La Gerarchia delle Memorie

Architettura, Progetto e Simulazione delle memorie cache

Le Gerarchie di memorie

- Un sistema di calcolo ideale dovrebbe essere dotato di un organo di memoria caratterizzato da
  - uno spazio di indirizzi grande a piacere
  - tempi di accesso piccoli a piacere
  - costo basso e indipendente dalla tecnologia impiegata.
- I dispositivi di memoria reali utilizzati nei sistemi di calcolo non godendo di tali proprietà sono caratterizzati da
  - soluzioni legati a specifiche tecnologie ed associati costi
  - classi di memorie caratterizzate principalmente dalla velocità di accesso e dalla capacità.

Dati di targa delle memorie

- tempo di accesso **ta**
  - il tempo impiegato per eseguire un ciclo completo di lettura/scrittura di un dato di assegnata granularità
- capacità di memoria **s**
  - la dimensione in byte (o parole di assegnato parallelismo) della memoria
- costo per byte **c**
  - serve a definire il costo completo C=c*s della memoria
- banda di trasferimento **b**
  - l'ampiezza di banda b caratterizza il volume di dati che una memoria è in grado di trasferire verso il generico processore che la utilizza
- quanto di trasferimento in multipli di byte (o parole macchina di assegnata lunghezza) **p**
  - esprime la grana (parallelismo) impiegato nel trasferimento dei dati da e per il processore connesso alla memoria.

Memoria: Aumento di Prestazioni

- A causa del ruolo critico che la memoria asusme quando si cerca un incremento nel throughput del sistema, essa viene detta: *The Von Neumann Bottleneck*

- Un modo per incrementare le prestazioni del sistema è quello di aumentare opportunamente la banda della memoria (bps accessibili). Tale incremento di banda è ottenibile in tre modi :

  - Aumentando la frequenza del clock di memoria

  - Aumentando la larghezza della word e quindi accedendo a più bit per ciclo

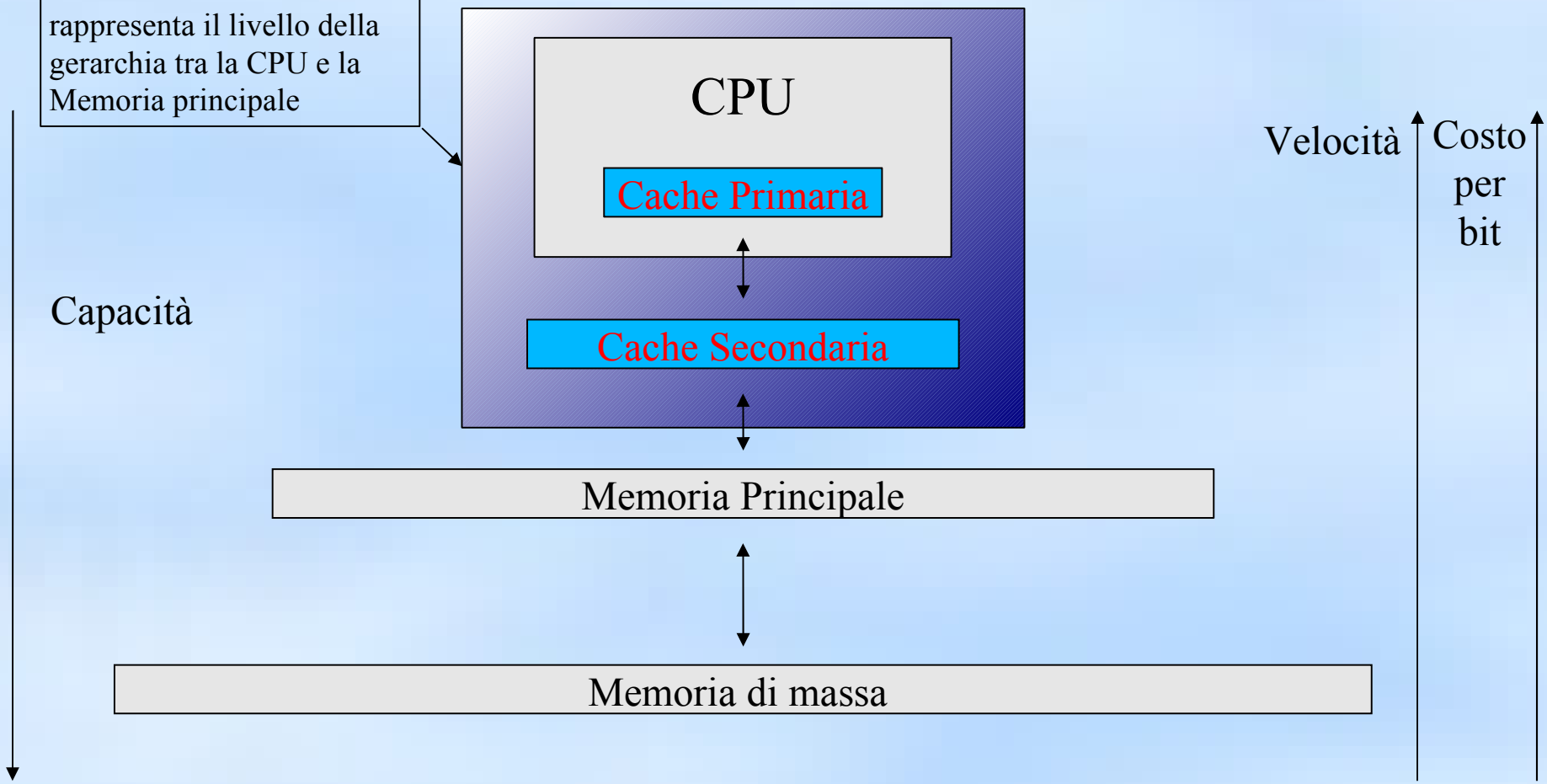
  - Replicando i moduli di memoria in accesso concorrente (simile alla precedente)

Soluzione :

- Aumentare la frequenza => Memorie più costose e di dimensioni ridotte

Gerarchia di Memorie

La memoria Cache rappresenta il livello della gerarchia tra la CPU e la Memoria principale



Principio di Località

Una delle principali proprietà dei programmi è quella della :

Locality of references (90/10 rule)

