

Università di Roma Tor Vergata  
Corso di Laurea triennale in Informatica  
**Sistemi operativi e reti**  
A.A. 2016-17

Pietro Frasca

# Lezione 11

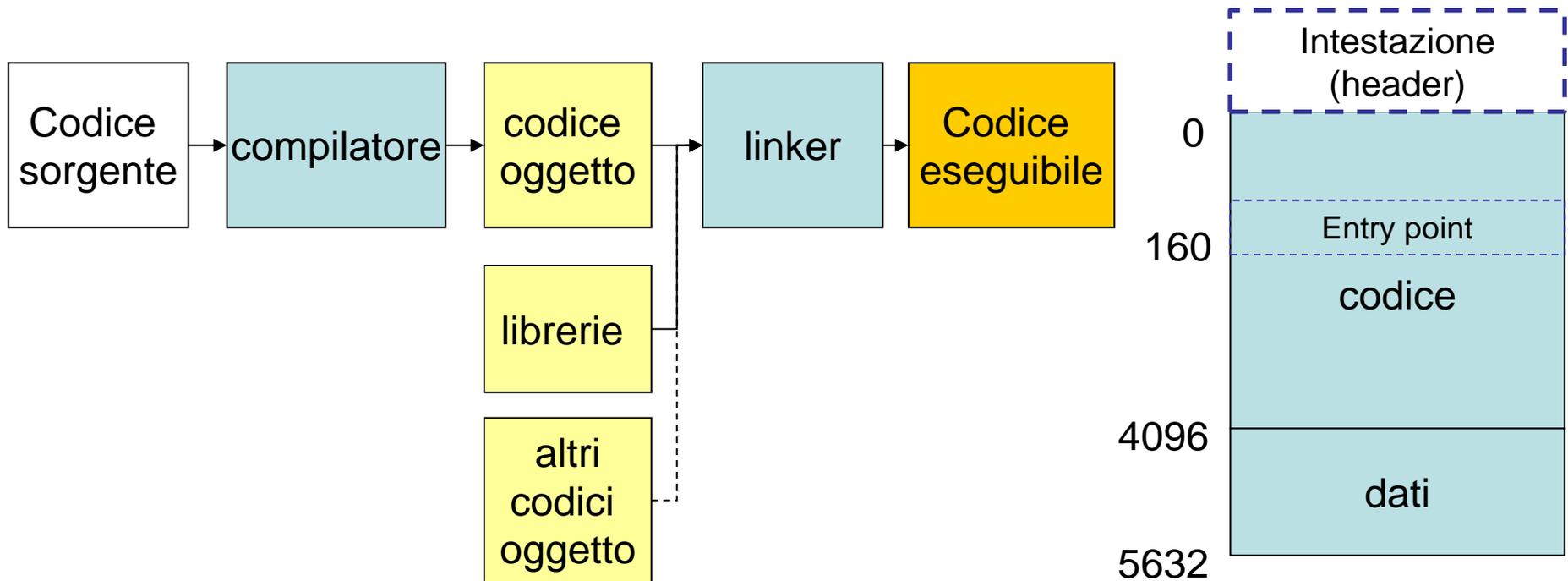
Giovedì 17-11-2016

# 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 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**.



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

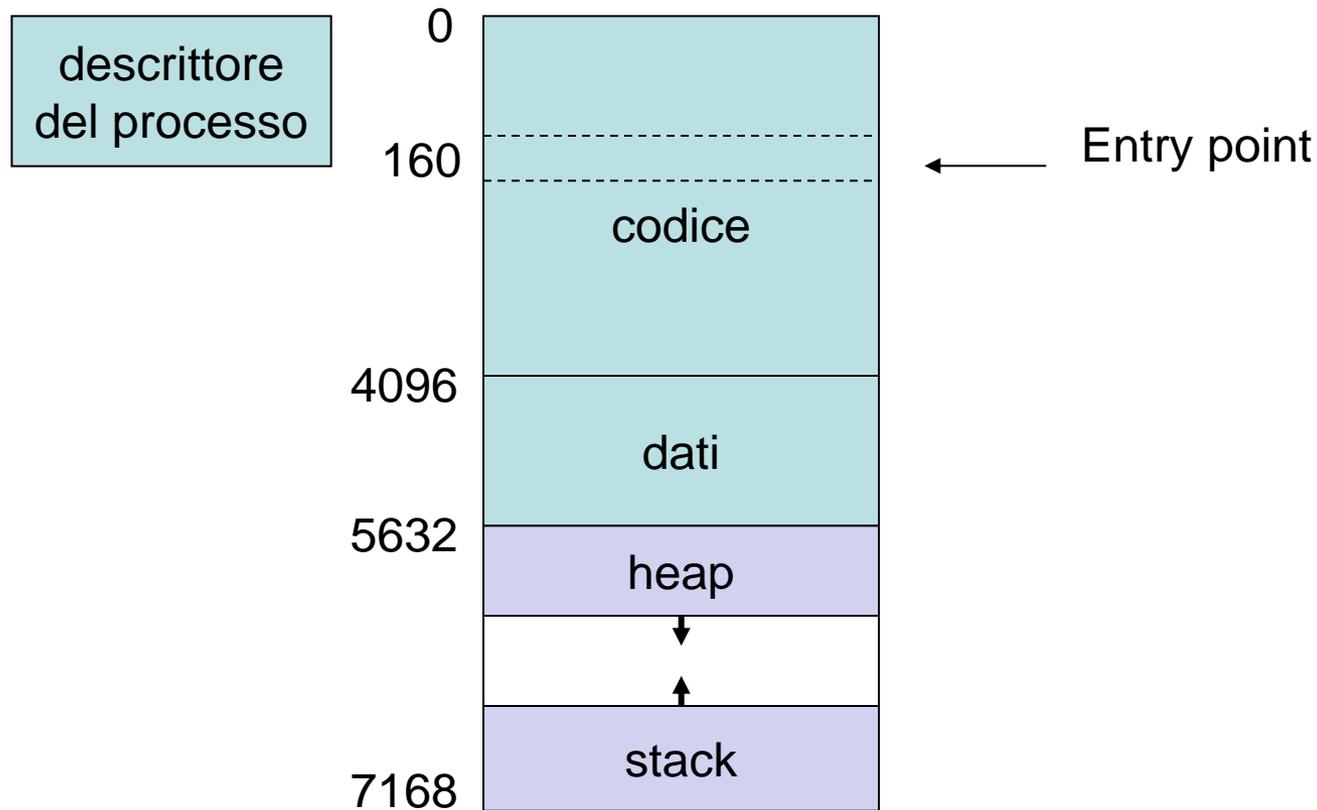


