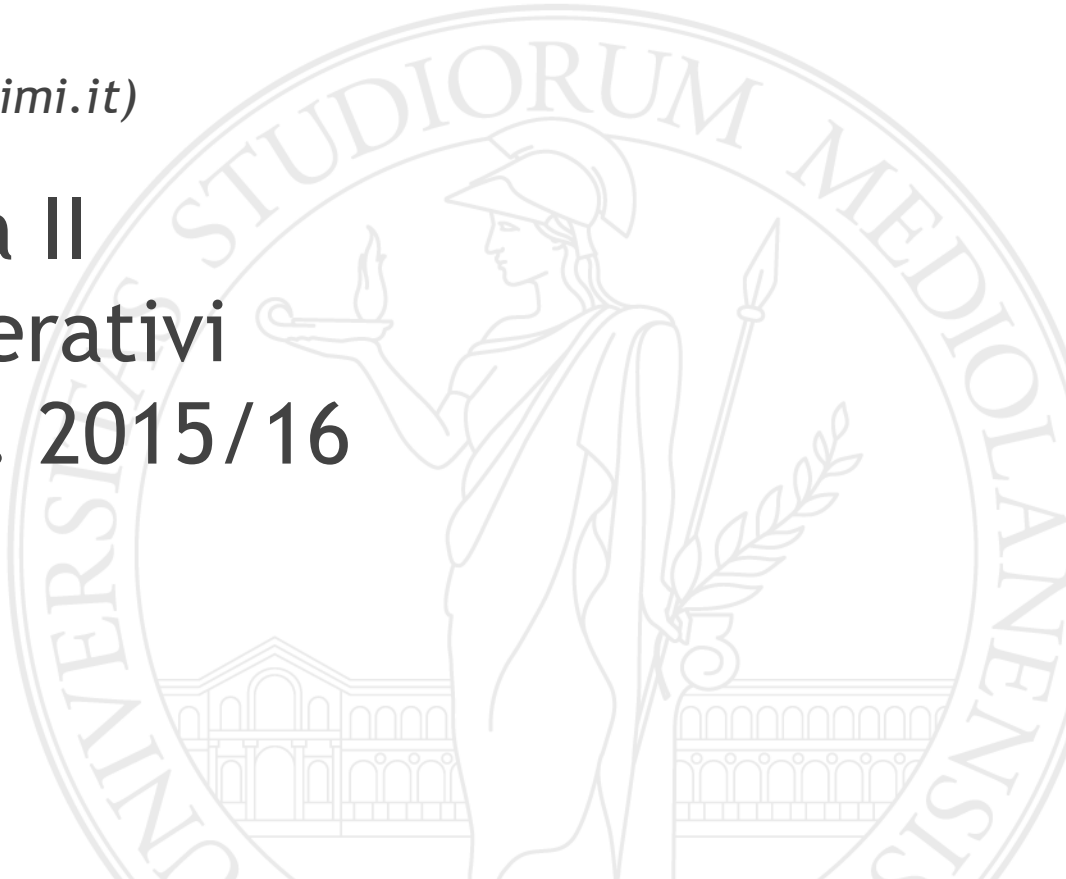
UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Alberto Ceselli
(alberto.ceselli@unimi.it)

Informatica II
Sistemi Operativi
DIGIP - a.a. 2015/16



Sistemi Operativi

(modulo di Informatica II)

Gestione della memoria centrale

Patrizia Scandurra

Università degli Studi di Bergamo

a.a. 2014-15

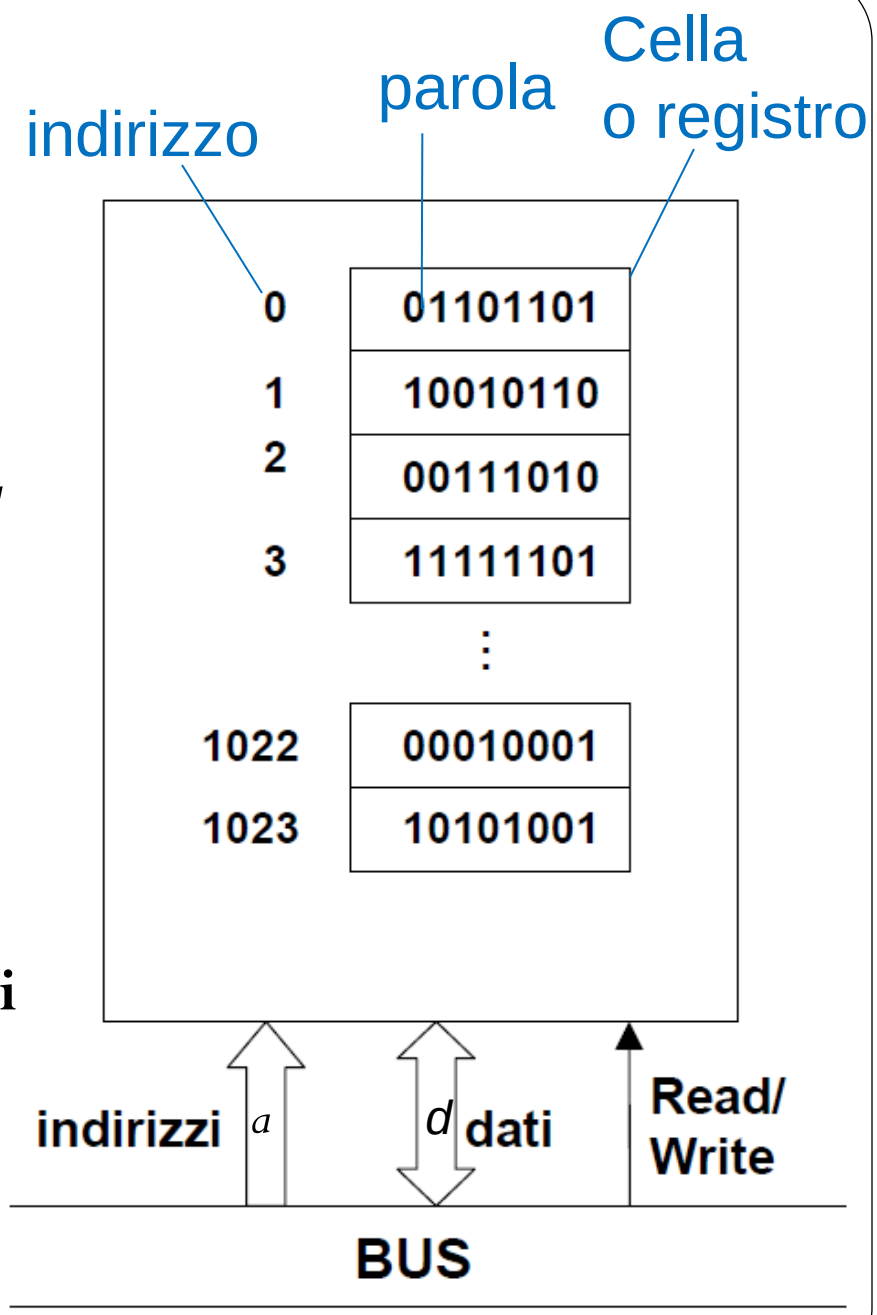
Sommario

- Sistemi multiprogrammati e il gestore della memoria
- Concetti generali
- Swapping
- Allocazione contigua di memoria
- Paginazione
- Segmentazione
- Segmentazione con paginazione

Memoria centrale

- Contenuto delle celle non riconoscibile
- Una istruzione o un dato possono occupare più celle consecutive
- Parallelismo di accesso è **l'ampiezza *d* della cella** e quindi della ***parola*** di memoria
  - Tipicamente, *d* è multiplo del byte:

  8 bit, 16 bit, 32 bit, 64 bit, 128 bit …
- Spazio di indirizzamento della CPU = **Max quantità di celle indirizzabili** = 2^a
  - Una evoluzione un po' diversa : 16 bit, 20 bit, 24 bit, 32 bit…



