Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2017-18

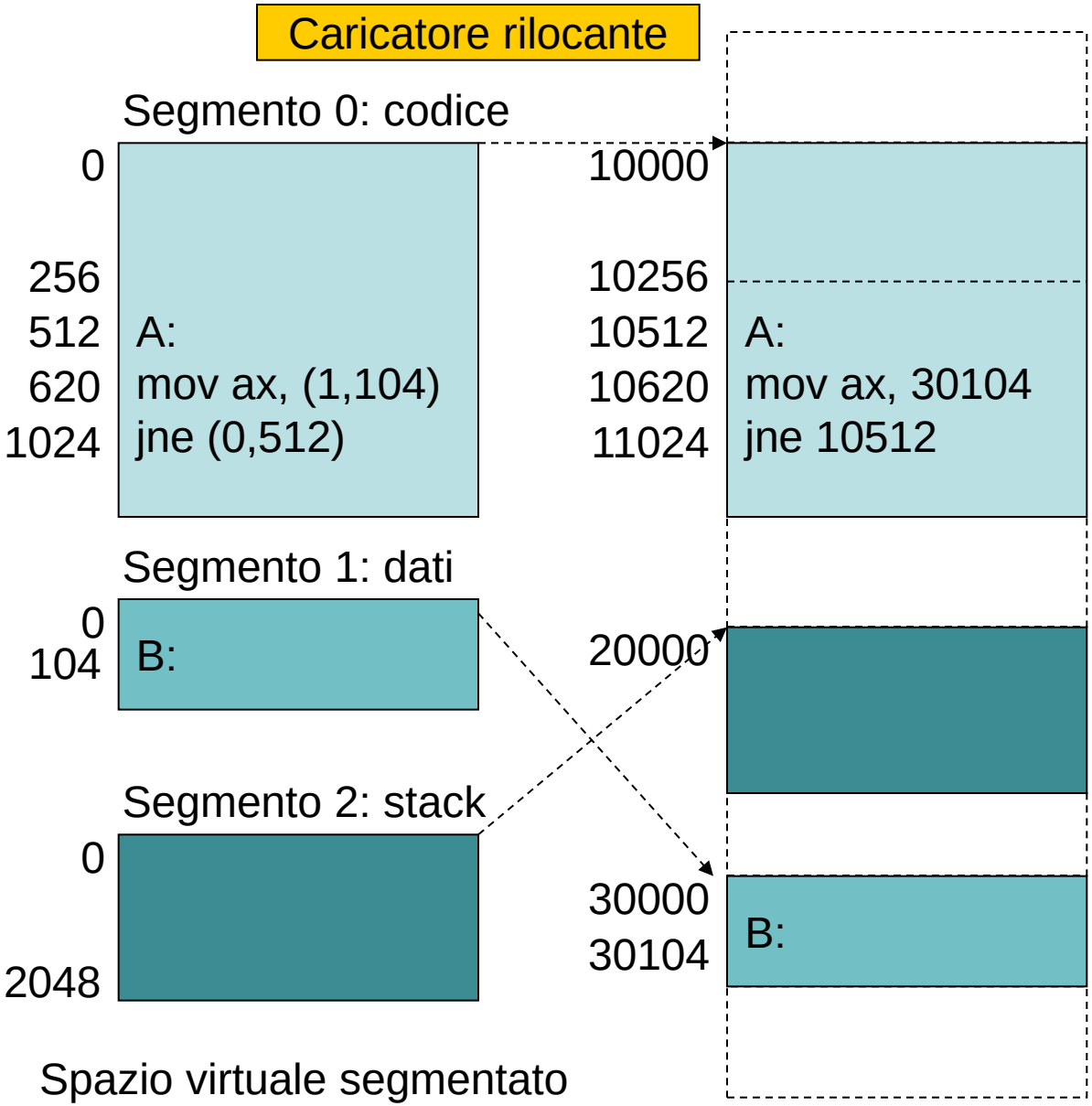
Pietro Frasca

Lezione 18

Martedì 5-12-2017

Partizioni multiple

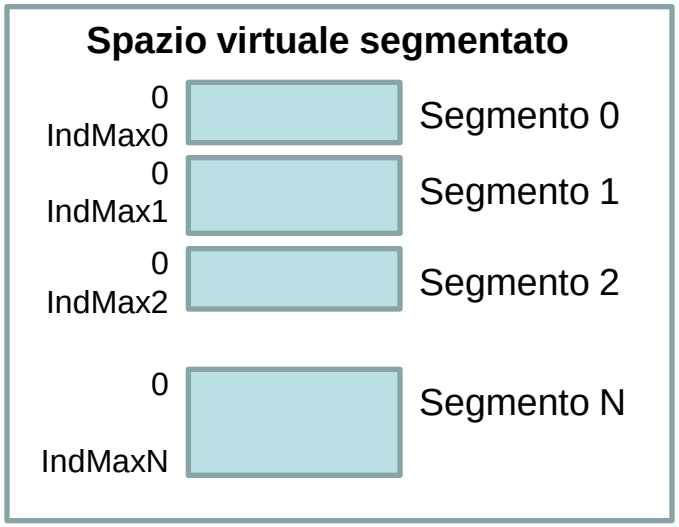
- La tecnica delle **partizioni multiple** richiede che lo spazio virtuale di un processo sia segmentato.
- Questa tecnica utilizza la **rilocazione statica** che consiste nel rilocare l'intero spazio virtuale del processo prima che questo inizi la sua esecuzione.
- Lo spazio virtuale del processo è diviso 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 diviso 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)
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

- 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 il supporto MMU*. Tale supporto, nel caso di processi con tre segmenti, deve possedere tre coppie di registri base/limite, una coppia per ogni segmento.*

