Memoria cache

Argomento:

- Gerarchia di memoria
- Principio di località di riferimento
- Organizzazione della cache

Materiale di studio

- Capitolo 4

Memoria

- Le principali caratteristiche di una memoria sono:
  - Dimensione
  - Tempo di accesso
  - Costo per bit

- Esistono dei compromessi:
  - Minor tempo di accesso, maggior costo per bit
  - Maggior capacità, minor costo per bit
  - Maggior capacità, maggior tempo di accesso

Memorie RAM

- Le memorie RAM garantiscono grandi dimensioni e basso costo per bit, ma sono lente!

- La velocità di trasferimento dati dalla memoria al processore è molto inferiore rispetto alla velocità di elaborazione dati della CPU.

- Molte computazioni sono rallentate dall'accesso alla memoria e non dalla CPU
  - La CPU spesso deve aspettare i dati in arrivo dalla memoria!

- E' improbabile che il gap venga colmato tecnologicamente

CPU vs memoria DRAM

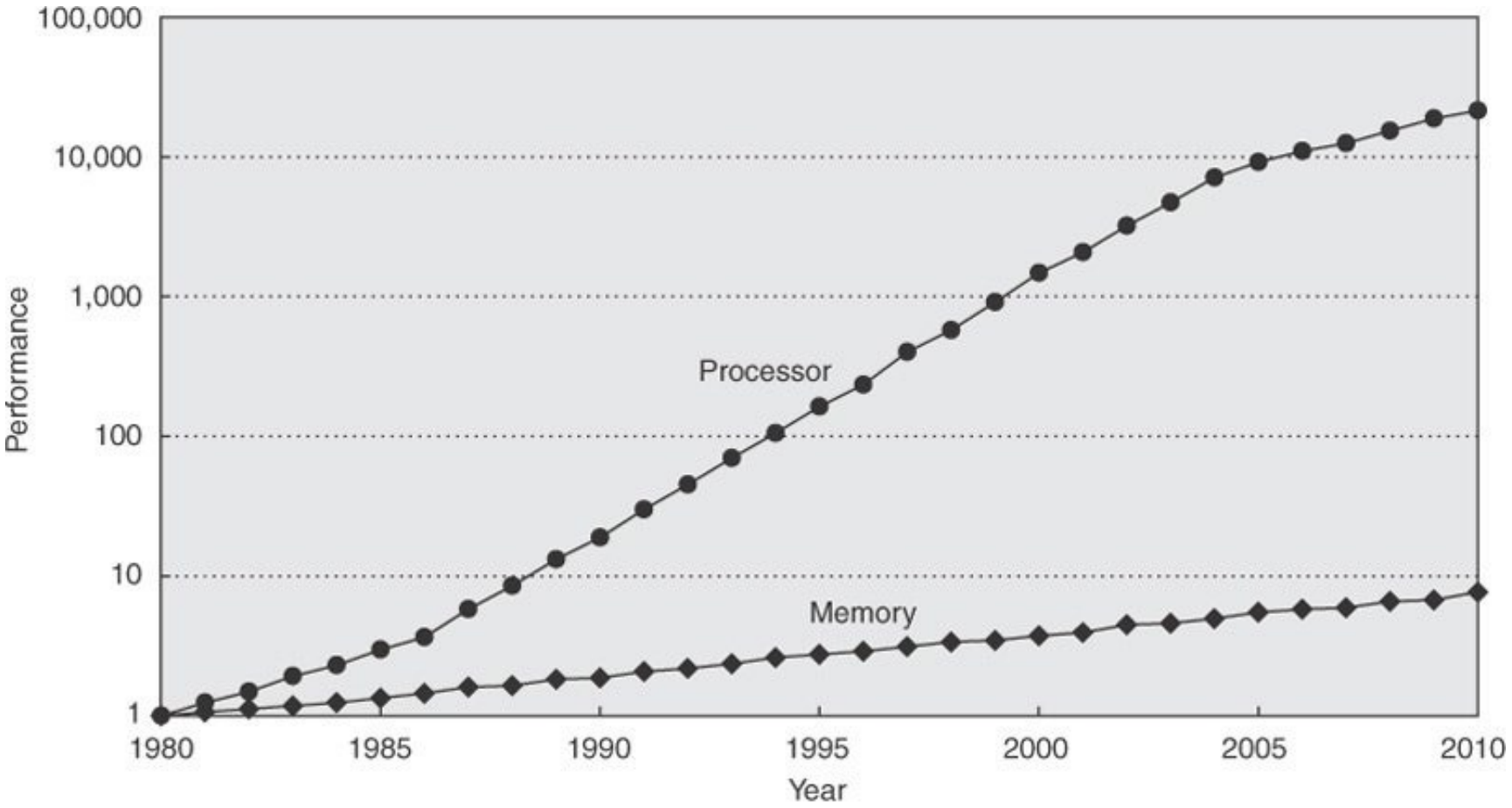


Immagine da Hennessy and Patterson,
Computer Architecture, 2012

Gerarchia di memoria

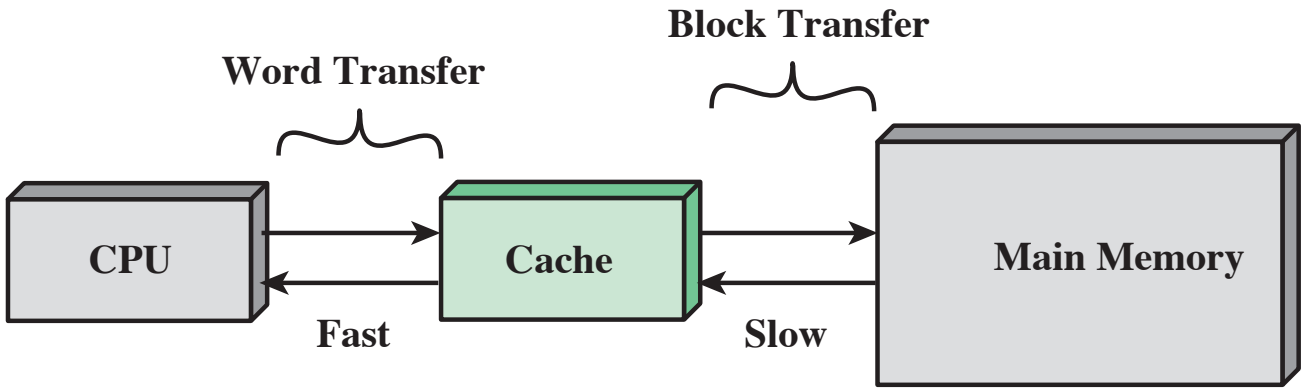
Come è possibile ridurre questo gap?

- Esistono memorie più veloci della RAM (ad es. 7 ns, invece di 70 ns) ma sono costose e di estensione ridotta (ad es. 512 Kbyte).

- Non è possibile realizzare una memoria che sia grande, veloce ed economica ma è possibile sviluppare una **gerarchia di memoria** che soddisfa tutti i requisiti.

Gerarchia di memoria

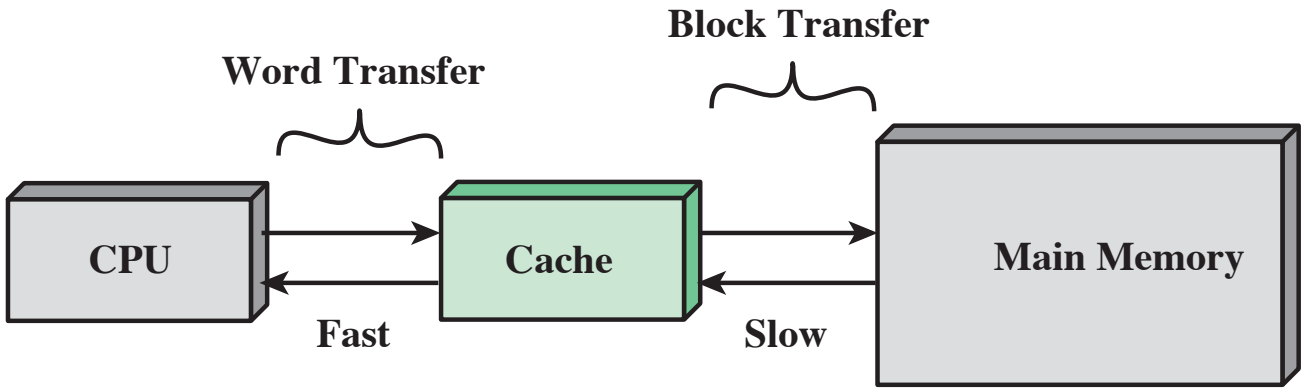
- La **memoria cache** è una piccola memoria che viene posizionata tra la memoria RAM e la CPU
  - È molto veloce rispetto alla memoria RAM
  - Ha estensione limitata ed è costosa (in € e in area)
- La cache contiene una copia dei dati presenti in memoria
- Per ogni richiesta di accesso ad una word W in memoria, il processore controlla se la word è presente in cache.



Gerarchia di memoria (2)

- Se la word W è presente (**cache hit**), la word viene consegnata al processore in pochi cicli di clock.

- Se la word non è presente (**cache miss**), un blocco di word consecutive contenenti W viene caricato in cache e la word consegnata al processore.
  - Operazione costosa: accede alla memoria RAM



Quando la gerarchia di memoria funziona?

