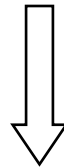


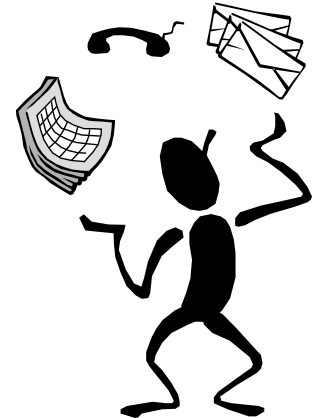
MEMORIA VIRTUALE

Idea di base

- In ogni momento *solo una parte del programma ha bisogno di risiedere in memoria*
- Lo spazio degli *indirizzi logici* puo' essere *molto piu' grande* di quello degli *indirizzi fisici*
- *Singole pagine* devono poter essere *swapped out/in*

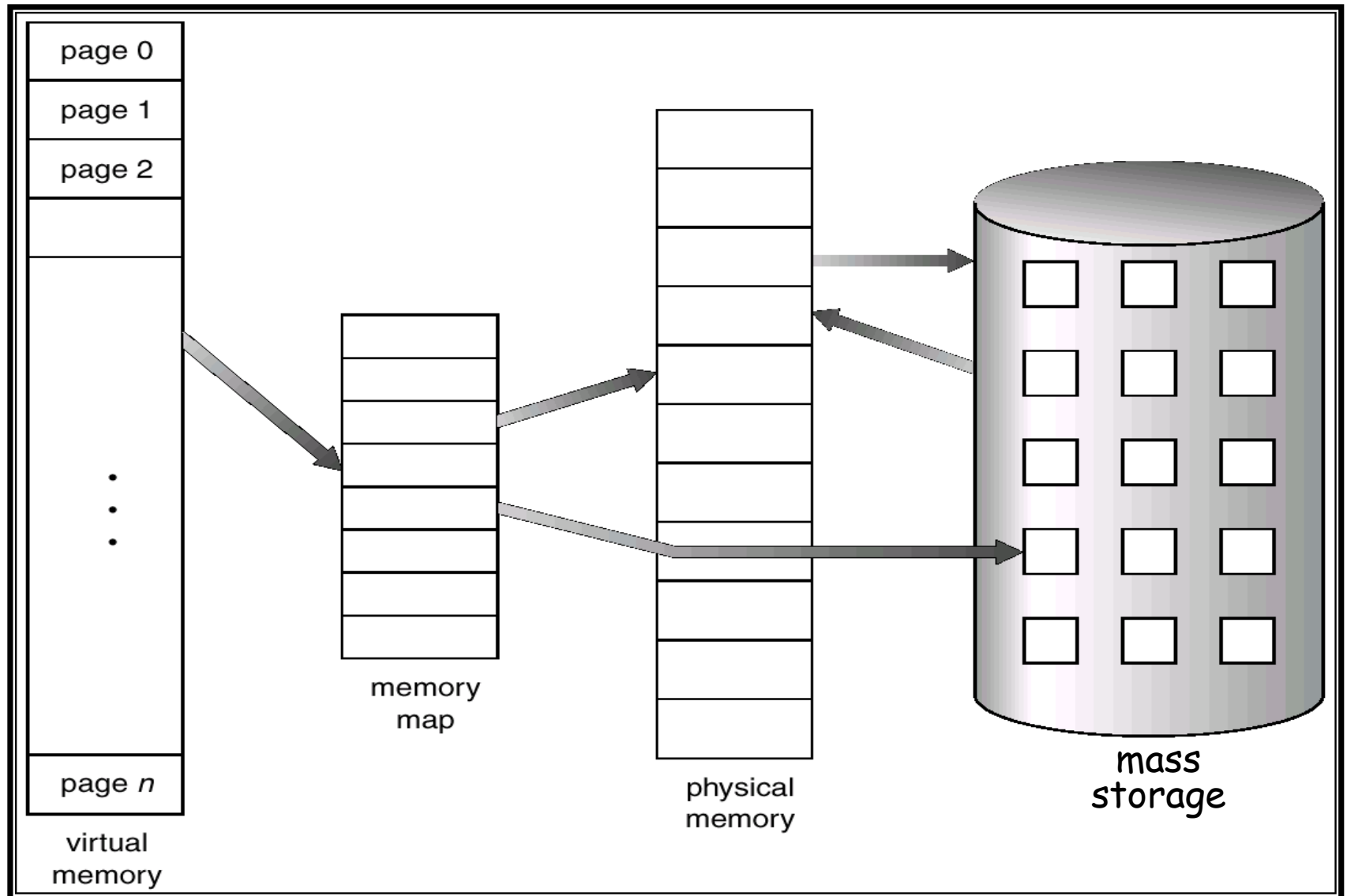


Paginazione su richiesta
Segmentazione su richiesta



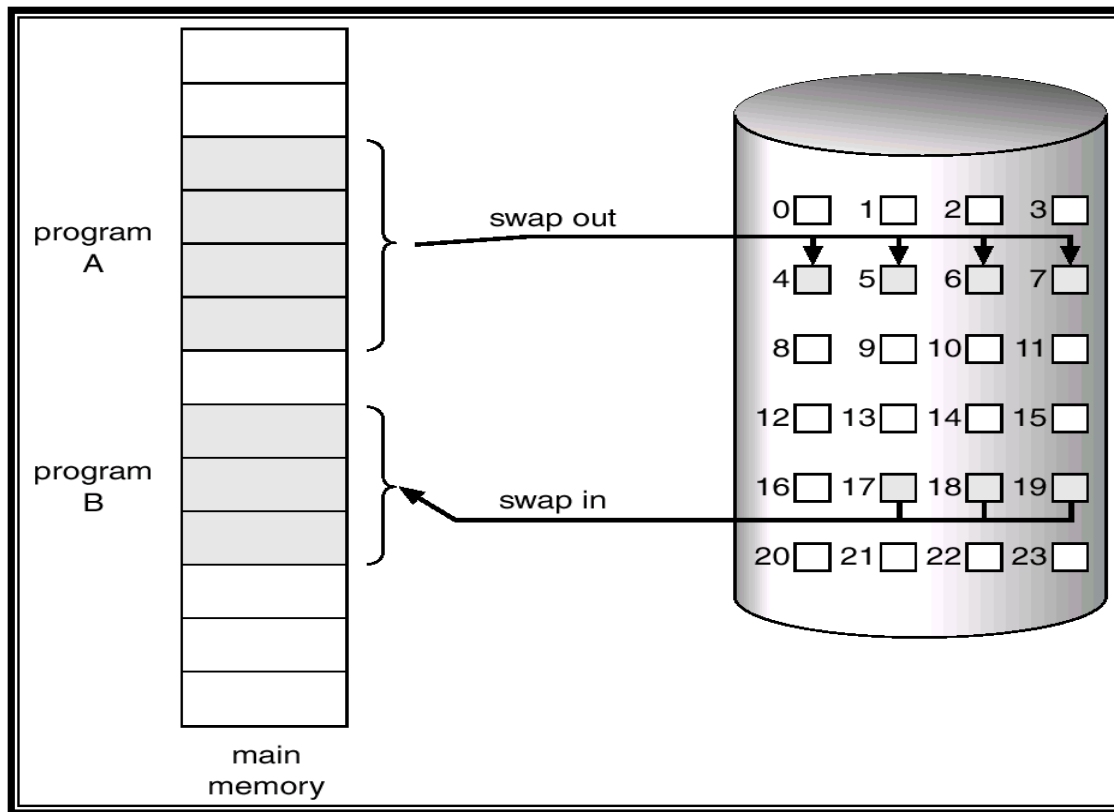
Ritorna il concetto di *overlays*, ma in modo *trasparente* in quanto tutto viene gestito da OS

