Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2016-17

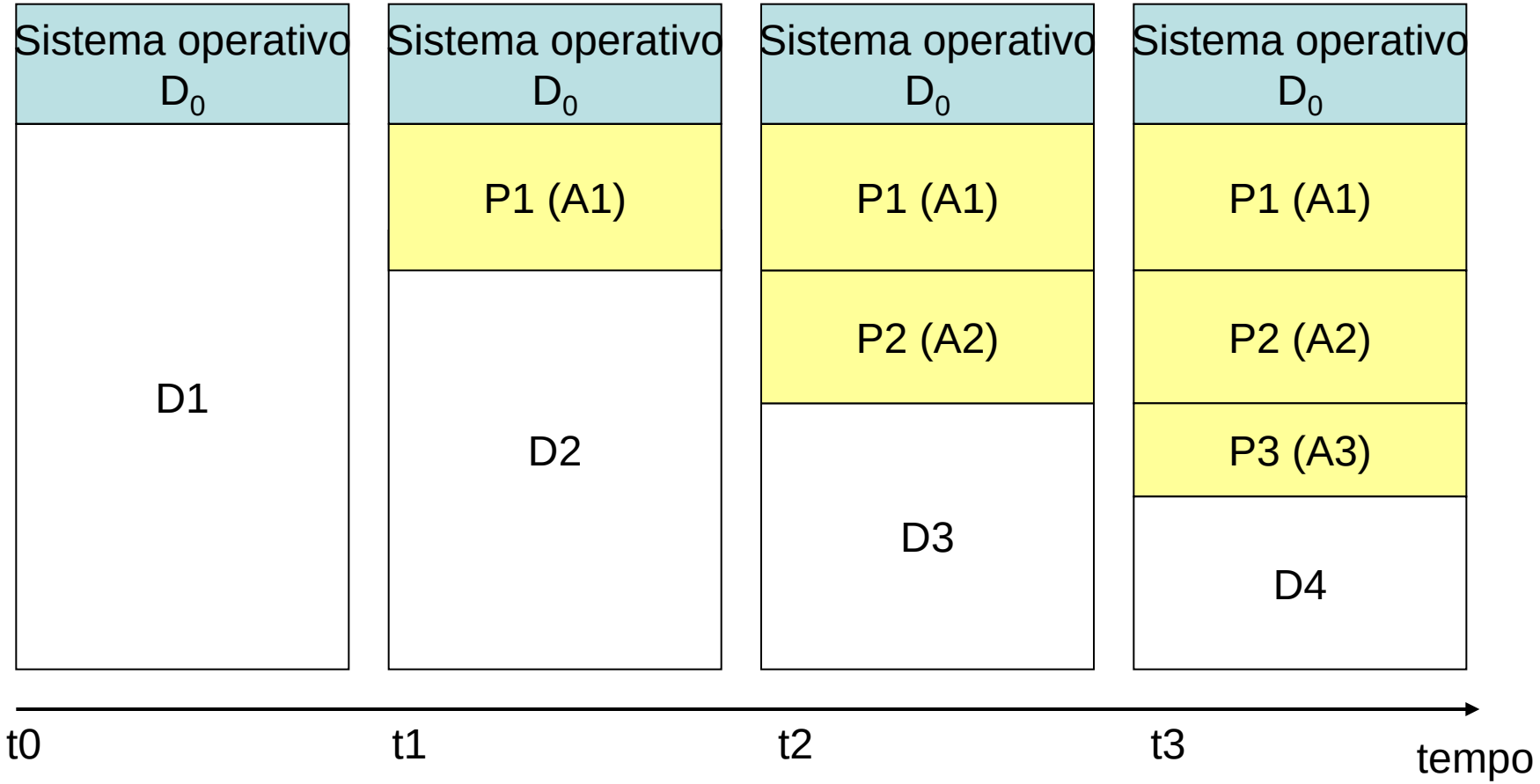
Pietro Frasca

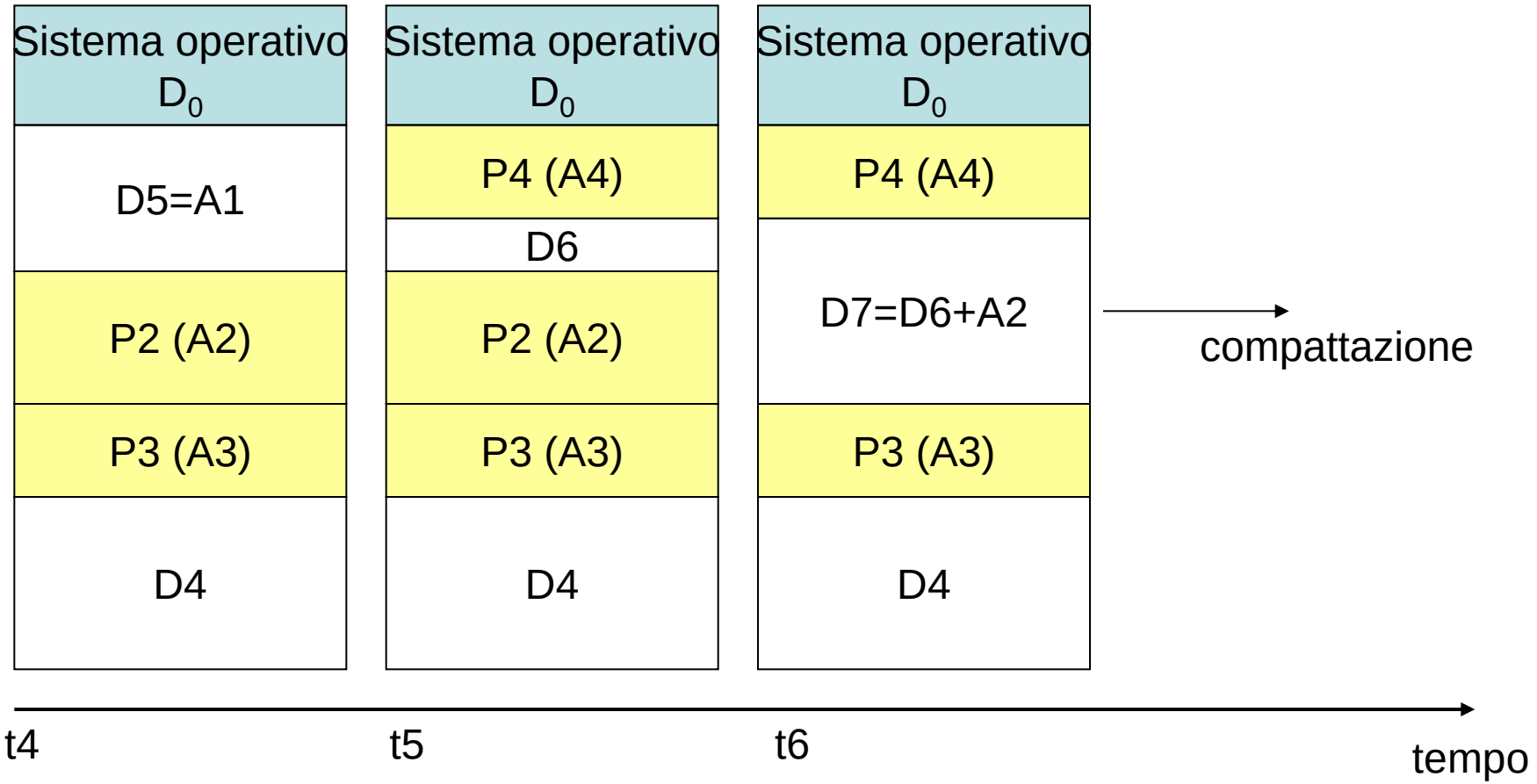
Lezione 12

Martedì 22-11-2016

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).

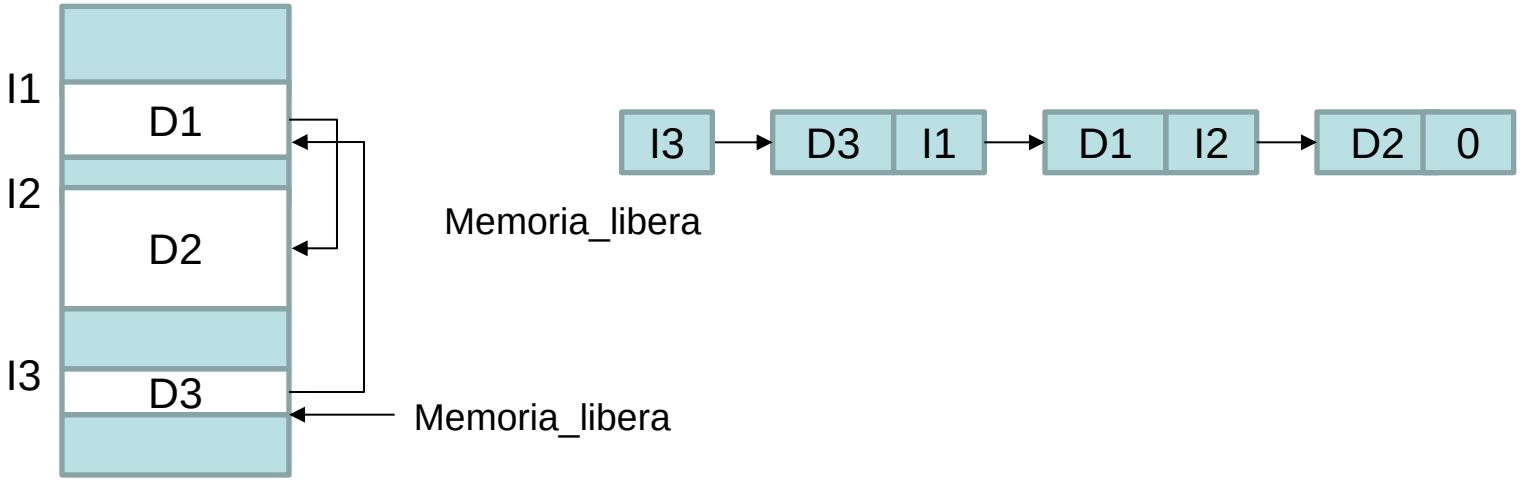
- In particolare, gestisce una lista concatenata, la **lista delle partizioni libere (free list)**, in cui ciascun elemento contiene la dimensione della partizione e l'indirizzo della successiva. Per velocizzare le operazioni di inserimento e cancellazione la