Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2020/2021

Corso di Laurea in Matematica

La Gestione della Memoria

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

1. Gestione della memoria
   - Requisiti di base
   - Partizionamento della memoria
   - Paginazione
   - Segmentazione
2. Memoria virtuale
   - Memoria virtuale: concetti generali
   - Memoria virtuale: supporto hardware
   - Memoria virtuale e sistema operativo

Gestione della memoria

Requisiti di base

Perché Gestire la Memoria (nel SO)

- La **memoria** è oggi a basso costo, con trend in diminuzione
- Ciò è motivato dal fatto che le moderne applicazioni richiedono sempre maggiore memoria
- Gestire la memoria include lo swap di blocchi di dati in memoria secondaria
- Questa gestione, essendo la memoria secondaria un dispositivo di I/O, è ovviamente lenta (più lenta del processore)
  - il SO deve pianificare lo swap in modo intelligente, così da massimizzare l'efficienza del processore
- La **gestione della memoria** deve garantire che ci sia sempre un numero ragionevole di processi pronti all'esecuzione, così da non lasciare inoperoso il processore

Requisiti per la Gestione della Memoria

I requisiti di base sono:

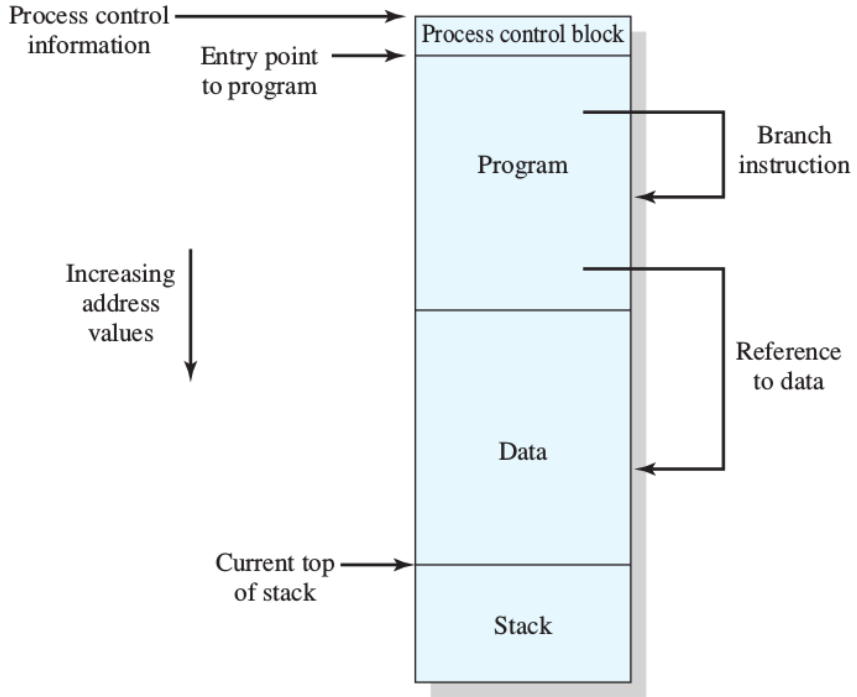
- **Rilocazione**
  - serve aiuto hardware
- **Protezione**
  - serve aiuto hardware
- **Condivisione**
- **Organizzazione logica**
- **Organizzazione fisica**

Requisiti: Rilocazione

- Il *programmatore* non sa (e non ha bisogno di sapere) in quale zona della memoria il programma verrà caricato
  - potrebbe essere swappato su disco, e al ritorno in memoria principale potrebbe essere in un'altra posizione
  - potrebbe essere in porzioni di memoria non contigue, oppure con alcune parti in RAM e altre su disco
  - in questo contesto, per *programmatore* si intende chi usa l'assembler o il compilatore
- I riferimenti alla memoria devono essere tradotti nell'indirizzo fisico *vero*
  - preprocessing o run-time
  - se a run-time, occorre supporto hardware

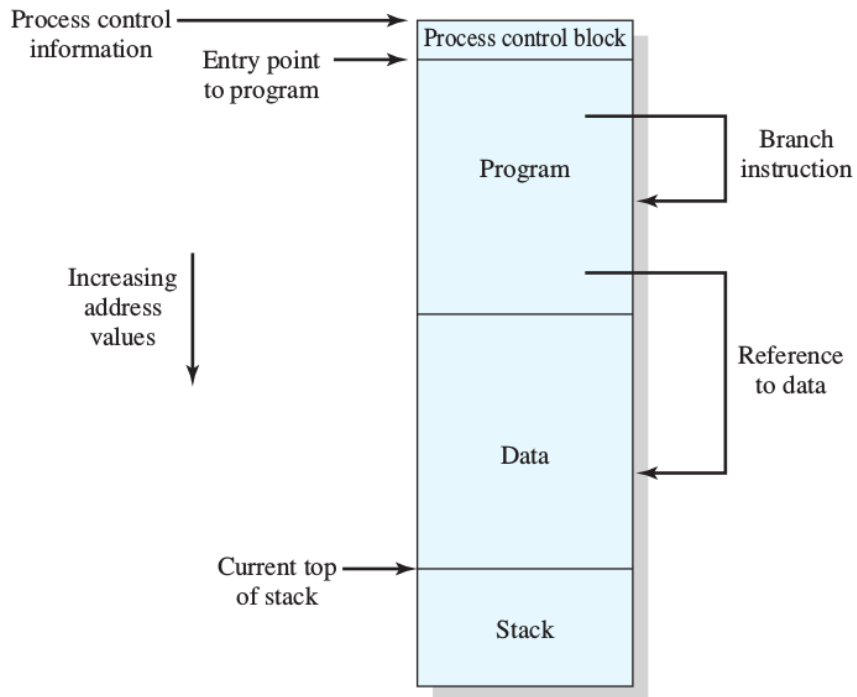
Rilocazione: gli Indirizzi nei Programmi

- In figura l'immagine di un processo
- Assumiamo occupi una porzione di memoria contigua
- Il sistema operativo deve conoscere:
  - la locazione delle informazioni di controllo del processo
  - la locazione dell'istruzione di inizio del programma per avviare il processo
  - la locazione della stack



Rilocazione: gli Indirizzi nei Programmi

- Il sistema operativo conosce questi indirizzi perchè trasferisce il programma in memoria
- Il processore deve usare gli indirizzi all'interno del programma:
  - per le istruzioni di salto (branch)
  - indirizzi dei dati presenti nelle istruzioni.
- Processore (hw) e SO (sw) traducono i riferimenti alla memoria in indirizzi fisici per ottenere la locazione corrente in memoria principale



Rilocazione a Run-Time senza Hardware Speciale

- Ogni volta che un processo viene riportato in memoria, potrebbe essere in un porzione diversa di memoria
- Nel frattempo, potrebbero essere arrivati altri processi e prenderne il posto
- Quindi, ad ogni ricaricamento in RAM, occorre individuare gli indirizzi presenti nel codice sorgente del processo e determinare i valori effettivi
- Troppo overhead per il SO, che viene quindi aiutato con soluzioni hardware

Indirizzi

Logici: il riferimento in memoria è indipendente dall'attuale posizionamento del programma in memoria

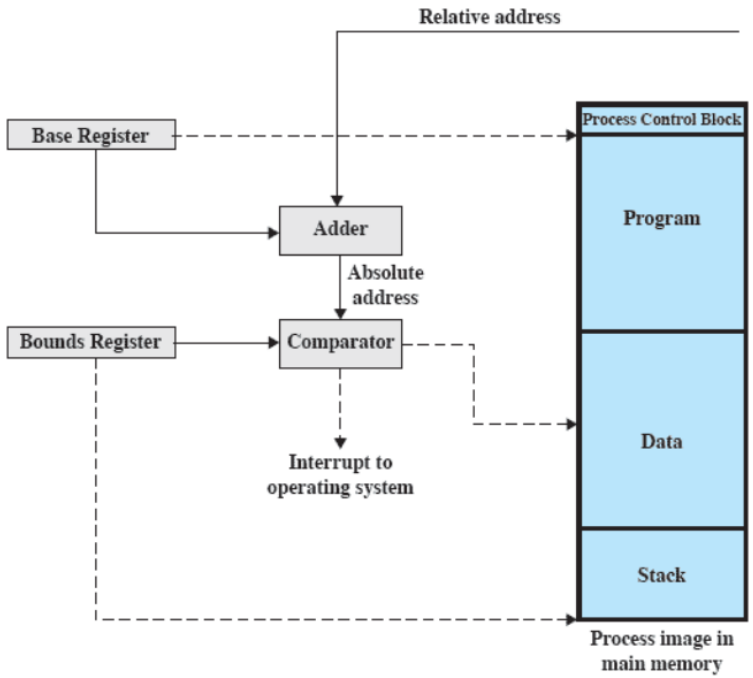
Relativi: il riferimento è espresso come uno spiazzamento rispetto ad un qualche punto noto

- caso particolare degli indirizzi logici

Fisici o Assoluti: il riferimento effettivo alla memoria

Rilocazione: gli Indirizzi nei Programmi

- Base register (registro base)
  - indirizzo di partenza del processo
- Bounds register (registro limite)
  - indirizzo di fine del processo
- Vengono settati quando il processo viene posizionato in memoria
  - mantenuti nel PCB del processo
  - passo 6 del process switch (slides sui processi)
  - vanno calcolati, non semplicemente ripristinati

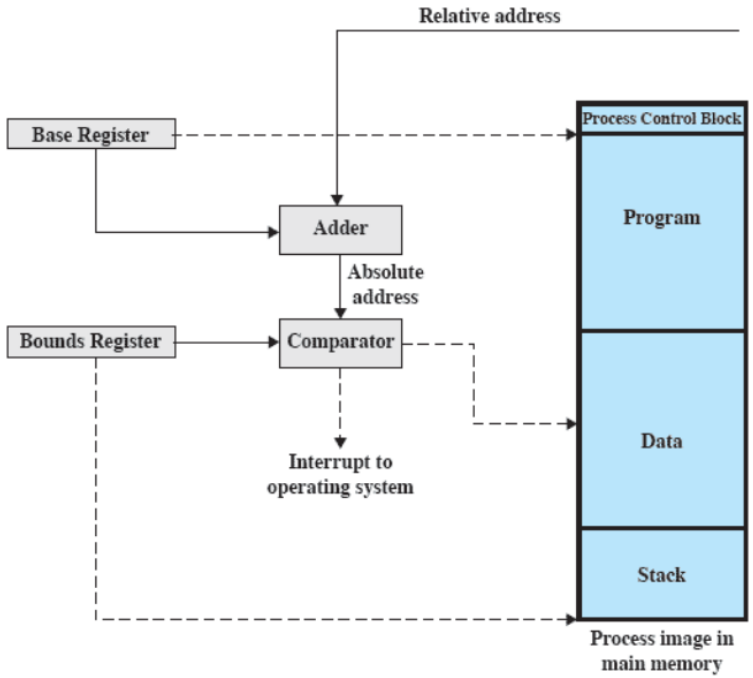


Non si tiene conto della memoria virtuale

Rilocazione: gli Indirizzi nei Programmi

- Il valore del registro base viene aggiunto al valore dell'indirizzo relativo per ottenere l'indirizzo assoluto
- Il risultato è confrontato con il registro limite
- Se va oltre, viene generato un interrupt per il sistema operativo

Non si tiene conto della memoria virtuale



Requisiti: Protezione

- I processi non devono poter accedere a locazioni di memoria di un altro processo, a meno che non siano autorizzati
- A causa della rilocazione, non si può fare a tempo di compilazione
- Quindi bisogna farlo a tempo di esecuzione
- E quindi serve aiuto hardware

Requisiti: Condivisione

- Deve essere possibile permettere a più processi di accedere alla stessa zona di memoria
  - ovviamente, solo se è effettivamente utile allo scopo perseguito dai processi
- Caso tipico: più processi vengono creati eseguendo più volte lo stesso sorgente
  - finché questi processi restano in esecuzione, è più efficiente che condividano il codice sorgente, visto che è lo stesso
- Ci sono anche casi in cui processi diversi vengono esplicitamente programmati per accedere a sezioni di memoria comuni
  - usando chiamate di sistema

Requisiti: Organizzazione Logica

- A livello hardware, la memoria è organizzata in modo **lineare**
  - sia RAM che disco
- A livello software, i programmi sono scritti in moduli
  - i moduli possono essere scritti e compilati separatamente
  - a ciascun modulo possono essere dati diversi permessi (sola lettura, sola esecuzione)
  - i moduli possono essere convidisi tra i processi
- Per facilitare la realizzazione dei punti precedenti, il SO usa la tecnica di gestione di memoria basata sulla segmentazione

Requisiti: Organizzazione Fisica

- Gestione del flusso tra RAM (piccola, veloce e volatile) e memoria secondaria (grande, lenta e permanente)
- Non può essere lasciata al programmatore:
  - la memoria potrebbe non essere sufficiente a contenere il programma ed i suoi dati
    - la tecnica dell'*overlaying* (sovrapposizione) permette a più moduli di essere posizionati nella stessa zona di memoria (in tempi diversi...), ma è difficile da programmare
  - il programmatore non sa quanta memoria avrà a disposizione
- Ci deve pensare il SO

Considerazioni

- La principale operazione di gestione della memoria è **portare i processi in memoria principale** per l'esecuzione da parte del processore
- Nei moderni sistemi multiprogrammati, ciò si ottiene grazie al sofisticato meccanismo di **memoria virtuale**
- La memoria virtuale si basa sulle due tecniche (una o entrambe) di **paginazione** e **segmentazione**
- Prima di esse vediamo tecniche più semplici che facilitano la comprensione della memoria virtuale: **partizionamento**

Gestione della memoria

Partizionamento della memoria

Partizionamento

- Cominciamo dal **partizionamento**:
  - uno dei primi metodi per la gestione della memoria
  - non più molto usato
- I due tipi di partizionamento sono:
  - **Partizionamento fisso**
  - **Partizionamento dinamico**

Partizionamento fisso uniforme

- Partizioni di ugual lunghezza
  - se un processo ha una dimensione minore o uguale della misura di una partizione, allora può essere caricato in una partizione libera
- Il sistema operativo può togliere un processo da una partizione a caricarne un altro (swap)
  - ad esempio, se tutte le partizioni sono occupate e nessuno dei processi attualmente in memoria è in stato *ready*

