Calcolo prestazioni cache (1)

Consideriamo gcc:

- miss rate x istruzioni = 2%
- miss rate x dati = 4%
- frequenza di letture e scritture=36%

Consideriamo inoltre un sistema con:

- CPU: Clock=3Ghz, CPI^{ideale}=1 (in assenza di stalli per accesso a RAM / 100% HIT RATIO)
- RAM: DDR 266 MHZ (2x133Mhz, 2 parole x ciclo di clock)

Calcoliamo:

Tempo di accesso alla RAM= 1000ns/133 cicli/ns= 7,51 ns (per la prima parola)
Miss penalty= 7,51 ns/ 0,3 cicli di clock/ns=25 cicli di clock

Sia I il numero totale di istruzioni del programma:
cicli di stallo complessivi per i miss sulle istruzioni: 2% * 25* I = 0,5 * I
cicli di stallo complessivi per i miss sui dati: 4% * 25* 36% *I=0,36 * I
Cicli di stallo complessivi per miss: (0,25+0,18)*I=0,86
$CPI^{effettivo}$=CPI^{ideale}+cicli di stallo per miss=1,86

Differenza percentuale tra $CPI^{effettivo}$ e CPI^{ideale}

$(CPI^{effettivo} - CPI^{ideale})/= CPI^{ideale}$ = 86%

E ora: Overclocking! (*)

Portiamo il clock della CPU a 6Ghz (…e muniamoci di una buon dissipatore!)

Dato che il tempo di accesso alla RAM rimane costante, la penalità di Miss raddoppia
Miss penalty= 2 * Miss Penalty con clock dimezzato =50 cicli di clock

Per cui i cicli di stallo complessivi per miss raddoppiano:0,86*2= 1,72
$CPI^{effettivo}$=CPI^{ideale}+cicli di stallo per miss=2,72

Tempo di esecuzione di un programma con I istruzioni su un processore di cui è noto il tempo di clock:

I*CPI*TEMPO DI CLOCK

Tempo di esecuzione a 3GHZ / Tempo di esecuzione a 6 GHZ =
= (I*CPI A 3 GHZ*TEMPO DI CLOCK A 3GHZ) / (I*CPI A 6 GHZ*TEMPO DI CLOCK A 6GHZ) =
= (I*CPI A 3 GHZ*TEMPO DI CLOCK A 3GHZ) / (I*CPI A 6 GHZ*1/2 * TEMPO DI CLOCK A 3GHZ) =
= CPI A 3 GHZ/ (CPI A 6 GHZ* ½) = 1,86 / (2,72 * 0,5) = 66%

Quindi raddoppiando la velocità del processore,
il tempo di esecuzione non è sceso al 50% bensì al 66%

(*) Si consiglia “caldamente” di NON ripetere l’esperimento a casa sui vostri PC…

Diminuire la penalità di miss con caches multi livello

- **Aggiunta di un secondo livelo di cache:**
  - **spesso la cache primaria è nello stesso chip del processore**
  - **si introduce un altro cache basato su tecnologia SRAM tra la cache primaria e la memoria principale (DRAM)**
  - **La penalità di miss è ridotta sensibilmente se il dato è trovato sulla cache di secondo livello**

- **Example:**
  - **CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access**
  - **Adding 2nd level cache with 20ns access time decreases miss rate to 2%**

- **Utilizzo di multilevel caches:**
  - **Si tenta di ottimizzare il tempo di hit sulla cache di primo livello**
  - **Si tenta di ottimizzare il tasso di miss sulla chache di secondo livello.**

Prestazioni delle cache a più livelli

Dato un Processore a 2 GHZ con CPI^{ideale}=1,0 (se tutti gli accessi sono risolti nella L1 cache).

Tempo di accesso a RAM=50ns.

Miss-rate L1 cache=5%

Calcoliamo Miss penalty= 50 ns/ 0,5 ns/cicli di clock = 100 cicli di clock

$CPI^{effettivo}$=CPI^{ideale}+cicli di stallo per istruzione dovuti ad accesso a RAM

$CPI^{effettivo}$=1+5%*100=**6**

Supponiamo di introdurre una cache di secondo livello con :

- Tempo di accesso= 5 ns ed
- ampia abbastanza da avere: Miss-rate L2 cache=2%

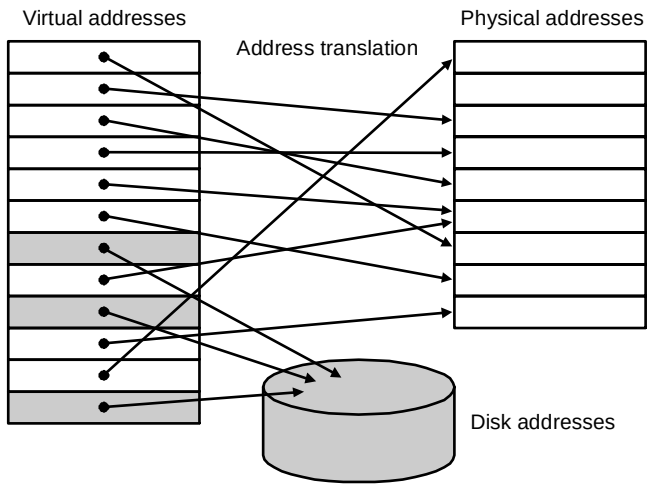
Miss penalty L2 cache= 5 ns/0,5 ns/cicli di clock = 10 cicli di clock

$CPI^{effettivo}$=CPI^{ideale}+cicli di stallo per istruzione dovuti ad accesso a RAM+cicli di stallo per istruzione dovuti ad accesso a L2 cache

