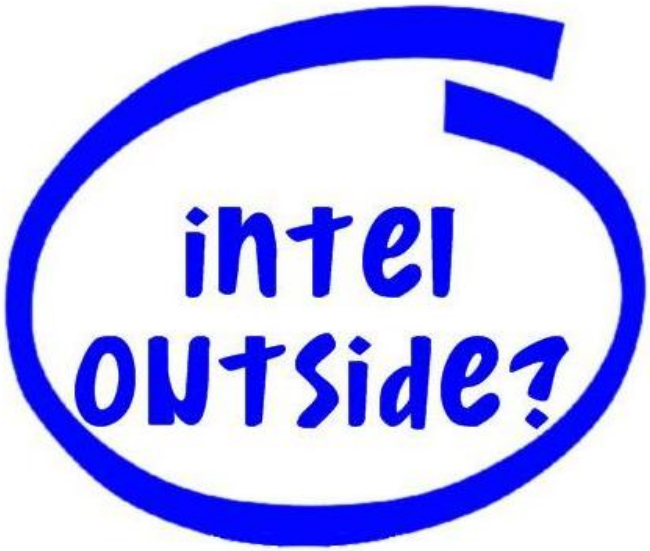
Architecture et Systèmes

Bastien DI PIERRO

Lyon Calcul

27 octobre 2018



Objectifs de ce module

- **Démystifier l'ordinateur**

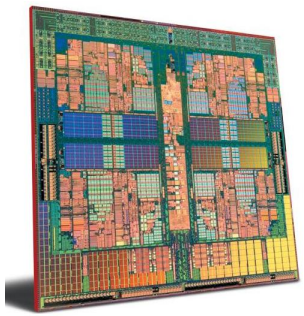
Objectifs de ce module

- **Démystifier l'ordinateur**
- **Modéliser et simuler un problème scientifique issu de la physique, biologie, chimie, ...**
  - Comprendre l'**architecture**
  - Adapter les **méthodes numériques, les algorithmes, les structures de données et la programmation**
  - Comprendre et expliquer le **comportement d'un programme**
  - **Optimiser** les codes de calcul
  - Réaliser des codes **portables et réutilisables**
  - **Paralléliser**

Importance des aspects matériels

La connaissance des architectures de calcul permet de :

- **choisir et d'adapter** son matériel en fonction des besoins
- **dimensionner** correctement en retenant le meilleur compromis : architecture équilibrée
- **comprendre** le comportement d'un programme
- **adapter** les méthodes numériques, les algorithmes, la programmation



Blue-gene P

GPU

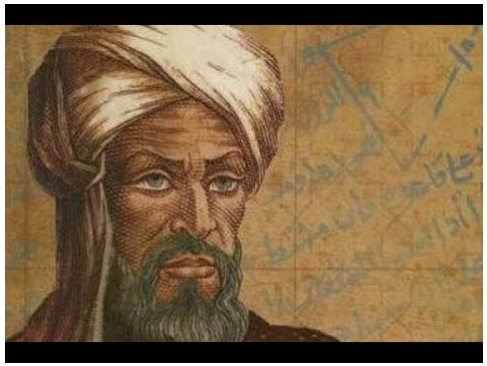
1. Historique
2. Architecture générale séquentielle
3. Architecures parallèles
4. Conclusions

1. Historique
   - Programmation
   - Machine de Turing
   - Architecture de Von-Neumann
2. Architecture générale séquentielle
3. Architecures parallèles
4. Conclusions

Historique - Programmation

- Muhammad Al-Khwarizmi (780-850)
  Mathématicien, Géologue, Astronome, Astrologue
  "Père de l'Algèbre", *The Compendious Book on Calculation by Completion and Balancing*, 830

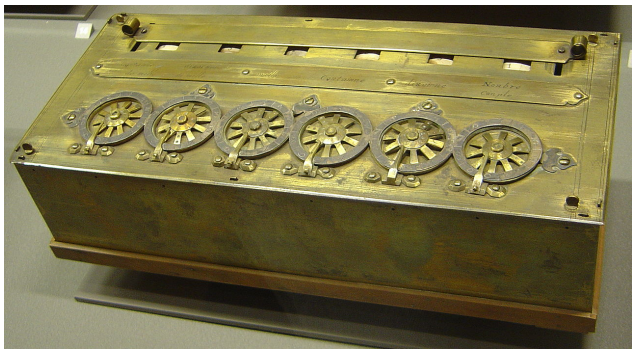
  - Donne son nom au mot algorithme
  - Succession d'opérations simples et non ambigües visant à résoudre un problème précis
  - Exemples : Surface d'un volume, tissage, règle d'un sport, recette de cuisine ...



Historique - Programmation

- Muhammad Al-Khwarizmi (780-850)
  Mathématicien, Géologue, Astronome, Astrologue
  "Père de l'Algèbre", *The Compendious Book on Calculation by Completion and Balancing*, 830

- Blaise Pascal (1623-1662)
  Mathématicien, Physicien, Théologue, Philosophe

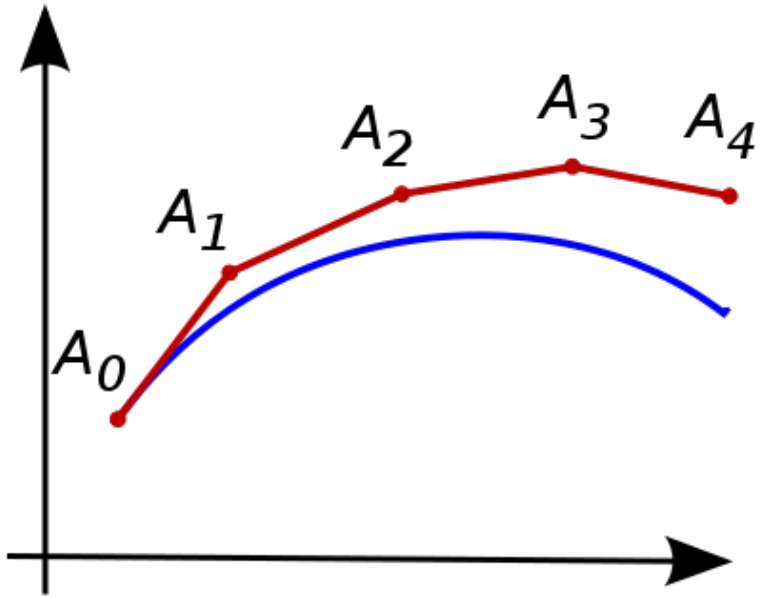


- La Machine à calculer (1642)
- Langage Pascal.

Historique - Programmation

- Leonhard Euler (1707-1783)
  Mathématicien, Physicien, Astronome, Ingénieur.

  1768, Institutionum calculi integralis
  Méthodes d'intégration approchées pour résoudre les EDO



Historique - Programmation

- Leonhard Euler (1707-1783)
  Mathématicien, Physicien, Astronome, Ingénieur.
- Ada Lovelace (1815-1852)
  Mathématicienne, Écrivaine
  Premier programme “informatique” (1842, nombres de Bernoulli), langage ADA.

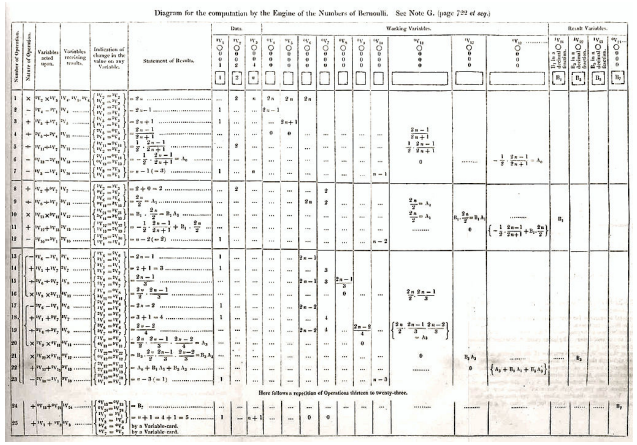


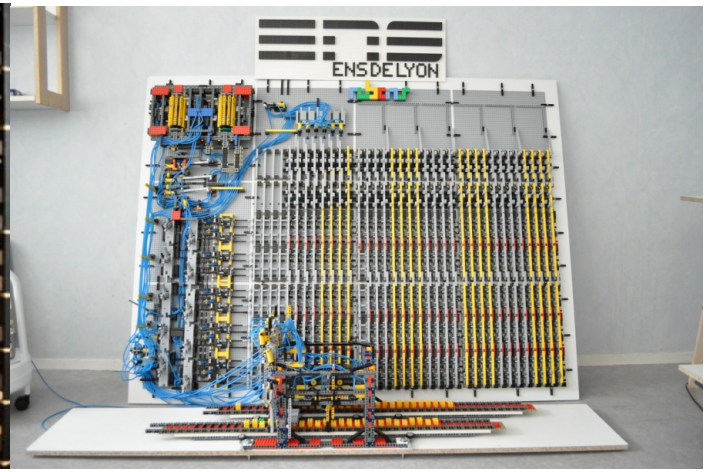
Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

1. Historique
   - Programmation
   - Machine de Turing
   - Architecture de Von-Neumann
2. Architecture générale séquentielle
3. Architecures parallèles
4. Conclusions

Historique - Premier ordinateur

- Allan Turing (1912-1954), Mathématicien, Logisticien
- Pionnier des sciences informatiques, informatique théorique
- Introduit la notion de "calculable" (calculus : petit caillou) $\Rightarrow$ Qu'est-ce qui est calculable ?
- Machine de Turing (Modèle Abstrait)
  $\Rightarrow$ Formalisme Mathématique
- Casse les codes allemands pendant WWII $\Rightarrow$ Estimation : 2 à 4 ans de gagnés

Machine de Turing



Machine minimaliste :

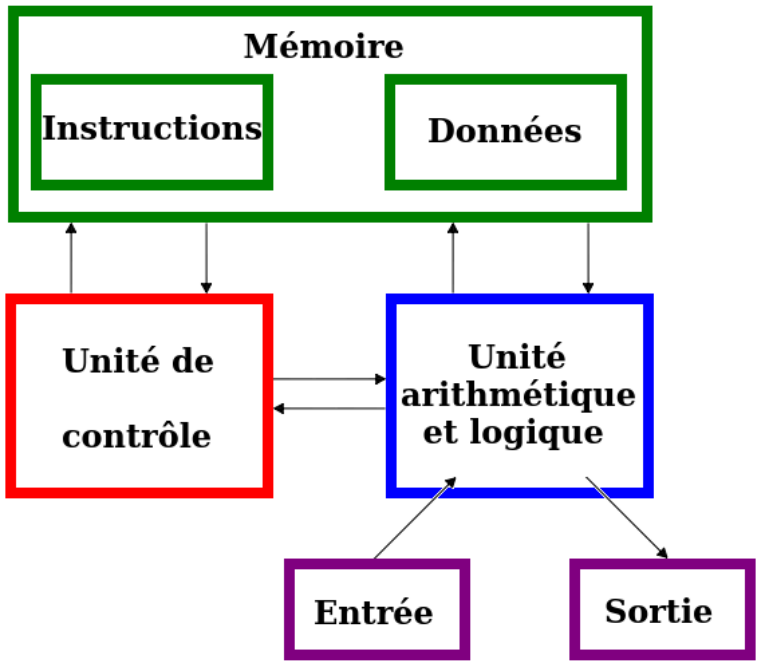
- Ruban (données)
- Tête lecture-écriture
- Registre d'état (mémoire)
- Table d'actions (calculs)

Inconvenient : 1 machine $\rightarrow$ 1 série d'instruction !

1. Historique
   - Programmation
   - Machine de Turing
   - Architecture de Von-Neumann
2. Architecture générale séquentielle
3. Architecures parallèles
4. Conclusions

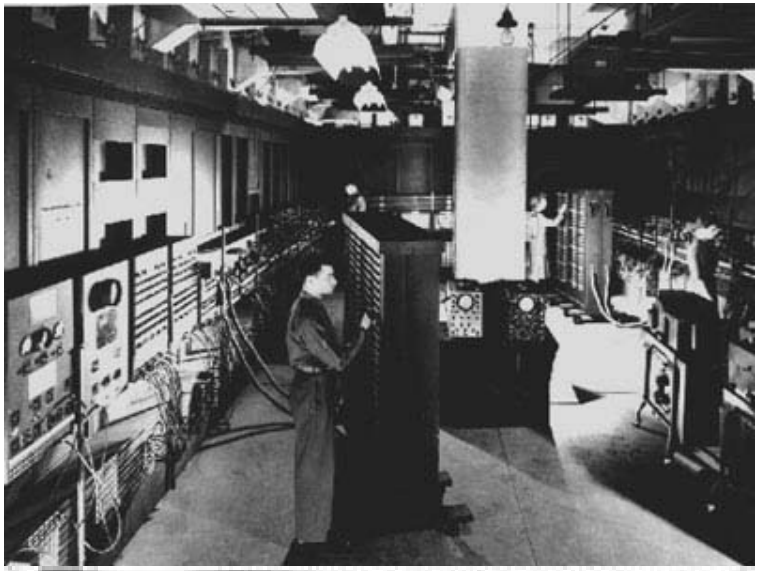
Architecture de Von-Neumann

- Jhon Von Neumann (1903-1957)
  Mathématicien, Physicien
- Idée : instructions et données stockées dans la même mémoire !
- Possibilité de changer les instructions, surtout en cours de calcul !



$\Rightarrow$ 1 machine $\approx$ infinité d'opérations !
$\Rightarrow$ Architecture de la plupart des ordinateurs actuels !

Quelques dinosaures



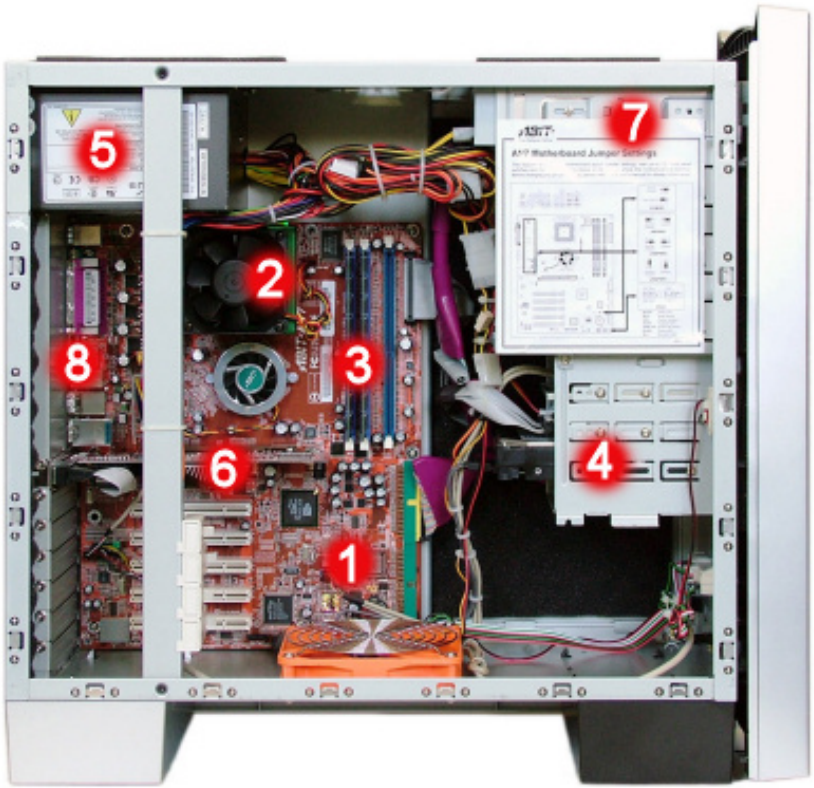
1 Historique

2 Architecture générale séquentielle
- Architecture générale
- Le processeur (CPU)
- La mémoire
- Communications internes et réseaux

