Plan du chapitre

1. *Les caches.*
2. *Organisation des caches et accès en lecture et écriture.*
3. *Exemples de caches.*
4. *Optimisation des caches.*
5. *Autres caches.*
6. *Conclusion.*

LES CACHES

1. Rappels sur les *mémoires associatives* souvent utilisées dans les caches.
2. Structures des caches, leurs techniques de gestion et des exemples.

- Une donnée est désignée par ***son adresse*** dans la mémoire de Neumann.
  - Les circuits ont alors accès au « contenu » de l'adresse.

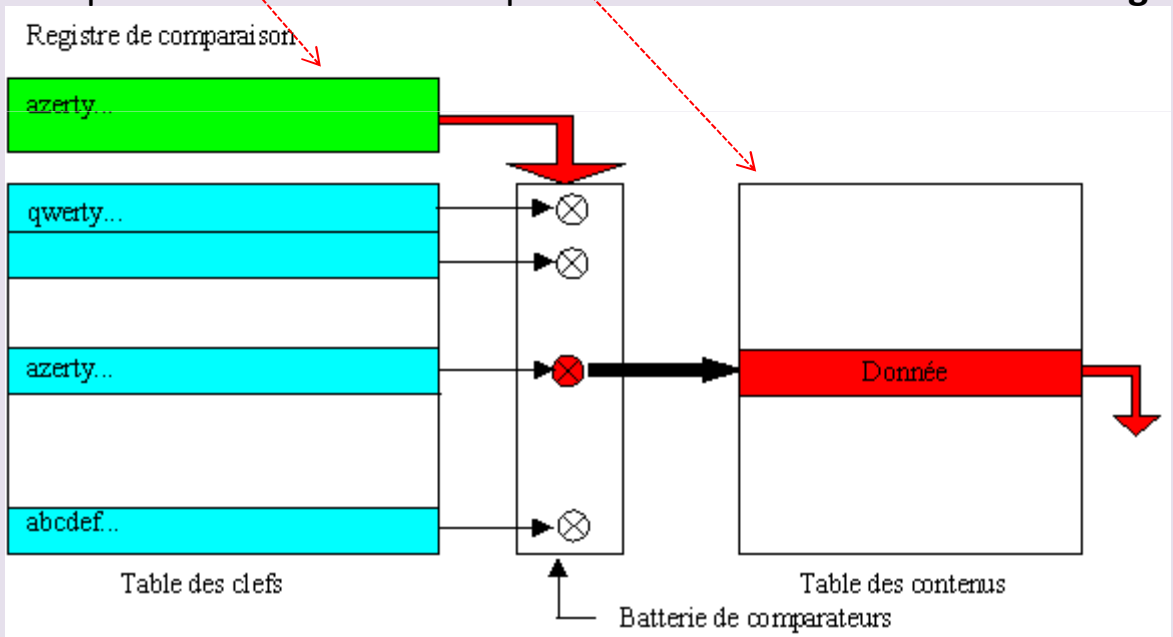
  ⇨ Si l'on cherche une valeur prédéterminée, **sans connaître son l'adresse**, il faut faire un balayage et des comparaisons successives pour trouver sa localisation et donc sa présence.

  ⇨ Pour faire directement ce type de recherche, on a imaginé des mémoires ***dites associatives.***

  → Leur entrée est un contenu et leur sortie est soit rien soit la donnée associée, adresse, pointeur ou autre si le contenu a été trouvé.

- **La *mémoire associative* ou *mémoire accessible par le contenu* (content adressable memory) est organisée en lignes,**
- **Chaque ligne est divisée en deux parties ou colonnes :**
  1. La colonne des **clefs** («tag» en anglo-saxon) contient la valeur, sujet de la comparaison
  2. L'autre colonne contient la **donnée associée** à la clef

  → Un ou des comparateurs servent à comparer le contenu fourni dans un « **registre** » et les clefs



PRINCIPE DE LA MÉMOIRE ASSOCIATIVE

- La comparaison entre le contenu du registre et les clefs est faite sur toutes les clefs **en une seule fois.**
  - → Cette recherche simultanée nécessite un câblage particulier et un jeu de comparateurs de la largeur de la clef pour chaque ligne.

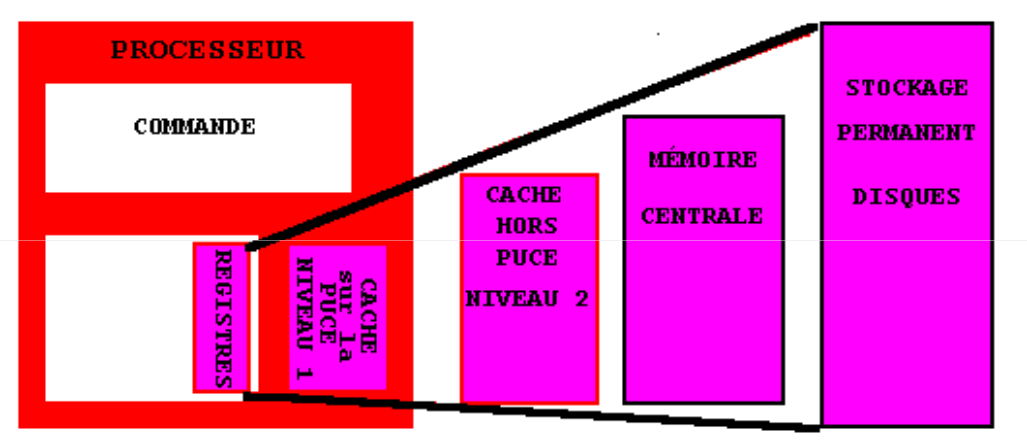
- Une mémoire associative est chère.
  - Son utilisation est limitée pour les caches, pour les tables en un coup d'œil ou TLB (*Translation Lookaside Buffer)*

Mémoire cache ou antémémoire

- En informatique un cache est une mémoire ***plus petite qu'une autre***

  et qui contient une ***partie des données de cette dernière***

- La figure suivante présente une hiérarchie de mémoires aujourd'hui courante



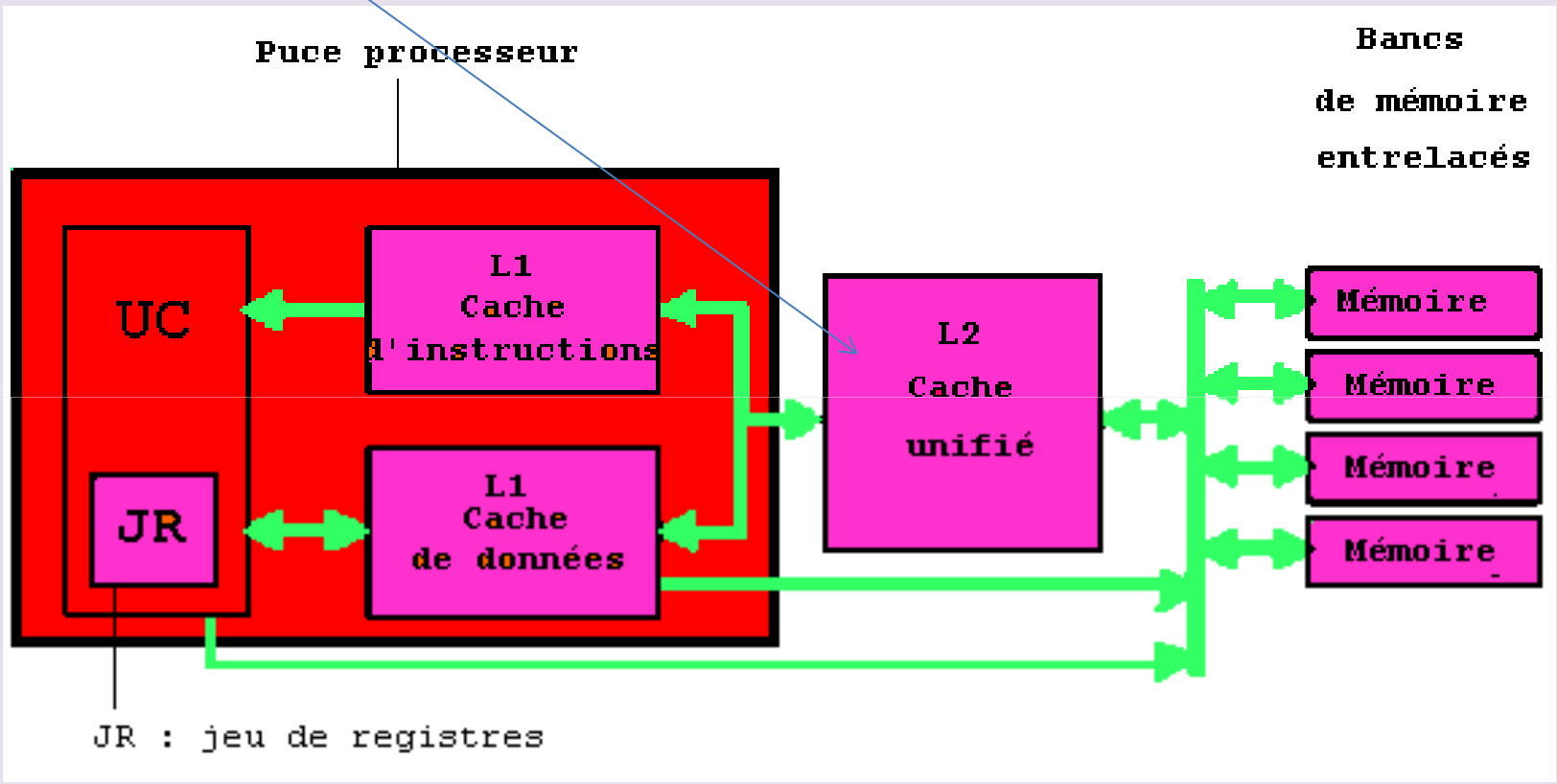
Désignation	Taille	Temps d'accès	Gestionnaire
Registres	32-512 octets	1-2 ns	compilateur
Cache niveau 1	1-16 ko	3-10 ns	matériel
Cache niveau 2	64-512 ko	25-50 ns	matériel
Mémoire (primaire)	16Mo-1 Go	60-250 ns	matériel + système
Disque (mémoire secondaire)	1Go-1 To	5-20 ms	système
CD-Rom (mémoire tertiaire)	600 Mo	100-500 ms	système
Bande (mémoire tertiaire)	très grande	1s-10 minutes	système

- **Remarque :** L'utilisation de caches est d'ordre général, par exemple :
  - → *cache de mémoire*, pour une partie de la mémoire centrale;
  - → *mémoire virtuelle*, pour une partie du système de fichiers du disque;
  - → *cache de disque*, pour une ou des pistes du disque;
  - → *tampons de translation*, pour la page des tables;
  - → *les marques de branchement*, pour les comportements précédents;
  - → *le cache de l'explorateur*, pour des pages du web;
  - → etc.

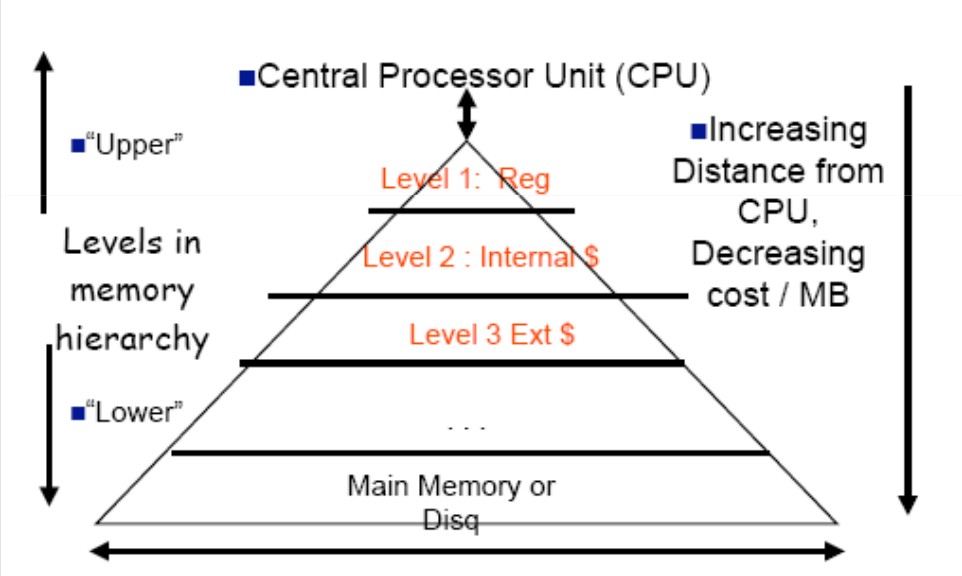
Rappel :

- ***Un cache unique*** peut se trouver dans le même circuit que le processeur.
- ***Deux caches*** peuvent y être.
  - Exemples :
    - → Comme dans l'Alpha 21164 où ils occupent 70% de la surface de la puce
    - → Dans les Pentium 4 et Xéon qui contiennent
      - ↗ 12 K de cache de micro opérations
      - ↗ 8 Ko de cache de données pour L1
      - ↗ 256 Ko de cache L2.
- S'il y a trois caches, on les distingue par les dénominations :
  1. ***Interne*** au `circuit processeur` ou primaire ou *de* ***premier niveau ou L1***;
  2. ***Interne*** ou ***externe*** au `circuit processeur`, ou secondaire ou de ***second niveau ou L2;***
  3. ***Externe*** ou de ***troisième niveau L3***.
- Le cache L1 est très souvent constitué de deux mémoires spécialisées accessibles simultanément (cache instruction et cache de données) (⇔ figure page suivante)

- Un cache unique peut se trouver dans le même circuit que le processeur.



