

Gerarchia di memorie

- Regole base dell'efficienza:

- Rendere di massima velocità il caso comune
- Piccolo significa (in generale) veloce

Nei programmi vi è una significativa presenza di località spaziale e temporale

- Quindi:

- Sfruttare la località spaziale e temporale, mantenendo il più vicino possibile alla CPU i dati e le istruzioni utilizzati più di frequente
- La gerarchia delle memorie deve prevedere in successione memorie sempre più grandi ma necessariamente più lente

Gerarchie di memorie

La memoria M sia caratterizzata dai seguenti parametri:

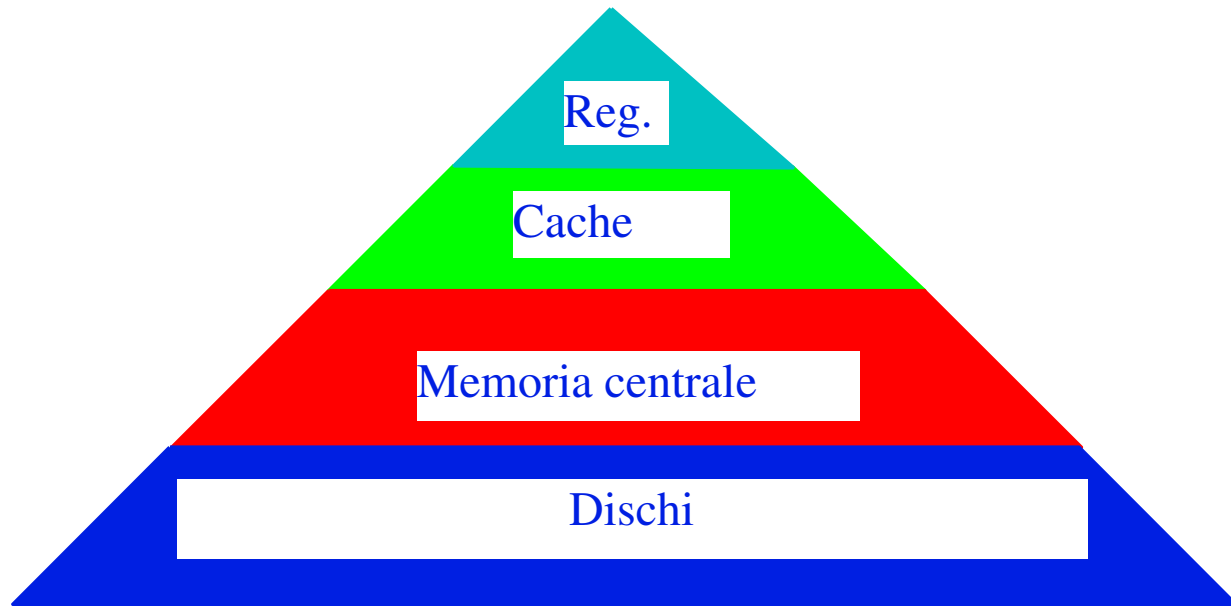
- T : tempo di accesso
- S : dimensione
- C : costo per byte

Date due memorie M_1 e M_2 , caratterizzate rispettivamente dalle terne (T_1, S_1, C_1) e (T_2, S_2, C_2) , con:

- $T_1 < T_2$
- $S_1 < S_2$
- $C_1 > C_2$

si vuole che la memoria gerarchica approssimi il meglio possibile la terna (T_1, S_2, C_2)

Gerarchie di memorie

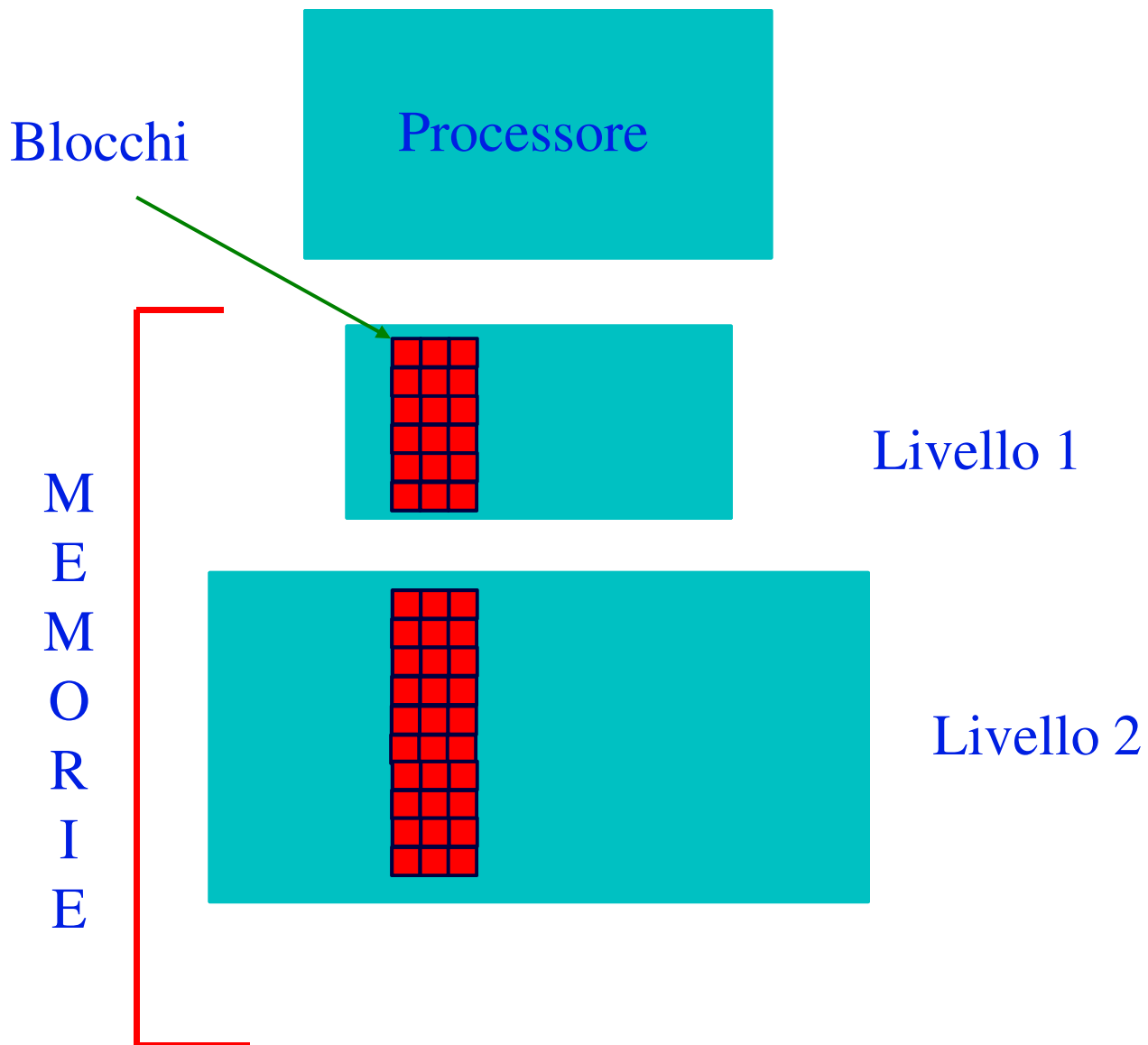


Parametri tipici in un PC moderno:

Tipo	Dimensioni	Velocità	Larghezza di banda (Mb/s)
Reg.	~ 1Kb	1-2 ns	2000-4000
Cache	~ 512 Kb	7-15 ns	1200- 600
Memoria	< 2 Gb	60 ns	200
Dischi	~ 100-200 Gb	10 ms	30

BLOCCHI

- Gerarchia di memoria: più livelli
- Blocco: unità di informazione minima scambiata fra livelli (ex: 32 bytes)



GENERALITA'

- **Hit**: un accesso alla memoria che trova l'informazione cercata nel livello corrente
- **Hit Rate**: frequenza di accessi trovati nel livello corrente (h , compreso tra 0 e 1)
- **Miss**: un accesso alla memoria che non trova l'informazione cercata nel livello corrente
- **Miss Rate**: $(1 - h)$
- **Hit Time**: tempo di accesso al livello corrente, **T_h**
- **Miss Penalty**: tempo per rimpiazzare un blocco prelevandolo dal livello successivo, **T_m** . Poi occorre comunque un **T_h** per accedere.

