



UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Sistemi Operativi

Memoria Virtuale

LEZIONE 23

prof. Antonino Staiano

Corso di Laurea in Informatica – Università di Napoli Parthenope

antonino.staiano@uniparthenope.it

Politiche di sostituzione delle pagine

- Obiettivo: sostituire una pagina che probabilmente non sarà referenziata nell'immediato futuro
- Esempi:
 - Strategia di **sostituzione pagina ottimale**
 - Minimizza il numero totale di page fault
 - Strategia di **sostituzione FIFO**
 - Strategia di **sostituzione LRU** (Least Recently Used)
 - Basi: località dei riferimenti
- **Stringhe di riferimento pagine**
 - Traccia delle pagine accedute da un processo durante le sue operazioni
 - Si associa ad ogni stringa di riferimento pagine una *stringa dei riferimenti temporale* t_1, t_2, t_3, \dots

Esempio: stringa di riferimento pagina

- Un computer supporta istruzioni di 4 byte di lunghezza
 - Usa una dimensione di pagina di 1KB
 - I simboli A e B del programma in esecuzione sono nelle pagine 2 e 5, rispettivamente

```
START 2040
READ B
LOOP MOVER AREG, A
      SUB   AREG, B
      BC    LT, LOOP
      ...
STOP
A     DS    2500
B     DS    1
END
```

Stringa riferimento pagina	1, 5, 1, 2, 2, 5, 2, 1
Stringa riferimento temporale	$t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, ..$

Sostituzione ottimale

- Significa prendere decisioni di sostituzione in modo che il numero di page fault sia il più piccolo possibile
 - Nessun'altra sequenza di decisioni di sostituzione pagina porta ad un numero inferiore di page fault
- Per realizzarlo, ad ogni page fault, la strategia di sostituzione dovrebbe considerare tutte le possibili decisioni alternative, analizzare le implicazioni per i page fault futuri e selezionare la miglior alternativa
 - Impossibile: il gestore non ha conoscenza del comportamento futuro del processo
 - Utile come tool analitico
 - Equivalente alla regola: sostituire la pagina il cui **riferimento successivo** è il più lontano nella stringa dei riferimenti di pagina (Belady, 1966)

Sostituzione FIFO

- Ad ogni page fault, la strategia sostituisce la pagina caricata in memoria prima di ogni altra pagina del processo
- Per semplificare il lavoro, il gestore memorizza nel campo *ref info* l'istante di caricamento di una pagina

Sostituzione LRU

- Usa la legge di località dei riferimenti
- Ad ogni page fault, è sostituita la pagina usata meno recentemente
- L'entrata della tabella delle pagine registra il tempo dell'ultimo riferimento alla pagina
 - Inizializzato quando la pagina è caricata
 - Aggiornata ad ogni riferimento

Esempio: sostituzione ottimale, FIFO, LRU

- Consideriamo le seguenti stringhe di riferimento pagina e stringhe di riferimento temporale per un processo P

Stringa riferimento pagina	0, 1, 0, 2, 0, 1, 2, ...
Stringa riferimento temporale	$t_1, t_2, t_3, t_4, t_5, t_6, t_7, \dots$

- Vediamo il funzionamento delle diverse strategie di sostituzione pagine con ***alloc* = 2**
 - alloc* rappresenta il numero di frame di pagina allocati al processo P

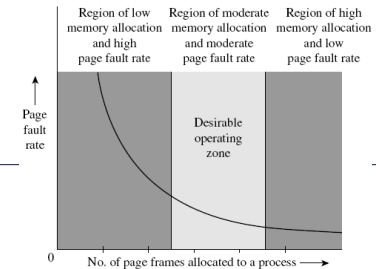
Esempio: sostituzione ottimale, FIFO, LRU