lista può essere realizzata con doppio puntatore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

Tecniche di allocazione mediante free list

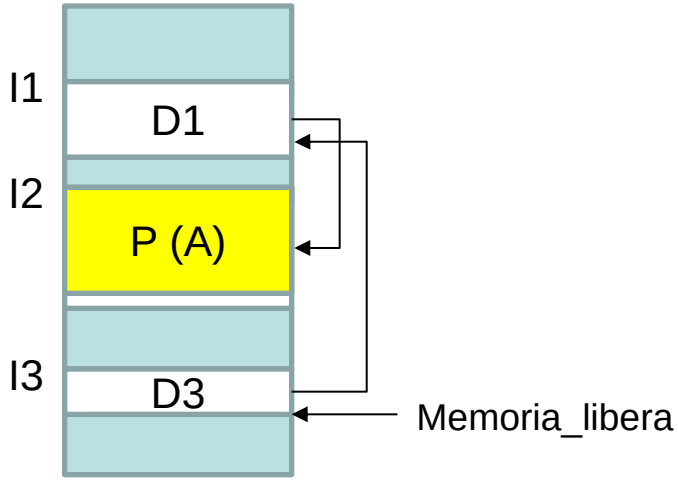
- Generalmente, nei sistemi che utilizzano la tecnica di gestione della memoria a partizione fisse, per allocare un processo in memoria, si utilizza un algoritmo, detto **best-fit**, che seleziona la partizione che ha una dimensione più vicina, in eccesso, alla dimensione dell'immagine del processo.
- 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.

