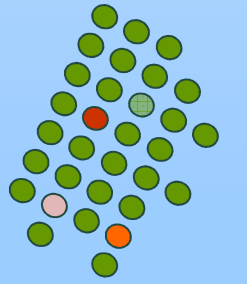
Memoria virtuale

Capitolo 9 - Silberschatz



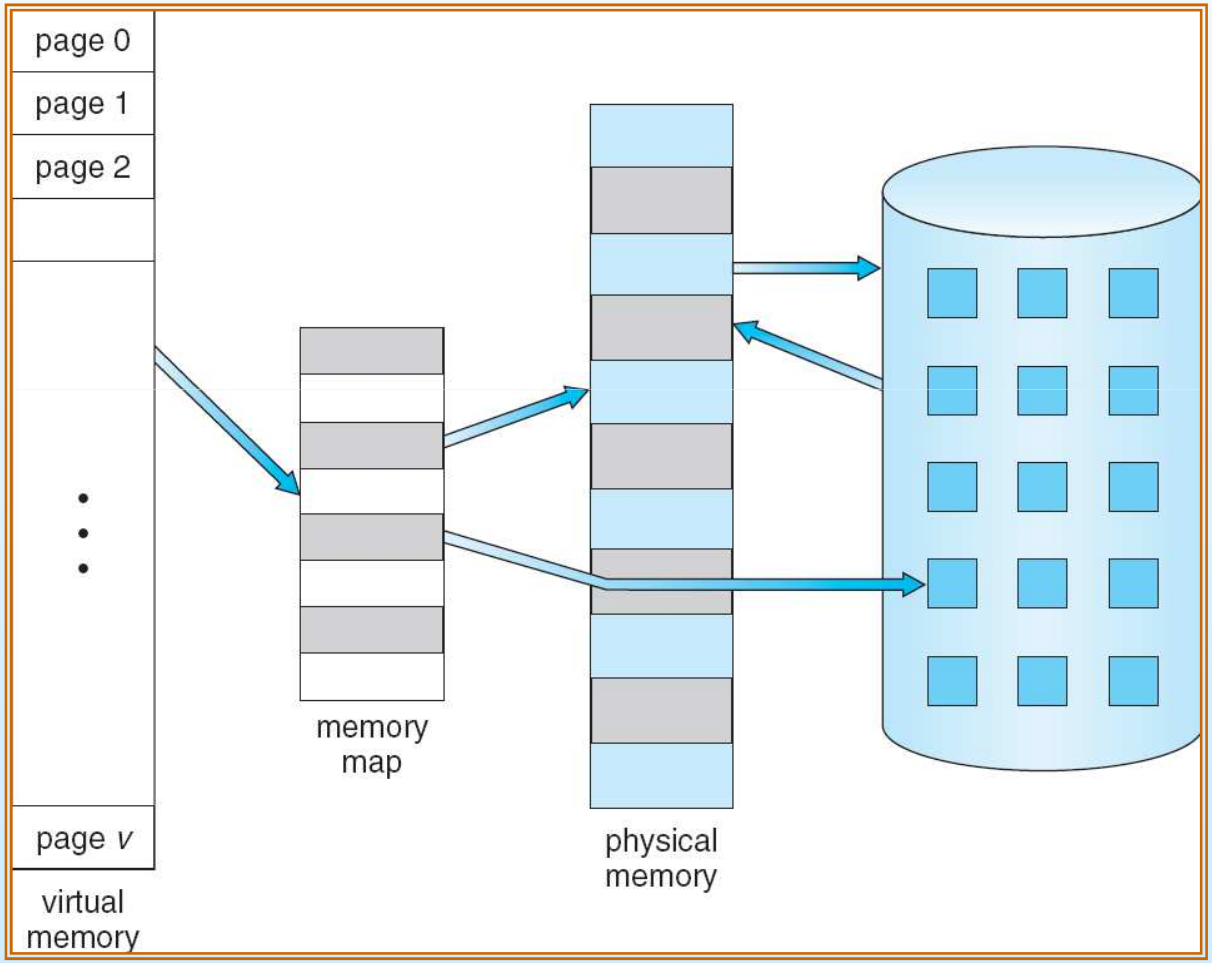
Processo in memoria

- Istruzioni in memoria prima di essere eseguite. Condizione necessaria e ragionevole … ma le dimensioni del programma devono essere correlate a quelle della memoria
- Spesso, i programmi contengono:
  - Codice per condizioni d'errore eseguite raramente
  - Tabelle e array sovradimensionati
  - Opzioni e caratteristiche utili raramente
- Soluzione: caricamento dinamico? Parziale e a carico del programmatore

Memoria virtuale

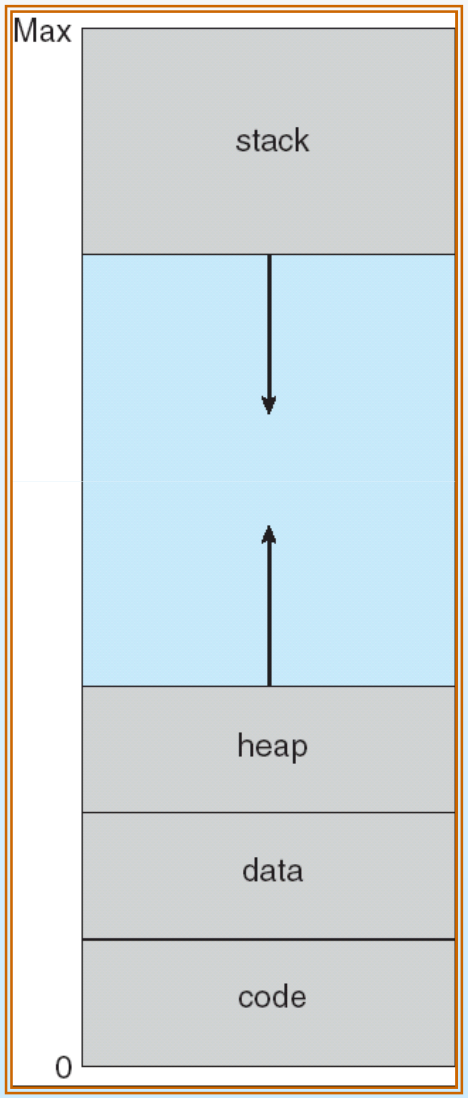
- Completa la separazione della memoria logica dalla memoria fisica.
  - Solo una parte del programma necessita di essere in memoria per l'esecuzione.
  - Lo spazio di indirizzamento logico può quindi essere più grande dello spazio di indirizzamento fisico.
  - Gli spazi di indirizzamento virtuali possono essere condivisi da diversi processi.
  - Maggiore efficienza nella creazione dei processi.

Memoria virtuale più grande della memoria fisica

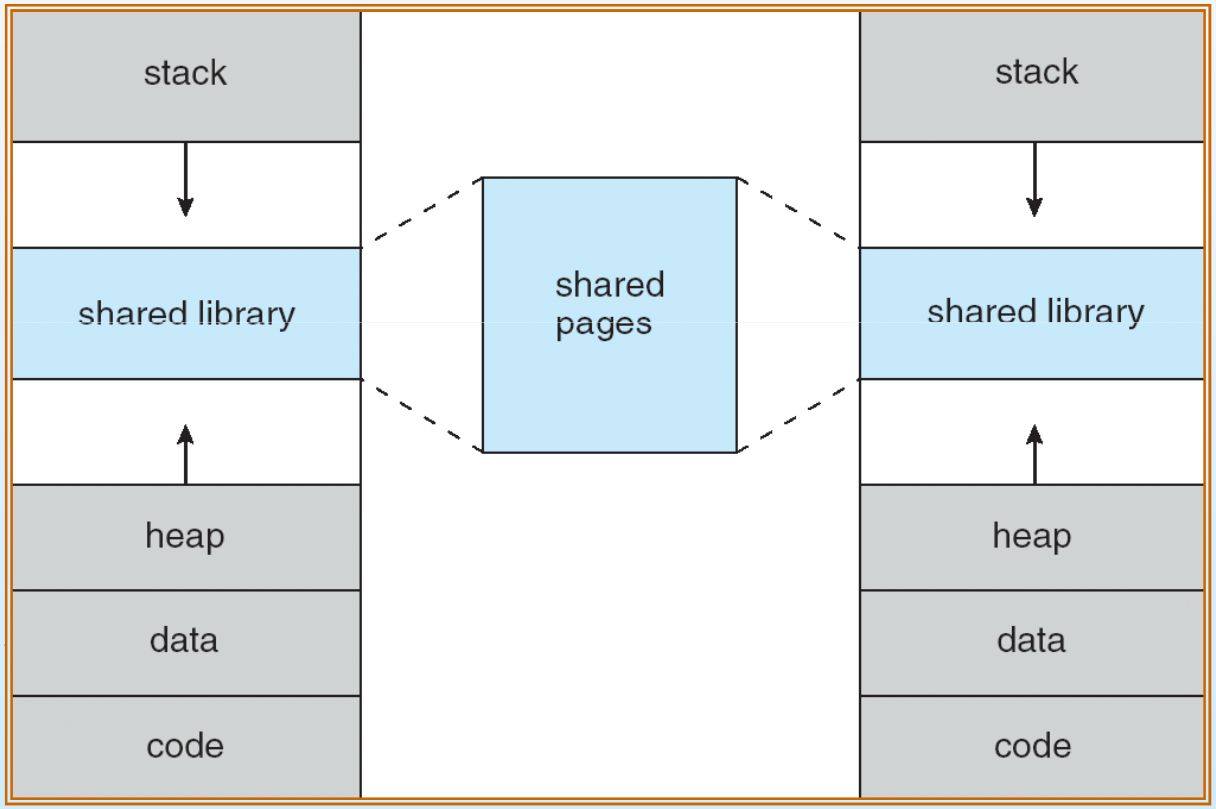


Spazio di indirizzamento virtuale

- Lo spazio tra l'heap e lo stack è spazio di indirizzamento virtuale del processo, ma richiede pagine fisiche realmente esistenti solo nel caso che l'heap o lo stack crescano



