# 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

## 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:

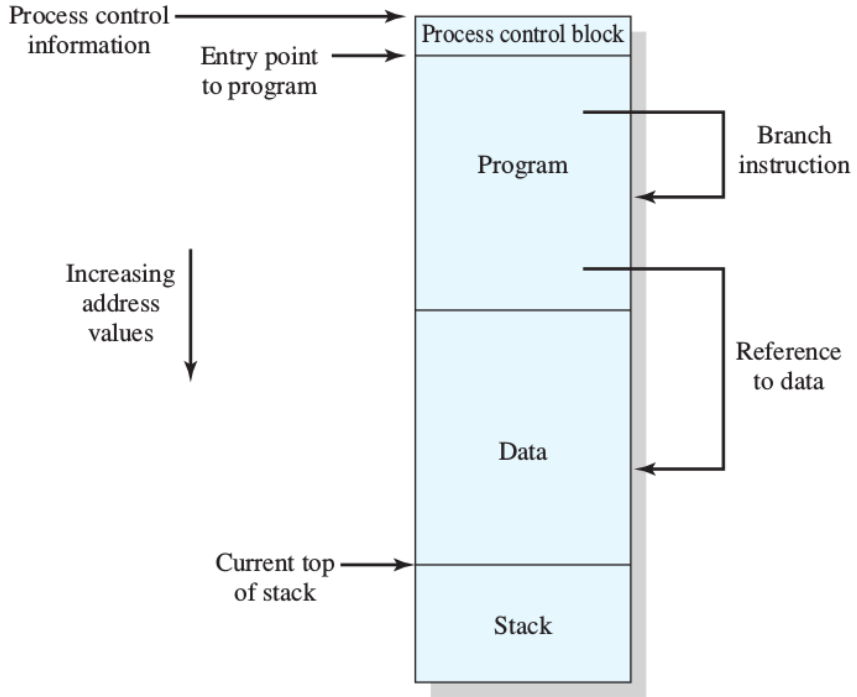
- **Rilocazione**
  - serve aiuto hardware
- **Protezione**
  - serve aiuto hardware
- **Condivisione**
- **Organizzazione logica**
- **Organizzazione fisica**

# 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

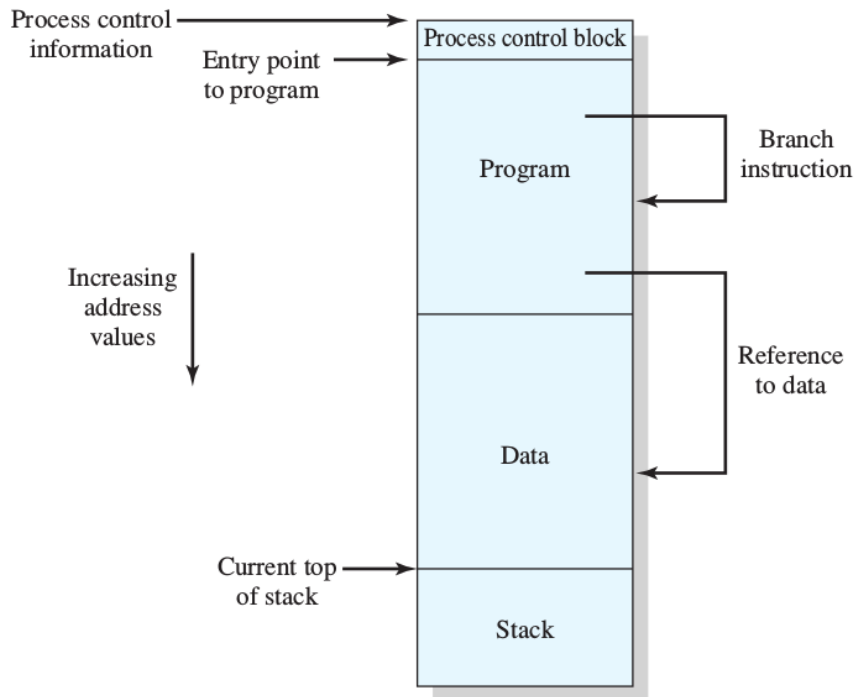
# Rilocazione: gli Indirizzi nei Programmi