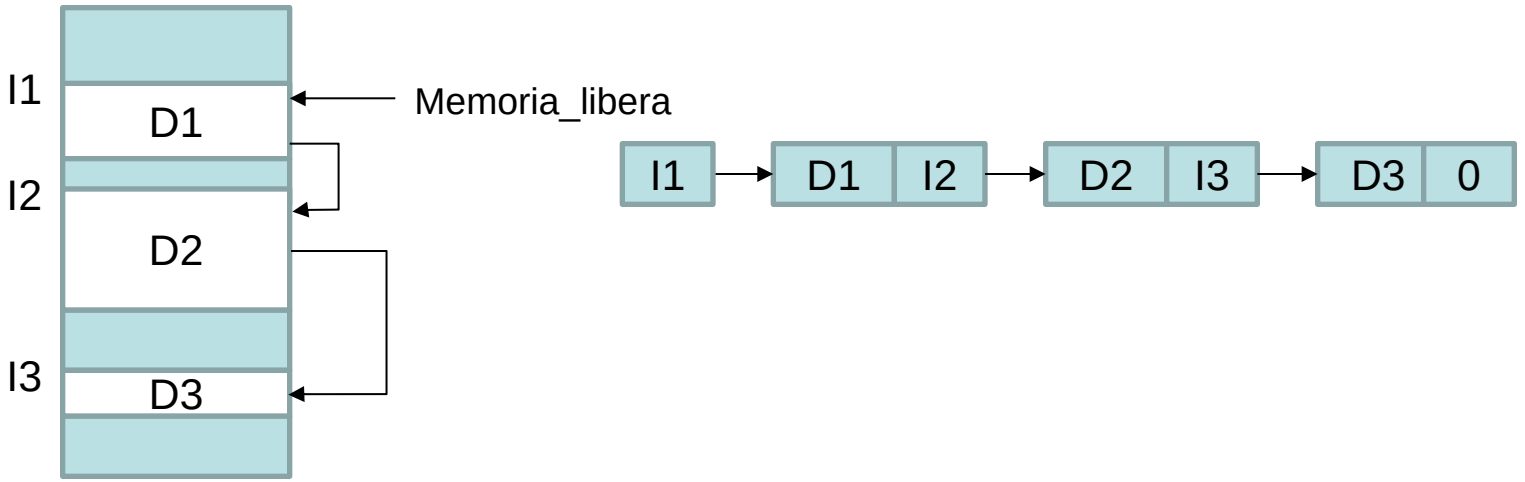


- Lo schema best-fit ha due inconvenienti:
  - Una volta effettuata l'allocazione, la parte della partizione non utilizzata è sicuramente quella di dimensioni più piccole e questo porta a far crescere la frammentazione della memoria.
  - In fase di rilascio è necessario verificare se la partizione liberata 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.



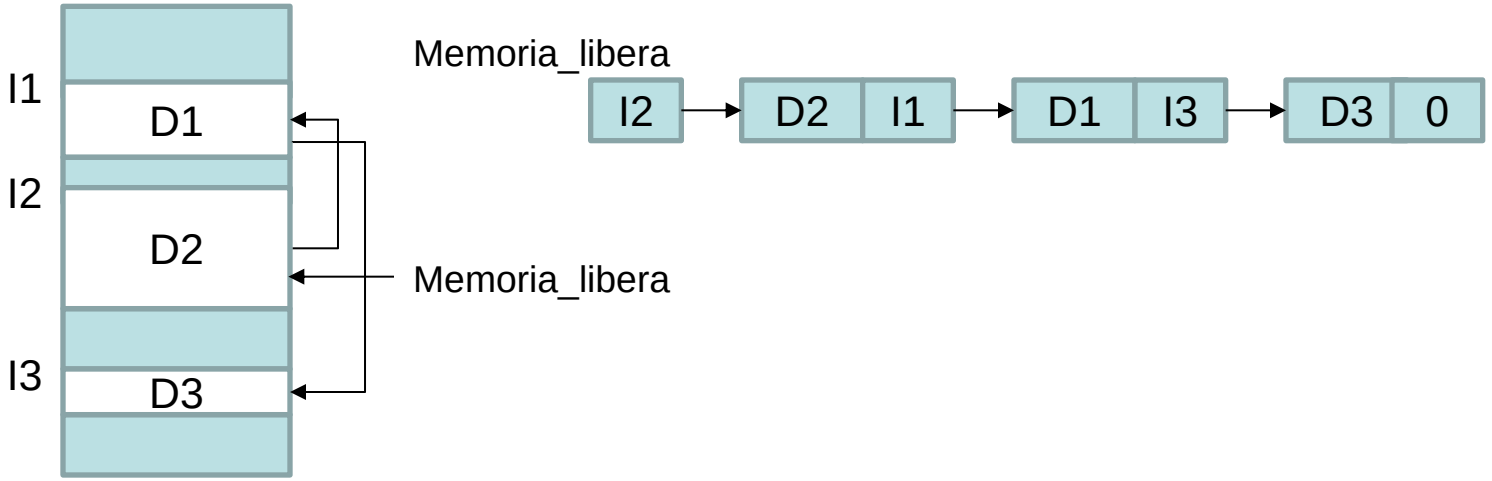
First-fit

- 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.



