Mémoire hiérarchique

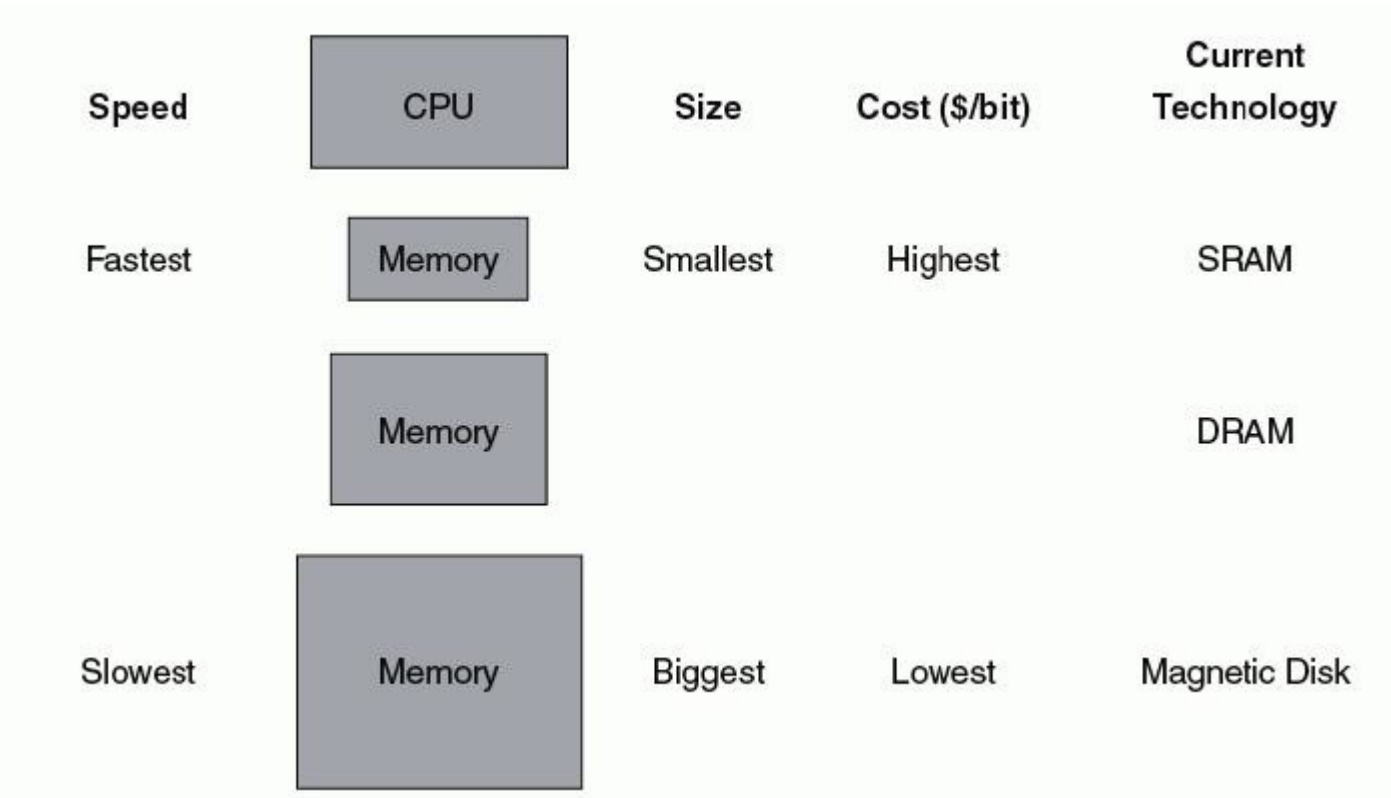
Mémoire

- Mémoire hiérarchique
  - Mécanismes permettant de rendre une mémoire de grande taille accessible aussi rapidement qu'une mémoire de petite taille pour le CPU
  - Idée sous-jacente : un programme n'accède pas à la totalité de son code et de ses données avec une probabilité uniforme
  - Principe de localité: ***les programmes accèdent à une portion relativement petite de leur espace d'adressage à un instant donné de leur exécution***

Principe de localité

- Deux différents types de localité
  - Localité temporelle : si une donnée mémoire est référencée, elle tendra à être référencée de nouveau prochainement
  - Localité spatiale : si une donnée mémoire est référencée, les données d'adresses proches tendront à être référencées prochainement
  - Conséquence : on peut tirer avantage du principe de localité en organisant la mémoire de manière hiérarchique
  - Une ***mémoire hiérarchique consiste en des niveaux multiples ayant des vitesses et des tailles différentes***