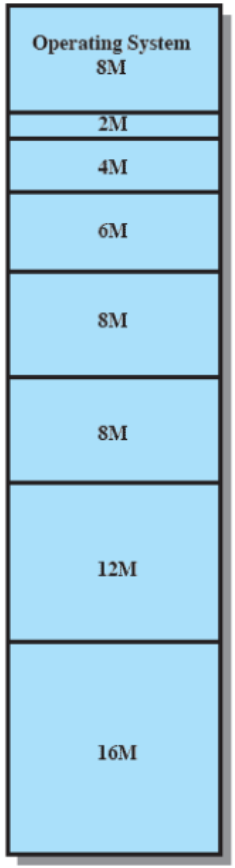
Partizionamento uniforme: problemi

I problemi del partizionamento fisso uniforme sono:

- Un programma potrebbe non entrare in una partizione
  - sta(va) al programmatore dividere il suo programma e usare l'*overlay*
- Uso inefficiente della memoria
  - ogni programma, anche il più piccolo, occupa un'intera partizione
  - problema della *frammentazione interna*

Partizionamento fisso variabile

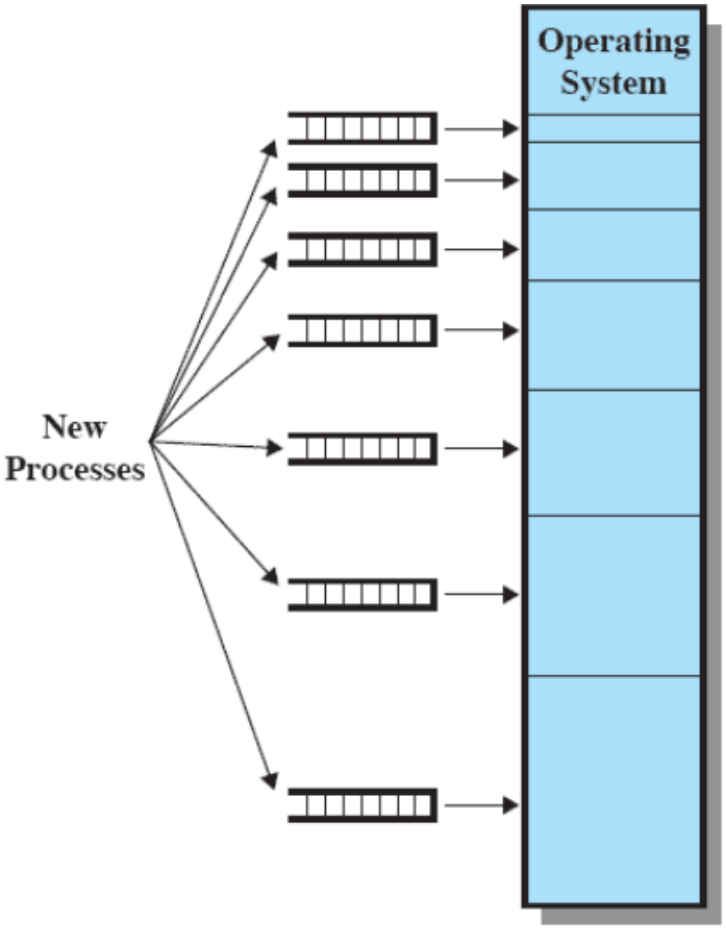
- Mitiga entrambi i problemi
  - ma non li risolve
- Nella figura, programmi più piccoli di 16M possono essere gestiti senza *overlay*
- Per quelli più piccoli, ci sono le partizioni più piccole
- È sempre partizionamento fisso: le partizioni sono quelle decise all'inizio e non cambiano più

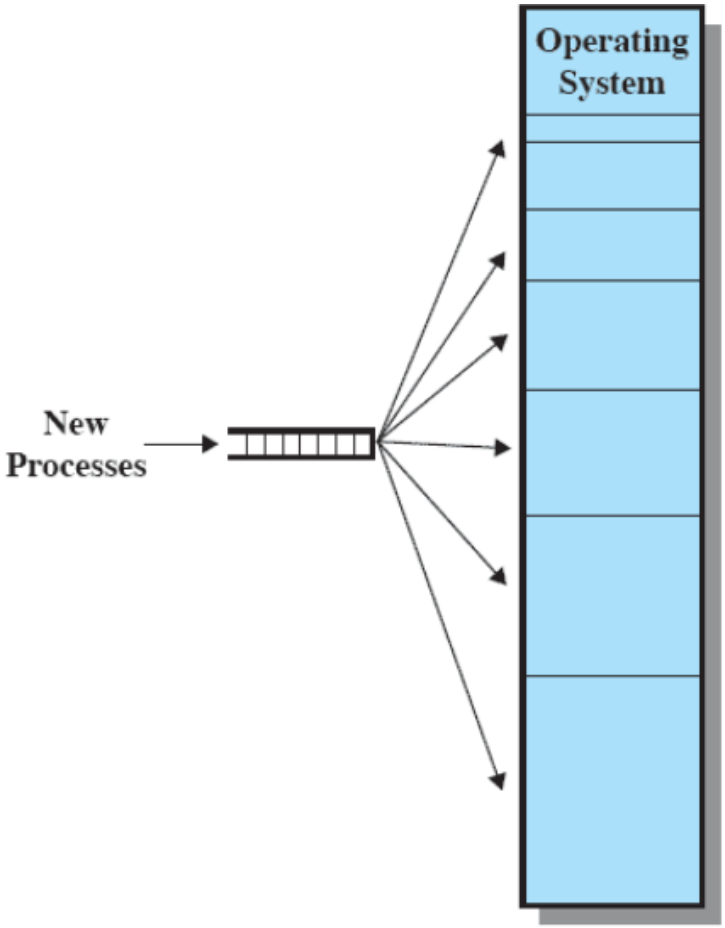


Algoritmo di posizionamento

- Partizioni di uguale lunghezza
  - se ci sono partizioni libere, ogni processo può andare in qualunque partizione, algoritmo banale: in ordine
  - se non ci sono partizioni libere, serve lo *swap* tra processi e la decisione riguarda lo *scheduling*
- Partizioni di diversa lunghezza
  - un processo va nella partizione più piccola che può contenerlo
  - questo minimizza la quantità di spazio sprecato
  - gestione a coda:
    - una coda per ogni partizione, ma potrebbereo rimanere inutilizzate le partizioni più grandi
    - oppure una coda unica per tutte le partizioni

Partizionamento fisso e Code





Partizionamento fisso: problemi irrisolti

L'introduzione del partizionamento fisso variabile non risolve tutti i problemi:

- C'è un numero massimo di processi in memoria principale
  - corrispondente al numero di partizioni deciso inizialmente
- Se ci sono molti processi piccoli, la memoria verrà usata in modo inefficiente
  - sia con le partizioni di lunghezza uguale che con quelle variabili

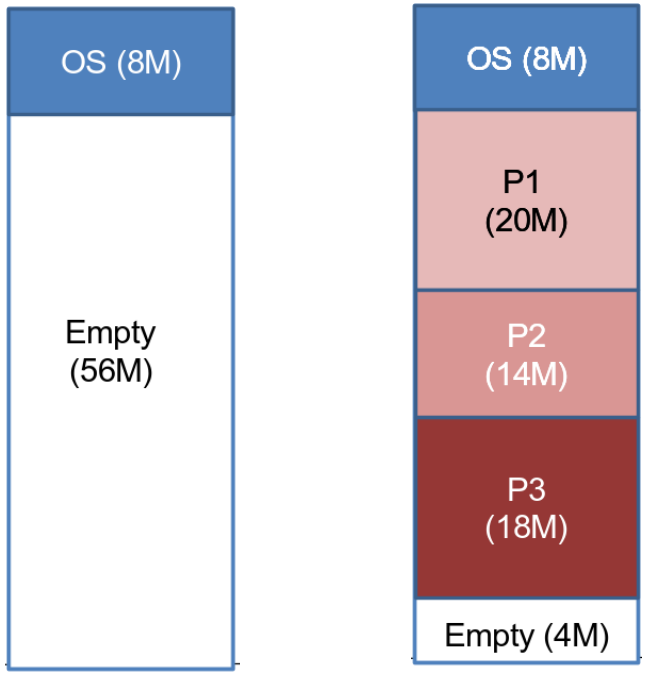
Partizionamento dinamico

- Alcuni problemi del partizionamento fisso vengono superati con il **partizionamento dinamico**
- Si tratta comunque di una tecnica soppiantata da tecniche più sofisticate
- Con il **partizionamento dinamico**:
  - Le partizioni variano sia in misura che in quantità
  - Per ciascun processo viene allocata esattamente la quantità di memoria che serve

Partizionamento dinamico

Esempio

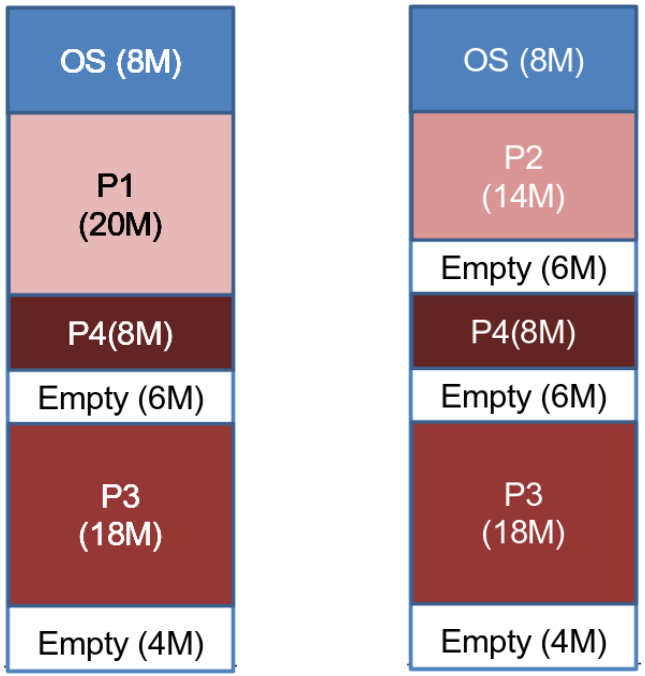
- All'inizio la memoria principale è vuota, eccetto per il SO
- Vengono poi caricati i primi tre processi - P1 di 20M, P2 di 14M, P3 di 18M - a partire da dove finisce il SO
- Resta libera una piccola porzione di memoria (4M), troppo piccola per il processo P4 di 8M
- Ad un certo punto nessuno dei processi in memoria è *ready*



Partizionamento dinamico

Esempio

- Il SO esegue uno swap portando P2 in memoria secondaria, guadagnando spazio a sufficienza per P4
- Si crea così un altro piccolo buco (6M)
- Si possono creare sempre più buchi, ad esempio se si riporta P2 in memoria principale facendo *swap* con P1



Partizionamento dinamico

- Andando avanti la memoria presenta sempre più buchi e l'utilizzazione della memoria è sempre meno efficiente
- Si ha il fenomeno di **frammentazione esterna**: la memoria che non è usata per nessun processo viene frammentata
- Si può risolvere con la **compattazione**
  - il SO sposta i processi in modo che siano contigui
  - ha un elevato overhead
- Si può anche ovviare usando algoritmi di rimpiazzamento sofisticati

Partizionamento dinamico

- Se ci sono più blocchi liberi, il SO deve decidere a quale blocco libero assegnare un processo
- Si usano essenzialmente tre algoritmi di posizionamento: **best-fit**, **first-fit** e **next-fit**

- Algoritmo **best-fit** (miglior blocco tra quelli adatti)
  - sceglie il blocco la cui misura è la più vicina (in eccesso) a quella del processo da posizionare
  - nonostante il nome, è quello con risultati peggiori
  - lascia frammenti molto piccoli
  - costringe a fare spesso la compattazione

Partizionamento dinamico

- Algoritmo **first-fit** (il primo blocco tra quelli adatti)
  - si scorre la memoria dall'inizio
  - si sceglie il primo blocco di memoria abbastanza grande
  - molto veloce
  - conti fatti, è(ra) il migliore
  - tende a riempire solo la prima parte della memoria

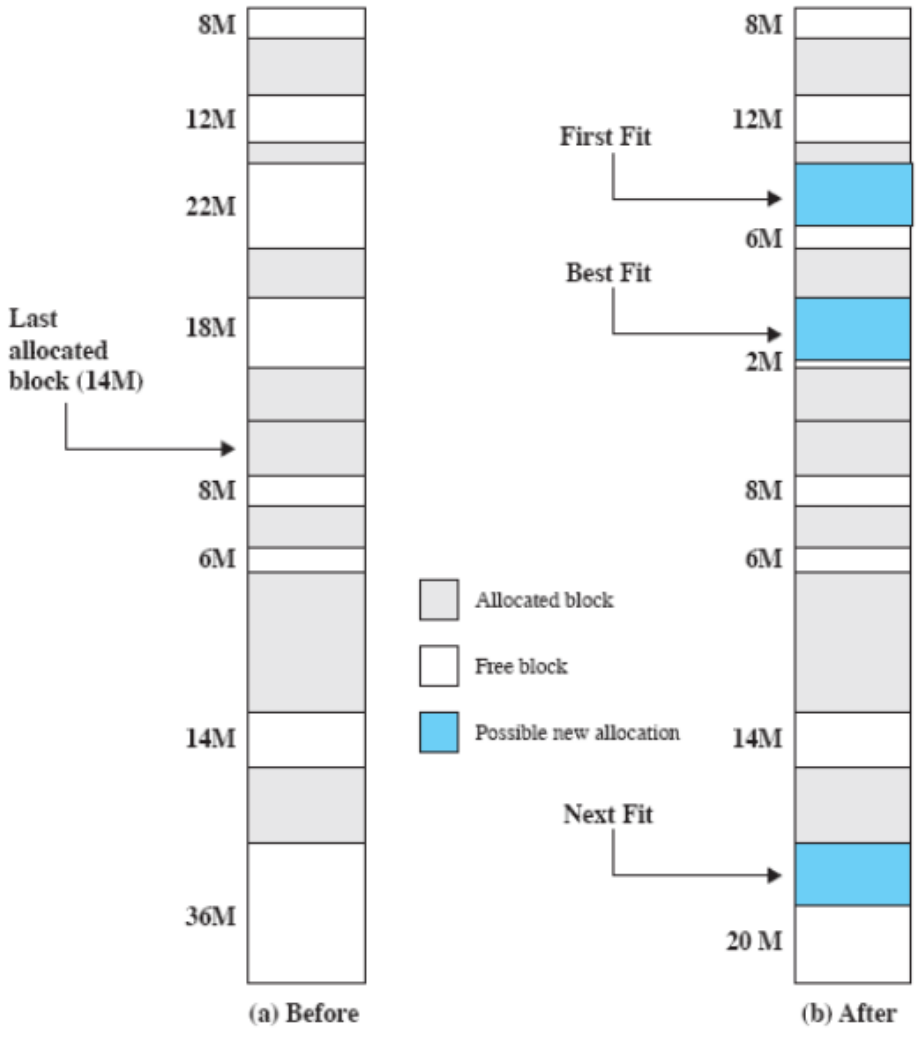
Partizionamento dinamico

- Algoritmo **next-fit** (il successivo tra quelli adatti)
  - come il first-fit, ma invece di partire ogni volta dall'inizio, si parte dall'ultima posizione assegnata ad un processo
  - assegna più spesso il blocco alla fine della memoria, che è quello più grande
  - ma il blocco più grande viene spezzato in blocchi più piccoli
  - occorre usare la compattazione per riottenere un blocco grande alla fine della memoria