... e una raffigurazione



Paginazione su richiesta

PAGER - uno *swapper* che non *trasferisce* processi interi, ma solo *singole pagine* e solo *quando queste sono necessarie*



Quella residente in memoria e' **solo una parte delle pagine** dei due processi A e B



Identificazione di pagine “valide”

Un *bit di validita'* e' associato ad ogni entrata della page table (1 \Rightarrow in memoria, 0 \Rightarrow non in memoria)

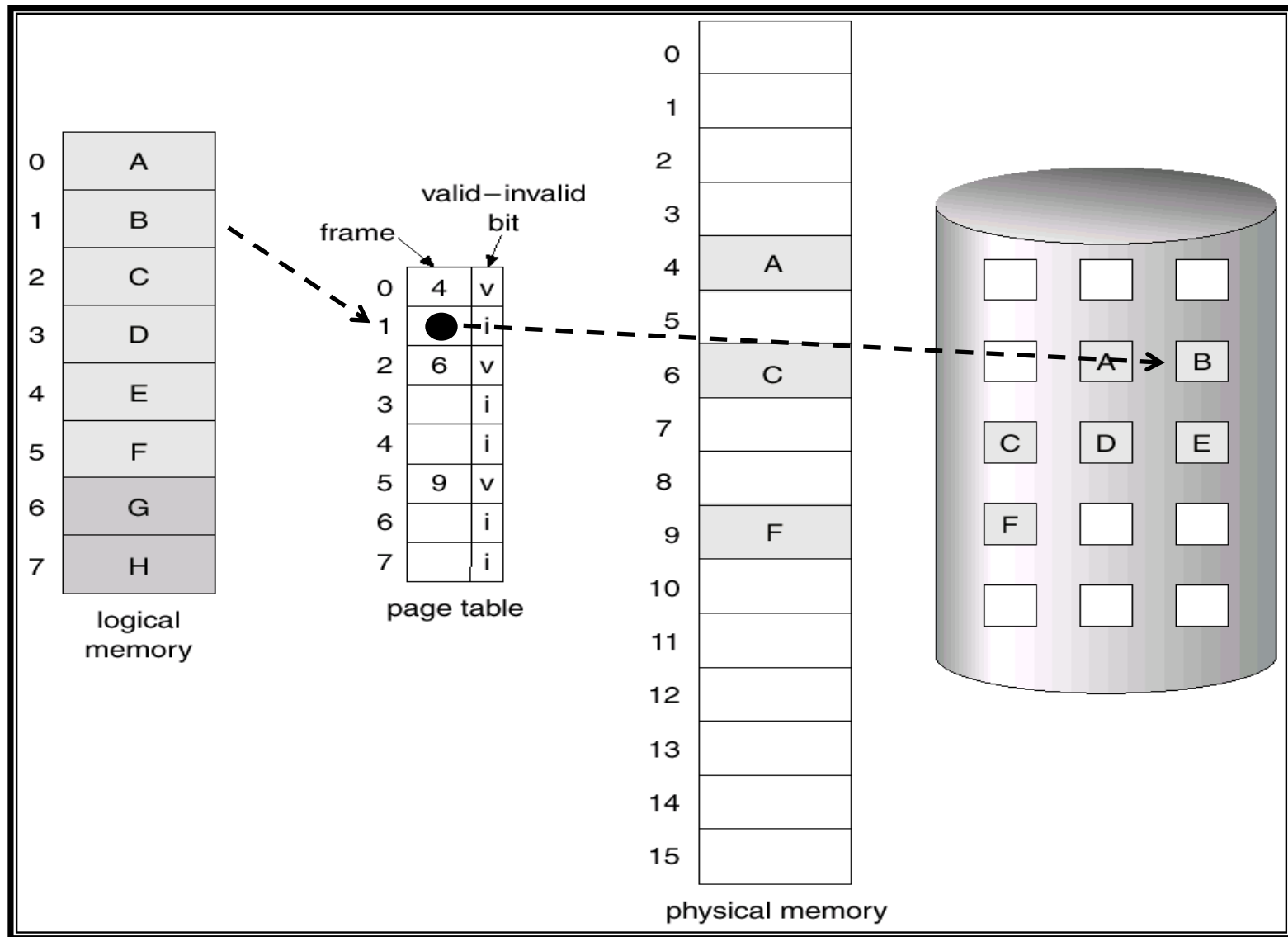
Frame #	valid-invalid bit
M	1
N	1
D	1
R	1
<i>N/A</i>	0
G	1
<i>N/A</i>	0
	0

page table

→ pagine che non appartengono allo spazio di indirizzi oppure pagine che non sono state caricate in memoria

Per queste ultime l'indirizzo potrebbe essere quello del settore di disco dove risiedono

... e una raffigurazione

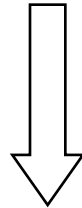


Page Fault : quando accade?!

Pagina *non residente in memoria*

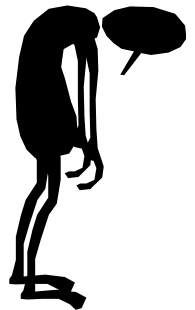
⇒ errore di predizione nell'insieme di pagine in memoria