Structure fondamentale d'une mémoire hiérarchique



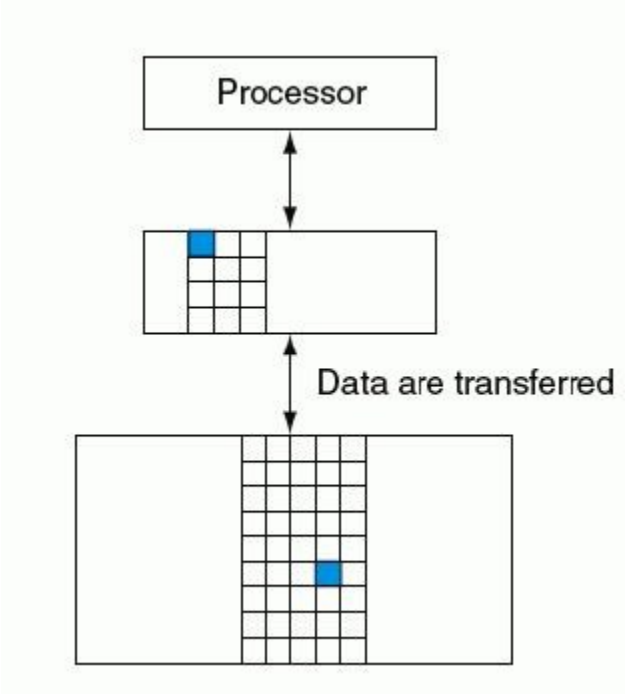
Mémoire hiérarchique : fondements

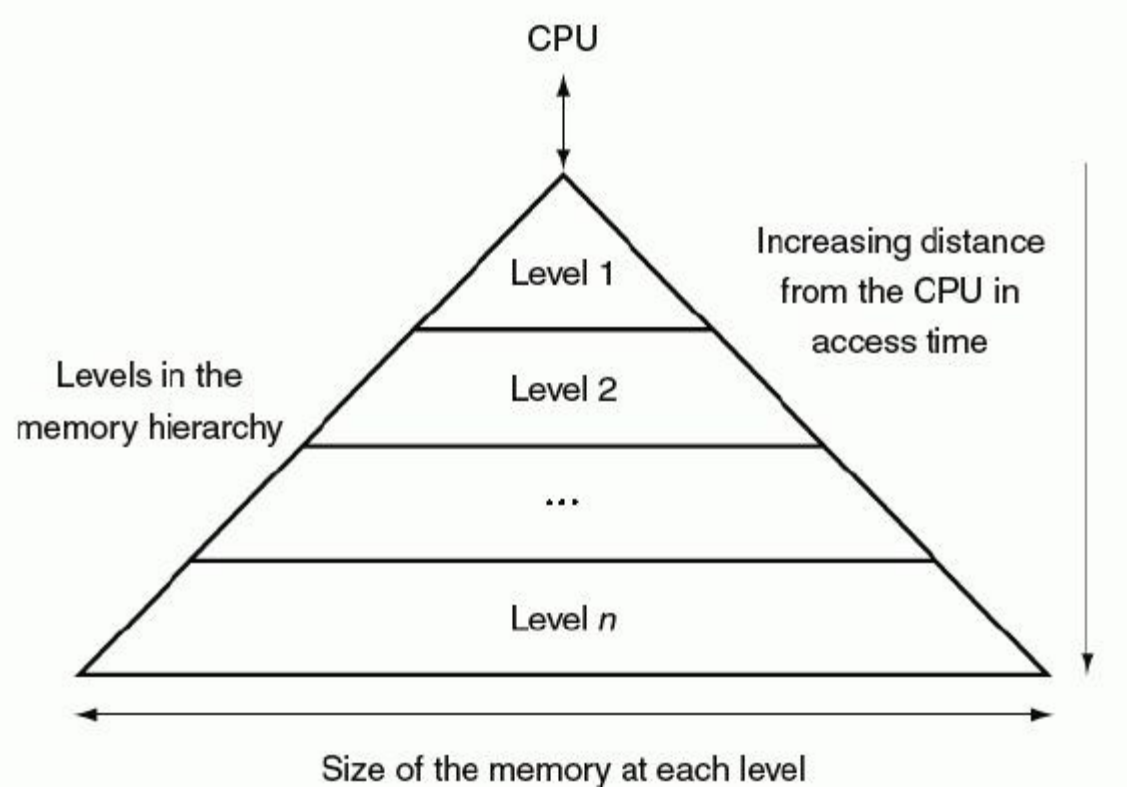
- Trois technologies fondamentales pour implanter une mémoire hiérarchique :
  - Pour la mémoire principale : DRAM (Dynamic Random Access Memory – 1 transistor/bit - rapide)
  - Pour les niveaux les plus proches du processeur : SRAM (Static Random Access Memory – 4 transistors/bit – très rapide)
  - Pour les niveaux les plus grands et lents : disques magnétiques ou mémoire flash

Mémoire hiérarchique : fondements

- Un niveau plus proche du processeur (niveau supérieur) est généralement un sous-ensemble de tout niveau plus éloigné (inférieur), et la totalité des données est stockée dans le niveau le plus bas
- Les données sont recopiées entre deux niveaux adjacents à un instant donné
  - ***Bloc/Ligne*** : unité minimale d'information qui peut être présente ou pas dans un niveau
  - ***Taux de succès (hit rate) :*** fraction des accès mémoire trouvés dans un niveau