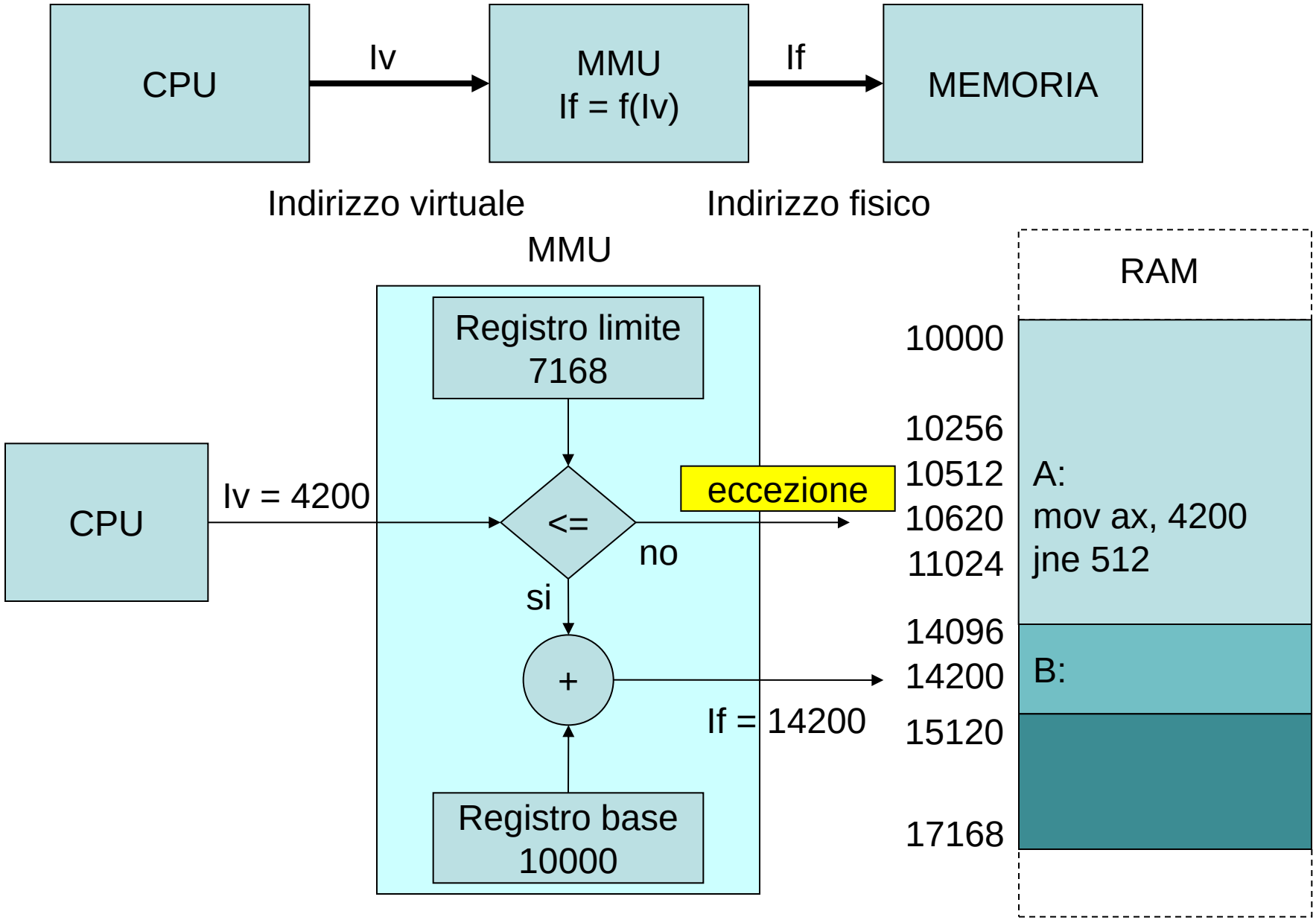
Worst-fit

- 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.



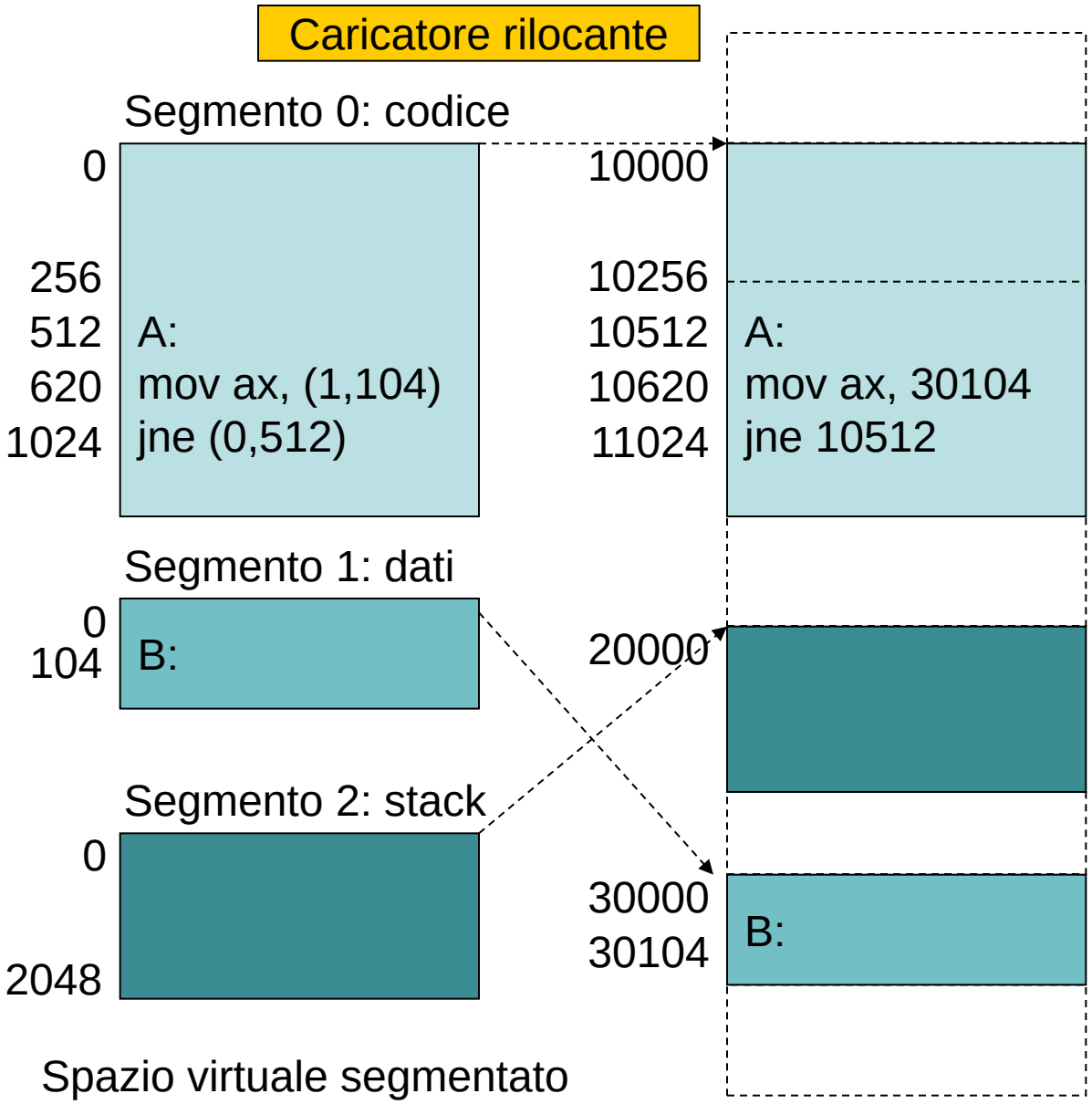
Protezione e condivisione di informazioni con spazio virtuale unico

- Il problema di garantire la **protezione** ai processi allocati contemporaneamente in memoria è spesso risolto a livello di architettura del processore, ad esempio utilizzando due particolari registri, il **registro base** e il **registro limite**, contenenti rispettivamente l'indirizzo iniziale e finale della partizione assegnata al processo in esecuzione.
- Ogni indirizzo generato dal processore è confrontato con i valori presenti nei due registri e nel caso in cui l'indirizzo sia al di fuori dell'intervallo delimitato dai due registri viene generata una interruzione di errore.
- Per quanto riguarda la **condivisione** di informazioni, il fatto di allocare in locazioni contigue uno spazio virtuale unico non consente ai processi di condividere informazioni.


