

Memoria virtuale

Anno Accademico 2020-2021
Graziano Pravadelli



UNIVERSITÀ
di **VERONA**

Dipartimento
di **INFORMATICA**

Sommario

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

Tutto in RAM?

Caratteristica degli schemi precedenti per la gestione della memoria

Intero programma caricato in memoria per essere eseguito



In generale, questo non è strettamente necessario

Solo una parte del programma può essere in memoria



Conseguentemente

Spazio indirizzi logici può essere molto più grande spazio
indirizzi fisici

Più processi possono essere mantenuti in memoria

Memoria virtuale

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
- Segmentazione su domanda

Paginazione su domanda



Paginazione su domanda – Caratteristiche

Principio

- Una pagina viene caricata in memoria solo quando necessario

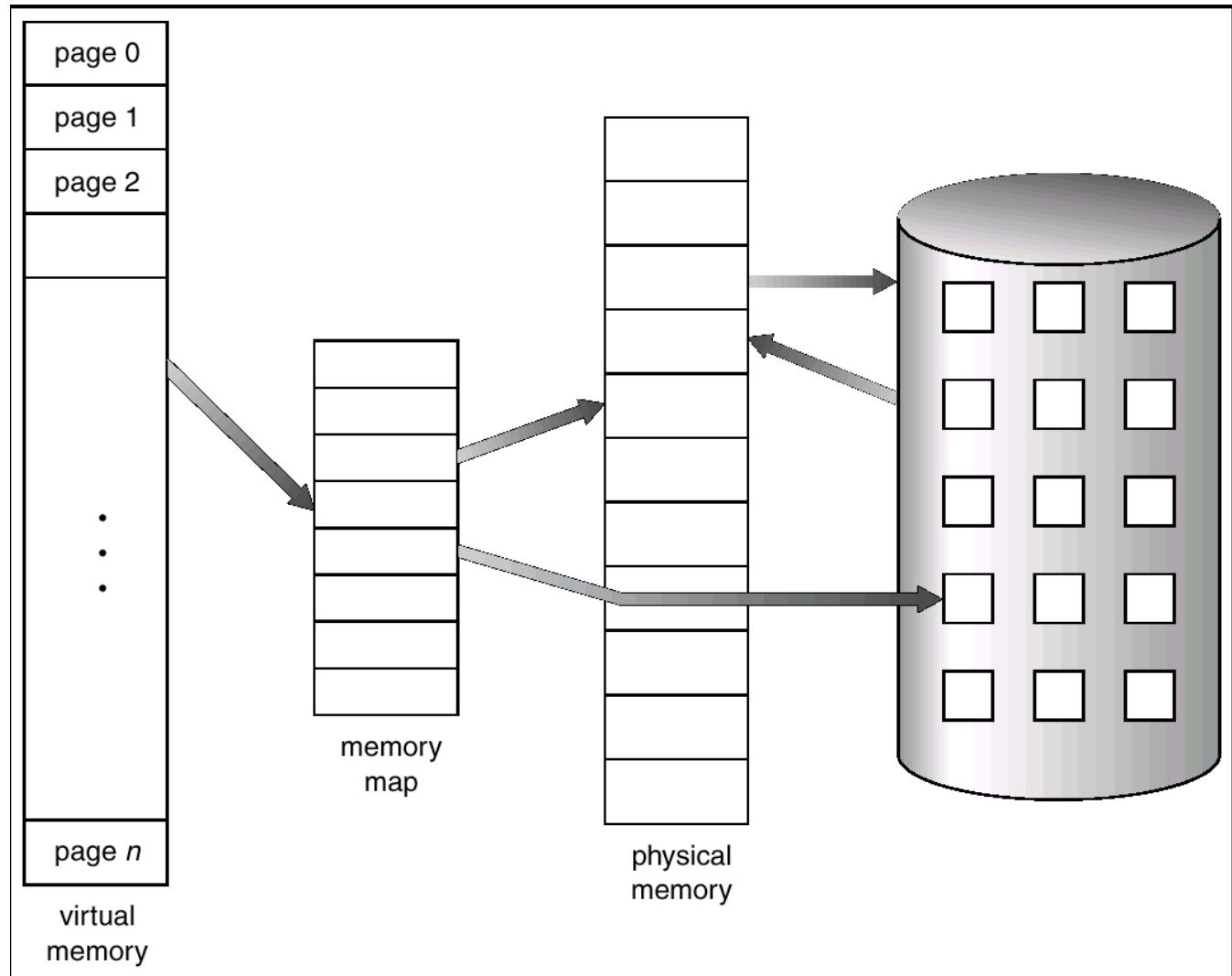
Vantaggi

- 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?

Schema concettuale



Page Fault

- Qual è lo stato di una pagina?
 - In ogni entry della page table, bit di validità
 - 1 ⇒ in memoria
 - 0 ⇒ non in memoria
 - Inizialmente tutti a 0
- Page fault
 - Quando durante la traduzione indirizzo logico/indirizzo fisico una entry ha bit di validità a 0