- 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

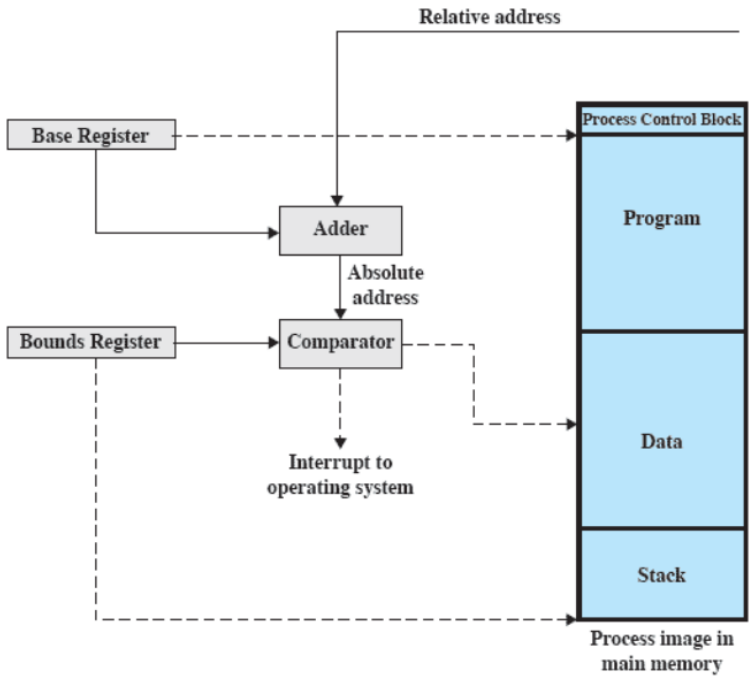
**Relativi:** il riferimento è espresso come uno spiazzamento rispetto ad un qualche punto noto

- 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

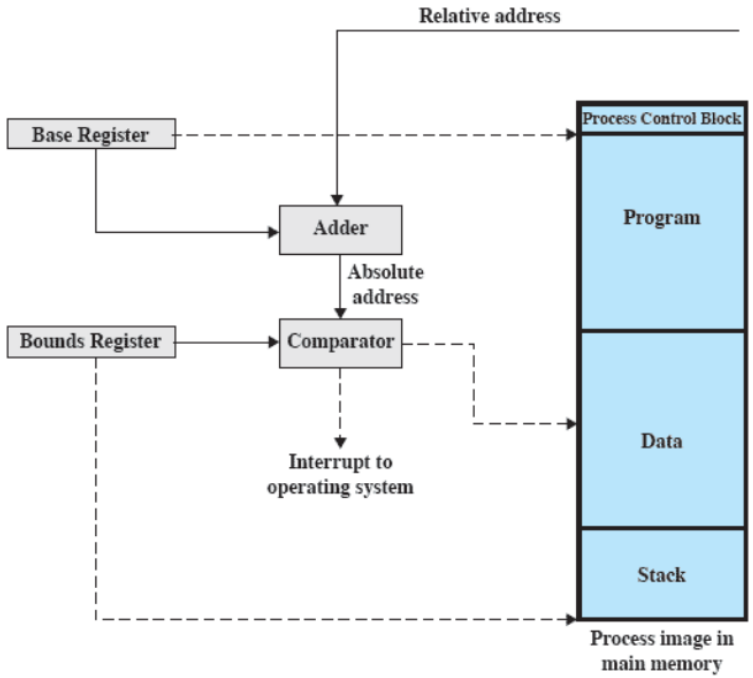



*Non si tiene conto della memoria virtuale*

# 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 **segmentazione** e **paginazione**
- Prima di esse vediamo tecniche più semplici che facilitano la comprensione della memoria virtuale

