

# Valeur d'accueil et de reconversion en informatique VARI1 NFP135

Daniel Porumbel (dp.cnam@gmail.com)

**Contributions aux slides :**  
Pierre Cubaud  
Joelle Delacroix

# Organisation du cours

20% architecture, systèmes, réseaux

40% langage *processing*

40% langage *java*

Notation :

40% examen

40% projet

20% deux mini-devoirs

La salle de cours le lundi soir peut changer chaque semaine.

Salle TP 31.1.71 :

- 18h00-20h00 tous les mardis soir

soit

- 10h00-12h00 tous les samedis matin (au moins les 3 premières semaines, en fonction du nombre d'inscrits)

Site web :

<http://cedric.cnam.fr/~porumbed/varil/>

- 1 Les premiers programmes/dessins en langage `Processing`
- 2 Comprendre l'ordinateur : notions d'architecture
  - Les couches et les fonctions de base d'un ordinateur
  - Processeur, Mémoires, Bus, Périphériques
  - Du langage `processing` à la couche matérielle

- 1 Les premiers programmes/dessins en langage `Processing`
- 2 Comprendre l'ordinateur : notions d'architecture
  - Les couches et les fonctions de base d'un ordinateur
  - Processeur, Mémoires, Bus, Périphériques
  - Du langage `processing` à la couche matérielle

# Un premier programme *Processing*

```
int i , j ;  
i=11;  
j=9;  
int somme = i+j ;  
println ( "La_somme_vaut_" +somme) ;
```

On observe que les variables sont de type `int`.

Les types principaux sont : `int`, `float`, `char`, `String`, `boolean`

- 1 écrire un programme qui calcule la somme de 3 valeurs de type *int*
- 2 tracer une ellipse de taille  $60 \times 40$
- 3 tracer une ellipse 10 fois plus grande sur une surface de dessin de taille  $800 \times 600$
- 4 tracer 4 rectangles collés

# Les instructions

Une instruction = une commande qui définit une action :

- une **déclaration** de variable :

```
int x
```

- une **affectation** : donner une valeur (ou le résultat d'une expression) à une variable déclarée

```
x=9
```

- une **conditionnelle** (un **if**) : exécuter une suite d'instructions si une expression est vérifiée

```
if (x<10)
```

```
    println(x)
```

```
int i=4;
int j=9;
int somme = i+j;
if (somme<20){
    println("La somme vaut " +somme);
}
```

# Les instructions

Une instruction = une commande qui définit une action :

- une **déclaration** de variable :

```
int x
```

- une **affectation** : donner une valeur (ou le résultat d'une expression) à une variable déclarée

```
x=9
```

- une **conditionnelle** (un **if**) : exécuter une suite d'instructions si une expression est vérifiée

```
if (x<10)
```

```
    println(x)
```

```
int i=4;
int j=9;
int somme = i+j;
if (somme<20){
    println("La_somme_vaut_" +somme);
}
```

# Le point-virgule

- une instruction de déclaration ou d'affectation sera toujours suivie d'un point-virgule ;
- on dit que le point-virgule valide l'instruction
- nous écrirons une instruction par ligne par souci de clarté.
  - il est possible d'écrire tout le programme en une seule ligne

# Le if-else

```
if (a<b){
    println ("inf");
} else {
    println ("sup");
}
```

à l'intérieur de chaque branche, on peut ajouter d'autres `if`

```
if (a<b){
    println ("inf");
} else {
    println ("sup");
    if (a==b){
        println ("=");
    }
}
```

# Le if-else

```
if (a<b) {  
    println ("inf");  
} else {  
    println ("sup");  
}
```

à l'intérieur de chaque branche, on peut ajouter d'autres `if`

```
if (a<b) {  
    println ("inf");  
} else {  
    println ("sup");  
    if (a==b) {  
        println ("=");  
    }  
}
```

# Quelques programmes en mode texte

Utiliser des conditionnelles `if ( . . . ) . . .` pour :

- afficher « somme » si  $a = b + c$ , où  $a$ ,  $b$  et  $c$  représentent 3 variables initialisées au début du programme
- afficher la valeur minimale de 3 variables  $a$ ,  $b$  et  $c$
- afficher “vrai” si 3 entiers  $a$ ,  $b$  et  $c$  sont triés  $a < b < c$ .

# Quelques programmes en mode texte

Utiliser des conditionnelles `if ( . . . ) . . . .` pour :

- afficher « somme » si  $a = b + c$ , où  $a$ ,  $b$  et  $c$  représentent 3 variables initialisées au début du programme
- afficher la valeur minimale de 3 variables  $a$ ,  $b$  et  $c$
- afficher “vrai” si 3 entiers  $a$ ,  $b$  et  $c$  sont triés  $a < b < c$ .

# Quelques programmes en mode texte

Utiliser des conditionnelles `if ( . . . ) . . . .` pour :

- afficher « somme » si  $a = b + c$ , où  $a$ ,  $b$  et  $c$  représentent 3 variables initialisées au début du programme
- afficher la valeur minimale de 3 variables  $a$ ,  $b$  et  $c$
- afficher “vrai” si 3 entiers  $a$ ,  $b$  et  $c$  sont triés  $a < b < c$ .

# Quelques programmes en mode graphique

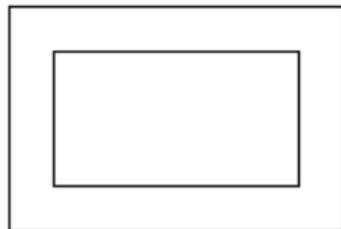
- 1 Tracer 3 cercles concentriques sur une toile de taille  $700 \times 700$
- 2 Tracer deux rectangles emboités :
  - le premier de taille  $500 \times 300$
  - le deuxième de taille  $400 \times 200$
- 3 Tracer deux cercles à l'intérieur d'un rectangle. Les deux cercles doivent se toucher entre eux.
- 4 Utiliser l'instruction `arc( . . . )` pour tracer un demi-cercle.

# Quelques programmes en mode graphique

① Tracer 3 cercles concentriques sur une toile de taille  $700 \times 700$

② Tracer deux rectangles emboîtés :

- le premier de taille  $500 \times 300$
- le deuxième de taille  $400 \times 200$



③ Tracer deux cercles à l'intérieur d'un rectangle. Les deux cercles doivent se toucher entre eux.

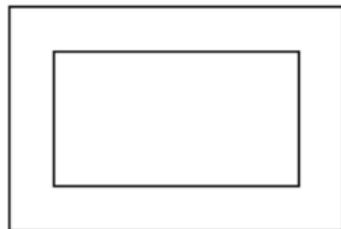
④ Utiliser l'instruction `arc(...)` pour tracer un demi-cercle.

# Quelques programmes en mode graphique

① Tracer 3 cercles concentriques sur une toile de taille  $700 \times 700$

② Tracer deux rectangles emboîtés :

- le premier de taille  $500 \times 300$
- le deuxième de taille  $400 \times 200$



③ Tracer deux cercles à l'intérieur d'un rectangle. Les deux cercles doivent se toucher entre eux.

④ Utiliser l'instruction `arc( . . . . )` pour tracer un demi-cercle.

# Des dessins en couleurs

- La couleur du dessin suivant est réglé par l'instruction `stroke(rouge,vert,bleu)` où rouge, vert et bleu representent des nombres de 0 à 255
- Utiliser `fill(rouge,vert,bleu)` pour le font de la figure (ellipse, rectangle, triangle)
- Tester aussi `noStroke();` et `strokeWeight(...);`
- Réaliser un par un les dessins ci-dessous

# Des dessins en couleurs

- La couleur du dessin suivant est réglé par l'instruction `stroke(rouge,vert,bleu)` où rouge, vert et bleu representent des nombres de 0 à 255
- Utiliser `fill(rouge,vert,bleu)` pour le font de la figure (ellipse, rectangle, triangle)
- Tester aussi `noStroke()`; et `strokeWeight(...)`;
- Réaliser un par un les dessins ci-dessous



# Bonus : utiliser des boucles `for`

Objectif : tracer plusieurs cercles à des positions aléatoires

- On peut obtenir une valeur aléatoire à l'aide de `random(...)`

Ajouter des couleurs aléatoires sur ces cercles...

Mélanger avec des triangles et rectangles de couleurs aléatoires

- 1 Les premiers programmes/dessins en langage `Processing`
- 2 Comprendre l'ordinateur : notions d'architecture
  - Les couches et les fonctions de base d'un ordinateur
  - Processeur, Mémoires, Bus, Périphériques
  - Du langage `processing` à la couche matérielle

# Comprendre la machine informatique

Pour comprendre le fonctionnement des ordinateurs, on va étudier :

- 1 architecture des ordinateurs
- 2 systèmes d'exploitation (prochain cours)
- 3 réseaux

- 
- Le terme « machine » ou « ordinateur » est à prendre dans son sens le plus large : « machine électronique capable d'exécuter des opérations arithmétiques et logiques »
  - Il peut désigner aussi bien un ordinateur de bureau ou portable (PC, Mac), un serveur de calcul ou encore un terminal mobile de type tablette ou smartphone.

