Sistemi Operativi
Gestione della Memoria (parte 1)

Docente: Claudio E. Palazzi

cpalazzi@math.unipd.it

Crediti per queste slides al Prof. Tullio Vardanega

Considerazioni preliminari – 1

- Nell’ottica degli utenti applicativi la memoria deve essere
  - Capiente
  - Veloce
  - Permanente (non volatile)
- Solo **l’intera** gerarchia di memoria nel suo insieme possiede tutte queste caratteristiche
- Il **gestore della memoria** è la componente di S/O incaricata di soddisfare le esigenze di memoria dei processi

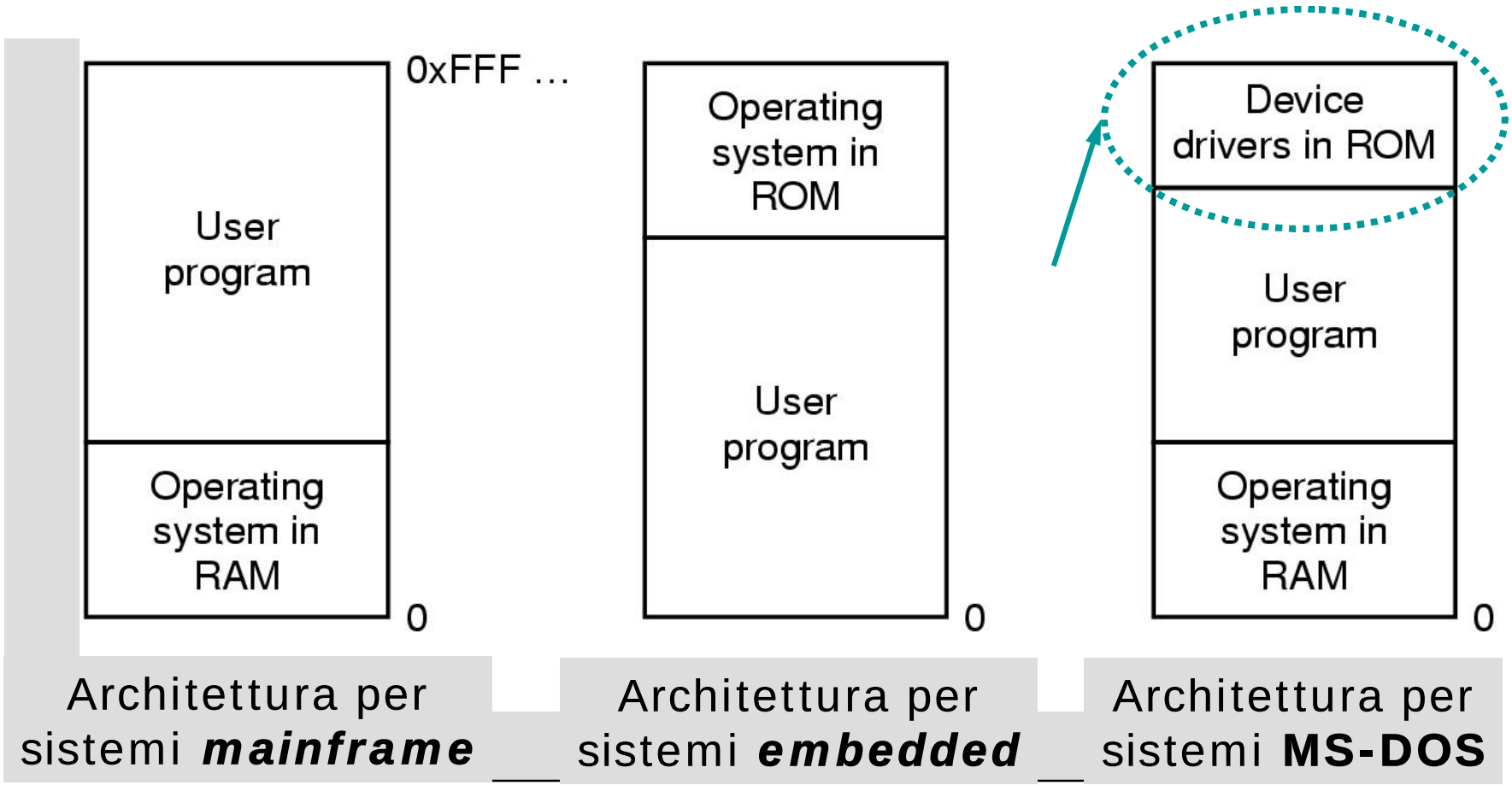
Considerazioni preliminari – 2

- Esistono due classi fondamentali di sistemi di gestione della memoria
  1. Per processi allocati in modo fisso
  2. Orientate a processi soggetti a migrazione da memoria principale a disco durante l'esecuzione
- La memoria disponibile è in generale **inferiore** a quella necessaria per tutti i processi attivi simultaneamente