Memoria segmentata

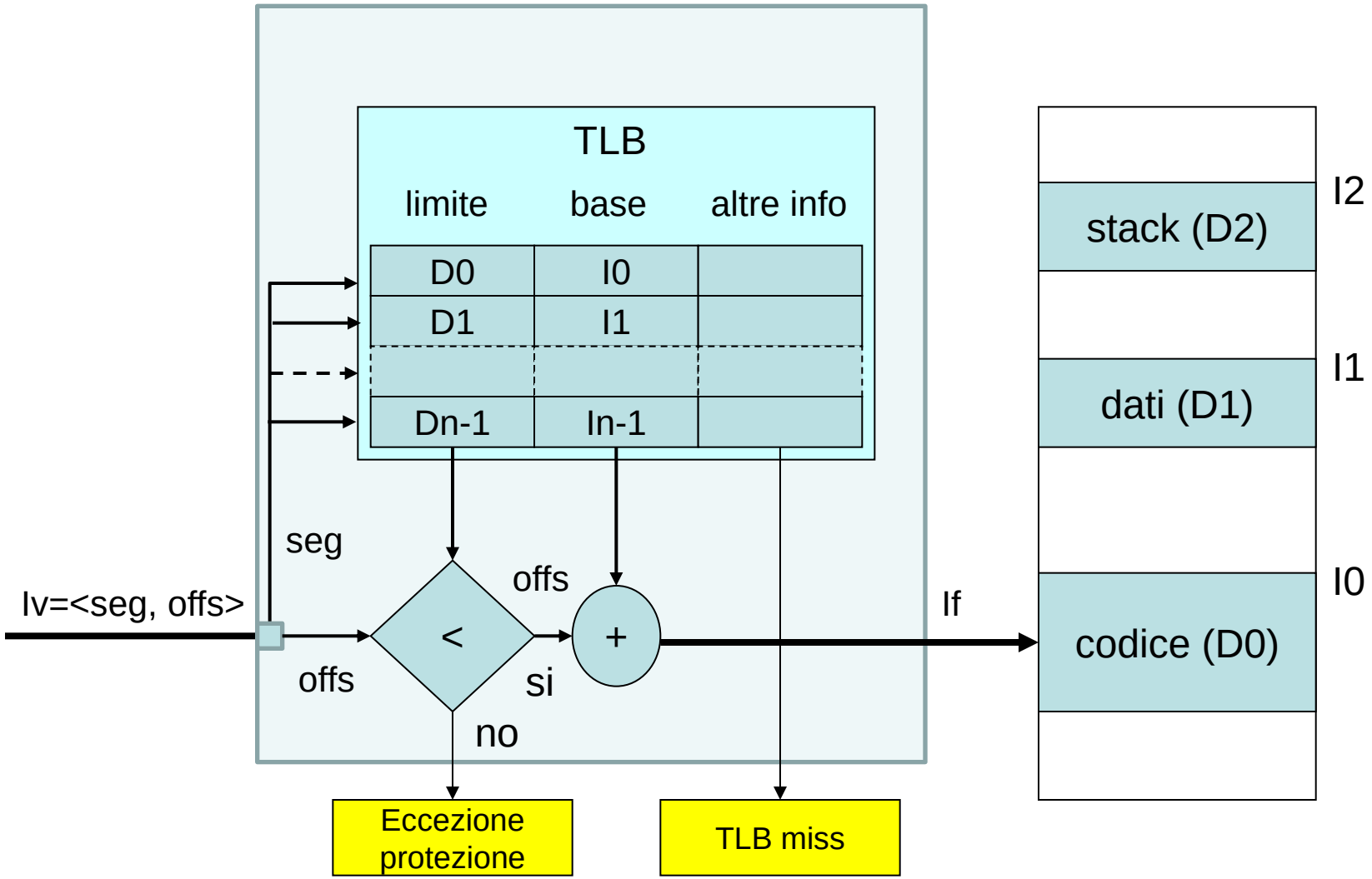
- La ***segmentazione*** è una tecnica di gestione della memoria simile a quella delle ***partizioni multiple*** con la variante di usare la rilocazione dinamica al posto della rilocazione statica.
- Nei moderni sistemi che utilizzano la segmentazione, è possibile strutturare lo spazio virtuale di un processo 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.



- Nelle architetture di processori con supporto di segmentazione, gli indirizzi virtuali sono generati con il seguente formato bidimensionale:

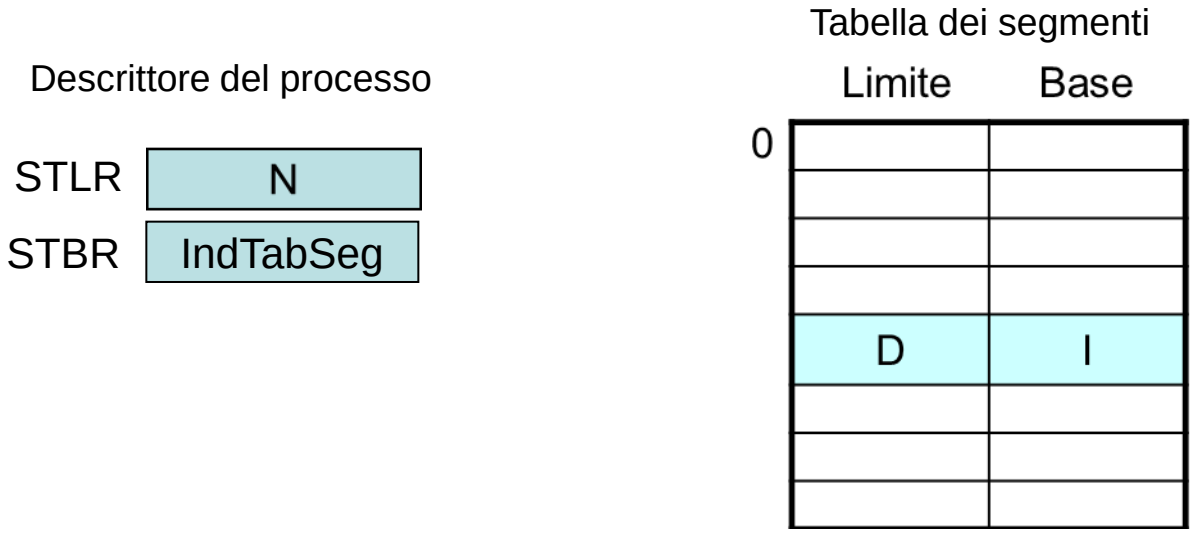
  Iv = <segmento, offset>

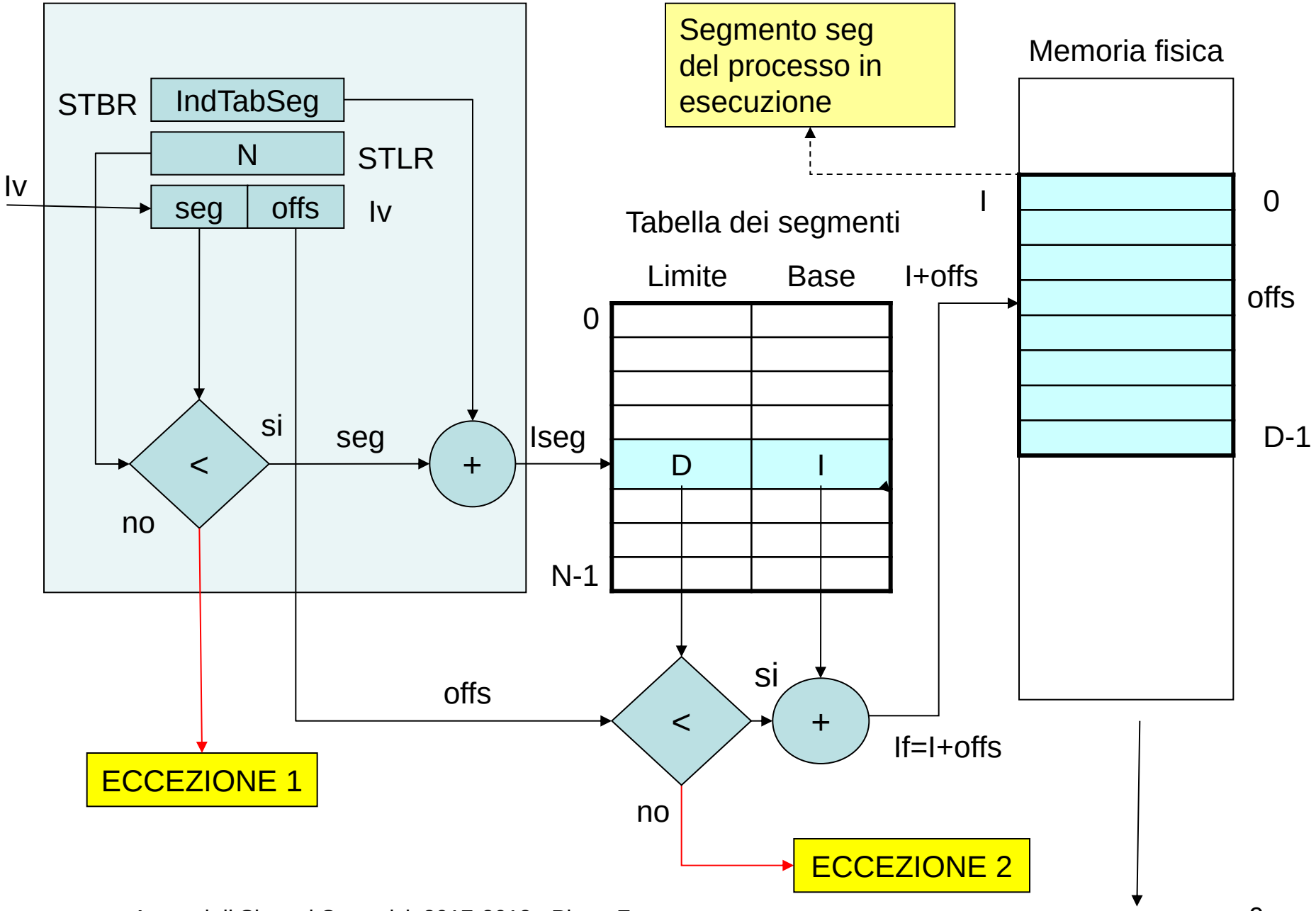
- Poiché i processori hanno un buffer **(TLB**, **Translation Lookaside Buffer**) composto da pochi registri associativi, tipicamente da 32 a 1024, in cui memorizzare i valori limite e base, nel caso in cui il numero di segmenti è elevato non è più possibile memorizzare nella TLB i dati relativi a tutti 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 TLB