Mémoire hiérarchique : fondements





Mémoire hiérarchique : performance

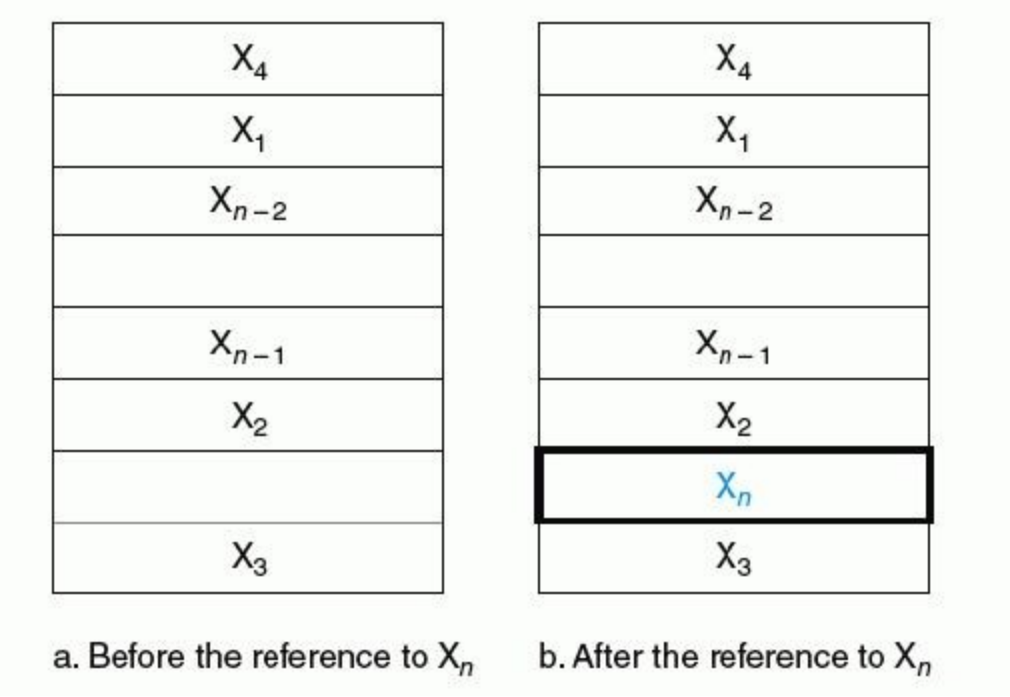
- La performance est la principale raison qui motive l'usage d'une mémoire hiérarchique :
  - ***Temps d'accès (hit time)*** : temps nécessaire pour accéder à un niveau de la hiérarchie, incluant le temps requis pour déterminer si l'accès est un succès (cache hit) ou un échec (cache miss, défaut de cache)
  - ***Pénalité de faute (miss penalty)*** : temps nécessaire pour charger (fetch) un bloc dans un niveau de mémoire à partir du niveau inférieur, incluant le temps d'accès au bloc, le temps de transmission entre niveaux, et le temps d'insertion dans le niveau dans lequel le défaut de cache a eu lieu

Mémoire hiérarchique : performance

- Les concepts utilisés pour implanter un système de mémoire hiérarchique affectent de nombreux autres aspects de l'architecture de l'ordinateur :
  - Comment le système d'exploitation gère la mémoire et les entrées/sorties
  - Comment les compilateurs génèrent le code
  - Comment les applications structurent leurs données

Cache simple

Les requêtes mémoire du processeur sont d'un mot à la fois et chaque bloc contient ici un seul mot.



a. Before the reference to X_n b. After the reference to X_n

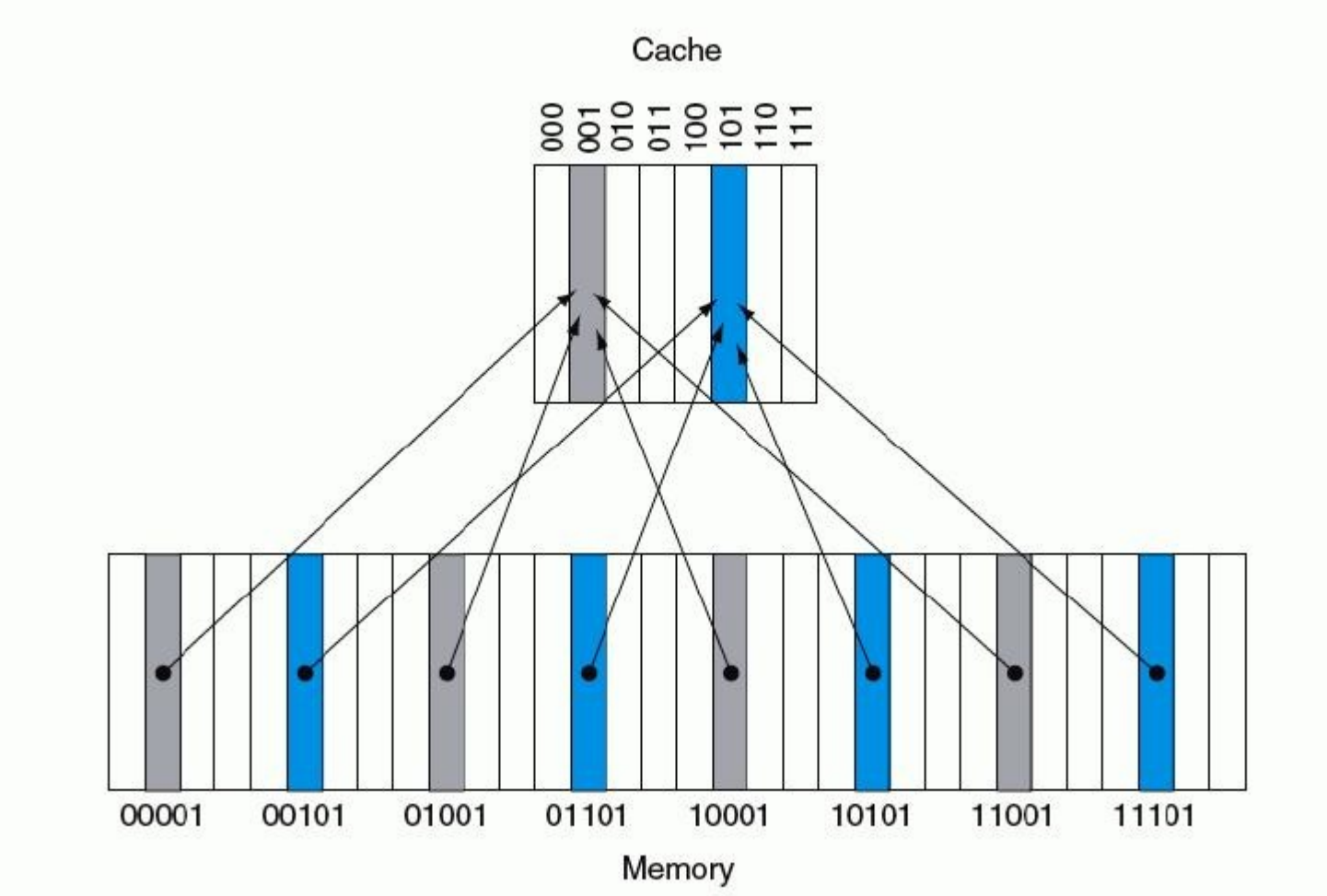
Questions :
Comment retrouver une donnée dans le cache ?
Comment savoir si une donnée est présente dans le cache ?

