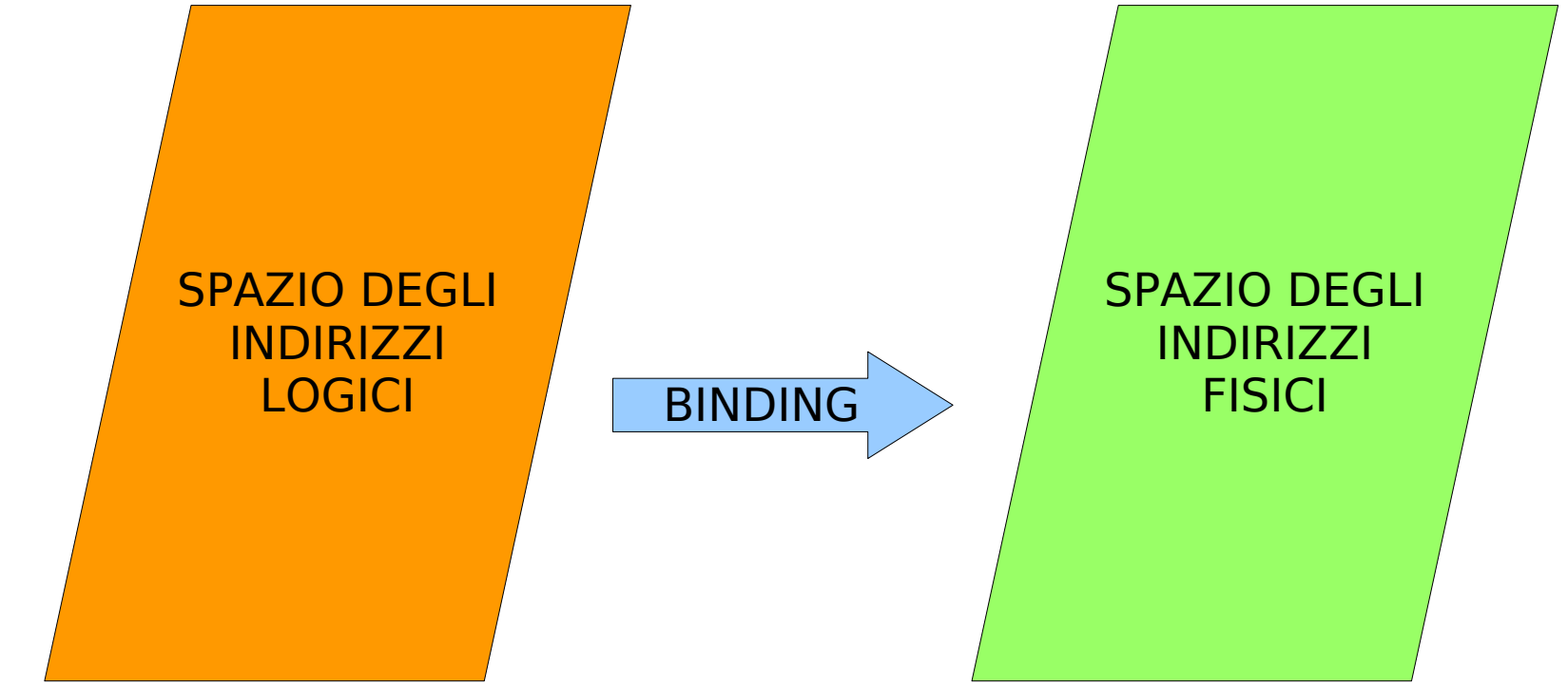
Spazi degli indirizzi di un processo

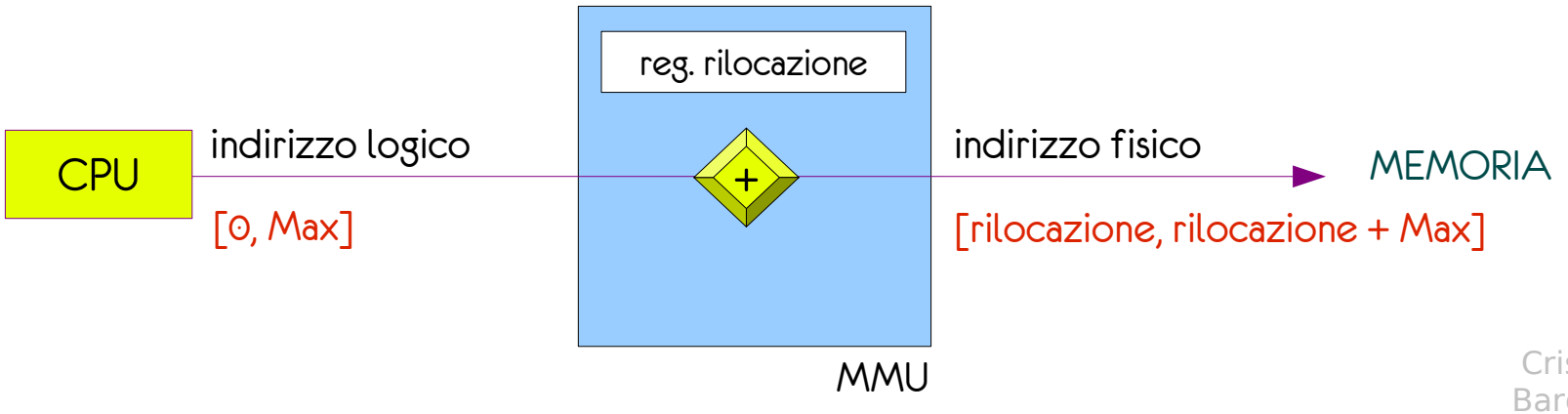


Indirizzi prodotti dalla CPU

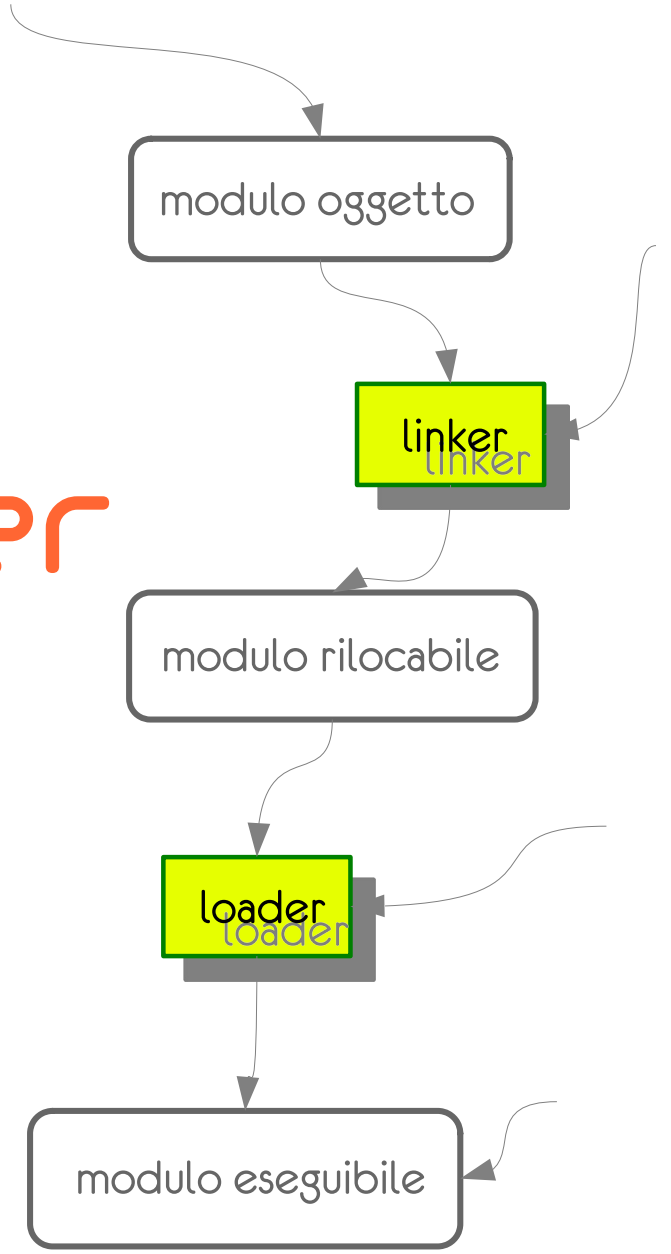
Indirizzi effettivi in RAM

Indirizzi logici e fisici: binding

- **Binding**: mapping dallo spazio degli indirizzi logici di un processo allo spazio dei suoi indirizzi fisici
- quando il binding viene fatto a tempo di compilazione o di caricamento:
  - indirizzo logico = fisico
- quando il binding viene fatto a tempo di esecuzione:
  - La corrispondenza deve essere calcolata: **lo spazio degli indirizzi logici ≠ dallo spazio degli indirizzi fisici**
- in questo caso il binding è a carico dell'MMU (memory management unit). L'MMU può essere realizzato in molti modi il più semplice è una generalizzazione del meccanismo basato sul registro base
- NB: la conversione è fatto SSE serve, cioè SSE si deve accedere alla memoria (lettura/scrittura)



