

Ch. 8

Hiérarchie de Mémoires

B. Quoitin
(bruno.quoitin@umons.ac.be)

Table des Matières

Technologies RAM

→ SRAM versus DRAM

- DRAM Synchrones (SDRAM)
- Latence des DRAMs

Mémoire Cache

- Localité
- Principes de fonctionnement
- Organisation
- Implémentation *fully-associative* / *direct-mapped* / *set-associative*
- Remplacement et ré-écriture

Mémoire Virtuelle

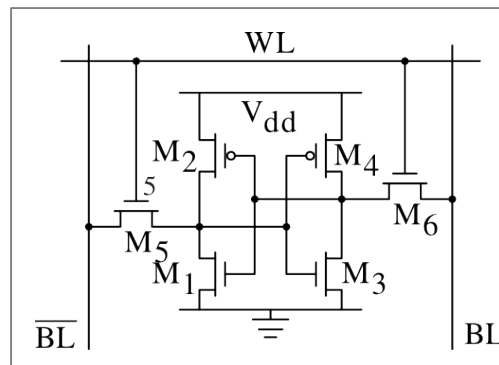
- Traduction d'adresses
- Table des pages
- Translation Lookaside Buffer

Technologies RAM

RAM Statique (SRAM)

- Une mémoire SRAM (*static random access memory*) stocke un bit comme un état dans une **bascule bistable** implémentée avec 6 transistors.

Cellule SRAM



M1 à M6 sont des transistors.
M1/M2 et M3/M4 forment deux
portes inverseuses.

BL donne le nouvel état du bit;
WL déclenche l'écriture.

- Avantages**
 - L'état de la bascule peut être changé très rapidement.
 - L'état de la bascule est conservé tant qu'elle est alimentée.
- Inconvénients**
 - Encombrement, coût, consommation électrique

Source : **What Every Programmer Should Know About Memory**, Ulrich Drepper, Red Hat Inc., November 2007.

Technologies RAM

RAM Dynamique (DRAM)

- Une mémoire DRAM (*dynamic random access memory*) stocke un bit comme une **tension dans un condensateur** de très faible capacité, i.e. de l'ordre de qqes 10^{aines} de femtofarads (10^{-15} F).



- Avantages**
 - Encombrement, coût, consommation électrique
- Inconvénients**
 - L'état se perd à cause de courants de fuite (en quelques millisecondes).
 - Il est nécessaire de **rafraîchir régulièrement** (toutes les 64 ms) chaque bit → accès complexe !

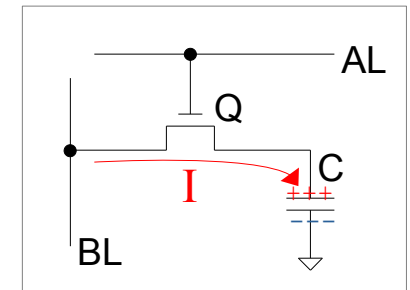
Source : **What Every Programmer Should Know About Memory**, Ulrich Drepper, Red Hat Inc., November 2007.

Technologies RAM

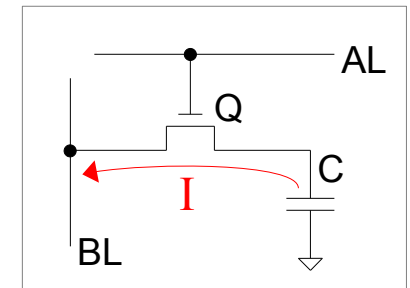
RAM Dynamique (DRAM)

- Si la ligne d'adresse AL vaut 0, le transistor Q est « ouvert » \Rightarrow aucune lecture ou écriture n'a lieu.
- Pour **écrire un bit à 1**, il faut que $AL=1$ et $BL=1$. Le condensateur C se charge.
- Pour **écrire un bit à 0**, il faut que $AL=1$ et $BL=0$. Le condensateur C se décharge.
- Pour **lire un bit**, il faut que $AL=1$ et BL soit connecté à un « *sense amplifier* » qui détermine si la charge correspond à 0 ou 1. Cette lecture nécessite de tirer un faible courant de C .

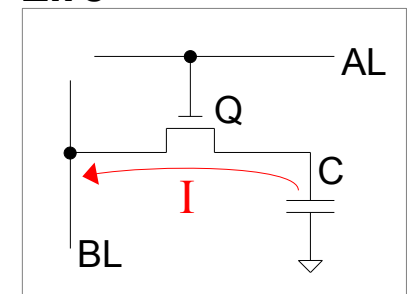
Ecrire 1



Ecrire 0



Lire



Technologies RAM

Comparaison SRAM et DRAM

	SRAM	DRAM
Technique	bascule bistable	condensateur
Nombre de transistors par cellule (bit)	typiquement 6	typiquement 1
Consommation électrique	grande	petite
Encombrement physique	grand	petit
Temps d'accès	petit	grand
Coût	grand	petit
Complexité de l'accès	simple	complexe (refresh cycles)

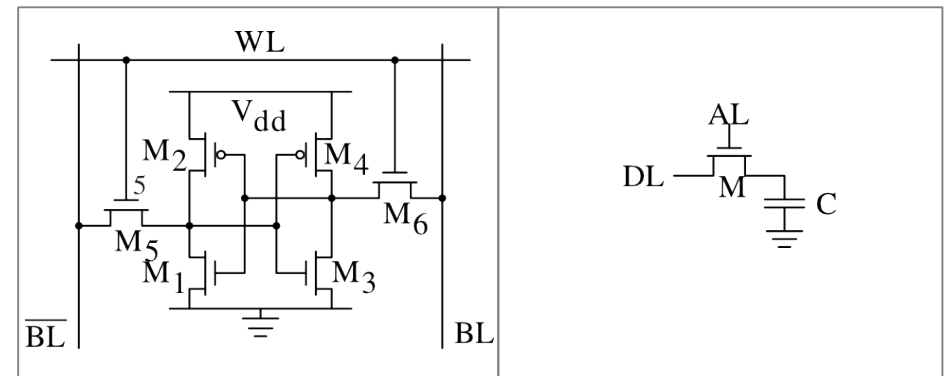


Table des Matières

Technologies RAM

- SRAM versus DRAM

DRAM Synchrones (SDRAM)

- Latence des DRAMs

Mémoire Cache

- Localité
- Principes de fonctionnement
- Organisation
- Implémentation *fully-associative* / *direct-mapped* / *set-associative*
- Remplacement et ré-écriture

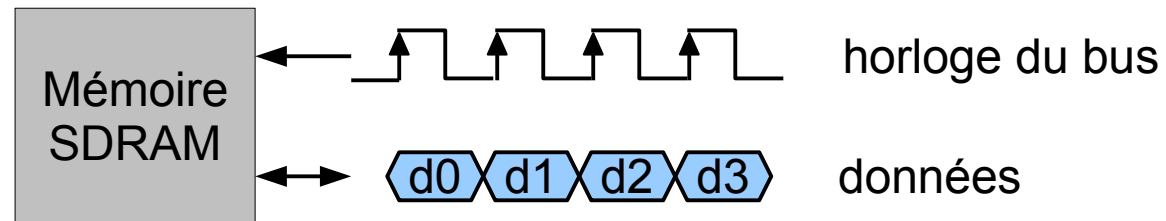
Mémoire Virtuelle

- Traduction d'adresses
- Table des pages
- Translation Lookaside Buffer

DRAM Synchrones

DRAM Synchrones (SDRAM)

- Les ordinateurs récents utilisent des mémoires **DRAM synchrones** (*synchronous dynamic random access memory* – SDRAM).
- Dans une SDRAM les transferts sont synchronisés avec l'horloge du bus système : un mot de donnée est transféré par cycle d'horloge.



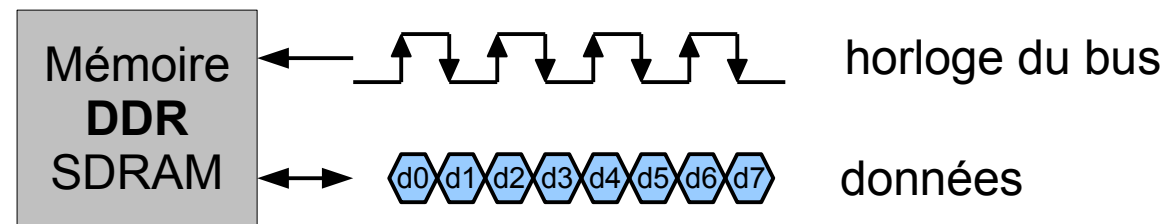
- Exemple : mémoire SDRAM « PC100 » sur un bus système de 64 bits cadencé à 100MHz
 - 100MHz → un cycle dure 10ns
 - 100MT/s (millions de transferts par seconde)
 - Chaque transfert concerne 64 bits → débit = **800 MB/s** (théorique!)

Note : « PC100 » est un standard publié par Intel en 1998 pour l'interconnexion des SDRAMs sur des cartes mère d'ordinateurs type PC.

DRAM Synchrones

Double Data Rate (DDR)

- Les mémoires SDRAM ont évolué vers un **double taux de transfert** (*double data rate* – **DDR**). Celles-ci permettent de transférer deux mots par cycle.



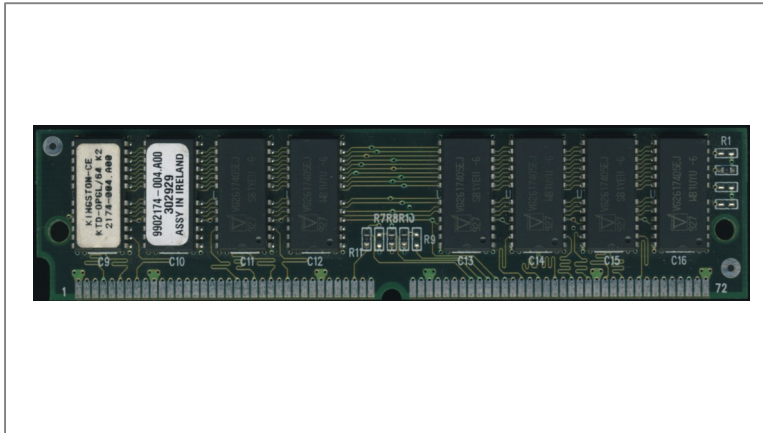
- Exemple : mémoire DDR SDRAM « PC1600 », bus système de 64 bits cadencé à 100MHz
 - un cycle dure 10ns
 - $200\text{MT/s} \rightarrow 200\text{ M} * 64\text{ bits par seconde} = 1600\text{ MB/s}$ (théorique!)

Note : D'autres évolutions de DDR comme DDR2 à DDR3 offrent des taux de transfert encore plus élevés.

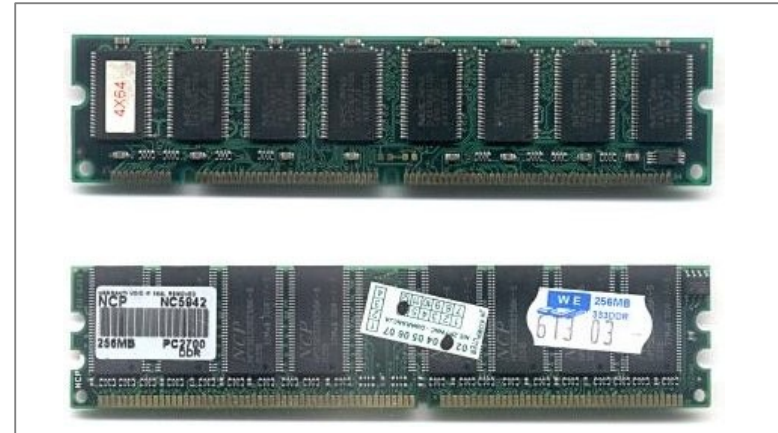
DRAM Synchrones

Modules mémoire

- Les mémoires SDRAM sont fournies sous forme de modules standards (JEDEC) enfichables sur la carte mère d'un ordinateur. Un module de mémoire contient plusieurs circuits intégrés mémoire qui sont utilisés en parallèle → augmente la largeur des données accédées.
- Exemples
 - SIMM – *Single Inline Memory Module* (largeur de 32 bits)
 - DIMM – *Dual Inline Memory Module* (largeur de 64 bits)



SIMM

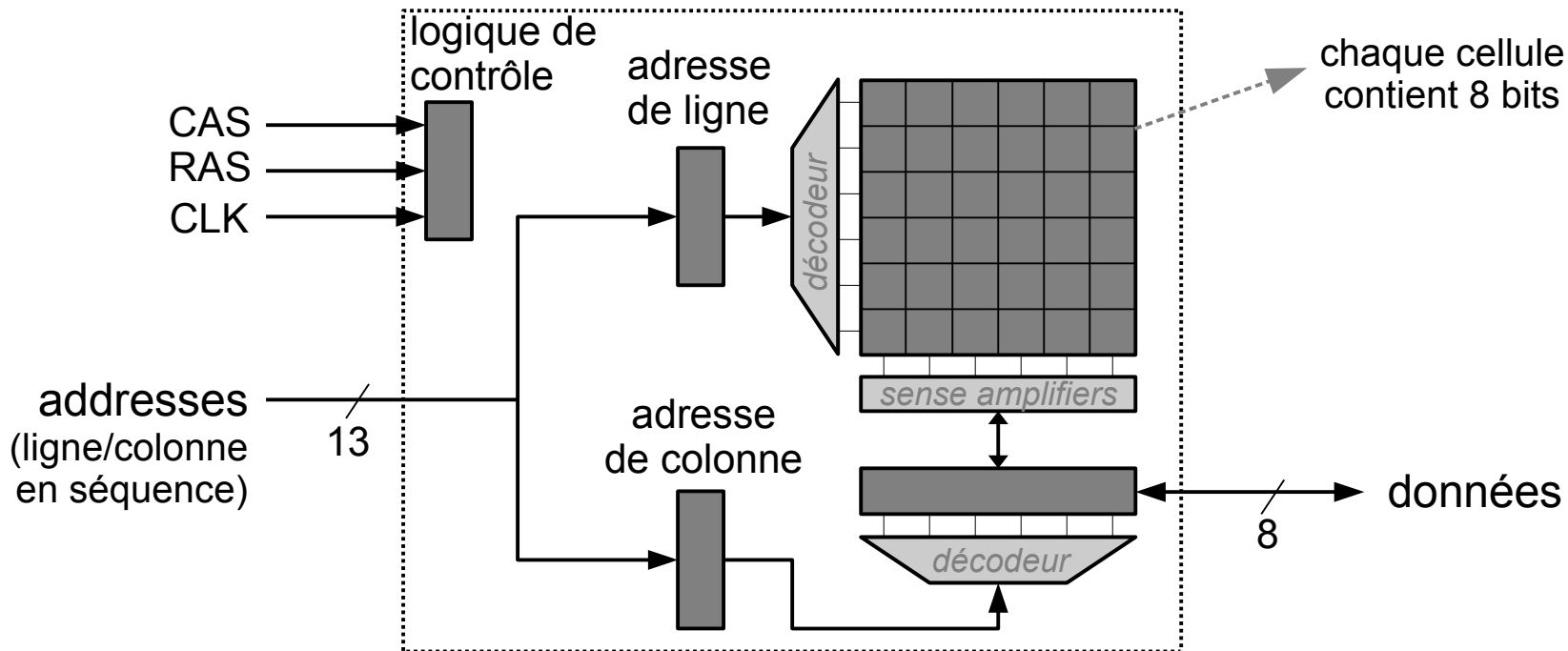


DIMM

DRAM Synchrones

Exemple – Micron Technology's 512 Mbits (64 M x 8)

- Composée de 64 millions de cellules de 8 bits
- Organisée en 8192 lignes x 8192 colonnes
- 13 lignes d'adresse ($2^{13} = 8192$)

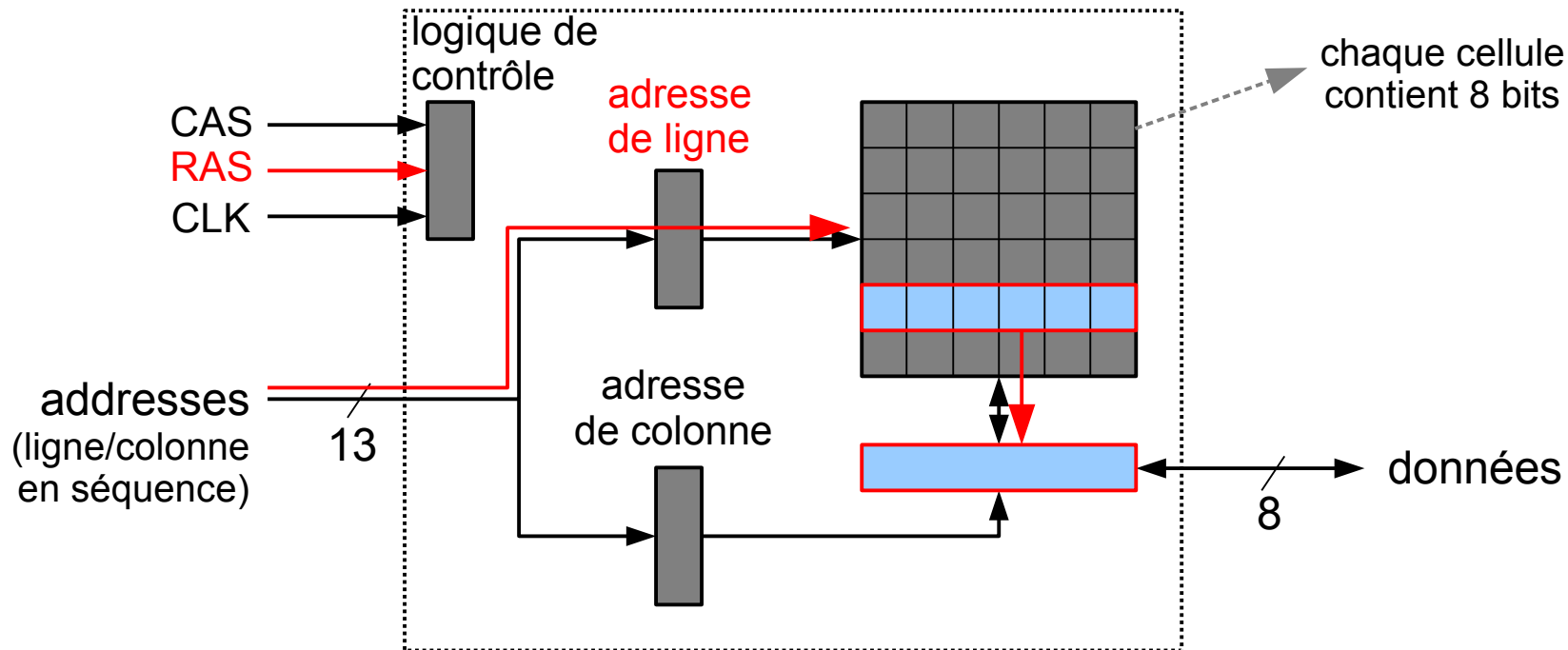


Reference : **MT48LC64M8A2** datasheet, Micron Technology Inc., 2000.

DRAM Synchrones

Exemple – Micron Technology's 512 Mbits (64 M x 8)

- Composée de 64 millions de cellules de 8 bits
- Organisée en 8192 lignes x 8192 colonnes
- 13 lignes d'adresse ($2^{13} = 8192$)

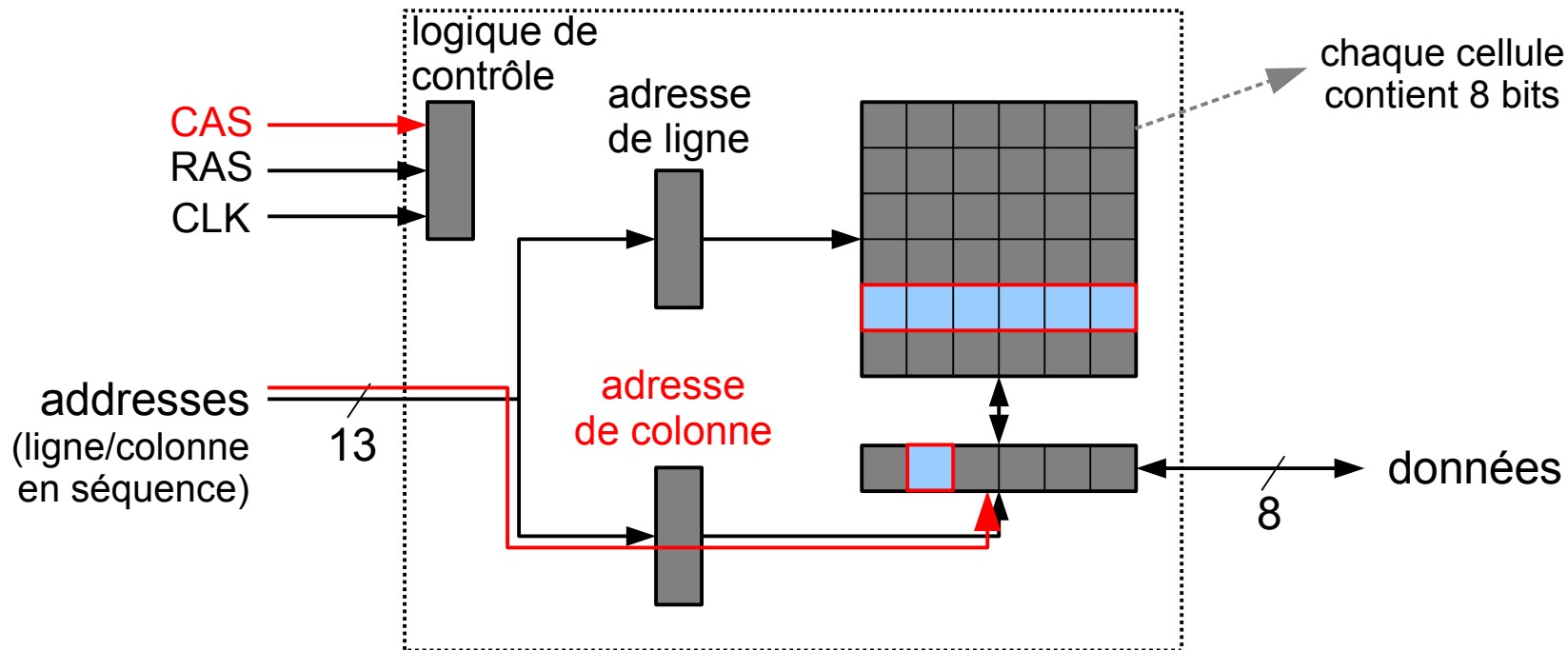


Reference : **MT48LC64M8A2** datasheet, Micron Technology Inc., 2000.

