جامعة باجي مختار -عنابة
Université Badji Mokhtar-Annaba

جامعة باجي مختار - عنابة
Faculté des Sciences
de l'Ingéniorat

M2 - SEM

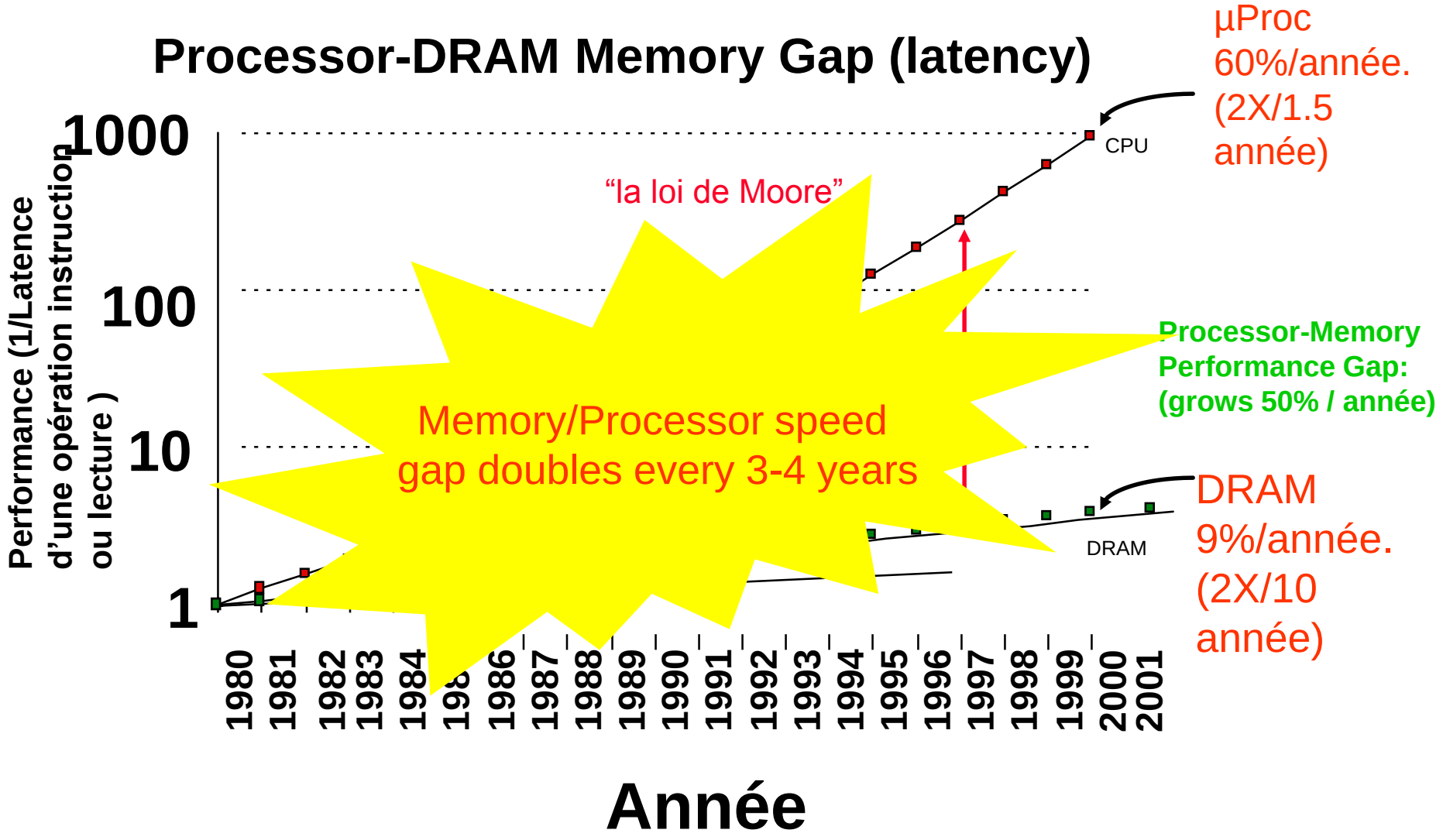
Concepts avancés d'Architecture

Cours 2.2 : LA MÉMOIRE CACHE

ANNÉE : 2020-2021

Pr R. BOUDOUR

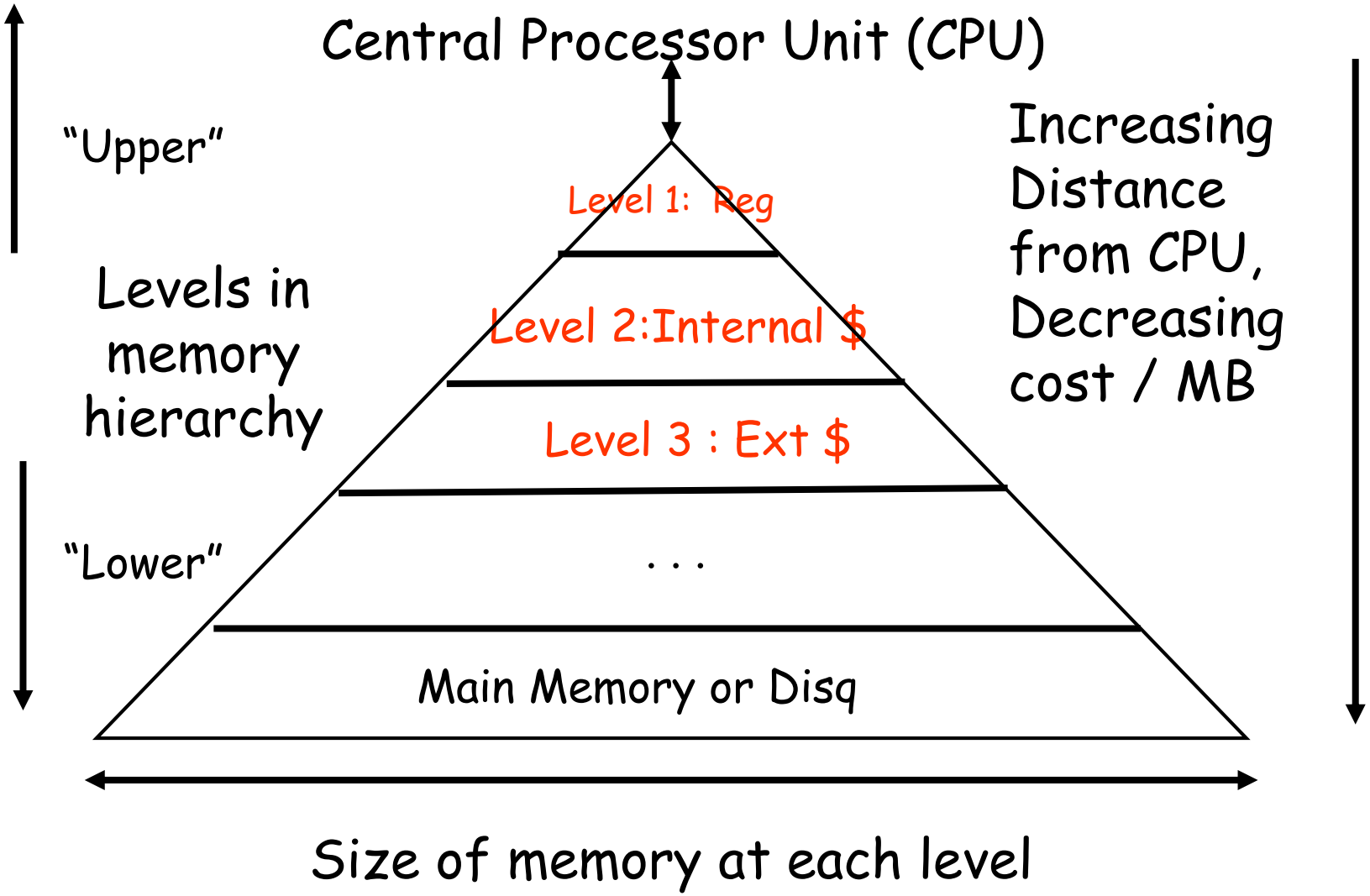
TROU entre la rapidité processeur & rapidité mémoire :



La mémoire devient le frein à l'augmentation des performances des processeurs : Mur mémoire

- Dans un système conventionnel, il n'y a pas qu'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) ou implicite (ex : défaut de cache)**

Memory Hierarchy



Notre objectif :
Donner à l'utilisateur l'illusion d'avoir la capacité de la mémoire la moins chère et la vitesse la plus rapide (cache interne)

- **Constat :**
  - Mémoires larges sont lentes,
  - Mémoires rapides sont petites.
- **Comment créer des mémoires de grandes tailles, rapides et pas chères ?**
  - **Réponse** : Hiérarchie de plusieurs niveaux de mémoires pouvant fonctionner simultanément.

La mémoire cache

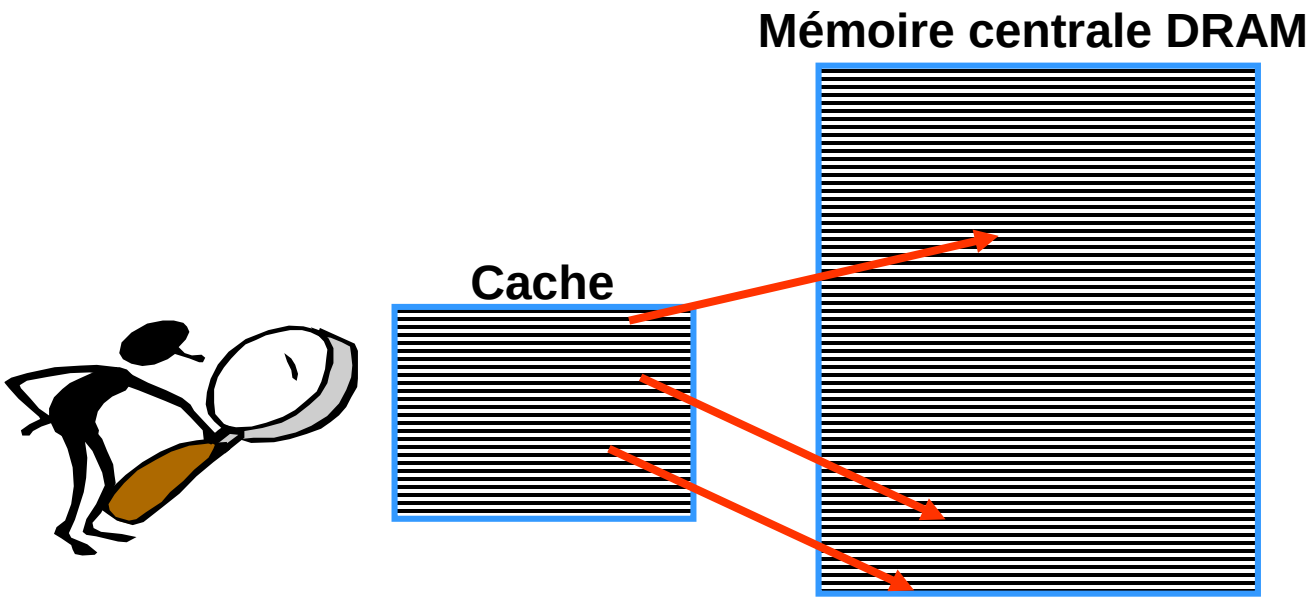
- **Un cache** : une copie de certains blocs de données présents en mémoire centrale (DRAM).
- **Cache Vs DRAM** : Rapide(10X) mais plus petite (100X)
- **Un bloc** = Une ligne de cache = plusieurs mots mémoire consécutifs en mémoire.
- **Succès/Echec**
  - Toutes les opérations mémoires (lecture et écriture) sont dirigées d'abord vers le cache pour tester si la donnée est présente.
  - Deux cas possibles :
    - **Donnée présente** : Succès ou hit, la donnée est délivrée au processeur
    - **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 = (nombre de succès) / (nombre d'accès total), $0 \leq h \leq 1$

La mémoire cache

Dans ce cours, on parle des données en cache, mais on peut aussi avoir des instructions en mémoire cache. Données = Données et instructions



Principe de fonctionnement du cache

- Cache fournit de façon transparente les données qui ont été chargées dans le passé.
  - Données déjà utilisées dans un passé proche.
  - Transparente : sans que le programmeur de bas niveau le demande.
  - On paye une seule fois la pénalité de la lenteur mémoire, même si la donnée est utilisée plusieurs fois.

Principe de fonctionnement du cache

- **Pre-fetching** : chargement à l'avance des données /instructions dont le processeur va en avoir besoin.
- Permet d'augmenter le taux de succès du cache
- Les algorithmes de pre-fetching sont basés sur le principe de la localité
  - **Localité temporelle** : garder dans le cache les dernières données manipulées par le programme.
  - **Localité spatiale** : charger en avance les données/instructions contiguës à une donnée/instruction référencée.

La localité des données

Principe de localité :

- Localité temporelle : Les données récemment utilisées (référencées) par le processeur seront probablement utilisées bientôt.
  - Il faut retenir (stocker) une donnée qui a été récemment utilisée.
- Localité Spatiale : Les données voisines de la donnée actuellement utilisée seront probablement bientôt utilisées.
  - 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)

Correspondance lignes cache/mémoire

La mémoire cache contient des **lignes** de mots de la mémoire centrale.

Exemple :

- Lignes de cache de 32 octets
- Mémoire cache de 512 Ko : 16384 lignes
- Mémoire centrale de 128 Mo
  - doit être gérée par les 512 Ko de cache et ses 16384 lignes.
  - 1 ligne du cache / 8192 octets (256 lignes) de la mémoire centrale.

Quelle est la relation entre les lignes du cache et celle de la mémoire centrale ?

Correspondance lignes cache/mémoire

Trois méthodes pour gérer la correspondance entre lignes du cache et lignes de la mémoire centrale :

- Correspondance directe (Direct mapping)
- Correspondance associative totale (Fully associative mapping)
- Correspondance associative par ensemble (N-way set associative mapping)

Correspondance directe

Une ligne mémoire ne peut aller que dans une ligne du cache

L lignes de cache

- la ligne d'adresse j de la mémoire centrale est gérée par la ligne i du cache
  - $i = j \mod L$
  - A partir de l'adresse d'une ligne mémoire, on sait dans quelle ligne du cache elle doit se trouver.

- une ligne de cache étant partagée par plusieurs lignes de la mémoire centrale, on garde donc l'information sur la donnée effectivement présente dans le cache

Correspondance directe

- On sait immédiatement où aller chercher la ligne (accès rapide)
- Nombreux défauts de cache conflictuels si on accède à des lignes de la mémoire centrale qui correspondent toutes à la même ligne du cache, tandis que d'autres lignes du cache ne sont pas utilisées.
- performances médiocres (taux de succès : 60-80 %)

Correspondance associative totale

Chaque ligne de la mémoire peut se trouver dans n'importe quelle ligne du cache.

Le cache contient le numéro de ligne de la mémoire centrale de chacune des informations.

- ⊕ grande souplesse d'utilisation permettant d'augmenter le nombre de succès (90-95 % de succès)
- ⊖ temps de comparaison plus long car adresse complète (temps d'accès plus long)

Correspondance associative par ensemble

Combinaison des deux méthodes précédentes pour pallier leur défaut respectif.

- le cache est divisé en ensembles de lignes, chaque ensemble contenant *N* lignes.
- Chaque ligne de la mémoire centrale est affectée à un ensemble

  ensemble = *numéro de ligne* mod *nombre d'ensembles*

- A l'intérieur d'un ensemble, correspondance associative totale.

