

Calcolatori Elettronici A

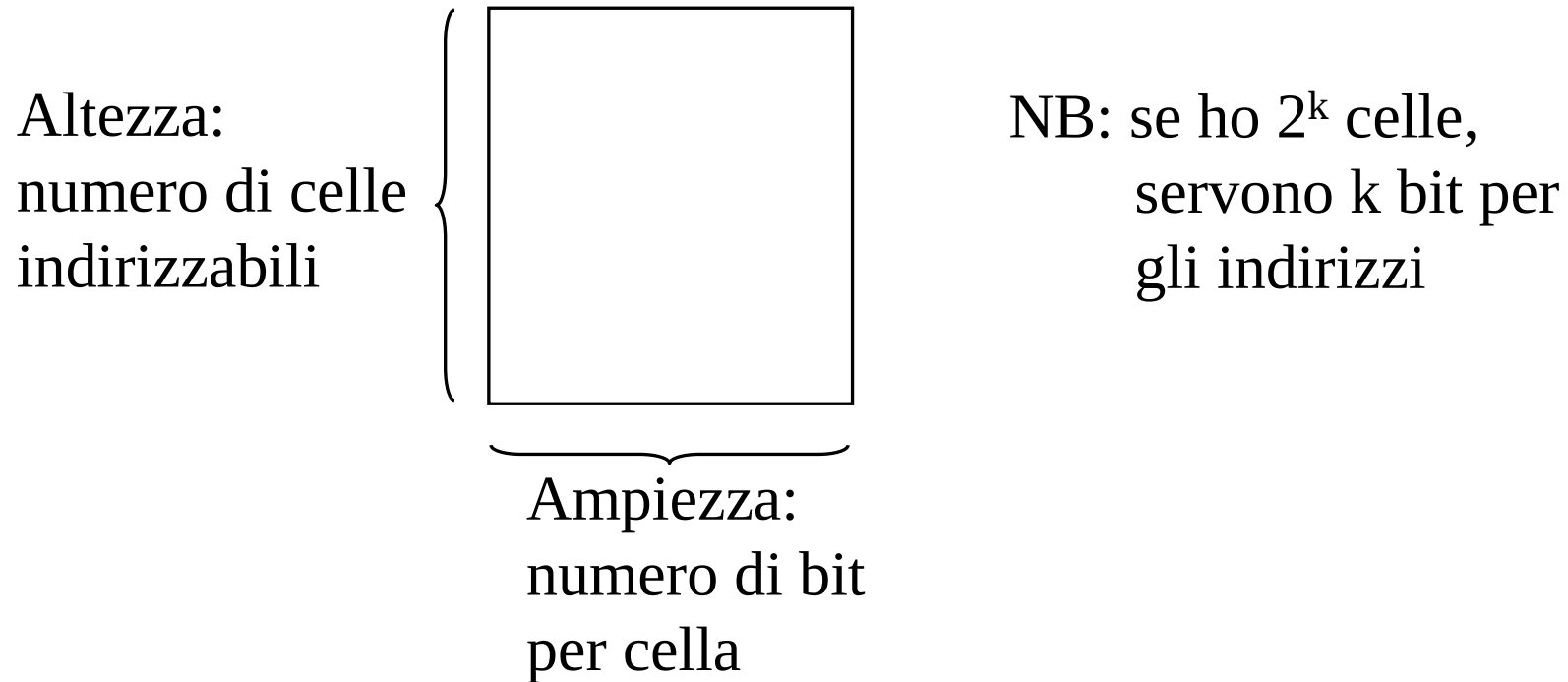
a.a. 2008/2009

Gerarchia di memorie: memorie cache

Massimiliano Giacomini

Tipologie e caratteristiche delle memorie

(soprattutto dal punto di vista circuitale e fisico)



⇒ Dimensione (in bit): altezza * ampiezza

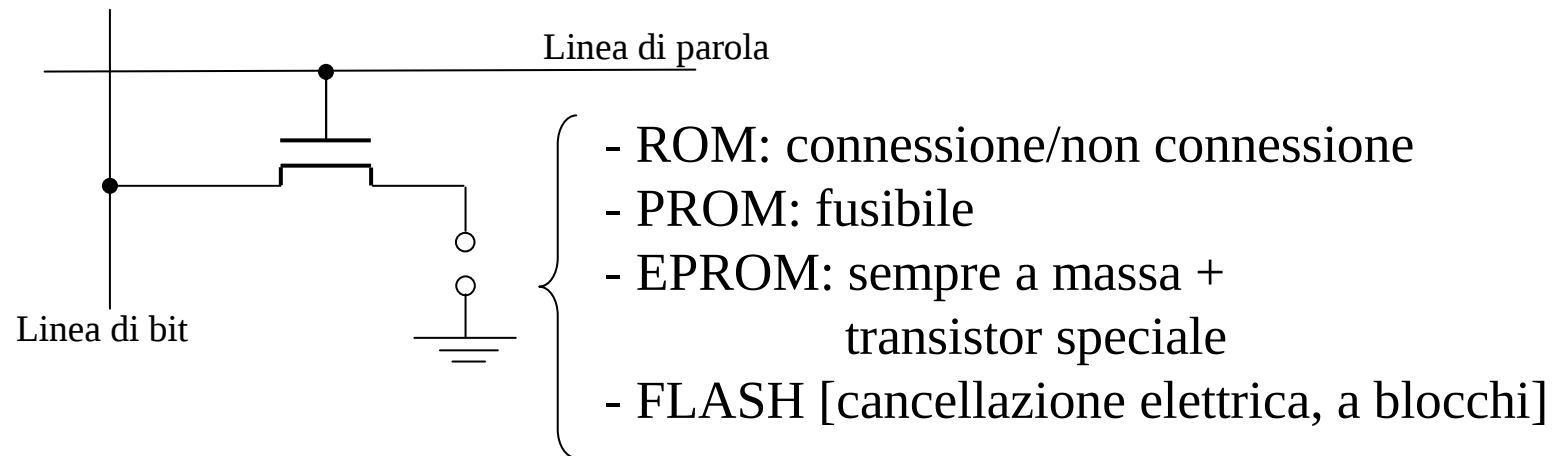
Esempio 32K x 8: altezza pari a 32768 locazioni di memoria ciascuna di 8 bit $\Rightarrow$ 15 linee di indirizzo e 8 linee di dato

Tipologie di memorie

- *Memorie a sola lettura*

(ROM, PROM, EPROM, EEPROM-Flash)

Cella:



Tipologie di memorie

- **Memorie volatili** (RAM): SRAM vs. DRAM

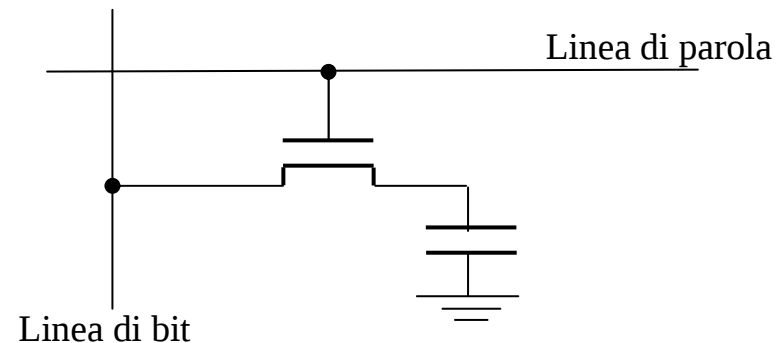
SRAM (RAM Statica)

- Cella di memoria costituita da latch (4-6 transistor)

- minor densità
- maggior costo
- maggior velocità

DRAM (RAM Dinamica)

- Cella di memoria costituita da un singolo transistor



- maggior densità
- minor costo
- minor velocità – 5/6 volte (e necessità di refresh)

Il quadro completo

- Static RAM (SRAM)
 - 0.5ns – 2.5ns, \$2000 – \$5000 per GB
- Dynamic RAM (DRAM)
 - 50ns – 70ns, \$20 – \$75 per GB
- Magnetic disk
 - 5ms – 20ms, \$0.20 – \$2 per GB
- **LA MEMORIA IDEALE**
 - Tempo di accesso della SRAM
in particolare nello stesso chip del processore $\Rightarrow$ piccola!
 - Capacità e costo/GB dei dischi

NB: con un processore che lavora a 1Ghz, $T_{\text{clock}} = 1\text{ns}$!!!

Il principio di località

Località temporale

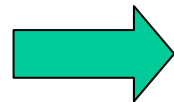
Istruzioni e dati cui si è fatto riferimento di recente tendono ad essere riutilizzati

- es: istruzioni all'interno di un ciclo (che possono anche riferire dati)

Località spaziale

Istruzioni e dati riferiti $\Rightarrow$ tendenza a riferire quelli vicini

- es: sequenze di istruzioni, array



IDEA: usare una *gerarchia di memorie*!

Gerarchia di memorie (1)

Tecnica per far apparire la memoria grande ma veloce : portare vicino alla CPU blocchi di parole che prevedibilmente saranno usate in base ai principi di località

