

Cours Langage – C - Algorithmique - Introduction -

CIPA3 Nantes

Cours de **Nils Beaussé et Bilel BENZIANE**
nils.beausse@isen-ouest.yncrea.fr
bilel.benziane@isen-ouest.yncrea.fr

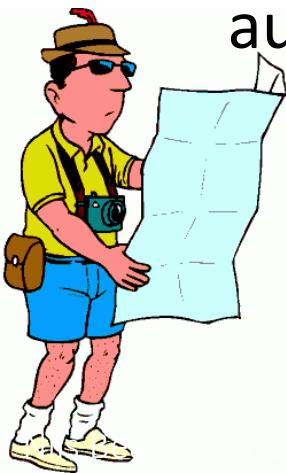
D'après les notes de cours et le
cours de Benoit Lardeux, Jean-
Benoît Pierrot, Pierre-Jean
Bouvet et Leandro Montero

Déroulement et Evaluation

- Contenu :
 - Le cours est séparé entre **partie théorique (cours et TD) et pratique (TP)**.
 - Le cours vise à homogénéiser la classe dans le domaine (car vos origines sont très différentes), de fait, les personnes qui ont déjà des connaissances dans le domaine risque de s'ennuyer durant la partie cours.
 - Il faudra faire avec ! **N'hésitez pas à aider vos camarades en difficulté !**
 - On essaiera de proposer des TPs avancés pour ceux qui connaissent déjà le sujet.
- Évaluation :
 - Épreuve théorique de 2h, une au milieu du module, une à la fin.
 - En monôme
 - Questions ouvertes (Programmation, debug de codes, ...)
 - Possibilité de QCM aléatoire en début de cours.

Qu'est-ce que l'algorithmique ?

- Avez-vous déjà ouvert un livre de recettes ?
Avez-vous déjà lu un mode d'emploi ?
 - Si oui, sans le savoir vous avez déjà **exécuté** un **algorithme** ...
- Avez-vous déjà indiqué le chemin à un touriste ? Avez-vous déjà demandé à quelqu'un de faire quelque chose pour vous au téléphone ?
 - Si oui, sans le savoir, vous avez déjà **fabriqué** et **fait exécuter** un **algorithme** ...



LASAGNES

INGRÉDIANTS

- 800GR DE VIANDE HACHÉE
- 2 OIGNONS
- 3 BRICK DE PURÉE DE TOMATE
- 2 BRICK DE BÉCHAMEL
- 1 PAQUET DE PÂTE À LASAGNES
- AIL
- FROMAGE RAPÉ
- SEL, POIVRE, HUILE D'OLIVE

RECETTE

ÉPLUCHER L'OIGNON ET L'AIL ET LES COUPER EN PETITS MORCEAUX.
LES FAIRE REVENIR DANS L'HUILE D'OLIVE ET AJOUTER, ENSUITE LA VIANDE HACHÉE.
FAIRE CHAUFFER À PART LA PURÉE DE TOMATES MÉLANGÉ À LA BÉCHAMEL.

PRENDRE UN PLAT À LASAGNES ET METTRE UN PEU DE SAUCE DANS LE FOND DU PLAT.
METTRE DESSUS LES PÂTES À LASAGNES.
MÉLANGER LA VIANDE HACHÉE AVEC LA SAUCE ET EN METTRE UNE COUCHE SUR LES PÂTES.

ET METTRE DU FROMAGE RAPÉ À CONVENANCE.
REMETTRE ENSUITÉ UNE COUCHE DE PÂTES, PUIS DE SAUCE AVEC LA VIANDE, PUIS DE FROMAGE.
REPETER L'OPÉRATION JUSQU'EN HAUT DU PLAT.
COUVRIR DE PAPIER D'ALU ET METTRE AU FOUR À 180 DEGRÉS.
LORSQUE QUE VOUS POUVEZ ENFONCER UN COUTEAU FACILEMENT C'CUIT.
IL RESTE PLUS QUÀ GRATINER ET SERVIR.

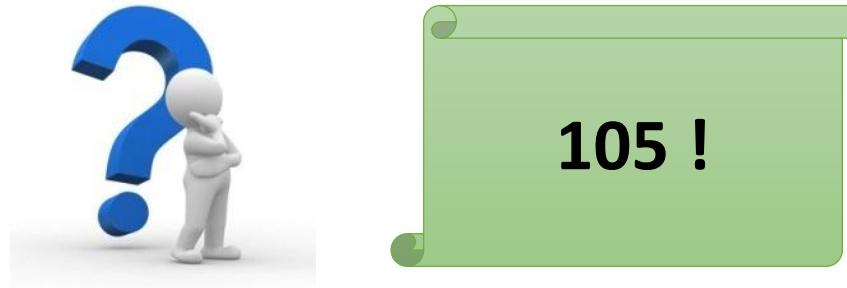
Un exemple d'algorithme

- Combien font 15×7 ?



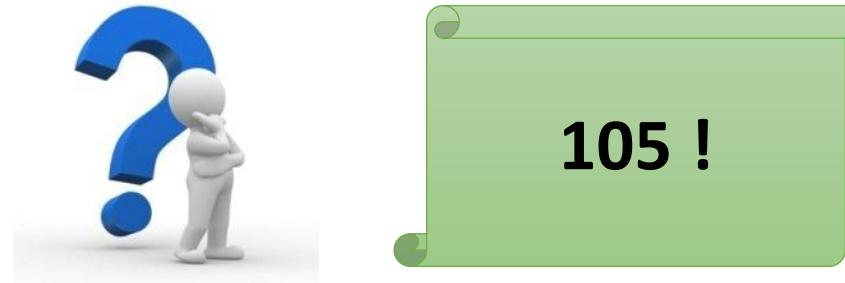
Un exemple d'algorithme

- Combien font 15×7 ?



Un exemple d'algorithme

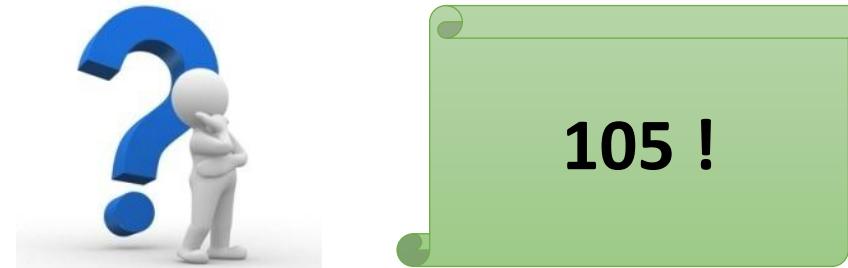
- Combien font 15×7 ?



- Comment avez-vous obtenu le résultat ?

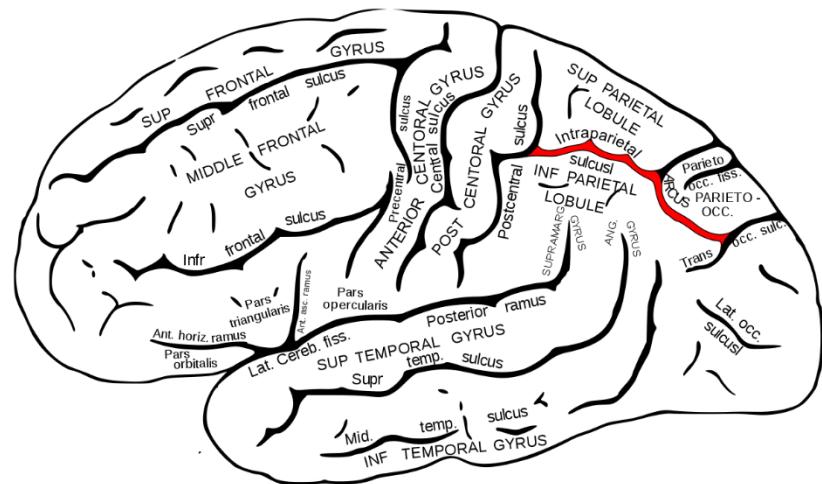
Un exemple d'algorithme

- Combien font 15×7 ?



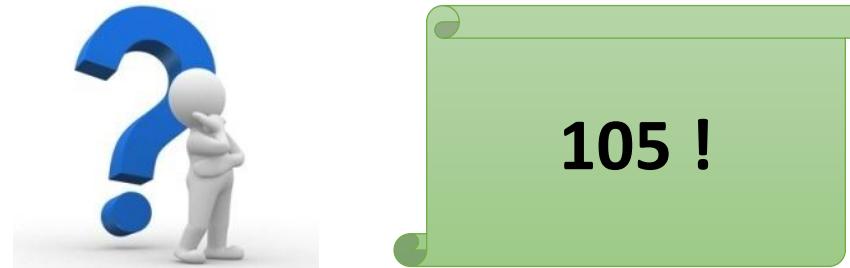
105 !

- Comment avez-vous obtenu le résultat ?

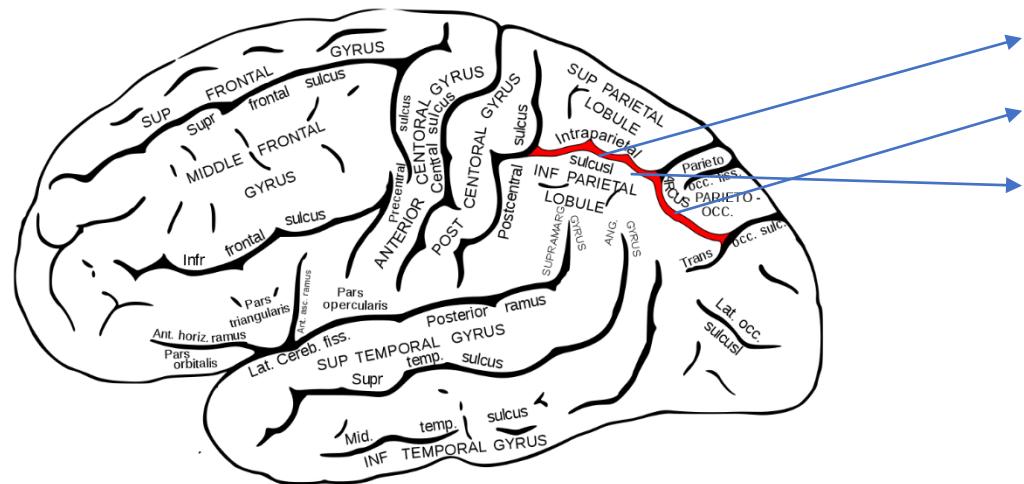


Un exemple d'algorithme

- Combien font 15×7 ?

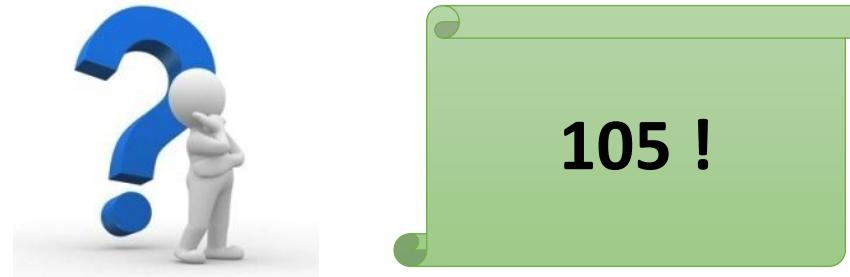


- Comment avez-vous obtenu le résultat ?

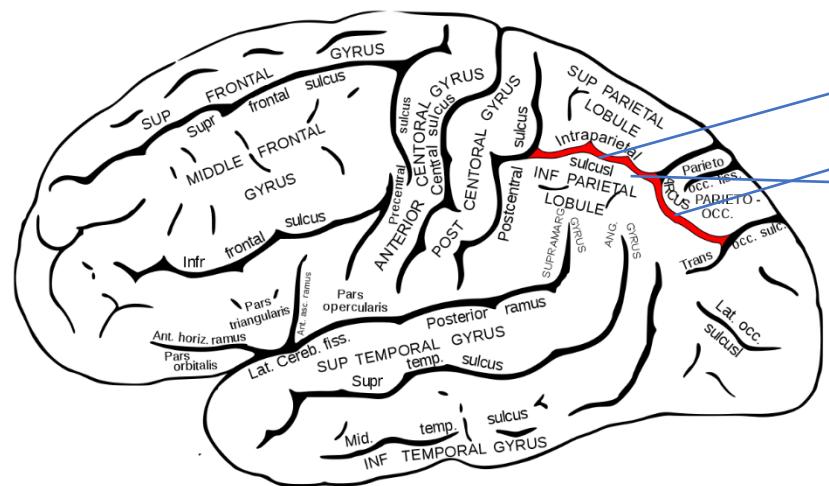


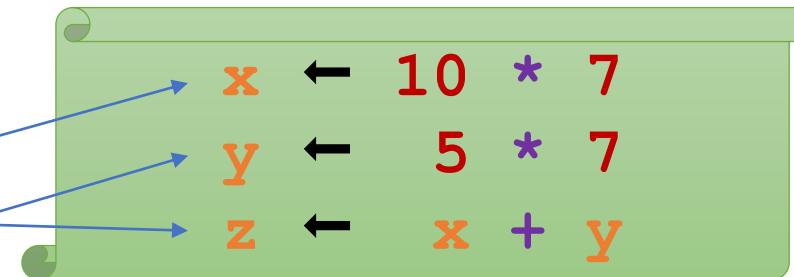
Un exemple d'algorithme

- Combien font 15×7 ?



- Comment avez-vous obtenu le résultat ?




$$\begin{aligned} x &\leftarrow 10 * 7 \\ y &\leftarrow 5 * 7 \\ z &\leftarrow x + y \end{aligned}$$

Pourquoi un algorithme se doit d'être clair ?

- Un **algorithme** doit contenir uniquement des **instructions compréhensibles** par **celui qui devra l'exécuter**
 - Ex : un mode d'emploi peut être clair pour certains et incompréhensible pour d'autres
- En **informatique**, on utilise seulement **4 familles d'instructions**
 - L'affectation de variables
 - La lecture / écriture
 - Les tests
 - Les boucles



Algorithmique pour quoi faire?

Résolution de problèmes par un ordinateur /microprocesseur :

Algorithmique pour quoi faire?

Résolution de problèmes par un ordinateur /microprocesseur :

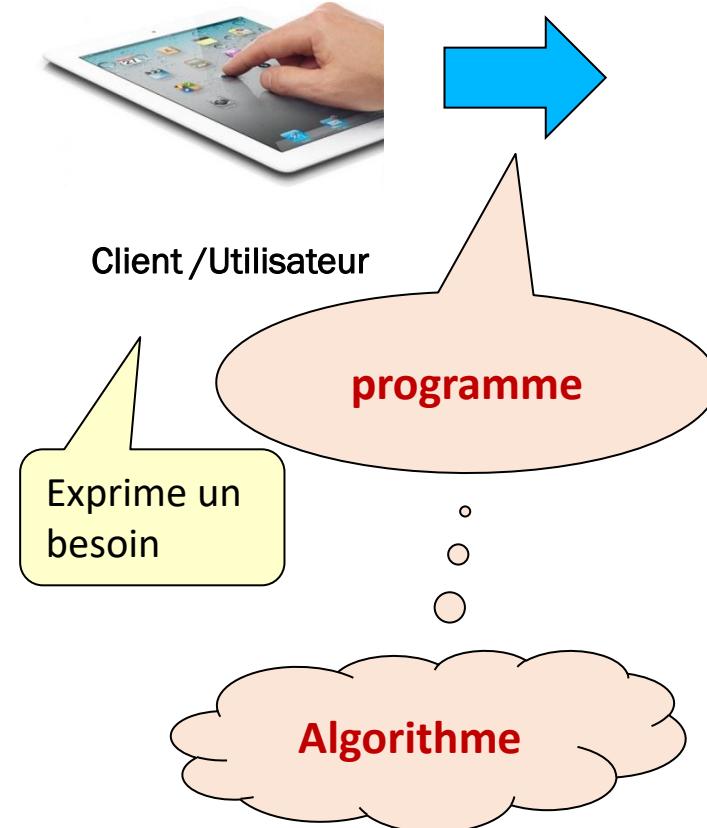


Client /Utilisateur



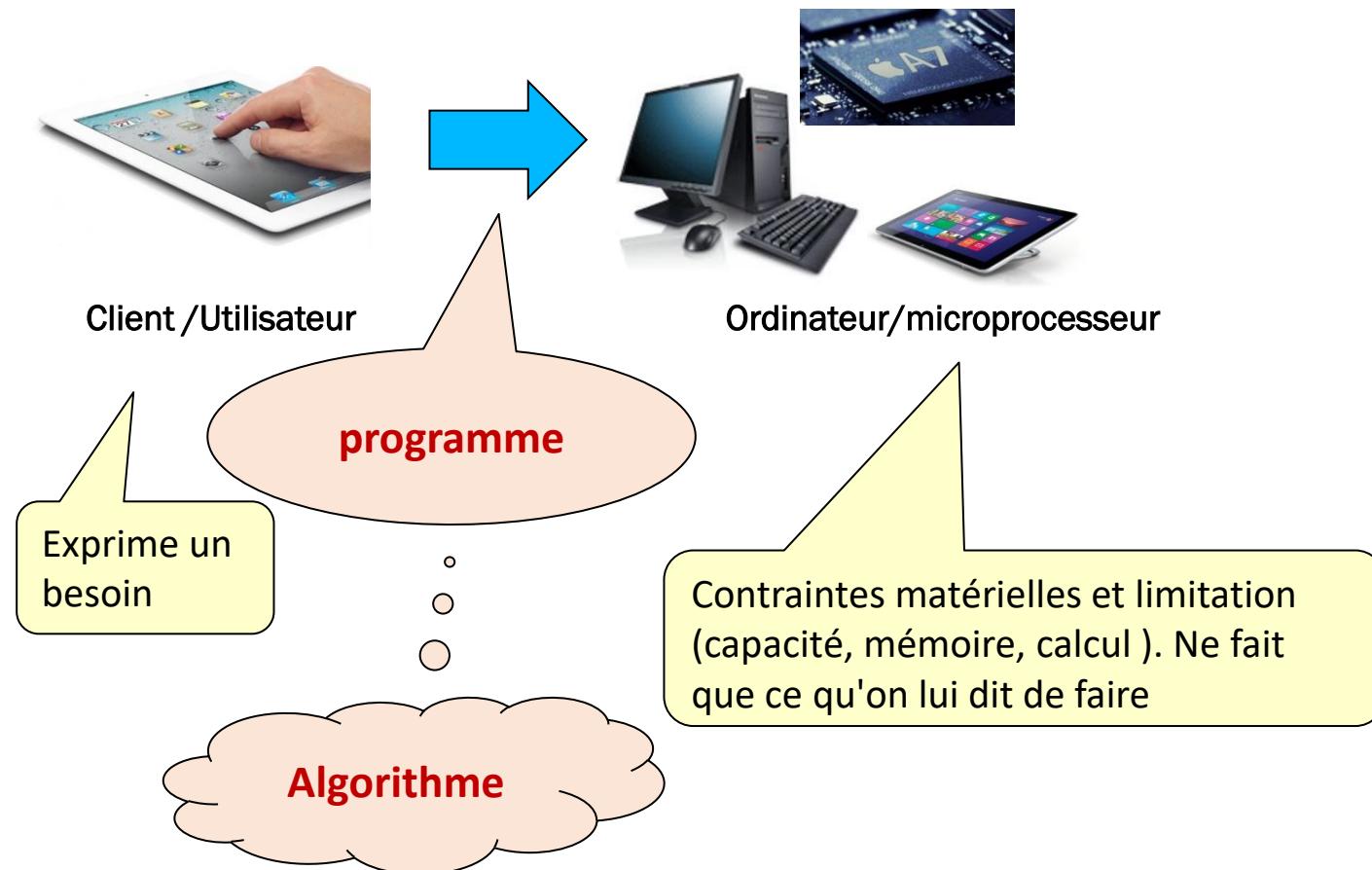
Algorithmique pour quoi faire?

R  solution de probl  mes par un **ordinateur /microprocesseur** :



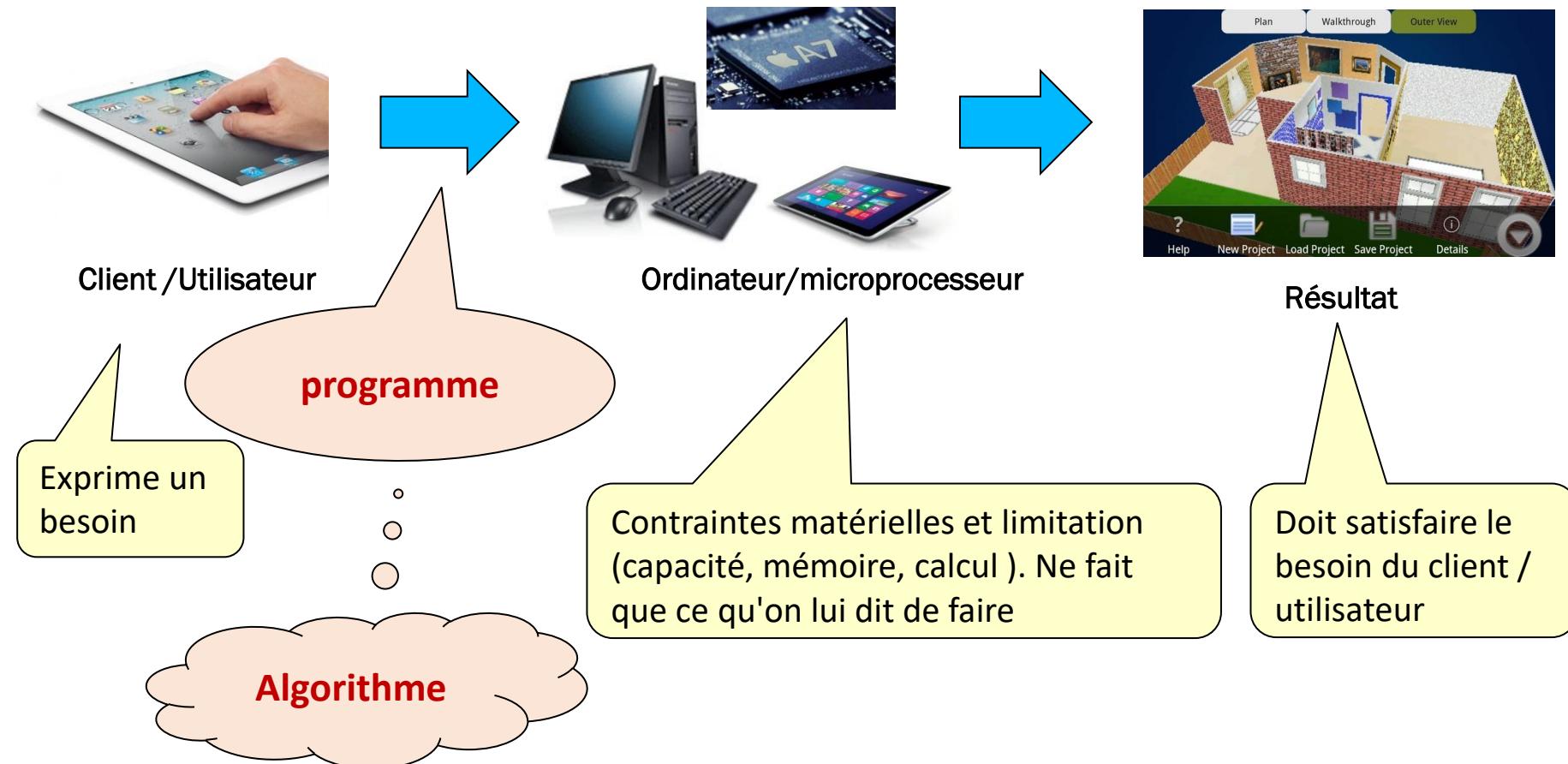
Algorithmique pour quoi faire?

Résolution de problèmes par un **ordinateur /microprocesseur** :



Algorithmique pour quoi faire?

Résolution de problèmes par un **ordinateur / microprocesseur** :



Des algorithmes partout ...



Quelles qualités pour faire de l'algorithmique ?

Quelles qualités pour faire de l'algorithmique ?

- **Intuition**

- Aucune recette ne permet de savoir a priori quelles instructions permettront d'obtenir le résultat voulu.
- Nécessite un certain nombre de **reflexes** qui s'acquièrent avec de **l'expérience**
- Le **raisonnement** au début laborieux va devenir **spontané** !

Quelles qualités pour faire de l'algorithmique ?

- **Intuition**

- Aucune recette ne permet de savoir a priori quelles instructions permettront d'obtenir le résultat voulu.
- Nécessite un certain nombre de **reflexes** qui s'acquièrent avec de **l'expérience**
- Le **raisonnement** au début laborieux va devenir **spontané** !

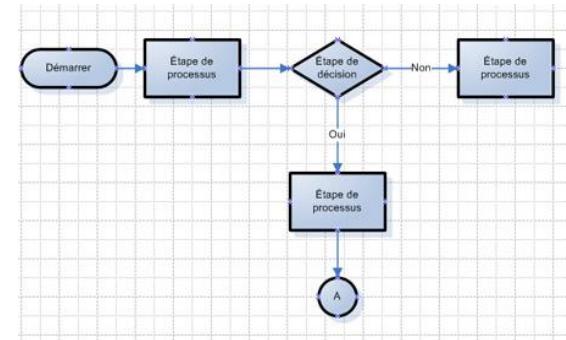
- **Rigueur et méthode**

- Il faut sans cesse **se mettre à la place de la machine**
- La **syntaxe** doit être obligatoirement respectée
- Nécessité de **vérifier méthodiquement** chaque suite d'instruction
- Apprendre à **chercher** et **corriger** des **erreurs** dans un programme

Objectifs (1/2)

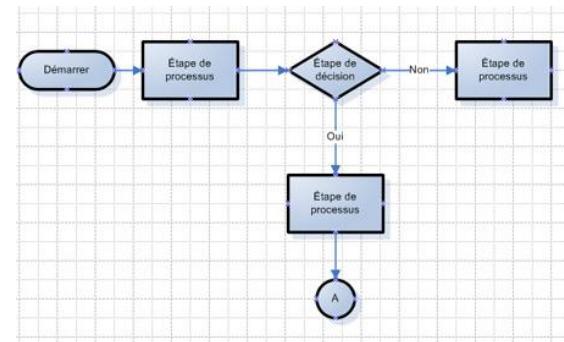
Objectifs (1/2)

- Présenter les **notions générales d'algorithme**
 - Variables, pointeurs, structures de données
 - Structure de contrôle d'exécution
 - Arbres
 - Sous-programmes
 - Gestion mémoire



Objectifs (1/2)

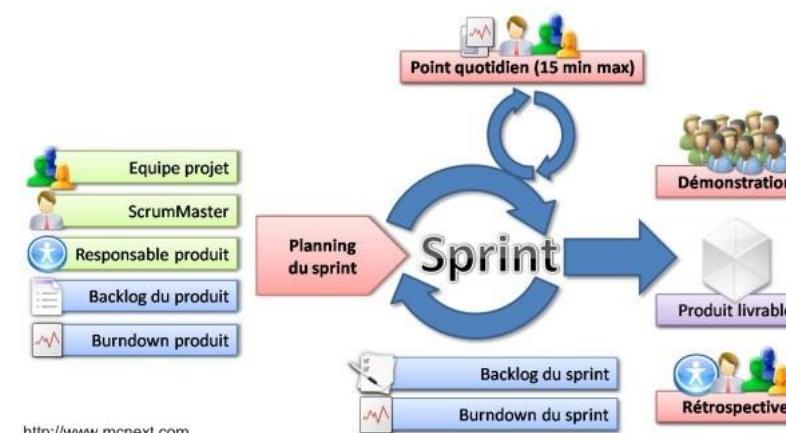
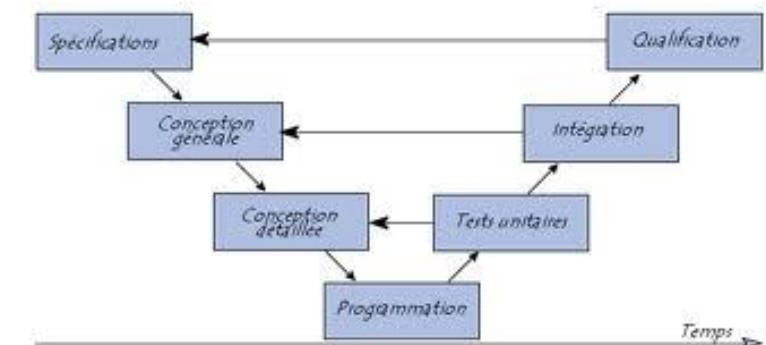
- Présenter les **notions générales d'algorithme**
 - Variables, pointeurs, structures de données
 - Structure de contrôle d'exécution
 - Arbres
 - Sous-programmes
 - Gestion mémoire
- Acquérir une **expérience pratique de la programmation**
 - Pseudo-langage - Langage C
 - Compilation
 - Notion de complexité - Applications aux tris
 - Preuve de programme



```
1 int fonct (float s) {  
2     s = 5.0;  
3     return 5;  
4 }  
  
5 void fons (int & i) {  
6     i = 10;  
7 }  
  
8 float * fon (float * r) {  
9     * r = 6.0;  
10    return r;  
11 }
```

Objectifs (2/2)

- Adopter une **méthodologie de programmation**
 - Spécification / Analyse
 - Conception
 - Réalisation
 - Documentation
 - Tests / Recette
 - Livraison



Plan du cours

- Introduction
 - Eléments de base de l’algorithmique / Généralités
 - Algorithmique et informatique : Du matériel au langage de programmation
- Langage C
 - Bases du langages C, TD et TPs



Éléments de Base de l’algorithmique et du C

Cours de **Nils Beaussé** et **Bilel BENZIANE**
nils.beausse@isen-ouest.yncrea.fr
bilel.benziane@isen-ouest.yncrea.fr

D’après les notes de cours et le
cours de Benoit Lardeux, Jean-
Benoît Pierrot, Pierre-Jean
Bouvet et Leandro Montero

Introduction



Un peu d'histoire



- Le mot algorithme vient du nom du mathématicien perse du IXème siècle av. JC **Abu Abdullah Muhammad ibn Musa al-Khwarizmi**
- Euclide (285 av. JC), Archimède (212 av. JC), Erastosthènes (276 av. JC), Averroès (1198) ont utilisé ce concept pour leurs propres travaux
- Le nom Khwarizmi apparaissait sous sa forme latinisée comme **Algorismus**, qui fut bientôt prise pour synonyme de «calcul décimal pour les opérations fondamentales» (XIIème siècle)
- Depuis le milieu du XXème siècle environ, il désigne non seulement une recette de calcul, mais toute **méthode séparable en opérations isolées** et pouvant en principe être effectuée par une machine.
- Le terme d'algorithme recouvre aujourd'hui un programme informatique écrit sans lien avec un logiciel ou un langage informatique particuliers. Le concept a été formalisé en 1936 avec les machines d'Alan Turing.