$CPI^{effettivo}$=1+5%*10+2%*100=**3,5**

Memoria Virtuale

- **La memoria principale può agire come una cache per i dispositivi di memorizzazione secondari (dischi)**



- **Vantaggi:**
  - **illusione di disporre di un quantitativo superiore di mem. fisica**
  - **Rilocazione dei programmi**
  - **Protezione dei singoli processi (spazi di indirizzamento separati)**

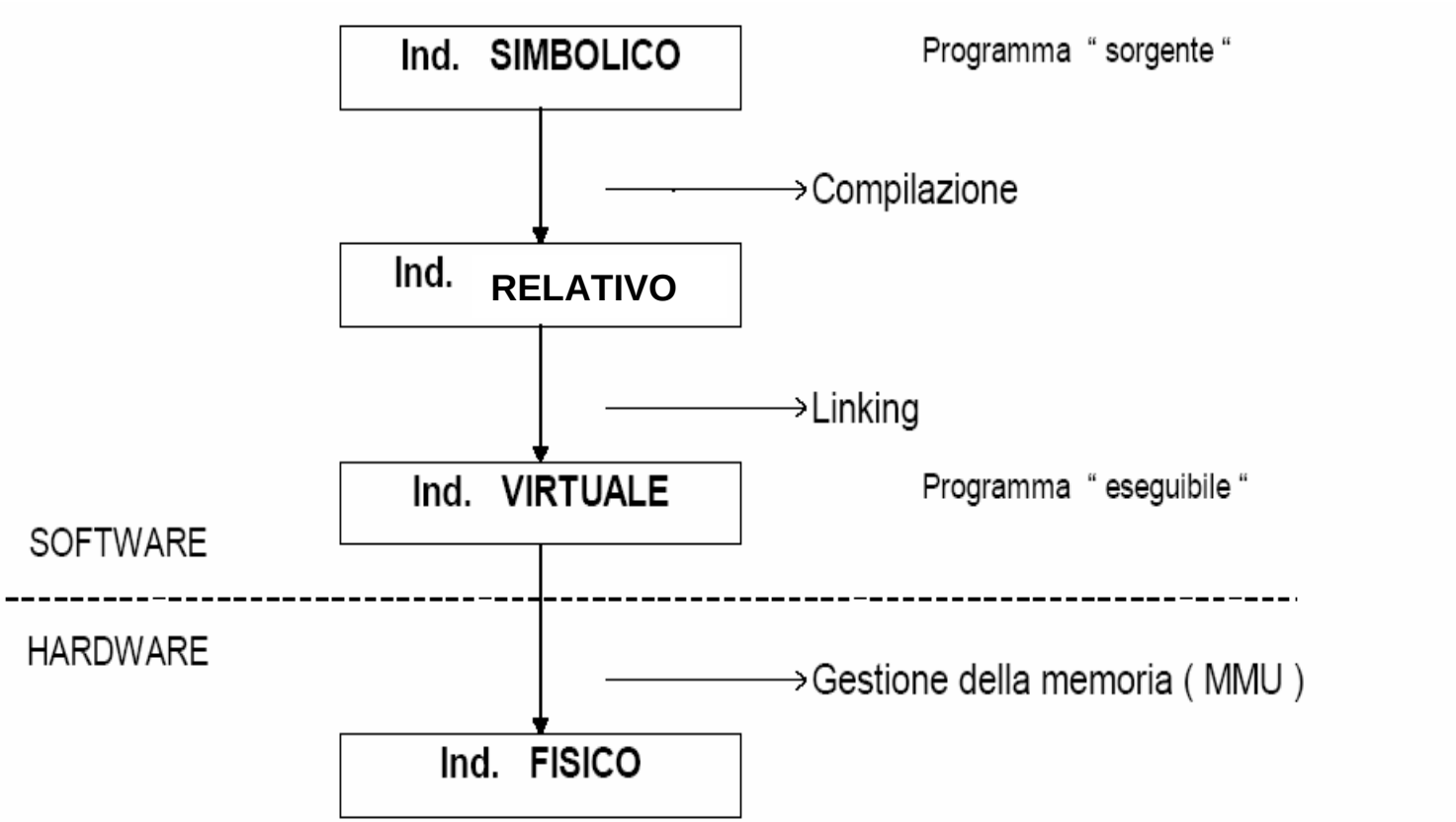
Rilocazione o Allocazione Dinamica

In un sistema multi-utente o multi-applicazione:

- **Il programmatore o il compilatore specificano gli indirizzi relativi all'interno di singoli blocchi di programma**
- **Il sistema operativo definisce in fase di esecuzione, le locazioni nelle quali il singolo blocco viene posto:**
  - **Il trucco consiste nel porre in corrispondenza gli indirizzi virtuali usati da un programma con i diversi indirizzi fisici prima che questi vengano usati per indirizzare la memoria.**

I 4 stadi della metamorfosi di un programma…

… e delle relative modalità di indirizzamento:



Paginazione vs Segmentazione

Gli attuali sistemi di memoria virtuale trattano le aree di memorie del programma (codice e dati) come un insieme di blocchi di dimensioni fisse dette pagine:

- **rilocare un programma corrisponde a trovare un numero sufficiente di pagine disponibili in memoria principale (page frames).**

In passato era diffuso uno schema con blocchi di dimensioni variabili, detti segmenti:

- **insieme di bytes contigui e posti in relazione logica, definiti dal compilatore a partire dai moduli o blocchi dei programmi di alto livello.**
- **Il segmento è il blocco elementare che viene trasferito fra le gerarchie.**

Segmenti

VANTAGGI

- La struttura del segmento corrisponde alla naturale suddivisione dei dati e dei programmi.
- E' agevole gestire meccanismi di separazione e di protezione per i dati dei singoli utenti.