- **La mémoire devient le frein à l'augmentation des perf des processeurs:**
  - Dans un système conventionnel, il n'y a pas une seule mémoire, mais plusieurs, Organisées en hiérarchie.
  - Il existe un protocole pour transférer les données d'un niveau à l'autre. Ce protocole peut être
    - ⇨ **Explicite** (***Load et Store***)
    - ⇨ **Implicite** (***ex : Défaut de Cache***)



Taille de mémoire à chaque niveau

➔ **Mémoire cache :**

- **Objectif : Donner à l'utilisateur l'illusion d'avoir la capacité de la mémoire la moins chère (Disque) et la vitesse de la plus rapide (cache interne)**

- ***Constat:*** Mémoires larges sont lentes
    - Mémoires rapides sont petites.

- **Question :** Comment créer des mémoires de grandes tailles, rapides et pas chères?
- ***Réponse :*** Hiérarchie de plusieurs niveaux de mémoires pouvant fonctionner simultanément.

- **La mémoire cache :**

➢ ***Un cache : contient*** une copie de certains blocs de données présents en mémoire centrale (DRAM).

➢ ***Cache Vs DRAM :*** Rapide(10X) mais plus petite (100X)

➢ ***Un bloc =*** Une ligne de cache = plusieurs mots mémoire consécutives en mémoire.

➢ ***Toutes les opérations mémoires (lecture et écriture) sont dirigés d'abords vers le cache pour tester si la donnée est présente.***

➢ ***Deux cas possible :***

1. ***Donnée présente*** ⇔ ***Succès ou hit***, la donnée est délivrée au processeur
2. ***Donnée absente*** ⇔ ***défaut ou miss***, la donnée est transférée vers le cache puis délivrée vers le processeur.

➢ **Taux de succès (hit ratio, noté h) : Fraction des accès au cache avec succès.**

$$h = \frac{\text{nombre de succès}}{\text{nombre d'accès total}}; \qquad 0 \le h \le 1$$

- **Le Cache fournit de façon transparente les données qui ont été chargées dans le passée ⇔ Données déjà utilisée dans un passée proche.**
- **Transparente ? ⇨ sans que le programmeur le demande.**
  - ☞ On paye une seule fois la *pénalité de la lenteur mémoire,* même si la données est utilisée plusieurs fois.
- ***Observations importantes:***
  - ✓ Les références aux données et surtout aux instructions ne sont pas, d'habitude, *indépendantes*.
  - ✓ Les programmes ont tendance à réutiliser les données et les instructions qu'ils ont *utilisées récemment*.

1. ***Localité spatiale:*** les éléments dont les adresses sont **proches** les unes des autres auront tendance à **être référencés** dans un **temps rapproché (**p.ex. instructions, images).

   ⇨ ***Conclusion :*** **Il faut charger de la DRAM non seulement la donnée dont on a besoin mais les données voisines : un bloc de plusieurs données (exemple 4 mots mémoire)**

2. ***Localité temporelle:*** les éléments auxquels on a eu accès récemment seront probablement accédés dans un futur proche (p.ex. boucles).

   - **Il faut retenir (stocker) une donnée qui a été récemment utilisée.**

- La suite présente les architectures et modèles de fonctionnement usuels des caches en répondant aux trois questions relatives à leurs techniques opératoires:

1. ***Les techniques de placement et d'accès*** :
   - **où placer les données venant de la mémoire** dans le cache et **comment le processeur** y trouve-t-il l'information ?
2. ***Les techniques de remplacement***,
   - **où et comment faire de la place** dans un ***cache plein ?***
3. ***Les techniques d'écriture*** :
   - **comment écrire un résultat** dans ***le cache, la mémoire, les deux ensemble ?***

- **Ces trois questions ne *sont pas indépendantes*, les *réponses non plus.***

- **Quand la donnée à lire est dans le cache, quelque soit le cache dont on parle, il y a *succès* ou *réussite* (*hit*), sinon il y a *échec* (*miss*).**

- **En cas d'échec, le bloc de données est lu dans la mémoire d'ordre supérieur, cache ou mémoire centrale et amené dans le cache en cause**

- **La performance apportée par le cache dépend :**
  - ↗ De la probabilité de succès **S** de lecture dans le cache;
  - ↗ Du temps d'accès en ***lecture dans le cache***, tout compris, en **T1** cycles;
  - ↗ Du temps de ***transfert du bloc de la mémoire vers le cache***, en **T2** cycles;
  - ↗ Du temps de lecture d'une donnée en ***mémoire, sans cache***, en **T3** cycles.

- Le temps moyen d'accès T_{moyen} est alors :

$$T_{moyen} = S \times T_1 + (1-S) \times (T_2+T_1)$$

- Le facteur de temps (accélération) est $A = T_3 / T_{moyen}$

- Exemple : pour :
  - **90 % de succès;**
  - **l'accès au cache en 1 cycle;**
  - **la lecture du bloc en 20 cycles (⇔ transfert de la mémoire vers le cache);**
  - une lecture directe, sans cache, en mémoire en 5 cycles,
    - $S = 0,9$ $T_1 = 1$ $T_2 = 20$ $T_3 = 8$

$$T_{moyen} = 0,9x1 + 0,1x21 = 3$$

  - le temps d'accès moyen à une donnée est de 3 cycles.
  - Le facteur de temps (accélération) est **A** = T_3 / T_{moyen} **= 8/3**

♦ Efficacité d'un cache :

- **Soit :**

  →**CI** = Nombre d'instructions

  →**Ref_ Mem** = Nombre moyen de références à la mémoire **d'une instruction**

  →**Tx_échec** = Fraction des accès pour lesquels l'information n'est pas dans le cache

  →**Pénal_échec** = Temps nécessaire pour amener l'info en cas d'échec

- **Alors :**

$$Cycles\_perdus = CI * Ref\_Mem * Tx\_échec * Pénal\_échec$$

♦ Un cache est fait pour :

1. **Contenir des octets de données,**

   → l'information additionnelle de position, c-à-d les adresses, doit **être la plus petite possible**, cette dernière est nommée **répertoire du cache** (adresse complète ou partielle);

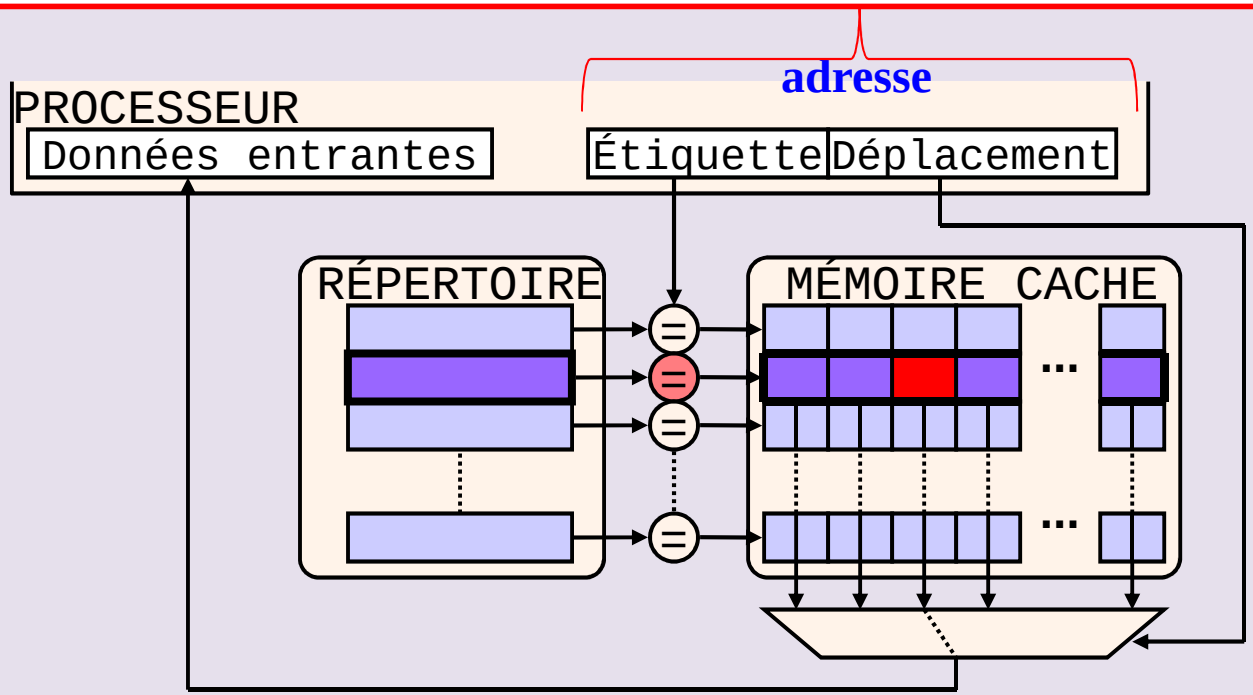
2. **Avoir le meilleur taux de réussite;**

   → pour l'augmenter, **le répertoire doit être grand**.

Tag (Etiquette) : Identificateur d'un bloc

- *Un bloc de cache* : groupe de plusieurs éléments consécutifs en mémoire (exemple 4 mots). ***C'est l'entité de donnée gérée par le cache***.
- Le cache et la mémoire sont divisés en blocs de même taille.
- *Exemple :*
  - ***Taille de bloc*** = 16 octets (4 mots : **4*4**)
  - ***Cache*** = 16 blocs (4*4*16 = 256 octets)
  - ***Mémoire*** = 128 blocs (**4*4***128 = 2KO)
- *Associé à un bloc :*
  - **Etiquette « Tag » :** une partie de l'adresse, mémorisé pour savoir ***si le bloc est présent.***
  - **Bit Valid** :
    - Si =1, alors bloc occupé,
    - Si = 0 alors bloc vide (libre)
  - **Bit Modifié**:
    - Si =1, alors le bloc **a subi une écriture,**
    - Si =0, alors le bloc **n'a jamais été modifié depuis sa lecture de la DRAM**.

♦ **Exemple :**



- **Un répertoire est associé à chaque cache associatif.**
  - **Le répertoire contient le numéro (adresse) de chaque bloc dans le cache, appelé étiquette.**
  - **Toutes les étiquettes dans le répertoire sont *examinées en parallèle.***
- **Pour un cache *totalement associatif*, chaque adresse contient donc une étiquette et un déplacement dans le bloc.**

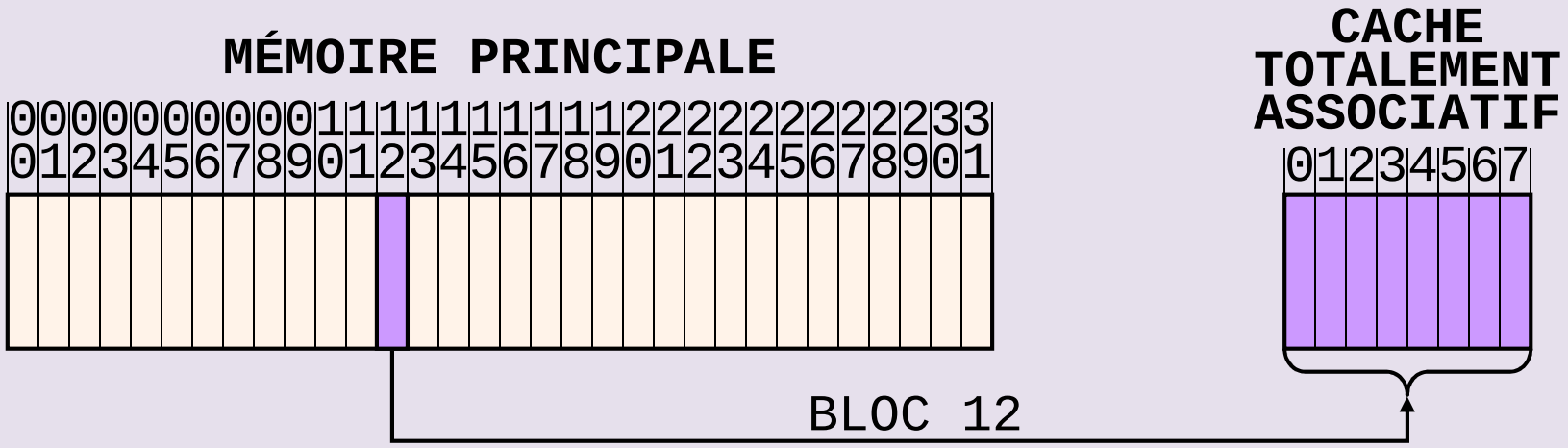
ORGANISATION DES CACHES ET ACCÈS

« EN LECTURE ET ÉCRITURE »

- *Le cache est une mémoire associative dans laquelle :*
  - *la **clef** est **une adresse;***
  - *le résultat cherché est nommé **granule** ou ligne de données dans la mémoire associative.*
- *Dans la suite on suppose un adressage par **octets.***
- *L'adresse émise par l'unité d'adressage est comparée simultanément avec toutes les adresses des informations rangées dans le cache.*
  - ***Cette réalisation complète nécessite un jeu de comparateurs pour chaque ligne**.*
  - ***Si l'adresse est trouvée**, l'information est lue dans le cache.*
  - ***Si non**, la ligne de mémoire ou granule qui la contient est chargée à partir **de la mémoire.***

Définition :

