3 – Gestione e organizzazione della memoria

Sommario

Organizzazione e gestione della memoria

- **Gerarchie di memoria**

Allocazione di memoria contigua e non contigua

Mono utente

- **Overlay**
- **Protezione in sistemi mono utente** **Elaborazione Batch a flusso singolo**

Multiprogrammazione

- **con partizioni fisse**
- **con partizioni variabili**
  - **strategie di posizionamento in memoria**
- **Swapping di memoria**

Memoria Virtuale

- **paginazione**
- **località**

3 – Gestione e organizzazione della memoria

Paginazione: algoritmi di sostituzione di pagina
- **Random**
- **First-In-First-Out (FIFO) e anomalie**
- **Least-Recently-Used (LRU)**
- **Least-Frequently-Used (LFU)**
- **Not-Recently-Used (NRU)**
- **Modifica a FIFO: Seconda-Chance e pagina clock**
- **Pagina lontana**

Modello Working Set

Sostituzione di pagina con Page-Fault-Frequency (PFF)

Problemi di progettazione di sistemi con Paginazione
- **Dimensione della pagina**
- **Comportamento dei programmi con paginazione**
- **Sostituzione di pagina globale vs. locale**

3 – Gestione e organizzazione della memoria

Segmentazione

Traduzione di indirizzi
Condivisione nei sistemi con segmentazione
Protezione e controllo degli accessi

Sistemi con Segmentazione/Paginazione

Traduzione dinamica degli indirizzi
Condivisione e protezione

Obbiettivi

- Necessità di gestione memoria reale (fisica)
- Gerarchie di memoria
- Allocazione di memoria contigua e non contigua
- Multiprogrammazione con partizioni fisse e variabili
- Swapping
- Strategia di posizionamento in memoria
- Memoria Virtuale vantaggi e svantaggi della paginazione a previsione e a richiesta
- problemi nella sostituzione delle pagine.
- confronto tra strategie di sostituzione delle pagine e ottimizzazione
- impatto della dimensione pagina su prestazioni della memoria virtuale
- comportamento del programma nella paginazione

Introduzione

- Memoria divisa in livelli
  - Memoria principale
    - relativamente costosa
    - relativamente con capacità limitata
    - Alte prestazioni
  - Memoria secondaria
    - Economica
    - Grande capacità
    - Lenta
  - La memoria principale richiede un'attenta gestione

Organizzazione della memoria

- La memoria può essere organizzata in modi diversi
  - Un processo utilizza tutto lo spazio di memoria
  - Ogni processo ottiene una propria partizione in memoria
    - Allocata dinamicamente o staticamente
- nota: i requisiti di memoria delle applicazioni tendono ad aumentare nel tempo e a saturare la capacità di memoria principale

Gestione e organizzazione della memoria

- Gli utenti, i programmatori richiederebbero che la memoria fosse
  - grande
  - veloce
  - non volatile

- Gerarchie di memoria
  - Memoria cache – piccola, veloce, costosa
  - Memoria principale – velocità media, prezzo medio
  - Memoria secondaria – molto grande (GB, TB), economica, lenta

Gestione della memoria

- Strategie per ottimizzare le prestazioni della memoria
  - Eseguite dal gestore di memoria che considera le gerarchie di memoria
    - Quale processo rimarrà in memoria?
    - A quanta memoria ogni processo ha accesso?
    - Dove posizionare in memoria ogni processo?

Gerarchie di memoria

- Memoria principale
  - Dovrebbe memorizzare solo i programmi e dati necessari al momento
- Memoria secondaria
  - Memorizza dati e programmi che non sono necessari al momento
- Memoria cache
  - Velocità molto alta
  - Di solito si trova sul processore stesso
  - I dati più usati sono copiati nella cache per un accesso più veloce
  - Una piccola cache è utile per migliorare le prestazioni
    - Sfrutta la località temporale

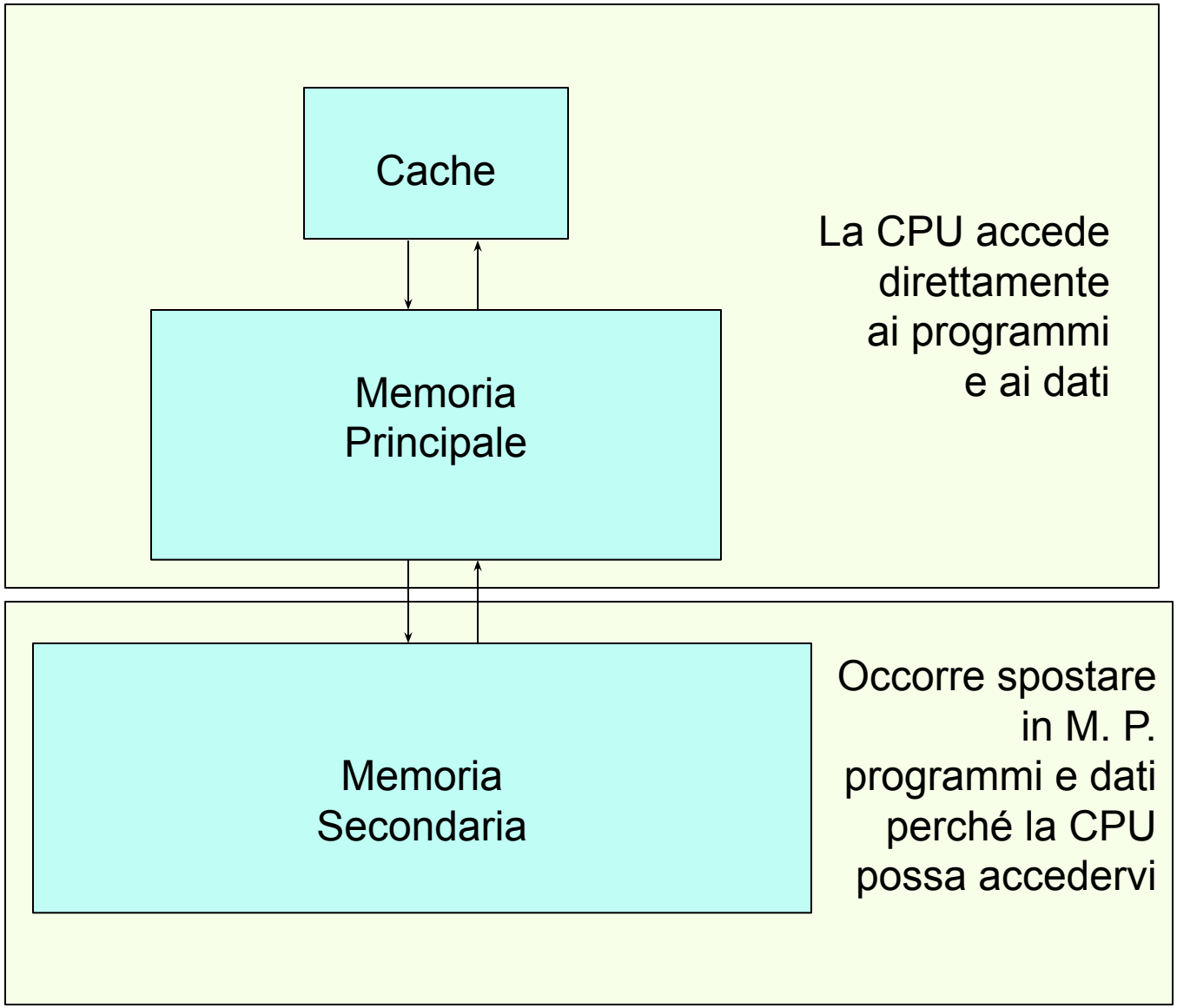
Gerarchie di memoria

Tempo di accesso alla memoria diminuisce

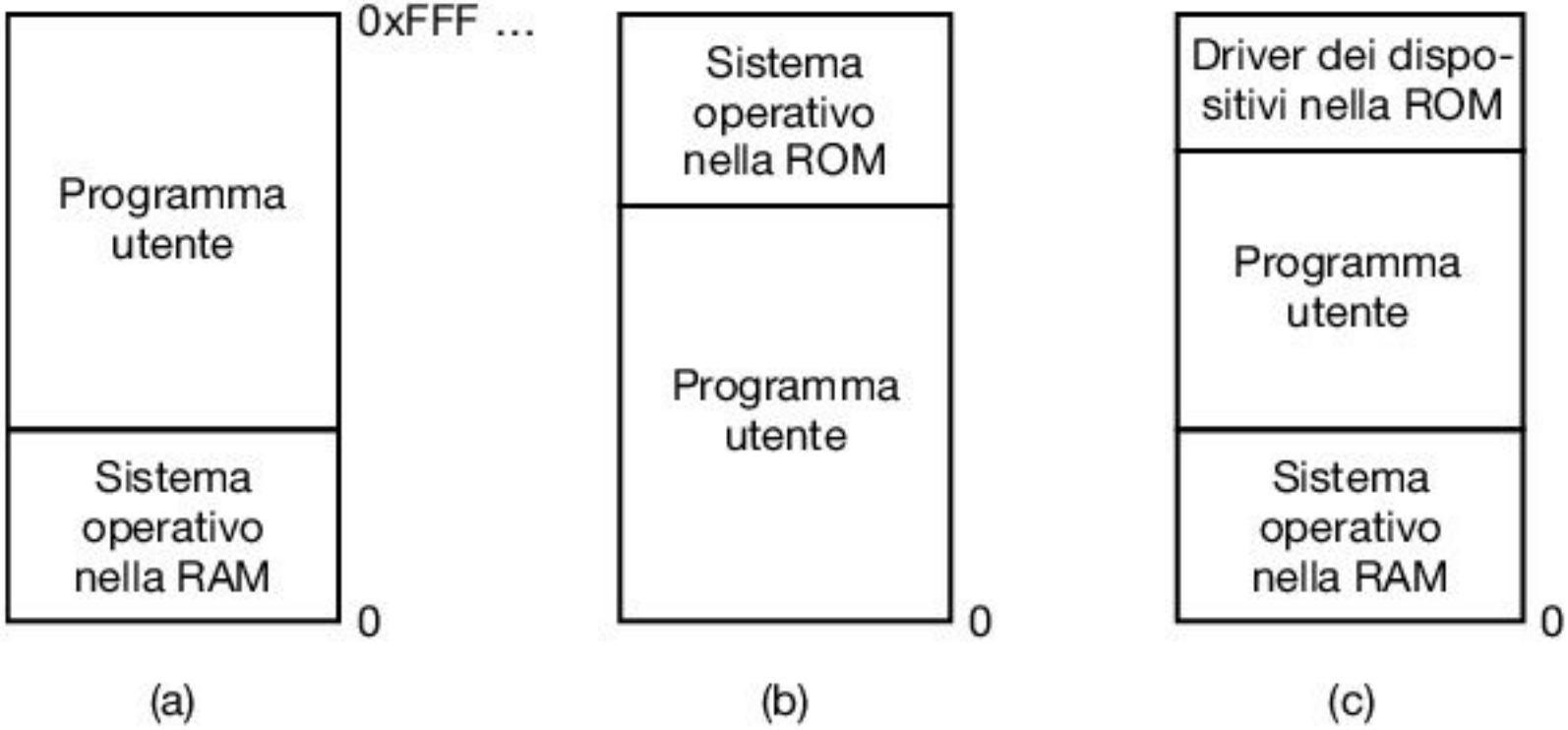
Velocità di accesso alla memoria aumenta

Costo per bit di memoria aumenta

Capacità di memoria diminuisce



Organizzazione della memoria



Semplice organizzazione di memoria con il sistema operativo e un solo processo utente – senza paginazione o swapping

a) usato nei mainframe e minicomputer, oggi poco usato

b) su alcuni palmari e sistemi integrati

c) sui primi PC, parte della ROM è la BIOS

Strategie di gestione della memoria

- Strategie divise in diverse categorie
  - Strategie di **fetch** - **quando**?
    - A richiesta o a previsione
    - Decide quando spostare la prossima sezione di programma e dati
  - Strategie di **posizionamento** - **dove**?
    - Decide dove inserire i dati e programmi in memoria principale
    - Esempi: Best-fit, first-fit, worst-fit
  - Strategie di **sostituzione** - **chi**?
    - Decide quali dati o programmi da rimuovere dalla memoria principale per creare spazio quando necessario