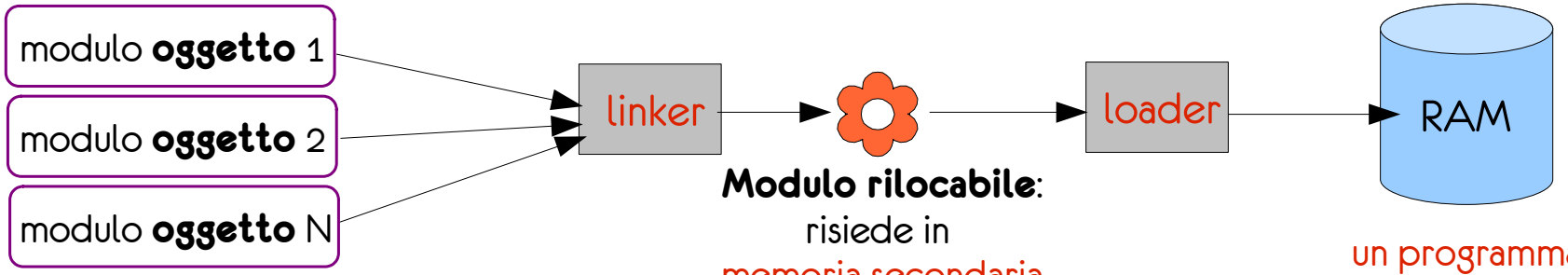
linker e loader



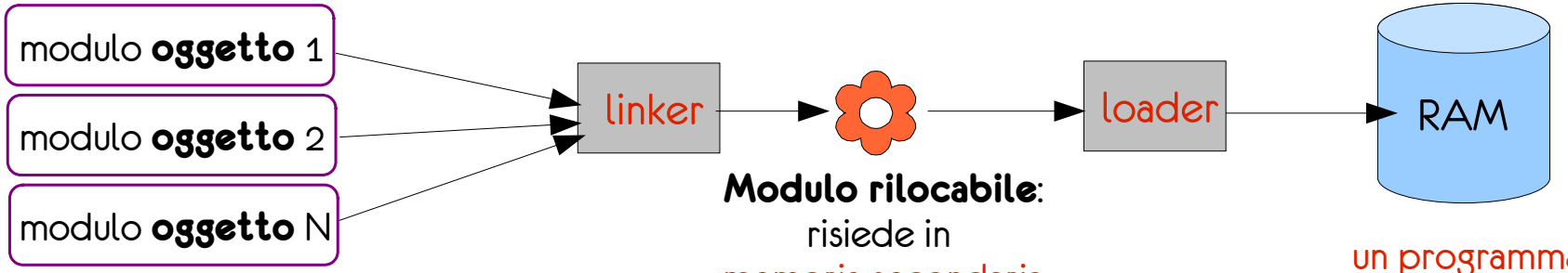
Cristina
Baroglio

Linking e loading

- linking: processo di composizione dei moduli che costituiscono un programma; associa ai nomi (di variabili o procedure), utilizzati da ciascun modulo e non definiti in esso, le corrette definizioni
- loading: copia un programma eseguibile (o parte di esso) nella RAM
- linking e loading sono statici quando precedono l'esecuzione
- linking e loading sono dinamici quando sono svolti durante esecuzione



Linking, loading e RAM

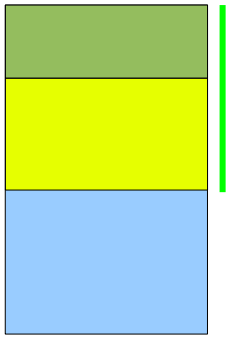


Modulo rilocabile:
risiede in
memoria secondaria

alcuni moduli corrispondono
a librerie di procedure

un programma
può essere ese-
guito solo se è
caricato in RAM

Approccio tradizionale



inserisco una copia del
codice di ogni libreria

eseguibile = composizione di
- file oggetto contenente il main
- file oggetti corrispondenti a librerie

L'intero file risultante è caricato in RAM:

1) una libreria può essere caricata **molte volte**, una copia per ogni programma che la include
2) carico anche le procedure **che non uso**

SPRECO !!!

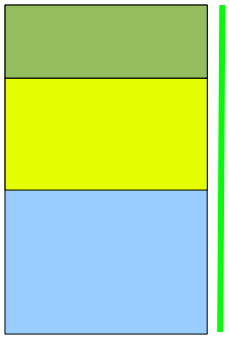
Linking e loading dinamici

- linking/loading: sono detti dinamici quando sono effettuati nella fase di esecuzione
- loading dinamico: una procedura è caricata in RAM quando occorre la sua prima invocazione (al suo primo utilizzo)
- linking dinamico: il collegamento del codice di una procedura al suo nome è effettuato alla sua prima invocazione. In questo caso il linker statico aggiunge solo uno *stub* della procedura in questione

Loading dinamico

- tutte le procedure risiedono in memoria secondaria sotto forma di codice rilocabile
- il codice di una procedura viene caricato nella RAM solo quando la procedura viene chiamata (per la prima volta)

- vantaggio rispetto a caricare un'intera libreria: si occupa meno RAM
- nota: occorre che il SO fornisca gli strumenti per realizzare librerie a caricamento dinamico



carico il codice di una procedura solo quando richiamata

In RAM solo una parte di programma:

- una procedura può essere caricata molte volte come parte di programmi diversi
- però carico solo le procedure che uso

eseguibile = composizione di
- file oggetto contenente il main
- rif. ai codici rilocabili

maggiore efficienza

Linking dinamico

- Rimanda il collegamento reale di una libreria alla fase di esecuzione
- dopo la compilazione, il linker statico arricchisce il “nucleo” del programma aggiungendo gli *stub* relativi alle procedure appartenenti alle librerie dinamiche usate
- **stub** = codice di riferimento, ha la seguente funzione
  - durante l'esecuzione, lo stub verifica se il codice della procedura è già stato caricato nella RAM:
    - se sì, sostituisce se stesso con l'indirizzo della procedura in questione
    - se no, causa il caricamento del codice della procedura e poi procede con la sostituzione come nel caso precedente
- **NB**: non importa se la procedura di libreria è stata caricata da un altro processo, tutti i processi fanno riferimento alla stessa copia del codice, la libreria risulta condivisa

Aggiornamento librerie condivise

- Il linking dinamico ha un notevole vantaggio:
- se aggiorno una libreria dinamica, automaticamente tutti i programmi che usano la libreria faranno riferimento alla nuova versione anche senza ricompilare (linking e compilazione sono spesso eseguiti entrambi dal compilatore)!
- se il linking fosse solo statico dovrei invece aggiornare il collegamento della libreria al programma, prima di utilizzare il medesimo

allocazione della ram

capitolo 8 del libro (VII ed.), da 8.3

Cristina
Baroglio

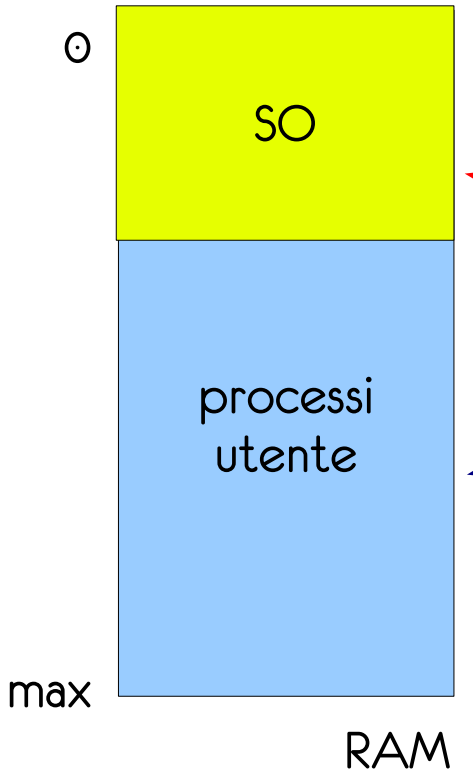
Sommario

- Lasciamo il Livello 0 per salire d'astrazione, passiamo al Livello 1
- Affronteremo ora il problema della gestione della memoria principale e, in particolare, della scelta dell'area da assegnare a un processo (quale memoria, quanta memoria, organizzata come)
- Tre approcci:
  1) Allocazione contigua
  2) Paginazione
  3) Segmentazione
- Ogni approccio fa riferimento a un modello di rappresentazione, che richiede apposite strutture dati e meccanismi di gestione