DRAM Synchrones

Exemple – Micron Technology's 512 Mbits (64 M x 8)

- Composée de 64 millions de cellules de 8 bits
- Organisée en 8192 lignes x 8192 colonnes
- 13 lignes d'adresse ($2^{13} = 8192$)

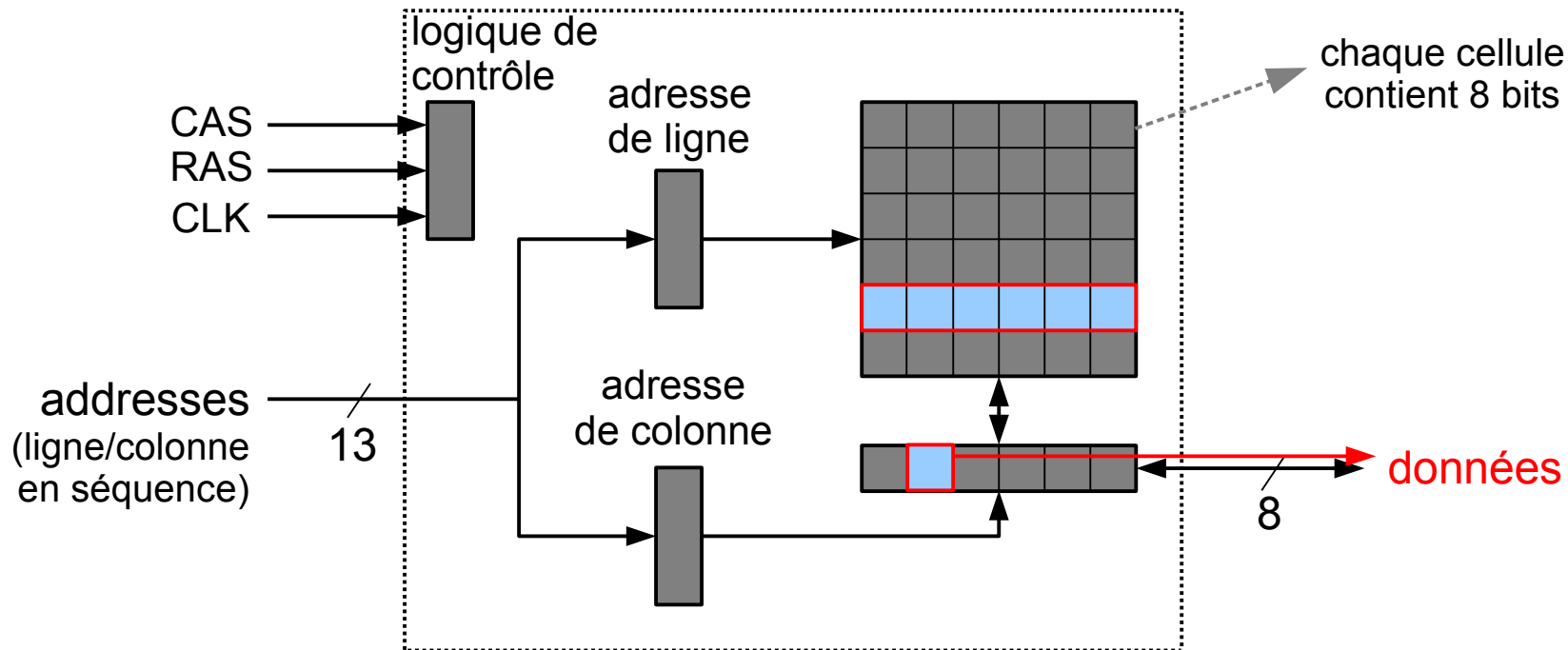


Reference : **MT48LC64M8A2** datasheet, Micron Technology Inc., 2000.

DRAM Synchrones

Exemple – Micron Technology's 512 Mbits (64 M x 8)

- Composée de 64 millions de cellules de 8 bits
- Organisée en 8192 lignes x 8192 colonnes
- 13 lignes d'adresse ($2^{13} = 8192$)

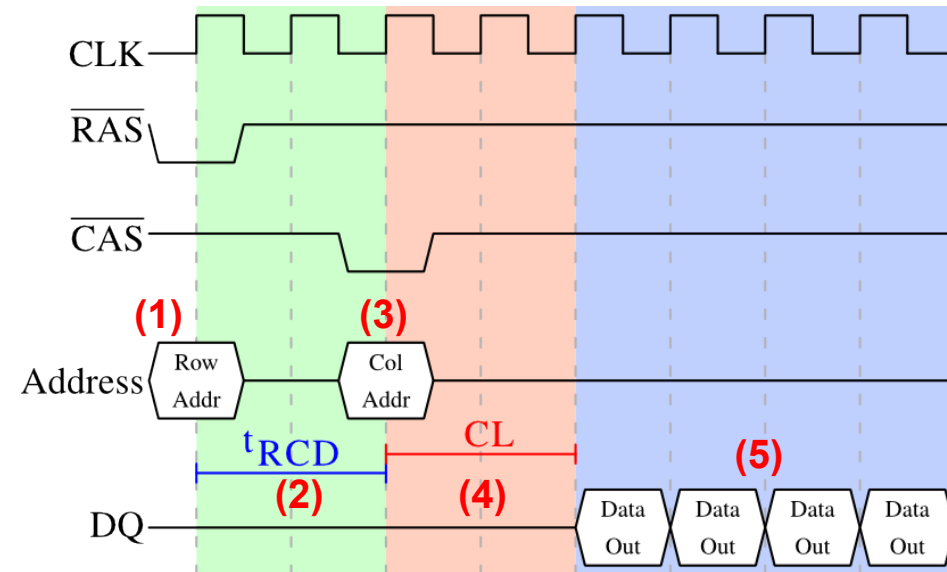


Reference : **MT48LC64M8A2** datasheet, Micron Technology Inc., 2000.

DRAM Synchrones

Protocole d'accès SDRAM

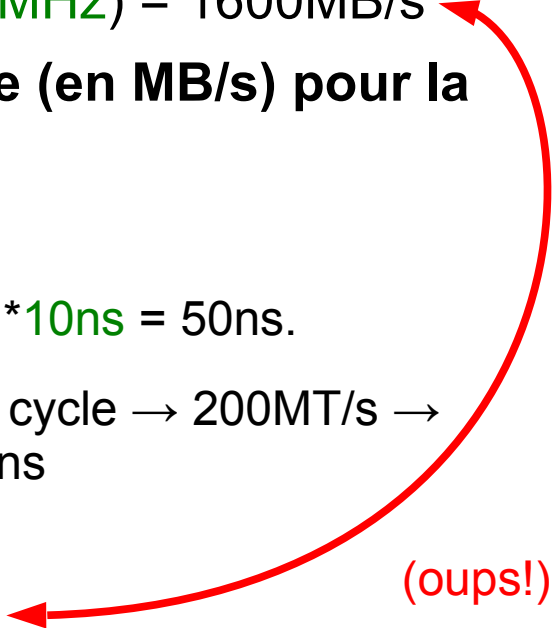
- Détail des étapes
 - Envoyer l'adresse de ligne (*row*) sur le bus d'adresses
 - Attendre t_{RCD} cycles (*RAS to CAS Delay*).
 - Envoyer l'adresse de colonne (*column*) sur le bus d'adresses
 - Attendre CL cycles (*CAS Latency*).
 - Lire / écrire un mot par cycle d'horloge (2 mots par cycle si DDR)



Source : **What Every Programmer Should Know About Memory**, Ulrich Drepper, Red Hat Inc., November 2007.

DRAM Synchrones

Impact de ce protocole sur le débit

- Exemple -- Mémoire DDR SDRAM « PC1600 »
 - bus à 100MHz (cycle=10ns).
 - Taille d'un mot : 64 bits (8 octets)
 - Cycles entre RAS et CAS (t_{RDC}) = 2
 - Cycles entre CAS et lecture des données (CL) = 3
 - Taux de transfert théorique = 64 bits / 8 * (2*100MHz) = 1600MB/s
 - **Question : Quel est le débit maximum atteignable (en MB/s) pour la lecture d'un mot unique ?**
 - Solution
 - Temps d'accès = (t_{RDC} + CL) * temps cycle = (2 + 3) * 10ns = 50ns.
 - Temps transfert 1 mot : DDR implique 2 transferts par cycle → 200MT/s → temps transfert 1 mot = temps cycle / 2 = 10ns / 2 = 5ns
 - Temps total = 50 ns + 5 ns = 55 ns
 - Débit maximum atteignable = ~18MT/s ou 145MB/s
- 

DRAM Synchrones

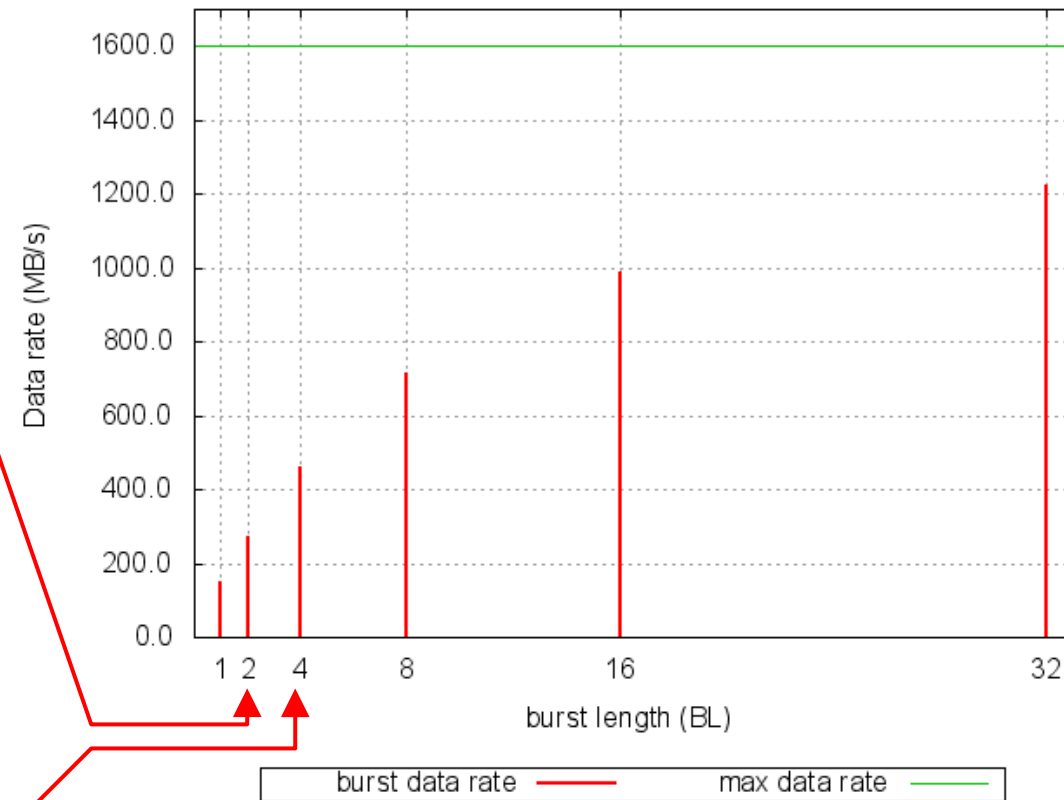
Accès en rafale (*burst*)

- Pour amortir le coût de l'accès sur la lecture de plusieurs mots, il est possible d'effectuer plusieurs lectures / écritures à des adresses successives en "***burst***".
- Tous les accès successifs ont lieu au sein d'une même ligne (parfois appelée « page »). L'adresse de colonne est incrémentée automatiquement au cours d'un *burst* pour obtenir les mots successifs. Elle repasse à zéro si au-delà de la longueur de ligne.
- La **longueur de burst** (*burst length* – BL) est le nombre de cellules lues / écrites en un accès. Typiquement 2, 4, 8 cellules ou page complète.

DRAM Synchrones

Application à l'exemple « PC1600 »

- Sans rafale : ~145 MB/s
- BL=2
 - $50 \text{ ns} + 2 * 5 \text{ ns} = 60 \text{ ns}$
 - ~16,66 MT de **2** mots/s
 - débit
 $= (16,66 \text{ MT/s}) * 2 * (8 \text{ B})$
 $= \sim 266,66 \text{ MB/s}$
- BL=4
 - $50 \text{ ns} + 4 * 5 \text{ ns} = 70 \text{ ns}$
 - ~14,29 MT de **4** mots/s
 - débit
 $= (\sim 14,29 \text{ MT/s}) * 4 * (8 \text{ B})$
 $= \sim 457,14 \text{ MB/s}$
- (...)



MT = million de transferts

DRAM Synchrones

Temps d'accès !!!

- En conclusion, le temps d'accès (des DRAM) a un **impact important sur le débit effectif** !
- Dans notre exemple, une mémoire DDR avec un débit théorique de 1600MB/s ne permet en pratique qu'un débit de 145 MB/s !
- Cependant, **si les données sont organisées intelligemment en mémoire** de façon à pouvoir lire plusieurs données situées à des adresses consécutives, il est possible d'effectuer des **lectures en “burst”** et de se rapprocher du débit théorique.

Table des Matières

Technologies RAM

- SRAM versus DRAM
- DRAM Synchrones (SDRAM)

→ Latence des DRAMs

Mémoire Cache

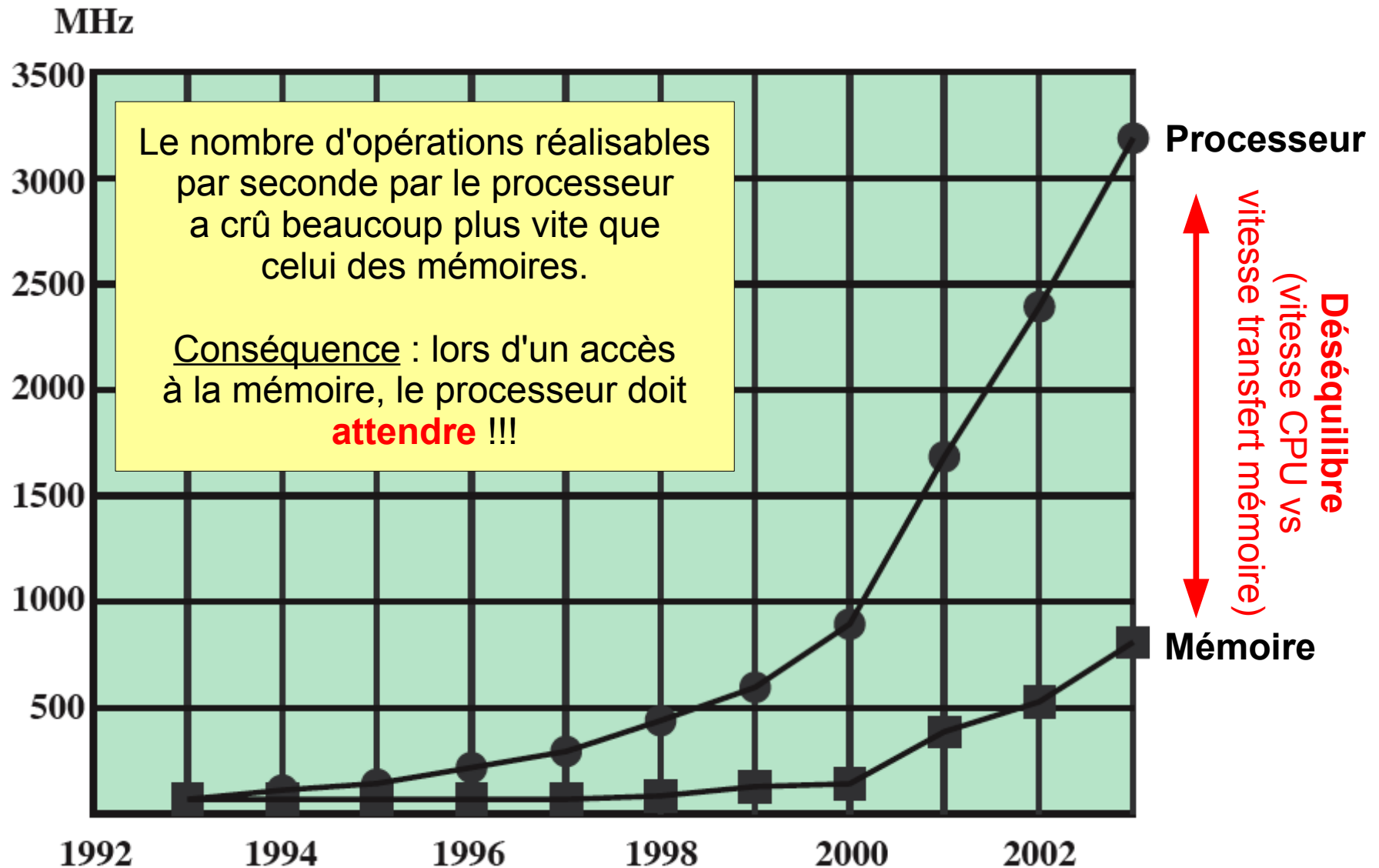
- Localité
- Principes de fonctionnement
- Organisation
- Implémentation *fully-associative* / *direct-mapped* / *set-associative*
- Remplacement et ré-écriture

Mémoire Virtuelle

- Traduction d'adresses
- Table des pages
- Translation Lookaside Buffer

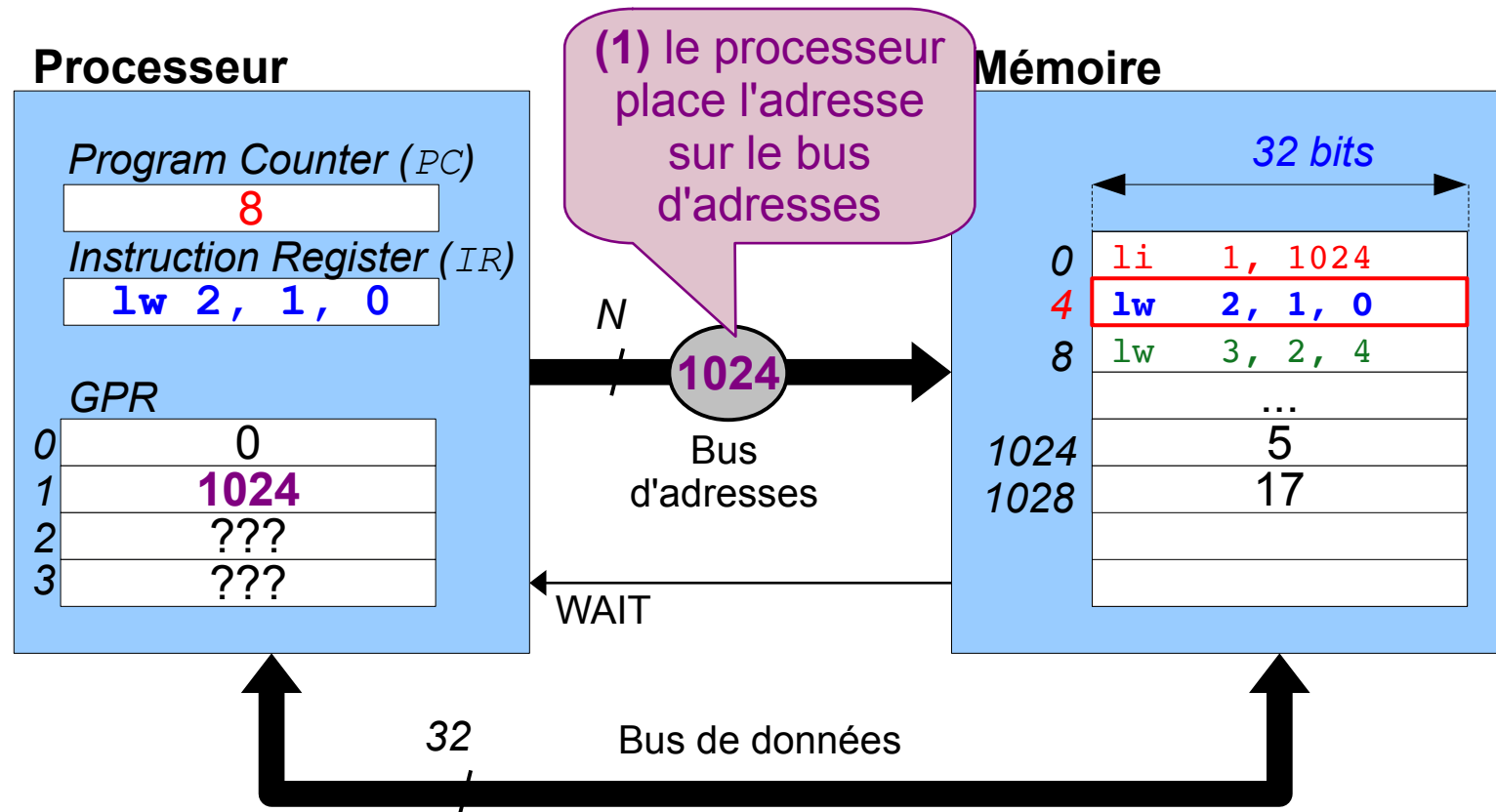
Goulot d'étranglement

Evolution à 2 vitesses...