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

Partizioni multiple

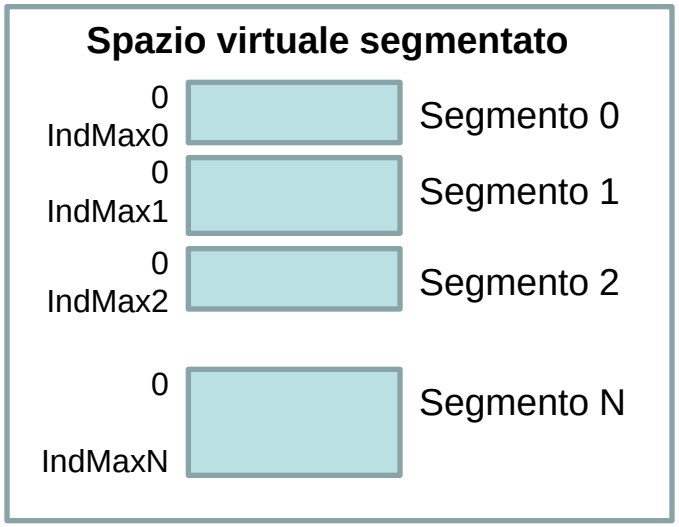
- La tecnica delle **partizioni multiple** richiede che lo spazio virtuale di un processo sia segmentato.
- Questa tecnica utilizza la **rilocazione statica** che consiste nel rilocare l'intero spazio virtuale del processo prima che questo inizi la sua esecuzione.
- Il linker divide lo spazio virtuale del processo in vari segmenti, ad esempio nei tre segmenti di codice, dati e stack.
- Con questa tecnica, a ogni processo è allocato un numero di partizioni pari al numero di segmenti in cui è stato partizionato lo spazio virtuale.
- I segmenti non necessariamente sono allocati in memoria in modo adiacente.
- Inoltre, essendo i segmenti separati più piccoli, rispetto al caso dello spazio virtuale unico, sarà più facile allocarli in memoria. La frammentazione esterna tende quindi a diminuire.


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

- Per eliminare completamente la frammentazione della memoria bisognerebbe unire tutte le partizioni libere e spostare anche le partizioni allocate ai processi, in modo da poter ottenere un’unica partizione libera contigua. Ma per spostare in memoria un processo è necessario ricorrere a tecniche di rilocazione dinamica, ad esempio usando le MMU. La MMU nel caso di processi con tre segmenti deve possedere tre coppie di registri base/limite, una coppia per ogni segmento.

Memoria segmentata

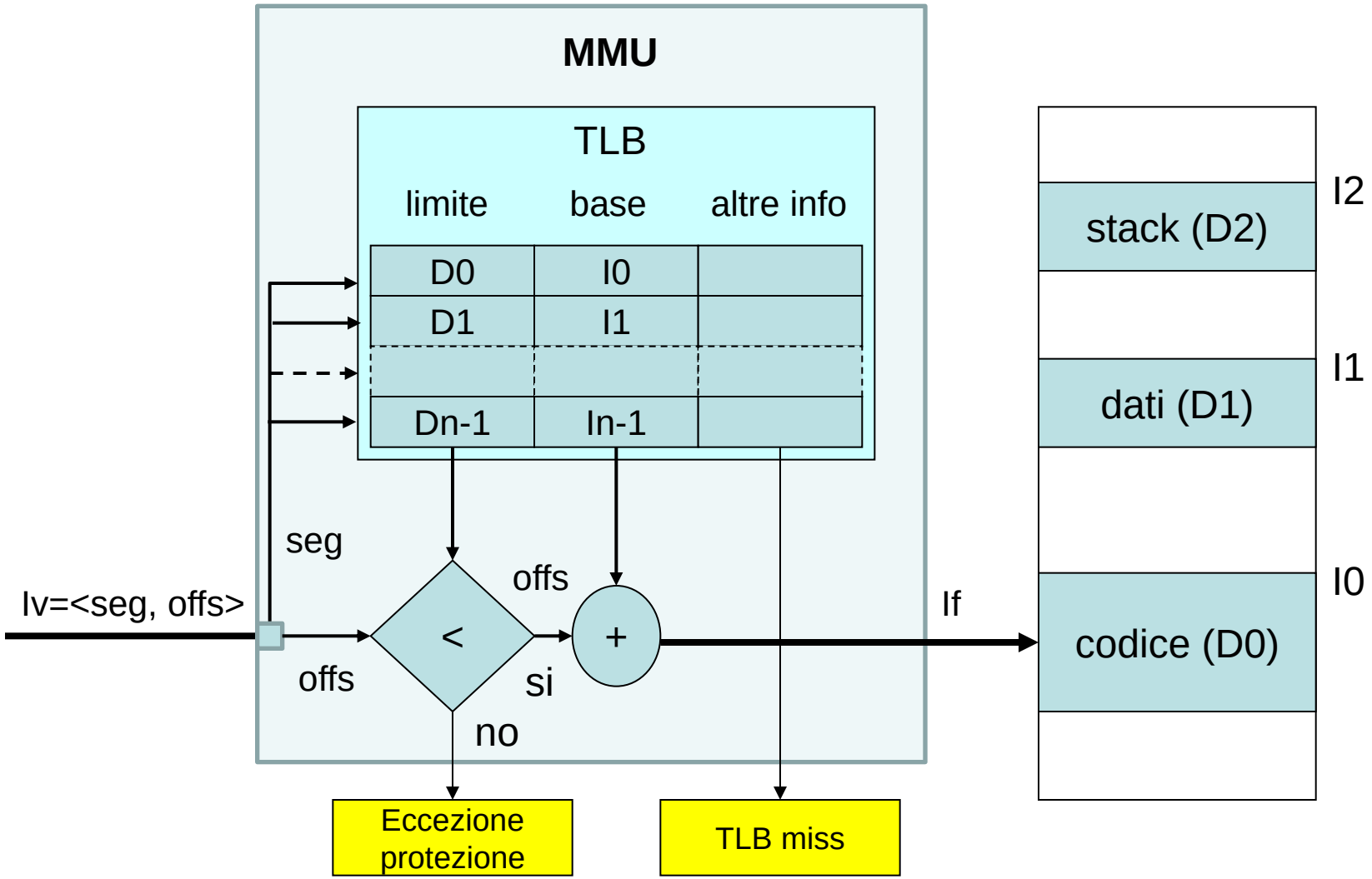
- La ***segmentazione*** è una tecnica di gestione della memoria simile a quella delle ***partizioni multiple*** con la variante di usare la rilocazione dinamica al posto della rilocazione statica.
- Nei moderni sistemi che utilizzano la segmentazione, il linker genera lo spazio virtuale assegnando un segmento per ogni modulo di programma quali le funzioni, procedure, strutture dati etc. In tal modo, lo spazio virtuale del processo, è costituito da molti segmenti che rispecchiano semanticamente la struttura del programma sorgente.