- 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 (PCB) 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 del processore, 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).**




Segmento seg del processo in esecuzione
Memoria fisica
STBR
IndTabSeg
N
STLR
lv
seg
offs
lv
Tabella dei segmenti
Limite
Base
I+offs
I
0
offs
D-1
0
N-1
D
I
si
seg
<
+
lseg
no
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 del processore.
- 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 i processori 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 **lv = <seg,offset>** la sua traduzione avviene dapprima ricercando i dati riguardanti il segmento ***seg*** nella TLB. Se i dati sono presenti nella TLB 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 TLB 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.
- La ricerca nelle TLB attuali è praticamente trascurabile.
- Ad esempio, 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 100 ns. Se, invece, la TLB non contiene la traduzione occorrono 100 ns per accedere alla tabella dei segmenti più 100 ns per accedere al dato desiderato in memoria; in totale sono necessari 100+100=200 ns.
- Per calcolare **il tempo effettivo di accesso** alla memoria bisogna tener conto della probabilità dei due casi:

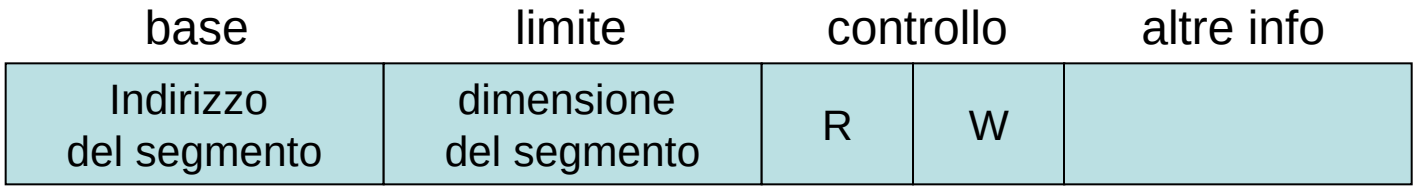
Tempo effettivo d'accesso = $p \cdot (T_{TLB}+T_M) + (1-p) \cdot (T_{TLB}+2 \cdot T_M)$

Per l'esempio precedente si ha:

Tempo effettivo d'accesso = $0.80 \cdot 100+0.20 \cdot 200=120$ 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.

