Gestione della memoria

Algoritmi di rimpiazzamento
Segmentazione

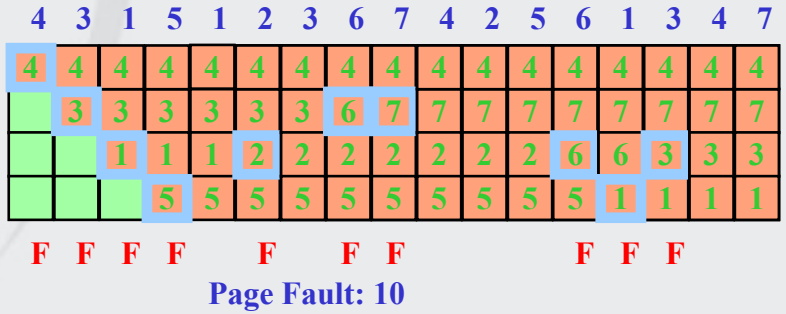
Algoritmi di rimpiazzamento

Per ogni page fault il SO deve:

- scegliere una pagina da rimuovere dalla memoria per creare spazio per la nuova pagina
  - se la pagina da rimuovere è stata modificata, la copia su disco deve essere aggiornata
  - se non c'è stata alcuna modifica, la nuova pagina sovrascrive quella da rimuovere
- Scegliendo la pagina da rimpiazzare con un algoritmo per sostituire pagine poco utilizzate (e non a caso) può portare a un miglioramento delle prestazioni

Algoritmo ottimo

- Ad ogni pagina viene assegnata un'etichetta corrispondente al numero di istruzioni che dovranno essere eseguite prima che la pagina venga referenziata
- L'algoritmo ottimo consiste nel rimuovere la pagina con l'etichetta maggiore



Algoritmo ottimo

- **Non è realizzabile** in quanto non è possibile conoscere a priori (quindi al momento del page fault) quali pagine verranno referenziate
  - Sarebbe necessario prevedere il futuro

Algoritmo ottimo

- Potrebbe essere utile eseguire il programma con un simulatore e tenere traccia dei riferimenti alle pagine
  - L'algoritmo ottimo può essere utilizzato per una seconda esecuzione utilizzando le informazioni sui riferimenti alle pagine della prima esecuzione
  - È utile come confronto per la valutazione degli algoritmi di rimpiazzamento pagine

Algoritmo FIFO

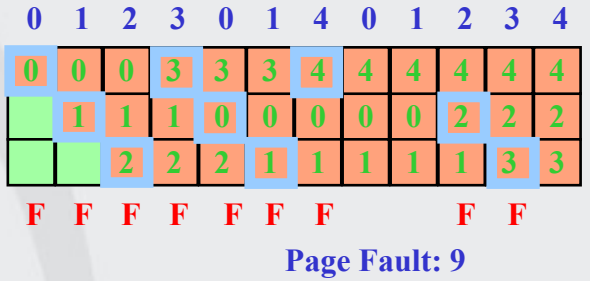
- Il SO mantiene una lista concatenata di tutte le pagine in memoria con:
  - la pagina più vecchia in testa
  - la pagina più recente in coda
- Quando avviene un page fault viene rimossa la pagina in testa alla lista e la nuova pagina viene aggiunta in coda alla lista
- L'algoritmo FIFO in questa forma viene utilizzato raramente infatti non è detto che la pagina più vecchia sia effettivamente la meno referenziata

Algoritmo FIFO

Page Fault: 12

Page Fault: 10

Anomalia di Belady



Page Fault: 9

Page Fault: 10

Algoritmo NRU

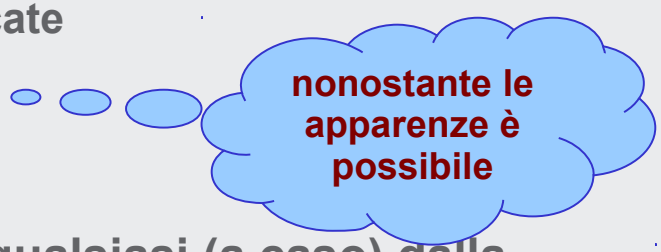
- La maggior parte dei computer con memoria virtuale prevede 2 bit per la raccolta di informazioni sull'utilizzo delle pagine:
  - il bit R: indica che la pagina è stata referenziata (letta o scritta)
  - il bit M: indica che la pagina è stata modificata (scritta)

Algoritmo NRU (Not Recently Used)

- Inizialmente i bit R e M vengono impostati a 0 dal SO
- Periodicamente (ad esempio ad ogni interrupt del clock), il bit R viene azzerato per distinguere le pagine non referenziate recentemente dalle altre

Algoritmo NRU

- Quando avviene un page fault, il SO controlla tutte le pagine e le divide in quattro classi in base al valore corrente dei bit R e M
  - Classe 0: non referenziate, non modificate
  - Classe 1: non referenziate, modificate
  - Classe 2: referenziate, non modificate
  - Classe 3: referenziate, modificate



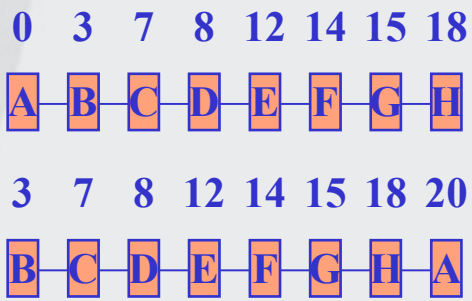
- L'algoritmo NRU rimuove una pagina qualsiasi (a caso) dalla classe di numero inferiore che sia non vuota
- Caratteristiche di NRU
  - è facile da implementare
  - ha prestazioni che, anche se non ottimali, sono spesso adeguate

Algoritmo Second Chance

- Con una semplice modifica dell'algoritmo FIFO è possibile evitare il problema della rimozione di pagine molto utilizzate
- Viene controllato il bit R della pagina più vecchia:
  - se vale 0, la pagina è sia vecchia che non utilizzata e viene rimpiazzata immediatamente
  - se vale 1, il bit viene azzerato e la pagina viene messa in coda alla lista come se fosse appena stata caricata in memoria

