Programmation parallèle

Cycle 2020-2021

ECE – Ing5 – Systèmes Embarqués

Etienne Hamelin & Alexandre Berne

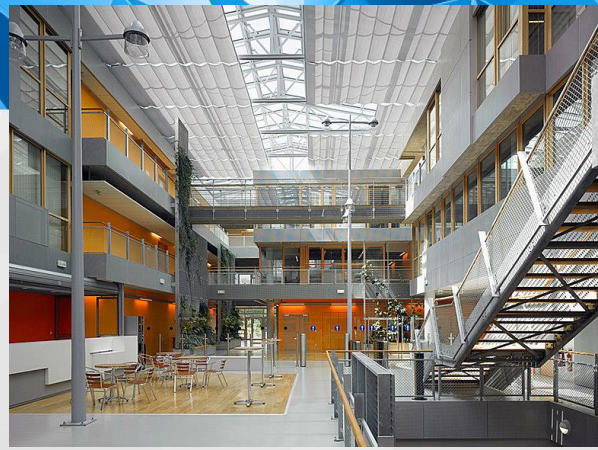
ehamelin@inseec-edu.com

aberne@inseec-edu.com

FAISONS CONNAISSANCE

Etienne HAMELIN – ehamelin@inseec-edu.com
Alexandre BERNE – aberne@inseec-edu.com

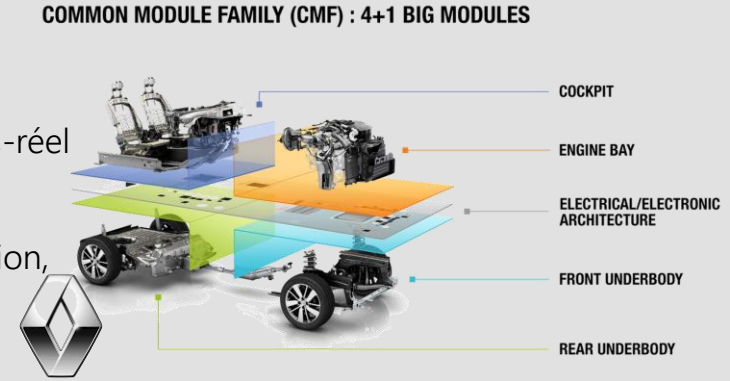
Quelques mots sur moi



- Etienne HAMELIN
  - Ingénieur Polytechnique & Télécom Paristech; CEA depuis 2011
- Alexandre BERNE
  - Ingénieur Systèmes Embarqués, CEA depuis 2011
- CEA LIST « Laboratoire d'Intégration des Systèmes & des Technologies »
  - ~750 permanents, sur le plateau de Saclay
  - Recherche appliquée en robotique, systèmes embarqués (HW/SW), …

  → nombreuses opportunités de stages, thèses, CDD, CDI !
  https://www.emploi.cea.fr/ ; https://www.theses-postdocs.cea.fr/

- Exemples de projets logiciel embarqué
  - RTOS, génération de code, paradigmes de programmation temps-réel
  - Projets R&D collaborative et pré-industrielle
  - Exemple: architecture SW pour les systèmes industriels, ou pour véhicule du futur Renault: SoCs multicoeurs, manycore, virtualisation, temps-réel safety & cybersecurity…



Et vous?

Planning prévisionnel

- Introduction & concepts
  - TP: Pthreads
- Cohérence & consistance mémoire
  - TP: OpenMP
- Equilibrage, placement des données
  - TP: Synchronisations
- Présentation du projet
  - Projet

Quelques ressources

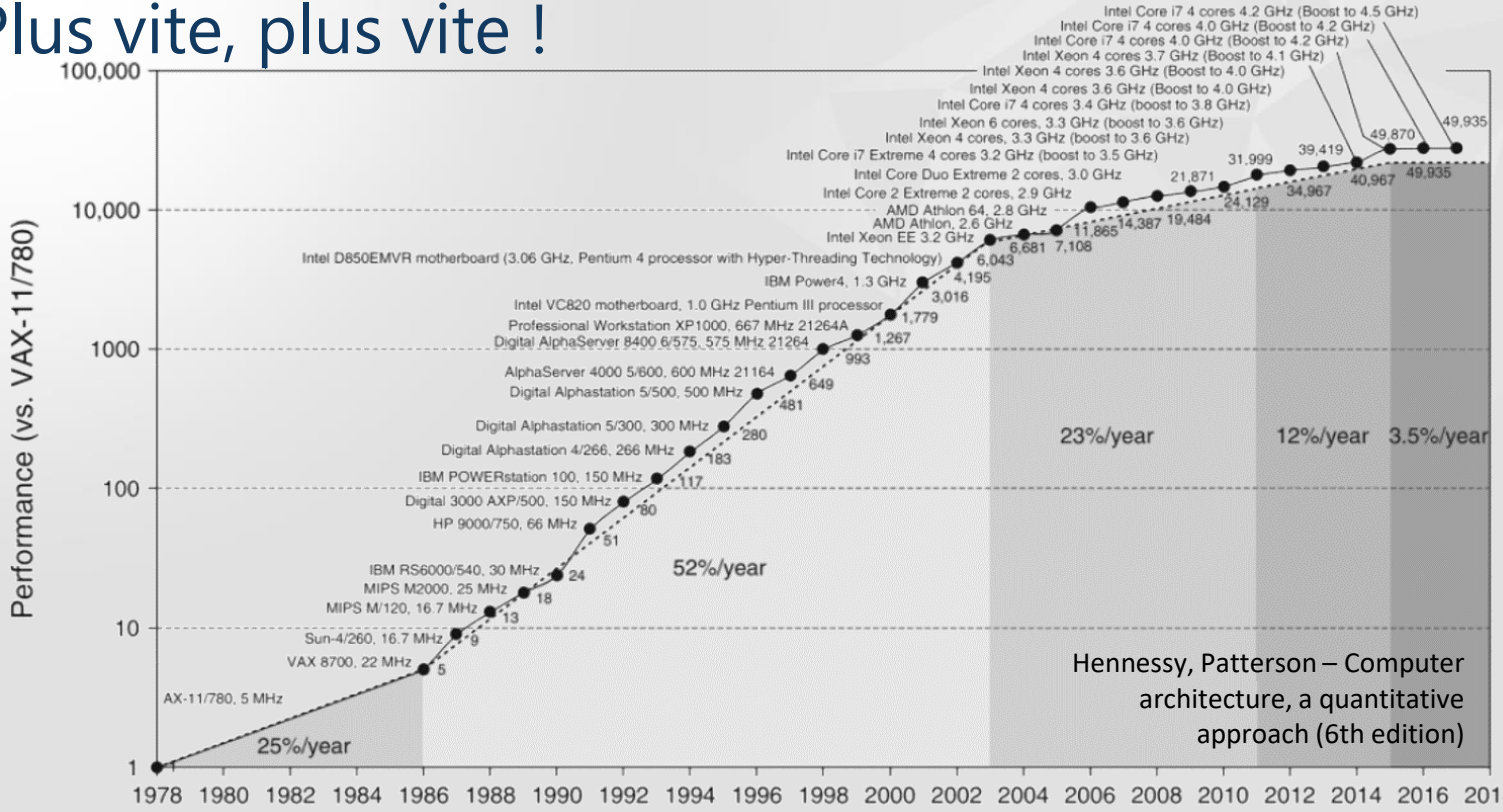
- Tutoriel sur la programmation parallèle https://computing.llnl.gov/tutorials/parallel_comp/
- Tutoriel Pthreads https://computing.llnl.gov/tutorials/pthreads/
- Site de la bibliothèque OpenMP http://openmp.org/
- Tutoriel OpenMP https://computing.llnl.gov/tutorials/openMP/
- Ouvrages de référence : « *Computer Architecture : A quantitative approach* », 6ème édition, Hennessy and Patterson

Séance 1

Introduction & Concepts

- 1960: mainframes
- 1970: « mini » ordinateur
- 1980: micro-ordinateur, PC
- 1990: Internet, téléphone cellulaire, PDA
- 2000: Web2.0, smartphones, 2G, systèmes embarqués
- 2010: IA, IoT, 3G/4G
- 2020: 5G, …?