1 Les premiers programmes/dessins en langage Processing

2 Comprendre l'ordinateur : notions d'architecture

- Les couches et les fonctions de base d'un ordinateur
- Processeur, Mémoires, Bus, Périphériques
- Du langage `processing` à la couche matérielle

# Machine informatique : couches génériques

## Couche logicielle

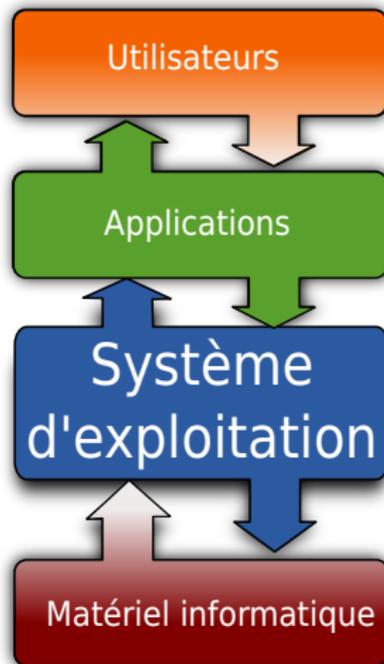
des programmes qui permettent à l'utilisateur de réaliser des tâches, y compris `processing` et `java`

## Couche Système d'Exploitation

dirige l'utilisation des ressources de la machine par les programmes de la couche logicielle

## Couche Matérielle

la machine physique y compris le processeur (CPU), la mémoire vive (RAM), disques durs, clés USB, imprimantes, etc.



# Fonctions du système d'exploitation (OS)

Le système d'exploitation : aussi appelé OS, de l'anglais Operating System.

---

## OS = Interface entre le matériel et les logiciels

Ressources matérielles : Processeur, mémoire vive (RAM), fichiers, réseaux, interface graphique utilisateur, périphériques, disques durs, contrôle d'accès pour plusieurs usagers simultanément, etc.

## Principaux OS

- Linux/Unix ses distributions (Ubuntu, Suze, Debian)
- Windows 95, Windows Vista, Windows 7, etc.
- MacOS, Android (basés sur des noyaux Linux)

# Composants couche physique (hardware)

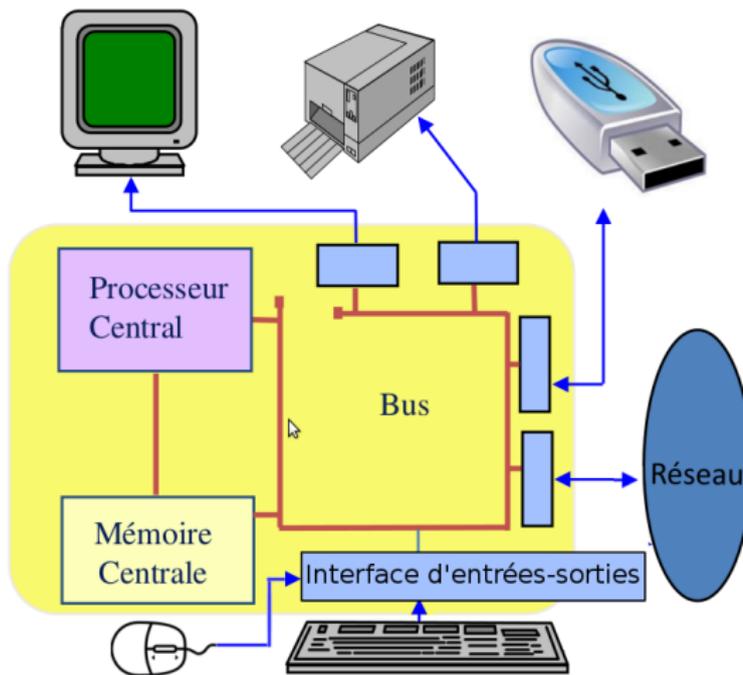
le **processeur** exécute les instructions machine, c'est le cerveau du système

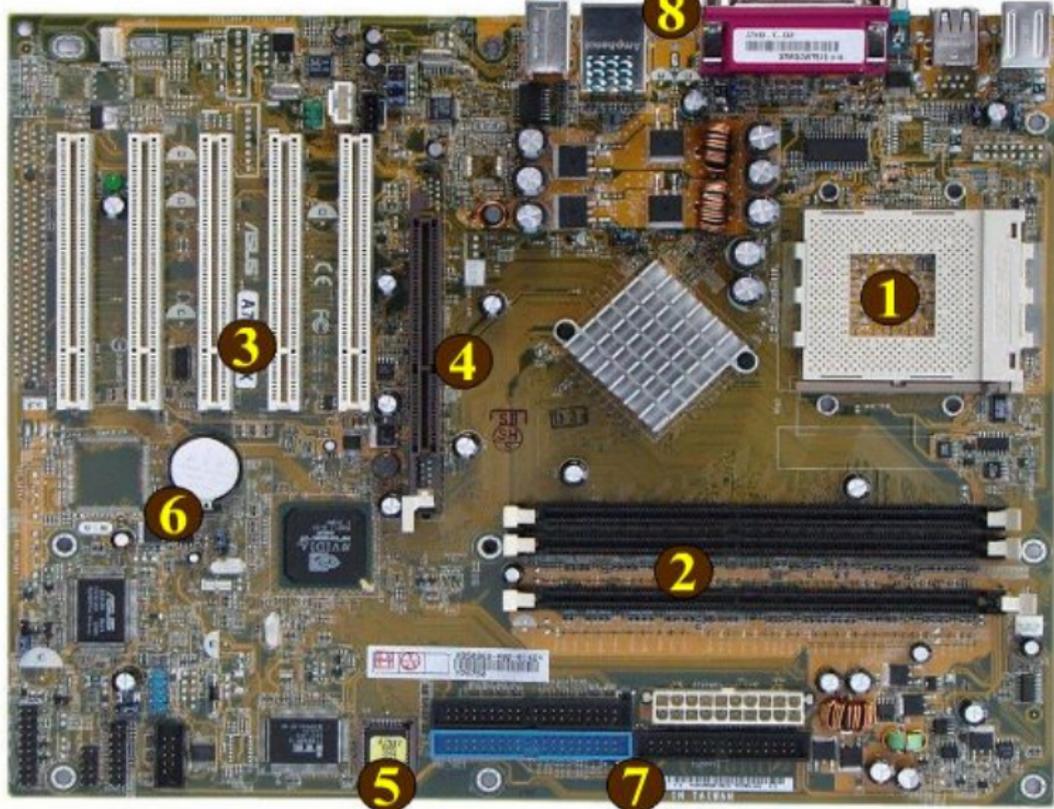
les **mémoires** vives (RAM, cache) stockent les données et les instructions

le **bus** permet le transfert de données entre les différents composants

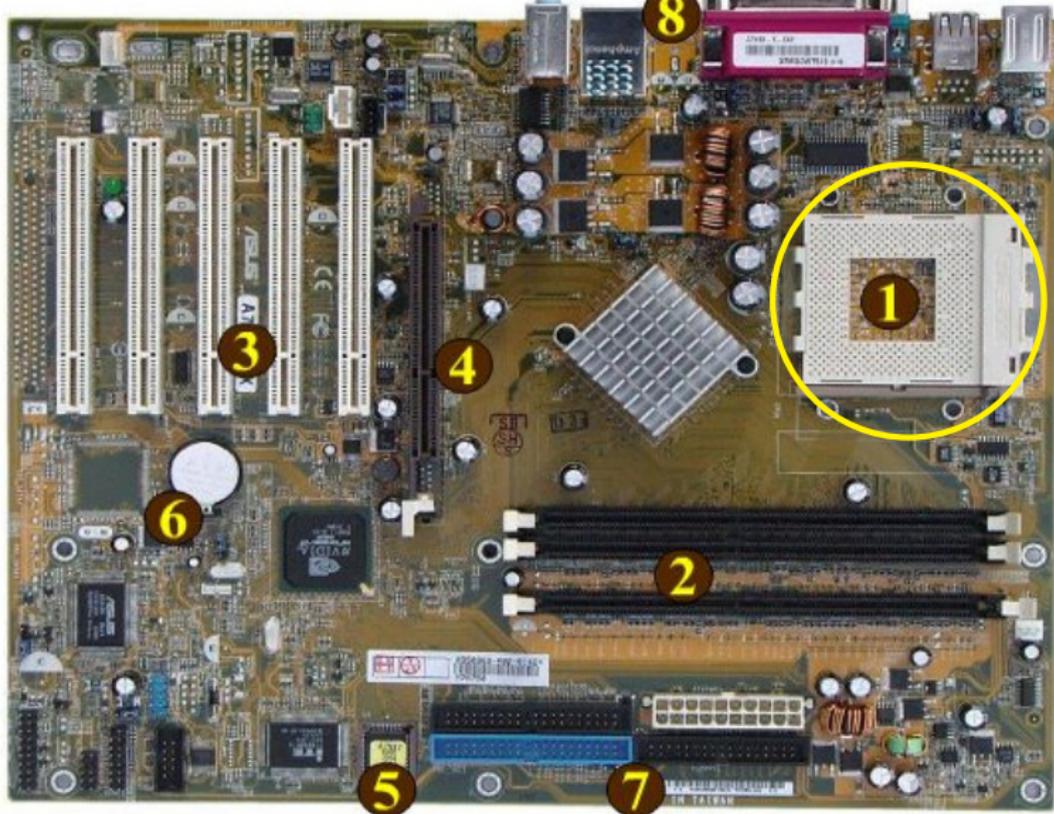
- souvent implémenté sur la carte mère.

les **périphériques** : disques durs, clés USB, imprimantes, moniteur (écran), clavier, souris, cartes d'extension (graphique), manettes de jeu, lecteurs de CD/DVD, etc.

