

Modulo Calcolatori Elettronici

Proff. Gian Luca Marcialis, Giulia Orrù, Lorenzo Putzu, Fabio Roli

Corsi di Laurea in Ingegneria Biomedica Ingegneria Elettrica, Elettronica ed Informatica

Contatti:

marcialis@unica.it, giulia.orrù@unica.it, lorenzo.putzu@unica.it

Capitolo 3

Unità di memoria

Fonti Principali: Patterson, A.D., Hennessy, J., "Struttura e progetto dei calcolatori elettronici", Zanichelli editore, 2019

Sommario

- Introduzione
- Caratteristiche di un sistema di memoria
- Gerarchie di memoria
- Memorie Interne
- Memoria Cache
- Memorie Esterne
- Codici a Correzione di Errore

Introduzione (1)

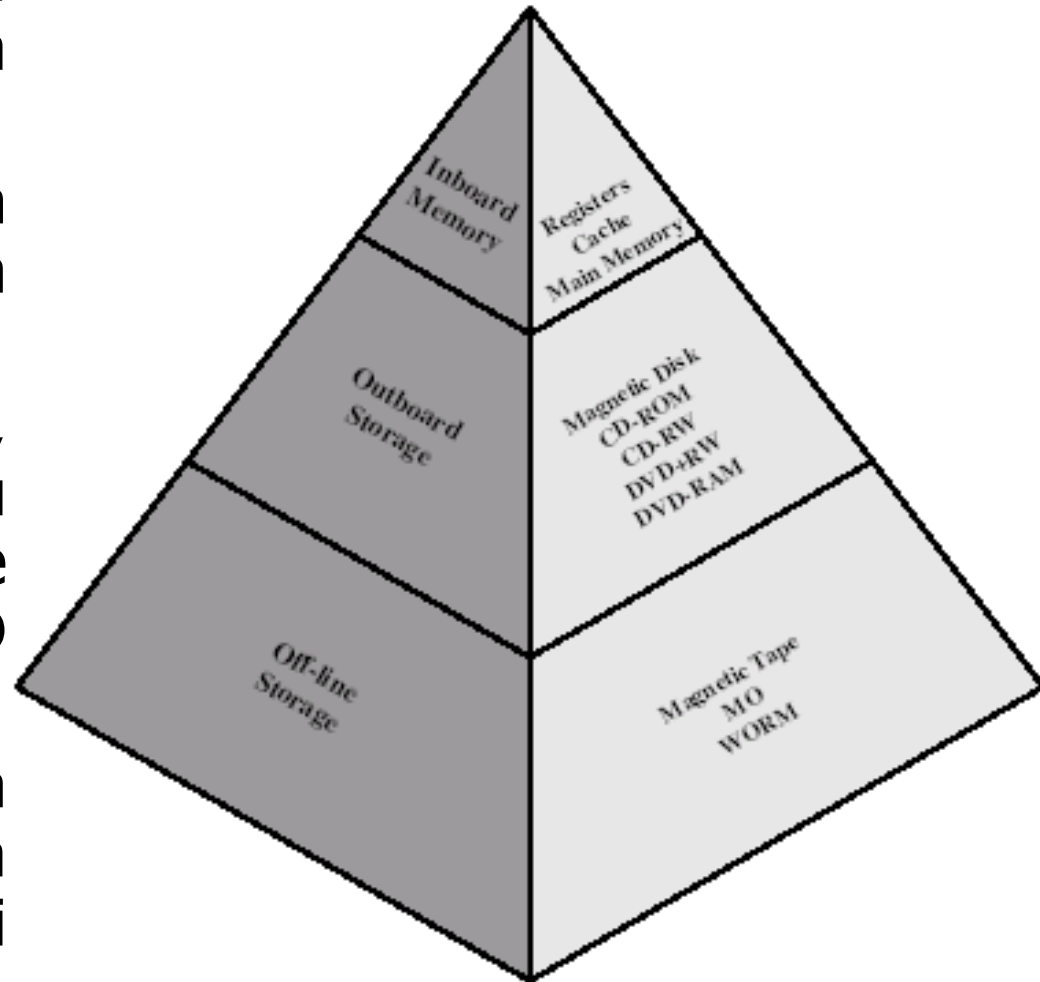
- Secondo lo schema di von Neumann, l'architettura di un calcolatore prevede tre unità fondamentali: l'unità centrale di elaborazione (CPU), la memoria, e l'unità di ingresso/uscita.
- La memoria è la parte del calcolatore che si occupa di immagazzinare dati, istruzioni e tutto ciò che occorre per il corretto funzionamento del sistema.
- Vedremo che è più corretto parlare di Sistema (od Unità) di Memoria, poiché la memoria di un moderno calcolatore è un sistema di memorizzazione complesso, composto da più moduli interagenti fra di loro

Introduzione (2)

- Classificazione elementare della memoria:
 - **interna**, se è direttamente accessibile dal processore
 - **esterna**, se è necessario anteporre un modulo di ingresso/uscita tra processore e memoria
- Poiché, attualmente, non vi è una tecnologia che soddisfi tutte i requisiti di un sistema di memoria “ideale” (basso costo, alta capacità, tempo di accesso ridotto), si usa un **sistema di memoria gerarchico**: il livello più alto, il più “vicino” al processore, è veloce e piccolo, il più basso è più lento e capiente.

Introduzione (Gerarchia di Memoria)

- In cima alla piramide ci sono i registri interni alla CPU.
- Come memoria interna, la cache e la memoria primaria
- Come memoria esterna, l'hard disk, altri livelli più bassi in cui si trovano le unità estraibili (dischi SSD estraibili, ecc.).
- Ogni anno la tecnologia evolve, ma resta ancora valido concetto di gerarchia



Caratteristiche di un sistema di memoria

I diversi tipi di memoria vengono classificati sulla base delle seguenti caratteristiche:

Location	Performance
Processor	Access time
Internal (main)	Cycle time
External (secondary)	Transfer rate
Capacity	Physical Type
Word size	Semiconductor
Number of words	Magnetic
Unit of Transfer	Optical
Word	Magneto-Optical
Block	Physical Characteristics
Access Method	Volatile/nonvolatile
Sequential	Erasable/nonerasable
Direct	Organization
Random	
Associative	

Collocazione della memoria

- Interna al processore
 - Registri
 - › si accede molto velocemente
 - › tipo di memoria **più costosa** e più piccola
- Interna
 - per eccellenza è la memoria principale
 - vi è anche la cache, memoria piccola e veloce, esterna alla CPU, usata come “appoggio” alla primaria
- Esterna (o secondaria)
 - consiste di tutti i dispositivi di memorizzazione accessibili dal processore tramite delle interfacce di I/O

Capacità

- Memoria interna
 - Tipicamente espressa in termini di byte (8 bit).
 - Le lunghezze più comuni di una parola ("word") sono 1, 2, 4, 8 byte.
 - Qualche volta i termini "parola" e "byte" vengono usati come sinonimi.
- Memoria esterna
 - Tipicamente espressa in termini di byte.

Caratteristiche del trasferimento dati

- Memoria interna: numero di bit letti o scritti in una volta (larghezza del bus)
 - Non è necessario che l' "unità" di trasferimento sia uguale alla parola o all'unità indirizzabile.
 - > Parola: unità "naturale" di rappresentazione del contenuto della memoria.
 - La lunghezza tipica è uguale al numero di bit usato per rappresentare un numero o un'istruzione; ci sono però parecchie eccezioni.
 - > Unità indirizzabile: spesso è la parola, ma alcuni sistemi consentono di indirizzare la memoria a livello di byte.
- Memoria esterna: si trasferiscono "blocchi", spesso molto più grandi di una singola parola.

Metodo di accesso (1)

- Sequenziale
 - L'accesso avviene secondo una specifica sequenza.
 - Il tempo di accesso dipende dal punto in cui il meccanismo di lettura/scrittura si trova e dal punto in cui si vuole arrivare.
 - Esempio «storico»: nastro magnetico.
- Diretto
 - I singoli blocchi di memoria hanno un unico indirizzo basato sulla posizione fisica.
 - Il meccanismo di lettura/scrittura si sposta vicino alla posizione richiesta, poi cerca in modo sequenziale.
 - Esempio: unità disco.

Metodo di accesso (2)

- Casuale («random»)
 - Il meccanismo di indirizzamento è direttamente collegato con le singole locazioni.
 - Il tempo di accesso è indipendente dalla posizione del dato richiesto.
 - Esempi: memoria principale, alcuni tipi di cache.
- Associativo
 - Accesso di tipo casuale per “contenuto”. Si accede al “blocco” di memoria con il contenuto desiderato.
 - Tempo di accesso costante, come per l’accesso casuale.
 - Esempio: memoria cache di tipo associativo.

Prestazioni (1)

- Tempo di accesso
 - Tempo che intercorre tra l'istante in cui un indirizzo è presentato alla memoria e l'istante in cui il dato viene memorizzato o reso disponibile.
 - Per la memoria ad accesso non casuale è il tempo necessario perché il meccanismo di lettura/scrittura si posizioni all'indirizzo richiesto.
- Tempo di ciclo di memoria
 - Tempo di accesso sommato al tempo necessario per rigenerare i dati prima di un secondo accesso, o sommato al tempo necessario per eseguire delle operazioni di "set up".

Prestazioni (2)

- Velocità di trasferimento (“transfer rate”)
 - velocità a cui i dati vengono trasferiti.
 - Per l’accesso casuale è pari all’inverso del tempo di ciclo.
 - Per l’accesso non casuale vale la seguente relazione:

$$T_N = T_A + N/R$$

- > T_N = tempo medio per leggere/scrivere N bit
- > T_A = tempo di accesso medio
- > N = numero di bit
- > R = “rate” di trasferimento (in bit per secondo, bps)

Tipi di memoria

- A semiconduttore («stato solido»)
 - RAM, ROM
- A tecnologia magnetica
 - hard disk
- A tecnologia ottica
 - DVD
- Magnetico-semiconduttore (dischi rigidi basati su tecnologia magnetica e a semiconduttore)
- La tendenza è sempre di più verso memorie a semiconduttore. Si usa spesso il termine «solid state», memorie a stato solido (Esempio: dischi SSD, Solid State Disk)

Caratteristiche fisiche

- Volatilità
 - Senza corrente elettrica l'informazione "decade".
 - Esempio di memoria volatile: memoria a semiconduttore.
 - Esempio di memoria non volatile: hard disk, vecchi CD.
- Cancellabilità
 - Ci sono memorie scrivibili una sola volta (PROM)
 - Esempio di memoria non cancellabile: ROM, vecchi CD
 - Esempio di memoria riscrivibile: RAM
- Consumo di potenza

Specifiche di progetto del sistema di memoria

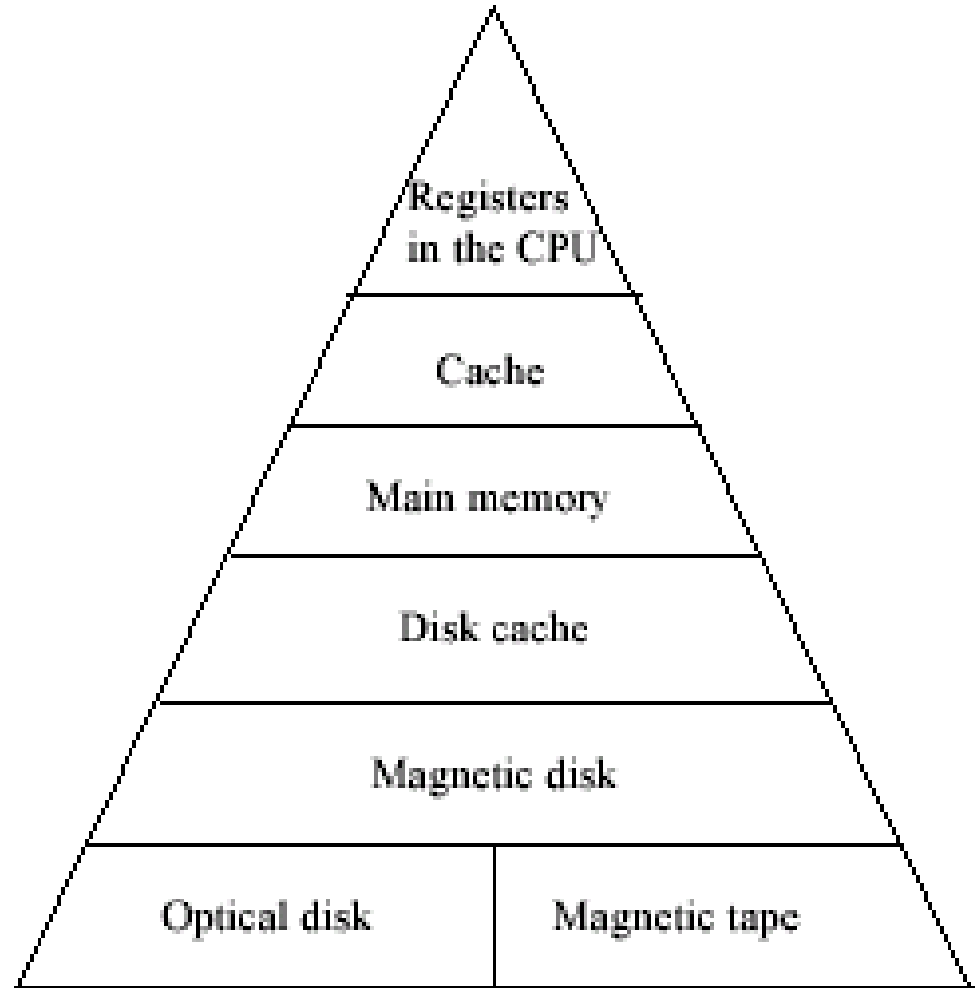
- Quanta memoria?
 - Si vorrebbe una capacità più alta possibile.
 - Quanto veloce?
 - Si vorrebbe raggiungere la velocità del processore, in modo che esso non abbia tempi di attesa.
 - Quanto costosa?
 - Il meno possibile
- Quindi la memoria dovrebbe essere grande, veloce, economica

Progetto del sistema di memoria

- Purtroppo allo stato attuale delle tecnologie utilizzabili valgono ancora queste considerazioni:
 - Maggior velocità (tempo accesso piccolo) implica maggior costo (per bit)
 - Maggior capacità implica minor costo
 - Maggior capacità implica minor velocità
- Il miglior compromesso ingegneristico viene cercato progettando un sistema di memoria **gerarchico**

Gerarchia di Memoria

- Il sistema gerarchico è utile se via via che si scende nella gerarchia è verificato che:
 1. diminuisce il costo per bit
 2. aumenta la capacità
 3. aumenta il tempo di accesso
 4. diminuisce la frequenza di accesso da parte del processore
- L'aspetto chiave è fare in modo che sia soddisfatto il più possibile il punto 4



Gerarchia di memoria

- Soluzione adottata per ottenere un **compromesso** tra costo, capacità e velocità.
- E' utile solo se vengono rispettate le seguenti condizioni via via che si scende nella gerarchia:
 - diminuisce il costo per bit
 - aumenta la capacità
 - aumenta il tempo di accesso
 - diminuisce la frequenza di accesso da parte del processore
- E' facile rendere vere le prime tre condizioni con una tecnologia appropriata.
- L'ultima si basa sul ***principio di località***.

Principio di località (1)

- Durante l'esecuzione di un programma, in un breve intervallo di tempo, è *probabile* che:
 - Si acceda a dati ed istruzioni contenuti in locazioni di memoria contigue ("cluster", gruppi, di locazioni)
 - > Es. Procedure
 - Vengano eseguite istruzioni contenute in locazioni *vicine* a quella dell'istruzione corrente
 - Queste istruzioni vengano *ripetute nel tempo*
 - > Esempio: i cicli ("loop").
- Si può quindi ipotizzare che gli accessi alla memoria coinvolgano nel tempo dei gruppi ("cluster") di locazioni

Principio di località (2)

- Ovviamente i “cluster” da utilizzare cambieranno nel corso dell’esecuzione del programma
- Ma il principio di località (“principle of locality of reference”) ipotizza che, in un breve lasso di tempo, la CPU dovrà accedere a ben determinati gruppi (“cluster”) di dati ed istruzioni
- Il punto chiave diventa quello di “organizzare” i dati lungo la gerarchia di memoria in modo che i “cluster” da usare in un certo istante si trovino ai livelli più alti della gerarchia
- E cambiare rapidamente i cluster memorizzati ai livelli alti, quando altri cluster diventano necessari. Si tratta quindi di far “scorrere” lungo la gerarchia i dati in modo che quelli che servono siano con alta probabilità ai livelli più alti (accessibili più velocemente)

Principio di località – Motivazioni (3)

1. I programmi hanno un flusso di esecuzione tipicamente **sequenziale**. Quindi la prossima istruzione da prelevare è quella “vicina”
2. E' raro avere molte chiamate a “procedure” nidificate in sequenza una dentro l'altra
3. Nei programmi si hanno spesso insiemi di istruzioni che vengono ripetuti per un certo tempo (“loop”)
4. Nei programmi si hanno spesso strutture dati “sequenziali” (vettori, matrici, ecc.)

Località Spaziale e Temporale

In letteratura si distingue a volte fra:

- Località Spaziale: è probabile che i dati/istruzioni che mi serviranno siano contigui a quelli che sto usando
- Località Temporale: è probabile che i dati/istruzioni che mi serviranno siano già in uso

Esempio (1)

Memoria a 2 livelli di gerarchia: M_1 e M_2

Caso tipico che vedremo: Cache + Memoria Primaria

Livello 1 contiene 1000 parole ed ha tempo di accesso pari a 0.1 micro secondi (Es. memoria “cache”)

Livello 2 contiene 100.000 parole ed ha tempo di accesso pari a 1 micro secondo (Es. memoria primaria)

Le parole vengono sempre “cercate” prima al Livello 1, se non le si trova si passa al Livello 2

Tutte le informazioni presenti in M_1 sono anche in M_2 , ma non viceversa (tipicamente capacità $M_1 \ll M_2$)

M_1 è più veloce e quindi più costosa di M_2

Esempio (2)

Quando l'informazione cercata non viene trovata in M_1 , allora viene presa da M_2 , ed inoltre si copia in M_1 un “blocco” di informazione contenente l'informazione cercata e quelle contigue

➤ L'operazione di ricopiare in M_1 un “blocco” contenente l'informazione cercata, e non trovata, serve per sfruttare il Principio di Località

Supponiamo ora che il 95% delle volte troviamo le parole nella memoria di Livello 1.