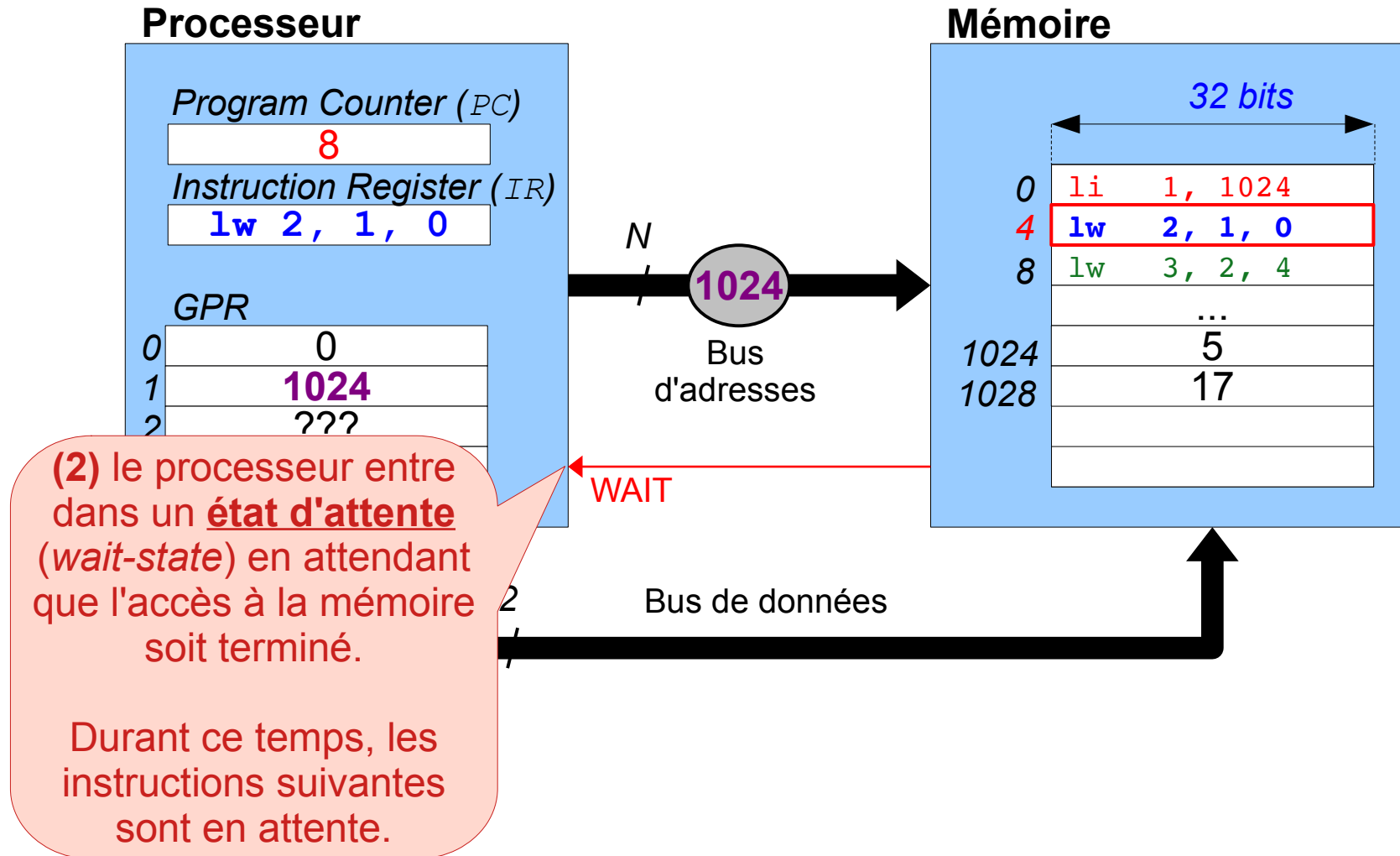
Goulot d'étranglement

Conséquences sur la Lecture en Mémoire



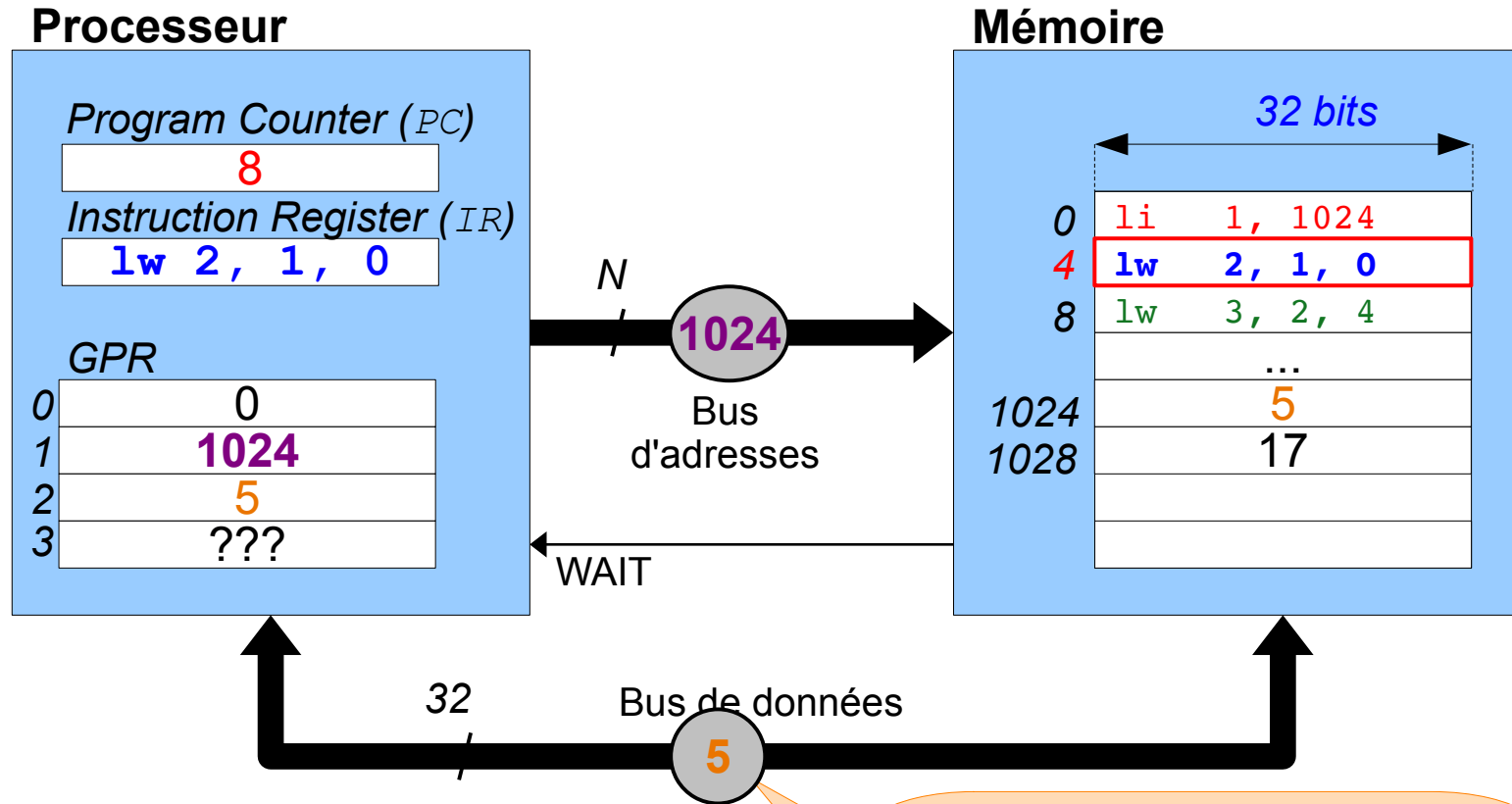
Goulot d'étranglement

Conséquences sur la Lecture en Mémoire



Goulot d'étranglement

Conséquences sur la Lecture en Mémoire



(3) la donnée lue est présentée par la mémoire sur le bus de données. Le processeur quitte l'état d'attente. L'exécution des instructions peut reprendre.

Goulot d'étranglement

Conséquences sur la Lecture en Mémoire

li	0, 1024
lw	0, 1, 0
lw	0, 2, 4
add	1, 1, 2

L'exécution de ces 4 instructions devrait prendre seulement **4 cycles CPU** mais la latence mémoire nécessite que le CPU attende durant 4 cycles supplémentaires, pour un total de **8 cycles**.

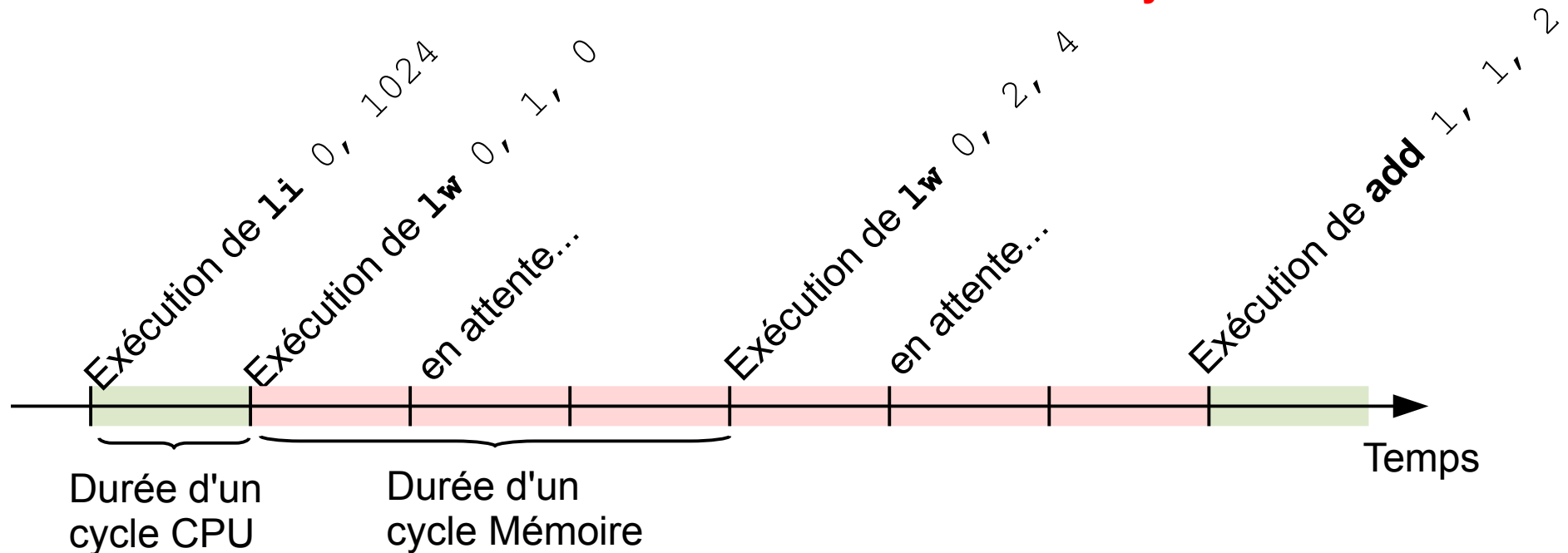


Table des Matières

Technologies RAM

- SRAM versus DRAM
- DRAM Synchrones (SDRAM)
- Latence des DRAMs

Mémoire Cache

Localité

- Principes de fonctionnement
- Organisation
- Implémentation *fully-associative* / *direct-mapped* / *set-associative*
- Remplacement et ré-écriture

Mémoire Virtuelle

- Traduction d'adresses
- Table des pages
- Translation Lookaside Buffer

Principe de localité

- Constatation : certains programmes ou parties de programmes effectuent des accès à la mémoire à des adresses proches les unes des autres. Il s'agit du **principe de localité**. Ce principe est à la base des mémoires cache !
- On distingue souvent deux types de localités
 - Localité temporelle : les cellules de mémoire accédées récemment ont une probabilité plus grande d'être accédées dans un futur proche.
 - Localité spatiale : les cellules de mémoire localisées à des adresses proches des cellules de mémoire accédées récemment ont une probabilité plus grande d'être accédées dans un futur proche.

Localité

Exemple – Localité des instructions

- Une boucle est un exemple typique de localité temporelle et spatiale : une série d'instructions consécutives en mémoire va être exécutée plusieurs fois.

Boucle en langage C

```
sum= 0;  
i= 0;  
  
while (i > 10) {  
    sum= sum + 1;  
    i= i+1;  
}
```

```
0x00000000  
0x00000004  
0x00000008  
  
0x0000000C  
0x00000010  
0x00000014  
0x00000018
```

↓
adresse mémoire
des instructions

Traduction en langage d'assemblage

```
li    $t0, 0  
li    $t1, 0  
li    $t2, 10  
while:  
    beq    $t0, $t2, end_while  
    addi   $t0, $t0, 1  
    addi   $t1, $t1, 1  
    j      while  
end_while:
```

- Typiquement : un programme passe **90%** de son temps dans **10%** de ses instructions.

Localité

Exemple – Localité des instructions

- Dans cet exemple, les instructions des adresses 0x0000000C à 0x00000018 sont lues plusieurs fois
- Il est donc intéressant de diminuer le coût d'accès mémoire pour ces adresses !

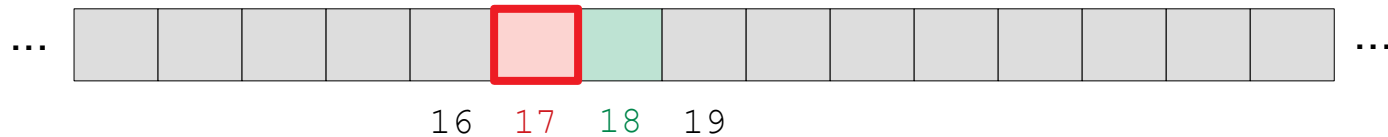
```
0x00000000    li    $t0, 0
0x00000004    li    $t1, 0
0x00000008    li    $t2, 10
               while:
0x0000000C    beq   $t0, $t2, end_while
0x00000010    addi  $t0, $t0, 1
0x00000014    addi  $t1, $t1, 1
0x00000018    j     while
               end_while:
```

Ordre d'accès	Adresse "fetchée"
0	0x00000000
1	0x00000004
2	0x00000008
3	0x0000000C
4	0x00000010
5	0x00000014
6	0x00000018
7	0x0000000C
8	0x00000010
9	0x00000014
...	...

Localité

Exemple – Localité spatiale des données

- Données multimedia : Lorsqu'un programme accède à des données multimedia (p.ex. vidéo, MP3, ...), il s'agit souvent de les lire séquentiellement. Si une vidéo est actuellement lue au temps 0:42:17, il y a une grande probabilité que prochainement les données relatives au temps 0:42:18 soient lues...



- Données bureautiques (p.ex. document texte, tableur). Un utilisateur (et donc le programme d'édition) ne touche généralement qu'à une partie du document à un moment donné.

A screenshot of a PDF viewer toolbar. The toolbar includes a search icon, a text field containing 'pdfLaTeX', and various editing and navigation icons such as back, forward, search, and print. Below the toolbar, a text box contains the following text:

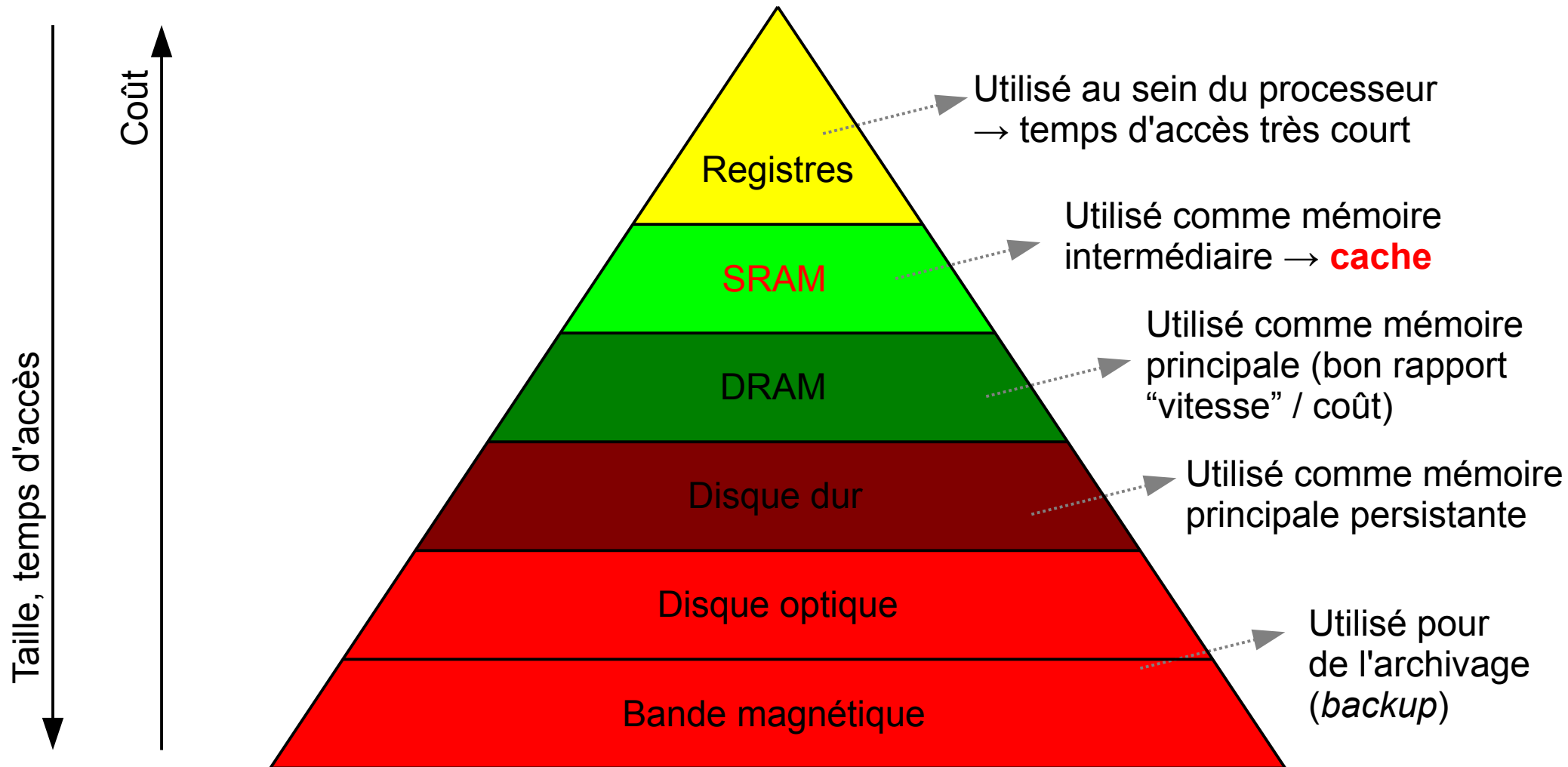
Examen du cours de Fonctionnement des Ordinateurs
Question 1 : calculez le *hit ratio* d'une cache de type *direct-mapped* lors de l'exécution du

Mémoire cache

- La localité des données et des instructions a inspiré l'insertion d'une mémoire spéciale appelée **mémoire cache** entre le processeur et la mémoire centrale.
 - 1) Toute donnée lue en mémoire centrale est **aussi copiée en mémoire cache**.
 - 2) Une fois une donnée placée en cache, **les accès suivants à celle-ci sont moins longs** car la mémoire principale ne doit pas être accédée.
 - 3) La mémoire cache repose sur la SRAM, plus rapide, mais aussi plus coûteuse, que la DRAM utilisée pour la mémoire centrale. La **quantité de mémoire cache est petite** en regard de celle de la mémoire centrale.
- Dans un système informatique, on essaye de placer dans les mémoires les plus rapides les données qui sont accédées le plus fréquemment. Cela donne lieu à une **hiérarchie de mémoire**.

Localité

Hiérarchie de Mémoires



Hiérarchie de mémoires

- Ordre de grandeur des caractéristiques de différents types de mémoires

	Taille (MB)	Temps d'accès (μ s)	Bande- passante (MB/s)	Coût (\$/MB)
Registre (flip-flops)	0.001	0.001	10000	100
SRAM	1	0.01	1000	5
DRAM	1000	0.1	100	0.05
Disque dur	1000000	1000	10	0.0005

Table des Matières

Technologies RAM

- SRAM versus DRAM
- DRAM Synchrones (SDRAM)
- Latence des DRAMs

Mémoire Cache

- Localité

Principes de fonctionnement

- Organisation
- Implémentation *fully-associative* / *direct-mapped* / *set-associative*
- Remplacement et ré-écriture

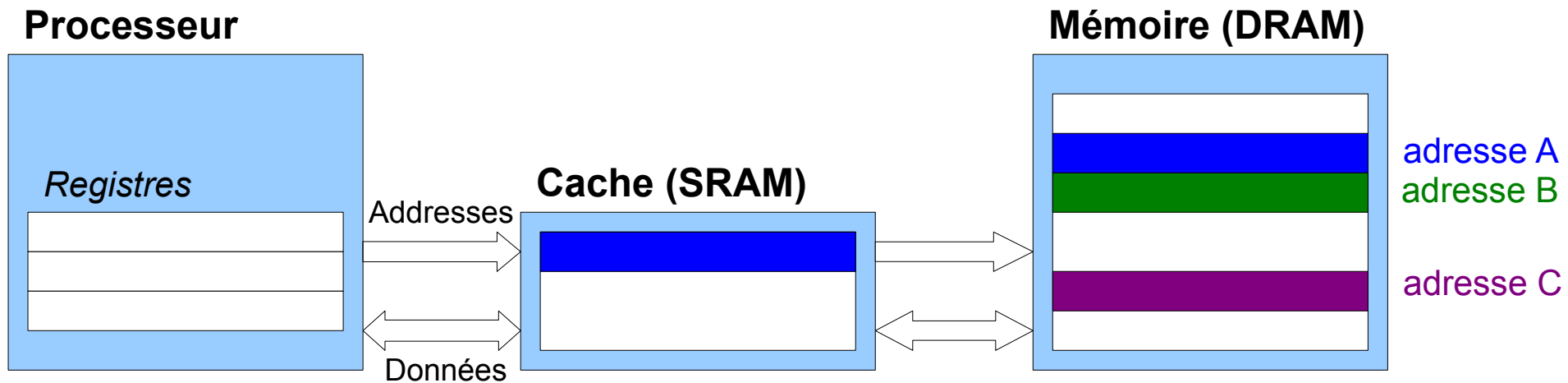
Mémoire Virtuelle

- Traduction d'adresses
- Table des pages
- Translation Lookaside Buffer

Principe des mémoires caches

Principe

- Une **mémoire cache** est une mémoire intermédiaire placée entre le processeur et la mémoire principale.

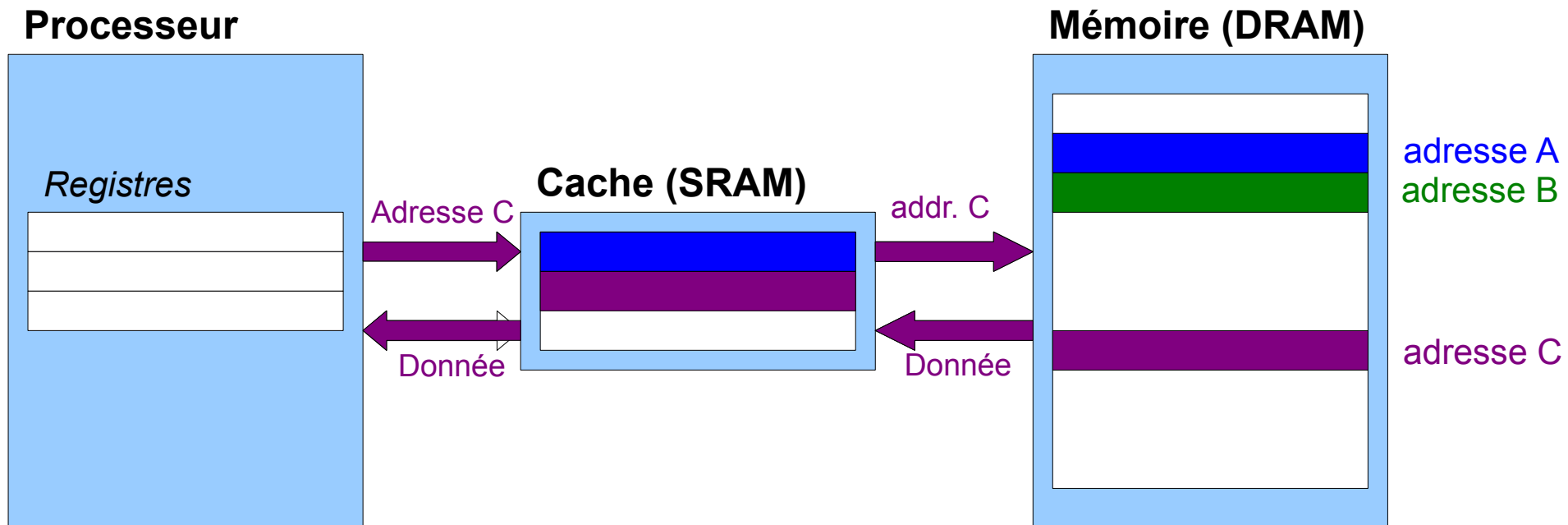


- La cache est plus petite que la mémoire principale (p.ex. 1M de cache versus 4GB de RAM).
- La cache contient un sous-ensemble de la mémoire principale. Le contenu de la cache évolue au cours du temps en fonction du comportement des programmes.
- La cache (SRAM) est plus rapide que la mémoire principale (DRAM).