Allocazione di memoria contigua vs. non contigua

- Modi di organizzare i programmi in memoria
  - Allocazione **contigua**
    - Un programma deve essere memorizzato come un unico blocco di indirizzi contigui
    - Può essere impossibile trovare un blocco abbastanza grande
    - basso overhead
  - Allocazione **non contigua**
    - Il programma è diviso in blocchi chiamati segmenti
    - Ogni segmento può essere allocato in diverse parti della memoria
    - Più facile trovare "buchi" in cui un segmento possa essere memorizzato
    - L'aumento del numero di processi che possono contemporaneamente essere in memoria compensa l'overhead sostenuto da questa tecnica

Allocazione di memoria contigua mono utente

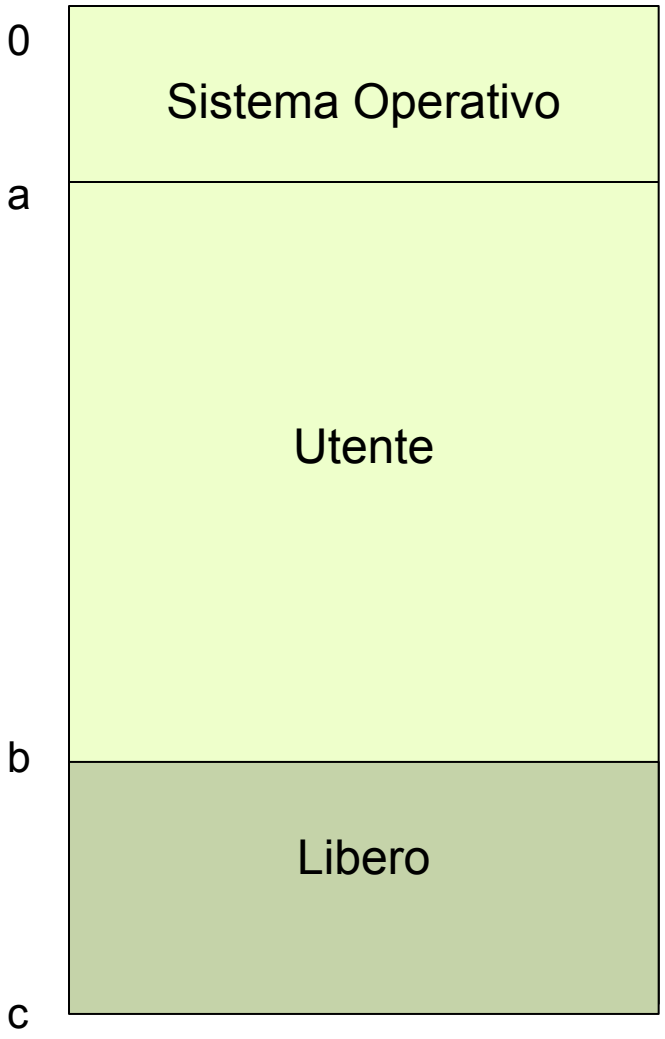
- Un utente ha il controllo dell'intero sistema
- Assenza di modello di astrazione della memoria

  - Le risorse non hanno bisogno di essere condivise
  - Originariamente senza S.O.

  - Il programmatore scrive il codice per eseguire la gestione delle risorse incluso I/O a livello macchina

  - Successivamente sviluppo di sistema di controllo dell'I/O Input-Output Control Systems (IOCS)
    - Librerie di codice già pronto per gestire i dispositivi I/O
    - Precursore di sistemi operativi

Allocazione di memoria contigua mono utente

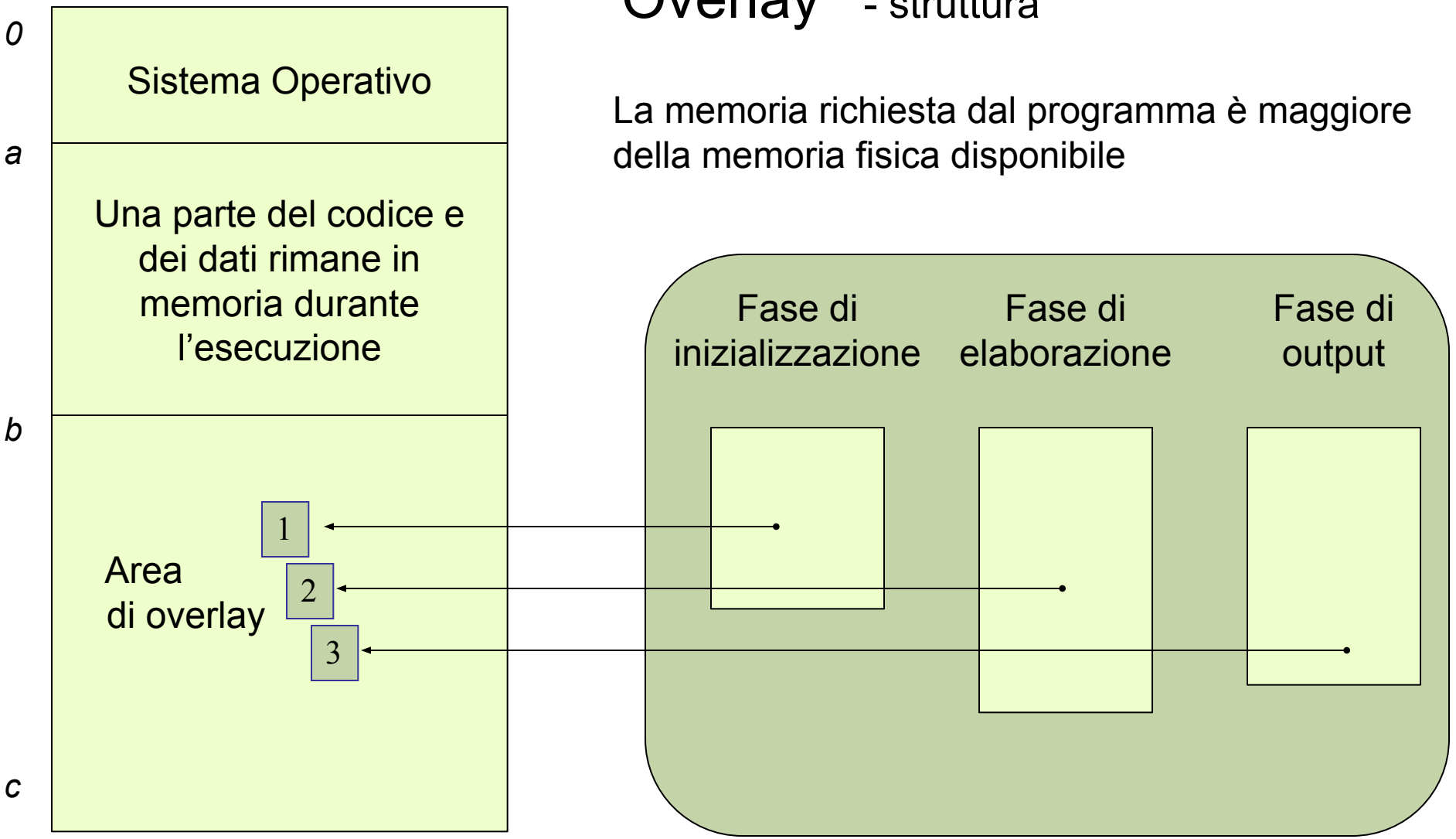


Overlay

- tecnica di programmazione per superare i limiti di allocazione contigua
  - Il programma è diviso in sezioni logiche
  - Si memorizzano soltanto le sezioni attive al momento
  - Svantaggi importanti
    - Difficile organizzare le sovrapposizioni (overlay) per utilizzare in modo efficiente la memoria principale
    - Complica la modifica ai programmi
  - La memoria virtuale ha un obiettivo simile
    - protegge i programmatori di questioni complesse come la gestione della memoria

Overlay - struttura

La memoria richiesta dal programma è maggiore della memoria fisica disponibile



1 Fase di inizializzazione a *b* e avvio

2 Carica la fase di esecuzione e avvio

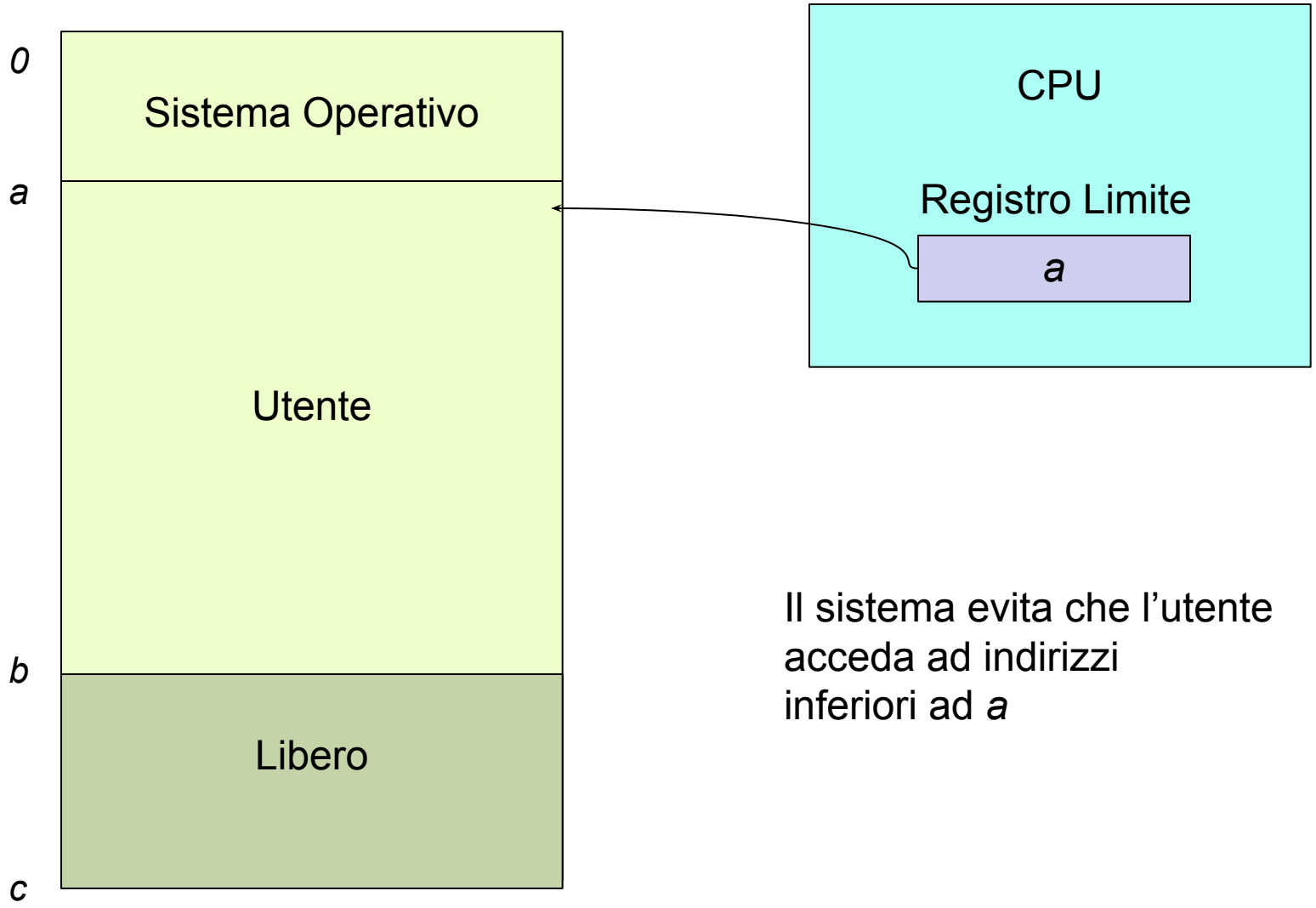
3 Carica la fase di output e avvio

Protezione in ambiente mono-utente

- S.O. non deve essere danneggiato da programmi
  - Il sistema non può funzionare se il S.O. viene sovrascritto
  - Registri limite (boundary)
    - Contiene l'indirizzo dove **inizia** lo spazio di memoria del programma
    - Ogni accesso alla memoria oltre al limite è negato
    - Può essere impostato solo da istruzioni privilegiate
    - Le applicazioni possono accedere alla memoria del S.O. per eseguire le procedure del S.O. utilizzando chiamate di sistema, che pone il sistema in modalità esecutiva
    - Rilocazione dinamica

