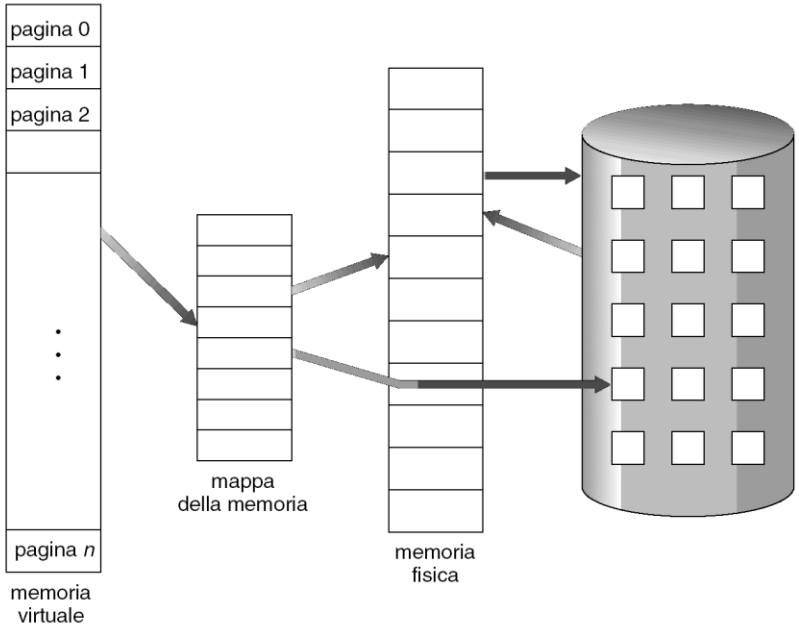
Capitolo 10: Memoria virtuale

- Ambiente.
- Richiesta di paginazione.
- Creazione del processo.
- Sostituzione della pagina.
- Allocazione dei frame.
- Thrashing.
- Altre considerazioni.

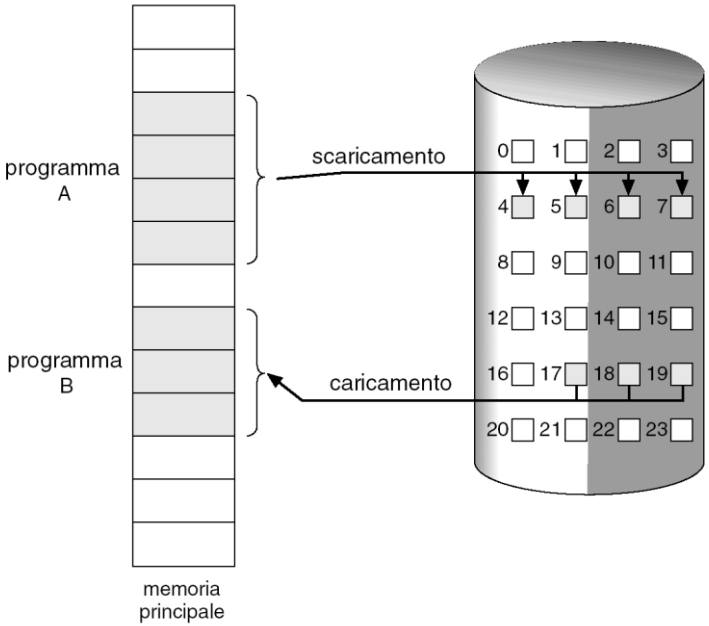
Ambiente

- **Memoria virtuale** – comporta 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 di quello fisico.
  - Più programmi possono funzionare contemporaneamente dato che ogni programma usa meno memoria fisica (aumento dell'utilizzo e del throughput della CPU).
  - Permette agli spazi di indirizzamento di essere condivisi da numerosi processi.
  - Maggiore efficienza nella creazione del processi (meno chiamate di I/O necessarie).
- La memoria virtuale rende più agevole il compito dei programmatori.
  - Non esistono vincoli sulla memoria disponibile o problemi di programmazione degli overlay.



Richiesta di paginazione

- Implementazione della memoria virtuale: introdurre una pagina in memoria solo se è necessario.
  - Meno dispositivi di I/O necessari.
  - Meno memoria necessaria.
  - Risposta più veloce.
  - Più utenti.
- La pagina è necessaria ⇒ riferimento ad essa:
  - riferimento non valido ⇒ termine del processo.
  - non in memoria ⇒ portare in memoria.



Bit valido-non valido

- Ogni elemento della tabella delle pagine è associato ad un bit valido-non valido (1 ⇒ in-memoria, 0 ⇒ non in memoria).
- Inizialmente il bit valido–non valido è impostato a 0 per tutti gli elementi.
- Esempio di un'istantanea della tabella delle pagine.

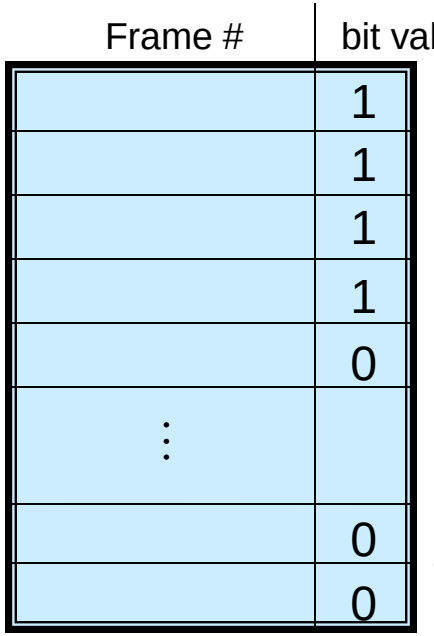
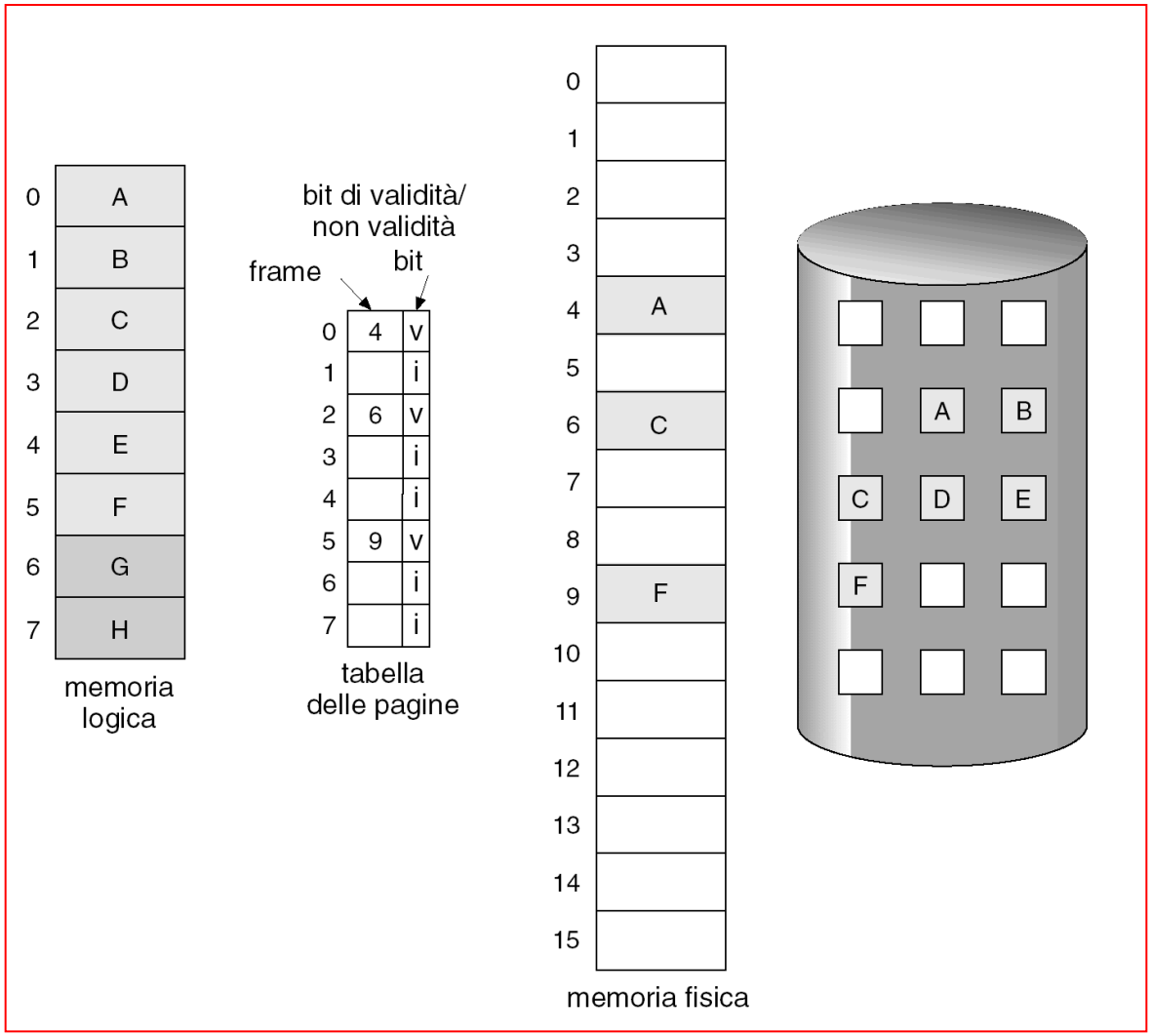


Tabella delle pagine

- Durante la traduzione dell'indirizzo, se il bit valido-non valido è 0 nell'elemento della tabella delle pagine ⇒ mancanza di pagina.