⇒ *trap a OS*

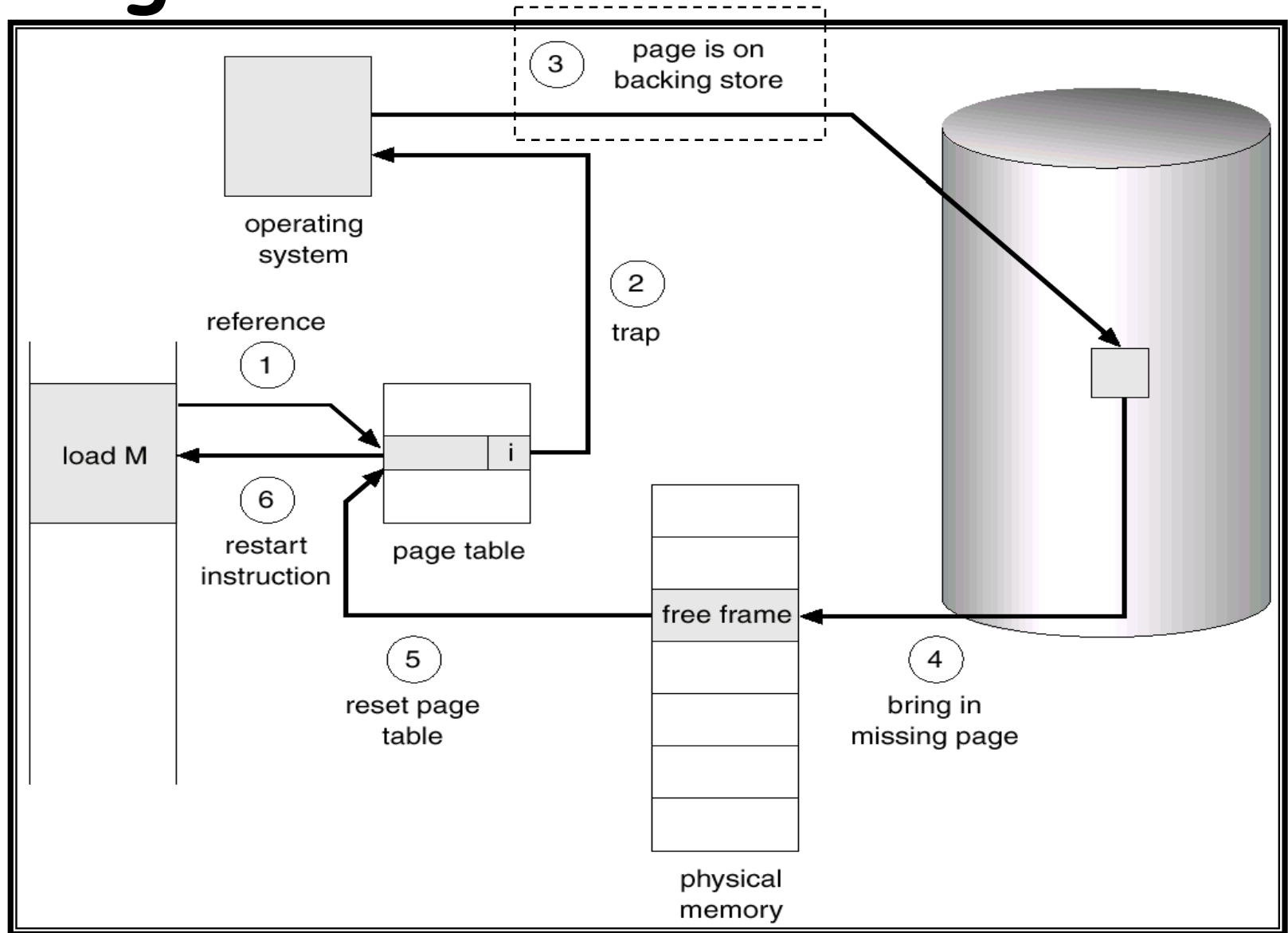


OS guarda allo spazio degli indirizzi per decidere:

- Riferimento non valido ⇒ abort
- *Riferimento valido ⇒ page fault*



Page Fault : che cosa si fa?!



Page Fault : che cosa si fa?!

- *Allocazione di un frame* dalla tabella dei frame liberi
- *Trasferimento della pagina* in memoria
- *Aggiornamento della tabella delle pagine* e dei relativi bit di validita'
- *Ripresa dell'esecuzione* dell'istruzione

Qualche considerazione

- In teoria un processo puo' addirittura cominciare *senza pagine in memoria*, e ad ogni richiesta ne vengono caricate
- La *localita' di riferimento* dovrebbe aiutare
- *Una sola istruzione* puo' anche causare un *multiplo page fault*, uno per l'istruzione stessa e vari per i dati
- Un page fault su dati puo' portare a *rieseguire l'intera istruzione*

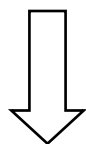
Prestazioni della paginazione su richiesta

- *Frequenza di page fault* $p : 0 \leq p \leq 1$
 - se $p = 0$ non ci sono page faults
 - se $p = 1$ ogni riferimento genera un page fault
- *Tempo di Accesso Effettivo (EAT)*

$EAT = \text{accesso a memoria} + p \cdot \text{overhead di page fault}$

Accesso a memoria = 100 nanosecondi

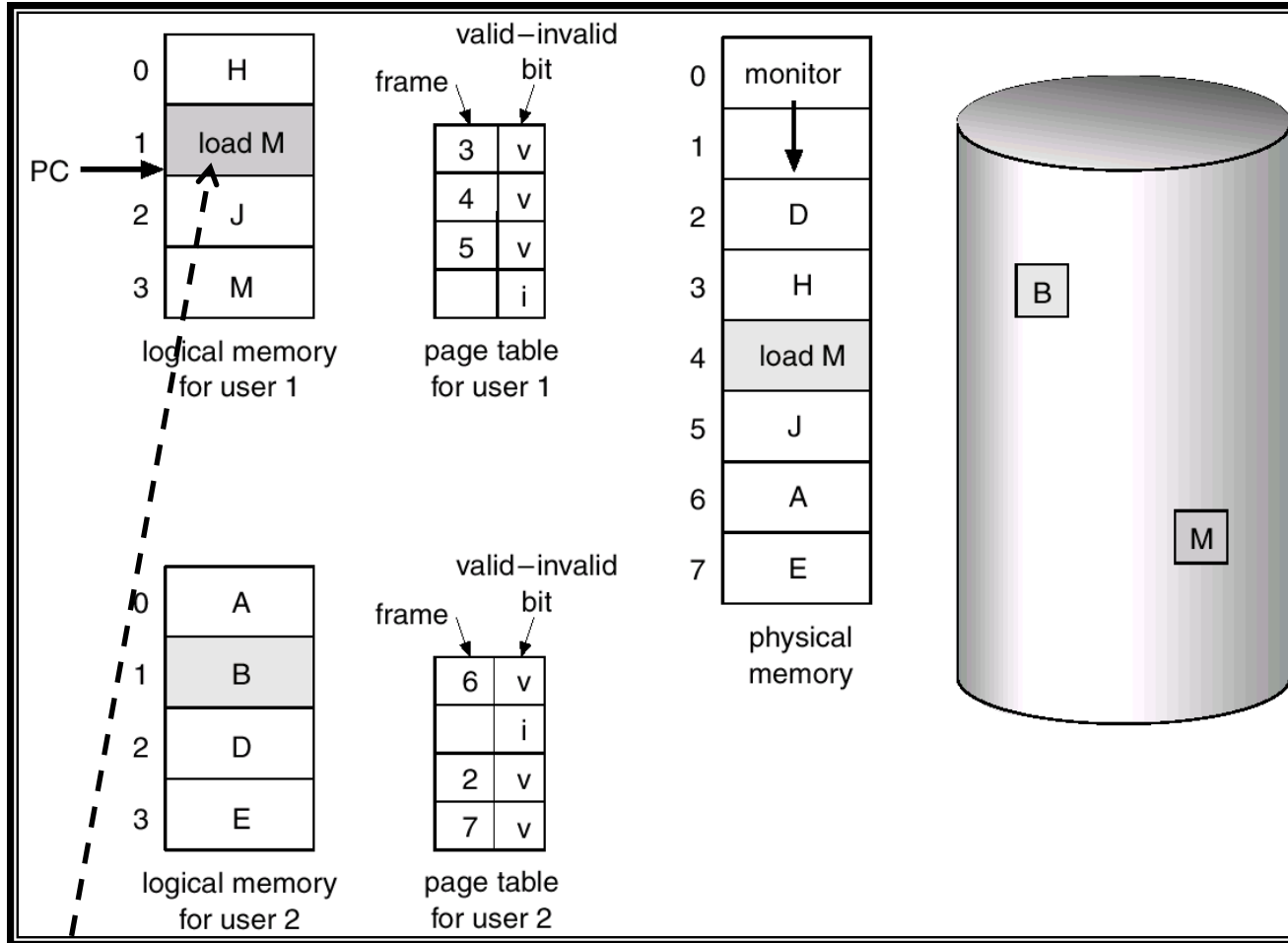
Overhead di page fault = 25 millisecondi



$$EAT = 100 + 25.000.000 \times p$$

Un rallentamento
entro il 10% puo' già
derivare da un page
fault ogni 2.500.000
accessi in memoria
circa

Cosa succede se non ci sono frame liberi?!



