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?

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

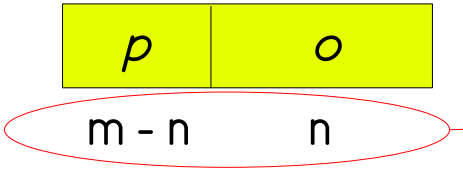
- quanti bit servono per rappresentare un numero di pagina?

$$m - n$$

- 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

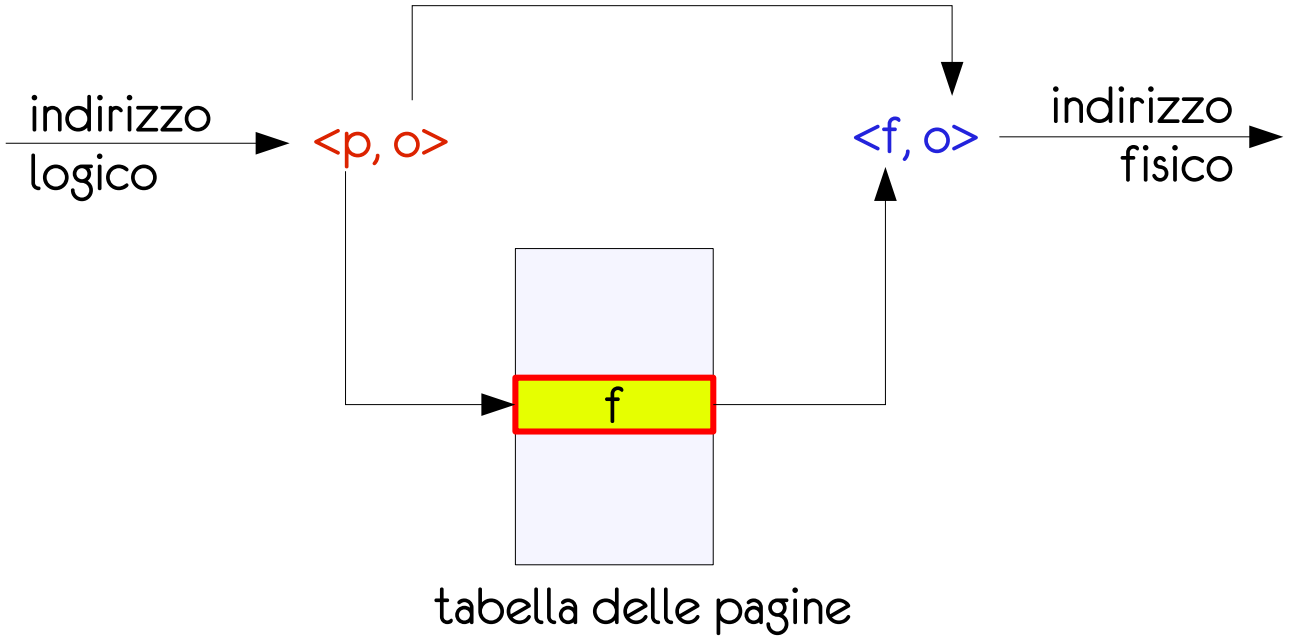
- Supponiamo di avere pagine di dimensione 512 byte e una RAM di dimensione 1MB, quante pagine avremo? Quanti bit occorrono per rappresentare indirizzi logici in questo contesto?
- Innanzi tutto devo riportarmi a potenze di 2 nella stessa unità di misura:
  - pagina: 512 byte = 2^9 byte
  - memoria logica: 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

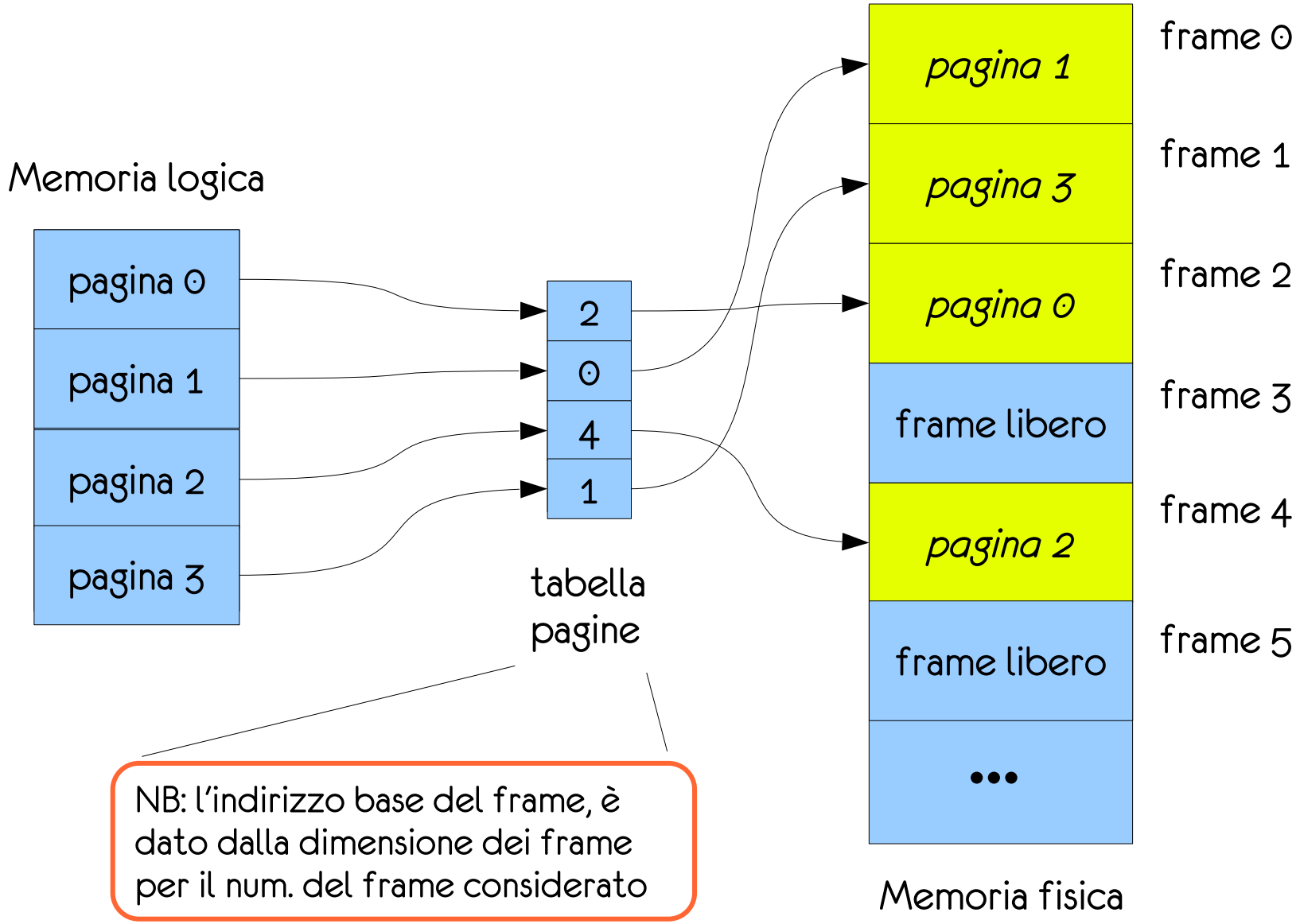
- Cosa vuol dire “rilocare il codice” in questo contesto?