Cache à correspondance directe

- Si chaque mot ne peut aller qu'à exactement une place dans le cache, alors le retrouver dans le cache est immédiat (cache à accès direct)
- La manière la plus simple d'assigner un emplacement du cache à un mot est de déterminer cet emplacement en fonction de l'adresse du mot en mémoire (*direct mapping*, correspondance directe)

(adresse du bloc) modulo (nombre de blocs dans le cache)

Cache à correspondance directe



Cache à correspondance directe

- Comment savoir si un mot est dans le cache ?
  - Chaque emplacement du cache peut contenir les données d'un nombre important de places mémoires différentes
  - Astuce : utiliser un ensemble de balises (tags)
    - Une balise est un champ dans une table annexée à la mémoire hiérarchique qui contient l'information suffisante pour déterminer si le bloc associé dans la hiérarchie correspond au bloc demandé
    - Un bit de validité est un champ additionnel des tables de la mémoire hiérarchique qui indique si le bloc associé dans la hiérarchie contient des données valides (validité : contenu bloc = données mémoire)

Accès au cache

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (7.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (7.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (7.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (7.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (7.6f)	$(10010_{two} \bmod 8) = 010_{two}$

Séquence d'accès mémoire à un cache de 8 blocs vide
(adresses réduites aux numéros de blocs)

Accès au cache

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Dlata
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memorlly (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

Accès au cache

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})

Taille d'un cache (correspondance directe)

- Le nombre total de bits requis pour un cache dépend de la taille du cache et de celle de l'adresse (stockage des données et balises)
  - exemple : adresse 32-bits
  - cache à correspondance directe
  - taille de cache : 2^n blocs, donc n bits d'adresse utilisés pour l'index (numéro de bloc associé)
  - taille de bloc : 2^m mots (2^{m+2} octets), donc m bits utilisés pour situer le mot dans le bloc, et 2 bits utilisés dans l'adresse pour les 4 octets de chaque mot

Taille d'un cache (correspondance directe)

- taille d'une balise est donc (ce qui reste) :

$$32 - (n+m+2)$$

- nombre total de bits dans un cache à correspondance directe :

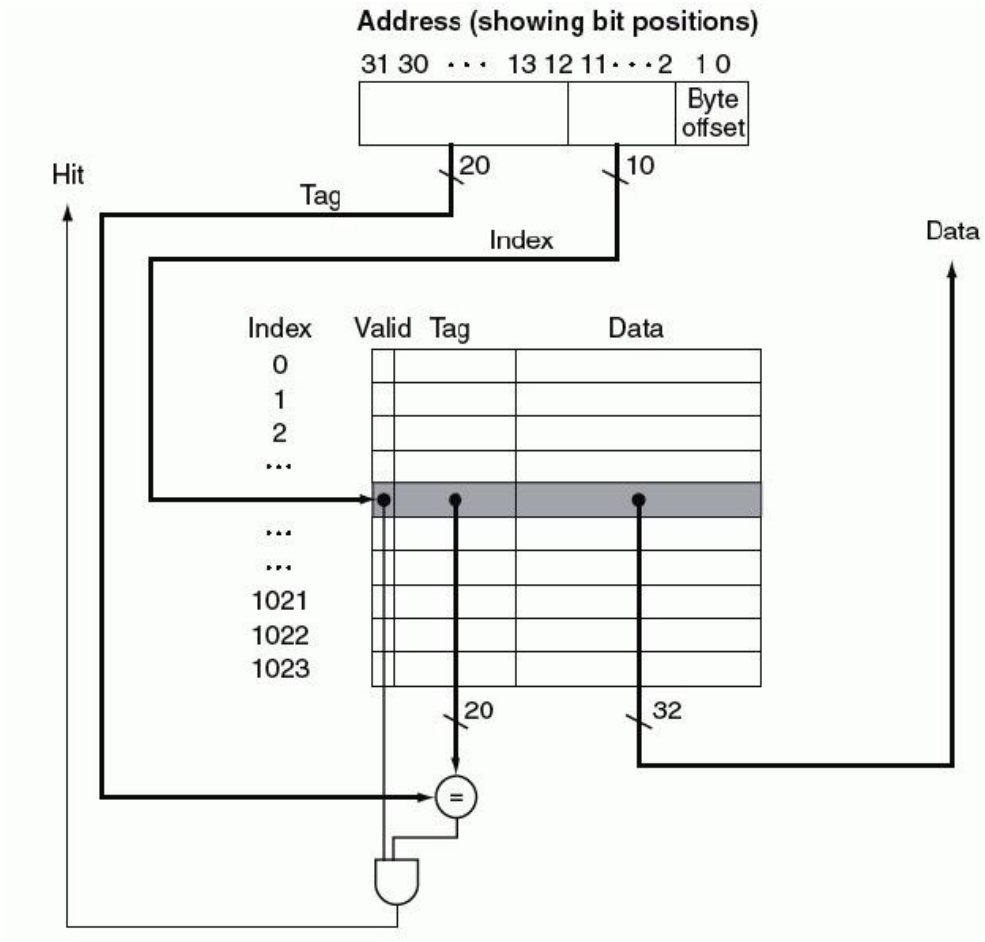
$$2^n \times (\text{block size} + \text{tag size} + \text{valid field size})$$

