

Sistemi Operativi 1

AA 2018/2019

Memoria Virtuale

Sommario

- Introduzione
- Paginazione su domanda
 - Rimpiazzamento delle pagine
 - Allocazione dei frame

Introduzione

- Caratteristica degli schemi precedenti per la gestione della memoria:
 - l'intero programma deve essere caricato in memoria per essere eseguito
 - In generale, questo non è strettamente necessario
 - Solo una parte del programma può essere in memoria
- Conseguenze:
 - Lo spazio degli indirizzi logici può quindi essere molto più grande dello spazio di indirizzi fisici
 - Più processi possono essere mantenuti in memoria

Introduzione

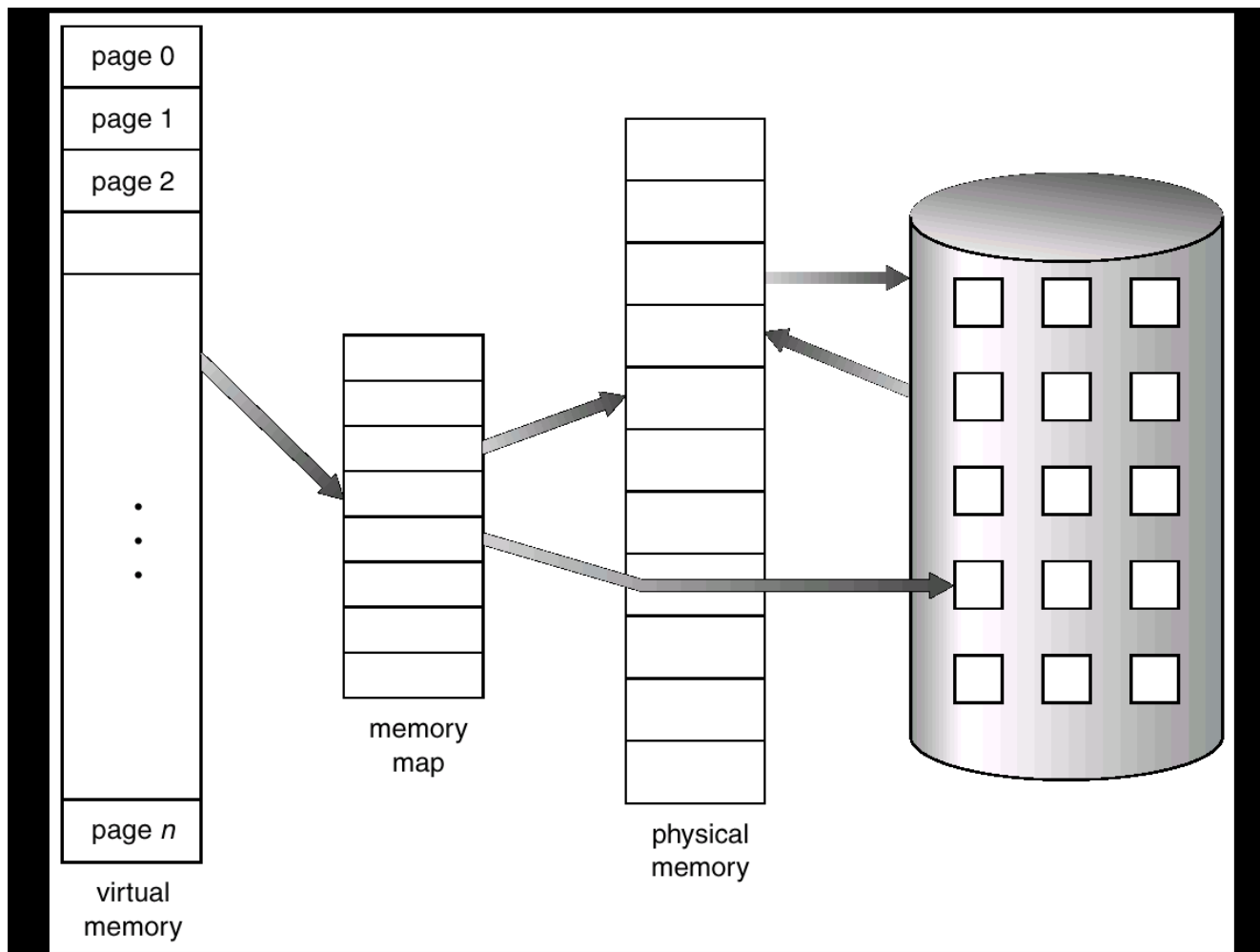
- Concetto chiave:
 - Possibilità di “swappare” pagine da e verso la memoria e non l'intero processo
 - La memoria virtuale permette separazione della memoria logica (utente) dalla memoria fisica
- Memoria virtuale = memoria fisica + disco
- Implementazione:
 - Paginazione su domanda (demand paging)
 - Segmentazione su domanda (demand segmentation)

PAGINAZIONE SU DOMANDA

Paginazione su domanda

- Principio:
 - Una pagina viene caricata in memoria solo quando necessario
- Vantaggi:
 - Meno richieste di I/O quando necessario swapping
 - Risposta più rapida
 - Meno memoria
 - Più processi hanno accesso alla memoria
- Fondamentale sapere lo stato di una pagina
 - In memoria oppure non in memoria?

Paginazione su domanda



Valid/invalid bit e page fault

- Qual è lo stato di una pagina?
 - Ad ogni entry della page table è associato un bit (valid/invalid)
 - 1 \Rightarrow in memoria,
 - 0 \Rightarrow non in memoria
 - Inizialmente sono tutti 0
 - Se durante la traduzione indirizzo logico/indirizzo fisico una entry ha valid/invalid bit 0, si ha un *page fault*