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

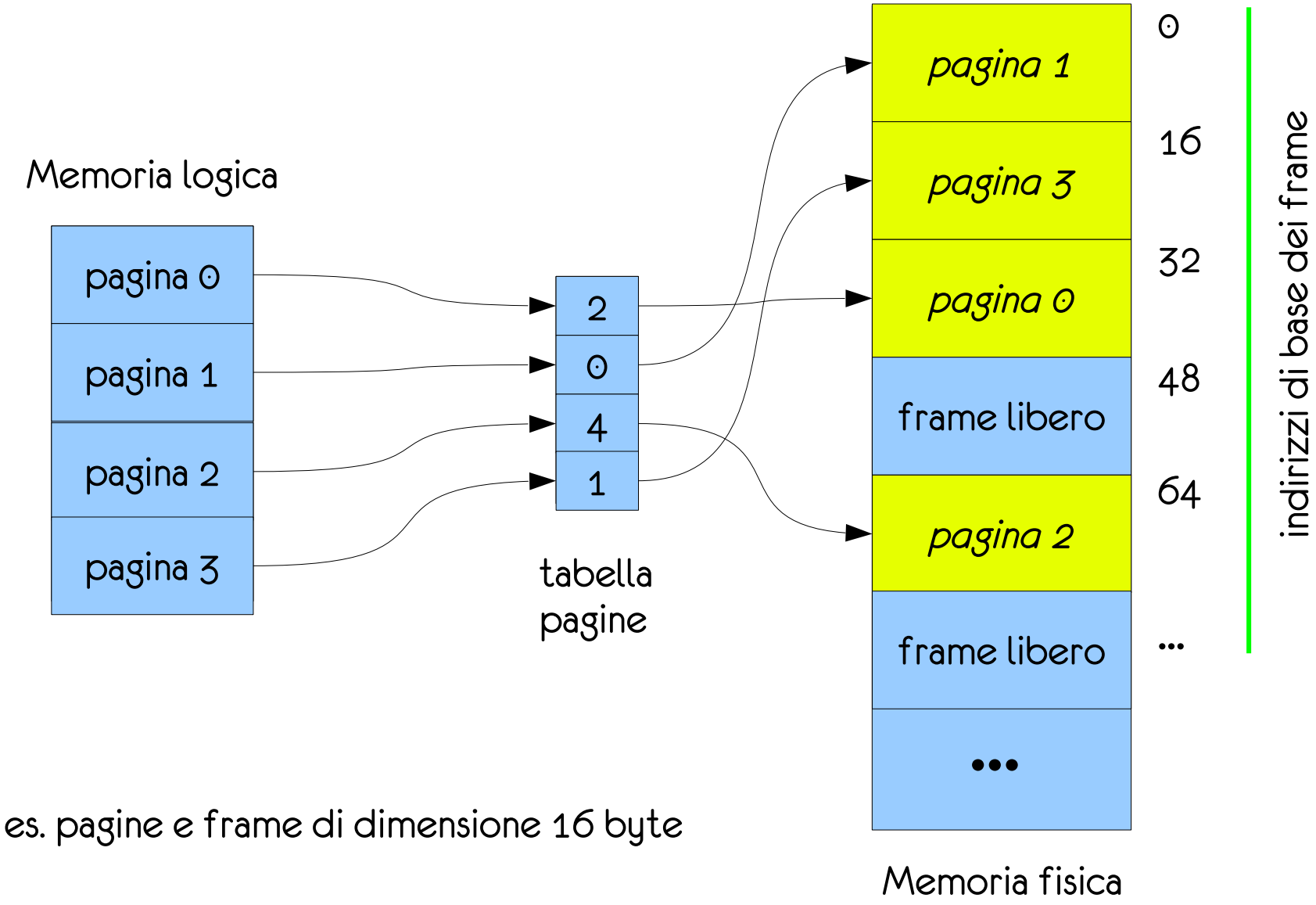
- Si può realizzare la rilocazione? Sì, il registro di rilocazione è sostituito dalla entry nella tabella delle pagine, corrispondente a p. Il valore f individua l'indirizzo di inizio del frame



