

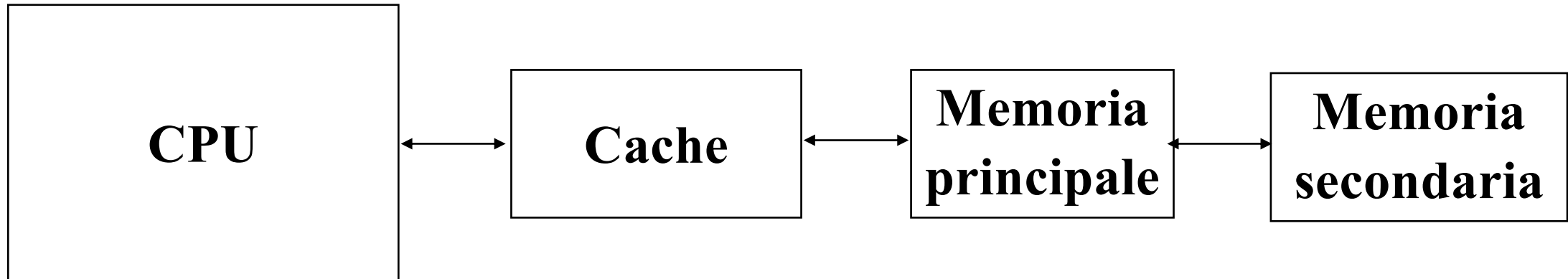
# Le memorie cache

Politecnico di Torino  
Dip. di Automatica e Informatica



# Introduzione

**Le memorie cache sono memorie di piccole dimensioni ma con elevata velocità interposte tra il processore e la memoria principale.**



# Località dei riferimenti

La presenza di una cache può migliorare le prestazioni di un sistema per via della *località dei riferimenti* osservabile nella maggioranza dei programmi.

Si esprime in due forme:

- *località temporale*: se all'istante  $t$  il programma fa accesso ad una cella di memoria, è molto probabile che il programma faccia nuovamente accesso alla stessa cella entro l'istante  $t + \Delta$
- *località spaziale*: se all'istante  $t$  il programma fa accesso alla cella di memoria di indirizzo  $X$ , è molto probabile che entro l'istante  $t + \Delta$  il programma faccia accesso anche alla cella di indirizzo  $X \pm e$ .

# Principio di funzionamento

**Se all'istante  $t$  (primo accesso ad un blocco di memoria da parte del programma) viene caricato nella cache l'intero blocco, ci sono alte probabilità che per un certo tempo  $\Delta$  il programma trovi nella cache tutte le parole di memoria cui deve fare accesso.**

# Prestazioni

Si definiscano le seguenti grandezze:

- $h$ : hit ratio della cache
- $C$ : tempo di accesso alla cache
- $M$ : tempo di accesso in memoria quando il dato non è in cache.

Il tempo medio di accesso in memoria per la CPU sarà

$$t_{medio} = hC + (1-h)M$$

Valori normali per  $h$  sono dell'ordine di 0,9.