Frame #	valid bit
	1
	1
	1
	1
	0
...	
	0
	0

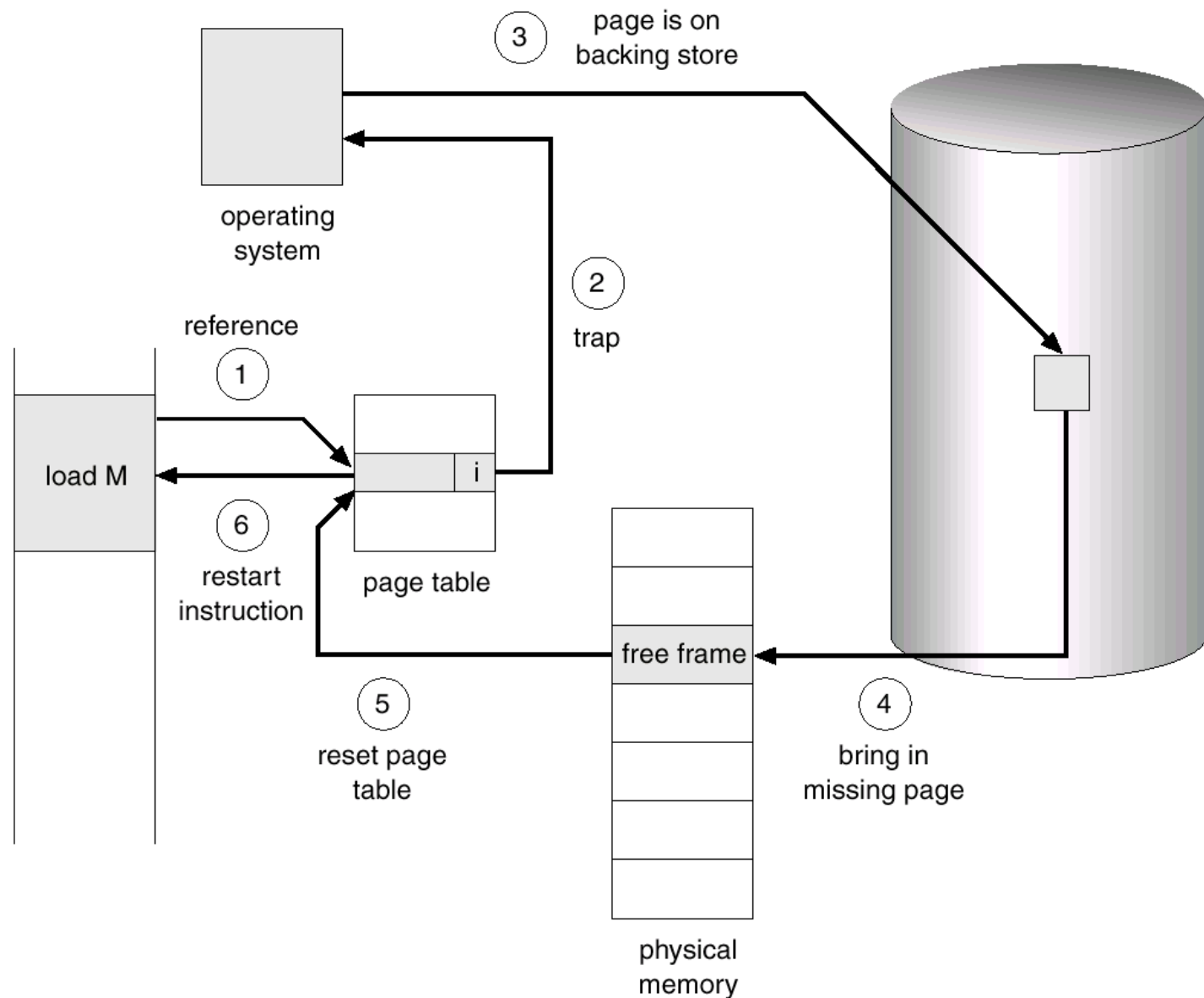
page table

Gestione dei page fault

- Page fault causa un interrupt al S.O.
 1. S.O. verifica una tabella (associata al processo)
 - Riferimento non valido \Rightarrow abort
 - Riferimento valido \Rightarrow attiva il caricamento della pagina
 2. Cerca un frame vuoto
 3. Swap della pagina nel frame (da disco)
 4. Modifica le tabelle
 - Page table: valid bit = 1
 - Tabella interna del processo: pagina in memoria
 5. Ripristina l'istruzione che ha causato il page fault
- NOTA: il primo accesso in memoria di un programma risulta essere sempre un page fault \rightarrow demand paging puro




Gestione dei page fault



Prestazioni della paginazione su domanda

- La paginazione su domanda influenza il tempo di accesso effettivo alla memoria (Effective Access Time, EAT)
- Tasso di page fault $0 \leq p \leq 1$
 - $p = 0$: nessun page fault
 - $p = 1$: ogni accesso è un page fault
- $EAT = (1 - p) * t_{\text{mem}} + p * t_{\text{page fault}}$


$$(T_{\text{MEM}} + T_{\text{TLB}}) * \alpha + (2 * T_{\text{MEM}} + T_{\text{TLB}}) * (1 - \alpha) \quad \text{es. Paginazione,}$$

Prestazioni della paginazione su domanda

- $T_{\text{page_fault}}$ è dato da 3 componenti principali:
 - servizio dell'interrupt
 - swap in (lettura della pagina)
 - costo del riavvio del processo
 - [swap out opzionale]

Prestazioni - esempio

- $t_{\text{mem}} = 100\text{ns}$
- $t_{\text{page fault}} = 1\text{ ms } (10^6\text{ ns})$
- $\text{EAT} = (1-p) * 100 + p * 10^6 = 100 - 100 * p + 1000000 * p = 100 * (1 + 9999p)\text{ns}$
- Per mantenere il peggioramento entro il 10% rispetto al tempo di accesso standard:
 - $100 * (1.1) > 100 * (1 + 9999p) \rightarrow p < 0.0001 \approx 10^{-4}$
 - 1 page fault ogni 10000 accessi!
- Fondamentale tenere basso il livello di page fault



Rimpiazzamento delle pagine

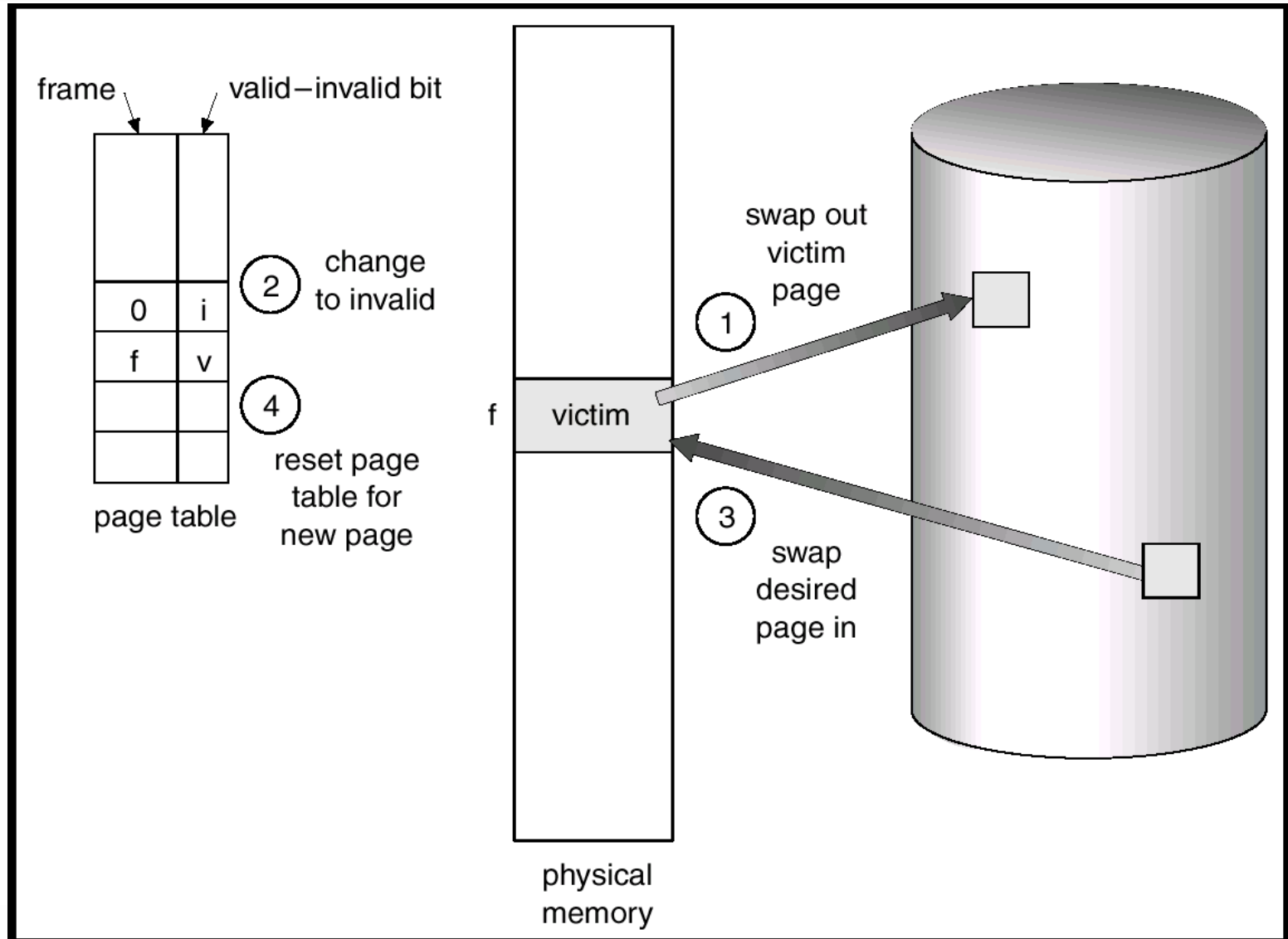
- Cosa succede se non ci sono pagine libere?
- Rimpiazzamento delle pagine:
 - Cerca pagine (frame) in memoria
 - Swap su disco di queste pagine
- Realizzazione:
 - Richiede opportuno algoritmo
 - Obiettivo: ottimizzazione delle prestazioni