Principe des mémoires caches

Cache miss

- Un **cache miss** est un accès mémoire à une adresse dont le contenu n'est pas actuellement en cache. Il est nécessaire d'effectuer une lecture en mémoire principale (lente).

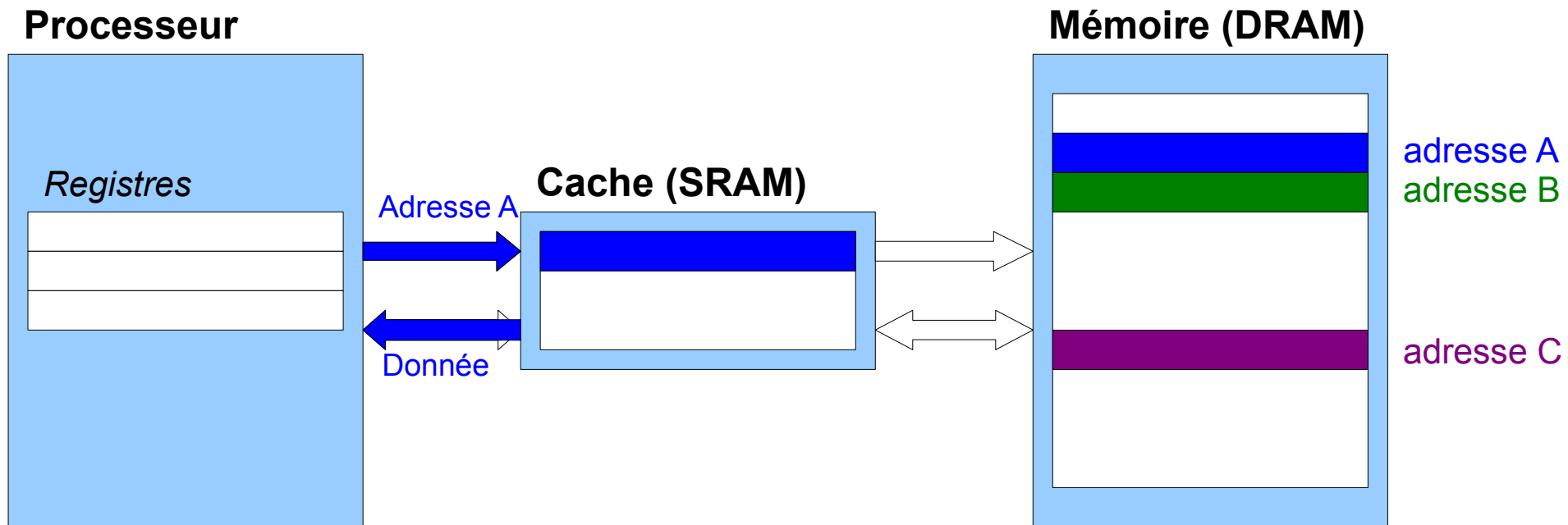


Dans l'exemple, la cellule mémoire d'**adresse C** est accédée par le processeur. Cette cellule ne se trouve pas encore en cache (c'est un *miss*). Un accès à la mémoire principale est requis. Après quoi, la donnée est placée en cache.

Principe des mémoires caches

Cache hit

- Un **cache hit** est un accès mémoire à une adresse dont le contenu se situe en cache. Aucun accès à la mémoire principale n'est nécessaire.



Dans l'exemple, la cellule mémoire d'*adresse A* est accédée par le processeur. Cette cellule se déjà trouve en cache (c'est un *hit*).

Principe des mémoires caches

Types de *cache misses*

- ***Compulsory miss***
 - lors du 1^{er} accès à un mot mémoire
- ***Capacity miss***
 - donnée éjectée de la cache
 - en raison de la taille limitée de la cache
- ***Conflict miss***
 - donnée éjectée de la cache
 - en raison de l'impossibilité de se trouver en cache en même temps qu'une autre donnée (voir plus loin « *direct-mapped caches* »)

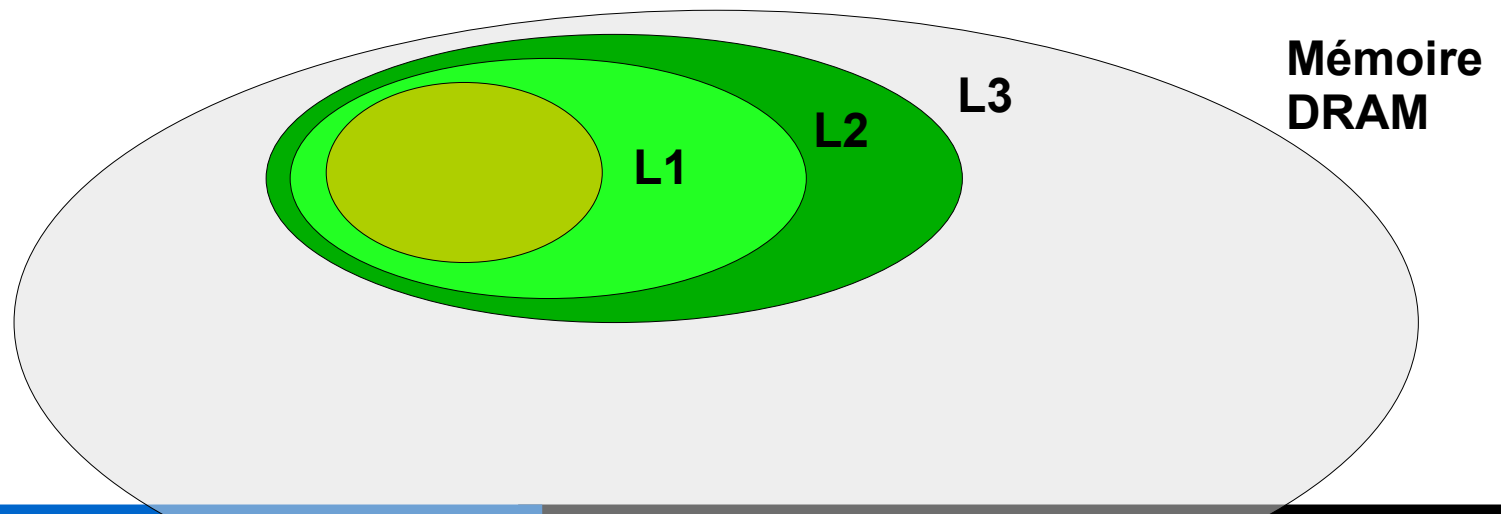
Principes des mémoires caches

Caches secondaires

- La mémoire cache est très coûteuse. Pour améliorer les performances à coût réduit, un **second niveau (L2)** voire un **troisième niveau (L3)** de cache sont introduits.
- Ces caches secondaires ont une latence plus grande que celle de la cache **L1**.

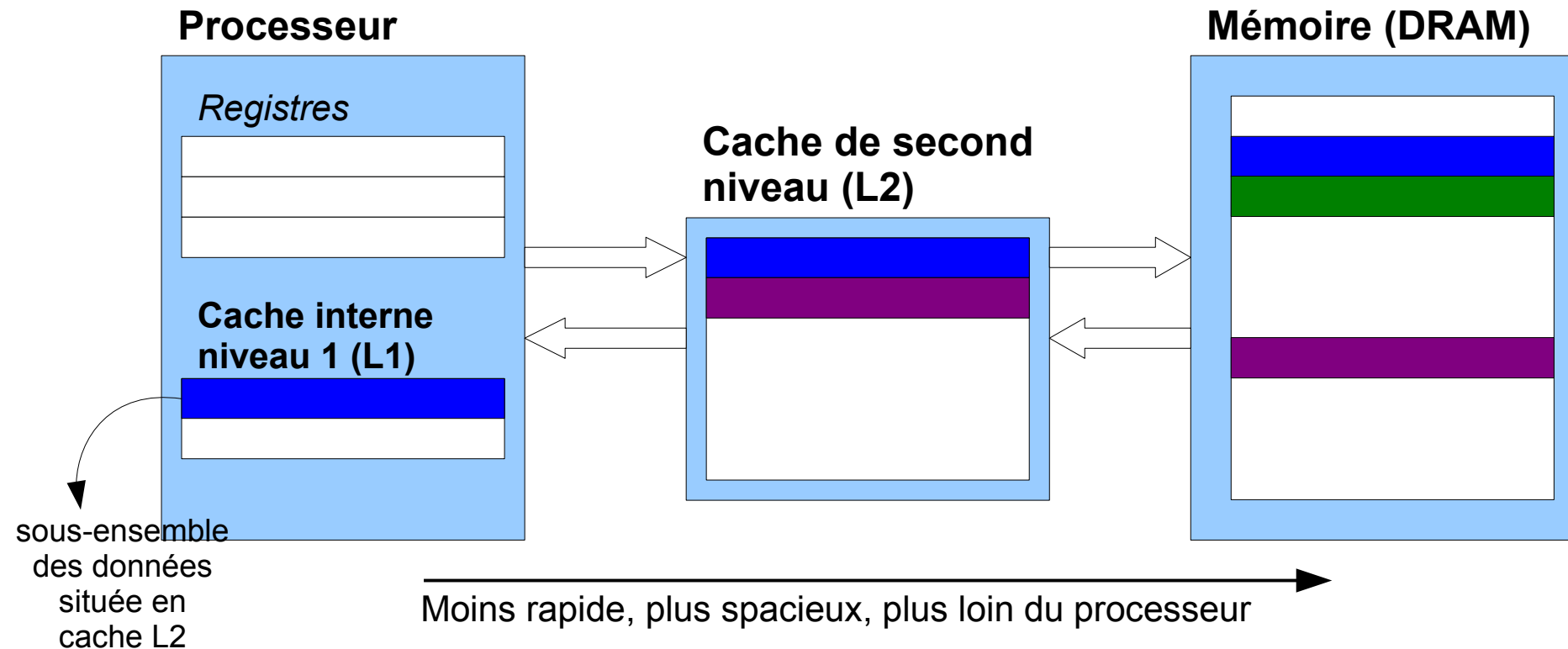
$$\text{latence(L1)} < \text{latence(L2)} < \text{latence(L3)} < \text{latence(mémoire)}$$

- Une cache secondaire contiendra l'ensemble des données situées dans la cache de niveau inférieur + d'autres données.



Principes des mémoires caches

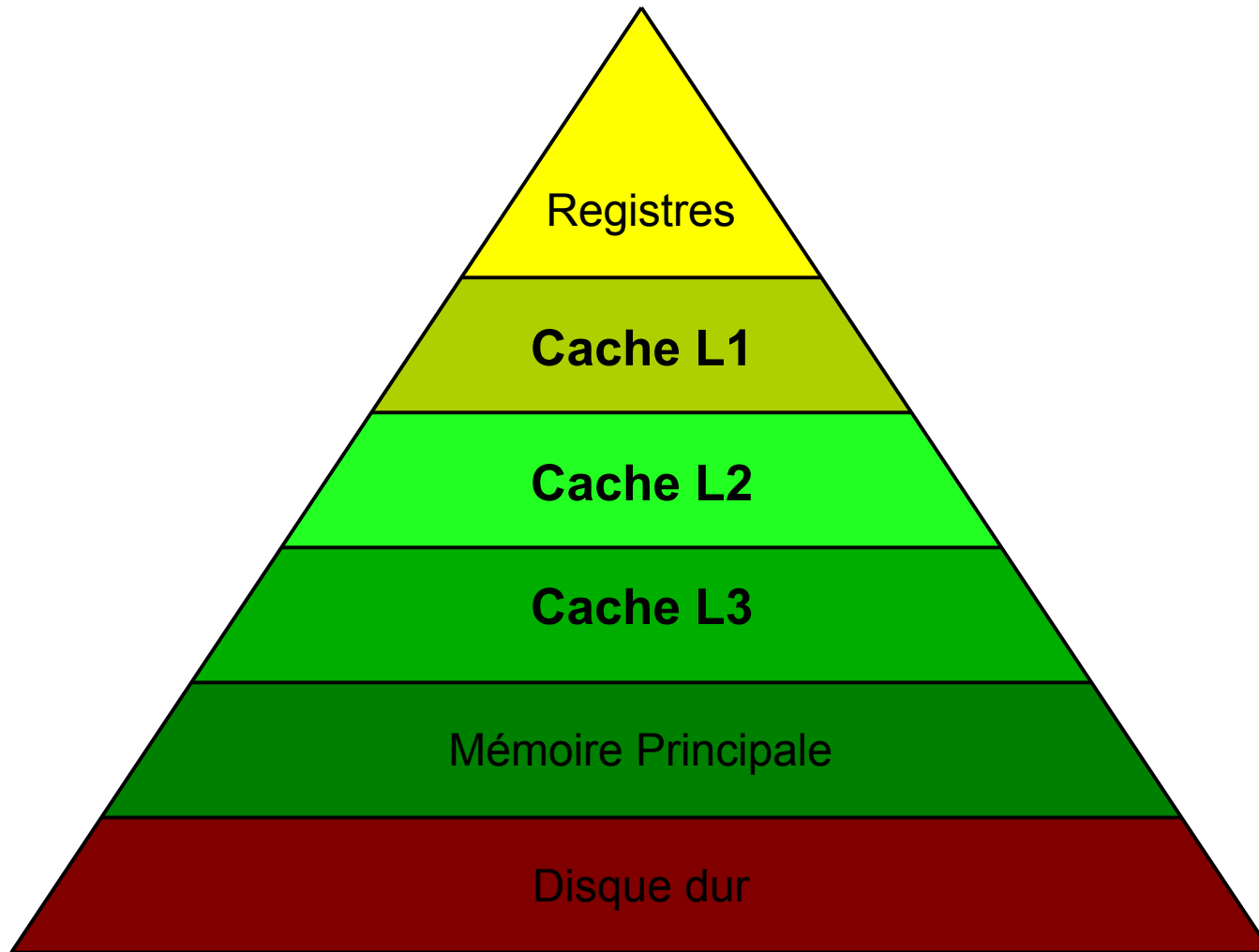
Caches secondaires



Note : Souvent, la cache L1 est intégrée au processeur. On parle alors de *on-chip* ou *on-die* cache. Beaucoup de processeurs multi-coeurs ont également une cache L2 *on-die*.

Principes des mémoires caches

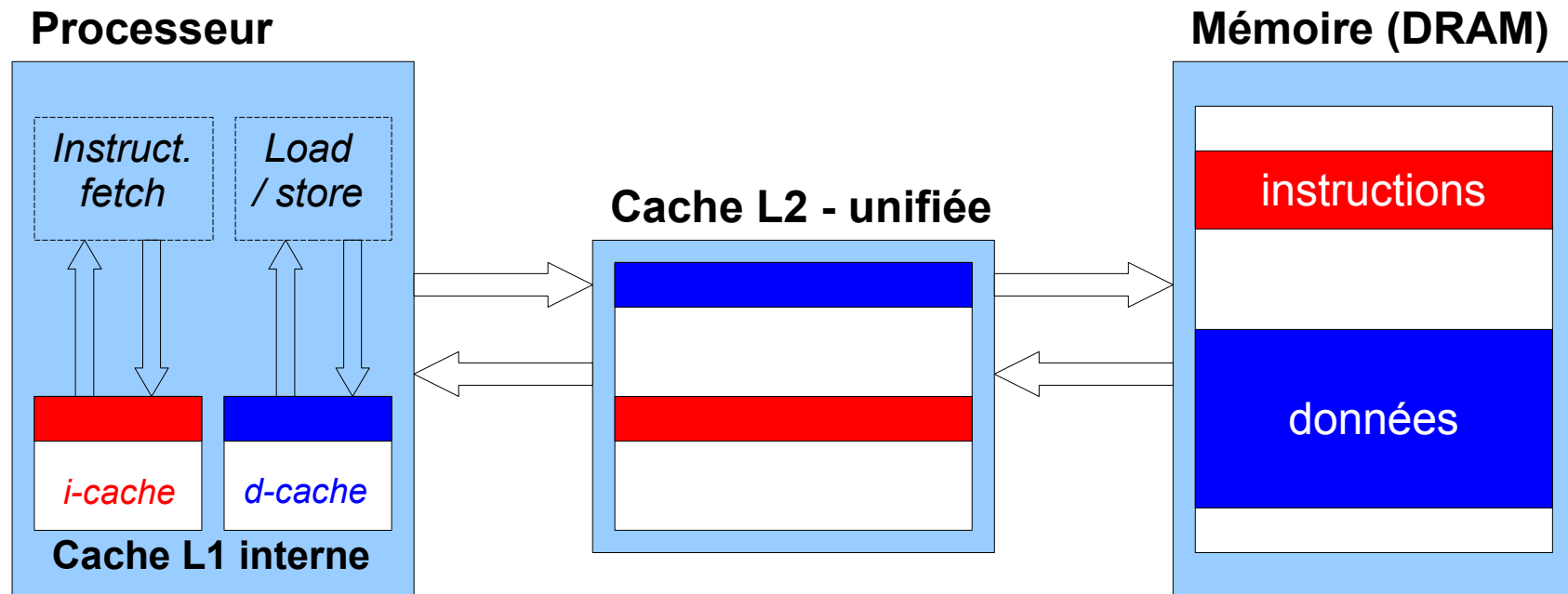
Hiérarchie de caches



Principes des mémoires caches

Caches séparées pour instructions/données

- La cache de premier niveau est souvent séparée pour les instructions (*fetch*) et pour les données (*load/store*). On parle alors d'*instruction-cache* (*i-cache*) et de *data-cache* (*d-cache*).
- Les caches qui ne font pas la distinction instructions/données sont appelées caches unifiées (*unified cache*).



Principes des mémoires caches

Performances de la cache

- Avant de mettre en oeuvre une mémoire cache, il faut s'assurer que cette cache va amener un gain de performance. Considérons un système avec 1 seul niveau de cache.
- Pour rappel,
 - **Miss** = accès à une adresse non présente en cache → accès nécessaire à la mémoire principale (lent).
 - **Hit** = accès à une adresse présente en cache → pas d'accès à la mémoire principale (rapide).
- On note N_{MISS} , le nombre de **miss**, et N_{HIT} , le nombre de **hit**.
- Le *Miss ratio* R_{MISS} (resp. *Hit ratio* R_{HIT}) est la proportion des accès qui sont des **miss**. (resp. des **hit**).

$$R_{MISS} = \frac{N_{MISS}}{N_{HIT} + N_{MISS}}$$

$$R_{HIT} = \frac{N_{HIT}}{N_{HIT} + N_{MISS}}$$

Principes des mémoires caches

Performances de la cache

- Etant donnés
 - T_{CACHE} = temps d'accès à la mémoire cache.
 - T_{MEM} = temps d'accès à la mémoire principale.
- Comparons le temps nécessaire pour accéder à la mémoire en moyenne dans deux cas distincts : l'un **sans cache** et l'autre **avec cache**.
- Système sans cache
 - Temps d'accès moyen sans cache $T_{MOY_MEM} = T_{MEM}$
- Système avec cache
 - Temps d'accès moyen avec cache $T_{MOY_CACHE} = T_{CACHE} + T_{MEM} \times R_{MISS}$
 - la cache est toujours accédée
 - la mémoire principale est accédée pour chaque miss

Principes des mémoires caches

Performances de la cache

- Pour que le système avec cache soit plus performant que celui sans cache, il faut que le temps d'accès moyen obtenu avec la mémoire cache, T_{MOY_CACHE} , soit strictement inférieur à celui obtenu sans utilisation de cache, i.e. T_{MOY_MEM}
- Pour cela, il faut que

$$T_{MOY_CACHE} = T_{CACHE} + T_{MEM} \times R_{MISS} < T_{MOY_MEM} = T_{MEM}$$

ou encore que

$$T_{CACHE} + T_{MEM} \times R_{MISS} < T_{MEM}$$

$$T_{CACHE} < T_{MEM} - T_{MEM} \times R_{MISS}$$

$$\frac{T_{CACHE}}{T_{MEM}} < 1 - R_{MISS}$$

$$\frac{T_{CACHE}}{T_{MEM}} < R_{HIT}$$

Principes des mémoires caches

Exemple

- Supposons un système tel que $T_{CACHE} = 10ns$ et $T_{MEM} = 100ns$
- Sur base de mesures du fonctionnement des programmes qui seront exécutés sur ce système, on s'attend à un *hit ratio* $R_{HIT} = 50\%$ (i.e. 50% des accès mémoire se font à partir de la cache)
- Performances **avec** la cache
 - Temps moyen d'accès = $10ns + 100ns * 50\% = 60ns$
- Performances **sans** la cache
 - Temps moyen d'accès = $100ns$
- Le système avec cache tient la route pour le *hit ratio* prévu. Ce que nous aurions pu vérifier avec

$$\frac{T_{CACHE}}{T_{MEM}} = 0,1 < R_{HIT} = 0,5$$

Table des Matières

Technologies RAM

- SRAM versus DRAM
- DRAM Synchrones (SDRAM)
- Latence des DRAMs

Mémoire Cache

- Localité
- Principes de fonctionnement

Organisation

- Implémentation *fully-associative* / *direct-mapped* / *set-associative*
- Remplacement et ré-écriture

