processus des incursions (accidentelles ou volontaires) ?
  - comment : nécessite CPU plus évolué

Évolution : Multitâches

- Mode protégé (*protected*)
- Chez *Intel* en 1982 : 80286

- Pas sur tous les CPU récents :
  - Microcontrôleur (ATmega168)



Arduino

Intel 80286

An 8MHz Intel 80286 Microprocessor

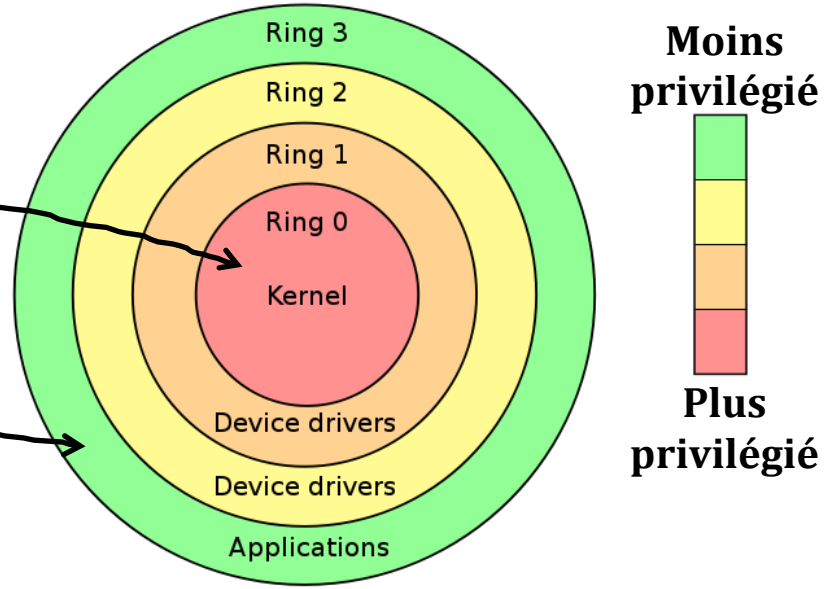
Produced	From 1982 to early 1990s
Common manufacturer(s)	Intel, IBM, AMD, Harris (Intersil), Siemens AG, Fujitsu
Max. CPU clock rate	6 MHz (4 MHz for a short time) to 25 MHz
Min. feature size	1.5µm
Instruction set	x86-16 (with MMU)
Package(s)	PGA, CLCC and PLCC 68 -pin

Mode protégé (*protected*)

- Pour avoir **superviseur (S.E.)**, mécanisme **matériel**
  - avoir des privilèges d'accès particuliers à chaque anneau
  - limiter comment on passe d'un anneau à l'autre

• Mode **noyau** (0) peut exécuter code 0,1,2,3

•Mode **utilisateur** (3) peut exécuter code 3 seulement



anneaux de protection (Intel)

CPU multitâches : 2 modes

Niveau privilège (CPU)	Système d'exploitation	Accès
Ring 0	**noyau** (*kernel*)	accès complet
Ring 3	**utilisateur** (*user*)	accès **limités** pour : instructions registres mémoire

- Avantages
  - contrôle serré des ressources, protège la mémoire
- Défaut : basculer (**`trap`**) d'un mode à l'autre est cher :
  - approx. 1500 cycles machines → **performance**

Niveau privilège processeur vs. admin

deux mécanismes différents!!!!

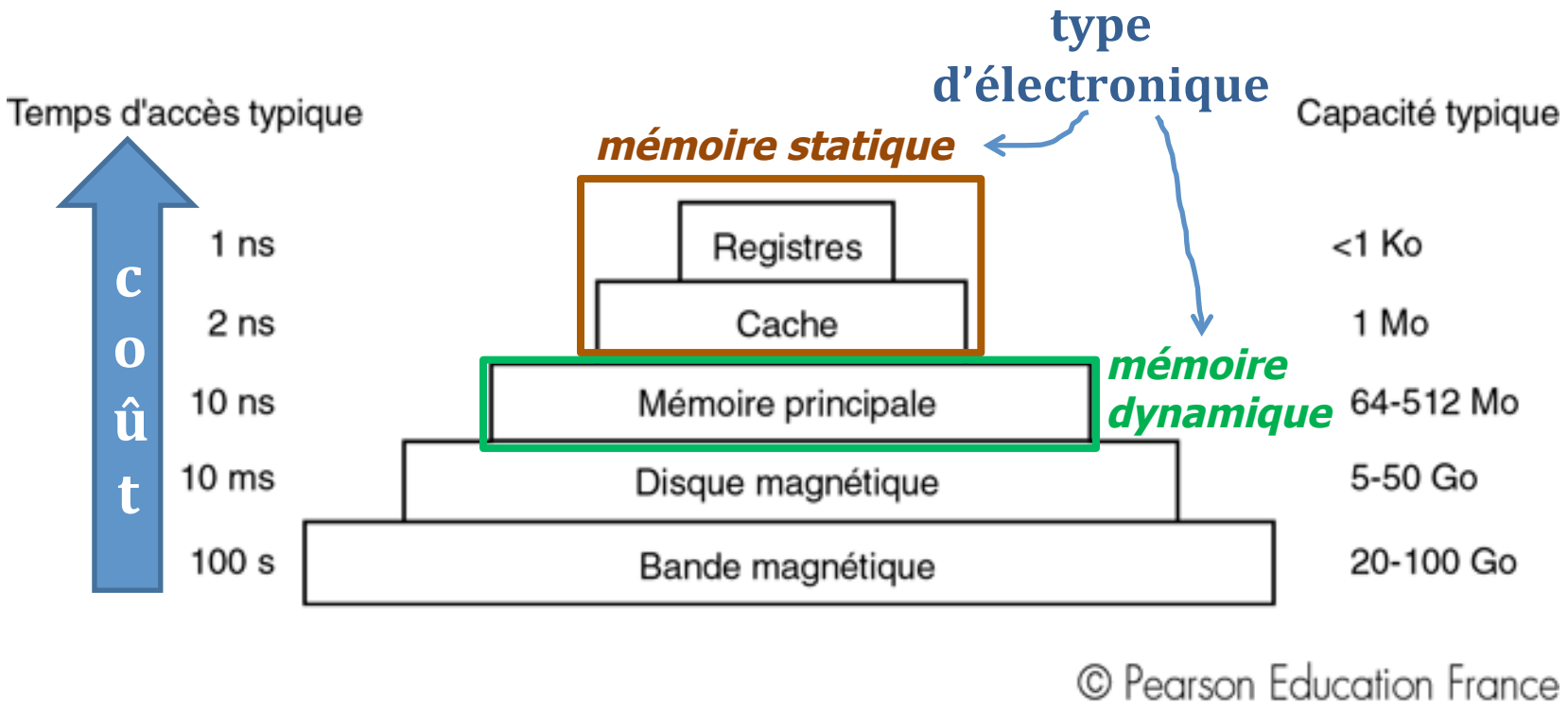
niveau privilège noyau (0) ≠ compte administrateur

