# Architettura degli Elaboratori I - B

## Le Memorie Cache

**Cash full/set-associative**

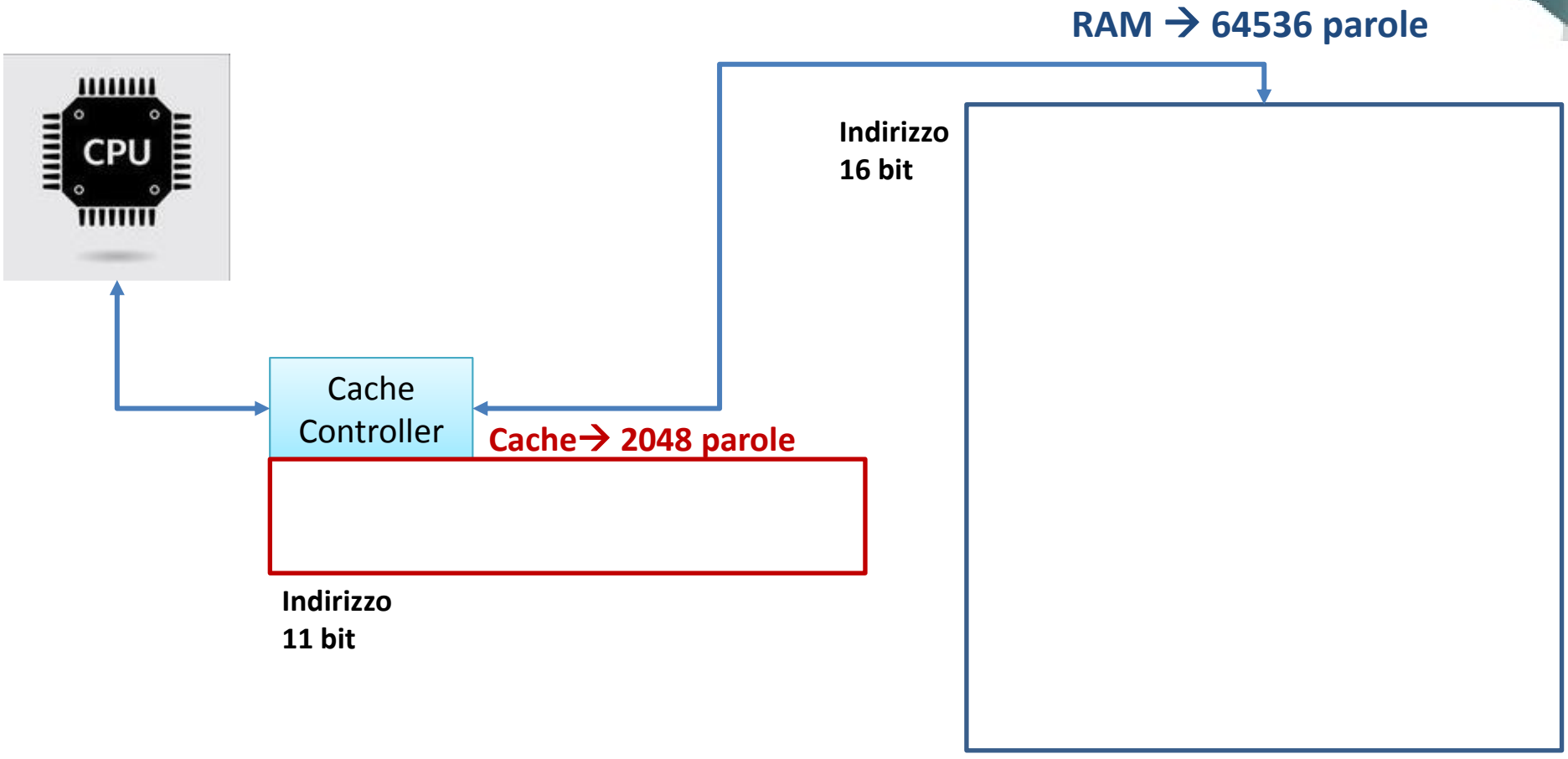
Daniel Riccio

Università di Napoli, Federico II

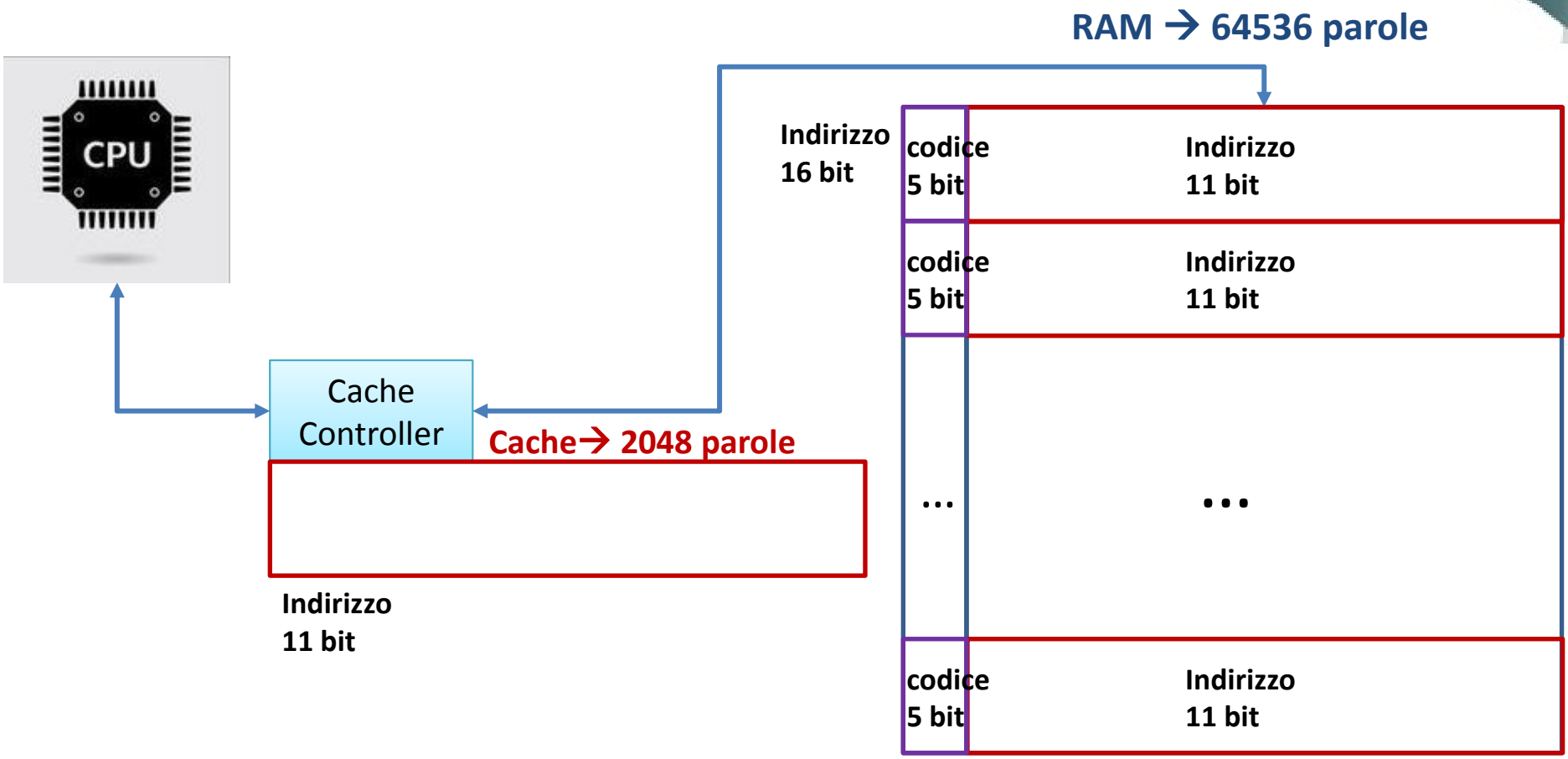
21 aprile 2018

**Fonti: lucidi degli anni precedenti**

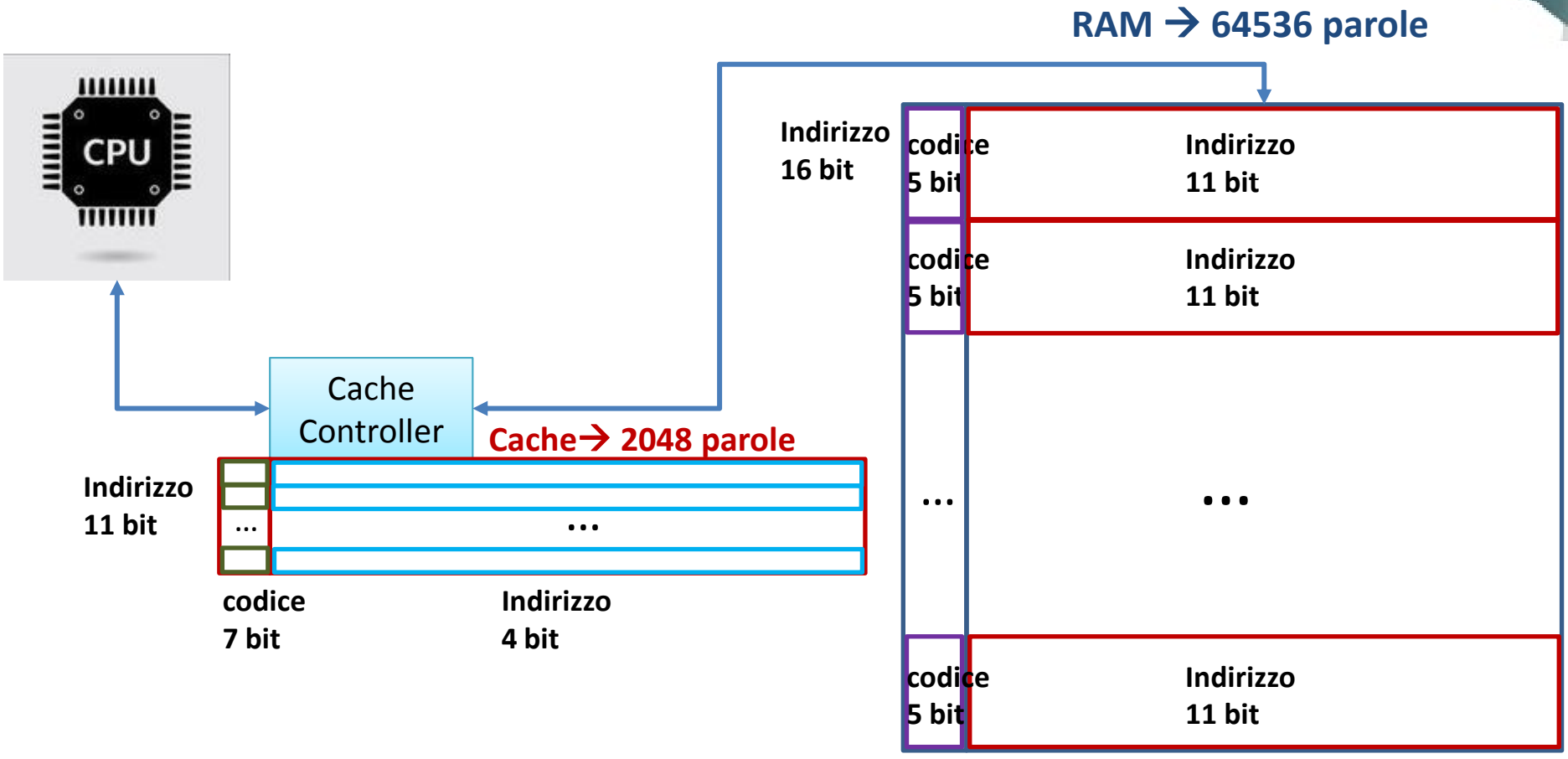