Partizionamento dinamico: Esempi di allocazione

La memoria **prima e dopo** l'allocazione di un blocco da 16M con i tre algoritmi

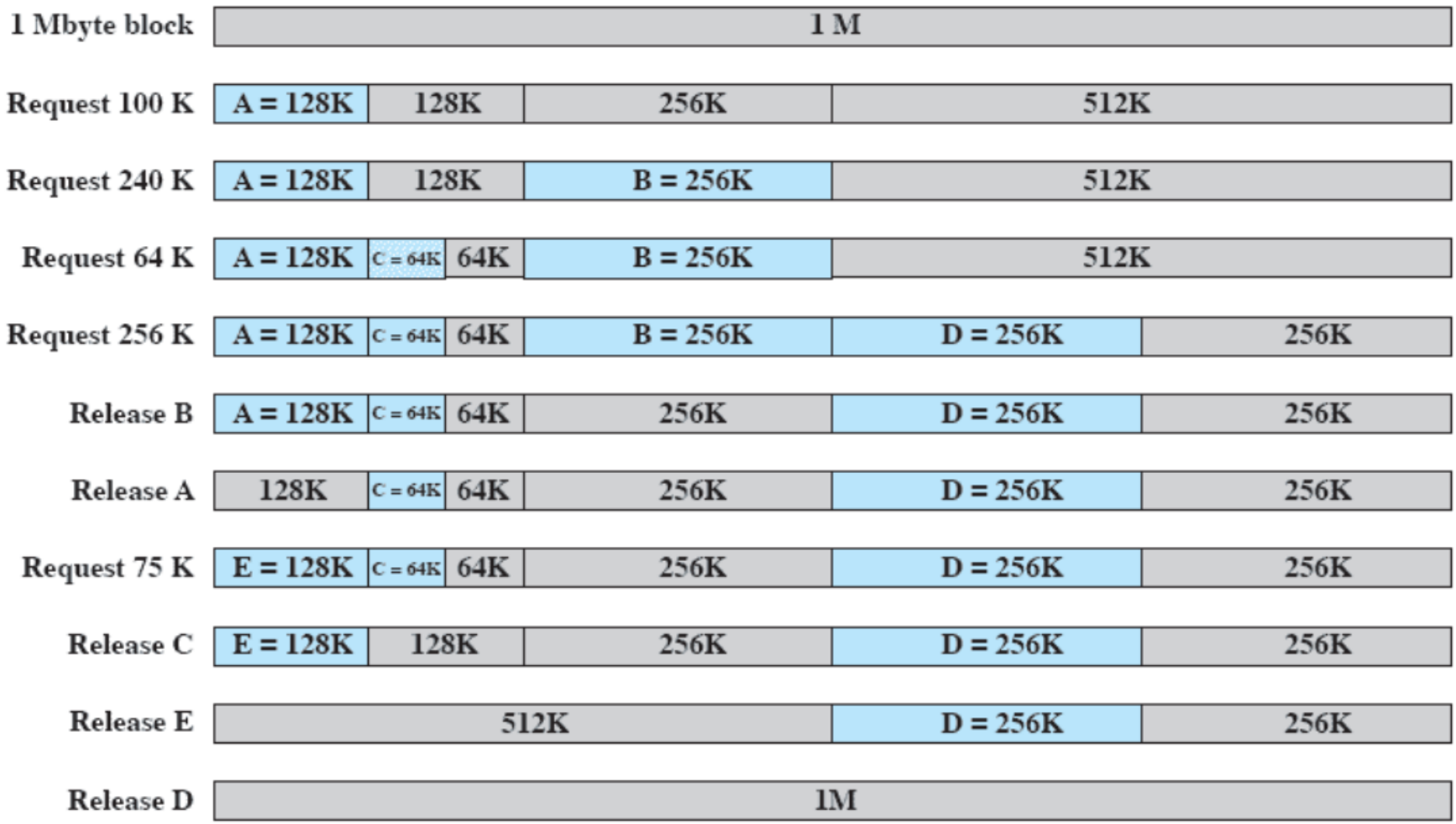
- **best-fit**
- **first-fit**
- **next-fit**



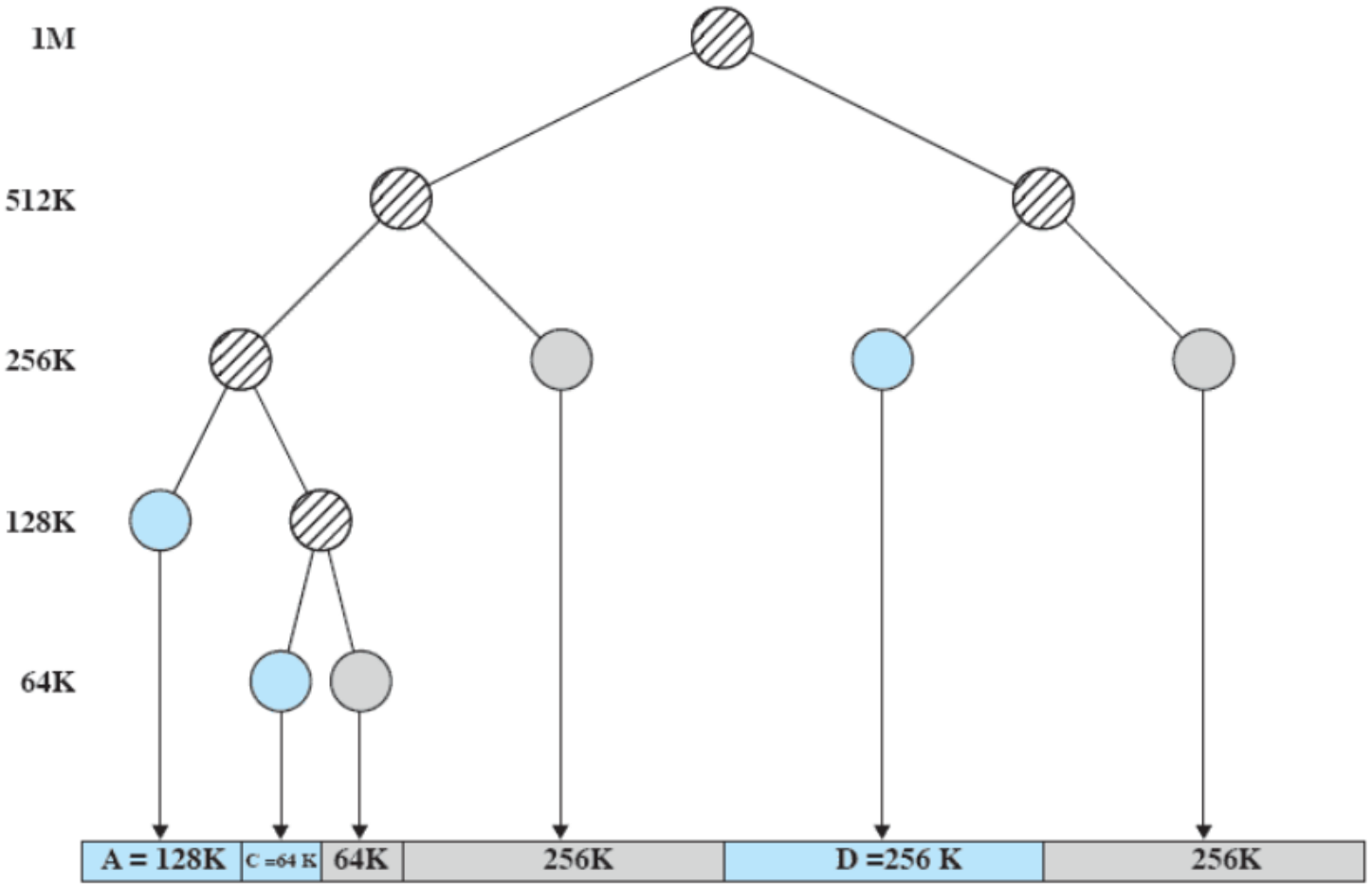
Buddy System (Sistema del Compagno)

- Compromesso tra partizionamento fisso e dinamico
- Siano:
  - 2^U la dimensione del blocco più grande di memoria (all'inizio tutta la memoria disponibile) della memoria
  - 2^L la dimensione del blocco più piccolo di memoria
  - s la dimensione del processo da mettere in RAM
- Si dimezza lo spazio fino a quando si trova un X t.c. $2^{X-1} < s \leq 2^X$, con $L \leq X \leq U$
  - una delle 2 porzioni viene usata per il processo
  - L serve per dare un lower bound e non creare partizioni troppo piccole
- Occorre tenere traccia delle porzioni già occupate
- Quando un processo finisce, se il *buddy* è libero si può fare una fusione

Esempio di Buddy System



Esempio di Buddy System: Rappresentazione ad Albero



Gestione della memoria

Paginazione

Paginazione (Semplice)

- Assumiamo che la memoria venga partizionata in parti piccole di grandezza uguale: **frame**
- Assumiamo che i processi vengano anch'essi partizionati in parti: **pagine**
- Ogni pagina, per essere usata, deve essere collocata in un frame
  - pagine contigue possono essere messe in frame distanti
  - in generale, una pagina può essere messa in un *qualunque* frame
  - ovviamente, una pagina ed un frame hanno la stessa dimensione

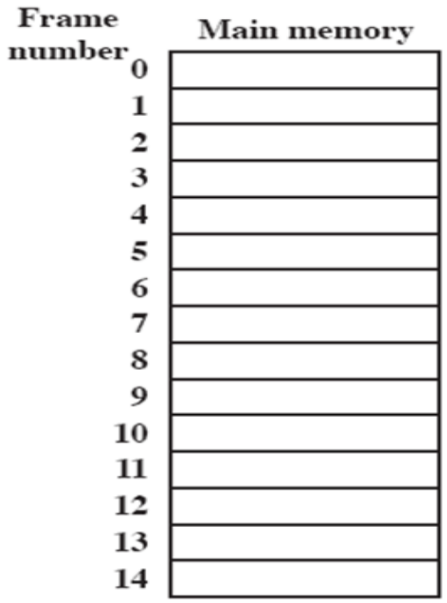
Paginazione

- I SO che adottano la paginazione mantengono una **tabella delle pagine** per ogni processo
- Per ogni pagina del processo, questa tabella specifica in quale **frame effettivo** si trova
- Un indirizzo di memoria può essere visto come un *numero di pagina* e uno *spiazzamento* al suo interno
- Quando c'è un process switch, la tabella delle pagine del nuovo processo deve essere ricaricata

Paginazione

Esempio

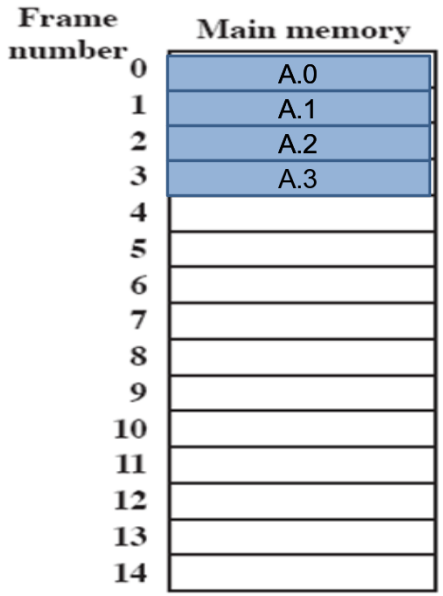
- Il SO deve sempre tenere aggiornata la lista dei frame liberi
- Quando è il momento di caricare un processo, il SO cerca il numero di frame liberi per caricare quel processo
- All'inizio tutti i frame sono liberi



Paginazione

Esempio

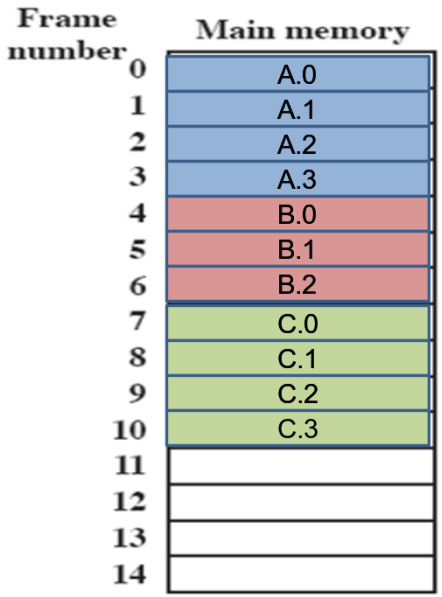
- Il processo A, memorizzato sul disco rigido, consiste di 4 pagine
- Quando è il momento di caricare il processo, il SO cerca 4 frame liberi
- Le pagine vengono caricate in memoria nei primi 4 frame