1. Quando il processore riutilizza frequentemente gli stessi dati → **Località di riferimento temporale**

2. Quando il processore richiede dati "vicini" in memoria in un breve intervallo di tempo → **Località di riferimento spaziale**

Esempi di località di riferimento

Somma degli elementi di un vettore V di n elementi:

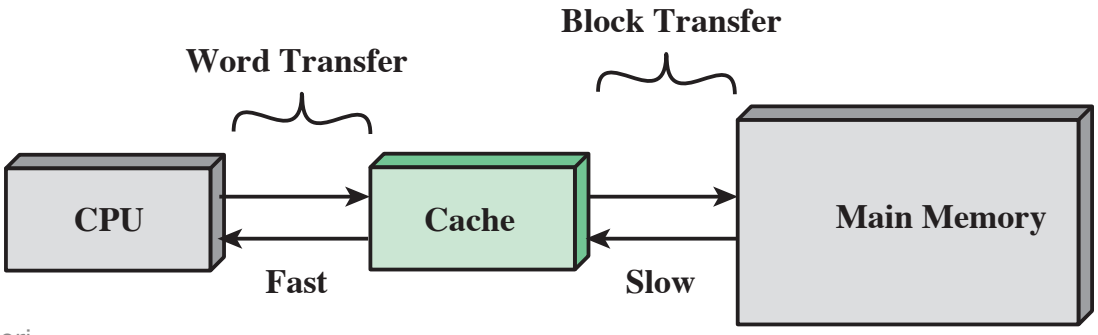
```
int sum = 0
for(int i =0; i<n; i++) sum = sum + V[i]
```

Località spaziale: il vettore V viene letto una posizione dopo l'altra.

Località temporale: le variabili `sum` e `i` vengono usate ad ogni iterazione

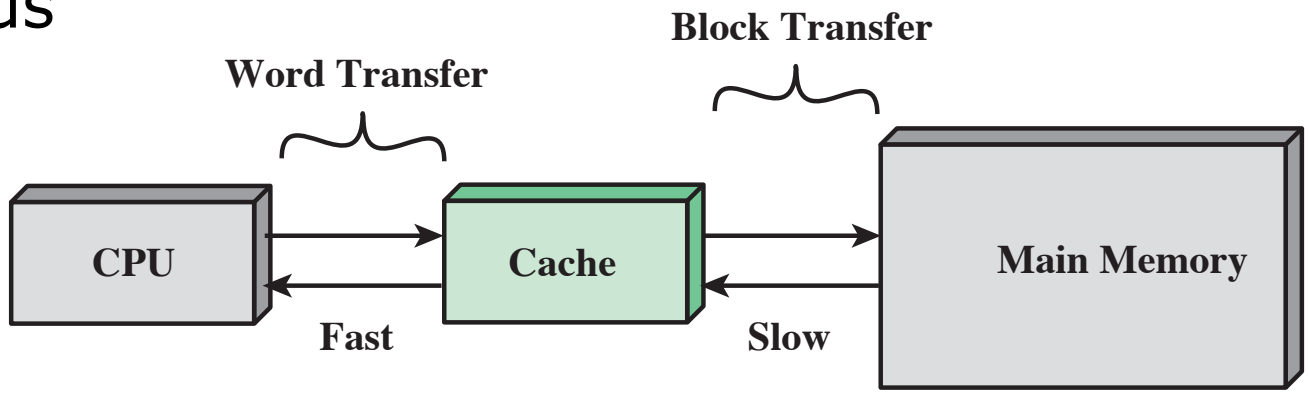
Località temporale

- Se i dati più utilizzati vengono mantenuti in cache, allora l'accesso risulta più veloce.
- I dati meno utilizzati risiedono in memoria: il loro accesso costa di più, ma vengono richiesti raramente
- Problemi:
  - La cache ha una dimensione limitata
  - Non è noto quali dati verranno utilizzati più frequentemente nel futuro



Località spaziale

- Dati "vicini" in memoria significa dati con indirizzi vicini
- Quando si accede in memoria per un dato all'indirizzo W, posso essere caricate le word "vicine", ad esempio quelle ad indirizzo W, W+1, W+2,…, W+K
- **Block transfer**: viene trasferito un blocco di K words



Costo medio di accesso alla memoria

- T_1: tempo per accedere ad un dato in cache
- T_2: tempo per accedere ad un dato in memoria
- $T_1 << T_2$
- p = percentuale degli accessi a dati in cache (**hit rate**)

- Costo medio T di un accesso:

$$\mathbf{T = T_1\, p + T_2\,(1\text{-}p)}$$

Costo medio di accesso alla memoria (2)

Esercizio:

- Cache hit: p = 95%
- Costo accesso cache T_1 = 0.01 μs
- Costo accesso memoria T_2 = 0.1 μs

- Costo medio

$T = T_1\, p + T_2\,(1\text{-}p) =$

$= 0.95 * 0.01\ \mu s + 0.05 * 0.1\ \mu s = 0.0145\ \mu s$

Esempio di riduzione accessi

Somma degli elementi di un vettore V di n elementi:

```
int sum = 0
for(int i =0; i<n; i++) sum = sum + V[i]
```

L'algoritmo richiede (circa):

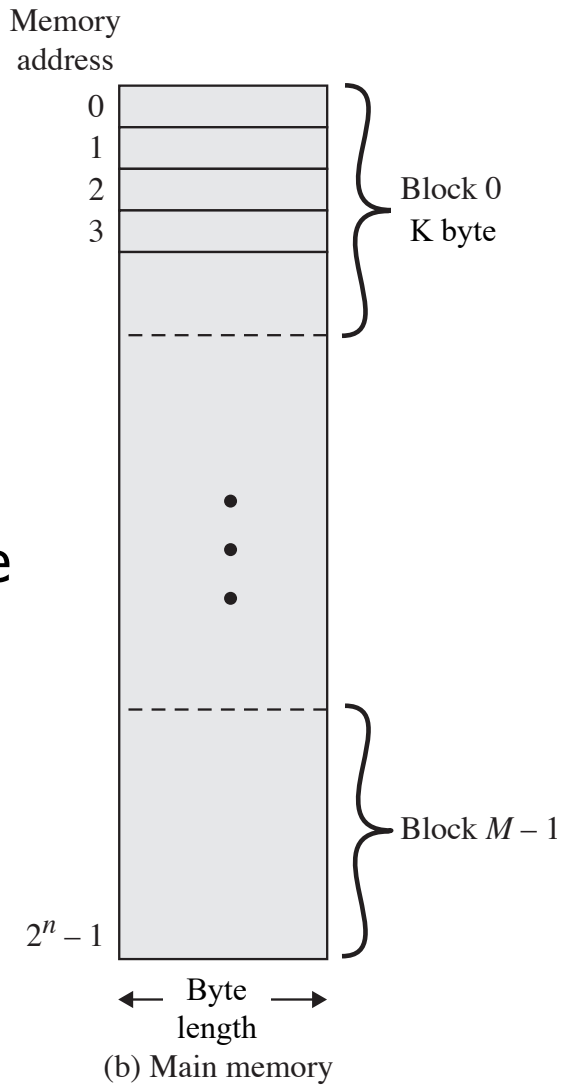
- n accessi in cache per recuperare gli elementi di V
- n/K accessi alla memoria per recuperare gli n/K blocchi che contengono V

- Senza cache, l'algoritmo richiederebbe n accessi in memoria

Struttura di cache e memoria

Memoria:

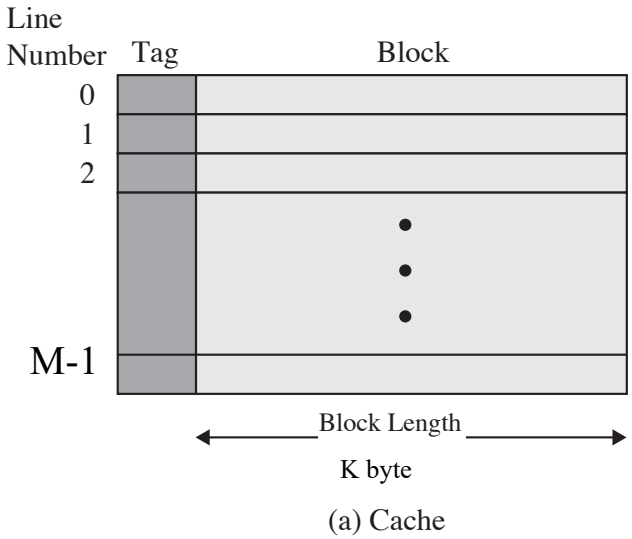
- Locazioni di un byte e indirizzi su n bit
- Accesso per word: indirizzo della word è l'indirizzo del byte meno significativo
- Locazioni raggruppati in **blocchi** di K byte