TEMPI DI TRASFERIMENTO

Tempo di accesso medio, T_a :

$$T_a = T_h + T_m(1-h)$$

N.B.: T_h è e sempre presente

Es. $T_h(\text{cache}) = 25 \text{ ns}$, $h = 0.85$

$T_m(\text{cache}) = T_h(\text{main memory}) = 200 \text{ ns}$

-> $T_a = 55 \text{ ns}$

T_a può anche esser espresso come:

$$T_a = T_m (1/k + (1-h))$$

Dove:

$k = T_m/T_h$ rapporto tra i tempi di accesso ai due livelli, low e high (lento e veloce) della gerarchia

--> se il rapporto k è alto (es., 10-20 tra dischi e main memory), T_a è dominato dal miss rate (Es: $T_m = 10 \text{ ms}$, miss rate dell'1% --> $T_a = 100 \mu\text{s}$)

Classificazione dei *miss*

- Inevitabili:
 - Primo accesso a un blocco
- Capacità
 - Miss che accadono quando si accede a un blocco sostituito da altri a causa della limitata capacità della memoria
- Conflitto
 - Miss che accadono perché i blocchi sono sostituiti a causa di una specifica strategia di "set-mapping", anche se la capacità della memoria non è esaurita

CACHE MEMORY

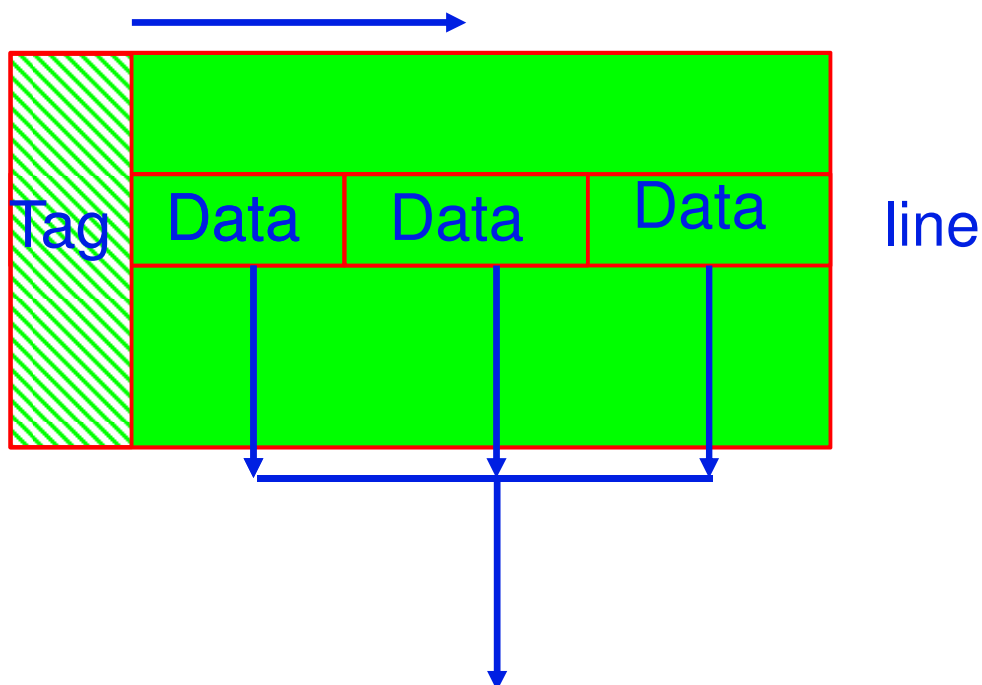
La CACHE MEMORY consiste di due parti:

CACHE DIRECTORY

contiene **tag** degli indirizzi e **bit di stato** (valid, dirty..)

CACHE RAM

consiste in un insieme di blocchi (**block frames, o linee**) contenenti i dati associati alla tag



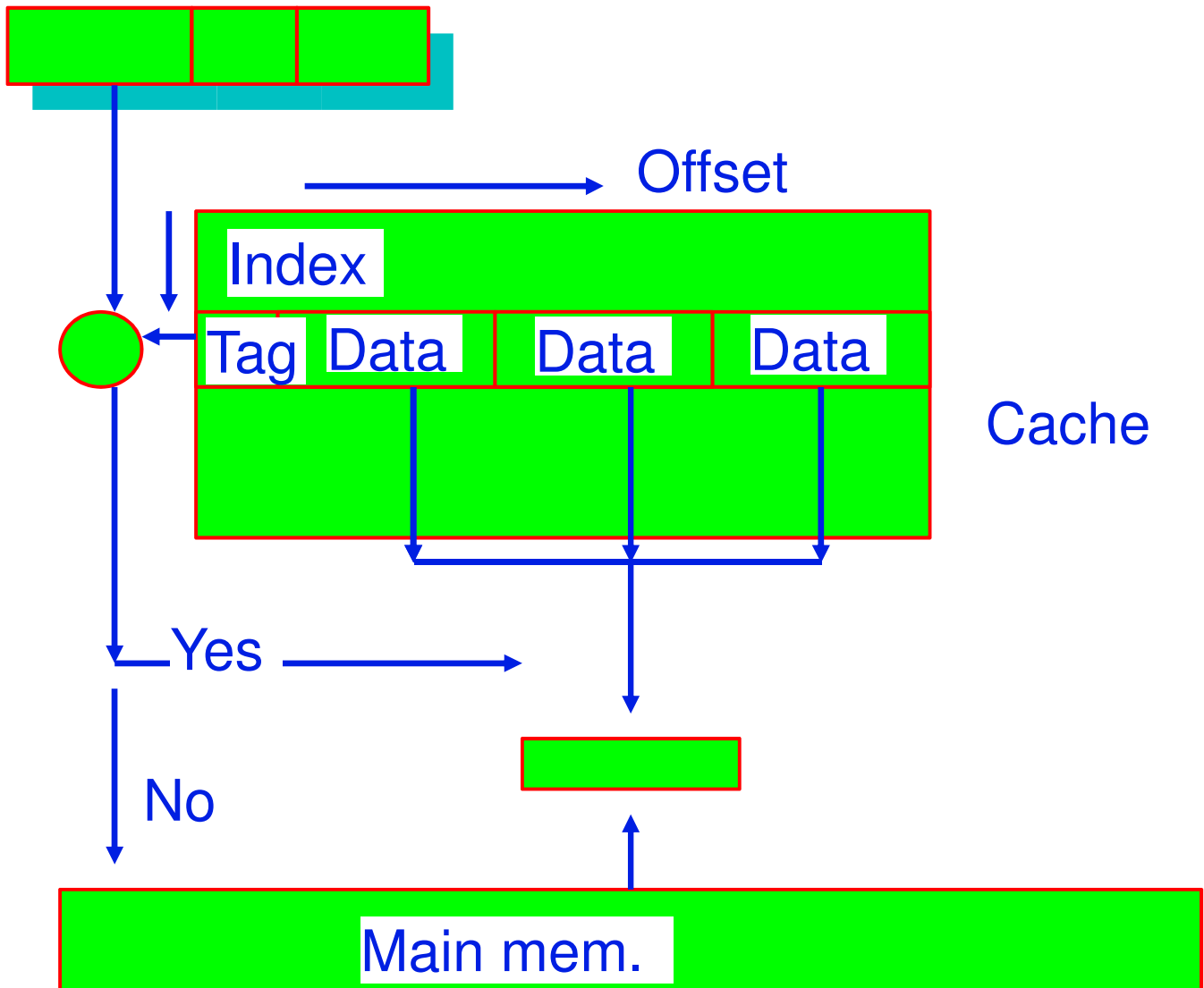
ORGANIZZAZIONE DI UNA CACHE

Etichetta (Tag)

Indice (Indice di blocco)

Offset (Spiazzamento nel blocco)

Tag Index Offset



Dimensione della CACHE

☐ Dimensione:

Capacità dei dati (niente tag e/o stato):

N delle linee * N byte in una linea =
 N dei set (2^{Index}) * N delle vie (associatività)
* N byte in una linea (2^{Offset})

☐ Tipiche dimensioni:

- ☐ On-chip: 16-64 K bytes
- ☐ On-board 256-1024 K bytes

☐ Dimensione del set (numero delle vie):

- ☐ Velocità della RAM dati/tags
- ☐ (numero di comparatori conveniente)

Tipo di CACHE

- ❖ Unificata (dati e istruzioni -> Von Neumann)
 - ❖ Meno costosa
 - ❖ Più versatile
- ❖ Dati e istruzioni separate (Harvard)
 - ❖ Doppia larghezza di banda (posso fare due accessi contemporanei)
 - ❖ Piazzamento più vicino alle porte I e D
 - ❖ Assenza di interlocks in caso di richieste simultanee
- ❖ Le CACHE dovrebbero essere divise se l'accesso simultaneo a dati e istruzioni è frequente (come nel caso dei processori moderni)

Esempio

- ❖ Scegliere o due CACHE da 16 Kbytes per dati e istruzioni (caso a) o una da 32 Kbytes unificata (caso b)

$$T_{cache} = 1 \text{ ciclo } T_{memory} = 10 \text{ cicli}$$

$$a) \quad I_{miss} = 5\% \text{ (Istruzioni)}$$

$$b) \quad D_{miss} = 6\% \text{ (Dati)}$$

75% degli accessi sono istruzioni

$$T_{avg} = (1 + 0.05 * 10) * 0.75 + (1 + 0.06 * 10) * 0.25 = 1.525$$

$$b) \quad Miss = 4\%$$

~~$$T_{avg} = (1 + 0.04 * 10) = 1.4$$~~

$$T_{avg} = 1.4 + \text{cicli persi per interferenza !!}$$

- ❖ Nei RISC i cicli persi per interferenza sono certamente **SUPERIORI** a 0.125 !