matériel **logiciel**

Un compte *admin* est un concept inconnu pour un processeur

Mémoire

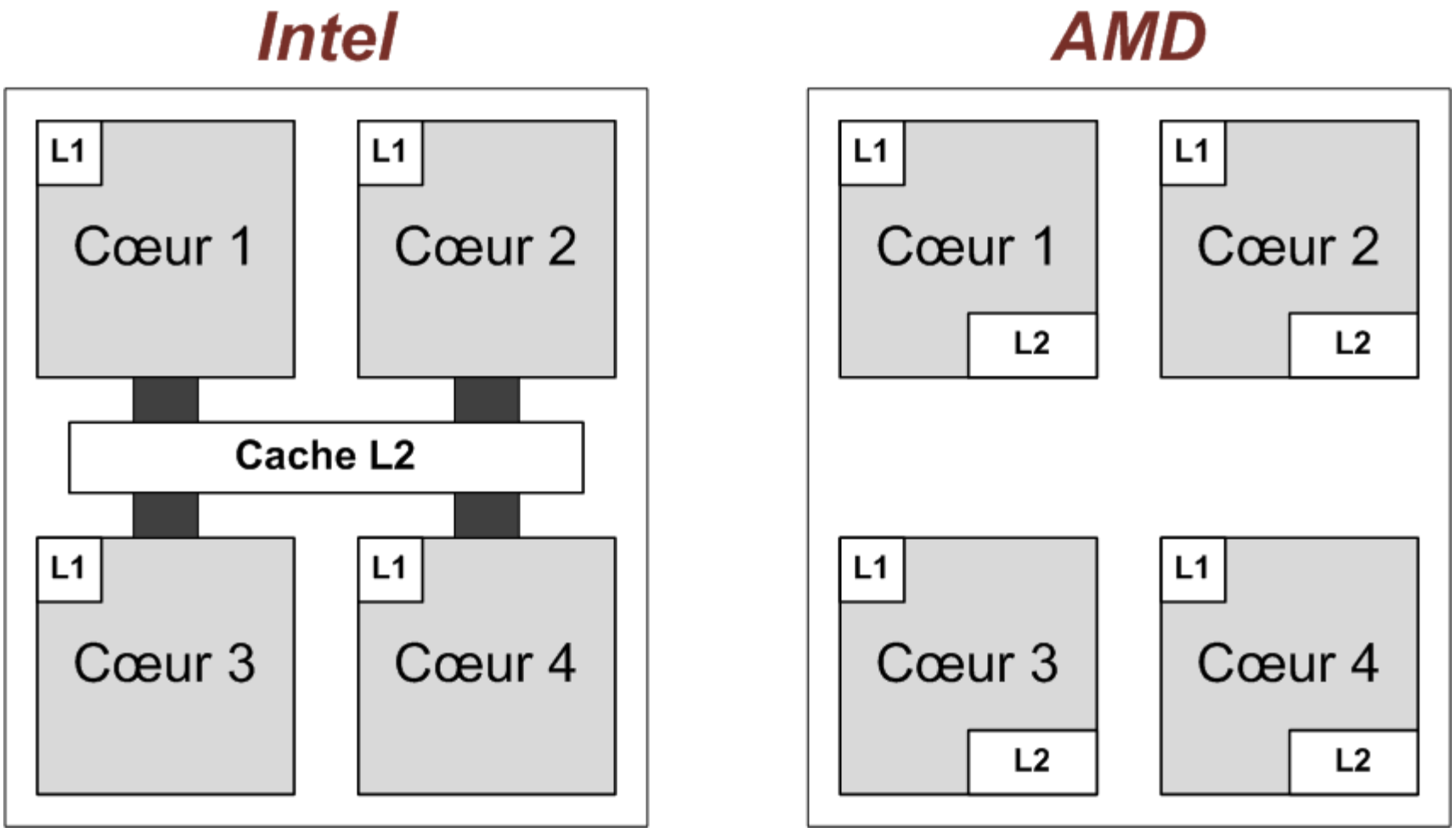
- Architecture mémoire : augmenter performance



statique : 6 transistors
dynamique : 1 transistor

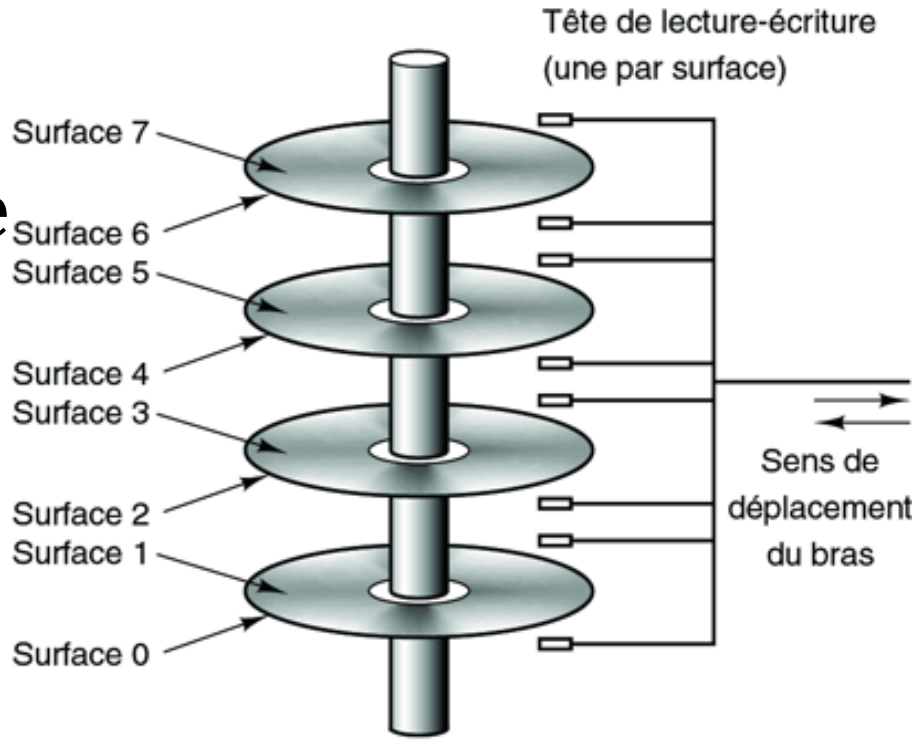
+ transistors == + plus d'espace sur chip

Mémoire cache : multi-coeurs



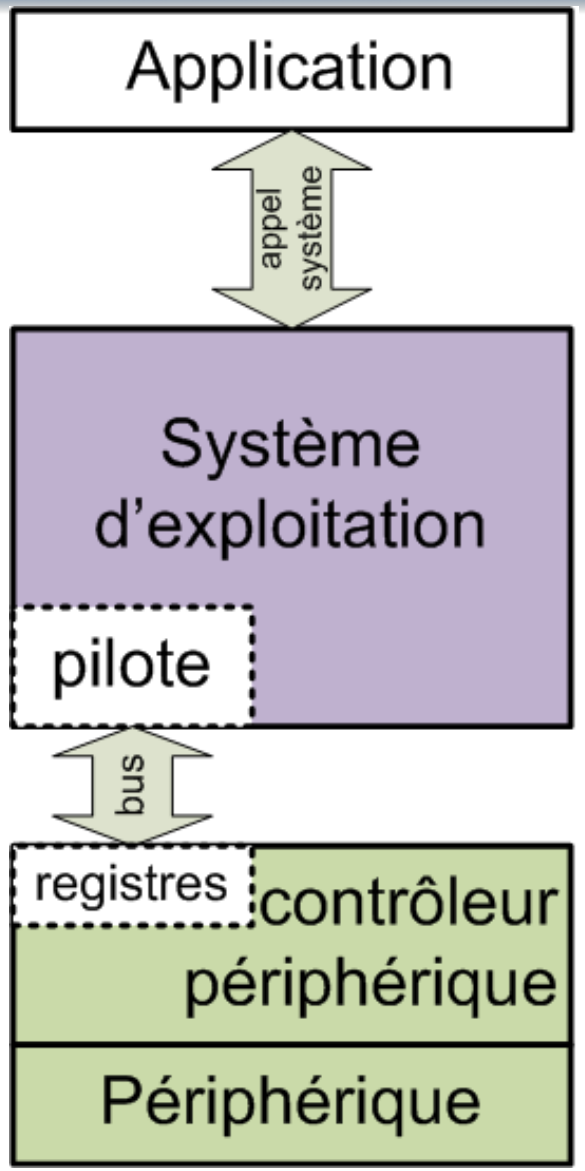
Disque

- Disques magnétiques
- Tête lecture-écriture
- Capacité de stockage élevée
  - 4TB vs *16 GB RAM*
- Non-volatile
- Temps accès : **~10 ms**
- Débit : 50-160 MB/s

