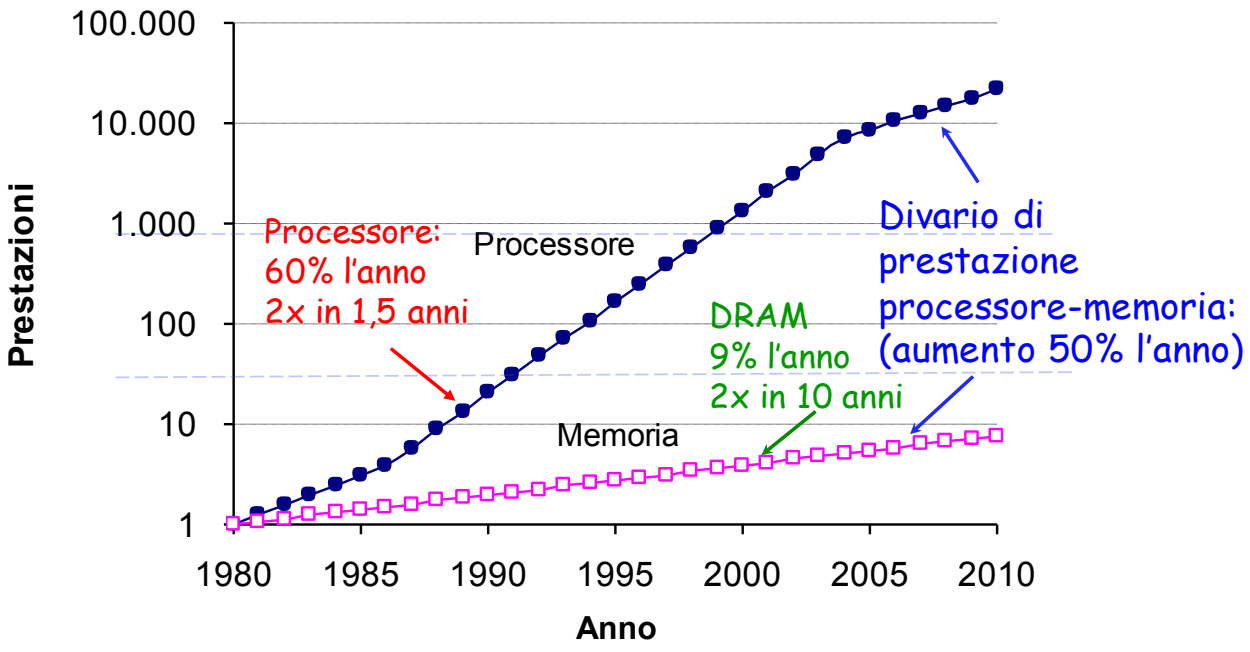
La Gerarchia di Memorie

Valeria Cardellini

Corso di Calcolatori Elettronici
A.A. 2019/20

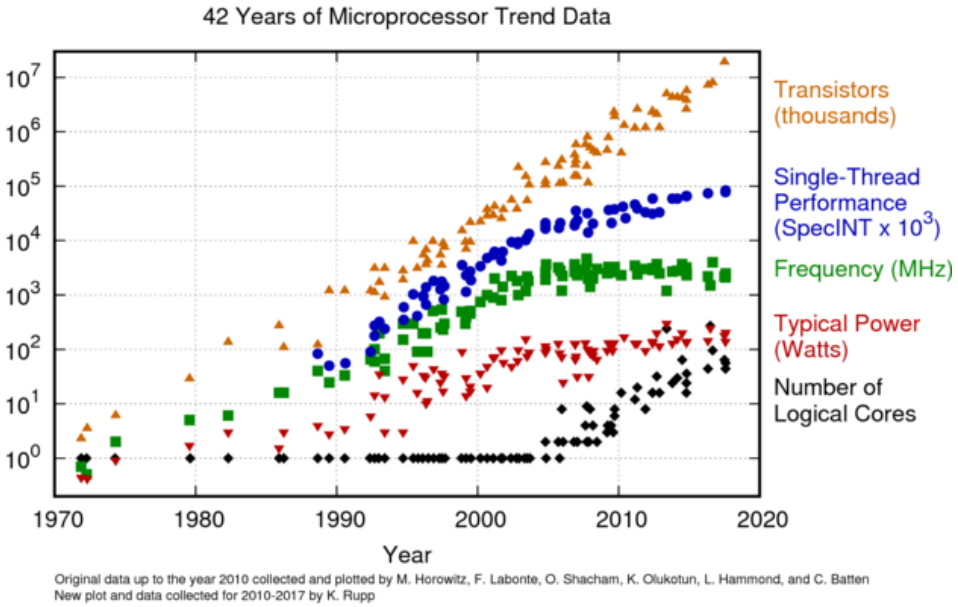
Divario delle prestazioni processore-memoria

- Il cosiddetto **memory wall**



Divario delle prestazioni processore-memoria

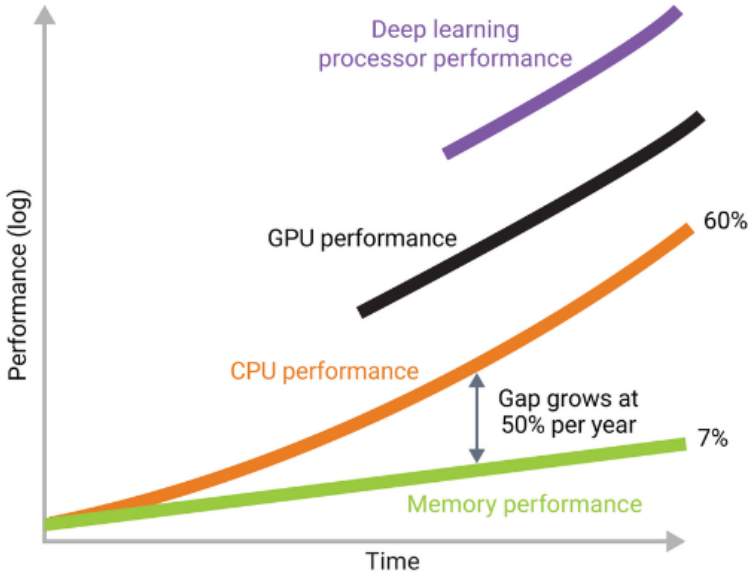
- … e il divario non è diminuito e non diminuirà



Source: https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Divario delle prestazioni processore-memoria

- … in particolare per applicazioni di AI (artificial intelligence)



Source: https://medium.com/@abruyns/memory-holds-the-keys-to-ai-adoption-5acd5e06508b

Divario delle prestazioni processore-memoria

- Soluzione: interporre delle memorie cache più piccole e veloci tra processore e DRAM
- Si viene a creare una **gerarchia di memorie**

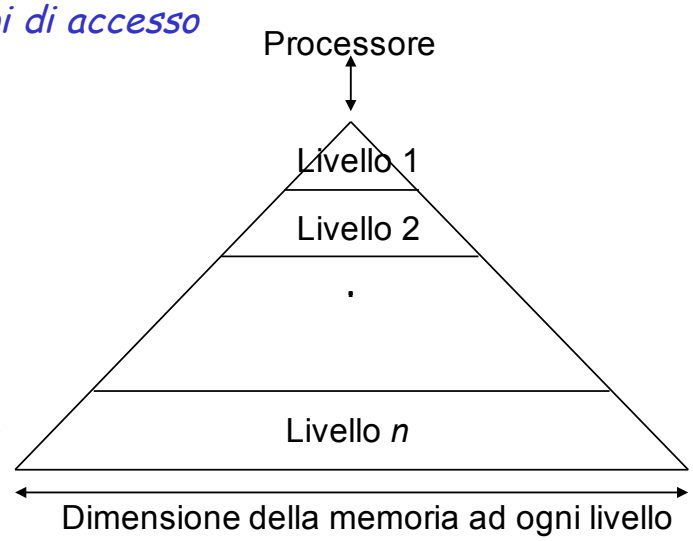
Obiettivo

- Illusione di avere a disposizione una memoria:
  - veloce (ritardo simile a quello del processore)
  - grande
  - economica
- Osservazioni:
  - Le memorie di grandi dimensioni sono lente
  - Le memorie veloci hanno dimensioni piccole
  - Le memorie veloci costano (molto) più di quelle lente
  - Non esiste una memoria che soddisfi simultaneamente tutti i requisiti!
- Come creare una memoria che sia grande, economica e veloce (per la maggior parte del tempo)?
  - **Gerarchia**
  - **Parallelismo**

La soluzione: gerarchia di memorie

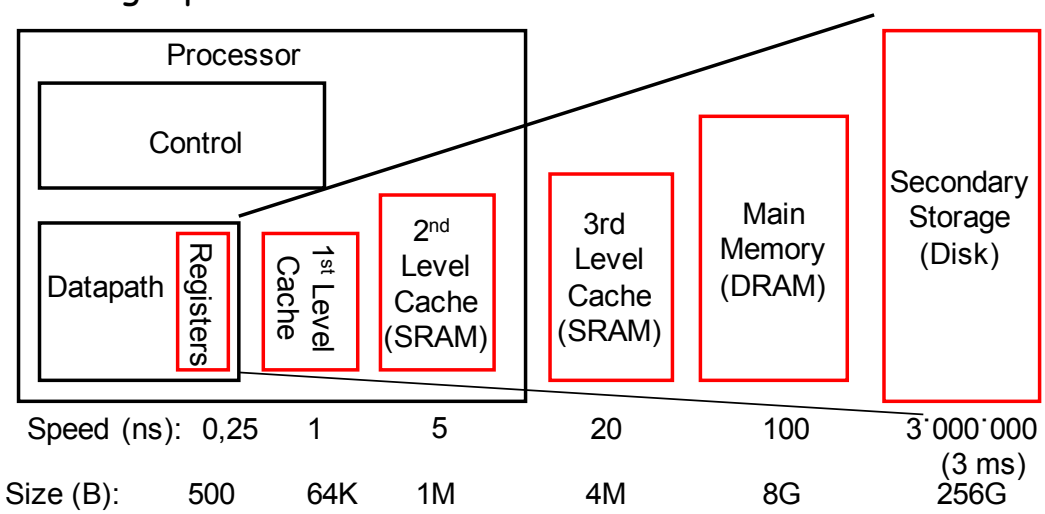
- Non un livello di memoria…
- Ma una **gerarchia** di memorie
  - *Ognuna caratterizzata da differenti tecnologie, costi, dimensioni e tempi di accesso*

- Aumenta il tempo di accesso
- Aumenta la capacità di memorizzazione
- Diminuisce il costo per bit



