Dicii

Università di Roma Tor Vergata

La gerarchia di memoria

Alessandro Pellegrini
a.pellegrini@ing.uniroma2.it

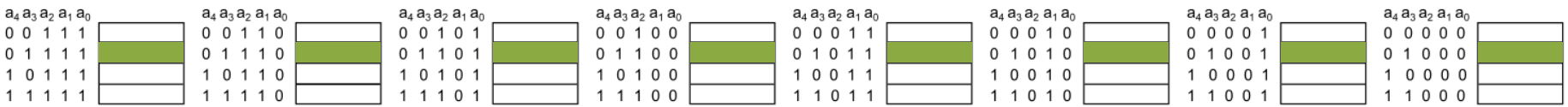
Memoria primaria: organizzazione logica

- Modello di memoria *flat*: la memoria è un lungo vettore di celle di memoria (byte)
- Ogni cella è identificata da un indirizzo
- Esempio di organizzazione logica a vettore di 32 celle
  - Indirizzo composto da 5 bit

Memoria primaria: organizzazione in moduli

- Problema:
  - il processore vuole leggere dati di 1, 2, 4, 8 byte
  - la memoria è indirizzata al byte
- Soluzione:
  - organizzazione in *moduli*
  - possibilità di recuperare più byte in parallelo
  - I tre bit meno significativi identificano il modulo, quelli più significativi la riga

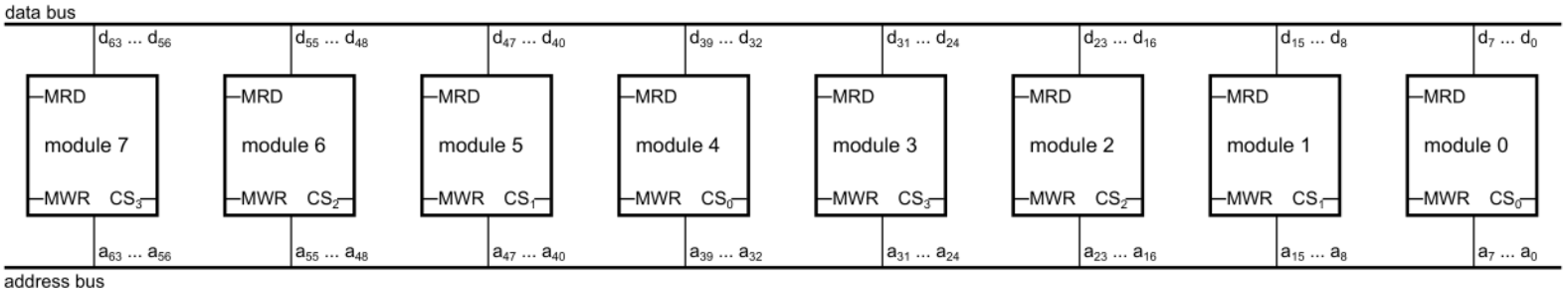
- Esempio: lettura della *riga* 01:



Memoria primaria: organizzazione in moduli

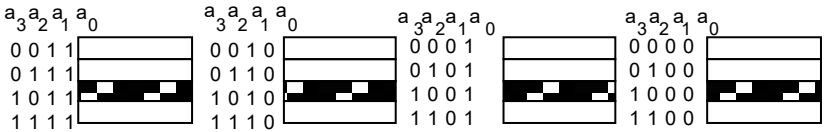
- Questa soluzione non è sufficiente:
  - Address bus e data bus sono a 64 bit
  - Il processore potrebbe voler leggere/scrivere meno di 8 byte

- Soluzione: introduzione del segnale Chip Select (CS)

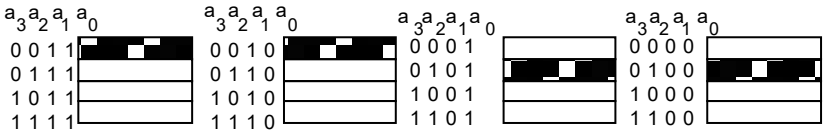


Memoria: allineamento e disallineamento

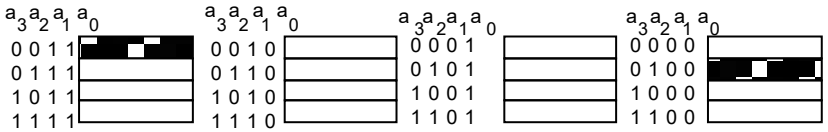
- Problema:
  - la memoria è indirizzabile al byte
  - Il dato potrebbe essere diposto su più righe
- Soluzione hardware: effettuare più letture, ricostruire il dato
- Soluzione software: *padding*



Quattro byte allineati sullo stesso indirizzo di riga.

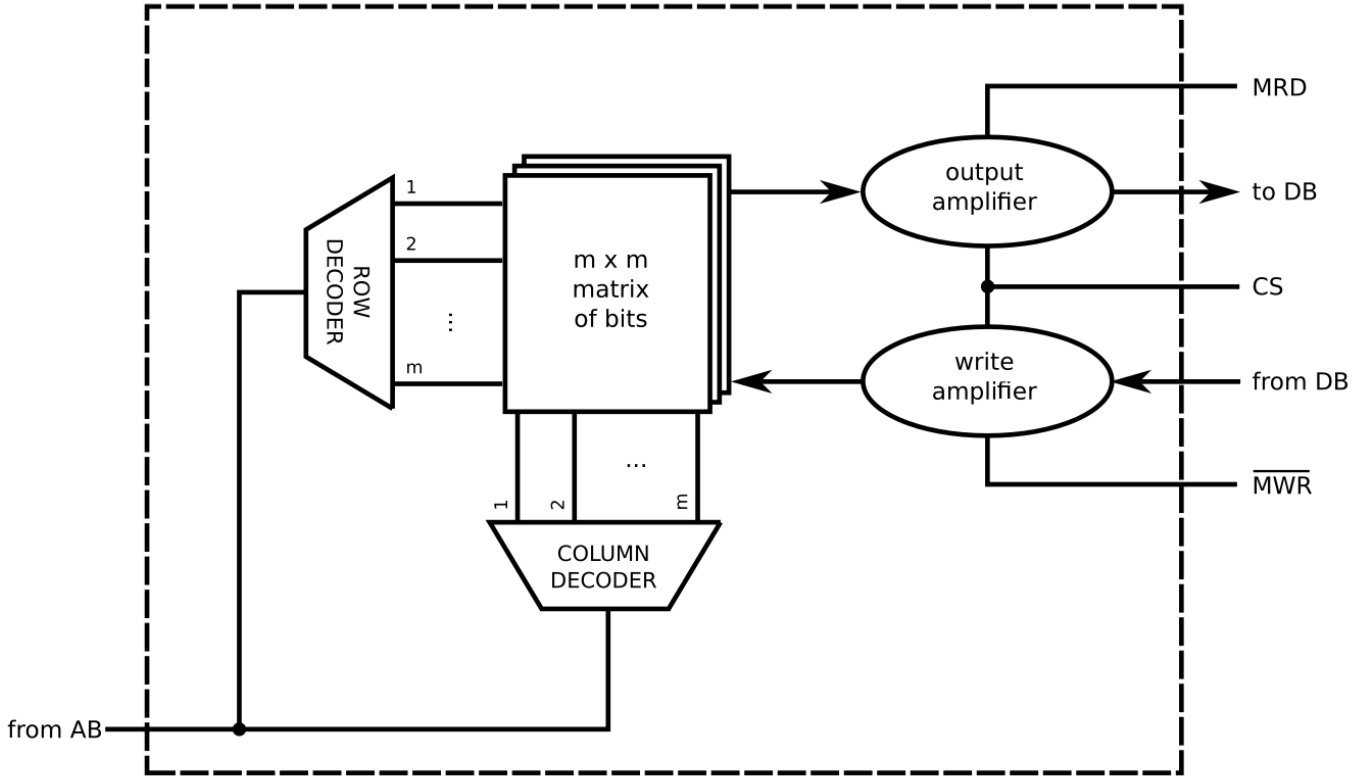


Quattro byte disallineati.