Tag: to identify the block

The « cache line » or « block » is the unit of data managed by the cache

Cache blocks and memory blocks have the same size

Etiquette

Associated with each line of cache other information:

- Tag : derived from the address of the cache which is used to determine whether a cache hit has occurred or not
- Valid bit : Cache line full (Valid) or free (not Valid)
- Modified Bit : indicating if the cache line has been modified (written)

Questions de conception

- Où peut-on placer un bloc dans le niveau supérieur ? (placement de bloc)
- Comment trouver un bloc s'il est présent dans le niveau supérieur ? (Identification de bloc)
- Quel bloc doit être remplacé en cas d'échec ? (remplacement de bloc)
- Qu'arrive-t-il lors d'une écriture ? (stratégie d'écriture)

Q1 : Mapping from Memory onto Cache

Placement algorithm determines how data in memory is mapped onto blocks in the cache

- The simplest scheme (Direct mapped cache DMC) restricts where in cache a given memory address can be located. Uses modulo addressing.

  1 memory adress associated with 1 $ block

- The most complex schemes (Fully associative cache FA) uses associative addressing which allows data from any address in memory to be placed in any block in cache

  1 memory adress associated with all $ blocks

Placement des données

Plusieurs stratégies :

- Direct mapping
- Fully associative mapping
- Set associative mapping ou n-way set assocoative

Direct mapping

Chaque donnée dans un seul emplacement (bloc) du cache

Calcul simple :

i = j modulo m

Où :

i : numéro de l'emplacement du cache

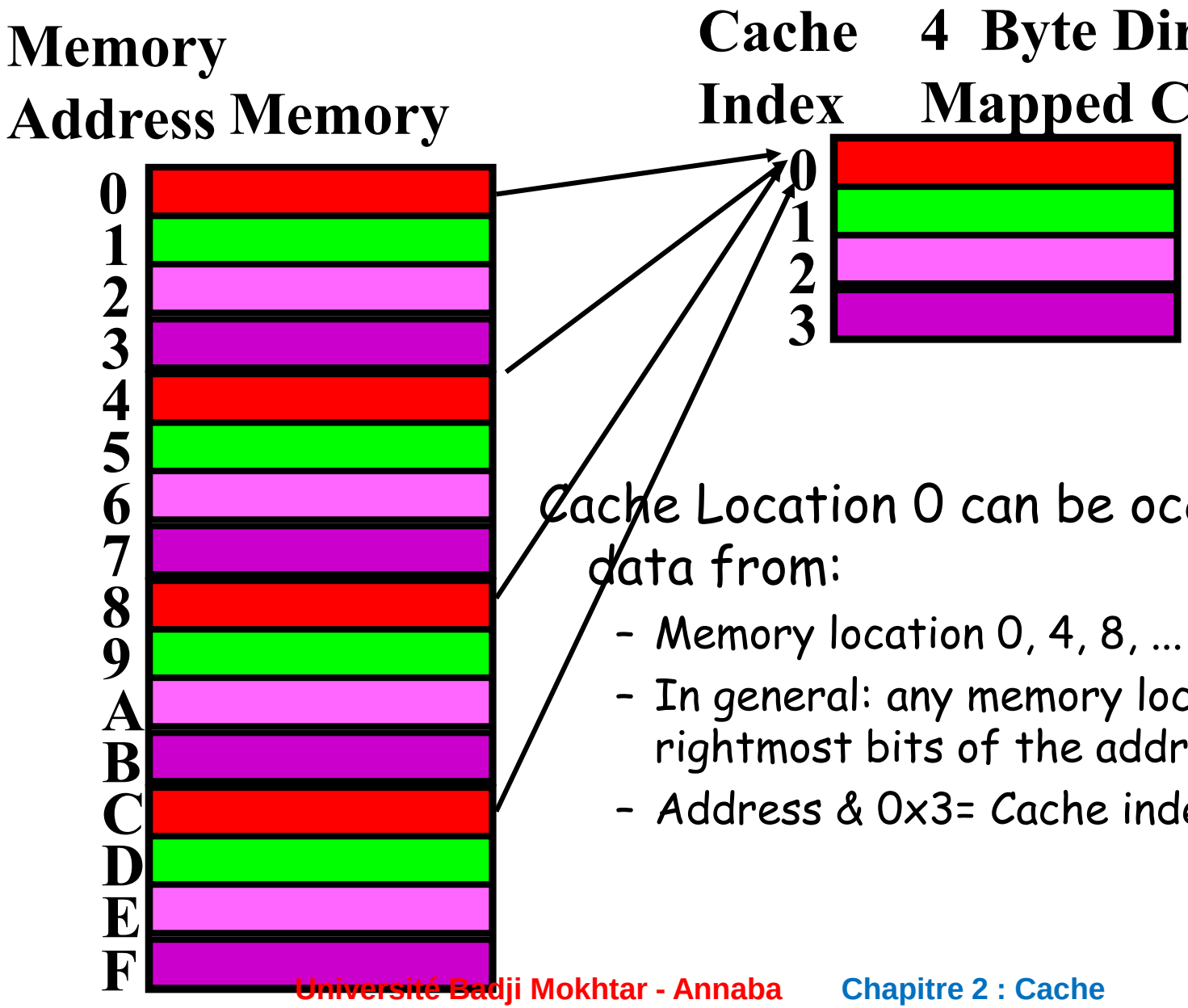
j : adresse mémoire (numéro bloc)

m : nombre d'emplacements du cache

Méthode

- L'index permet l'accès à un emplacement du cache
- Un emplacement contient une etiq. + une donnée
- L'étiquette obtenue est comparée à étiq. de l'adresse. Si les deux étiquettes sont :
  - les mêmes ⇒ Succès
  - différentes ⇒ Echec

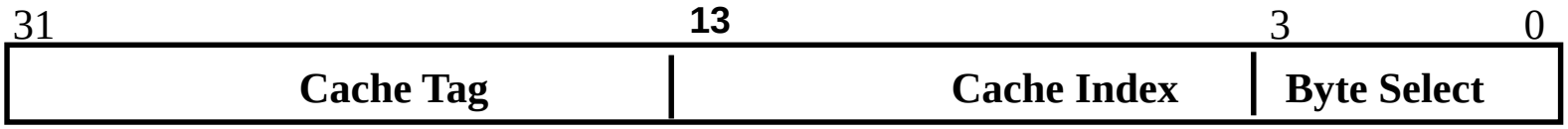
Simplest Cache: Direct Mapped



Cache Location 0 can be occupied by data from:

- Memory location 0, 4, 8, ...
- In general: any memory location whose 2 rightmost bits of the address are 0s
- Address & 0x3= Cache index

DMC



- 32-bit byte addressing
- Cache line 16 bytes
- Cache size is 16 KBytes,

✓First 4 bits of address : offset in the cache line,

✓Next 10 bits determine which block in cache the data must be stored

✓Remaining 18 bits of redundant addressing tells us what 256K possible memory blocks may be mapped to the same cache block.

The cache must store these 18 bits as a tag in order to know which of these blocks is currently stored there.

Accessing data in a direct mapped cache DMC

So lets go through accessing some data in a direct mapped, **16KB cache, 1 Block = 16 Bytes**

- 16 byte blocks x 1024 cache blocks

3 Addresses divided (for convenience) into Tag, Index, Byte Offset fields

```
00000000000000000 0000000001 0100
00000000000001000 0000000011 0100
00000000000000010 0000000001 0100
```

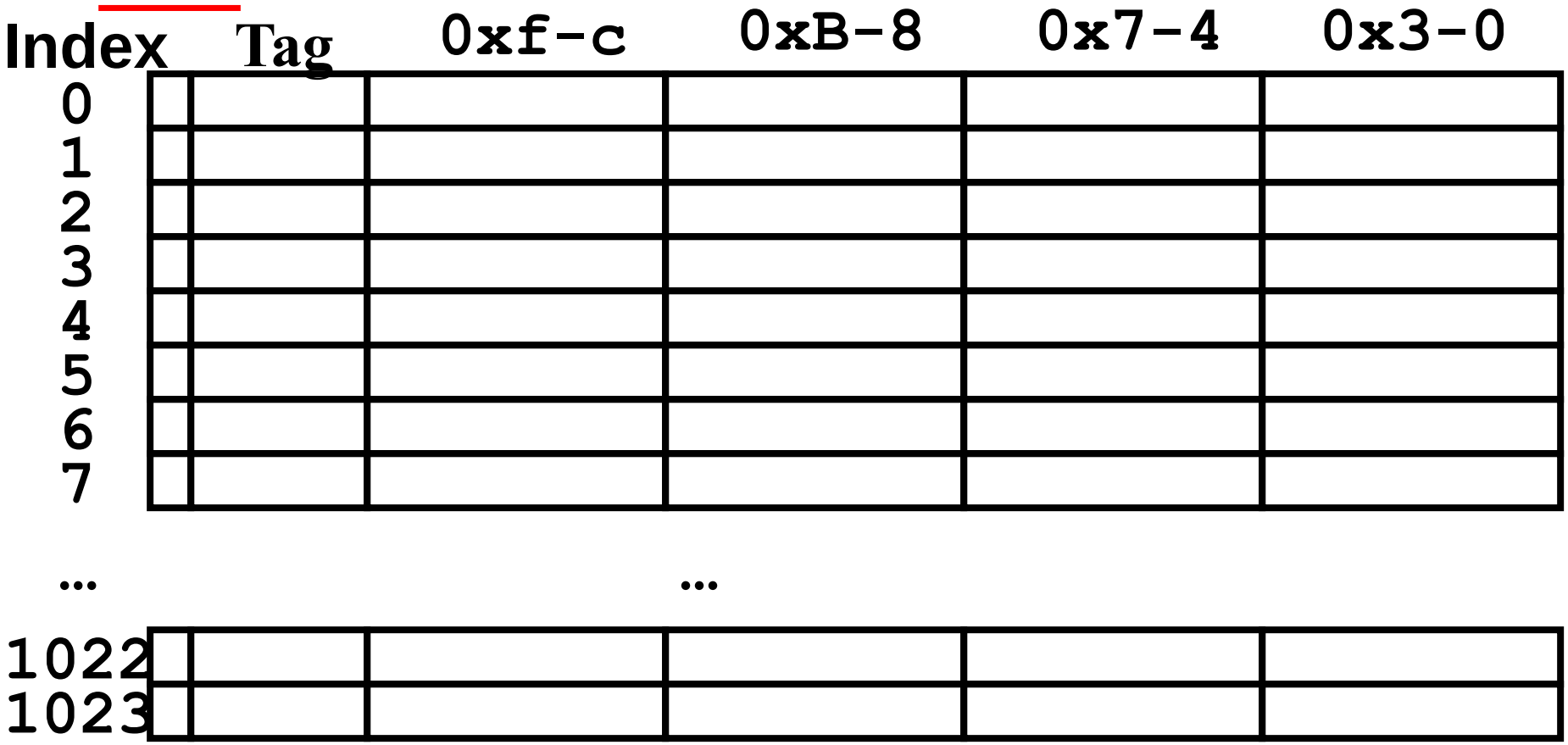
Tag — 18 bits

Index — 10 bits

Offset — 4 bits — Numéro de l'octet dans le bloc

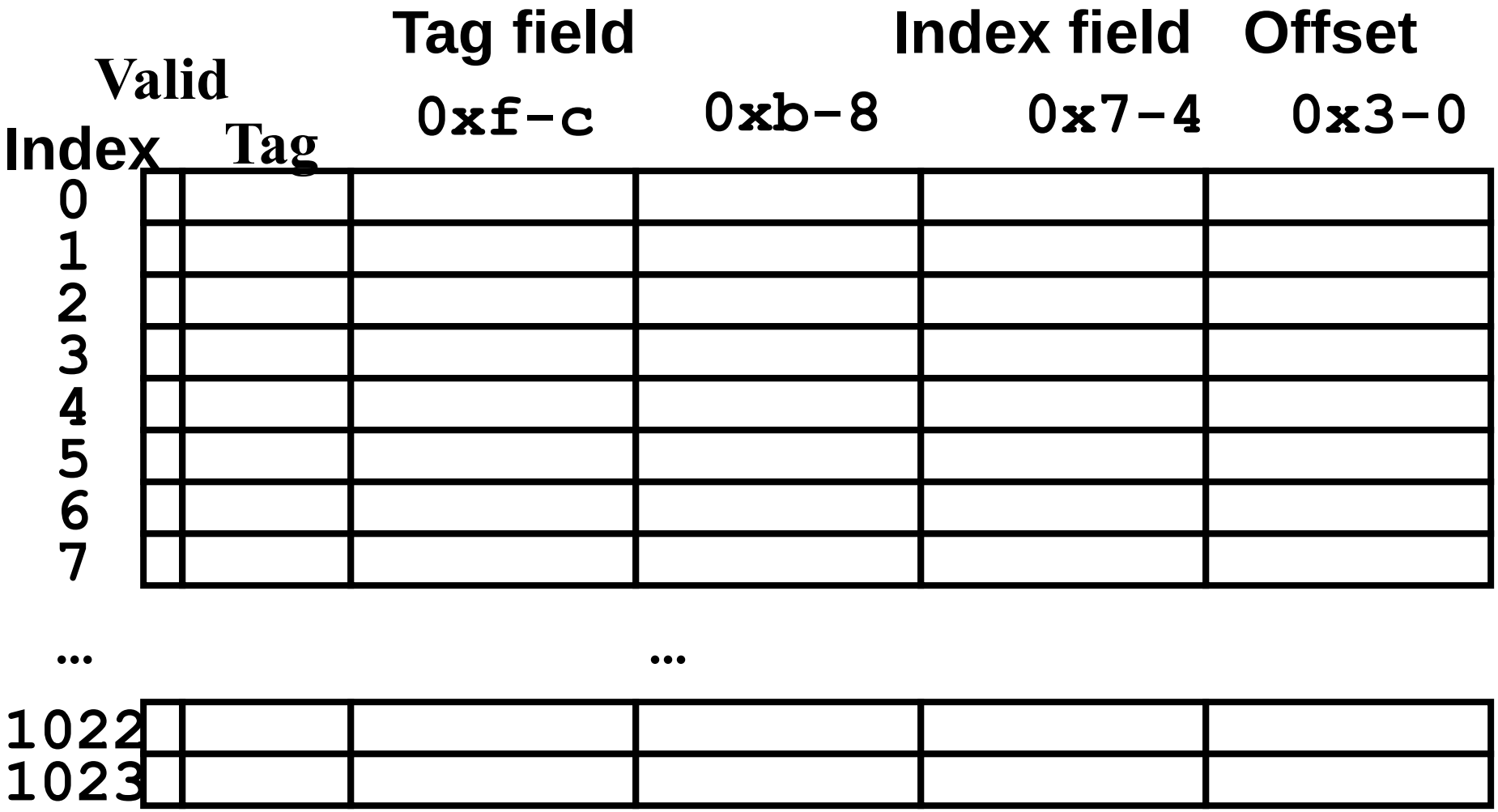
16 KB Direct Mapped Cache, 16B blocks

Valid bit ⇒ no address match when "power on" cache
(Not valid ⇒ no match even if tag = addr)