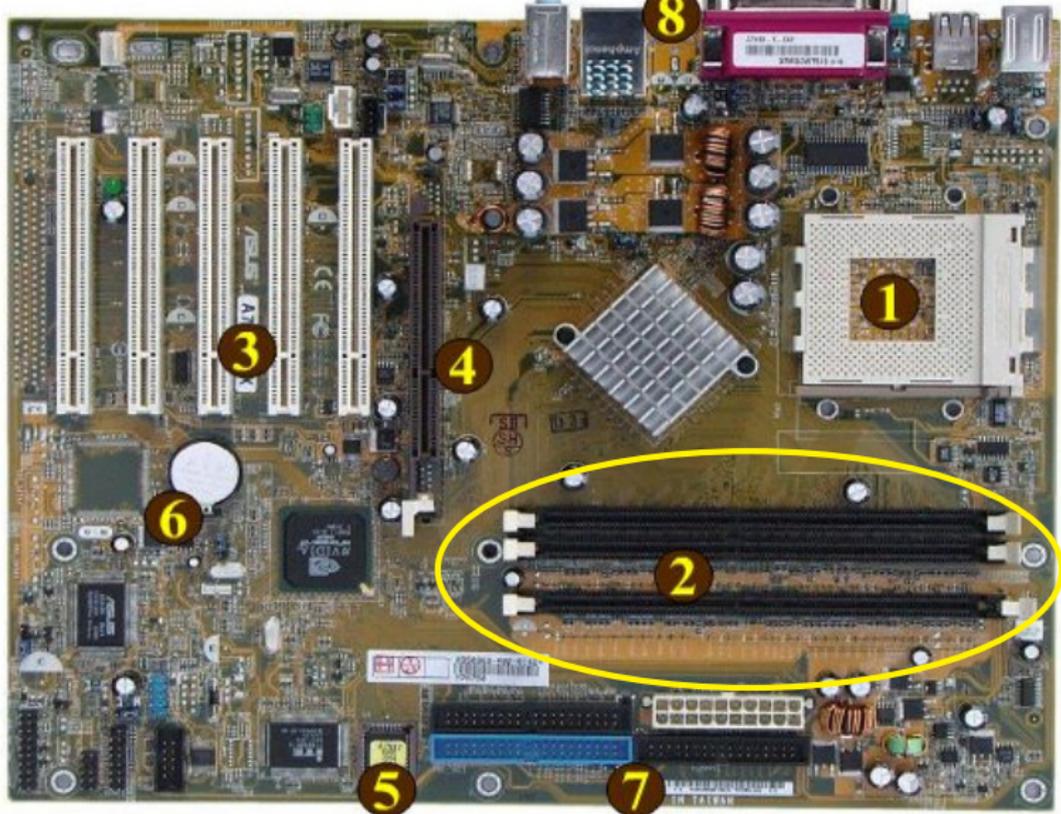




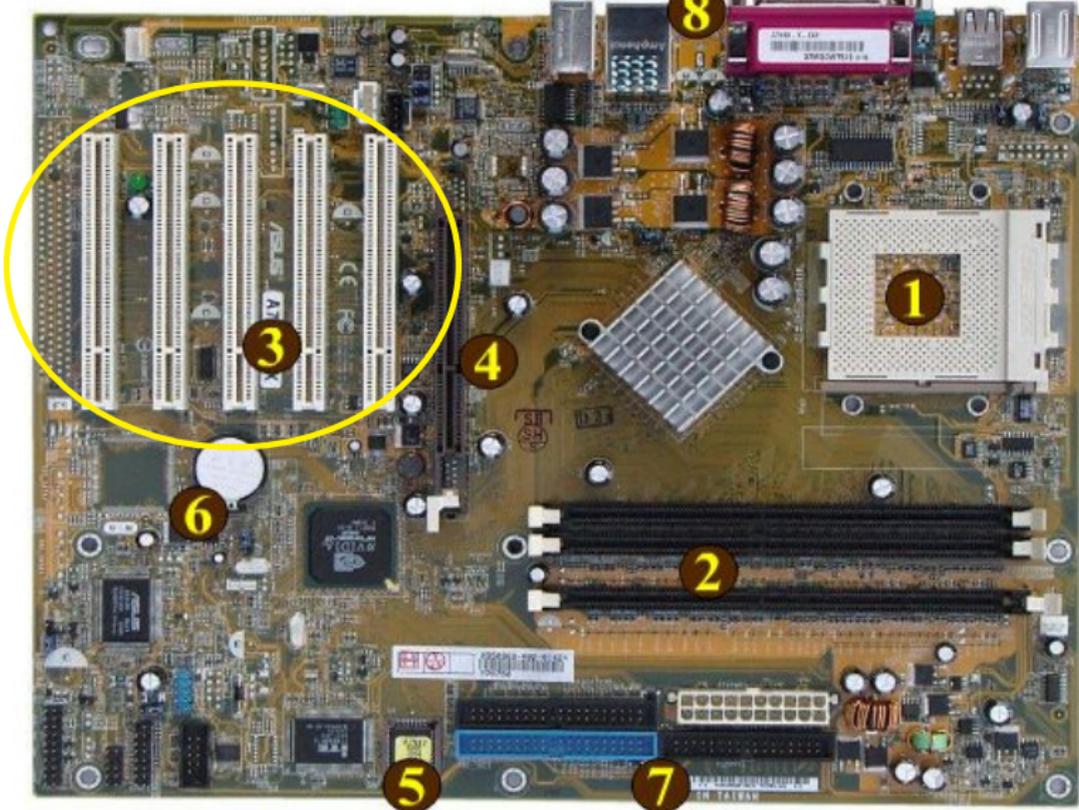
À l'intérieur du boîtier on peut trouver cette carte mère. Elle est le socle utilisé par tous les éléments de la machine pour communiquer.



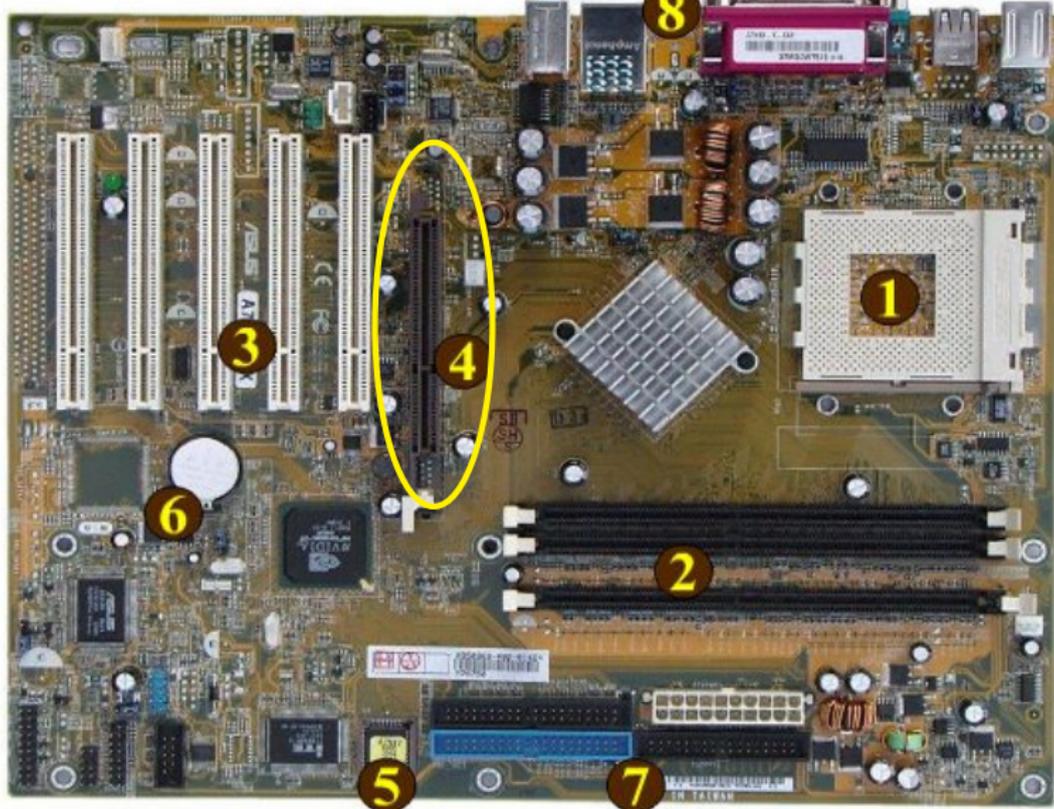
1 Ce gros carré blanc est le *socket* (socle) du processeur.