- I Programmi tendono a riusare i dati e le istruzioni che hanno usato di recente
- Una regola empirica largamente verificata asserisce che un programma in media trascorre il 90% del suo tempo di esecuzione in una porzione di codice che rappresenta circa il 10% dell'intero programma (la stessa regola vare per i riferimenti sui dati)
- Molte istruzioni, quindi, in aree ben localizzate di un programma vengono eseguite ripetutamente in un determinato periodo, e si accede al resto del programma relativamente di rado

Basandosi sulle ultime istruzioni eseguite da un programma è possibile predire con ragionevole accuratezza quali istruzioni e quali dati del programma verranno utilizzati nel prossimo futuro

Principio di Località

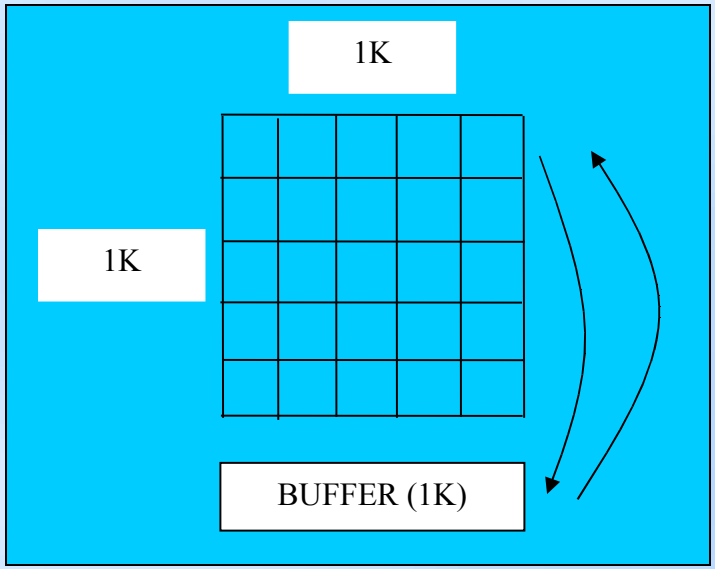
- *Località Temporale:*
  - Rappresenta la probabilità (alta) che un'istruzione eseguita di recente venga eseguita di nuovo entro breve tempo.
- *Località Spaziale:*
  - Rappresenta la probabilità (alta) che istruzioni vicine ad un'istruzione eseguita di recente siano anch'esse eseguite nel prossimo futuro .
  - La vicinanza è espressa in termini di indirizzi delle istruzioni

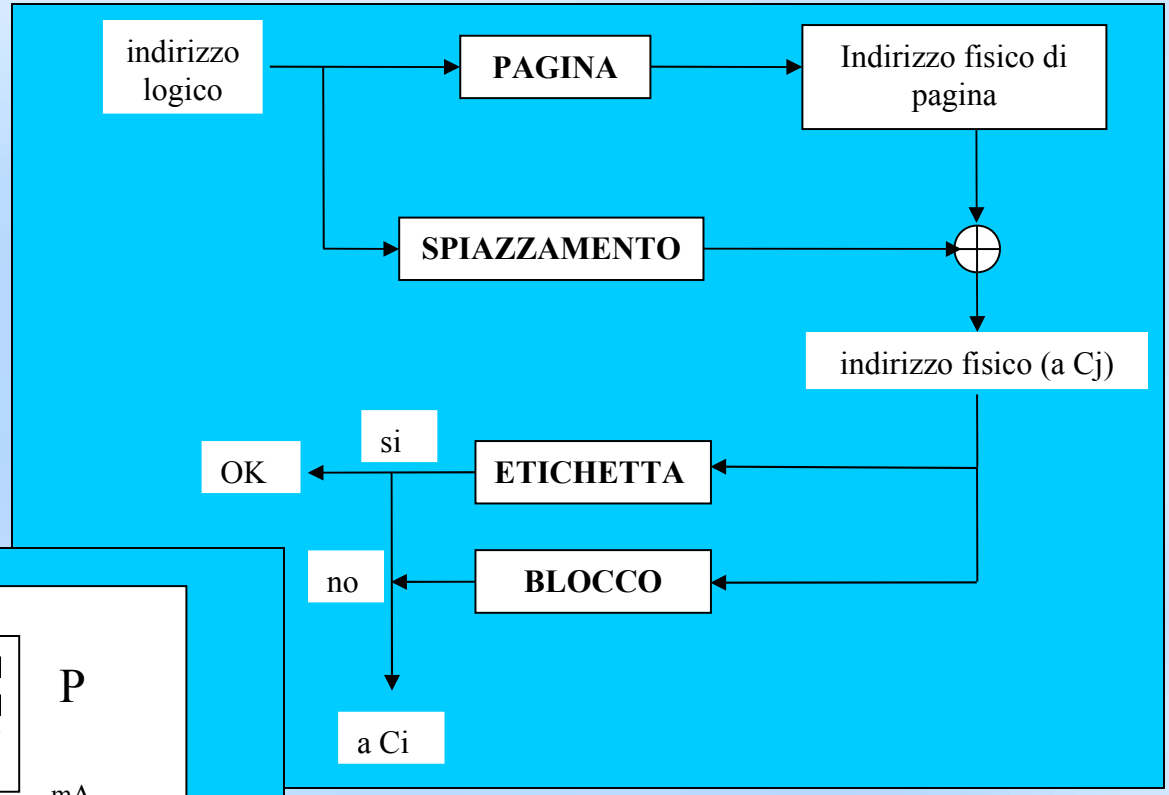
Principio di Località

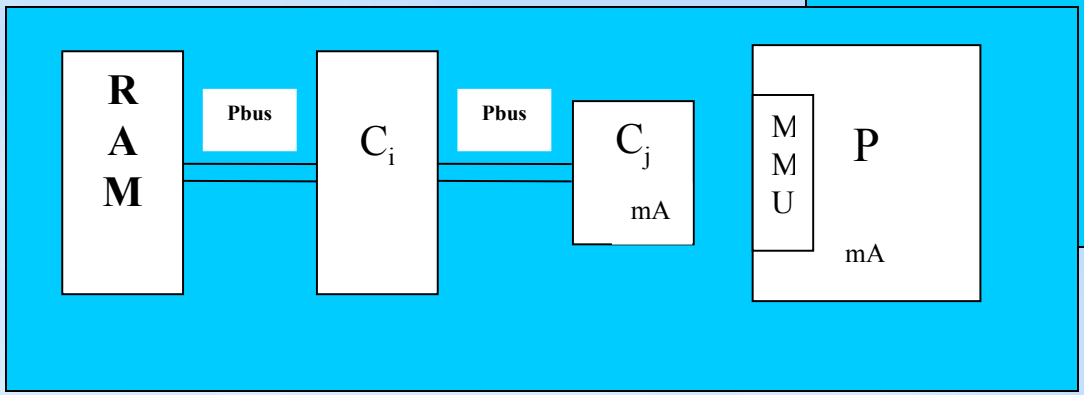
Località Temporale → Portare un elemento (istruzioni o dati) nella cache quando viene richiesto per la prima volta, in modo che rimanga a disposizione nel caso di una nuova richiesta.