Read 0000000000000000 0000000001 0100

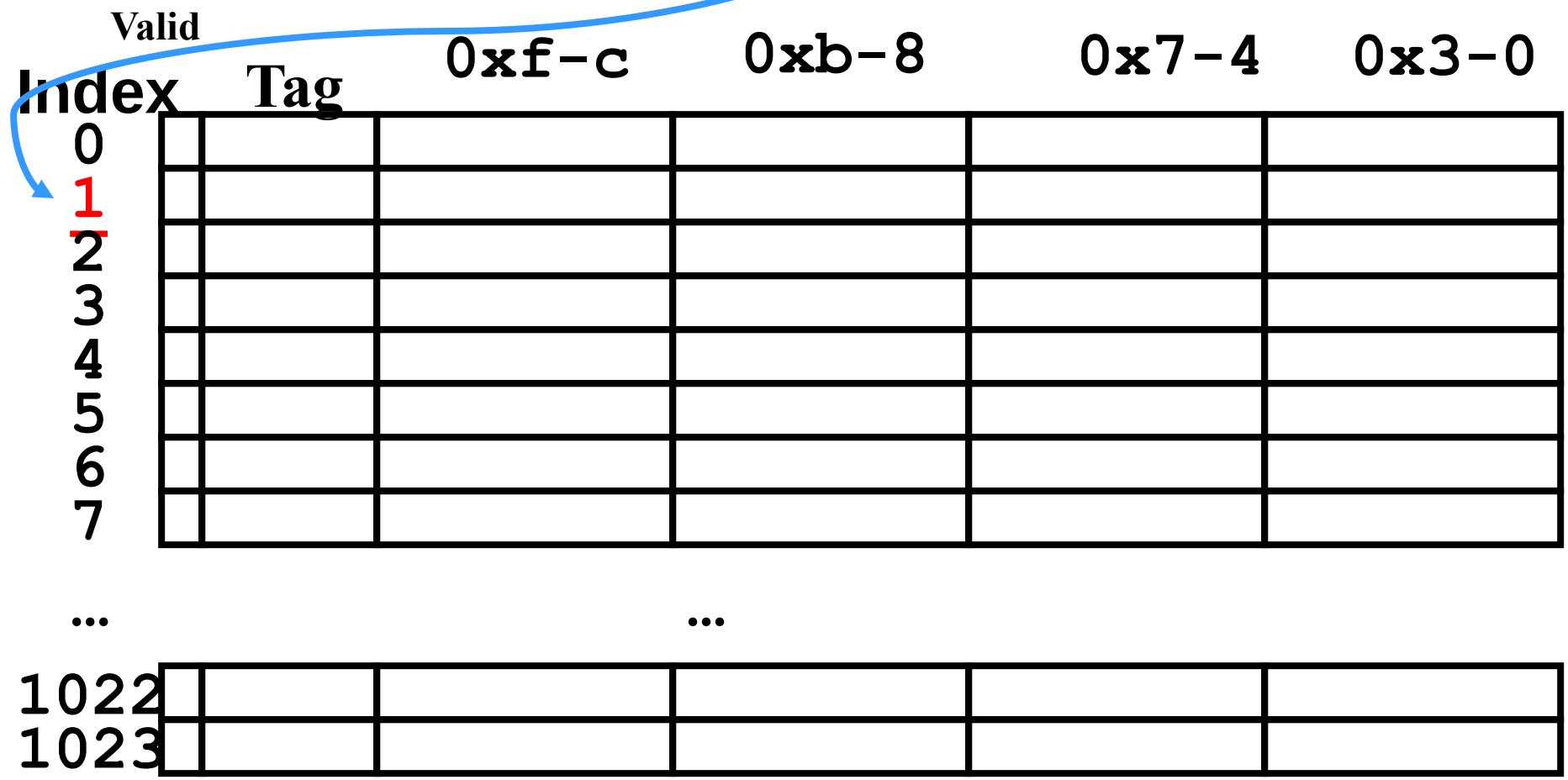
0000000000000000 0000000001 0100



So we read block 1 (0000000001)

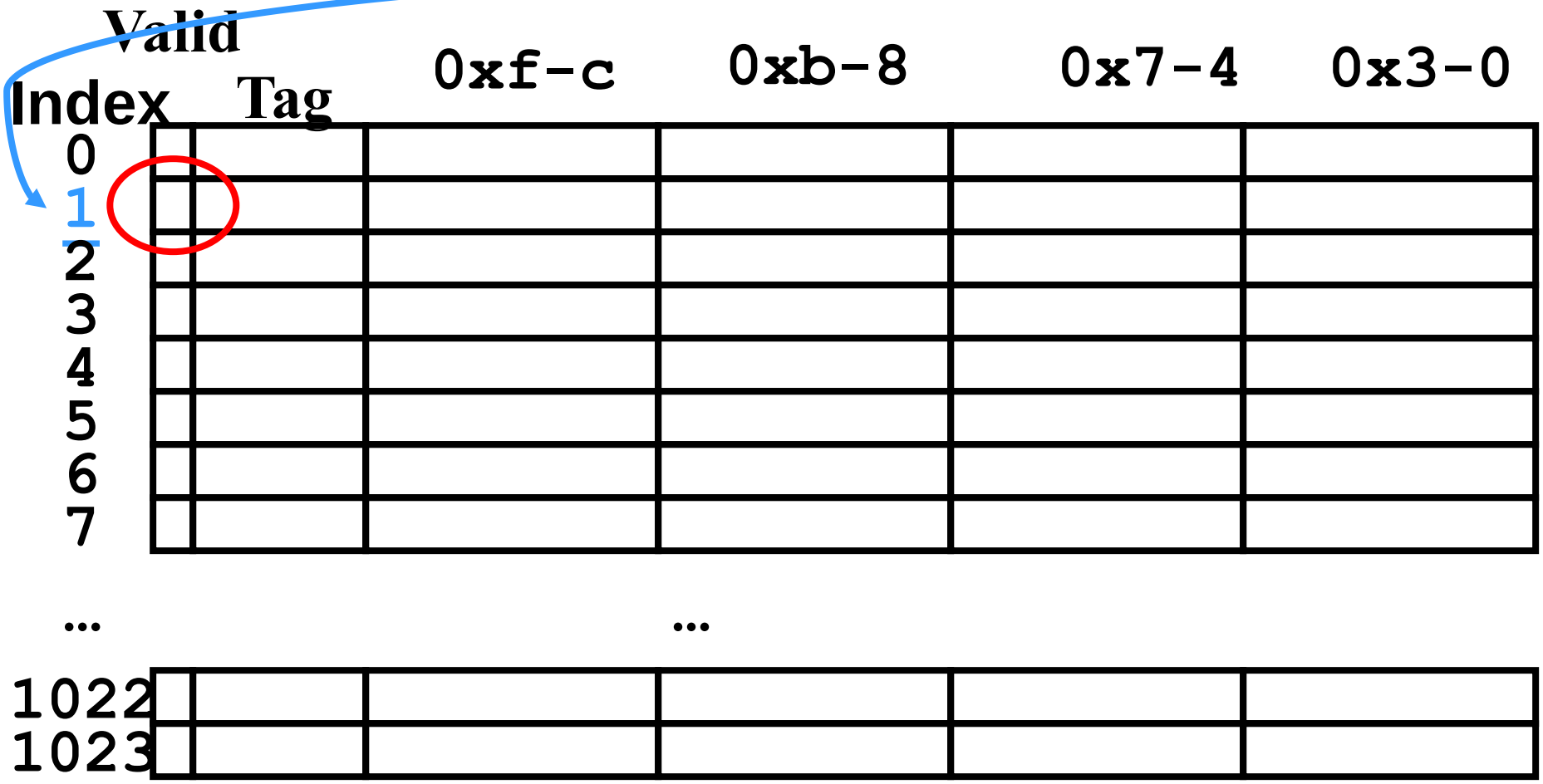
0000000000000000 0000000001 0100

Tag field Index field Offset



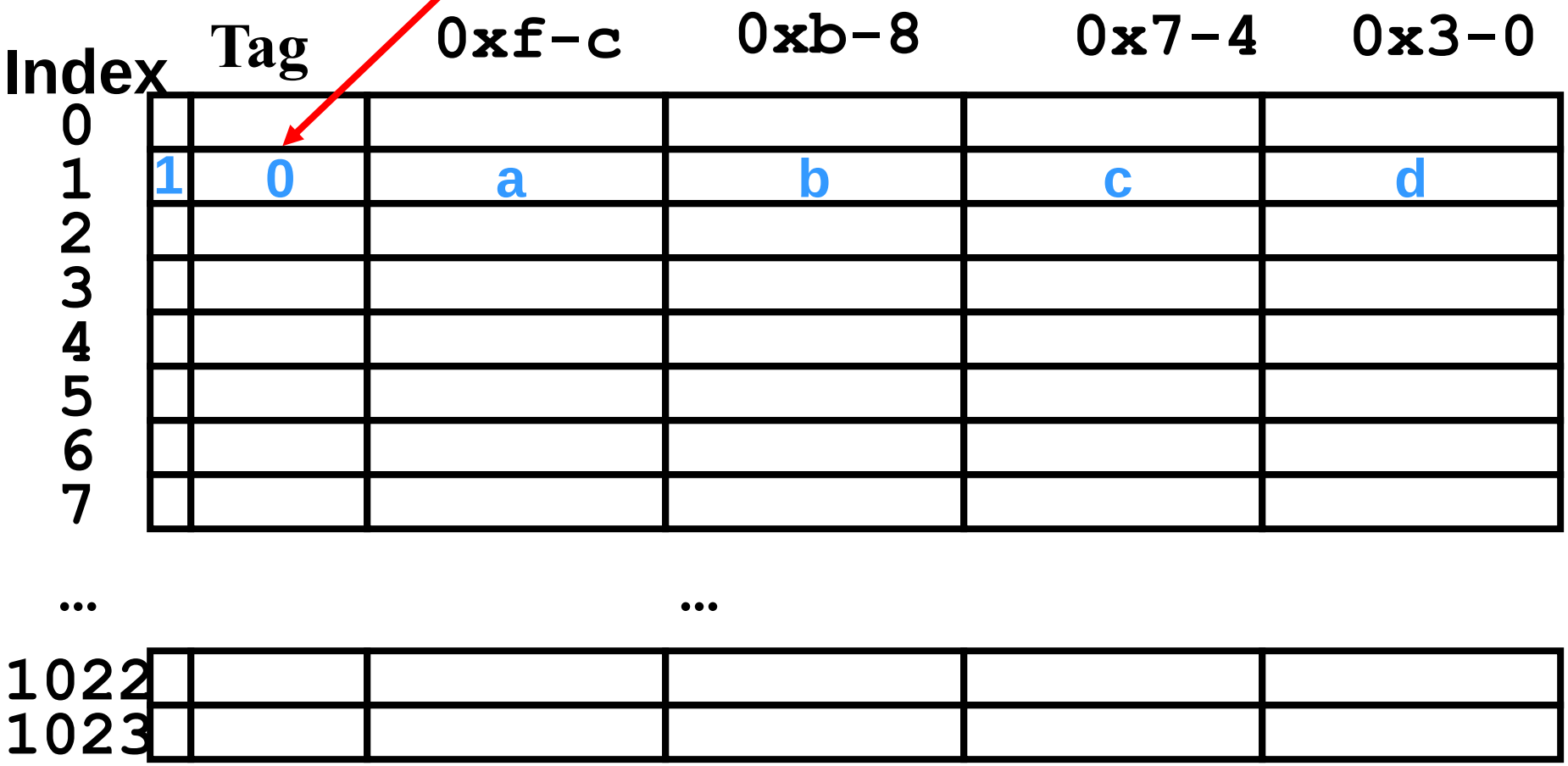
No valid data => Miss (cold)

00000000000000000000 0000000001 0100



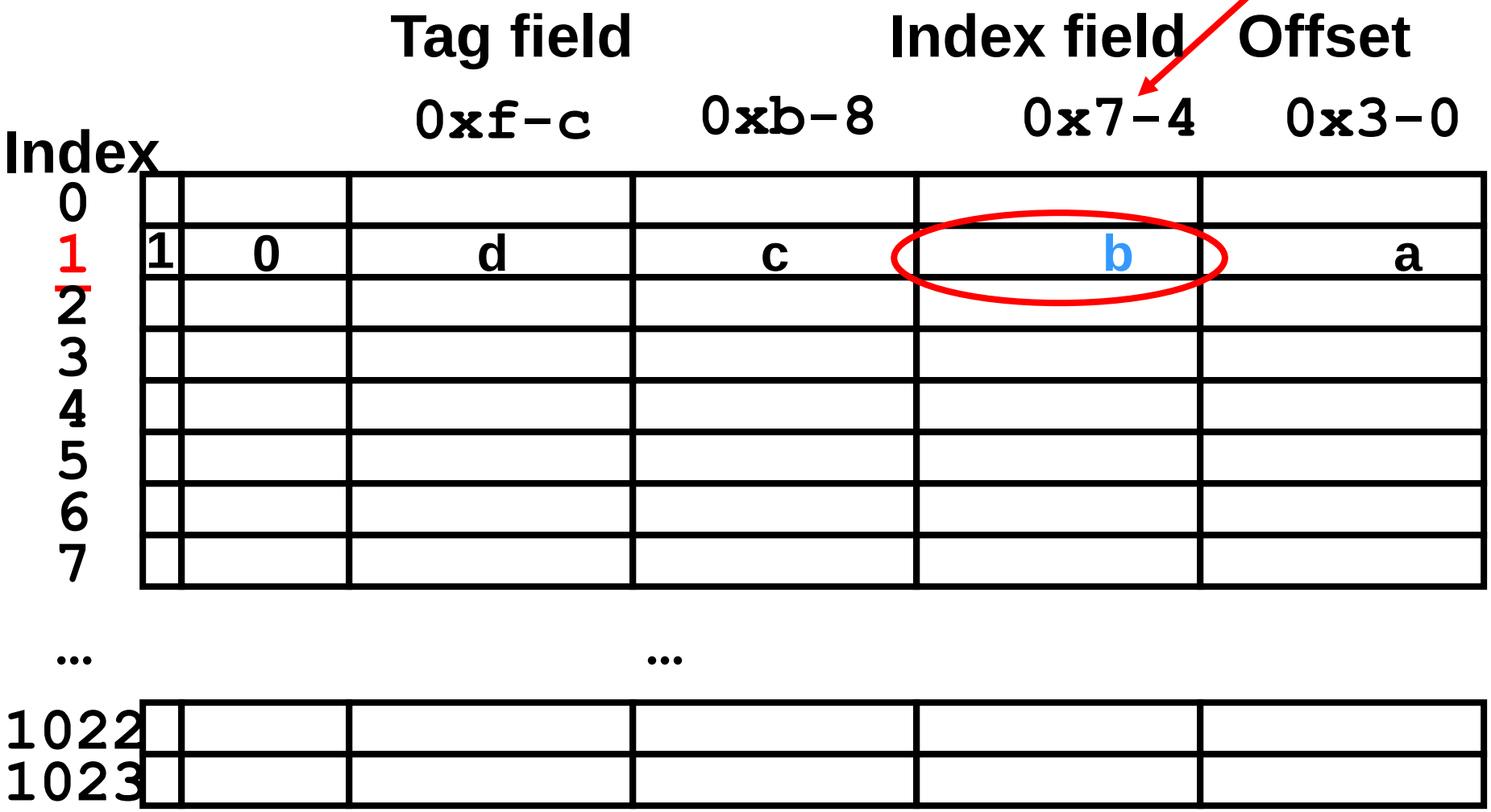
So load that data into cache, setting tag, valid