3 Architecures parallèles

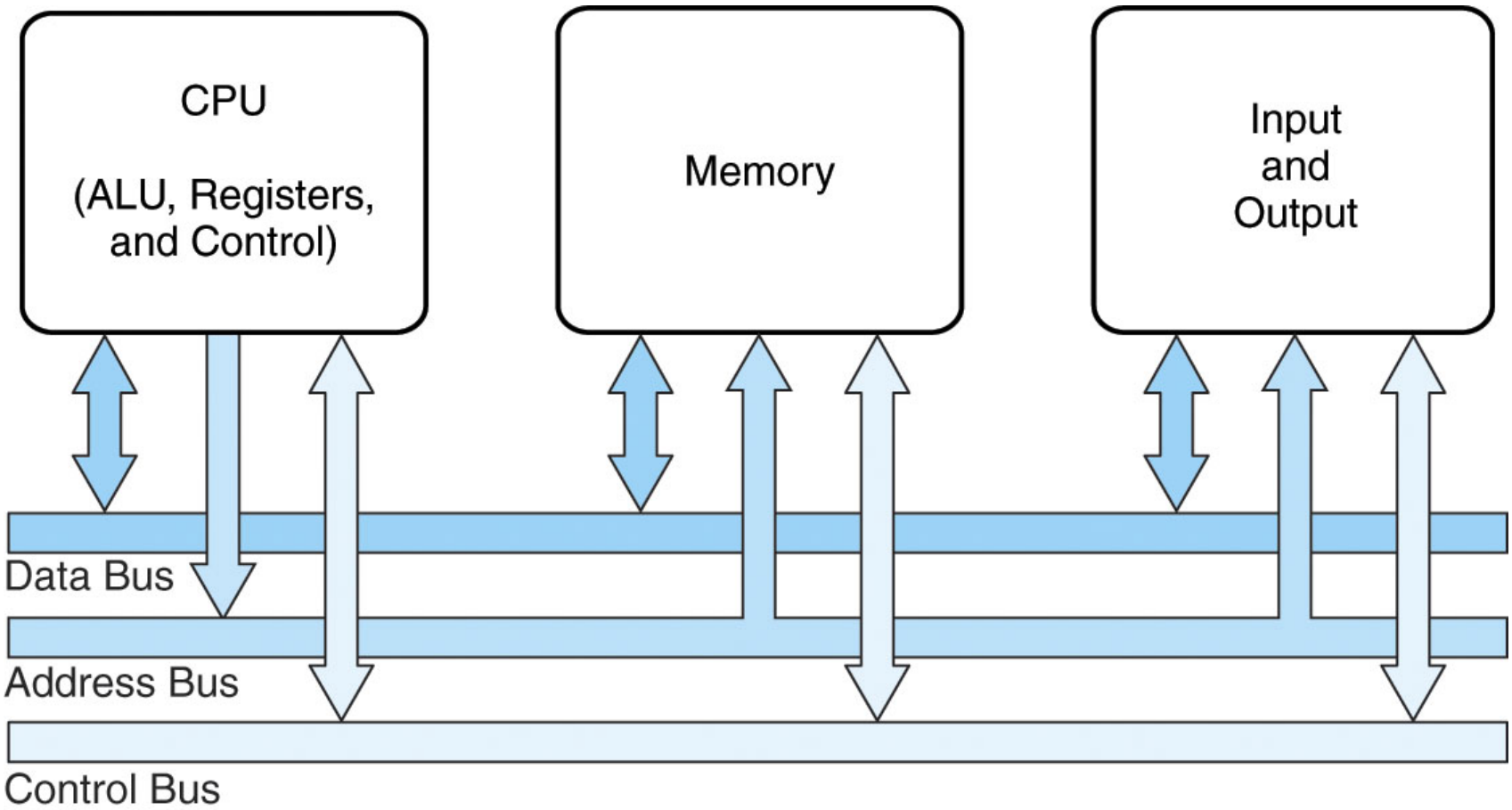
4 Conclusions

Les différents composants



1. Carte mère
2. Processeur
3. Mémoire vive (RAM)
4. Disque dur (ROM)
5. Alimentation
6. Carte graphique

Architecture générale



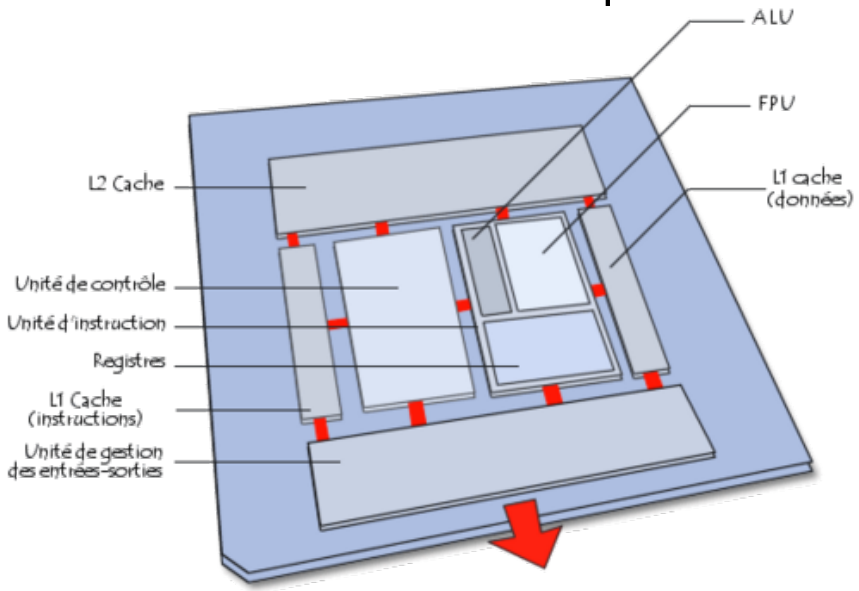
1. Historique
2. Architecture générale séquentielle
   - Architecture générale
   - Le processeur (CPU)
   - La mémoire
   - Communications internes et réseaux
3. Architecures parallèles
4. Conclusions

Le processeur

- Unité centrale de traitement (UCT), Central Processing Unit (CPU)
  $\Rightarrow$ Cerveau de l'ordinateur !
- Exécute les instructions machines (ex : addition, sinus, logarithme, ...)
- Traite généralement des données binaires
- Possède une mémoire interne
- Différentes famille :
  - 80x86
  - IA-64
  - PowerPC
  - SPARC
  - ARM
  - ...

Composition d'un cœur

- **Unité de commande** : permet de séquencer les instructions
- **Unité Arithmétique et Logique (UAL)**, calculs arithmétiques élémentaires sur des nombres entiers et opérations logiques
- **Unité de calcul en virgule flottante** (en anglais Floating Point Unit - FPU) : traitement des calculs sur des nombres à virgule
- **Registres** : mémoires de petite taille (quelques octets) très rapides
- **Mémoires caches** : diminuent les temps d'accès à la mémoire



Caractéristique d'un processeur

- **Nombre de cœurs** : le processeur peut intégrer plusieurs noyaux de calcul appelés cœurs

Caractéristique d'un processeur

- **Nombre de cœurs** : le processeur peut intégrer plusieurs noyaux de calcul appelés cœurs
- **Fréquence d'horloge** : vitesse de fonctionnement du processeur = nombre de **cycles** que le processeur est capable d'effectuer par seconde
  - **Cycle** = plus petite unité de temps au niveau du processeur. Chaque opération/instruction nécessite au minimum un cycle, et plus souvent plusieurs
  - $1 GHz = 10^9 Hz = 10^9 \mathrm{cycle/s}$

Caractéristique d'un processeur

- **Nombre de cœurs** : le processeur peut intégrer plusieurs noyaux de calcul appelés cœurs
- **Fréquence d'horloge** : vitesse de fonctionnement du processeur = nombre de **cycles** que le processeur est capable d'effectuer par seconde
- **Jeu d'instructions** : ensemble des opérations qu'un processeur peut exécuter (+, -, *, ,/, sin, ...)
  Conséquences directes

Caractéristique d'un processeur

- **Nombre de cœurs** : le processeur peut intégrer plusieurs noyaux de calcul appelés cœurs
- **Fréquence d'horloge** : vitesse de fonctionnement du processeur = nombre de **cycles** que le processeur est capable d'effectuer par seconde
- **Jeu d'instructions** : ensemble des opérations qu'un processeur peut exécuter (+, -, *, ,/, sin, ...)
- **Largeur (32 ou 64 bits)** : notamment du bus de données et des registres internes. Bon indicateur de la quantité d'informations que celui-ci peut gérer en un temps donné

Caractéristique d'un processeur

- **Nombre de cœurs** : le processeur peut intégrer plusieurs noyaux de calcul appelés cœurs
- **Fréquence d'horloge** : vitesse de fonctionnement du processeur = nombre de **cycles** que le processeur est capable d'effectuer par seconde
- **Jeu d'instructions** : ensemble des opérations qu'un processeur peut exécuter (+, -, *, ,/, sin, ...)
- **Largeur (32 ou 64 bits)** : notamment du bus de données et des registres internes. Bon indicateur de la quantité d'informations que celui-ci peut gérer en un temps donné

Adressage mémoire

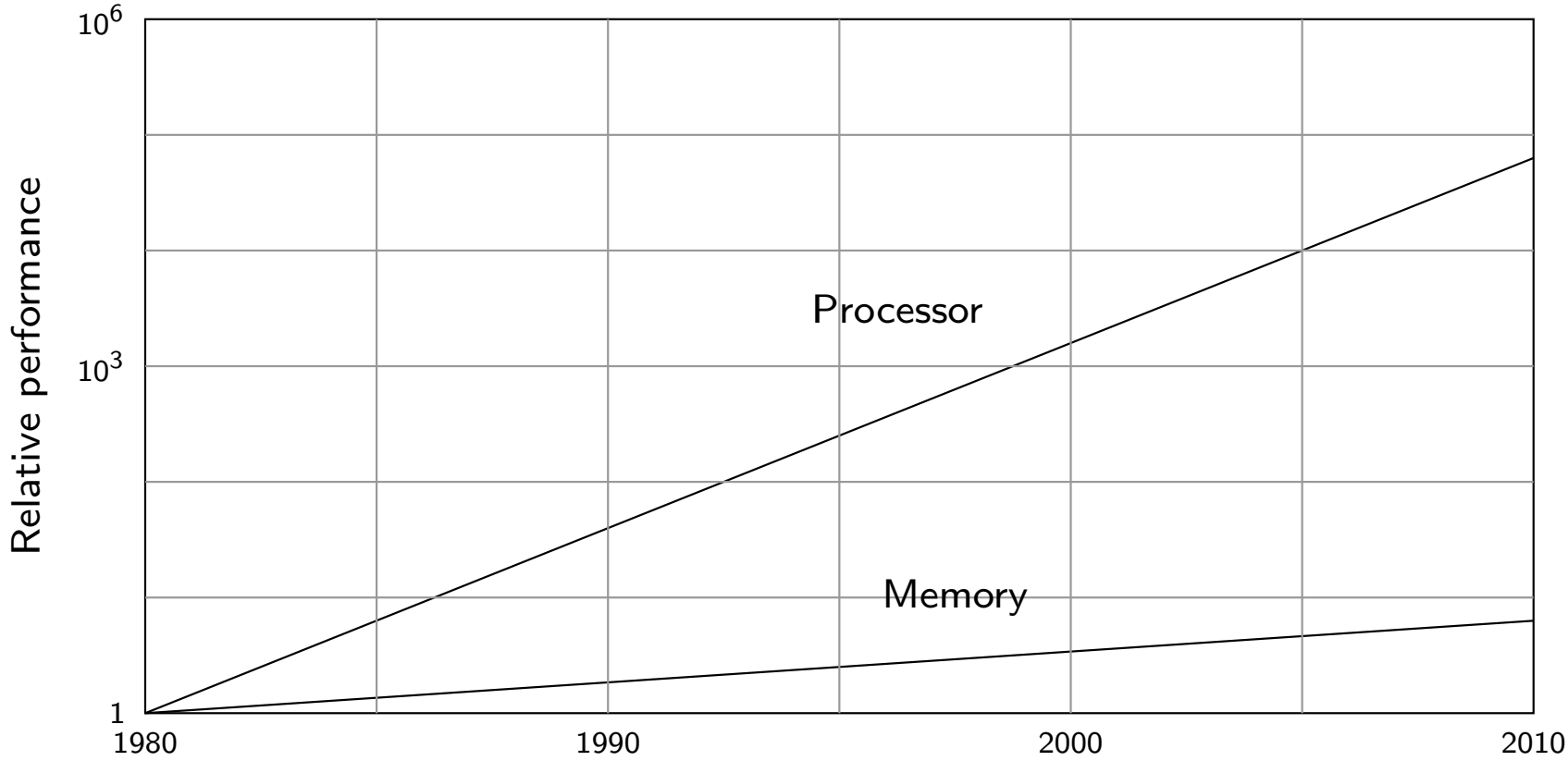
Les **processeurs 32 bits** ne peuvent pas adresser plus de 4 gigaoctets (2^{32}) de mémoire centrale, tandis que les **processeurs 64 bits** peuvent adresser 16 exaoctets (2^{64}) de mémoire.

Flops et performance du processeur

- La performance s'exprime en **opérations à virgule flottante par seconde** = FLoating point Operations Per Second
- **Puissance crête** (point de vue théorique) : mesure les performance des **unités de calcul en virgule flottante** (FPU) contenues dans le cœur.
  - **Exemple sur un processeur Intel Sandybridge**
    6 coeurs, 3.2GHz, registres vectoriels 16 (simple précision) ou 8 (double)
    Performance crête d'un processeur :
    $$6 \times 3.2 \times 16 = 307.2 \text{ GFLOPS}$$
- **D'un point de vue pratique**, la puissance d'une machine dépend de l'**ensemble de ses composants** :
  - accès mémoire
  - vitesse des bus
  - système d'exploitation
  - charge de la machine

1. Historique
2. Architecture générale séquentielle
   - Architecture générale
   - Le processeur (CPU)
   - La mémoire
   - Communications internes et réseaux
3. Architecures parallèles
4. Conclusions

Memory Wall : Vitesse CPU >> Vitesse Memoire



- Mémoire : élément très pénalisant !

Définitions

bit et octect

- **bit (b)** : binary digit, plus petite unité d'information d'un composant en informatique. Vaut 0 ou 1.
- **octet (o) ou byte (B)** = composé de 8 bits
- $1ko = 1000$ octets, $1kibio = 2^{10}$ octets $= 1024$ octets $= 8192$ bits

Bande passante : Débit d'informations

- Mesurée généralement en **octets** (**byte**) par seconde (o/s, ou B/s) ou en **bits** par seconde (bit/s ou bps)

Latence

Temps minimum d'établissement de la connexion : indépendant de la quantité de données à transporter.

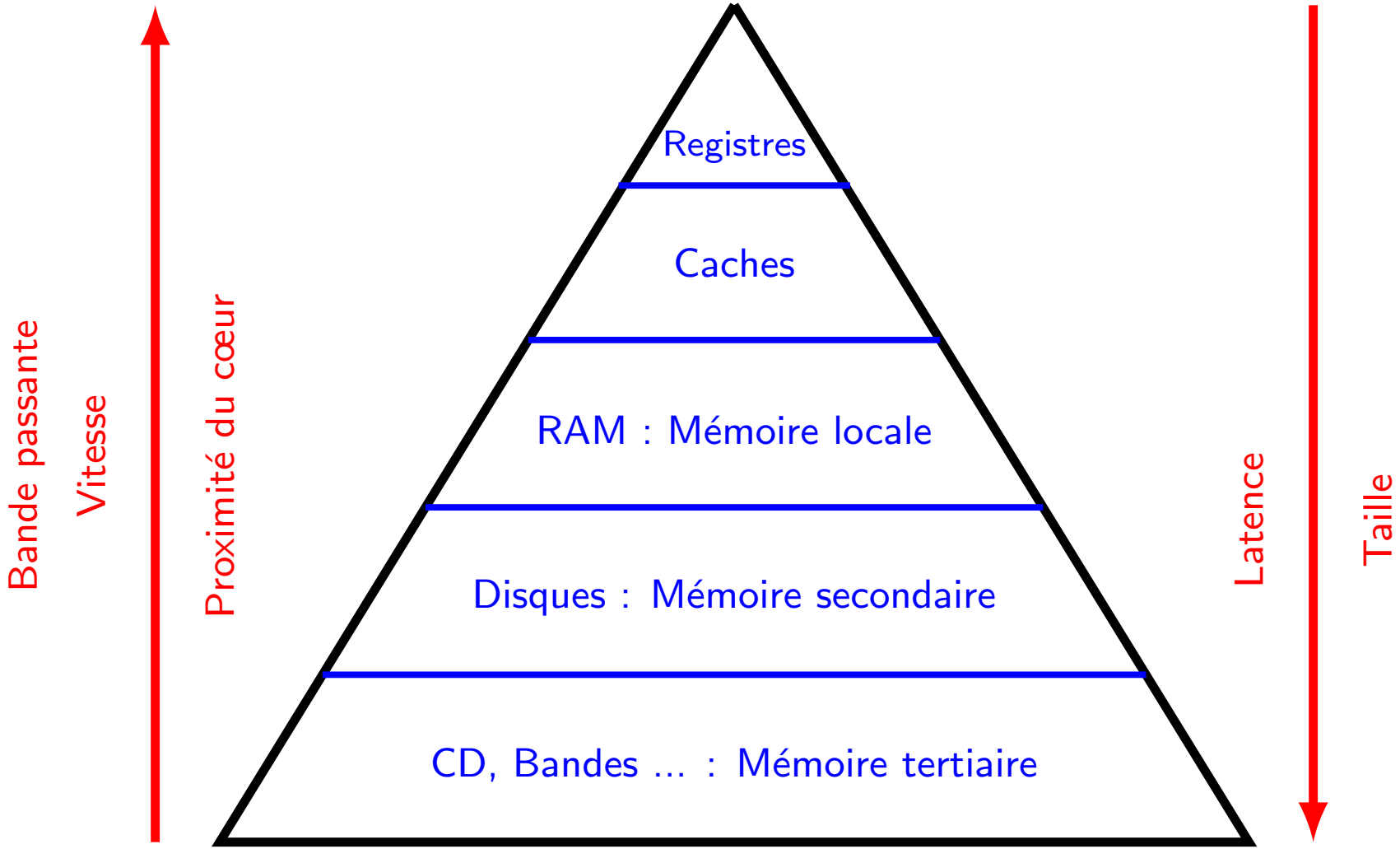
- Mesurée généralement en **secondes**

Différents types de mémoires

- Cache : petite mémoire **interne processeur** en accès très rapide
- RAM : Mémoire centrale de l'ordinateur, située sur la carte mère. Mémoire vidangée à chaque reboot.
- Disque dur : Stockage des données, sauvegarde au long terme.
- Quelques ordres de grandeurs :

Type	**Taille**	**Vitesse**	**Coût/bit**
Registre	$<$ 1 KB	$<$ 1 ns	$$$$
SRAM On-chip	8 KB - 6 MB	$<$ 10 ns	$$$
SRAM Off-chip	1 MB - 16 MB	$<$ 20 ns	$$
DRAM	64 MB - 1 TB	$<$ 100 ns	$
Flash	64 MB - 32 GB	$<$ 100 μs	c
Disk	40 GB - 1 PB	$<$ 20 ms	$\sim$0

Hiérarchie Mémoire



Problématique

Memory wall

- L'accès aux données est un des principaux facteurs limitant la performance
- Les machines sont déséquilibrées

Equilibre d'une machine

- Ratio entre la bande passante mémoire et la performance crête

$$B_m = \frac{\text{memory bandwith (GWords/sec)}}{\text{peak performance (GFlops/sec)}}$$

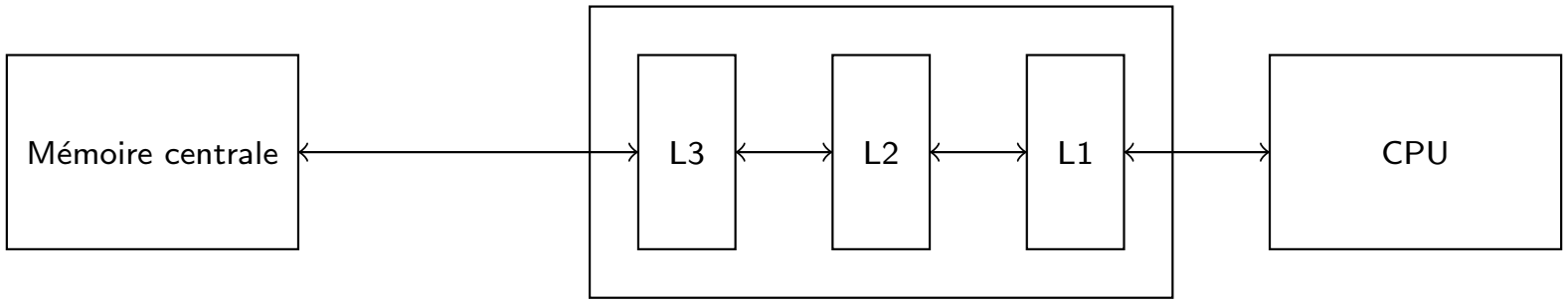
data path	**balance** (W/F)
cache	0.5 - 1.0
machine (RAM)	0.03 - 0.5
GBit ethernet	0.0001 - 0.0007
disque	0.0001 - 0.01

Accès mémoire : problématique

Quels facteurs influencent les performances des accès mémoires ?