- Poiché lo spazio virtuale del processo è segmentato, per rilocare dinamicamente gli indirizzi virtuali generati dalla CPU è necessario l'uso della MMU.
- Nelle architetture di processori con supporto di segmentazione , la CPU genera indirizzi virtuali con il seguente formato bidimensionale:

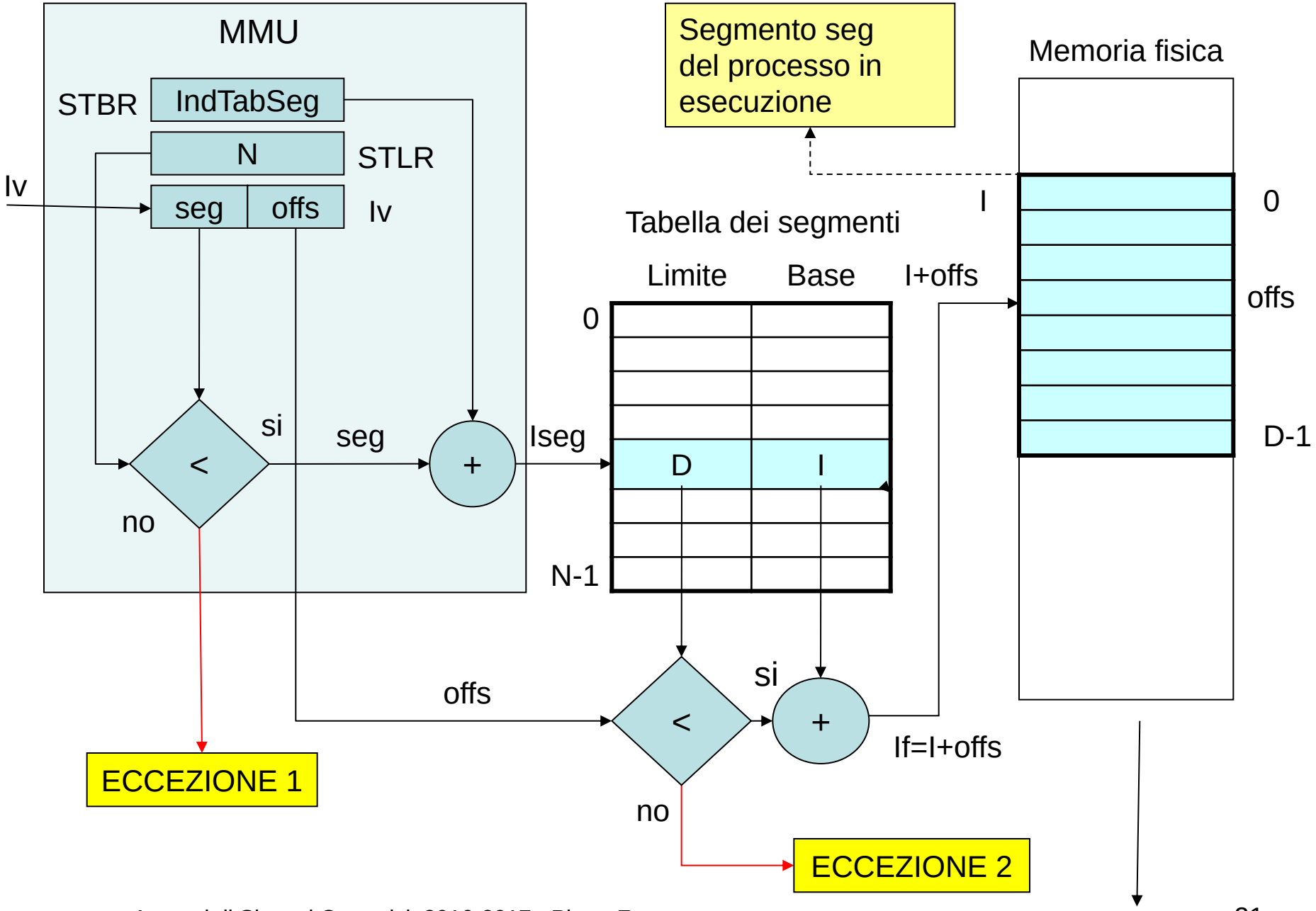
  Iv = <segmento, offset>

- **Dato che** le MMU hanno un buffer **(TLB**, **Translation Lookaside Buffer**) composto da pochi registri associativi, tipicamente da 32 a 1024, in cui memorizzare i valori limite e base, nel caso in cui il numero di segmenti è elevato non è più possibile memorizzare nei registri della MMU i dati relativi a tutti i segmenti.
- Ad esempio i microprocessori Intel della famiglia IA-32 hanno uno spazio virtuale che può avere fino a 2^{14} (16384) segmenti.



Traduzione degli indirizzi con MMU

- Pertanto i valori relativi all'indirizzo base e limite di ogni segmento sono memorizzati in una tabella, detta ***tabella dei segmenti,*** contenuta nella memoria principale e gestita dal kernel***.***
- Il **descrittore del processo** contiene, oltre alle informazioni già descritte, due campi relativi al suo spazio virtuale. Il primo contiene **il numero di segmenti** in cui è suddiviso lo spazio virtuale e il secondo memorizza **l'indirizzo fisico della tabella dei segmenti**.
- Questi due valori sono caricati in due registri della MMU, quando il processo passa in esecuzione. I due registri sono spesso indicati con i termini **STLR (Segment Table Limit Register)** e **STBR (Segment Table Base Register).**


MMU
STBR
IndTabSeg
N
STLR
lv
seg
offs
lv
<
si
no
seg
+
Iseg
Segmento seg
del processo in
esecuzione
Memoria fisica
I
0
Tabella dei segmenti
Limite
Base
I+offs
0
offs
D
I
D-1
N-1
offs
<
si
+
If=I+offs
ECCEZIONE 1
no
ECCEZIONE 2

