

# Architecture des ordinateurs (X31I050)

Frédéric Goualard

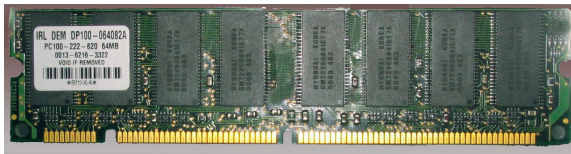
Laboratoire d'Informatique de Nantes-Atlantique, UMR CNRS 6241  
Bureau 112, bât. 11  
[Frederic.Goualard@univ-nantes.fr](mailto:Frederic.Goualard@univ-nantes.fr)

# Caches

RAM (*Random Access Memory*) :

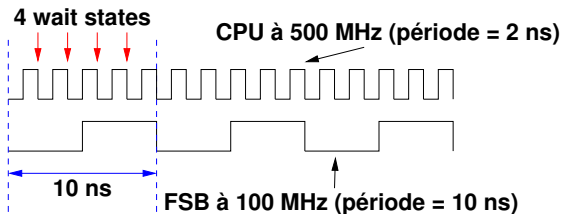
- ▶ Adressage direct et non séquentiel
- ▶ Mémoire volatile : contenu disparaît à l'extinction du PC
- ▶ **Vision conceptuelle** : tableau d'octets

Barette SIMM (*Single In-line Memory Module*) :



*Temps d'accès mémoire* : temps mis entre une demande d'opération (lecture/écriture) par le CPU et la fin de son exécution.

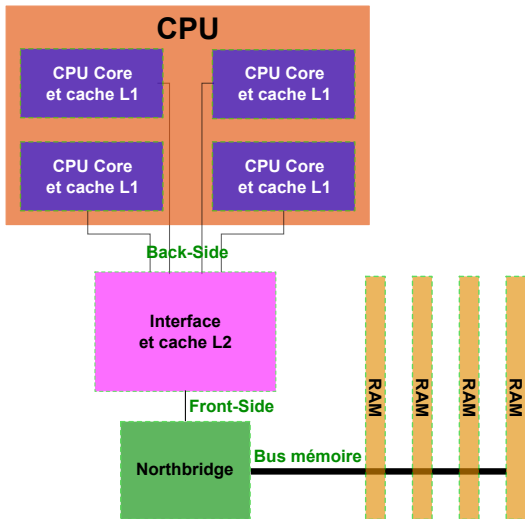
- ▶ CPU et RAM travaillent à des vitesses très différentes
- ▶ Échange de données à une cadence différente de celle de l'horloge du CPU
  - ➡ le FSB (*Front-Side Bus*) a sa propre horloge (66–133 MHz)
  - ➡ Protocole d'échange semi-synchrone
  - ➡ Insertion d'états d'attente du CPU (*wait states*)



- ▶ Introduction des *wait states* : coûteux en temps
- ▶ Élimination des *wait states*?
  - ▶ Utilisation de mémoire plus rapide (cher)
  - ▶ Utilisation des propriétés des programmes

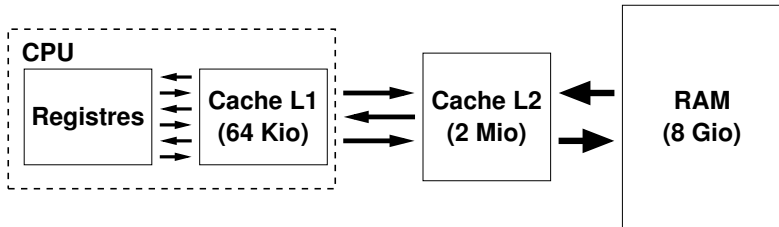
```
for (int i=0; i <10; ++i) {  
    T[i] = 0;  
}
```

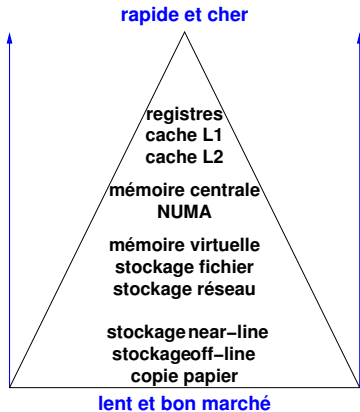
- ▶ Localité temporelle des références :
  - ▶ *i* est accédé répétitivement
- ▶ Localité spatiale des références :
  - ▶ Toutes les cases consécutives en mémoire représentant *T* sont accédées



Idee : utiliser de la mémoire très rapide en petite quantité pour stocker les cases de la RAM récemment accédées ou susceptibles de l'être bientôt