Plus vite, plus vite !



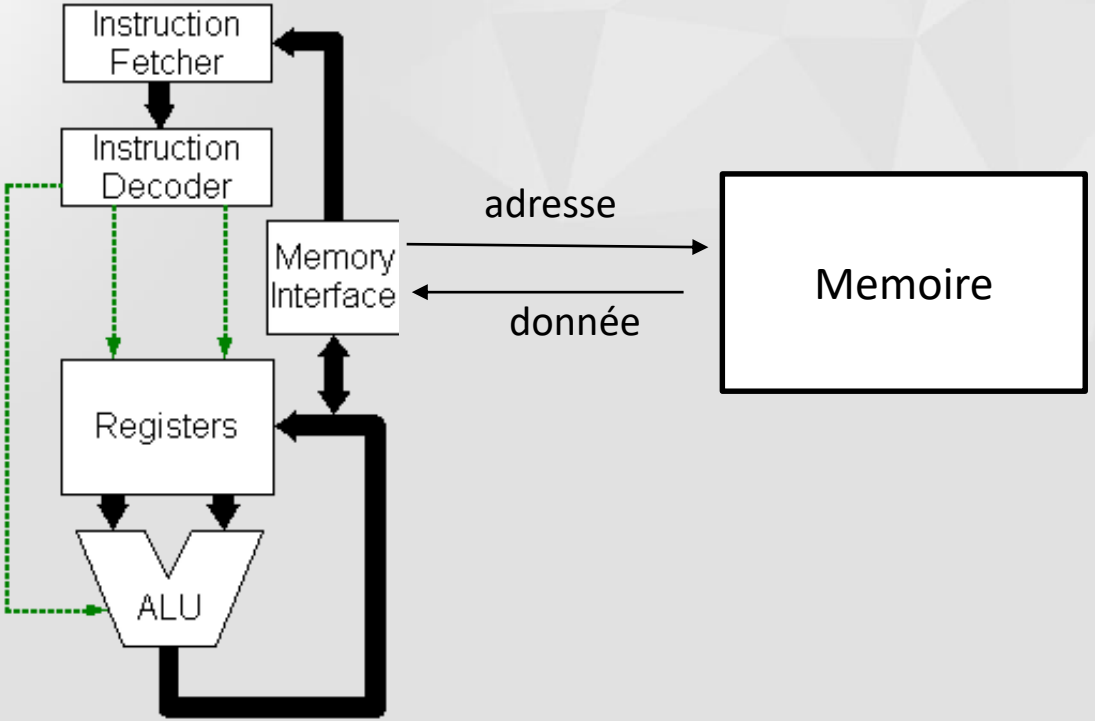
Hennessy, Patterson – Computer architecture, a quantitative approach (6th edition)

Limites techniques

- Finesse de gravure
  - Intel Skylake (2015): 10nm, gravure ExtremeUV
- Dissipation de chaleur
  - Chaque transistor dissipe un peu de chaleur
  - Refroidissement à l'air
- Microarchitecture
  - hyperthreading,
  - pipelines,
  - exécution out-of-order, renommage de registres
  - exécution spéculative
- Circuits multicoeur
- Accélérateurs hybride
  - GPU,
  - accélérateurs réseaux de neurones ex. Google TPU (tensor processing unit)

Quelques rappels & définitions

Architecture d'un processeur (simple)



- Programme
  - Séquence d'instructions,
  - Dans un langage à la sémantique précise
    - exemples C, C++, Rust, Java, Python…
  - Compilé (ou interprété) en « instructions machine » de l'ISA
- *ISA instruction-set architecture* « jeu d'instructions »
  - grandes familles: x86, ARM, RISC-V

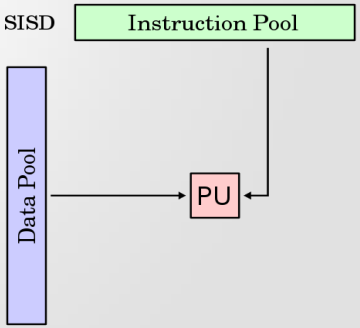
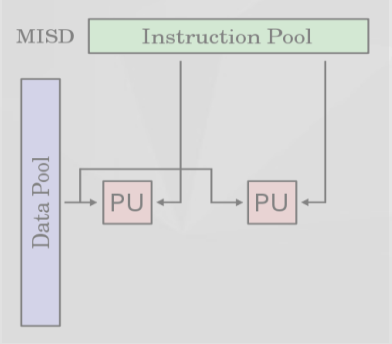
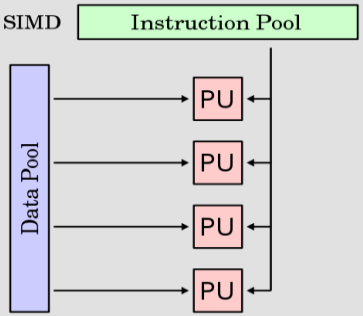
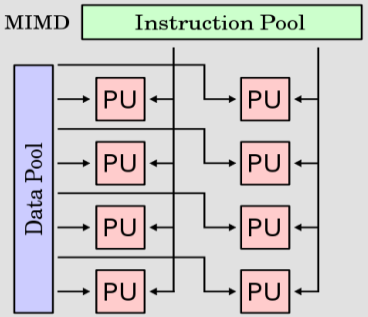
```
ADD R1, #0x1234 ; LDR R2, [R1, #5]; BNE loop
```

  - *contrat* entre le programmeur et le matériel
- Microarchitecture
  - Hors contrat: optimisations matérielles
  - pipelines, caches, hyperthreading, exécution spéculative, …

Taxonomie de Flynn (1960)

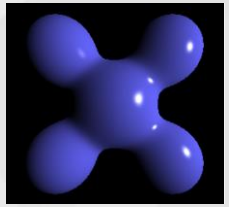
- Programme = *instructions* opèrent sur des *données*
- Flot d'instructions SI/MI
  - 1 instruction à la fois (*single instruction*)
  - plusieurs instructions en même temps (*multiple instruction*)
- Flot de données SD/MD
  - 1 donnée traitée à la fois (*single data*)
  - plusieurs données traitées simultanément (*multiple data*)

Taxonomie de Flynn

	Single instruction	Multiple instruction
Single data	CPU monoprocesseur conventionnel, microcontrôleurs 	 Plusieurs instructions en même temps, sur un seul flot de données (très rare)
Multiple data	Processeur vectoriel Ex: IBM Cell (PS3) Extensions vectorielles d'ISA x86:MMX, SSE, 3DNow! ARM:NEON VFP 	 Serveurs, circuits multicoeur… → **SPMD:** single program, multiple data → **MPMD:** multiple programs