- L’uso della tabella dei segmenti caricata in RAM rallenta la funzione di traduzione degli indirizzi rispetto al caso in cui i valori base e limite di ogni segmento sono contenuti nella veloce TLB della MMU.
- Per accedere ad una locazione di memoria sono ora **necessari due accessi**: il primo accesso alla tabella dei segmenti e il secondo alla locazione vera e propria.
- Tuttavia, anche se le MMU hanno una **TLB** composta da pochi registri associativi, la traduzione degli indirizzi si ottiene per la maggior parte attraverso essa, senza accedere alla tabella in RAM.
- Infatti, quando la CPU genera un indirizzo virtuale **`Iv = <seg,offset>`** la sua traduzione avviene dapprima ricercando i dati riguardanti il segmento ***seg*** nella MMU. Se i dati sono presenti nella MMU la traduzione degli indirizzi si ottiene in base ai valori base e limite presenti nel registro associativo, altrimenti la traduzione avviene in base allo schema della figura precedente e nella MMU sono copiati i dati relativi alla traduzione.

- Giacché un programma è strutturato in moduli, gli accessi in memoria sono spesso localizzati. Pertanto, si ha che con pochi registri associativi si possono tradurre l'80% degli indirizzi.
- La percentuale di volte che la traduzione dell'indirizzo avviene con la **TLB** è detta **hit ratio**. Un hit ratio pari a 0.8 significa che l'80% delle traduzioni è risolto con la TLB.
- Ad esempio, se la ricerca nella TLB richiede 10 ns e sono necessari 100 ns per accedere alla memoria, allora nel caso in cui la traduzione avviene con la TLB, un accesso ad una locazione di memoria, per un'istruzione o un dato, richiede 110 ns. Se, invece, la TLB non contiene la traduzione occorrono 10 ns per accedere alla TLB, più 100 ns per accedere alla tabella dei segmenti più 100 ns per accedere al dato desiderato in memoria; in totale sono necessari 10+100+100=210 ns.
- Per calcolare **il tempo effettivo di accesso** alla memoria bisogna tener conto della probabilità dei due casi:

Tempo effettivo d'accesso = $p \cdot (T_{TLB}+T_M) + (1-p) \cdot (T_{TLB}+2 \cdot T_M)$

Per l'esempio precedente si ha:

Tempo effettivo d'accesso = $0.80 \cdot 110+0.20 \cdot 210=130$ ns

- La segmentazione dello spazio virtuale complica notevolmente la funzione di traduzione degli indirizzi ma d'altra parte presenta grandi vantaggi.
- Due importanti vantaggi prodotti dalla segmentazione sono relativi alla **protezione** e alla **condivisione** dei segmenti.
- La segmentazione consente di ottenere vari tipi di controllo quando un processo accede alla memoria.
- Oltre ai i due controlli di protezione evidenziati nella figura precedente, ad ogni segmento possono essere associati **vari diritti di accesso**, come ad esempio il diritto in accesso in **sola lettura** (segmento di codice, segmento contenente solo costanti), **scrittura** (segmento dati), **lettura e scrittura** (segmento dati) etc.

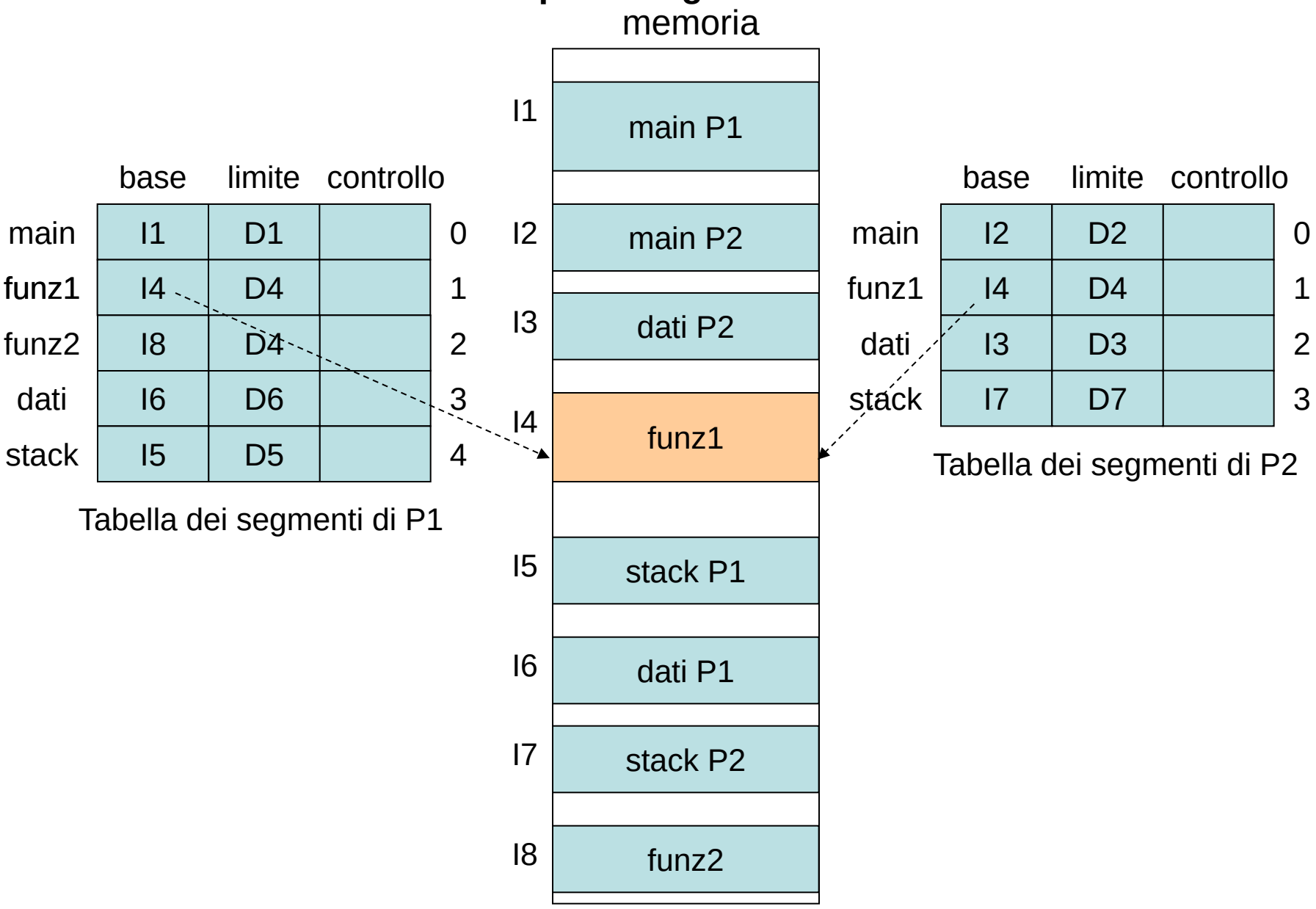
- Per consentire questi controlli nella tabella dei segmenti si aggiungono nuovi campi, come mostrato nella figura seguente.