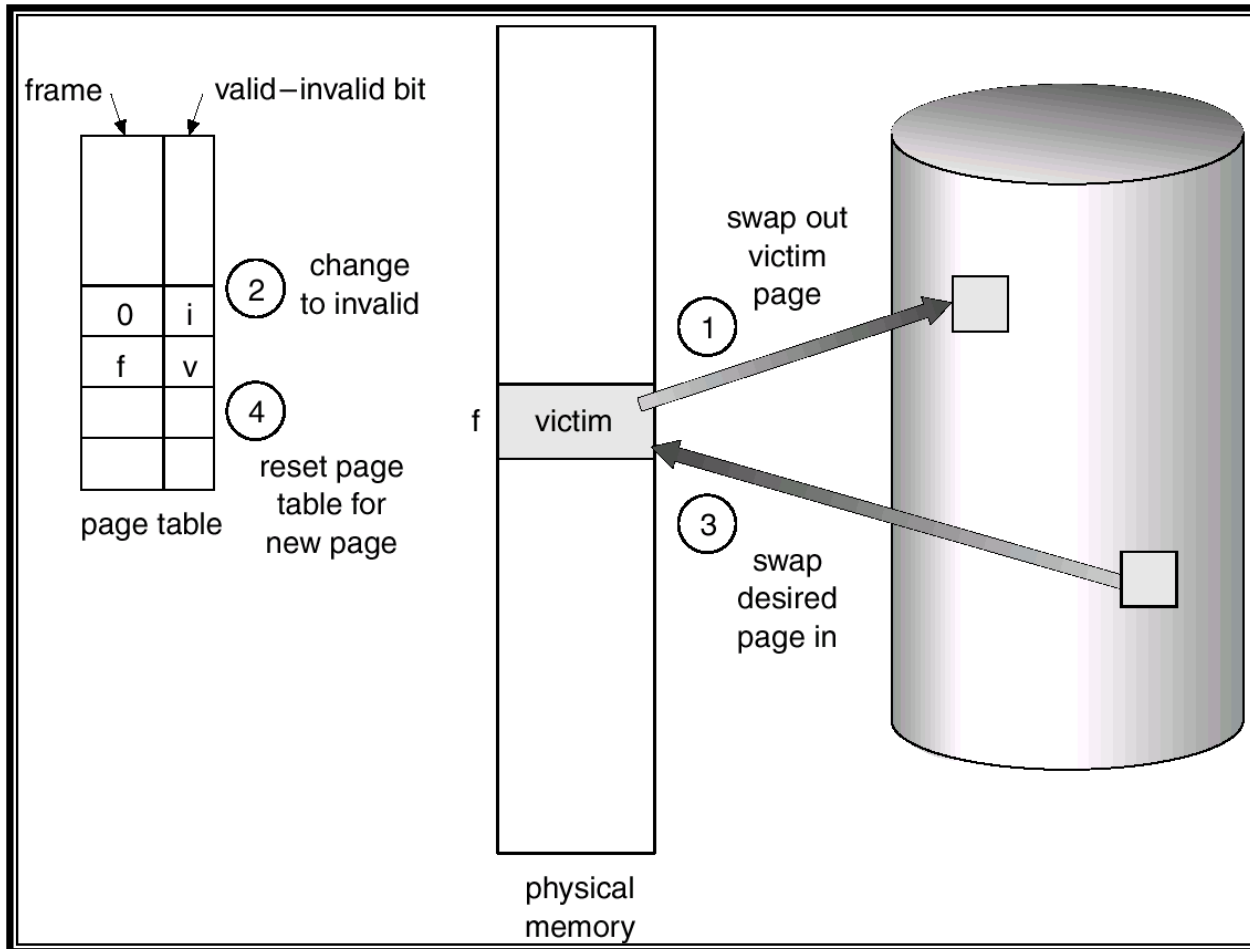
Si e' destinato un numero di frame (**fisso o variabile**) al processo inferiore al numero di pagine che quel processo potrebbe voler utilizzare

Un'istruzione in questa pagina richiede il caricamento della pagina M

Soluzione al problema

- Si opera la *sostituzione di una pagina* liberando così il frame sul quale essa è allocata
- Si complica quindi la procedura di page fault in quanto ora richiede un *doppio trasferimento di pagina*
- Si può introdurre il *dirty bit* per evitare di salvare una pagina su disco se non è stata modificata

Quale pagina sara' la vittima?

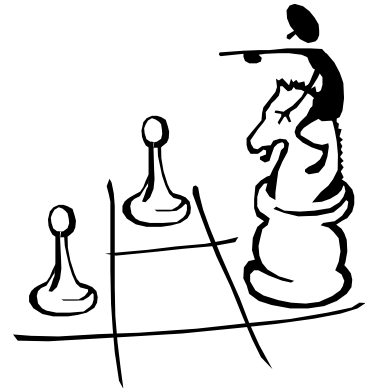


**Algoritmi di
sostituzione
pagina**

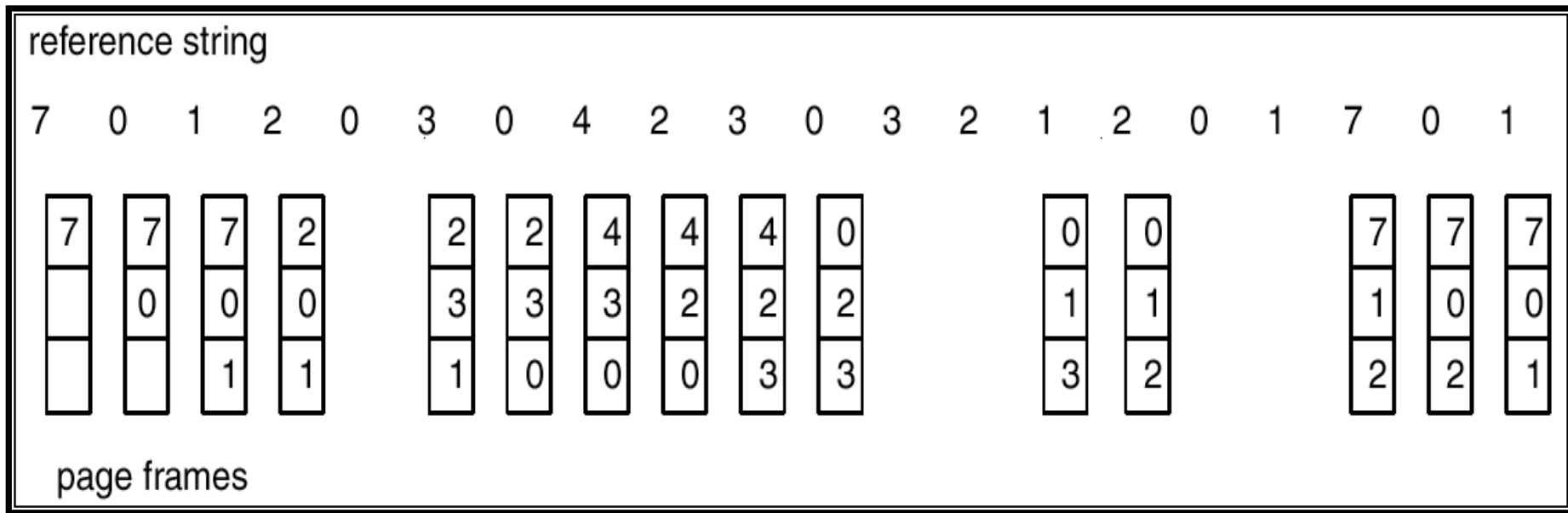
Parametri input : stringa dei riferimenti, numero dei frame
Parametro di valutazione : frequenza di page fault

Algoritmi di sostituzione pagina

- First-In-First-Out
- Ottimale
- Least Recently Used (LRU)
- LRU approssimati



First-In-First-Out (FIFO)



FIFO - *Anomalia di Belady* : piu' frames \Rightarrow meno page faults

- *Stringa dei riferimenti* : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames

9 page faults

1	4	5
2	1	3
3	2	4

Come si spiega?

