

ALGORITMI DI SOSTITUZIONE DELLE PAGINE

- Quando si verifica un errore di pagina (**page fault**), il sistema operativo deve scegliere una pagina da rimuovere dalla memoria (più correttamente **frame** o **frame di pagina**) per far spazio alla pagina che manca.
- Se la pagina da rimuovere è stata modificata mentre era in memoria, deve essere riscritta sul disco prima di essere sovrascritta.
- Ci sono molti metodi per la scelta della miglior pagina candidata per la rimozione, l'idea di base è di minimizzare il numero di page fault futuri selezionando quelle pagine che sono usate raramente (e che quindi è giusto che siano sul disco e non in memoria).
- Un problema analogo si ha quando occorre sostituire i dati nella cache, quindi i procedimenti che verranno mostrati hanno una valenza del tutto generale.

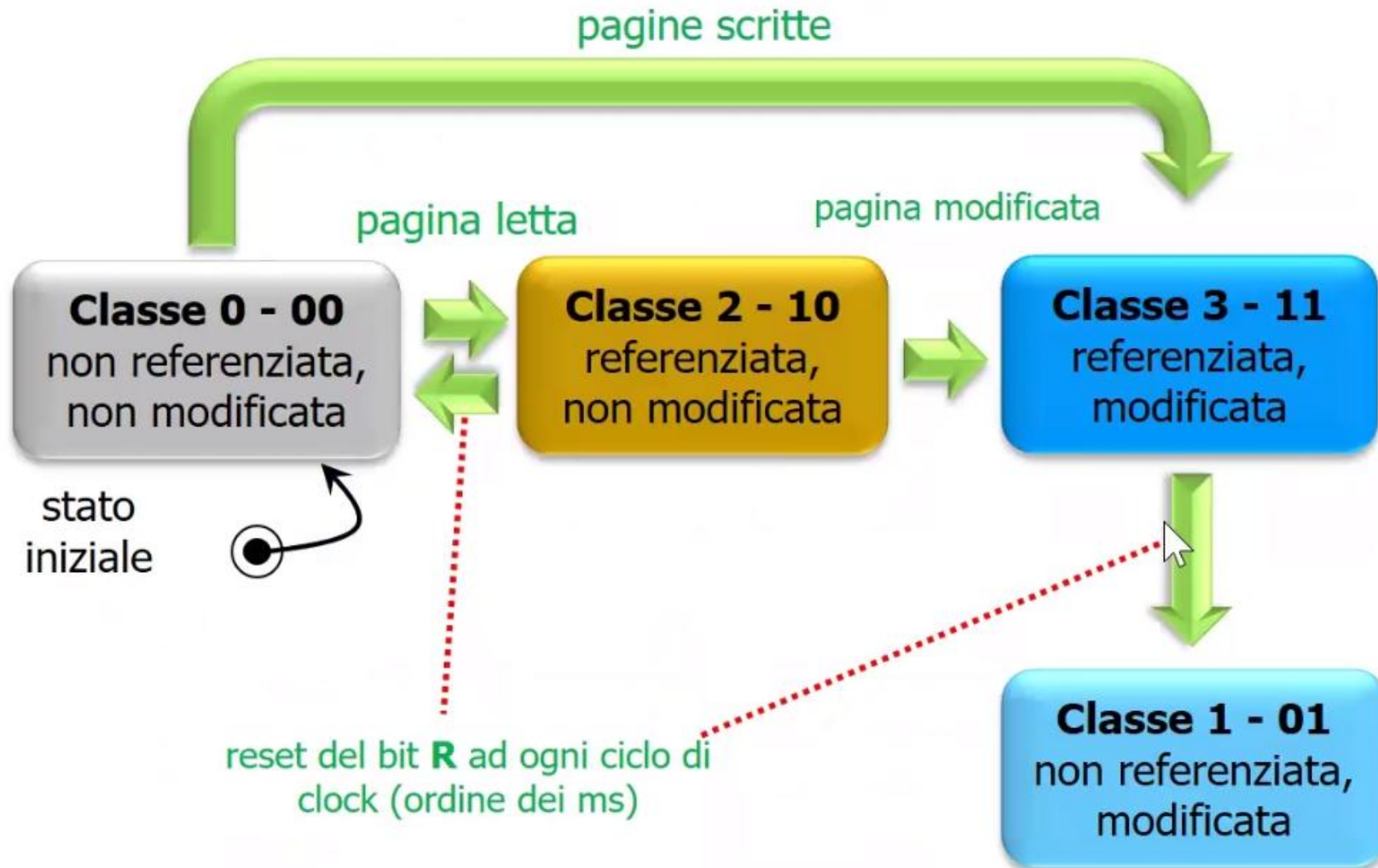
L'algoritmo ottimo di sostituzione delle pagine

- L'algoritmo ottimo di sostituzione delle pagine è facile da descrivere, **ma impossibile da realizzare**.
- Quando accade un **page fault** alcune pagine sono già in memoria, quindi una di queste pagine sarà utilizzata dall'istruzione successiva.
- Se si etichettasse ogni pagina con il numero di istruzioni che sarà eseguito prima che quella pagina sia referenziata, l'algoritmo diventa banale «occorre rimuovere la pagina con il numero più grande».
- Purtroppo al momento dell'errore di pagina, il sistema operativo non sa quando ciascuna pagina verrà referenziata di nuovo.

