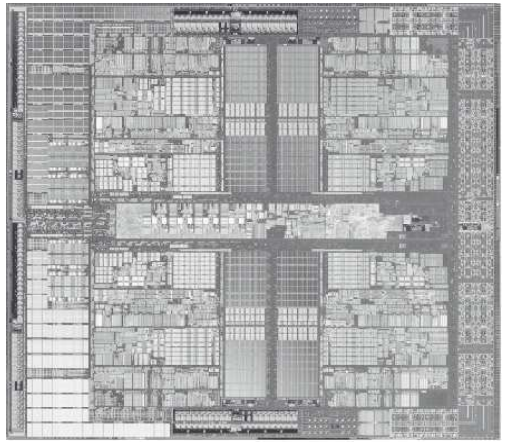
Università degli Studi di Cassino e del Lazio Meridionale



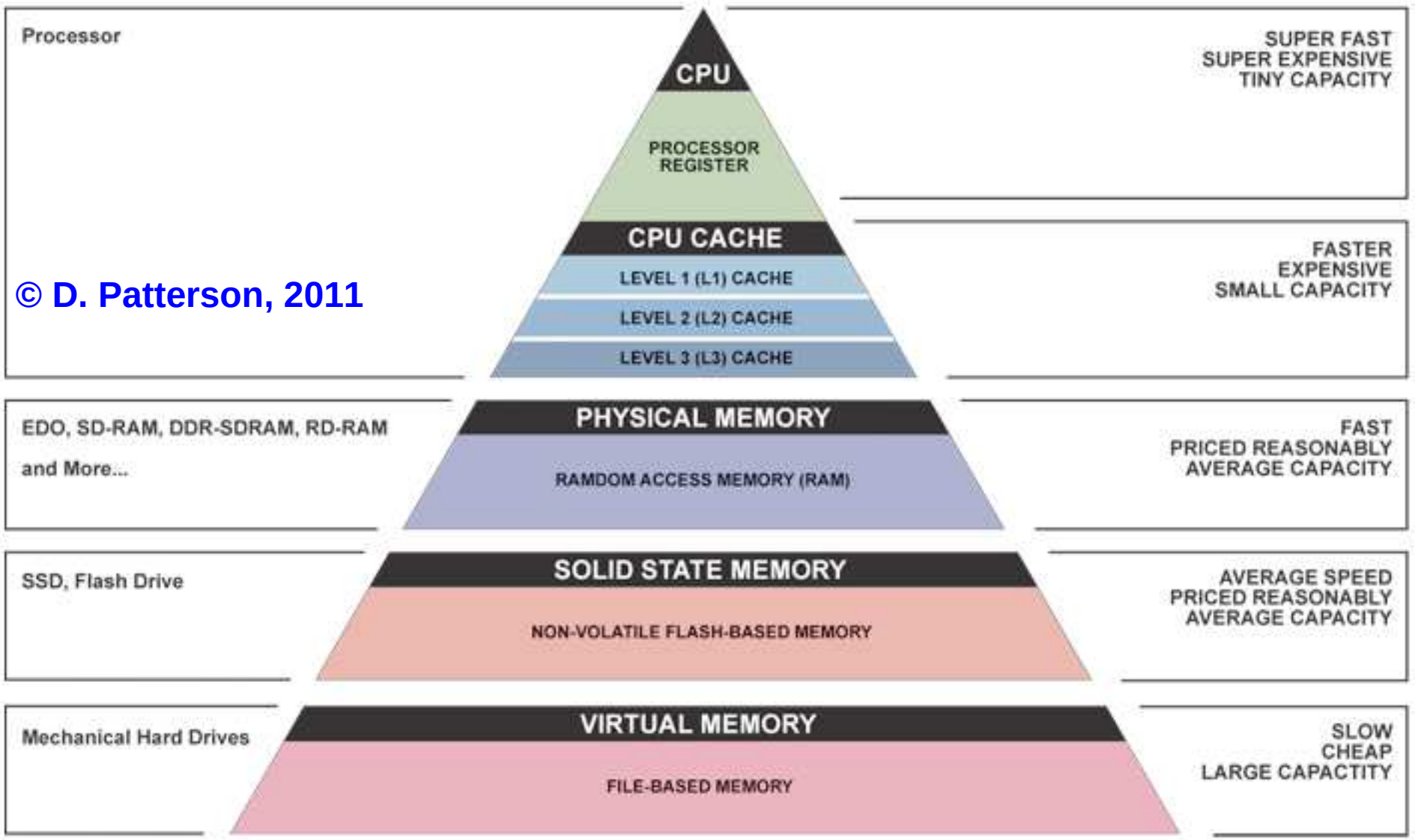
Corso di Calcolatori Elettronici

Cache

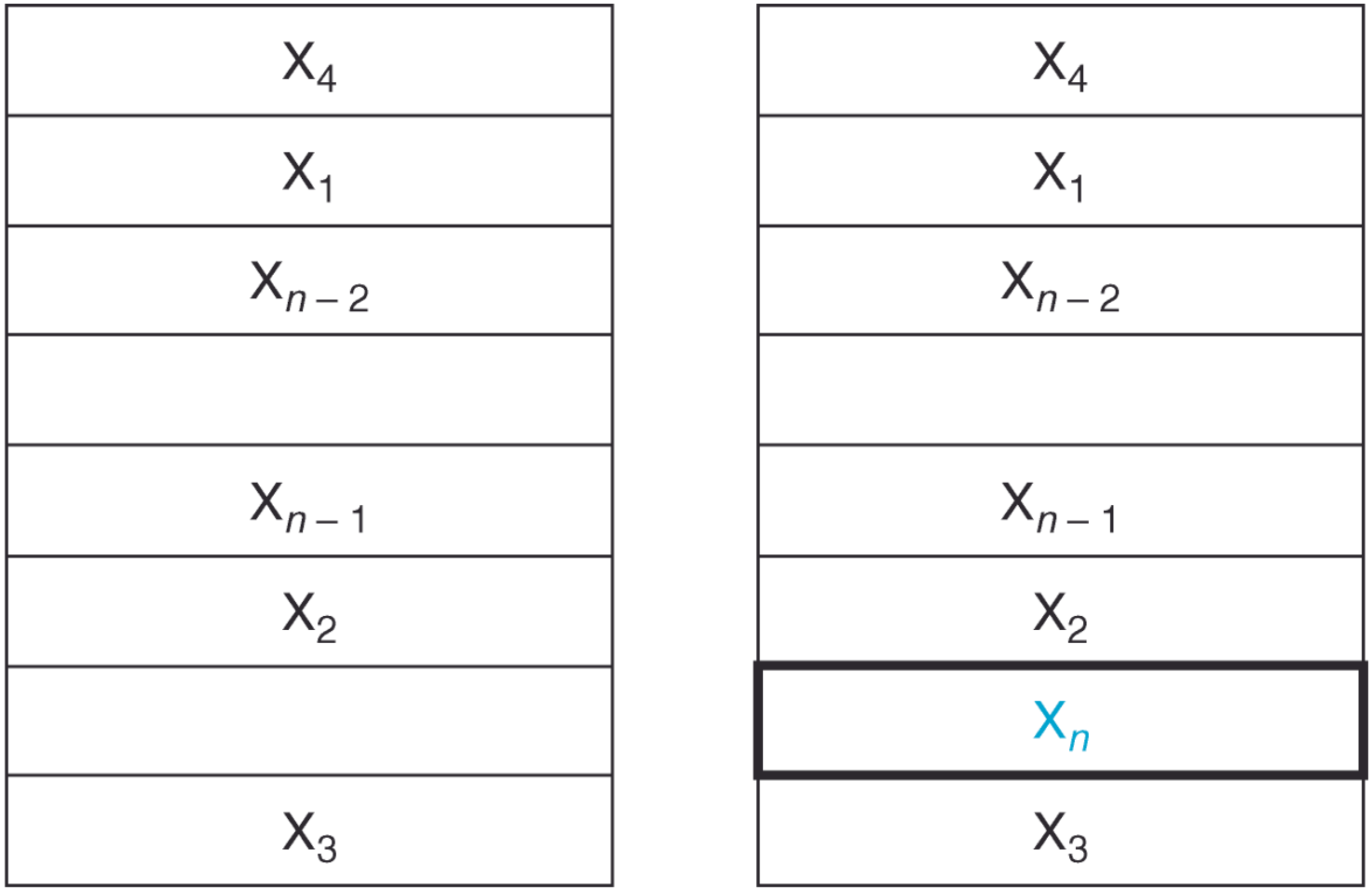
Anno Accademico 2011/2012

Francesco Tortorella

La cache nella gerarchia di memoria



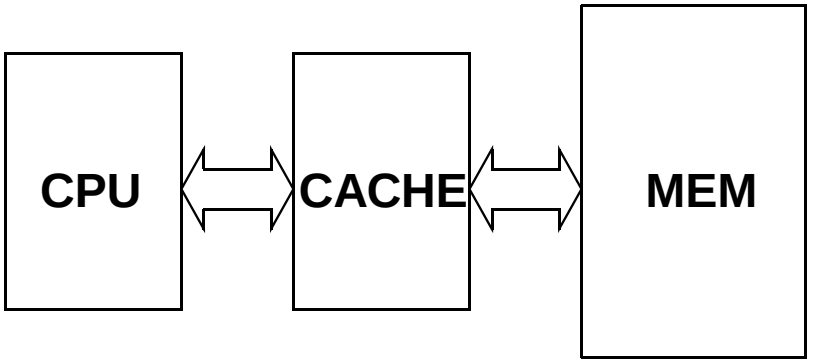
Funzionamento della cache



a. Prima di richiedere il dato X_n

b. Dopo avere richiesto il dato X_n

4 decisioni da prendere



1. Dove posizionare un blocco ?
2. Come reperire un blocco ?

} **Tecnica di indirizzamento**

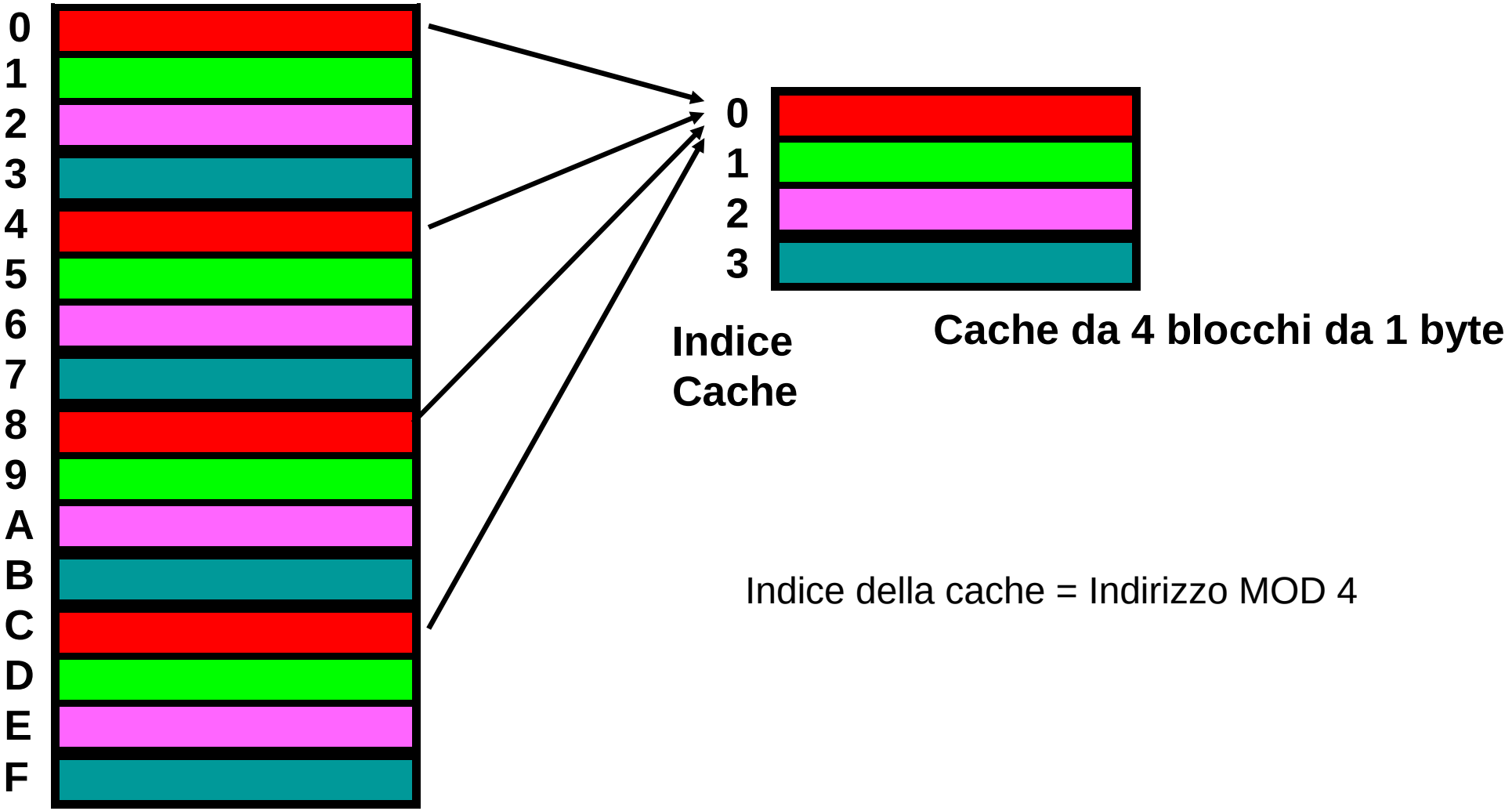
3. Quale blocco sostituire in corrispondenza di un miss ?

Algoritmo di sostituzione

4. Come gestire un'operazione di scrittura ?

Strategia di aggiornamento

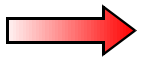
Cache ad accesso diretto (Direct Mapped Cache)



