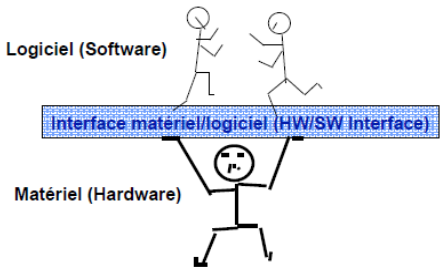
Master 2 : Systèmes embarqués et Mobilité

Matière : Concepts avancés d'architecture

Cours 2.3 : Caches

Année 2020-2021

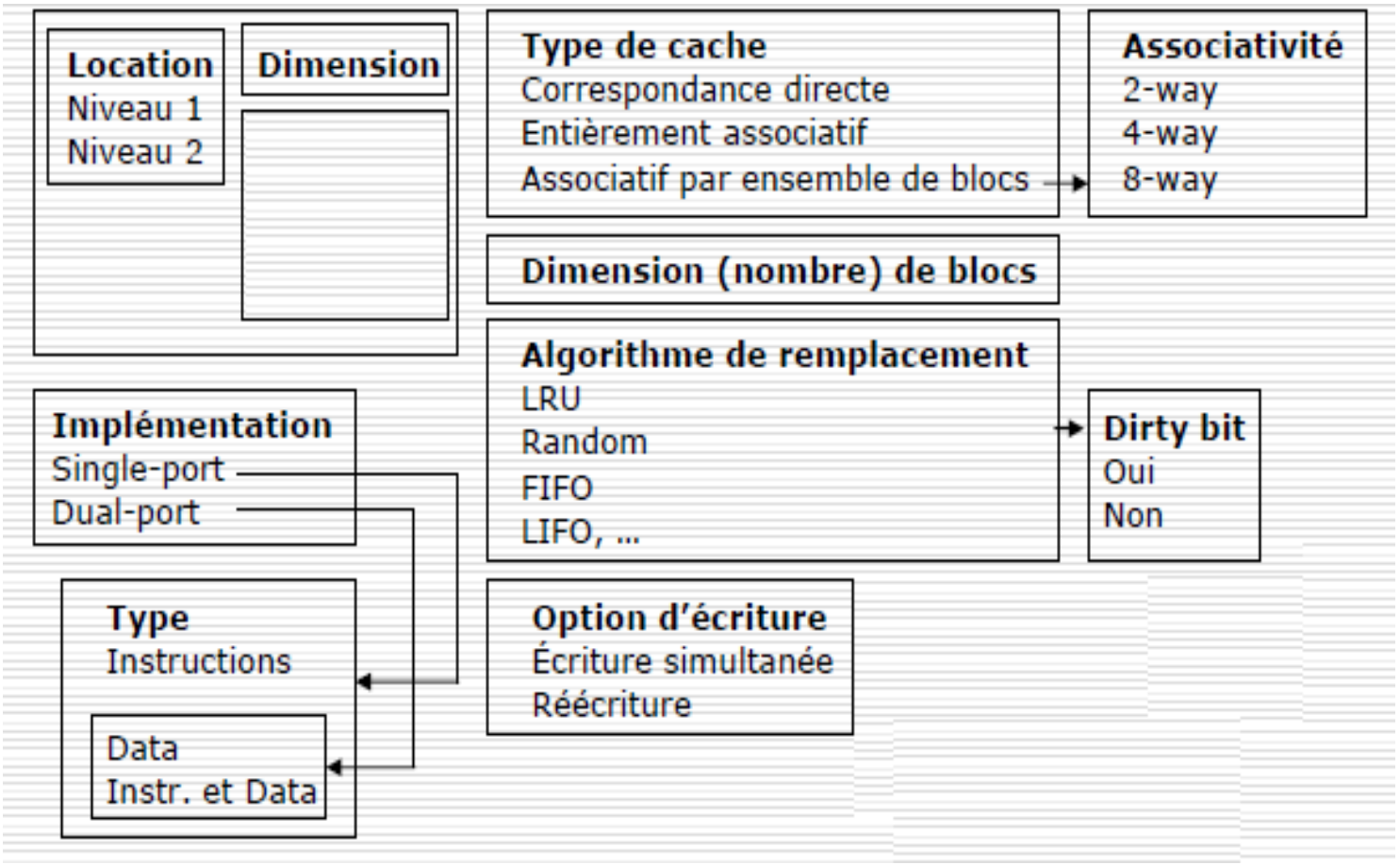
Pr R. BOUDOUR



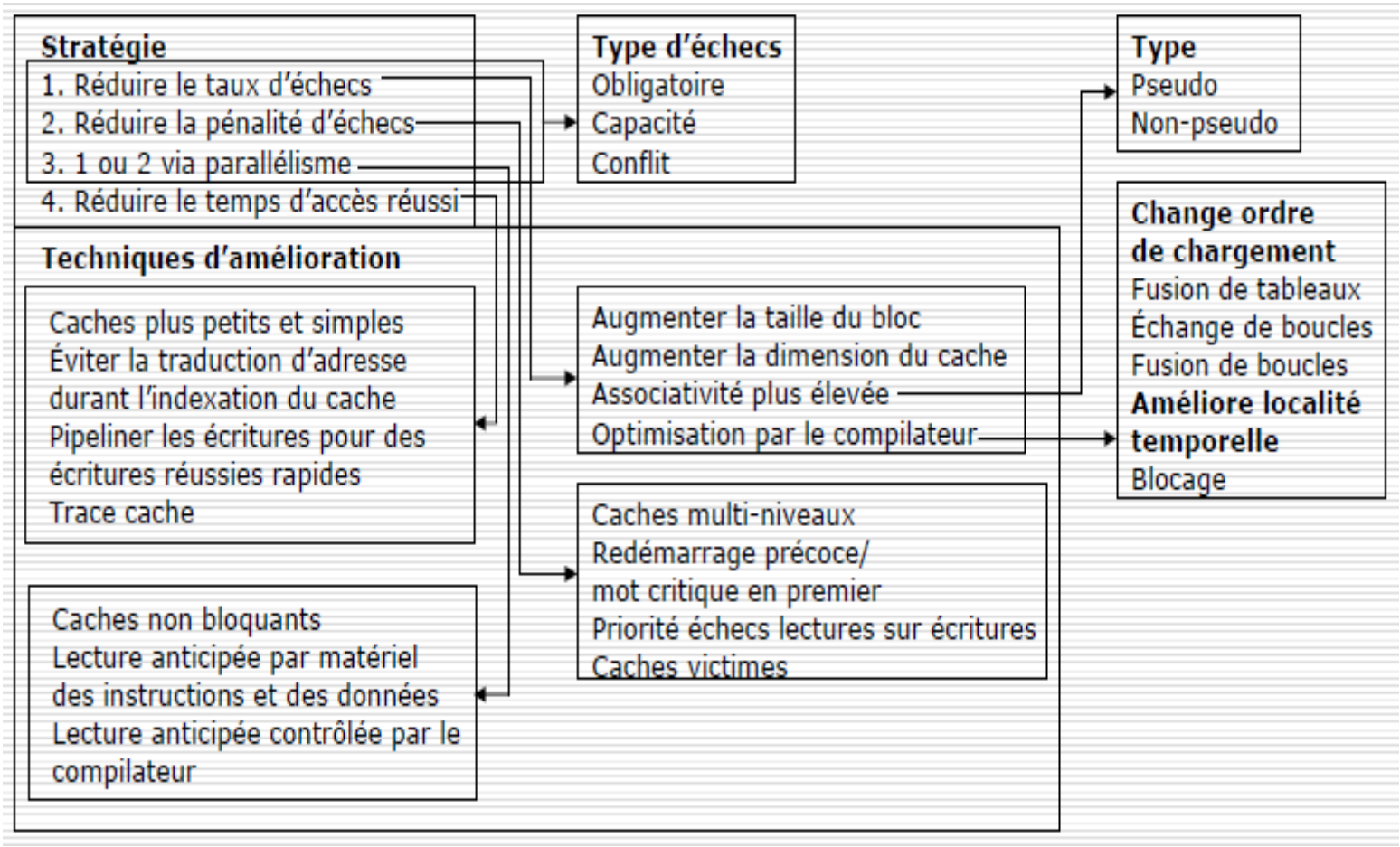
TP2 : Simulateurs de cache

- **Mise en oeuvre de ces simulateurs et leurs critiques**
  - http://williamstallings.com/COA/Animation/Links.html
  - **http://www.ecs.umass.edu/ece/koren/architecture/Cache/frame0.htm**
  - **http://cairn.enssat.fr/enseignements/Archi/SimuleCache/**
  - **gem5.org** The gem5 Simulator :
    A modular platform for computer-system architecture research
    http://learning.gem5.org/tutorial/index.html

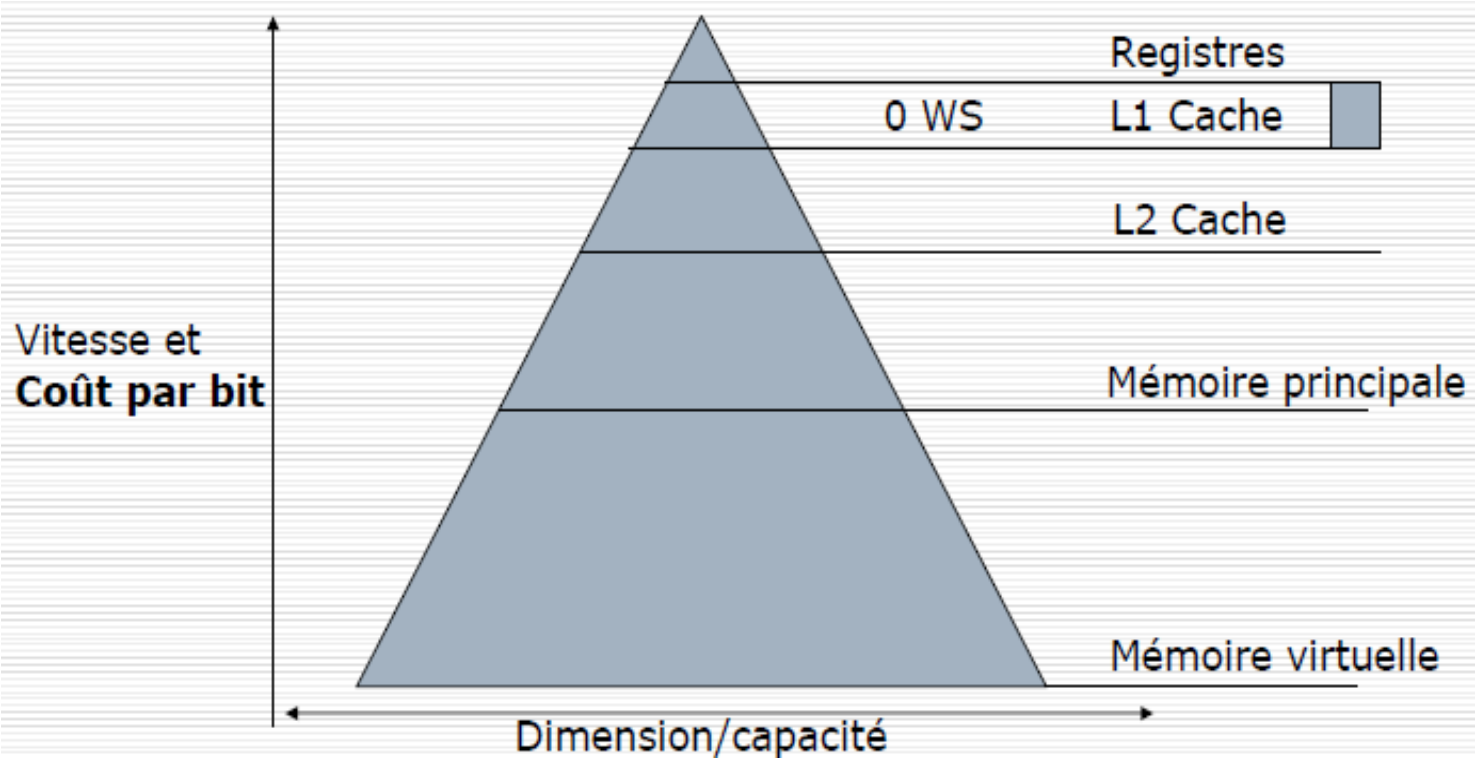
Cache


Location
Niveau 1
Niveau 2
Dimension
Type de cache
Correspondance directe
Entièrement associatif
Associatif par ensemble de blocs
Associativité
2-way
4-way
8-way
Dimension (nombre) de blocs
Algorithme de remplacement
LRU
Random
FIFO
LIFO, ...
Implémentation
Single-port
Dual-port
Dirty bit
Oui
Non
Type
Instructions
Data
Instr. et Data
Option d'écriture
Écriture simultanée
Réécriture