- dans cet exemple, avec une taille de bloc de 2^m mots, et 1 bit de validité, le nombre total de bits du cache est

$$2^n x(2^m x32+(32-n-m-2)+1)$$

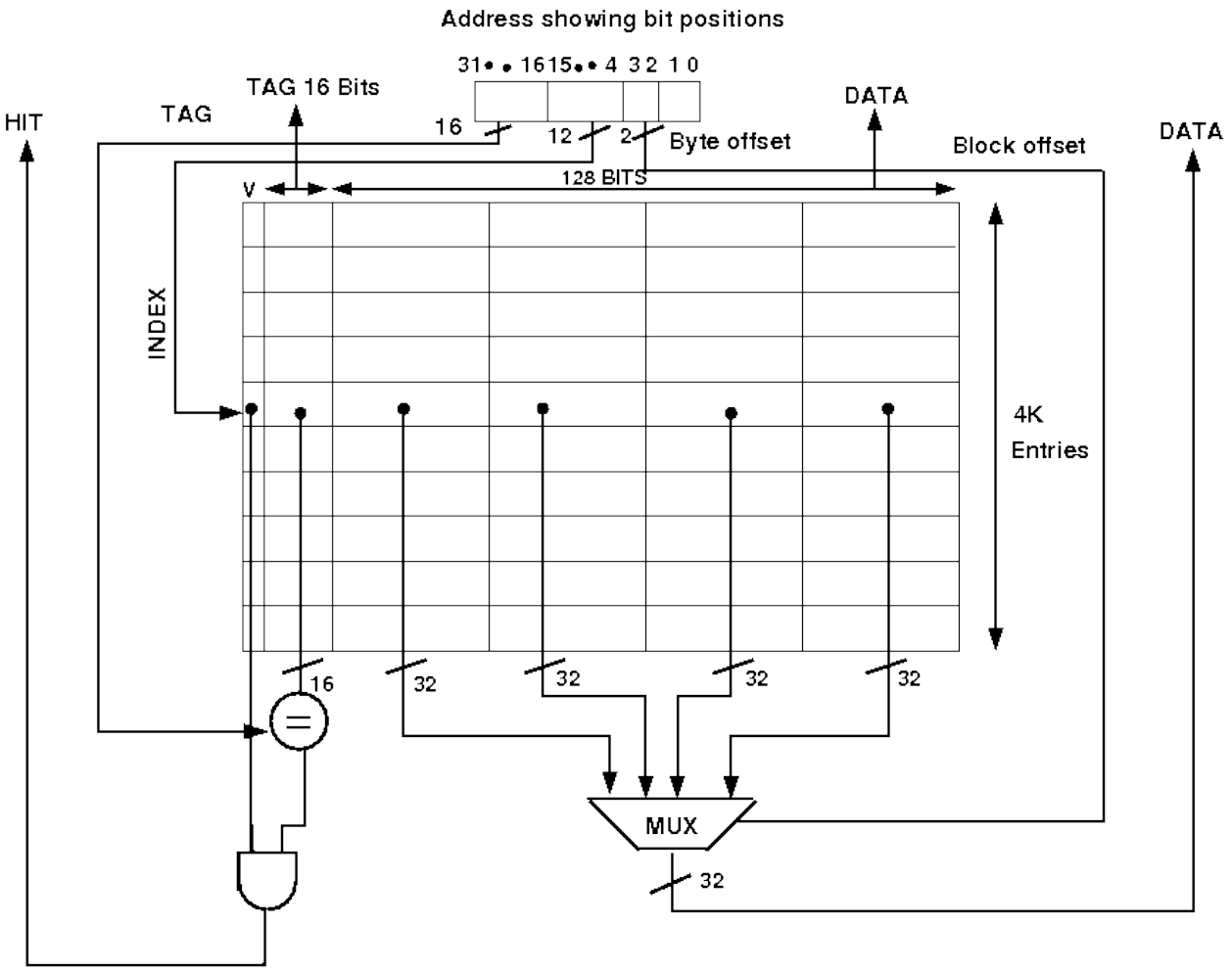
Taille d'un cache (correspondance directe)

exemple : chaque bloc contient ici un seul mot



Taille d'un cache (correspondance directe)

exemple : chaque bloc contient ici quatre mots



Gestion des fautes (cache miss)

