# Lezione 20 [29/11/22]

## Sostituzione delle pagine

In presenza di un page fault dovuto ad un riferimento ad una pagina non in memoria, si dovrà portare la pagina in memoria

Un problema da discutere è la **sovrallocazione:** durante l'esecuzione di un processo utente si verifica un page fault. Il S.O. determina la locazione in cui risiede al pagina desiderata, ma poi scopre che la lista dei frame liberi è vuota: tutta la memoria è in uso

Se nessun blocco di memoria è libero, si potrà liberarne uno inutilizzato, scrivendo il suo contenuto nell'area di avvicendamento e modificando la tabella delle pagine (e le altre tabelle) per indicare che quella pagina non è più in memoria

Il blocco di memoria liberato si potrà usare per caricare la pagina che ha causato l'eccezione



La procedura di servizio dell'eccezione di pagina mancante verrà modificata in modo da includere l'eventuale sostituzione della pagina

Solo le pagine che sono state modificate vengono riscritte su disco

La sostituzione delle pagine completa la separazione tra memoria logica e memoria fisica: possiamo avere una memoria virtuale molto ampia con una memoria fisica più piccola

### Schema di base

La procedura di servizio dell’eccezione di pagina mancante deve essere modificata in modo da includere l’eventuale sostituzione della pagina:

1. Si individua la locazione nel disco della pagina richiesta
2. Si cerca un blocco di memoria libero:
    a. Se esiste, lo si usa
    b. Altrimenti si impiega un algoritmo di sostituzione delle pagine per scegliere un blocco di memoria “vittima”
    c. Si scrive la pagina “vittima” nel disco; si modificano adeguatamente le tabelle delle pagine e dei frame
3. Si scrive la pagina richiesta nel blocco di memoria appena liberato e si modificano le tabelle delle pagine e dei blocchi di memoria
4. Si riavvia il processo utente

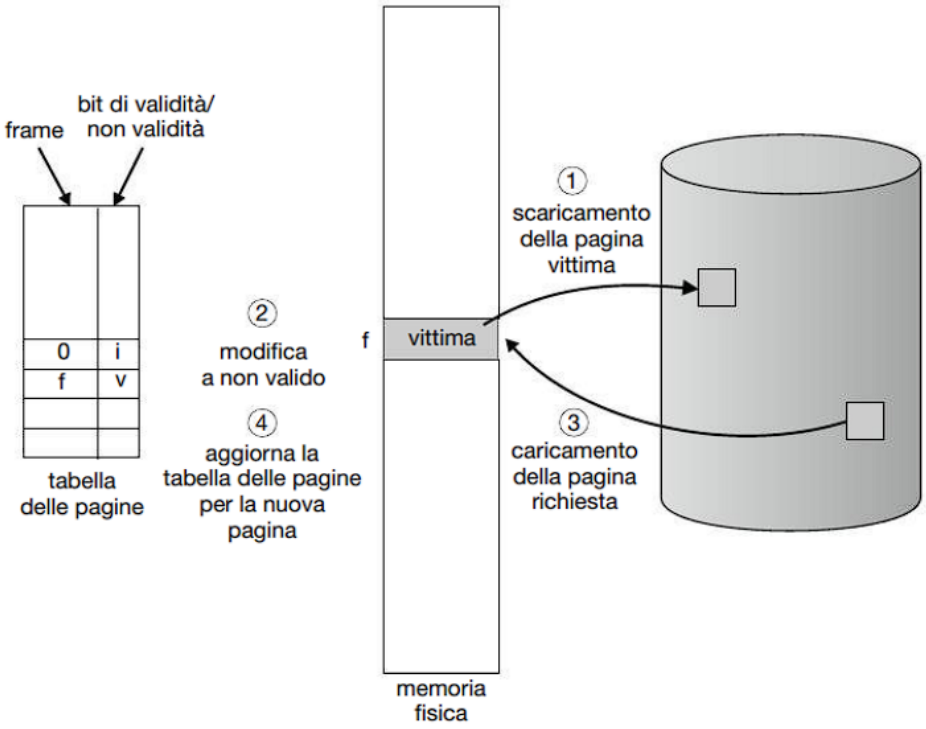
L’architettura fisica del sistema di calcolo può disporre di un **bit di modifica (modify dirty bit)**, associato ad ogni pagina, che si imposta a 1 automaticamente ogni volta che la pagina viene modificata