base	limite	controllo		altre info
Indirizzo del segmento	dimensione del segmento	R	W	

descrittore della tabella del segmento

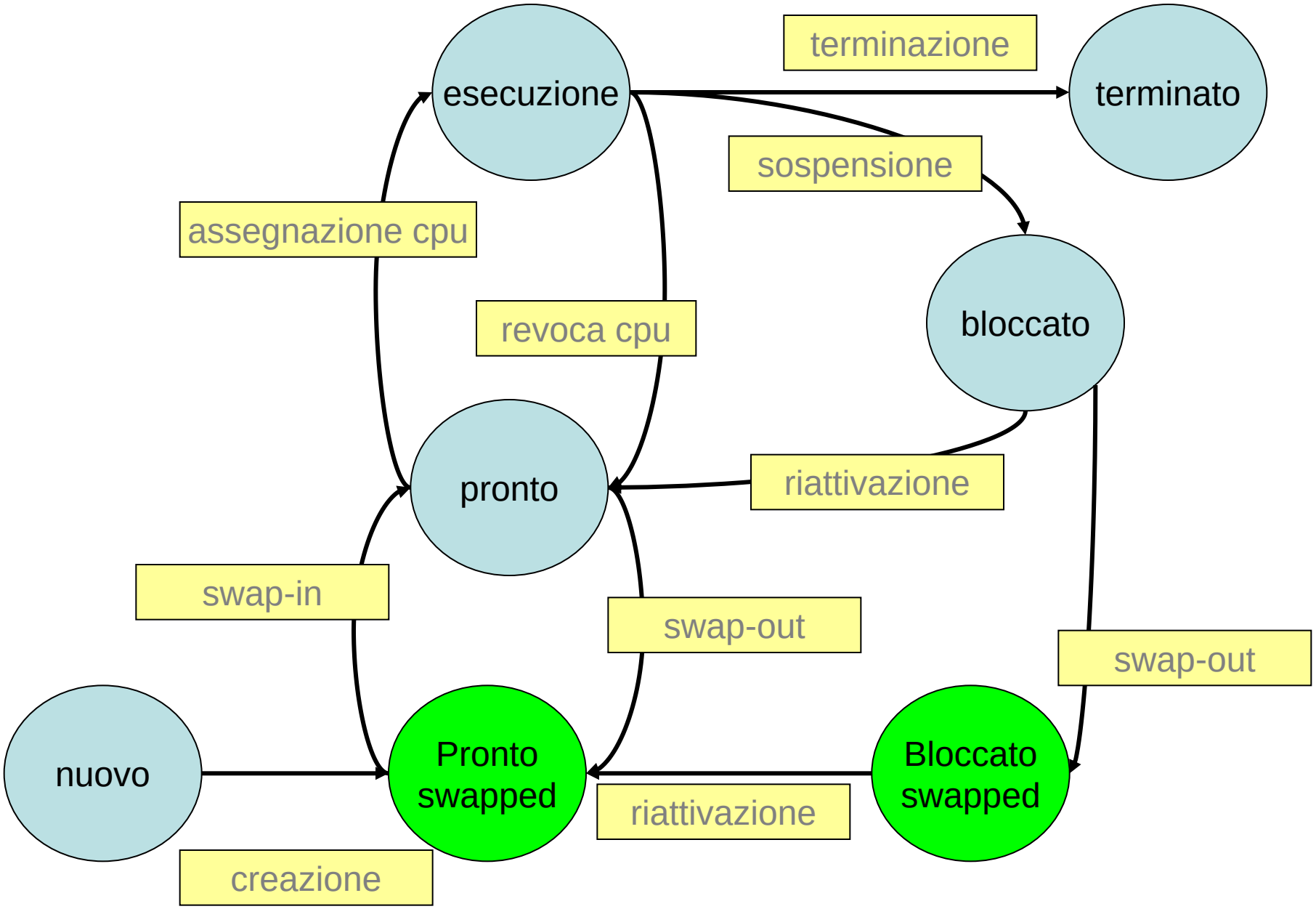
- Come già detto, la segmentazione consente una corrispondenza semantica tra moduli di un programma e segmenti. Pertanto è possibile condividere alcuni segmenti tra più processi, come mostrato nella figura seguente. Nella figura sono rappresentati gli spazi virtuali di due processi P1 e P2 mediante le relative tabelle dei segmenti. P1 ha 5 segmenti, P2 ne ha 4. Il segmento **funz1** è condiviso.

Esempio di segmento condiviso



- Da quanto fin ora descritto, con la tecnica della segmentazione un processo si può trovare in due condizioni:
  1. **Allocato in memoria**
     tutti i suoi segmenti sono allocati nella memoria fisica
  2. **Non allocato in memoria**
     tutti i suoi segmenti sono nella swap-area, su disco.

- Per tener conto di questa situazione è necessario aumentare gli stati in cui si può trovare un processo.
- Si aggiungono due stati detti **pronto-swapped** e **bloccato-swapped** per rappresentare i processi quando non hanno lo spazio virtuale allocato in memoria fisica.


esecuzione
terminazione
terminato
sospensione
assegnazione cpu
bloccato
revoca cpu
pronto
riattivazione
swap-in
swap-out
swap-out
nuovo
Pronto swapped
Bloccato swapped
riattivazione
creazione

- Come abbiamo già detto, le transizioni (assegnazione CPU) dallo stato di pronto allo stato di esecuzione e quella dallo stato di pronto-swapped allo stato di pronto (swap-in) sono eseguite rispettivamente dallo **scheduler a breve termine** e dallo **scheduler a medio termine**.
- Il caricamento in memoria dell'intero spazio virtuale, prima dell'esecuzione del processo, implica che la dimensione dello spazio virtuale debba essere inferiore alla dimensione della memoria fisica disponibile.