⇒ minimizzazione del # di page fault

Rimpiazzamento delle pagine

- Gestione dei page fault in caso di assenza di frame liberi:
 1. S.O. verifica una tabella (associata al processo) per capire se si tratta di page fault o violazione di accesso
 2. Cerca un frame vuoto
 - Se esiste, OK (salta a 4)
 - Se non c'è, si usa un algoritmo di rimpiazzamento delle pagine per scegliere un frame vittima
 3. Swap della vittima su disco
 4. Swap della pagina nel frame da disco
 5. Modifica le tabelle (page table, bit validità)
 6. Ripristina l'istruzione che ha causato il page fault

Rimpiazzamento delle pagine



Rimpiazzamento delle pagine

- In assenza frame liberi, sono necessari due accessi alla memoria:
 - Uno per swap out “vittima”
 - Uno per swap in frame da caricare
- Risultato:
 - tempo di page fault raddoppiato!
- Ottimizzazione:
 - usare un bit nella page table (bit di modifica - dirty bit)
 - Messo a 1 se la pagina è stata modificata (scrittura) dal momento in cui viene caricata
 - Solo le pagine che risultano modificate (dirty=1) vengono scritte su disco quando diventano “vittime”

Paginazione su domanda

- Problematiche
 - Rimpiazzamento delle pagine
 - Quale pagina rimpiazzare?
 - Allocazione dei frame
 - Quanti frame assegnare ad un processo al momento dell'esecuzione?

ALGORITMI DI RIMPIAZZAMENTO DELLE PAGINE

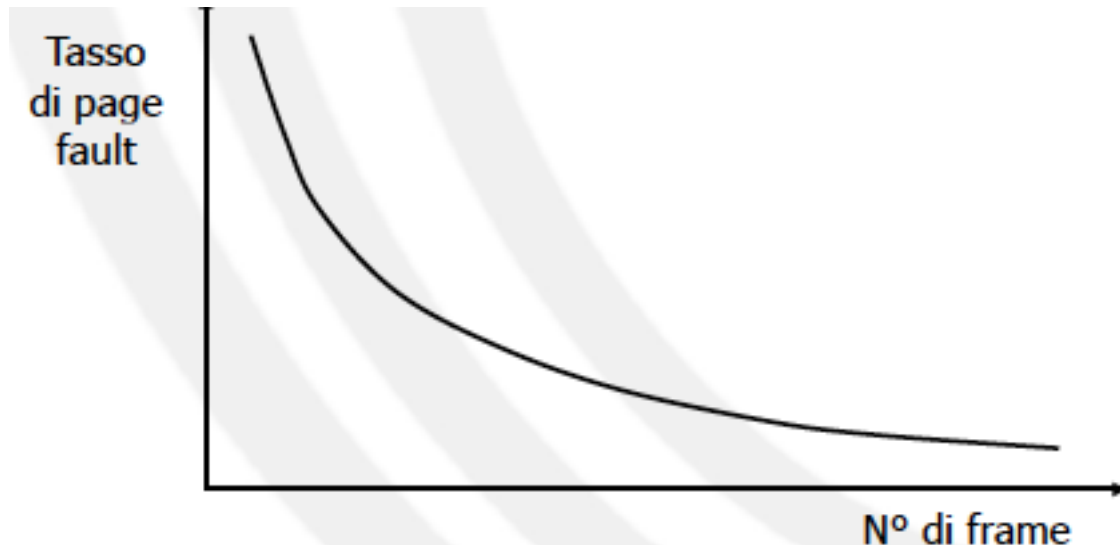
Algoritmi di rimpiazzamento delle pagine

- Obiettivo = minimo tasso di page fault
- Valutazione:
 - Esecuzione di una particolare stringa di riferimenti a memoria (reference string)
 - Calcolo del # di page fault sulla stringa
 - Necessario sapere il # di frame disponibili per il processo
- Esempio (dimensione pagina = 100 byte)
 - Indirizzi: 100, 604, 128, 130, 256, 260, 264, 268
 - Reference string: 1, 6, 1, 1, 2, 2, 2, 2
 - In realtà la reference string è solo: 1, 6, 1, 2
 - Accessi consecutivi alla stessa pagina causano al massimo 1 page fault



