



# Capitolo 9: Memoria Virtuale

- Introduzione
- Paginazione su richiesta
- Copiatura su scrittura
- Sostituzione delle pagine
- Allocazione dei frame
- Attività di paginazione degenere (trashing)
- File mappati in memoria
- Allocazione di memoria del kernel
- Altre considerazioni
- Esempi tra i sistemi operativi



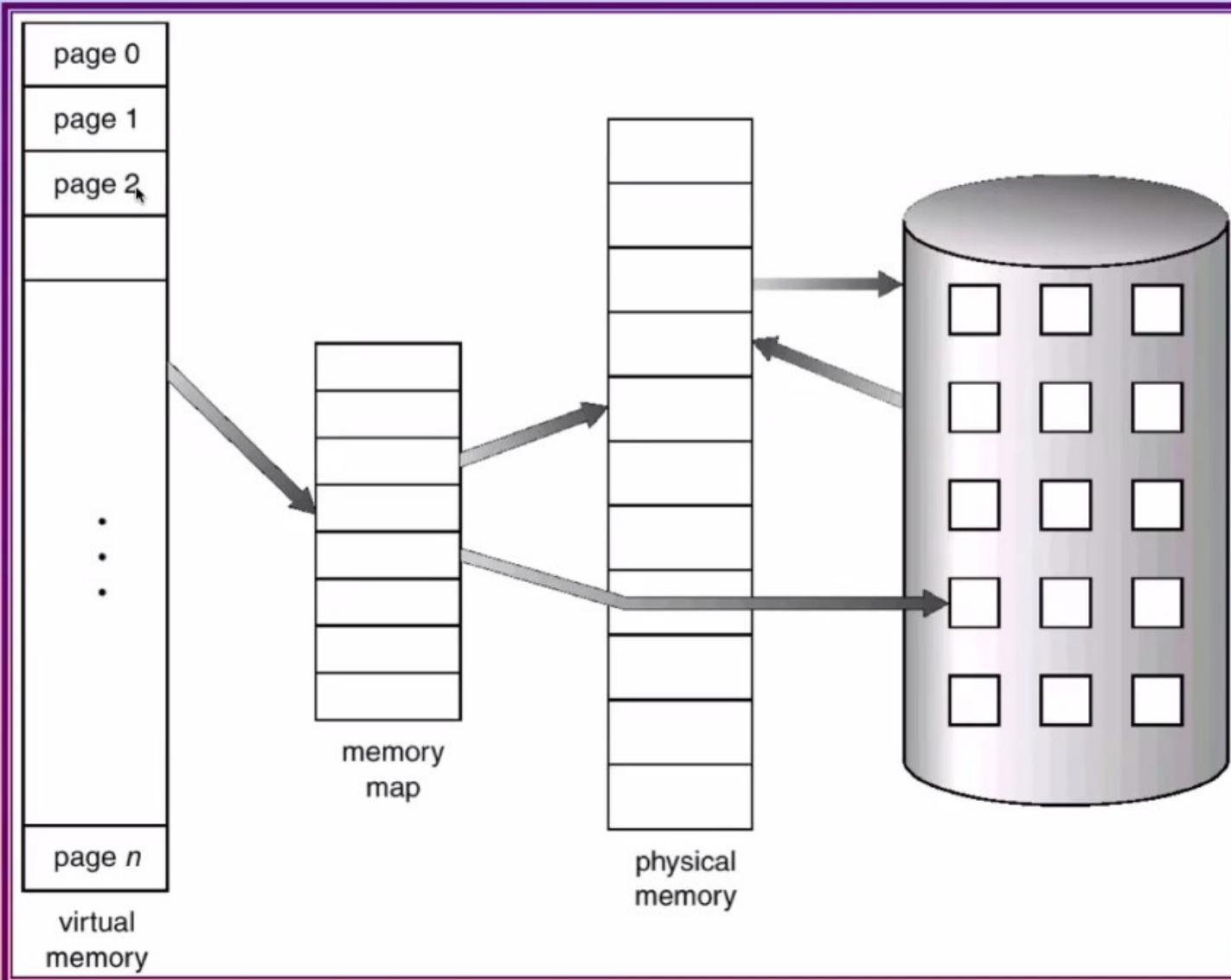


# Introduzione

- La memoria virtuale e' una tecnica che permette di eseguire processi che possono anche non risiedere completamente in memoria.
  - Permette la separazione della memoria logica dell'utente dalla memoria fisica:
    - ◆ Solo parte del programma viene caricato in memoria per essere eseguito.
    - ◆ Lo spazio di indirizzi logici può quindi essere più grande dello spazio degli indirizzi fisici.
    - ◆ Permette di condividere lo stesso spazio indirizzi tra più processi.
    - ◆ Permette tecniche più efficienti di creazione di processi.
  - La memoria virtuale puo' essere implementata tramite:
    - ◆ **paginazione su richiesta** (anziche' effettuare swapping sulla memoria di tutto il processo, si utilizza uno swapper *pigro*, che non riversa mai una pagina in memoria a meno che non sia necessaria - *pager*)
    - ◆ **segmentazione su richiesta**
- 



# Schema di memoria virtuale più grande della memoria fisica.





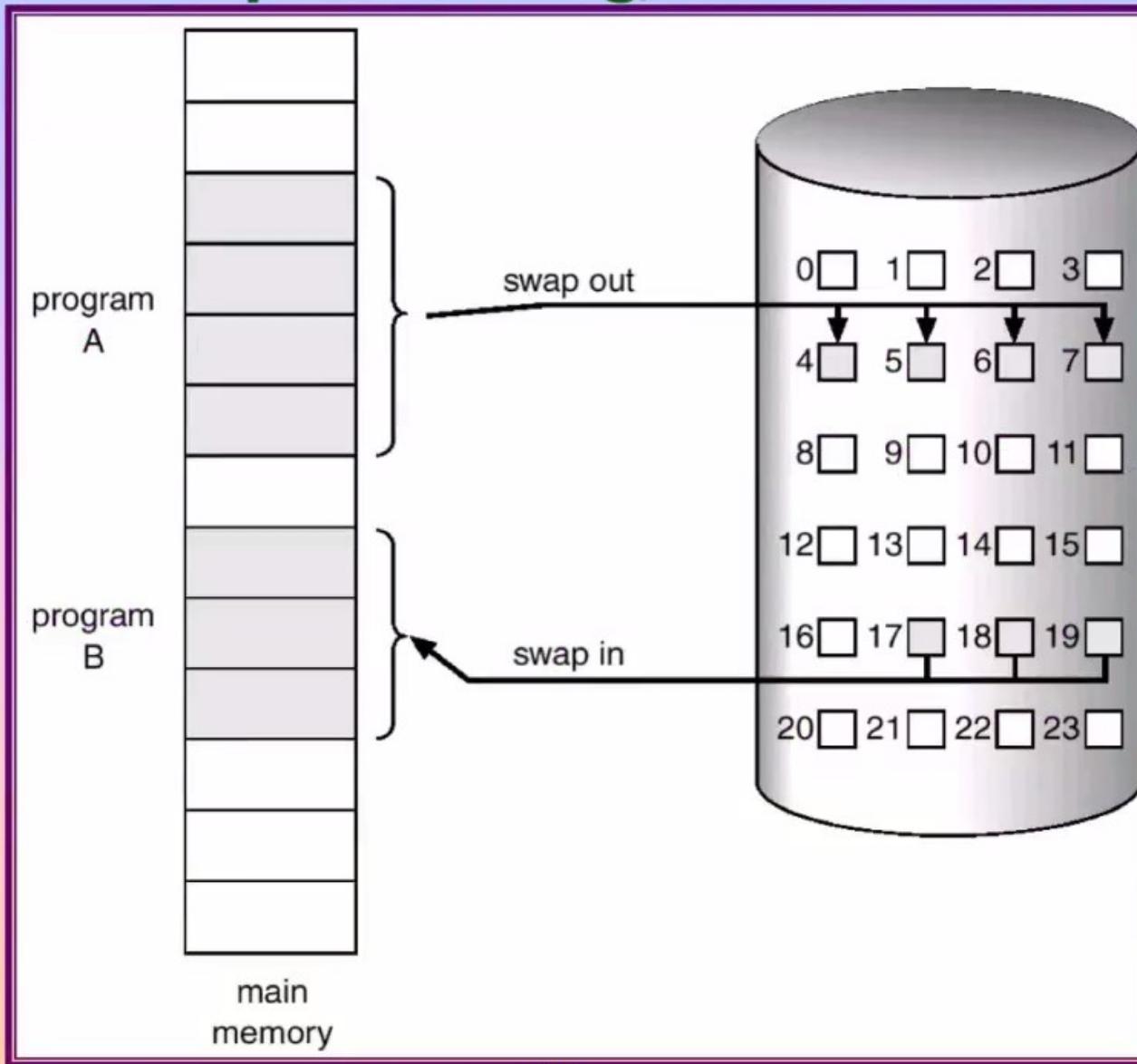
# Paginazione su richiesta

- La maggior parte dei sistemi operativi utilizza il metodo della paginazione su richiesta.
- Anziché caricare in memoria l'intero processo, come nella paginazione, si carica una pagina in memoria solo quando essa è necessaria.
  - ◆ Per caricare un programma utente in memoria è necessario un minore tempo di I/O (aumentando la velocità di esecuzione).
  - ◆ Nessun vincolo sulla quantità di memoria fisica necessaria ad un processo.
  - ◆ Aumento del grado di multiprogrammazione grazie al minore utilizzo della memoria fisica.
- Una pagina è necessaria.  $\Rightarrow$  c'è stato un riferimento ad essa.
- Se il processo tenta di accedere ad una pagina che non era stata caricata nella memoria, il sistema genera una *eccezione di pagina mancante* (page fault trap).
  - ◆ Il riferimento non era valido  $\Rightarrow$  si termina il processo (abort)
  - ◆ Il riferimento era valido  $\Rightarrow$  si porta la pagina in memoria