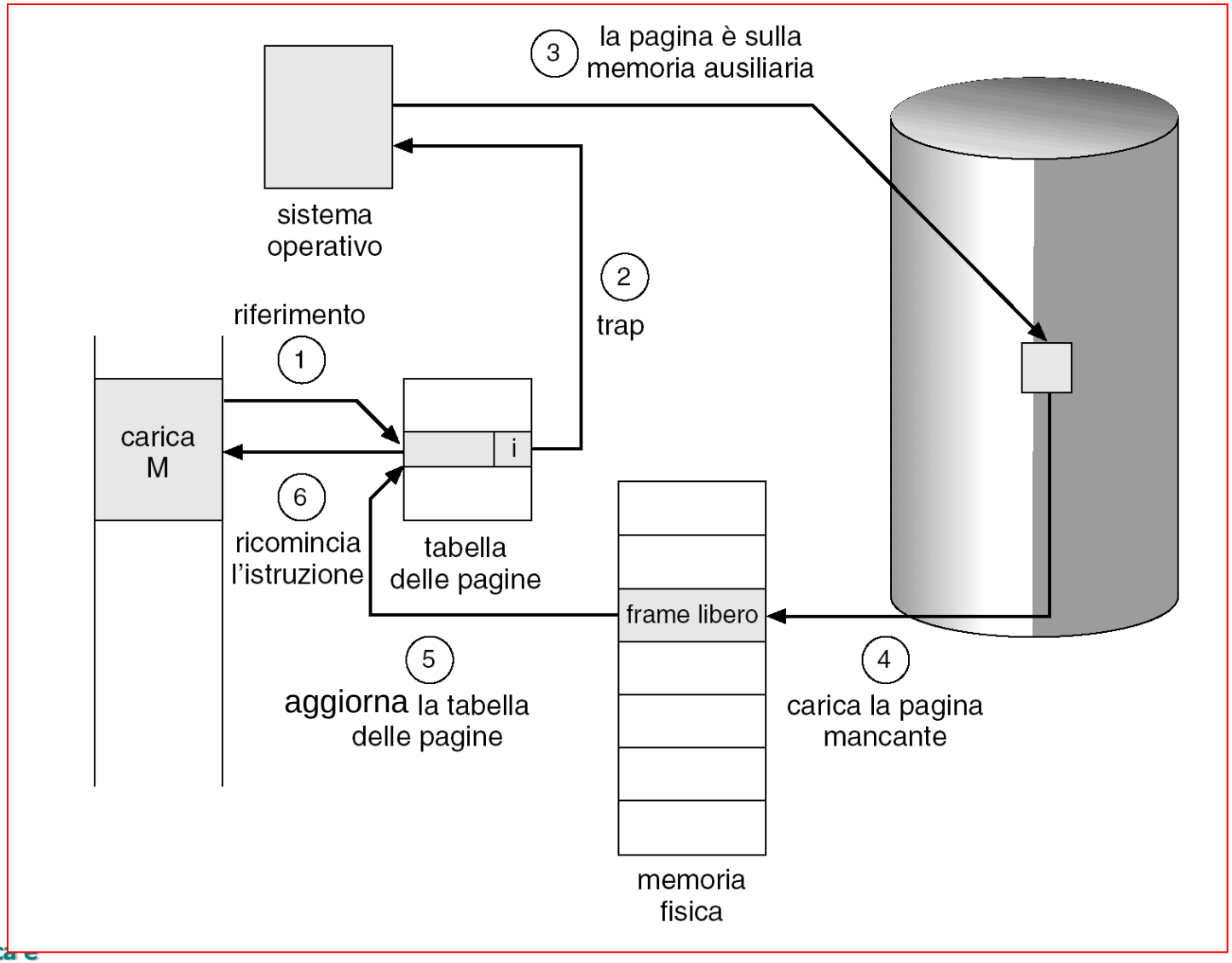
Tabella delle pegine quando alcune pagine non sono nella memoria centrale



Mancanza di pagina

- Se il processo accede ad una pagina contrassegnata come non valida ⇒ trap di mancanza di pagina (page fault).
- Il sistema operativo esamina un'altra tabella per decidere:
  - Riferimento non valido ⇒ termine del processo.
  - Solo non in memoria.
- Cercare un frame libero.
- Spostare la pagina nel frame.
- Modificare la tabella, validità del bit = 1.
- Fare ripartire l'istruzione:
  - block move
  - auto increment/decrement location

Passi necessari per trattare una mancanza di pagina



Cosa succede se non ci sono frame liberi?

- Sostituzione di pagina – trovare alcune pagine in memoria, ma non veramente in uso, e scambiarle di posto.
  - algoritmo
  - prestazione – L'algoritmo deve essere tale per cui venga minimizzato il numero di page fault futuri,
- La stessa pagina può essere portata in memoria più volte.

Prestazione della richiesta di paginazione

- Probabilità di mancanza di pagina $0 \leq p \leq 1.0$
  - se $p = 0$ non ci sono mancanze di pagina;
  - se $p = 1$, ogni riferimento è una mancanza di pagina.

- Tempo di accesso effettivo (EAT)

$$\begin{aligned} EAT = (1 - p) \times & \text{ accesso alla memoria} \\ & + p \text{ (mancanza di pagina} \\ & + [\text{scambio della pagina fuori }] \\ & + \text{scambio della pagina dentro} \\ & + \text{ripresa)} \end{aligned}$$

Esempio di domanda di paginazione

- Tempo di accesso alla memoria (MA) = 200 nanosecondi.
- **50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.**
- Tempo di swap della pagina = 8 msec = 8.000.000 nsec

$$EAT = (1 – p) \times 200 + p\ (8.000.000)$$

$$200 + 7.999.800\ P \qquad \text{(in nsec)}$$

Se p=0,001 allora

EAT = 8.200 nanosecondi (>40 volte MA!)

Processo di creazione

- La memoria virtuale permette altri benefici durante la creazione del processo:

  - Copia durante la scrittura

  - File mappati in memoria

Copia durante la scrittura

- La copia durante la scrittura permette ai processi padre e figlio di *condividere* inizialmente le stesse pagine in memoria.
- Se entrambi i processi scrivono in una pagina condivisa viene creata una copia della pagina condivisa.
- La copia durante la scrittura copia solo le pagine che sono state modificate da entrambi i processi.
- Le pagine libere sono allocate da *pool* di pagine *riempi con zero alla richiesta.*