La soluzione: gerarchia di memorie

- Obiettivi della gerarchia di memorie:
  - Fornire una quantità di memoria pari a quella disponibile nella tecnologia più economica
  - Fornire una velocità di accesso pari a quella garantita dalla tecnologia più veloce

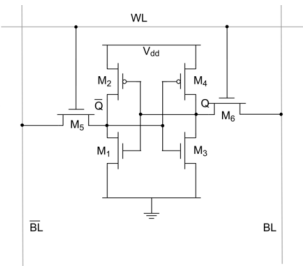


Tecnologie di memoria

- Memorie SRAM
- Memorie DRAM
- Memorie flash
- Dischi rigidi

SRAM

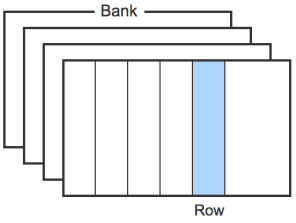
- **SRAM** (*Static Random Access Memory*)
  - Tipo di memoria RAM volatile
    - ✓ Tempo di accesso costante (indipendente dalla locazione della cella di memoria)
  - Organizzata come array di memoria
    - ✓ Singola cella di memoria corrisponde a flip-flop di tipo latch e richiede da 4 a 6 transistor
    - ✓ Da 6 a 8 transistor per bit (2 per controllare l'accesso alla cella)
  - Non richiede refresh dei dati
    - ✓ Tempo di accesso simile al tempo di ciclo
  - Bassi consumi
- Rispetto a DRAM, SRAM è:
  - Più veloce (no refresh)
  - Più costosa per bit (6 transistor per bit)
  - A parità di bit, occupa un'area maggiore



Cella di memoria SRAM

DRAM

- **DRAM** (*Dynamic Random Access Memory*)
  - Tipo di memoria RAM volatile
    - ✓ Tempo di accesso costante
  - Bit memorizzato come carica in un condensatore
    - ✓ 1 transistor per bit (per controllare l'accesso alla cella)
  - Richiede un circuito aggiuntivo per il refresh periodico della carica
    - ✓ Occorre riscrivere periodicamente il contenuto della cella di memoria perché il condensatore non mantiene la carica per un tempo indefinito
  - Rispetto a SRAM, maggiore densità e minor costo per bit
  - Organizzata in banchi
    - ✓ Ogni banco composto da una serie di righe che vengono rinfrescate



DRAM

- **SDRAM**: memoria DRAM sincrona
  - Attende il segnale di clock per rendere effettivi i cambiamenti
  - Prestazioni migliori

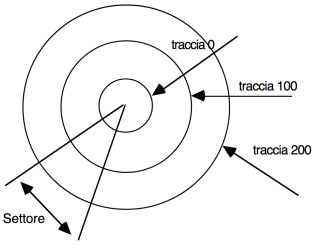
- SDRAM DRR (Double Data Rate)
  - Ulteriore miglioramento della SDRAM
    - ✓ I dati vengono trasmessi sia sul fronte di salita che sul fronte di discesa del segnale di clock

Memorie flash

- Memoria di tipo EEPROM (*Electrically Erasable Programmable Read-Only Memory*)
- Non è volatile
  - Informazioni memorizzate in un vettore di floating gate MOSFET
    - ✓ Transistor ad effetto di campo in grado di mantenere la carica elettrica per un tempo lungo
    - ✓ Ogni transistor costituisce una "cella di memoria" che conserva il valore di un bit
    - ✓ Memorie flash più recenti utilizzano delle celle multilivello che permettono di registrare il valore di più bit attraverso un solo transistor
- Costa leggermente meno della DRAM
- Rispetto a DRAM, più lenta in lettura e decisamente più lenta in scrittura

Disco rigido (hard disk)

- Costituito da uno o più piatti, con due superfici, in rapida rotazione intorno a un perno centrale e da due testine per piatto
- Ciascuna superficie ha una serie di cerchi concentrici o tracce e viene suddivisa in spicchi radiali di ugual grandezza, chiamati settori
  - Un disco nuovo non è suddiviso in tracce e settori, perché diversi sistemi operativi possono richiedere il loro tipo particolare di suddivisione
  - Il processo di suddivisione del disco in tracce e settori è detto formattazione



Disco rigido (hard disk)

- La testina si sposta longitudinalmente lungo le tracce
- Tutte le tracce equidistanti dal centro formano un cilindro
- I dati sono scritti occupando posizioni successive lungo le tracce
  - Corrispondono a uno stato di polarizzazione (positiva o negativa) del materiale magnetico che costituisce i dischi
- Ogni blocco è selezionabile mediante la terna <superficie, traccia, settore> (indirizzo)

Disco rigido (hard disk)

- Accesso ai dati
  - Spostamento della testina verso la traccia richiesta ➔ tempo di ricerca (*seek time*)
  - Attesa affinché il settore arrivi sotto la testina ➔ latenza di rotazione (*rotational latency*)
    - ✓ Da 5400 a 1500 giri al minuto (RPM)
  - Trasferimento dei dati in (o da) memoria centrale, solitamente eseguito da un processore dedicato (Direct Memory Access, DMA) ➔ tempo di trasferimento (*transfer time*)
- Latenza media di rotazione:

$$\text{Average rotational latency} = \frac{0.5 \text{ rotation}}{5400 \text{ RPM}} = \frac{0.5 \text{ rotation}}{5400 \text{ RPM} \,/\, \left(60 \frac{\text{seconds}}{\text{minute}}\right)}$$

$$= 0.0056 \text{ seconds}$$

$$= 5.6 \text{ ms}$$

Unità di memoria a stato solido

- SSD (Solid State Drive)
  - Dispositivo di memoria di massa basato su semiconduttore, che utilizza memoria allo stato solido (solid-state storage), in particolare memoria flash, per l'archiviazione dei dati
  - All'interno dell'SSD non c'è alcun disco, né di tipo magnetico, né di altro tipo
  - La maggior parte delle unità a stato solido utilizza la tecnologia delle memorie flash NAND, che permette una distribuzione uniforme dei dati e di "usura" dell'unità

Principio di località

- Perché la gerarchia di memoria raggiunge l'obiettivo?
- Principio di ***località***

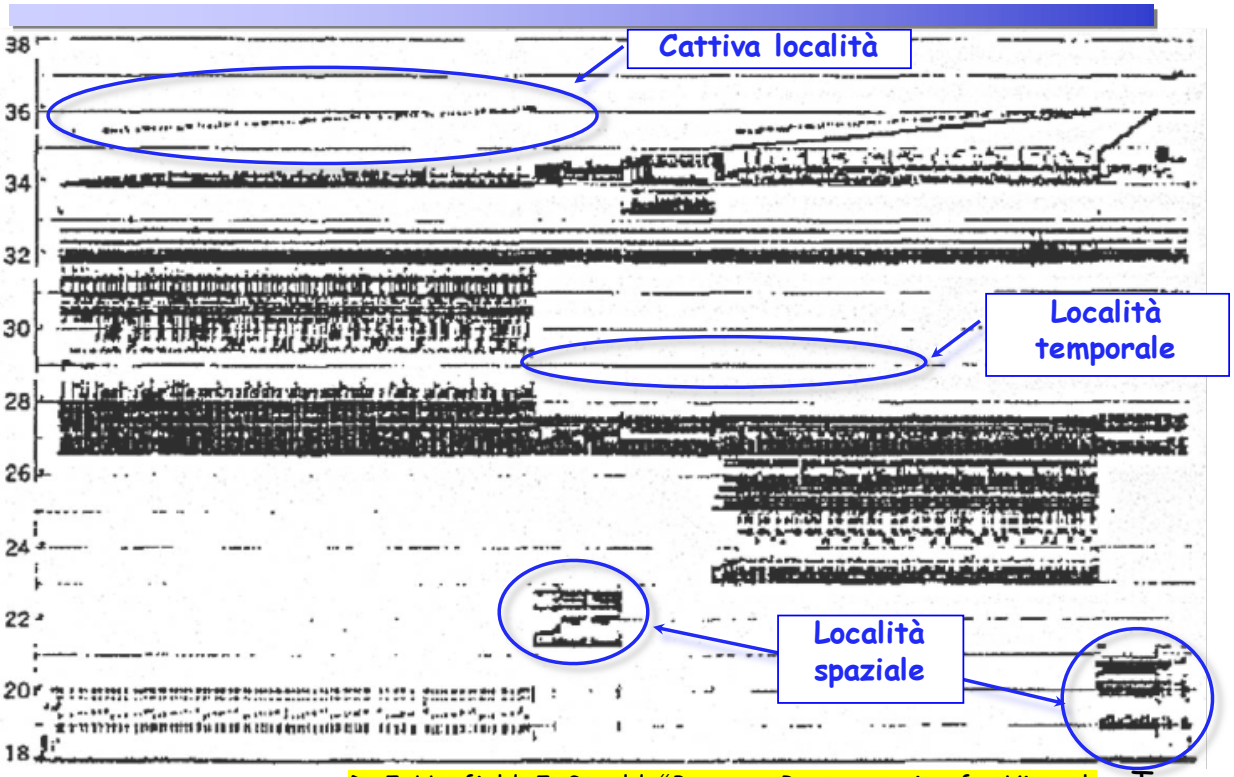
- Esistono due tipi differenti di località
  - Località ***temporale*** (nel tempo)
  - Località ***spaziale*** (nello spazio)

Località temporale e località spaziale

- Località ***temporale*** (nel tempo):
  - Se un elemento di memoria (dato o istruzione) è stato acceduto, tenderà ad essere acceduto nuovamente entro breve tempo
  - *Caso tipico*: le istruzioni ed i dati in un ciclo saranno acceduti ripetutamente

- Località ***spaziale*** (nello spazio):
  - Se un elemento di memoria (dato o istruzione) è stato acceduto, gli elementi i cui indirizzi sono vicini tenderanno ad essere acceduti entro breve tempo
  - *Casi tipici:* gli accessi agli elementi di un array presentano un' elevata *località spaziale*; nell' esecuzione di un programma è altamente probabile che la prossima istruzione sia contigua a quella in esecuzione

Principio di località: un esempio grafico



D. J. Hatfield, J. Gerald, "Program Restructuring for Virtual Memory", IBM Systems Journal 10(3): 168-192, 1971

Principio di località e applicazioni

- La località dipende fortemente dall'applicazione
  - Alta (sia temporale che spaziale) per cicli interni di breve lunghezza che operano su dati organizzati in vettori
  - Bassa nel caso di chiamate ripetute a procedure
- In alcune applicazioni i dati hanno località di un solo tipo
  - Es.: dati di tipo streaming in elaborazione video (non hanno località temporale)
  - Es.: coefficienti usati in elaborazioni di segnali o immagini (si usano ripetutamente gli stessi coefficienti, non c'è località spaziale)

