Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2021-2022

Pietro Frasca

Lezione 16

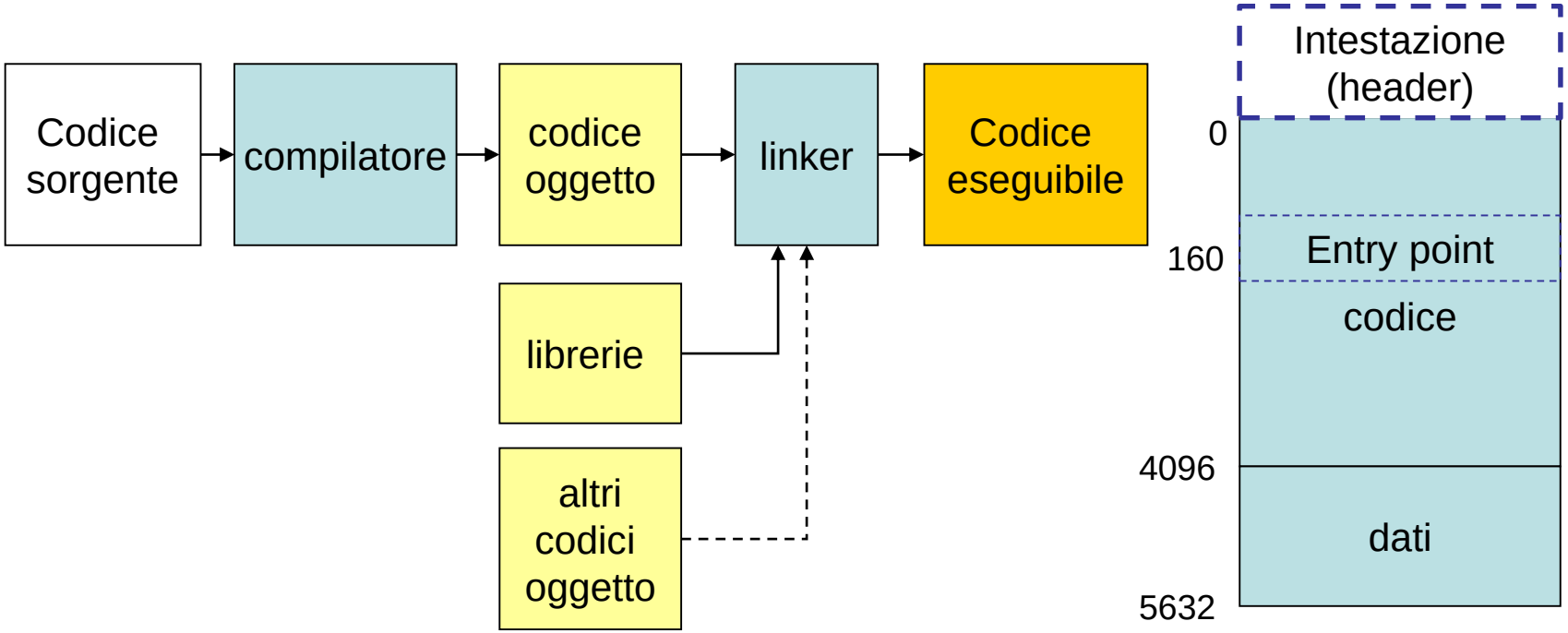
Giovedì 2-12-2021

Gestione della memoria

- La memoria principale è una risorsa importante che va gestita in modo efficiente. Il componente che gestisce la memoria è detto ***gestore della memoria***. Il suo compito è tenere traccia di quali parti di memoria sono in uso, allocare la memoria ai processi in base alle loro richieste e liberarla quando i processi terminano.
- Analizzeremo varie tecniche di gestione della memoria dalle più semplici alle più complesse che dipendono sia dall'architettura del calcolatore, in particolare del processore, sia dall'architettura software del SO.
- Prima di mostrare le tecniche introduciamo alcuni concetti e definizioni basilari.

Creazione di un file eseguibile

- Un programma è costituito da un in insieme di moduli scritti mediante un linguaggio (anche più di uno) di programmazione (**codice sorgente**). Ciascun modulo sorgente viene compilato (dal compilatore) producendo il modulo oggetto il quale viene collegato con eventuali altri moduli oggetto e/o librerie dal **linker**, producendo il **codice eseguibile** che viene caricato in memoria dal **caricatore**.



Lo **spazio virtuale di un processo**: memoria necessaria a contenere il suo codice, i dati su cui operare e lo stack (compreso l'heap).

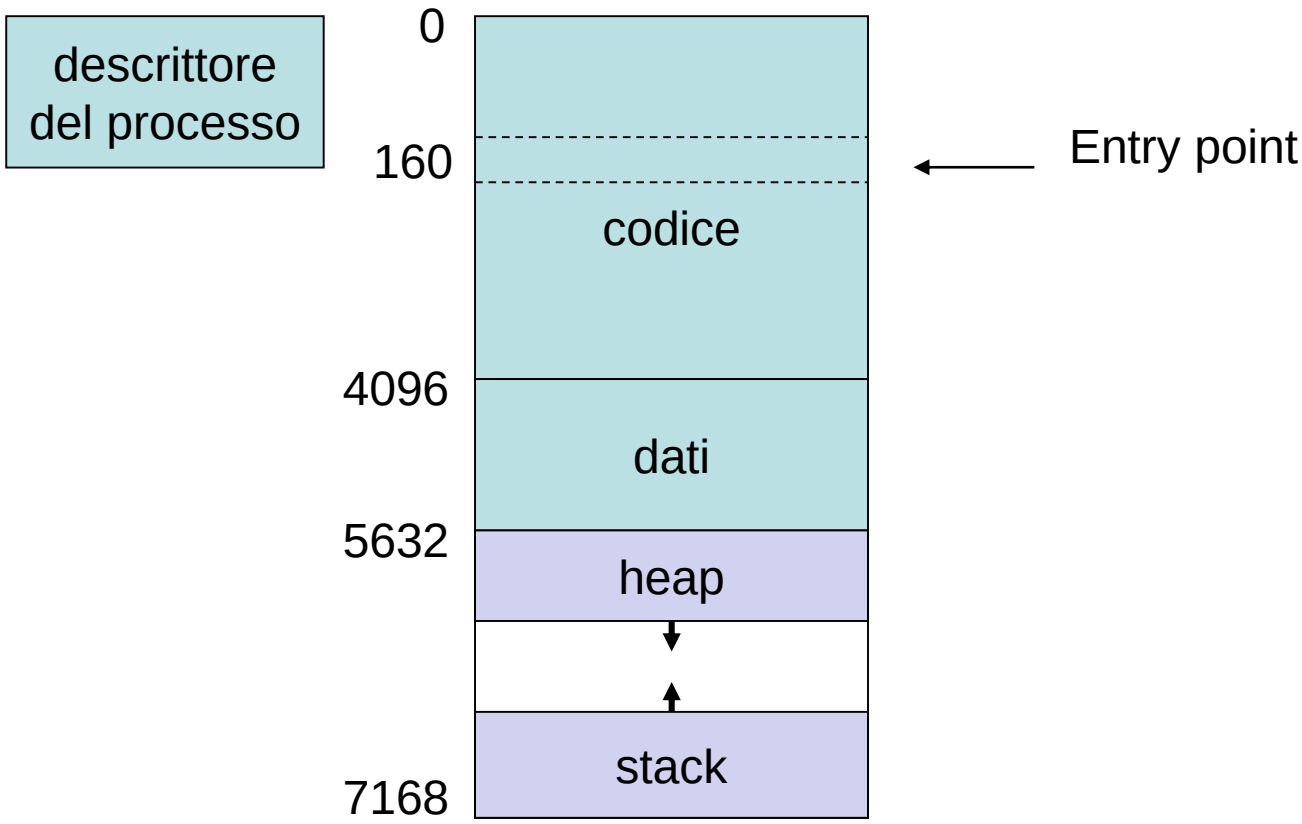


Immagine di un processo

Organizzazione della spazio virtuale

- **Spazio virtuale unico**

  Inizialmente, consideriamo il caso in cui lo spazio virtuale del processo sia costituito da **un unico spazio** di indirizzi virtuali (indirizzi compresi tra 0 e 7168).

  Supponiamo che il linker generi indirizzi contigui per tutti i moduli che compongono il programma.