Immagine di un processo

# Organizzazione della memoria virtuale

- **Spazio virtuale unico**

Inizialmente, consideriamo il caso in cui la memoria virtuale del processo sia costituita 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.

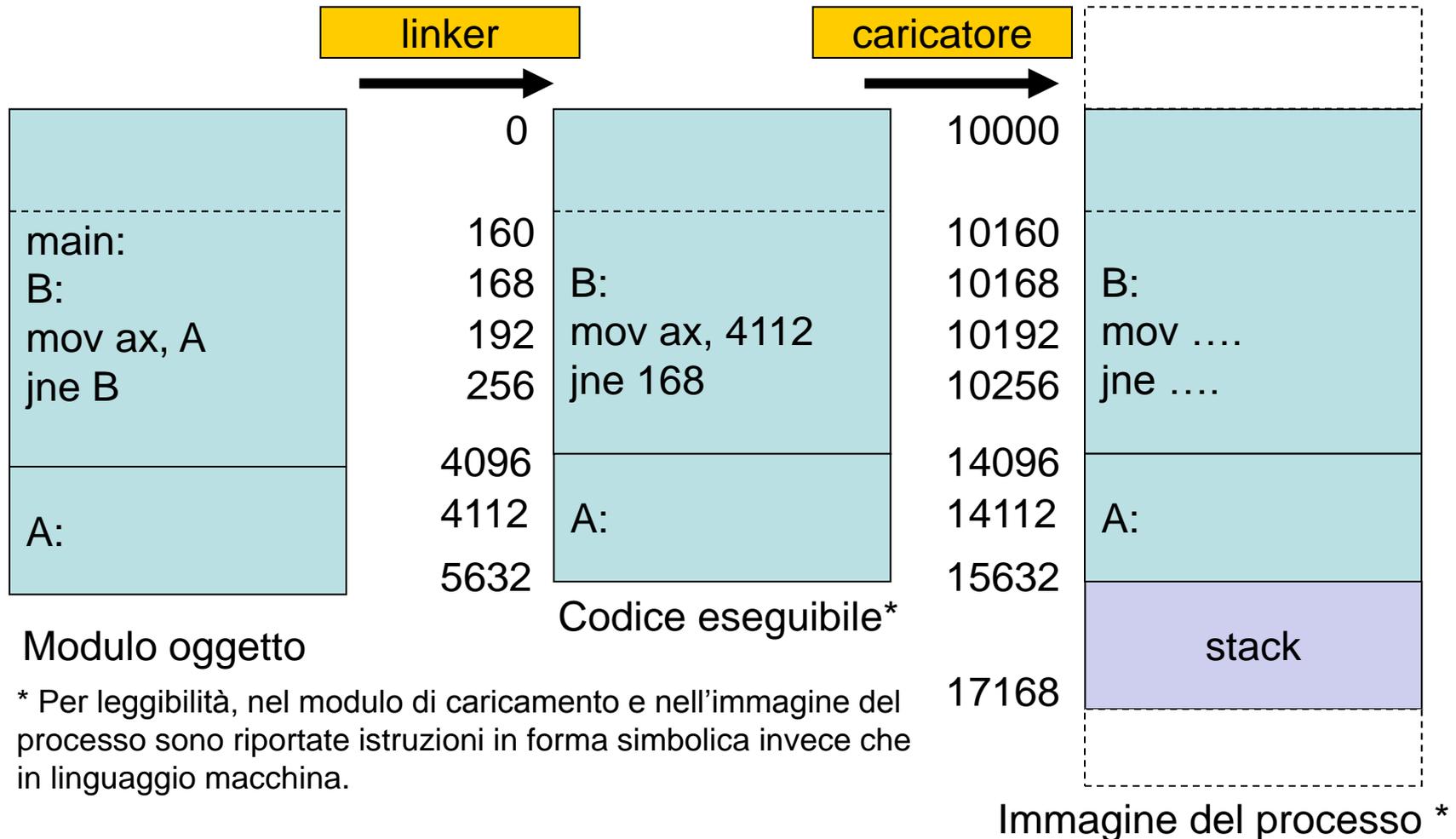
In particolare la struttura del processo visto è composta da tre parti, dette **segmenti**, distinte:

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

## Rilocazione statica e dinamica

- Generalmente, la memoria virtuale di un processo sarà caricata in memoria fisica a partire da qualsiasi indirizzo **I** 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



$$I_f = f(I_v)$$

- La funzione di rilocazione fa corrispondere ad ogni indirizzo virtuale **I<sub>v</sub>** un indirizzo fisico **I<sub>f</sub>**.
- Un esempio di funzione di rilocazione è data da:

$$I_f = I_v + I_0$$

dove **I<sub>0</sub>** è una costante che indica l'indirizzo di base.

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

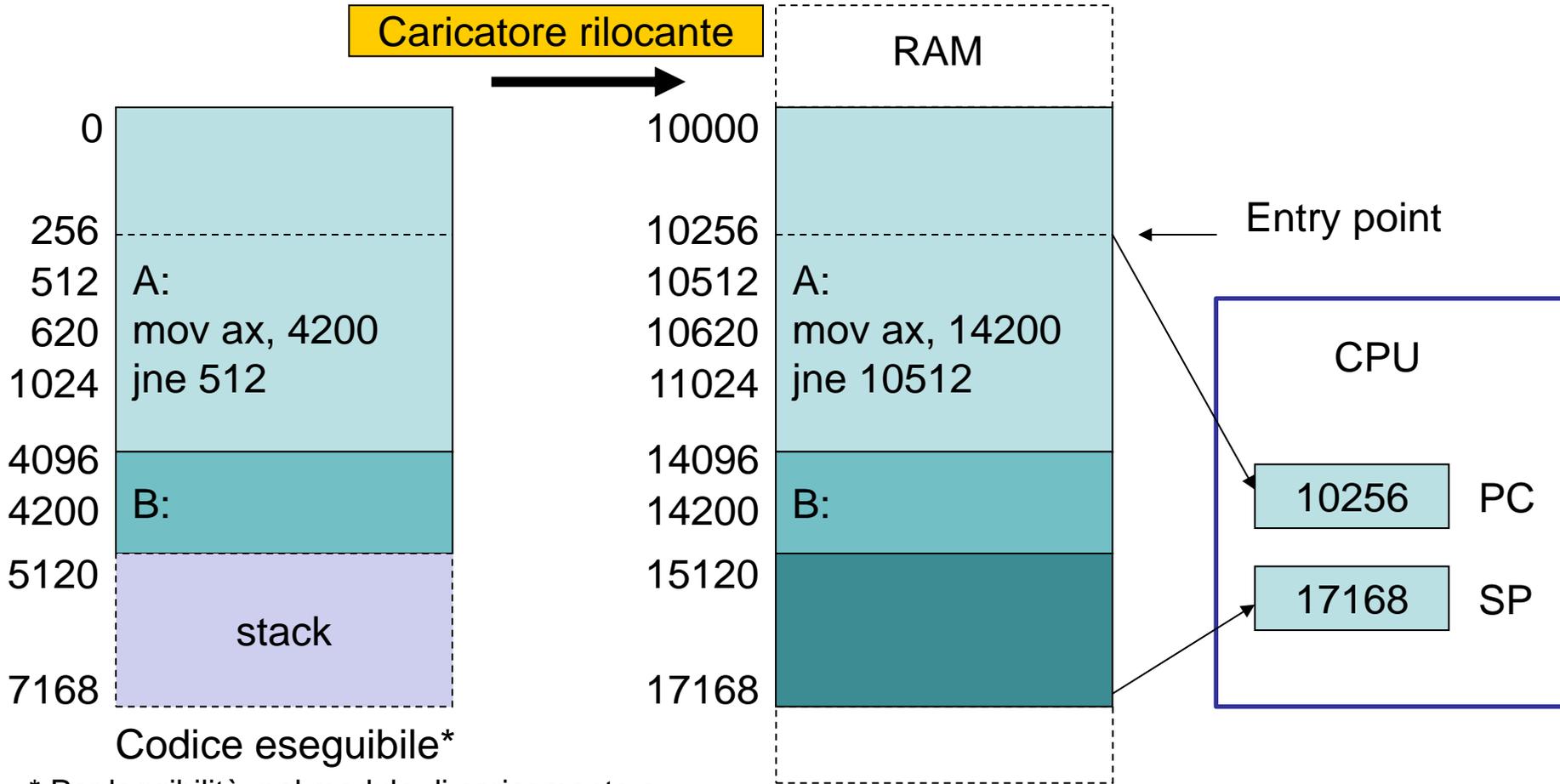
$$I_f = I_v + 10000$$

- 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

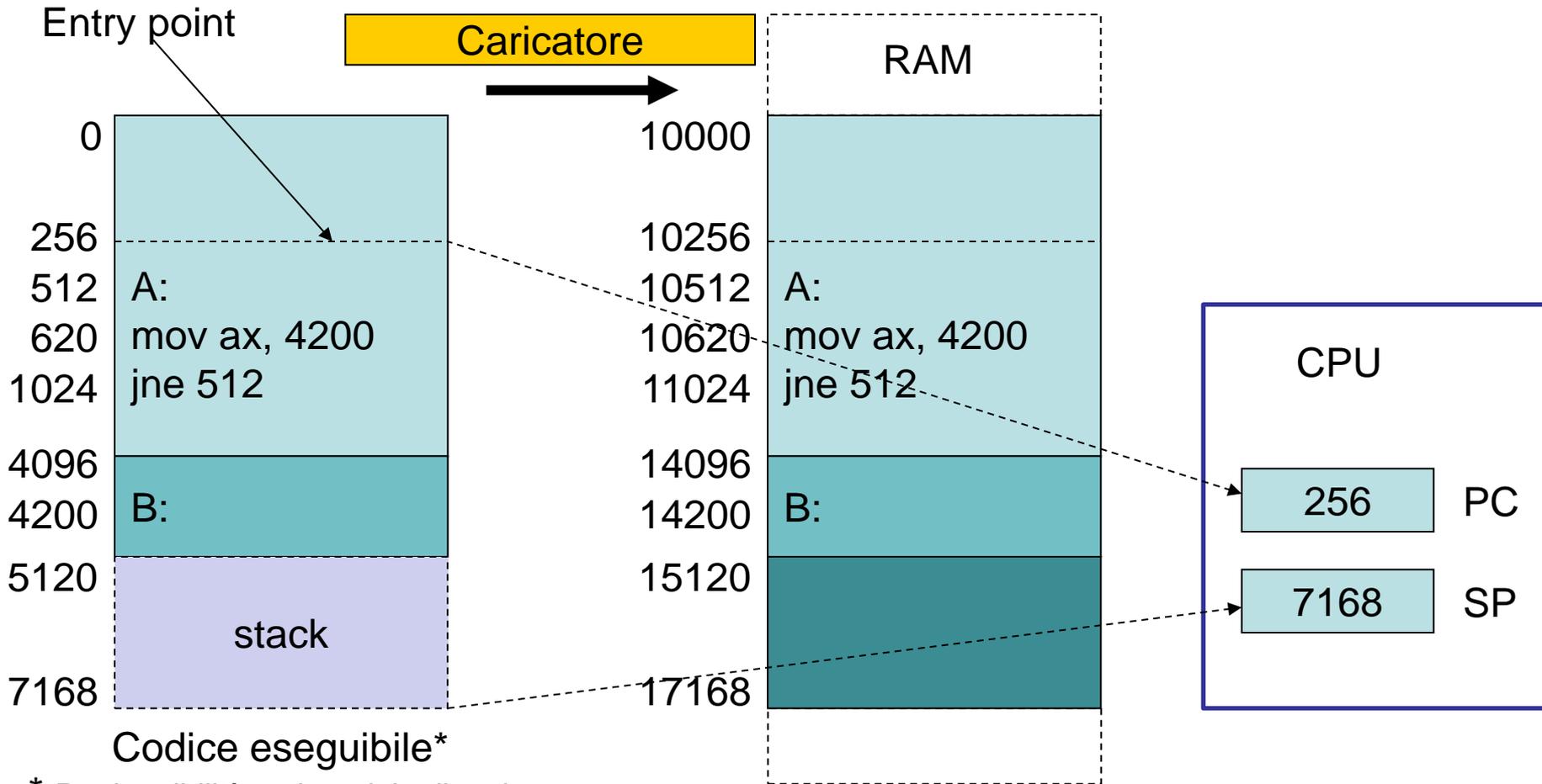
- Con la rilocazione dinamica, la traduzione degli indirizzi da virtuali a fisici avviene durante l'esecuzione del processo.
- Il caricatore trasferisce in memoria direttamente il contenuto del codice eseguibile contenente gli indirizzi virtuali, senza modificarli.
- Nei registri **PC** (program counter)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