Paginazione

Esempio

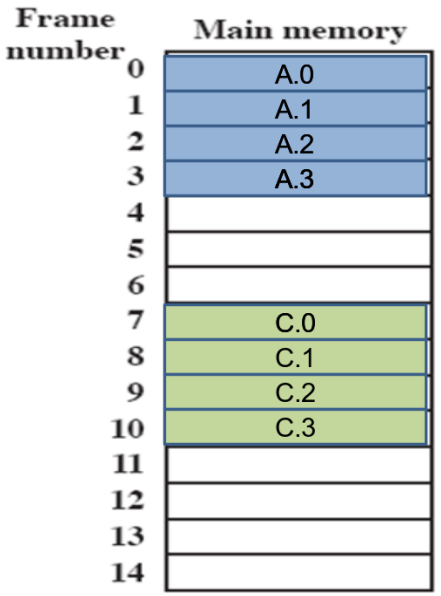
- Successivamente vengono caricati il processo B, che consiste di 3 pagine, e poi il processo C, che consiste di 4 pagine
- Ad un certo punto tutti i processi sono bloccati e il SO vuole caricare un nuovo processo



Paginazione

Esempio

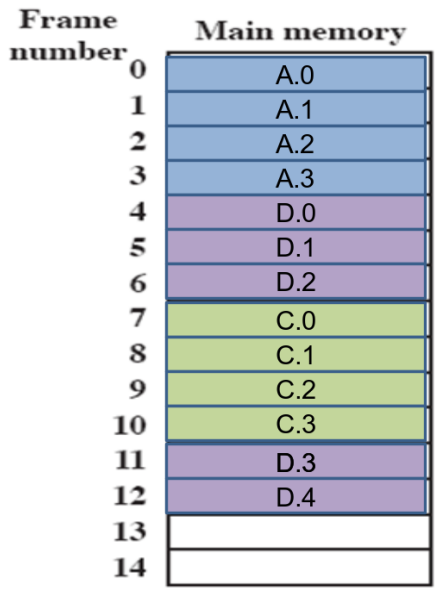
- Il processo B (bloccato) viene scelto per essere swappato in memoria secondaria e viene portato nello stato Suspended
- Il SO vuole poi portare in memoria principale il processo D, che consiste di 5 pagine
- Non ci sono però 5 frame liberi contigui



Paginazione

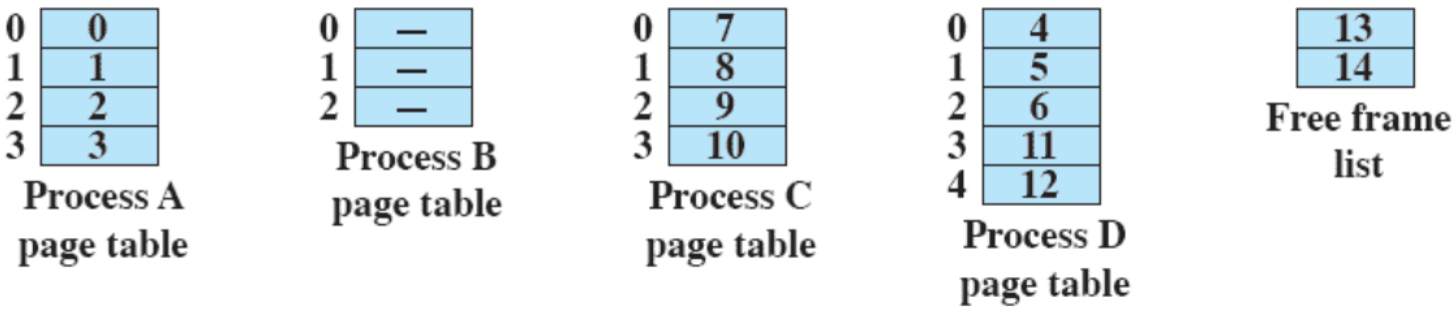
Esempio

- Le cinque pagine del processo D vengono caricate nei frame 4-5-6 e 11-12
- Per supportare questa organizzazione serve una **tabella delle pagine** per ogni processo per ricordare in quali dei frame sono allocate le diverse pagine
- Con il partizionamento dinamico, non sarebbe stato possibile caricare D in memoria



Paginazione: Esempio

Tabelle delle pagine per i processi attivi (non suspended)



Paginazione

- Per ottenere l'indirizzo fisico non basta avere solo base register, ma si usa la tabella delle pagine
- La traduzione da indirizzo logico a indirizzo fisico è fatta con il supporto dell'hardware.
- Il processore usa l'informazione riguardante il frame in cui collocata la pagina presente nella tabella delle pagine
- L'indirizzo logico (page number-offset) viene trasformato in indirizzo fisico (frame number-offset)

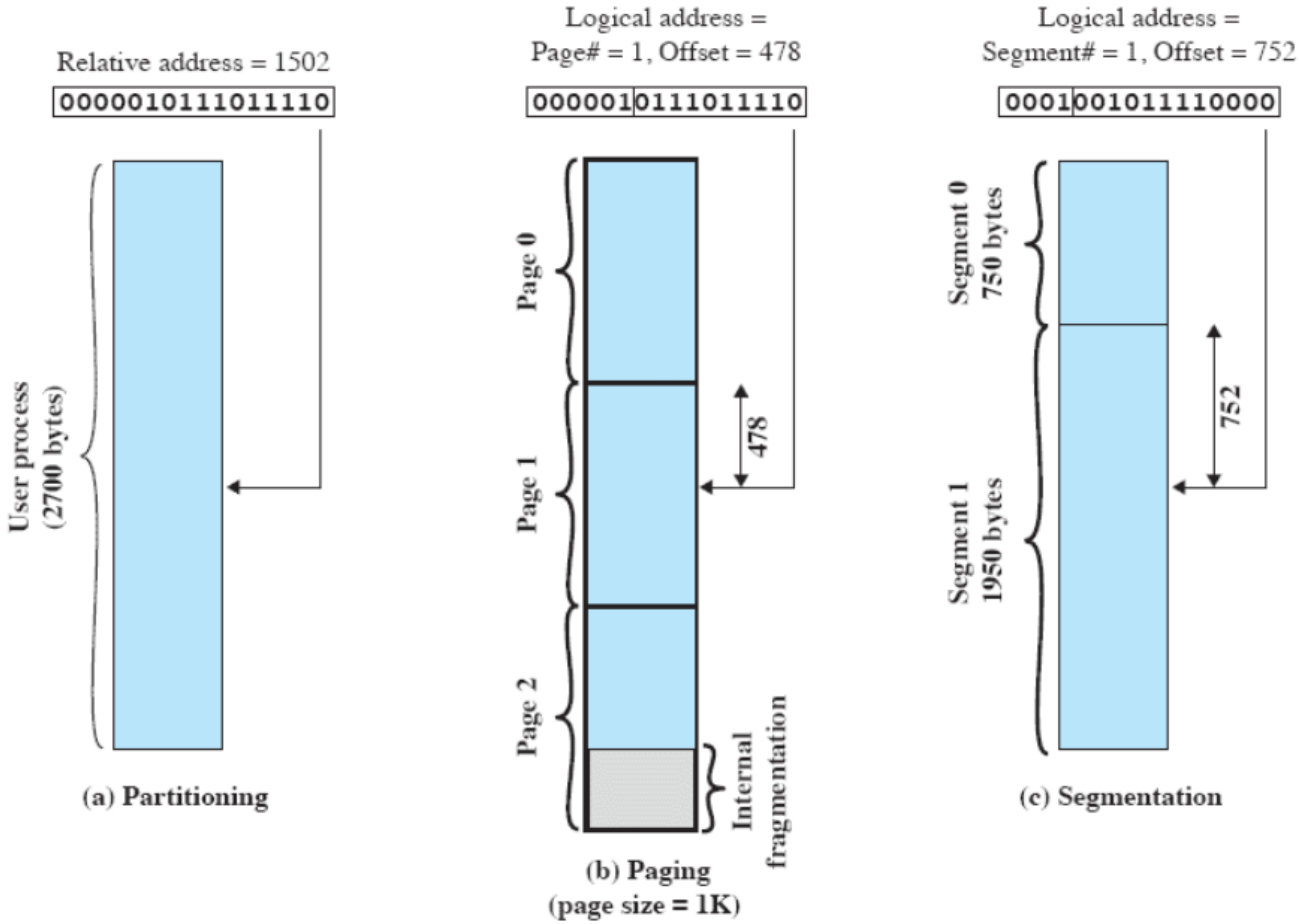
Gestione della memoria

Segmentazione

Segmentazione (Semplice)

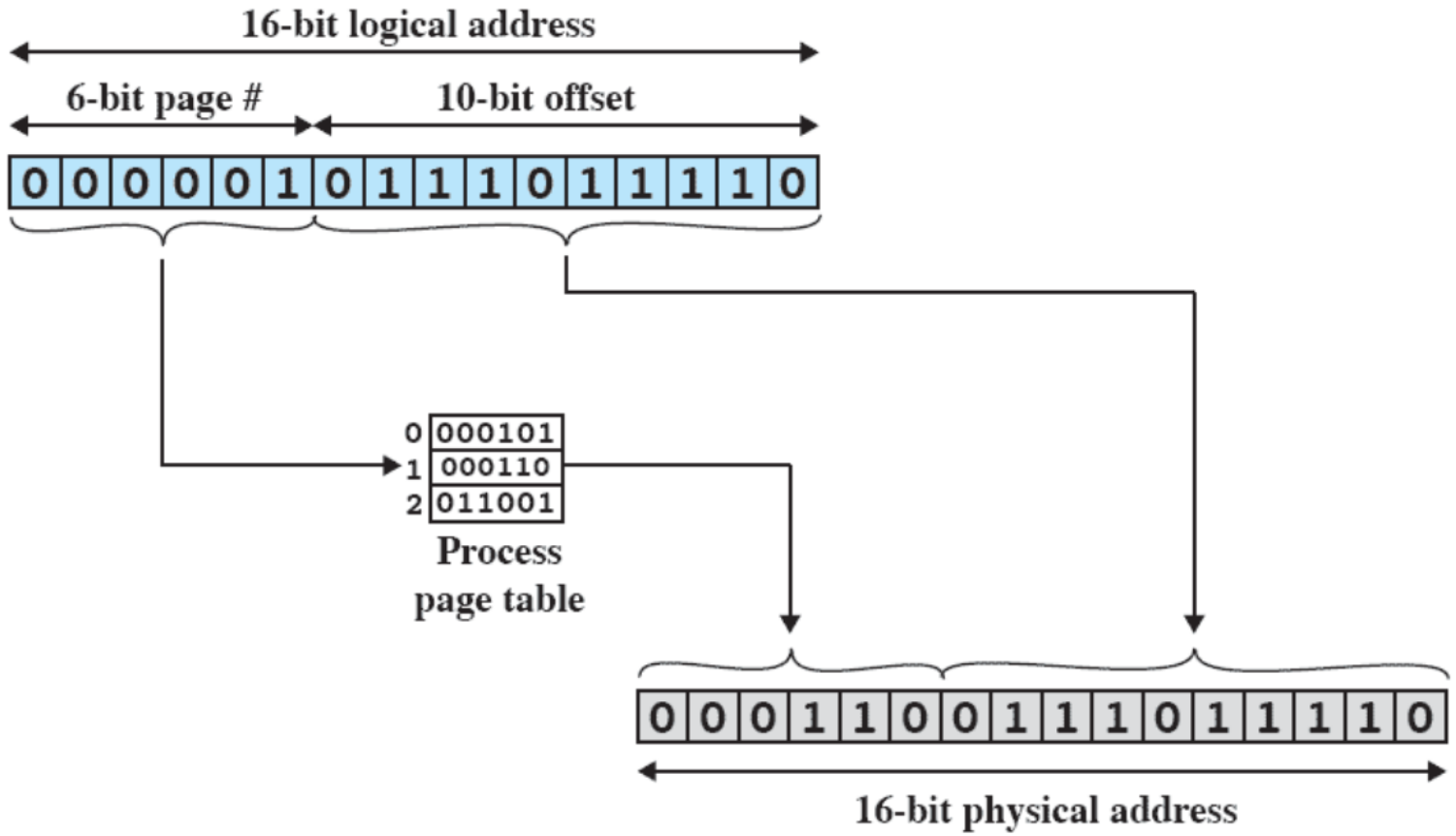
- I programmi vengono divisi in **segmenti**:
  - di dimensione (lunghezza) variabile
  - con un limite massimo alla dimensione
- Simile al partizionamento dinamico
  - ma con una differenza fondamentale: il programmatore o il compilatore devono gestire esplicitamente la segmentazione
  - cioè dire quanti segmenti ci sono e qual è la loro dimensione
  - e metterli effettivamente in RAM
  - invece a *risolvere gli indirizzi* ci pensa il SO, con supporto hardware
- Un indirizzo di memoria è un numero di segmento e uno spiazzamento al suo interno