Allora il tempo di accesso medio ad una parola si può scrivere come:

$$0.95 * 0.1 \mu s + 0.05 (0.1 \mu s + 1 \mu s) = 0.15 \mu s$$

Tempo di Accesso Gerarchia a due Livelli

E' facile capire che, in generale, il tempo medio di accesso ad un sistema di memoria a due livelli si può scrivere come:

$$T_s = H T_1 + (1 - H) (T_1 + T_2) = T_1 + (1 - H) T_2$$

T_1 :tempo di accesso memoria di primo livello

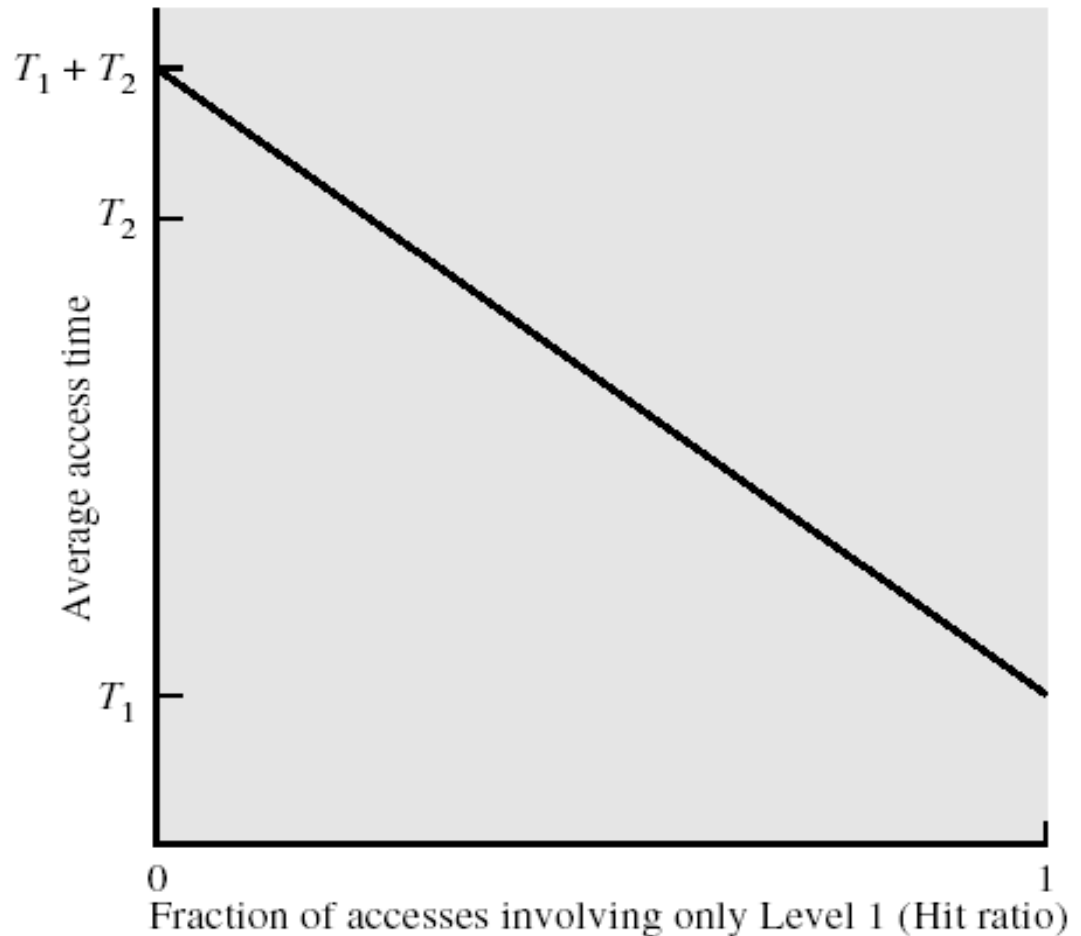
T_2 :tempo di accesso memoria di secondo livello

H : Hit ratio (percentuale di volte che il dato cercato è trovato nella memoria M_1 di primo livello)

N.B. Questa sarà la formula che useremo per il calcolo del tempo di accesso alla gerarchia cache-memoria primaria

Prestazioni Sistema Gerarchico a due livelli

- Hit ratio: percentuale degli accessi al sistema per cui la parola cercata viene trovata nella memoria di Livello 1
- T_1 : tempo di accesso alla memoria di Livello 1
- T_2 : tempo di accesso alla memoria di Livello 2



RAM (Random Access Memory)

- Memoria di lettura/scrittura a semiconduttore
 - La lettura e la scrittura avvengono tramite l'uso di segnali elettrici.
- Memoria volatile
- Può essere statica o dinamica (DRAM):
 - Dinamica
 - › L'informazione è data dalla presenza/assenza di carica nelle capacità.
 - › Richiede quindi un periodico "refresh".
 - Statica
 - › Uso di flip-flop tradizionali.
 - › Più veloce e costosa della RAM dinamica.

Memoria RAM di MacBook Pro, 13-inch, 2019

- 16 GB RAM (16 GB 2133 MHz LPDDR3 SDRAM)
- SDRAM: Synchronous Dynamic Random Access Memory
- Si tratta quindi di RAM «dinamiche» di tipo sincrono
- Tecnologia LPDD: Low-Power Double Data Rate
- Sono delle LPDDR SDRAM, tecnologia LP (low power), a basso consumo.

ROM (Read Only Memory)

- Memoria di sola lettura, non volatile
- Applicazioni:
 - Microprogrammazione
 - Programmi di sistema
 - Librerie per funzioni usate di frequente
- Vantaggi
 - Il contenuto della ROM è permanentemente in memoria principale.
- Svantaggi
 - Costo fisso per la fabbricazione di ogni chip.
 - Non sono ammessi errori, altrimenti la ROM è da cambiare.

Memorie «flash»

- La **memoria flash**, anche chiamata **flash memory**, è un tipo di memoria a stato solido e non volatile, che per le sue prestazioni può anche essere usata come memoria a lettura-scrittura.
- Grazie alla velocità e non-volatilità è usata frequentemente nelle fotocamere digitali, nei lettori di musica portatili, nei cellulari.
- E' spesso usata come un piccolo «hard disk» piuttosto che come RAM



Sponsorizzato ⓘ

Flash Drive Universale 128 GB, Chiavetta USB per Memoria Flash Esterna in Lega di Zinco USB 3.0 Compatibile per iPhone, iPad, iPod, Mac,...

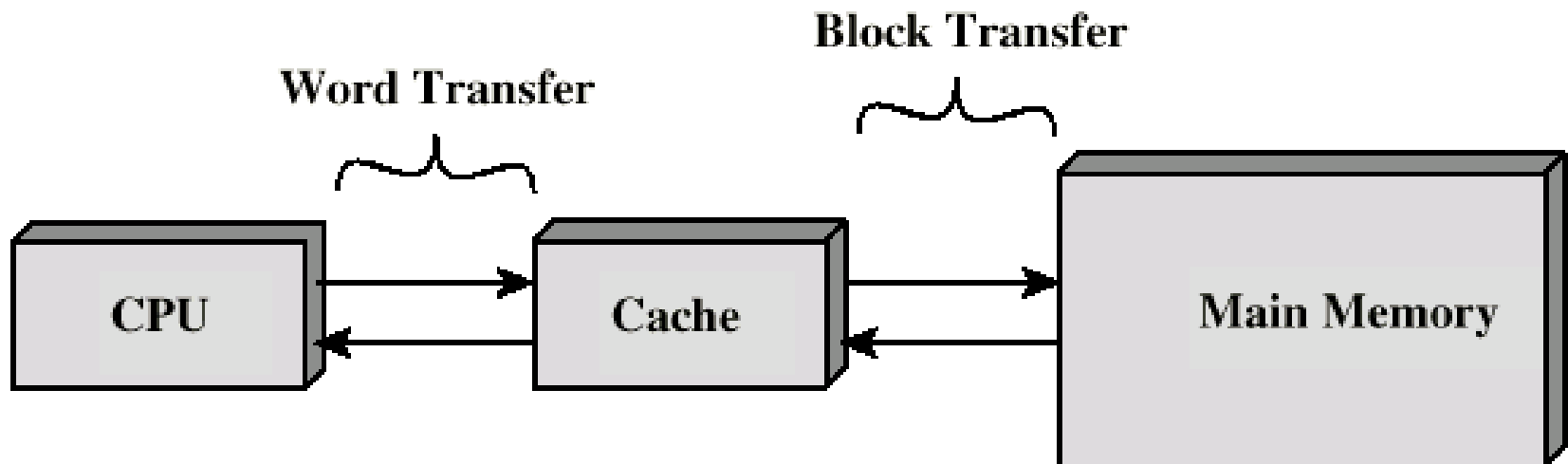
★★★★☆ ~ 658

49,49€

✓prime Spedizione GRATUITA giovedì 27 febbraio

Memoria Cache

- Usata per rendere veloce, grande e relativamente poco costosa la memoria vista dal processore.
- Contiene una copia di una **porzione** di memoria principale.
- Può essere posta all'interno della CPU o in un modulo a parte.

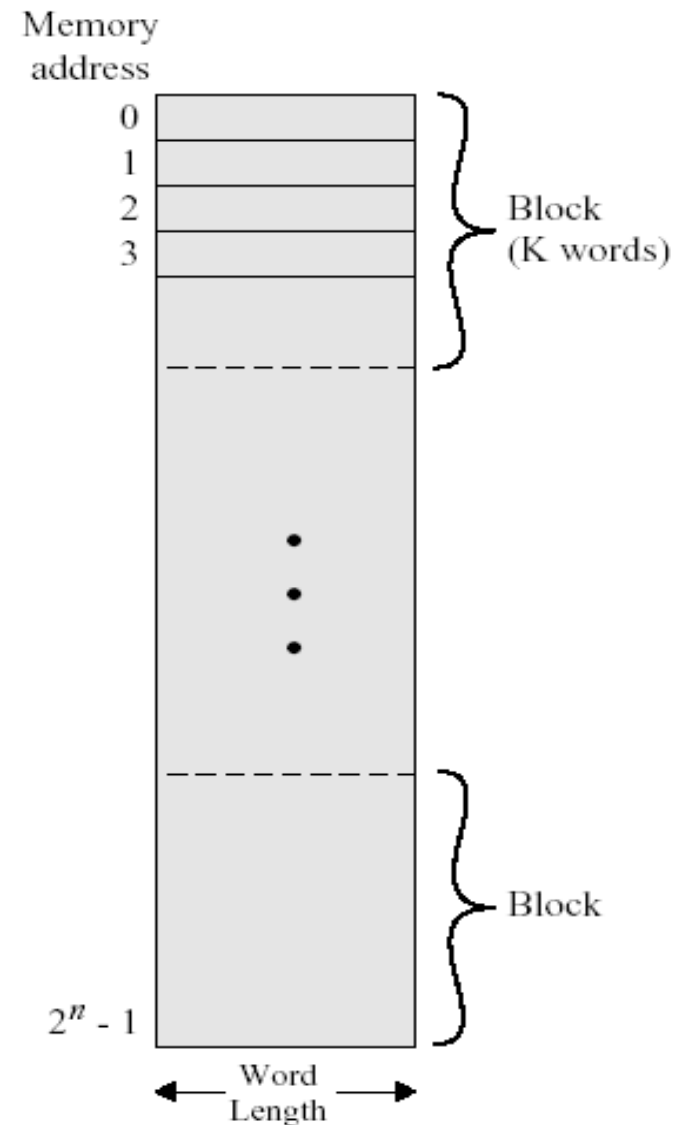


Struttura logica della memoria primaria

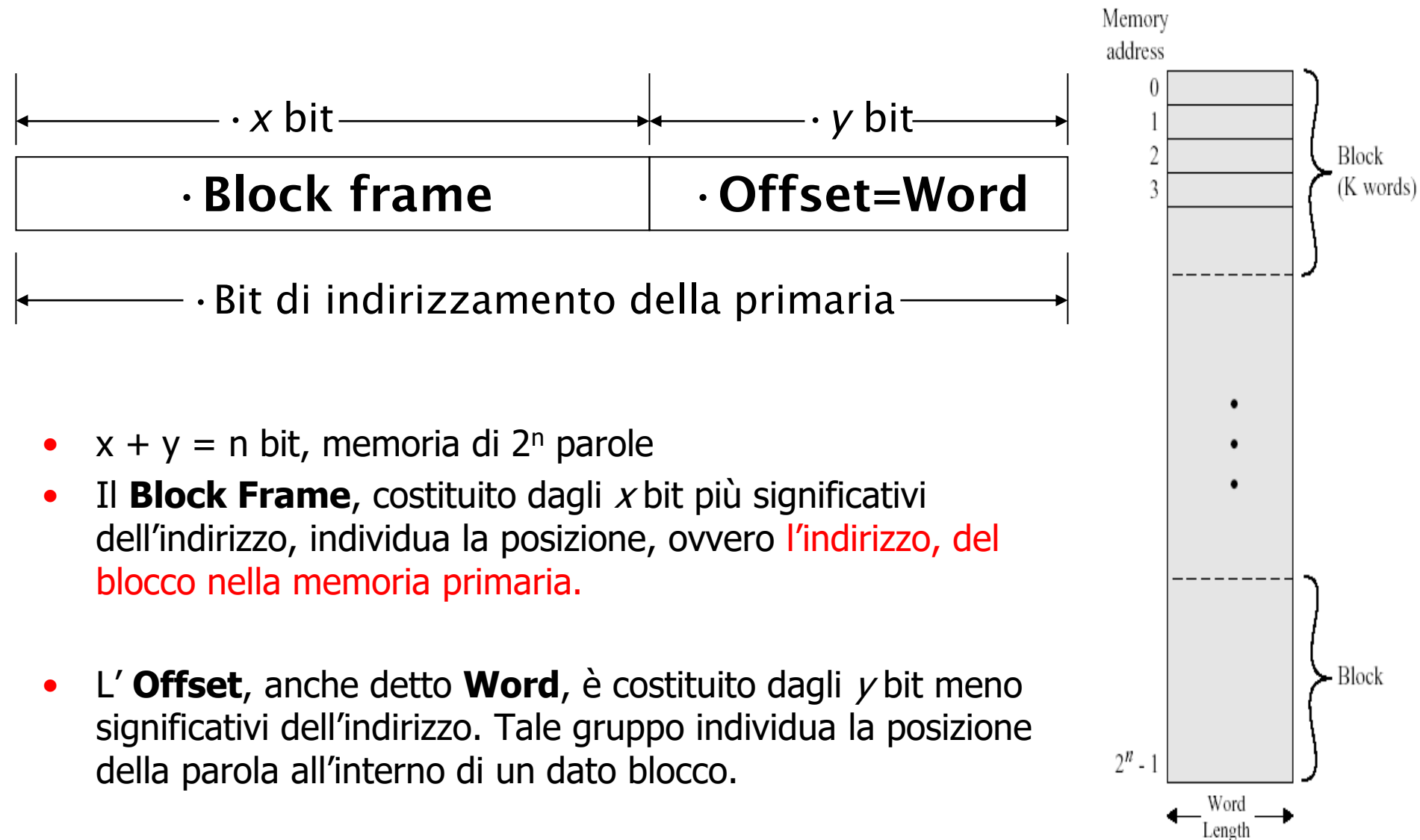
- E' costituita di 2^n parole indirizzabili, dove n è il numero di bit dell'indirizzo.
- E' suddivisa in blocchi di k parole (word).
- Quindi il numero totale di blocchi nella memoria principale è

$$M = 2^n/k$$

num. Blocchi = $2^{\text{num. Bit indir.}} / \text{num. Parole per blocco}$



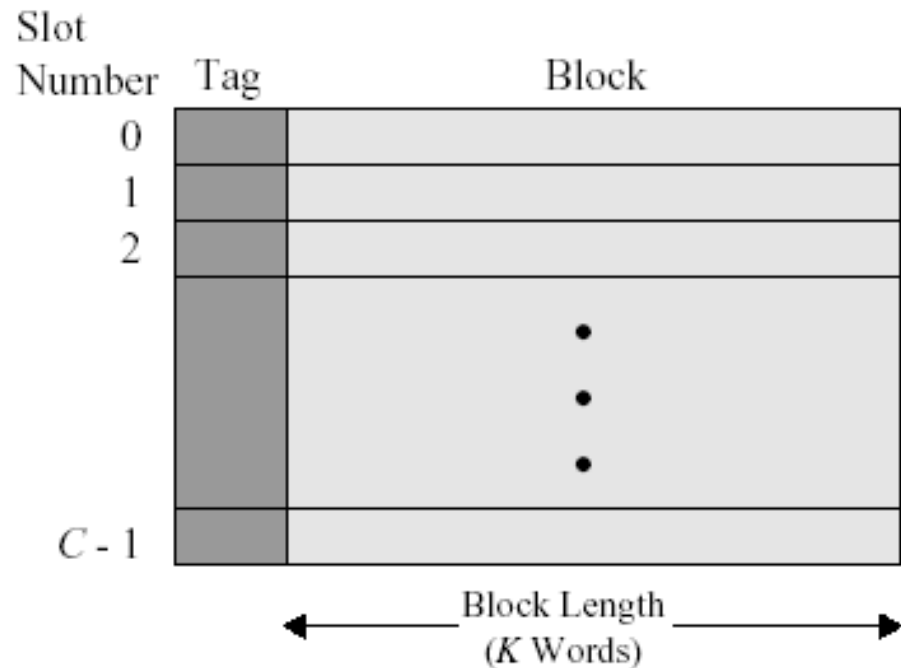
Indirizzamento della memoria primaria



- $x + y = n$ bit, memoria di 2^n parole
- Il **Block Frame**, costituito dagli x bit più significativi dell'indirizzo, individua la posizione, ovvero **l'indirizzo, del blocco nella memoria primaria**.
- L' **Offset**, anche detto **Word**, è costituito dagli y bit meno significativi dell'indirizzo. Tale gruppo individua la posizione della parola all'interno di un dato blocco.