# Trasferimento di una memoria paginata nello spazio contiguo di un disco





# Bit di validità

- A ciascuna pagina è associato un bit di validità ( $1 \Rightarrow$  in memoria,  $0 \Rightarrow$  non in memoria), inizializzato in partenza a 0.
- Esempio di tabella delle pagine con bit di validità:

Frame #	valid-invalid bit
	1
	1
	1
	1
	0
:	
	0
	0

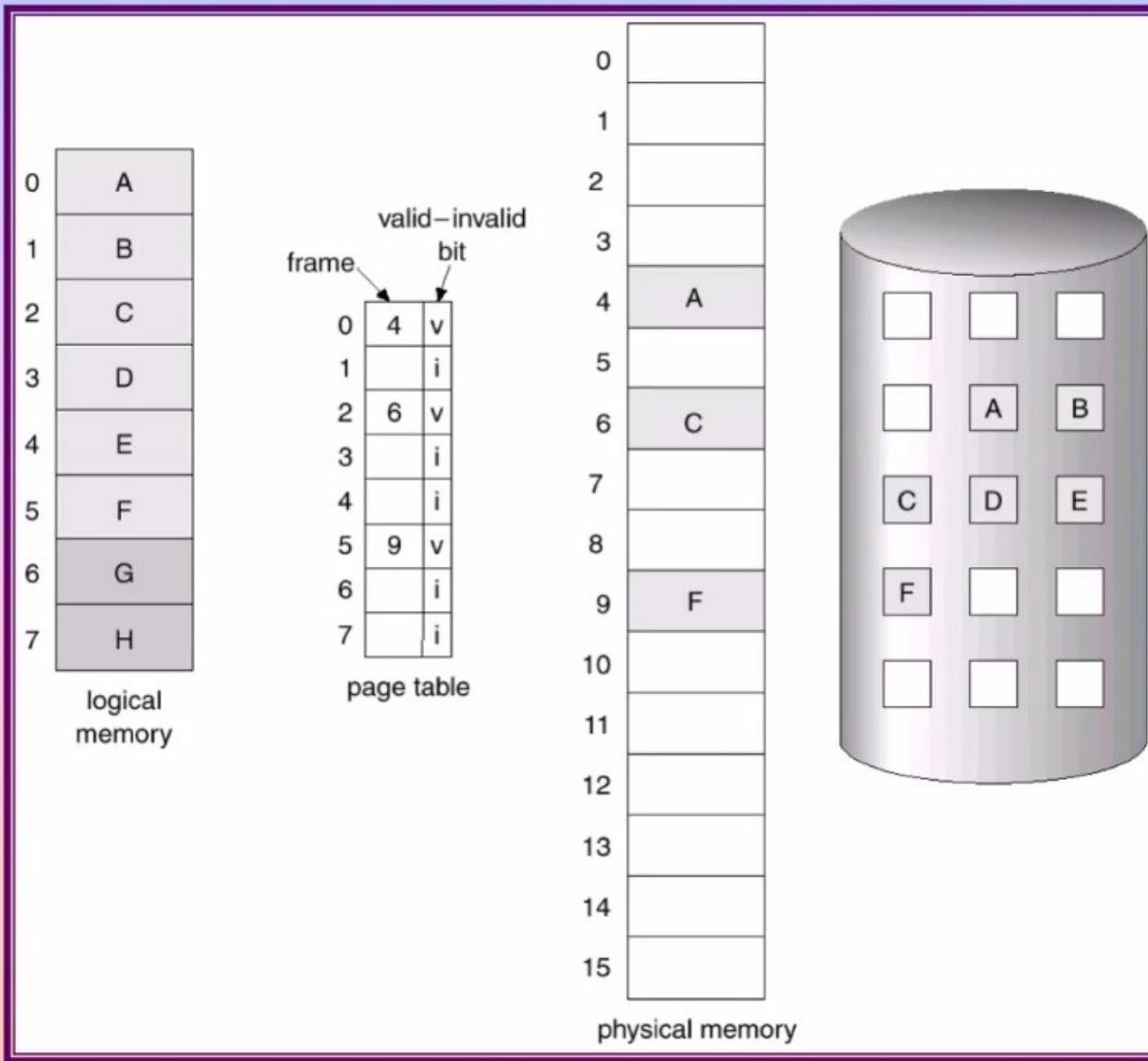
page table

- Se il processo tenta di accedere ad una pagina che non era stata caricata in memoria (bit di validità 0)  $\Rightarrow$  page fault.





# Tabella delle pagine quando alcune pagine non si trovano nella memoria centrale.





# Meccanismi di ausilio alla paginazione su richiesta

- *Tabella delle pagine*: questa tabella ha la capacità di contrassegnare un elemento come non valido attraverso un bit di validità oppure un valore speciale dei bit di protezione.
- *Memoria ausiliaria* : in generale il sistema operativo utilizza un file system ad alta velocità per la partizione di swap.
- Ad esempio il sistema operativo Linux utilizza una partizione distinta da quella del file system convenzionale in modo da velocizzare la gestione dei dati.
- La paginazione su richiesta può avere un effetto significativo sulle prestazioni generali di un sistema di calcolo.