Sistemi monoprogrammati – 1

- Esegue un solo processo alla volta
- La memoria disponibile è ripartita solo tra quel processo e il S/O
- L'unica scelta progettuale rilevante in questo caso è decidere dove allocare la memoria (dati e programmi) del S/O
- La parte di S/O ospitata in RAM è però solo quella che contiene l'ultimo comando invocato dall'utente

Sistemi monoprogrammati – 2



Architettura per sistemi ***mainframe*** | Architettura per sistemi ***embedded*** | Architettura per sistemi **MS-DOS**

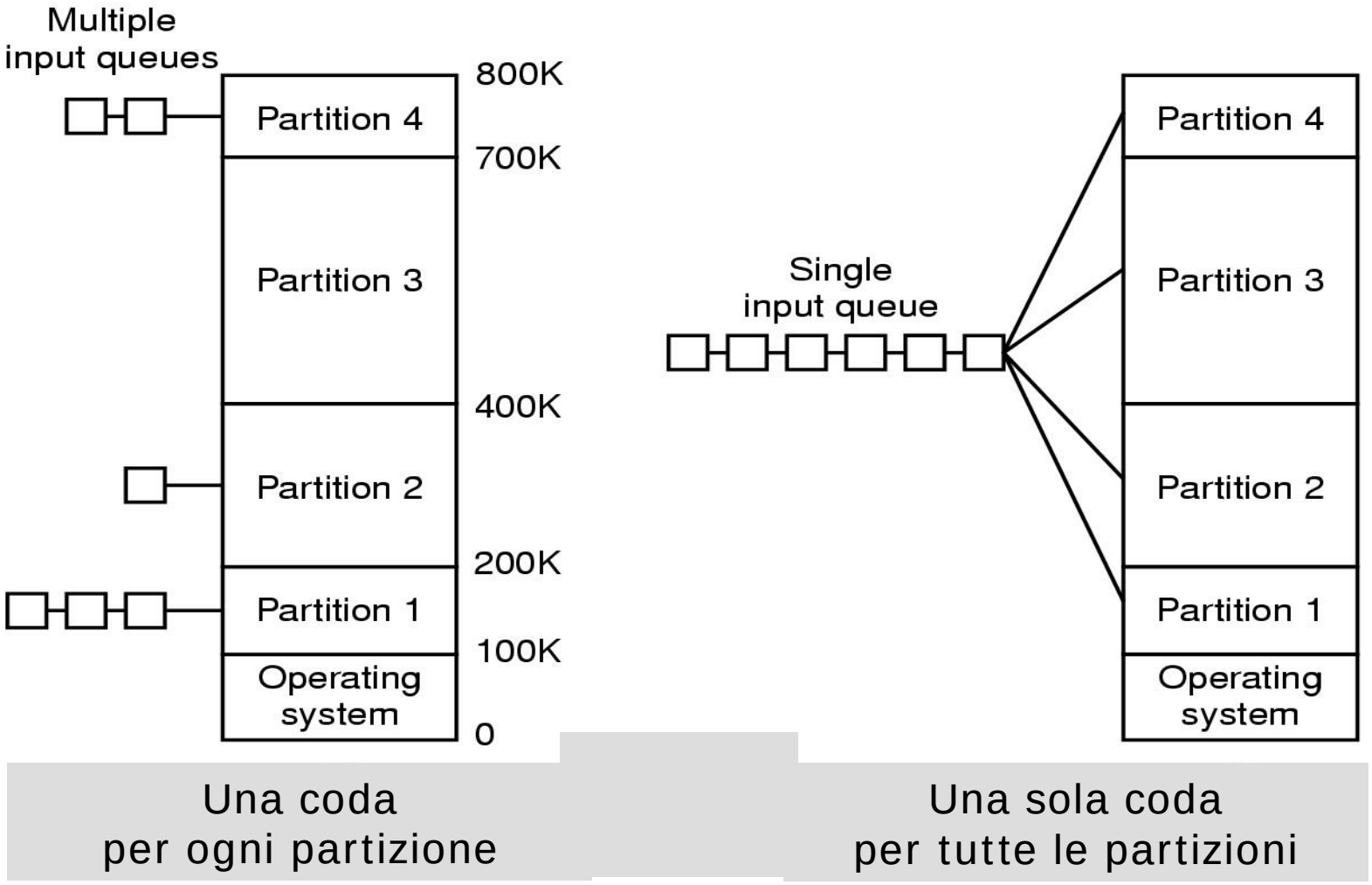
Sistemi multiprogrammati – 1

- La forma più rudimentale di gestione della memoria per questi sistemi crea una partizione per ogni processo
  - Staticamente all'avvio del sistema
  - Le partizioni possono avere dimensione diversa
- Il problema diventa assegnare dinamicamente processi a partizioni
  - Minimizzando la frammentazione interna

Sistemi multiprogrammati – 2

- A ogni nuovo processo (o lavoro) viene assegnata la partizione di dimensione più appropriata
  - Una coda di processi per partizione
  - Scarsa efficacia nell'uso della memoria disponibile
- Assegnazione opportunistica
  - Una sola coda per tutte le partizioni
    - Quando si libera una partizione questa viene assegnata al processo a essa **più adatto** e più avanti nella coda
    - Oppure assegnata al "**miglior**" processo scandendo l'intera coda
      - I processi più "piccoli" sono discriminati quando invece meriterebbero di essere privilegiati in quanto più interattivi