# Nelle puntate precedenti…



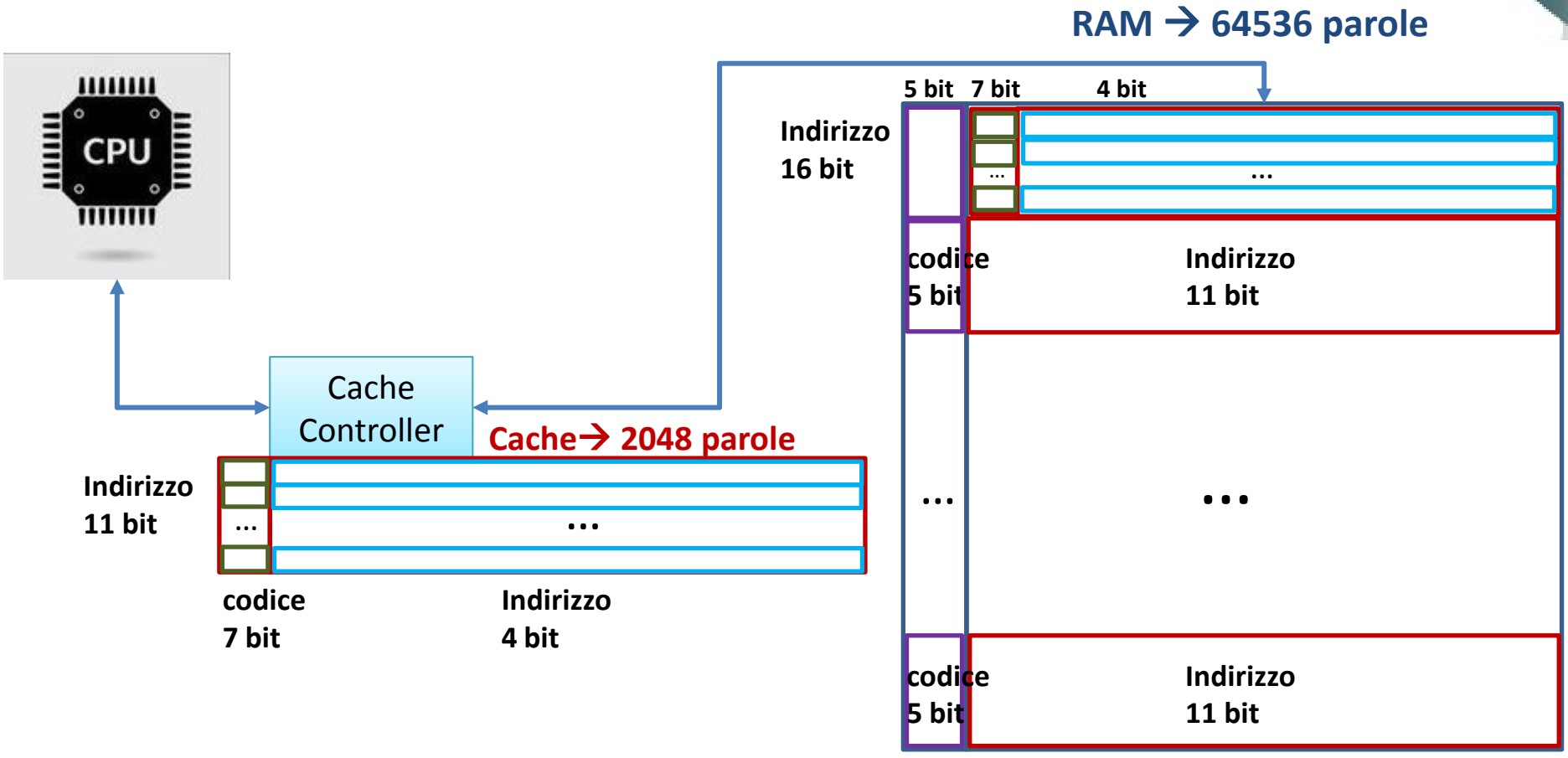
# Nelle puntate precedenti…



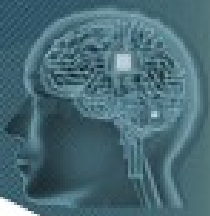
# Nelle puntate precedenti…



# Nelle puntate precedenti…



# Cache completamente associative ( 1 / 2 )

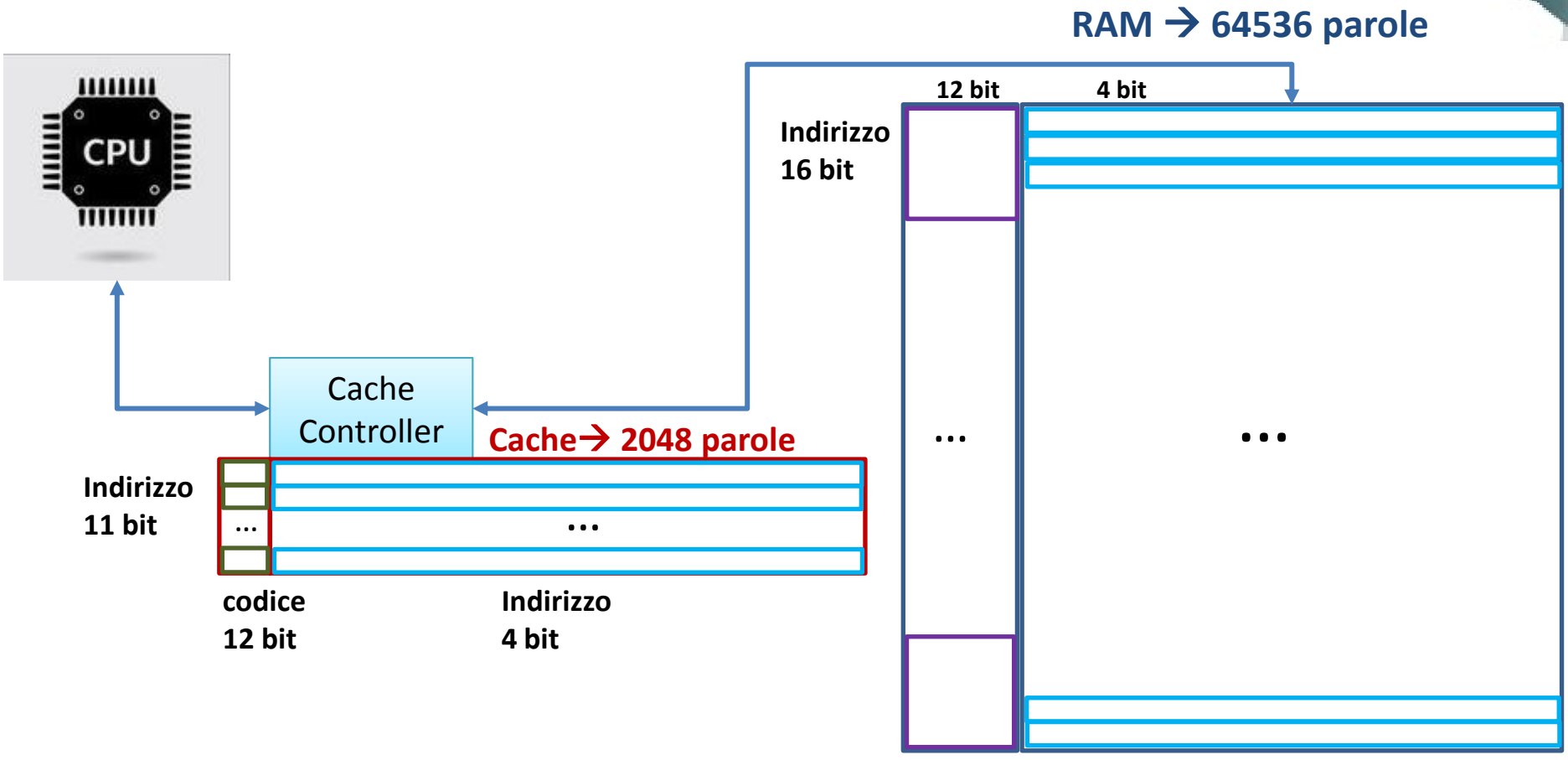


Esiste un modo di aggancio molto più flessibile, in cui **un blocco della memoria principale può essere posizionato in un qualsiasi punto della cache**.

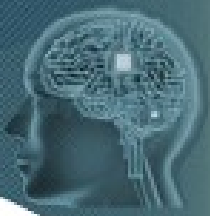
Se ipotizziamo, come in precedenza, **2048 locazioni nella cache divise in 128 blocchi da 16 locazioni**, sono necessari **12 bit di *etichetta*** per identificare uno dei 4096 blocchi della memoria principale, quando questo risiede nella cache.

I bit di etichetta di un indirizzo provenienti dalla CPU sono confrontati con l'etichetta di ogni blocco della memoria cache per vedere se il blocco cercato è presente.

# Nelle puntate precedenti…



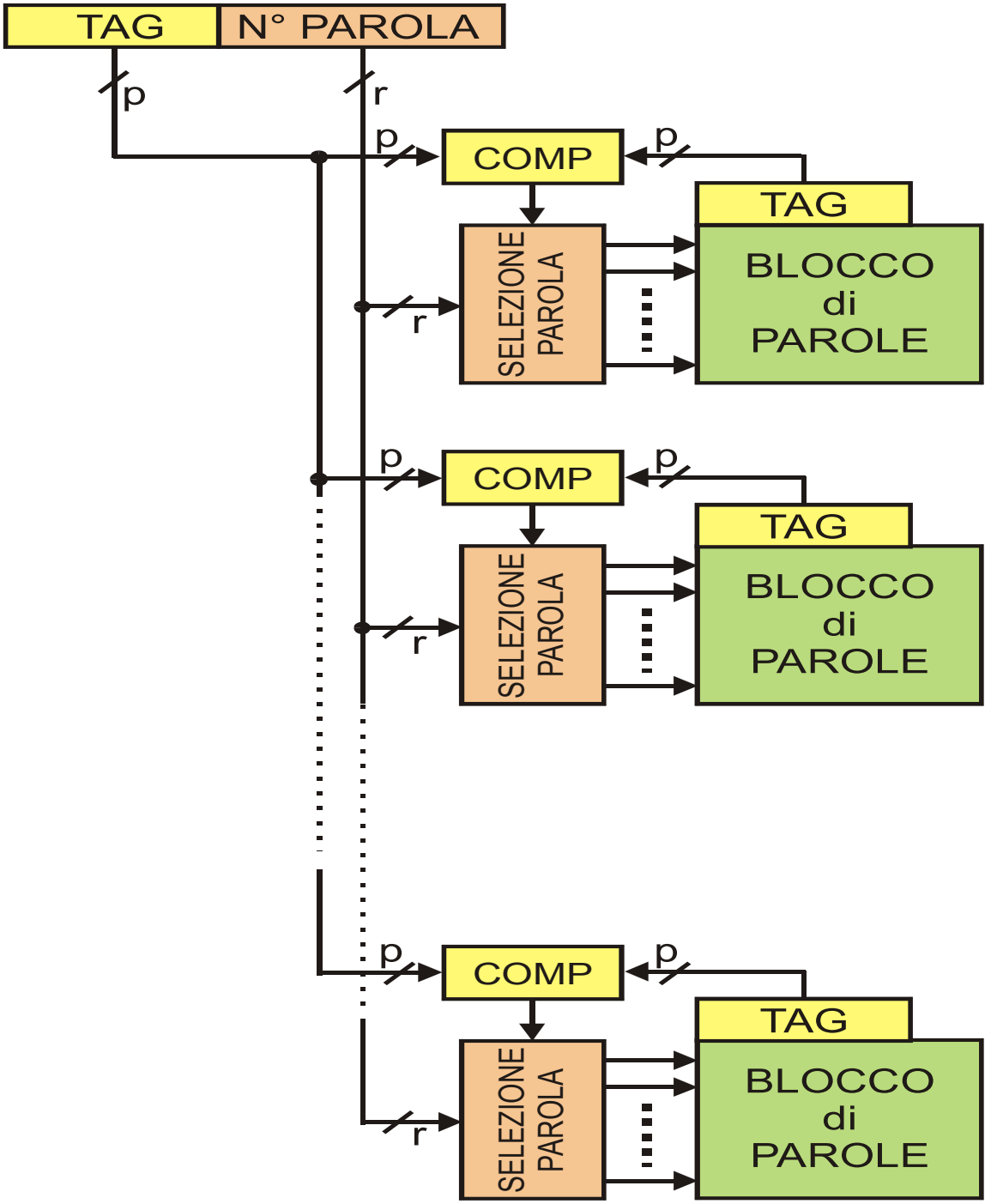
# Cache completamente associative ( 2 / 2 )