Allocazione contigua

- Allocazione contigua
  - rilocazione e protezione
  - partizioni multiple
  - frammentazione

Introduzione



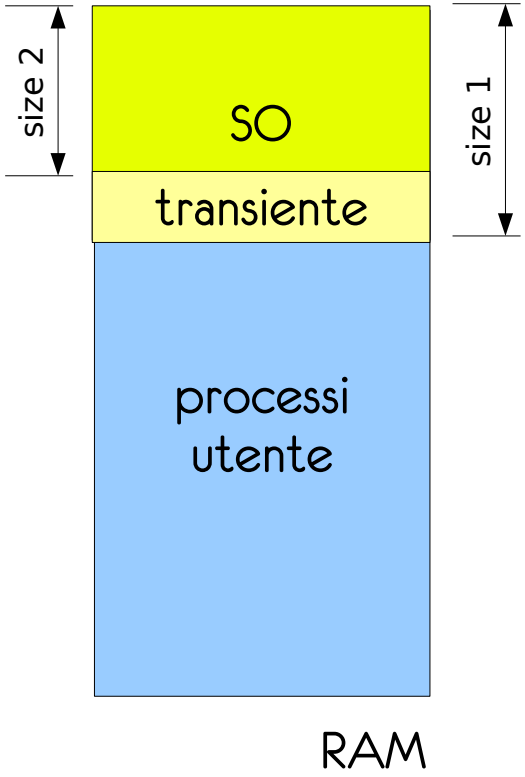
Nel modello ad allocazione contigua si suddivide la RAM in due parti:

- una riservata al SO, posizionata solitamente in memoria bassa (la posizione dipende dalla posizione del vettore delle interruzioni)
- l'altra riservata ai processi utente

occorre proteggere la partizione di memoria riservata al SO da letture/scritture ad opera di processi utente. Inoltre devo proteggere in modo analogo le aree di RAM riservate ai diversi processi utente

Ciò è facilmente realizzabile utilizzando un registro di rilocazione per la conversione di indirizzi logici in indirizzi fisici

Introduzione



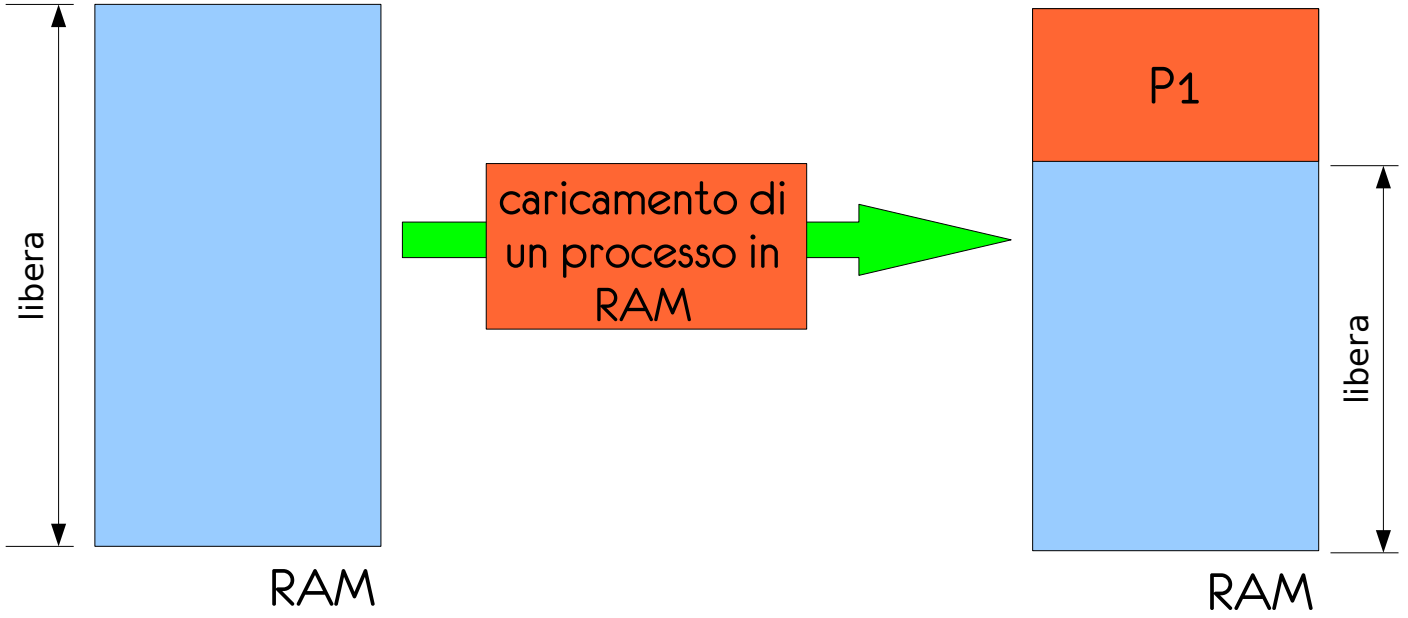
Il codice di SO caricato può a sua volta essere suddiviso in un nucleo di base sempre necessario e una parte che può essere utile o meno a seconda della circostanze (codice transiente).

Il codice transiente può essere rimosso dalla RAM quando non serve e aggiunto quando serve

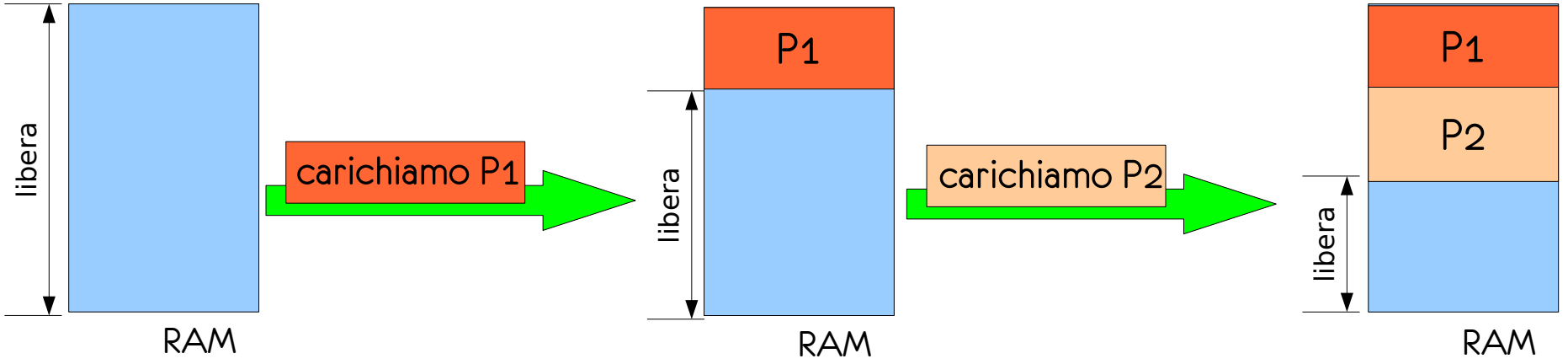
In queste circostanze occorre poter modificare la partizione riservata al SO

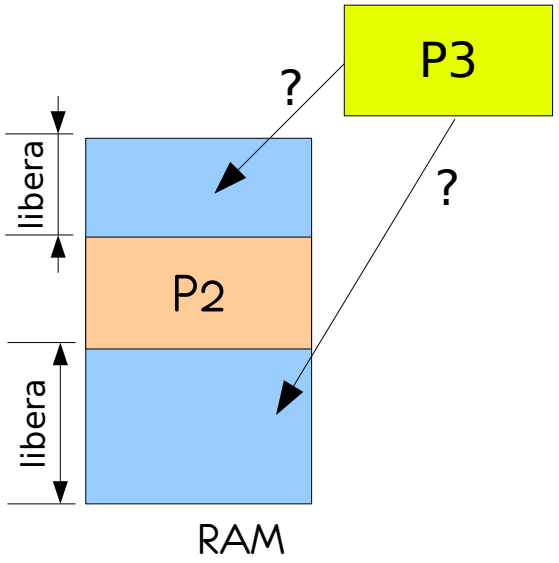
Allocazione a partizioni multiple

- Vediamo ora come funziona il meccanismo di allocazione della memoria ai processi nel modello ad allocazione contigua
- Lo schema seguito si chiama “a partizioni multiple”
- All'inizio tutta la RAM (esclusa la porzione per il SO) è libera:



Allocazione a partizioni multiple





P1 è terminato

In generale il SO deve mantenere una lista delle porzioni libere
Una porzione libera è detta "buco"

Un nuovo processo può essere caricato in RAM SSE esiste un buco abbastanza grande per contenere la sua immagine

In generale sarà possibile scegliere fra diverse opzioni ...