# Struttura di una cache

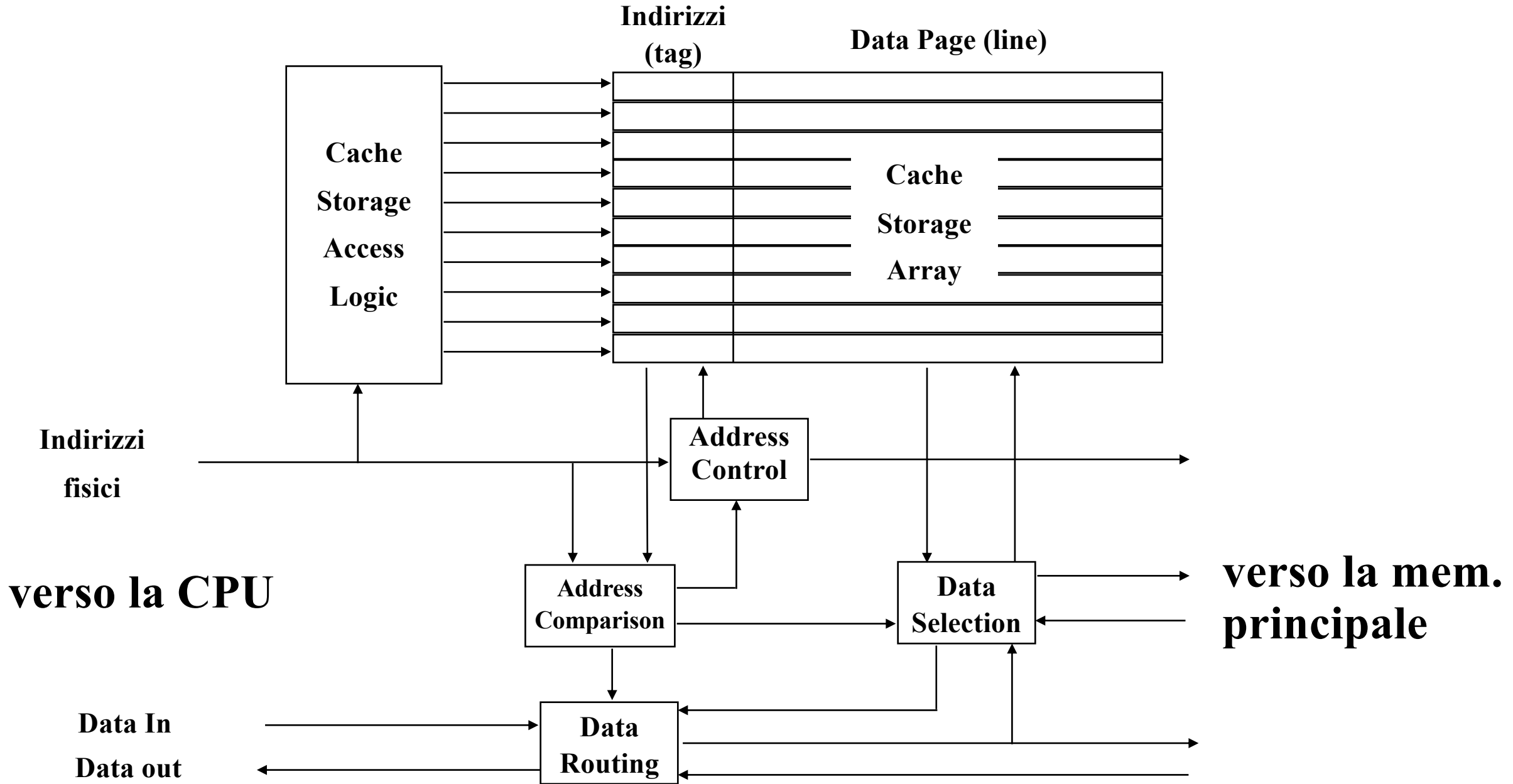
Una cache contiene al suo interno un certo numero di *linee*.

Una linea contiene un blocco di memoria. Ad ogni linea è associato un campo *tag*, che indica il blocco di memoria presente nella linea in quel momento.

La cache contiene inoltre la logica per

- intercettare gli indirizzi prodotti dal processore
- controllare al proprio interno l'eventuale presenza del blocco a cui il processore vuole fare accesso
- eventualmente caricare da memoria il blocco.

# Struttura



# Funzionamento della cache

**La cache si interpone tra processore e memoria principale.**

**Ogni volta che il processore esegue un accesso alla memoria la cache**

- **intercetta l'indirizzo**
- **verifica se il blocco cui appartiene la parola è presente nella cache, controllando il valore dei tag**
- **se sì: estrae la parola dal blocco e la fornisce alla CPU al posto (e prima) della memoria principale (*hit*)**
- **se no: provvede a caricare nella cache l'intero blocco di cui la parola fa parte (*miss*).**



# Prestazioni

**In caso di hit, la cache riduce i tempi di accesso di un fattore dipendente dal rapporto tra i tempi di accesso della cache e della memoria principale.**