Not Recently Used (NRU)

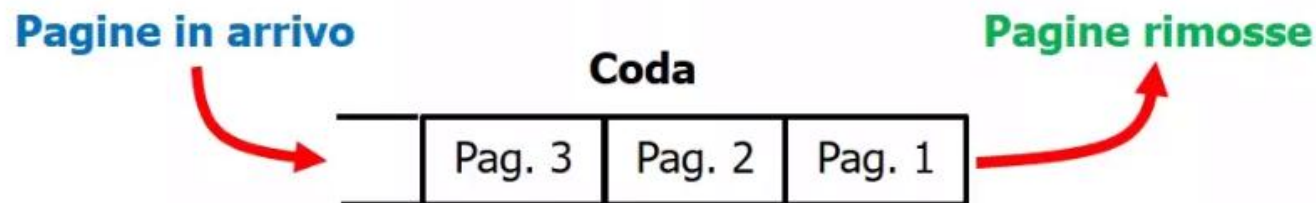
- Per permettere al sistema operativo di raccogliere statistiche utili sull'utilizzo pagina, la maggior parte dei computer con memoria virtuale utilizza due bit di stato associato a ciascuna pagina:
 - **R** viene impostato ogni volta che la pagina è referenziata (in lettura o in scrittura).
 - **M** viene impostato quando la pagina viene scritta.
- I bit sono contenuti in ciascuna riga della tabella delle pagine.
- Quando un processo viene avviato, il sistema operativo imposta a 0 tutti i bit di pagina. Ad ogni impulso di clock il bit R viene azzerato, per distinguere le pagine che non sono state utilizzate recentemente.
- Quando si verifica un **page fault**, il sistema operativo controlla tutte le pagine e li separa in 4 categorie:
 - **Classe 0:** non referenziata (R = 0), non modificata (M = 0).
 - **Classe 1:** non referenziata (R = 0), modificata (M = 1).
 - **Classe 2:** referenziata (R = 1), non modificata (M = 0).
 - **Classe 3:** referenziata (R = 1), modificata (M = 1).

Diagramma di transizione di stato



First In First Out (FIFO)

- Il sistema operativo mantiene una lista di tutte le pagine attualmente in memoria: la più recente in coda e quella meno recente in testa.

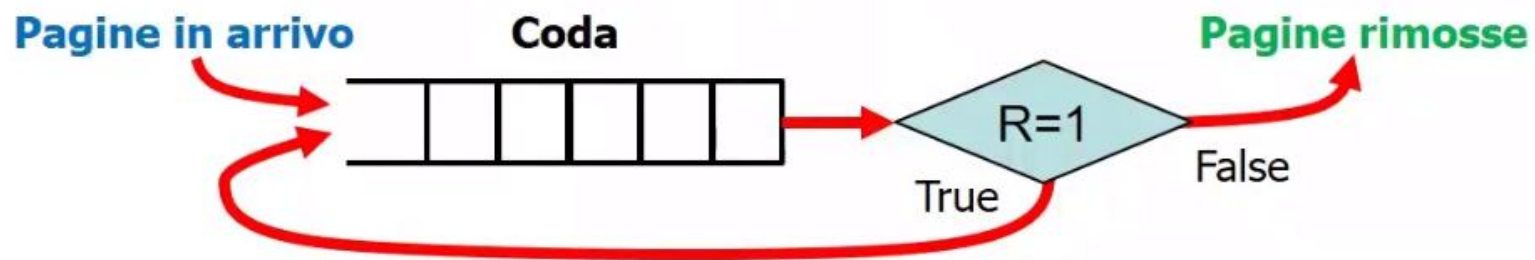


- Quando accade un errore di pagina, la pagina in testa è rimossa e la nuova pagina è aggiunta alla coda della lista.
- Poiché la gestione FIFO non tiene conto dell'utilizzo della pagina ma solo dell'ordine di caricamento in memoria, l'algoritmo è raramente utilizzato.

Seconda Chance

- Una semplice modifica al FIFO che evita il problema di buttare via una pagina usata frequentemente è di ispezionare il **bit R** della pagina che è stata caricata da più tempo (in testa alla coda), se è:
 - **0**, significa che **non è stata utilizzata** recentemente quindi si può sostituire.
 - **1**, significa che è stata utilizzata recentemente quindi **non** si può sostituire. Il bit R viene azzerato e alla pagina viene data una «**seconda chance**» reinserendola in coda alla lista.

La ricerca prosegue con la successiva pagina in testa.



Clock

- L'algoritmo **seconda chance** è un buon metodo ma è inefficiente perché le pagine sono sempre in movimento nella coda.
- Un approccio migliore è quello di mantenere tutti i frame di pagina in una lista circolare nella forma di un orologio da cui il nome dell'algoritmo.
- La lancetta indica la pagina più vecchia.
- Quando si verifica un **page fault**, si controlla il **bit R** della pagina sotto la lancetta, se:
 - **0**, la pagina è sostituita con la nuova pagina e la lancetta si muove sulla successiva posizione.
 - **1**, esso viene azzerato e la lancetta si muove alla successiva posizione fino non trova una pagina con il **bit R = 0**.



Least Recently Used (LRU)

- Una buona approssimazione dell'algoritmo ottimo si basa sull'osservazione che le pagine usate più frequentemente dalle ultime istruzioni lo saranno anche nelle successive.
- Al contrario, le pagine che non sono state utilizzate da molto tempo probabilmente resteranno inutilizzate.
- Questa idea suggerisce un metodo: quando si verifica un errore di pagina, meglio buttare via la pagina che è rimasta inutilizzata per più tempo.
- **LRU** è teoricamente fattibile ma non è economico.

Least Recently Used (LRU)

- Per realizzare l'algoritmo **LRU** viene utilizzato una lista collegata di tutte le pagine in memoria, con quella utilizzata più recentemente in testa e quella utilizzata meno recentemente in coda.
- La complicazione deriva dal fatto che l'elenco deve essere aggiornato ad ogni riferimento alla memoria.
- Trovare una pagina nella lista e cambiargli di posizione richiede tempo.
- Sono possibili due realizzazioni hardware che utilizzano un:
 - contatore a 64 bit.
 - matrice binaria.