00000000000000000000 0000000001 0100



Read from cache at offset, return word b

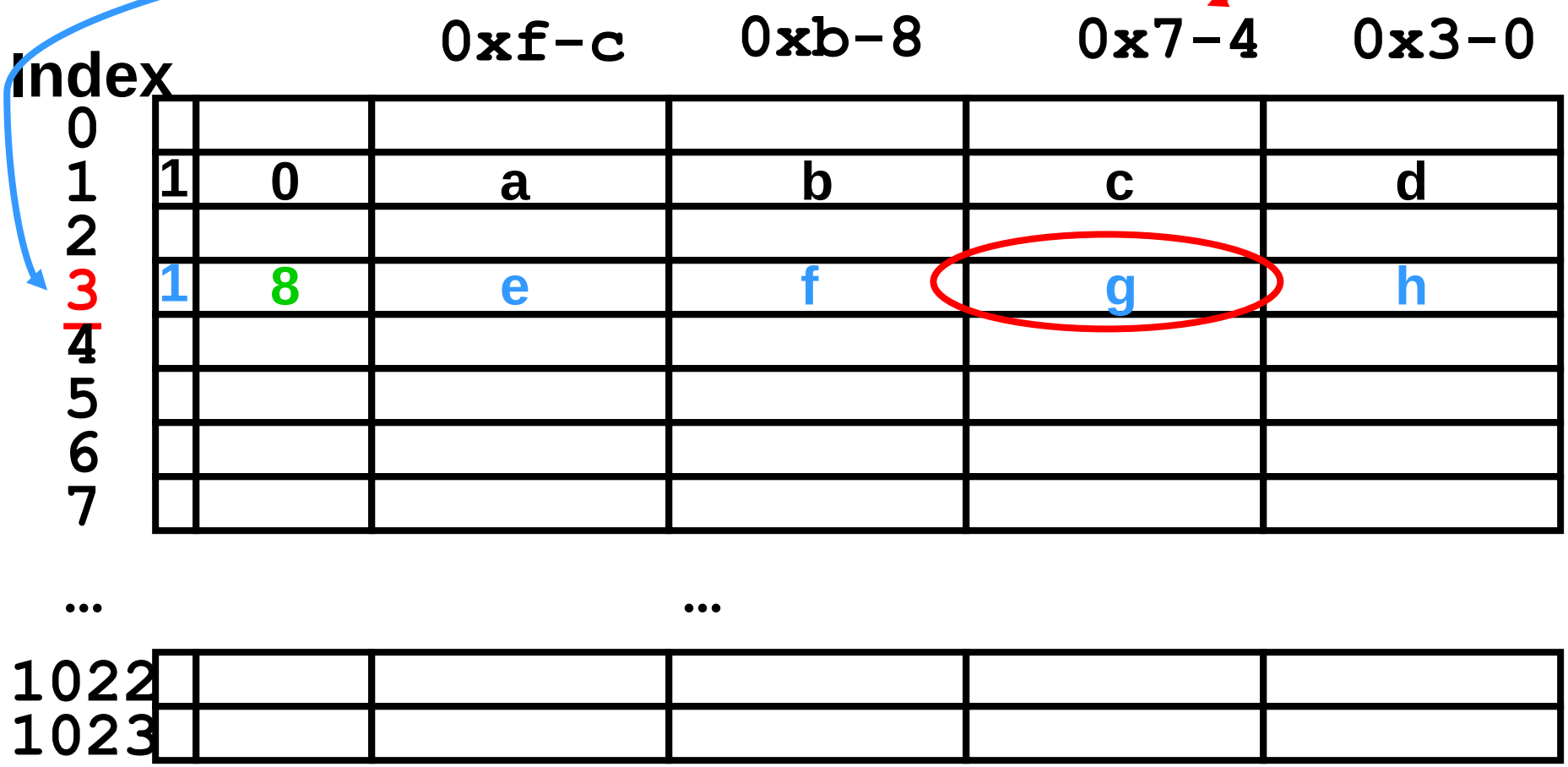
0000000000000000000 0000000001 0100



Load cache block , return word g

0000000000000001000 0000000011 0100

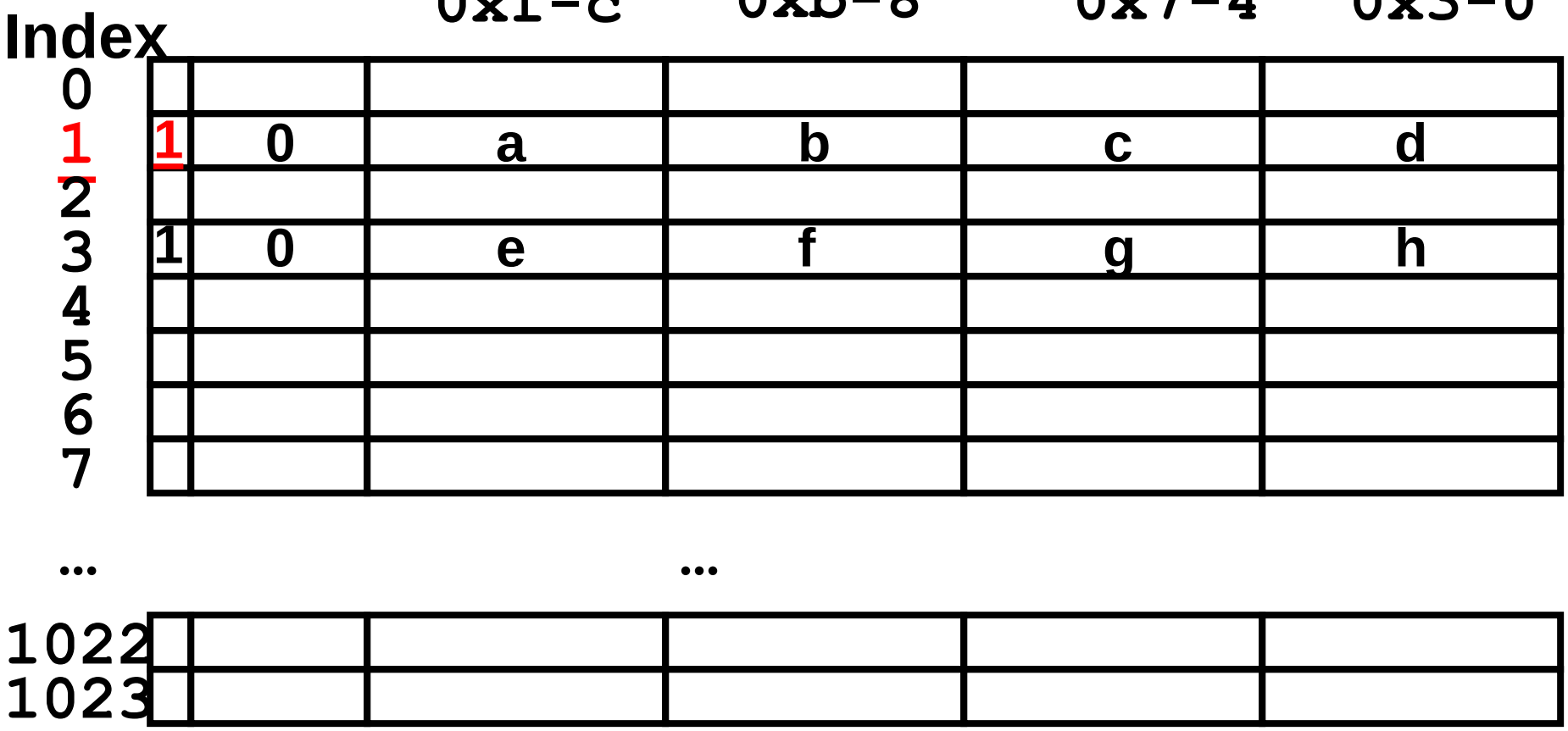
Index field Offset



So read Cache Block 1, Data is Valid

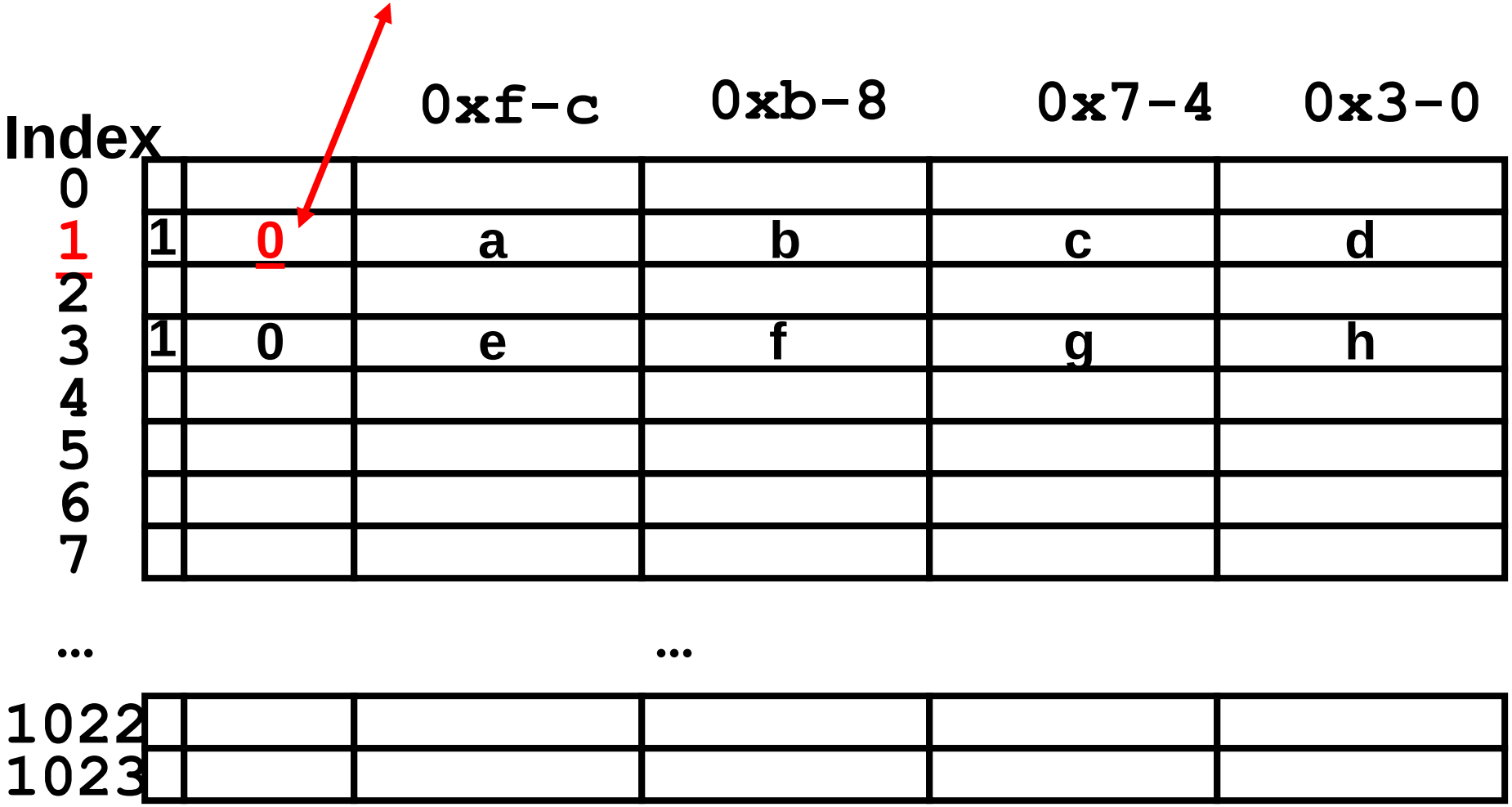
0000000000000010 0000000001 0100

Tag field | Index field | Offset



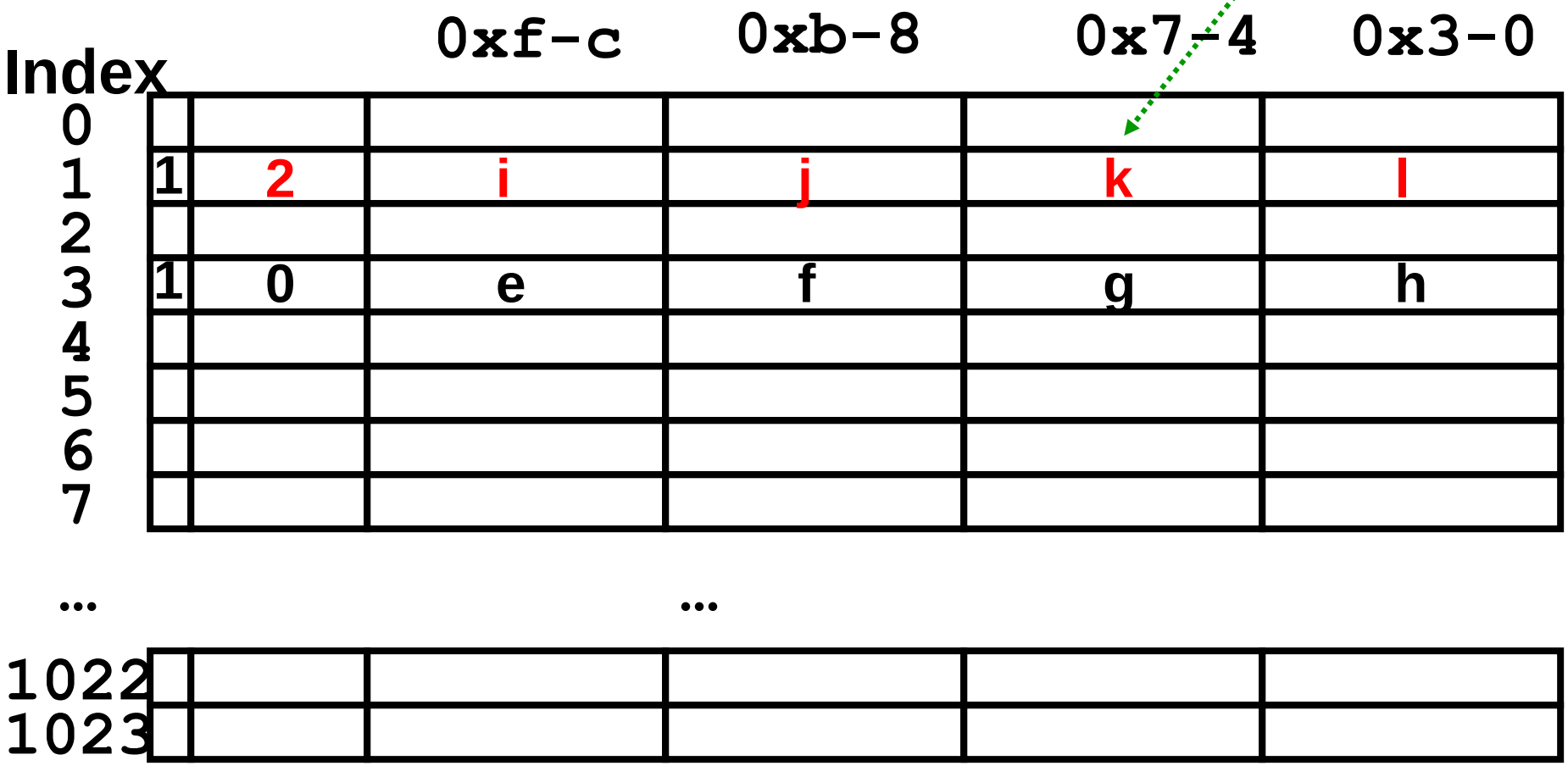
Cache Block 1 Tag does not match (0 != 2)