SIMD: extensions x86 / SSE, ARM/NEON

Registres 128 bits (4 x flottant 32b) *xmm{n}*



```
movaps xmm0, [u];
```

$$xmm0 = \begin{pmatrix} u_x \\ u_y \\ u_z \\ u_w \end{pmatrix}$$

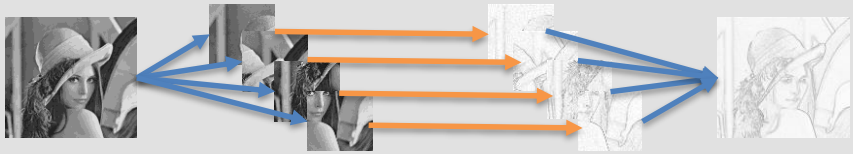
```
addps xmm0, [v] ;
```

$$xmm0 += \begin{pmatrix} u_x \\ u_y \\ u_z \\ u_w \end{pmatrix} + \begin{pmatrix} v_x \\ v_y \\ v_z \\ v_w \end{pmatrix} = \begin{pmatrix} u_x + v_x \\ u_y + v_y \\ u_z + v_z \\ u_w + v_w \end{pmatrix}$$

En pratique: librairies mathématiques, moteurs 3D, IA, …

SPMD

– Traitement du signal



MPMD

– Noyau Windows + browser + applications + …

Focus de ce cours

Séance 1

Analyse quantitative

- Latence
  - Temps de réponse
  - Temps de traitement
  - $Lat = t_{end} - t_{start}$

} Focus des TPs

- Volume
  - Bande passante (volume traité par unité de temps): $\frac{Volume}{Lat}$
  - Ex. *frames per second* (traitement vidéo), *requests per second* (serveur), Go/s (traitement de données), …
- Métriques additionnelles
  - Utilisation de ressources : nombre de CPUs, espace mémoire utilisé, utilisation du réseau, …
  - Efficacité énergétique (perf/Watt)

Performance

- $perf \sim \frac{1}{Latence}$
- Programme performant = temps court

Accélération « *Speedup* »

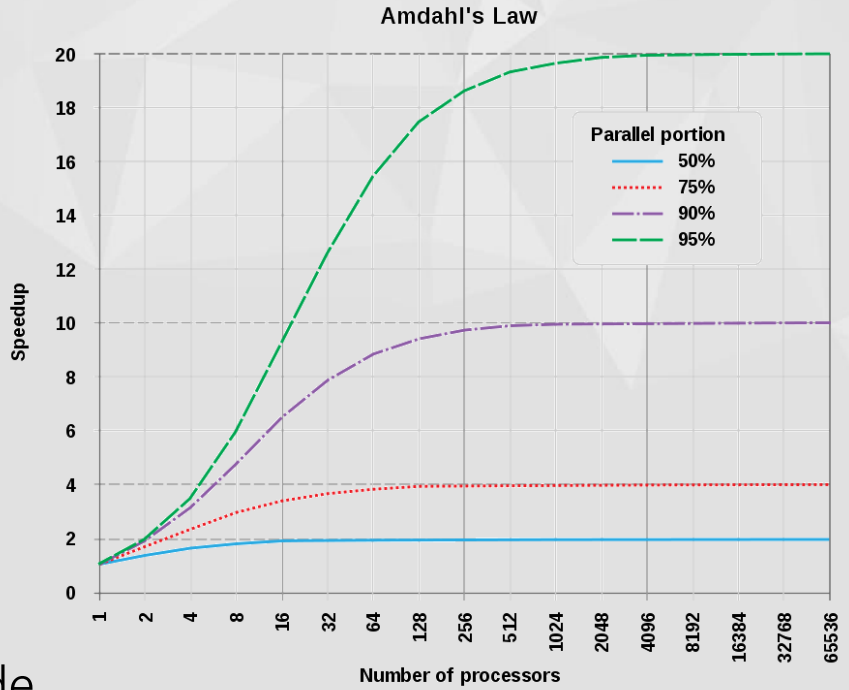
- $S = \frac{Lat_{avant}}{Lat_{après}}$ ⚠ *pas l'inverse…*
- programme plus rapide : $S > 1$

- Ex: $Lat = 1{,}2s$, $Lat' = 0{,}87s$:
- $S = \frac{1{,}2}{0{,}87} = 1{,}38 = 138\%$ *« accélération de +38% »*

- Parallélisation idéale
  - sur p processeurs: $S(p) = p$
- Dans la vie réelle: Loi d'Amdahl
  - Une fraction f du code est accélérée d'un facteur p:

$$S(f,p) = \frac{1}{1 - f + \frac{f}{p}}$$

  - Limite: $\lim_{p \to \infty} S(f,p) = \frac{1}{1-f}$
  - Note: f est une proportion du temps de calcul $0 \leq f \leq 1$.



Démonstration

- Latence avant parallélisation:

$$Lat = (1 - f) \cdot Lat + f \cdot Lat$$

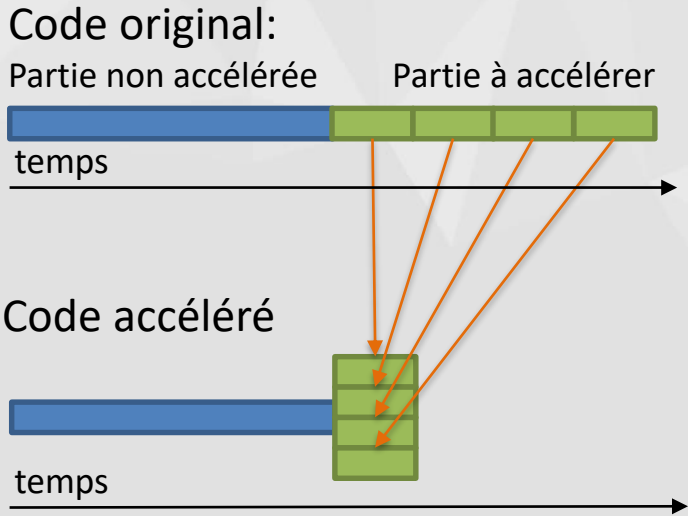
Fraction f accélérée p fois:

$$Lat' = (1 - f) \cdot Lat + \underbrace{\frac{f \cdot Lat}{p}}_{\text{accélérée p fois}}$$

- Speedup $S(f, p) = \frac{Lat}{Lat'}$
- $S(f, p) = \frac{Lat}{(1-f) \cdot Lat + \frac{f \cdot Lat}{p}} = \frac{1}{1 - f + \frac{f}{p}}$

Exemple

- Accélération 4x d'un code qui prend la moitié du temps
- $f = 50\%$, $p = 4$
- $S(f, p) = \frac{1}{1 - 0.5 + \frac{0.5}{4}} = \frac{1}{0.5 + 0.125} = \frac{1}{0.625} = 1{,}6$



Application

```
void fun() {
      fun1();    /* 30ms */
      fun2();    /* 75ms */
      fun3();    /* 45ms */
}
```

- Quel speedup sur `fun()` si j'accélère 5 fois `fun1()` ?
- Quel speedup sur `fun()` si j'accélère 1,5 fois `fun2()` ?

