

# Gestione della memoria

- La memoria principale(**RAM**) è una risorsa importante che deve essere gestita con attenzione.
- Il concetto di **gerarchia di memoria**:
  - i computer attuali possono avere pochi MB di memoria **cache** volatile, costosa e molto veloce, qualche GB di **memoria principale**, di media velocità e prezzo, e qualche TB di **storage** su disco non volatile, economico e lento.
  - La parte del sistema operativo che gestisce (in parte) la gerarchia della memoria è chiamata **gestore della memoria**.

## Nessuna astrazione della memoria

- Il modo più semplice per gestire la memoria è senza forme di astrazione: i processi la «vedono» direttamente così com'è costituita (numero di celle e dimensione del contenuto).
- I primi mainframe (prima del 1960), minicomputer (prima del 1970) e PC (prima del 1980) non avevano alcuna astrazione di memoria.
- Per eseguire un programma è necessario caricarlo nella memoria principale.
- Più programmi possono stare in memoria ma è necessario che non si sovrappongano!

## Nessuna astrazione della memoria

- Anche senza astrazione di memoria il modello fisico può essere organizzato in vari modi:
  - SO in fondo alla memoria nella RAM (mainframe e minicomputer).
  - SO in testa alla memoria nella ROM (palmari e sistemi embedded).
  - SO in fondo alla memoria nella RAM, driver delle periferiche in testa alla memoria nella ROM (PC con MS-DOS).
- Il primo e il terzo modello hanno lo svantaggio che un errore nel programma utente può eliminare il sistema operativo!

## Esecuzione di più programmi

- Anche senza l'astrazione di memoria, è possibile fornire all'utente l'idea di avere in esecuzione più programmi «contemporaneamente».
- Il sistema operativo salva l'intero contenuto della memoria in un file sul disco, carica ed esegue il programma successivo (*swapping*).
- Con l'aggiunta di alcuni hardware speciali, è possibile eseguire più programmi contemporaneamente, anche senza swapping.
- La mancanza di una astrazione memoria è ancora comune nei dispositivi quali radio, lavatrici e forni a microonde: il software indirizza la memoria in modo assoluto e funziona solo perché i programmi sono noti in anticipo e gli utenti non sono (ancora) liberi di eseguire il proprio software sul proprio tostapane.

## Lo spazio di indirizzamento

Esporre la memoria fisica ai processi può presentare delle criticità:

- ① se i programmi utente possono indirizzare ogni byte della memoria, possono facilmente sovrascrivere il sistema operativo.
- ② è complicato avere un'organizzazione rigida della memoria che consenta a più programmi di essere in esecuzione contemporaneamente.

# La nozione di spazio degli indirizzi

- Per consentire a più applicazioni di essere in memoria allo stesso tempo senza interferire devono essere risolti due problemi: **protezione e riposizionamento**.
- Una soluzione iniziale fu adottata dall'IBM 360 che etichettava pezzi di memoria con una chiave di protezione e, poi, confrontava la chiave del processo di esecuzione con quella di ogni parola di memoria:
  - questo approccio non risolveva il secondo problema del posizionamento dei programmi, anche se può essere risolto spostandoli al momento del caricamento con aggravio dei tempi e della complessità.

## Registro base e limite

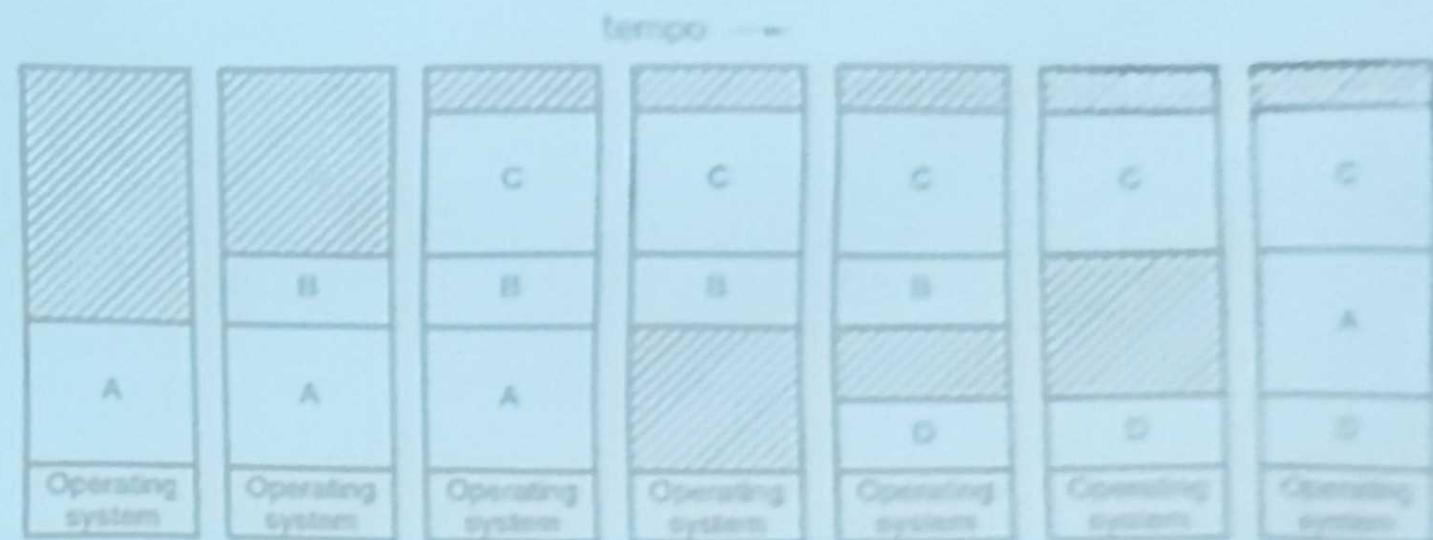
- Questa soluzione utilizza una versione particolarmente semplice di rilocazione dinamica: mappa ogni spazio indirizzi del processo su una diversa parte della memoria fisica.
- La soluzione classica è quella di dotare ogni CPU con due registri hardware speciali: **registro base** e **registro limite**.
- I programmi sono caricati in locazioni di memoria consecutive.
- Quando viene eseguito un processo, il registro base viene caricato con l'indirizzo fisico di memoria dove inizia il programma e il registro limite con la lunghezza del programma.
- Ogni volta che un processo accede alla memoria l'hardware della CPU aggiunge automaticamente il valore base all'indirizzo verificando se l'indirizzo calcolato non eccede il limite di memoria riservato al processo (registro limite).
- In molte implementazioni, i registri base e limite sono protetti (solo il sistema operativo li può modificare).
- Uno svantaggio della rilocazione per mezzo dei registri base e limite è la necessità di eseguire un'addizione e un confronto ad ogni riferimento di memoria: il confronto può essere fatto in fretta, ma le addizioni sono lente se non si utilizzano circuiti sommatori speciali.

# Swapping

- Se la memoria fisica del computer è abbastanza grande da contenere tutti i processi, gli schemi descritti sono validi.
- Normalmente, la quantità totale di RAM necessaria a tutti i processi è spesso molto superiore a quella che può essere contenuta nella memoria.
- Per gestire la memoria sono possibili due approcci generali:
  - Lo **swapping**: consiste nell'inserire ogni processo, farlo girare per un po' e poi rimetterlo sul disco. I processi inattivi sono per lo più memorizzati sul disco, quindi non occupano memoria quando non sono in esecuzione.
  - La **memoria virtuale**: permette di eseguire i programmi anche quando si trovano solo parzialmente nella memoria principale.

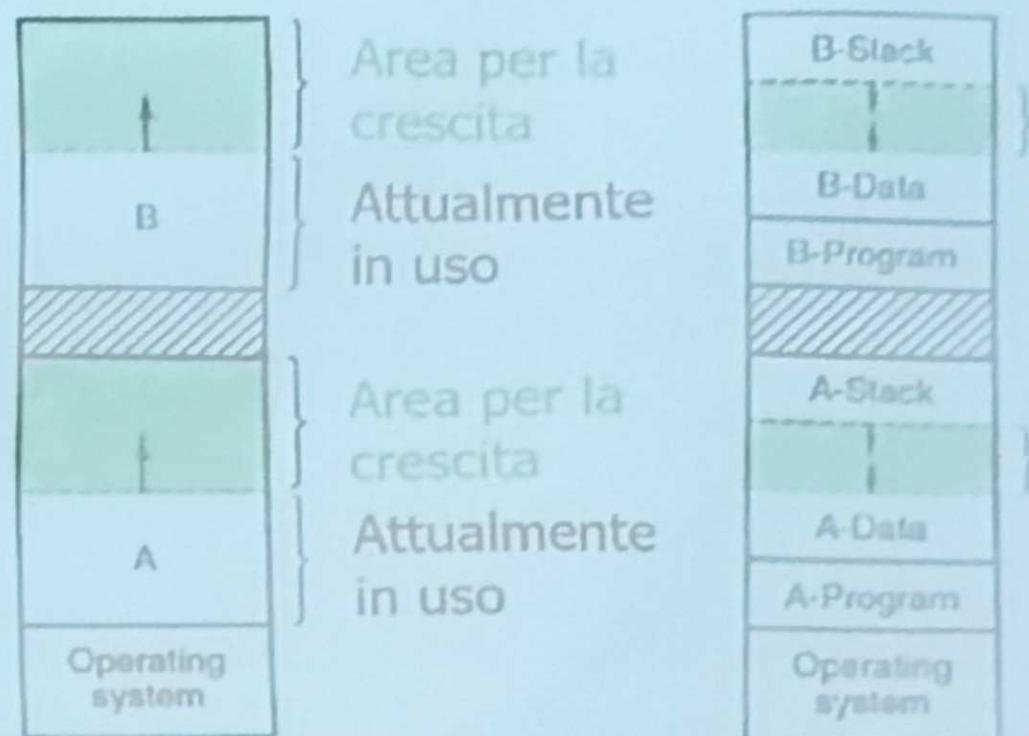
# Swapping

- Inizialmente il processo A è in memoria, sono poi creati (o caricati) i processi B e C. Dopo A è salvato sul disco.
- Viene creato D e B termina. Finalmente viene ricaricato A ma in una posizione diversa: i suoi indirizzi devono essere rilocati dal software o dall'hardware durante l'esecuzione del programma.



# Swapping

- Lo swapping crea dei buchi nella memoria che è possibile compattare solo spostando i processi, raramente è eseguita poiché si tratta di una operazione onerosa per la CPU.
- Se i processi hanno una dimensione fissa, l'allocazione è semplice: il sistema operativo alloca esattamente ciò che è necessario.
- Normalmente i processi crescono di dimensione, attraverso l'allocazione dinamica della memoria heap o lo stack, quindi è buona norma riservare uno spazio in più riservato alla crescita.

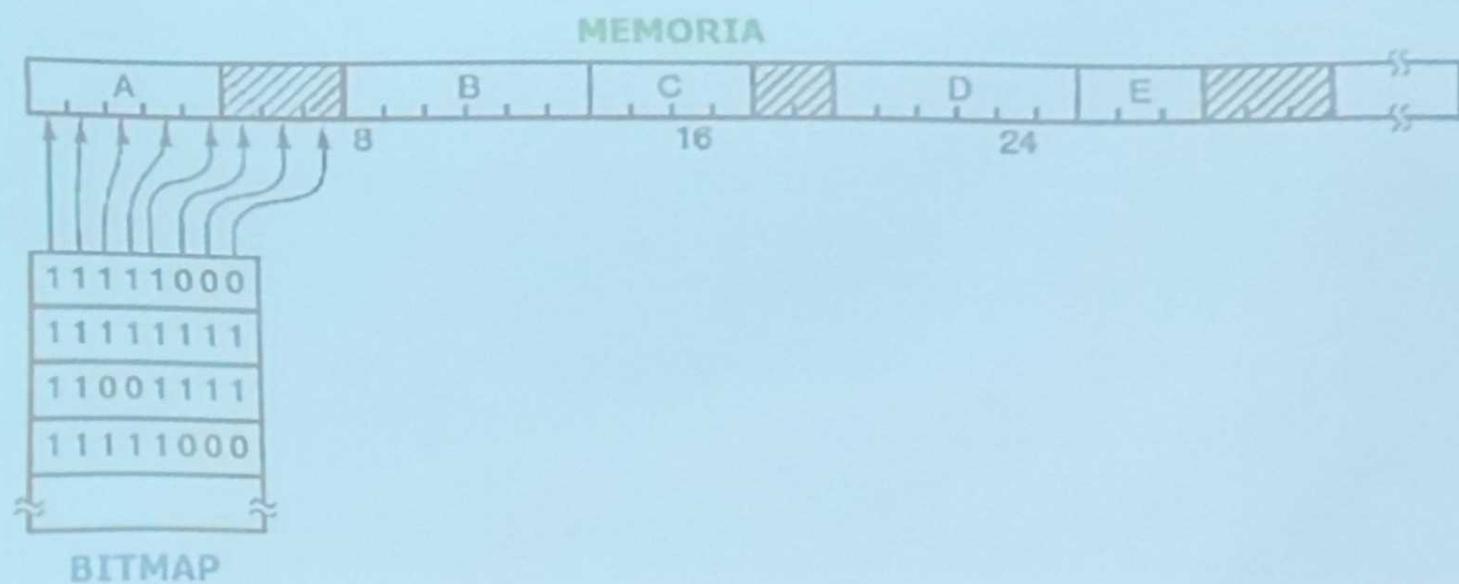


# Gestione della memoria libera

- Quando la memoria viene assegnata dinamicamente, il sistema operativo deve gestirla.
- Ci sono due modi per tenere traccia dell'utilizzo della memoria:
  - Bitmap
  - Liste collegate di allocazione

# Gestione della memoria con bitmap

- Con una bitmap, la memoria è divisa in unità di allocazione.
- La dimensione di una unità di allocazione può essere di qualche parola fino a diversi KB.
- La bitmap ha bit quante le unità di allocazione della memoria:
  - se il bit è 0 l'unità è libera,
  - se 1 è occupata.

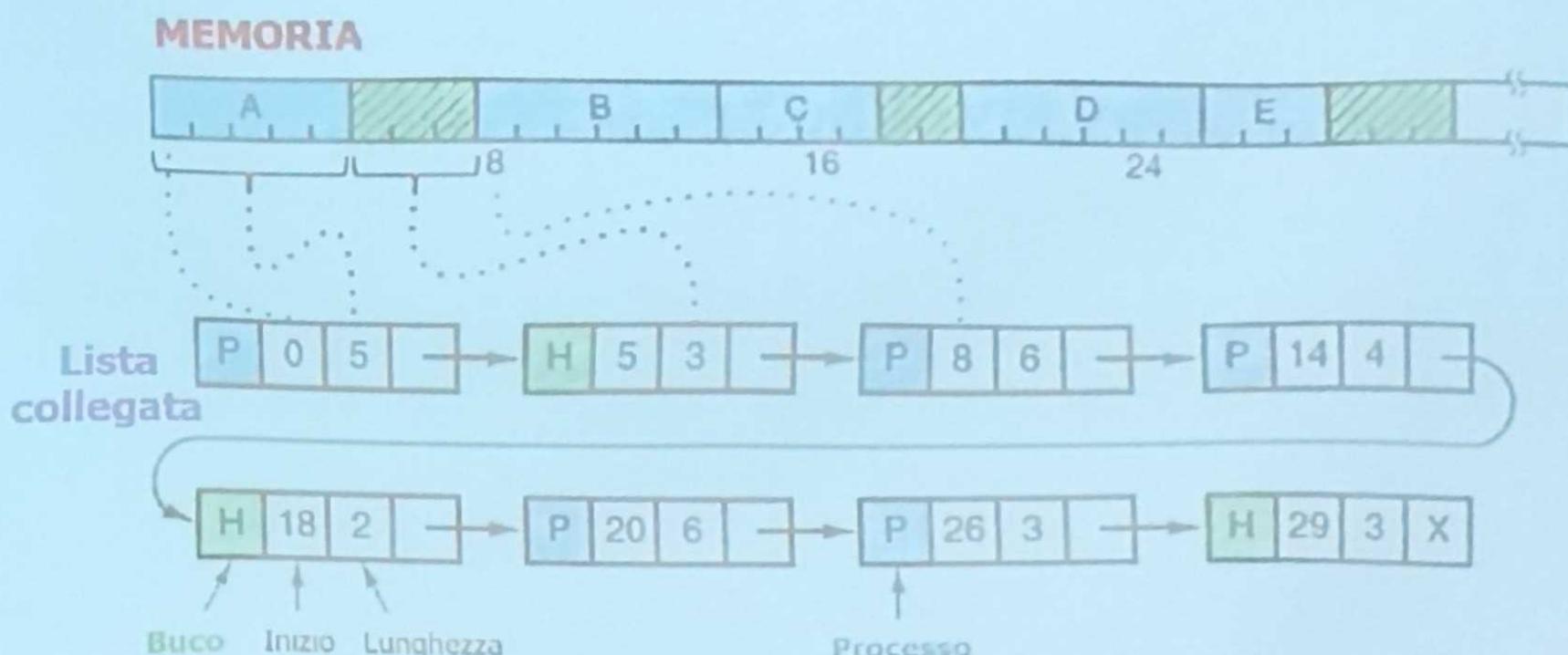


## Gestione della memoria con bitmap

- La dimensione dell'unità di allocazione gioca un ruolo chiave nella progettazione dei sistemi di gestione della memoria: più piccola è l'unità di allocazione maggiore è la dimensione della bitmap.
- Un'unità di allocazione di 4 byte (= 32 bit) richiede solo 1 bit della mappa.
- Una memoria di  $32^n$  bit che utilizza 32 bit per unità di allocazione richiederà  $32^n - 1$  bit per la mappa, cioè  $1/32$  ( $\sim 3\%$ ) dello spazio della memoria.
- Se si sceglie l'unità di allocazione grande, la bitmap sarà più piccola, ma si correrà il rischio di sprecare memoria nell'unità se la dimensione del processo non è esattamente multiplo di quella dell'unità di allocazione.
- Il problema principale di questa soluzione è che quando dobbiamo portare in memoria un processo di  $k$  unità, il gestore deve ricercare nella bitmap  $k$  bit liberi consecutivi (la ricerca sequenziale è un'operazione lenta).

# Gestione della memoria con liste collegate

- Un altro modo per gestire la memoria è attraverso una lista collegata che memorizzi le posizioni dei segmenti occupati dai processi (P) e quelle dei segmenti liberi (H).



# Gestione della memoria con liste collegate

- Nell'esempio precedente, la lista collegata è ordinata per indirizzo.
- Il vantaggio è che, quando un processo termina o viene fatto lo swap su disco, aggiornando della lista è facile.
- Quando viene creato un nuovo processo o viene caricato dal disco possono essere utilizzati vari algoritmi per trovargli una posizione in memoria:
  - First fit
  - Next fit
  - Best fit
  - Worst fit
  - Quick fit
- Si assume che il gestore della memoria conosca quanta memoria occorre allocare.

## Gestione della memoria con liste collegate: first fit

- **first fit** è il più semplice algoritmo.
- Il gestore della memoria scorre la lista finché non trova uno spazio libero abbastanza grande da contenere il processo:
  - se lo spazio libero è maggiore dello spazio richiesto viene suddiviso in due parti: una per il processo e uno per la memoria non utilizzata. Altrimenti si usa tutto lo spazio.
- **first fit** è un algoritmo veloce perché cerca di non scorrere la lista: il primo spazio libero è quello buono.

## Gestione della memoria con liste collegate: next fit

- Una piccola variante del first fit è il next fit.
- Funziona allo stesso modo del first fit, ma tiene traccia della posizione dove ha trovato l'ultimo spazio libero.
- La prossima volta che è chiamato a cercare uno spazio libero, avvia la ricerca dalla posizione dove si è interrotta l'ultima volta, invece di tornare sempre all'inizio come fa il first fit.
- Le simulazioni mostrano che il next fit dà performance leggermente peggiori di first fit.

## Gestione della memoria con liste collegate: best fit e worst fit

- **Best fit** scorre tutta la lista alla ricerca del più piccolo spazio che riesce a contenere il processo richiesto.
- L'idea è che in questo modo si cerca di minimizzare lo spreco di spazio con quello che si adatta meglio alla dimensione cercata.
- **Best fit** è più lento degli altri due perché deve scorrere tutta la lista ogni volta che viene chiamato.
- Un po' sorprendentemente tende a riempire la memoria con piccoli spazi vuoti inutilizzabili e quindi spreca più memoria di **first** e **next fit**.
- **first fit** genera mediamente buchi più grandi
- **Worst fit** funziona con logica inversa a **best fit**, cerca di prendere il più grande buco disponibile, in modo che il nuovo spazio vuoto sarà il più grande possibile.
- Le simulazioni dimostrano che anche **worst fit** non è un buon algoritmo di allocazione della memoria.

## Gestione della memoria con liste collegate: quick fit

- Quick fit mantiene elenchi separati per alcuni dei formati più comunemente richiesti: una lista per la lista degli spazi da 4KB, una per quelli da 8KB, e così via.
- Trovare uno spazio libero della dimensione desiderata è estremamente veloce.
- Ha lo stesso svantaggio di tutti gli schemi ordinati per dimensione, quando un processo termina o viene inviato sul disco, trovare i suoi vicini per unire gli spazi vuoti è un processo dispendioso.
- Se non sono fusi insieme gli spazi liberi la memoria si frammenta rapidamente in un gran numero di piccoli buchi in cui non entra nessun processo.

# Memoria virtuale

- I registri base e limite possono essere utilizzati per creare l'astrazione degli spazi di indirizzo.
- C'è un altro problema da risolvere: la gestione del bloatware.
- Le dimensioni del software aumentano molto più velocemente di quelle della memoria (che pure sta aumentando).
- Il problema dei programmi più grandi della memoria esiste fin dall'inizio dell'informatica.

# Paging

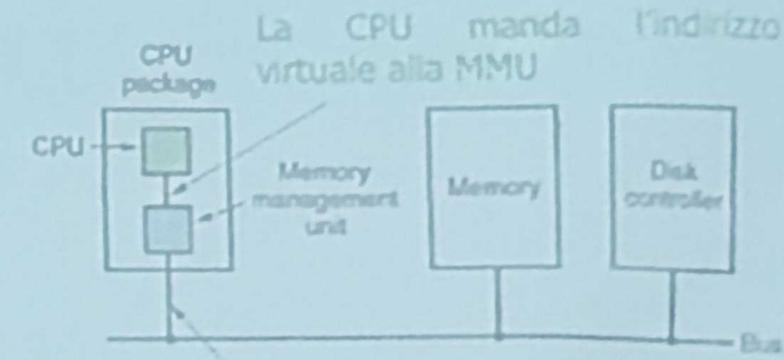
- L'idea di base della memoria virtuale è che ogni programma ha uno spazio di indirizzi composto da pagine.
- Ogni pagina è un intervallo contiguo di indirizzi ed è mappata sulla memoria fisica.
- Per eseguire il programma non è necessario mantenere tutte le pagine nella memoria fisica.
- Quando il programma fa riferimento a una parte del suo spazio di indirizzi che si trova:
  - in memoria fisica, l'hardware esegue la mappatura necessaria *on-the-fly*.
  - non in memoria fisica, il sistema operativo avverte che manca il pezzo e riesegue l'istruzione fallita.

## Paging come estensione di registro base e registro limite

- La memoria virtuale è una generalizzazione dell'idea del **registro base** e **limite**.
- Già nel 8088 ci sono registri base separati (ma non aveva i registri limite) per il testo e per i dati.
- Con la memoria virtuale, invece di avere rilocazione separata solo per i segmenti di testo e dati, l'intero spazio di indirizzamento può essere rimappato sulla memoria fisica in pagine di stessa dimensione.
- La memoria virtuale funziona bene in un sistema multiprogrammato, con bit e pezzi di molti programmi contemporaneamente in memoria.
- Mentre un programma è in attesa che venga caricata la pagina richiesta, la CPU può essere assegnata ad un altro processo.

# Paging

- La maggior parte dei sistemi di memoria virtuale utilizza una tecnica chiamata paginazione o paging.
- Quando un programma esegue l'istruzione  
`MOV REG , 1000`  
copia il contenuto della locazione di memoria 1000 nel registro REG.
- In un computer con memoria virtuale l'indirizzo non finisce sul bus indirizzi ma in una unità chiamata MMU (Memory Management Unit)
- Un bit di presenza/assenza tiene traccia di quali pagine sono presenti fisicamente in memoria.



La MMU manda l'indirizzo fisico  
alla memoria

# Paging

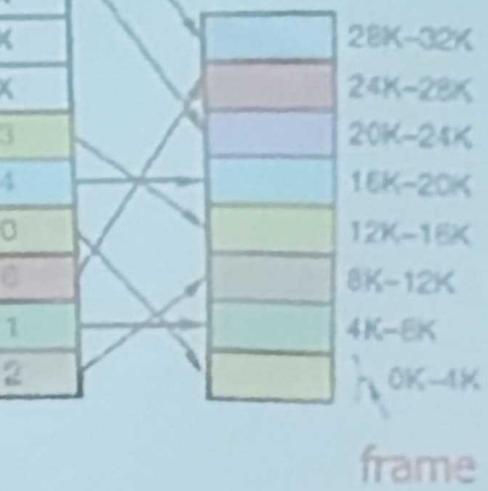
- Nei computer senza memoria virtuale, l'indirizzo virtuale viene messo direttamente sull'address bus (la parola di memoria fisica è coincide con quella virtuale).
- Lo spazio di indirizzo virtuale è suddiviso in unità di dimensione fissa chiamati pagine.
- Le unità fisiche corrispondenti alle pagine (virtuali) nella memoria sono chiamate frame o page frame che hanno in genere le stesse dimensioni.
- Nella figura pagina il frame 0K-4K indica l'intervallo di indirizzi da 0 a 4095. Il successivo frame (4K-8K) fa riferimento agli indirizzi da 4096 al 8191 e così via.

Spazio di  
indirizzamento  
virtuale

60K-64K	X
56K-60K	X
52K-56K	X
48K-52K	X
44K-48K	7
40K-44K	X
36K-40K	5
32K-36K	X
28K-32K	X
24K-28K	X
20K-24K	3
16K-20K	4
12K-16K	0
8K-12K	0
4K-8K	1
0K-4K	2

Pagina virtuale

Spazio di  
indirizzamento  
fisico

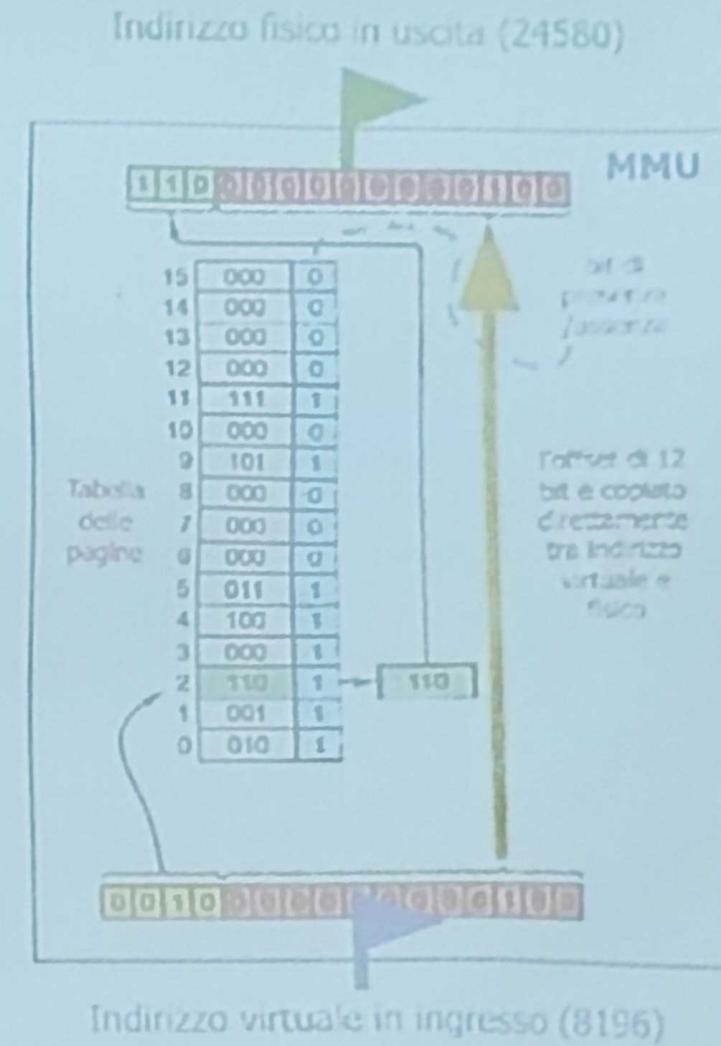


# Paging

- Quando una pagina virtuale non è nella memoria, la MMU si accorge che il riferimento nell'istruzione non è in memoria e causa un errore di pagina (**page fault**) al sistema operativo.
  - 1 Il sistema operativo seleziona il frame meno utilizzato e lo scrive sul disco (se è stato modificato)
  - 2 Poi sovrascrive il suo contenuto con la pagina appena caricata
  - 3 Cambia la mappa all'interno della MMU
  - 4 Riavvia l'istruzione che aveva causato la trap
- All'interno della MMU c'è una tabella delle pagine che permette di traslare un indirizzo virtuale in uno fisico.

# MMU: indirizzo virtuale → indirizzo fisico

- Una parte dell'indirizzo virtuale è utilizzato come indice nella tabella delle pagine, il resto come offset all'indirizzo estratto dalla tabella.



## Struttura della riga nella tabella delle pagine

- Anche se la struttura fisica del record dipende dalla macchina, il tipo di informazioni presenti è generalmente lo stesso:

N° Frame	Presenza / Assenza	Protezione	Pagina Modificata	Pagina Referenziata	Disabilita caching	
----------	--------------------	------------	-------------------	---------------------	--------------------	--

- Il numero del frame (o page frame).
- Il bit di presenza/assenza, quando è 0 significa che la pagina virtuale non è in memoria (page fault) e va caricata.
- I bit di protezione specificano che tipo di accesso è consentito (come i bit rwx di UNIX).
- Il bit pagina modificata (o sporca) è asserito quando la pagina è stata modificata, prima di sovrascriverla in memoria e va scritta sul disco.
- Il bit pagina referenziata è impostato se è stato fatto un accesso in lettura o in scrittura alla pagina. È utilizzato come criterio per lo swap delle pagine quando si verifica un page fault.
- Il bit disabilita cache permette di disabilitare il caching della pagina.

# 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

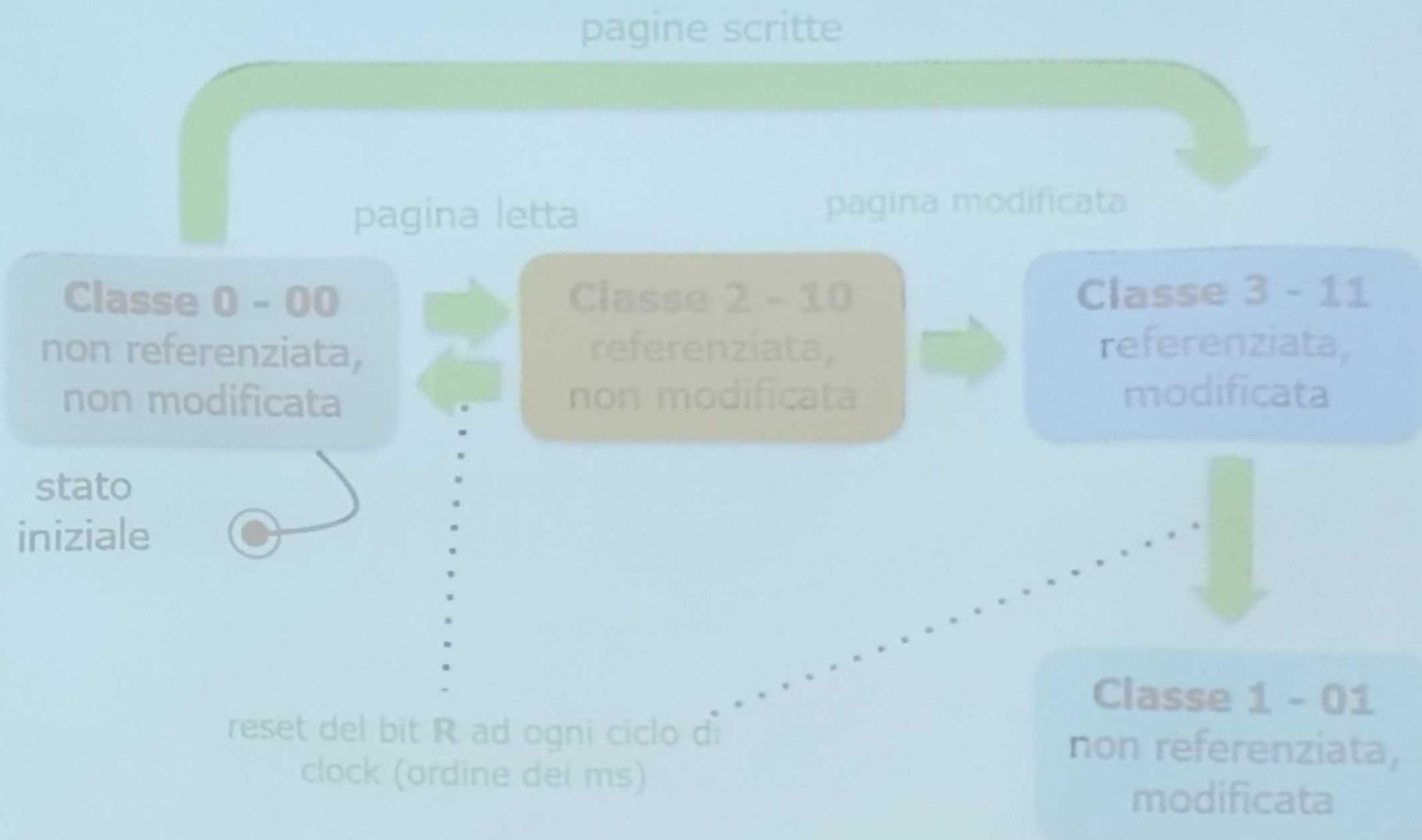
- L'algoritmo ottimo di sostituzione delle pagine è facile da descrivere, ma **impossibile da realizzare**.
- Quando accade un **page fault** alcune pagine sono già in memoria, quindi una di queste pagine sarà utilizzata dall'istruzione successiva.
- Se si etichettasse ogni pagina con il numero di istruzioni 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	Referenziata	Modificata
Classe 0	No ( $R = 0$ )	No ( $M = 0$ )
Classe 1	No ( $R = 0$ )	Sì ( $M = 1$ )
Classe 2	Sì ( $R = 1$ )	No ( $M = 0$ )
Classe 3	Sì ( $R = 1$ )	Sì ( $M = 1$ )

# Diagramma di transizione di stato

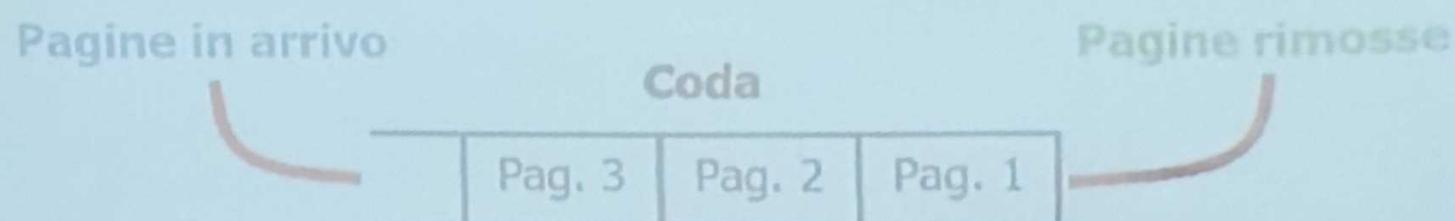


## Conclusioni su NRU

- Il ciclo di clock non azzerà il bit M perché la pagina deve essere scritta su disco prima di essere rimossa dalla memoria.
- L'algoritmo Not Recently Used (LRU) rimuove una pagina a caso dalla classe con il numero più basso.
- Filosofia: è meglio rimuovere una pagina non modificata a cui non è stato fatto alcun riferimento da almeno un ciclo di clock (tipicamente circa 20 msec) rispetto ad una pagina che è in uso.
- NRU è un algoritmo di semplice comprensione e realizzazione con performance adeguate ma non ottimali.

## First in First Out

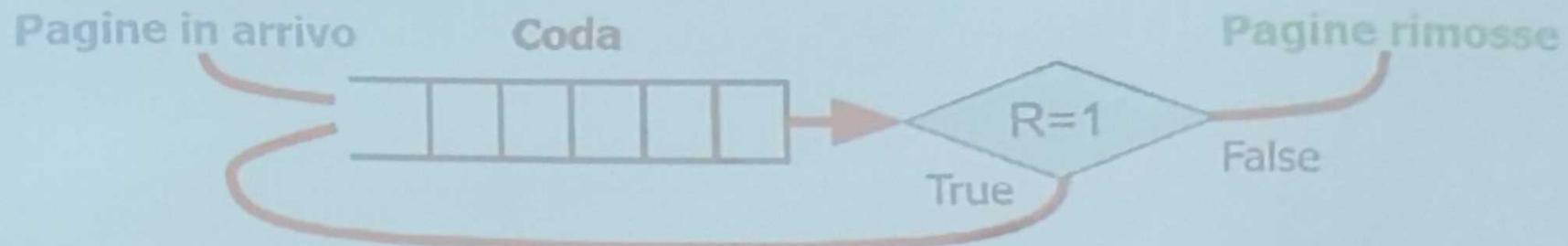
- Il sistema operativo mantiene una lista di tutte le pagine attualmente in memoria: la più recente in coda e quella meno recente in testa.



- 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.

## Second 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  $R = 0$

- 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 cambiagli 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  $n \times n$  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.

1) imposta  
tutti i bit  
della riga  
a 1

	Pag. 0	
0	0 1 2 3	
1	0 0 0 0	
2	0 0 0 0	
3	0 0 0 0	

2) imposta  
tutti i bit  
della  
colonna a  
0

	Pag. 1	
0	0 0 0 0	
1	0 1 1 1	
2	1 0 0 1	
3	1 0 0 0	

Pagina	0	1	2	3
Frequenza	2	2	2	3

	Pag. 2	
0	0 0 0 0	0
1	1 0 0 0	1
2	1 1 0 0	2
3	1 1 0 1	3

	Pag. 3	
0	0 1 0 0	4
1	0 0 0 0	0
2	1 1 0 1	12
3	1 1 1 0	14

## Not Frequently Used (NFU)

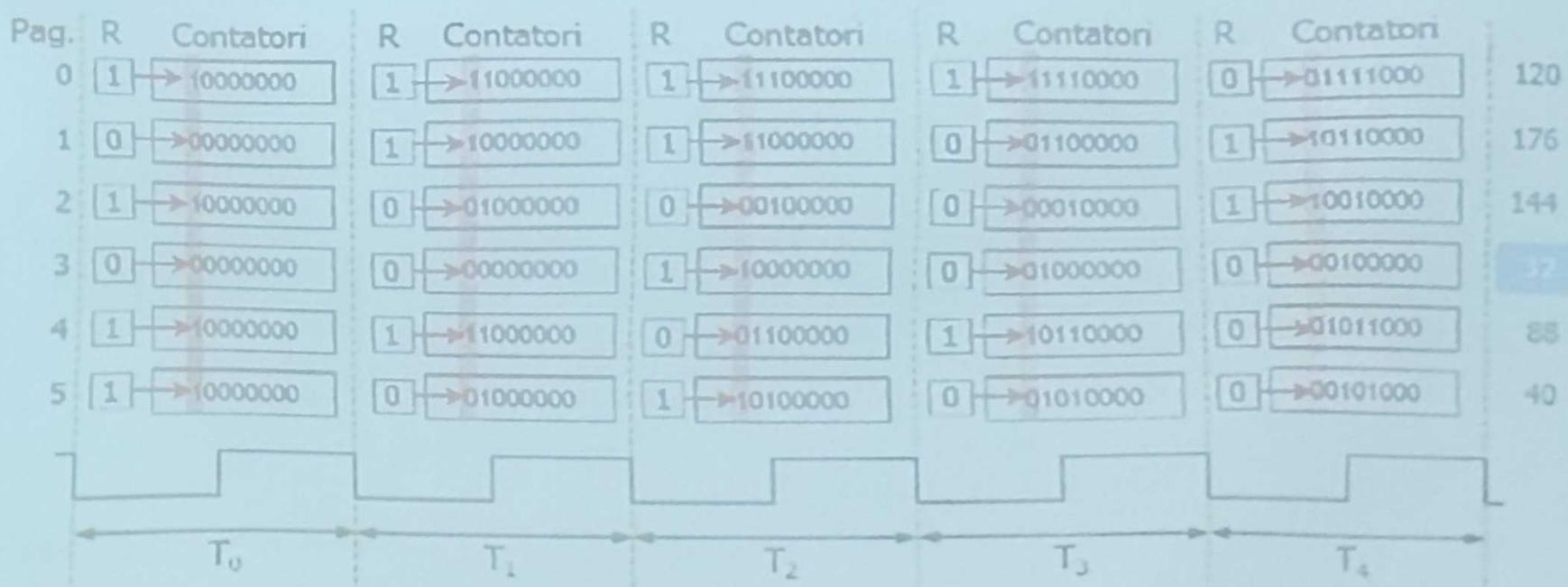
- Anche se entrambi gli algoritmi LRU sono fattibili, non esistono hardware che li realizzino.
- Un altro algoritmo che può essere implementato via software è l'NFU (Not Frequently Used):
  - Ogni pagina ha associato un contatore, inizialmente pari a zero.
  - Ad ogni interruzione di clock, il sistema operativo esegue la scansione di tutte le pagine in memoria.
  - Per ogni pagina, viene sommato 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 di 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 utilizzate 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  $\tau$  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  $\tau$  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:

età = tempo attuale - tempo di ultimo utilizzo

- se  $\text{età} > \tau$  la pagina non è più nel Working Set e può essere sostituita con una nuova.



Introduzione  
ai Sistemi  
Operativi

A. Simonetta

Introduzione  
ai Sistemi  
operativi

Processi e  
thread

Comunicazione  
tra processi  
(IPC)

Scheduling

Gestione della  
memoria

File System

Input e  
Output

Deadlock

## 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 peggio 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 ( $\tau$ ), 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 ( $\text{bit } R = 0$ ) AND ( $\text{età} > \tau$ ) AND ( $\text{bit } M = 0$ ), non è nel Working Set ed esiste una copia valida sul disco. La pagina può essere sovrascritta dalla nuova.
- Se ( $\text{bit } R = 0$ ) AND ( $\text{età} > \tau$ ) AND ( $\text{bit } M = 1$ ), 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.



Introduzione  
ai Sistemi  
Operativi

A. Simonetta

Introduzione  
ai Sistemi  
operativi

Processi e  
thread

Comunicazione  
tra processi  
(IPC)

Scheduling

Gestione della  
memoria

File System

Input e  
Output

Deadlock

## 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 sostituzione delle pagine

Algoritmo	Caratteristiche
Ottimo	Non realizzabile ma utile come riferimento
LRU (Least Recently Used)	Eccellente ma difficile da realizzare con precisione
NRU (Not Recently Used)	Approssimazione dell'LRU
FIFO (First-In, First Out)	Potrebbe buttare via pagine importanti, non utile
Seconda chance	Miglioramento del FIFO
Clock	Versione del Seconda chance, migliore nei tempi
NFU (Not Frequently Used)	Approssimazione rossa dell'LRU
Aging	Efficiente, approssima bene l'LRU
Working Set	Troppo costoso da realizzare
WSClock	Efficiente e valido