Techniques d'amélioration

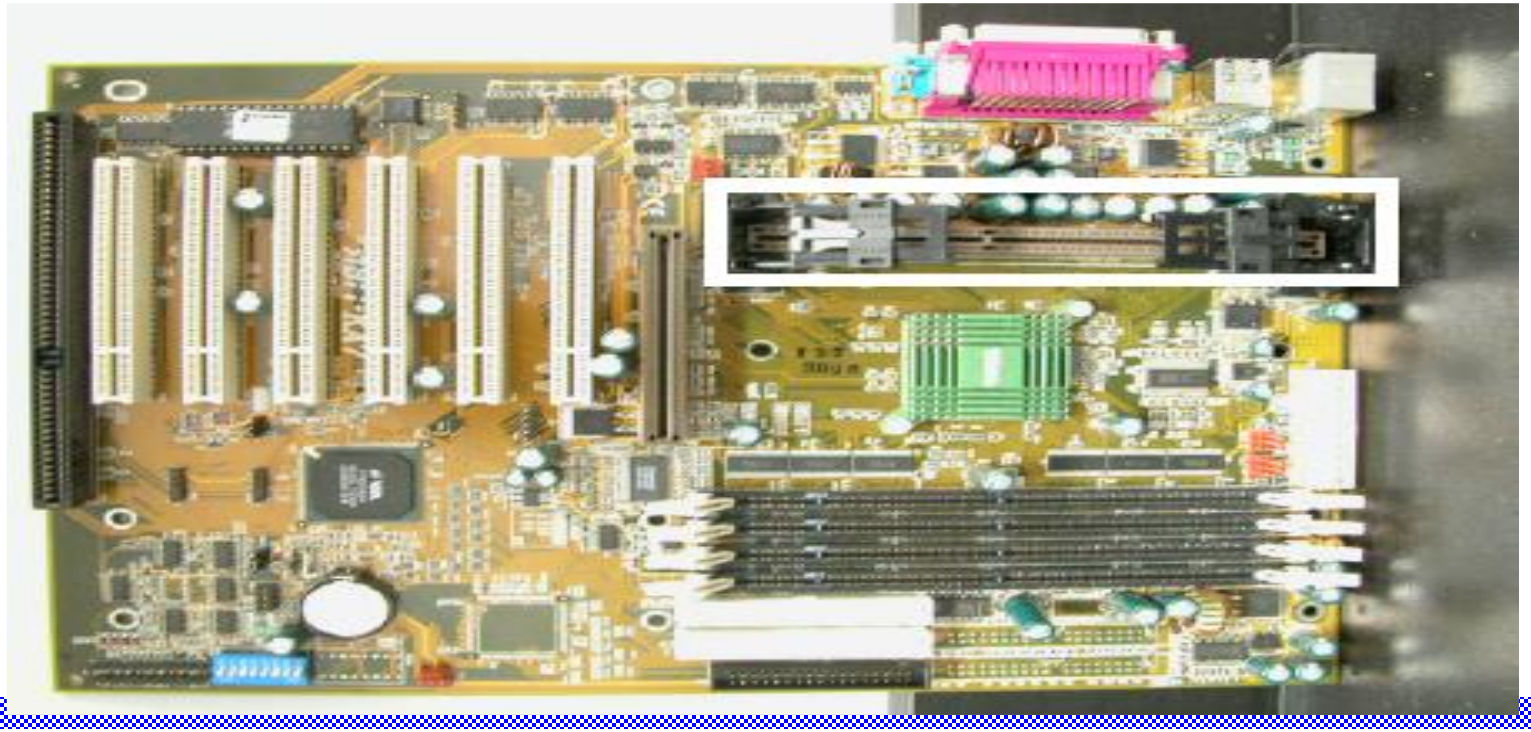


Concept de hiérarchie mémoire



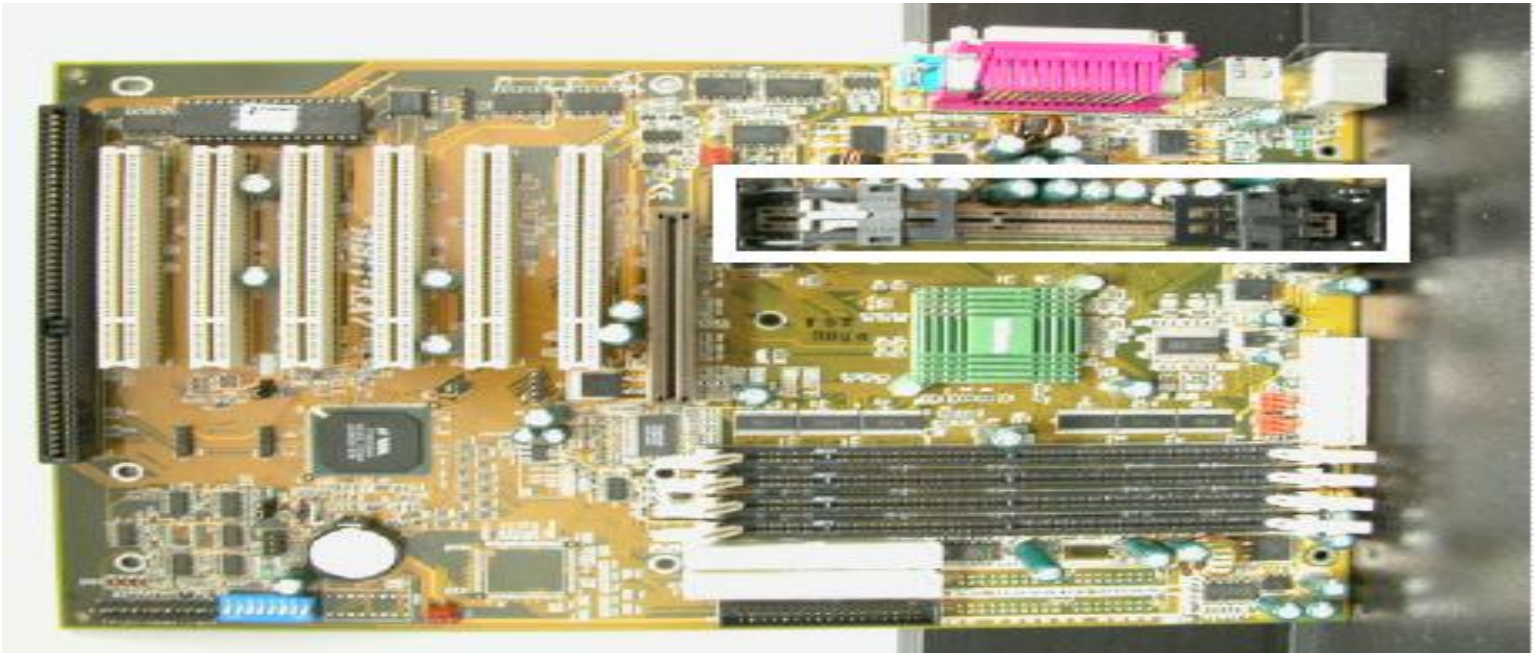
Concept de hiérarchie mémoire

Registres:
Dimension: <1KB
Temps d'accès: 0.25-0.5 ns *(Note 1/1ns = 1GHz)*
Gérés par: Compilateur



Concept de hiérarchie mémoire

- **Cache:**
  Dimension: <16MB
  Temps d'accès: 0.5-25 ns *(Note: court délais supplémentaires ex. décodeur plus complexe)*
  Géré par: Hardware (matériel)



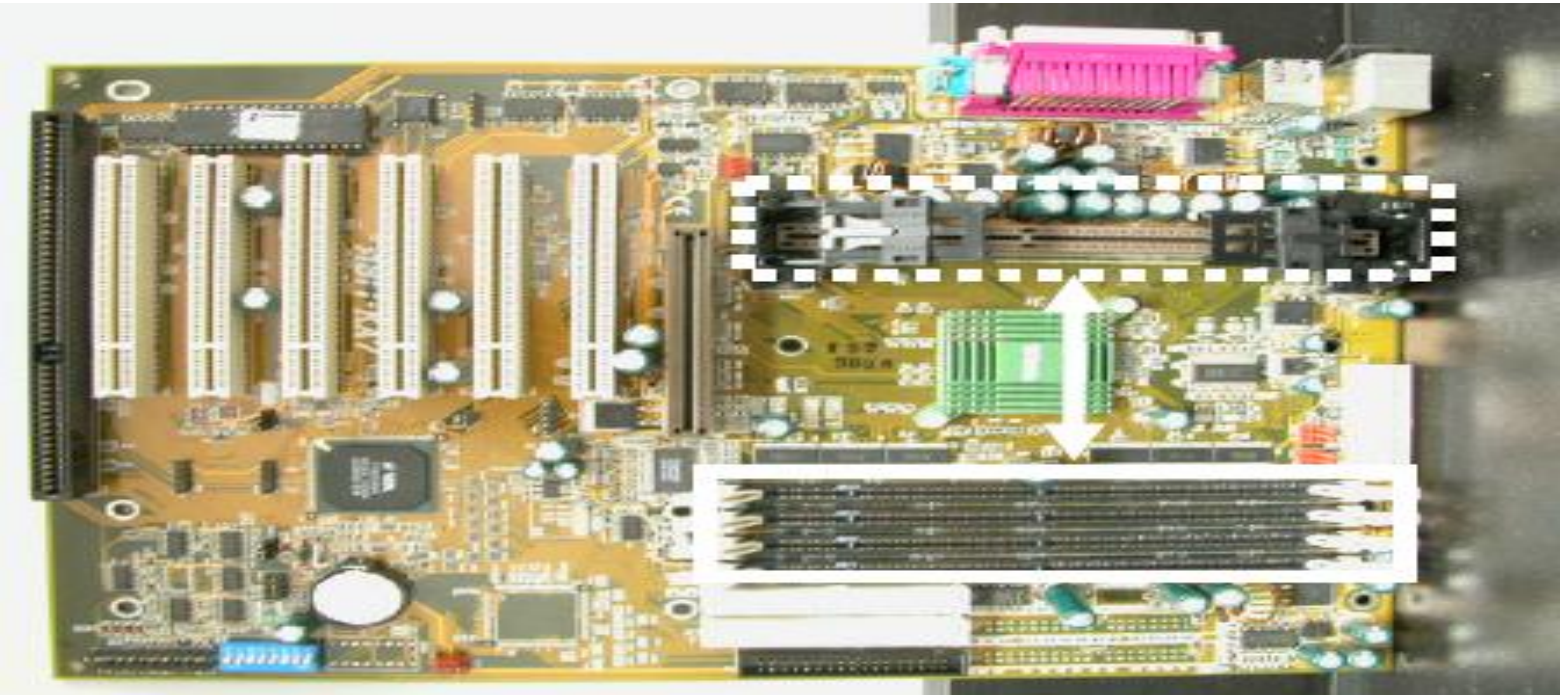
Concept de hiérarchie mémoire

Mémoire principale:

Dimension: <16GB

Temps d'accès: 80-250 ns

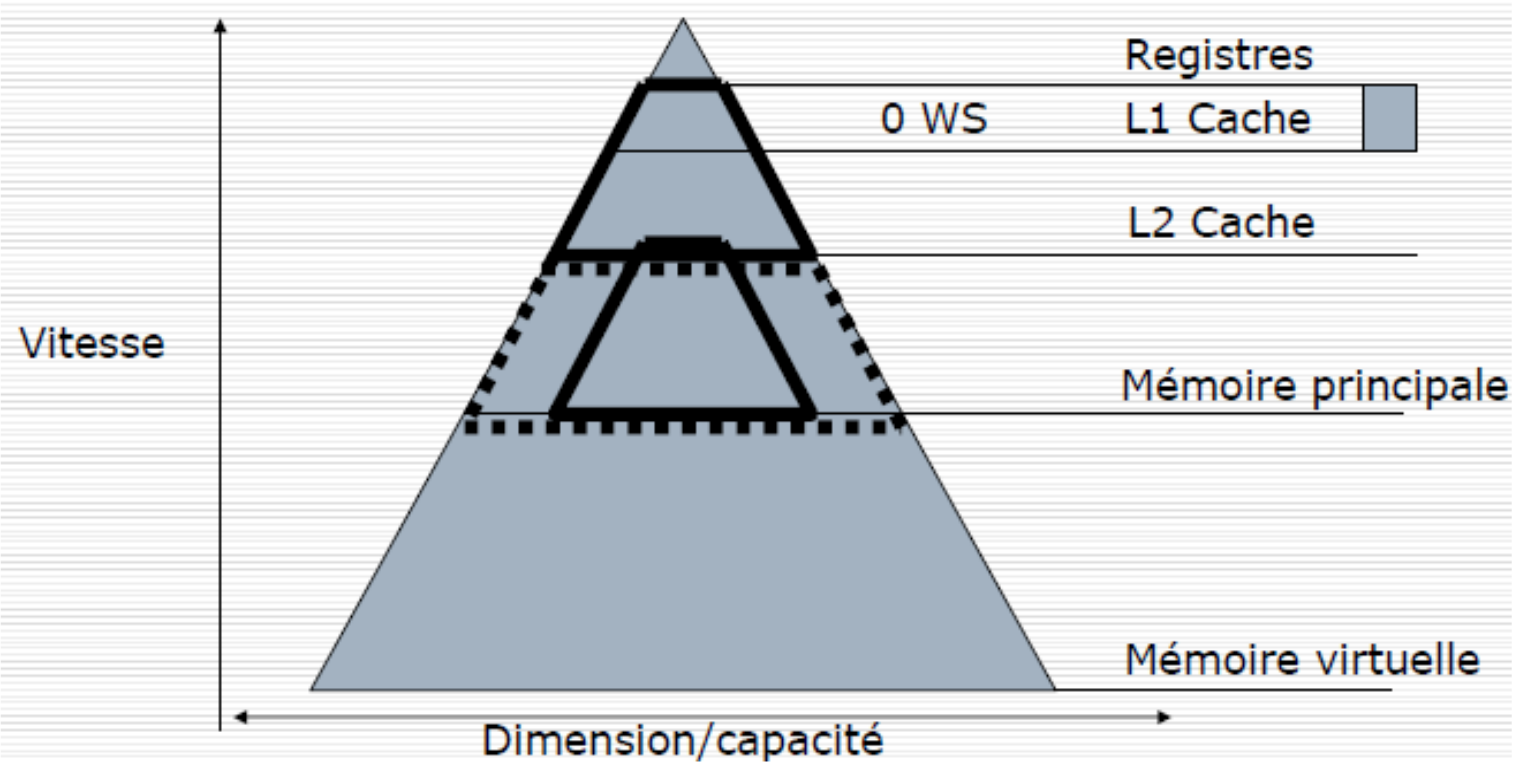
Géré par: Operating System -OS (système d'exploitation)