## 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*

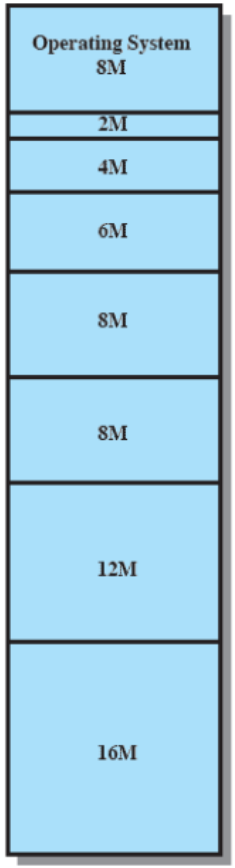
# Partizionamento uniforme: problemi

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

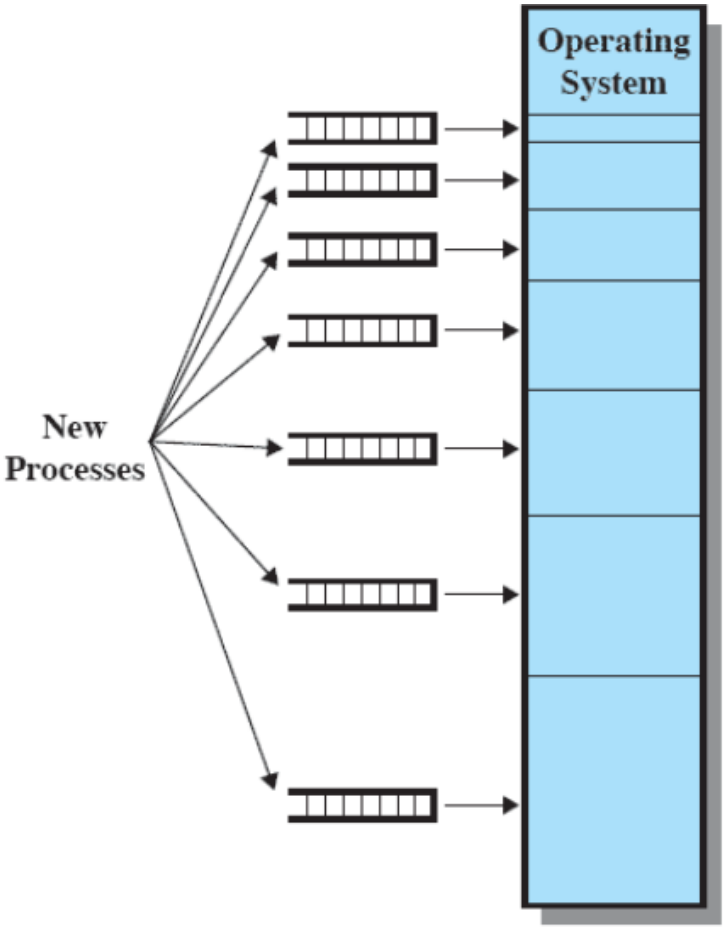
- Mitiga entrambi i problemi
  - ma non li risolve
- Nella figura, programmi più piccoli di 16M possono essere gestiti senza *overlay*
- Per quelli più piccoli, ci sono le partizioni più piccole
- È sempre partizionamento fisso: le partizioni sono quelle decise all'inizio e non cambiano più