Dimensione dei blocchi

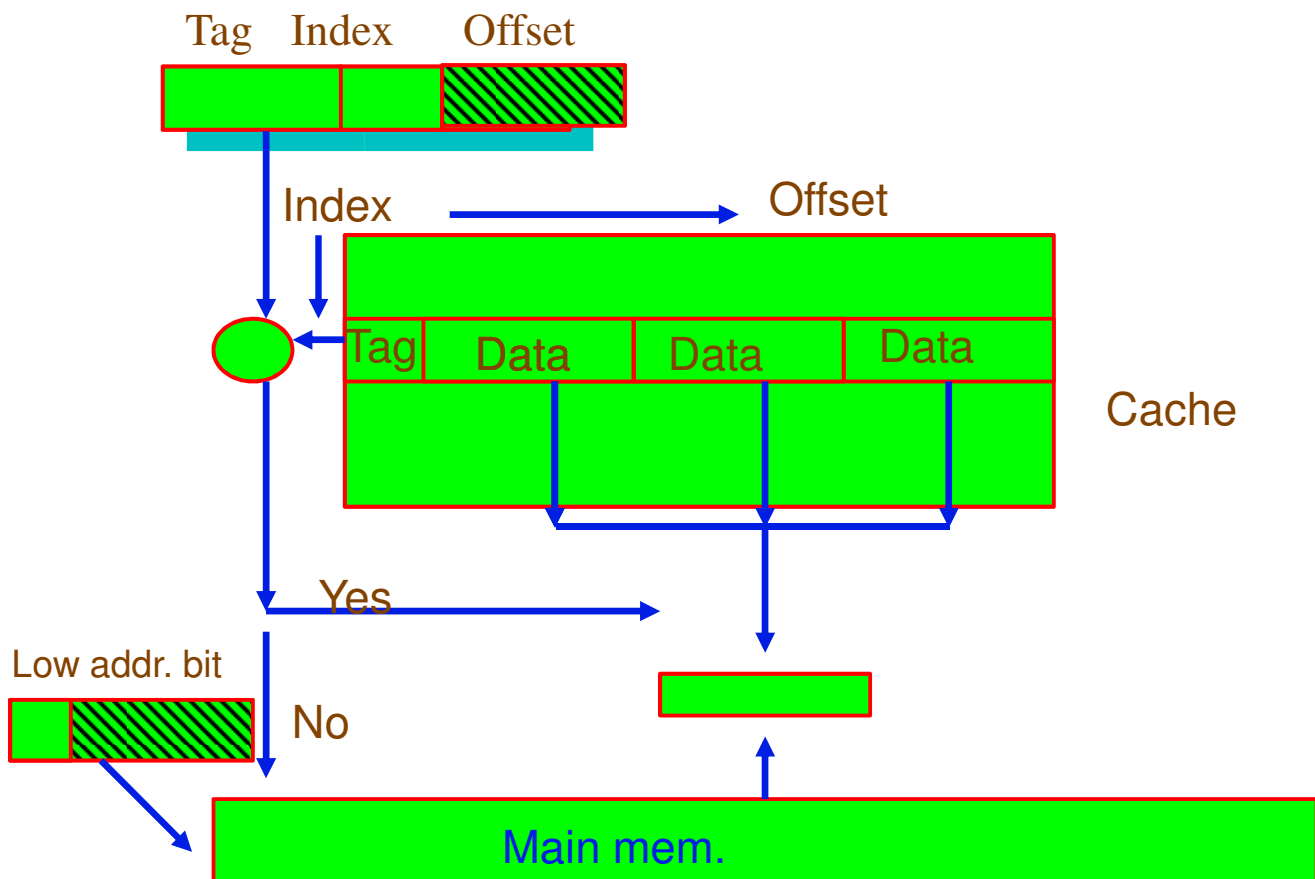
- ❖ Un blocco (linea) è la quantità di dati che:
 - a) è associata a un TAG
 - b) è trasferita da e verso la memoria
- ❖ Le cache più sofisticate permettono differenti dimensioni
- ❖ Blocchi troppo piccoli:
 - Non sfruttano la località spaziale
 - Non ammortizzano il tempo di accesso alla memoria
 - Hanno un overhead di TAG eccessivo
- ❖ Blocchi troppo grandi
 - Trasferiscono dati che non vengono poi usati
 - Rimpiazzano prematuramente dati utili
- ❖ La dimensione ottimale dei blocchi è tipicamente compresa fra 16 e 64 bytes

Associatività

DIRECT MAPPED Un blocco può essere posto in un solo indirizzo sulla cache.

Es.: tag|index|offset : 20|7|5

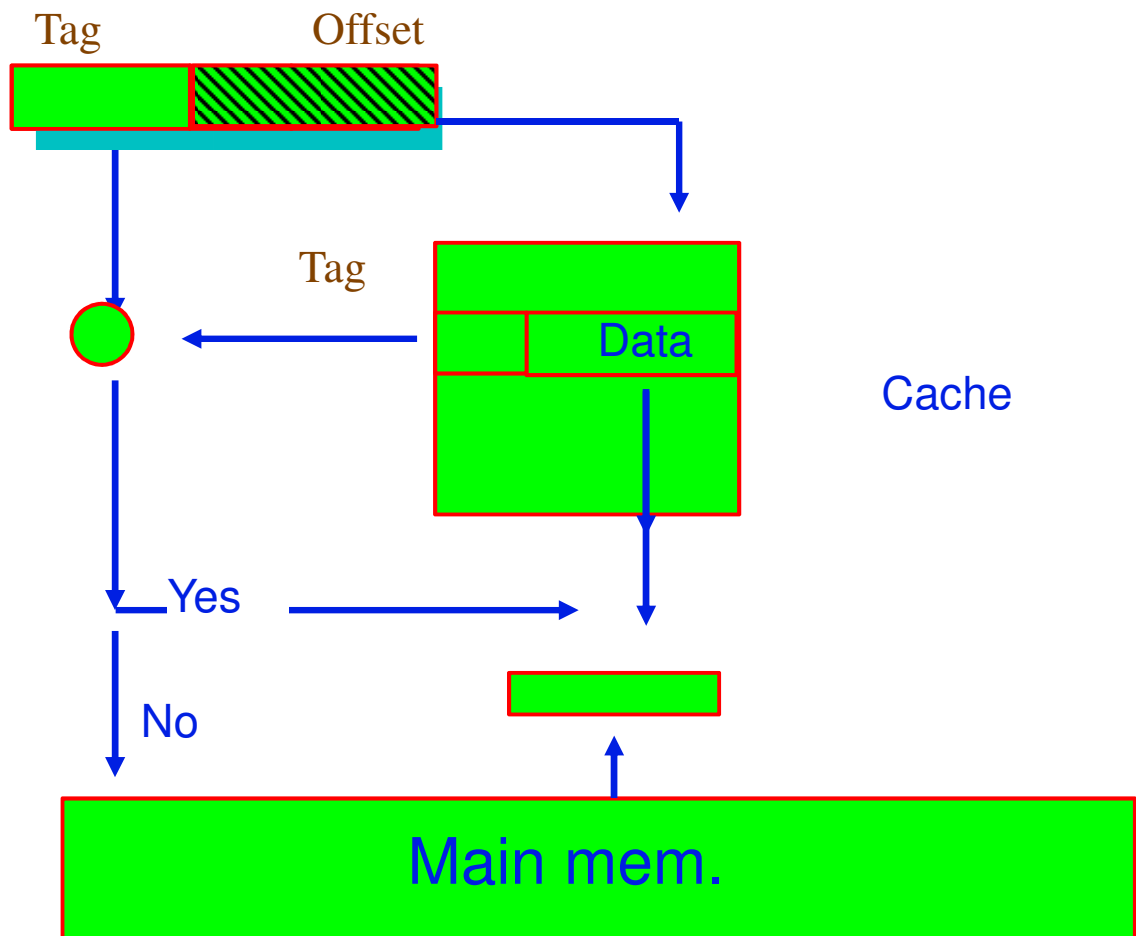
N.B.: Tutti i blocchi aventi uguali index e offset e diverso tag sono mappati allo stesso indirizzo sulla cache