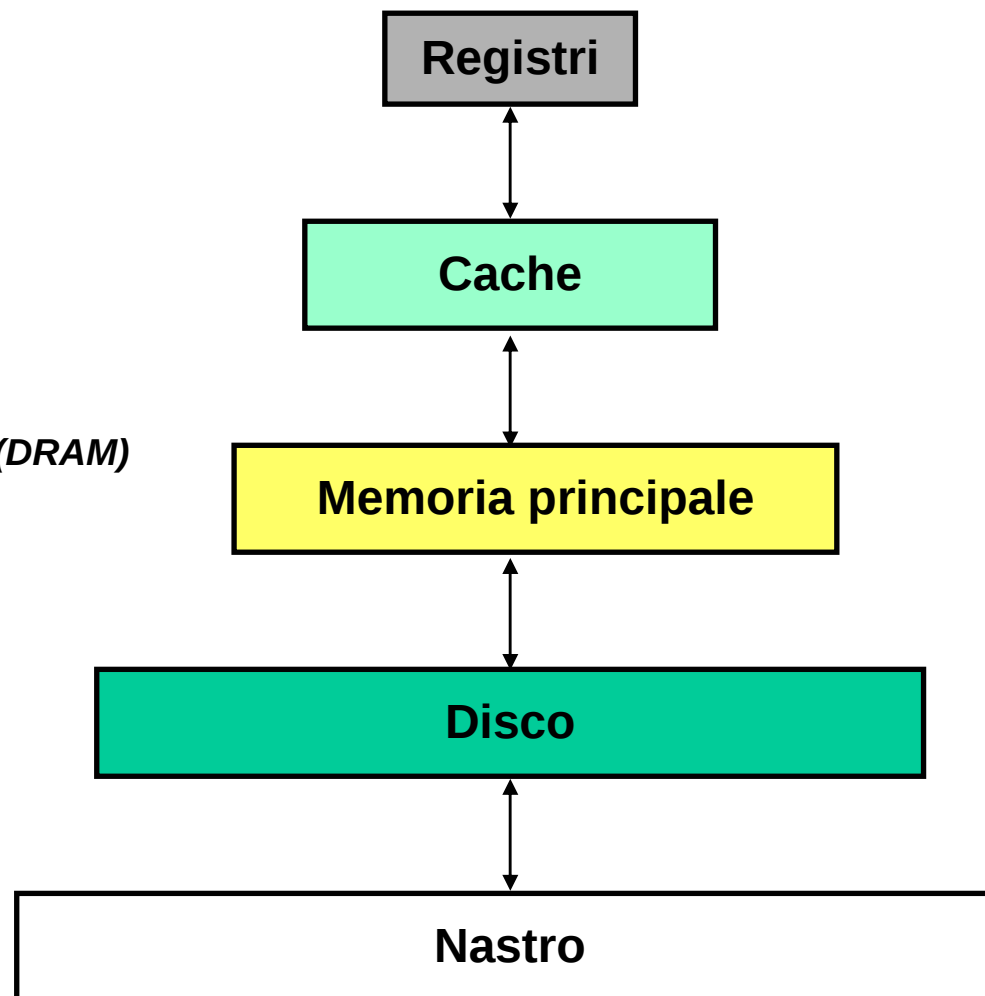
Registri CPU
< 1 KByte
0.25 – 0.5 ns

Cache (SRAM)
< 8 MByte
0.5 – 25 ns
100-250 \$ / MByte

Memoria principale (DRAM)
< 4 GByte
10 – 250 ns
5-10 \$ / MByte

Disco
G Byte,
10 ms
0.1-0.2 \$ / MByte

Nastro
G Byte - TByte
sec-min



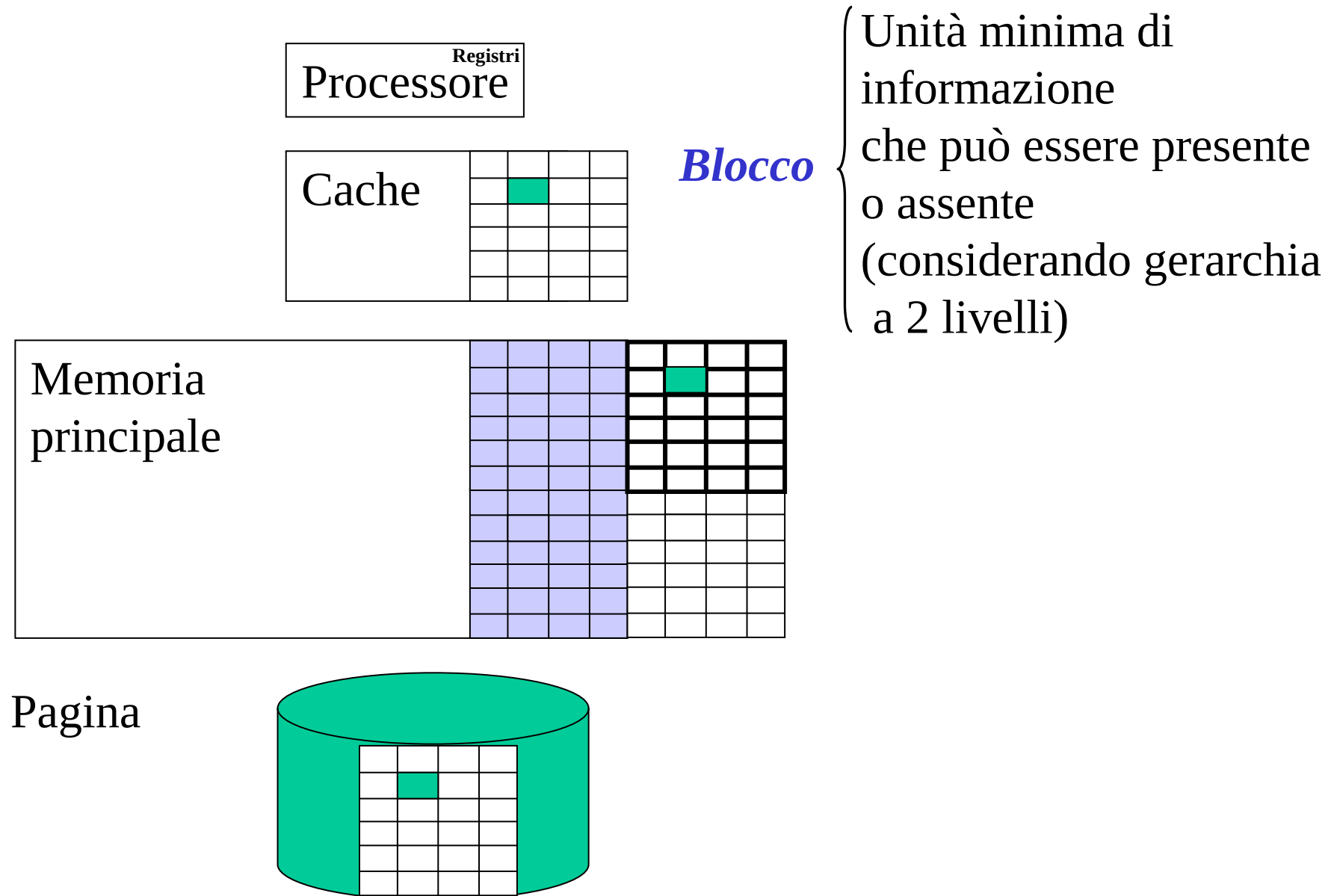
Livello superiore

+ veloce

+ grande

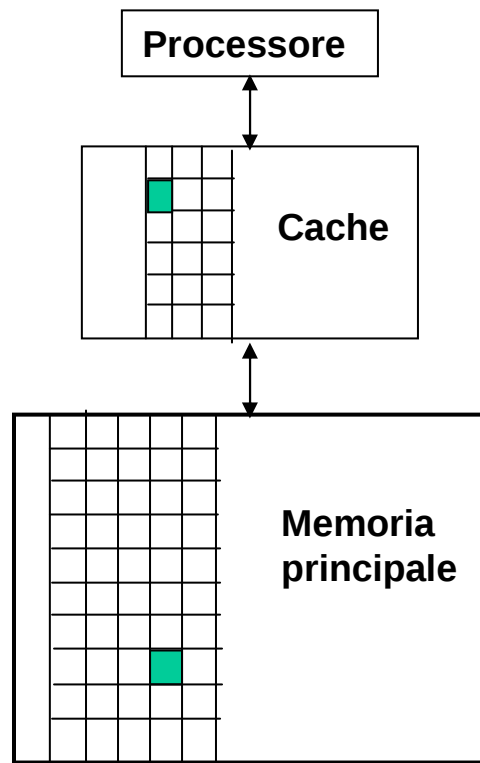
Livello inferiore

Gerarchia di memorie (2)



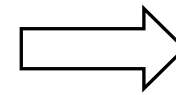
Hit e miss

- Se il dato richiesto dal processore compare in uno dei blocchi presenti nel livello superiore, la richiesta ha successo (*hit*).

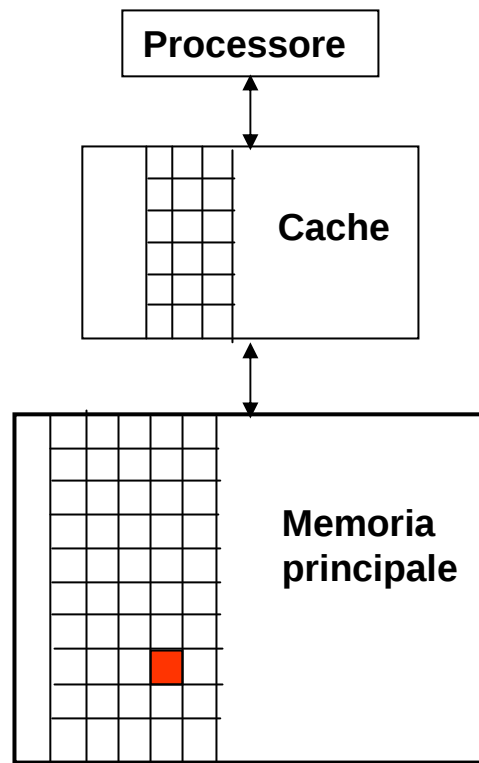


Hit e miss

- Se il dato richiesto dal processore non compare in uno dei blocchi presenti nel livello superiore, la richiesta fallisce (*miss*).



Il blocco viene reperito al livello inferiore



Miss:

- **Stallo** della CPU
- richiesta alla memoria princ. del blocco contenente il dato cercato
- copia in cache
- ripetizione dell'accesso in cache.

Prestazioni della memoria cache

Frequenza di hit (hit rate)

- frazione degli accessi che possono essere risolti nella cache

Frequenza di miss (miss rate)

- frazione degli accessi non risolti nella cache

$$\text{NB: } f_{\text{miss}} = 1 - f_{\text{hit}}$$

Tempo di hit

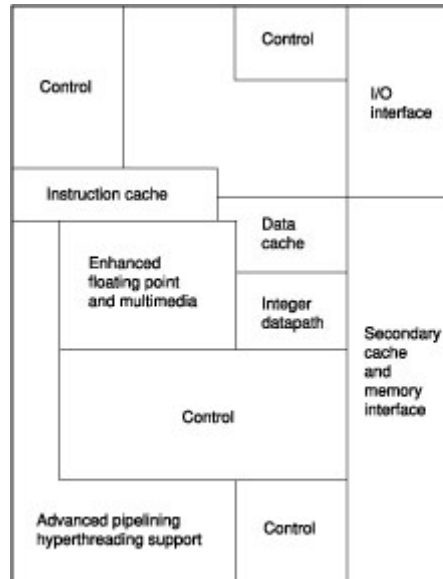
- tempo necessario per accedere alla cache in caso di hit
(include il tempo per determinare hit vs miss)

Penalità di miss

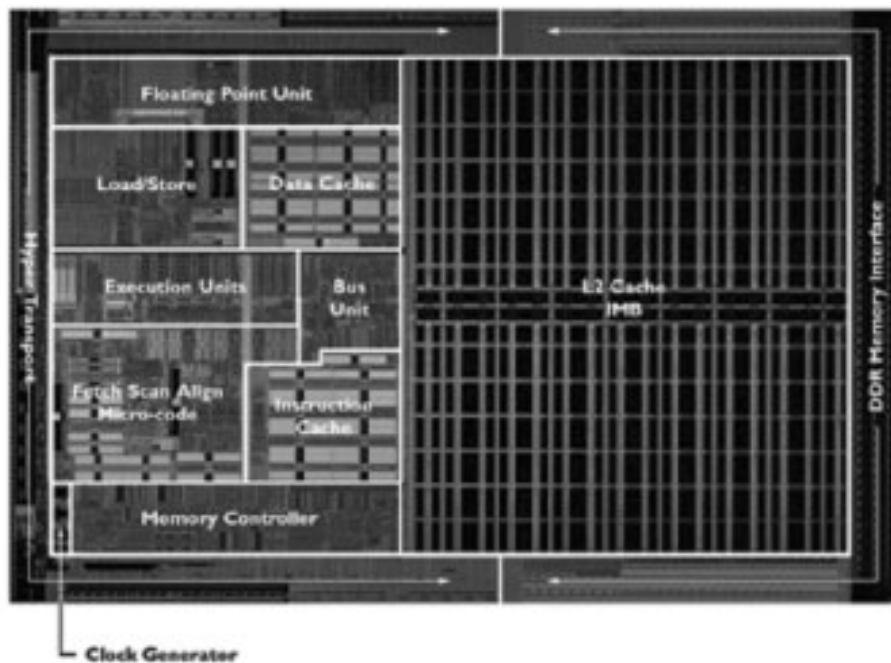
- tempo per sostituire il blocco del livello superiore con quello del livello inferiore + tempo per produrre il dato al processore

$$\text{NB: } T_{\text{miss}} = T_{\text{hit}} + \text{PEN}_{\text{miss}}$$

Quanto importante è la cache?