- **Localisation**: cache, RAM, disque ?
- **Manière** dont on y accéde :
  - au travers d'un chipset (jeu de composants électroniques qui permet le contrôle des échanges d'informations)
  - directement par le processeur
  - via le processeur voisin ...

Plus une donnée est **proche** du processeur, plus elle est accédée **rapidement**.



Principes de localité

Localité spatiale

Lorsqu'un programme accède à une donnée ou à une instruction, il est probable qu'il accédera ensuite aux données ou instructions **voisines**

Localité temporelle

Lorsqu'un programme accède à une donnée ou à une instruction, il est probable qu'il y accédera à nouveau dans un **futur proche**

Exemple

```
subroutine sumVec(vec, n)
    integer :: n
    integer :: vec(n)
    integer :: i,sum=0
    do i = 1, n
        sum = sum + vec(i)
    end do
end subroutine
```

- Bonne localité spatiale des données du tableau `vec` : accès en séquence
- Bonne localité temporelle de la donnée `sum` : accès fréquent
- Bonne localité spatiale et temporelle des instructions : accès en séquence et fréquemment

Fonctionnement des caches

- Le cache est divisé en **lignes (ou blocs) de mots**
- 2 niveaux de granularité :
  - le CPU travaille sur des mots (par ex 32 ou 64 bits)
  - les transferts mémoire se font par ligne (ou bloc, par ex 256 octets)
- Exploitation de la **localité spatiale** : le cache contient des copies des mots par ligne de cache
- Exploitation de la **localité temporelle** : choix judicieux des lignes de cache à retirer lorsqu'il faut rajouter une ligne à un cache déjà plein

Lorsque le processeur tente d'accéder à une information (instruction ou donnée)

Si l'information se trouve dans le cache (**hit**), le processeur y accède sans état d'attente, sinon (**miss**) le cache est chargé avec un bloc d'informations de la mémoire ($\sim 1\ ms$).

Quelles conséquences pour nous ?

Exemple du parcours d'une matrice

- On considère un **tableau 2D**
- **Allocation mémoire** : les données sont stockées dans un bloc **mémoire contigu** sous la forme d'un vecteur
- En **C/C++** : stockage des lignes les unes derrière les autres
- En **Fortran** : stockage des colonnes les unes derrière les autres

for $i \in 1, n$ **do**
  for $j \in 1, n$ **do**
    $a_{i,j}x_j = 0.$
  end for
end for

for $i \in 1, n$ **do**
  for $j \in 1, n$ **do**
    $a_{j,i}x_i = 0.$
  end for
end for

Une version est bonne, l'autre pas ! Démonstration par l'exemple

Stockage : Goulot d'étranglement

Le débit global est dépendant de **toute la chaîne** d'I/Os (baie, serveur, interconnexion ...).

- La **capacité de stockage** augmente beaucoup plus vite que la **vitesse d'accès**
- Le **débit** vers les disques augmente moins vite que la **puissance de calcul**
- La **quantité de données générée** augmente avec la puissance de calcul
- L'approche classique un fichier/processus génère de plus en plus de **fichiers**
- Trop de fichiers = **saturation des systèmes de fichiers** (nombre maximum de fichiers et espace gaspillé à cause des tailles de bloc fixes)
- Le **temps** passé dans les I/O croît (surtout pour les applications massivement parallèles)

1. Historique
2. Architecture générale séquentielle
   - Architecture générale
   - Le processeur (CPU)
   - La mémoire
   - Communications internes et réseaux
3. Architecures parallèles
4. Conclusions

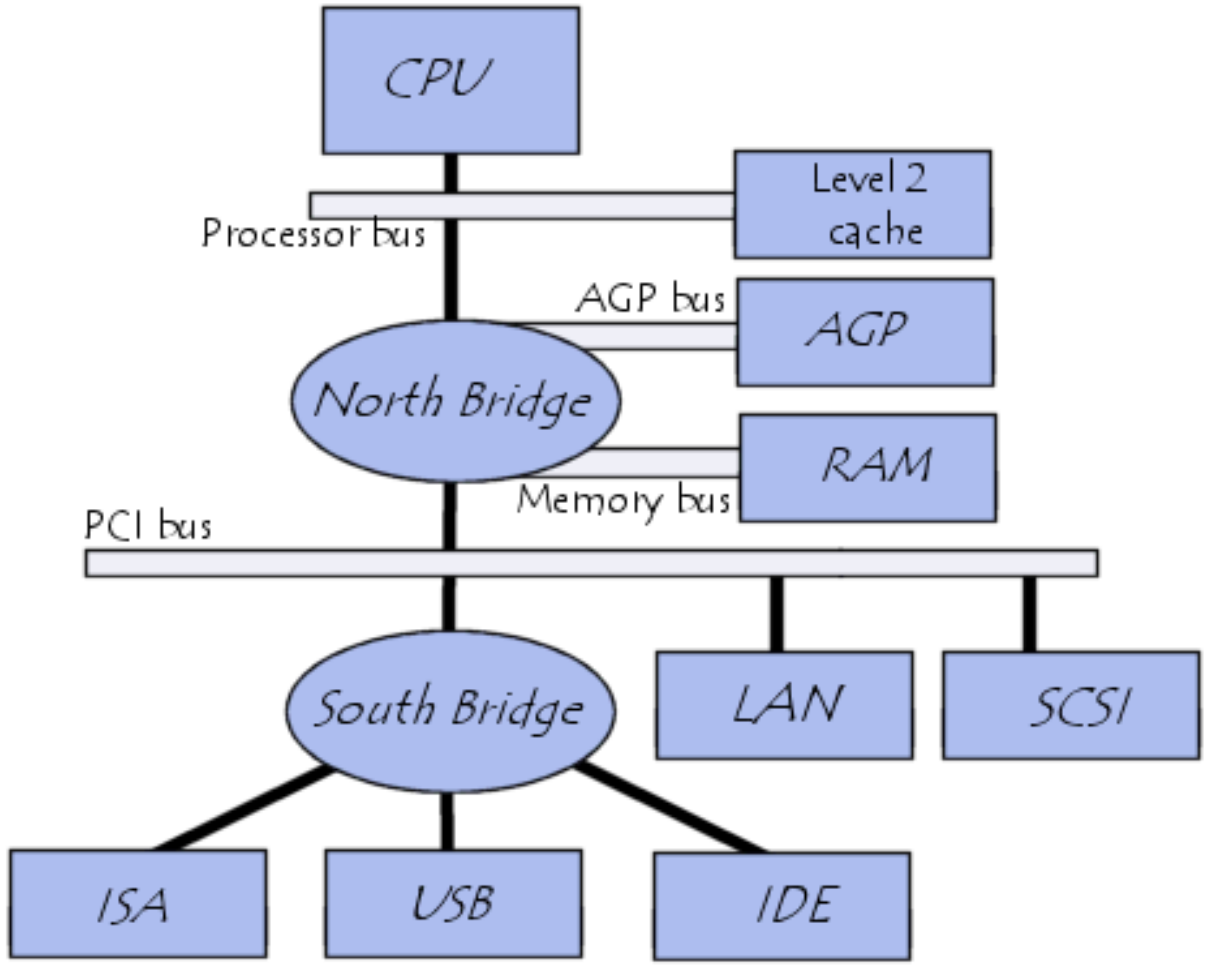
Quelques définitions

- **Bus** = ensemble de liaisons physiques pouvant être exploitées en commun par plusieurs éléments matériels afin de communiquer.
- **Bande passante** = débit d'informations ; quantité d'informations échangée par unité de temps
- **Latence** = temps de réponse du bus à une requête de transfert, temps minimum d'établissement de la connexion : indépendant de la quantité de données à transporter

Ordres de grandeur

Protocole	Usage	Débit théorique maximal
SATA 3.0	Disque dur	750 Mo/s
USB 2.0	Périphériques externes	60 Mo/s
DDR3	RAM	17000 Mo/s
ADSL	Réseau domestique	8 Mb/s

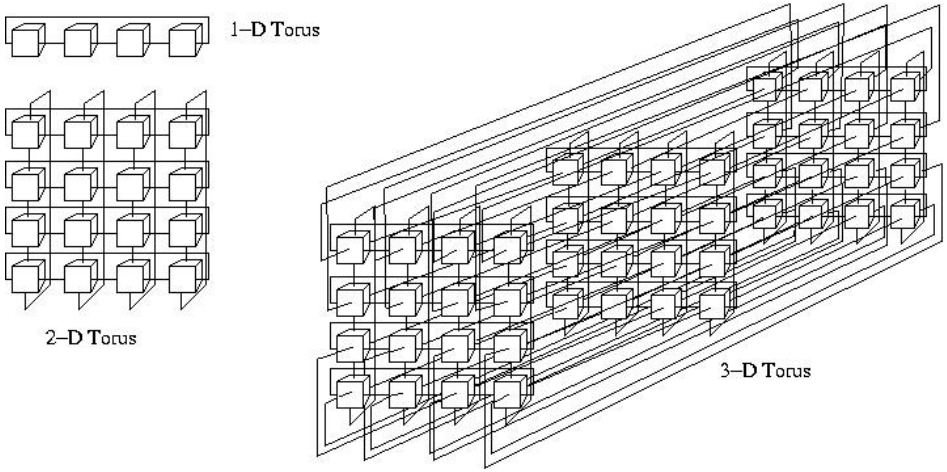
Communications internes

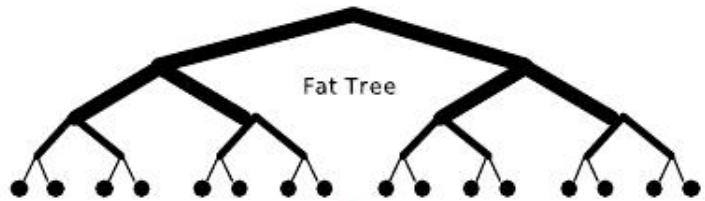


Chipset : gère les flux de données

Caractéristiques des réseaux

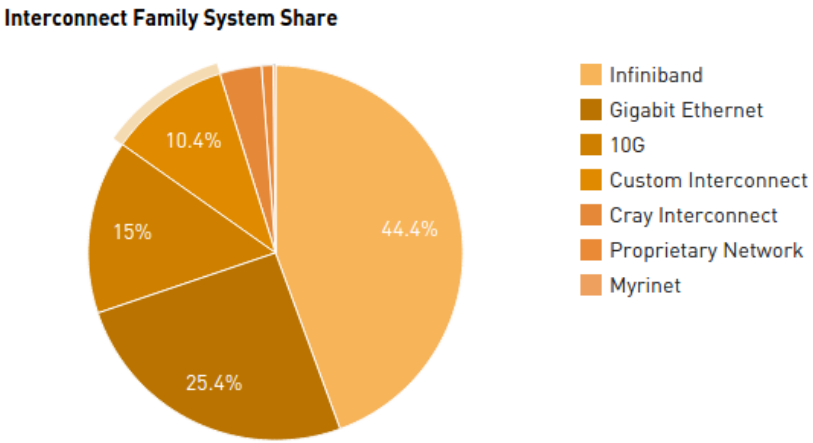
- **La bande passante (bandwidth)** est le débit maximal.
  - La bande passante effective est généralement inférieure à la bande passante physique
- **La latence** : temps d'initialisation de la communication.
- **La distance maximum** entre deux nœuds.
- **La topologie**: tore, fat-tree ...





Technologies réseaux

Technologie	Latence	Bande passante
1 Gb Ethernet	-	1 Gb/sec
10 Gb Ethernet	$\sim$10 μs	10 Gb/sec
Infiniband	$<$ 2 μs	10, 20 & 40 Gb/sec
Myrinet	2.5 - 5.5 μs	10 Gb/sec

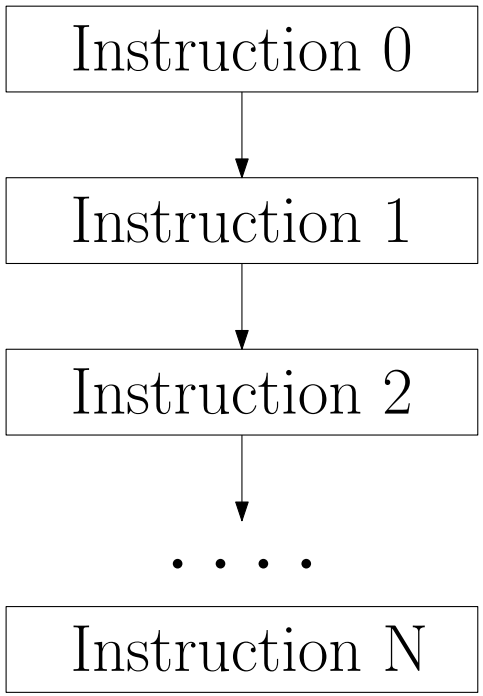


Répartition des familles d'interconnexion, top 500, juin 2014

1. Historique
2. Architecture générale séquentielle
3. Architecures parallèles
   - Introduction
   - Multi-coeur
   - GPU
   - Many-coeur
   - Mémoire distribuée
4. Conclusions

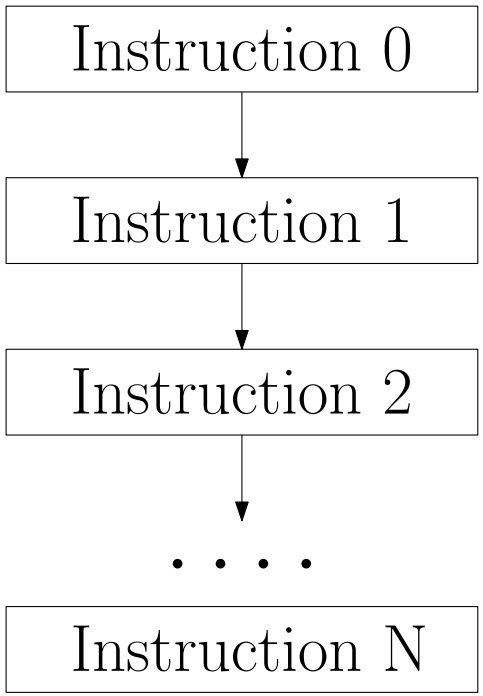
Concept : calcul séquentiel

- **Calcul séquentiel :**
  Les instructions (et accès mémoires) s'exécutent les unes après les autres.
- **Conjecture de Moore :**
  "La puissance double tous les 18 mois !"



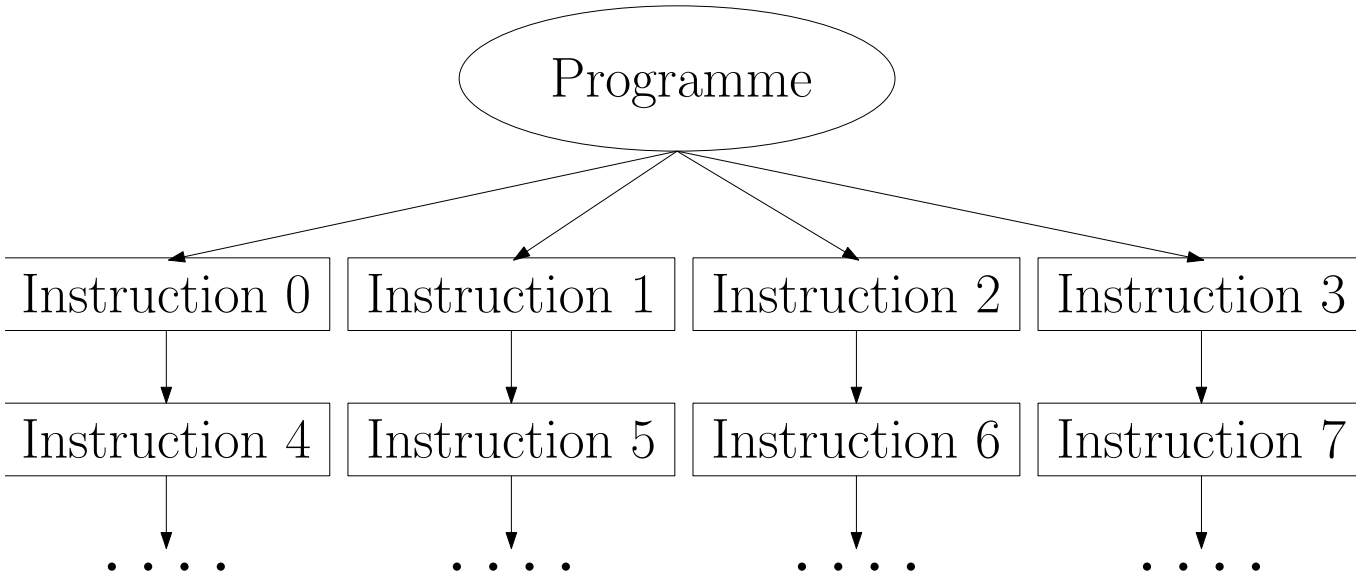
Concept : calcul séquentiel

- **Calcul séquentiel :**
  Les instructions (et accès mémoires) s'exécutent les unes après les autres.
- **Conjecture de Moore :**
  "La puissance double tous les 18 mois !"
- **Limitation 1 : consommation**
  Puissance $\propto$ fréquence^3
  (Curie : $\approx$ 3.5MW $\approx$ Bron / 4)
- **Limitation 2 : miniaturisation**
  Prochain processeur Intel
  $\Rightarrow$ gravés à 14nm ($\approx$ 100 atomes)
  $\Rightarrow$ les coûts explosent !