Associatività

FULLY ASSOCIATIVE Ogni blocco può essere messo ovunque nella cache

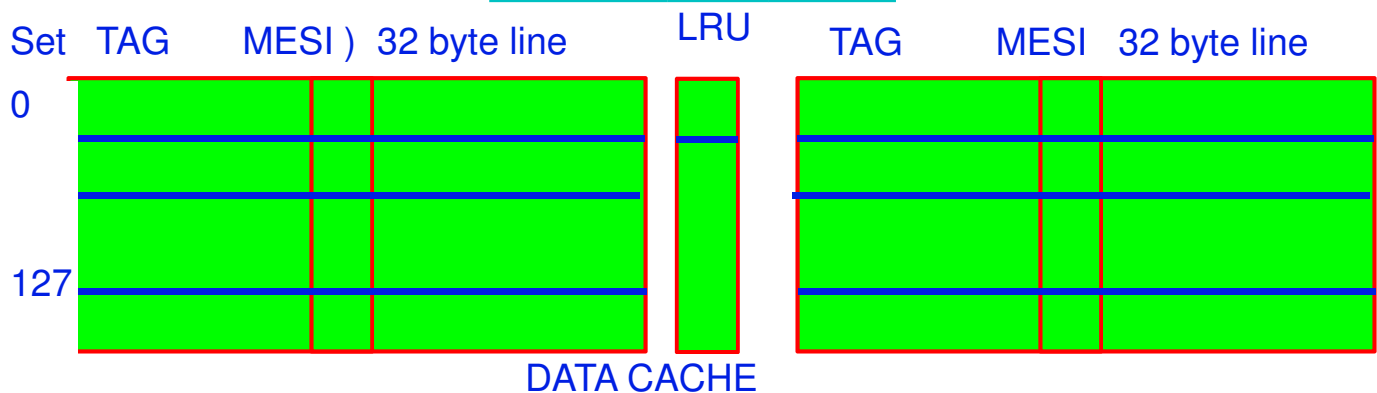
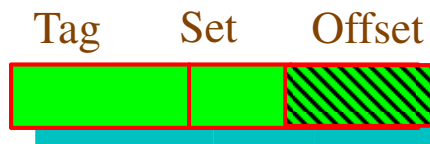
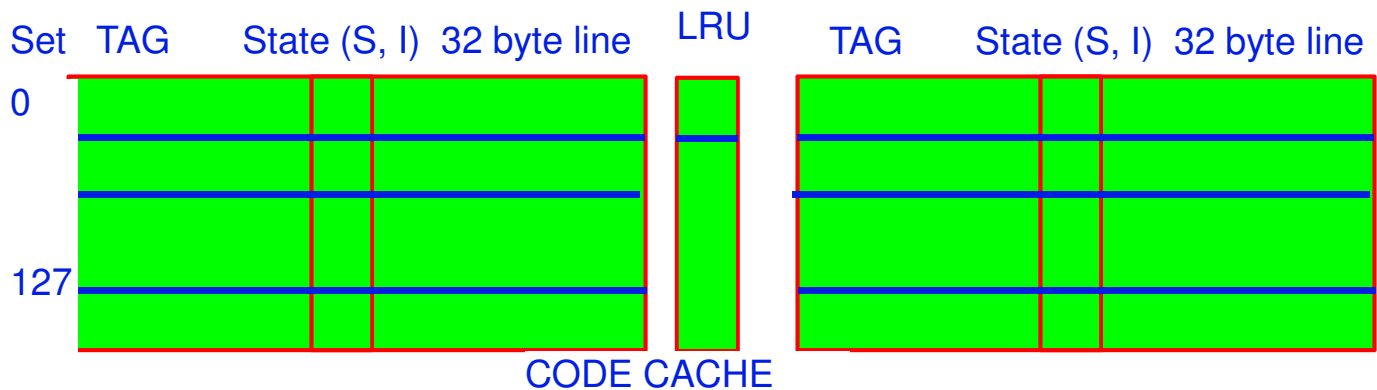
- * manca la parte index (posso piazzarlo ovunque, in qualsiasi linea – blocco)
- * più costosa e più efficiente nell'utilizzo della capacità



Associatività

SET ASSOCIATIVE (n-way) Ogni blocco può essere messo solo in un insieme di **n** blocchi fissati, chiamato **SET**; **n** è detto numero delle vie.

Es. Cache a 2 vie nel Pentium (8K)



LRU = Least Recently Used

Associatività

- Associatività più larga:
 - Minore numero di miss
 - Intuitivamente soddisfacente
- Minore associatività
 - Minore costo
 - Accesso più rapido (hit)
- Tipici valori di associatività:
 - 1 è il mapping diretto
 - 2, 4, 8, 16 per associatività per set
 - Tutti i blocchi: associatività totale

Politiche di rimpiazzamento

Ad accesso diretto: non ci sono alternative

Nel caso full o n-way associative, l'ideale sarebbe sostituire il blocco con minore probabilità di utilizzo, ovvero il cui utilizzo futuro è più lontano, ma come saperlo? Quindi:

STRATEGIA FIFO

STRATEGIA RANDOM

rimpiazzamento casuale o pseudo-casuale

STRATEGIA LRU

Least Recently Used

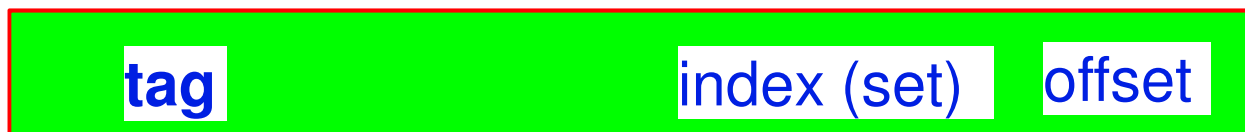
viene sostituito il blocco che è rimasto più a lungo inutilizzato

Block addr											3	2
LRU											1	0

Es dati 4 blocchi, inizialmente LRU=0

Cache PENTIUM 1

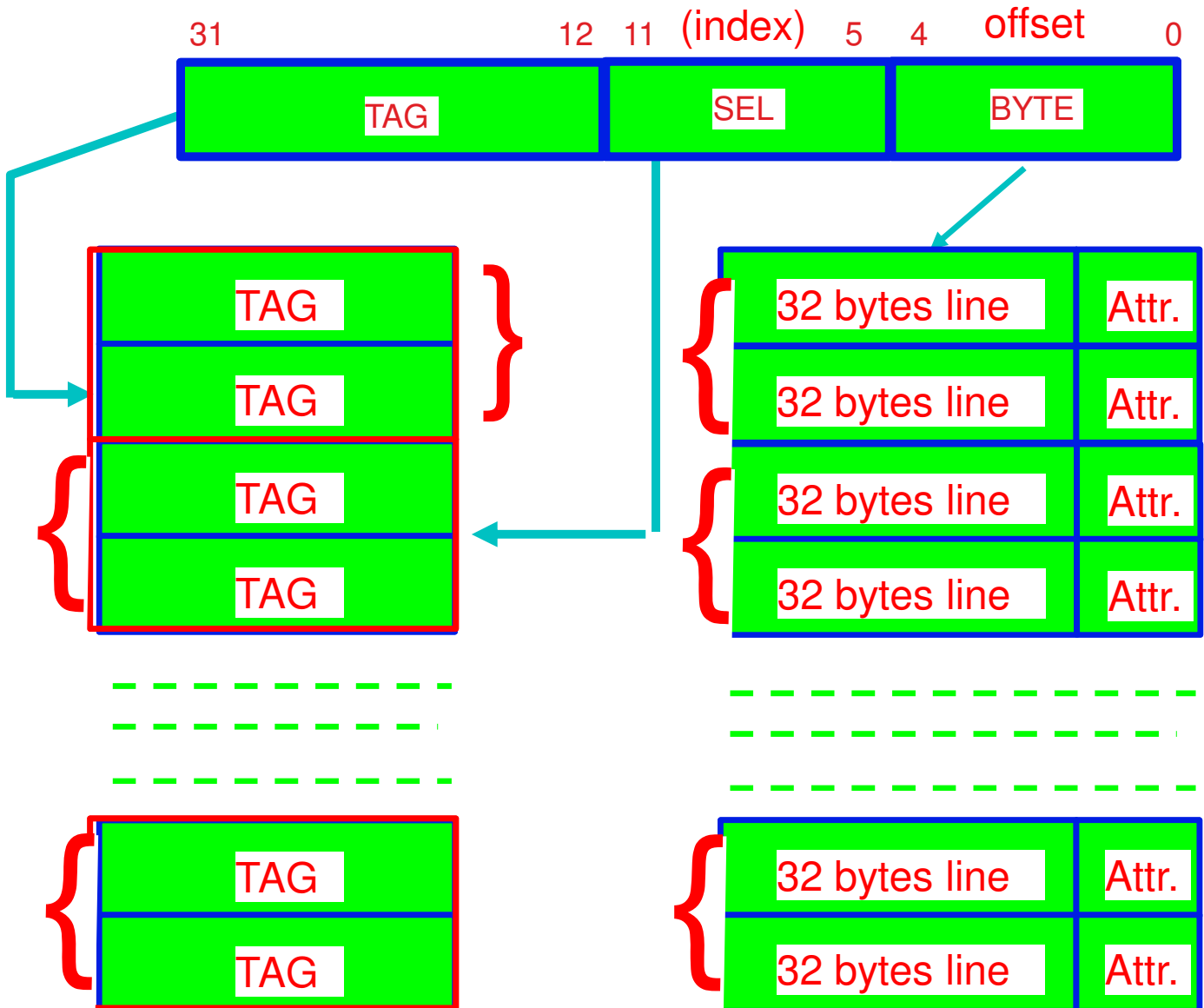
- Dato = 32 byte (copre 5 bit di indirizzo) - Blocco
- Index = 128 valori (copre 7 bit di indirizzo)
- Tag = $32 - (5+7) = 20$ bit



A31.....A12|A11...A5 |A4. .A0

- La cache del PENTIUM (dati e istruzioni) è associativa a 2 vie
- Dimensione di ciascuna cache: $128 * 2 * 32 = 8$ Kbytes
- Spesso esistono cache di secondo livello esterne al chip: ad es. 256 KB direct-mapped con 8192 set
- Algoritmo di rimpiazzamento dei dati nelle caches : **LRU**
- Data cache: tripla porta (due pipelines e un inquiry)
- Code cache: tripla porta (due accessi di linea e un inquiry)
- Cache disabilitabili via hardware o software
- Per ogni linea di **data** cache vi sono **due bit di stato** (uno per sapere se il dato è stato modificato rispetto all'originale letto dalla memoria centrale)
- Per ogni linea di **code** cache vi è **un bit di stato**
- In entrambi i casi vi è il bit di LRU per ogni set di due linee che indica quale linea del set è stata impiegata più recentemente