Esempio di paginazione

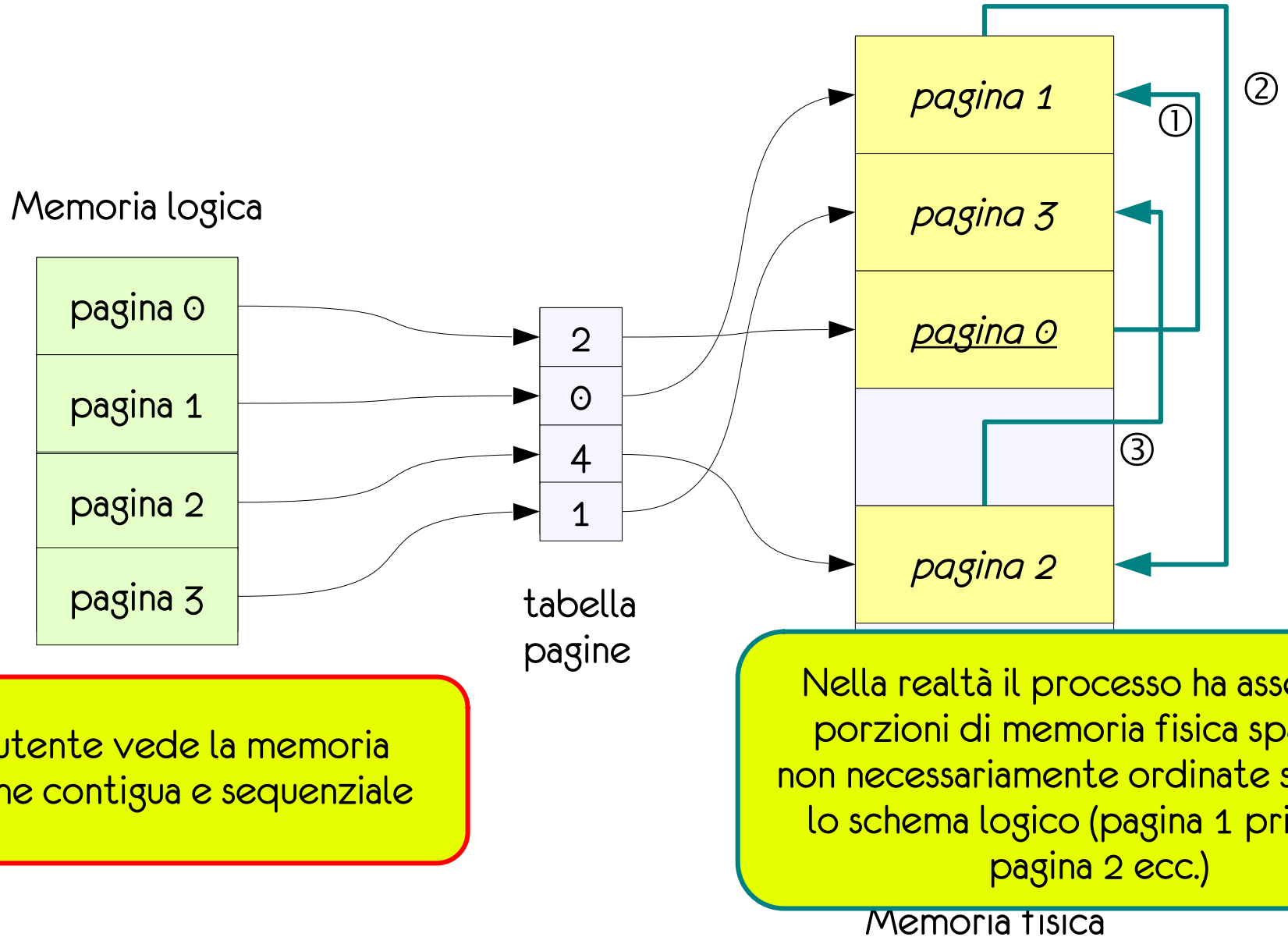


Esempio di paginazione



es. pagine e frame di dimensione 16 byte

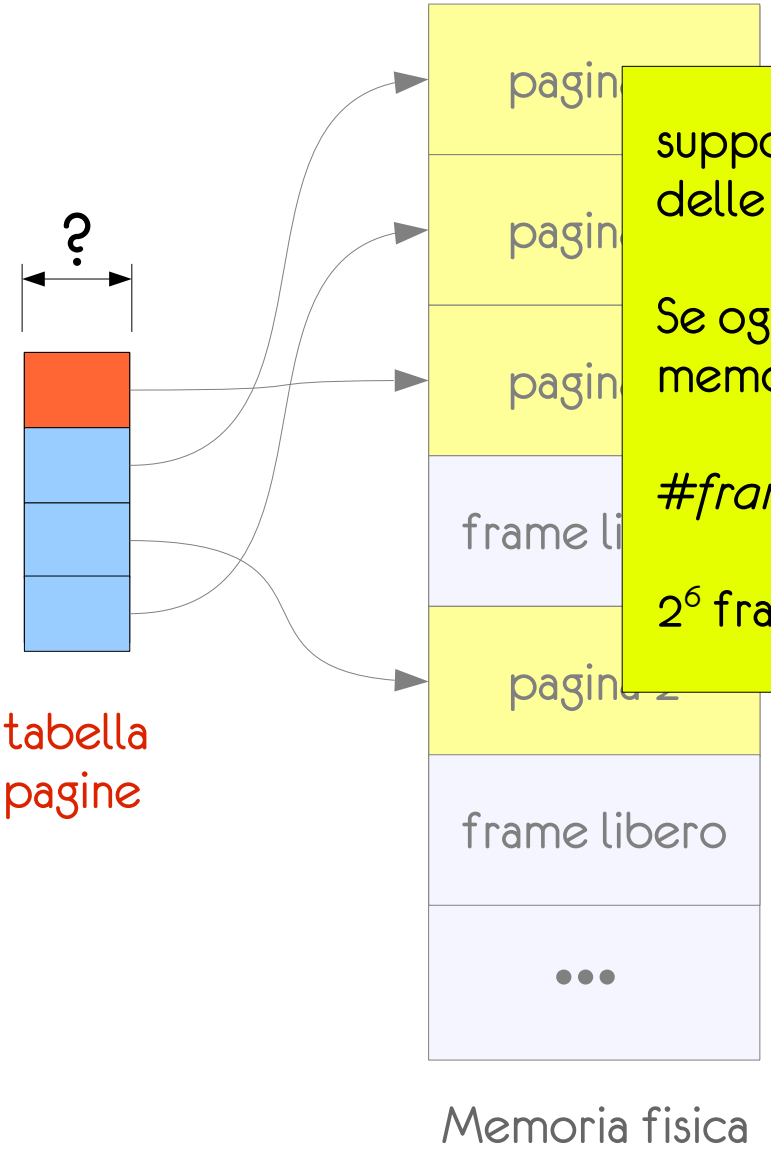
Esempio di paginazione



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

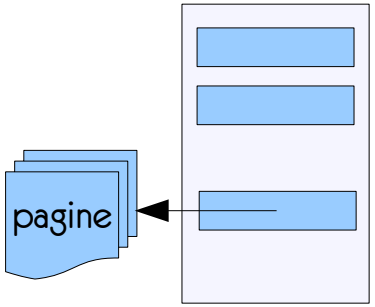
- **Frammentazione interna**: in media possiamo dire che avremo ½ pagina non utilizzata per ogni processo
- Dimensione ottimale delle pagine: occorre trovare un compromesso fra limitare il problema della frammentazione e ridurre i costi dell'I/O:
  - **pagine piccole**: riducono la frammentazione
  -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

Quante tabelle delle pagine?

- ogni processo in RAM ha la propria tabella delle pagine ∈ PCB
- inoltre il SO mantiene (in copia unica) numero e lista dei frame liberi (tabella dei frame). Quando si carica un processo:

```
if (#frame liberi ≥ #pagine del processo) {
     foreach (pagina da caricare) {
        <<crea una entry nella tab. delle pag. del processo>>
        <<nuova entry = num. di un frame libero>>
        <<aggiorna tabella dei frame>>
        <<carica pagina>>
     }
 }
 else ???
      /* per ora diciamo che il processo non verrà caricato */
```

Architettura di paginazione



Il PCB di un processo contiene un riferimento alle pagine del processo stesso (codice e dati su memoria secondaria)

Quando la CPU è assegnata a un processo il dispatcher carica le sue pagine nella RAM impostando i valori della tabella delle pagine del processo e della tabella dei frame

Com'è implementata la tabella delle pagine?