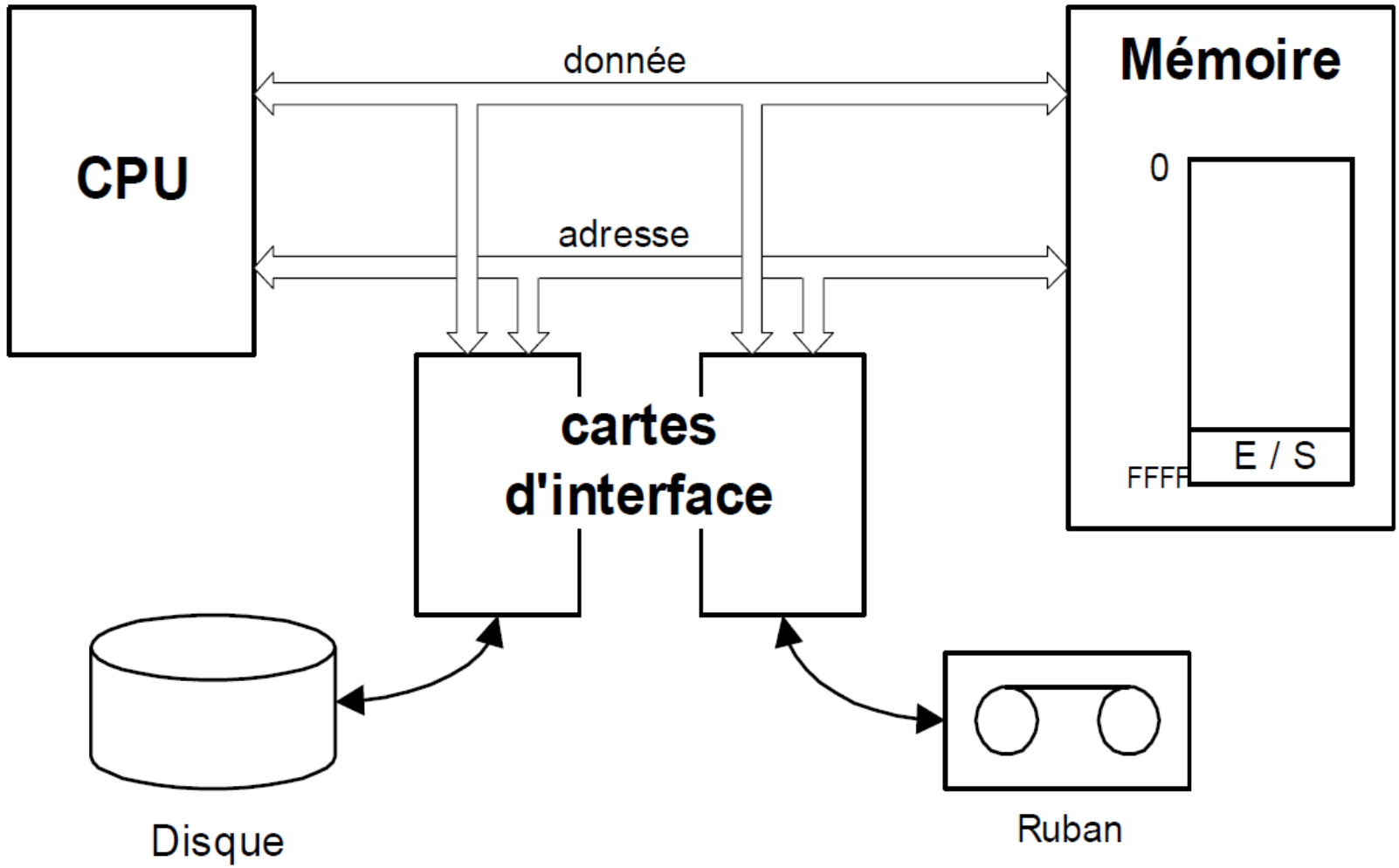


Périphériques d'entrées/sorties (E/S)

- Pilote de périphérique
- Fournis par le manufacturier
  - 1 pilote par système d'exploitation
- Pilote tourne généralement en mode **noyau** (*kernel)*
  - accès sans restrictions aux bus

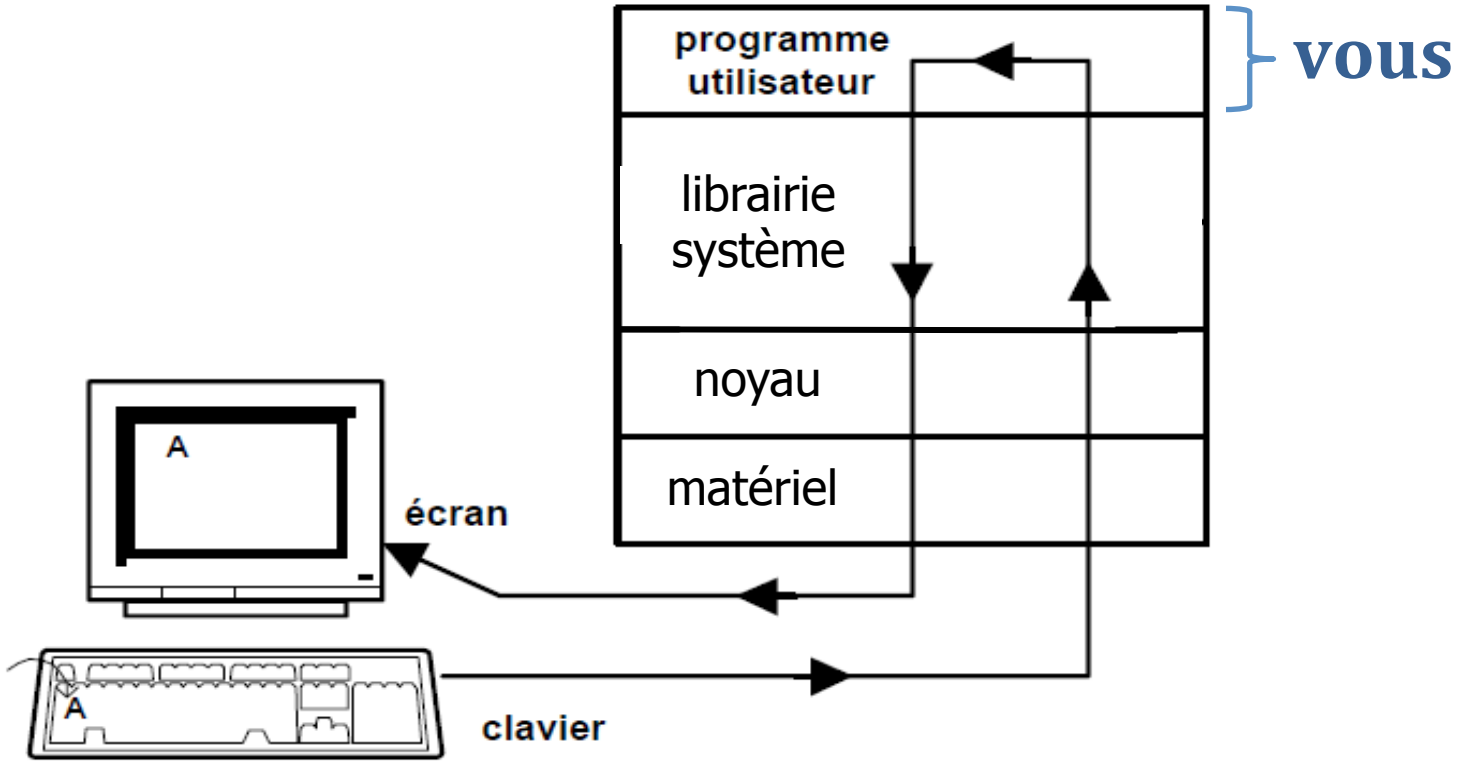


Bus de données (simplifiée)



Structure du système d’exploitation

Structure en couche du S.E.



Les informations traversent les couches du système

Composantes du S.E.

- **Le noyau**
  - partie centrale du S.E.
  - y consacrer la majorité du temps en classe
  - tourne niveau privilège **noyau (0)** du processeur
- **La librairie système**
  - tourne niveau privilège **utilisateur (3)**
  - implémente interface pour C/C++
- **Les programmes systèmes**
  - `ls`, `cp`, `mv`

Le noyau (kernel)

- Partie essentielle du S.E.
- Les fonctionnalités communes des S.E. y sont généralement implémentées
- Beaucoup d'information y réside
  - tableaux gestion processus/mémoire/fichier
- Tourne au niveau de privilège **noyau (0)**
  - Accès à toutes les informations /périphériques
  - Protège S.E. et information contre les utilisateurs
  - « Blindage »

SYSTÈMES D'EXPLOITATION I

Chapitre 1 (suite)

Dr. Mohamed Barkaoui
Département Informatique FSB

Interruption et S.E.

- Utilisés abondamment
  - périphériques
    - interruption matérielle sur touche enfoncée/relâchée
  - ordonnanceur préemptif
    - interruption périodique sur horloge
  - appels systèmes

- Impact sur le processeur et temps d'exécutions

Interruption (rappel)

- Déroutement du fil d'exécution en cours
  - division par zéro,
  - instruction invalide,
  - erreur mémoire (*segmentation fault*),
  - temporisateur (*timer*),
  - entrée/sortie,
  - définissable
  - etc..

0	divide error
1	debug exception
2	NMI interrupt
3	Breakpoint
4	INTO-detected Overflow
5	BOUND range exceeded
6	Invalid opcode
7	coprocessor not available
8	double fault
9	coprocessor segment overrun
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	reserved
16	coprocessor error
17-31	reserved
32-255	maskable interrupts

Liste des interrupts pour IA-32

Interruption et registre

Privilège

utilisateur (3)

mémoire

A=?