Cache PENTIUM



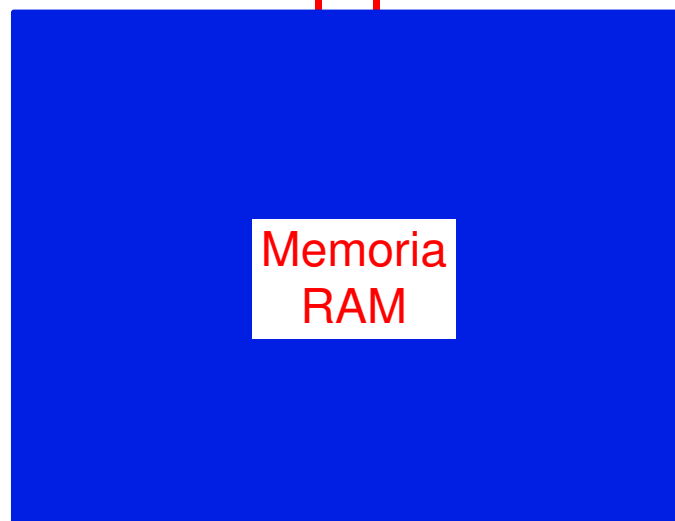
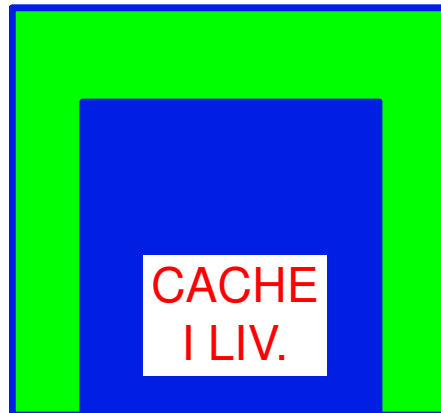
- TAG -> 20 bit
- 128 set - associatività a due vie

Gestione letture e scritture

- ❖ Gestione della lettura:
 - ❖ Se hit, ok
 - ❖ Se miss, alloca (read-allocate)
- ❖ Gestione della scrittura più complessa
 - ❖ Le letture si fanno contemporaneamente al confronto dei tags. Le scritture no (e quindi le scritture sono spesso più lente)
- ❖ "Hit" in scrittura
 - ❖ Scrittura in cache e in memoria: write-through (store through)
 - ❖ Scrittura solo nella cache:-> write-back (store in + copy-back), più efficiente
- ❖ In caso di miss si rimpiazzano i blocchi ?
 - ❖ Si - write-allocate, quasi mai usato
 - ❖ No - no-write-allocate (un dato scritto viene anche letto: allocazione a carico della read)

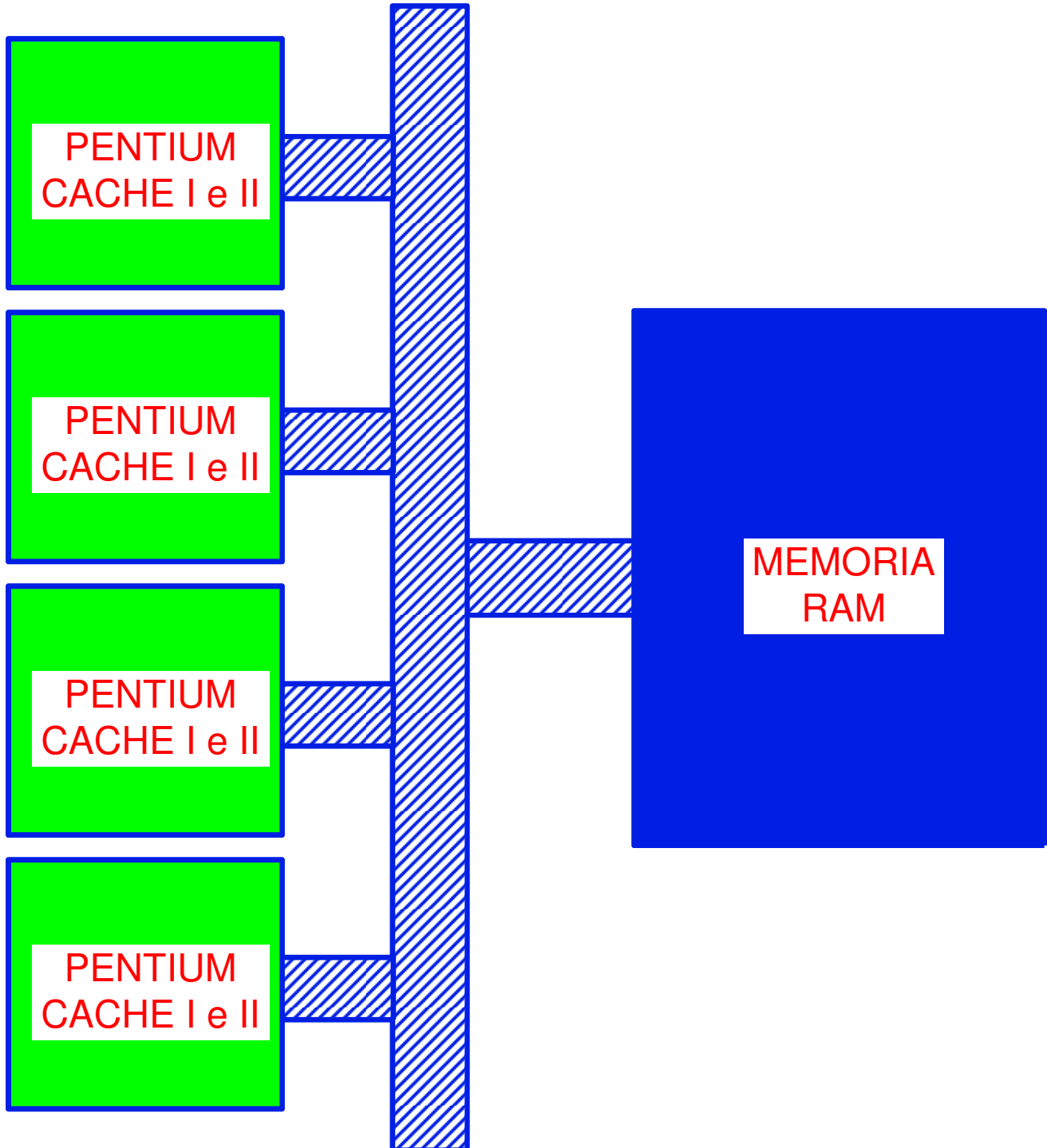
Cache multiple

PENTIUM μ P



Multiprocessor

BUS MULTIPROCESSOR



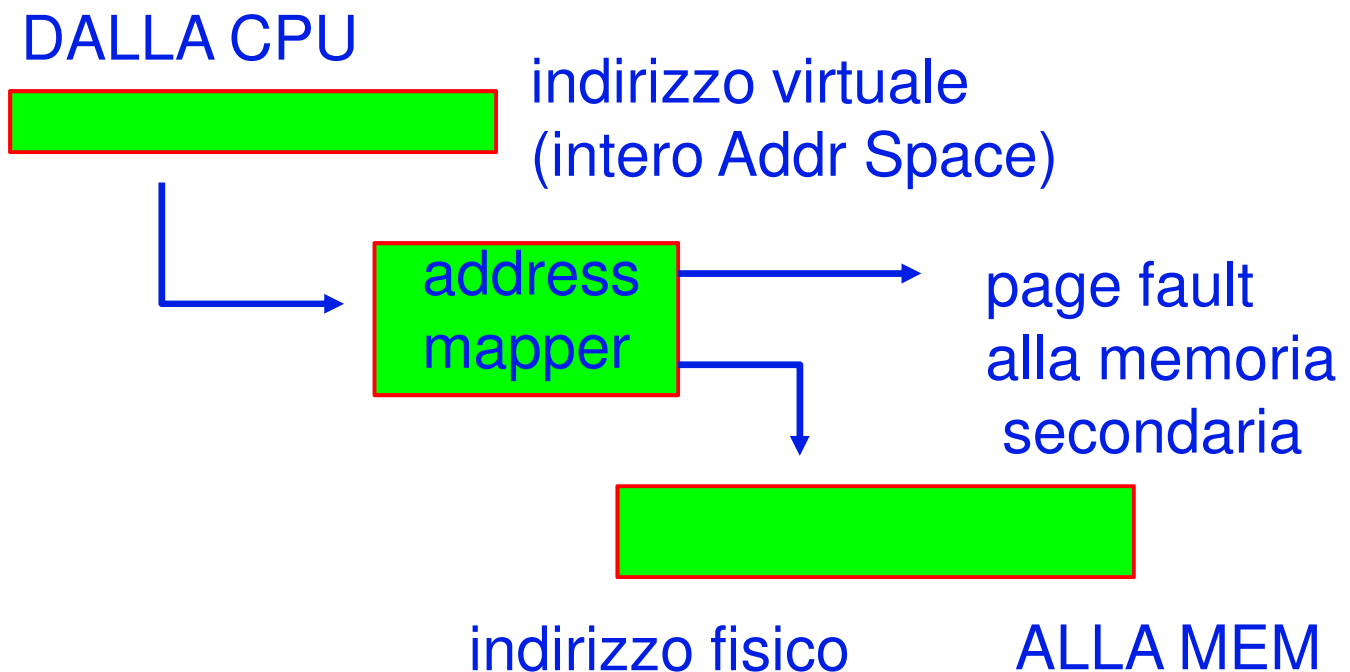
M.E.S.I.