File mappati in memoria

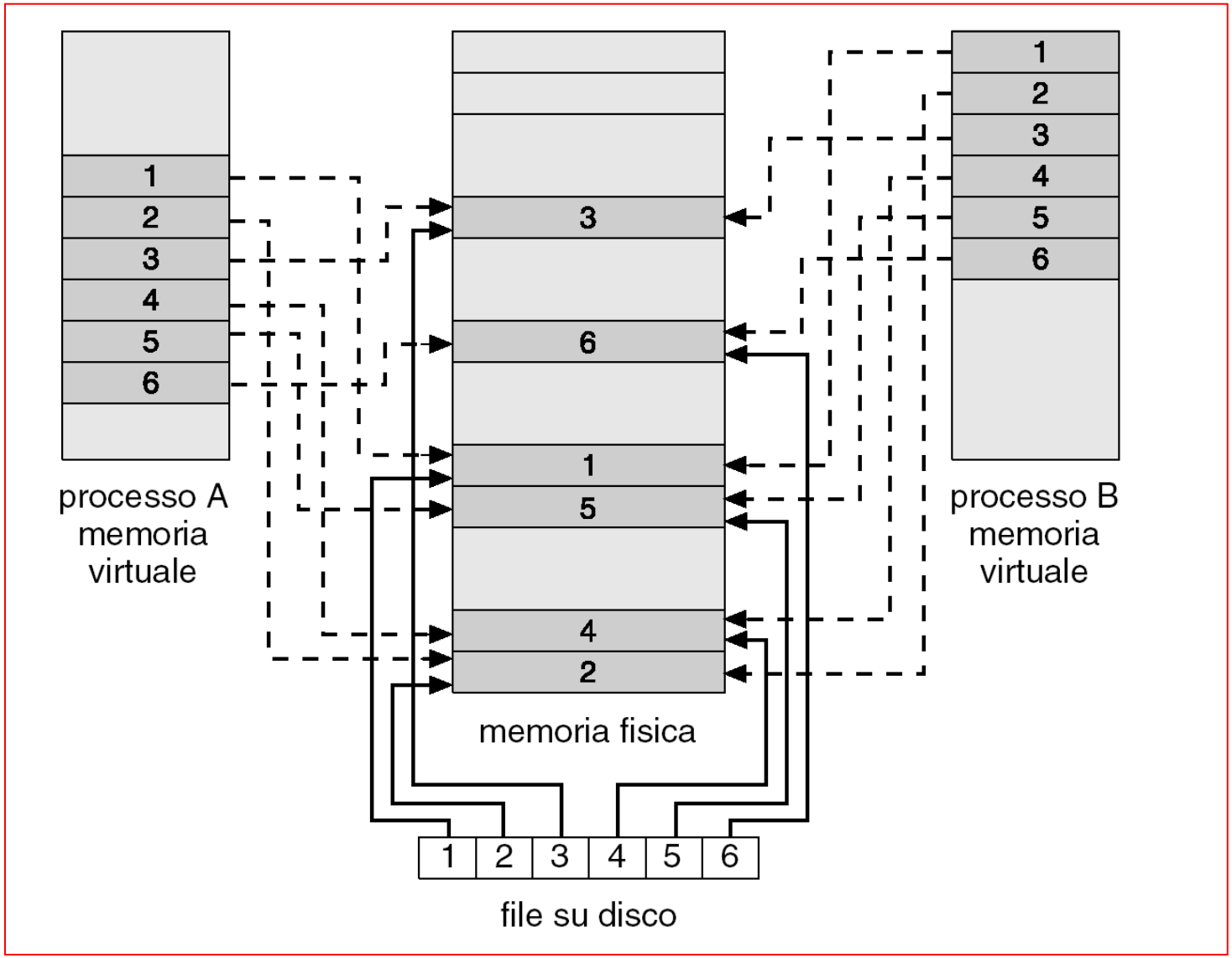
- Un file I/O mappato in memoria permette ad un file I/O di essere trattato come accesso alla procedura di gestione di memoria *mappando* un blocco del disco ad una pagina in memoria centrale.

- L'accesso iniziale al file procede con la richiesta di paginazione. Una porzione del file delle dimensioni di una pagina viene letta dal file system in una pagina fisica. Le successive letture e scritture del file sono trattate come accessi alle procedure di gestione della memoria.

- Alcuni sistemi operativi forniscono una mappatura della memoria solo con una chiamata specifica di sistema come read() write() e usano le chiamate standard di sistema per effettuare tutte le operazioni di I/O sul file.

- I processi possono avere il permesso di mappare lo stesso file in modo concorrente, per consentire la condivisione dei dati.

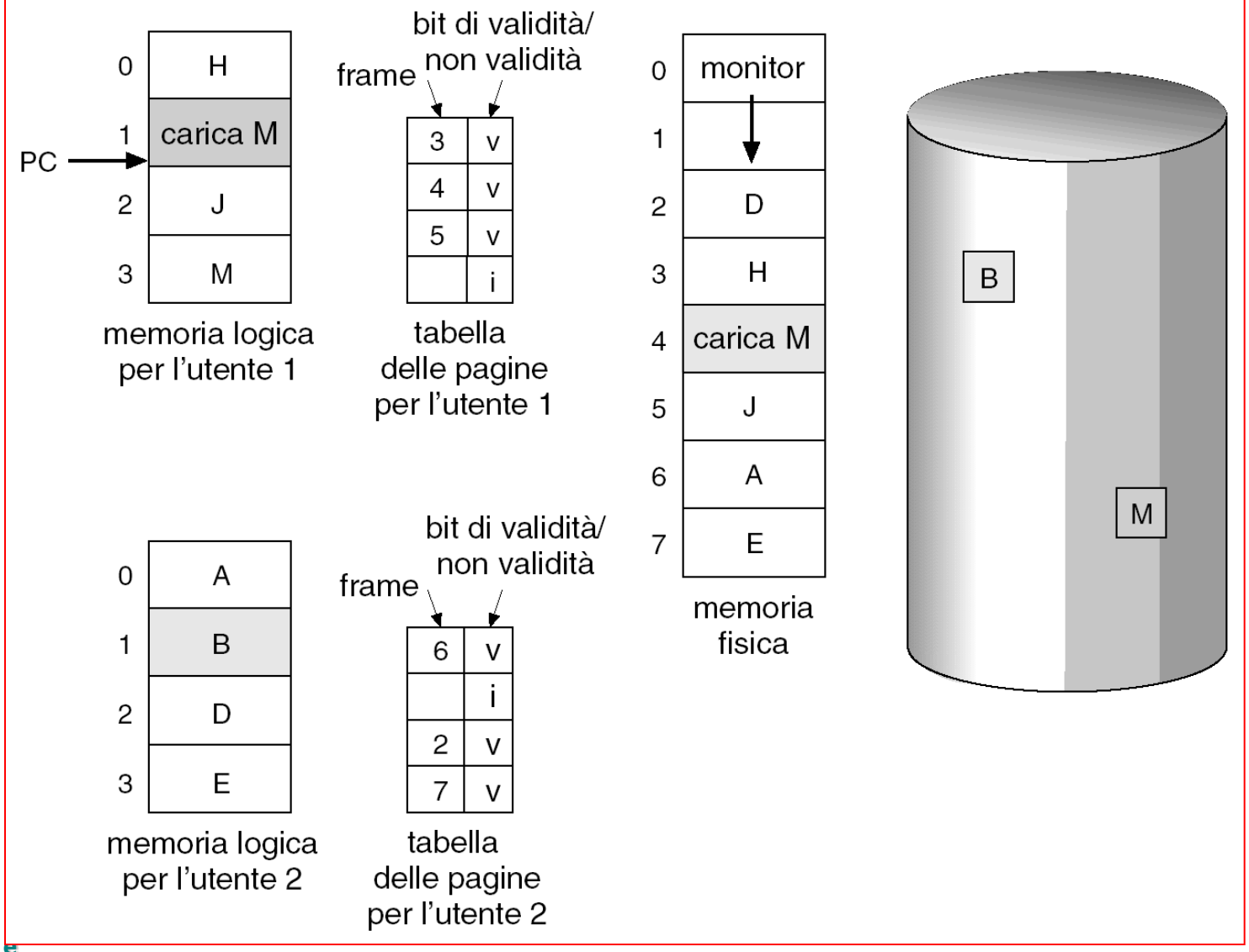
File mappati in memoria (Cont.)



Sostituzione della pagina

- Evitare la sovra-allocazione della memoria attraverso la procedura della mancanza di pagina che include la sostituzione di pagina.
- Usare un *bit di modifica* (modify bit) per ridurre il dispendio di trasferimento di pagine – solo le pagine modificate sono scritte su disco.
- La sostituzione della pagina completa la separazione fra memoria logica e memoria fisica – si può fornire un'enorme memoria virtuale su una memoria fisica più piccola.

Necessità di sostituzione della pagina



Sostituzione di base della pagina

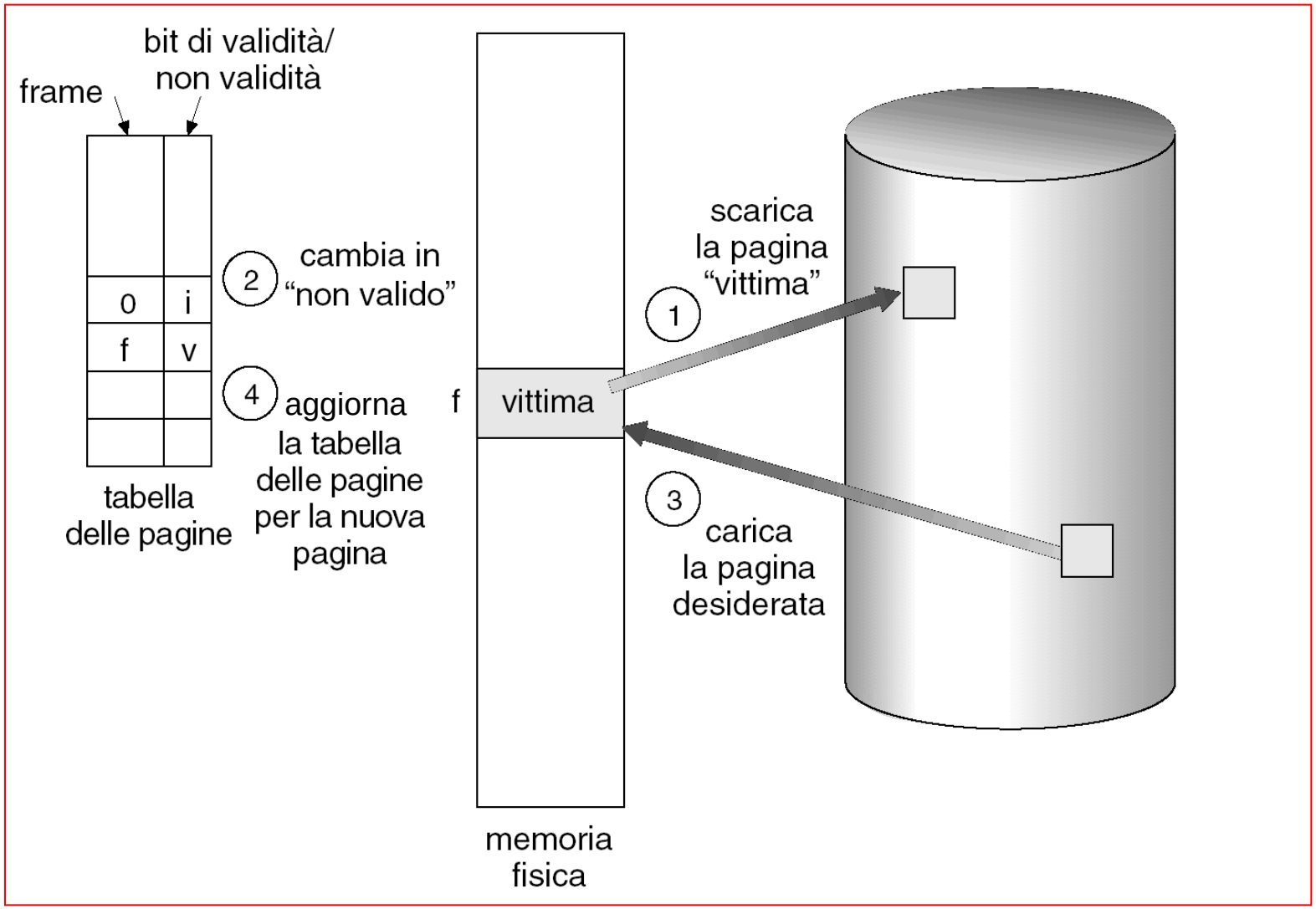
1. Trovare la posizione della pagina desiderata sul disco.