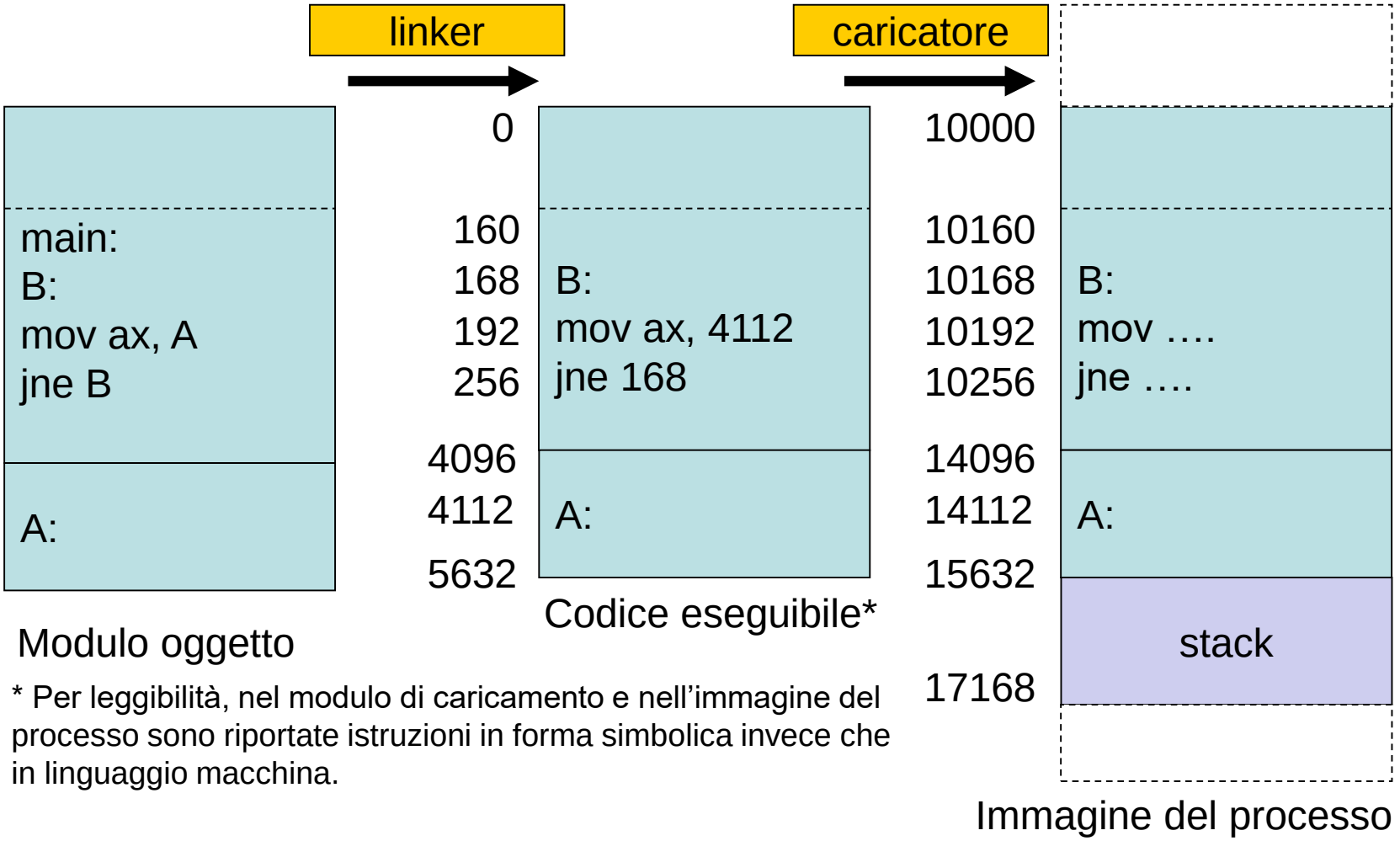
  In particolare la struttura del processo visto è composta da tre parti, dette **segmenti**, distinte:

  - **segmento del codice (text)**
  - **segmento dati**
  - **segmento dello stack (non consideriamo l'heap)**

Rilocazione statica e dinamica

- Generalmente, lo spazio virtuale di un processo sarà caricato in memoria fisica a partire da qualsiasi indirizzo **l** diverso da **0**.
- Pertanto distinguiamo gli indirizzi di memoria virtuale di un processo detti ***indirizzi virtuali (o logici)***) dagli indirizzi di memoria fisica (***indirizzi fisici***).
- Ad esempio, nel caso di un processo con un immagine come nella figura precedente, l'insieme di tutti gli indirizzi virtuali del processo (***spazio degli indirizzi virtuali***) è dato dai valori compresi tra 0 e 7168 (7 KB). Nel caso in cui il processo fosse caricato in memoria a partire dall'indirizzo **10000**, gli indirizzi fisici ad esso associati (***spazio degli indirizzi fisici***) sarebbero compresi tra **10000** e **17168** (10000+7168).
- I due spazi di indirizzi non sono indipendenti ma sono legati da una funzione detta **funzione di rilocazione.**

Generazione della memoria virtuale e dell'immagine del processo



* Per leggibilità, nel modulo di caricamento e nell'immagine del processo sono riportate istruzioni in forma simbolica invece che in linguaggio macchina.

$$If = f(Iv)$$

- La funzione di rilocazione fa corrispondere ad ogni indirizzo virtuale **Iv** un indirizzo fisico **If**.
- Un esempio di funzione di rilocazione è data da:

$$If = Iv + I0$$

dove **I0** è una costante che indica l'indirizzo di base.

- Nel esempio mostrato la funzione di rilocazione è la seguente:

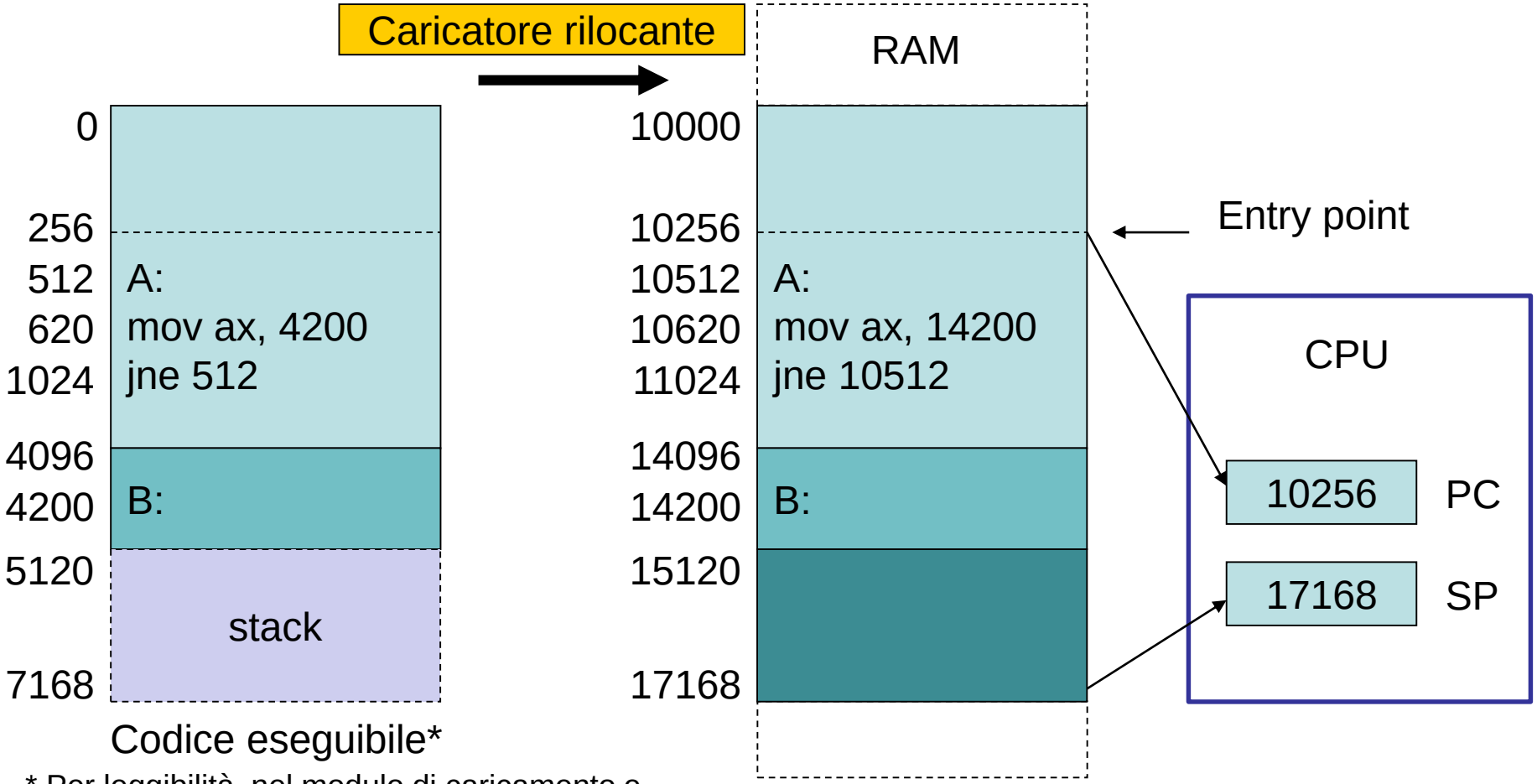
$$If = Iv + 10000$$

- Le tecniche per effettuare la rilocazione degli indirizzi sono di due tipi: **rilocazione statica** e **rilocazione dinamica.**