- **Si un bloc peut être placé n'importe où dans le cache, celui-ci est appelé totalement associatif (*fully associative*).**
- **Ce type de placement est le plus efficace, mais aussi les plus compliqué à mettre en œuvre, nécessitant une logique de contrôle très complexe.**

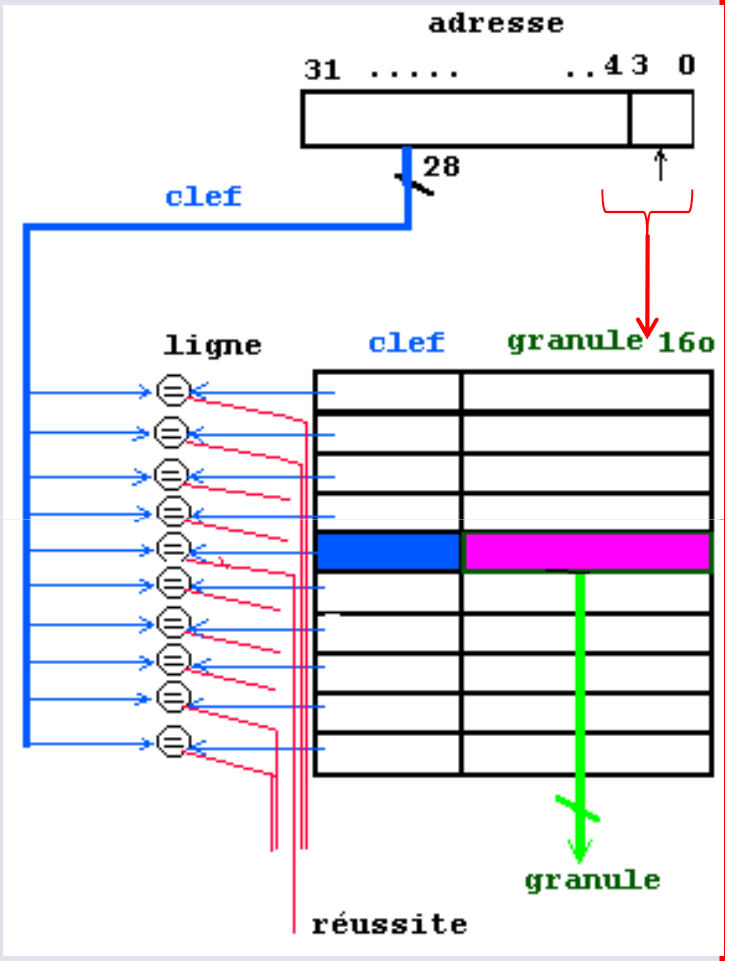


♦ ***Le schéma de principe de ce cache est :***

→ *Les quelques bits (ici 4) de poids faible sont* ***l'adresse de l'octet dans la ligne****.*

→ *Le principe même de ce cache impose autant de jeux de 28 comparateurs binaires que de lignes du cache.*

→ *La réalisation de ces comparateurs en grand nombre est souvent estimée* ***trop coûteuse.***



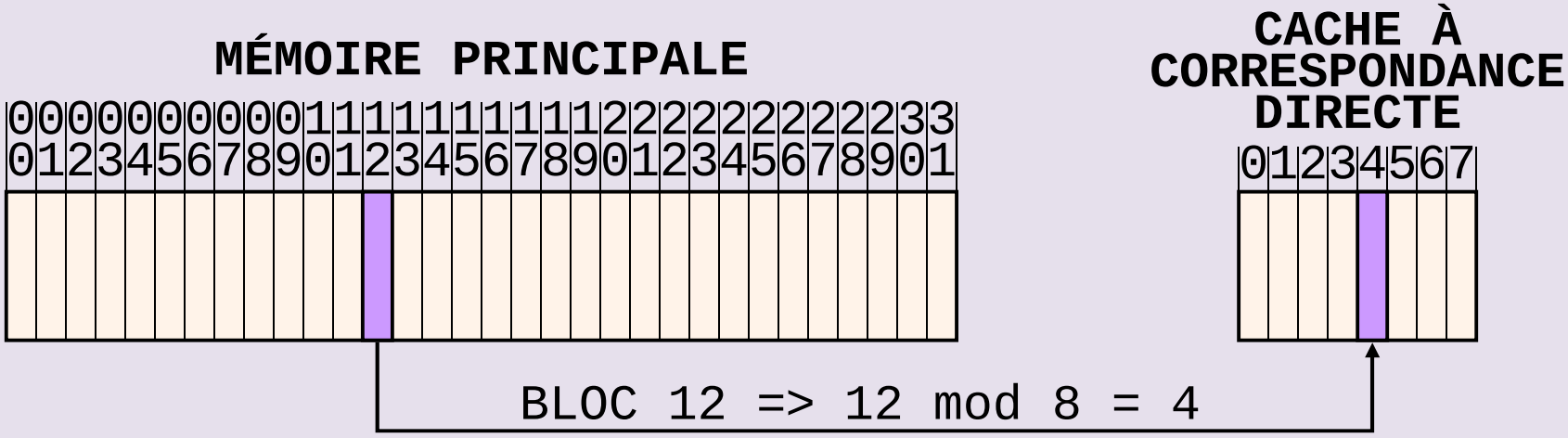
Définition :

- **Si chaque bloc a uniquement une seule place possible dans le cache, celui-ci est appelé à correspondance directe (*direct-mapped*).**
- **La correspondance est généralement la suivante:**

> **(*numéro de bloc)* mod (*nombre de blocs dans le cache)***

Remarque :

- **Cette stratégie de placement est la plus simple à mettre en œuvre, mais aussi la moins performante.**



Placement de bloc - Stratégie II – correspondance directe (direct mapped cache)

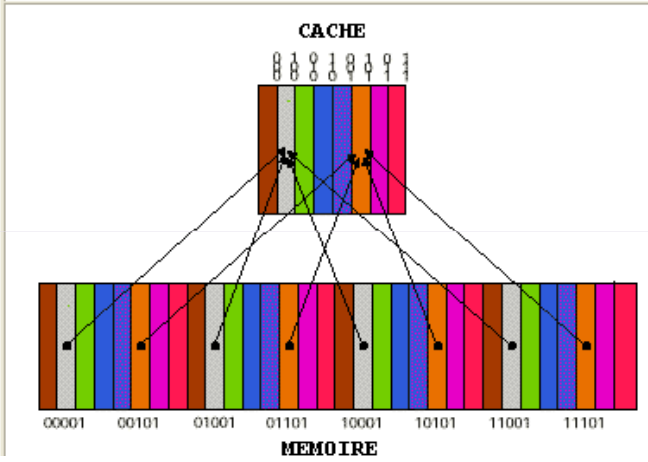
- **Une adresse en mémoire est alors considérée comme formée de trois parties :**

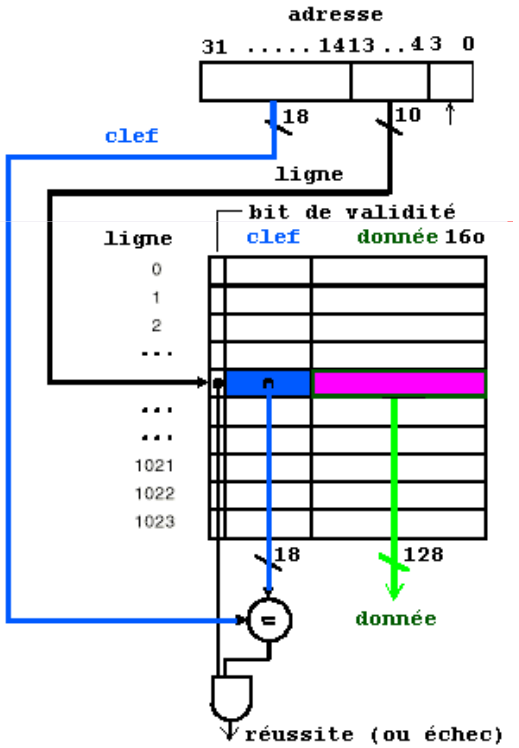
1. *la partie haute, **numéro de bloc en mémoire « tag ; étiquette »;***
2. *la partie intermédiaire, **numéro de ligne dans le cache (⇔ index ou position : 1 à 1023);***
3. *la partie basse, **numéro dans la ligne du cache (1 parmi 16 o).***

adresse physique	ligne du cache	rang en ligne	numéro de bloc
2896	181	0	0
441931	996	11	26
1679053	492	13	102
etc..			

correspondance entre les 8 lignes d'un cache et les blocs de mémoire. Un cache de 1024 lignes n'est pas représentable avec un code de couleurs.

correspondant à un cache réel de 16k octets en 1024 lignes de 16 octets.





2896 = 00000000000000000000101101010000

N°bloc « tag, étiquette » = 0

Ligne (index) =181 ⇔ 010110101

Rang en ligne = 0

32 blocs

441931 =00000000000011010111110010011011

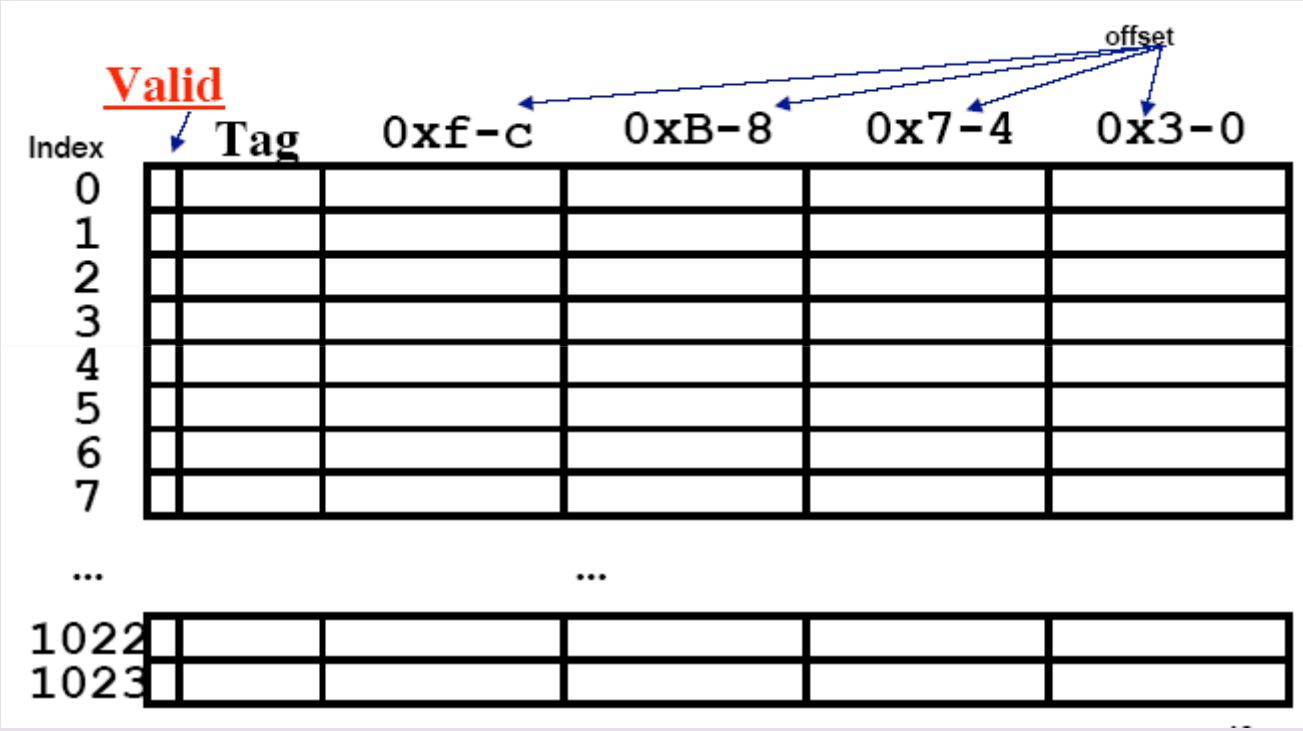
N°bloc « tag, étiquette » = 26 ⇔ 00000000000011010

Ligne « index » = 996 ⇔1111100100

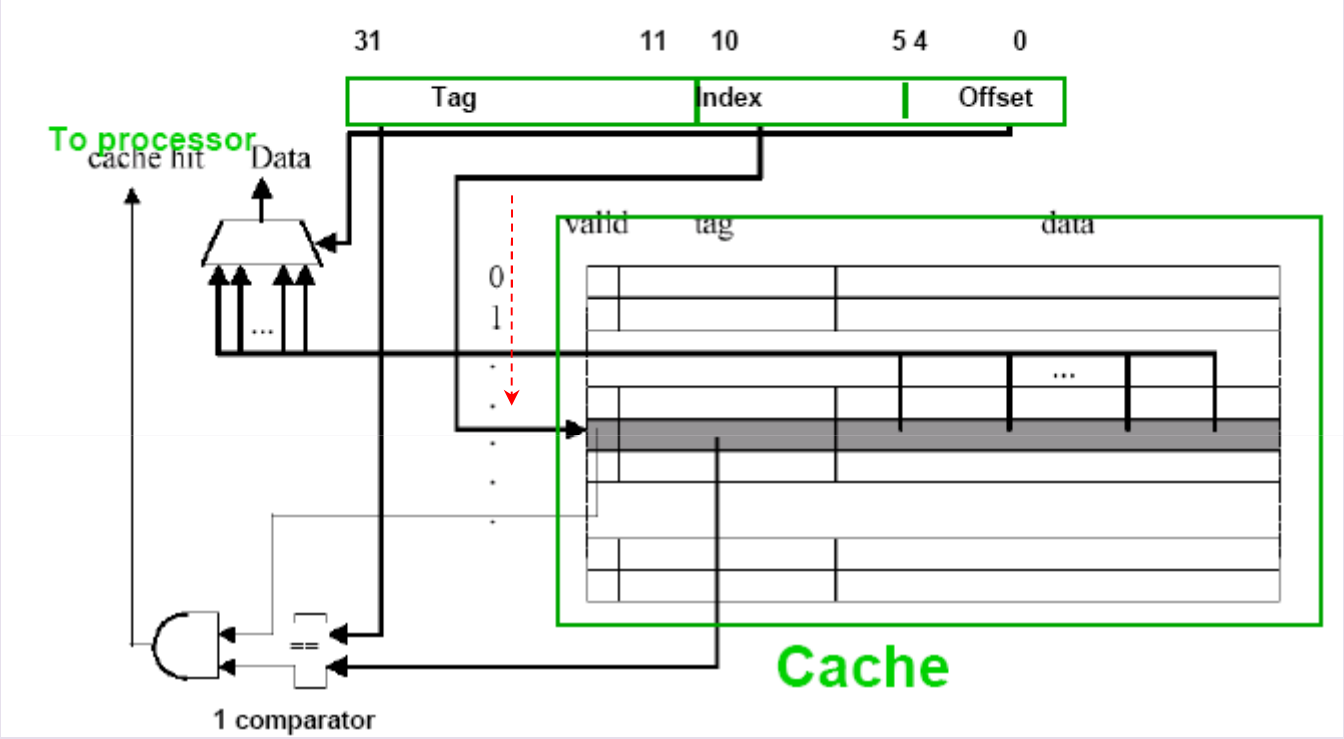
Rang en ligne =11 ⇔ 1011

Le N°de bloc « tag; étiquette » doit être stocké pour ne pas confondre les blocs.