Protezione in un ambiente mono-utente

in sistema contiguo



Il sistema evita che l'utente acceda ad indirizzi inferiori ad *a*

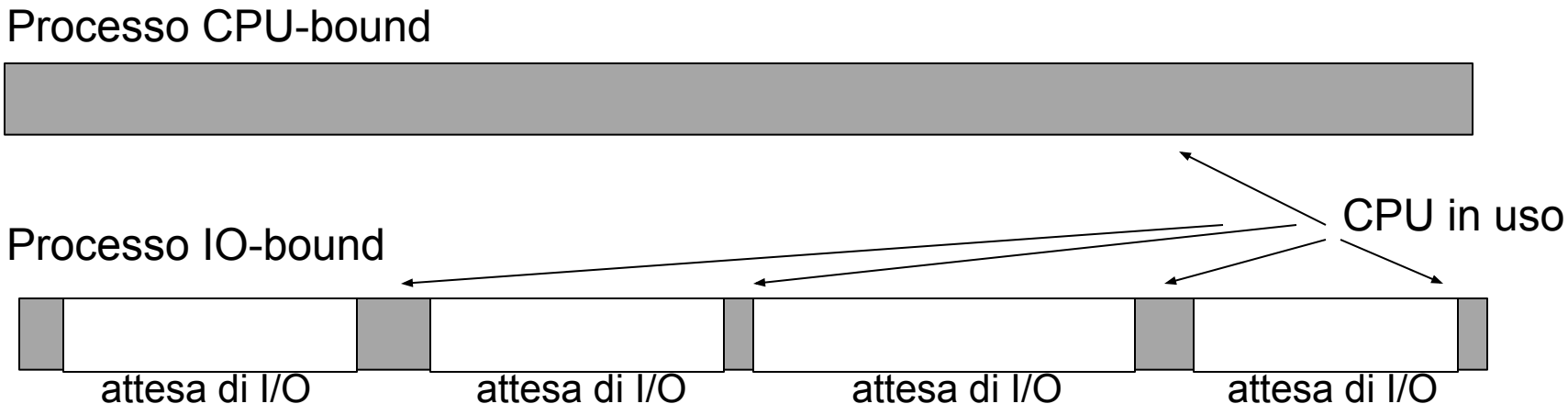
Single-Stream Batch Processing

- I primi sistemi richiedevano un tempo di *setup* rilevante
  - Spreco di tempo e risorse
  - Automatizzare *setup* e *teardown* porta ad una miglior efficienza

- *Batch processing*
  - Processore con un flusso di job stream letti in *job control language*
    - Definisce ogni job e come configurarlo

Multiprogrammazione a partizioni fisse

- Le richieste I/O possono vincolare un processore per lunghi periodi
  - La multiprogrammazione è una soluzione
    - Il processo che non usa attivamente un processore dovrebbe rilasciarlo ad altri
    - Richiede che diversi processi **risiedano** in memoria contemporaneamente

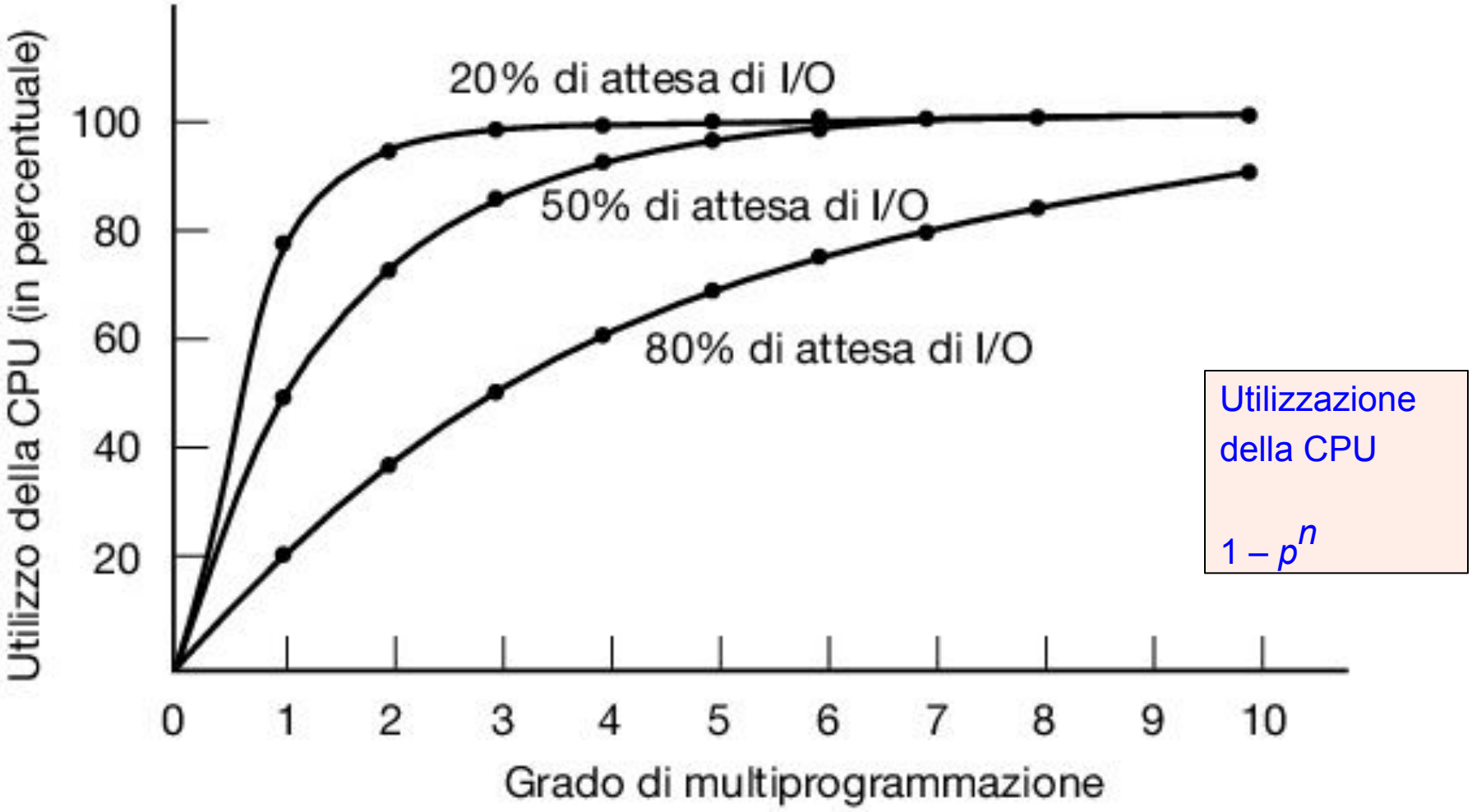


Utilizzo del processore su un sistema mono-utente.
Nota: In molti single-user job, l'attesa di I/O sono molto maggiori rispetto ai periodi di utilizzo del processore indicato in figura

Modello di multiprogrammazione

utilizzazione della CPU in funzione del grado di multiprogrammazione

n processi in memoria, *p* frazione di tempo di attesa di I/O (probabilità)
(processi indipendenti)



Multiprogrammazione a partizioni fisse

- Multiprogrammazione a partizioni fisse
  - Ogni processo attivo riceve un blocco di dimensioni fisse della memoria
  - Il processore passa rapidamente fra i processi
  - Illusione di simultaneità
  - Maggior richiesta di memoria
  - I registri *boundary* multipli proteggono dai possibili danni