tramite un insieme di registri

PRO: i registri sono ad accesso veloce

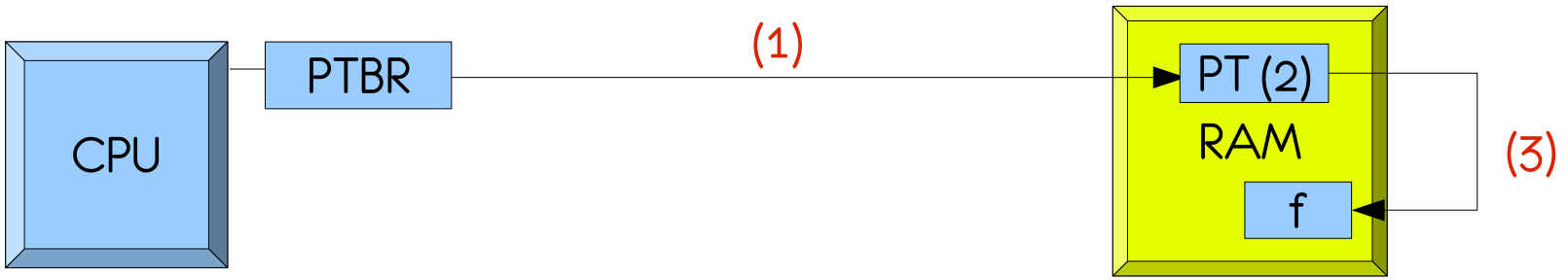
CONTRO: troppo pochi, ne serve uno per ogni pagina del processo

Accettabile se #pagine ≦ 256

viene memorizzata nella RAM si usa un solo registro (PTBR, page table base register) che contiene l'indirizzo di base della tabella

il dispatcher dovrà modificare solo il valore di questo registro

Problema: tempi di accesso



(1) tramite PTBR accedo alla RAM per individuare la page table

(2) tramite la page table costruisco l'indirizzo fisico a cui accede

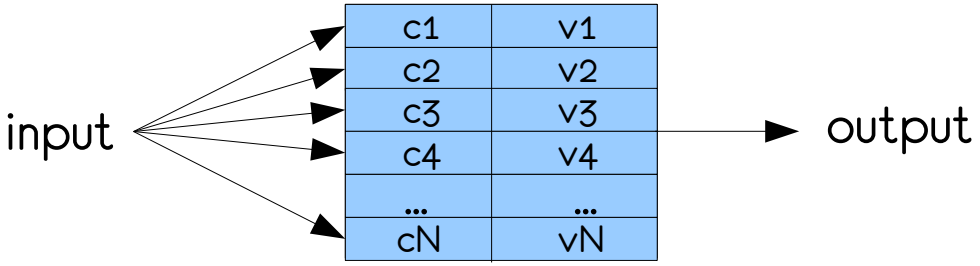
(3) accedo all'indirizzo fisico di interesse

Il numero di accessi alla RAM è raddoppiato perché ogni volta devo prima accedere alla tabella delle pagine (PT) e poi all'indirizzo che mi interessa

Il raddoppio degli accessi è inaccettabile, rallenta troppo l'esecuzione

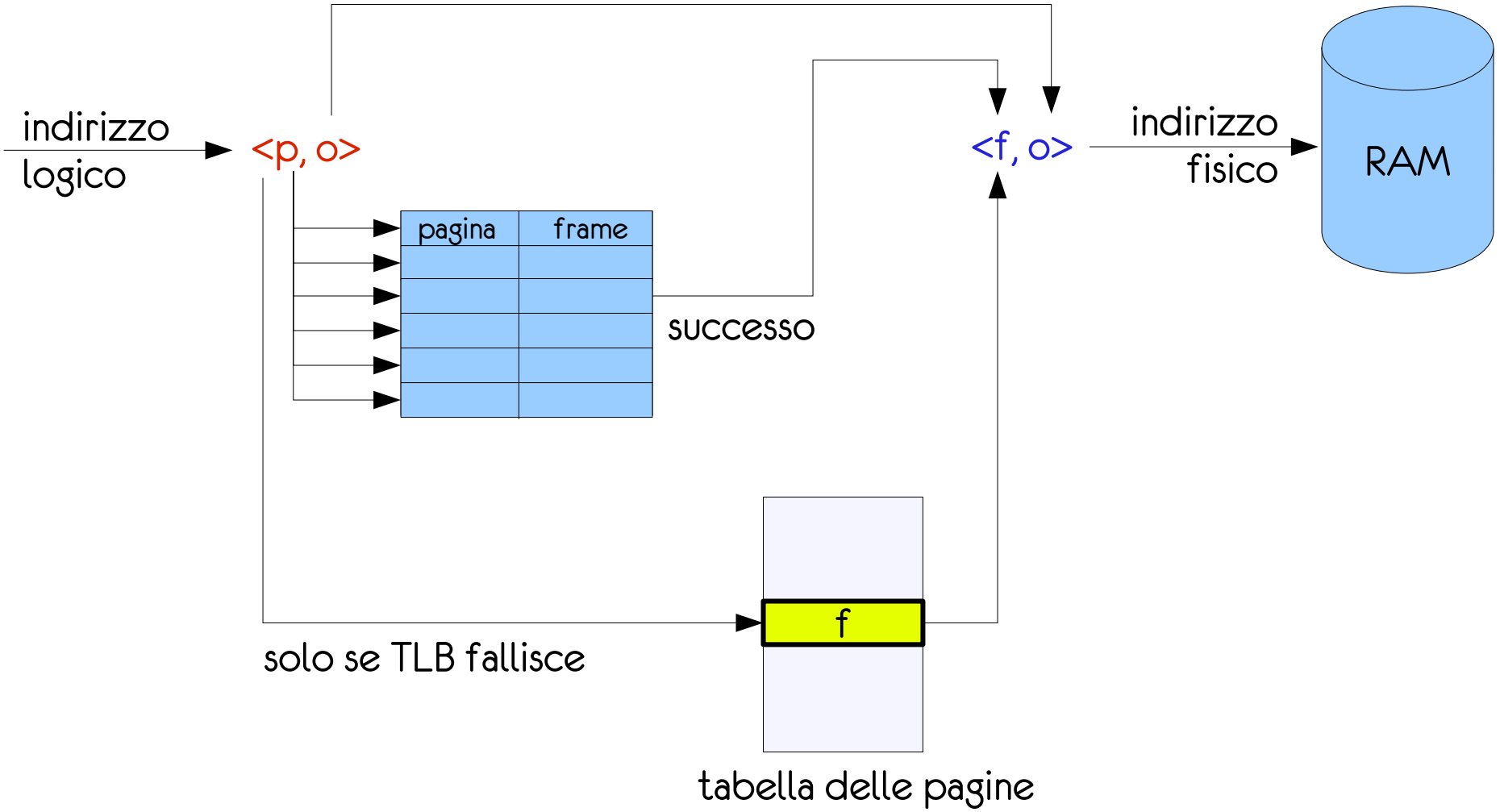
Soluzione: Translation look-aside buffer

TLB (translation look-aside buffer): è una cache molto veloce contenente delle coppie <chiave, valore>. Quando riceve un input lo confronta contemporaneamente con tutte le chiavi. Se trova una chiave corrispondente restituisce il valore associato.