Parallélisme:
le début des problèmes…

- Exemple typique d'un log sur microcontrôleur:

```
[REP-ZMTP   00:00:08.093 130]There are 1 events
[REP-ZMTP   00:00:08.0[REQ    00:00:08.093 131]
   Server has a request
nt has sent a request
[REP-ZMTP   00:00:08.093 132]Received 1 bytes
```

- Plusieurs threads appellent `printf()`

Etienne HAMELIN – ehamelin@inseec-edu.com
Alexandre BERNE – aberne@inseec-edu.com

Appels concurrents d'une fonction "simple"

```
int count(int n)
{
	static int counter = 0;
	counter += n;
	return counter;
}
```

Thread1:
```
x=count(1);
```

Thread2:
```
y=count(1000);
```


Etienne HAMELIN – ehamelin@inseec-edu.com
Alexandre BERNE – aberne@inseec-edu.com

Appels concurrents d'une fonction "simple"

```
int count(int n)
{
    static int counter = 0;
    counter += n;
    return counter;
}
```

Thread1: x = count(1)
R0=1
R1←@counter
R1 = R1 + R0
@counter ←R1

Thread2: y = count(1000)
R0=1000
R1←@counter
R1 = R1 + R0
@counter ←R1

Etienne HAMELIN – ehamelin@inseec-edu.com
Alexandre BERNE – aberne@inseec-edu.com

Appels concurrents d'une fonction "simple"

```
int count(int n)
{
    static int counter = 0;
    counter += n;
    return counter;
}
```

Thread1: x=count(1)
R0=1
R1←@counter
… préempté par Thread2…
.
.
.
…puis reprend…
R1 = R1 + R0
@counter ←R1

Thread2: y=count(1000)

R0=1000
R1←@counter
R1 = R1 + R0
@counter ←R1

Etienne HAMELIN – ehamelin@inseec-edu.com
Alexandre BERNE – aberne@inseec-edu.com

La fonction count() n'est pas *thread-safe* !

```
int count(int n)
{
    static int counter = 0;
    counter += n;
    return counter;
}
```

Modifications concurrentes d'une variable partagée entre threads…

Etienne HAMELIN – ehamelin@inseec-edu.com
Alexandre BERNE – aberne@inseec-edu.com

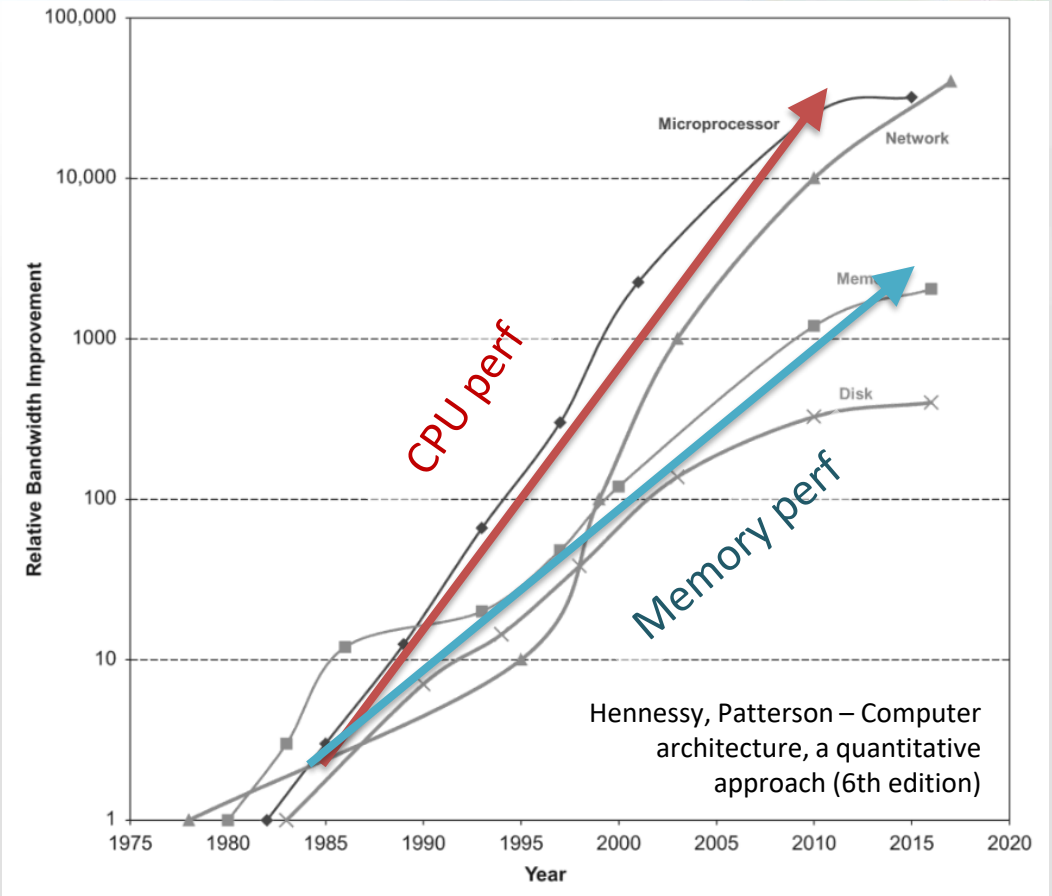
- Bugs de concurrence
  - *Data race*, *race condition*,
  - Très difficiles à reproduire et à analyser
  - Bug rare → **plus grave** : on peut faire des milliers de tests et ne pas le voir
- Exemples:
  - Therac25 : 6 patients massivement irradiés, 5 tués
  - blackout américain 2003 : 50M personnes sans électricité pendant >6h, 6G$, ~100 morts
  - Quantité de virus informatiques
- Quand vous programmerez des voitures, avions, etc.:

Un programme correct vaut mieux qu'un programme performant faux!

Etienne HAMELIN – ehamelin@inseec-edu.com
Alexandre BERNE – aberne@inseec-edu.com

Hiérarchie mémoire & cache

Problème adressé

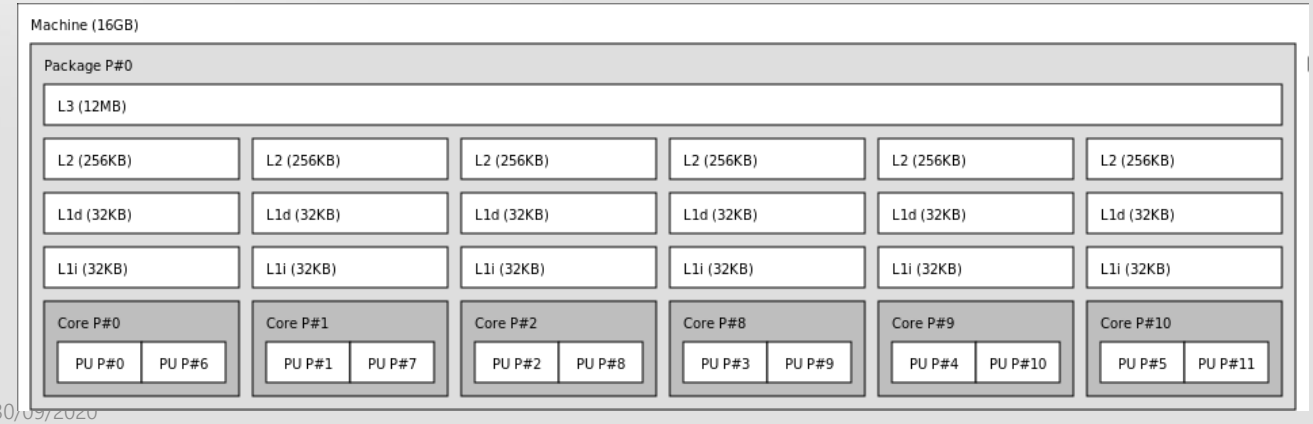


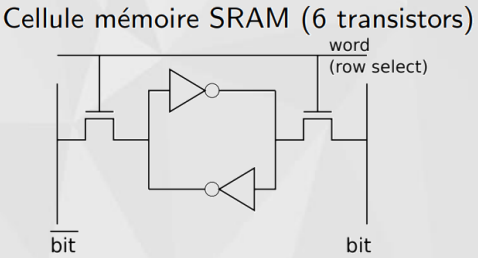
Qu'est-ce qu'un cache « antémémoire »