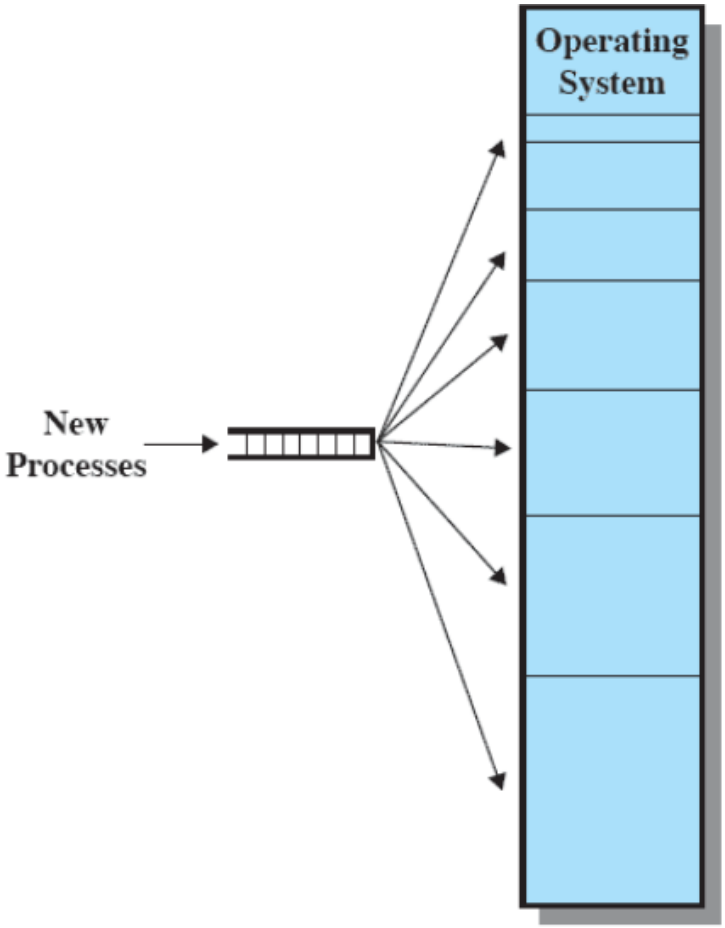


# Algoritmo di posizionamento

- 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

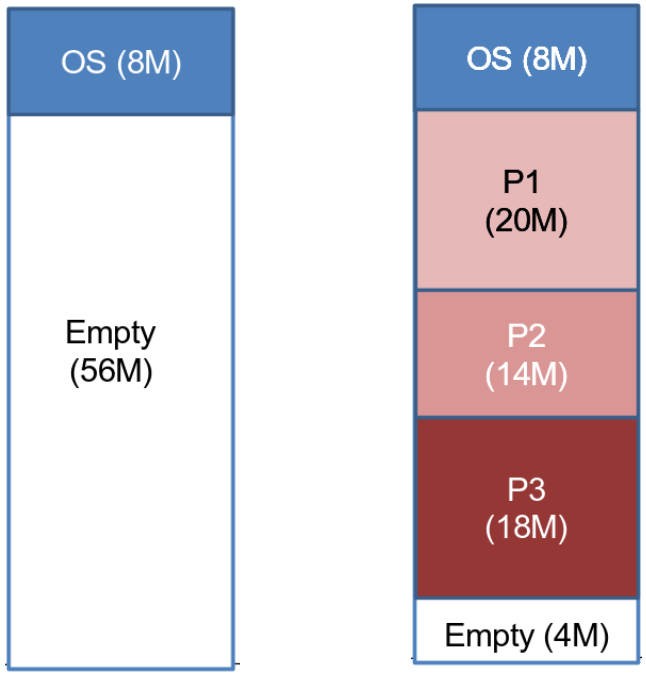
# Partizionamento dinamico

- 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

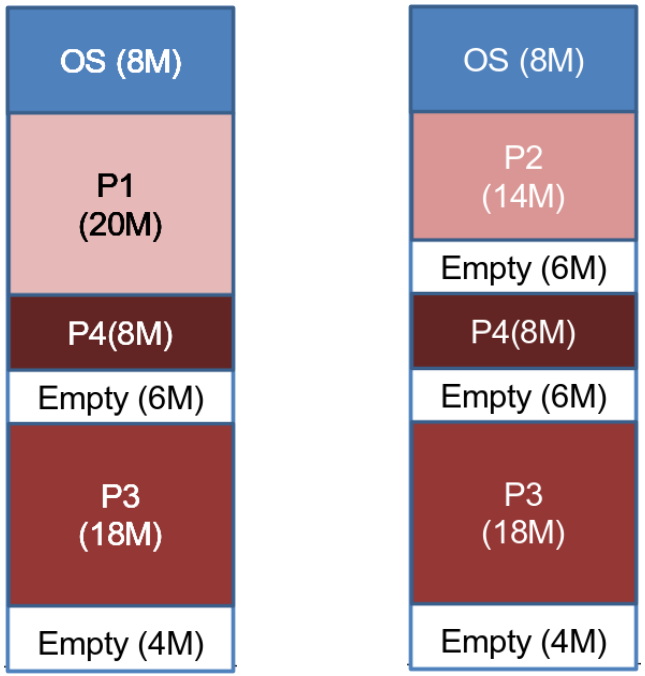
- All'inizio la memoria principale è vuota, eccetto per il SO
- Vengono poi caricati i primi tre processi - P1 di 20M, P2 di 14M, P3 di 18M - a partire da dove finisce il SO
- Resta libera una piccola porzione di memoria (4M), troppo piccola per il processo P4 di 8M
- Ad un certo punto nessuno dei processi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