SVANTAGGI

I segmenti hanno dimensioni fisiche diverse, quindi:

- allocazione complessa
- frammentazione (esterna)
- necessità di ricompattazione della memoria.

Il processo di allocazione gestisce la disponibilità della memoria fisica e cerca di ottimizzare:

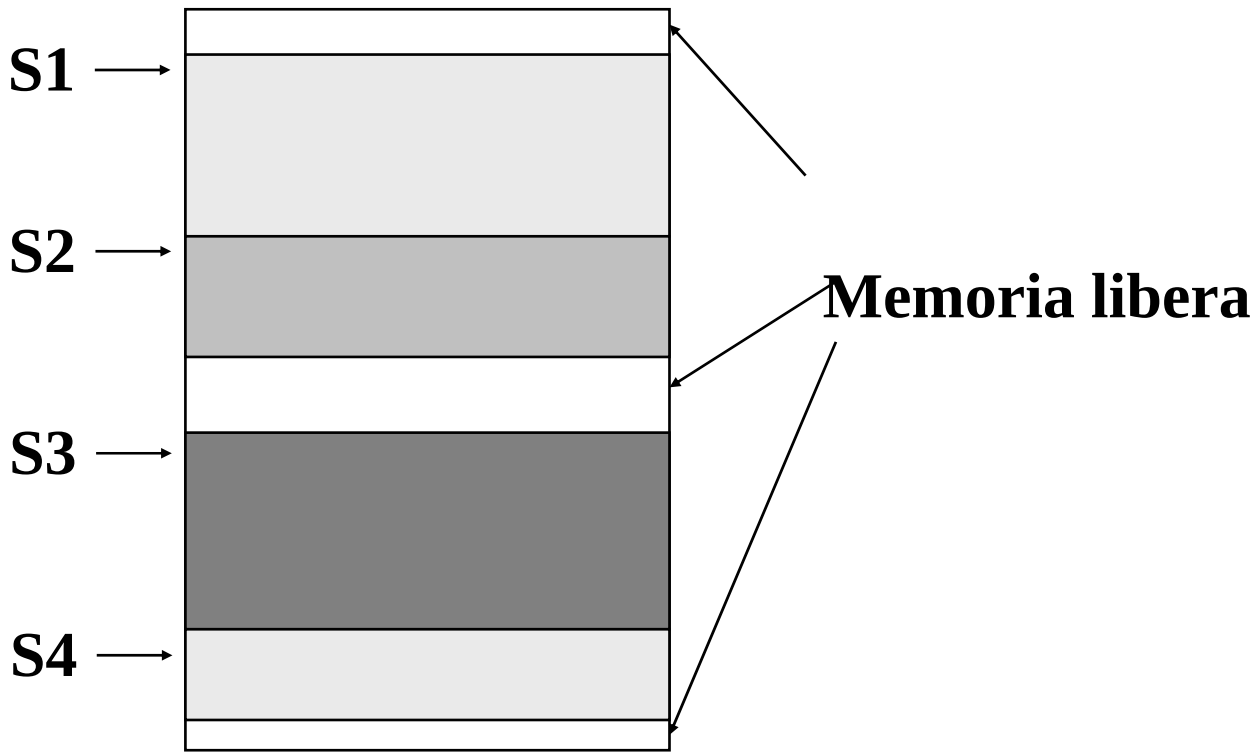
- tasso di successo (HA),
- tempo di accesso (TA),
- utilizzo dello spazio (u).

CAUSE PER IL MANCATO UTILIZZO DELLO SPAZIO

- Regioni vuote. Frammenti di memoria non utilizzati.
- Regioni occupate dal sistema di gestione.
- Regioni occupate da informazioni che non vengono utilizzate.

FRAMMENTAZIONE ESTERNA

La frammentazione esterna riduce l'utilizzazione della memoria quando risulta necessario disporre i dati dei segmenti in celle di memoria contigue. Lo stesso problema si verificava sui sistemi di memoria secondaria (dischi) quando i ogni file doveva essere memorizzato in segmenti adiacenti.