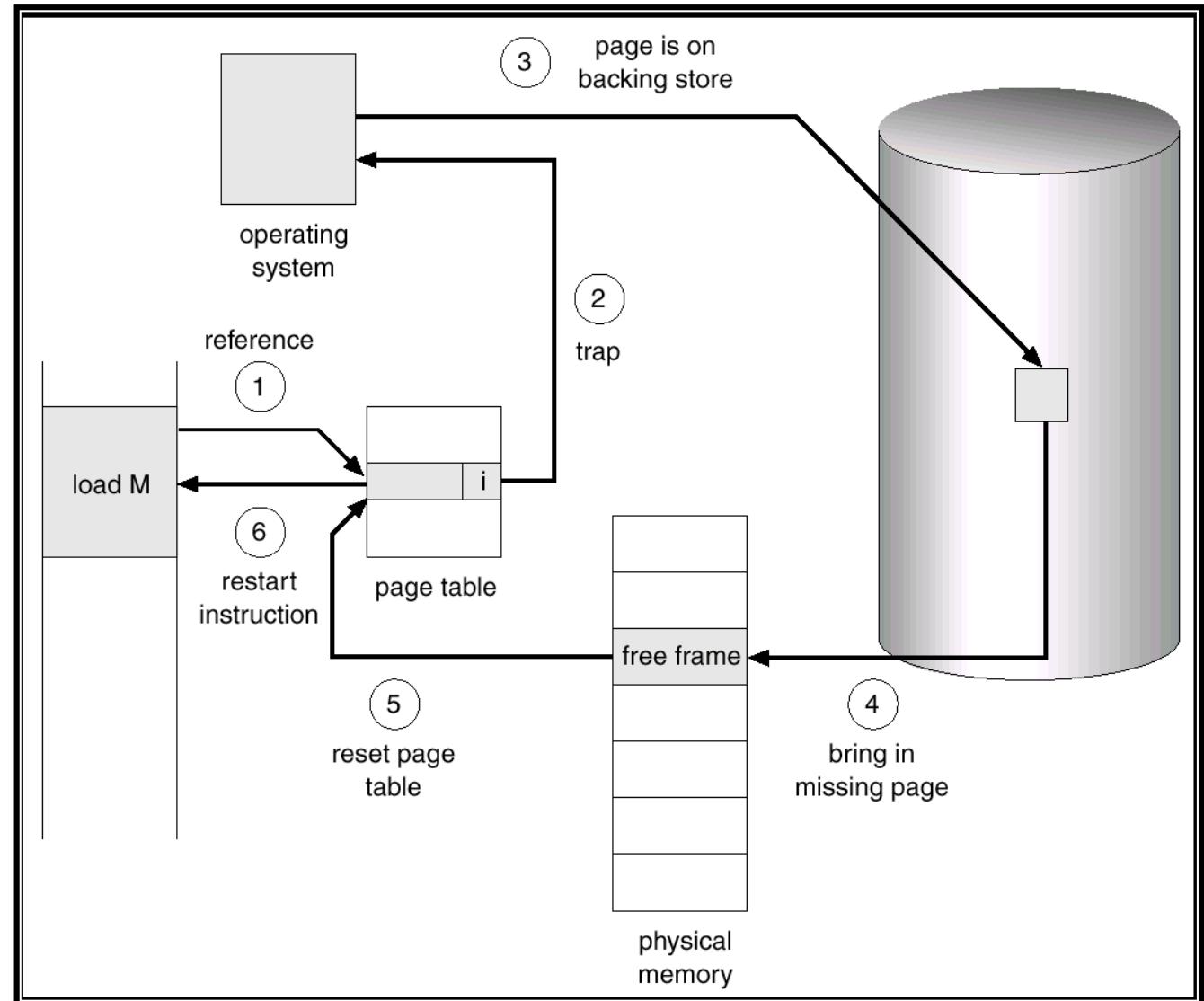
Frame #	Valid bit
	1
	1
	1
	1
	0
...	
	0
	0

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 tavole
 - Page table: valid bit = 1
 - Tabella interna del processo: pagina in memoria
 5. Ripristina l'istruzione che ha causato il page fault

Gestione dei Page Fault



Prestazioni – Impatto su EAT

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

Prestazioni – Costo del page fault

$t_{\text{page fault}}$ è dato da 3 componenti principali:

- Servizio dell'interrupt
- Swap in (lettura della pagina)
- Costo del riavvio del processo
- [Swap out opzionale]

Prestazioni – Esempio

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

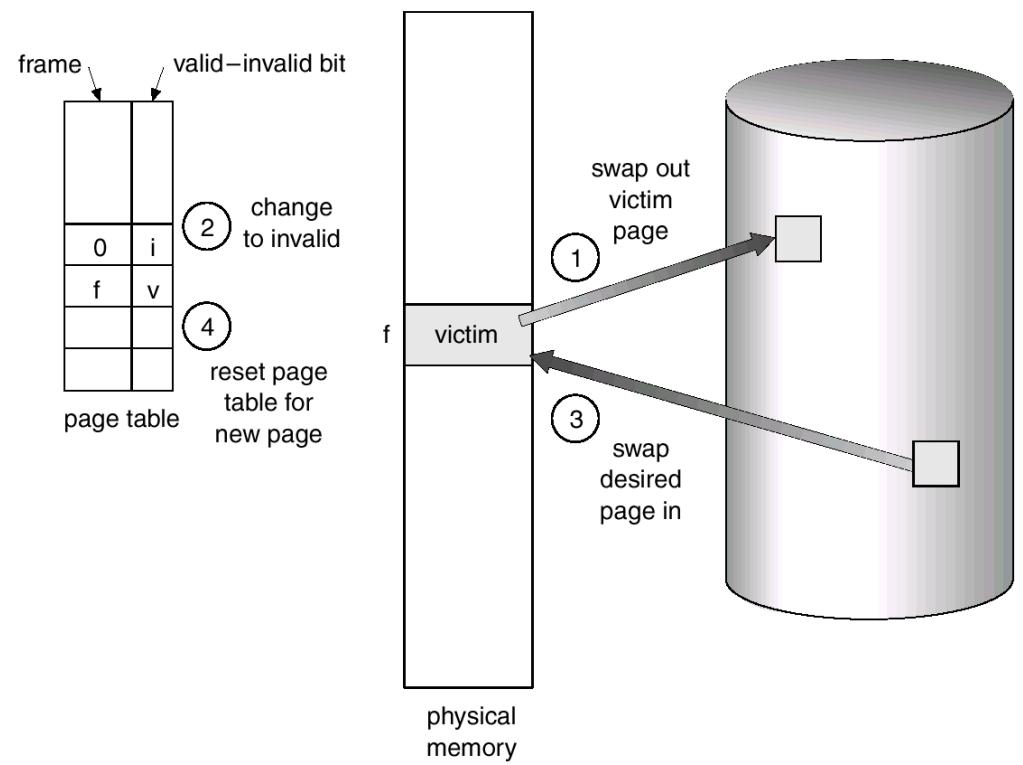
Richiede opportuno algoritmo

Obiettivo: ottimizzazione prestazioni
⇒ minimizzazione # di page fault

Gestione dei Page Fault

- In caso di assenza di frame liberi:
 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**
 - Se esiste, salta a 4
 - Se non esiste, usare algoritmo di rimpiazzamento delle pagine per scegliere 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