**In caso di miss, la cache risponde in due possibili modi:**

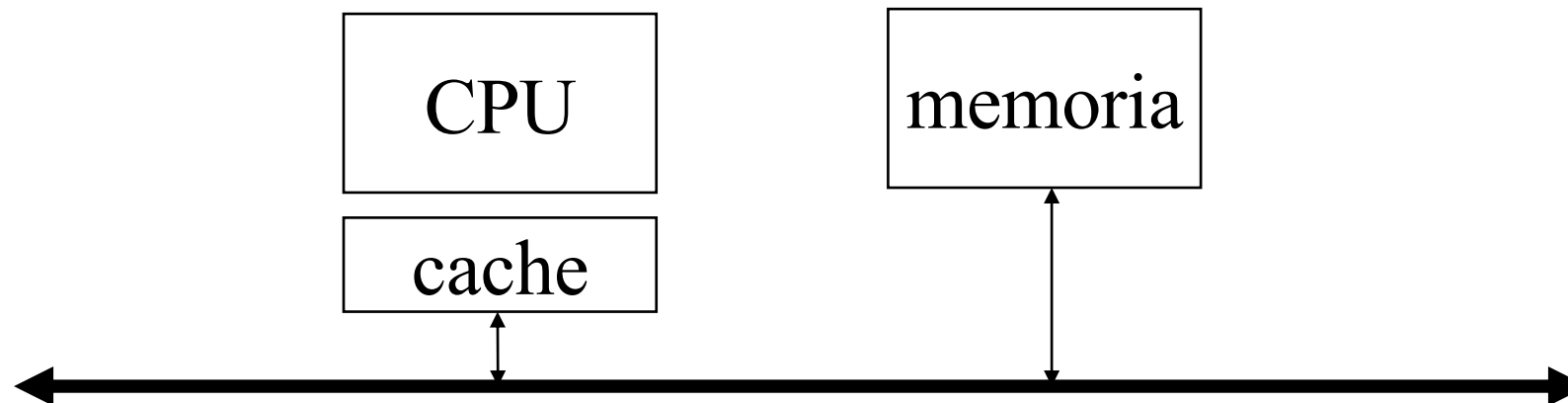
- **Accede alla memoria e carica l'intero blocco mancante; poi fornisce la parola richiesta. Il tempo di accesso è quindi superiore a quello di accesso alla memoria senza cache.**
- **Accede alla memoria e fornisce subito la parola richiesta; poi provvede al caricamento del blocco (*load-through* o *early restart*). Questa tecnica richiede un maggior costo dell'hardware della cache, ma il miss ha un impatto più limitato sulle prestazioni della cache.**

# Posizione della cache

**La cache è normalmente collocata tra la CPU e il bus, anziché tra la memoria principale e il bus.**

**I vantaggi che si ottengono in questo modo sono:**

- **si alleggerisce il carico del bus**
- **la soluzione è compatibile con un'architettura multiprocessore.**



# **Instruction Cache e Data Cache**

**In alcuni casi vi sono cache separate per dati e istruzioni; in altri casi ve ne è solo una per istruzioni e dati.**

**La cache per le istruzioni è in genere più semplice da gestire di quella per i dati, in quanto le istruzioni non possono essere modificate.**

# Architettura Harvard

**Se esistono due cache, l'architettura del sistema ricade nello schema noto come *architettura Harvard*, caratterizzato dal fatto che esistono due memorie separate per dati e codice.**

**L'architettura Harvard si contrappone a quella di von Neumann.**

# Parametri caratteristici

**Sono:**

- **Dimensione della cache**
- **Funzione di traduzione (*Mapping*)**
- **Algoritmo di rimpiazzamento**
- **Metodo di aggiornamento della memoria principale**
- **Dimensione dei blocchi.**

# **Dimensione della cache**

**La scelta della dimensione ottimale è influenzata da:**

- **costo**
- **prestazioni: al crescere delle dimensioni, le cache diventano più lente.**

**Dimensioni frequenti vanno da qualche KB a qualche MB.**

# **Funzione di traduzione**

## ***(mapping)***

**Definisce in quale linea della cache è eventualmente posizionato un certo blocco di memoria.**

**Si deve garantire (ad un costo accettabile) che la cache possa rapidamente verificare se contiene il dato corrispondente ad un certo indirizzo.**

**Possibili soluzioni sono:**

- memoria associativa (troppo costosa)**
- scansione sequenziale (troppo lenta)**
- meccanismo di mapping degli indirizzi (soluzione normalmente usata).**

# Meccanismi di mapping

I principali sono tre:

- *direct mapping*
- *associative mapping* (raramente adottato)
- *set associative mapping*.