Concept : Architecture parallèle

- **Calcul parallèle** : Ensemble de techniques logicielles et matérielles permettant l'exécution simultanée de séquences d'instructions indépendantes, sur des processeurs différents.



Pourquoi paralléliser ?

- **Grands challenges** :
  - météo et climat
  - biologie
  - géophysique
  - astrophysique
  - chimie
  - nucléaire
- **Applications commerciales** :
  - bases de données parallèles
  - exploration pétrolière
  - moteurs de recherche web
  - réalité virtuelle

✓ Traiter des problèmes **plus grands et/ou plus complexes**.

✓ Mais aussi **exploiter les architectures actuelles** !

La multiplication du nombre d'unités de calcul ne divise pas spontanément le temps d'exécution des programmes !

Demonstration

Classification de Flynn

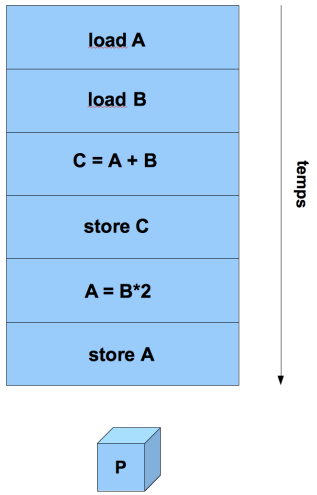
Les programmes et les architectures sont classés selon le type d'organisation du flux de données et du flux d'instructions.

- 2 états possibles pour chacune : "single" ou "multiple"

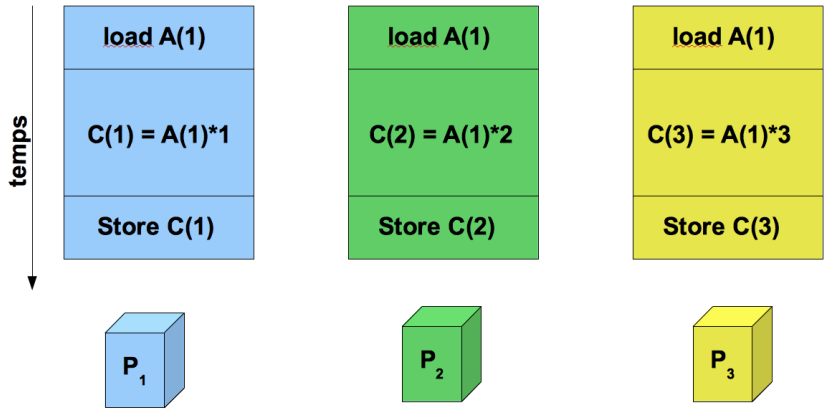
SISD Single Instruction, Single Data $\Longrightarrow$ machines séquentielles	**SIMD** Single Instruction, Multiple Data $\Longrightarrow$ processeurs vectoriels, GPU
MISD Multiple Instruction, Single Data $\Longrightarrow$ rare	**MIMD** Multiple Instruction, Multiple Data $\Longrightarrow$ multiproc, multicores

SISD, MISD

SISD : architecture séquentielle avec un seul flot d'instructions, un seul flot de données, exécution déterministe.



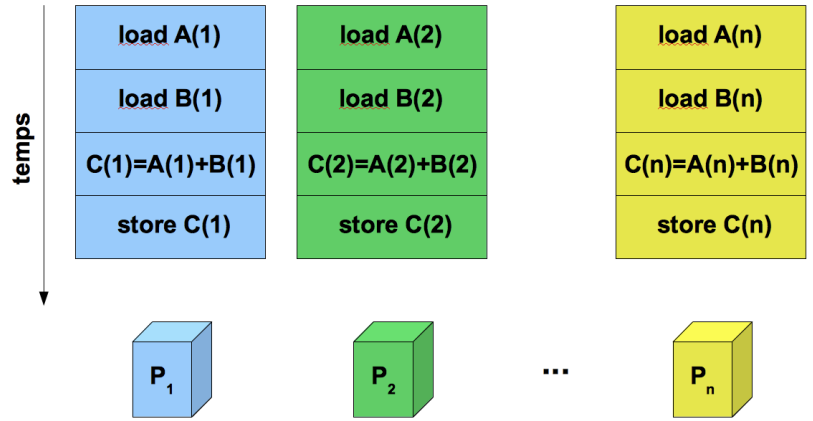
MISD : un seul flot de données alimente plusieurs unités de traitement
$\Rightarrow$ unités de traitement indépendantes
$\Rightarrow$ instructions indépendantes



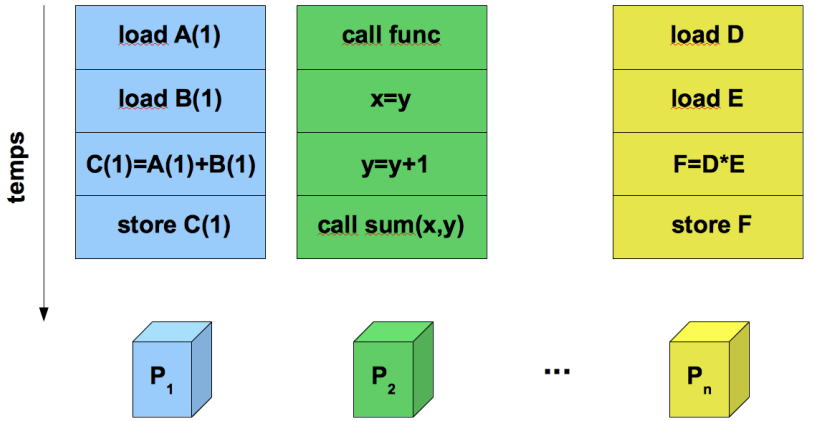
Ex d'utilisation : exécution en // de plusieurs algos de cryptographie pour le décodage d'1 même message.

SIMD, MIMD

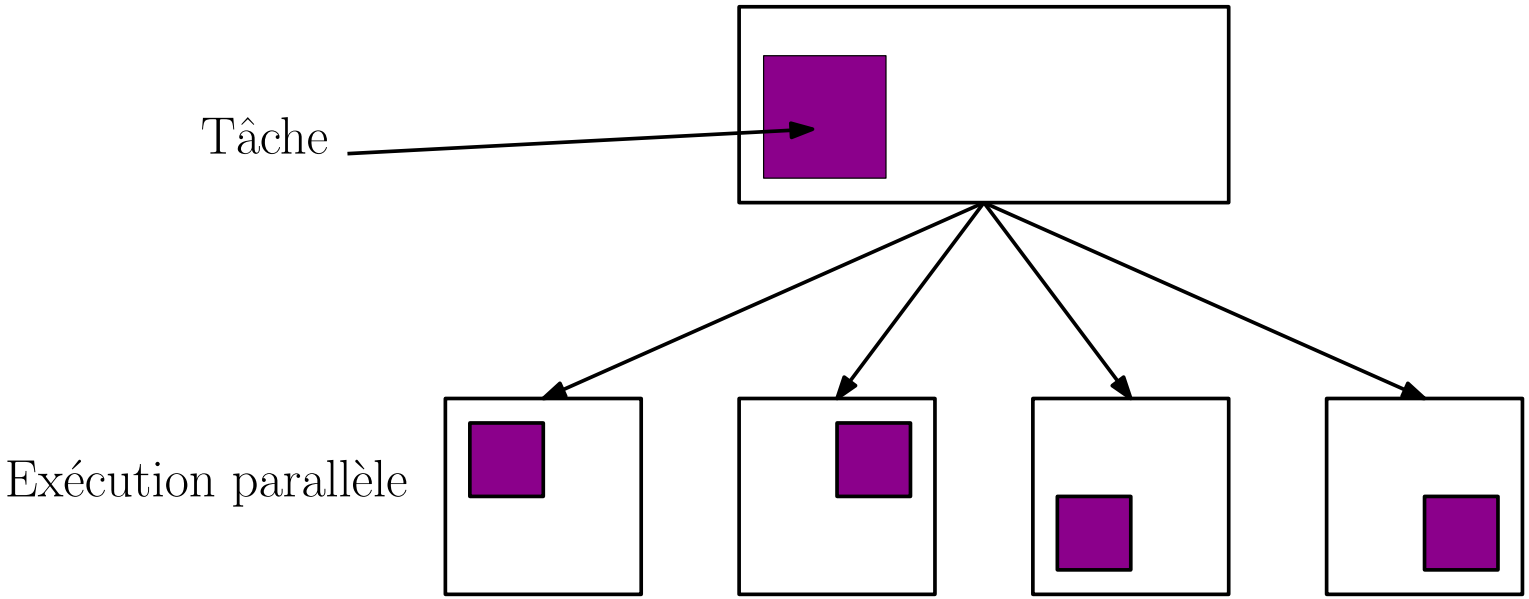
SIMD architecture //
⇒ même instruction
⇒ données différentes
Exemple : rameurs



MIMD architecture la plus courante.
⇒ instructions indépendantes
⇒ données différentes
Exemple : équipe foot

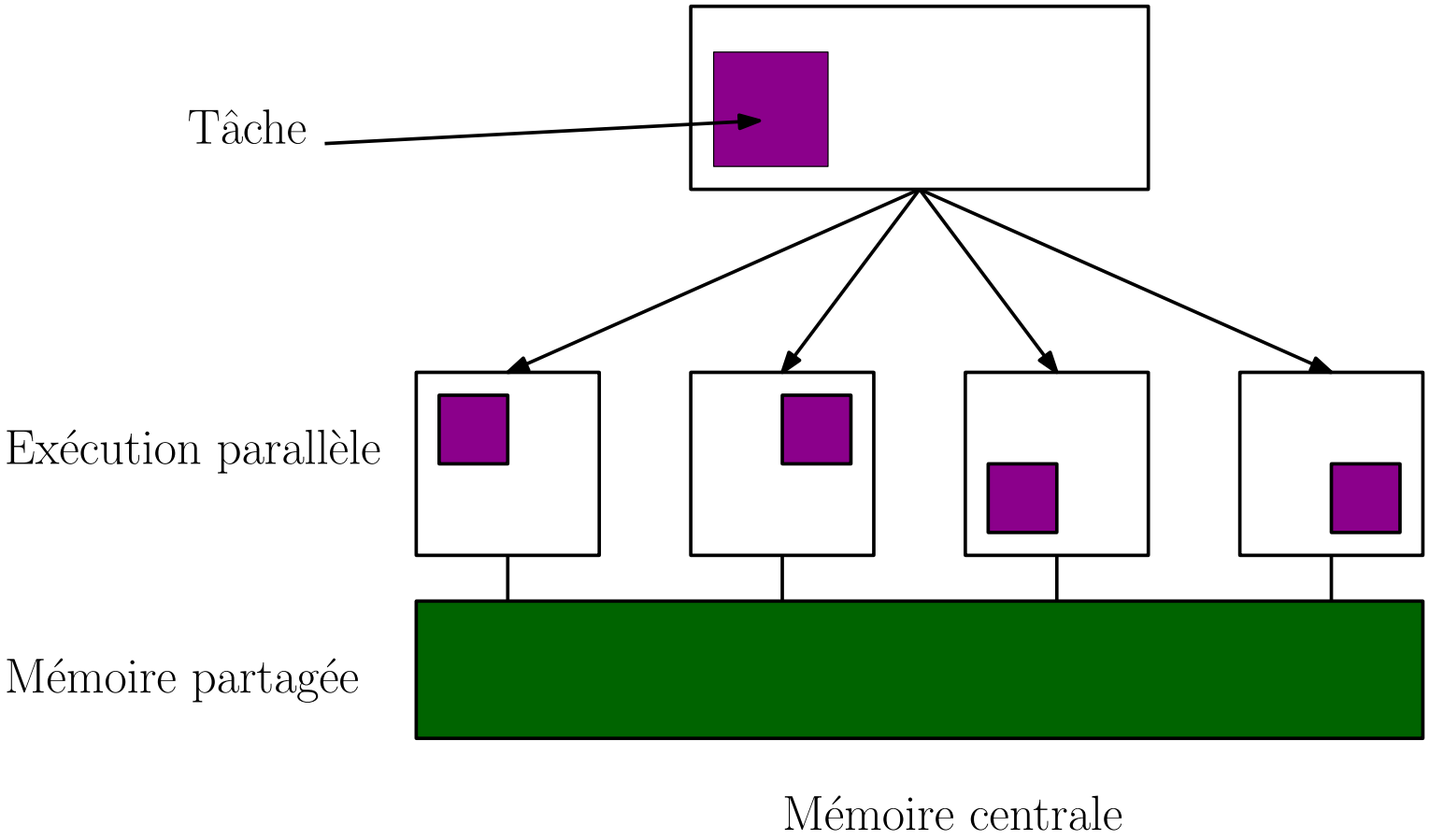


Terminologie



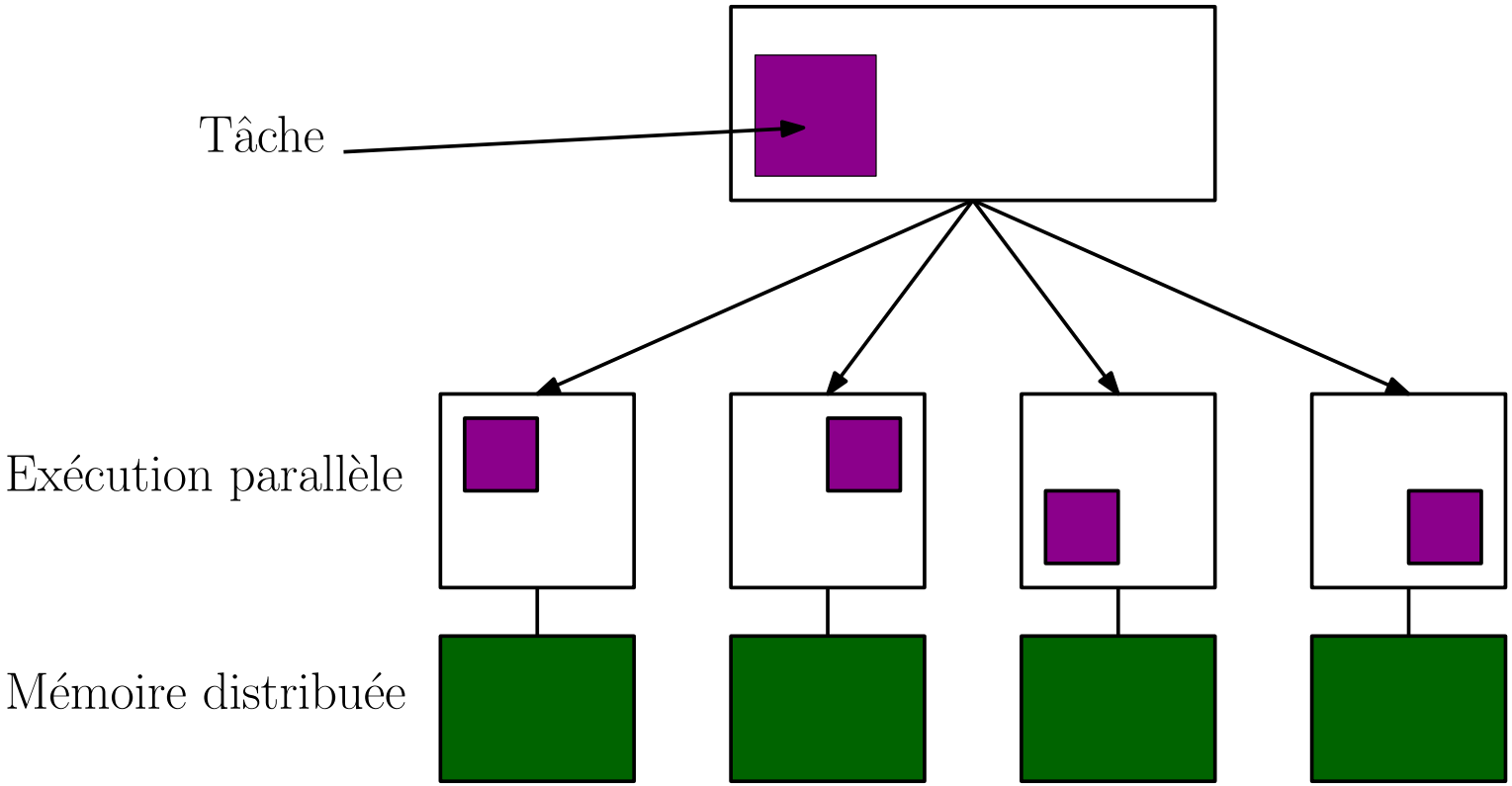
Exécution d'une tâche sur plusieurs unités de calculs

