Architecture avancée

Cache

2018

Mémoire : la dure réalité

Mémoire idéale

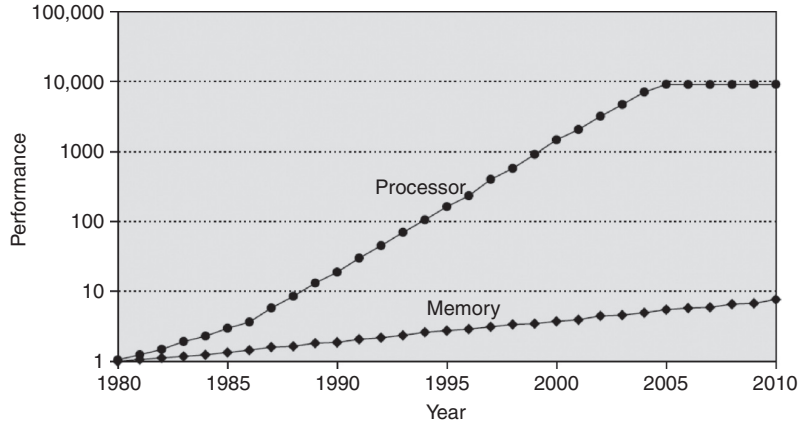
- rapide (temps d'accès très faible et débit énorme)
- très grande
- économique
- mais aussi privative et persistante (non volatile)

Quelques mémoires trouvées dans le commerce en 2018

Type de mémoire	taille	prix (€/Go)	latence (ns)	débit (Mo/s)
SRAM (module)	8 Mo	10000	1	
DRAM (DDR4)	8 Go	10	15	25000
Flash (DD SSD)	1 To	0,25	10^5	500
Disque Magnétique	5 To	0,04	10^7	

Réalité

- au mieux 2 caractéristiques parmi les 3
- Divergence de performances entre les mémoires et les processeurs
  - cycle CPU $\approx$ temps d'accès DRAM en 1980
  - performance depuis : CPU +50%/an vs DRAM +7%/an



Système mémoire

- Prendre le meilleur de chaque technologie et combiner intelligemment
  $\Rightarrow$ Hiérarchie mémoire : Registres du CPU $<$ Caches $<$ Mémoire principale $<$ Disque Dur
- Gestion : compilateur pour les registres/matériel pour les caches/système d'exploitation pour le reste
- Analyse de performance
  - Succès (Hit) ou échec (Miss) d'accès à la donnée dans un niveau de la hiérarchie
  - AMAT (Average Memory Access Time)
    $AMAT = t_{cache} + \tau_M * t_{mem}$
  - exemple d'un cache temps d'accès de 1 cycle avec taux d'échec de 10% et d'une mémoire ayant un taux d'accès de 100 cycles
- Cache idéal : $\tau_M \rightarrow 0$
  Comment prédire et anticiper l'utilité future des données et des instructions ?

Système mémoire

- Prendre le meilleur de chaque technologie et combiner intelligemment
  $\Rightarrow$ Hiérarchie mémoire : Registres du CPU $<$ Caches $<$ Mémoire principale $<$ Disque Dur
- Gestion : compilateur pour les registres/matériel pour les caches/système d'exploitation pour le reste
- Analyse de performance
  - Succès (Hit) ou échec (Miss) d'accès à la donnée dans un niveau de la hiérarchie
  - AMAT (Average Memory Access Time)
    $AMAT = t_{cache} + \tau_M * t_{mem}$
  - exemple d'un cache temps d'accès de 1 cycle avec taux d'échec de 10% et d'une mémoire ayant un taux d'accès de 100 cycles
- Cache idéal : $\tau_M \rightarrow 0$
  Comment prédire et anticiper l'utilité future des données et des instructions ?
- localité temporelle
- localité spatiale $\Rightarrow$ Ligne de cache (cache block)