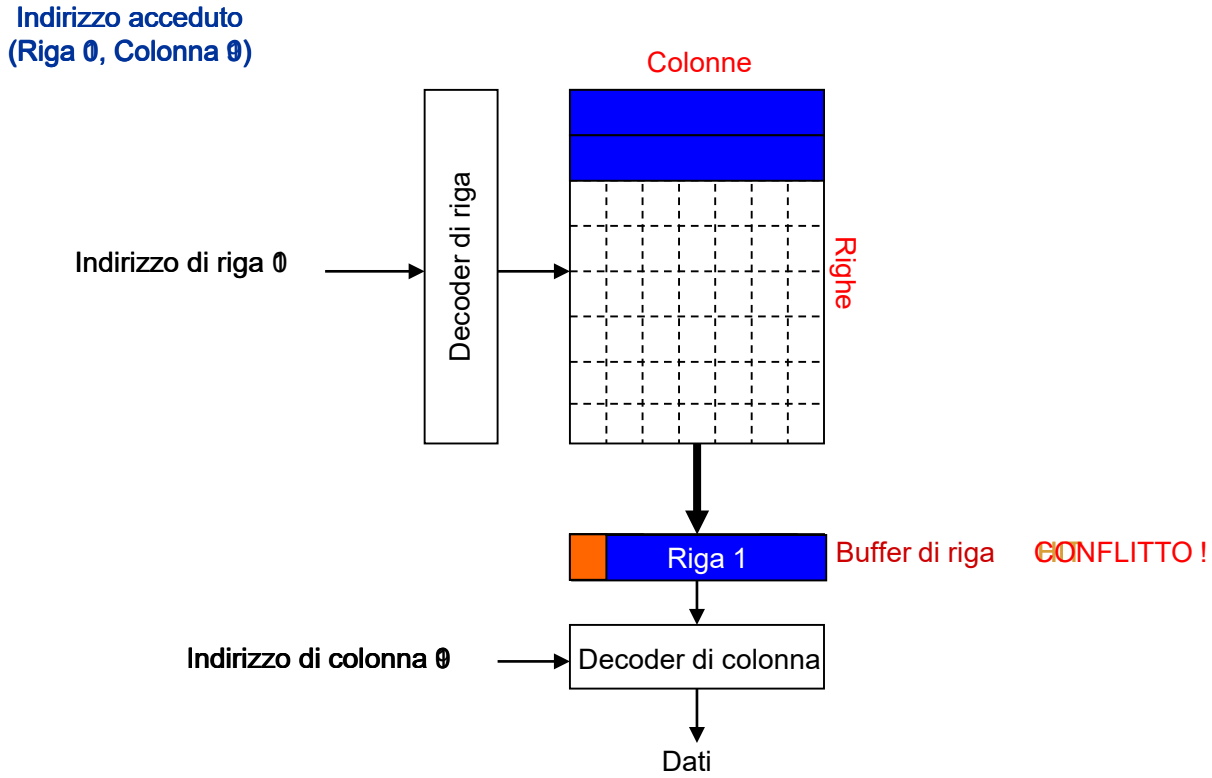


Due byte disallineati

Organizzazione a matrice

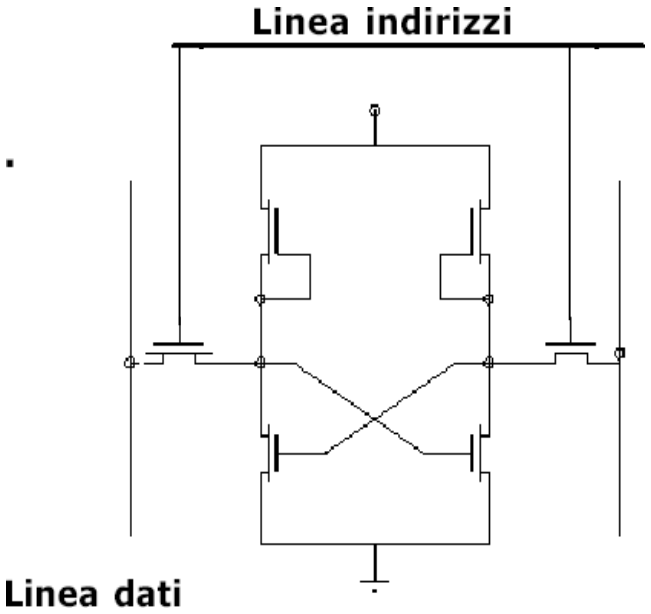


Organizzazione a matrice: funzionamento

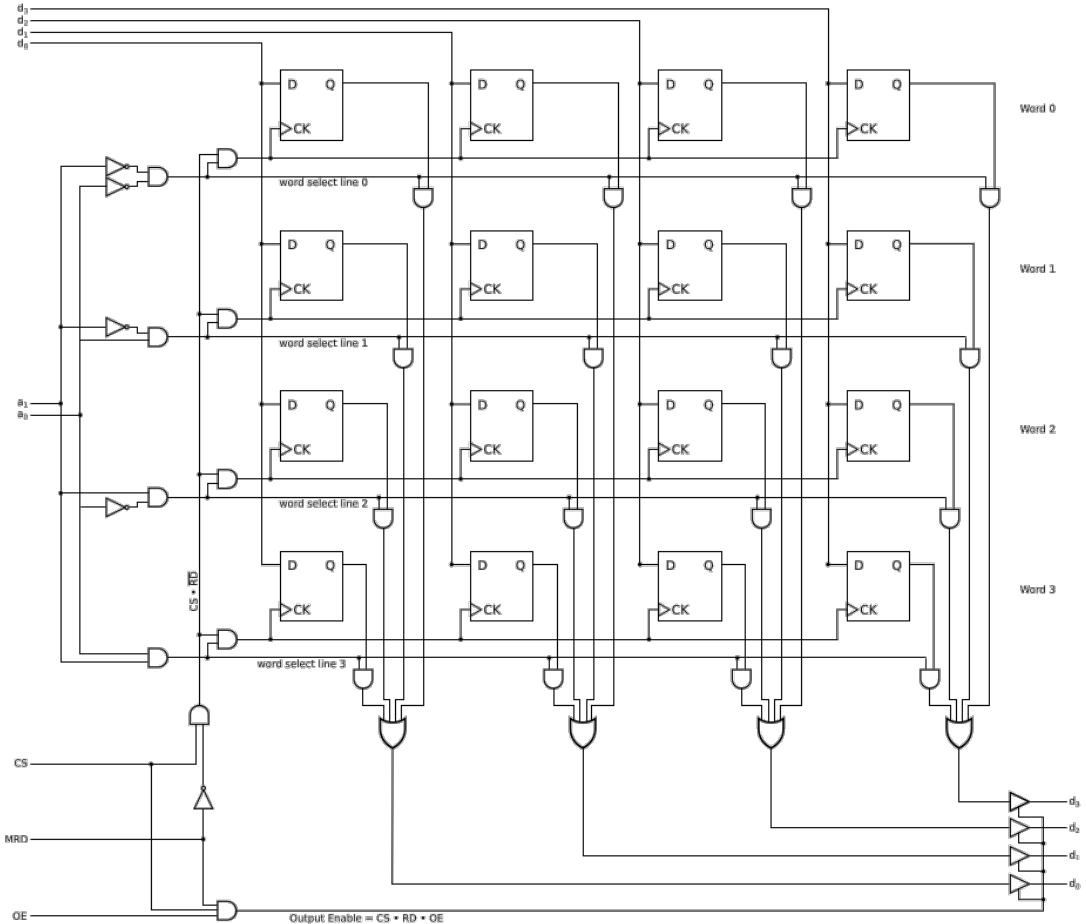


Le memorie RAM statiche

- La cella elementare è costituita da 6 transistor MOS che formano un flip-flop
- L'informazione permane stabile in presenza della tensione di alimentazione
- Tempi di accesso rapidi
- Costi elevati



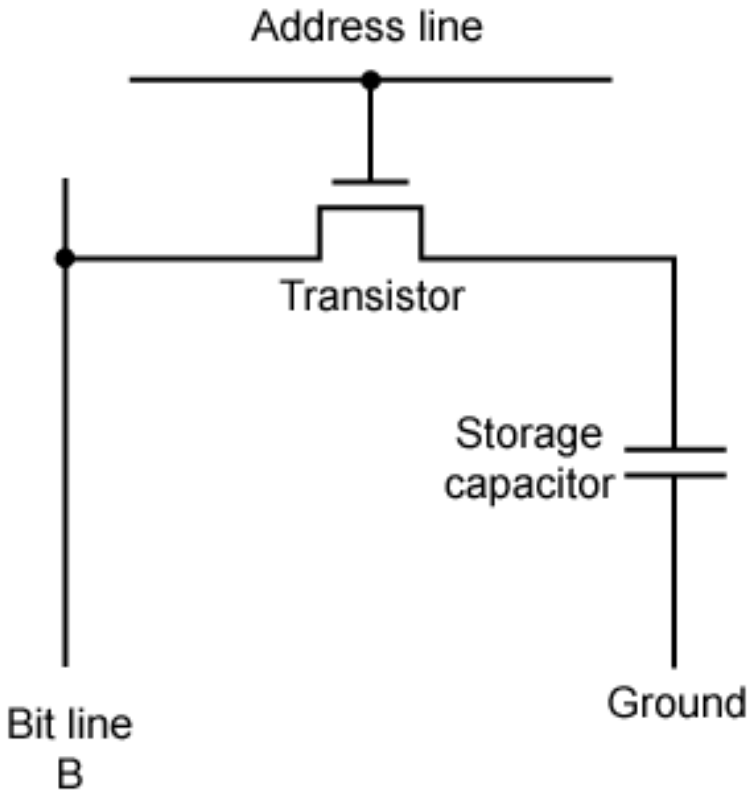
Organizzazione logica di una memoria RAM statica



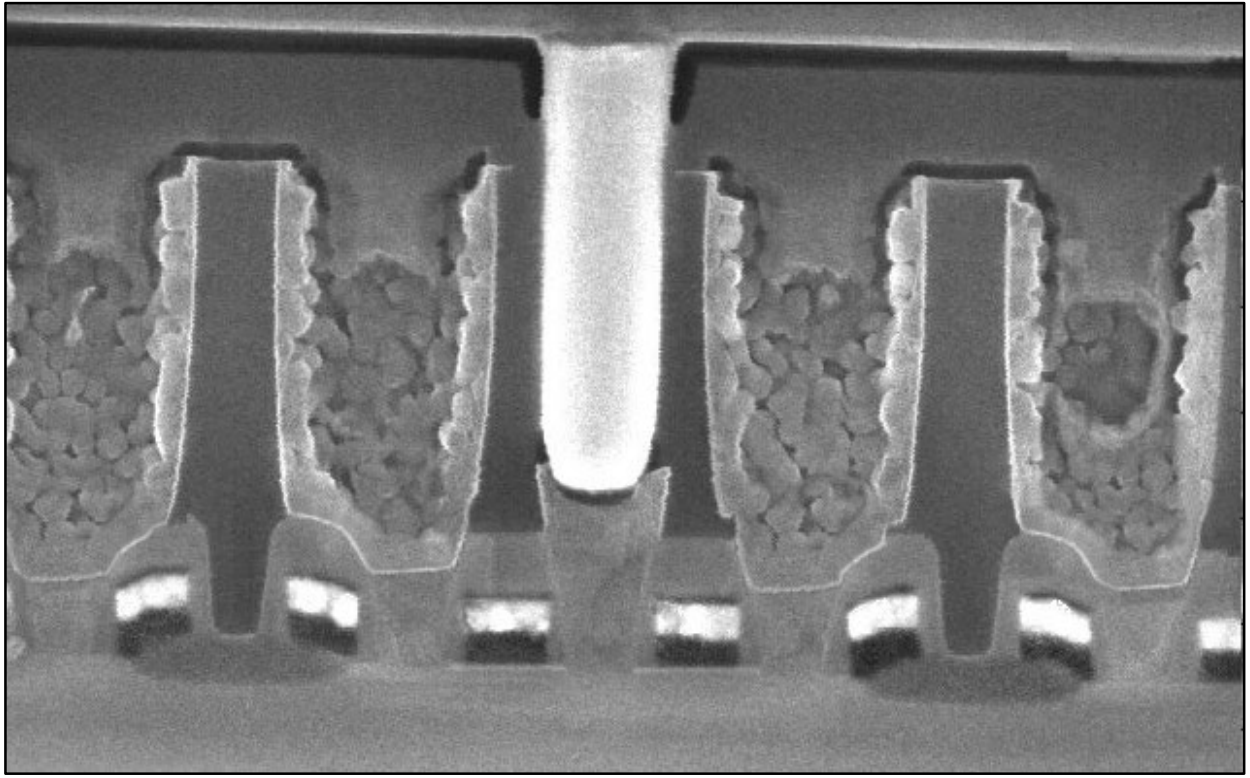
- La matrice di bit è realizzata da un insieme di flip-flop
- I flip-flop sono circondati da un circuito combinatorio per la lettura/scrittura dei dati

Le memorie RAM dinamiche

- La cella elementare è costituita da un condensatore che viene caricato (1) o scaricato (0)
- Tempi di accesso alti
- La semplicità della cella consente capacità molto elevate in spazi (e costi) contenuti