Località Spaziale → Invece di portare dalla memoria principale alla cache un elemento alla volta, è conveniente portare un insieme (*block)* di elementi che risiedono in indirizzi adiacenti

Il Principio di Funzionamento







Struttura di una Cache

- Parametri Caratteristici
- Funzione di mapping
- Algoritmo di Sostituzione
- Gestione della Coerenza

Parametri Caratteristici

- *Block Size*
  - *Un Blocco è un insieme di indirizzi contigui di una qualche dimensione.*
  - *Per la proprietà di località spaziale conviene portare in cache un insieme di elementi facenti riferimento ad indirizzi contigui*
- *Hit Time*
  - *La memoria cache può memorizzare solo un piccolo sottoinsieme di blocchi presenti in memoria principale*
  - *Il numero di cicli di clock necessari per caricare un elemento che si trova in cache (in genere coincidente con 1 ciclo di clock)*

Parametri Caratteristici

- *Miss Penalty*
  - *Quando un elemento non viene trovato in cache si ha un miss*
  - *La Miss Penalty rappresenta il numero di cicli di clock necessari per caricare l'elemento dalla memoria principale nella cache (o eventualmente direttamente nel processore)*
  - *(Access Time)*
    - *Tempo di accesso alla memoria principale (o a una cache di livello superiore)*
  - *(Transfer Time)*
    - *Tempo di trasferimento del blocco in cui è presente l'elemento referenziato dalla memoria principale alla memoria cache.*

Parametri Caratteristici

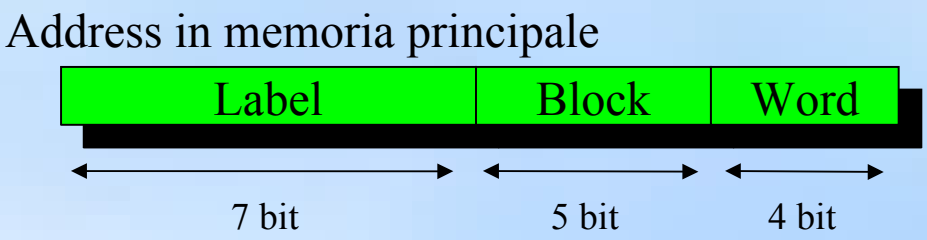
- *Miss Rate*
  - *Percentuale di miss (relativo all'architettura del sistema, alla funzione di mapping, agli algoritmi di gestione della coerenza, all'applicazione in esecuzione ...)*
- *Cache size*
  - *Raramente supera 1 Mbyte*
- *Average Memory – access time = t(hit) + (1-hit_rate) t(miss)*
  - *t(miss) in genere è circa 10 volte t(hit)*

Funzione di Mapping

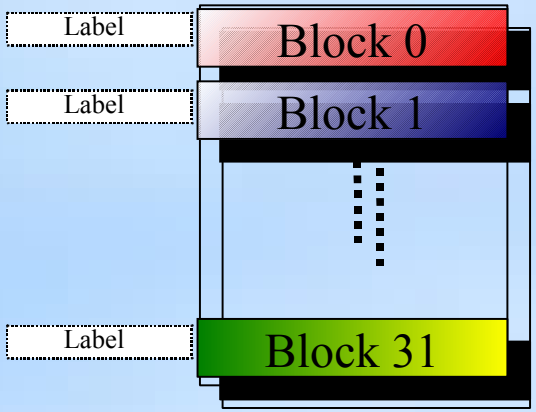
- Rappresenta un altro parametro identificativo della cache
- Specifica la corrispondenza tra i blocchi della memoria principale e quelli della cache. Vengono distinte in questo modo tre categorie di cache
  - *Direct Mapped*
  - *Fully Associative*
  - *Set Associative*
- *Per gli esempi si farà riferimento ad una memoria principale di 1 Mbyte, organizzata in blocchi di 16 word,ognuna di 16 bit (64 Kwords) ed una cache di 32 blocchi di 16 word ciascuno (8 Kbyte)*

Direct Mapped Cache

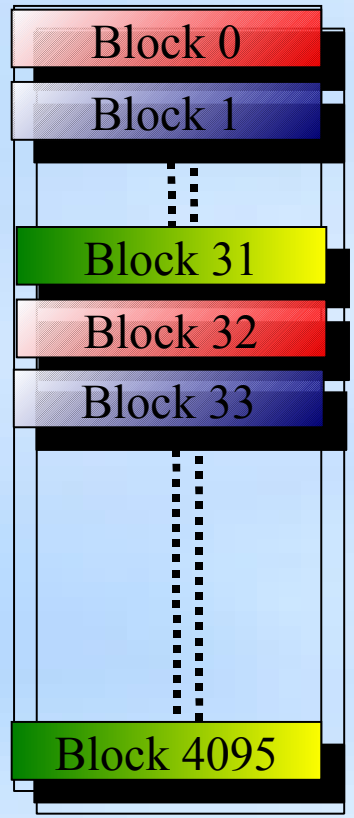
- Ogni blocco può essere inserito in un solo posto nella cache.
- La funzione di mapping è di solito una funzione modulo



- La presenza o meno del blocco in cache viene verificata comparando il campo Label nell'address con il campo Label nella cache
- Il campo Block identifica il blocco all'interno della cache
- Il campo word indirizza la word nel blocco