- 4 frames

10 page faults

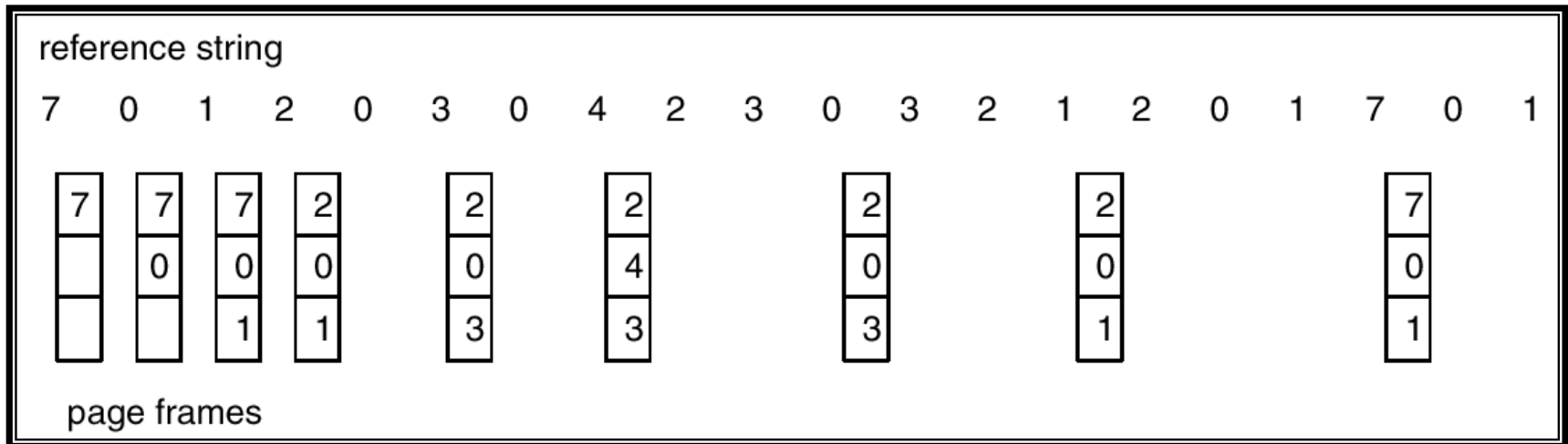
1	5	4
2	1	5
3	2	
4	3	

*Anche la sequenza di
prima soffre di tale
anomalia?*

Algoritmo Ottimale



IDEA : si rimpiazza *la pagina che verra' utilizzata piu' tardi*



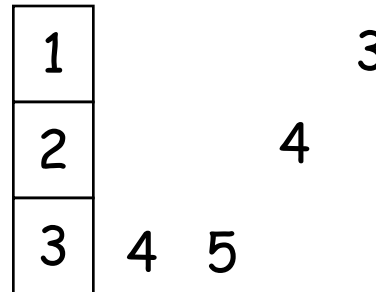
Come possiamo conoscere in anticipo i riferimenti?!
(*analogia con SJF!*)

Viene usata in effetti solo come *parametro di confronto* per la frequenza di page fault (ovviamente minima in questa!)

... e sull' esempio di prima

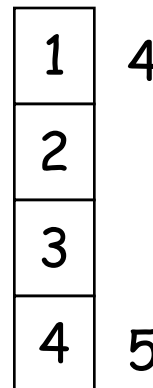
- *Stringa dei riferimenti*: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames

7 page faults



- 4 frames

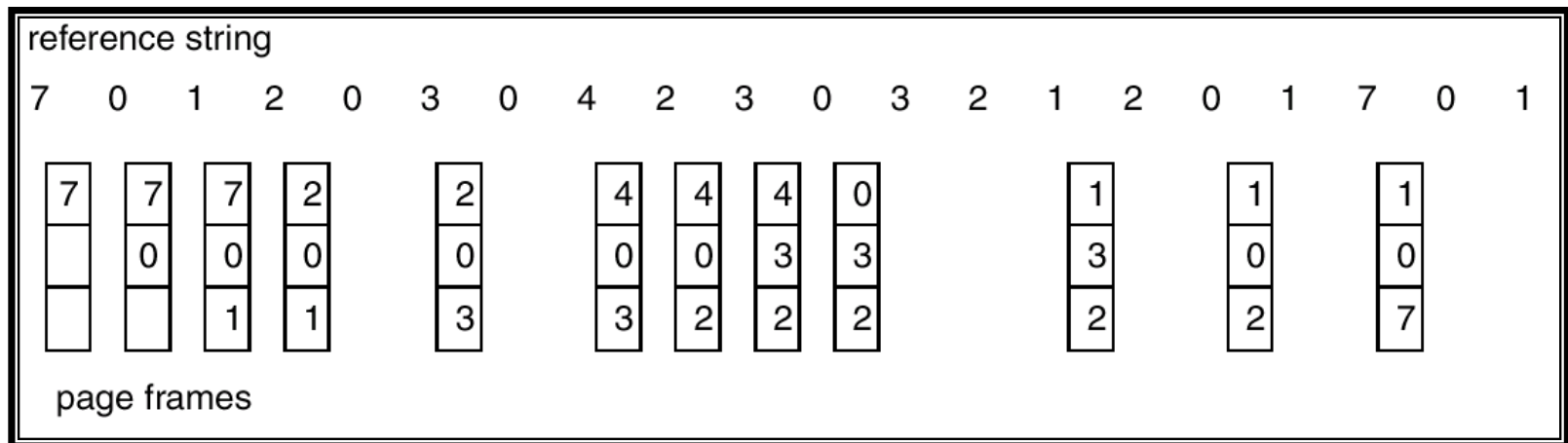
6 page faults



Least Recently Used (LRU)

Via di mezzo tra il considerare *l'istante di arrivo in memoria* (il passato: FIFO) e *l'utilizzazione della memoria* (il futuro: OPT)

Ipotesi di localita': futuro vicino = passato recente \Rightarrow
 si sostituisce la ***pagina utilizzata meno di recente***



E' ottimale tra quelle che guardano solo all'indietro !

Come tener traccia del recente utilizzo delle pagine?



Clock counter

- Ogni pagina ha un *registro associato*
- Ogni volta che si fa un riferimento alla pagina, il valore di un *clock counter logico* viene *copiato nel registro associato* alla pagina
- In caso di sostituzione di pagina si cerca e si seleziona quella con *il valore del registro piu' piccolo*

Come tener traccia del recente utilizzo delle pagine?