Sistemi multiprogrammati – 3



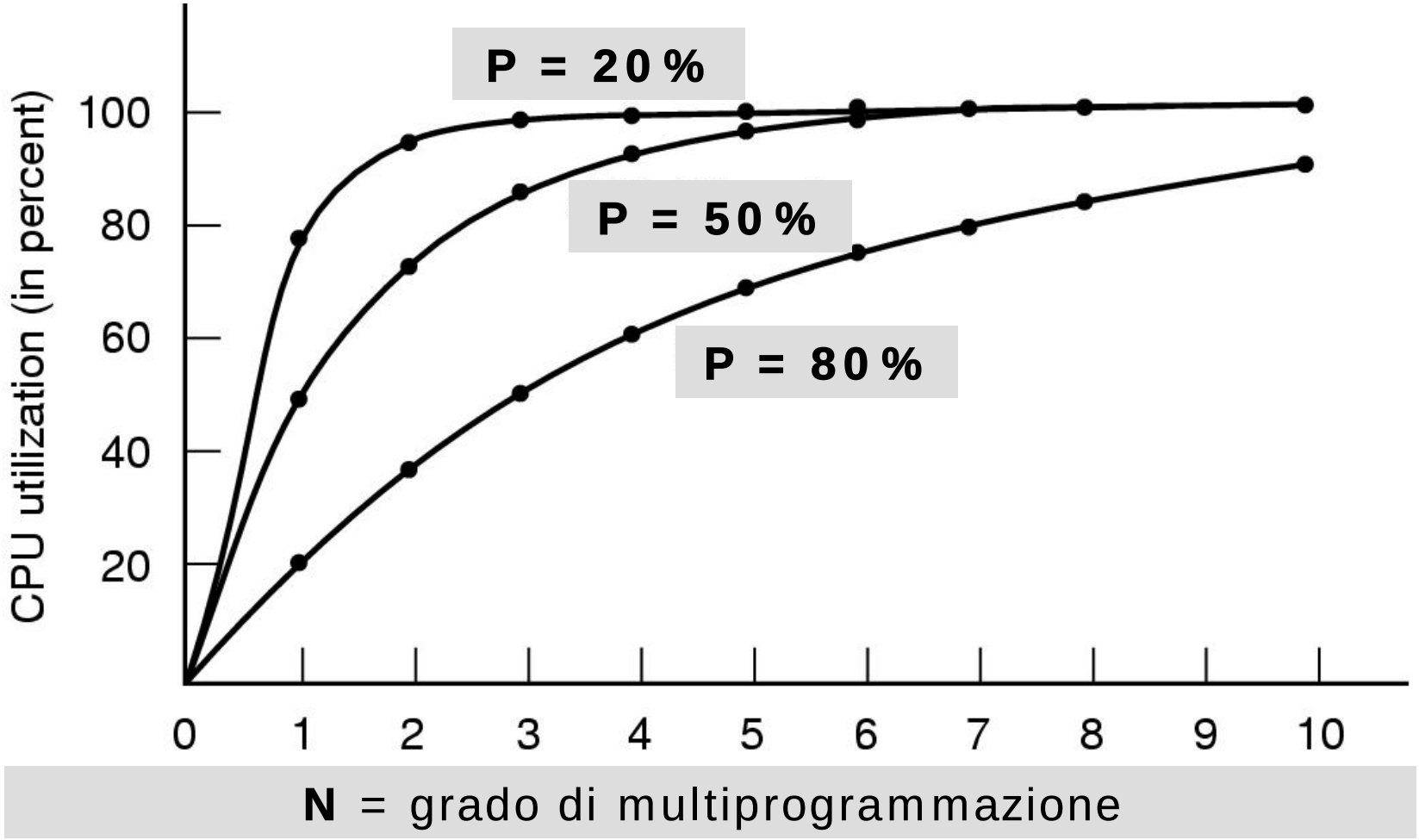
Una coda
per ogni partizione

Una sola coda
per tutte le partizioni

Valutazione dei vantaggi della multiprogrammazione – 1

- Valutazione **probabilistica** di quanti processi debbano eseguire in parallelo per massimizzare l'utilizzazione della CPU
  - Sotto l'ipotesi che
    - Ogni processo impegni il P% del suo tempo in attività di I/O
    - N processi simultaneamente in memoria
  - L'utilizzo stimato della CPU allora è $1 – P^N$

Valutazione dei vantaggi della multiprogrammazione – 2



Esempio Progettazione Memoria

- Si consideri un computer con 32 MB di memoria e 80% di attesa I/O media per ogni processo
  - 16 MB riservati per il sistema operativo
  - 4 MB riservati per ciascun processo
  - In totale si hanno quindi 4 processi simultaneamente in memoria
  - Con P = 0,8 si ha una utilizzazione della CPU di $1 - 0{,}8^4 =$ **60%**
- Aggiungendo altri 16 MB
  - Si possono avere 8 programmi simultaneamente in memoria
  - Con P = 0,8 si ha una utilizzazione della CPU di $1 - 0{,}8^8 =$ **83%**
- Aggiungendo altri 16 MB
  - Si possono avere 12 programmi simultaneamente in memoria
  - Con P = 0,8 si ha una utilizzazione della CPU di $1 - 0{,}8^{12} =$ **93%**

Rilocazione e protezione

- **Rilocazione**
  - Interpretazione degli indirizzi emessi da un processo in relazione alla sua collocazione corrente in memoria
    - Occorre distinguere tra riferimenti assoluti **permissibili** al programma e riferimenti relativi da rilocare
- **Protezione**
  - Assicurazione che ogni processo operi soltanto nello spazio di memoria a esso permissibile
    - Soluzione storica adottata da IBM
      - Memoria divisa in blocchi (2 kB) con codice di protezione per blocco (4 *bit*)
      - La PSW di ogni processo indica il suo codice di protezione
      - Il S/O blocca ogni tentativo di accedere a blocchi con codice di protezione diverso da quello della PSW corrente
    - Soluzione combinata (rilocazione + protezione)
      - Un processo può accedere memoria solo tra la **base** e il **limite** della partizione a esso assegnata
      - Valore **base** aggiunto al valore di ogni indirizzo riferito (operazione costosa)
      - Il risultato confrontato con il valore **limite** (operazione veloce)