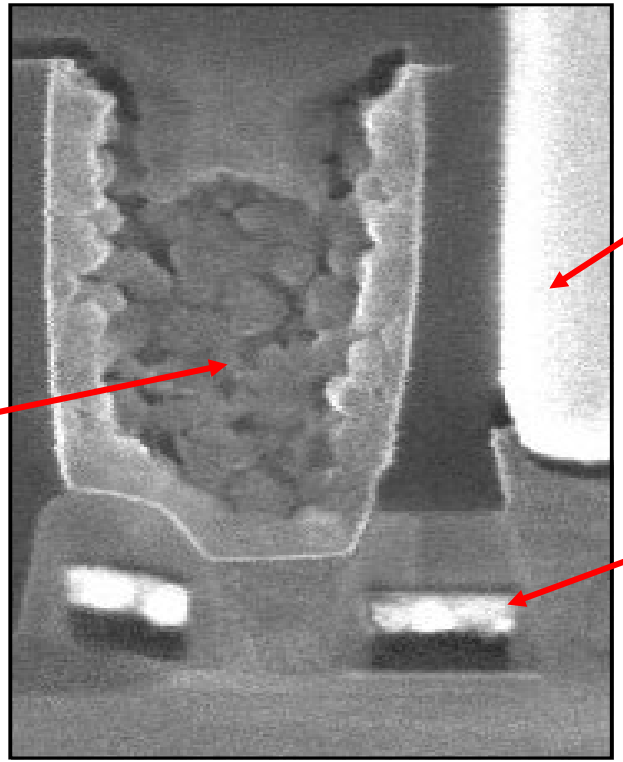


DRAM: come è fatta fisicamente



Cella di memoria DRAM

1 bit

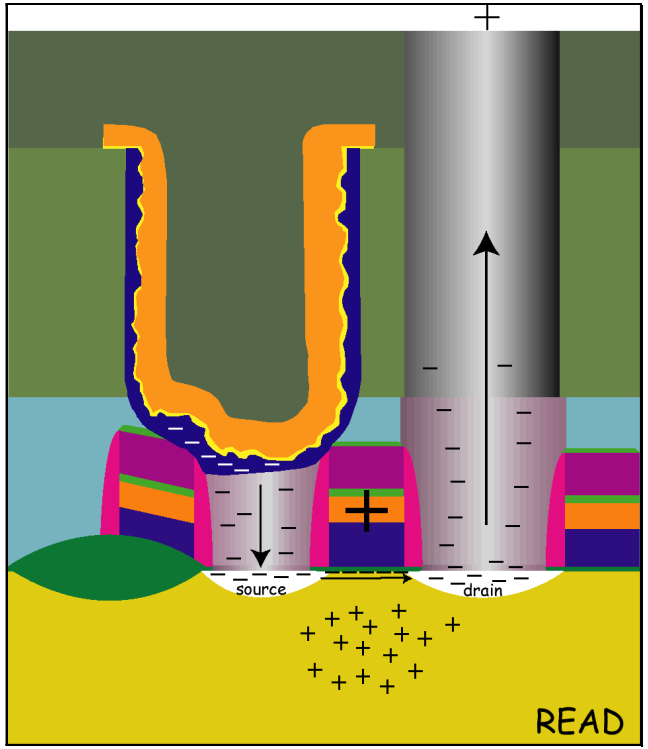


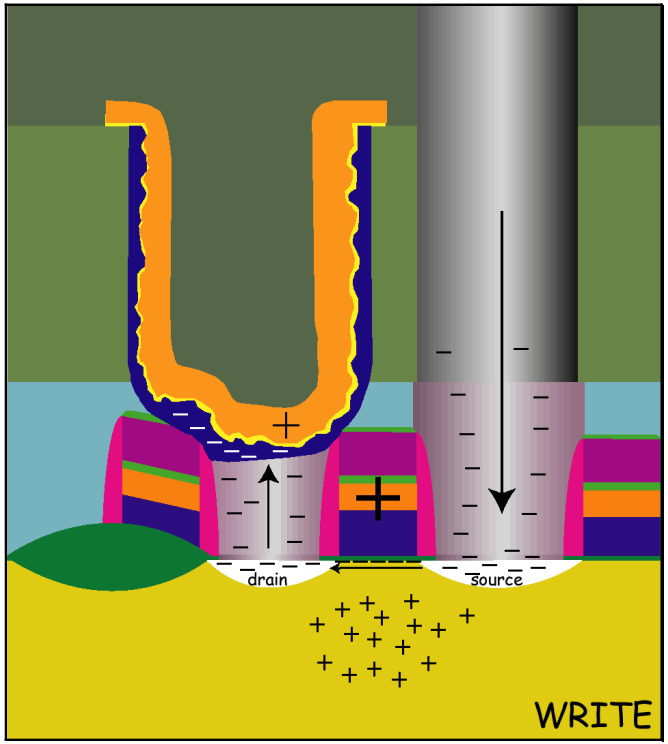
Linea di colonna

Condensatore

Gate/linea di riga

Operazioni di lettura e scrittura

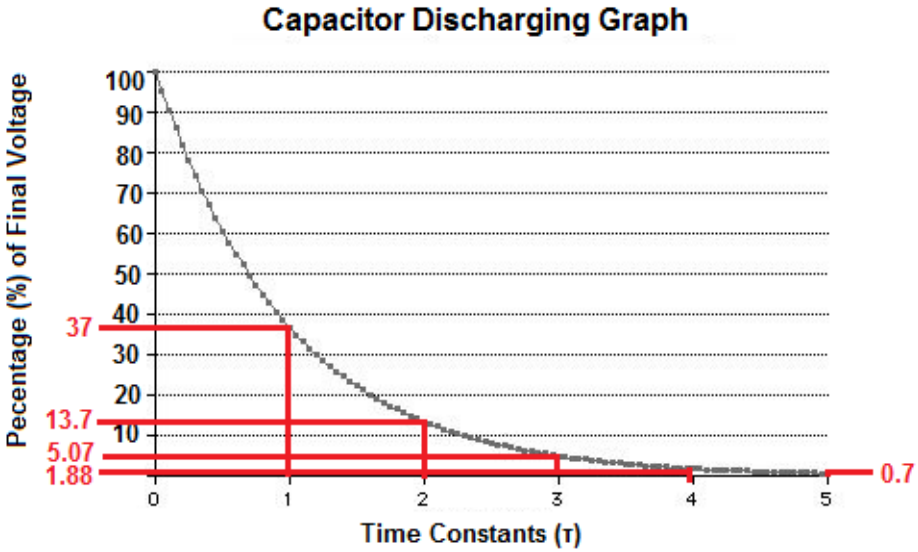




DRAM Refresh

- Problema: i condensatori si scaricano naturalmente
- *Costante temporale*: il tempo necessario a far scaricare il condensatore del 63%

- Dopo questo tempo, l'informazione è persa
  - Non è più possibile distinguere tra 0 e 1

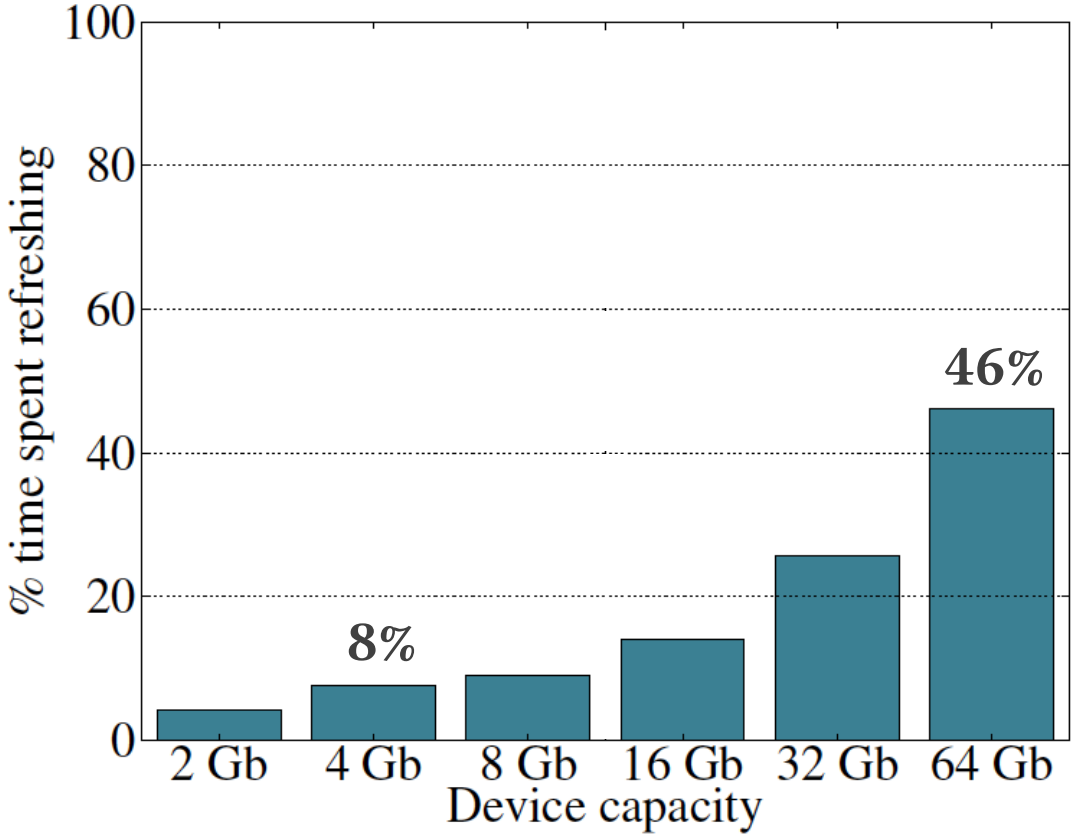


DRAM Refresh

- Il controllore della memoria deve “rinfrescare” periodicamente tutte le righe per ripristinare la carica
- Valore tipico dell’intervallo: 64 ms
- Problematiche del refresh:
  - *Consumo energetico*: ogni refresh consuma energia
  - *Degradazione delle prestazioni*: durante il refresh, una riga non è disponibile

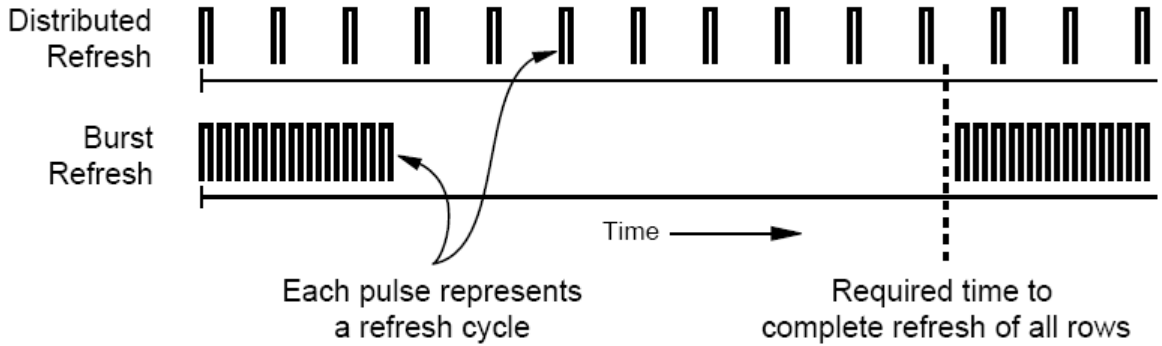
- Lunghe pause nella possibilità di accesso alla memoria durante il refresh
- Le dimensioni e le velocità delle memorie sono direttamente limitate dalla necessità di refresh

Overhead del Refresh



Refresh distribuito

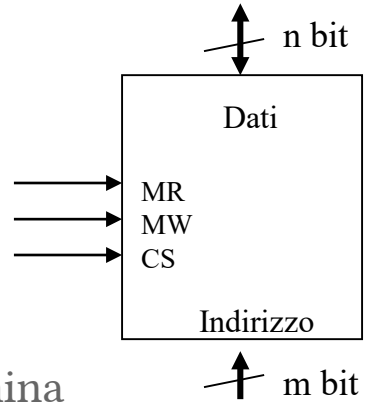
- Refresh a burst: ogni 64ms si effettua il refresh di *tutte* le righe
- Refresh distribuito: le righe sno soggette a refresh in istanti temporali differenti



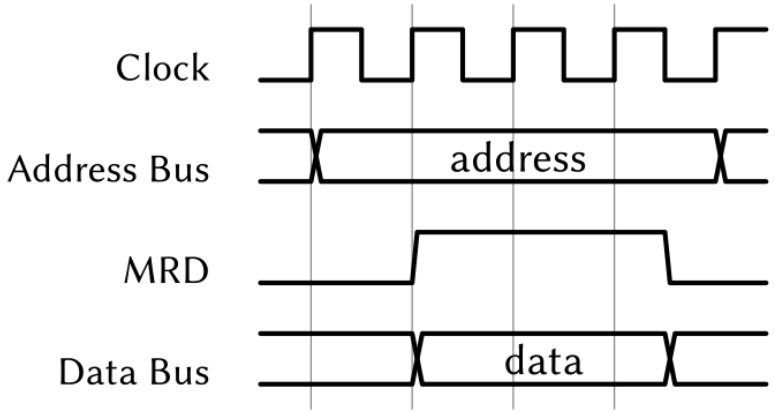
- Il refresh distribuito elimina (statisticamente) le lunghe pause

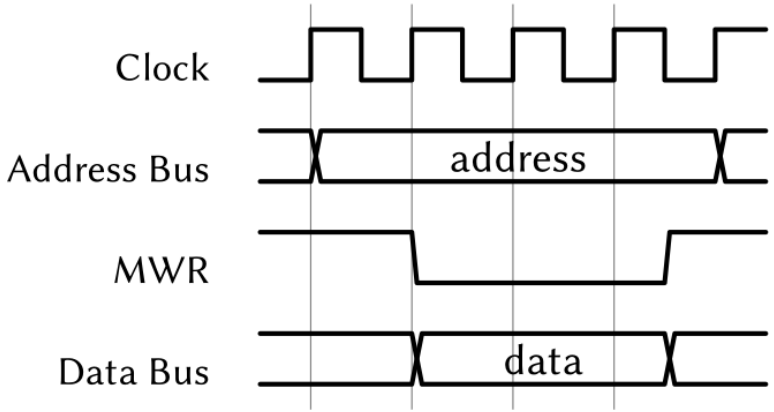
Interazione sincrona con la memoria

- Funzionalmente è caratterizzata dai seguenti segnali
  - Indirizzo della parola da leggere/scrivere
  - MR, affermato se si vuole leggere
  - MW, affermato se si vuole scrivere
  - CS, Abilita l'intero modulo
  - Dati