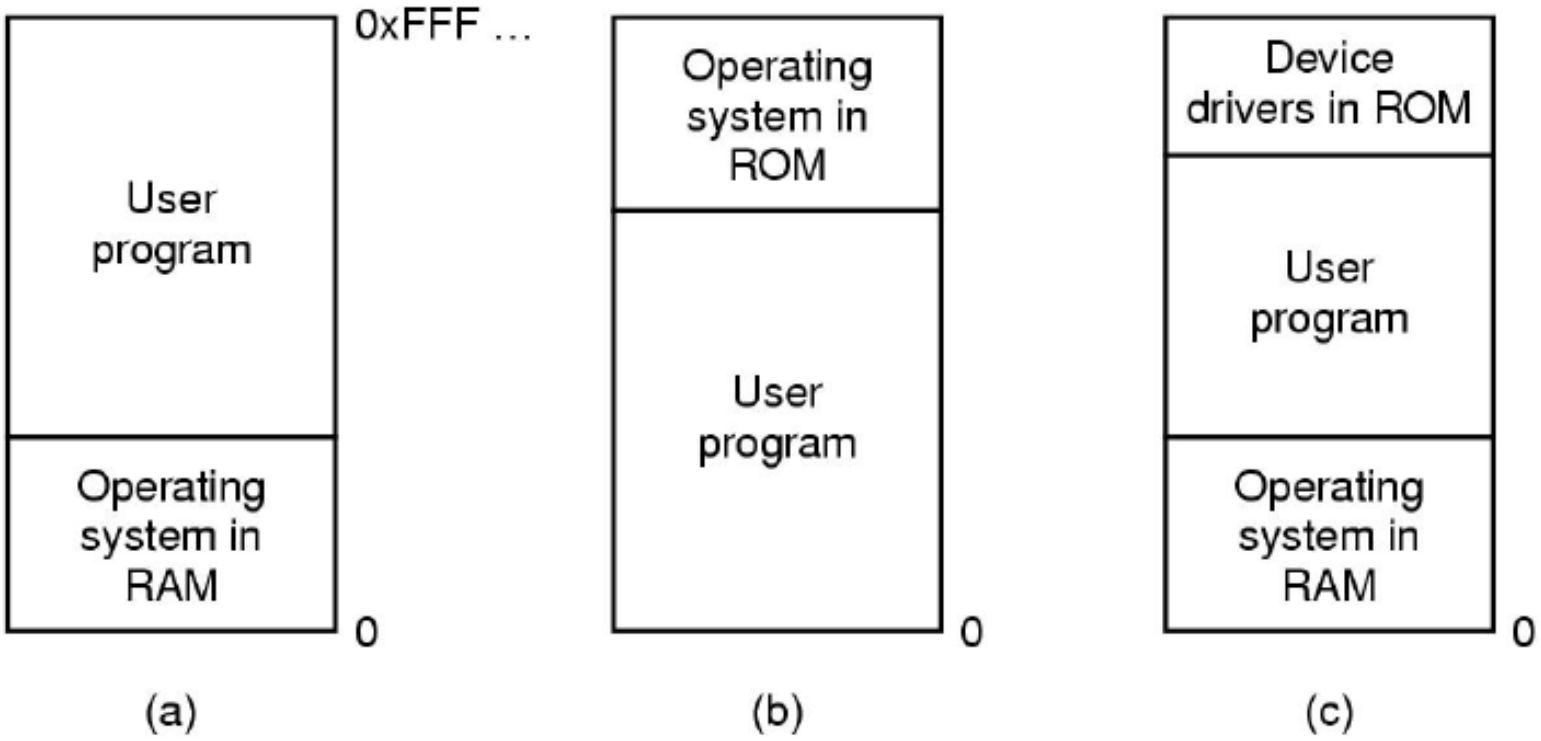
Attivazione di un programma

- Per essere eseguito, un programma deve essere portato (almeno in parte!) in memoria centrale ed "essere attivato come *processo*"a partire da un indirizzo
  - Quando un programma non è in esecuzione, non è strettamente necessario che stia in memoria centrale

- *Coda di entrata* – **processi su disco in formato binario (eseguibile)** che sono in attesa di essere caricati in memoria centrale per l'esecuzione

Monoprogrammazione

- Un solo programma in memoria (obsoleto)
- Programma+OS come in (a), (b) o (c) (DOS)



Sistemi multiprogrammati

- Più processi sono contemporaneamente pronti in memoria per l'esecuzione
  - processi nel sistema devono coesistere nello stesso spazio di indirizzamento fisico
  - i processi devono coesistere in memoria anche con il SO

- Tutto ciò comporta due principali necessità:
  - **Condivisione della memoria**
    - La memoria è logicamente partizionata in un'area di sistema e una per i processi utente
  - **Separazione degli spazi di indirizzamento**
    - Le differenti aree di memoria devono essere separate
    - in modo da non permettere ad un processo utente di corrompere il SO o addirittura bloccare il sistema

Gestore della memoria

- Ha il compito di gestire la memoria centrale (e una parte della memoria di massa) al fine di supportare l'esecuzione parallela dei proccessi
  - Spazio logico (o virtuale) di indirizzamento

- Funzioni principali:
  - Allocazione
  - Protezione
  - Condivisione controllata
  - Sfruttamento delle *gerarchie di memoria*

Concetti generali

- Indirizzi logici e indirizzi fisici
- Protezione
- Collegamento (*binding*) degli indirizzi logici agli indirizzi fisici
  - collegamento in compilazione
  - collegamento in caricamento
  - collegamento in esecuzione
- Caricamento dinamico

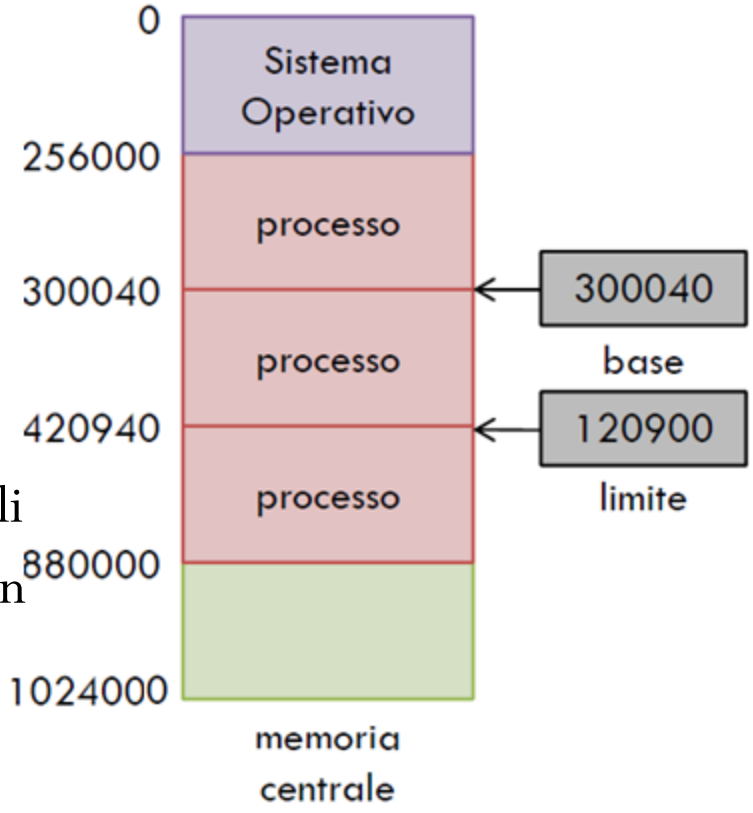
HW di base per la separazione degli spazi di indirizzamento

Consiste nell'utilizzare due registri:

- **Registro base**: contiene il più piccolo indirizzo fisico ammesso
- **Registro limite**: contiene la dimensione dell'intervallo ammesso
  - Solo il SO può accedere a questi registri

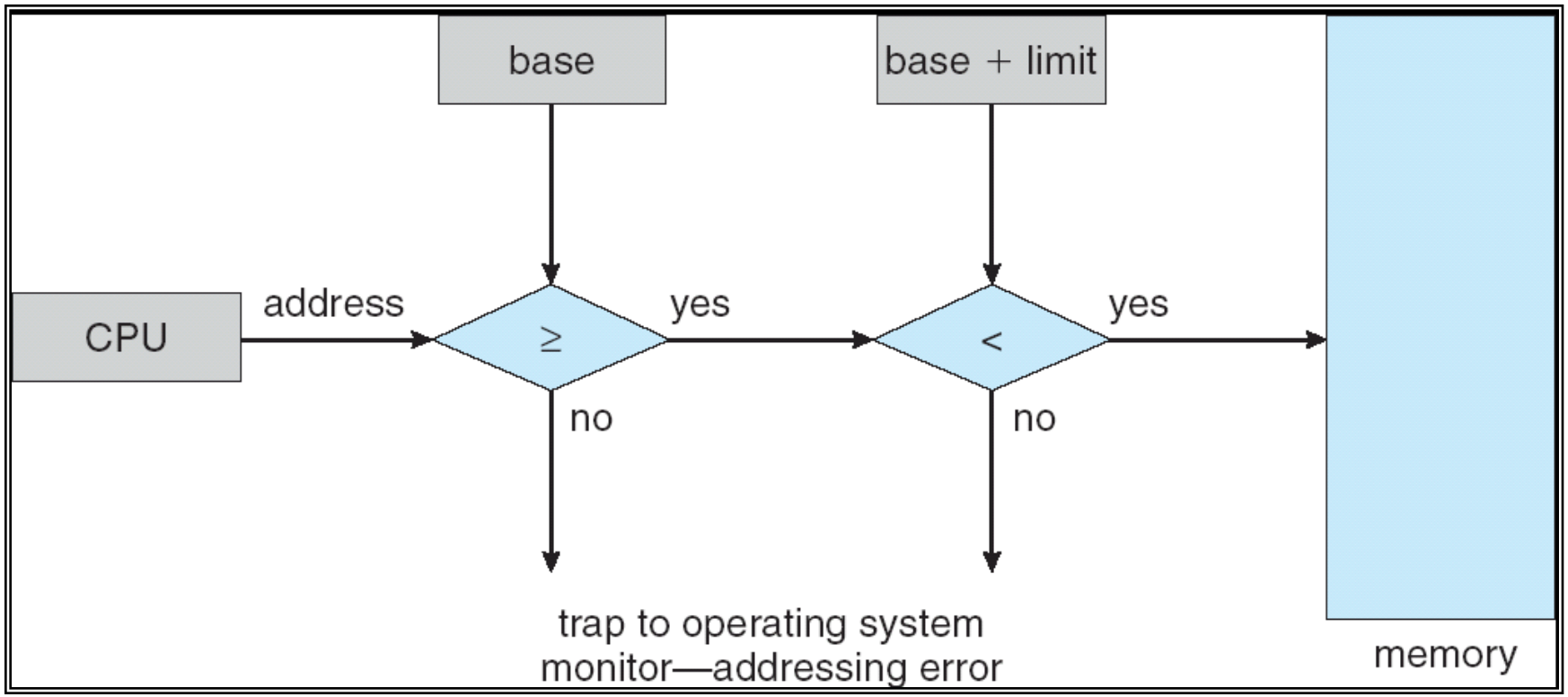
ed impedisce ai programmi utente di modificarli