Rilocazione statica

- Tecnica più semplice. Tutti gli indirizzi virtuali sono rilocati prima che il processo inizi la sua esecuzione.
- La rilocazione è eseguita dal caricatore (***caricatore rilocante***) durante la fase di caricamento, modificando tutti gli indirizzi da virtuali a fisici che sono ottenuti sommando a ciascun indirizzo virtuale l'indirizzo iniziale di caricamento.
- Nel momento della creazione del processo nel registro **PC** è caricato l'indirizzo fisico dell'**entry point** del programma e nello **SP** l'indirizzo fisico relativo alla **base dello stack**.
- La figura seguente mostra l'immagine di un processo rilocata a partire dall'indirizzo 10000.

Rilocazione statica



* Per leggibilità, nel modulo di caricamento e nell'immagine del processo sono riportate istruzioni in assembler invece che in linguaggio macchina.

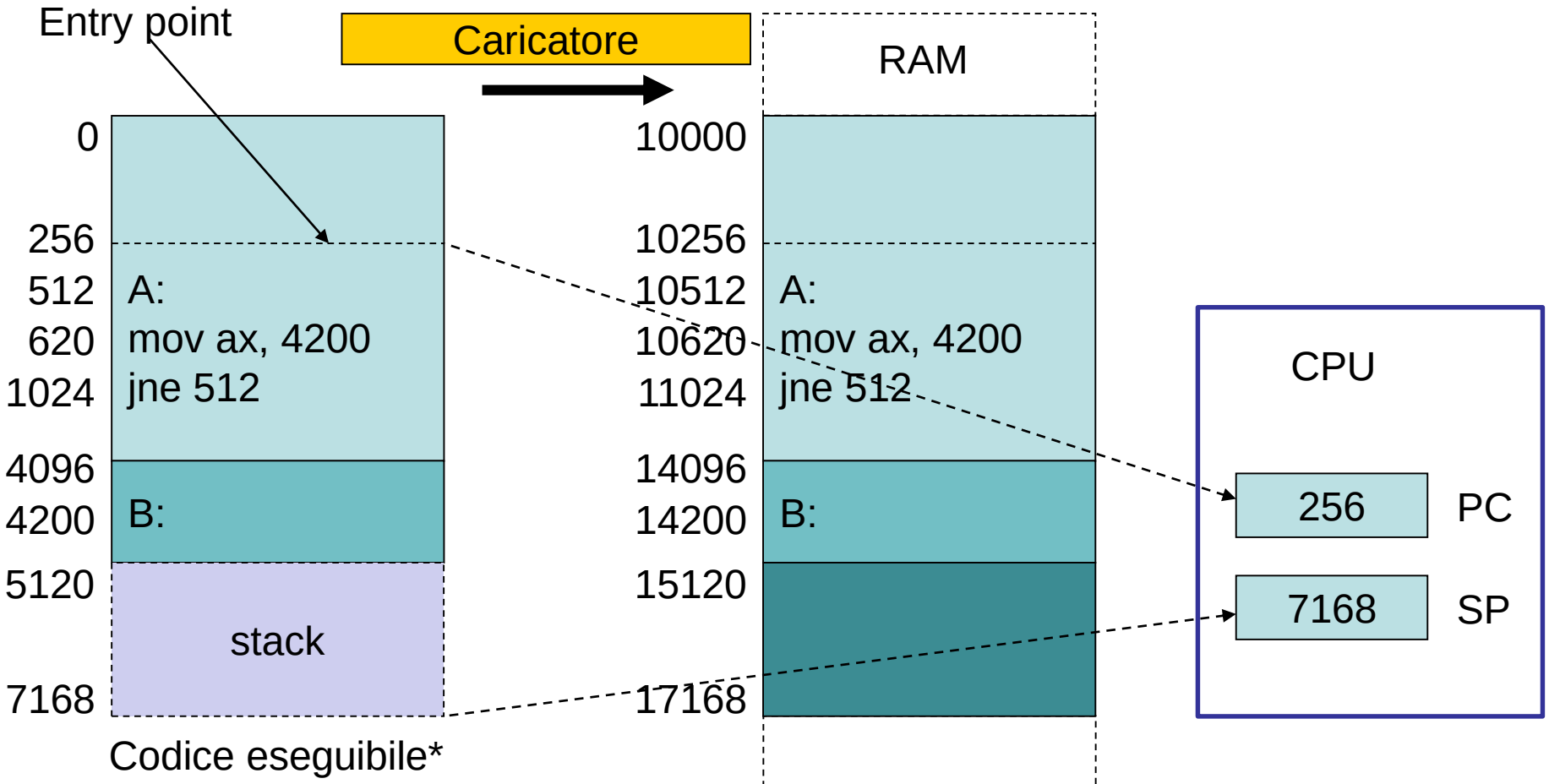
Rilocazione dinamica

- Con la rilocazione dinamica, la traduzione degli indirizzi da virtuali a fisici avviene durante l'esecuzione del processo.
- Il caricatore trasferisce in memoria direttamente il contenuto del codice eseguibile contenente gli indirizzi virtuali, senza modificarli.
- Nei registri **PC (**program counter) e **SP** (stack pointer) sono caricati i valori degli indirizzi virtuali relativi all'entry point del programma e della base dello stack.
- L'architettura di un processore che consente di effettuare la rilocazione dinamica deve avere un supporto hardware (spesso chiamato **MMU** (**Memory Management Unit**)) che implementa la funzione di rilocazione.

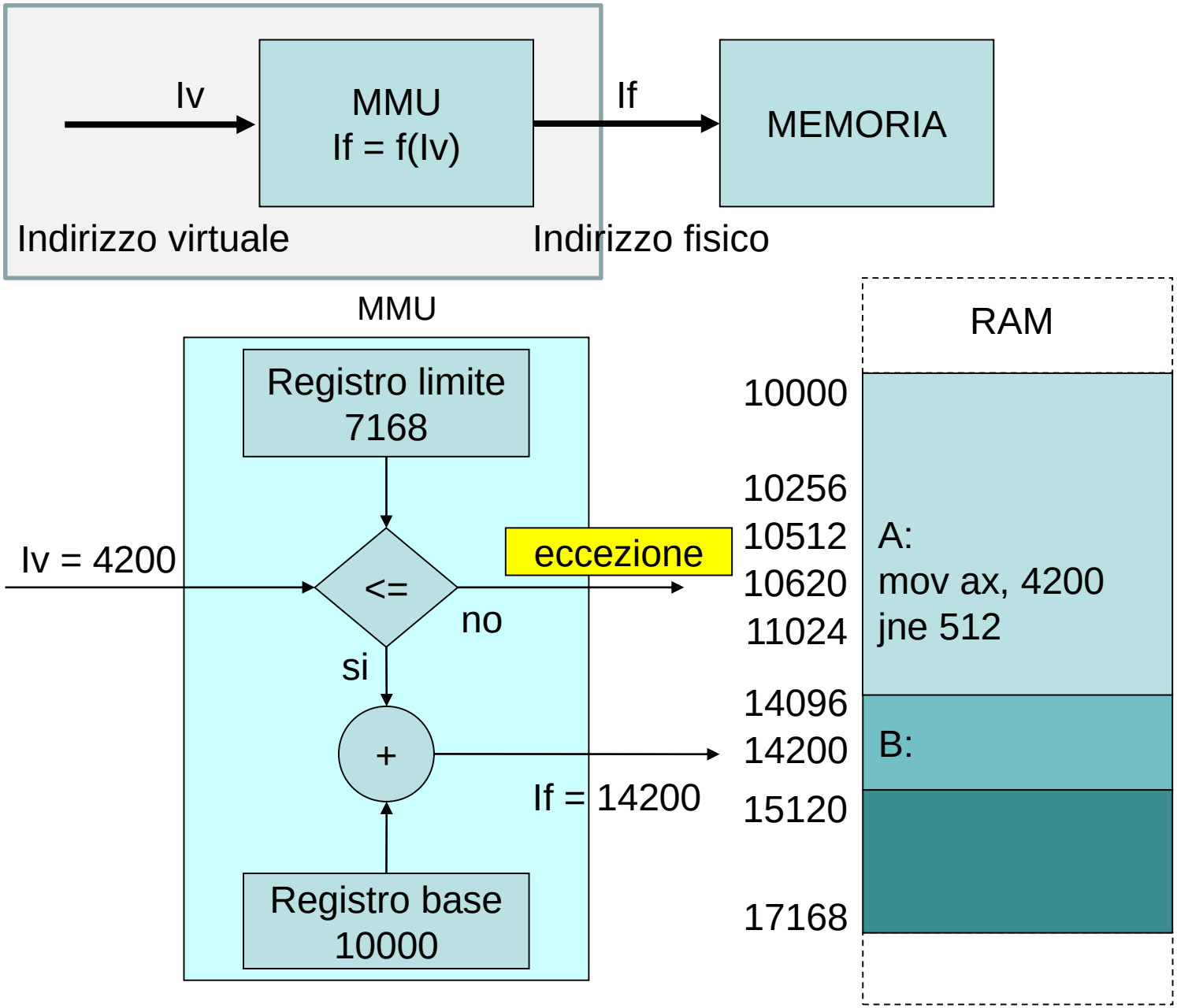
  If = f(Iv)