Mémoire Virtuelle

- Traduction d'adresses
- Table des pages
- Translation Lookaside Buffer

Organisation

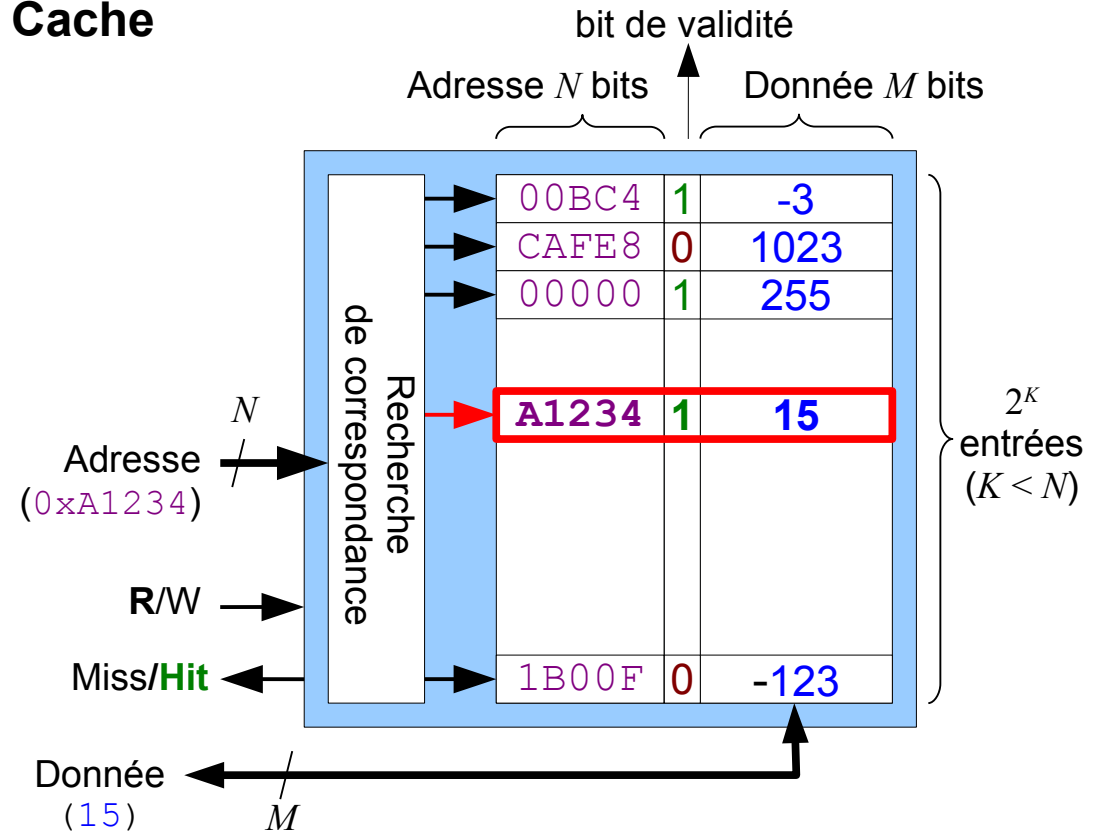
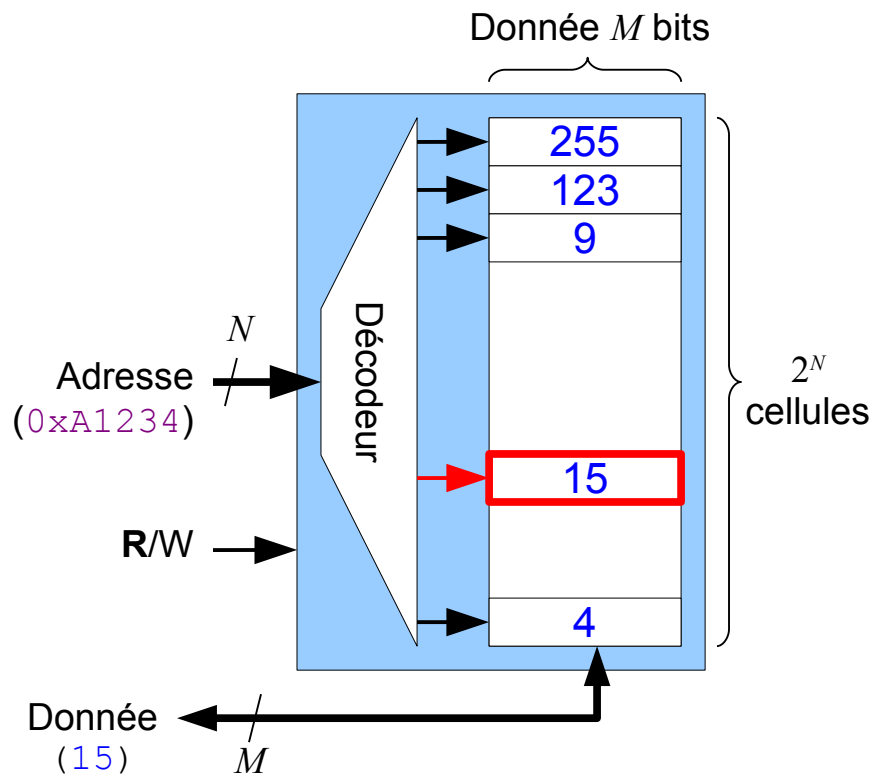
RAM versus Cache

- Mémoire cache et RAM ne fonctionnent pas de la même façon.
- **RAM : 2^N cellules de M bits** (taille = $2^N * M / 8$ octets)
 - Cellule = uniquement donnée.
 - Nombre de cellules = nombre d'adresses → Il y a une correspondance unique entre une adresse A et sa cellule.
 - Rôle : donner accès à la valeur de la cellule d'adresse A
- **Cache : 2^K entrées** ($K \ll N$)
 - Entrée de la cache peut contenir une donnée se trouvant à n'importe quelle adresse en RAM.
 - Entrée = triplet (adresse en RAM, bit de validité, donnée).
 - Rôle : donner accès à la valeur de la cellule d'adresse A si cette cellule est présente (Hit) sinon retourner une erreur (Miss).

Organisation

RAM versus Cache

- Illustration des différences entre RAM (à gauche) et Cache (à droite).





Recherche de correspondance en Cache

- Comment retrouver en cache la donnée associée à une adresse A ?
- Première méthode : recherche exhaustive

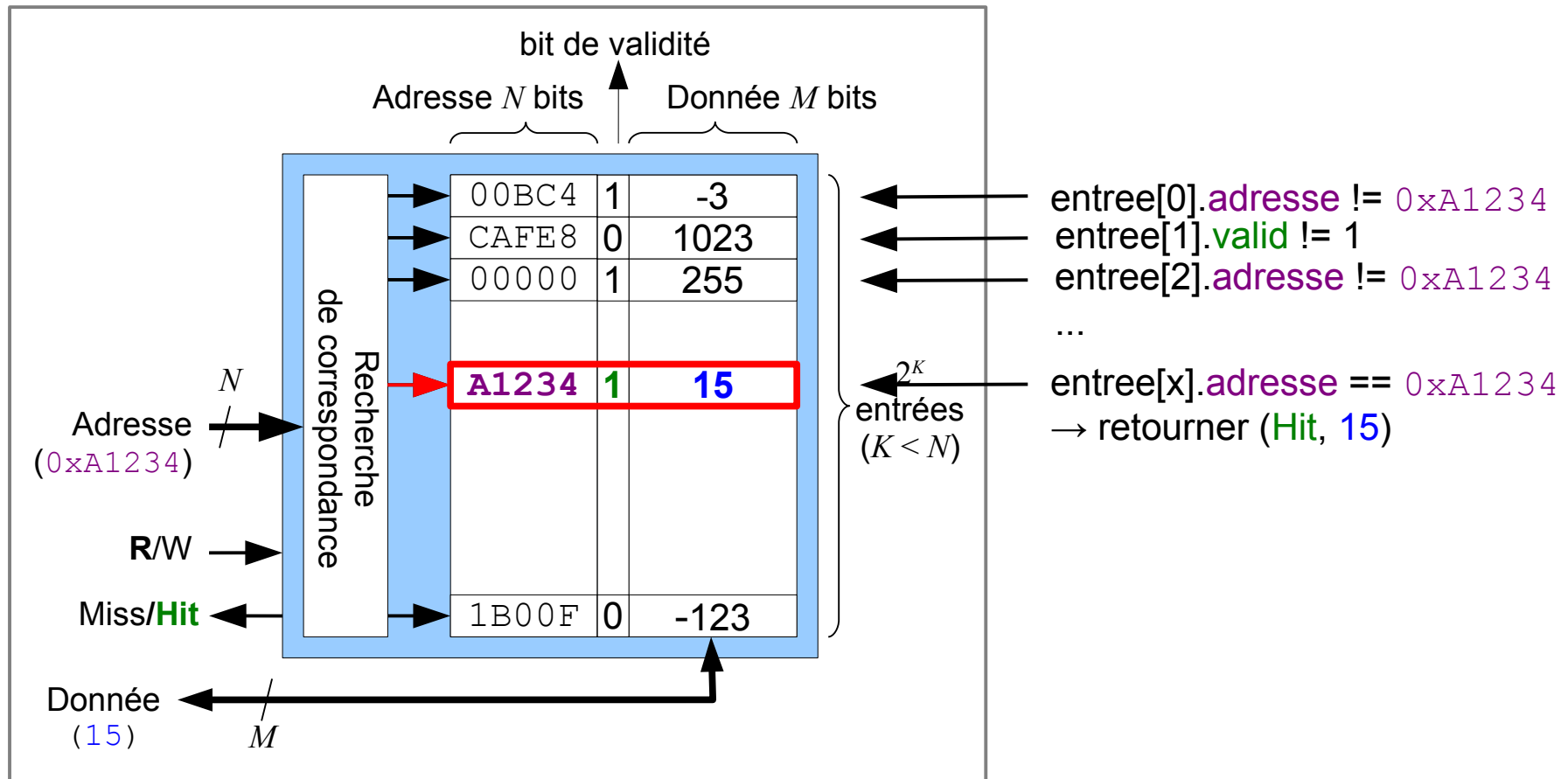
```
pour chaque entrée  $i$  ( $0 \leq i < 2^K$ ) effectuer  
  si ( entree[ $i$ ].valid == 1 ) alors  
    si ( entree[ $i$ ].adresse ==  $A$  ) alors  
      /* La donnée est trouvée */  
      retourner (Hit, entree[ $i$ ].donnee).  
  /* La donnée n'est pas trouvée */  
  retourner (Miss, ?).
```

- **Inconvénient** : la recherche exhaustive n'est pas efficace
 - Un **Hit** nécessite en moyenne le parcours de 2^{K-1} entrées
 - Un **Miss** nécessite le parcours de toutes les entrées (2^K)



Exemple – Cache hit

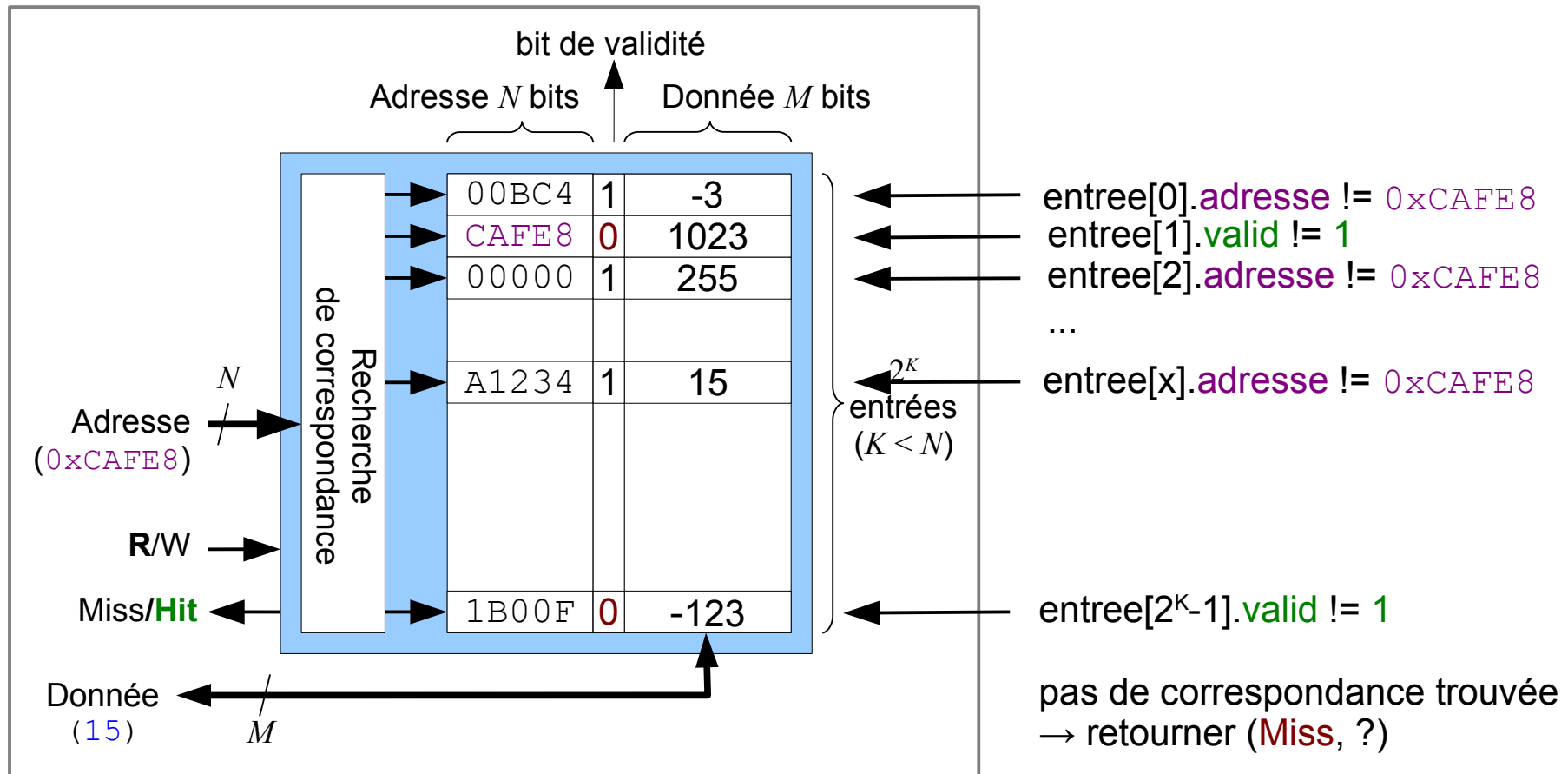
- Exemple : recherche de la cellule d'adresse $0xA1234$





Exemple – Cache miss

- Exemple : recherche de la cellule d'adresse $0x\text{CAFE}8$



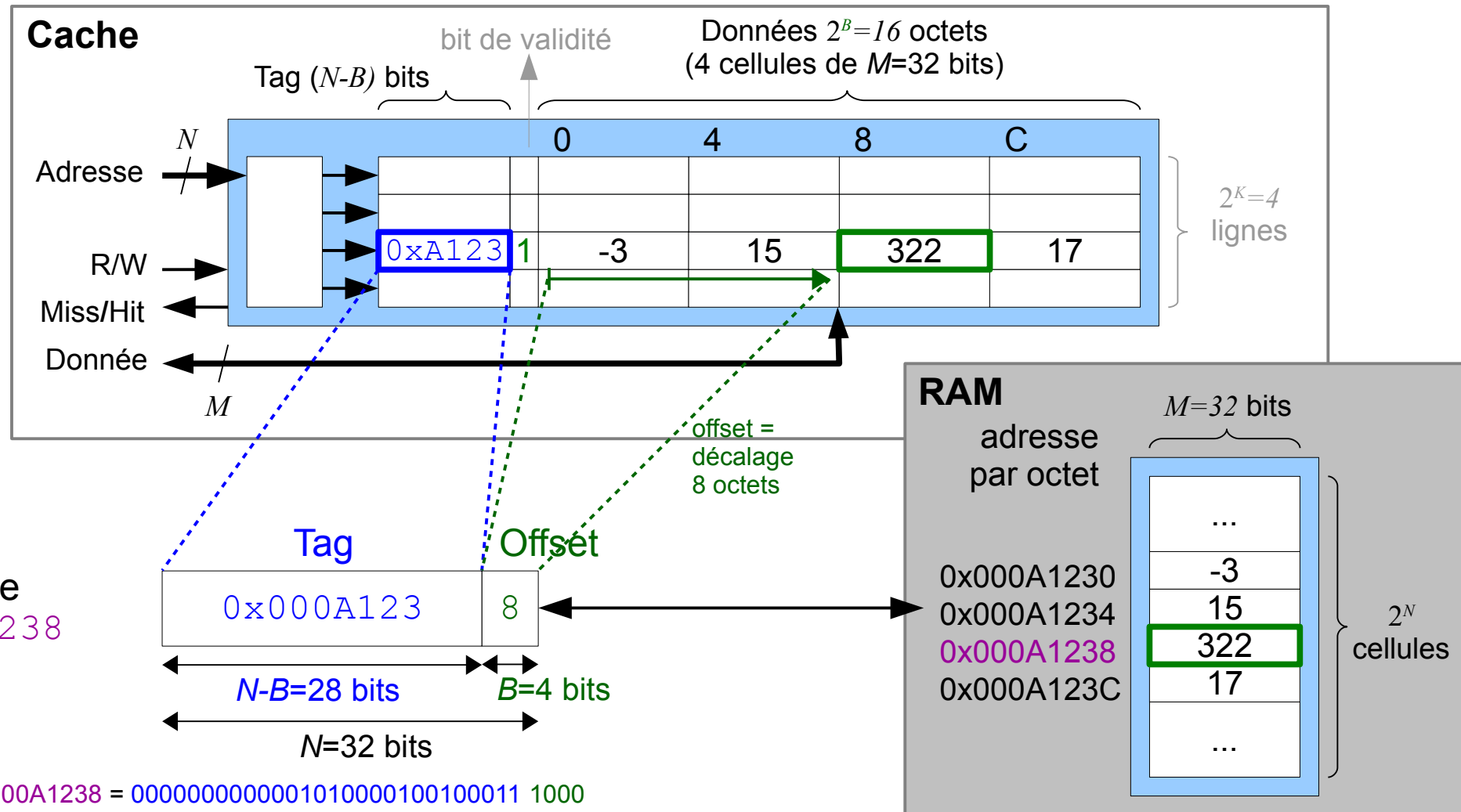
Organisation

Multiples cellules dans une entrée de cache

- Les entrées des caches contiennent souvent plusieurs cellules de RAM
→ entrées nommées **lignes de cache**
 - Exploiter la localité spatiale des données (p.ex. cas des cellules d'un tableau)
 - Favoriser les lectures en « burst » dans la RAM : permettent d'amortir le temps d'accès des DRAM sur plusieurs lectures à des adresses consécutives).
- Supposons que chaque ligne contienne 2^B octets.
 - L'adresse A d'une ligne désigne le premier octet de la ligne.
 - Les adresses des cellules suivantes sont obtenues en ajoutant un décalage (**offset**) par rapport à l'adresse A .
 - L'adresse A est alignée sur la taille d'une entrée → les B bits de poids faible de A valent 0 → ne stocker que les $N-B$ bits de poids fort de l'adresse, appelés **tag**.

Organisation

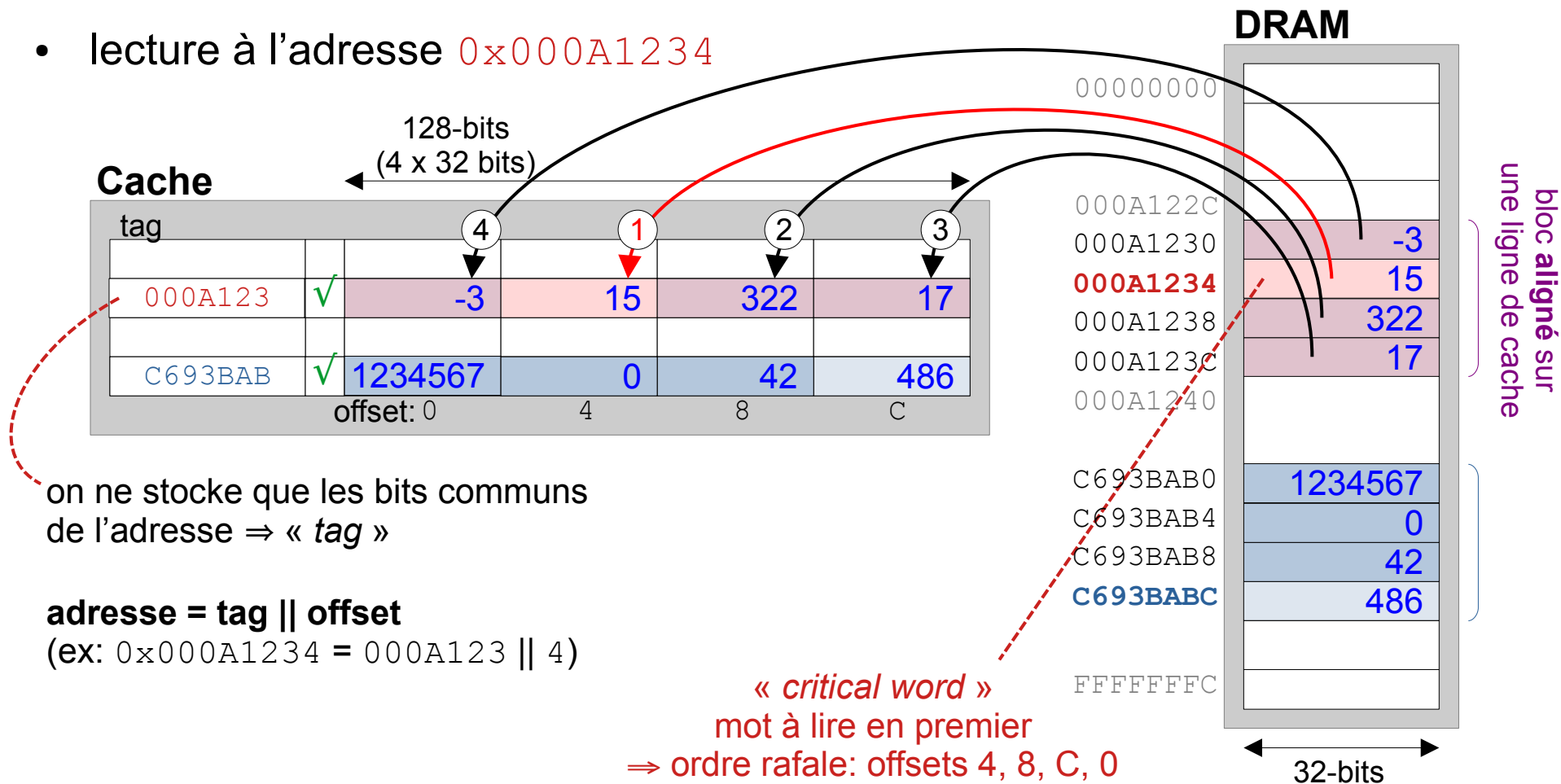
Multiples cellules dans une entrée de cache



Organisation

Cache + lecture DRAM en rafale

- cellule mémoire 32-bits, ligne cache 4 x 32-bits
- lecture DRAM en rafale, nombre de cellules BL=4
- lecture à l'adresse $0 \times 000A1234$



Organisation

Stratégies d'organisation

- Il existe plusieurs moyens d'organiser une mémoire cache. Nous allons nous intéresser à 3 aspects de cette organisation et étudier plusieurs stratégies.
- **Structure de la cache**
 - Correspondance entre un *tag/adresse* et une donnée. Nous allons étudier trois stratégies: *fully-associative*, *direct-mapped* et *set-associative*.
- **Stratégie de remplacement**
 - Lorsqu'une nouvelle entrée doit être stockée dans la cache, il faut décider quelle entrée va être remplacée. Nous évoquerons deux stratégies: *Least Recently Used* (LRU) et *aléatoire*.
- **Stratégie de ré-écriture en mémoire**
 - Après une écriture, le contenu de la cache doit éventuellement être synchronisé avec le contenu de la mémoire. Deux stratégies: *write-through* et *write-back*.