Algoritmi di rimpiazzamento delle pagine

- Tasso di page fault è inversamente proporzionale al numero di frame

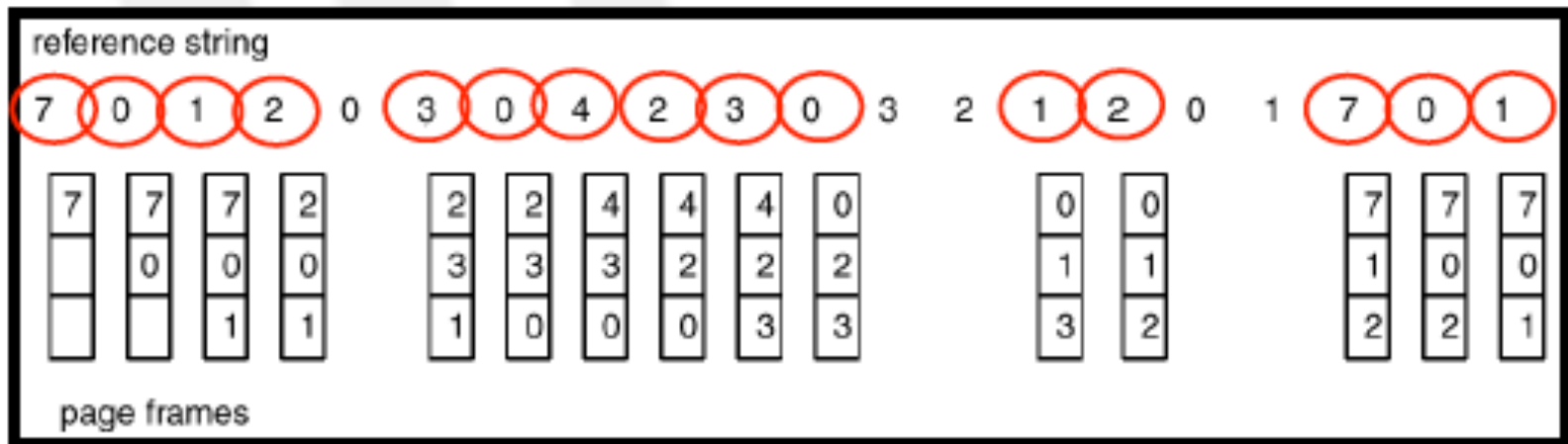


Algoritmo FIFO (First-In-First-Out)

- FIFO = prima pagina introdotta è la prima ad essere rimossa
- Algoritmo “cieco”: non viene valutata l'importanza della pagina rimossa
 - **Importanza = frequenza di riferimento**
- Tende ad aumentare il tasso di page fault
- Soffre dell'anomalia di Belady

Algoritmo FIFO - esempio

- Consideriamo una memoria con 3 frame
- Usando FIFO si hanno 15 page fault



Anomalia di Belady - esempio

- Reference string

1	2	3	4	1	2	5	1	2	3	4	5
---	---	---	---	---	---	---	---	---	---	---	---

- Con 3 frame 9 page fault

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

Anomalia di Belady

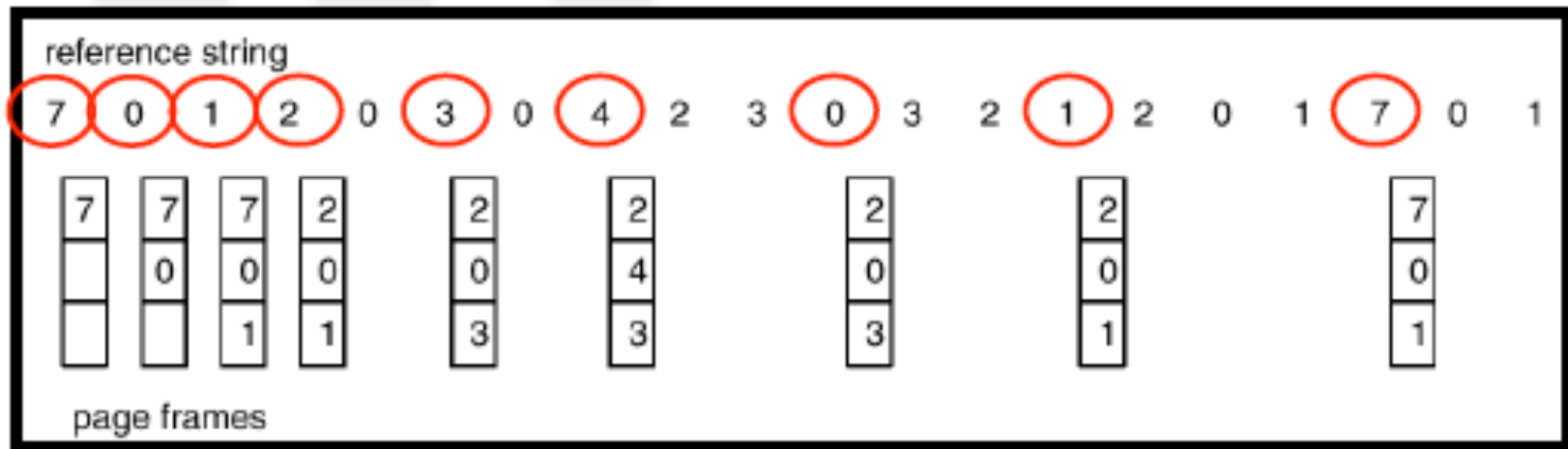
- # di page fault può non decrescere all'aumentare del numero di frame usando FIFO
 - A volte più frames \Rightarrow più page fault

Algoritmo ideale

- Garantisce minimo numero di page fault
- Idea: rimpiazza le pagine che non saranno usate per il periodo di tempo più lungo
- Problema: come ricavare questa informazione?
 - Richiede conoscenza anticipata della stringa dei riferimenti
 - Situazione simile a SJF
 - Implementazione impossibile
 - Possibili approssimazioni
- Utile come riferimento per altri algoritmi

Algoritmo ideale - esempio

- Con l'algoritmo ideale abbiamo 9 page fault



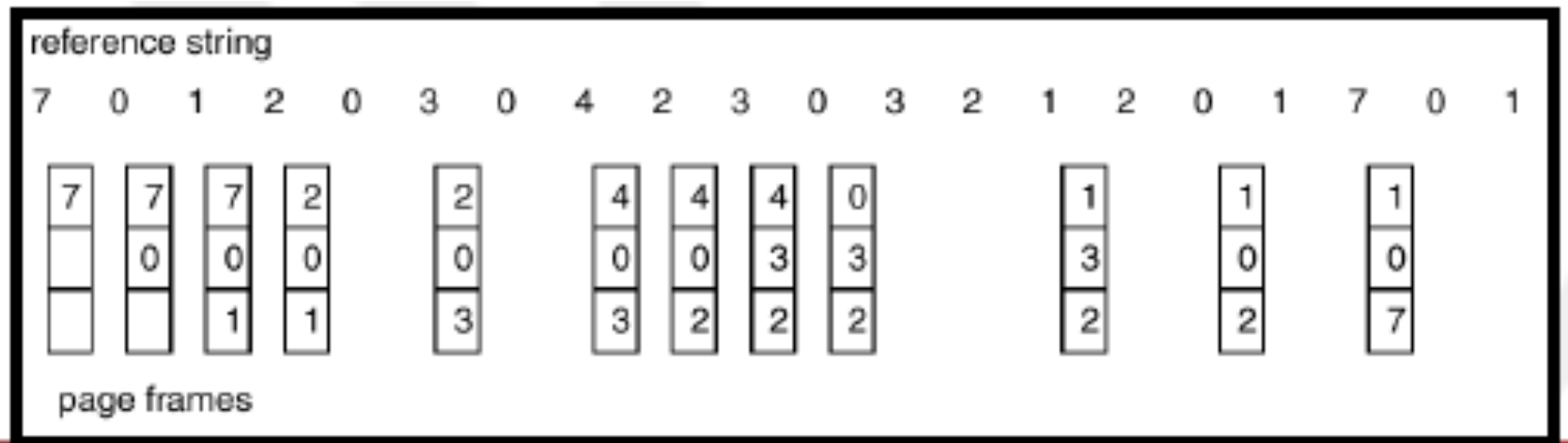
Algoritmo LRU – (Least Recently Used)