Qu'est-ce qu'un Algorithme?

Un **algorithme** est un processus systématique de résolution, par le calcul, d'un problème permettant de **présenter les étapes vers le résultat** à une **autre personne physique** (un autre humain) ou **virtuelle** (un calculateur). En d'autres termes, un algorithme est un énoncé d'une **suite d'opérations** permettant de **donner la réponse à un problème**.

Wikipedia

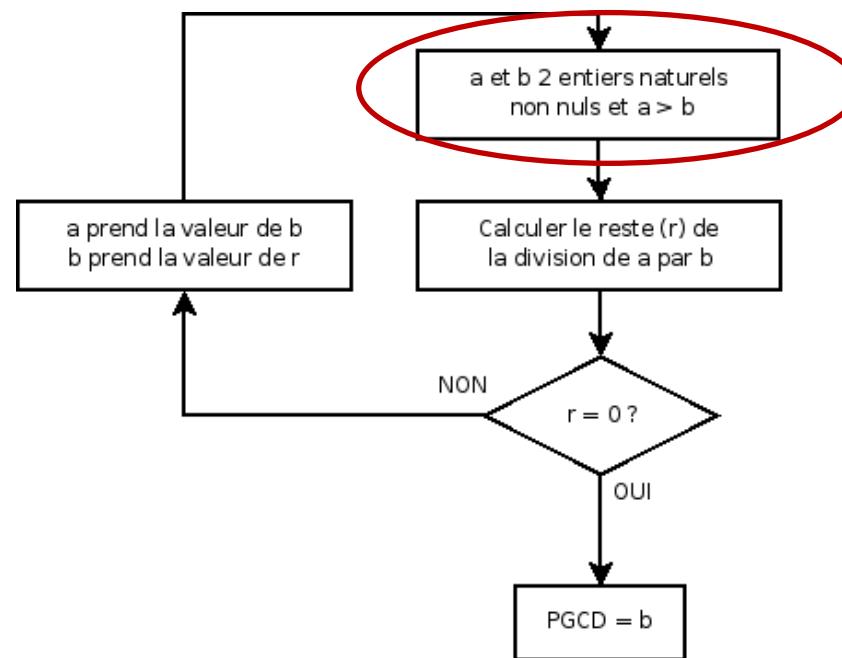
Les classes d'Algorithmes

- **Algorithmes séquentiels**
 - Les opérations s'exécutent en séquence : c'est la base sur un ordinateur
- Algorithmes Parallèles
 - Les opérations s'exécutent en parallèle : cela nécessite un matériel le permettant
- Algorithmes Répartis ou Distribués
 - Les opérations s'exécutent sur un réseau de matériel

Vu dans
d'autres
cours,
spéciali-
tés etc.

Un algorithme célèbre...

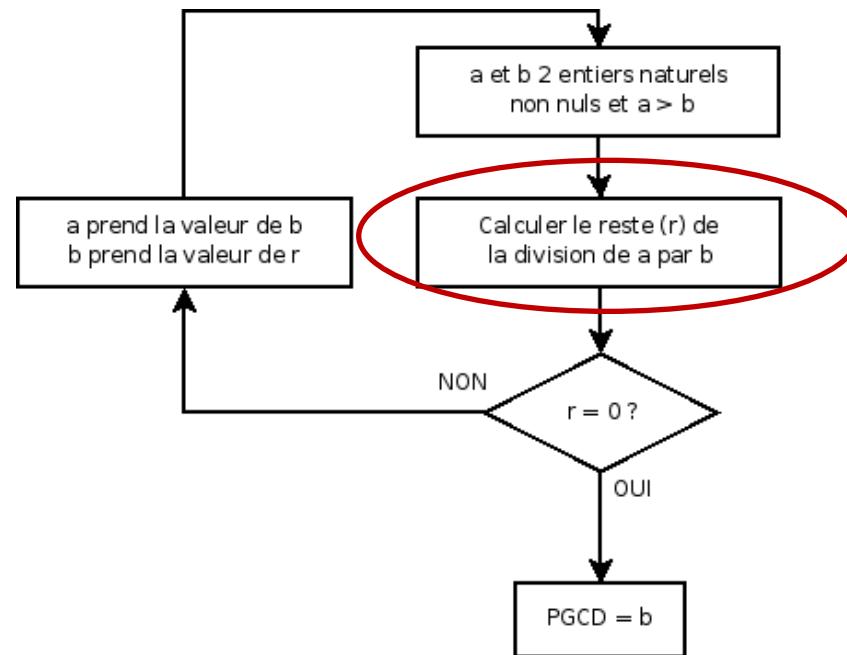
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|---|
| 1 | 186 | 54 | |
| 2 | | | |
| 3 | | | |
| 4 | | | |

Un algorithme célèbre...

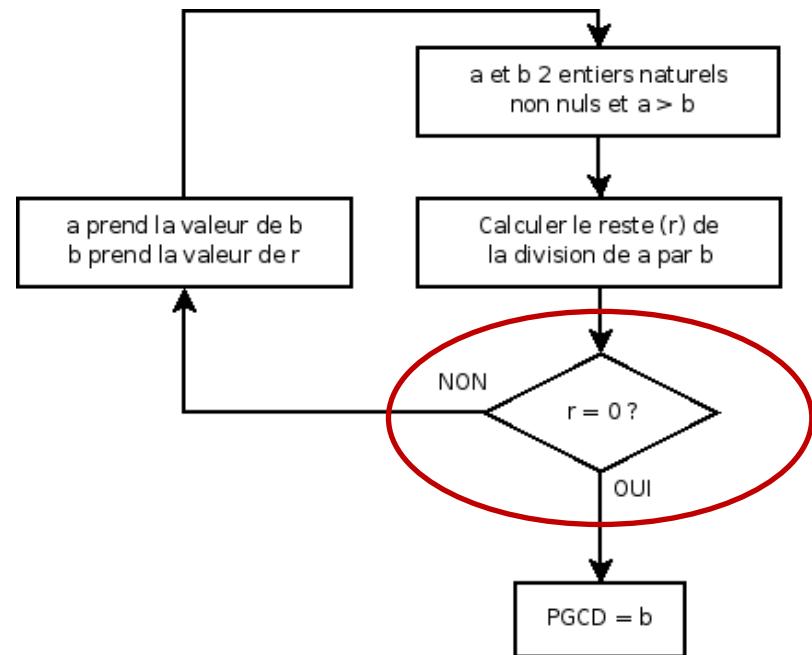
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | | | |
| 3 | | | |
| 4 | | | |

Un algorithme célèbre...

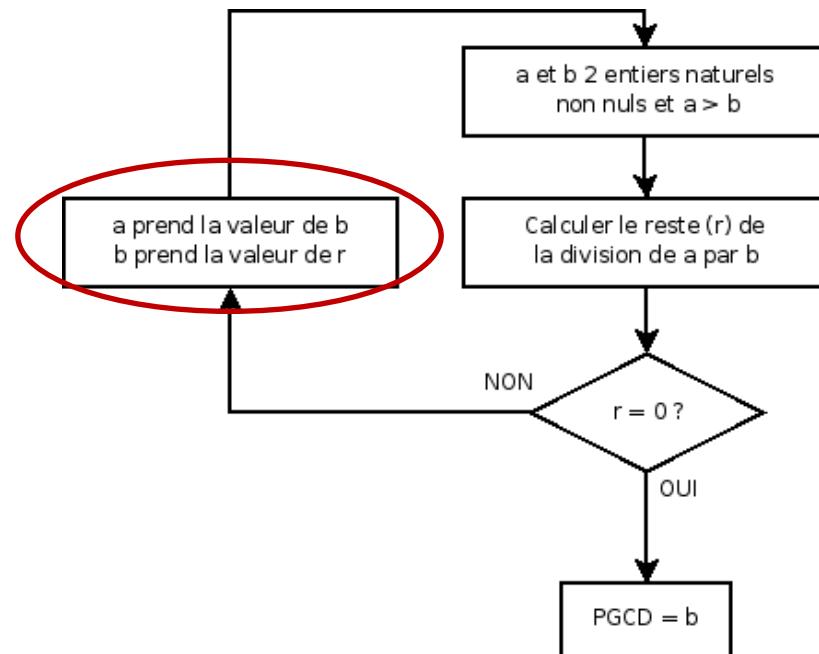
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | | | |
| 3 | | | |
| 4 | | | |

Un algorithme célèbre...

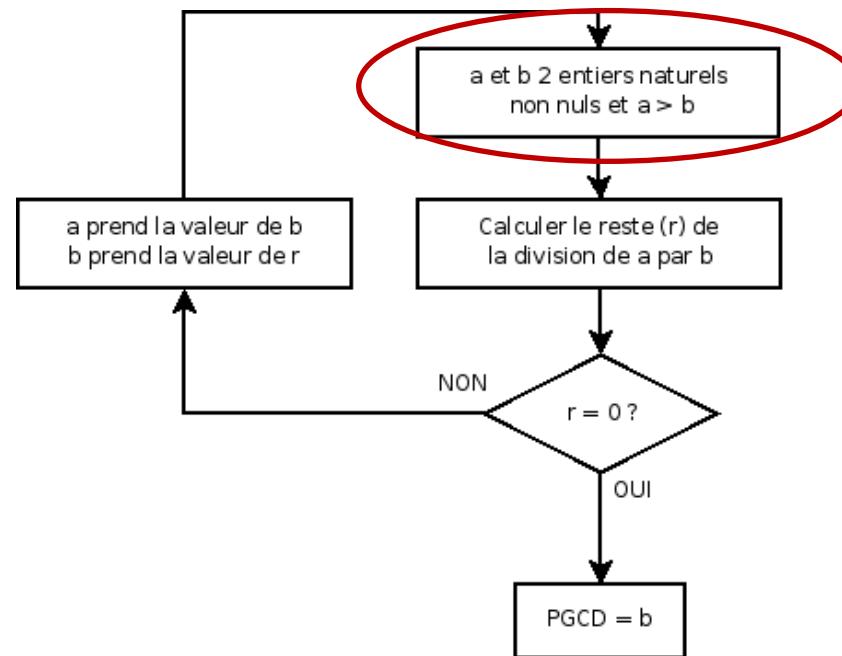
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | 54 | 24 | |
| 3 | | | |
| 4 | | | |

Un algorithme célèbre...

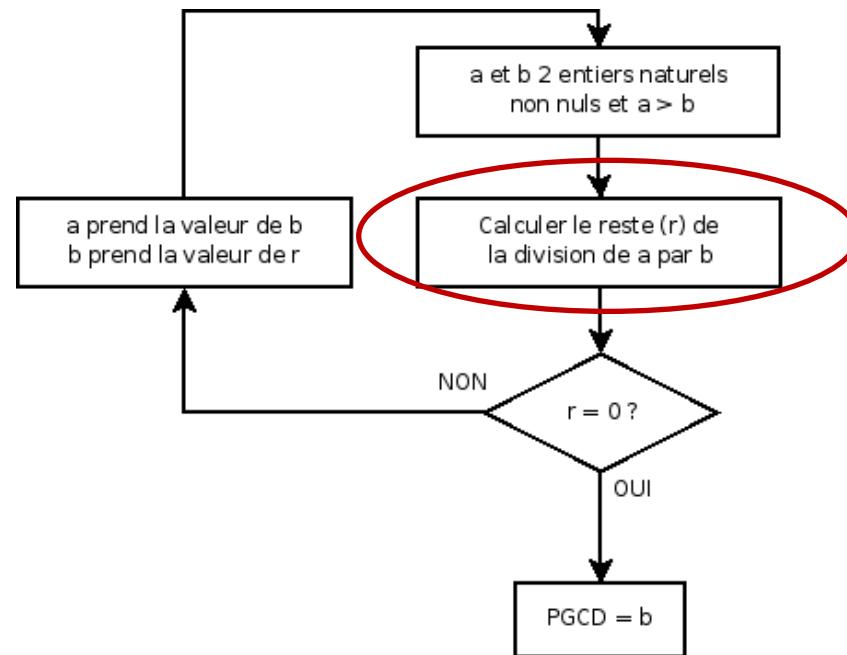
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | 54 | 24 | |
| 3 | | | |
| 4 | | | |

Un algorithme célèbre...

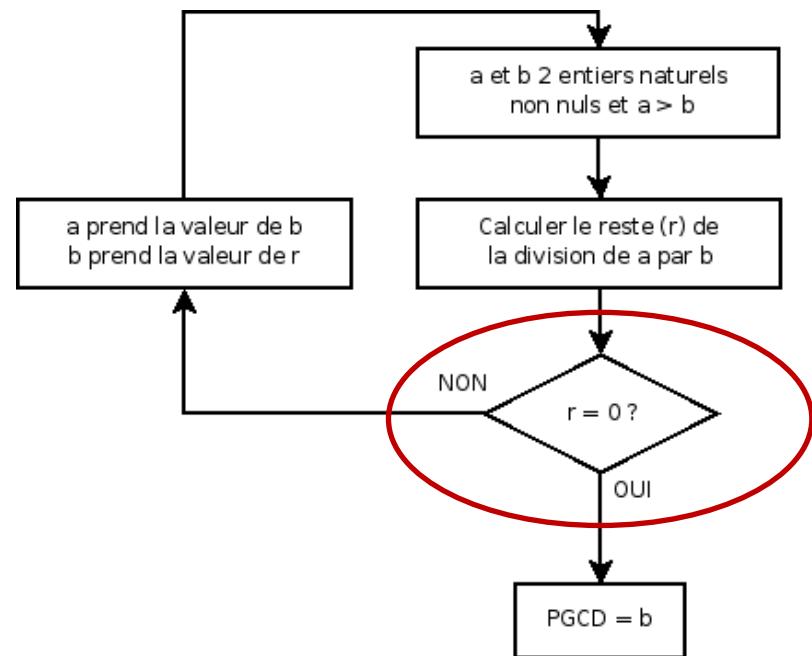
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | 54 | 24 | 6 |
| 3 | | | |
| 4 | | | |

Un algorithme célèbre...

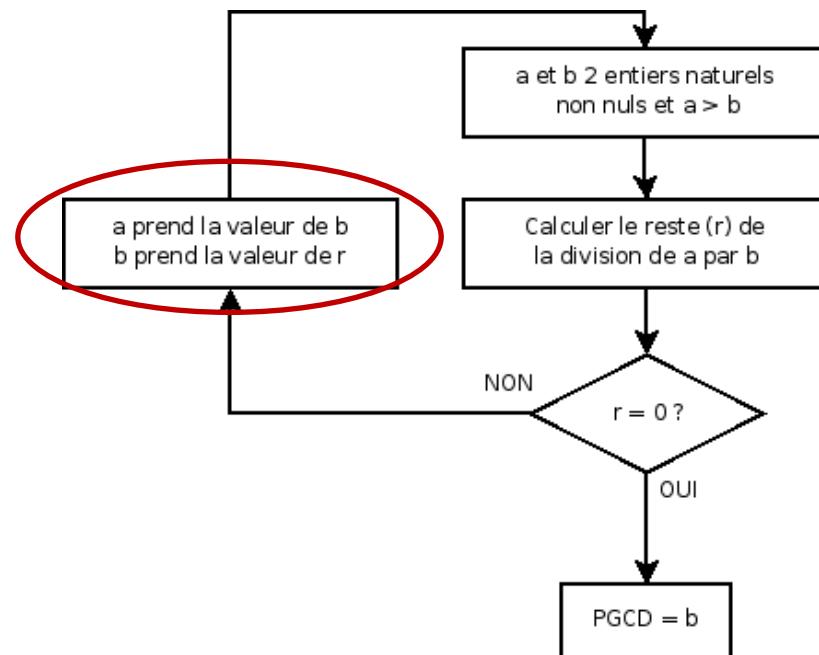
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | 54 | 24 | 6 |
| 3 | | | |
| 4 | | | |

Un algorithme célèbre...

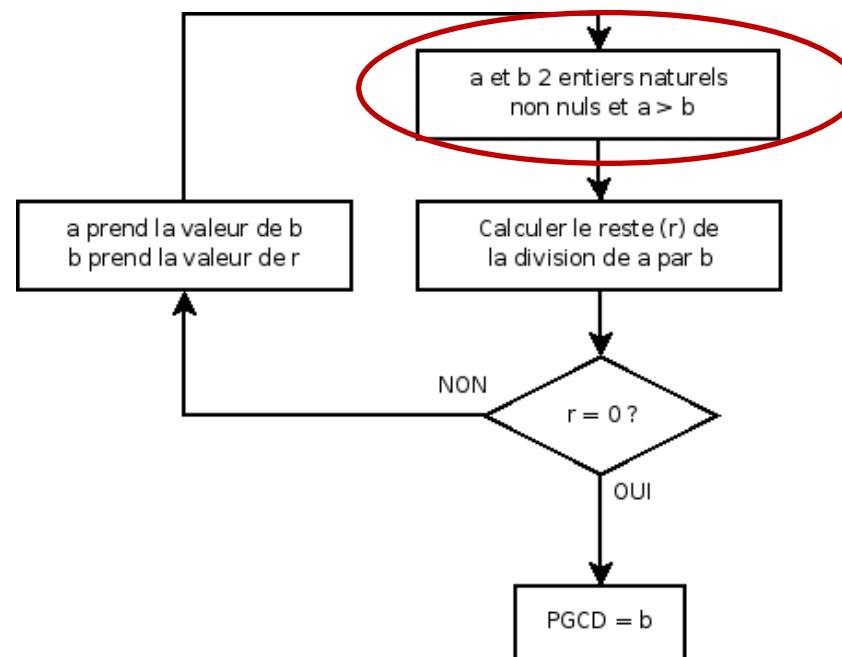
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | 54 | 24 | 6 |
| 3 | 24 | 6 | |
| 4 | | | |

Un algorithme célèbre...

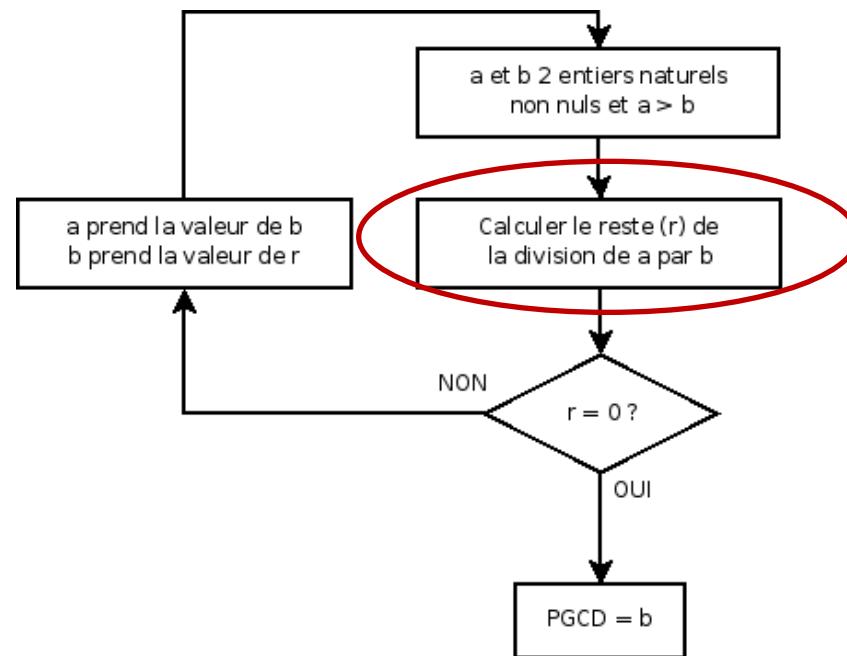
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | 54 | 24 | 6 |
| 3 | 24 | 6 | |
| 4 | | | |

Un algorithme célèbre...

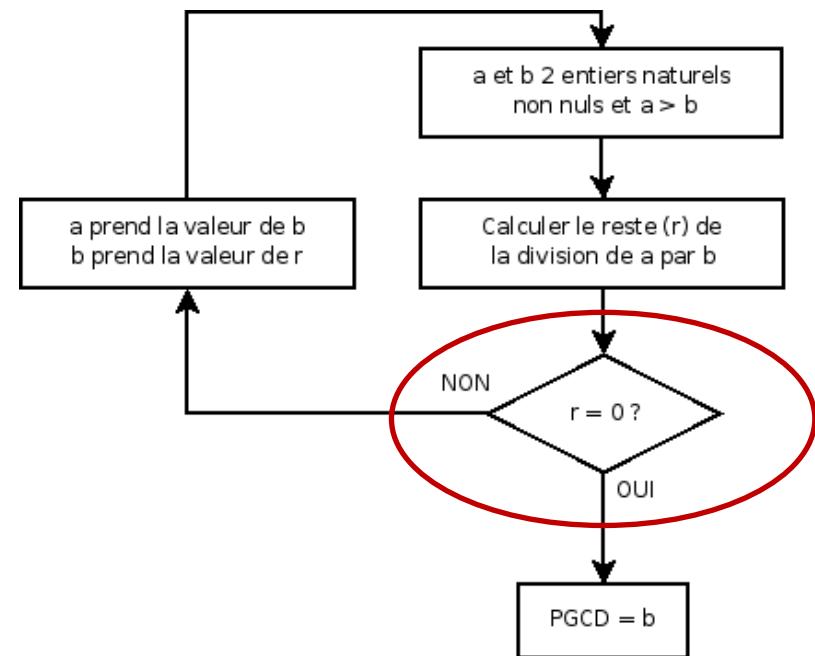
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | 54 | 24 | 6 |
| 3 | 24 | 6 | 0 |
| 4 | | | |

Un algorithme célèbre...

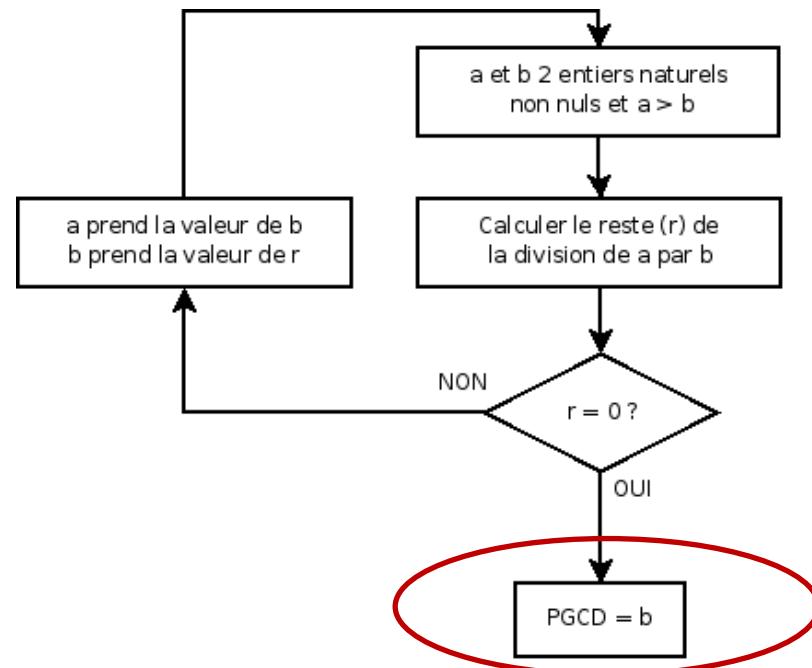
- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | 54 | 24 | 6 |
| 3 | 24 | 6 | 0 |
| 4 | | | |

Un algorithme célèbre...

- Algorithme d'Euclide (vers 300 av. J.-C)
 - Permet de calculer le **Plus Grand Commun Diviseur (PGCD)** de deux nombres



| # | a | b | r |
|---|-----|----|----|
| 1 | 186 | 54 | 24 |
| 2 | 54 | 24 | 6 |
| 3 | 24 | 6 | 0 |
| 4 | | | |

A red arrow points from the value "6" in the third row of the table to the word "PGCD" at the bottom, indicating that the algorithm has found the greatest common divisor.

Un algorithme célèbre...

- Les premiers algorithmiques, comme celui-ci, étaient souvent destinés à réaliser des opérations mathématiques, mais aujourd’hui les algorithmes réalisent des successions d’actions très diverses :
 - De la recette de cuisine,
 - À l’évolution d’un personnage dans un jeux vidéo,
 - En passant par l’intelligence artificielle,
 - etc.

Généralités

Un algorithme doit être...

- Lisible :
 - L'algorithme doit être **compréhensible** (même par un non-informaticien)
- De haut niveau :
 - L'algorithme doit pouvoir être **traduit en n'importe quel langage de programmation**, il ne doit donc pas faire appel à des notions techniques relatives à un programme particulier ou bien à un système d'exploitation donné
- Précis :
 - Chaque élément de l'algorithme ne doit pas porter à confusion, il est donc important **de lever toute ambiguïté**

Un algorithme doit être...

- Concis :
 - Un algorithme ne doit pas dépasser une page. Si c'est le cas, il faut **décomposer** le problème en plusieurs **sous-problèmes**
- Structuré :
 - Un algorithme doit être composé de **différentes parties** facilement **identifiées**

Un algorithme doit être...

- Concis :
 - Un algorithme ne doit pas dépasser une page. Si c'est le cas, il faut **décomposer** le problème en plusieurs **sous-problèmes**
- Structuré :
 - Un algorithme doit être composé de **différentes parties** facilement **identifiées**



Un algorithme doit suivre un **certain formalisme**



Écrire formellement les algorithmes, quelques exemples

Pourquoi écrire formellement un algorithme ?