Struttura della cache

- E' costituita di C linee.
- Ciascuna linea può contenere un blocco della memoria principale. Un blocco è un insieme di parole.
- Il numero di linee è molto minore del numero di blocchi della memoria principale !
- L'informazione sta nel "blocco". Il "tag" serve per l'indirizzamento della cache (vedi dopo)



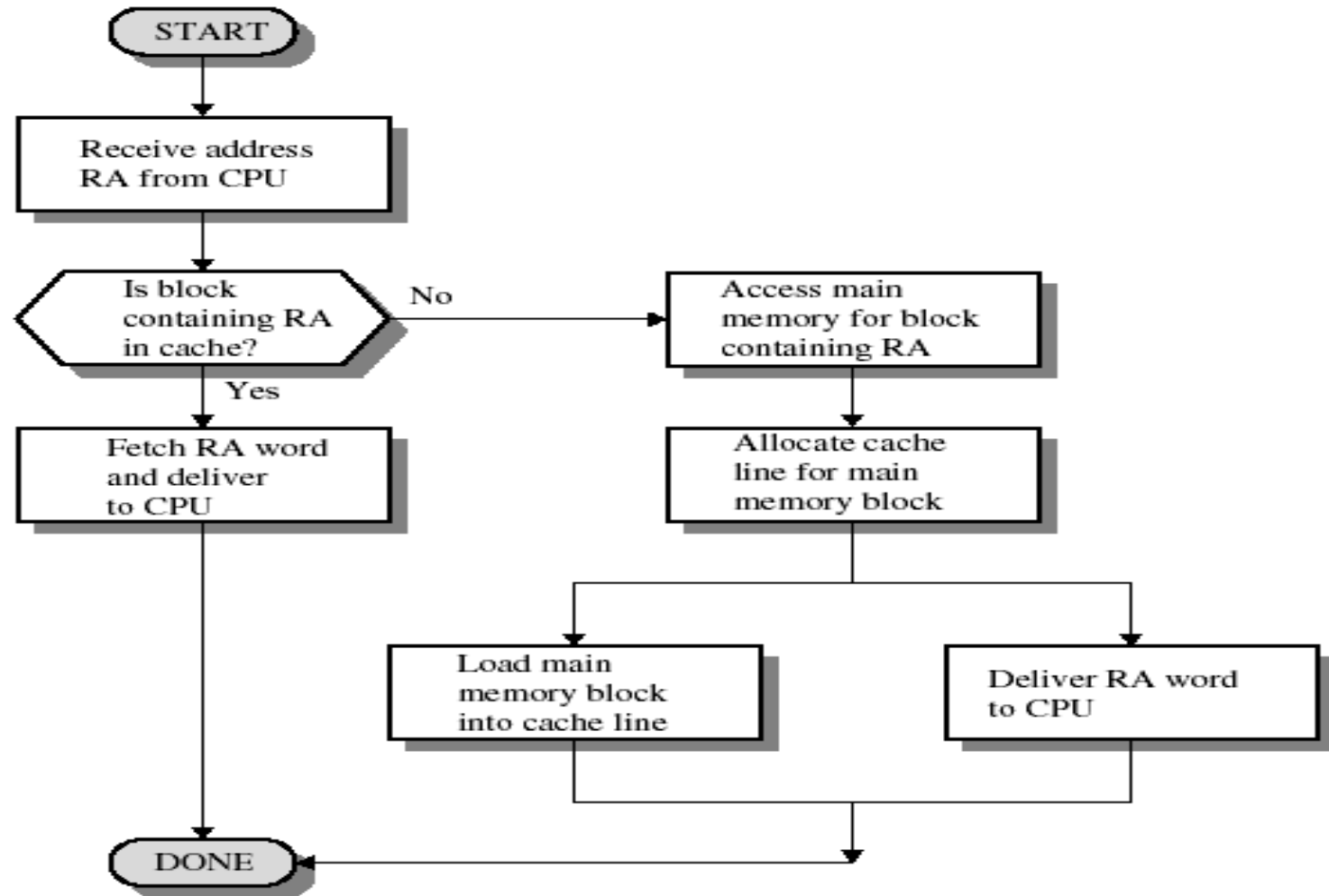
Concetti chiave

- Cache piccola e veloce. In un certo intervallo di tempo, può contenere solo un piccolo sotto insieme dei blocchi di primaria
- Poiché ci sono più blocchi di primaria che linee di cache, una certa linea non può essere assegnata in modo univoco ad un unico blocco di primaria
- Il “tag” identifica il blocco che in un certo istante è presente in una linea di cache. *Vediamo nelle pagine seguenti come è fatto il “tag” (etichetta)*
- Voglio fare in modo che nella maggior parte degli accessi del processore alla memoria la parola cercata sia già in cache
- Parleremo di cache “hit” e cache “miss”

Funzionamento di principio della cache

- La CPU richiede una parola di memoria primaria inviando **l'indirizzo** che la parola ha **in primaria**
- Si controlla se la parola è presente in cache.
- Se è presente, si prende la parola dalla cache.
 - Ciò avviene molto velocemente, perchè la cache è una memoria con basso tempo di accesso.
- Se non è presente, si copia dalla memoria principale in cache il blocco a cui appartiene la parola.
- **Si copia un blocco, e non una sola parola, per sfruttare il Principio di Località**
- Poi si trasferisce la parola dalla cache alla CPU.
- La cache include un campo, detto “tag”, per identificare quali blocchi della memoria principale sono presenti nelle sue “linee” in un certo istante.

Algoritmo di principio per lettura cache



Progettazione della cache

Vi sono diverse implementazioni, distinguibili tramite i seguenti elementi chiave, che descriviamo di seguito in questo capitolo.

Cache Size	Write Policy
Mapping Function	Write through
Direct	Write back
Associative	Write once
Set Associative	Line Size
Replacement Algorithm	Number of caches
Least recently used (LRU)	Single or two level
First in first out (FIFO)	Unified or split
Least frequently used (LFU)	
Random	

Grandezza della cache

- Si cerca il miglior compromesso:
 - Costo
 - › Si vorrebbe una cache piccola per avere basso costo e basso tempo di accesso
 - Velocità
 - › Si vorrebbe una cache grande perché possa contenere più blocchi di memoria primaria e si verifichi più spesso un “hit”.
 - › Tuttavia “cercare” i blocchi in cache richiede più tempo se essa è più grande.
- Valori tipici: **si va dai KB ai MB.**
- Grandezza cache dipende ovviamente da quanto il “programma” rispetta il Principio di Località. Non esiste quindi un valore assoluto ottimale.

Indirizzamento della cache

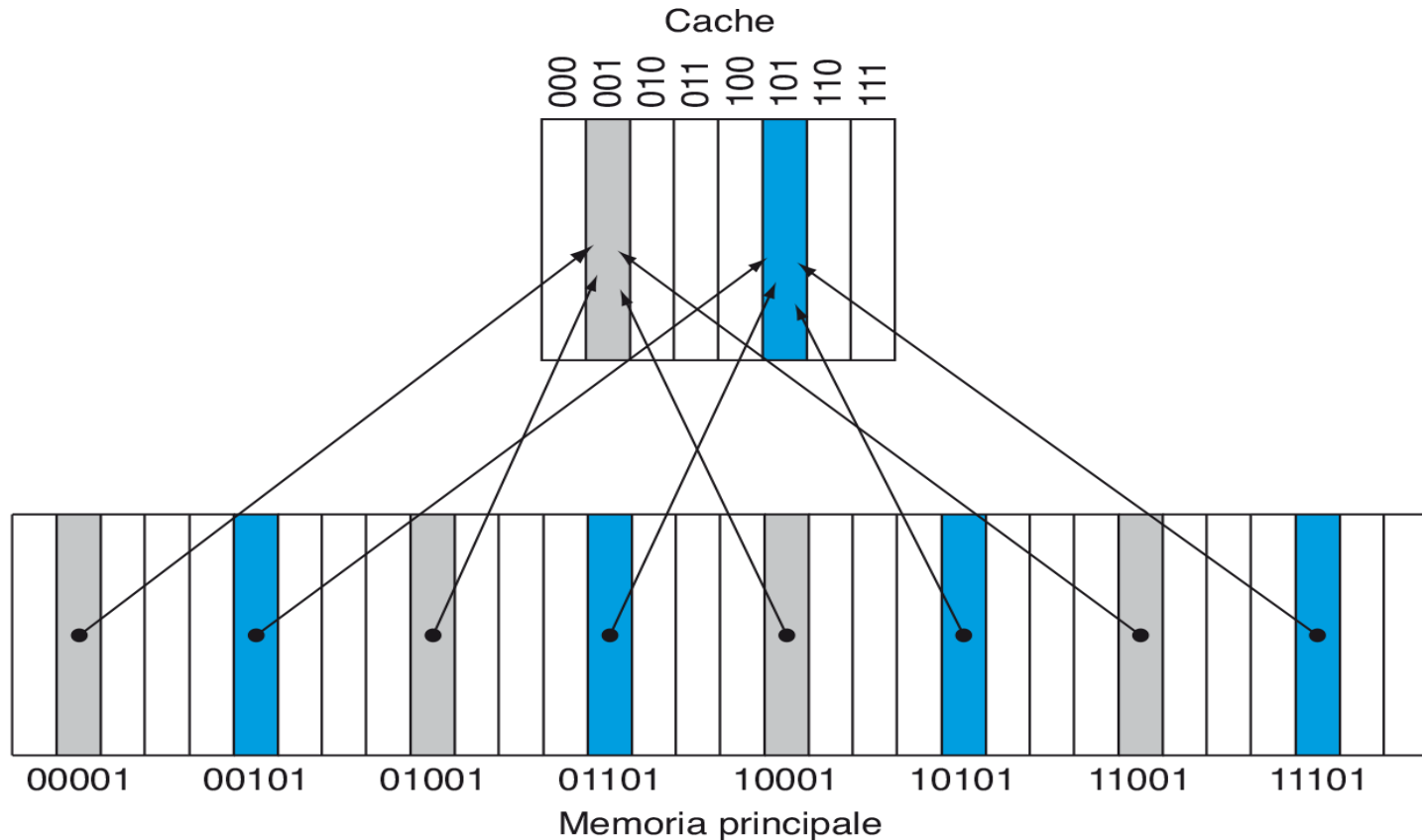
- Presenta una certa complessità perchè vi sono più blocchi di memoria primaria che linee=blocchi di cache.
 - Indirizzamento = Determinare quale blocco è presente in una certa linea di cache in un certo istante.
- Vengono usati tre metodi di indirizzamento:
 - diretto
 - associativo
 - associativo su insiemi

Metodo diretto

- Ogni blocco della memoria primaria può occupare solo una certa linea=blocco di cache.
- Questa organizzazione della cache viene detta a **mappatura diretta** ("direct mapped cache")
- Come si realizza la mappatura diretta ? Come cioè si calcola l'indirizzo del blocco di cache nel quale è "mappato" un certo blocco di memoria primaria ?

Metodo diretto: calcolo indirizzo

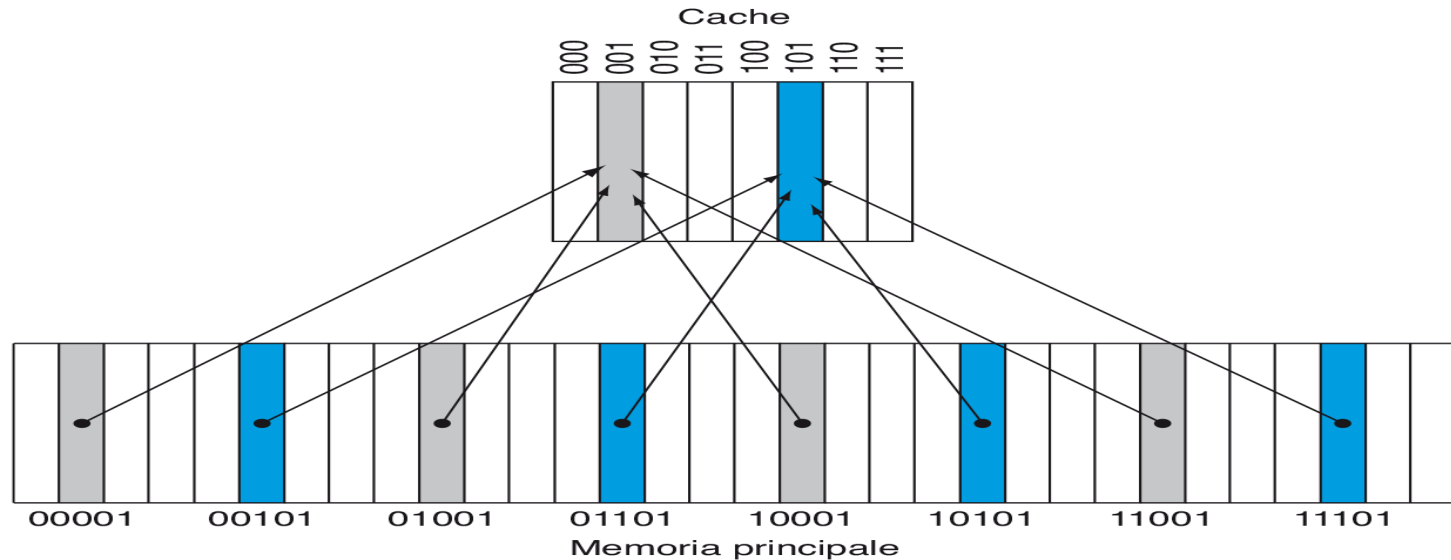
Patterson, D.A., Hennessy, J.L., "Struttura e progetto dei calcolatori elettronici", Zanichelli editore, 2015.



- Cache con 8 blocchi, memoria principale con 32 blocchi

Metodo diretto: calcolo indirizzo

Patterson, D.A., Hennessy, J.L., "Struttura e progetto dei calcolatori elettronici", Zanichelli editore, 2015.

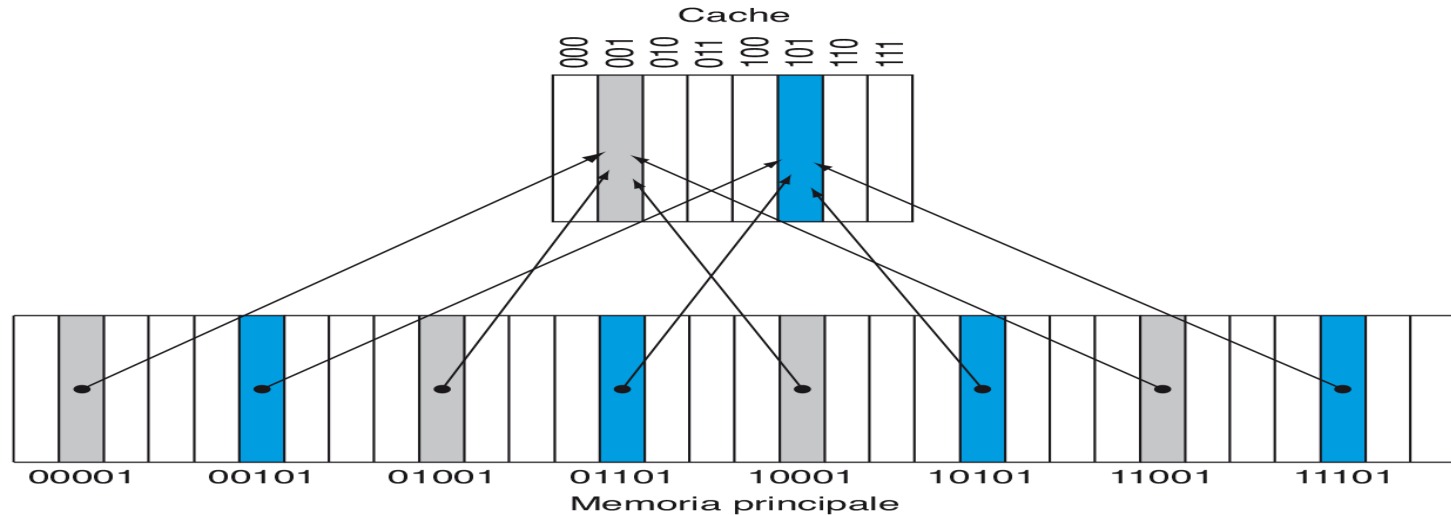


Se i è l'indirizzo della linea=blocco della cache, j l'indirizzo del blocco di memoria e m il numero di linee della cache. L'allocazione di un blocco di memoria primaria in cache segue la semplice legge:

$$i = j \text{ modulo } m$$

Metodo diretto: ruolo del “tag” (1)

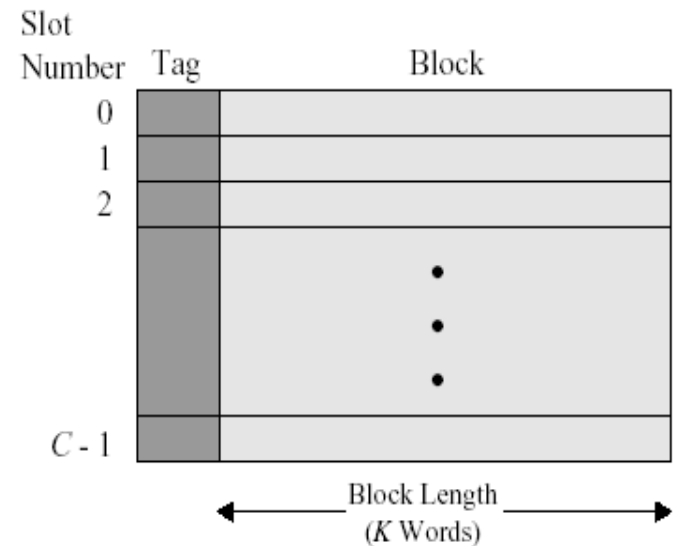
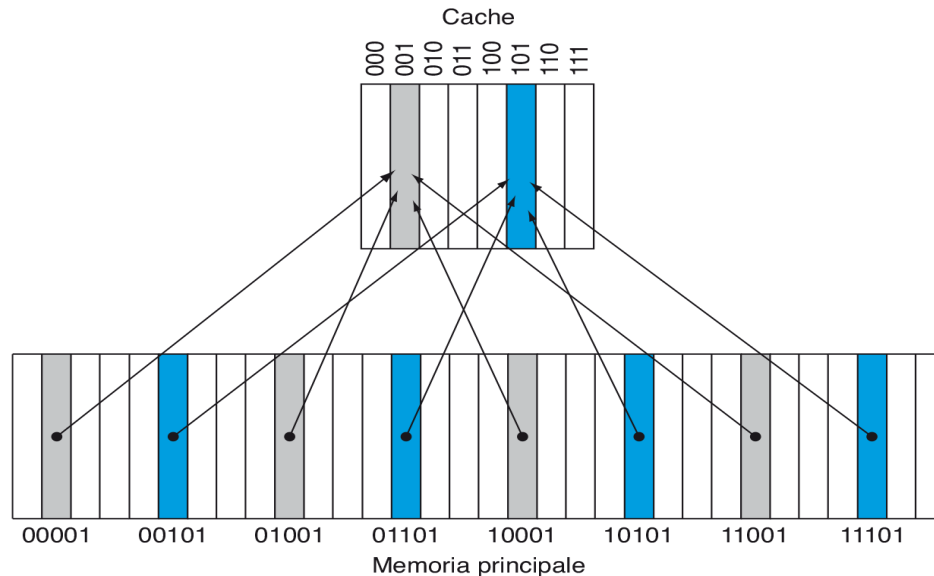
Patterson, D.A., Hennessy, J.L., "Struttura e progetto dei calcolatori elettronici", Zanichelli editore, 2015.