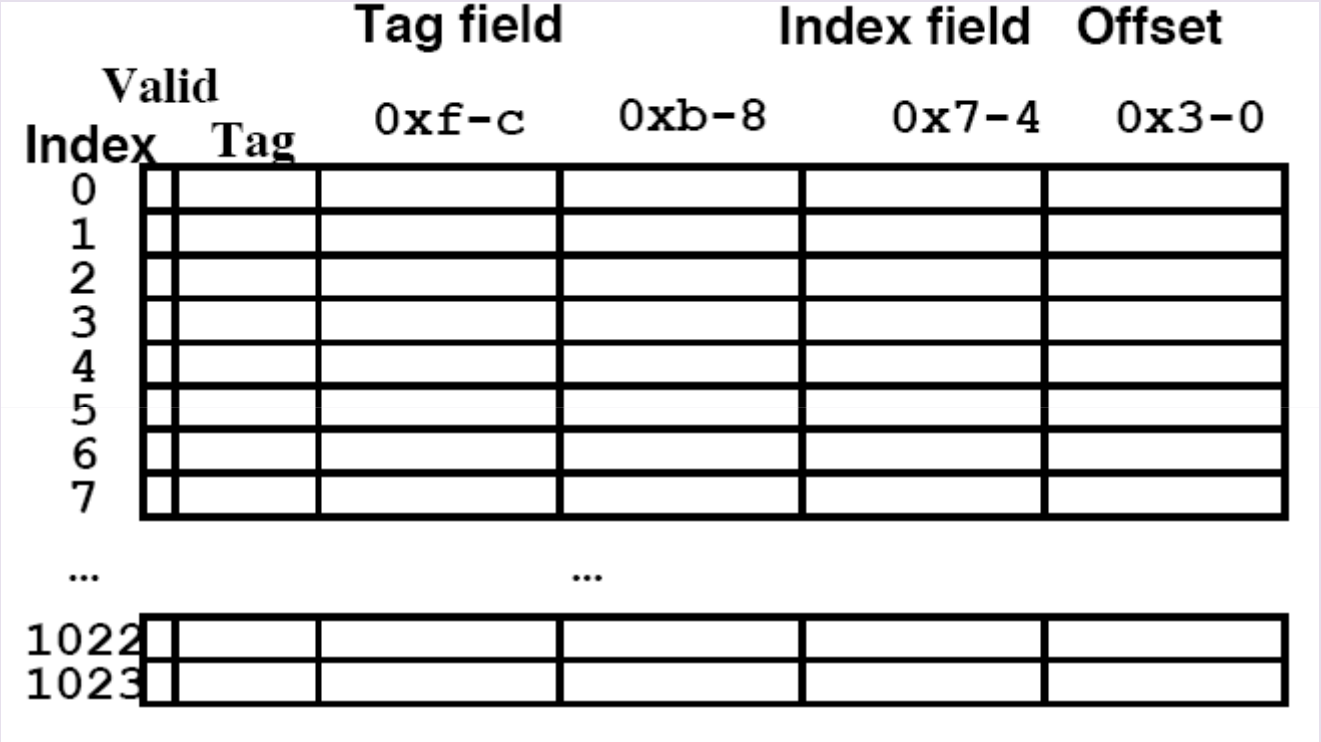
- Cache de 1024 ligne de 4 octets (⇔ 6 KB)
- 4 octets par ligne



A l'intérieure de la cache

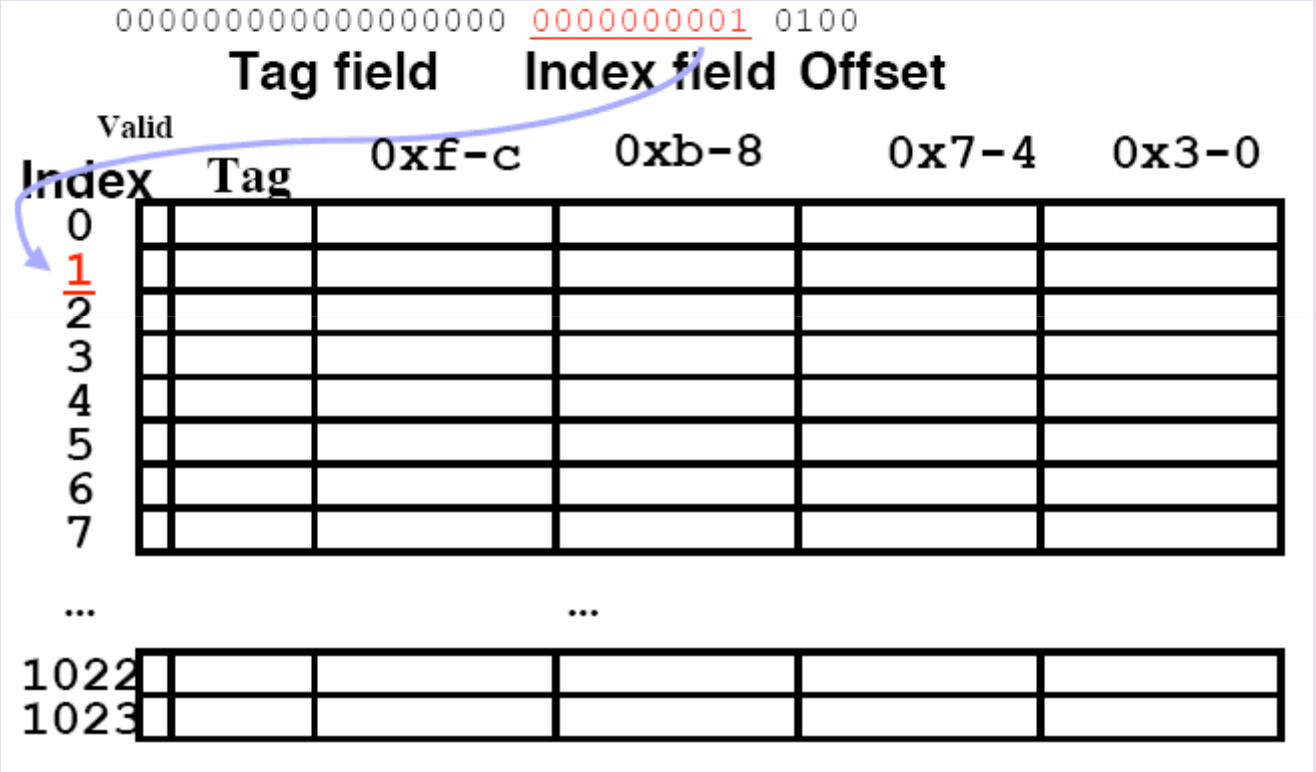


- Lecture du premier bloc:
  - 00000000000000000 0000000001 0100

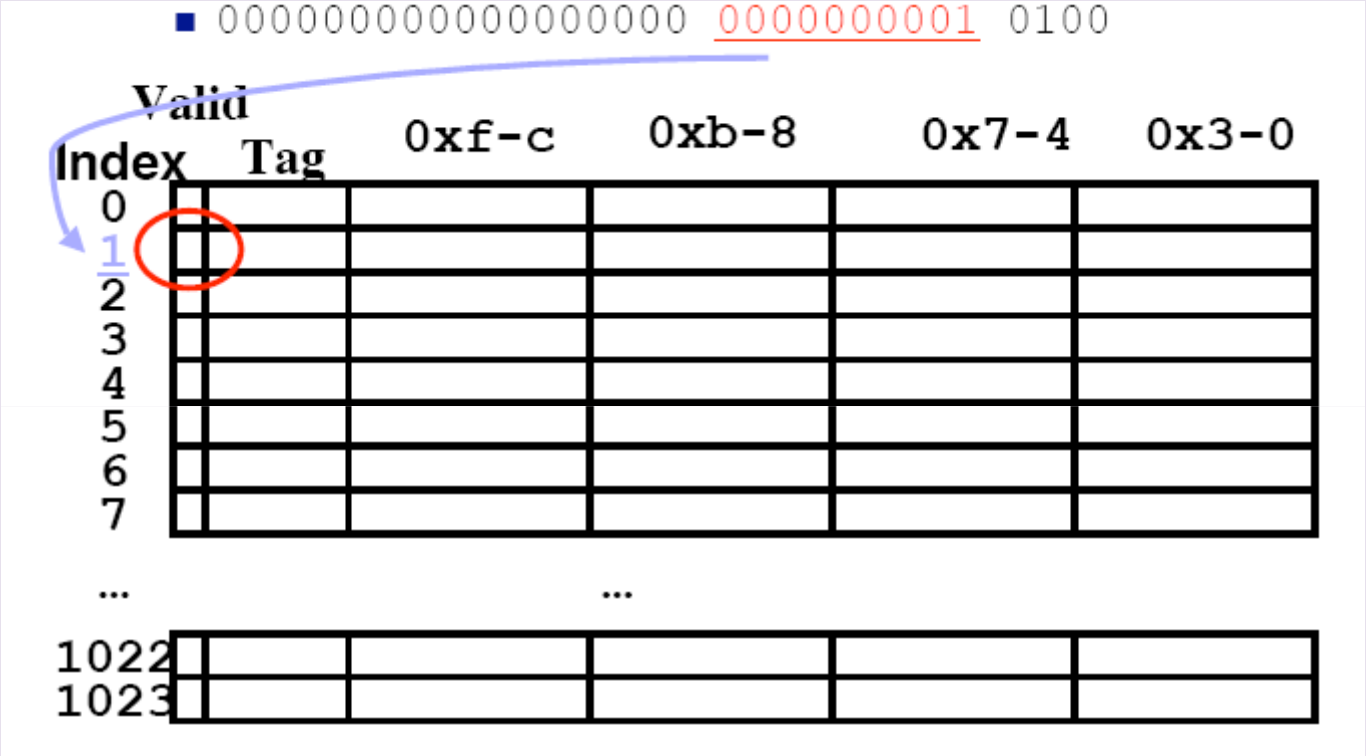


Lecture du bloc 1 (0000000001)