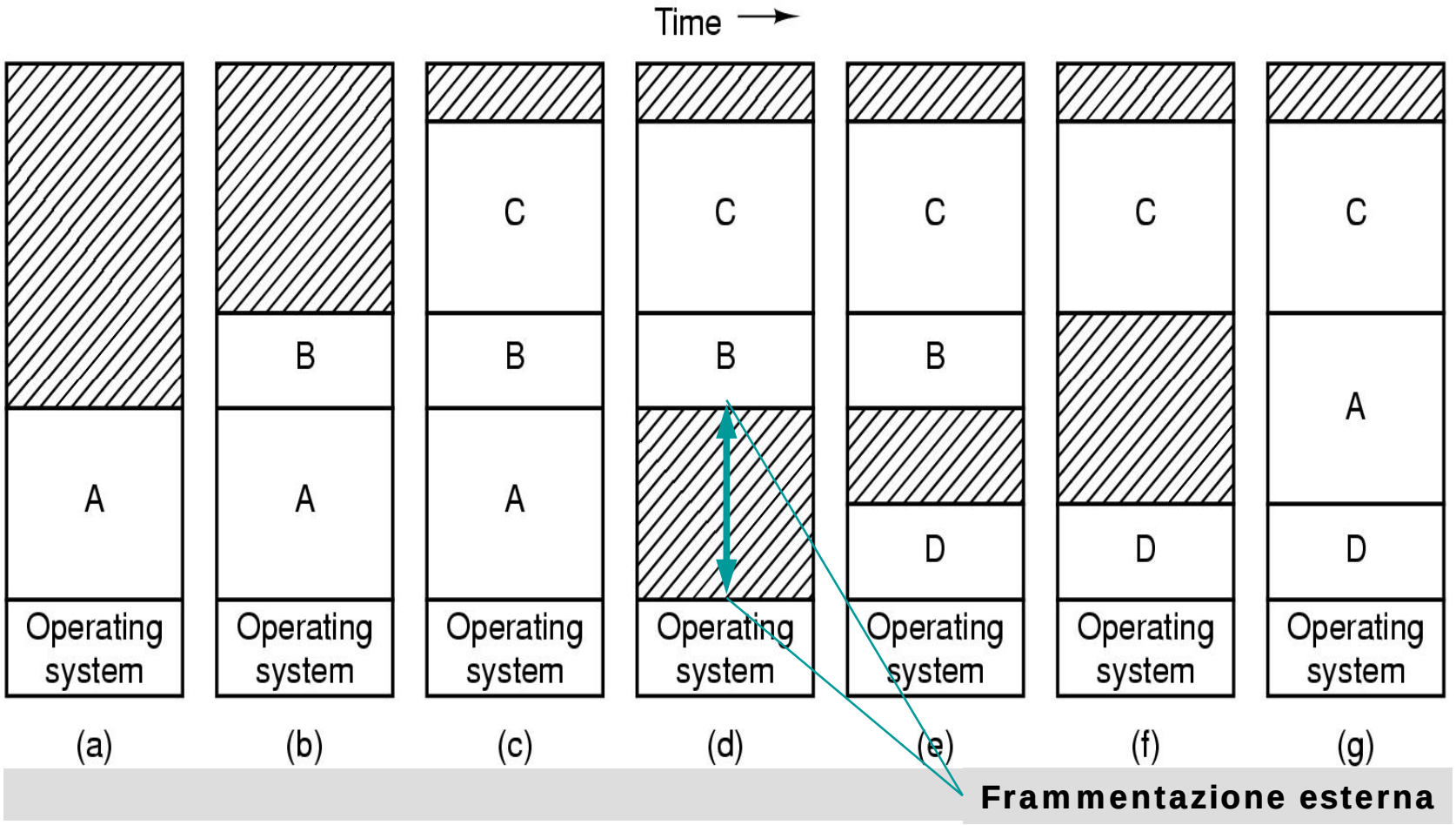
Swapping – 1

- La tecnica più rudimentale per alternare processi in memoria principale **senza** garantire allocazione fissa
- Trasferisce processi **interi** e assegna partizioni **diverse** nel tempo
- Il processo rimosso viene salvato su memoria secondaria
  - Ovviamente solo le parti modificate