Table des Matières

Technologies RAM

- SRAM versus DRAM
- DRAM Synchrones (SDRAM)
- Latence des DRAMs

Mémoire Cache

- Localité
 - Principes de fonctionnement
 - Organisation
- ➡ Implémentation *fully-associative* / *direct-mapped* / *set-associative*
- Remplacement et ré-écriture

Mémoire Virtuelle

- Traduction d'adresses
- Table des pages
- Translation Lookaside Buffer

Structure des caches

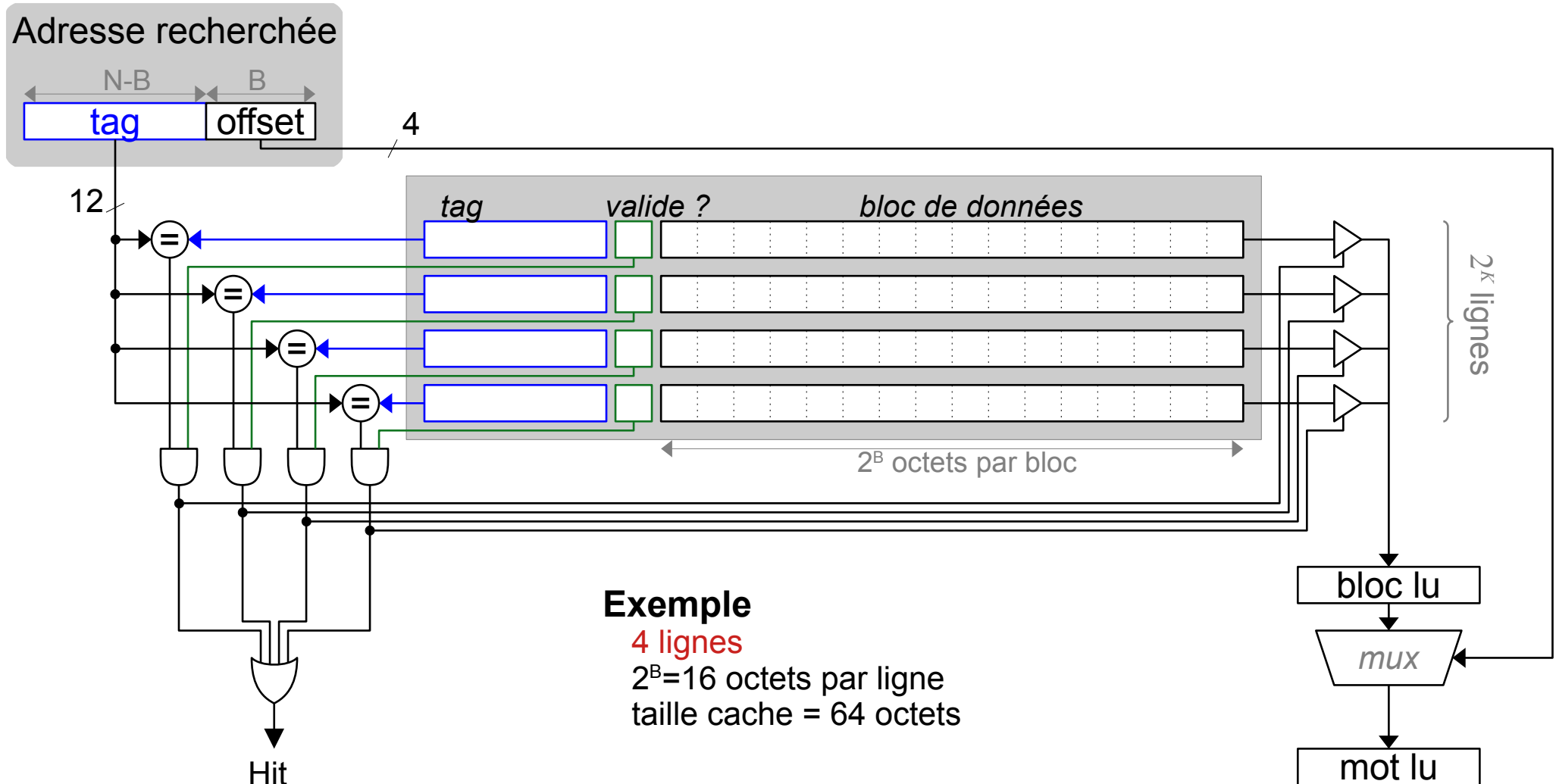
Cache fully-associative

- Une cache **fully-associative** permet de stocker des données provenant de la mémoire dans n'importe quelle ligne de la cache.
- Chaque ligne de cache contient un *tag*, un bit de validité et un bloc de données. Le *tag* indique l'adresse (en DRAM) d'où provient le bloc de données.
- Parcourir séquentiellement toutes les lignes de la cache pour trouver l'adresse recherchée prendrait trop de temps. Par conséquent, **l'adresse est comparée au tags de toutes les lignes en même temps** ! Cette approche nécessite un comparateur par ligne (coûte cher et consomme plus d'énergie).

Structure des caches

Cache fully-associative

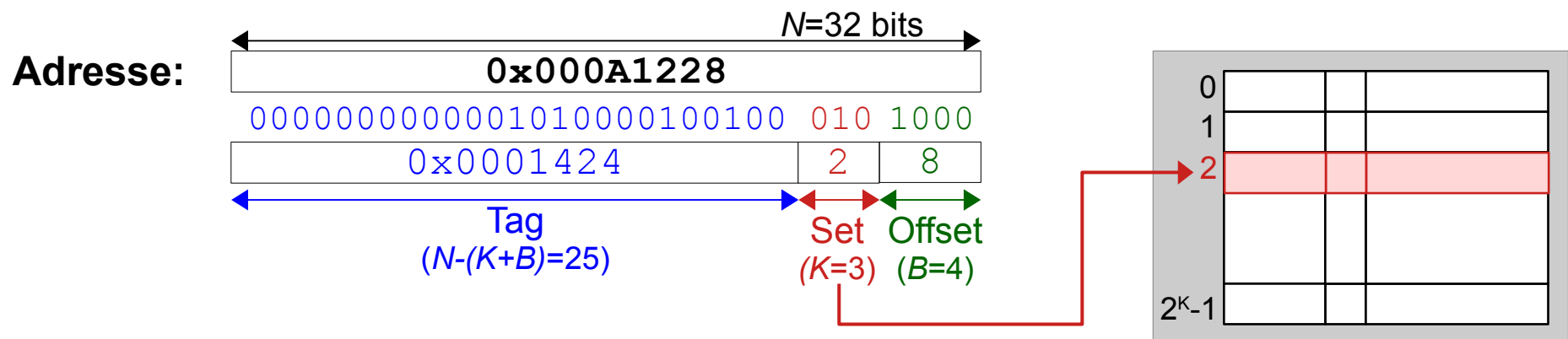
- Les *tags* de toutes les lignes sont comparés en parallèle.



Structure des caches

Cache direct-mapped

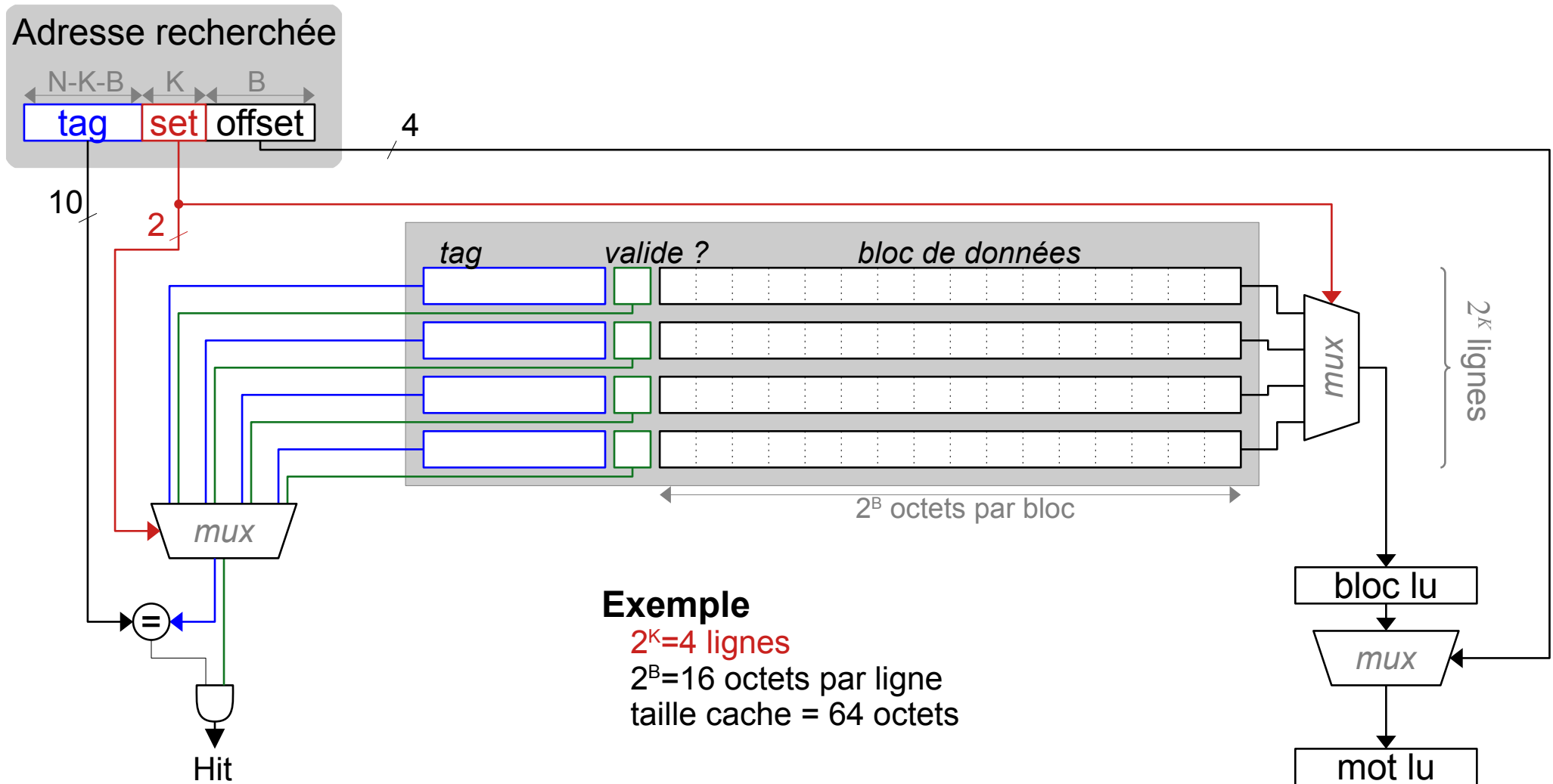
- La cache **direct-mapped** est une réponse aux inconvénients de la cache *fully-associative*. Elle permet de garder les temps d'accès les plus courts possibles tout en évitant d'avoir un comparateur par ligne.
- La ligne où stocker un bloc est désignée directement par un sous-ensemble des bits de l'adresse.
- Une adresse est découpée comme suit
 - **Offset** (B bits) : positionne chaque octet dans la ligne.
 - **Set** (K bits) : sélectionne une ligne parmi 2^K lignes
 - **Tag** ($N-K-B$ bits) : permet de vérifier si la ligne contient la cellule d'adresse recherchée.



Structure des caches

Cache direct-mapped

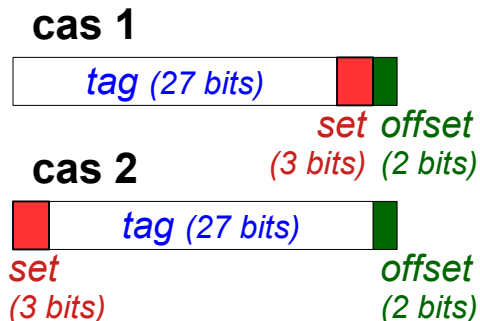
- La partie *set* de l'adresse désigne une ligne unique pour une adresse.



Structure des caches

Cache direct-mapped

- Pourquoi **set** est-il pris « au milieu » de l'adresse et pas à une extrémité ?
 - Localité spatiale \Rightarrow adresses ont un préfixe commun
 \Rightarrow préférable de prendre **set** dans le suffixe de l'adresse (bits de poids faible) plutôt que dans le préfixe (bits de poids fort)
- Illustration
 - Soit cache de 8 lignes ($K=3$) de 4 octets ($B=2$)
 - Séquence d'accès 0xA1230, 0xA1234, 0xA1238, 0xA123C



Adresse	set (cas 1)	set (cas 2)
0x000A1230	100	000
0x000A1234	101	000
0x000A1238	110	000
0x000A123C	111	000

Observation : le cas 2 donne une même valeur de **Set** à des adresses proches :-(

Structure des caches

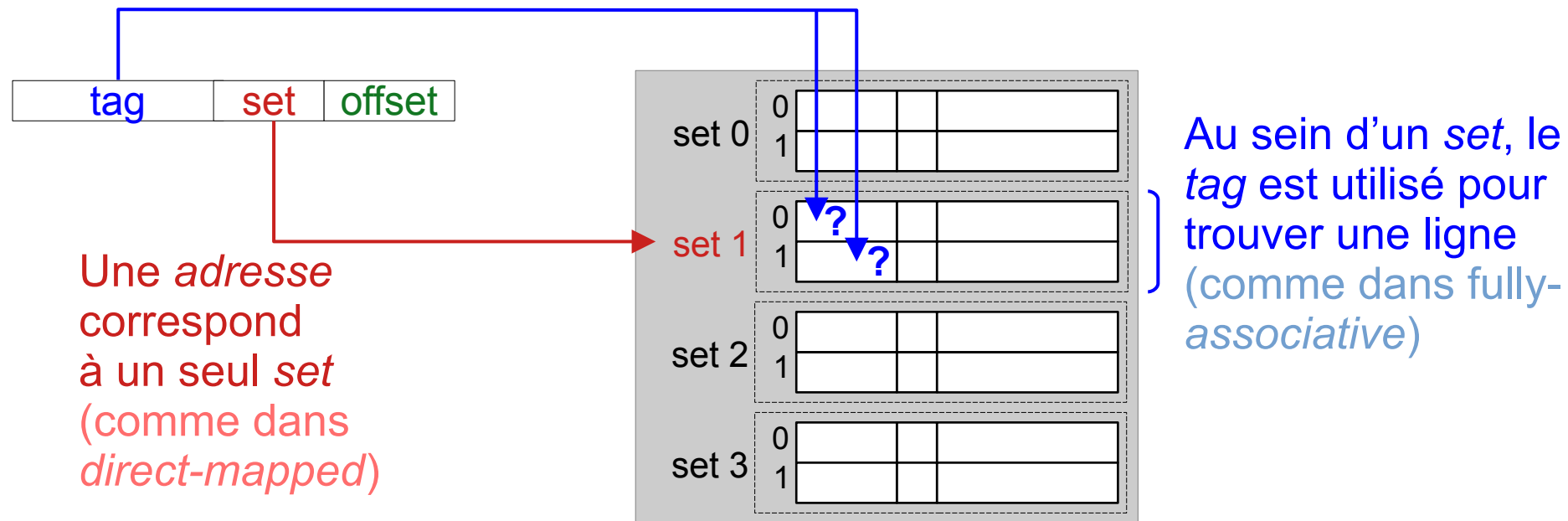
Cache direct-mapped

- Une cache *direct-mapped* ne peut stocker en même temps des blocs dont les adresses x et y sont telles que $\text{set}(x) = \text{set}(y)$ car elles devraient occuper la même ligne.
- Cette situation est appelée « *conflit de cache* » (*cache contention*) et cause des « *conflict misses* ».
- Exemple: conflit de cache
 - Supposons une cache *direct-mapped* de 512 octets, organisée en 32 lignes ($K=5$) de 16 octets ($B=4$).
 - Le processeur charge successivement les données situées aux adresses suivantes:
 - $0xA1334 \Rightarrow 1010 \ 0001 \ 0011 \ 0011 \ 0100$
 - $0xBAD37 \Rightarrow 1011 \ 1010 \ 1101 \ 0011 \ 0111$
 - Ces deux données doivent être placées dans la même ligne (19) !

Structure des caches

Cache set-associative

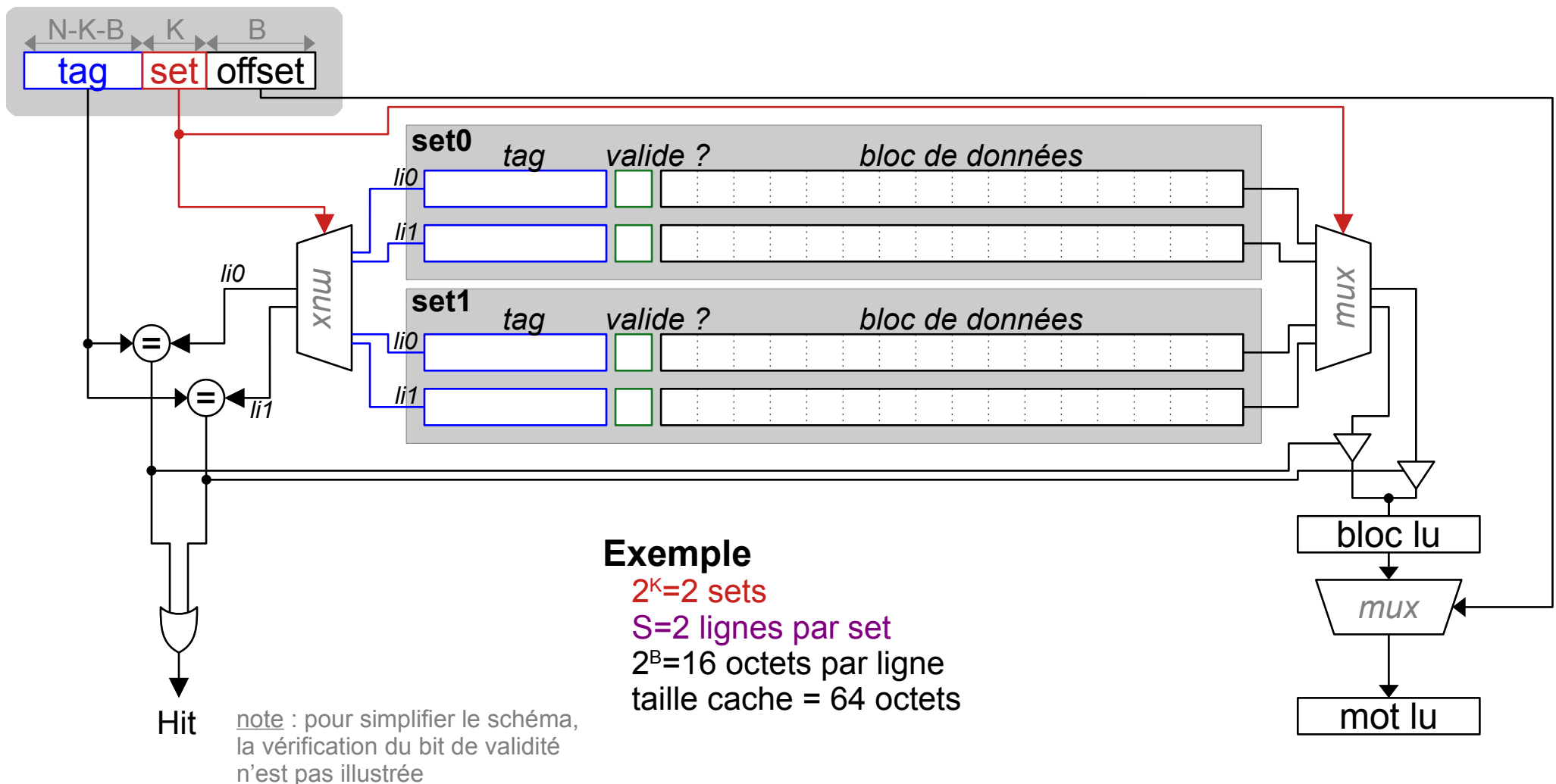
- La cache **set-associative** est une solution intermédiaire entre les caches *fully-associative* et *direct-mapped*.
- Les lignes de la cache sont groupées en 2^K ensembles (sets) comprenant chacun S lignes. Le nombre de lignes par ensemble est appelé le **degré d'associativité**. On parle de cache *S-way set-associative*.



Structure des caches

Cache set-associative

- La cache est divisée en **sets**. Une adresse désigne un set unique. Au sein de chaque **set**, les **tags** sont comparés en parallèle.



Structure des caches

Cache set-associative

- Avec une cache *set-associative*, il est possible de conserver simultanément dans la cache des données dont les adresses donnent la même valeur des *K bits Set*.
- Exemple: conflit résolu
 - Supposons une cache *2-way set-associative* de 1024 octets, organisée en 32 sets ($K=5$) de *2 lignes* de 16 octets ($B=4$).
 - Le processeur charge successivement les données situées aux adresses suivantes:
 - $0xA1334 \Rightarrow 1010 \ 0001 \ 0011 \ 0011 \ 0100$
 - $0xBAD37 \Rightarrow 1011 \ 1010 \ 1101 \ 0011 \ 0111$
 - Ces deux données peuvent cohabiter dans le même set (**19**), mais dans deux lignes différentes.

Table des Matières

Technologies RAM

- SRAM versus DRAM
- DRAM Synchrones (SDRAM)
- Latence des DRAMs

Mémoire Cache

- Localité
- Principes de fonctionnement
- Organisation
- Implémentation *fully-associative* / *direct-mapped* / *set-associative*

➡ Remplacement et ré-écriture

Mémoire Virtuelle

- Traduction d'adresses
- Table des pages
- Translation Lookaside Buffer

Remplacement

Stratégie de remplacement

- Que faire lorsque la cache est pleine ?
- Il faut remplacer une ligne déjà utilisée.
- Dans une cache direct-mapped, il n'y a pas le choix.
- Dans les mémoires *fully-associative* et *set-associative*, il est nécessaire de se poser la question suivante : **où placer une nouvelle paire (tag, donnée) lors d'un miss ?**

Exemple: cache 2-way set-associative de 32 octets

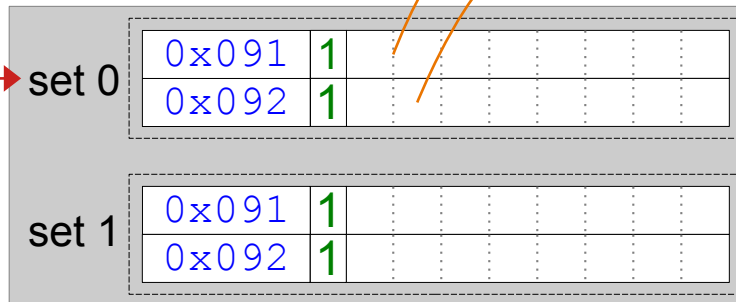
adresses en cache

0x0910	0000	1001	0001	0	000
0x0920	0000	1001	0010	0	000
0x0918	0000	1001	0001	1	000
0x0928	0000	1001	0010	1	000

nouvelle adresse

0x0930	0000	1001	0011	0	000
--------	------	------	------	---	-----

Cache



Mémoire

(...)	
0910	
0914	
0918	
091C	
0920	
0924	
0928	
092C	
0930	
0934	
(...)	

Remplacement

Stratégie de remplacement

- Un grand nombre de stratégies sont possibles. Cependant, toutes ne sont pas faciles à implémenter matériellement.
- Least Recently Used (LRU) :
 - La cache remplace la ligne qui n'a plus été utilisée depuis le plus longtemps.
 - Pour réaliser cette stratégie, il faut garder trace de l'ordre dans lequel les entrées sont accédées. Avec un degré d'associativité de 2, c'est facile (comment?). Cela devient coûteux en *hardware* pour un degré d'associativité > 4 .
- Aléatoire :
 - La cache sélectionne une ligne “au hasard”.
 - Peut être effectué efficacement et à bas coût en hardware.
- Il existe d'autres stratégies telles que *first-in-first-out* (FIFO), *round-robin*, *least-frequently-used* (LFU), ...