Prestazioni – Rimpiazzamento 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

- Bit di modifica (*dirty bit*) nella page table:
 - 1 se la pagina è stata modificata (scrittura) dal momento in cui viene caricata
 - Solo pagine con *dirty bit* = 1, scritte su disco quando diventano “vittime”

Paginazione su domanda – Problematiche

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

Algoritmi di rimpiazzamento delle pagine



Introduzione

Obiettivo = Minimizzazione tasso di page fault

Valutazione

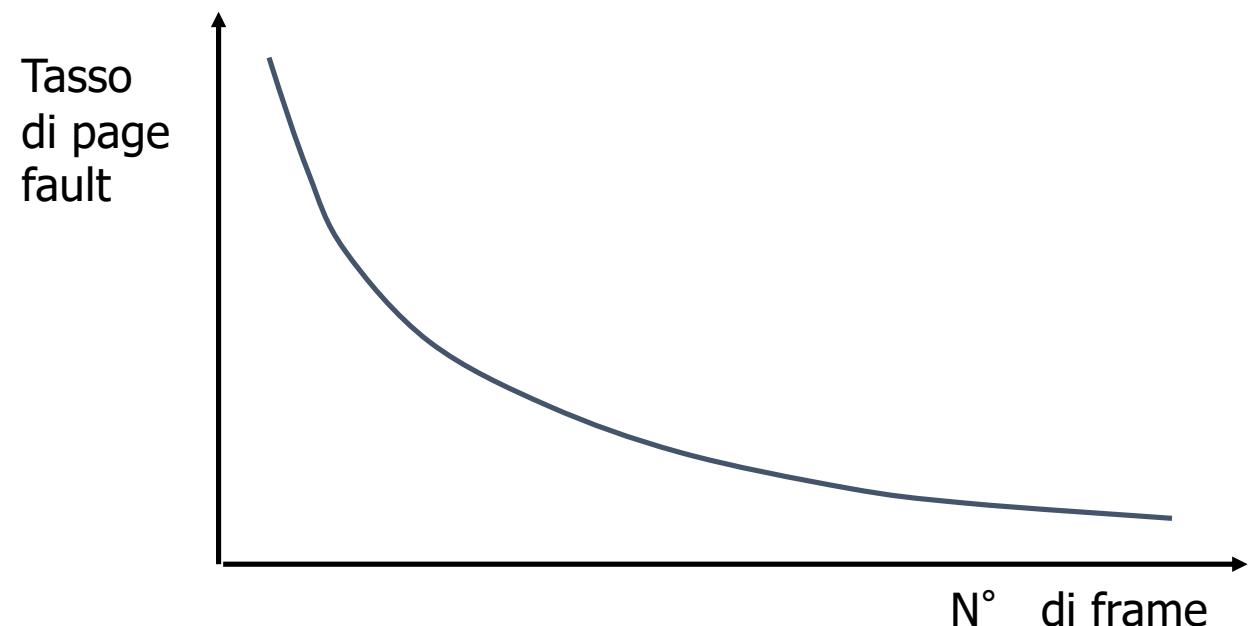
- Esecuzione di una particolare stringa di riferimenti a memoria (*reference string*)
- Calcolo # page fault sulla stringa
- Necessario sapere # frame disponibili per il processo

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

Page fault vs. Numero frame?

- Inversamente proporzionale



Algoritmo FIFO

FIFO = prima pagina introdotta è la prima ad essere rimossa

Algoritmo “cieco”

- Non valutata l'importanza della pagina rimossa
- Importanza = frequenza di riferimento

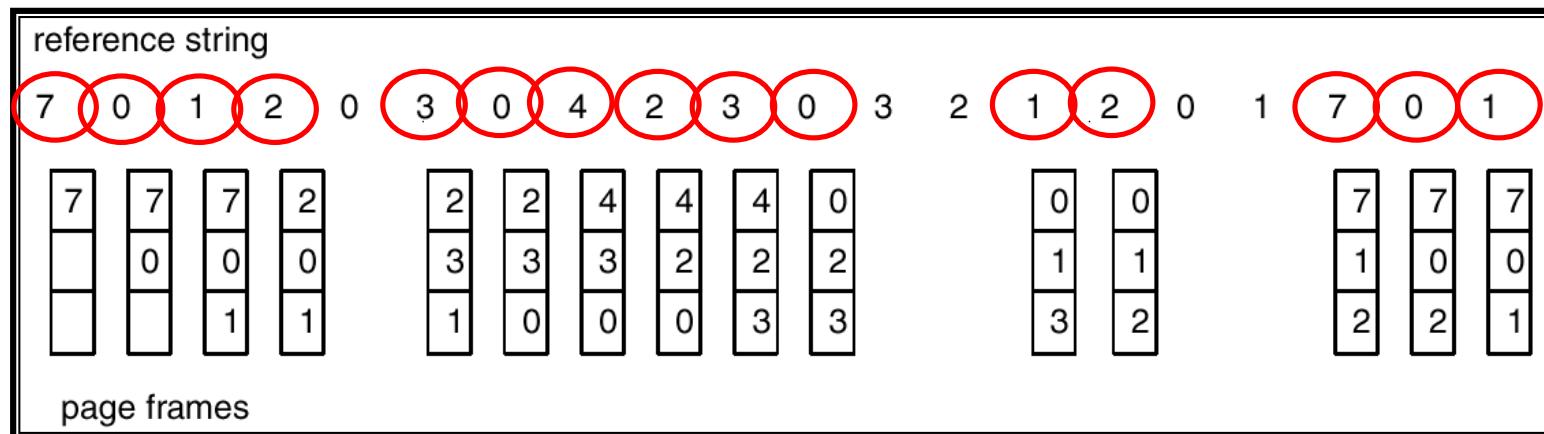
Tende ad aumentare il tasso di page fault