Indice della cache = Indirizzo MOD 4

Problemi dell'accesso diretto

- Quale blocco di memoria è presente nella cache ?
- Come gestire un blocco formato da più di 1 byte ?



Si divide l'indirizzo di memoria in 3 campi:
tag, index, e byte offset

index riporta l'indice del blocco all'interno della cache;

offset riporta lo spiazzamento dei dati di interesse all'interno del blocco

tag riferisce se il dato al blocco indirizzato nella cache corrisponde all'indirizzo desiderato

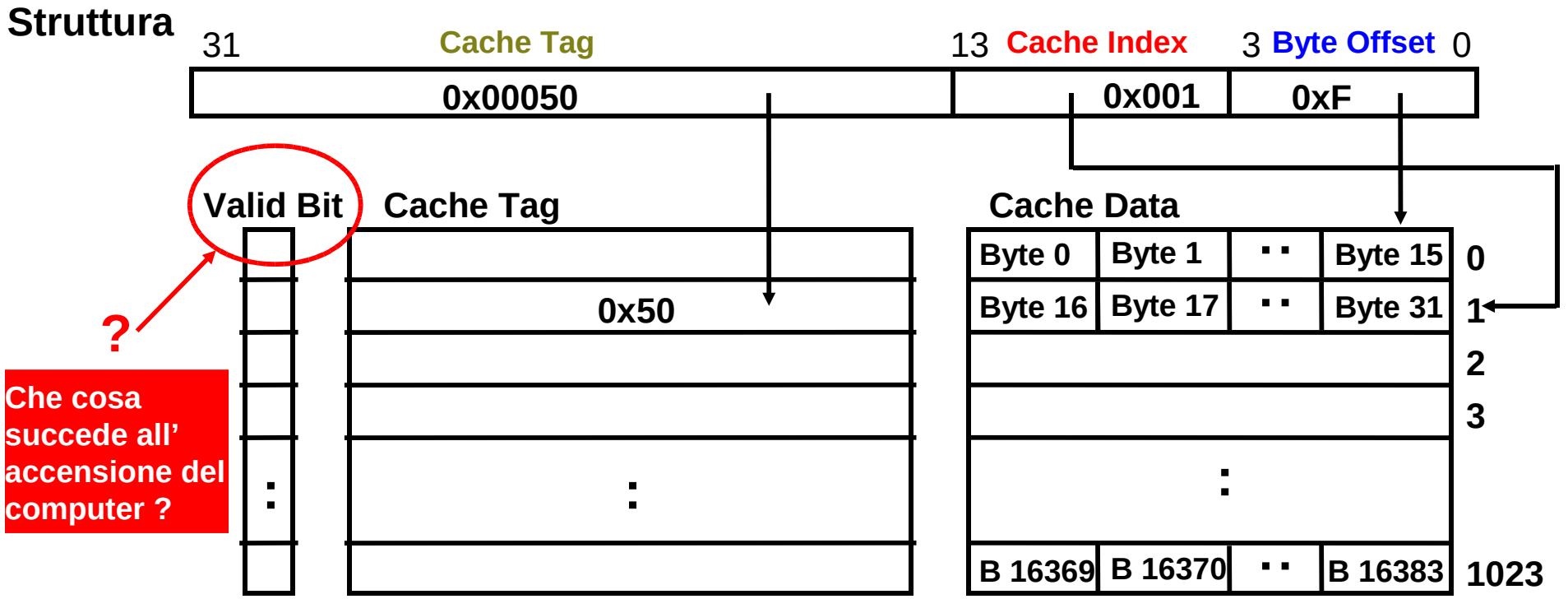
Esempio: cache a.d. da 16 KB (blocchi da 16 B)

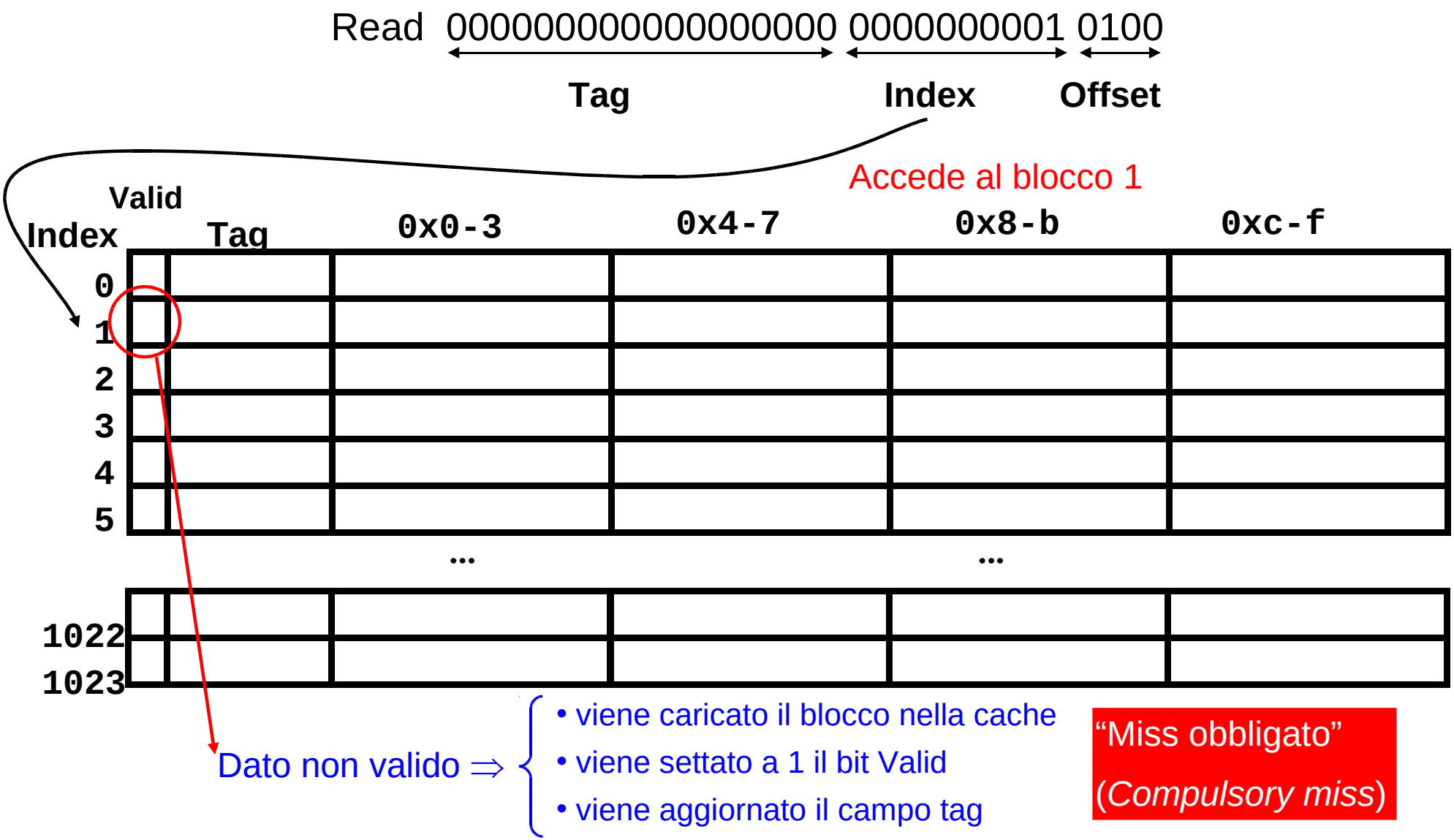
- dimensione dell'indirizzo di memoria: 32 bit
- dimensione della cache: 2^N bytes (N=14)
- dimensione del blocco (linea): 2^M bytes (M=4)
  - gli N bit meno significativi identificano il byte nella cache
  - i 32-N bit più significativi costituiscono il tag