$$If = f(Iv)$$

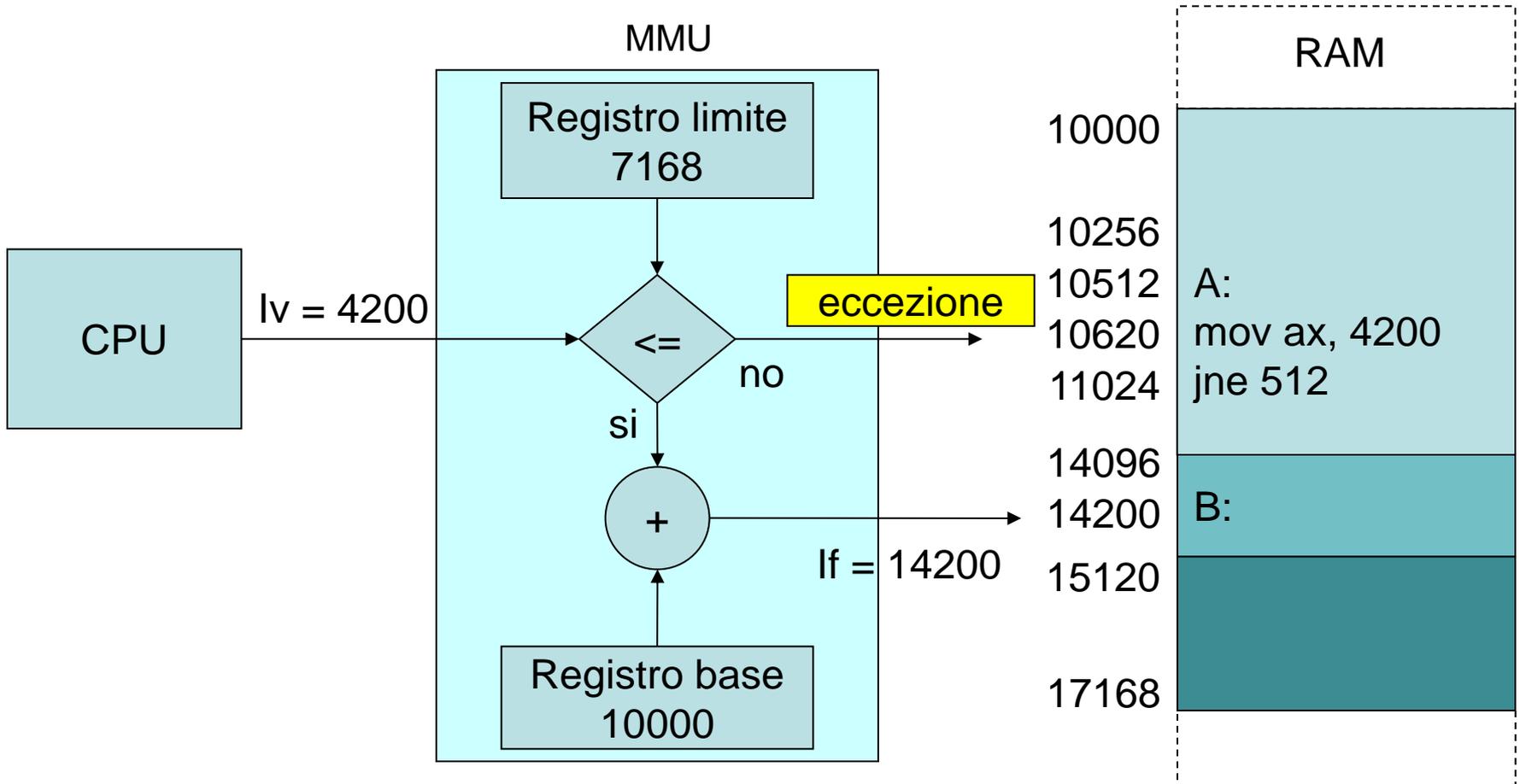
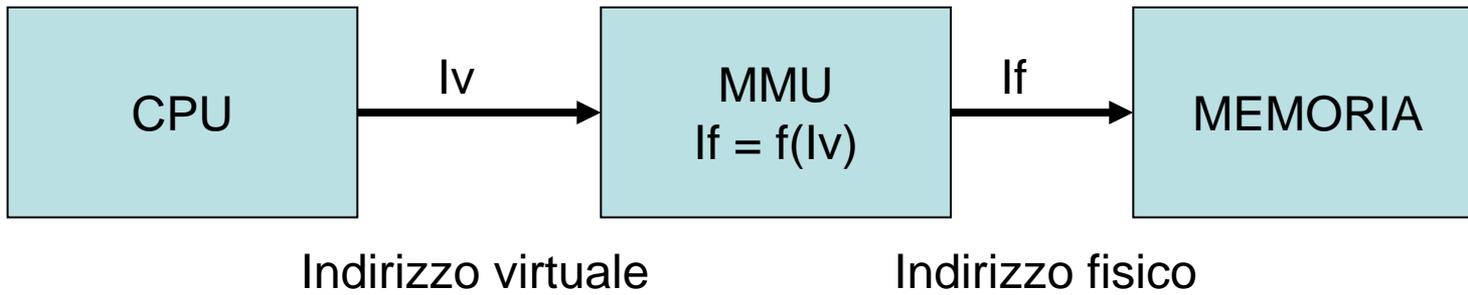
traducendo dinamicamente ogni indirizzo virtuale nel corrispondente indirizzo fisico.

- L'MMU può essere o meno incluso nel processore.