  - Il SO può anche effettuare in questo modo un *dump (copia)* della memoria per scopi diagnostici



Protezione della memoria

- Per assicurare che non ci siano accessi illegali in memoria, la CPU confronta ogni indirizzo generato dal processo con i valori contenuti nel registro base e nel registro limite



Indirizzamento (1)

- La CPU esegue programmi che elaborano insiemi di dati in base ad una sequenza di istruzioni
- La CPU genera i riferimenti in memoria **per dati e istruzioni**, cioè gli *indirizzi (logici)*
  - *Indirizzamento o associazione (binding) delle istruzioni e dei dati alla memoria*
- I programmi utente passano attraverso più *stadi* prima di essere eseguiti, e in tali stadi **gli indirizzi cambiano "formato"** :
  - Compilazione (e linking)
  - Caricamento
  - Esecuzione

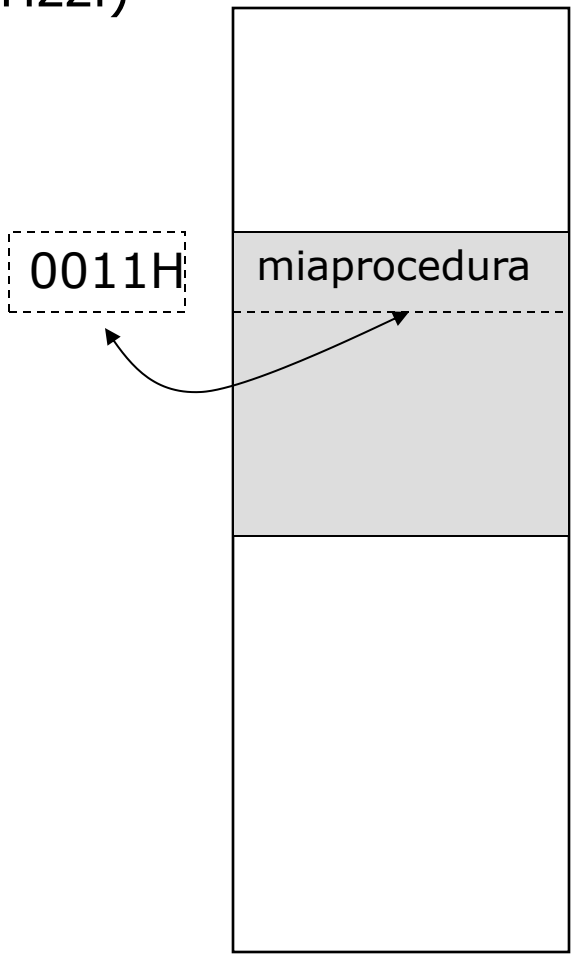
Indirizzamento (2)

LINKING (**collegamento** di varie procedure/moduli)
e **BINDING** (**associazione** degli indirizzi)

miaprocedura(V_1,...,V_n);

CALL miaprocedura

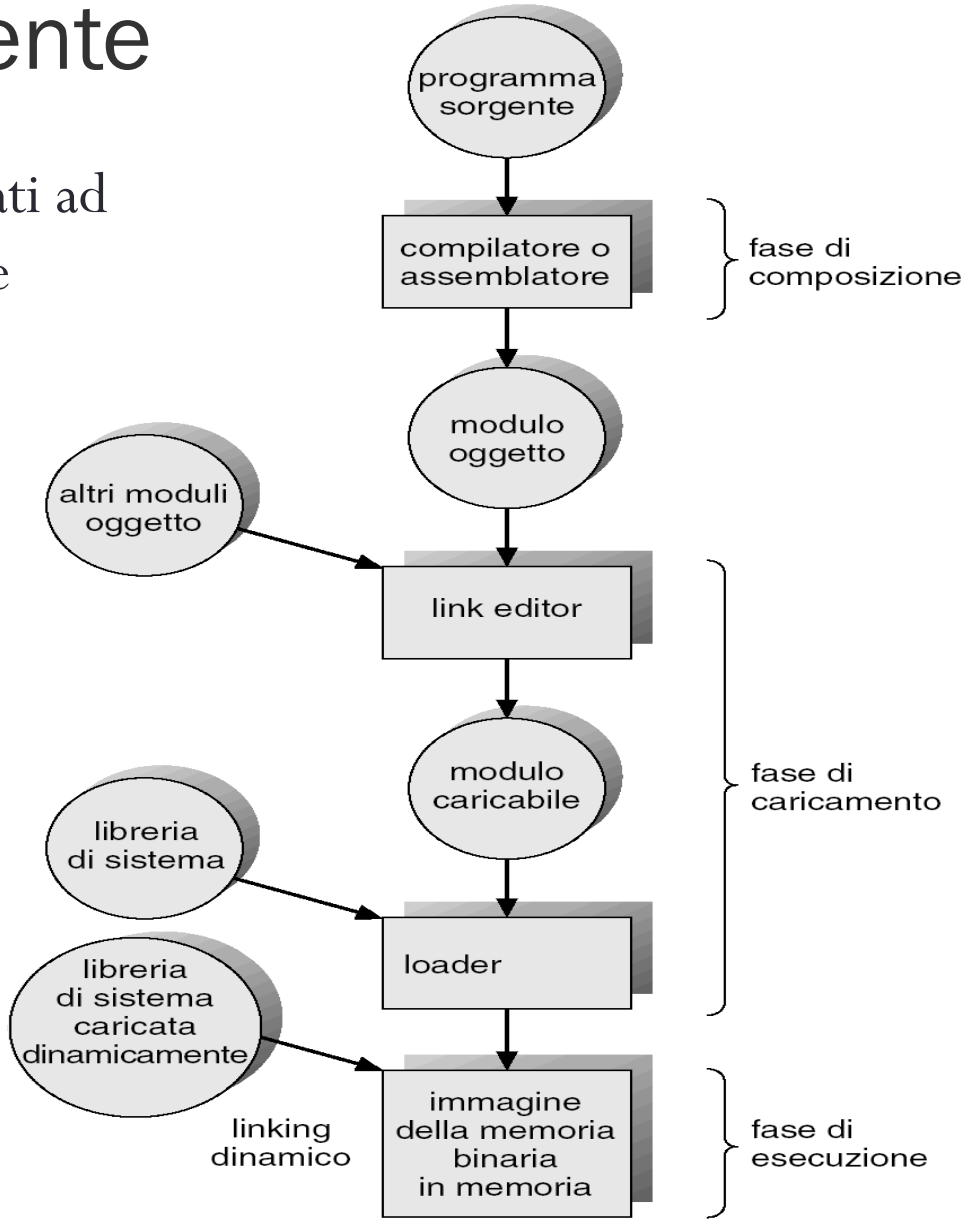
CALL 0011H



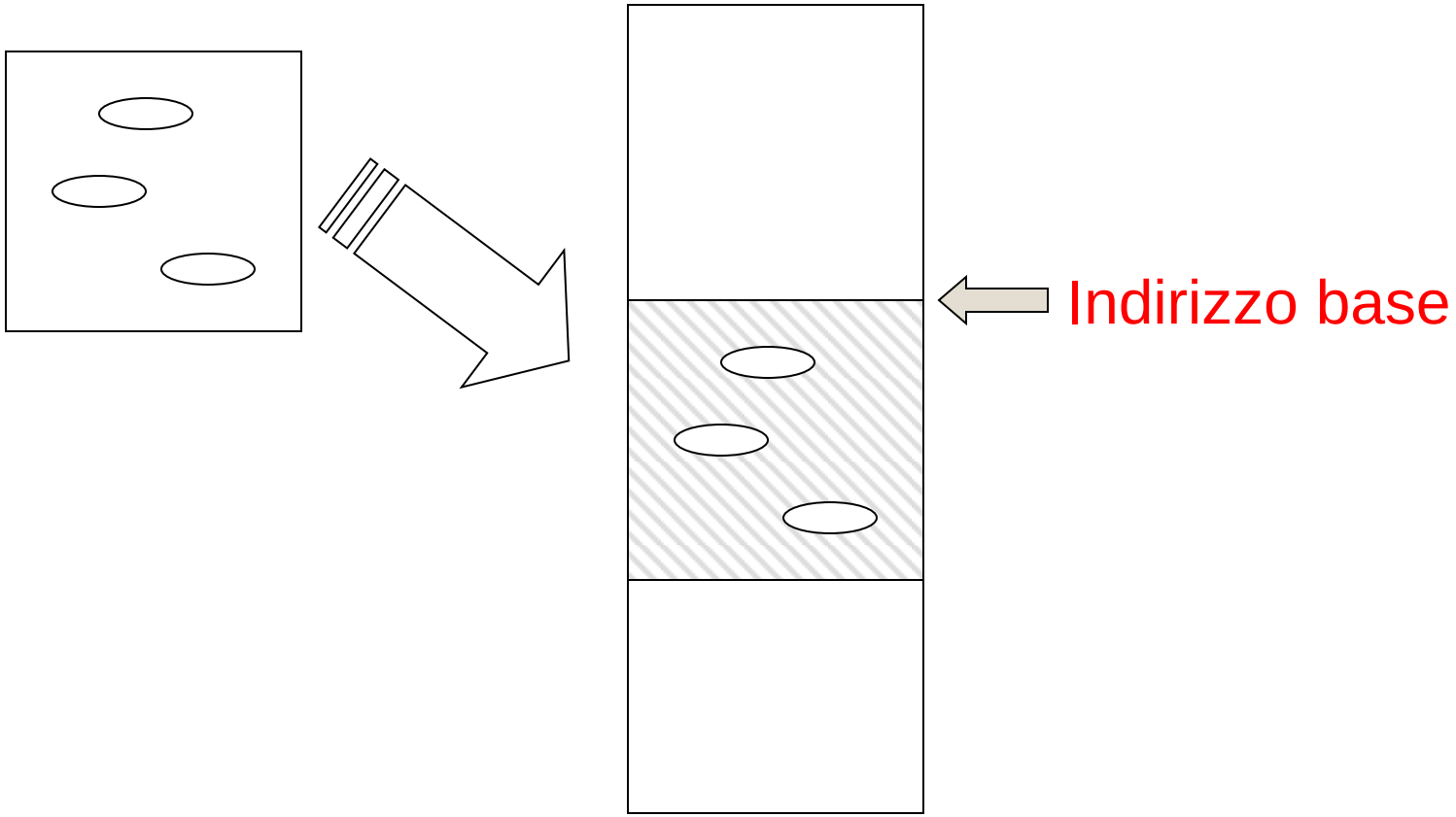
Processo a più fasi per l'esecuzione di un programma utente