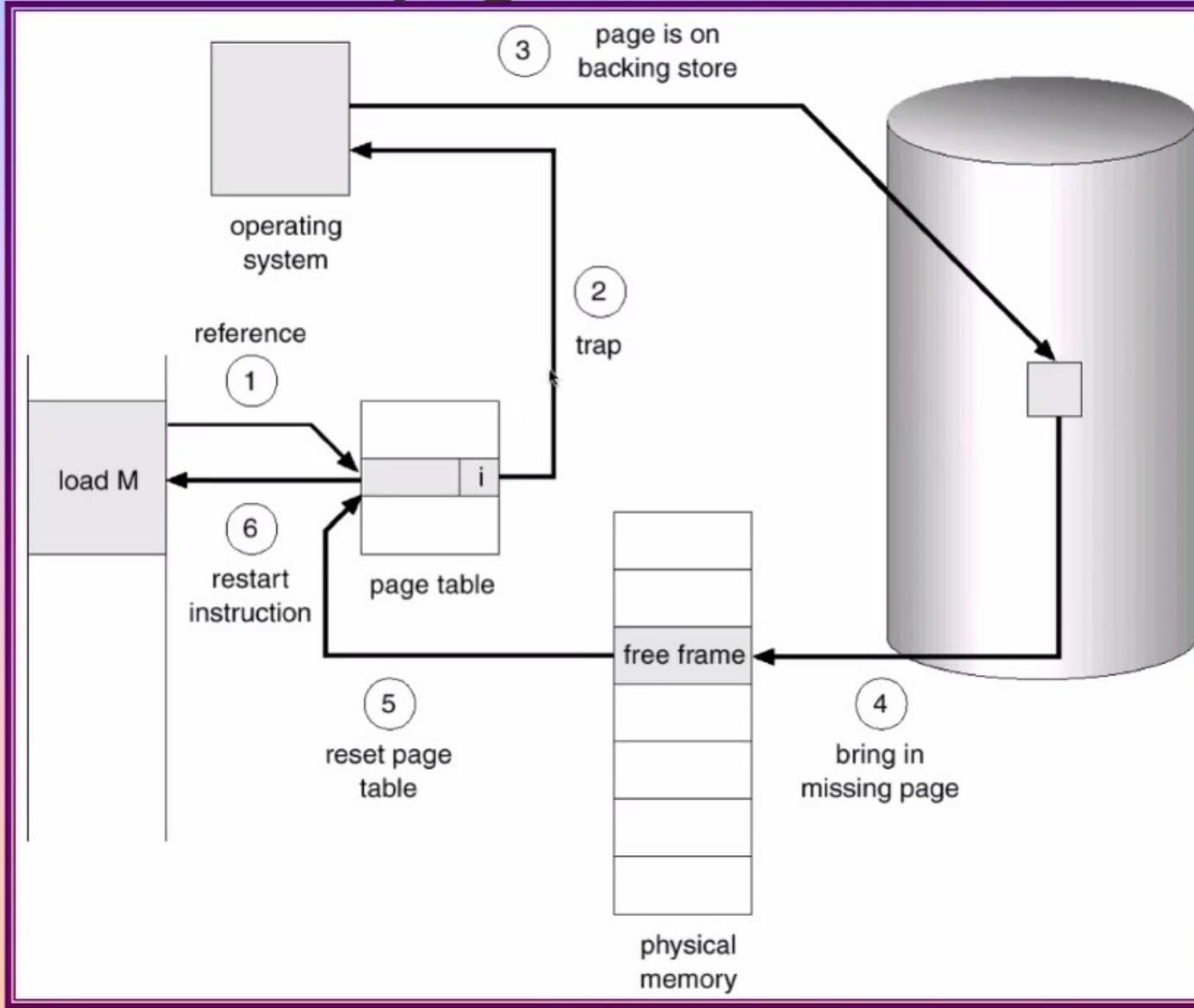


# Gestione del page fault

- Quando viene referenziata una pagina non in memoria, ci sarà un segnale di errore ⇒ page fault
- La procedura di gestione del page fault è suddivisa in sei fasi:
  - 1 Si controlla la tabella interna al processo per stabilire se il riferimento alla pagina e' valido o non valido.
  - 2 Se il riferimento non e' valido si termina il processo, altrimenti se e' valido ma la pagina non e' presente in memoria, se ne effettua l'inserimento.
  - 3 Viene individuato un frame libero, per esempio prelevandolo dall'elenco dei blocchi liberi.
  - 4 Si programma un'operazione sui dischi per trasferire la pagina desiderata nel blocco di memoria appena assegnato (swap in).
  - 5 A swap in terminato, si modifica la tabella interna conservata con il processo e la tabella delle pagine per indicare che la pagina e' disponibile nella memoria centrale.
  - 6 Si riavvia l' istruzione interrotta dal segnale di eccezione di indirizzo illegale. Il processo può accedere alla pagina.
- E' addirittura possibile avviare l'esecuzione di un processo senza caricare inizialmente alcuna pagina in memoria.



# Fasi di gestione di un'eccezione di pagina mancante





## Che succede se non c'è un frame libero?

- Rimpiazzamento di pagina – si cerca una pagina non utilizzata tra quelle in memoria, e la si memorizza nella memoria ausiliaria (swap out).
  - ◆ algoritmo
  - ◆ prestazioni – l' algoritmo dovrà minimizzare il numero di page fault futuri.
- Quindi alcune pagine dovranno essere portate in memoria più volte.





# Prestazioni della paginazione su richiesta

- La paginazione su richiesta può avere un effetto rilevante sulle prestazioni di un calcolatore.
- Tempo di accesso alla memoria  $\Rightarrow ma$  (normalmente varia da 10 a 200 nanosecondi finché non si verifica un page fault)
- Probabilità di una assenza di pagina (page fault rate):  $0 \leq p \leq 1.0$ 
  - ◆ se  $p = 0$  nessun page fault
  - ◆ se  $p = 1$ , ogni riferimento genera un page fault
- Tempo di accesso effettivo (Effective Access Time - EAT)  
$$EAT = ((1 - p) \times ma) + (p \times \text{tempo di gestione dell'assenza di pagina})$$
- Per calcolare il tempo d'accesso effettivo occorre conoscere il tempo necessario alla gestione di un'assenza di pagina.





# Gestione di una assenza di pagina

- 1 Segnale di eccezione al sistema operativo.
- 2 Salvataggio dei registri utente e dello stato del processo.
- 3 Determinazione della natura di eccezione di pagina mancante del segnale di eccezione.
- 4 Controllo della correttezza del riferimento alla pagina e determinazione della locazione della pagina nel disco
- 5 Lettura dal disco e trasferimento in un blocco di memoria libero:
  - a) attesa nella coda del dispositivo fino a che la richiesta di lettura non è servita
  - b) attesa del tempo di posizionamento e latenza del dispositivo
  - c) inizio del trasferimento della pagina in un blocco di memoria libero
  - d) durante l'attesa, assegnazione della CPU ad un altro utente
  - e) segnale di interruzione emesso dal controllore del disco (I/O completato)
  - f) salvataggio dei registri e dello stato dell'altro processo utente





## Gestione di una assenza di pagina (II)

- 6 Determinazione della provenienza dal disco dell'interruzione.
- 7 Aggiornamento della tabella delle pagine e di altre tabelle per segnalare che la pagina richiesta è in memoria.
- 7 Attesa che la CPU sia nuovamente assegnata a questo processo.
- 8 Recupero dei registri utente, dello stato del processo e della nuova tabella delle pagine, quindi ripresa dell'istruzione interrotta.





# Prestazioni della paginazione su richiesta: un esempio

- Considerando un tempo medio di servizio dell'eccezione di pagina mancante di 25 millisecondi e un tempo di accesso alla memoria di 100 nanosecondi, il tempo effettivo d'accesso in nanosecondi è  
$$\begin{aligned} EAT &= (1 - p) \times 100 + p \times (25 \text{ millisecondi}) = \\ &= (1 - p) \times 100 + p \times 25.000.000 = \\ &= 100 + 24.999.900 \times p \end{aligned}$$
- Il tempo d'accesso effettivo è direttamente proporzionale alla frequenza delle assenze di pagina.
- Ad es. se un accesso su 1000 accusa un'assenza di pagina, il tempo d'accesso effettivo è di 25 microsecondi.
- Se si desidera un rallentamento inferiore al 10%:  $p < 0,0000004$ .
- Cioè per mantenere a un livello ragionevole il rallentamento dovuto alla paginazione, si può permettere un'assenza di pagina ogni 2.500.000 accessi alla memoria.





# Creazione ed esecuzione di processi

- La memoria virtuale offre altri vantaggi per quel che riguarda la creazione ed esecuzione dei processi.
- Copiatura su scrittura (Copy-on-Write):
  - ◆ Questa tecnica permette una creazione dei processi molto rapida e minimizza il numero di pagine che si devono assegnare al nuovo processo
- Associazione dei file alla memoria (Memory-Mapped Files):
  - ◆ Questa tecnica semplifica e velocizza gli accessi al file system trattando l'I/O su file tramite la paginazione in memoria centrale, invece che tramite le chiamate a sistema di **read()** e **write()**.





# Copiatura su scrittura (copy-on-write)

- *fork* crea un processo figlio come duplicato del genitore.
- Nella versione originale *fork* creava per il figlio una copia dello spazio di indirizzi del genitore, duplicando le pagine appartenenti al processo genitore.
- Considerando che spesso ad una *fork* segue una *exec*, questa copiatura risulta inutile.
- La tecnica di *copiatura su scrittura* (*Copy-on-Write - COW*) permette sia al processo genitore che al processo figlio di condividere inizialmente le stesse pagine di memoria.
- Se uno dei due processi modifica una pagina, solo allora la pagina è duplicata.
- COW permette una creazione di processi più efficiente e rapida.
- Quando una pagina deve essere duplicata, si attinge ad un pool di pagine libere utilizzate anche per gestire l'ingrandimento di strutture come pile o *heap* di un processo, e che vengono prima riempite di zeri e poi assegnate (*azzeramento su richiesta*)