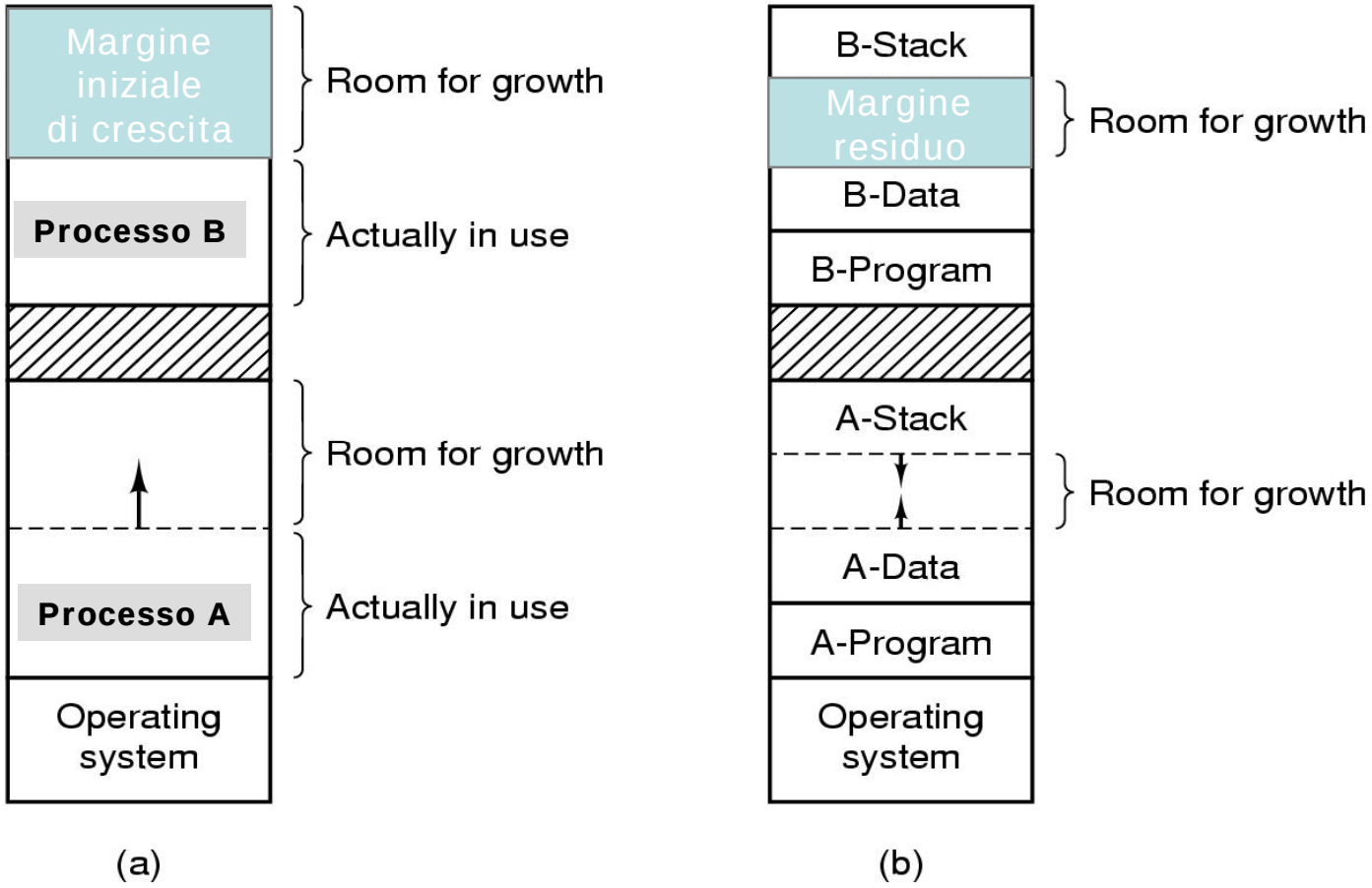
Swapping – 2

- Processi diversi richiedono partizioni di ampiezze diverse assegnate *ad hoc*
  - Rischio di frammentazione **esterna**
  - Occorre ricompattare periodicamente la memoria principale
    - Pagando un costo temporale importante!
      - Spostando 4 B in 40 ns., servono 5.37 s. per una RAM ampia 512 MB
- Le dimensioni di memoria di un processo possono variare nel tempo!
  - Difficile ampliare dinamicamente l'ampiezza della partizione assegnata
  - Meglio assegnare **con margine**