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

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

4. Riprendere il processo.

Sostituzione dalla pagina

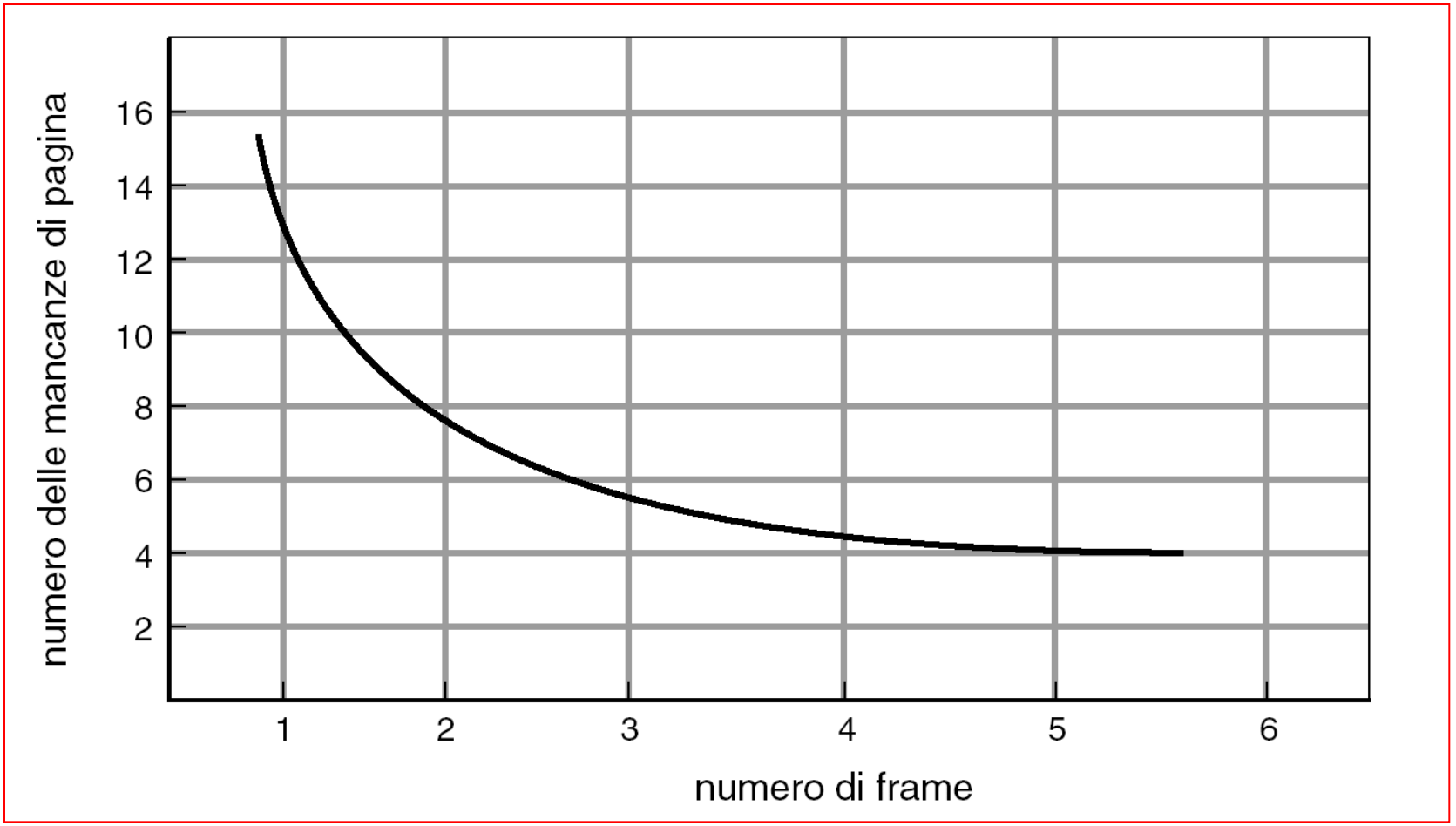


Algoritmo di sostituzione della pagina

- Desiderare il più basso tasso di mancanza di pagine.
- Un algoritmo viene valutato facendolo operare su esempi di riferimento di sequenze di richieste di pagine e calcolando il numero di page fault.
- In tutti i nostri esempi la stringa di riferimento è:

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

Grafico delle mancanze di pagina in funzione del numero di frame



Algoritmo FIFO

- Stringa di riferimento: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- 3 frame (in memoria possono esserci per processo 3 pagine allo stesso tempo).

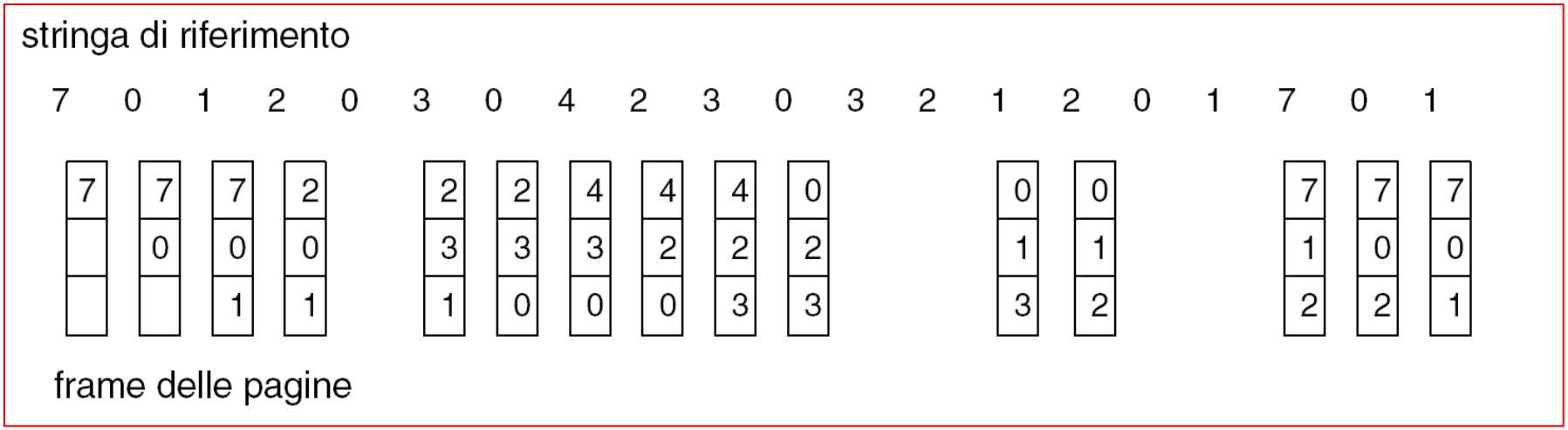
1	1	4	5	
2	2	1	3	9 mancanze di pagina
3	3	2	4	