Associativité et placement d'une ligne de cache

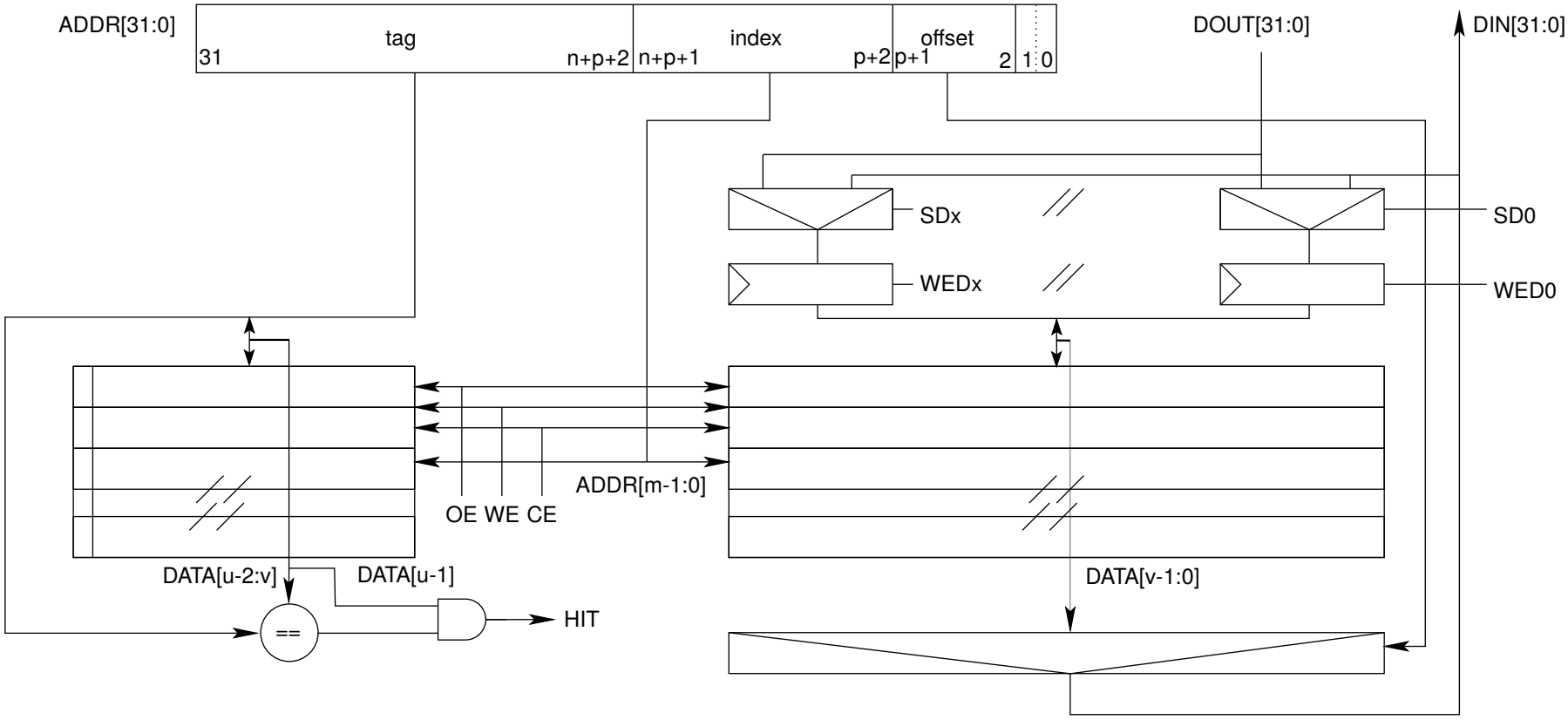
- où se trouve une donnée dans le cache ?
- Associativité = # d'emplacements autorisé pour une donnée dans le cache
- associativité réduit les conflits
- Organisation du cache :
  - cache à correspondance directe (associativité = 1)
    ligne placée à @ligne % #lignes
  - cache complètement associatif (associativité = #lignes)
  - généralisation : cache associatif par ensemble de N voies (associativité = N)
    ligne placée dans une des voies de l'ensemble : @ligne % #ensembles