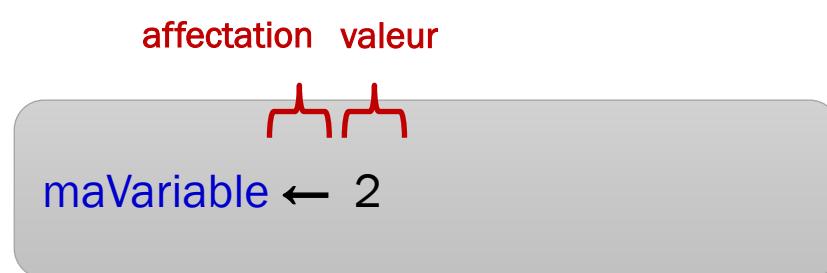
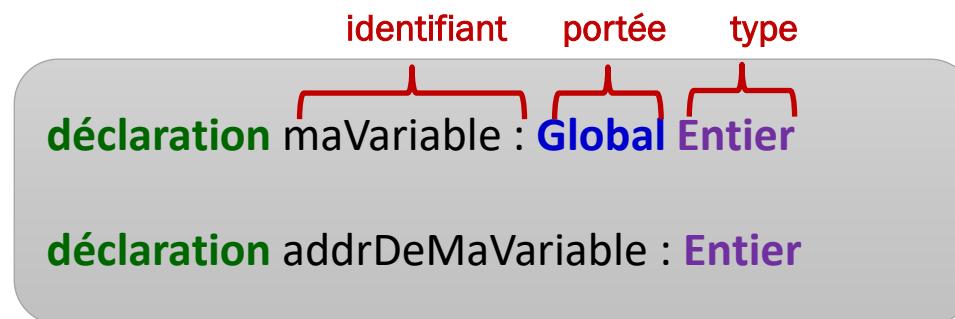
- Quelques soit le langage on peut écrire un algorithme soit en le dessinant (en UML par exemple), soit en l'écrivant (en français).
- On vous donne dans la suite quelques exemples pour que vous soyez capable de le faire.

Le code :

- Le code d'un algorithme est une succession d'opération.
- Cette succession d'opération est encapsulée dans des bouts de code qu'on peut réutiliser, on va parler de **fonction**.
- La fonction principale qui est appelée au début du programme est simplement appelée **le programme**.

Variable

- Dans un programme ou une fonction on peut retenir une valeur, on va parler de « **variable** ». (note, si on veut qu'une variable soit accessible partout on parle de variable globale, et si on ne veut pas qu'elle soit modifiée on parle de constante.)



En C ou en Java :

```
int maVariable;  
maVariable = 2;
```

En Python:

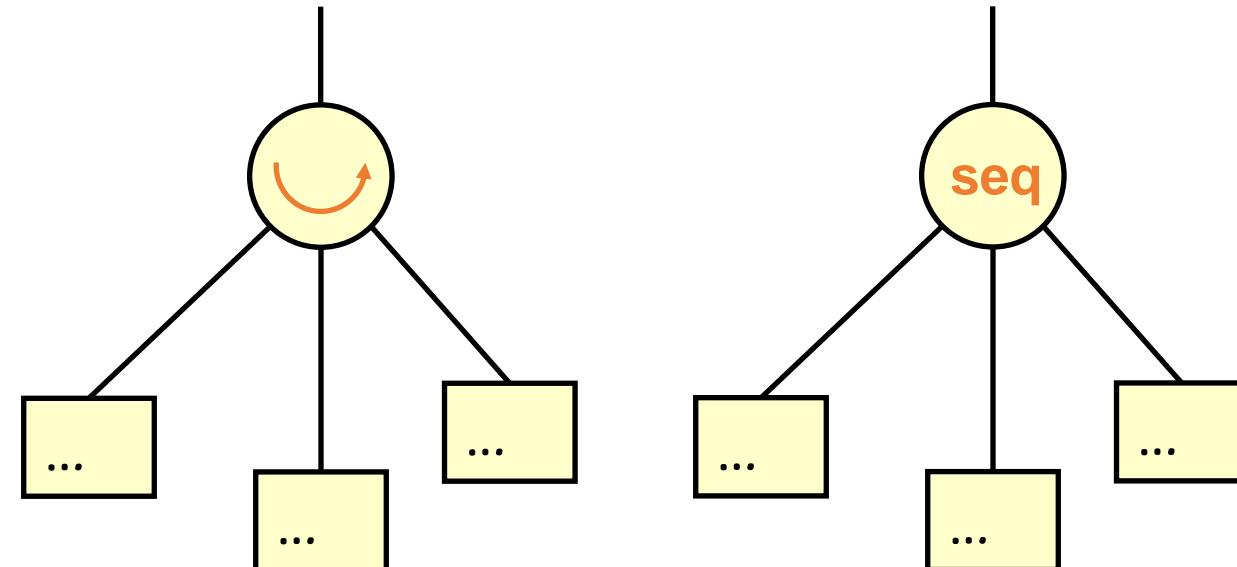
```
maVariable=2
```

En Python la déclaration est implicite gérée au moment de l'affectation

Séquence

- Les actions sont exécutées l'une après l'autre
 - Notion de bloc d'instructions

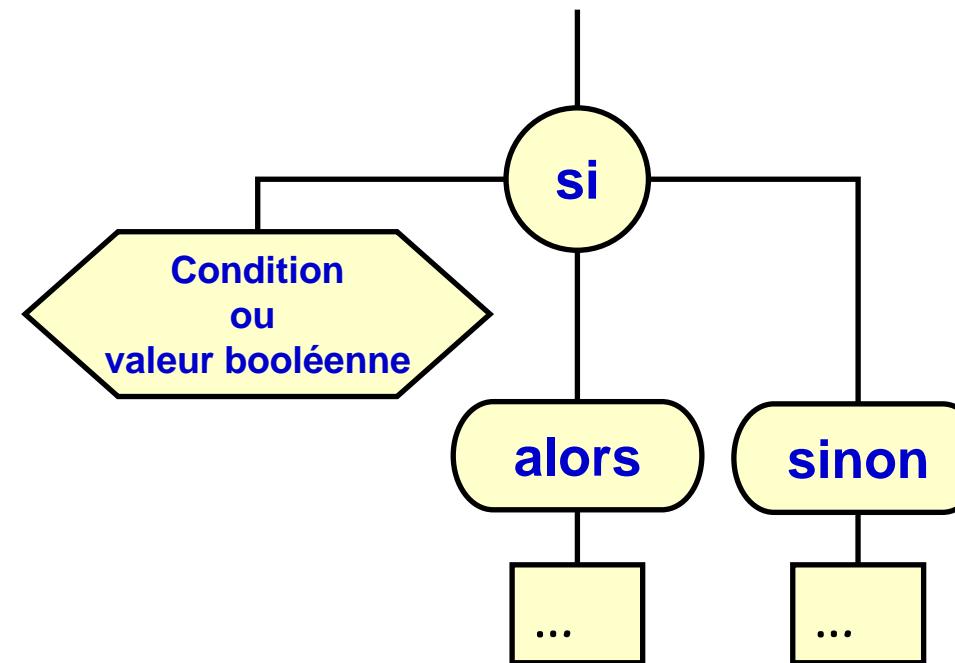
Faire
...
...
Fait



- Actions possibles
 - Affectation : opérations arithmétiques (ou logiques)
 - Affectation : appels à fonction
 - Appels à procédures
 - Opérations d'entrées/sorties (lire, écrire, imprimer, ...)

Alternative

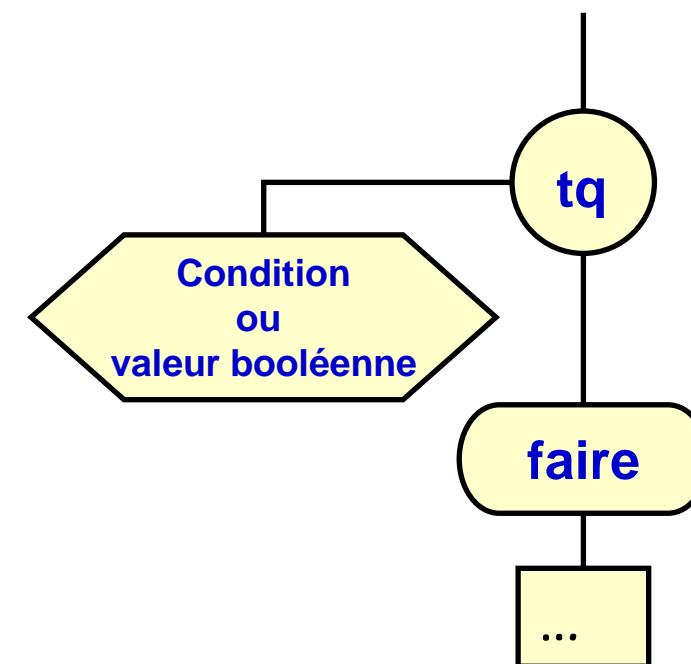
- Choix d'actions en fonction d'une valeur booléenne



Répétition (1)

- Répétition avec test en entrée

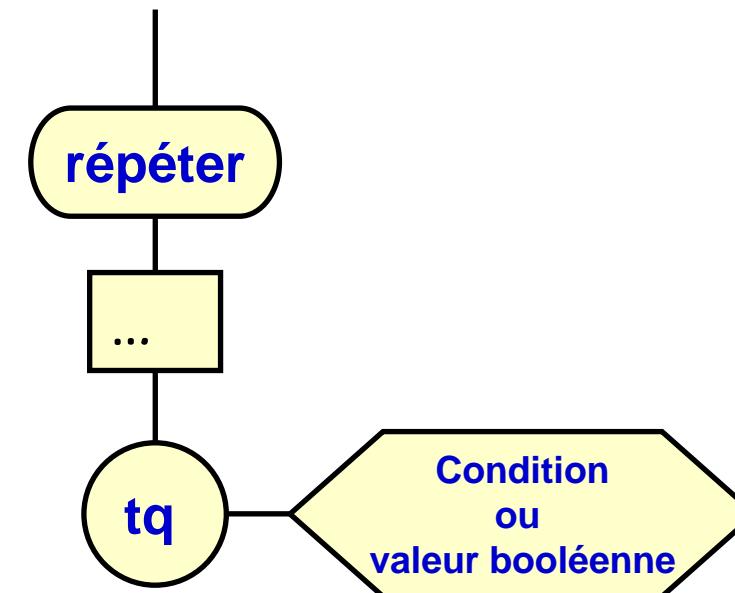
Tant que (condition ou valeur booléenne) faire
...
Fin tant que



Répétition (2)

- Répétition avec test en sortie

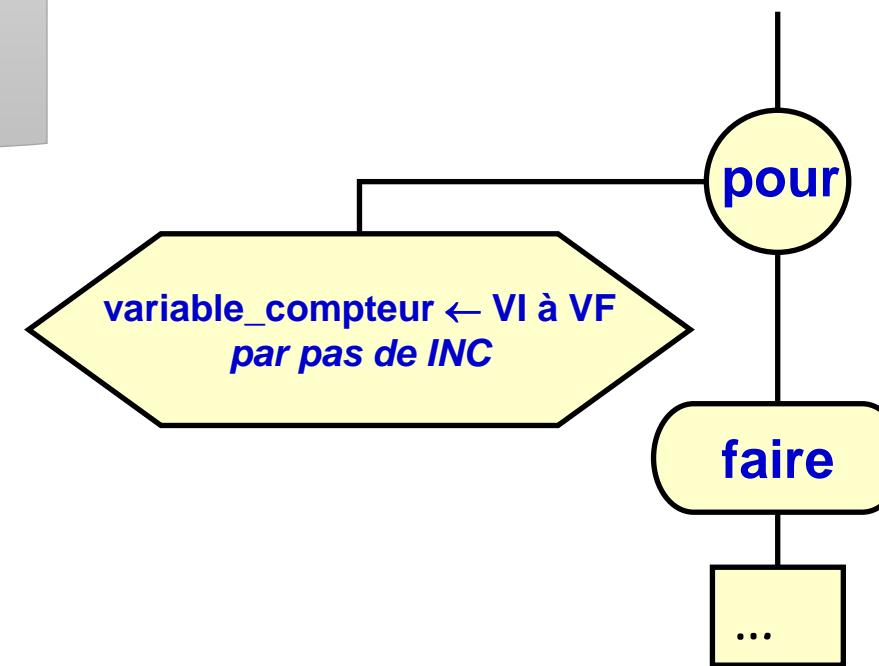
Répéter
...
Tant que (condition ou valeur booléenne)



Répétition (3)

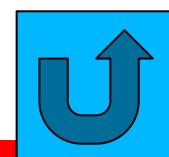
- Répétition avec variable compteur (itération)

```
Pour (variable_compteur ← VI à VF par pas de INC) faire  
    ...  
Fin Pour
```



Exemple 1 : Valeur absolue

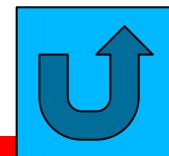
*Proposez l'algorithme d'une fonction nommée **abs** (sous forme de pseudo-langage) permettant de calculer la valeur absolue d'un réel x. Ecrivez également un exemple de programme principal appelant cette fonction et affichant le résultat. On pourra utiliser le programme **afficher(E x : Réel)***



Exemple 1 : Valeur absolue

Proposez l'algorithme d'une fonction nommée **abs** (sous forme de pseudo-langage) permettant de calculer la valeur absolue d'un réel x . Ecrivez également un exemple de programme principal appelant cette fonction et affichant le résultat. On pourra utiliser le programme **afficher(E x : Réel)**

```
Fonction abs(E x : Réel ) :  
Réel  
Déclaration xabs : Réel  
Début  
    Si x ≥ 0 alors  
        xabs ← x  
    Sinon  
        xabs ← -1 * x  
    FinSi  
    retourner xabs  
Fin abs
```

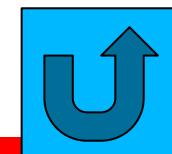


Exemple 1 : Valeur absolue

Proposez l'algorithme d'une fonction nommée **abs** (sous forme de pseudolangage) permettant de calculer la valeur absolue d'un réel x . Ecrivez également un exemple de programme principal appelant cette fonction et affichant le résultat. On pourra utiliser le programme **afficher(E x : Réel)**

Fonction abs(E x : Réel) :
Réel
Déclaration xabs : Réel
Début
 Si $x \geq 0$ **alors**
 xabs $\leftarrow x$
 Sinon
 xabs $\leftarrow -1 * x$
 FinSi
 retourner **xabs**
Fin abs

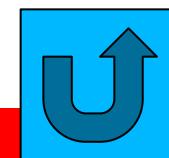
Programme test
Déclaration x,xabs : Réel
Début
 x $\leftarrow -3.14$
 xabs $\leftarrow \text{abs}(x)$
 afficher(xabs)
Fin test



Exemple 2 : Le problème de la reprographie

Un magasin de reprographie facture 0,10 euros les dix premières photocopies, 0,09 euros les vingt suivantes et 0,08 euros au-delà.

*Ecrivez une fonction nommée **calculPrix** qui calcule le prix de n photocopies. On vous demande ensuite d'écrire le programme principal qui appelle la fonction de calcul pour $n=55$ et qui affiche le résultat.*

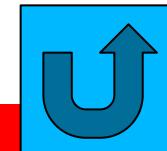


Exemple 2 : Le problème de la reprographie

Un magasin de reprographie facture 0,10 euros les dix premières photocopies, 0,09 euros les vingt suivantes et 0,08 euros au-delà.

*Ecrivez une fonction nommée **calculPrix** qui calcule le prix de n photocopies. On vous demande ensuite d'écrire le programme principal qui appelle la fonction de calcul pour $n=55$ et qui affiche le résultat.*

```
Fonction calculPrix (E n : Entier ) : flottant
Déclaration prix : Flottant
Début
    si n <= 10
        prix ← n*0.1
    sinon    si n <= 30
        prix ← (n-10)*0.09+1
    sinon
        prix ← (n-
30)*0.08+2.8
    FinSi
    FinSi
    retourner prix
Fin calculPrix
```



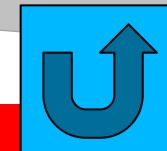
Exemple 2 : Le problème de la reprographie

Un magasin de reprographie facture 0,10 euros les dix premières photocopies, 0,09 euros les vingt suivantes et 0,08 euros au-delà.

Ecrivez une fonction nommée **calculPrix** qui calcule le prix de n photocopies. On vous demande ensuite d'écrire le programme principal qui appelle la fonction de calcul pour $n=55$ et qui affiche le résultat.

```
Fonction calculPrix (E n : Entier ) : flottant
Déclaration prix : Flottant
Début
    si n <= 10
        prix ← n*0.1
    sinon
        si n <= 30
            prix ← (n-10)*0.09+1
        sinon
            prix ← (n-
30)*0.08+2.8
        FinSi
    FinSi
    retourner prix
Fin calculPrix
```

```
Programme test
Déclaration n : Entier
                    prix : Flottant
Début
    n ← 55
    prix ← calculPrix(n)
    afficher(prix)
Fin test
```





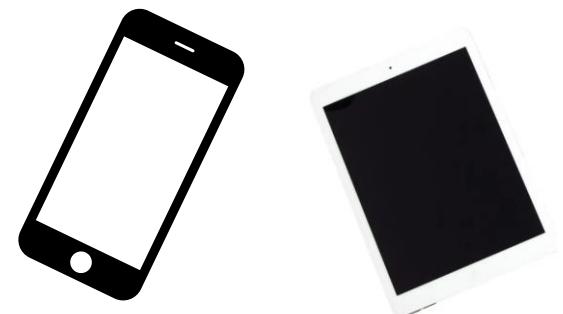
Algorithmique et informatique : Du matériel au langage de programmation

Cours de **Nils Beaussé** et **Bilel BENZIANE**
nils.beausse@isen-ouest.yncrea.fr
bilel.benziane@isen-ouest.yncrea.fr

D'après les notes de cours et le
cours de Benoit Lardeux, Jean-
Benoît Pierrot, Pierre-Jean
Bouvet et Leandro Montero

Généralités

- Objectif :
 - Vocabulaire
 - Compréhension de base
- Smartphones et tablettes sont aussi des ordinateurs
- Evalué dans le QCM final



Questions ouvertes :

- Qu'est-ce qu'un ordinateur ?
- Quelles sont les 2 fonctions principales d'un ordinateur ?

Généralités

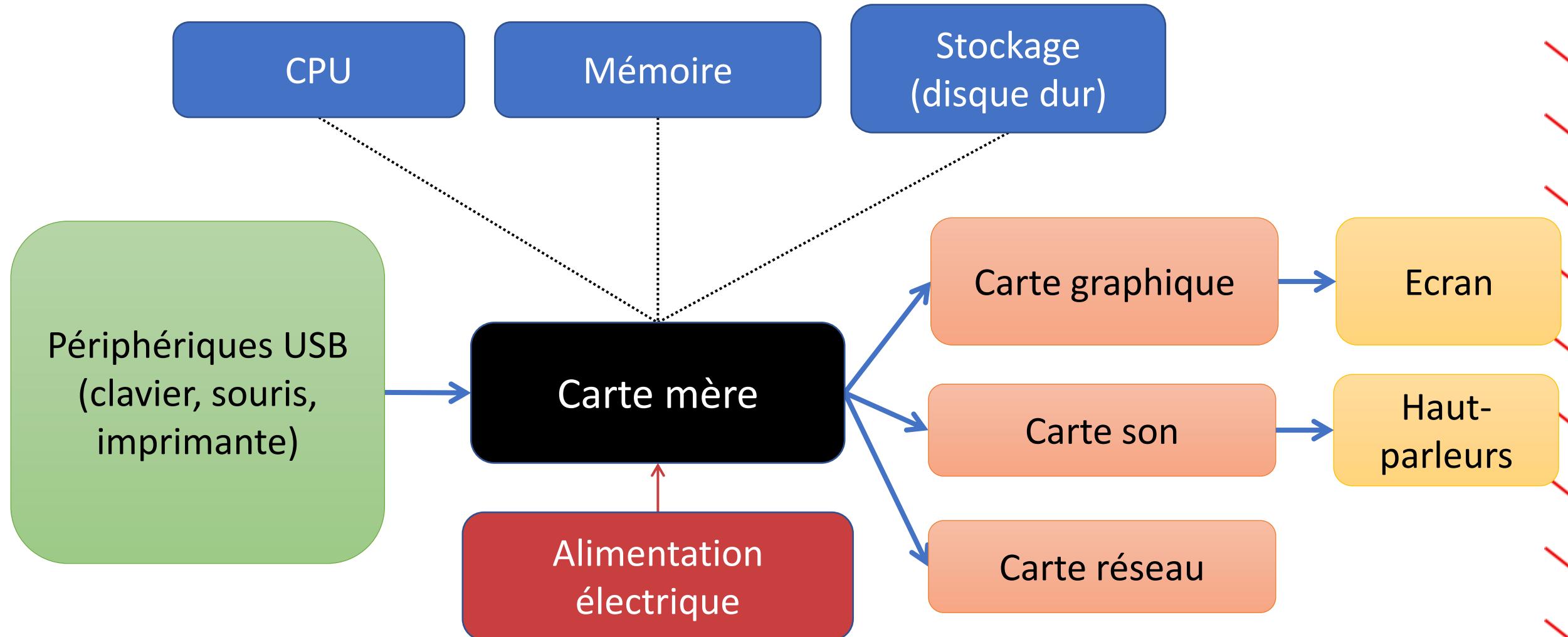
- Fonction d'un ordinateur :
 - Stocker des données
 - Données personnelles, images, fichiers textes...
 - Exécuter des programmes qui traiteront les données
 - Programme = Suites d'instructions traitées par l'ordinateur pour effectuer des tâches.

Fonctionnement d'un ordinateur

- On peut voir un ordinateur comme un système disposant d'entrée et de sorties.
- On parle en général de « **périphériques** » pour tout ce qui rattaché à un ordinateur.



Fonctionnement d'un ordinateur



Fonctionnement d'un ordinateur

- Cette simplification permet de bien comprendre mais :

Fonctionnement d'un ordinateur

- Cette simplification permet de bien comprendre mais :
 - La réalité est plus nuancée ...

Fonctionnement d'un ordinateur

- Cette simplification permet de bien comprendre mais :
 - La réalité est plus nuancée ...
- En effet : Sur du matériel moderne ->

Fonctionnement d'un ordinateur

- Cette simplification permet de bien comprendre mais :
 - La réalité est plus nuancée ...
- En effet : Sur du matériel moderne ->
 - Les périphériques d'entrées reçoivent également des informations de l'ordinateur (demande d'information au branchement, éclairage etc.)

Fonctionnement d'un ordinateur

- Cette simplification permet de bien comprendre mais :
 - La réalité est plus nuancée ...
- En effet : Sur du matériel moderne ->
 - **Les périphériques d'entrées** reçoivent également des informations de l'ordinateur (demande d'information au branchement, éclairage etc.)
 - **Les périphériques de sorties** envoient également des informations à l'ordinateur (synchronisation de l'écran etc.)

Fonctionnement d'un ordinateur

- Cette simplification permet de bien comprendre mais :
 - La réalité est plus nuancée ...
- En effet : Sur du matériel moderne ->



Fonctionnement d'un ordinateur

- Cette simplification permet de bien comprendre mais :
 - La réalité est plus nuancée ...
- En effet : Sur du matériel moderne ->

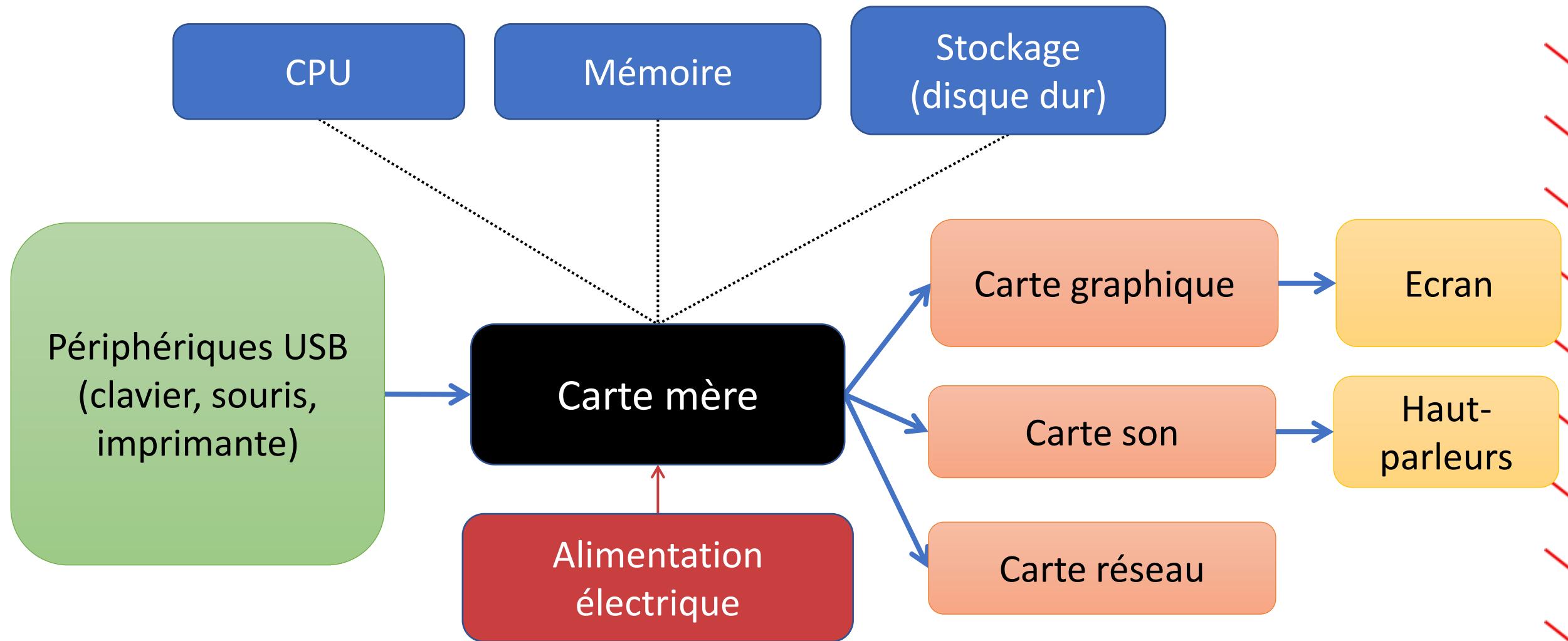


Fonctionnement d'un ordinateur

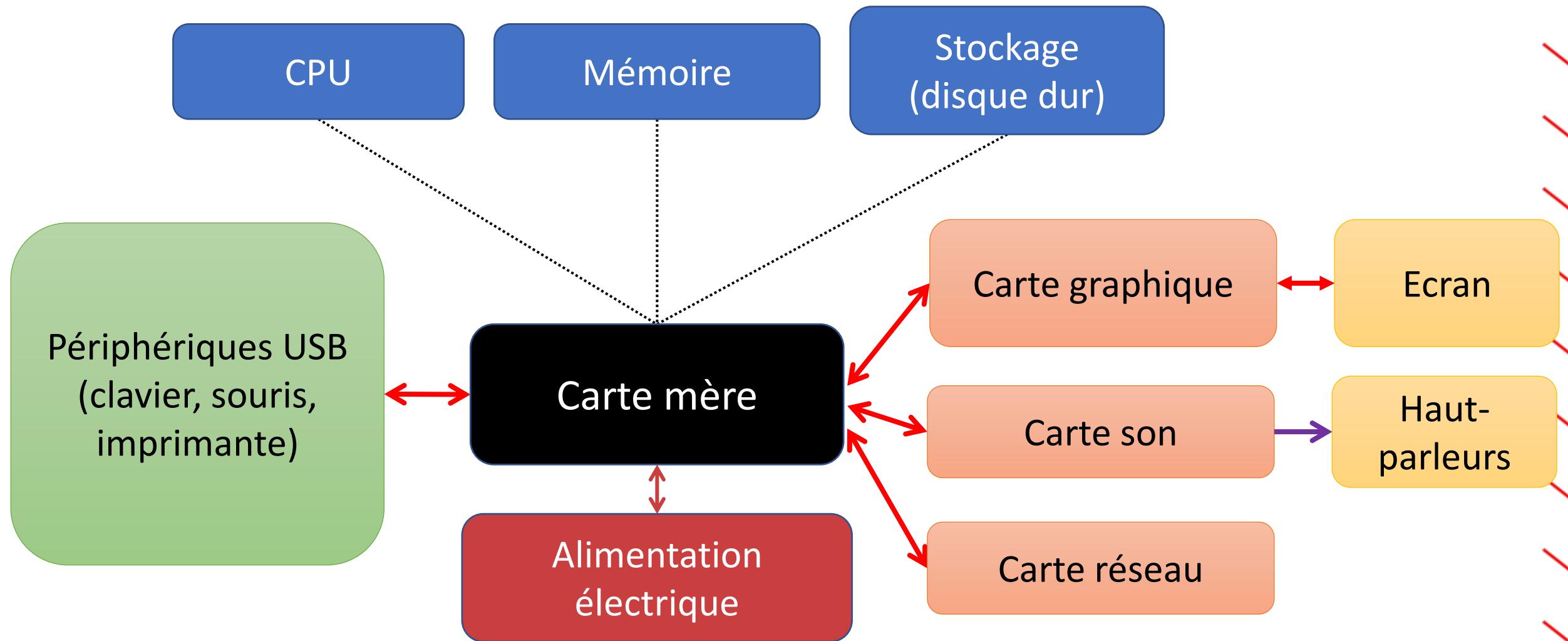
- Cette simplification permet de bien comprendre mais :
 - La réalité est plus nuancée ...
- En effet : Sur du matériel moderne ->