- 4 frame

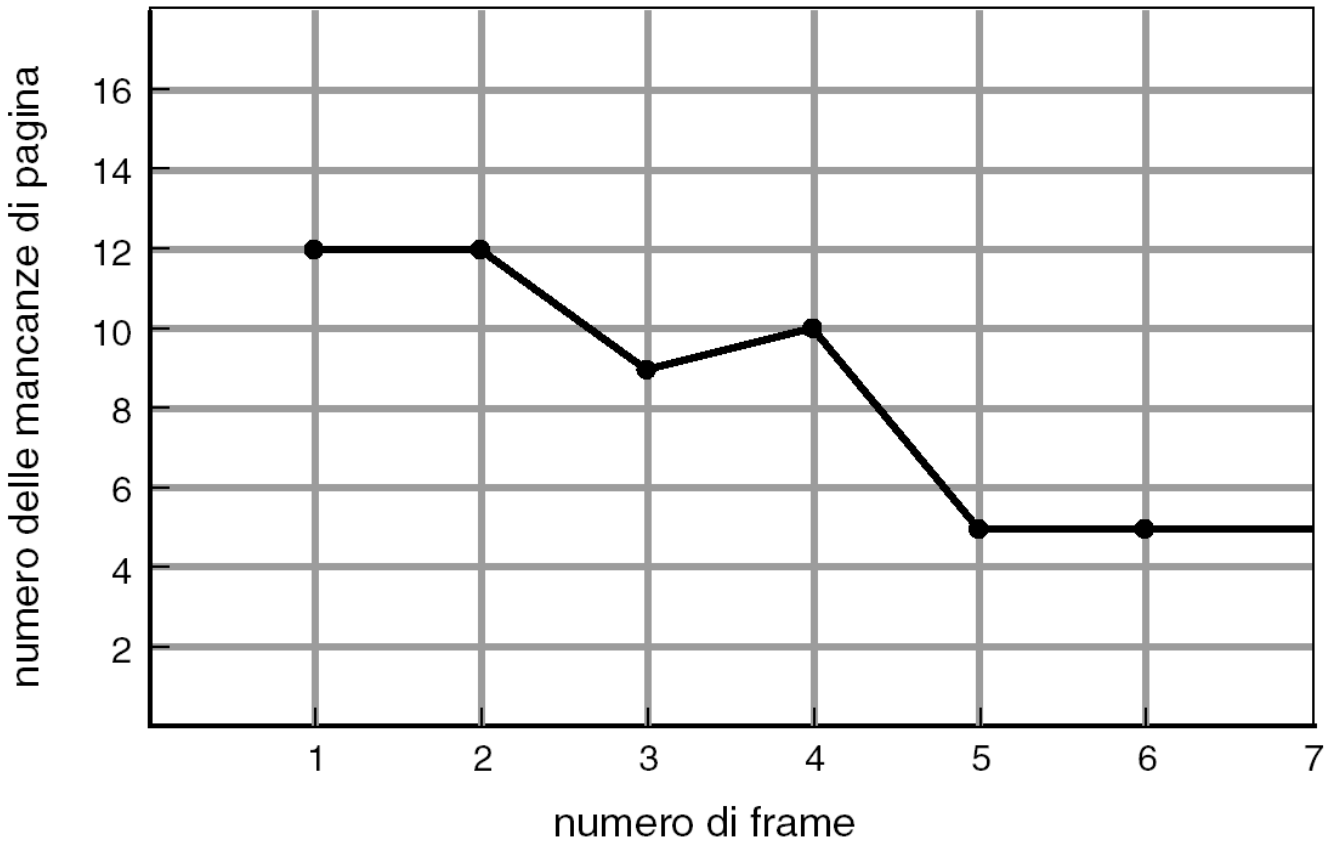
1	1	5	4	
2	2	1	5	10 mancanze di pagina
3	3	2		
4	4	3		

- Sostituzione FIFO – Anomalia di Belady
  - più frames $\Rightarrow$ meno mancanze di pagina.

Sostituzione FIFO della pagina



Sostituzione FIFO che ilustra l'anomalia di Belady



Algoritmo ottimale

- Sostituire la pagina che non sarà usata per il più lungo periodo di tempo.
- 4 esempi di frame:

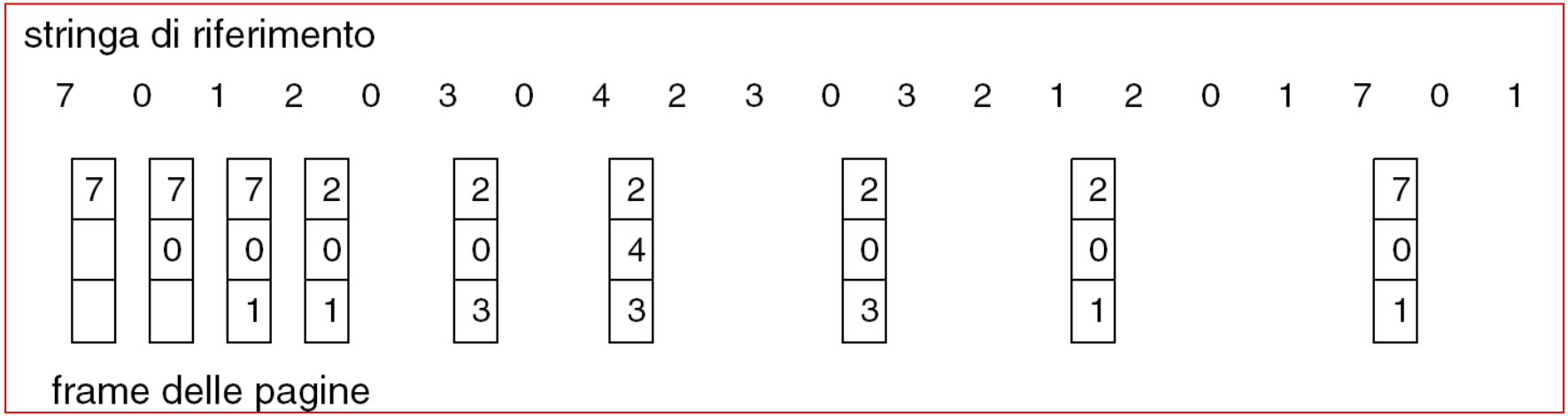
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	4
2	
3	
4	5

6 mancanze di pagina

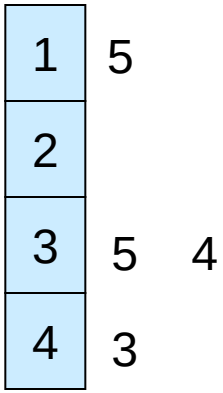
- Come sapete questo?
- Usato per misurare quanto sono buone le prestazioni dell'algoritmo.

Sostituzione ottimale della pagina



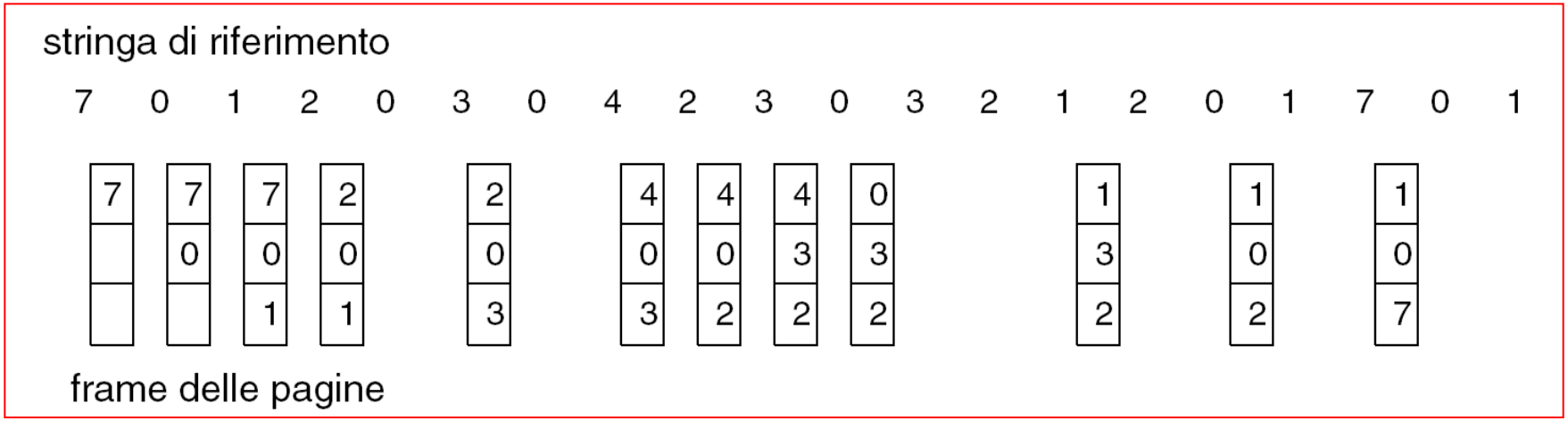
Algoritmo LRU

- Stringa di riferimento: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.



- Implementazione del contatore.
  - Ogni pagina ha un contatore; ogni volta che si fa riferimento ad una pagina il contenuto del registro dell'orologio viene copiato nel campo del tempo d'uso della tabella delle pagine per quella pagina.
  - Quando una pagina deve essere cambiata si guarda il contatore per determinare quali devono essere cambiati.