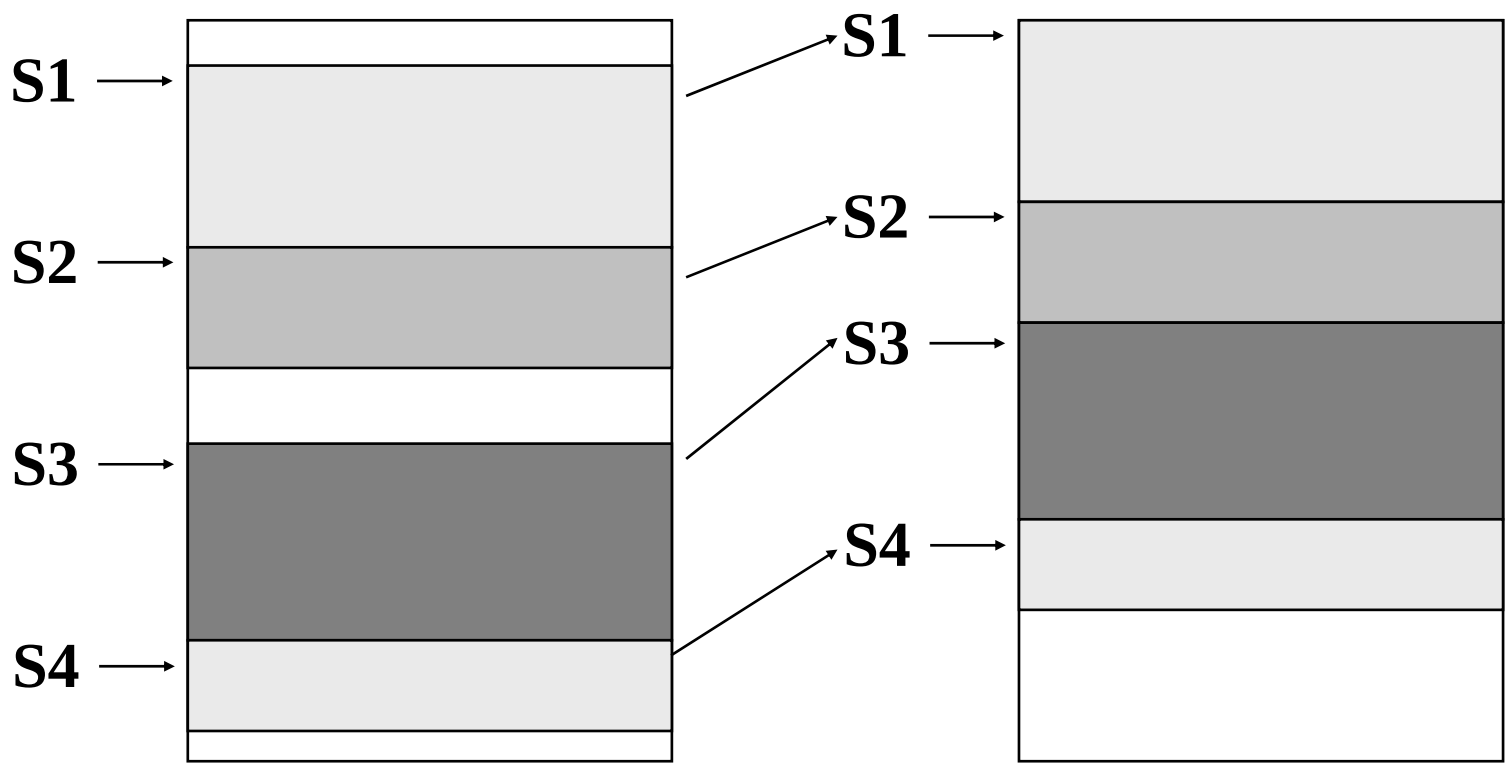


RICOMPATTAZIONE

La memoria non può essere sfruttata completamente.

Tra le aree occupate rimangono zone di piccola dimensione nelle quali non possono essere allocati segmenti.

Per migliorare l'utilizzazione della memoria: ricompattazione!



Gli albori della memoria virtuale: OVERLAY

In principio se un programma eccedeva le dimensione della memoria fisica, il programmatore doveva direttamente gestire lo spostamento dei dati dalla memoria secondaria alla memoria principale.

1) **Identificazione delle parti mutualmente esclusive che a run-time venivano sovrascritte (overlay=sovrapposte).**
2) **Tali parti venivano caricate e scaricate sotto il diretto controllo del programma.**
3) **Occorreva accertarsi che un il programma non cercasse mai di accedere ad un overlay non caricato e che gli overlay non eccedessero mai la dimensione della memoria fisica**

Contro:

Notevole carico sui programmatori (meccanismi di gestione molto complessi + necessità di disporre di implementazioni efficienti per non penalizzare le performance del sistema)

PAGINAZIONE

Per trasferire dati tra memoria principale e secondaria non si usano blocchi logici (segmenti) ma blocchi di dimensione fissa o PAGINE.

La Tabella Pagine contiene informazioni relative a:

- Indirizzo fisico.
- Presenza in memoria.
- Se modificato da quando in memoria.
- Diritti di accesso.

L'uso delle pagine al posto dei segmenti come blocco elementare per il trasferimento dei dati nella gerarchia di memoria formata dalle memorie primaria e secondaria consente di semplificare le procedure di trasferimento e di allocazione.

VANTAGGI

- Allocazione più semplice (dimensione fissa del blocco).
- Assenza di frammentazione esterna.

SVANTAGGI

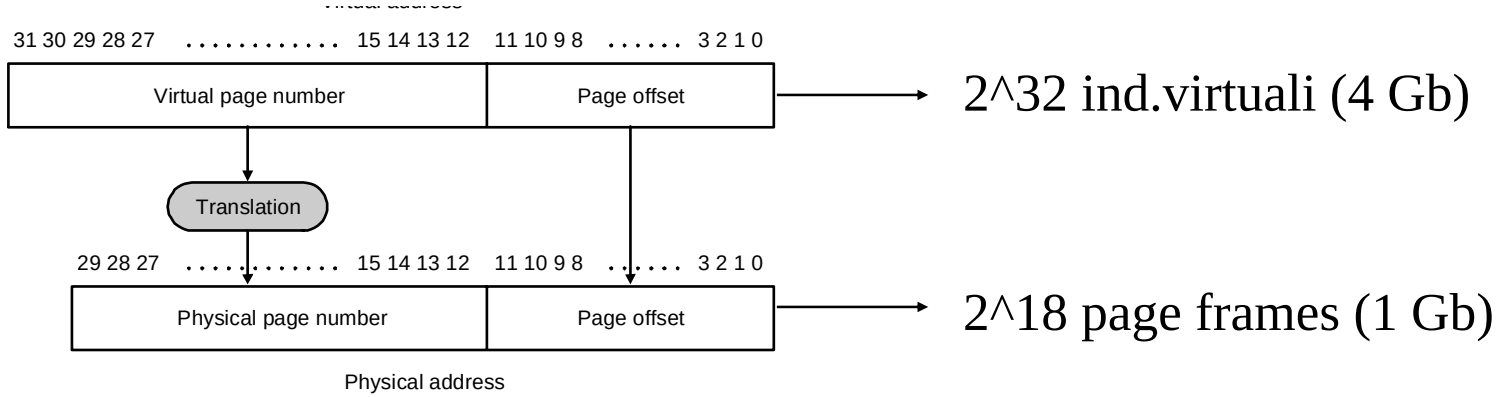
- Maggiore difficoltà di gestione delle proprietà.
- Frammentazione interna - La dimensione dei dati da trasferire non è un multiplo intero delle pagine.