Molte architetture adottano un TLB in associazione alla tabella delle pagine per limitare il problema del doppio accesso alla RAM, chiavi = numeri di pagina, valori = # frame

Schema d'uso


indirizzo
logico
<p, o>
pagina
frame
successo
<f, o>
indirizzo
fisico
RAM
f
solo se TLB fallisce
tabella delle pagine

Hit ratio

- Col termine **hit ratio** si intende la percentuale di successo relativo al reperimento di una pagina tramite TLB (la pagina è accessibile tramite il TLB)
- Dire che l'hit ratio è pari al 92% significa che in 92 casi su 100 il frame è stato individuato attraverso il TLB e solo nei restanti 8 si è dovuto accedere alla tabella delle pagine
- L'hit ratio consente di calcolare il tempo medio effettivo di accesso a una pagina

Hit ratio

Esempio

- Supponiamo che il tempo di accesso al TLB sia di 20 nsec mentre l'accesso alla RAM richiede 100 nsec
- Proviamo a calcolare il tempo medio effettivo necessario per accedere a una pagina nel caso in cui l'hit ratio è pari all'82% e nel caso in cui è pari al 95%

Hit ratio

- **Ipotesi:**
  - tempo di accesso al TLB = 20 nsec
  - tempo di accesso alla RAM = 100 nsec
  - hit ratio = 82% = 0.82
    (oppure 95%=0.95)
  - percentuale di accessi tramite page table = 18% = 0.18 (oppure 5%=0.05)

Hit ratio

- **Calcolo**
  - tempo di accesso a una pagina tramite TLB:
    - T_{tlb} = accesso a TLB + accesso RAM = 20 + 100 nsec = 120 nsec
  - tempo di accesso a una pagina tramite tabella delle pagina:
    - T_{pt} = 2 x accesso RAM = 100 + 100 nsec = 200 nsec
  - 82%: t.di a. medio = (0,82 * T_{tlb}) + (0,18 * T_{pt}) = 0,82 x 120 + 0,18 x 200 = 134,4 nsec
  - 95%: t.di a. medio = (0,95 * T_{tlb}) + (0,05 * T_{pt}) = 0,95 x 120 + 0,05 x 200 = 124 nsec

Aggiornamento del TLB

- Quando si inserisce una nuova coppia nel TLB?
- Quando non trovo una pagina di interesse (*TLB miss*):
  - se c'è spazio nel TLB si inserisce la nuova coppia <pagina, frame>
  - se non c'è spazio, si sostituisce una coppia già presente con quella nuova, di solito viene scelta per la sostituzione la coppia usata meno di recente
- l'idea di fondo è che quando accedo ad una nuova pagina, verosimilmente l'avrò bisogno per un po' di tempo
- NB: alcune delle pagine usate come chiavi nel TLB corrispondono ai processi principali del SO, le loro entry sono considerate non sovrascrivibili -> ho pagine di processi diversi!!!

Context switch

- Il TLB contiene dati (quasi) esclusivamente concernenti il processo running
- Quando si effettua un context switch occorre aggiornare il contenuto del TLB? Vorrei attuare un meccanismo di protezione che impedisca a un processo di accedere alle pagine di un altro

Context switch

- Due possibili soluzioni:
  - Alcuni TLB arricchiscono l'informazione contenuta in essi, aggiungendo un ASID (identificatore univoco dello spazio degli indirizzi di un processo). Un ASID identifica in modo univoco un processo. Uso l'informazione relativa a una pagina nel TLB, solo se l'ASID del processo running corrisponde all'ASID della pagina
  - Soluzione alternativa: il dispatcher svuota il TLB ad ogni context switch

Protezione

- Un aspetto ulteriore concerne la **modalità di accesso** alle pagine, memorizzata nella tabella delle pagine del processo
- Una pagina può essere accessibile in lettura, lettura-scrittura, esecuzione, oppure essere invalida (non appartiene allo spazio degli indirizzi del processo).
- Si possono avere pagine non valide quando la tabella delle pagine di un processo è più piccola dello spazio riservato ad essa nella RAM
- Se un processo cerca di accedere a una pagina non valida, viene generato un interrupt
- La modalità di accesso viene aggiunta alla tabella delle pagine