Soffre dell'anomalia di Belady

- A volte più frames \Rightarrow più page fault

Algoritmo FIFO – Esempio

- Consideriamo una memoria con 3 frame
- 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

- Con 4 frame 10 page fault

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

Algoritmo ideale

Garantisce minimo numero di page fault

Idea: rimpiazza le pagine che non saranno usate per il periodo di tempo più lungo

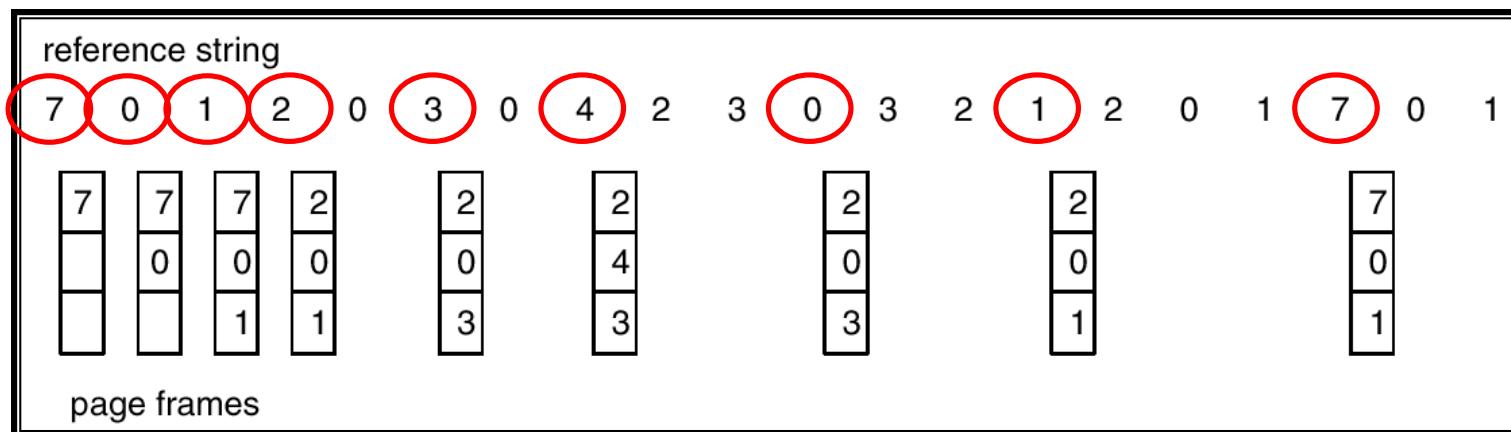
Problema: come ricavare questa informazione?

- Richiede conoscenza anticipata della stringa dei riferimenti (simile a SJF)
- Implementazione difficile, richiede supporto HW

Utile come riferimento per altri algoritmi ma necessarie approssimazioni

Algoritmo ideale – Esempio

- 9 page fault



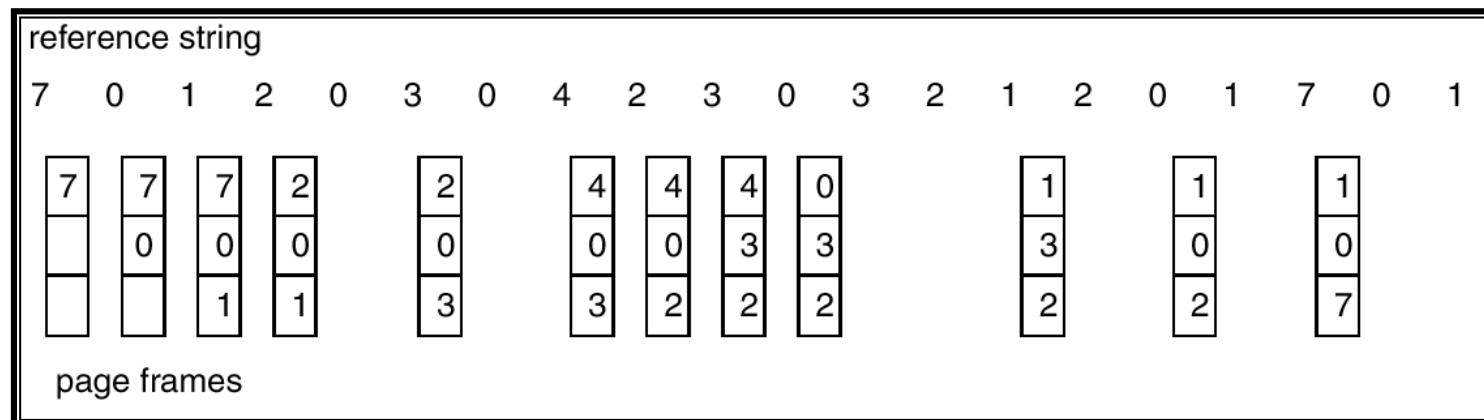
Algoritmo LRU (Least Recently Used)

- Approssimazione dell'algoritmo ottimo
 - Usare il passato recente come previsione del futuro
 - Si rimpiazza la pagina che non viene usata da più tempo (*)

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

- 12 page fault
 - Migliore del FIFO
 - Peggior dell'ideale





Algoritmo LRU – Implementazione?