Réalisation de mémoire

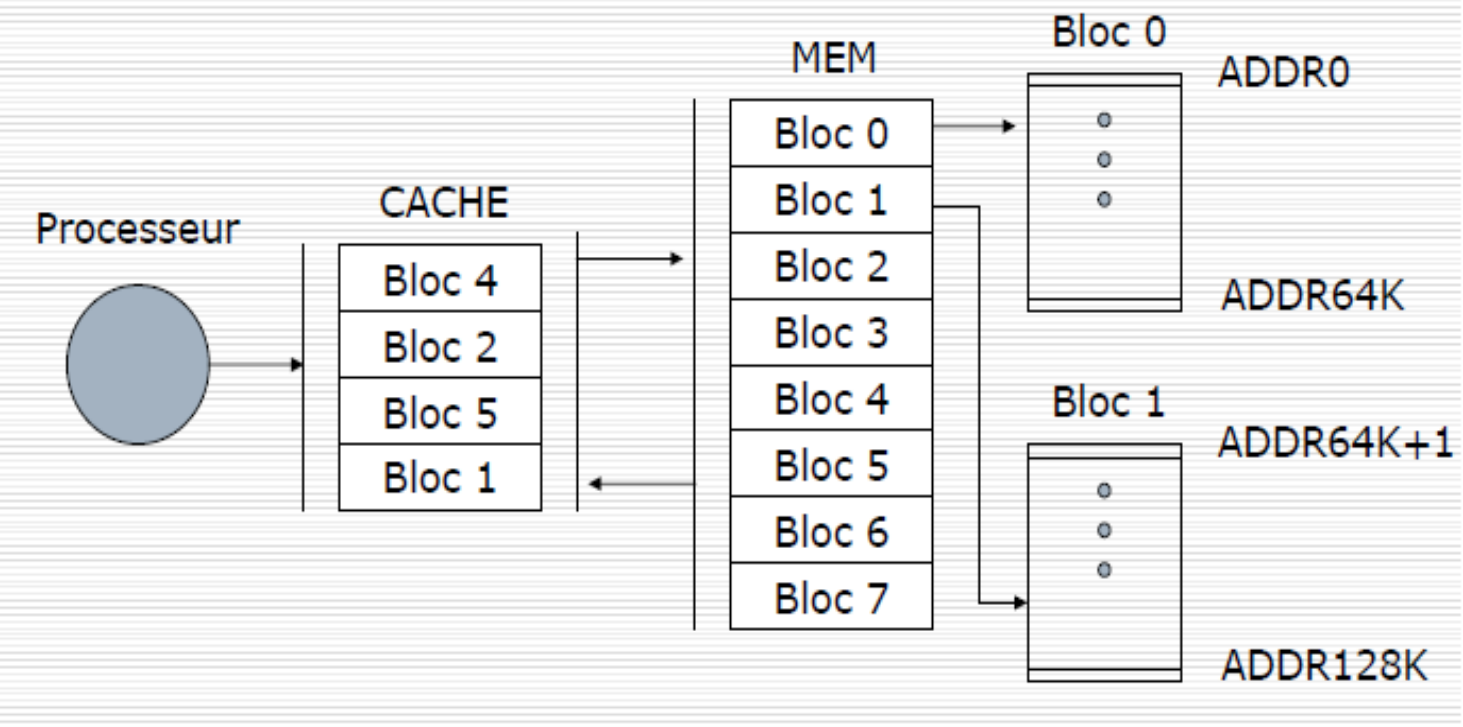
- **Pour les DRAM**
  - Un bit = un transistor et un condensateur
  - Le condensateur stocke la valeur binaire
  - Doit être rafraîchi régulièrement pour conserver la valeur stockée dans le condensateur
    - Ralentit la vitesse d'accès à la mémoire
- **Pour les SRAM**
  - Un bit = 4 transistors = 2 portes NOR
  - Peut notamment les construire à partir de bascules RS

Copie d'une portion de MP dans le cache



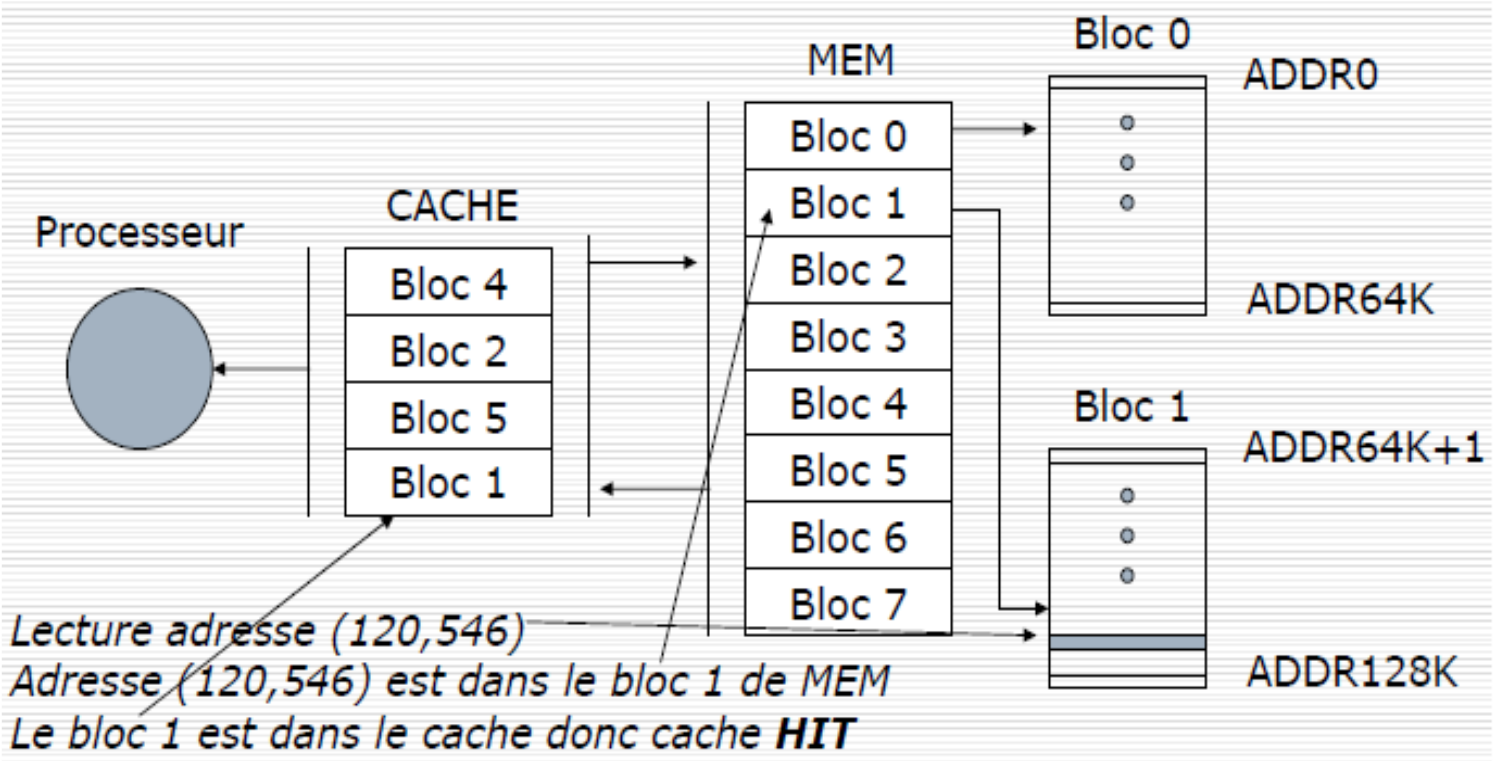
Cache

- **Cache à 4 blocs, mémoire à 8 blocs**



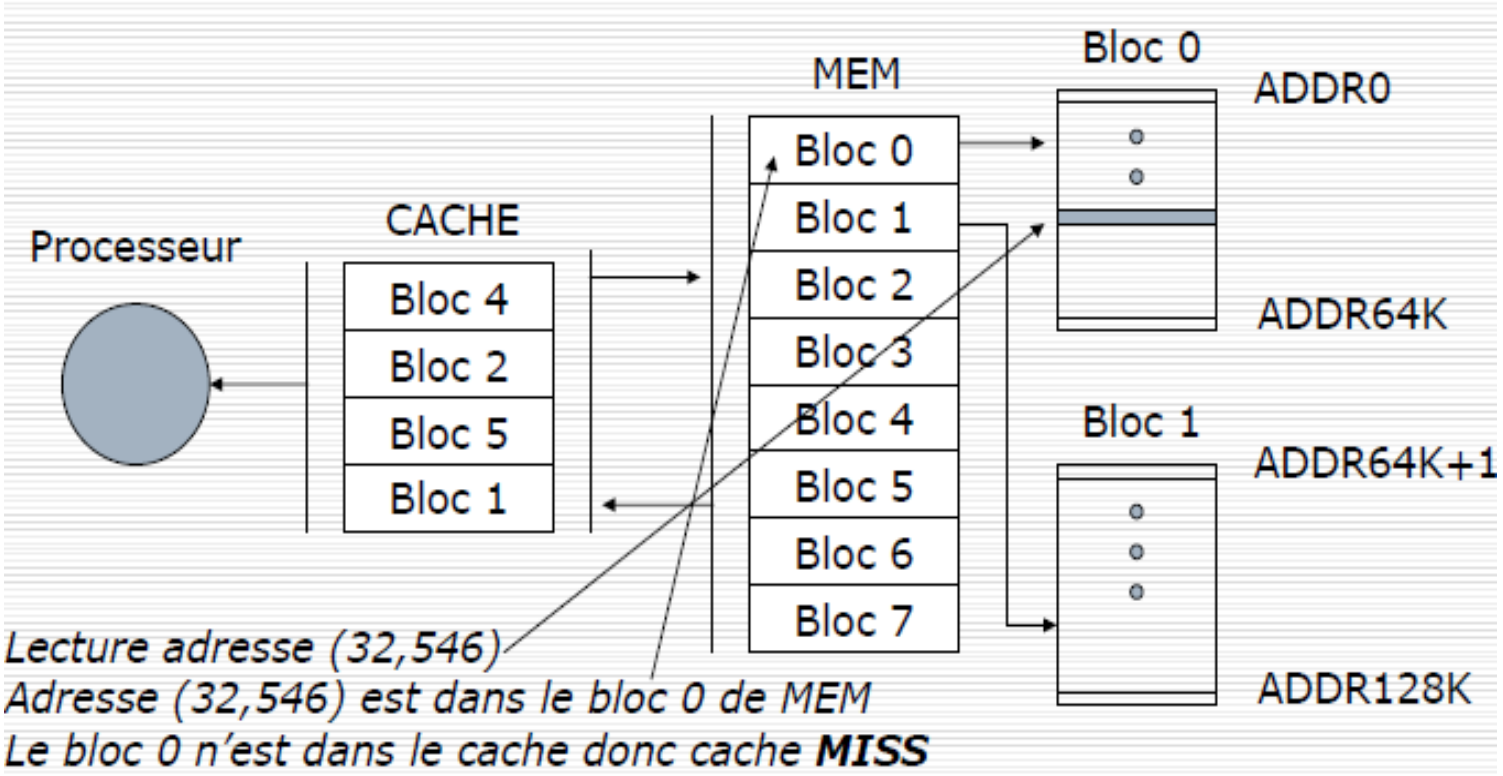
Cache

- **Fonctionnement intuitif : Succès (Hit)**



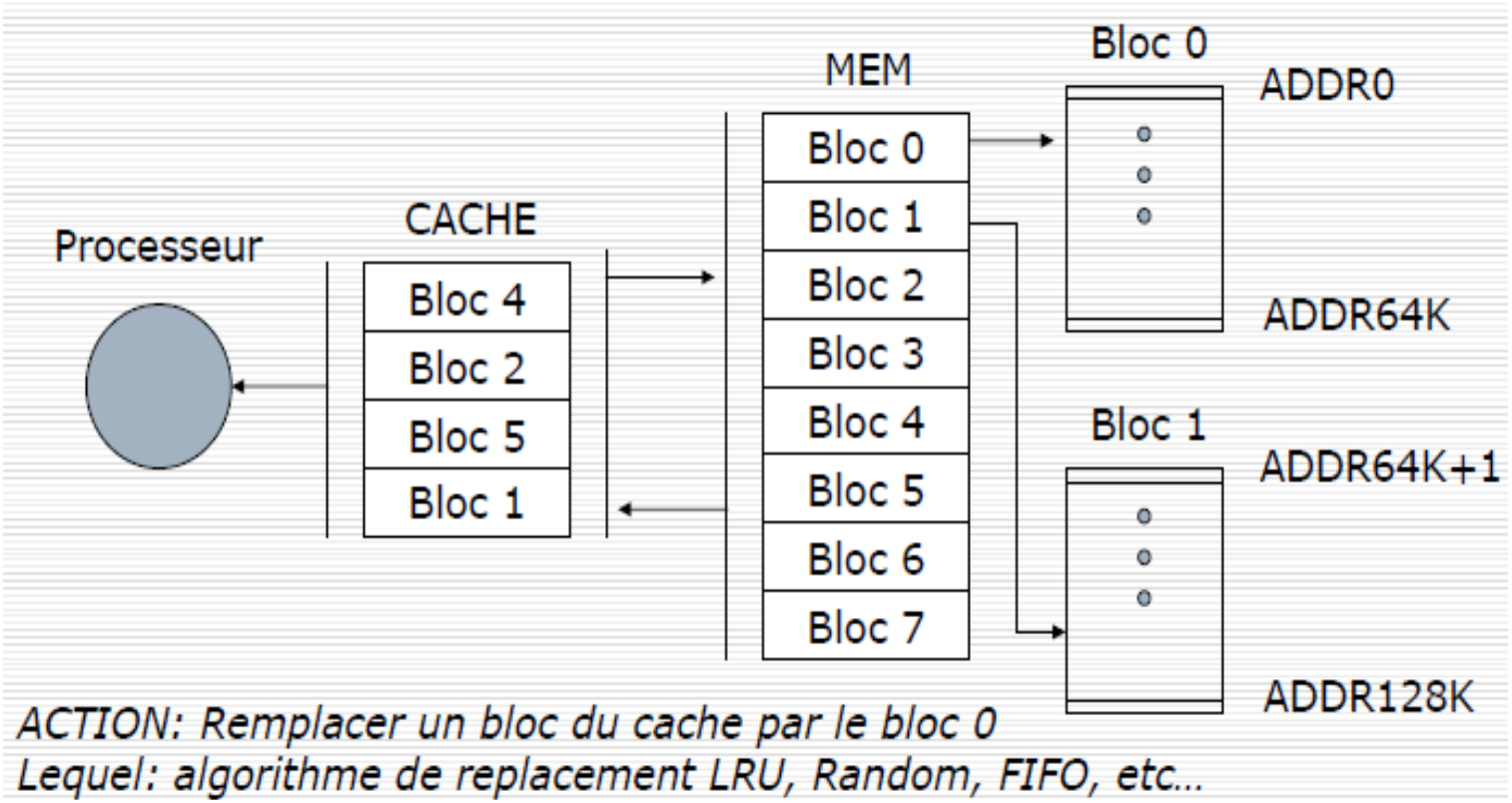
Cache

❑ Fonctionnement intuitif : Echec (Miss)