# Rilocazione dinamica



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



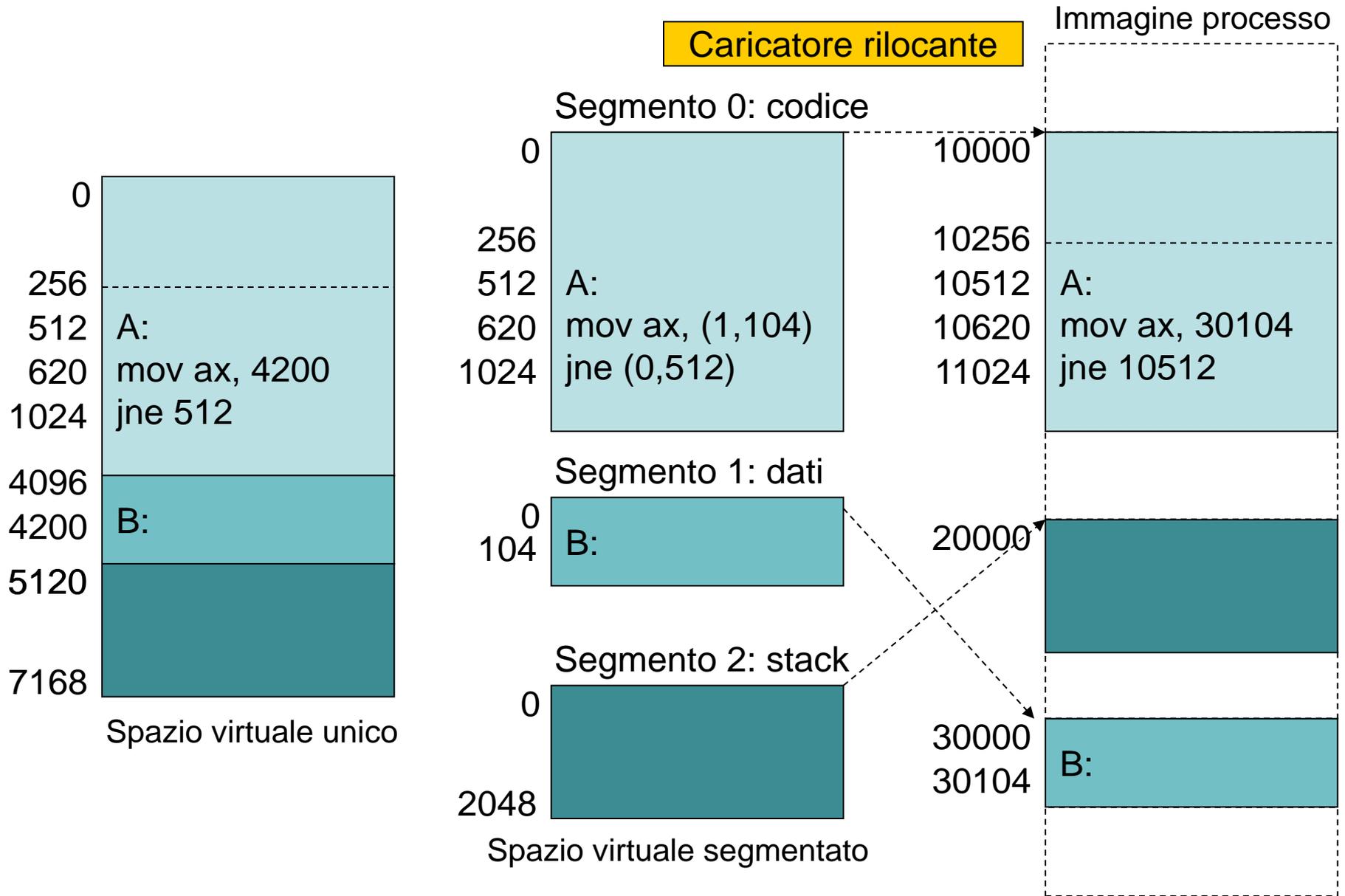
- Con la rilocazione dinamica, quando ad un processo viene revocata la memoria (funzione di `swap_out`) e successivamente viene riallocata una diversa partizione (funzione di `swap_in`), in quanto il processo torna nello stato di esecuzione, viene caricato nel registro base l'indirizzo iniziale della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

# Organizzazione della memoria virtuale con spazio virtuale segmentato

- Negli esempi precedenti abbiamo supposto che la memoria virtuale del processo sia costituita 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 visto è composto 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.