Struttura di cache e memoria

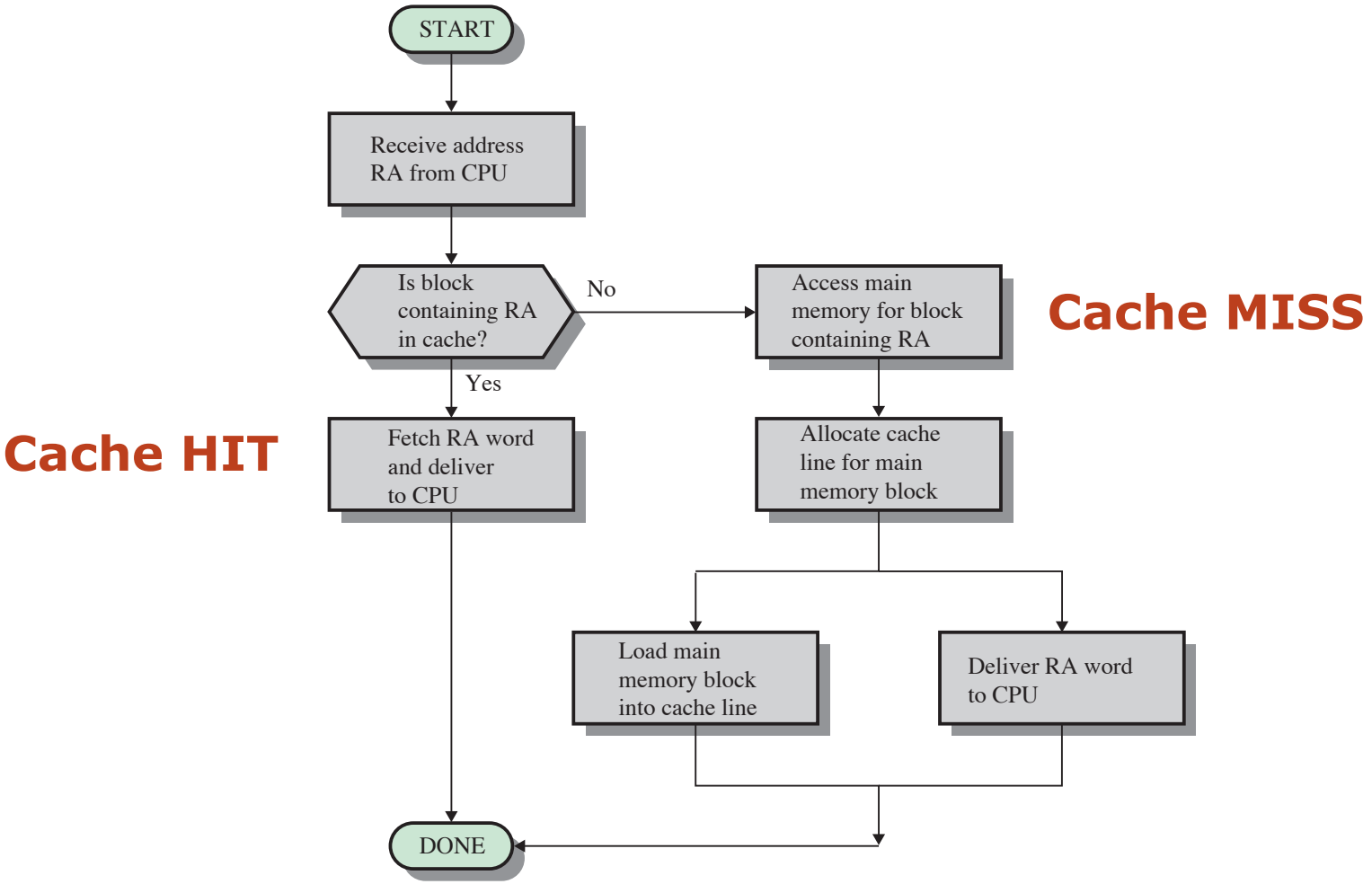
Cache:

- Contiene $M=2^r$ **linee**
- Ogni **linea** contiene un blocco di K byte (2^w byte) e un **TAG** che identifica il contenuto



(a) Cache

Accesso ad un indirizzo in memoria



Indirizzo di un blocco

L'indirizzo di un blocco che contiene una certa word si ottiene dagli n-w bit più significativi dell'indirizzo della word

Esempio:

- n=32 bit; w = 8 bit; s = 24 bit.
- Indirizzo 0xABCD1248 → blocco 0xABCD12; posizione 0x48 nel blocco

Tipi di cache

Come vengono mappati i blocchi di memoria sulle linee di cache?

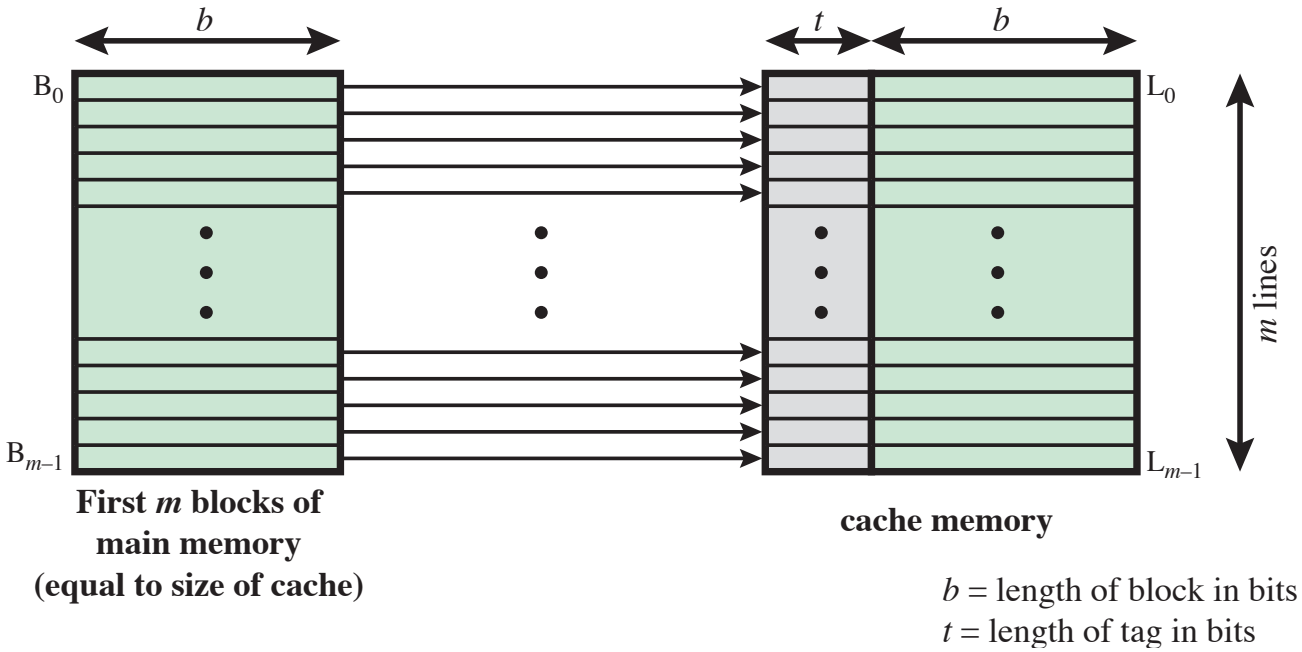
Tre approcci:

- Cache a **mappature diretta**
- Cache **completamente associativa**
- Cache **associativa a k-vie**

Cache a mappatura diretta

Nella **cache a mappatura diretta**, il blocco in memoria con indice j viene mappato sulla linea i con:

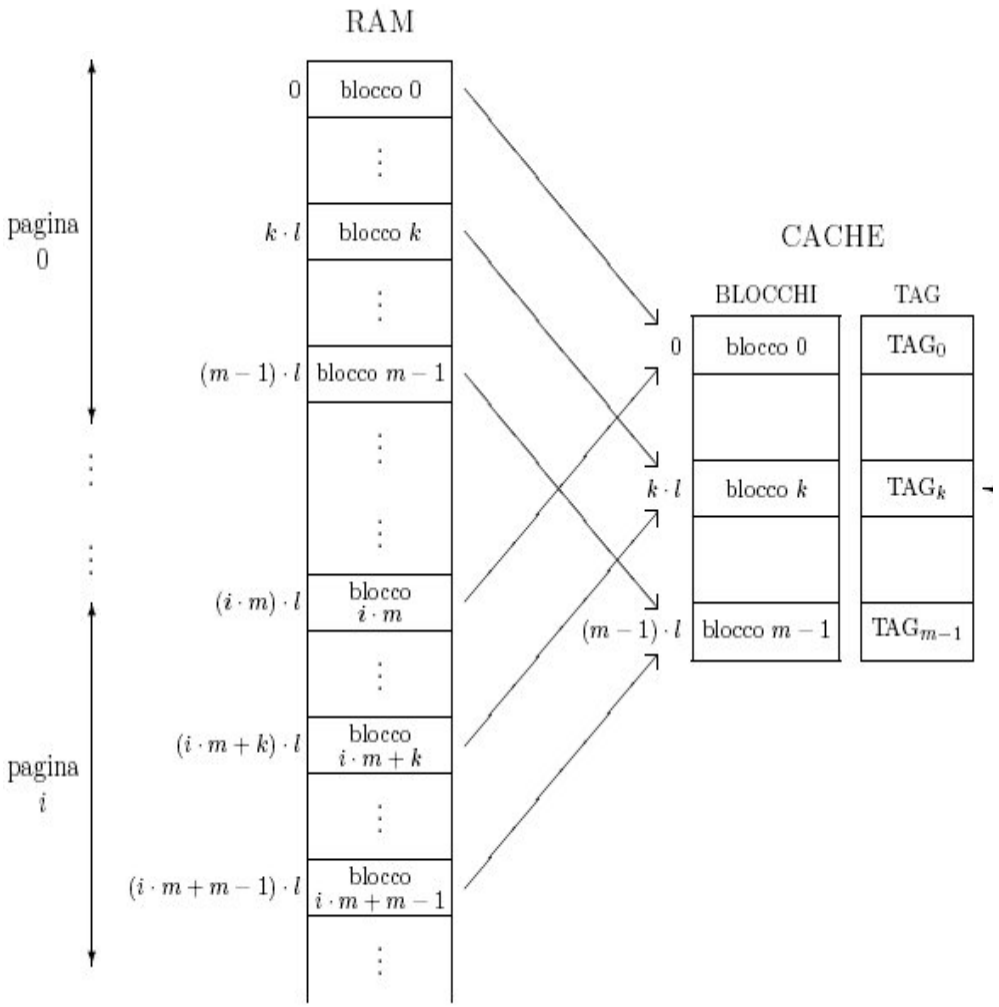
i = j mod m



Cache a mappatura diretta (2)

E' come se la RAM fosse divisa in pagine e ciascuna pagina in blocchi:

- pagine di dimensione uguale a quella della cache
- blocchi di dimensione uguale a quelli della cache,
- indice di un blocco all'interno della propria pagina è l'indice della linea in cui sarà caricato nella cache.