Libreria condivisa usando la memoria virtuale



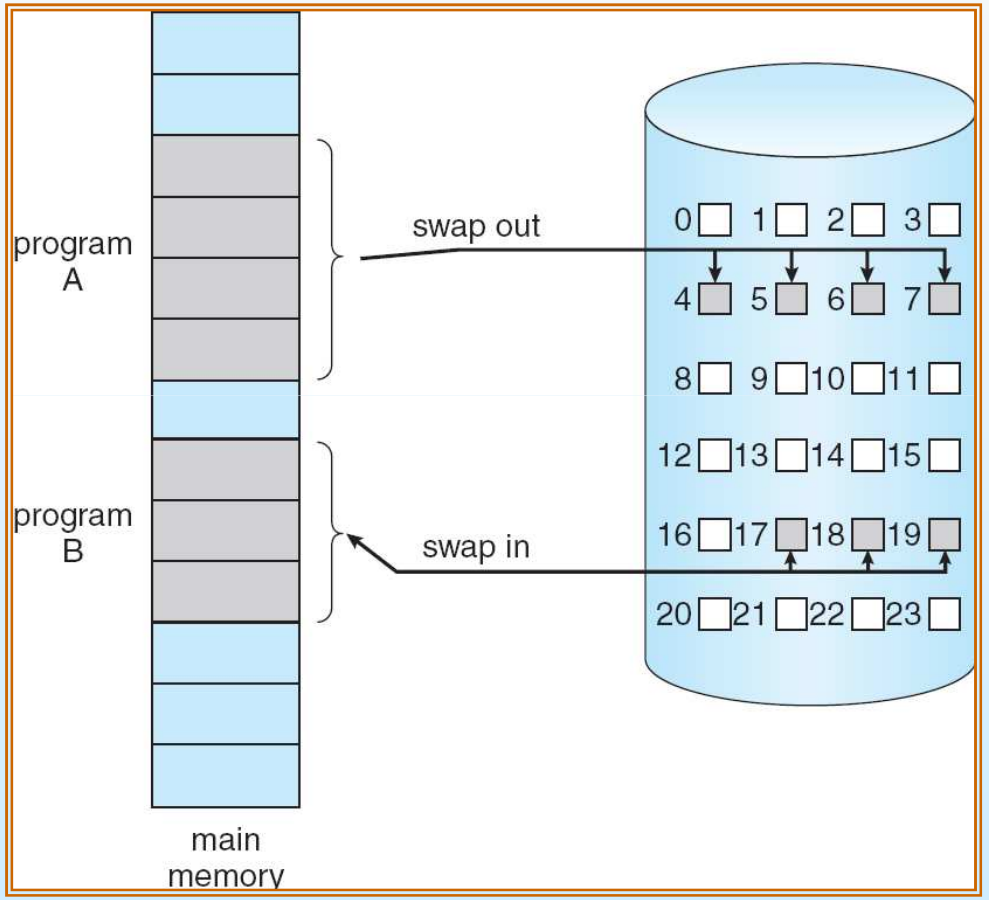
Memoria virtuale

- La memoria virtuale può essere implementata attraverso:
  - Paginazione su richiesta (demand paging).
  - Segmentazione su richiesta (demand segmentation).

Paginazione su richiesta

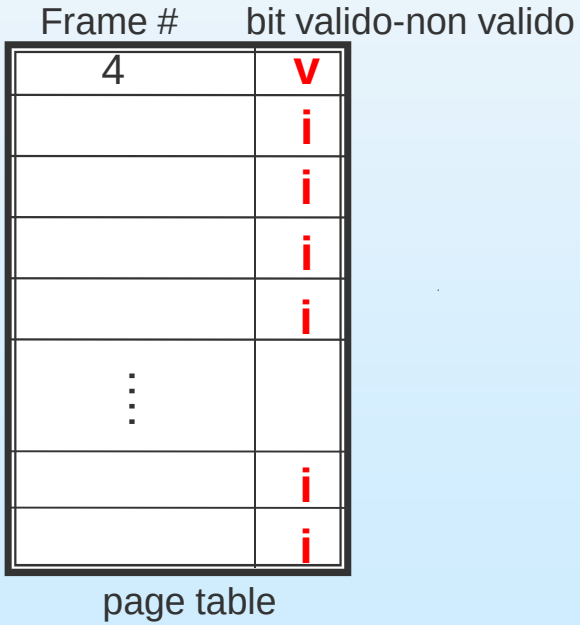
- Introdurre una pagina in memoria solo se necessario
  - Meno I/O
  - È necessaria meno memoria.
  - Più utenti.
- La pagina è necessaria ➔ riferimento ad essa:
  - riferimento non valido ➔ termine del processo.
  - non in memoria ➔ portare in memoria.

Trasferimento di pagine dal disco



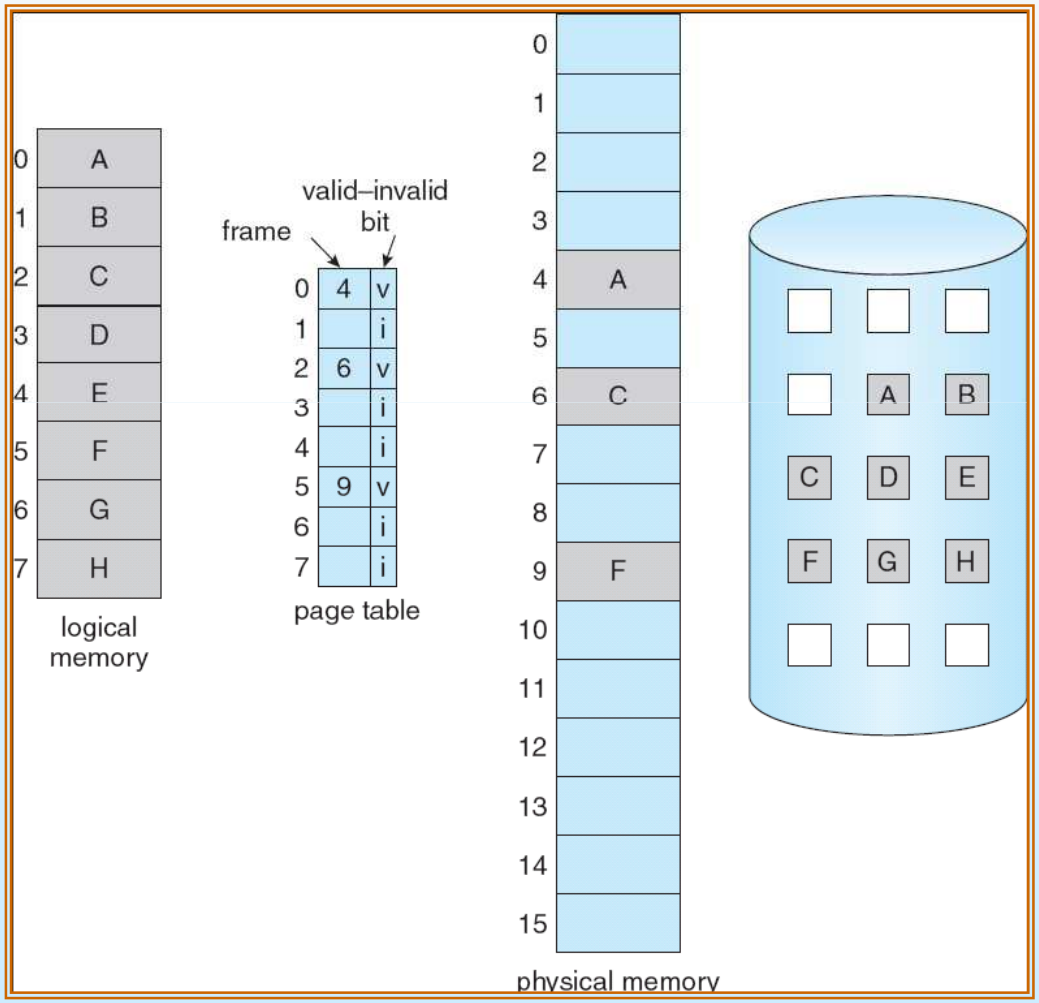
Bit di validità

- Ad ogni elemento della tabella delle pagine è associato un bit (**v** ⇒ in-memoria, **i** ⇒ non-in-memoria)
- Inizialmente il bit valido–non valido è impostato a **i** per tutti gli elementi
- Esempio di un'istantanea della tabella delle pagine:



- Durante la traduzione dell'indirizzo, se il bit è **i** nell'elemento della tabella delle pagine ➔ mancanza di pagina **i** ⇒ fault.

Memoria e disco



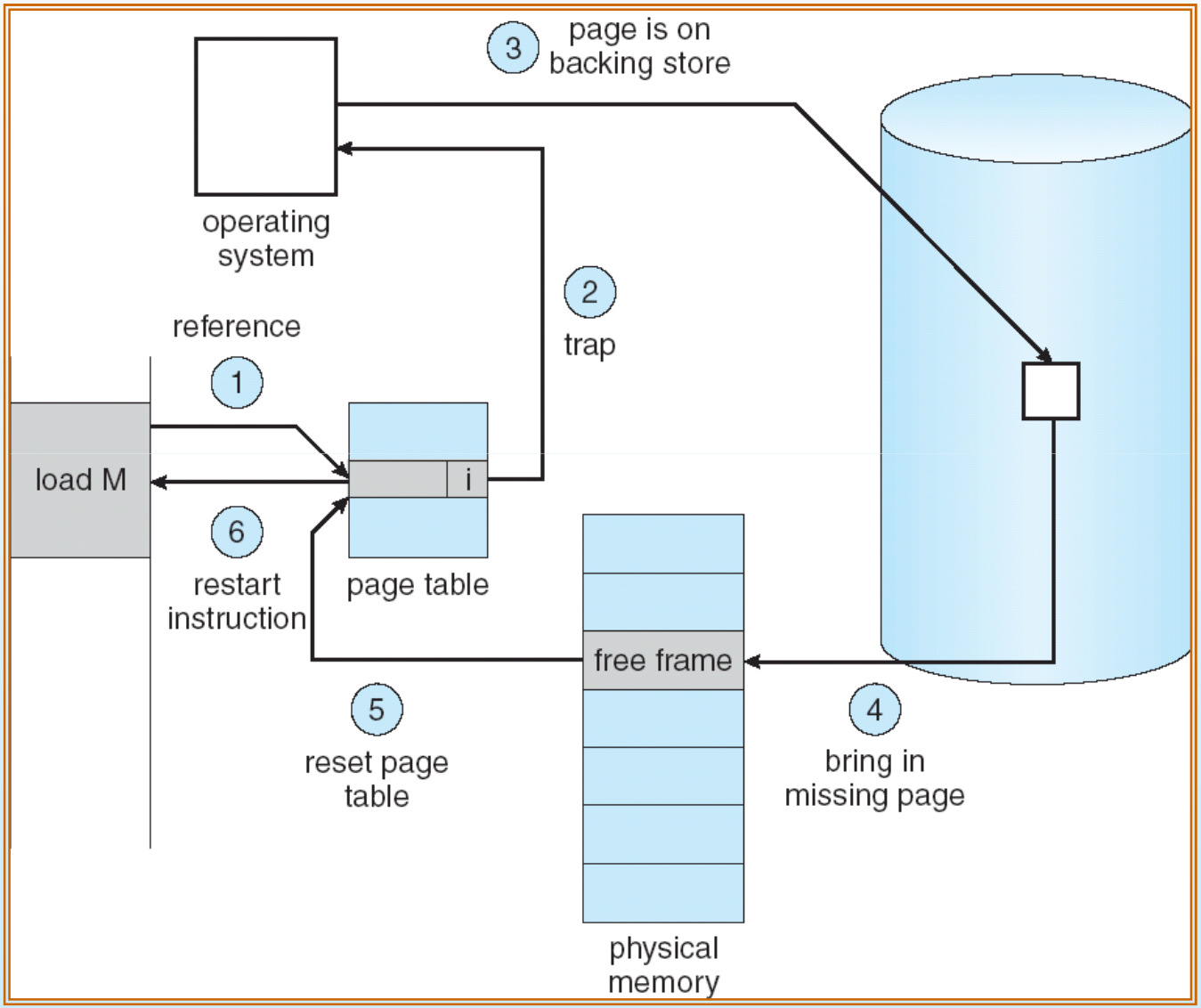
Mancanza di pagina

- Quando avviene un riferimento ad una pagina non presente in memoria (cioè una pagina contrassegnata come non valida), si provoca una trap al sistema operativo:

page fault

- Il sistema op. esamina una tabella interna per decidere:
  - Riferimento non valido (non ci si riferisce alla mem del processo) ➔ termine del processo.
  - Riferimento valido. Pagina non in memoria.
- Cercare un frame libero.
- Spostare la pagina nel frame.
- Modificare la tabella della pagine, validità del bit = v.
- Riavviare l'istruzione che ha causato il page fault

Passi per gestire un Page Fault



Istruzioni pazialmente eseguite

- Riavvia l'istruzione
  - Add (a,b,c)
    - Se eseguendo tale istruzione manca la pagina dove collocare il risultato c, si carica la pagina dove deve essere conservato c e poi viene rieseguita tutta l'istruzione (caricamento di a, con accesso alla sua pagina - caricamento di b, con accesso alla sua pagina - somma – memorizzazione in c)
  - MVC (move character)
    - Muove una sequenza di byte da una locazione ad un altra. Se durante tale movimento si scopre che una delle 2 sequenze esce dal confine di una pagina, si carica la pagina e riparte l'istruzione

Gestione di pagine fault: passi

1. Segnale di eccezione al SO
2. Salvataggio registri e stato
3. Verifica se interruzione dovuta a mancanza pagina
4. Determinazione locazione su disco
5. Lettura da disco – attese: coda/latenza/posizionamento
6. Allocazione CPU ad altro processo durante l'attesa
7. Interruzione controller disco per I/O completato
8. Salvataggio registri e stato dell'altro processo
9. Verifica interruzione dal disco
10. Aggiornamento tabelle
11. Attesa CPU nuovamente allocata
12. Recupero registri utente, stato e ripresa

Prestazione della paginazione su richiesta

- Probabilità di page-fault $0 \leq p \leq 1$
  - se $p = 0$ non ci sono mancanze di pagine
  - se $p = 1$, ogni riferimento è una mancanza di pagina

- Tempo di accesso effettivo (EAT)

$$\text{EAT} = (1 - p) \times \text{accesso alla memoria}$$

$$+ p \text{ (page fault overhead}$$

$$+ \text{lettura della pagina}$$

$$+ \text{overhead ripresa)}$$

Esempio di paginazione su richiesta

- Tempo di accesso alla memoria = 200 nanosecondi
- Tempo medio di gestione di un page fault = 8 millisecondi =8,000,000 nanosecondi

$$EAT = (1 - p) \times 200 + p \times 8{,}000{,}000$$

$$= 200 + p \times 7{,}999{,}800$$

- Se un accesso su 1000 causa un page fault, allora

$$EAT = 8199{,}8 \text{ nanosecondi}$$

il tempo di accesso effettivo è 40 volte superiore al tempo di accesso alla memoria centrale!!

Creazione di processi

- La memoria virtuale offre altri benefici durante la creazione del processo:
  - È inutile duplicare tutto il codice se, come spesso accade, c'è una successiva chiamata ad exec
    - Copia su scrittura (copy-on-write)

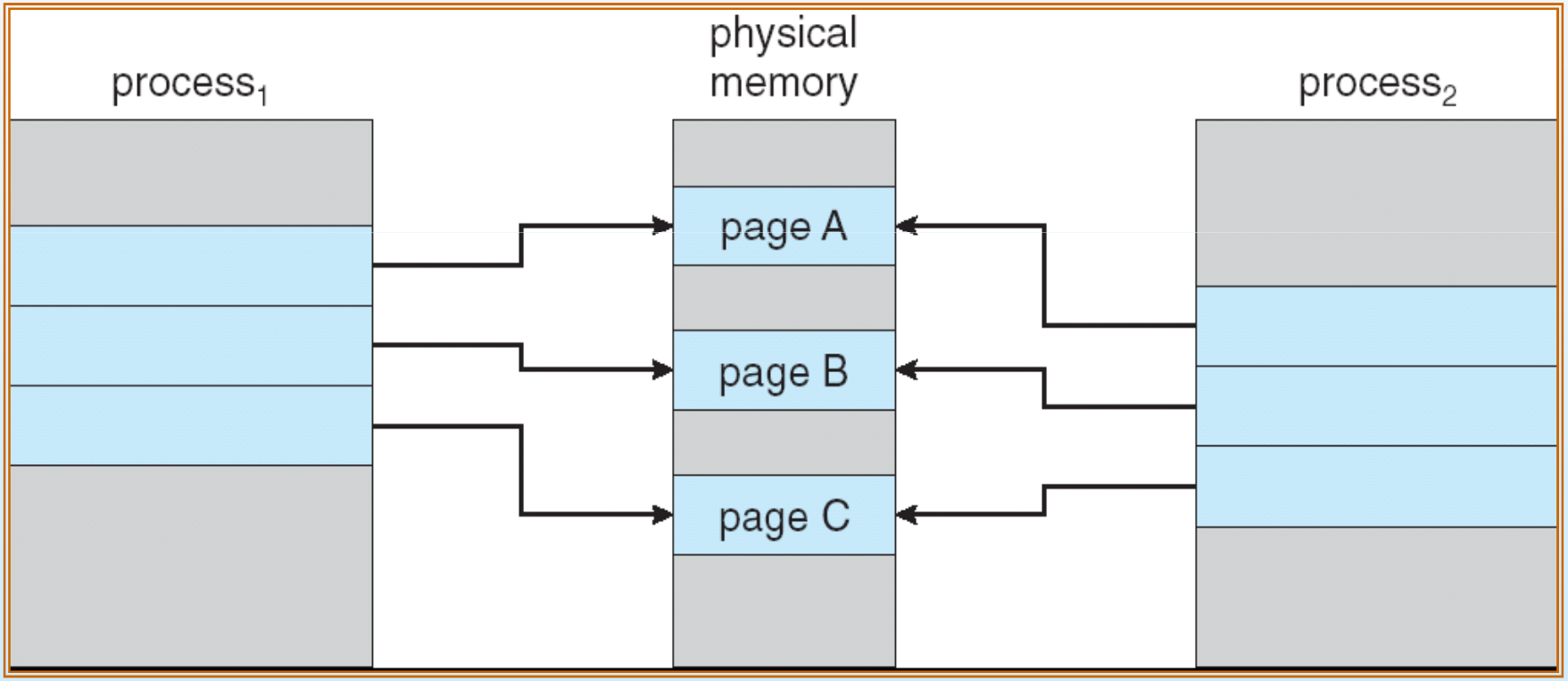
Copia su scrittura

- La copia su scrittura permette ai processi padre e figlio di *condividere* inizialmente le stesse pagine in memoria. Quando uno dei due processi scrive in una pagina condivisa, allora viene creata una copia della pagina condivisa.

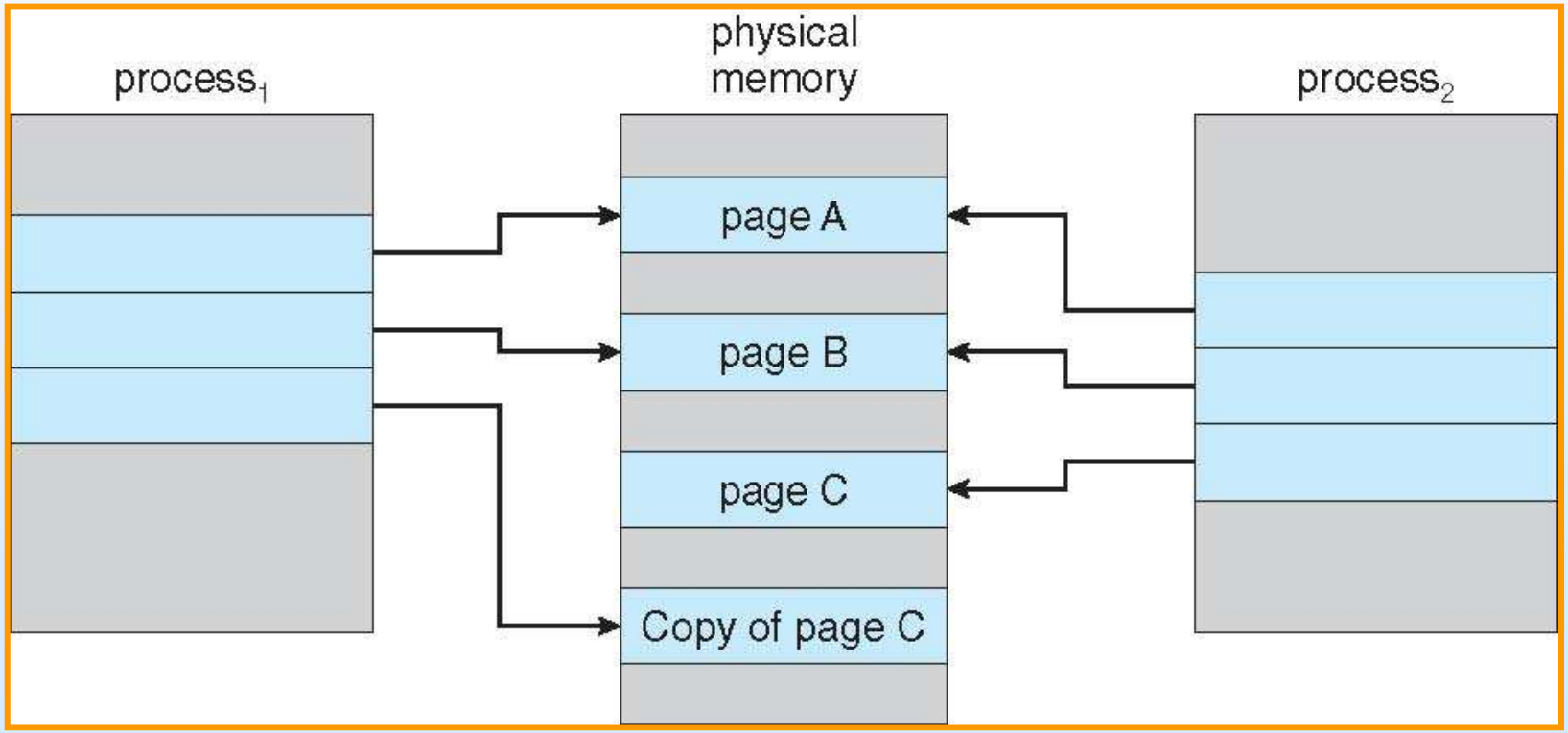
- La copia su scrittura fornisce una maggiore efficienza nella creazione dei processi, poichè solo le pagine modificate vengono copiate.