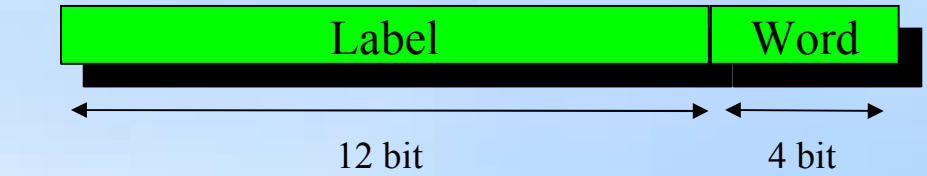


Cache

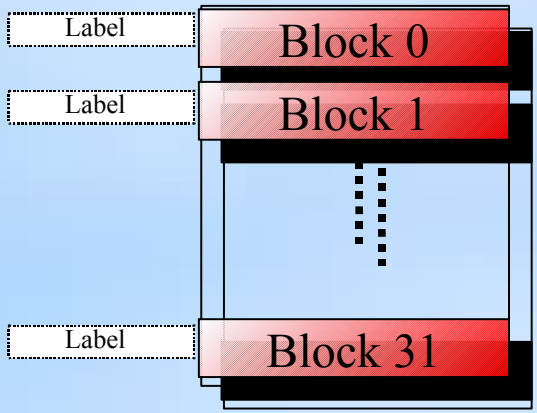


Fully Associative Cache

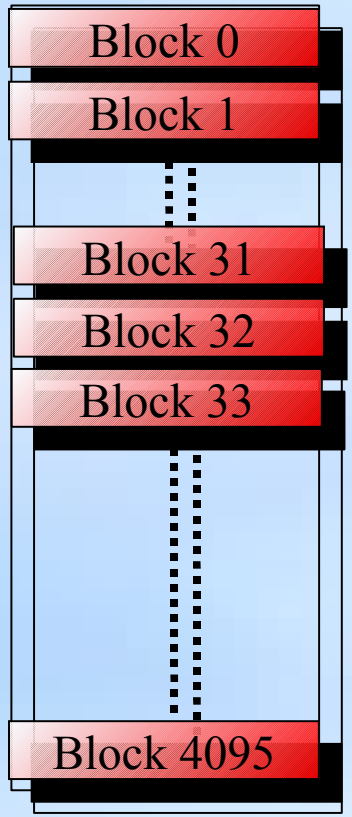
- Ogni blocco può essere inserito ovunque nella cache



- La presenza o meno del blocco in cache viene verificata comparando il campo Label nell'address con il campo Label nella cache
- Il campo word indirizza la word nel blocco (*Block Offset*)

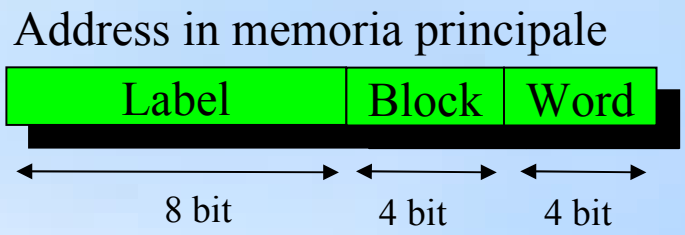


Cache

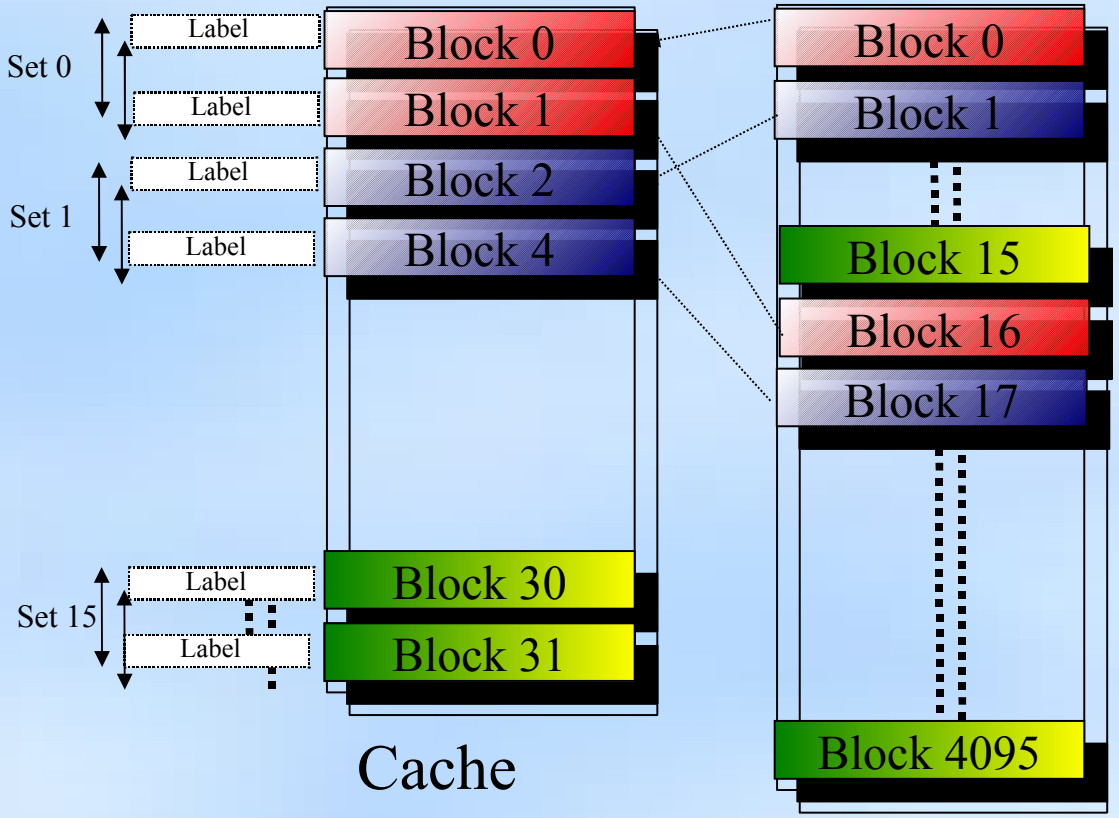


Set Associative Cache

- Un blocco può essere posizionato in un ristretto insieme (*set*) di blocchi nella cache.