Livelli nella gerarchia di memorie

- Basandosi sul principio di località, la memoria di un calcolatore è realizzata come una gerarchia di memorie
- **Registri**
  - La memoria più veloce, intrinsecamente parte del processore
  - Gestiti dal compilatore (che alloca le variabili ai registri, gestisce i trasferimenti allo spazio di memoria)
- **Cache di primo livello (L1 cache)**
  - Sullo stesso chip del processore, tecnologia SRAM
  - I trasferimenti dalle memorie di livello inferiore sono completamente gestiti dall'hardware
  - Di norma, la cache è trasparente al programmatore e al compilatore (vi sono delle eccezioni)
  - Può essere *unificata* (unica cache per dati e istruzioni) oppure possono esserci cache *separate* per istruzioni e dati (**I-cache e D-cache**)

Livelli nella gerarchia di memorie

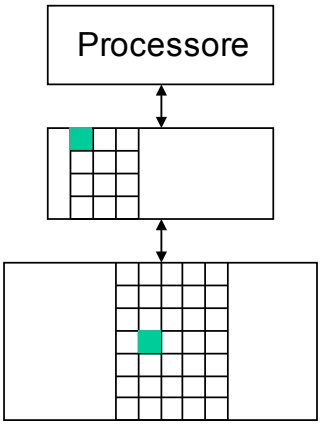
- **Cache di secondo (L2) e terzo livello (L3)**
  - Può essere sia sullo stesso chip del processore (L2 o L3 cache), sia su un chip separato (L3)
  - Tecnologia SRAM
  - Il numero dei livelli di cache e delle loro dimensioni dipendono da vincoli di prestazioni e costo
  - Come per la cache di primo livello, i trasferimenti dalla memoria di livello inferiore sono gestiti dall' hardware e la cache è trasparente al programmatore e al compilatore
- **Memoria RAM**
  - Tecnologia DRAM (SDRAM DDR)
  - I trasferimenti dalle memorie di livello inferiore sono gestiti dal sistema operativo (memoria virtuale) e dal programmatore

Livelli nella gerarchia di memorie

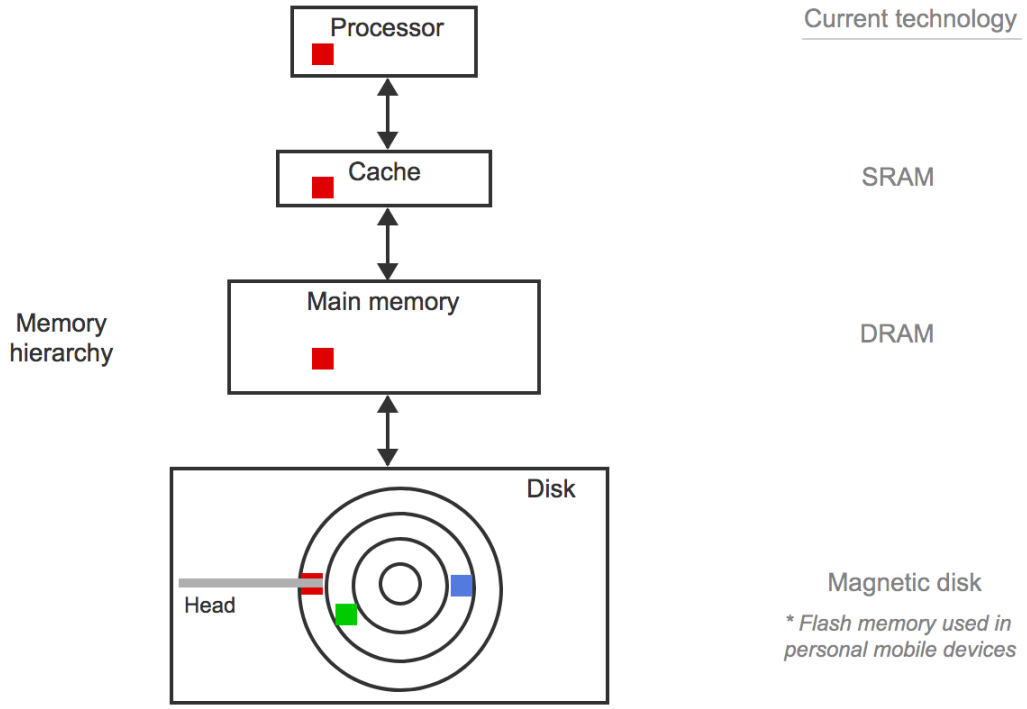
- Livelli di memoria *inclusivi*
  - Un livello superiore della gerarchia (più vicino al processore) contiene un sottoinsieme di informazioni dei livelli inferiori
  - Tutte le informazioni sono memorizzate nel livello più basso
  - Solo il livello massimo di cache (L1 cache) è acceduto direttamente dal processore
- *Migrazione* delle informazioni fra livelli della gerarchia
  - Le informazioni vengono di volta in volta copiate solo tra livelli adiacenti

Migrazione delle informazioni

- ***Blocco***: minima unità di informazione che può essere trasferita tra due livelli adiacenti della gerarchia
  - La dimensione del blocco influenza direttamente la larghezza (banda) del bus
- ***Hit*** (successo): l'informazione richiesta è presente nel livello acceduto
- ***Miss*** (fallimento): l'informazione richiesta non è presente nel livello acceduto
  - Deve essere acceduto il livello inferiore della gerarchia per recuperare il blocco contenente l' informazione richiesta

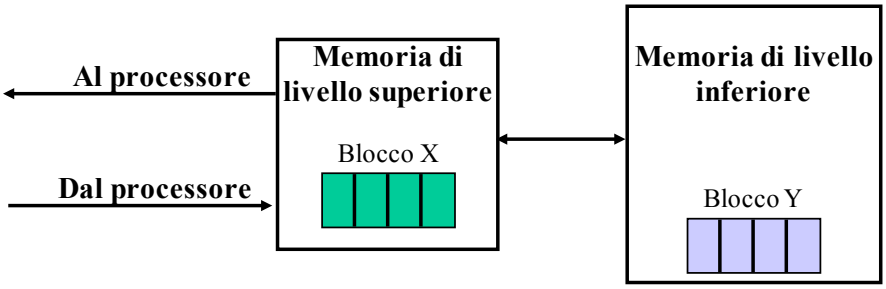


Migrazione delle informazioni



Come sfruttare il principio di località

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



Strategia di utilizzo della cache

- Cache strutturata in ***linee***
  - Ogni linea contiene un blocco (più parole: da 4 a 64 byte)
- La prima volta che il processore richiede un dato in memoria si ha un ***cache miss***
  - Il blocco contenente il dato viene trasferito dal livello inferiore di memoria e viene copiato anche nella cache

Strategia di utilizzo della cache

- Le volte successive, quando il processore richiede l'accesso alla memoria
  - Se il dato è presente in un blocco contenuto nella cache, la richiesta ha successo ed il dato viene passato direttamente al processore
    - ✓ Si verifica un ***cache hit***
  - Altrimenti la richiesta fallisce ed il blocco contenente il dato viene caricato anche nella cache e passato al processore
    - ✓ Si verifica un ***cache miss***

- Obiettivo: aumentare quanto più possibile il tasso di cache hit

Alcune definizioni

- ***Hit rate*** (frequenza dei successi): frazione degli accessi in memoria risolti nel livello superiore della gerarchia di memoria

  Hit rate = numero di hit / numero di accessi in memoria
- ***Miss rate*** (frequenza dei fallimenti): 1 - hit rate
- ***Hit time*** (tempo di successo): tempo di accesso alla cache in caso di successo
- ***Miss penalty*** (penalità di fallimento): tempo per trasferire il blocco dal livello inferiore della gerarchia
- ***Miss time***: tempo per ottenere l'elemento in caso di miss

  miss time = miss penalty + hit time
- Tempo medio di accesso alla memoria (***AMAT***)

$$AMAT = c + (1-h) \cdot m$$

c: hit time h: hit rate
1-h: miss rate m: miss penalty

Le decisioni per la gerarchia di memorie

Quattro decisioni da prendere:

1. Dove si può posizionare un blocco nel livello gerarchico più alto (***posizionamento del blocco***)
2. Come si trova un blocco nel livello gerarchico più alto (***identificazione del blocco***)
   - Le prime due decisioni sono collegate e rappresentano le tecniche di indirizzamento di un blocco
3. Quale blocco nel livello gerarchico più alto si deve sostituire in caso di miss (***algoritmo di sostituzione***)
4. Come si gestiscono le scritture (***strategia di aggiornamento*** o write strategy)

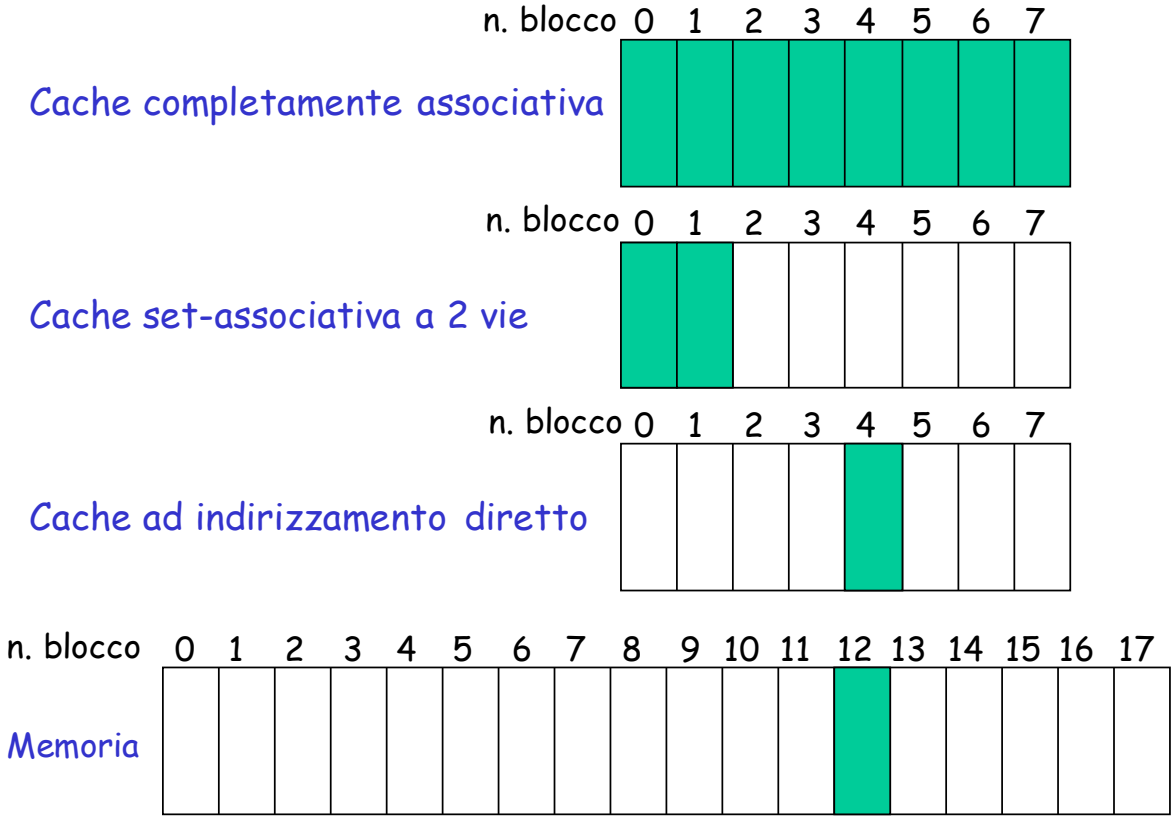
- Consideriamo "livello gerarchico più alto" = cache (per ora un solo livello di cache)