- Approssimazione dell'algoritmo ottimo
 - Usare il passato recente come previsione del futuro
 - Si rimpiazza la pagina che non viene usata da più tempo (*)
- Esempio: 4 frame → 8 page fault

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

Algoritmo LRU - esempio

- Con l'algoritmo LRU abbiamo 12 page fault
 - Migliore del FIFO
 - Peggior dell'ideale



Algoritmo LRU

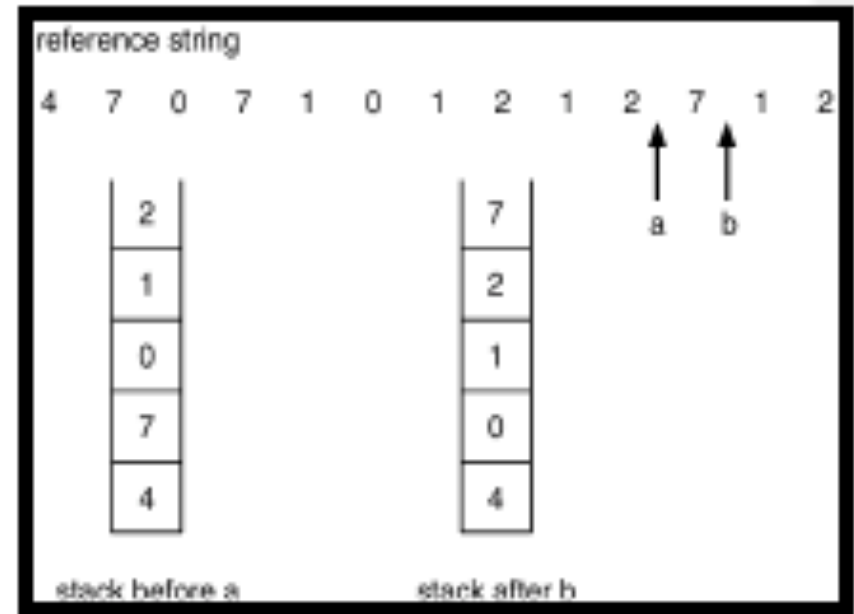
- Problema: implementazione?
 - Non banale ricavare il tempo dell'ultimo utilizzo
 - Può richiedere notevole HW aggiuntionale

Algoritmo LRU - implementazioni

- Tramite contatore
 - Ad ogni pagina è associato un contatore
 - Ogni volta che la pagina viene referenziata, il clock di sistema è copiato nel contatore
 - Rimpiazza la pagina con il valore più piccolo del contatore
 - Bisogna cercarla!

Algoritmo LRU - implementazioni

- Tramite stack
 - Viene mantenuto uno stack di numeri di pagina
 - Ad ogni riferimento ad una pagina, questa viene messa in cima allo stack
 - L'aggiornamento richiede estrazione di un elemento interno allo stack
 - Fondo dello stack = pagina LRU
 - Nessuna ricerca della pagina da rimpiazzare!



Modifica stack o copia del
tempo di sistema richiede
supporto HW!

Algoritmo LRU - implementazioni

- Uso del bit di reference
 - Associato ad ogni pagina, inizialmente = 0
 - Quando la pagina è referenziata, messo a 1 dall'HW
 - Rimpiazzamento: sceglie una pagina che ha il bit a 0
 - Approssimato: non viene verificato l'ordine di riferimento delle pagine (chi è stato riferito prima?)
- Alternativa
 - Uso più bit di reference (registro di scorrimento) per ogni pagina
 - Bit aggiornati periodicamente (es. ogni 100ms)
 - Uso dei bit come valore intero per scegliere la LRU
 - Pagina LRU = pagina con valore del registro di scorrimento + basso

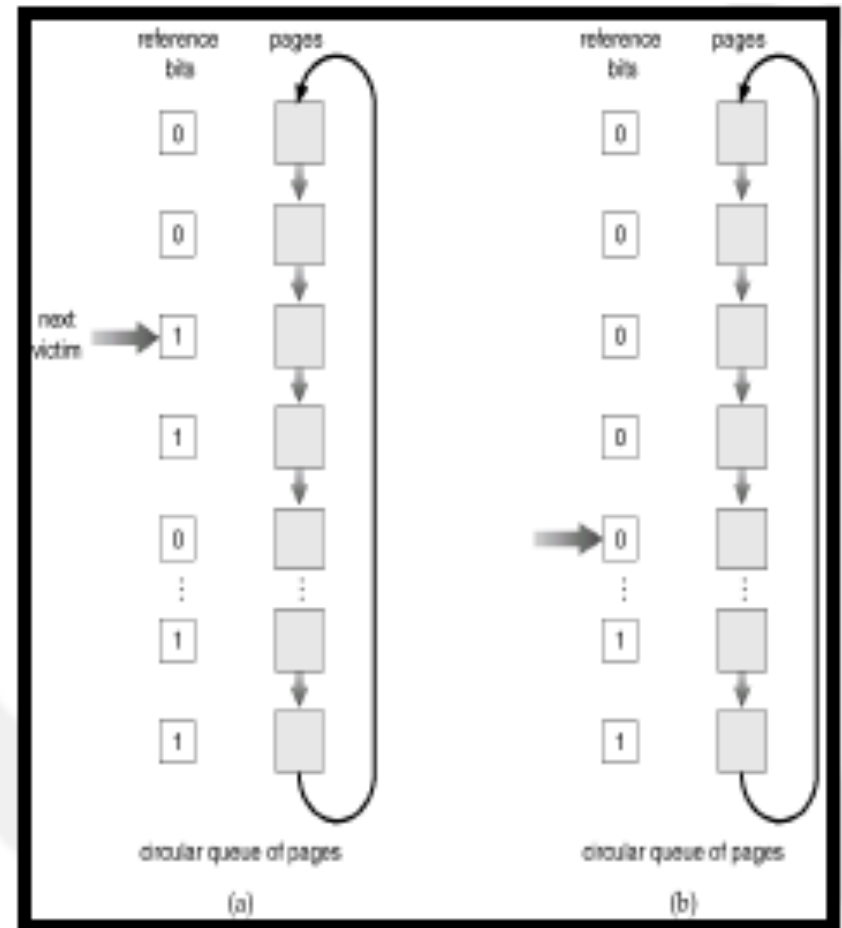


Algoritmo LRU - approssimazioni

- Tecniche basate su conteggio
 - Algoritmo LFU (Least Frequently Used)
 - Mantiene un conteggio del # di riferimenti fatti ad ogni pagina
 - Rimpiazza la pagina con il conteggio più basso
 - Può non corrispondere a pagina “LRU”
 - Es.: se ho molti riferimenti iniziali, una pagina può avere conteggio alto e non essere eseguita da molto tempo
 - Algoritmo MFU (Most Frequently Used)
 - Opposto di LFU
 - La pagina con il conteggio più basso è probabilmente stata appena caricata e dovrà essere presumibilmente usata ancora (località dei riferimenti)