(Modified – Exclusive - Shared – Invalid)

- Protocollo per sistemi multiprocessore
- Per ogni linea presente nella cache è necessario sapere in quale stato si trova per motivi di coerenza delle caches presenti nei vari processori (2 bit)
- M - modified
 - La linea è disponibile in una sola cache ed è stata modificata senza essere stata riscritta sulla memoria. Alla linea si può accedere senza ciclo di bus
- E - exclusive
 - La linea è disponibile in una sola cache ma non è stata modificata. Una scrittura la porta in M
- S - shared
 - La linea potenzialmente è presente in più caches. Una scrittura sulla cache locale causa un WRITE-THROUGH e invalida la stessa linea nelle altre caches (che non sono più aggiornate, stato I)
- I - invalid
 - La linea non è disponibile nella cache e l'accesso in lettura causa una allocazione (LINE FILL - riempimento). Una scrittura provoca solo un WRITE THROUGH
- La cache del codice può essere solo S o E, non può essere modificata

Memoria virtuale

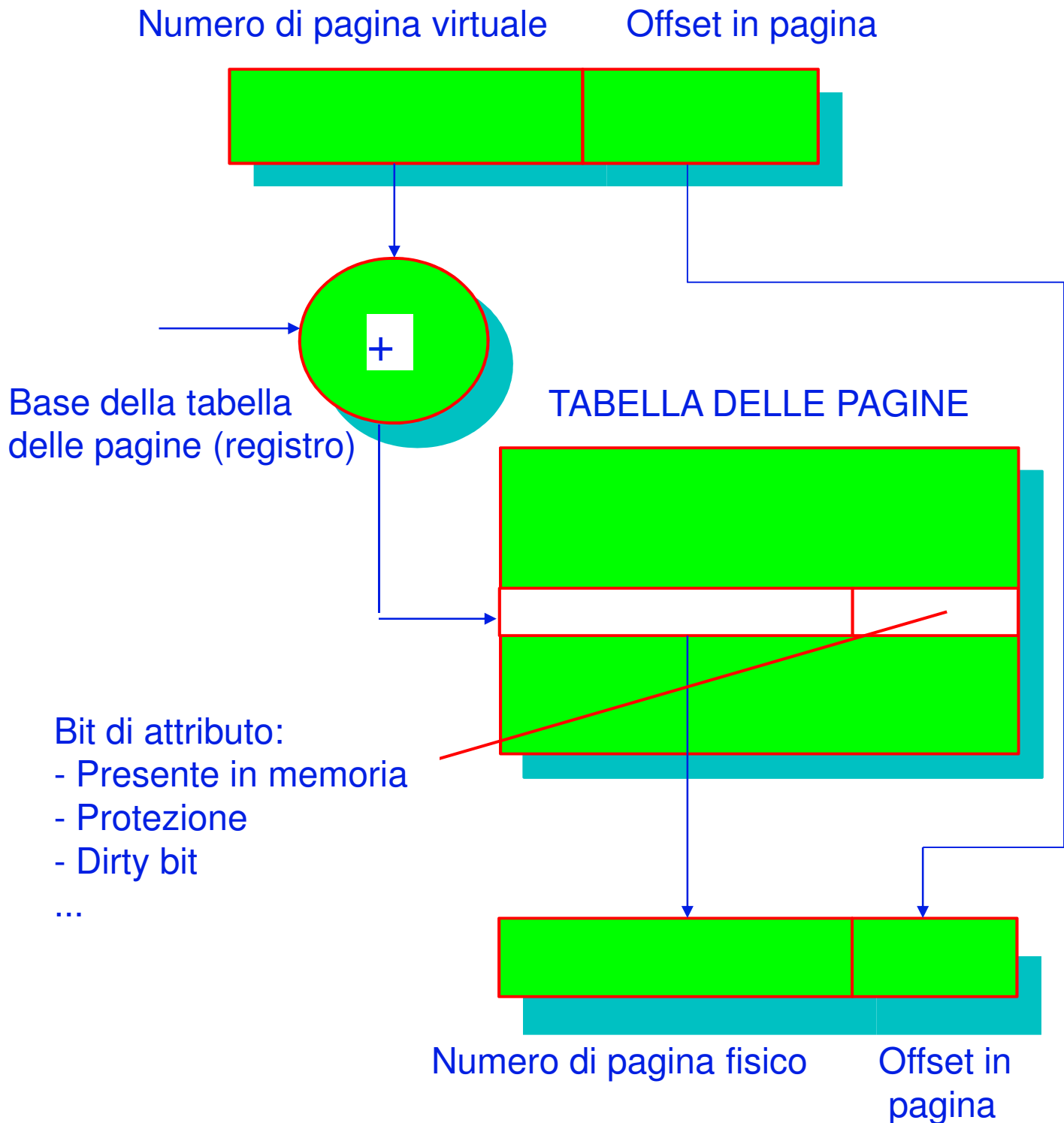
- Gerarchia fra memoria centrale e hard disk
- Motivazioni originali
 - Ampliare lo spazio di indirizzamento
 - Evitare gli overlays (trasferimenti di parti del programma sotto controllo del programmatore)
- Motivazioni odierne in aggiunta alle motivazioni originali
 - Rilocazione
 - Protezione
 - Condivisione
 - Uso sparso della memoria



PAGINAZIONE

- I blocchi (**pagine**) hanno dimensioni tipiche da 512 K a 4 M bytes
- Posizionamento delle pagine
 - **Totalmente associativo: la pagina può essere allocata ovunque in memoria centrale**
- Identificazione delle pagine
 - Traduzione dell'indirizzo (virtuale -> fisico)
 - Indiretto con tabelle delle pagine
 - Traduzione "cached" in un buffer di traslazione
- Rimpiazzamento delle pagine
 - Quasi sempre del tipo Least Recently Used (LRU), dividendo le pagine in due gruppi, di uso recente e remoto
- Strategie di scrittura
 - Write-back (usando un bit di pagina sporca, modifico in ram e poi trasferisco i dati aggiornati nel disco)

Traduzione dell'indirizzo



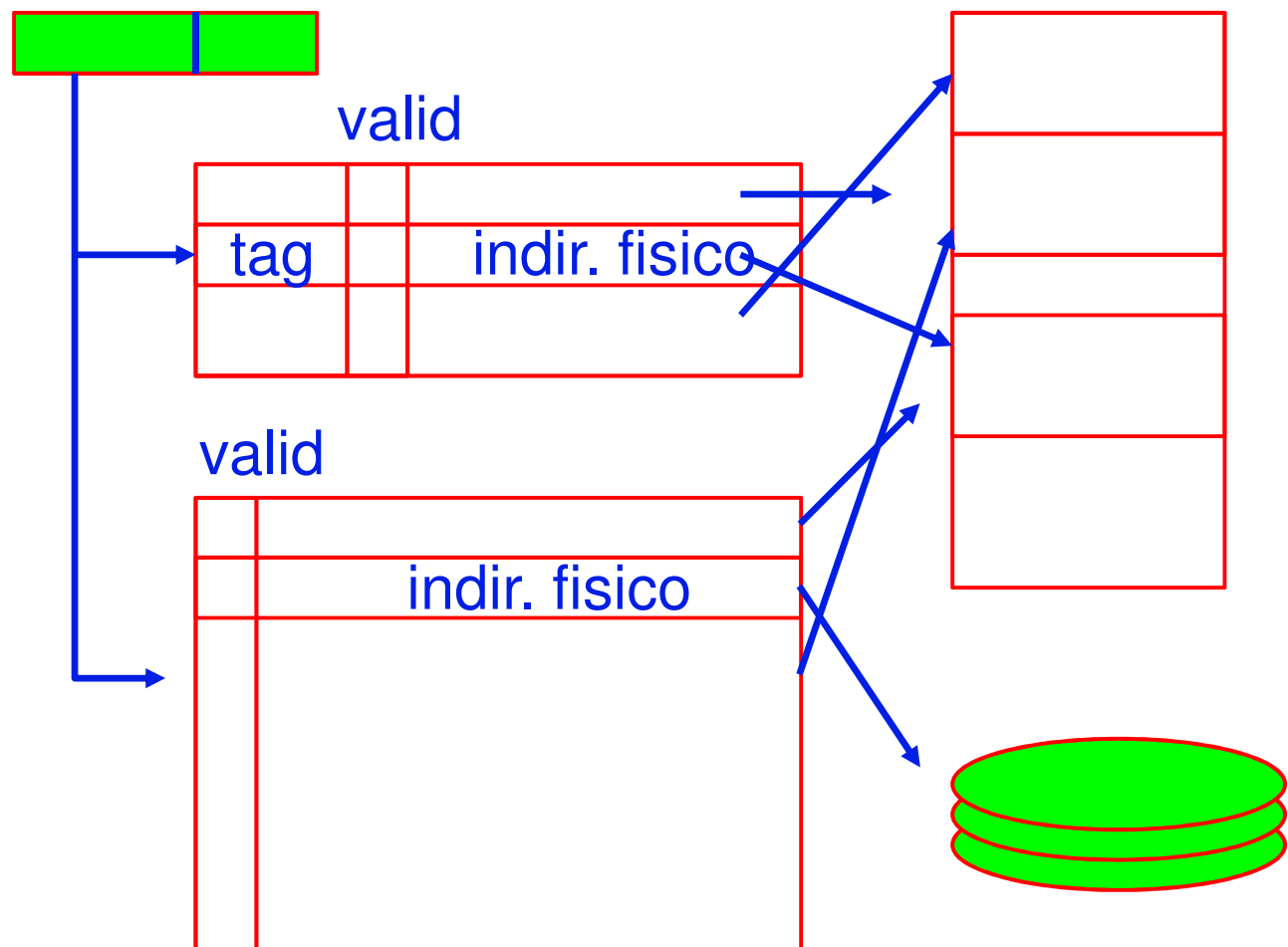
N.B.: Due accessi in memoria;
Spesso poi la tabella è a due livelli
-> tre accessi in memoria, TROPPO LENTO !!