Indirizzi Logici



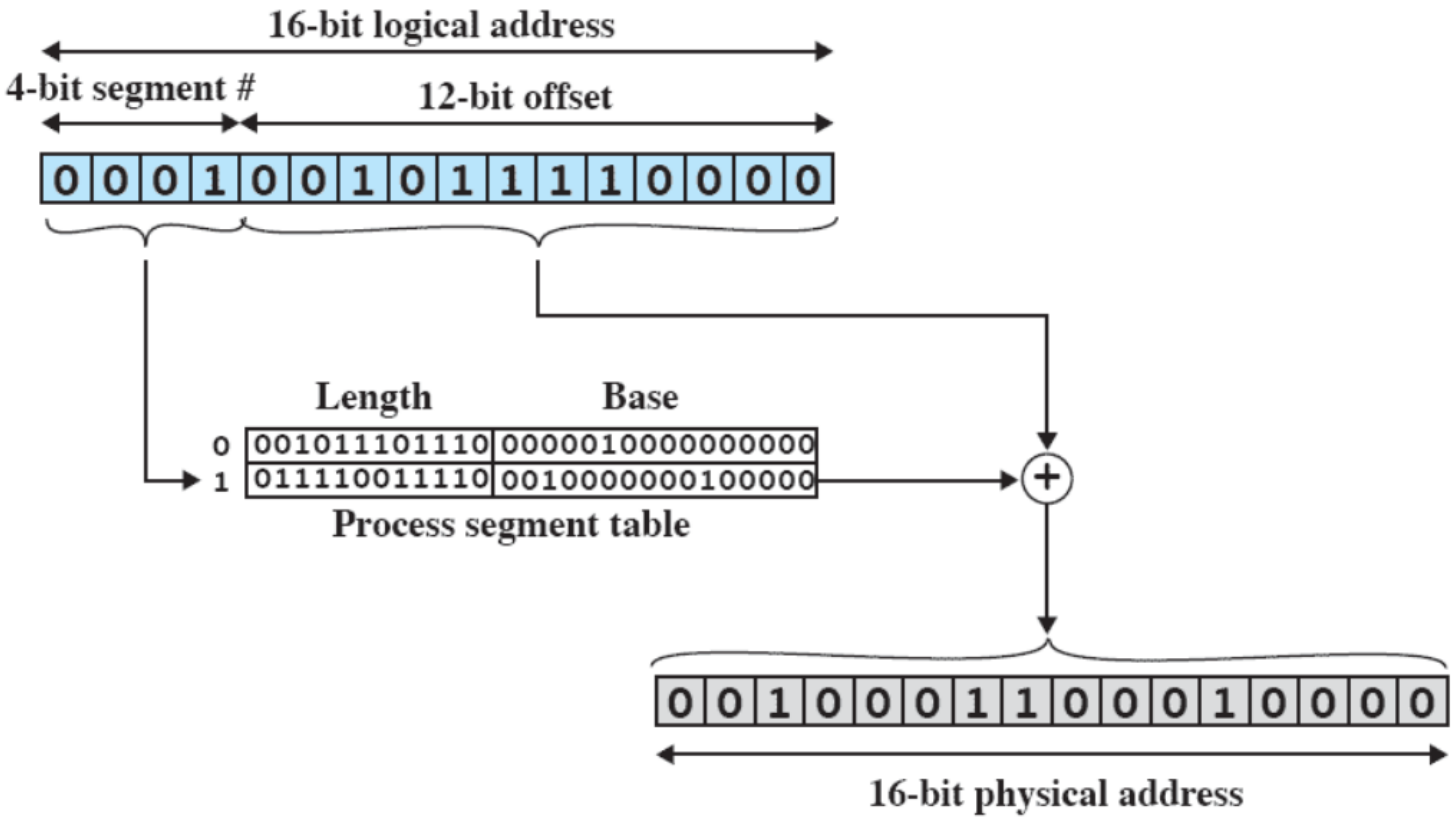
Paginazione

Per ogni processo, il numero di pagine è al più il numero di frames (non sarà così con la memoria virtuale)



Segmentazione

Con la segmentazione le cose sono leggermente diverse
Si usa la tabella dei segmenti (analoga alla tabella della pagine)



(b) Segmentation

Memoria virtuale

Memoria virtuale: concetti fondamentali

Gestione della Memoria: concetti fondamentali

- Il confronto tra partizionamento fisso e dinamico con **paginazione** e **segmentazione**, danno l'intuizione della svolta nella gestione della memoria che ha portato alla **memoria virtuale**
- Due caratteristiche di **paginazione** e **segmentazione** sono la chiave della svolta

Gestione della Memoria: concetti fondamentali

1. Tutti i riferimenti di memoria in un processo sono *indirizzi logici* tradotti in indirizzi fisici a tempo di esecuzione
   - così un processo può essere spostato più volte dalla memoria principale alla secondaria e viceversa durante l'esecuzione, occupando ogni volta zone di memoria diverse
2. Un processo può essere spezzato in *più parti* (pagine o segmenti), che non necessariamente occuperanno una zona contigua di memoria principale
   - si sfrutta la traduzione dinamica dell'indirizzo e la tabella della pagine o dei segmenti

La svolta: idea chiave

L'**idea chiave** della svolta è basata sulle seguenti osservazioni:

- Non occorre che tutte le pagine o tutti i segmenti di un processo siano in memoria principale durante l'esecuzione (e il processo venga concesso il processore)
- Se la successiva istruzione da eseguire e i dati su cui eseguirla sono in memoria principale, allora l'esecuzione può andare avanti (almeno per un po')

Memoria Virtuale: Terminologia

Memoria virtuale: schema di allocazione di memoria, in cui la memoria secondaria può essere usata come se fosse principale

- gli indirizzi usati nei programmi e quelli usati dal sistema sono diversi
- c'è una fase di traduzione automatica dai primi nei secondi
- la dimensione della memoria virtuale è limitata dallo schema di indirizzamento, oltre che ovviamente dalla dimensione della memoria secondaria
- la dimensione della memoria principale, invece, non influisce sulla dimensione della memoria virtuale

Memoria Virtuale: Terminologia

Indirizzo virtuale: l'indirizzo associato ad una locazione della memoria virtuale, alla quale si accede come se fosse parte della memoria principale

Spazio degli indirizzi virtuali: la quantità di memoria virtuale assegnata ad un processo

Spazio degli indirizzi: la quantità di memoria assegnata ad un processo

Indirizzo reale: indirizzo di una locazione di memoria principale

Come realizzare la memoria virtuale

- Il SO porta in memoria principale alcuni **pezzi** (pagine per *paging* o segmenti per *segmentation*) del programma
- All'inizio vengono portati solo (uno o) pochi pezzi, cioè pezzo iniziale di programma e pezzo iniziale di dati
- La porzione di processo in memoria principale viene chiamato **resident set** (*insieme residente*)
- Se il processore trova un indirizzo logico che non è residente in memoria principale, genera un interrupt per *memory access fault*

Come realizzare la memoria virtuale

- Il SO mette il processo in modalità blocked: è una richiesta di I/O a tutti gli effetti
- Affinchè il processo possa riprendere l'esecuzione, il SO deve portare in memoria principale il pezzo di programma contenente l'indirizzo logico che ha causato l'interruzione
- Vengono eseguite le seguenti operazioni:
  - SO esegue richiesta di lettura su disco (I/O)
  - Un altro processo viene portato in esecuzione
  - Quando il pezzo mancante è stato portato in memoria principale, il controllo viene ridato al SO tramite un'interruzione
  - Il SO porta il processo blocked a ready

Vantaggi per il sistema

1. Più processi possono essere in memoria principale
   - Solo alcune parti di ciascun processo vengono portete in memoria principale
   - Questo vuol dire che è molto probabile che ci sia sempre almeno un processo ready
   - Uso più efficiente del processore
2. Un processo potrebbe anche richiedere più dell'intera memoria principale
   - Una delle principali complicazioni per il programmatore (conoscere la dimensione della memoria e dividere il proprio programma) viene eliminata
   - Con la memoria virtuale basata su paginazione o segmentazione, se ne occupa il sistema operativo con il supporto dell'hardware
   - Il programmatore vede la memoria grande come il disco rigido

Memoria Reale e Virtuale

- **Memoria reale**: è la memoria principale (la RAM)
- **Memoria virtuale**: é quella percepita dal programmatore e corrisponde alla memoria secondaria (cioè al disco rigido)
  - permette di avere una multiprogrammazione elevata
  - libera il programmatore dai vincoli della memoria (principale)

Problemi: thrashing

- La memoria virtuale basata su paginazione oppure su paginazione + segmentazione è diventata una componente fondamentale dei moderni SO
- Però è stata oggetto di molte discussioni in passato
- **Esempio**:
  - abbiamo un programma molto grande che ha bisogno di un grande numero di array di dati di grandi dimensioni
  - se c'è un salto a un'istruzione o servono dati non presenti in memoria principale viene generata un'interruzione per *page o segmentation fault*
  - salti e riferimenti a porzioni diverse di dati sono molto frequenti
- Se la memoria principale è piena e ci sono molti processi attivi, ogni volta che c'è un memory fault il SO deve gestire lo swap di processi

Problemi: thrashing

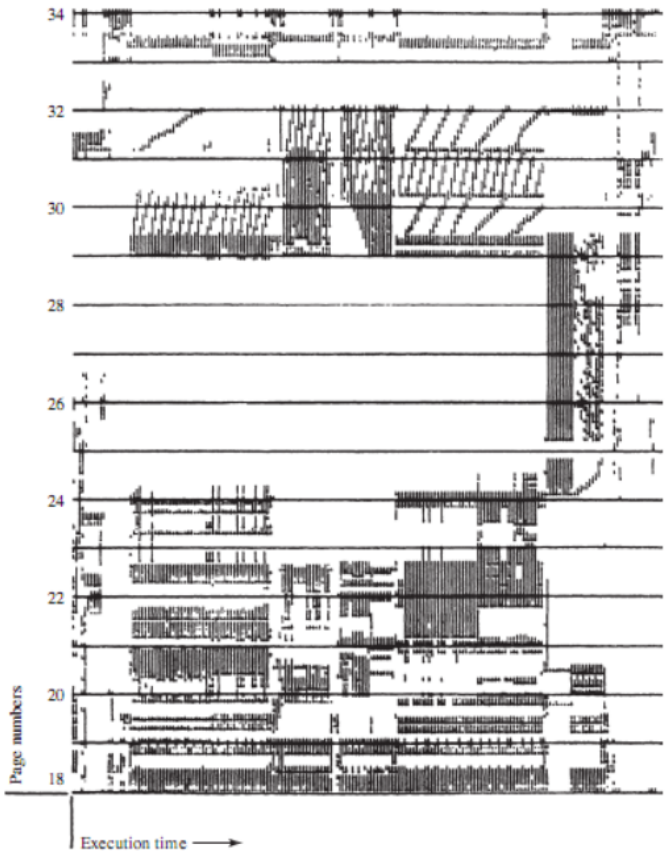
- Il rischio è incorrere nel fenomeno del **trashing**: il SO impiega la maggior parte del suo tempo a swappare pezzi di processi, anziché eseguire istruzioni
- Per evitarlo, o almeno minimizzarlo, il SO cerca di indovinare quali pezzi di processo saranno usati con minore o maggiore probabilità nell'immediato futuro
  - ovvero, quale sarà la prossima istruzione da eseguire o i prossimi dati richiesti
- Questo tentativo di *divinazione* avviene sulla base della storia recente

Principio di Località

- A tale scopo si usa il **principio di località**
- I riferimenti che un processo fa tendono ad essere vicini
  - sia che si tratti di dati che di istruzioni
- Quindi solo pochi pezzi di processo saranno necessari di volta in volta
- Quindi si può prevedere abbastanza bene quali pezzi di processo saranno necessari nel prossimo futuro
- Concludendo, la memoria virtuale può funzionare (e funziona) bene

Pagine e Località: Esempio

Di volta in volta, i riferimenti sono confinati ad un sottoinsieme delle pagine



Gestione della memoria

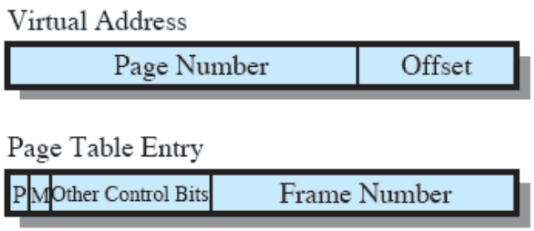
Memoria virtuale: supporto hardware

Memoria Virtuale: Supporto Richiesto

- Paginazione e segmentazione devono essere supportati dall'hardware
  - alcune operazioni sarebbero troppo lunghe se fatte in software dal SO
  - in particolare, la traduzione degli indirizzi è hardware
- Il SO deve essere in grado di muovere pagine e/o segmenti dalla memoria principale alla secondaria

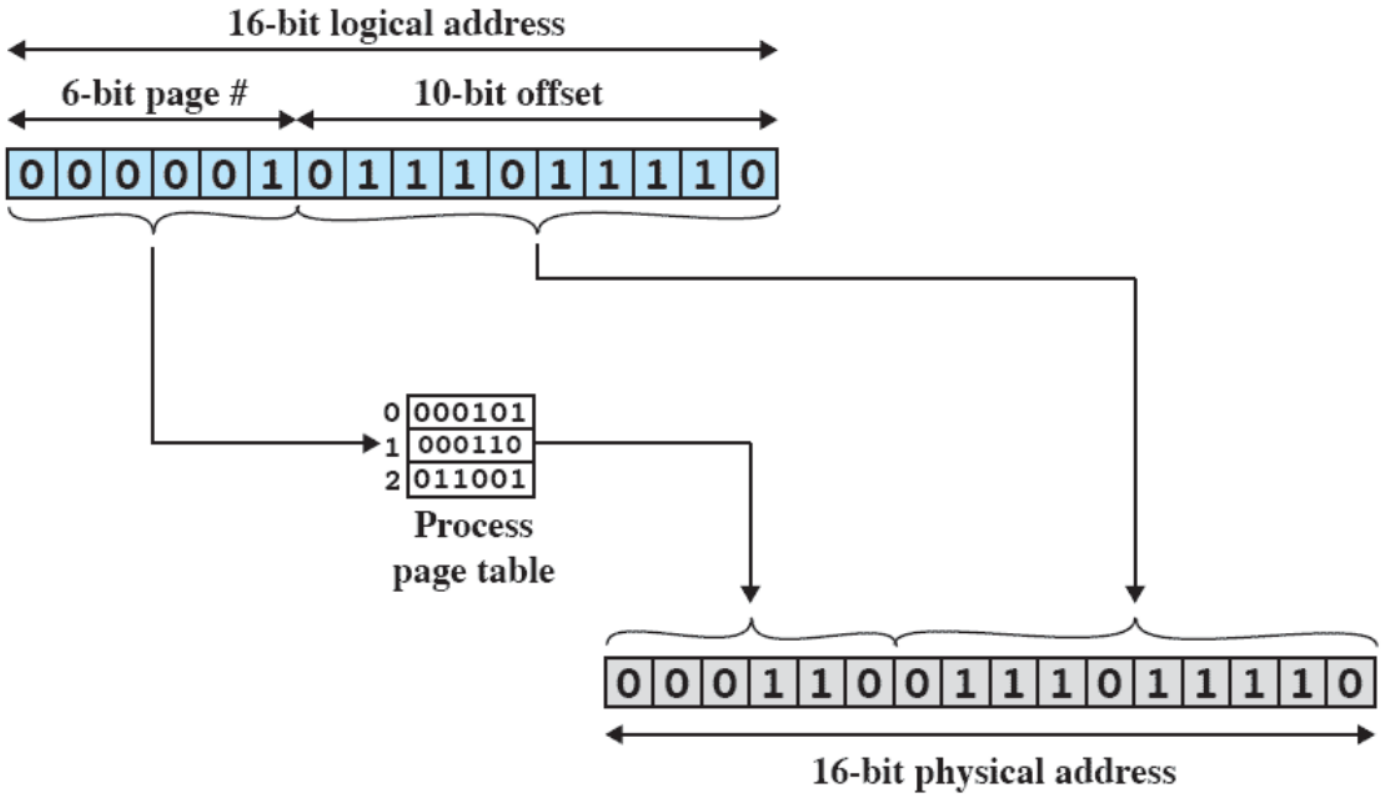
Paginazione

- Ogni processo ha una sua tabella delle pagine
  - il control block di un processo punta a tale tabella
- Ogni riga di questa tabella contiene:
  - il numero di frame in memoria principale
  - non c'è il numero di pagina: è direttamente usato per indicizzare la tabella
  - un bit per indicare se è in memoria principale o no (**P**)
  - un altro bit per indicare se la pagina è stata modificata in seguito all'ultima volta che è stata caricata in memoria principale (**M**)