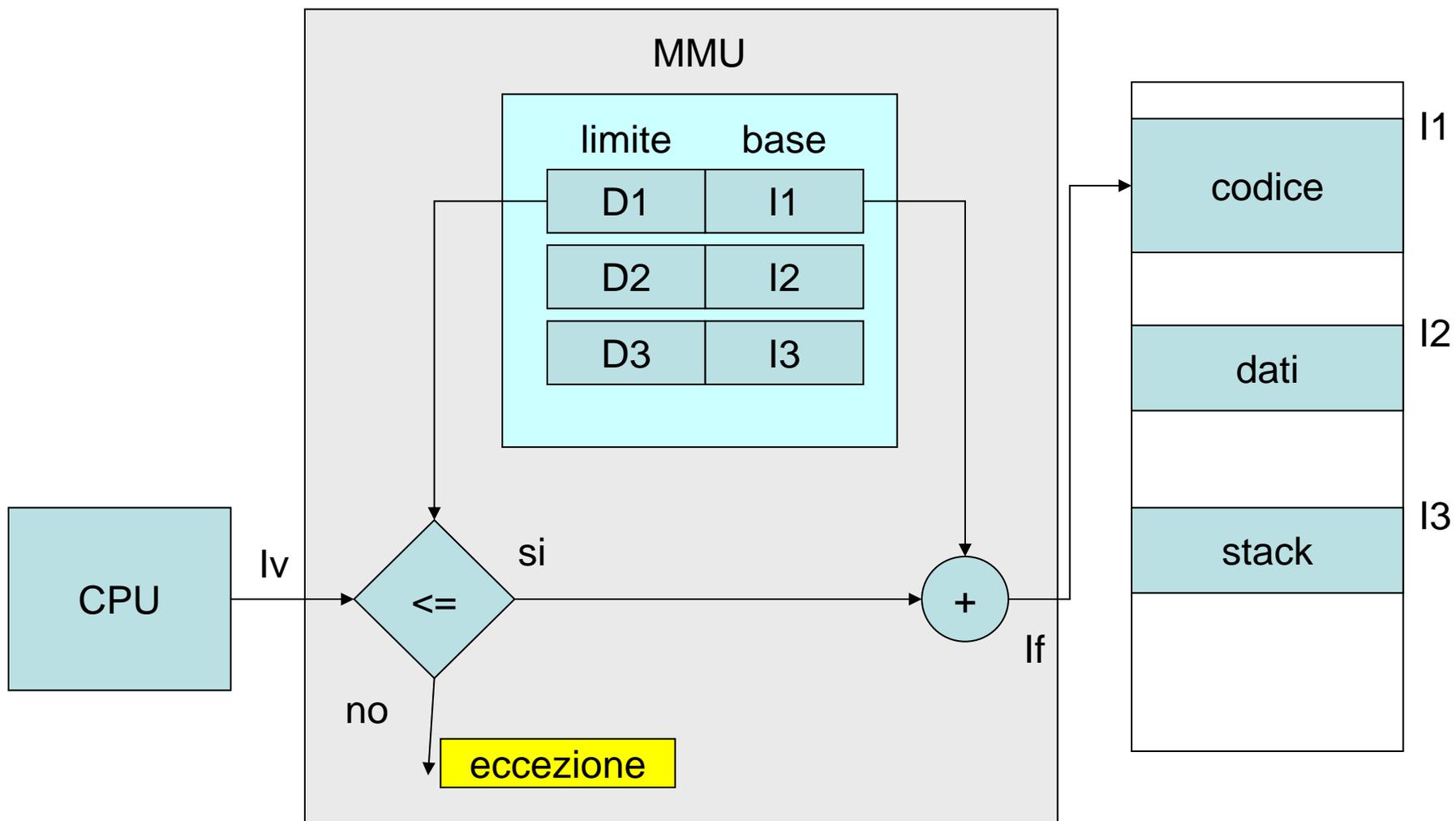
- Nel caso di rilocalizzazione statica, il caricamento di un processo comporterà l'allocazione di tre aree di memoria nelle quali trasferire i tre segmenti. In questo caso ci saranno tre funzioni di rilocalizzazione con tre valori iniziali diversi, uno per ogni segmento.
- Nel caso di rilocalizzazione dinamica, ora la 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**.

- 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.



## Tecnica della segmentazione

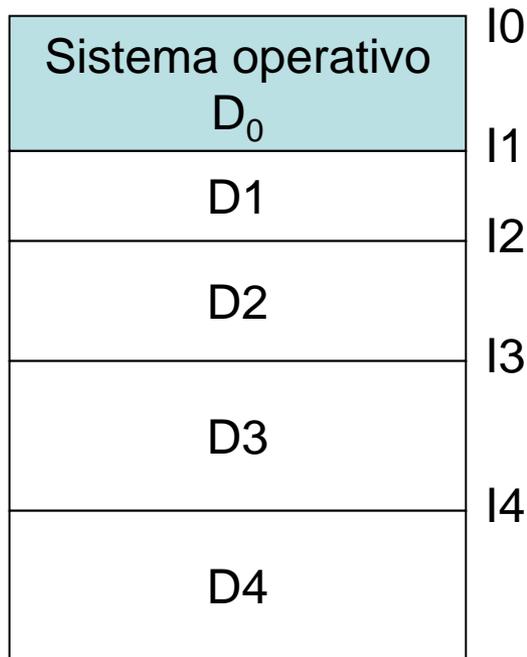
# Memoria partizionata

- Le tecniche di memoria partizionata prevedono che la memoria sia suddivisa in partizioni. Utilizzano il **rilocamento statico**, 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**.