  traducendo dinamicamente ogni indirizzo virtuale nel corrispondente indirizzo fisico.

Rilocazione dinamica



* Per leggibilità, nel modulo di caricamento e nell'immagine del processo sono riportate istruzioni in assembler invece che in linguaggio macchina.

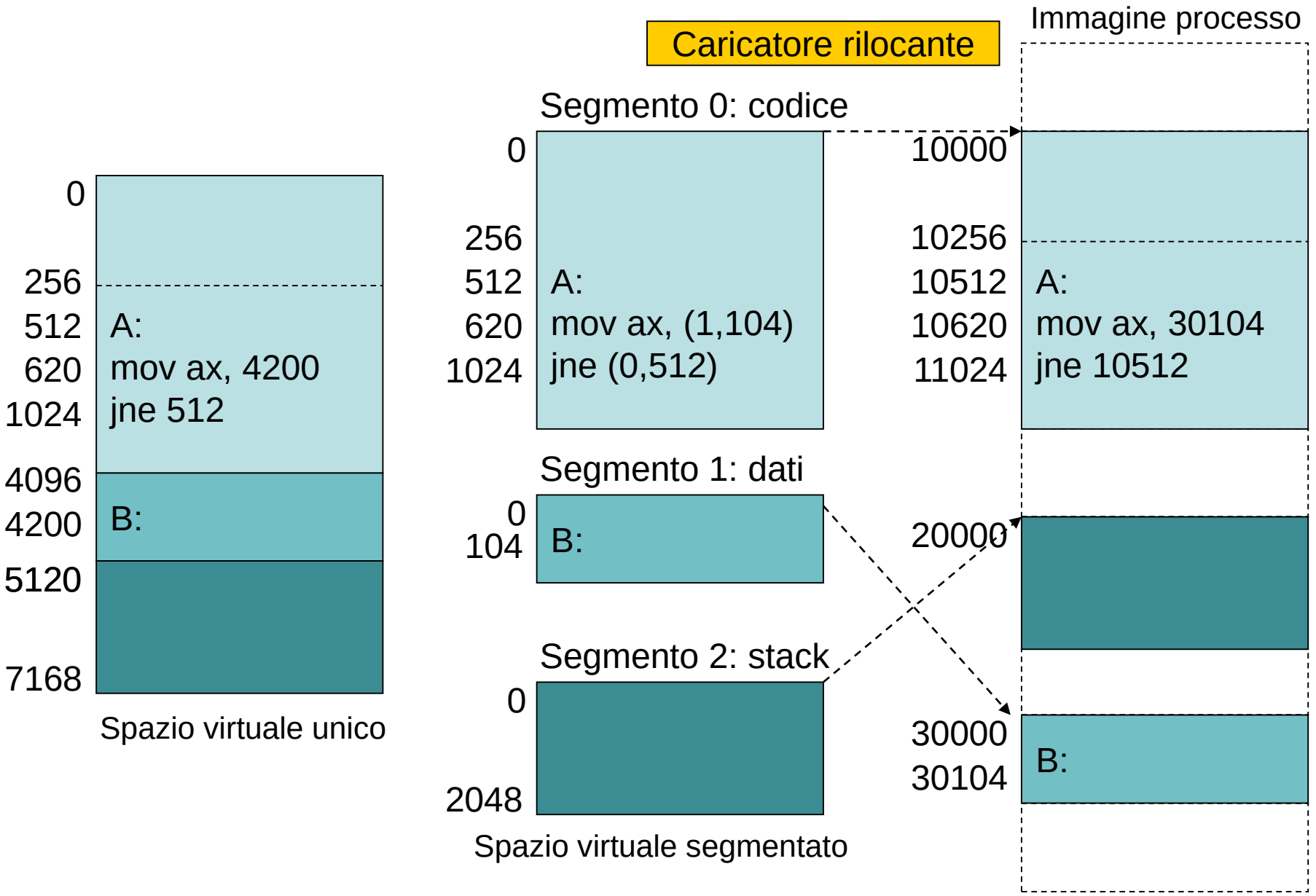

Iv
MMU
If = f(Iv)
If
MEMORIA
Indirizzo virtuale
Indirizzo fisico
MMU
Registro limite
7168
Iv = 4200
<=
eccezione
no
si
+
If = 14200
Registro base
10000
RAM
10000
10256
10512
10620
11024
14096
14200
15120
17168
A:
mov ax, 4200
jne 512
B:

- Con la rilocazione dinamica, quando ad un processo è revocata la memoria (funzione di swap_out) e successivamente viene riallocata una diversa partizione (funzione di swap_in), in quanto il processo torna nello stato di esecuzione, viene caricato nel registro base l’indirizzo iniziale della nuova area di memoria assegnata al processo.

Organizzazione dello spazio virtuale con modello di memoria segmentato

- Negli esempi precedenti abbiamo supposto che lo spazio virtuale di un processo sia costituito da **un unico spazio** di indirizzi virtuali (indirizzi compresi tra 0 e 7168) e abbiamo supposto che il linker assegnasse indirizzi contigui a tutti i moduli che compongono il programma.
- In particolare la struttura del processo descritta è composta da tre **segmenti** distinti:
  - **segmento del codice (text)**
  - **segmento dati**
  - **segmento dello stack**
- Molti processori forniscono il supporto per il **modello di memoria segmentato**.
- La memoria appare a un programma come un gruppo di blocchi di indirizzi indipendenti chiamati **segmenti**. Codice, dati e stack sono generalmente contenuti in segmenti separati.