# Direct Mapping

Ogni blocco  $i$  della memoria principale è messo in corrispondenza fissa con una linea  $k$  della cache.

La funzione di trasformazione è

$$k = i \bmod N$$

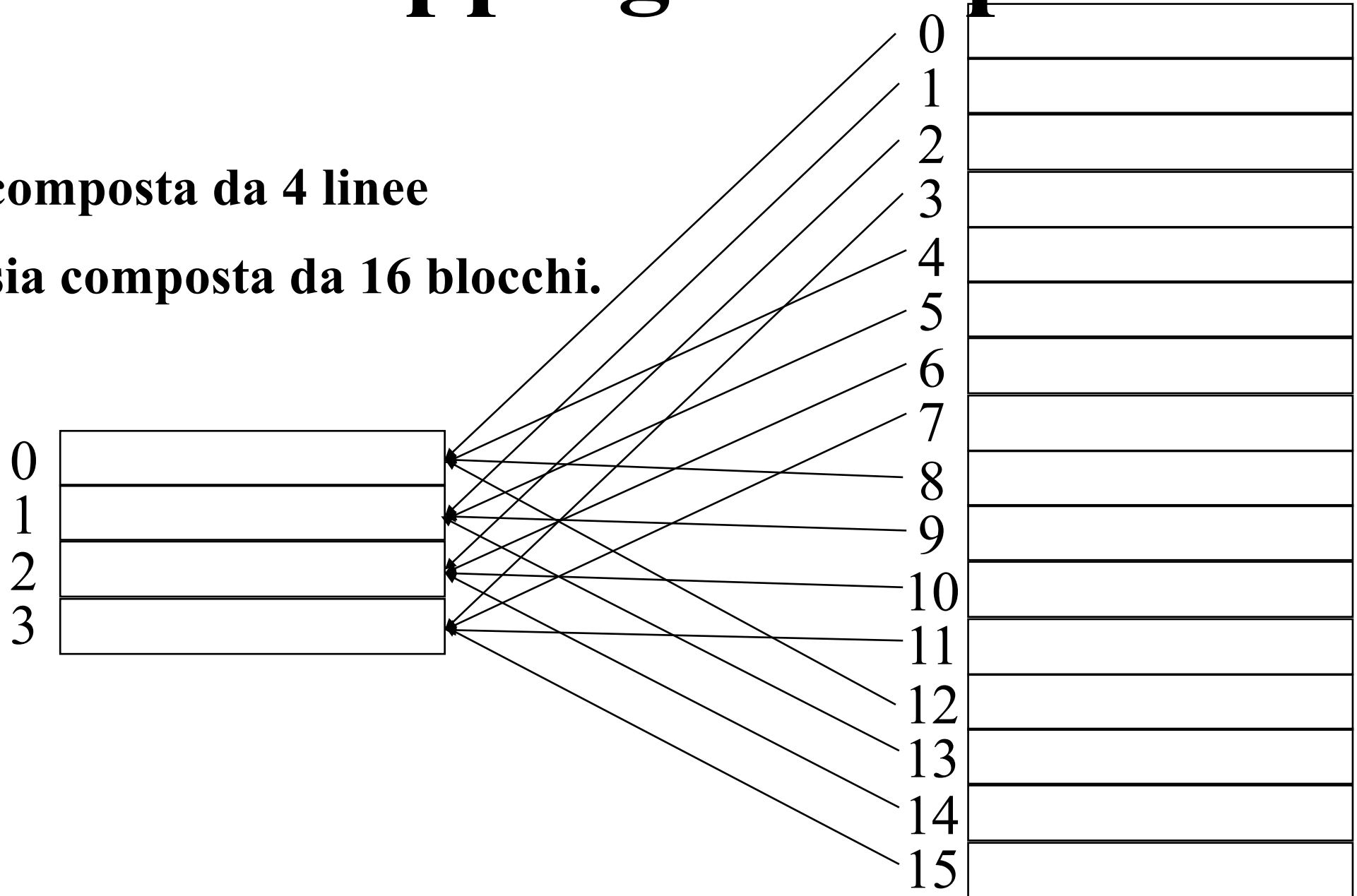
dove  $N$  è il numero di linee della cache.