Stratégie de ré-écriture

- Lorsque le processeur écrit une donnée vers la mémoire, celle-ci est d'abord écrite dans la cache. Il existe deux grandes stratégies pour déterminer quand la donnée sera finalement écrite en mémoire principale.
- Write-through
 - La donnée est écrite en cache **et** simultanément en mémoire.
 - Avantage: simple à implémenter.
 - Désavantage: coût écriture en DRAM.
- Write-back
 - La donnée est seulement écrite en cache. La donnée n'est écrite en mémoire que lorsque la ligne de cache est éjectée de la cache (i.e. lors d'un remplacement).
 - Avantage: plusieurs écritures dans une ligne de cache ne nécessitent pas plusieurs écritures en mémoire. La ré-écriture d'une ligne de cache complète est plus efficace.

Ré-écriture

Stratégie de ré-écriture

- Pour implémenter la stratégie de ré-écriture « write-back », chaque ligne de cache est munie d'un bit supplémentaire « **dirty** ».
- Lorsque ce bit vaut 1, cela indique que le bloc concerné diffère en cache et en mémoire. Il faudra le ré-écrire en mémoire lorsqu'il sera éjecté.

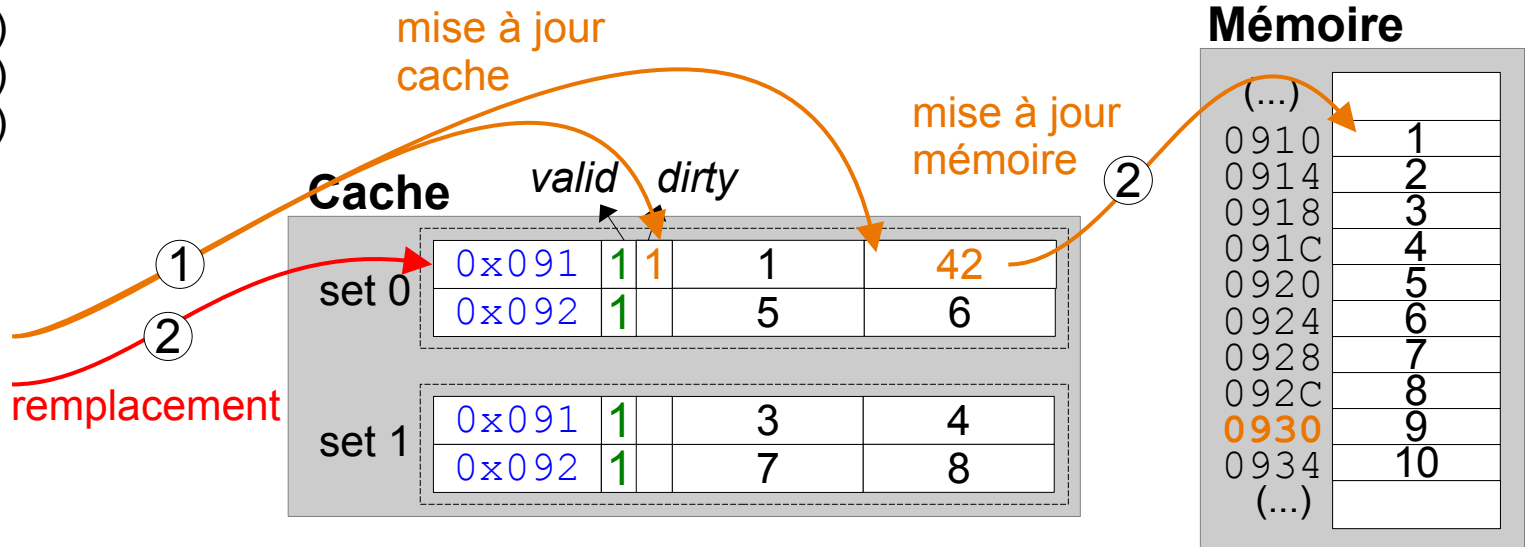
Exemple: cache 2-way set-associative de 32 octets

adresses en cache

0x0910 (set 0, ligne 0)
0x0920 (set 0, ligne 1)
0x0918 (set 1, ligne 0)
0x0928 (set 1, ligne 1)

instructions

```
li a0, 42  
sw a0, 0x0914  
lw a1, 0x0930
```



Structure des caches

Résumé

- Différents paramètres régissent l'organisation d'une mémoire cache. Il est important de bien comprendre comment ils s'articulent entre eux.

Paramètres:

Taille de la cache (octets) **C**

Taille d'une adresse mémoire (bits) **N**
(dépend du bus mémoire)

Taille des blocs (octets) 2^B
(dépend de la DRAM)

Degré d'associativité **S**

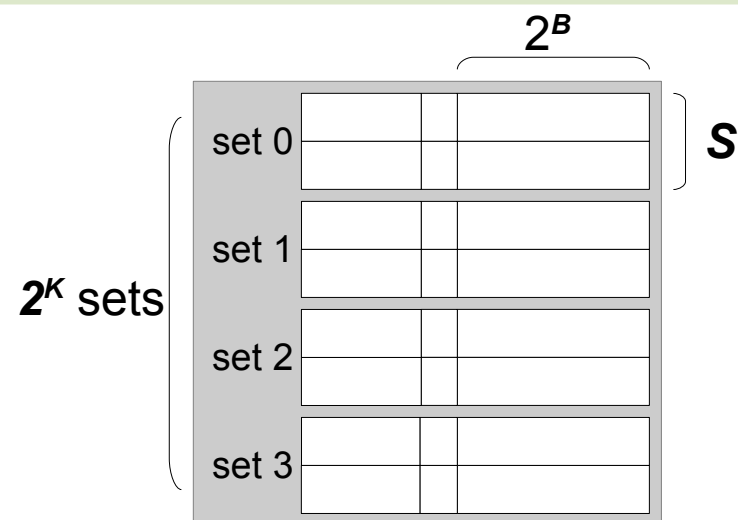
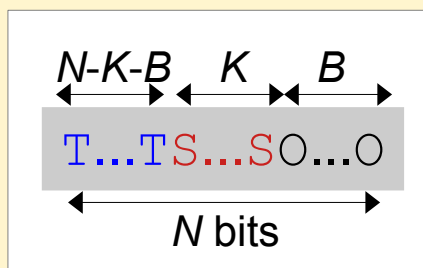
Paramètres dérivés:

Nombre total de lignes = $C / 2^B$

Nombre de sets $2^K = C / (S \times 2^B)$

Taille des tags (bits) = $N-K-B$

Format d'une adresse : (tag, set, offset)



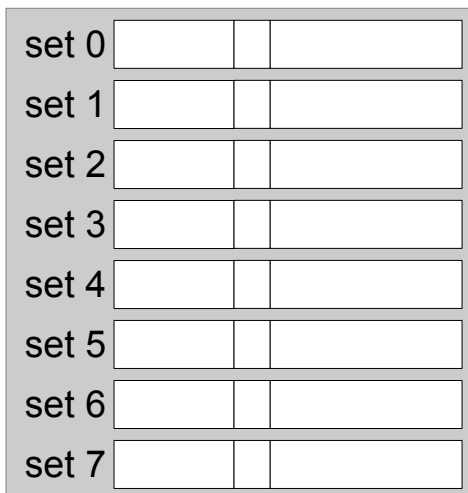
Taille totale cache incluant tag et bit validité = $2^K \times S \times (N-K-B + 1 + 2^{(B+3)})$, exprimée en bits

Structure des caches

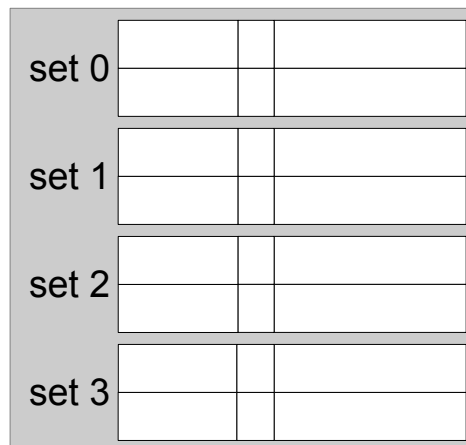
Résumé

- Une cache de taille donnée peut être organisée de différentes façons, en fonction du **degré d'associativité S**.
- Prenons l'exemple d'une cache de taille **C=128 octets** dans laquelle chaque ligne contient un **bloc de 16 octets (B=4)**; le nombre de lignes est donc 8

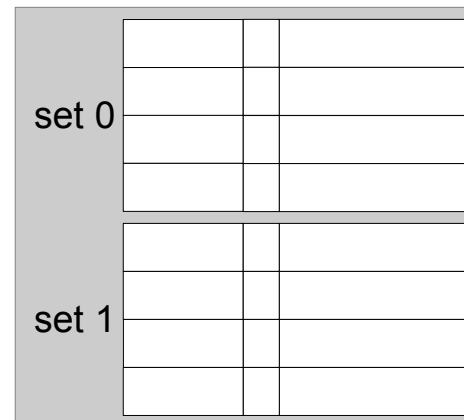
direct-mapped
(1-way set-associative)



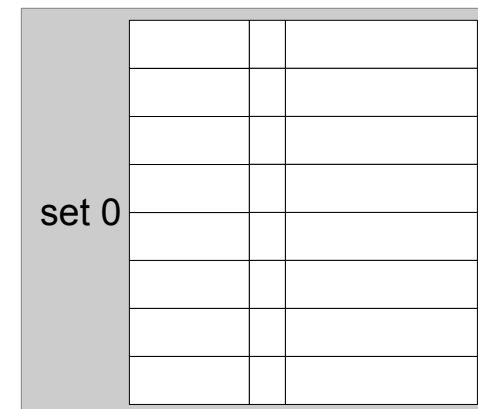
2-way set-associative



4-way set-associative



fully-associative
(8-way set-associative)



degré d'associativité croissant

Structure des caches

Exercice – Simulation de caches

- Considérez les caches suivantes
 - **A** : cache *4-way set-associative* 128 octets, 16 lignes de 8 octets
 - **B** : cache *direct-mapped* 128 octets, 16 lignes de 8 octets
 - la stratégie de remplacement est LRU
- Le processeur effectue des lectures aux adresses suivantes et dans cet ordre :
 - 0xA3C9, 0xA3CB, 0xB5EA, 0xB5E1,
0xB7E5, 0xB9C9, 0xA3C9, 0xB5E1,
0xB5EA, 0xB5E1, 0x12AA, 0x122A,
0xA3C8, 0xA3CE
- Déterminez le *hit ratio* pour chaque cache.

Structure des caches

Solution

- Cache A : 4-way set-associative

Hit ratio = 7/14 = **50 %**

0xA3C9	tag=0x51E	set=1	offset=1	MISS	line=0
0xA3CB	tag=0x51E	set=1	offset=3	HIT	line=0
0xB5EA	tag=0x5AF	set=1	offset=2	MISS	line=1
0xB5E1	tag=0x5AF	set=0	offset=1	MISS	line=0
0xB7E5	tag=0x5BF	set=0	offset=5	MISS	line=1
0xB9C9	tag=0x5CE	set=1	offset=1	MISS	line=2
0xA3C9	tag=0x51E	set=1	offset=1	HIT	line=0
0xB5E1	tag=0x5AF	set=0	offset=1	HIT	line=0
0xB5EA	tag=0x5AF	set=1	offset=2	HIT	line=1
0xB5E1	tag=0x5AF	set=0	offset=1	HIT	line=0
0x12AA	tag=0x095	set=1	offset=2	MISS	line=3
0x122A	tag=0x091	set=1	offset=2	MISS	line=2 REPLACE (tag=0x5CE)
0xA3C8	tag=0x51E	set=1	offset=0	HIT	line=0
0xA3CE	tag=0x51E	set=1	offset=6	HIT	line=0

Structure des caches

Solution

- Cache B : direct-mapped

Hit ratio = $5/14 = 35,7\%$

0xA3C9	tag=0x147	set=9	offset=1	MISS	line=0	
0xA3CB	tag=0x147	set=9	offset=3	HIT	line=0	
0xB5EA	tag=0x16B	set=D	offset=2	MISS	line=0	
0xB5E1	tag=0x16B	set=C	offset=1	MISS	line=0	
0xB7E5	tag=0x16F	set=C	offset=5	MISS	line=0	REPLACE (tag=0x16B)
0xB9C9	tag=0x173	set=9	offset=1	MISS	line=0	REPLACE (tag=0x147)
0xA3C9	tag=0x147	set=9	offset=1	MISS	line=0	REPLACE (tag=0x173)
0xB5E1	tag=0x16B	set=C	offset=1	MISS	line=0	REPLACE (tag=0x16F)
0xB5EA	tag=0x16B	set=D	offset=2	HIT	line=0	
0xB5E1	tag=0x16B	set=C	offset=1	HIT	line=0	
0x12AA	tag=0x025	set=5	offset=2	MISS	line=0	
0x122A	tag=0x024	set=5	offset=2	MISS	line=0	REPLACE (tag=0x025)
0xA3C8	tag=0x147	set=9	offset=0	HIT	line=0	
0xA3CE	tag=0x147	set=9	offset=6	HIT	line=0	

Table des Matières

Technologies RAM

- SRAM versus DRAM
- DRAM Synchrones (SDRAM)
- Latence des DRAMs

Mémoire Cache

- Localité
- Principes de fonctionnement
- Organisation
- Implémentation *fully-associative* / *direct-mapped* / *set-associative*
- Remplacement et ré-écriture



Mémoire Virtuelle

- Traduction d'adresses
- Table des pages
- Translation Lookaside Buffer

Mémoire Virtuelle

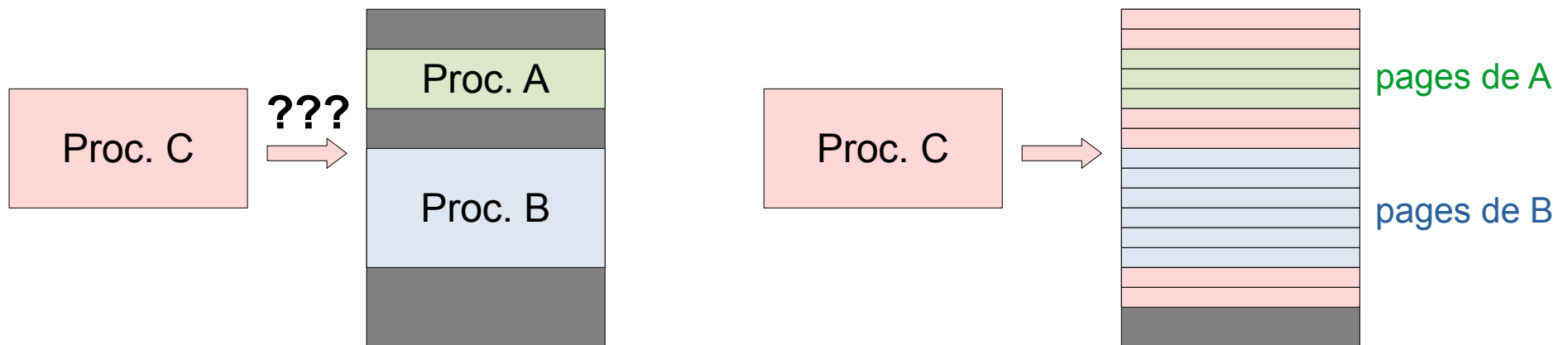
Objectifs

- Mémoire de grande taille, à faible latence et coût
 - Donner l'illusion d'une mémoire de grande taille
 - Seule une partie de cette mémoire est présente à un moment donné en mémoire physique
 - Mémoire physique agit comme "cache" pour la mémoire secondaire
- Protection des processus
 - Plusieurs programmes / processus sont exécutés "au même moment" par le processeur et résident donc tous en mémoire.
 - Protéger les programmes les uns des autres : un programme n'a pas accès à l'espace mémoire d'un autre (sauf si permission).

Mémoire Virtuelle

Pagination (*Paging*)

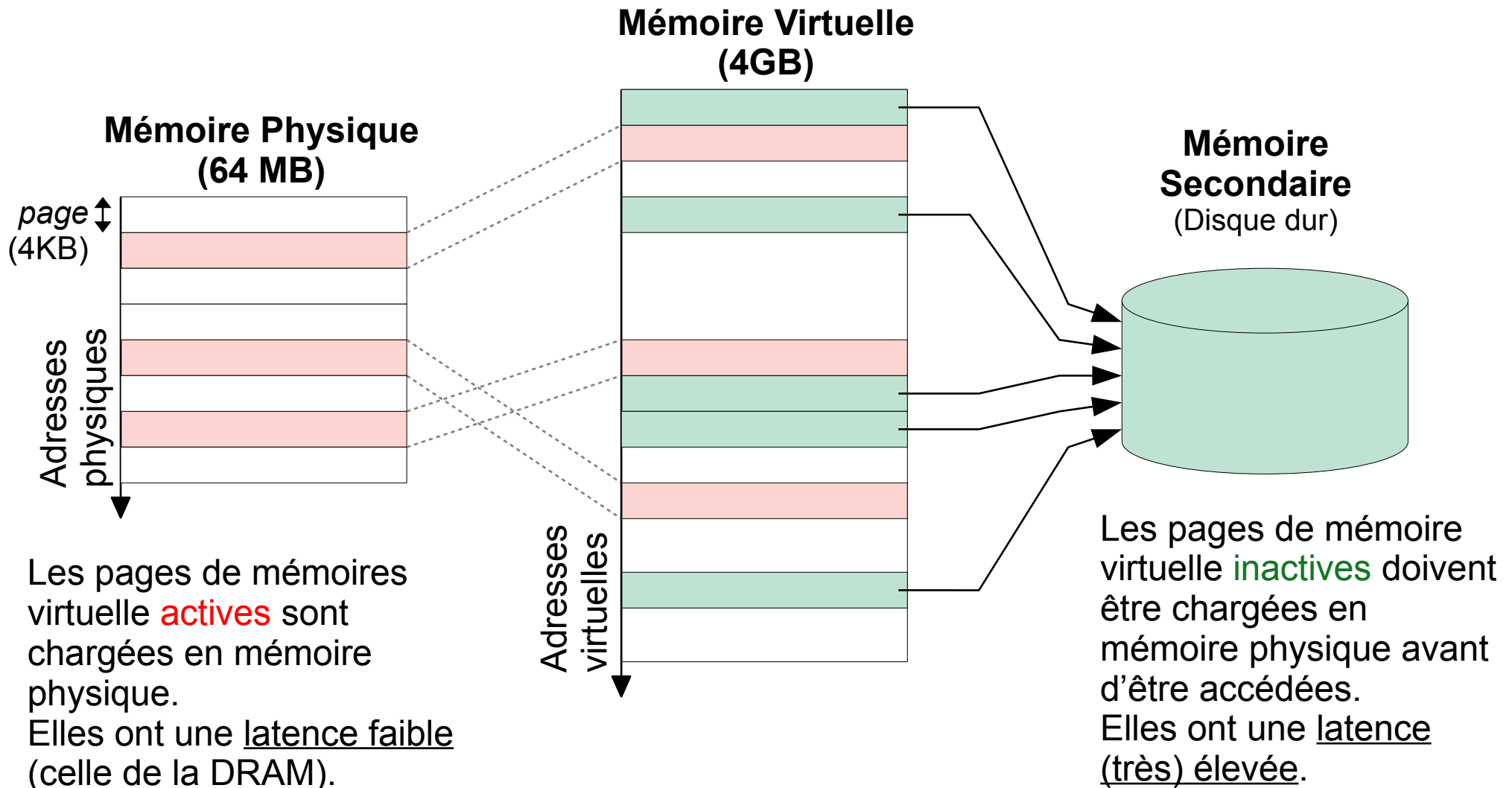
- Différents programmes (processus) peuvent cohabiter en mémoire sur un même ordinateur. Ces programmes ont des tailles différentes.
- Si la mémoire est allouée de manière monolithique, c.-à-d. que l'entièreté de la mémoire nécessaire à un programme est allouée en un seul bloc, des situations de fragmentation de la mémoire peuvent mener à l'impossibilité de charger un programme en mémoire.



- Pour pallier ce problème, la mémoire est découpée en petits blocs de même taille appelés **pages**. Chaque programme reçoit les pages qui lui sont nécessaires.

Mémoire Virtuelle

Architecture générale



Mémoire Virtuelle

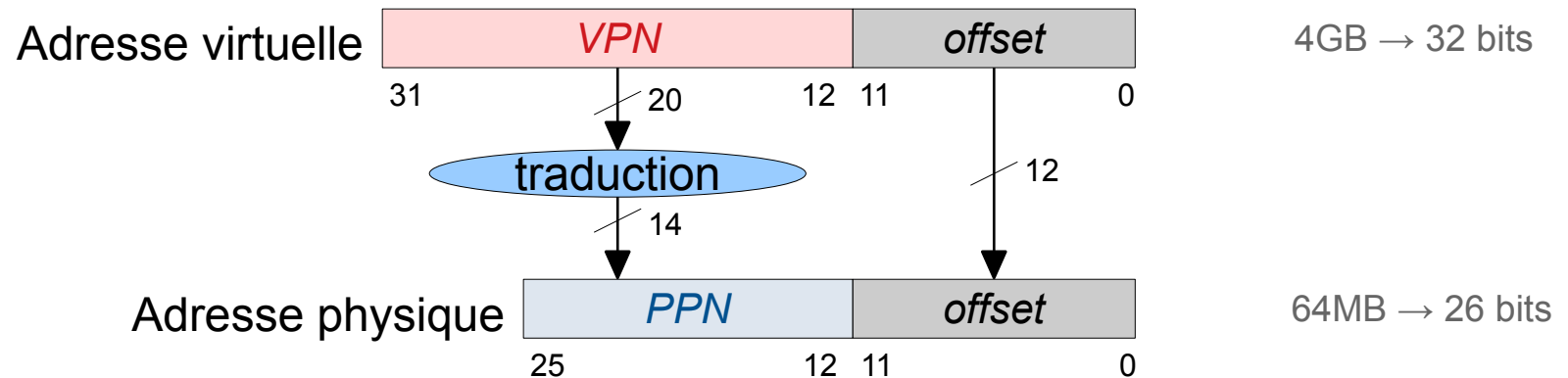
Pages

- La mémoire est découpée en **pages** de même taille. La taille typique d'une page est **4 KB**.
 - La mémoire physique contient des **pages physiques**. Chaque page physique a une **adresse physique**.
 - La mémoire virtuelle contient des **pages virtuelles**. Chaque page virtuelle a une **adresse virtuelle**.
- Il y a plus de pages virtuelles que de pages physiques. Les adresses virtuelles sont donc plus longues que les adresses physiques.
- Une page virtuelle peut être dans deux états différents
 - **active** : chargée en mémoire physique – il y a alors correspondance entre cette page virtuelle et une page physique
 - **inactive** : réside en mémoire secondaire (p.ex. disque dur ou SSD)

Mémoire Virtuelle

Traduction d'adresses

- La correspondance entre une adresse virtuelle et une adresse physique est appelée **traduction d'adresse**.



- Une adresse virtuelle est composée d'un **numéro de page virtuel (VPN)** et d'un offset dans la page
- Une adresse physique est composée d'un **numéro de page physique (PPN)** et d'un offset dans la page
- La traduction ne porte que sur les **VPN** → **PPN**. Le déplacement (*offset*) à l'intérieur d'une page n'est pas affecté par la traduction.