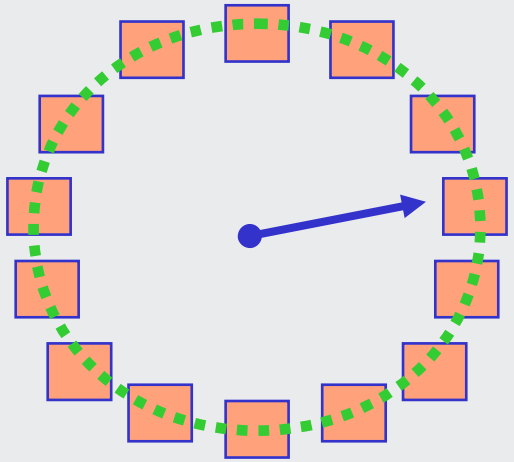
Algoritmo Second Chance

- L'algoritmo si basa sulla ricerca di una pagina che non sia stata referenziata nel precedente intervallo di clock
  Se tutte le pagine sono state referenziate degenera nell'algoritmo FIFO puro



Algoritmo clock

- L'algoritmo second chance è inefficiente perché sposta costantemente le pagine sulla lista
- Un approccio migliore consiste nel tenere tutte le pagine su una lista circolare nella forma di un orologio con una lancetta che punta alla pagina più vecchia



Algoritmo clock

- Quando avviene un page fault:
  - se il bit R della pagina vale 0, la pagina puntata dalla lancetta viene rimossa, la nuova pagina viene inserita nell'orologio al suo posto e la lancetta viene spostata in avanti di una posizione
  - se il bit R della pagina vale 1, viene azzerato e la lancetta viene spostata in avanti di una posizione
- Questo processo viene ripetuto finché non viene trovata una pagina con R = 0
- Differisce dall'algoritmo second chance solo per l'implementazione

Algoritmo LRU

- È una buona approssimazione all'algoritmo ottimo e si basa sulla seguente osservazione:
- le pagine molto utilizzate nelle ultime istruzioni saranno probabilmente molto utilizzate nelle successive istruzioni così come le pagine poco utilizzate recentemente continueranno a non essere utilizzate per molto tempo

Algoritmo LRU (Least Recently Used)

- Quando avviene un page fault viene rimpiazzata la pagina non utilizzata da più tempo

Algoritmo LRU

- È un algoritmo costoso poiché deve essere mantenuta una lista concatenata di tutte le pagine in memoria (ordinata in base al tempo trascorso dall'ultimo utilizzo), questa lista deve essere aggiornata ad ogni riferimento in memoria
- Eseguire, ad ogni istruzione, operazioni di ricerca e manipolazione in una lista concatenata è molto costoso in termini di tempo
- Ci sono soluzioni per l'implementazione di LRU con hardware speciale

Algoritmo LRU

LRU: implementazione H/W

- Sequenza di pagine: 0 1 2 3 2 1 0 3 2 3

```
0 0111 0011 0001 0000 0000
1 0000 1011 1001 1000 1000
2 0000 0000 1101 1100 1101
3 0000 0000 0000 1110 1100

0 0000 0111 0110 0100 0100
1 1011 0011 0010 0000 0000
2 1001 0001 0000 1101 1100
3 1000 0000 1110 1100 1110
```

  - Ad ogni accesso viene messa ad **uno la riga** corrispondente alla pagina
  - poi viene messa a **zero la colonna** corrispondente

Algoritmo NFU

- Una approssimazione dell'algoritmo LRU è

  Algoritmo NFU (Not Frequently Used)

- Ad ogni pagina viene associato un contatore inizialmente posto a 0
- Ad ogni clock viene sommato al contatore il bit R
- Al momento di un page fault viene rimpiazzata la pagina con contatore minimo

Algoritmo NFU

- Il problema è che NFU non ***dimentica*** nulla:
  - se una pagina è stata molto utilizzata non verrà più rimossa anche se non più utile
- È stata proposta una versione modificata con **invecchiamento (aging)**:
  - ad ogni clock il contatore scorre a destra introducendo R come bit più significativo

Bit R	101011	110010	110101	100010	011000
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00010000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000
	t=1	t=2	t=3	t=4	t=5

Modello a Working Set

- Nel modello più semplice di paginazione le pagine vengono caricate in memoria solo al momento del page fault, si parla di paginazione su richiesta (on demand)
- La maggior parte dei processi esibiscono la località di riferimenti cioè durante qualsiasi fase dell'esecuzione il processo fa riferimenti ad un piccolo insieme di pagine
- L'insieme di pagine che un processo sta correntemente utilizzando viene detto working set
- Più è alto il numero pagine in memoria più è probabile che il working set sia completamente in memoria (e quindi non avverranno page fault)
- Un programma che causa page fault per ogni piccolo insieme di istruzioni viene detto in thrashing

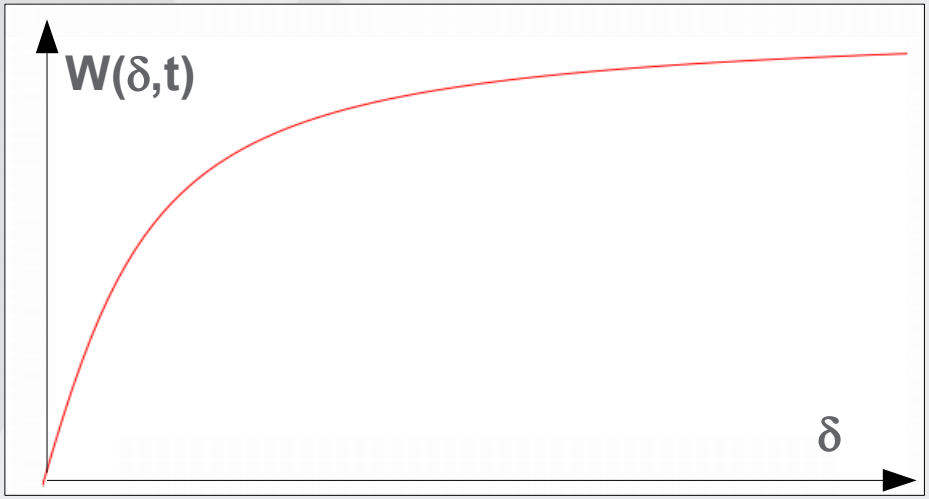
Working Set

- Il WS con parametro δ al tempo t, $W(\delta,t)$ è l'insieme delle pagine a cui ci si è riferiti nelle ultime δ unità di tempo
- Proprietà:

$$W(\delta,t) \subseteq W(\delta+1,t)$$

$$1 \leq |W(\delta,t)| \leq \min(\delta,N)$$

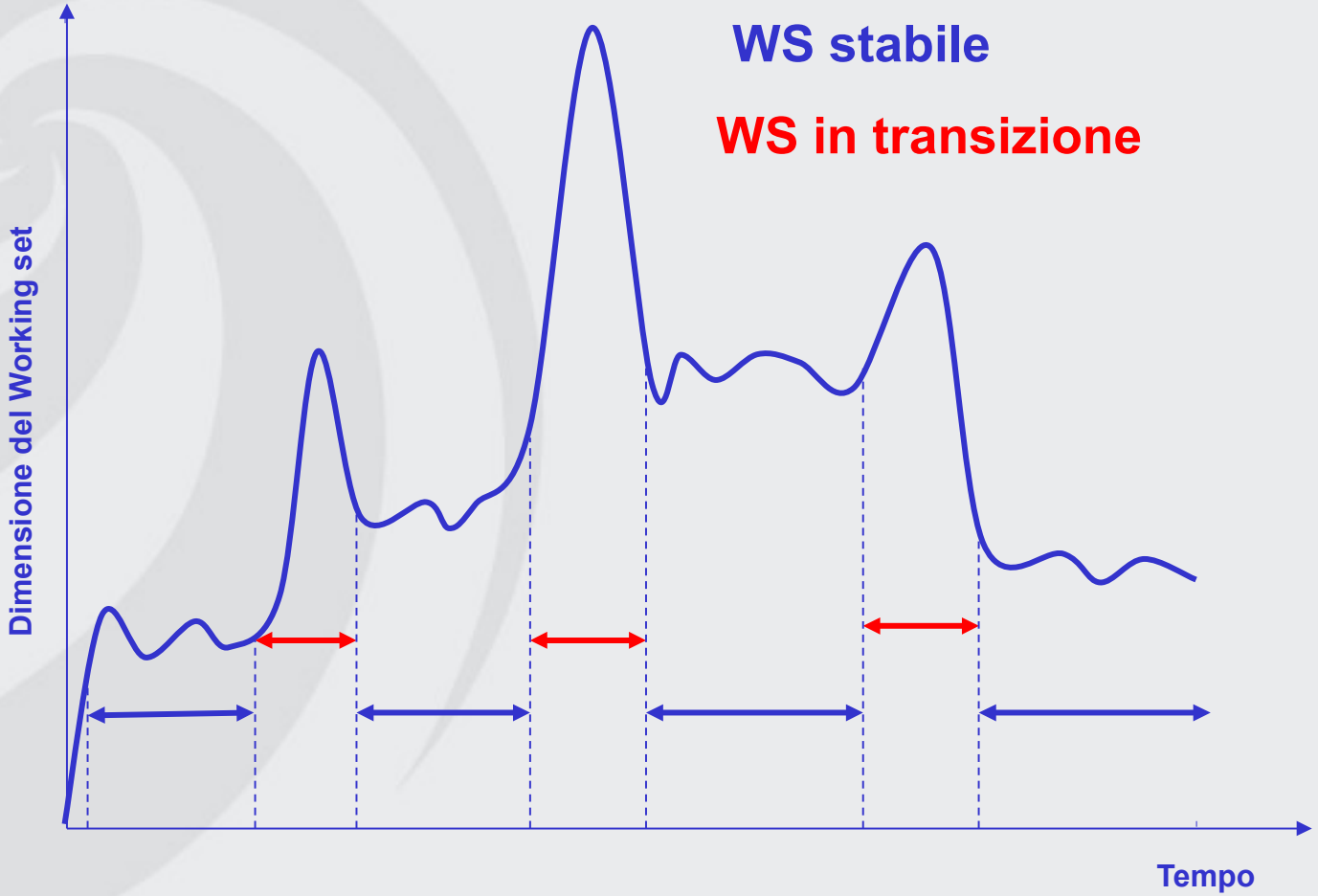
N numero di pagine necessarie al processo



Working set

Sequenza di riferimenti		Dimensione della finestra (2÷5)			
24		24	24	24	24
15		24 15	24 15	24 15	24 15
18		15 18	24 15 18	24 15 18	24 15 18
23		18 23	15 18 23	24 15 18 23	24 15 18 23
24		23 24	18 23 24	-	-
17		24 17	23 24 17	18 23 24 17	15 18 23 24 17
18		17 18	24 17 18	-	18 23 24 17
24		18 24	-	24 17 18	-
18		-	18 24	-	24 17 18
17		18 17	24 18 17	-	-
17		17	18 17	-	-
15		17 15	17 15	18 17 15	24 18 17 15
24		15 24	17 15 24	17 15 24	-
17		24 17	-	-	17 15 24
24		-	24 17	-	-
18		24 18	17 24 18	24 17 18	15 24 17 18

Working set


WS stabile
WS in transizione
Dimensione del Working set
Tempo

Prestazioni e page fault

Qual è la perdita di prestazioni per un tasso di page fault di 1 su 100000?

- access time senza page fault: 100 ns
- tempo di accesso a memoria
  0.5 * 20 ms + 0.5 * 40 ms = 30 ms
  con un rimpiazzamento di pagine del 50%

Effective Access Time (p = probabilità di page fault)

EAT =

p * accesso con page fault +
(1-p) accesso senza page fault

EAT =

0.00001 * 30 ms + (1-0.00001) * 100 ns =
300 ns + 99.999 ns = 400 ns

rappresenta una perdita di prestazioni del 300%

Prestazioni e page fault

Quale è la percentuale di fault per un 10% di perdita di prestazioni (EAT=110 ns)?

110 ns

> p * 30 ms + (1-p) * 100 ns
> p * 30,000,000 ns + 100ns

10 ns > p * 30,000,000 ns

p < 10/30,000,000 (1 su 3 milioni)

Località

- Per località si indica che il prossimo accesso di memoria sarà nelle vicinanze di quello corrente
  - località spaziale: il prossimo indirizzo differirà di poco
  - località temporale: la stessa cella (o pagina) sarà molto probabilmente riutilizzata nel futuro prossimo
- La località è la ragione che giustifica un basso numero di page fault (altrimenti non spiegabile)

Località

```
int mat[1024][1024];

max=mat[0][0];
for(i=0; i<1024; i++)
  for(j=0; j<1024; j++)
    if(max<mat[i][j])
      max = mat[i][j];
```

Località temporale
continuo ad usare
le variabili i, j

Località spaziale
dopo mat[10][101]
uso mat[10][102]

Località

```
int mat[1024][1024];

max=mat[0][0];
for(i=0; i<1024; i++)
  for(j=0; j<1024; j++)
    if(max<mat[j][i])
      max = mat[j][i];
```

- Questa seconda soluzione potrebbe causare un numero notevole di page fault (**perché?**)