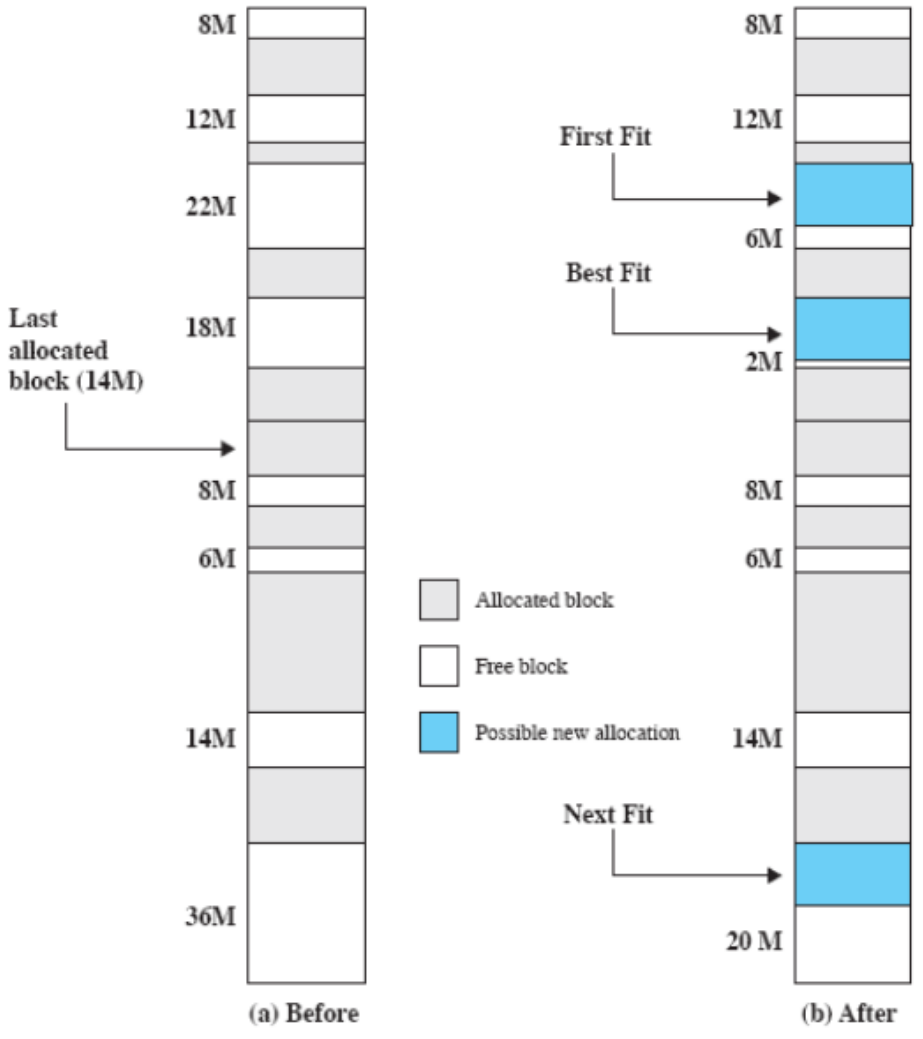
# Partizionamento dinamico

- 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