0000000000000000010 0000000001 0100

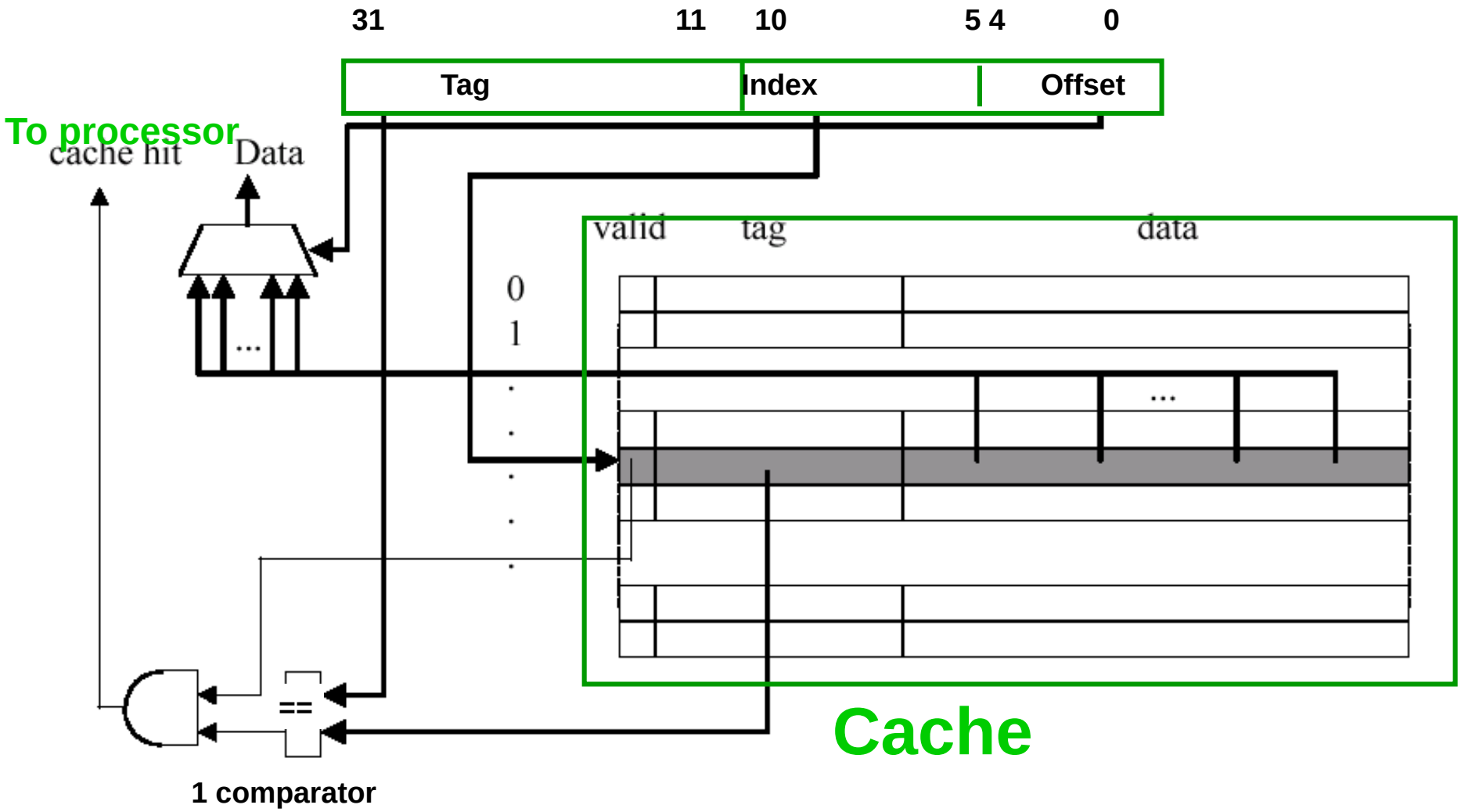


Miss, replace block 1 with new data & tag

0000000000000000010 0000000001 0100



DMC


31
11
10
5 4
0
Tag
Index
Offset
To processor
cache hit
Data
valid
tag
data
0
1
...
==
Cache
1 comparator

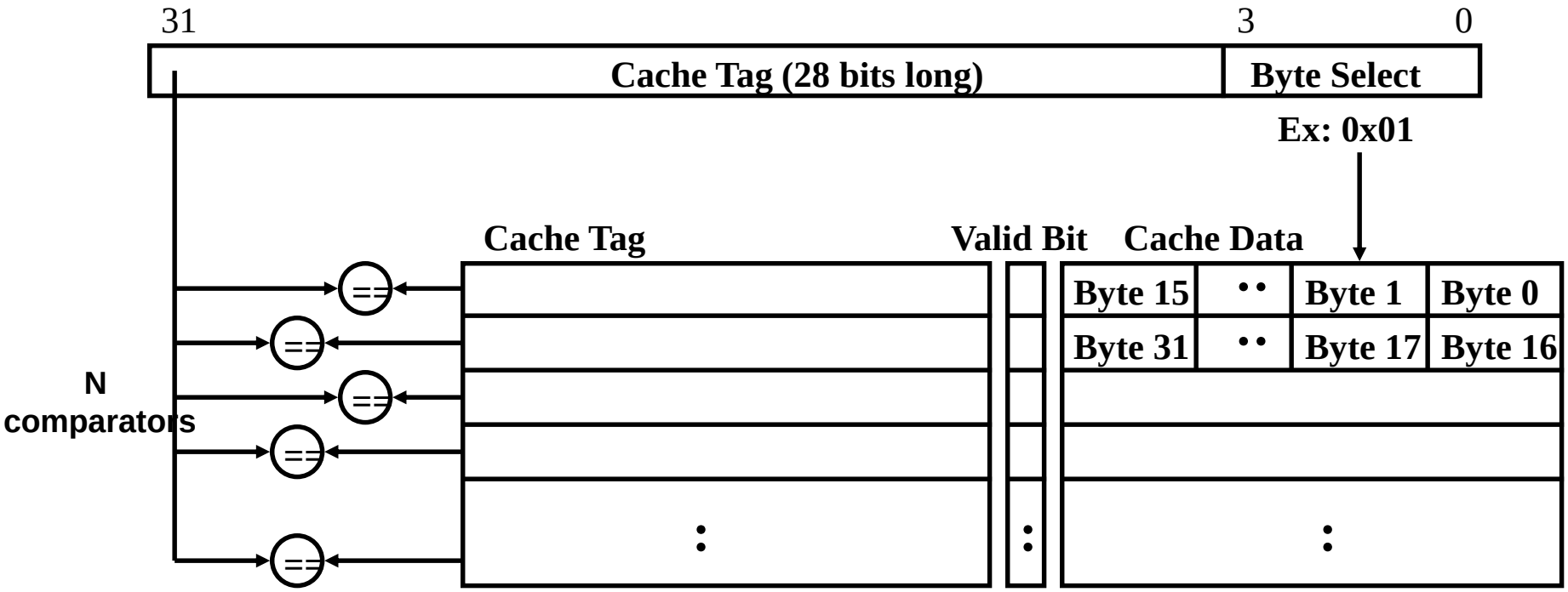
Avantages et Inconvénients

- **Avantages**
  - Pas d'algorithme de remplacement
  - Matériel simple et peu couteux
  - Rapide
- Inconvénients
  - Ratio de succès faible
  - Performances décroissent si même index

Tendance *: Direct mapping adapté aux mémoires caches plus grandes*

Another Extreme Example: Fully Associative (FA)

- Forget about the Cache Index
- Compare the Cache Tags of all cache entries in parallel
- We need N = 1024 (28-bit) comparators



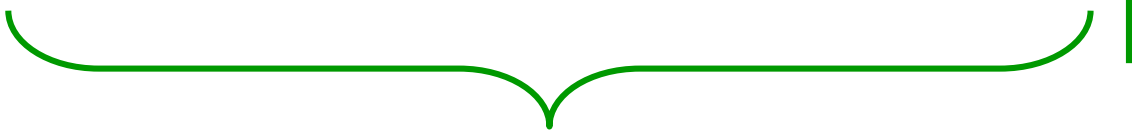
Reprendre les mêmes séquences avec FA

So lets go through accessing some data in a FA, 16KB cache

- 16 byte blocks x 1024 cache blocks

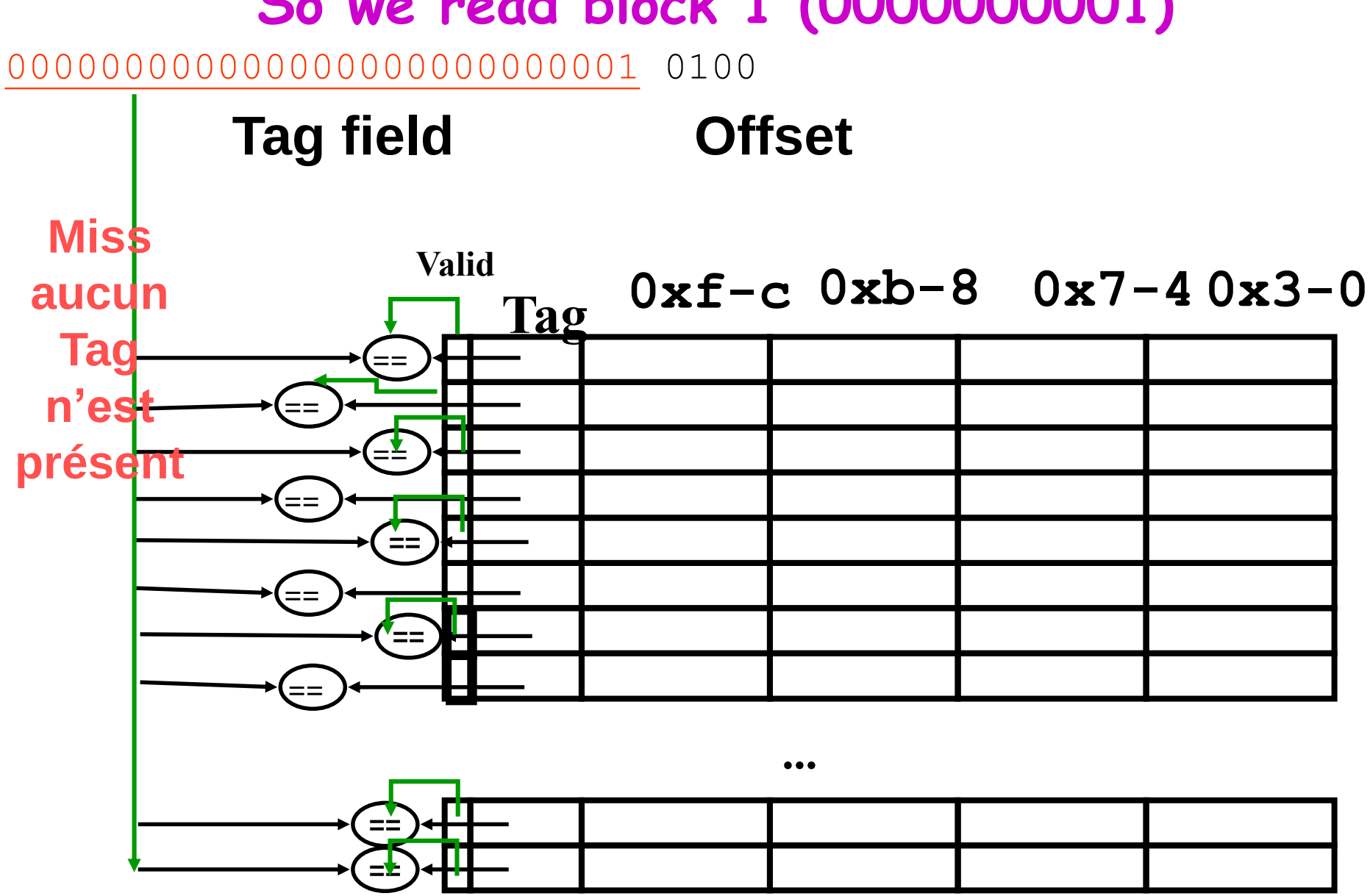
2 Addresses divided (for convenience) into Tag, Byte Offset fields

```
0000000000000000000 0000000001 0100
0000000000000000000 0000000011 0100
0000000000000000010 0000000001 0100
```



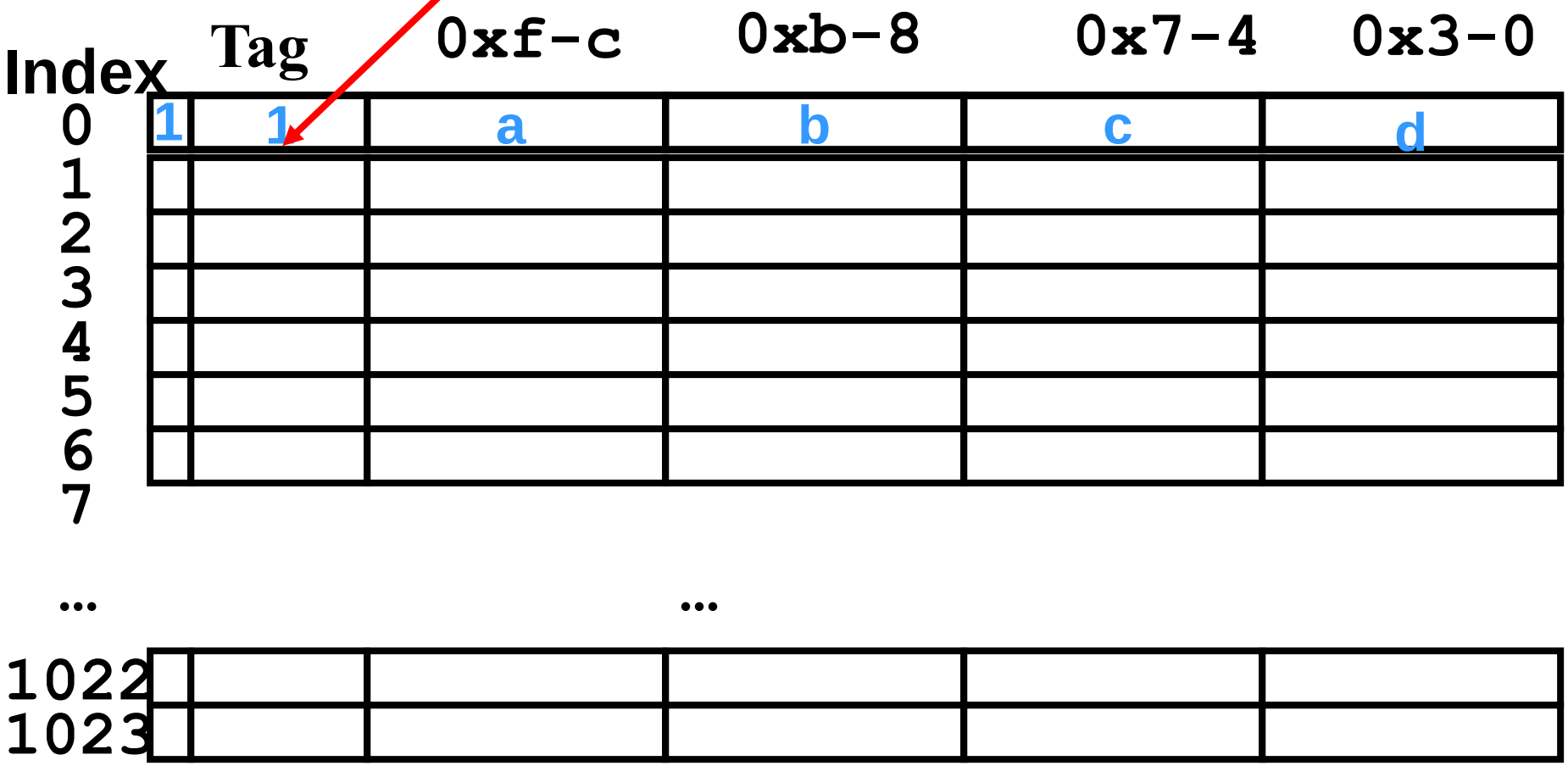
Tag Offset

So we read block 1 (0000000001)