Algoritmo LRU - approssimazioni

- Rimpiazzamento second chance (clock)
 - Principio base = FIFO circolare
 - Basato su bit di reference
 - Bit a 0 → rimpiazza
 - Bit a 1 →
 - Metti a 0, ma lascia la pagina in memoria
 - Analizza la pagina successiva (in ordine circolare) usando la stessa regola
- Varianti
 - Più bit di reference



ALLOCAZIONE DEI FRAME

Allocazione dei frame

- Data una memoria con N frame e M processi è importante scegliere bene quanti frame allocare ad ogni processo
- Vincoli:
 - Ogni processo necessita di un minimo numero di pagine per poter essere eseguito
 - Dettato dal fatto che l'istruzione interrotta da un page fault deve essere fatta ripartire
 - Conseguenza: # minimo di pagine = massimo numero di indirizzi specificabile in una istruzione

Allocazione dei frame

- Esempio:
 - IBM 370: 6 pagine per gestire l'istruzione move
 - Istruzione: 6 byte, può coinvolgere due pagine
 - 2 pagine per gestire la sorgente
 - 2 pagine per gestire la destinazione
- Valori tipici: 2...4 frame
- Schemi di allocazione dei frame:
 - Fissa
 - Un processo ha sempre lo stesso numero di frame
 - Variabile
 - Il numero di frame allocati a un processo può variare durante l'esecuzione

Contesto del rimpiazzamento

- In caso di page fault dove si scelgono le vittime?
 - Rimpiazzamento locale
 - ogni processo seleziona vittime solo tra i frame suoi
 - Rimpiazzamento globale
 - un processo sceglie un frame dall'insieme di tutti i frame
 - un processo può prendere frame di un altro processo
 - Migliora throughput → + usato del rimpiazzamento locale

<i>Contesto</i> <i>Allocazione</i>	Locale	Globale
Fissa	X	No
Variabile	X	X

Allocazione fissa

- Allocazione in parti uguali
 - Dati m frame e n processi, alloca ad ogni processo m/n frame
- Allocazione proporzionale
 - Alloca secondo la dimensione del processo
 - Può non essere un parametro significativo!
 - La priorità di un processo può essere più significativa della sua dimensione

s_i = size of process p_i

$$S = \sum s_i$$

m = total number of frames

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

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

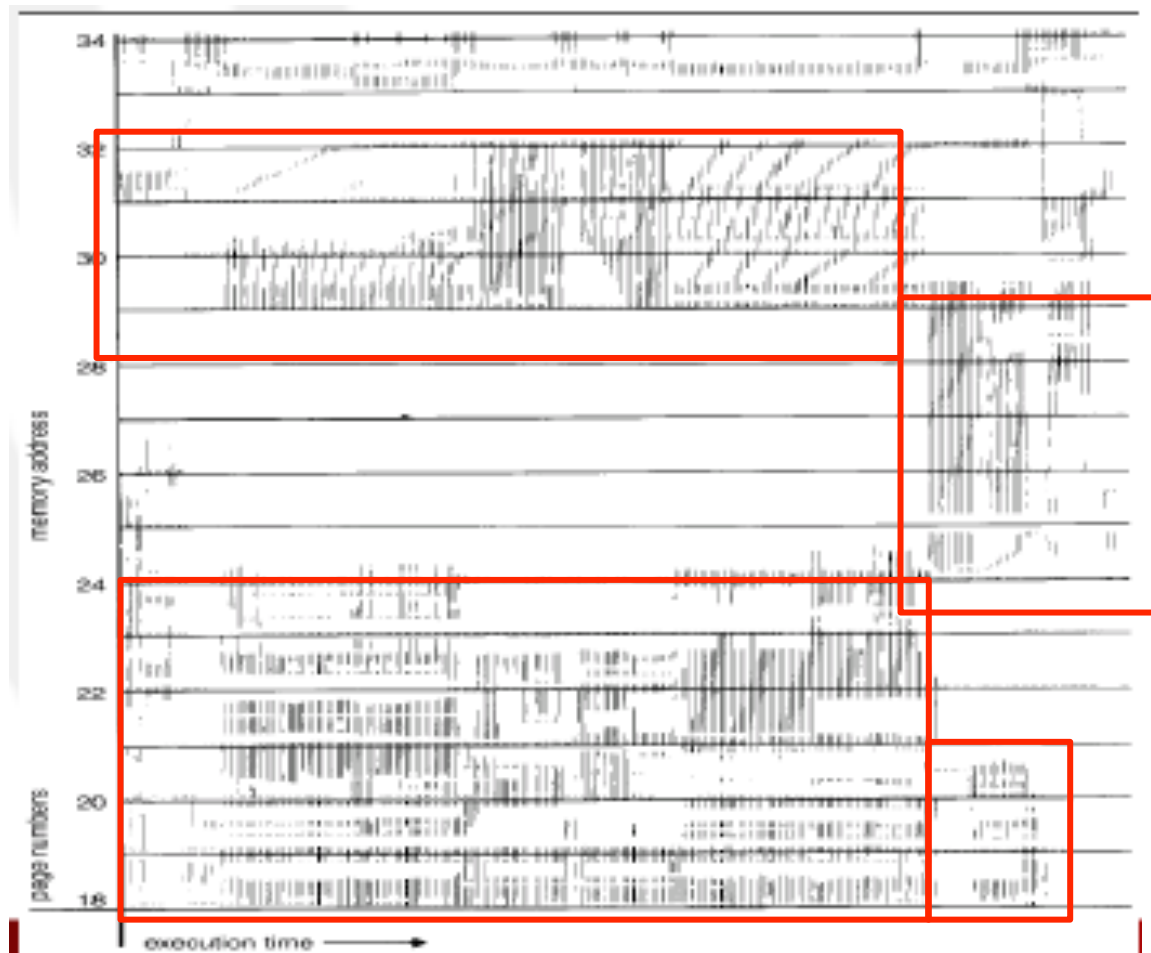
$$a_1 = \frac{10}{137} \times 64 \approx 5$$

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

Allocazione variabile

- Permette di modificare dinamicamente le allocazioni ai vari processi
- Problema
 - In base a cosa modifico?
- Due soluzioni
 - Calcolo del *working set*
 - Calcolo del *page fault frequency* (PFF)