Posizionamento del blocco

- Tre categorie di organizzazione della cache in base alla restrizioni sul posizionamento del blocco in cache
- Dove può essere posizionato un blocco?

1. In una sola posizione della cache:
   - cache ad ***indirizzamento diretto*** (a mappatura diretta o direct mapped cache)
2. In un sottoinsieme di posizioni della cache:
   - cache ***set-associativa a N vie*** (set-associative cache)
3. In una qualunque posizione della cache:
   - cache ***completamente associativa*** (fully-associative cache)

Posizionamento del blocco

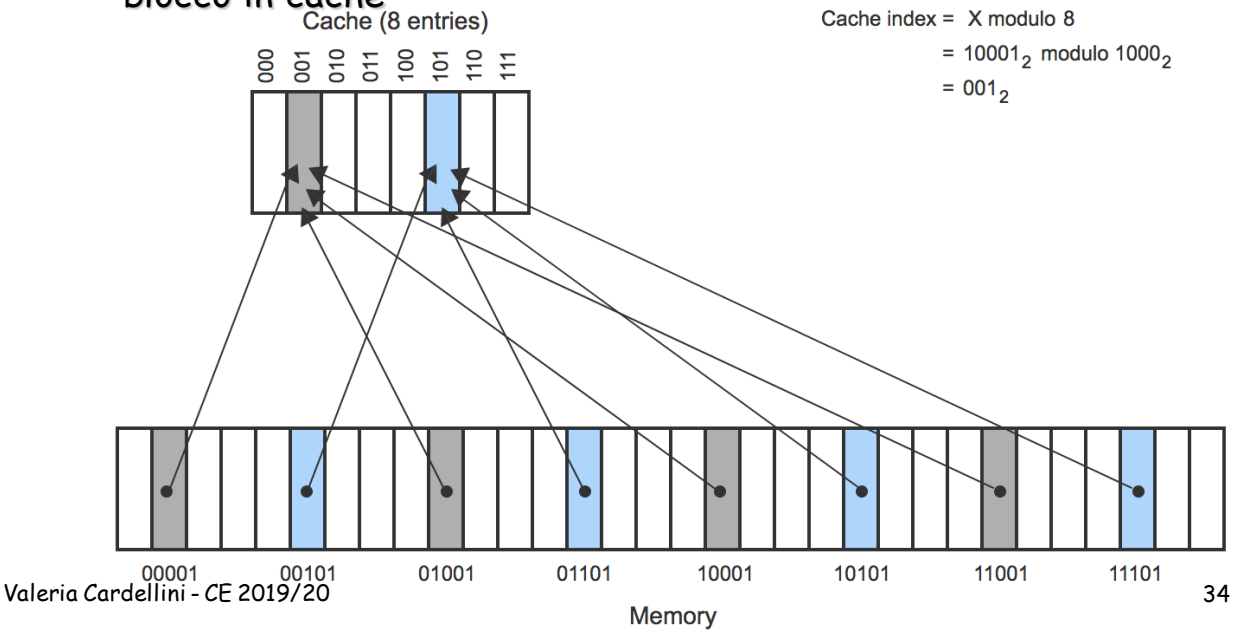


Cache ad indirizzamento diretto

- Ogni blocco nello spazio degli indirizzi della memoria può essere posizionato in uno e un solo blocco in cache
  - N_B: numero di blocchi in cache
  - B_{AC}: indirizzo del blocco in cache
  - B_{AM}: indirizzo del blocco in memoria
  - $B_{AC} = B_{AM}$ modulo N_B
- L'indirizzo del blocco in cache (detto *indice* della cache) si ottiene usando i $\log_2(N_B)$ bit meno significativi dell'indirizzo del blocco in memoria
  - Attenzione: la definizione si modifica opportunamente se il blocco contiene più parole (vedi slide 40)
- Tutti i blocchi della memoria che hanno i $\log_2(N_B)$ bit meno significativi dell'indirizzo uguali vengono mappati *sullo stesso* blocco di cache

Cache ad indirizzamento diretto: esempio

- Cache ad indirizzamento diretto con 8 blocchi
  - I blocchi di memoria con indirizzi 00001, 01001, 10001, 11001 hanno gli ultimi $\log_2 8=3$ bit uguali: sono mappati nello stesso blocco in cache



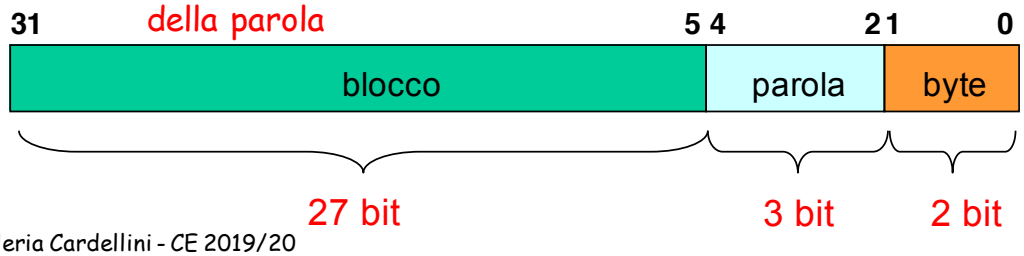
Ricerca di un blocco in cache

- Problema da risolvere: Come cercare se il byte identificato da un indirizzo di memoria è in un blocco di memoria presente nella memoria cache?

1. Come è organizzato un indirizzo di memoria per accedere alla memoria RAM?
   - Problema: questa organizzazione non contiene informazioni sufficienti per accedere alla memoria cache
2. Soluzione: organizzare la memoria cache come una memoria associativa
   - Strutturando anche in modo opportuno l'indirizzo di memoria

Esempio: indirizzo di memoria

- Indirizzi di memoria a 32 bit
- Parole di 4 byte
- Blocco di memoria di 32 byte (2^5 byte = 8 parole)
- Organizzazione dell'indirizzo di memoria
  - I 27 ($32-\log_2(32)=27$) bit più significativi dell'indirizzo di memoria identificano il blocco di memoria
  - I successivi 5 bit identificano le parole all'interno del blocco di memoria ed i singoli byte all'interno della parola
    - ✓ I primi 3 bit per il numero della parola all'interno del blocco
    - ✓ Gli ultimi 2 bit rappresentano il numero del byte all'interno della parola

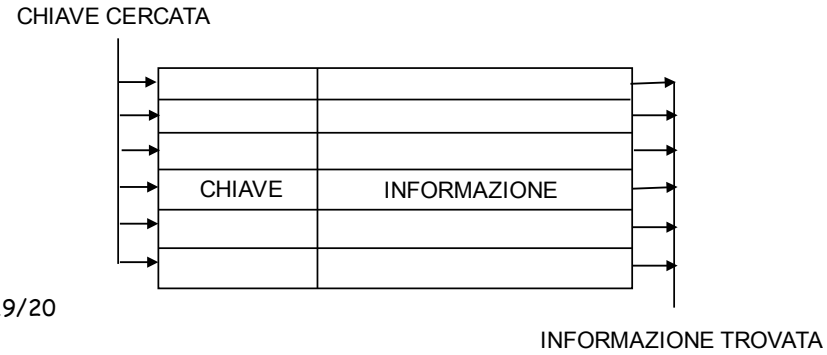


Ricerca di un blocco in cache

- Una cache contiene un sottoinsieme di blocchi di memoria di indirizzo non contiguo
- Come risalire dall'indirizzo di memoria alla sua posizione in cache?
  - Non è possibile utilizzare il normale meccanismo di indirizzamento della memoria
    - ✓ Si fornisce un indirizzo
    - ✓ Viene letto il dato che si trova all'indirizzo specificato
- Soluzione: organizzare la memoria cache come una **memoria associativa**

Memoria associativa

- Ciascun elemento di una memoria associativa è composto da due parti: la *chiave* e l'*informazione*
- L'accesso ad un elemento viene effettuato non in base all'indirizzo ma a parte del suo contenuto (la chiave)
- L'accesso *associativo* avviene in un unico ciclo
- Nel caso di una memoria cache:
  - La chiave è il numero del blocco di dati
  - L'informazione è il blocco di dati



Contenuto di una linea di cache

- In una cache ad indirizzamento diretto ogni linea di cache (o slot) include:
  - **Bit di validità**: indica se il blocco di dati contenuto nella linea di cache è valido
    - ✓ All'avvio, tutte le linee di cache sono non valide (*compulsory miss*)
  - **Tag** (o etichetta): consente di individuare in modo univoco il blocco di dati in memoria che è stato caricato nella linea di cache
    - ✓ E' la chiave della memoria associativa
  - **Blocco di dati**, formato da una o più parole
    - ✓ E' l'informazione della memoria associativa

Struttura dell'indirizzo e della linea di cache

- Spazio di memoria di 2^n byte, diviso in blocchi da 2^r byte
- Cache ad indirizzamento diretto di capacità pari a 2^s linee
- Si associa ad ogni blocco la linea di cache indicata dagli ***s*** bit meno significativi (non considerando l'offset) del suo indirizzo
  - Se il blocco è in cache, può essere soltanto in quella linea e lì bisogna cercarlo
- Il tag è dato dagli ***n-r-s*** bit più significativi dell'indirizzo
- Il tag è contenuto nella linea di cache e permette di distinguere tra tutti i blocchi di memoria che *condividono* la stessa linea di cache