Poiché ogni blocco di cache può contenere più blocchi di memoria primaria, come si capisce se il blocco di cache “indirizzato” contiene il blocco di primaria, e quindi la parola, cercata ?

La risposta sta nel concetto di “tag”

Metodo diretto: ruolo del “tag” (2)



Diversi blocchi di primaria che sono mappati nello stesso blocco di cache sono distinti per mezzo di una diversa “etichetta” (“tag”)

Come si definisce e calcola il “tag” ?

Metodo diretto: formato indirizzo

Tag (t bit)	Linea (Cache Index) (r bit)	Word (w bit)
-------------	-----------------------------	--------------

Cache Index = indirizzo-blocco-primaria **modulo** numero-linee-cache

N.B. gli $n=t+r+w$ bit sono i bit totali di indirizzamento della memoria

- I w bit meno significativi identificano la "word" (al limite un byte) cercata all'interno del blocco
- Gli s bit (r+t bit) più significativi specificano un blocco di memoria e, a loro volta, sono divisi in:
 - r bit meno significativi che indirizzano la linea di cache (vengono a volte detti "cache index").
 - t bit più significativi che rappresentano il "**tag**" (etichetta).

Indirizzamento della cache

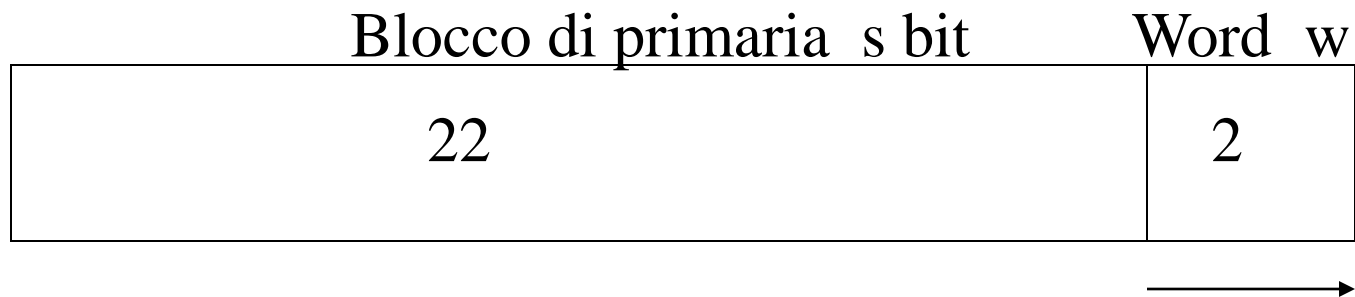
- A titolo di semplice esempio facciamo ora riferimento ad un sistema cache-primaria con le seguenti caratteristiche:
 - Cache di 64 KByte
 - Linee (Blocchi) di cache da 4 byte
 - > La cache contiene 16K (2^{14}) blocchi da 4 byte
 - Memoria primaria da 16 Mbyte
 - > Quindi si avranno 4M blocchi da 4 byte nella primaria
 - Indirizzamento della primaria a 24 bit (sono gli $n=t+r+w$ bit della trasparenza precedente)
 - > ($2^{24}=16M$)
 - Stiamo indirizzando il singolo byte (parola=byte)

Metodo diretto: formato dell'indirizzo

Tag t	Linea (Cache Index) r	Word w
8	14	2

- indirizzo complessivo da 24 bit (indirizzamento primaria)
 - 2 bit per indirizzare il byte all'interno del blocco (blocchi da 4 byte)
 - identificatore del blocco di cache da 22 bit
 - tag da 8 bit ($22-14=8$)
 - Indirizzo linea di cache da 14 bit (la cache ha 16K (2^{14}) linee da 4 byte)
 - Ricordiamo che non possono ovviamente essere memorizzati due blocchi nella stessa linea di cache.
- Ricerca dei dati nella cache sarà fatta trovando la linea e confrontando i tag (vedi trasparenze successive).

Metodo diretto: calcoli utili (1)



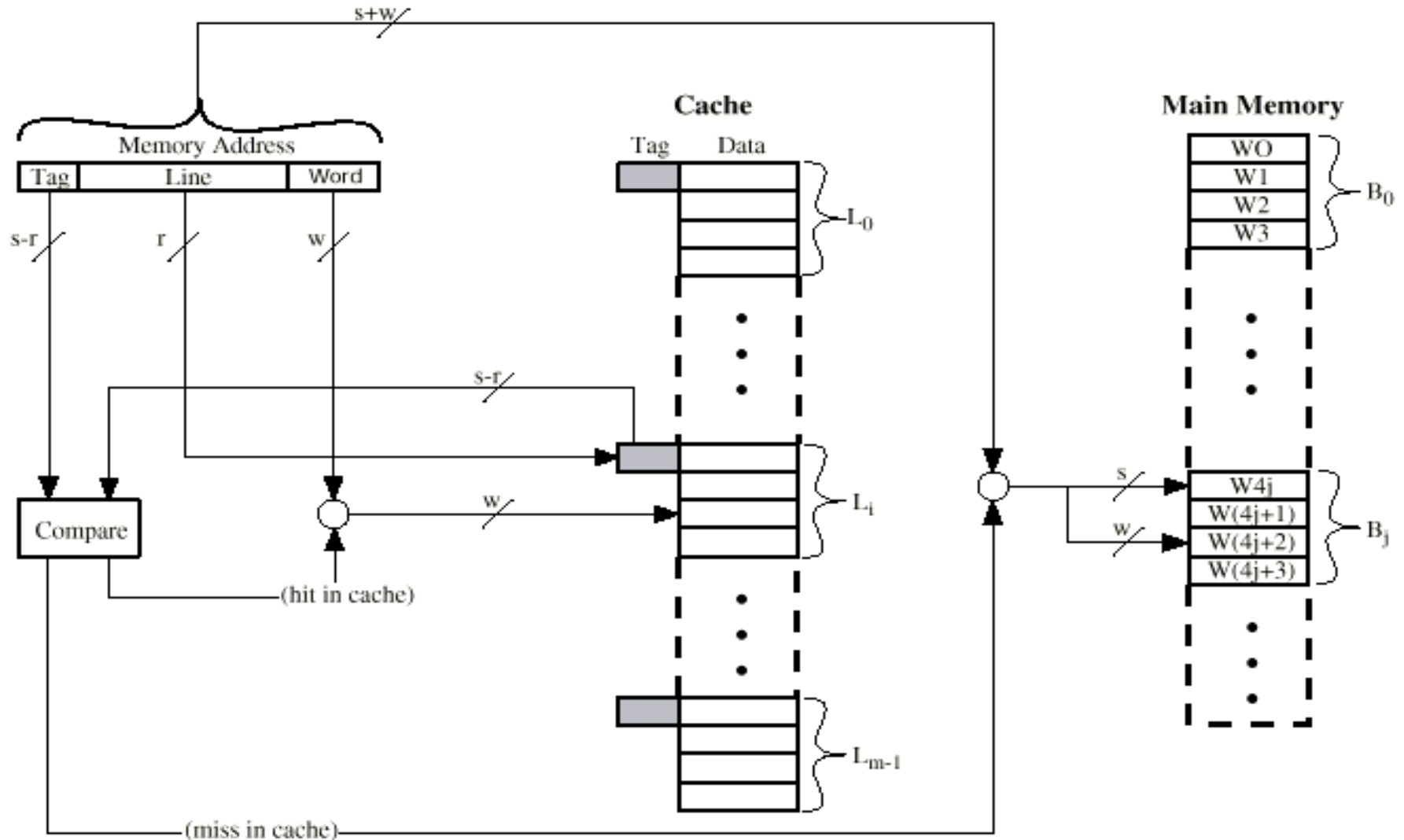
- Dato l'indirizzo a 24 bit di una parola in memoria principale, si determina l'indirizzo del blocco di primaria (s bit) facendo scorrere l'indirizzo a destra di w posizioni, cioè tante quanti sono i bit di identificazione della parola (cioè si divide per 2^w).
 - indirizzo_blocco = $\text{Int}(\text{indirizzo_complessivo} / 2^w)$
 - indirizzo_word = $\text{Mod}(\text{indirizzo_complessivo} / 2^w)$

Metodo diretto: calcoli utili (2)

Tag s-r	Linea r
8	14

- Dato l'indirizzo di blocco di memoria primaria di cui alla trasparenza precedente (s bit), per determinare il tag e l'indirizzo della linea di cache si fa scorrere l'indirizzo del blocco a destra di r posizioni, cioè si divide per 2^r (numero linee della cache).
 - indirizzo_tag = $\text{Int}(\text{indirizzo_blocco} / 2^r)$
 - indirizzo_linea = $\text{Mod}(\text{indirizzo_blocco} / 2^r)$

Metodo diretto: Funzionamento concettuale



Metodo diretto: Mappatura fra Cache e Memoria Primaria

Se i è l'indirizzo della linea della cache, j l'indirizzo del blocco di memoria e m il numero di linee della cache. L'allocazione di un blocco di memoria primaria in cache segue la legge:

$$i = j \text{ modulo } m$$

Nel nostro esempio con $m=16K=2^{14}$, $i = j \text{ modulo } 2^{14}$

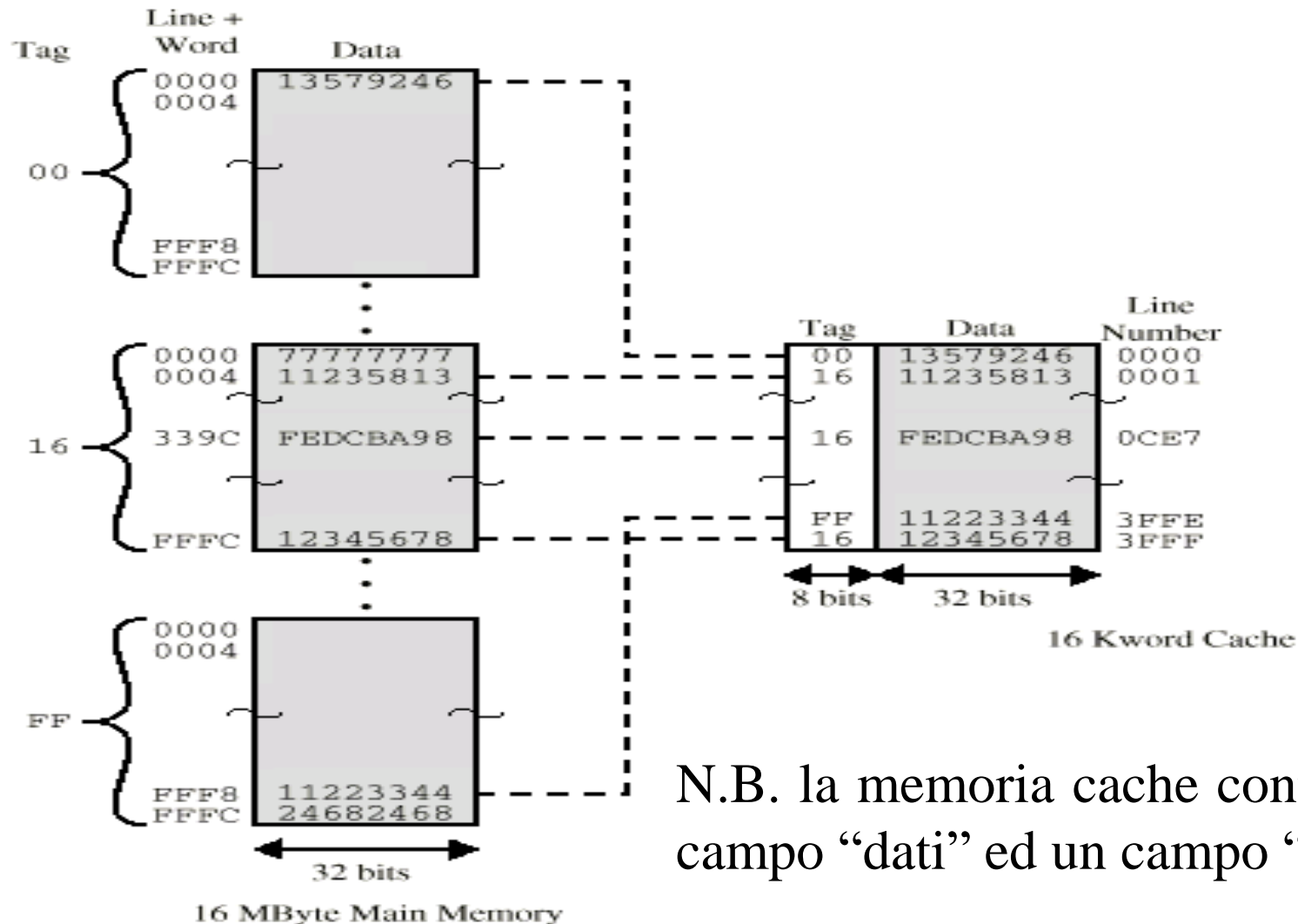
Linea di Cache	Indirizzo di inizio del blocco di memoria primaria allocato nella linea di cache
0	000000, 010000,.....,FF0000
1	000004, 010004,.....,FF0004
· · m-1	00FFFC, 01FFFC,.....,FFFFFC

Metodo diretto: Mappatura fra Cache e Memoria Primaria

- linea cache 0: dopo blocco di primaria con indirizzo esadecimale di **inizio** 000000 abbiamo blocco con indirizzo di **inizio** esadecimale 010000 che corrisponde al blocco $m=16K=2^{14}= 00\ 0100\ 000$. Il blocco $m=16K=2^{14}$ inizia all'indirizzo 010000 perché i blocchi contengono 4 byte= 2^2
- $m=2^{14}= 00\ 0100\ 000 \times 2^2 = 0\ 0001\ 0\ 0\ 0\ 0$

Linea di Cache	Indirizzo di inizio del blocco di memoria primaria allocato nella linea di cache
0	000000, 010000,.....,FF0000
1	000004, 010004,.....,FF0004
▪ ▪ m-1	00FFFC, 01FFFC,.....,FFFFFC

Metodo diretto: esempio



N.B. la memoria cache contiene un campo “dati” ed un campo “tag”

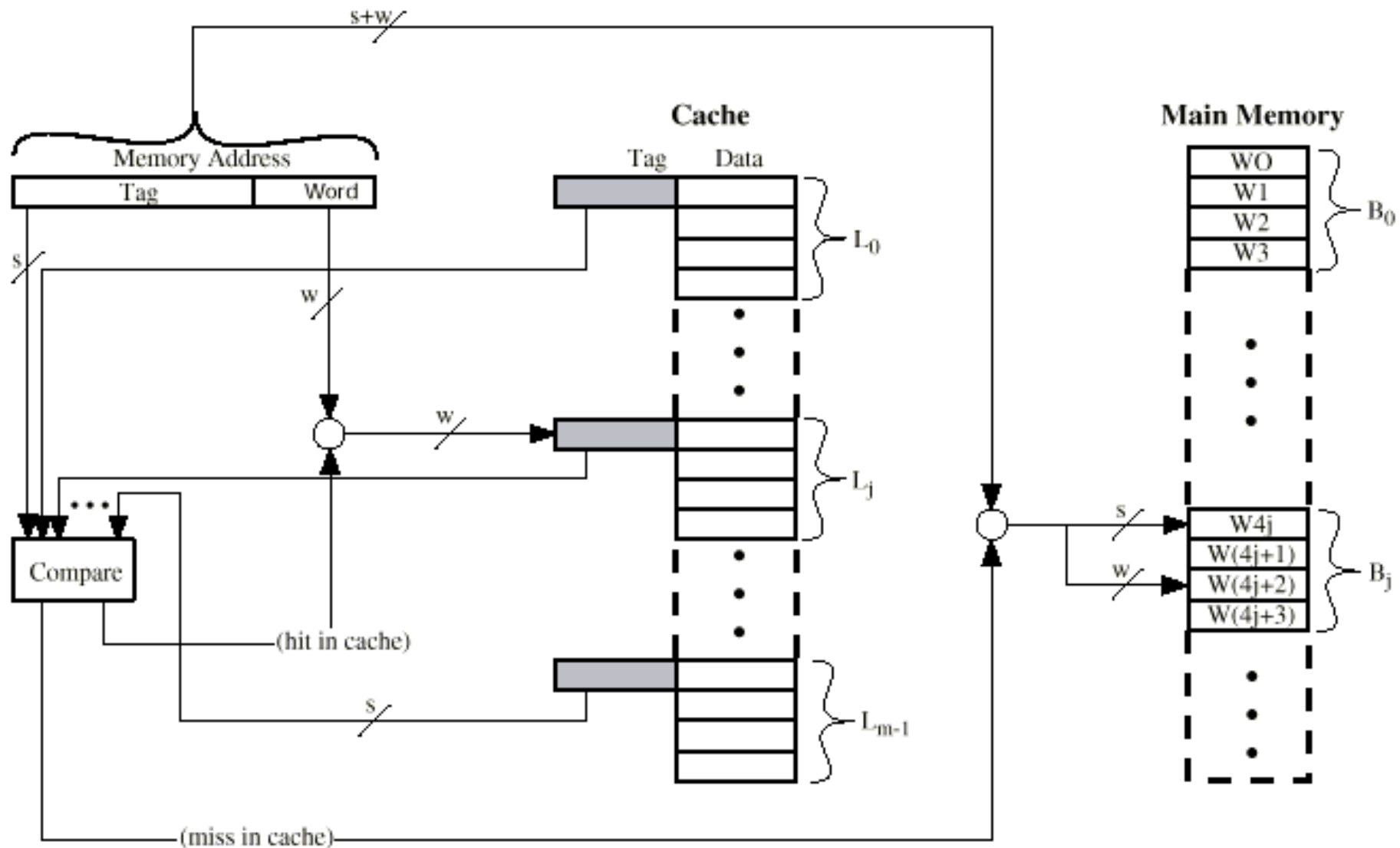
Metodo diretto: vantaggi e svantaggi

- Vantaggi
 - Semplice
 - Poco costoso da implementare
- Principale svantaggio
 - Locazione fissa per un dato blocco
 - Se vengono richiesti alternativamente due blocchi mappati nella stessa linea di cache, questi vengono continuamente scambiati e riducono notevolmente l' "hit ratio".