Quella descritta è la cosiddetta tecnica di ***indirizzamento completamente associativo***; essa consente la massima libertà nella scelta della locazione della cache in cui posizionare il blocco di memoria, quindi lo spazio della cache può essere utilizzato in modo molto più efficiente.

Un nuovo blocco che deve essere copiato nella cache ne fa uscire un altro già residente **solo se la cache è già piena**.
In tal caso, è però necessario, **utilizzare un algoritmo per selezionare il blocco che deve essere sostituito**.
Come discuteremo più avanti, ci sono numerosi ***algoritmi di sostituzione*** che si possono usare.

# Cache *completamente associative*



# La ricerca associativa è complessa

Il costo della ricerca di un blocco in una memoria cache completamente associativa è notevole, mentre in una cache ad indirizzamento diretto esso è praticamente nullo.

Nel caso di posizionamento completamente associativo **nell'esempio che stiamo sviluppando sarebbe necessario esaminare 128 etichette per determinare la disponibilità nella cache di un blocco della memoria centrale**.

Una operazione di questo tipo si chiama ***ricerca associativa*** o ***ricerca per contenuto***, in quanto corrisponde all'operazione che si fa quando si vuole sapere se un oggetto od un cognome sono presenti in un elenco.

# Le CAM (Content Addressable Memory)

Se si adoperasse per la ricerca una logica seriale, il tempo necessario sarebbe moltiplicato per il numero di confronti da effettuare e ciò porterebbe a tempi di accesso inammissibili.

**La ricerca viene effettuata con un algoritmo parallelo del tipo visto nello schema a blocchi precedente**.

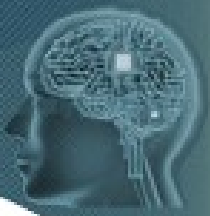
Le memorie in cui la ricerca viene effettuata fornendo in ingresso il contenuto cercato e ricevendo in risposta la posizione che esso occupa, se è presente, si dicono "***Content Addressable Memory***" (**CAM**).

# Le CAM (continua)

**Bisogna notare che in una cache completamente associativa la CAM non è l'intera memoria, ma solo la parte costituita dalla logica di selezione e dai registri etichetta.**
I blocchi sono invece piccole memorie tradizionali (ciascuna locazione viene selezionata attraverso il suo indirizzo) associate a ciascun registro d'etichetta.

Le CAM sono ormai disponibili anche come componenti di sistemi non di calcolo.

# Le cache set-associative



Esiste una combinazione delle due tecniche di indirizzamento. I blocchi della cache sono raggruppati in insiemi (***set***), ed il meccanismo di associazione è congegnato in modo che **ciascun blocco della memoria principale è associato ad un particolare insieme della cache ma può risiedere in una qualsiasi posizione all'interno di esso**.

**Avere più possibilità di posizionamento di un blocco in un insieme riduce la possibilità di dover attivare meccanismi di sostituzione di blocchi nella cache, anche quando essa non è completa, e riduce il numero di etichette tra cui effettuare la ricerca associativa**.

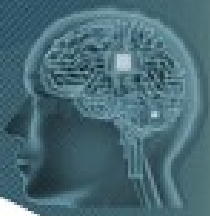
# Le cache set-associative (continua)

Con la solita cache da **2048 parole**, suddivisa in **128 blocchi** da **16 locazioni** ciascuna, se si sceglie di avere **4 blocchi per insieme**, si hanno **32 insiemi**.

Se la memoria principale ha i soliti **4096 blocchi** (64k locazioni) essi sono individuabili attraverso i **12 bit** più significativi di indirizzo, comuni alle 16 parole di ciascun blocco.

Ad ogni insieme della cache corrispondono, allora, **128 blocchi** della memoria principale ma solo fino ad **un massimo di 4** di essi possono trovare effettivamente posto contemporaneamente nella cache.