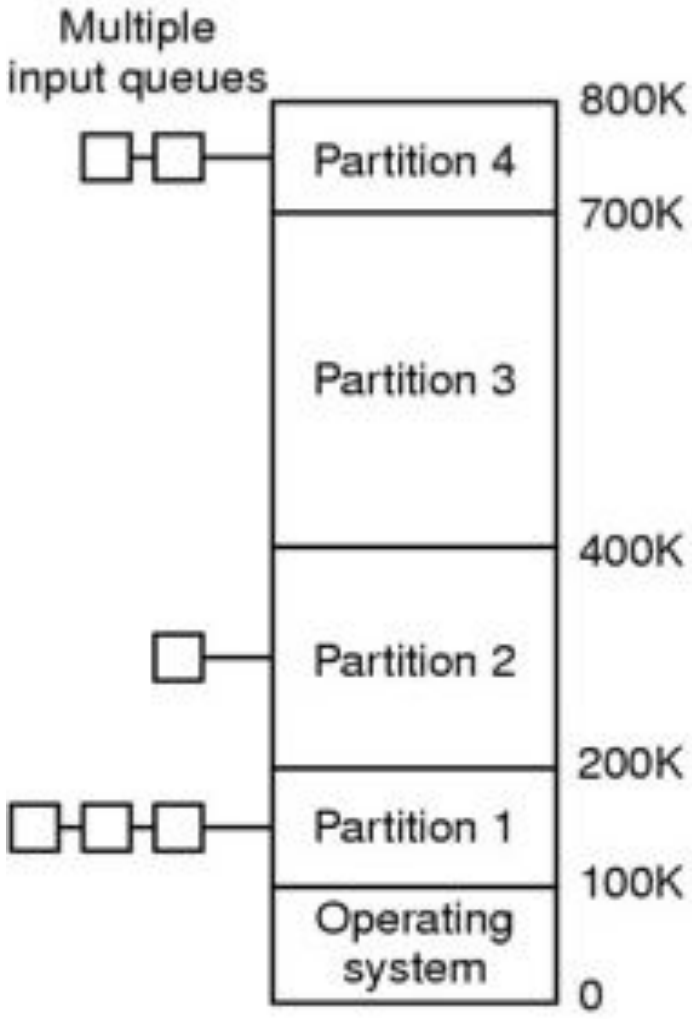
Multiprogrammazione a partizioni fisse

con compilazione e caricamento con indirizzi assoluti

Job eseguibili solo nella Partizione 4

Job eseguibili solo nella Partizione 2

Job eseguibili solo nella Partizione 1



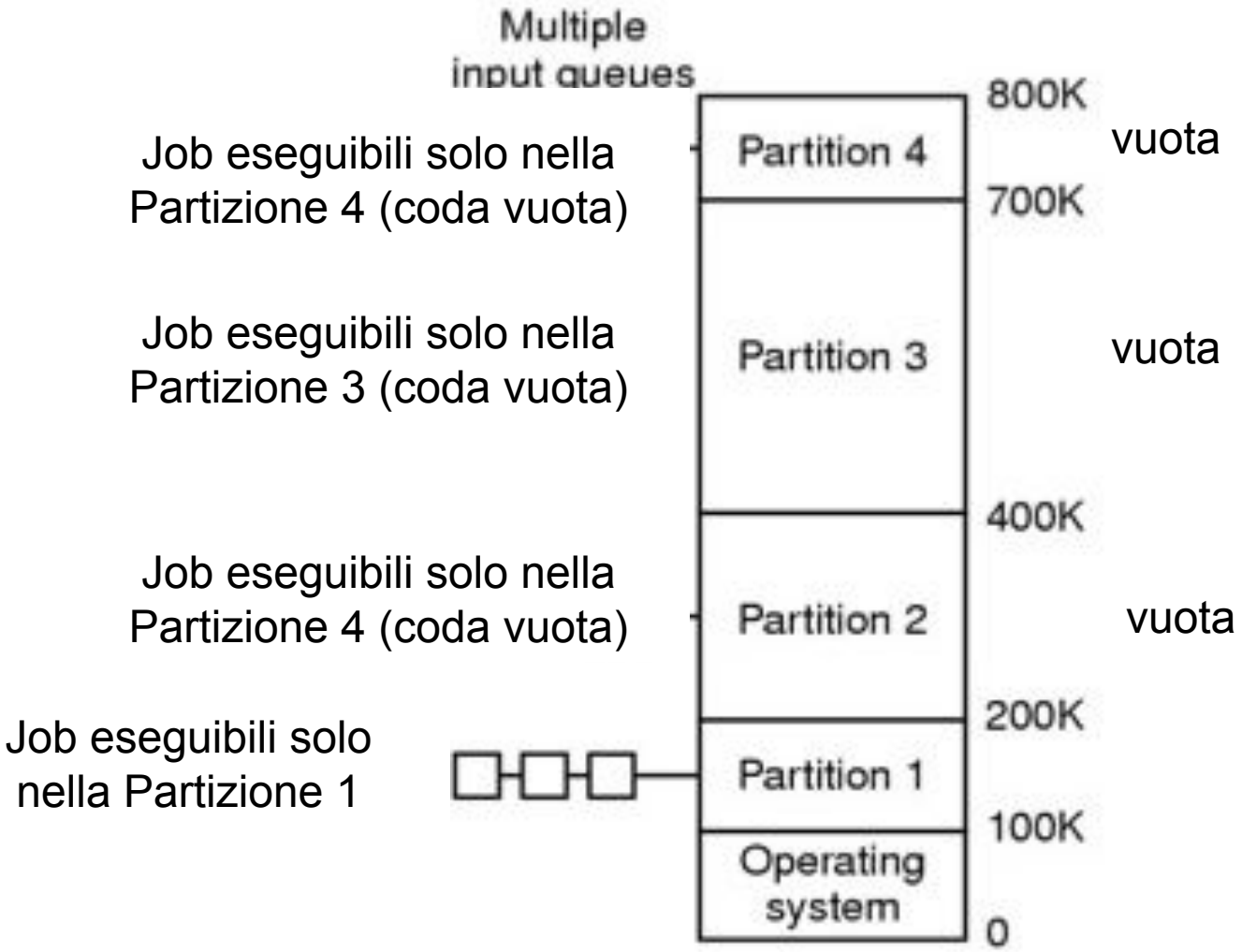
Code separate per le partizioni

Multiprogrammazione a partizioni fisse

- Svantaggi delle partizioni fisse
  - Le prime implementazioni usavano indirizzi assoluti
    - Se la partizione richiesta era occupata, il codice non poteva essere caricato
    - Successivamente il problema è stato superato con compilatori rilocanti
    - Sviluppo di compilatori con rilocazione, assemblatori, linker e loader: esecuzione in qualsiasi area di memoria
    - Maggior overhead

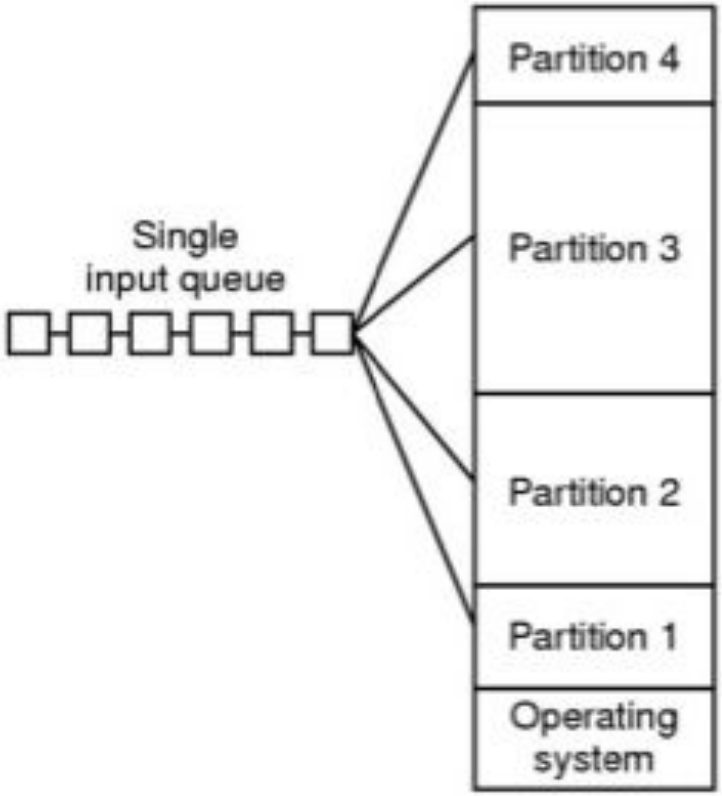
Multiprogrammazione a partizioni fisse

Spreco di memoria sotto multiprogrammazione con partizione fissa con compilazione e caricamento con indirizzi assoluti



Multiprogrammazione a partizioni fisse

con compilazione e caricamento con indirizzi rilocabili



Coda unica per tutte le partizioni

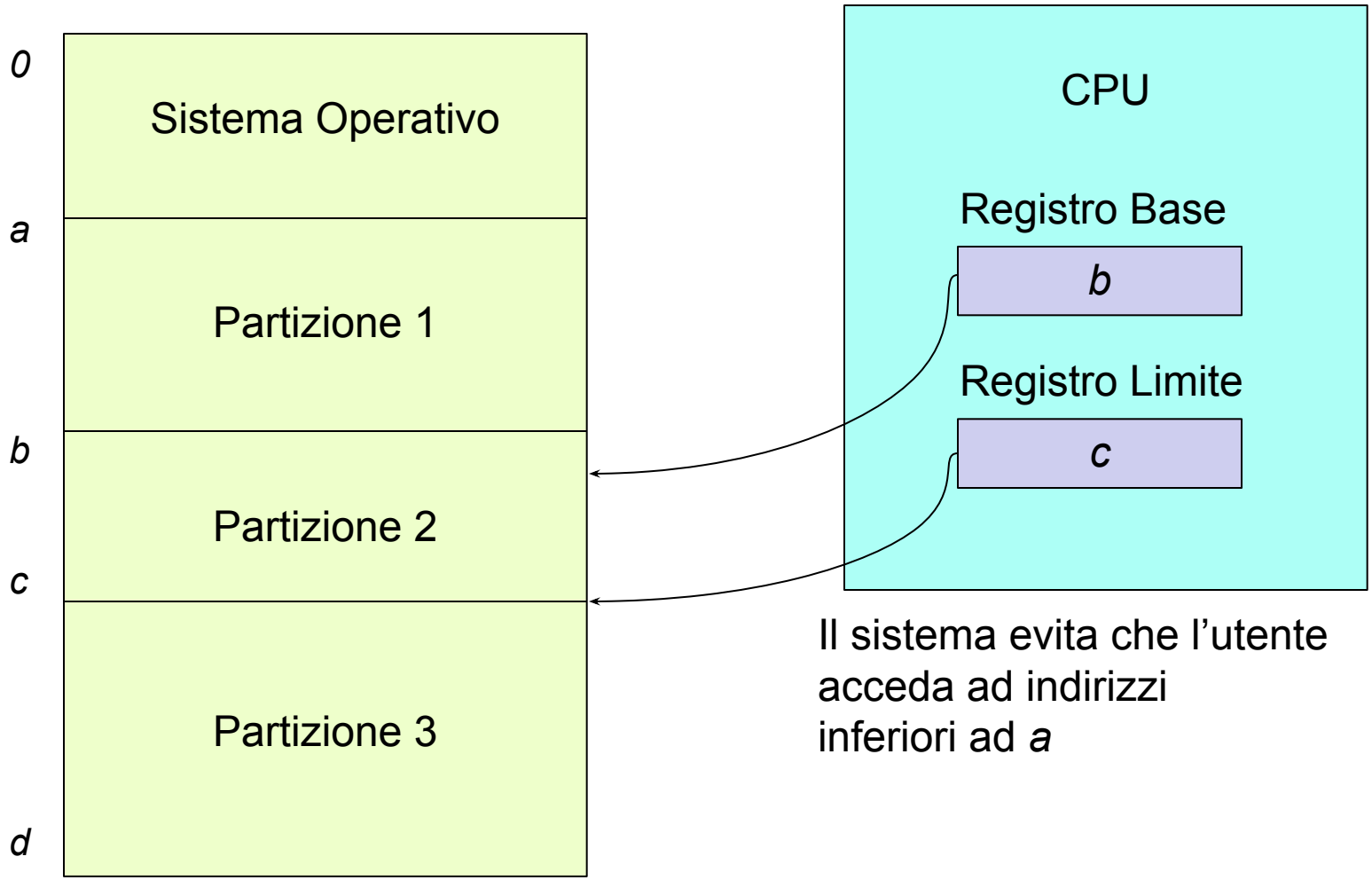
Si può eseguire un job in una partizione qualsiasi purché lo possa contenere

Multiprogrammazione a partizioni fisse

- Protezione
  - del S.O. da processo
  - del processo dagli altri processi
  - può essere implementato da più registri *boundary*, chiamati registro base e registro limite (anche basso e alto)
  - controllo che le richieste siano interne all'intervallo [base, limite]
  - chiamate di sistema per accedere ai servizi del S.O.

Multiprogrammazione a partizioni fisse

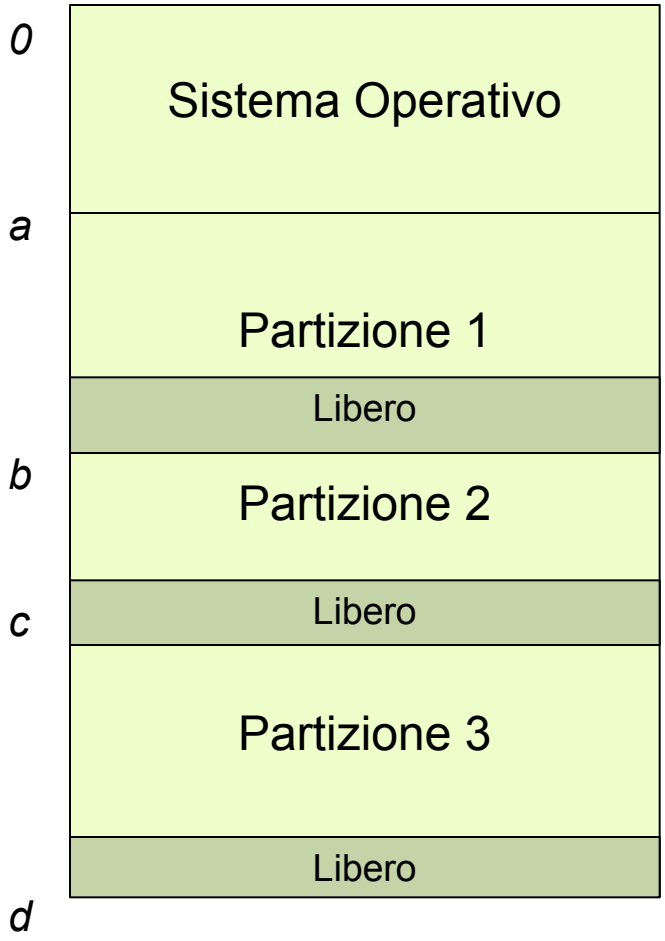
Protezione della memoria in sistemi multiprogrammati con allocazione contigua