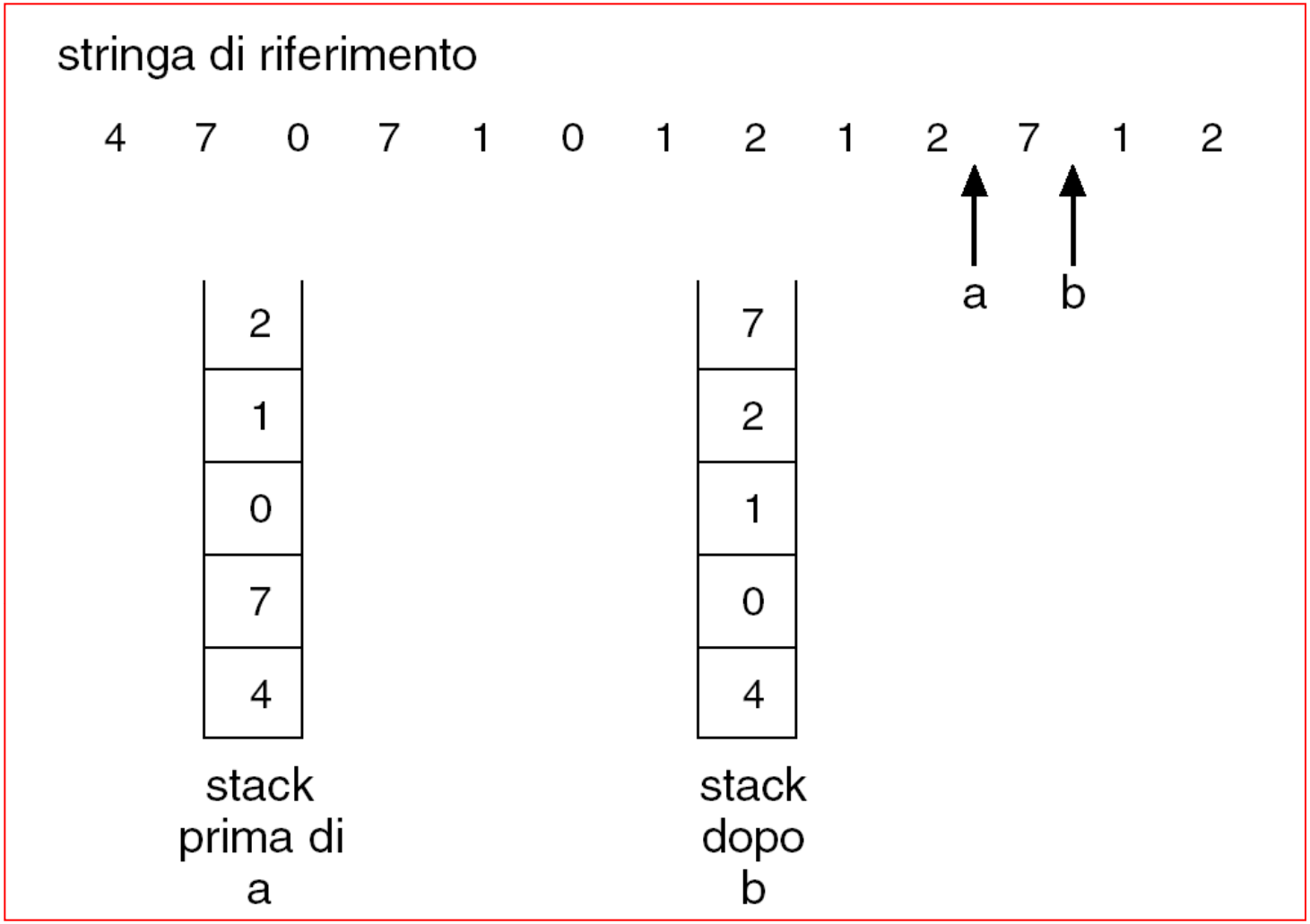
Sostituzione LRU della pagina



Algoritmo LRU (Cont.)

- Implementazione dello stack – mentenere uno stack dei numeri di pagina in una lista a doppio collegamento:
  - Riferimento ad una pagina:
    - mettere la pagina in cima allo stack;
    - richiede di cambiare 6 puntatori.
  - Nessuna ricerca per la sostituzione.

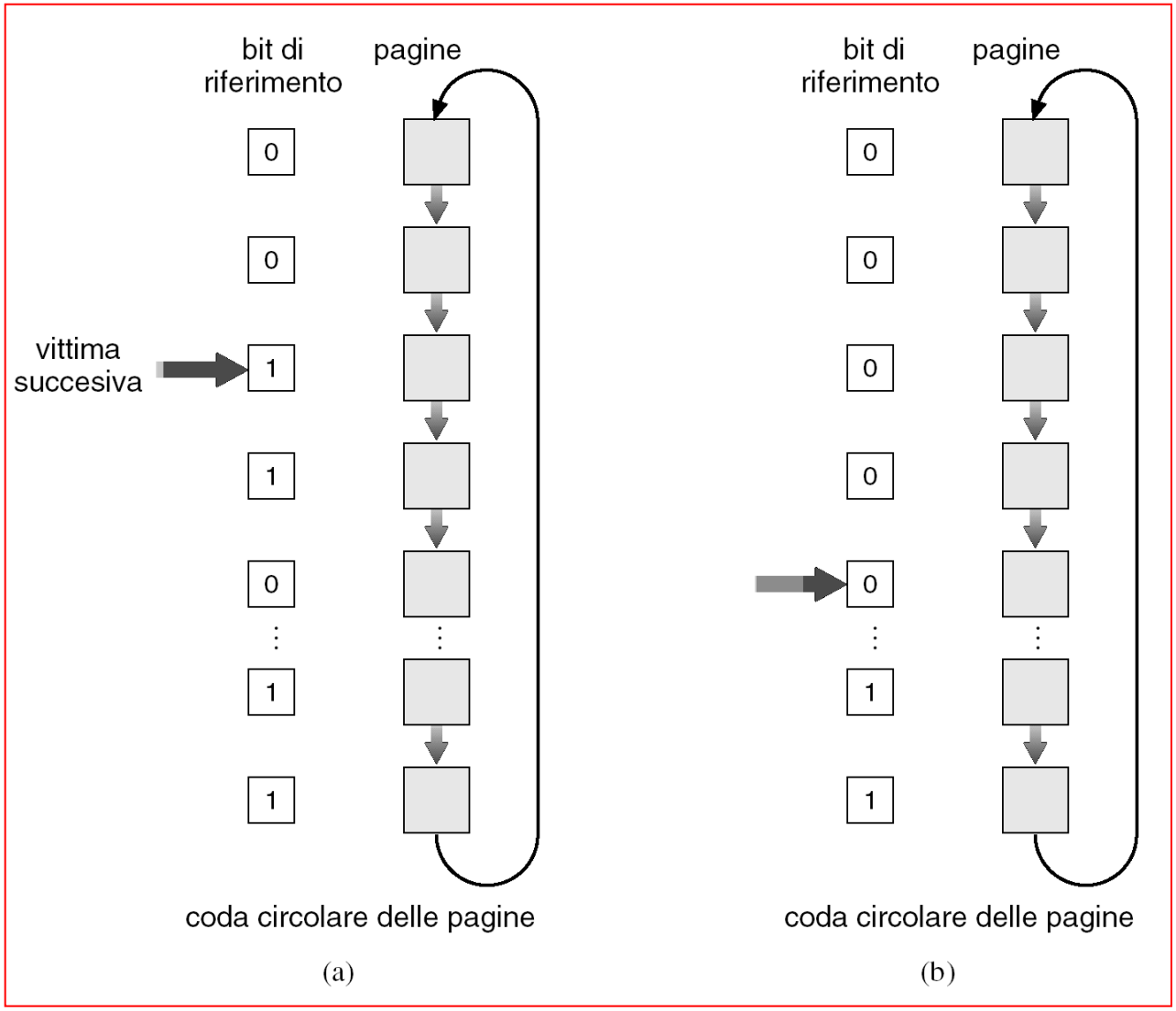
Uso di uno stack per registrare i riferimenti alle pagine usate più di recente



Approssimazione dell'algoritmo LRU

- 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). Non se ne conosce l'ordine tuttavia.
- Seconda possibilità.
  - Bisogno di un bit di riferimento.
  - Le pagine sono disposte in una lista circolare
  - Quando occorre selezionare una pagina vittima comincio una scansione lungo la lista:
    - se una pagina ha il bit di riferimento a 1 lo pongo a zero e passo alla successiva (la pagina rimane in memoria);
    - altrimenti la seleziono per essere sostituita.

Algoritmo della seconda possibilità (orologio) per la sostituzione dela pagina



Algoritmo di conteggio

- Tenere un contatore del numero di riferimenti che sono stati fatti ad ogni pagina.
- Algoritmo LFU: sostituisce la pagina con il più basso conteggio.
- Algoritmo MFU: basato sul fatto che la pagina con il conteggio più basso è stata probabilmente appena portata dentro e deve ancora essere usata.