Traduzione degli Indirizzi

Tipicamente ci sono più pagine che frames, quindi non è realistico lo stesso numero di bit (nell'esempio qui sotto basta pensare che le righe della tabella delle pagine contengono più bit)



Traduzione degli Indirizzi

Realizzazione hardware

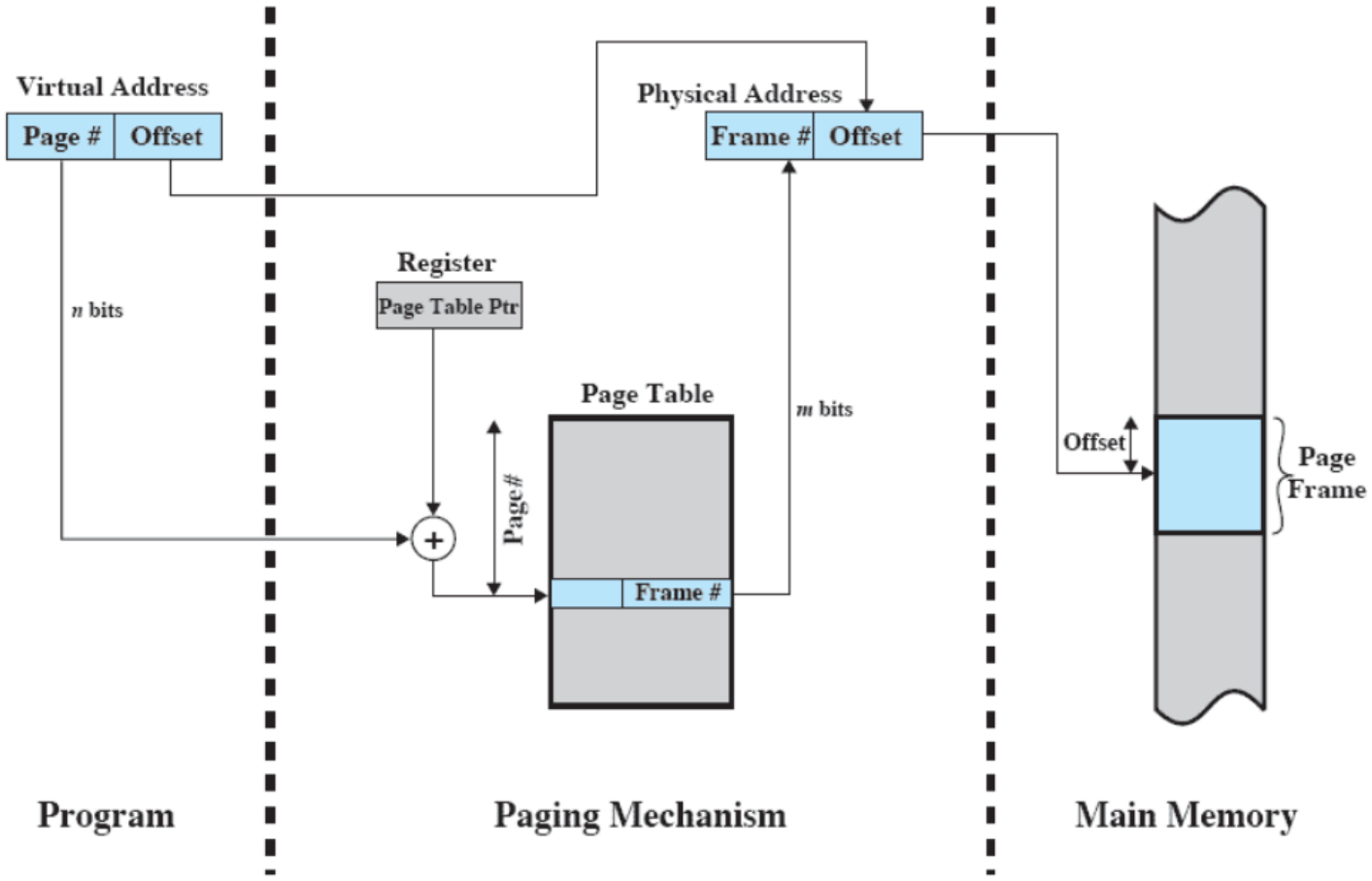


Tabelle delle Pagine

- 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**: 8GB di spazio virtuale, 1kB per ogni pagina
  - Per ogni tabella delle pagine, ovvero per ogni processo, si hanno $\frac{2^{30+3}}{2^{10}} = 2^{23}$ righe
  - Ogni riga occupa: 1 byte di controllo + $\log_2$(size RAM in frames) byte
  - Se la RAM è da 4GB (architettura a 32-bit) si hanno 4 bytes
    - cioè 32 bit - 10 bit = 22 bit per i frame, quindi 3 bytes, più il byte di controllo
  - Quindi c'è un overhead di $4 \cdot 2^{23} = 2^{23+2} = 32$MB per ogni tabella delle pagine, cioè per ogni processo
  - 30 processi occupano circa 1GB di RAM (cioè un quarto) per le sole strutture di overhead

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)

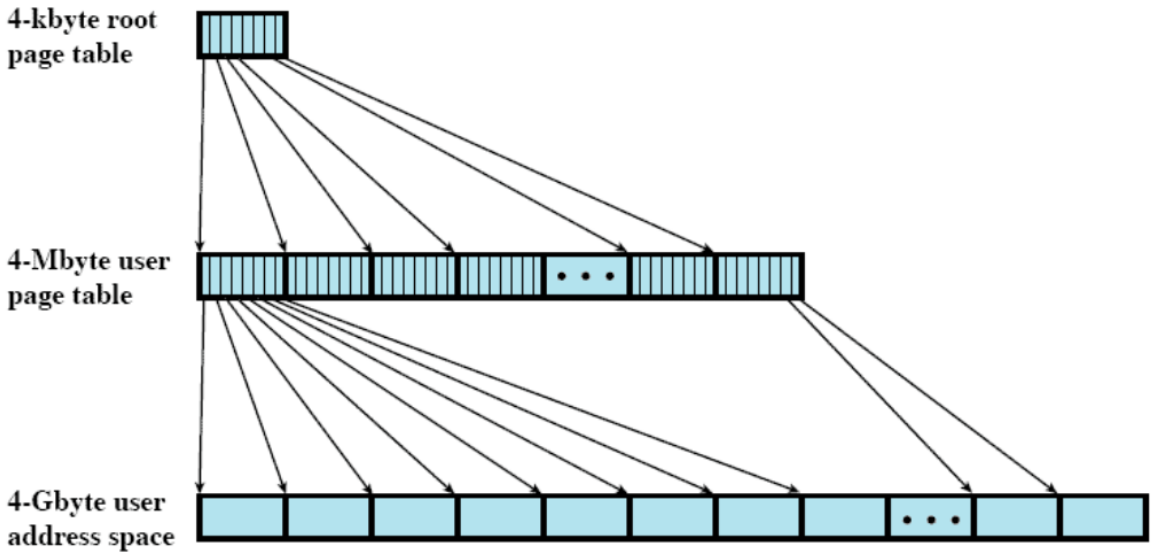
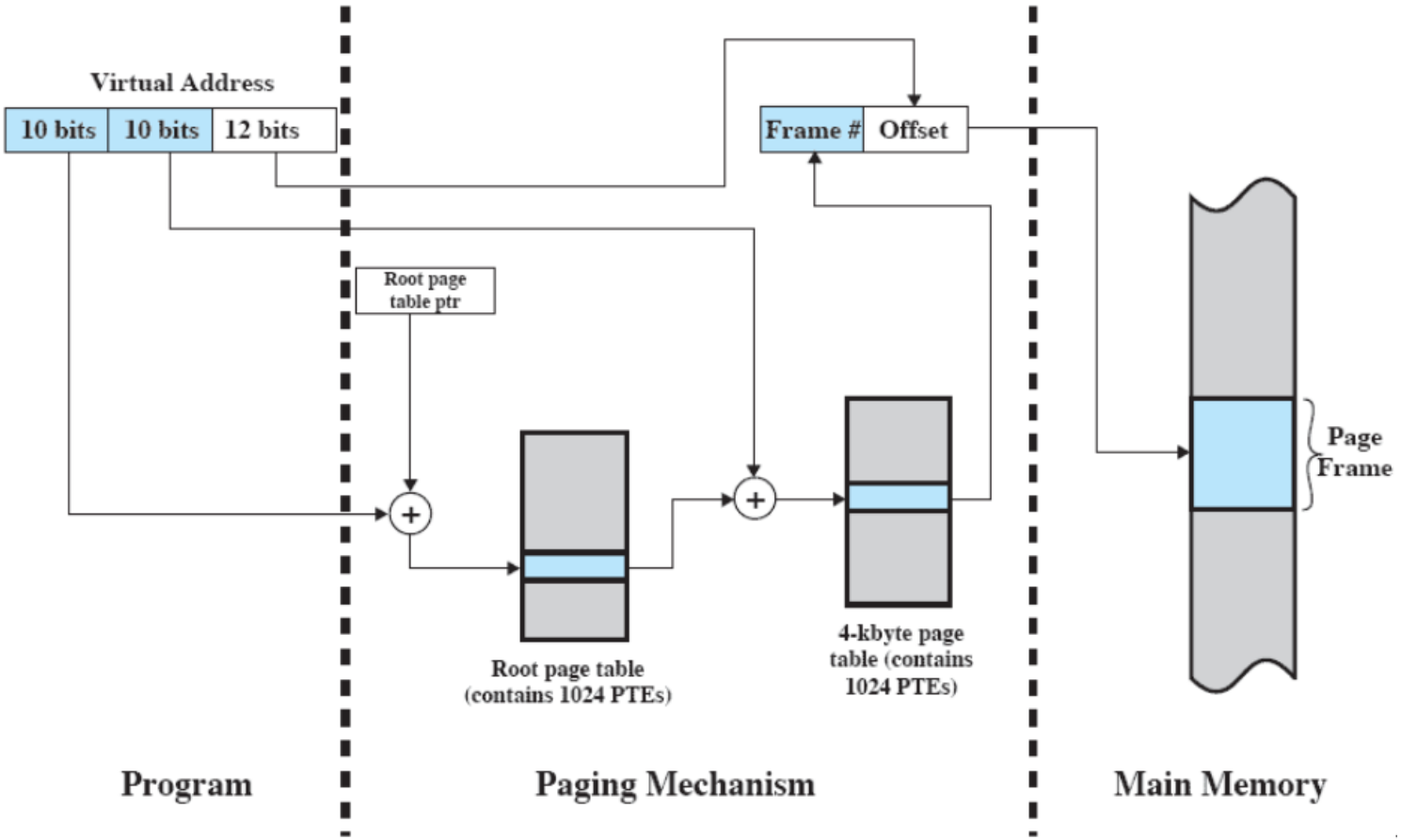


Tabelle delle Pagine a 2 Livelli

- **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



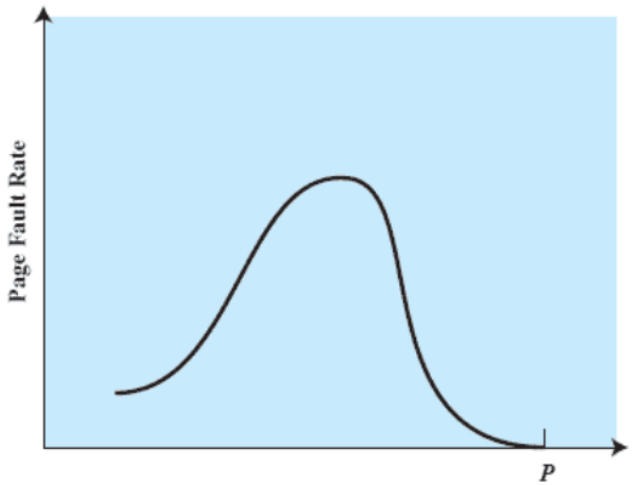
Dimensione delle Pagine

- 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

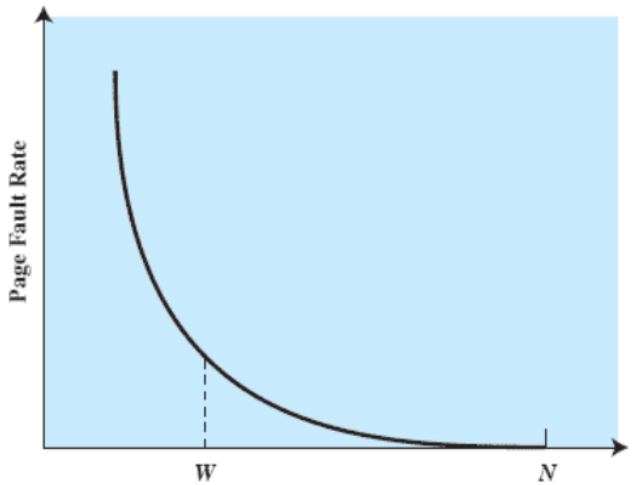
Dimensione delle Pagine

- 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

Page Faults vs. Dimensione Pagine



(a) Page Size



(b) Number of Page Frames Allocated

P = size of entire process
W = working set size
N = total number of pages in process

Con pagine grandi, pochi fault di pagina, ma poca multiprogrammazione!