Il sistema evita che l'utente acceda ad indirizzi inferiori ad *a*

Multiprogrammazione a partizioni fisse

- Svantaggi delle partizioni fisse
  - **Frammentazione interna**
    - Il processo non occupa un'intera partizione, sprecando memoria
    - Impossibilità di usare parte della memoria libera
    - Possibilità di avere processi troppo grandi da non per poter essere inseriti in nessuna parte
  - Maggior overhead
    - Compensati da maggiori utilizzo delle risorse



Multiprogrammazione a partizioni variabili

Alternativa alle partizioni fisse: progetto con partizioni **variabili**

- I job sono allocati dove possono essere contenuti

–Nessuno spazio sprecato all'inizio

–Impossible la frammentazione interna

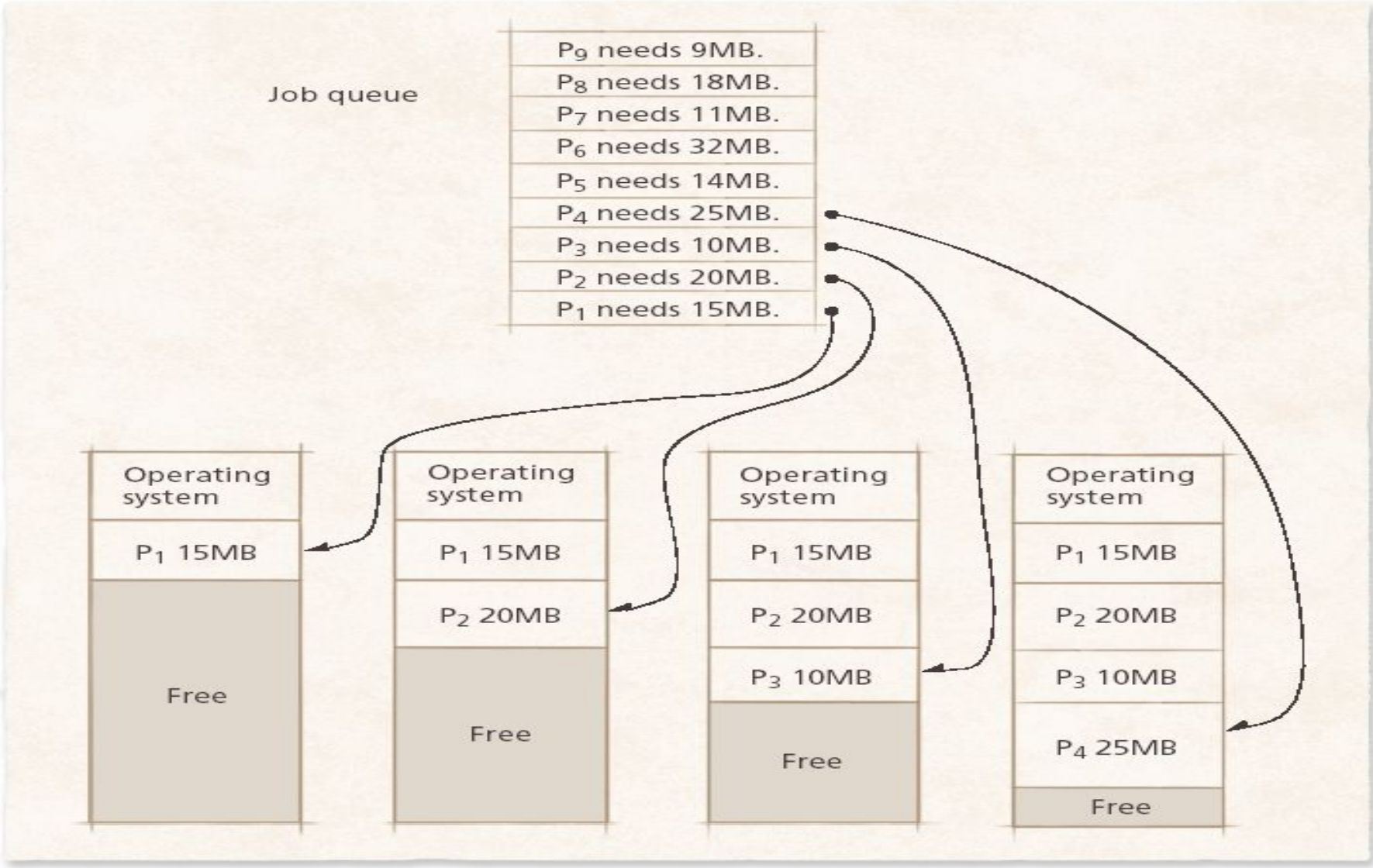
- Le partizioni sono esattamente delle dimensioni necessarie

–La **frammentazione esterna** può verificarsi quando i processi sono rimossi

- Lasciando 'buchi' troppo piccoli per nuovi processi
- Alla fine è possibile che non ci siano spazi ('buchi') abbastanza grandi per nuovi processi

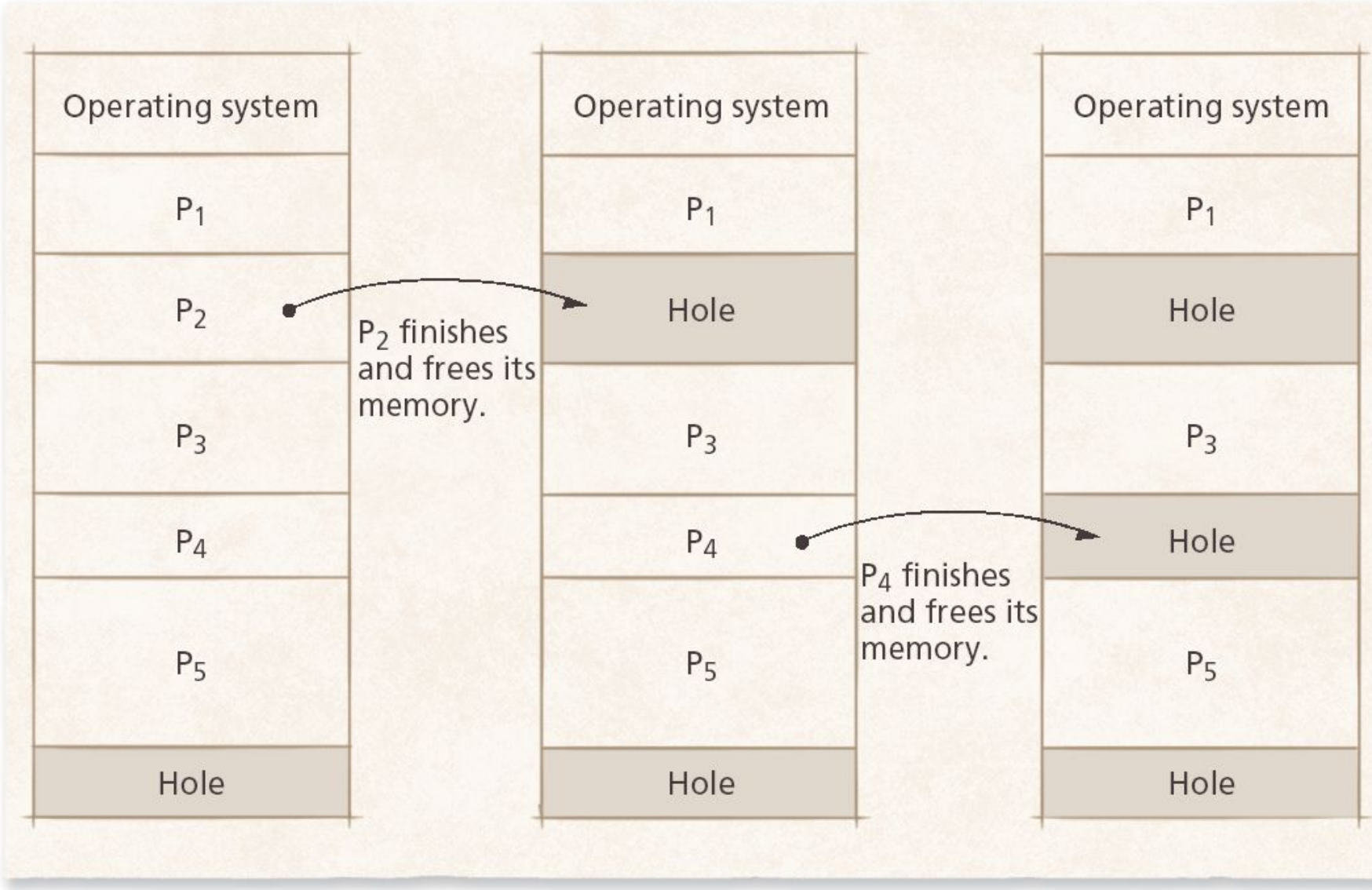
Multiprogrammazione a partizioni variabili

Assegnamento iniziale delle partizioni in multiprogrammazione a partizioni variabili



Partizioni variabili - caratteristiche

'Buchi' di memoria con multiprogrammazione a partizioni variabili

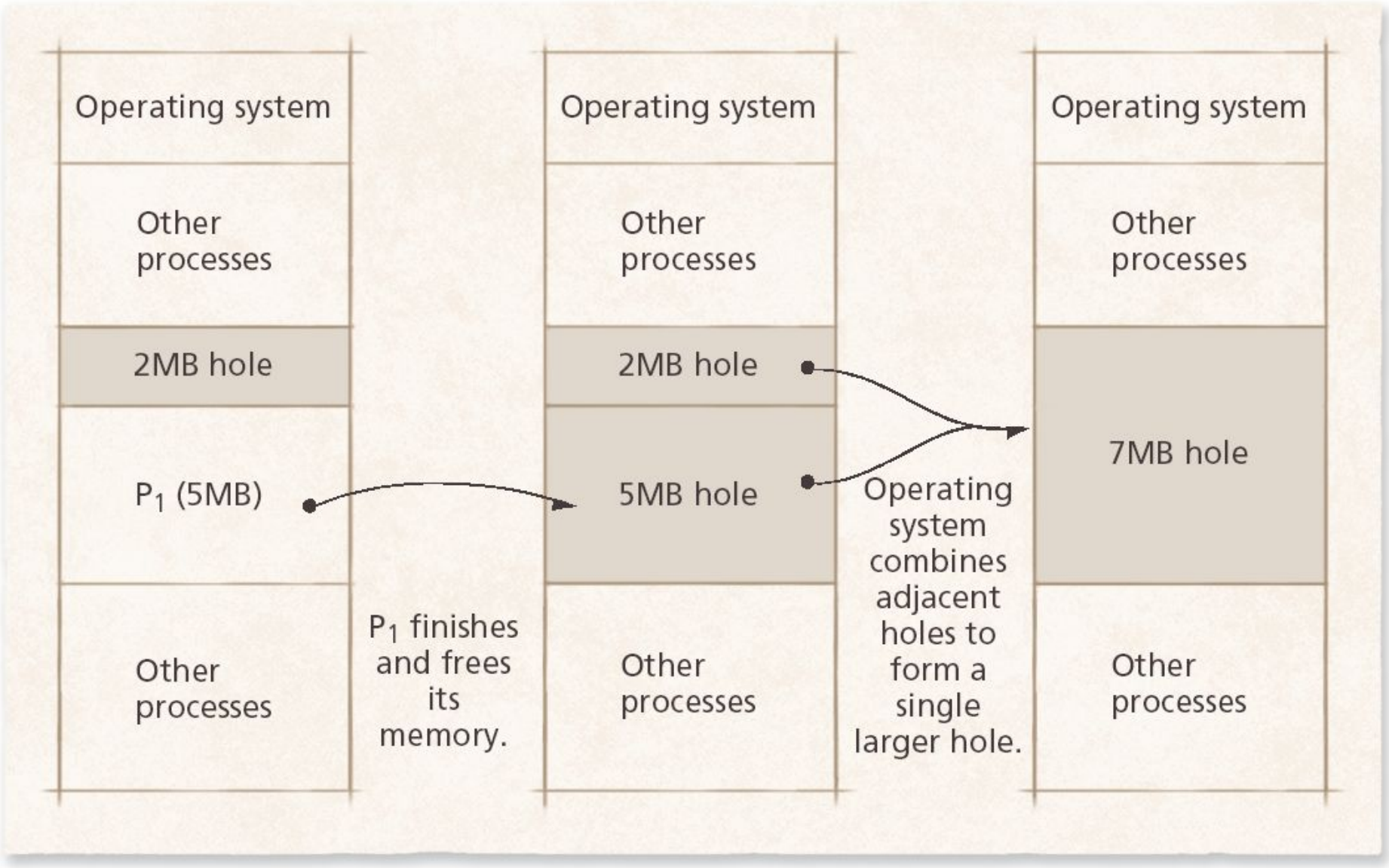


Partizioni variabili - caratteristiche

- Diversi modi per combattere la frammentazione esterna
  - **Coalescenza**
    - Combinare i blocchi liberi **adiacenti** in un unico grande blocco
    - Spesso non è sufficiente per recuperare quantità di memoria realmente significative
  - **Compattazione**
    - A volte chiamato garbage collection (da non confondere con il GC in linguaggi orientati agli oggetti)
    - Riorganizza la memoria in un unico blocco contiguo di spazio libero e un unico blocco contiguo di spazio occupato
    - Rende **tutto** lo spazio libero disponibile
    - Overhead significativo