Allocazione dei frame

- Ogni processo ha bisogno di un numero **minimo** di pagine.
- Esempio: IBM 370 – **6 pages to handle SS MOVE instruction:**
  - **instruction is 6 bytes, might span 2 pages.**
  - **2 pagine to handle from.**
  - **2 pagine to handle to.**
- Due principali schemi di allocazione.
  - Allocazione fissa.
  - Allocazione a priorità.

Allocazione fissa

- Allocazione omogenea – per esempio, se 100 frame e 5 processi, ognuno prende 20 pagine.
- Allocazione proporzionale – si assegna la memoria disponibile ad ogni processo in base alle dimensioni di quest'ultimo.
  - $s_i = \text{size of process } p_i$
  - $S = \sum s_i$
  - $m = \text{total number of frames}$

$$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$$

$$m = 64$$

$$s_i = 10$$

$$s_2 = 127$$

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

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

Allocazione a priorità

- Usare uno schema di allocazione proporzionale che usa le priorità piuttosto che la dimensione.

- Se il processo P_i genera una mancanza di pagina,
  - selezionare per la sostituzione uno dei suoi frame.
  - selezionare per la sostituzione un frame da un processo con un numero di priorità basso.

Allocazione globale e locale

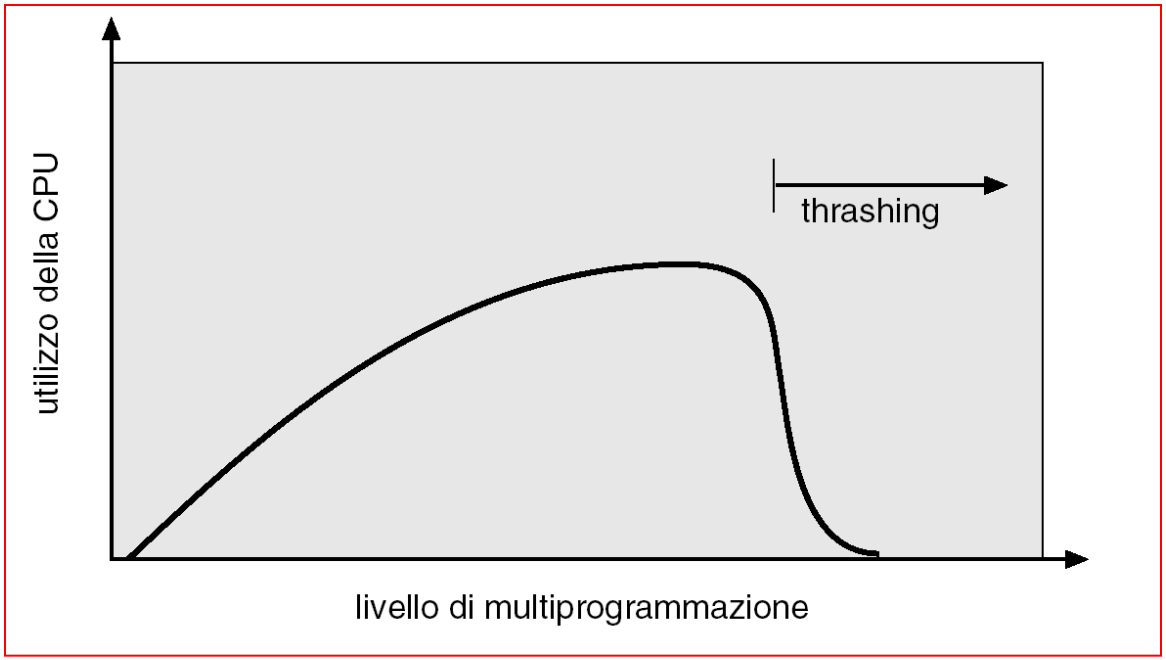
- **Sostituzione globale** – permette ad un processo di selezionare un frame di sostituzione a partire dall'insieme di tutti i frame,anche se quel frame è correntemente allocato a qualche altro processo – un processo può prendere un frame da un altro.
- **Sostituzione locale** – ogni processo effettua la scelta solo nel proprio insieme di frame allocati.

Thrashing

- Se un processo non ha abbastanza pagine, il tasso di mancanza di pagina è molto alto. Questo comporta:
  - basso utilizzo della CPU;
  - il sistema operativo ritiene che sia necessario aumentare il livello di multiprogrammazione;
  - un altro processo aggiunto al sistema.

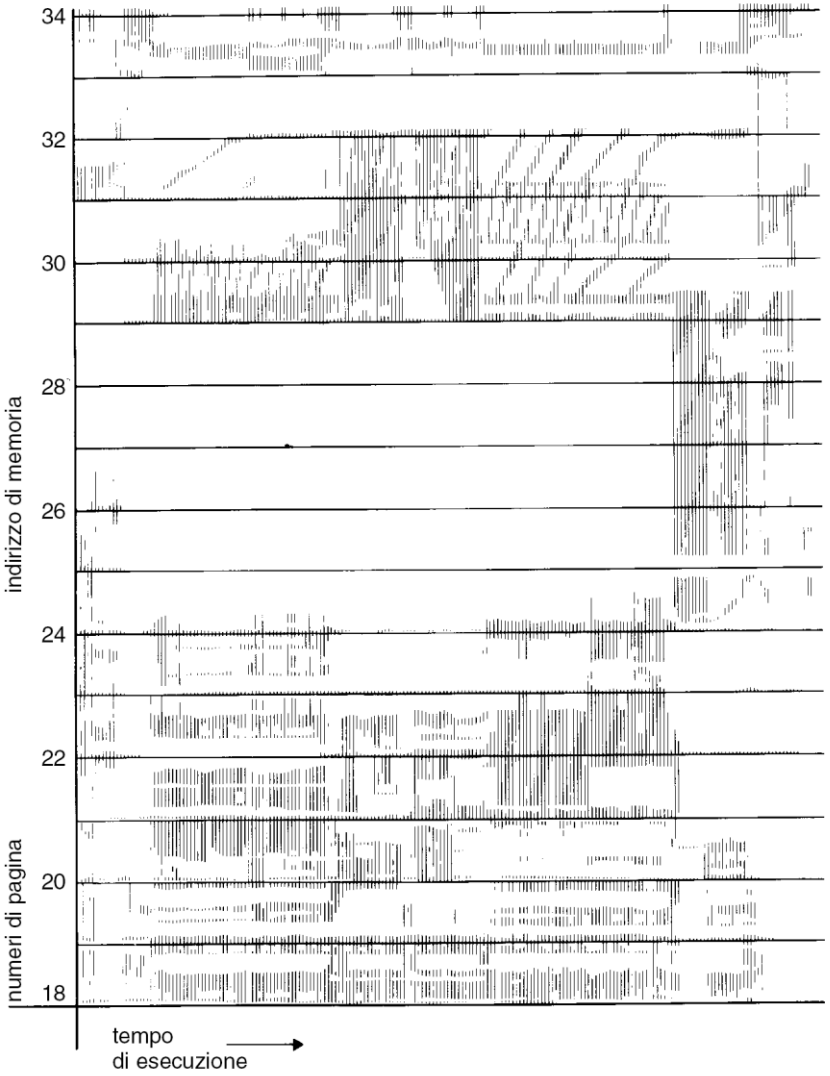
- **Thrashing** ≡ un processo spende più tempo nella paginazione che nella propria esecuzione.

Thrashing (Cont.)



- Perchè la paginazione lavora?
  Modello di località:
  - il processo si muove da una località all'altra,
  - le località possono sovrapporsi.
- Perchè si verifica il trashing?
  Σ dimensione della località > dimensione totale della memoria.

Località in un modello di riferimento alla memoria



Il modello working set

- $\Delta \equiv$ finestra di working-set $\equiv$ un numero fisso di riferimenti di pagina.
  Esempio: 10,000 istruzioni.
- WSS_i (working set del processo P_i) =
  numero totale di pagine con riferimenti 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 frames.
- Se $D > m \Rightarrow$ thrashing.
- Se $D >$ m, allora viene sospeso uno dei processi.