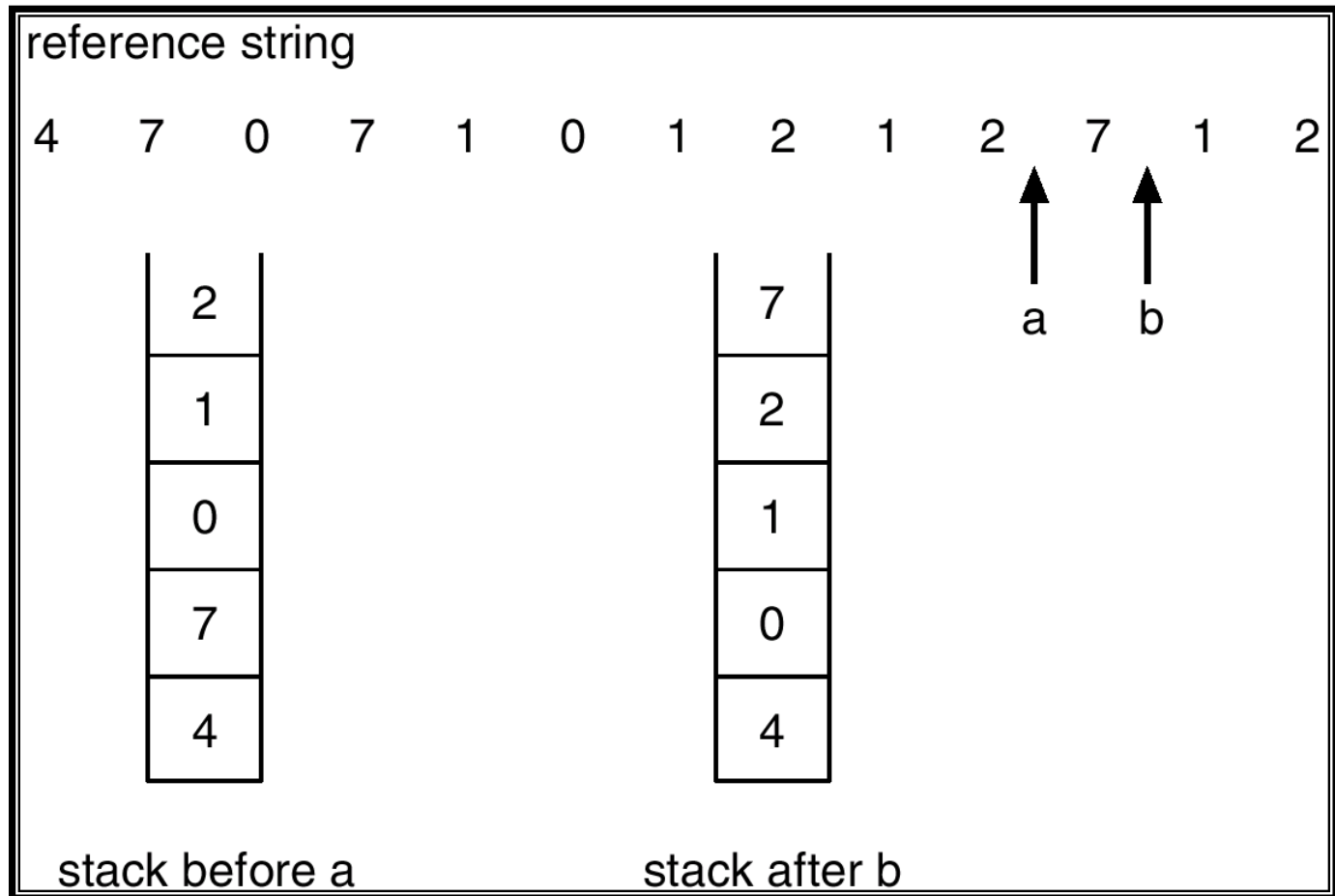
Stack

- Uno *stack di numeri di pagine* viene introdotto
- Ogni volta che si fa un riferimento alla pagina, il *numero di pagina emerge sul top dello stack*
- In caso di sostituzione di pagina si seleziona quella *sul fondo dello stack*
- Questo *evita di dover fare una ricerca di un valore* (come nel caso del clock counter) solo a condizione che una *struttura con doppi puntatori* venga implementata



Come tener traccia del recente utilizzo delle pagine?

Stack



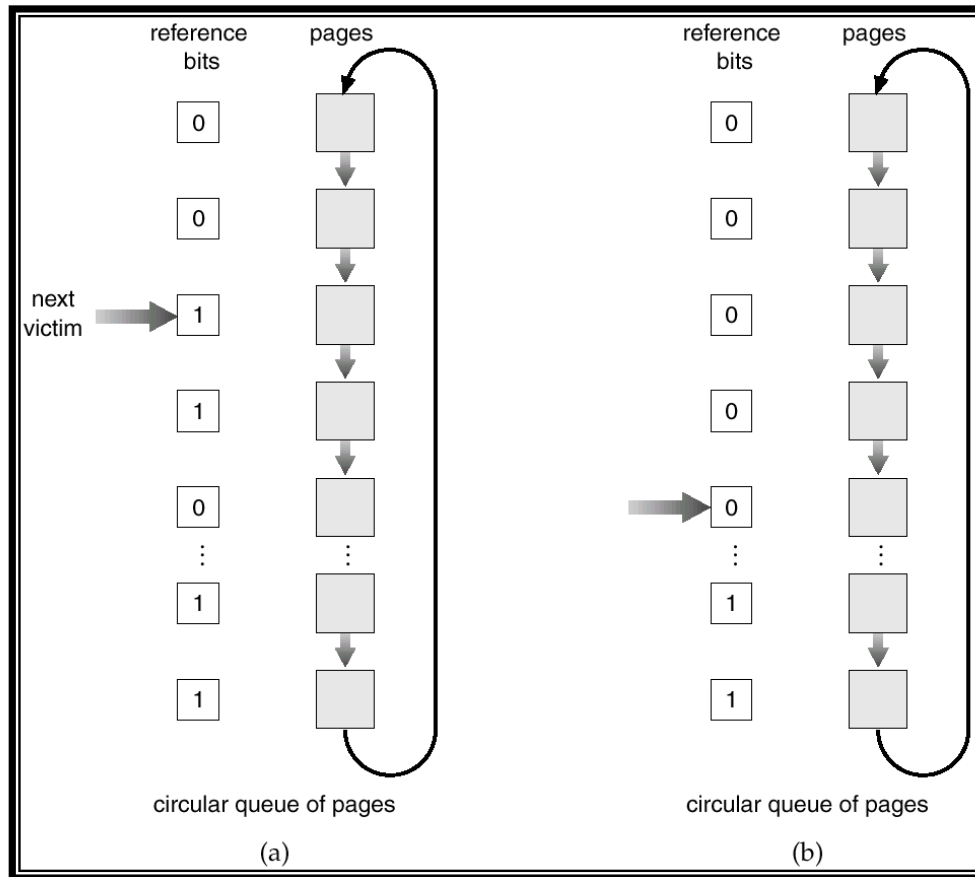
Approssimazioni ad LRU

Bit supplementari di riferimento

- Ad ogni pagina viene associato un *bit di riferimento*
- Un *vettore di scorrimento di bit* (p.es., di taglia 8) viene anche mantenuto per ogni pagina
- Quando si fa un riferimento alla pagina *il bit viene settato a 1*
- Ad intervalli regolari tutti *i bit vengono trasferiti nel vettore* mediante uno *shift verso destra*
- In caso di sostituzione di pagina si seleziona quella con *il piu' piccolo valore di vettore*

Approssimazioni ad LRU

Algoritmo di seconda chance



Si tratta in pratica di una strategia **FIFO** che, quando trova il bit di riferimento a 1, **da' una seconda chance** a quella pagina rimettendo il bit a 0 e cercandone un'altra

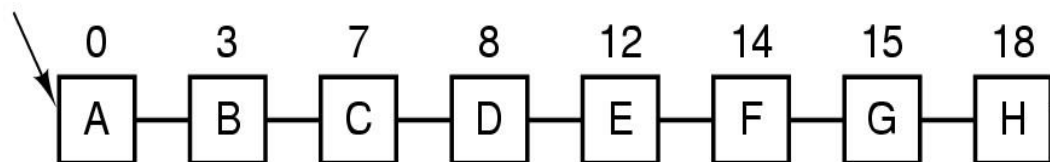
Si puo' introdurre anche un bit di modifica in modo che le classi di pagine diventino quattro