P4

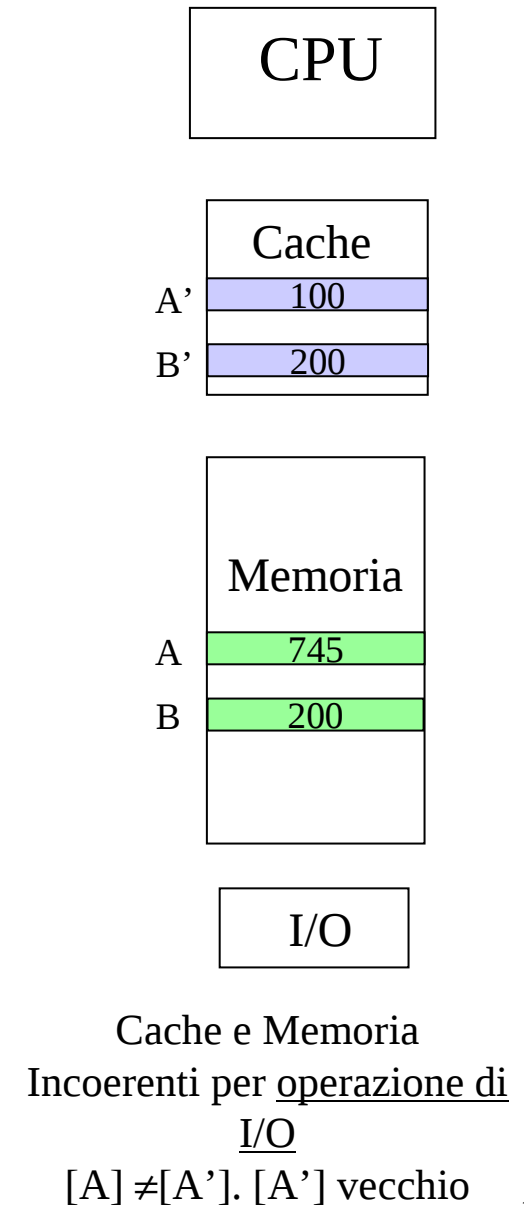
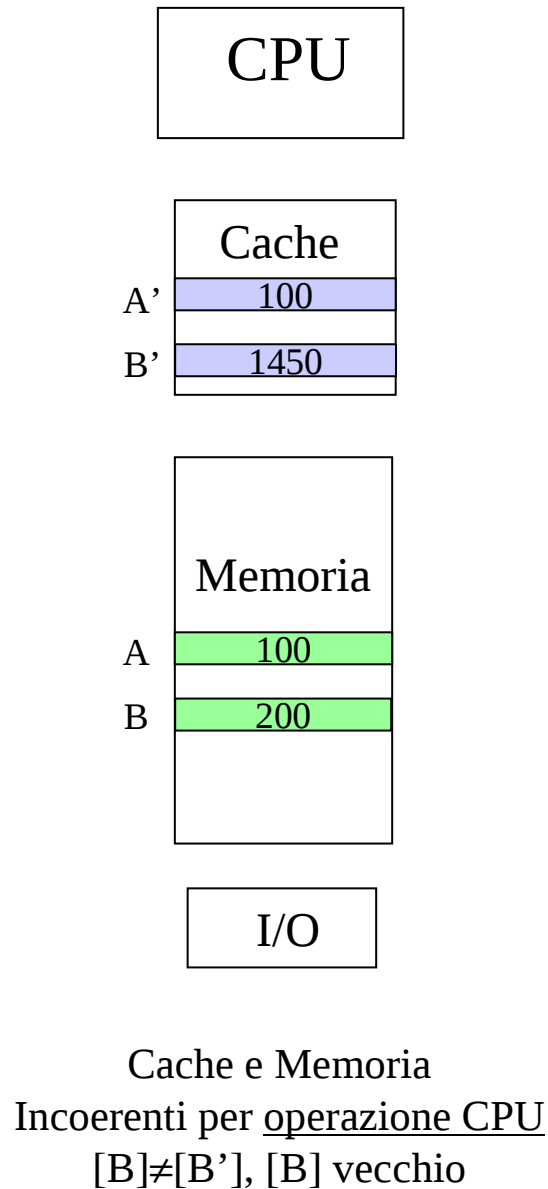
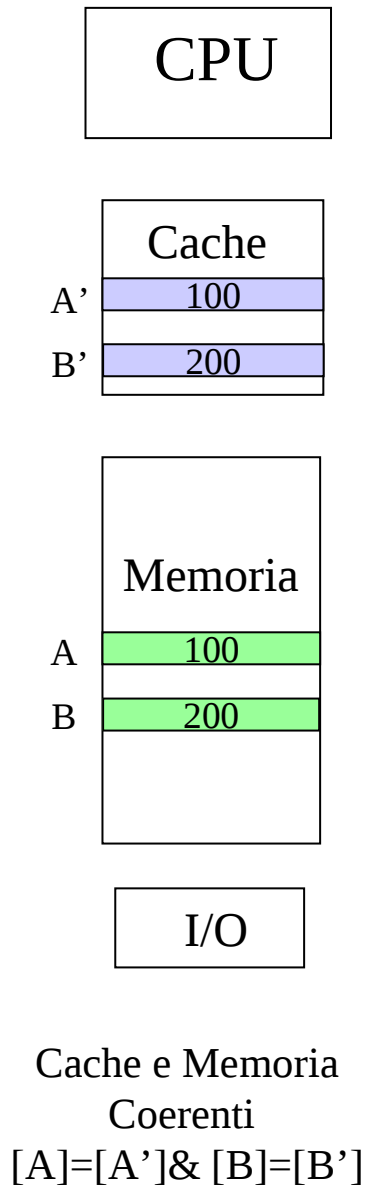


AMD Opteron

I problemi

- *Problema del piazzamento del blocco:*
 - se cerco un blocco (o un indirizzo) dove può essere nella cache?
(o anche: dove copio un blocco prelevato da memoria centrale?)
 - e come faccio a sapere se un blocco è presente nella cache?
- *Problema della sostituzione del blocco:*
 - se devo spostare un blocco dalla memoria centrale alla memoria cache e la cache è piena, quale tra i blocchi presenti sostituisco?
[problema che non è indipendente dal precedente]
- *Problema della coerenza dei dati:*
 - nella gerarchia di memoria ho più copie ripetute dello stesso dato: come faccio a mantenere la coerenza (e fornire quella attuale)?
[vedremo che questo ha a che fare con la strategia di scrittura]

Introduzione al problema della coerenza dei dati

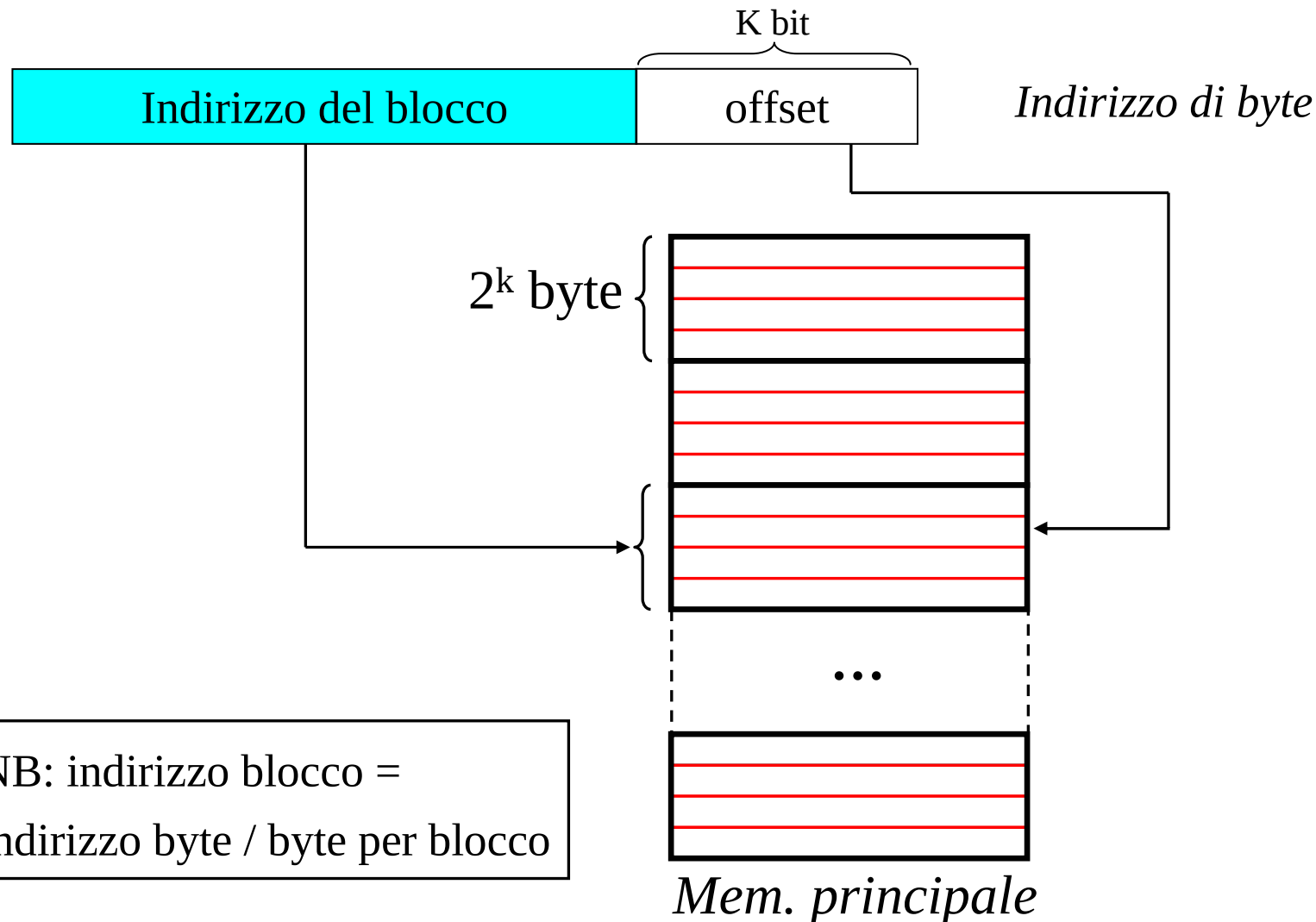


Il problema del piazzamento del blocco

- Dato un indirizzo di un blocco nella memoria principale, qual è la sua posizione nella memoria cache?
⇒ necessaria corrispondenza tra l'indirizzo in memoria del blocco e la locazione nella memoria cache.
- Questa corrispondenza dipende dall'architettura della memoria cache:
 - **Cache a corrispondenza diretta**
 - **Cache set-associativa a n vie**
 - **Cache completamente associativa**

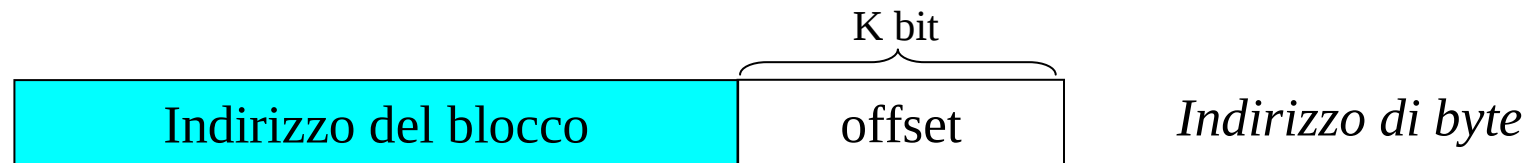
Un primo concetto: indirizzo del blocco vs. offset