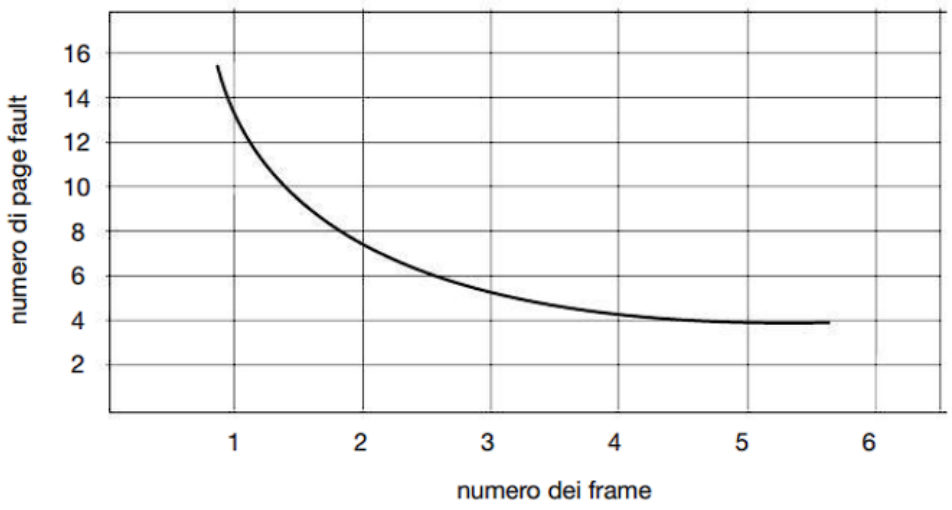
Quando si sceglie che pagina sostituire, si controlla il suo bit di modifica:

- se è 1 → la pagina è stata modificata quindi deve essere scritta nel disco
- se è 0 → la pagina non è stata modificata, quindi non c’è bisogno che venga scritta nel disco: la pagina c’è già

Sostituzione di una pagina:




Numero di assenze di pagina rispetto al numero di frame:




## Algoritmi di sostituzione delle pagine

Per realizzare la paginazione su richiesta è necessario risolvere due problemi principali:

- **Algoritmo di allocazione dei frame**
- **Algoritmo di sostituzione delle pagine**



Esistono molti algoritmi di sostituzione delle pagine, probabilmente ogni S.O. ha un proprio schema di sostituzione

Scopo:

- Minimizzare il tasso di page fault

Valutazione di un algoritmo:

- Va fatta effettuandone l'esecuzione su una particolare **successione di riferimenti** alla memoria e calcolando il numero di page fault

Queste successioni si possono generare artificialmente oppure derivano dall'analisi di un sistema reale

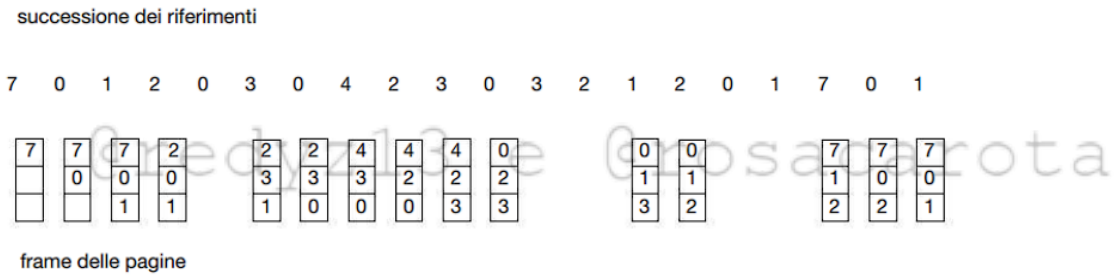
Aumentando il numero dei frame il numero di page fault diminuisce

Negli esempi seguenti, consideriamo, per una memoria con tre frame, la seguente successione di riferimenti:

- $7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1$

## Sostituzione delle pagine secondo l'ordine di arrivo (FIFO)

L'algoritmo di sostituzione delle pagine più semplice è un algoritmo FIFO in cui se si deve sostituire una pagina si sostituisce quella presente in memoria da più tempo (si associa a ogni pagina l'istante di tempo in cui quella pagina è stata portata in memoria)



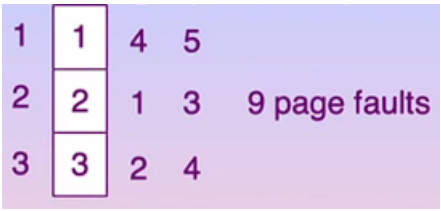

Nell'esempio si hanno complessivamente 15 assenze di pagina

Quest'algoritmo è semplice da capire e da programmare, ma le sue prestazioni non sono buone