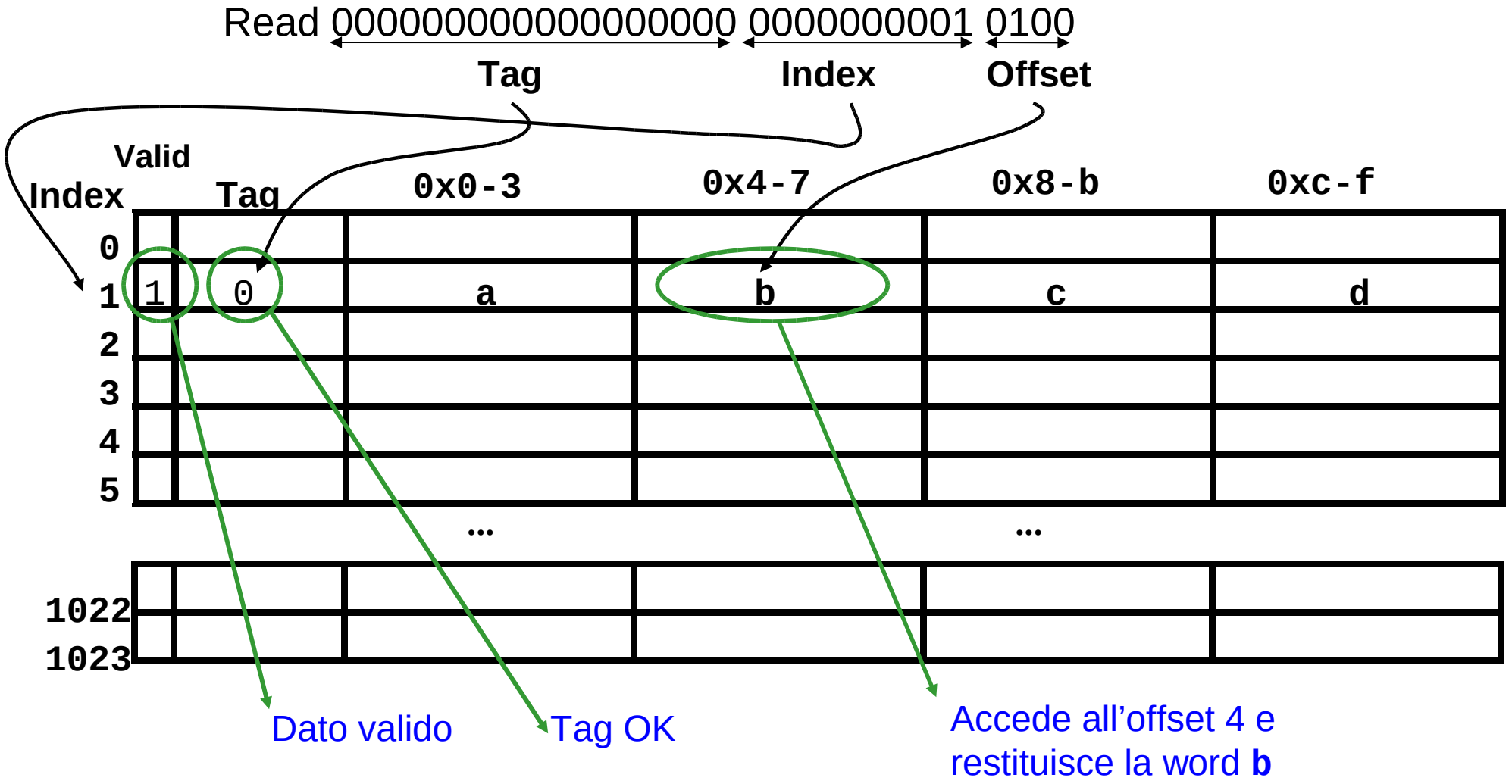
M per l'offset

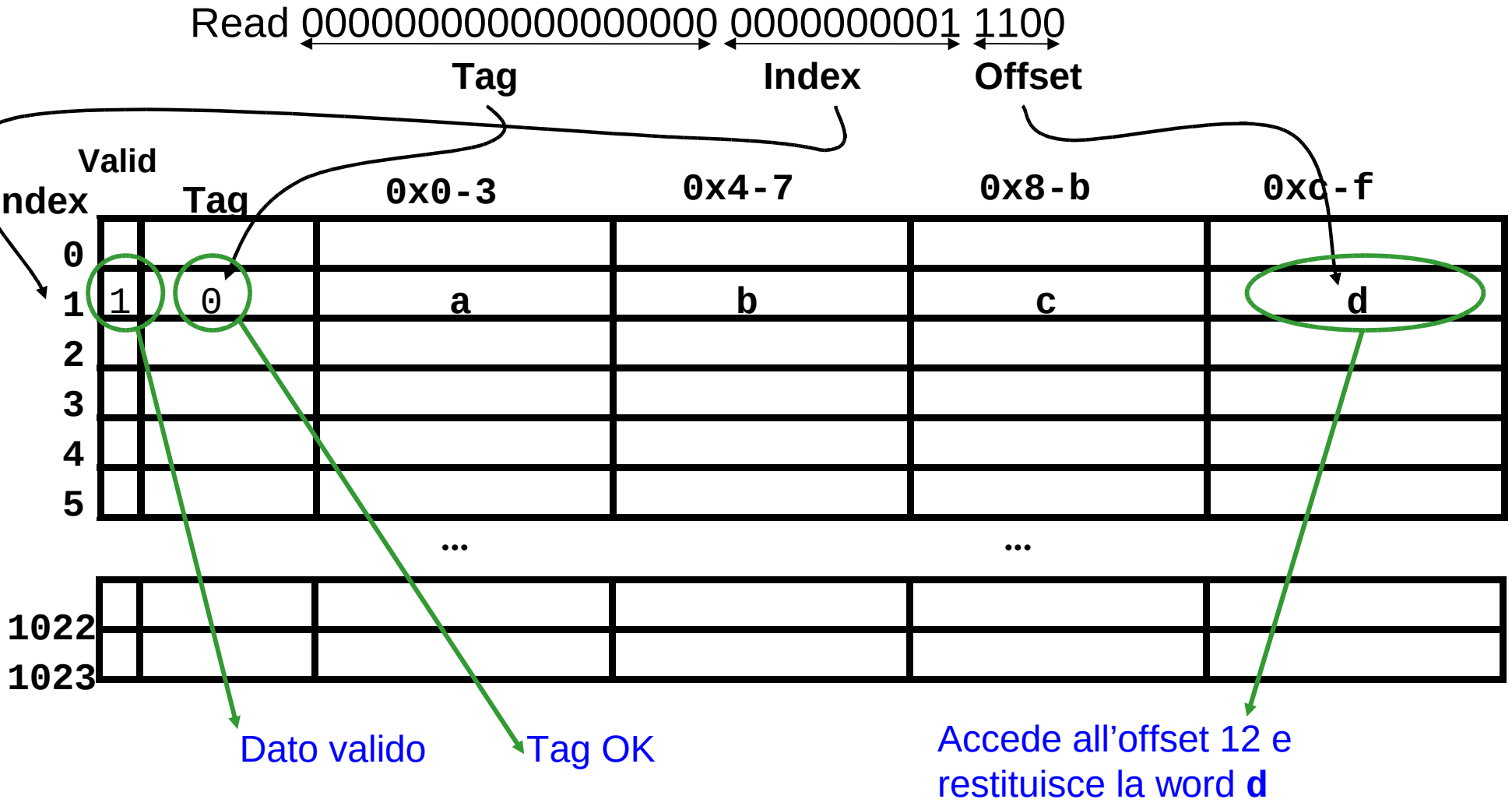
N-M per l'index

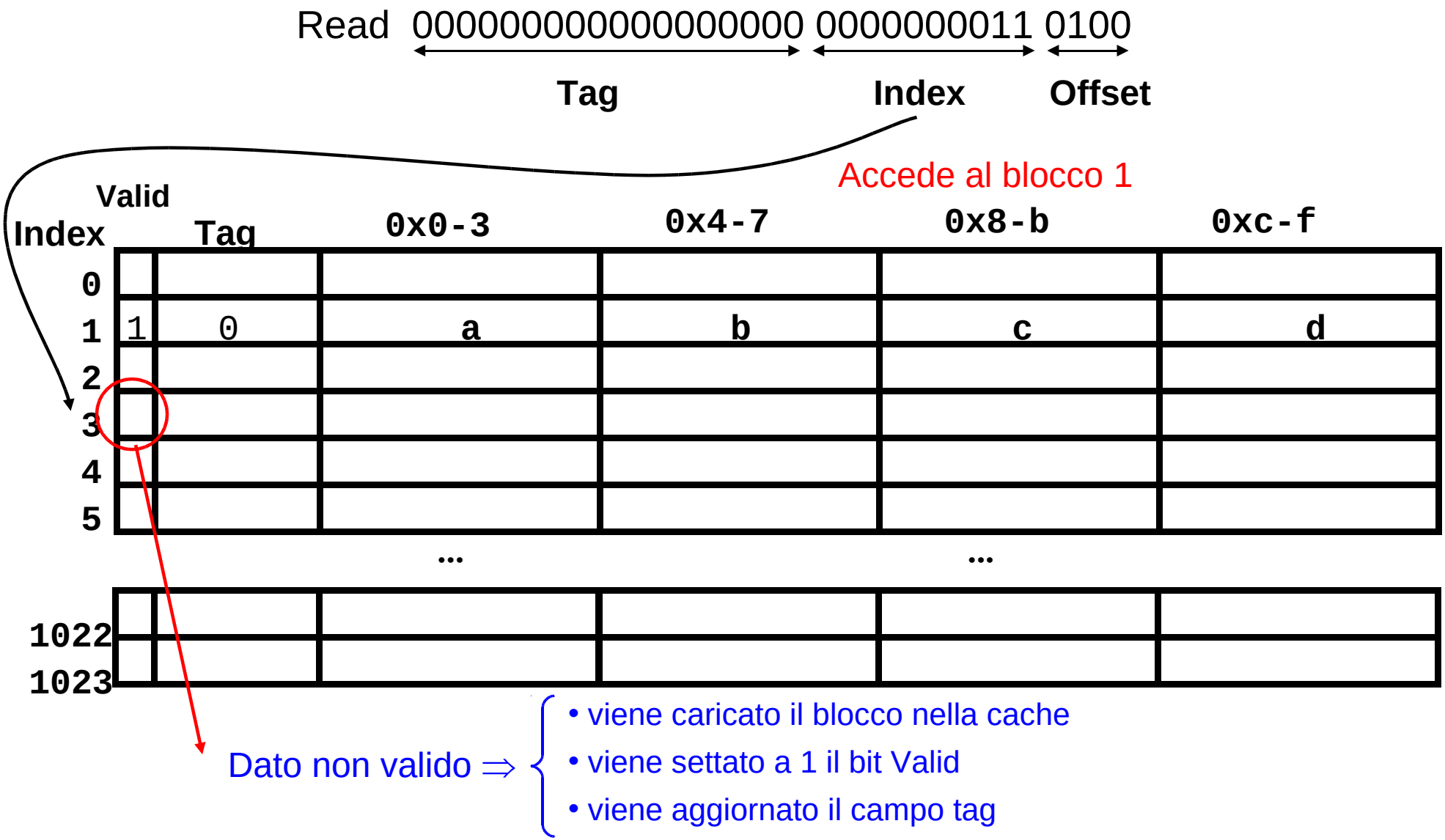
Struttura

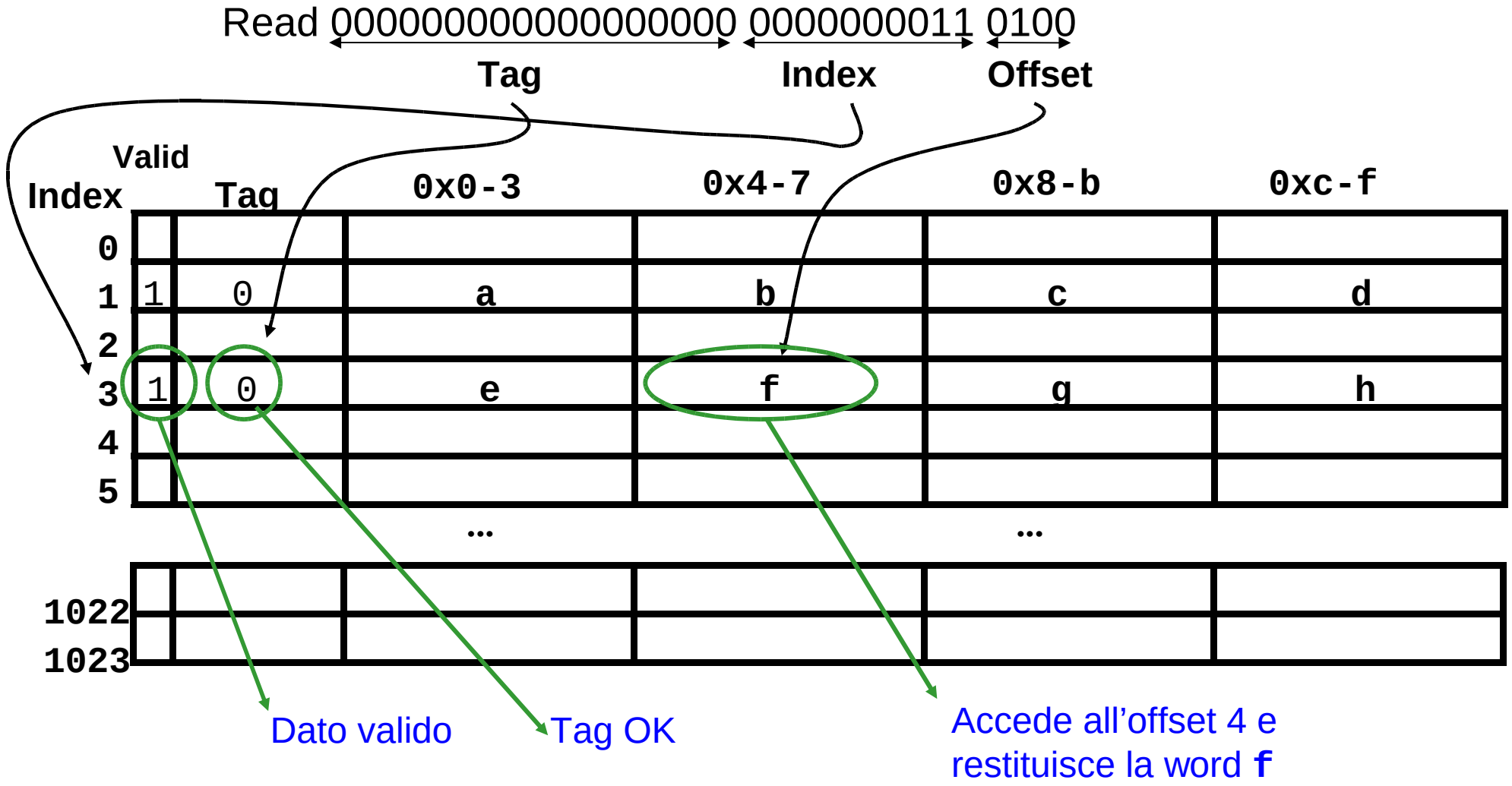


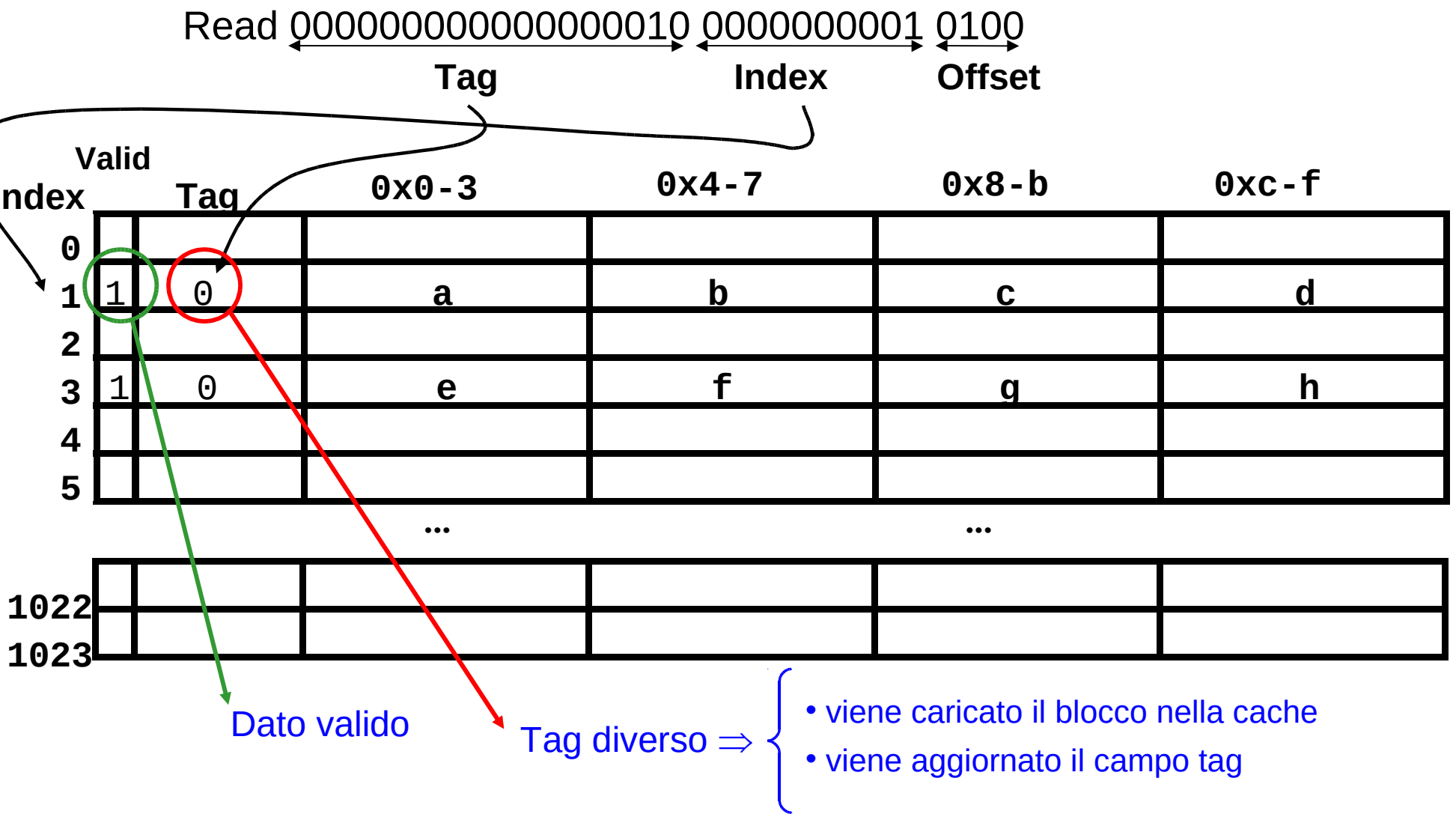

Read 0000000000000000 0000000001 0100
Tag
Index
Offset
Accede al blocco 1
Valid
Index
Tag
0x0-3
0x4-7
0x8-b
0xc-f
0
1
2
3
4
5
...
...
1022
1023
Dato non valido ⇒
• viene caricato il blocco nella cache
• viene settato a 1 il bit Valid
• viene aggiornato il campo tag
"Miss obbligato"
(Compulsory miss)