Cache

❑ Fonctionnement intuitif : Echec (Miss)

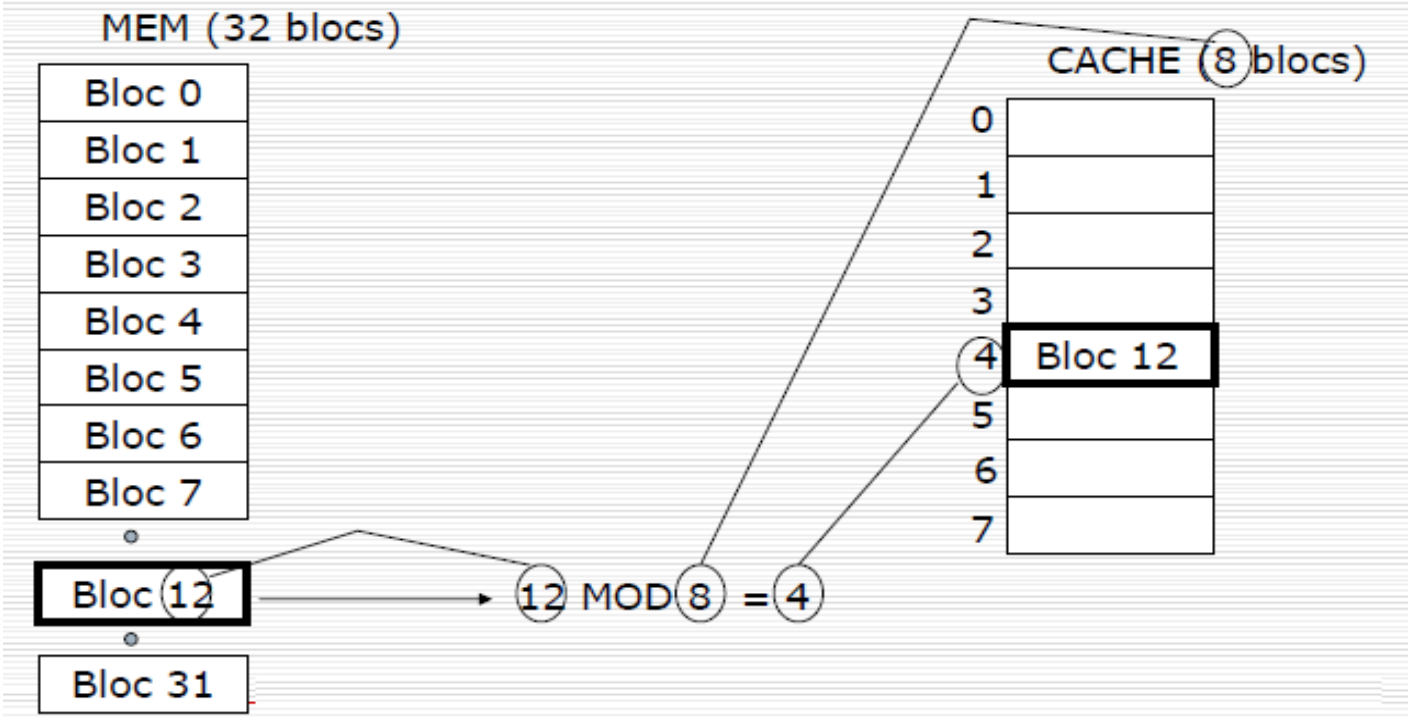


Cache : organisation

- Cache est le nom généralement donné au premier niveau de hiérarchie mémoire que l'on rencontre en quittant l'UCT.
- Il existe trois types d'organisation de cache:
  - Si chaque bloc a uniquement un seule place possible dans le cache, c'est un cache à correspondance directe (*direct mapped*).
    (Numéro de bloc) mod (nombre de blocs dans le cache)
  - Si un bloc peut être placé n'importe où dans le cache, c'est un cache totalement associatif (*fully associative*).
  - Si un bloc peut être placé dans un ensemble restreint de places, c'est un cache associatif par ensemble de blocs (*set associative*). L'ensemble est choisi par sélection de bits :
    (Numéro de bloc) mod (Nombre d' ensembles dans le cache)

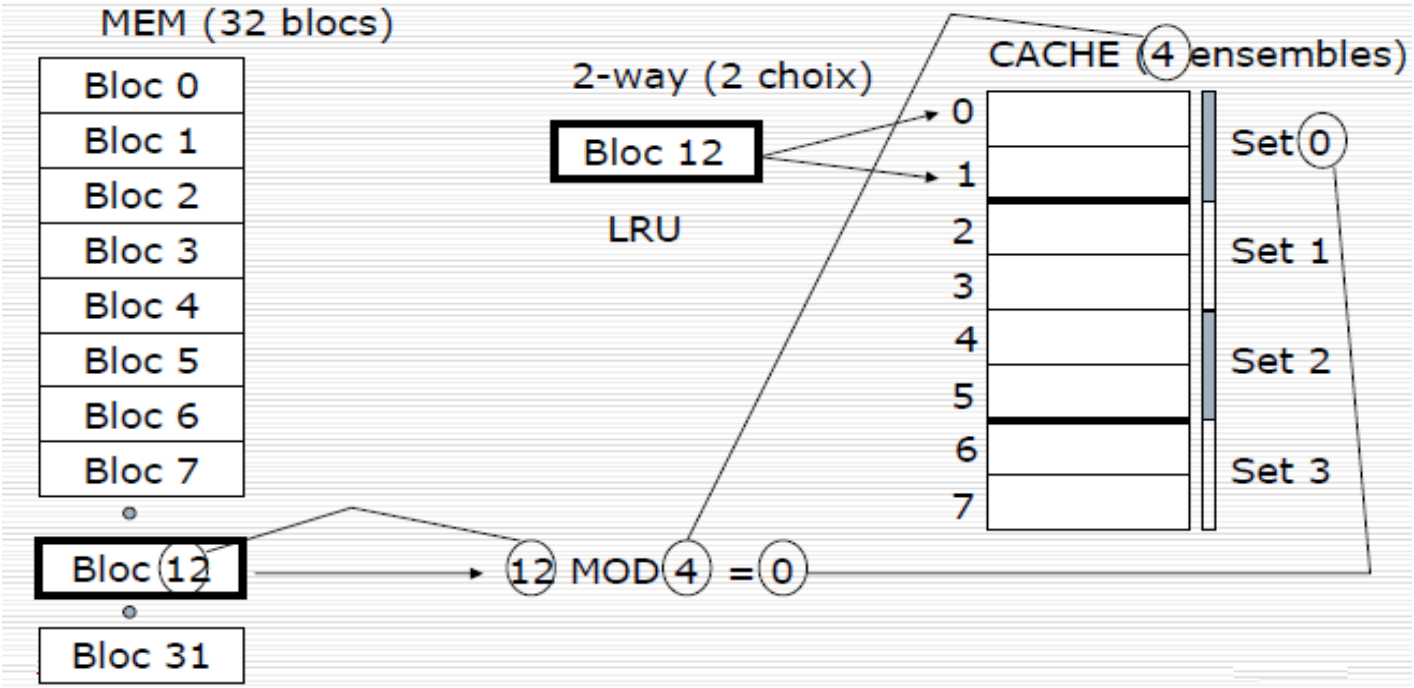
Cache : organisation

- Cache à 8 blocs, mémoire à 32 blocs
  - Correspondance directe



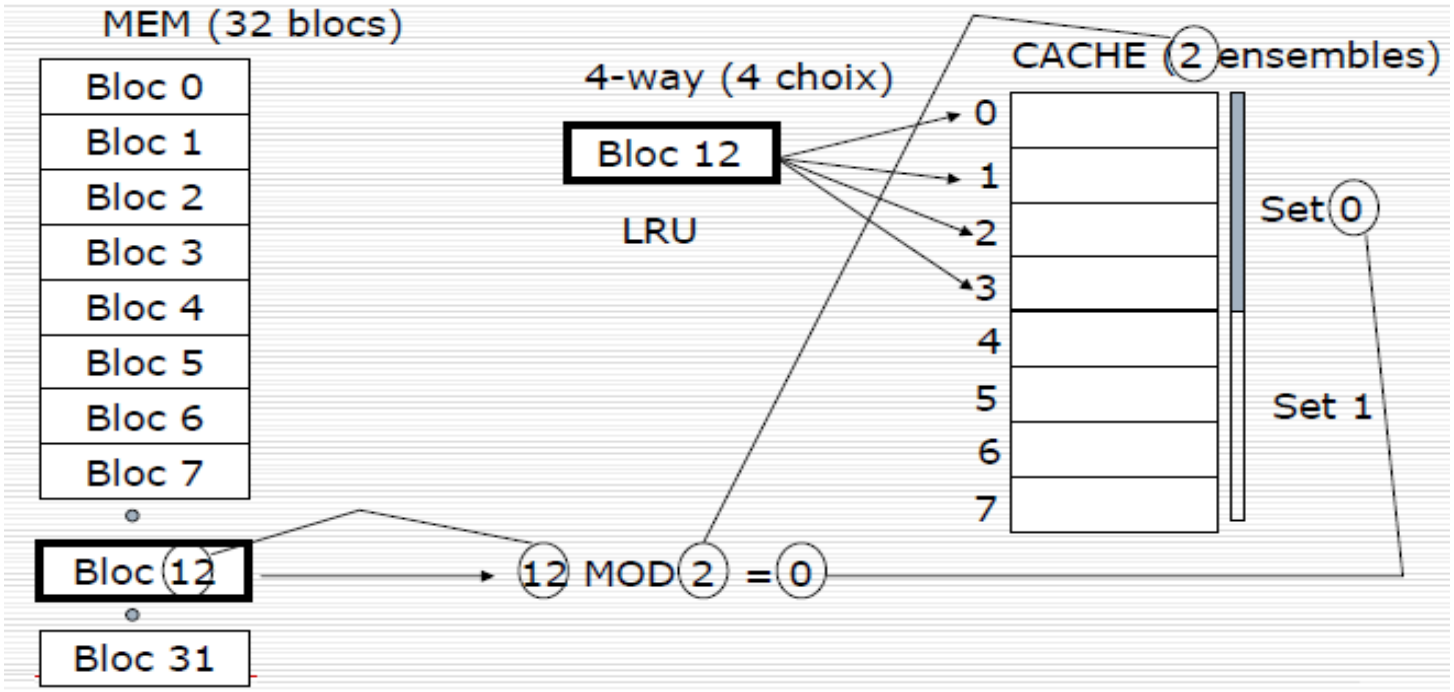
Cache : organisation

- Cache à 8 blocs, mémoire à 32 blocs
  - Associativité par ensemble (2-way set)



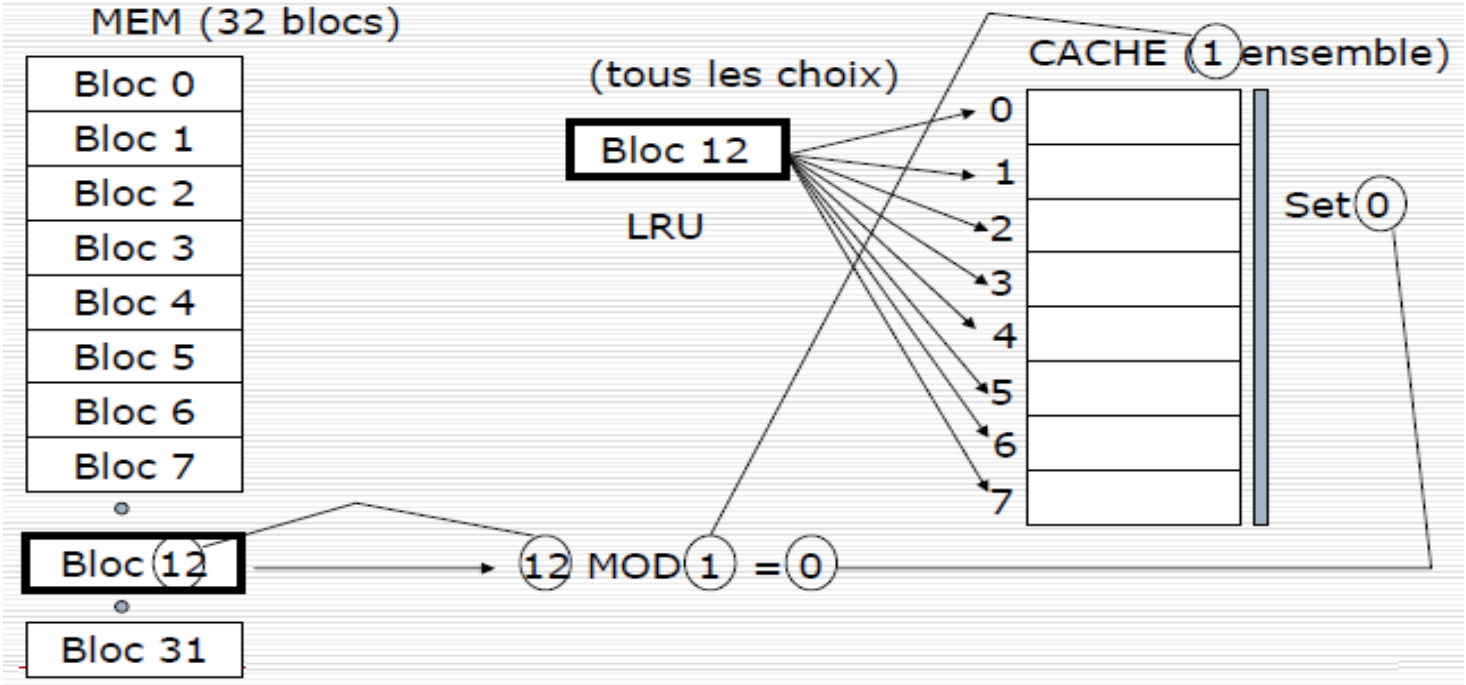
Cache : organisation

- Cache à 8 blocs, mémoire à 32 blocs
  - Associativité par ensemble (4-way set)