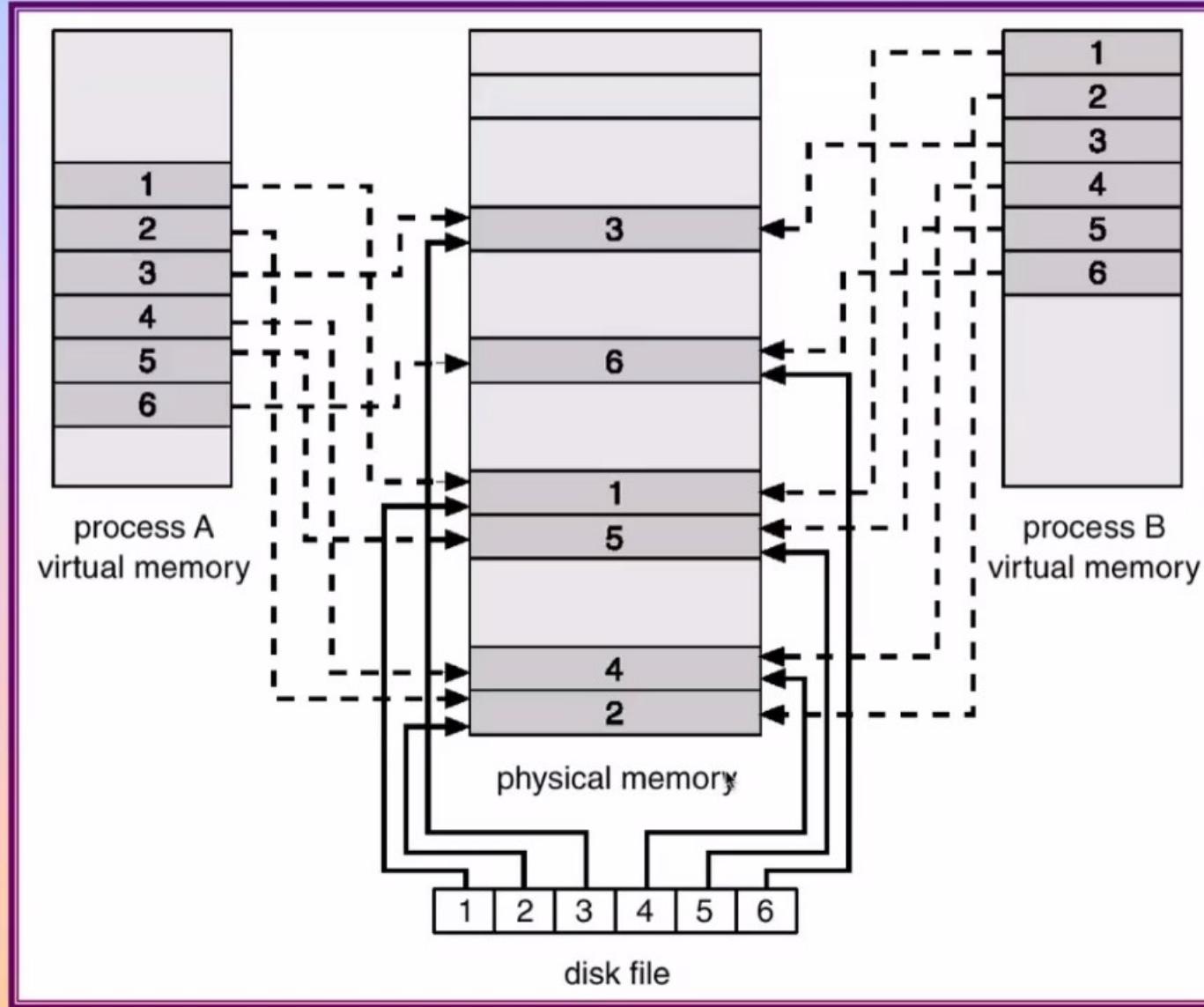
# Associazione dei file alla memoria

- Consideriamo un accesso sequenziale in lettura o scrittura ad un file su disco,
  - ◆ ogni operazione di lettura o di scrittura necessiterà di una chiamata al sistema e di un accesso al disco.
- L' **associazione alla memoria** permette di trattare l' I/O di file su disco come un generico accesso a memoria
  - ◆ facendo corrispondere un blocco del disco ad una pagina (o più pagine) di memoria.
- L'accesso a file avviene inizialmente tramite l'ordinario meccanismo di paginazione su richiesta,
  - ◆ cui segue, però, il caricamento in una pagina fisica di una porzione di file della dimensione di una pagina, leggendola dal file system.
- Le operazioni successive di lettura o scrittura si gestiscono come normali accessi alla memoria.
- Questa tecnica permette inoltre l'associazione dello stesso file alla memoria virtuale di più processi allo scopo di condividere dati.
- Le chiamate a sistema per l'associazione alla memoria possono anche disporre della funzione di copiatura su scrittura,
  - ◆ permettendo ai processi di condividere un file per la lettura, mantenendo però una propria copia dei dati da essi modificati.





# Associazione dei file alla memoria: un esempio





# Sostituzione delle pagine

- In presenza di una page fault dovuto ad un riferimento ad una pagina non in memoria, quindi, si dovrà portare la pagina in memoria.
- Se nessun blocco di memoria è libero, si potrà liberarne uno inutilizzato,
  - ◆ scrivendo il suo contenuto nell'aria di avvicendamento e modificando la tabella delle pagine (e le altre tabelle) per indicare che quella pagina non è più in memoria.
- Il blocco di memoria liberato si potrà usare per caricare la pagina che ha causato l'eccezione.
- La procedura di servizio dell'eccezione di pagina mancante verrà modificata in modo da includere l'eventuale sostituzione della pagina.
- L'architettura fisica del sistema di calcolo può disporre di un *bit di modifica* (*modify - dirty bit*), associato ad ogni pagina che mette a 1 automaticamente ogni volta che la pagina viene modificata.
- Solo le pagine che sono state modificate vengono riscritte su disco.
- La sostituzione delle pagine completa la separazione tra memoria logica e memoria fisica: possiamo avere una memoria virtuale molto ampia con una memoria fisica più piccola.



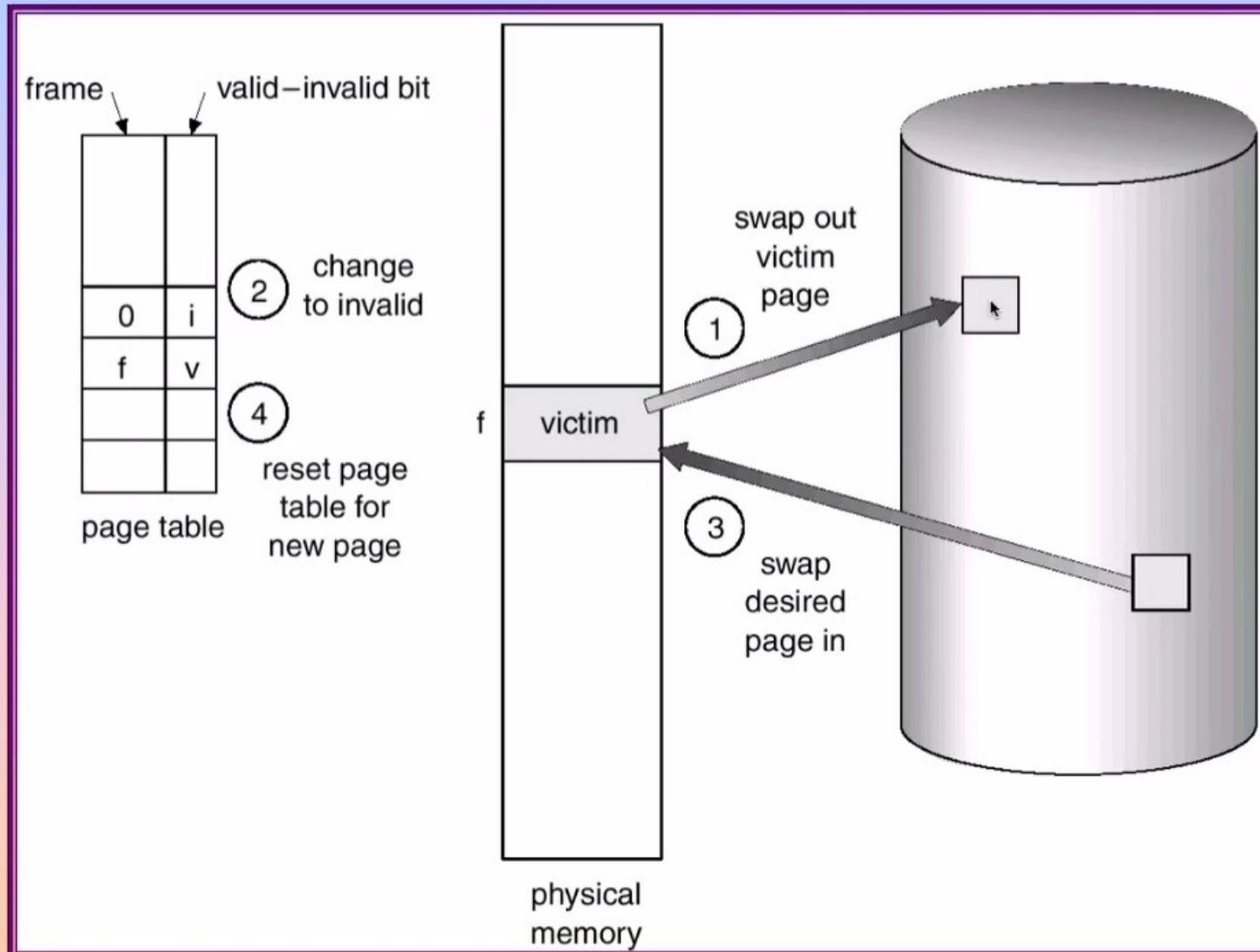


# Schema di base

- La procedura di servizio dell'eccezione di pagina mancante deve modificata in modo da includere l'eventuale sostituzione della pagina:
  - 1- Si individua la locazione nel disco della pagina richiesta
  - 2- Si cerca un blocco di memoria libero:
    - a) se esiste, lo si usa;
    - b) altrimenti si impiega un algoritmo di sostituzione delle pagine per scegliere un blocco di memoria "vittima";
    - c) si scrive la pagina "vittima" nel disco; si modificano adeguatamente le tabelle delle pagine e dei blocchi di memoria;
  - 3- Si scrive la pagina richiesta nel blocco di memoria appena liberato e si modificano le tabelle delle pagine e dei blocchi di memoria;
  - 4- Si riavvia il processo utente.
- Esistono diversi algoritmi usati per la sostituzione delle pagine.
- Un algoritmo di solito viene valutato effettuandone l'esecuzione su di una particolare stringa di riferimenti e calcolando il numero di page fault che esso causa.