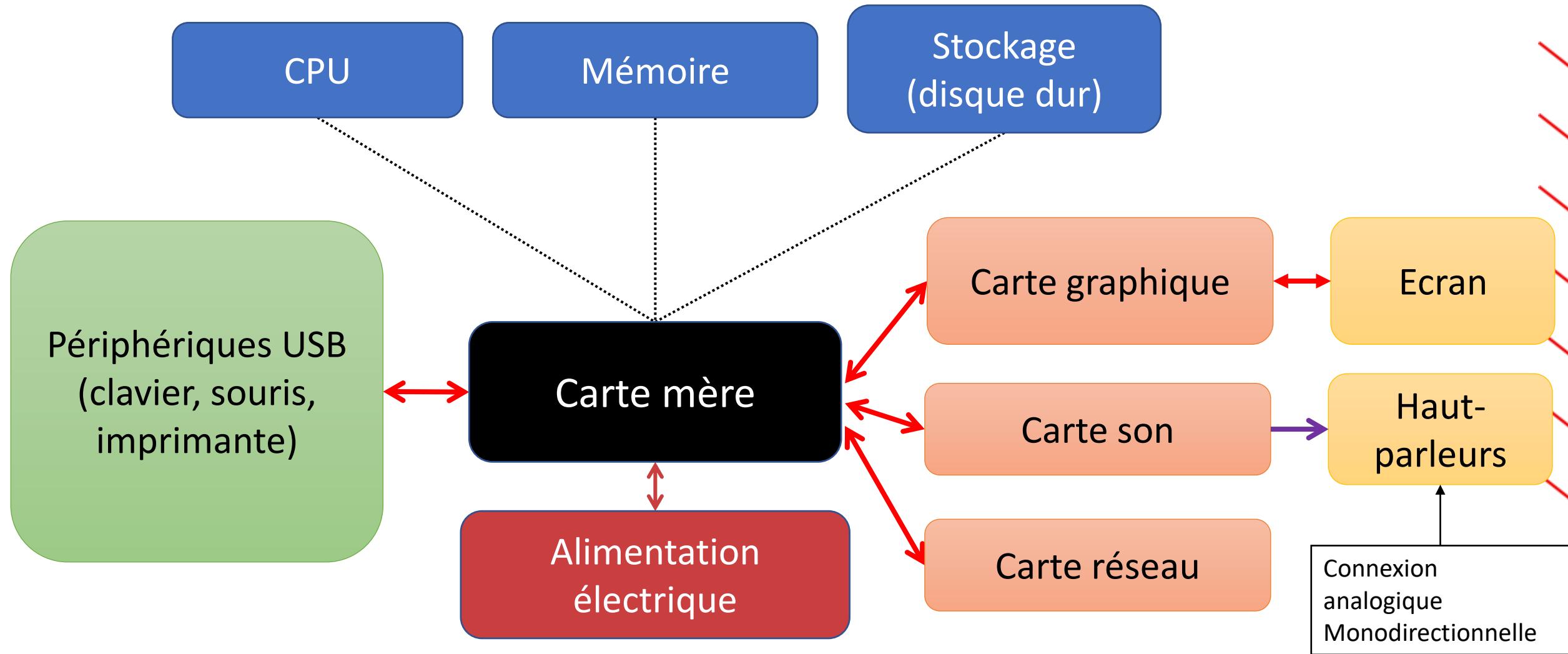
Fonctionnement d'un ordinateur



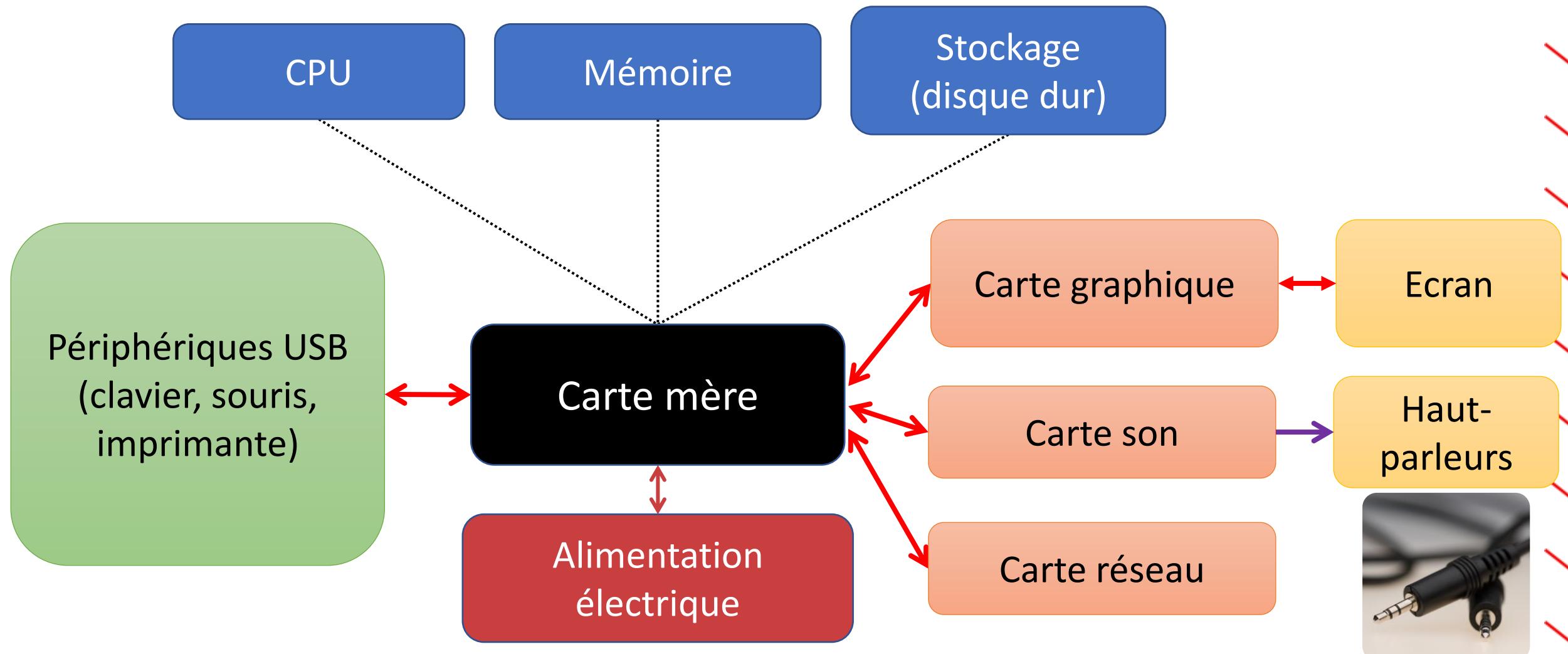
Fonctionnement d'un ordinateur



Fonctionnement d'un ordinateur



Fonctionnement d'un ordinateur



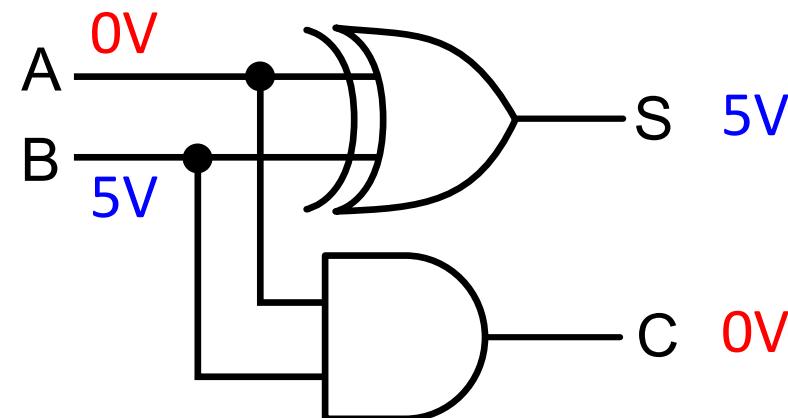
Fonctionnement du processeur

Fonctionnement du processeur

= « microprocesseur »
= « CPU » (Central Processing Unit)

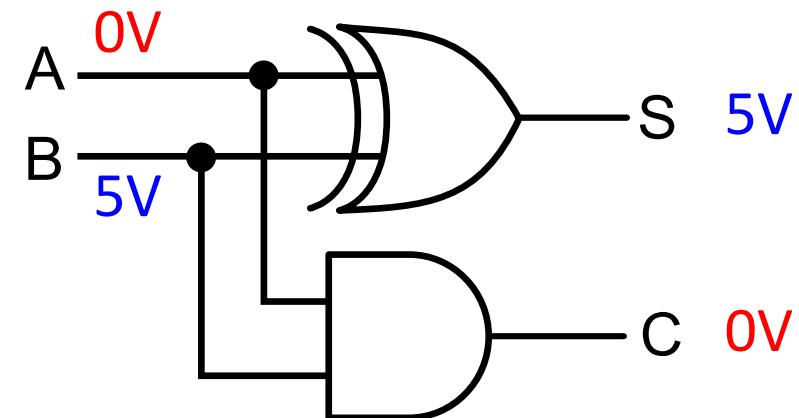
Principe de base

- Un processeur est basée sur le principe d'assemblages de composant électronique.
- L'ensemble permet de donner des **entrées binaires** sous forme de signaux électrique ($0 \Rightarrow 0$ | $5V \Rightarrow 1$ par exemple)
- On souhaite obtenir de l'autre coté de l'assemblage un résultat (addition de deux chiffres par exemple)
- Exemple : $0+1 = 1$



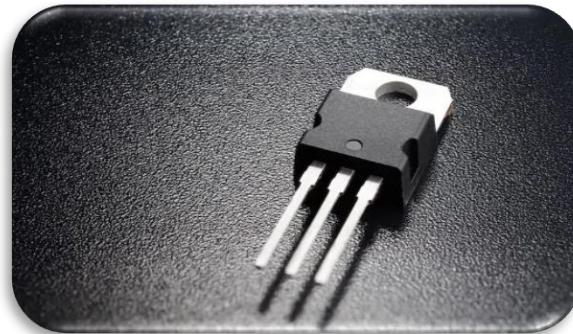
Principe de base

- Pour ces circuités électroniques on parle souvent de « porte logique »
- Sur l'exemple précédent, en haut, une porte XOR et en bas une porte ET



Principe de base

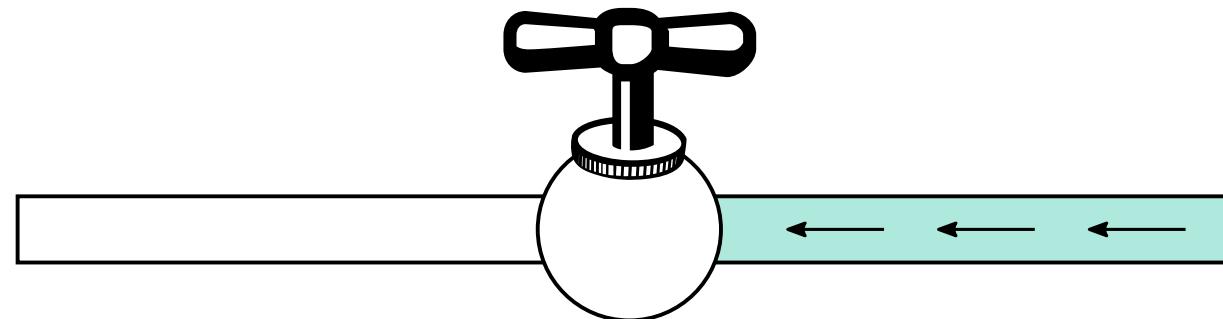
- Ces portes sont employées pour simplifier la compréhension, mais en réalité elles sont elles mêmes des assemblages de **transistors**.



- Le transistor est un composant électronique qui a la propriété d'agir comme un **robinet pour l'électricité**.

Principe de base

- Le transistor est un composant électronique qui a la propriété d'agir comme un **robinet pour l'électricité**.

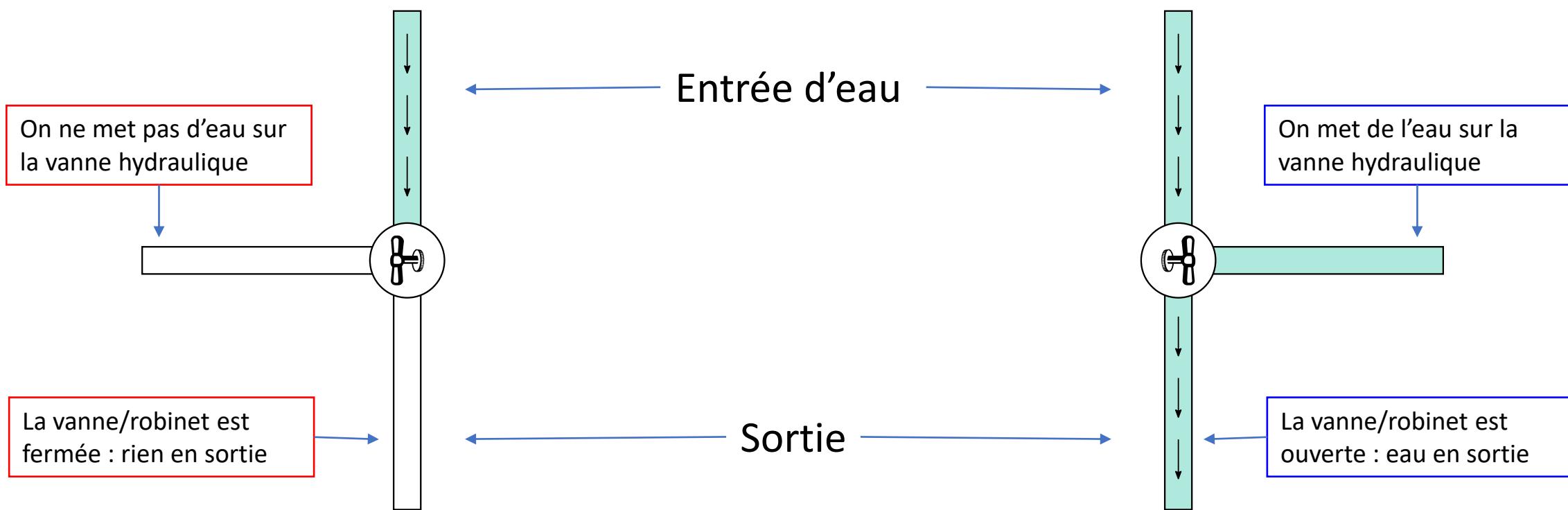


Principe de base

- Le transistor est un composant électronique qui a la propriété d'agir comme un **robinet pour l'électricité**.
- Considérons que le robinet lui-même est hydraulique (il s'active avec la pression de l'eau). Qu'obtient-on ?

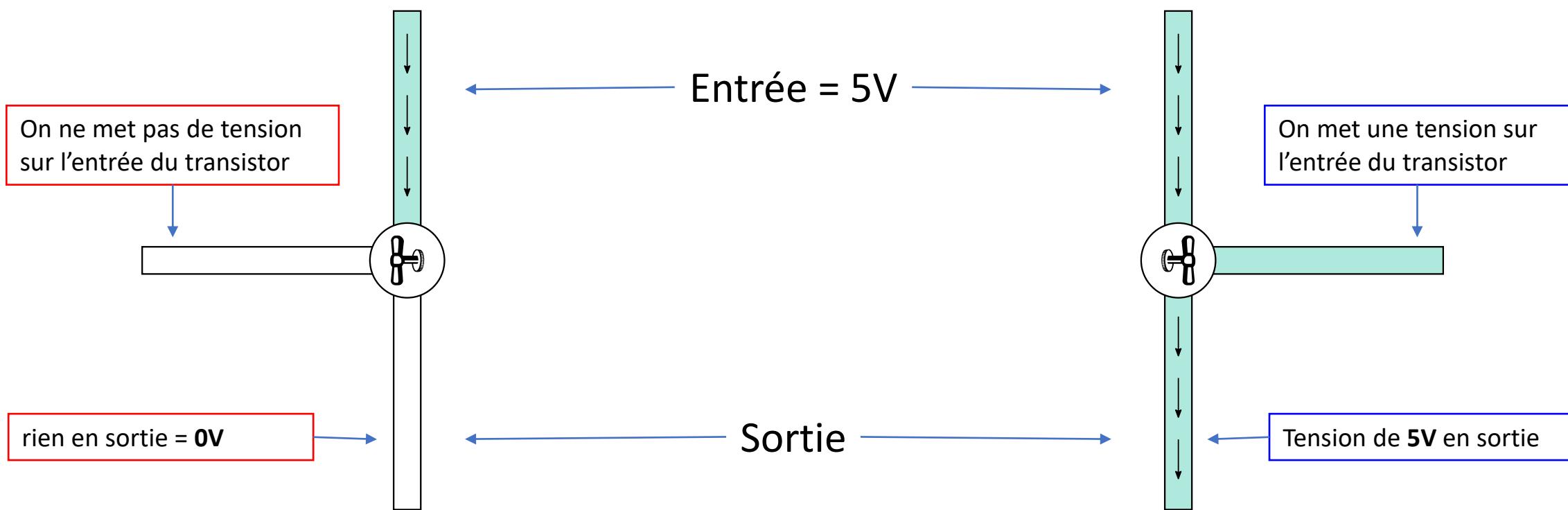
Principe de base

- Considérons que le robinet lui-même est hydraulique (il s'active avec la pression de l'eau). Qu'obtient-on ?



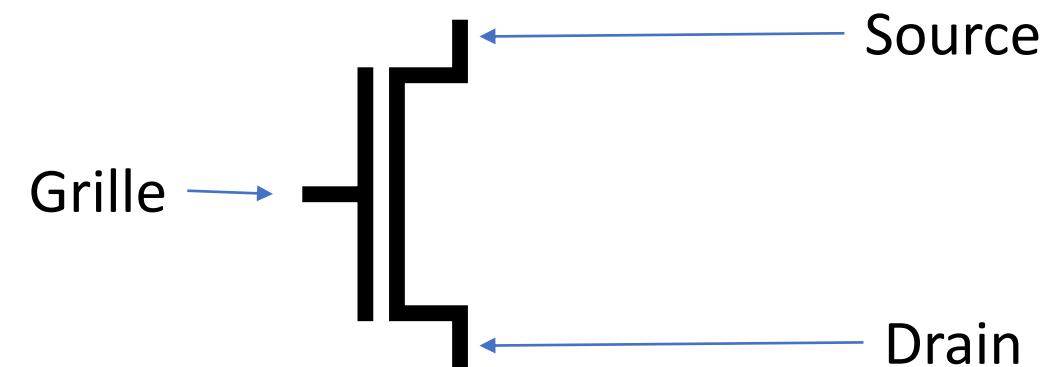
Principe de base

- Les transistors fonctionnent essentiellement de la même manière avec de l'électricité !



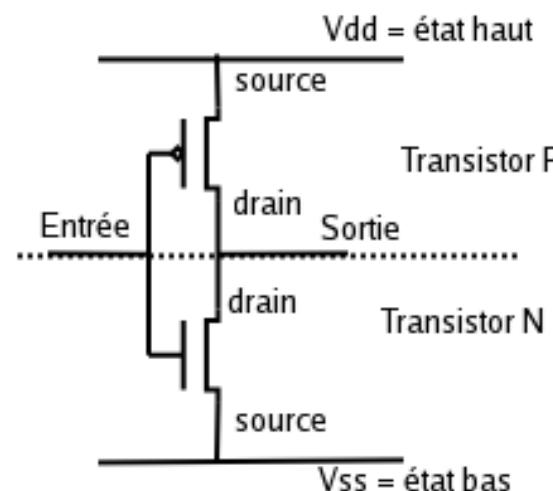
Principe de base

- En électronique on écrira plutôt les choses ainsi :

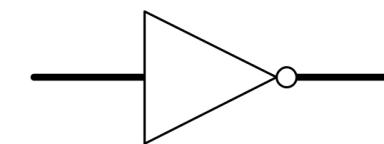


Principe de base

- Avec peu de transistors on peut réaliser très vite des portes logiques élémentaires



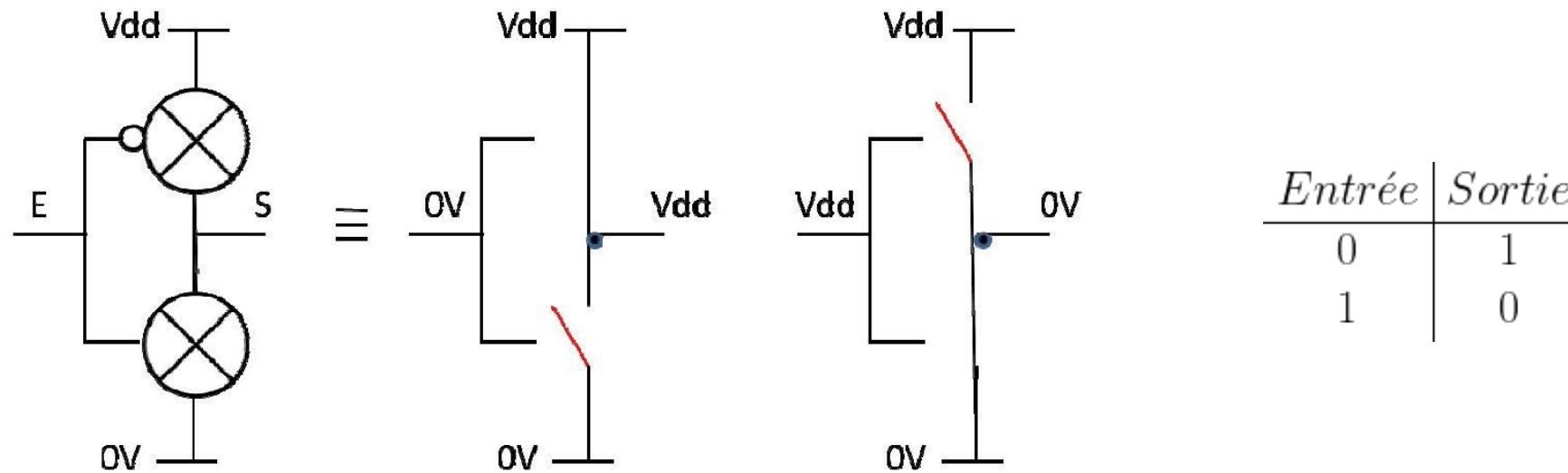
= inverseur =



| <i>Entrée</i> | <i>Sortie</i> |
|---------------|---------------|
| 0 | 1 |
| 1 | 0 |

Principe de base

- Avec peu de transistors on peut réaliser très vite des portes logiques élémentaires

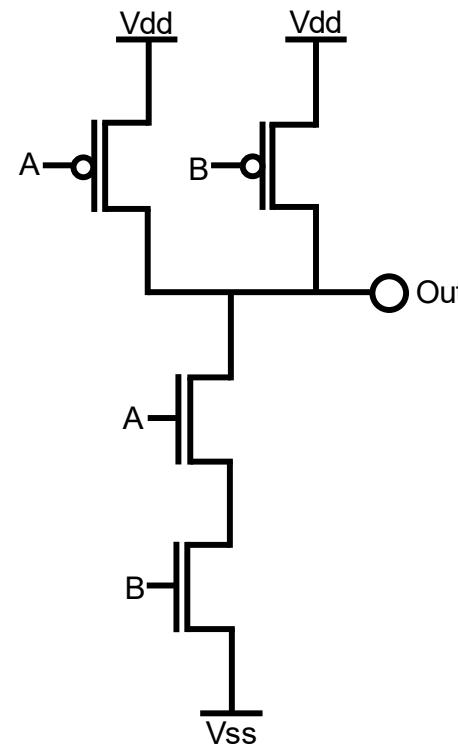


Note : dans la vrai vie, pendant un tout petit instant les deux transistors sont passant, ce qui conduit à la consommation de l'énergie.

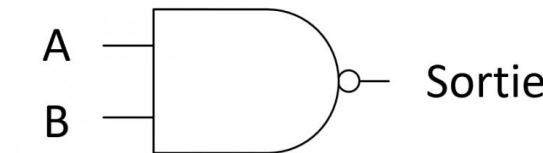
Plus la porte change d'état souvent, plus le phénomène se reproduit = chauffe = destruction si trop = limite de vitesse de l'ensemble.

Principe de base

- Avec peu de transistors on peut réaliser très vite des portes logiques élémentaires



= « NON – ET (NAND) » =



| A | B | Sortie |
|---|---|--------|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Principe de base

- Etc....
- En combinant ces assemblages élémentaires on obtient les portes :
 - Et
 - Ou
 - Inverse
 - Non-ET
 - Non-Ou

Principe de base

- La combinaison de ces portes nous permet de réaliser n'importe quel type d'opération sur des chiffres binaires, notamment :
 - Addition
 - Multiplication
 - Soustraction
 - Etc.
- On peut également les boucler pour réaliser de la **mémoire**.
- **Vous verrez tout ceci plus en détail en électronique et physique.**

Pourquoi le processeur ?

- Au départ on se contentait de circuits qui n'avaient qu'un seul rôle.
- Mais on devait alors rentrer manuellement les entrées électroniques dans le circuit (un circuit pour les additions, un autre pour les soustractions etc.)
- Ce type de circuit simple n'est pas pratique : il a fallut rajouter des indicatifs au début des nombres binaires pour qu'un circuit secondaire viennent orienter le résultat (par exemple 1 au début = addition, 0 = soustraction etc.)
- Il a fallut en plus rajouter des mémoires pour pouvoir enchaîner les opérations ! ($A+B=C$ $C*D = \text{résultat}$)

Pourquoi le processeur ?

Cet ensemble de circuit de base (additionneur / multiplicateur etc.)

+

Les circuits permettant de récupérer et stocker les résultats dans une zone ou une autre de la mémoire

+

Les mécanismes permettant de choisir quelle type d'opérations faire

=

Processeur

Programme

Programme

- Exemple de programme :

```
x = 4  
y = 8  
if x == 10:  
    y = 9  
else:  
    x = x+1  
    z = 6
```

Que fais ce programme ?

Programme

- Exemple de programme :

```
x = 4
y = 8
if x == 10:
    y = 9
else:
    x = x+1
    z = 6
```

Que fais ce programme ?