- Assumiamo memoria (cache e princ.) suddivisa in blocchi di 2^k byte
- Ogni indirizzo può essere pensato suddiviso in due campi

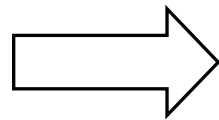
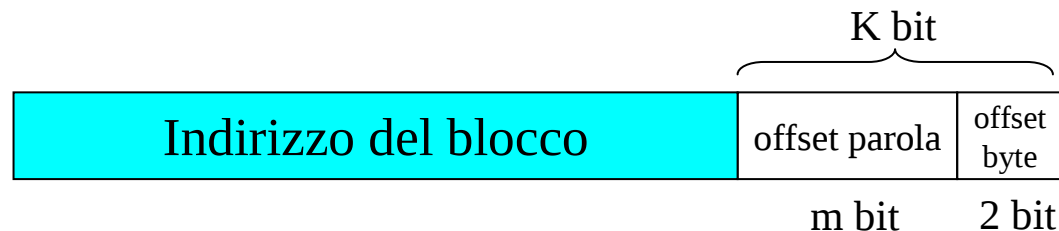


Un primo concetto: indirizzo del blocco vs. offset

- Assumiamo memoria (cache e princ.) suddivisa in blocchi di 2^k byte
- Ogni indirizzo può essere pensato suddiviso in due campi

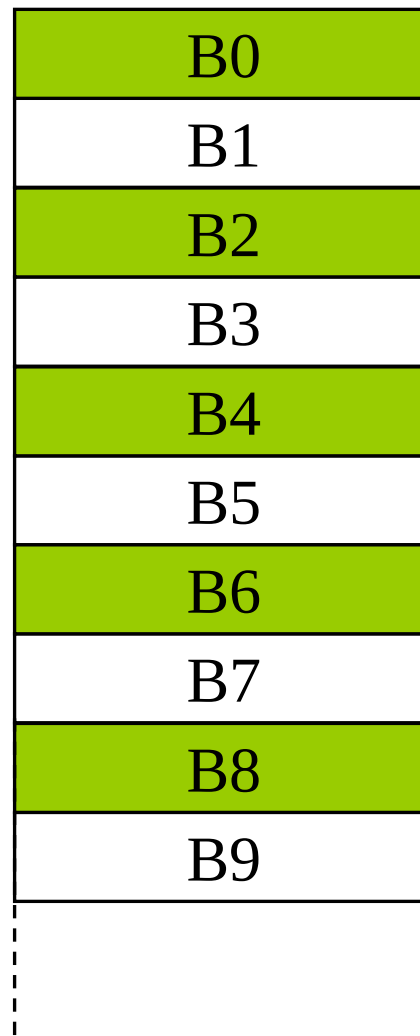


Se ciascun blocco ha m parole di 4 byte ciascuna, possiamo suddividere il campo offset in due sottocampi



FAREMO SEMPRE RIFERIMENTO
ALL'INDIRIZZO DEL BLOCCO!

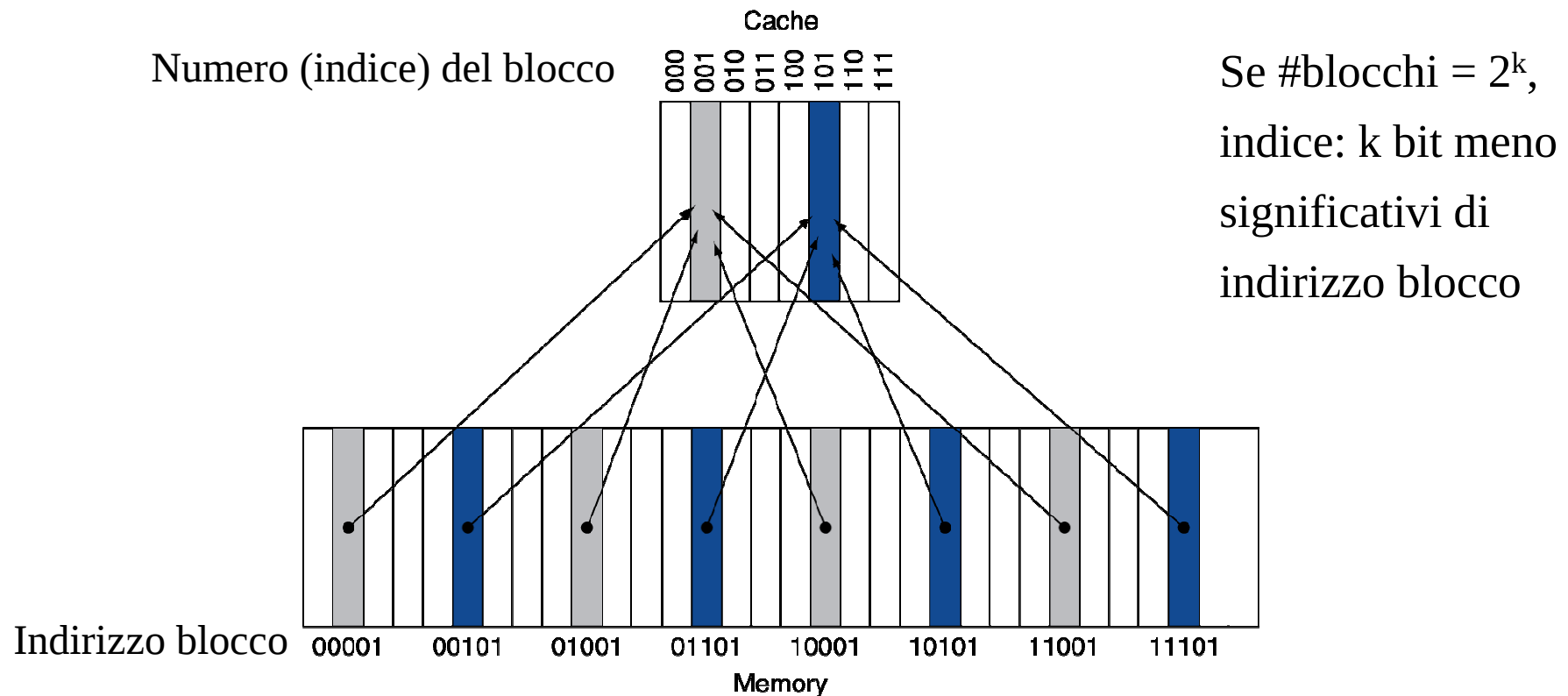
Indirizzo del blocco	offset parola	offset byte
----------------------	---------------	-------------



Cache a corrispondenza diretta (direct mapped)

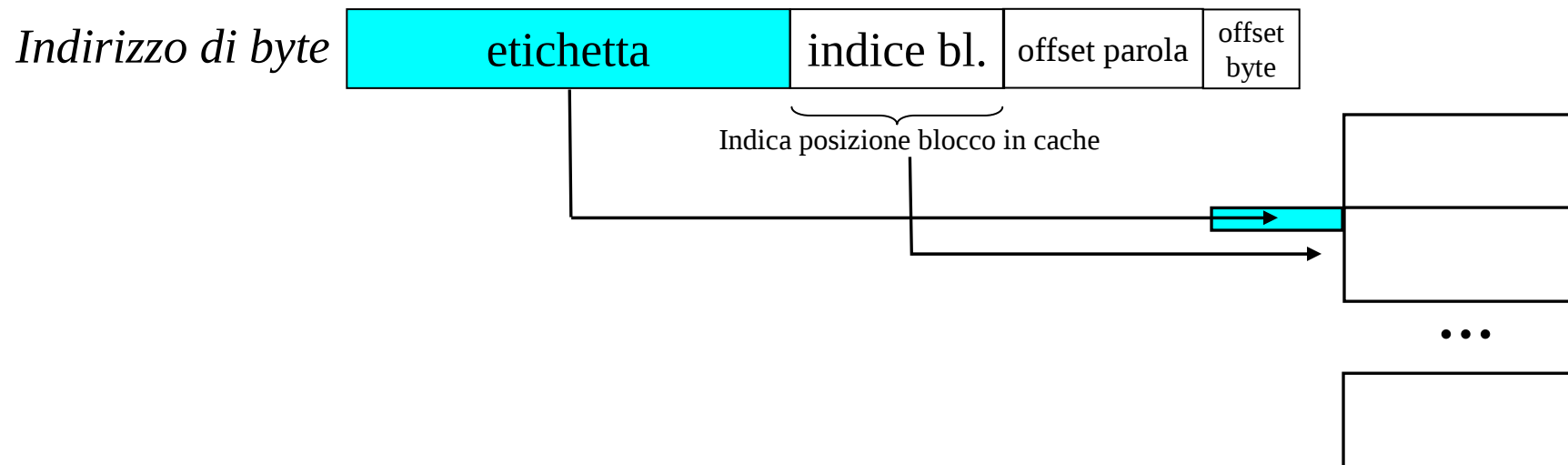
- Ogni blocco (e quindi ogni indirizzo di memoria) può risiedere solo in una posizione nella memoria cache

⇒ (indice blocco) = (indirizzo blocco) modulo (# blocchi in cache)



PROBLEMA 1: come sapere se un dato indirizzo è nella cache o no?

⇒ Per ogni blocco nella cache, serve memorizzare *etichetta* (**TAG**) che contiene i bit più significativi dell'indirizzo di blocco



PROBLEMA 2: come sapere se le informazioni (compresa tag) di un blocco sono valide?

⇒ *bit di validità* (**valid bit**) per ogni blocco o elemento (all'inizio indica non valido per tutti gli elementi!)

UN ESEMPIO (1)

- cache con 8 blocchi
(NB: indirizzo blocco = indirizzo di parola se 1 word/blocco)

Stato iniziale

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

UN ESEMPIO (2)

- cache con 8 blocchi

Block addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

UN ESEMPIO (3)

Block addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

UN ESEMPIO (4)

Block addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

UN ESEMPIO (5)