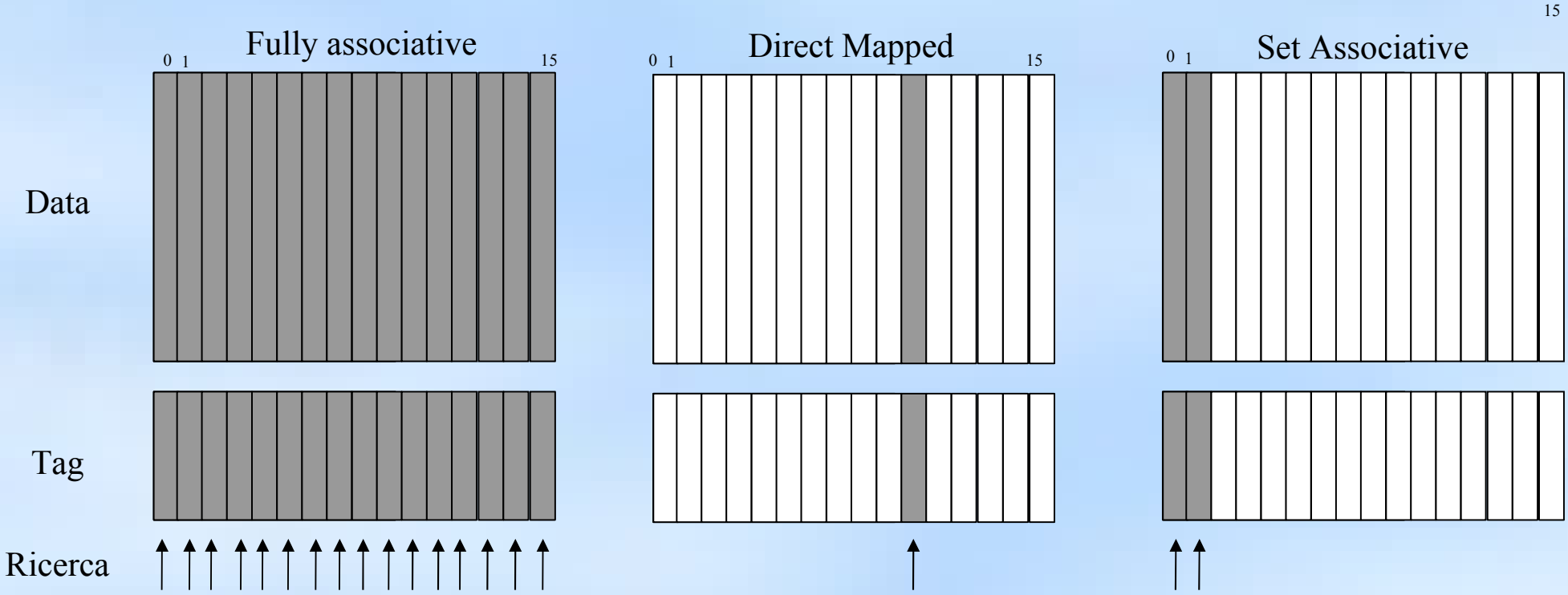
- Un blocco viene prima mappato in un set e può essere inserito ovunque nel set
- La funzione di mapping del set in genere è una funzione in modulo
- Se ci sono *n* blocchi in ogni *set* la cache viene detta : *n-way set associative*



Ricerca di un Blocco in una Cache

- La cache include un campo Label (*Tag*) per ogni blocco per ricavare l'indirizzo del *block-frame*

- Ai fini della ricerca di un blocco in una cache tutte le label che possono contenere informazioni su di esso vengono confrontate (*comparate*) in parallelo. (Anche il dato viene letto parallelamente al confronto in modo da averlo immediatamente pronto se si ha un hit)



Considerazioni

- *Bisogna tenere in conto dello spazio (<=> costo) occupato dalle Lable (una per ogni blocco)*

(direct mapped -> set associative -> full associative)

- *Spesso c'è bisogno di etichettare i blocchi in cache che sono stati modificati*
  - *Viene utilizzato in genere anche un bit per la modifica (dirty bit)*
  - *Serve solo nei sistemi che non fanno uso del metodo write throught*

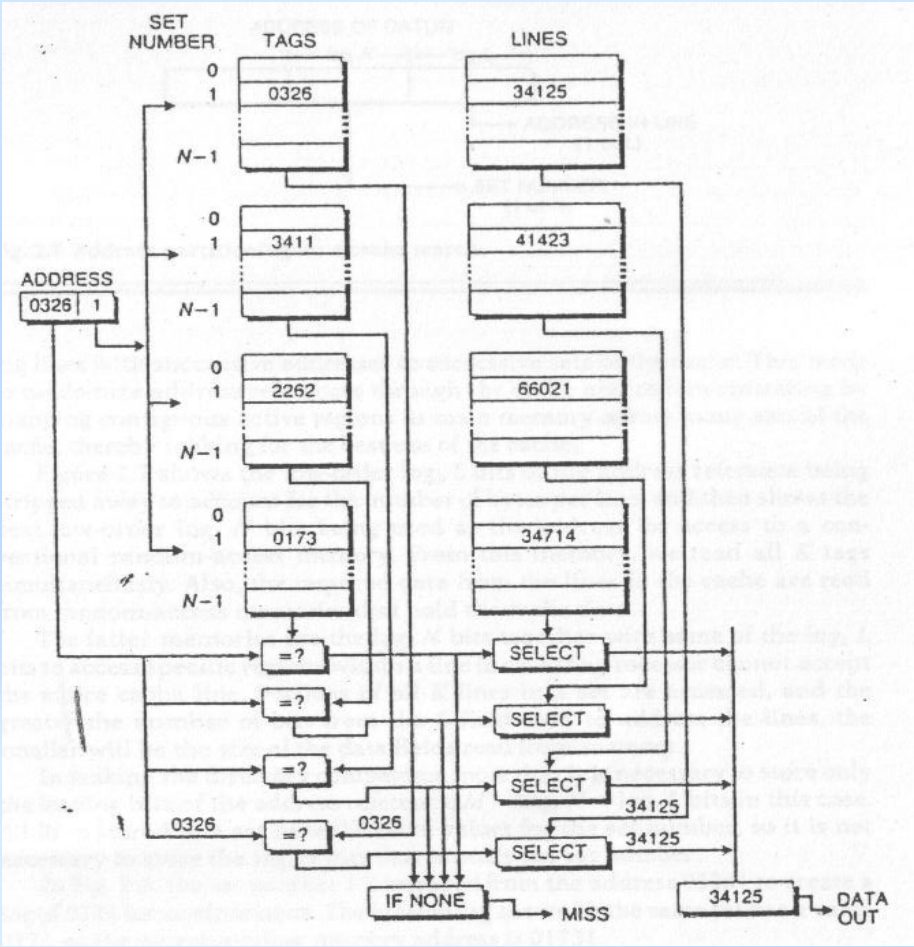
Considerazioni

- Una cache potrebbe contenere informazioni non piu' valide
  - C'è bisogno di un meccanismo per invalidare i blocchi nella cache: il modo più comune per farlo è quello di aggiungere un bit alla label (*valid bit)*
  - *Il bit viene posto a 0 all'accensione e ogni volta che nella memoria principale vengono caricati programmi e dati nuovi da disco (DMA). I dati non passano per la cache per motivi sia di costo che di prestazioni*
  - *Viene posto a 1 la prima volta che il blocco viene caricato dalla memoria principale*
  - *Ogni volta che un blocco in memoria principale viene aggiornato con codice e dati che non sono passati per la cache, viene effettuato un controllo per determinare se l'elemento è presente nella cache. Se lo è il suo bit di validità viene azzerato garantendo che non ci siano dati obsoleti in cache*