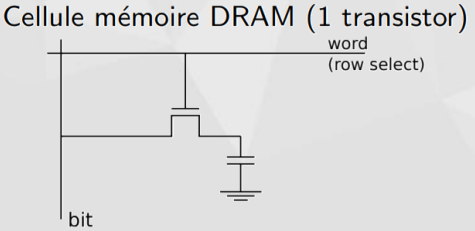
- Mémoire intégrée au CPU
- SRAM: vitesse ≫ DRAM, € ≫ DRAM

Cache multiniveau

- `lstopo (apt-get install hwloc)`

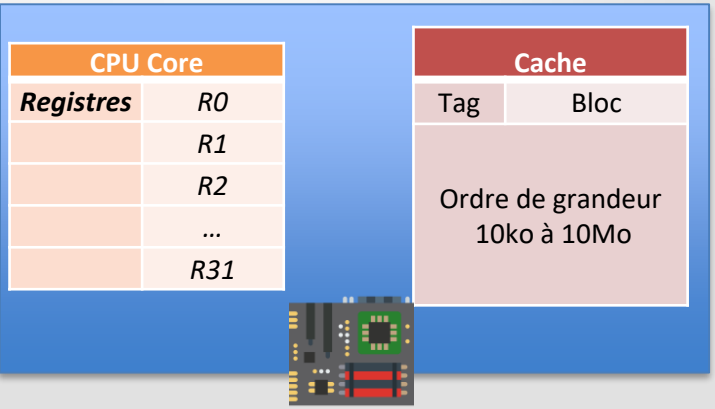


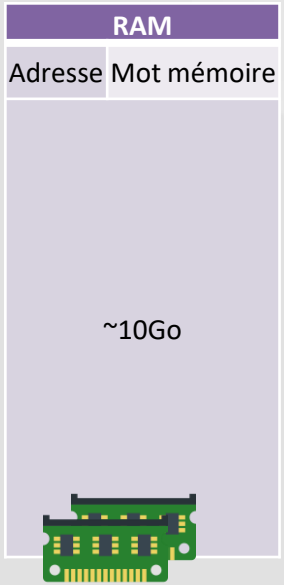




Comment fonctionne un cache ?

Principe de cache



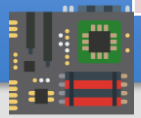


 Icônes: flaticon.com

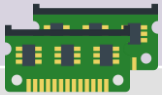
Principe de cache

CPU Core	
Registres	*R0*
	R1
	R2
	…
	R31

Cache	
Tag	Bloc
Ordre de grandeur 10ko à 10Mo	



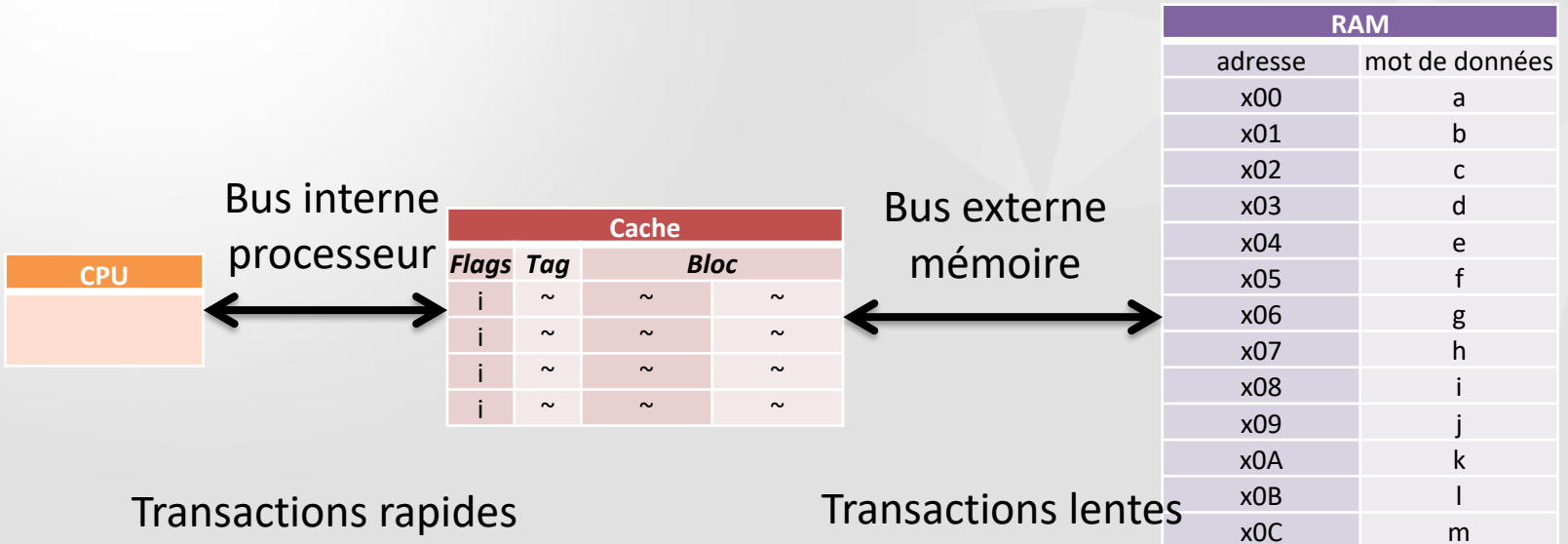
RAM	
Adresse	Mot mémoire
~10Go	



Disk	
Chemin (path)	Fichier
~100Go à 1To	

Internet/Cloud	
URL	Ressource
1 million d'exaoctets 10^{24}o (?)	

Fonctionnement d'un cache



- Localité temporelle
  - Adresse accédée récemment → probabilité d'être réutilisée bientôt

  ⇒ politiques de remplacement
- Localité spatiale
  - Adresse accédée récemment → probabilité d'utiliser les adresses adjacentes

  ⇒ gestion par blocs
- Exemple

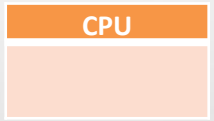
```
for (i=0; i < n; i++) {
  for (j=0; j < n; j++) {
    y[i] += a[i][j] * x[j];
}}
```

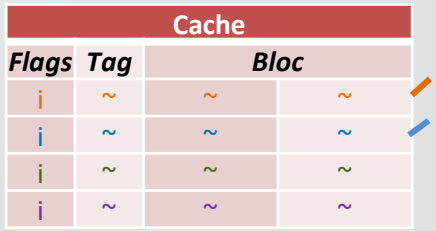
Paramètres caractéristiques d'un cache

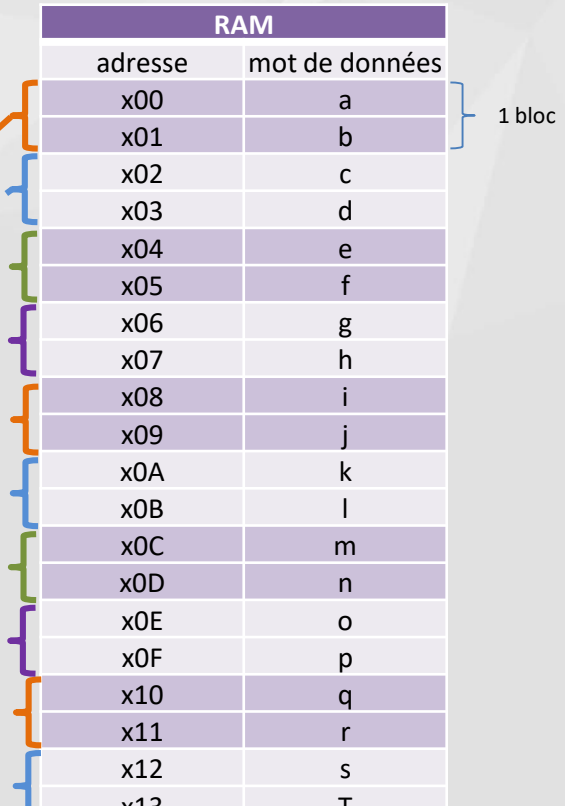
- Capacité du cache, taille de bloc
- Degré d'associativité
  - combien de choix ai-je pour placer un bloc de données?
- Lors des remplacements, quelle ligne dois-je évincer ?
  - politique de remplacement
- Comment propager les écritures?
  - politique d'écriture
- Comment gérer les accès concurrents?
  - politique de cohérence