- L'unité de contrôle doit détecter un défaut de cache et traiter l'échec en allant charger les données requises depuis la mémoire
  - La gestion des défauts de cache est faite en collaboration avec l'unité de contrôle du processeur, et avec un contrôleur séparé qui initie l'accès mémoire et recharge le cache
    - Envoi de la valeur/adresse du PC à la mémoire
    - Accès mémoire principale en lecture, attente acquittement
    - Ecriture de la ligne (bloc) du cache, en y plaçant les données de la mémoire dans la partie correspondante de la ligne
    - Ecriture des bits de poids fort de l'adresse dans le champ balise
    - Mise à 1 (activation) du bit de validité
    - Redémarrage de l'exécution de l'instruction à la première étape (chargement de l'instruction qui va cette fois-ci trouver la donnée voulue dans le cache)

Gestion des écritures

- Les accès en écriture sont gérés différemment :
  - ***Write-through*** (écriture immédiate): protocole dans lequel les accès en écriture mettent toujours à jour à la fois le cache et la mémoire, ce qui induit une cohérence permanente cache/ mémoire
  - ***Write buffer*** (tampon d'écriture): les données sont stockées dans une file en attendant d'être écrites en mémoire
  - ***Write-back*** (écriture différée): protocole dans lequel les accès en écriture sont gérés de façon à ne mettre à jour que les valeurs du cache, jusqu'à ce qu'un rechargement de bloc soit effectué, auquel cas le bloc modifié est réinjecté au niveau inférieur de la mémoire - un bit « sale » (dirty) indique si le bloc concerné a été modifié

Performance de cache

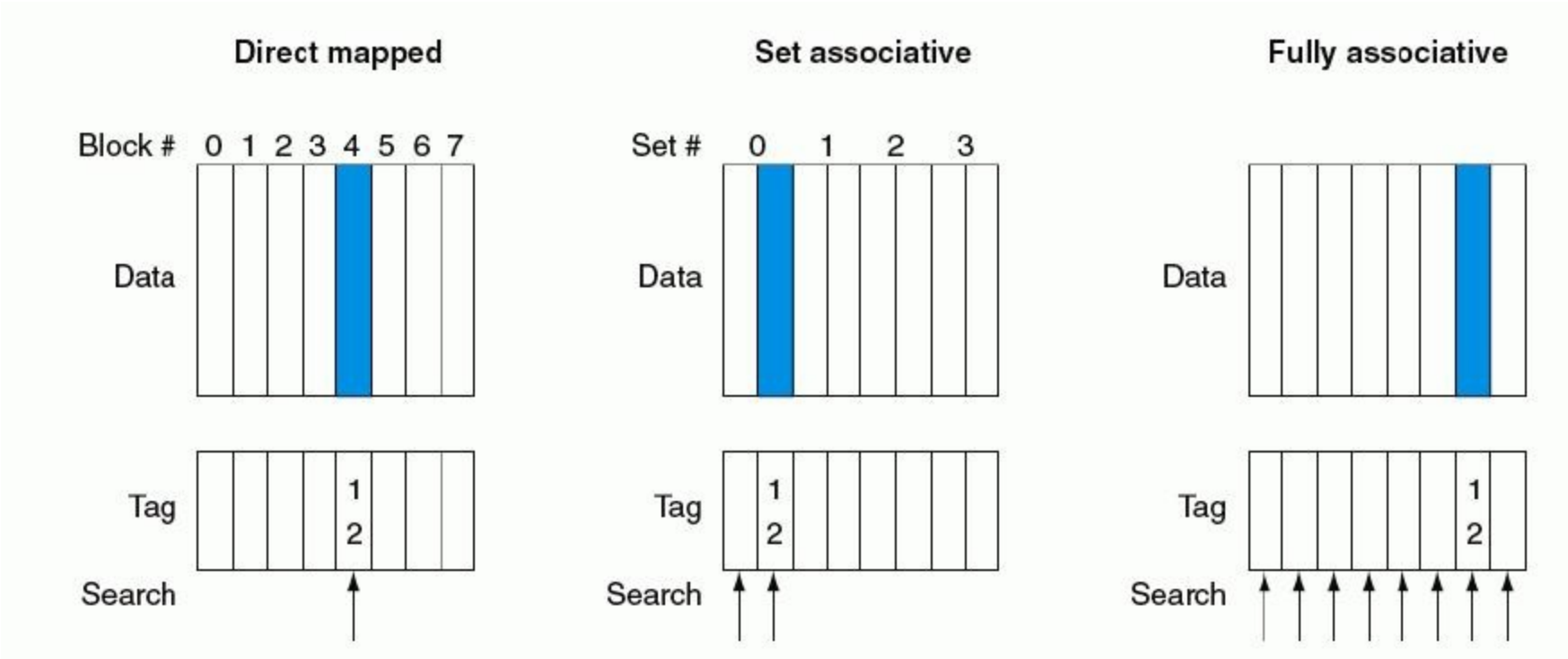
- Temps CPU requis pour les interactions cache-mémoire :
  - temps CPU = (cycles d'horloge pour l'exécution de l'instruction par le CPU + cycles d'horloge de paralysie mémoire) x période horloge
  - cycles de paralysie mémoire = cycles de paralysie en lecture + cycles de paralysie en écriture
  - cycles de paralysie en lecture = taux d'accès en lecture du programme x taux de fautes de lecture x pénalité de faute de lecture
  - cycles de paralysie en écriture plus difficile à quantifier (et très dépendants de la stratégie)