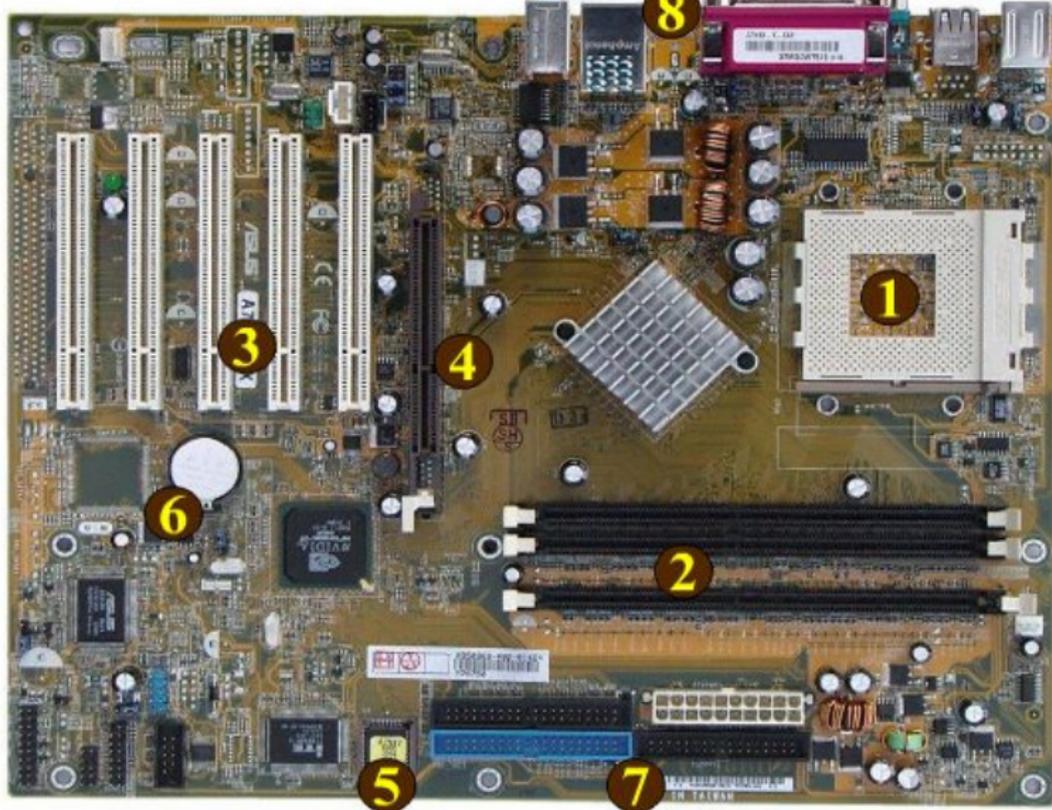
**2** Emplacements (slots) pour accueillir les barrettes de RAM.



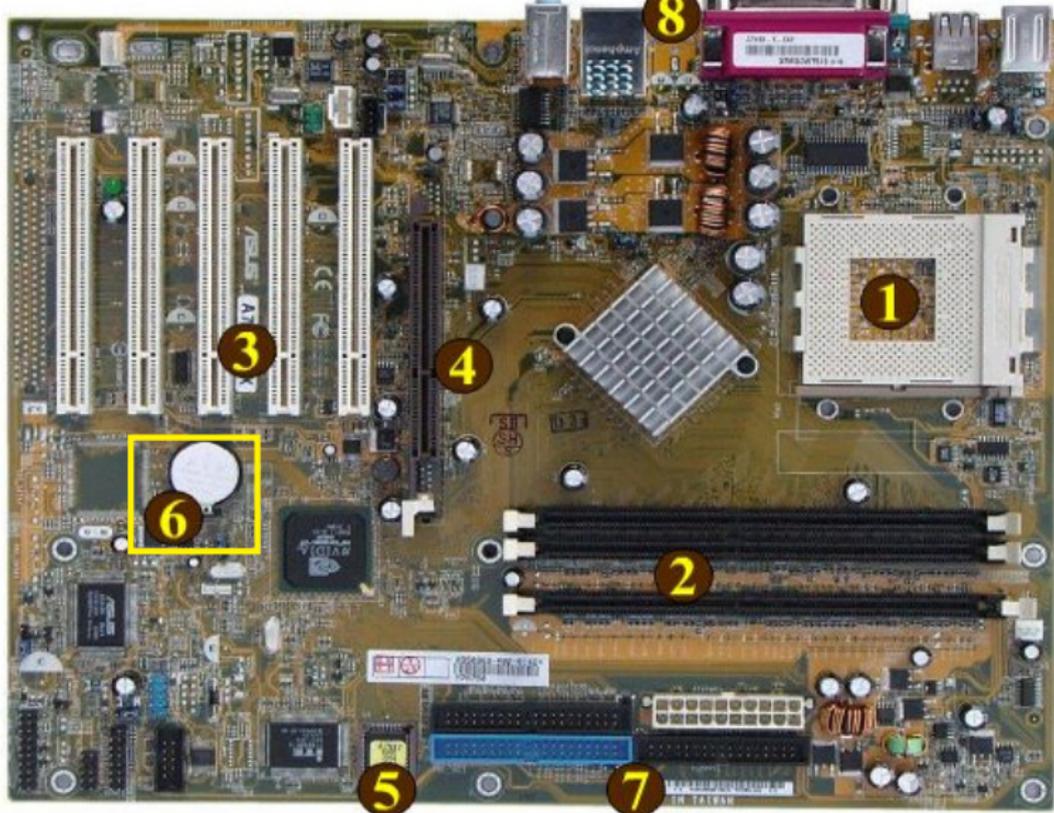
**3** Ces grandes barres blanches sont destinées à accueillir divers types de cartes d'extension (son, graphique), elles s'appellent ports PCI (Peripheral Component Interconnect).



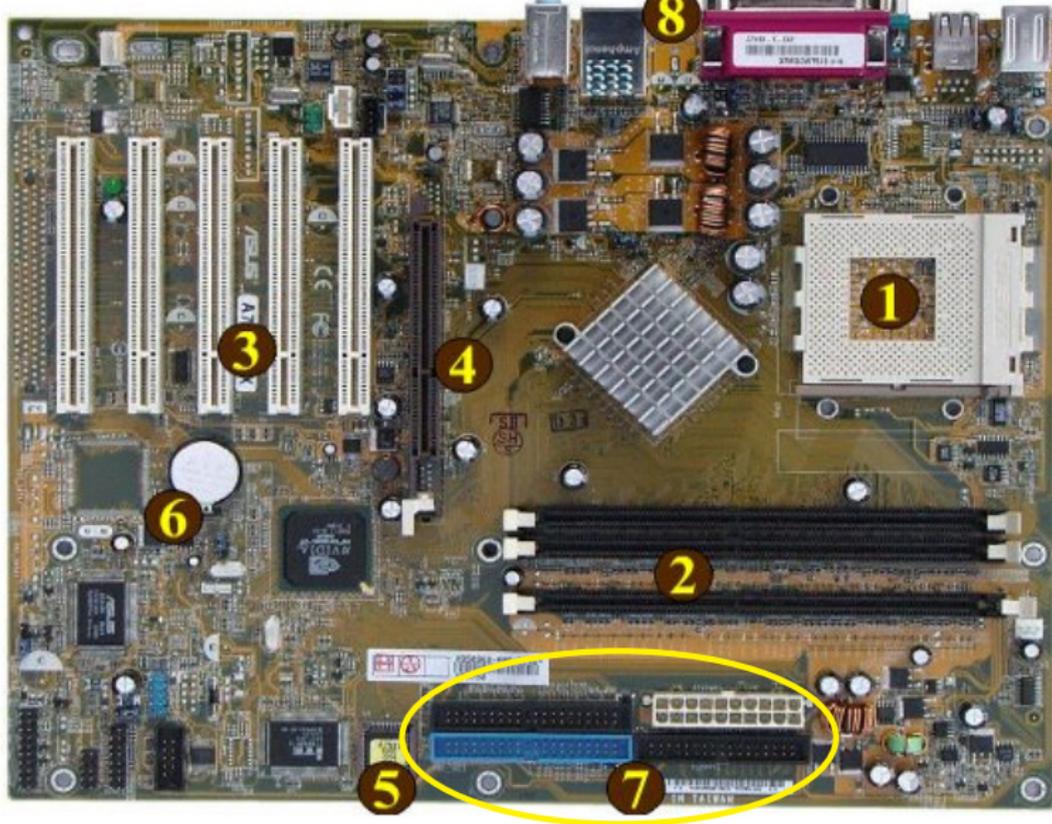
**4** Un port plus rapide pour la carte graphique (*Accelerated Graphics Port*). Les ordinateurs plus récents utilisent le plus rapide *PCI Express*.