Savoir si ma donnée est dans le cache

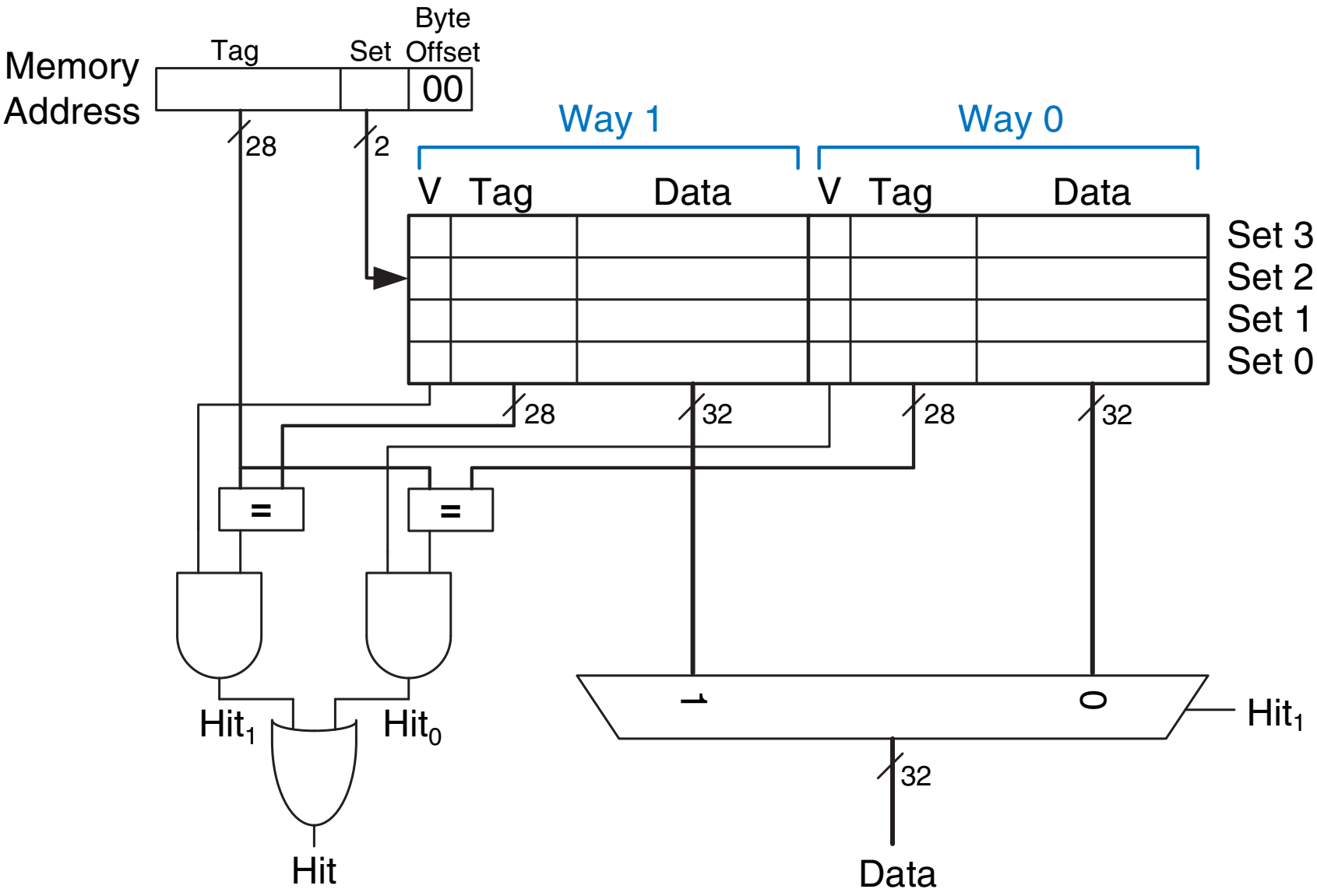
- Où = surjection
- Est-ce la bonne donnée ? = besoin d'une bijection
- tag = identifiant unique d'une ligne d'adresse
- Y a t'il une donnée dans le cache ?
  Besoin d'un bit de validité par ligne de cache
- Structure de l'entrée du cache