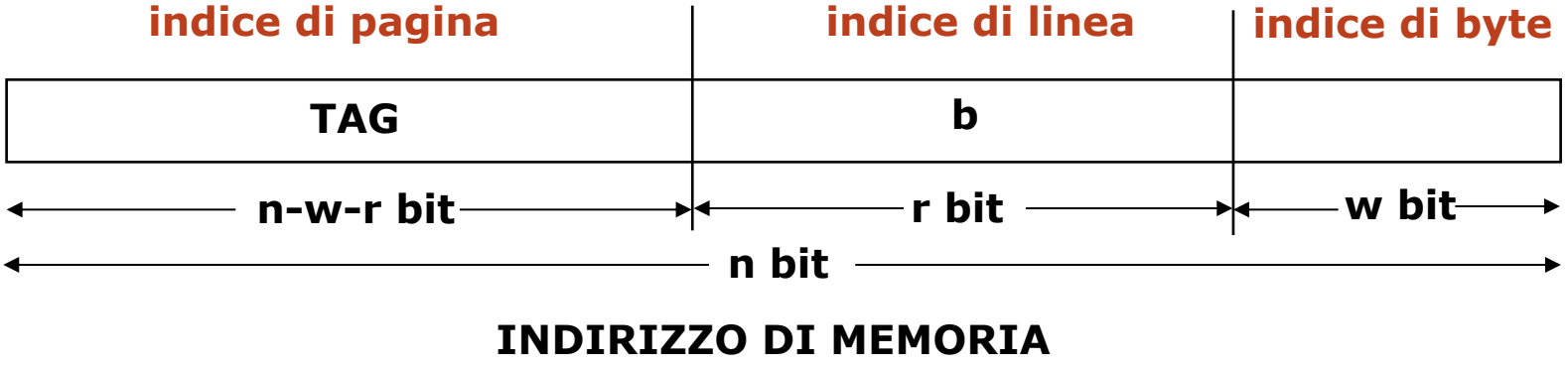
Indirizzi nella cache a mappatura diretta

Gli indirizzi di memoria sono divisi in campi:

- i w bit meno significativi: indice del byte nel blocco;
- i rimanenti s = n-w bit comuni a tutti i byte del blocco;

Questi s bit sono ora così ripartiti:

- r bit meno significativi: indice b della linea di cache
- n-w-r bit: TAG = indice della pagina nella RAM.



Esempio di indirizzo

- Memoria con indirizzi da 32 bit (n=32)
- Cache da 4K linee (m=2^{12}, r=12)
- Linee/blocchi da 256=2^8 byte

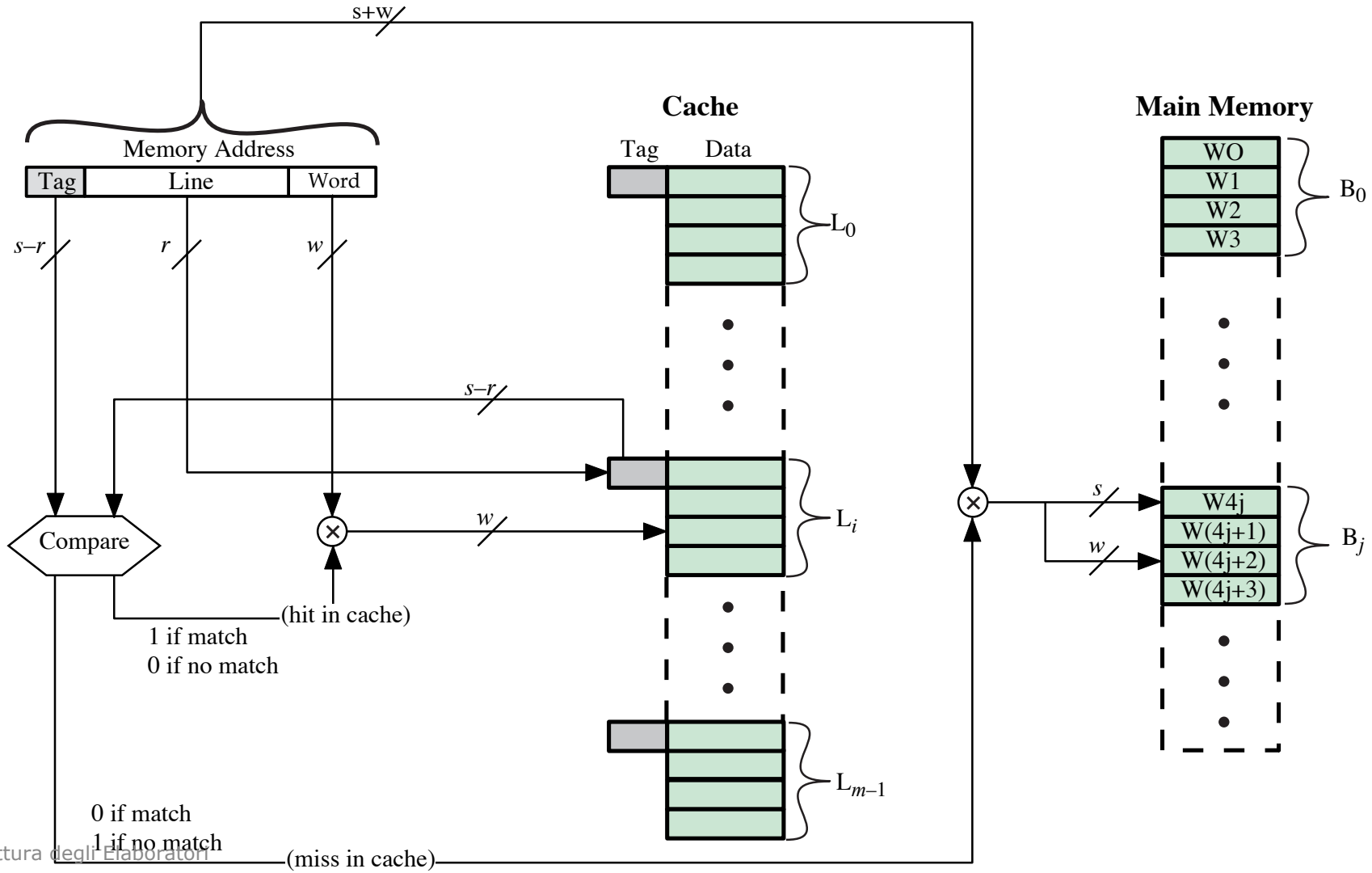
- Indirizzo byte 0xABCD1248
  - Indirizzo del byte nel blocco: 0x48
  - Indirizzo blocco (s=12 bit): 0xABCD12
  - Indirizzo pagina (TAG, n-r-w = 12 bit): 0xABC
  - Indirizzo linea in cache (r bit): 0xD12

Funzionamento cache a mappatura diretta

Quando il processore genera un indirizzo di memoria:

- Gli r bit centrali forniscono l'indice b della linea che potrebbe contenere il blocco.
- Il TAG dell'indirizzo (n-w-r bit più significativi dell'indirizzo) viene confrontato con il TAG associato alla linea b.
  - Se uguale (HIT), l'accesso si risolve nella cache
  - Altrimenti (MISS), il blocco b-esimo nella cache appartiene ad un'altra pagina, e allora:
    - viene recuperato il blocco con un accesso alla RAM
    - il blocco di RAM e il suo TAG vengono inseriti nella linea b della cache

Funzionamento cache a mappatura diretta (2)



Mappatura diretta: pro e contro

- **Pro**:
  - Organizzazione semplice

- **Contro**:
  - La linea da sovrascrivere è predeterminata: un blocco non può essere inserito in una linea scelta tra quelle che presumibilmente non servono più
  - Hit rate decisamente più basso (minore efficienza).

Limite della mappatura diretta

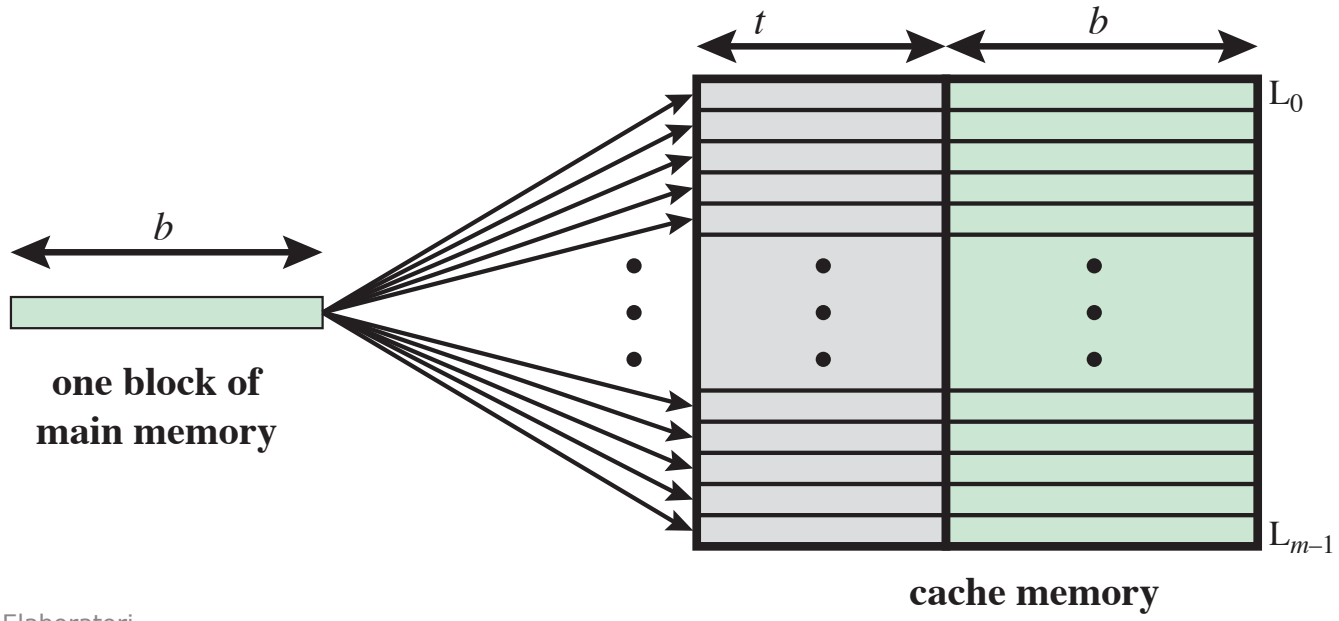
- Si considerino gli accessi agli indirizzi (n=32, w=8, r=12) con cache inizialmente vuota:

  0x00000000, 0xFF000000, 0x00000080, 0xFF000020

- Problema:
  - L'indirizzo 0x00000000 carica il blocco 0x000000 nella linea 0
  - L'indirizzo 0xFF000000 carica il blocco 0xFF0000 nella linea 0 e sovrascrive il valore precedente anche se le altre linee di cache sono vuote
  - L'indirizzo 0x00000080 carica nuovamente il blocco 0x000000 nella linea 0 e sovrascrive di nuovo il precedente valore
  - L'indirizzo 0xFF000020 carica il blocco 0xFF0000 nella linea sovrascrive di nuovo il precedente valore

Cache completamente associativa

- Nella cache **completamente associativa**, un blocco viene inserito in una linea di cache vuota.
- Se tutte le linee in cache sono occupate, si sceglie quale linea svuotare con una politica di rimpiazzo



Indirizzi nella cache compl. associativa

Se n è il numero di bit di un indirizzo di memoria, gli indirizzi dei K byte di ciascun blocco (2^w byte) hanno gli n-w bit più significativi uguali (TAG), mentre i w bit meno significativi forniscono l'indice di byte all'interno del blocco

Funzionamento cache a completamente associativa (1)

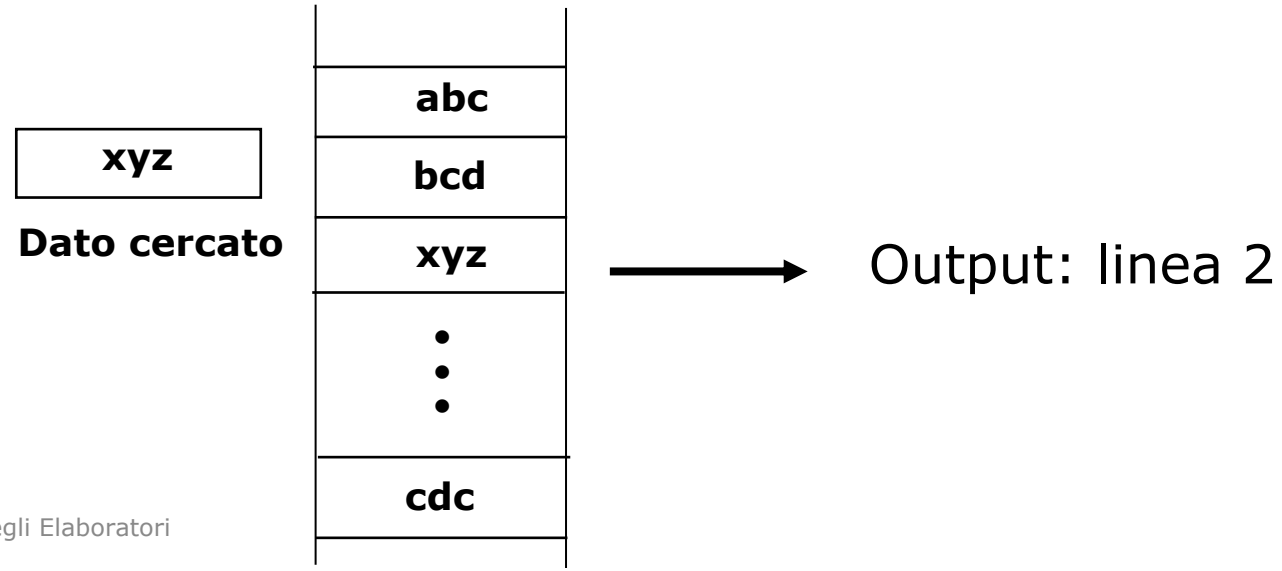
Quando il processore genera un indirizzo di memoria: gli n-w bit più significativi dell'indirizzo vengono confrontati con tutti i TAG delle m linee

- Se il TAG viene trovato (HIT), l'accesso si risolve nella cache
- Altrimenti (MISS), il blocco k-esimo nella cache appartiene ad un'altra pagina, e allora:
  - viene recuperato il blocco con un accesso alla RAM
  - il blocco di RAM e il suo TAG vengono inseriti in una linea vuota della cache; se la cache è piena, una linea viene liberata.

La ricerca del TAG viene velocizzata con l'utilizzo di una memoria associativa

La memoria associativa o CAM

- **Memoria associativa** o CAM (Content Addressable Memory): è una memoria in grado di effettuare, in parallelo, il confronto tra un dato cercato e tutti i dati in essa contenuti.
- La memoria restituisce gli indirizzi dei dati in essa contenuti che siano uguali al dato cercato.

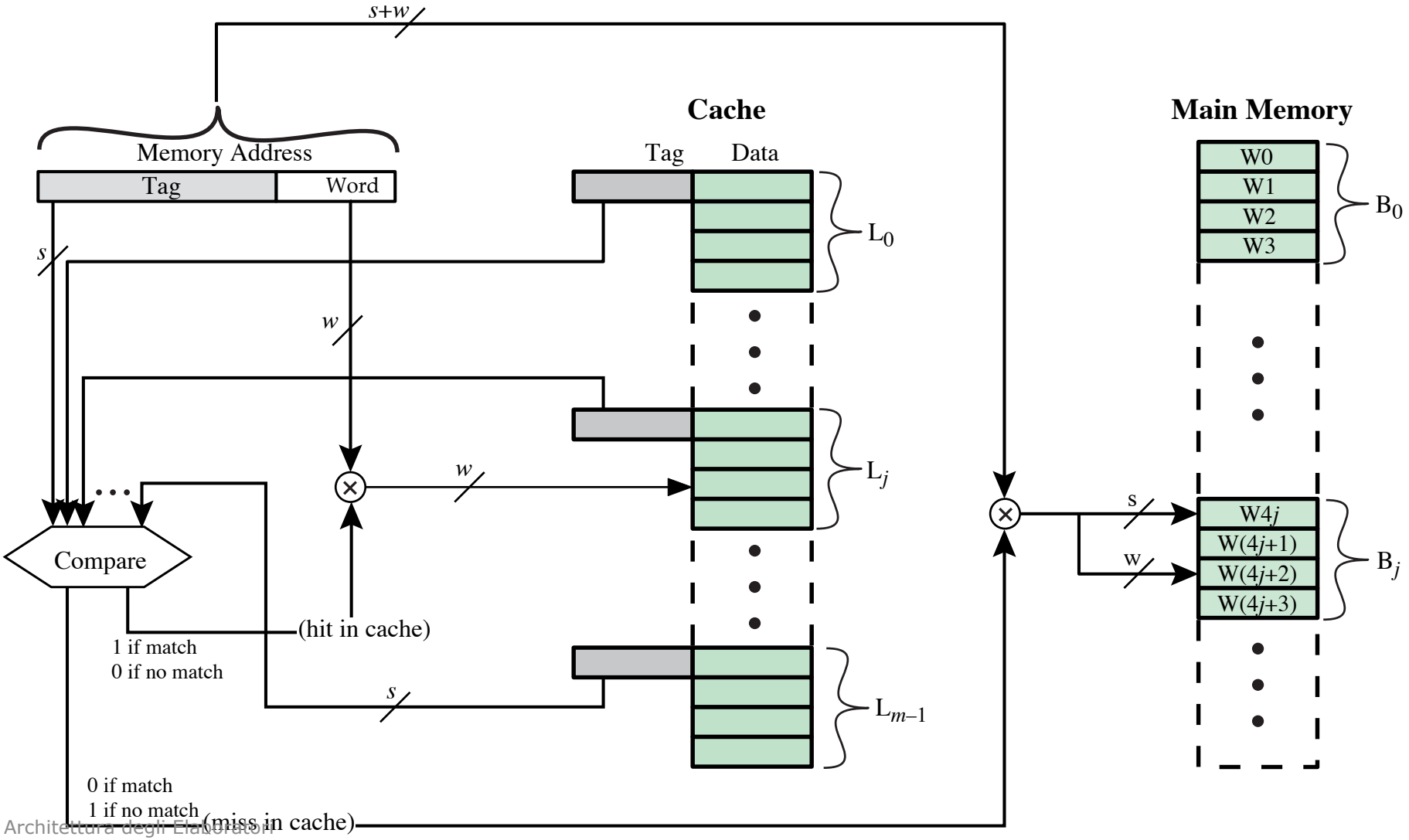


La memoria associativa o CAM (2 di 2)

- La lettura in una memoria associativa è simmetrica rispetto alla RAM:
  - la RAM riceve un indirizzo e restituisce il dato contenuto a quell'indirizzo;
  - la CAM riceve un dato e restituisce l'indirizzo (o gli indirizzi) delle celle che contengono quel dato.

- Di conseguenza l'hardware che realizza una CAM è molto complesso e molto costoso.