0, 1, 0, 2, 0, 1, 2, ...
 $t_1, t_2, t_3, t_4, t_5, t_6, t_7, \dots$

Time instant	Page ref	Optimal			FIFO			LRU		
		Valid bit	Ref info	Replacement	Valid bit	Ref info	Replacement	Valid bit	Ref info	Replacement
t_1	0	0	1	–	0	1 t_1	–	0	1 t_1	–
		1	0		1	0		1	0	
		2	0		2	0		2	0	
t_2	1	0	1	–	0	1 t_1	–	0	1 t_1	–
		1	1		1	1 t_2		1	1 t_2	
		2	0		2	0		2	0	
t_3	0	0	1	–	0	1 t_1	–	0	1 t_1 t_3	–
		1	1		1	1 t_2		1	1 t_2	
		2	0		2	0		2	0	
t_4	2	0	1	Replace 1 by 2	0	0	Replace 0 by 2	0	1 t_3	Replace 1 by 2
		1	0		1	1 t_2		1	0	
		2	1		2	1 t_4		2	1 t_4	
t_5	0	0	1	–	0	1 t_5	Replace 1 by 0	0	1 t_5	–
		1	0		1	0		1	0	
		2	1		2	1 t_4		2	1 t_4	
t_6	1	0	0	Replace 0 by 1	0	1 t_5	Replace 2 by 1	0	1 t_5	Replace 2 by 1
		1	1		1	1 t_6		1	1 t_6	
		2	1		2	0		2	0	
t_7	2	0	0	–	0	0	Replace 0 by 2	0	0	Replace 0 by 2
		1	1		1	1 t_6		1	1 t_6	
		2	1		2	1 t_7		2	1 t_7	

Page Fault
 Ottimale: 4
 FIFO: 6
 LRU: 5

Politiche di sostituzione pagina (cont.)



- Per ottenere caratteristiche desiderabili di page fault, i page fault non dovrebbero aumentare quando si incrementa l'allocazione della memoria
 - La politica deve avere la proprietà dello stack (o inclusione)

Una politica di sostituzione di pagina possiede la **proprietà dello stack** se $\{p_i\}_n^k \subseteq \{p_i\}_m^k$ per tutti gli n, m tali che $n < m$

dove $\{p_i\}_n^k$ indica l'insieme delle pagine in memoria al tempo t_k^+ se $\text{alloc}_i = n$ durante l'intera attività del processo P_i (t_k^+ implica l'istante dopo t_k ma prima di t_{k+1})

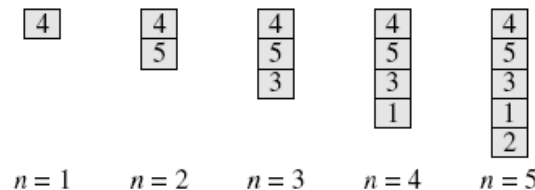


Figure 12.16 $\{p_i\}_n^k$ for different n for a page replacement policy processing the stack property.