Il modello working set

tabella di riferimento delle pagine

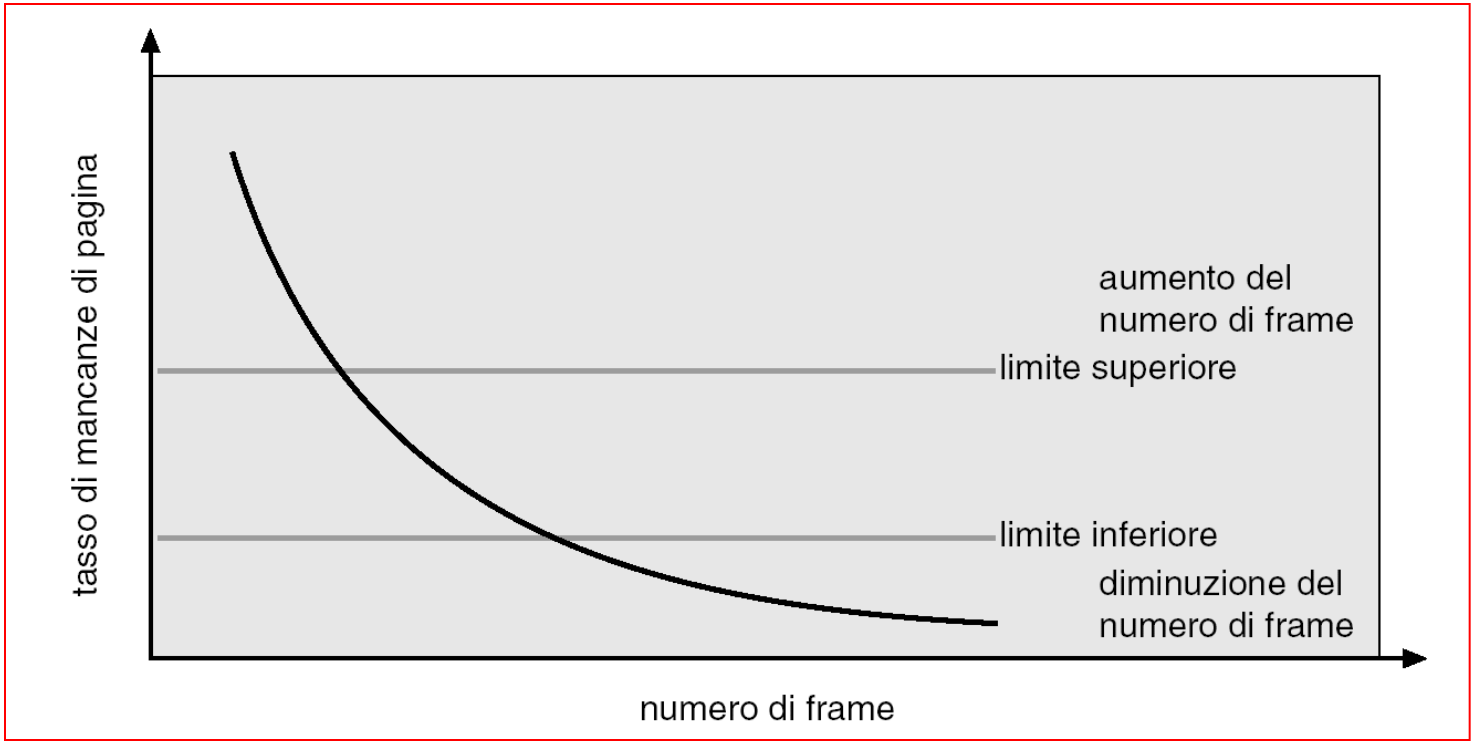
. . . 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 . . .

$WS(t_1) = \{1,2,5,6,7\}$ $WS(t_2) = \{3,4\}$

Mantenere traccia del working set

- Si può approssimare il modello working set con un interrupt di un temporizzatore a intervalli fissi di tempo e un bit di riferimento.
- Esempio: Δ = 10,000
  - Interrupt di temporizzatore ogni 5000 riferimenti.
  - Tenere in memoria 2 bit 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 in memoria = 1 $\Rightarrow$ la pagina è nel working set.
- Perchè non è del tutto preciso?
- Incremento = 10 bits e interrupt ogni 1000 riferimenti.

Frequenza delle mancanze di pagina



- Stabilire un tasso accettabile di mancanze di pagina:
  - Se il tasso attuale è troppo basso, il processo può avere troppi frames.
  - Se il tasso attuale è troppo alto, il processo ha bisogno di più frames.

Altre considerazioni

- Prepaginazione .
- La dimensione della pagina:
  - frammentazione,
  - dimensione della tabella delle pagine,
  - dispositivi di I/O,
  - località.

Altre considerazioni (Cont.)

- **Estensione della TLB** – quantità di memoria accessibile dalla TLB.

- Estensione della TLB = (dimensione TLB) X (dimensione pagina).

- Idealmente il working set per un processo è memorizzato nella TLB. Se non è così il processo utilizzerà un tempo considerevole per risolvere i riferimenti di memoria nella tabella delle pagine piuttosto che nella TLB.

Estensione della TLB

- **Incremento della dimensione della pagina**. Questo può portare ad un incremento della frammentazione se non tutte le applicazioni richiedono un dimensione grande di pagina.

- **Utilizzare pagine a multiple dimensioni**. Questo permette alle applicazioni che necessitano di pagine di maggiori dimensioni l'opportunità di usarle senza aumentare la frammentazione.

Altre considerazioni (Cont.)

- Struttura del programma:
  - **int A[][] = new int[1024][1024];**
  - Ogni riga è memorizzata in una pagina.
  - Programma 1

```
for (j = 0; j < A.length; j++)
    for (i = 0; i < A.length; i++)
        A[i,j] = 0;
```

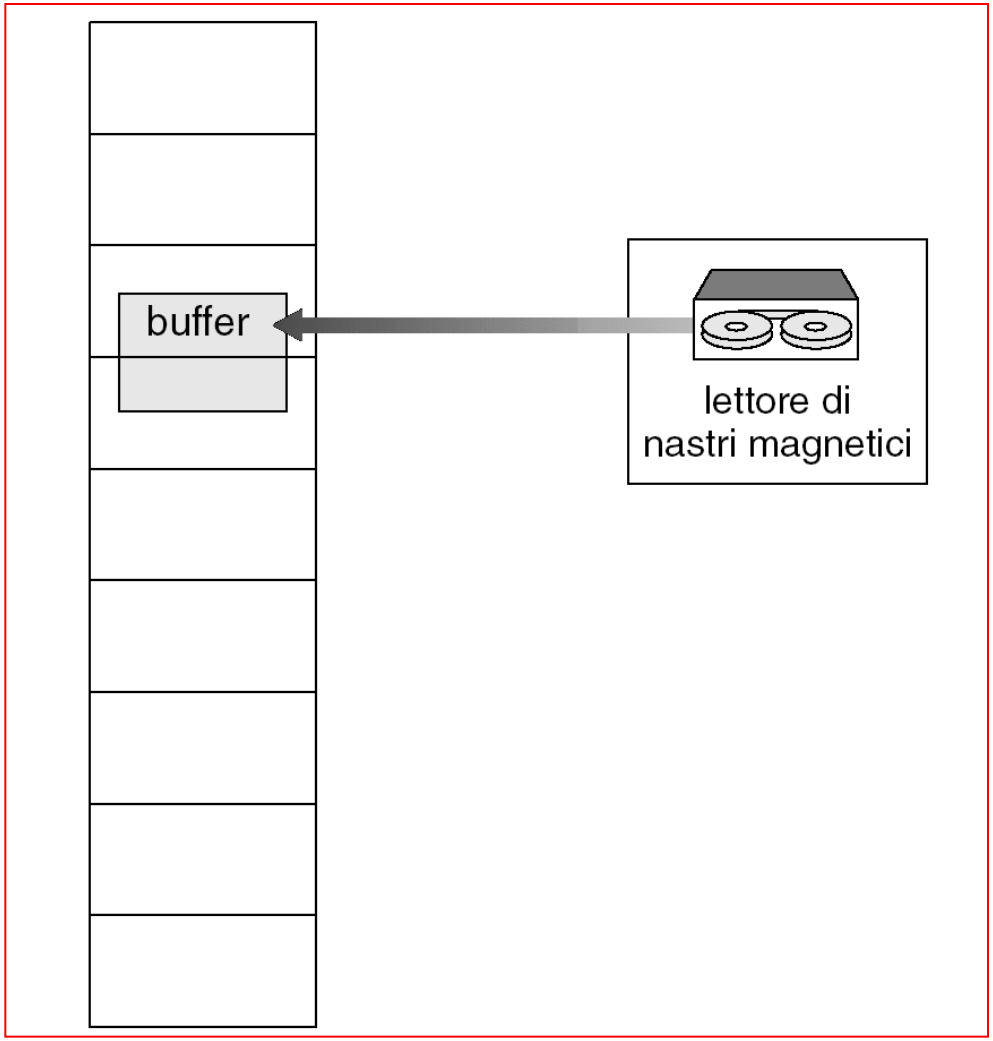
    1024 x 1024 mancanze di pagina.

  - Programma 2

```
for (i = 0; i < A.length; i++)
    for (j = 0; j < A.length; j++)
        A[i,j] = 0;
```

    1024 mancanze di pagina.

Ragione per cui i frame usati per operazioni di I/O devono essere in memoria.



Domanda di segmentazione

- Usata quando l'hardware è insufficiente per implementare la domanda di paginazione.
- OS/2 alloca la memoria in segmenti, che ne mantiene traccia attraverso i descrittori dei segmenti.
- Il descrittore di un segmento contiene un bit valido per indicare se il segmento è correntemente in memoria:
  - se il segmento è nella memoria principale, cotinua l'accesso;
  - se non è in memoria il segmento fallisce.