Registre **EIP**

Registre **EAX**

----

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP

pile **utilisateur**

Interruption et registre

action CPU: **charge instruction**

Privilège: **utilisateur (3)**

mémoire: **A=?**

Registre **EIP**: **0x001F138E**

Registre **EAX**: **----**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP ⇨ pile **utilisateur**

Interruption et registre

action CPU: **exécute instruction**

Privilège

utilisateur (3)

mémoire

A=3

Registre **EIP**

0x001F138E

Registre **EAX**

----

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP

pile **utilisateur**

Interruption et registre

action CPU: **charge instruction**

Privilège: **utilisateur (3)**

mémoire: **A=3**

Registre **EIP**: **0x001F1395**

Registre **EAX**: **----**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP ⇨ pile **utilisateur**

Interruption et registre

action CPU: **exécute instruction**

Privilège

utilisateur (3)

mémoire

A=3

Registre **EIP**

0x001F1395

Registre **EAX**

3

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP

pile **utilisateur**

Interruption et registre

action CPU: **charge instruction**

Privilège: **utilisateur (3)**

mémoire: **A=3**

Registre **EIP**: **0x001F1398**

Registre **EAX**: **3**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP ⇨ pile **utilisateur**

Interruption et registre

action CPU: **exécute instruction**

Privilège: **utilisateur (3)**

mémoire: **A=3**

Registre **EIP**: **0x001F1398**

Registre **EAX**: **7**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP ⇨ pile **utilisateur**

Interruption et registre

action CPU: **interruption 0x44**

Privilège

utilisateur (3)

mémoire

A=3

Registre **EIP**

0x001F1398

Registre **EAX**

7

INTERRUPTION 0x44!

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP

pile **utilisateur**

Interruption et registre

action CPU: **interruption 0x44**

Privilège: **utilisateur (3)**

mémoire: **A=3**

Registre **EIP**: **0x001F1398**

Registre **EAX**: **7**

code du processus

```
int A;
001F138E  mov   dword ptr [A],3
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

INTERRUPTION 0x44!

- **CPU consulte la table des interruptions IDT**

table des interruptions

code	adresse
0x44	**0x00000A45**

Interruption et registre

action CPU: **interruption 0x44**

Privilège: **noyau (0)**

mémoire: **A=3**

Registre **EIP**: **0x001F1398**

Registre **EAX**: **7**

code du processus

```
int A;
...
001F138F  mov  dword ptr [A],3
...
001F1395  mov  eax,dword ptr [A]
001F1398  add  eax,4
001F139B  mov  dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov  eax,666
...
```

INTERRUPTION 0x44!

- **CPU consulte la table des interruptions IDT**
- **Change niveau privilège vers noyau (0) et pile vers noyau**

table des interruptions

code	adresse
0x44	**0x00000A45**

pile **noyau** ← **ESP**

pile **utilisateur**

Interruption et registre

action CPU: **charge adresse interrupt 0x44**

Privilège: **noyau (0)**

mémoire: **A=3**

Registre **EIP**: **0x00000A45**

Registre **EAX**: **7**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

ESP

pile **noyau**

pile **utilisateur**

Interruption et registre

action CPU: **charge instruction**

Privilège: **noyau (0)**

mémoire: **A=3**

Registre **EIP**: **0x00000A45**

Registre **EAX**: **7**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau** ← **ESP** → pile **utilisateur**

Interruption et registre

action CPU: **exécute instruction**

Privilège: **noyau (0)**

mémoire: **A=3**

Registre **EIP**: **0x00000A45**

Registre **EAX**: **666**

écrase la valeur ☹

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau** ← **ESP** → pile **utilisateur**

Interruption et registre

action CPU: **continue…**

Privilège: **noyau (0)**

mémoire: `A=3`

Registre **EIP**: `0x00000A45`

Registre **EAX**: `666`

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau** ← **ESP** → pile **utilisateur**

Interruption et registre

action CPU: **continue…**

Privilège: **noyau (0)**

mémoire: **A=3**

Registre **EIP**: **0x00000A45**

Registre **EAX**: **666**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

retour de l'interruption

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau** ⇦ **ESP** pile **utilisateur**

Interruption et registre

action CPU: **restaurer le registre de pointeur de pile**

Privilège: **noyau (0)**

mémoire: **A=3**

Registre **EIP**: **0x00000A45**

Registre **EAX**: **666**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP ⇨ pile **utilisateur**

Interruption et registre

action CPU: **restaurer EIP**

Privilège: **noyau (0)**

mémoire: `A=3`

Registre **EIP**: `0x001F139B`

Registre **EAX**: `666`

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP ⇨ pile **utilisateur**

Interruption et registre

action CPU: **retour au privilège utilisateur**

Privilège: **utilisateur (3)**

mémoire: **A=3**

Registre **EIP**: **0x001F139B**

Registre **EAX**: **666**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP ⇨ pile **utilisateur**

Interruption et registre

action CPU: **charge instruction**

Privilège: **utilisateur (3)**

mémoire: **A=3**

Registre **EIP**: **0x001F139B**

Registre **EAX**: **666**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP ⇨ pile **utilisateur**

Interruption et registre

action CPU: **exécute instruction**

Privilège

utilisateur (3)

mémoire

```
A=666
```

Registre **EIP**

0x001F139B

Registre **EAX**

666

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

pile **noyau**

ESP

pile **utilisateur**

Interruption et registre

action CPU: **exécute instruction**

Privilège: **utilisateur (3)**

mémoire: **A=666**

réponse attendue est A=7!!!

Registre **EIP**: **0x001F139B**

Registre **EAX**: **666**

code du processus

```
        int A;
        A = 3;
001F138E  mov   dword ptr [A],3
        A = A + 4;
001F1395  mov   eax,dword ptr [A]
001F1398  add   eax,4
001F139B  mov   dword ptr [A],eax
```

code de l'interrupt

```
00000A45  mov   eax,666
          ...
```

table des interruptions

code	adresse
0x44	0x00000A45

d'où l'importance de préserver le contexte!

pile **noyau**

ESP ⇨ pile **utilisateur**

Contexte du processeur

- Une photo instantanée de l'état du CPU (registres)