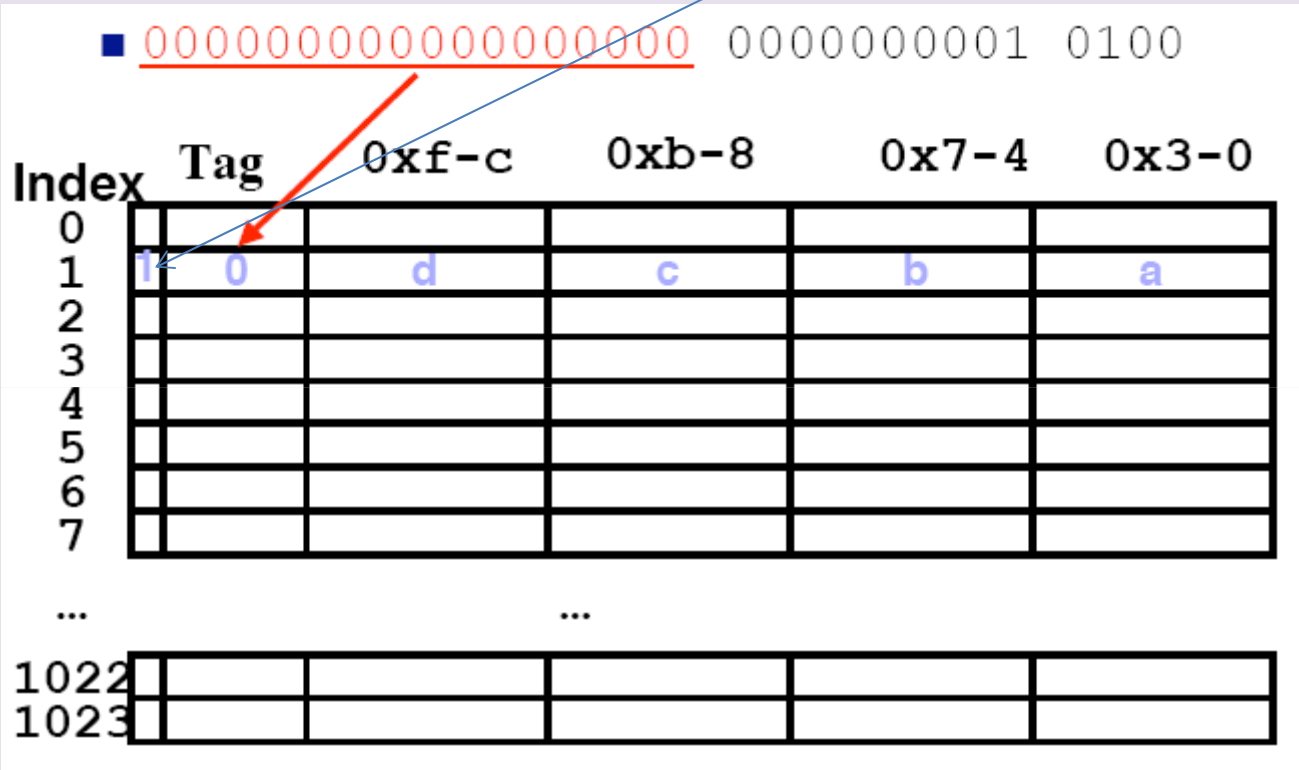
0000000000000000 0000000001 0100



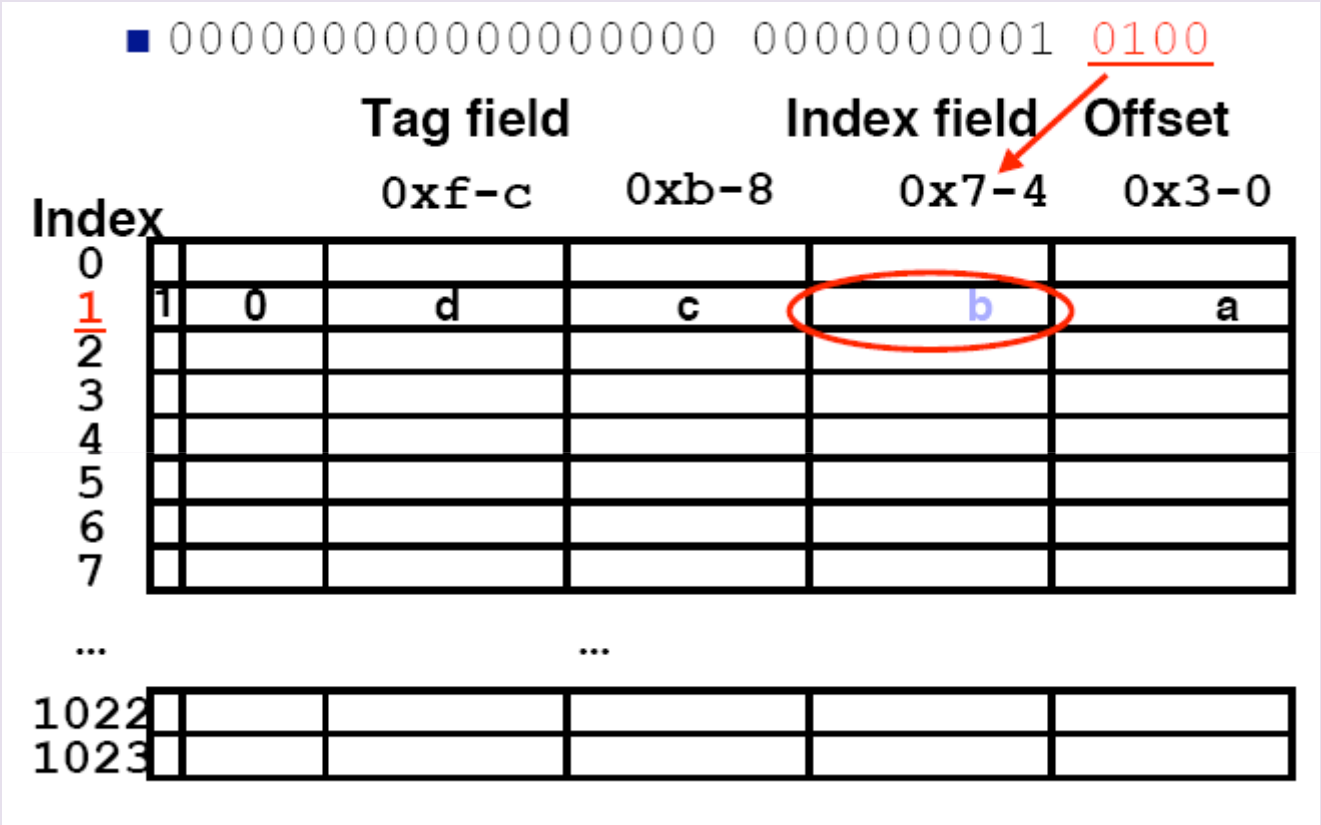
- Données non valide => Miss (cold)
  0000000000000000 0000000001 0100



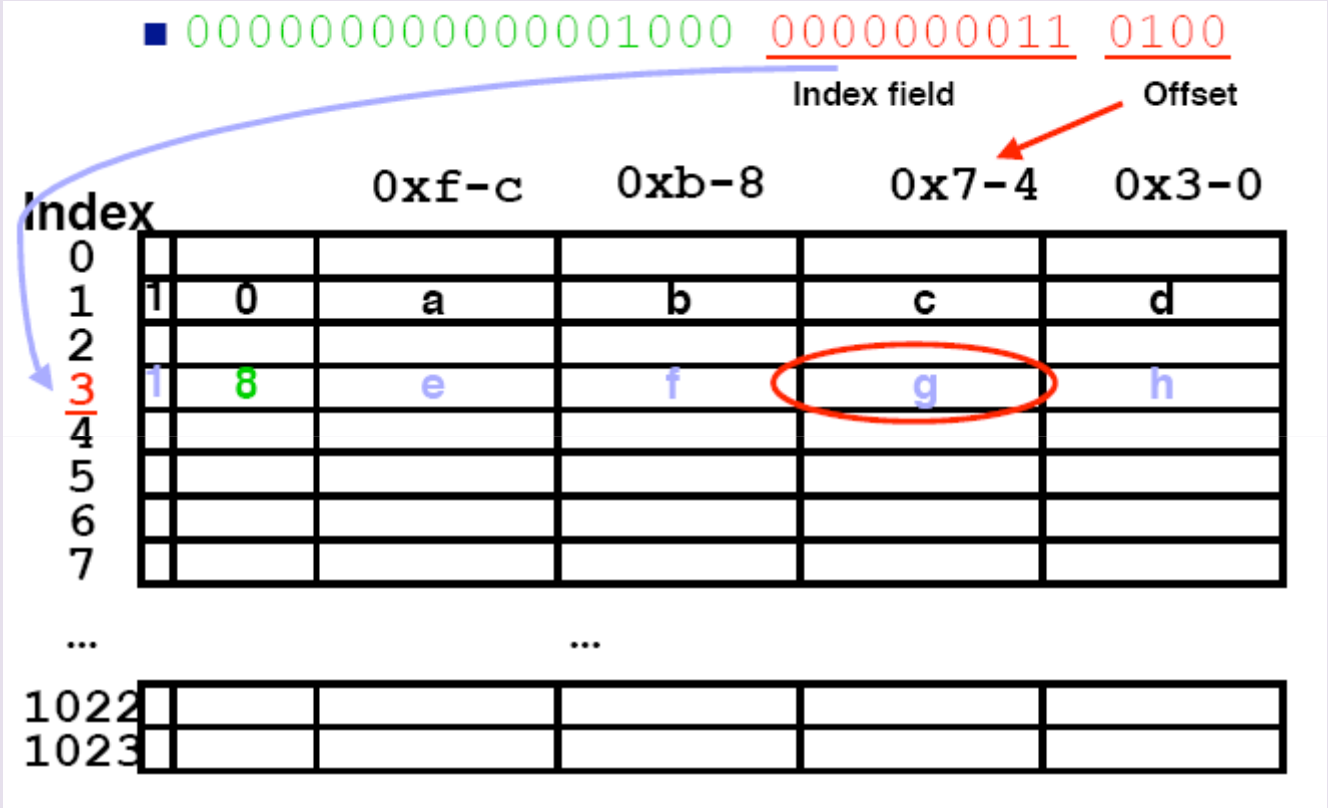
- Chargé la donné en cache, positionné tag & valid
  - 00000000000000000 0000000001 0100



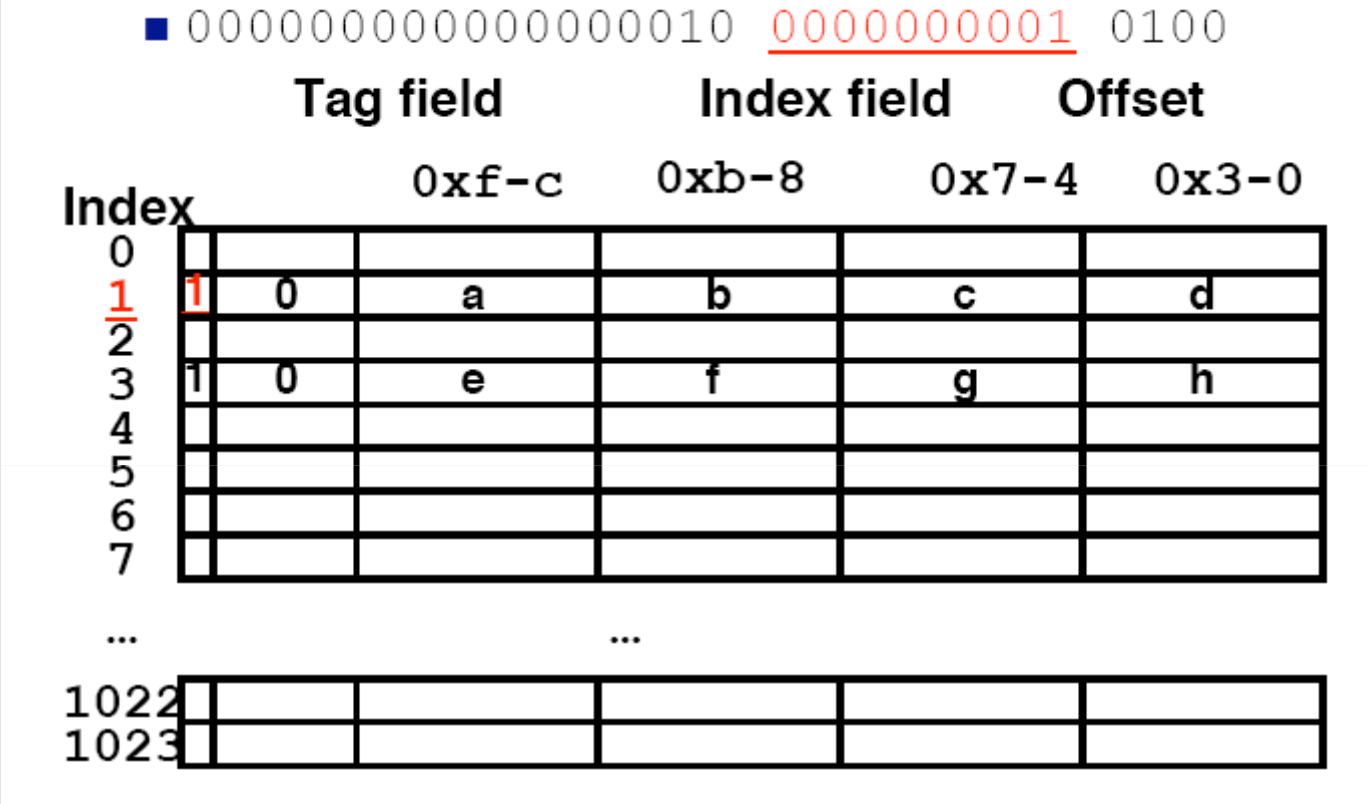
- Lecture depuis l'offset, renvoyer le mot b