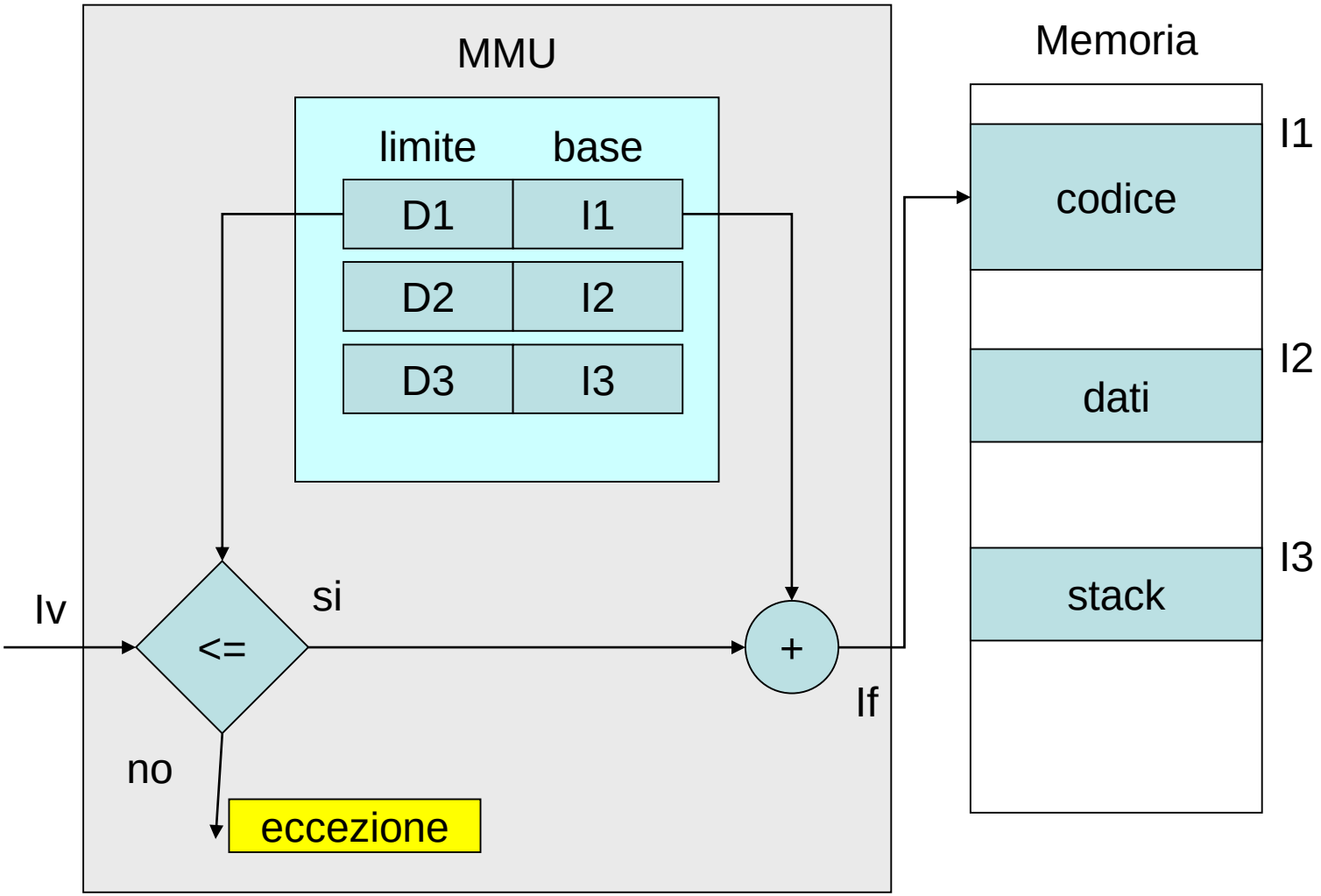
- Il linker può essere realizzato in modo che generi il modulo eseguibile, in tre segmenti separati, uno per il codice (segmento 0), uno per i dati (segmento 1) e uno per lo stack (segmento 2), ciascuno contenente indirizzi che iniziano da 0. In tal caso si parla di **spazio virtuale segmentato**.
- In tal modo, nel codice eseguibile, ogni indirizzo virtuale del processo sarà rappresentato mediante una coppia di valori interi **<segmento, indirizzo>**, dove:
  - **segmento** indica uno dei segmenti
  - **indirizzo** specifica l'indirizzo della locazione all'interno del segmento.
- Ogni segmento può essere rilocato in memoria fisica indipendentemente dagli altri.


Caricatore rilocante
Immagine processo
Segmento 0: codice
0
256
512
620
1024
A:
mov ax, (1,104)
jne (0,512)
10000
10256
10512
10620
11024
A:
mov ax, 30104
jne 10512
Segmento 1: dati
0
104
B:
20000
Segmento 2: stack
0
2048
30000
30104
B:
Spazio virtuale segmentato
0
256
512
620
1024
4096
4200
5120
7168
A:
mov ax, 4200
jne 512
B:
Spazio virtuale unico

- Nel caso di rilocazione statica, il caricamento di un processo richiederà l'allocazione di tre aree di memoria nelle quali trasferire i tre segmenti. In questo caso ci saranno tre funzioni di rilocazione con tre valori iniziali diversi, uno per ogni segmento.
- Nel caso di rilocazione dinamica, ora il supporto MMU deve avere tre coppie di registri limite e tre coppie di registri base per rilocare dinamicamente i tre segmenti.
- Per selezionare la giusta coppia di registri base-limite è necessario che la CPU generi gli indirizzi virtuali sotto forma di coppie di numeri interi

  <segmento, indirizzo>

  in modo tale che il numero di segmento sia usato per selezionare l'opportuna coppia di registri.
- L'architettura di un processore che può indirizzare più segmenti prende il nome di ***architettura con indirizzamento segmentato***.



Tecnica della segmentazione

- Il numero di segmenti potrebbe essere superiore ai tre descritti. Ad ogni modulo di programma come funzioni, struttura dati, etc. può essere associato un segmento. Questo consente di caratterizzare il segmento in modo più accurato, tramite determinate proprietà come quelle relative ad esempio alla protezione e alla condivisione.

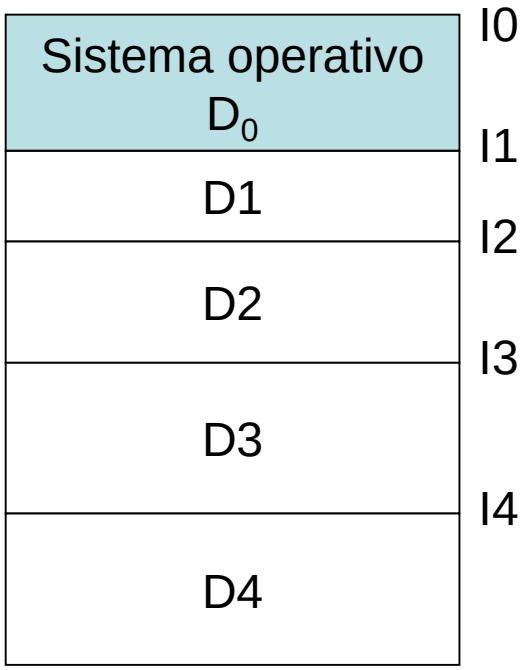
Memoria partizionata

- Le tecniche di memoria partizionata prevedono che la memoria sia suddivisa in partizioni.
- Utilizzano ila **rilocazione statica**, operano su immagini di processo **con spazio virtuale unico** il quale è caricato interamente in memoria.
- Il gestore della memoria provvede ad allocare memoria ad un processo cercando una partizione **libera** che possa contenere lo spazio virtuale da caricare.
- Il caricatore rilocherà nella partizione libera lo spazio virtuale del processo e la partizione rimarrà assegnata al processo fino alla sua terminazione. Tuttavia la partizione può essere revocata al processo per via di una operazione di **swap_out**. In tal caso sarà ricaricato nella stessa partizione con l'operazione di **swap_in**.

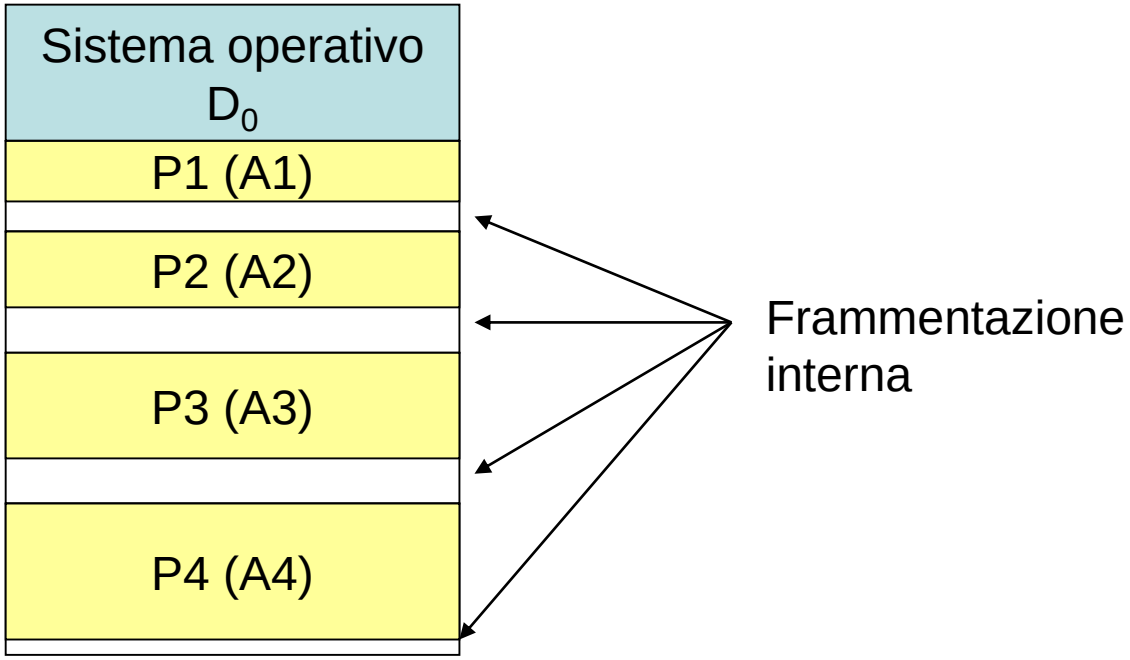
- La memoria partizionata si utilizza in base a due tecniche:
  - **partizioni fisse**
  - **partizioni variabili**