Prima che process$_1$ modifichi page C

Processo padre e processo figlio condividono le pagine



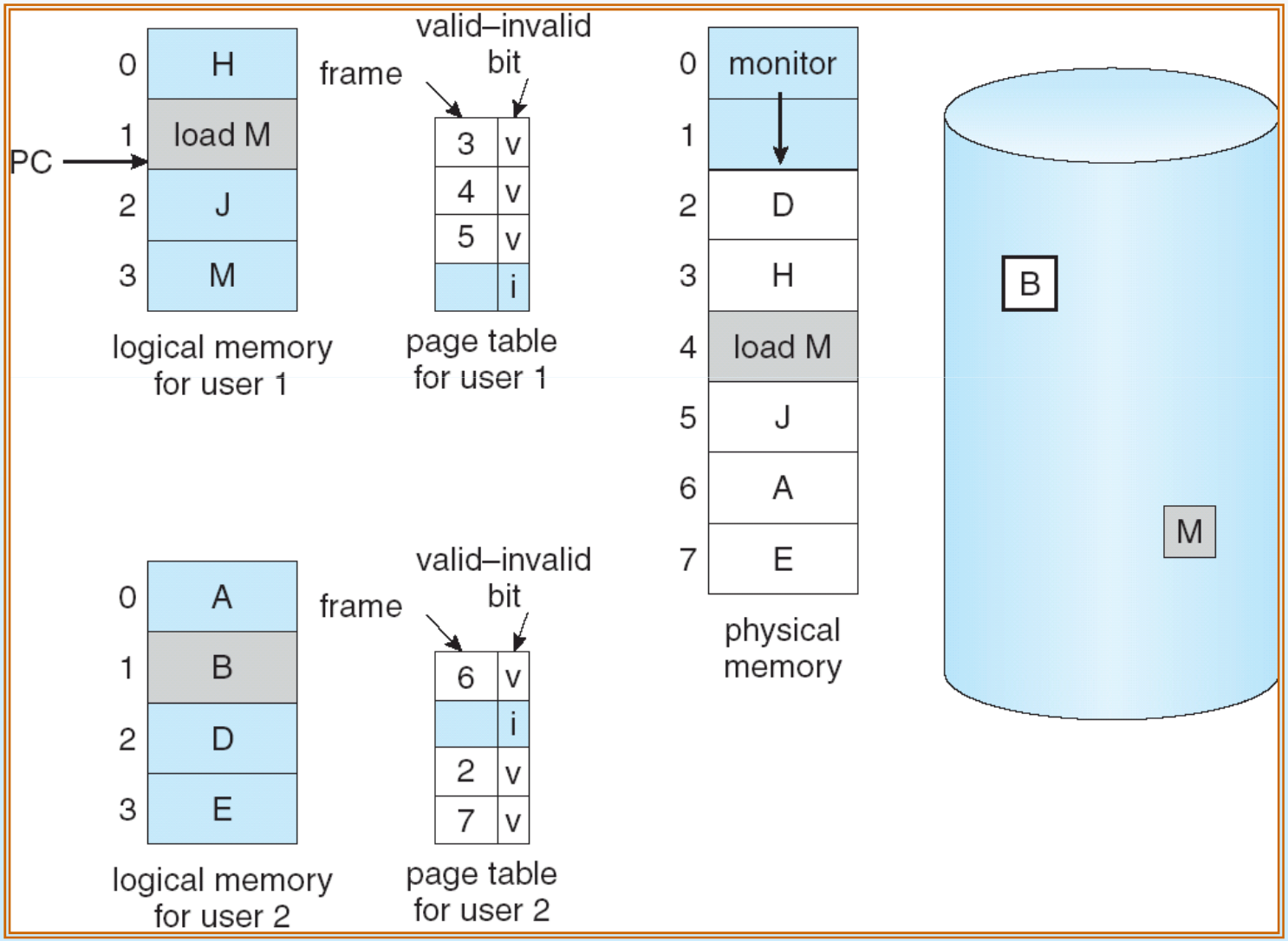
Dopo che Process$_1$ ha modificato Page C



Sovrallocazione

- Cosa succede se non ci sono frame liberi?
- Sostituzione di pagina – trovare alcune pagine in memoria, ma non veramente in uso, e spostarle sul disco.
  - algoritmo di sostituzione per scegliere frame vittima
  - prestazione – l'algoritmo deve minimizzare il numero di page fault
- La stessa pagina può essere portata in memoria più volte.

Necessità di sostituzione pagina



Gestione del page fault con sostituzione di pagina

Modifica della procedura di gestione di page fault per includere la sostituzione di pagina.

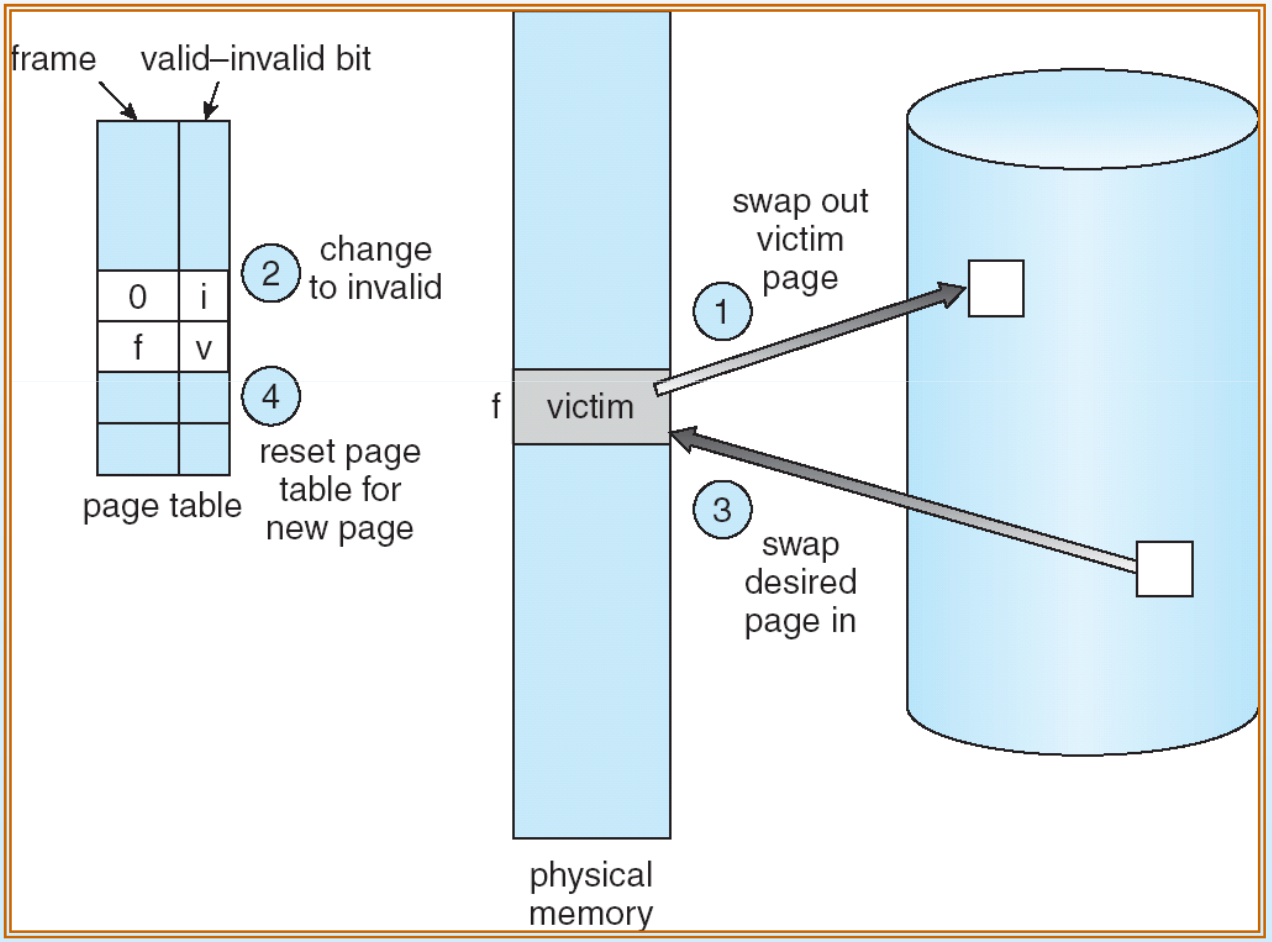
1. Individuare la posizione della pagina desiderata sul disco.

2. Trovare un frame libero:
   - se c'è un frame libero, usarlo;
   - se non c'è nessun frame libero, usare l'algoritmo di sostituzione delle pagine per selezionare un frame **vittima**

3. Leggere la pagina desiderata nel frame libero; aggiornare le tabelle dei frame e delle pagine.

4. Riprendere il processo

Sostituzione della pagina



Sostituzione di pagina

- Se non esiste alcun frame libero c'è bisogno di due trasferimenti di pagine
- Soluzione: *bit di modifica* (modify bit) – solo le pagine modificate sono scritte su disco.
- Questa tecnica vale anche per le pagine a sola lettura.
- La sostituzione della pagina completa la separazione fra memoria logica e memoria fisica.