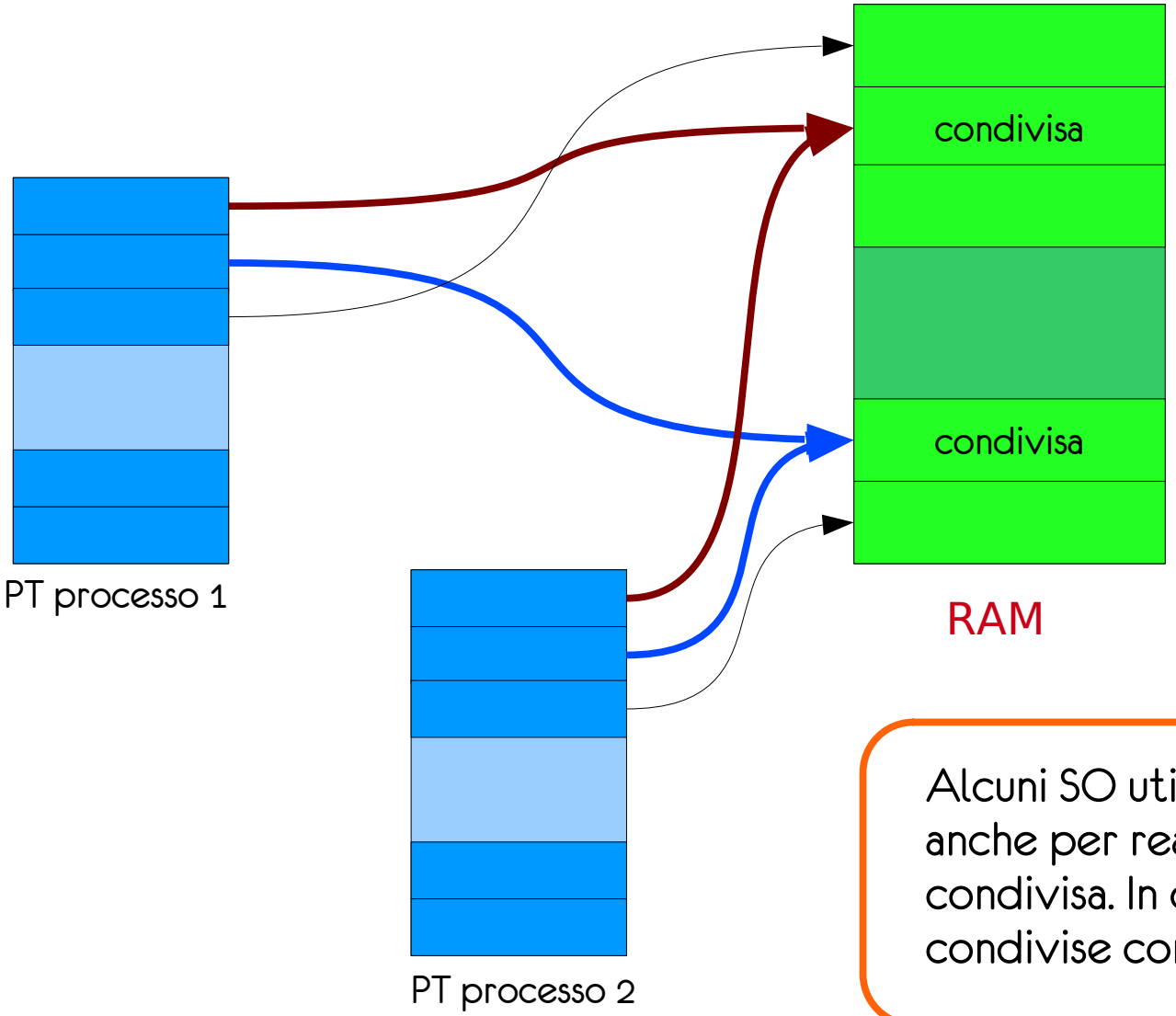
Condivisione delle pagine

- Spesso nei sistemi in time-sharing diversi utenti usano lo stesso programma contemporaneamente
- In un contesto di multi-programmazione in senso lato, succede spesso che più processi utilizzino la stessa libreria contemporaneamente

Condivisione delle pagine

- Supponiamo che un programma di questo tipo (es. compilatore) occupi 200KB, che 4 utenti lo stiano usando, che i loro processi abbiano 1 pagina di dati e che le pagine siano grandi 50KB: complessivamente saranno occupate 20 pagine di RAM: 1000KB
- se fosse possibile far condividere il codice sarebbero invece sufficienti 400KB di RAM (50 x 4 per i dati + 200 per il codice in copia unica)
- NB: solo le pagine di codice possono essere condivise, pagine accessibili in sola lettura. Si parla di **codice rientrante**

Condivisione delle pagine



Alcuni SO utilizzano questo sistema anche per realizzare la memoria condivisa. In questo caso le pagine condivise contengono dati

Struttura della tabella delle pagine

capitolo 8 del libro (VII ed.), da 8.5

Cristina
Baroglio

Elenco

- paginazione multi-livello:
  - struttura gerarchica ad albero
- hash table:
  - spesso usata per architetture oltre i 32 bit
- tabella delle pagine invertita:
  - approccio diverso in cui si ha una sola tabella delle pagine globale anziché una per ciascun processo

Paginazione multi-livello

- La dimensione della tabella delle pagine dipende dalla **dimensione dello spazio degli indirizzi logici**
- Consideriamo:
  - indirizzi a 32 bit (permettono di fare riferimento a 2^{32} parole di memoria),
  - con pagine grandi 4KB (cioè 2^{12})
- in quante pagine verrà suddiviso lo spazio degli indirizzi logici?

Paginazione multi-livello

- In quante pagine verrà suddiviso lo spazio degli indirizzi logici?

  $2^{32} / 2^{12} = 2^{20}$

- cioè la tabella delle pagine potrà avere fino a 2^{20} entry (precisamente 1,048,576)

Paginazione multi-livello

- In quante pagine verrà suddiviso lo spazio degli indirizzi logici?

  $2^{32} / 2^{12} = 2^{20}$

- cioè la tabella delle pagine potrà avere fino a 2^{20} entry (precisamente 1,048,576)
- se ogni entry occupa 4byte, una tabella delle pagine potrà occuperà **4MB** !!!

Paginazione multi-livello

- **Osservazione**:

  nella realtà nessun processo usa l'intero spazio degli indirizzi, ne usa molto meno

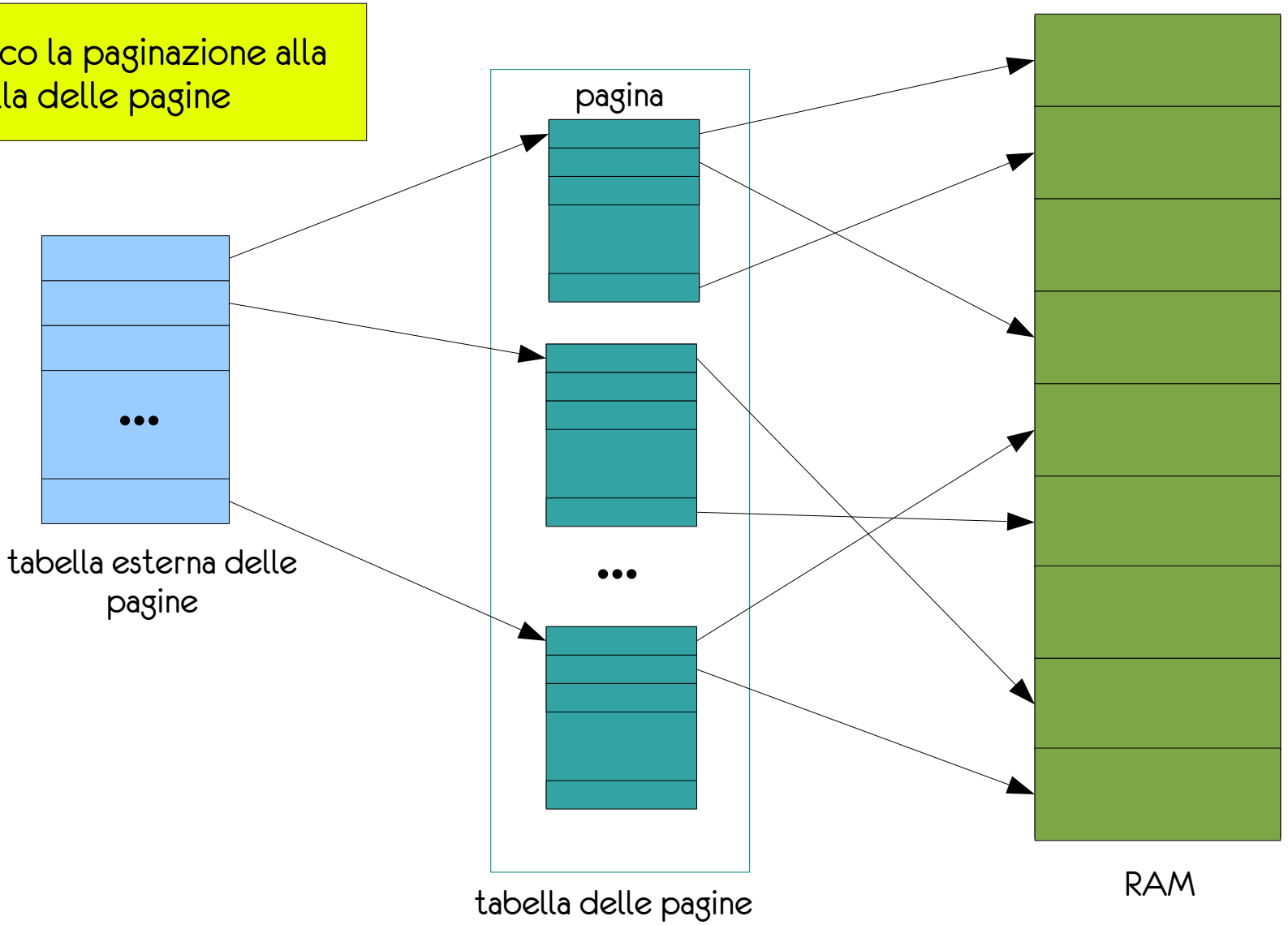
  la maggior parte della tabella sarebbe inutilizzata, tuttavia i processi possono avere tabelle delle pagine molto pesanti