Cache : organisation

- Cache à 8 blocs, mémoire à 32 blocs
  - Associativité totale



Cache

- **Associativité – Impact: Algorithmes de remplacement et dimension (Statistiques: échecs par 1000 instructions) – 2-way set associative**

Dimension	LRU	Random	FIFO
16KB	114.1	117.3	115.5
64KB	103.4	104.3	103.9
256KB	92.2	92.1	92.5

Cache

- **Associativité – Impact: Algorithmes de remplacement et dimension (Statistiques: échecs par 1000 instructions) – 4-way set associative**

Dimension	LRU	Random	FIFO
16KB	111.7	115.1	113.3
64KB	102.4	102.3	103.1
256KB	92.1	92.1	92.5

Cache

- **Associativité – Impact: Algorithmes de remplacement et dimension (Statistiques: échecs par 1000 instructions) – 8-way set associative**

Dimension	LRU	Random	FIFO
16KB	109.0	111.8	110.4
64KB	99.7	100.5	100.3
256KB	92.1	92.1	92.5

Cache : Observations

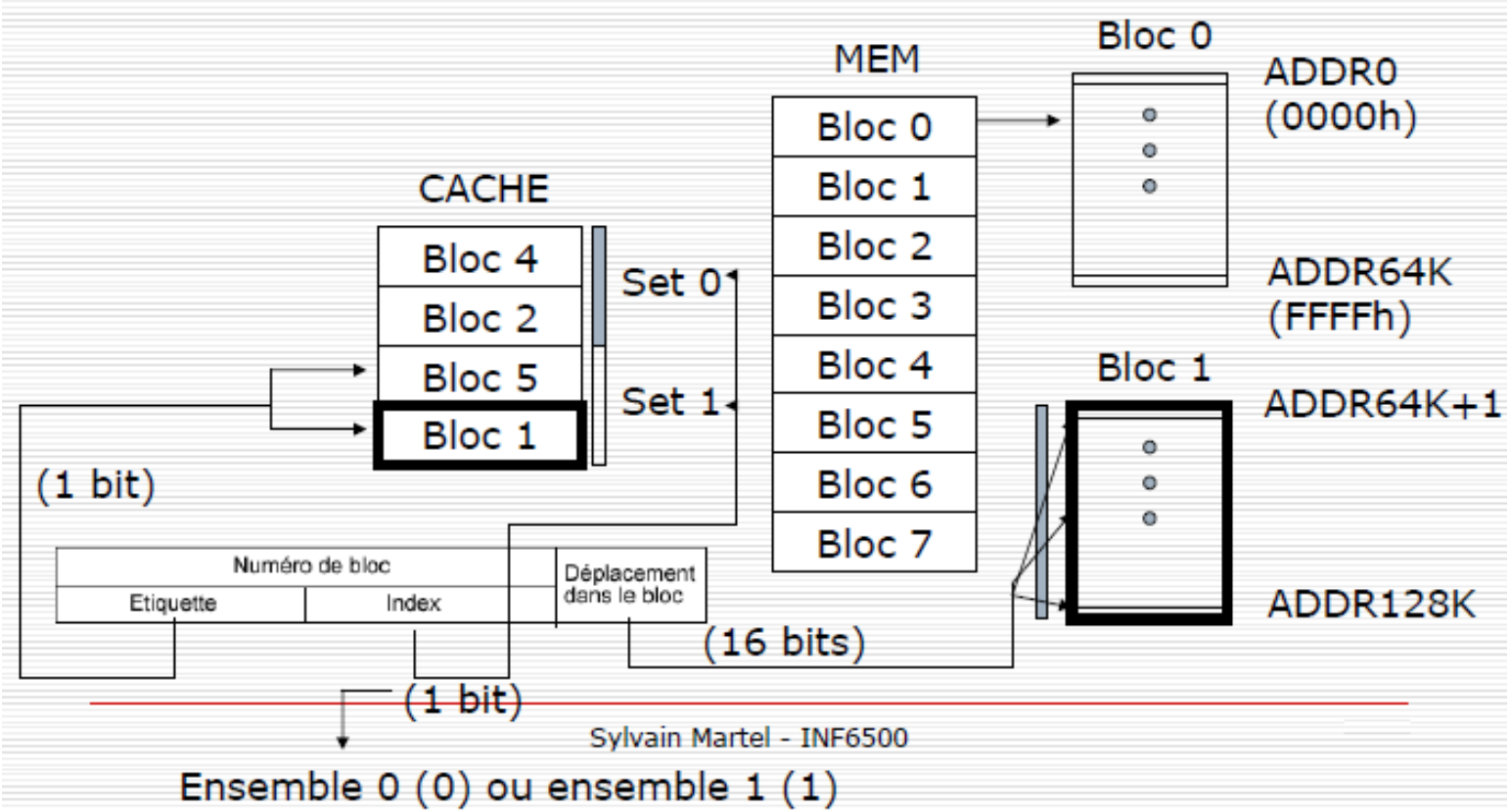
- Peu de différence entre LRU et random pour les dimensions plus grandes de cache
- LRU meilleure pour caches de dimensions plus petites
- FIFO généralement plus performant que random pour les caches de plus petites dimensions

Cache

- **Associativité – Impact : Niveaux d'associativité (Statistiques: échecs par 1000 instructions) Algorithme de remplacement : LRU**

Dimension	2-way	4-way	8-way
16KB	114.1	111.7	109.0
64KB	103.4	102.4	99.7
256KB	92.2	92.1	92.1

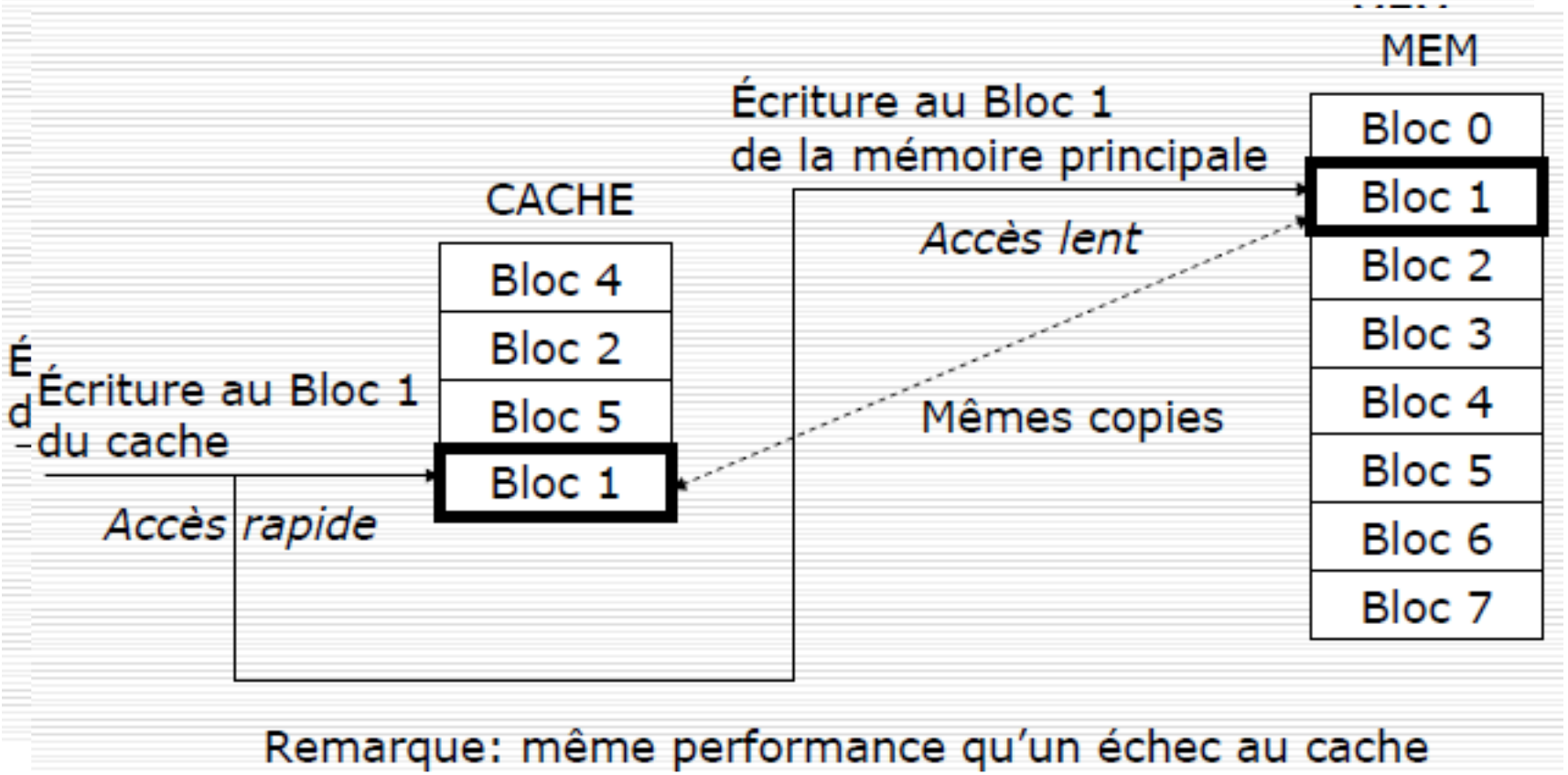
Cache : Blocs


MEM
Bloc 0
Bloc 1
Bloc 2
Bloc 3
Bloc 4
Bloc 5
Bloc 6
Bloc 7
CACHE
Bloc 4
Bloc 2
Bloc 5
Bloc 1
Set 0
Set 1
(1 bit)
Numéro de bloc
Etiquette
Index
Déplacement dans le bloc
(16 bits)
(1 bit)
Ensemble 0 (0) ou ensemble 1 (1)
Bloc 0
ADDR0
(0000h)
ADDR64K
(FFFFh)
Bloc 1
ADDR64K+1
ADDR128K
Sylvain Martel - INF6500

Cache

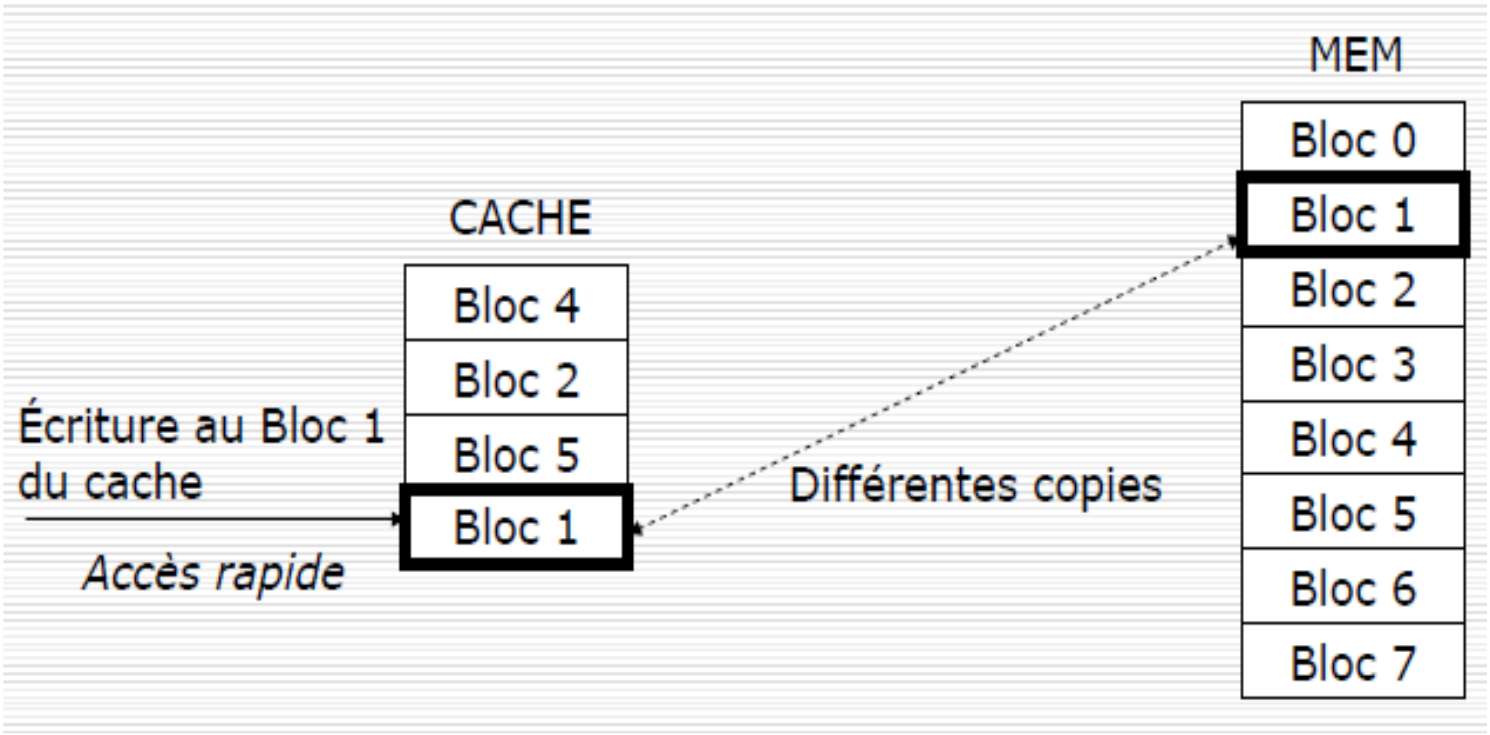
- Les lectures dominent les accès cache (25% du trafic du cache de données pour les écritures). Il faut donc que celles-ci soit optimisées afin de diminuer l'attente des processeurs.
- Il est possible de lire le bloc pendant que l'étiquette est lue et comparée. Ce n'est pas le cas pour l'écriture.
- Les écritures prennent plus de temps que les lectures.
- Il existe deux options pour écrire dans le cache:
  - L'écriture simultanée (*write through*): L'information est écrite à la fois dans le bloc du cache et dans le bloc de la mémoire.
  - La réécriture (*write/copy back)*: L'information est écrite uniquement dans le bloc du cache. Le bloc modifié du cache est recopié en mémoire principale uniquement quand il est remplacé.