- Non banale ricavare il tempo dell'ultimo utilizzo
- Può richiedere notevole HW addizionale

Algoritmo LRU - implementazioni

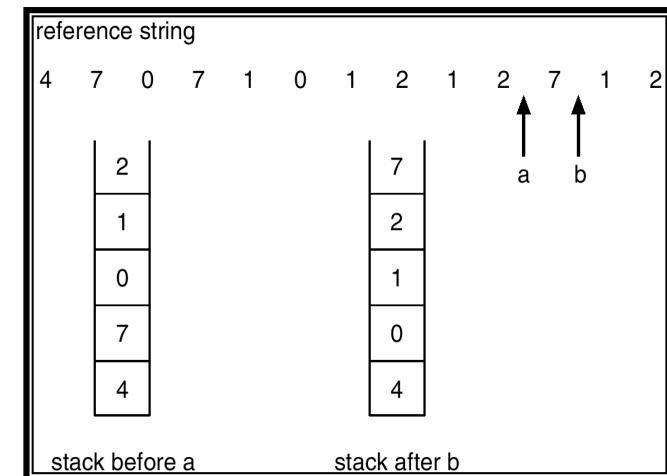
Tramite contatore

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

Algoritmo LRU - implementazioni

Tramite stack

- Mantenuto 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 - approssimazioni

Uso del bit di reference

- Associato a ogni pagina, inizialmente = 0
- Quando la pagina è referenziata, messo a 1 dall'HW
- Rimpiazzamento: sceglie una pagina che ha il bit a 0
- Non viene verificato l'ordine di riferimento delle pagine (chi è stato riferito prima?)

Alternativa

- Uso di 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

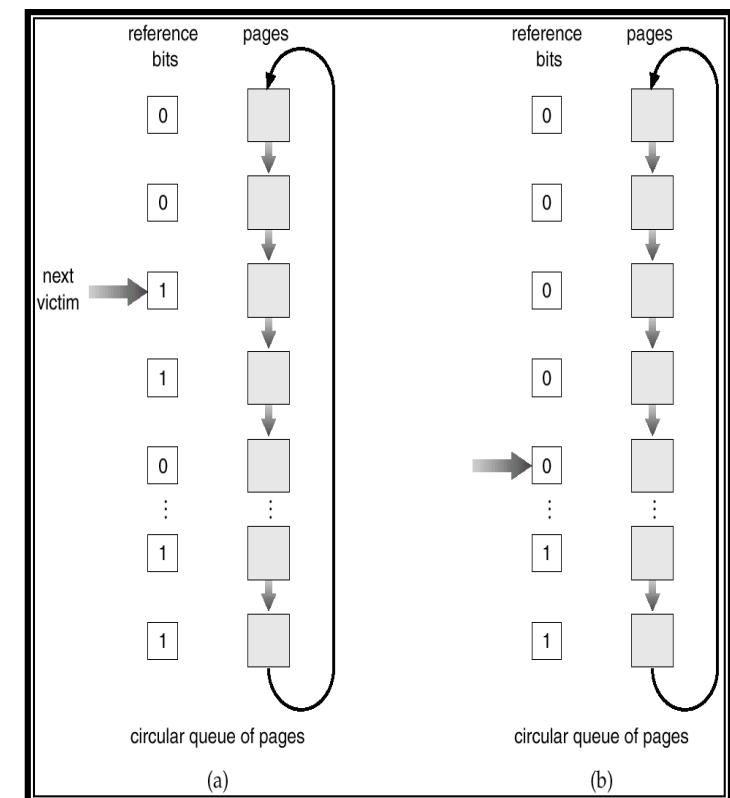
Algoritmo LRU - approssimazioni

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



Algoritmo LRU - approssimazioni

Tecniche basate su conteggio

Algoritmo LFU (Least Frequently Used)

- Conteggio # di riferimenti fatti a 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)

Allocazione dei frame



Allocazione dei frame

Data una memoria con N frame e M processi

- Quanti frame allocare a ogni processo?