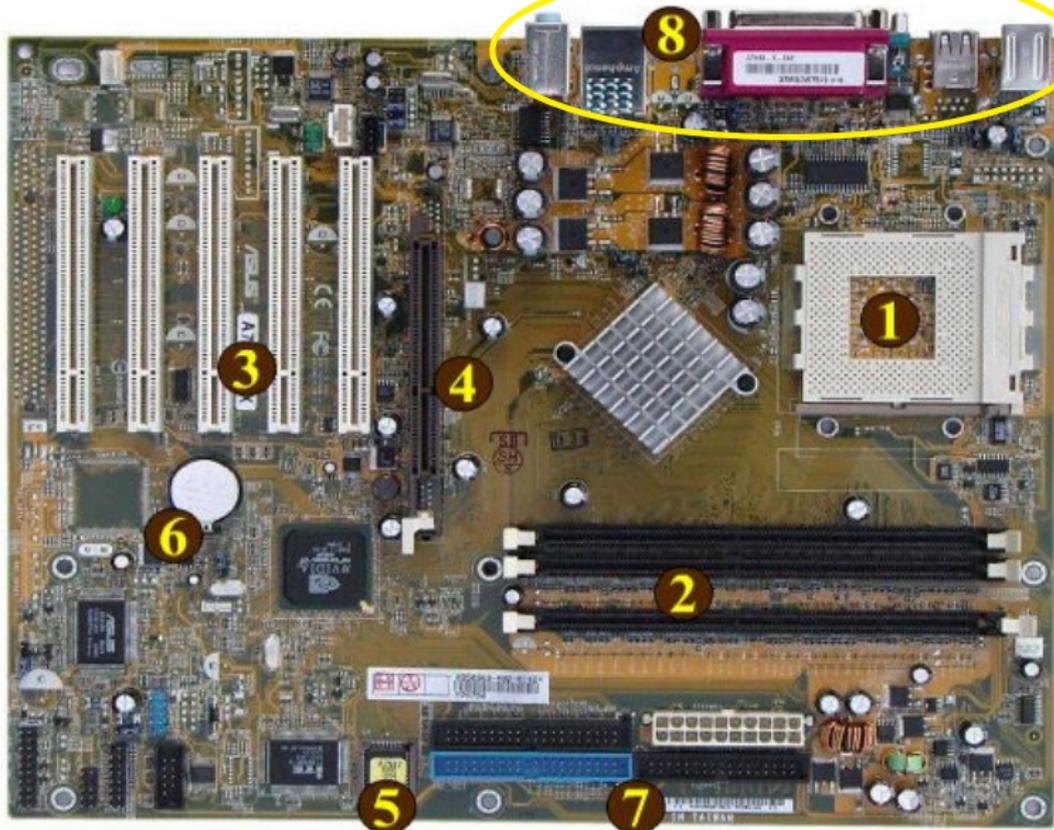
**5** Cette petite puce contient le BIOS (Basic Input/Output System), c.à.d., un petit programme qui s'exécute en premier lorsqu'on allume la machine et qui permet de la configurer.



**6** Cette pile sert à alimenter le BIOS ; c'est grâce à elle que l'ordinateur retient l'heure même lorsqu'il n'est pas allumé.



**7** Le connecteur blanc sert à l'alimentation électrique. Les autres trois sont des ports IDE pour connecter des disques durs ou des lecteurs CD/DVD.



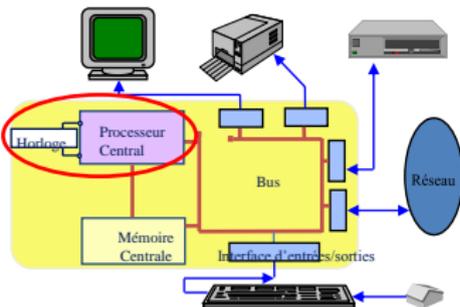
**8** Sur le côté de la carte mère on retrouve les connecteurs (clavier, souris, réseaux, USB).

1 Les premiers programmes/dessins en langage Processing

2 Comprendre l'ordinateur : notions d'architecture

- Les couches et les fonctions de base d'un ordinateur
- **Processeur, Mémoires, Bus, Périphériques**
- Du langage `processing` à la couche matérielle

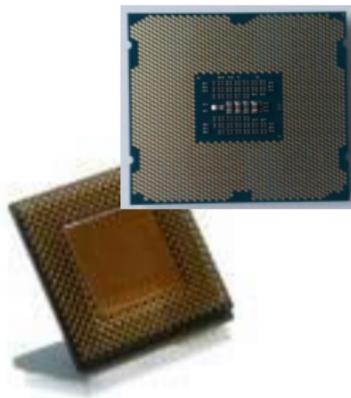
# Le processeur



Le **processeur** (CPU, pour *Central Processing Unit*) est le cerveau de l'ordinateur. Il permet de manipuler, des données et des instructions codées sous forme binaires.

Le **processeur** est un circuit électronique cadencé au rythme d'une horloge interne qui envoie des impulsions, appelées « **top** ». La **fréquence d'horloge**, correspond nombre d'impulsions par seconde. Elle s'exprime en Hertz (Hz).

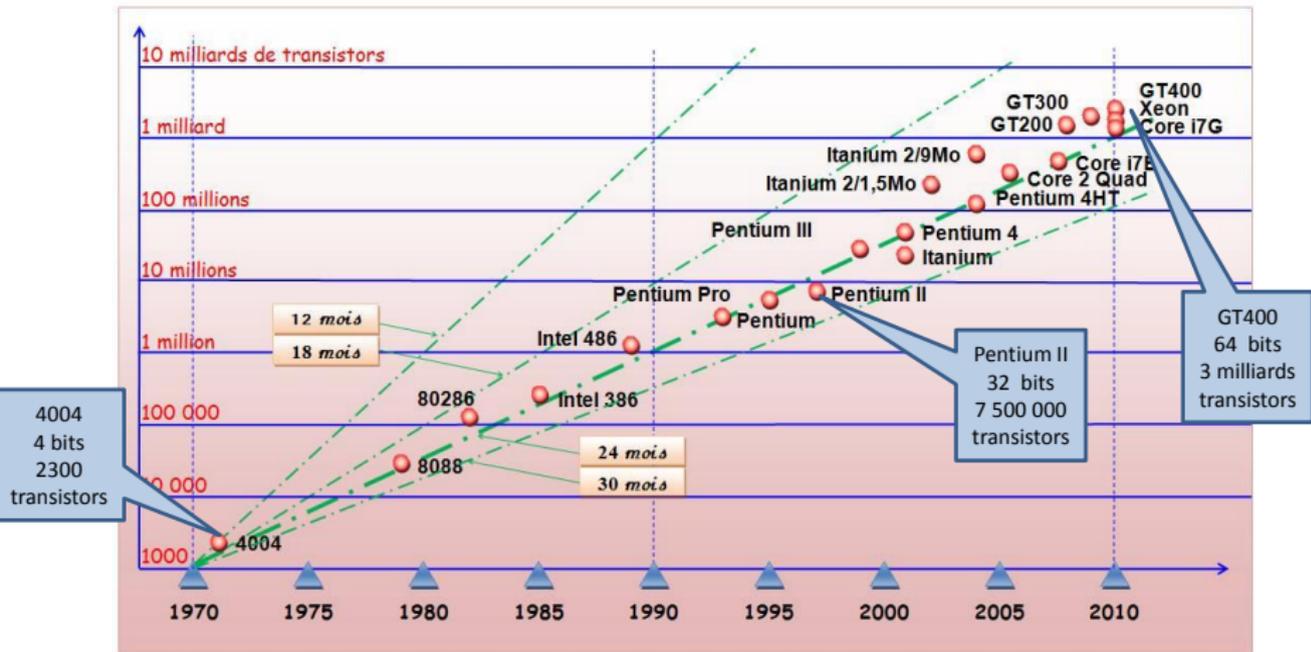
- Ordinateur à 2 GHz → l'horloge envoie 200 000 000 000 battements par seconde.



Circuits électroniques composés de millions de transistors placés dans un boîtier comportant des connecteurs d'entrée-sortie, surmonté d'un ventilateur.

→ **circuit intégré** ou **puce**

# Le processeur : performances



La **Loi de Moore** a été exprimée en 1965 par *Gordon Moore*, un des trois fondateurs d'Intel. Elle postule que le nombre de transistors sur une puce double tous les deux ans.

# Mémoires : bits, octets, méga-octets

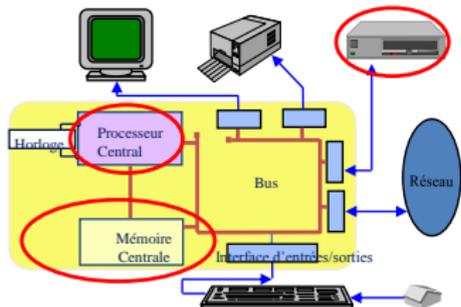
- La donnée de base manipulée par la machine est le **bit**. Il peut prendre deux valeurs : 0 et 1.
- Un octet (ou byte) est une chaîne de 8 bits. Un octet permet de coder des valeurs numériques ou jusqu'à 256 caractères différents.

1	0	1	0	0	0	1	1
Écriture de $2^7+2^5+2^1+2^0$ , c'est-à-dire 163 dans un octet.							