Località di esecuzione - esempio

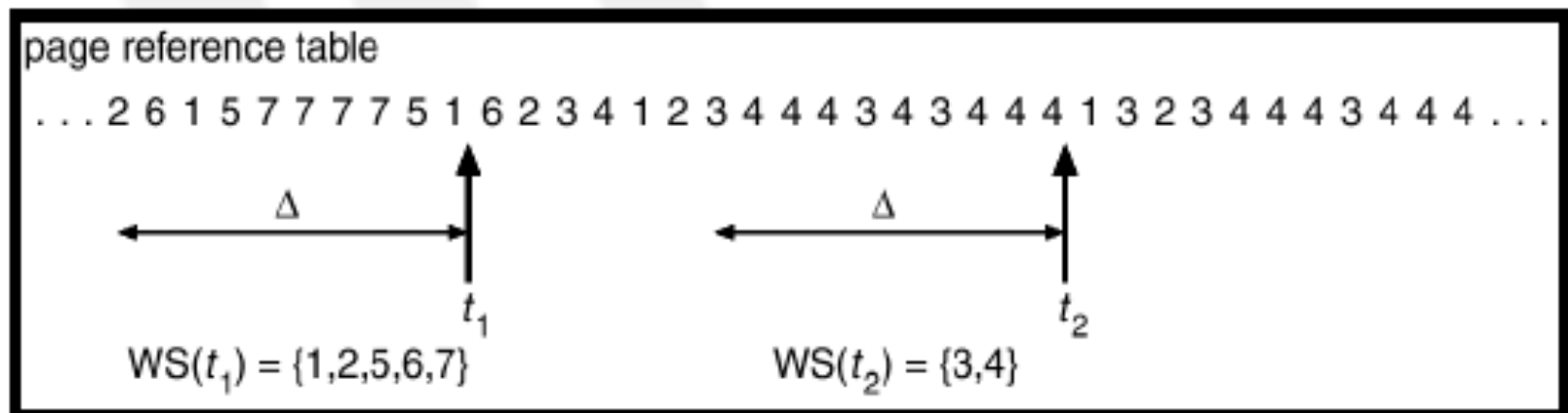


Calcolo del working set

- Un criterio per rimodulare l'allocazione dei frame consiste nel calcolare in qualche modo quali sono le richieste effettive di ogni processo
- In base al modello della località
 - Un processo passa da una località (di indirizzi) all'altra durante la sua esecuzione
 - Esempio: array, procedure, moduli
- Idealmente
 - Un processo necessita di un numero di frame pari alla sua località
 - Come la misuro?

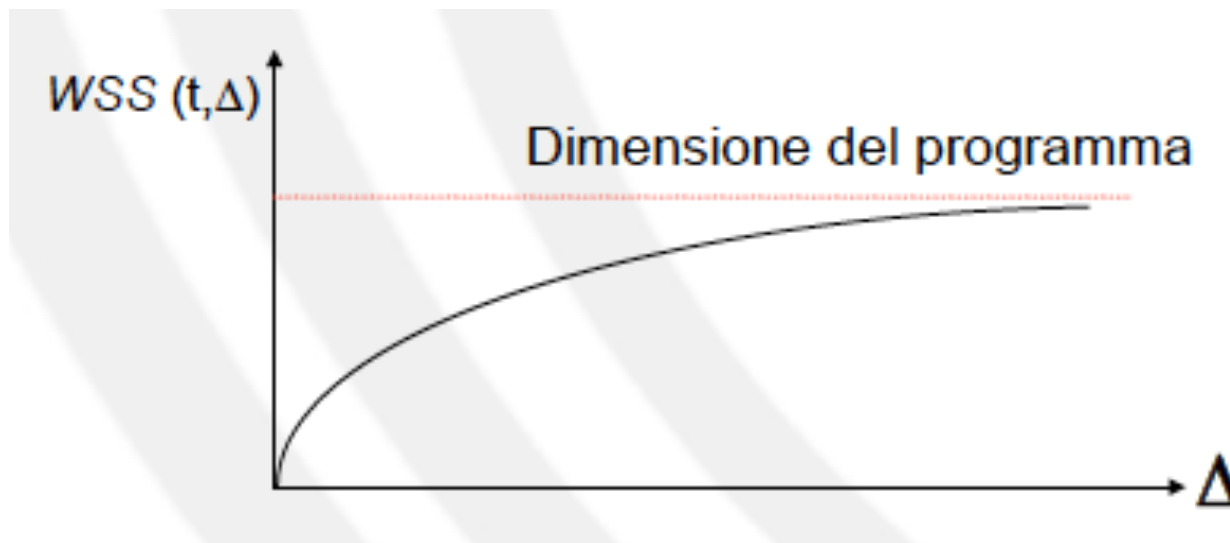
Modello del working set

- frame sufficiente a mantenere in memoria il suo working set $WS_i(t, \Delta)$
 - Numero di pagine referenziate nell'intervallo di tempo $[t-\Delta, t]$ più recente (finestra del working set)
 - se Δ piccolo = poco significativo
 - se Δ troppo grande = può coprire varie località
 - $\Delta = \infty \Rightarrow$ tutto il programma



Modello del working set

- $WSS_i(t, \Delta)$ = dimensione di $WS_i(t, \Delta)$ in funzione del tempo



Modello del working set

- Allora come misuro il working set?
 - Approssimazione tramite timer e bit di reference
 - Uso di un timer che interrompe periodicamente la CPU
 - All'inizio di ogni periodo, bit di reference posti a 0
 - Ad ogni interruzione del timer, le pagine vengono scandite
 - Quelle con bit di reference = 1 \Rightarrow sono nel working set
 - Quelle con bit di reference = 0 \Rightarrow vengono scartate
 - Accuratezza aumenta in base al n° di bit e alla frequenza delle interruzioni

Modello del working set

- Richiesta totale di frame
 - $D = \sum_i WSS_i$
- Cosa succede se $D >$ numero totale di frame?
 - Si verifica un fenomeno noto come thrashing
 - Un processo spende tempo di CPU continuando a “swappare” pagine da e verso la memoria
 - Conseguenza di un basso numero di frame
 - Risultato di un “circolo vizioso”