Terminologie



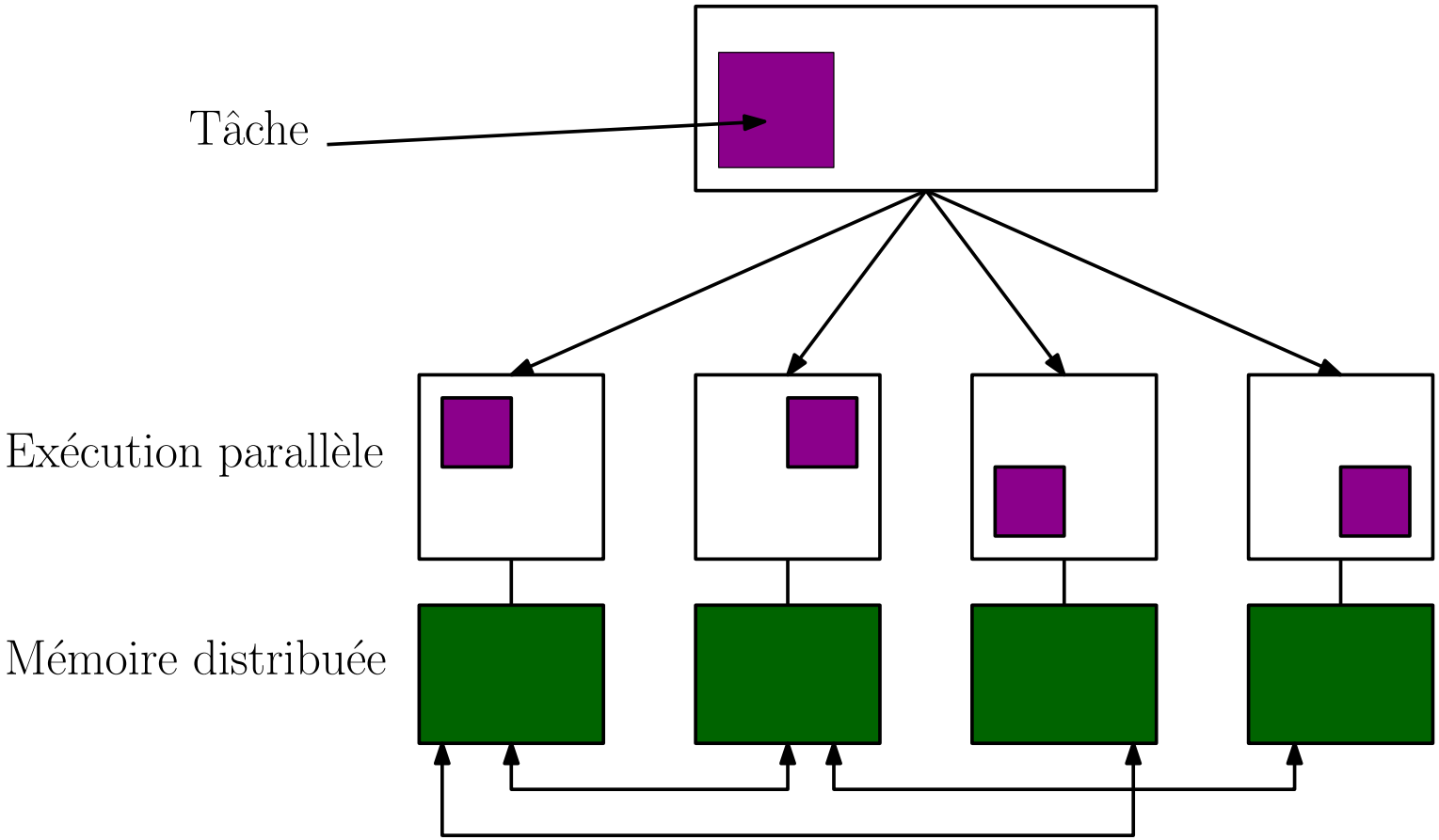
Toutes les unités de calculs partagent la même mémoire centrale

Terminologie



Chaque unité de calcul a sa propre mémoire centrale

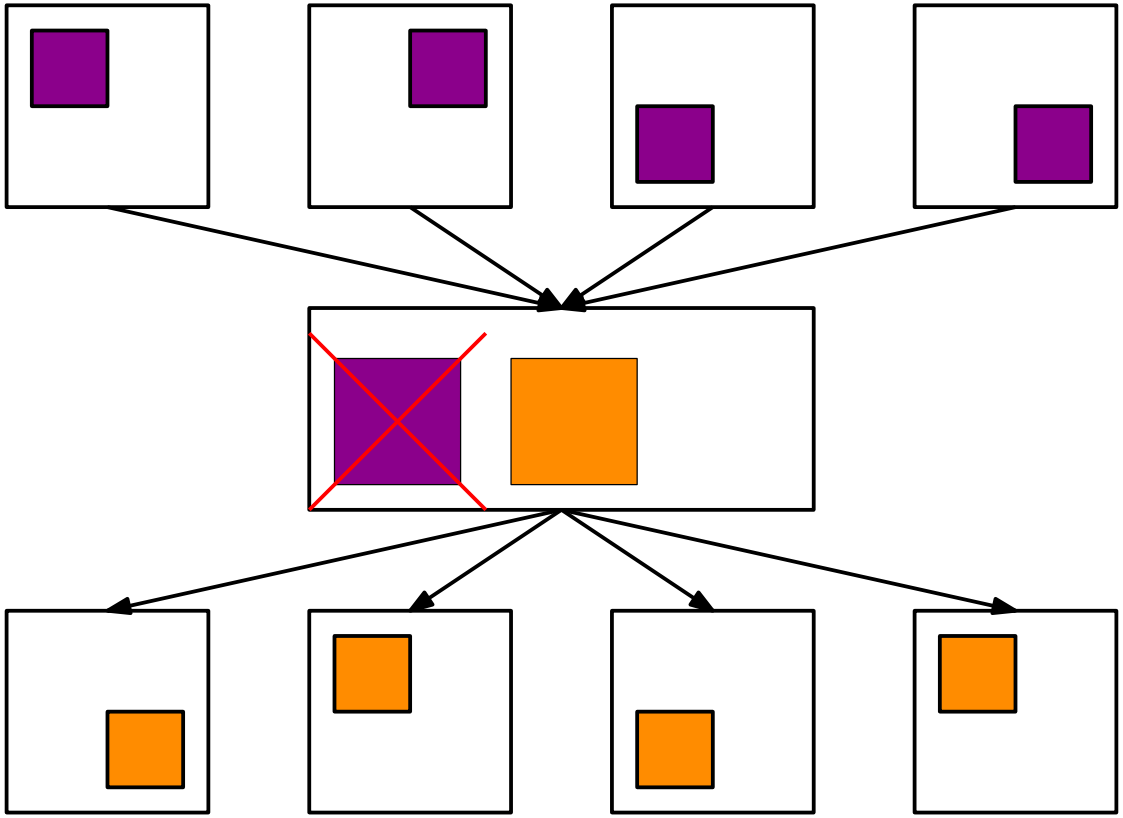
Terminologie



Communication : Échange de données entre les unités de calcul

Terminologie

Synchronisation



Synchronisation : coordination des tâches. Point où les unités doivent “s’attendre” pour poursuivre le calcul

Parallélisation efficace

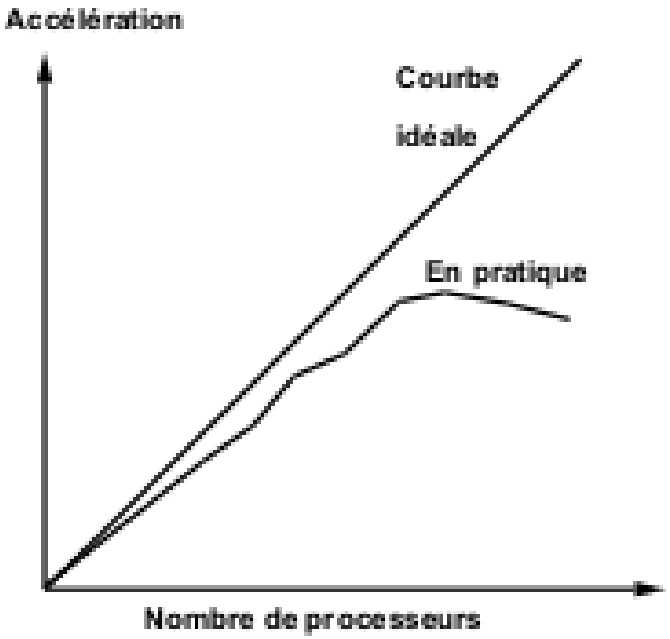
- **Accélération et efficacité** sont une mesure de la qualité de la parallélisation.
- Soit $T(p)$ le temps d'exécution sur p processeurs.
- L'**accélération** $A(p)$ **et l'efficacité** $E(p)$ sont définies comme étant :

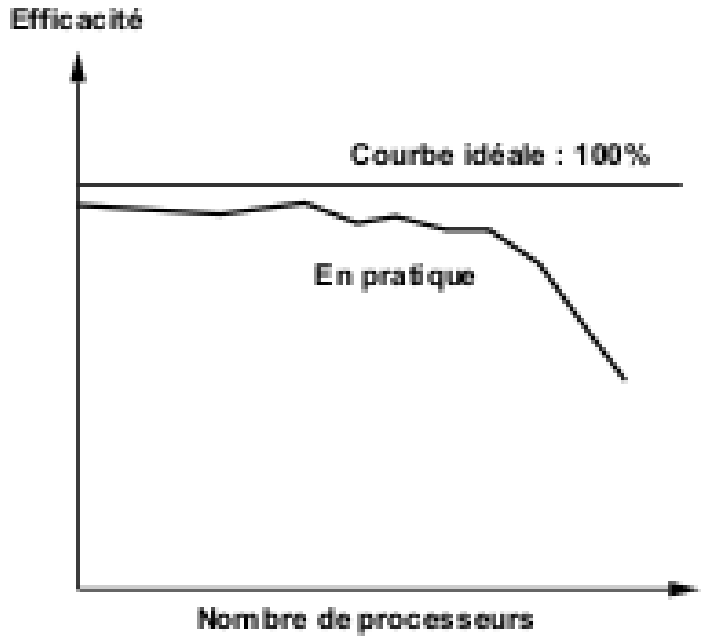
$$\begin{aligned} A(p) &= T(1)/T(p) \ \ (p = 1, 2, 3...) \\ E(p) &= A(p)/p \end{aligned}$$

- Pour une **accélération parallèle parfaite**, on obtient :

$$\begin{aligned} T(p) &= T(1)/p \\ A(p) &= T(1)/T(p) = T(1)/(T(1)/p) = p \\ E(p) &= A(p)/p = p/p = 100\% \end{aligned}$$

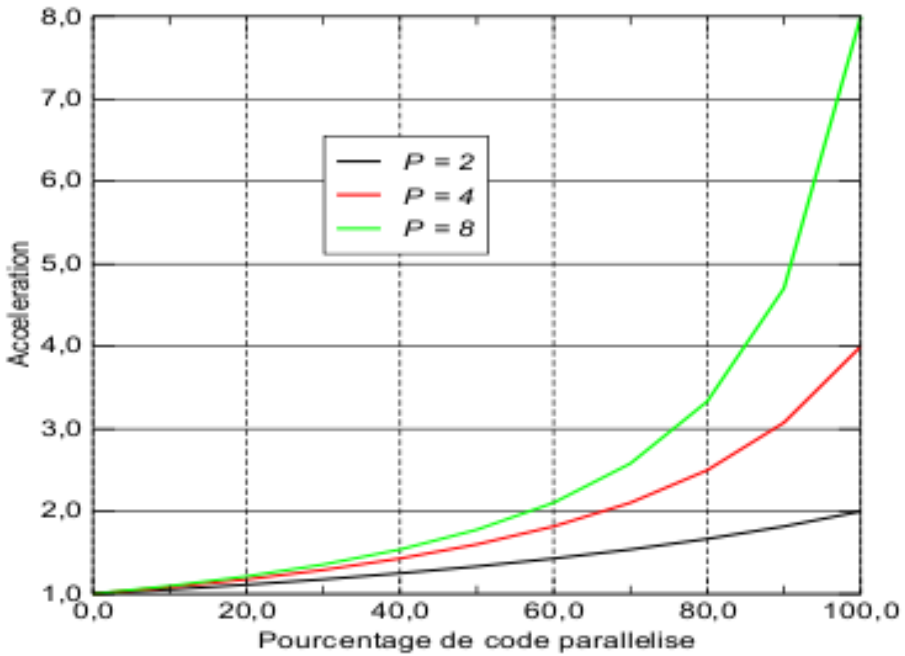
Accélération et efficacité





- Les programmes **scalables** demeurent efficaces pour un grand nombre de processeurs (scalables = passages à l'échelle).

Loi d'Amdahl

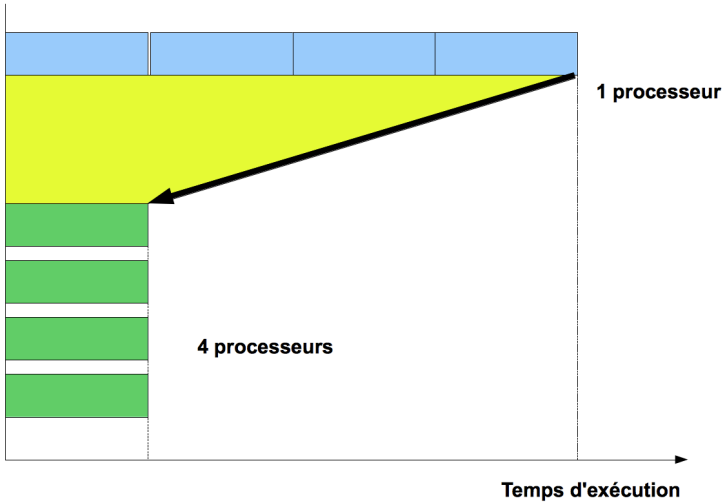


Loi d'Amdahl : **la partie non parallèle d'un programme limite les performances et fixe une borne supérieure à la scalabilité.**

1. Historique
2. Architecture générale séquentielle
3. Architecures parallèles
   - Introduction
   - Multi-coeur
   - GPU
   - Many-coeur
   - Mémoire distribuée
4. Conclusions

Multi-coeur

Processeur multi-coeur : Processeur possédant plusieurs coeurs de calcul **physiques** fonctionnant simultanément



- Architecture de la plupart des processeurs actuels !

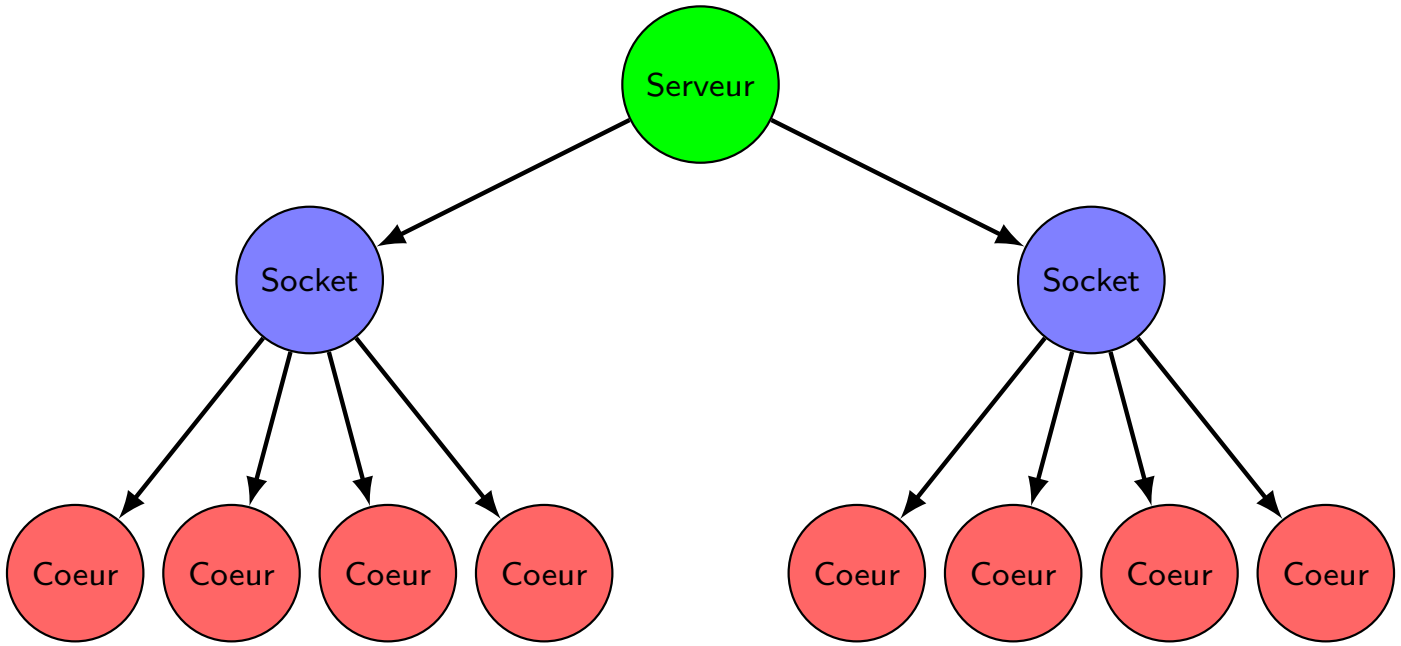
CPU, processeur, cœur, socket ?

CPU, processeur, cœur, socket ?

CPU, Processeur : Central Processing Unit

Cœur : élément du processeur capable d'exécuter des programmes de façon autonome

Socket : réceptacle du processeur sur la carte mère



Serveur bi-sockets quadri-coeurs

Fonctionnement : les points importants

- Il existe des mécanismes complexes pour augmenter les performances : **pipelining, processeur superscalaire, exécution spéculative, hyperthreading ...**

- **Instructions vectorielles** : le processeur, en fonction de son jeu d'instructions, est capable d'appliquer la même instruction simultanément à plusieurs données (SSE, AVX).

→ **Parallélisme multi-niveaux**

Pipeline

- L'exécution d'une instruction est décomposée en une **succession d'étapes**.
- Chaque étape correspond à l'utilisation d'**une des fonctions** du micro-processeur.
- Lorsqu'une instruction se trouve dans l'une des étapes, les composants associés aux autres étapes ne sont **pas utilisés**.

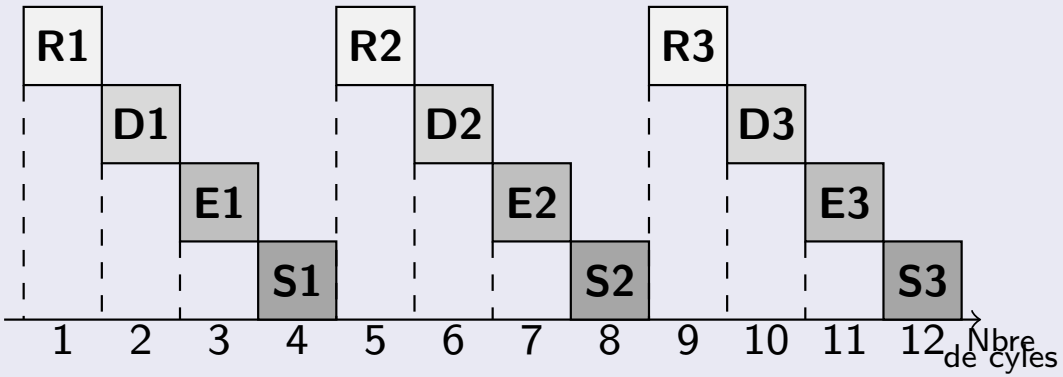
Problèmes potentiels

- Le **temps de traitement de chaque étape** doit être à peu près égal sinon les unités rapides doivent attendre les plus lentes.
- Des **"aléas"** si une instruction produit un résultat utilisé par l'instruction suivante par exemple, ou quand deux instructions ont besoin d'utiliser la même ressource.