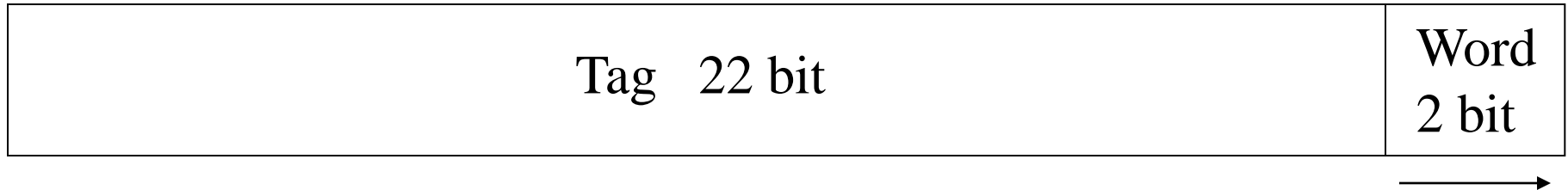
Metodo associativo

- Un blocco della memoria principale può essere caricato all'interno di qualsiasi linea di cache.
- L'indirizzo di memoria principale è interpretato come "tag" e "word" (non c'è cache index).
- Il tag identifica un blocco di memoria primaria.
- Ciascun campo tag nella cache viene esaminato per un confronto al fine di individuare il blocco di memoria cercato.
- La ricerca in cache diventa più complessa, dovendo effettuare dei confronti multipli.

Metodo associativo: Funzionamento

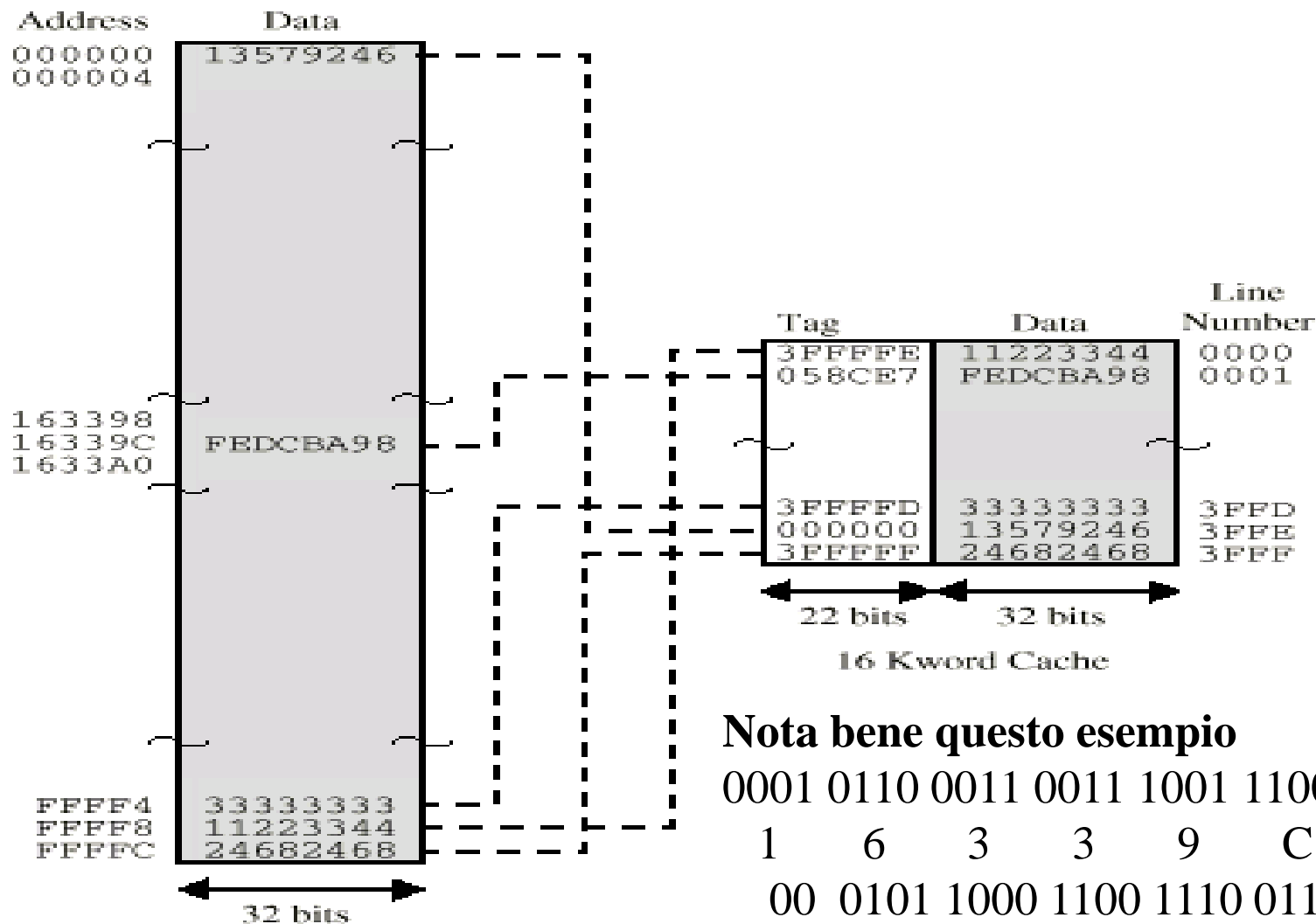


Metodo associativo: formato dell'indirizzo



- Il tag da 22 bit deve essere memorizzato in cache per ogni linea di dati (da 4 byte).
- Si confrontano il campo del tag di tutte le linee con il tag in entrata (“indirizzo”) nella cache.
- I 2 bit meno significativi dell’indirizzo identificano quale parola da 1 byte è richiesta nel blocco di dati da 4 byte.
- Per ottenere il tag si fa scorrere l’indirizzo verso destra di 2 posizioni, cioè del numero di bit necessari per indirizzare una parola; quindi si divide per 2^2 . In generale:
 - $\text{tag} = \text{Int}(\text{indirizzo_complessivo} / 2^w)$
 - $\text{word} = \text{Mod}(\text{indirizzo_complessivo} / 2^w)$

Metodo associativo: esempio



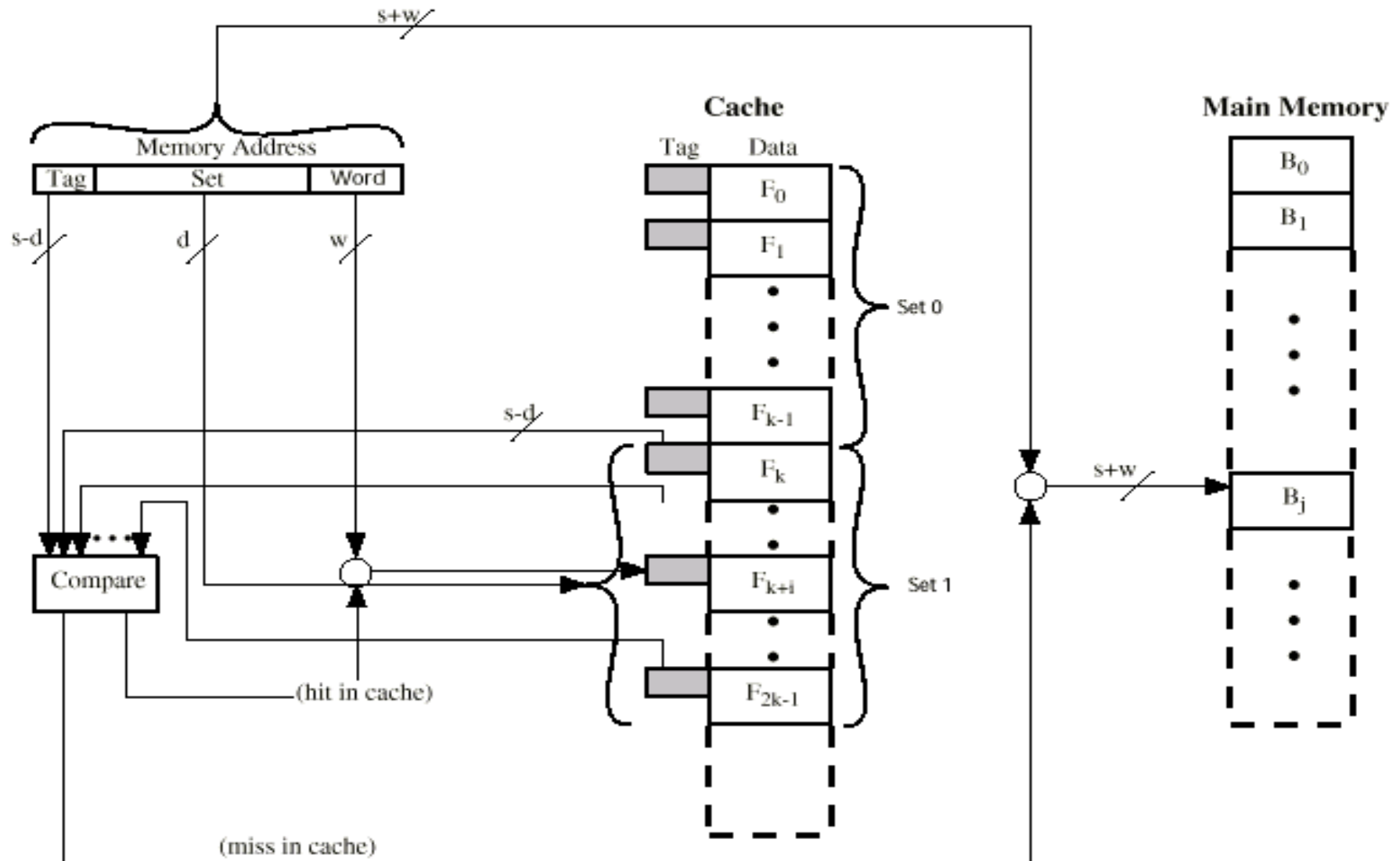
Metodo associativo: vantaggi e svantaggi

- Principale vantaggio
 - flessibilità nell'allocazione dei blocchi in cache
- Principale svantaggio
 - complessità del «circuitto» richiesto per esaminare i “tag” di tutte le linee di cache in parallelo

Metodo associativo su insiemi

- E' un compromesso tra i due metodi precedenti.
 - Non si è obbligati ad allocare un blocco in una sola linea di cache e non si deve confrontare tutti i campi tag.
- La cache è divisa in v insiemi ("set").
- Ogni set è composto da k linee.
- Un dato blocco può essere mappato in qualsiasi linea all'interno di un set.
 - Es.: il blocco B può stare in qualsiasi linea del set i .
- Situazione tipica: 2 linee per set
 - 2 possibili mappature associative.
 - Un dato blocco può essere in una delle 2 linee di un solo set.

Metodo associativo su insiemi da K linee: Funzionamento di principio



Metodo associativo su insiemi da 2 linee: formato dell'indirizzo

Tag s-r	Set r	Word w
9	13	2

- Il campo del set identifica l'insieme di cache su cui procedere.
- Si confronta poi il campo “tag” per verificare se il blocco richiesto è in quell'insieme (set).

Metodo associativo su insiemi da 2 linee: formato dell'indirizzo

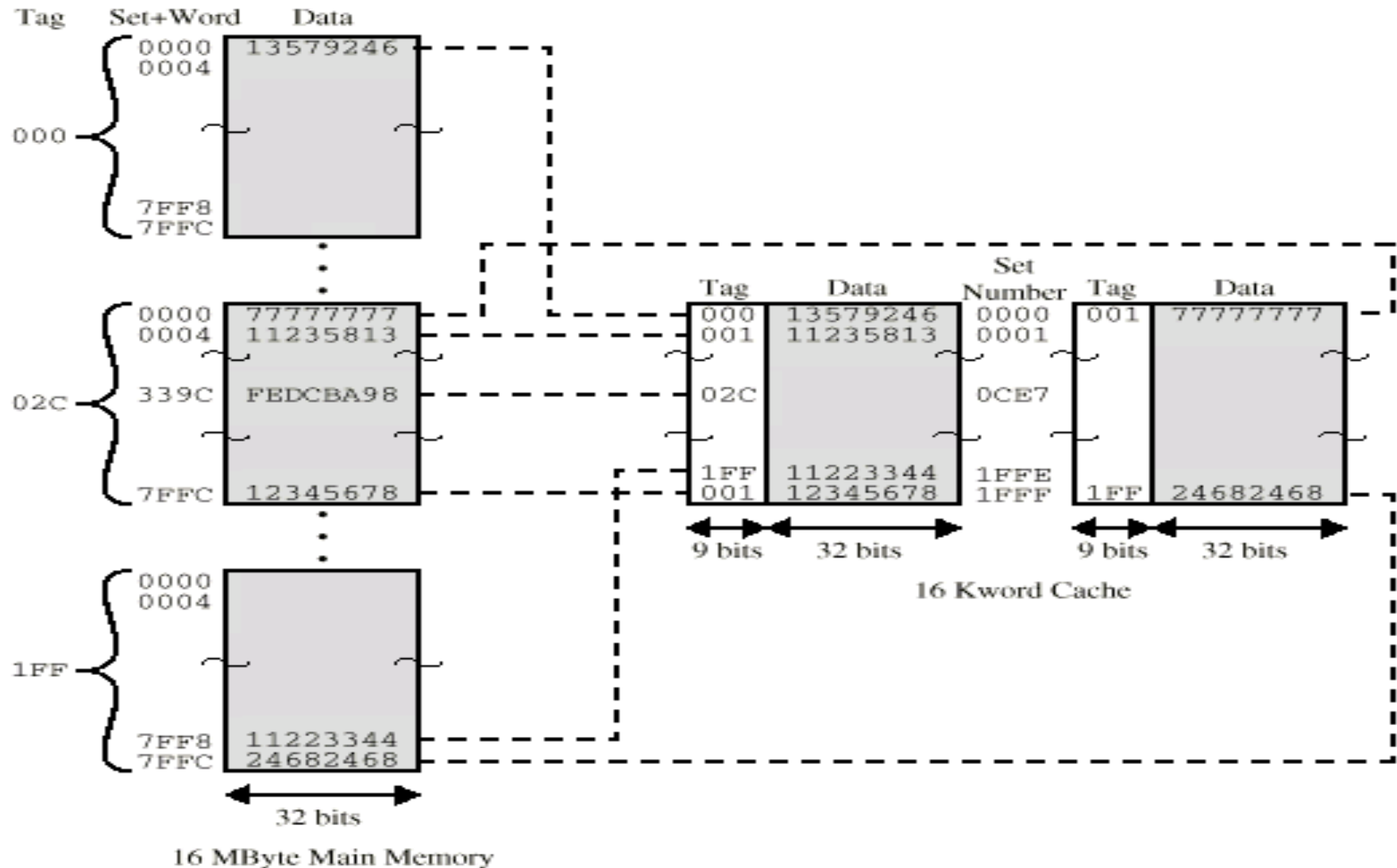
Tag s-r	Set r	Word w
9	13	2

- Esempio

Indirizzo	Tag	Dato	Indirizzo del Set
—1FF 7FFC	1FF	12345678	1FFF
—001 7FFC	001	11223344	1FFF

- Indirizzo set+word = 7FFC, da cui si può ricavare indirizzo "set" dividendo per 4 (dimensione blocchi cache), che corrisponde ad uno scalamento verso destra di 2 posizioni
- Per cui 0111 1111 1111 1100 diventa 01 1111 1111 1111=1FFF
- Notare che i due blocchi hanno stesso set index ma diverso tag

Metodo associativo su insiemi di 2 linee: esempio



Metodo associativo su insiemi da 2 linee: relazioni utili (1)


Blocco s	Word w
22	2



- Per determinare l'indirizzo del blocco di memoria si deve traslare l'indirizzo complessivo di tante posizioni quanti sono i bit necessari per identificare una parola; ciò corrisponde a dividere per 2^w .
 - indirizzo_blocco = $\text{Int}(\text{indirizzo_complessivo}/2^w)$
 - indirizzo_word = $\text{Mod}(\text{indirizzo_complessivo}/2^w)$

Metodo associativo su insiemi da 2 linee: relazioni utili (2)

Tag s-r	Insieme r
9	13



- Per determinare il tag si fa scorrere l'indirizzo a destra del numero di bit necessario per rappresentare gli insiemi, cioè si divide per 2^r .
 - tag = $\text{Int}(\text{indirizzo_blocco} / 2^r)$
 - indirizzo_insieme = $\text{Mod}(\text{indirizzo_blocco} / 2^r)$

Algoritmi di rimpiazzamento (1)

- Utilizzati nel caso di cache “piena”
 - se deve essere caricato un ulteriore blocco, se ne deve scegliere uno da eliminare dalla cache.
- Metodo diretto
 - La scelta è obbligata poiché ogni blocco può essere allocato solo in una certa linea di cache.
 - Si rimpiazza il blocco presente nella linea corrispondente con il nuovo in ingresso alla cache.
- Nel caso dei metodi associativo ed associativo su insiemi la politica di rimpiazzamento è più flessibile (vedi dopo)

Algoritmi di rimpiazzamento: metodi associativo e assoc. su insiemi

- Perché sia più veloce, l'algoritmo è implementato in hardware.
- Least Recently Used (LRU)
 - Viene rimpiazzato il blocco meno usato di recente.
 - Facile implementazione con un set da due linee
 - > Ogni linea ha un bit che viene settato a 1 quando la linea è referenziata.
 - > Al momento del rimpiazzamento si prende la linea con questo bit a zero.
 - E' il più efficiente
- First In First Out (FIFO)
 - Si rimpiazza il blocco che è presente in cache da più tempo
- Least Frequently Used (LFU)
 - Si rimpiazza il blocco che è stato usato meno frequentemente
- Scelta a caso

Politica di scrittura

- Non si deve sovrascrivere un blocco presente in cache prima di averlo aggiornato nella memoria principale.
- Si hanno alcuni problemi (problema della “coerenza”)
 - Ci può essere più di un processore, ciascuno con la propria cache.
 - Alcune periferiche possono avere l’accesso diretto alla memoria principale.
 - Quindi può essere invalidato il contenuto di una cache o della memoria principale.

La tecnica “write through”

- Tutte le operazioni di scrittura vengono effettuate nella memoria principale ogni volta che si scrive sulla cache.
- Ogni CPU può monitorare il traffico della memoria principale per evitare l'incoerenza dei dati nella propria cache.
- Si genera notevole traffico col rischio di un collo di bottiglia.

La tecnica “write back”

- Inizialmente si scrive solo nella cache.
- Vi è un bit in ciascuna linea di cache, settato solo nel caso occorra aggiornare il corrispondente blocco.
- Si aggiorna un blocco in memoria principale, solo se il bit vale 1, quando tale blocco va rimpiazzato in cache
- Le periferiche devono accedere alla memoria principale attraverso la cache
- N.B.: solo il 15% dei riferimenti alla memoria comportano “aggiornamenti” dei dati (scritture)

Dimensione di ciascuna linea

- Più grande è maggiormente si sfrutta il principio di località.
- Se è però troppo grande:
 - Ci stanno pochi blocchi in cache e aumenta il tempo di caricamento in cache.
 - Le prime e le ultime parole all'interno di un blocco sono troppo lontane per poter sfruttare il principio di località.
- Il valore ottimo dipende da molte variabili, tra cui il particolare programma da eseguire.
- Valori tipici: da 2 a 8 parole.