Page faults

Se i dati non sono in memoria principale, occorre recuperarli dal disco.

Enorme penalità di miss (milioni di cicli), il cui costo è dovuto in gran parte al prelievo della prima parola:

- **le pagine sono scelte sufficientemente grandi da (e.g., 4KB)**
- **ridurre i page faults è importante:**
  - **collocazione completamente associativa**
  - **Implementazione di politiche di gestione complesse (LRU)**
- **È possibile gestire i faults via software (overhead trascurabile rispetto a disco): supporto per algoritmi di collocazione più intelligenti.**
- **Usare il write-through è troppo costoso così si usa il writeback (scrittura di una pagina per volta + dirty bit)**



Page Faults

Il sistema operativo tiene traccia di quali processi e quali indirizzi virtuali usino una data pagina fisica.

Se le pagine fisiche sono esaurite, in caso di page fault, occorre sostituire qualche pagina (swap out) in base ad un criterio che minimizzi la possibilità di dover accedere a quella pagina di lì a breve: LRU.

Approssimazione di LRU tramite l'uso di un bit di riferimento settato ad uno ogni volta che si accede alla pagina e periodicamente resettato.

Quando la politica di gestione della gerarchia non è efficiente si verificano troppi page faults, si ha il fenomeno di *Thrashing:*

La CPU spende il proprio tempo ad allocare e disallocare aree di memoria senza più svolgere lavoro utile.

↓

CAUSA:

cattivo dimensionamento della memoria rispetto ai dati utilizzati dal programma che opera sul sistema.

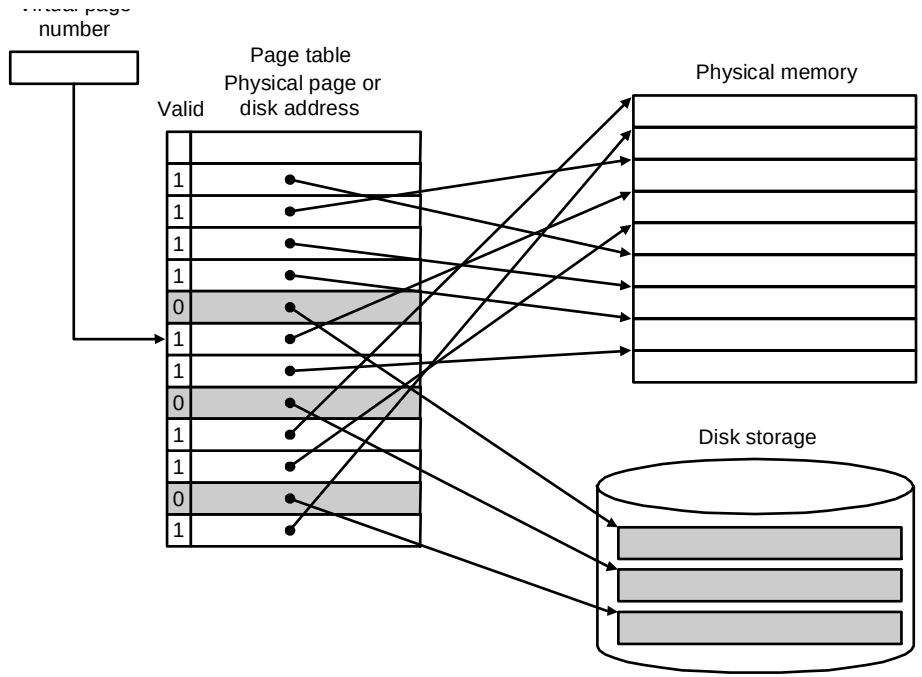
↓

Page Tables

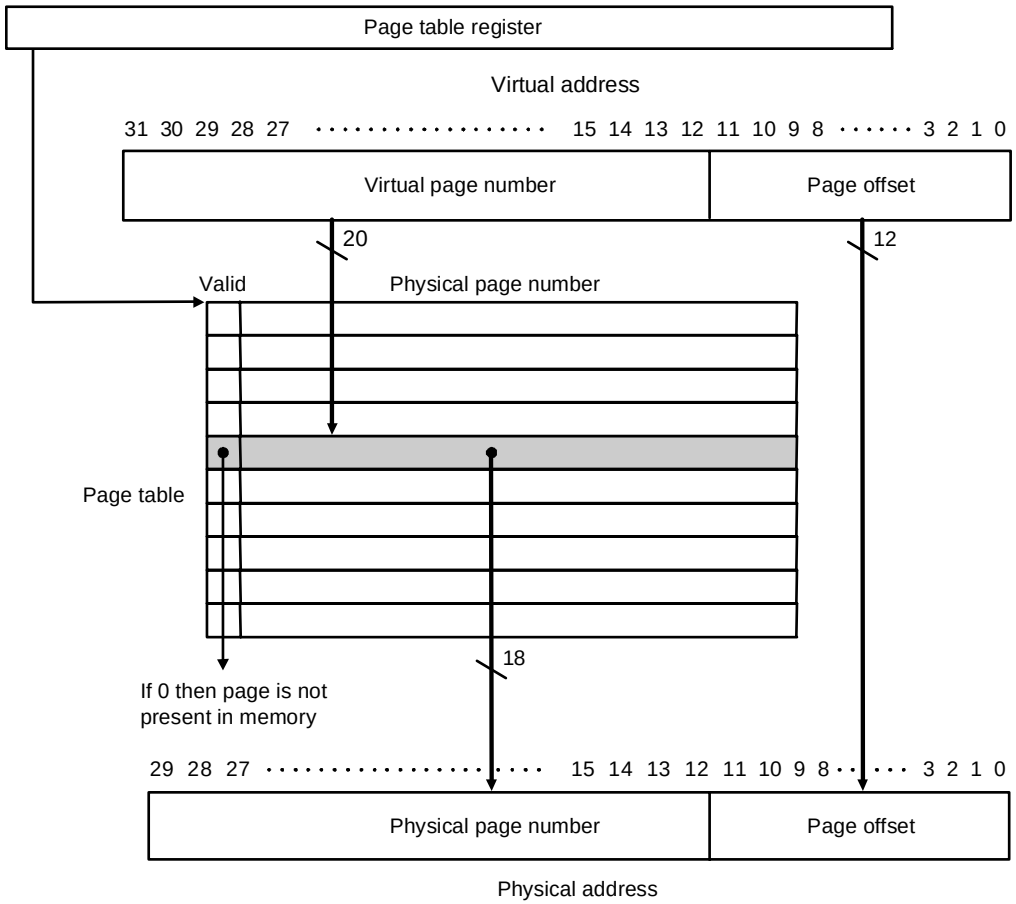
Collocazione completamente associativa: ritrovamento pagine difficoltoso!