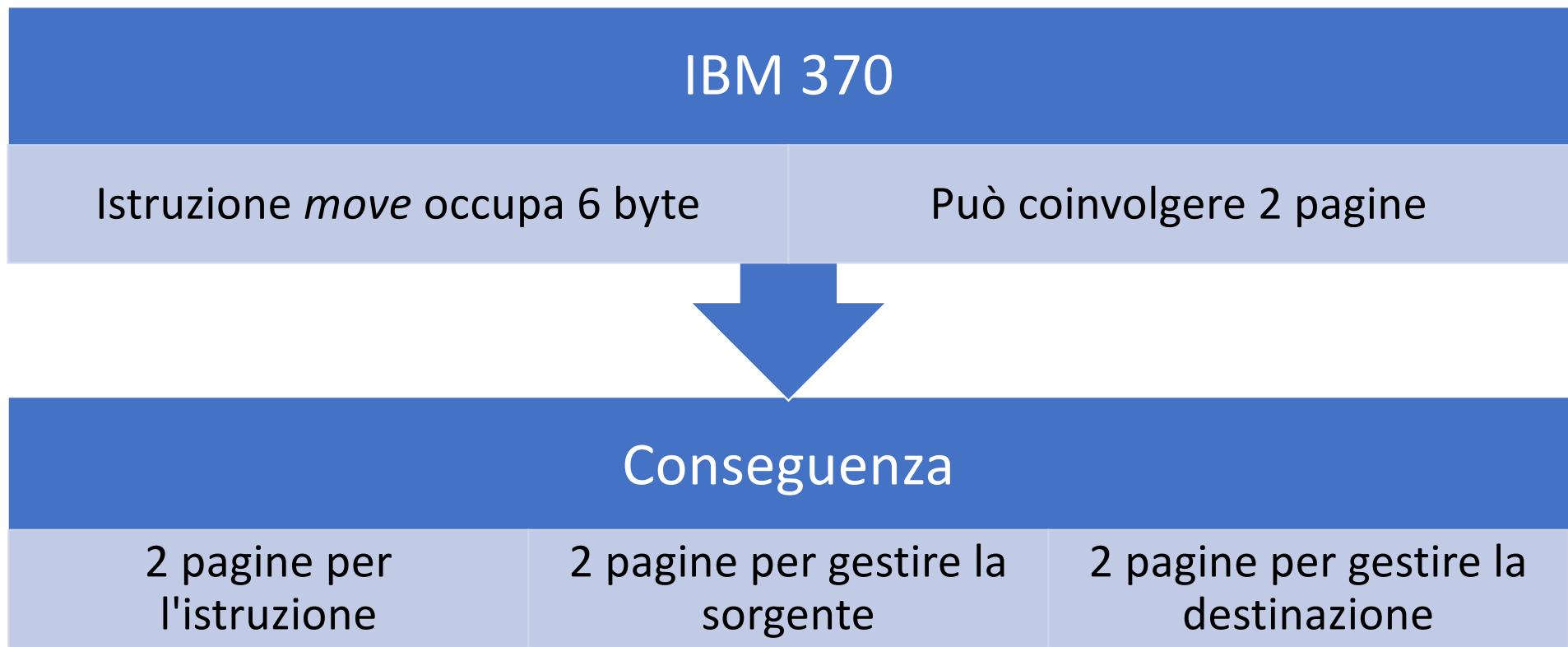
Vincoli

- Ogni processo necessita di un minimo numero di pagine per essere eseguito
 - Istruzione interrotta da un page fault deve essere fatta ripartire

Conseguenza

- # minimo di pagine = # massimo indirizzi specificabili in una istruzione
- Valori tipici: 2-4 frame

Allocazione dei frame – Esempio



Allocazione dei frame – Schemi di allocazione

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

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

Dove si scelgono le vittime?

Rimpiazzamento locale

- Ogni processo seleziona vittime solo tra i frame suoi

Rimpiazzamento globale

- Processo sceglie frame dall'insieme di tutti i frame
- Processo può prendere frame di un altro processo
- Migliora throughput
 - Preferibile al rimpiazzamento locale

Allocazione fissa

Allocazione in parti uguali

- Dati m frame e n processi, alloca a ogni processo m/n frame

Allocazione proporzionale

- Allocata 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 = allocation for p_i = $\frac{s_i}{S} \times m$

$$m = 64$$

$$s_1 = 10$$

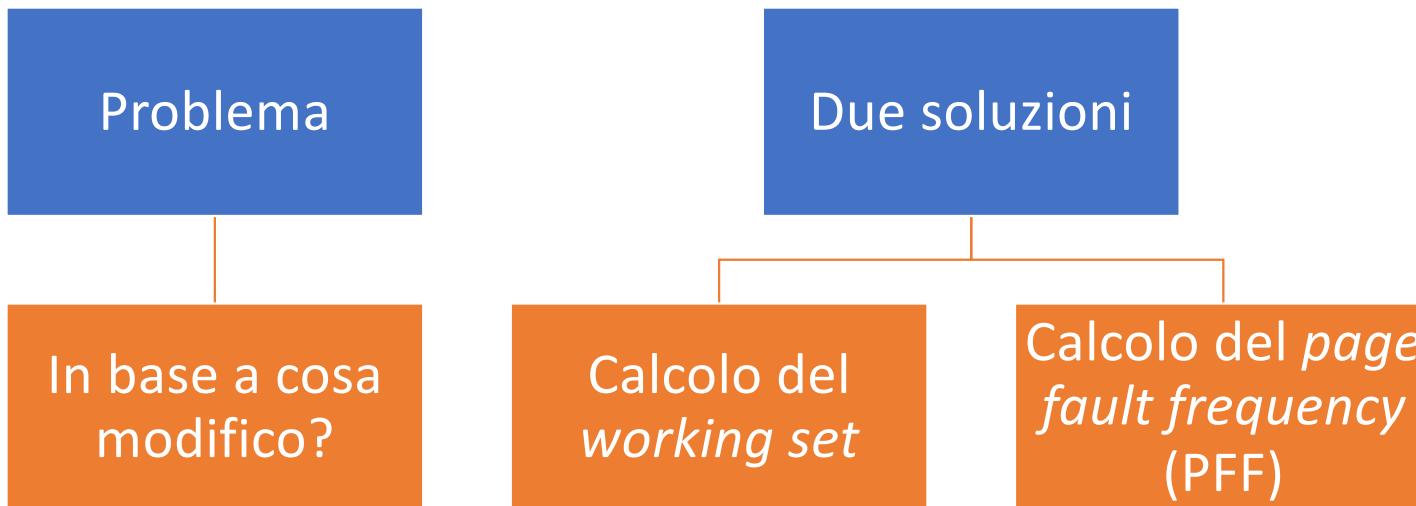
$$s_2 = 127$$

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

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

Allocazione variabile

Permette di modificare dinamicamente le allocazioni ai vari processi



Working set

Criterio per rimodulare l'allocazione dei frame in base alle richieste effettive di ogni processo

