



## Gerarchie di memoria

### *Politiche di rimpiazzo dei blocchi*

Quale blocco conviene sostituire in cache per effettuare uno swap?  
(*Penalità di miss*)

- **Casuale**, per occupazione omogenea dello spazio
- **First-In-First-Out (FIFO)**, per sostituire il blocco rimasto più a lungo in cache
- **Least Frequently Used (LFU)**, per sostituire il blocco con meno accessi
- **Least Recently Used (LRU)**, per preservare *località temporale*

P(miss)		rimpiazzo casuale			rimpiazzo LRU		
N-way		2	4	8	2	4	8
Ampiezza cache	16 KB	5,69	5,29	4,96	5,18	4,67	4,39
	64 KB	2,01	1,66	1,53	1,88	1,54	1,39
	256 KB	1,17	1,13	1,12	1,15	1,13	1,12

## Gerarchie di memoria

### *Il problema della scrittura*



La scrittura dei dati determina *incoerenza* tra il blocco in cache e quello nei livelli superiori

#### ■ 'Write through'

- ☐ Scrittura *contemporanea* in cache e nel livello di memoria inferiore
- ☐ Aumento di traffico per frequenti scritture nel medesimo blocco, ma i dati sono sempre coerenti tra i livelli
- ☐ Si ricorre a buffer di scrittura *asincroni* (differiti) verso la memoria.

*N.B.: La memoria contiene istruzioni e dati, e solo il 50% delle operazioni sui dati sono scritture (circa 12 % del totale)*

## Gerarchie di memoria

### *Il problema della scrittura*



#### ■ 'Write back'

- ☐ Scrittura in memoria inferiore *differita* al rimpiazzo del blocco di cache corrispondente
- ☐ Occorre ricordare se sono avvenute operazioni di scrittura nel blocco
- ☐ Consente ottimizzazione del traffico tra livelli
- ☐ Causa periodi di incoerenza (problemi con moduli di I/O e multiprocessori con cache locale)

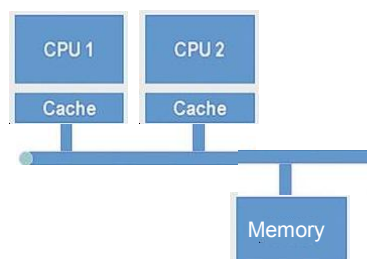
## Gerarchie di memoria

### *Il problema della scrittura*



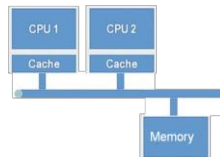
#### ■ Scenario particolarmente problematico

- ☐ Più dispositivi (es. processori) connessi allo stesso bus con cache locale
- ☐ Memoria centrale condivisa



# Gerarchie di memoria

## *Il problema della scrittura*

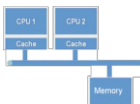


### ■ Modifica dati in una cache

- ☐ invalida la parola corrispondente in memoria centrale
- ☐ invalida la parola corrispondente nelle altre cache che la contengono
- ☐ write through non risolve il problema (risolve solo l'inconsistenza della memoria centrale)

# Gerarchie di memoria

## *Il problema della scrittura*



### ■ Possibili soluzione

- ☐ Monitoraggio del bus con write through
  - Controllori cache intercettano modifiche locazioni condivise
- ☐ Trasparenza hardware
  - Hardware aggiuntivo: modifica a M → modifica tutte cache
- ☐ Memoria noncacheable
  - Solo una porzione di M è condivisa e non cachable (accessi a M condivisa generano miss)

# Gerarchie di memoria

## *Il problema dei 'miss'*



- Miss di **primo accesso**, *inevitabile* e non riducibile
- Miss per **capacità insufficiente**, quando la cache *non può* contenere tutti i blocchi necessari all'esecuzione del programma
- Miss per **conflitto**, quando *più* blocchi possono essere mappati (con associazione diretta o a gruppi) su *uno* stesso gruppo

# Gerarchie di memoria

## *Il problema dei 'miss'*



- Tecniche "classiche" di soluzione
  - Maggior dimensione di blocco
    - Buona per fruire di *località spaziale*
    - Causa incremento di miss per conflitto (meno blocchi disponibili)
  - Maggiore associatività
    - Causa incremento del tempo di localizzazione in gruppo (hit)
    - Soggetta alla 'regola del 2:1'
      - Una cache ad N blocchi con associazione diretta ha una probabilità di miss pressoché uguale ad una cache di dimensione N/2 con associazione a 2 vie

# Gerarchie di memoria

## *Il problema dei 'miss'*



### ■ Altre tecniche

- Cache multilivello (cache *on-chip* L1 e/o L2)
- Separazione tra cache *dati* e cache *istruzioni*
- Ottimizzazione degli accessi mediante compilatori
  - Posizionamento accurato delle procedure ripetitive
  - Fusione di vettori in strutture (località spaziale)
  - Trasformazioni di iterazioni annidate (località spaziale)
  - ...

# Gerarchie di memoria

## *Es.: Fusione di vettori in strutture*

```

/* prima della ottimizzazione */
int val[SIZE];
int key[SIZE];

merged_array[0] /* dopo l'ottimizzazione */
struct merge {
    int val;
    int key;
};
struct merge merged_array[SIZE];
          
```

MEMORIA

MEMORIA

```

struct merge {
    int val;
    int key;
};
          
```

# Gerarchie di memoria

## Es.: *Iterazioni annidate*

```
/* prima della ottimizzazione */
for (j=0;j<100;j=j+1)
  for (i=0;i<5000;i=i+1)
    x[i][j] = 2*x[i][j];
```

```
/* dopo l'ottimizzazione */
for (i=0;i<5000;i=i+1)
  for (j=0;j<100;j=j+1)
    x[i][j] = 2*x[i][j];
```