Swapping – 3


Time
C
B
A
D
Operating system
(a)
(b)
(c)
(d)
(e)
(f)
(g)
Frammentazione esterna

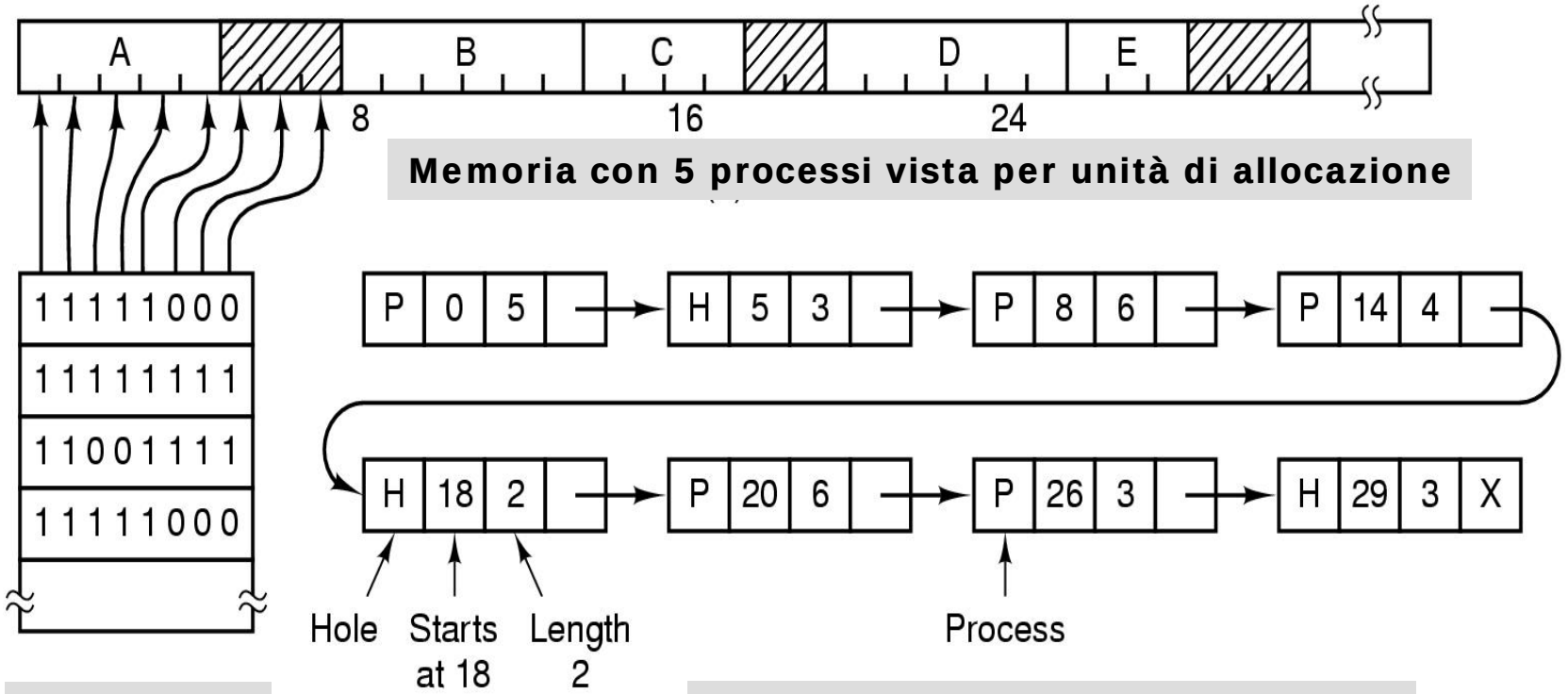
Swapping – 4


Margine iniziale di crescita
Room for growth
Processo B
Actually in use
Room for growth
Processo A
Actually in use
Operating system
(a)
B-Stack
Margine residuo
Room for growth
B-Data
B-Program
A-Stack
Room for growth
A-Data
A-Program
Operating system
(b)

Strutture di gestione – 1

- Con memoria principale allocata dinamicamente è essenziale tenere traccia del suo stato d'uso
- Due strategie principali
  - **Mappe di *bit***
    - Memoria vista come insieme di **unità di allocazione** (1 *bit* per unità)
      - Unità piccole → struttura di gestione grande
        - » Esempio: Unità da 32 *bit* e RAM ampia 512 MB → struttura ampia 128 M *bit* = 16 MB → 3.1 % (= 1/32)

Strutture di gestione – 2



Strutture di gestione – 3

- La strategia alternativa usa **liste collegate**
  - Nella sua versione più semplice la memoria è vista a segmenti
    - Segmento = processo | spazio libero tra processi
    - Ogni elemento di lista rappresenta un segmento
      - Ne specifica punto di inizio, ampiezza e successore
      - Liste ordinate per indirizzo di base
  - Varie strategie di allocazione
    - *First fit* : il primo segmento libero ampio abbastanza
    - *Next fit* : come *First fit* ma cercando sempre avanti
    - *Best fit* : il segmento libero più adatto
    - *Worst fit* : sempre il segmento libero più ampio
    - *Quick fit* : liste diverse di ricerca per ampiezze “tipiche”