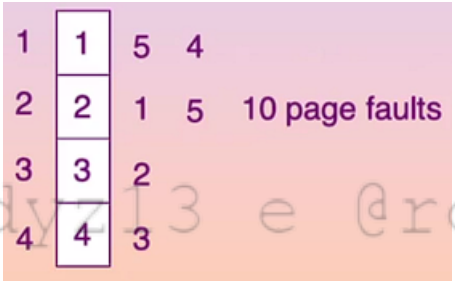
La pagina sostituita potrebbe essere ormai inutile oppure potrebbe essere ancora utile: ad esempio se contiene una variabile molto usata, ma inizializzata in precedenza e quindi sostituirla potrebbe generare successivi page fault

Si consideri invece la seguente successione di riferimenti:

- $1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$
- 3 frames (3 pagine possono contemporaneamente essere in memoria per ogni processo)




Con l'utilizzo di 4 frames:

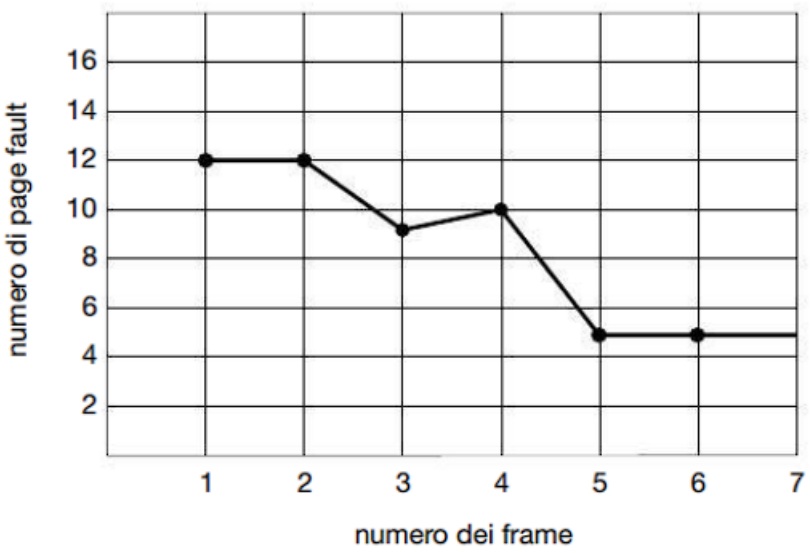





FIFO - **Belady's Anomaly:**

- il tasso di page fault aumenta all'aumentare del numero di frame
  - Più frames → più page fault

FIFO - **L'anomalia di Belady:**




## Sostituzione ottimale delle pagine

Un **algoritmo ottimale di sostituzione delle pagine** è quello che presenta il tasso minimo di page fault e non presenta l'anomali di Belady(a volte detto OPT o MIN): si sostituisce la pagina che non si userà per il periodo di tempo più lungo



L'uso di questo algoritmo assicura la frequenza di assenze di pagina più bassa possibile per un numero fisso di frame

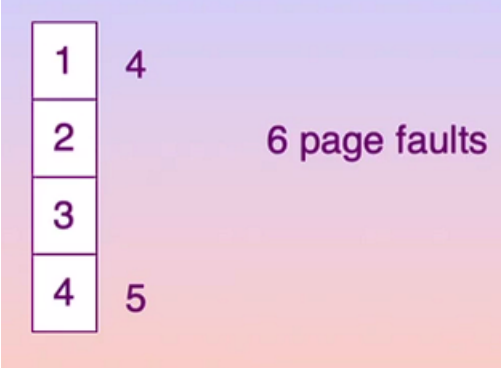
Sfortunatamente l'algoritmo ottimale di sostituzione delle pagine è difficile da realizzare perché richiede la conoscenza futura della successione di riferimenti

- Situazione analoga all'algoritmo SJF di scheduling della CPU

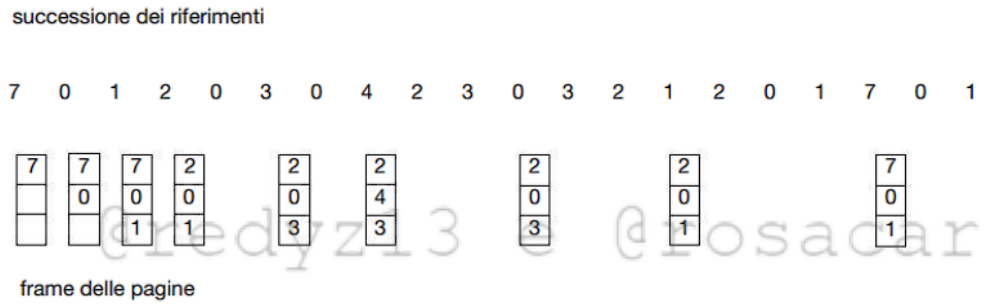
Quindi l'algoritmo ottimale si utilizza soprattutto per studi comparativi e per valutare le prestazioni di altri algoritmi