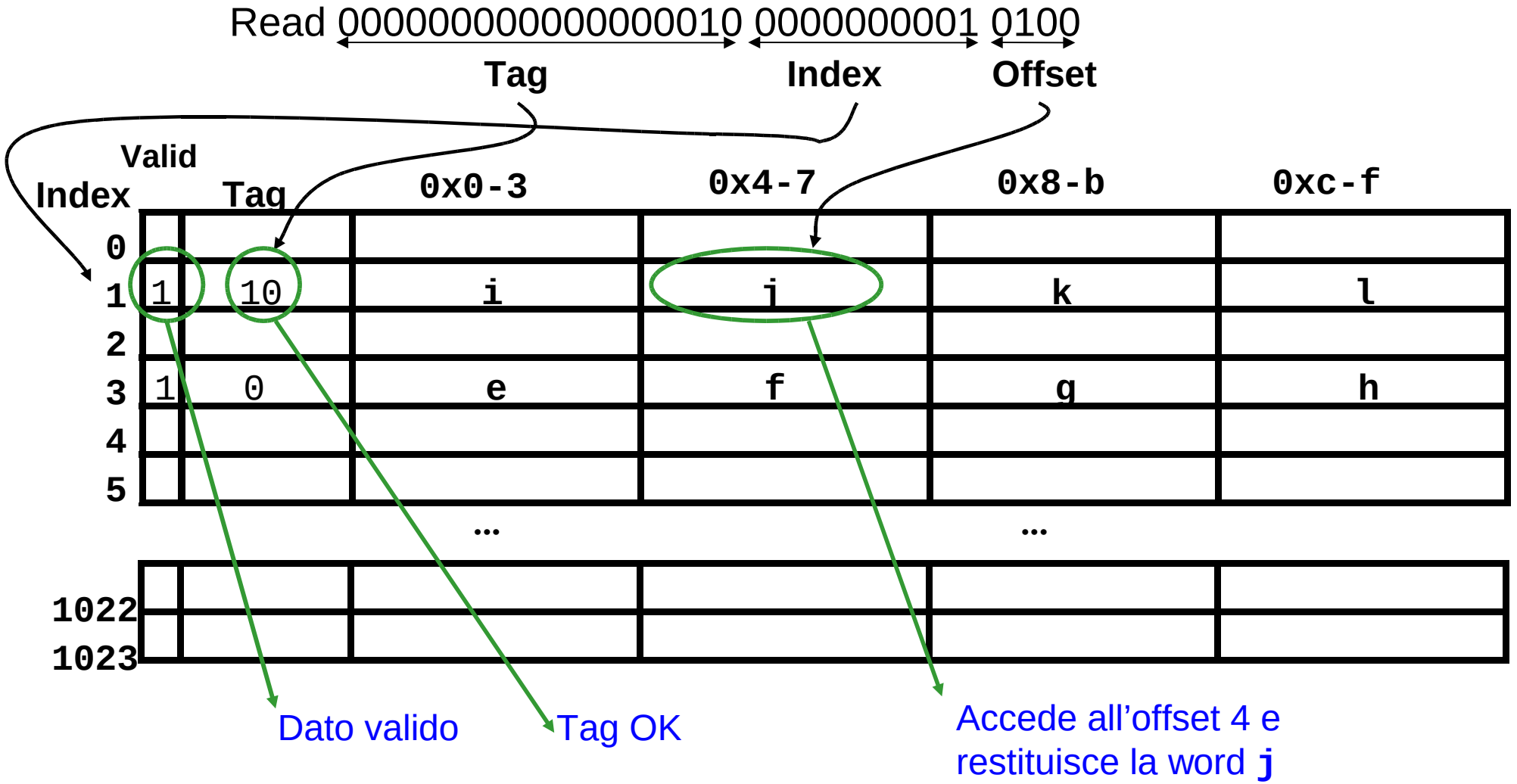

Read 00000000000000000 0000000001 0100
Tag
Index
Offset
Valid
Index
Tag
0x0-3
0x4-7
0x8-b
0xc-f
0
1
2
3
4
5
1
0
a
b
c
d
...
...
1022
1023
Dato valido
Tag OK
Accede all'offset 4 e
restituisce la word b


Read 00000000000000000 0000000001 1100
Tag
Index
Offset
Valid
Index
Tag
0x0-3
0x4-7
0x8-b
0xc-f
0
1
2
3
4
5
1
0
a
b
c
d
...
...
1022
1023
Dato valido
Tag OK
Accede all'offset 12 e
restituisce la word d


Read 0000000000000000 0000000011 0100
Tag
Index
Offset
Accede al blocco 1
Valid
Index
Tag
0x0-3
0x4-7
0x8-b
0xc-f
0
1
2
3
4
5
1
0
a
b
c
d
...
...
1022
1023
Dato non valido ⇒
• viene caricato il blocco nella cache
• viene settato a 1 il bit Valid
• viene aggiornato il campo tag


Read 0000000000000000 0000000011 0100
Tag
Index
Offset
Valid
Index
Tag
0x0-3
0x4-7
0x8-b
0xc-f
0
1
2
3
4
5
1022
1023
1 0 a b c d
1 0 e f g h
...
...
Dato valido
Tag OK
Accede all'offset 4 e
restituisce la word f

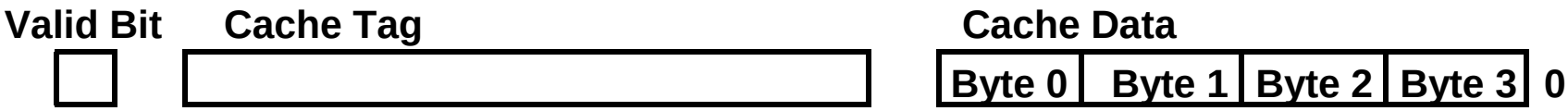

Read 0000000000000010 0000000001 0100
Tag
Index
Offset
Valid
Index
Tag
0x0-3
0x4-7
0x8-b
0xc-f
0
1
1
0
a
b
c
d
2
3
1
0
e
f
g
h
4
5
...
...
1022
1023
Dato valido
Tag diverso ⇒
• viene caricato il blocco nella cache
• viene aggiornato il campo tag


Read 0000000000000010 0000000001 0100
Tag
Index
Offset
Valid
Index
Tag
0x0-3
0x4-7
0x8-b
0xc-f
0
1
2
3
4
5
1022
1023
1
10
i
j
k
l
1
0
e
f
g
h
...
...
Dato valido
Tag OK
Accede all'offset 4 e
restituisce la word j

Scelta della dimensione del blocco

- Fissata la quantità di memoria a disposizione della cache, bisogna decidere l'ampiezza del singolo blocco
- In linea di principio, una dimensione ampia del blocco permette di sfruttare al meglio la località spaziale
- Potrebbero esserci delle controindicazioni ?
    - vediamo un esempio limite ...

Cache a singolo blocco

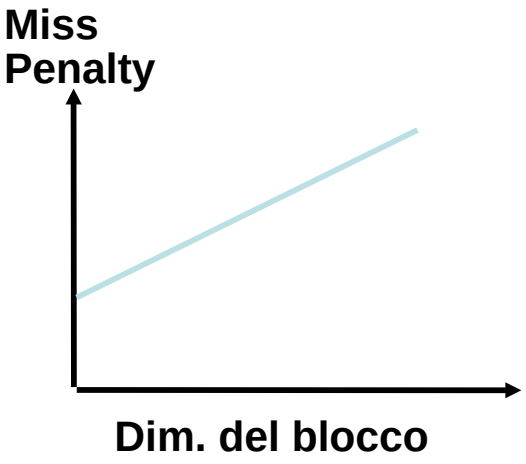


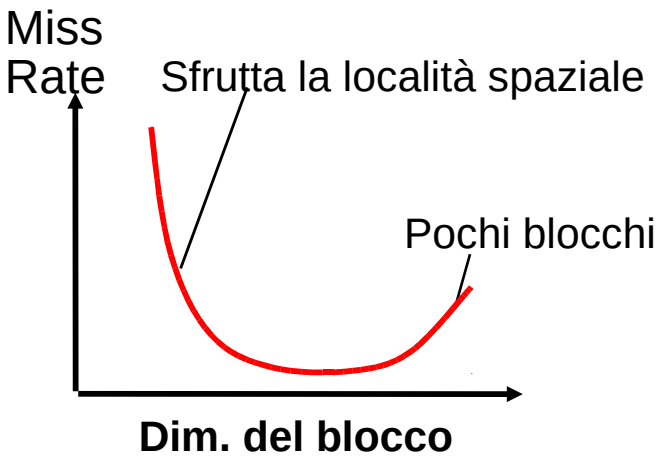
- Dimensione della cache = 4 bytes dimensione del blocco = 4 bytes
  - cache con 1 solo blocco a massima ampiezza
- Se si realizza l'accesso ad un dato, è probabile che presto vi sarà ancora un accesso, ma è improbabile che questo nuovo accesso si verificherà immediatamente.
  - L'accesso successivo produrrà ancora un miss
    - si trasferiscono continuamente dati verso la cache, scaricandoli prima che possano essere nuovamente usati
    - Effetto Ping Pong