tag	index	décalage ligne	décalage mot

Implémentation d'un cache à correspondance directe

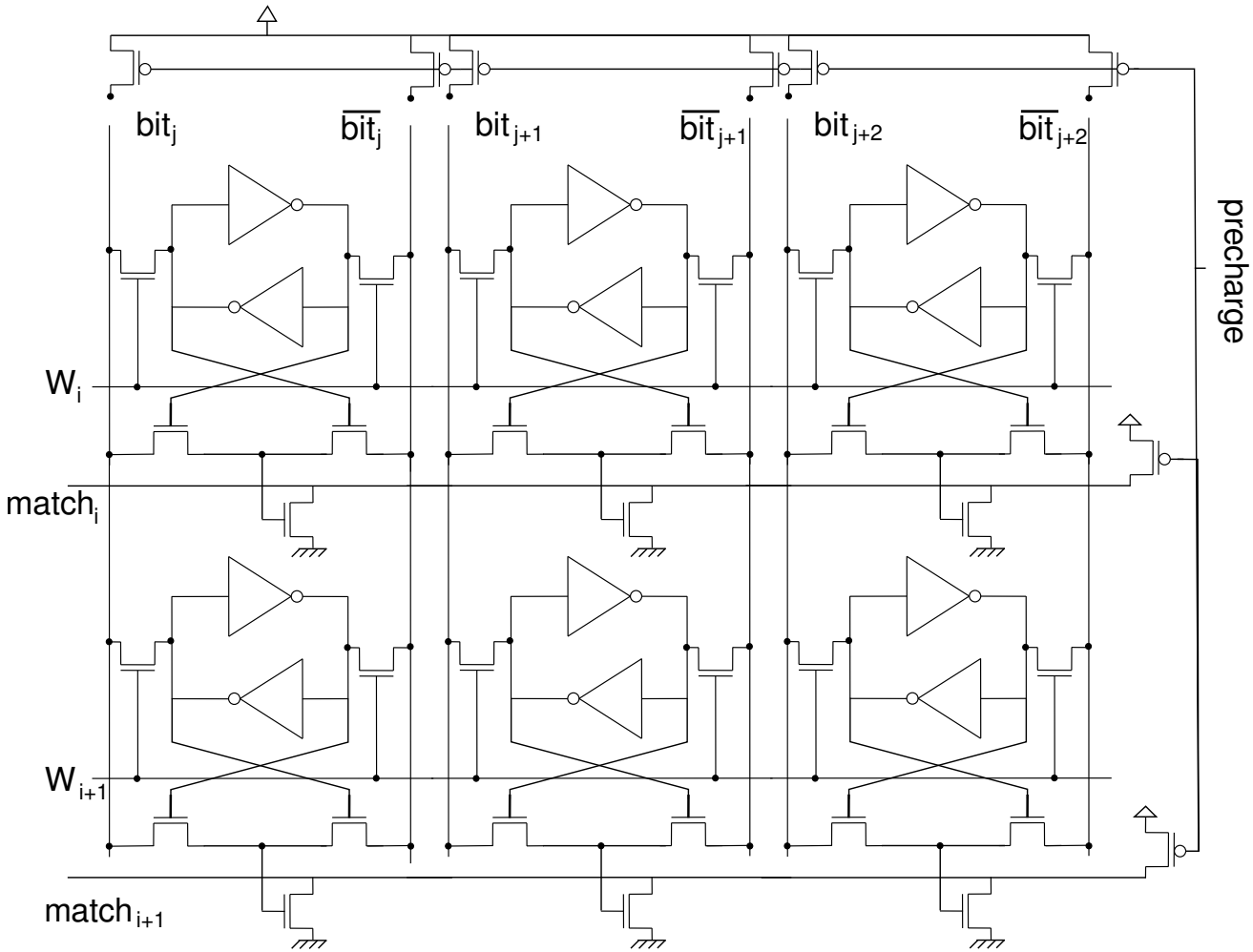


Implémentation d'un cache associatif par ensemble à 2 voies



Cache complétement associatif

- Ligne placée n'importe où dans le cache
- Un comparateur par ligne ?
- Changement de point mémoire ⇒ CAM (Mémoire Adressable par son contenu)



Comment faire de la place dans le cache ?