- Traité par votre ordinateur en binaire
 - Suite de 0 et de 1
 - Qui ont du sens pour la machine
 - Manipulées ensuite par le processeur de votre machine
 - Cette suite de 0 et de 1 est dite « Langage Machine »

Introduction : programme

- Un langage de programmation est transformé en langage machine
 - Par un compilateur (ex : C)
 - Par un interpréteur (ex : python)

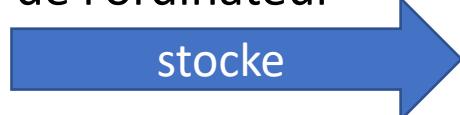
```
x = 4  
y = 8  
if x == 10:  
    y = 9  
else:  
    x=x+1  
z=6
```



- Le processeur exécute le langage machine

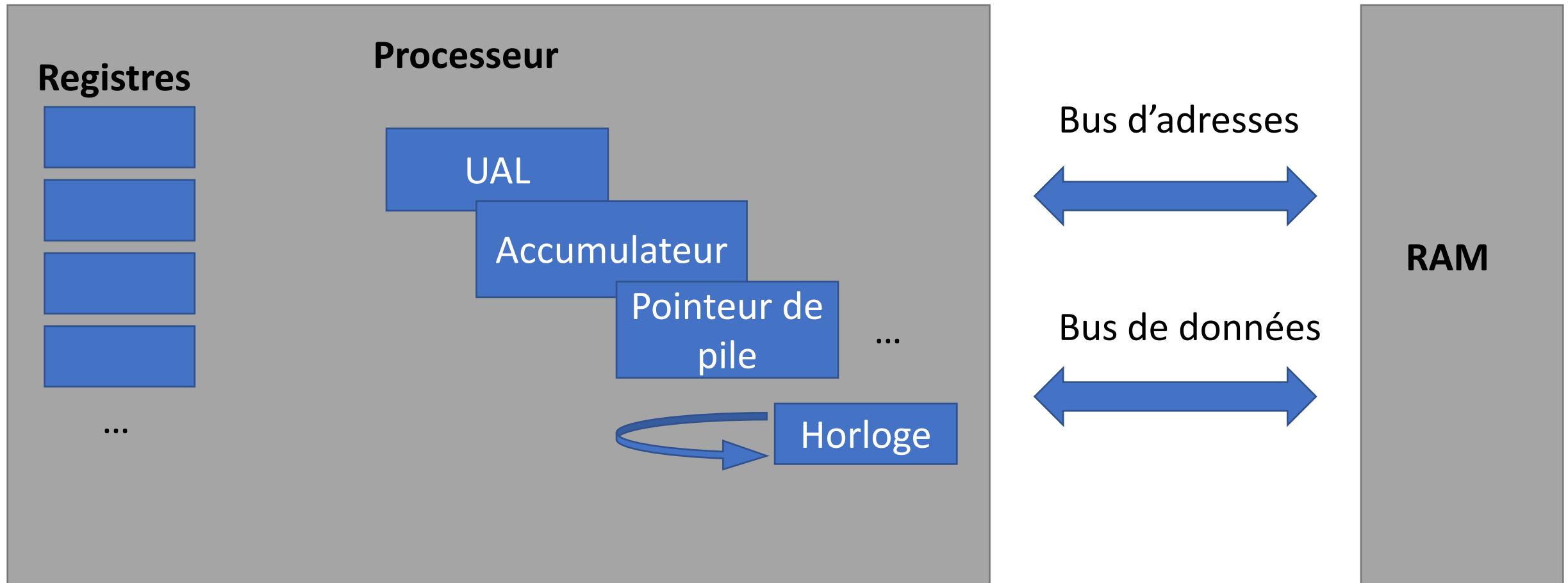
```
MOV R0, #4  
STR R0,30  
MOV R0, #8  
STR R0,75  
LDR R0,30  
CMP R0, #10  
BNE else  
MOV R0, #9  
STR R0,75  
B endif  
else:  
R0,30  
ADD R0, R0, #1  
STR R0,30  
endif:  
MOV R0, #6  
STR R0,23  
HALT
```

Capacité du processeur

- Processeur : cerveau de l'ordinateur
 - Mémoire  stocke  Instructions  exécutées par  Processeur
 - Transfert par Bus
 - Processeur traite les données présentes dans ses registres
- Traitements possibles :
 - instructions mathématiques simples : + - / x ...
 - opérateurs logiques : > < égalité, ...
 - décalages de bits,
 - lecture dans un registre,
 - saut à un endroit du code...
- instructions décrites en assembleur.
 - Pas un langage de programmation
 - Mais traduction du langage machine

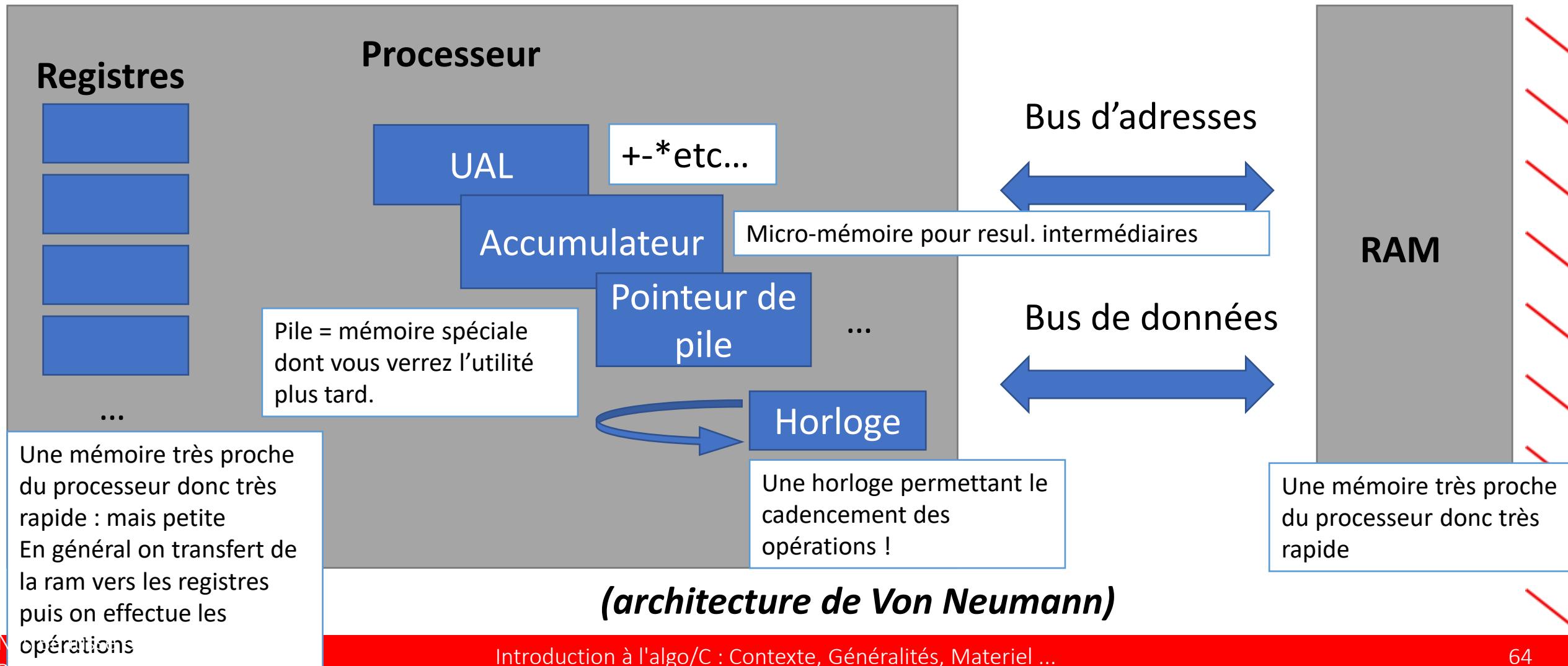
movb \$0x61,%al  10110000 01100001

Fonctionnement du processeur

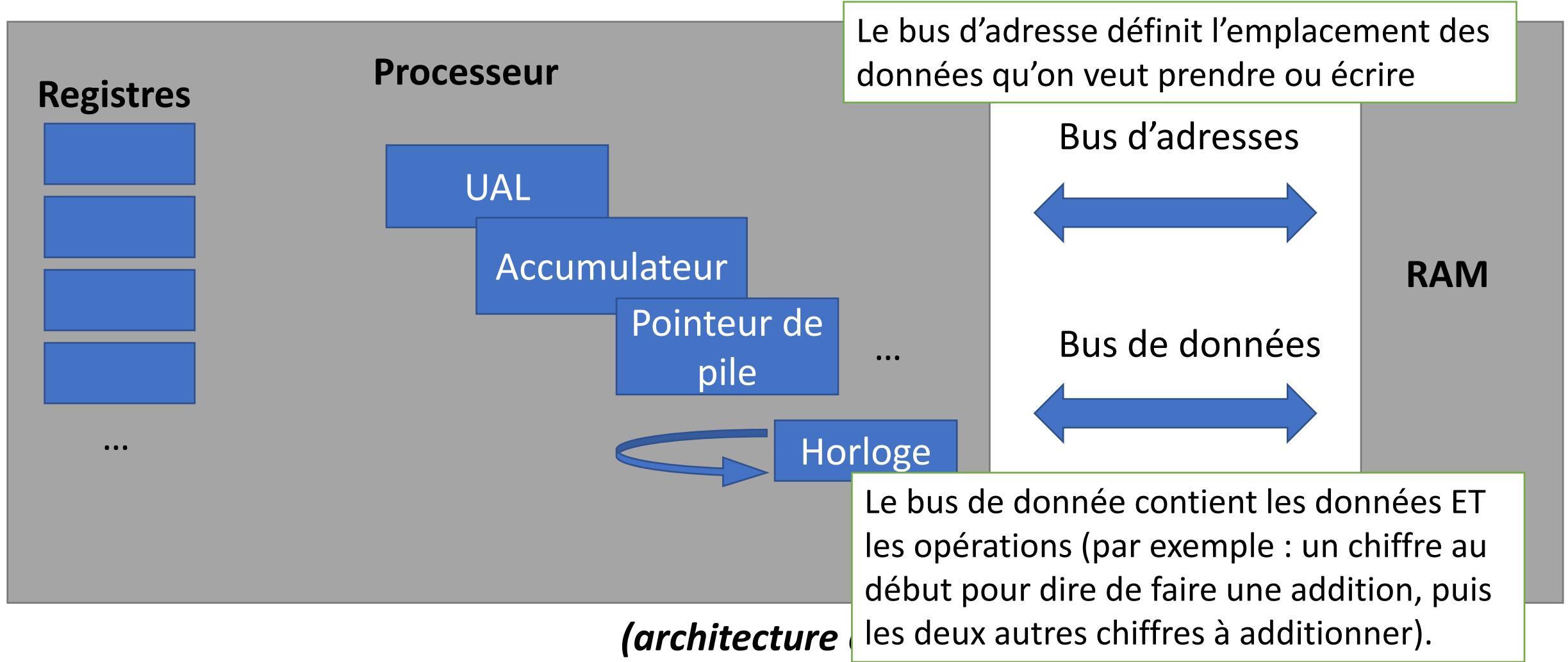


(architecture de Von Neumann)

Fonctionnement du processeur

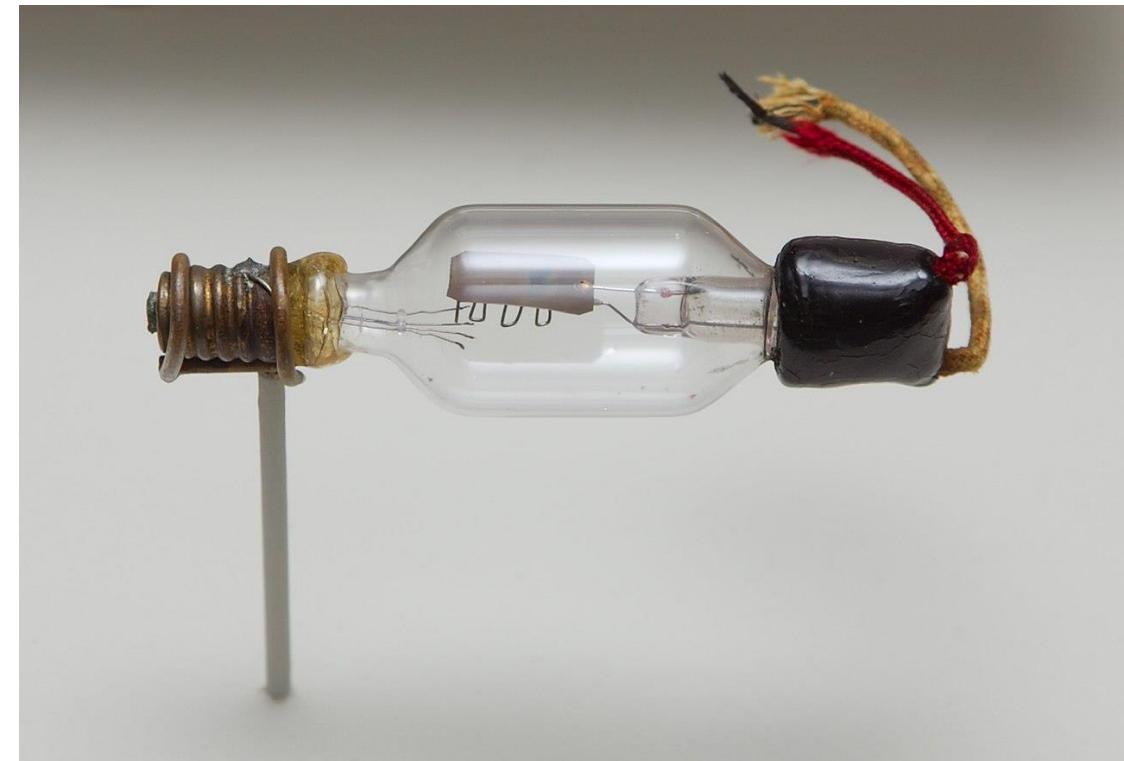
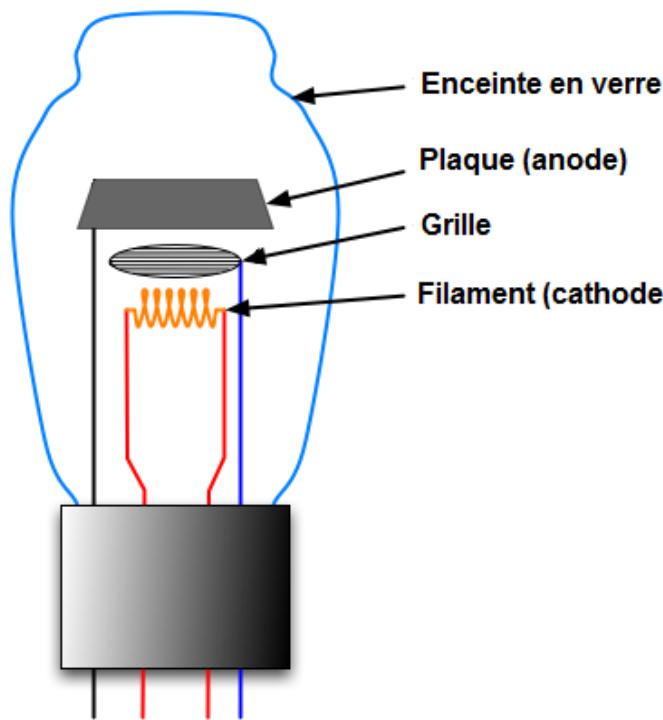


Fonctionnement du processeur



Les ancêtres

- À l'époque des premiers ordinateurs, le transistor n'existait pas, on utilisait un équivalent : le triode, qui ressemble à une grosse ampoule :



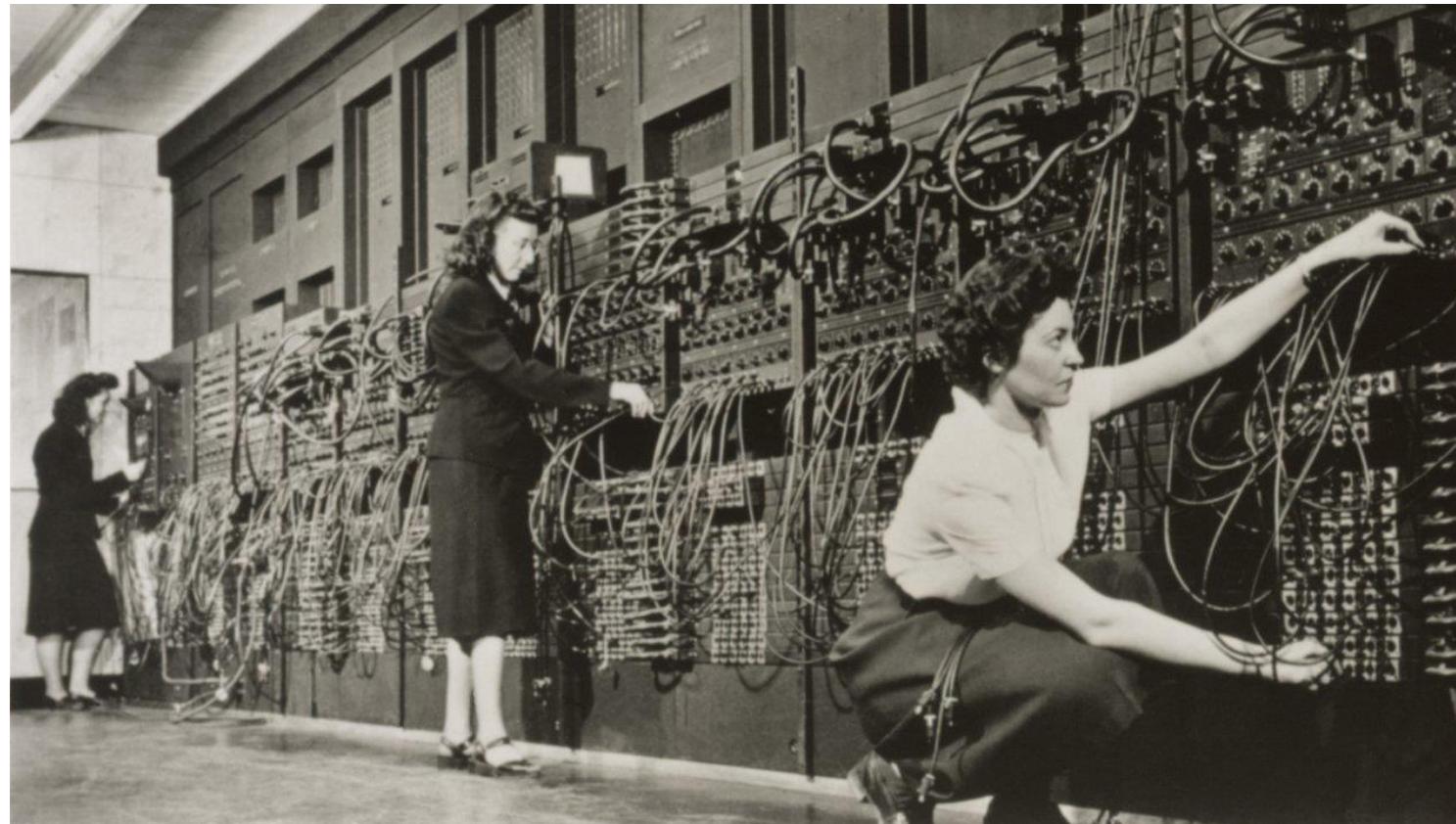
Les ancêtres

Au départ ces systèmes n'étaient pas utilisés pour mettre en œuvre l'algèbre de boole (ce qu'on a vu) mais pour servir d'amplificateur.

- Diode = 1904
- Triode = 1907
- Flip-Flop (base de la mémoire) = 1919
- Additionneur binaire à base de relais (George Stibitz) = 1937

Les anc  tres

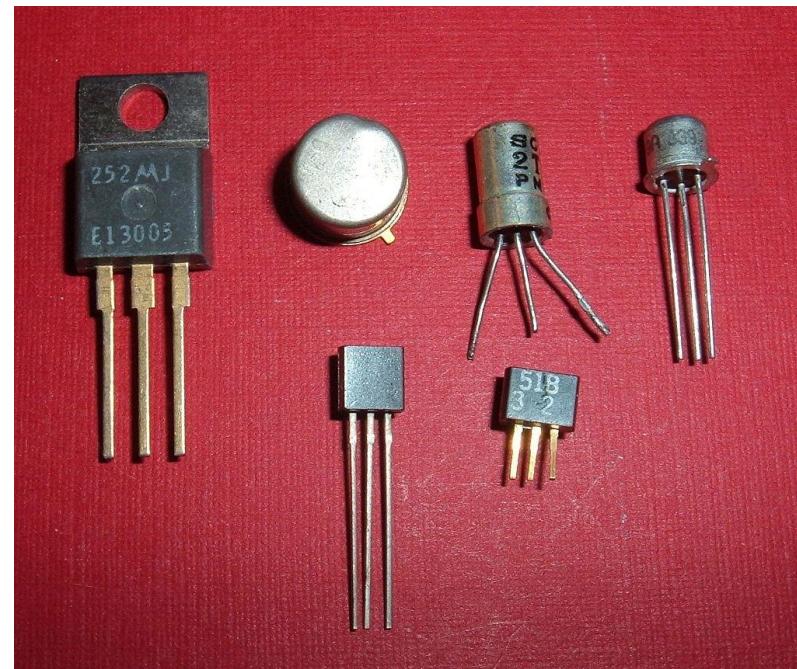
- Cela avait tendance   donner des ordinateurs tr  s volumineux :



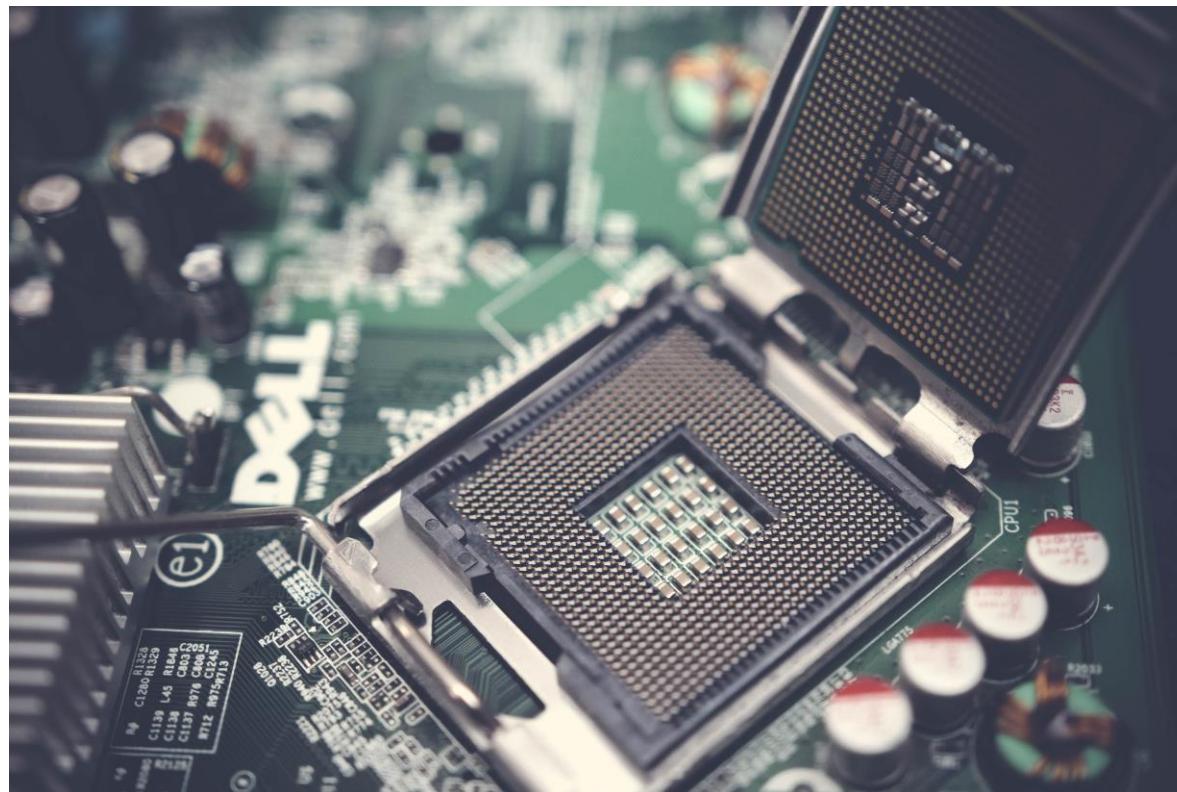
ENIAC (1945)

Suite ...

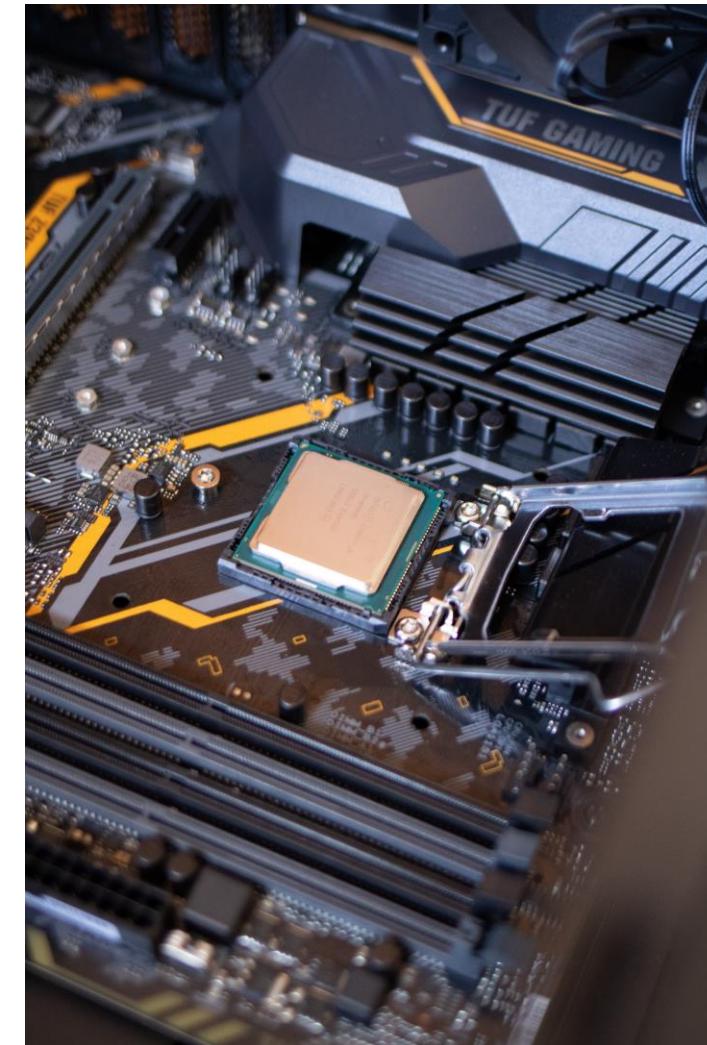
- Par la suite on invente le transistor, et on le miniaturise de plus en plus, permettant l'apparition des ordinateurs d'aujourd'hui.

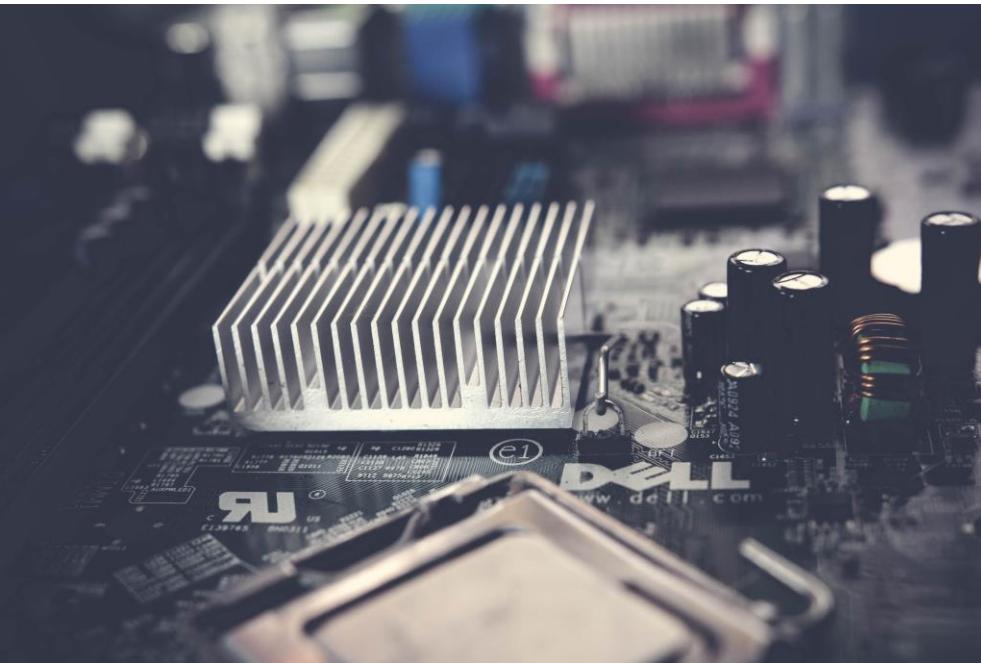


On peut aujourd'hui réaliser des circuits complets de transistors de la taille de 7nm (record actuel), ceci permet d'avoir des **circuits très compact** !