Esempio per 4 frames

Successione di riferimenti: $1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$




Con la sequenza precedente:




Con solo 9 assenze di pagina la sostituzione ottimale risulta migliore della FIFO (15 assenze)

- Ignorando le prime 3 assenze che si verificano con tutti gli algoritmi, la sostituzione ottimale è 2 volte migliore della FIFO

Nessun algoritmo potrà gestire questa successione di riferimenti a tre frames con meno di 9 page fault

## Sostituzione delle pagine usate meno frequentemente (LRU)

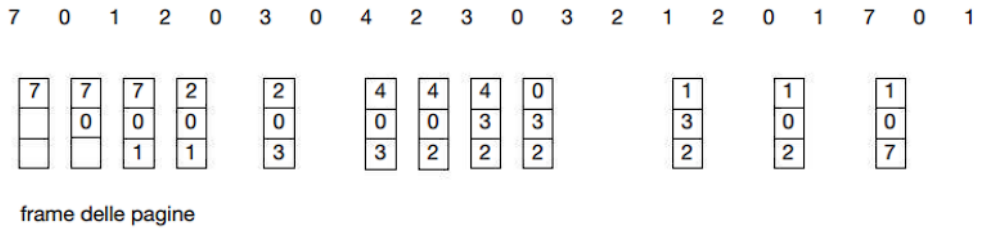
La differenza principale tra un algoritmo FIFO è che esso guarda a quando la pagina è stata caricata, invece un algoritmo OTP guarda a quando una pagine è stata **usata**

Sostituisce la pagina che non è stata usata per il periodo di tempo più lungo

Successione di riferimenti: $1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5$




Con la sequenza precedente:




## Implementare LRU

Problema:

- Determinare un ordine per i frame secondo il momento dell’ultimo uso

Due soluzioni:

- Contatori
- Stack

**Contatori:**

- Si assegna alla CPU un contatore che si incrementa ad ogni riferimento alla memoria
- Si assegna ad ogni entry della tabella delle pagine un ulteriore campo, **momento di utilizzo**
- Ogni volta che si fa riferimento ad una pagina viene copiato il valore del contatore della CPU nel campo momento di utilizzo corrispondente per quella pagina nella tabella delle pagine
  - In questo modo è sempre possibile conoscere il momento in cui è stato fatto l’ultimo riferimento ad ogni pagina
- Quando una pagina deve essere sostituita si guardano i valori dei contatori e si sostituisce la pagina con il valore associato più piccolo



**Stack:**

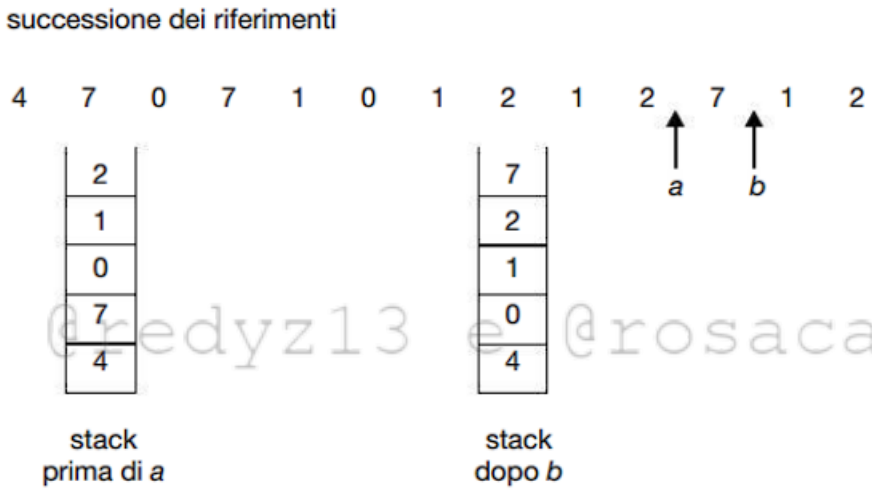
- Mantenere uno stack dei numeri di pagina in una lista doppiamente concatenata
- Ogni volta che si fa riferimento ad una pagina, la si estrae dallo stack e la si mette in cima allo stack stesso
- In questo modo in cima allo stack si trova sempre la pagina usata per ultima, mentre in fondo si trova la pagina usata

meno recentemente

- Nessuna ricerca per la sostituzione di pagina
  - Basta prelevare dal fondo
- Ogni aggiornamento dello stack è un po’ più costoso, bisognerà spostare dei puntatori