Strategia di fetch

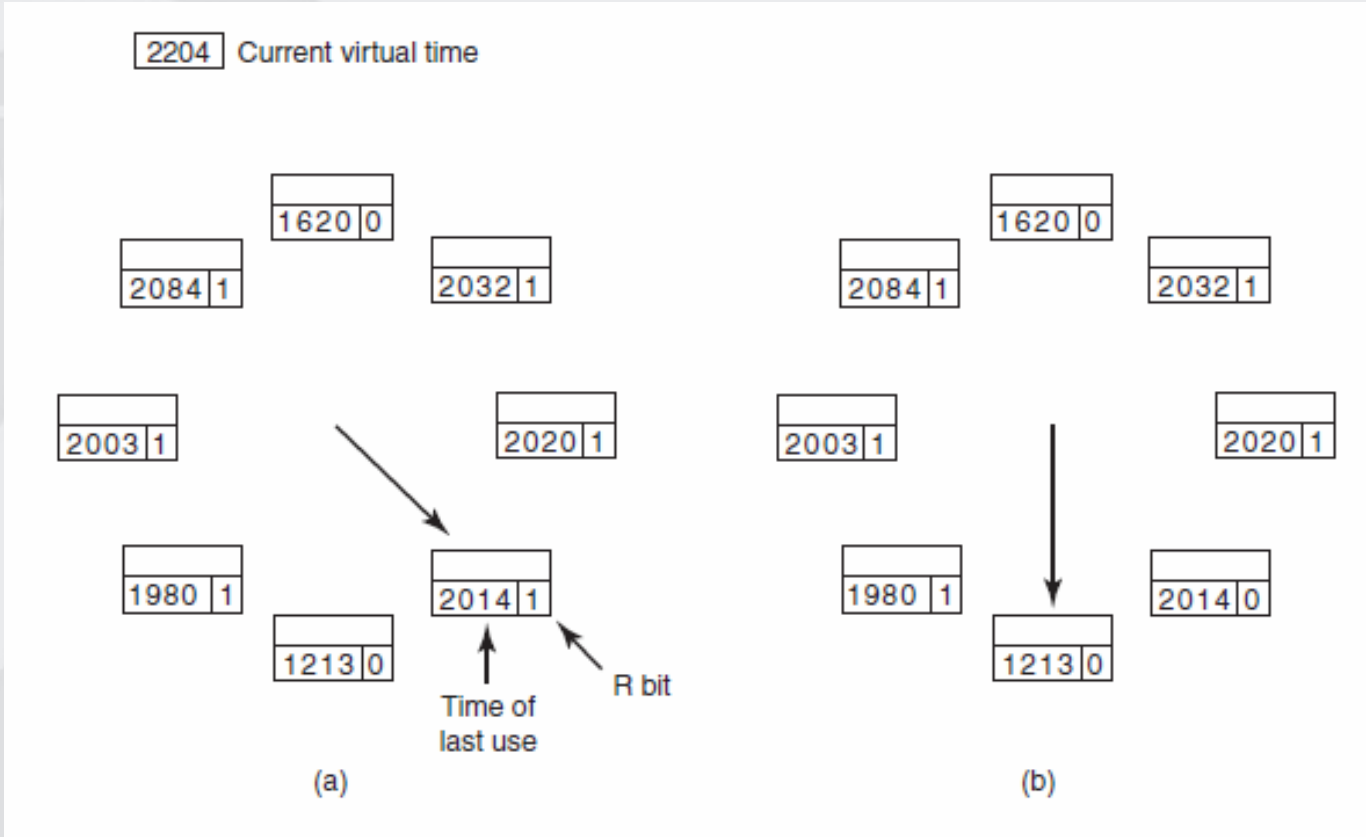
Stabilisce quando una pagina deve essere trasferita in memoria

- La paginazione su richiesta (demand paging) porta una pagina in memoria solo quando si ha un riferimento a quella
  - Si hanno molti page fault quando un processo parte
- Prepaging porta in memoria più pagine del necessario
  - È più efficiente se le pagine su disco sono contigue

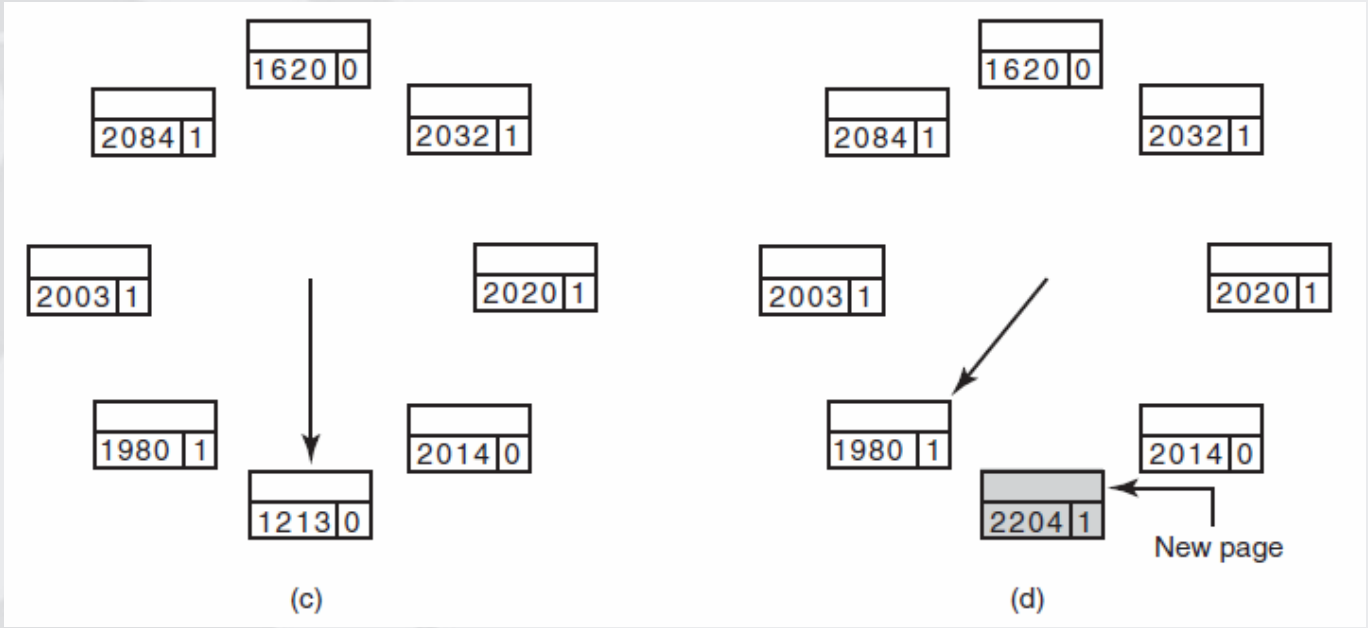
Prepaging

- Per risolvere il problema del thrashing molti sistemi a paginazione utilizzano il prepaging prima di eseguire un processo, vengono caricate in memoria le pagine del working set relative al processo
- Per implementare il modello Working Set è necessario che il Sistema Operativo tenga traccia di quali pagine sono nel working set
  Un modo per controllare questa informazione è quello di utilizzare l'algoritmo di invecchiamento
- Le informazioni sul working set possono essere utilizzate per migliorare le prestazioni dell'algoritmo clock (questo nuovo algoritmo viene detto wsclock): la pagina viene sostituita solo se non appartiene al working set