- ▶ Mémoire cache de niveau 1 (cache L1) sur le CPU
- ▶ Mémoire cache de niveau 2 (cache L2) à côté de la RAM





- NUMA : *Non-Uniform Memory Access* (e.g., mémoire sur un autre processeur)  
near-line : jukebox automatisé  
off-line : montage manuel



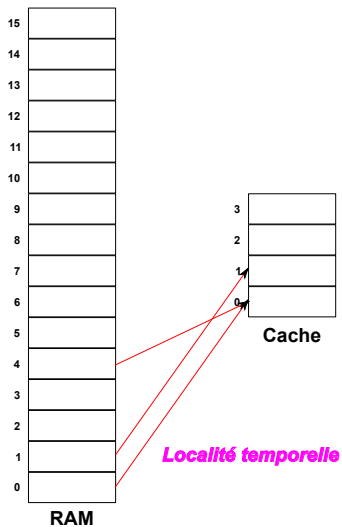
**Succès.** La donnée recherchée est trouvée au niveau attendu

**Défaut.** La donnée recherchée n'est pas trouvée au niveau attendu

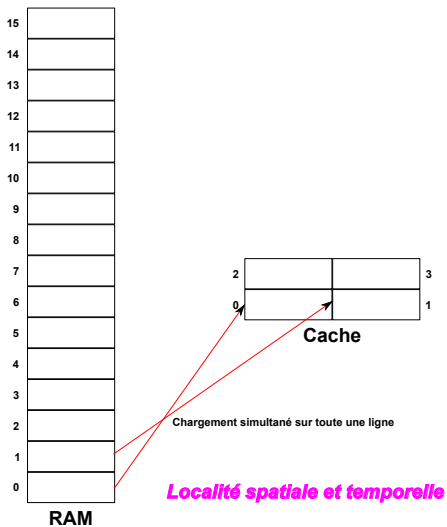
**Taux de succès.** Part des accès donnant lieu à un succès

**Taux de défaut.** Part des accès donnant lieu à un défaut

## Cache à correspondance directe :



## Cache à correspondance directe :



**# Naïf**

```
for i in range(0,n):
    for j in range(0,n):
        M3(i,j) = 0
        for k in range(0,n):
            M3(i,j) += M1(i,k)*M2(k,j)
```

**# Twisted**

**# Initialisation à 0 de M3**

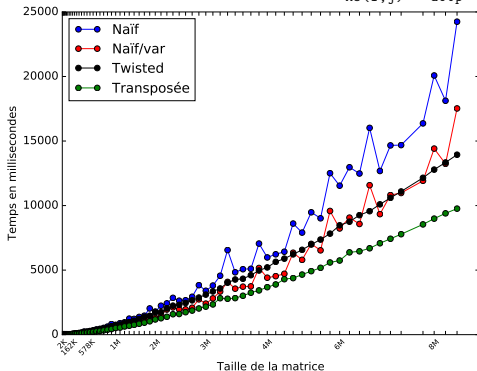
```
for i in range(0,n):
    for k in range(0,n):
        for j in range(0,n):
            M3(i,j) += M1(i,k)*M2(k,j)
```

**# Naïf/var**

```
for i in range(0,n):
    for j in range(0,n):
        dotp = 0
        for k in range(0,n):
            dotp += M1(i,k)*M2(k,j)
        M3(i,j) = dotp
```

**# Transposée**

```
M2 = transpose(M2)
for i in range(0,n):
    for j in range(0,n):
        dotp = 0
        for k in range(0,n):
            dotp += M1(i,k)*M2(j,k)
        M3(i,j) = dotp
```



```
# Naïf
for i in range(0,n):
    for j in range(0,n):
        M3(i,j) = 0
        for k in range(0,n):
            M3(i,j) += M1(i,k)*M2(k,j)
```

```
# Twisted
# Initialisation à 0 de M3
for i in range(0,n):
    for k in range(0,n):
        for j in range(0,n):
            M3(i,j) += M1(i,k)*M2(k,j)
```

```
# Naïf/var
for i in range(0,n):
    for j in range(0,n):
        dotp = 0
        for k in range(0,n):
            dotp += M1(i,k)*M2(k,j)
        M3(i,j) = dotp
```

```
# Transposée
M2 = transpose(M2)
for i in range(0,n):
    for j in range(0,n):
        dotp = 0
        for k in range(0,n):
            dotp += M1(i,k)*M2(j,k)
        M3(i,j) = dotp
```

Matrice 1024 * 1024		
Algorithme	L1 misses	LLC misses
Naïf	1 083 412 120	748 184 605
Naïf/var	1 080 288 538	683 645 734
Twisted	138 176 251	992 215
Transposée	137 610 611	1 137 862

**Fin du cours**