Livelli di cache

- Oggi vengono usati fino a **tre livelli** di cache.
- La cache è divisa in una cache dedicata alle **istruzioni** ed in una cache dedicata ai **dati**.
- Si parla di cache di livello 1 (**L1**): quella più piccola e veloce che può avere capacità dell'ordine del KB (ad es. 256 KB). Questo livello è spesso composto da una cache istruzioni e una dati. La prima contiene le istruzioni che la CPU dovrebbe eseguire successivamente. Solitamente in questo livello vengono conservati i dati più utilizzati.
- **L2** è il secondo livello di cache. Con capacità dell'ordine dei MB (es. **8 MB**) ed è più lento rispetto al precedente.
- Il terzo e ultimo livello è livello **L3**, che può arrivare a capacità delle decine di **MB**.

Esempi “storici” di uso cache in CPU

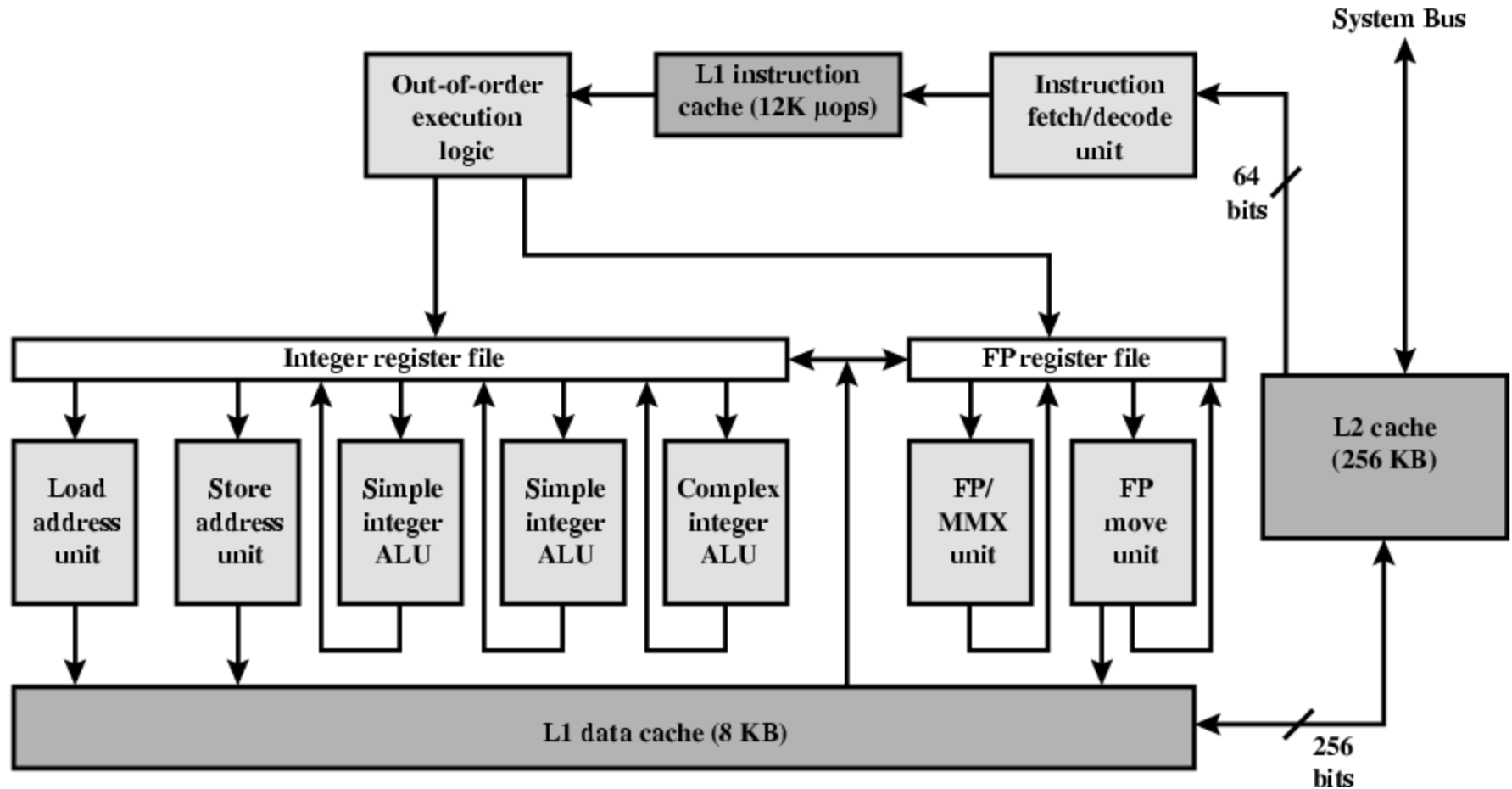
- Intel 80386 (1985) – Non supportava la cache interna.
- 80486 (1989) – Possedeva una cache interna da 8KB, in cui ogni linea poteva contenere un blocco da 16 byte ed era organizzata col metodo associativo su insiemi con 4 linee per set.
- Pentium (tutte le versioni, 1993-1999) – due cache interne
 - Una per i dati ed una per le istruzioni

Esempi “storici” di uso cache in CPU

Pentium 4 (anni 2000-2008)

- Pentium 4 – Le cache interne (L1), una per dati e una per istruzioni:
 - Dimensione: 8K byte
 - Dimensione linee: 64 byte
 - Metodo usato: associativo su insiemi con quattro linee per set
- Pentium 4 – La cache esterna (L2):
 - alimenta entrambe le cache interne
 - Dimensione: 256K byte
 - Dimensione linee: 128 byte
 - Metodo usato: associativo su insiemi con otto linee per set
 - E' implementata con SRAM (Static RAM)

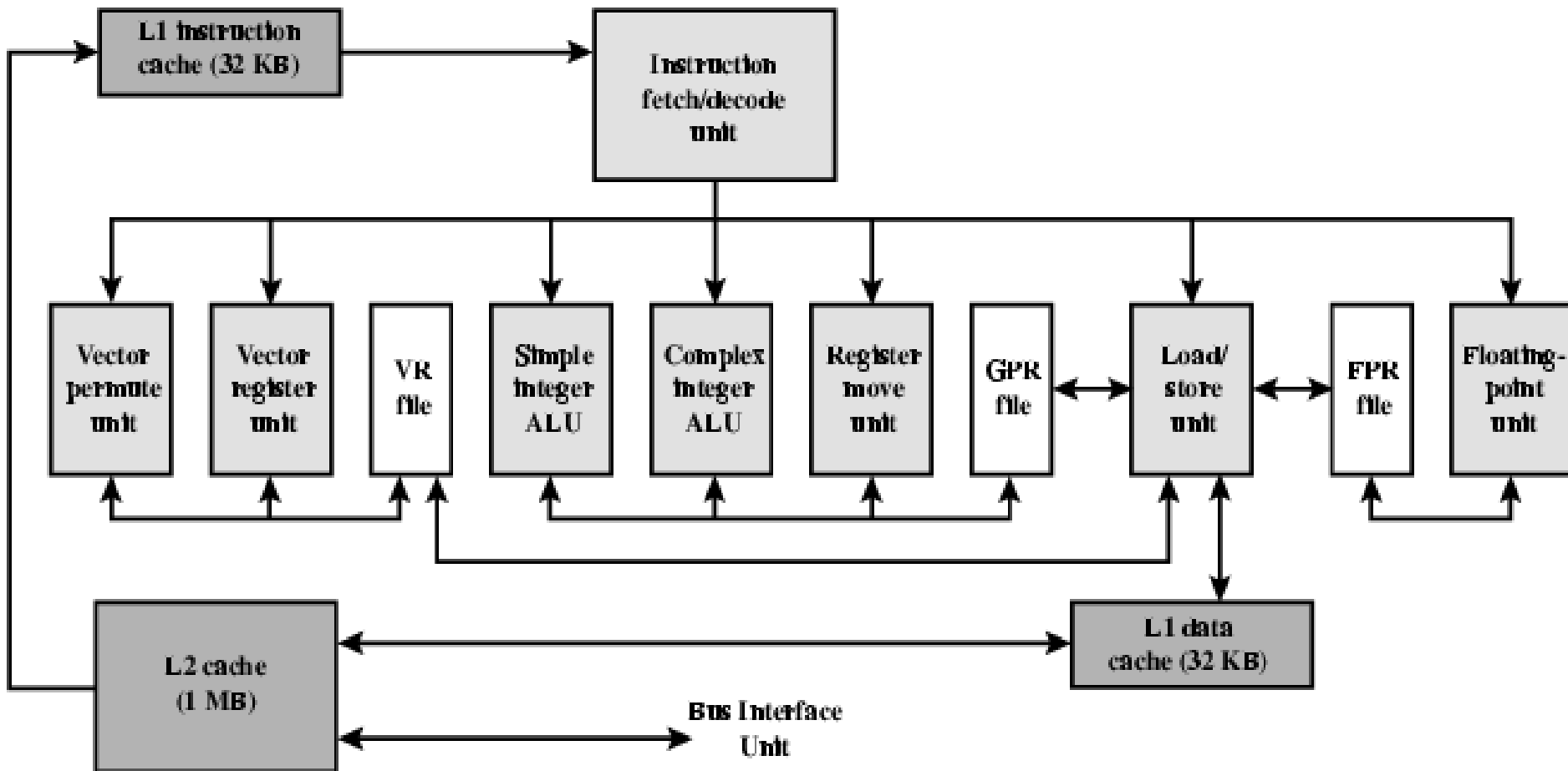
Pentium 4: Schema Semplificato



Esempi “storici” di uso cache in CPU: Power PC (Apple-IBM-Motorola)

- Power PC 601 (1984) – singola cache da 32 Kbyte con metodo associativo su insiemi, con otto linee per set
- 603 – due cache, ciascuna da 8 Kbyte, con metodo associativo su insiemi con due linee per set
- 604 – cache complessiva: 32 Kbyte
- 610 – cache complessiva: 64 Kbyte
- G3 & G4 (usati su computer Apple Power Mac negli anni '90)
 - 64 Kbyte di cache interna
 - > metodo associativo su insiemi con otto linee per set
 - 256k, 512k o 1Mbyte di cache esterna
 - > metodo associativo su insiemi con due linee per set

PowerPC G4: Schema Semplificato



Cenni storici sulla capacità della cache

Processor	Type	Year of Introduction	L1 cache ^a	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTA ^b	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—

^a Two values separated by a slash refer to instruction and data caches

^b Both caches are instruction only; no data caches

Tempo di Accesso Cache-Primaria

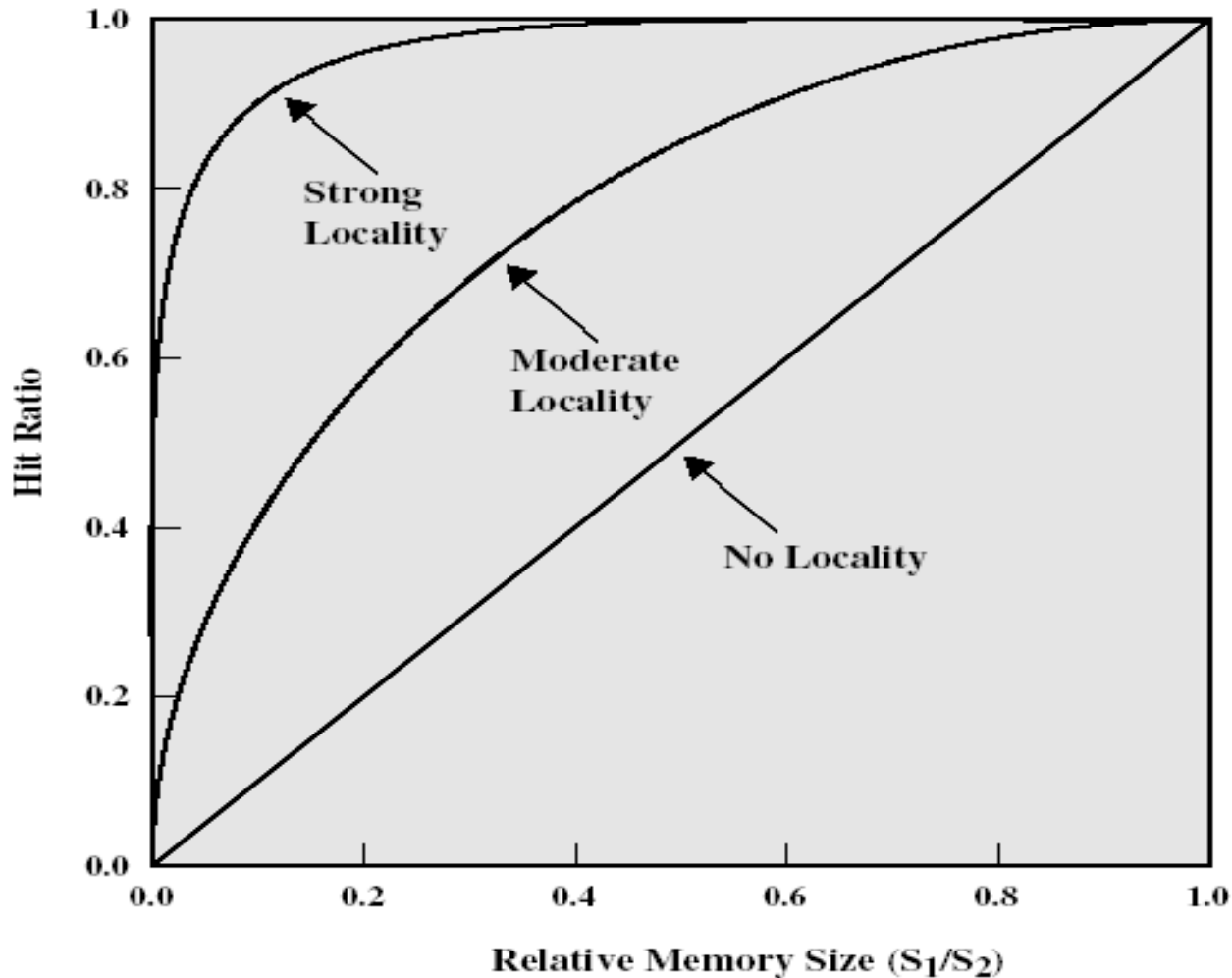
$$T_s = H T_1 + (1 - H) (T_1 + T_2) = T_1 + (1 - H) T_2$$

T_1 :tempo di accesso alla cache

T_2 :tempo di accesso alla memoria primaria

H : Hit ratio (percentuale di volte che il dato cercato è trovato nella memoria cache)

Dimensionamento Capacità Gerarchia



Tempo di accesso a gerarchie a più livelli

- Tempo medio di accesso al livello i-esimo: t_i
- Hit ratio
 - probabilità di trovare una parola al livello i-esimo: H_i
 - calcolato in termini di frequenza di successi
 - $H_{i+1} > H_i ; H_{N-1} = 1$
 - > i dati contenuti al livello i-esimo sono un sottoinsieme dei dati contenuti al livello (i+1)-esimo
- Hit ratio condizionale
 - probabilità di trovare una parola al livello i-esimo dato che **non** la si è trovata al livello (i-1)-esimo:

$$H_{i|\overline{i-1}}$$

Tempo medio di accesso alla gerarchia

- Due livelli:

$$T = H_0 t_0 + (1 - H_0)(t_1 + t_0)$$

- Tre livelli:

$$T = H_0 t_0 + (H_1 - H_0)(t_1 + t_0) + (1 - H_1)(t_2 + t_1 + t_0)$$

$$T = H_0 t_0 + (1 - H_0)H_{1|0}(t_1 + t_0) + (1 - H_0)(1 - H_{1|0})(t_2 + t_1 + t_0)$$

$$T = t_0 + (1 - H_0)t_1 + (1 - H_1)t_2$$

- In generale, con N livelli, da 0 a N-1 (si ricava per induzione):

$$T = t_0 + \sum_{i=1}^{N-1} (1 - H_{i-1})t_i$$

Unità di Memoria Esterna

- Tutti i dispositivi di memoria non direttamente accessibili dal processore.
- Principali dispositivi:
 - Dischi magnetici
 - > Fissi
 - > Removibili
 - > RAID (Redundant Array of Independent Disks)
 - Dischi ottici
 - > DVD
 - Unità esterne allo stato solido
 - Nastri magnetici
 - Dischi Magneto-Ottici

Dischi Magnetici

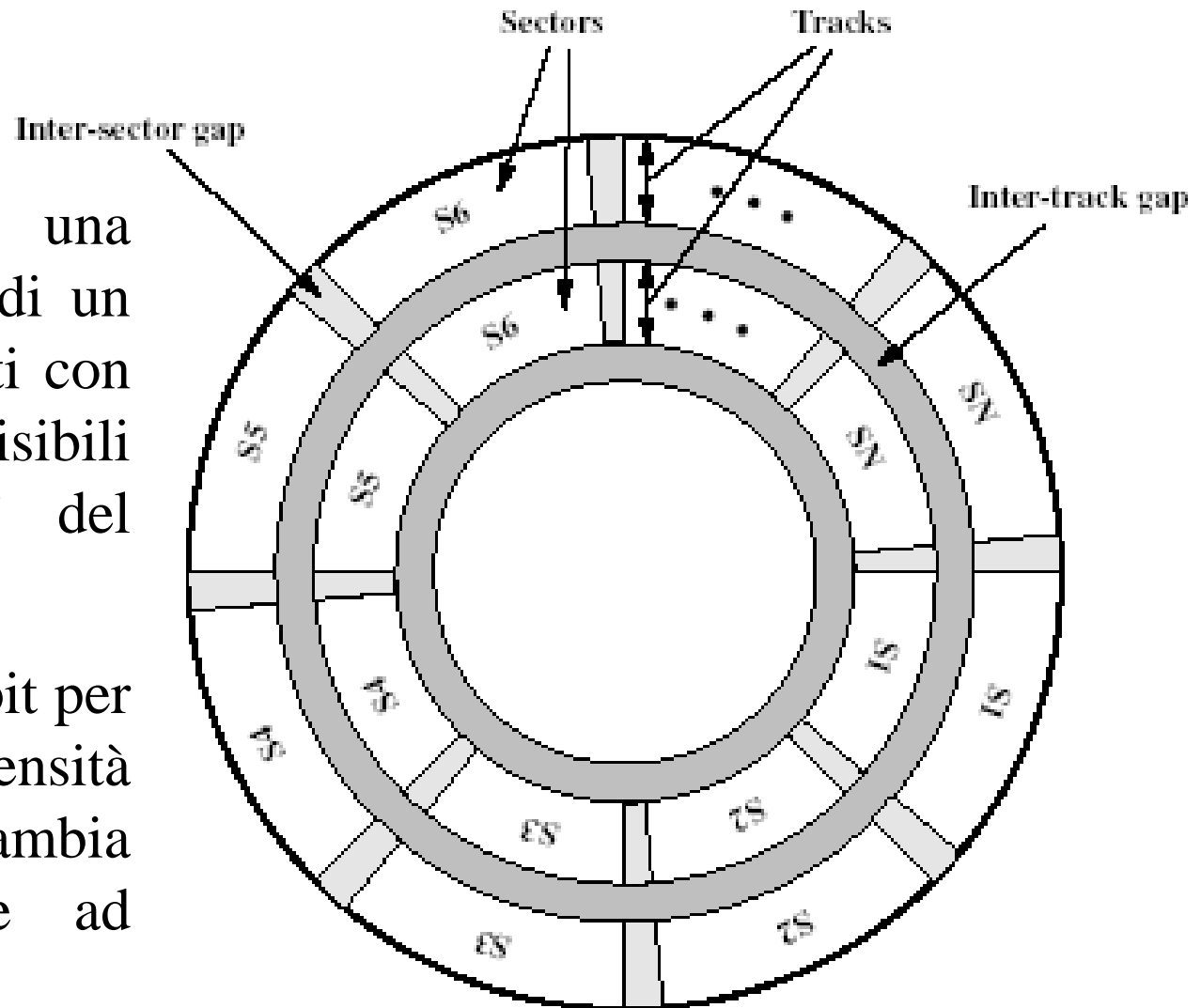
- E' un piatto circolare di metallo, o di plastica, ricoperto con materiale magnetico.
- Il meccanismo di lettura/scrittura è una spira conduttrice detta **testina**.
- Scrittura
 - Si crea un campo magnetico con un'opportuna corrente che passa nella spira.
- Lettura
 - Il campo magnetico del disco produce una corrente nella spira.
- Durante un'operazione di lettura/scrittura la testina è ferma sopra il piatto che ruota.