Indirizzo
dimensione: n bit

Linea di cache
dimensione: 2^r byte + (*n-r-s*+1) bit

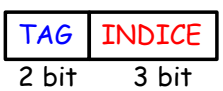
Esempio: cache hit

- Nell'esempio:
  - Indirizzo di memoria a 5 bit (n=5)
  - Blocco di dati contenente una sola parola (r=0)
  - Cache con 8 linee ($s=\log_2 8=3$)
  - Tag a 2 bit (n-r-s=2)
- Parola di indirizzo 10111 presente in cache (il tag corrisponde) e linea di cache valida: cache hit

Memory reference:

10111_{two} hit

Struttura indirizzo di memoria



Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111	Y	10_{two}	Memory(10111_{two})

Valid data — Requested word in cache

10 = 10

Esempio: cache miss

- Parola di indirizzo 10000 non è presente in cache (10≠01 il tag della linea di cache non corrisponde): cache miss

Memory reference:

10111_{two} hit

10000_{two} miss

Index	V	Tag	Data
000	Y	01_{two}	Memory (01000_{two})
001	Y	11_{two}	Memory (11001_{two})
010	N		
011	N		
100	Y	00_{two}	Memory (00100_{two})
101	N		
110	N		
111	Y	10_{two}	Memory(10111_{two})

Requested word not in cache

$10 \neq 01$

Esempio: cache miss

- Parola di indirizzo 10000 non è presente in cache (il tag della linea di cache non corrisponde): cache miss
- La parola viene caricata dalla memoria ed inserita nella corrispondente linea di cache

Memory reference:

10111_{two} hit

10000_{two} miss (fetched from memory)

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	Y	11_{two}	Memory (11001_{two})
010	N		
011	N		
100	Y	00_{two}	Memory (00100_{two})
101	N		
110	N		
111	Y	10_{two}	Memory(10111_{two})

Esempio: cache miss causa linea non valida

- Parola di indirizzo 11101 e linea di cache non valida: cache miss
- La parola verrà caricata dalla memoria ed inserita nella corrispondente linea di cache

Memory reference:

10111_{two} hit

10000_{two} miss (fetched from memory)

11101_{two} miss

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	Y	11_{two}	Memory (11001_{two})
010	N		
011	N		
100	Y	00_{two}	Memory (00100_{two})
101	N		
110	N		
111	Y	10_{two}	Memory(10111_{two})

Invalid data

Esempio: cache inizialmente vuota

- La cache è inizialmente vuota

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

a. The initial state of the cache after power-on

- Parola di indirizzo 10110: cache miss (linea non valida) e caricata dalla memoria

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Esempio: cache inizialmente vuota

- Parola di indirizzo 11010: cache miss e caricata dalla memoria

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

- Parola di indirizzo 10000: cache miss e caricata dalla memoria

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

Esempio: cache inizialmente vuota

- Parola di indirizzo 00011: cache miss e caricata dalla memoria

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

- Parola di indirizzo 10010: cache miss (tag non corrispondente) e caricata dalla memoria

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

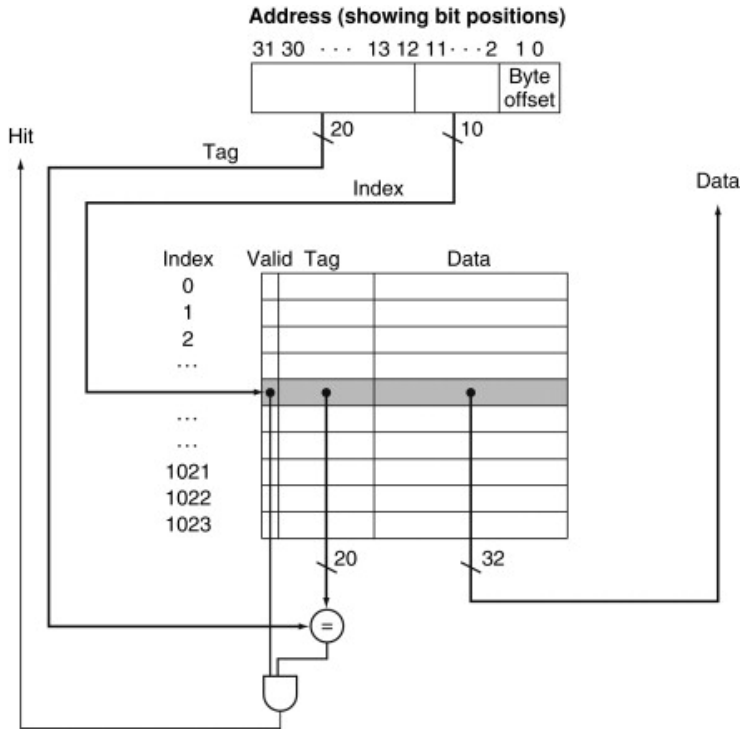
f. After handling a miss of address (10010_{two})

Un esempio più realistico

- Indirizzi a 32 bit
- Blocco di dati da 32 bit (4 byte)
- Cache con 1K di linee
- Quindi: n=32, r=$\log_2(4)$=2, s=$\log_2(1K)$=$\log_2(1024)$=10
- La struttura dell'indirizzo è:
  - I 20 bit più significativi dell'indirizzo rappresentano il tag
  - I successivi 10 bit rappresentano il numero del blocco in cache (l'indice della cache)
  - Gli ultimi 2 bit rappresentano il numero del byte all'interno del blocco (l'offset)

Accesso in cache

- Consideriamo l'esempio precedente
- Si confronta il Tag dell'indirizzo con il Tag della linea di cache individuata tramite l'indice
- Si controlla il bit di validità (Valid)
- Viene segnalato l'hit (o miss) al processore
  - Hit=1: cache hit
  - Hit=0: cache miss
- Viene trasferito il blocco di dati

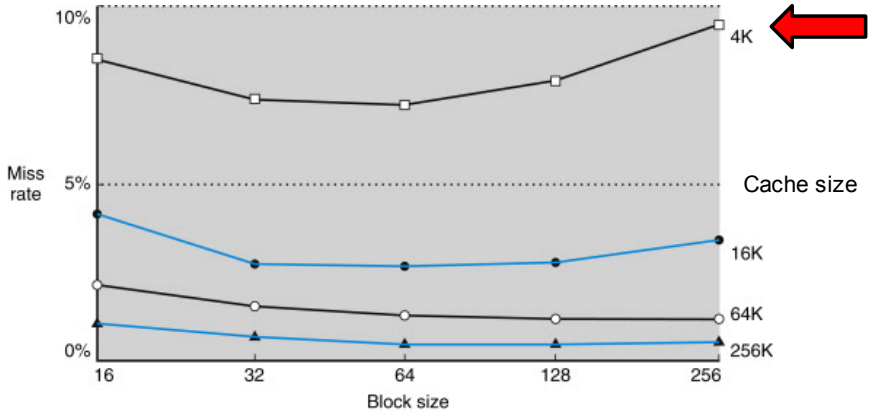


Dimensione cache ad indirizzamento diretto

- Quanti bit in totale sono necessari per una cache ad indirizzamento diretto?
  - Ciascuna linea di cache ha una dimensione pari a 2^r byte + $(n-r-s+1)$ bit
  - Nella cache ci sono 2^s linee
  - Quindi occorrono $2^s(2^{r+3}+n-r-s+1)$ bit
  - Overhead in bit = $2^s(n-r-s+1) / (2^s\ 2^{r+3}) = (n-r-s+1)/2^{r+3}$
- Esempio
  - Indirizzi a 32 bit
  - Cache con 16 KB di dati
  - Blocchi da 4 parole e parole da 4 B
  - Quindi:
    $n = 32$
    $s = \log_2(16KB/16B) = \log_2(2^{10}) = 10$
    $r = \log_2(4*4) = \log_2(16) = 4$
  - Occorrono $2^{10}(2^7+32-10-4+1)$ bit = 147 Kb = 18,375 KB
  - Overhead = 2,375KB/16KB ~ 15%

Scelta della dimensione del blocco

- In generale, una dimensione ampia del blocco permette di sfruttare la località spaziale, ma…
  - Blocchi più grandi comportano un miss penalty maggiore
    - ✓ E' necessario più tempo per trasferire il blocco
- Se la dimensione del blocco è troppo grande rispetto alla dimensione della cache, aumenta il miss rate



Gestione di cache hit e cache miss

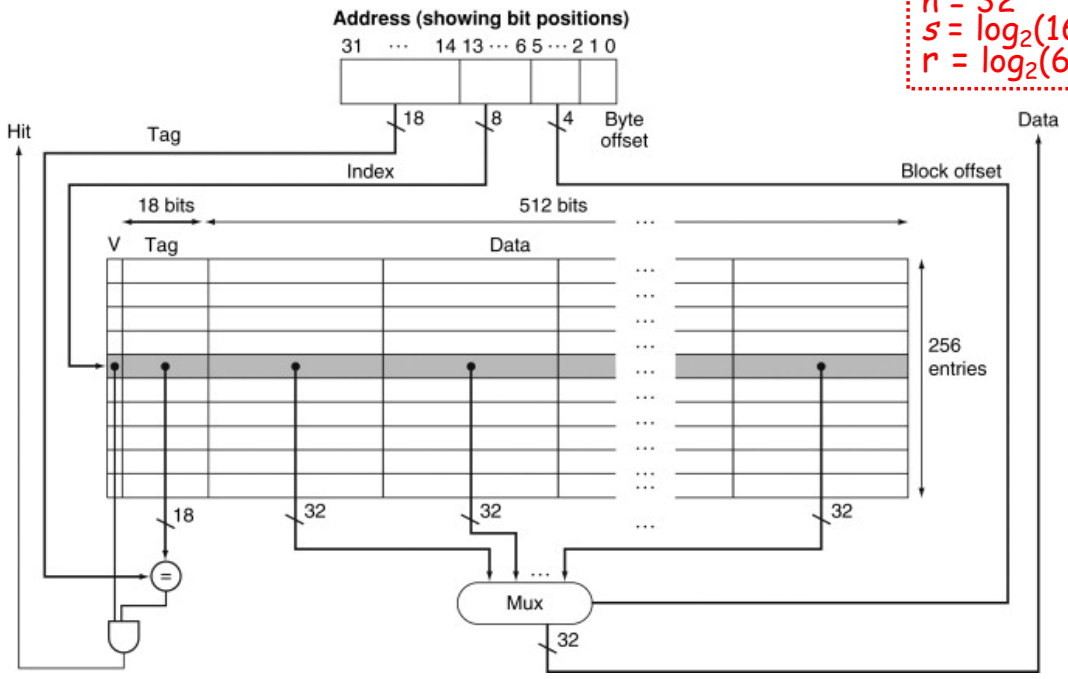
- In caso di hit: continua
  - Accesso al dato dalla memoria dati = cache dati
  - Accesso all'istruzione dalla memoria istruzioni = cache istruzioni