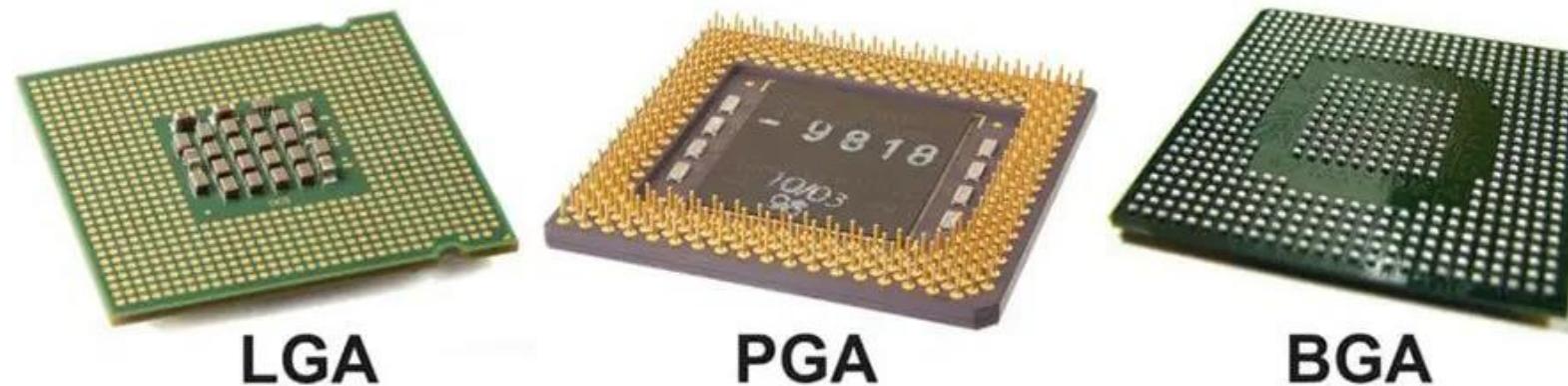


Microprocesseur dans son socket





Microprocesseur sous un dissipateur thermique



Exemples de connectique

LGA

PGA

BGA

Résumé sur le processeur

Résumé sur le processeur

- Constitué de milliers de circuits élémentaires :

Résumé sur le processeur

- Constitué de milliers de circuits élémentaires :
 - Opérateurs

Résumé sur le processeur

- Constitué de milliers de circuits élémentaires :
 - Opérateurs
 - Registres

Résumé sur le processeur

- Constitué de milliers de circuits élémentaires :
 - Opérateurs
 - Registres
- Ces circuits sont :

Résumé sur le processeur

- Constitué de milliers de circuits élémentaires :
 - Opérateurs
 - Registres
- Ces circuits sont :
 - alimentés

Résumé sur le processeur

- Constitué de milliers de circuits élémentaires :
 - Opérateurs
 - Registres
- Ces circuits sont :
 - alimentés
 - Cadencés

Résumé sur le processeur

- Constitué de milliers de circuits élémentaires :
 - Opérateurs
 - Registres
- Ces circuits sont :
 - alimentés
 - Cadencés
- **-> Rôle de l'horloge**

Résumé sur le processeur

- Constitué de milliers de circuits élémentaires :
 - Opérateurs
 - Registres
- Ces circuits sont :
 - alimentés
 - Cadencés
- **-> Rôle de l'horloge**
 - Faire que les calculs soient faits au même moment

Résumé sur le processeur

- Constitué de milliers de circuits élémentaires :
 - Opérateurs
 - Registres
- Ces circuits sont :
 - alimentés
 - Cadencés
- **-> Rôle de l'horloge**
 - Faire que les calculs soient faits au même moment
 - Gère le nombre d'opérations par seconde (la cadence)

Résumé sur le processeur

- Constitué de milliers de circuits élémentaires :
 - Opérateurs
 - Registres
- Ces circuits sont :
 - alimentés
 - Cadencés
- **-> Rôle de l'horloge**
 - Faire que les calculs soient faits au même moment
 - Gère le nombre d'opérations par seconde (la cadence)
 - surcadénçage / surcadencement / overclocking

Résumé sur le processeur

Résumé sur le processeur

- Quelques composants du microprocesseur :

Résumé sur le processeur

- Quelques composants du microprocesseur :
 - UAL : Unité Arithmétique et Logique : effectue les opérations de calculs / logiques de base

Résumé sur le processeur

- Quelques composants du microprocesseur :
 - UAL : Unité Arithmétique et Logique : effectue les opérations de calculs / logiques de base
 - L'accumulateur : registre spécialisé pour recueillir les résultats de l'UAL

Résumé sur le processeur

- Quelques composants du microprocesseur :
 - UAL : Unité Arithmétique et Logique : effectue les opérations de calculs / logiques de base
 - L'accumulateur : registre spécialisé pour recueillir les résultats de l'UAL
 - Pointeur de pile (renseignement sur la position d'un programme en mémoire)

Résumé sur le processeur

- Quelques composants du microprocesseur :
 - UAL : Unité Arithmétique et Logique : effectue les opérations de calculs / logiques de base
 - L'accumulateur : registre spécialisé pour recueillir les résultats de l'UAL
 - Pointeur de pile (renseignement sur la position d'un programme en mémoire)
 - Bus de données : conducteurs destinés au transfert de données et d'instructions

Résumé sur le processeur

- Quelques composants du microprocesseur :
 - UAL : Unité Arithmétique et Logique : effectue les opérations de calculs / logiques de base
 - L'accumulateur : registre spécialisé pour recueillir les résultats de l'UAL
 - Pointeur de pile (renseignement sur la position d'un programme en mémoire)
 - Bus de données : conducteurs destinés au transfert de données et d'instructions
 - Bus d'adresses : transporte les adresses mémoires auxquelles le processeur souhaite accéder

Résumé sur le processeur

- Quelques composants du microprocesseur :
 - UAL : Unité Arithmétique et Logique : effectue les opérations de calculs / logiques de base
 - L'accumulateur : registre spécialisé pour recueillir les résultats de l'UAL
 - Pointeur de pile (renseignement sur la position d'un programme en mémoire)
 - Bus de données : conducteurs destinés au transfert de données et d'instructions
 - Bus d'adresses : transporte les adresses mémoires auxquelles le processeur souhaite accéder
 - ...

Résumé sur le processeur

- Quelques composants du microprocesseur :
 - UAL : Unité Arithmétique et Logique : effectue les opérations de calculs / logiques de base
 - L'accumulateur : registre spécialisé pour recueillir les résultats de l'UAL
 - Pointeur de pile (renseignement sur la position d'un programme en mémoire)
 - Bus de données : conducteurs destinés au transfert de données et d'instructions
 - Bus d'adresses : transporte les adresses mémoires auxquelles le processeur souhaite accéder
 - ...
 - Unité de calcul en virgule flottante, ...

Carte mère

Carte mère

- La carte mère porte les composants nécessaires au fonctionnement de votre ordinateur:

Carte mère

- La carte mère porte les composants nécessaires au fonctionnement de votre ordinateur:
 - Processeur

Carte mère

- La carte mère porte les composants nécessaires au fonctionnement de votre ordinateur:
 - Processeur
 - RAM

Carte mère

- La carte mère porte les composants nécessaires au fonctionnement de votre ordinateur:
 - Processeur
 - RAM
 - Mémoire cache, mémoires mortes

Carte mère

- La carte mère porte les composants nécessaires au fonctionnement de votre ordinateur:
 - Processeur
 - RAM
 - Mémoire cache, mémoires mortes
 - Liens entre les composants (chipset notamment)

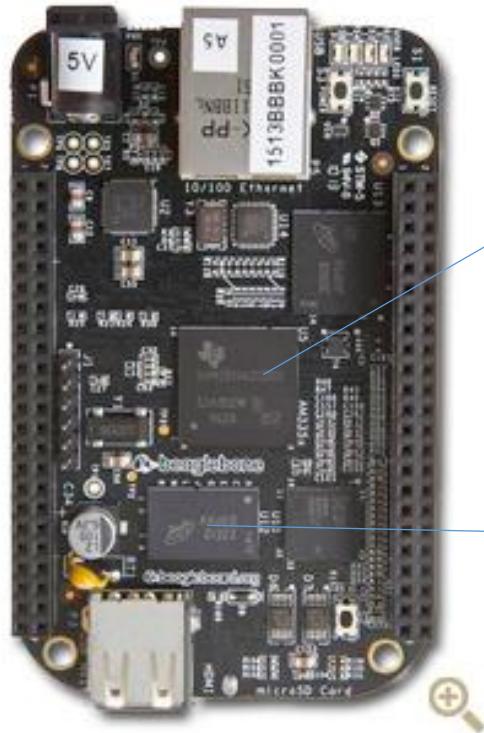
Carte mère

- La carte mère porte les composants nécessaires au fonctionnement de votre ordinateur:
 - Processeur
 - RAM
 - Mémoire cache, mémoires mortes
 - Liens entre les composants (chipset notamment)
- Elle accueille des emplacement pour accueillir des périphériques, de la RAM ou une carte graphique

Carte mère

- La carte mère porte les composants nécessaires au fonctionnement de votre ordinateur:
 - Processeur
 - RAM
 - Mémoire cache, mémoires mortes
 - Liens entre les composants (chipset notamment)
- Elle accueille des emplacement pour accueillir des périphériques, de la RAM ou une carte graphique
- **C'est elle qui fait le lien entre le calculateur principal (le processeur) et le reste des parties spécialisées de l'ordinateur.**

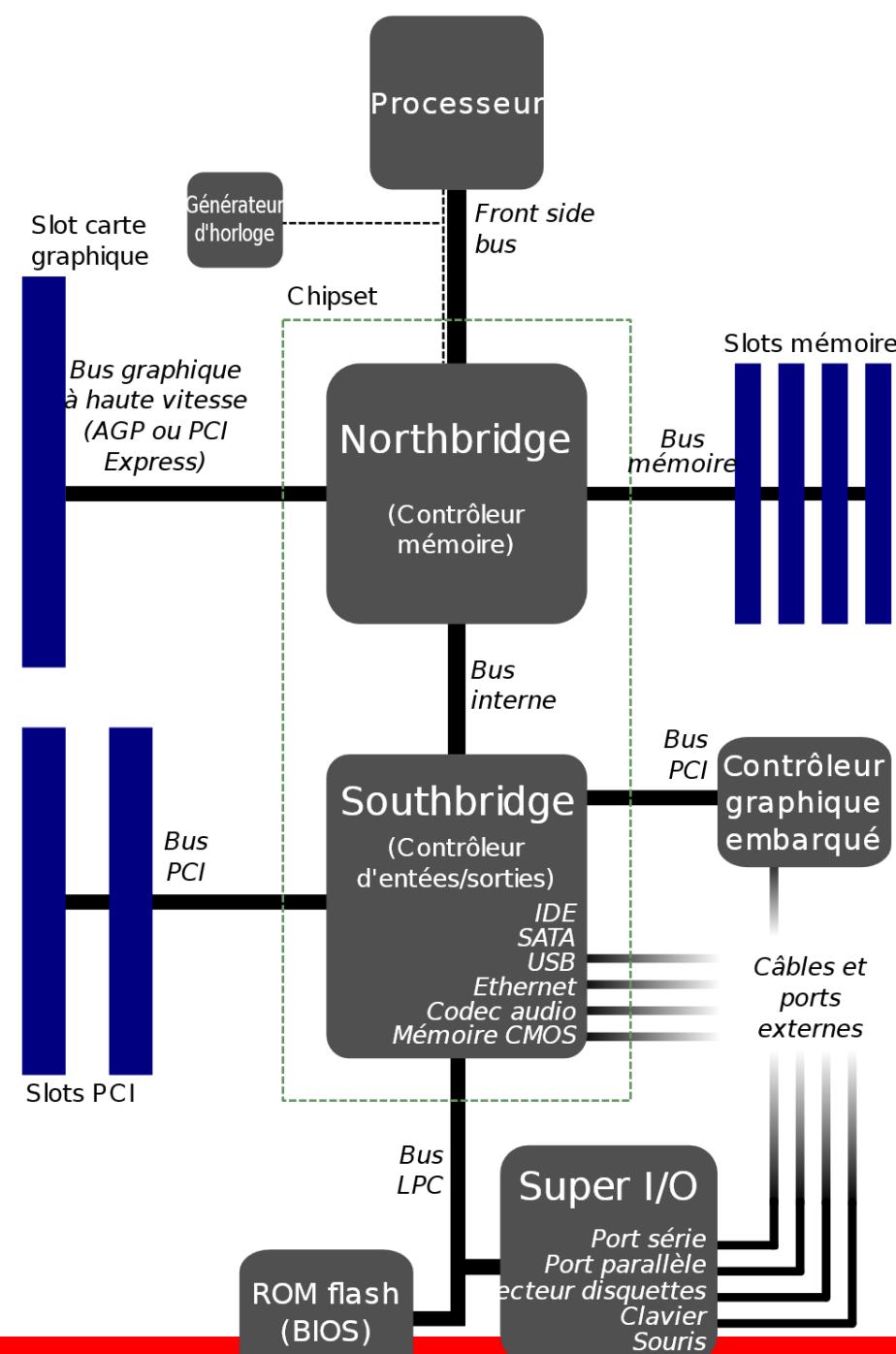
Dans l'ordinateur...



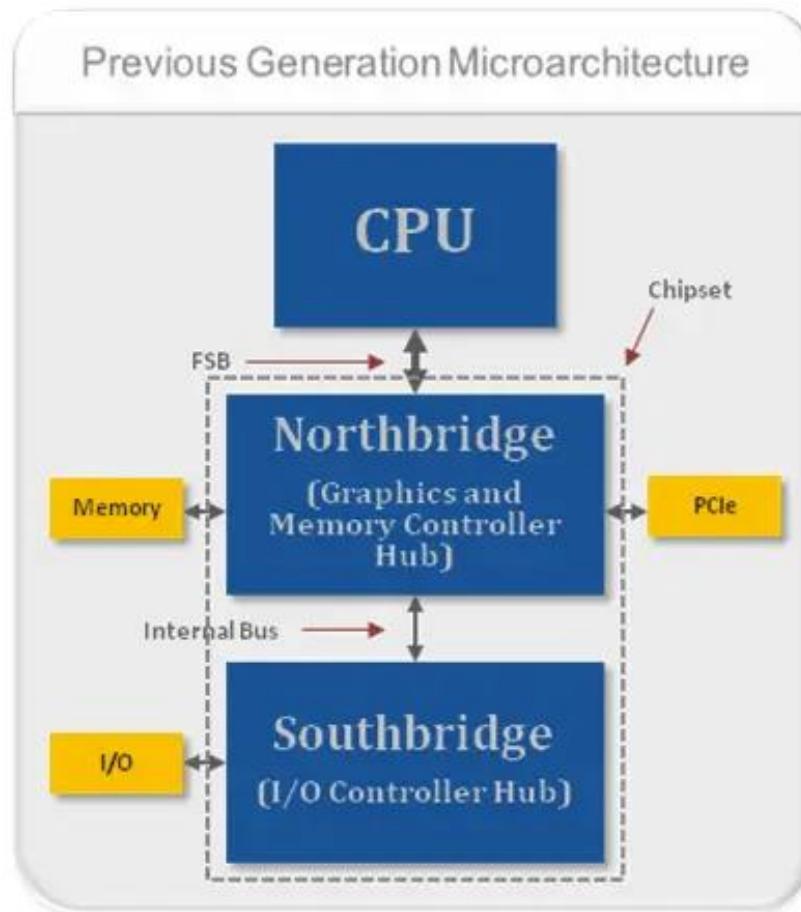
Une carte BeagleBone



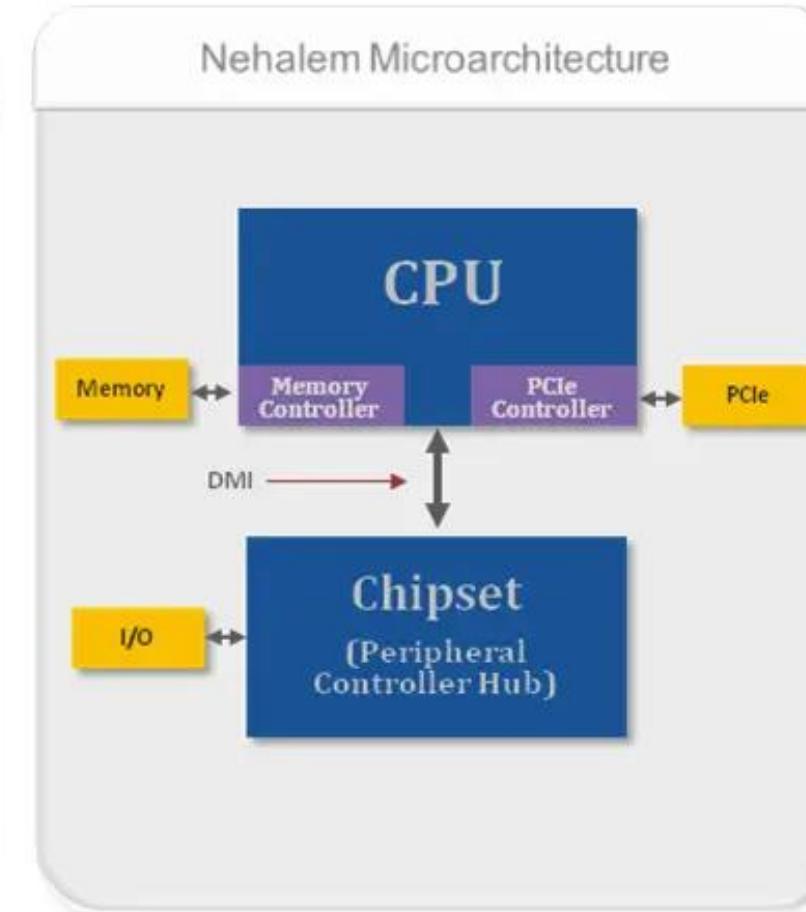
Votre PC



Une intégration dans le CPU ...



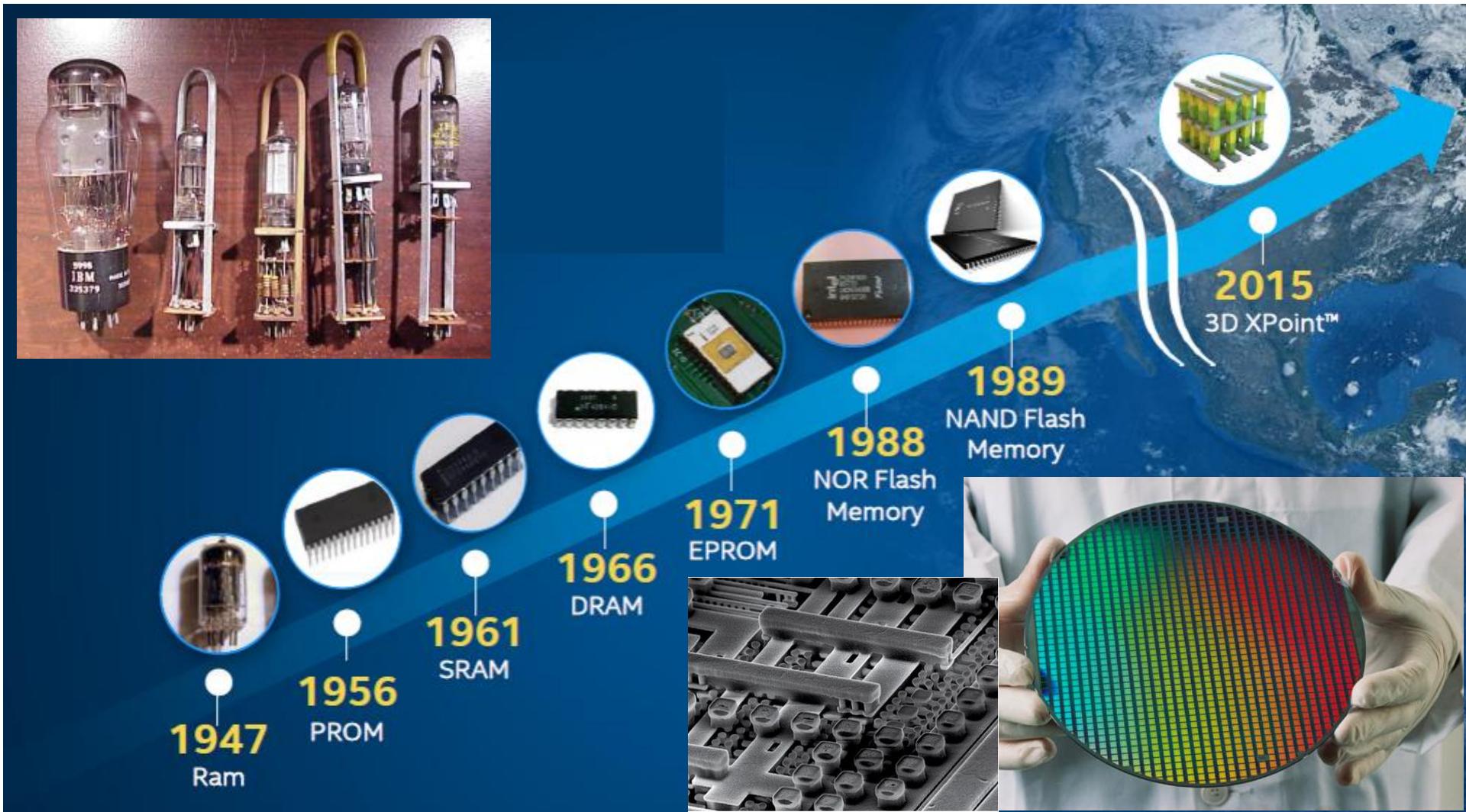
Ancien principe



Nouveau principe

Mémoire

Evolution des technologies de mémoire



Mémoire : généralités

Qu'une donnée soit stockée en mémoire par un courant électrique, par une polarité sur une bande magnétique (ou un trou dans un support physique), le fonctionnement est le même sur tous les ordinateurs :

Mémoire : généralités

Qu'une donnée soit stockée en mémoire par un courant électrique, par une polarité sur une bande magnétique (ou un trou dans un support physique), le fonctionnement est le même sur tous les ordinateurs :

- La présence d'une information dans une case mémoire est représentée par un 1, son absence par un 0. Il s'agit d'un bit.

Mémoire : généralités

Qu'une donnée soit stockée en mémoire par un courant électrique, par une polarité sur une bande magnétique (ou un trou dans un support physique), le fonctionnement est le même sur tous les ordinateurs :

- La présence d'une information dans une case mémoire est représentée par un 1, son absence par un 0. Il s'agit d'un bit.
- Les données sont regroupées par groupes de 8 pour être lues par les composants de l'ordinateur. 8 bits sont nommés un octet. (Attention, en anglais on parle de **byte** (différent de bit !))

Mémoire : généralités

Qu'une donnée soit stockée en mémoire par un courant électrique, par une polarité sur une bande magnétique (ou un trou dans un support physique), le fonctionnement est le même sur tous les ordinateurs :

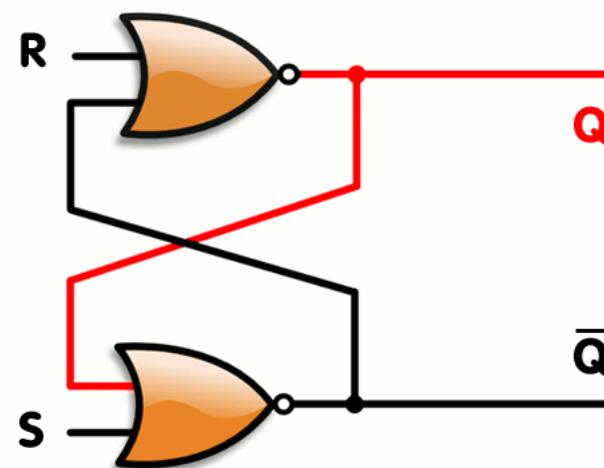
- La présence d'une information dans une case mémoire est représentée par un 1, son absence par un 0. Il s'agit d'un bit.
- Les données sont regroupées par groupes de 8 pour être lues par les composants de l'ordinateur. 8 bits sont nommés un octet. (Attention, en anglais on parle de **byte** (différent de bit !))
 - Exemple :
 - les bus actuels transportent 64 bits (ou 8 octets)
 -  : Cet octet vaut 90, ou représente le caractère Z

Mémoire : principe

- Il existe différents types de mémoires utilisant différents principes, selon les usages :
 - Registre
 - Mémoire vive de type RAM
 - Mémoire morte non effaçable
 - Mémoire morte effaçable
 - Bande magnétique
 - Disque dur
 - CD/DVD/Blu-ray
 - SSD

Mémoire vive : principe

- Il existe plusieurs méthodes pour retenir de l'information binaires
- Exemple : le Flip-Flop (ou bascule)



| S | R | Q | \bar{Q} | remarque |
|---|---|-----|-----------|---------------|
| 0 | 0 | q | \bar{q} | mémoire |
| 0 | 1 | 0 | 1 | mise à 0 |
| 1 | 0 | 1 | 0 | mise à 1 |
| 1 | 1 | 0 | 0 | état interdit |

Mémoire vive

Mémoire vive

- Dans le schéma de fonctionnement du processeur, la mémoire présentée est la RAM (Random Access Memory) : c'est le type de mémoire le plus classique dans un PC.

Mémoire vive

- Dans le schéma de fonctionnement du processeur, la mémoire présentée est la RAM (Random Access Memory) : c'est le type de mémoire le plus classique dans un PC.
- Elle est basée sur le principe du flip-flop présenté précédemment.

Mémoire vive

- Dans le schéma de fonctionnement du processeur, la mémoire présentée est la RAM (Random Access Memory) : c'est le type de mémoire le plus classique dans un PC.
- Elle est basée sur le principe du flip-flop présenté précédemment.
- Caractéristiques :

Mémoire vive

- Dans le schéma de fonctionnement du processeur, la mémoire présentée est la RAM (Random Access Memory) : c'est le type de mémoire le plus classique dans un PC.
- Elle est basée sur le principe du flip-flop présentée précédemment.
- Caractéristiques :
 - Elle est plutôt rapide d'accès (plus que les systèmes de stockage)

Mémoire vive

- Dans le schéma de fonctionnement du processeur, la mémoire présentée est la RAM (Random Access Memory) : c'est le type de mémoire le plus classique dans un PC.
- Elle est basée sur le principe du flip-flop présenté précédemment.
- Caractéristiques :
 - Elle est plutôt rapide d'accès (plus que les systèmes de stockage)
 - Elle peut être réécrite à volonté

Mémoire vive

- Dans le schéma de fonctionnement du processeur, la mémoire présentée est la RAM (Random Access Memory) : c'est le type de mémoire le plus classique dans un PC.
- Elle est basée sur le principe du flip-flop présentée précédemment.
- Caractéristiques :
 - Elle est plutôt rapide d'accès (plus que les systèmes de stockage)
 - Elle peut être réécrite à volonté
 - **Elle est volatile** (s'efface lorsque l'ordinateur reboote) (d'où le « vive »).

Mémoire vive

- Dans le schéma de fonctionnement du processeur, la mémoire présentée est la RAM (Random Access Memory) : c'est le type de mémoire le plus classique dans un PC.
- Elle est basée sur le principe du flip-flop présenté précédemment.
- Caractéristiques :
 - Elle est plutôt rapide d'accès (plus que les systèmes de stockage)
 - Elle peut être réécrite à volonté
 - **Elle est volatile** (s'efface lorsque l'ordinateur reboote) (d'où le « vive »).
 - Elle a une taille limite (sur les machines récentes > 8 Go)



RAM DDR4 sans et avec dissipateur thermique

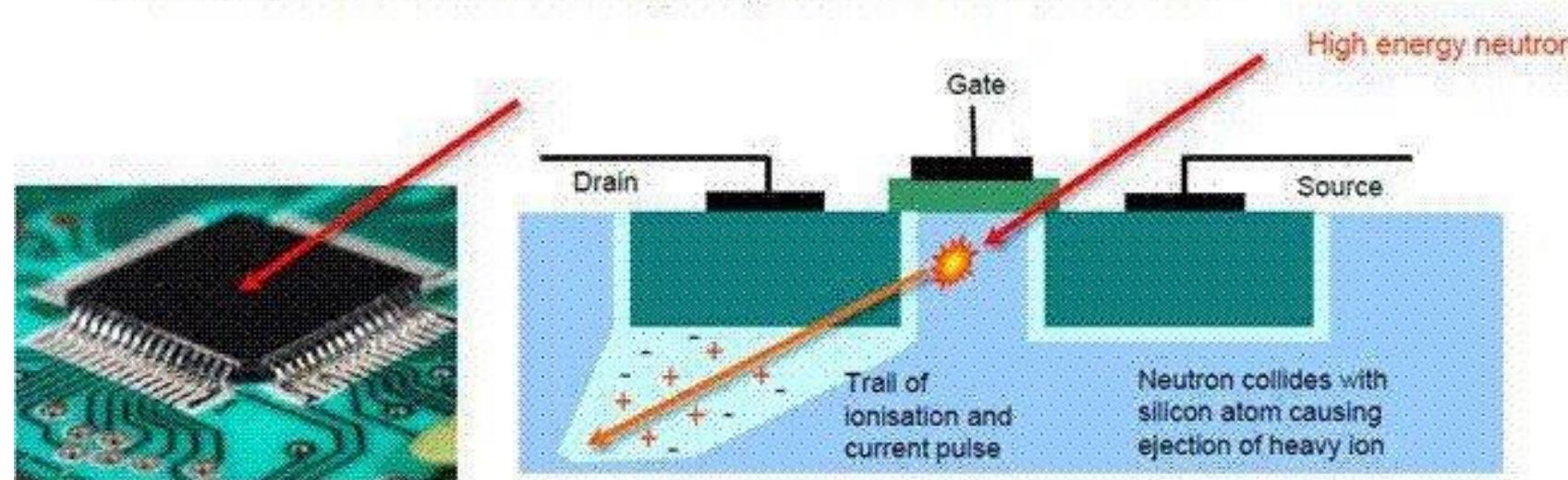
Mémoire vive

- Pour éviter les erreurs (rayon cosmiques par exemple, venant changer un bit au hasard) on peut lui adjoindre une puce supplémentaire qui va permettre de vérifier par un calcul que chaque bit est OK et de le rétablir si besoin.