Cache : Ecriture simultanée

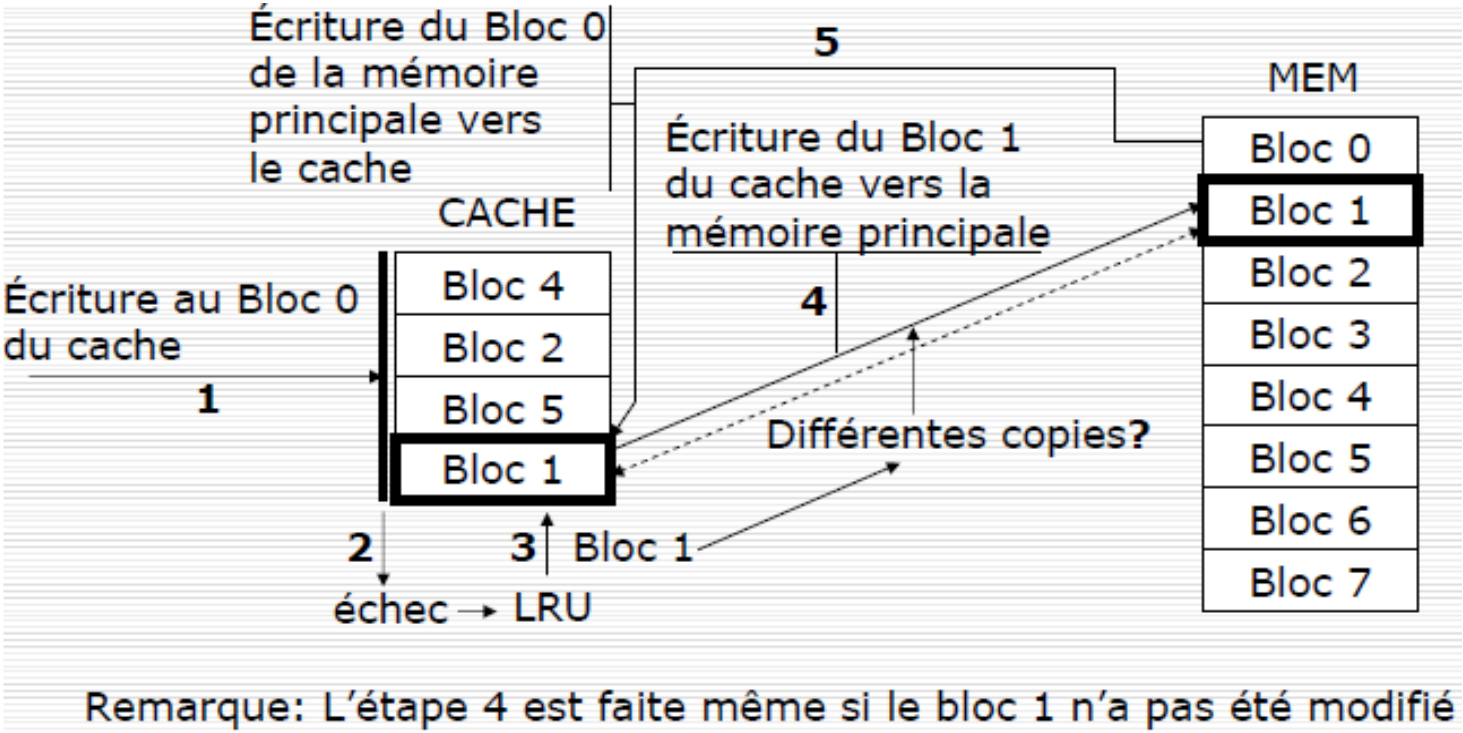


Remarque: même performance qu'un échec au cache

Cache : Réécriture



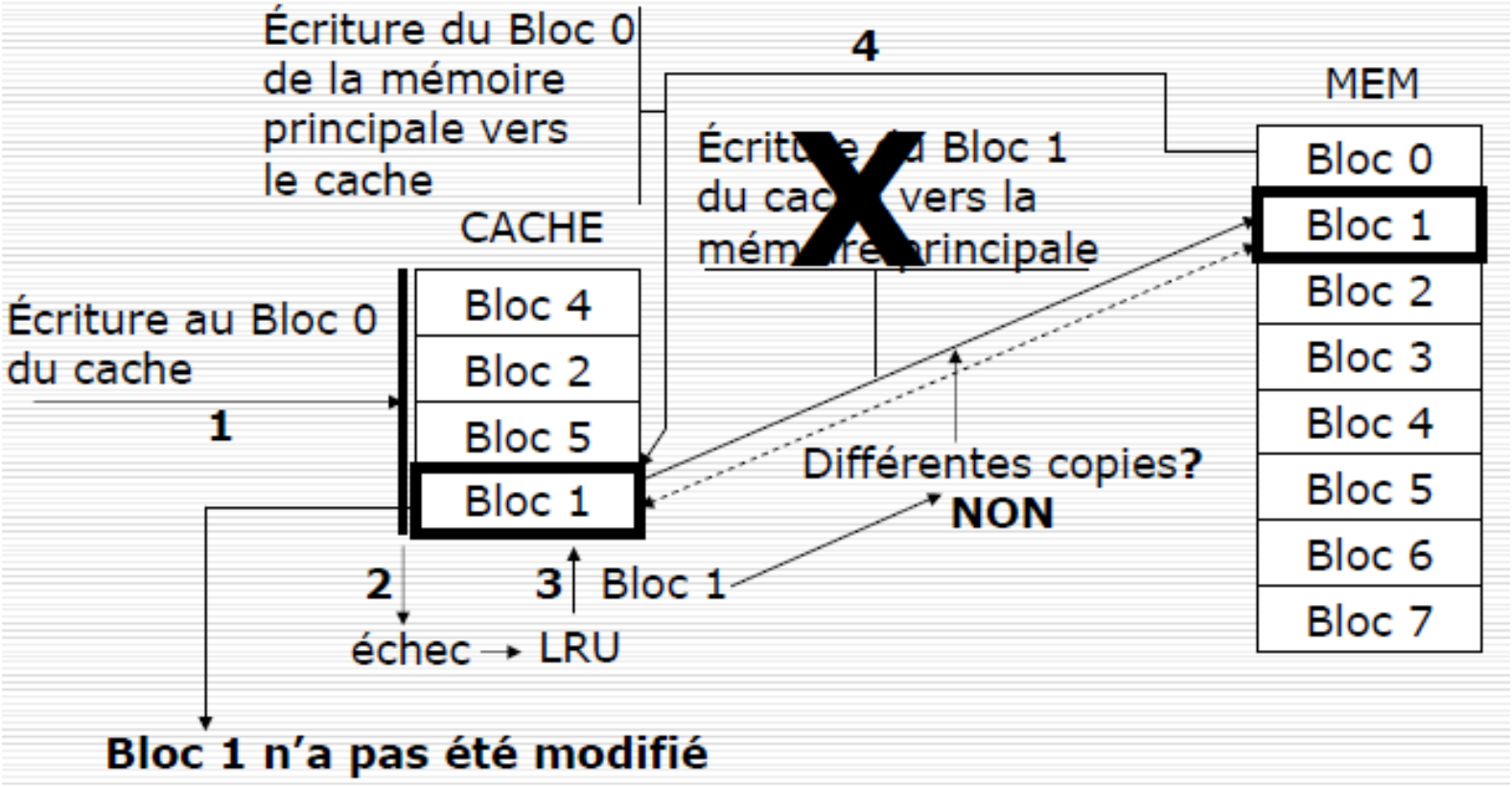
Cache : Réécriture



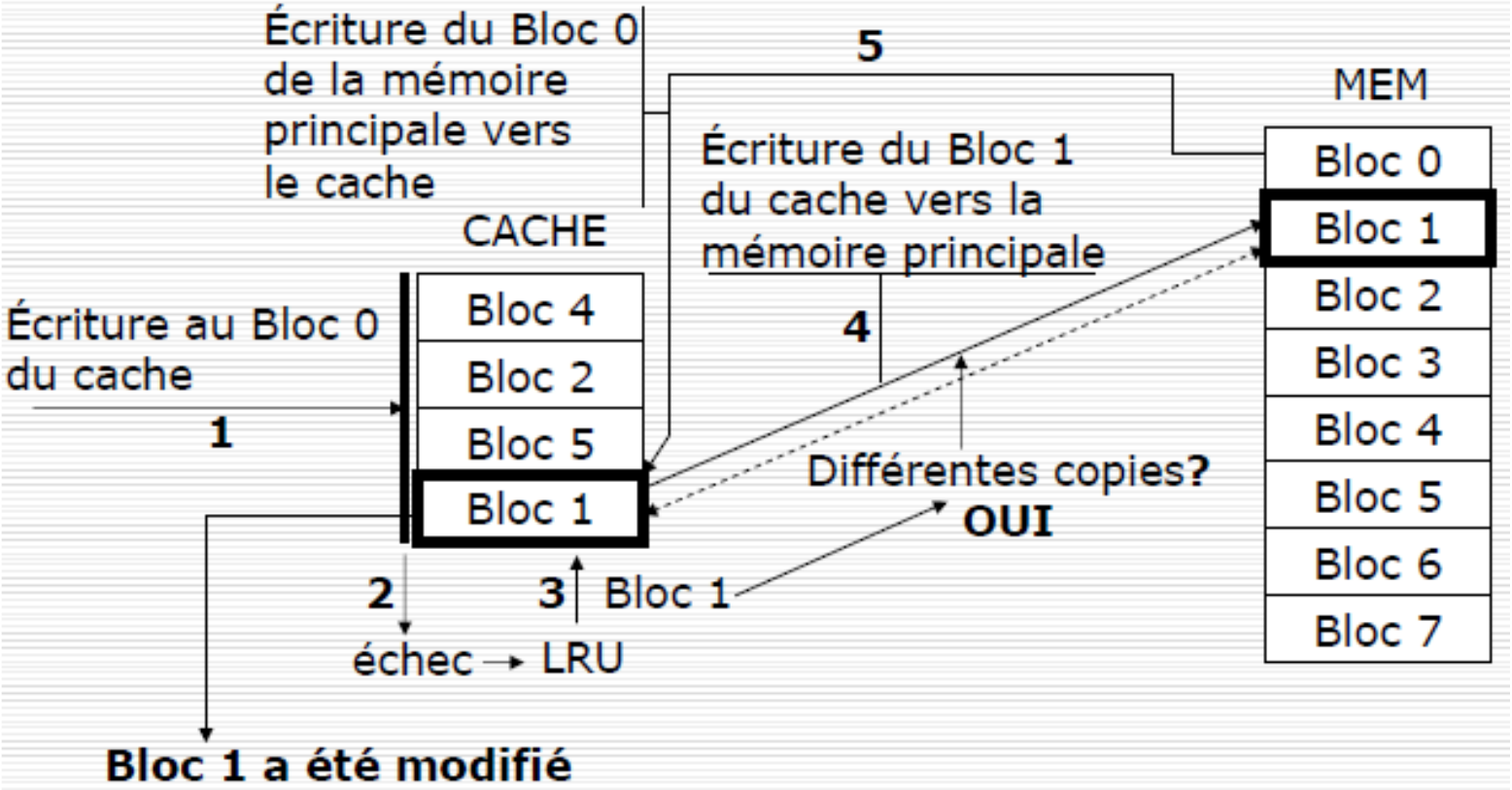
Cache : Dirty bit

- Pour diminuer le nombre de remplacements, on ajoute un bit modifié (*dirty bit*).
  - Ce bit indique si le bloc a été modifié on non.
  - S'il ne l'a pas été, le bloc n'est pas écrit lors d'un défaut de cache puisque l'information est la même.