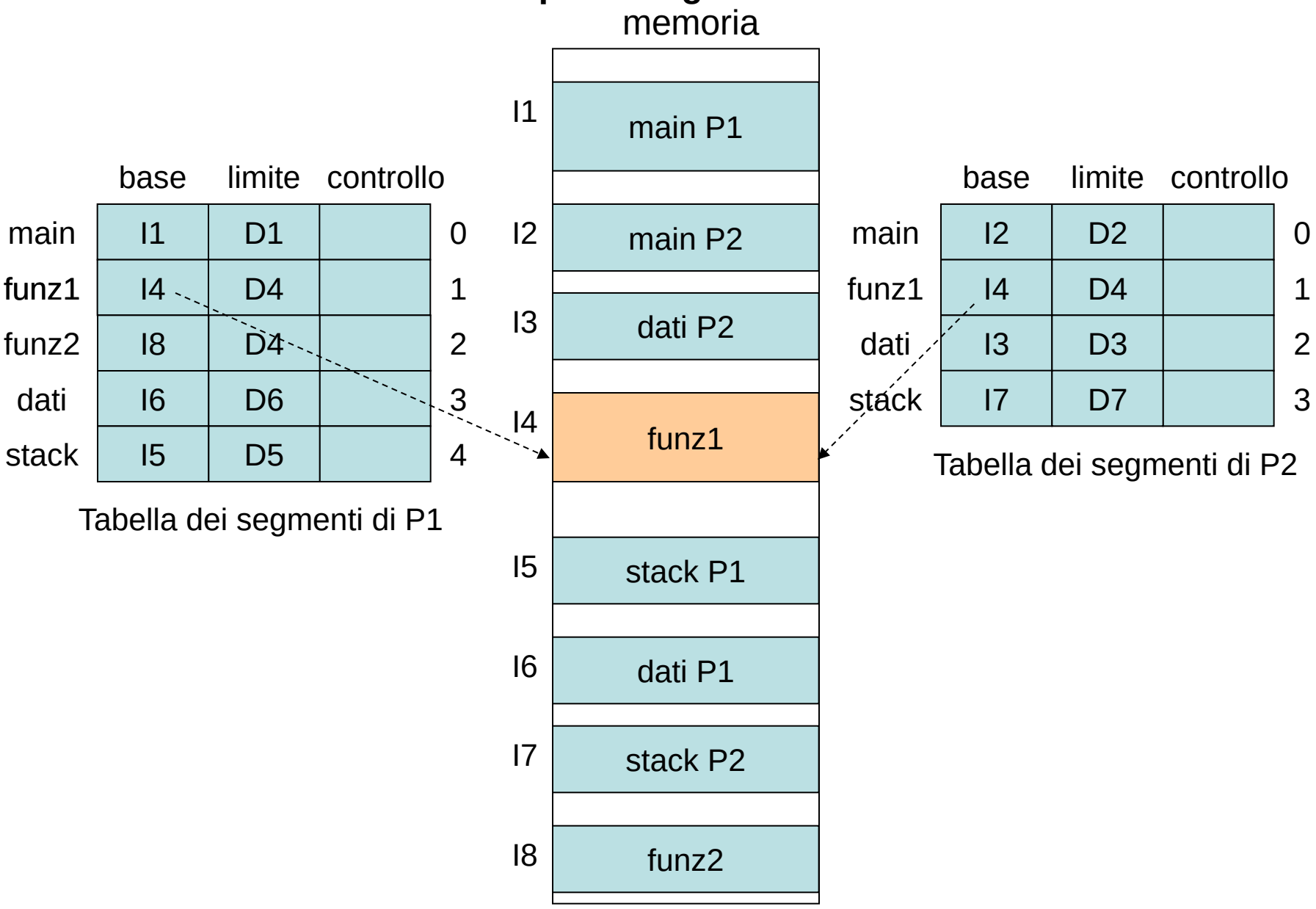
- Per consentire questi controlli nella tabella dei segmenti si aggiungono nuovi campi, come mostrato nella figura seguente.