Fonctionnement d'un cache

Cache
direct-mapped
4 lignes
blocs de 2 octets







Lecture

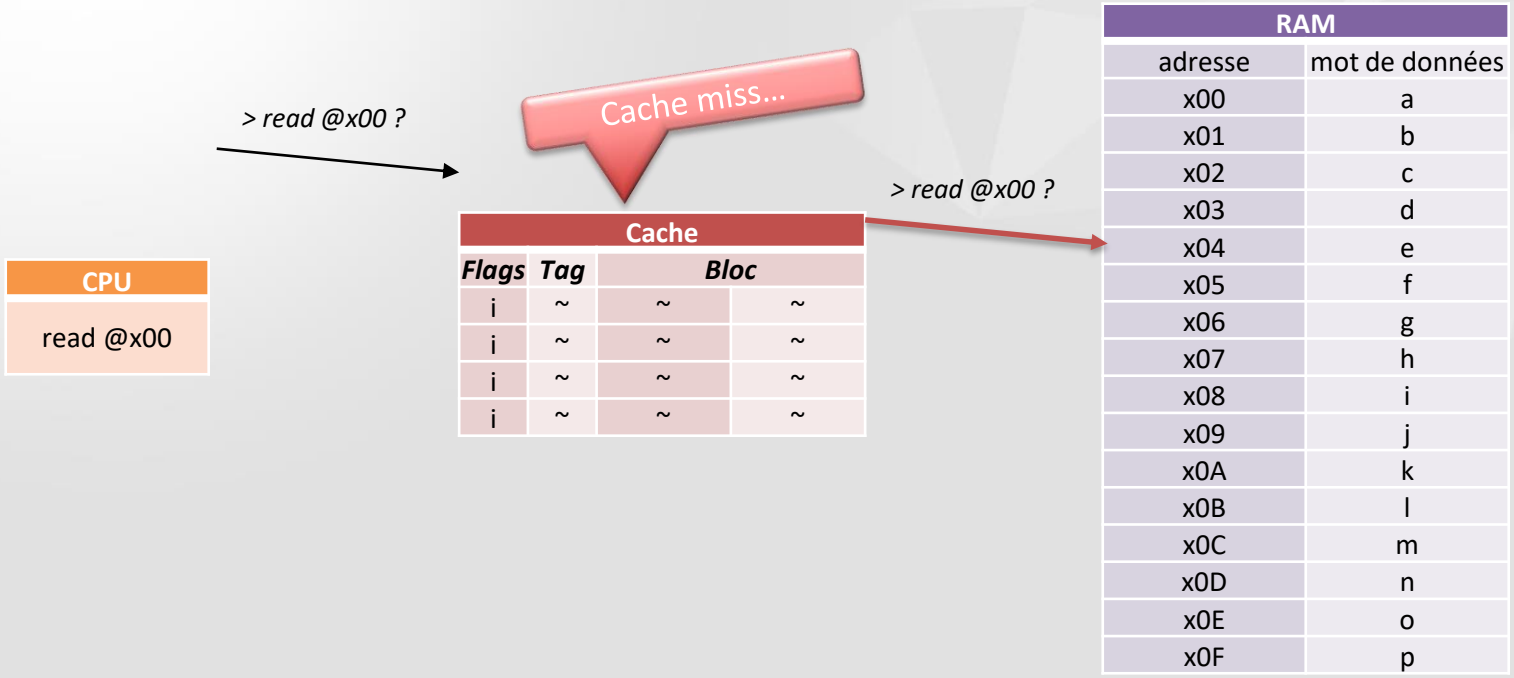
> read @x00 ?

CPU
read @x00

Cache			
Flags	***Tag***	***Bloc***	
i	~	~	~
i	~	~	~
i	~	~	~
i	~	~	~

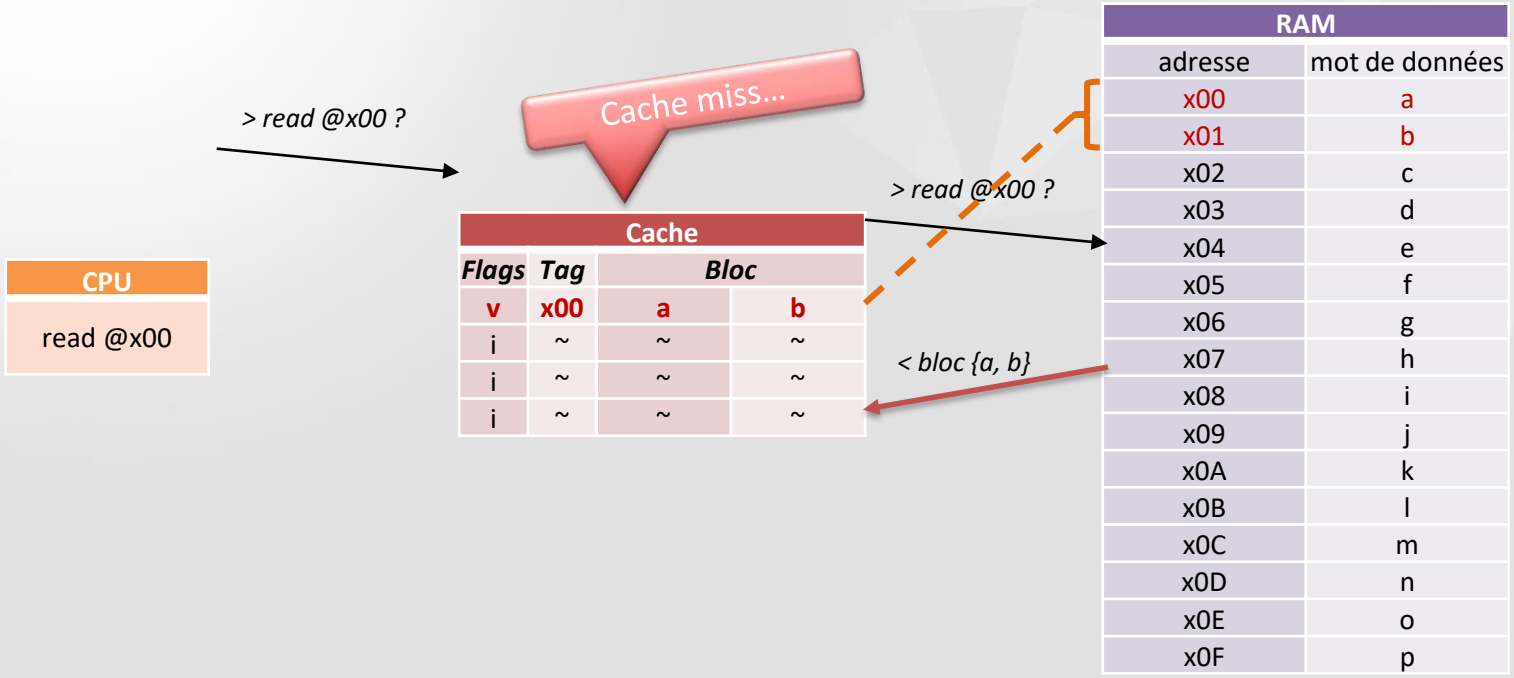
RAM	
adresse	mot de données
x00	a
x01	b
x02	c
x03	d
x04	e
x05	f
x06	g
x07	h
x08	i
x09	j
x0A	k
x0B	l
x0C	m
x0D	n
x0E	o
x0F	p