Tabella completa che indicizza con il numero di pagina dall'indirizzo virtuale e contiene l'indirizzo del corrispondente page frame.

- La tabella delle pagine risiede in memoria principale
- Ciascun programma ha la propria page table (protezione)
- Bit di validità per determinare page fault
- L'hw mette a disposizione il *page table register* per indicare la posizione della page table in memoria (context switch rapidi)
- No tag=> 1 elemento per ogni pagina virtuale.



Page Tables



#Virtual Pages=$2^{32}/2^{12}=2^{20}$

Bits per entry: 32 per allineamento

Dimensione Page Table :

2^{20}*4bytes=4MB **per processo**!

Soluzioni:

- **Registro di limite** che punta alla fine della P.T. =>Crescita uni-direzionale.
- Doppia tabella (heap & stack) + **doppio registro di limite** (uno per segmento)
- **Organizzazioni a più livelli**:
  - Primo livello tiene informazione per grossi blocchi (segmenti di 64-256 pagine). Solo se qualche pagina del segmento è allocata punta alla relativa page table.
  - Bit + significativi dell'indirizzo per primo livello.
  - Buona gestione spazio indirizzamento sparso
- **Paginazione della page table**….

Velocizzare la traduzione degli indirizzi

- **Intuizione: Sfruttare l'alta probabilità di località spaziale e temporale per gli indirizzi nella stessa pagina!**
- **Una cache per le traduzioni degli indirizzi: TLB, translation lookaside buffer: ciascun blocco del TLB contiene l'indirizzo del page frame (calcolato a partire dal page table register e dall'offset presente nella page table in ram) associato ad una data pagina virtuale**

Valori tipici per il TLB:

•Dimensione del TLB: 32 - 4096 elementi

•Dimensione del blocco: 1-2 elementi della page table.

•Tempo di HIT: 0,5 – 1 ciclo

•Penalità di Miss: 10-30 cicli

•Frequenza di Miss: 0,01% - 1%

Velocizzare la traduzione degli indirizzi

- **Intuizione: Sfruttare l'alta probabilità di località spaziale e temporale per gli indirizzi nella stessa pagina!**
- **Una cache per le traduzioni degli indirizzi: TLB, translation lookaside buffer.**