Exemple de pipeline à 4 étages

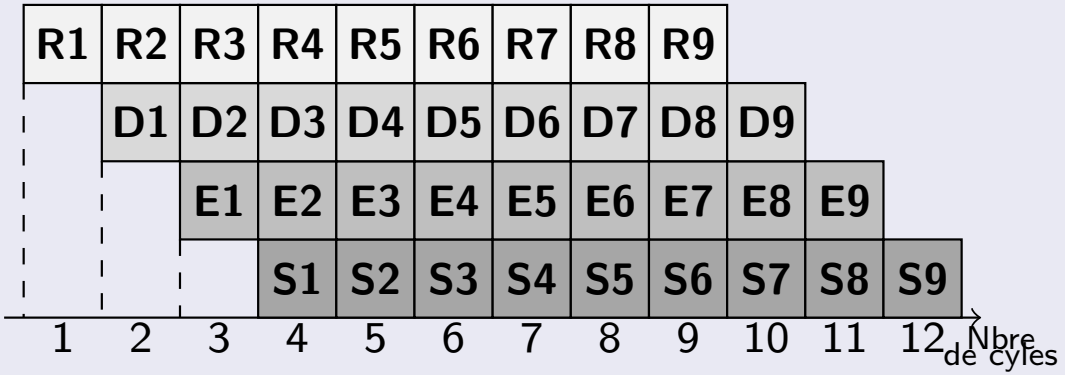
Recherche	Décodage	Exécution	Sauv. résultat

Modèle classique



Exemple de pipeline à 4 étages

Modèle pipeliné à 4 étages



- Chaque “travailleur” effectue le même travail sur des objets différents
- S’il y a 4 étapes dans la chaîne, 4 objets peuvent être traités simultanément
- Et si **ces étapes prennent le même temps**, la chaîne est continuellement occupée

Hyperthreading

- **SMT = Simultaneous MultiThreading**, plus connu sous le nom d'Hyperthreading chez Intel.
- **Principe** : créer deux cœurs logiques sur une seule puce, chacun doté de ses propres registres. Ces deux unités partagent les éléments du cœur physique comme le cache et le bus système.
- Dégradation des performances individuelles des threads mais amélioration des performances de l'ensemble.

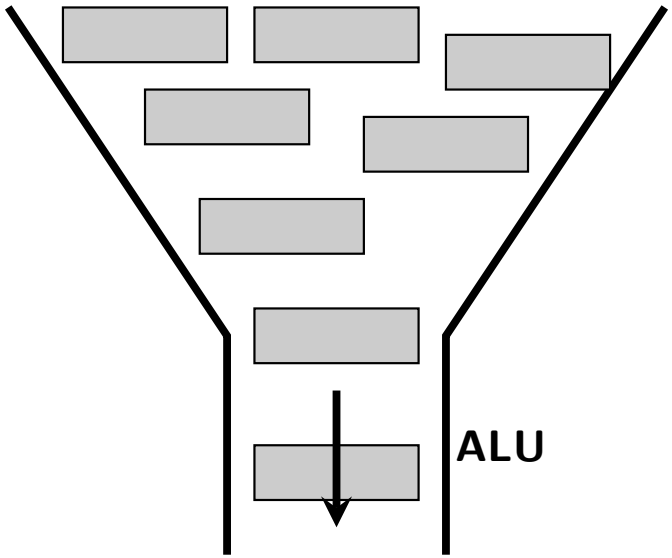
Attention

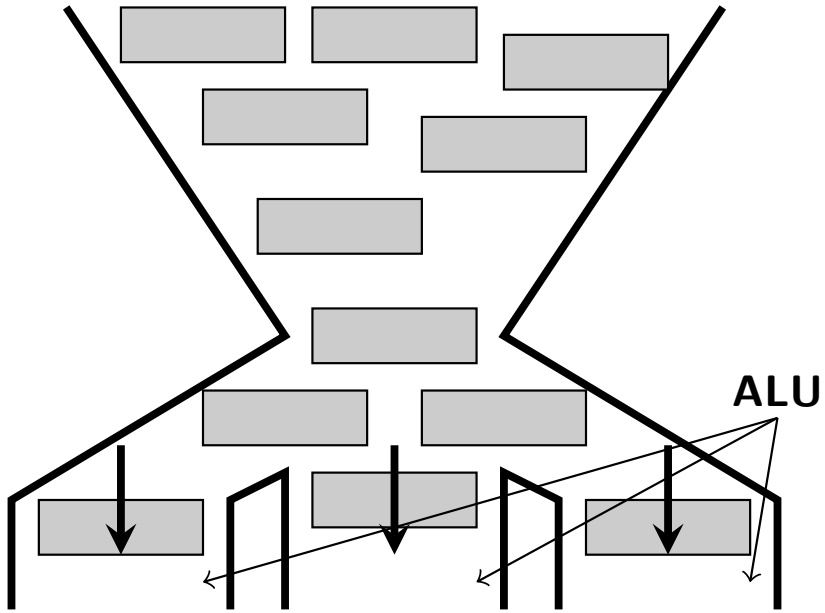
Ce fonctionnement est généralement activé par défaut.

Architecture superscalaire

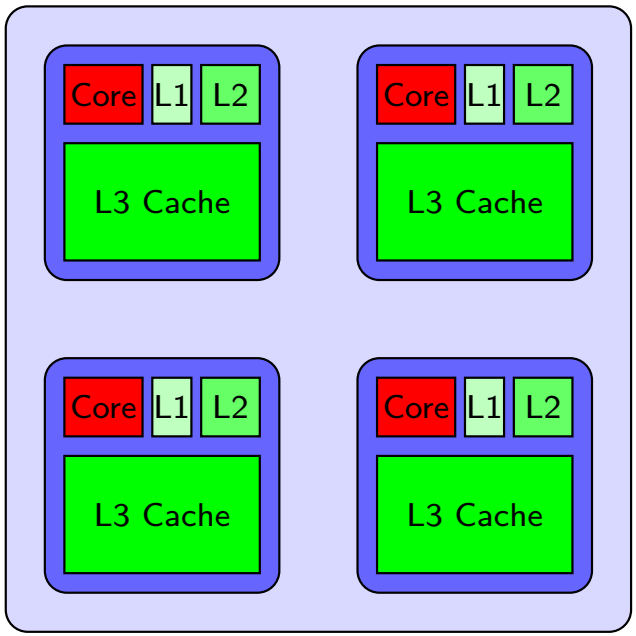
Principe

- Exécuter **plusieurs instructions en même temps**.
- Doter le micro-processeur de **plusieurs unités de traitement** travaillant en parallèle
- Nécessite un **flot de données/instructions important**

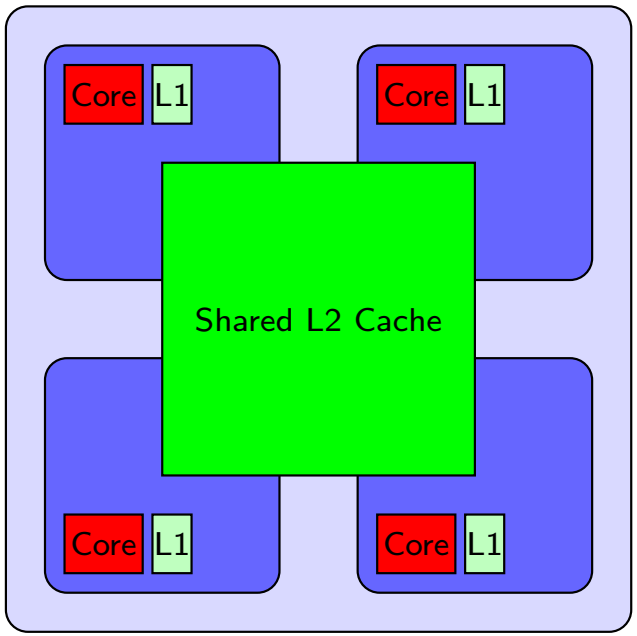




Organisation des caches



Caches indépendants



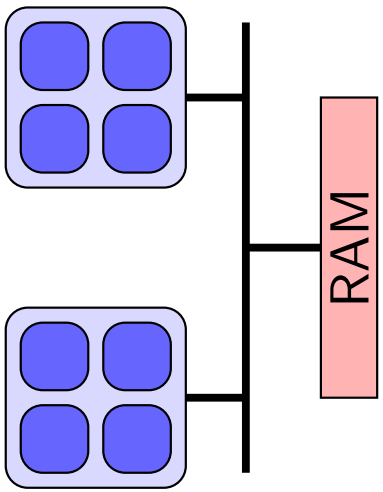
Caches partagés

- Problème de **cohérence**
- Problème d'**accès concurrent**

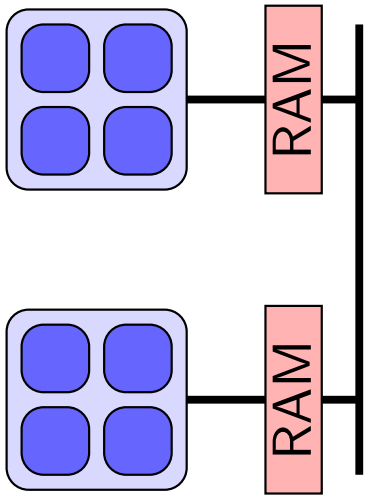
UMA/NUMA

- **Modèle NUMA (Non Uniform Memory Access)** : les processeurs sont capables d'accéder à la totalité de la mémoire du système mais les temps d'accès mémoire sont non uniformes en fonction de la distance mémoire/cœur.

Architecture UMA



Architecture NUMA



Balance

Avantages

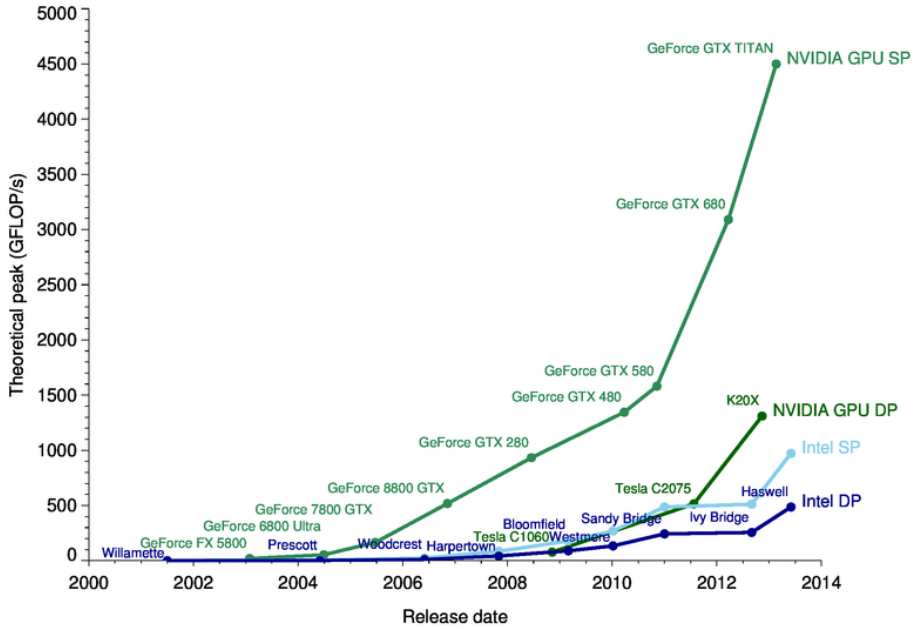
- Mémoire globale ⇒ facile à programmer
- Mémoire proche des CPU ⇒ accès rapide

Inconvénients

- Gros trafic de données ⇒ mauvaise scalabilité
- Programmeur : gérer la synchronisation (mémoire globale)
- Très coûteux !

1. Historique
2. Architecture générale séquentielle
3. Architecures parallèles
   - Introduction
   - Multi-coeur
   - GPU
   - Many-coeur
   - Mémoire distribuée
4. Conclusions

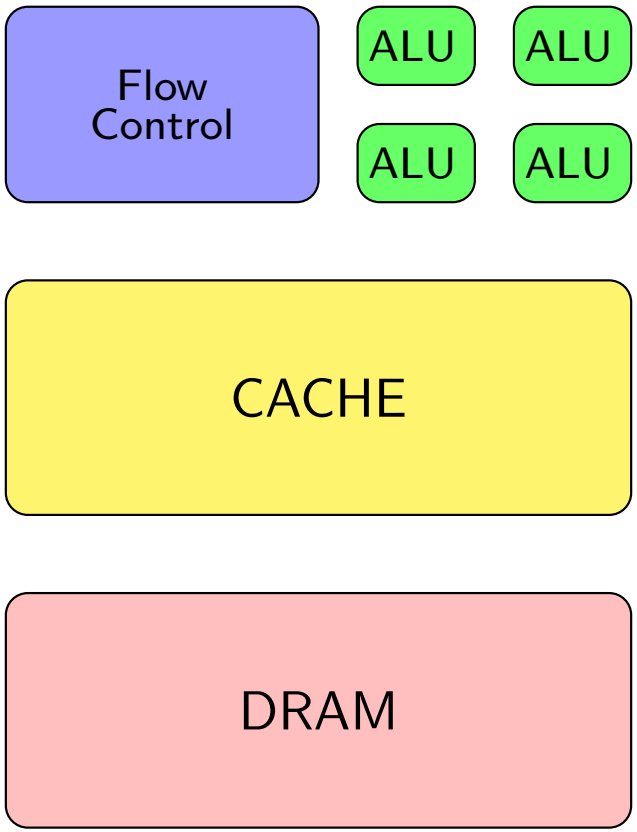
Pourquoi les GPU?



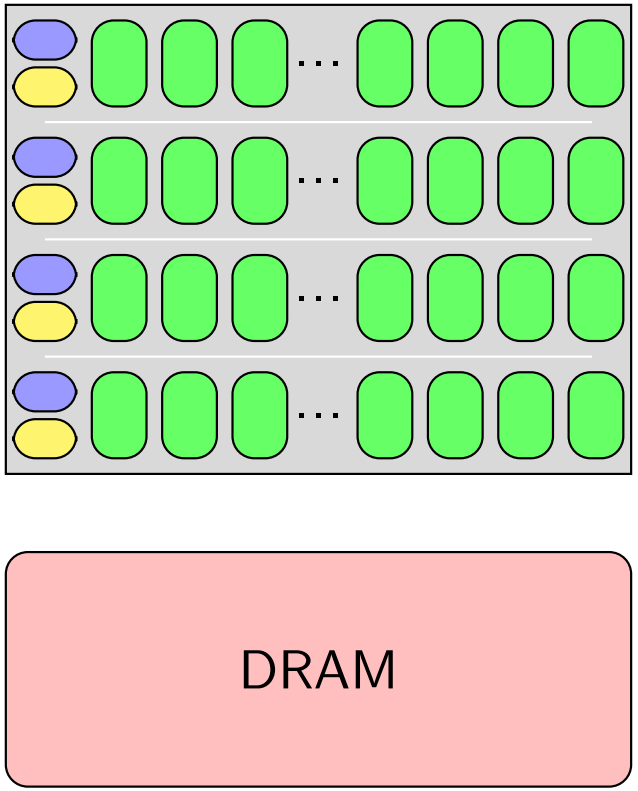
À **performance théorique** égale, les plateformes à base de GPU:

- occupent moins de place
- sont moins chères
- sont moins consommatrices d'électricité