Il calcolo di  $k$  può essere eseguito in modo semplice prendendo i bit meno significativi del numero identificativo del blocco ( $i$ ).

# Direct Mapping: esempio

Si assuma che

- la cache sia composta da 4 linee
- la memoria sia composta da 16 blocchi.



# Direct Mapping

## **Vantaggi:**

- **la funzione è facilmente implementabile in hardware (i bit meno significativi dell'identificativo del blocco individuano la linea di cache)**

## **Svantaggi:**

- **se il programma fa accesso frequentemente a 2 blocchi corrispondenti alla stessa linea della cache, ad ogni accesso si verifica un miss.**

# Direct Mapping: struttura dell'indirizzo

Ciascun indirizzo può essere scomposto in tre parti:



Identifica il blocco di memoria.  
È il campo che viene confrontato  
con il tag.

Identifica la parola  
all'interno del blocco

Attraverso la funzione di mapping

# Associative Mapping

**Ogni blocco della memoria principale può essere memorizzato in un qualsiasi blocco della cache.**

**Vantaggi:**

- **massima flessibilità nella scelta del blocco di cache da usare**

**Svantaggi:**

- **complessità dell'hardware di ricerca (di solito si adotta una memoria associativa).**

# Struttura dell'indirizzo

**Nel caso di associative mapping, ciascun indirizzo viene scomposto dalla cache in due parti:**



Identifica il blocco di memoria.  
È il campo sul quale  
la cache esegue il controllo.

Identifica la parola  
all'interno del blocco

# Set Associative Mapping

## Caratteristiche:

- le linee della cache sono suddivise in  $S$  insiemi, ciascuno composto da  $W$  linee
- un blocco  $i$  è associato all'insieme  $k$  se  $k=i \bmod S$
- il blocco  $i$  può essere messo in una qualunque delle  $W$  linee dell'insieme  $k$ .

Si parla quindi di cache set associative a  $W$  vie ( $W$ -ways).

Valori comuni di  $W$  sono 2 e 4.

Se  $S = N$  ( $N$  è la dimensione della cache) si ha il *direct mapping*; se  $S=1$  si ha l'*associative mapping*.

# Struttura dell'indirizzo

**Nel caso di set associative mapping, ciascun indirizzo viene scomposto dalla cache in tre parti:**



Identifica il blocco di memoria.  
È il campo che viene confrontato  
con il tag.

Identifica la parola  
all'interno del blocco



# Algoritmo di rimpiazzamento

Definisce quale linea di cache deve venire utilizzata per memorizzare un blocco di memoria, tra quelle associate al blocco (nel caso di associative o set associative mapping).

Viene scelto tra:

- *LRU* (Least Recently Used): il più utilizzato
- *FIFO* (First-In First-Out): il più economico
- *LFU* (Least Frequently Used): teoricamente il più efficace
- *random*: semplice ed efficiente.

# Aggiornamento della memoria principale

La CPU ha normalmente un canale di connessione diretto con la memoria principale; questo permette 2 possibili meccanismi di aggiornamento della memoria principale:

- *write-back*
- *write-through.*

# Write-Back

Per ogni blocco nella cache viene tenuto aggiornato un flag (*dirty bit*), che ricorda se il blocco è stato modificato o meno da quando è stato caricato nella cache.

Quando un blocco viene eliminato dalla cache e il dirty bit è settato, il blocco viene copiato dalla cache nella memoria principale.

**Svantaggi:**

- la sostituzione è più lenta, perché a volte richiede la copiatura in memoria del blocco sostituito
- nei sistemi multiprocessore si può avere inconsistenza tra le cache di diversi processori
- il ripristino dei dati della memoria dopo eventuali *system failure* può non essere possibile.