WSClock - R=1


2204 Current virtual time
1620 0
2084 1
2032 1
2003 1
2020 1
1980 1
2014 1
1213 0
Time of last use
R bit
(a)
1620 0
2084 1
2032 1
2003 1
2020 1
1980 1
2014 0
1213 0
(b)

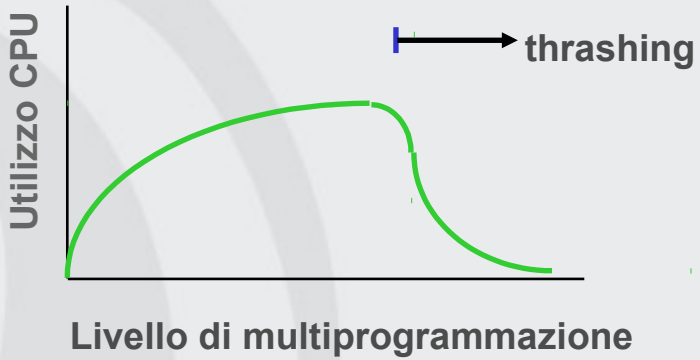
WSClock - R=0


1620 0
2084 1
2032 1
2003 1
2020 1
1980 1
2014 0
1213 0
(c)
1620 0
2084 1
2032 1
2003 1
2020 1
1980 1
2014 0
2204 1
New page
(d)

Algoritmi Locali e Globali

- Come dovrebbe essere allocata la memoria tra i processi eseguibili e concorrenti?
- Gli algoritmi di rimpiazzamento di pagine che usano strategie **locali** assegnano a ogni processo una quantità fissa di memoria
- Gli algoritmi **globali** allocano dinamicamente i frame tra i processi eseguibili
- In generale gli algoritmi globali danno risultati migliori, in particolare quando la dimensione del working set varia durante la ***vita*** di un processo, tuttavia il sistema deve continuamente decidere quanti frame assegnare a ogni processo

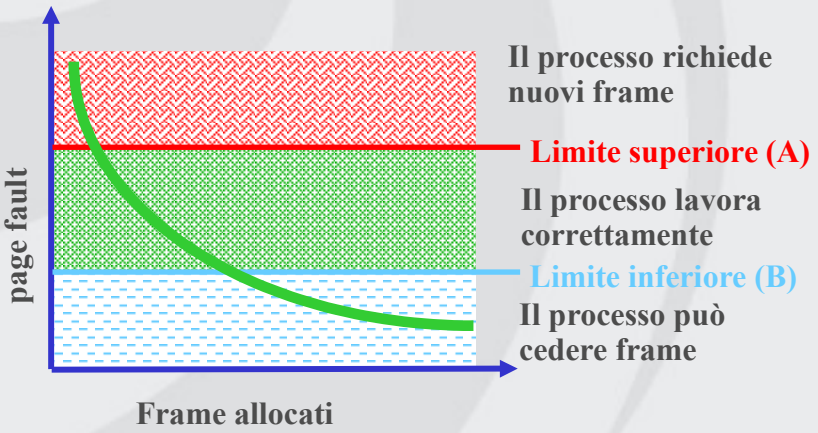
Thrashing



Spesso il thrashing è causato da un numero eccessivo di processi in memoria

Page Fault Frequency

- Utilizzando la gran parte degli algoritmi la frequenza di fault decresce all'aumentare del numero di pagine assegnate



L'algoritmo Page Fault Frequency cerca di mantenere la frequenza degli errori in un intervallo ragionevole:

- Se il numero di processi è troppo grande per mantenerli tutti sotto **A** qualche processo viene rimosso
- Se un processo è sotto **B** allora ha troppa memoria e parte delle sue pagine possono essere utilizzate da altri processi

Paging daemon

- È un processo in background che viene risvegliato periodicamente per ispezionare lo stato della memoria
- Se il numero di frame liberi in memoria non è sufficiente si occupa di selezionare, tramite l'algoritmo di rimpiazzamento pagine prescelto, le pagine da eliminare (se queste sono state modificate, vengono riscritte su disco)
- Vantaggio: migliori prestazioni rispetto alla ricerca di un frame libero nel momento in cui serve

Dimensione delle pagine

- La dimensione delle pagine è un parametro del sistema, le motivazioni nella scelta rispondono a esigenze contrastanti:
- A favore di pagine ***piccole***:
  - Un blocco di memoria non riempirà esattamente un numero intero di pagine, mediamente quindi metà dell'ultima pagina viene *sprecata*
- A favore di pagine ***grandi***:
  - con pagine piccole è necessaria una tabella delle pagine grande
  - il trasferimento di una pagina piccola da disco richiede quasi lo stesso tempo di una pagina grande

Esempio

- **s** dimensione media processo (128K)
- **p** dimensione pagine
- **e** dimensione di un elemento nella tabella delle pagine (8 byte)

- **s/p** numero di pagine in memoria e quindi se/p dimensione tabella
- **p/2** memoria sprecata

il costo allora è dato da `costo = se/p + p/2`

derivando rispetto a p

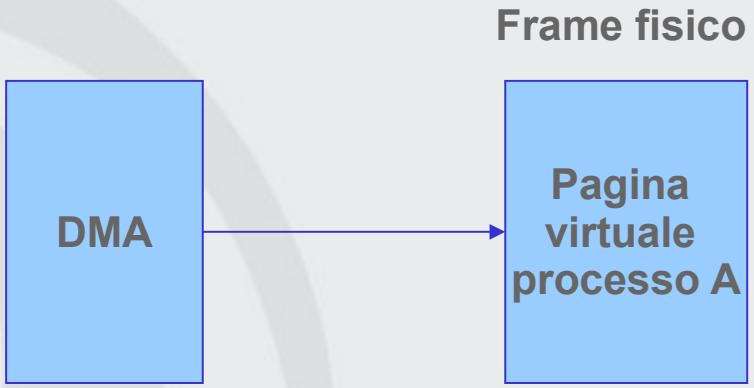
$-se/p^2 + 1/2 = 0$ da cui `p = sqrt(2se)` (=1448 byte in pratica 1 o 2 K