Réduction des fautes de cache

- Différents mécanismes pour placer les blocs dans le cache :
  - ***Fully associative*** (cache entièrement associatif) : structure de cache dans laquelle un bloc peut être placé à n'importe quel emplacement du cache
  - ***Set-associative*** (cache associatif par ensembles) : cache ayant un nombre fixé d'ensembles d'emplacements (au moins 2) dans lesquels chaque bloc peut être placé
    - Un cache associatif par ensembles ayant ***n*** emplacements est appelé cache à correspondance associative par ensembles de n
    - Un cache à correspondance associative par ensembles de n contient donc des ensembles constitués de n blocs
    - Pour connaître l'ensemble dans lequel un bloc peut être placé :

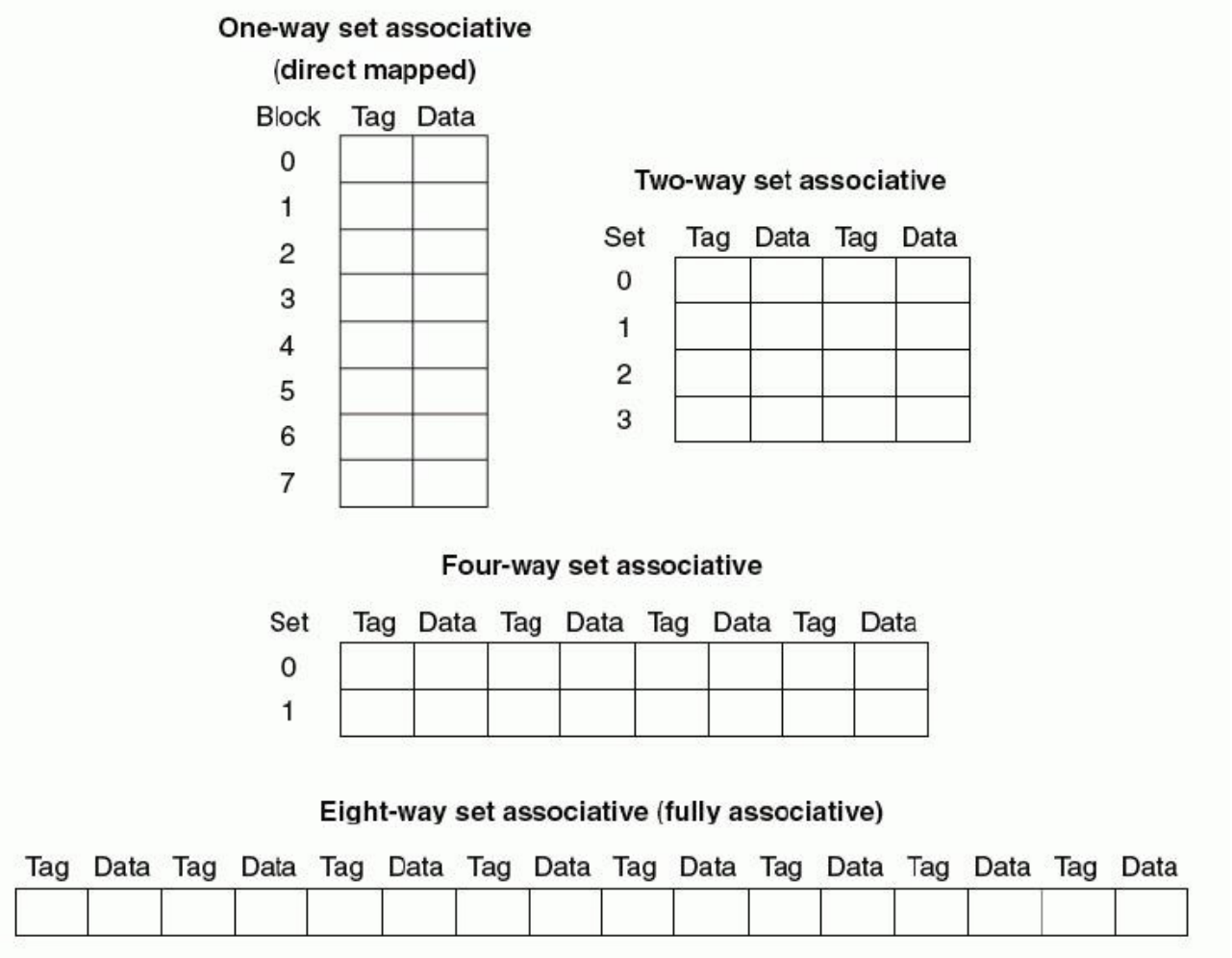
      (numéro de bloc) modulo (nombre d'ensembles dans le cache)
    - Conséquence : blocs consécutifs dans ensembles consécutifs