Fonctionnement d'un cache

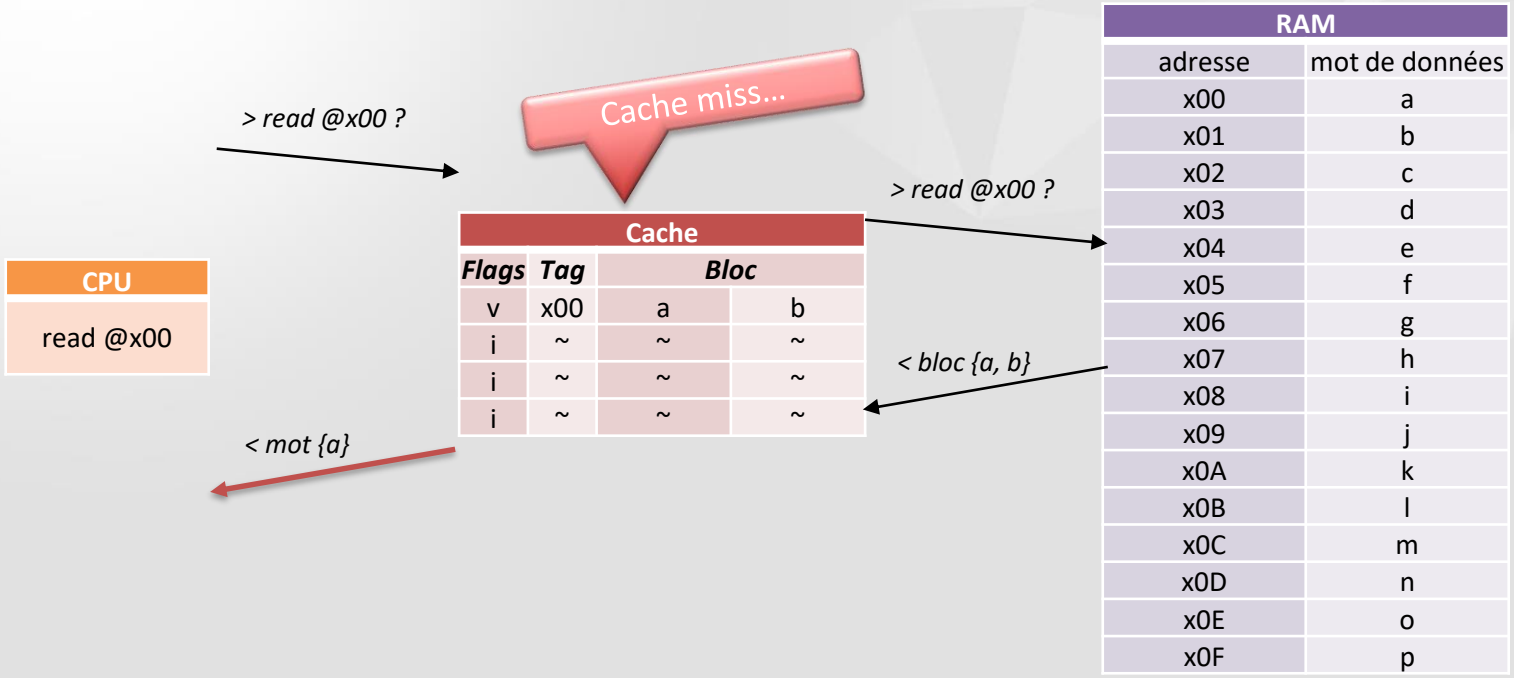


Lecture / 3

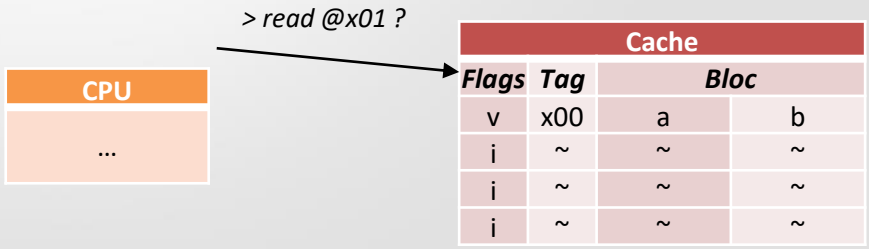
Fonctionnement d'un cache



Fonctionnement d'un cache

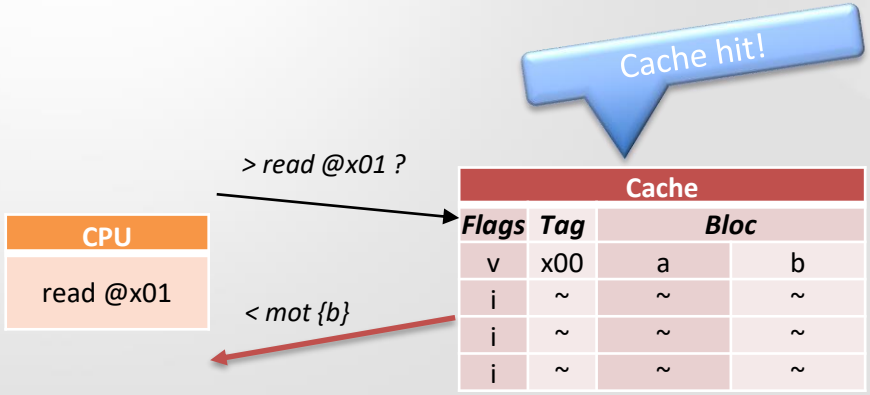


Lecture / cache hit



RAM	
adresse	mot de données
x00	a
x01	b
x02	c
x03	d
x04	e
x05	f
x06	g
x07	h
x08	i
x09	j
x0A	k
x0B	l
x0C	m
x0D	n
x0E	o
x0F	p

Fonctionnement d'un cache



RAM	
adresse	mot de données
x00	a
x01	b
x02	c
x03	d
x04	e
x05	f
x06	g
x07	h
x08	i
x09	j
x0A	k
x0B	l
x0C	m
x0D	n
x0E	o
x0F	p