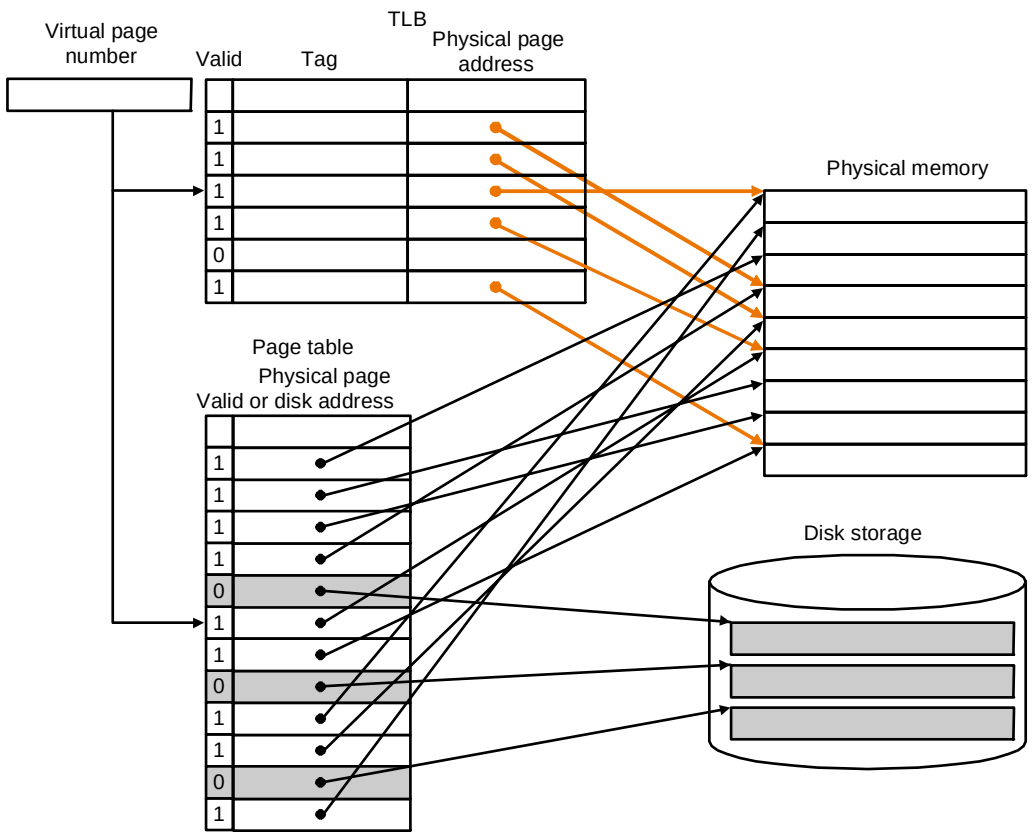
- **Contenuto del TLB:**

Campo TAG, contenente una porzione del numero di pagina virtuale

Indirizzo fisico della pagina

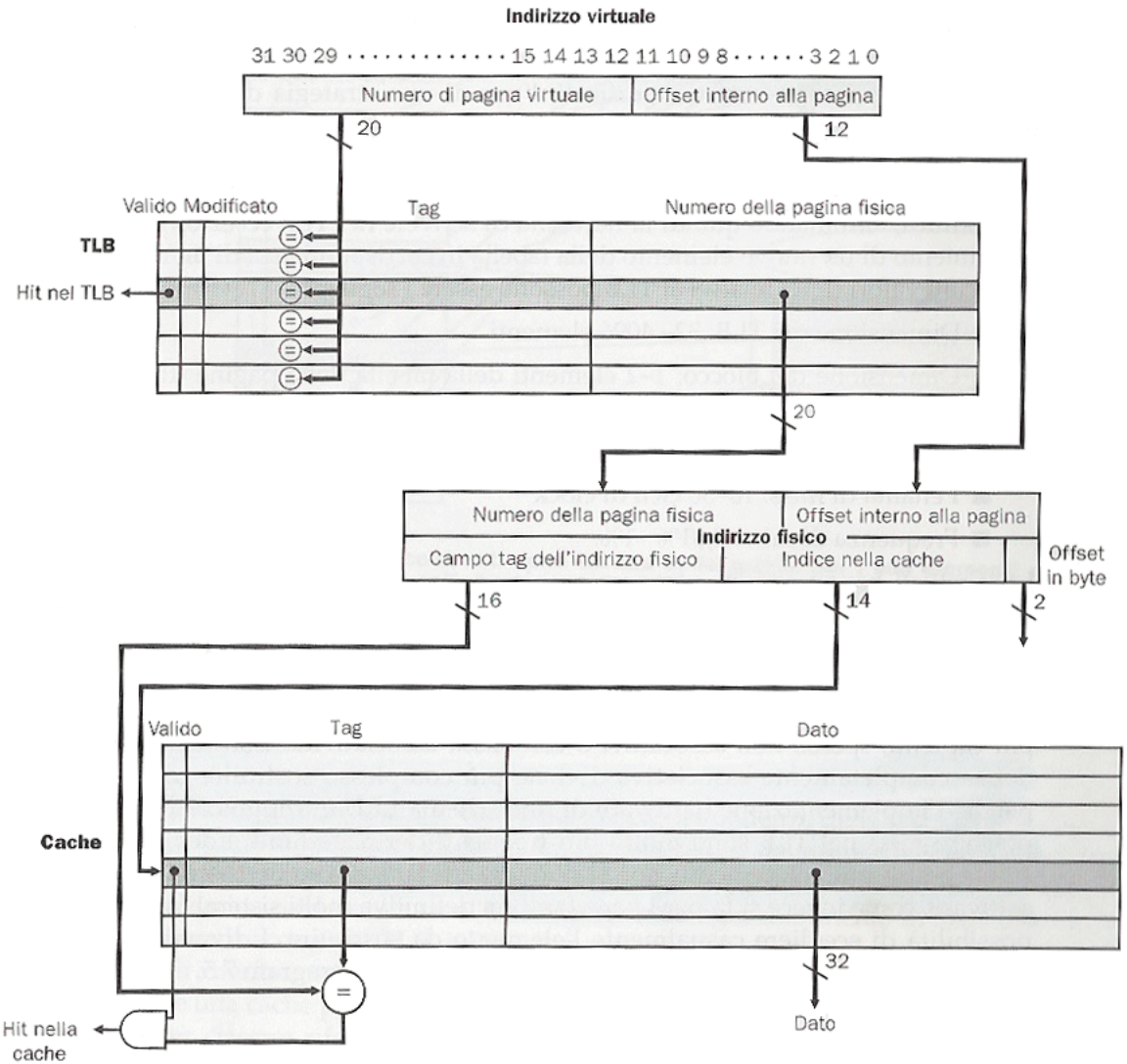
Bit di validità, riferimento e di modifica.