- Il y a deux usages pour quantifier les kilooctets (**Koctets**), mégaoctet (**MOctets**) et gigaoctets (**Goctets**)

<b>Koctets</b>	1000 octets	$2^{10} = 1024$ octets
<b>Moctets</b>	1.000.000 octets	$2^{20} = 1.048.576$ octets
<b>Goctets</b>	$10^9$ octets	$2^{30}$ octets

# Les mémoires de l'ordinateur



Une « **mémoire** » est un composant électronique capable de stocker temporairement des informations

- Une mémoire est caractérisée par :
  - Sa **capacité**, représentant le volume global d'informations (en bits) que la mémoire peut stocker (par exemple 1 Goctets, soit  $2^{30}$  octets, soit  $2^{30} * 8$  bits).
  - Son **temps d'accès**, correspondant à l'intervalle de temps entre la demande de lecture/écriture et la disponibilité de la donnée.
- L'ordinateur contient différents niveaux de mémoire, organisés selon une **hiérarchie mémoire**.

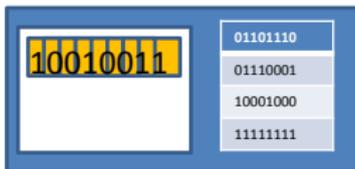
# Les mémoires de l'ordinateur

- L'ordinateur contient différents niveaux de mémoire, organisés selon une **hiérarchie mémoire**.

Mémoires vives : mémoires **volatiles** :



Mémoires de masse :  
mémoires **permanentes**



**REGISTRES**  
N bits (32, 64)  
1 nanoseconde

**Mémoires Caches**  
Koctets (L1,L2)  
5 nanosecondes

**Mémoires Centrales**  
Goctets  
10 nanosecondes

**Mémoires de masse**  
500 Goctets - Toctets  
5 millisecondes

Au plus près du cpu

Capacité, vitesse

Au plus loin du cpu

Mémoire volatile : le contenu de la mémoire n'existe que si il y a une alimentation électrique (typiquement les mémoires caches et mémoire centrale)  
Mémoire permanente, de masse : mémoire de grande capacité dont le contenu demeure même sans alimentation électrique (typiquement le disque dur)

# La vitesse (débit) des mémoires

## Décrypter “temps d'accès 10 nanosecondes”

- 10 nanosecondes =  $10 \frac{1}{10^9} = \frac{1}{10^8}$  secondes  $\implies 10^8$  accès par seconde (100MHz)
- système de 64 bits  $\implies$  un accès = 8 octets
- vitesse mémoire =  $10^8 \cdot 8 = 800$  Moctets par seconde (MBs)  
**Attention :** MBs (octets par sec.)  $\neq$  Mbs (bits par sec.)

Formule : temps accès x nanos  $\implies$  vitesse =  $8000/x$  MBs

- temps accès 30nanosec  $\implies$  vitesse = 266MBs
- temps accès 0.6nanosec  $\implies$  vitesse = 13GBs

```
sudo lshw -short -C memory
```

Observation : Un CPU à 1000MHz peut exécuter une instruction par nanosec. Mais si la RAM a un temps d'accès de 10 nanosec, le CPU est ralenti, sauf s'il utilise la cache.

# La vitesse (débit) des mémoires

## Décrypter “temps d'accès 10 nanosecondes”

- 10 nanosecondes =  $10 \frac{1}{10^9} = \frac{1}{10^8}$  secondes  $\implies 10^8$  accès par seconde (100MHz)
- système de 64 bits  $\implies$  un accès = 8 octets
- vitesse mémoire =  $10^8 \cdot 8 = 800$  Moctets par seconde (MBs)  
**Attention :** MBs (octets par sec.)  $\neq$  Mbs (bits par sec.)

Formule : temps accès x nanos  $\implies$  vitesse =  $\boxed{8000/x \text{ MBs}}$

- temps accès 30nanosec  $\implies$  vitesse = 266MBs
- temps accès 0.6nanosec  $\implies$  vitesse = 13GBs

```
sudo lshw -short -C memory
```

Observation : Un CPU à 1000MHz peut exécuter une instruction par nanosec. Mais si la RAM a un temps d'accès de 10 nanosec, le CPU est ralenti, sauf s'il utilise la cache.

# La vitesse (débit) des mémoires

## Décrypter “temps d'accès 10 nanosecondes”

- 10 nanosecondes =  $10 \frac{1}{10^9} = \frac{1}{10^8}$  secondes  $\implies$   $10^8$  accès par seconde (100MHz)
- système de 64 bits  $\implies$  un accès = 8 octets
- vitesse mémoire =  $10^8 \cdot 8 = 800$  Moctets par seconde (MBs)  
**Attention :** MBs (octets par sec.)  $\neq$  Mbs (bits par sec.)

Formule : temps accès x nanos  $\implies$  vitesse = 8000/x MBs

- temps accès 30nanosec  $\implies$  vitesse = 266MBs
- temps accès 0.6nanosec  $\implies$  vitesse = 13GBs

 Observation : Un CPU à 1000MHz peut exécuter une instruction par nanosec. Mais si la RAM a un temps d'accès de 10 nanosec, le CPU est ralenti, sauf s'il utilise la cache.

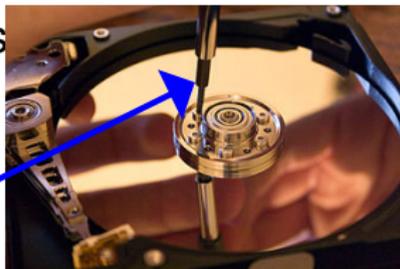
# Exemples de vitesses de mémoires/réseaux

- mémoires RAM années 1990  $\approx \pm 200\text{MBs}$
  - mémoires RAM années 2000  $\approx \pm 1500\text{MBs}$
  - mémoires RAM années 2010  $\approx \pm 10\text{GBs}$
- 
- disques durs années 2000  $\approx 25\text{MBs}$
  - disques durs années 2010  $\approx 150\text{MBs}$  – 600MBs
  - USB 2.0 : max 60MBs, souvent  $\approx 10\text{MBs}$
  - USB 3.0 : max 500MBs
- 
- Cable ethernet (RJ45) : Cat 5  $\approx 10\text{MBs}$ , Cat 6  $< 120\text{MBs}$
  - Wi-Fi  $\approx 1\text{-}3\text{MBs}$  en moyenne (802.11a ou 802.11g)
    - beaucoup plus avec de nouveaux standards (802.11ad)
    - doubler la distance  $\implies$  diviser le débit par 3 ou 4 approx.
  - ADSL  $\approx < 12\text{MBs}$  en moyenne, fibre  $\approx$  jusqu'à 120MBs
  - Youtube en 720p HD nécessite  $< 0.5\text{MBs}$

# Exemples de vitesses de mémoires/réseaux

- mémoires RAM années 1990  $\approx \pm 200\text{MBs}$
- mémoires RAM années 2000  $\approx \pm 1500\text{MBs}$
- mémoires RAM années 2010  $\approx \pm 10\text{GBs}$

- 
- disques durs années 2000  $\approx 25\text{MBs}$ 
    - il y a une tête de lecture mécanique



- disques durs années 2010  $\approx 150\text{MBs}$  – **600MBs**
  - SSD (*Solid State Drive*) sans aucun élément mécanique
- USB 2.0 : max 60MBs, souvent  $\approx 10\text{MBs}$
- USB 3.0 : max 500MBs

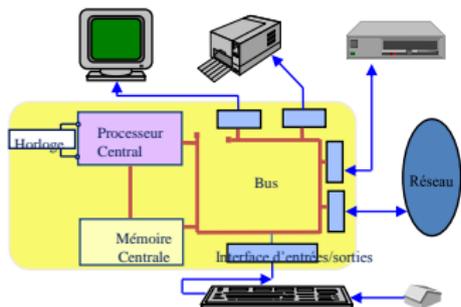
- 
- Cable ethernet (RJ45) : Cat 5  $\approx 10\text{MBs}$ , Cat 6  $< 120\text{MBs}$
  - Wi-Fi  $\approx 1\text{-}3\text{MBs}$  en moyenne (802.11a ou 802.11g)
    - beaucoup plus avec de nouveaux standards (802.11ad)
    - doubler la distance  $\implies$  diviser le débit par 3 ou 4 approx.