Architecture comparée d'un CPU et d'un GPU



CPU

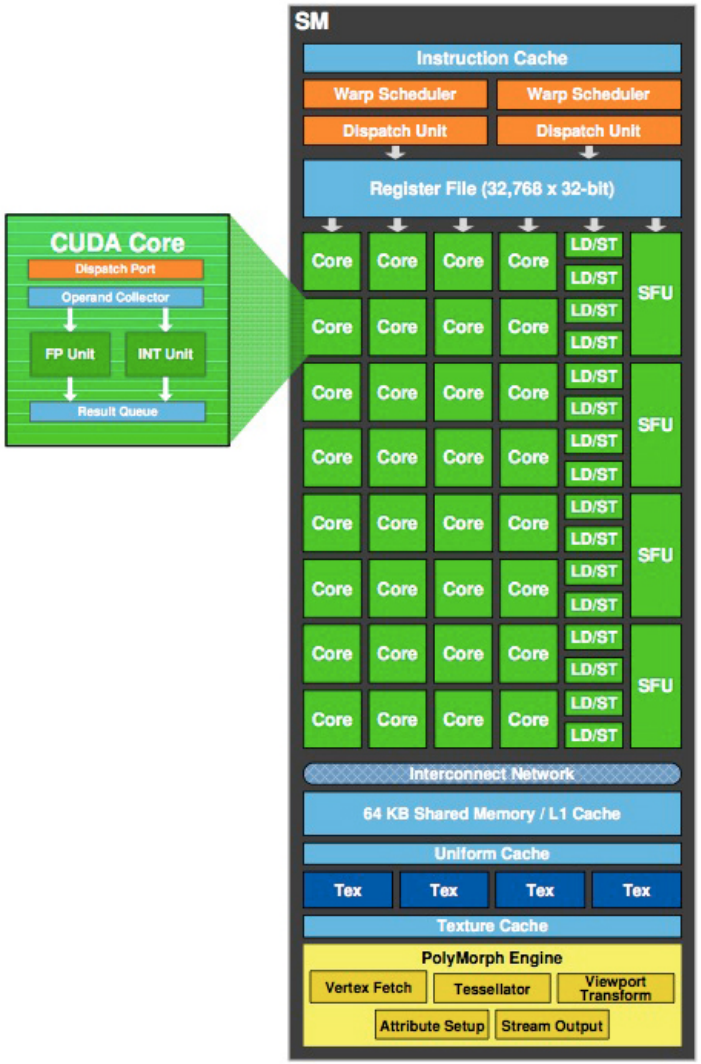


GPU

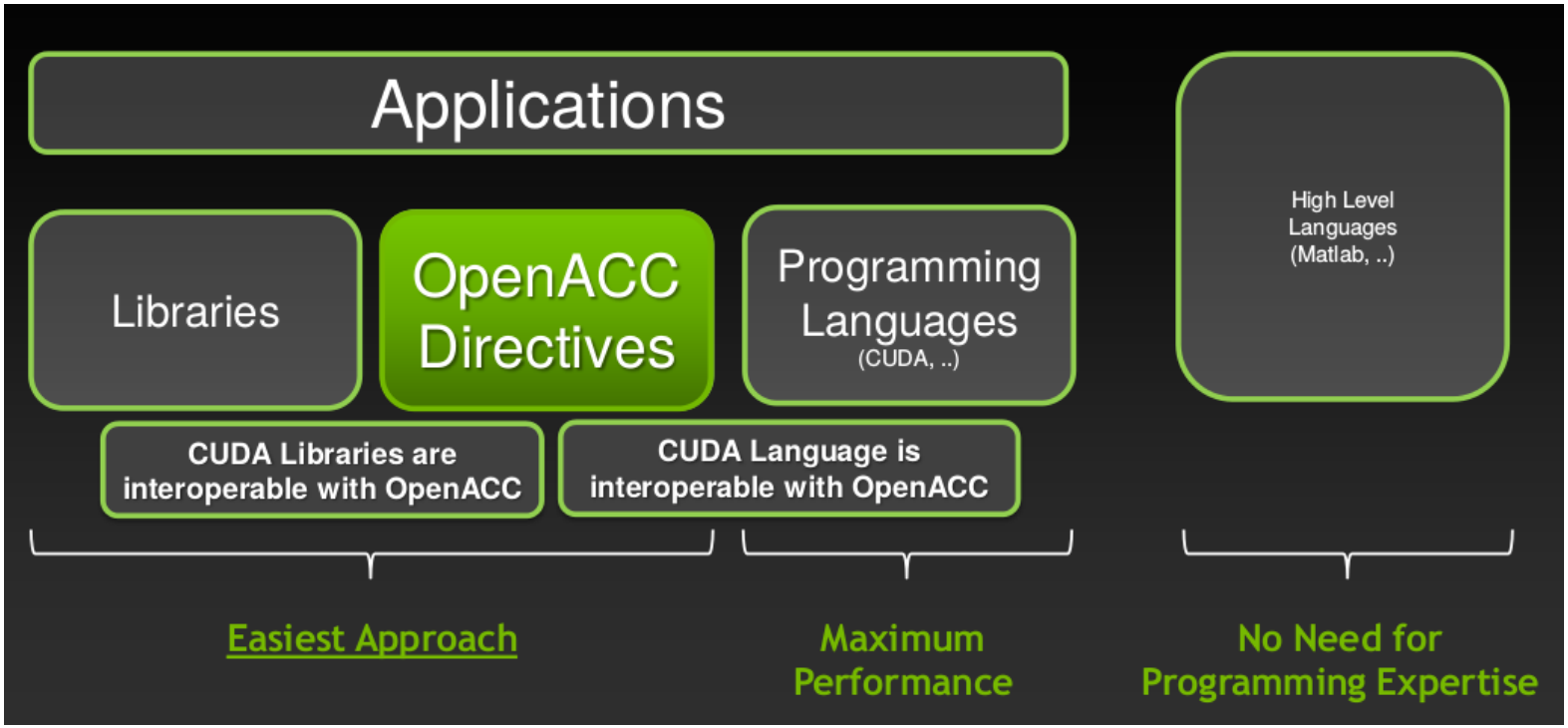
Caractéristiques d'un GPU

Deux composants principaux :

- **Mémoire globale** : similaire à la RAM du CPU, accessible à la fois par le GPU et le CPU.
  - Actuellement, jusqu'à 16 GB par GPU.
  - Bande passante jusqu'à 250 GB/s
- **Streaming Multiprocessors (SMs)** : en charge des calculs. Chaque SM possède ses propres :
  - unités de contrôle
  - registres
  - pipelines d'exécution
  - caches



Accélérer les applications grâce au GPU



Balance

Avantage

- Milliers/millions d'unités de calcul
- Moindre coût

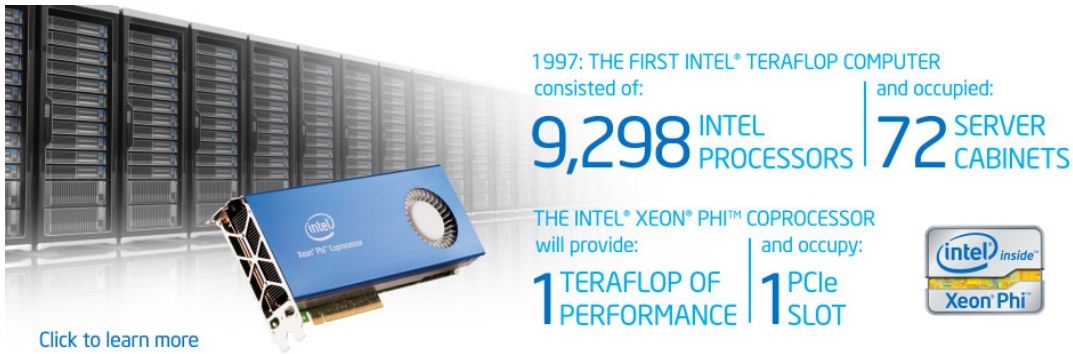
Inconvenients

- Milliers/millions d'unités de calcul
- Complexe à exploiter (gestion mémoire)
- Tranferts CPU/GPU coûteux

1. Historique
2. Architecture générale séquentielle
3. Architecures parallèles
   - Introduction
   - Multi-coeur
   - GPU
   - Many-coeur
   - Mémoire distribuée
4. Conclusions

Many-coeur

- Apparition sur le marché de cartes **Many Integrated Core** : les Xeon Phi©
- Intel Xeon Phi 5110P: 72 coeurs à 1500 MHz, 16 Go de mémoire



Spécificités

Une architecture x86 native - en fait à la fois un complément idéal à Xeon et un concurrent crédible aux accélérateurs GPU.

Programmer sur le Phi

- La carte exécute son **propre OS** (adresse IP propre)
  communication : bus PCIe
- Connexion : ssh $\Rightarrow$ un serveur classique.
- Deux modes d'exploitation :
  - déporter les parties parallèles sur le Phi (openMP) : **offload**
  - **exécuter nativement** tout le code sur le Phi
- **72 cœurs, 288 threads**
- 32 Mb Cache L2

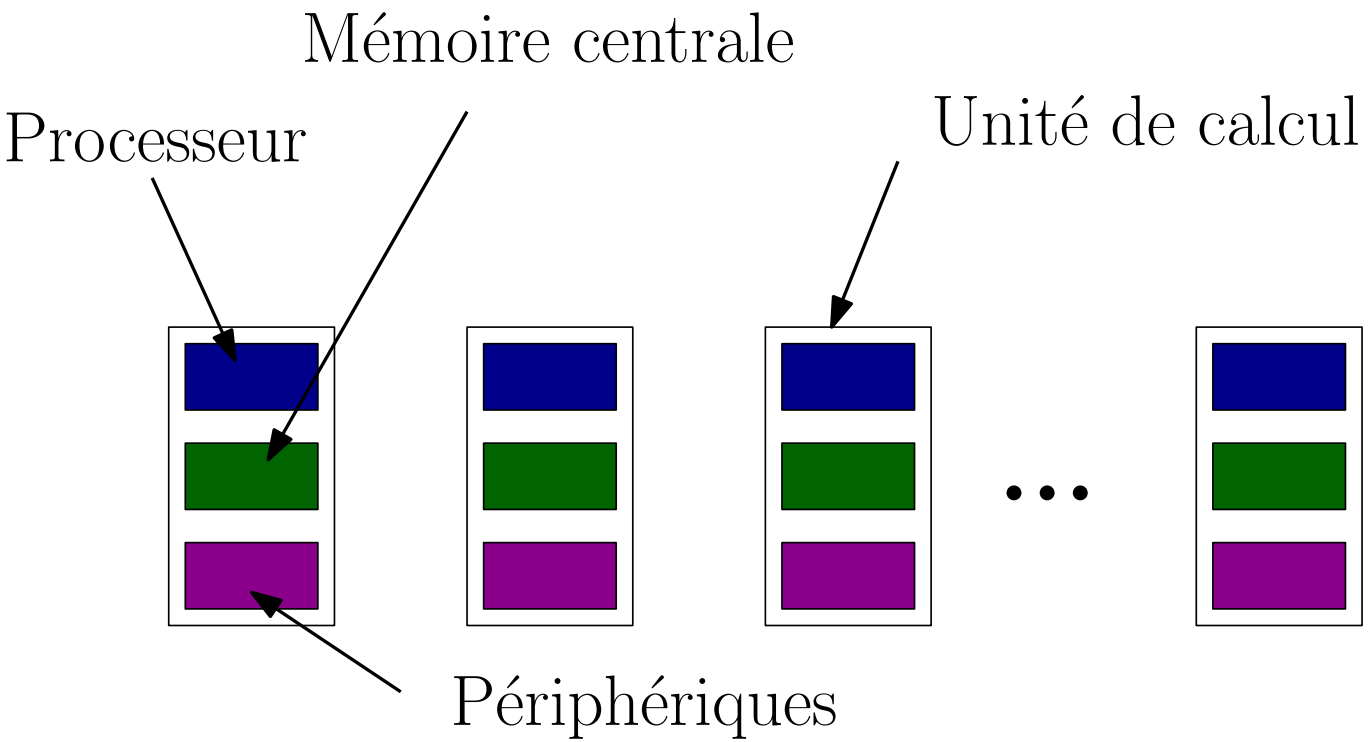
Pour exploiter efficacement un Phi

- Application **massivement** parallèle
- Boucles **vectorisées** (alignement des données)
- **Localité** des données optimisée

1. Historique
2. Architecture générale séquentielle
3. Architecures parallèles
   - Introduction
   - Multi-coeur
   - GPU
   - Many-coeur
   - Mémoire distribuée
4. Conclusions

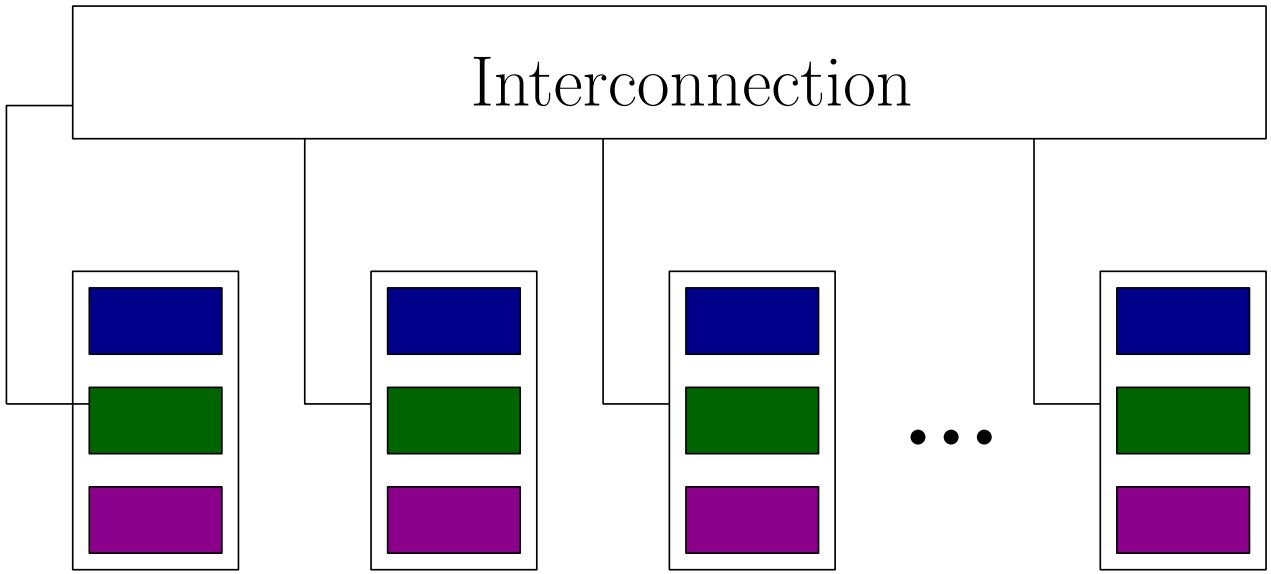
Architecture à mémoire distribuée : exemples

Architecture à mémoire distribuée



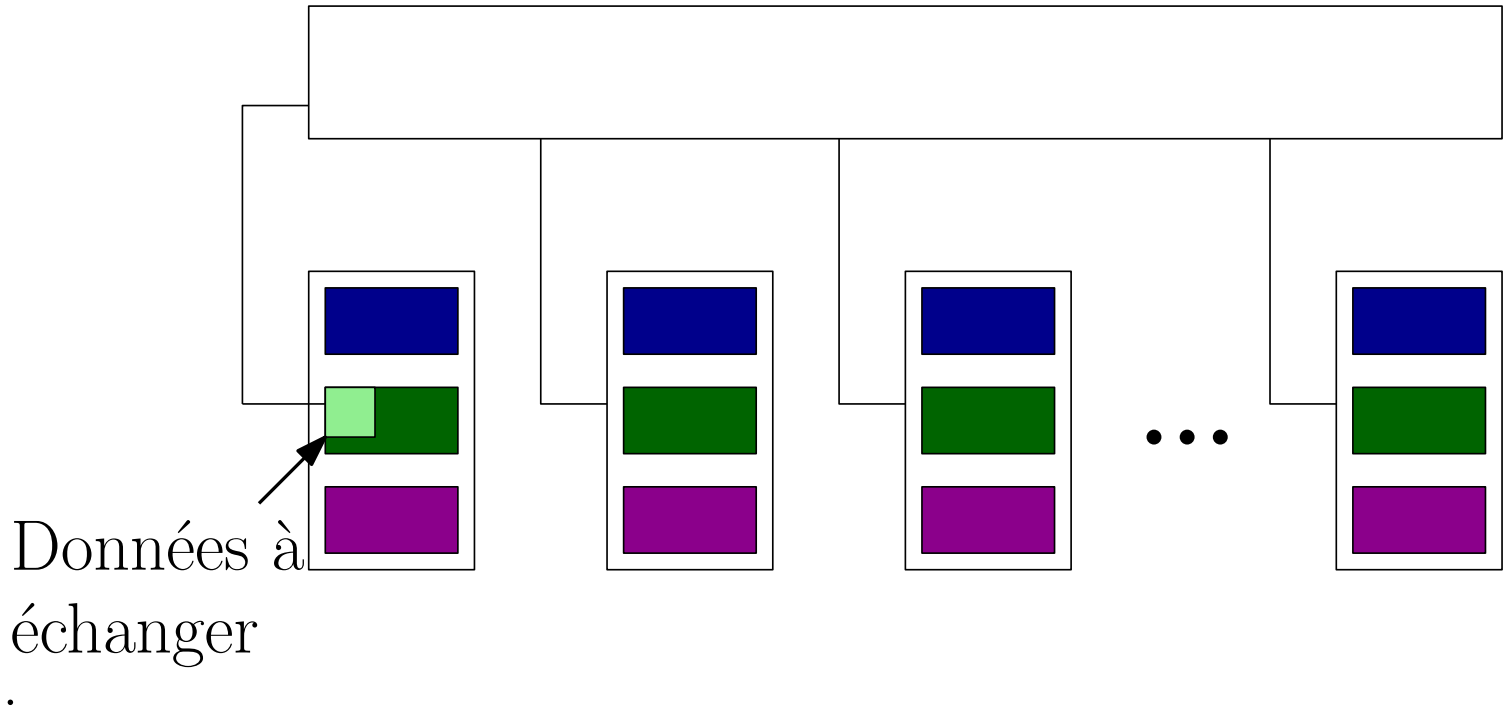
On dispose de N “ordinateurs” ... travaillant sur la même tâche

Architecture à mémoire distribuée



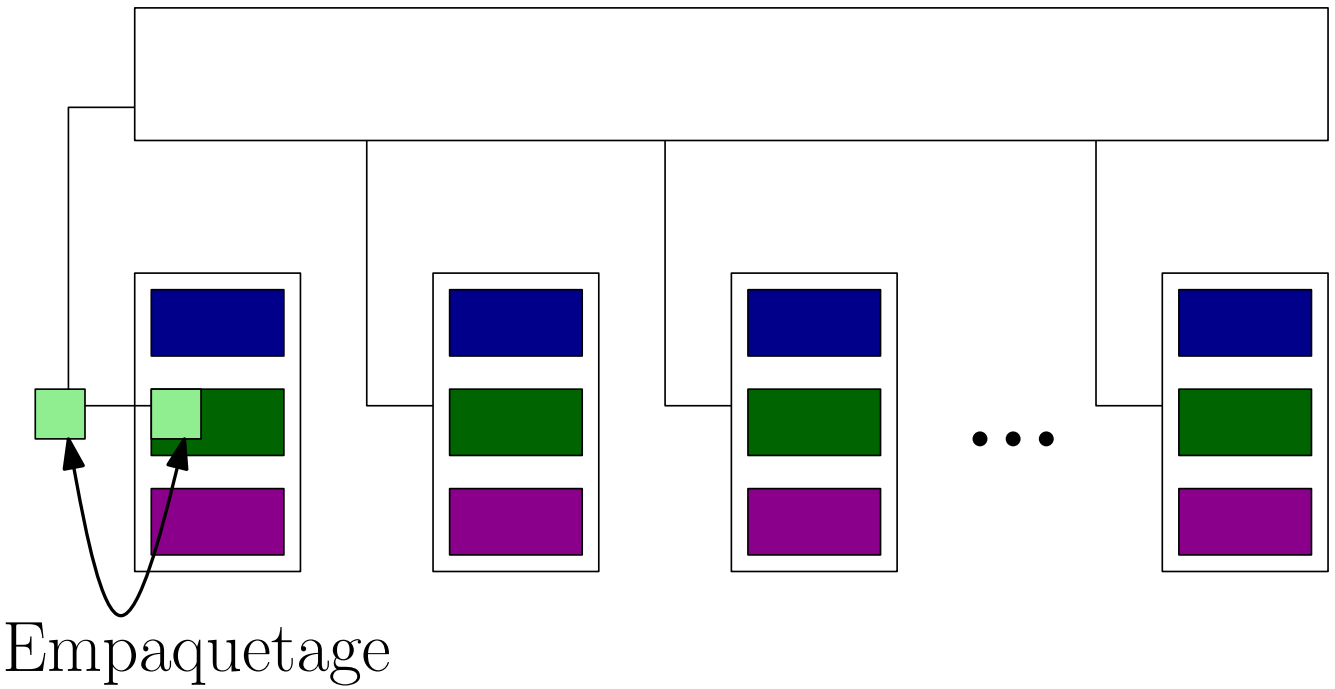
On dispose de N “ordinateurs” ... reliés par un réseau !