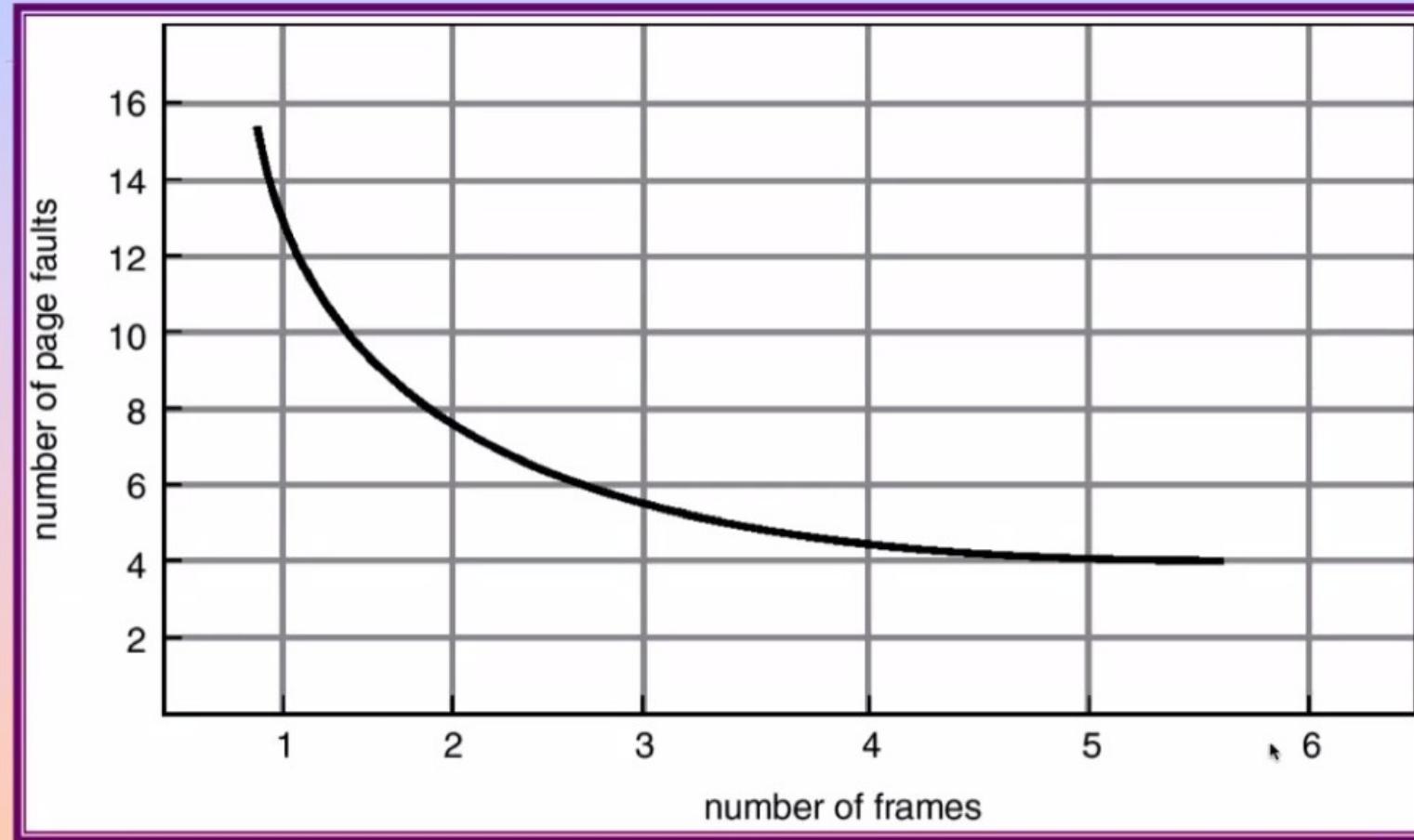


# Sostituzione di una pagina





# Numero di assenze di pagina rispetto al numero di frame





# Algoritmi di sostituzione delle pagine

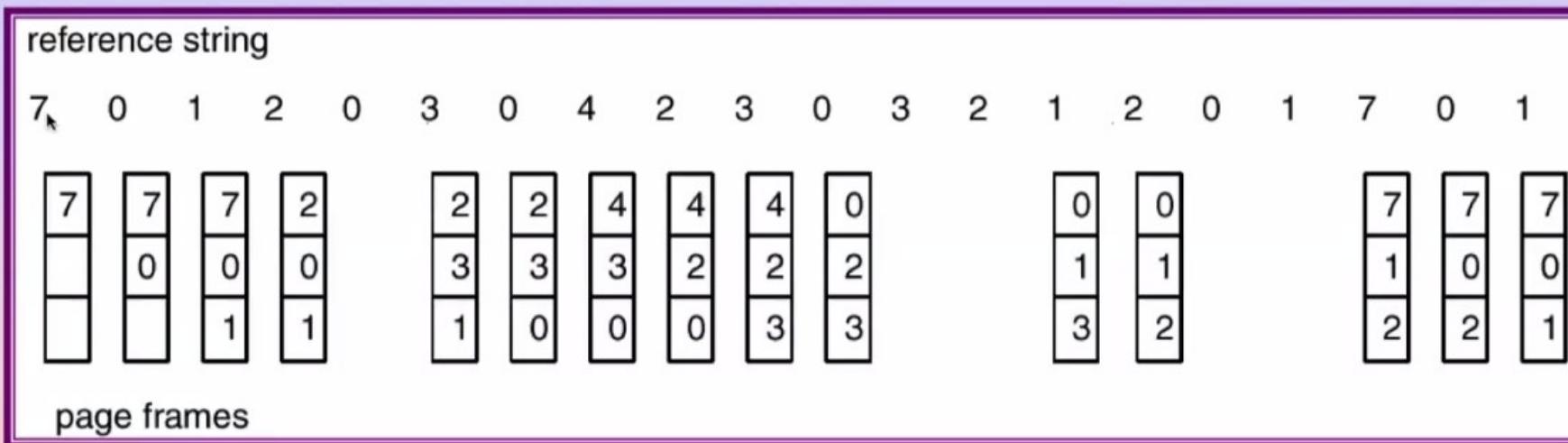
- Per realizzare la paginazione su richiesta è necessario risolvere due problemi principali:
  - ◆ algoritmo di allocazione dei frame
  - ◆ algoritmo di sostituzione delle pagine
- Esistono molti algoritmi di sostituzione delle pagine,
  - ◆ probabilmente ogni SO ha un proprio schema di sostituzione.
- Scopo:
  - ◆ minimizzare il tasso di page-fault!
- Valutazione di un algoritmo:
  - ◆ va fatta effettuandone l'esecuzione su una particolare successione di riferimenti alla memoria e calcolando il numero di page fault
- Queste successioni si possono generare artificialmente oppure derivano dall'analisi di un sistema reale.
- Negli esempi seguenti, consideriamo, per una memoria con tre frame, la seguente successione di riferimenti:
  - ◆ 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1





# Sostituzione delle pagine secondo l'ordine di arrivo (FIFO)

- L'algoritmo di sostituzione delle pagine più semplice è un algoritmo FIFO in cui se si deve sostituire una pagina si sostituisce quella presente in memoria da più tempo.



- Nell'esempio si hanno complessivamente 15 assenze di pagina.
- Problema: la pagina sostituita potrebbe essere ormai inutile
  - potrebbe invece essere ancora utile: ad esempio se contiene una variabile molto usata e quindi sostituirla potrebbe generare successivi page fault.