Considerazioni

- Quando deve essere inserito un blocco in un set già pieno :
  - Una cache direct mapped possiede un solo blocco disponibile per ogni blocco in memoria principale: Un nuovo blocco che deve essere copiato nella cache in blocco già pieno provoca *sempre* una copia del vecchio blocco vecchio in memoria principale
  - Una cache fully associative ha a disposizione tutta la cache per il caching dei blocchi: Un nuovo blocco che deve essere copiato nella cache fa uscire un altro già residente solo se l'intera cache è già piena
  - Una cache set-associativa ha per ogni blocco in memoria principale uno spazio pari al suo grado di associatività: Un nuovo bloccohe deve essere copiato nella cache fa uscire un altro già residente solo se il set ad esso associato è già pieno.

Un Esempio Completo



Algoritmi di Sostituzione

- Quando si ha un *cache miss* su un blocco:
  - Se la cache ha blocchi disponibili per inserire un nuovo blocco, si procede al caricamento dalla memoria nel set relativo al blocco
  - In una Direct Mapped Cache se il blocco è occupato, si procede alla sostituzione del blocco (se il *dirty bit == 0, non si effettua la copia in memoria principale)*
  - *Per le cache Set e Fully associative, si cerca nel set un blocco col dirty bit ancora a 0 (non modificato) e se viene trovato si procede alla sostituzione. Nel caso tutti i blocchi nel set siano stati modificati, si deve procedere alla ricerca del "migliore" blocco da sostituire. Gli approcci piu' utilizzati sono 2 : Random e Least recently Used (LRU) replacement*

Algoritmi di Sostituzione

- Random
  - Viene usato per distribuire uniformemente l'allocazione dei blocchi. Il blocco candidato viene scelto in modo *pseudocasuale. Utilizzare un algoritmo pseudocausale e quindi riproducibile risulta particolarmente indicato per facilitare le operazioni di debugging*
- *LRU*
  - *Riduce la probabilità di sostituire blocchi che verranno acceduti nel prossimo futuro.*
  - *Il blocco sostituito è quello che non si usa da piu' tempo*

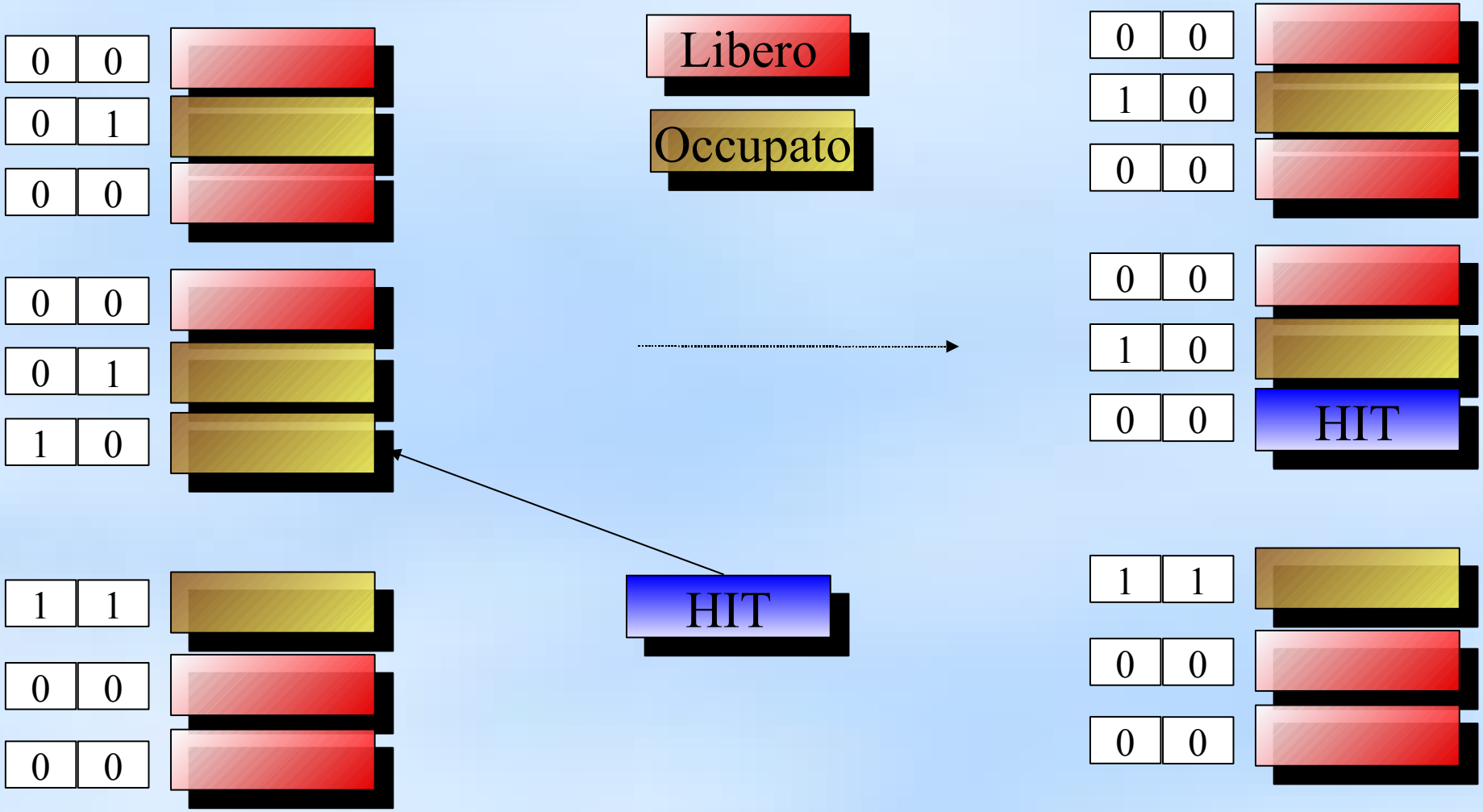
LRU

- *Il controllore ha bisogno di mantenere traccia degli accessi ai blocchi mentre l'elaborazione prosegue*
- *Viene utilizzato un contatore per ogni blocco*
  - *Quando c'e' un hit :*
    - *Il contatore del blocco dell'hit viene azzerato*
    - *I contatori con valori inferiori a quello dell'hit vengono incrementati*
    - *Gli altri rimangono invariati*
  - *Se c'e' un miss e c'è un blocco libero*
    - *Viene inserito il nuovo blocco con contatore=0*
    - *Tutti gli altri contatori vengono incrementati di 1*