- Pour un cache associatif, quelle ligne évincer en cas de défaut ?
- Stratégie de remplacement optimale : oracle
  Evincement de la ligne non utilisée pour la plus grande période de temps (passée et future)
- Stratégies de remplacement réelles :
  - Aléatoire
    Pas cher (compteur ou générateur pseudo-aléatoire) , efficace si l'associativité est grande ou si le pire cas importe
  - FIFO : évincement de la ligne insérée la première
    Repose sur le principe de localité temporelle
    pas trop cher, pas trop efficace
  - LFU (Least Frequently Used)
    Repose sur le principe de localité temporelle
  - LRU (Least Recently Used)
    Algorithme optimal sans la connaissance du futur

Implémentations de LRU

- $N!$ ordre pour un cache associatif à N voies
- Codage optimal nécessitant $\lceil lg(N!) \rceil$ bits de plus par ensemble
- Dur à exploiter en pratique $\Rightarrow$ codage sous-optimal ou approximation
  - Codage en matrice triangulaire NxN
  - Codage en pile
  - Pseudo LRU arbre
  - Pseudo LRU - basé MRU

Codage en matrice triangulaire

- c_{ij} bit indiquant si l'instant du dernier usage de la voie i est postérieur à celui de la voie j dans le même ensemble
- $\frac{N(N-1)}{2}$ bits placés dans une matrice triangulaire NxN
- Accès à la voie $k \Rightarrow$ tous les bits de la ligne k à 1 et tous ceux de la colonne k à 0 $\Rightarrow c'_{ij} = \overline{hit_j}.(hit_i + c_{ij})$
- Voie LRU $\Leftrightarrow$ que des 0 sur sa ligne et que des 1 sur sa colonne
- Intéressant pour $N \leq 4$

3				
2				c_{23}
1			c_{12}	c_{13}
0		c_{01}	c_{02}	c_{03}
	0	1	2	3

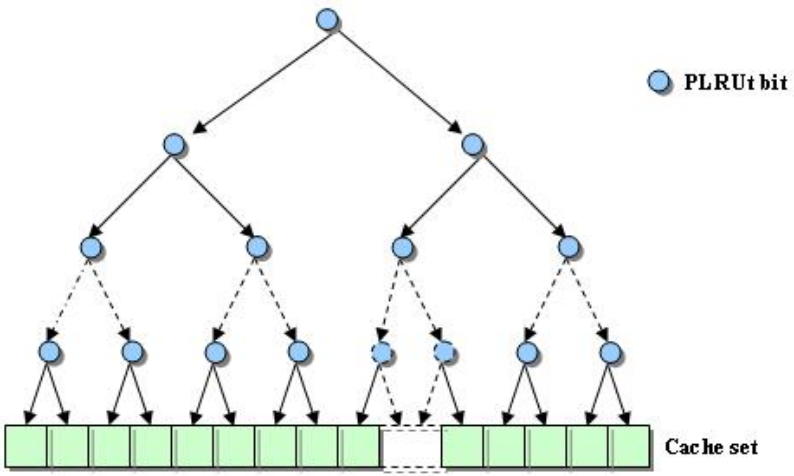
$$LRU_0 = \overline{c_{01}}.\overline{c_{02}}.\overline{c_{03}}$$
$$LRU_1 = c_{01}.\overline{c_{12}}.\overline{c_{13}}$$
$$LRU_2 = c_{02}.c_{12}.\overline{c_{23}}$$
$$LRU_3 = c_{03}.c_{13}.c_{23}$$