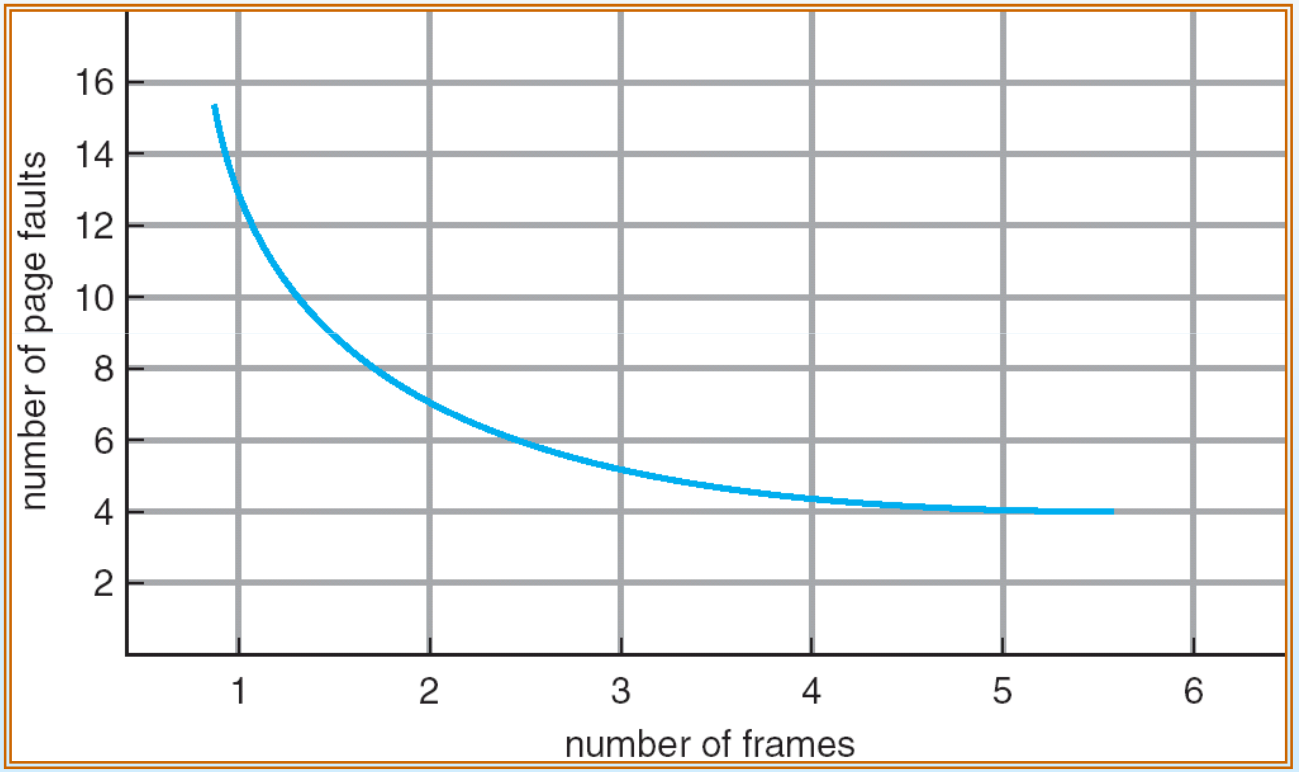
Algoritmo di sostituzione delle pagine

- Si desidera il più basso tasso possibile di page fault
- Un algoritmo viene valutato facendolo operare su una particolare stringa di riferimenti alla memoria e calcolando il numero di page fault
- La stringa di riferimanto che useremo è:

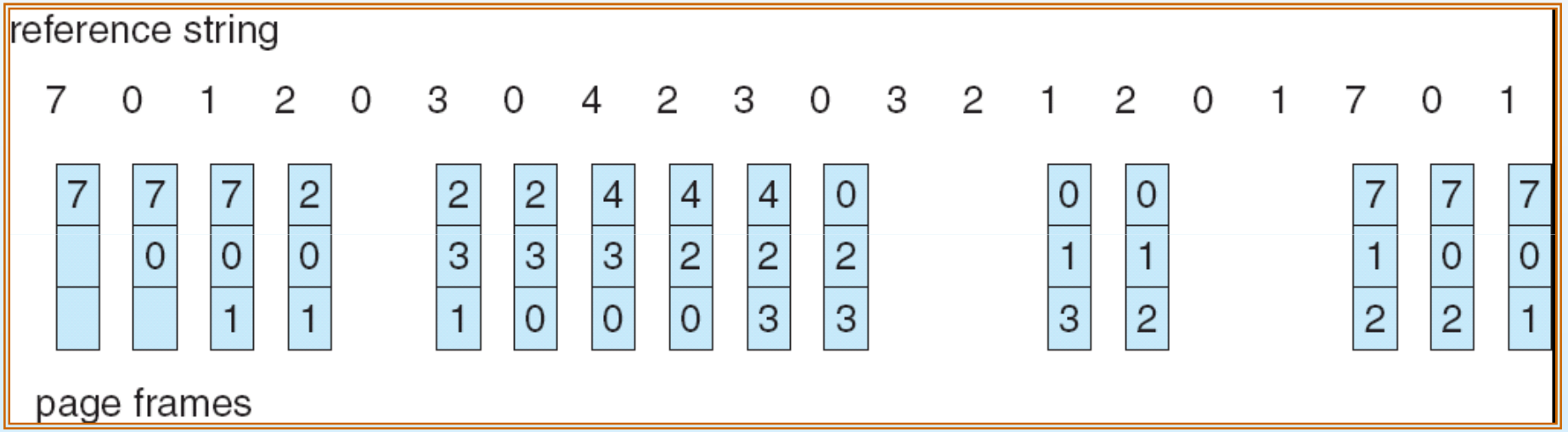
  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

- Numero dei frame disponibili è essenziale per la valutazione

Mancanza di pagina e numero di frame



Sostituzione First-In-First-Out (FIFO) della pagina



Algoritmo FIFO

- Stringa di riferimenti: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pagine possono essere in memoria per ogni processo)