LRU con contatore a 64 bit

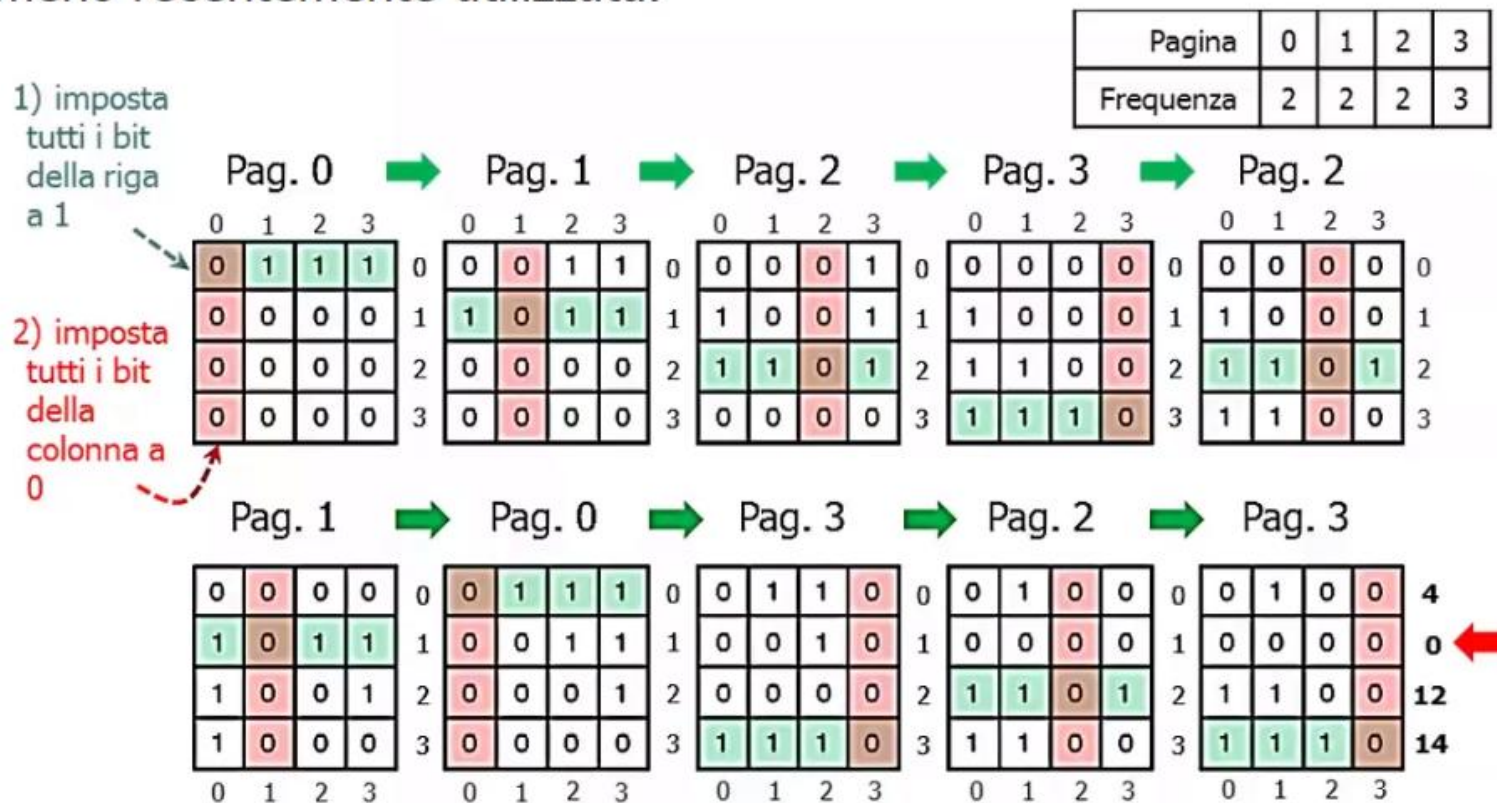
- Il contatore viene incrementato automaticamente dopo ogni istruzione.
- Il contatore è memorizzato in ogni riga nella tabella delle pagine.
- Ad ogni riferimento in memoria, il contatore di ciascuna pagina referenziata è aggiornato nella tabella delle pagine.
- Quando si verifica un **page fault**, il sistema operativo cerca nella tabella delle pagine quella con il minimo valore del contatore che corrisponde alla pagina utilizzata meno recentemente.

LRU con matrice binaria

- Per una macchina con **n** pagine fisiche (frame), l'hardware LRU può mantenere una matrice di **n x n** bit (inizialmente tutti a zero).
- Ogni volta che la pagina **k** viene referenziata, l'hardware prima imposta tutti i bit della riga **k** a 1, poi imposta tutti i bit della colonna **k** a 0.
- La riga il cui valore binario è più basso indica il frame meno recentemente utilizzato.

LRU con matrice binaria

- Per esempio si consideri un sistema con 4 frame, quest'ultimi sono utilizzati nell'ordine: 0, 1, 2, 3, 2, 1, 0, 3, 2 e 3. La pagina 1 è quella meno recentemente utilizzata.



Not Frequently Used (NFU)