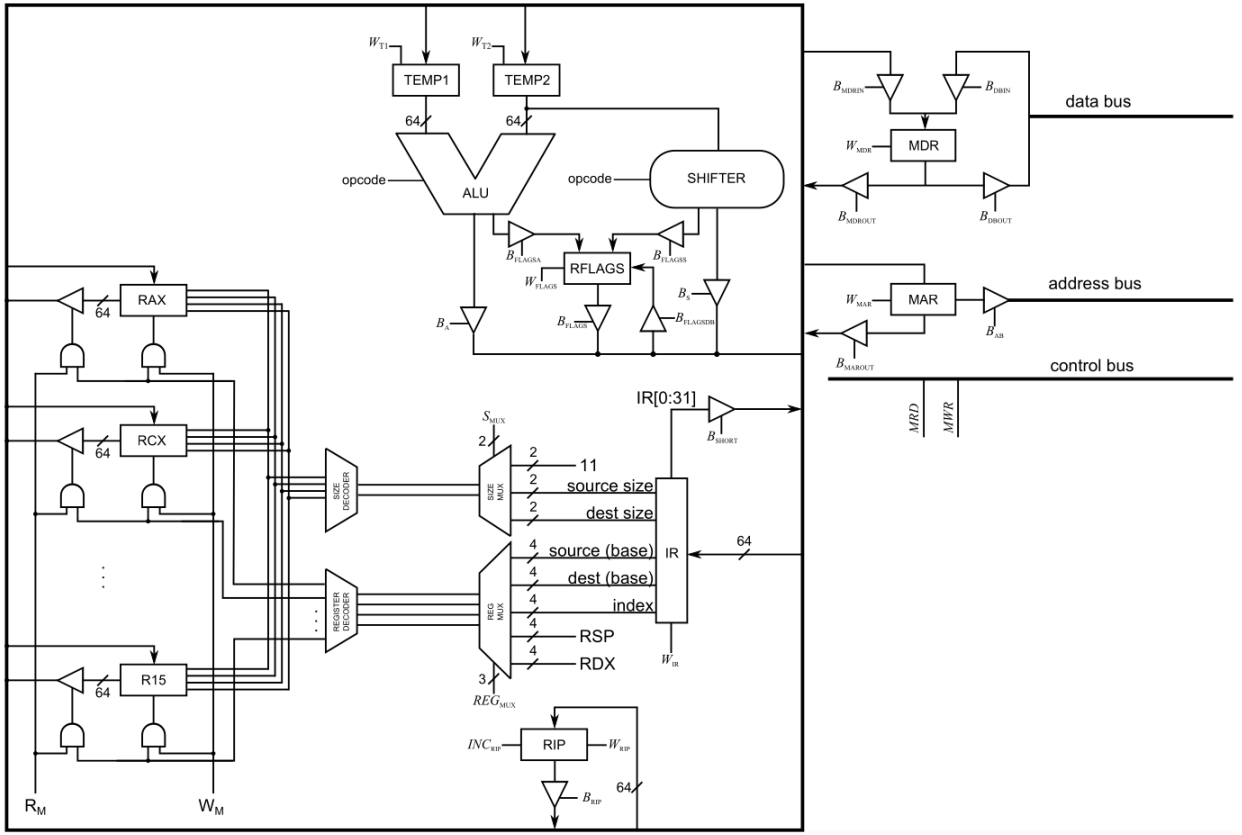


- Il tempo richiesto dalla memoria per fornire un dato determina la *latenza di accesso*

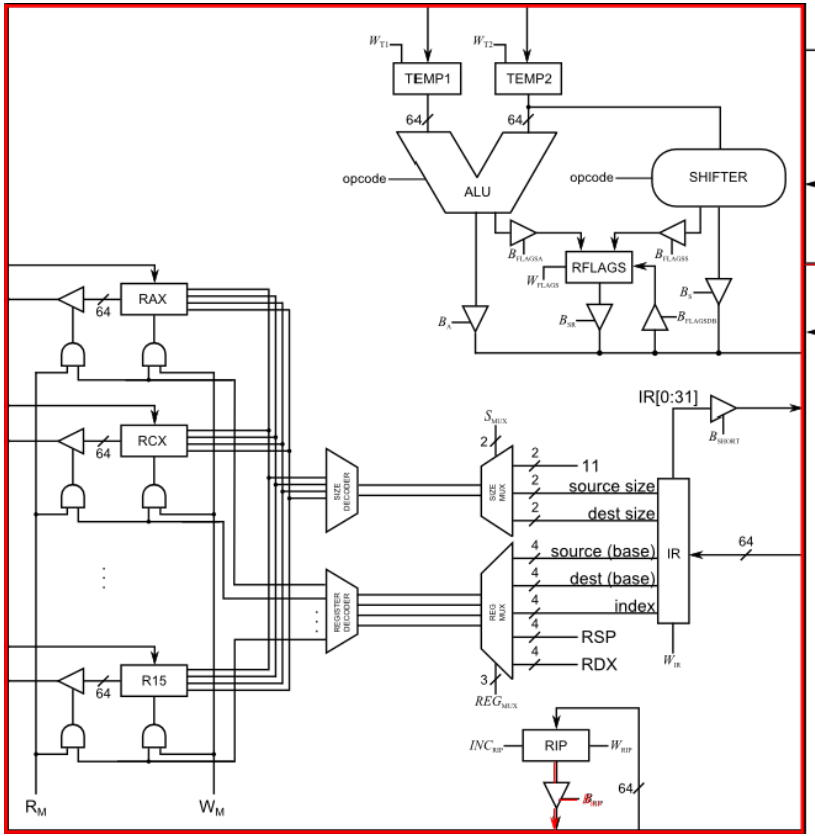


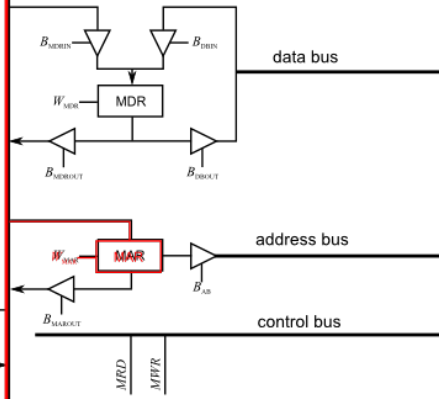


Latenza di accesso: la fase di fetch



Latenza di accesso: la fase di fetch



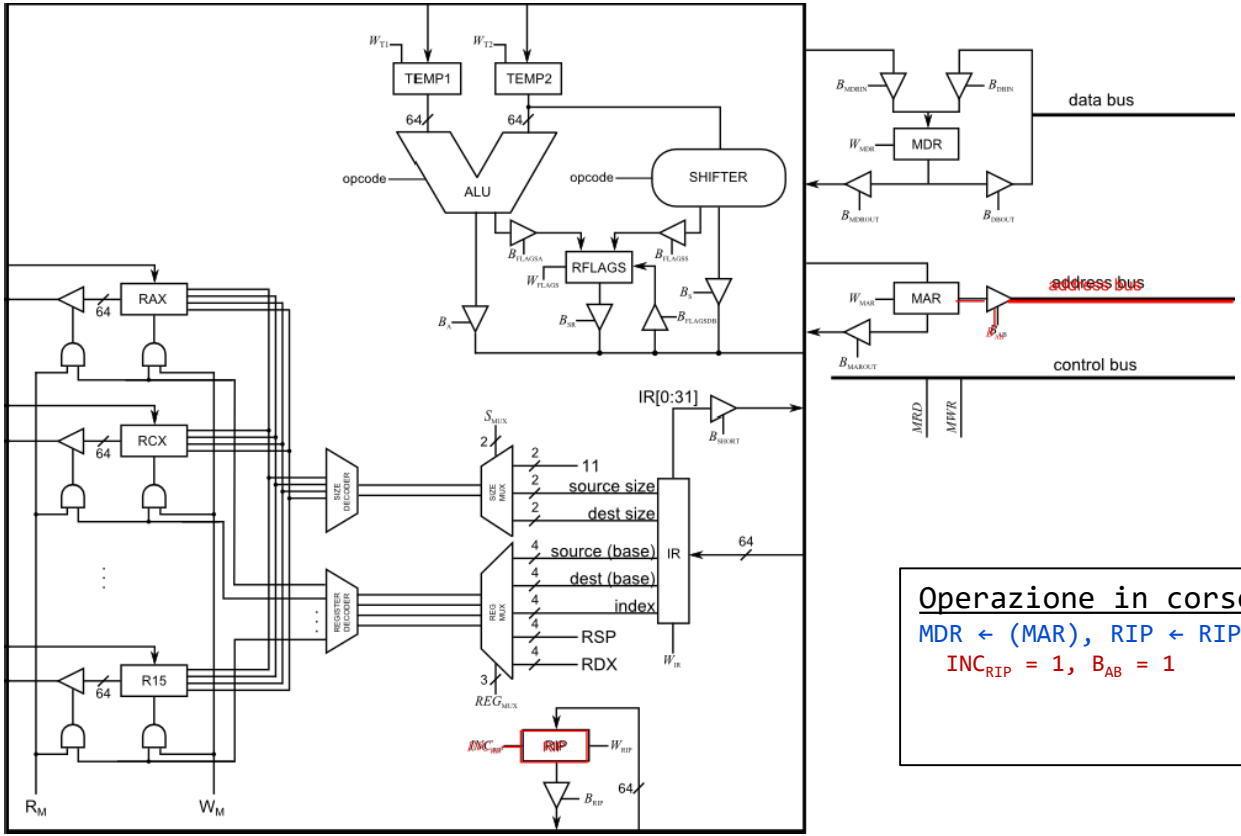


Operazione in corso:

MAR ← RIP:

$B_{RIP} = 1$, $W_{MAR} = 1$

Latenza di accesso: la fase di fetch

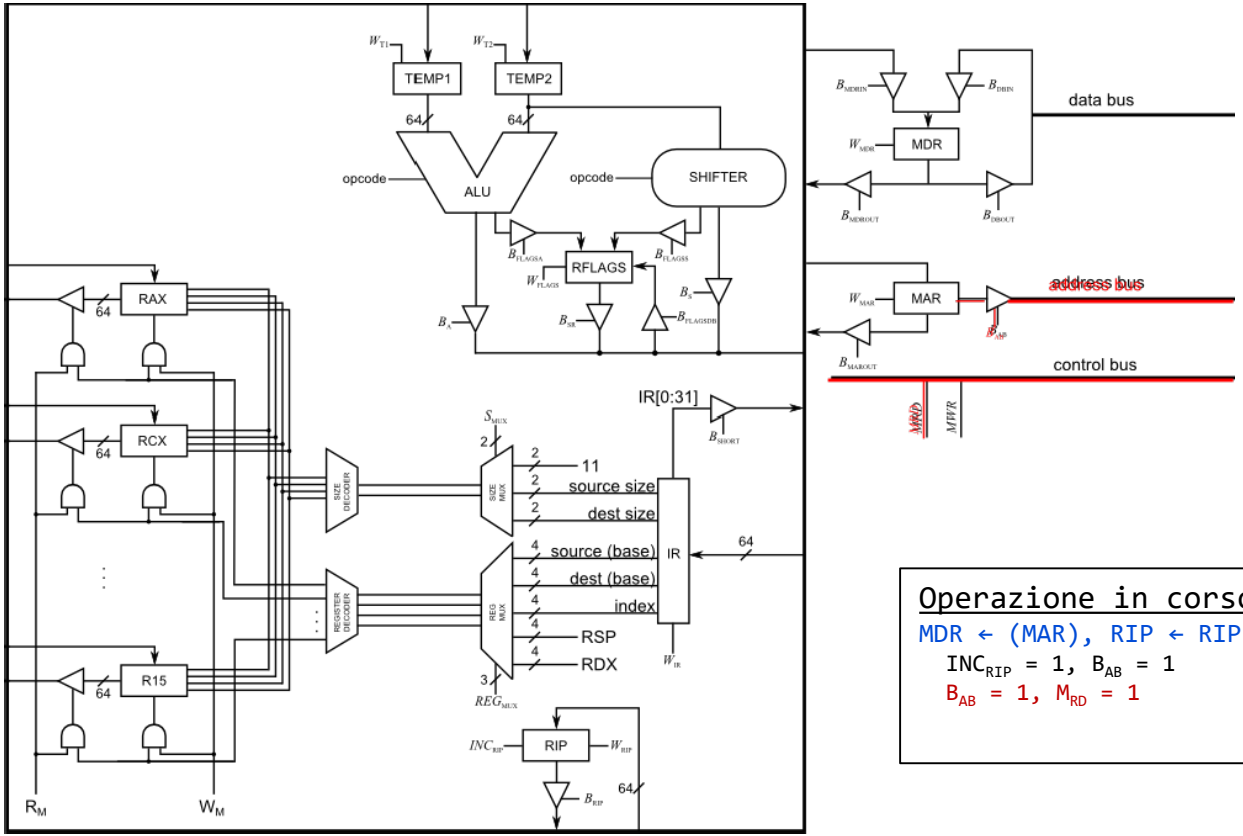


Operazione in corso:

MDR ← (MAR), RIP ← RIP + 8:

INC_{RIP} = 1, B_{AB} = 1

Latenza di accesso: la fase di fetch



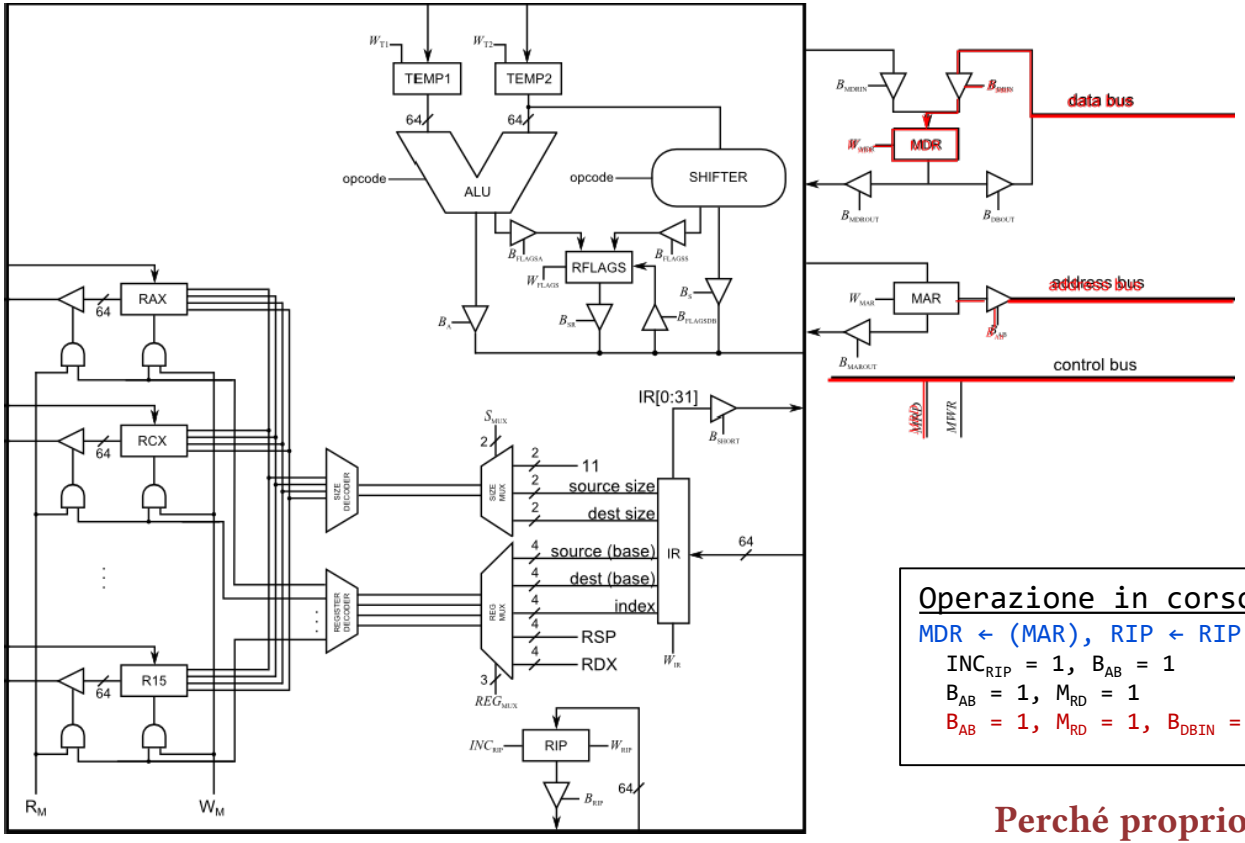
Operazione in corso:

MDR ← (MAR), RIP ← RIP + 8:

INC_{RIP} = 1, B_{AB} = 1

B_{AB} = 1, M_{RD} = 1

La fase di fetch



Operazione in corso:

MDR ← (MAR), RIP ← RIP + 8:

INC_{RIP} = 1, B_{AB} = 1

B_{AB} = 1, M_{RD} = 1

B_{AB} = 1, M_{RD} = 1, B_{DBIN} = 1, WM_{DR} = 1

Perché proprio 3 cicli?

Interazione sincrona con la memoria

- Parametri costruttivi delle memorie determinano la latenza di accesso
- Questi parametri sono alcuni dei fattori di compatibilità tra memoria e CPU

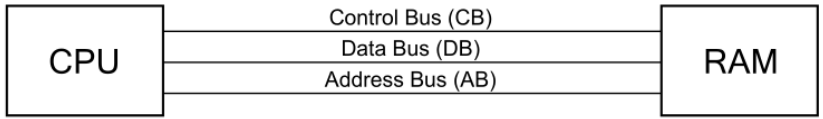
Parametri della temporizzazione della memoria

- L'esecuzione di un comando (lettura/scrittura) è soggetto a dei tempi definiti da alcuni parametri

- **Column Address Strobe (CAS) latency**: il ritardo tra il comando READ e il momento in cui i dati sono disponibili.
  - Espresso in cicli di clock per le DRAM sincrone
  - Espresso in nanosecondi per le DRAM asincrone
- **Ritardo tra indirizzo di riga e indirizzo di colonna**: Il numero minimo di cicli di clock necessari tra l'apertura di una riga di memoria e l'accesso alle colonne al suo interno. Sommato alla CAS latency dà il tempo necessario a leggere il primo bit dal data bus.
- **Tempo di precaricamento di riga**: tempo necessario in caso di conflitto per caricare una riga
- **Tempo di attivazione di riga**: cattura il tempo di refresh

Differenze di velocità tra CPU e DRAM

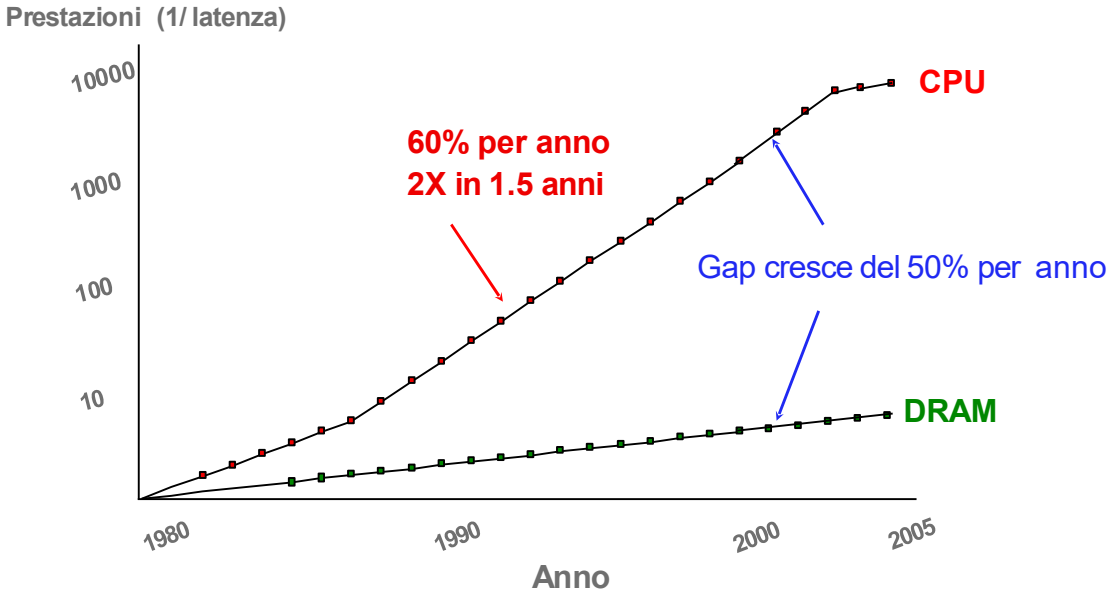
- Nell'architettura di von Neumann il canale di comunicazione tra CPU e memoria è il punto critico (*collo di bottiglia*) del sistema



- La tecnologia consente di realizzare CPU sempre più veloci e memorie sempre più grandi (legge di Moore)
- Tuttavia, la velocità di accesso alle memorie non cresce così rapidamente come la velocità della CPU

Gap prestazionale tra CPU e memoria

- Il gap prestazionale tra memoria e CPU è cresciuto nel tempo
- Come si può risolvere questa differenza?



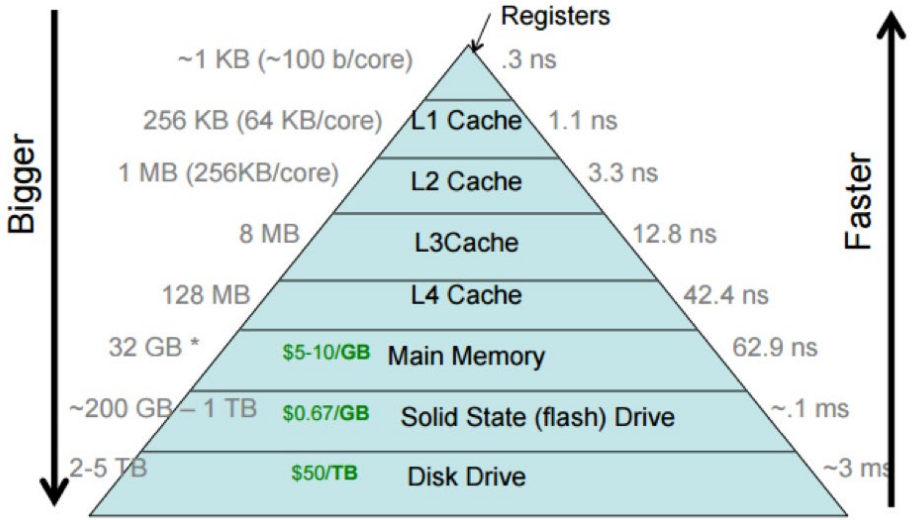
La gerarchia di memoria

- La soluzione ottimale per un sistema di memoria è:
  - costo minimo
  - capacità massima
  - tempo di accesso minimo

- Una soluzione approssimata è quella di implementare una **gerarchia** di memoria:
  - tecnologie diverse per soddisfare al meglio ciscauno dei requisiti
  - la gerarchia cerca di ottimizzare globalmente i parametri

La gerarchia di memoria

- Più livelli di memoria
- Al crescere della distanza dalla CPU decresce il costo e cresce la capacità di memorizzazione
- In ogni istante di tempo i dati sono copiati solamente tra ciascuna coppia di livelli adiacenti
  - Ci si può concentrare su due soli livelli