# First-In-First-Out (FIFO)

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

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

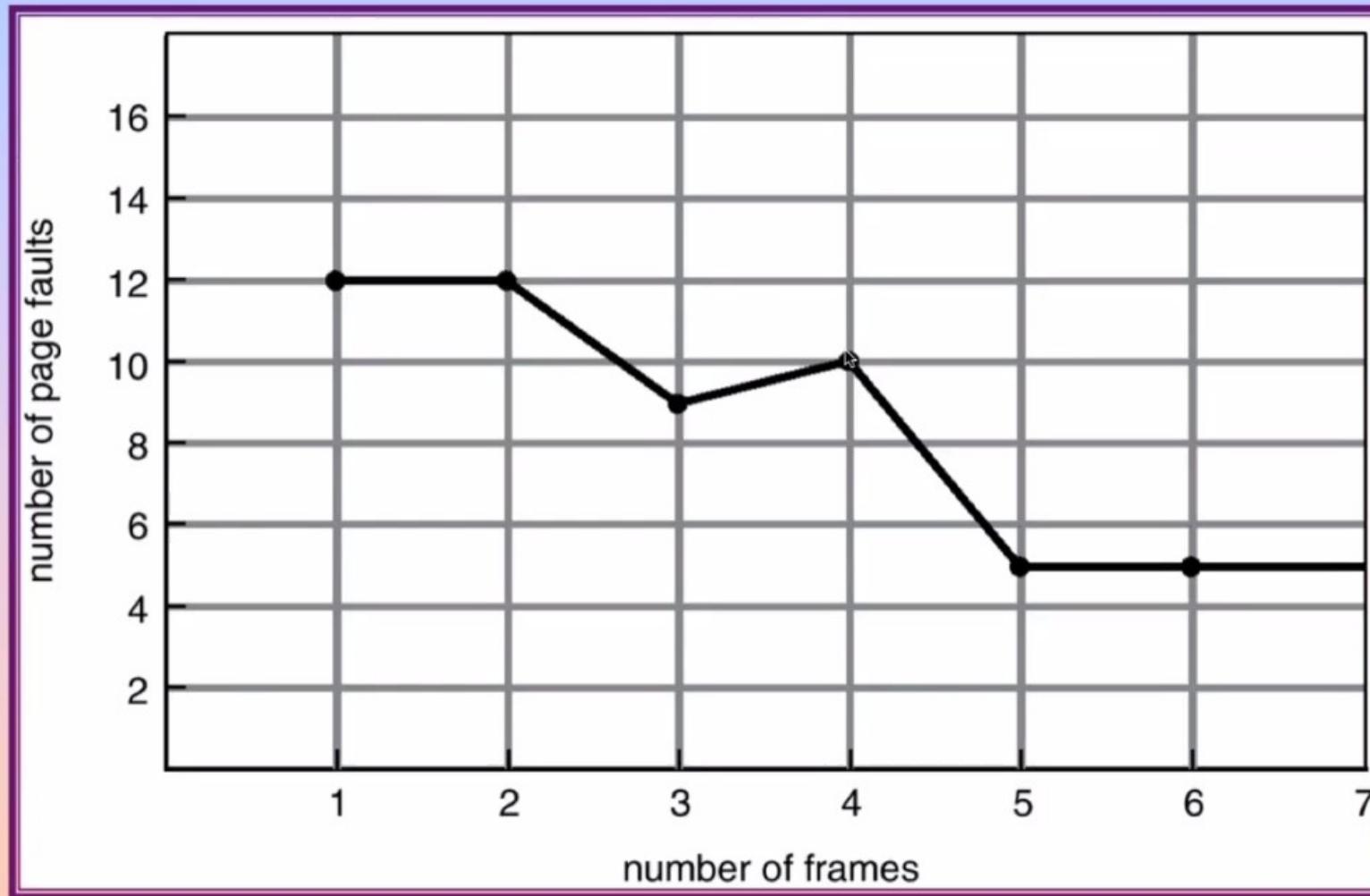
- 4 frames

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

- FIFO – Belady's Anomaly
  - ◆ più frames  $\Rightarrow$  più page faults



# FIFO: l'anomalia di Belady





# Sostituzione ottimale delle pagine

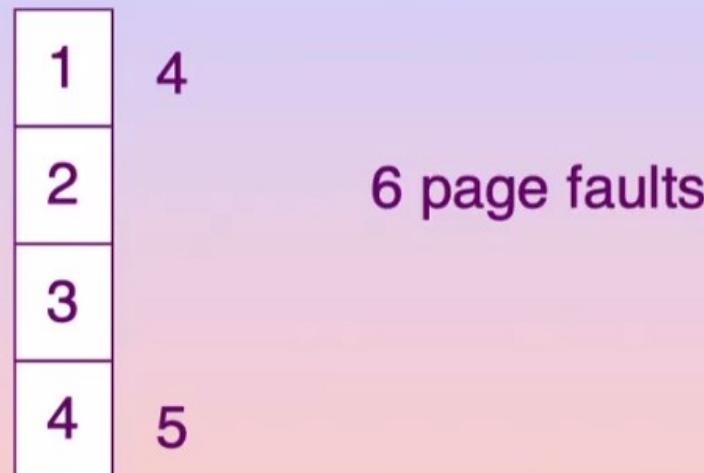
- In un algoritmo ottimale di sostituzione delle pagine (a volte detto OPT o MIN):
  - ◆ si sostituisce la pagina che non si userà per il periodo di tempo più lungo.
- L'uso di questo algoritmo assicura la frequenza di assenze di pagina più bassa possibile per un numero fisso di frame.
- Sfortunatamente l'algoritmo ottimale di sostituzione delle pagine è difficile da realizzare perché richiede la conoscenza futura della successione di riferimenti.
  - ◆ una situazione analoga all'algoritmo SJF di scheduling della CPU...
- Quindi l'algoritmo ottimale si utilizza soprattutto per studi comparativi e per valutare le prestazioni di altri algoritmi.





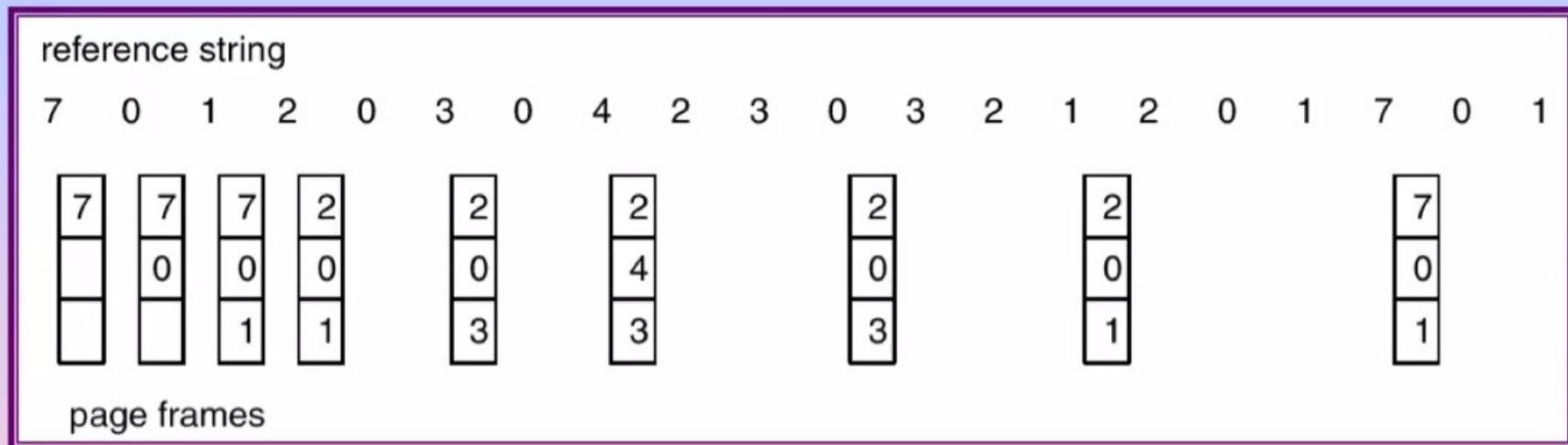
# Algoritmo ottimale

- Esempio per 4 frames
- Successione di riferimenti: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5





## Algoritmo ottimale (II)



- Con solo 9 assenze di pagina la sostituzione ottimale risulta migliore della FIFO (15 assenze).
  - ◆ Ignorando le prime 3 assenze che si verificano con tutti gli algoritmi la sostituzione ottimale e' 2 volte migliore della FIFO.
- Nessun algoritmo potrà gestire questa successione di riferimenti a tre frames con meno di 9 page faults.



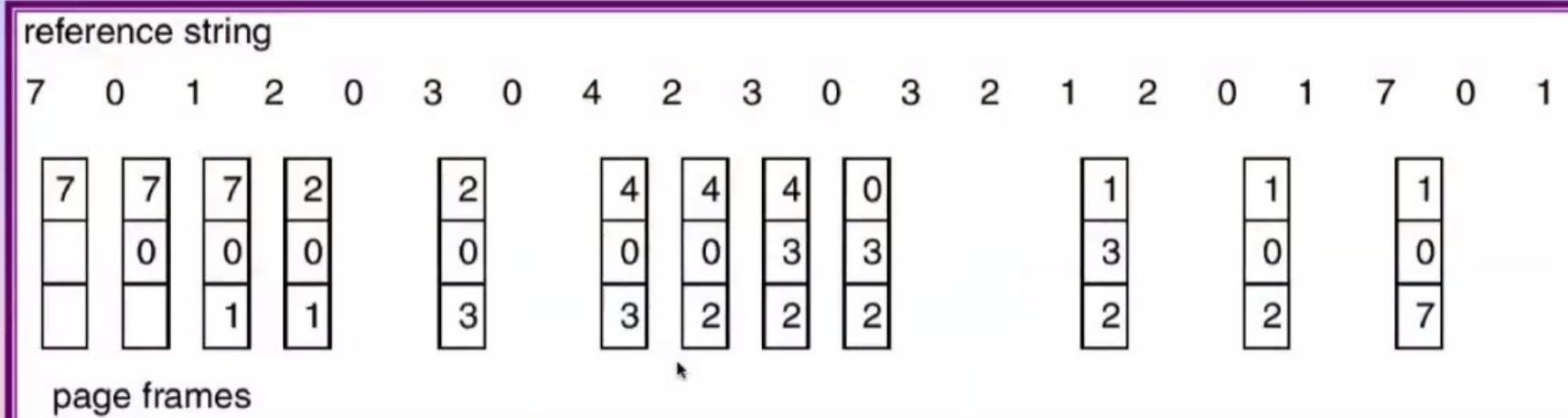


# Sostituzione delle pagine usate meno frequentemente (LRU)

- Sostituisce la pagina che non è stata usata per il periodo di tempo più lungo.
- Successione di riferimenti: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



# LRU





# Implementare LRU: Contatori

## ■ Problema:

- ◆ determinare un ordine per i frame secondo il momento dell'ultimo uso.

## ■ Due soluzioni:

- ◆ Contatori
- ◆ Stack

## ■ Contatori:

- ◆ 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.
  - ✓ In questo modo è sempre possibile conoscere il momento in cui è stato fatto l'ultimo riferimento ad ogni pagina.
- ◆ Quando una pagina deve essere sostituita si guardano i valori dei contatori e si sostituisce la pagina con il valore associato più piccolo.