```
// A struct describing a Task State Segment.
struct tss_entry_struct {
  u32int prev_tss; // The previous TSS
  u32int esp0;     // The stack pointer to load when we change to kernel mode.
  u32int ss0;      // The stack segment to load when we change to kernel mode.
  u32int esp1;     // Unused...
  u32int ss1;
  u32int esp2;
  u32int ss2;
  u32int cr3;
  u32int eip;
  u32int eflags;
  u32int eax;
  u32int ecx;
  u32int edx;
  u32int ebx;
  u32int esp;
  u32int ebp;
  u32int esi;
  u32int edi;
  u32int es;   // The value to load into ES when we change to kernel mode.
  u32int cs;   // The value to load into CS when we change to kernel mode.
  u32int ss;   // The value to load into SS when we change to kernel mode.
  u32int ds;   // The value to load into DS when we change to kernel mode.
  u32int fs;   // The value to load into FS when we change to kernel mode.
  u32int gs;   // The value to load into GS when we change to kernel mode.
  u32int ldt;  // Unused...
  u16int trap;
  u16int iomap_base;
} __attribute__((packed));
```

Exemple de contexte

tiré de PrettyOS

processeur INTEL IA-32

Préserver l'état (contexte) du CPU

- Pour restituer l'état original du CPU avant interrupt
- Séquence lors du déclenchement de l'interruption :
  - Le CPU place sur la pile le PC (EIP)
  - Le CPU charge le PC (EIP) avec vecteur interruption de l'IDT
  - Une procédure en assembleur/CPU sauve les registres (contexte)
  - Une procédure en assembleur/CPU pointe vers une autre pile
  - Service d'interruption s'exécute

PC : Program Counter/Compteur ordinal

Préserver l'état (contexte) du CPU

- Pour restituer l'état original du CPU avant interrupt
- Séquence lors du déclenchement de l'interruption :
  - Le CPU place sur la pile le PC (EIP)
  - Le CPU charge le PC (EIP) avec vecteur interruption de l'IDT
  - Une procédure en assembleur/CPU sauve les registres (contexte)
  - Une procédure en assembleur/CPU pointe vers une autre pile
  - Service d'interruption s'exécute
- Changement de **contexte** : *context switch*
  - processus A → processus B
  - thread A1 → thread A2
  - processus A (**utilisateur(3)**) → processus A (**noyau(0)**)

appel système

Interruption : coût

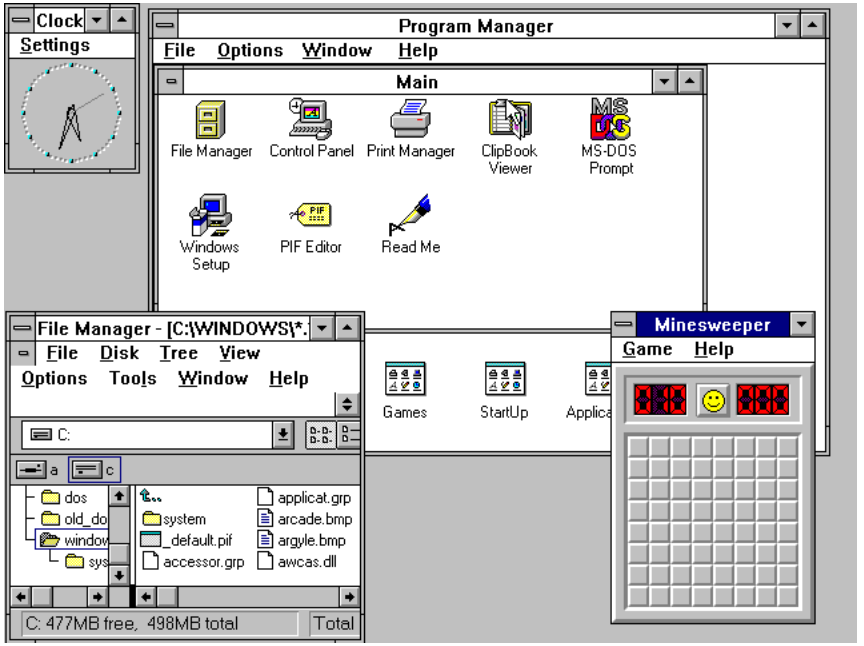
- Entre autres :
  - Doit placer le **contexte** du CPU en mémoire
  - Doit remettre le CPU dans le même **contexte** après
  - Pollution de la mémoire cache

- Plusieurs centaines de cycles-machines
- Éviter d'avoir des interruptions trop fréquentes…

  …mais système moi réactif (temps réel?)

Partage du CPU : coopératif

- Partager de façon **coopérative**?
- Si application est « égoïste » (ne libère pas volontairement le CPU), tout le système est bloqué
- **Windows 3.x** : les messages du S.E. devaient être traité le plus rapidement possible



(Windows == Event driven)

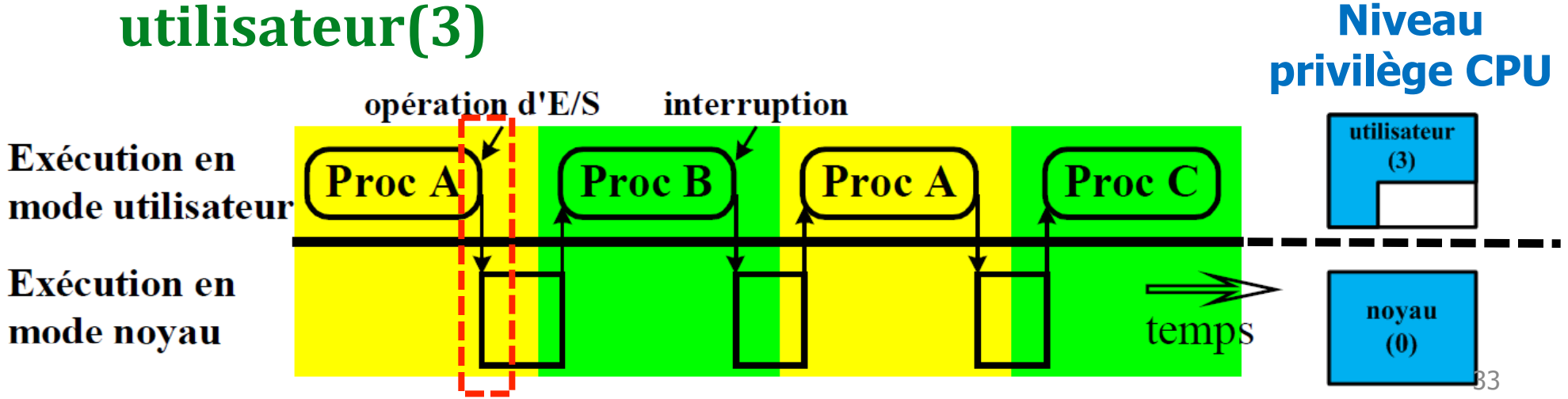
Partage du CPU : préemption par interruption

- Pour éviter qu'une seule application ne prenne tout le CPU, le S.E. doit **forcer** des changements de contexte
- Mécanisme : interruption programmée sur horloge (*timer*)
- Timer sur le CPU : **registre** décrémenté à chaque coup d'horloge
- Quand (`registre` == 0), CPU fait une interruption → IDT → va appeler le S.E. en mode **noyau (0)**

Appel système

Appel système

- Un appel système :
  - est une fonctionnalité fournie par le noyau du S.E.
  - fait par l'entremise de la librairie système
  - doit tourner en mode **noyau(0)**
    - tâches qui nécessite accès complet à la machine (info, bus, etc)
  - l'appelant (votre programme) tourne en mode **utilisateur(3)**



Exemple appels systèmes Linux/UNIX

- **`read/write`** : lecture/écriture fichier
- **`brk`** : gestion de la mémoire
- **`fork`** : créer un nouveau processus
- **`waitpid`** : attendre la fin d'un processus
- **`kill`** : tuer un processus
- **`time`** : avoir l'heure

} **autour de 300**

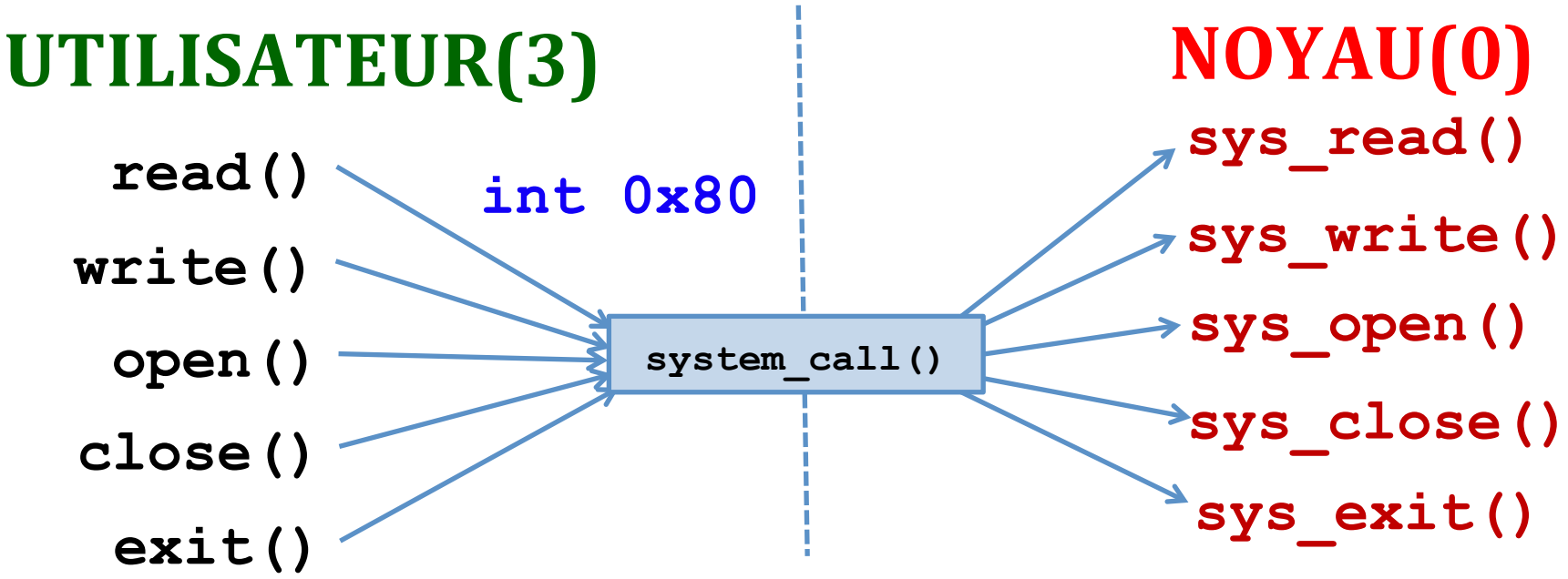
Liste : http://www.digilife.be/quickreferences/qrc/linux%20system%20call%20quick%20reference.pdf

Appel système : changement privilège

- Nécessite un changement de niveau de privilège du CPU : **UTILISATEUR (3)** → **NOYAU (0)**
  - ne peut pas permettre à un utilisateur de le changer à sa guise
- Fait par
  - interruption logicielle **`INT`** ***n*** ou
  - instruction machine spéciale **`SYSCALL`**/**`SYSENTER`**

Déroutement noyau (Linux) par int. logicielle