Latenza di accesso: un esempio

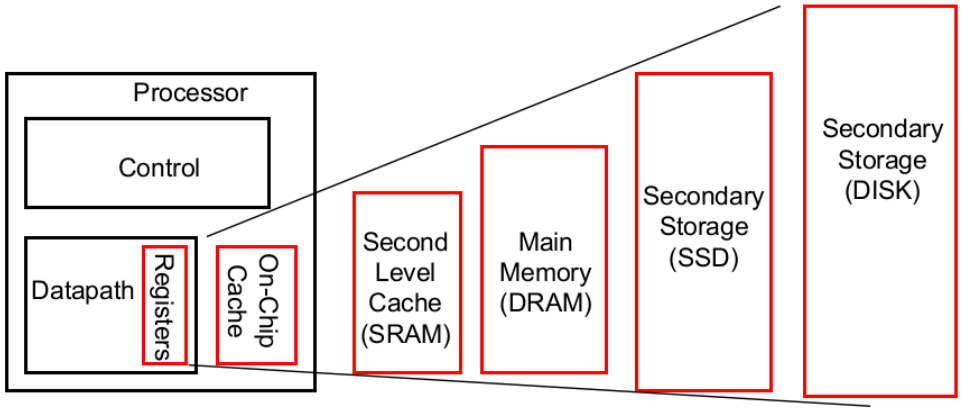
- Intel Core i9-10900K (10th gen), Q2 2020

- Tempi e frequenze *indicativi* per accedere ad un blocco di dati in modo casuale

	Reg.	L1i	L1d	L2	L3	RAM	Disco
Dimensione	1K	32 kB	32 kB	256 kB	20 MB	128GB	TB
Latenza (cicli)	1	4	4	12	57	253	10^7
Latenza (sec)	0,2ns	0,8ns	0,8ns	2,4ns	11,4ns	50,5ns	0,45ms - 2ms
Hz / IOPS	3,70G	1,25G	1,25G	416M	87M	19M	50-100000

Efficienza nell'uso delle cache

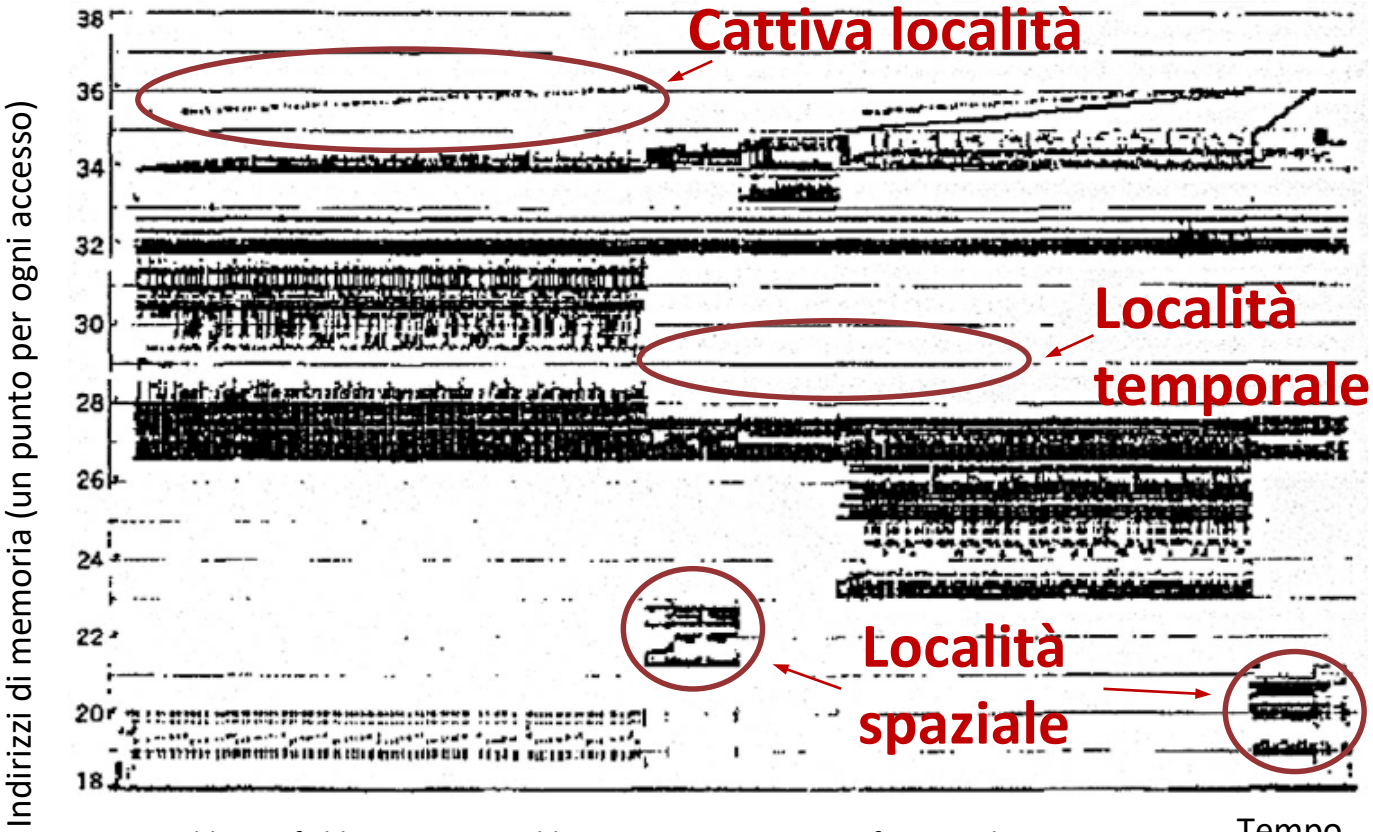
- Una cache (un *deposito*) è più piccola dell'intera memoria principale di lavoro (RAM)
- Le cache mantengono *sottoinsiemi* dei dati (copie)
- Come possiamo aumentare le probabilità di trovare i dati che ci servono "vicini" alla CPU?



Principio di località

- Un sistema di memoria gerarchico è *efficiente* solo se gli schemi di accesso ai dati sono prevedibili
  - L'efficienza dipende quindi fortemente dall'applicazione!
- **Principio di località**: *in un dato istante, un programma tende ad accedere ad una porzione relativamente piccola del suo spazio di indirizzamento* (working set) *sia per quanto riguarda i dati che le istruzioni*
- **Località temporale**: se si è effettuato un accesso ad un elemento, è *probabile* che nel breve tempo si effettuerà un altro accesso allo stesso elemento (esempio: le istruzioni in un ciclo)
- **Località spaziale**: se si è effettuato un accesso ad un elemento, è *probabile* che nel breve tempo si effettuerà un accesso ad un elemento vicino (esempio: accesso sequenziale ad un vettore)

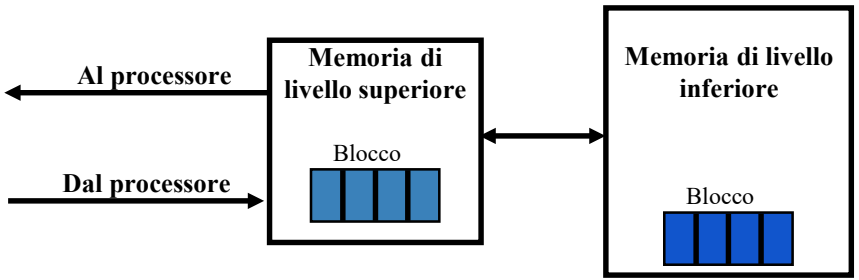
Principio di località



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168192 (1971)

Come sfruttare il principio di località

- Per sfruttare la *località spaziale*:
  - spostare blocchi contigui tra livelli della gerarchia
- Per sfruttare la *località temporale*:
  - tenere i blocchi acceduti più frequentemente vicino al processore



Alcune definizioni

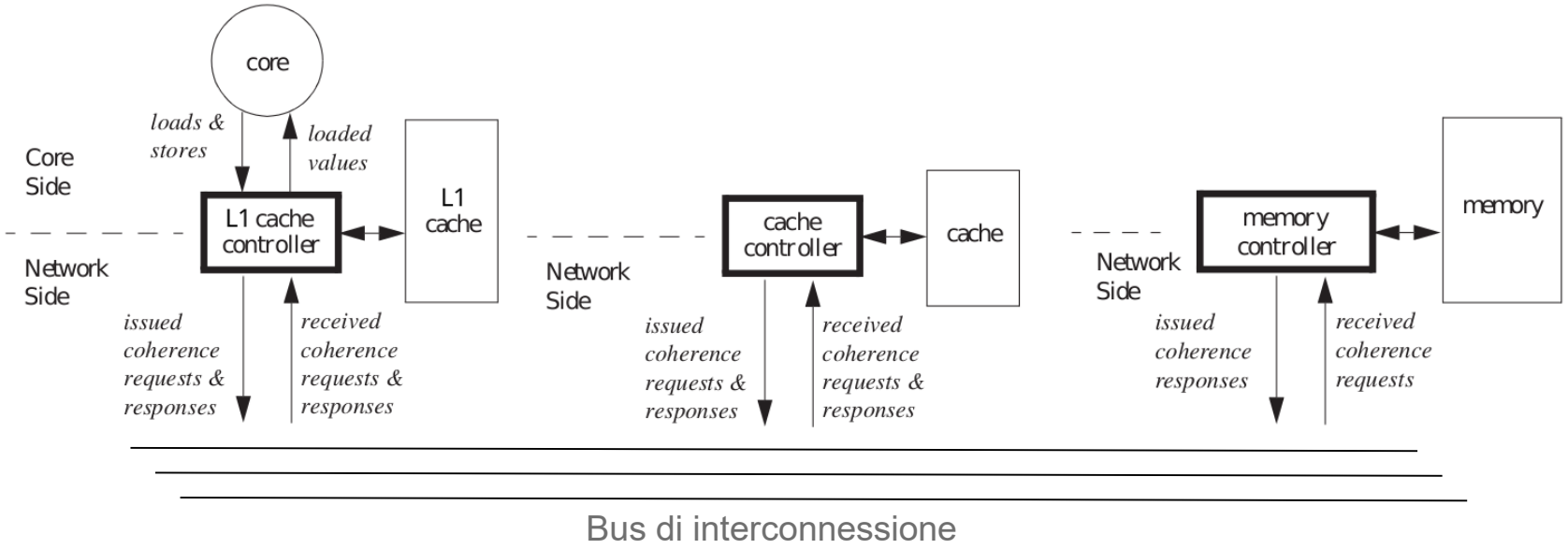
- **Hit rate** (frequenza di successo): frazione degli accessi in memoria risolti in un dato livello della gerarchia

  Hit rate = numero di hit / numero di accessi a memoria

- **Miss rate** (frequenza di fallimento): 1 - hit rate
- **Hit time** (tempo di successo): tempo di accesso al livello in caso di successo
- **Miss penalty** (penalità di fallimento): tempo per trasferire il blocco dal livello inferiore della gerarchia
- **Miss time** (tempo di fallimento): tempo per ottenere l'elemento in caso di miss

  miss time = miss penalty + hit time

Cache e controllori della memoria



Strategie di utilizzo delle cache

- La cache è strutturata in **linee**
  - Ogni linea contiene un **blocco** (tipicamente 64 byte), ossia la *minima unità di informazione* che viene trasferita tra livelli
- Il processore interagisce soltanto con il **controllore della cache di primo livello**
- Se il dato richiesto si trova nella cache di primo livello, si ha un **cache hit**
- Se il dato richiesto non è presente, si verifica un **cache miss**
  - Il controllore di livello L1 “richiede” al controllore di livello L2 il dato
  - Può verificarsi un hit o un miss nel livello L2
  - In caso di hit in L2, il dato viene copiato in L1
  - In caso di miss in L2, il controllore di livello L2 “richiede” il dato al controllore di livello L3
  - Lo stesso schema si verifica per l’accesso alla RAM