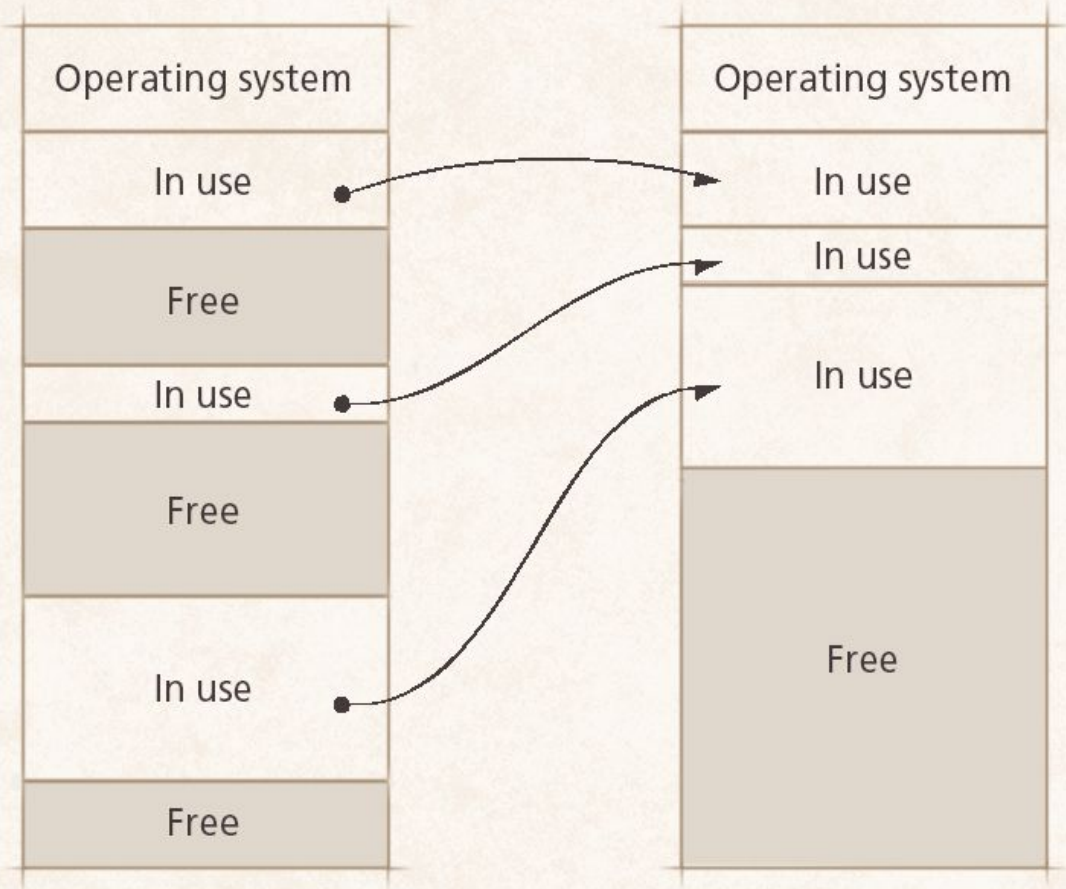
Partizioni variabili - caratteristiche

Coalescenza di 'buchi' di memoria in multiprogrammazione con partizioni variabili



Partizioni variabili - caratteristiche

Compattamento di memoria in multiprogrammazione con partizioni variabili



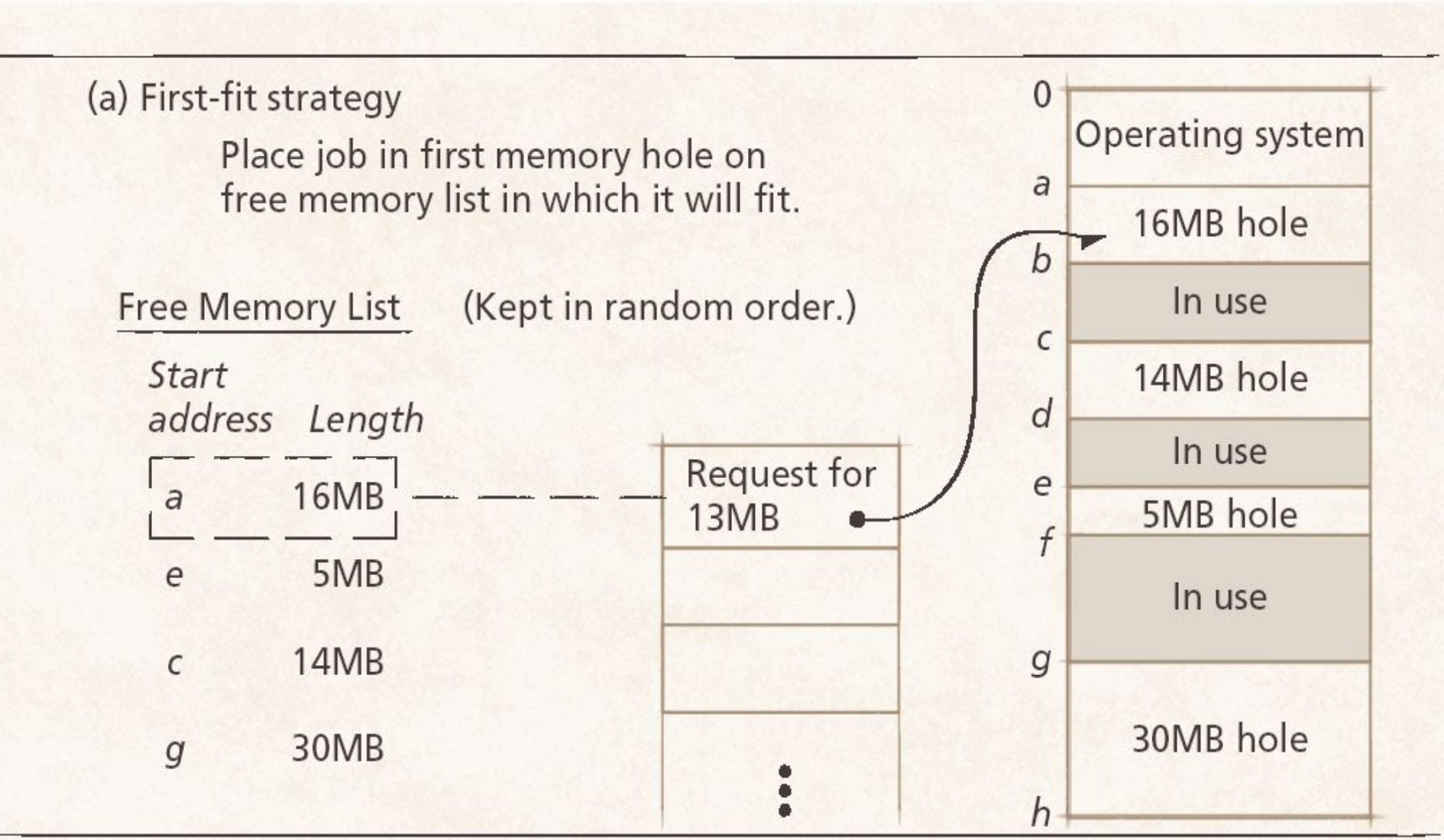
Operating system places all "in use" blocks together leaving free memory as a single large hole.

Strategie di posizionamento in memoria

- Dove mettere i processi in arrivo
  - Strategia **First-fit**
    - Il processo è allocato nel primo spazio libero trovato di dimensioni sufficienti
    - Semplice, basso overhead del tempo di esecuzione
  - Strategia **Best-fit**
    - Il processo è allocato nello spazio che lascia il minimo spazio inutilizzato
    - Maggior overhead del tempo di esecuzione
  - Strategia **Worst-fit**
    - Il processo è allocato in che lascia il massimo spazio inutilizzato
    - Lascia un altro grande 'buco', rendendo più probabile che un altro processo possa utilizzarlo