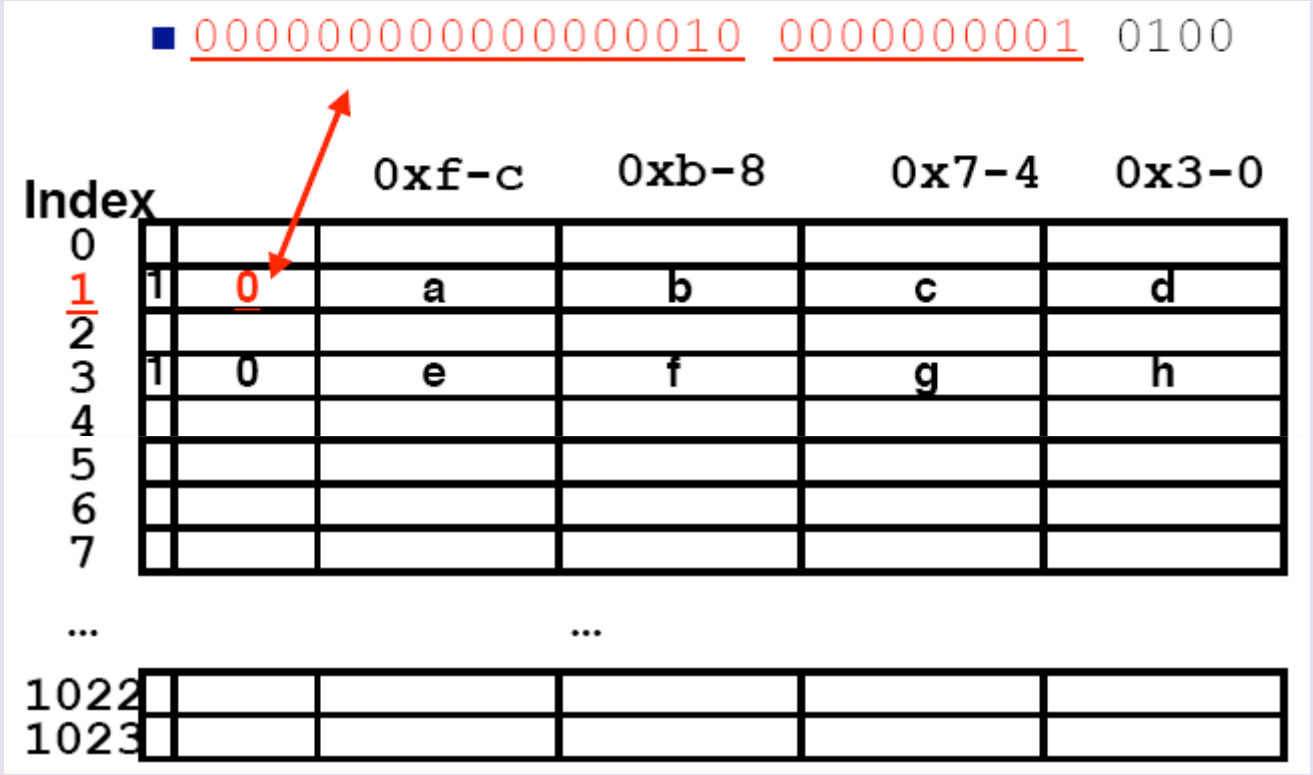
- Nouvelle lecture (No 2)



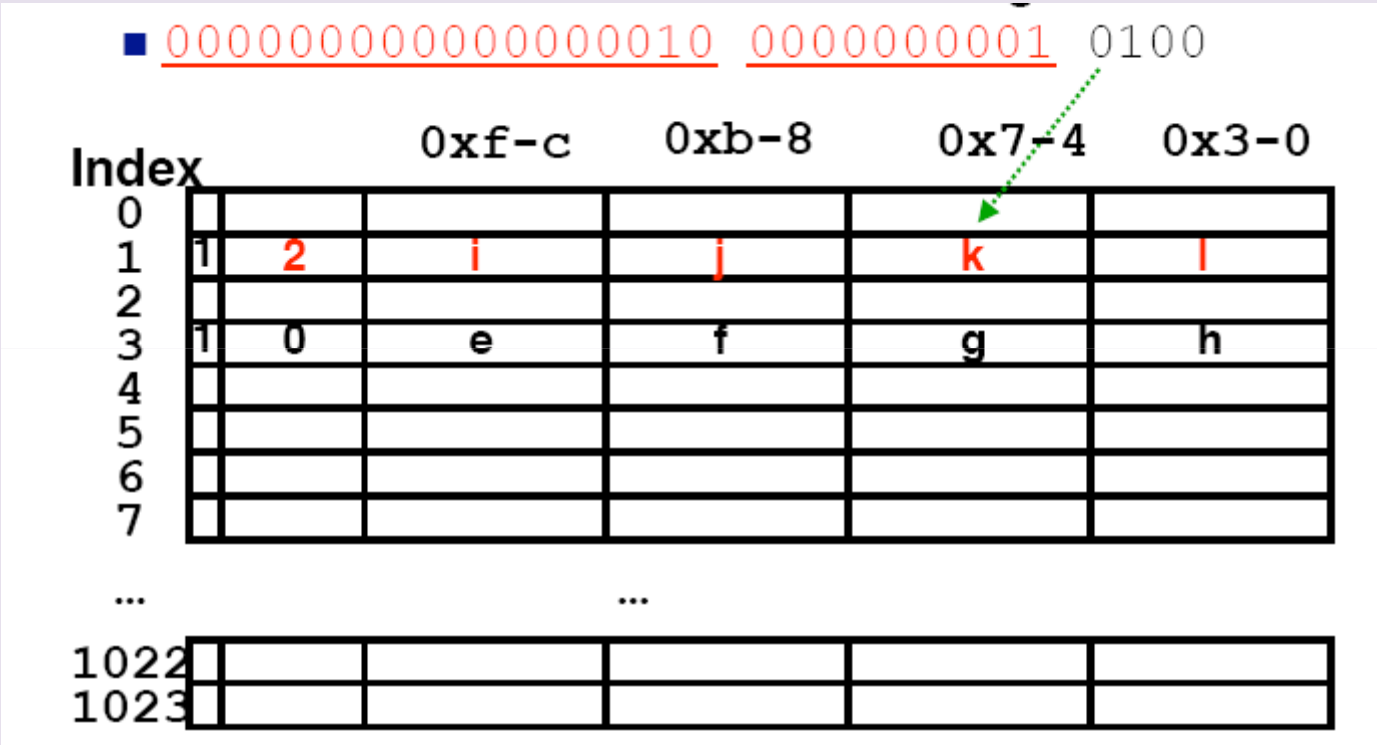
- Nouvelle Cache No 3, Block 1, Data is Valid



- Le tag du Block 1 ne convient pas !!! (0 != 2)



- Miss, remplacer block 1 avec nouvelle
  - données data & tag



- Les caches à accès direct permettent de s'affranchir du coût des ***N* comparateurs** indispensables dans le cas des caches associatifs.
- L'accès à chacune des lignes se fait à partir d'un ***index*** constitué par ***p*** bits de l'adresse émise

- *Cette organisation est intermédiaire entre les deux précédentes.*
  - L'accès totalement associatif est très coûteux quand le nombre de lignes est grand.
  - L'accès direct est incomparablement moins coûteux mais n'utilise pas au mieux l'espace de mémoire du cache.

- **Dans l'organisation associative par ensemble de blocs,**
  - Le cache est divisé en **sous ensembles** d'emplacements ou blocs.
  - La question est : que va-t-on faire dans chaque bloc et comment relier ces blocs entre eux ?
  - **Deux variantes** existent dans lesquelles **« chaque bloc est géré » :**

    → comme une mémoire associative et la relation entre les blocs relève **de l'accès direct (rare);**

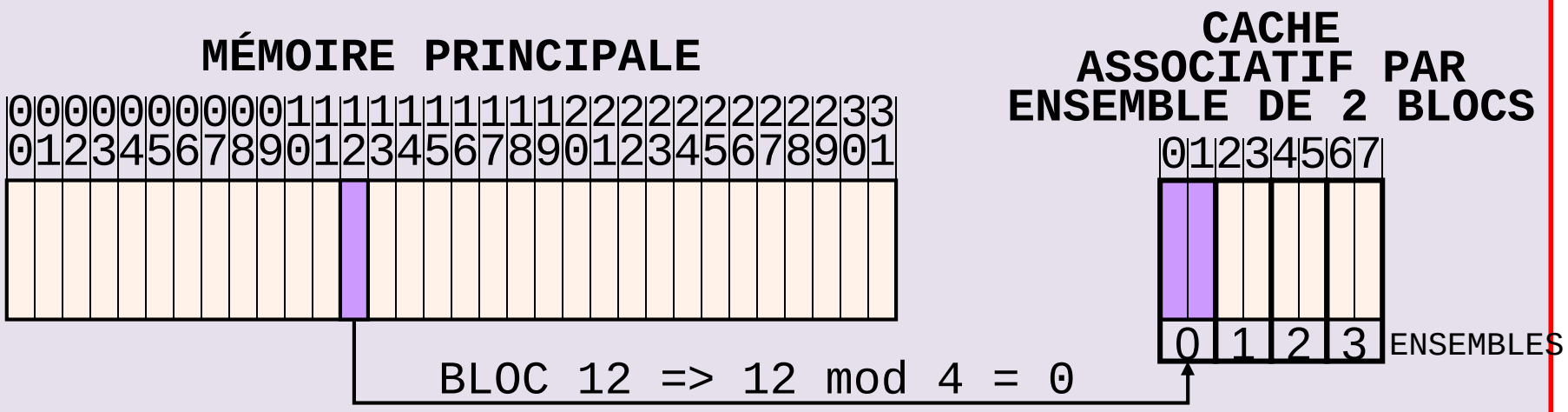
    → **en accès direct** et on utilise **l'accès associatif entre les blocs**.

Placement de bloc - Stratégie III – cache associative par ensemble de blocs (set associative)

Si un bloc peut être placé dans un ensemble restreint de places dans le cache, celui-ci est dit associatif par ensemble de blocs (*set-associative*).

- Un ensemble est un groupe de blocs dans le cache.
- Le bloc est placé n'importe où dans un ensemble habituellement choisi par:

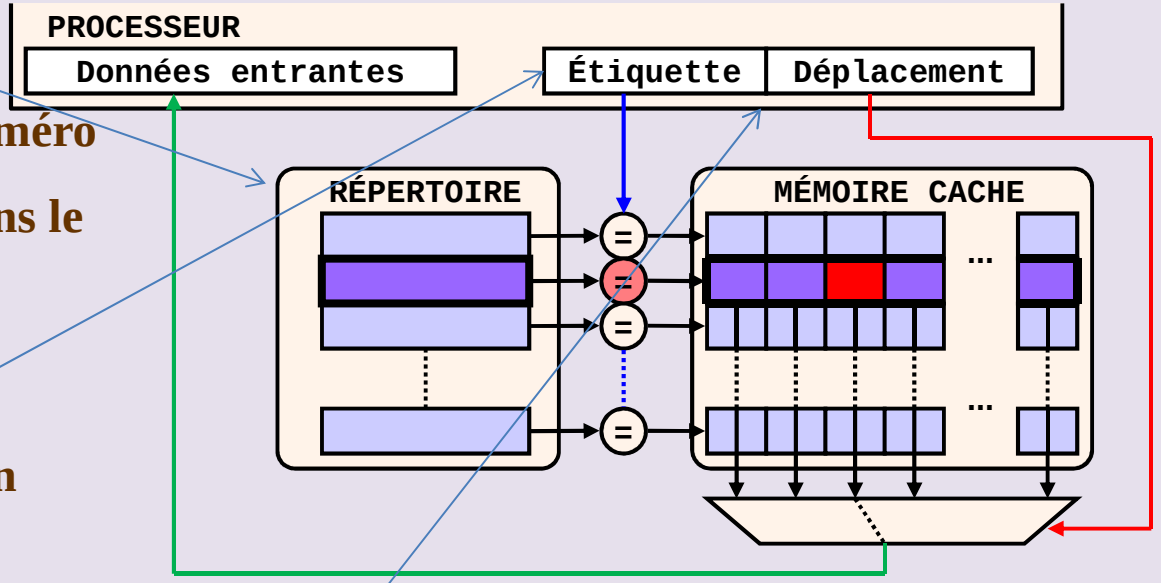
numéro de bloc* mod *nombre d'ensembles dans le cache



Ces caches demandent une quantité de logique de contrôle proportionnelle au nombre d'ensembles.

Identification de bloc

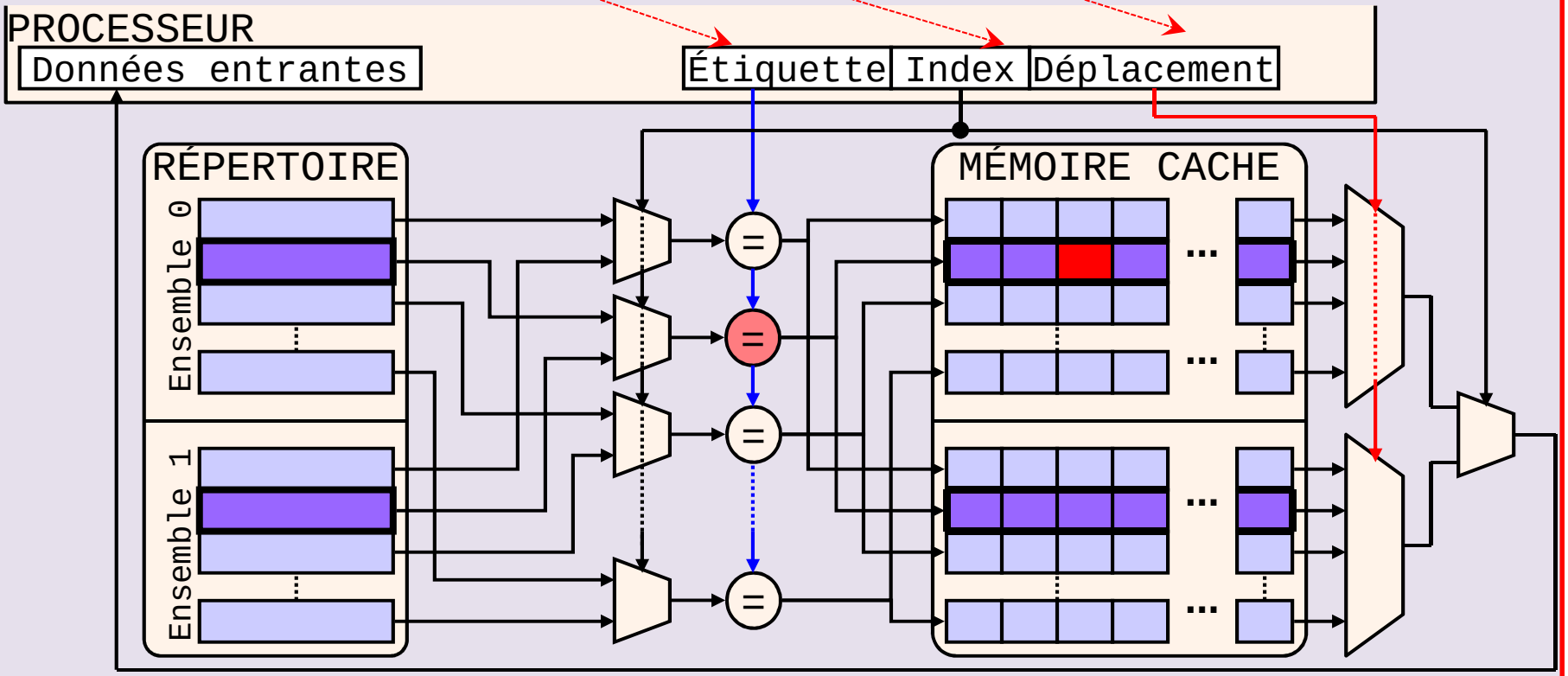
- Un **répertoire** est associé à chaque cache **associatif**.
- Le répertoire contient le numéro (adresse) de chaque bloc dans le cache, appelé **étiquette**.
- Toute les étiquettes dans le répertoire sont examinées en parallèle.
- **Pour un cache totalement associatif**, chaque adresse contient donc une étiquette et un **déplacement** dans le bloc.