Criteri di scelta

- **Best-fit**: scelgo la porzione più piccola fra quelle adeguate a contenere l'immagine del processo
- **First-fit**: scelgo la prima porzione sufficientemente grande, trovata scandendo la lista dei buchi liberi
- **Worst-fit**: scelgo la porzione più grande fra quelle libere

Criteri di scelta

- **Qual'è la migliore?** Per capirlo occorre introdurre la nozione di frammentazione della memoria
- Di per sé per **frammentazione** si intende lo spezzettamento della memoria in tante parti. Si dice che si ha:
  - frammentazione esterna, se queste parti sono abbastanza grandi da essere utilizzabili (es. per contenere un processo)
  - frammentazione interna, se sono molto piccoli, praticamente inutilizzabili. In questo caso il frammento viene unito alla partizione precedente.

Criteri di scelta

- La frammentazione è un problema.
- L’analisi statistica mostra che con il first-fit, ogni N blocchi di memoria allocati si perde uno spazio pari a 0.5*N blocchi a causa della frammentazione (regola del 50%)
- In pratica 1/3 della memoria risulta inutilizzabile

RAM ALLOCATA RAM INUTILIZZABILE

Criteri di scelta

- In generale worst-fit è la strategia peggiore,
- First-fit e best-fit non sono sempre l'uno meglio dell'altro però computazionalmente first-fit è una tecnica meno costosa

Combattere la frammentazione

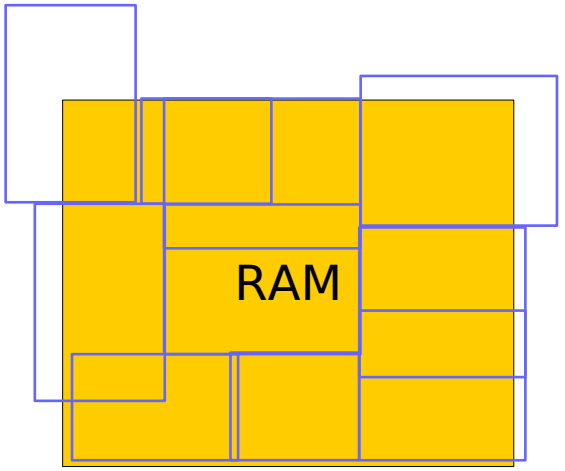
- È possibile combattere la frammentazione attuando di tanto in tanto una politica di compattamento: spostare le immagini dei processi in memoria dimodoché risultino contigue
- Il compattamento è applicabile solo se il binding fra indirizzi logici e fisici è effettuato a tempo di esecuzione

allocazione contigua, binding e rilocazione

Cristina
Baroglio

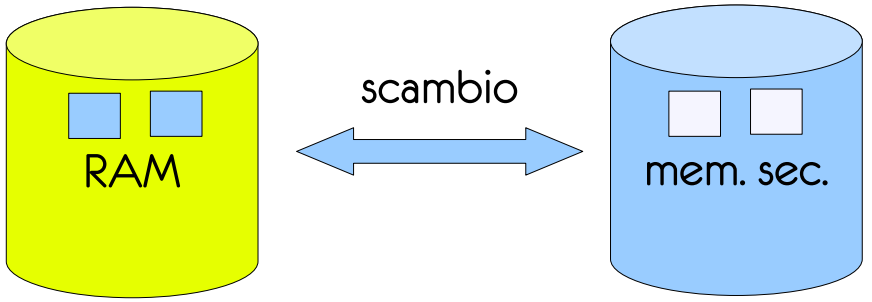
Swapping

- Scheduling di medio termine (già accennato)
- La RAM ha dimensione limitata
- Può succedere che i processi running e ready siano così tanti da richiedere complessivamente una quantità di memoria maggiore di quella offerta dalla RAM

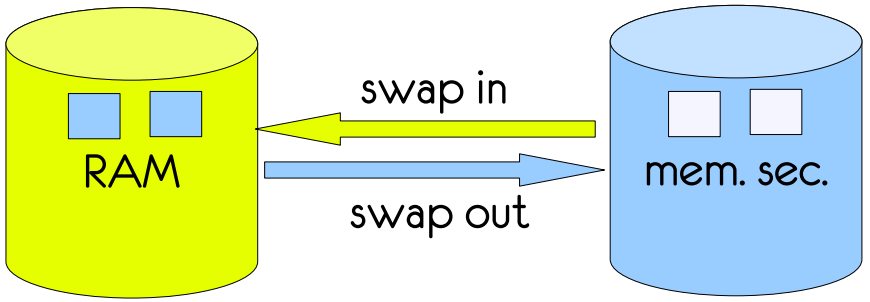


Swapping

- Scheduling di medio termine (già accennato)
- Soluzione: mantenere una parte dei processi ready in memoria secondaria ed effettuare di tanto in tanto lo swapping (lo scambio) fra processi in RAM e processi in memoria secondaria



Swapping



- swap in: carico l'immagine di un processo ready da memoria secondaria (anche detta backing store) in RAM
- swap out: scarico l'immagine di un processo che non è in esecuzione in memoria secondaria
- per motivi di efficienza è importante che i processi in testa alla ready queue siano conservati nella RAM, gli altri possono essere conservati in memoria secondaria

Swapping e binding

- L'immagine di un processo può fare swap-out e dopo un po' essere ricaricata in RAM (es. round-robin)
- In questo caso posso ricollocarla in una porzione qualsiasi della RAM?

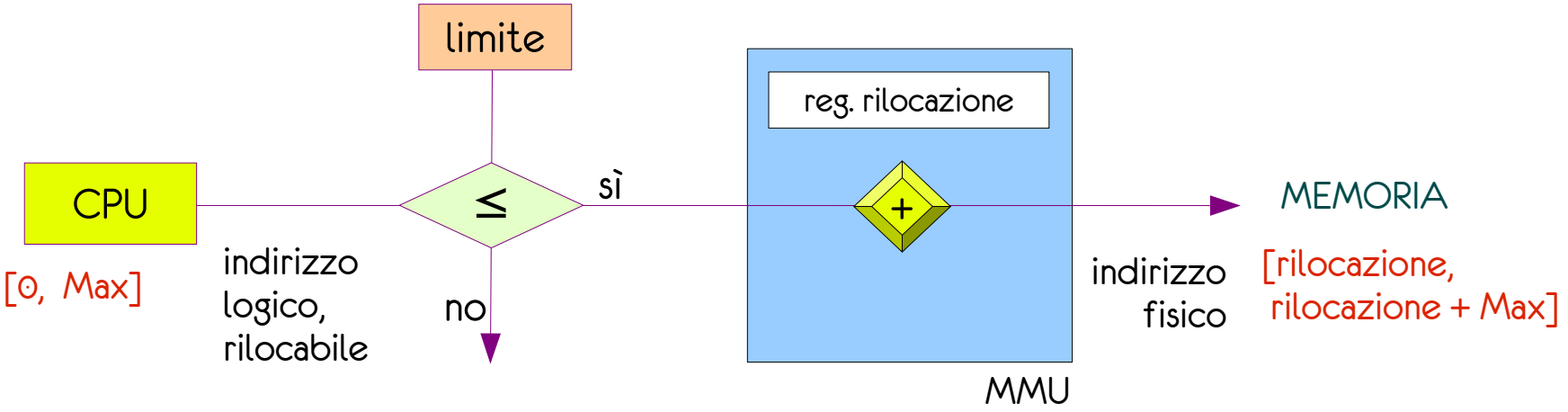
Swapping e binding

- La collocazione dipende dal **quando** viene effettuato il binding delle variabili:
  - se il **codice non è rilocabile** allora l'immagine del processo dovrà rioccupare la stessa sezione di RAM
  - se è **rilocabile** (in particolare, se il binding è dinamico) questo non è necessario

---

- Si può implementare la rilocazione se la RAM è gestita secondo il modello dell'allocazione contigua?
- Come avviene il binding?

Rilocazione e protezione



registro limite e registro di rilocazione vengono caricati durante il context switch
il contenuto del registro di rilocazione può variare nel tempo

L'uso dei registri limite e di rilocazione è possibile solo se l'HW dispone di queste strutture ⇒ la realizzazione dell'approccio a memoria contigua può essere realizzato solo previa presenza di un adeguato supporto HW

qualche dettaglio sullo swapping

Tempo di swapping

- Il tempo necessario al completamento dello swapping è dato dal tempo di swap-out + tempo di swap-in
- dipende dalla dimensione delle immagini dei processi coinvolti e dal tempo di trasferimento da/a memoria secondaria
- **Esempio**: se il tempo di trasferimento è pari a 1MB/sec, di quanto tempo ho bisogno per trasferire un processo con un'immagine da 100KB?

$$100\text{KB}/(1\text{MB}/sec) = (100\text{KB}/1000\,KB)\,sec = 0.1\,sec = 100\,msec$$

- di solito si usano i millisecondi come unità di misura
- ...