# Write-through

**Ogni volta che la CPU esegue un'operazione di scrittura, la esegue sia sul dato nella cache che in quello nella memoria principale.**

**La perdita di efficienza che ne deriva è limitata dal fatto che le operazioni di scrittura sono di solito molto meno numerose di quelle di lettura.**

# Dimensioni dei blocchi

**Al crescere delle dimensioni del blocco (e a parità di dimensioni complessive della cache) si verificano 2 fenomeni:**

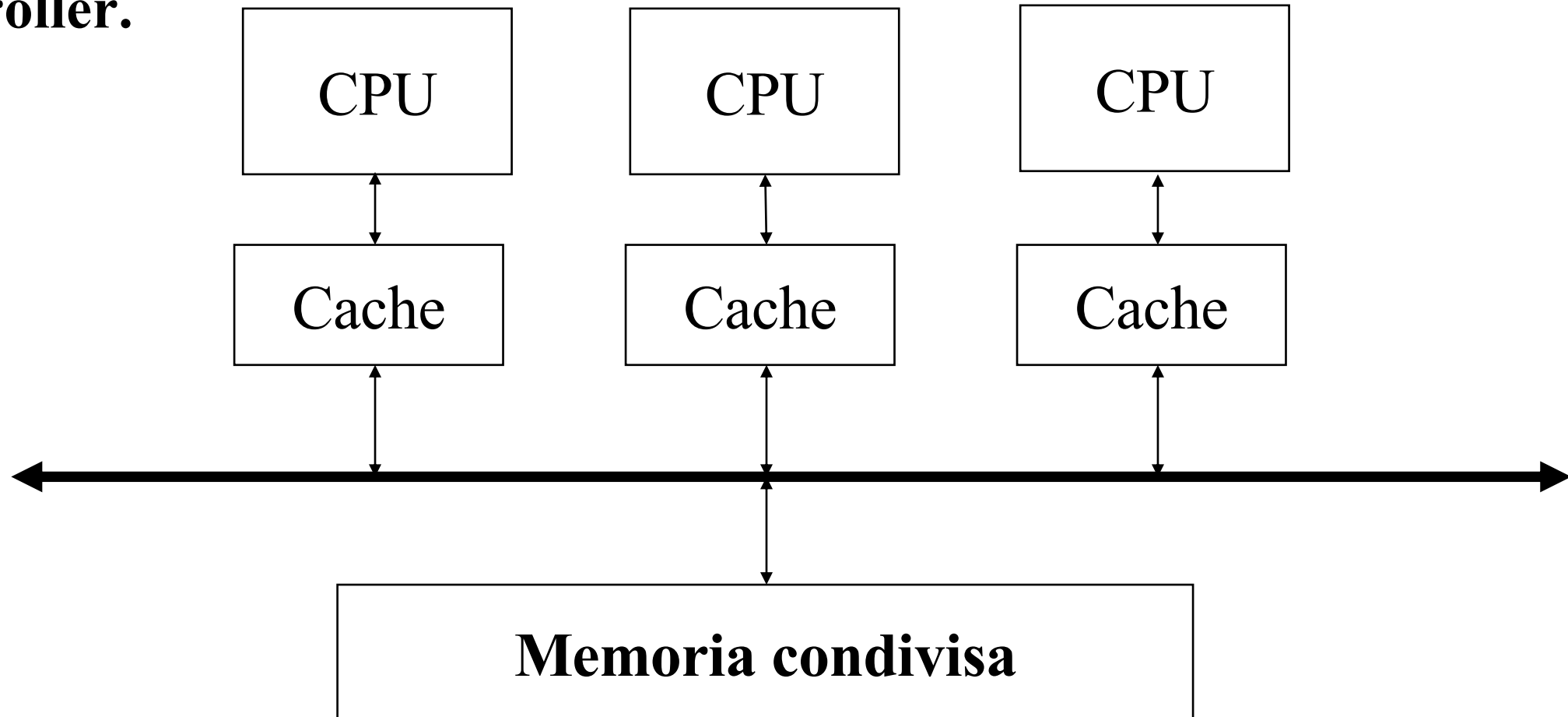
- dapprima la hit ratio cresce (grazie alla località dei riferimenti)**
- poi comincia a decrescere (perché diminuisce il numero di blocchi in cache).**

**Valori frequenti sono da 4 a 32 byte.**

# Coerenza della cache

**È un problema nei sistemi a multiprocessore con memoria condivisa, nei quali ogni processore ha una sua cache.**

**Problemi analoghi si possono avere se il sistema utilizza un DMA controller.**



# Bit di validità

Per ottenere la coerenza delle cache si introduce per ogni linea di cache un *bit di validità*.