Cache : Echec avec dirty bit



Cache : Echec avec dirty bit

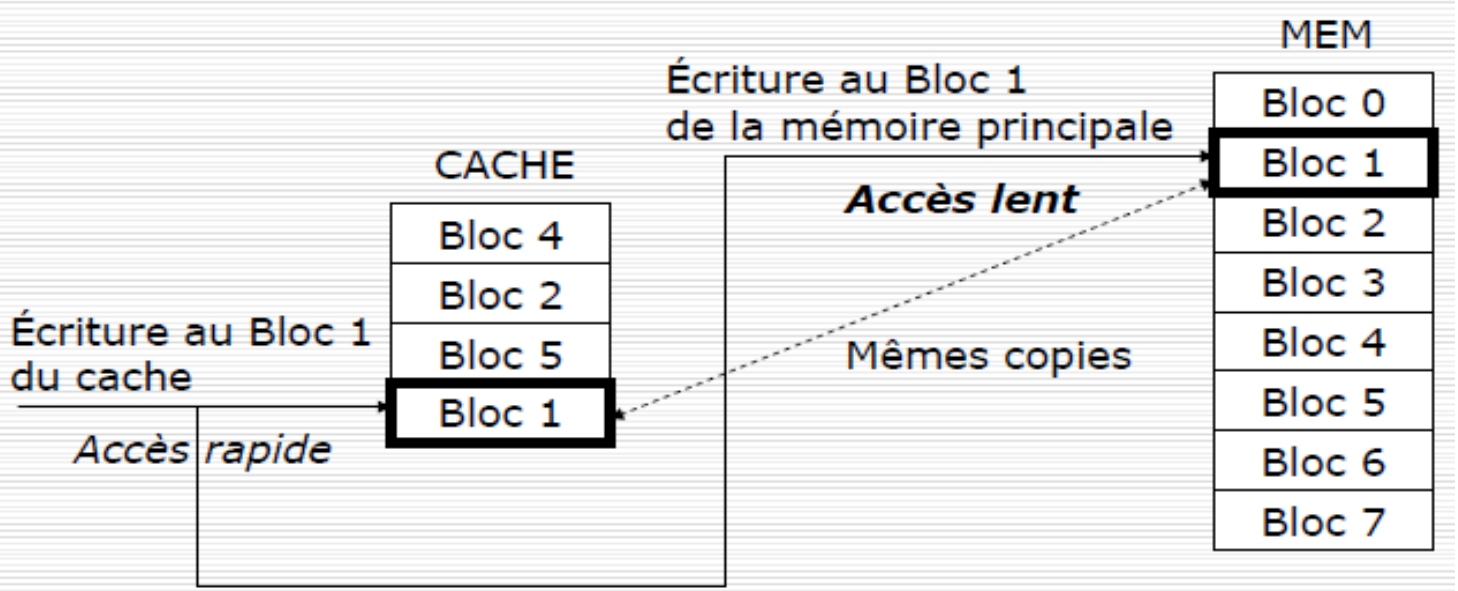


Cache : Tampon d'écriture

- L'UCT doit attendre la fin des écritures lors d'écritures simultanées. On dit alors que l'UCT est en suspension d'écriture (*write stall*).
- Un tampon d'écriture (*write buffer)* permet à l'UCT de continuer son traitement une fois que la donnée est écrite dans le tampon.

Cache

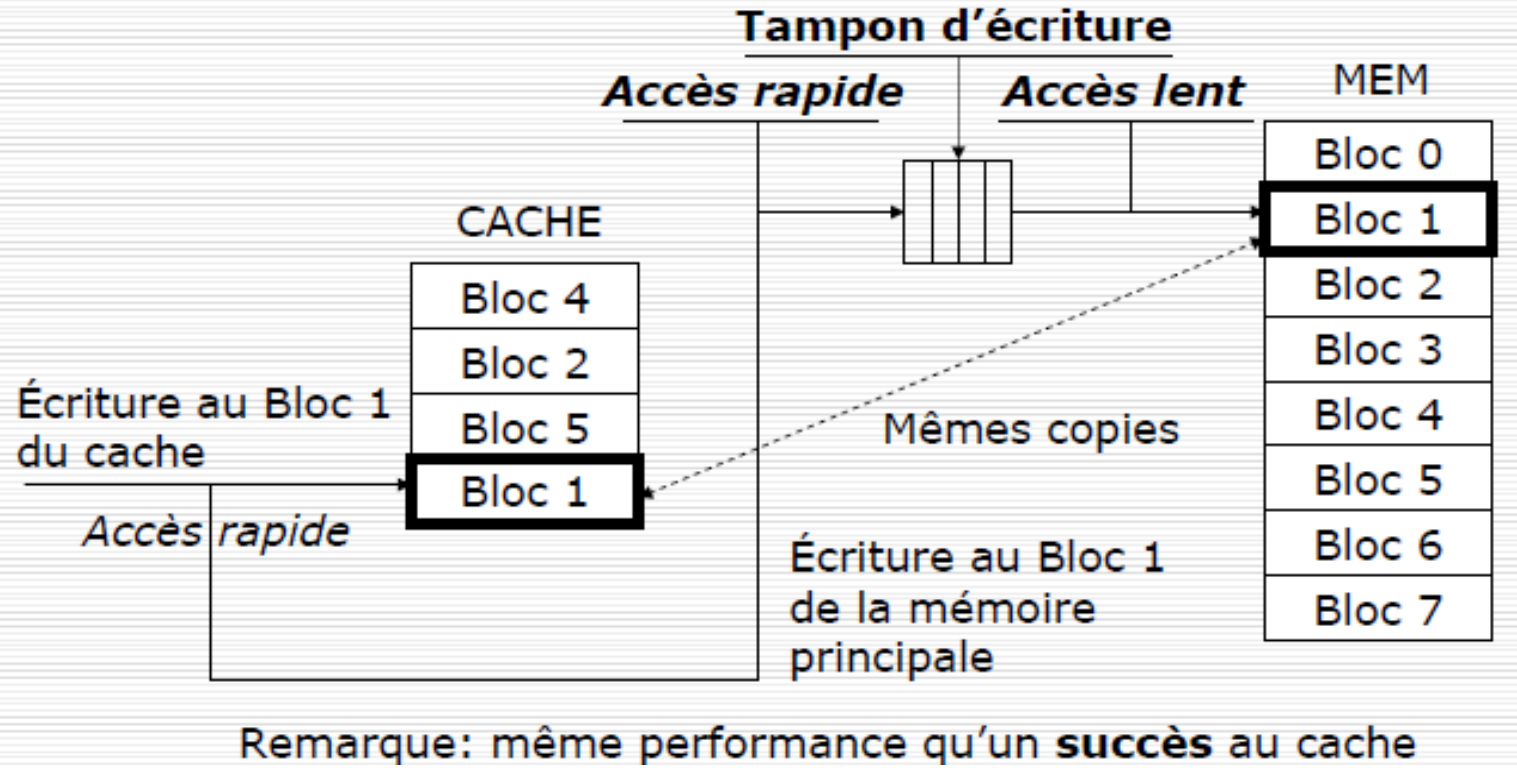
- La base des caches (L'écriture simultanée **sans** tampon d'écriture – succès au cache)



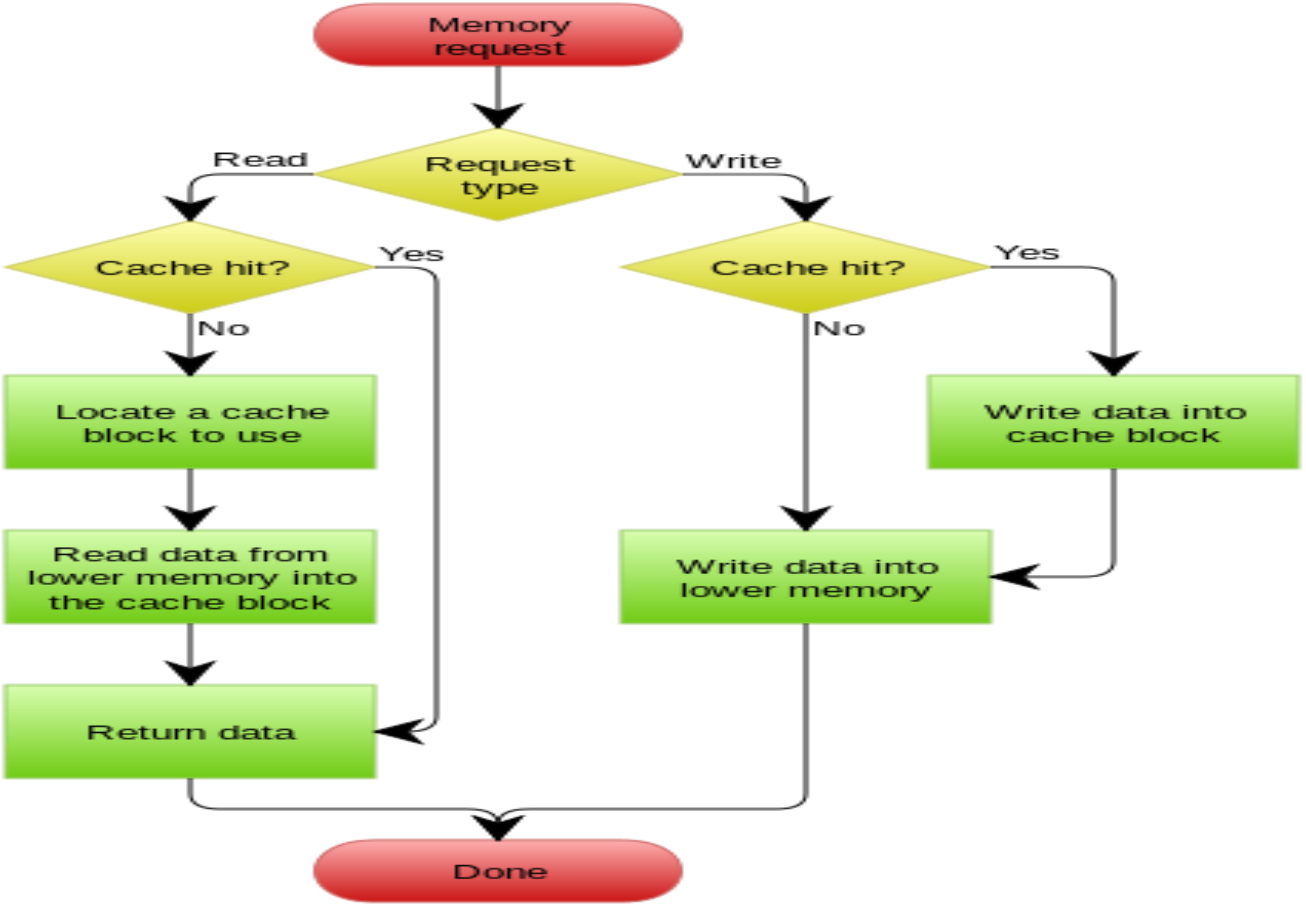
Remarque: même performance qu'un échec au cache

Cache

- La base des caches (L'écriture simultanée **avec** tampon d'écriture – succès au cache)



Ecriture immédiate ou WT



Réécriture différée ou WB

WT vs WB

- Write-through is slower
+ But cleaner (memory always consistent)

+ Write-back is faster
- But complicated when multi cores sharing memory

Stratégies pour améliorer la performance des caches

- Améliorer les performances des caches implique :
  - **Réduire le taux d'échecs au cache**
  - Réduire la pénalité d'échec
  - Réduire le temps d'accès réussi au cache

Types de défauts

- Il existe trois types d'échecs:
  - **Obligatoire** (compulsory) : Le premier accès à un bloc qui n'est pas dans le cache, et il doit être transféré dans le cache ; ils sont aussi appelés échecs de démarrage à froid.
  - **Capacité** (capacity) : Si le cache ne peut contenir tous les blocs nécessaires pendant l'exécution d'un programme, des échecs de capacité interviendront à cause de blocs écartés puis rappelés plus tard.
  - **Conflit** (conflict): Si la stratégie de placement des blocs est associative par ensemble ou à correspondance directe, les échecs de conflit interviennent parce qu'un bloc peut être écarté et rappelé plus tard si trop de blocs doivent être placés dans son ensemble.

Types de défauts

- Types de taux d'échecs % /100% d'échecs – LRU

Statistiques pour 4 KB cache

Associativité	Compulsif	Capacité	Conflit
1-way	0.1	72	28
2-way	0.1	93	7
4-way	0.1	99	1
8-way	0.1	100	0

Statistiques pour 512 KB cache

Associativité	Compulsif	Capacité	Conflit
1-way	0.8	66	33
2-way	0.9	71	28
4-way	1.1	91	8
8-way	1.1	95	4

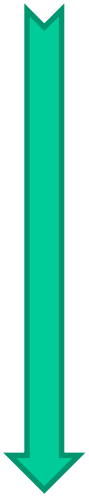
Observations

- Le nombre d'échecs de type compulsif est indépendant de la dimension du cache
- Le nombre d'échecs de type capacité diminue lorsque la dimension du cache augmente
- Le nombre d'échecs de type conflit diminue lorsque l'associativité augmente

Réduire le taux d'échecs au cache

- **Première solution : Une taille de bloc plus grande**.
  - L'augmentation de la taille du bloc, diminue les échecs obligatoires en raison de la localité spatiale.
  - Elle augmente aussi la pénalité d'échec
    - moins de blocs dans le cache,
    - bloc plus grand à remplacer,
  - plus d'échecs de type conflit.