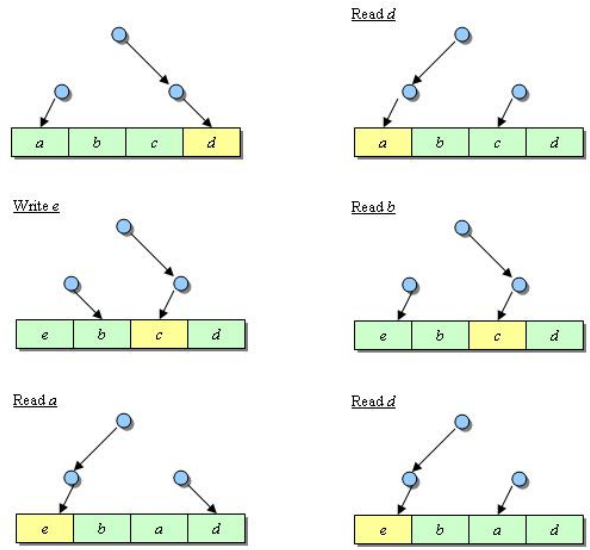
Codage en pile

- Une pile ordonnée selon les accès pour chaque ensemble
- Besoin de $N.lg(N)$ bits/ensemble mais plus couteux que le codage triangulaire pour $N \leq 4$
- Schéma au tableau

Pseudo LRU arbre

- Utilisation d'un ordre partiel fidèle sur le bas de la liste
- Utilisation d'un arbre binaire qui désigne l'élément à éliminer: $N-1$ bits par ensemble
- En cas de hit sur une voie, on inverse les flèches emmenant vers cette voie.
- Assez efficace quand N est grand
- Utilisé dans de vrais caches (PowerPC)





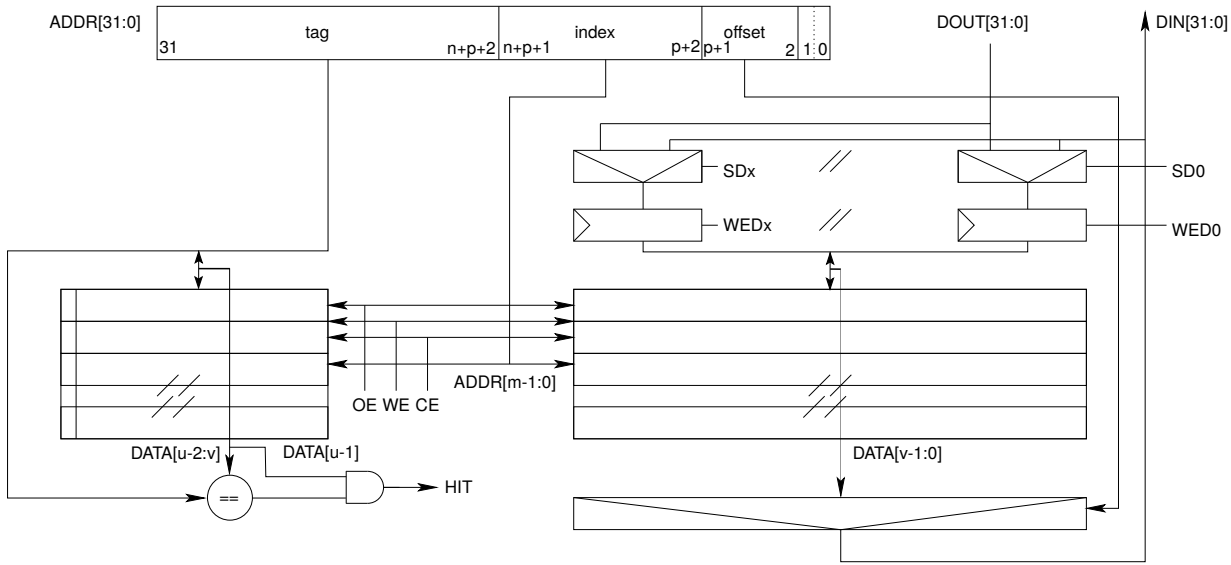
Pseudo LRU - basé MRU

- N bits par ensemble, 1 bit par voie
- Utilisation en MRU (Most Recently Used) :
  - $MRU_i = 1$ si accès à la voie i de l'ensemble
  - $\forall j, MRU_j = 1 \Rightarrow \forall j \neq i, MRU_j = 0$
  - Evincement de la première à 0
- Marche très bien malgré son faible coût
- Utilisé dans de nombreux caches (ARM)