Se è disattivato, significa che il blocco presente in quella linea ha un valore diverso dal corrispondente blocco nella memoria principale. In tal caso ogni accesso al blocco comporta un miss.

# Soluzioni

Nei sistemi multiprocessore si usa di solito il meccanismo di write-through.

Inoltre, per garantire la coerenza delle cache si possono usare le seguenti soluzioni:

- *Bus Watching con Write-through*: il controllore di ciascuna cache rileva le operazioni di Write-through sul bus, e invalida (attraverso il bit di validità) le linee corrispondenti nella propria cache;
- *Non-cacheable Memory*: la memoria condivisa non può essere trasferita nelle cache.



# **Cache di primo, secondo e terzo livello**

**Può essere conveniente avere più livelli di cache:**

- **una cache di primo livello (L1), più piccola e veloce**
- **una cache di secondo livello (L2), più lenta ma più grande**
- **una cache di terzo livello (L3), ancora più lenta e grande.**

# **Cache di primo, secondo e terzo livello - funzionamento**

**Ogni volta che il processore esegue un accesso in memoria**

- **Si controlla se la parola sta in L1**
  - **Se sì, si accede a L1**
  - **Se no, si controlla se la parola sta in L2**
    - + **Se sì, si accede a L2 ed eventualmente si aggiorna L1**
    - + **Se no, si controlla se la parola sta in L3**
      - \* **Se sì, si accede a L3 ed eventualmente si aggiorna L2**
      - \* **Se no, si accede alla memoria principale ed eventualmente si aggiorna L3.**

# **Cache di primo e secondo livello - prestazioni**

**Aggiungendo una cache di secondo livello**

- **in caso di hit nella C1 il tempo di accesso dipende solo dalla velocità della C1**
- **in caso di miss nella C1 il tempo di accesso**
  - **viene comunque ridotto se la parola si trova nella C2**
  - **resta pari al tempo di accesso alla memoria principale in caso contrario.**

**La scelta delle memorie che fanno parte dei vari livelli di cache (in termini di dimensione, tipo e meccanismo di funzionamento) determina se il sistema produce vantaggi o meno.**

# **Esempio: FreeScale PowerPC 603**

**Il PowerPC 603 fu il primo processore ad implementare l'architettura a 32 bit del PowerPC, con frequenze tra 80 e 100 MHz.**

**Per qualche anno fu usato nei computer Apple, ma la sua applicazione principale è stata in campo embedded, dove sono ancora molto utilizzate le successive versioni da esso derivate.**

**Esiste un solo livello di cache, separata per dati e istruzioni: ogni cache ha dimensione pari a 16 kbyte e utilizza il mapping set associative a 4 vie con algoritmo di sostituzione LRU. Ogni linea contiene 32 byte, e ci sono quindi 128 insiemi.**

**È possibile aggiungere esternamente una cache di secondo livello di 1 Mbyte.**

# Esempio: AMD Zambezi

**Zambezi è una CPU di fascia alta della famiglia AMD Fusion: include fino a 8 core, ciascuno equipaggiato con una sua cache di primo livello (L1).**