Architecture à mémoire distribuée



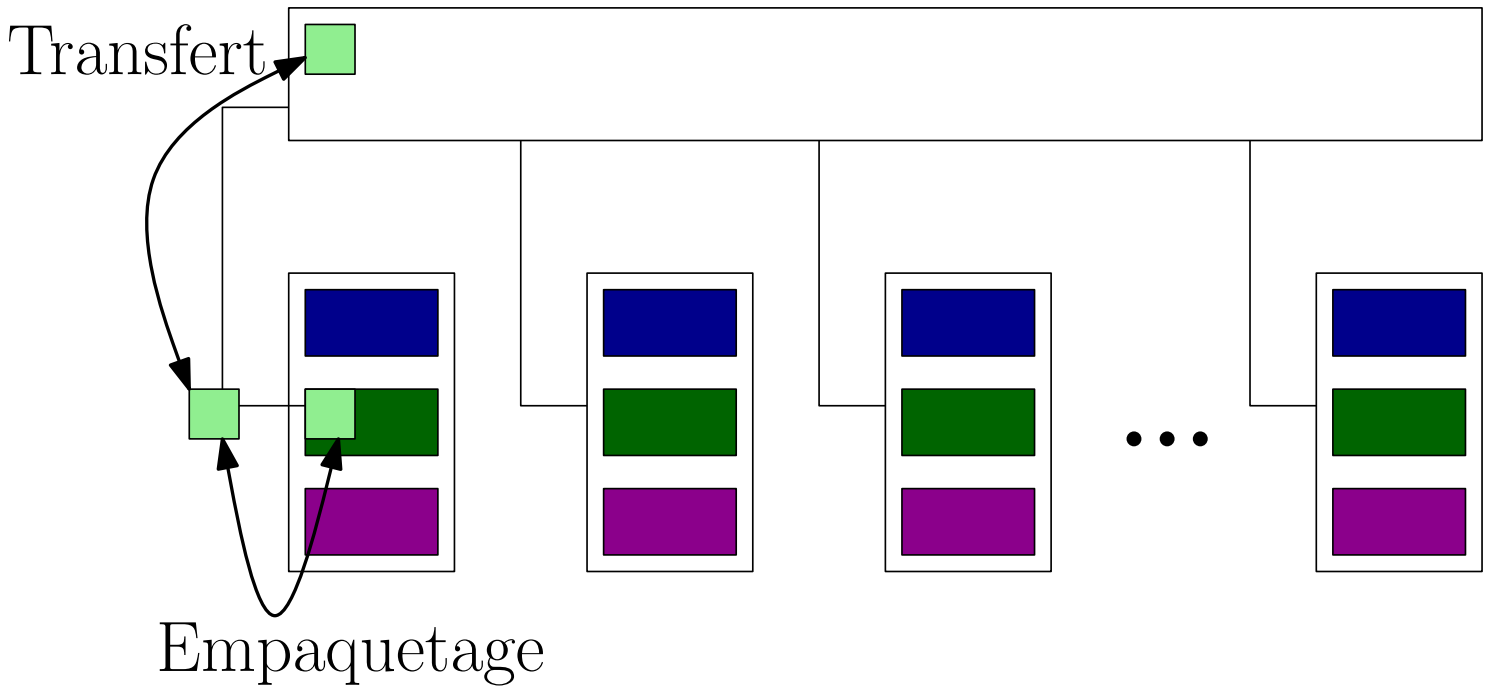
Communication : données à échanger

Architecture à mémoire distribuée



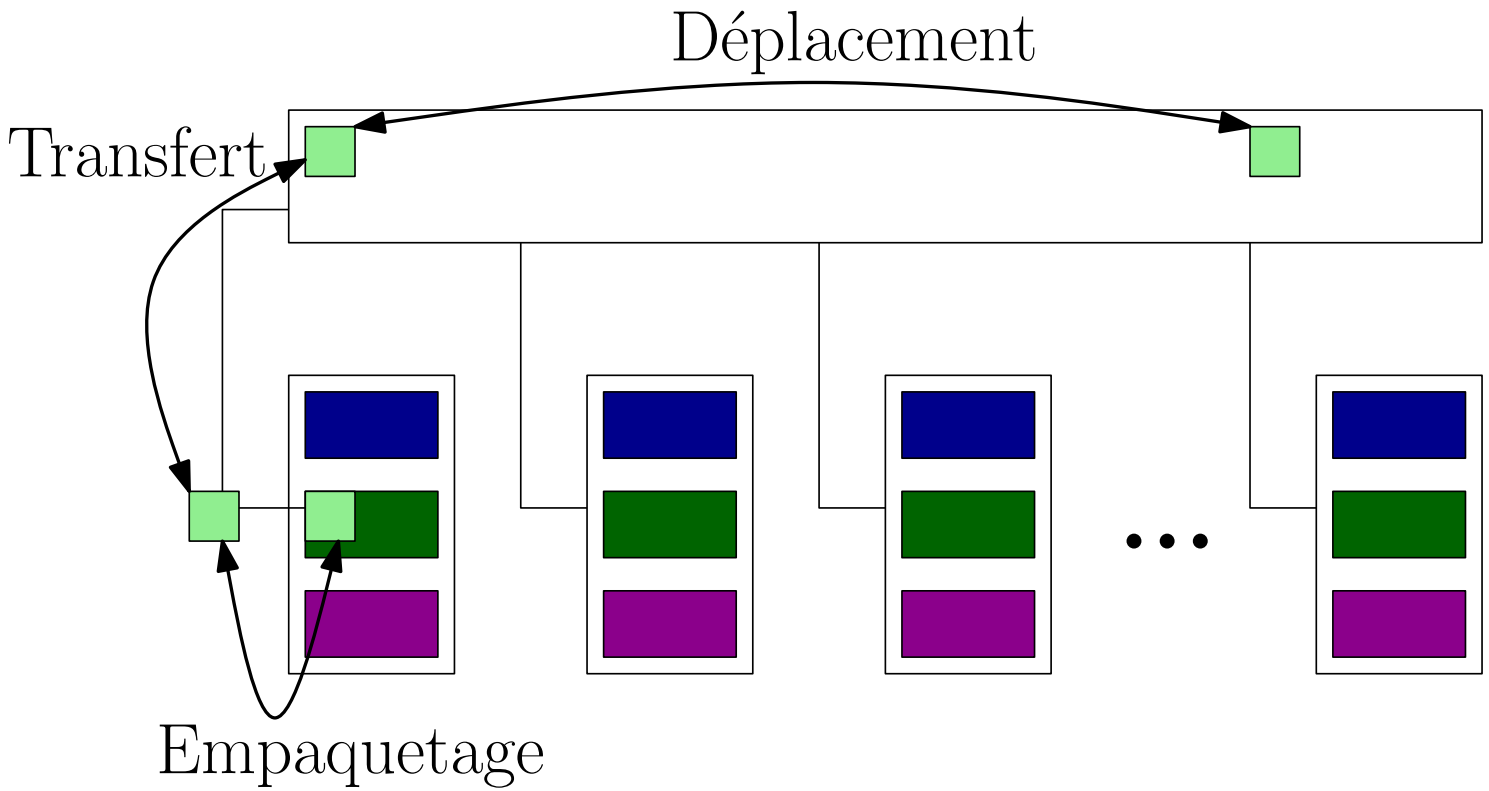
Communication : données à échanger ⇒ empaquetées (envoyeur)

Architecture à mémoire distribuée



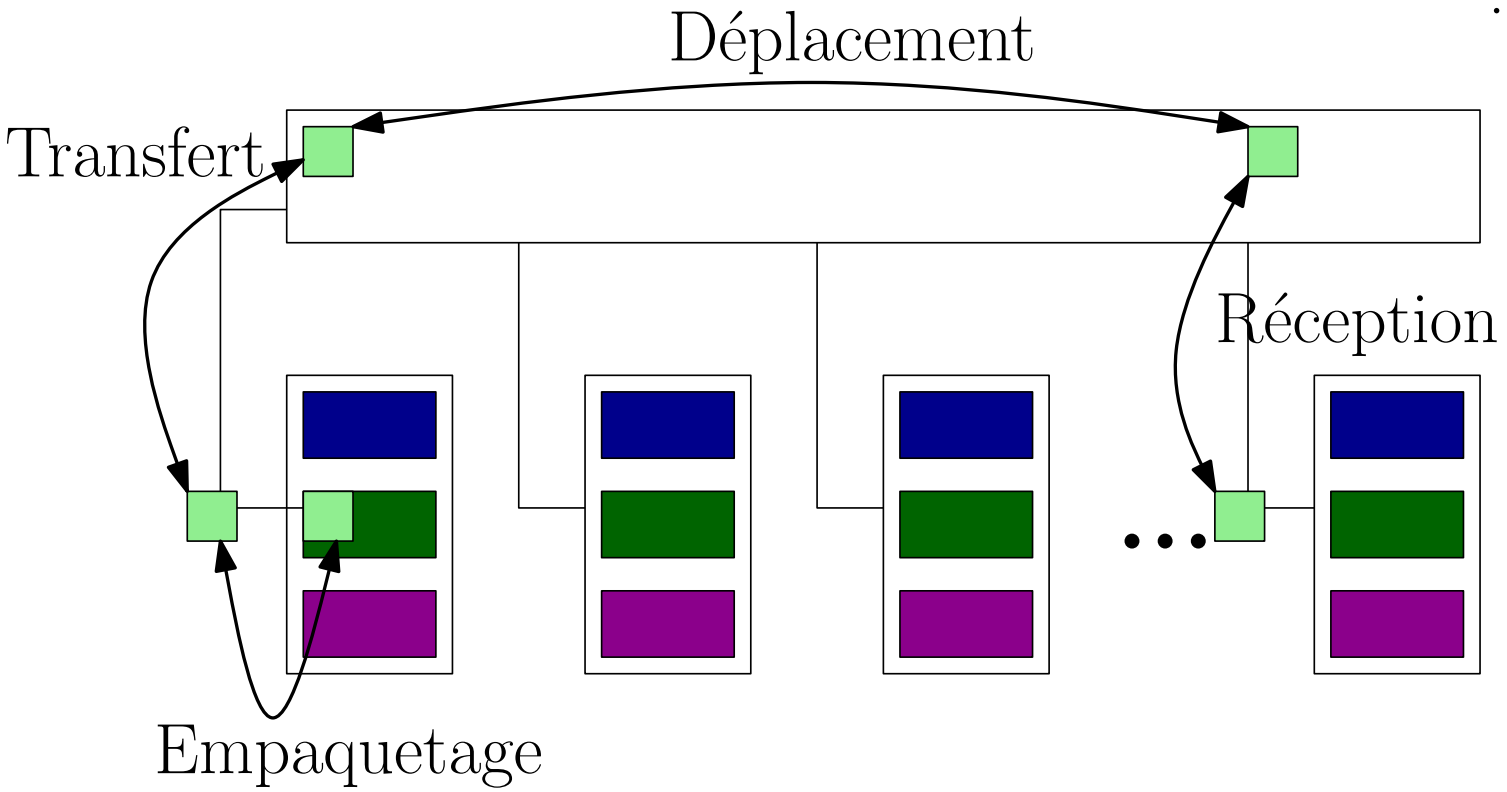
Communication : données à échanger $\Rightarrow$ transférées sur le réseau

Architecture à mémoire distribuée



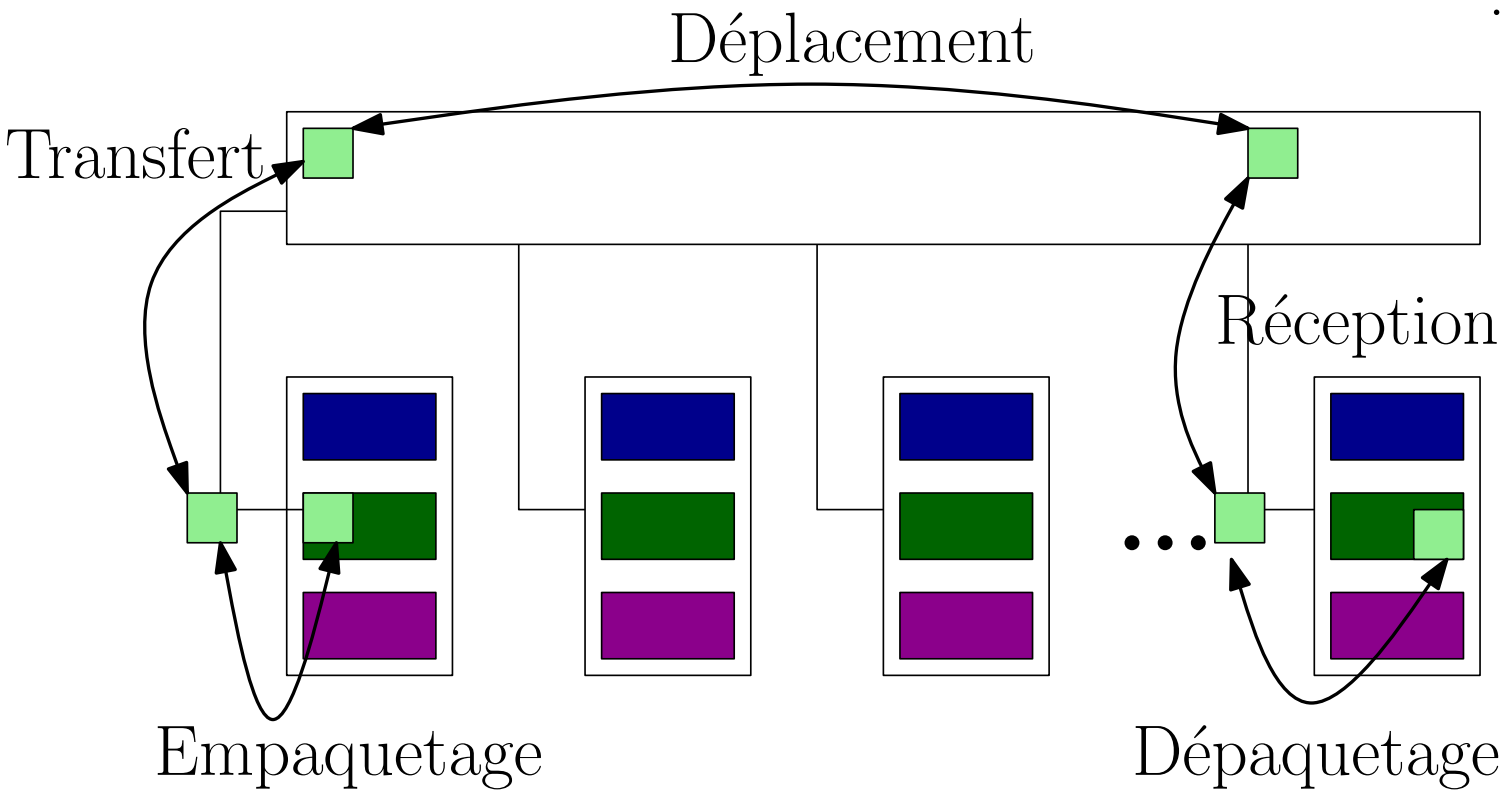
Communication : données à échanger ⇒ déplacées sur le réseau

Architecture à mémoire distribuée



Communication : données à échanger ⇒ réceptionnées

Architecture à mémoire distribuée



Communication : données à échanger ⇒ dépaquetées/stockées

Balance

Avantages

- Puissance de calcul élevée
- Augmentation proportionnelle des CPU et de la mémoire
- Accès rapide à la mémoire locale
- Coût raisonnable

Inconvénients

- Programmation complexe : gestion explicite mémoire, échange, synchronisation
- Temps d'accès à la mémoire non-locale très lent

Architecture mémoire hybride

- La plupart des calculateurs actuels sont aujourd'hui un **mixte de mémoire partagée et mémoire distribuée**.
- La brique de base (noeud) est un multiprocesseur à **mémoire partagée**.
- Ces briques sont interconnectées par un réseau (type ethernet, Infiniband, ...).
- À noter qu'on trouve aussi de l'hybride entre CPU et cartes accélératrices.

Conclusions

Fréquence d'horloge, nombre de cœurs, nombre d'unités fonctionnelles, fréquence et capacité mémoire, caches, stockage, interconnexions internes et réseaux ...

Tous les éléments ont leur importance ; l'architecture doit être équilibrée pour que l'ensemble soit performant

Au niveau technologique

- Beaucoup de **cœurs**
- Des cœurs **hybrides** CPU / cartes accélératrices
- Une différence entre la vitesse de calcul et les **transferts mémoire et I/O**

Conclusions

Au niveau applicatif

- Nécessité d'une **bonne connaissance des architectures**, notamment au niveau de la mémoire.
- Plus de croissance des performances au niveau du cœur $\Longrightarrow$ **parallélisation obligatoire** pour un gain de performance.
- Nécessité d'élaborer des algorithmes et des programmes capables d'exploiter un **grand nombre de processeurs**.
- Nécessité d'exploiter le parallélisme aux différents **niveaux du hardware**.
- Programmation sur des **architectures hybrides** avec différents types de processeurs.