La maggior parte dei SO usa dimensioni di pagina da 512 byte a 64K

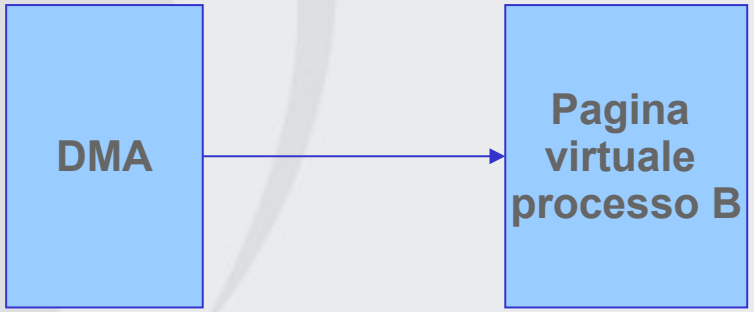
Interfaccia della memoria virtuale

- In alcuni sistemi avanzati i programmatori hanno un certo controllo sulla mappa di memoria
- Un motivo per consentire ai programmatori il controllo sulla mappa di memoria è di consentire a due o più programmi di condividere la stessa memoria: se è possibile dare un nome ad una zona di memoria, un processo può darlo ad un altro processo in modo che quest'ultimo possa inserire la pagina nella sua tabella
- La condivisione delle pagine può implementare un sistema a scambio di messaggi ad elevate prestazioni

Blocco di pagine in memoria



L'algoritmo globale toglie il frame ad A e lo assegna a B



Il DMA va a scrivere i dati nello spazio di B, devo bloccare la pagina in memoria (pinning)

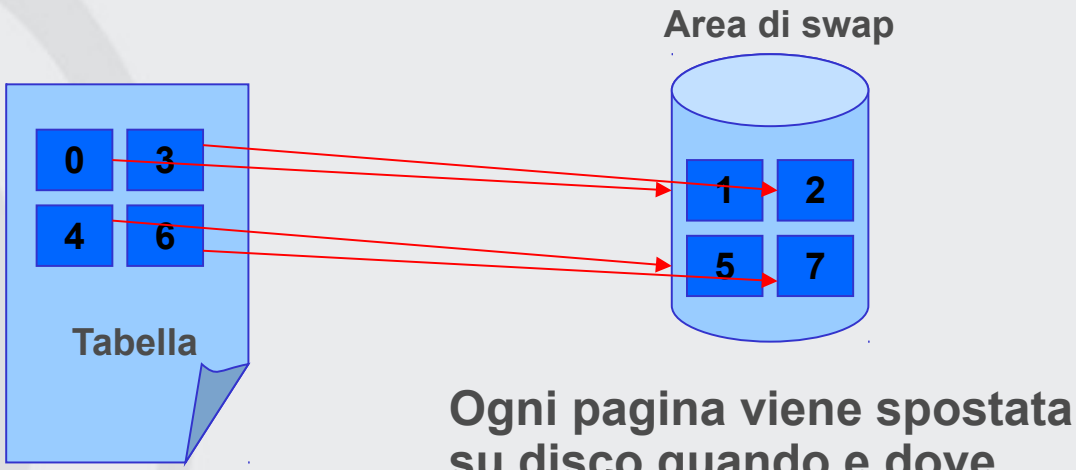
Memoria secondaria

L'indirizzo virtuale mi permette di ottenere l'indirizzo su disco

Due approcci:

- Tutto il processo viene caricato nell'area di swap
- Tutto il processo viene caricato in memoria

Memoria secondaria



Ogni pagina viene spostata su disco quando e dove necessario

Non occorre duplicare codice, file mappati in memoria, …

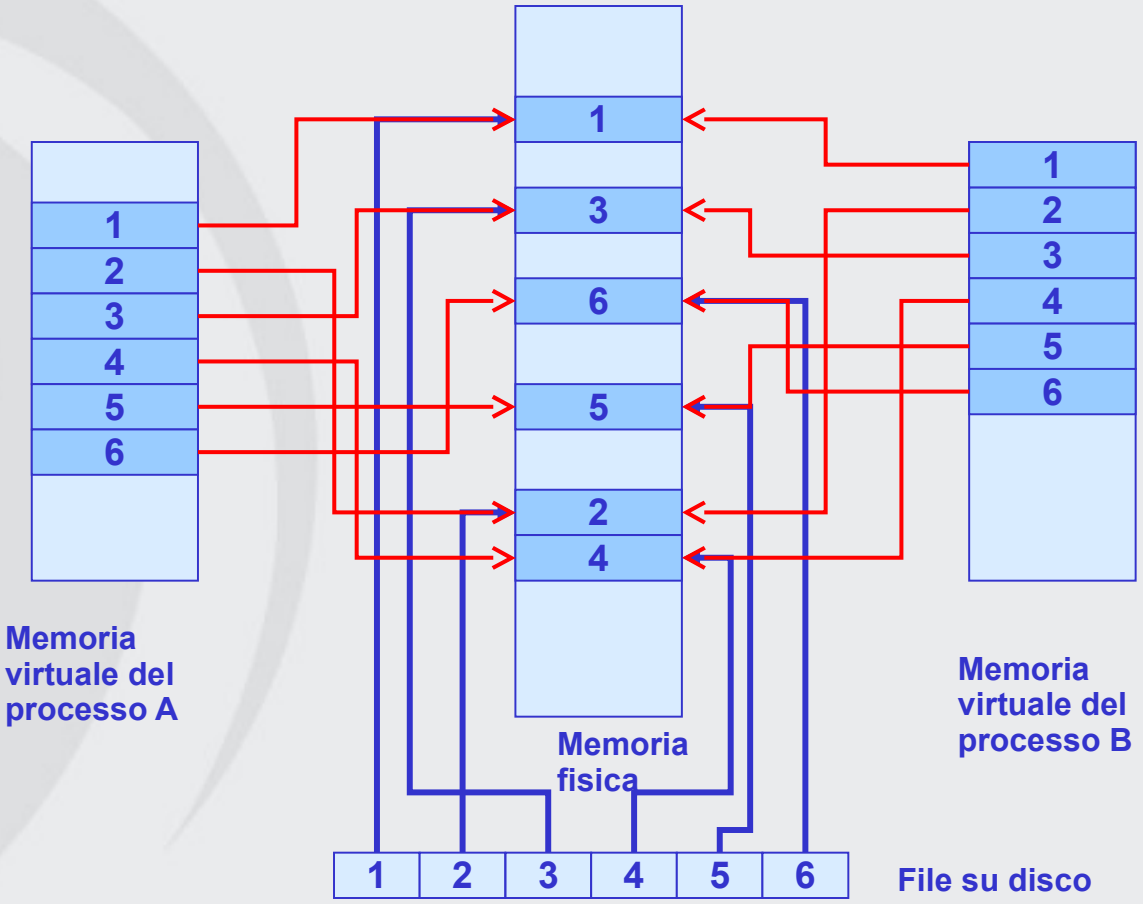
Memory-Mapped File

- Memory-mapped file I/O consente che le operazioni di I/O possano essere trattate come un normale accesso in memoria *mappando* i blocchi su disco su pagine in memoria