Paginazione multi-livello

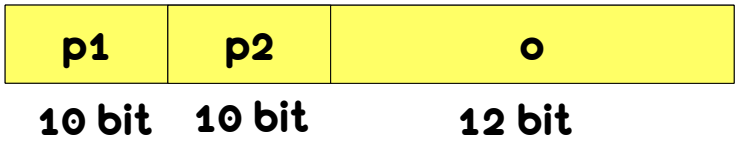
- **Conclusione**: l'allocazione della tabella delle pagine non può essere contigua ma va suddivisa in tante parti organizzate in una struttura
- **Nella paginazione multi-livello si usa un albero**

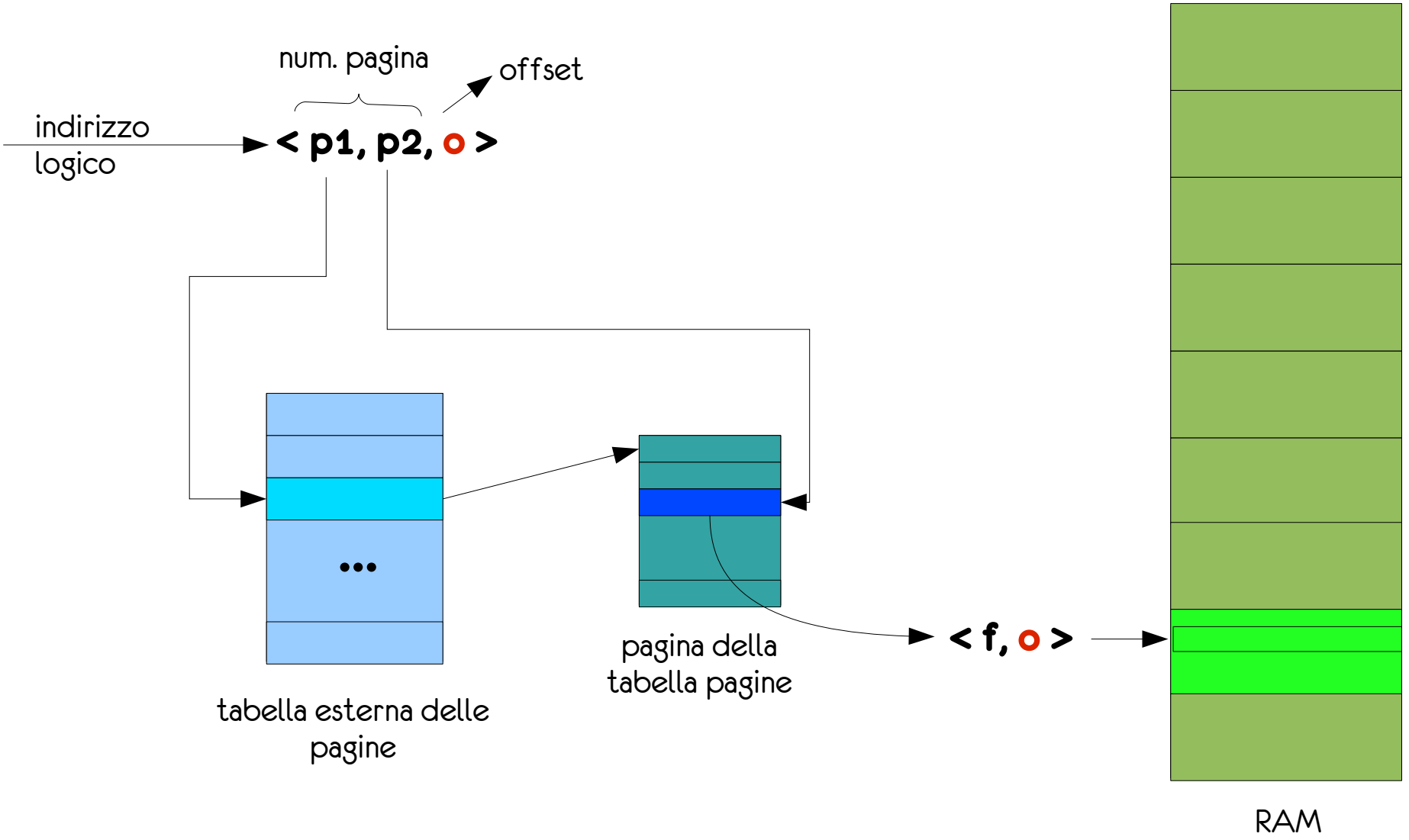
Paginazione a due livelli

applico la paginazione alla tabella delle pagine



Indirizzi logici





Esempi e commenti

- Architettura SPARC (di Sun):
  paginazione a 3 livelli
- Motorola 68030 a 32 bit:
  paginazione a 4 livelli

Esempi e commenti

- La paginazione multilivello non è appropriate per architetture a 64 bit: la tabella esterna risulterebbe troppo grande
- Si dovrebbero aggiungere troppi livelli ulteriori di paginazione (es. UltraSPARC ne avrebbe richiesti 7!) e ciò renderebbe troppo costoso l'accesso alla RAM qualora si verificassa un TLB miss
- Diverse architetture a 64 bit ad alte prestazioni usano tabelle delle pagine inverse (vedremo fra poco) unitamente a tabelle hash