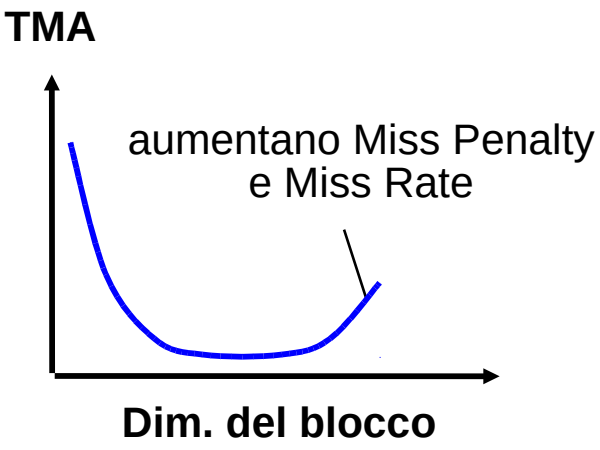
Scelta della dimensione del blocco

- In generale, una ampia dimensione per il blocco permette di sfruttare la località spaziale, MA:
  - blocchi di grossa dimensione comportano maggiori miss penalty:
    - è necessario più tempo per trasferire il blocco
  - se la dimensione del blocco è troppo grossa rispetto alla dimensione della cache, il miss rate aumenta
    - il numero di blocchi nella cache è insufficiente
- Bisogna minimizzare il tempo medio di accesso:

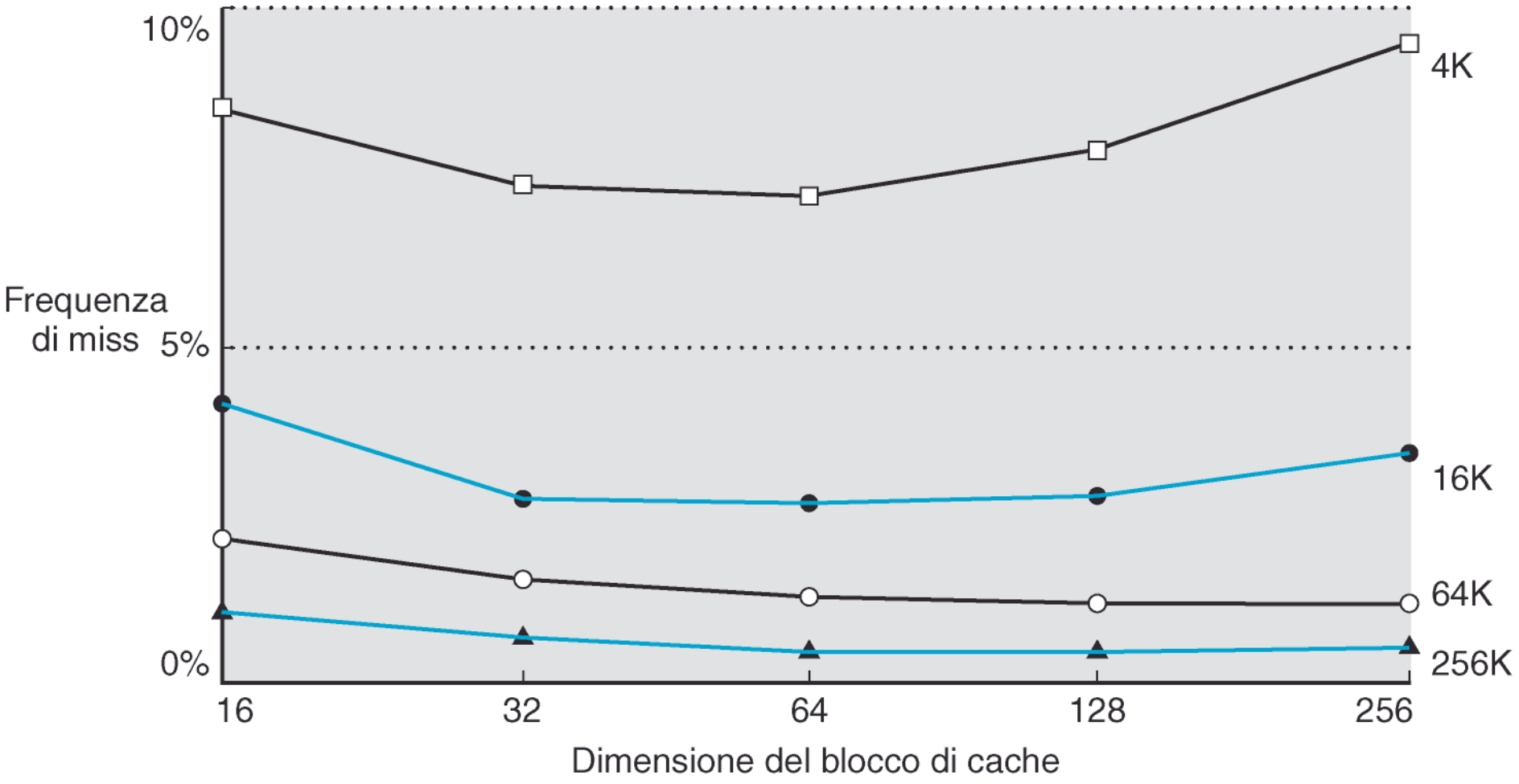
  Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate







Prestazioni al variare della dimensione del blocco

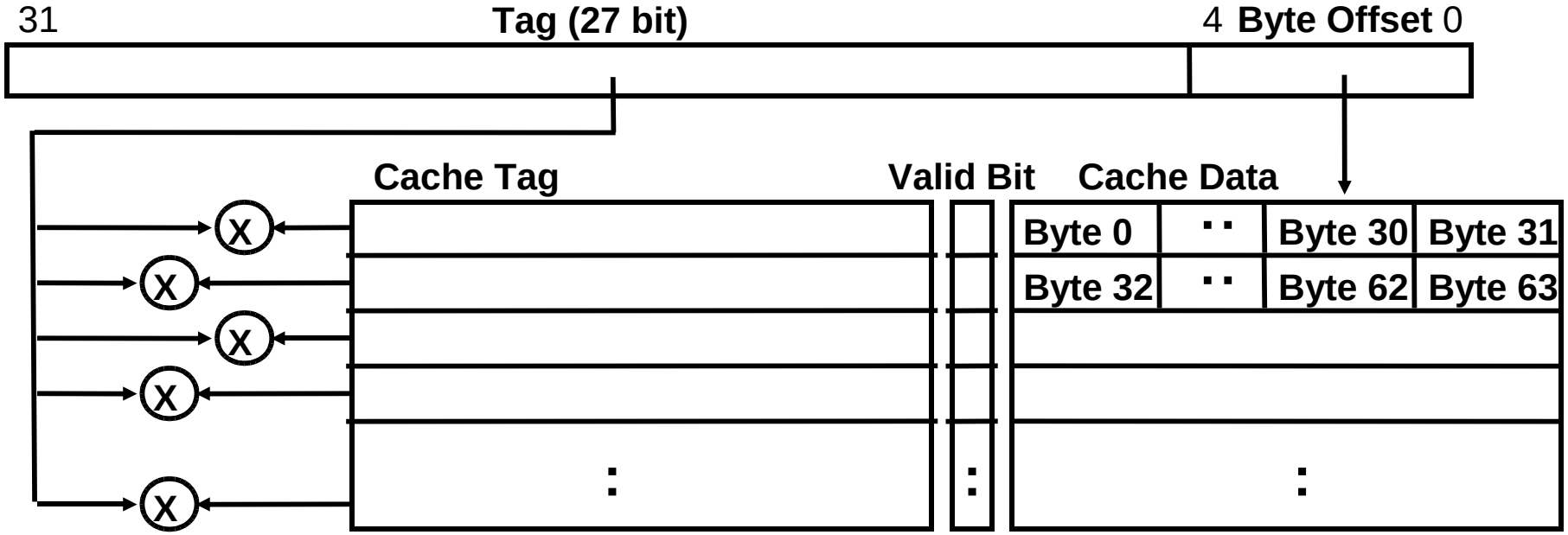


Eliminare i conflitti ...

- Nella cache ad accesso diretto i miss sono interamente dovuti a conflitto sull'indice di cache (*Conflict Miss*), in quanto indirizzi di memoria diversi sono mappati sullo stesso indice.
- Possibili soluzioni:
    - aumentare la dimensione della cache
    - realizzare un accesso indipendente dall'indice di cache (*cache fully associative*)

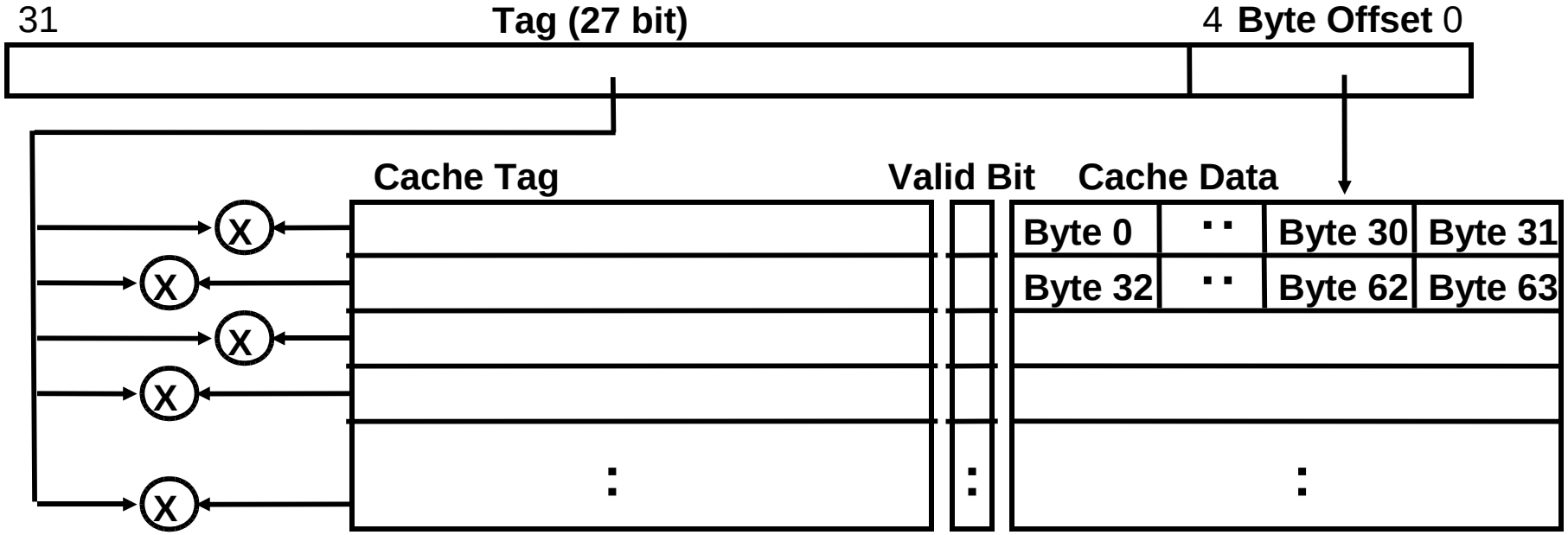
Cache Fully Associative (1/2)