# Implementare LRU: Stack

## ■ Stack:

- ◆ Mantenere uno stack dei numeri di pagina in una lista doppiamente linkata.
- ◆ Ogni volta che si fa riferimento ad una pagina, la si estraе dallo stack e la si mette in cima allo stack stesso.
- ◆ In questo modo in cima allo stack si trova sempre la pagina usata per ultima, mentre in fondo si trova la pagina usata meno recentemente.

## ■ Nessuna ricerca per la sostituzione di pagina:

- ◆ basta prelevare la pagina dal fondo;

## ■ Ogni aggiornamento dello stack e' un pò più costoso, bisognerà spostare dei puntatori.

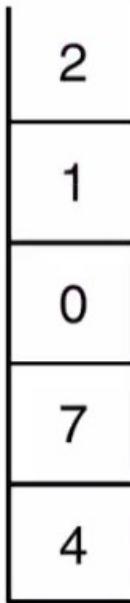
## ■ Né la sostituzione ottimale né quella LRU sono soggette all'anomalia di Belady.



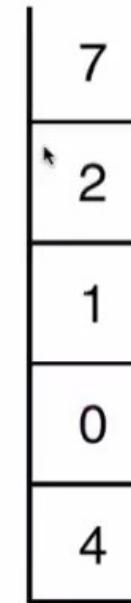
# Implementare LRU: Stack (II)

reference string

4 7 0 7 1 0 1 2 1 2 7 1 2



stack before a



stack after b

a  
b



# Sostituzione delle pagine per approssimazione a LRU

- Sono pochi i sistemi in grado di avere una architettura (contatori, stack) adatta alla gestione della sostituzione delle pagine usate meno frequentemente (LRU).
- Molti sistemi tentano delle approssimazioni.
- Molte architetture offrono come ausilio un *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).
  - ◆ In questo modo però non e' possibile conoscere l'ordine d'uso delle pagine.





# Algoritmo con bit supplementari di riferimento

- E' possibile conservare in una tabella in memoria una serie di bit per ogni pagina.
- Ad intervalli regolari (ad es. ogni 100 millisecondi) il sistema operativo può spostare il bit di riferimento di ciascuna pagina nel bit più significativo della sequenza
  - ◆ traslando gli altri a destra e scartando il bit meno significativo.
- Ad es. se il registro a scorrimento relativo ad una pagina è di 8 bit
  - ◆ 00000000 significa che la pagina non è stata usata da 8 periodi di tempo.
  - ◆ 11000000 significa che la pagina e' stata usata più recentemente di una che ha 01111111.





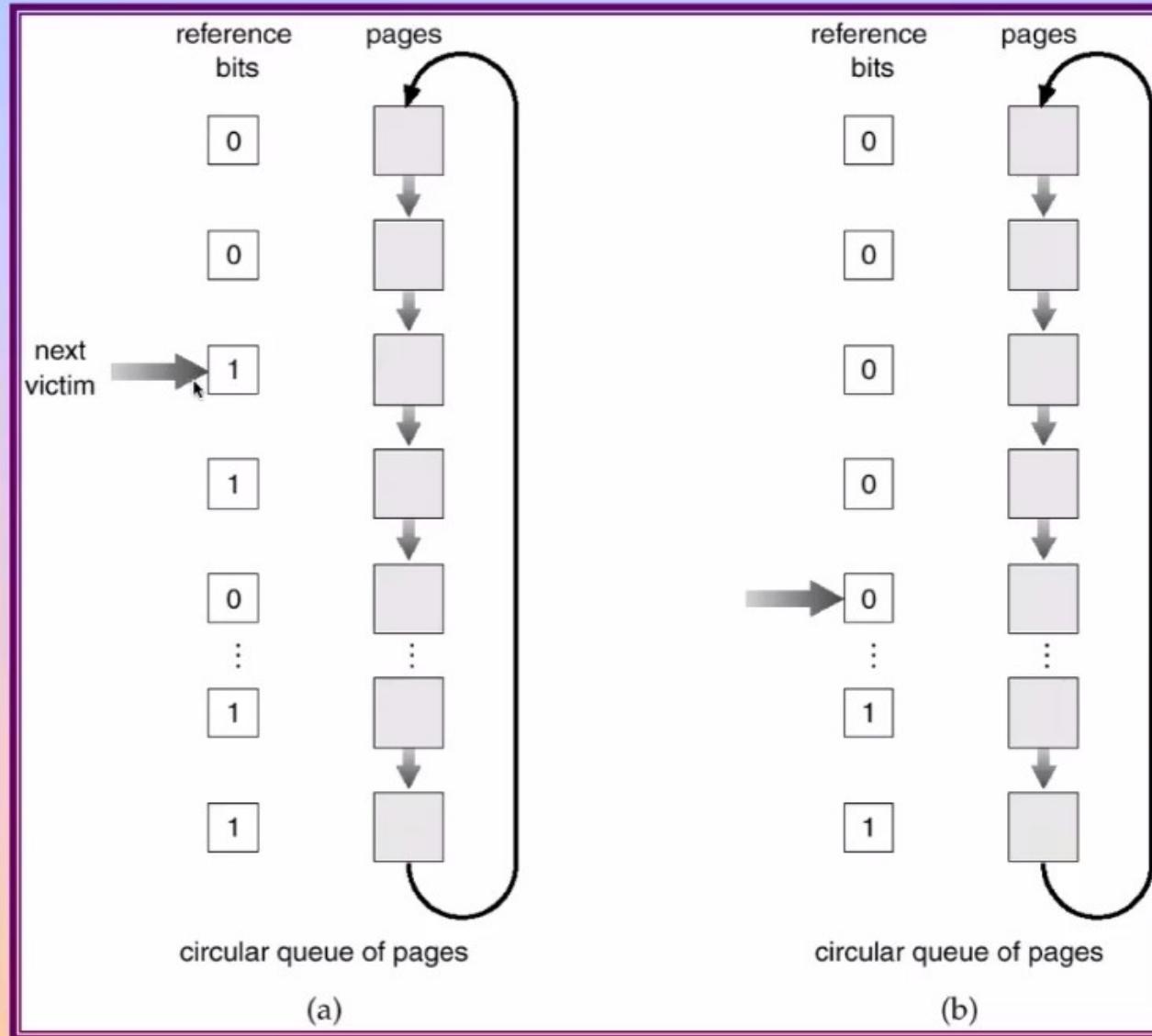
# Algoritmo con seconda chance

- L'algoritmo di base per la sostituzione con la seconda chance è un algoritmo di sostituzione di tipo 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 se il bit di riferimento è 0 la si seleziona per essere sostituita.





# Algoritmo con seconda chance (II)





# Algoritmo con seconda chance migliorato

- Raffinamento dell'algoritmo della seconda chance ottenuto considerando il bit riferimento ed il bit di modifica come una coppia ordinata con cui si possono ottenere quindi:
  - ◆ (0,0) non usato di recente, non modificato
    - ✓ migliore pagina da sostituire
  - ◆ (0,1) non usato di recente, modificato
    - ✓ non ottimale perchè prima di essere sostituita deve essere scritta in memoria secondaria
  - ◆ (1,0) usato di recente, non modificato
    - ✓ probabilmente la pagina sarà presto ancora usata
  - ◆ (1,1) usato di recente, modificato
    - ✓ probabilmente la pagina sarà presto ancora usata e dovrà essere scritta in memoria secondaria prima di essere sostituita
- Ogni pagina rientra in una di queste quattro classi.
- Alla richiesta di una sostituzione
  - ◆ si applica la stessa strategia della seconda chance e si sostituisce la prima pagina che si trova nella classe minima non vuota.





# Sostituzione della pagine basata su 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 (Most Frequently Used):
  - ◆ sostituisce la pagina con il conteggio più alto.





# Algoritmi con memorizzazione transitoria delle pagine

- I sistemi hanno generalmente un gruppo di frame liberi per soddisfare le richieste velocemente (pool of free frames).
- Quando si verifica un page fault si sceglie un frame vittima ma se deve essere scritto in memoria secondaria si trasferisce la pagina richiesta in un frame libero del gruppo.
- Questa procedura permette al processo di ricominciare senza attendere che la pagina vittima sia scritta in memoria secondaria.
- Pagine modificate già scelte come vittime sono poi scritte sul disco periodicamente in background ed aggiunte al pool of free frames.
- E' anche possibile la ricerca nel pool dei frame liberi in memoria di una pagina precedentemente sostituita e nuovamente necessaria.
- Se il frame non è stato riallocato, questa è probabilmente ancora in memoria e non è stata sovrascritta.





# Allocazione delle pagine

- Ogni processo ha bisogno di un numero minimo di pagine in memoria
- Inoltre alcune istruzioni potrebbero essere costituite da più parole macchina e quindi indirizzare memoria che è a cavallo di più pagine.
- Il numero minimo di frame per ciascun processo è definito dall'architettura.
- Il numero massimo è invece definito in base alla quantità di memoria fisica disponibile.
- Esistono due principali schemi di allocazione:
  - ◆ Allocazione statica (o fissa)
  - ◆ Allocazione dinamica (o a priorità)





# Allocazione uniforme e proporzionale

## ■ Allocazione uniforme

- ◆ avendo m frame ed n processi, si assegnano m/n frame a ciascun processo
  - ✓ ad 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.