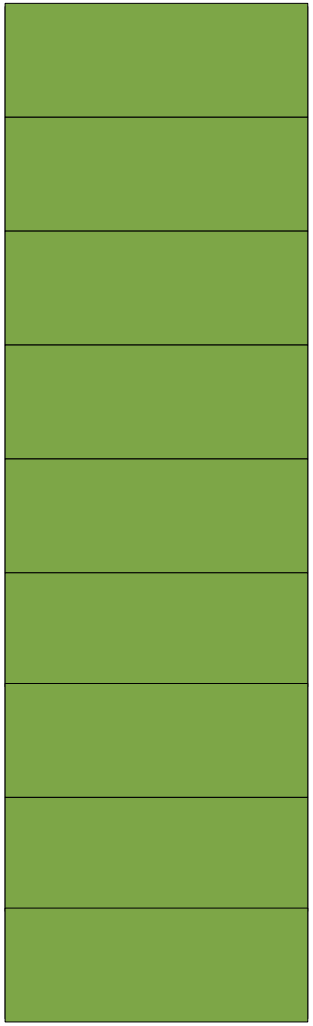
Vantaggi e svantaggi delle tabelle delle pagine

- **Vantaggi**
  - rappresentazione naturale se si adotta un approccio processo-centrico
  - la tabella delle pagine è ordinata in modo tale che sia semplice per il SO identificare il punto in cui si trova l'indirizzo del frame di interesse

Vantaggi e svantaggi delle tabelle delle pagine

- **Svantaggi**
  - una tabella delle pagine può essere grandissima (un elemento per ogni pagina del processo), contenere milioni di elementi e occupare troppo spazio di quella RAM che dovrebbe contribuire a gestire
- **Esistono approcci alternativi?**
- **invertiamo il fuoco e ricominciamo il discorso a partire dalla RAM anziché dai processi**

Tabella invertita



RAM

La RAM è suddivisa in frame, ogni frame può essere libero oppure contenere la pagina di un processo

Posso quindi pensare di mantenere una sola tabella con una entry di questo tipo per ogni frame:

< pid, p >

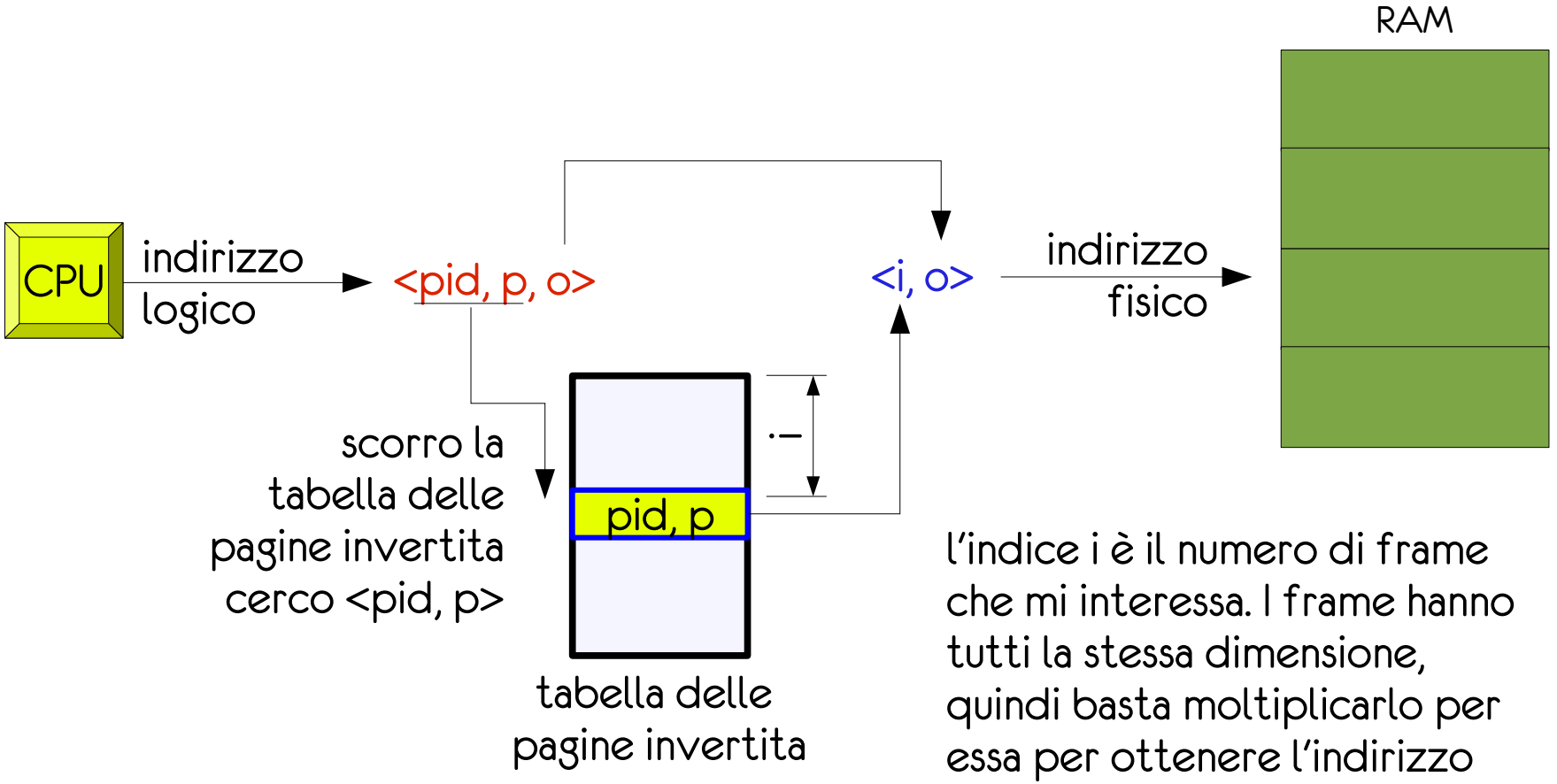
pid = processo, p = pagina

Gli indirizzi logici saranno quindi triple di questo tipo:

< pid, p, o>

pid = processo, p = pagina, o = offset

Struttura



l'indice i è il numero di frame che mi interessa. I frame hanno tutti la stessa dimensione, quindi basta moltiplicarlo per essa per ottenere l'indirizzo base del frame

Vantaggi e svantaggi

- **Vantaggi**
  - rappresentazione più compatta
  - richiede meno spazio in memoria
- **Svantaggi**
  - tempi d'accesso aggravati dalla ricerca in tabella
  - per ovviare a questo limite talvolta la tabella delle pagine invertita è implementata come una hash table talvolta ci si appoggia a un TLB