# L’algoritmo di posizionamento ( 1 / 2 )



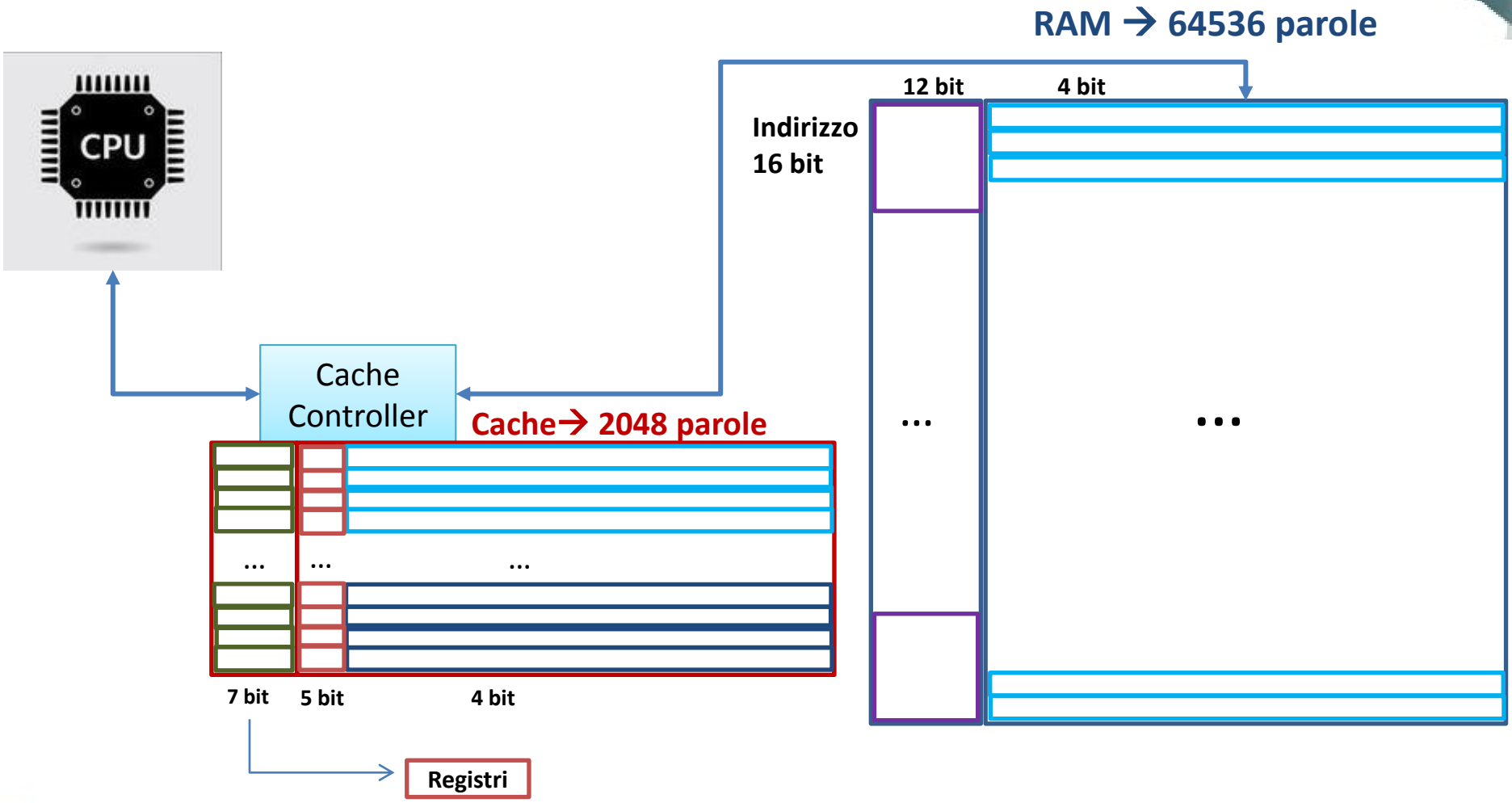
- I 128 blocchi della memoria 0, 32, 64, 96, 128, ..., 4064 puntano all'insieme 0 della cache, in una qualsiasi delle quattro posizioni.
- I blocchi 1, 33, 65, 97, 129, … 4065, sono associati all’insieme 1 della cache e così via per gli insiemi 2, 3 4 e fino a quello 31.
- Quando un blocco viene caricato nella cache, i 5 bit meno significativi dei 12 bit dell’indirizzo individuano l’insieme della cache dove esso deve essere posizionato; in un blocco vuoto.
- I successivi 7 bit più significativi dell’indirizzo vengono trascritti nel registro della cache associato al blocco, e le 16 parole di programma memorizzate nelle locazioni.

# L’algoritmo di posizionamento ( 2 / 2 )

1. Quando si effettua un accesso, i 4 bit meno significativi dell’indirizzo sono inizialmente ignorati, serviranno successivamente per individuare su quale delle 16 parole del blocco si deve operare.
2. I successivi 5 bit (***campo set***) indicano l’insieme dove cercare, mentre i 7 bit più significativi indicano il blocco cercato.
3. Questi bit debbono, quindi, essere comparati associativamente con le etichette dei quattro blocchi dell'insieme, per controllare se il blocco desiderato è presente o meno nell’insieme.

Questa ricerca associativa su solo 4 etichette è certamente molto più semplice e più veloce che in una memoria completamente associativa.

# Nelle puntate precedenti…



# Tabella degli indirizzi

| 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | BLOCCHI M. PRINC. | INSIEMI | ETICH. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 16 | 16 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 30 | 30 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 31 | 31 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 32 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 33 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 34 | 2 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 64 | 0 | 2 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 65 | 1 | 2 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 66 | 2 | 2 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2064 | 16 | 64 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 127 | 31 | 3 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 128 | 0 | 4 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 4072 | 16 | 127 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 4095 | 31 | 127 |
| 64 | 32 | 16 | 8 | 4 | 2 | 1 | 16 | 8 | 4 | 2 | 1 | | | |

# Qualche esempio numerico

Se, per esempio, si vuole trasferire il blocco 2323 della memoria principale, il suo indirizzo è: 1001000 10011 XXXX. Esso andrà nell'insieme 19 della cache con l'etichetta 72.

Viceversa, se si vuole accedere alla locazione 32034 ($810A_{hex}$) che ha indirizzo 1000000 10000 1010$_b$, si ignorano inizialmente i 4 bit meno significativi e si va ad esaminare il contenuto dei 4 blocchi dell'insieme 16 della cache, dove potremmo trovare, per esempio, il blocco 4072, il blocco 16 e come terzo blocco il 2064, cui appartiene la locazione cercata.