L'associazione di istruzioni e dati ad indirizzi di memoria può essere eseguita in diverse fasi

- Associazione in fase di
  - Compilazione (e linking)
  - caricamento
  - esecuzione

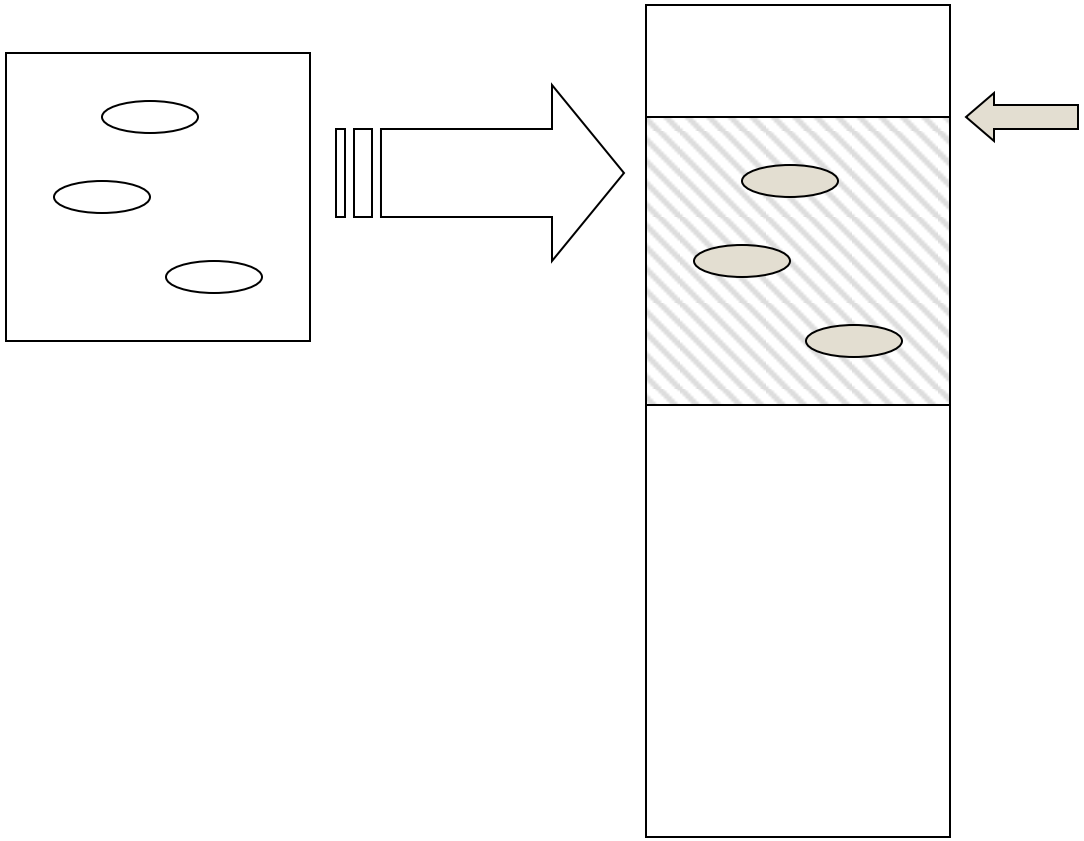


Collegamento in compilazione



Caricamento statico in posizione fissa (codice assoluto)

Collegamento in caricamento

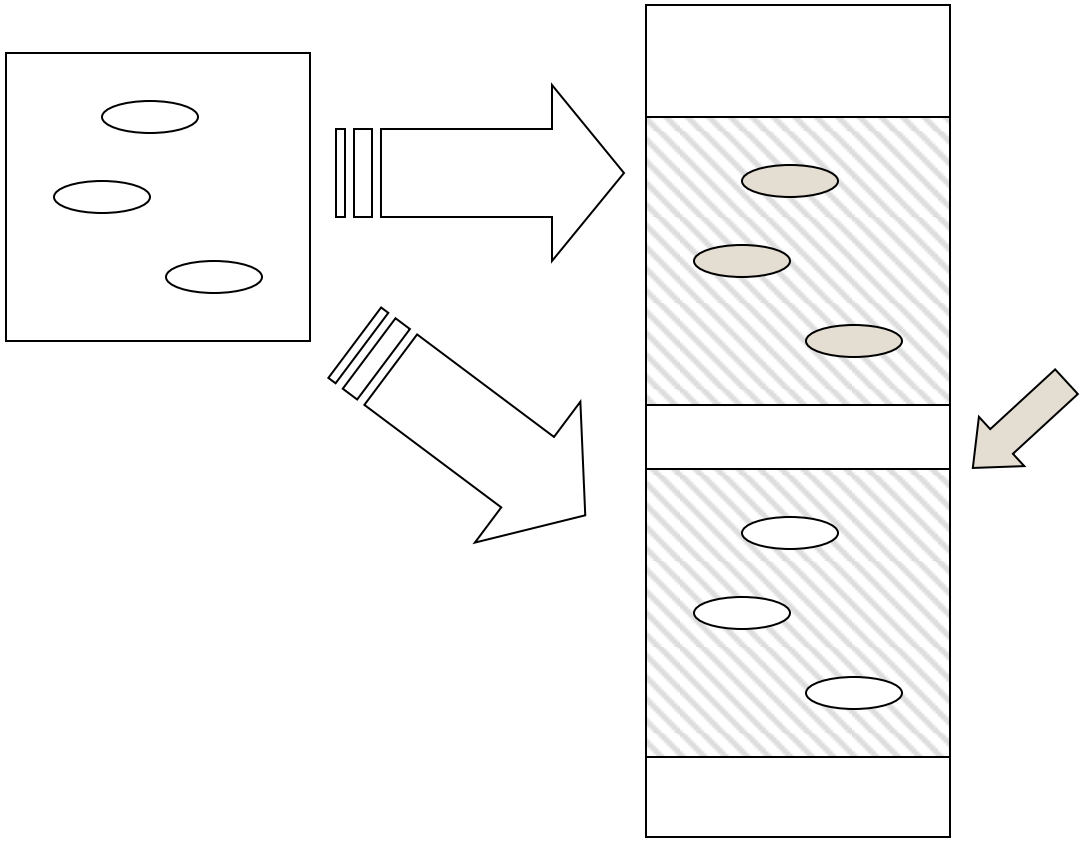


Indirizzo base

contenuto nel **registro di rilocazione** (o **registro base**)