Tempo di swapping

- Supponendo identici tempo di swap-out e tempo di swap-in complessivamente occorreranno circa 200 msec
- il “circa” è dovuto al tempo necessario a posizionare la testina del disco

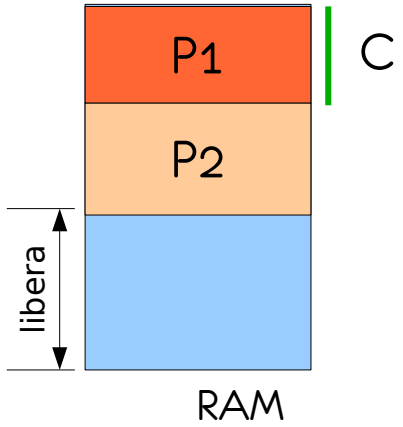
Commenti

- Un processo può essere oggetto di swapping SSE non ha in atto operazioni di I/O perché le operazioni di I/O non possono essere effettuate su variabili residenti in memoria secondaria
- **Esempi**
  - Unix (prime versioni): di base lo swapping era disabilitato, si attiva solo quando il carico del sistema è molto elevato
  - Windows 3.1: lo swapping era attivato solo a carico elevato ed era effettuato manualmente dall'utente

fine introduzione sulla gestione della RAM

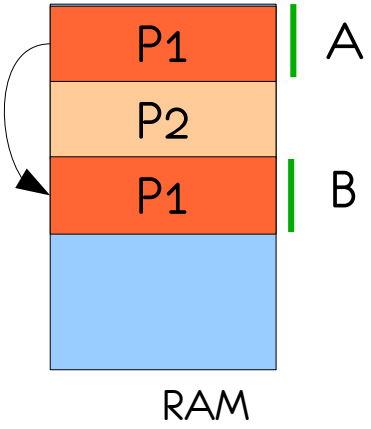
Paginazione della memoria

- La paginazione è un meccanismo di gestione della RAM alternativo all'allocazione contigua
- detto “**spazio degli indirizzi di un processo**” l'insieme di tutti gli indirizzi a cui il processo ha accesso, carattestica fondamentale della paginazione è che essa consente allo spazio degli indirizzi fisici di un processo di non essere contiguo



allocazione contigua

sp. indirizzi di P1 = C



paginazione

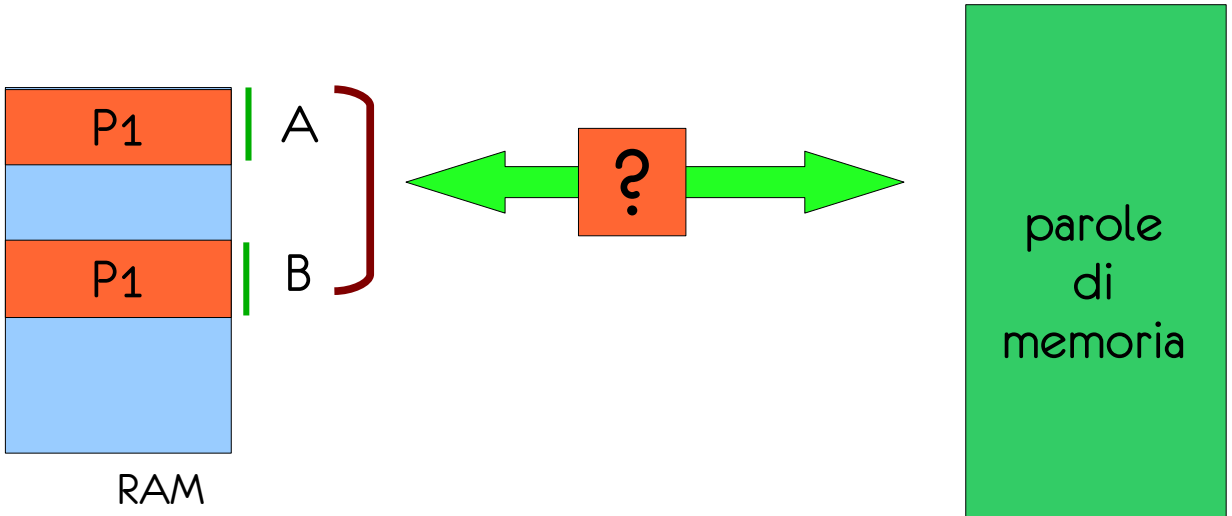
sp. indirizzi di P1 = A ∪ B

È UN VANTAGGIO?

Paginazione

- È un vantaggio
- Consente di far (de)crescere in modo dinamico lo spazio riservato a un processo, semplicemente (togliendo) aggiungendo delle pagine
- Quindi per esempio posso mantenere in RAM solo una porzione del codice di un processo, aggiungendo via via altre parti utili, con riferimento all'esecuzione corrente
- La gestione del codice transiente del SO diventa molto più semplice e naturale
- Paginazione e architettura di una macchina sono strettamente correlate: la paginazione è possibile solo avendo un opportuno supporto hardware
- Vediamo ora le strutture necessarie per realizzare questo modello ...

Pagine e strutture di supporto



paginazione

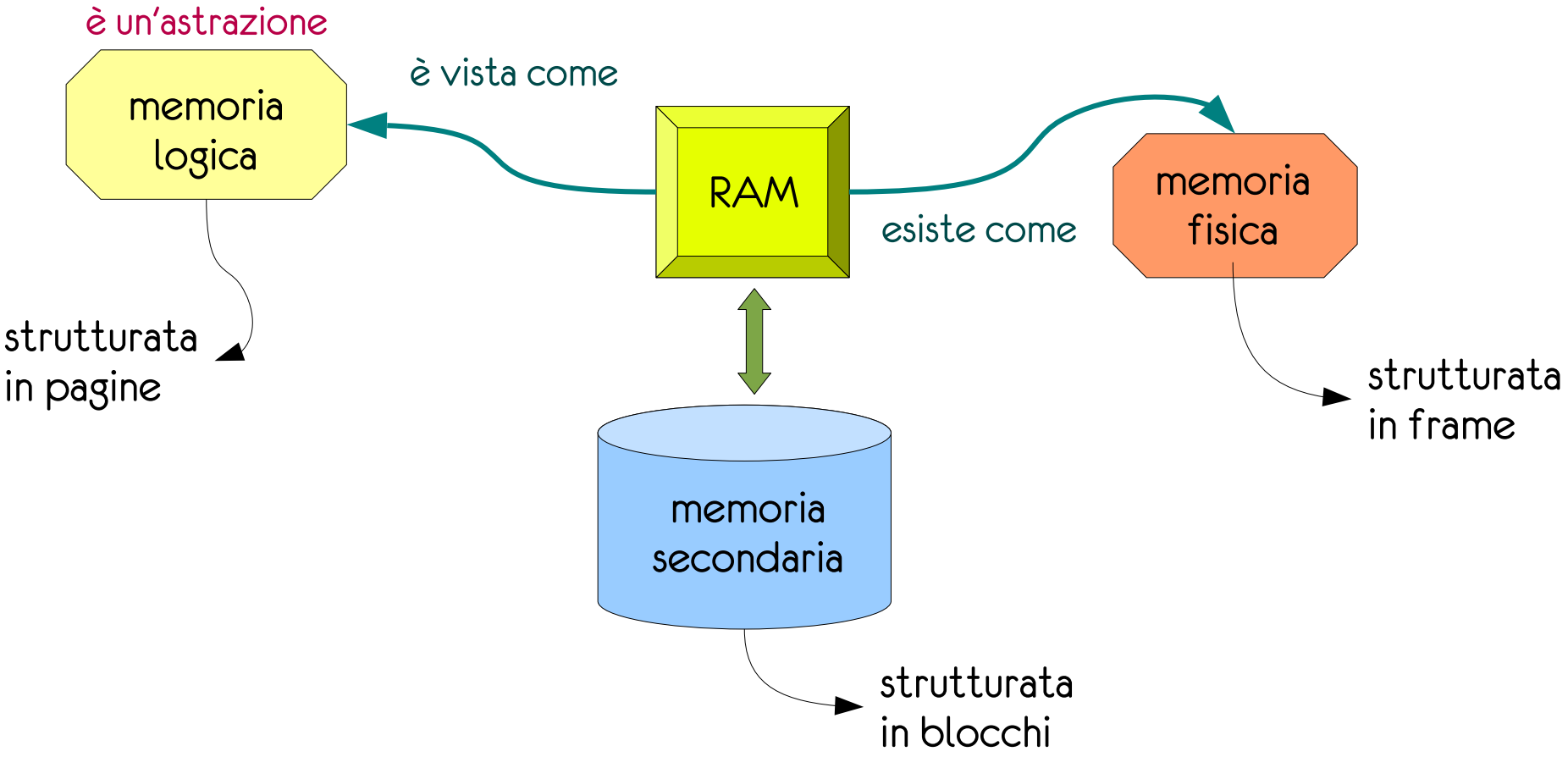
sp. indirizzi di P1 = A ∪ B

(1) Come posso associare porzioni di processo a porzioni di RAM?