Né la soluzione ottimale, né la LRU hanno l’anomalia di Belady e, insieme ad altri con questa caratteristica, fanno parte di una serie di algoritmi chiamati **algoritmi di stack**

Implementazione LRU con stack:






## Sostituzione delle pagine per approssimazione a LRU

Sono pochi i sistemi in grado di avere una architettura (contatori, stack) adatta alla gestione della sostituzione delle pagine usate meno frequentemente (LRU)

Molti sistemi tentano delle approssimazioni

Molte architetture offrono come ausilio un **bit di riferimento:**

- Ad ogni pagina è associato un bit, inizialmente = 0
- Quando la pagina è referenziata (a cui si è fatto riferimento) il bit è impostato a 1
- Rimpiazza la pagina che è a 0 (se ne esiste una)
- In questo modo però non è possibile conoscere l'ordine d'uso delle pagine

## Algoritmo con bit supplementari di riferimento

È possibile conservare in una tabella in memoria una serie di bit per ogni pagina

Ad intervalli regolari (ad es. ogni 100 millisecondi) il sistema operativo può spostare il bit di riferimento di ciascuna pagina nel bit più significativo della sequenza, traslando gli altri a destra e scartando il bit meno significativo

Ad es. se il registro a scorrimento relativo ad una pagina è di 8 bit:

- 00000000 significa che la pagina non è stata usata da 8 periodi di tempo
- 11000000 significa che la pagina è stata usata più recentemente di una che ha 01111111

Considerando questi 8 bit come interi senza segni, la pagina con il numero minore è quella che può essere sostituita



## Algoritmo con seconda chance

L'algoritmo di base per la sostituzione con la seconda chance è un algoritmo di sostituzione di tipo FIFO

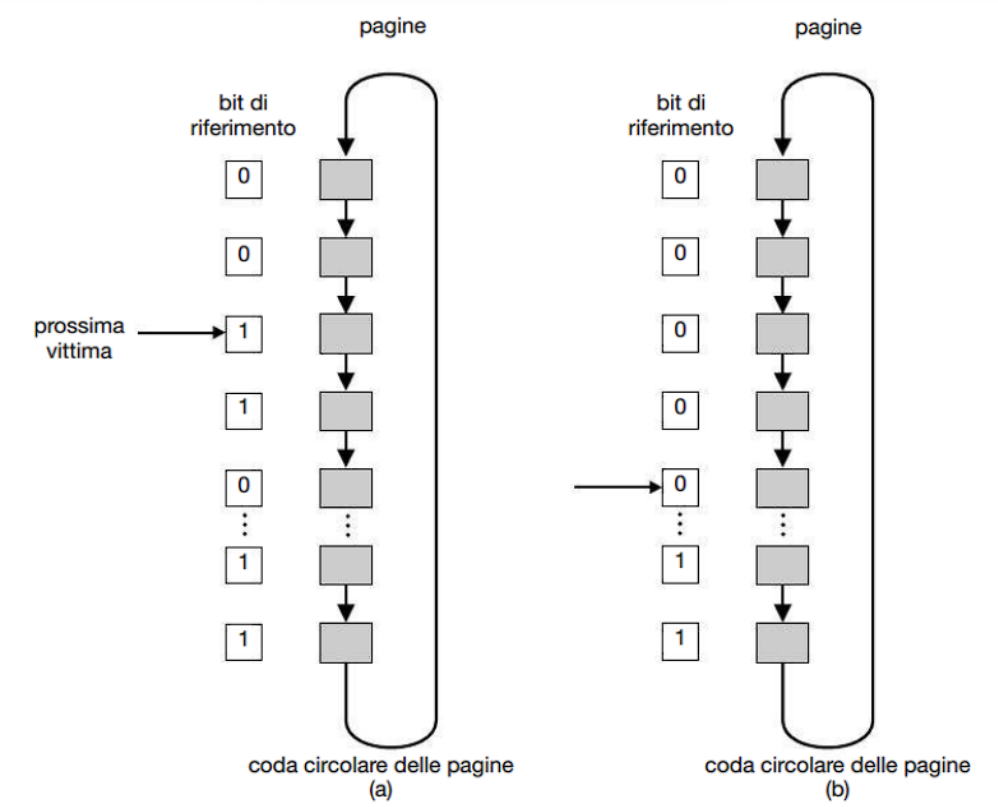
Le pagine sono disposte in una lista circolare