Caricamento statico
con rilocazione del codice durante il caricamento

Collegamento in caricamento

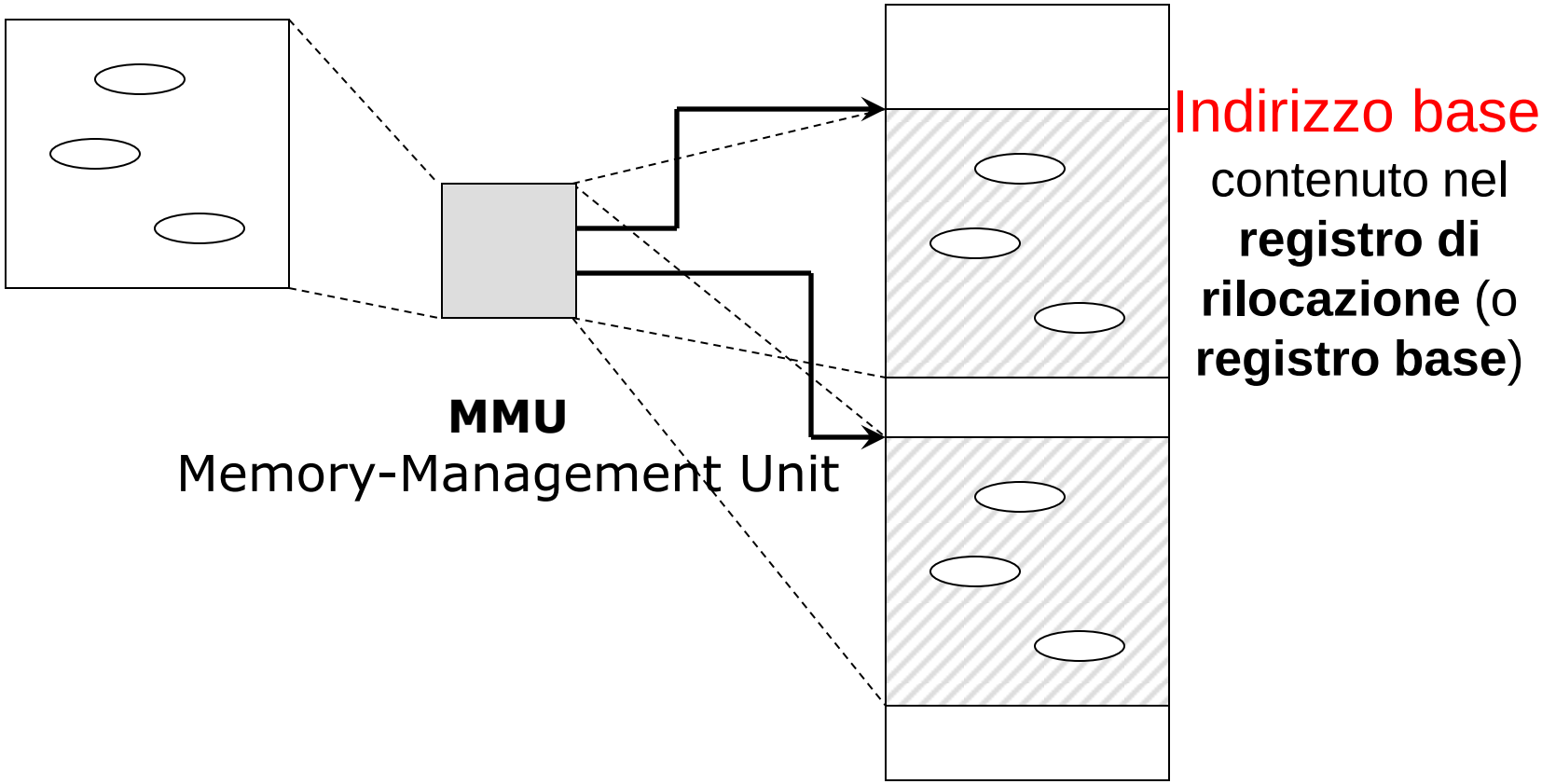


Indirizzo base

contenuto nel **registro di rilocazione** (o **registro base**)

Caricamento statico
con rilocazione del codice durante il caricamento

Collegamento in esecuzione



Caricamento statico
con rilocazione del codice in esecuzione

Spazio di indirizzamento logici e fisici a confronto

- Concetti basilari per un'adeguata gestione della memoria
  - ***Indirizzo logico*** – indirizzo generato dalla CPU; anche definito come *indirizzo virtuale*
  - ***Indirizzo fisico*** – indirizzo visto dalla memoria

- I metodi di collegamento degli indirizzi in fase di compilazione e di caricamento generano indirizzi logici e fisici identici,
- ma lo schema di collegamento degli indirizzi in fase di esecuzione da luogo a indirizzi logici e fisici (in generale) diversi

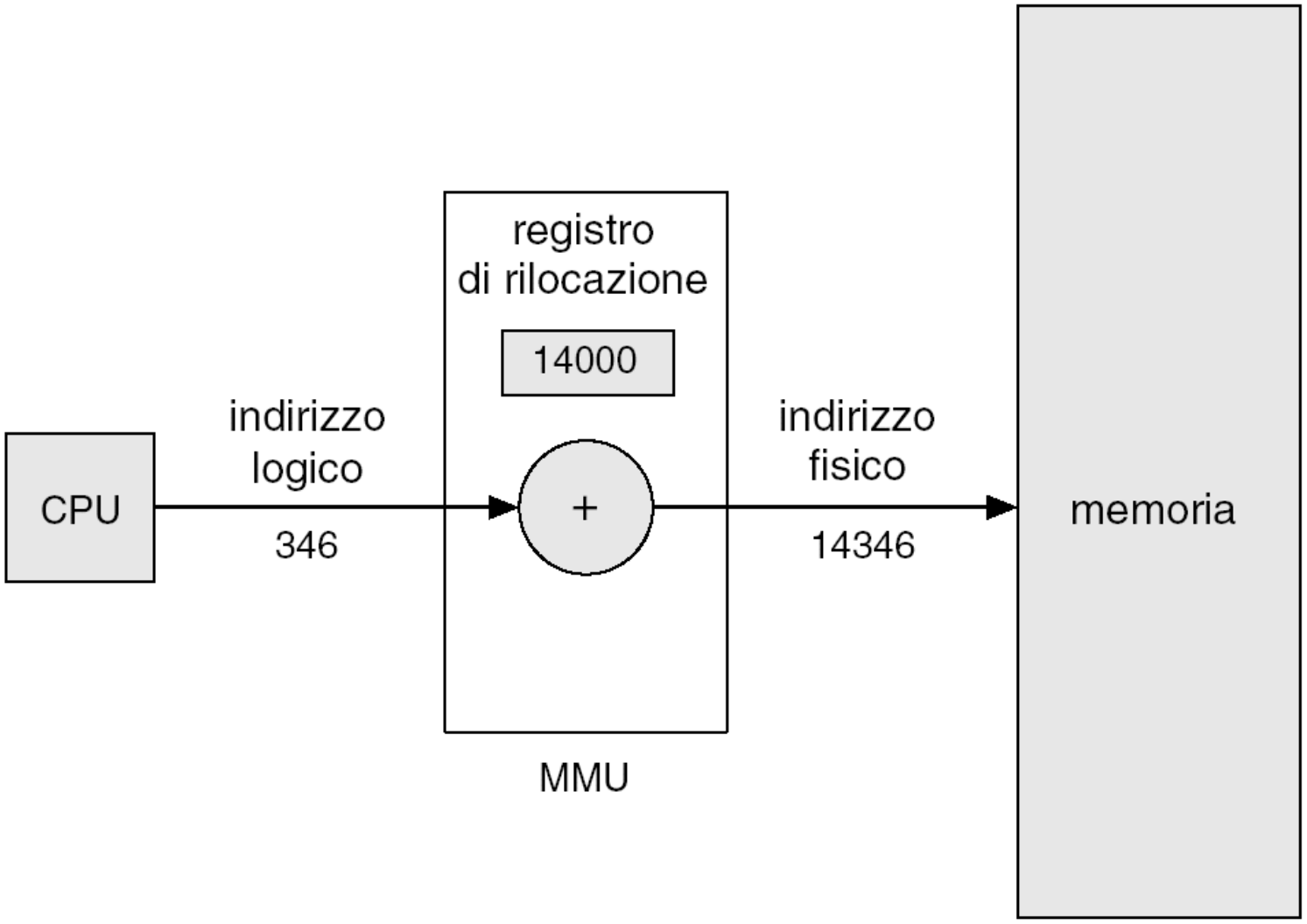
Unità di gestione della memoria centrale (MMU)

- **Dispositivo hardware** che realizza la trasformazione dagli indirizzi logici a quelli fisici in fase di esecuzione

- Nello schema di MMU, il valore nel registro di rilocazione è aggiunto ad ogni indirizzo generato da un processo nel momento in cui è trasmesso alla memoria

- Il programma utente interagisce con gli indirizzi logici; non vede mai gli indirizzi fisici reali

Rilocazione dinamica mediante un registro di rilocazione



Caricamento

- Caricamento statico: l'intero programma e tutti i suoi dati sono in memoria fisica
  - La dimensione di un programma non deve superare la dimensione della memoria (fisica) disponibile
- Caricamento dinamico: si carica una porzione di programma solo quando viene richiamata
  - Perché occupare memoria caricando tutto il programma?

Caricamento dinamico

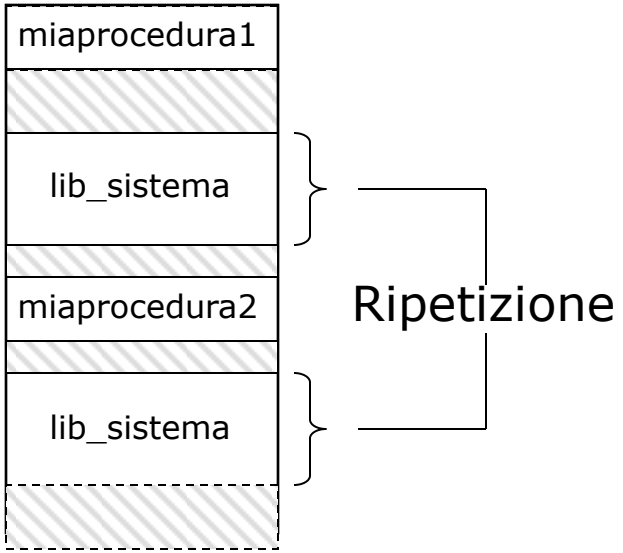
- **Una procedura non è caricata finchè non è chiamata**
- Migliore utilizzo dello spazio di memoria; una procedura inutilizzata non viene mai caricata
- Utile quando sono necessarie grandi quantità di codice per gestire situazioni che si presentano raramente
- Non richiede un supporto speciale da parte del SO, spetta al programmatore strutturare il programma in procedure

Collegamento (linking) dinamico

- **Il collegamento con le librerie è posposto fino al tempo di esecuzione**
- Una piccola parte di codice, **immagine** (***stub***), indica come individuare la procedura di libreria desiderata residente in memoria o come caricarla se non è già presente
- L'immagine rimpiazza sè stessa **con l'indirizzo della procedura** e la esegue
- Il SO deve controllare se la procedura necessaria è nello spazio di memoria di un altro processo o consentire l'accesso a più processi agli stessi indirizzi di memoria
- Il collegamento dinamico è particolarmente utile con le *librerie condivise*

Collegamento dinamico e librerie condivise

SENZA
COLLEGAMENTO DINAMICO
(COLLEGAMENTO STATICO)



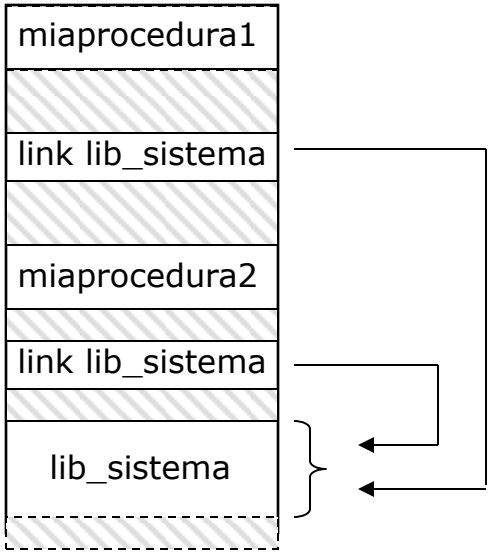
CALL miaprocedura1

CALL lib_sistema

CALL miaprocedura2

CALL lib_sistema

CON
COLLEGAMENTO DINAMICO



Allocazione della memoria

Due macro-approcci:

- **Allocazione contigua**:

tutto lo spazio assegnato ad un programma deve essere formato da celle consecutive

- **Allocazione non contigua**

E' possibile assegnare ad un programma aree di memorie separate

La MMU deve essere in grado di gestire la conversione degli indirizzi in modo coerente

Allocazione statica e dinamica

E ancora..