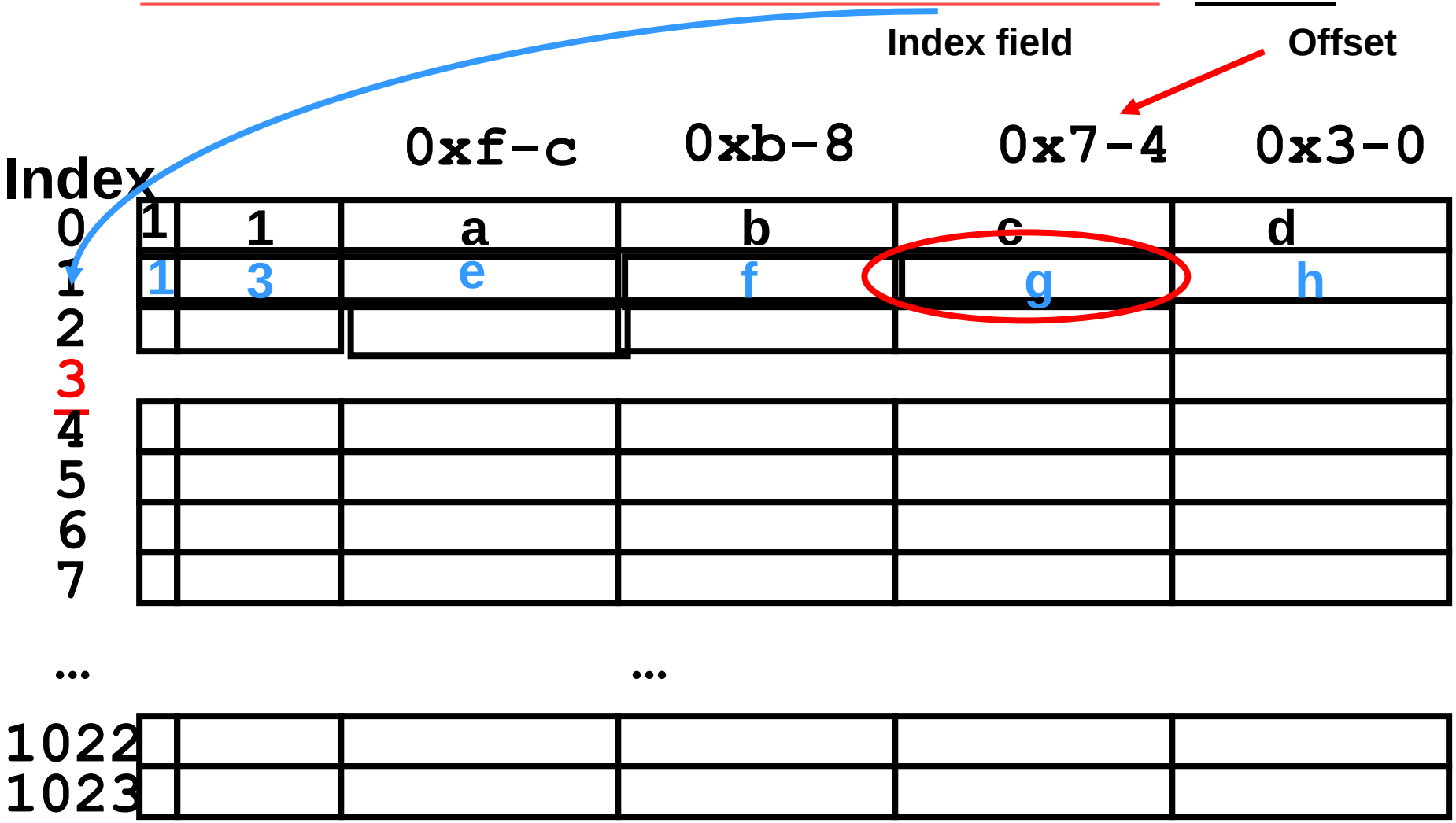
So load that data into cache, setting tag, valid

00000000000000000000 0000000001 0100



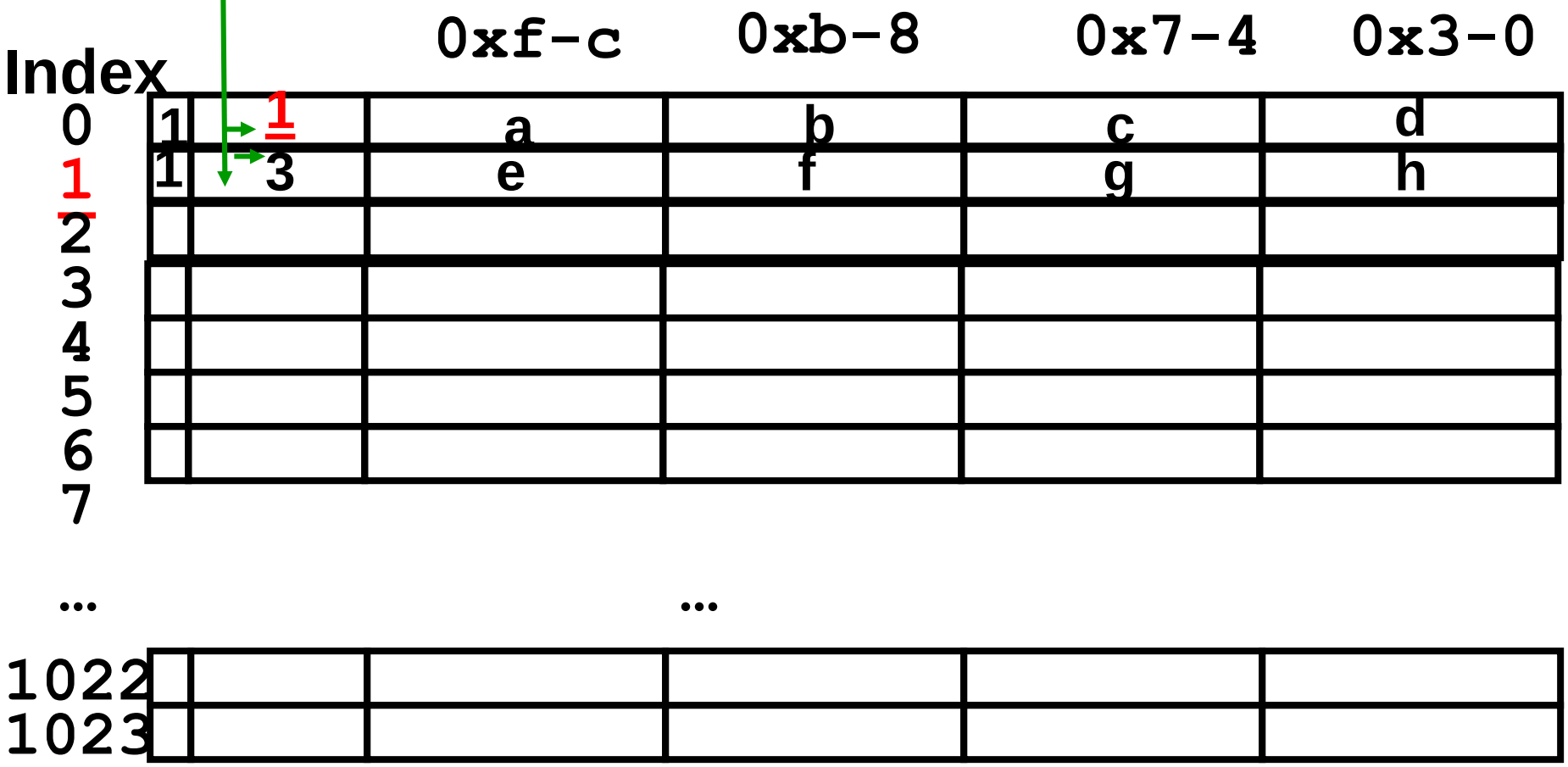
Load cache block 3, return word 4

0000000000000000000 0000000011 0100



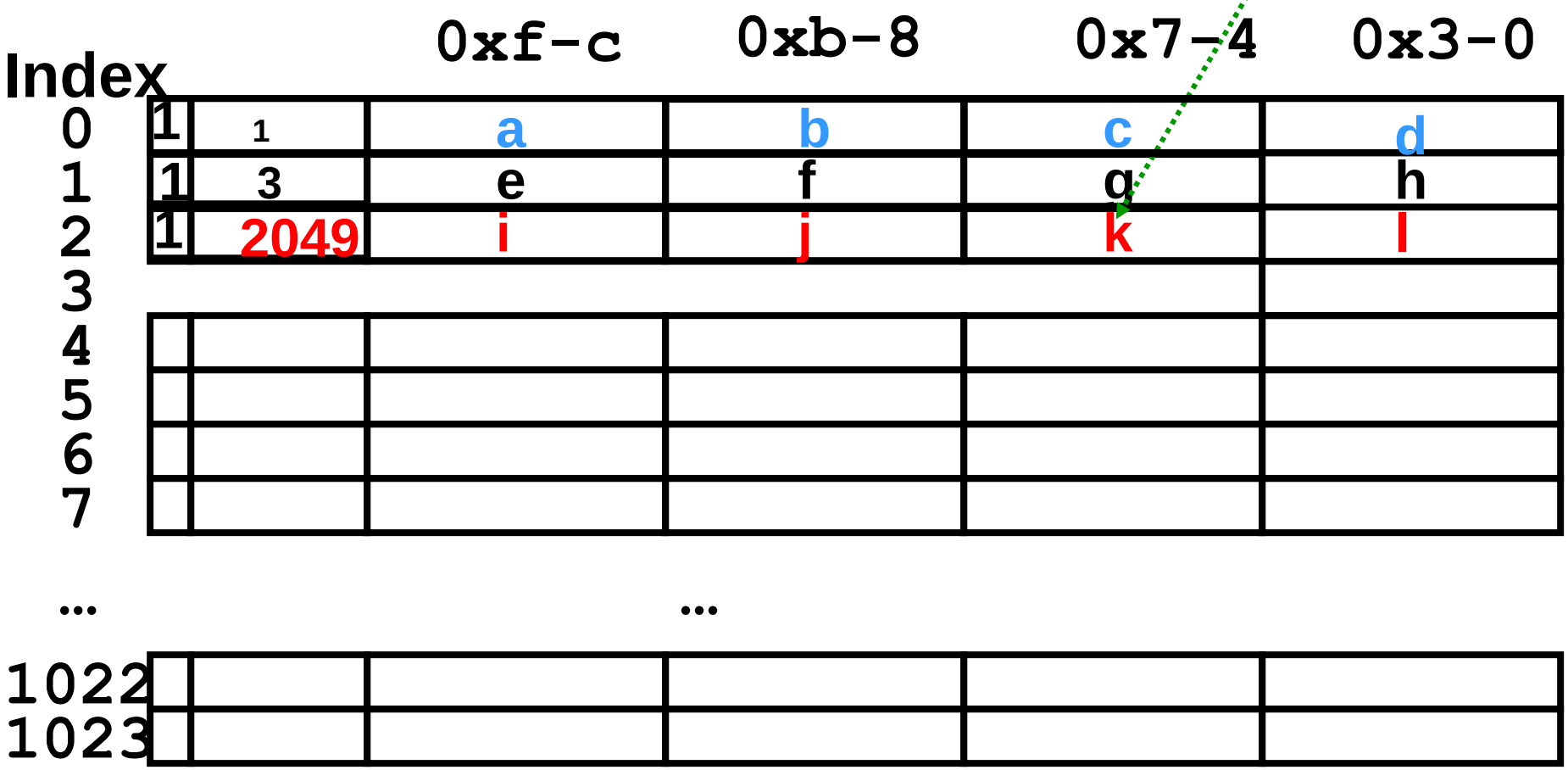
Cache Block 1 Tag does not match

0000000000000000010 0000000001 0100



Miss, no need to replace block 1
new data & tag in 2 entry

00000000000000000010 0000000001 0100



Avantages et Inconvénients

- **Avantages**
  - Flexibilité
  - Plusieurs lignes dans le cache avec le même numéro
- **Inconvénients**
  - Coût hardware du comparateur
  - Algorithme de remplacement (lequel enlever ?)
  - Taille réduite

Compromise:
N-way Set Associative Cache

- Compromis entre direct et full associative mapping
- Permet un nombre limité de blocs (lignes) avec le même index mais des tag différents
- Le cache est divisé en « sets » de blocs. Chaque set contient le même nombre de blocs
- Chaque bloc a son propre Tag, qui associé à l'index, identifie le bloc

Compromise: N-way Set Associative Cache

N-way set associative:

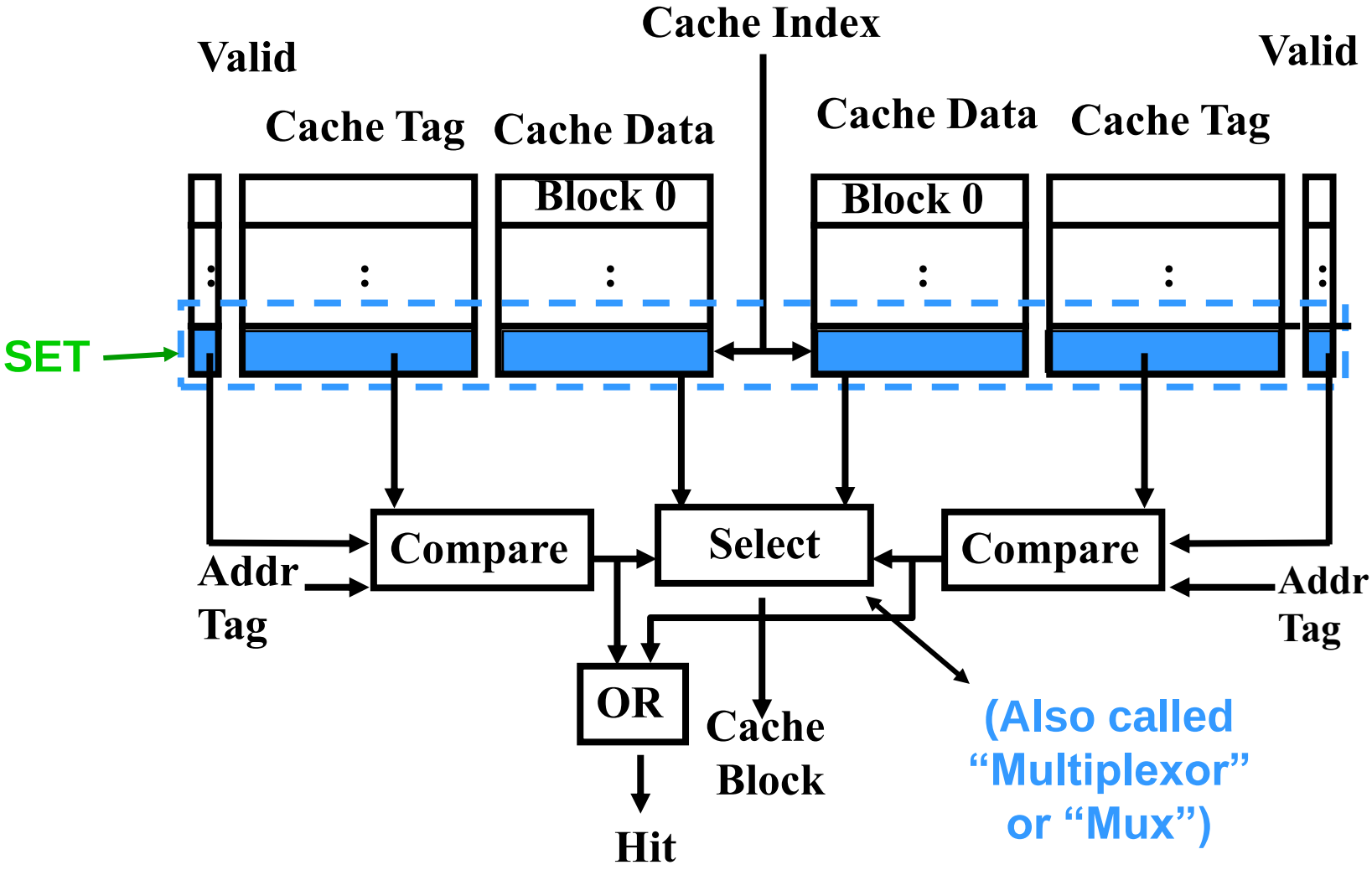
N entries for each Cache Index

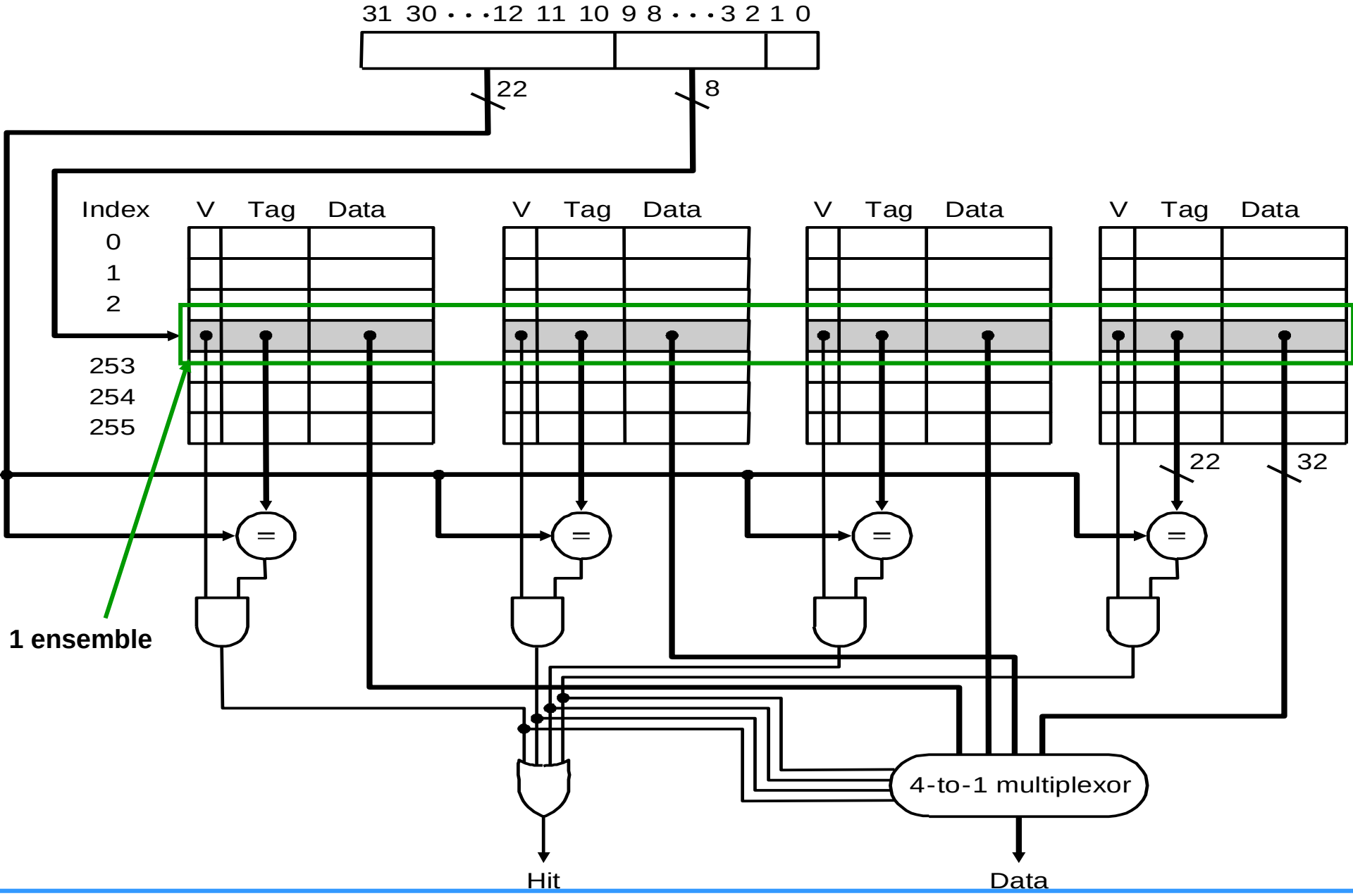
- N direct mapped caches operate in parallel
- Select the one that gets a hit

Example: 2-way set associative cache

- Cache Index selects a "set" from the cache
- The 2 tags in set are compared in parallel
- Data is selected based on the tag result (which matched the address)

Compromise: 2-way Set Associative Cache

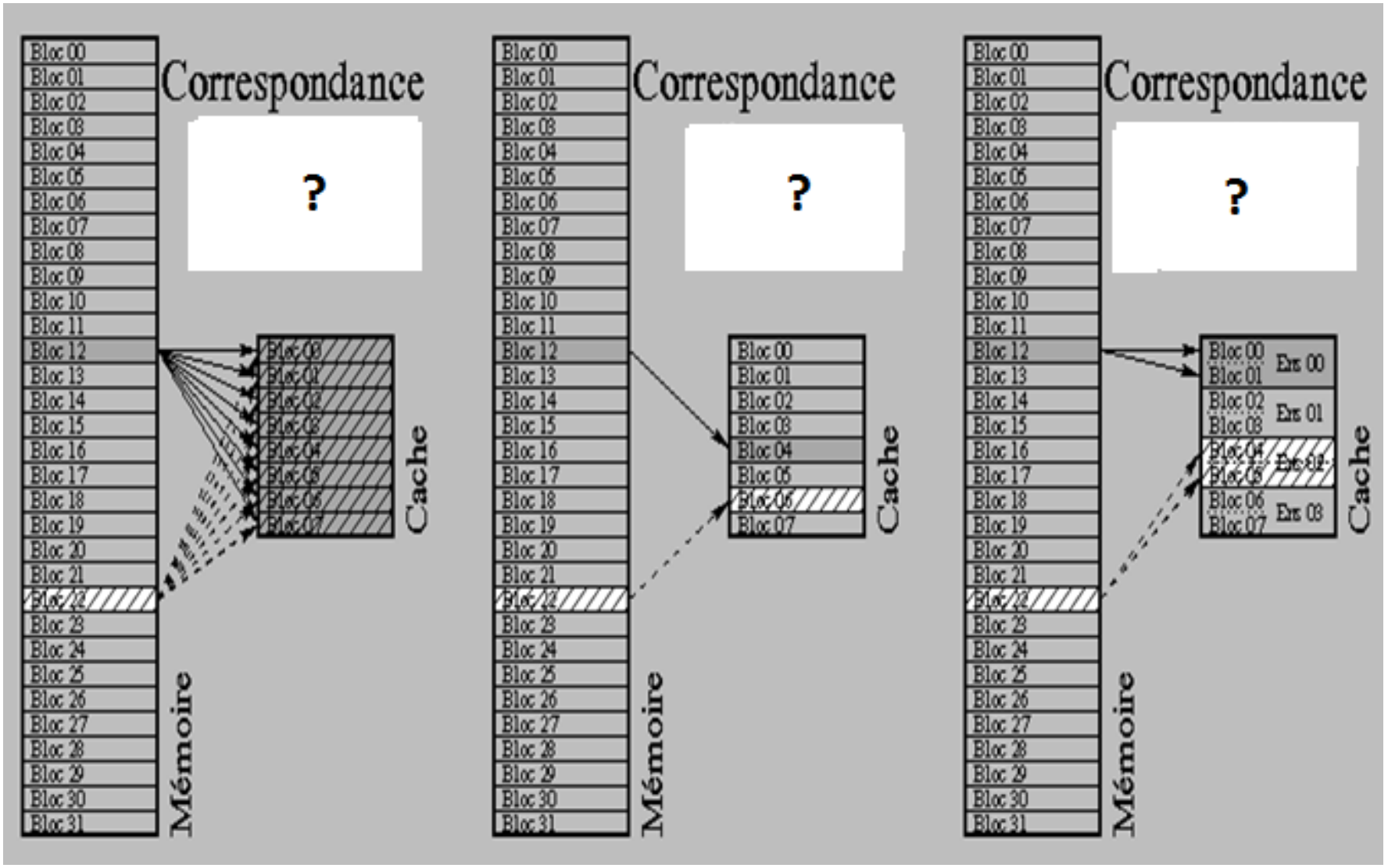



31 30 · · · 12 11 10 9 8 · · · 3 2 1 0
22
8
Index
V
Tag
Data
0
1
2
253
254
255
22
32
=
1 ensemble
4-to-1 multiplexor
Hit
Data

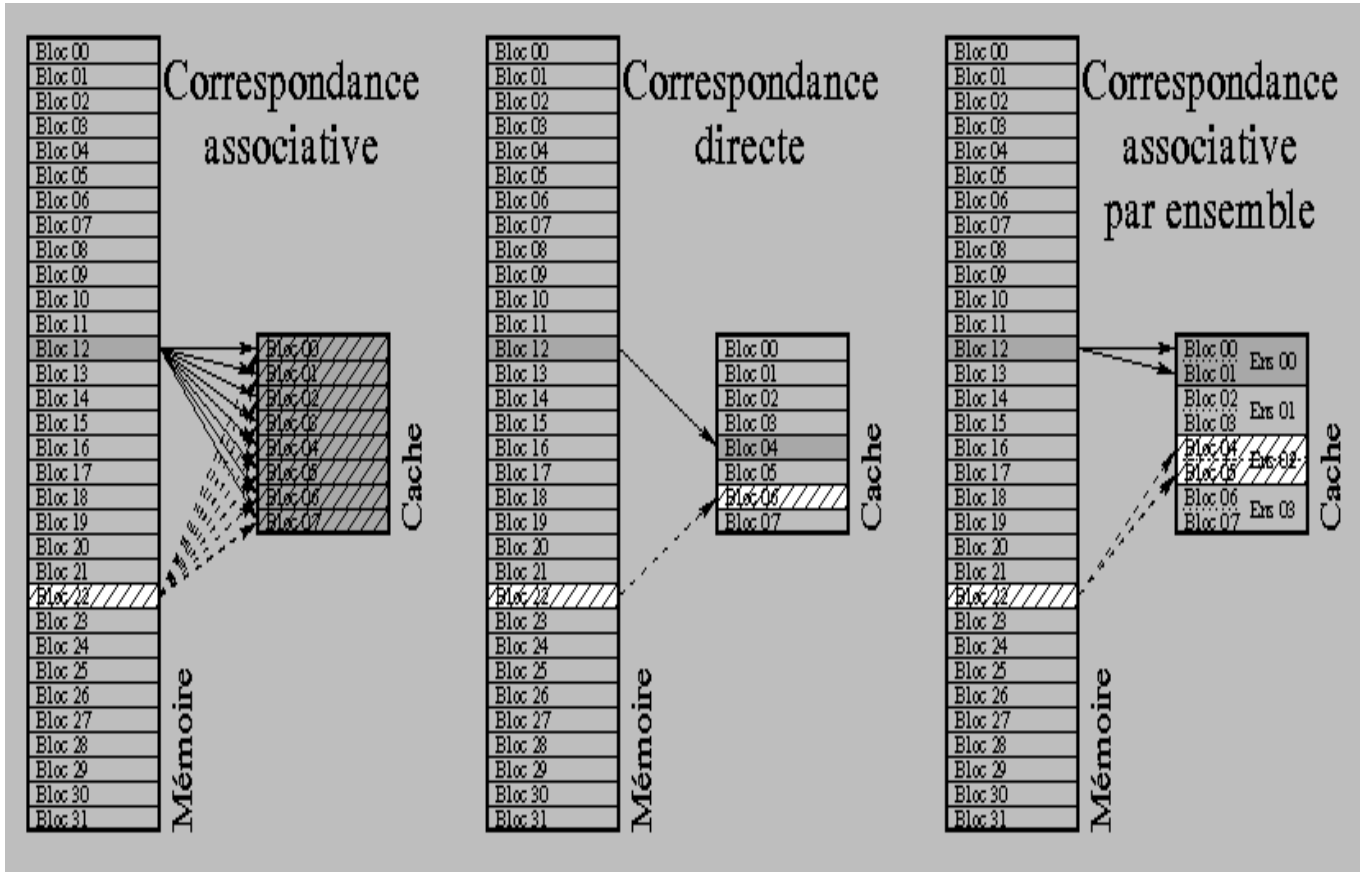
Avantages

- **Avantages**
  - Moins de comparateurs
  - Algorithmes de remplacement seulement sur le set
  - Plus courant avec deux blocs par set

Résumé


Bloc 00
Bloc 01
Bloc 02
Bloc 03
Bloc 04
Bloc 05
Bloc 06
Bloc 07
Bloc 08
Bloc 09
Bloc 10
Bloc 11
Bloc 12
Bloc 13
Bloc 14
Bloc 15
Bloc 16
Bloc 17
Bloc 18
Bloc 19
Bloc 20
Bloc 21
Bloc 22
Bloc 23
Bloc 24
Bloc 25
Bloc 26
Bloc 27
Bloc 28
Bloc 29
Bloc 30
Bloc 31
Mémoire
Correspondance
?
Cache
Ens 00
Ens 01
Ens 02
Ens 03

Résumé


Correspondance associative
Correspondance directe
Correspondance associative par ensemble
Mémoire
Cache
Bloc 00
Bloc 01
Bloc 02
Bloc 03
Bloc 04
Bloc 05
Bloc 06
Bloc 07
Bloc 08
Bloc 09
Bloc 10
Bloc 11
Bloc 12
Bloc 13
Bloc 14
Bloc 15
Bloc 16
Bloc 17
Bloc 18
Bloc 19
Bloc 20
Bloc 21
Bloc 22
Bloc 23
Bloc 24
Bloc 25
Bloc 26
Bloc 27
Bloc 28
Bloc 29
Bloc 30
Bloc 31
Ens 00
Ens 01
Ens 02
Ens 03

Q3 : Block Replacement Policy

- Cache à correspondance directe
  - 1 seule place possible dans le cache
  - pas de décision à prendre
- Associatif par ensemble de N blocs
  - Choix entre N positions possibles dans le cache
- Totalement associatif
  - Chaque case mémoire peut être placée n'importe où dans le cache

Replacement Algorithms (1)
Direct mapping

- No choice
- Each block only maps to one line
- Replace that line

Replacement Algorithms (2) Associative & Set Associative

- N-way Set Associative or Fully Associative have choice where to place a block, (which block to replace)
  - Of course, if there is an invalid block, use it
- When need to evict a cache block, choose one
  - Hardware implemented algorithm (speed)
  - Least Recently used (LRU)
  - e.g. in 2 way set associative
    - Which of the 2 block is lru ?
  - First in first out (FIFO)
    - replace block that has been in cache longest
  - Least frequently used
    - replace block which has had fewest hits
  - Random

Q4: What happens on a write?

- Consistance entre cache et mémoire
  (les deux versions de la donnée peuvent être différentes)

- Nécessité de conserver la cohérence dans les systèmes multiprocesseurs (mémoire partagée et caches multiples) ou avec des entrées sorties sur la mémoire

Q4: What happens on a write ?

- Must not overwrite a cache block unless main memory is up to date
- Multiple CPUs may have individual caches
- I/O may address main memory directly

- Write through — The information is written to both the block in the cache and to the block in the lower-level memory.
- Write back — The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.

Write through

- All writes go to main memory as well as cache
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
- Lots of traffic
- Slows down writes
- Remember bogus write through caches!

Write back

- Updates initially made in cache only
- Update bit for cache slot is set when update occurs
- If block is to be replaced, write to main memory only if update bit is set
- Other caches get out of sync
- I/O must access main memory through cache
- N.B. 15% of memory references are writes

Echecs écritures

- ***Write allocate*** —The block is allocated on a write miss, followed by the write hit actions above.
  In this natural option, write misses act like **read misses**.
- ***No-write allocate***—This apparently unusual alternative is write misses do *not* affect the cache. Instead, the block is modified only in the lower level memory.

- Thus, blocks stays out of the cache in no-write allocate until the program tries to read the blocks, but even blocks that are only written will still be in the cache with write allocate.