Statistiques – Taux d'échecs vs. Dimension du bloc



Dimension du bloc	Cache 4K	Cache 64K	Cache 256K
16	8.57%	2.04%	1.09%
64	7.00%	1.06%	0.51%
256	9.51%	1.15%	0.49%

Trade off (Compromis)

- **Pour un cache de taille donnée, une large taille de blocs signifie :**
  - Peu de lignes
  - Donc peu d'étiquettes (étiquettes petites pour des caches associatives)
  - Moins de coût
  - Peu d'échecs à froid
- **Mais aussi :**
  - Peu de blocs disponibles (for scattered accesses : accès dispersés))
  - Plus de conflits
  - Et large pénalité d'échec (time to fetch block)

Réduire le taux d'échecs au cache

- **Seconde solution: Une mémoire cache plus grande**.
  - L'augmentation de la taille du cache,
    - diminue les échecs de capacité.
    - Elle diminue aussi le taux d'échec du cache
- Exemple de performance :

Cache 1-way en ko	% total d'échecs	% total d'échecs capacité
4 ko	9,8%	7%
8 ko	6,8%	4,4%
16 ko	4,9%	4%
256ko	0,8%	0,5%

Réduire le taux d'échecs au cache

- **Troisième solution: Une associativité plus élevée.**
  - Une associativité élevée améliore les taux d'échecs
    - Efficacité : 8-way set ≈ associativité totale.
    - **Loi empirique 2:1** (un cache de taille N a environ le même taux d'échec qu'un cache associatif à deux blocs par ensemble de taille N/2).
  - Une associativité élevée augmente le temps d'accès réussi

Remarque : Améliorer un des aspects du temps d'accès mémoire se fait au dépens d'un autre

Réduire le taux d'échecs au cache

- Statistiques de pourcentage du taux d'échecs pour différents niveaux d'associativité

Associativité	Cache 4K	Cache 64K	Cache 512K
1-way	9.8%	3.7%	0.8%
2-way	7.6%	3.1%	0.7%
4-way	7.1%	3.0%	0.6%
8-way	7.1%	2.9%	0.6%

Réduire le taux d'échecs au cache

- Règle 2:1 cache s'applique pour dimensions de 128 KB et moins :
  - Un cache à correspondance directe de dimension N a approximativement le même taux d'échecs que 2-way set associative de dimension N/2
    - Note : Un cache à correspondance directe est plus simple
    - mais a plus de SRAM que cache associatif de même performance,
    - cependant il est possible que la complexité de la logique de contrôle du cache associatif engendre des délais additionnels donc baisse de la fréquence horloge
    - ou compensation avec SRAM plus rapide donc augmentation de $,

Réduire le taux d'échecs au cache

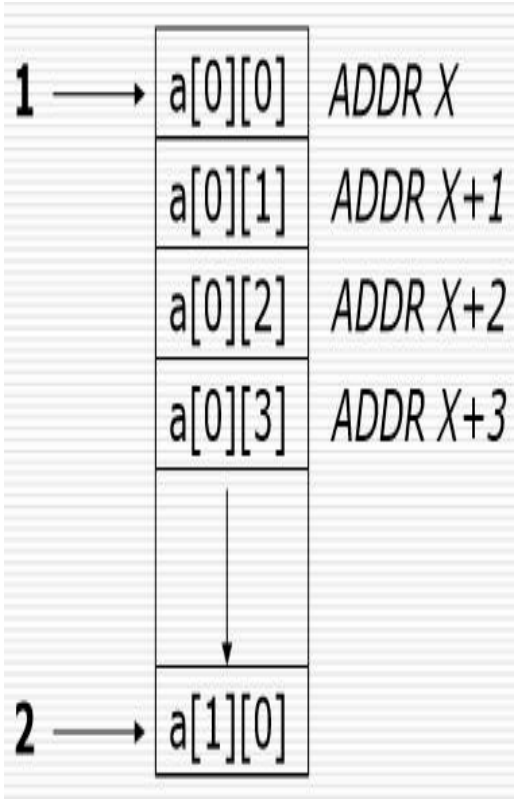
- **Autre solution : Optimisation du compilateur**
  - Aucune modification matérielle nécessaire.
  - Tente d'améliorer la localité spatiale et temporelle des données.

Ordre de chargements

```
for (i = 0 ; i<1 ; i = i + 1)
        for(j = 0 ; j < 100 ; j = j+1)
                a[j][i]=…
```

- Le tableau **a** n'est pas écrit dans l'ordre de chargement en mémoire,
- il ne bénéficie donc pas d'une localité spatiale.

Accès

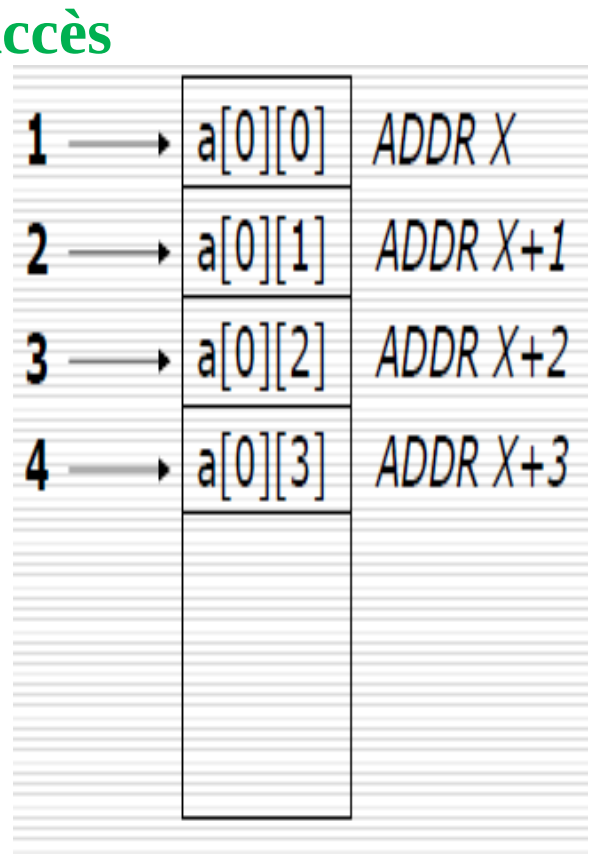


a[i][j]

Ordre de chargements

```
for (i = 0 ; i<1 ; i = i + 1)
        for(j = 0 ; j < 100 ; j = j+1)
                a[i][j]=…
```

- Le tableau a est écrit dans l'ordre de chargement en mémoire,
- Il bénéficie donc d'une localité spatiale.



La fusion de tableaux

- **La fusion de tableaux** (augmente la localité spatiale)
  - Combiner des tableaux indépendants en un seul tableau composé de telle sorte qu'un seul bloc de cache contienne les éléments voulus
    - **Avant :**

```
int val [size]
int key[size]
```

    - **Après :**

```
struct merge {
    intval;
    intkey;
};
Structmerge merged_array[size]
```

Echange de boucles

- **Réduction du taux d'échec : Optimisation du compilateur**
  - Certains programmes ont des boucles imbriquées qui accèdent aux données en mémoire de manière non séquentielle. Le simple échange de l'ordre d'imbrication des boucles peut faire correspondre l'ordre d'accès des données à leur ordre de rangement.
  - Cette technique réduit les échecs en améliorant la localité spatiale. Le réordonnancement utilise au maximum les données dans un bloc du cache avant qu'elles soient écartées.
  - Le code d'origine parcourant la mémoire par pas de 100 mots alors que la version revue accède à tous les mots d'un bloc de cache avant de passer au suivant. Cette optimisation améliore les performances du cache sans changer le nombre d'instructions exécutées.

Echange de boucles

- **Réduction du taux d'échec : Optimisation du compilateur**
  - **Avant :**

```
for  (j = 0 ; j< 100 ; j = j + 1)
        for (i = 0 ; i < 5000 ; i = i + 1=
          x[i][j] = 2 * x[i][j] ;
```

  - **Après :**

```
for  (i = 0 ; i < 5000 ; i = i + 1)
        for (j = 0 ; j < 100 ; j = j + 1=
          x[i][j] = 2 * x[i][j] ;
```

Fusion de boucles

- **Réduction du taux d'échec : Optimisation du compilateur**
  - Certains programmes ont des sections de code séparées
    - accédant à certains tableaux avec les mêmes boucles,
    - exécutant différents calculs sur les données communes.
  - En fusionnant le code en une seule boucle,
    - les données qui sont amenées dans le cache peuvent être utilisées de nombreuses fois avant d'être rejetées.
    - Ici contrairement à la précédente technique, le but de cette optimisation est de réduire les échecs par amélioration de la localité temporelle.

Fusion de boucles

- **Réduction du taux d'échec : Optimisation du compilateur**
  - **Avant :**

```
for (i = 0 ; i < N ; i = i + 1)
        for (j = 0 ; j<N; j = j + 1
                a[i][j] = 1/b[i][j] * c[i][j] ;
 for (i = 0 ; i < N ; i = i + 1)
        for (j = 0 ; j<N; j = j + 1
                d[i][j] = a[i][j] * c[i][j] ;
```

  - **Après :**

```
for (i = 0 ; i < N ; i = i + 1)
        for (j = 0 ; j<N; j = j + 1)
{
                a[i][j] = 1/b[i][j] * c[i][j] ;
                d[i][j] = a[i][j] * c[i][j] ;
}
```

Synthèse : Cache L1

- Le cache L1 est habituellement à l'intérieur du même circuit intégré que le microprocesseur, souvent imbriqué dans l'architecture même du processeur.
- Petit, car espace sur le microprocesseur limité (128 ko pour les premiers ATHLON, 32 ko pour les pentiums 2 et 3)
  - Divisé en deux pour les données et pour les programmes.
  - Très utilisé.

Synthèse : Caches L2, L3 et autres

- Les ordinateurs modernes ont au moins deux niveaux de cache et souvent trois.
- Le cache L2 est plus gros que la cache L1 (256 ko à 2 Mo).
- Les caches L2 et L3 sont habituellement à l'extérieur du microprocesseur (mais de plus en plus à l'intérieur). Indépendants de l'architecture du microprocesseur lorsqu'à l'extérieur.
- Habituellement, les données et les instructions sont gérées de la même façon dans les caches L2 et L3 (elles sont dites unifiées). Cependant, les instructions et les données sont de plus en plus souvent séparées dans les caches de niveau 2 (L2) voire de niveau 3.
- Moins rapide que L1 mais environ 10 fois plus rapide que la mémoire.
- Dans certains systèmes, il y a 4 niveaux de cache…
- Dans les ordinateurs modernes, il y a des caches dans les disques durs, et pour plusieurs périphériques. Ces caches contiennent les données du disque ou celles du périphérique qui les sont plus susceptibles d'être accédées prochainement

Synthèse : Organisation en niveaux

- Cache organisé en plusieurs niveaux :
  - Jusqu'à 3 niveaux (L1, L2 et L3)
    - De rares processeurs avec un L4
    - L3 généralement utilisé pour l'échange de données entre cores dans un processeur multi-core
    - L1 et L2 généralement spécifique à un core
    - Cache L1 généralement scindé en 2 parties : un dédié aux instructions et l'autre aux données
  - Cache L*i+1* joue le rôle de cache pour le niveau L*i*
  - Cache L*i+1* plus grand

Synthèse : Organisation en niveaux

- **Relation entre les niveaux de cache**
  - Cache inclusif
    - Le contenu du niveau L1 se trouve aussi dans L2
  - Cache exclusif
    - Le contenu des niveaux L1 et L2 sont différents
- **Cache inclusif**
  - Taille globale du cache : celle du cache L2
- **Cache exclusif**
  - Taille globale du cache : taille L1 + taille L2

Synthèse : Organisation en niveaux

- **Avantages et inconvénients de chaque approche**
  - Exclusif
    - Cache plus grand au total
    - L2 de taille quelconque
    - Mais doit gérer la non duplication des données : prend du temps, L2 moins performant
  - Inclusif
    - Cache L2 plus performant
    - Mais taille totale plus faible
    - Et contraintes sur la taille de L2 (ne doit pas être trop grand par rapport à L1)
  - Cache inclusif : historiquement le plus utilisé
    - Mais on trouve aussi des CPU à cache exclusif (AMD)

Synthèse : Performances

- **Choix de taille du cache et du nombre de niveaux**
  - Augmentation de la taille de la mémoire cache
  - Augmente le taux de succès
  - Mais ne gagne pas forcément en performance car augmentation du temps d'accès proportionnellement à la taille
- **Augmentation du nombre de niveaux (plus de 3)**
  - Pas de gain supplémentaire vraiment significatif
- **Cache inclusif**
  - Le cache L2 doit être bien plus grand que le cache de niveau L1 car sinon on stocke peu de données supplémentaires dans L2 par rapport à L1 et donc taux de succès de L2 faible
- **Ne pas oublier le coût élevé de ce type de RAM**

Synthèse : Conception de cache

- **Nécessité de déterminer des paramètres :**
  - Cache size
  - Block size (aka line size), *aka* = ***a****lso* ***k****nown* ***a****s*
  - Number of ways of set-associativity (1, N)
  - Eviction policy
  - Number of levels of caching, parameters for each
  - Separate I-cache from D-cache, or Unified cache
  - Prefetching policies / instructions
  - Write policy

Conclusion

- Performances globales de la mémoire cache dépendent de plusieurs facteurs corrélés
  - Taille de la mémoire cache et organisation des niveaux
  - Méthode d'association des lignes entre mémoire centrale et cache
  - Rapidité d'accès à la bonne ligne dans le cache
  - Méthode de remplacement des lignes
  - Algorithmes de *pre-fetching*
    - Chargement en avance dans le cache de données/instructions dont le processeur devrait avoir besoin
- But : un taux de succès global qui doit être le plus élevé possible tout en assurant un temps d'accès bas
  - Généralement autour de 90% de nos jours


Thank's

Exercice

- Décrire les étapes nécessaires pour que le microprocesseur écrive une donnée en mémoire si :
  - l'adresse mémoire n'est dans aucun cache;
  - Il y a deux niveaux de cache (L1 et L2);
  - le système utilise la stratégie "write-through".

QCM

- **Lequel des énoncés suivants est faux :**
  - Une taille de bloc optimale réduit le taux d'échec d'un cache.
  - Le programmeur peut contribuer à réduire le taux d'échec d'un cache.
  - Un cache dont l'associativité est plus élevée a un temps d'accès réussi plus bas.
  - Un cache totalement associatif de 1024 octets a un taux d'échec plus bas qu'un cache à correspondance directe de 512 octets à 2 blocs/ensemble.

QCM

- **Quelle est la pire chose qui peut arriver lors d'un accès à la mémoire**
  - Un miss du cache L1,
  - Un miss du cache L2,
  - Un miss de TLB,
  - Un accès à la table de page,
  - Un bit de validité à 0 dans la table des pages
  - Aucune de ces réponses

QCM

Quel type de cache est le mieux adapté pour un programme effectuant beaucoup de lecture séquentielle dans un énorme tableau?

a) Direct-mapped cache avec des blocs de 8 mots
b) Direct-mapped cache avec des blocs d'un mot
c) Fully-associative cache avec des blocs d'un mot
d) Fully-associative cache avec des blocs de 4 mots
e) Aucune de ces réponses

QCM

Quel type de cache est le mieux adapté pour un programme effectuant beaucoup d'écriture à répétition dans des emplacements aléatoires en mémoire?

a) Direct-mapped cache avec write-back
b) Direct-mapped cache avec write-through
c) Set-associative cache avec write-back
d) Set-associative cache avec write-through
e) Aucune de ces réponses

QCM

Comment un cache simule-t-il l'algorithme LRU?

a) Avec un compteur de signaux
b) Avec un bit de référence remis à 0 périodiquement
c) Avec un circuit logique
d) C'est le système d'exploitation qui le gère pour le cache
e) LRU n'existe pas dans un cache
f) Aucune de ces réponses