- **Un miss nel TLB può corrispondere a due situazioni:**
  1) dato presente in ram
  2) page fault
- La sostituzione di un elemento del TLB (in seguito ad un miss) genera la ricopiatura in RAM dei bit di stato

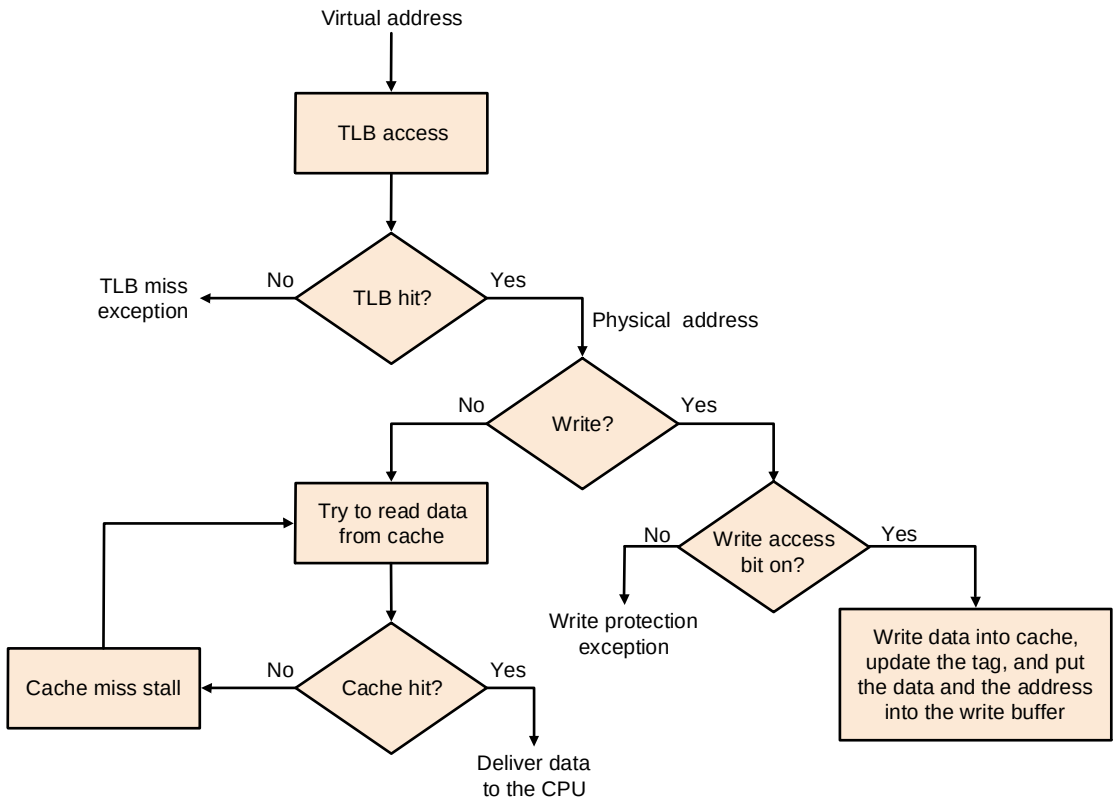


TLB e CACHE nella DECStation3100

- **Cache a corrispondenza diretta**
- **TLB completamente associativo.**
- **Cache indicizzata tramite indirizzi fisici, per cui, prima di accedervi, è sempre necessario utilizzare il TLB.**
- **Speciale gruppo di istruzioni usato dal SO per caricare linee del TLB**
- **Elemento da rimpiazzare nel TLB scelto casualmente.**



TLBs and caches



Cache indicizzate virtualmente

PRO:

- **Indicizzando la cache con indirizzi virtuali si evita di dover ricorrere al TLB per accedere ad un elemento della cache. Il TLB viene usato solo in caso di cache miss.**

CONTRO:

- **Aliasing: In caso di condivisione di pagine fisiche tra diversi programmi con indirizzi virtuali diversi, in cache potrebbero essere presenti più riferimenti allo stessa pagina. Se in scrittura se ne altera solo uno, la modifica non viene vista da tutti i processi.**

Meccanismi di protezione

PROBLEMA:

Nonostante più processi condividano (e riusino) la stessa memoria principale occorre garantire che non sia possibile per un processo leggere o scrivere all'interno dello spazio di indirizzamento di un altro processo o del sistema operativo:

OSSERVAZIONE:

Ciascun processo ha la propria page table, per cui è sufficiente che:

1) il SO deve impedire che un processo possa cambiare il riferimento alla propria page table.
2) Il SO deve poter modificare le tabelle delle pagine.

SOLUZIONE:

1) Porre le page tables nello spazio di indirizzamento del SO.
2) In caso di richiesta di condivisione di pagine tra processi è compito del sistema operativo effettuare le necessarie operazioni sulle rispettive page tables.

SUPPORTO HW:

1) (Almeno) 2 modi di funzionamento che indicano se è in esecuzione un processo utente o il SO.
2) Impedire ad un processo che gira in modo utente di accedere ad alcune informazioni di stato della CPU (le relative operazioni sono eseguibili solo in modo kernel):
    1) Bit modo utente/modo kernel
    2) Page table register
    3) TLB
3) Permettere il passaggio sicuro da modo utente a modo kernel e viceversa:
    - System Call: Il meccanismo è simile a quello delle interruzioni hw (context switch), in questo caso si parla di interruzioni software.

Meccanismi di protezione, invalidamento TLB e CACHE

PROBLEMA:

- In caso di cambio di processo o di contesto, in presenza di TLB, non è sufficiente cambiare il puntatore alla page table.
- Occorre invalidare il TLB così da forzarlo a caricare le traduzioni degli indirizzi del nuovo processo.

ATTENZIONE:

- Se la frequenza di cambio di contesto è troppo alta le prestazioni si deteriorano!

ALTERNATIVA: Identificatore di processo

1) Il problema è legato al fatto che due processi distinti utilizzano lo stesso insieme di indirizzi virtuali => E' possibile estendere lo spazio di ind. Virtuale con un identificatore di processo. Tale identificatore (in genere pochi bit) sono memorizzati all'interno del TAG del TLB: si ha un hit nel TLB solo la traduzione per quell'indirizzo virt. è stata precedentemente scritta dallo stesso processo attualmente in esecuzione.

… e per le cache?

Problemi (e soluzioni) analoghe possono sorgere con la cache, nel caso in cui essa sia indicizzata tramite indirizzi virtuali.