Quando occorre selezionare una pagina vittima inizia la scansione della lista:

- Se una pagina ha il bit di riferimento a 1 lo si pone a 0 e si passa alla successiva (la pagina rimane in memoria - le si dà una seconda chance)
- Altrimenti se il bit di riferimento è 0 la si seleziona per essere sostituita

Un metodo per implementare la seconda chance (detta anche orologio) è l’utilizzo di una coda circolare in cui il puntatore (detto lancetta) indica qual è la prima pagina da sostituire. Quando serve un frame, si fa avanzare il puntatore finché non trovi in corrispondenza di una pagina con il bit dii riferimento a 0; ad ogni passo si azzera il bit edi riferimento che si è esaminato. una volta che si è trovata la pagina vittima, si sostituisce e si inserisce la pagina nuova nella posizione di quella di prima

Se tutti i bit di riferimento sono impostati a 1 , il puntatore percorre un ciclo lungo tutta la coda dando a tutti una seconda chance: si trasforma in una sostituzione FIFO






## Algoritmo con seconda chance migliorato

Raffinamento dell'algoritmo della seconda chance ottenuto considerando il bit di riferimento ed il bit di modifica come una coppia ordinata con cui si possono ottenere quindi:

- (0,0) Non usato di recente, non modificato
  - Migliore pagina da sostituire
- (0,1) Non usato di recente, modificato
  - Non ottimale perché prima di essere sostituita deve essere scritta in memoria secondaria

- (1,0) Usato di recente, non modificato
  - Probabilmente la pagina sarà presto ancora usata
- (1,1) Usato di recente, modificato
  - Probabilmente la pagina sarà presto ancora usata e dovrà essere scritta in memoria secondaria prima di essere sostituita

Ogni pagina rientra in una di queste quattro classi

Alla richiesta di una sostituzione si applica la stessa strategia della seconda chance e si sostituisce la prima pagina che si trova nella classe minima non vuota

Differenza con il precedente: si dà la preferenza alle pagine modificate, in modo da ridurre gli I/O

## Sostituzione delle pagine basata su conteggio

Tenere un contatore del numero di riferimenti che sono stati fatti ad ogni pagina



Algoritmo **LFU (Least Frequently Used):**

- Sostituisce la pagina con il più basso conteggio

  Il problema di questo algoritmo è che, nel caso ci fosse una pagina che inizialmente viene usata molto (quindi ha un conteggio molto alto) essa rimarrà in memoria anche s e non è più necessaria

Algoritmo **MFU (Most Frequently Used):**

- Sostituisce la pagina con il conteggio più alto

  Si basa sul fatto che una pagina con conteggio basso potrebbe essere stata appena inserita e non utilizzata

## Algoritmi con memorizzazione transitoria delle pagine (Algoritmi con buffering delle pagine)

I sistemi hanno generalmente un gruppo di frame liberi per soddisfare le richieste velocemente **(pool of free frames)**

Quando si verifica un page fault si sceglie un frame vittima, ma se deve essere scritto in memoria secondaria si trasferisce la pagina richiesta in un frame libero del gruppo

Questa procedura permette al processo di ricominciare senza attendere che la pagina vittima sia scritta in memoria secondaria

Pagine modificate già scelte come vittime sono poi scritte sul disco periodicamente in background ed aggiunte al pool of free frames

È anche possibile la ricerca nel pool dei frame liberi in memoria di una pagina precedentemente sostituita e nuovamente necessaria

Se il frame non è stato riallocato, questa è probabilmente ancora in memoria e non è stata sovrascritta



## Allocazione delle pagine

Ogni processo ha bisogno di un numero minimo di pagine in memoria

Inoltre alcune istruzioni potrebbero essere costituite da più parole macchina e quindi indirizzare memoria che è a cavallo di più pagine

Il numero minimo di frame per ciascun processo è definito dall’architettura

Il numero massimo è invece definito in base alla quantità di memoria fisica disponibile

Esistono due principali schemi di allocazione:

- **Allocazione statica (o fissa)**
- **Allocazione dinamica (o a priorità)**

## Allocazione uniforme e proporzionale

**Allocazione uniforme:**

- Avendo $m$ frame ed $n$ processi, si assegnano $\frac{m}{n}$ frame a ciascun processo
  - Ad es. se 100 frame e 5 processi, ognuno prende 20 frame