Echecs écritures

❑ Exemple :

❑ Assume a fully associative write back cache with many cache entries that starts empty. Below is a sequence of five memory operations (the address is in parentheses):

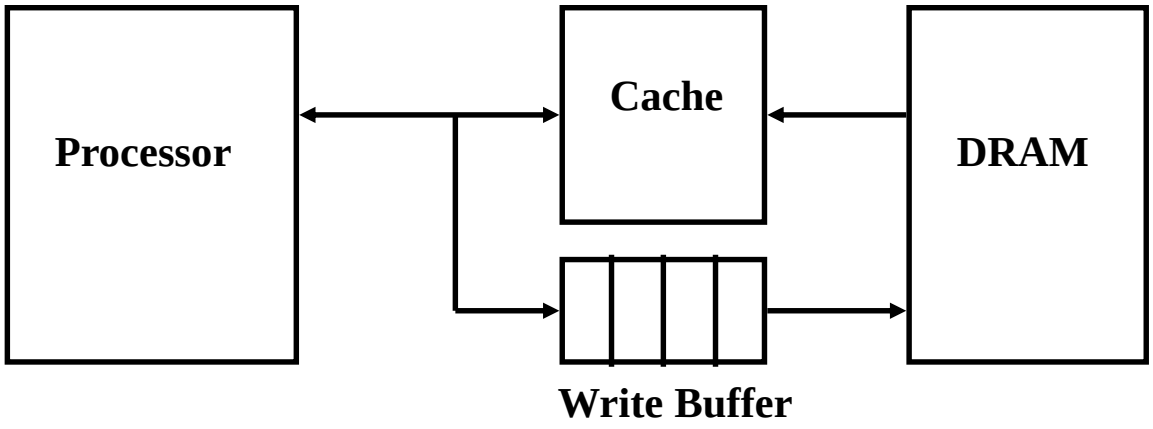
Write Mem[100];
WriteMem[100];
Read Mem[200];
WriteMem[200];
WriteMem[100].

What are the number of hits and misses with using no-write allocate versus write allocate?

❑ For no-write allocate, the address 100 is not in the cache, and there is no allocation on write, so the first two writes will result in misses. Address 200 is also not in the cache, so the read is also a miss. The subsequent write to address 200 is a hit. The last write to 100 is still a miss. The result for no write allocate is 4 misses and 1 hit.

❑ For write allocate, the first accesses to 100 and 200 are misses, and the rest are hits since 100 and 200 are both found in the cache. Thus, the result for write allocate is 2 misses and 3 hits.

A Write Buffer as a solution to speed up writes in WT

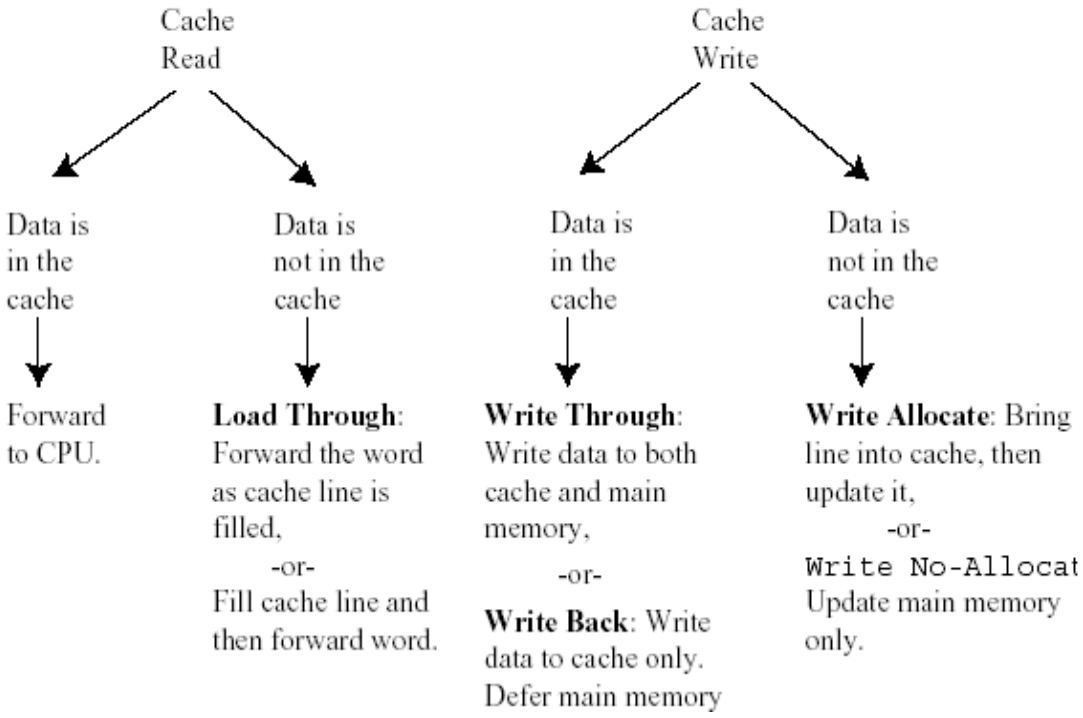


- Deux étapes :
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory

°Write buffer is just a FIFO

Statatégies

Cache Read and Write Policies



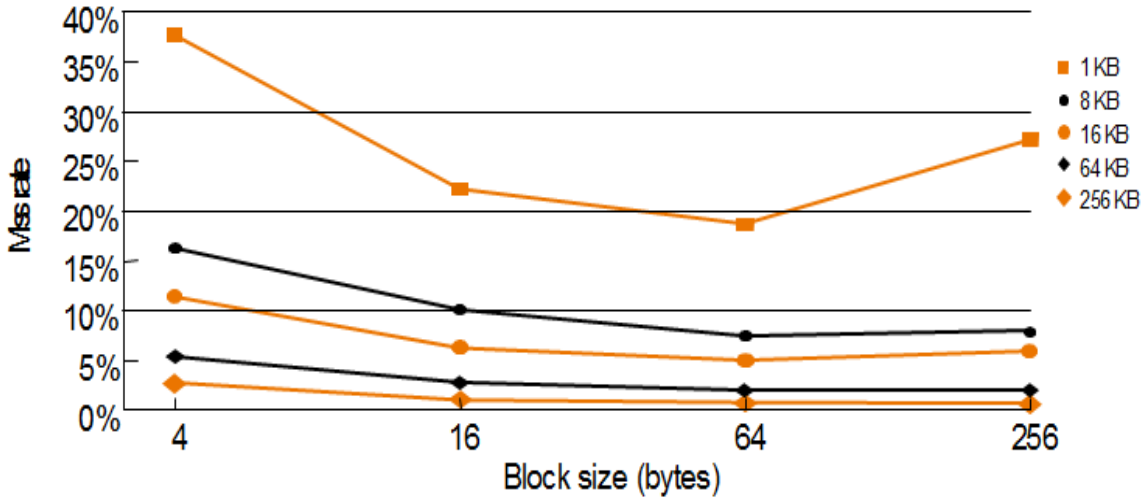
Line Size

- Retrieve not only desired word but a number of adjacent words as well
- Increased block size will increase hit ratio at first
  - the principle of locality
- Hit ratio will decreases as block becomes even bigger
  - Probability of using newly fetched information becomes less than probability of reusing replaced
- Larger blocks
  - Reduce number of blocks that fit in cache
  - Data overwritten shortly after being fetched
  - Each additional word is less local so less likely to be needed
- No definitive optimum value has been found
- 8 to 64 bytes seems reasonable
- For HPC systems, 64- and 128-byte most common

Optimal block size : Example

Le tableau ci-dessous montre des valeurs des taux d'échec tracés dans la figure . On suppose que le système mémoire a un temps de démarrage de 40 cycles et délivre ensuite 16 octets tous les 2 cycles d'horloge. Ainsi, il peut fournir 16 octets en 42 cycles, 32 octets en 44 cycles, etc. Quelle taille de bloc a le temps d'accès mémoire moyen minimum pour chaque taille de cache de ce tableau ?

Taille du bloc	Taille du cache				
	1k	4k	16k	64k	256k
16	15,05%	8,57%	3,94%	2,04%	1,09%
32	13,34%	7,24%	2,87%	1,35%	0,70%
64	13,76%	7,00%	2,64%	1,06%	0,51%
128	16,64%	7,78%	2,77%	1,02%	0,49%
256	22,01%	9,51%	3,29%	1,15%	0,49%



Optimal block size : Example

- Le temps d'accès mémoire moyen est :

  Temps moyen d'accès mémoire = Temps succès + Taux d'échec x Pénalité d'échec

- Considérons les cas extrèmes pour un temps d'accès réussi de 1 cycle
  - **1er Cas** : taille cache =1 ko , taille bloc = 16 o, taux échecs = 15,05%

    le temps d'accès mémoire moyen pour un bloc de 16 octets dans un cache de 1Ko est :

    Temps moyen d'accès mémoire = 1+ (15,05% x 42) = 7,321 cycles
  - **Dernier cas** : taille cache = 256 ko, taille bloc = 256 O, taux d'échecs = 0,49%

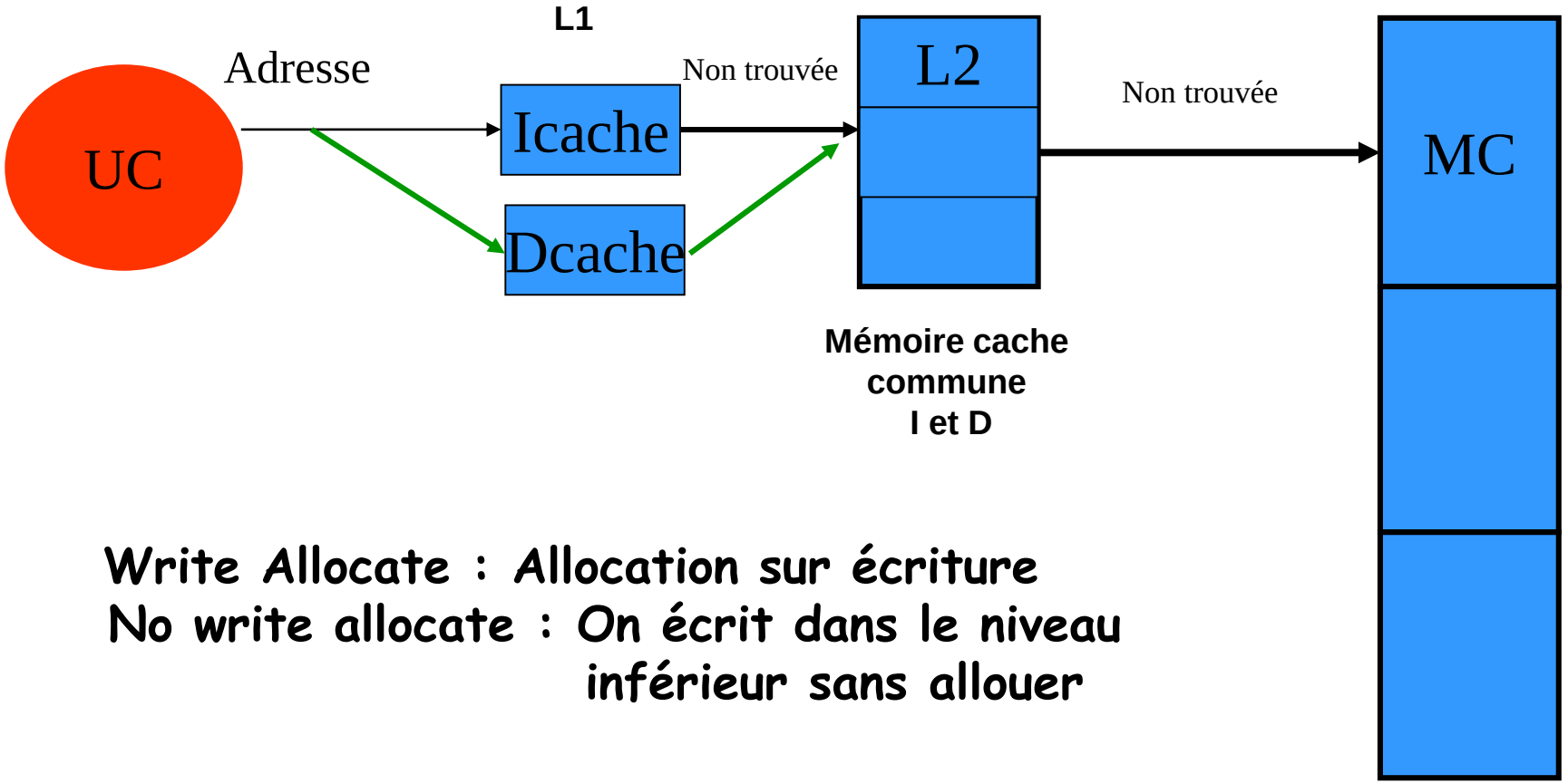
    Temps moyen d'accès mémoire = 1 + 0,49% x 72 = 1,3528 cycles

    ≈ 1,353 cycles

Multilevel Caches

- High logic density enables caches on chip
  - Faster than bus access
  - Frees bus for other transfers
- Common to use both on and off chip cache
  - L1 on chip, L2 off chip in static RAM
  - L2 access much faster than DRAM or ROM
  - L2 often uses separate data path
  - L2 may now be on chip
  - Resulting in L3 cache
  - Bus access or now on chip…

Mémoire cache à plusieurs niveaux



Write Allocate : Allocation sur écriture
No write allocate : On écrit dans le niveau inférieur sans allouer

Unified vs Split Caches

- One cache for data and instructions or two, one for data and one for instructions
- Advantages of unified cache
  - Higher hit rate
    - Balances load of instruction and data fetch
    - Only one cache to design & implement
- Advantages of split cache
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
    - Important in pipelining

Cache structure in modern microprocessors

Processor	Icache	Dcache	L2cache	Block size Bytes
AMD Athlon	64KB - 2 way Set Assoc	64 KB - 2 way set asso	256 KB - 16 way set assoc	64
Pentium III	16 KB – 2 way set assoc	16 KB – 2way set assoc	256 KB – 8 set assoc	32
Pentium4	12 K μOperations 120 bits	8 KB – 4 way set Assoc	256 KB – 8 set Assoc	128

Classification of cache misses

The three types:

Compulsory (or cold) miss : The first reference to a block is always a miss

Capacity miss : If the space is not sufficient to host the data or code that have been accessed.

Conflict miss: Two memory blocks may be mapped to the same cache block with a direct or set-associative address mapping even if there is still unused space in cache

Amélioration de performances

Temps moyen d'accès mémoire =

Temps succès +

Taux d'échec x

x Pénalité d'échec

Amélioration de performances

- Action sur les facteurs de l'équation de performance :
  - Réduire le taux d'échec
  - Réduire la pénalité d'échec
  - Réduire le temps d'accès réussi

Amélioration de performances

- **Réduire le taux d'échec**
  - Taille de bloc plus élevée
  - Associativité plus élevée
  - Cache victime
  - Caches pseudo-associatif
  - Lecture anticipée par matériel
  - Lecture anticipée contrôlée par compilateur
  - Optimisation du compilateur

Amélioration de performances

- **Réduire la pénalité d'échec**
  - Donner priorité aux échecs lectures sur les écritures
  - Le placement par sous bloc
  - Le redémarrage précoce et le mot critique en premier
  - Les caches non bloquants pour réduire les suspensions sur les défauts cache
  - Les caches de second niveau

Amélioration de performances

- **Réduire le temps d'accès réussi**
  - Des caches petits et simples
  - Eviter la traduction d'adresse durant l'indexation du cache
  - Pipeliner les écritures pour des écritures réussies rapides

Questions

Q1. Comparer les 3 organisations en faisant des évaluations de performances à l'aide d'un simulateur de cache par exemple

Q2. Expliquer une méthode d'amélioration des performances associée à chacun des facteurs significatifs

Q3. Que suggérez-vous pour remédier au problème de cohérence de cache dans un système multiprocesseur (mémoire partagée et caches multiples) ou avec des entrées/sorties sur mémoire


Thank's