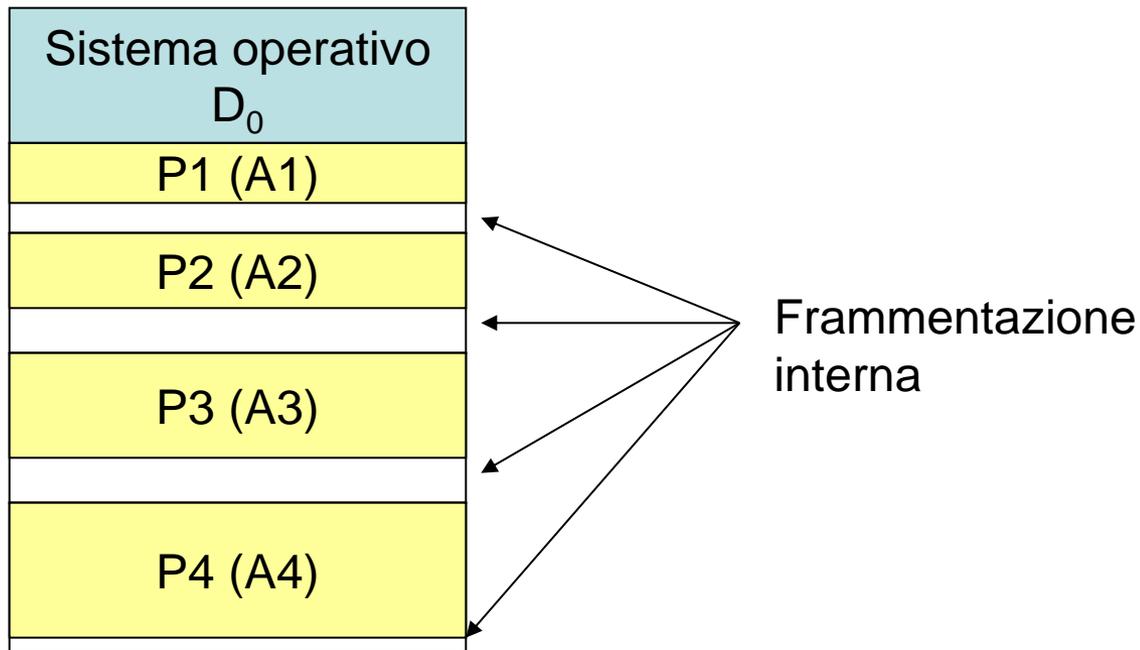
- La memoria partizionata si utilizza in base a due tecniche:
  - **partizioni fisse**
  - **partizioni variabili**

# 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

$$\sum (D_i - A_i) \quad (i=1, \dots, N-1)$$

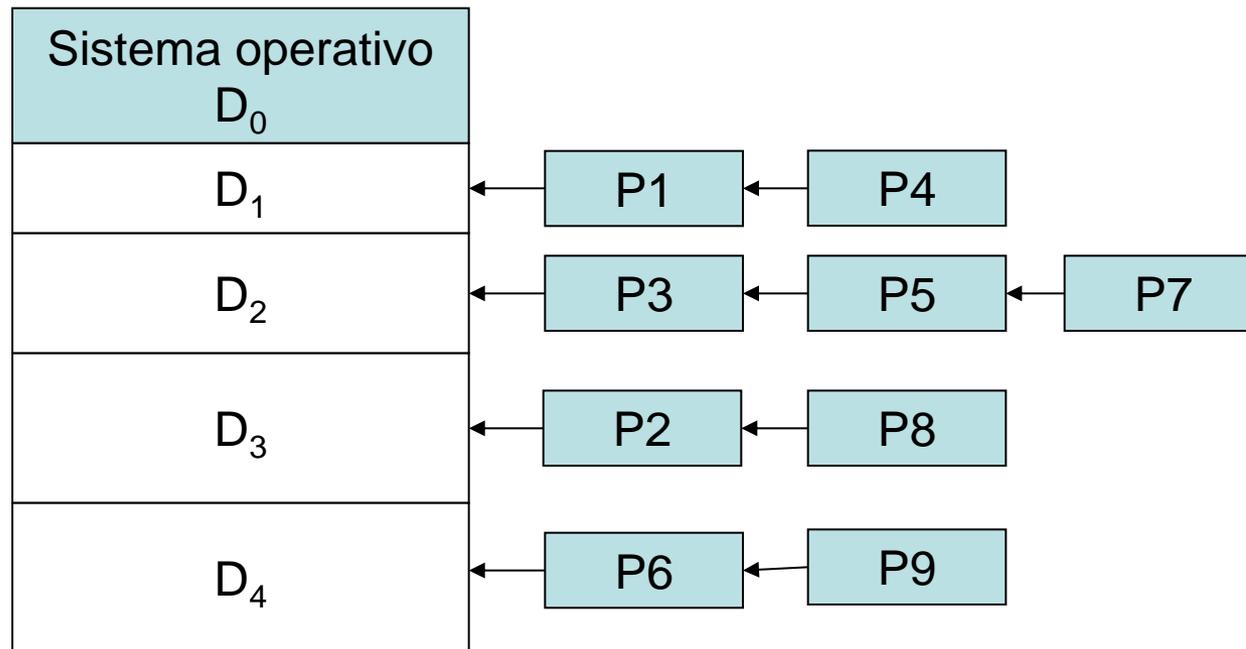
$D_i$ : dimensione della partizione  $i$

$A_i$ : dimensione dell'immagine del processo  $P_i$

inutilizzati potrebbe essere considerevole.

- La tecnica delle partizioni, anche se utilizza la rilocalizzazione statica degli indirizzi, consente l'allocazione dinamica della memoria ai processi.
- Per essere eseguito, un processo deve trovarsi in memoria centrale, ma si può trasferire temporaneamente in memoria secondaria da cui si riporta in memoria centrale 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 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 rilocalizzazione statica, un processo che è stato scaricato dalla memoria deve essere ricaricato nello spazio di memoria occupato in precedenza.
- Se la rilocalizzazione 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:  
$$100 \text{ MB} / 50 \text{ MB al secondo} = 2 \text{ secondi.}$$

- 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.