Block addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

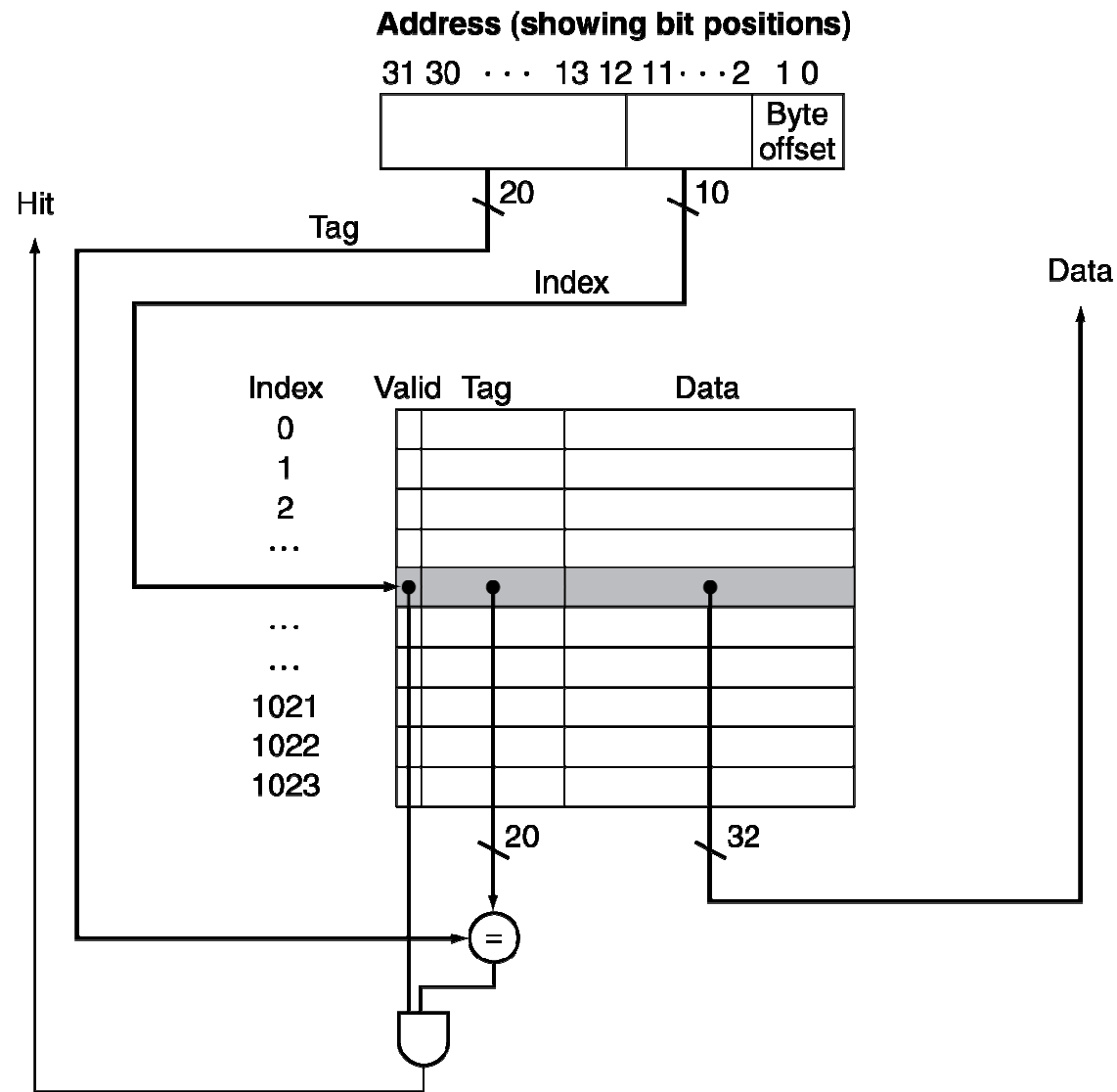
Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

UN ESEMPIO (6)

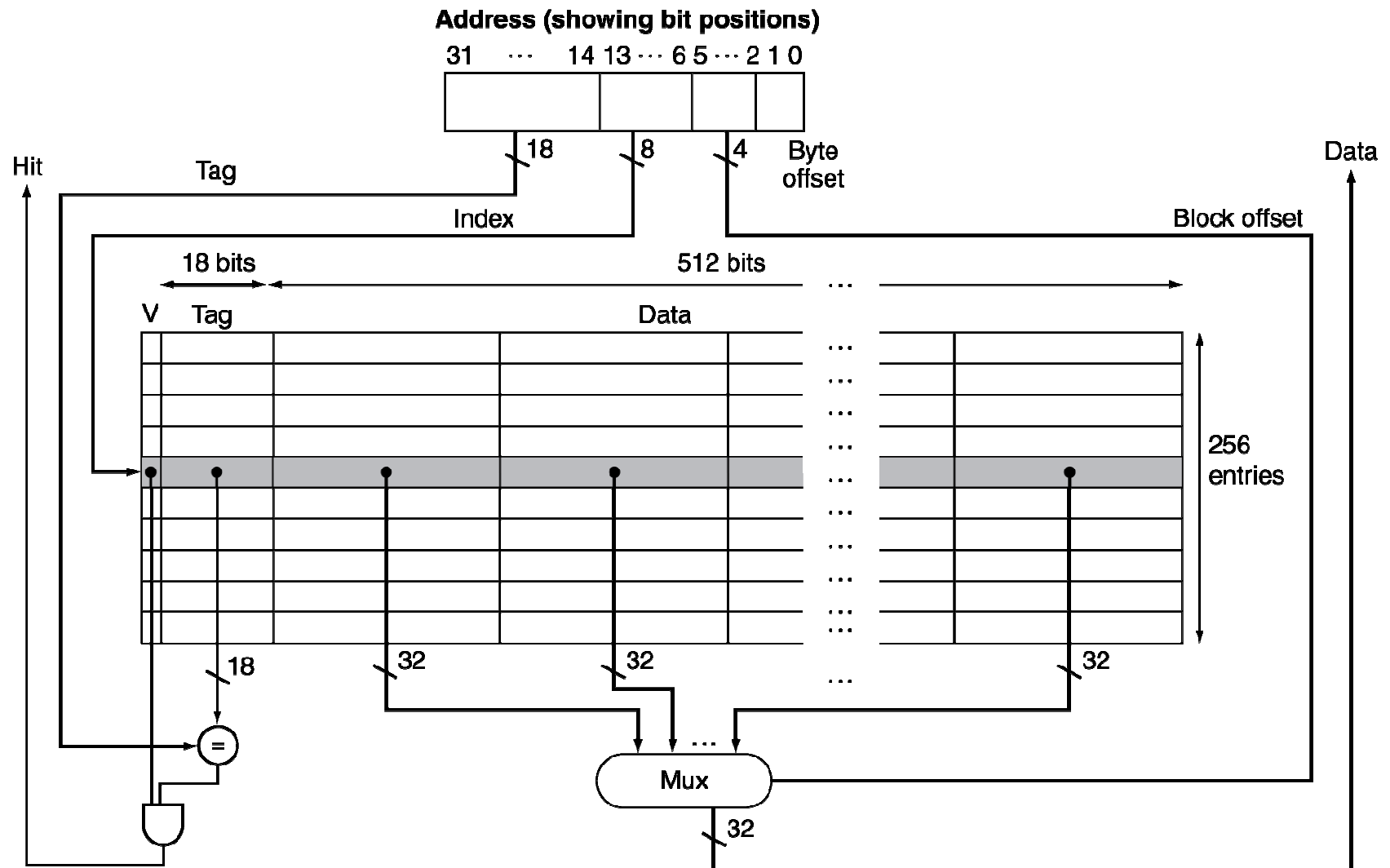
Block addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Schema con blocchi di una parola (ciascuna di 4 byte)

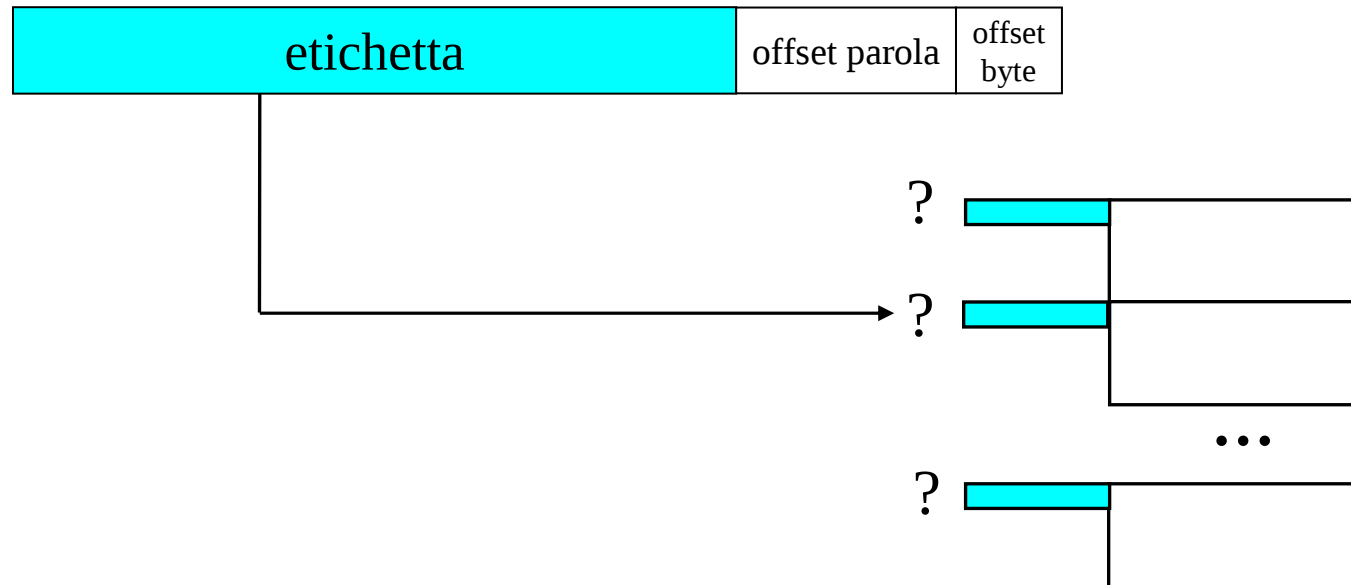


Schema con blocchi di 16 parole



Cache completamente associativa

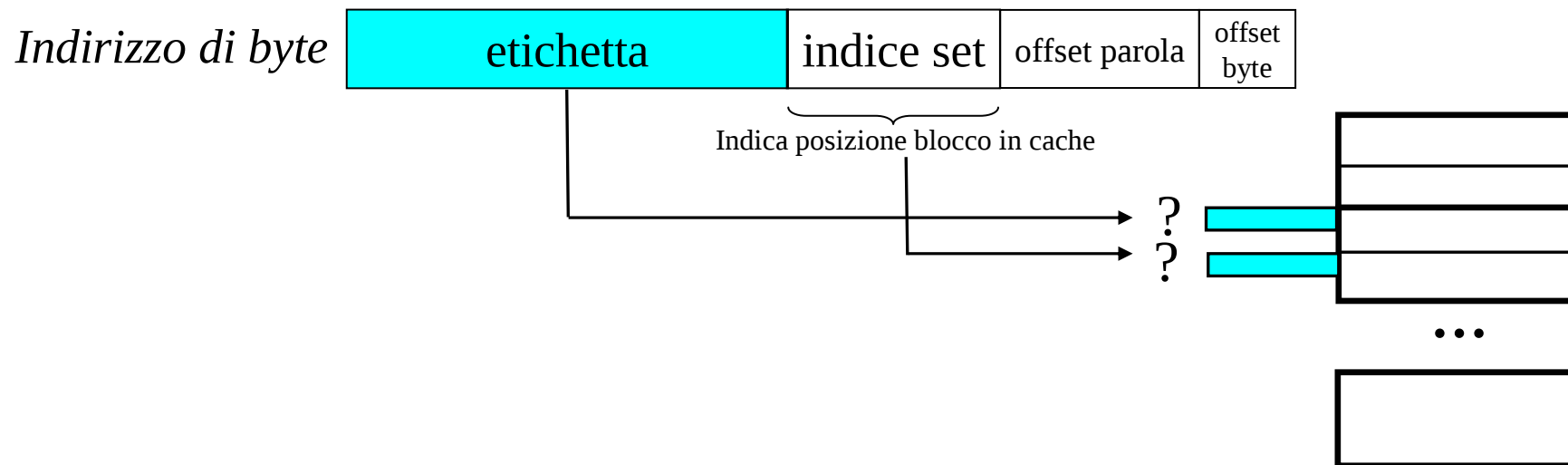
- Un blocco può essere posto in qualsunque posizione nella cache
- In pratica, l'indice del blocco scompare (non ho una posizione fissa) e ogni volta occorre esaminare tutti i blocchi della cache



- Vantaggio: tende a diminuire la frequenza di miss
- Svantaggi: maggior costo (comparatori) + può aumentare T_{hit} e T_{clock}