(2) Come posso organizzare le diverse porzioni in un tutt'uno?

Vedere la RAM come array non è più così comodo ...

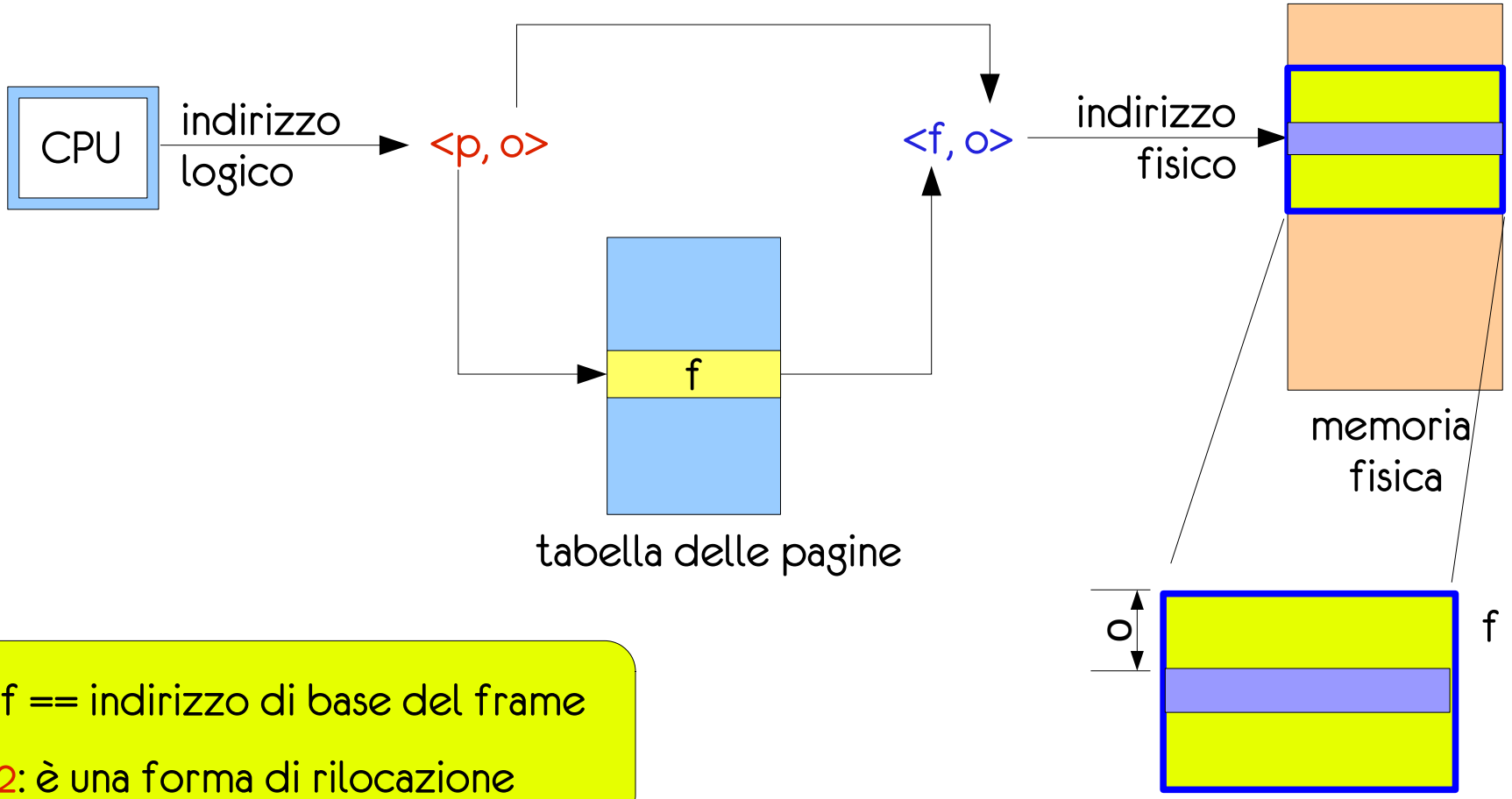
Pagine, frame e blocchi



blocchi, frame e pagine sono tutti termini che fanno riferimento a porzioni di memoria di uguali dimensioni ma appartenenti a viste/elementi diversi

Pagine e frame

Il primo tipo di struttura di cui abbiamo bisogno serve a fare il binding fra indirizzi logici e indirizzi fisici. In questo contesto un indirizzo logico è una coppia <numero_di_pagina, offset>, un indirizzo fisico è una coppia data a <id_frame, offset>



Nota: f == indirizzo di base del frame

Nota 2: è una forma di rilocazione

Indirizzi logici

- La dimensione è la stessa per tutte le pagine ed è definita dall'architettura; è una potenza di 2 normalmente compresa fra 512 byte e 16 MB
- Supponiamo che la dimensione di una pagina sia 2^n e la dimensione della memoria logica sia 2^m, in quante pagine sarà suddivisa la memoria logica?

$$2^m / 2^n = 2^{m-n}$$

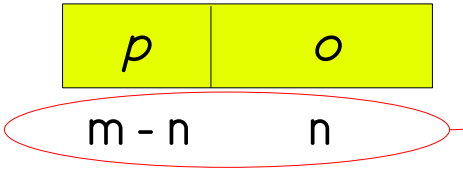
- quanti bit servono per rappresentare un numero di pagina?

$$m - n$$

- quanti bit occorrono quindi per rappresentare lo scostamento all'interno di una pagina?

$$n$$

- quindi un indirizzo logico:



Nota: è giusto che la somma faccia m?

Indirizzi logici: esempio

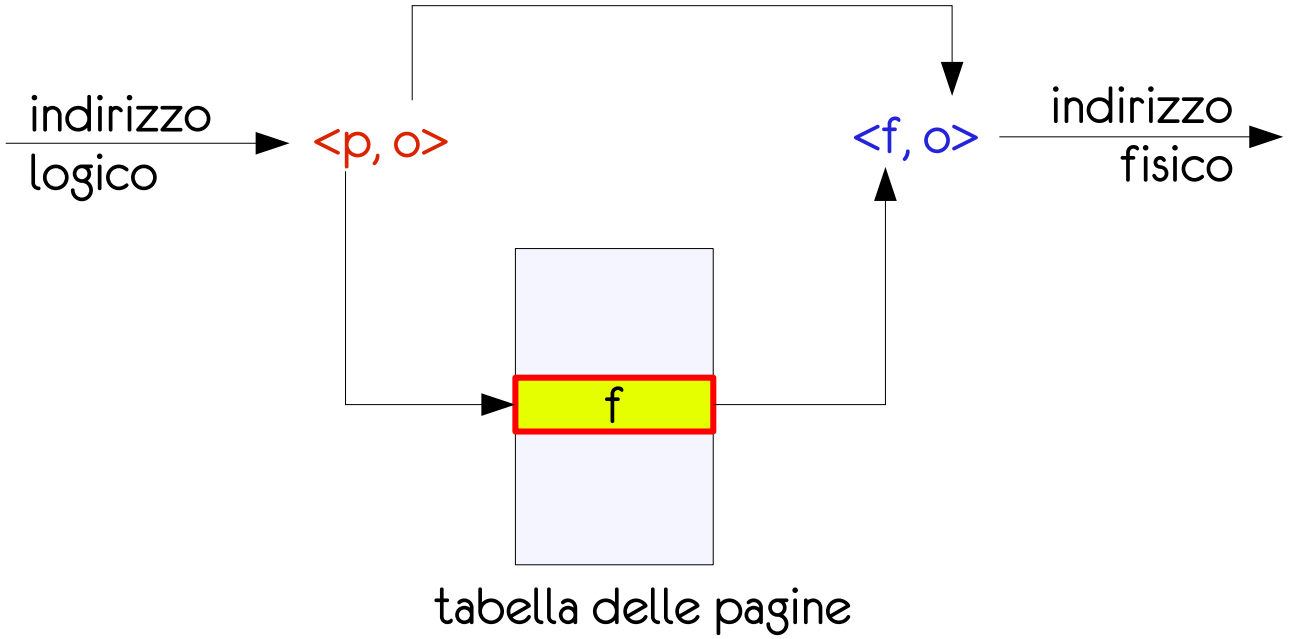
- Supponiamo di avere pagine di dimensione 512 byte e una RAM di dimensione 1MB, quante pagine avremo? Quanti bit occorrono per rappresentare indirizzi logici in questo contesto?
- Innanzi tutto devo riportarmi a potenze di 2 nella stessa unità di misura:
  - pagina: 512 byte = 2^9 byte
  - memoria logica: 1MB = 2^{20} byte
- Ora possiamo fare i conti:
  - numero di pagine necessarie: $2^{20} / 2^9 = 2^{20-9} = 2^{11}$
  - per rappresentare il numero di pagina occorrono 11 bit
  - per rappresentare l'offset occorrono): 9 bit
- num bit per le pagine + num bit x l'offset = 11 + 9 = 20