Types de caches : résumé

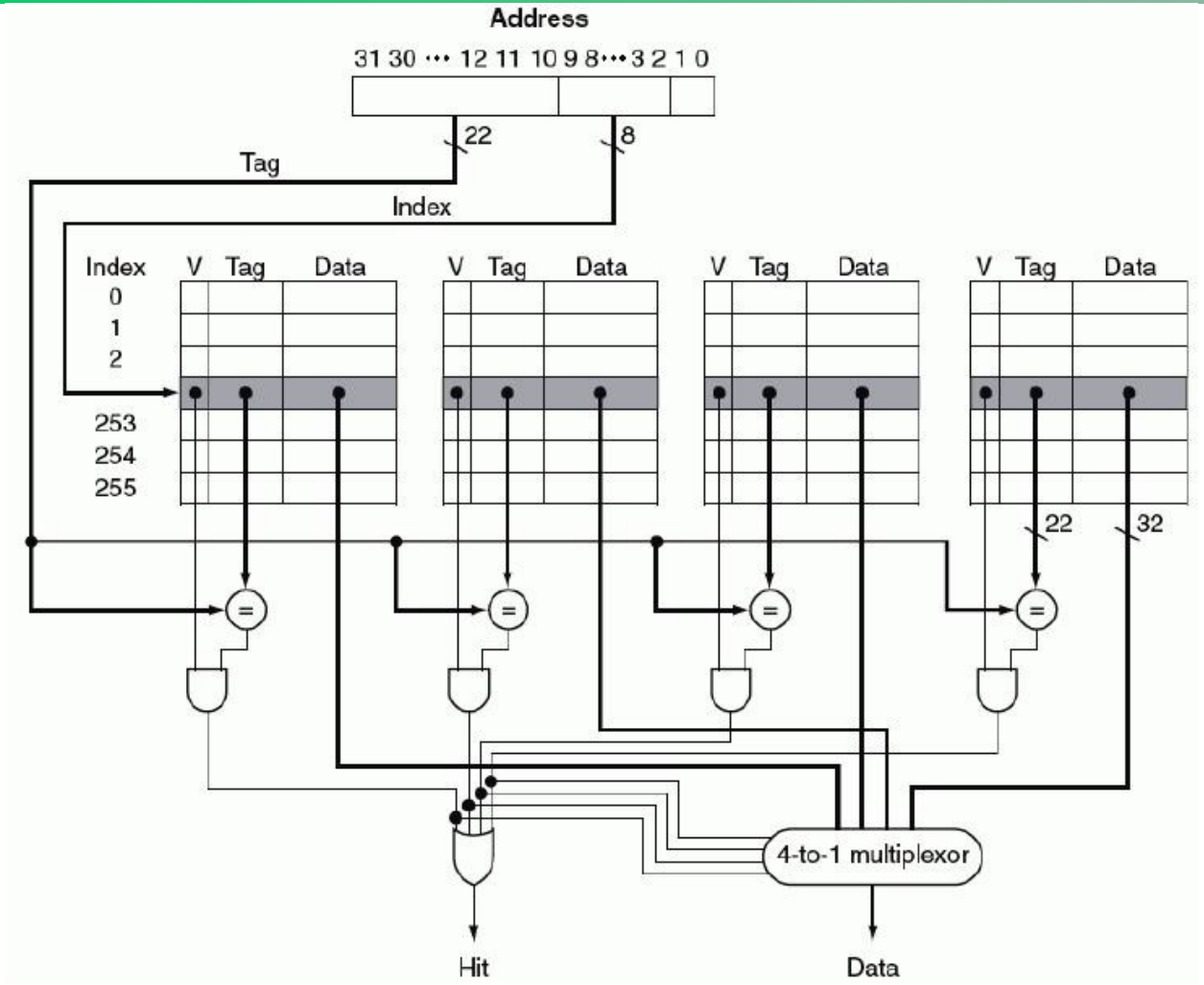


Bloc mémoire d'adresse 12 dans un cache à 8 blocs

Cache associatif par ensembles



Cache associatif par ensembles



Rechargement de blocs

- Dans un cache associatif, il y a plusieurs possibilités pour choisir où placer le bloc demandé, et donc choisir quel bloc remplacer :
  - ***Least recently used*** (LRU) : le bloc remplacé est celui qui a été inutilisé depuis le plus long temps, donc utilisé le moins récemment (utilisation d'une file FIFO)
  - ***Most recently used*** (MRU) : principe inverse, utile pour les situations avec accès cycliques (utilisation d'une pile LIFO)
  - ***Least frequently used*** (LFU) : le bloc remplacé est celui qui a été utilisé le moins souvent (utilisation de compteurs)
  - ***aléatoire, …etc***

Architecture de cache

- La performance d'un cache ne dépend pas seulement de son organisation, mais aussi de la façon dont différents caches peuvent être organisés
  - ***Caches séparés*** : distinguer un cache de données et un cache d'instructions, comme dans le MIPS, peut permettre des optimisations spécifiques (par exemple les instructions sont le plus souvent lues, pas écrites)
  - ***Cache multiniveaux*** : on peut réaliser une mémoire hiérarchique avec 2 niveaux de cache ou plus, et non pas seulement le cache et la mémoire principale