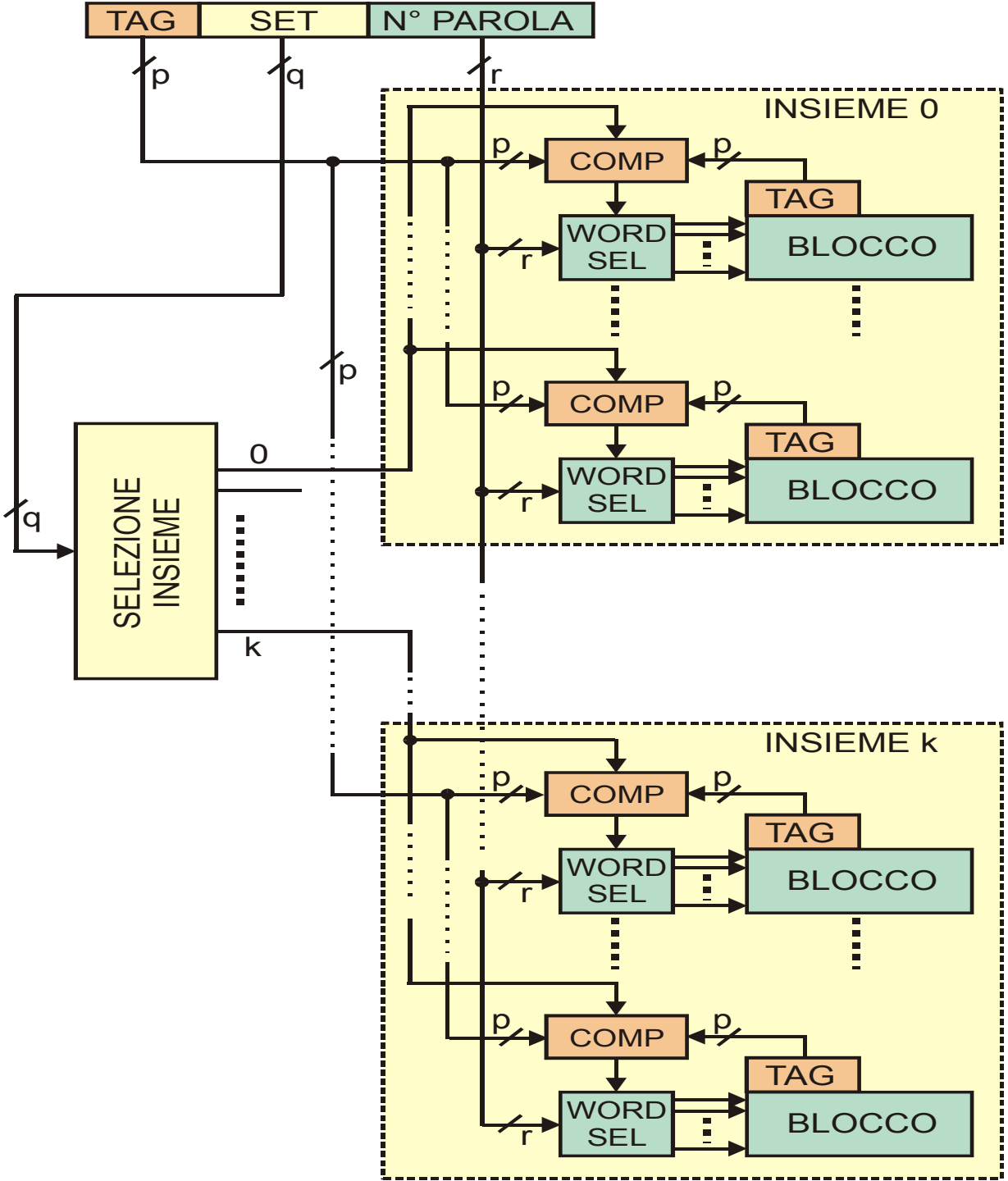
Il blocco 2064 sarà individuato attraverso una ricerca associativa sulle etichette dei 4 blocchi.

# Considerazioni sistemistiche

Il numero di blocchi per insieme costituisce il parametro che può essere selezionato per soddisfare i requisiti di un particolare calcolatore.

Con i numeri della memoria principale e della cache dell'esempio, si può scegliere un insieme di due blocchi, ed il campo set risulterà di 6 bit (64 insiemi).
Se si sceglie di avere otto blocchi per insieme, il campo set sarà di di 4 bit e cosi via.

# Lo schema a blocchi



# La cache set-associativa è un compromesso

- La condizione limite, di 128 blocchi per insieme, non richiede alcun bit per il campo set e corrisponde alla tecnica **completamente associativa**, con 12 bit di etichetta.

- L'altra condizione limite è costituita dalla presenza di un blocco per insieme, che corrisponde alla cache ad indirizzamento diretto.

# Il bit di validità

- Un ulteriore bit di controllo, chiamato ***bit di validità***, è necessario per ogni blocco.
- **Questo bit indica se il blocco contiene o meno dati validi**.
- Non deve però essere confuso con il ***bit di modifica***, menzionato in precedenza, il bit di modifica, che indica se il blocco è stato modificato o meno durante il periodo in cui è stato nella cache, serve soltanto nei sistemi che non fanno uso del metodo ***write-through***.

**I bit di validità dei blocchi che si trovano nella cache vengono tutti posti a 0 nell'istante iniziale di accensione del sistema e, in seguito, ogni volta che nella memoria principale vengono caricati programmi e dati nuovi dal disco**.

# Il Direct Memory Access - DMA

I trasferimenti dal disco alla memoria principale vengono effettuati mediante un meccanismo detto di DMA (Direct Memory Access). Questa sigla specifica che durante il trasferimento la CPU non interviene, perché è la logica di controllo del disco che gestisce le linee indirizzo e le linee dati, fornendo i segnali di controllo necessari.

Come vedremo successivamente, in una operazione di questo genere vengono trasferite grandi quantità di istruzioni e dati organizzati in entità dette "***pagine***" ciascuna delle quali di solito contiene migliaia di ***locazioni*** e quindi centinaia di ***blocchi***.
Normalmente, le pagine di programma e quelle di dati vanno nella memoria centrale senza coinvolgere la cache.

# Gestione del bit di validità

**Il bit di validità di un certo blocco della cache viene posto a "1" la prima volta che esso è chiamato a contenere istruzioni o dati trasferiti dalla memoria principale**; poi, ogni volta che le locazioni di un blocco della memoria principale vengono interessate da un trasferimento di nuove istruzioni o di nuovi dati dal disco, viene effettuato un controllo per determinare se qualcuno dei blocchi che stanno per essere sovrascritti fossero o meno presenti nella cache.

Se nella cahe c'è una copia del blocco, il suo bit di validità viene posto a "0"; in tal modo, si garantisce che nella cache non ci siano dati obsoleti.