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)

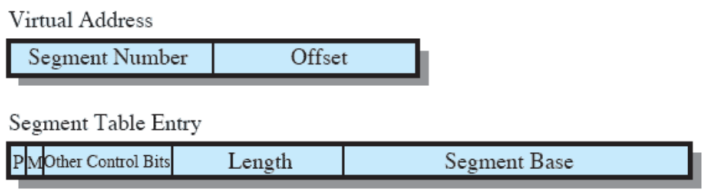
Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbyes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
IBM POWER	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

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
- 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

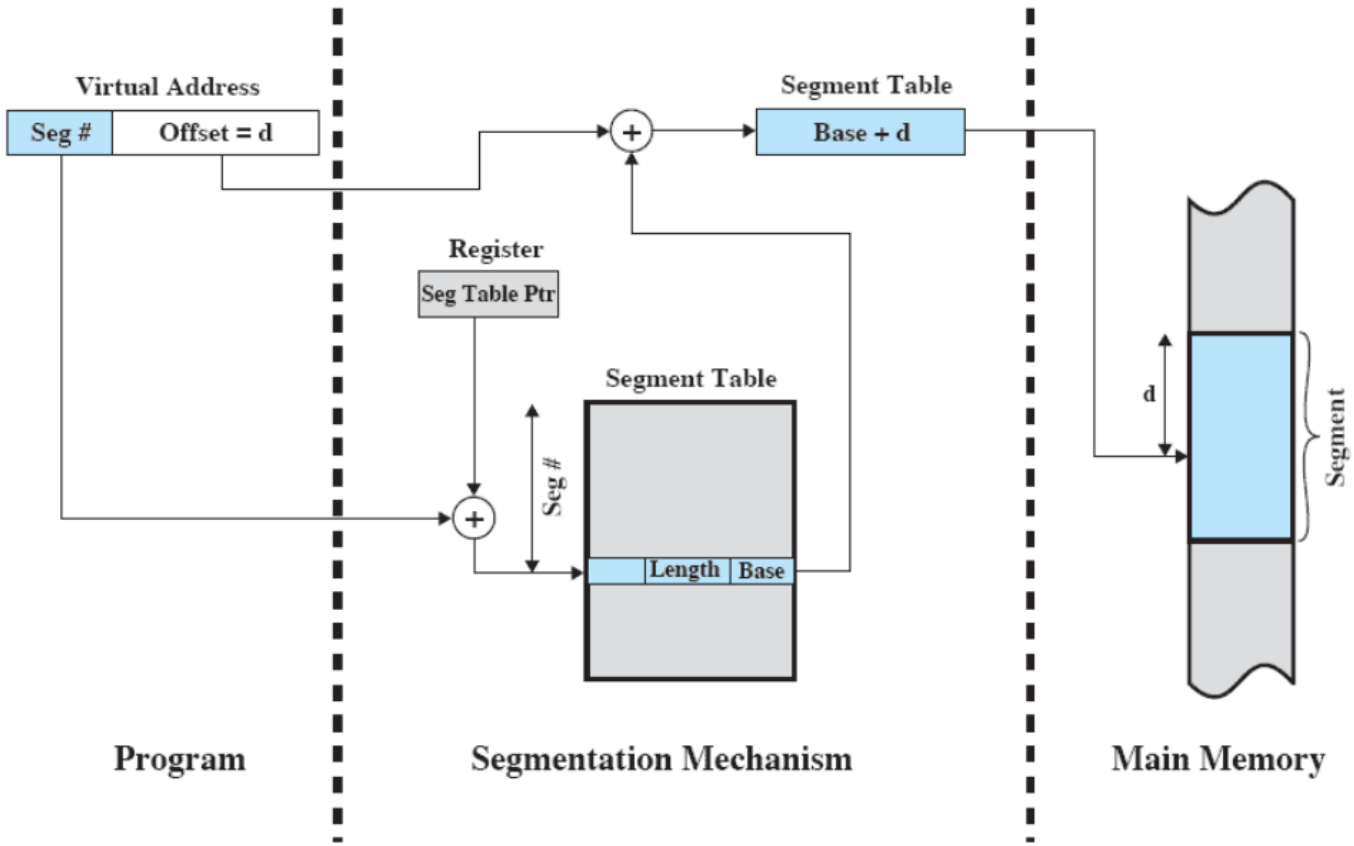
Segmentazione: Organizzazione

- Ogni processo ha una sua tabella dei segmenti
  - il control block di un processo punta a tale tabella
- Ogni riga di questa tabella contiene:
  - l'indirizzo di partenza (in memoria principale) del segmento
  - la lunghezza del segmento
  - un bit per indicare se il segmento è in memoria principale o no
  - un altro bit per indicare se il segmento è stato modificato in seguito all'ultima volta che è stato caricato in memoria principale



Segmentazione: Traduzione degli Indirizzi

Realizzazione hardware



Paginazione + Segmentazione

- 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
- Se ogni segmento viene diviso in più pagine si ha **paginazione + segmentazione**

Paginazione + Segmentazione

Virtual Address

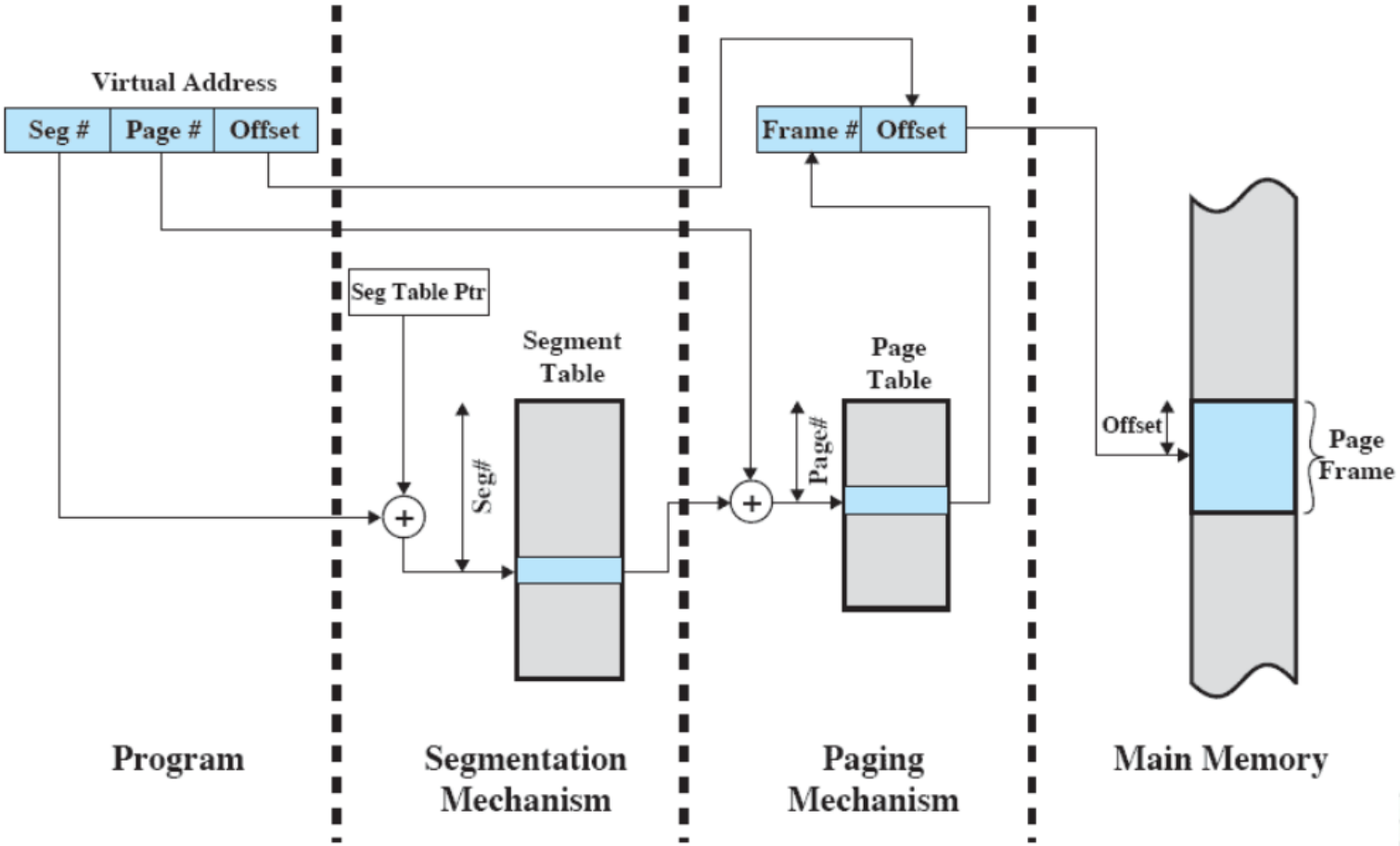
Segment Table Entry

Page Table Entry

P= present bit
M = Modified bit

Paginazione + Segmentazione: Traduzione degli Indirizzi

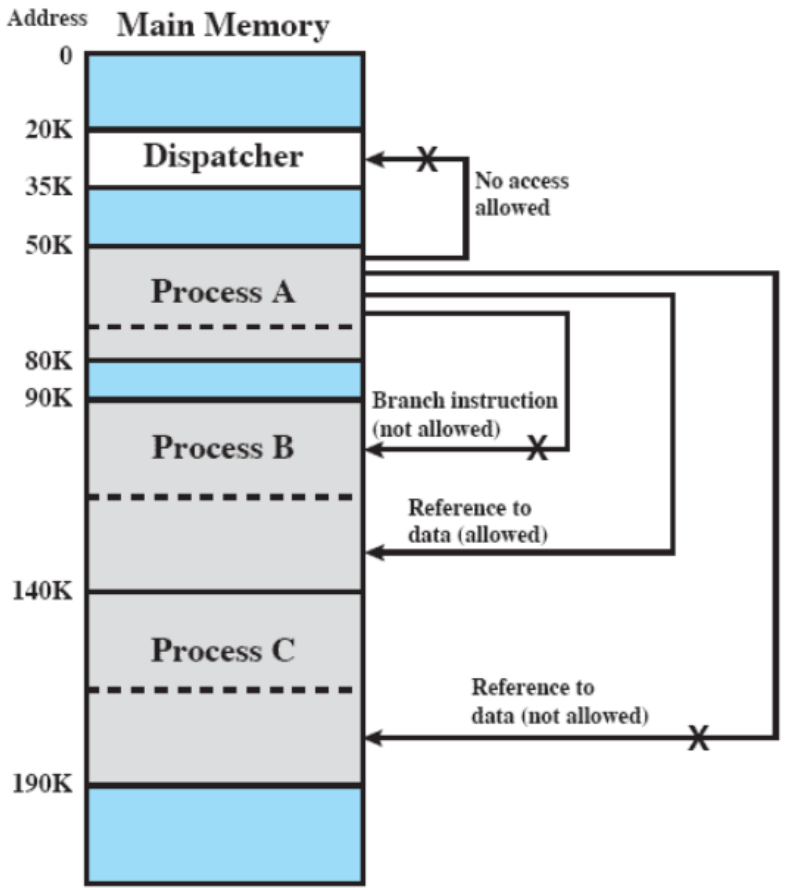
Realizzazione hardware



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

- Usare o no la memoria virtuale?
- Usare solo la paginazione, segmentazione o entrambi?
- 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 faults, perchè causano un notevole overhead
- L'overhead include decidere quali pagine rimpiazzare, quale altro processo mandare in esecuzione e gestire il process switch
- Infine si vuole minimizzare la probabilità che il processo in esecuzione faccia riferimento a un'istruzione o un dato non presente in memoria principale

Gestione della Memoria: Decisioni

- La performance ottenute con le scelte fatte dipende da vari fattori:
  - dimensione della memoria principale
  - differenza di velocità tra memoria principale e secondaria
  - numero di processi che competono per l'uso delle risorse

Elementi centrali per il progetto del SO

- Politica di prelievo (*fetch policy*)
- Politica di posizionamento (*placement policy*)
- Politica di sostituzione (*replacement policy*)
- Altri:
  - gestione del resident set
  - politica di pulitura
  - controllo del carico

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

Gestione del Resident Set

- Il problema della gestione del **Resident set** è legato al la politica di sostituzione (ma non lo approfondiamo)
- Risponde a 2 necessità:
  - quanti frame di RAM vanno allocati per ogni processo in esecuzione (*attivo*)
    - *resident set management* propriamente detto
  - quando si deve scegliere un frame per una sostituzione, bisogna scegliere solo tra i frame che appartengono al processo corrente (che ha causato il fault), oppure si può sostituire un frame qualsiasi?
    - *replacement scope*

Replacement Policy

- A prescindere dalla **politica di sostituzione** ci possono essere frame bloccati
- Il **Frame Locking** comporta che il frame bloccato non possa essere sostituito
  - Si fa a livello di kernel del sistema operativo
  - È sufficiente assegnare un bit ad ogni frame
  - Vengono bloccati i frame del sistema operativo, ed eventualmente quelli di altri processi

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:

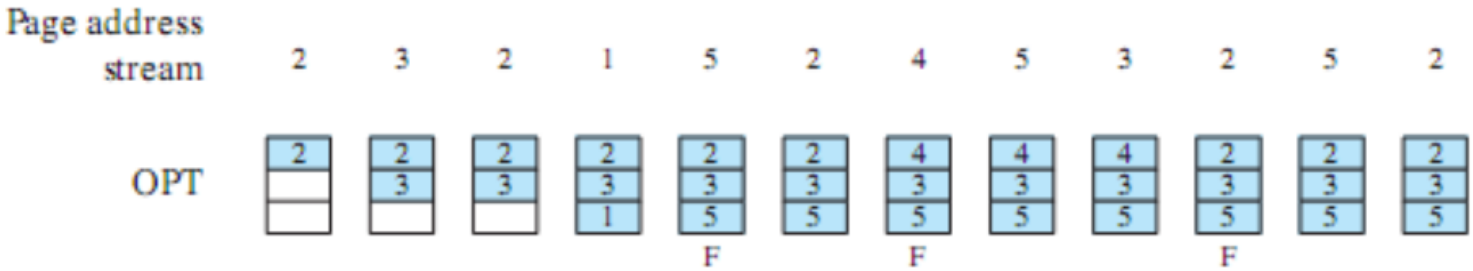
2 3 2 1 5 2 4 5 3 2 5 2

- 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 futuro
- Ovviamente, non è implementabile
- È però definibile sperimentalmente
- Usata per confronti sperimentali