- non viene impiegata alcuna indicizzazione
- vengono confrontati in parallelo i tag di tutti i blocchi appartenenti alla cache
- Esempio:
  - indirizzo da 32 bit
  - dimensione di blocco = 32 byte -> tag da 27 bit
  - N blocchi -> necessari N comparatori da 27 bit



Cache Fully Associative (1/2)

- non viene impiegata alcuna indicizzazione
- vengono confrontati in parallelo i tag di tutti i blocchi appartenenti alla cache
- Esempio:
  - indirizzo da 32 bit
  - dimensione di blocco = 32 byte -> tag da 27 bit
  - N blocchi -> necessari N comparatori da 27 bit

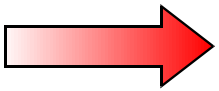


Cache fully associative (2/2)

- Per una cache fully associative il Conflict Miss è nullo per definizione.
- I miss sono generati solo dalla capacità insufficiente della cache (*Capacity Miss*)
- Problemi:
    - hardware molto complesso
    - praticamente realizzabile solo con un piccolo numero di blocchi

Cache Set Associative

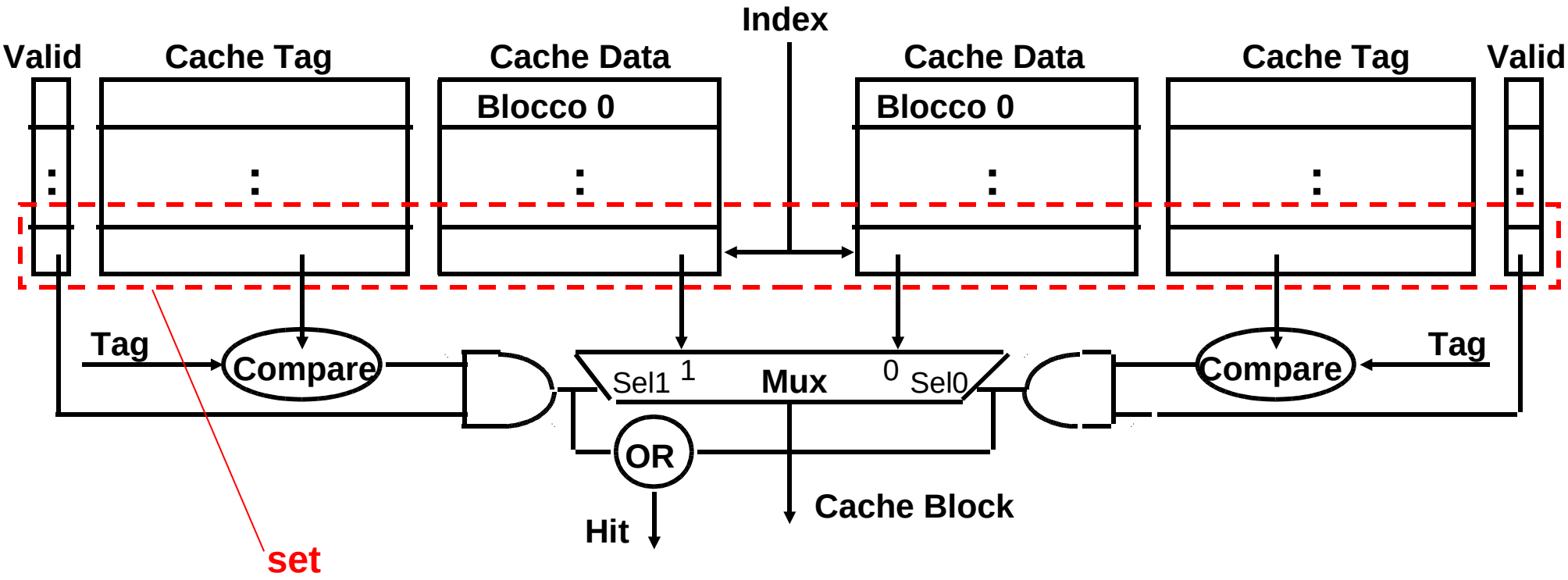
- I blocchi appartenenti alla cache sono raggruppati in *set*
- in una cache set associative ad N vie (N-way set associative) ogni set raggruppa N blocchi
- ogni indirizzo di memoria corrisponde ad un unico set della cache (accesso diretto tramite indice) e può essere ospitato in un blocco qualunque appartenente a quel set
- stabilito il set, per determinare se un certo indirizzo è presente in un blocco del set è necessario confrontare in parallelo i tag di tutti i blocchi



Vengono combinati gli approcci ad accesso diretto e fully associative

Esempio: Cache 2-way Set Associative

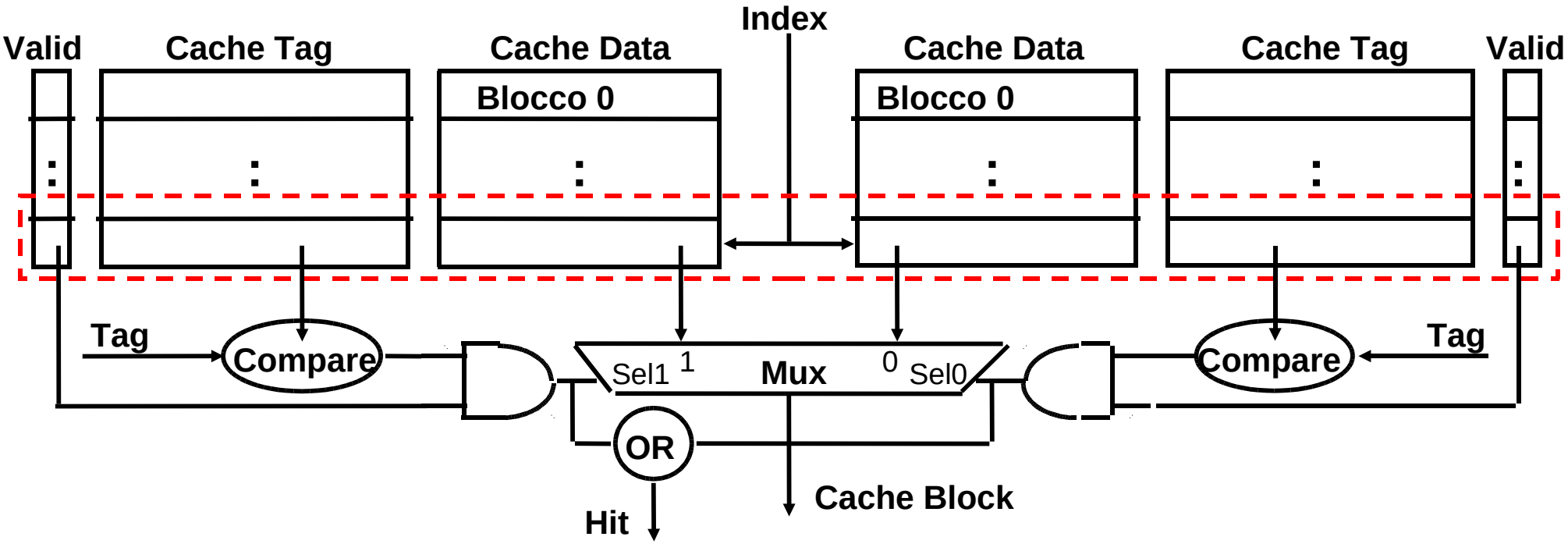
- tramite il campo indice dell'indirizzo viene selezionato un set
- i due tag nel set sono confrontati in parallelo
- il blocco viene selezionato sulla base del risultato dei confronti



Svantaggi della cache set associative

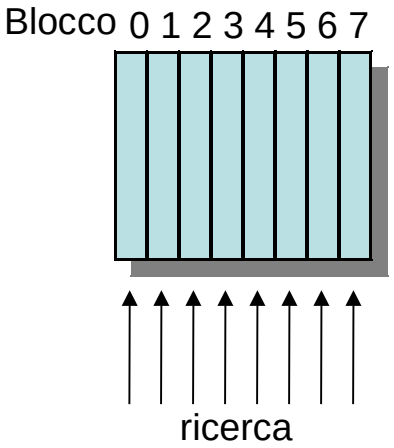
- Cache N-way set associative a confronto con cache ad accesso diretto:
  - N comparatori invece di 1
  - ulteriore ritardo fornito dal multiplexer
  - il blocco è disponibile dopo la decisione Hit/Miss e la selezione del set

- In una cache ad accesso diretto, il blocco è diponibile prima della decisione Hit/Miss

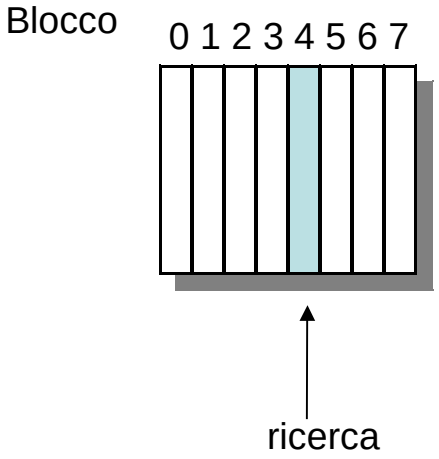


Confronto tra le tecniche di indirizzamento

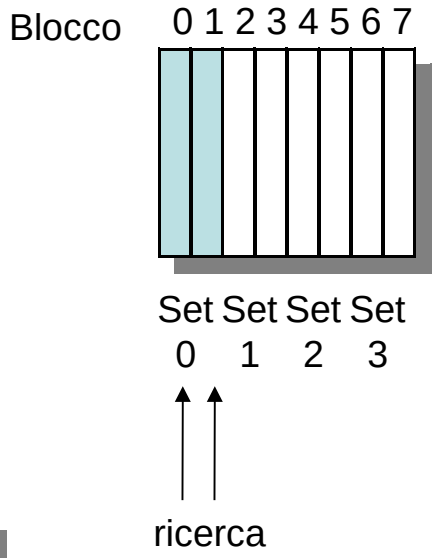
Fully associative:
il blocco 12 può essere disposto ovunque