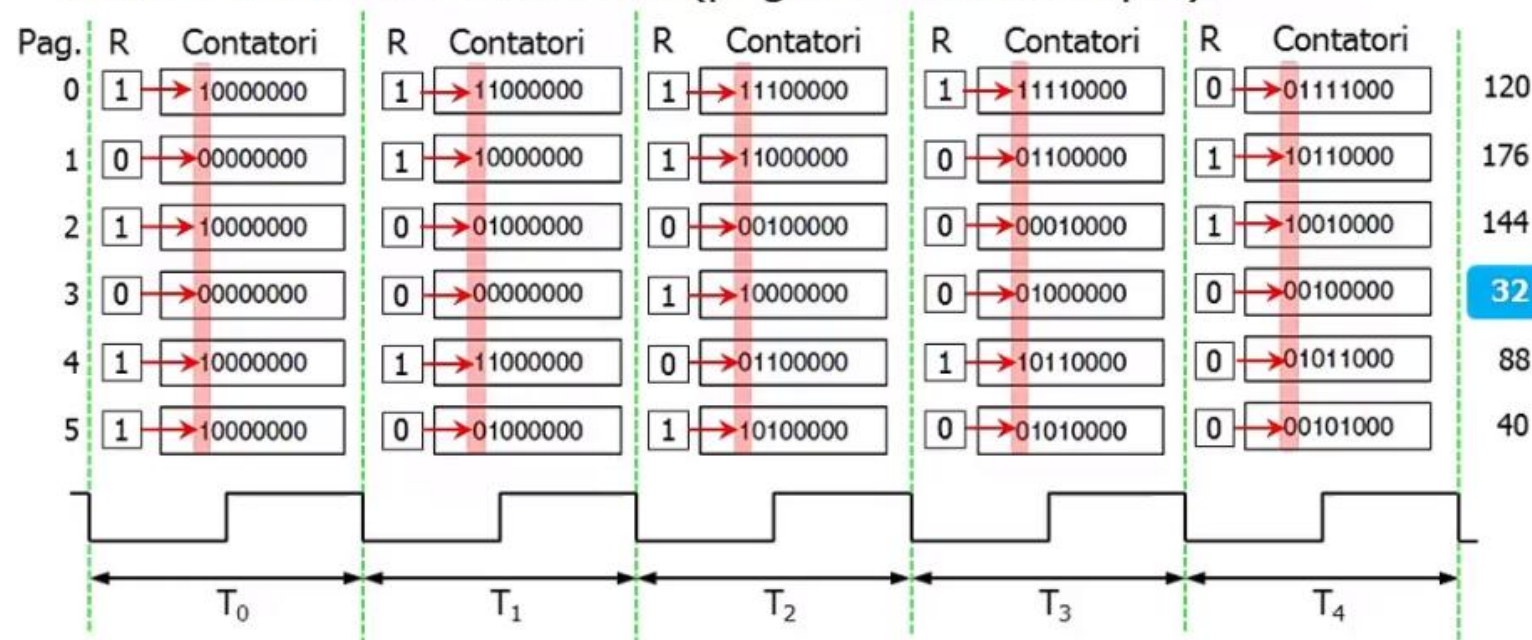
- Anche se entrambi gli algoritmi **LRU** sono fattibili, non esistono hardware che li realizzino.
- Un altro algoritmo che può essere implementato via software è l'**NFU (Not Frequently Used)**:
- Ogni pagina ha associato un contatore, inizialmente pari a zero.
- Ad ogni interruzione di clock, il sistema operativo esegue la scansione di tutte le pagine in memoria.
- Per ogni pagina, viene sommati il **bit R** al contatore.
- Così i contatori mantengono il numero di riferimenti fatti a ciascuna pagina.
- Quando si verifica un errore di pagina, la pagina con il contatore più basso è scelta per la sostituzione.

Problemi del NFU

- Il principale problema del **NFU** è che non dimentica nulla.
- Ad esempio, in un compilatore a più passaggi, le pagine utilizzate più frequentemente durante il primo passaggio possono ancora avere un conteggio alto anche quando nei passaggi successivi saranno **inutili**!
 - spesso questa circostanza si verifica quando il primo passaggio ha un tempo di esecuzione maggiore degli altri passaggi.
- Di conseguenza, il sistema operativo rimuoverà pagine utili invece che quelle non più in uso.

Aging

- Consideriamo una memoria con 6 pagine. Inizialmente i contatori delle 6 pagine sono tutti azzerati. Ad ogni ciclo di clock si esegue uno shift a destra e si aggiunge il bit R nella posizione più significativa.
- Quando si verifica un **page fault**, viene rimossa la pagina che ha il minore valore del contatore (pagina 3 nell'esempio).



Confronto tra Aging e LRU

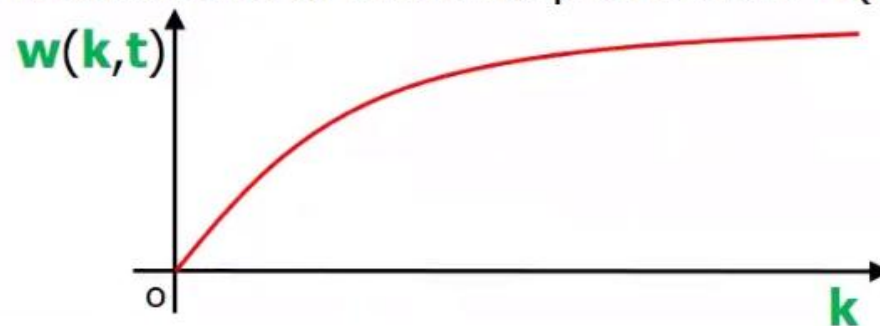
- Utilizzando un solo bit per intervallo di clock, sarà possibile registrare al massimo un riferimento per pagina anche se ce ne fossero di più nel medesimo intervallo di tempo.
- Nell'algoritmo di **Aging** i contatori sono codificati con un numero finito di bit (8 nell'esempio), questo limita l'orizzonte di analisi a quel numero di cicli di clock.
- In genere, 8 bit forniscono un arco temporale sufficiente (160 ms) considerando un periodo di circa 20 ms.

Working Set

- La gestione delle pagine più semplice si ha quando i processi sono avviati senza pagine in memoria: la prima istruzione genera un **page fault** e causa una trap per il sistema operativo che deve caricare la pagina che lo contiene.
- Questa strategia si chiama **demand paging** perché le pagine vengono caricate su richiesta e non in anticipo.
- Nella maggior parte dei casi vale un **principio località** dei riferimenti, il che significa che durante l'esecuzione un processo utilizza una frazione piccola delle sue pagine: il **Working Set**.
- Se l'intero **Working Set** è in memoria il processo non causerà errori di pagina.
- Ad ogni **page fault** la CPU perde circa **10 ms** per recuperare la pagina dal disco, mentre l'esecuzione di una istruzione richiede pochi **ns**!
- Se un processo causa un errore di pagina ogni poche istruzioni si dice **processo thrashing**.

Working Set

- Molti sistemi di paging cercano di assicurarsi che il **Working Set** del processo sia in memoria prima di eseguirlo (**Working Set Model**). È stato progettato per ridurre notevolmente il tasso di errore di pagina.
- Il caricamento delle pagine prima di eseguire il corrispondente processo è chiamato **prepaginazione**.
- Si noti il cambiamento del **Working Set** nel corso del tempo.
- Molti programmi non referenziano lo spazio di indirizzamento in modo uniforme, ma l'insieme dei riferimenti tendono a raggrupparsi in un numero ridotto di pagine.
- In ogni istante di tempo (**t**) esiste un insieme di tutte le pagine utilizzato dai **k** riferimenti di memoria più recenti: **w(k,t)**.



Working Set

- $w(k,t)$ è una funzione monotona non decrescente in k che tende al numero delle pagine del programma.
- La causa del comportamento asintotico in k è perché con k piccolo il $w(k,t)$ cresce rapidamente, mentre al crescere di k il $w(k,t)$ cresce più lentamente perché approssima il limite della funzione.
- Esiste un intervallo di valori di k in cui $w(k,t)$ è quasi uniforme.
- La **pre-paginazione** consiste nel caricamento di queste pagine prima che il processo sia avviato.
- L'algoritmo utilizza un registro a scorrimento con k posizioni in cui finisce ogni riferimento delle pagine in memoria.
 - Ad ogni **page fault** è necessario ricaricare il registro e riordinarlo (questo fatto ne rende difficile l'utilizzo).

Approssimazione del Working Set

- Invece di considerare i **k** riferimenti a ritroso, si può utilizzare l'approssimazione di considerare l'insieme di pagine utilizzate negli ultimi **τ** ms del tempo di esecuzione.
- Con questa definizione è molto più facile da lavorare.
- Ogni processo misura il suo tempo di esecuzione (**tempo virtuale attuale**).
- Con questa approssimazione, il **Working Set** di un processo diventa l'insieme delle pagine referenziate negli ultimi **τ** secondi del suo tempo virtuale.

Approssimazione del Working Set

- Nella tabella delle pagine la riga contiene:
 - Il tempo (**h**) da quando la pagina è stata utilizzata.
 - Il **bit R** se è stata referenziata nell'ultimo ciclo di clock.
 - il **bit M** se la pagina è stata modificata.
- Quando si verifica un **page fault**, si cerca nella tabella delle pagine la pagina da rimuovere, se il **bit R** è:
 - **1**, il **tempo virtuale attuale** è scritto nella riga della pagina ed essa non è candidata per la rimozione.
 - **0**, si calcola l'età della pagina:


$$\text{età} = \text{tempo attuale} - \text{tempo di ultimo utilizzo}$$

se **età** > τ , la pagina non è più nel **Working Set** e può essere sostituita con una nuova.

Approssimazione del Working Set

- se età $\leq \tau$, la pagina è ancora nel **Working Set** è temporaneamente risparmiata, ma la pagina più anziana (max età) è contrassegnata.
- Se dopo aver ispezionato l'intera tabella delle pagine non viene trovata alcuna pagina candidata (tutte le pagine sono nel **Working Set**), è rimossa quella più anziana già contrassegnata (**bit R=0**).
- Nel peggiore dei casi che tutte le pagine siano state referenziate (**bit R=1**) durante l'ultimo ciclo di clock, è rimossa una a caso tra quelle pulite (**bit M=0**).

WSClock

- Gli algoritmi che si basano sul **Working Set** sono lenti, perché ad ogni **page fault** occorre scorrere tutta la tabella delle pagine al fine di trovare la pagina da rimuovere.
- Un algoritmo migliore è basato su una variante dell'algoritmo clock che si chiama **WSClock**.
- Grazie alla sua semplicità di realizzazione ed alle sue prestazioni, è praticamente quello più utilizzato. 
- La struttura dei dati è una lista circolare di frame di pagina, come nell'algoritmo clock (inizialmente questo elenco è vuoto).
- Quando la prima pagina viene caricata, viene aggiornato l'elenco.
- Ogni riga della tabella contiene il tempo di utilizzo (**τ**), il **bit R** e il **bit M**.

WSClock

- Come accade nell'algoritmo clock ad ogni **page fault** la lancetta punta alla pagina da esaminare.
- Se il **bit R = 1** significa che la pagina è stata appena utilizzata (non è una buona candidata alla rimozione). Il **bit R** è impostato a 0.
- La lancetta passa alla pagina successiva nella sua corsa alla ricerca di una pagina da rimuovere.
- Se (**bit R = 0**) AND (**età > τ**) AND (**bit M = 0**), non è nel **Working Set** ed esiste una copia valida sul disco. La pagina può essere sovrascritta dalla nuova.
- Se (**bit R = 0**) AND (**età > τ**) AND (**bit M = 1**), non è nel **Working Set** e non esiste una copia valida sul disco. Per evitare un cambiamento di processo, viene schedulata la scrittura sul disco e la lancetta passa alla pagina successiva.

WSClock

- *Cosa succede se la lancetta torna al punto di partenza senza aver trovato la pagina da rimuovere?*
- Esistono due possibilità:
 - È stata programmata una (o più) una scrittura/e.
 - Non sono presenti richieste di scritture pendenti.
- Nel primo caso, la lancetta ricomincia il ciclo. Dal momento che una o più operazioni di scrittura sono state programmate, al termine dell'attività di scrittura una pagina sarà contrassegnata come pulita (**bit M=0**). La prima pagina pulita incontrata sarà rimossa.
- Nel secondo caso, tutte le pagine sono nel **Working Set** (non ci sono pagine pulite), allora è scelto come vittima la pagina corrente e viene scritta sul disco.




Quadro riassuntivo degli algoritmi di sostituzione delle pagine

- Non c'è modo di prevedere i riferimenti futuri ad una pagina, l'algoritmo ottimo è utile solo come un punto di riferimento.
- L'algoritmo **LRU** divide le pagine in 4 classi a seconda dei **bit R** e **bit M**: è scelta una pagina a caso nella classe 0. **LRU** è facile da realizzare, è molto grezzo e ne esistono di migliori.
- **FIFO** mantiene l'ordine di caricamento in memoria delle pagine. La rimozione della pagina più vecchia è semplice, ma la pagina potrebbe essere ancora in uso: non è utilizzabile.
- **Seconda chance**: è una versione applicabile del **FIFO** che verifica se la pagina è in uso prima di rimuoverla. Questa modifica migliora le prestazioni.
- **Clock** è semplicemente una diversa realizzazione del precedente. Ha le stesse performance, ma richiede meno tempo per l'esecuzione dell'algoritmo.

Quadro riassuntivo degli algoritmi di sostituzione delle pagine

- **LRU** è un eccellente algoritmo, ma non può essere attuato senza un hardware dedicato.
- **NFU** è un tentativo grezzo di approssimare **LRU**.
- **Aging** è una buona approssimazione dell'**NFU** e può essere efficacemente realizzato.
- L'algoritmo del **Working Set** dà prestazioni ragionevoli, ma richiede lunghi tempi di elaborazione.
- **WSClock** è una variante del working set che dà buoni risultati ed è anche efficiente.

Quadro riassuntivo degli algoritmi di sostituzione delle pagine

Algoritmo	Caratteristiche
<i>Ottimo</i>	<i>Non realizzabile, ma utile come riferimento</i>
NRU (Not Recently Used)	Approssimazione molto semplice dell' LRU
FIFO (First-In, First-Out)	Potrebbe buttare via pagine importanti, non utile così com'è
Seconda chance	Miglioramento del FIFO
Clock	Altra versione del Seconda chance, poco migliore nei tempi
LRU (Least Recently Used)	Eccellente, ma difficile da realizzare con precisione
NFU (Not Frequently Used)	Approssimazione rozza dell' LRU
Aging	Efficiente, approssima bene l' LRU
Working Set	Troppo costo da realizzare
WSClock	Efficiente e valido 

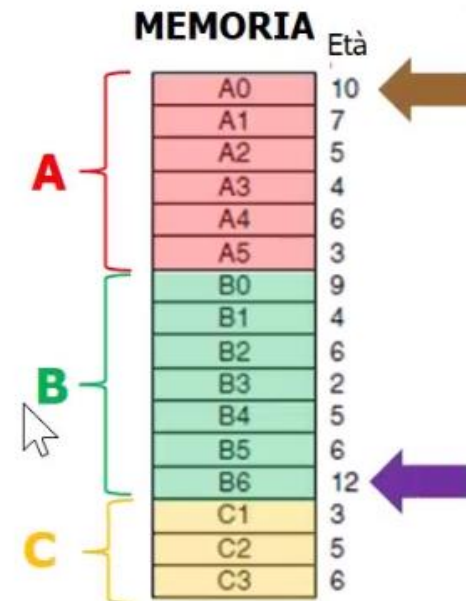
PROBLEMATICHE DI PROGETTAZIONE DEI SISTEMI DI PAGING

- I progettisti di sistemi operativo devono considerare altri aspetti al fine di ottenere buone prestazioni dai sistemi di paging:
 - Politiche di allocazione globale e locale.
 - Controllo del carico.
 - Dimensioni della pagina.
 - Dati e istruzioni su spazi separati.
 - Pagine condivise.



Confronto tra politiche di allocazione

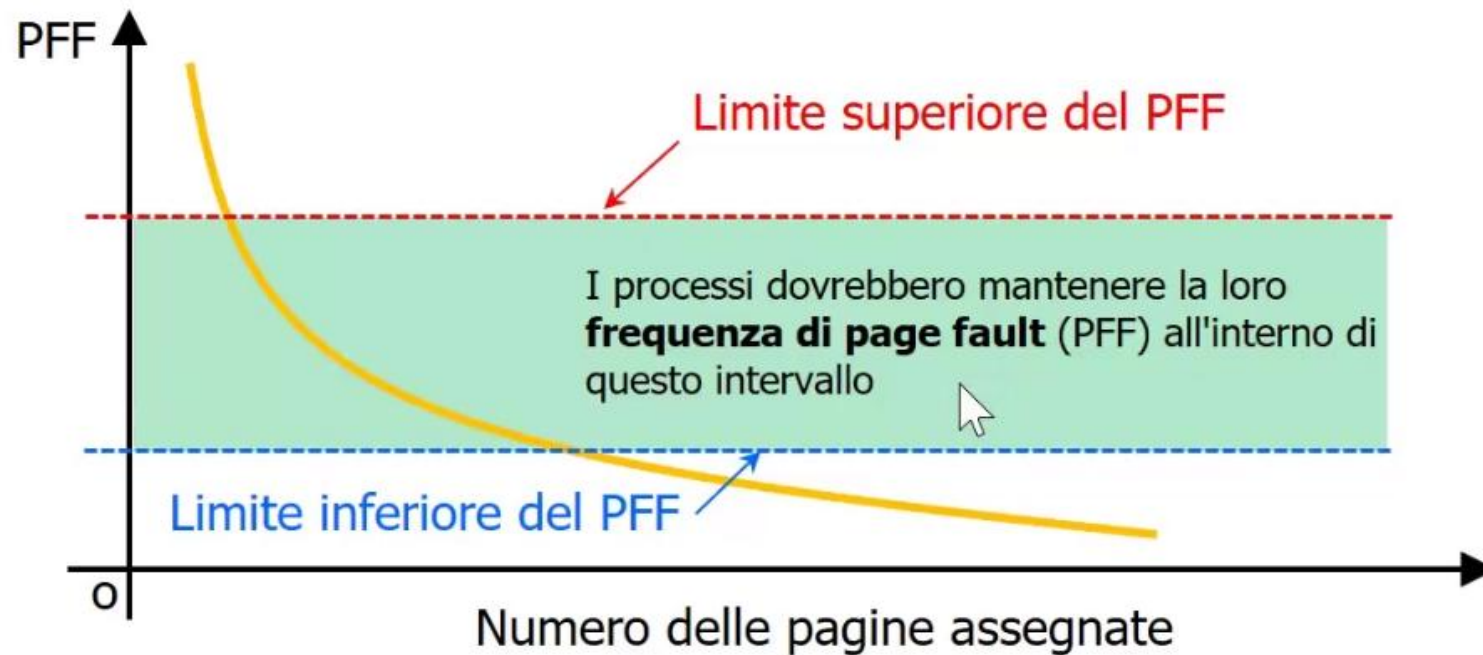
- Il problema principale associato a questa scelta è la quantità di memoria che dovrebbe essere ripartito tra processi eseguibili.
- Supponiamo che ci siano tre processi in esecuzione: **A**, **B** e **C**.
- Se **A** ottiene un errore di pagina, l'algoritmo di sostituzione delle pagine deve cercare la pagina meno usata in:
 - 1) Tutte le pagine in memoria
 - 2) Solo quelle di A



Politiche di allocazione globali

- Gli algoritmi globali funzionano meglio quando la dimensione del **Working Set** può variare nel corso della vita di un processo:
 - Se si utilizza un algoritmo locale e il **Working Set** cresce, si ottiene del thrashing, anche in presenza di molti frame di pagina liberi.
 - Se il **Working Set** si riduce, gli algoritmi locali sprecano memoria.
- Un metodo è di allocare dinamicamente le pagine di un processo (partendo da un valore iniziale proporzionale alla sua dimensione) facendo in modo che la frequenza dei **page fault** (**PFF**, Page Fault Frequency) al secondo sia contenuta in un certo intervallo.

Frequenza di Page Fault (PFF)



Confronto tra algoritmi

- **FIFO**, **LRU** e tutte le approssimazioni dell'**LRU** possono lavorare sia con politiche di sostituzione delle pagine locali sia globali.
- **Working Set** e **WSClock** possono lavorare **esclusivamente** con politiche di sostituzione delle pagine locali.

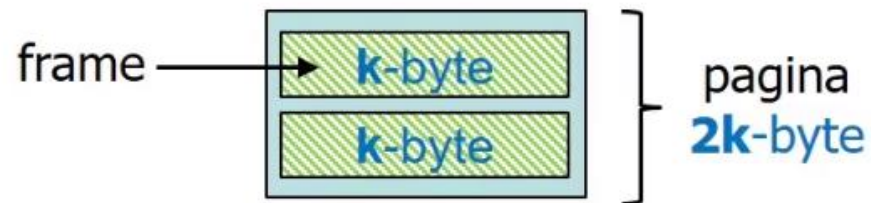


Controllo del carico

- Anche se si utilizza il migliore algoritmo di sostituzione delle pagine e la migliore assegnazione delle pagine, in sistema può andare in **sovraccarico**.
- Ad esempio, se il **Working Set** di tutti i processi superano la capacità della memoria si verifica del **thrashing**.
- Una spia può essere la necessità dei processi di più memoria mentre nessuno ne richiede di meno.
 - l'unica soluzione reale è di mandare dei processi su disco (swapping).
- Nel decidere quale processo mantenere in memoria e quale mandare su disco occorre considerare:
 - Il grado di multiprogrammazione.
 - La dimensione del processo.
 - La frequenza di paginazione (PFF).
 - Il tipo di processo (I/O o CPU bound)

Dimensione delle pagine

- La dimensione della pagina è un parametro definito dal sistema operativo.
- Anche se l'hardware è stato progettato per lavorare con frame di **k**-byte, il sistema operativo può scegliere di lavorare con multipli di **k**.

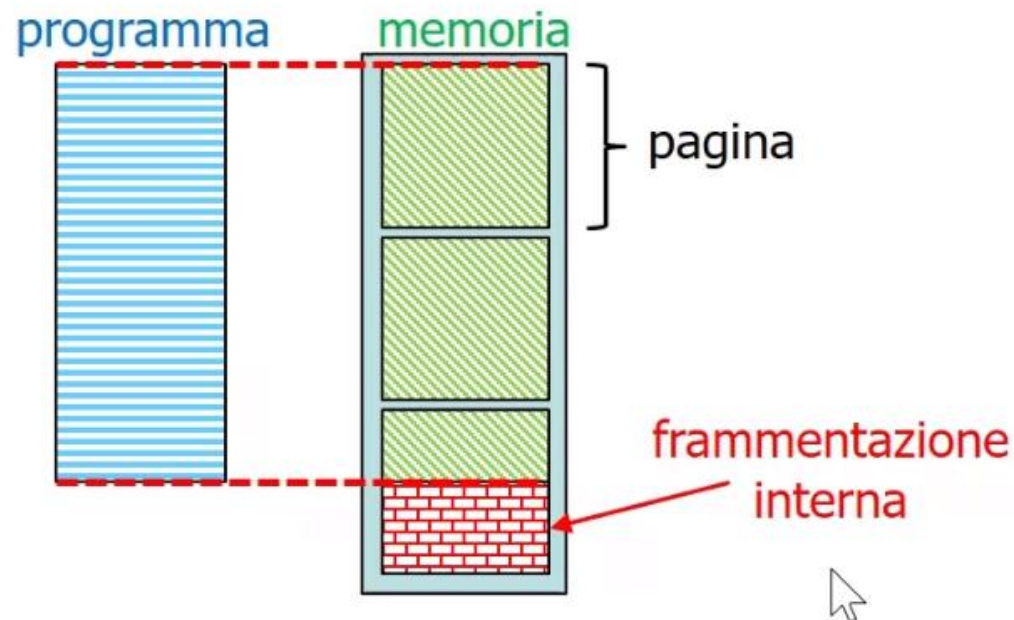


- La scelta della dimensione della pagina è un fattore critico poiché dipende da vari aspetti tra loro correlati.



Dimensione delle pagine

- La dimensione dei programmi, dei dati o del segmento di stack non sarà mai un multiplo esatto della dimensione della pagina e quindi ci sarà sempre dello spreco di memoria (**frammentazione interna**).



Dimensione delle pagine

- Per minimizzare lo spazio (inutilizzato) all'interno di pagine, esse dovrebbero essere più **piccole** possibili.
- Pagine **piccole** permette maggiore efficienza per l'esecuzione dei processi: si caricano in memoria solo le componenti che servono.
- Pagine **piccole** richiedono tabelle delle pagine enormi.
- Poiché i trasferimenti su disco sono di una pagina per volta e la maggior parte del tempo speso dal disco è per posizionare la testina (tempo di latenza e seek) è meglio avere pagine **grandi**:
 - caricare 64 pagine da 512 byte (32kB) richiede $64 \times 10 \text{ ms} = 640 \text{ ms}$,
 - caricare 4 pagine da 8 kB (32kB) richiede $4 \times 12 \text{ ms} = 48 \text{ ms}$.



Dimensione delle pagine

- In alcune macchine, la tabella delle pagine deve essere caricata nei registri hardware ogni volta che la CPU passa da un processo all'altro.
 - In questo caso è meglio avere pagine **grandi** poiché si riduce il tempo necessario per caricare i registri.
- Gli attuali computer utilizzano pagine da **1, 4 o 8 kB**.
- Con il crescere delle dimensioni delle memorie anche la dimensione delle pagine tende a crescere anche se in modo non lineare.
- Quadruplicando la dimensione della RAM raramente si raddoppia la dimensioni della pagina.



Spazio dati e istruzioni distinti

- la maggior parte dei computer ha un unico spazio di indirizzamento che contiene sia i programmi sia i dati.
- Se questo spazio di indirizzi è abbastanza grande, tutto funziona bene.
- Quando è troppo piccolo costringe i programmi ad adattarsi nell'intero spazio di indirizzi.



Spazio dati e istruzioni distinti

- Una soluzione è quella di utilizzare spazi di indirizzo separati per le istruzioni (testo programma) e per i dati (utilizzati per la prima volta dal PDP-11): **I-space** e **D-space**.
- In questo modo entrambi gli spazi possono essere paginati l'uno indipendentemente dall'altro.



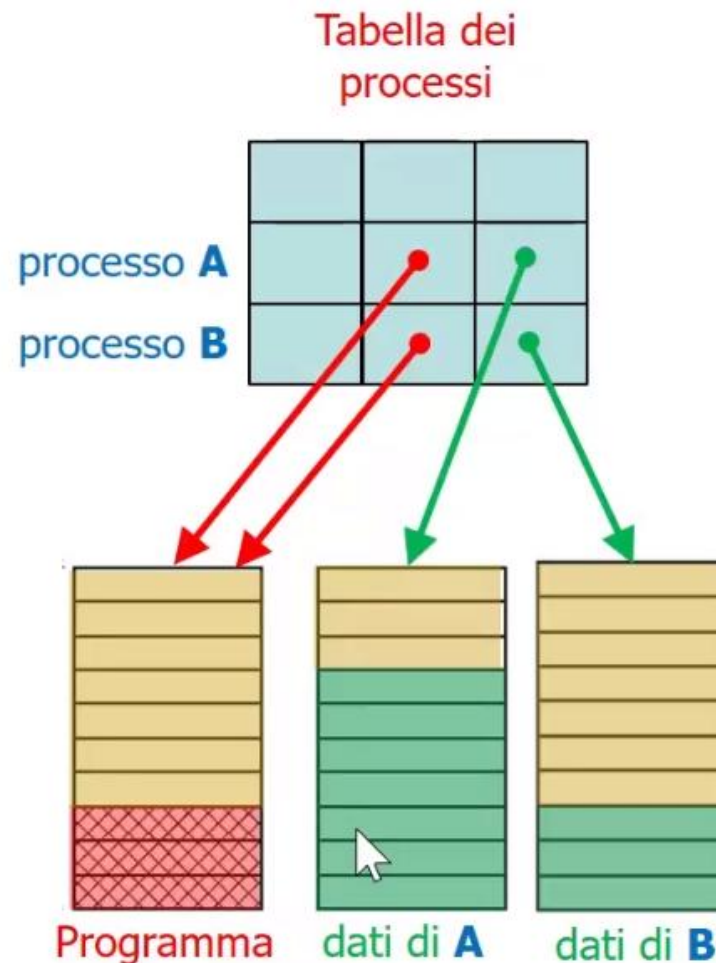
Pagine condivise

- Un altro problema di progettazione è la condivisione.
- In un grande sistema multiprogrammato, è normale che più utenti eseguano lo stesso programma contemporaneamente.
- Per evitare di avere due copie della stessa pagina in memoria allo stesso istante di tempo, si possono **condividere le pagine**.
- Il problema è che non tutte le pagine sono condivisibili.
- In particolare, le pagine che sono di sola lettura, come il testo del programma, possono essere condivisi più agevolmente, ma pagine di dati devono essere gestite con attenzione.




Pagine condivise

- Supponiamo che i processi **A** e **B**, che condividono pagine di codice, stiano entrambi eseguendo un editor.
- Se lo scheduler decide di rimuovere **A** dalla memoria, eliminando tutte le sue pagine e riempiendo i frame vuoti con qualche altro programma provocherà un gran numero di **page fault** (processo **B**) che riporteranno di nuovo le pagine in memoria.



Pagine di dati condivisi

- La condivisione dei dati è più difficile di quella del codice, anche se non è impossibile.
- In UNIX dopo una **fork()**, il processo genitore e quello figlio sono tenuti a condividere sia il testo del programma e sia dati.
- In un sistema di paging, ognuno di questi processi ha una sua riga nella tabella delle pagine ed entrambe puntano allo stesso insieme di pagine (non avviene nessuna copia e tutte le pagine dati sono contrassegnate come **read-only**).
- Se i processi leggono i dati tutto funziona bene.
 - Non appena un processo tenta di scrivere, la violazione al **read-only** provoca una **trap** al sistema operativo e viene fatta la copia della pagina che ha generato l'errore.
 - Le due copie ora hanno vita indipendente. 
- Le performance sono migliorate perché le copie sono ridotte al minimo indispensabile (**copy on write**).

Conclusioni

- È stato analizzato inizialmente l'algoritmo di sostituzione delle pagine ottimale che è un algoritmo teorico ed utilizzato come riferimento.
- Sono stati analizzati i diversi algoritmi di sostituzione delle pagine (**NRU**, **FIFO**, **Seconda chance**, **Clock**, **LRU**, **Working Set**, **WSClock**) e affrontato le caratteristiche computazionali di ciascuno di essi.
- Intodotto le problematiche esistenti nella progettazione dei sistemi di paging.