- **Allocazione statica**:
  - un programma deve mantenere la propria aerea di memoria dal caricamento alla terminazione
  - non è possibile rilocare il programma durante l'esecuzione

- **Allocazione dinamica**: durante l'esecuzione, un programma può essere spostato all'interno della memoria

Tecniche di gestione della memoria: problema ricorrente

- **Frammentazione interna**: lo spazio allocato in eccesso rispetto alle esigenze dei processi, inutilizzabile perché allocato
- **Frammentazione esterna**: lo spazio libero in aree troppo piccole per essere utili

Tutte le tecniche di gestione della memoria soffrono, in varia misura, di frammentazione interna

La frammentazione esterna può essere ridotta:

- Tecniche di compattamento basate sulla rilocazione (se i processi sono rilocabili dinamicamente)
- Paginazione

Tecniche di gestione della memoria

Il gestore della memoria si può basare su diversi meccanismi utilizzandoli in base a opportune politiche:

Allocazione contigua (statica e dinamica):

- **Swapping**
- **A Partizioni: (a) singola (b) partizioni multiple fisse (c) partizioni multiple variabili**

Allocazione non contigua:

- **Paginazione**
- **Segmentazione**
- **Segmentazione con paginazione**

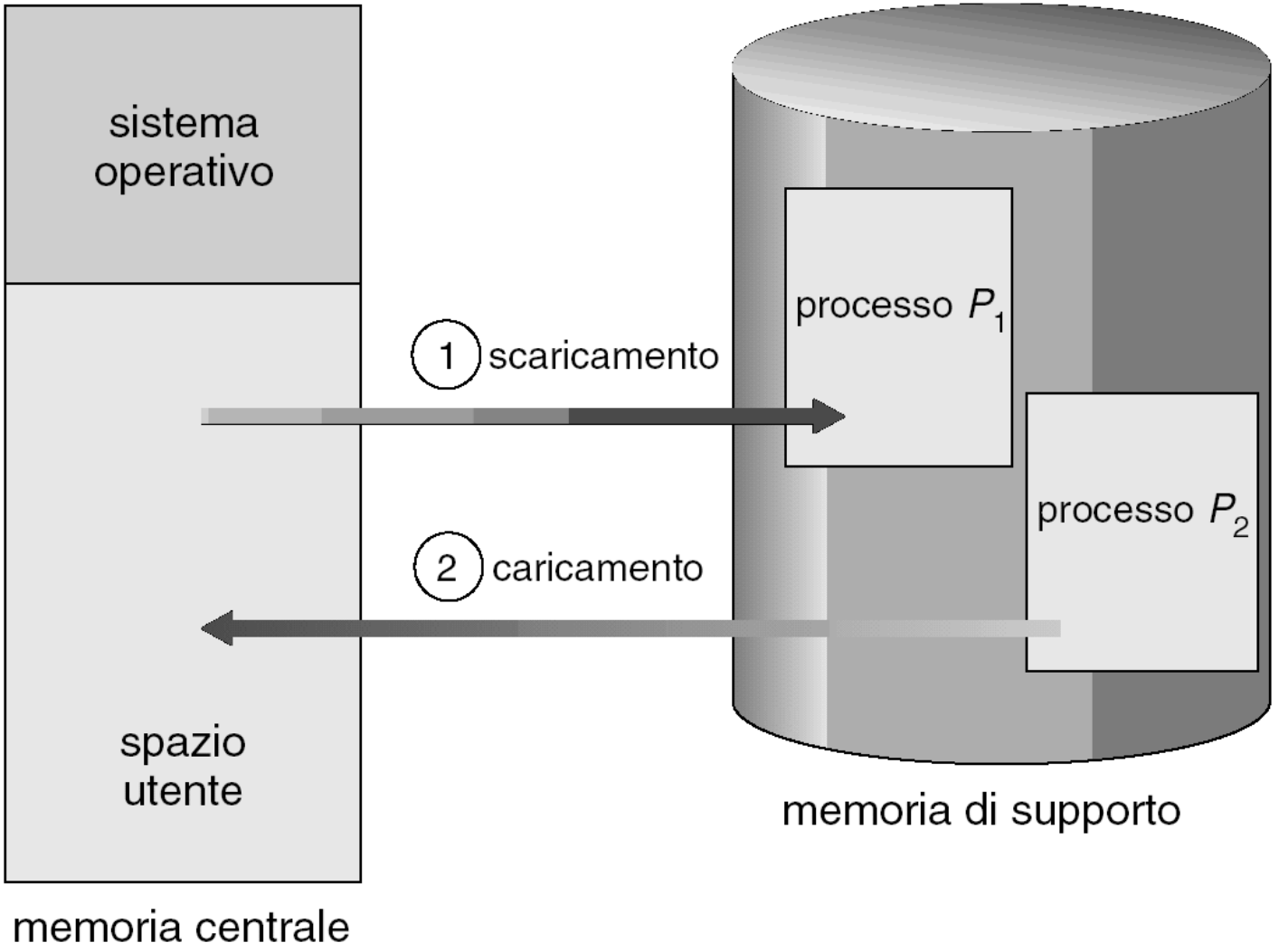
Swapping (1)

- **Un processo può essere temporaneamente scambiato** (swapped)
  - **spostandolo dalla memoria centrale ad una "memoria temporanea"** (area di swap)
  - **e poi in seguito riportato "interamente" in memoria centrale** per continuarne l'esecuzione

- Memoria temporanea – disco veloce abbastanza grande
  - da accogliere le copie di tutte le immagini della memoria centrale per tutti gli utenti,

  e che fornisce accesso diretto
  - a queste immagini

Visione schematica dello swapping



Swapping (2)

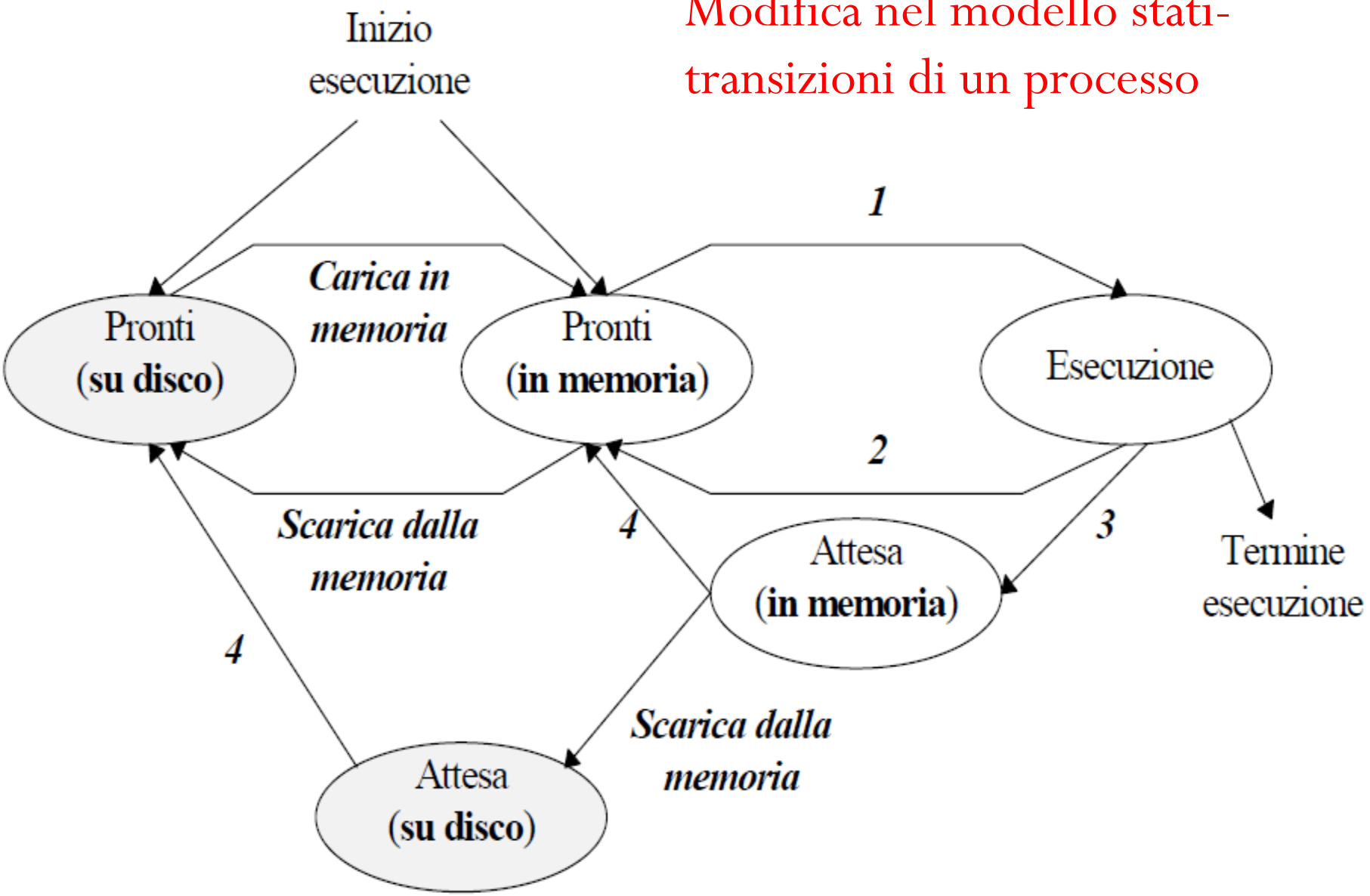
- **Permette di gestire più processi di quelli che fisicamente sono caricabili in memoria!**