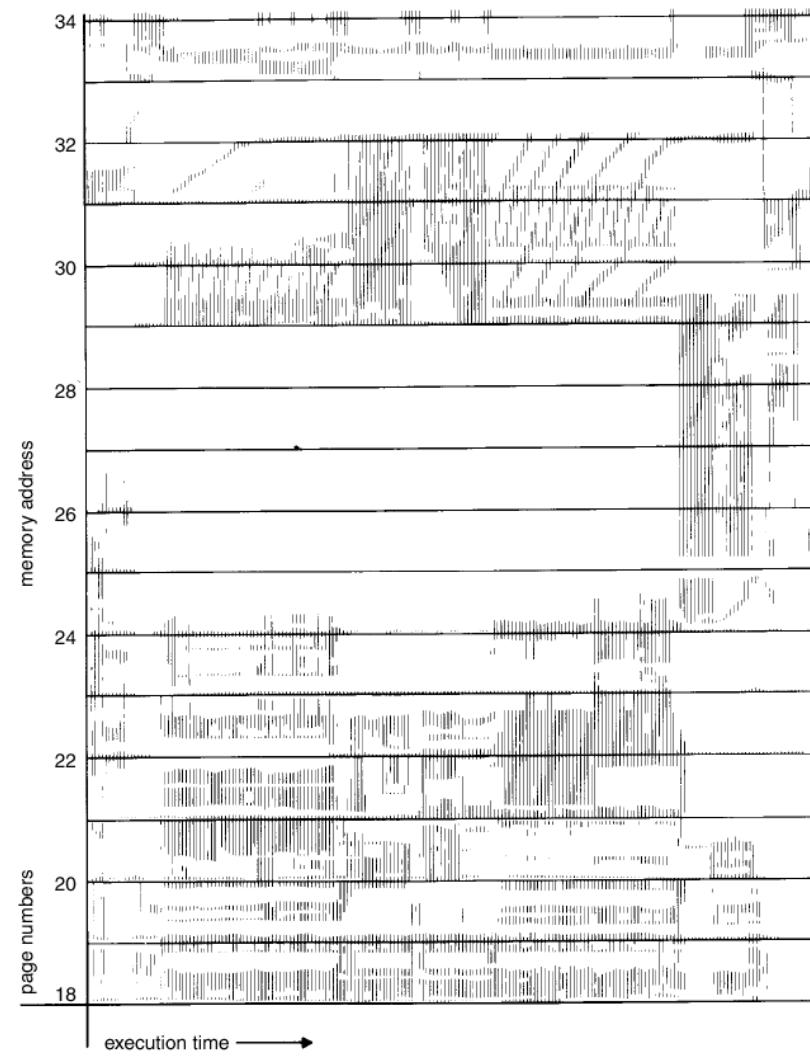
Sfrutta modello della località

Un processo passa da una località (di indirizzi) all'altra durante la sua esecuzione

Un processo necessita di un numero di frame pari alla sua località

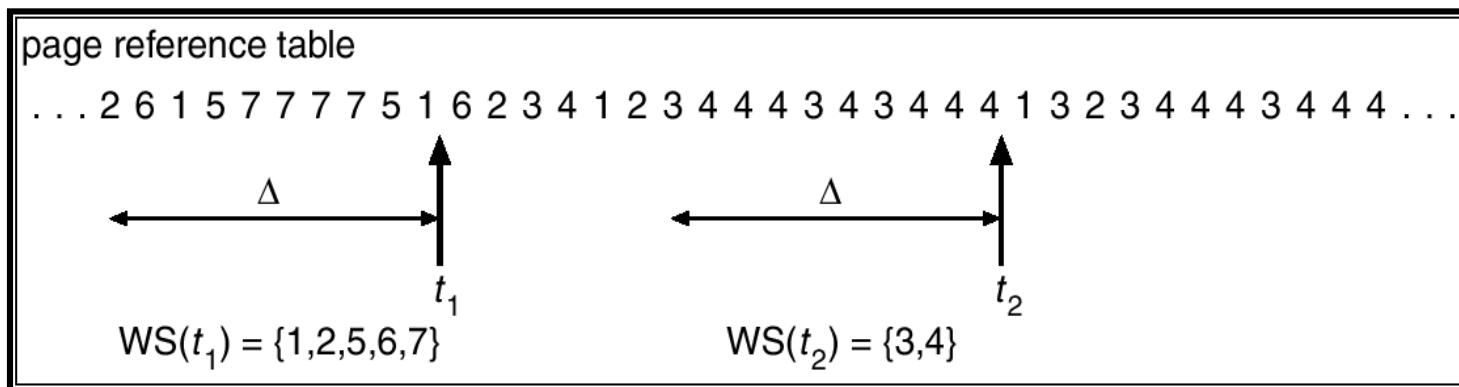
Come la misuro?

Località di esecuzione - Esempio



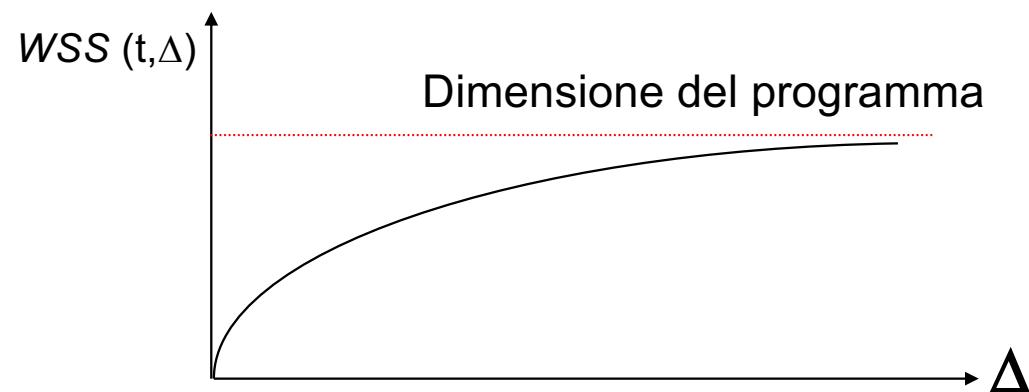
Calcolo del working set

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



Dimensione working set vs. Tempo

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



Cosa succede se i frame sono pochi?

$$D = \sum_i WSS_i = \text{richiesta totale di frame}$$

Se $D >$ numero totale frame

Si verifica **TRASHING**



Un processo spende tempo di CPU continuando a “swappare” pagine da e verso la memoria