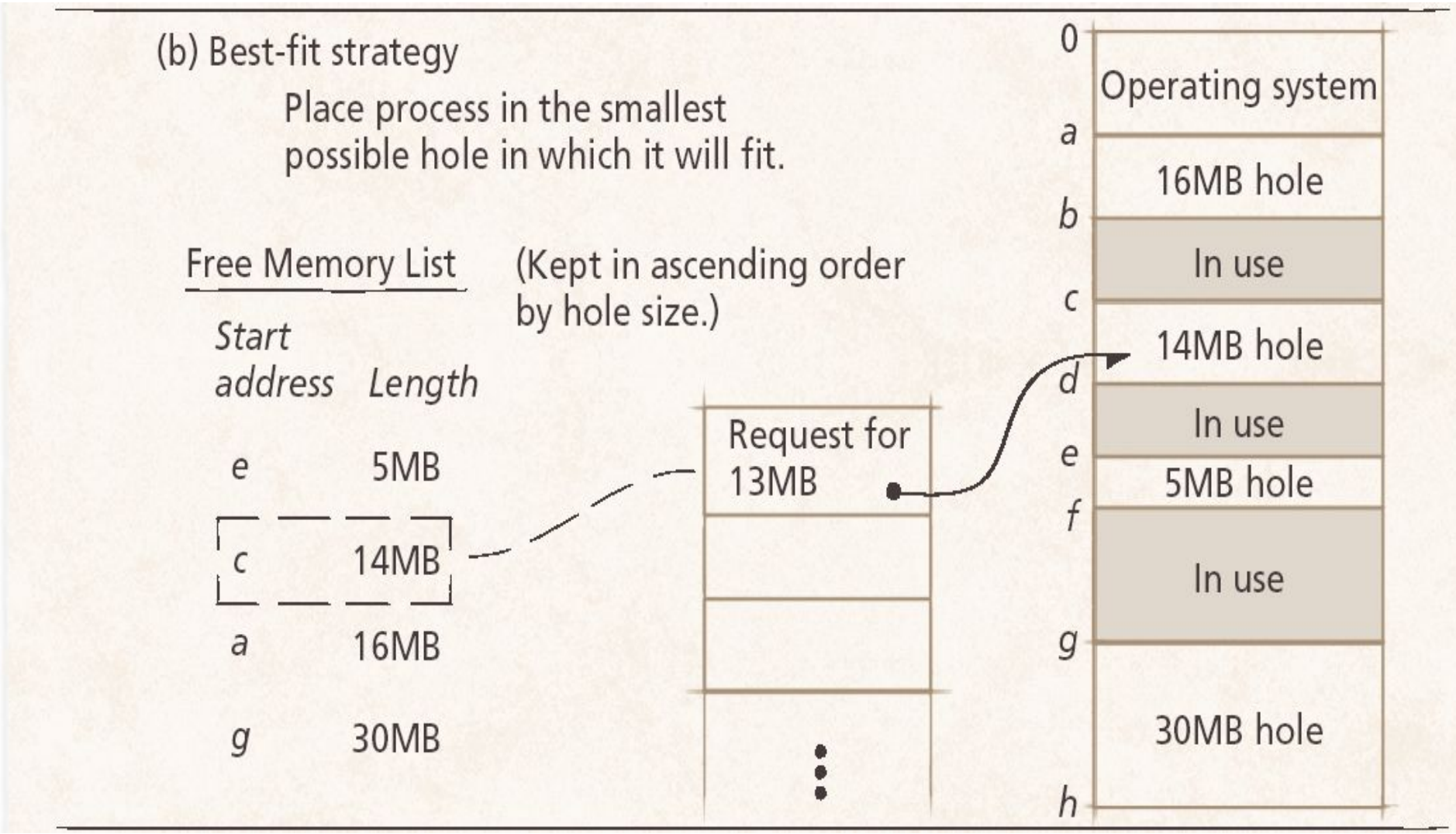
Strategie di posizionamento in memoria

Strategia **First-fit**



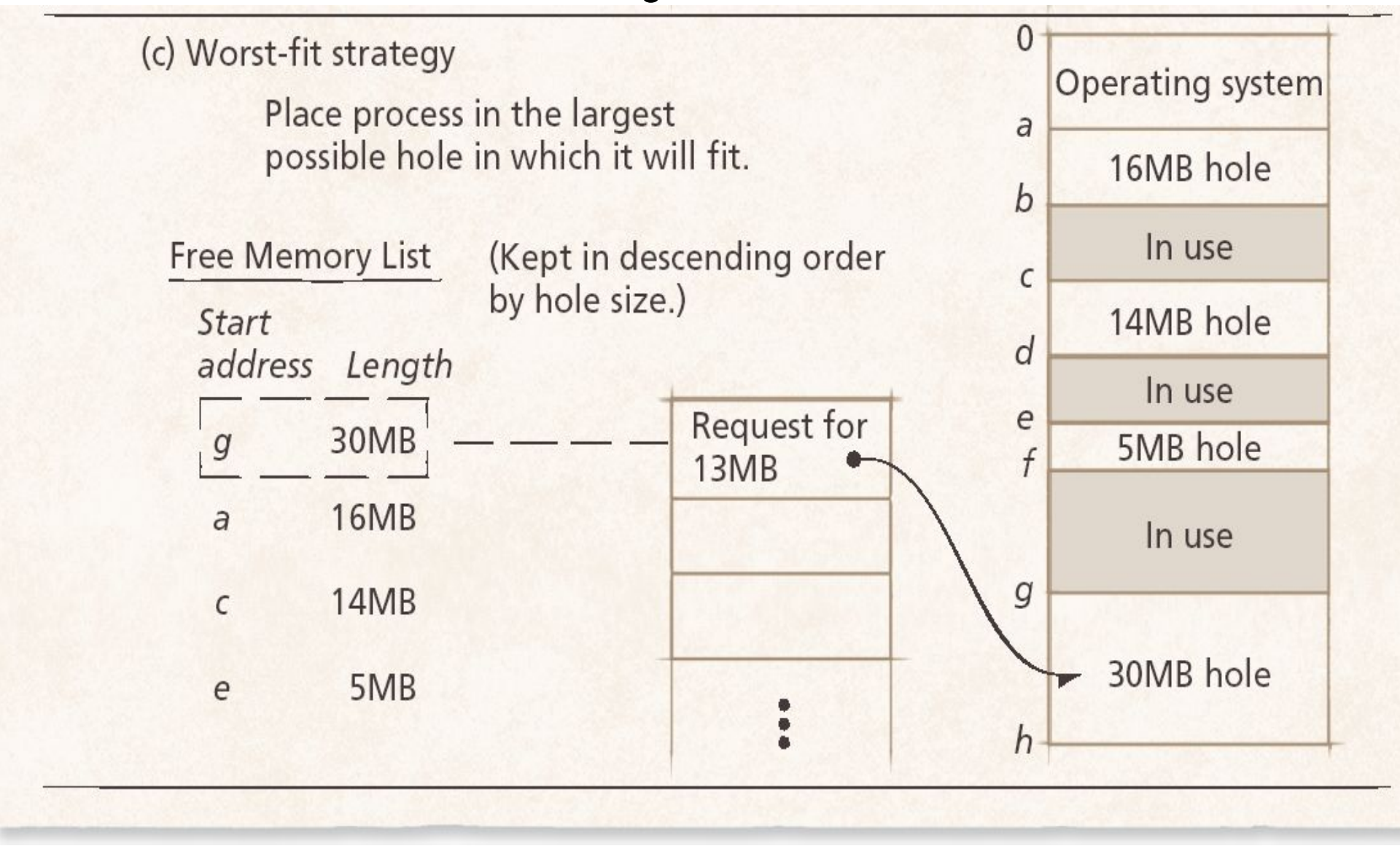
Strategie di posizionamento in memoria

Strategia **Best-fit**



Strategie di posizionamento in memoria

Strategia **Worst-fit**



Multiprogrammazione con Swapping di memoria

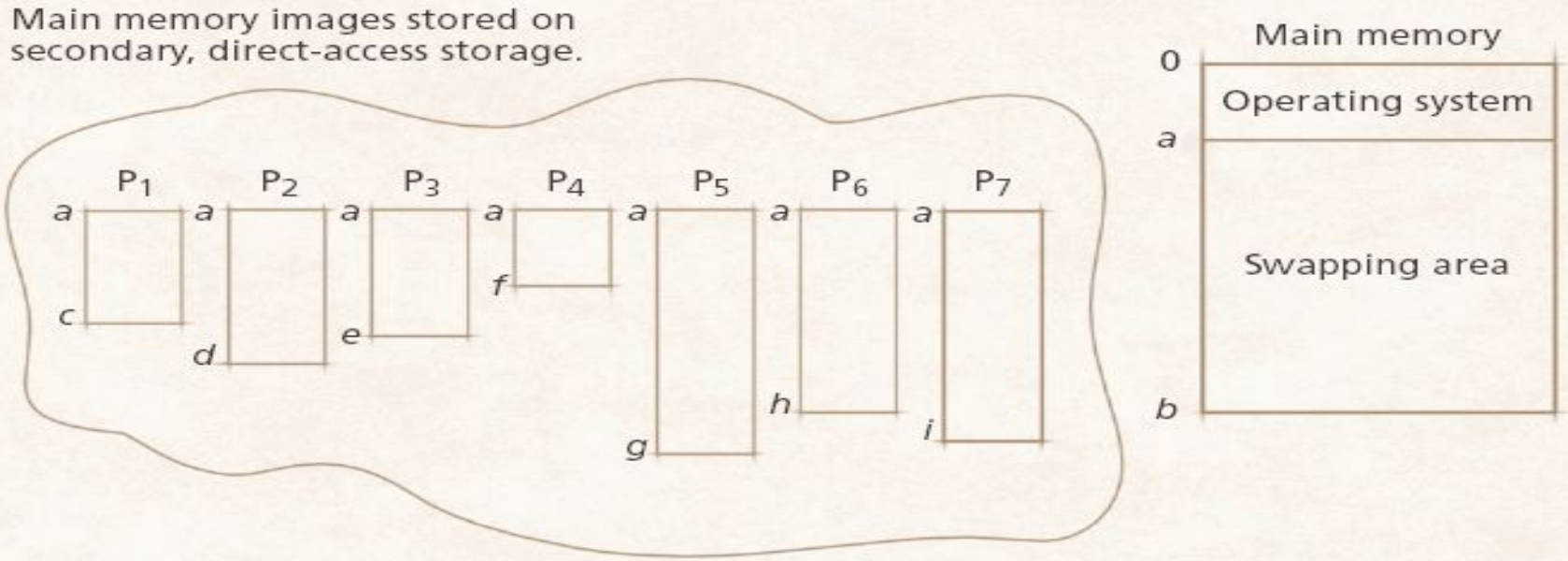
- Osservazione: non occorre mantenere i processi inattivi in memoria

  - Swapping
    - Solo il processo attualmente in esecuzione è in memoria principale
      - Gli altri sono temporaneamente spostati in memoria secondaria
      - Massimizza memoria disponibile
      - Overhead significativo al cambio di contesto

    - Soluzione ancora migliore: mantenere in memoria più processi in una sola volta
      - Meno di memoria disponibile
      - Tempi di risposta molto minori
      - Simile a *paging*

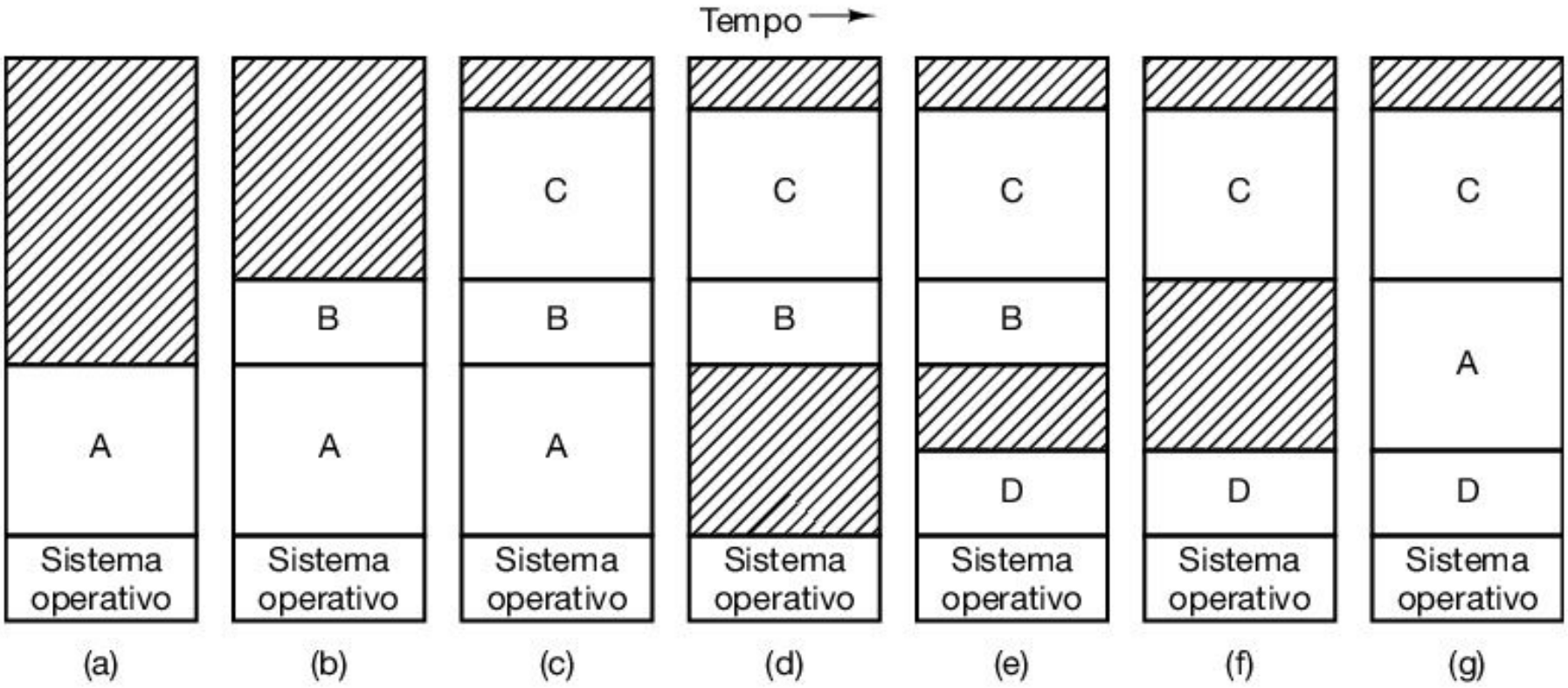
Multiprogrammazione con Swapping di memoria

Esempio: sistema in cui un solo processo per volta è in memoria principale



1. Only one process at a time resides in main memory.
2. That process runs until
   a) I/O is issued,
   b) timer runs out or
   c) voluntary termination.
3. System then swaps out the process by copying the swapping area (main memory) to secondary storage.
4. System swaps in next process by reading that process's main memory image into the swapping area. The new process runs until it is eventually swapped out and the next user is swapped in, and so on.