- Quando si richiede l'uso di un blocco su disco, questo viene caricato in memoria
  - L'accesso risulta semplificato in quanto non si usano più le chiamate di sistema read() write()

- Più processi possono condividere in memoria gli stessi file

Memory Mapped Files



Memory Mapped Files

```
#include <sys/mman.h> // Memory Management

int main(int argc, char *argv[])
{
  char *pt, pt1[1024];
  int fd, len=sizeof(pt1), offset=0;

  fd = open(__FILE__, O_RDONLY);
  pt = mmap(NULL, len, PROT_READ,MAP_SHARED,fd, offset);

  memcpy(pt1, pt, len);

  return 0;
}
```

Segmentazione

- Con l'approccio precedente la memoria virtuale è unidimensionale (gli indirizzi vanno da 0 all'indirizzo massimo): può essere utile mantenere due o più spazi di indirizzamento separati

Esempio

Un compilatore ha molte tabelle che vengono costruite durante la compilazione e che crescono durante la compilazione

- In uno spazio di indirizzamento unidimensionale ci possono essere tabelle con molto spazio libero e altre completamente occupate o addirittura che necessitano altro spazio
- Ci sono alcune possibili soluzioni ma ciò che effettivamente serve è liberare l'utente dal compito di gestire tabelle che si espandono e contraggono

Segmentazione

- La soluzione diretta e generale a problemi di questo tipo è la segmentazione
- Ogni segmento è una sequenza lineare di indirizzi da 0 a un massimo
  - la lunghezza di ogni segmento può variare da 0 a un massimo consentito e può variare durante l'esecuzione
  - segmenti diversi possono avere lunghezze diverse

Segmentazione

- Per specificare un indirizzo in una memoria segmentata (o bidimensionale), il programma deve fornire un indirizzo costituito da due parti:
  1. numero di segmento
  2. indirizzo all'interno del segmento
- un segmento può contenere una procedura, un array, uno stack o un insieme di variabili ma solitamente non contiene un insieme misto di questi
- Un segmento è un'entità logica della quale il programmatore è cosciente e utilizza come una singola entità logica

Vantaggi della segmentazione

- Semplifica la gestione di strutture dati che crescono o diminuiscono
- Se ogni procedura occupa un segmento separato, con indirizzo di inizio all'interno del segmento pari a 0, la segmentazione semplifica il linking di procedure compilate separatamente
- Facilita la condivisione di procedure o dati tra alcuni processi
- Dato che ogni segmento è un'entità logica nota al programmatore (procedura, array o stack), segmenti diversi possono avere diversi tipi di protezione

Confronto fra segmentazione e paginazione

Considerazioni	Paginazione	Segmentazione
Il programma è a conoscenza del metodo?	No	Sì
Quanti spazi di indirizzamento ci sono?	1	Molti
È possibile che lo spazio di indirizzamento superi la dimensione fisica della memoria?	Sì	Sì
È possibile distinguere e proteggere separatamente procedure e dati?	No	Sì
È facile gestire tabelle di dimensioni variabili?	No	Sì
Facilita la condivisione di procedure fra gli utenti?	No	Sì
Perché è stato proposto questo metodo?	Per avere uno spazio di indirizzamento di grandi dimensioni	Per poter distinguere codice e dati in spazi di indirizzamento logicamente separati e per facilitare la condivisione e la protezione

Implementazione della segmentazione pura

- L'implementazione di paginazione e segmentazione ha una differenza fondamentale: le pagine hanno dimensione fissa, i segmenti variabile
- Se segmenti vengono continuamente caricati e sostituiti in memoria, dopo un certo tempo si formeranno in memoria delle zone non utilizzate:
  è il fenomeno del **checkerboarding** (frammentazione esterna)
- Una possibile soluzione è la compattazione