VPN : Virtual Page Number ; PPN : Physical Page Number

Mémoire Virtuelle

Traduction d'adresses

- Une page virtuelle doit pouvoir être chargée à n'importe quel endroit en mémoire physique. Par conséquent un **VPN** doit pouvoir être traduit en n'importe quel **PPN**.
- La correspondance **VPN** → **PPN** est conservée dans une table appelée **Table des Pages**.
 - Contient une entrée par **VPN**; Le **VPN** sert d'index
 - Chaque entrée comprend *1 bit de validité* et un **PPN**
 - Stockée en mémoire physique⁽¹⁾. Son emplacement est typiquement désigné par un registre spécial (*page table register*).

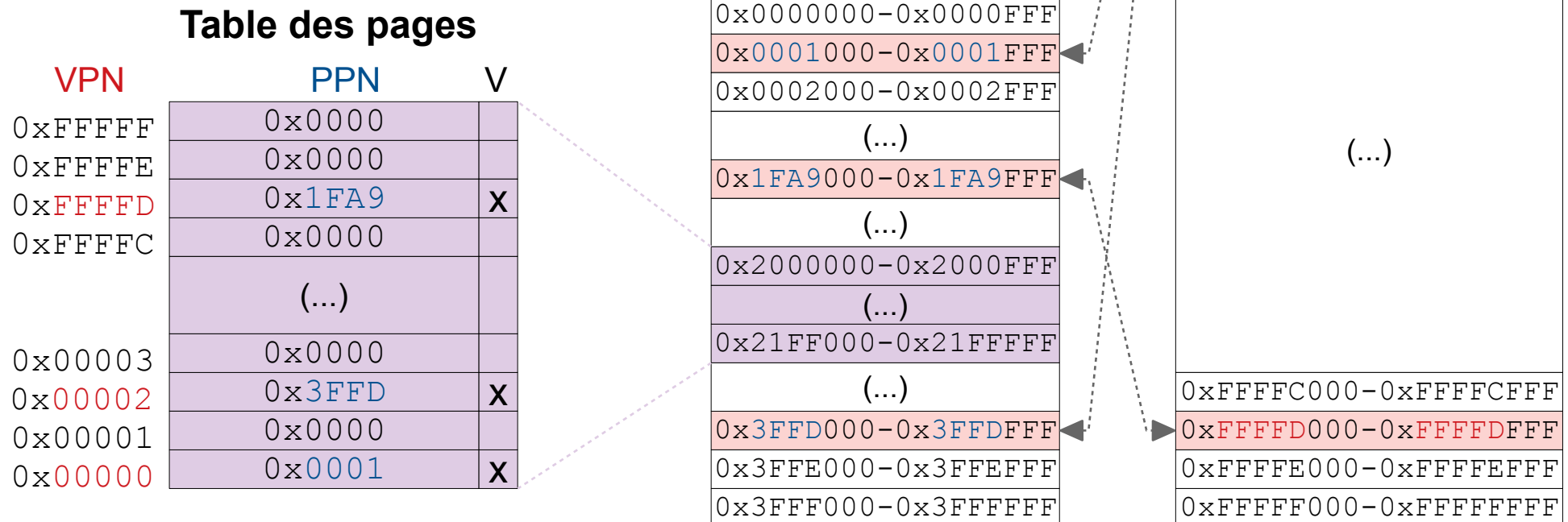
VPN	PPN	V
0xFFFFF	0x0000	
0xFFFFE	0x0000	
	(...)	
0x00001	0x0000	
0x00000	0x0001	X

Note (1) : utiliser une mémoire spéciale pour stocker la table des pages n'est pas envisageable car le nombre de pages virtuelles est très grand (p.ex. 4GB ⇒ ~1M pages de 4KB)

Mémoire Virtuelle

Table des Pages – Exemple

- Supposons
 - La mémoire physique contient **64 MB** (2^{26} octets)
 - La mémoire virtuelle fait **4 GB** (2^{32} octets)
 - Chaque page fait **4 KB** (2^{12} octets)



Mémoire Virtuelle

Table des Pages – Exemple

- Supposons
 - La mémoire physique contient **64 MB** (2^{26} octets)
 - La mémoire virtuelle fait **4 GB** (2^{32} octets)
 - Chaque page fait **4 KB** (2^{12} octets)
- Donc,
 - Il y a $4 \text{ GB} / 4 \text{ KB} = \mathbf{1 \text{ M}}$ de pages virtuelles (2^{20})
 - Une adresse physique fait **26 bits** (14 pour le PPN, 12 pour l'offset).
 - Une adresse virtuelle fait **32 bits** (20 pour le VPN, 12 pour l'offset).
 - Chaque entrée de la table des pages contient : 1 bit de validité + 14 bits de PPN. Pour simplifier, on considère 2 octets par entrée.
 - Il y a **2^{20} entrées** (= nombre de pages virtuelles)
 - La taille totale de la table des pages est de $2 \text{ octets} * 2^{20} = \mathbf{2 \text{ MB}}$, ce qui occupe $2 \text{ MB} / 4 \text{ KB} = \mathbf{512 \text{ pages}}$.

Mémoire Virtuelle

Traduction d'adresses

- Pour chaque accès mémoire, le processus de traduction est le suivant

- 1) Le processeur extrait le **VPN** de l'**adresse virtuelle**
- valide 2) Il consulte la **table de pages** pour obtenir le **PPN**, en utilisant le **VPN** comme index (nécessite un accès mémoire)
- 3) Si l'entrée est **invalide**, une exception **Page Fault** est déclenchée.
 - Le gestionnaire associé charge la page virtuelle depuis la mémoire secondaire (p.ex. disque dur).
 - La page des tables est mise à jour.
- 4) Il complète le **PPN** avec l'*offset* pour obtenir l'**adresse physique**
- 5) La lecture ou l'écriture à l'adresse physique peut alors avoir lieu

Mémoire Virtuelle

Translation Lookaside Buffer (TLB)

- Avec le processus décrit précédemment, chaque lecture/écriture en mémoire **nécessite deux accès** :
 - consultation de la table des pages
 - lecture/écriture en mémoire physique
- Afin de supprimer cette surcharge, les entrées récemment consultées de la page des tables sont conservée dans une mémoire cache matérielle appelée **Translation Lookaside Buffer** (TLB).
 - Il s'agit d'une cache *fully-associative*. Elle possède un nombre limité d'entrées (vu son coût) : typiquement 16 à 512 entrées.
 - Chaque entrée de la TLB possède
 - un **VPN**
 - un **PPN**
 - un **bit de validité**

VPN	PPN	V
0x00002	0x3FFD	x
0x00000	0x0001	x
0xFFFFD	0x1FA9	x
0x00000	0x0000	

Mémoire Virtuelle

Traduction d'adresses avec TLB

- Pour chaque accès mémoire, le processus de traduction est le suivant

1) Le processeur extrait le **VPN** de l'**adresse virtuelle**

2) Il consulte le **TLB**

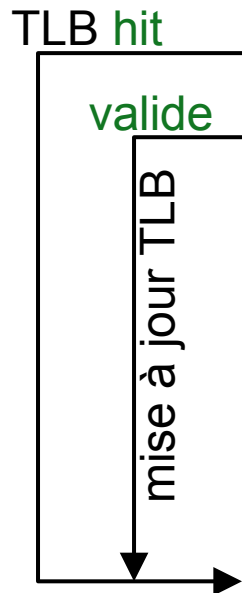
3) En cas de TLB **miss**, il consulte la **table de pages** pour obtenir le PPN, en utilisant le **VPN** comme index.

4) Si l'entrée est **invalide**, une exception *Page Fault* est déclenchée.

- Le gestionnaire associé charge la page virtuelle depuis la mémoire secondaire (p.ex. disque dur).
- La table des pages et le TLB sont mis à jour.

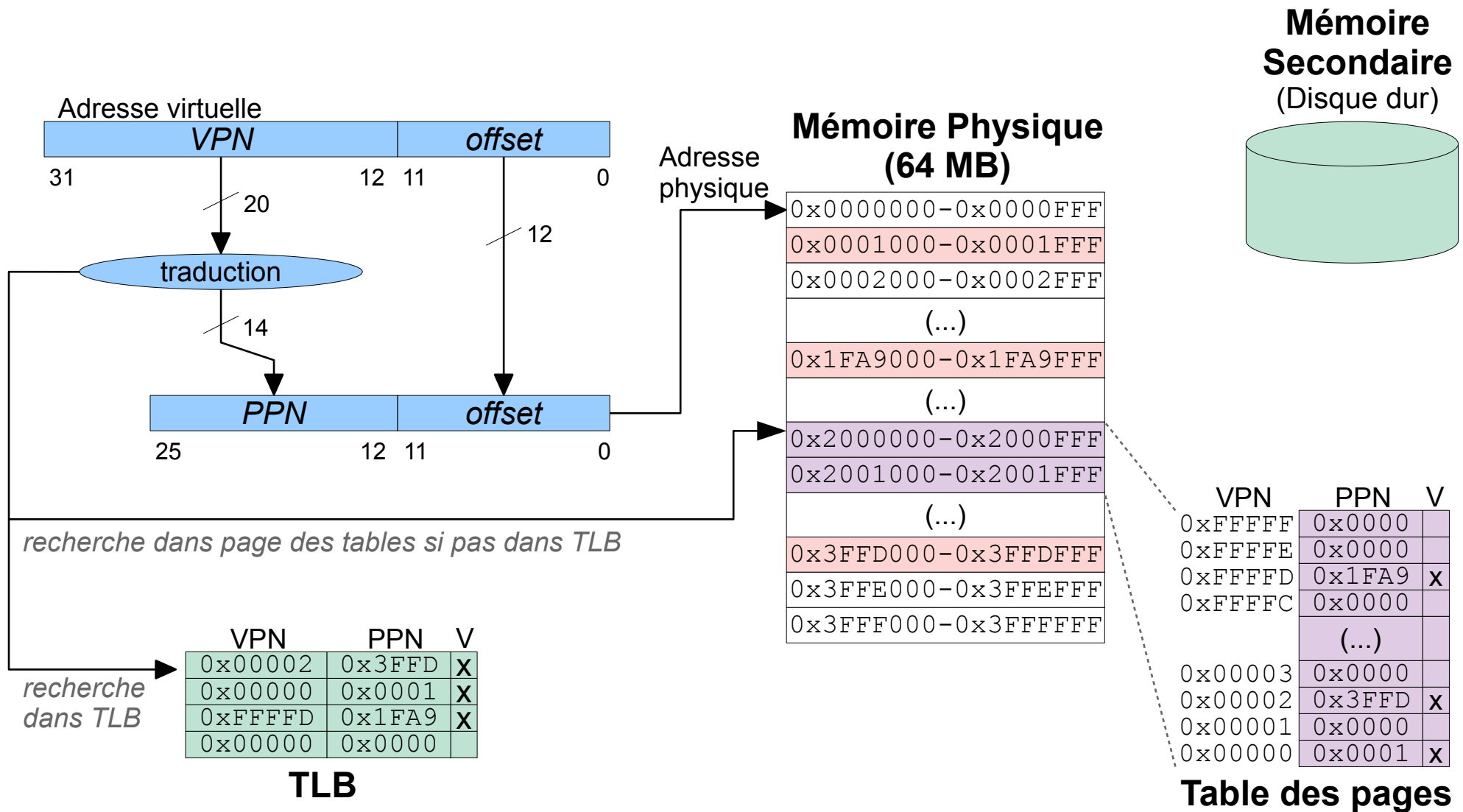
5) Il complète le **PPN** avec l'*offset* pour obtenir l'**adresse physique**

6) La lecture ou l'écriture à l'adresse physique peut alors avoir lieu



Mémoire Virtuelle

Translation Lookaside Buffer

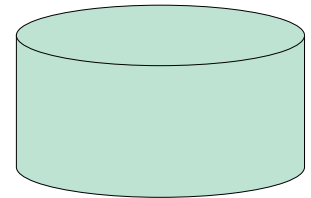


Mémoire Virtuelle

Exemple 1 – le plus favorable

- Accès à l'adresse virtuelle 0x00002A9B

**Mémoire
Secondaire**
(Disque dur)



**Mémoire Physique
(64 MB)**

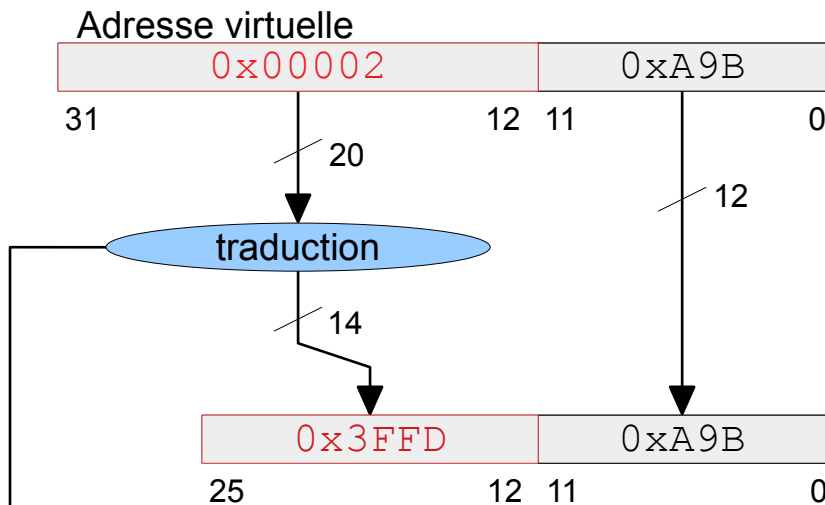
0x0000000-0x0000FFF
0x0001000-0x0001FFF
0x0002000-0x0002FFF
(...)
0x1FA9000-0x1FA9FFF
(...)
0x2000000-0x2000FFF
0x2001000-0x2001FFF
(...)
0x3FFD000-0x3FFDFFF
0x3FFE000-0x3FFEFFF
0x3FFF000-0x3FFFFF

Adresse
physique

(2)
accès

VPN	PPN	V
0xFFFFF	0x0000	
0xFFFFE	0x0000	
0xFFFFD	0x1FA9	x
0xFFFFC	0x0000	
(...)		
0x00003	0x0000	
0x00002	0x3FFD	x
0x00001	0x0000	
0x00000	0x0001	x

Table des pages



(1)

**VPN en
cache ?**

0x00002

HIT :-)

VPN	PPN	V
0x00002	0x3FFD	x
0x00000	0x0001	x
0x00000	0x0000	
0x00000	0x0000	

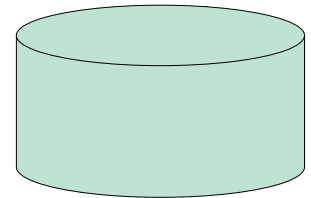
TLB

Mémoire Virtuelle

Exemple 2 – TLB miss

- Accès à l'adresse virtuelle 0xFFFFD254

**Mémoire
Secondaire**
(Disque dur)



**Mémoire Physique
(64 MB)**

Adresse physique	0x0000000-0x0000FFF
	0x0001000-0x0001FFF
	0x0002000-0x0002FFF
	(...)
(3) accès	0x1FA9000-0x1FA9FFF
	(...)
	0x2000000-0x2000FFF
	0x2001000-0x2001FFF
	(...)
	0x3FFD000-0x3FFDFFF
	0x3FFE000-0x3FFEFFF
	0x3FFF000-0x3FFFFF

(2b)

page active :-)

VPN	PPN	V
0xFFFFF	0x0000	
0xFFFFE	0x0000	
0xFFFFD	0x1FA9	x
0xFFFFC	0x0000	
	(...)	
0x00003	0x0000	
0x00002	0x3FFD	x
0x00001	0x0000	
0x00000	0x0001	x

Table des pages

(2a) *lecture*
page des tables
(accès mémoire)

mise à jour TLB

(2c)

TLB

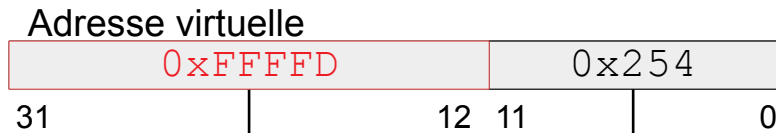
VPN	PPN	V
0x00002	0x3FFD	x
0x00000	0x0001	x
0x00000	0x0000	
0x00000	0x0000	

(1)

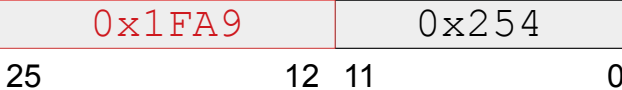
*VPN en
cache ?*

0xFFFFD

MISS :-)



traduction



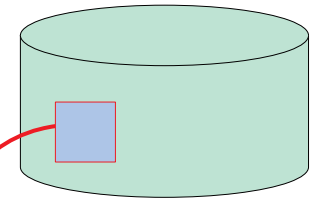
Mémoire Virtuelle

Exemple 3 – Page Fault

- Accès à l'adresse virtuelle 0x00003BEC

(3a) *gestionnaire exception :
chargement de la page
à partir de la
mémoire secondaire*

**Mémoire
Secondaire**
(Disque dur)



**Mémoire Physique
(64 MB)**

0x0000000-0x0000FFF
0x0001000-0x0001FFF
0x0002000-0x0002FFF
(...)
0x1FA9000-0x1FA9FFF
(...)
0x2000000-0x2000FFF
0x2001000-0x2001FFF
(...)
0x3FFD000-0x3FFDFFF
0x3FFE000-0x3FFEFFF
0x3FFF000-0x3FFFFF

(2b) *page inactive :-(
PAGE FAULT
exception*

VPN	PPN	V
0xFFFFF	0x0000	
0xFFFFE	0x0000	
0xFFFFD	0x1FA9	x
0xFFFFC	0x0000	
(...)		
0x00003	----	
0x00002	0x3FFD	x
0x00001	0x0000	
0x00000	0x0001	x

Table des pages

Adresse
physique
(4)
accès

(2a) *lecture
page des tables
(accès mémoire)*

(1) *VPN en
cache ?
0x00003
MISS :-(
mise à jour TLB*

VPN	PPN	V
0x00002	0x3FFD	x
0x00000	0x0001	x
0x00000	0x0000	
0x00000	0x0000	

TLB

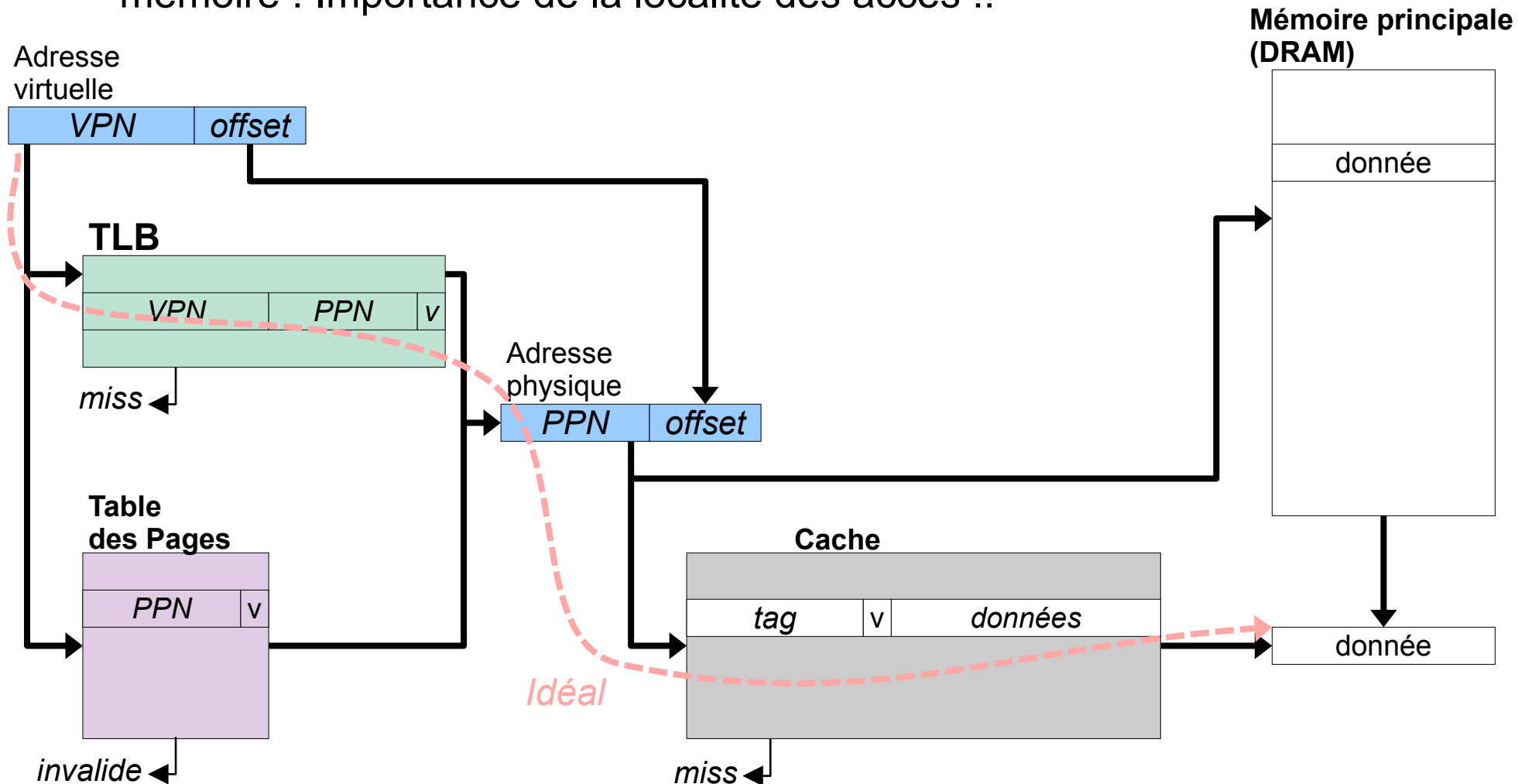
mise à jour TLB

(3b) *mise à jour
table des pages*

Hiérarchie de Mémoire

Conclusion

- Complexité de l'implémentation matérielle nécessaire à un seul accès mémoire ! Importance de la localité des accès !!



Références

Computer Organization and Design, 4th edition, D. Patterson and J. Hennessy, Morgan-Kaufman, 2009

CODE: The Hidden Language of Computer Hardware and Software, C. Petzold, Microsoft Press, 1999

A Practical Introduction to Computer Architecture, D. Page, Springer-Verlag, 2009

What Every Programmer Should Know About Memory, Ulrich Drepper, Red Hat Inc., 2007

Computer Organization and Architecture – Designing for Performance, 8th edition, W. Stallings, Pearson, 2010.

Inside the Machine – An Illustrated Introduction to Microprocessors and Computer Architecture, J. Stokes, Ars Technica Library, No Starch Press, 2006

Memory Systems: Cache, DRAM, Disks, Bruce Jacob, Spencer W. Ng, and David T. Wang. Morgan Kaufmann Publishers, September 2007.

Datasheet of Synchronous DRAM MT48LCxxx, Micron Technology, 2000