Il bit di validità a "0" candida immediatamente il blocco ad essere sovrascritto.

# Svuotamento (flush) della cache

Una situazione dello stesso tipo si determina quando viene effettuato un trasferimento tramite DMA dalla memoria al disco e la cache utilizza un protocollo write-back.
**In questo caso, i dati nella memoria potrebbero non riflettere i cambiamenti che possono essere stati fatti sulla copia dei dati presenti nella cache.**

Una possibile soluzione a tale problema consiste nello ***svuotamento*** (***flush***) della cache forzando i dati con il bit di modifica attivo a essere ricopiati nella memoria principale prima che venga effettuato il trasferimento tramite DMA.

# “Coerenza” della cache

Il sistema operativo è in grado di fare tutto tutte le operazioni necessarie in modo molto efficiente, senza peggiorare significativamente le prestazioni, visto che le operazioni di ingresso/uscita di scrittura non sono molto frequenti.

Questa necessità di garantire che due diverse entità (i sottosistemi della CPU e del DMA nel caso presente) utilizzino la stessa copia dei dati viene chiamata problema della ***coerenza della cache***.

# Algoritmi di sostituzione ( 1 / 2 )

- In una cache ad indirizzamento diretto, la posizione di ogni blocco è predefinita, quindi non esiste alcuna strategia di sostituzione.

- Nelle memorie cache ***associative*** e ***set-associative***, esiste invece una certa flessibilità.

**Quando un nuovo blocco deve essere portato nella cache, e tutte le posizioni che potrebbe occupare contengono dati validi, il controllore della cache deve decidere quale blocco, tra quelli esistenti, soprascrivere**.

Questa è una decisione importante, perché la scelta potrebbe essere determinante per le prestazioni del sistema.

# Algoritmi di sostituzione ( 2 / 2 )

In generale, l'obiettivo è quello di **mantenere nella cache quei blocchi che hanno una maggior possibilità di essere nuovamente utilizzati nel prossimo futuro**. ma, non è facile prevedere i blocchi a cui si farà riferimento.

Una possibile strategia è di tener presente che la ***località dei riferimenti*** suggerisce che i blocchi a cui c'è stato accesso di recente hanno elavata probabilità di essere utilizzati nuovamente entro breve tempo.

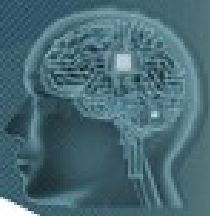
# Sostituzione con algoritmo LRU

Quando bisogna eliminare dalla cache un blocco, **è sensato soprascrivere quello a cui non si accede da più tempo**. Tale blocco prende il nome di blocco ***utilizzato meno di recente*** (***Least Recently Used***, **LRU**), e la tecnica si chiama ***algoritmo di sostituzione LRU***.

**Per utilizzare l'algoritmo LRU, il controllore della cache deve mantenere traccia di tutti gli accessi ai blocchi mentre l'elaborazione prosegue**.

Il blocco LRU di una memoria cache ***set-associativa*** con insiemi di quattro blocchi, può, ad esempio, essere indicato da un contatore di 2 bit associato a ciascun blocco, il cui contenuto sia opportu-namente gestito secondo un protocollo automatico

# Gestione del contatore di accessi



**Quando si ha un successo nell'accesso al blocco (*hit*):**

- il contatore di quel blocco viene posto a 0. I contatori con valori originariamente inferiori a quelli del blocco a cui si accede, vengono incrementati di uno, mentre tutti gli altri rimangono invariati.

**Quando, invece, l'accesso fallisce (*miss*)** si aprono due possibilità:

- l'insieme non è pieno, il contatore associato al nuovo blocco caricato dalla memoria principale viene posto a 0 e il valore di tutti gli altri contatori viene incrementato di l.
- l'insieme è pieno, si rimuove il blocco, il cui contatore ha valore 3, e si pone il nuovo blocco al suo posto, mettendo il contatore a 0. Gli altri tre contatori dell'insieme vengono incrementati di un'unità.

**E' facile verificare che i valori dei contatori dei blocchi occupati sono sempre diversi.**

# Alternative all'algoritmo LRU

L'algoritmo LRU è stato utilizzato ampiamente, con buone prestazioni, in numerosi schemi di accesso. Ma in alcune situazioni, per esempio nel caso di accessi sequenziali a un vettore di elementi che è leggermente troppo grande per poter stare tutto nella cache, le prestazioni sono molto scadenti

Le prestazioni dell'algoritmo LRU possono essere migliorate introducendo una piccola dose di casualità nella scelta del blocco da sostituire.

Sono stati proposti molti altri algoritmi di sostituzione più o meno complessi, che richiedono talvolta anche notevoli complicazioni hardware, ma l'algoritmo più semplice, e spesso anche piuttosto efficace, consiste nello **scegliere in modo completamente casuale il blocco da sostituire**.

# Considerazioni sulle prestazioni ( 1 / 2 )

Le prestazioni di un computer dipendono da due fattori:

1. velocità con cui possono essere portate nella CPU **le istruzioni macchina**;
2. velocità con cui queste possono essere eseguite.

**La velocità dell'esecuzione può essere aumentata agendo sulla struttura dei controlli della CPU, ma la velocità di accesso alle istruzione è almeno altrettanto importante.**

# Considerazioni sulle prestazioni ( 2 / 2 )

La gerarchia di memoria che vede la cache come elemento tampone tra CPU e memoria principale che, a sua volta, è un elemento di raccordo verso la memoria di massa (disco fisso) nasce dalla necessità di ottenere il miglior rapporto possibile prezzo / prestazioni.

Il principale scopo di questa gerarchia è quello di creare una memoria che la CPU possa vedere come caratterizzata da un breve tempo di accesso e da una grande capacità.

Ogni livello della gerarchia gioca un ruolo importante.

Anche la velocità e l'efficienza dei trasferimenti dei **dati** fra i vari livelli della gerarchia sono fattori di enorme importanza.