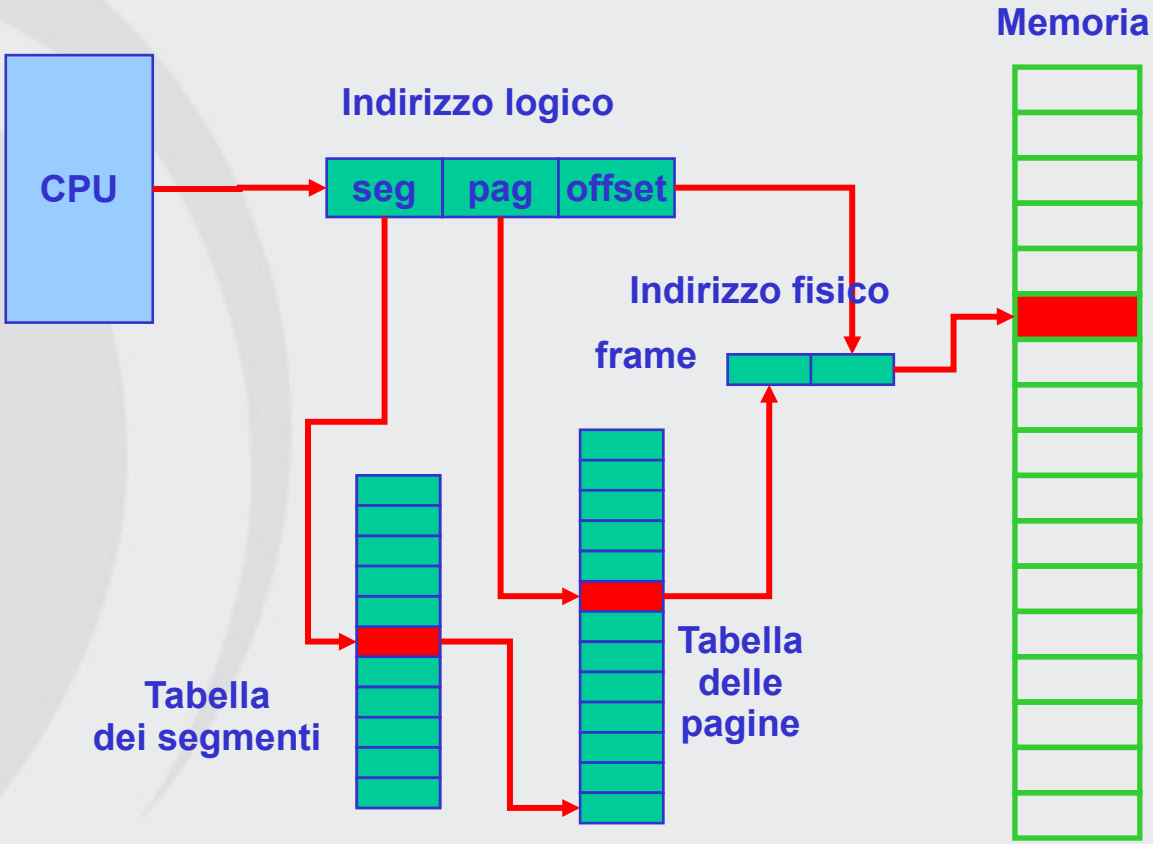
Segmentazione con paginazione

- La tecnica è stata proposta per sfruttare contemporaneamente i vantaggi della paginazione e della segmentazione
- Un indirizzo virtuale è costituito da tre parti (esempio MULTICS)

Indice di segmento	Numero di pagina	Offset interno alla pagina
18 bit	6 bit	10 bit

I processori INTEL a 32 bit implementano un meccanismo simile

Segmentazione con paginazione



CVMLab
Computer Vision
& Multimedia Lab

Esercizi

Università
degli Studi
di Pavia

Rimpiazzamento di pagine

- Descrivere gli algoritmi di rimpiazzamento delle pagine in memoria

  Mostrare come esempio l'effetto con la sequenza di accessi:

  8, 6, 3, 9, 1, 3, 2, 6, 3, 8, 1

  avendo a disposizione 3 pagine.

Rimpiazzamento di pagine

- **Algoritmo ottimo**

  8, 6, 3, 9, 1, 3, 2, 6, 3, 8, 1
  8, 8, 8, 9, 1, 1, 2, 2, 2, 8, 1
  _, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
  _, _, 3, 3, 3, 3, 3, 3, 3, 3, 3

- **Algoritmo FIFO**

  8, 6, 3, 9, 1, 3, 2, 6, 3, 8, 1
  8, 8, 8, 9, 9, 9, 9, 6, 6, 6, 1
  _, 6, 6, 6, 1, 1, 1, 1, 3, 3, 3
  _, _, 3, 3, 3, 3, 2, 2, 2, 8, 8

Rimpiazzamento di pagine

- Algoritmo LRU

  8, 6, 3, 9, 1, 3, 2, 6, 3, 8, 1
  8, 8, 8, 9, 9, 9, 2, 2, 2, 8, 8
  _, 6, 6, 6, 1, 1, 1, 6, 6, 6, 1
  _, _, 3, 3, 3, 3, 3, 3, 3, 3, 3

- Algoritmo a pila

  8, 6, 3, 9, 1, 3, 2, 6, 3, 8, 1
  _, 8, 6, 3, 9, 1, 3, 2, 6, 3, 8
  _, _, 8, 6, 3, 9, 1, 3, 2, 6, 3

Memoria

Descrivere gli algoritmi di rimpiazzamento delle pagine in memoria

Mostrare l'effetto con la sequenza di accessi:

9, 5, 3, 2, 4, 3, 7, 5, 3, 9, 4, 3, 5, 2, 3

avendo a disposizione 4 pagine.

Algoritmo ottimo

9	5	3	2	4	3	7	5	3	9	4	3	5	2	3
9	9	9	9	9	9	9	9	9	9	4	4	4	4	4
	5	5	5	5	5	5	5	5	5	5	5	5	2	2
		3	3	3	3	3	3	3	3	3	3	3	3	3
			2	4	4	7	7	7	7	7	7	7	7	7

Algoritmo FIFO

9	5	3	2	4	3	7	5	3	9	4	3	5	2	3
9	9	9	9	4	4	4	4	4	9	9	9	9	9	9
	5	5	5	5	5	7	7	7	7	4	4	4	4	4
		3	3	3	3	3	5	5	5	5	5	5	2	2
			2	2	2	2	2	3	3	3	3	3	3	3

Algoritmo LRU

9	5	3	2	4	3	7	5	3	9	4	3	5	2	3
9	9	9	9	4	4	4	4	4	9	9	9	9	2	2
	5	5	5	5	5	7	7	7	7	4	4	4	4	4
		3	3	3	3	3	3	3	5	5	5	5	9	9
			2	2	2	2	5	5	3	3	3	3	3	3

Algoritmo LRU – algoritmo a pila

9	5	3	2	4	3	7	5	3	9	4	3	5	2	3
9	5	3	2	4	3	7	5	3	9	4	3	5	2	3
	9	5	3	2	4	3	7	5	3	9	4	3	5	2
		9	5	3	2	4	3	7	5	3	9	4	3	5
			9	5	5	2	4	4	7	5	5	9	4	4

				9	9	5	2	2	4	7	7	7	9	9
						9	9	9	2	2	2	2	7	7

Tempo medio di accesso

Calcolare la probabilità massima di page fault perché si abbia un tempo medio di accesso ai dati di 12 nanosecondi, con le seguenti ipotesi:

* tempo di accesso in memoria 10 nanosecondi.
* tempo di accesso a disco 30 millisecondi.
* percentuale di pagine modificate 33%.

EAT =

p * accesso con page fault +
(1-p) accesso senza page fault

Tempo medio di accesso

Calcolare la probabilità massima di page fault perché si abbia un tempo medio di accesso ai dati di 12 nanosecondi, con le seguenti ipotesi:

* tempo di accesso in memoria 10 nanosecondi.
* tempo di accesso a disco 30 millisecondi.
* percentuale di pagine modificate 33%.

2/3*30+1/3*(30+30)=40 ms (tempo di accesso a disco pesato)

$P*40\ 10^6 + (1-P)\ 10 < 12 \quad \sim \quad P*40\ 10^6 < 2$

$P = 1 / (20\ 10^6)$

Working set

- Dato il codice seguente (si può ipotizzare che sia parte di una funzione) dare una stima della dimensione del working set durante l'esecuzione del ciclo.

```
#define SIZE 10000000 // 10^7

int *vettore = malloc(SIZE*sizeof(int));
int somma[1000], i;
// ...
for(i=0; i<SIZE; i++)
      somma[i % 1000] += vettore[i];
```