adresse

Pour un cache associatif par ensemble, une adresse cherchée par le processeur peut être partagée en 3 champs:

1. **L'étiquette est la partie de l'adresse du bloc utilisée pour la comparaison.**
2. **L'index identifie l'ensemble.**
3. **Le déplacement est l'adresse dans le bloc du mot cherché.**



Protocoles de remplacement

- Placer les données dans le cache et gérer son accès est une chose; continuer à l'alimenter alors ***qu'il est plein demande*** une **technique de remplacement**.
- ☞ **Hypothèse :** On suppose que toutes les lignes possibles **sont occupées par des données valides** et qu'il y a lieu de charger un autre élément parce qu'il est demandé.

- Les techniques de remplacement pour les caches associatifs et les caches associatifs par blocs sont de même nature que celles de la **pagination.**

- Dans le cache à accès direct, ***un seul emplacement est possible***, il n'y a pas lieu à protocole.
- Pour les autres caches, on trouve des réalisations diverses :

1. Le ***premier entré, premier sorti***, encore dite FIFO (first in first out).
   - Cette technique est la plus simple à concevoir mais assez compliquée à réaliser.
   - Sa gestion nécessite un compteur circulaire ou une liste chaînée qui conserve ***l'ordre de chargement.***
   - De plus, elle est éloignée de l'optimum. Elle n'est plus utilisée dans les caches de processeurs
2. Le ***choix aléatoire*** (random choice) est plus simple à réaliser.
   - La ligne est choisie de manière aléatoire ou plutôt pseudo aléatoire.
   - Pour cela on peut utiliser la valeur d'un compteur ***convenablement brouillé***.
   - Cette technique est peu coûteuse en câblage.
   - Elle est loin de l'optimum.

3. Le ***moins récemment utilisé*** ou LRU (least recently used). Il lui faut un mécanisme matériel qui s'apparente à celui mis en œuvre dans le FIFO.
   1. À chaque **accès réussi** le numéro de ligne est amené en tête de liste.
   2. Lors d'un échec, on choisit la ligne dont le numéro **est en queue de liste(⇔ la plus ancienne ou le moins récemment utilisé)**.
   3. Le remplacement fait, ce numéro est placé en tête.
   4. Cette liste peut être gérée par un jeu de registres, des tableaux booléens ou encore par ***des compteurs associés chacun à un emplacement***.

- **On manipule les compteurs comme suit :**
  - à chaque accès réussi, le compteur correspondant est mis à zéro.
  - Les compteurs des autres emplacements sont incrémentés de 1.
  - L'emplacement dont le compteur a le **contenu le plus élevé est le plus anciennement utilisé.**

4. Le ***non récemment utilisé*** ou NRU (not recently used).
   - Chaque ligne a un compteur qui contient l'attribut de la ligne pris parmi les suivants :
     0 pas d'accès récent, pas modifiée;
     1 pas d'accès récent, modifiée;
     2 accès récent, pas modifiée;
     3 accès récent, modifiée.

Rappels :

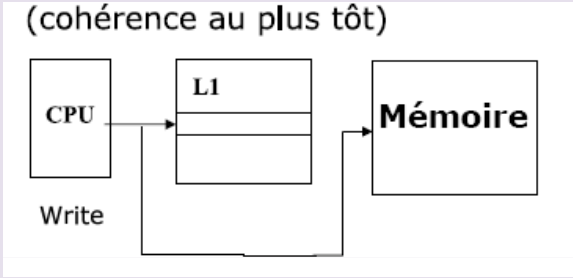
- **Les accès en lecture sur le cache sont les plus nombreux (lecture d'opérandes, accès aux instructions) et ne posent pas de réels problèmes**

- **Alors que les accès en écriture, moins nombreux, nécessitent une gestion rigoureuse de la cohérence des données au travers de la hiérarchie mémoire.**

- **La stratégie d'écriture est fondamentale pour la performance du cache de données (environ 25% des accès à ce cache sont des écritures).**

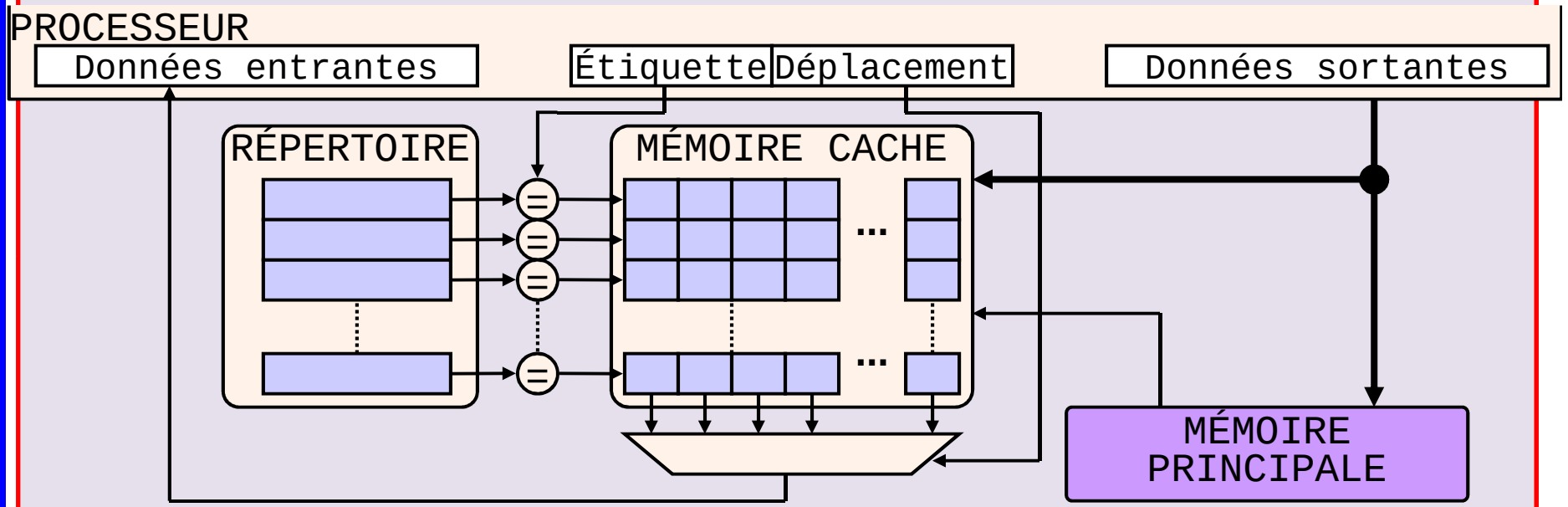
- **Les protocoles :** ***Pour une adresse présente dans le cache, on pratique :***

1. *L'**écriture simultanée*** (ou write-through).
   - Une ligne du cache modifiée (ou dirty line) est **écrite quasi simultanément dans le cache et dans la mémoire** qui le suit dans la hiérarchie.



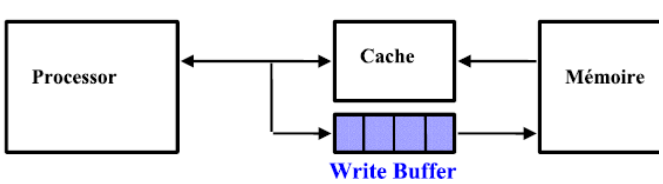
   - La cohérence est garantie entre ces deux niveaux au prix d'une ***augmentation du trafic sur le ou les bus.***
   - Dans des systèmes à quatre niveaux : L1, L2, mémoire centrale et disque, il faut préciser quel est le couple ou les couples où l'écriture simultanée est appliquée.
     - On peut avoir **écriture simultanée** dans L1 et L2 mais pas dans la mémoire centrale, ou encore entre la mémoire et le disque, mais cela relève du système d'exploitation.

- **Exemple :**


PROCESSEUR
Données entrantes
Étiquette
Déplacement
Données sortantes
RÉPERTOIRE
MÉMOIRE CACHE
MÉMOIRE PRINCIPALE

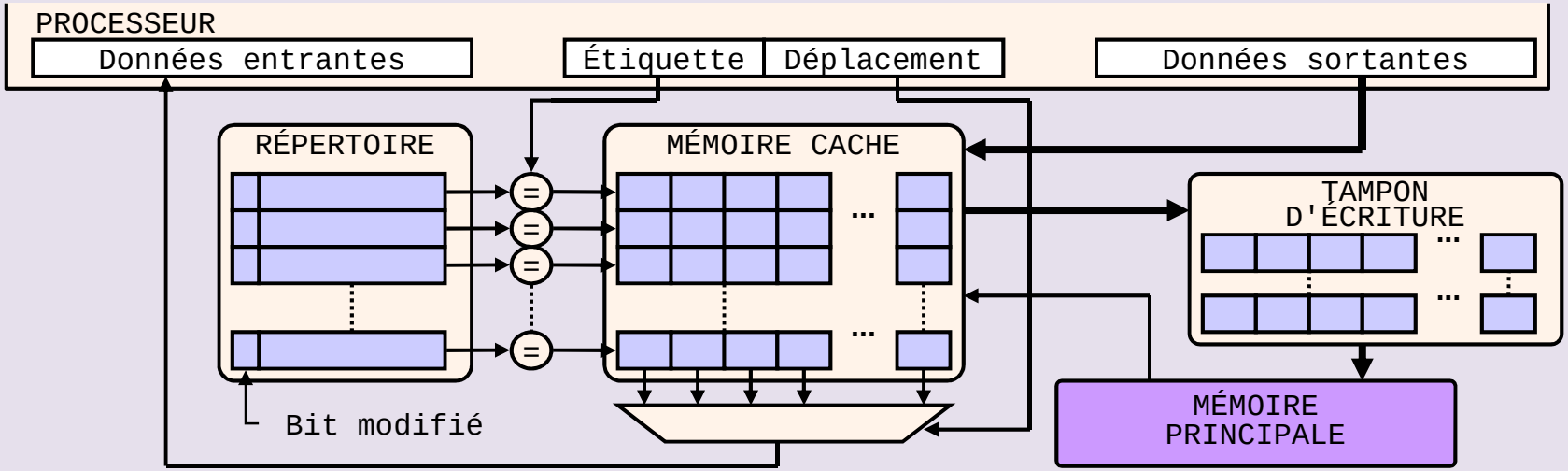
2. *L'écriture différée* (posted-write).

- **Principe : La recherche de données en mémoire principale lors d'un échec de lecture dans le cache est prioritaire par rapport à l'écriture. Cette dernière peut donc être suspendue en utilisant un tampon d'écriture (*write buffer*).**



- Buffer : FIFO entre cache et mémoire
- Proc : écrit les données dans le buffer et le cache
- **Contrôleur mémoire** : écrit le contenu de write buffer dans la mémoire

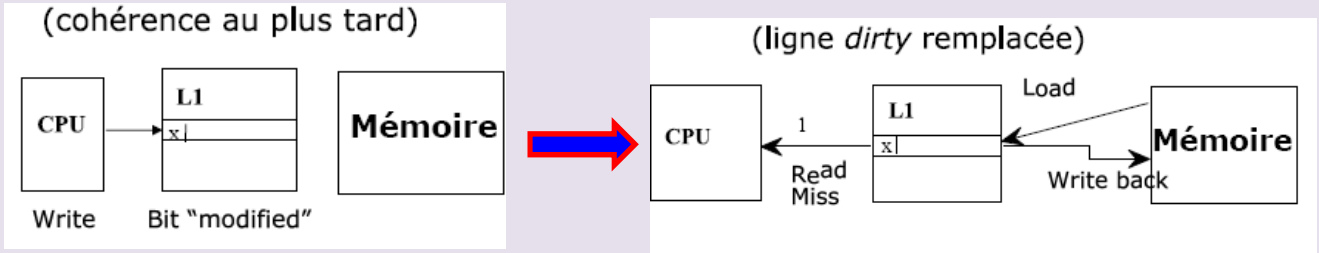
- **On place la donnée à écrire dans un tampon d'écriture (write-buffer) dont le contenu sera reporté ensuite (*La donnée est écrite seulement en cache. La modification du bloc en mémoire est faite plus tard (lors de la suppression du bloc du cache)*)**



Cependant, les tampons d'écriture compliquent les choses car ils peuvent contenir des valeurs modifiées d'un bloc qui fait l'objet d'un échec en lecture.