Approssimazioni ad LRU

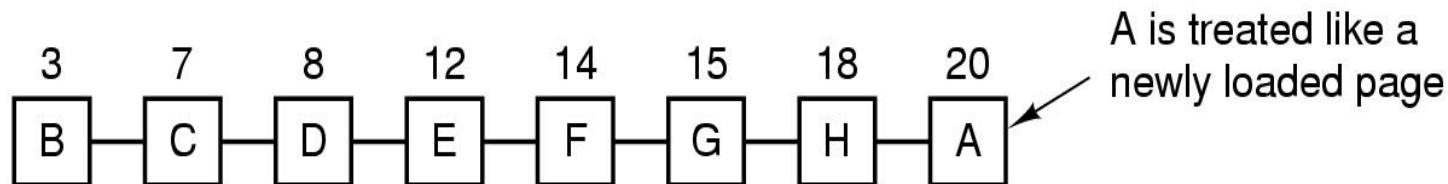
Algoritmo di seconda chance

Altro modo di vederla

Page loaded first



(a)



(b)

Supporto agli algoritmi di rimpiazzamento pagine

- Si puo' introdurre *un pool di frame liberi* per virtualizzare la cancellazione di una pagina...
... tale pool puo' essere anche utilizzato per ritrovare una pagina appena cancellata
- Il **dirty bit** (bit di modifica) puo' essere utilizzato laddove un LRU approssimato mi dia piu' di una pagina candidata vittima
- Si potrebbe anche pensare di *memorizzare sulla memoria di massa a intervalli regolari* tutte le pagine modificate e aggiornarne i bit, in modo da evitare tale overhead in caso di sostituzione




Allocazione di frame

Ogni processo ha dei *limiti sul numero di frame* che possono essere allocati ad esso

Numero massimo : determinato dall'architettura dell'elaboratore e dal grado di multiprocessing

Numero minimo : determinato dal set di istruzioni

dipende da



Numero massimo di frame ai quali una singola istruzione puo' fare riferimento...
... e quindi in parte dai livelli di indirizzazione delle istruzioni

Criteri di allocazione di frame

- Fissa: Uguale
- Fissa: Proporzionale (taglia/priorita')
- Variabile

Tutte prevedono una *dinamica riallocazione* del totale numero di frame in caso di *creazione* o *cancellazione di un processo*

Un importante parametro da tenere in conto nell'allocazione



Sostituzione globale o locale delle pagine ?!

- **Globale** : il frame puo' essere selezionato tra tutti, anche tra quelli di altri processi...

(... la **taglia di allocazione** di un processo **puo' variare**, ma è più difficile controllare la frequenza di page fault perche' influenzata da fattori esterni)

- **Locale** : il frame deve essere selezionato solo tra quelli allocati al processo stesso...

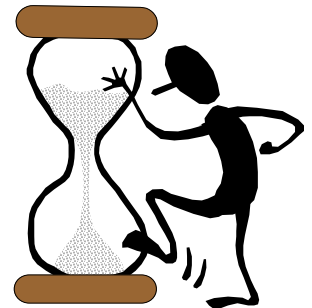
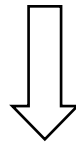
(... la **taglia di allocazione** di un processo **e' fissa**, ma si potrebbero non “vedere” all' occorrenza pagine poco usate)

Thrashing

Se un processo non ha abbastanza pagine, la frequenza di page fault e' molto alta. Questo porta a :

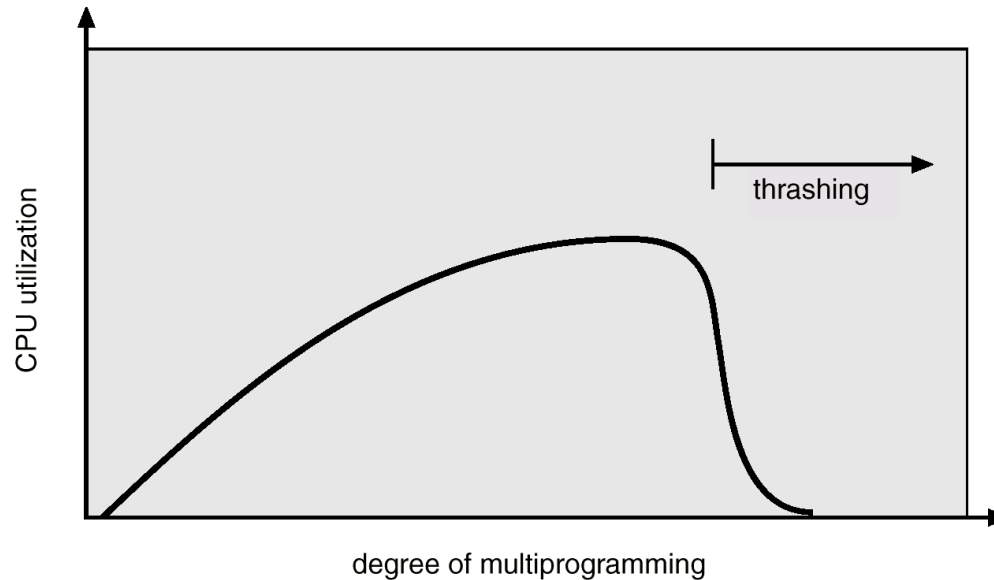
- *Bassa utilizzazione della CPU*
- *OS pensa che si deve aumentare il grado di multiprogrammazione, e quindi aggiunge un altro processo*

... e cosi' via...



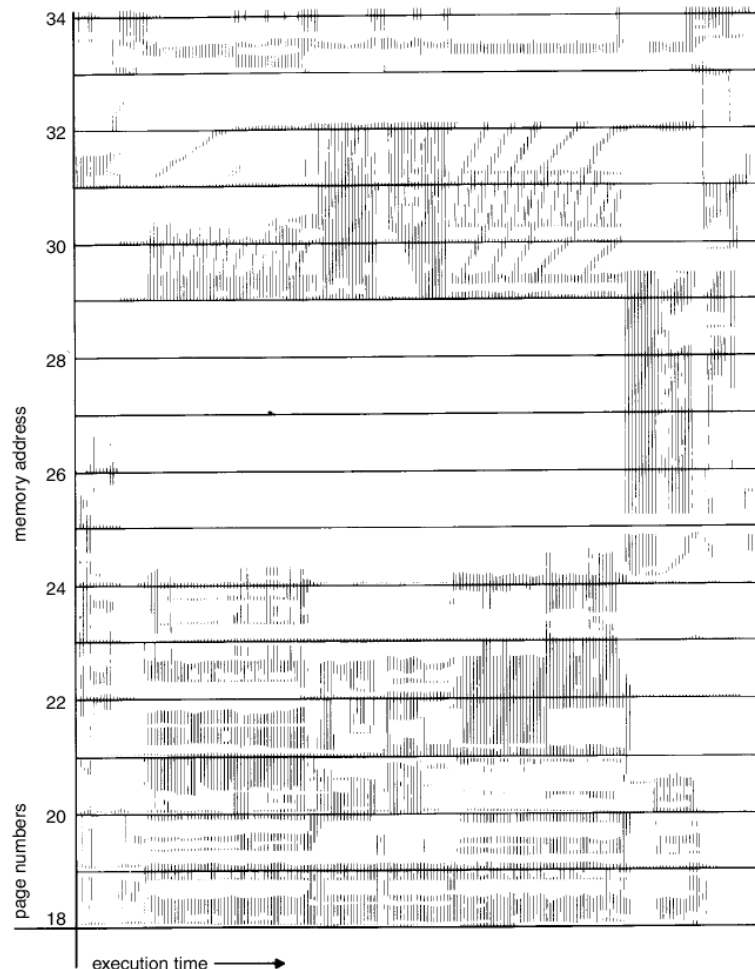
Thrashing \equiv i processi perdono piu' tempo per la paginazione che per l'esecuzione