- In caso di miss:
  - Stallo del processore (come nel pipelining) in attesa di ricevere l'elemento dalla memoria
  - Invio dell'indirizzo al controller della memoria
  - Reperimento dell'elemento dalla cache
  - Caricamento dell'elemento in cache
  - Ripresa dell'esecuzione

Esempio: il processore Intrisity FastMATH

- Processore embedded basato su architettura MIPS
  - Cache istruzioni e cache dati ad indirizzamento diretto e separate, da 16 KB ciascuna e con blocchi di 16 parole (parole da 32 bit)

n = 32
$s = \log_2(16KB/64B) = 8$
$r = \log_2(64) = 6$



Sostituzione nelle cache ad indirizzamento diretto

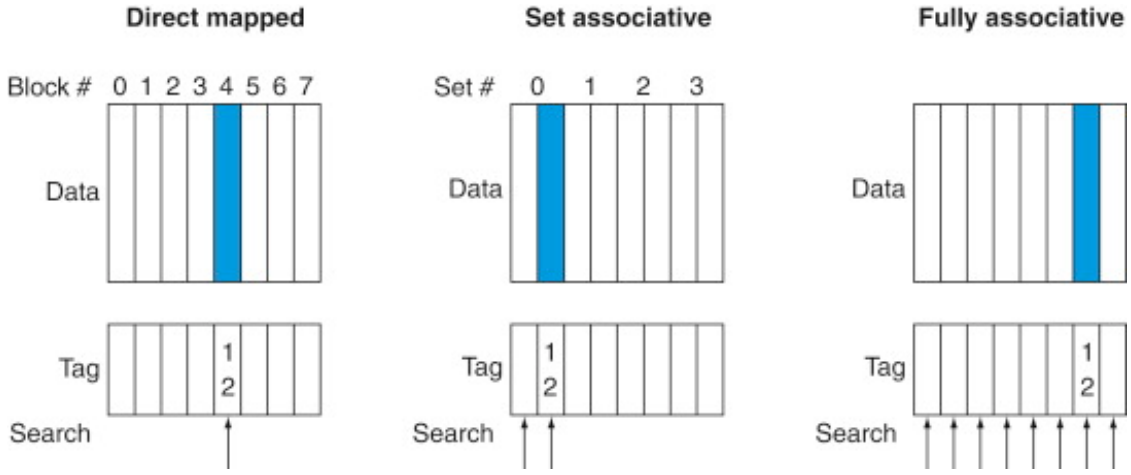
- Semplice: se il blocco di memoria viene mappato in una linea di cache già occupata, si elimina il contenuto precedente della linea e si rimpiazza con il nuovo blocco
  - I miss sono dovuti a conflitti sull'indice di cache (*conflict miss*)
- La sostituzione non tiene conto della località temporale
  - Può accadere che il blocco sostituito venga nuovamente usato a breve
  - Si può verificare il fenomeno del *thrashing*, che causa un degrado delle prestazioni

Cache ad indirizzamento diretto: riassumendo

- Vantaggi della cache ad indirizzamento diretto
  - Di facile implementazione
  - Occupa poca area sul chip
  - La ricerca di un blocco è veloce
- Svantaggi
  - Politica di sostituzione non molto efficiente

Come ridurre i cache miss?

- E' possibile ridurre il numero di cache miss con un posizionamento più flessibile dei blocchi
- Esaminiano le **cache set-associative**



Cache completamente associativa

- Nessuna restrizione sul posizionamento
- Ogni blocco di memoria può essere mappato in una qualsiasi linea di cache
  - Non ci sono conflict miss, ma i miss sono generati soltanto dalla capacità insufficiente della cache (*capacity miss*)
- Il contenuto di un blocco in cache è identificato mediante l'*indirizzo completo* di memoria
  - Il tag è costituito da tutto l'indirizzo della parola di memoria
- La ricerca viene effettuata mediante confronto *in parallelo* dell'indirizzo cercato con *tutti i tag* in cache
- Problemi
  - Maggiore complessità dell'hardware
  - Praticamente realizzabile solo con un piccolo numero di blocchi

Cache set-associativa a N vie

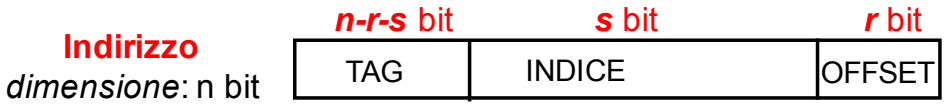
- Compromesso tra soluzione ad indirizzamento diretto e completamente associativa
- La cache è organizzata come insieme di set, ognuno dei quali contiene N blocchi (N: *grado di associatività*)
- Anche la memoria è vista come organizzata in set
  - Ogni set della memoria viene correlato ad uno e un solo set della cache con una filosofia ad indirizzamento diretto
- Ogni indirizzo di memoria corrisponde ad un unico set della cache (individuato tramite l'indice) e può essere ospitato in un blocco qualunque appartenente a quel set
  - Stabilito il set, per determinare se un certo indirizzo è presente in un blocco del set è necessario confrontare in parallelo tutti i tag dei blocchi nel set
- Si attenua il problema della collisione di più blocchi sulla stessa linea di cache

Confronto tra organizzazioni

- Una cache da 8 blocchi organizzata come
  - Ad indirizzamento diretto
  - Set-associativa a 2 vie
  - Set-associativa a 4 vie
  - Completamente associativa

Cache set-associativa a N vie

- L'indirizzo di memoria ha la stessa struttura dell'indirizzo per la cache ad indirizzamento diretto
  - Ma stavolta l'indice identifica il *set*



- A parità di dimensioni della cache, aumentando l'associatività di un fattore 2
  - Raddoppia il numero di blocchi in un set e si dimezza il numero di set
  - L'indice si riduce di un bit, il tag aumenta di un bit
  - Il numero dei comparatori raddoppia (confronti effettuati in parallelo)
- Cache set-associativa a N vie: N comparatori

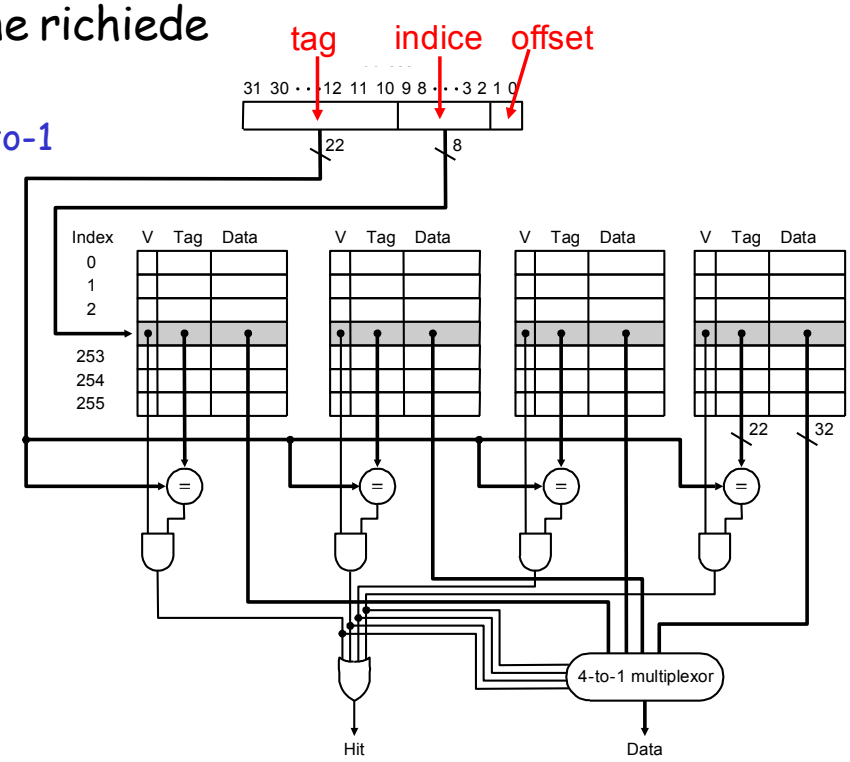
Esempio: cache set-associativa a N vie

- Indirizzi di memoria a 32 bit
- Cache set-associativa a 4 vie da 4 KB
- Blocco di dimensione pari a 1 parola (4 byte)
- Quindi:
  - $n = 32$
  - Numero di blocchi = dim. cache/dim. blocco = 4KB/4B = 1K
  - Numero di set = dim. cache/(dim.blocco x N) = 4KB/(4B x 4) = = 1K/4 = $2^8 \rightarrow s = \log_2(2^8) = 8$
  - $r = \log_2(4) = 2$
- Quindi la struttura dell'indirizzo è:
  - I 22 bit (n-r-s) più significativi dell'indirizzo rappresentano il tag
  - I successivi 8 bit (s) rappresentano il numero di set
  - Gli ultimi 2 bit (r) rappresentano il numero del byte all'interno del blocco

Cache set-associativa a 4 vie

- L'implementazione richiede
  - 4 comparatori
  - 1 multiplexer 4-to-1

- ✓ Tramite l'indice si seleziona uno dei 256 set
- ✓ I 4 tag del set sono confrontati in parallelo con il tag dell'indirizzo
- ✓ Il blocco di dati viene selezionato sulla base del risultato dei confronti