# Exemples de vitesses de mémoires/réseaux

- mémoires RAM années 1990  $\approx \pm 200\text{MBs}$
  - mémoires RAM années 2000  $\approx \pm 1500\text{MBs}$
  - mémoires RAM années 2010  $\approx \pm 10\text{GBs}$
- 
- disques durs années 2000  $\approx 25\text{MBs}$
  - disques durs années 2010  $\approx 150\text{MBs}$  – 600MBs
  - USB 2.0 : max 60MBs, souvent  $\approx 10\text{MBs}$
  - USB 3.0 : max 500MBs
- 
- Cable ethernet (RJ45) : Cat 5  $\approx 10\text{MBs}$ , Cat 6  $< 120\text{MBs}$
  - Wi-Fi  $\approx 1\text{-}3\text{MBs}$  en **moyenne** (802.11a ou 802.11g)
    - beaucoup plus avec de nouveaux standards (802.11ad)
    - doubler la distance  $\implies$  diviser le débit par 3 ou 4 approx.
  - ADSL  $\approx < 12\text{MBs}$  en moyenne, fibre  $\approx$  jusqu'à 120MBs
  - Youtube en 720p HD nécessite  $< 0.5\text{MBs}$

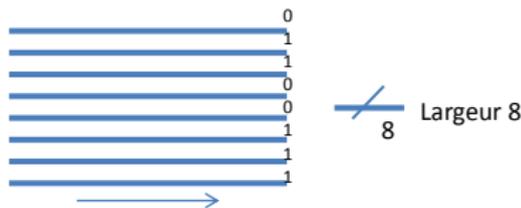
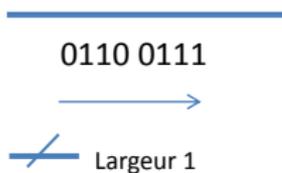


# Les bus de l'ordinateur



Un « **bus** » est un composant électronique permettant à différents composants de l'ordinateur de s'échanger des informations

- Bus système (*Front Side Bus FSB*) permet le communication entre le processeur et le mémoire centrale.
- Bus d'extension permet aux autres éléments de l'ordinateur de communiquer entre eux.
- Bus série, bus parallèle, largeur de bus



# Rappels composants de base du processeur

L'horloge fournit un signal régulier pour synchroniser tout le fonctionnement du CPU

Unité arithmétique et logique effectue les calculs sur des données écrites dans des registres et stocke le résultats dans un autre registre

- registres : petites mémoires internes très rapides, pouvant être accédées facilement. La taille des registres dépend de l'architecture (32 bits ou 64 bits).

Unité d'interface mémoire transfert les données à partir des mémoires vers les registres et vice-versa

Unité de comande décode les instructions, fait appel à l'unité arithmétique/logique ou à l'unité d'interface mémoire.

# Fonctionnement du processeur

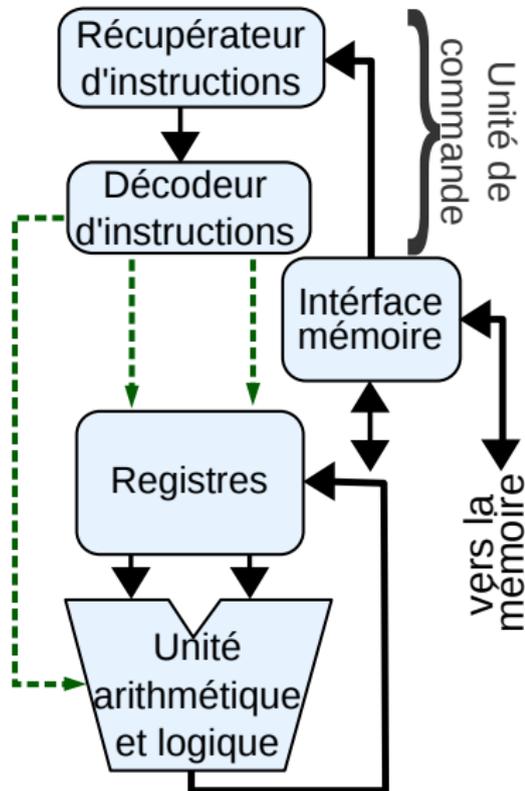
Plusieurs étapes :

1. L'unité de commande demande à l'interface mémoire de récupérer une instruction (en. : instruction fetch)

2. Le décodeur découpe l'instruction en plusieurs parties telles qu'elles puissent être gérés par l'unité arithmétique-logique et les registres (flèches vertes)

3. L'unité arithmétique et logique réalise l'instruction décodée, ex., l'addition de deux nombres stockés dans des registres

4. Écrire le résultat dans un registre et par la suite dans la mémoire, à l'aide de l'interface mémoire



# Fonctionnement du processeur

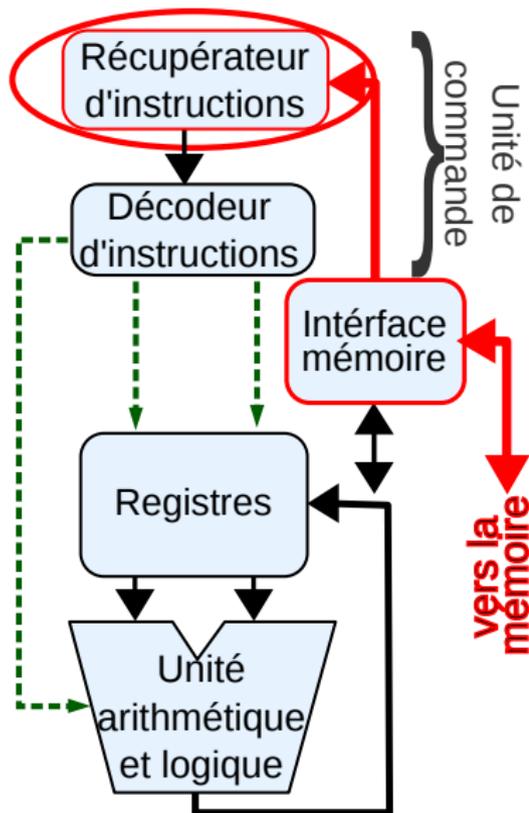
Plusieurs étapes :

1. **L'unité de commande** demande à l'interface mémoire de récupérer une instruction (en. : instruction fetch)

2. **Le décodeur** découpe l'instruction en plusieurs parties telles qu'elles puissent être gérés par l'unité arithmétique-logique et les registres (flèches vertes)

3. **L'unité arithmétique et logique** réalise l'instruction décodée, ex., l'addition de deux nombres stockés dans des registres

4. **Écrire le résultat dans un registre** et par la suite dans la mémoire, à l'aide de l'interface mémoire





# Fonctionnement du processeur

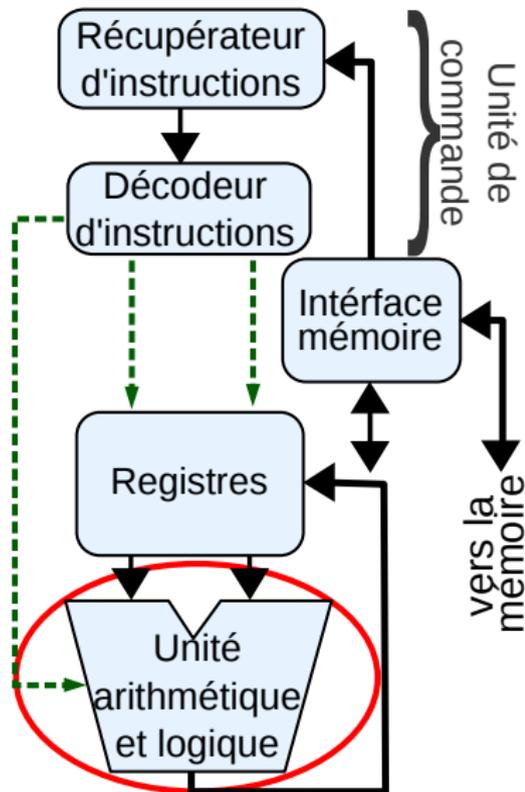
Plusieurs étapes :

1. **L'unité de commande** demande à l'interface mémoire de récupérer une instruction (en. : instruction fetch)

2. **Le décodeur** découpe l'instruction en plusieurs parties telles qu'elles puissent être gérés par l'unité arithmétique-logique et les registres (flèches vertes)

3. **L'unité arithmétique et logique** réalise l'instruction décodée, ex., l'addition de deux nombres stockés dans des registres

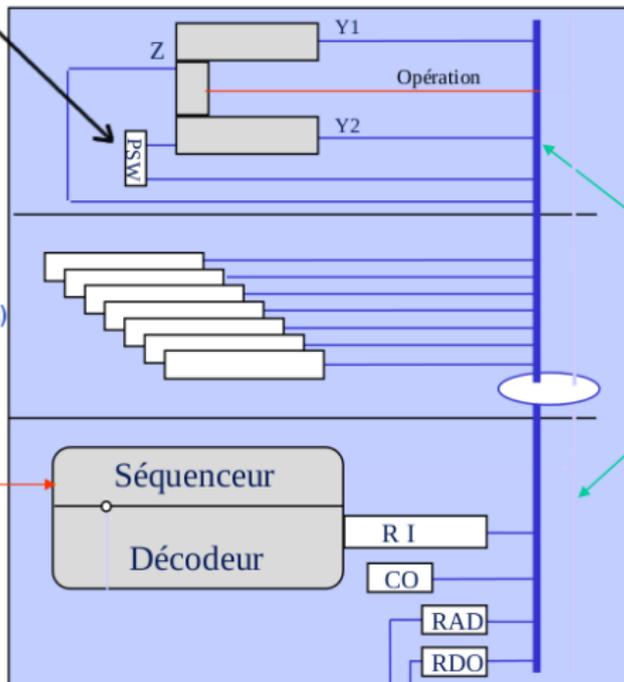
4. **Écrire le résultat dans un registre** et par la suite dans la mémoire, à l'aide de l'interface mémoire



# 1. Processeur (Unité Centrale)

Registre d'état (PSW pour Processor Status Word)