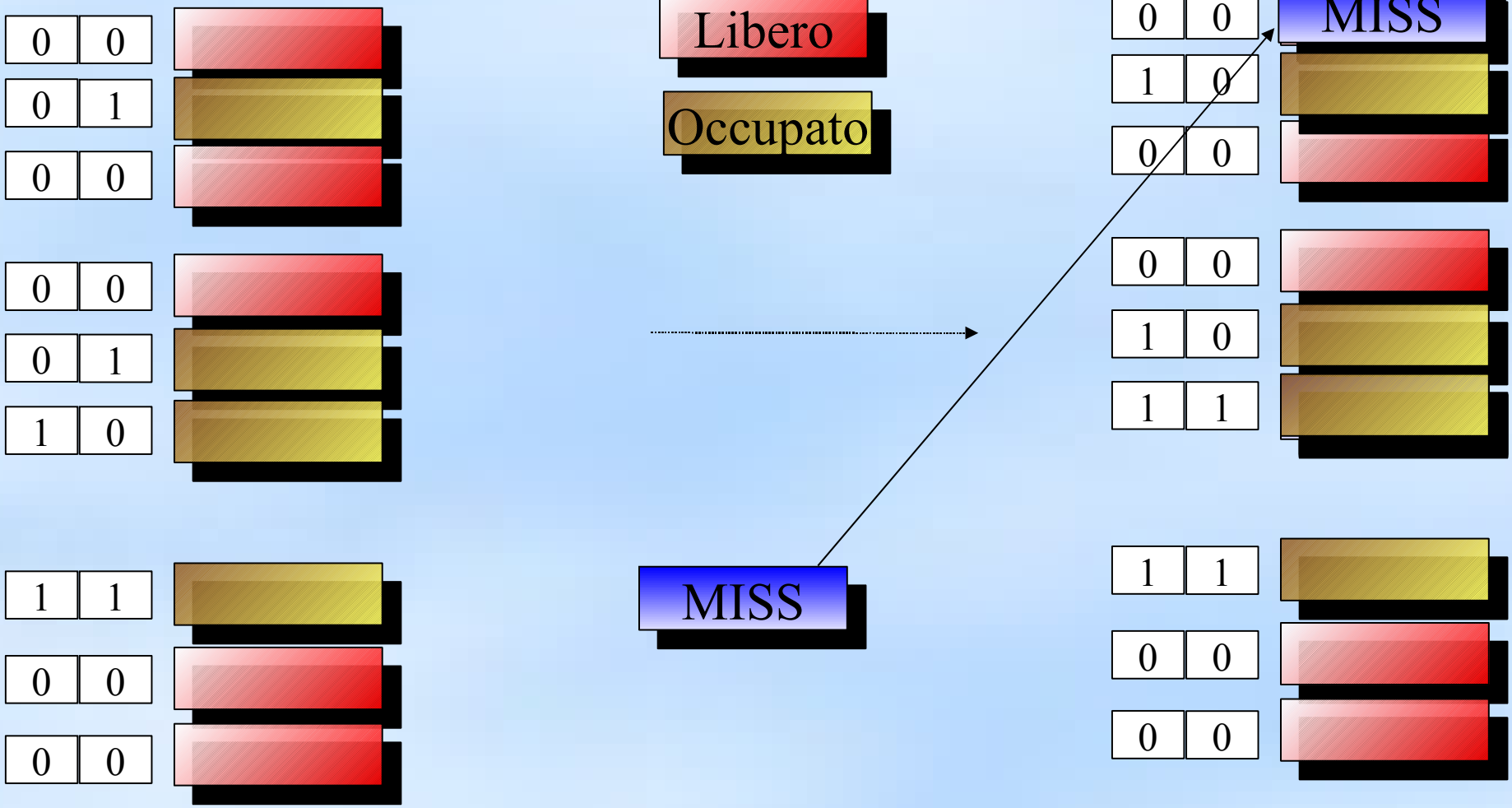
LRU

- *Se c'e' un miss e tutti i blocchi nel set sono occupati :*
  - *Si sposta un blocco con il valore più alto di conteggio*
  - *Si inserisce il nuovo blocco con il contatore a 0*
  - *Tutti gli altri contatori vengono incrementati di 1*