**Allocazione proporzionale:**

- Si assegna la memoria disponibile ad ogni processo in base alle dimensioni di quest'ultimo
  - $s_i$ = dimensione del processo $p_i$
  - $S = \sum s_i$ dimensione di tutti i processi
  - $m$ = numero totale di frame
  - $a_i$ = numero di frame allocati per $p_i = \frac{s_i}{S} \times m$
    Si sceglie ogni $a_i$ in modo che sia un intero maggiore del numero minimo di frame richiesti e in modo che la somma di tutti gli $a_i$ non sia maggiore di m
- Esempio:
  - $m = 64$
  - $s_1 = 10$
  - $s_2 = 127$
  - $a_i = \frac{10}{137} \times 64 \approx 5$
  - $a_2 = \frac{127}{137} \times 64 \approx 59$

### Allocazione a priorità

Una variante dello schema precedente è quella di usare uno schema di allocazione proporzionale basato sui valori delle priorità piuttosto che delle dimensioni

Ad esempio se il processo $P_i$ genera un page fault:

- Per la sostituzione si selezionerà uno dei suoi frame
- Oppure il frame di un processo con priorità inferiore rispetto a $P_i$

Si potranno avere anche scelte basate su combinazioni di dimensioni e priorità

## Sostituzione globale e locale

Un importante fattore che riguarda il modo in cui si assegnano i frame è la modalità di sostituzione delle pagine

Gli algoritmi di sostituzione si possono classificare in due categorie generali:

- **Sostituzione globale:**
  - Permette di selezionare un frame di sostituzione a partire dall'insieme di tutti i frame, anche se quel frame è correntemente allocato a qualche altro processo
  - Un processo può quindi prendere un frame da un altro processo
- **Sostituzione locale:**
  - Vengono considerati solo i frame allocati al processo



Con la sostituzione locale il numero di frame assegnati ad un processo non cambia

Con quella globale potrebbe accadere che per la sostituzione si selezionino frame allocati ad altri processi, aumentando quindi il numero di frame assegnati a quel processo

Un problema per la sostituzione globale è che non si può controllare il proprio tasso di page fault poiché non dipende solo dalla propria paginazione ma anche da quella degli altri

- Generalmente la sostituzione globale genera maggiore produttività ed è il metodo più usato

## Paginazione degenere (thrashing)

Se un processo non ha abbastanza pagine, il tasso di page fault aumenta

Scenario:
Il S.O. controlla l'utilizzo di CPU, se questo è basso aggiunge processi per aumentare il livello di multiprogrammazione. Nel caso in cui questo nuovo processo che entra in esecuzione abbia bisogno di più frame rispetto agli altri e in caso di sostituzione delle pagine globale ci saranno page fault da parte sia di questo processo che da parte dei processi a cui vengono sottratte le pagine. Per avere lo scaricamento e il caricamento i processi vengono messi nella coda del dispositivo di paginazione, svuotando così la coda dei processi pronti. Se la coda dei processi pronti risulta vuota, il lo scheduler inizia ad aumentare di nuovo il grado di multiprogrammazione mandando processi in esecuzione: si spenderà, quindi. Tutto il tempo nella paginazione piuttosto che per l'esecuzione ei processi stessi. Siamo in una situazione di trashing



Detto con le parole delle slide in modo brutto:

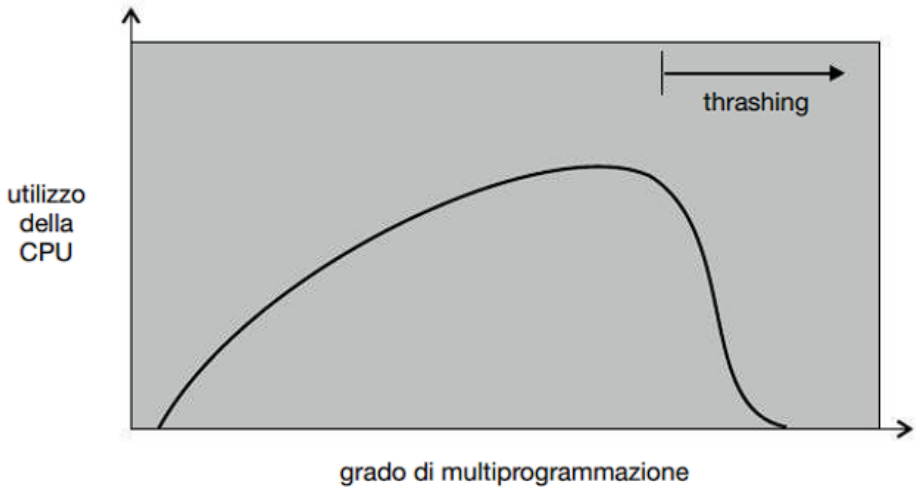
Questo comporta:

- Riduzione utilizzo della CPU
- Quindi il sistema operativo potrebbe ritenere che sia necessario aumentare il livello di multiprogrammazione
- Quindi un altro processo potrebbe essere aggiunto al sistema (potenzialmente rallentandolo ancora di più)

**Thrashing** = si spende più tempo nella paginazione che nella esecuzione dei processi

- Un processo impiega il suo tempo facendo quasi sempre solo swapping di pagine di memoria

Thrashing:






## Modello di località di esecuzione del processo

Gli effetti di questa situazione si possono limitare usando un algoritmo di sostituzione locale o un algoritmo di sostituzione per priorità

Con la sostituzione locale, un processo, anche se ricade nell'attività di paginazione degenere, non può sottrarre frame ad un altro processo e quindi provocarne a sua volta la degenerazione

Per evitare il thrashing occorrerebbe sapere quali (oltre che quante pagine) servono

**Modello di località** di esecuzione del processo:

- Un processo durante la sua esecuzione si muove da una località all'altra
- Una località è un'insieme di pagine usate attivamente insieme
- Un programma è formato da parecchie località diverse che sono sovrapponibili
  - Ad esempio quando si invoca una procedura, questa definisce una nuova "località" in cui si fanno riferimenti alla memoria per le istruzioni della procedura, le variabili locali, etc.

Perché si verifica il thrashing?

- Dimensione della località > numero di frame assegnati al processo

## Modello dell'insieme di lavoro (Modello del working set)



Il modello dell'insieme di lavoro (**working set**) è basato sull'ipotesi di località

Il modello usa un parametro $\Delta$ per definire la finestra dell'insieme di lavoro (**finestra del working set**)

L'idea è quella di esaminare, come approssimazione della località del programma, i più recenti $\Delta$ riferimenti alle pagine (che rappresentano l'**insieme di lavoro** o **il working set**)

Se una pagina è utilizzata allora si trova nel working set altrimenti esce dal working set

- $\Delta$ = **working set window (finesta dell'insieme di lavoro)** = un numero fisso di riferimenti di pagina

    - Ad esempio: 10.000 istruzioni

La caratteristica più importante dell'insieme di lavoro è la sua dimensione

- Calcolando la dimensione dell'insieme di lavoro per ogni processo si può determinare la richiesta totale di frame

$WSS_i$ (working set del processo $P_i$):

- Numero totale di pagine cui $P_i$ si riferisce nel più recente $\Delta$ (varia nel tempo)
    - Se $\Delta$ è troppo piccolo non comprenderà l'intera località
    - Se è troppo grande può sovrapporre parecchie località
    - Se $\Delta = \infty \Rightarrow$ il working set è l'insieme delle pagine toccate durante l'esecuzione del processo

$D = \sum WSS_i$ richiesta globale dei frame

- Se $D > m$ (dove $m$ è il numero di frame liberi) $\Rightarrow$ thrashing
- Se $D > m$, allora occorre sospendere uno dei processi

Una volta scelto $D$ il sistema operativo controlla l'insieme di lavoro di ogni processo e gli assegna un numero di frame sufficiente



Se i frame ancora liberi sono in numero sufficiente si può iniziare un nuovo processo

Se la somma delle dimensioni degli insiemi di lavoro aumenta, superando il numero di frame disponibili, il S.O. individua un processo da sospendere:

- Scrive in memoria secondaria le pagine di quel processo
- Assegna i suoi frame ad altri processi
- Il processo sospeso potrà essere ripreso successivamente

Questa strategia impedisce il trashing, mantenendo il grado di multiprogrammazione più alto possibile ed ottimizzando, quindi, l’utilizzo della CPU

## Frequenza delle assenze di pagine

È possibile utilizzare una strategia basata sul controllo della **frequenza di dei page fault** per prevenire la paginazione degenere

Stabilire un tasso “accettabile” di page fault

- Se il tasso attuale è troppo basso, il processo potrebbe usare troppi frame
- Se il tasso attuale è troppo alto, il processo ha bisogno di più frame

Frequenza dei page fault: