Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2023/2024

Corso di Laurea in Matematica

La Gestione della Memoria - Parte 3

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

SAPIENZA
UNIVERSITÀ DI ROMA

Argomenti trattati

Gestione della memoria

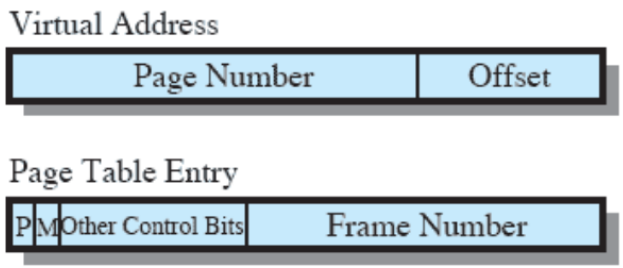
Memoria virtuale: supporto hardware

Memoria Virtuale: Supporto Richiesto

- Paginazione e segmentazione vengono utilizzate per realizzare la memoria virtuale
- Il SO deve essere in grado di muovere pagine e/o segmenti dalla memoria principale alla secondaria
- La realizzazione di alcune operazioni deve avere il supporto dell'hardware
  - infatti alcune operazioni sarebbero troppo lunghe se realizzate in modo software dal SO
  - ad esempio, la traduzione degli indirizzi avviene tramite hardware

Paginazione

- Per la paginazione abbiamo visto che:
  - ogni processo ha una sua tabella delle pagine
  - quando tutte le pagine vengono caricate in memoria principale, viene caricata anche la tabella delle pagine
  - il PCB punta alla tabella
- Ogni riga della tabella contiene:
  - il numero di frame in memoria principale (il numero di pagina è usato per indicizzare la tabella)
  - un bit (**P**) per indicare se la pagina è in memoria principale
  - un bit (**M**) per indicare se la pagina è stata modificata da quando è stata caricata in memoria principale



Traduzione degli Indirizzi

Realizzazione hardware dell'indirizzo fisico

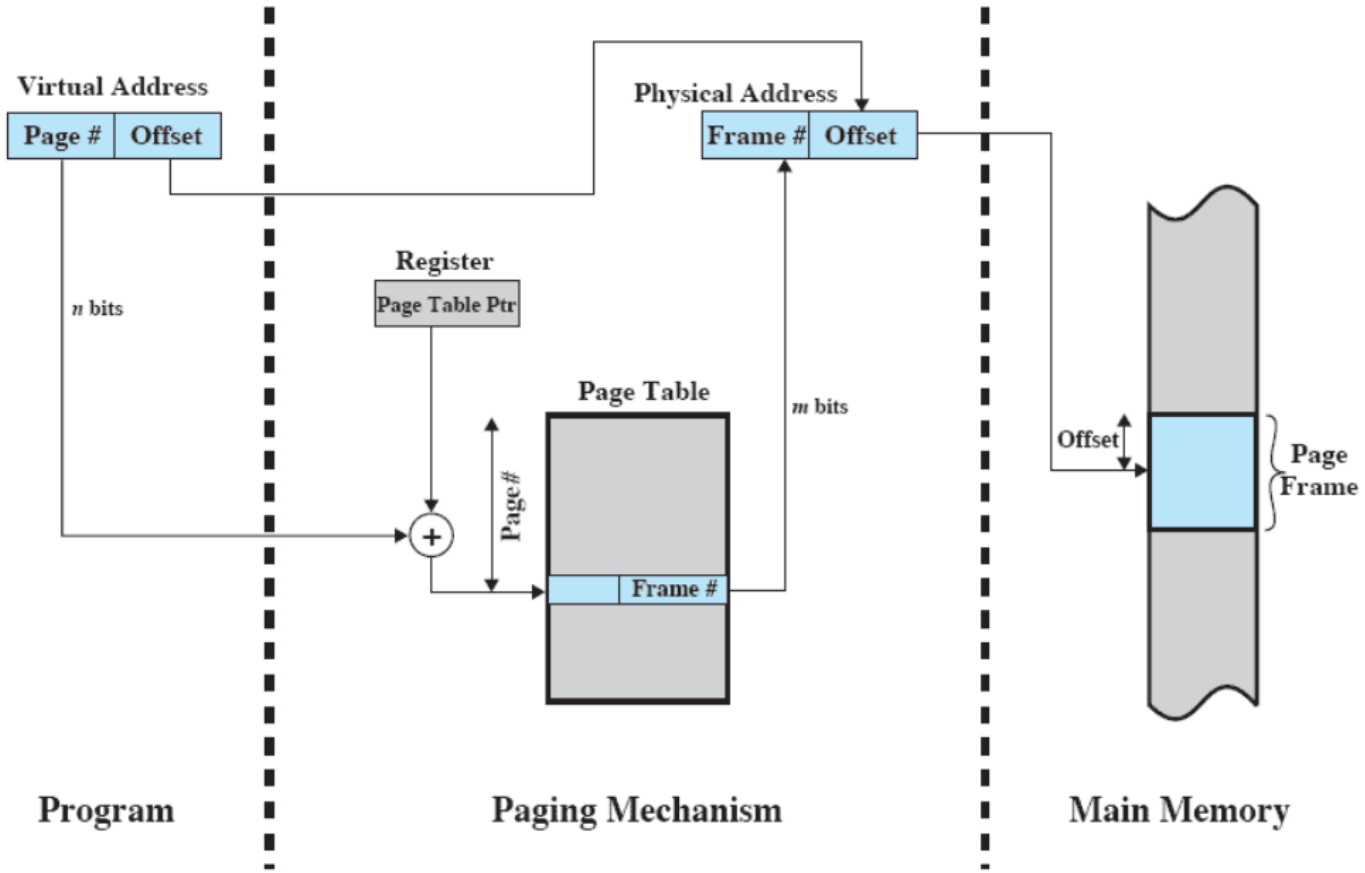


Tabelle delle Pagine

- Le tabelle delle pagine potrebbero contenere molti elementi
- Possono essere anch'esse divise in pagine e potenzialmente essere swappate su disco
  - alcuni processori (ad es. Pentium) richiedono che la tabella delle pagine di ciascun processo occupi al più una pagina
- Quando un processo è in esecuzione, viene assicurato che almeno una parte della sua tabella delle pagine sia in memoria principale

Tabelle delle Pagine

- **Esempio**: Assumiamo che lo spazio virtuale sia 8GB ($2^3 \cdot 2^{30}$) e ogni pagina occupi 1kB 2^{10}
  - Quindi una tabella delle pagine, relativa a un singolo processo, può avere $\frac{2^{30+3}}{2^{10}} = 2^{23}$ righe
  - Ogni riga occupa: 1 byte di controllo + $\log_2$(size RAM in frames) byte
    - Ad esempio, se la RAM è da 4GB (architettura a 32-bit) si hanno 4 bytes
    - cioè 32 bit - 10 bit = 22 bit per i frame, cioè 3 bytes, più il byte di controllo
  - Quindi per ogni tabella delle pagine (cioè per ogni processo) c'è un overhead di $4 \cdot 2^{23} = 2^{23+2} = 32$MB
  - Se ci sono 30 processi, l'overhead (occupazione) in RAM è di 1GB (cioè un quarto) per le sole tabelle delle pagine

Tabella delle Pagine a 2 Livelli

Alcuni processori usano uno schema a due livelli per organizzare tabelle delle pagine grandi: una directory delle pagine in cui ogni elemento punta ad una tabella delle pagine

Ovviamente, il processore deve avere hardware dedicato per i 2 livelli di traduzione (il SO si deve adattare all'hardware)

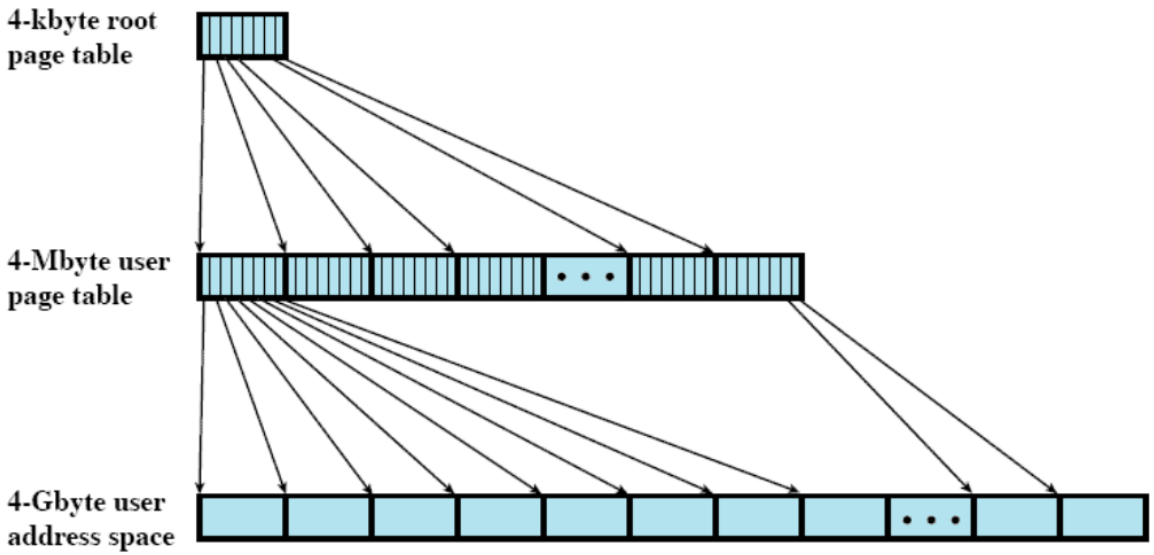
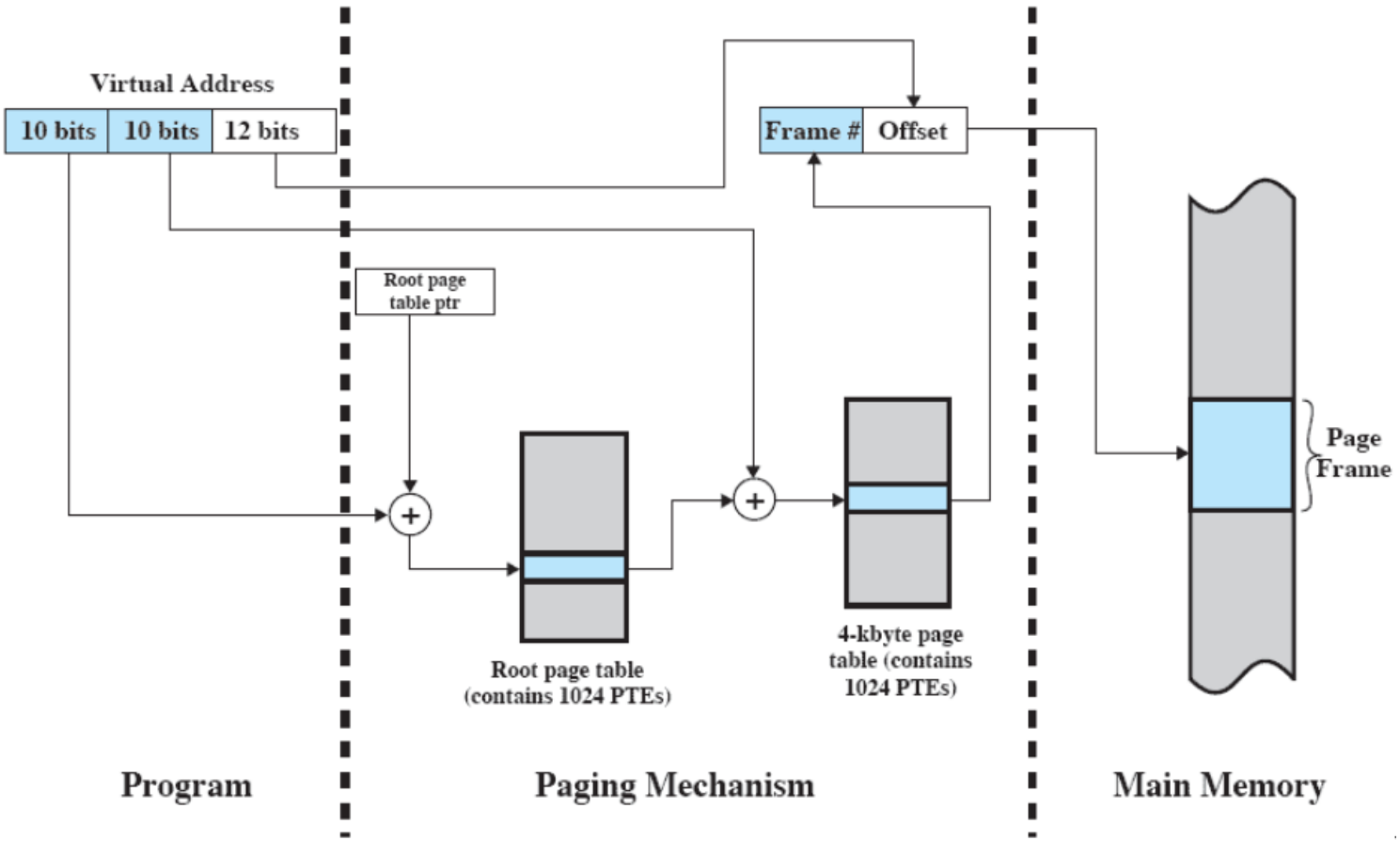


Tabelle delle Pagine a 2 Livelli

- **Esempio**: 8GB di spazio virtuale $\rightarrow$ 33 bits di indirizzo
  - Suddividiamo i 33 bit (ad es.): 15 bit primo livello (*directory*), 8 bit di secondo livello, 10 bit rimanenti per l'offset
    - spesso i processori impongono che una page table di secondo livello entri in una pagina (ad es. Pentium)
    - così, effettivamente, essa occupa $2^8 \cdot 2^2 = 2^{10}$ bytes (1kB)
  - Per ogni processo, l'overhead è $2^{23+2} = 32$MB, più l'occupazione del primo livello: $2^{15+2} = 128$kB: sempre all'incirca 32MB
  - Però è più facile paginare la tabella delle pagine: in RAM basta che ci sia il primo livello più una tabella del secondo
  - quindi l'overhead scende a $2^{15+2} + 2^{8+2} = 128$kB
  - Con RAM di 4 GB, occorrono 2000 processi per occupare circa un quarto delle RAM con strutture di overhead

Tabella a 2 Livelli: Traduzione

Realizzazione hardware



Dimensione delle Pagine

- La dimensione delle pagine rappresenta un'importante decisione hardware
- Più piccola è una pagina, minore è la frammentazione all'interno delle pagine
- Ma è anche maggiore il numero di pagine per processo
- Ciò implica una una tabella delle pagine più grande (per ogni processo)
- E quindi porzioni delle tabelle delle pagine di processi attivi finiscono in memoria secondaria
- La memoria secondaria è ottimizzata per trasferire grossi blocchi di dati, quindi meglio avere pagine grandi

Dimensione delle Pagine

- Le cose diventano più complicate se si considera l'effetto della dimensione delle pagine sui page fault
- Se le pagine sono piccole, più pagine di un processo possono risiedere in memoria centrale e, dopo un tempo di avvio, i fault diminuiscono
- Se si considera una dimensione maggiore, si perde il principio di località e si ha un maggior numero di page fault
- Con pagine molto grandi, i page fault saranno pochi e non ce ne sono affatto se tutto il processo è in memoria principale
- Le cose sono ulteriormente complicate dal fatto che il tasso di page fault è anche determinato dal numero di frame allocati per processo

Dimensione delle Pagine in Alcuni Sistemi

- Le moderne architetture HW possono supportare diverse dimensioni delle pagine (anche fino ad 1GB)
- Il sistema operativo ne sceglie una: Linux sugli x86 va con 4kB
- Le dimensioni più grandi sono usate in sistemi operativi di architetture grandi: cluster, grandi server, ma anche per i sistemi operativi stessi (kernel mode)

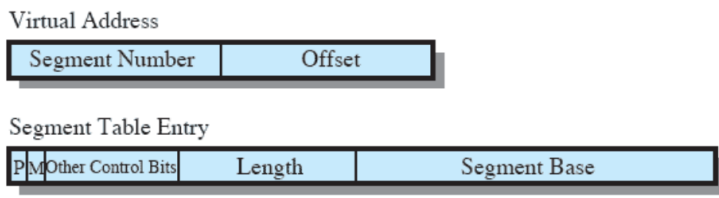
Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbyes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
IBM POWER	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Segmentazione

- La **segmentazione** permette al *programmatore* di vedere la memoria come un insieme di spazi (segmenti) di indirizzi
- La dimensione degli indirizzi può essere variabile ed anche dinamica
- I vantaggi della segmentazione sono:
  - Semplifica la gestione delle strutture dati che crescono
  - Permette di modificare e ricompilare i programmi in modo indipendente
  - Permette di condividere dati
  - Permette di proteggere dati

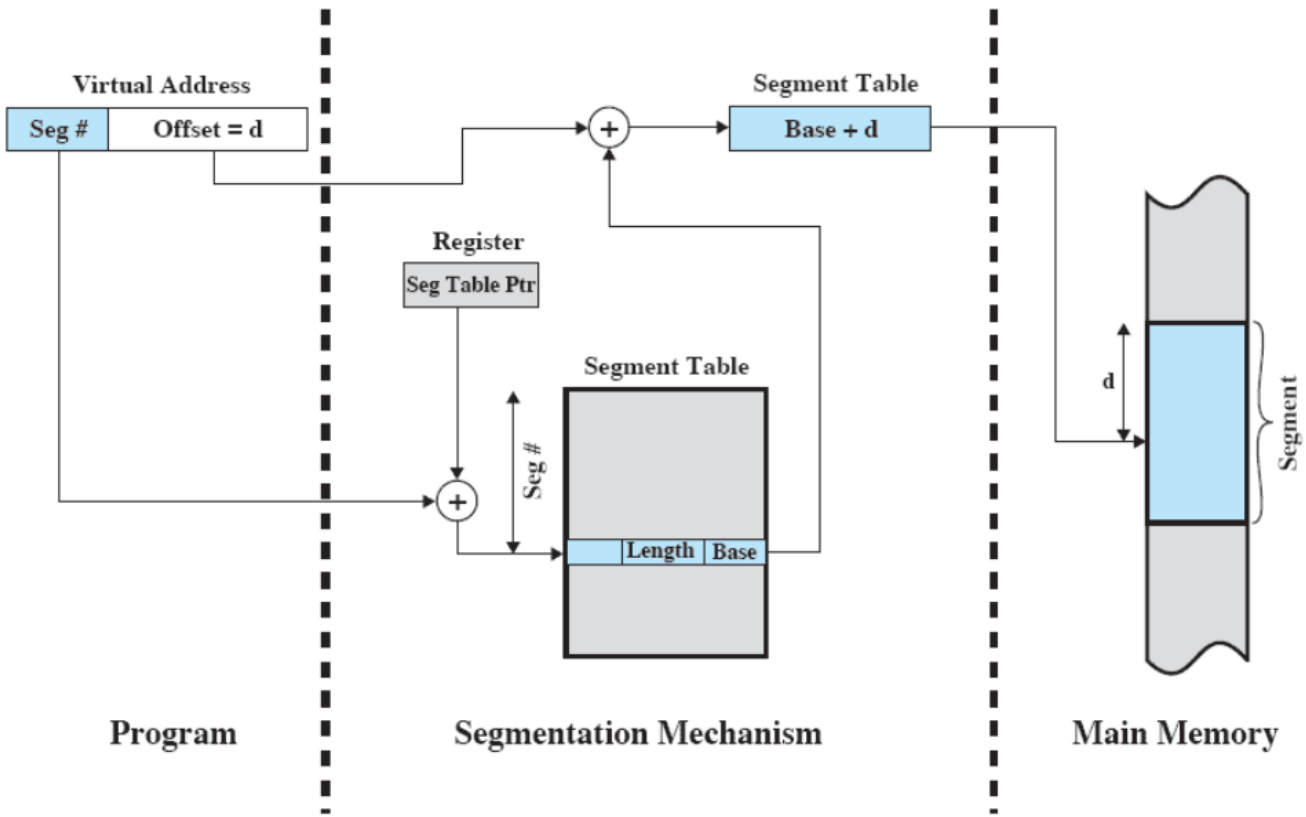
Segmentazione: Organizzazione

- Per la segmentazione abbiamo visto che:
  - ogni processo ha una sua tabella dei segmenti
  - quando tutti i segmenti vengono caricati in memoria principale, viene caricata anche la tabella dei segmenti
  - il PCB punta a tale tabella
- Ogni riga di questa tabella contiene:
  - l'indirizzo iniziale (in memoria principale) del segmento
  - la lunghezza del segmento
  - un bit (**P**) per indicare se il segmento è in memoria principale o no
  - un bit (**M**) per indicare se il segmento è stato modificato da quando è stato caricato in memoria principale



Segmentazione: Traduzione degli Indirizzi

Realizzazione hardware



Paginazione + Segmentazione

- Sia **paginazione** che **segmentazione** hanno dei punti di forza
- La **paginazione** è trasparente al programmatore
  - nel senso che il programmatore non ne è (o non ne deve essere) a conoscenza
  - vale anche per il compilatore
- La **segmentazione** è visibile al programmatore
  - ovviamente, se programma in assembler
  - altrimenti, ci pensa il compilatore ad usare i segmenti
- Si possono combinare le due tecniche e avere **paginazione + segmentazione** dividendo ogni segmento in più pagine

Paginazione + Segmentazione

Virtual Address

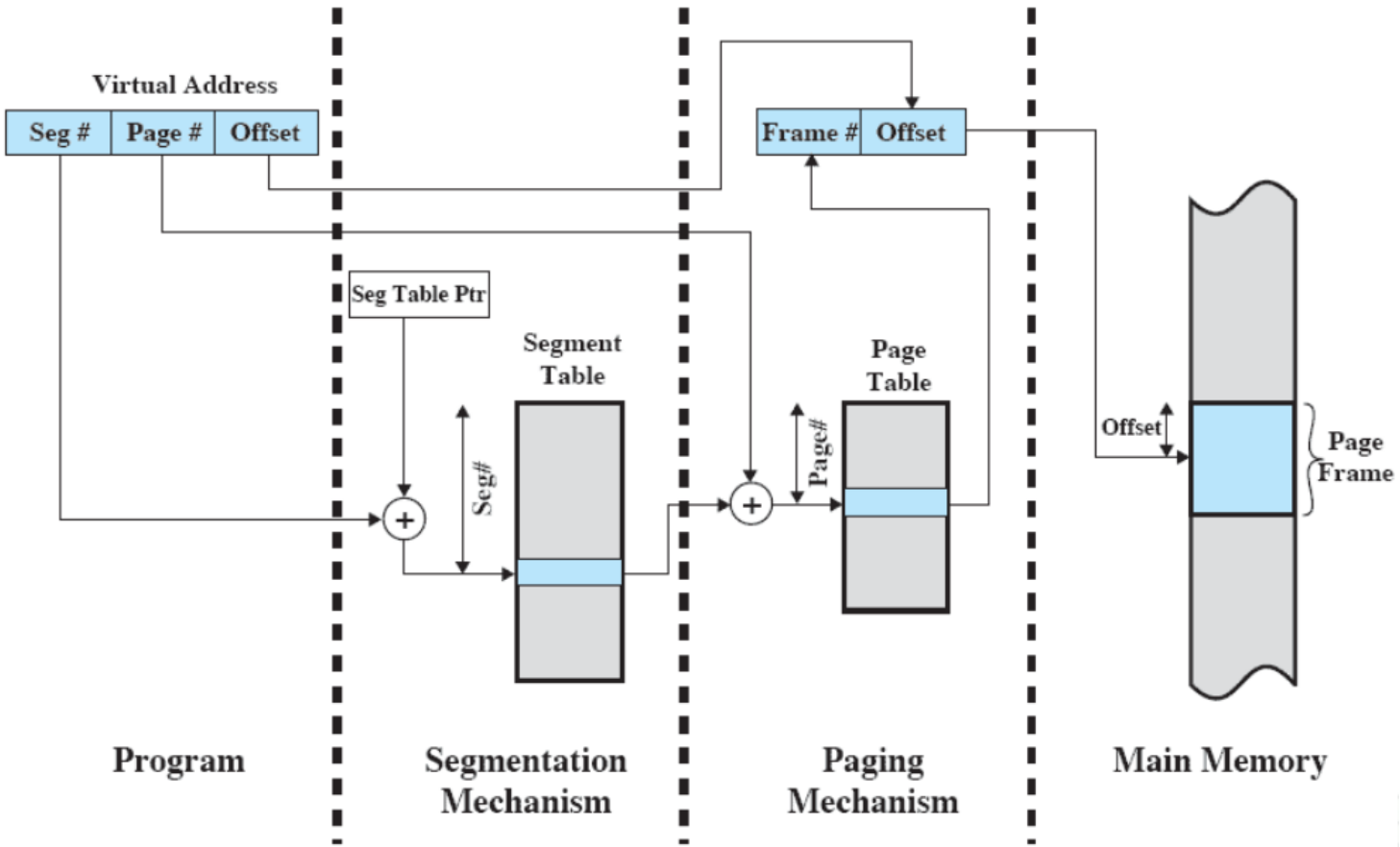
Segment Table Entry

Page Table Entry

P= present bit
M = Modified bit

Paginazione + Segmentazione: Traduzione degli Indirizzi

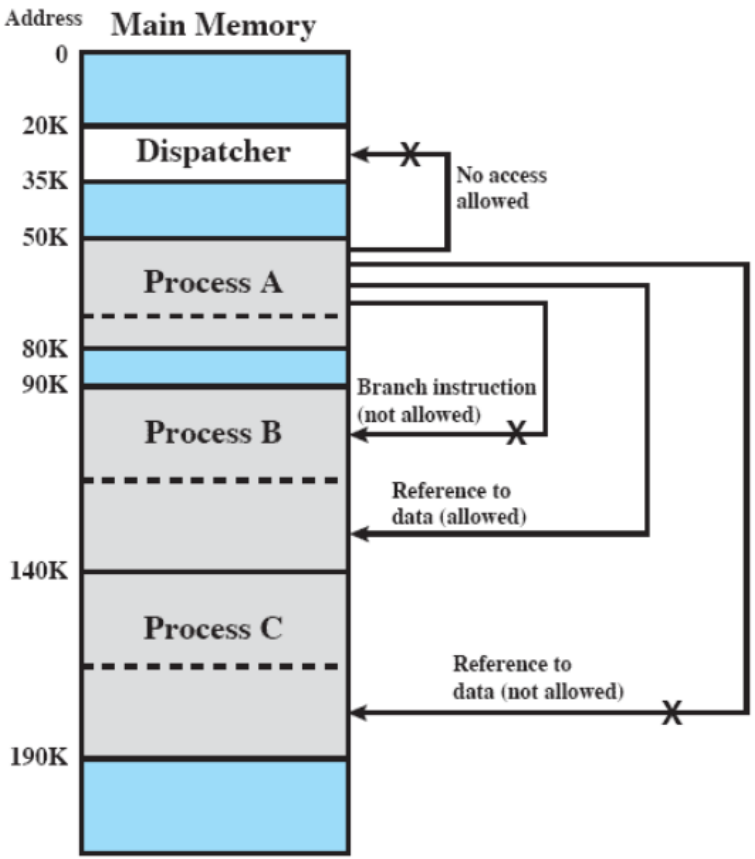
Realizzazione hardware



Protezione e Condivisione

- Con la segmentazione, implementare protezione e condivisione viene naturale
- Dato che ogni segmento ha una base ed una lunghezza, è facile controllare che i riferimenti siano contenuti nel giusto intervallo
- Per la condivisione, basta dire che uno stesso segmento serve più processi

Protezione


Address
Main Memory
0
20K
35K
50K
80K
90K
140K
190K
Dispatcher
Process A
Process B
Process C
No access allowed
Branch instruction (not allowed)
Reference to data (allowed)
Reference to data (not allowed)

Gestione della memoria

Memoria virtuale e sistema operativo

Gestione della Memoria: Decisioni

Progettare la porzione di SO che si occupa della gestione della memoria comporta una serie di decisioni

- Usare o no la memoria virtuale?
- Usare solo la paginazione, segmentazione o entrambe?
- Che algoritmi usare per gestire i vari aspetti della gestione della memoria?

I primi due punti dipendono dall'hardware disponibile, mentre l'ultimo punto è responsabilità del SO

Gestione della Memoria: Decisioni

- In ogni caso si vuole minimizzare il tasso di page fault, perchè causano un notevole overhead di tempo
- L'overhead include:
  - la decisione di quali pagine rimpiazzare
  - la scelta del processo da mandare in esecuzione
  - la gestione del process switch
- Quindi, è importante minimizzare la probabilità che il processo in esecuzione faccia riferimento a un'istruzione o un dato non presente in memoria principale generando page fault

Elementi centrali per il progetto del SO

La progettazione della gestione della memoria riguarda principalmente le seguenti politiche:

- Politica di prelievo (*fetch policy*)
- Politica di posizionamento (*placement policy*)
- Politica di sostituzione (*replacement policy*)

Non è possibile stabilire quale delle politiche possibili sia migliore in assoluto

Fetch Policy

- La **politica di prelievo** determina quando una data pagina debba essere portata in memoria principale
- Si usano principalmente due politiche:
  - paginazione su richiesta (*demand paging*)
  - prepaginazione (*prepaging*)
- **N.B.** Quando un processo viene sospeso e swappato in memoria secondaria, tutte le sue pagine vengono spostate e al ritorno tutte le pagine vengono ricaricate

Demand Paging e Prepaging

- **Demand paging**
  - una pagina viene portata in memoria principale nel momento in cui un qualche processo la richiede
  - molti page fault nei primi momenti di vita del processo
  - man mano che pagine vengono caricate i fault diminuiscono per il principio di località
- **Prepaging**
  - vengono portate in memoria principale più pagine di quelle richieste
  - ovviamente, si tratta di pagine vicine a quella richiesta (si può fare efficientemente sul disco)
  - non è una politica efficiente se poi le pagine caricate non vengono utilizzate

Placement policy

- La **politica di posizionamento** serve a decidere dove mettere una pagina in memoria principale, *quando c'è almeno un frame libero*
  - se non ci sono frame liberi, allora *replacement policy*
- La pagina può essere messa ovunque, grazie all'hardware per la traduzione degli indirizzi
- Tipicamente, la pagina viene messa nel primo frame libero
  - dove per *primo* si intende il frame con indirizzo più basso

Algoritmi di Sostituzione

I principali algoritmi (di base) per la selezione della pagina da sostituire sono:

- Sostituzione ottima (**Optimal**)
- Sostituzione della pagina usata meno di recente (**LRU: Least Recently Used**)
- Sostituzione a coda (**FIFO: First In First Out**)
- Sostituzione ad orologio (**clock**)

Algoritmi di Sostituzione

- Gli esempi riportati nel seguito usano tutti la stessa sequenza di richieste a pagine:

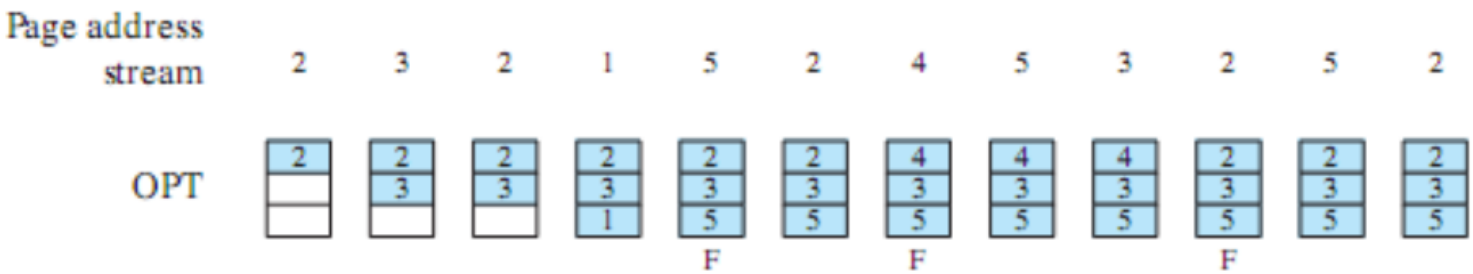
2 3 2 1 5 2 4 5 3 2 5 2

- Si suppone inoltre che ci siano solo 3 frame in memoria principale

Sostituzione Ottima

- Con la politica di sostituzione **ottima** si sostituisce la pagina che verrà richiesta più in là nel tempo
- Ovviamente, non è implementabile perchè il SO non ha conoscenza di ciò che avverrà nel futuro
- È utilizzata come standard per valutare altre politiche
- Infatti visto che nessuna politica può fare meglio dell'ottimo serve a valutare quanto si sia distanti

Sostituzione ottima sull'esempio



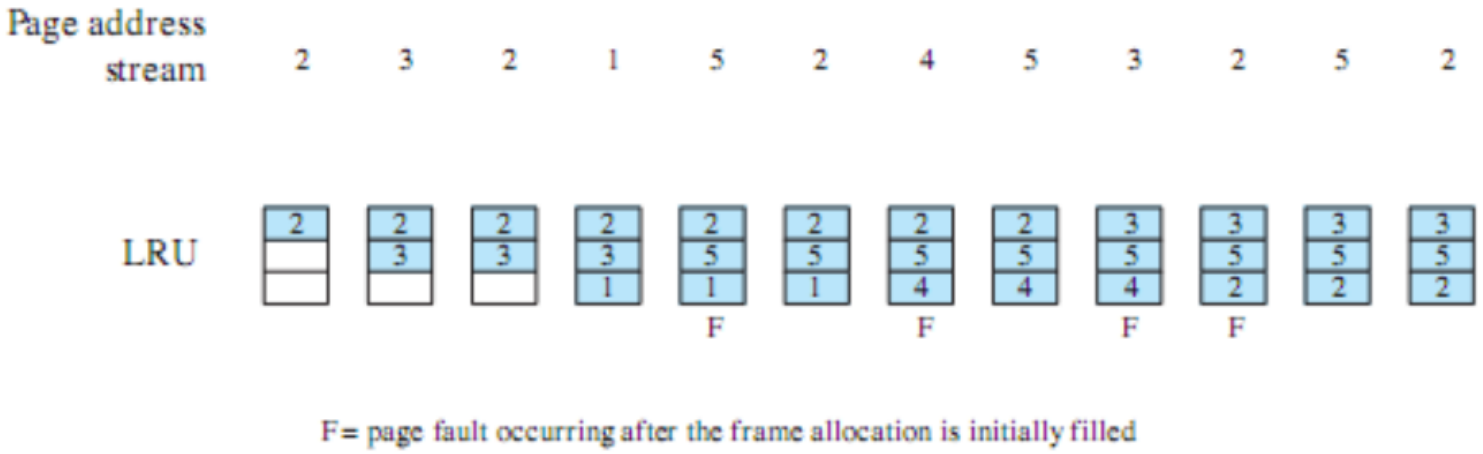
F= page fault occurring after the frame allocation is initially filled

Risultato: 3 page faults

Sostituzione LRU

- Con la politica **LRU** si sostituisce la pagina cui non è stato fatto riferimento per il tempo più lungo
- Basandosi sul principio di località, dovrebbe essere la pagina che ha meno probabilità di essere usata nel prossimo futuro
- L'implementazione è difficile:
  - occorre etichettare ogni frame con il tempo dell'ultimo accesso
  - e poi confrontare tutti i tempi
  - anche per la cache si usa questa tecnica ma è implementata in hardware, cosa che per la memoria secondaria non si può fare

Sostituzione LRU sull'esempio

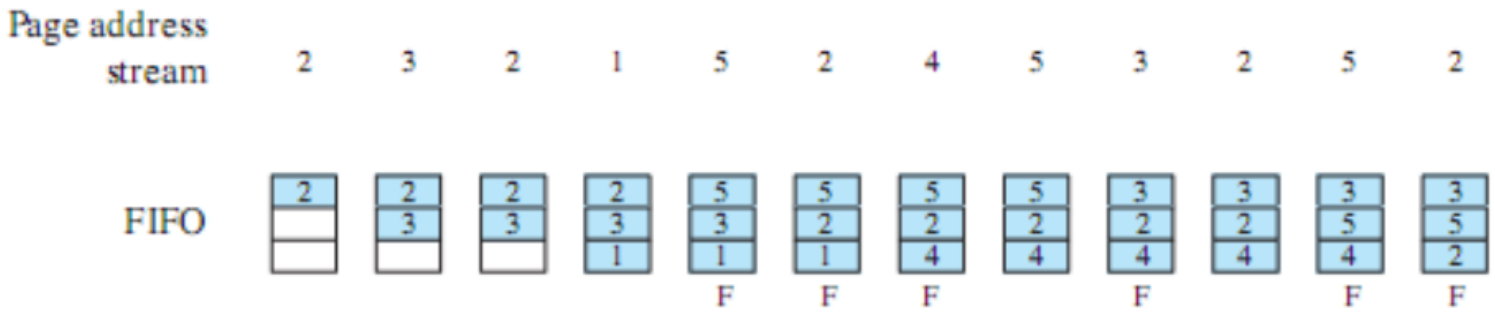


Risultato: 4 page faults (vicino all'ottimo)

Sostituzione FIFO

- Con la politica **FIFO** i frame allocati ad un qualche processo sono trattati come una coda circolare
- Da questa coda, le pagine vengono rimosse a turno (*round robin*)
- L'implementazione è semplice
- Si rimpiazzano le pagine che sono state in memoria per più tempo
  - però non è detto che non servano più: magari alcune di loro hanno molti accessi

Sostituzione FIFO sull'esempio



F= page fault occurring after the frame allocation is initially filled

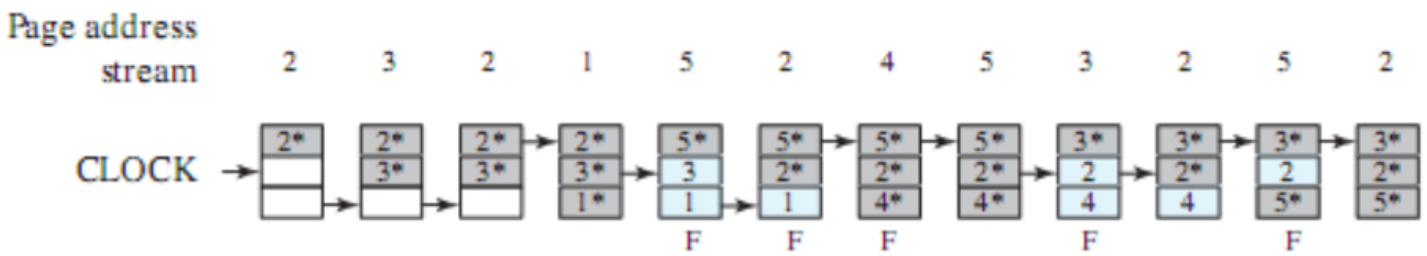
Risultato: 6 page faults

N.B. FIFO non *vede* che le pagine 2 e la 5 sono molto richieste

Sostituzione dell'orologio

- La politica del **clock** è un compromesso tra LRU e FIFO
- Si usa uno *use bit* per ogni frame, per indicare se la pagina caricata nel frame è stata riferita
- Il bit è settato ad 1 quando la pagina viene caricata in memoria principale, e poi rimesso ad 1 per ogni accesso
- Quando occorre sostituire una pagina, il SO cerca il frame adatto come nella FIFO
- Ma seleziona il frame contenente la prima pagina che ha lo *use bit* a 0
- Se invece incontra una pagina che lo ha a 1, lo mette a 0 e procede con la successiva

Sostituzione dell'orologio sull'esempio

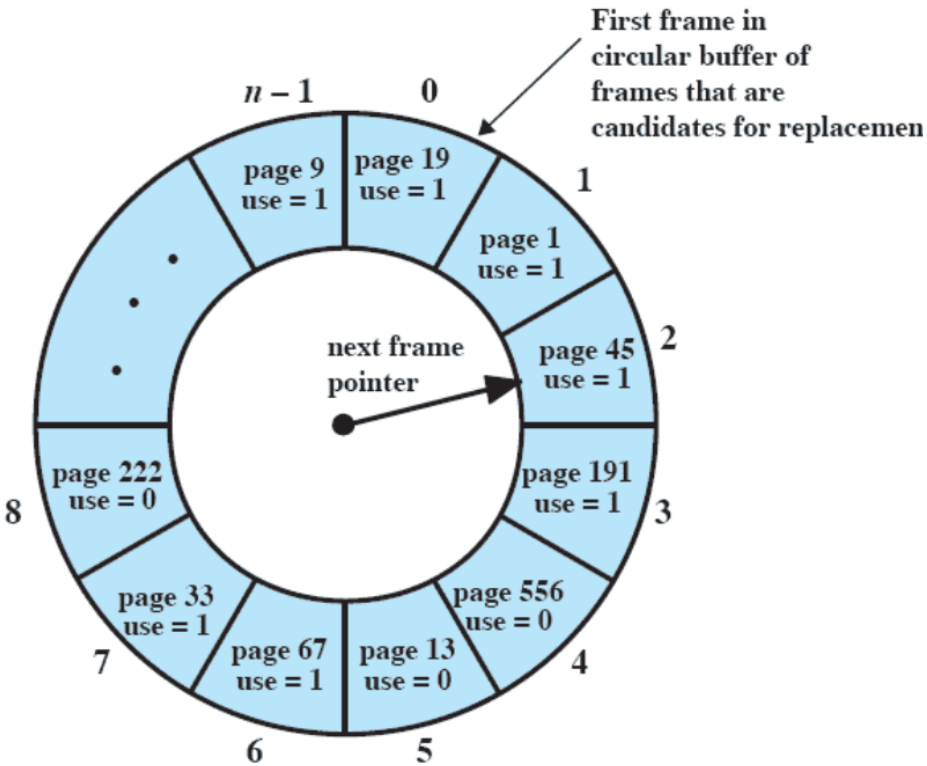


F= page fault occurring after the frame allocation is initially filled

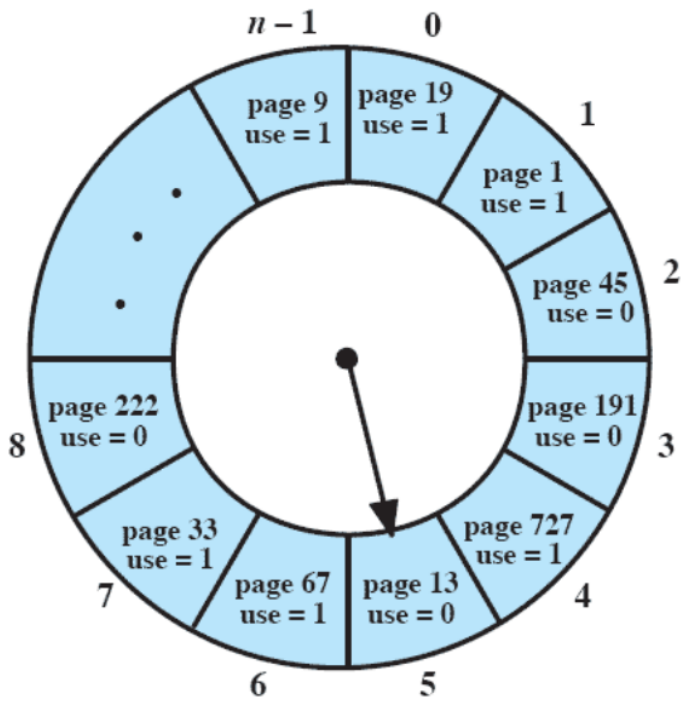
Risultato: 5 page faults

N.B. La politica dell'orologio si accorge che le pagine 2 e 5 sono molto richieste

Politica dell'Orologio



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Algoritmi di sostituzione sull'esempio

F= page fault occurring after the frame allocation is initially filled