memoria centrale

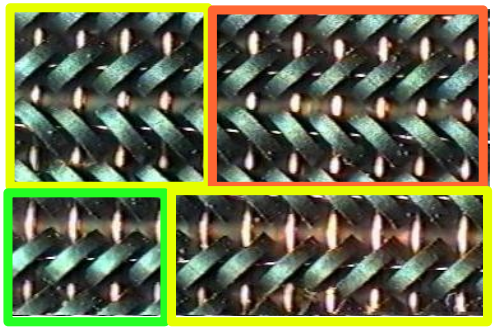
capitolo 8 del libro (VII ed.)

Cristina
Baroglio

Introduzione

- La memoria principale, come la CPU, è una risorsa condivisa fra i processi
- È necessario imporre dei meccanismi di gestione
- Alcuni metodi vengono messi a disposizione già a livello HW
- Altri metodi sono realizzati a un livello di astrazione superiore, fra questi:
  - paginazione della memoria
  - segmentazione della memoria
- La scelta del metodo da adottare dipende da diversi fattori, *in primis* l'architettura considerata

ripartiamo dal livello 0 (HW) e pian piano costruiamo le sovra-strutture necessarie



Core Memory - RAM - 1951

SOVRASTRUTTURA
suddivisione fra i processi

Livello Hardware

- RAM: unica memoria di grandi dimensioni direttamente accessibile alla CPU
- la CPU preleva le istruzioni da eseguire (operazione fetch) e i dati da elaborare (load) dalla RAM e memorizza in essa i risultati dell'elaborazione (store). Ciclo:
  - fetch: individua tramite program counter la prossima istruzione e caricala nell'instruction register
  - decode: decodifica l'operazione tramite un codice operativo
  - execute: individua e carica i dati richiesti ed esegui l'operazione
- **NB**: la CPU riceve ed utilizza un flusso di indirizzi di memoria, non sa come questi siano generati né a cosa servano
- Concentriamoci quindi per ora sui **singoli indirizzi**

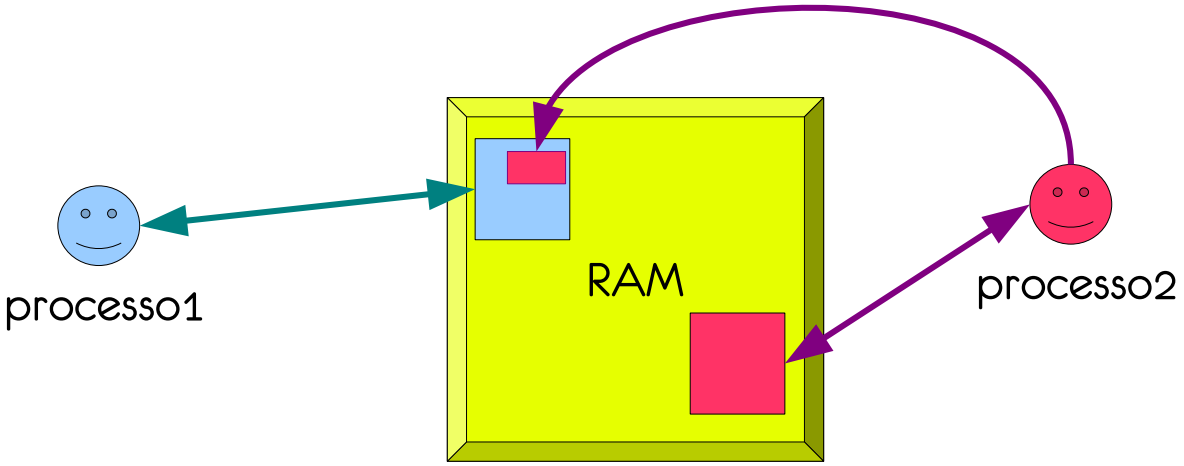
Schema generale: livello 0

sono incorporati nella CPU, ciò consente di effettuare una o più operazioni che usano solo i registri per ogni ciclo di clock

La RAM è accessibile via bus
talvolta l'accesso alla RAM richiede diversi cicli di clock durante i quali la CPU è in attesa

Per consentire un uso migliore della CPU spesso si frappone un buffer (cache) fra RAM e CPU. Scopo della cache: mediare la lentezza di interazione con la RAM mettendo a disposizione dati "probabilmente utili"

Memoria e processi



Processo 1 e Processo 2 usano ciascuno una porzione di RAM …

… e se per errore Processo 2 scrivesse un proprio dato sopra a un'istruzione di Processo 1?

Occorre attuare meccanismi di protezione

La RAM è condivisa da vari processi, per ogni processo la RAM contiene il suo codice (completo o parziale) e i dati da elaborare. È necessario far sì che un processo non vada a sovrascrivere il/i codice/dati di un altro (specie se quest'ultimo è un processo del SO!!!)

In altri termini occorre definire meccanismi che consentono di **assegnare a ogni processo un'area di memoria separata**

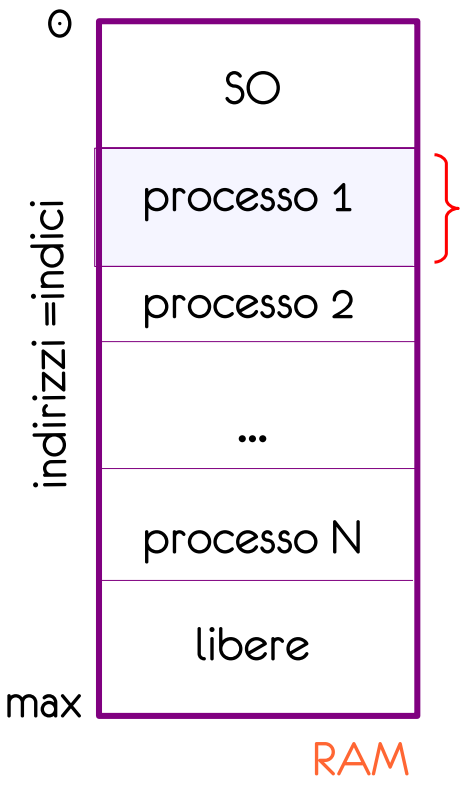
Processi e spazi degli indirizzi

- Un processo in esecuzione è caricato in RAM (codice, dati)
- L'esecuzione di un processo comporta la produzione di indirizzi (di istruzioni, di dati)
- La CPU produce un flusso di indirizzi che consentono al processo l'accesso a elementi contenuti in RAM
- Occorre controllare che un processo acceda solo agli indirizzi appartenenti alla porzione di RAM ad esso assegnata. Esempio di codice che produce errori di accesso in RAM:

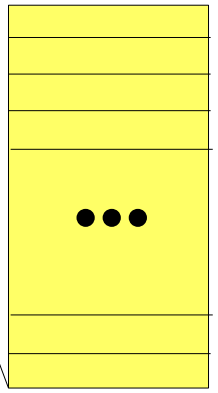
```
for (i=0 ;  ;  i++) mio_array[i] = 0;
```

- Vi sono diversi modi di realizzare il mapping processo-RAM, per cominciare consideriamo il più semplice

Registro base e limite 1/3



porzione di RAM del processo 1



indice *j* (indirizzo) della prima parola di memoria

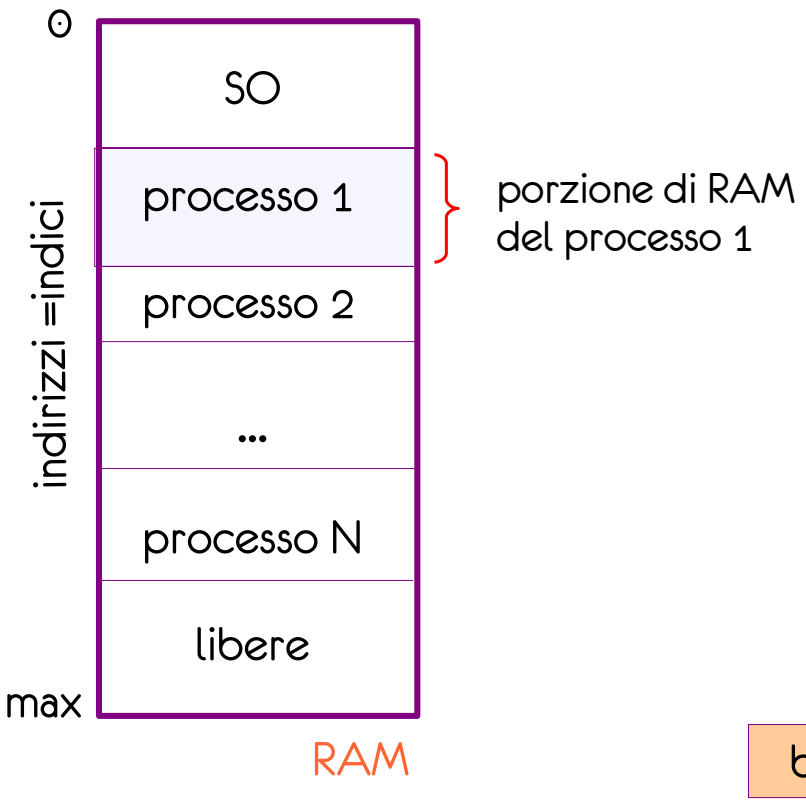
indice *k* (indirizzo) dell'ultima parola di memoria

Una porzione di RAM è, a basso livello, una porzione di un array

Per identificare una porzione di un array ho bisogno di due indici: l'indice del primo elemento e l'indice dell'ultimo elemento.

Posso conservare tale coppia di valori per ciascun processo

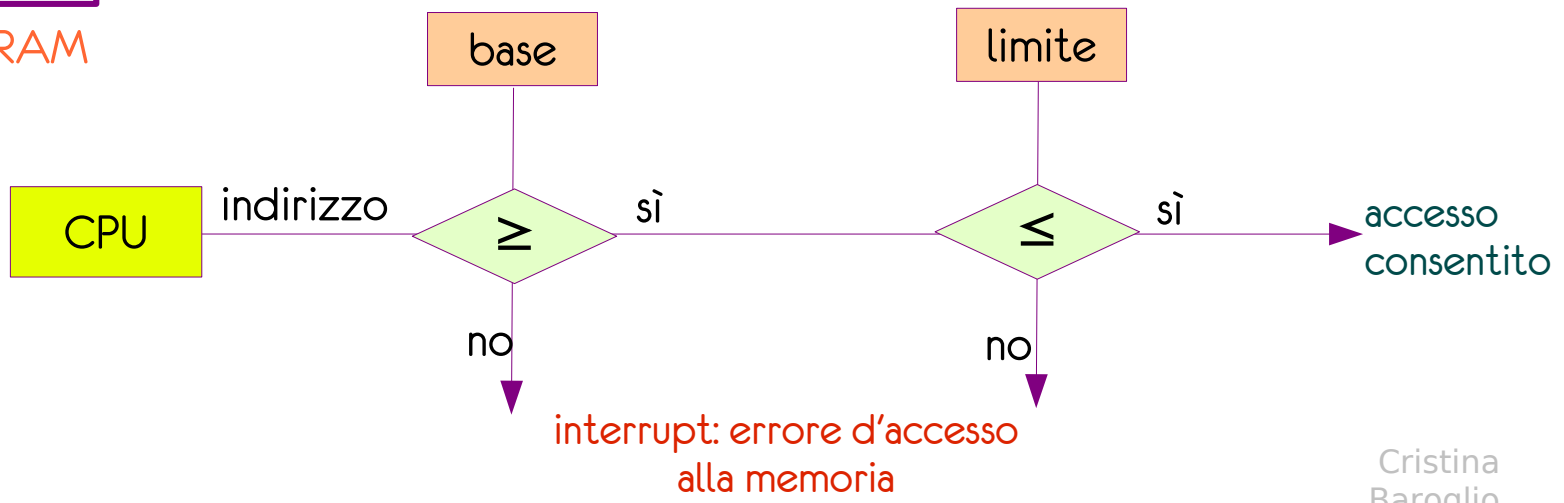
Registro base e limite 2/3



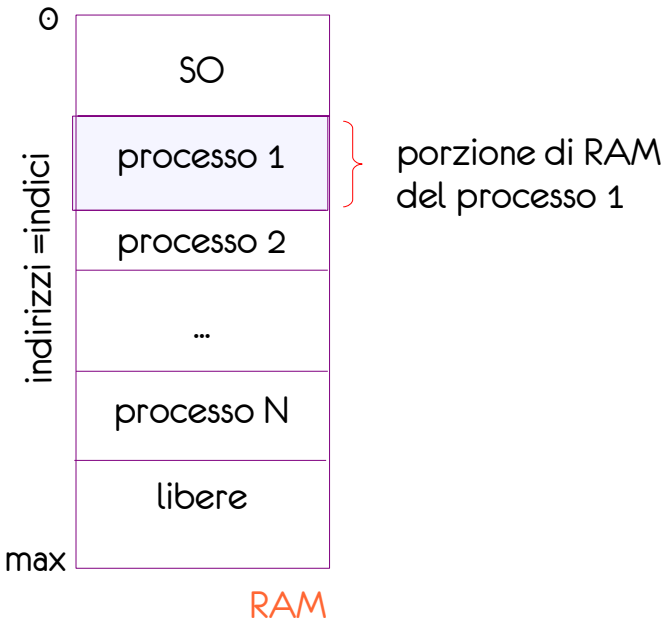
Il SO (e solo il SO) può eseguire un'istruzione che carica tali valori in due registri (registro base e registro limite)

Meccanismo di controllo:

Prodotto un indirizzo la CPU lo confronta con il registro base e il registro limite



Registro base e limite 3/3

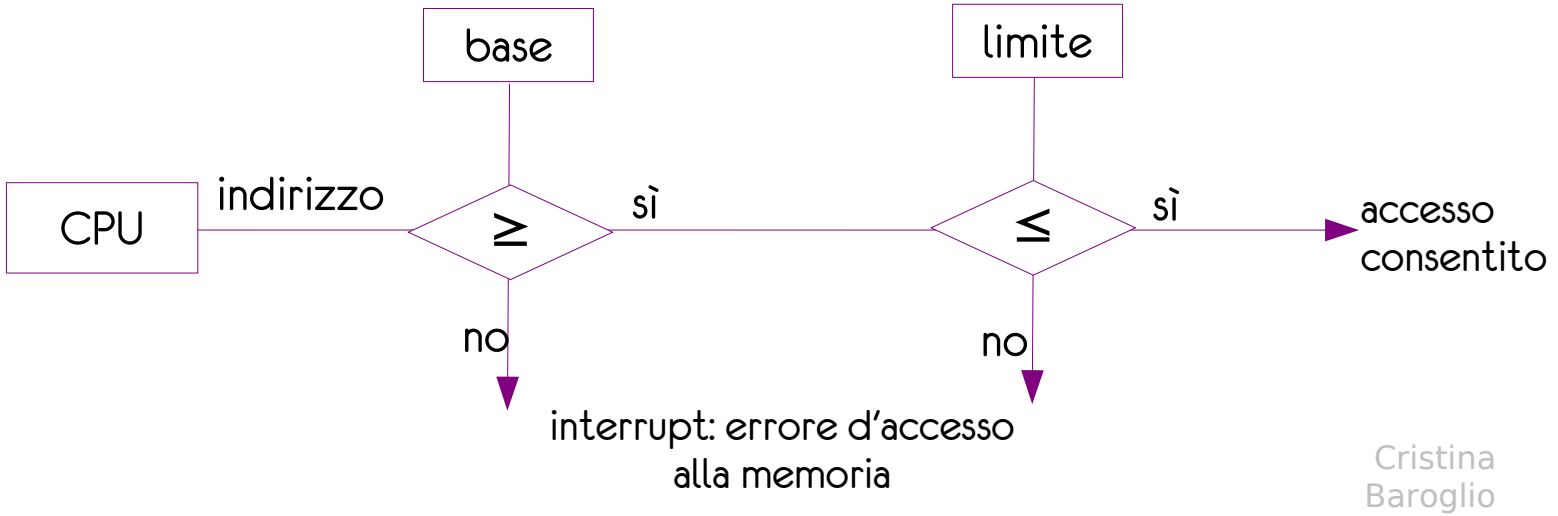


Il SO (e solo il SO) può eseguire un'istruzione che carica tali valori in due registri (registro base e registro limite)

A quale porzione di RAM può accedere il SO?
Tutta, perché per esempio si occupa di caricare in RAM i processi pronti da eseguire

Meccanismo di controllo:

Prodotto un indirizzo la CPU lo confronta con il registro base e il registro limite



Binding fra variabili e indirizzi

```
...
void inizializzaRandom(grafo *G) {
    int numN, i, j;
    printf("\n Quanti nodi vuoi creare
    scanf("%d", &numN);

    (*G).nodes = (nodo *) malloc(sizeo
    (*G).numN = numN;
    (*G).edges = (connessioni)malloc(s

    for (i=0; i<numN; i++) {
        (*G).nodes[i].dato = i; /
        (*G) nodes[i] visitato = 0;
                                co*)malloc(sizeof(arco) * numN);
    }

....
```

uso nomi di variabili

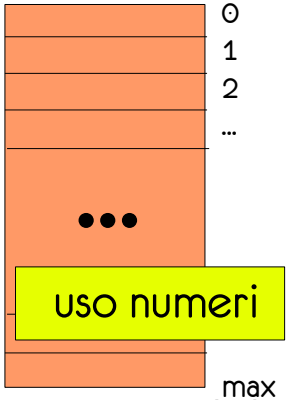
un processo esegue un programma scritto in un linguaggio di programmazione

all'interno del programma si dichiarano e si utilizzano delle **variabili** (i, j, G, numN, ...) e delle **procedure** (somma, cerca, ...)

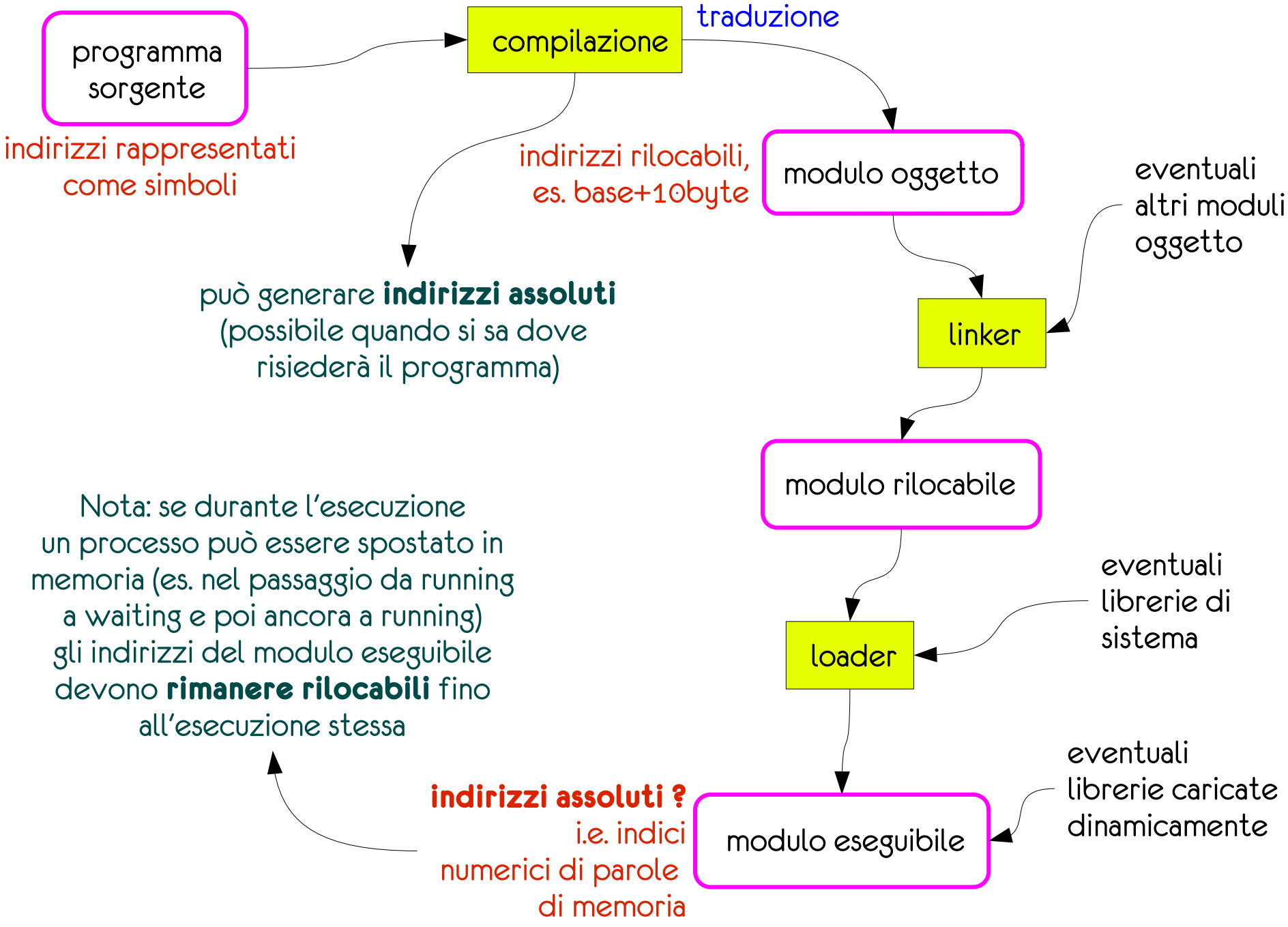
a livello di RAM si utilizzano degli indirizzi cioè degli **indici** di parole di memoria

quali passaggi ci sono nel mezzo?

e se ho più esecuzioni contemporanee dello stesso programma?



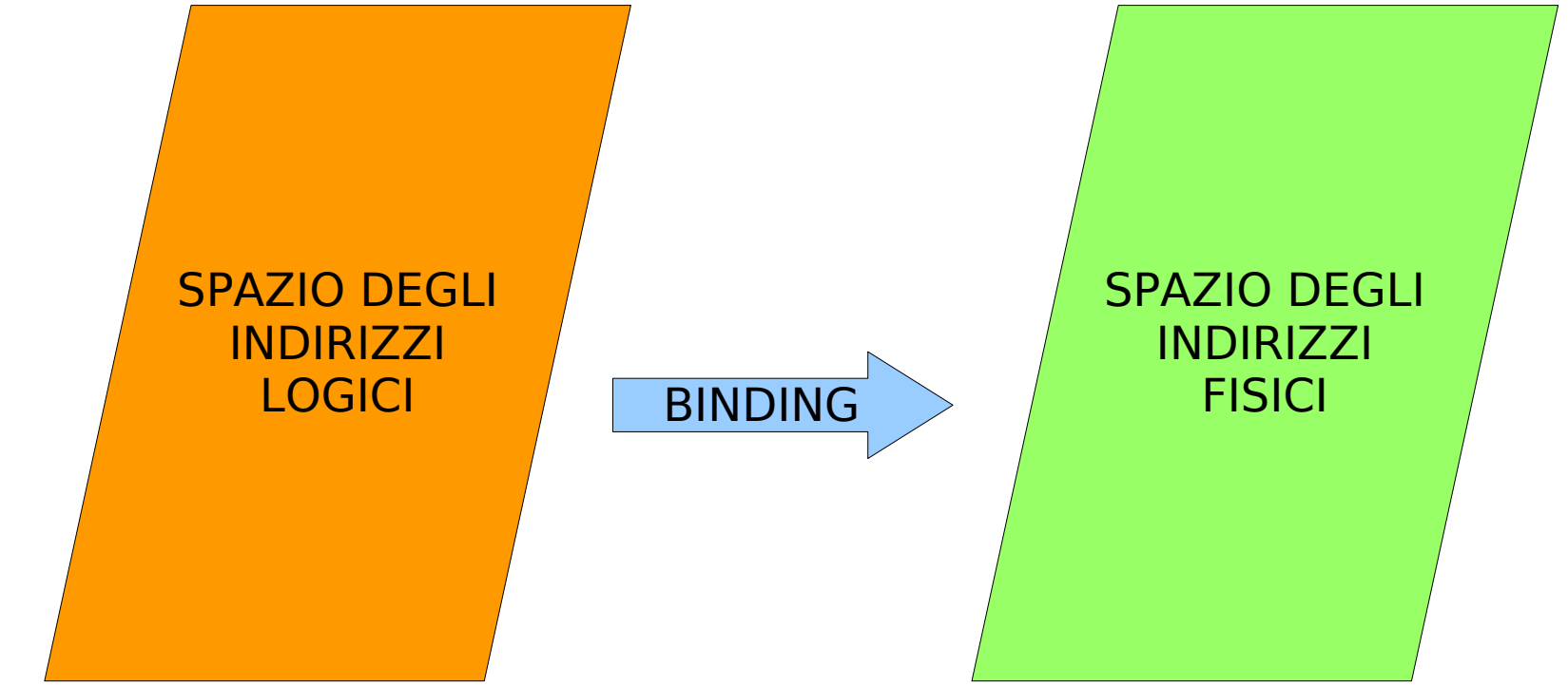
uso numeri


programma sorgente
compilazione
traduzione
indirizzi rappresentati come simboli
indirizzi rilocabili, es. base+10byte
modulo oggetto
eventuali altri moduli oggetto
può generare **indirizzi assoluti** (possibile quando si sa dove risiederà il programma)
linker
modulo rilocabile
Nota: se durante l'esecuzione un processo può essere spostato in memoria (es. nel passaggio da running a waiting e poi ancora a running) gli indirizzi del modulo eseguibile devono **rimanere rilocabili** fino all'esecuzione stessa
eventuali librerie di sistema
loader
indirizzi assoluti ? i.e. indici numerici di parole di memoria
modulo eseguibile
eventuali librerie caricate dinamicamente

Indirizzi logici e fisici

- un indirizzo prodotto dalla CPU è detto indirizzo logico
- d'altro canto ogni parola di memoria ha un proprio indirizzo fisico

Spazi degli indirizzi di un processo



Indirizzi prodotti dalla CPU

Indirizzi effettivi in RAM