Mémoire vive

- Pour éviter les erreurs (rayon cosmiques par exemple, venant changer un bit au hasard) on peut lui adjoindre une puce supplémentaire qui va permettre de vérifier par un calcul que chaque bit est OK et de le rétablir si besoin.

A Single Event Effect (SEE) is when a highly energetic particle (neutron), present in the environment, strikes sensitive regions of an electronic device disrupting its correct operation

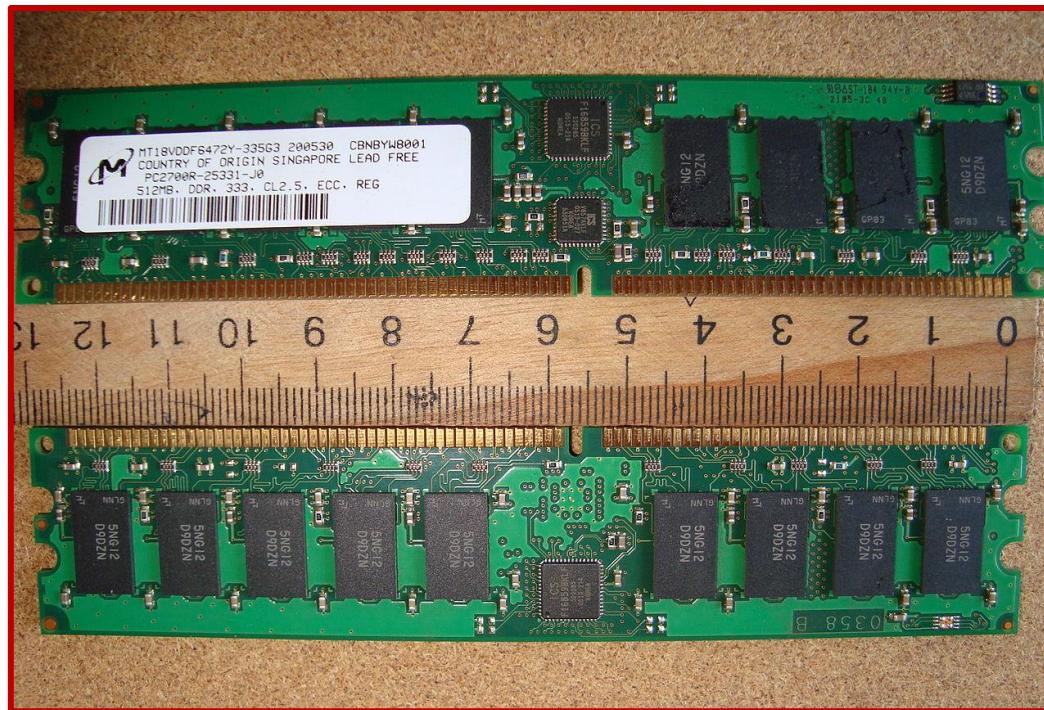


Mémoire vive

- Pour éviter les erreurs (rayon cosmiques par exemple, venant changer un bit au hasard) on peut lui adjoindre une puce supplémentaire qui va permettre de vérifier par un calcul que chaque bit est OK et de le rétablir si besoin.

Mémoire vive

- Pour éviter les erreurs (rayon cosmiques par exemple, venant changer un bit au hasard) on peut lui adjoindre une puce supplémentaire qui va permettre de vérifier par un calcul que chaque bit est OK et de le rétablir si besoin.



On parle de mémoire
ram « ECC »

Mémoire morte

M  moire morte

- Historiquement : m  moire non effa  ble

Mémoire morte

- Historiquement : mémoire non effaçable
- Intérêt : garder des données / programmes en permanence sur la machine.

Mémoire morte

- Historiquement : mémoire non effaçable
- Intérêt : garder des données / programmes en permanence sur la machine.
 - Exemple : le BIOS, les clés USB etc.

Mémoire morte

- Historiquement : mémoire non effaçable
- Intérêt : garder des données / programmes en permanence sur la machine.
 - Exemple : le BIOS, les clés USB etc.
- Les plus simples de ces mémoires sont non réinscriptibles et non programmable : ROM (read only memory)

Mémoire morte

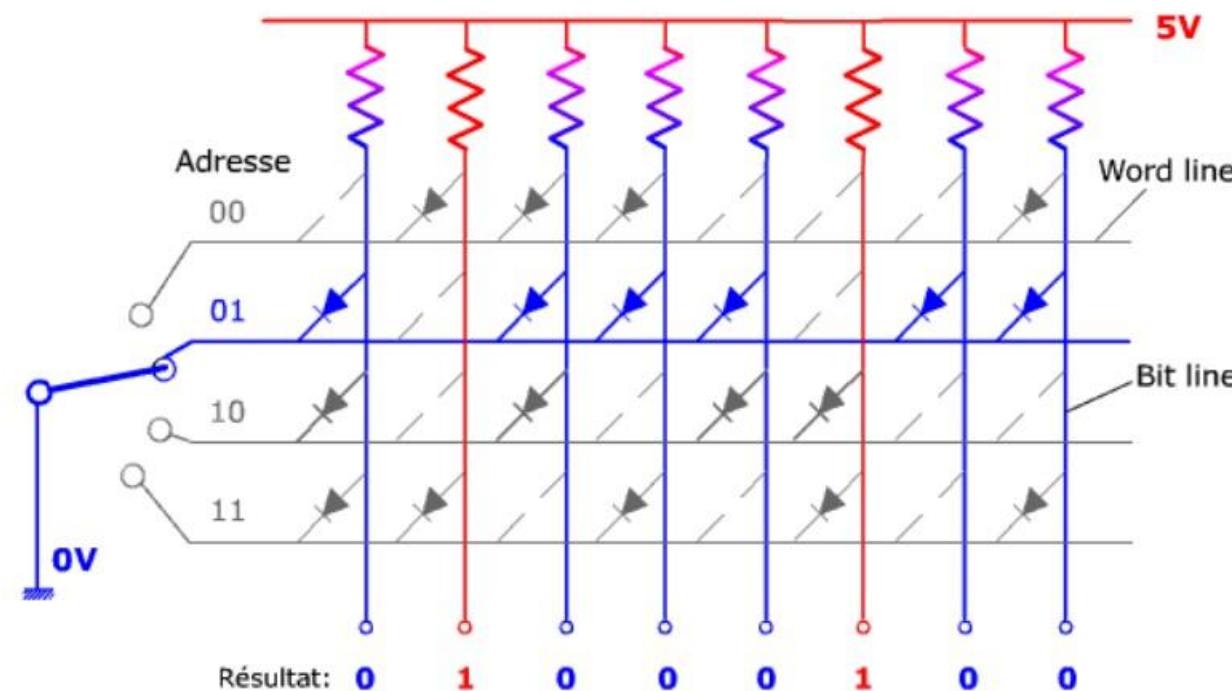
- Historiquement : mémoire non effaçable
- Intérêt : garder des données / programmes en permanence sur la machine.
 - Exemple : le BIOS, les clés USB etc.
- Les plus simples de ces mémoires sont non réinscriptibles et non programmable : ROM (read only memory)
- Les PROM sont programmable, mais une seule fois.

Mémoire morte

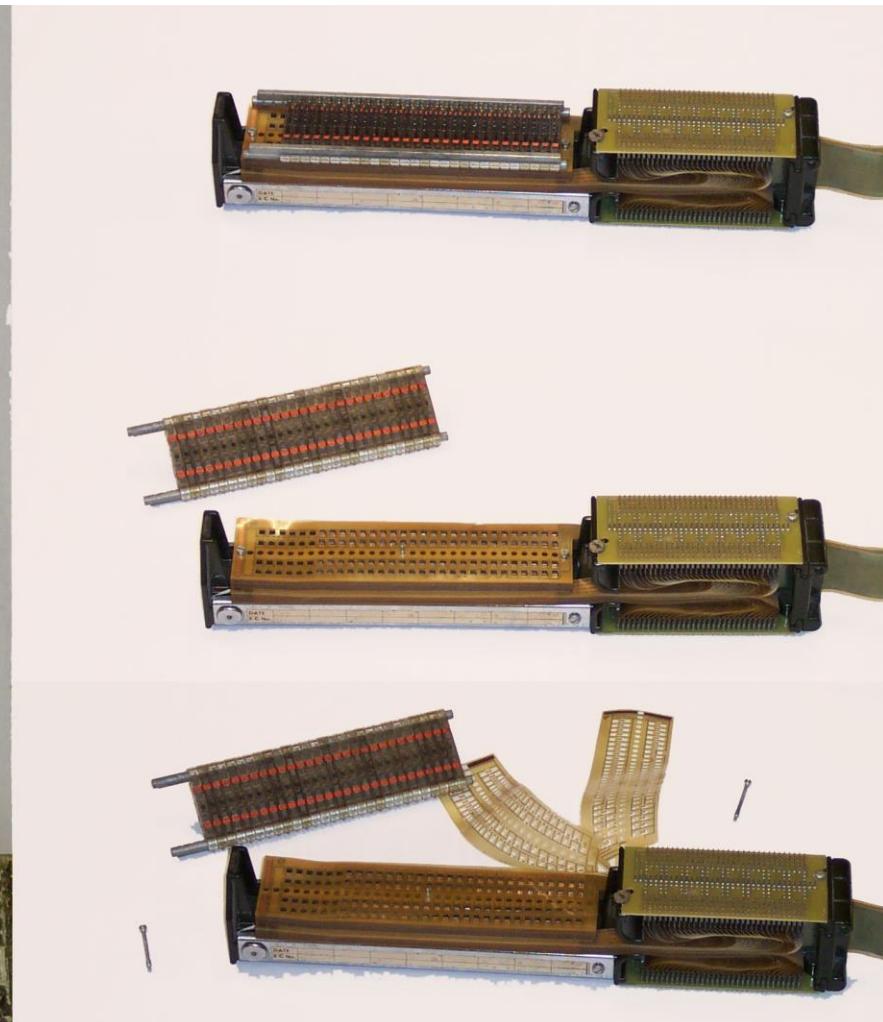
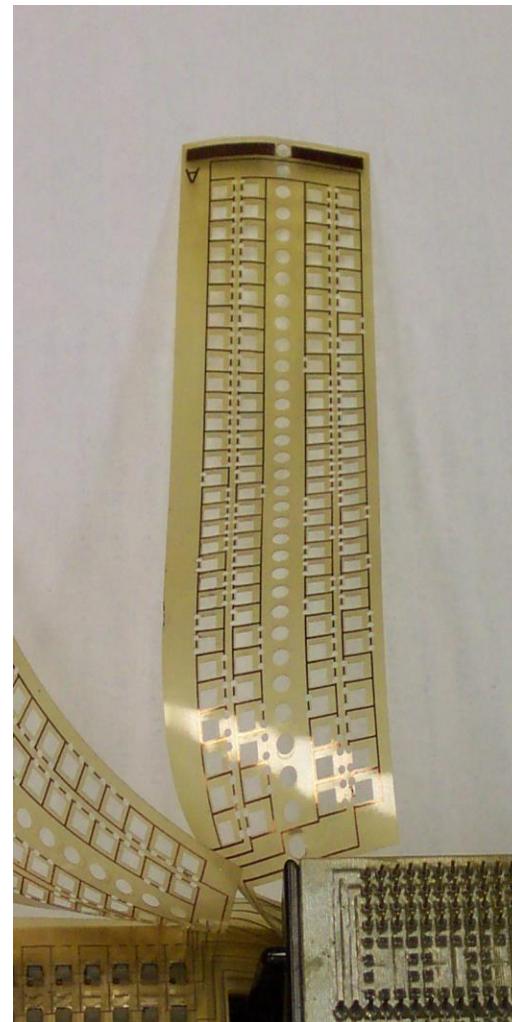
- Historiquement : mémoire non effaçable
- Intérêt : garder des données / programmes en permanence sur la machine.
 - Exemple : le BIOS, les clés USB etc.
- Les plus simples de ces mémoires sont non réinscriptibles et non programmable : ROM (read only memory)
- Les PROM sont programmable, mais une seule fois.
- Les mémoires mortes aujourd’hui sont réinscriptibles (EPROM ou EEPROM)

ROM

- Principe très simple : pour une adresse donnée on utilise des diodes qui lie une fois pour toute certaine ligne à cette adresse.



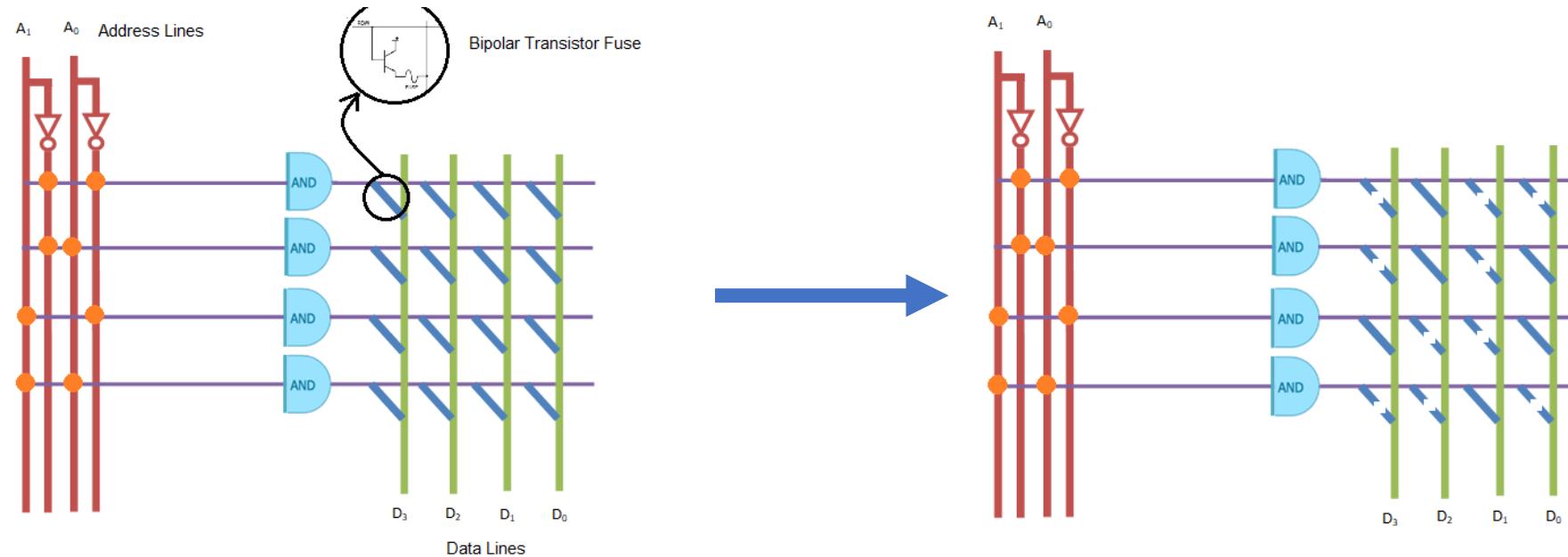
ROM



Un ancien type de rom sous forme de bande.

PROM

- Les PROM sont programmables contrairement aux ROM, mais une seule fois. Le principe est simple, ce sont des fusibles qu'on fait claquer une fois pour toute en appliquant une tension forte, créant ainsi un système par ligne comme précédemment.



PROM

- Les PROM sont programmables contrairement aux ROM, mais une seule fois. Le principe est simple, ce sont des fusibles qu'on fait claquer une fois pour toute en appliquant une tension forte, créant ainsi un système par ligne comme précédemment.



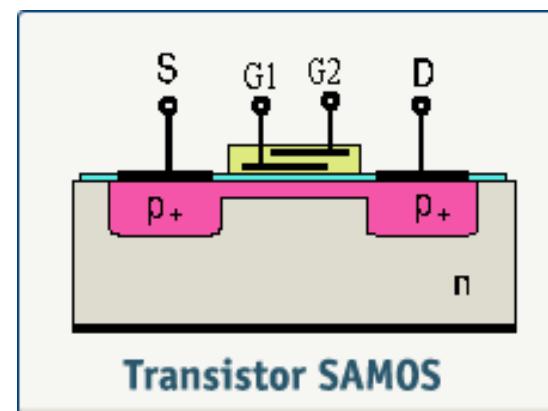
EPROM

EPROM

- Les EPROM sont programmables et réinscriptibles :

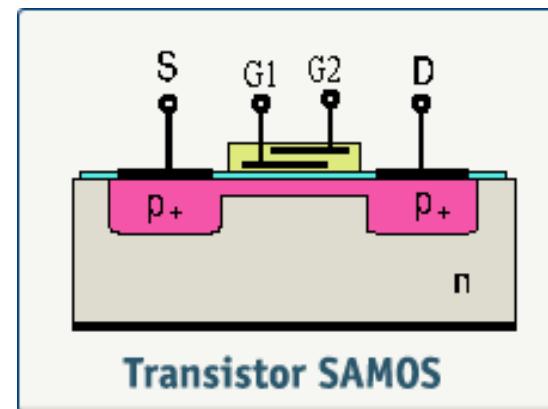
EPROM

- Les EPROM sont programmables et réinscriptibles :
- Le principe est d'utiliser une seconde grille « flottante » dans le transistor :



EPROM

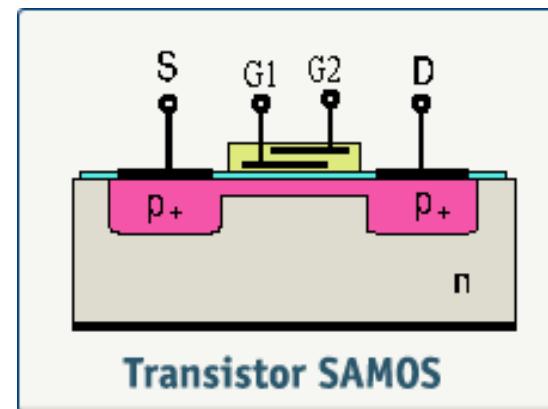
- Les EPROM sont programmables et réinscriptibles :
- Le principe est d'utiliser une seconde grille « flottante » dans le transistor :



- En appliquant une tension suffisamment forte, les électrons viennent se loger sur cette grille flottante qui n'est connectée à rien. Vu qu'elle est isolée de l'extérieur, son potentiel peut se maintenir **des années**.

EPROM

- Les EPROM sont programmables et réinscriptibles :
- Le principe est d'utiliser une seconde grille « flottante » dans le transistor :



- En appliquant une tension suffisamment forte, les électrons viennent se loger sur cette grille flottante qui n'est connectée à rien. Vu qu'elle est isolée de l'extérieur, son potentiel peut se maintenir **des années**.
- La grille flottante va alors agir sur le transistor en privilégiant ou non le passage des électrons (robinet plus ou moins ouvert/fermé).

EEPROM

- Les EPROMs n'étaient effaçable qu'en illuminant une partie du dispositif de lumière ultra-violet. Ce qui n'était pas pratique (et tout était effacé en une fois).
 - Mais elles se reconnaissent aisément vu que le circuit est visible !

EEPROM

- Les EPROMs n'étaient effaçable qu'en illuminant une partie du dispositif de lumière ultra-violet. Ce qui n'était pas pratique (et tout était effacé en une fois).
 - Mais elles se reconnaissent aisément vu que le circuit est visible !



EEPROM

- Les EPROMs n'étaient effaçable qu'en illuminant une partie du dispositif de lumière ultra-violet. Ce qui n'était pas pratique (et tout était effacé en une fois).
 - Mais elles se reconnaissent aisément vu que le circuit est visible !

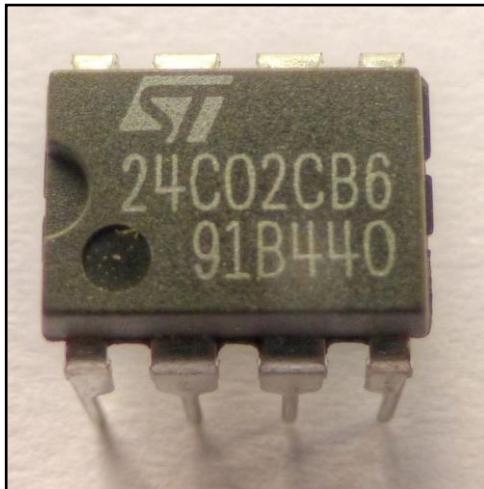
EEPROM

- Les EPROMs n'étaient effaçable qu'en illuminant une partie du dispositif de lumière ultra-violet. Ce qui n'était pas pratique (et tout était effacé en une fois).
 - Mais elles se reconnaissent aisément vu que le circuit est visible !
- En améliorant la technologie les EEPROM (**electrically erasable programmable read-only memory**) permettent l'effacement électrique.
- **C'est devenu la base de ce qu'on appelle la mémoire FLASH !**

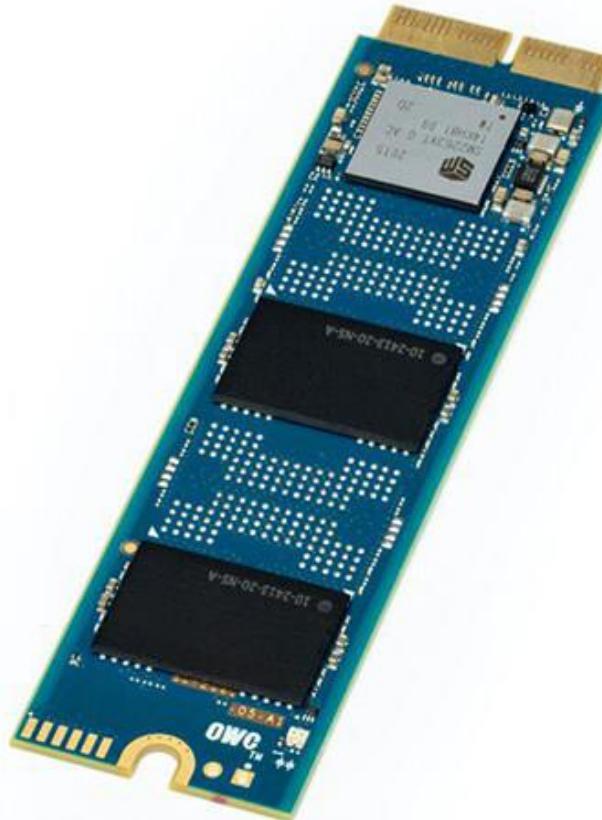
EEPROM et Flash

Ce type de mémoire est utilisée aujourd'hui :

- À la place des ROM : pour stocker des choses qui n'ont pas vocation à changer :
 - Le **Bios** de l'ordinateur par exemple.
- Comme base pour les dispositifs de type **clé USB** ou sur les carte mémoires (photo etc.).
- Comme base pour les dispositifs comme les **disques SSD**



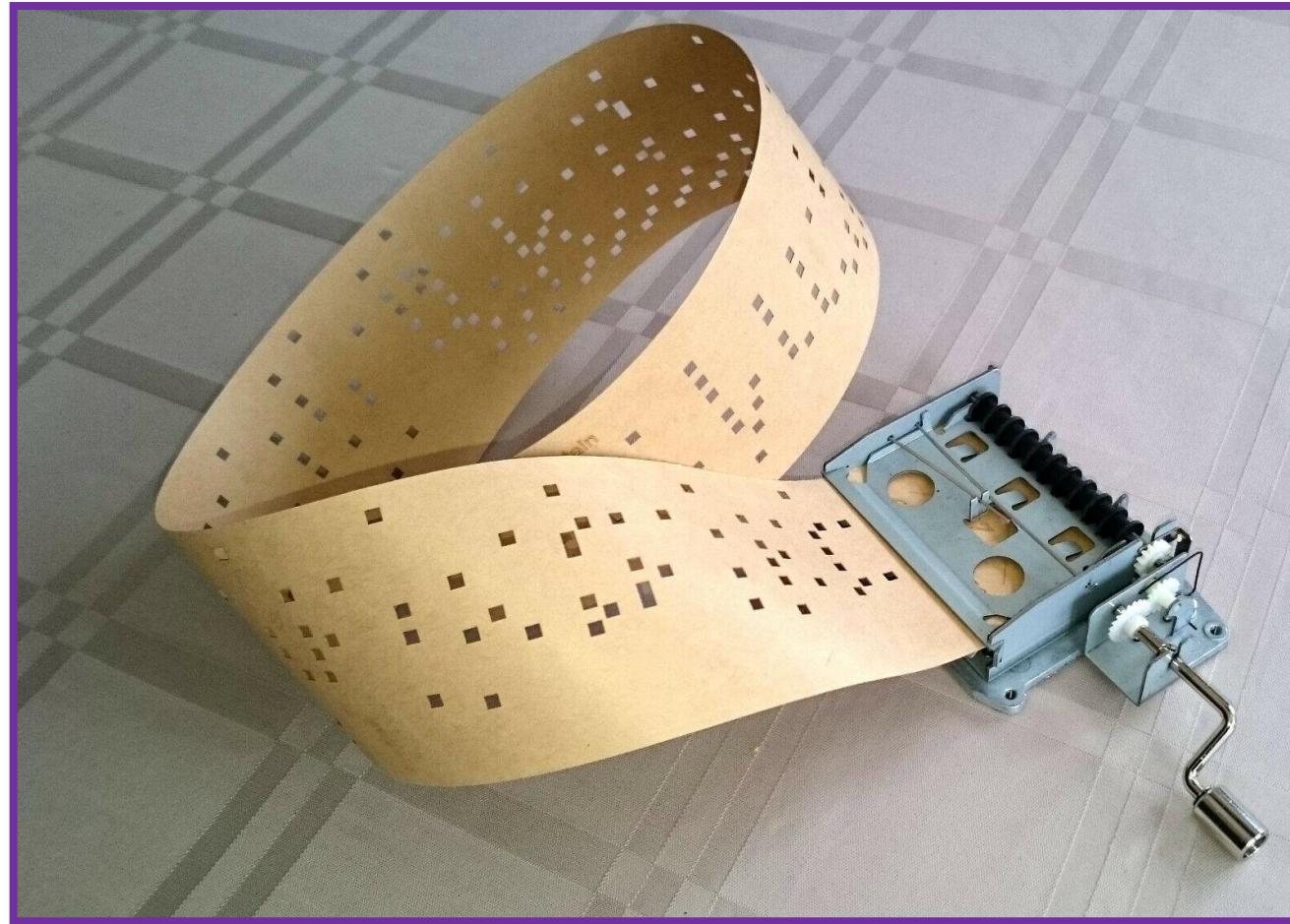
SSD



Systèmes de stockage

- Du fait de sa volatilité et de sa faible capacité, il est nécessaire de disposer de système permettant de stocker de l'information de manière pérenne.
- Les premiers systèmes :
 - Anciennement (mais toujours existant pour certains): bandes magnétiques, disques optiques, cartes perforées, disquettes...

Cartes perforées



Carte pour la musique (boîte à musique, orgue de barbarie etc.)