Organizzazione Dati e Formattazione di un disco

- Organizzazione e Formattazione (vedi anche figura della pagina seguente):
- Tracce
 - Cerchi concentrici della stessa larghezza della testina.
 - Sono separate con dei “gap” per ridurre gli errori di allineamento della testina e le interferenze tra i campi magnetici.
 - In un disco ce ne possono essere da 500 a 2000 per faccia.
- Settori
 - Regioni in cui blocchi di dati vengono trasferiti dal/nel disco.
 - Separati tra loro da dei “gap”.
 - Possono essere di dimensione differente.
 - Tipicamente ci sono 10-100 settori per traccia.

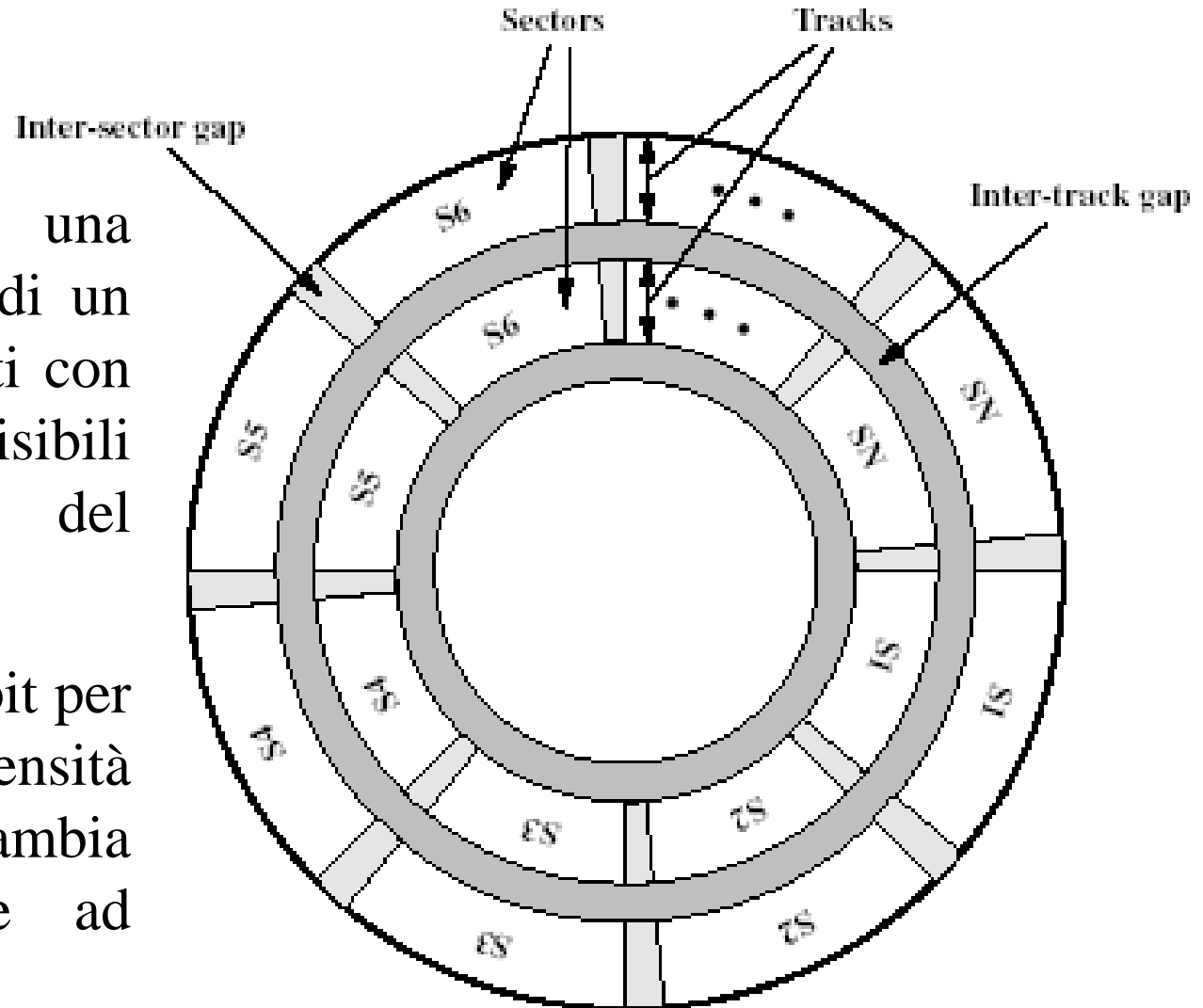
Organizzazione dei dati su di un disco

- Inizio/fine di una traccia, inizio/fine di un settore sono indicati con “control data” visibili solo dal “driver” del disco
- Stesso numero di bit per traccia. Quindi la densità dei dati (bit/inch) cambia da tracce interne ad esterne

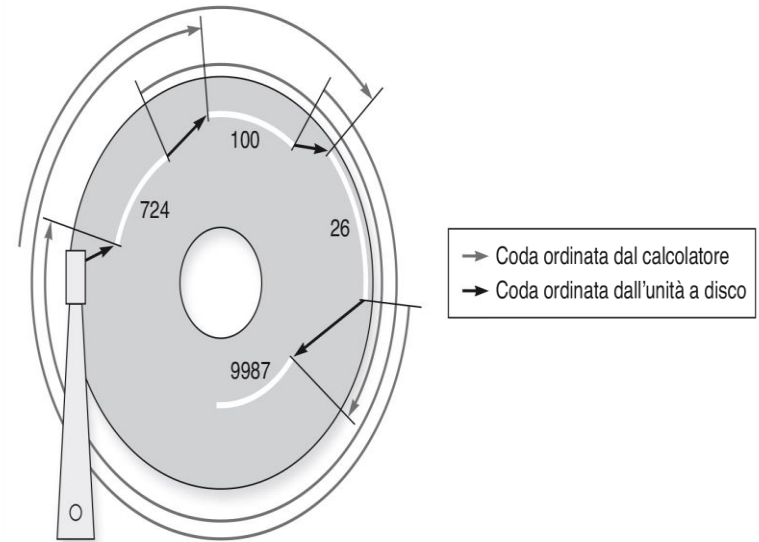


Organizzazione dei dati su di un disco

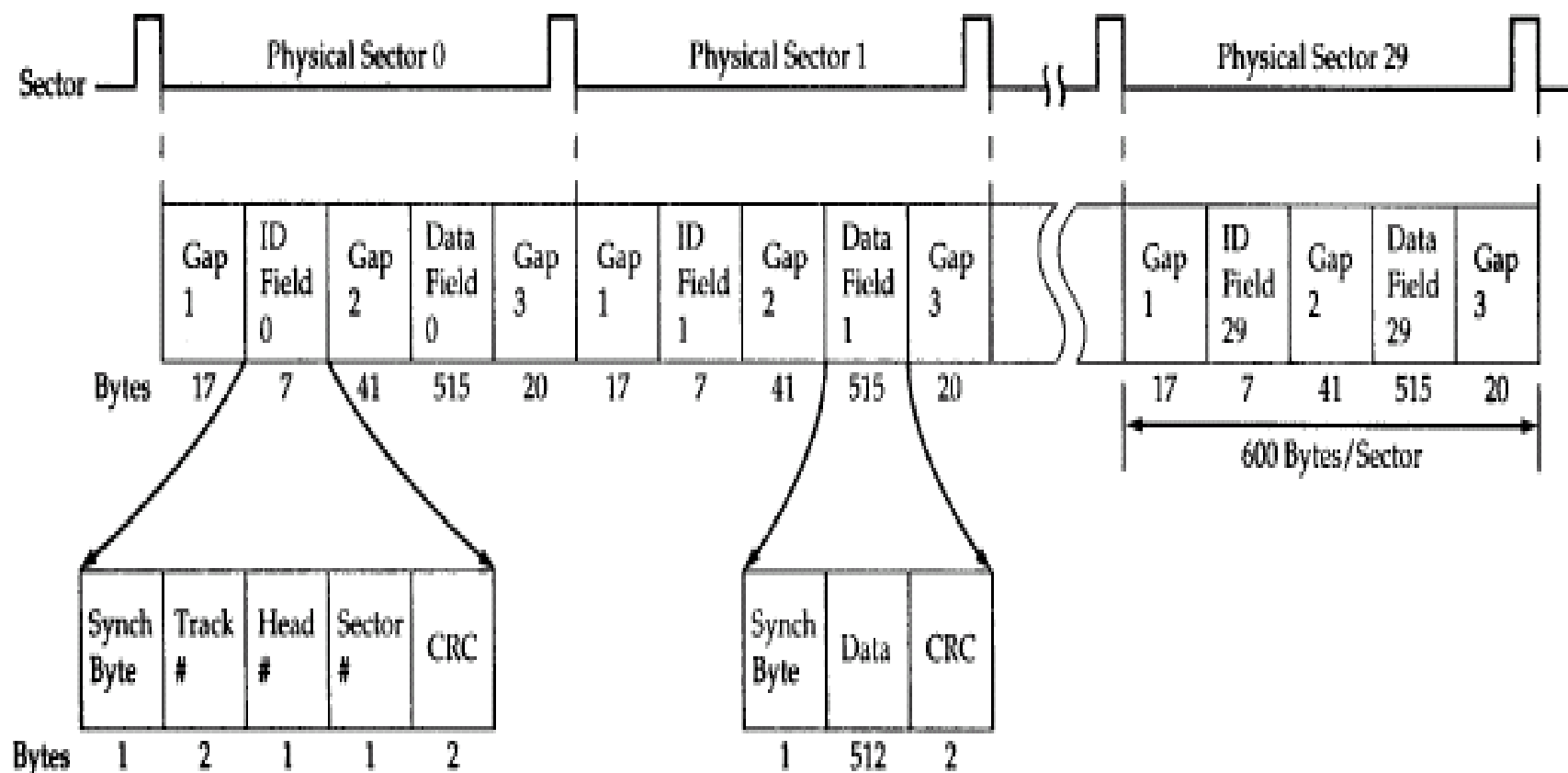
- Inizio/fine di una traccia, inizio/fine di un settore sono indicati con “control data” visibili solo dal “driver” del disco
- Stesso numero di bit per traccia. Quindi la densità dei dati (bit/inch) cambia da tracce interne ad esterne



Organizzazione unità disco



Esempio di Formattazione Dati



Parametri di prestazione di un disco (2)

- Il tempo di accesso al disco si suddivide in tempo di posizionamento e di latenza.
- Tempo di posizionamento
 - Tempo richiesto perché la testina si posizioni sulla traccia richiesta.
 - Formula approssimata:
$$T_s = m \times n + s$$
 - > T_s = Tempo di “seek” (posizionamento)
 - > n = numero di tracce attraversate
 - > m = costante dipendente dal drive del disco
 - > s = tempo di partenza (“start up time”)
 - Valori tipici:
 - > $m = 0.3 \text{ ms}$ e $s = 20 \text{ ms}$ (Hard disk economico di un PC)
 - > $m = 0.1 \text{ ms}$ e $s = 3 \text{ ms}$ (Hard disk più costoso)

Parametri di prestazione di un disco (3)

- Tempo di rotazione o di latenza
 - Tempo necessario perché il settore richiesto si posizioni sotto la testina.
 - Il disco ruota alla velocità di 3600 rpm (valore tipico); *mediamente*, il settore sarà sotto la testina mezzo giro dopo l'inizio della rotazione.
- $T_l = \frac{1}{2} T_r$
- > T_l = tempo di latenza; T_r tempo di una rotazione
 - > r = velocità di rotazione in rpm (round per minute)
- Esempio: se la velocità è 3600 rpm (rotazioni per minuto), si compie un giro completo in 16.6 ms; il tempo di latenza viene preso, per approssimazione, pari alla metà: 8.3 ms.

Parametri di prestazione di un disco (4)

- Tempo di trasferimento
 - Tempo necessario per leggere o scrivere i dati dalla/nella posizione richiesta.
- $T_t = b/rN$
 - > T_t = tempo di trasferimento
 - > b = numero di byte da trasferire
 - > N = numero di byte in una traccia
 - > r = velocità di rotazione
- Il tempo medio totale è dato dalla somma dei tempi di posizionamento, latenza e trasferimento.

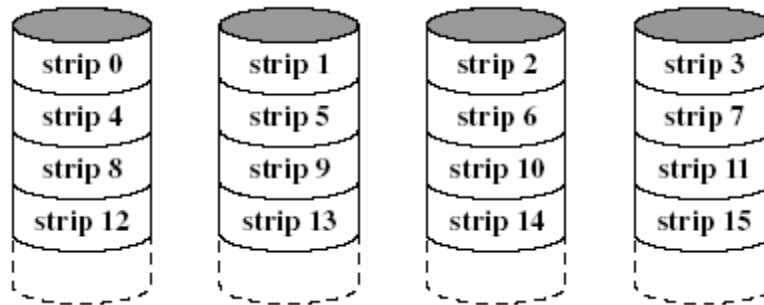
RAID

Redundant Array of Independent Disk

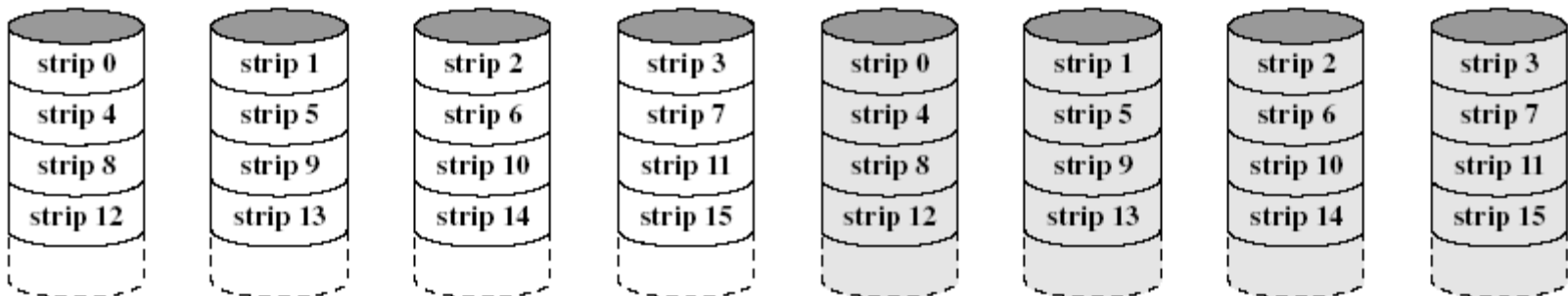
- Insieme di dischi che operano indipendentemente l'uno dall'altro e in parallelo.
 - Richieste separate possono essere eseguite in parallelo.
 - Una singola richiesta può essere eseguita in parallelo.
- Usato per diminuire il divario di velocità tra disco e CPU.
- Unico sistema di dischi “paralleli” standardizzato.
- Costituito da sette livelli (da 0 a 6)

Livelli del RAID (1)

- Livello 0
 - Unico livello non ridondante



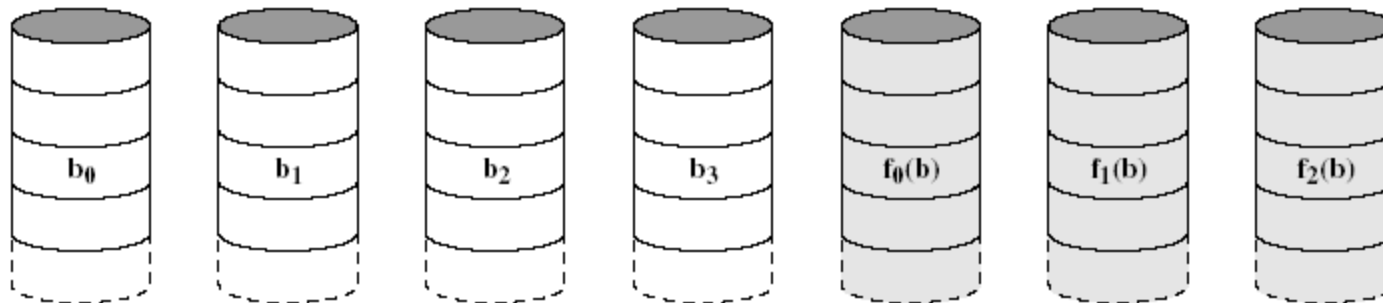
- Livello 1
 - I dati vengono scritti due volte e letti da una delle due copie



Livelli del RAID (2)

- Livello 2

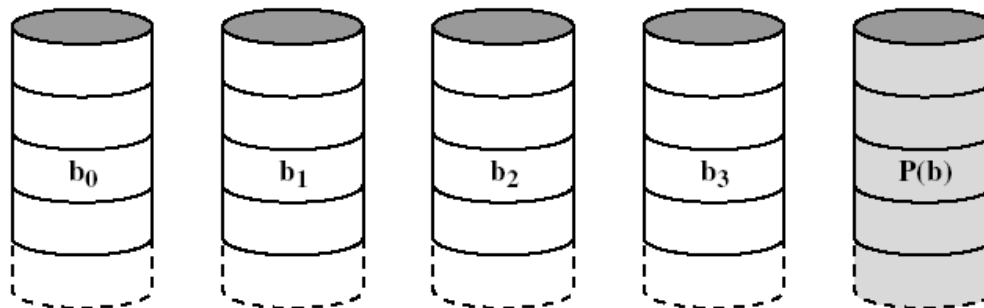
- Uso codice di Hamming.