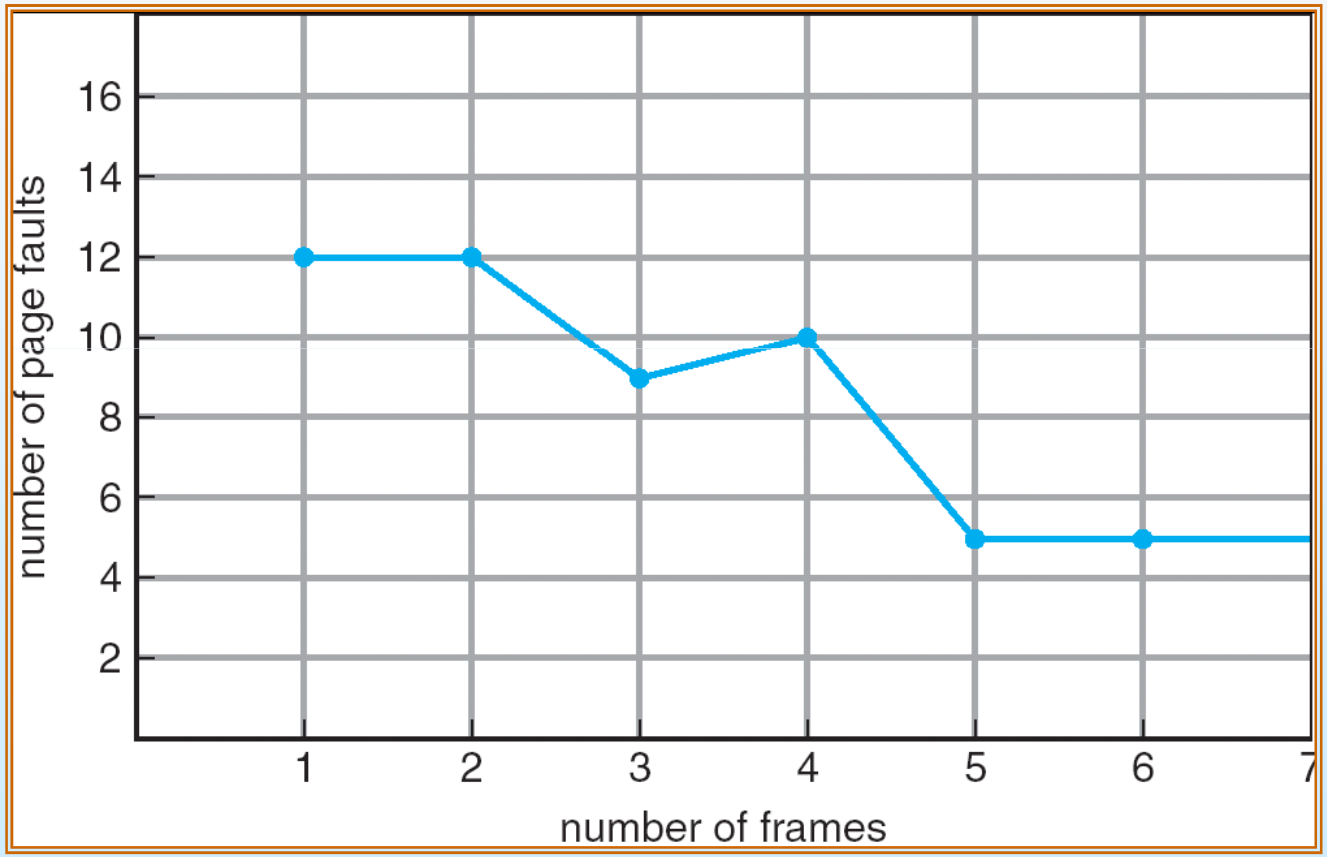
1	1	4	5	9 page faults
2	2	1	3	
3	3	2	4	

- 4 frames

1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

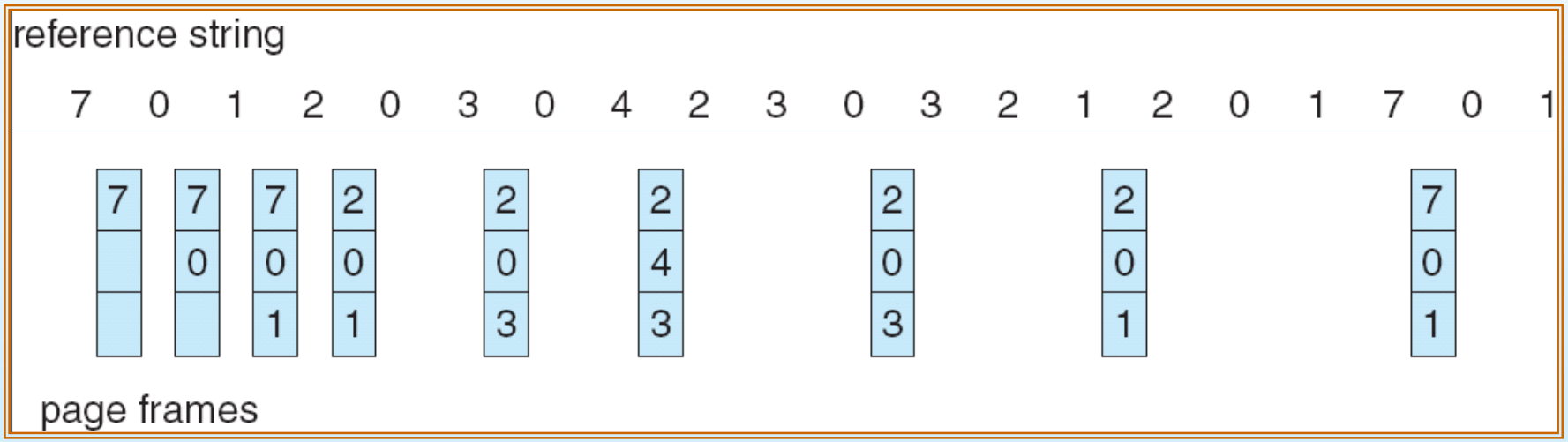
Sostituzione FIFO - Belady's Anomaly: più frames $\Rightarrow$ più page faults

FIFO: l'anomalia di Belady



Sostituzione ottimale delle pagine

Sostituisce la pagina che non si userà per il periodo di tempo più lungo.



Algoritmo ottimale

- Riprendere la pagina che non sarà usata per un lungo periodo di tempo
- Esempio: 4 frames

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

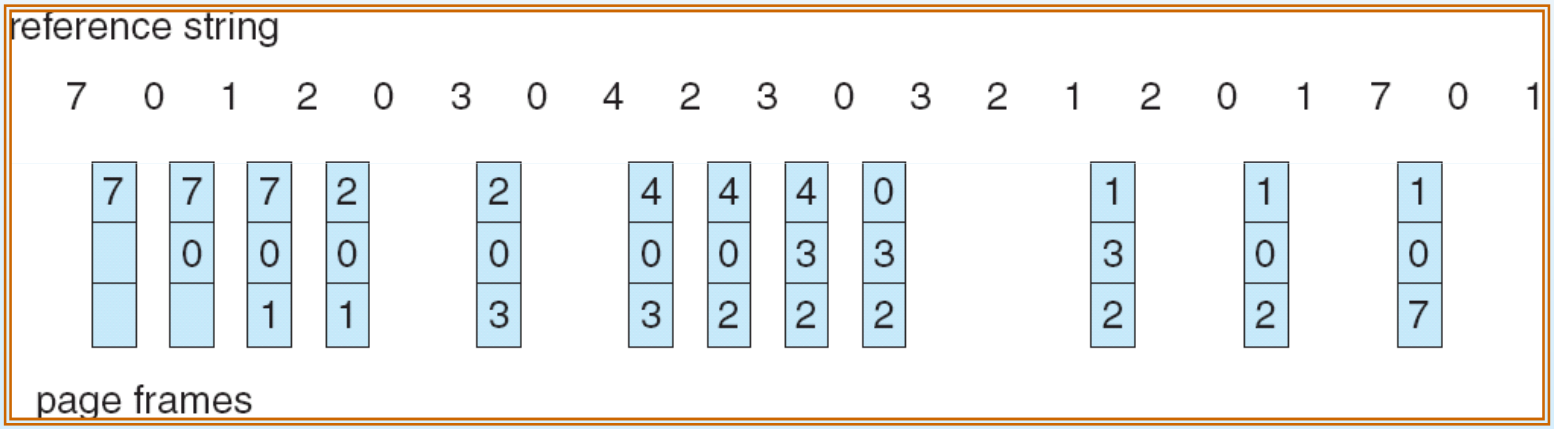
1	4
2	
3	
4	5

6 page faults

- Questo algoritmo richiede la conoscenza della sequenza di riferimenti futuri. Come avere questa informazione?
- Usato per misurare quanto sono buone le prestazioni di un algoritmo

Algoritmo LRU (Last Recently Used)

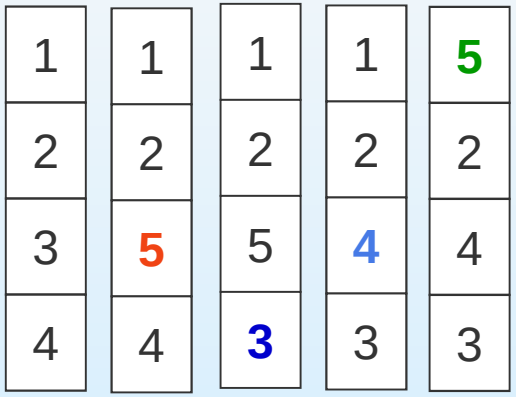
Sostituisce la pagina che non è stata usata per il periodo di tempo più lungo.



Algoritmo Least Recently Used (LRU)

- Esempio: 4 frames

1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**



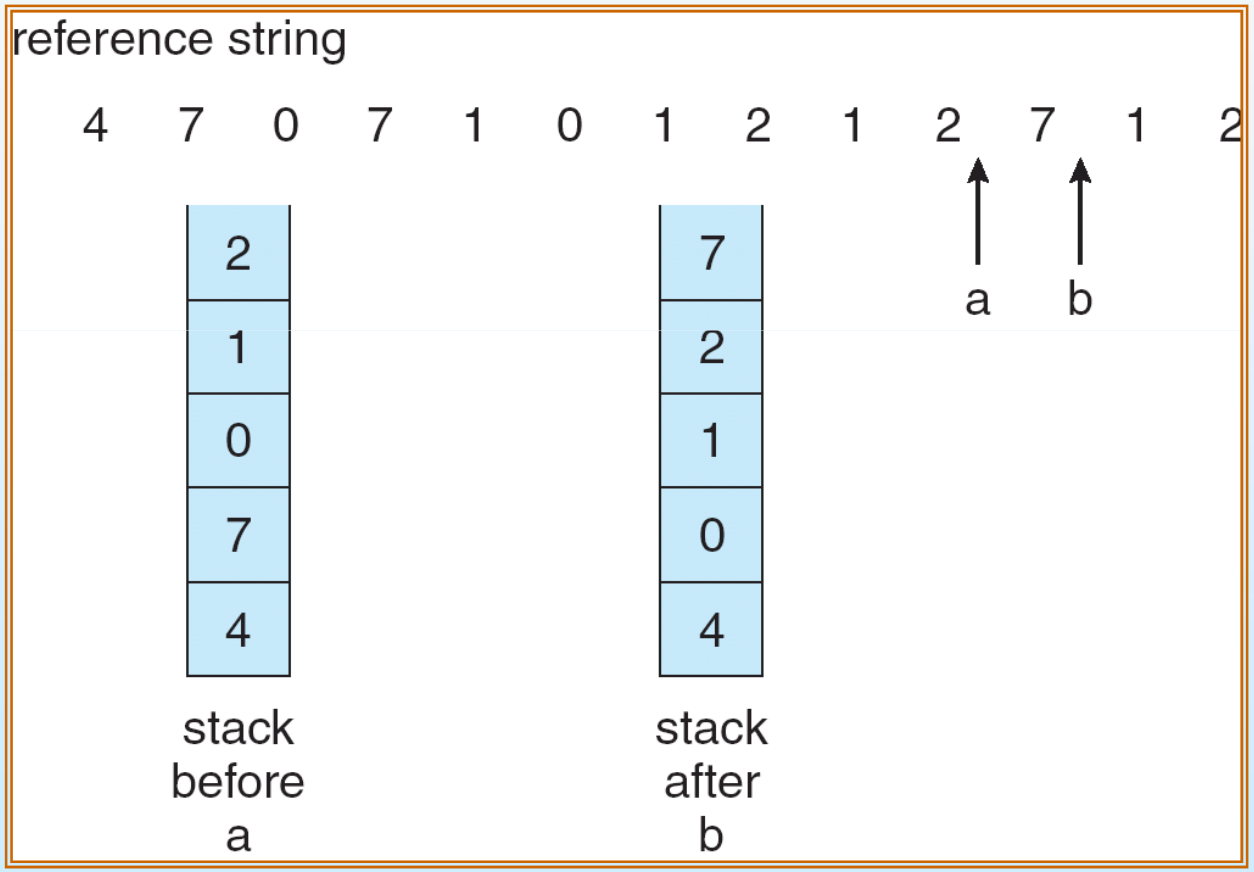
Implementare LRU

- Problema: determinare un ordine per i frame secondo il momento dell'ultimo uso fatto
- Implementazione del contatore.
  - Si assegna alla CPU un contatore che si incrementa ad ogni riferimento alla memoria. Si assegna ad ogni entry della tabella delle pagine un ulteriore campo.
  - Ogni volta che si fa riferimento ad una pagina viene copiato il valore del contatore della CPU nel campo corrispondente per quella pagina nella tabella delle pagine.
  - Quando una pagina deve essere sostituita si guardano i valori dei contatori.

Implementare LRU

- Implementazione dello stack:
  - mantenere uno stack dei numeri di pagina in una lista a doppio collegamento
  - Riferimento ad una pagina:
    - mettere la pagina in cima allo stack;
    - nessuna ricerca per la sostituzione: basta prelevare la pagina dal fondo;
    - ogni volta che si fa riferimento ad una pagina già presente nello stack bisogna prelevare tale riferimento dal centro
  - richiede di cambiare al più 6 puntatori.