Traduzione degli indirizzi

Per rendere più veloce l'accesso alle pagine (dato che anche la tabella delle pagine è in memoria):

TLB Translation lookaside buffer

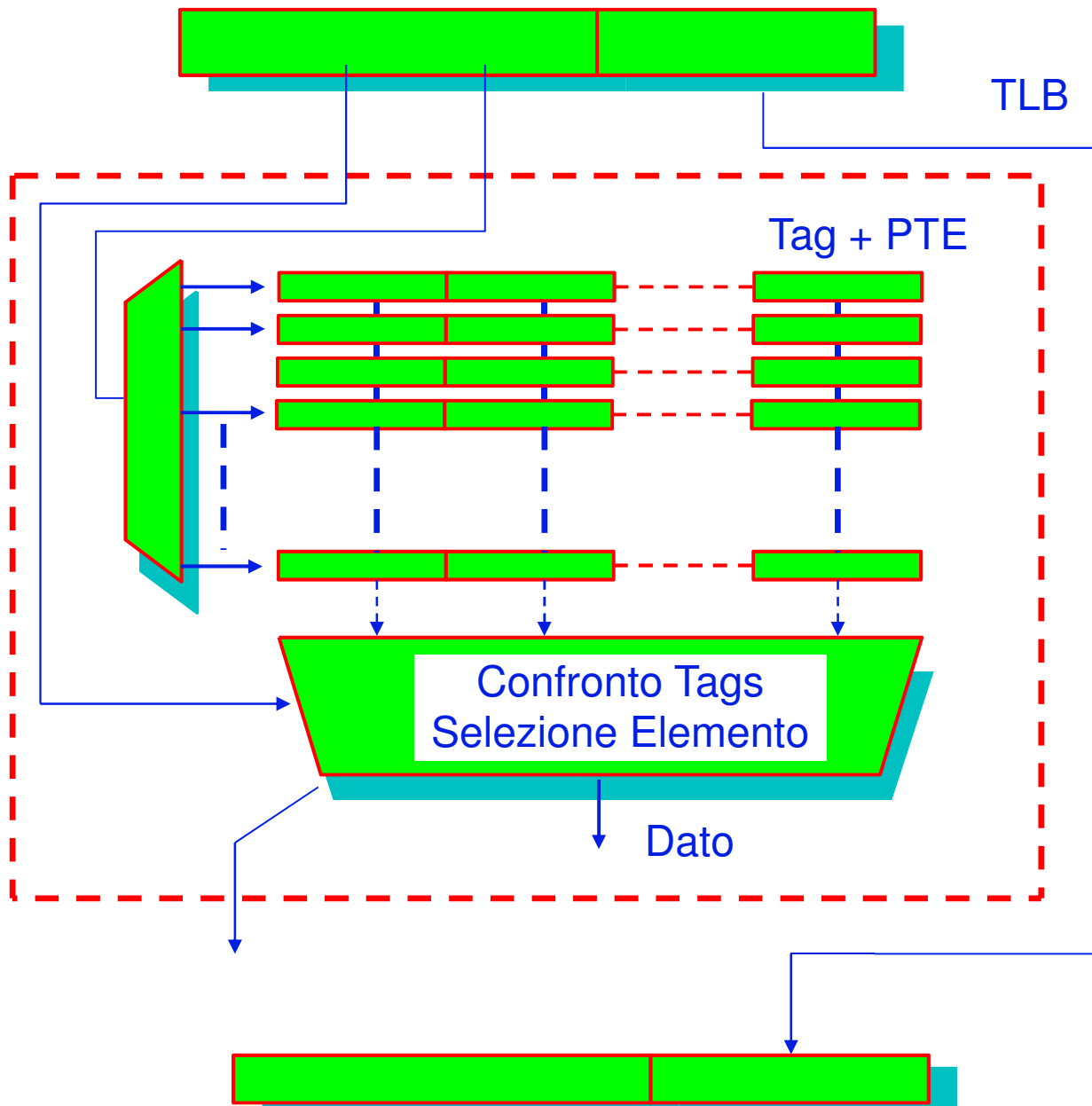
una cache dedicata per la tabella delle pagine