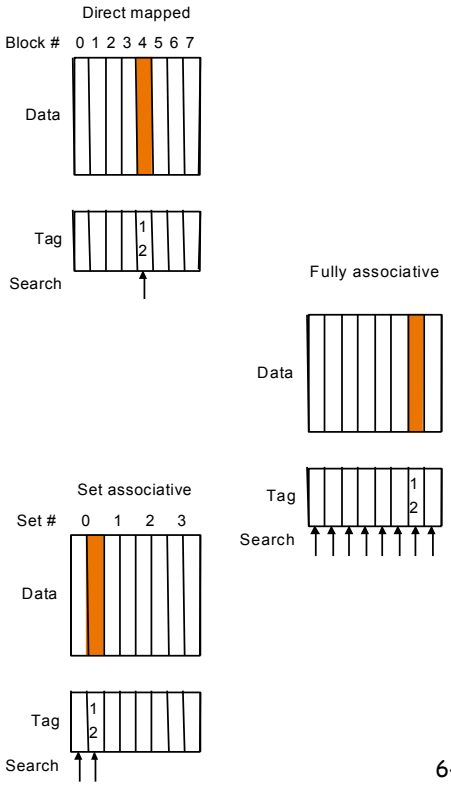


Dimensione del tag e associatività

- Aumentando il grado di associatività
  - Aumenta il numero dei comparatori e dei bit per il tag
- Esempio
  - Cache con 4K blocchi, blocco di 4 parole, indirizzo a 32 bit
  - $r = \log_2(4*4)=4 \rightarrow n-r = (32-4) = 28$ bit per tag e indice
  - Cache ad indirizzamento diretto
    - ✓ $s = \log_2(4K) = 12$
    - ✓ Bit di tag totali = (28-12)*4K = 64K
  - Cache set-associativa a 2 vie
    - ✓ $s = \log_2(4K/2) = 11$
    - ✓ Bit di tag totali = (28-11)*2*2K = 68K
  - Cache set-associativa a 4 vie
    - ✓ $s = \log_2(4K/4) = 10$
    - ✓ Bit di tag totali = (28-10)*4*1K = 72K
  - Cache completamente associativa
    - ✓ s = 0
    - ✓ Bit di tag totali = 28*4K*1 = 112K

Identificazione del blocco e associatività

- Cache a mappatura diretta
  - Identificazione della posizione del blocco in cache
  - Verifica del tag
  - Verifica del bit di validità
- Cache completamente associativa
  - Verifica dei tag di tutti blocchi in cache
  - Verifica del bit di validità
- Cache set-associativa a N vie
  - Identificazione della posizione dell'insieme in cache
  - Verifica di N tag dei blocchi nel set
  - Verifica del bit di validità



Incremento dell'associatività

- Principale vantaggio
  - Diminuzione del miss rate
- Principali svantaggi
  - Maggior costo implementativo
  - Incremento dell'hit time
- La scelta tra cache ad indirizzamento diretto, set-associativa e completamente associativa dipende dal costo dell'associatività rispetto alla riduzione del miss rate

Sostituzione nelle cache completamente associative e set-associative

- Quale blocco sostituire in caso di capacity miss?
- Cache completamente associativa: ogni blocco è un potenziale candidato per la sostituzione
- Cache set-associativa a N vie: bisogna scegliere tra gli N blocchi del set
  - Come? Politica di sostituzione

Sostituzione nelle cache completamente associative e set-associative

- Come scegliere il blocco da sostituire?
- Politica di sostituzione **Random**
  - Scelta casuale
- Politica di sostituzione Least Recently Used (**LRU**)
  - Sfruttando la località temporale, il blocco sostituito è quello che non si utilizza da più tempo
  - Ad ogni blocco si associa un contatore all'indietro, che viene portato al valore massimo in caso di accesso e decrementato di 1 ogni volta che si accede ad un altro blocco
- Politica di sostituzione First In First Out (**FIFO**)
  - Si approssima la strategia LRU selezionando il blocco più vecchio anziché quello non usato da più tempo

Problema della strategia di scrittura

- Le scritture sono meno frequenti delle letture
- Le prestazioni sono migliori per le letture
  - La lettura può iniziare non appena è disponibile l'indirizzo del blocco, prima che sia completata la verifica del tag
  - La scrittura deve aspettare la verifica del tag
- In conseguenza di un'operazione di scrittura effettuata su un blocco presente in cache, i contenuti di quest'ultima saranno diversi da quelli della memoria di livello inferiore
  - Occorre definire una strategia per la gestione delle scritture
  - Strategia write-through
  - Strategia write-back

Strategia write-through

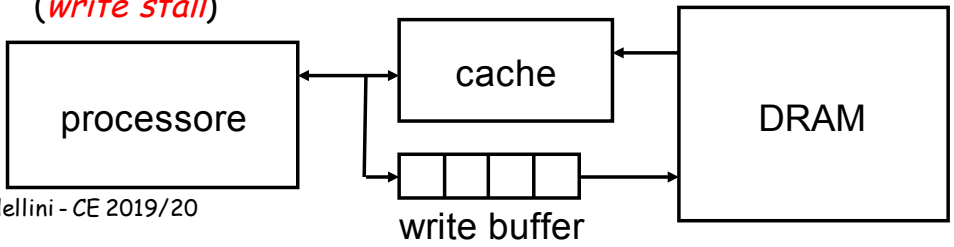
- Scrittura *immediata*: il dato viene scritto simultaneamente sia nel blocco della cache sia nel blocco contenuto nella memoria di livello inferiore
- Vantaggi
  - E' la soluzione più semplice da implementare
  - Si mantiene la **coerenza** delle informazioni nella gerarchia di memorie
- Svantaggi
  - Le operazioni di scrittura vengono effettuate alla velocità della memoria di livello inferiore → diminuiscono le prestazioni
  - Aumenta il traffico sul bus di sistema
- Alternative
  - Strategia **write-back**
  - Utilizzo di un **write buffer**

Strategia write-back

- Scrittura *differita*: i dati sono scritti solo nel blocco presente in cache; il blocco modificato viene trascritto nella memoria di livello inferiore solo quando viene sostituito
  - Dopo la scrittura, cache e memoria di livello inferiore sono *inconsistenti*
  - Il blocco in cache può essere in due stati (*dirty bit*):
    - ✓ clean: non modificato
    - ✓ dirty: modificato
- Vantaggi
  - Le scritture avvengono alla velocità della cache
  - Scritture successive sullo stesso blocco alterano solo la cache e richiedono una sola scrittura nel livello inferiore di memoria
- Svantaggi
  - Ogni sostituzione del blocco (ad es. dovuto a read miss) può provocare un trasferimento in memoria

Strategia write-through con write buffer

- Buffer per la scrittura (*write buffer*) interposto tra cache e memoria di livello inferiore
  - Il processore scrive i dati in cache **e** nel write buffer
  - Il controller della memoria scrive il contenuto del write buffer in memoria: la scrittura avviene in modo asincrono e indipendente
- Il write buffer è gestito con disciplina FIFO
  - Numero tipico di elementi del buffer: 4
  - Efficiente se frequenza di scrittura << 1/write cycle di DRAM
    - ✓ Altrimenti, il buffer può andare in saturazione ed il processore deve aspettare che le scritture giungano a completamento (*write stall*)



Prestazioni delle cache

- Il tempo di CPU può essere suddiviso in due componenti
  CPU time = (CPU execution clock cycles + memory-stall clock cycles) × clock cycle time
- Gli stalli in attesa della memoria sono dovuti ad operazioni di lettura o scrittura
  memory-stall clock cycles = read-stall cycles + write-stall cycles
- Gli stalli per operazioni di lettura sono dati da:
  read-stall cycles = reads/program × read miss rate × read miss penalty
- Usando la strategia write-through con write buffer, gli stalli per operazioni di scrittura sono dati da:
  write-stall cycles = (writes/program × write miss rate × write miss penalty) + write buffer stalls

Prestazioni delle cache (2)

- Nella maggior parte delle organizzazioni di cache che adottano la strategia write-through, il miss penalty per scritture è uguale a quello per letture
  memory-stall clock cycles = memory access/program × miss rate × miss penalty
  O anche:
  memory-stall clock cycles = instructions/program × misses/instruction × miss penalty

Cache L1

- Quali alternative per cache di primo livello?
  - Cache unificata
  - Cache dati (D-cache) e istruzioni (I-cache) separate
- Cache separate possono essere ottimizzate individualmente
  - La I-cache ha un miss rate più basso della D-cache
  - La I-cache è di tipo read-mostly
    - ✓ Località spaziale molto buona (tranne che nel caso di chiamate a procedura molto frequenti)
  - La località della D-cache è fortemente dipendente dall'applicazione

Cause dei cache miss

- **Compulsory miss**
  - Detti anche miss al primo riferimento: al primo accesso il blocco non è presente in cache
  - Non dipendono dalle dimensioni e dall'organizzazione della cache
- **Capacity miss**
  - Durante l'esecuzione del programma, alcuni blocchi devono essere sostituiti
  - Diminuiscono all'aumentare delle dimensioni della cache
- **Conflict miss**
  - Può accadere di ricaricare più tardi nello stesso set i blocchi che si sono sostituiti
  - Diminuiscono all'aumentare delle dimensioni della cache e del grado di associatività
  - "Regola del 2:1": il miss rate di una cache ad indirizzamento diretto di N byte è circa pari a quello di una cache set-associativa a 2 vie di N/2 byte

Migliorare le prestazioni delle cache

- Consideriamo il tempo medio di accesso in memoria (AMAT)
  - AMAT = hit time + miss rate × miss penalty
- Come possiamo migliorare AMAT?
  - Riducendo i fattori presenti nella formula
- Analizziamo alcune ottimizzazioni delle cache per
  - Ridurre il miss penalty
  - Ridurre il miss rate
  - Ridurre miss penalty e miss rate tramite parallelismo

Ridurre il miss penalty

- Esaminiamo due soluzioni per ridurre il miss penalty
  - Cache multi-livello
  - Victim cache

Cache multi-livello

- Gerarchia composta da due o più livelli di cache
  - Cache più piccole e veloci vicino al processore
  - Cache più grandi e lente scendendo nella gerarchia
  - Anche 3 livelli di cache
- Nel caso di 2 livelli (L1 e L2):
  - Cache di primo livello abbastanza piccola da essere molto veloce
  - Cache di secondo livello abbastanza grande da soddisfare molti degli accessi che altrimenti andrebbero in DRAM

$AMAT = hit\ time_{L1} + miss\ rate_{L1} \times miss\ penalty_{L1}$

$miss\ penalty_{L1} = hit\ time_{L2} + miss\ rate_{L2} \times miss\ penalty_{L2}$

$AMAT = hit\ time_{L1} + miss\ rate_{L1} \times (hit\ time_{L2} + miss\ rate_{L2} \times miss\ penalty_{L2})$

  - $miss\ rate_{L2}$ si misura sugli accessi alla cache L1 che non hanno avuto successo

Cache multi-livello: prestazioni

- Consideriamo L1 e L2

$AMAT = hit\ time_{L1} + miss\ rate_{L1} \times miss\ penalty_{L1}$

$miss\ penalty_{L1} = hit\ time_{L2} + miss\ rate_{L2} \times miss\ penalty_{L2}$