Politiche di inclusività nelle cache

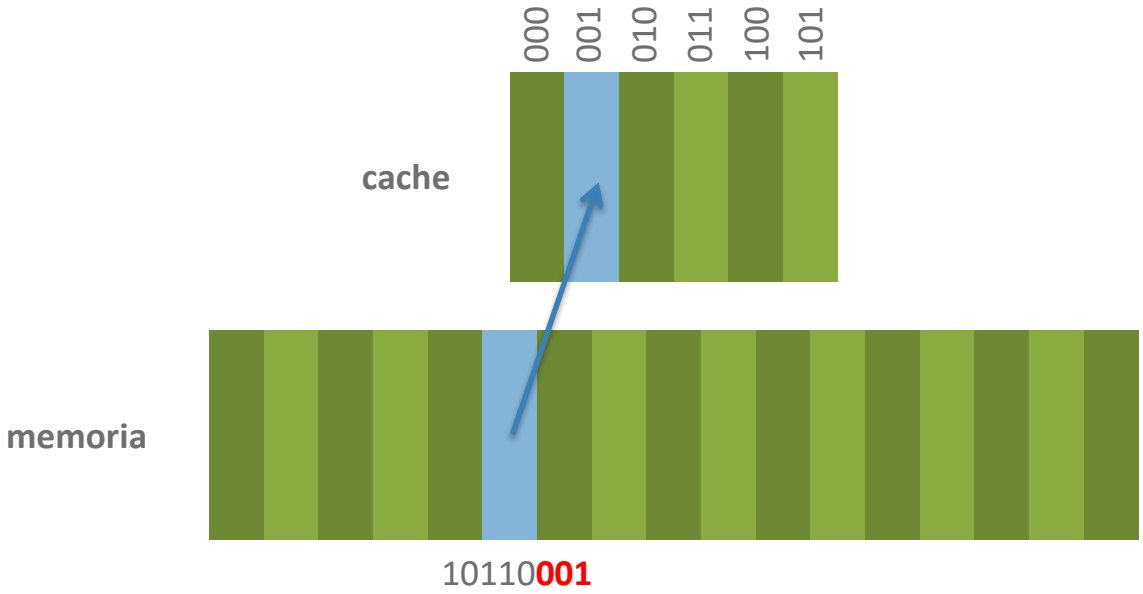
- Cache **inclusive**:
  - Un livello superiore della gerarchia (più vicino al processore) contiene un sottoinsieme di informazioni dei livelli inferiori
  - Tutte le informazioni sono memorizzate nel livello più basso
  - Solo il livello massimo di cache (L1 cache) è acceduto direttamente dal processore
- Cache **non inclusive** (o esclusive):
  - Alcuni livelli possono essere *non inclusivi*
  - Un miss ad un livello superiore (es L1), può comportare l'accesso diretto in memoria
  - In questo caso, il dato è scritto solo in L1, ma non in L2
  - Se per scrivere in L1 è necessario rimpiazzare un blocco, questo "scende" in L2

Informazioni di controllo

- Una linea di cache non contiene solo il blocco, ma anche **informazioni di controllo** (la *directory*)
- La dimensione in bit di una linea di cache è quindi maggiore della dimensione del blocco stesso
- Le informazioni di controllo consentono di implementare politiche di accesso alle cache più efficienti
- Il *protocollo* di gestione delle cache determina la quantità di informazioni di controllo che sono necessarie per ciascuna linea di cache

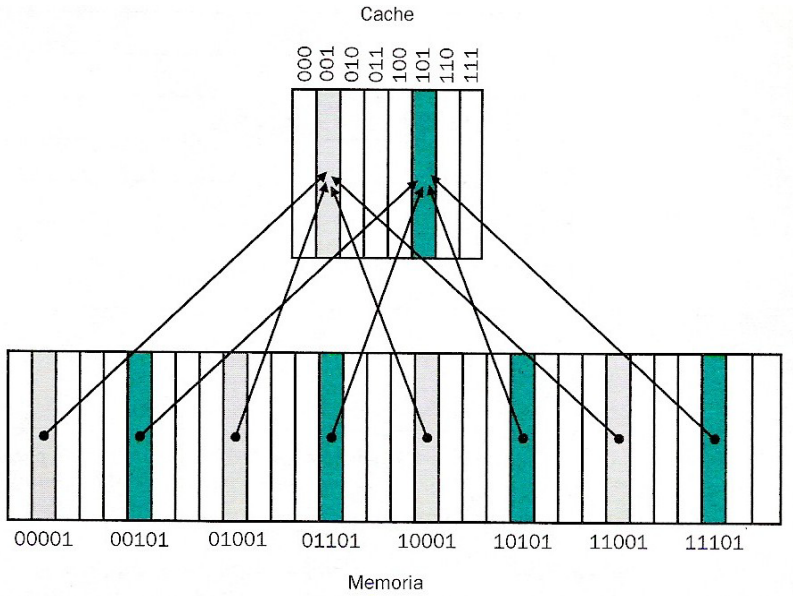
Determinare la posizione del blocco

- Una cache è più piccola della memoria
- Come decidiamo in quale linea memorizzare un blocco?
- Strategia **direct mapping**: si utilizza una *funzione hash basata sull'estrazione*:
  - esempio: prendo gli n bit meno significativi dell'indirizzo del blocco
- Funziona perché le cache hanno tipicamente una dimensione 2^n



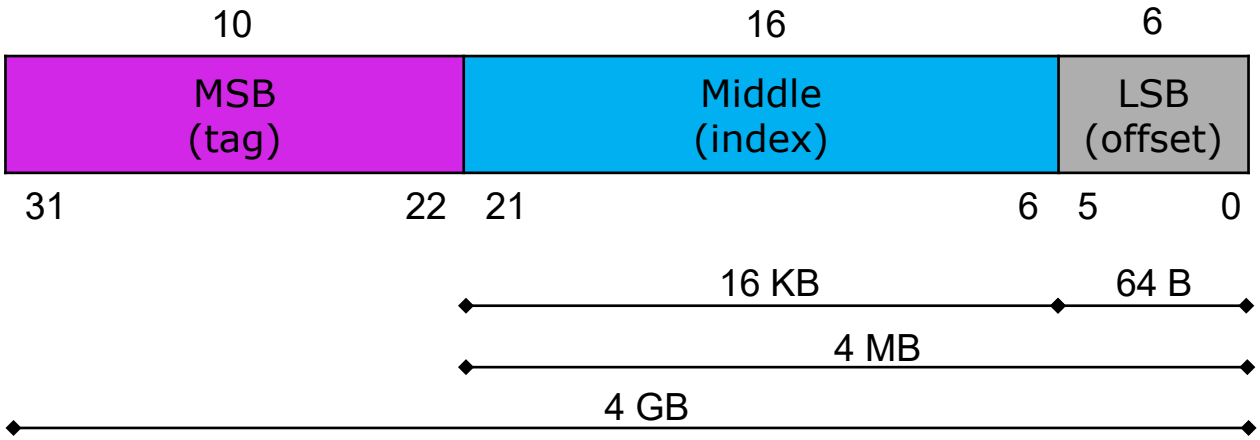
Problema: collisione sulle linee di cache

- Ad una posizione in cache, sono associate più parole di memoria (*sinonimi*)
- Come è possibile sapere se il dato presente nella cache corrisponde effettivamente alla parola richiesta?
- Soluzione: utilizzare un **tag** per risolvere le collisioni

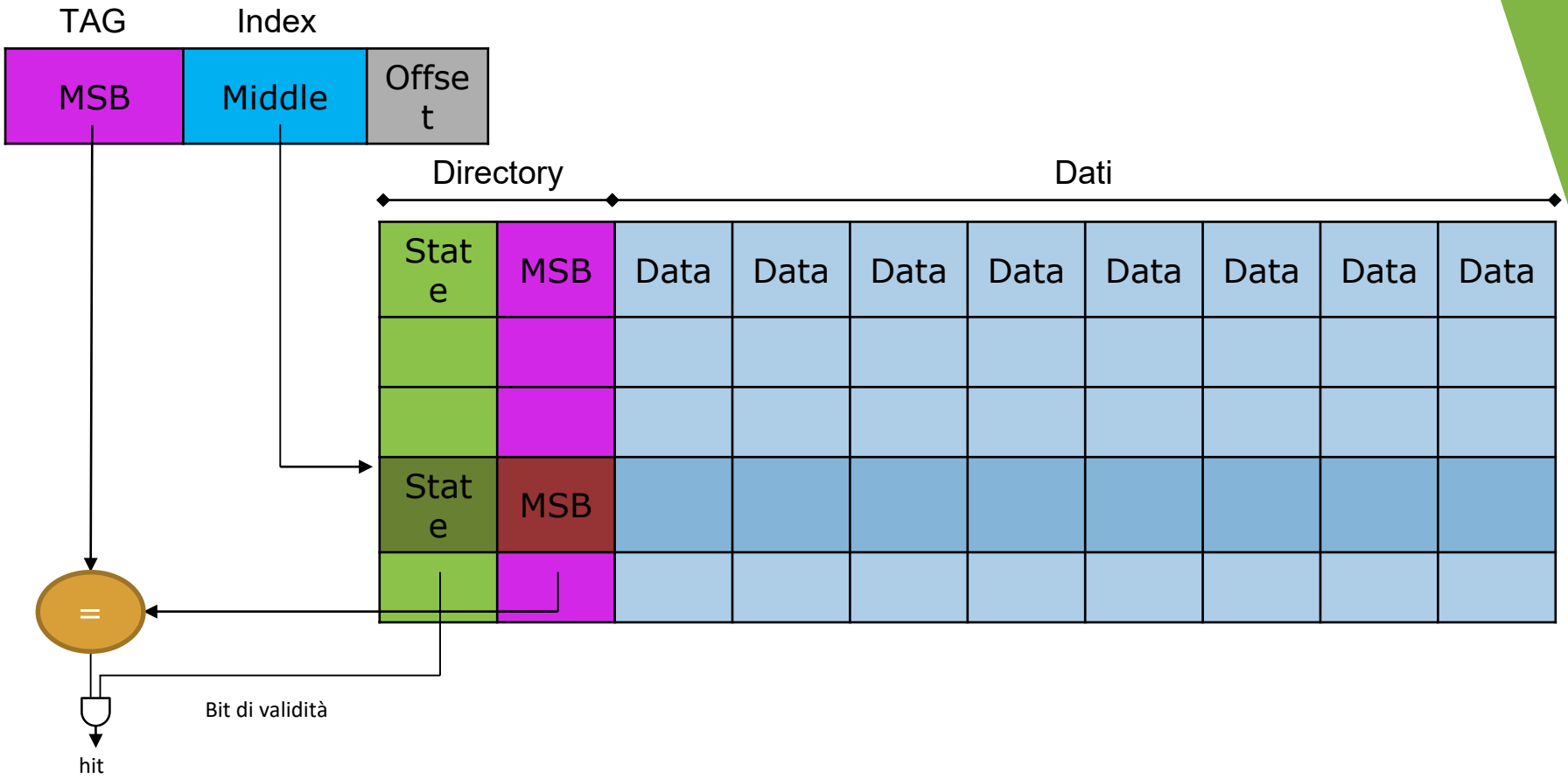


Direct Mapped Cache (cache ad accesso diretto)

- L'indirizzo di memoria viene scomposto in tre parti:
  - Tag: i bit più significativi, per discriminare *sinonimi*
  - Index: bit intermedi, per selezionare una linea di cache
  - Offset: i bit meno significativi per selezionare la parola esatta all'interno del blocco