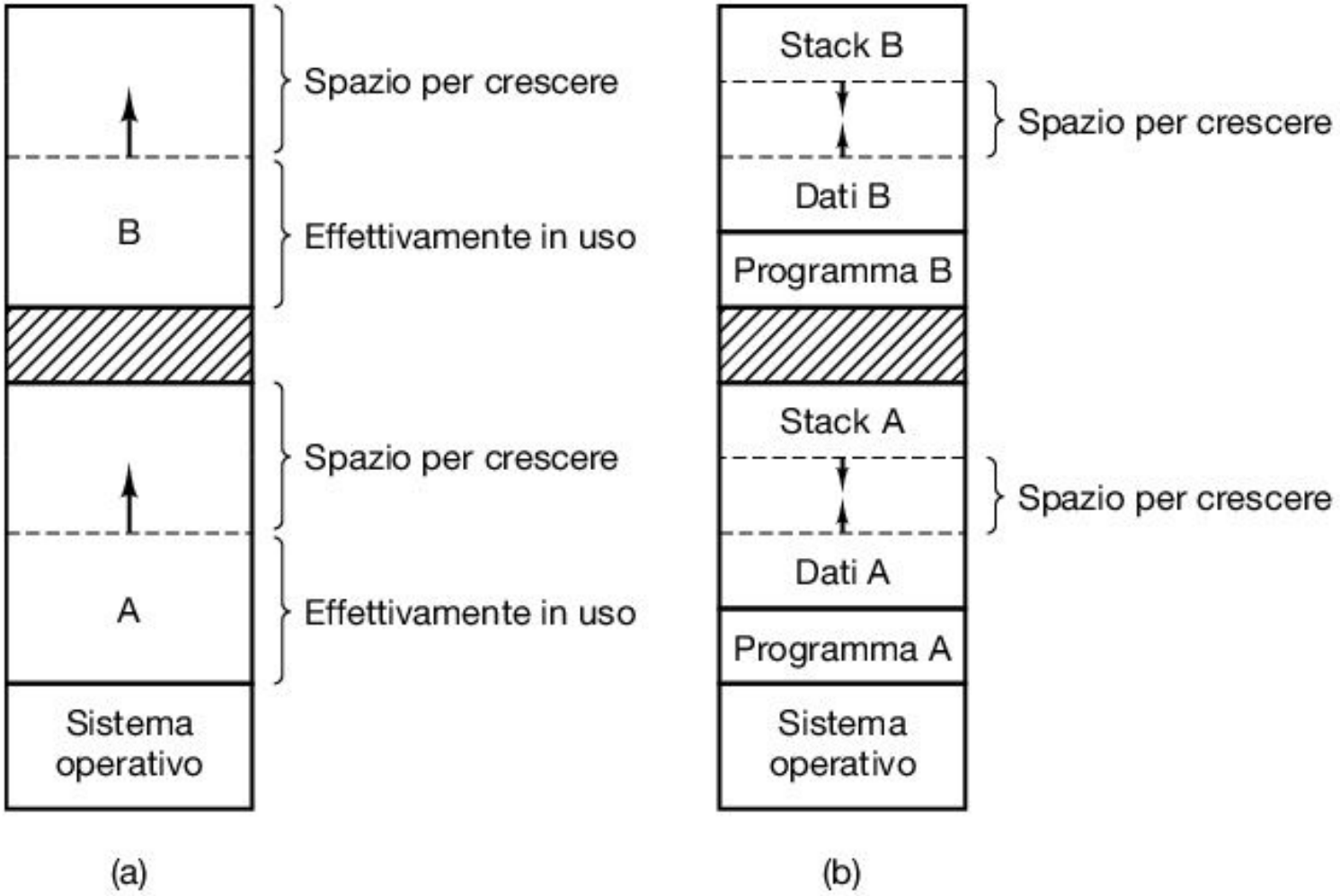
Multiprogrammazione con Swapping di memoria



Allocazione della memoria: modifiche a
- all'arrivo dei processi in memoria
- al rilascio dei processi dalla memoria

Le aree in grigio sono di memoria libera

Multiprogrammazione con Swapping di memoria



a) Allocazione dello spazio per un segmento di dati che cresce

b) Allocazione dello spazio per uno stack che cresce e un segmento di dati che cresce

Gestione della memoria libera

- L'allocazione dinamica della memoria richiede la gestione della memoria libera

  – Mappa di bit

    Memoria organizzata in unità

    Ad ogni unità corrisponde un bit nella mappa

  – Liste collegate

    Lista dei segmenti di memoria allocato e liberi

    Ogni segmento o è allocato ad un processo o è libero

    Ogni elemento della lista indica

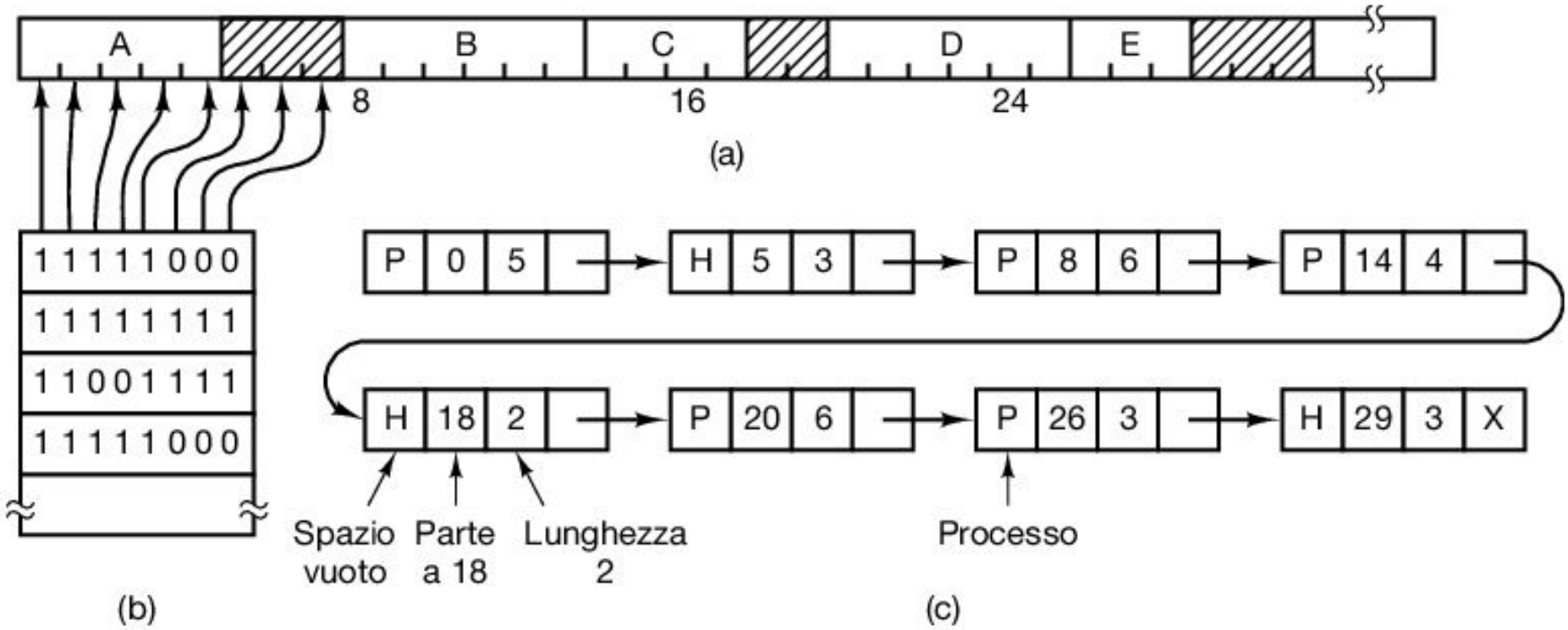
      se processo (P) o vuoto (H)

      l'indirizzo da cui parte

      la lunghezza

      il puntatore al successivo elemento

Gestione della memoria libera con mappa di bit e liste



Parte di memoria con 5 processi, 3 spazi liberi

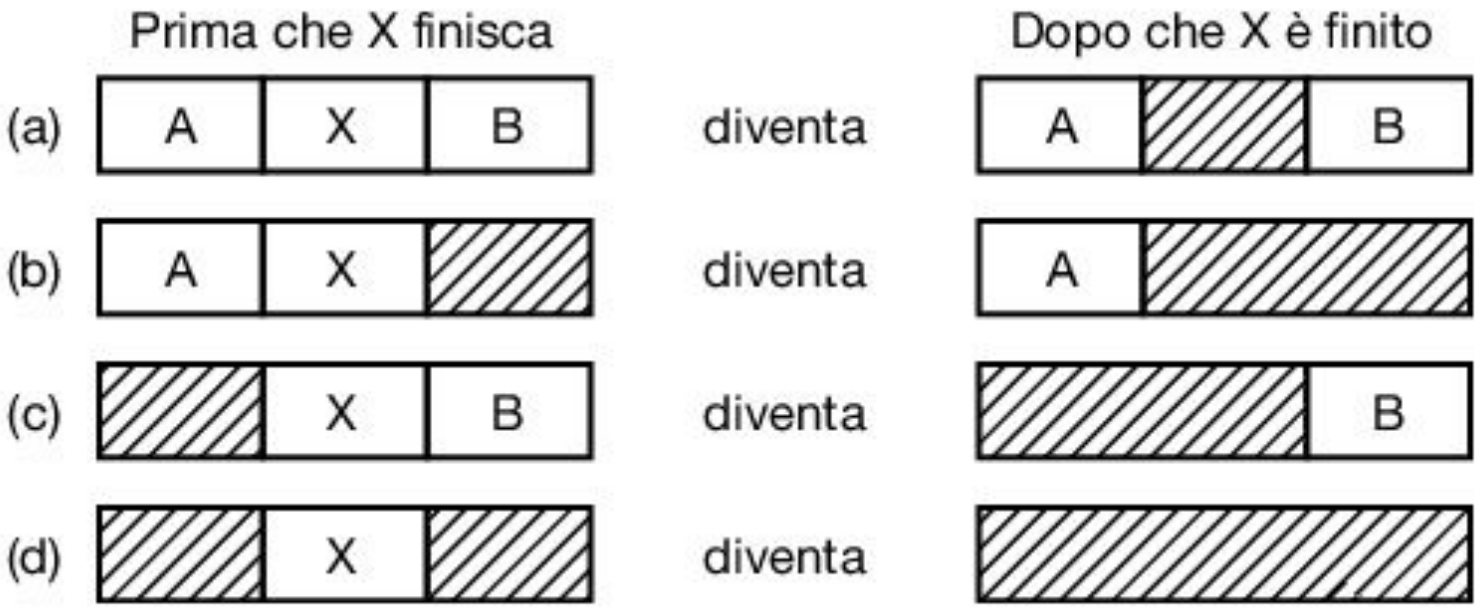
Le posizioni indicano le unita di allocazione di memoria

Le aree in grigio indicano la memoria libera

b) Mappa di bit corrispondente

c) Informazioni della mappa di bit in forma di lista collegata

Gestione della memoria libera con liste collegate



Quattro possibili combinazioni di vicini per il processo X in fase di chiusura