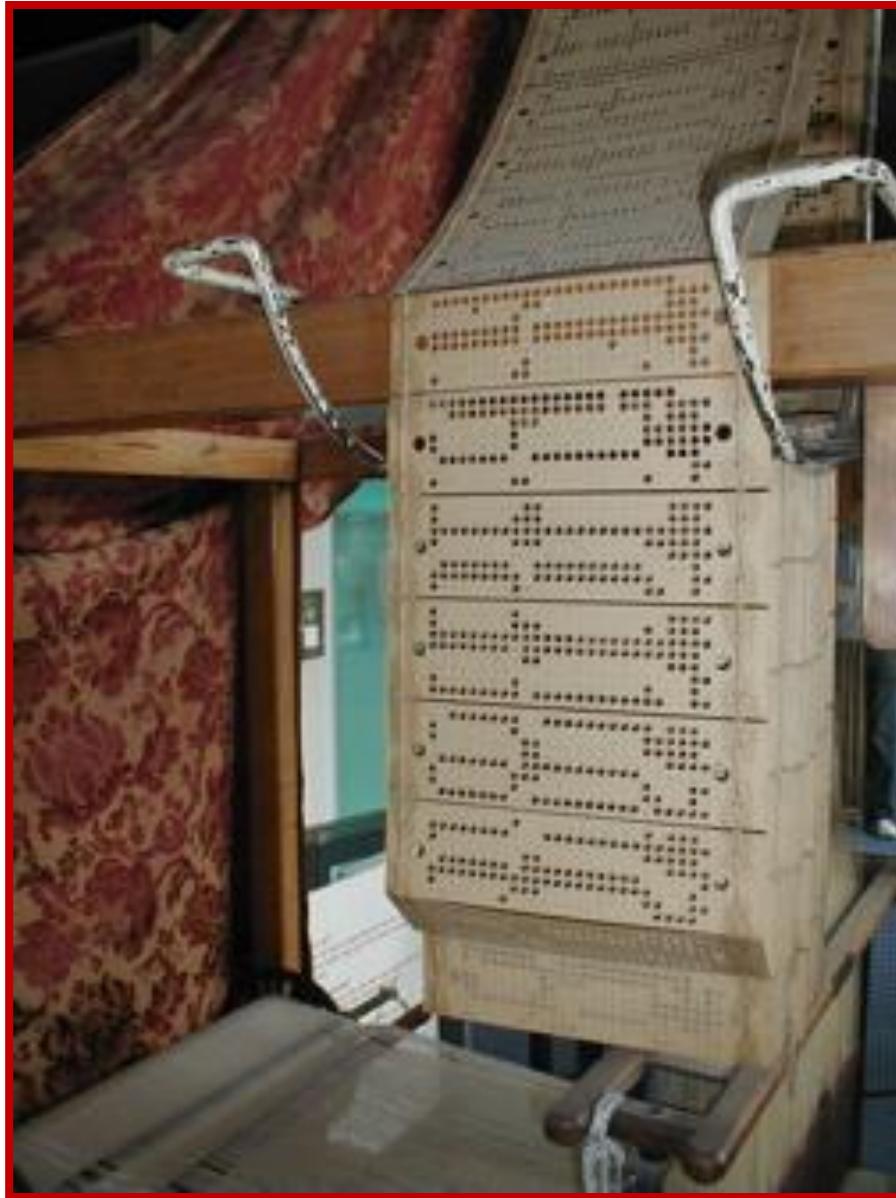
Cartes perforées

Carte pour la musique (boîte à musique, orgue de barbarie etc.)



Carte pour la musique (boite à musique, orgue de barbarie etc.)

Cartes perforées



perforées

Carte perforée pour « programmer »
des métiers à tisser mécaniques.
Métier à Tisser Jacquart : 1745

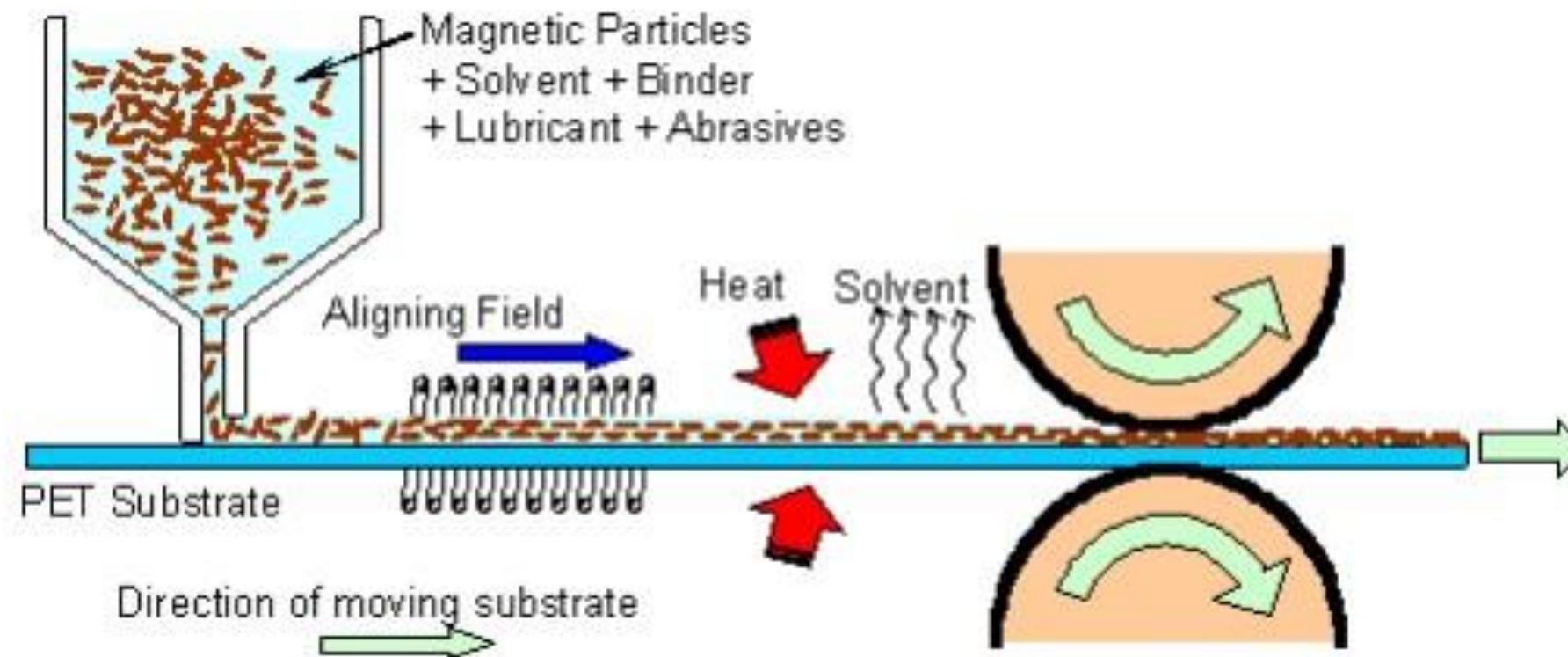
Cartes perforées

Cartes perfor  es



Carte perfor  e pour machine IBM : Lu
avec un syst  me de brosses m  tallique

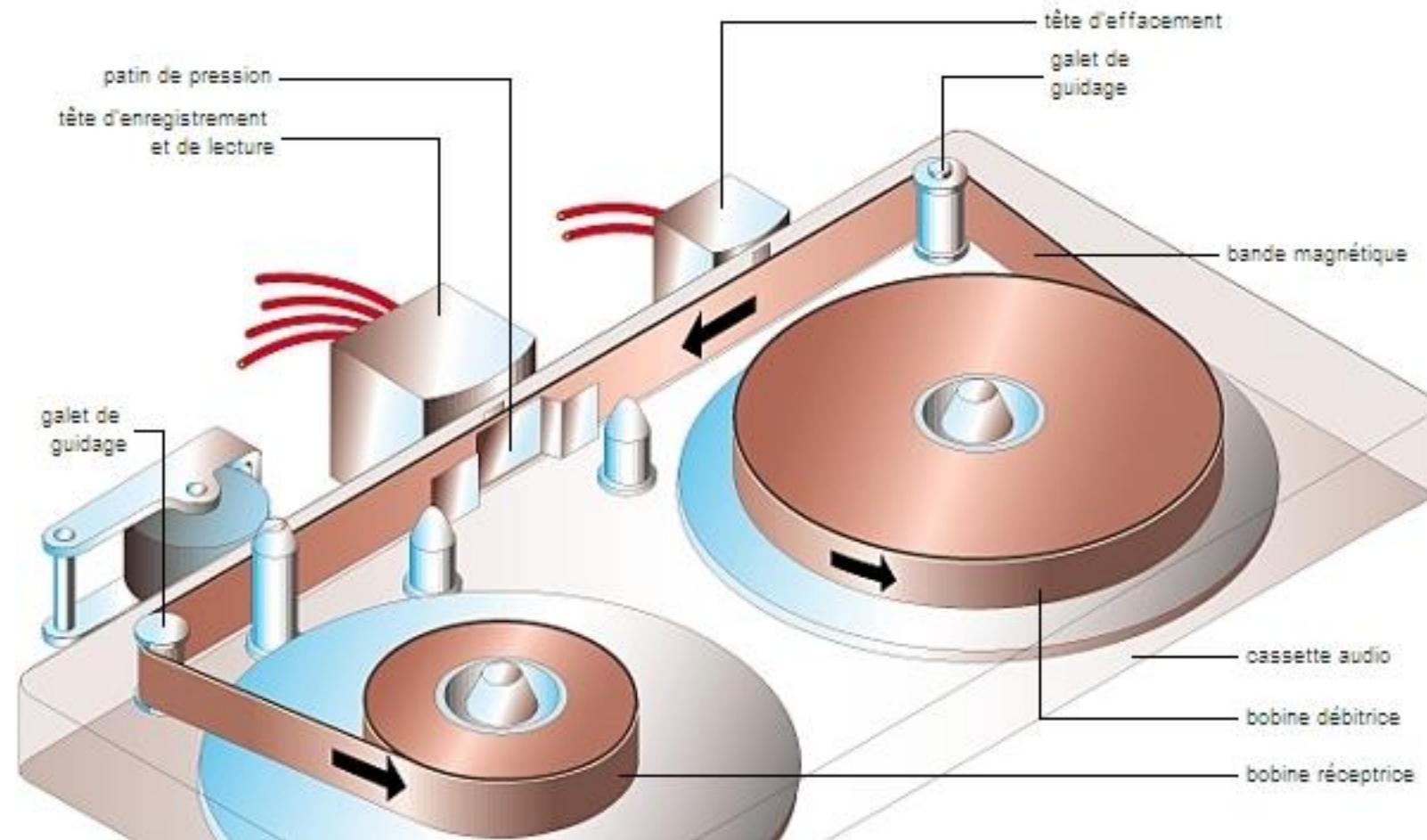
Bande magnétique



Bande magnétique



Bande magnétique



Disque optique

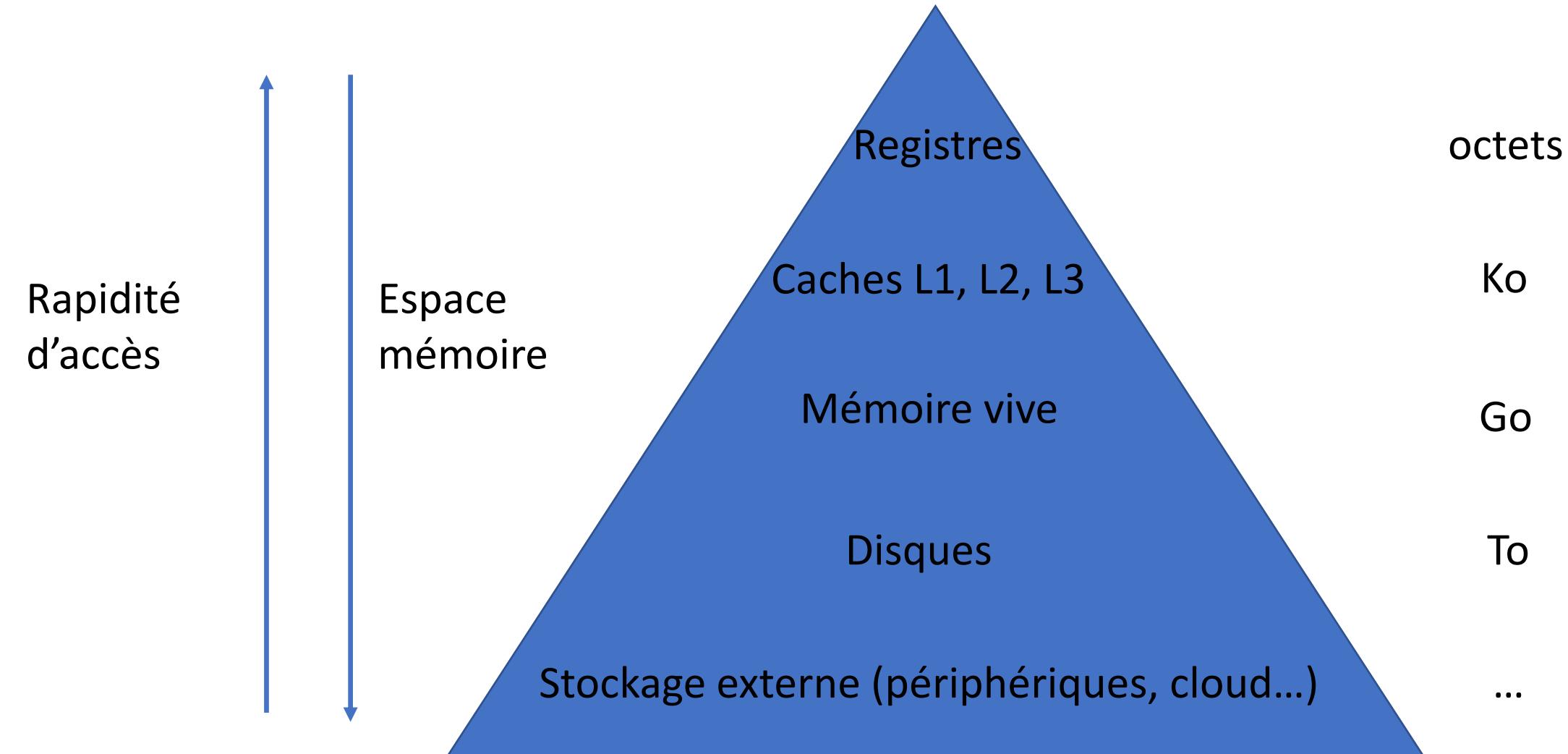
- CD ..
- DVD ...
- BlueRay ...

Disque dur et SSD

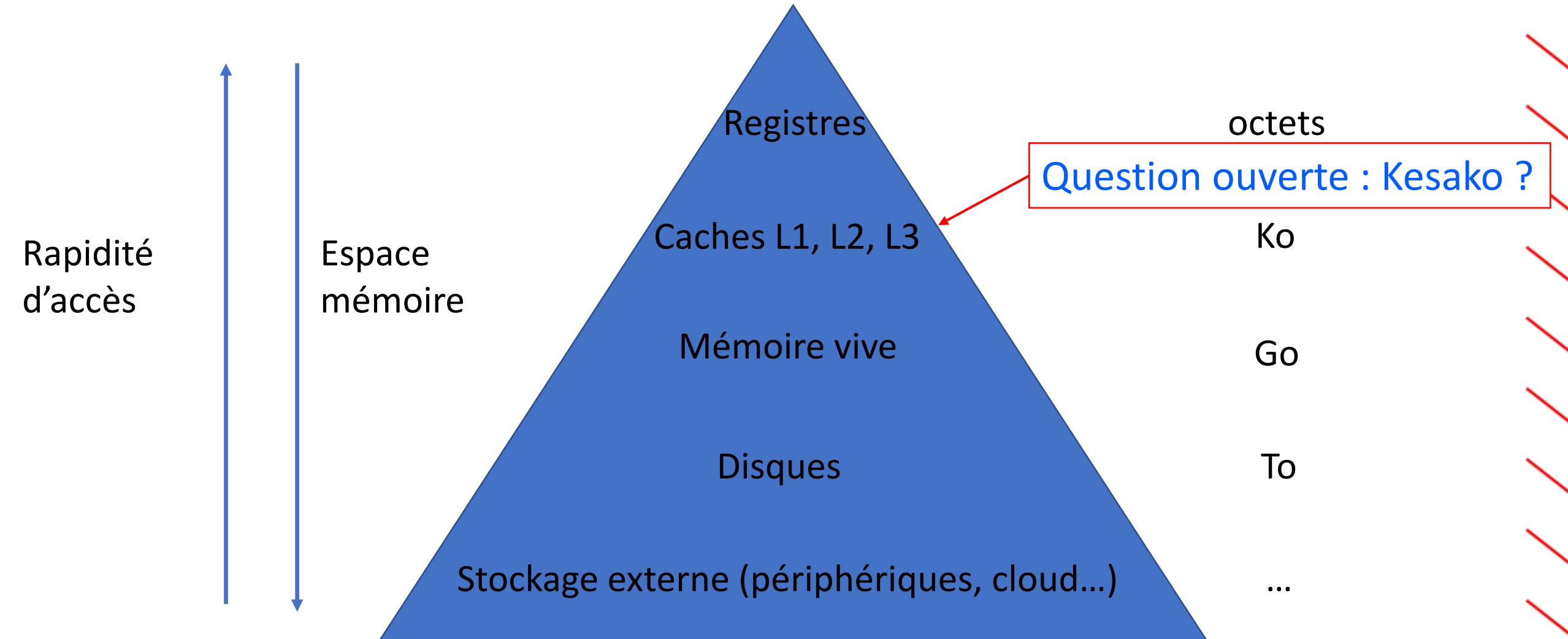
- Dans vos PCs :
 - Disques durs (HDD) empilement de disques magnétiques associées à leur tête de lecture
 - SSD (Solid-State Drive): plus rapide : constitué de mémoires à semi-conducteurs, plus cher



Fonctionnement global de la mémoire



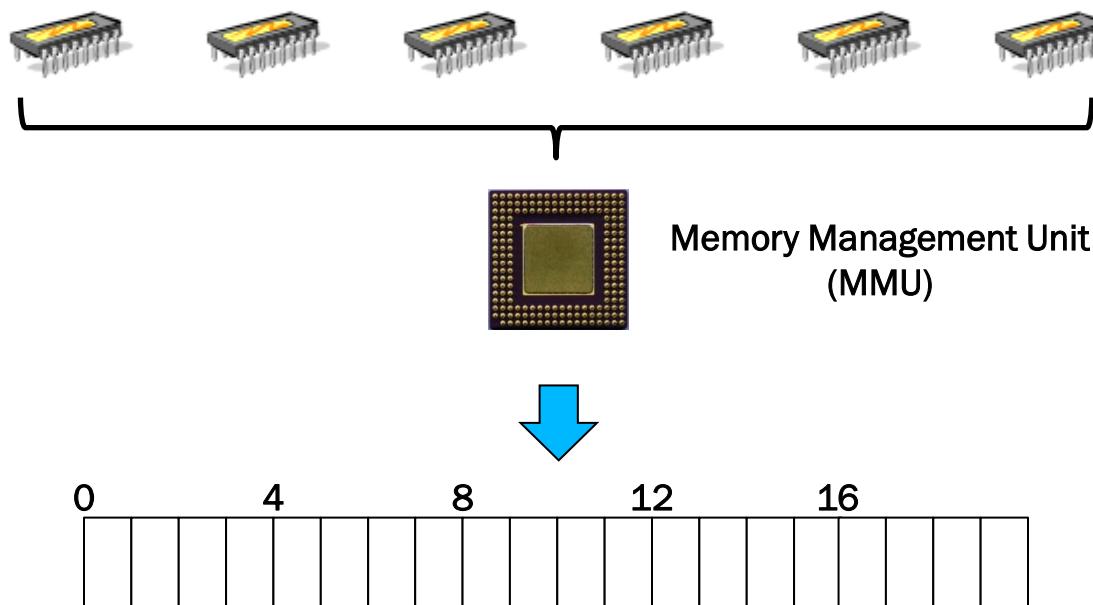
Fonctionnement global de la mémoire



Mémoire matérielle et gestion logicielle

Comment le système voit il la mémoire ?

- Vue par le système (OS/application) via des adresses logicielles : **en clair, un numéro sur une ligne**



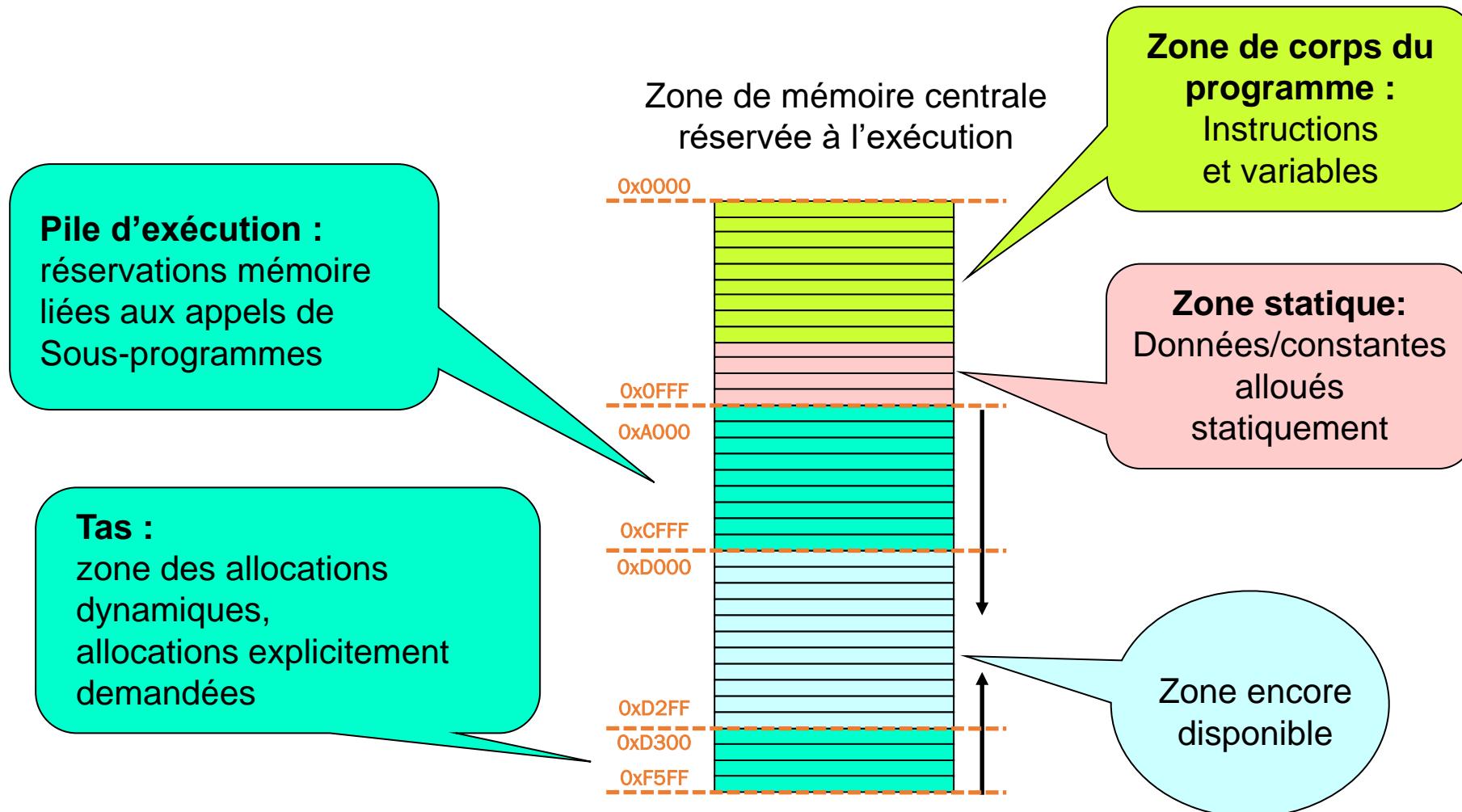
Allocation de mémoire

- Le système peut dire à la mémoire qu'elle va être utilisée, de deux manières :
 - **Allocation statique** : à la compilation (ce qui laisse le système capable de mieux gérer l'ensemble)
 - **Allocation dynamique** : à l'exécution (ce qui laisse l'utilisateur libre, mais ce qui complexifie la tâche pour le système car il ne sait pas ce qu'il devra ou non réservé)
- La gestion de la mémoire à l'exécution est plus délicate
 - Certains langages n'autorisent que l'allocation statique
 - Possibilité de fuites mémoires (on dit au système qu'on souhaite réservé une zone mémoire et on oublie de le lui rendre)

Allocation de mémoire (suite)

- Allocation statique
 - **Etape 1** : à la compilation, le compilateur calcule la taille mémoire nécessaire à la bonne exécution du programme
 - **Etape 2** : à l'exécution, le système d'exploitation vérifie s'il dispose suffisamment de mémoire pour l'exécution du programme ; dans le cas contraire, l'exécution est annulée
- Allocation dynamique
 - **Pile d'exécution (stack)**: zone mémoire qui va permettre l'appel de sous-programmes en autorisant la réservation d'emplacements pour paramètres et variables locales
 - **Tas (heap)**: zone mémoire permettant l'allocation (et la libération) de mémoire à des moments arbitraires du programme (responsabilité du programmeur)

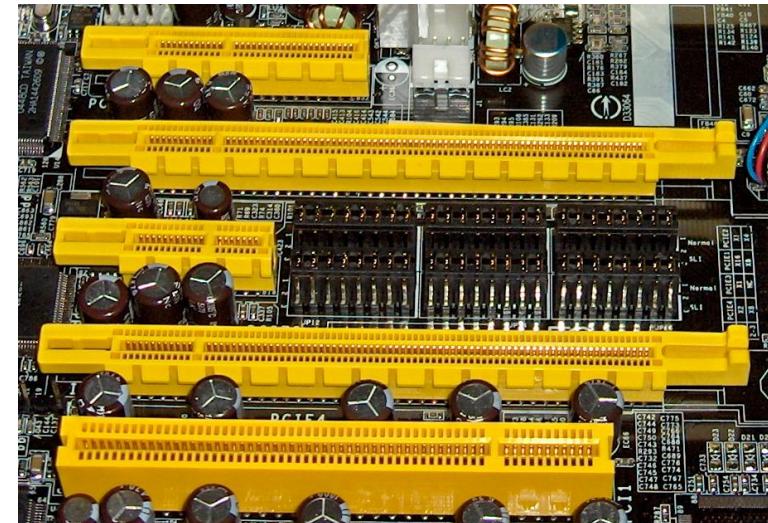
Zones mémoires



Cartes d'extensions et périphériques

Cartes d'extensions

- Certaines fonctionnalités ne sont ajoutées au sein d'un ordinateur que par l'ajout de cartes d'extensions optionnelles au sein de "slots"
 - Slot pour carte graphique (actuellement : PCIe)



- Typiquement : Carte graphique, Carte son, Carte réseau, Carte Wifi etc.
- De plus en plus, la carte mère inclut la plupart de ces fonctionnalités (wifi, réseau, son) ou le CPU (graphisme) mais des cartes plus performantes peuvent être rajoutées !

Carte graphique

- Ce sont les cartes qui exécutent les calculs spécifiquement graphiques de votre ordinateur. Ce sont des calculs massivement parallèle (pixel par pixel)
- Elles sont aujourd’hui adaptées à tous types d’algorithmes massivement parallèle.
- Ce sont des classes d’algorithme spécifique, on ne les verrapar dans ce cours car il s’agit de programmation avancée.

Périphériques

- Un périphérique ajoute des fonctionnalités à un ordinateur
- Périphériques d'entrées : tout ce qui permet à la machine d'obtenir des informations depuis l'extérieur (sens utilisateur -> ordinateur)
 - Clavier, souris, caméra, microphone, scanner, lecteur d'empreintes...
- Périphériques de sortie : ce qui permet à l'utilisateur de renvoyer une information à l'extérieur
 - Écran, haut-parleurs, imprimante
- Il existe aussi des périphériques d'entrée/sortie :
 - Disque dur, graveur de CD/ DVD, clé USB ou encore écran tactile ou manette à retour de force

Hardware et software

UEFI

- L'UEFI est un programme stocké sur une mémoire morte sur la carte mère.
 - LE BIOS (Basic Input-Output System) est l'ancienne version de ce programme, il est présent sur les plus anciennes machines
- Le rôle de l'UEFI est de démarrer le système d'exploitation (booter) de votre machine
 - Vérifie la présence des composants nécessaires
 - Met en mémoire les pilotes nécessaires au démarrage

Pilotes

- Les pilotes (ou drivers) sont des programmes dont le but est de gérer les périphériques
 - Sans pilote, un équipement connecté à un ordinateur ne pourra pas être utilisé
 - Un pilote « traduit » les demandes de l'utilisateur (ou de l'ordinateur) dans « le langage » du périphérique.
 - exemple : imprimer ce fichier excel sur une feuille 14 recto-verso en noir et blanc / afficher ce fichier excel à l'écran / quand j'appuie sur la touche 'A' ce caractère est entré dans l'ordinateur / quand je pousse ma souris, le pointeur monte...
- Les pilotes des équipements présents sur vos machines sont préinstallés et la plupart des équipements actuels sont « plug-and-play »
 - L'équipement quand il est branché est automatiquement détecté et lance l'installation de son pilote.
 - Parfois, un programme doit parler dans le langage du pilote, ce qui demande un apprentissage supplémentaire, l'exemple classique étant le langage de la carte graphique (DirectX, OpenGL, Vulkan) pour faire des affichages.

Le langage C ? Cours suivant !