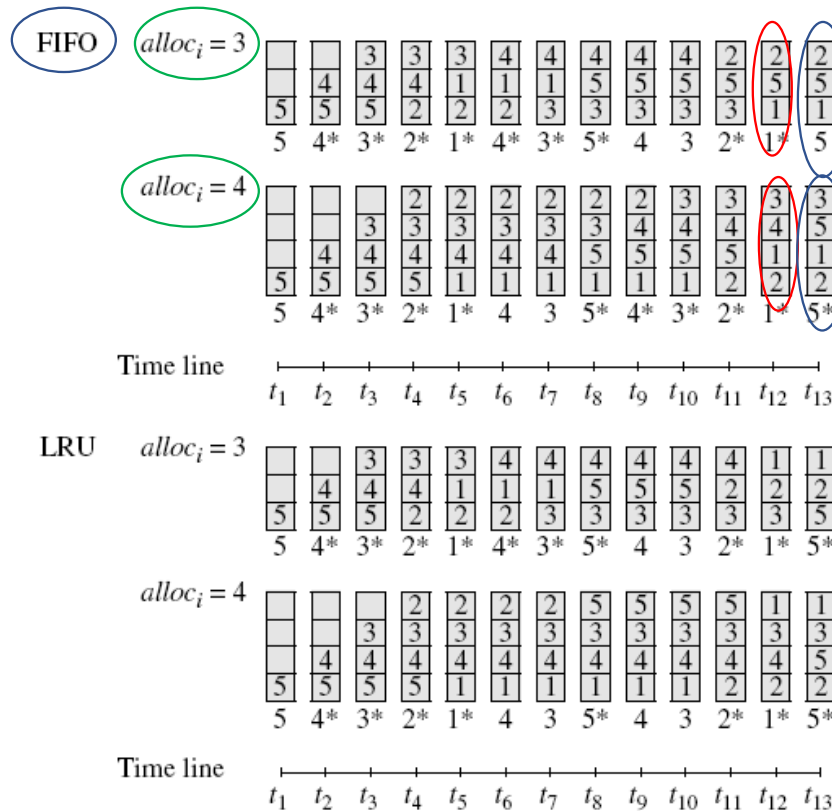
Proprietà dello stack

- Consideriamo due esecuzioni del processo P_i , con $alloc_i = n$ e $alloc_i = m$, rispettivamente, dove $n < m$
- Se una politica ha la proprietà dello stack, allora negli stessi istanti durante le operazioni di P_i nelle due esecuzioni, tutte le pagine che erano in memoria con $alloc_i = n$ sarebbero in memoria anche quando $alloc_i = m$
- Inoltre, la memoria contiene anche $m-n$ pagine aggiuntive del processo
 - Se una di tali pagine sarà riferita in pochi riferimenti successivi di P_i , il page fault si verifica se $alloc_i = n$, ma non se $alloc_i = m$
 - Quindi il page fault è più elevato se $alloc_i = n$ rispetto ad $alloc_i = m$

Problemi con politica FIFO

Page reference string 5, 4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5, ...

Reference time string $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}$



* Riferimenti a pagine che causano page fault e sostituzione di pagina

La strategia di sostituzione FIFO non possiede la proprietà stack

Politiche di sostituzione pagina (cont.)

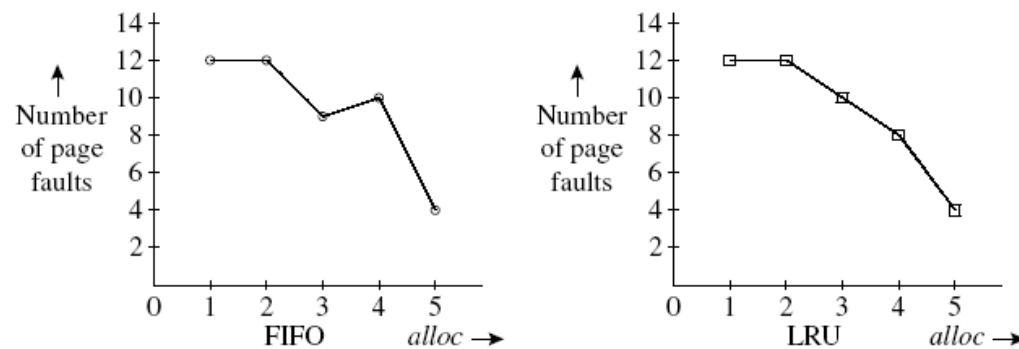


Figure 12.18 (a) Belady's anomaly in FIFO page replacement; (b) page fault characteristic for LRU page replacement.

- Il gestore della memoria virtuale non può usare una politica FIFO
 - Aumentare l'allocazione ad un processo può incrementare la frequenza di page fault del processo
 - Renderebbe impossibile controllare il thrashing

Politiche di sostituzione pagina in pratica

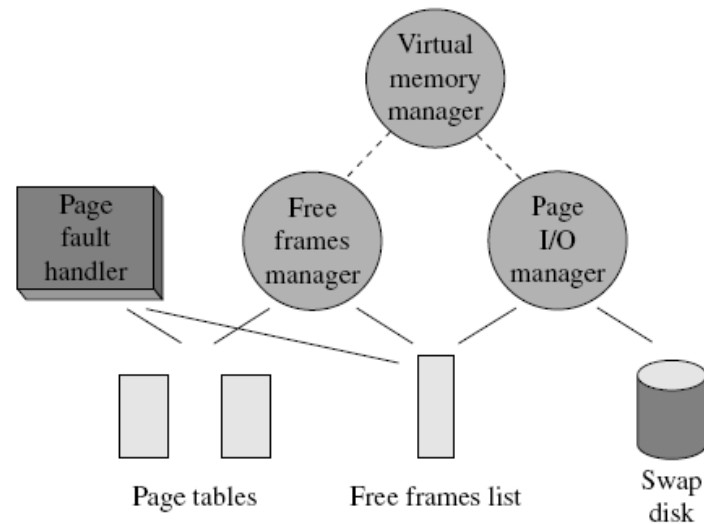


Figure 12.19 Page replacement in practice.