Généralités sur l'écriture

- Moins d'écritures que de lecture (ex benchmark MIPS: 10% de SW et 26% de LW)
- Écriture généralement non bloquante
- Taille de l'écriture variable (1 à 8 octets) $\neq$ lecture par ligne

Implémentation :



Write through vs write back

- 2 stratégies d'écriture vis-à-vis de la hiérarchie mémoire
- Write through :
  - Donnée systématiquement écrite dans la ligne de cache et dans le niveau mémoire inférieur
  - + : simple, pas de soucis de cohérence de données dans la hiérarchie
- Write back :
  - Donnée écrite dans la ligne de cache
  - Donnée écrite dans le niveau mémoire inférieur uniquement lors du remplacement de la ligne
  - Besoin d'un bit indiquant que la ligne n'est plus propre (dirty bit)
  - + : réduit les accès inutiles à la mémoire (traffic, consommation)

Allocation et écriture

- Que fait-on en cas de défaut de cache durant une écriture ?
- 2 stratégies :
  - Write allocate : allocation d'une ligne dans le cache et écriture dedans
  - No write allocate : écriture seulement dans le niveau mémoire inférieur
- Combinaisons classiques : WT-WNA, WB-WA

Pistes pour l'optimisation d'un cache

Average memory access time = Hit time + Miss rate x Miss penalty

- Réduire la pénalité de défaut (cache multiniveau)
- Réduire le temps de succès
- Réduire le taux de défaut