Fonctionnement d'un cache

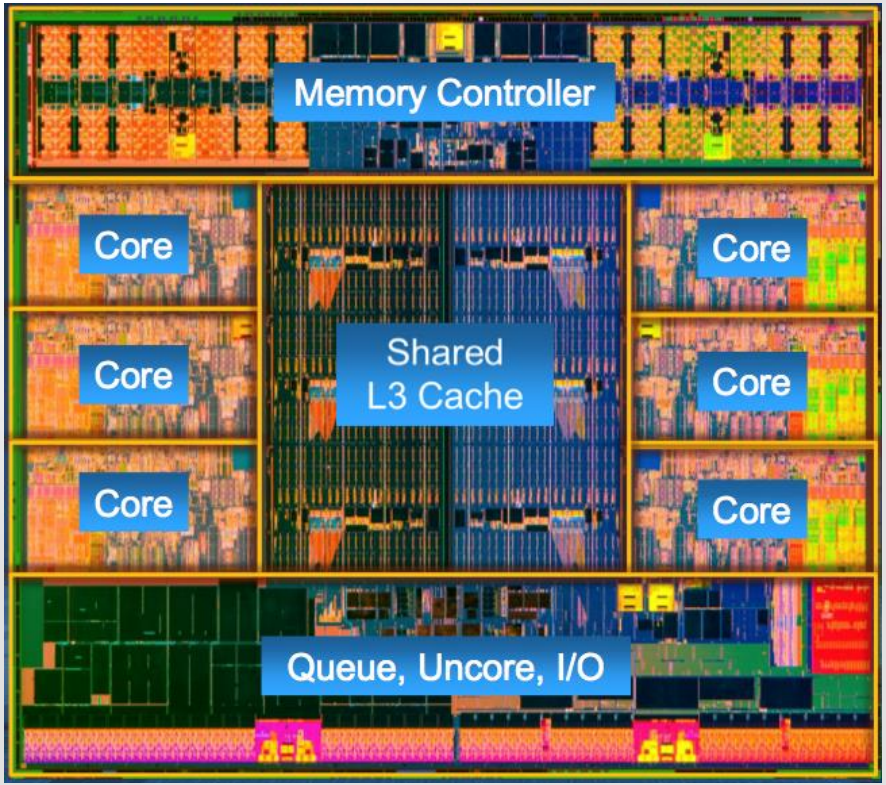
CPU

Cache hit!

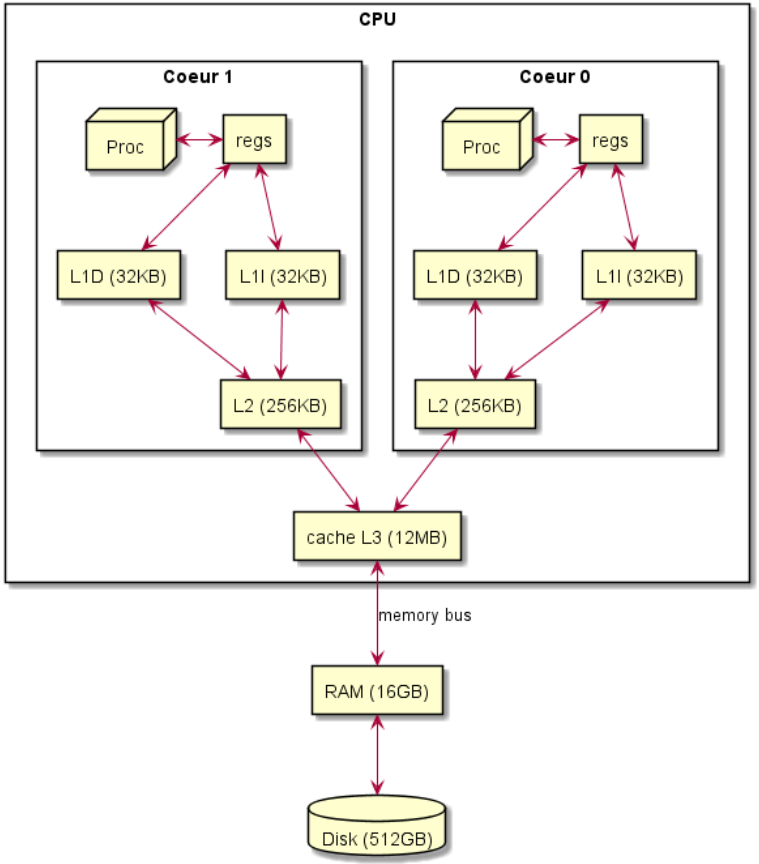
Cache			
Flags	***Tag***	***Bloc***	
v	x00	a	b
i	~	~	~
i	~	~	~
i	~	~	~

RAM	
adresse	mot de données
x00	a
x01	b
x02	c
x03	d
x04	e
x05	f
x06	g
x07	h
x08	i
x09	j
x0A	k
x0B	l
x0C	m
x0D	n
x0E	o
x0F	p

Hiérarchie mémoire

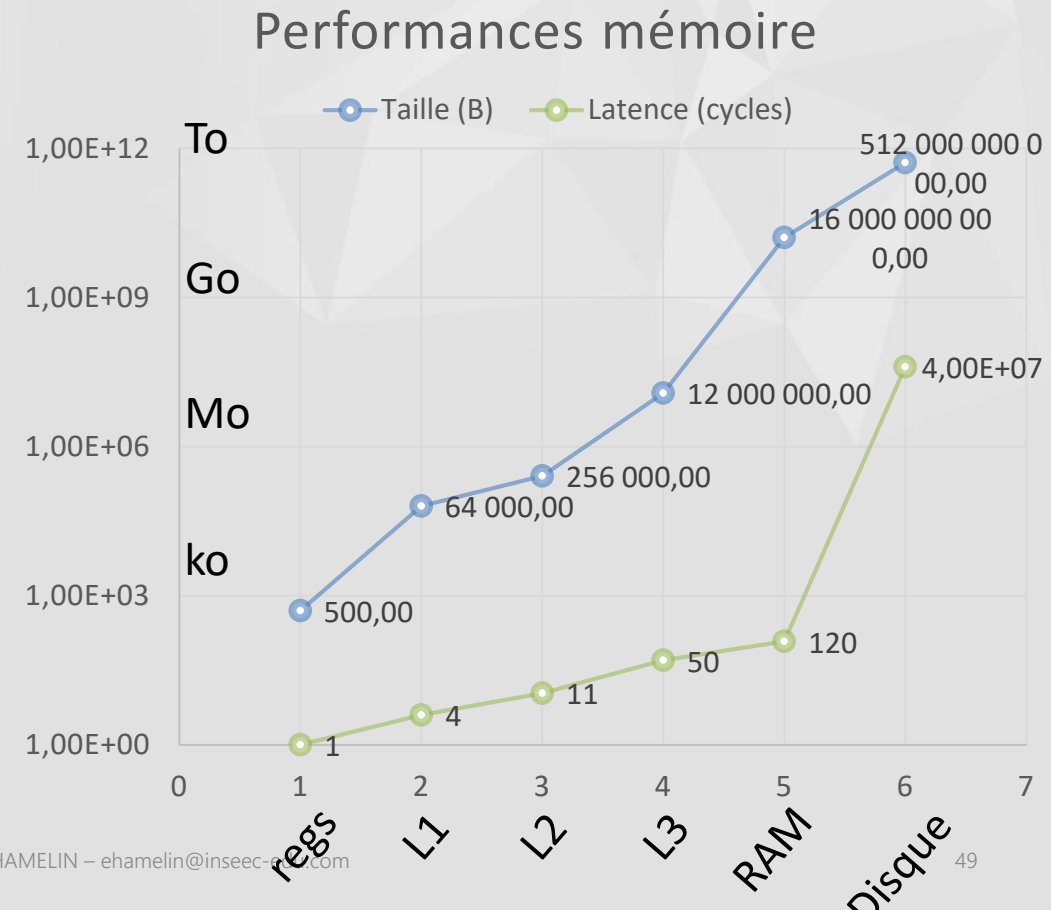


Intel Core™ i7-4960X



Latences

Mémoire	Temps d'accès (cycles)	Volume (octets)
Registres	1	~500 o
Cache L1	4	32 ko
Cache L2	12	256 ko
Cache L3	50	12 Mo
RAM	120	16 Go
Disque	40 millions (~10ms)	512 Go



Séance 1

Introduction & Concepts: récap

Le récap'

- Rappels: architecture d'un CPU,
- Quête historique de performance
- Taxonomie
  - SISD, SIMD, MIMD: *{Single/Multiple}-Instruction, {S/I}-Data*
- Analyse quantitative
  - Métriques, latence, *speedup*,
  - Loi d'Amdahl
- Risques associés à la concurrence
- Fonctionnement d'un cache