descrittore della tabella del segmento

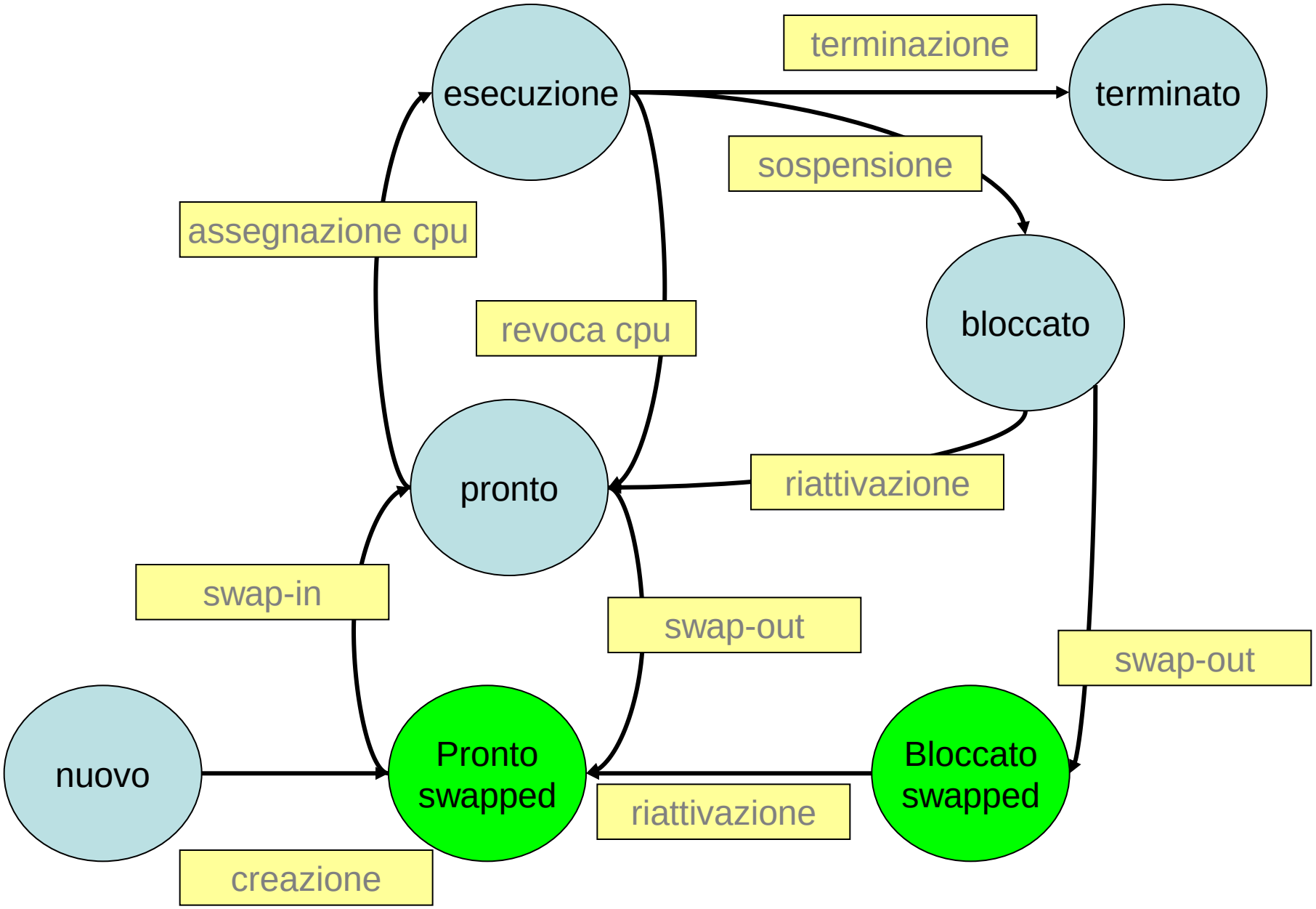
- Come già detto, la segmentazione consente una corrispondenza semantica tra moduli di un programma e segmenti. Pertanto è possibile condividere alcuni segmenti tra più processi, come mostrato nella figura seguente. Nella figura sono rappresentati gli spazi virtuali di due processi P1 e P2 mediante le 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.