Direct Mapped Cache: accesso in lettura



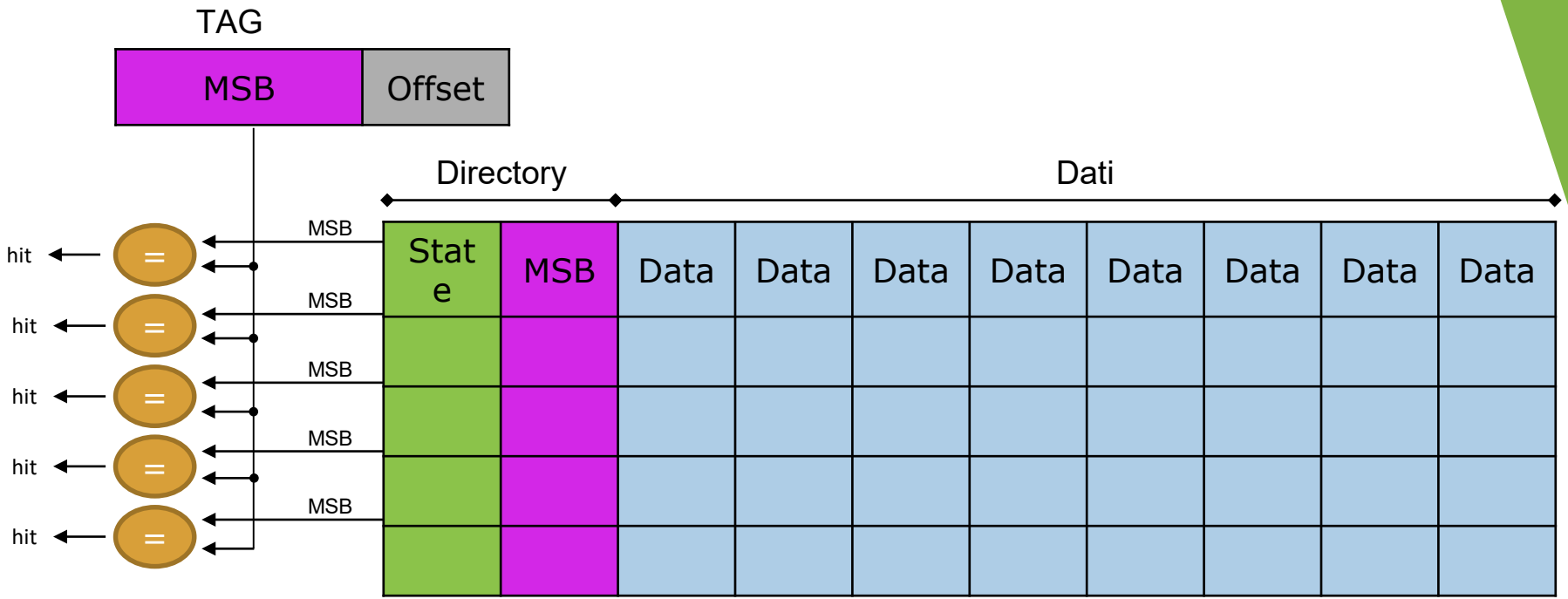
Cache completamente associativa

- Nelle cache completamente associative (*fully associative*), ogni blocco può essere memorizzato in qualsiasi linea di cache

- **Vantaggi**:
  - utilizzo massimo della cache
  - riduzione dei conflitti

- **Svantaggi**:
  - Per trovare un blocco, occorre cercarlo in tutta la cache
  - La ricerca sequenziale aumenterebbe troppo la latenza
  - È necessario cercare in parallelo: si aggiunge un comparatore per ciascuna linea
  - Il costo è molto elevato

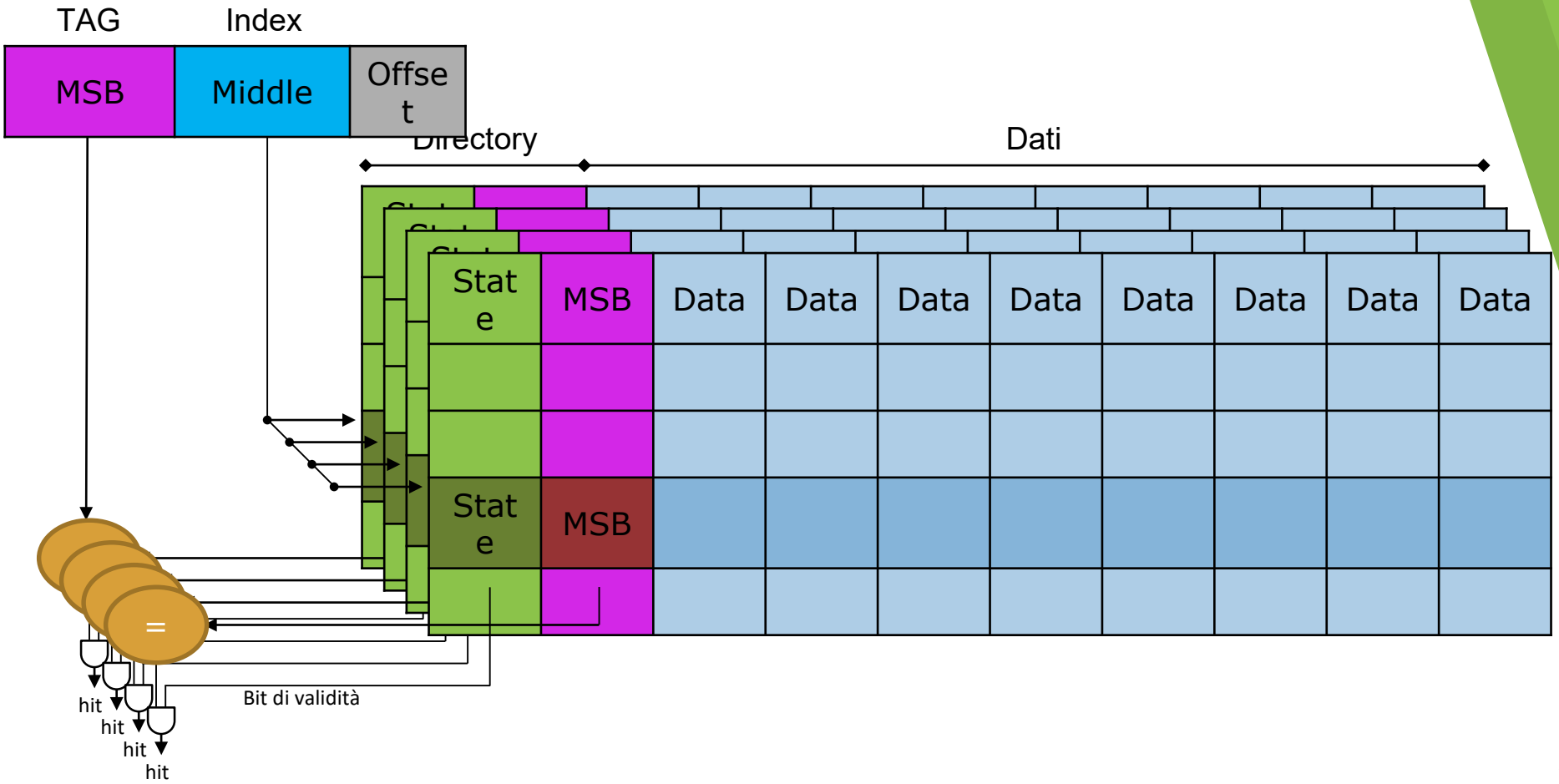
Fully Associative Cache: accesso in lettura



Cache set associative

- Le cache ad indirizzamento diretto il numero di conflitti può essere elevato
- Le cache completamente associative hanno un utilizzo migliore, ma un costo elevato
- Un buon compromesso dalle cache **set associative** (o *n-way set associative*):
  - ciascun blocco può essere memorizzato in un numero di linee $n \geq 2$ (un *insieme*)
  - la ricerca di un blocco richiede il confronto di al più n tag
  - Il costo risulta più accettabile
  - è la tecnologia tipicamente utilizzata dalle cache dei processori moderni

4-way Set Associative Cache: accesso in lettura



Politiche di rimpiazzo (eviction)

- Quando è disponibile più di una entry per memorizzare un dato (completamente associativa e set associativa) è necessario scegliere una *vittima* per il rimpiazzo
- Le principali politiche sono:
  - **Least Recently Used** (LRU): nello stato viene mantenuto un *contatore di età*: viene resettato quando si carica un blocco, ad ogni accesso i contatori del set sono incrementati di uno
  - **First-in First-out** (FIFO): come LRU, ma l'incremento avviene solo in caso di rimpiazzo
  - **Least Frequently Used** (LFU): di difficile implementazione
  - **Round Robin**: usato nelle cache completamente associative. Utilizza un puntatore per tenere traccia della prossima linea da rimpiazzare, aggiornato ad ogni rimpiazzo
  - **Random**: come round robin, ma il puntatore viene aggiornato ad ogni accesso

Gestione delle operazioni di scrittura

- Il processore interagisce soltanto con la cache L1
- Le cache mantengono *copie* dei dati
- Quando si aggiorna un blocco, è necessario rendere coerenti le altre copie
  - Questo vuol dire che non si può scrivere in memoria se prima non si è caricato in cache il dato!
- Due principali strategie:
  - **Write through**: ad ogni scrittura in L1, le copie nei livelli inferiori vengono aggiornate immediatamente
    - Di facile implementazione
    - Offre delle prestazioni nettamente minori
  - **Write back**: le copie nei livelli inferiori vengono aggiornate solo quando si effettua un rimpiazzo
    - Di più complessa implementazione
    - Le prestazioni sono nettamente superiori

Prestazioni delle cache

- Il tempo di esecuzione di un programma può essere approssimato come:

$$T_{exec} = n_{istruzioni} \cdot CPI \cdot \frac{sec}{ciclo}$$

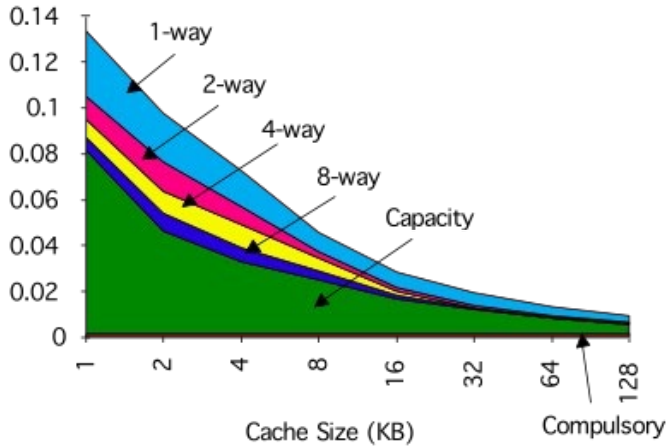
- Il numero di *cicli di clock per istruzione* (CPI) dipende dal Tempo Medio di Accesso in Memoria (TMAM):

$$TMAM = Hit\ Time\ +\ (Miss\ Rate \cdot Miss\ Penalty)$$

- Se si riesce a ridurre TMAM, probabilmente possono aumentare le prestazioni del programma
- Cause di miss:
  - miss obbligatori
  - miss di conflitto
  - miss di capacità

Miss di conflitto

- Nel caso di cache n-way associative, se sono utilizzati M blocchi in conflitto ma N < M, si genera un miss di conflitto
- Prima soluzione: aumentare n

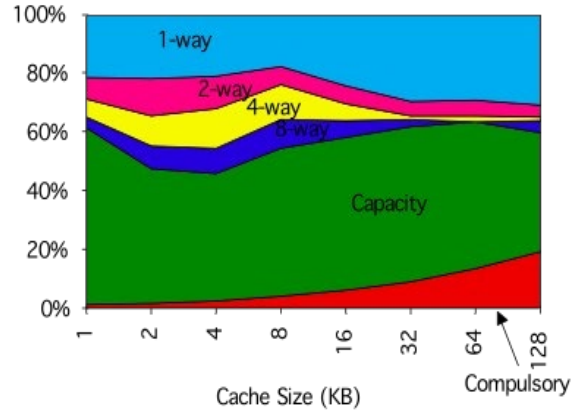


- Seconda soluzione: aumentare la dimensione del blocco
  - perché funziona?

Altre cause di miss

- Miss **obbligatori**: la prima volta che si accede un dato, quello non sarà certamente presente in cache
  - Soluzione: *cache prefetching*. Si cerca di "indovinare" i blocchi che potranno essere richiesti in futuro
- Miss di **capacità**: la cache non può contenere tutti i blocchi richiesti dal programma
  - Soluzione: aumentare la dimensione delle cache
  - Ovviamente, il costo aumenta!

Percentuale di miss rate dei vari tipi



Miss rate e dimensione dei blocchi (cache set-associativa)

- Domanda: perché al crescere della dimensione del blocco il miss rate prima diminuisce e poi ritorna a crescere?