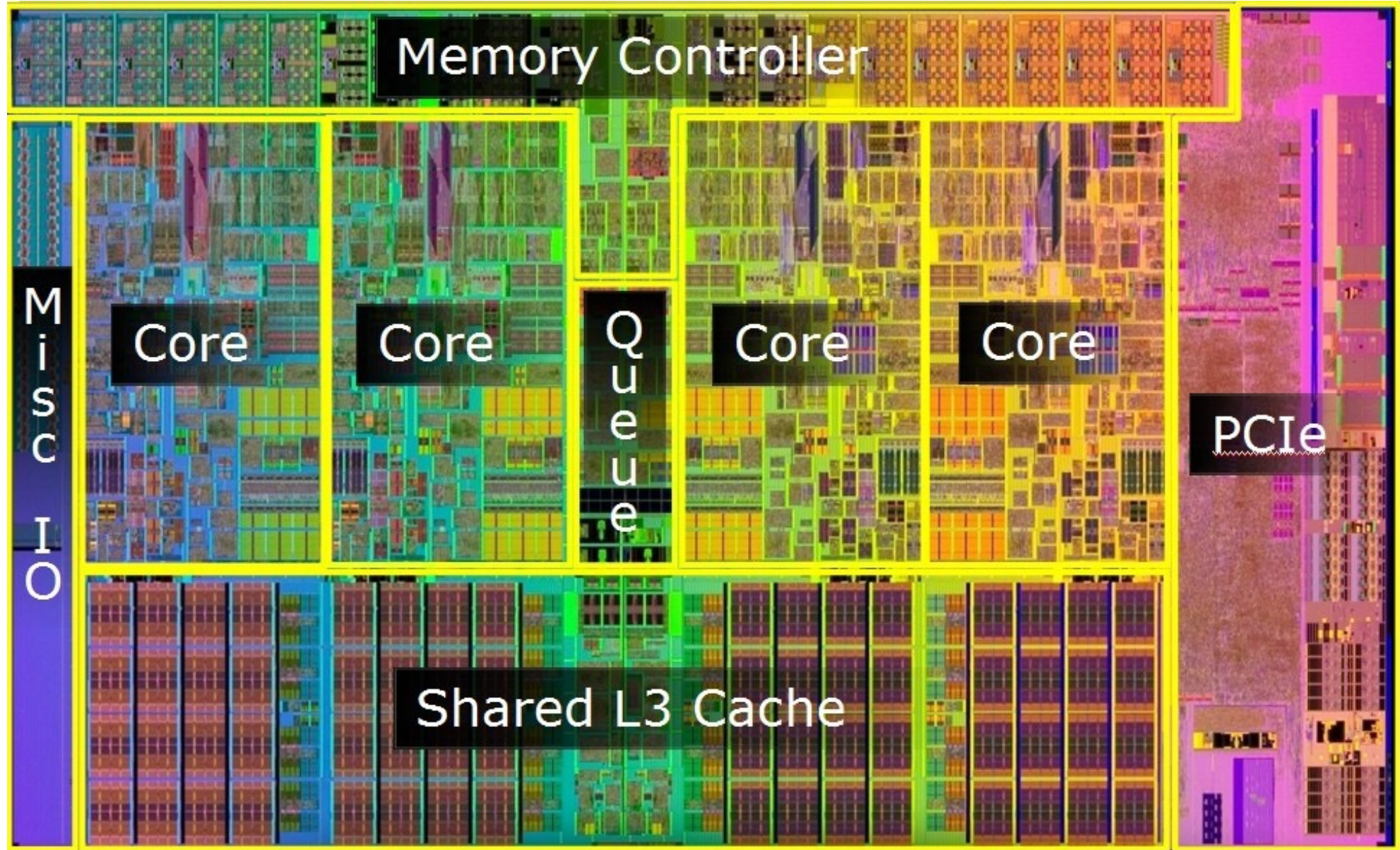
**Ogni coppia di core utilizza una cache di secondo livello (L2) di 2 o 4 Mbyte.**

**I core dello stesso dispositivo condividono la cache di terzo livello (L3) di 8 Mbyte.**





# Esempio: Intel Lynnfield



# **Esempio di calcolo: specifiche**

**Si consideri una cache con le seguenti caratteristiche:**

- dimensione pari a 64KByte (solo dati)**
- direct mapping**
- blocchi di 4 byte**
- indirizzi su 32 bit.**

**Si vuole determinare la struttura della cache (numero delle linee, dimensione del campo tag).**

# **Esempio:**

## **struttura della cache**

**Poichè ogni blocco è composto da 4 byte, la memoria è composta da  $2^{30}$  blocchi.**

**Le linee sono in numero pari a  $64\text{KByte}/4=16\text{K}=2^{14}$ .**

**Il campo tag deve poter identificare il blocco presente nella linea.**

**In prima battuta il campo tag dovrebbe quindi occupare 30 bit. Poiché però in una generica linea di cache sono memorizzati solo i blocchi il cui indice ha i 14 bit bassi coincidenti con l'indice della linea, il campo tag ha dimensione pari a 16 bit.**

**La cache ha quindi le seguenti dimensioni complessive:**

$$40 \qquad 2^{14} \times (32 + 16) = 2^{14} \times 48 = 768\text{Kbit} = 96\text{KByte}$$