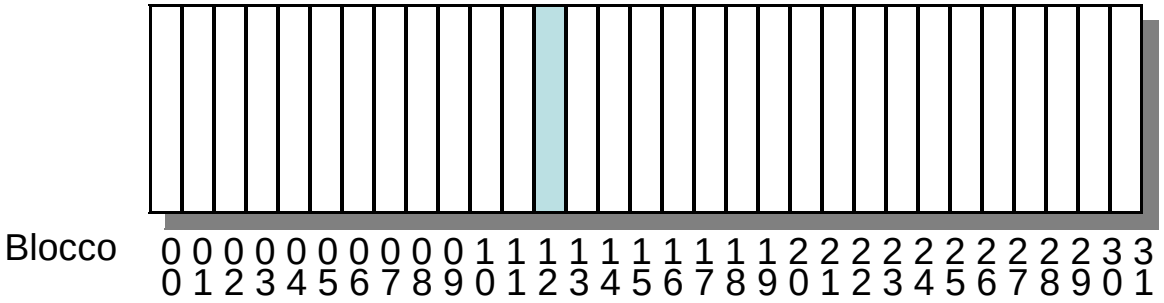


Direct mapped:
il blocco 12 può essere disposto solo sul blocco 4 (12 mod 8)



Set associative:
il blocco 12 può essere disposto ovunque nel set 0 (12 mod 4)





Algoritmi per la sostituzione di blocchi (1/2)

- In corrispondenza di un miss, è necessario introdurre nella cache un nuovo blocco, eventualmente sostituendo un blocco già presente.
- Nella cache ad accesso diretto non c'è scelta, in quanto i miss sono generati da conflitto sull'indice (conflict miss).
- Nelle cache associative (N-way Set Associative o Fully Associative) c'è la possibilità di scegliere quale blocco sostituire (capacity miss).
- Nel caso ci sia un blocco non valido, si sostituisce questo.
- Altrimenti, due schemi possibili:
  - **LRU**
  - **Random**

Algoritmi per la sostituzione di blocchi (2/2)

- **LRU**
  - Viene sostituito il blocco che è stato utilizzato meno di recente (*Least Recently Used*). Solitamente, è il blocco che ha minore probabilità di essere impiegato di nuovo.
  - E' quindi necessario tenere memoria dei cache hit relativi ad ogni blocco.
- **Random**
  - LRU difficile da gestire in caso di grado di associatività elevato (possibile per una cache 2-way set associative, proibitivo per un'associatività > 4)
  - Alternativa: si sostituisce un blocco a caso.
  - Vantaggi
    - facile da implementare (soprattutto in HW)
    - comportamento predicibile
    - non presenta comportamenti di tipo worst case

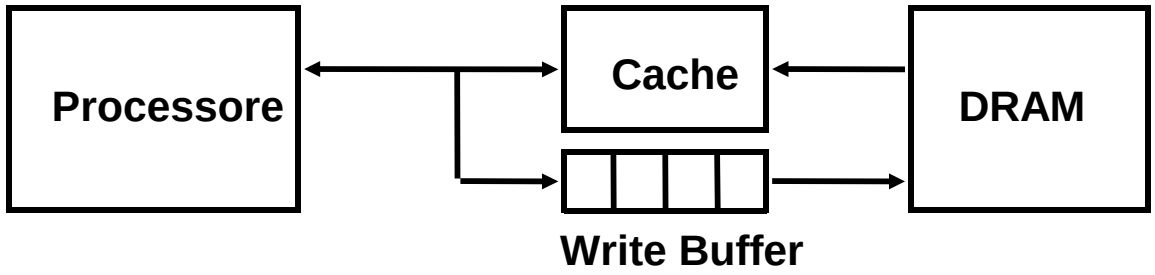
Gestione delle operazioni di scrittura

- In conseguenza di un'operazione di scrittura, effettuata su un blocco presente nella cache, i contenuti di quest'ultima saranno diversi dai contenuti della memoria di livello inferiore.
- Soluzione più semplice: il dato viene scritto sia nel blocco della cache sia nel blocco contenuto nella memoria di livello inferiore. (**Write through**)
- Problema: in questo modo le operazioni di scrittura vengono effettuate alla velocità della memoria di livello inferiore !
- Alternative:
  - modalità **Write Back**
  - utilizzo di un **Write Buffer**

Modalità Write Back

- I dati sono scritti solo sul blocco presente nella cache
- Il blocco modificato è trascritto in memoria principale solo quando viene sostituito
  - Il blocco può essere in due stati: non modificato (clean) o modificato (dirty)
- Vantaggi:
  - Scritture successive sulla stessa locazione alterano solo la cache
- Svantaggi:
  - Ogni sostituzione di blocco (dovuti p.es. a read misses) può provocare un trasferimento in memoria

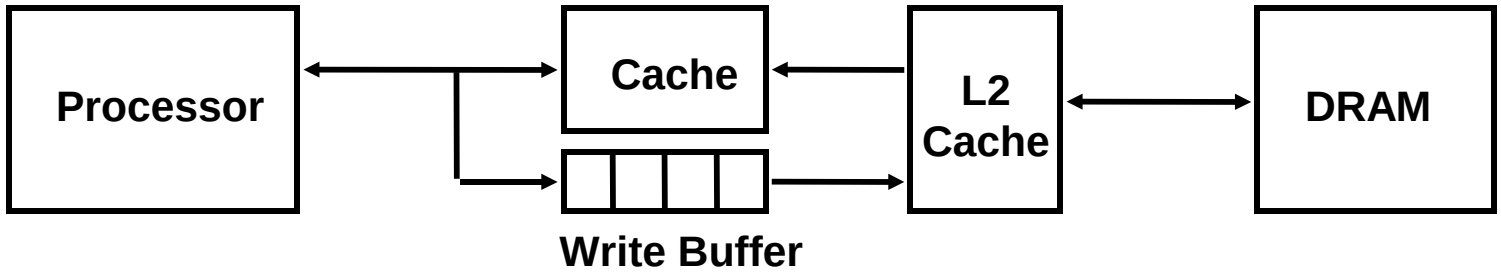
Write Through con Write Buffer



- Viene disposto un buffer per la scrittura (Write Buffer) tra la cache e la memoria
  - il processore scrive i dati nella cache e nel buffer
  - il controller della memoria scrive i contenuti del buffer in memoria
- Il Write Buffer funziona con strategia FIFO:
  - numero tipico di elementi: 4
  - funziona se la frequenza di scrittura << 1 / write cycle della DRAM
  - altrimenti, il buffer può andare in saturazione.

Saturazione del Write Buffer

- Nel caso in cui la frequenza di scrittura è superiore a 1 / DRAM write cycle (es.: ciclo della CPU troppo breve o troppe istruzioni di scrittura successive), il buffer andrà in overflow qualunque sia la sua dimensione.
- Possibili soluzioni:
  - modificare la modalità della cache in write back
  - inserire una cache di secondo livello (L2) di tipo write back



Gestione del write miss

- In corrispondenza di un read miss, bisogna trasferire il blocco relativo nella cache così da poter completare la lettura. Come operare in presenza di un write miss ?
- Opzione 1: stessa tecnica impiegata nel read. L'intero blocco viene trasferito nella cache dove viene poi modificato (Write Allocate).

- Opzione 2: viene modificata solo la memoria di livello inferiore; la cache non subisce alcuna modificata (No Write Allocate)

Cache a più livelli: perché ?

- Obiettivo: minimizzare il tempo medio di accesso:

  TMA = Hit Time x (1 - Miss Rate)
  + Miss Penalty x Miss Rate

- Finora si è cercato di diminuire il Miss Rate agendo su:
  - dimensioni del blocco
  - dimensioni della cache
  - grado di associatività

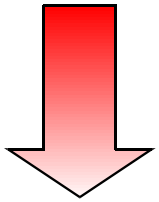
Cache a più livelli: perché ?

- Opportuno agire anche sul Miss Penalty
- Inizialmente, il Miss Penalty era intorno ai 10 cicli di clock di processore
- Attualmente, per un processore a 2,4 GHz (T=0.4 ns), con una DRAM che richiede 80 ns per un ciclo completo di lettura/scrittura il Miss Penalty è circa 200 cicli di clock

- Soluzioni possibili:
  - Aumento delle prestazioni della memoria centrale ($$$)
  - introduzione di una cache di secondo livello (*L2 Cache*)

Analisi del TMA con cache a più livelli

- TMA = L1 hit time * L1 hit rate + L1 miss penalty * L1 miss rate
- Nel valutare il miss penalty per la cache L1 bisogna ora considerare che un miss penalty su L1 comporta un accesso su L2



TMA =

L1 hit time * L1 hit rate +

(L2 hit time * L2 hit rate + L2 miss penalty * L2 miss rate)* L1 miss rate

Un po' di conti

- Ipotesi:
  - L1 hit time = 1 ciclo
  - L1 hit rate = 95%
  - L2 hit time = 5 cicli
  - L2 hit rate = 85%
  - tempo di ciclo della memoria principale = 200 cicli
- Senza cache L2:
  TMA = 1*0.95 + 200 * (1-0.95)
  = 0.95 + 200 * 0.05 = **10.95** cicli di clock
- Con la cache L2:
  TMA = 1*0.95 + (5*0.85 + 200*0.15) *(1-0.95)
  = 0.95 + (34.25) * 0.05 = **2.66** cicli di clock

Parametri tipici

- **Cache di primo livello**
  - Dimensione: decine di Kb
  - Hit time: ~ 1 ciclo di clock
  - Miss rate: 1-5 %
  - Divisa in cache istruzioni e cache dati
- **Cache di secondo livello**
  - Dimensione: centinaia di Kb
  - Hit time: < 10 cicli di clock
  - Miss rate: 10-20 %
  - Comune a istruzioni e dati

Un caso reale: Intel Core I7 (Nehalem)

- **L1:** 64 Kb/ core
  - 32 Kb istruzioni + 32 Kb dati
  - 8-way set associative
  - 64 byte/linea
- **L2:** 256 Kb/core
  - 8-way set associative
  - 64 byte/linea
- **L3:** 8 Mb *Smart Cache*©
  - Ripartita dinamicamente tra i core ed il processore grafico