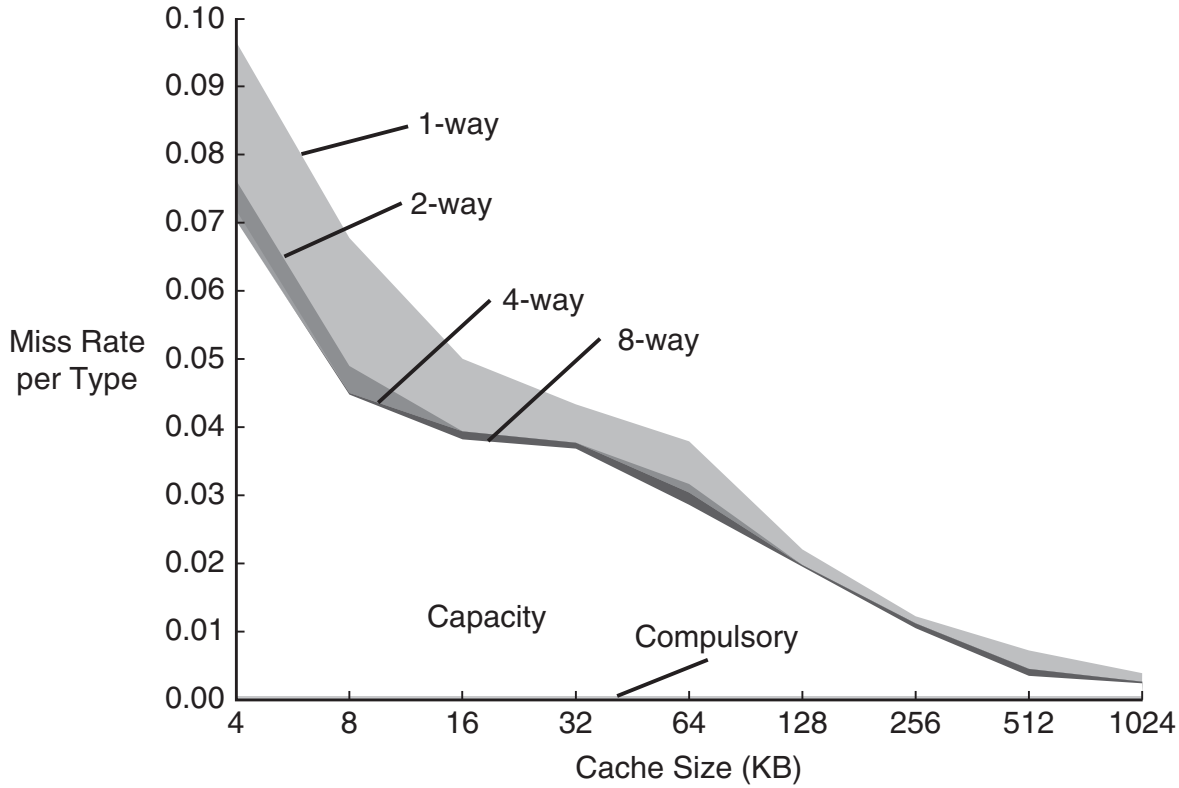
Réduction du temps de succès

- Chemin critique typique : lecture de la mémoire contenant le tag + comparaison au tag de l'adresse
- Seules options architecturales :
  - Cache petit
  - Cache simple
  - Impact négatif sur le taux de défaut
- Autres options : prédiction

Réduction du taux de défaut

Raisons d'un défaut (3C) :

- Compulsory/obligatoire :
  défaut de 1^{re} référence
- Capacité :
  défaut disparaissant dans un cache plus grand
- Conflictuel :
  défaut disparaissant dans un cache complétement associatif



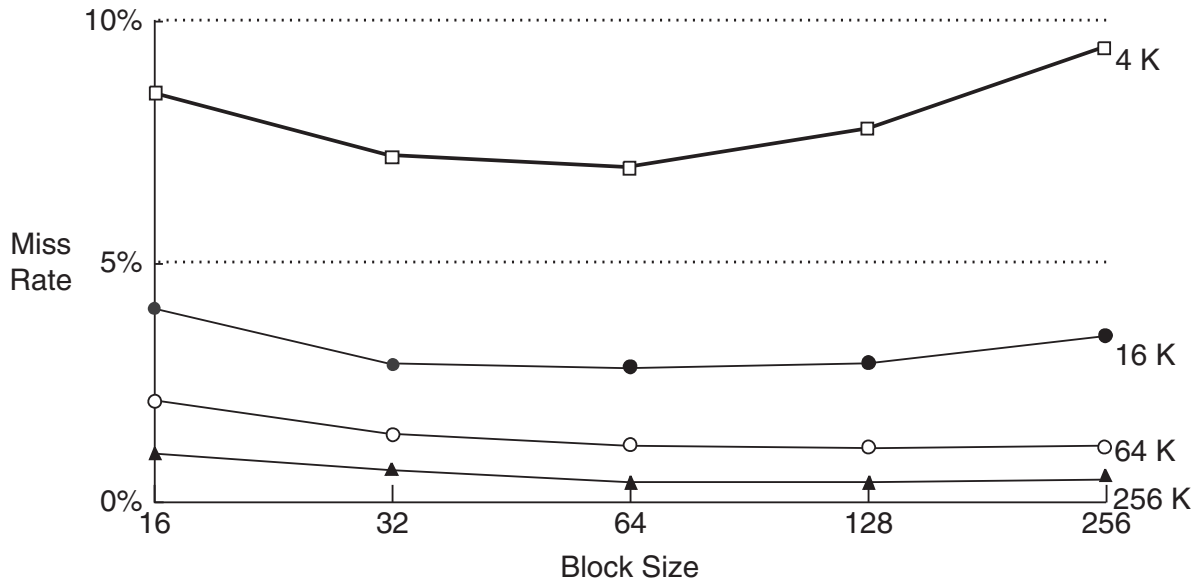
Optimisations :

- Plus grand cache $\Rightarrow$ - sur hit time
- Plus grande associativité $\Rightarrow$ - sur hit time
  - Cache DM de taille M $\Leftrightarrow$ cache 2-way SA de taille M/2 (pour M<128ko)

Jouer sur la taille d'une ligne

Impact d'une augmentation d'une ligne de cache

- ↘ les défauts obligatoires (localité spatiale)
- ↗ les défauts conflictuels et capacitifs
- ↗ la pénalité de défaut



Optimisations logicielles

Le compilateur peut réduire le taux de défaut en :

- Réarrangeant le code et les données
  E.g. alignement du code ou des données sur les lignes du cache
- Échange de boucles
- Blocking