Partizioni fisse

- La memoria è divisa in **N** partizioni di dimensioni fisse specificate da un **indirizzo** e da una **dimensione** che sono definite nella fase di configurazione del sistema operativo.
- La prima partizione è riservata al SO e le restanti saranno allocate a **N-1** processi.



- Un processo è allocato in una partizione che possa contenere la sua immagine.
- Questa tecnica produce il fenomeno della **frammentazione interna** in quanto è poco probabile che le immagini dei processi abbiano dimensioni coincidenti a quelle delle partizioni.



- La somma delle dimensioni dei frammenti

$$\sum (D_i - A_i) \qquad (i=1,..N-1)$$

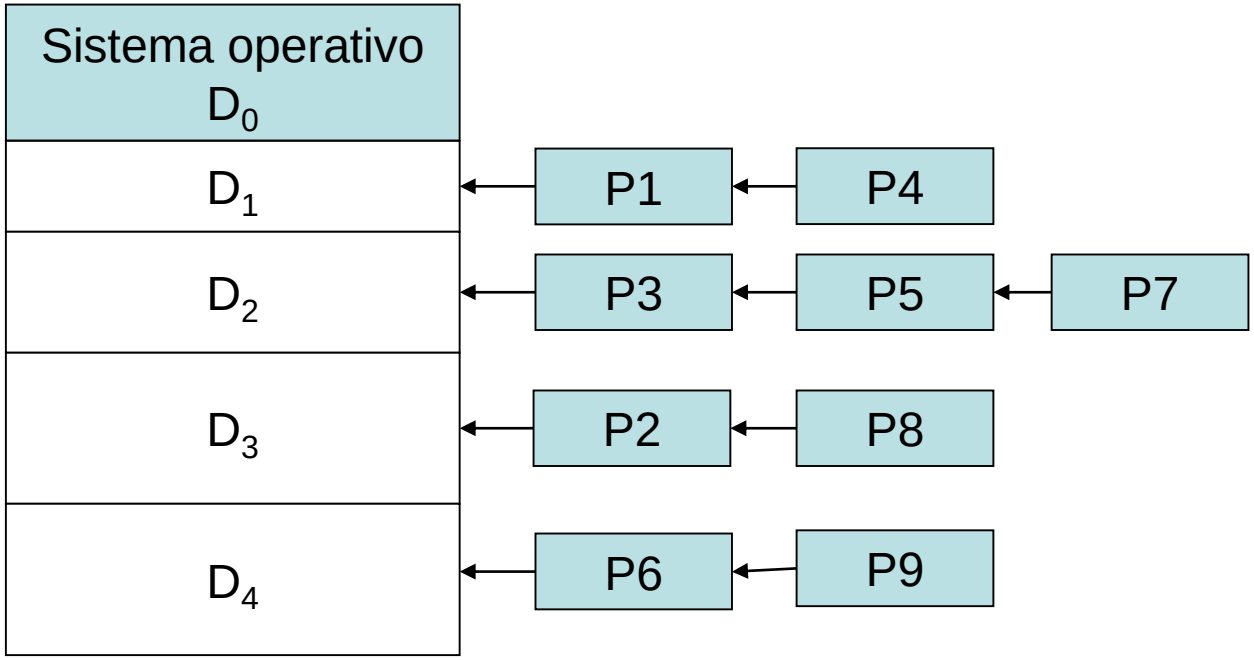
Di: dimensione della partizione i

Ai: dimensione dell'immagine del processo Pi

inutilizzati potrebbe essere considerevole.

- La tecnica delle partizioni, anche se utilizza la rilocazione statica degli indirizzi, consente l'allocazione dinamica della memoria ai processi.
- Per essere eseguito, un processo deve trovarsi in memoria principale, ma si può trasferire temporaneamente in memoria secondaria da cui si riporta in memoria principale nel momento di riprendere l'esecuzione. Nel frattempo, la memoria liberata può essere assegnata ad un altro processo. Questo procedimento si chiama **swapping** (avvicendamento o scambio). Quindi , con le funzioni di **swap_out** e successivamente di **swap_in** è possibile riallocare il processo nella stessa partizione.

- Il kernel gestisce, per ogni partizione, una coda di descrittori di processi (PCB) che saranno assegnati a quella partizione.
- Generalmente, la coda è gestita con politica round robin. Pertanto, la partizione è revocata al processo cui è allocata e viene assegnata al primo della coda. Il processo a cui è stata revocata la memoria è inserito in fondo alla coda.



Code di processi per le singole partizioni

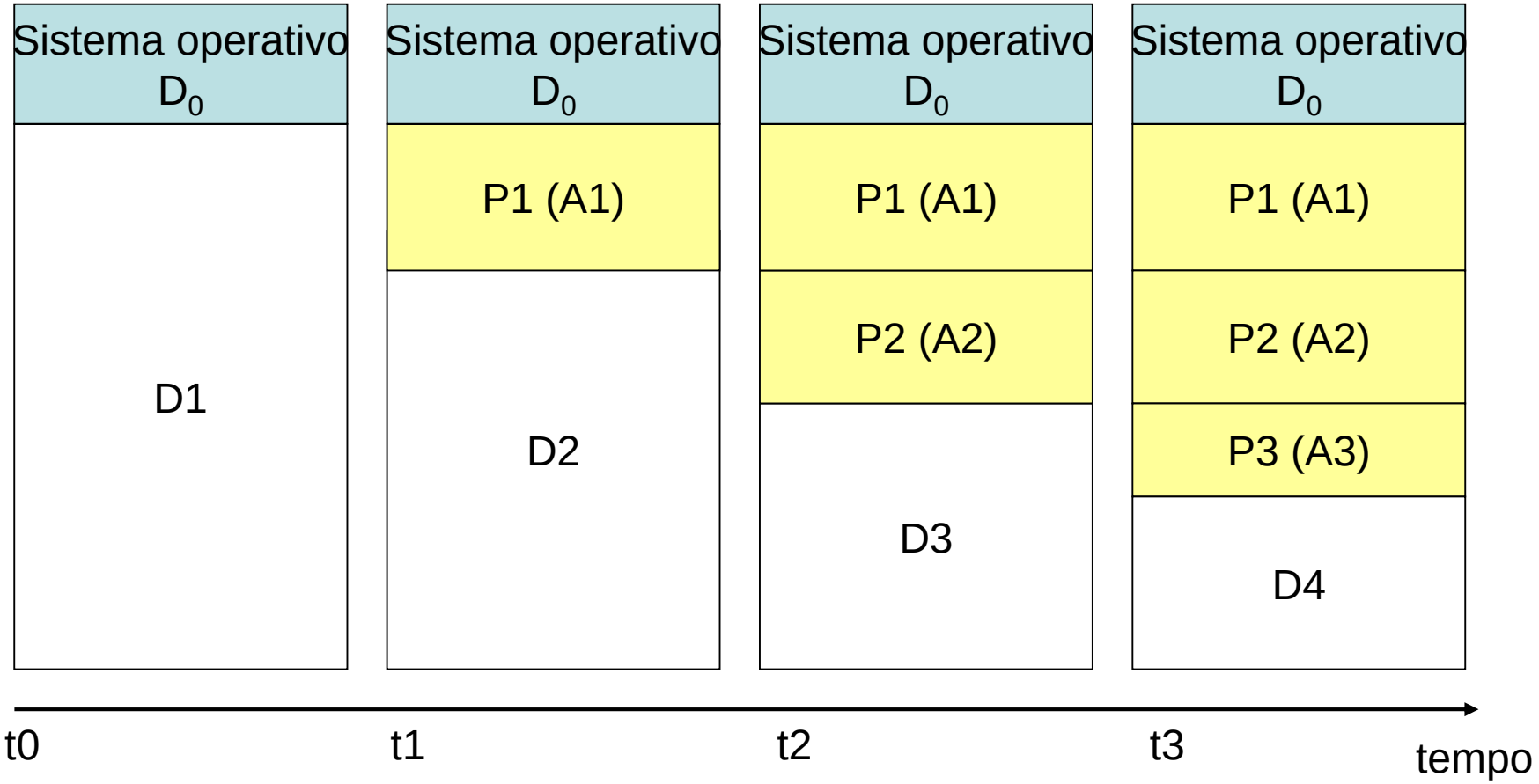
- L’avvicendamento dei processi può eseguirsi anche in base ad algoritmi di scheduling basati sulle priorità.
- Nei sistemi che utilizzano la rilocazione statica, un processo che è stato scaricato dalla memoria deve essere ricaricato nello spazio di memoria occupato in precedenza.
- Se la rilocazione degli indirizzi logici agli indirizzi fisici della memoria avviene durante la fase di esecuzione, un processo può essere ricaricato in uno spazio di memoria diverso, poiché gli indirizzi fisici si calcolano nella fase d'esecuzione.
- In un sistema d'avvicendamento così descritto, il tempo di cambio di contesto è abbastanza elevato. Per avere un’idea delle sue dimensioni si pensi a un processo applicativo con immagine di 100 MB e a una memoria secondaria costituita da un disco rigido con velocità di trasferimento di 50 MB al secondo.
- Il trasferimento effettivo del processo di 100 MB da e in memoria richiede:

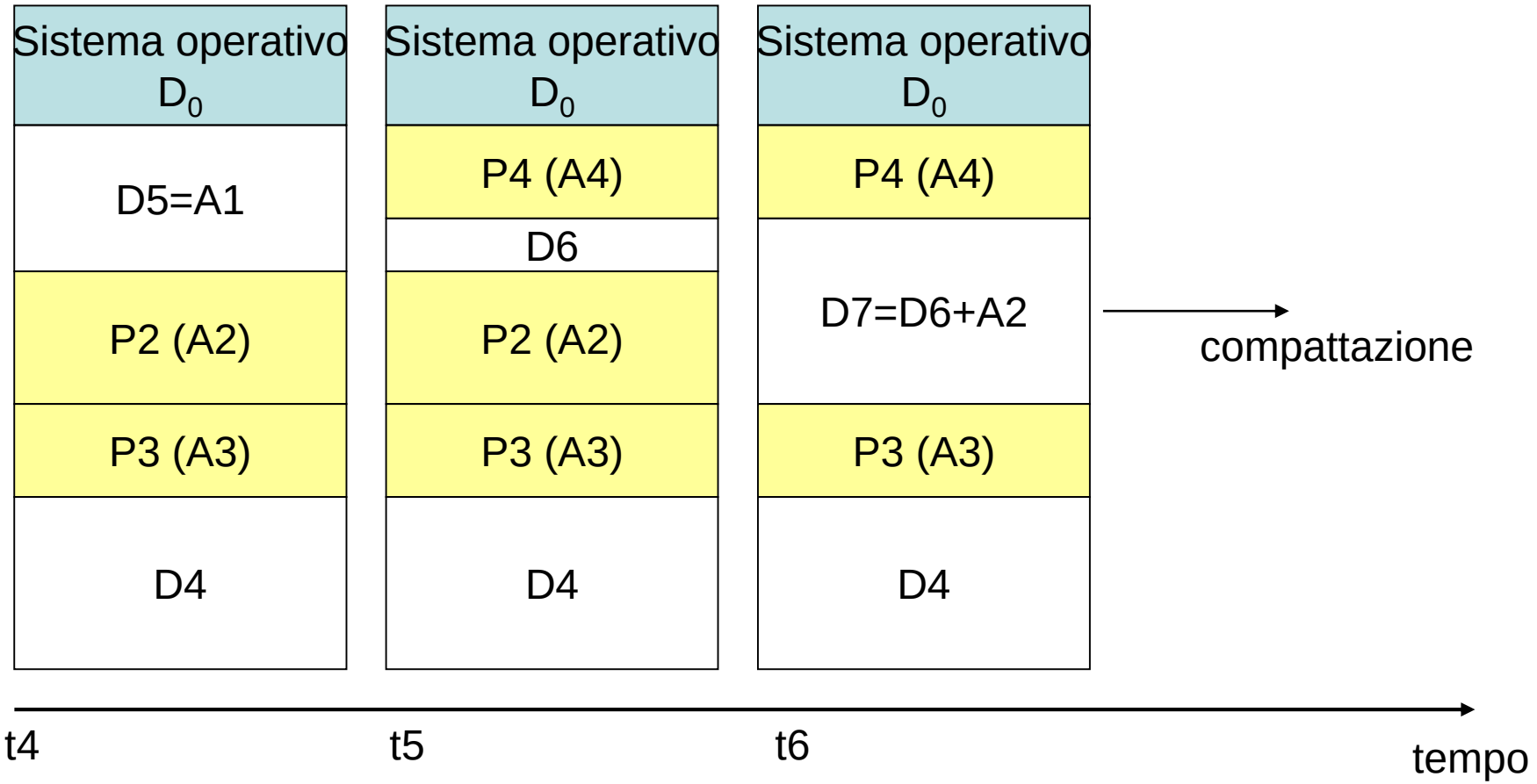
  100 MB /50 MB al secondo = 2 secondi.

- Supponendo che il ritardo medio generato dal dispatcher nell’eseguire il cambio di contesto sia di 8 millisecondi, l'operazione richiederebbe 2008 millisecondi. Poiché l’avvicendamento richiede lo scaricamento e il caricamento di un processo, il tempo totale è di circa 4016 millisecondi.
- Occorre notare che la maggior parte del tempo d'avvicendamento è data dal tempo di trasferimento che è direttamente proporzionale alla dimensione dell’immagine del processo.
- La tecnica di allocazione a partizioni fisse è semplice, richiede un basso overhead di sistema ma risulta **inefficiente** per l’uso della memoria a causa della frammentazione interna.

Partizioni variabili

- Un miglioramento dell'uso della memoria si ha con la tecnica a partizioni variabili.
- Inizialmente la memoria è divisa in due partizioni, D0 riservata al SO e D1 per allocare le immagini dei processi.
- Al momento della creazione del primo processo P1 di dimensione A1, viene creata in D1 una partizione di dimensione A1 in cui viene allocato P1.
- Dopo l'allocazione di P1 resta libera una partizione di dimensione D2=D1-A1, che può essere utilizzata per allocare nuovi processi.
- La figura seguente mostra lo stato di allocazione della memoria dopo assegnazione di partizioni a 3 processi P1, P2 e P3 rispettivamente di dimensioni A1, A2 e A3.


Sistema operativo
D_0
D1
Sistema operativo
D_0
P1 (A1)
D2
Sistema operativo
D_0
P1 (A1)
P2 (A2)
D3
Sistema operativo
D_0
P1 (A1)
P2 (A2)
P3 (A3)
D4
t0
t1
t2
t3
tempo



- Poiché i processi sono creati e terminano continuamente, la memoria è dinamicamente suddivisa in partizioni libere separate da partizioni occupate dai processi.

- Se si liberano due partizioni adiacenti, queste sono compattate.
- La tecnica a partizioni variabili elimina il problema della frammentazione interna, generato dallo schema a partizioni fisse, ma produce **frammentazione esterna**, che si ha quando la somma delle dimensioni delle partizioni libere è superiore alla dimensione dell'immagine di un nuovo processo, ma non esiste una singola partizione di dimensione sufficiente per poter essere allocata al nuovo processo. Infatti la **rilocazione statica** non consente di compattare le partizioni libere non adiacenti.
- Per memorizzare lo stato di allocazione della memoria, il gestore della memoria mantiene una struttura dati nella quale memorizza il numero di partizioni libere e le loro definizioni (ad esempio indirizzo iniziale e dimensione).