- Livello 3

- Si genera un bit di parità facendo l'OR esclusivo tra i corrispondenti bit negli altri dischi.

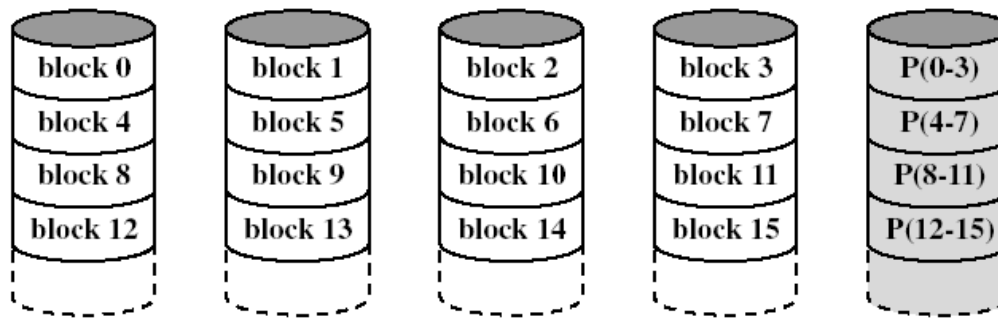
- Vi è solo un disco ridondante.



Livelli del RAID (3)

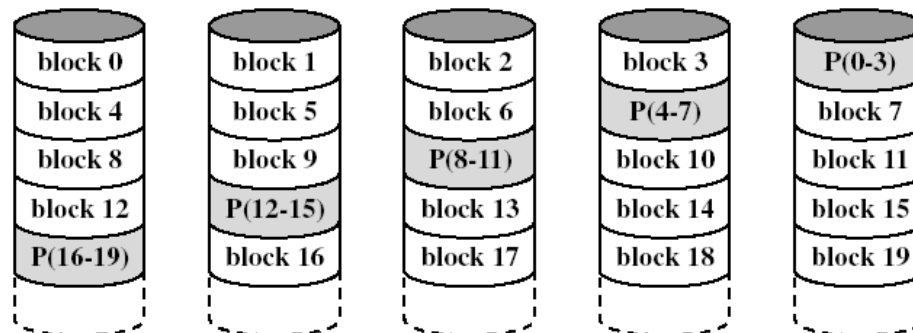
- Livello 4

- Nel disco ridondante ogni riga contiene la parità bit-a-bit delle corrispondenti righe dei dischi di dati.



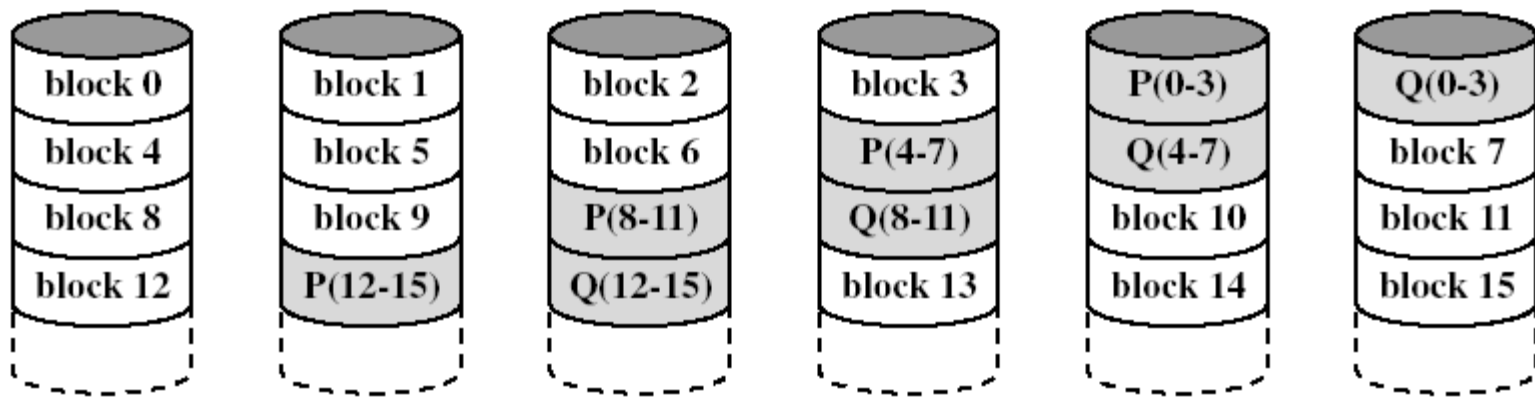
- Livello 5

- Si calcolano i bit di parità come nel livello 4, però questi sono distribuiti uniformemente su tutti i dischi.



Livelli del RAID (4)

- Livello 6
 - Ridondanza duale: si calcolano due diverse parità, distribuite uniformemente su tutti i dischi.

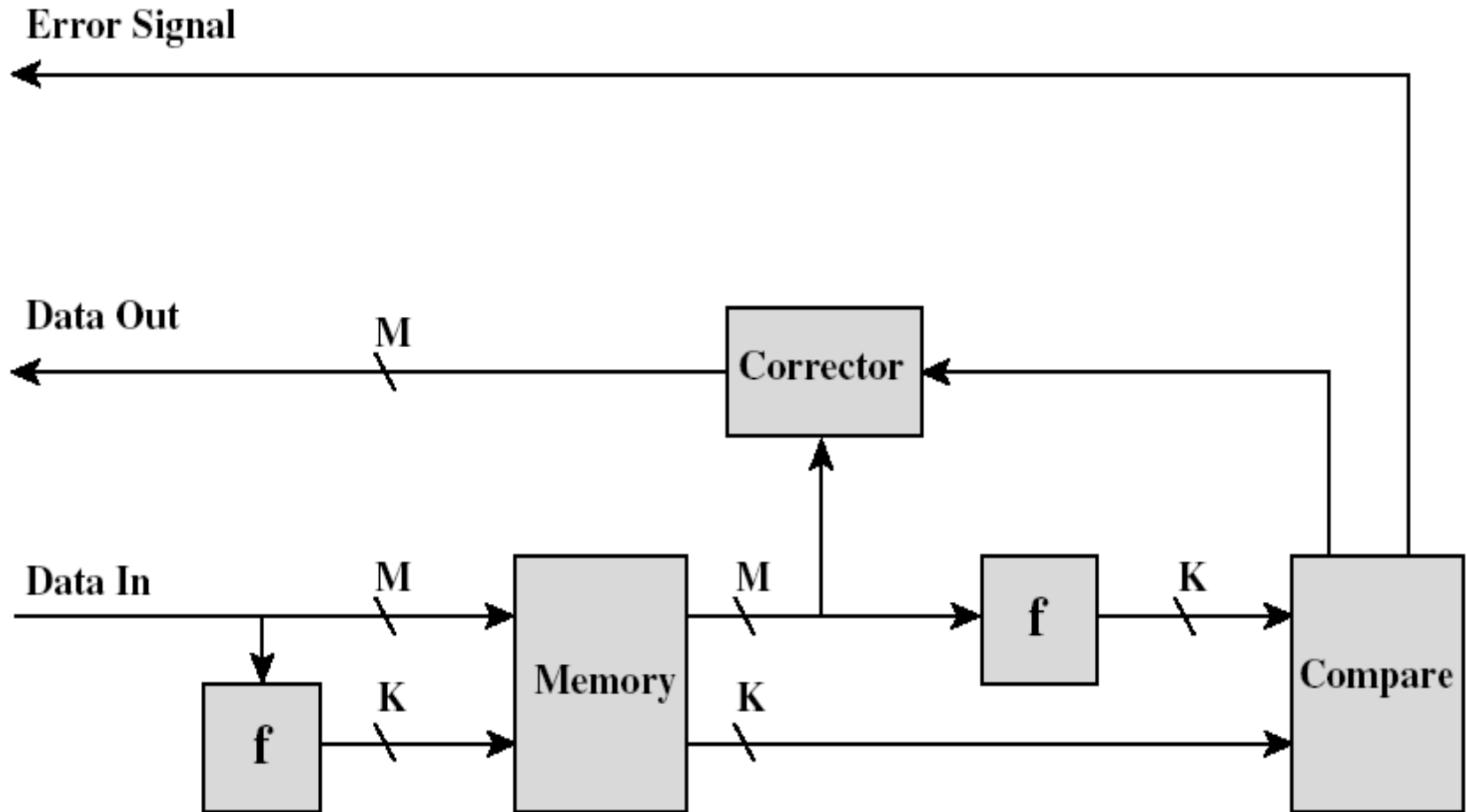


Codici a Correzione d'Errore

(Error-Correcting Codes)

- Una memoria è soggetta ad errori. Il valore dei bit immagazzinati può cambiare nel tempo per malfunzionamenti (difetti di fabbricazione, problemi di alimentazione, etc.)
- Durante la comunicazione di dati fra le unità l'informazione è soggetta a varie cause di deterioramento e disturbo
- Diventa così necessario usare opportune tecniche di rilevazione e correzione di eventuali errori
- I codici a correzione d'errore hanno il compito di rivelare e, quando possibile, correggere gli errori
- Molti sistemi di memoria utilizzano codici a correzione d'errore
- Tali codici sono usati anche per rilevare e correggere errori che avvengono durante le trasmissioni fra diversi dispositivi

Codici a Correzione d'Errore: Schema di Principio

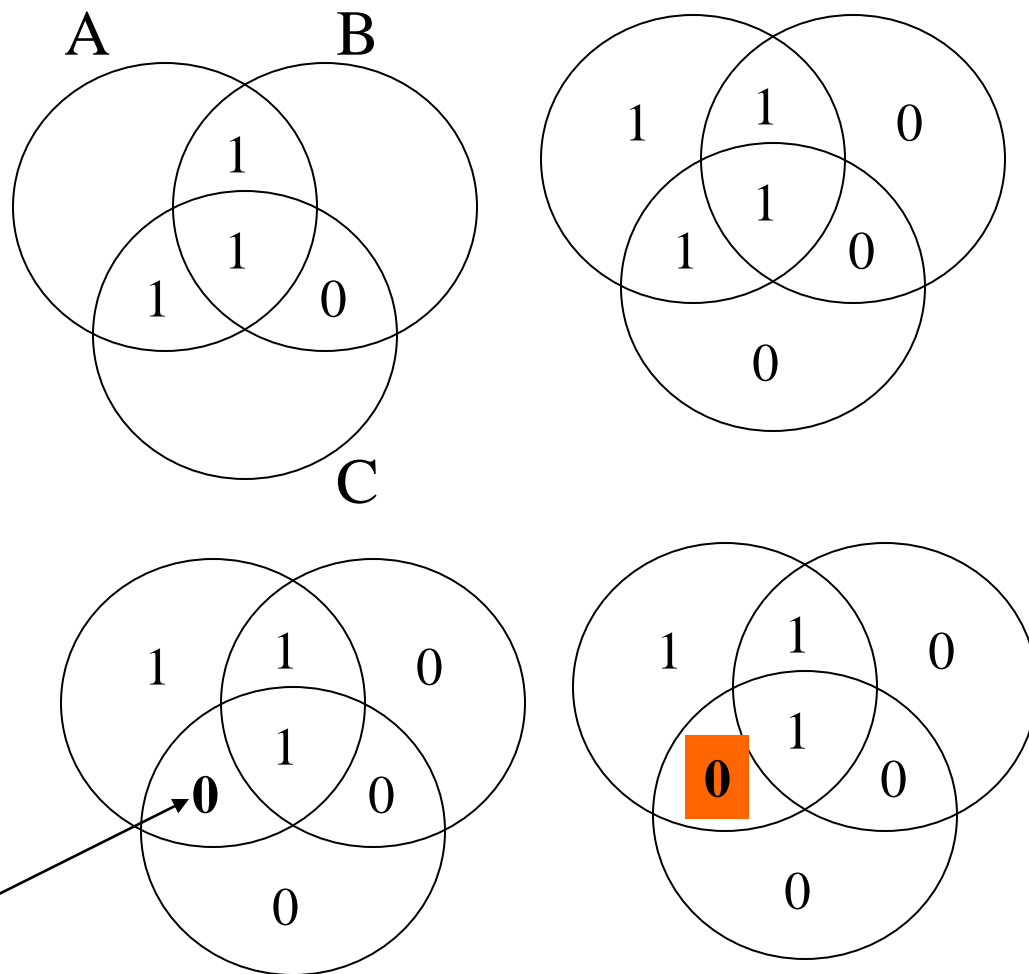


Il codice di Hamming

- E' il codice più semplice, ideato da Richard Hamming dei Bell Laboratories
- E' basato sull'aggiunta di un numero di bit, detti “di controllo”, alla stringa originale di bit d'informazione
- Tali bit hanno il compito di rilevare la presenza di errori nella stringa originale, e di fornire la posizione del bit errato nella stringa
 - si noti la differenza tra rilevazione e correzione
- A ciascun bit di controllo è associato un gruppo di bit della stringa di informazione
- I gruppi di bit non sono disgiunti
 - un bit di informazione è controllato da almeno due bit di controllo

Codice di Hamming – Idea di Base

- Esempio per “parola” di 4 bit usando i Diagrammi di Venn
- I 4 bit di informazione sono messi nelle intersezioni dei 3 cerchi
- Le restanti parti dei cerchi sono riempite con i bit di “parità” (bit di controllo)
- Se ipotizziamo che il bit segnato con la freccia diventi errato
- E’ possibile rivelare l’errore perché è nell’intersezione di A e C, ma non B



I bit di controllo nella codifica di Hamming

- Il numero di bit di controllo è deciso in base alla relazione:

$$2^K - 1 \geq N + K$$

- K: numero minimo di bit di controllo
 - N: numero di bit della stringa originale (stringa di “dati”/ “informazione”)
- Infatti bastano $2^K - 1$ configurazioni dei bit di controllo per rilevare e correggere un errore tra gli $N + K$ bit codificati. La restante configurazione dei K bit controllo serve per segnalare l'assenza di errori

Formato del codice di Hamming

- Ogni bit di controllo “controlla” un gruppo di bit della stringa di informazione (bit controllati)
- Nel codice di Hamming la scelta è tale che la stringa finale (bit di controllo + bit d’informazione) è così formattata:
 - I bit di controllo sono nelle posizioni della stringa con valore potenza di due (2^0 , 2^1 , etc.)
 - tutte le altre posizioni sono coperte dai bit della stringa di informazione, a partire dalla posizione meno significativa
- Un bit d’informazione è controllato dal bit i -esimo di controllo se la configurazione binaria che identifica quel bit d’informazione ha il valore dell’ i -esimo bit pari a 1 (vedi esempio successivo)

Esempio (1)

- 8 bit di informazione: M1, M2,..M8
- 4 bit di controllo (N.B., in seguito li indicheremo con numeri progressivi)
- La stringa finale, bit di controllo + bit d'informazione, è di 12 bit
- I bit di controllo stanno nelle posizioni $2^0, 2^1, 2^2, 2^3$

Bit Position	Position Number				Check Bit	Data Bit
12	1	1	0	0		M8
11	1	0	1	1		M7
10	1	0	1	0		M6
9	1	0	0	1		M5
8	1	0	0	0	C8	
7	0	1	1	1		M4
6	0	1	1	0		M3
5	0	1	0	1		M2
4	0	1	0	0	C4	
3	0	0	1	1		M1
2	0	0	1	0	C2	
1	0	0	0	1	C1	

Esempio (2)

I bit di controllo sono calcolati come:

$$C1 = \text{EXOR} (M1, M2, M4, M5, M7)$$

$$C2 = \text{EXOR} (M1, M3, M4, M6, M7)$$

$$C4 = \text{EXOR} (M2, M3, M4, M8)$$

$$C8 = \text{EXOR} (M5, M6, M7, M8)$$

In generale, il bit in posizione n è “controllato” da quei bit di controllo C_i tali che :

$$\text{à } C_i 2^i = n$$

La “stringa” di errore

- Ad ogni bit di controllo è associato un bit di errore e_i . La stringa dei bit di errore viene utilizzata in fase di rivelazione degli errori
- Il valore del bit di controllo i -esimo C_i è tale che:

$$e_i = C_i \oplus \{parità\text{BitControllati}\} = 0$$

- Poiché i gruppi di bit controllati non sono disgiunti:
 - se un solo bit della stringa di errore non è nullo, allora il relativo bit di controllo è stato alterato, ma non ci sono errori nella stringa di informazione
 - se più bit della stringa di errore non sono nulli, allora c'è un errore nella stringa di informazione

Codice di Hamming: esempio

- N=8 bit di informazione
— B0, B1, B2, B3, B4, B5, B6, B7
- K=4 bit di controllo
— C0, C1, C2, C3 (stringa di controllo)
— E0, E1, E2, E3 (stringa di errore)
- Al centro della tabella abbiamo le configurazioni della stringa di errore
- Da quanto detto in precedenza, i bit della stringa di informazione controllati da C0 sono B0, B1, B3, B4, B6
- Ad es. il bit di controllo C0 sarà tale che:

$$e_0 = C0 \oplus B0 \oplus B1 \oplus B3 \oplus B4 \oplus B6 = 0$$

1	0001	C0
2	0010	C1
3	0011	B0
4	0100	C2
5	0101	B1
6	0110	B2
7	0111	B3
8	1000	C3
9	1001	B4
10	1010	B5
11	1011	B6
12	1100	B7

I bit controllati

Quando la parità tra i bit in **neretto** non è nulla significa che uno dei bit è errato

1	0001	C0	0001	C0	0001	C0	0001	C0
2	0010	C1	00 10	C1	0010	C1	0010	C1
3	0011	B0	0011	B0	0011	B0	0011	B0
4	0100	C2	0100	C2	0 100	C2	0100	C2
5	0101	B1	0101	B1	0101	B1	0101	B1
6	0110	B2	01 10	B2	0110	B2	0110	B2
7	0111	B3	0111	B3	0111	B3	0111	B3
8	1000	C3	1000	C3	1000	C3	1000	C3
9	1001	B4	1001	B4	1001	B4	1001	B4
10	1010	B5	10 10	B5	1010	B5	1010	B5
11	1011	B6	1011	B6	1011	B6	1011	B6
12	1100	B7	1100	B7	1100	B7	1100	B7