Sostituzione Ottimale sull'Esempio



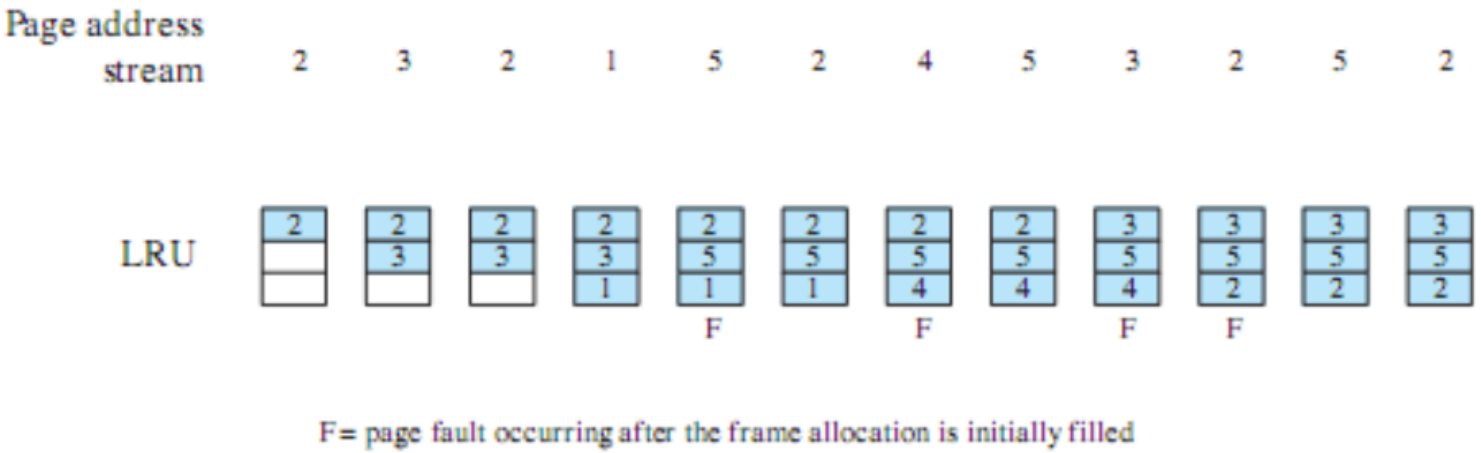
F = page fault occurring after the frame allocation is initially filled

Risultato: 3 page faults

Sostituzione LRU

- Con la politica **LRU** si sostituisce la pagina cui non sia 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
  - anche per la cache si usa questa tecnica ma è implementata in hardware
  - ma per la memoria secondaria non si può fare in hardware (sarebbe troppo costoso)

Sostituzione LRU sull'Esempio



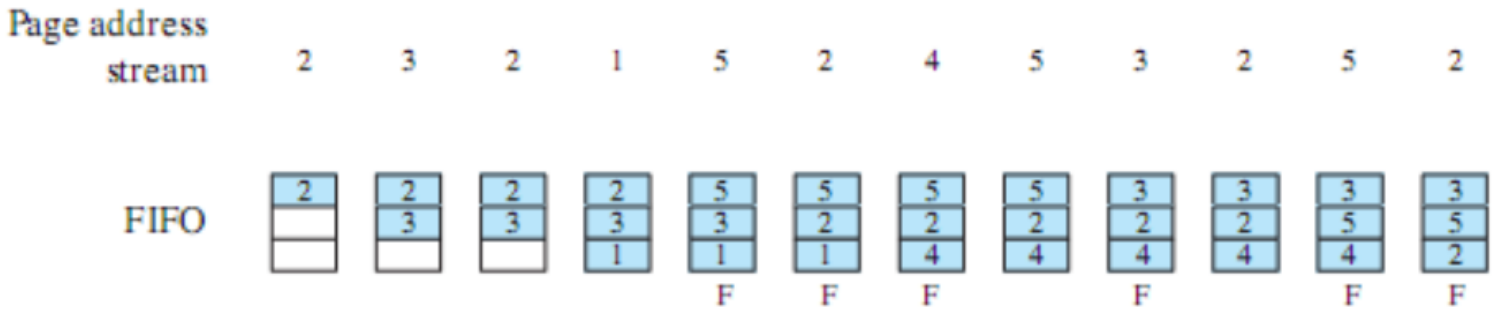
F= page fault occurring after the frame allocation is initially filled

Risultato: 4 page faults, quasi come l'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



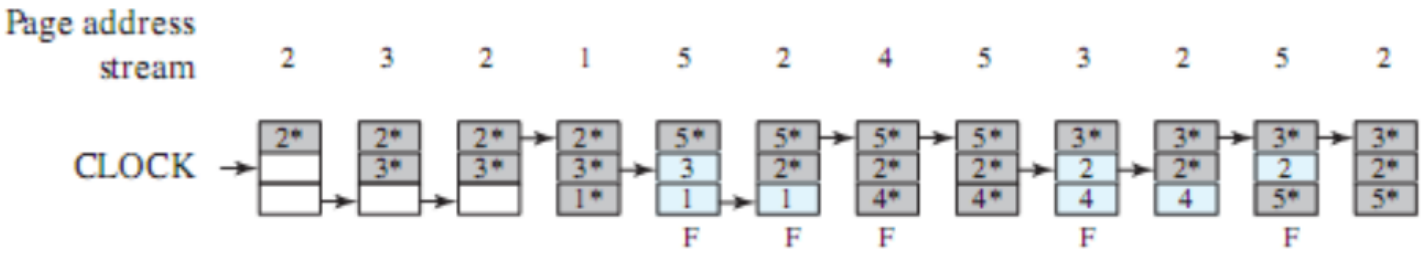
Risultato: 6 page faults

Non si accorge che la 2 e la 5 sono molto richieste

Sostituzione dell'Orologio

- Con 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

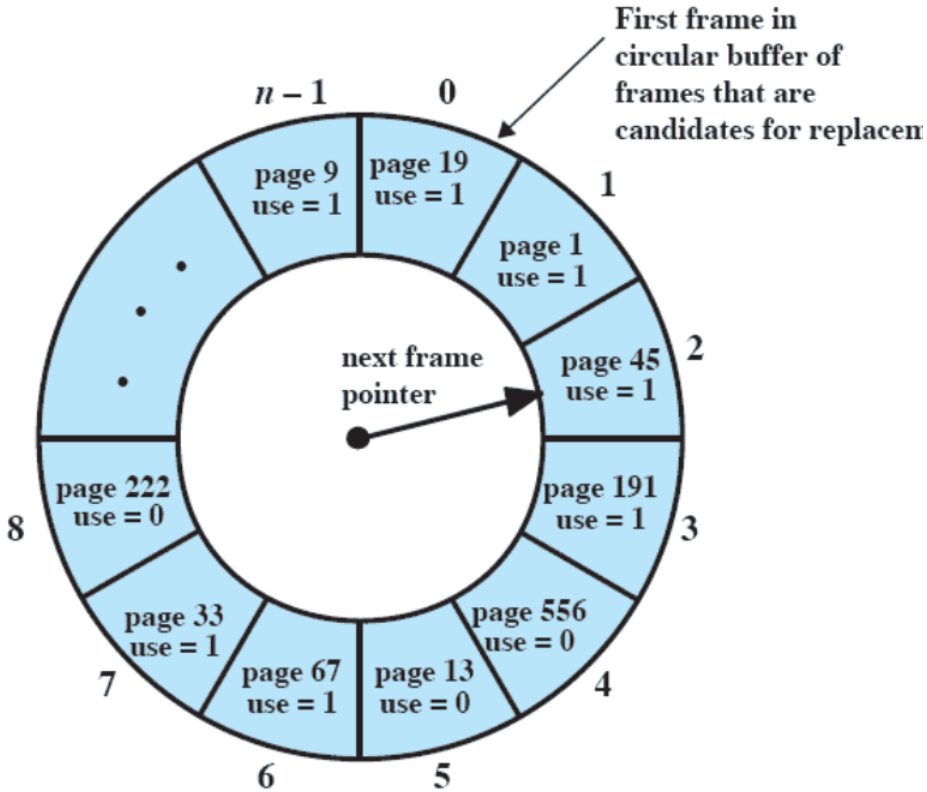


F= page fault occurring after the frame allocation is initially filled

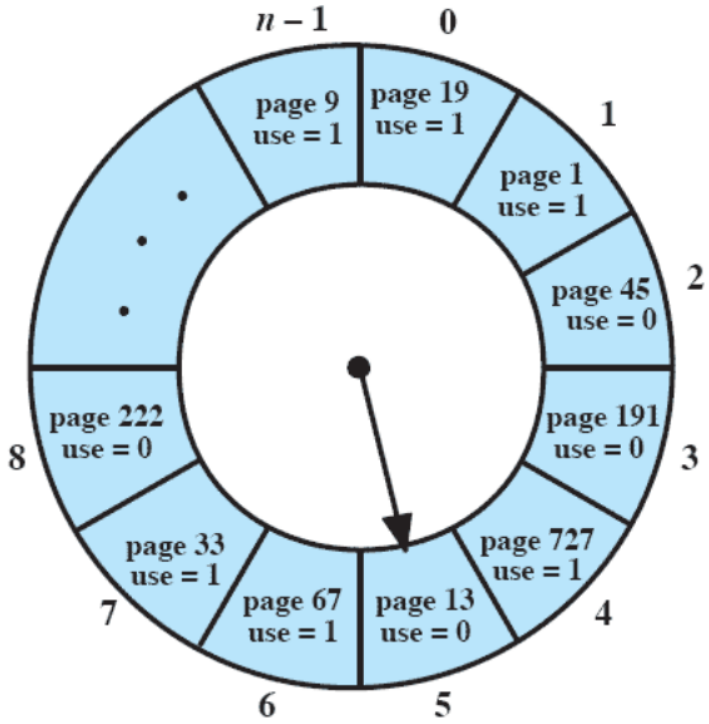
Risultato: 5 page faults

Si accorge che la 2 e la 5 sono molto richieste

Politica dell'Orologio

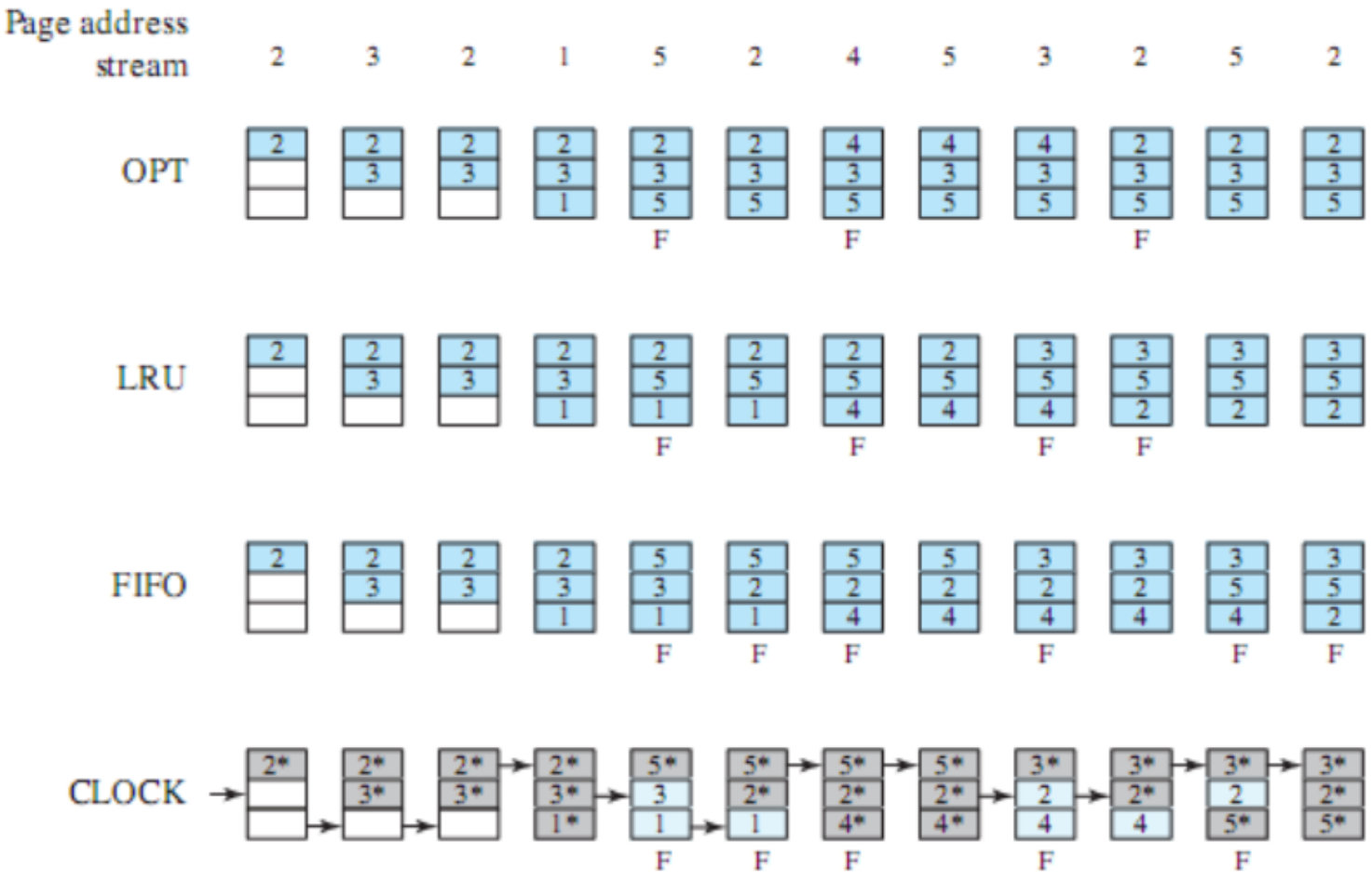


(a) State of buffer just prior to a page replacement



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

Algoritmi di sostituzione sull'esempio



Algoritmi di sostituzione: Confronto