Funzionamento cache a completamente associativa



Cache completamente associativa: pro e contro

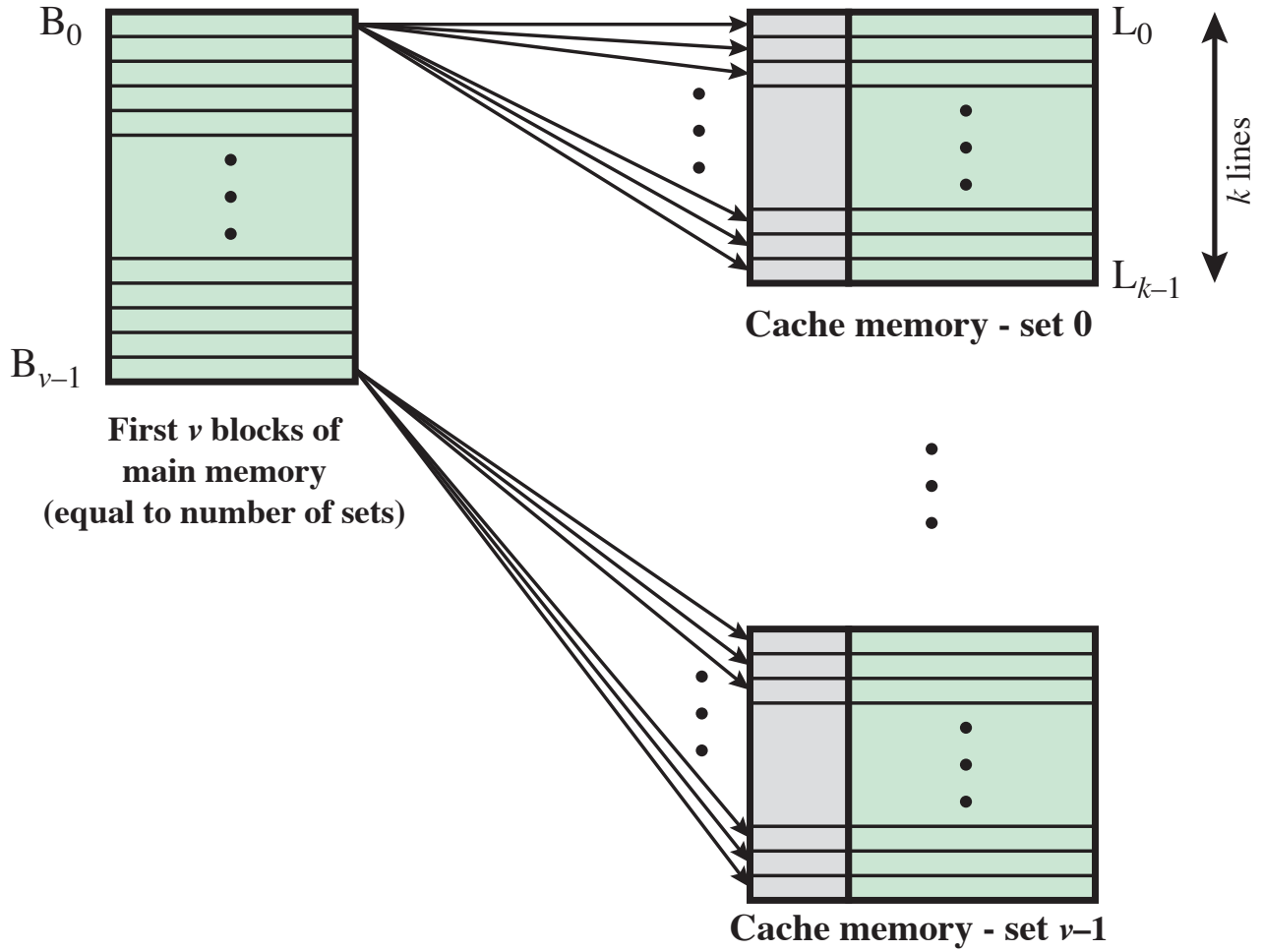
- **Pro**:
  - Hit rate più alto della cache a mappatura diretta (miglior sfruttamento della cache)

- **Contro**:
  - Organizzazione costosa, richiede una CAM

Cache set-associativa a k vie

- Un compromesso tra l'efficienza della organizzazione completamente associativa e il basso costo di quella a mappatura diretta, si può ottenere replicando k volte quest'ultima (**cache set-associativa a k-vie**):
  - per ogni indice di blocco b, anziché esserci una sola linea nella cache, ve ne sono k;
  - a ciascun **set di k linee** è associata una CAM (di k elementi) che contiene i TAG dei blocchi presenti nel set.
- Come nell'organizzazione a mappatura diretta, la RAM è ancora divisa in pagine e ciascuna pagina in blocchi:
  - l'indice di blocco individua il set in cui sarà caricato
  - un blocco di indice b, può essere caricato in una qualsiasi delle k linee nel b-esimo set.

Cache set-associativa a k vie (2)



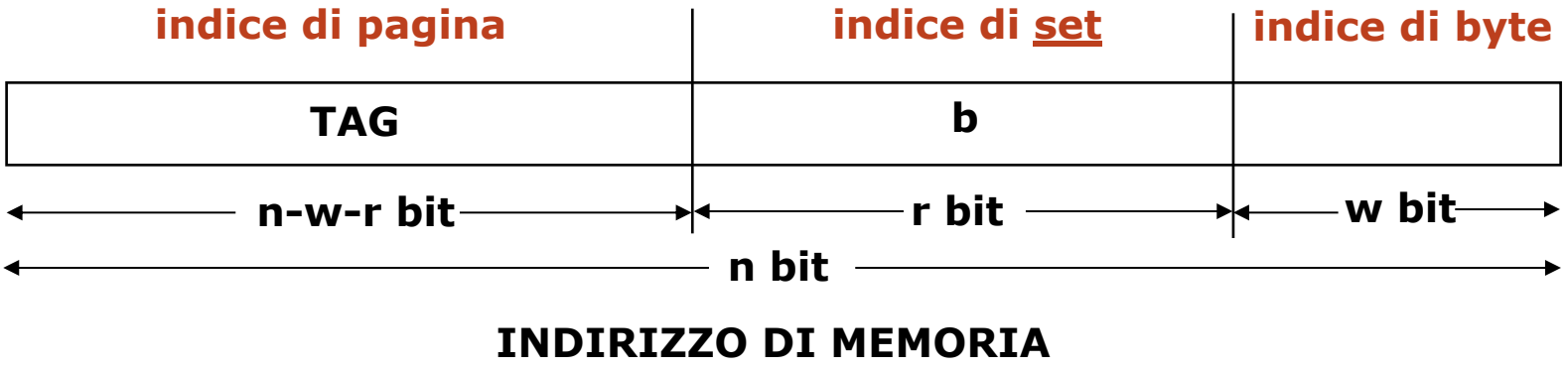
Indirizzi nella cache set-associativa

Gli indirizzi di memoria sono divisi in campi:

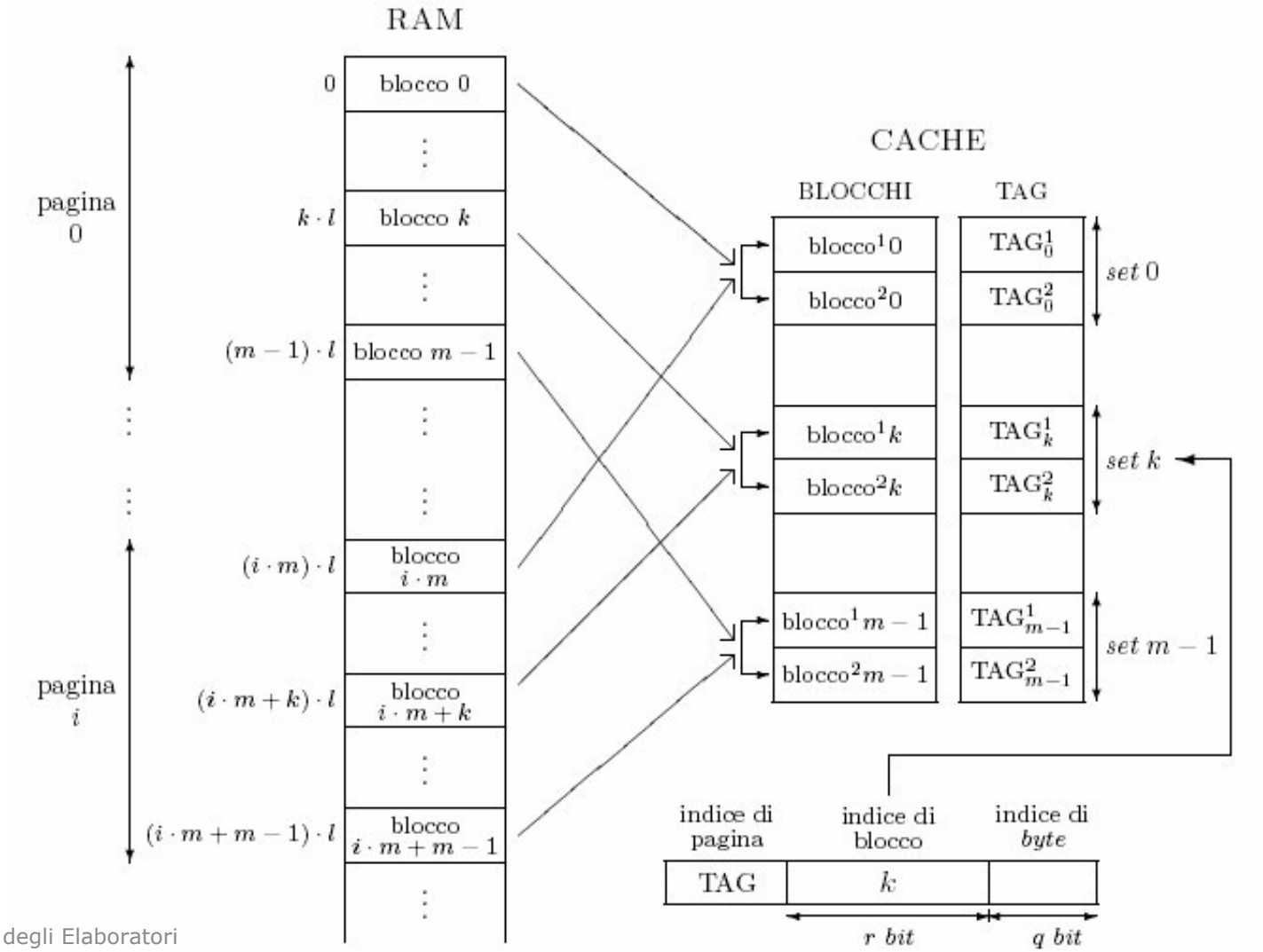
- i w bit meno significativi: indice del byte nel blocco;
- i rimanenti s = n-w bit comuni a tutti i byte del blocco;

Questi s bit sono ora così ripartiti:

- r bit meno significativi: indice b della linea di cache
- n-w-r bit: TAG = indice della pagina nella RAM.