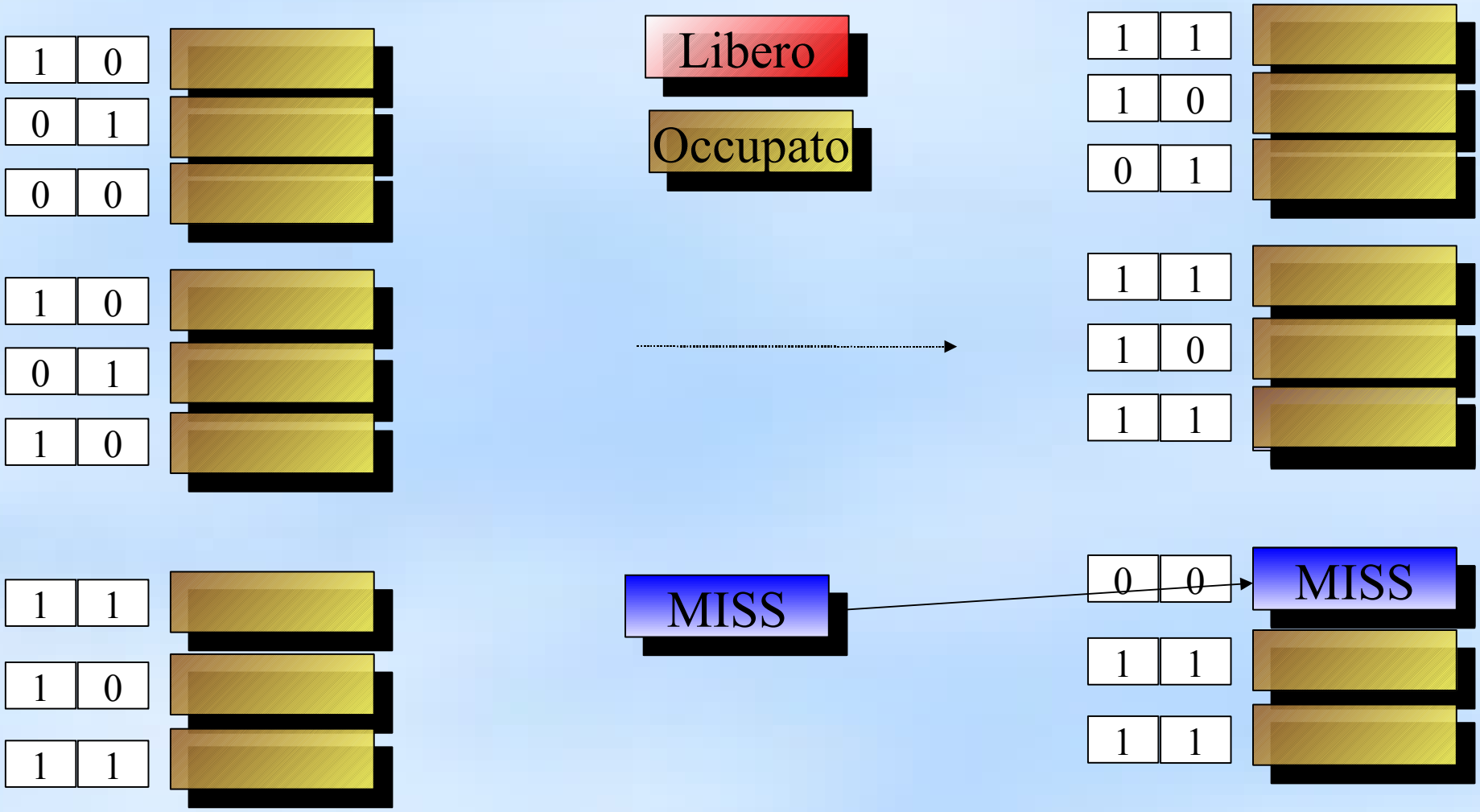
LRU


Libero
Occupato
0 0
0 1
0 0
0 0
1 0
0 0
1 1
0 0
0 0
0 0
1 0
0 0
0 0
1 0
0 0
HIT
1 1
0 0
0 0
HIT

LRU


Libero
Occupato
0 0
0 1
0 0
0 0
0 1
1 0
1 1
0 0
0 0
MISS
0 0
1 0
0 0
0 0
1 0
1 1
1 1
0 0
0 0

LRU


Libero
Occupato
1 0
0 1
0 0
1 0
0 1
1 0
1 1
1 0
1 1
1 1
1 0
0 1
1 1
1 0
1 1
MISS
0 0
MISS
1 1
1 1

Considerazioni

- Risultati sperimentali mostrano che l'algoritmo Random ha prestazioni pressocchè identiche all' LRU quando la cache supera i 64 Kbyte
- L'Hardware per implementare l'algoritmo Random è molto più semplice di quello di LRU
- Esistono altri algoritmi di sostituzione (come il FIFO) ma in genere hanno performance peggiori dell'algoritmo Random e sono più difficili da implementare...

Gestione della Coerenza

- Le operazioni di lettura sono predominanti sulle operazioni di scrittura (tutte le operazioni su istruzioni sono solo operazioni di lettura)
- Nel caso di una operazione di lettura, la lettura del dato puo' essere fatta parallelamente al confronto delle label. Se c'e' un Miss non ci sono problemi perchè la cache non è stata comunque sporcata
- Nel caso di una operazione di scrittura, non è possibile cominciare l'operazione finchè non si ha un'hit.
- Visto che l'operazione di accesso all'area dati della cache, non puo' avvenire in parallelo con il confronto delle label, in genere, un'operazione di scrittura impiega più tempo di una di lettura.

Operazioni di Write

Ci sono due politiche di base per le operazioni di write :

- *Write Through (o store through)*
  - *L'informazione viene scritta in intrambi i blocchi (cache e memoria principale)*
- *Write Back (o copy back)*
  - *L'informazione viene scritta solo nel blocco della cache. Questi viene scritto nella memoria principale solo quando il blocco deve essere scambiato.*
  - *Fa uso del dirty bit.*

Vantaggi e Svantaggi

Sia Write back che write through hanno i loro pro e contro

- *Write Through*
  - *PRO: La memoria principale ha è sempre allineata con la cache (importante per sistemi multiprocessore)*
  - *PRO: più facilmente implementabile*
  - *CONTRO: scritture alla velocità della memoria più lenta*
  - *CONTRO: 1 scrittura per ogni modifica del blocco=> occupa più banda (e quindi viene utilizzata per lo piu' tra cache L1 e cache L2)*
- *Write Back*
  - *PRO: scritture alla velocità della memoria cache*
  - *PRO: scritture multiple sullo stesso blocco richiedono 1 sola scrittura nella memoria piu' lenta (meno occupazione di banda)*

Vantaggi e Svantaggi

- *Write Through*
  - *Quando una CPU attende il completamento di una scrittura si dice in write stall.*
  - *Un'ottimizzazione comune per ridurre lo stallo è quello di utilizzare dei write buffers che permettono al processore di continuare l'esecuzione del programma mentre la memoria viene aggiornata*

Write Miss

Nel caso in cui si abbia un write miss si hanno due opzioni:

- *Write Allocate :*
  - *Il blocco viene caricato in memoria e si effettua l'operazione di write (che genererà un hit) sulla cache*
  - *In genere utilizzata assieme a Write-Back (le future modifiche si faranno sul blocco nella cache)*
- *No Write Allocate :*
  - *Il blocco viene modificato direttamente nella memoria principale e non viene inserito nella cache*
  - *In genere utilizzata assieme a Write-Through (le scritture avvengono comunque anche nella memoria principale ed è inutile inserire il blocco nella cache)*