Unité Arithmétique et Logique



Bus Interne

Données

Commandes

horloge

Unité de Commande

Séquenceur

Décodeur

RI

CO

RAD

RDO

Commandes Lecture/Ecriture

Adresses

Données

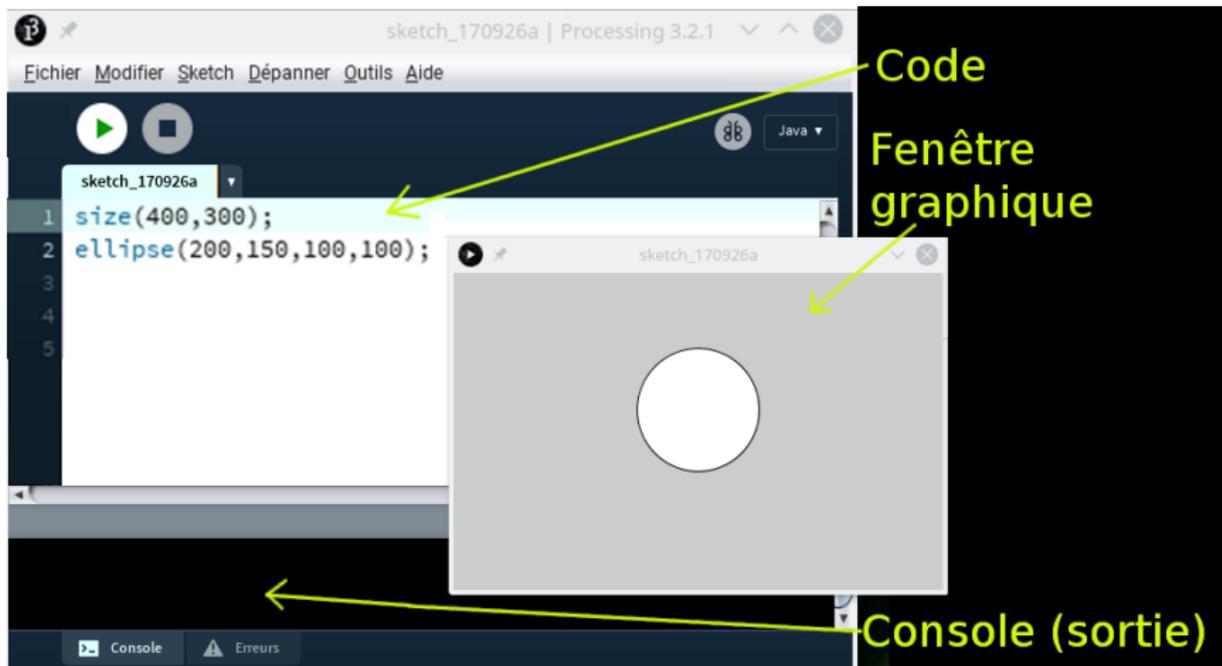
BUS Processeur /Mémoire

1 Les premiers programmes/dessins en langage Processing

2 Comprendre l'ordinateur : notions d'architecture

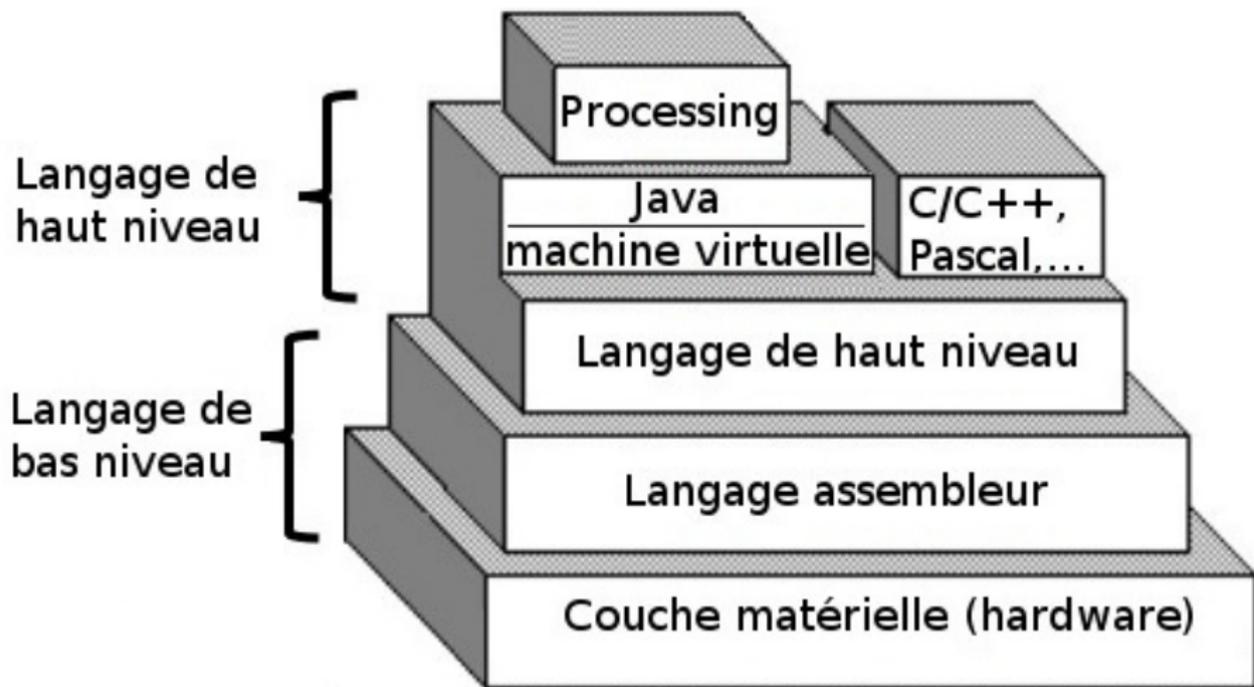
- Les couches et les fonctions de base d'un ordinateur
- Processeur, Mémoires, Bus, Périphériques
- Du langage `processing` à la couche matérielle

# Exemple de programme Processing



- Est-ce que l'exécution est la même sur toute machine ?
- Quelles opérations/calculs sont réellement exécutés par la machine ?
  - La fonction `ellipse(...)` cache des appels au système

# Langages de haut niveau et de bas niveau



Tout langage est traduit finalement en langage assembleur !  
Mais il peut y avoir une série de traductions : Processing → Java → Machine Virtuelle → Assembleur

# Le codage d'un problème ...

Programme Somme

1. a=5
2. b=7
3. PRINT (a+b)

Programme en langage de haut niveau

- instructions de haut niveau
- instructions indépendantes de machine



Compilateur

Système d'exploitation

*Gérer et partager le matériel*

Machine physique "matérielle"

processeur



Bus

```
01101110111110011
01111010001011100
10111101110111111
00111011110111011
00111111000111101
```

Mémoire centrale

Programme **en assembleur** à exécuter : instructions machine et valeurs **en binaire**

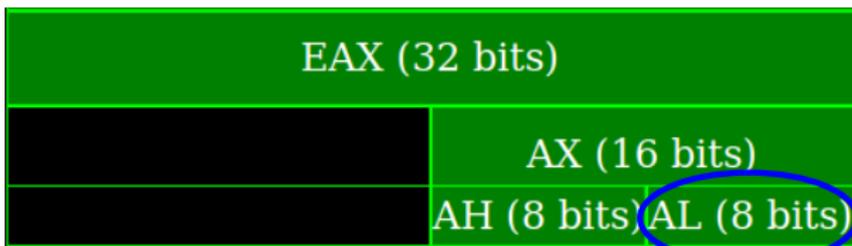
- L'assembleur est un langage qui permet d'écrire directement **des instructions machine pour le processeur**
- Voici le code assembleur qui effectue la somme de deux variables

$$z = x + y$$

```
mov AL, x ;copier x dans le registre AL
mov BL, y ;copier y dans le registre BL
add AL, BL ;faire la somme:instruction AL=AL+BL

mov z, AL ;copier registre AL -> variable z
```

Rappels organisation  
registres :



```
0x00e00000 55 8b ec 81 ec a8 01 00 00 53 56 57 eb 02 eb 05
0x00e00010 e8 f9 ff ff ff 5b 83 eb 15 89 9d 60 ff ff ff 89
0x00e00020 bd 58 fe ff ff .....
```

```
0x00e00000 55          PUSH EAX
0x00e00001 8bec        MOV EAX, EBX
0x00e00003 81eca8010000  ADD EBX, 0x1a8
0x00e00009 53          PUSH EBX
0x00e0000a 56          PUSH EAX
0x00e0000b 57          PUSH ESI
0x00e0000c eb02        JMP 0xe00010
0x00e0000e eb05        JMP 0xe00015
0x00e00010 e8f9ffffff  CALL 0xe0000e
0x00e00015 5b          POP EBX
0x00e00016 83eb15     SUB EBX, 0x15
0x00e00019 899d60ffff MOV [EAX-0xa0], EBX
0x00e0001f 89bd58fefff .....
```

## Instructions Classiques Assembleur

**PUSH** : ajouter au sommet de la pile

**POP** : récupérer le sommet de la pile

**JMP** : sauter à une autre ligne

**MOV** : copier un registre  
(ou case mémoire)  
dans un autre