Cache set-associativa a n vie

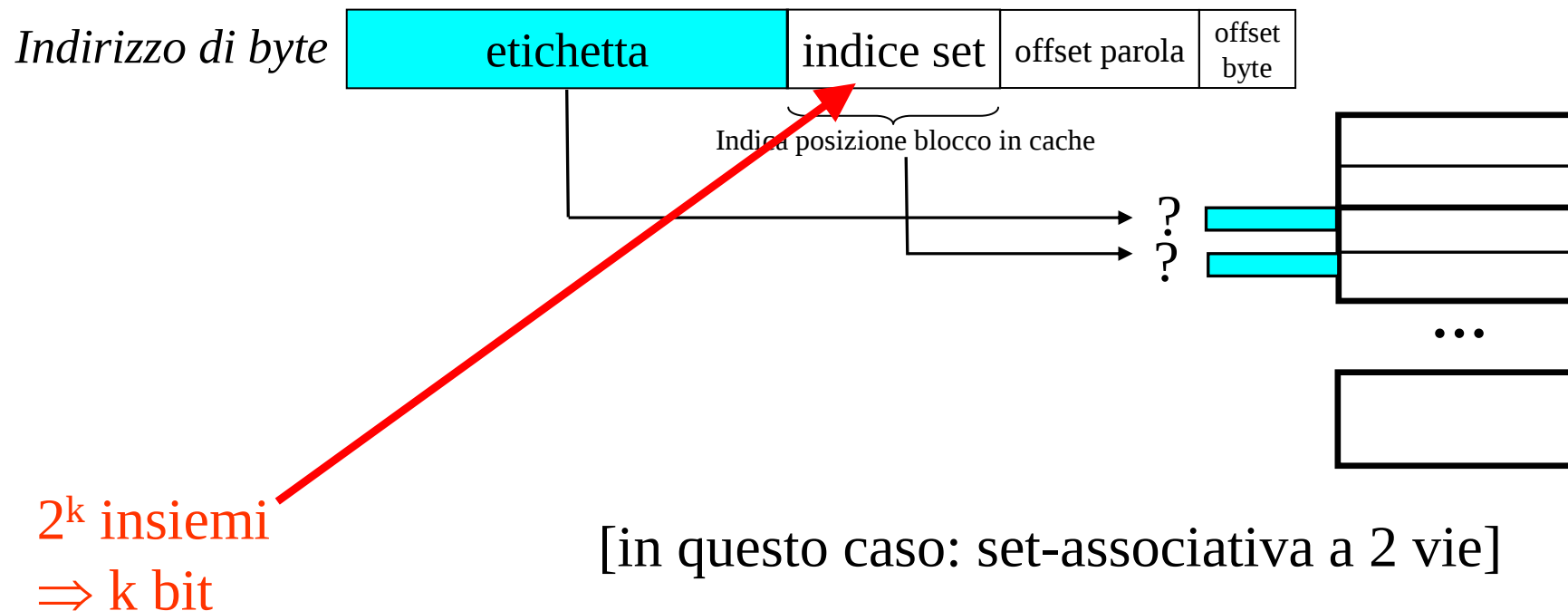
- La cache è suddivisa in *insiemi* (*set*), ognuno con *n* blocchi
- Un blocco può essere messo in un solo insieme (determinato dal suo indirizzo) ma può occupare uno qualunque degli *n* blocchi componenti



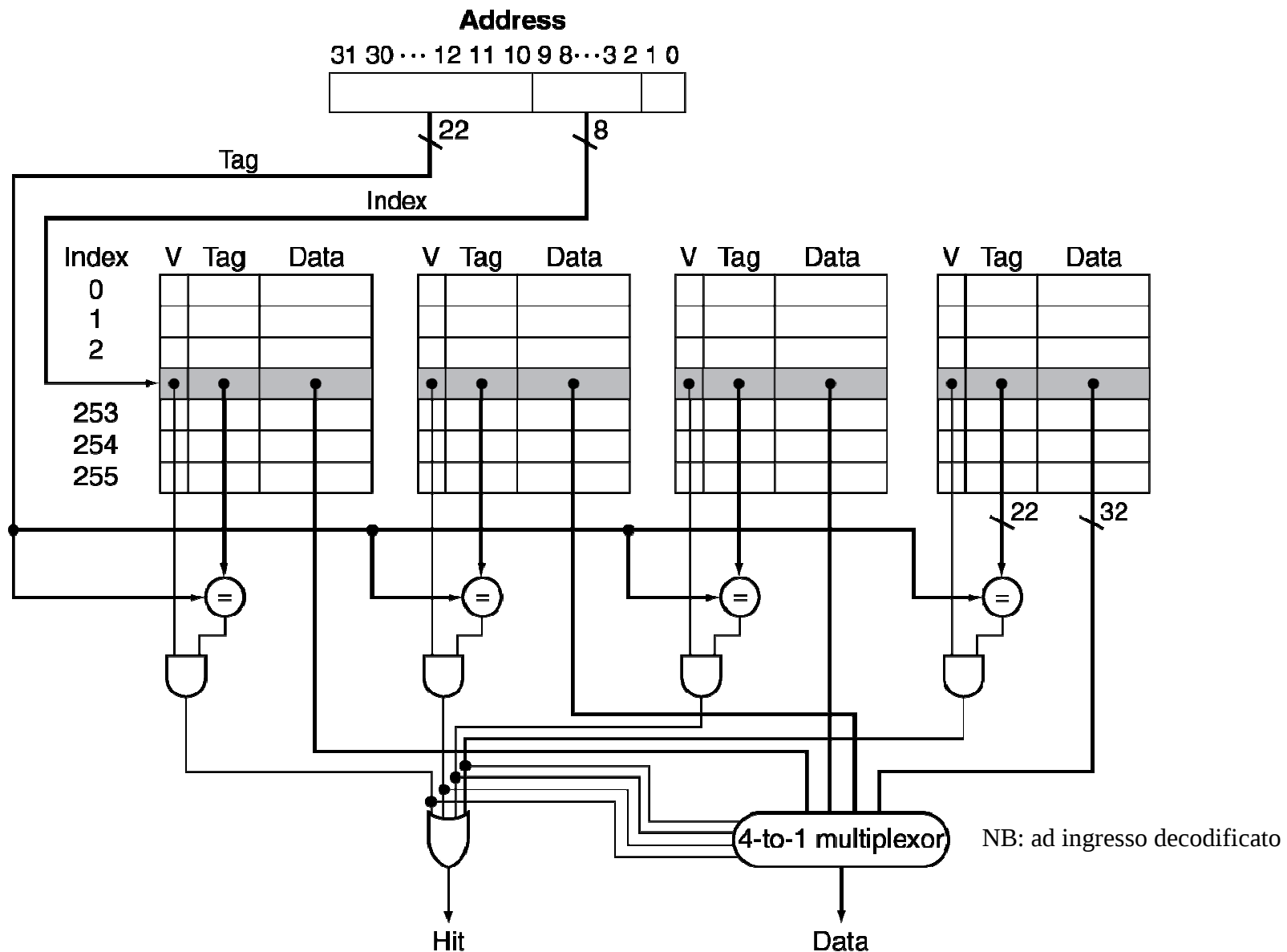
[in questo caso: set-associativa a 2 vie]

Cache set-associativa a n vie

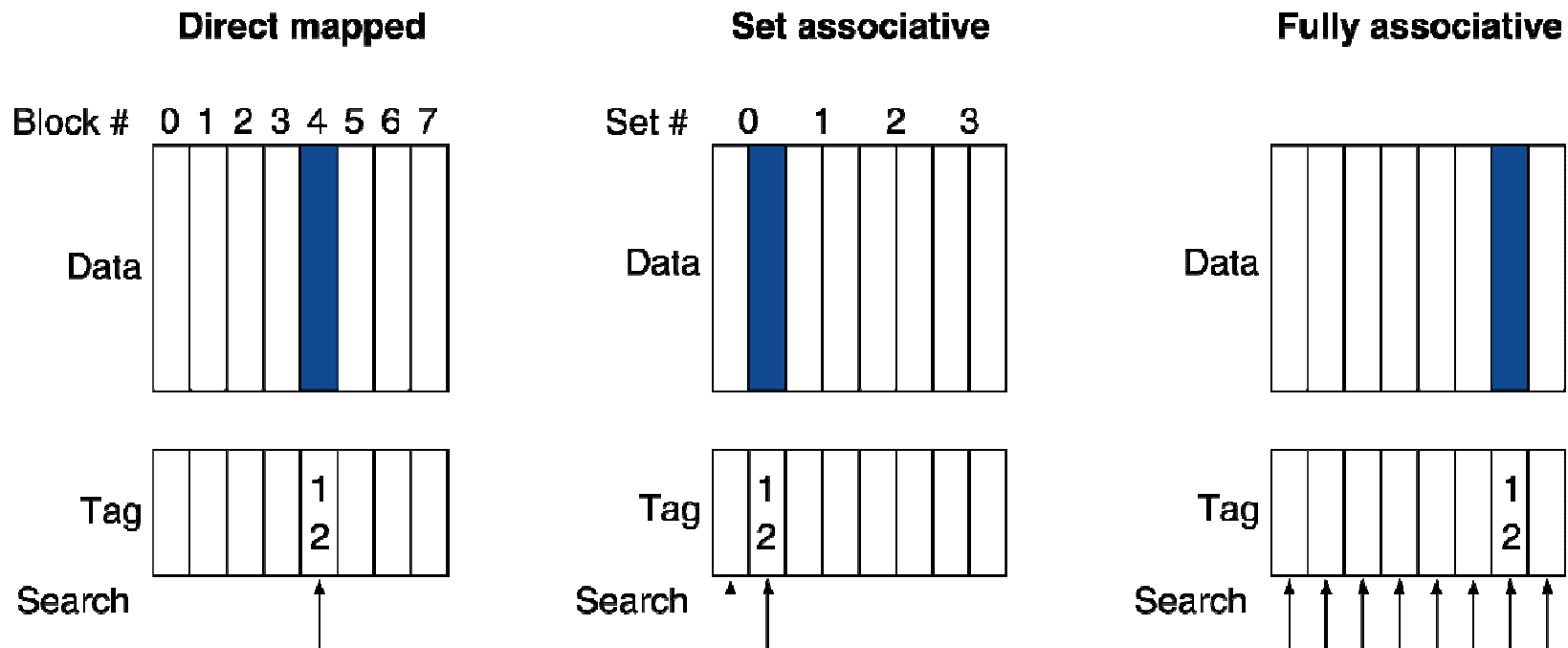
- La cache è suddivisa in *insiemi* (*set*), ognuno con *n* blocchi
- Un blocco può essere messo in un solo insieme (determinato dal suo indirizzo) ma può occupare uno qualunque degli *n* blocchi componenti



Cache set-associativa a 4 vie (256 insiemi) con blocchi di 1 parola



Una visione di insieme: identificazione del blocco



- Set-associativa a 1 via: equivale a corrispondenza diretta
 - calcolo posizione, verifica etichetta e verifica bit valido
- Set-associativa a n vie:
 - calcolo set, verifica etichette dell'insieme e bit valido
- Set-associativa a $\#blocchi$ vie: equivale a completamente associativa
 - confronto etichetta in ogni blocco e verifica bit valido

Una visione di insieme: diversi gradi di associatività

- Per una cache con 8 blocchi

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Una visione di insieme: indirizzi

- Indirizzamento diretto: blocco memoria in posizione fissa



- Indirizzamento set-associativo



- Indirizzamento completamente associativo



Il problema della sostituzione del blocco

- Nel caso di cache a corrispondenza diretta:
c'è solo un blocco da sostituire!
- Nel caso di cache set-associativa (e completamente associativa):
scelta di un blocco all'interno del set con diverse politiche:
 - **LRU** (least recently used)
ideale, ma costoso per cache con grado di associatività > 2
 - LRU approssimato
ad esempio, per set-associative a 4 vie:
2 coppie di blocchi con 1 bit LRU per coppia e per blocco
 - **random**
NB: per grado 2, f_{miss} 1.1 volte rispetto a LRU
e la differenza decresce per gradi di associatività maggiori

Il problema della coerenza dei dati

Esaminiamo cosa succede quando la CPU legge o scrive nella CACHE

LETTURA

- hit: la parola/byte richiesta viene consegnata al processore
- miss (tipicamente richiede più cicli di clock):
 - stallo della CPU in fase di fetch
(contenuto registri visibili e di appoggio non deve cambiare)
 - un controllore separato accede alla memoria principale
 - > invio di PC-4 alla memoria principale
 - > lettura dalla memoria del blocco
 - > aggiornamento del tag con i bit più significativi di PC
 - CPU riprende ripetendo il fetch, questa volta con hit!