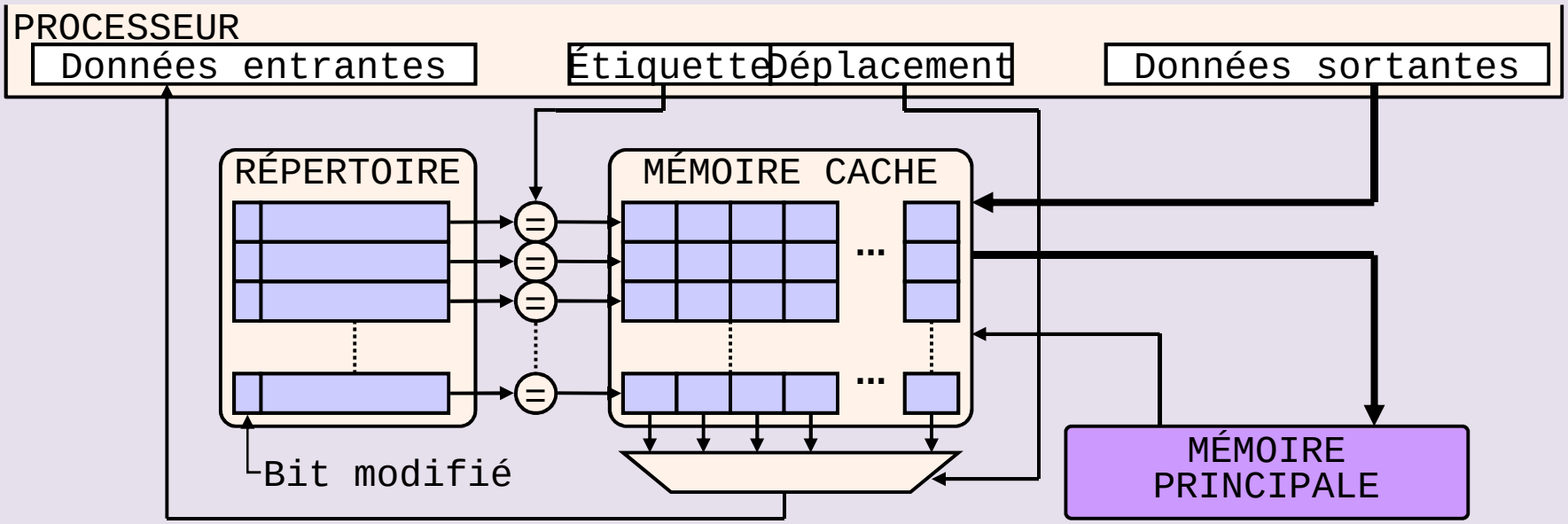
3. *La **recopie*** (write-back) : **la ligne est marquée modifiée et conservée telle quelle dans le cache.**

- La réécriture ou recopie ou rangement (*write-back* ou *copy-back*) consiste à écrire uniquement dans le bloc du cache.



- **Le bloc modifié du cache est recopié en mémoire principale *uniquement quand il est remplacé*.**

  Un bit modifié (*modified bit*) dans le répertoire du cache indique si le bloc a été modifié.



Le processeur cherche un mot d'abord dans le cache. En cas d'échec (*cache miss*), le mot est cherché en mémoire et son bloc est copié dans le cache pour un accès futur. En cas de succès (*cache hit*), la mémoire principale n'est pas accédée.

- Le temps passé à attendre une réponse de la mémoire (suspension mémoire ou attente mémoire) a un impact fondamental sur le temps d'exécution d'un programme:

Temps d'exécution = Temps de cycle x

(# cycles d'exécution + # cycles d'attente mémoire)

- **Définitions :**
  - La pénalité d'accès : est le temps (en nombre des cycles) nécessaire pour transférer une donnée de la mémoire au processeur
  - Pénalité d'échec = temps de remplacement d'un bloc dans le niveau supérieur
  - *Le taux d'échec, est le rapport du nombre d'échecs (défauts) de cache sur le nombre d'accès mémoire.*

 # cycles d'attente mémoire = # accès mémoire ∞

(Pénalité d'accès + Taux d'échec x Pénalité d'échec)

 # cycles d'attente mémoire = # accès mémoire ∞

(Pénalité d'accès + Taux d'échec x Pénalité d'échec)

Conclusion :Pour améliorer cette performance, il y a trois moyens:

1. **Réduire le taux d'échec**
2. **Réduire la pénalité d'échec (⇔donnée hors cache)**
3. **Réduire le temps de l'accès réussi au cache**

D'abord, il faut remarquer qu'il y a trois catégories d'échec lors de l'accès à un cache:

1. **Échec obligatoire (ou de première référence : Miss à froid ou cold miss):** lors de son premier accès, un bloc n'est sûrement pas dans le cache.
2. **Échec de capacité « Défaut de capacité »** si le cache ne peut contenir tous les blocs nécessaires pendant l'exécution d'un programme, des blocs seront écartés puis rappelés plus tard.
3. **Échec de conflit (ou de collision ou d'interférence : Miss de conflit):** dans des caches ***associatifs par ensemble*** ou à ***correspondance directe***, un bloc peut être écarté et rappelé plus tard si trop de blocs doivent être placés dans son ensemble

   ↔ *(2 blocs mémoires peuvent être placés dans le même bloc du cache. L'un éjecte l'autre alors qu'il y a de la place libre, ou il y a des blocs qui ne sont plus utilisés)*

❑ **Une solution:**

- **Augmenter la taille des blocs pour mieux exploiter la localité spatiale des références.**
  - → **En même temps, des blocs plus gros peuvent augmenter les échecs de conflit et même les échecs de capacité si le cache est petit.**
  - → **Plusieurs paramètres architecturaux définissent la taille optimale des blocs.**
- **Une associativité plus élevée peut améliorer les taux d'échec.**
  - ➢ **Des lois empiriques sont généralement appliquées.**
- **Par exemple, l'associativité à 8 blocs est aussi efficace que l'associativité totale pour les tailles de cache plus courantes.**

Malheureusement, une associativité plus élevée ne peut être obtenue qu'au prix d'un temps d'accès réussi plus élevé (dû à une logique de contrôle plus compliquée).

Une manière d'améliorer les taux d'échec sans affecter la fréquence d'horloge du processeur est de *lire à l'avance* les éléments *avant qu'ils soient demandés par le processeur*.

- **Données et instructions peuvent être lues à l'avance, soit directement dans le cache, soit dans un tampon externe.**

Cette lecture anticipée est fréquemment faite par matériel.

- **Exemple : le processeur DEC Alpha AXP 21064 lit deux blocs lors d'un échec en lecture des instructions: le bloc demandé et le suivant. Ce dernier est placé dans un tampon de flux d'instructions.**

❑Observations :

- **Un tampon capable de stocker un seul bloc d'instructions peut éviter de 15% à 25% des échecs.**

Une alternative à la lecture anticipée matérielle est que le compilateur introduise des instructions de préchargement pour aller chercher la donnée avant qu'elle ne soit nécessaire. Il y a plusieurs types de lecture anticipée:

- **La lecture anticipée des registres, qui charge la valeur dans un registre.**

- **La lecture anticipée du cache, qui charge la valeur uniquement dans le cache et pas dans les registres.**

Cette prélecture n'a de sens que si le processeur peut travailler pendant que la donnée lue à l'avance est en cours de lecture. Ceci suppose que le cache peut continuer à fournir des informations au processeur pendant qu'il attend l'arrivée de la donnée anticipée.

Cette solution, qui ne nécessite pas de matériel supplémentaire, est basée sur l'observation que le CPU a juste besoin d'un mot d'un bloc à la fois. (⇔ *Ne pas attendre que tout le bloc mémoire soit chargé pour utiliser les données)*

1. Le **redémarrage précoce**: aussitôt que le mot demandé du bloc arrive, il est envoyé au CPU qui redémarre (⇔ dès que la donnée nécessaire est chargée, l'utiliser)
2. **Le mot critique en premier**: le mot demandé est cherché en mémoire et envoyé au CPU d'abord, ensuite le restant du bloc est copié dans la cache pendant que la CPU continue l'exécution; cette solution est aussi appelée **lecture enroulée** (⇔ charger en premier la donnée ayant produit l'échec.)

Généralement, ces techniques ne sont profitables que pour de très grosses tailles de blocs.

- **Question :**
  - Est-ce qu'il faut **accélérer le cache** pour suivre la vitesse croissante du CPU ou **agrandir le cache** pour suivre la taille croissante de la mémoire principale?

    → Les deux!
- En ajoutant un **deuxième niveau de cache** entre le cache originel et la mémoire;
  - Le premier niveau peut être assez petit pour correspondre au temps de cycle du CPU,
  - Alors que le second niveau peut être assez grand pour capturer beaucoup d'accès qui iraient à la mémoire principale, en réduisant **ainsi la pénalité d'échec effective.**

  ⇒ Les paramètres du cache de second niveau sont donc différents de ceux du cache de premier niveau: il peut être plus lent mais aussi plus grand, permettant un plus haut degré d'associativité ainsi qu'une taille de blocs plus grande.

- Une partie importante du ***temps d'accès réussi au cache*** est le *temps nécessaire pour* **comparer le contenu du répertoire de cache à l'adresse,**

  ⇒ ***fonction de la taille du cache.***

- Un dispositif matériel plus petit est plus rapide, et un cache petit améliore certainement le temps d'accès réussi

  - (il est aussi critique de garder un cache suffisamment petit pour qu'il puisse tenir dans le même circuit que le processeur).

- Pour la même raison, il est important de garder un cache simple, p.ex. en utilisant la correspondance directe.

- Ainsi, la pression pour un cycle d'horloge rapide encourage la réalisation de caches petits et simples pour les caches primaires.

- Même un cache petit et simple doit prendre en compte **la traduction d'adresse virtuelle** venant du CPU en adresse physique pour accéder à la mémoire principale.

- Une solution **est d'éviter la traduction d'adresse durant l'indexation du cache** (cache virtuel***), puisque les accès réussi sont beaucoup plus courants que les échecs.***

- **Le microprocesseur *Motorola 68030*, ancien,**
  - A deux caches intégrés de 16 lignes de mots de 32 bits soit un total de 256 octets.
  - Le cache peut être vu comme une extension de 64 registres de 32 bits. Ce type d'assimilation devient rare.

- **Le *MIPS R10000* a deux niveaux de caches.**
  - Deux caches primaires de 32 Ko, programme et données dans la puce.
  - Un cache externe unique de 512 Ko de 16 Mo peut être ajouté, son contrôleur est dans la puce processeur. Il faut donc prévoir ce circuit de type SRAM sur la carte. Tous ces caches sont **associatifs par blocs** et fonctionnent **en LRU.**

  - Ce système de caches a deux originalités :
    - → Le décodage partiel des instructions est fait lors de leur chargement dans le cache de premier niveau.
    - → Le cache de données fonctionne en recopie; une prédiction du bloc du cache externe qui sera lu.

- **Le contrôleur de mémoire cache *Intel 82385* est donné pour 95% d'accès réussis (?) pour une taille de cache de 32Ko.**
  - → Il utilise la technique d'écriture **posted-write.**
  - → À partir du Pentium Pro, les deux caches primaire et secondaire sont intégrés dans la puce processeur.
  - → La taille du cache secondaire est donc limitée par rapport à celle du MIPS R10000 mais les temps d'accès sont meilleurs.
  - → Le cache L2 inséré dans la puce processeur du Pentium IV transfère 256 bits par cycle d'horloge.

- **Le cache de l'*Alpha 21064* a les caractéristiques et performances suivantes:**
  - → deux caches de 8192 octets, instructions et données;
  - → blocs de 32 octets;
  - → ***correspondance directe*** (pas de mémoire associative);
  - → **écriture simultanée** avec néanmoins **un petit tampon de 4 blocs;**
  - → écriture non attribuée;
  - → le n°de bloc est sur 29 bits, l'adresse dans le blo c sur 5 bits;
  - → le n° de bloc est décomposé en **21 bits d'étiquette** et **8 bits d'index** dans le cache;
  - → si tout fonctionne, la donnée est lue avec les étiquettes **en 2 cycles;**
  - → sinon, l'UC est arrêtée pendant 10 cycles pour lire 32 octets;
  - → en écriture, la donnée est récrite dans le cache si elle y est présente, puis récrit; en mémoire centrale via le tampon.

- **Les caches, tant de mémoire centrale que de disque, sont de plus en plus nombreux et volumineux, eu égard :**
  - → à la baisse du prix des mémoires, même pour celles qui ont des temps d'accès très courts;
  - → à l'utilité de ces caches en termes de performances observables;
  - → à la plus grande intégration qui autorise l'insertion des caches dans le circuit microprocesseur.
- Rappel :
  - Les caches n'ont pas d'utilité intrinsèque. Ils atténuent l'effet de la différence des performances des deux dispositifs entre lesquels on les place
- **La paire de caches L1 et L2 est devenue la règle.**
  - → Le cache L1 est déjà toujours dans la puce processeur, cela commence à être le cas du cache L2.
  - → Le troisième cache parfois nommé L0 se répand rapidement. Il est apparu pour les AMD K6-3, Itanium d'Intel, machine PowerMac G4, etc. Il est probable que cette course ressemblera à celle des pipelines qui sont passés de 3 à 5 puis 20 étages