Schema di cache set-associativa a 2 vie

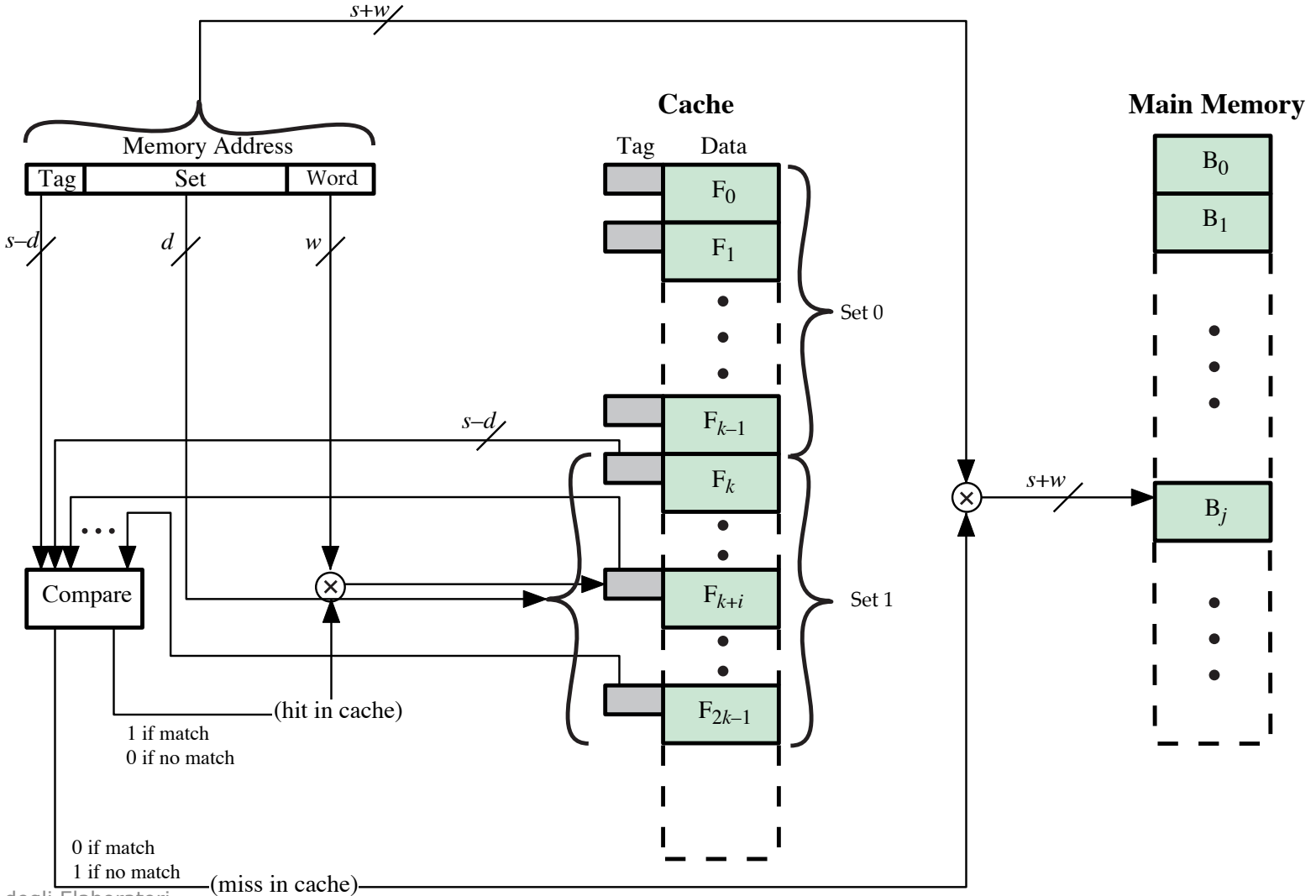


Funzionamento set associativa

Quando il processore genera un indirizzo di memoria:

- Gli r bit che forniscono l'indice b del blocco, individuano il set nella cache in cui si trova il blocco, se è presente;
- Gli n-w-r bit più significativi dell'indirizzo (TAG) vengono confrontati (in parallelo) con il contenuto della CAM:
  - se trovato (HIT), l'accesso si risolve nella cache,
  - altrimenti (MISS):
    - viene effettuato l'accesso alla RAM e, contestualmente,
    - il blocco di RAM caricato viene trasferito in una delle n linee del set (scelta con modalità FIFO o LRU) e il suo TAG nel corrispondente elemento della CAM.

Funzionamento set associativa



Confronti con le altre organizzazioni

- Set-associativa vs mappatura diretta:
  - è più efficiente: hit rate maggiore; il blocco da rimpiazzare non è predeterminato, ma può essere scelto, tra quelli del set, con un criterio che renda meno probabili i miss ;
  - è più costosa: richiede una CAM per ciascun set ;
- Set-associativa vs completamente associativa:
  - è meno efficiente: hit rate inferiore; la scelta del blocco da rimpiazzare non è fatta tra tutti i blocchi della cache, ma è limitata a quelli di un solo set ;
  - è meno costosa: tante piccole CAM, una per ciascun set, costano meno di una CAM unica (grande).
- È un buon compromesso: è la soluzione più diffusa.

Algoritmi di rimpiazzo

- Quando, in seguito a un miss, è necessario ricopiare nella cache un nuovo blocco, normalmente è necessario sovrascriverne un blocco già presente.
- La politica con cui si sceglie il blocco da sovrascrivere è molto importante per l'efficienza della cache.
- Due algoritmi di rimpiazzo comuni sono:
  - **FIFO**: si sovrascrive la linea contenente il blocco che sta nella cache da più tempo;
  - **LRU** (Least Recently Used): si sovrascrive la linea che da più tempo non subisce accessi; si possono usare uno o più dei bit di controllo associati alla linea, per tener traccia del tempo trascorso dall'ultimo accesso.

Scrittura di un blocco

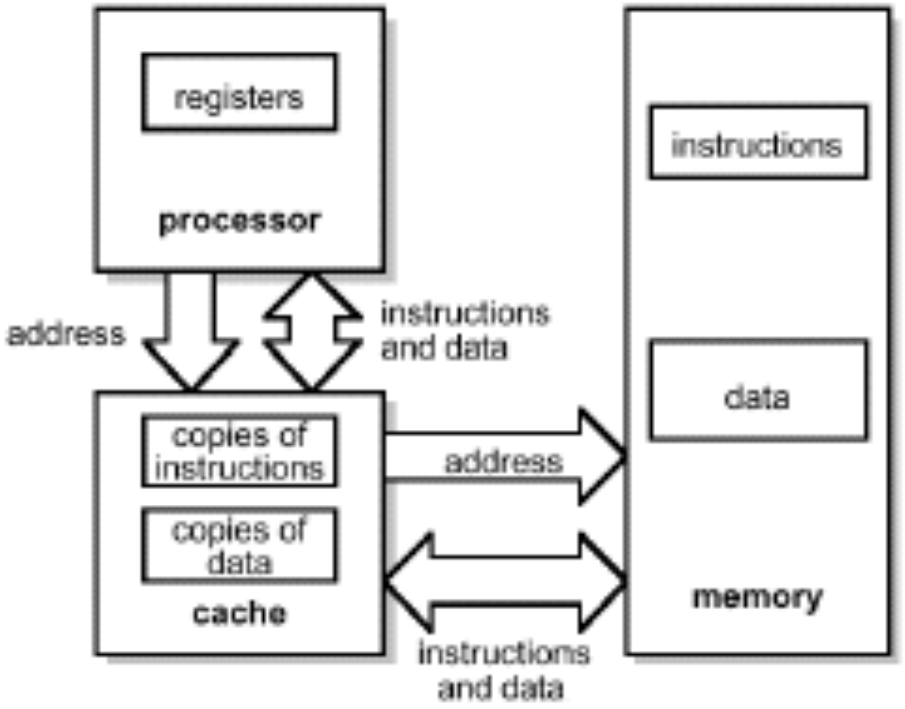
- Se il blocco selezionato per essere sovrascritto ha subito accessi per scrittura mentre si trovava nella cache (un bit di controllo, **dirty bit**, lo segnala), allora la sua copia in RAM è obsoleta:
  - prima di sovrascriverlo, il contenuto del blocco va ricopiato in RAM (**copy-back**).
- Un tecnica alternativa, usata per tenere aggiornate le copie in RAM dei blocchi, consiste nell'effettuare tutte le operazioni di scrittura sia nella cache sia anche in RAM (**write-through**).
- Con entrambe le tecniche, la scrittura in RAM avviene di solito tramite un write buffer, ad accesso veloce, in modo che il processore non debba attendere che il trasferimento in RAM sia completato.