Uso di uno stack per registrare i riferimenti alle pagine usate più di recente



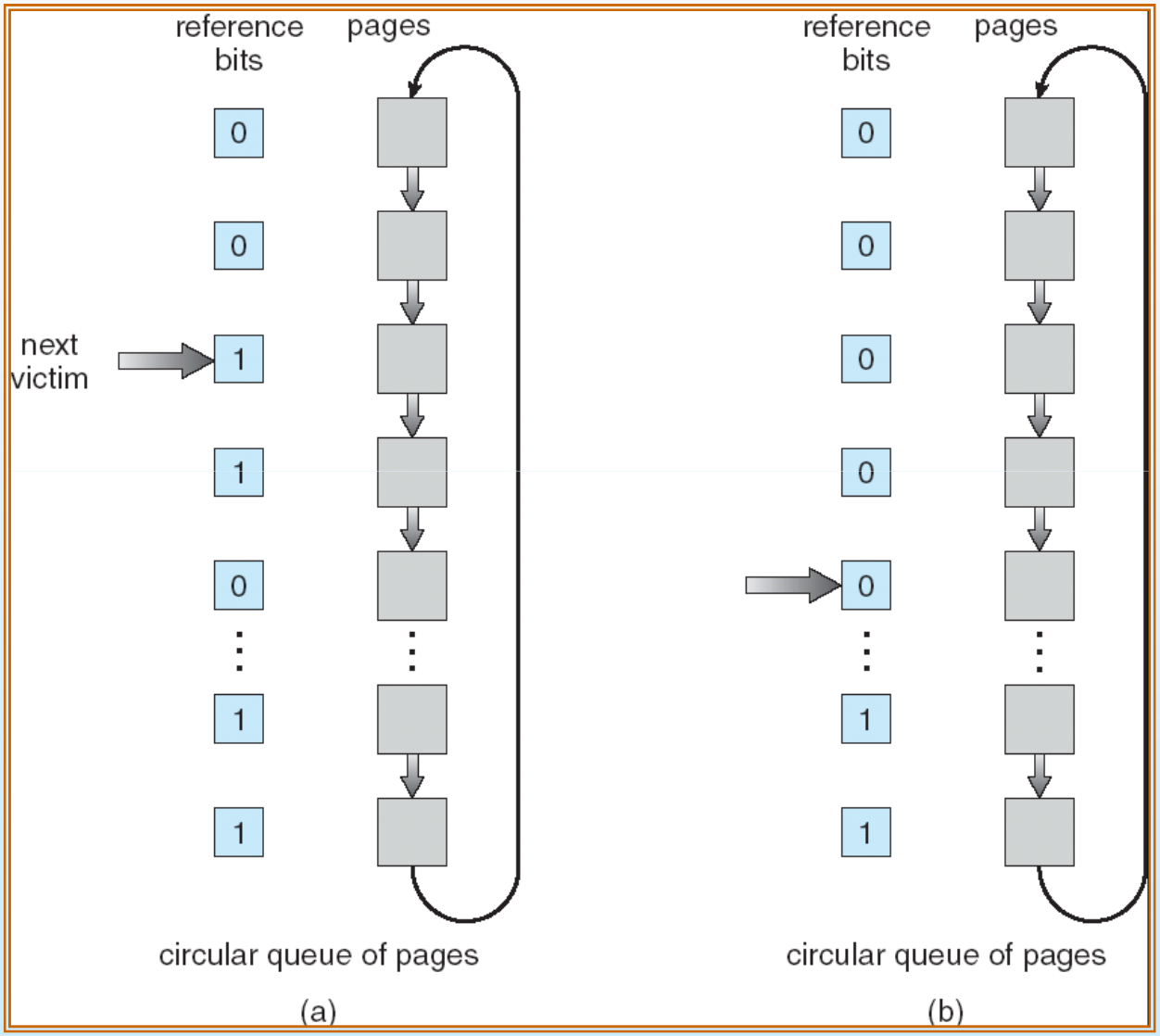
Approssimazione dell' algoritmo LRU

- Sia l'algoritmo ottimale che quello LRU non sono soggetti all'anomalia di Belady.
- Ma sono pochi i sistemi in grado di avere una architettura(contatori, stack) per la gestione di LRU.
  - Molti sistemi tentano delle approssimazioni.
- Architettura: bit di riferimento.
  - Ad ogni pagina è associato un bit, inizialmente = 0.
  - Quando la pagina è referenziata il bit è impostato a 1.
  - Rimpiazzare la pagina che è a 0 (se ne esiste una).

Algoritmo della seconda chance

- Miglioramento FIFO.
  - Le pagine sono disposte in una lista circolare
  - Quando occorre selezionare una pagina vittima inizia la scansione della lista:
    - se una pagina ha il bit di riferimento a 1 lo si pone a 0 e si passa alla successiva (la pagina rimane in memoria – le si dà una seconda chance);
    - altrimenti si seleziona per essere sostituita.

Algoritmo della Seconda Chance



Altre approssimazioni

- Raffinamento dell'algoritmo della seconda chance

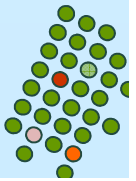
  (bit riferimento, bit modifica)

  - (0,0) non usato di recente, non modificato
  - (0,1) non usato di recente, modificato
  - (1,0) usato di recente, non modificato
  - (1,1) usato di recente, modificato

- Più bit di riferimento (e.g., tenuti in un registro a 8 bit). Periodicamente, allo scadere di un intervallo temporale, il registro viene shiftato a destra di un bit, e il bit di riferimento della pagina viene copiato nel bit più significativo del registro.

Algoritmi di conteggio

- Tenere un contatore del numero di riferimenti che sono stati fatti ad ogni pagina.
- Algoritmo LFU (Least Frequently Used): sostituisce la pagina con il più basso conteggio.
- Algoritmo MFU (Mostl Frequently Used): : sostituisce la pagina con il conteggio più alto.



Bufferizzazione delle pagine

- Pool di frame liberi per soddisfare le richieste velocemente.
- Quando si sceglie una pagina vittima, la si trasferisce nel pool dei frame liberi.
- Pagine modificate del pool dei frame liberi sono scritte sul disco periodicamente in background.
- Ricerca nel pool dei frame liberi in memoria di una pagina precedentemente sostituita e nuovamente necessaria. Probabilmente ancora in memoria e non sovrascritta se il frame non è stato riallocato.

Allocazione dei frame

- Ogni processo ha bisogno di un numero **minimo** di pagine.
- Esempio: IBM 370 – 6 pagine per gestire l'istruzione MVC**:**
  - L'istruzione richiede 6 byte, che possono estendersi su 2 pagine**.**
  - 2 pagine per gestire il "from" dell'istruzione.
  - 2 pagine per gestire il "to" dell'istruzione.
- Due principali schemi di allocazione.
  - Allocazione fissa.
  - Allocazione a priorità.

Allocazione uniforme e proporzionale

- Allocazione uniforme – avendo m frame ed n processi, si assegnano m/n frame a ciascun processo (Es: se 100 frame e 5 processi, ognuno prende 20 frame).

- Allocazione proporzionale – si assegna la memoria disponibile ad ogni processo in base alle dimensioni di quest'ultimo.

$s_i = \text{size del processo } p_i$

$S = \sum s_i$

$m = \text{numero totale di frames}$

$a_i = \text{numero di frame allocati per } p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_1 = 10$

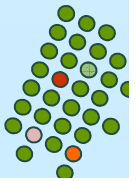
$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 \approx 5$

$a_2 = \frac{127}{137} \times 64 \approx 59$

Allocazione a priorità

- Usare uno schema di allocazione proporzionale basato sui valori delle priorità piuttosto che delle dimensioni.
- Combinazione dimensioni e priorità.



Allocazione globale e locale

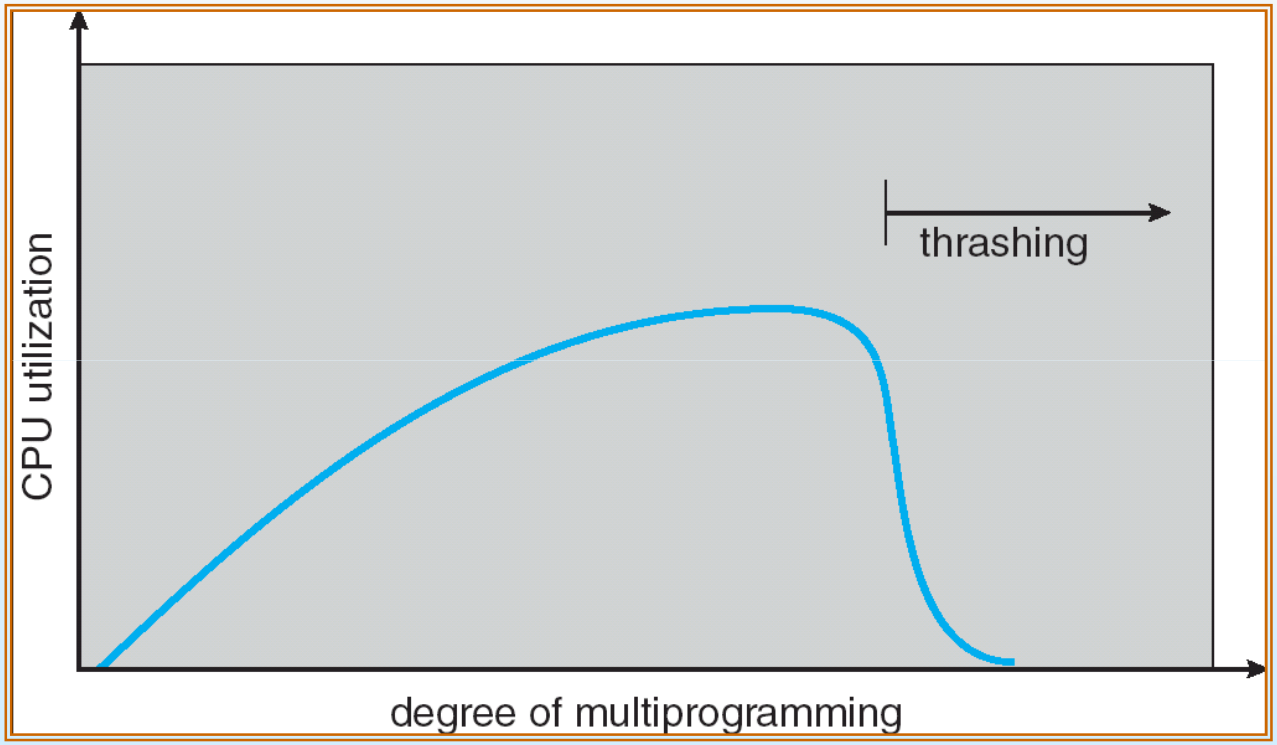
- **Sostituzione globale** – permette di selezionare un frame di sostituzione a partire dall'insieme di tutti i frame, anche se quel frame è correntemente allocato a qualche altro processo, i.e., un processo può prendere un frame da un altro processo.

- **Sostituzione locale** – vengono considerati solo i frame allocati al processo.