Diagramma di utilizzazione CPU



- L'utilizzo della CPU *cresce al crescere del grado di multiprogrammazione* fino ad una soglia
- Superata questa soglia i processi tenderanno a generare una quantità sempre crescente di page fault e quindi *staranno tutti accodati ad attendere il pager* e l'utilizzo della CPU scenderà molto rapidamente

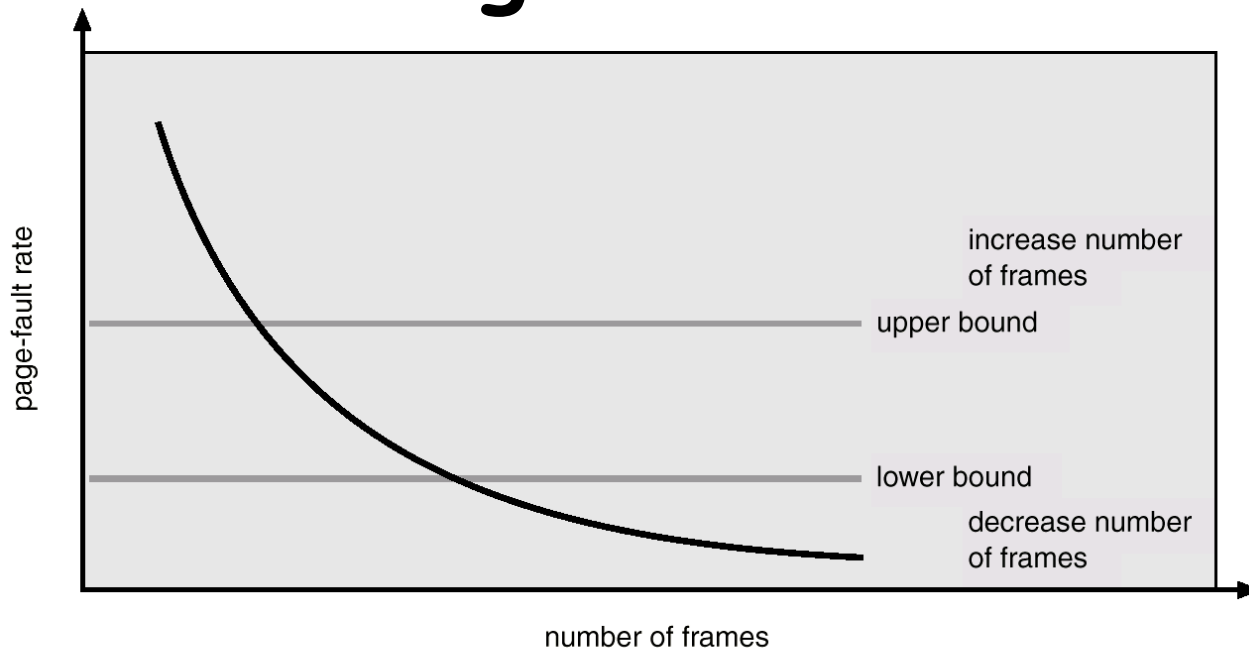
Come ci aiuta il concetto di *localita'* di un processo



Basarsi sulla *localita'* dei processi per prevedere il numero di pagine che occorrono onde evitare sostituzioni frequenti

Il numero di frame allocati ad un processo dovrebbe variare nel tempo

Modello a controllo di frequenza dei Page-Fault

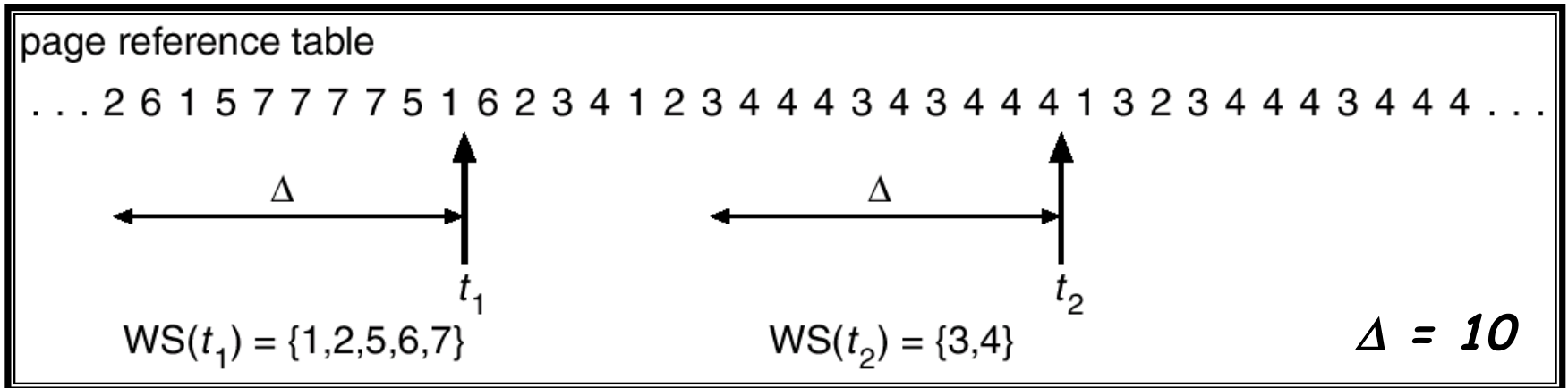


Si stabilisce un intervallo di *frequenze di page-fault accettabile* :

- Se la frequenza corrente e' troppo bassa il processo perde frame
- Se la frequenza corrente e' troppo alta il processo guadagna frame

Modello Working-Set

- $\Delta \equiv$ *finestra* di working-set \equiv numero prefissato di riferimenti di pagina, per esempio 10 riferimenti
- *Working set* del processo P_i = lista di pagine referenziate negli ultimi Δ riferimenti
- WSS_i = *taglia del working set* di P_i (puo' variare nel tempo pur non variando Δ !)



Qualche considerazione sul modello

La scelta di Δ

Δ *troppo piccolo* : non include tutta la corrente localita'

Δ *grande* : sovrappone differenti localita'

$\Delta = \infty$: include l'intera sequenza di riferimenti del processo

Come tenere traccia del Working Set

- Δ e' una finestra in movimento \Rightarrow si approssima
- *Bit di riferimento + timer*
- *Esempio* : $\Delta = 10,000$ riferimenti
 - Il timer interrompe ogni 5000 riferimenti
 - In memoria 2 bits supplementari per ogni pagina
 - Ad ogni interrupt del timer si copiano i bit di riferimento di ogni pagina in uno di quelli supplementari e si resettano a 0
 - Tutte le pagine che hanno almeno uno dei 2 bits a 1 stanno nel working set
- Si puo' migliorare la tecnica *aumentando il numero di bit supplementari e rendendo piu' frequenti le interruzioni del timer* (ex. 10 bits e interruzioni ogni 1000 riferimenti)