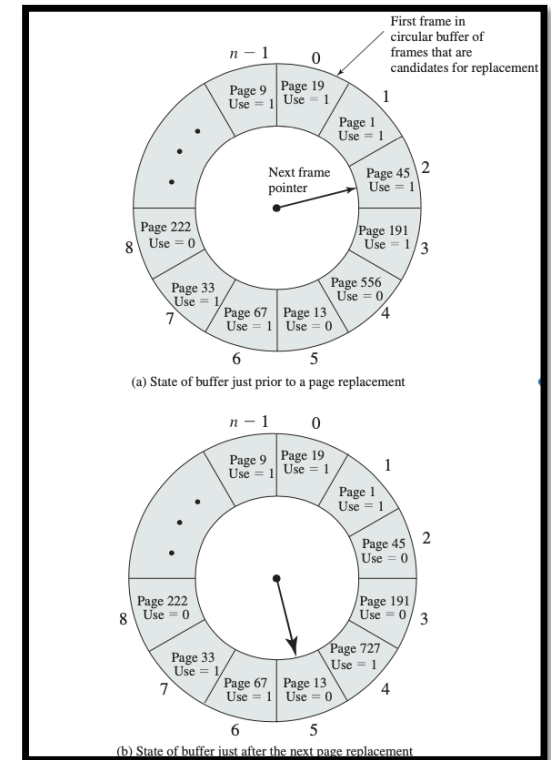
- Il gestore della memoria virtuale ha due thread demoni
 - Il gestore dei frame liberi implementa la politica di sostituzione pagina
 - Il gestore dell'I/O di pagina esegue le operazioni page-in/out

Politiche di sostituzione pagina in pratica (cont.)

- La sostituzione LRU non è fattibile
 - I computer non forniscono sufficienti bit nel campo *ref info* per memorizzare l'istante dell'ultimo riferimento
- La maggior parte dei computer forniscono un singolo bit di riferimento
 - Le politiche *Not Recently Used* (NRU) usano questo bit
 - La strategia NRU più semplice: **sostituisce una pagina non referenziata** e **resetta tutti i bit di riferimento** se **tutte le pagine sono state referenziate**
 - Gli algoritmi clock forniscono migliore discriminazione di pagina resettando i bit di riferimento periodicamente
 - Algoritmo di clock one-handed
 - Algoritmo di clock two-handed
 - Puntatore di reset (RP) e puntatore di controllo (EP)
 - Le pagine di tutti i processi in memoria sono immesse in una lista circolare e sono usati dei puntatori che si spostano sulle pagine ripetutamente
 - E' esaminata la pagina puntata da un puntatore
 - È intrapresa un'azione su di essa
 - Il puntatore è aggiornato per puntare alla prossima pagina

Clock ad una Lancetta (seconda chance)

- Il puntatore punta a qualche specifica pagina
- Quando è necessaria una sostituzione, il SO controlla se la pagina puntata ha il **ref bit** a 1
 - Se sì -> usata di recente
 - Non è una buona candidata per la sostituzione, il suo ref bit è resettato ed il puntatore passa alla pagina successiva
 - Se no -> non è stata usata di recente
 - La pagina viene sostituita