Thrashing

- Se un processo non ha abbastanza pagine, il tasso di page fault aumenta. Questo comporta:
  - riduzione utilizzo della CPU;
  - il sistema operativo ritiene che sia necessario aumentare il livello di multiprogrammazione;
  - un altro processo aggiunto al sistema.
- **Thrashing** ≡ si spende più tempo nella paginazione che nella esecuzione dei processi.

Thrashing

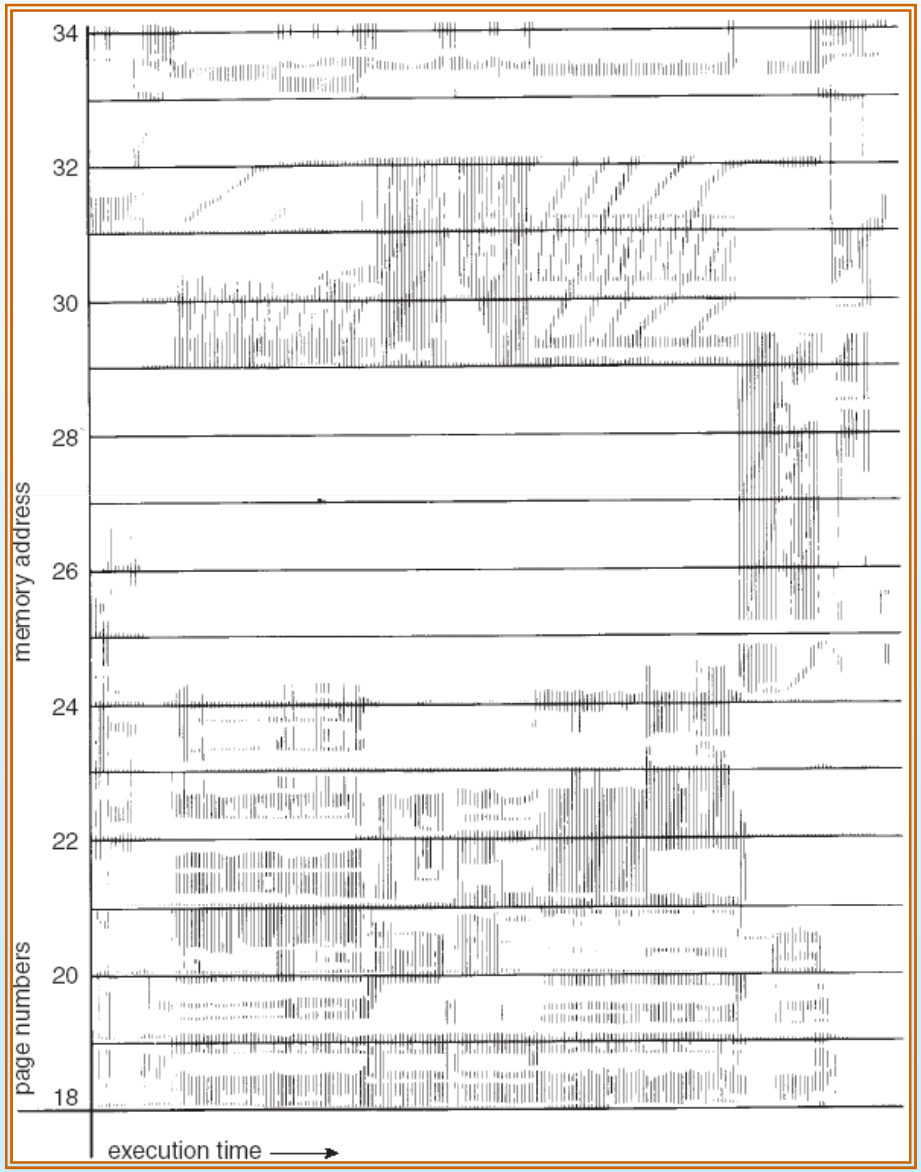


Paginazione e Thrashing

- Per evitare il thrashing occorrerebbe sapere quali, oltre che quante pagine, servono.
- Modello di località:
  - il processo si muove da una località all'altra,
  - le località possono sovrapporsi.
- Perchè si verifica il trashing?

  dimensione della località > numero di frame assegnati.

Località in una sequenza di riferimenti alla memoria



Modello del working-set

- $\Delta \equiv$ finestra di working-set $\equiv$ un numero fisso di riferimenti di pagina
  Esempio: 10,000 instruzioni

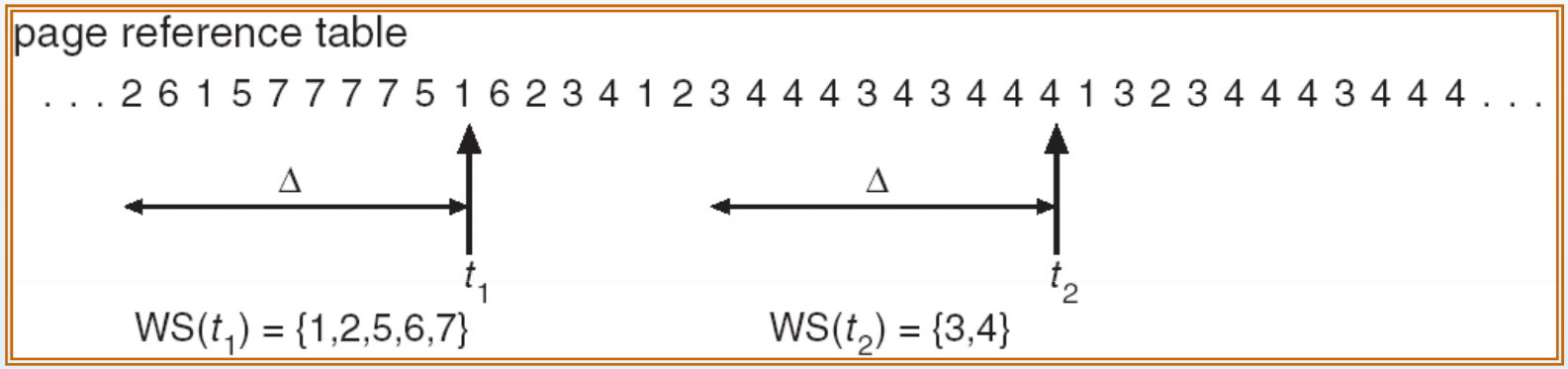
- WSS_i (working set del processo P_i) =
  numero totale di pagine cui P_i si riferisce nel più recente Δ (varia nel tempo)
  - se Δ è troppo piccolo non comprenderà l'intera località
  - se Δ è troppo grande può sovrapporre parecchie località
  - se $\Delta = \infty \Rightarrow$ il working set è l'insieme delle pagine toccate durante l'esecuzione del processo

- $D = \Sigma\ WSS_i \equiv$ richiesta globale dei frame

- Se $D > m \Rightarrow$ Thrashing

- Se $D >$ m, allora occorre sospendere uno dei processi

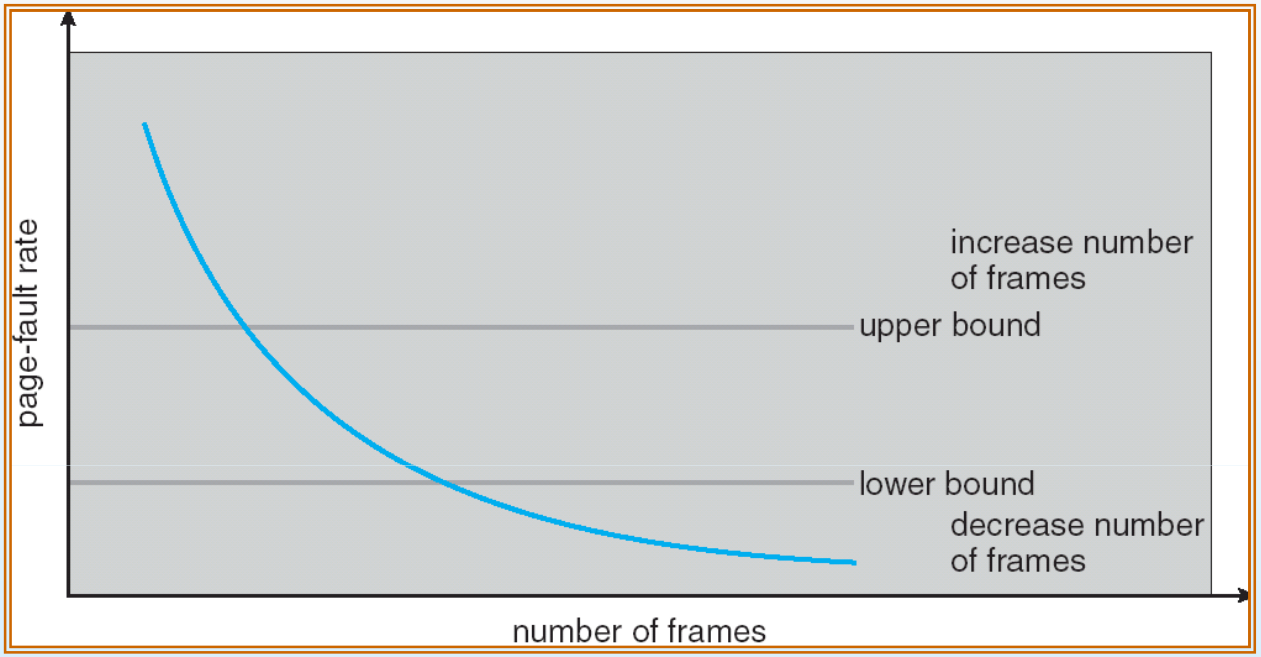
Working-set model



Mantenere traccia del Working Set

- Si può approssimare il modello del working set con interrupt a intervalli fissi di tempo generati da un temporizzatore e un bit di riferimento
- Esempio: Δ = 10,000
  - Interrupt ogni 5000 riferimenti
  - Tenere in memoria 2 bits per ogni pagina
  - Ogni volta che si riceve l'interrupt del temporizzatore si copiano e si azzerano i valori del bit di riferimento per ogni pagina
  - Se uno dei bit è uguale a 1 $\Rightarrow$ la pagina è nel working set
- Perchè non è del tutto preciso?
- Incremento = 10 bit e interrupt ogni 1000 riferimenti (il costo per servire tali segnali di interruzione più frequenti aumenta in modo corrispondente)

Frequenza di Page-Fault



- Stabilire un tasso “accettabile” di page-fault
  - Se il tasso attuale è troppo basso, il processo può avere troppi frame
  - Se il tasso attuale è troppo alto, il processo ha bisogno di più frame