- C'è un periodico scambio tra processi in memoria e su disco (*swapping*)
  - Se ne occupa lo *scheduler a medio termine*

- Quando il processo uscente subisce uno *swap out* viene copiato il descrittore di processo su memoria di massa
    - È inutile salvare le istruzioni: basta ricaricarle dal testo del programma memorizzato nel *file system*

Swapping (3)

Modifica nel modello stati-transizioni di un processo



Swapping (4)

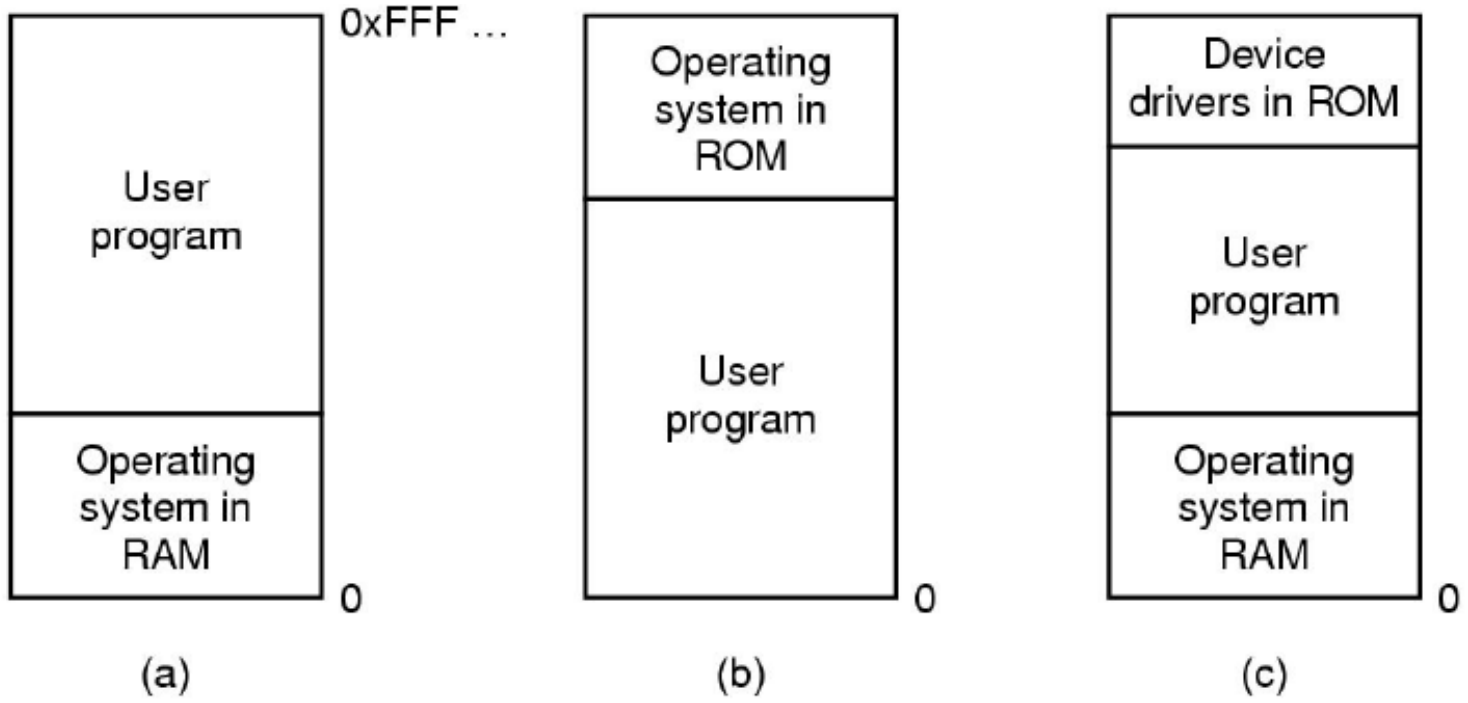
- Molto **comune nei sistemi con schedulatore Round Robin**
  - Il processo che finisce il quanto di tempo subisce lo swap-out
  - Lo swapping è soggetto ad altri vincoli
    - Se i processi sono impegnati in un I/O asincrono non possono essere scaricati per problemi di inconsistenza delle aree di memoria soggette ad I/O
- Roll out, roll in – **variante dello swapping** usata **per algoritmi di schedulazione basati sulla priorità**
  - un processo a bassa priorità è scambiato con un processo ad alta priorità
  - in modo che quest'ultimo possa essere caricato ed eseguito
- Versioni modificate di swapping si trovano in molti SO (ad esempio UNIX, Linux, e Windows)
  - swapping attivato/disattivato automaticamente in presenza di sovraccarico

Swapping (5)

- La maggior parte del **tempo di swap** è **tempo di trasferimento** (**context-switch**)
- Il tempo totale di trasferimento è *direttamente proporzionale* alla quantità di memoria spostata
- *Esempio:*
  - Con: un programma di 100 MB ed una velocità di trasferimento della memoria temporanea di 50 MB/s
  - Si ha T*trasferimento* = 100 (MB) / 50 (MB/s) = 2 s = 2000 ms
    T*trasferimento* + 8 ms (tempo di *latenza media*) = 2008 ms
  - Tempo di swap (**bidirezionale**) = 2008 * 2 = 4016 ms

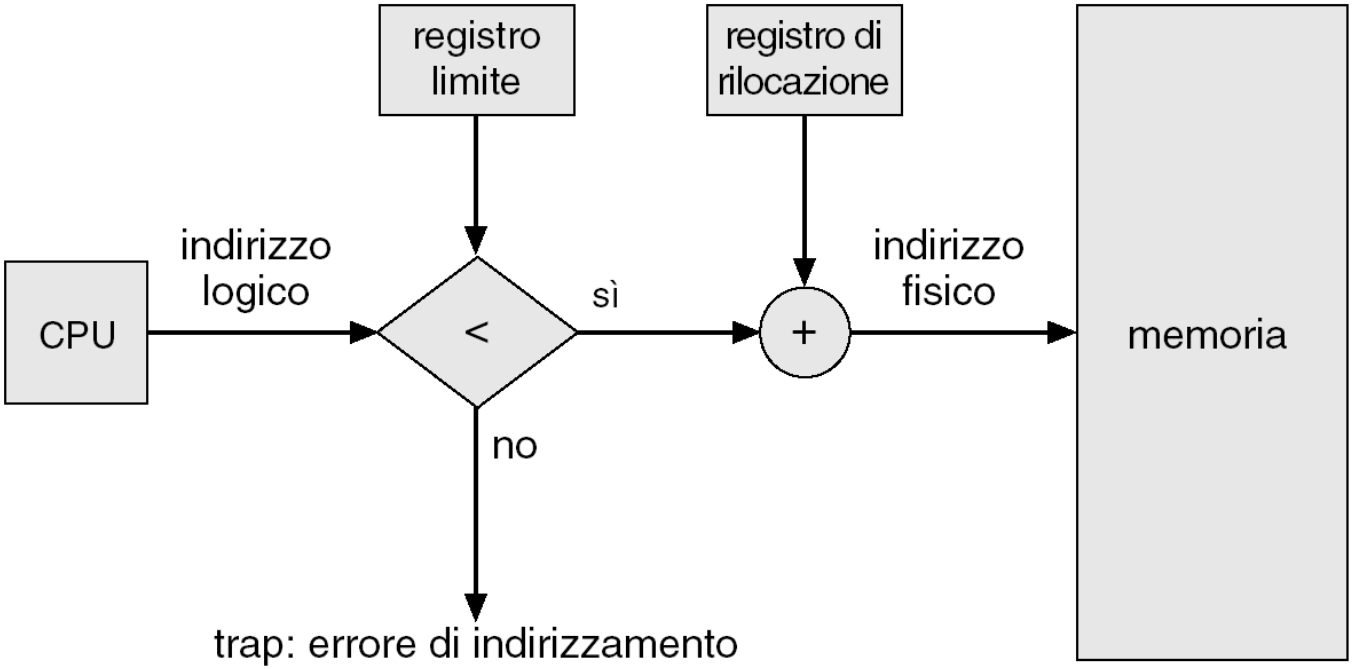
Allocazione a Partizione Singola (1)