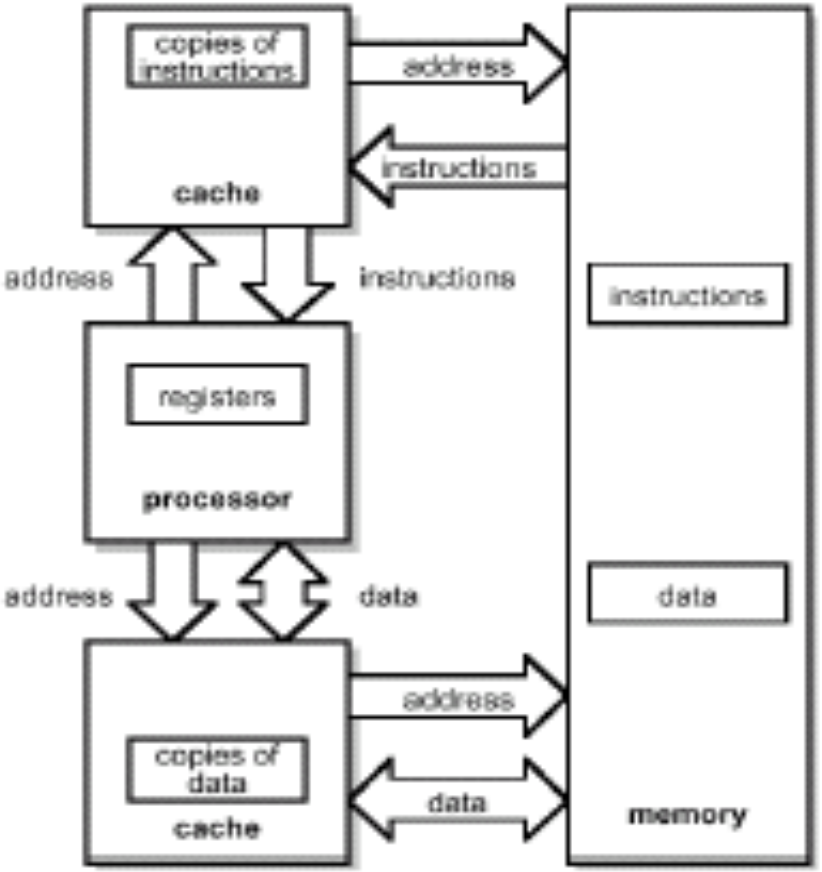
I bit di controllo della cache

- Già visti:
  - dirty bit (copy-back),
  - bit indicanti il tempo trascorso dall'ultimo accesso (LRU).
- Un altro bit di controllo utile:
  - bit di validità: indica se il blocco è una copia valida del corrispondente blocco in RAM (può essere invalidato in seguito a modifiche subite da quest'ultimo per trasferimenti DMA o in sistemi multiprocessore).

- Dimensione cache = numero line * (dimensione blocco + dimensione TAG + numero bit controllo)

Cache unica o distinta per istruzioni e dati

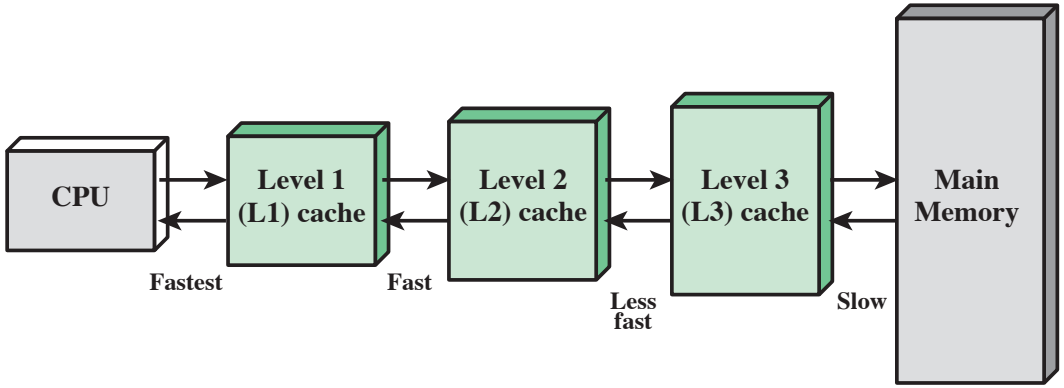




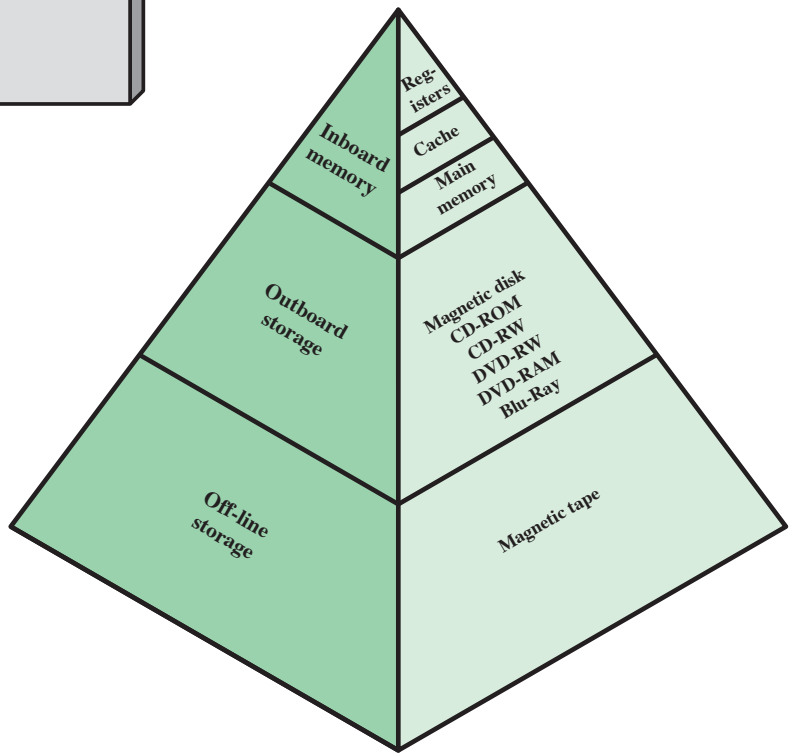
Gerarchia multilivello

- Possono essere presenti più livelli di cache:
  - una cache di 1° livello, quasi sempre integrata nello stesso chip del processore, ad accesso rapidissimo;
  - una cache di 2° livello, talvolta esterna al chip del processore, ad accesso rapido;
  - a volte anche una cache di 3° livello.
- Il livello più vicino al processore conterrà i dati che serviranno al processore nell'immediato futuro;
- I dati che serviranno più avanti vengono contenuti nei livelli più lontani che sono più lenti ma più capienti.

Gerarchia multilivello



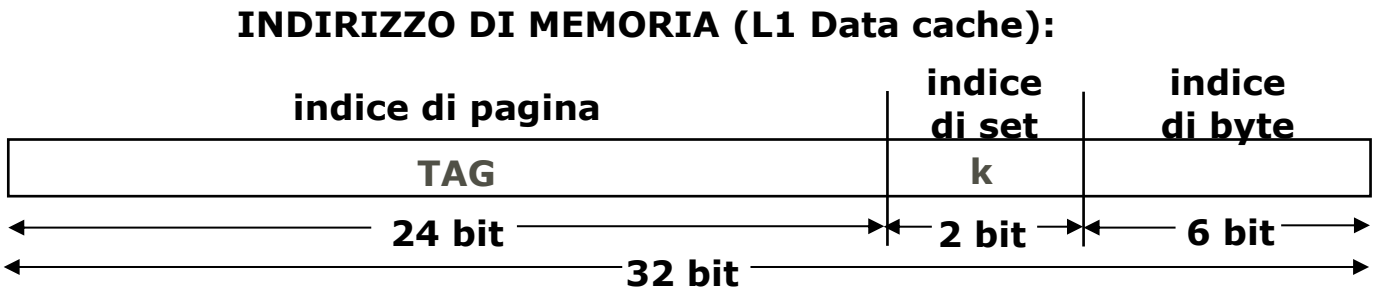
(b) Three-level cache organization



Cache nel Broadcom BCM2837B0, Cortex-A53 (Raspberry Pi A+)

Caches:

- instruction L1 cache: dimensione 48 KB, blocco 64 byte, 2-way
- data L1 cache: dimensione 32KB, blocco 64 byte, 4-way
- L2 cache: dimensione 1MB, blocco 64 byte, 16-way
- algoritmo di rimpiazzo: pseudo-random,



Esercizio

- Si supponga che una memoria cache per i dati adotti uno schema a mappatura diretta con 16 byte per blocco e 32 blocchi complessivi. Allora, per una memoria indirizzata a byte, il byte di indirizzo 0xB425, se presente nella cache, si trova nel blocco di indice I (0..31) e nella posizione relativa (all'interno del blocco) B (0..15). (si indichino i valori di I e B come numeri interi con notazione in base dieci.) Il suo tag è T.

- I = ...
- B = ...
- T = ...

Esercizio (soluzione)

- Si supponga che una memoria cache per i dati adotti uno schema a mappatura diretta con 16 byte per blocco e 32 blocchi complessivi. Allora, per una memoria indirizzata a byte, il byte di indirizzo 0xB425, se presente nella cache, si trova nel blocco di indice I (0..31) e nella posizione relativa (all'interno del blocco) B (0..15). Il suo tag è T.

- I = 2
- B = 5
- T = 0x5A

Esercizio

- Una memoria *cache* ha tempo di accesso pari a 62.5ns; con un *hit rate* del 80%, il tempo medio di accesso alla memoria è di 90ns. Allora la memoria convenzionale ha un tempo di accesso t pari a:

- t=**.....**

Esercizio (soluzione)

- Una memoria *cache* ha tempo di accesso pari a 62.5ns; con un *hit rate* del 80%, il tempo medio di accesso alla memoria è di 90ns. Allora la memoria convenzionale ha un tempo di accesso t pari a:

- T=200 ns

Esercizio (II compitino 2017, esercizio 6)

- Un sistema ARM è dotato di una memoria cache di tipo set-associativo costituita da NS set da 4 vie e ciascuna via è costituita da un blocco da 32 byte. Associati a ciascun blocco, la cache comprende anche 2 bit di controllo e 25 bit, destinati a contenere il TAG del blocco. Ciascun set è pertanto dotato di una propria memoria associativa contenente i TAG dei blocchi presenti nel set. Tenendo presente che il processore genera indirizzi da 32 bit, si indichino: il numero NS di set della cache e il numero B di bit di cui è costituita ciascuna delle NS memorie associative della cache.
- NS = ……. B = ……. bit

Esercizio (soluzione)

- Un sistema ARM è dotato di una memoria cache di tipo set-associativo costituita da NS set da 4 vie e ciascuna via è costituita da un blocco da 32 byte. Associati a ciascun blocco, la cache comprende anche 2 bit di controllo e 25 bit, destinati a contenere il TAG del blocco. Ciascun set è pertanto dotato di una propria memoria associativa contenente i TAG dei blocchi presenti nel set. Tenendo presente che il processore genera indirizzi da 32 bit, si indichino: il numero NS di set della cache e il numero B di bit di cui è costituita ciascuna delle NS memorie associative della cache.
- NS = 4 B = 100 bit