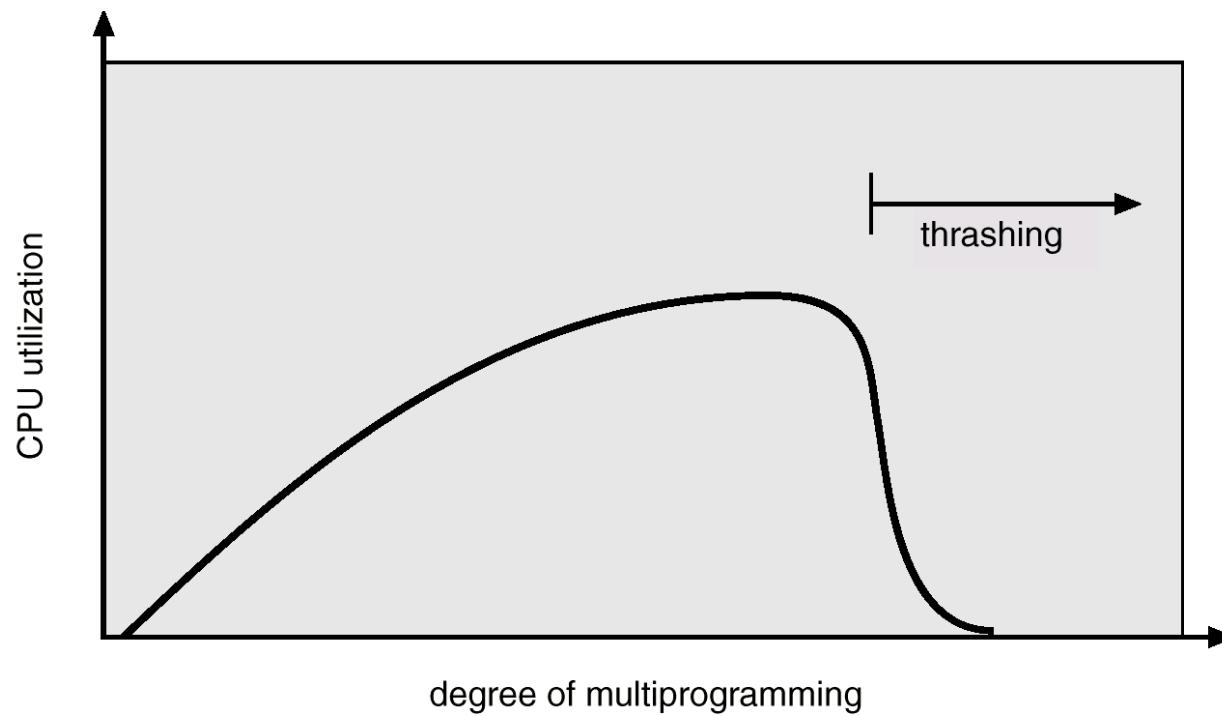
Conseguenza di un basso numero di frame

Risultato di un “circolo vizioso”

Trashing

- Numero di frame allocati a processo < soglia minima, tasso di page-fault tende a crescere
- Questo porta a:
 - Abbassamento utilizzo CPU causato da processi in attesa di gestire il page fault
 - 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!
- Necessario stimare con esattezza # frame necessari a un processo per non entrare in trashing

Diagramma del thrashing



Allora come misuro il working set?

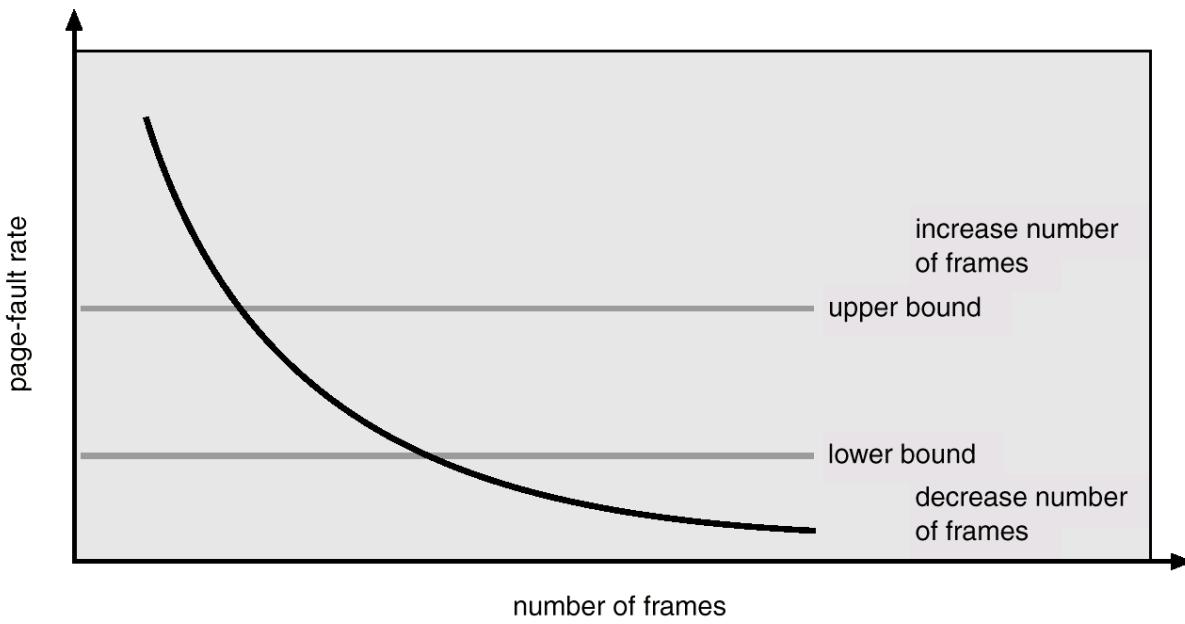
Approssimazione tramite timer e bit di reference

- Timer interrompe periodicamente la CPU
- All'inizio di ogni periodo, bit di reference posti a 0
- A 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 a

- # bit
- frequenza delle interruzioni

Oppure uso la frequenza dei page fault



- Soluzione 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 – Dimensione pagina

Frammentazione

- pagine grandi = frammentazione interna significativa

Località

- pagine grandi = granularità grande, devo trasferire anche ciò che non è necessario

Dimensione page table

- pagine piccole = molte entry

I/O overhead

- pagina piccola = costo di lettura/scrittura non ammortizzato

Pagine
piccole

Pagine
grandi

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
- for j := 1 to 1024 do
 for i := 1 to 1024 do
 A[i,j] := 0;
1024 x 1024 page fault
- for i := 1 to 1024 do
 for j := 1 to 1024 do
 A[i,j] := 0;
1024 page fault

Altre considerazioni – Frame locking

In alcuni casi particolari,
esistono frame che non devono
essere (mai) rimpiazzati

- Frame per pagine del kernel