SCRITTURA può generare inconsistenze!

Usando la tecnica del *write-through*

- hit:
 - il dato è scritto sia nella cache sia in memoria
- miss:
 - trasferimento del blocco da memoria a cache
 - scrittura del dato sia nella cache sia in memoria

⇒ Tecnica costosa in termini di tempo (attesa del tempo di accesso della memoria principale)

⇒ necessario l'uso di *write buffer*

- stallo in hit solo quando il buffer è pieno
- maggiore è la frequenza di scrittura maggiore deve essere la capienza del buffer

Usando la tecnica del *write-back*

- hit:
 - il dato è scritto solo nella cache
- miss:
 - trasferimento del blocco da memoria a cache
 - scrittura del dato solo nella cache
- Il blocco è trasferito in memoria principale solo quando deve essere rimpiazzato e solo se è stato effettivamente modificato da una scrittura (uso di un *dirty bit*)
- Anche in questo caso si può usare un *write buffer* quando un blocco deve essere rimpiazzato e scritto in DRAM

ESAMINIAMO COSA SUCCEDDE QUANDO DISPOSITIVI TRASFERISCONO DATI DA/A MEM. PRINCIPALE (es: DMA)

- ogni volta che DMA trasferisce dati da dispositivo a memoria DRAM:
 - se il blocco è presente in cache il sistema operativo
pone il bit di validità a 0 [dati incoerenti con DRAM]
- ogni volta che DMA trasferisce dati da DRAM [blocco] a dispositivo:
 - se cache usa write-through: nessun problema (DRAM coerente)
 - con write-back: la memoria può non riflettere la cache,
 - una possibilità è lo svuotamento (flush) cache:
 - i blocchi con dirty bit a 1 sono trasferiti
 - in memoria DRAM

PRESTAZIONI: VALUTAZIONE QUALITATIVA

Vediamo come diverse scelte di progetto influiscono su:

- frequenza di miss
- tempo di hit
- tempo (o penalità) di miss

considerando in particolare:

- il grado di associatività della cache
- la dimensione dei blocchi
- la tecnica di scrittura
- il progetto del sistema di memoria (cache-DRAM)

Influenza del grado di associatività sulle prestazioni

Vantaggi aumento del grado di associatività

- Frequenza di miss tende a diminuire (riduzione dei miss che derivano da competizione per un blocco, cfr. esempio successivo)
- Ma il vantaggio è marcato da 1 a 2 vie, poi diminuisce
- Ed il vantaggio è minore per cache più grandi (con molti blocchi)

Svantaggi

- Maggior costo (n vie $\Rightarrow n$ comparatori)
- Potenziale aumento T_{hit} e T_{clock}

⇒ SCELTA: compromesso costo di un miss
vs. costo per implementare associatività
(sia costo hw che aumento T_{hit})

Esempio: influenza del grado di associatività (4 blocchi)

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	Mem[8]			
0	0	miss	Mem[0]			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	Mem[8]		Mem[6]	

1

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	miss	Mem[0]	Mem[6]		
8	0	miss	Mem[8]	Mem[6]		

2

Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	Mem[8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	Mem[8]	Mem[6]	

4

Influenza della dimensione dei blocchi

- Blocchi grandi $\Rightarrow$ riducono frequenza di miss
 - a causa della località spaziale

PERO'

- Blocchi grandi $\Rightarrow$ aumenta la competizione per i blocchi e quindi aumenta la frequenza di miss!
 - perché si riduce il numero di blocchi
- Blocchi grandi $\Rightarrow$ aumenta la penalità di miss
 - perché il tempo di trasferimento dal livello superiore aumenta
 - Strategie per ridurre penalità di miss:
 - early restart (processore prosegue appena arriva la parola)
 - requested word first (poi si prosegue con quelle successive e riprendendo dall'inizio del blocco)
 - Nota: non agiscono su tempo di latenza prima parola

La tecnica di scrittura: write-through vs. write-back

Vantaggi write-back (e svantaggi write-through)

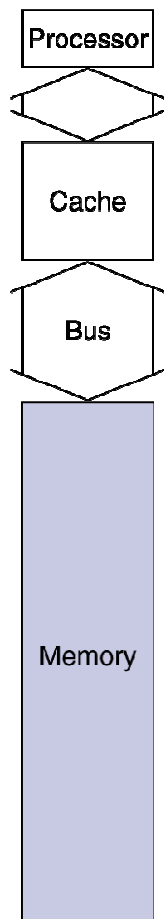
- In scrittura non occorre aspettare il tempo di accesso della memoria principale (cfr. però uso di buffer nel write-through)
- Si fanno meno scritture sulla memoria principale (scrittura di un blocco solo quando è rimpiazzato)
- Si possono sfruttare i vantaggi della scrittura di interi blocchi (ottimizzazione del sistema di memoria per trasferimento blocchi)

Svantaggi write-back (e vantaggi write-through)

- I miss sono più semplici e meno costosi nel write-through:
 - non occorre mai rimpiazzare un blocco perché DRAM è coerente (cfr. però uso di write buffer nel caso write-back)
- Scrittura è più complessa con write-back: ogni volta che si scrive occorre prima controllare l'etichetta (pericoloso sovrascrivere!)

Progetto del sistema di memoria

- Obiettivo: $\uparrow$ banda bus cache-memoria $\Rightarrow$ $\downarrow$ penalità di miss
- e così posso permettermi blocchi più grandi...
- Problema: il ciclo di bus $\gg T_{\text{clock}}$ processore (fino a 10 volte)



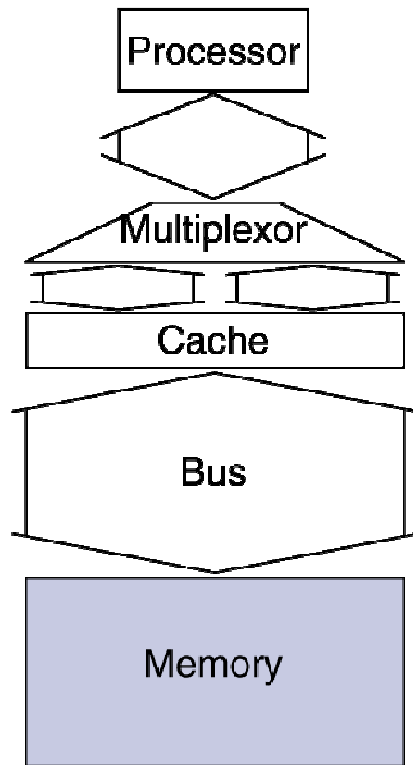
a. One-word-wide
memory organization

Esempio:

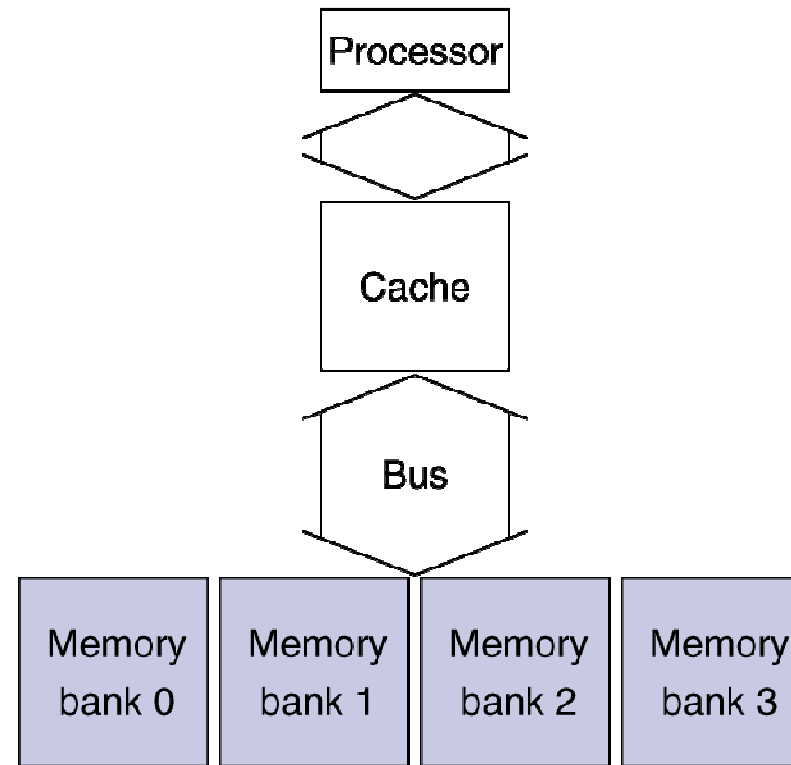
- cache con blocchi di 4 parole
- 1 ciclo bus per indirizzo, 15 per accesso DRAM, 1 per trasferimento dati

$\Rightarrow$ Per un miss: $1 + 4 \cdot 15 + 4 = 65$ cicli di bus

$\Rightarrow 16/65 = 0.25$ byte per ciclo



b. Wider memory organization



c. Interleaved memory organization

Es: bus largo 2 parole:

$$1 + 2 \cdot (15 + 1) = 33 \text{ cicli}$$

$$\Rightarrow 16/33 = 0.48 \text{ byte/ciclo}$$

(maggior costo e MUX + logica di controllo può aumentare T_{acc})