Paginazione e rilocazione

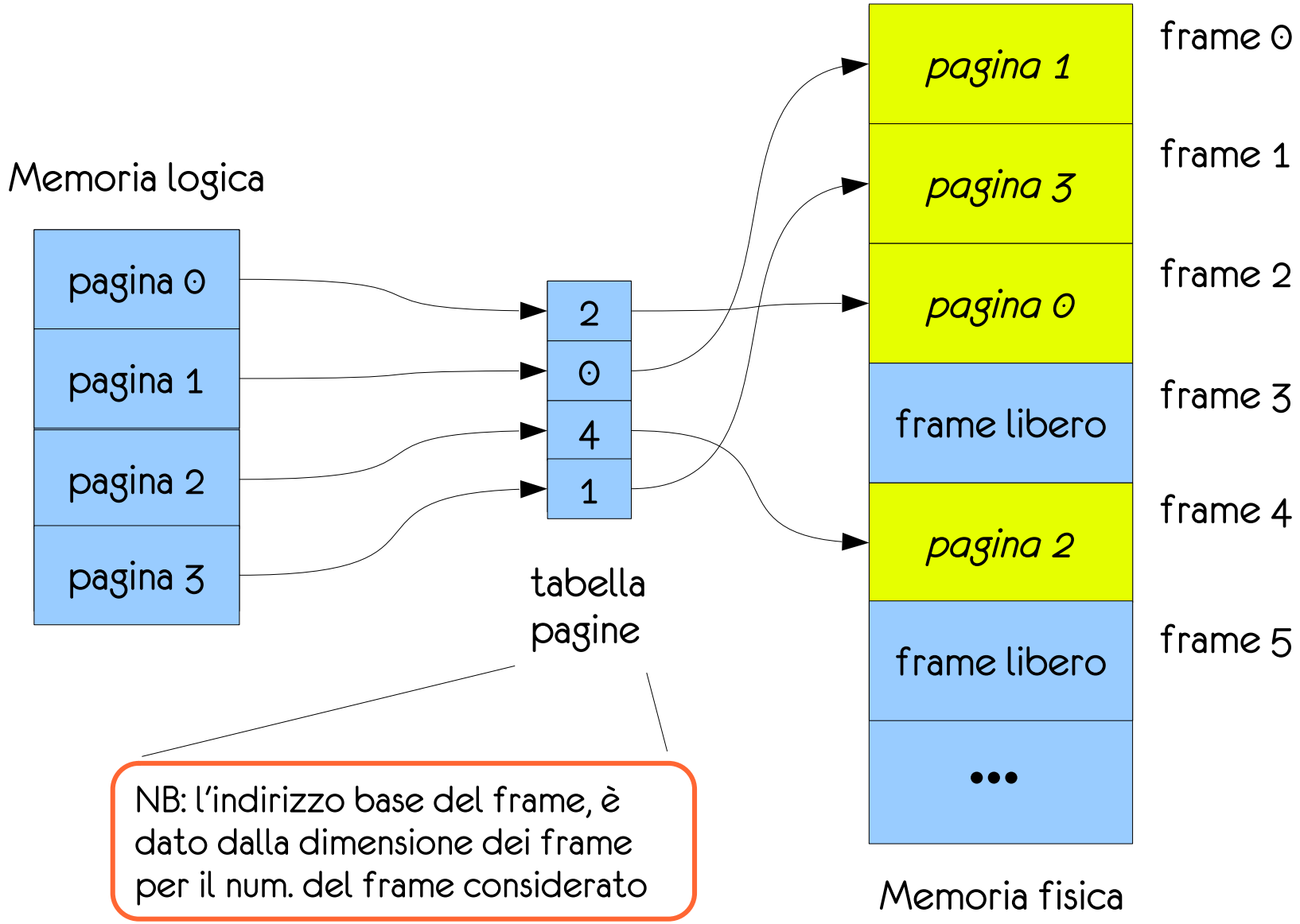
- Cosa vuol dire “rilocare il codice” in questo contesto?

consentire l'accesso a una pagina, indipendentemente dal frame in cui è caricata

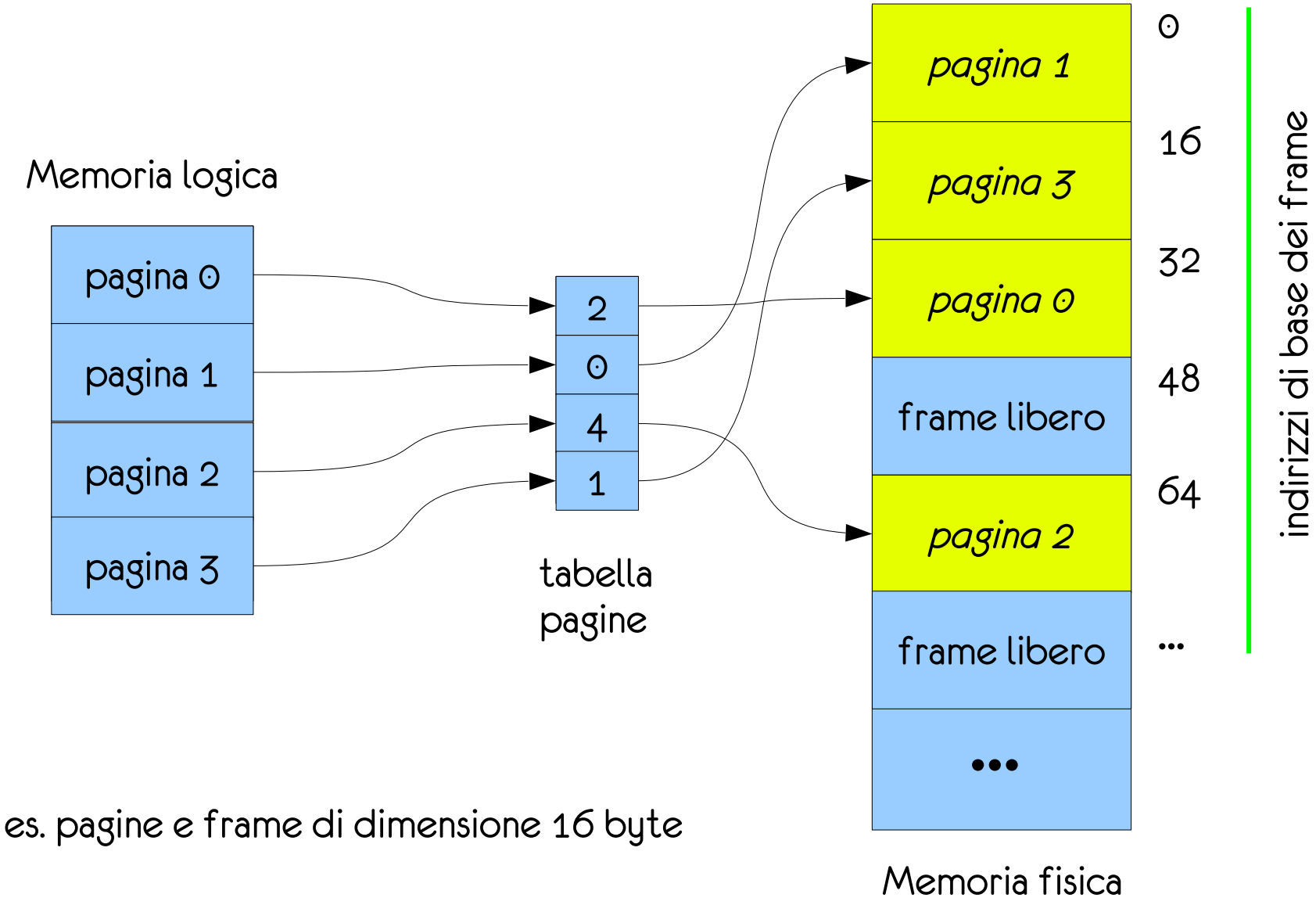
- Si può realizzare la rilocazione? Sì, il registro di rilocazione è sostituito dalla entry nella tabella delle pagine, corrispondente a p. Il valore f individua l'indirizzo di inizio del frame