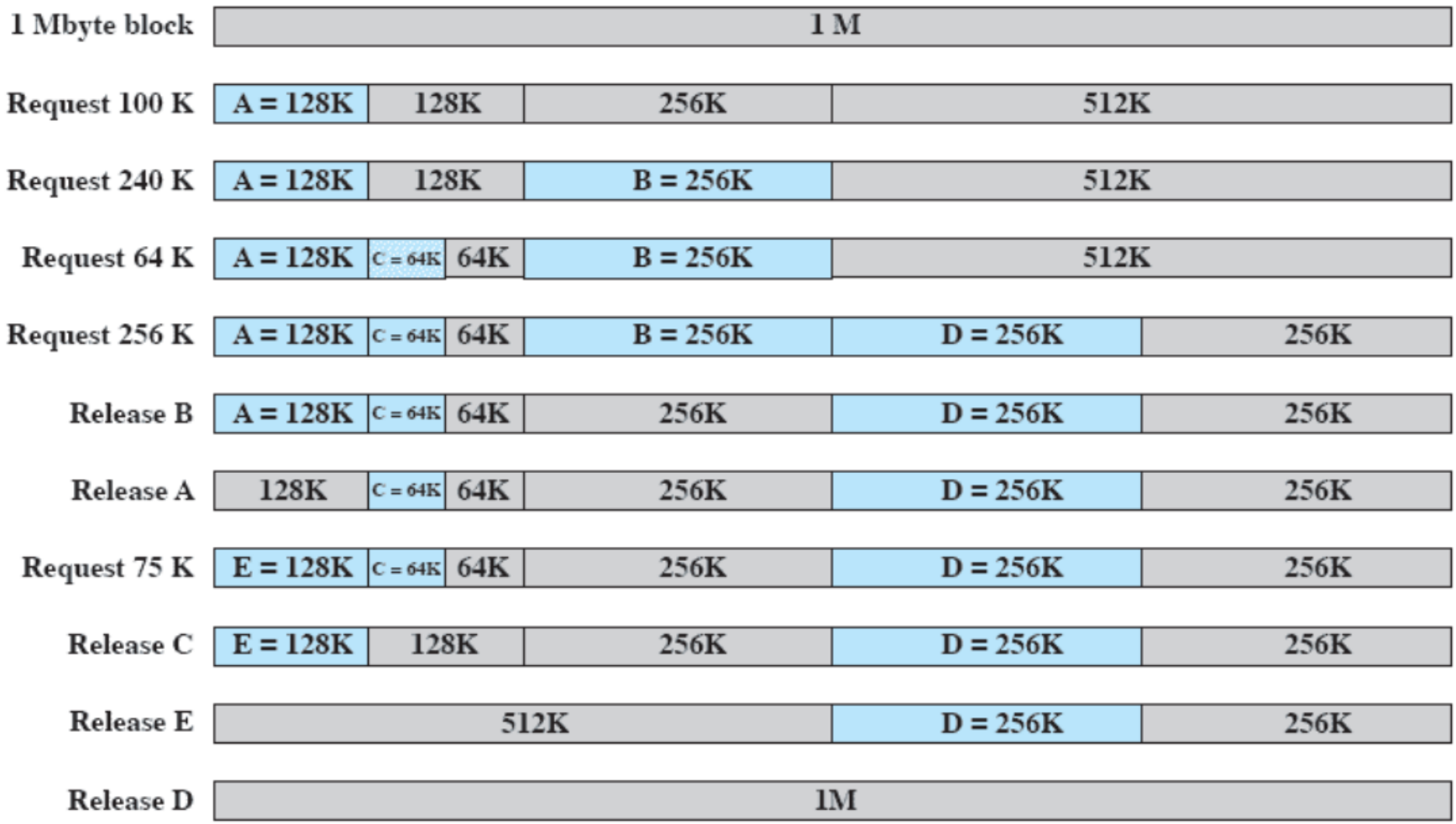
- **best-fit**
- **first-fit**
- **next-fit**



# 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