—  $s_i$  = dimensione del processo  $p_i$

—  $S = \sum s_i$

—  $m$  = numero totale di frames

—  $a_i$  = 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

- Una variante dello schema precedente è quella di usare uno schema di allocazione proporzionale basato sui valori delle priorità piuttosto che delle dimensioni.
- Ad esempio se il processo  $P_i$  genera un page fault,
  - ◆ per la sostituzione si selezionerà uno dei suoi frames,
  - ◆ oppure il frame di un processo con priorità inferiore rispetto a  $P_i$ .
- Si potranno avere anche scelte basate su combinazioni di dimensioni e priorità.





# Sostituzione globale e locale

- Un importante fattore che riguarda il modo in cui si assegnano i frame è la modalità di sostituzione delle pagine.
- Gli algoritmi di sostituzione si possono classificare in due categorie generali:
  - ◆ 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.
- Con la sostituzione locale il numero di frame assegnati ad un processo non cambia.
- Con quella globale potrebbe accadere che per la sostituzione si selezionino frame allocati ad altri processi,
  - ◆ aumentando quindi il numero di frame assegnati a quel processo.
  - ◆ Generalmente la sostituzione globale genera maggiore produttività ed è il metodo più usato.



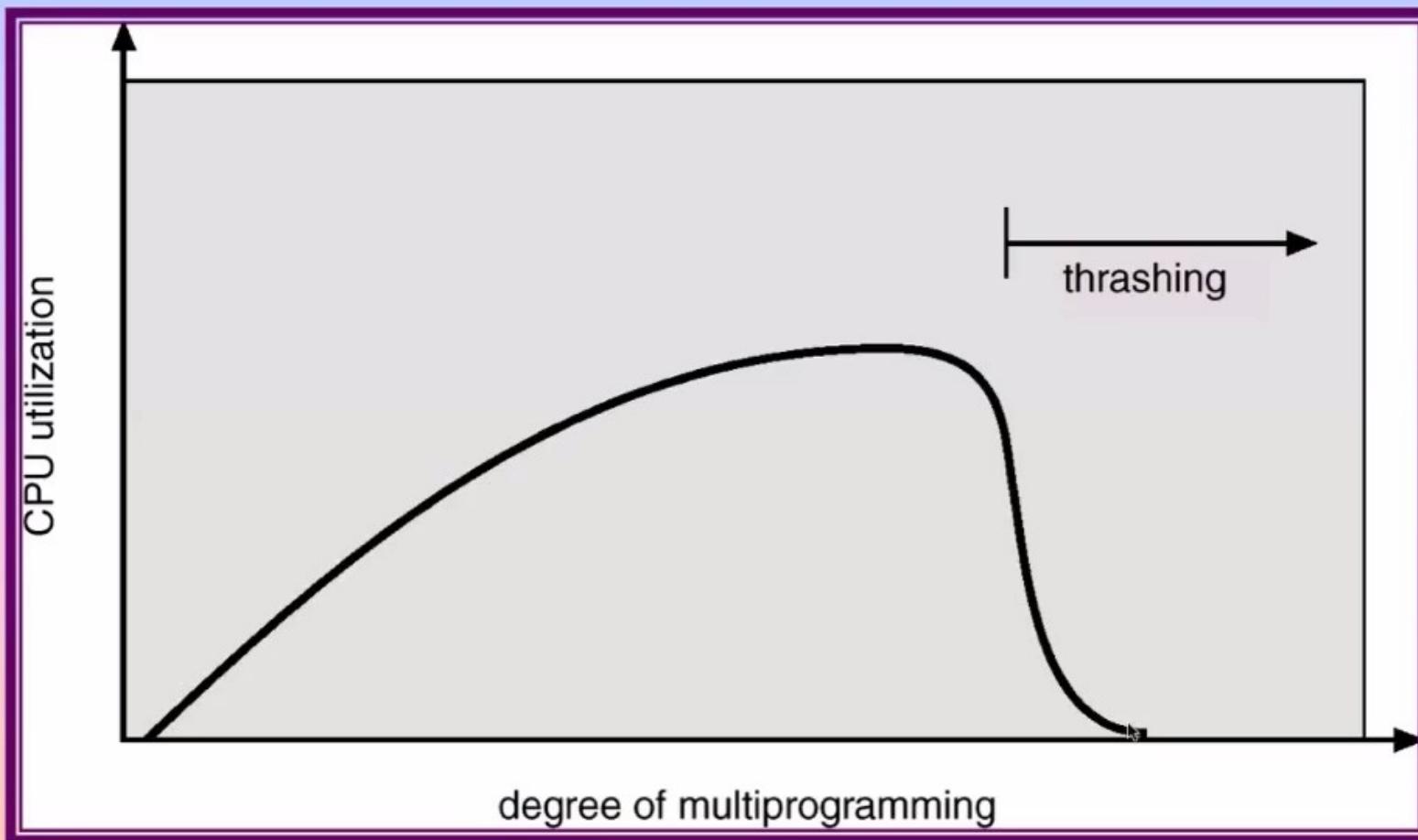


# Paginazione degenera (thrashing)

- Se un processo non ha abbastanza pagine, il tasso di page fault aumenta.
- Questo comporta:
  - ◆ riduzione utilizzo della CPU;
  - ◆ quindi il sistema operativo potrebbe ritenere che sia necessario aumentare il livello di multiprogrammazione;
  - ◆ quindi un altro processo potrebbe essere aggiunto al sistema (potenzialmente rallentandolo ancora di più).
- Thrashing  $\equiv$  si spende più tempo nella paginazione che nella esecuzione dei processi.
  - ◆ un processo impiega il suo tempo facendo quasi sempre solo swapping di pagine di memoria.



# Thrashing





# Modello di località di esecuzione del processo

- Gli effetti di questa situazione si possono limitare usando un algoritmo di sostituzione locale o un algoritmo di sostituzione per priorità.
- Con la sostituzione locale un processo, anche se ricade nell'attività di paginazione degenera, non può sottrarre frame ad un altro processo e quindi provocarne a sua volta la degenerazione.
- Per evitare il thrashing occorrerebbe sapere quali (oltre che quante pagine) servono.
- Modello di località di esecuzione del processo:
  - ◆ un processo durante la sua esecuzione si muove da una località all'altra.
  - ◆ Una località è un insieme di pagine usate attivamente,
  - ◆ un programma è formato da parecchie località diverse che sono sovrapponibili.
  - ✓ Ad esempio quando si invoca una procedura , questa definisce una nuova “località” in cui si fanno riferimenti alla memoria per le istruzioni della procedura, le variabili locali, etc..
- Perchè si verifica il thrashing?
  - ◆ dimensione della località > numero di frame assegnati.





# Modello dell'insieme di lavoro

- Il modello dell'insieme di lavoro è basato sull'ipotesi di località.
- Il modello usa un parametro  $\Delta$  per definire la finestra dell'insieme di lavoro.
- L'idea è quella di esaminare, come approssimazione della località del programma, i più recenti  $\Delta$  riferimenti alle pagine ("insieme di lavoro").
  - ◆  $\Delta \equiv$  working-set window (finestra dell'insieme di lavoro)  $\equiv$  un numero fisso di riferimenti di pagina
    - ✓ Ad esempio: 10,000 istruzioni
- La caratteristica più importante dell'insieme di lavoro è la sua dimensione.
  - ◆ Calcolando la dimensione dell'insieme di lavoro per ogni processo si può determinare la richiesta totale di frame.
- $WSS_i$  (working set del processo  $P_i$ ):
  - ◆ numero totale di pagine cui  $P_i$  si riferisce nel più recente  $\Delta$  (varia nel tempo)
    - ✓ se  $\Delta$  è troppo piccolo non comprenderà l'intera località
    - ✓ se 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





# Modello dell'insieme di lavoro (II)

- $D = \sum WSS_i$ , richiesta globale dei frame.
  - ◆ Se  $D > m$  (dove  $m$  è il numero di frame liberi)  $\Rightarrow$  Thrashing.
  - ◆ Se  $D > m$ , allora occorre sospendere uno dei processi.
- Un volta scelto  $D$  il Sistema Operativo controlla l'insieme di lavoro di ogni processo e gli assegna un numero di frame sufficiente.
- Se i frame ancora liberi sono in numero sufficiente si può iniziare un nuovo processo.
- Se la somma delle dimensioni degli insiemi di lavoro aumenta superando il numero di frame disponibili SO individua un processo da sospendere.
  - ◆ Scrive in memoria secondaria le pagine di quel processo.
  - ◆ Assegna i suoi frame ad altri processi.
  - ◆ Il processo sospeso potrà essere ripreso successivamente.
- Questa strategia impedisce la paginazione degenera
  - ◆ mantenendo il grado di multiprogrammazione più alto possibile ed ottimizzando, quindi, l'utilizzo della CPU.





# Frequenza delle assenze di pagine

- E' possibile utilizzare una strategia basata sul controllo della frequenza di assenze di pagine per prevenire la paginazione degenera.
- Stabilire un tasso "accettabile" di page-fault
  - ◆ Se il tasso attuale è troppo basso, il processo potrebbe usare troppi frame.
  - ◆ Se il tasso attuale è troppo alto, il processo ha bisogno di più frame.