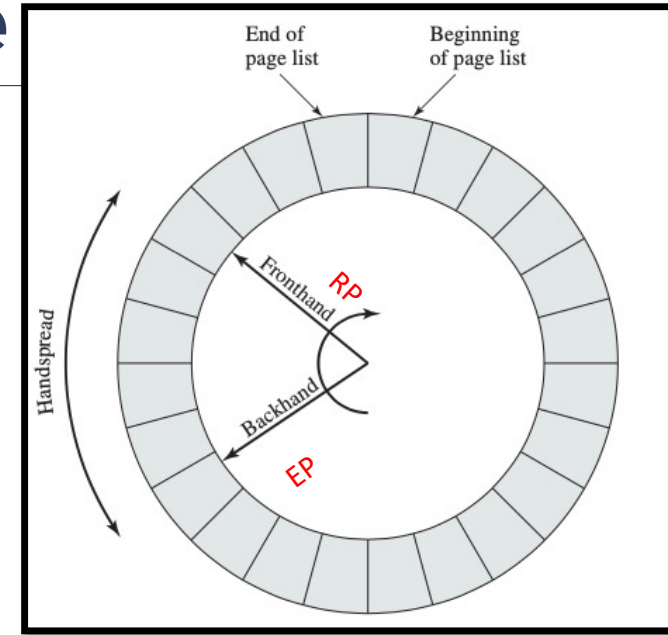


Problemi del clock ad una lancetta

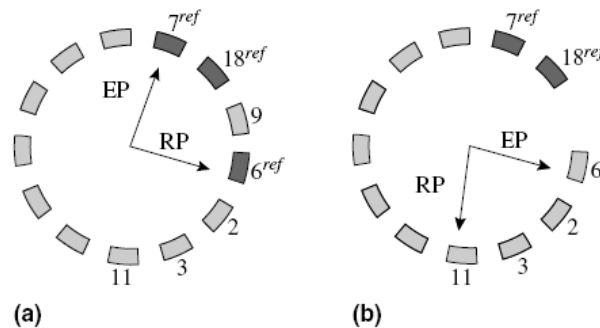
- Resetta il **ref bit** quando una pagina viene considerata per la rimozione, per cui a seconda della posizione e della dimensione della memoria può succedere che
 - Tutti i ref bit sono impostati a 1 (per cui ogni accesso causa un loop su un elevato numero di candidati)
 - Quasi ogni bit è 0 (si degenera in una politica FIFO)
- La soluzione è disaccoppiare la parte di sostituzione dal reset
 - Usando due lancette che esaminano i frame

Algoritmi di clock a due Lancette

- Nel clock a due lancette
 - Sono gestiti due puntatori
 - Puntatore di reset (RP)
 - Usato per resettare i bit di riferimento
 - Puntatore di controllo (EP)
 - Usato per controllare i bit di riferimento
 - RP ed EP sono incrementati simultaneamente
 - Il frame di pagina a cui punta EP è aggiunto alla lista dei frame liberi se il suo **ref bit** è 0
 - Il **ref bit** del frame puntato da RP è impostato a 0
 - La distanza tra EP e RP definisce quanto **recente** è una pagina
 - Il ref bit di una pagina sarà impostato ad 1 se è stato acceduto più recentemente rispetto a quando è passato il puntatore di reset



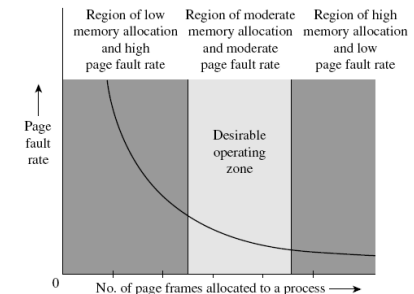
Esempio: algoritmo di clock a due lancette



- Entrambi i puntatori sono avanzati simultaneamente
- Le proprietà dell'algoritmo sono definite dalla distanza dei puntatori:
 - Se la distanza è troppo lunga, l'algoritmo approssima il clock ad una lancetta
 - Potremmo dover esaminare molti frame prima di individuarne uno non usato
 - Se la distanza è troppo breve le pagine raramente hanno la possibilità di essere usate dopo che i bit sono stati resettati ma prima del passaggio del puntatore di verifica
 - L'algoritmo diventa un algoritmo FIFO
- La distanza corretta dipende dal carico di lavoro ed è una misura di quanta località ha il processo
 - Una maggiore distanza significa più storia la cui utilità dipende da quanta località ha il programma

Controllare l'allocazione di memoria ad un processo

- Al processo P_i sono allocate $alloc_i$ frame di pagina
- *Allocazione di memoria fissa*
 - Fissa alloc in modo statico; usa sostituzioni di pagina locali
- *Allocazione di memoria variabile*
 - Usa sostituzioni di pagina globali e/o locali
 - Se è usata una sostituzione locale, il gestore periodicamente determina il valore corretto di alloc per un processo
 - Può usare il modello working set



Working set: *l'insieme di pagine di un processo che sono state referenziate nelle precedenti Δ istruzioni del processo, dove Δ è un parametro di sistema*