- Tous les appels systèmes transitent à travers une seule fonction noyau **`system_call()`**, via interruption logicielle (`set_system_gate(0x80,&system_call)`)



- Vous n'appelez donc jamais directement les fonctions **`sys_xxx( )`**

Aiguillage dans le noyau

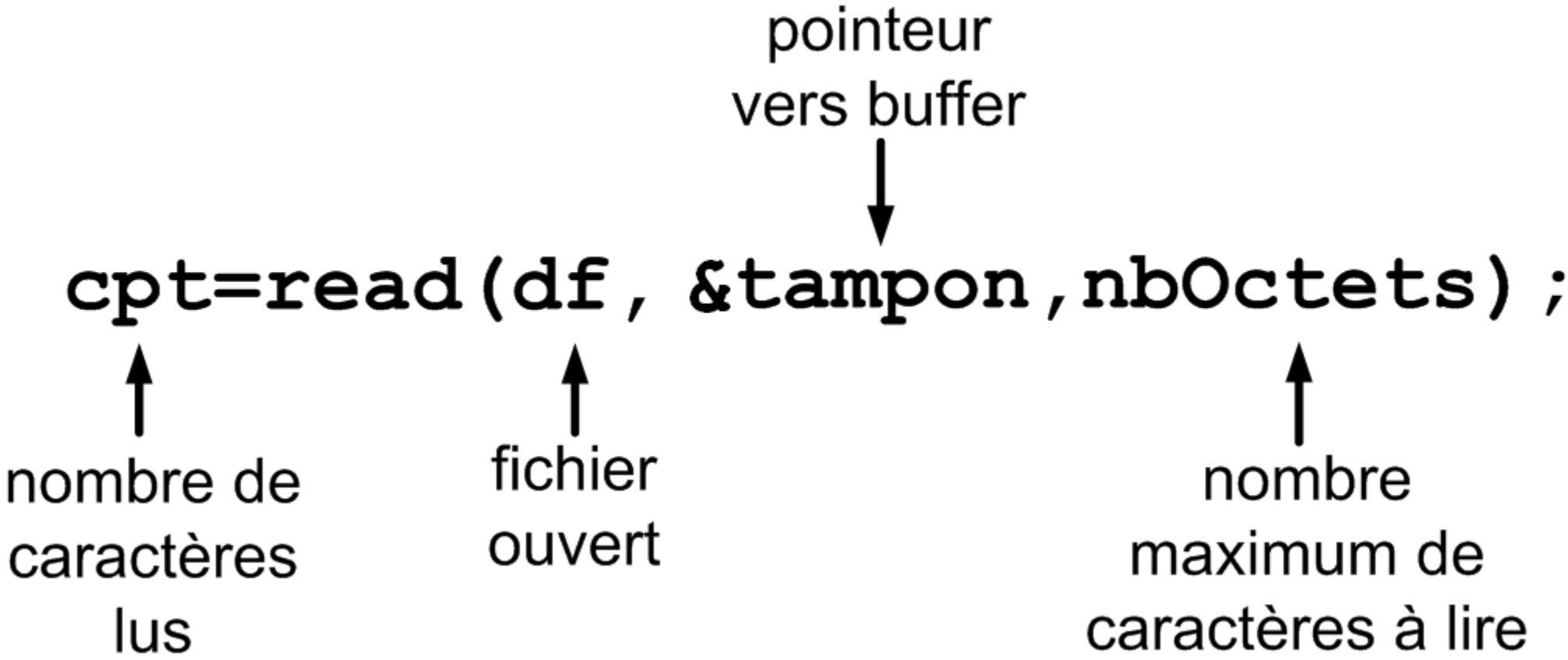
- Comment le noyau sait quelle fonction exécuter : code numérique unique (placé dans registre EAX)

1	exit	terminate the current process	...	...	...
2	fork	create a child process	185	capset	set process capabilities
3	read	read from a file descriptor	186	sigaltstack	set/get signal stack context
4	write	write to a file descriptor	187	sendfile	transfer data between file descriptors
5	open	open a file or device	188	getpmsg	(unimplemented)
6	close	close a file descriptor	189	putpmsg	(unimplemented)
7	waitpid	wait for process termination	190	vfork	create a child process and block parent

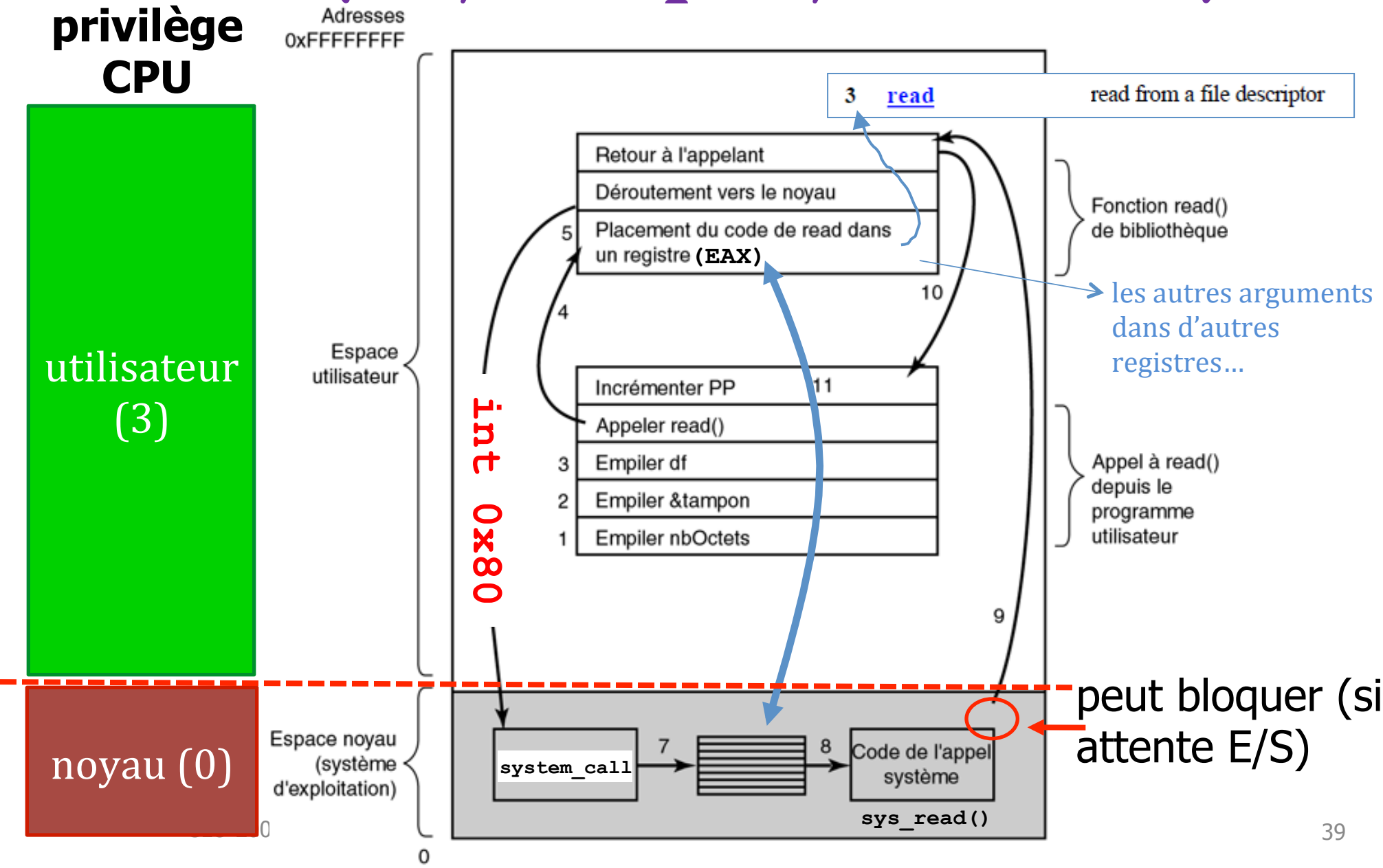
```
exit(-1)
syscall(SYS_exit,-1)
syscall(1,-1)
```

} exactement la même chose

Exemple d'exécution d'un appel système : `read()`



`read(df,&tampon,nbOctets);`



Par opposition : appel de fonction

- Un appel de fonction correspond à un saut à une adresse unique par fonction

- Pas de changement de niveau de privilège

Systèmes
d'Exploitation

Synchronisation des Processus

Sections critiques,
Sémaphores

Présentation
Synchronisation des Processus

- Sur une plateforme multiprogrammée, les processus ont généralement besoin de communiquer pour compléter leurs tâches.

- Un processus est dit indépendant, s'il n'affecte pas les autres processus ou ne peut pas être affecté par eux.
  - Un processus qui ne partage pas de données avec d'autres processus est indépendant.

- Un processus est dit coopératif s'il peut affecter les autres processus en cours d'exécution ou être affecté par eux.
  - Un processus qui partage des données avec d'autres processus est un processus coopératif.

- Les données partagées par les processus coopératifs se trouvent en mémoire principale ou en mémoire secondaire dans un fichier
  - Il y a alors risque d'incohérence des données.

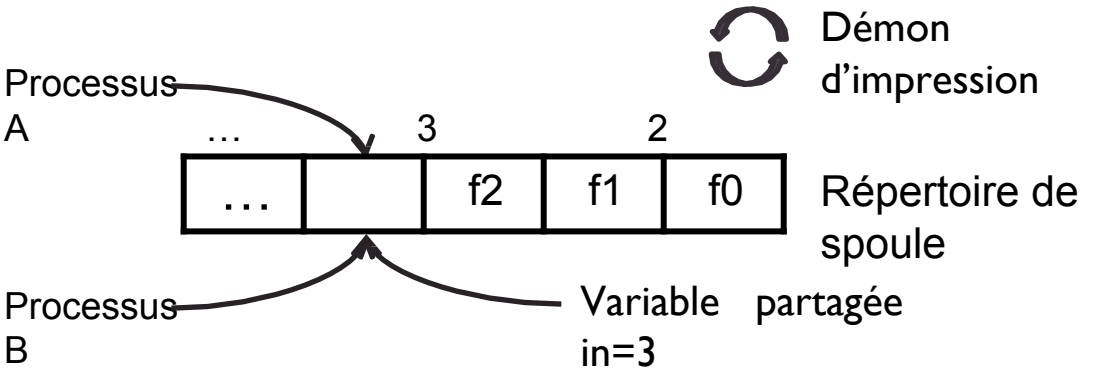
Synchronisation des Processus

- L'exécution d'un processus peut être affectée par l'exécution des autres processus ou il peut affecter lui-même leurs exécutions

- La communication interprocessus est assurée généralement via des données partagées qui peuvent se trouver dans la mémoire principale ou dans un fichier.
  - Les accès concurrents (simultanés) à des données partagées peuvent conduire à des incohérences dans les résultats obtenus.

Exemple Illustratif
Synchronisation des Processus

Exemple : la spoule d'impression



lire *in*,
next_free_slo
t = *in*
Entrée[next_free_slot] = fichier
in = *in* +1

Schéma d'exécution:

A : l ire in ,
next_free_slot = 3
Préemption: la CPU bascule vers le processus B

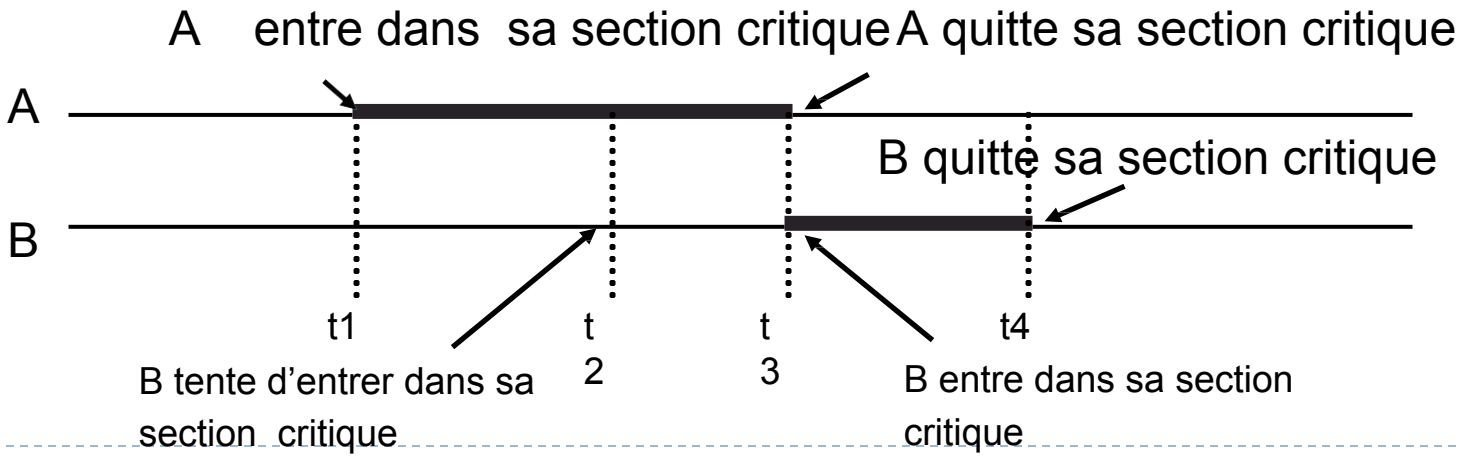
B : lire in,
next_free_slo
t = 3,
entrée3 = fichierB, in = 4

A : entrée3 = fichierA,
in = 4

⇒ Problème : le fichier B ne sera jamais imprimé

Sections Critiques et Exclusion Mutuelle
Synchronisation des Processus

- Le problème précédent est dû aux conflits d'accès à la même ressource.
- La partie du programme à partir de laquelle on accède à la ressource partagée est appelée section (région) critique.
- Solution:
  - L'exclusion mutuelle est une méthode qui assure qu'un seul processus est autorisé d'accéder à une ressource partagée; les autres processus seront exclus de la même activité.



Sections critiques et exclusion mutuelle
Synchronisation des Processus

- Quatre conditions doivent être vérifiées pour assurer une bonne synchronisation des processus :

1. Exclusion Mutuelle: Deux processus ne doivent pas se trouver simultanément dans leurs sections critiques.
2. Progression : Aucun processus à l'extérieur de sa section critique ne doit bloquer les autres processus.
3. Attente bornée : Aucun processus ne doit attendre indéfiniment pour entrer dans sa section critique.
4. Aucune hypothèse : Il ne faut pas faire d'hypothèse quant à la vitesse ou le nombre de processeurs

Synchronisation des Processus

EXCLUS ION MUTUELLE PAR ATTENTE ACTIVE

Approche de Synchronisation
Exclusion Mutuelle par Attente Active

- Un processus désirant entrer dans une section critique doit être mis en attente jusqu'a ce que la section critique devienne libre.
- Un processus quittant la section critique doit le signaler aux autres processus.

Algorithme d'accès à une section critique :