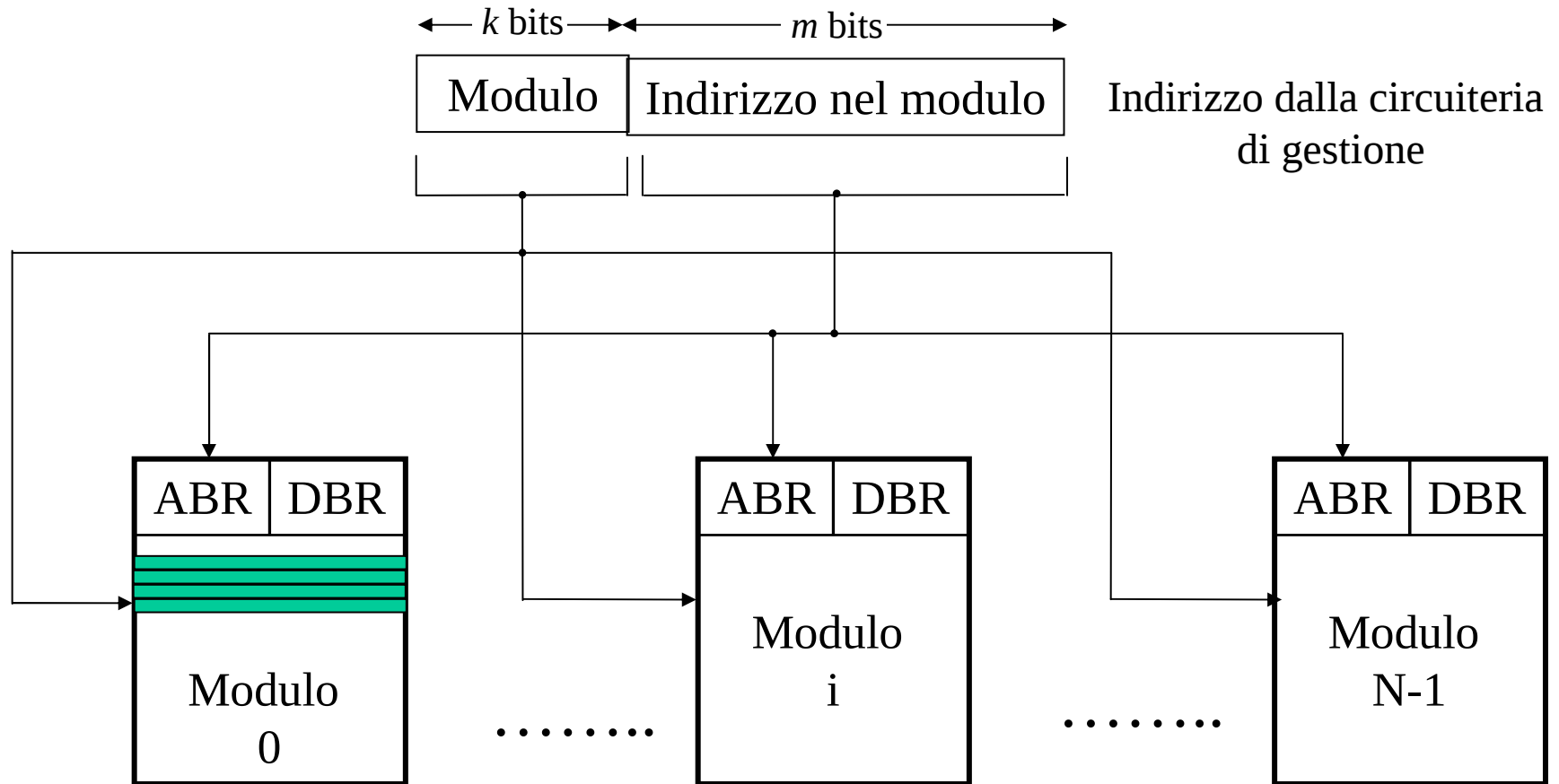
Es: invio indirizzo in parallelo

$$1 + 15 + 4 \cdot 1 = 20 \text{ cicli}$$

$$\Rightarrow 16/20 = 0.80 \text{ byte/ciclo}$$

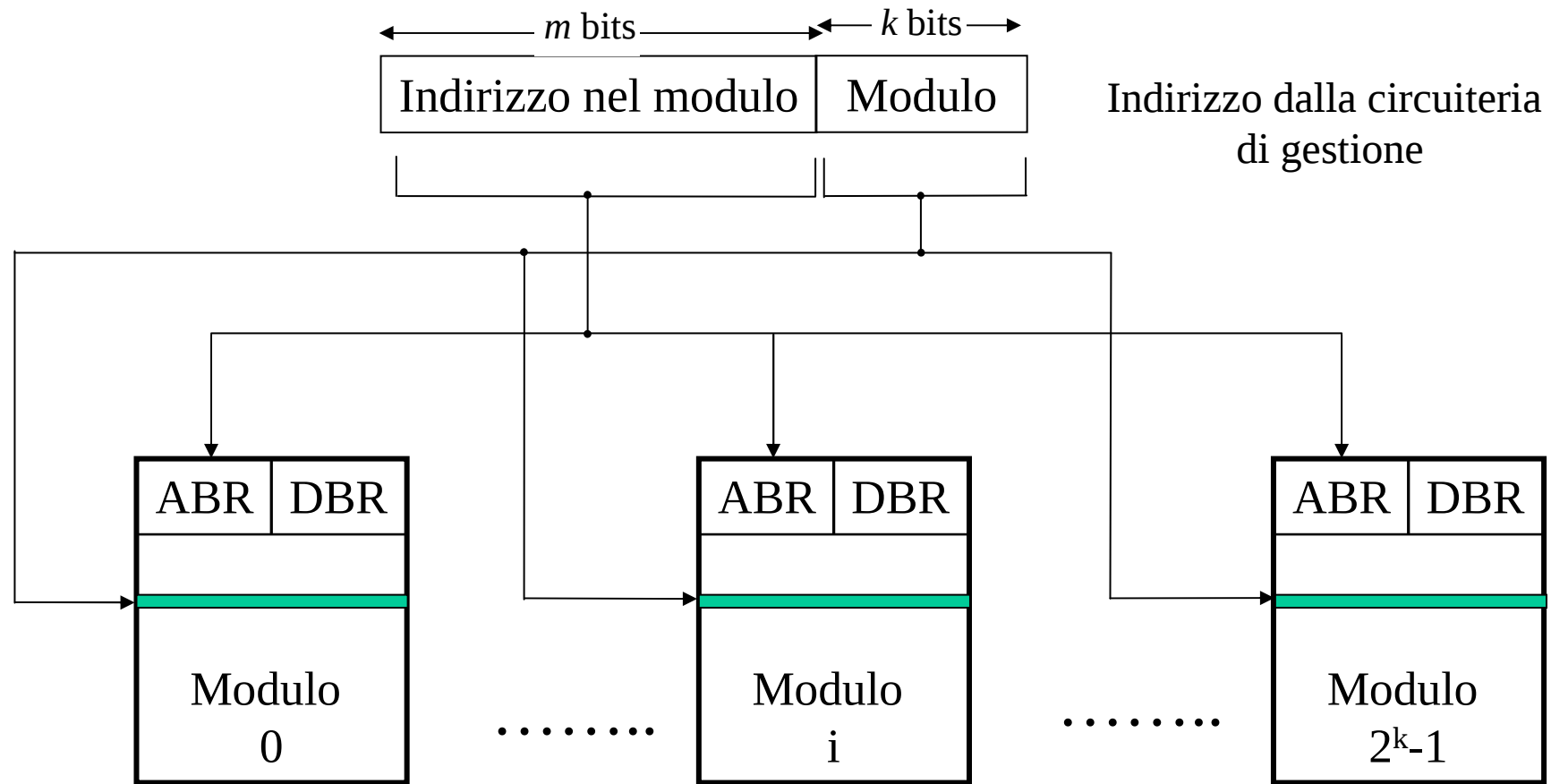
(ed in scrittura si può scrivere in parallelo su tutti i banchi)

Senza interallacciamento: le parole di un blocco sono memorizzate consecutivamente in un modulo



NB: ha comunque il vantaggio che, intanto, DMA può accedere a moduli che non si trovano nel blocco acceduto correntemente

Con interallacciamento: Le parole di un blocco sono sparpagliate e memorizzate ognuna in un modulo diverso



➡ Consente di attivare in parallelo i chips DRAM, risparmiando cicli

NOTA

L'ampiezza di banda è aumentata anche dall'uso di moderne DRAM

- SDRAM (Synchronous DRAM):
modalità burst
- DDR SDRAM (Double Data Rate SDRAM):
lavorano sia sul fronte di salita sia sul fronte di discesa

SCHEMA RIASSUNTIVO

	T_{hit}	f_{miss}	PEN_{miss}
Dim. blocchi		↓	↑
Associatività	↑	↓	
Banda bus			↓
Write back* vs. write-through	↓		↑

* i buffer rendono più complicata la comparazione

IMPATTO DELLA CACHE SU PRESTAZIONI

$$T_{\text{CPU}} = \# \text{cicli} * T_{\text{clock}}$$

└─ Esecuzione delle istruzioni + cicli di stallo per cache

└─
I * CPI_{id}

└─
- hit
- miss in lettura
- miss in scrittura

Assunzione: T_{hit} pari a T_{clock} (hit impiega sempre 1 ciclo di clock)

- write-through: write buffer per evitare overhead in scrittura
- write-back: uso store-buffer per poter effettuare la scrittura in un ciclo di clock (altrimenti me ne servono due)

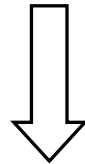
Cicli di stallo in lettura

- write-through: caricamento blocco DRAM $\rightarrow$ CACHE
- write-back: può dover anche rimpiazzare un blocco
(CACHE $\rightarrow$ DRAM), ma si usa *write buffer*

Cicli di stallo in scrittura

- write-through: caricamento di un blocco
- write-back: caricamento di un blocco (+ rimpiazzamento)

+ CICLI DI STALLO DOVUTI A WRITE-BUFFER



ASSUNZIONE: TRASCURABILI

In formule:

$$\begin{aligned} T_{\text{CPU}} &= (I * \text{CPI}_{\text{id}} + C_{\text{miss}}) * T_{\text{clock}} \\ &= I * (\text{CPI}_{\text{id}} + \delta_{\text{miss}}) * T_{\text{clock}} \end{aligned}$$

Ovvero: $\text{CPI} = \text{CPI}_{\text{id}} + \delta_{\text{miss}}$

$$\delta_{\text{miss}} = \delta_{\text{miss-lettura}} + \delta_{\text{miss-scrittura}}$$

$$= f_{\text{lettura}} * f_{\text{miss-lettura}} * P_{\text{lettura}} + f_{\text{scrittura}} * f_{\text{miss-scrittura}} * P_{\text{scrittura}}$$

NB: δ_{miss} indica i cicli di stallo medi per istruzione!

Tempo medio di accesso alla memoria (Average Memory Access Time)

$$T_{AMAT} = T_{HIT} + f_{miss} * P_{miss}$$

caratterizza la cache tenendo conto anche del tempo di hit

Esempio

- CPU con $T_{clock}=1ns$, $T_{hit}= 1$ ciclo, $P_{miss} = 20$ cicli, $f_{miss} = 5\%$

(NB: f_{miss} calcolata complessivamente)

- $T_{AMAT} = 1ns + 0,05*20ns = 2ns$

(2 cicli di clock)

CACHE MULTILIVELLO

- Per avere tempi di accesso paragonabili a T_{clock} processore:
 - cache integrata all'interno del chip processore
 - piccole dimensioni e capacità

$\Rightarrow T_{\text{hit}}$ ridotto, ma aumenta f_{miss}
- Oltre alla cache L1, si usa una cache L2 (interna o esterna):
 - ad ogni miss della cache L1, si accede a cache L2 con tempo di accesso molto inferiore rispetto a DRAM (riduzione P_{miss})
 - solo se ho un miss anche in cache L2 si accede a DRAM
- Alcuni sistemi usano anche una cache L3

CONSIDERAZIONI DI PROGETTO

- Progetto cache L1 orientato a minimizzare T_{hit}
 - $\Rightarrow$ L1 di capacità inferiore vs. L2 (anche > 10 volte)
 - $\Rightarrow$ ed anche rispetto ad un progetto con una sola cache (miss diventano meno costosi)
- Progetto cache L2 orientato a minimizzare *frequenza di miss globale* (ovvero: % accessi in DRAM per istruzione)
 - $\Rightarrow$ più capiente rispetto a L1
 - $\Rightarrow$ ed anche rispetto ad un progetto con una sola cache (T_{hit} meno critico, interviene solo nei miss L1)