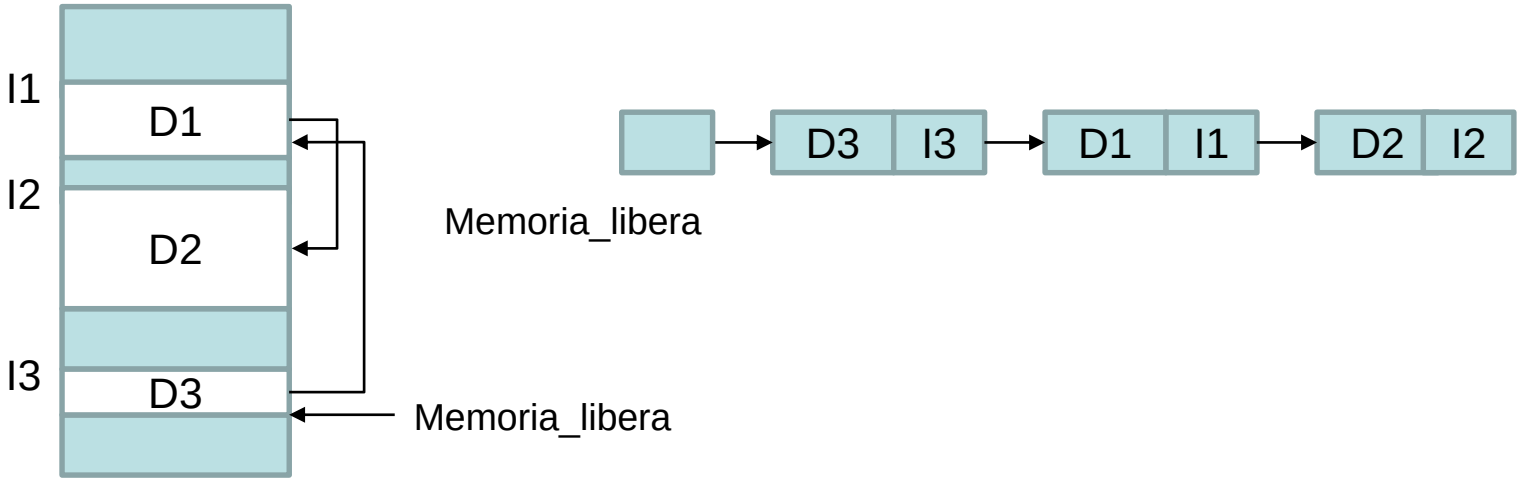
- In particolare, gestisce una lista concatenata, la **lista delle partizioni libere (free list)**, in cui ciascun elemento contiene la dimensione della partizione e l'indirizzo della successiva. Per velocizzare le operazioni di inserimento e cancellazione la lista può essere realizzata con doppio puntatore
- L'indirizzo della prima partizione libera è contenuto in una variabile di sistema (memoria_libera).

Tecniche di allocazione mediante free list

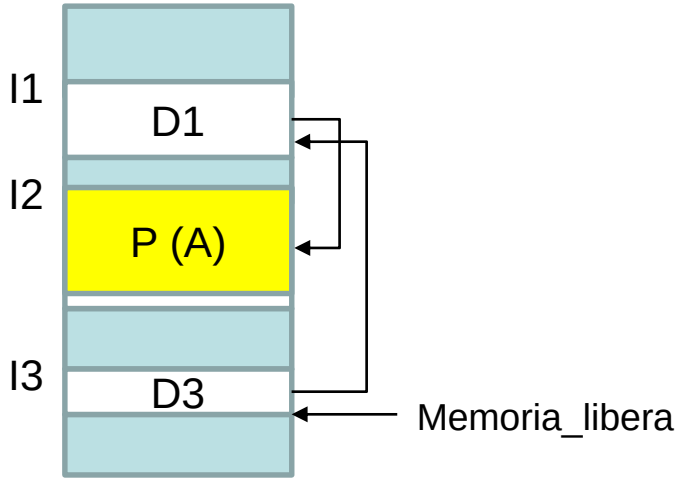
- Generalmente, nei sistemi che utilizzano la tecnica di gestione della memoria a partizione fisse, per allocare un processo in memoria, si utilizza un algoritmo, detto **best-fit**, che seleziona la partizione che ha una dimensione più vicina, in eccesso, alla dimensione dell'immagine del processo.
- Nei sistemi che utilizzano la tecnica delle partizioni variabili per allocare un processo in memoria possono essere utilizzate varie tecniche free-list, tra le le quali:
  - Best-fit
  - First-fit
  - Worst-fit

Best-fit

- L’algoritmo best-fit gestisce la lista delle partizioni libere ordinandola per **valori crescenti delle dimensioni delle partizioni**. Quando un processo, la cui immagine ha dimensione **A**, richiede una partizione, la lista viene scandita e la prima partizione di dimensione maggiore di A è sicuramente quella più vicina alla dimensione dell’immagine del processo.

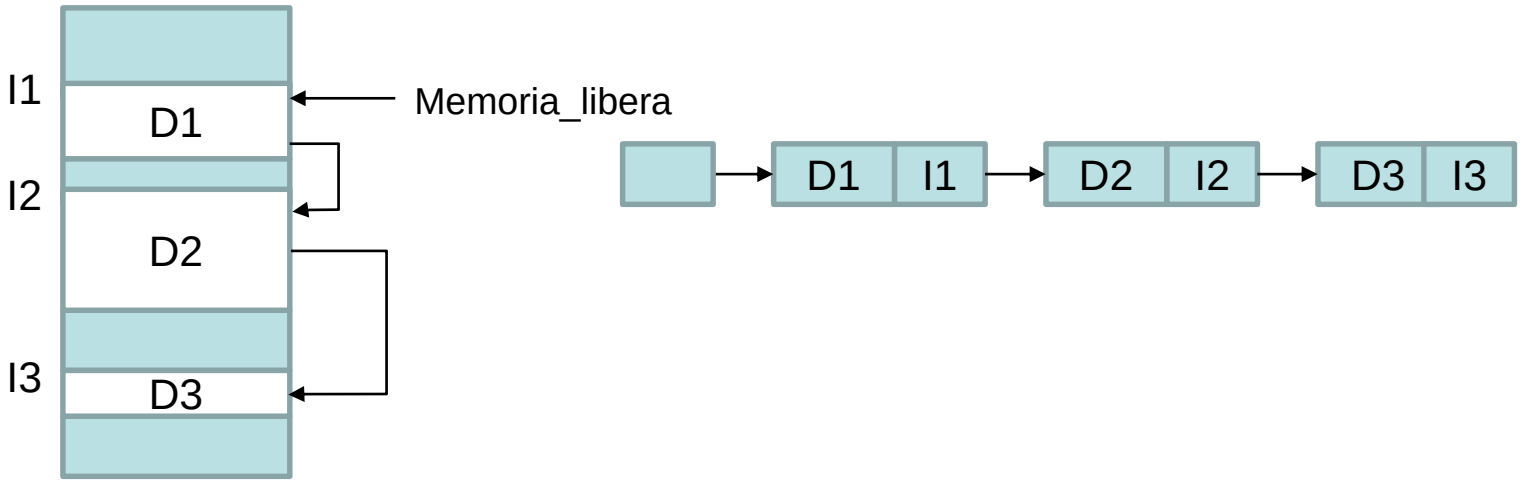


- Lo schema best-fit ha due inconvenienti:
  - Una volta effettuata l'allocazione, la parte della partizione non utilizzata è sicuramente quella di dimensioni più piccole e questo porta a far crescere la frammentazione della memoria.
  - In fase di rilascio è necessario verificare se la partizione liberata è adiacente ad una o a due partizioni libere per compattarle insieme. Questa verifica implica una scansione dell'intera lista per controllare l'esistenza di eventuali adiacenze.



First-fit

- L’algoritmo first-fit, più spesso usato, (con prestazioni migliori del best-fit) consiste nel mantenere la **lista ordinata per indirizzi crescenti** delle partizioni. Quando un processo richiede una partizione di dimensione A, la lista viene scandita ed è assegnata al processo la prima partizione di dimensione maggiore di A. Questo schema risulta particolarmente efficiente durante la fase di rilascio. Infatti per la compattazione basta soltanto verificare se le partizioni adiacenti sono libere.



Worst-fit

- La lista delle partizioni libere viene **ordinata per dimensioni decrescenti delle dimensioni delle partizioni**. Quando un processo di dimensione A richiede una partizione, la lista viene scandita e la prima partizione trovata di dimensione maggiore di A è sicuramente quella più lontana alla dimensione A. Questo criterio è più adatto per lo schema a partizioni variabili, in quanto è più probabile che lo spazio rimasto consenta l'allocazione di altri processi.