```
Entrer_Section_Critique ()  /     attente si SC non libre    /
Section_Critique()
Quitter_Section_Critique()*/     un seul processus* en SC/
```

- L'attente peut être :
  - Active : la procédure *Entrer_Section_Critique* est une boucle dont la condition est un test qui porte sur des variables indiquant la présence ou non d'un processus en Section critique.
  - Non active : le processus passe dans l'état endormi et ne sera réveillé que lorsqu'il sera autorisé à entrer en section critique.

- Solution 1: Masquage des interruptions
  - Lorsqu'un processus entre en section critique il doit masquer les interruptions.
  - Pas de commutation de contexte
  - Lorsqu' il quitte sa section critique il doit restaurer les interruptions.
  - C'est une solution matérielle qui permet de résoudre complètement le problème. Mais elle est dangereuse en mode utilisateur s'il oublie de restaurer les interruptions.

Solutions de l'Exclusion Mutuelle par Attente Active

Exclusion Mutuelle par Attente Active

- Solution 2: Variable de verrouillage
  - Un verrou est variable binaire partagée qui indique la présence d'un processus en section critique.
    - si verrou=0 alors section critique libre
    - si verrou=1 alors section critique occupée
  - Cette solution ne garantit pas l'exclusion mutuelle car le verrou est une variable partagée qui peut constituer aussi une section critique.