Traduzione dell'indirizzo

Numero di pagina virtuale

Offset in pagina



Translation lookaside buffer = TLB

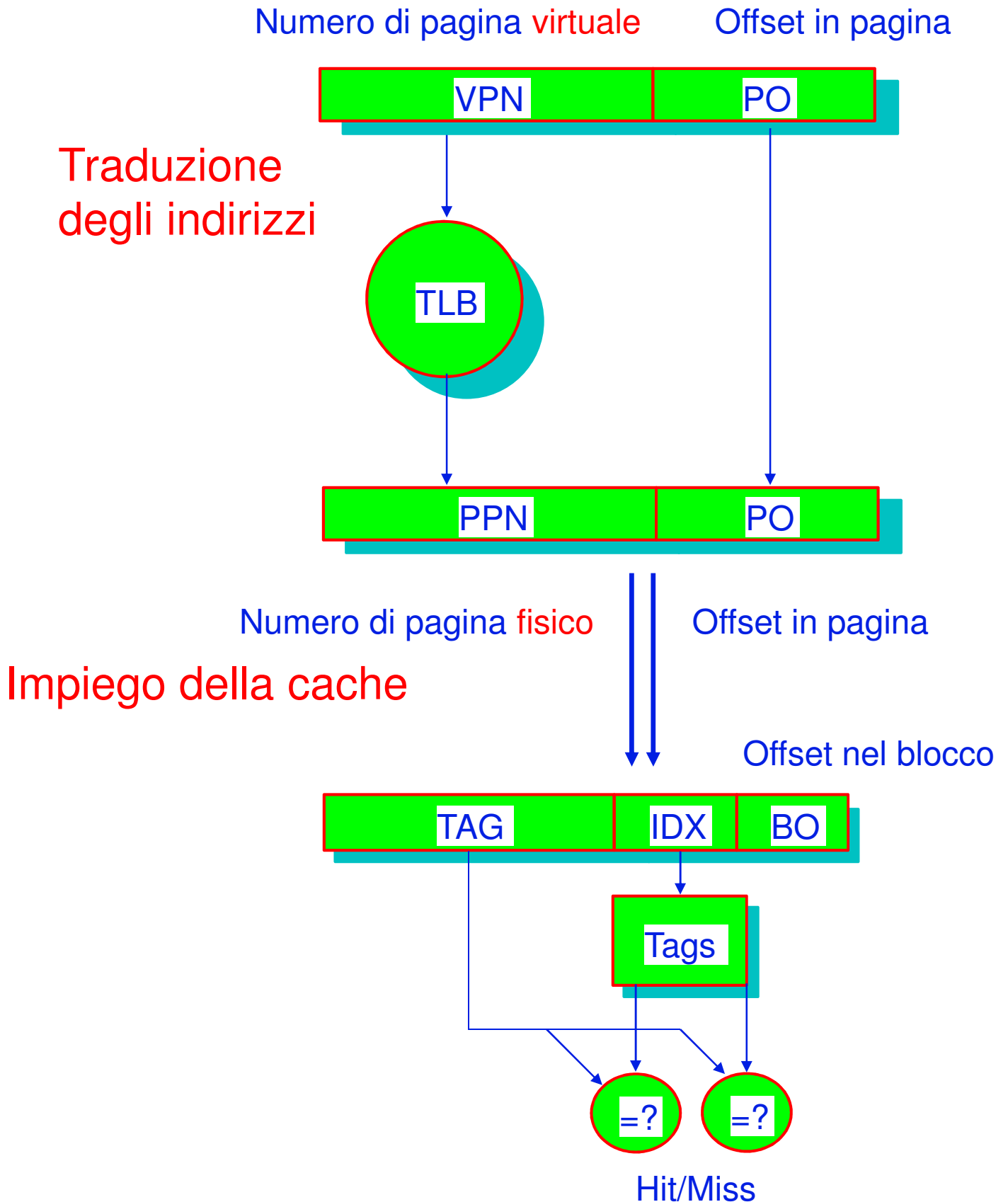
PTE=Page Table Entries

Numero di elementi 8 -> 1024

Memoria Virtuale e Cache

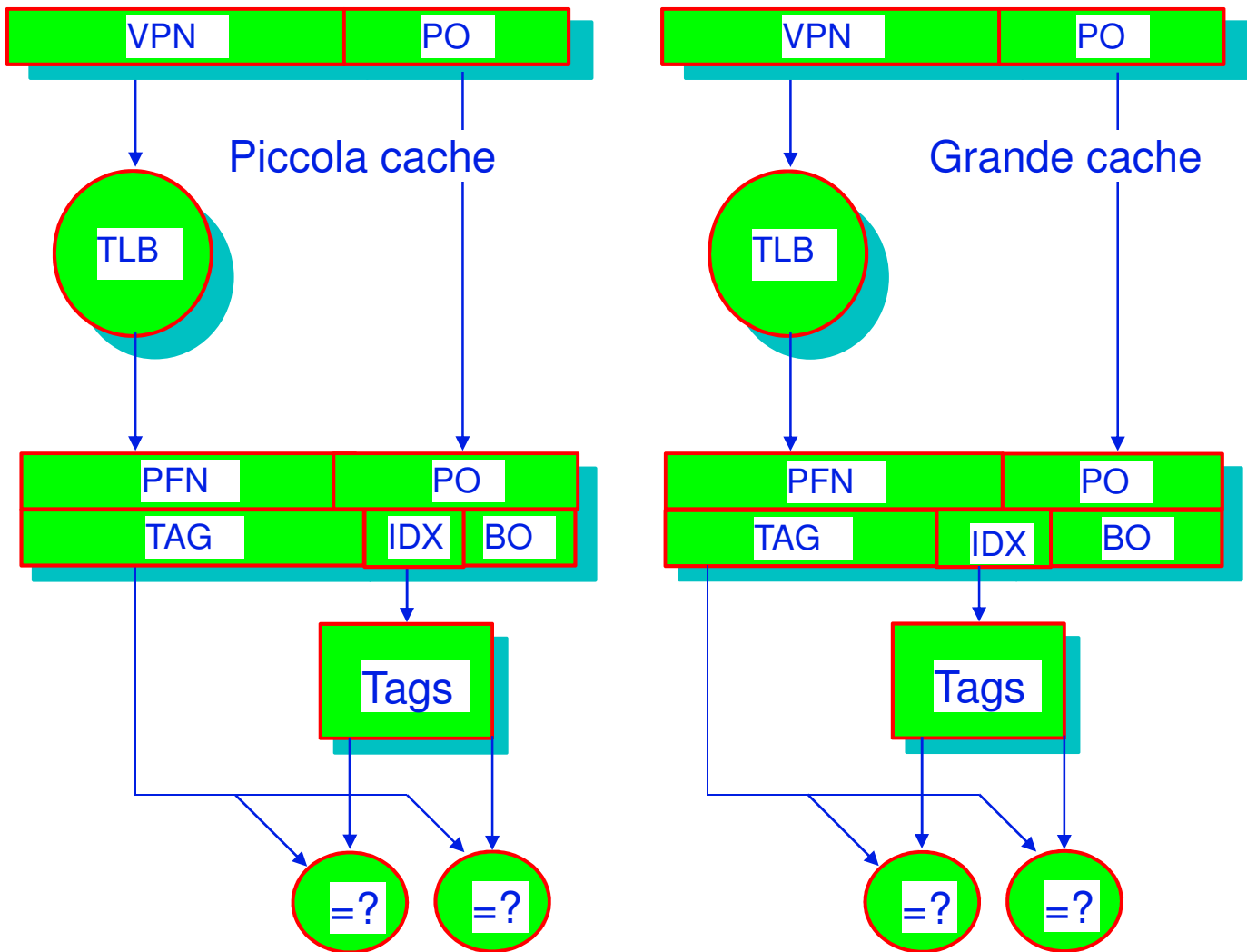
- ❖ Esiste interazione tra memoria virtuale e cache.
- ❖ La memoria virtuale introduce nel sistema un nuovo spazio di indirizzi (indirizzi virtuali) in aggiunta a quello fisico (indirizzi fisici)
- ❖ E' possibile mappare le cache sull'uno o sull'altro spazio.
- ❖ Esempio: processore RISC Intel i860 -> **cache mappate sullo spazio virtuale** (intuitivamente migliore dal punto di vista della località spazio-temporale, perché gli indirizzi generati dai programmi sono quelli a monte della traslazione virtuale-fisico)
- ❖ Svantaggi: gli indirizzi virtuali sono accessibili soltanto all'interno del chip. Gli indirizzi fisici sono invece accessibili sia dentro il chip (a valle della traslazione virtuale-fisico), sia fuori (sull'address bus) -> più adatti per realizzare cache interne ed esterne
- ❖ Esempio: Pentium, **cache mappate sullo spazio fisico**
- ❖ Svantaggio: occorre prima traslare l'indirizzo virtuale in fisico, per poi accedere alla cache-> lentezza
- ❖ Il caso più critico è il più frequente, quando la traslazione ha successo nel TLB

Caches e TLB



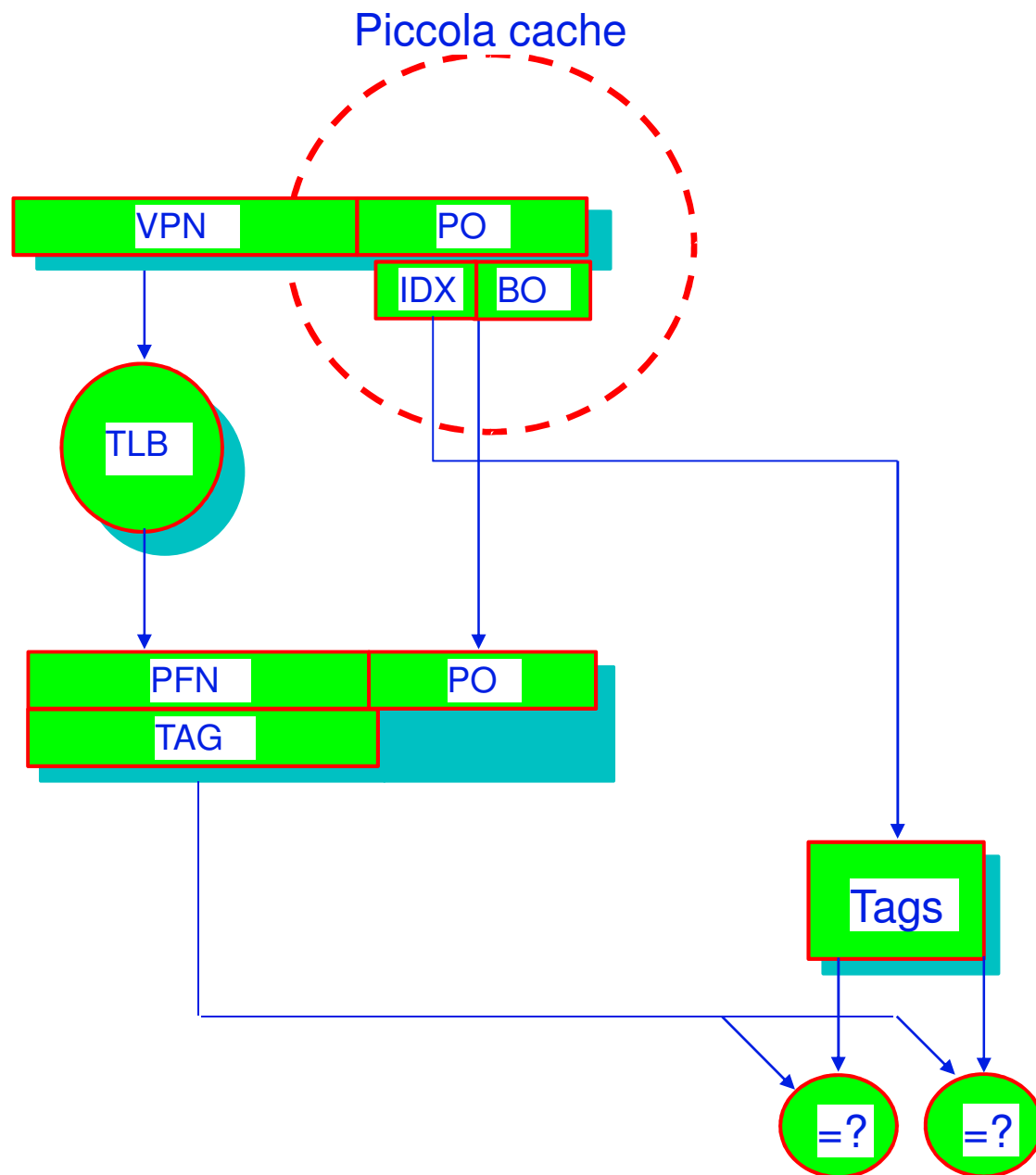
Caches e TLB

Traduzione dell'indirizzo prima dell'uso della cache



- ❖ Processo lento
- ❖ N.B.: si definisce "piccola" una cache in cui il numero di bit di Index + Offset è \leq di Page Offset

Accesso cache/TLB in parallelo



Per una cache "piccola", è possibile utilizzare immediatamente il campo Index per la selezione del set e preparare le tag per il confronto, mentre la traslazione procede in parallelo.