  - $miss\ rate_{L2}$ si misura sugli accessi alla cache L1 che non hanno avuto successo

- Sostituendo:

$AMAT = hit\ time_{L1} + miss\ rate_{L1} \times (hit\ time_{L2} + miss\ rate_{L2} \times miss\ penalty_{L2})$

Ridurre il miss penalty con cache multi-livello

- La cache L2 ha senso solo se è più grande della cache L1
- La velocità della cache L1 influenza il ciclo di clock del processore
- La velocità della cache L1 influenza solo il miss penalty della cache L1 (miss penalty$_{L1}$)
- I dati presenti nella cache L1 sono presenti anche nella cache L2? Due soluzioni:
  - *Multi-level inclusion*: i dati in L1 sono *sempre* presenti in L2
    - ✓ Si mantiene la consistenza tra cache e I/O
  - *Multi-level exclusion*: i dati in L1 non sono *mai* presenti in L2
    - ✓ Soluzione ragionevole per cache L2 di piccole dimensioni

Victim cache

- Idea: un blocco scartato potrebbe essere di nuovo utile a breve
- Si inserisce una piccola cache associativa (detta *victim cache*) fra la cache ed il suo percorso di riempimento
  - La victim cache contiene solo i blocchi scartati dalla cache in caso di miss
  - In caso di miss, i blocchi scartati e messi nella victim cache vengono verificati prima di passare alla memoria di livello inferiore
  - Se il blocco richiesto è nella victim cache, si scambia questo blocco con quello in cache
- La victim cache è utile per ridurre il miss penalty causato dai miss di conflitto

Ridurre il miss rate

- Per ridurre il miss rate si può:
  - Aumentare la dimensione del blocco
    - ✓ Si sfrutta il principio di località spaziale e si riducono i compulsory miss (ma aumenta miss penalty)
  - Aumentare la dimensione della cache
    - ✓ Si riducono i capacity miss ed i conflict miss (ma aumentano hit time e costo)
  - Aumentare il grado di associatività
    - ✓ Si riducono i conflict miss (ma aumenta hit time)

Soluzioni già esaminate

- Ottimizzazioni del compilatore

Ottimizzazioni del compilatore

- Le soluzioni già esaminate per ridurre il miss rate richiedono modifiche dell'hardware
- Ottimizzazioni del codice effettuate durante la compilazione permettono di ridurre il numero di miss *senza* modificare l'hardware
  - Ottimizzazioni orientate a risolvere separatamente instruction miss e data miss
  - Le tecniche di ottimizzazione attuate dal compilatore riorganizzano il codice in modo da migliorare la località temporale e/o spaziale
- Analizziamo due ottimizzazioni che riducono il *data miss rate*:
  - **Loop interchange**
  - **Blocking**

Loop interchange

- Obiettivo del **loop interchange**: migliorare la *località spaziale*
- Il compilatore scambia l'ordine di cicli annidati in modo che si acceda ai dati nell'ordine in cui questi sono memorizzati
- Esempio
  - In C le matrici (array bidimensionali) sono memorizzate per righe (*row major order*)

Loop interchange

```
/* Before */                    /* After */
for (j=0; j<100; j++)            for (i=0; i<5000; i++)
  for (i=0; i<5000; i++)           for (j=0; j<100; j++)
    A[i][j] = 2*A[i][j];             A[i][j] = 2*A[i][j];
```

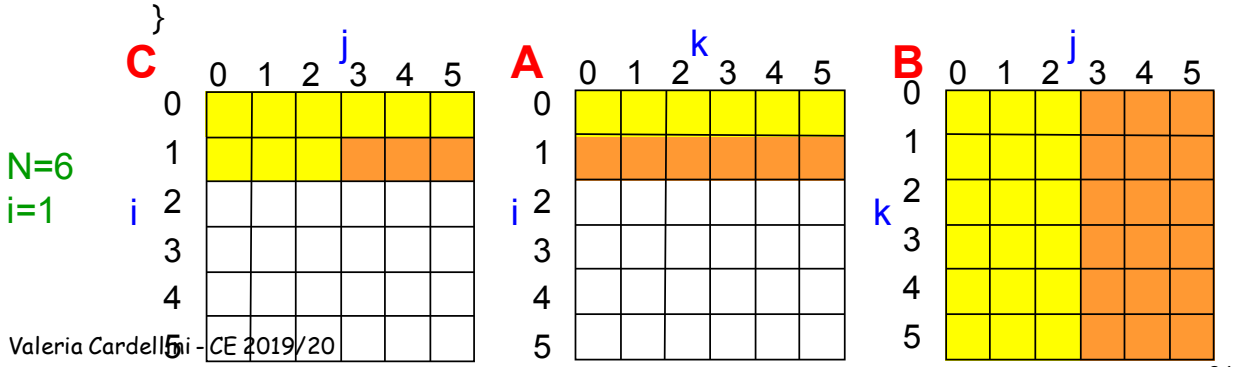

Blocking

- Obiettivo del **blocking**: migliorare la *località temporale*
- Esempio: in un algoritmo in cui ad alcune matrici si accede per righe e ad altre per colonne, memorizzare le matrici per righe piuttosto che per colonne non migliora le prestazioni
- Con il blocking l'algoritmo, invece di operare su righe o colonne intere, opera su sottomatrici (*blocchi*)
  - Lo scopo è massimizzare l'accesso ai dati in cache prima che vengano sostituiti

Blocking (2)

- Esempio: moltiplicazioni tra matrici quadrate y e z

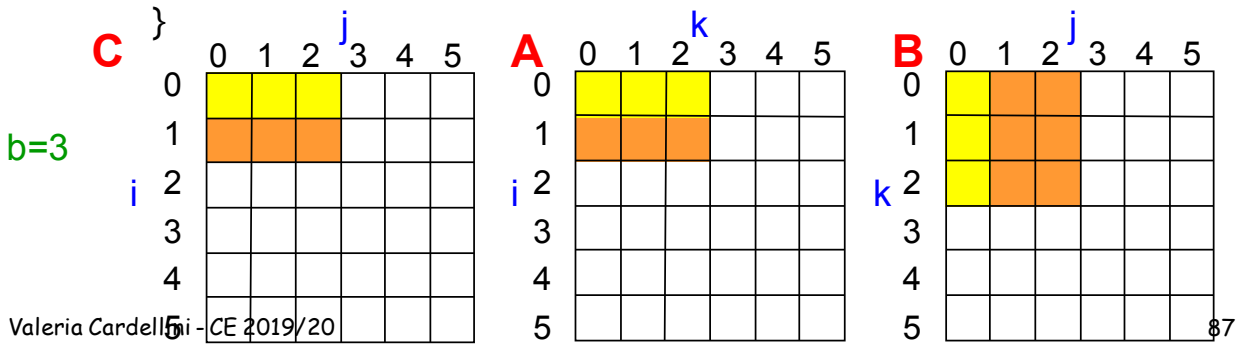
```
/* Before */
for (i=0; i<N; i++)
        for (j=0; j<N; j++) {
                r = 0;
                 for (k=0; k<N; k++)
                         r += A [i][k]*B[k][j];
                C[i][j] = r;
        }
```



Blocking (3)

```
/* After */
for (jj=0; jj<N; jj+=b)
  for (kk=0; kk<N; kk+=b)
    for (i=0; i<N; i++)
        for (j=jj; j<min(jj+b, N); j++) {
                r = 0;
                 for (k=kk; k<min(kk+b, N); k++)
                         r += A[i][k]*B[k][j];
                C[i][j] += r;
        }
```

- Il codice lavora su una sottomatrice di dimensione bxb
- b è chiamato *fattore di blocking*



Ridurre miss penalty e miss rate tramite parallelismo

- Idea: sovrapporre l'esecuzione del programma all'attività della gerarchia di memorie
- Diverse soluzioni, tra queste esaminiamo il prefetching
  - Hardware
  - Software controllato dal compilatore

Prefetching hardware di istruzioni e dati

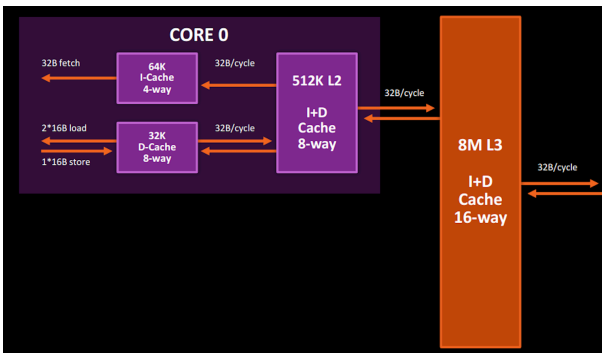
- Idea base: lettura *anticipata* (*prefetch*) di elementi (istruzioni o dati) prima che siano richiesti dal processore
- Prefetching di istruzioni
  - Spesso eseguito dall'hardware
  - In caso di miss, il processore legge il blocco richiesto e quello immediatamente consecutivo. Il blocco richiesto viene portato in cache, quello anticipato viene posto nell'instruction *stream buffer* (ISB)
  - Se un blocco richiesto è presente nell'ISB, lo si legge dall'ISB, si cancella la richiesta alla cache, si invia la prossima richiesta di prefetch
- Prefetching di dati
  - Simile a quello delle istruzioni

Prefetching controllato dal compilatore

- Il compilatore inserisce istruzioni di prefetch per richiedere i dati prima che occorrano
  - Register prefetch: carica il valore in un registro
  - Cache prefetch: carica i dati in cache

Un esempio di gerarchia di memorie (solo cache)

- Gerarchia di memorie cache di AMD Zen
  https://en.wikichip.org/wiki/amd/microarchitectures/zen
  - Cache L1I (istruzioni) da 64 KB per core, set-associativa a 4-vie, 256 set, linea di cache da 64 B
  - Cache L1D (dati) da 32KB per core, set-associativa a 8-vie, 64 set, linea di cache da 64 B, strategia write-back, latenza 4/8 cicli per int/float
  - Cache L2 (unificata per dati e istruzioni) da 512 KB per core, set-associativa a 8-vie, 1024 set, linea di cache da 64 B, strategia write-back, latenza 12 cicli
  - Cache L3 da 8 MB, condivisa da tutti i core, set-associativa a 16-vie, 8192 set, linea di cache da 64 bit, latenza 40 cicli



Per concludere: da ricordare!

- "Latency numbers **every programmer should know**"
  http://bit.ly/2pZXIU9