```
void entrer_Section_Critique ()
{
  while (verrou == 1);     /* attente active */
  verrou=1 ;
}
```

```
void quitter_Section_Critique ()
{
  verrou=0 ;
}
```

Exclusion Mutuelle par Attente Active

- Solution 2-bis : Variable de verrou en matériel: TSL (Test and Set Lock)
  - Aussi appelée TAS pour Test And Set.
  - Est une instruction indivisible: réalisée une seule fois par le matériel.

  - (1) Lit le contenu de LOCK dans REGISTRE
  - (2) Ecrit 1 dans adresse mémoire LOCK

//(1) et (2) sont indivisibles
// (TSL instruction atomique)

```
TSL REGISTER, LOCK
```

```
|pseudo-assembleur
entrer_Section_Critique:
        TSL REGISTER, LOCK      | copie lock dans reg et la définit à 1
        CMP REGISTER, #0        | lock etait-elle à 0
        JNE entrer_SC           | si elle était pas à 0, boucle
        RET                     | retourne à appelant, entre dans SC
```

```
quitter_Section_Critique:
MOVE LOCK, #0          | stocke un 0 dans lock
RET                    | retourne à l'appelant
```

Exclusion Mutuelle par Attente Active

- Solution 3: Alternance stricte
  - Tour est une variable partagée qui indique le numéro de processus autorisé à entrer en section critique.
  - L'alternance stricte est une solution simple et facile a implémenter.
  - Mais, un processus qui possède Tour peut ne pas être intéressé immédiatement par la section critique et en même temps il bloque un autre processus qui la demande.
  - Problème de Progression

```
void entrer_Section_Critique (int process)
{
   while (Tour!=process) ; /* attente active */
}
```

```
void quitter_Section_Critique ()
{
   Tour = (Tour+1) %N ;
}
```

Exclusion Mutuelle par Attente Active

- Solution 4: Alternance stricte v2
  - *Interested* est un tableau de booléens partagé ayant une case pour chaque processus qui veut entrer en SC
  - Initialement, toutes les valeurs du tableau sont à *false*
  - Quand un processus exécute *entrer_SC*:
    - Il met sa variable *interested* à true
    - Il attend si l'autre processus est intéressé
  - Problème d'interblocage
    - Excès de courtoisie!

```
void entrer_Section_Critique (int process)
{
        interested[process] = true;
        while (interested[1-process]) ; * /□ attente active /
}
```

```
void quitter_Section_Critique (int process)
{
   interested[process]= false;
}
```

Solutions de l'Exclusion Mutuelle par Attente Active

Exclusion Mutuelle par Attente Active

- Solution 5: Alternance stricte v2
  - Cette solution combine les deux version de l'alternance stricte
  - Cette solution assure les quatre conditions de bonne synchronisation.
  - Mais, le processus qui attend sa section critique consomme du temps processeur inutilement (attente active).

```
#define FAUX  0
#define VRAI 1
#define N 2

int tour ;      /* à qui le tour */
int interesse[N] ;   /* initialisé à FAUX */
void entrer_Section_Critique (int process)
{
        int autre ;
    (1)  autre = 1-process ;
    (2)  interesse[process]=VRAI;   /* process est intéressé */
    (3)  tour = process ;           /* demander le tour */
        while (tour == process && interesse[autre] == VRAI) ;
}
     (A)    (B)
```

```
Void quitter_Section_Critique (int process)
{
 (4) interesse[process]=FAUX ;
}
```

Peterson N-processus

Exclusion Mutuelle par Attente Active

```
# define FALSE  0
# define N    10   /□ First process is indicated with 1, not 0 /
int turn[N];
int stage[N + 1 ];
```

```
void enterRegion( int process)
{
  int i, j ;
  for ( i = 1 ; i <= N - 1; i++) {  stage[process] = i;
      turn[i] =  process;
      for ( j = 1 ; j <= N ; j ++) {  if ( j == process)
              continue;
         while ( stage[ j ] >= i && turn[i] == process);
      }
  }
}
```

```
void leave Region( int
process)
{
  stage[process] = FALSE;
}
```

Synchronisation des Processus

EXCLUSION MUTUELLE SANS ATTENTE ACTIVE

Exclusion mutuelle sans attente active

Exclusion Mutuelle Sans Attente Active

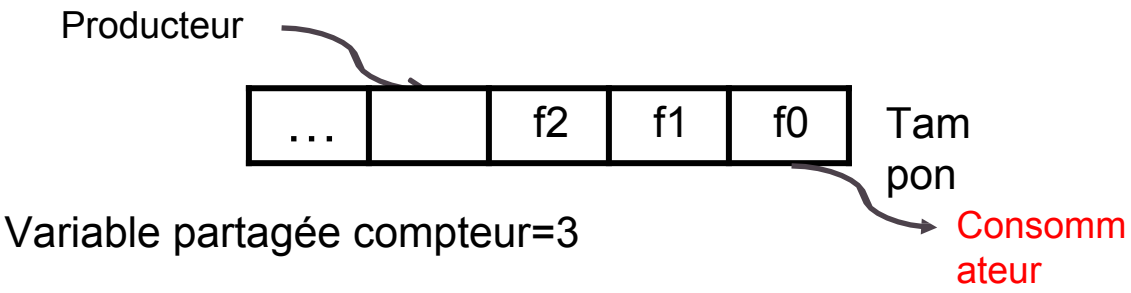
- L'idée est qu'un processus qui ne peut pas entrer en section critique passe à l'état bloqué au lieu de consommer le temps processeur inutilement. Il sera réveillé lorsqu'il pourra y entrer.

- Les primitives Sleep et Wakeup:
  - Le système d'exploitation offre deux appels système:
    1. Sleep (dormir) qui bloque le processus appelant.
    2. Wakeup (réveiller) qui réveille le processus donné en argument.

Producteur-Consommateur

Exclusion Mutuelle Sans Attente Active

- Application des primitives Sleep et Wakeup au modèle Producteur Consommateur:



- Deux processus (le producteur et le consommateur) coopèrent en partageant un même tampon:
  - Le producteur produit des objets qu'il dépose dans le tampon.
  - Le consommateur retire des objets du tampon pour les consommer.

Producteur-Consommateur

Exclusion Mutuelle Sans Attente Active

```
#define N 100 /* taille du tampon */
int compteur = 0; /* objets dans tampon */
void producteur(){
while(TRUE)
   {
      produire_objet();
      if(compteur==N)sleep();
      mettre_objet();
      compteur=compteur+1;
      if(compteur==1)
          wakeup(consommateur);
    }
}
```

```
void consommateur(){ while(TRUE)
  {
     if(compteur==0)
                sleep
     (); retirer_objet()
     compteur=compteur–1; if(compteur==N-1)
          wakeup(producteur);
     consommer_objet(...);
   }
}
```

Producteur-Consommateur

Exclusion Mutuelle Sans Attente Active

- Analyse de cette solution :
  - L'accès à la variable compteur n'est pas protégé, ce qui peut entraîner des incohérences dans les valeurs prises par cette variable.
  - Réveils perdus :
    - c'est le principal défaut de ce mécanisme
    - Un signal *wakeup* envoyé à un processus qui ne dort pas (encore) est perdu.
      - Considérer le scénario où une préemption a lieu entre le *if* et le *sleep* du producteur, par exemple.

Sémaphores
Exclusion Mutuelle Sans Attente Active

- **Pour remédier au problème es réveils en attente (les wakeup perdus), l'idée est d'employé une variable entière appelée: Sémaphore à laquelle est associée une file d'attente des processus bloqués.**

- sémaphore=0 → aucun réveil n'est mémorisé
- sémaphore>0 → un ou plusieurs réveils sont en attente

- Un sémaphore s est manipulé par les opérations atomiques suivantes:

1. down(s) :
   - if (val(s)==0) sleep();
   - val(s)--;
2. up(s) :
   - wakeup();
   - s++;

Sémaphores
Exclusion Mutuelle Sans Attente Active

- Pour assurer l'exclusion mutuelle un sémaphore peut être programmé de la manière suivante :


```
initialisation mutex = 1        /* nombre de processus autorisés à entrer
                                   simultanément dans la section critique  */

down (mutex)
<section_critique>  up
(mutex)
```

Sémaphores

Exclusion Mutuelle Sans Attente Active

- Application au modèle Producteur / Consommateur :

- Trois sémaphores sont nécessaires:
  - plein : compte le nombre de places occupées
  - vide : compte le nombre de places libres
  - mutex : assure que le producteur et le consommateur n'accèdent jamais en même moment à la mémoire tampon.

Producteur/Consommateur avec Sémaphores

Exclusion Mutuelle Sans Attente Active

```
#define N 100                    // taille du tampon
semaphore mutex = 1;             // contrôle d'accès section critique
semaphore vide = N;              // contrôle les emplacements vide
semaphore plein = 0;             // contrôle les emplacements plein

void producteur ( ) {
while (TRUE){
        produire_objet() ;
        down(vide);
        down(mutex);
        mettre_objet() ;         // SC
        up(mutex);
        up( plein)
}
}
```

```
void consommateur ( ) {
while (TRUE){
        down( plein);
        down(mutex);
        retirer_objet()  // SC
        up(mutex);
        up(vide);
        consommer_objet(…) ;
        }
}
```