Esempio di paginazione

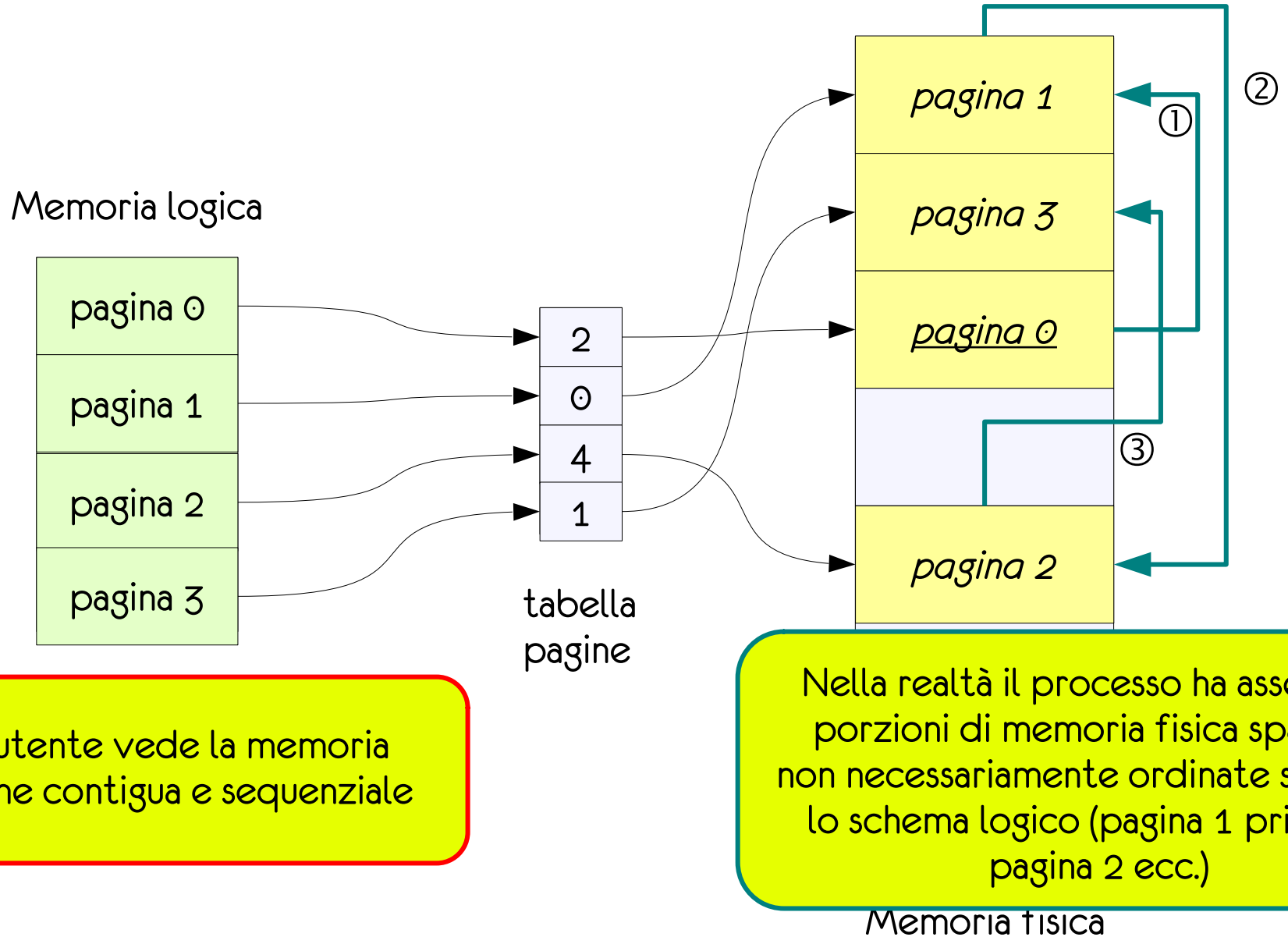


Esempio di paginazione



es. pagine e frame di dimensione 16 byte

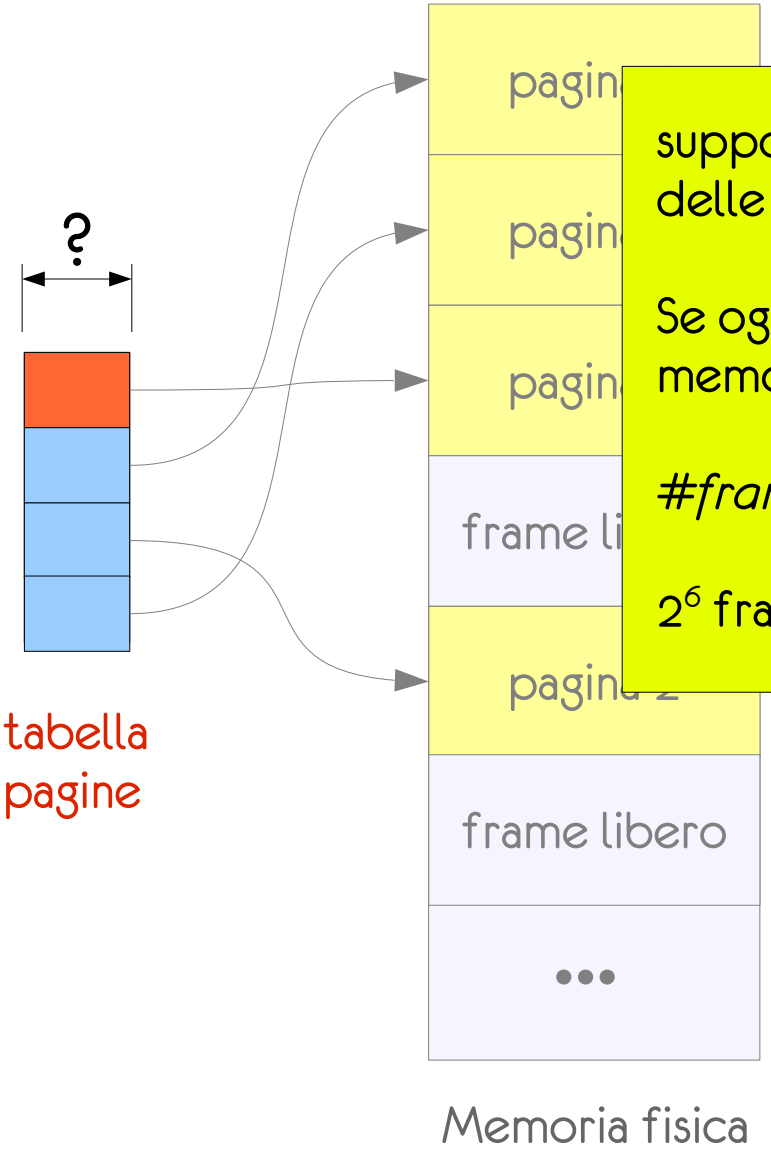
Esempio di paginazione



L'utente vede la memoria come contigua e sequenziale

Nella realtà il processo ha assegnate porzioni di memoria fisica sparse e non necessariamente ordinate secondo lo schema logico (pagina 1 prima di pagina 2 ecc.)

Qualche conto



supponiamo che un elemento della tabella delle pagine sia grande 4 byte

Se ogni frame ha dimensione 4 KB, a quanta memoria fisica riusciamo ad accedere?

*#frame indirizzabili * dimensione di un frame*

2^6 frame * 4 KB = 2^2 * 2^6 KB = 2^8 KB = 256 KB

Paginazione e frammentazione

- La paginazione elimina il problema della frammentazione esterna però permane il problema della frammentazione interna
- Ogni processo può avere allocate un numero di pagine diverso, quante di queste presenteranno frammentazione interna?
- Soltanto l'ultima perché raramente la dimensione di un processo sarà un multiplo della dimensione di una pagina, quindi in generale avrò bisogno di *N pagine più un pezzetto* per ciascun processo
-

Paginazione e frammentazione

- **Frammentazione interna**: in media possiamo dire che avremo ½ pagina non utilizzata per ogni processo
- Dimensione ottimale delle pagine: occorre trovare un compromesso fra limitare il problema della frammentazione e ridurre i costi dell'I/O:
  - **pagine piccole**: riducono la frammentazione
  - **pagine grandi**: migliori quando occorre trasferire da/a memoria secondaria grosse quantità di dati (carico una pagina invece di tante in sequenza)