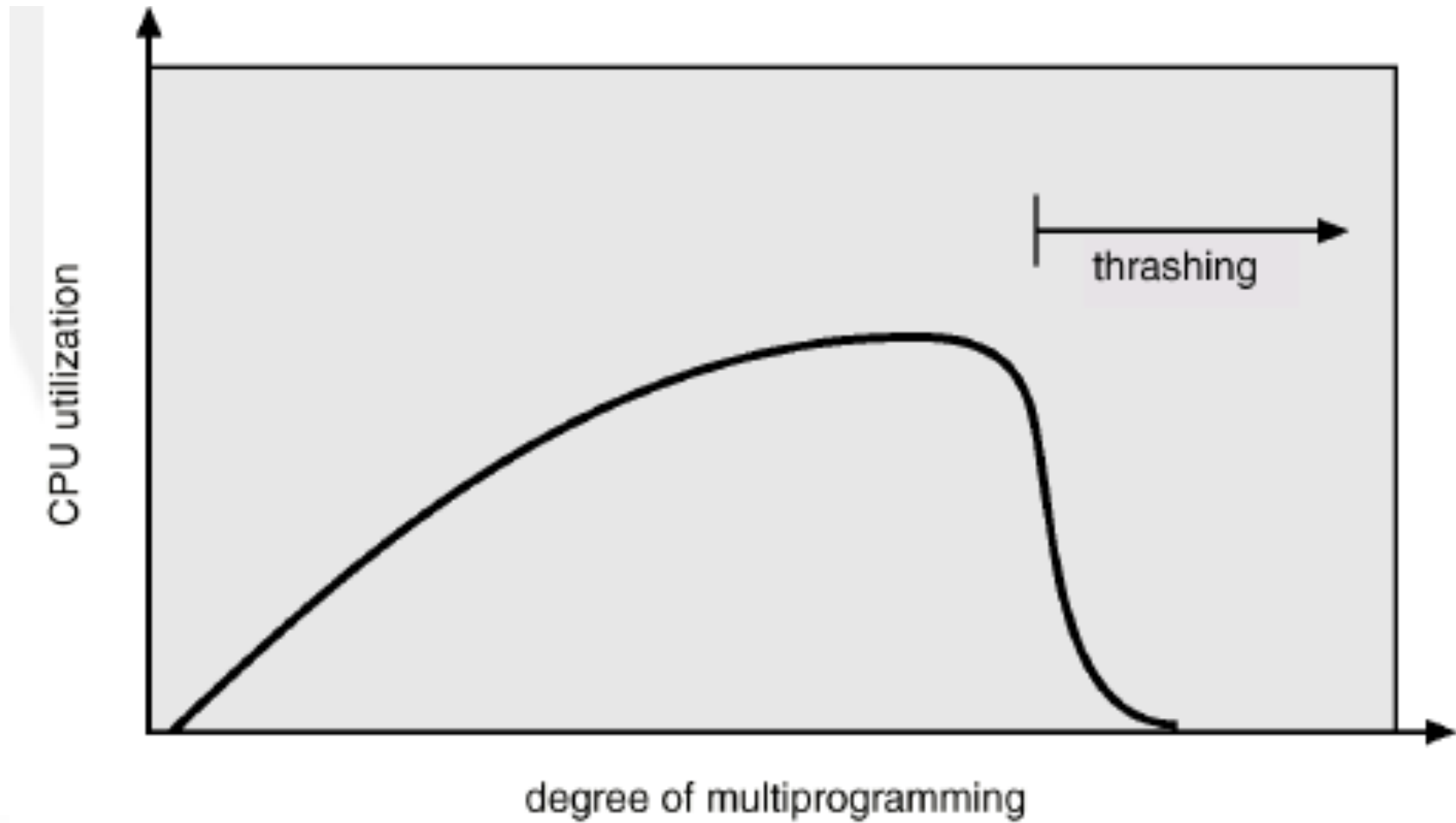
Thrashing

- Se il numero di frame allocati ad un processo scende sotto un certo minimo, il tasso di page-fault tende a crescere
- Questo porta a:
 - abbassamento dell'utilizzo della CPU dovuto al fatto che alcuni processi sono in attesa di gestire il page fault
 - Il S.O. tende ad aumentare il grado di multiprogrammazione aggiungendo processi
 - I nuovi processi “rubano” frame ai vecchi processi → grado di page fault aumenta ulteriormente
 - Ad un certo punto il throughput precipita!
 - Bisogna stimare con esattezza il numero di frame necessari a un processo per non entrare in thrashing

Circolo vizioso

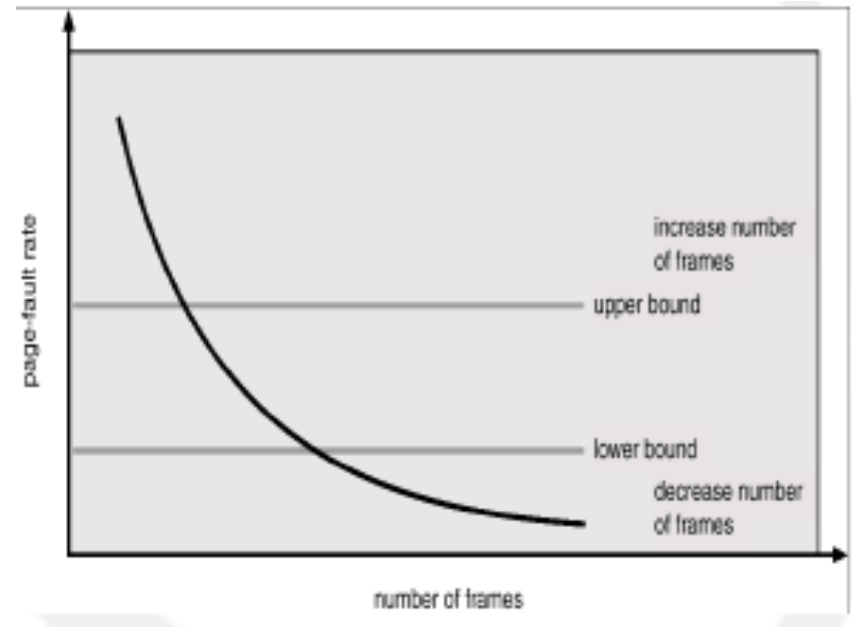


Diagramma del thrashing



Frequenza dei page fault

- Soluzione alternativa (più accurata)
 - Stabilire un tasso di page-fault “accettabile”
 - Se quello effettivo è troppo basso, il processo rilascia dei frame (ne ha troppi)
 - Se troppo alto, il processo ottiene più frame



Altre considerazioni

- Selezione della dimensione della pagina
 - Problemi/vincoli
 - Frammentazione → pagine piccole
 - pagine grandi = frammentazione interna significativa
 - Dimensione page table → pagine grandi
 - pagine piccole = molte entry
 - I/O overhead → pagine grandi
 - pagina piccola = costo di lettura/scrittura non ammortizzato
 - Località → pagine piccole
 - pagine grandi = granularità grande, devo trasferire anche ciò che non è necessario

Altre considerazioni

- Struttura dei programmi influisce sul numero di page fault

- Esempio:

- Array A[1024,1024] of integer
 - Una riga memorizzata in una pagina
 - Un solo frame assegnato al processo

- Programma 1

```
for j := 1 to 1024 do
  for i := 1 to 1024 do
    A[i,j] := 0;
```

1024 x 1024 page fault

- Programma 2

```
for i := 1 to 1024 do
  for j := 1 to 1024 do
    A[i,j] := 0;
```

1024 page fault



Altre considerazioni

- Blocco di frame (frame locking)
 - In alcuni casi particolari, esistono frame che non devono essere (mai) rimpiazzati!
 - Frame corrispondenti a pagine del kernel
 - Frame corrispondenti a pagine usate per trasferire dati da/verso I/O