- Imposta alloc alla dimensione del working set

Working set

- Le precedenti Δ istruzioni costituiscono la **finestra del working set**
- $WS_i(t, \Delta)$ working set P_i al tempo t per la finestra Δ
- $WSS_i(t, \Delta)$ dimensione del working set $WS_i(t, \Delta)$
 - Numero di pagine in $WS_i(t, \Delta)$
- $WSS_i(t, \Delta) \leq \Delta$
 - Una pagina può essere referenziata più di una volta in una finestra di WS
- Un allocatore di memoria basato su working set o mantiene l'intero working set in memoria oppure sospende il processo
 - Al tempo t , per il processo P_i
 - $alloc_i = WSS_i$ oppure $alloc_i = 0$
 - La strategia assicura buoni hit ratio in memoria (località dei riferimenti)
 - Previene il thrashing

Working set: gradi di multiprogrammazione

- L'allocatore working set varia il grado di multiprogrammazione sulla base delle dimensioni dei working set dei processi
 - Se $\{P_k\}$ è l'insieme dei processi in memoria,
 - si decide di abbassare il grado di multiprogrammazione se

$$\sum_k WSS_k > \#frame$$

- Si incrementa il grado di multiprogrammazione se $\sum_k WSS_k < \#frame$ ed esiste P_g tale che

$$WSS_g \leq \left(\#frame - \sum_k WSS_k \right)$$

- Il gestore della memoria virtuale mantiene $alloc_i$ e WSS_i per ogni P_i
 - Per **abbassare il grado di multiprogrammazione** il gestore sceglie un processo, P_i , da sospendere
 - Esegue un page-out per ogni pagina modificata di P_i e cambia lo stato dei frame a libero
 - $alloc_i$ è impostato a 0, mentre WSS_i rimane inalterato
 - Per **incrementare il grado di multiprogrammazione**, si ripristina P_i e si pone $alloc_i = WSS_i$
 - Carica la pagina di P_i che contiene la prossima istruzione da eseguire, le altre pagine sono caricate in corrispondenza dei page fault
 - Alternativamente, si caricano tutte le pagine di WS_i , ma ridondanza dei caricamenti possibile

Implementazione del working set

- Costoso determinare $WS_i(t, \Delta)$ e $alloc_i$ ad ogni istante t
 - Soluzione: determinare i working set periodicamente
 - Gli insiemi determinati alla fine di un intervallo sono usati per decidere i valori di alloc da usare per il prossimo intervallo

60 frame liberi da allocare a P_1, P_2, P_3 e P_4

Process	t_{100}		t_{200}		t_{300}		t_{400}	
	WSS	alloc	WSS	alloc	WSS	alloc	WSS	alloc
P_1	14	14	12	12	14	14	13	13
P_2	20	20	24	24	11	11	25	25
P_3	18	18	19	19	20	20	18	18
P_4	10	0	10	0	10	10	12	0

Figure 12.21 Operation of a working set memory allocator.

Copy-on-Write

- Caratteristica usata per conservare memoria quando i dati nelle pagine condivise possono essere modificate, ma i valori modificati sono riservati ad un processo
 - E' utilizzato un flag copy-on-write, **c**, nelle entrate della tabella delle pagine

E' fatta una copia privata della pagina k quando A la modifica

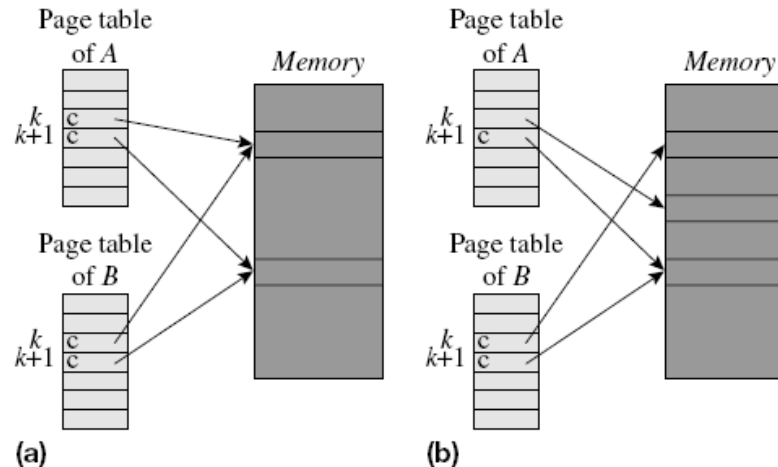


Figure 12.24 Implementing copy-on-write: (a) before and (b) after process A modifies page k .