- La memoria centrale è divisa in **due partizioni**:
  - Una per il SO (di solito collocato nella memoria bassa) e una per un processo utente (solitamente collocato nella memoria alta)
  - É uno schema di **allocazione contigua e statica**



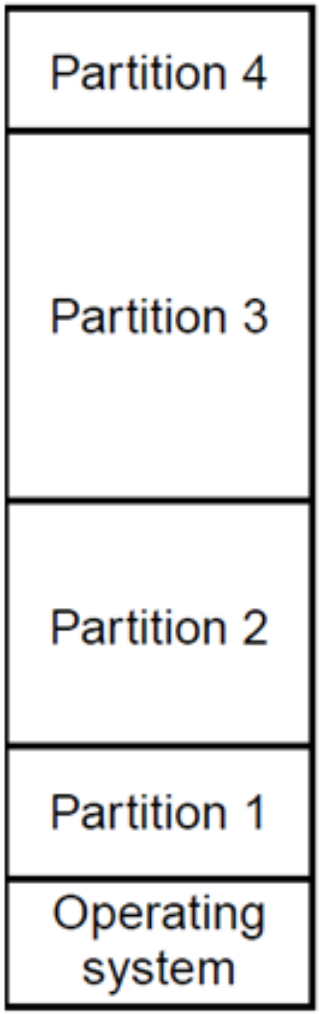
Allocazione a Partizione Singola (2)

- Per le applicazioni che devono essere caricate in memoria, si utilizza una **coda di input** dei processi
- **Spazi di indirizzamento dei processi delimitati** tramite un *registro base (o di rilocazione) e un registro limite:* locazioni che vanno da *base* a *base+limite-1*



Allocazione a Partizioni Multiple Fisse (1)

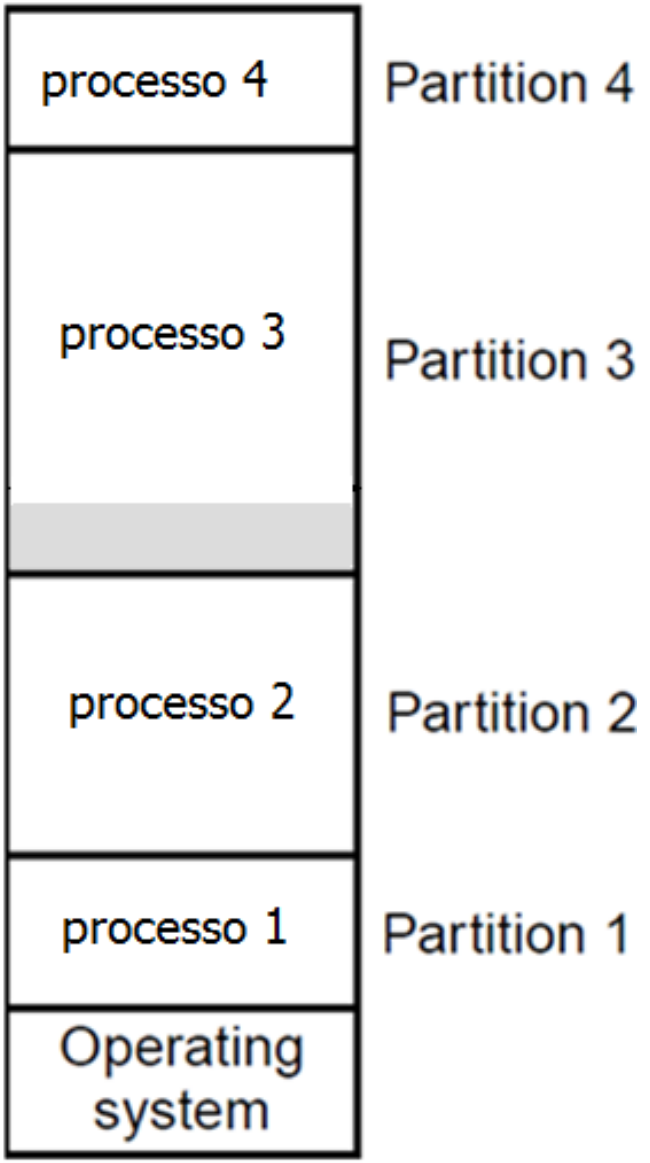
- La memoria è divisa in un **numero fisso *n* di aree** dette **partizioni** **di dimensioni** possibilmente **diverse**
  - Ogni partizione contiene un processo ed è identificata da coppia di registri *base-limite*
  - Quando c'è una partizione libera, un processo viene caricato in essa ed è pronto per essere schedulato per l'esecuzione
- É necessario conoscere la *dimensione del processo* prima di attivarlo; in fase di context switch il SO carica:
  - nel registro di rilocazione (base) l'indirizzo iniziale della partizione
  - nel registro limite la dimensione del processo



Allocazione a Partizioni Multiple Fisse (2)

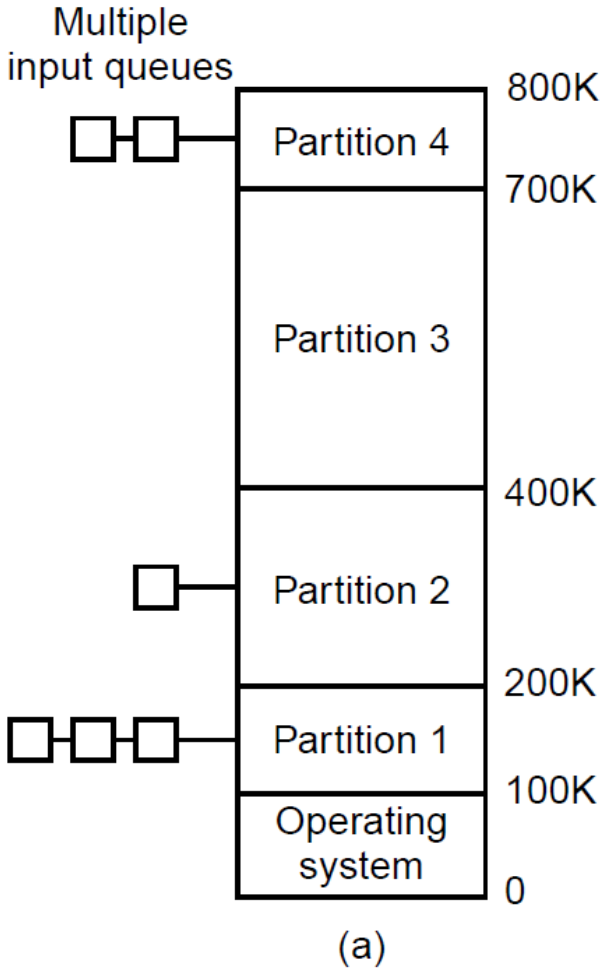
- **Processi piccoli**: causano spreco di spazio di memoria (frammentazione interna)

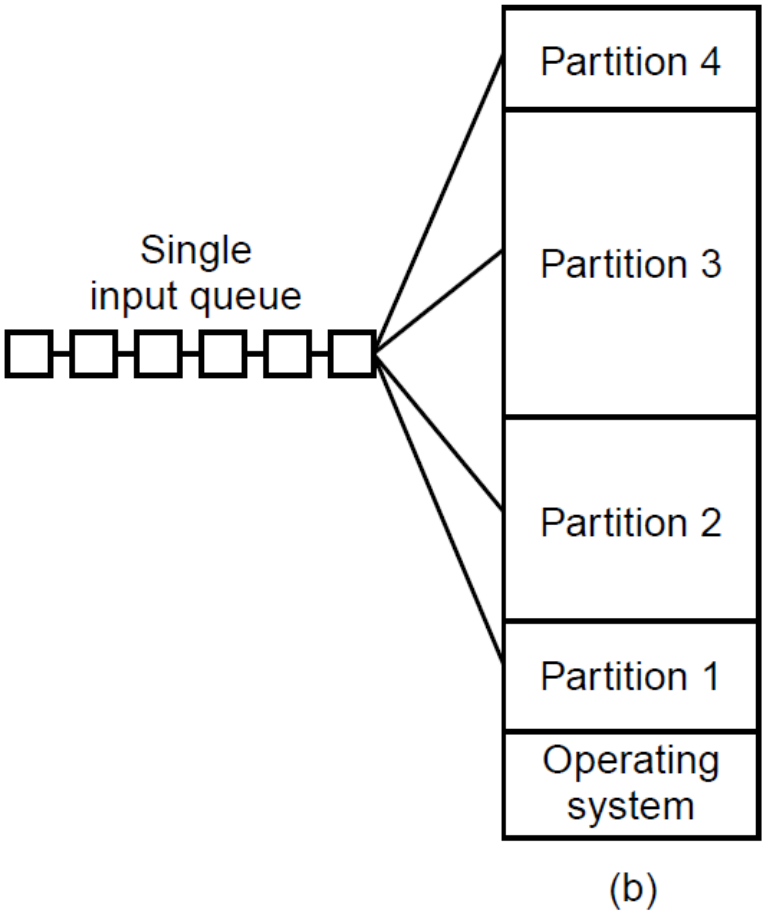
- **Processi grandi**: più grandi della più grande partizione non possono essere eseguiti
  - Aumentando la dimensione delle partizioni diminuisce il grado di multiprogrammazione,
  - e aumenta la frammentazione interna…



Allocazione a Partizioni Multiple Fisse (3)

- Il SO utilizza delle **code di input** per scegliere come allocare le partizioni ai processi: **(a) una coda per partizione** o **(b) una singola coda per tutte le partizioni**

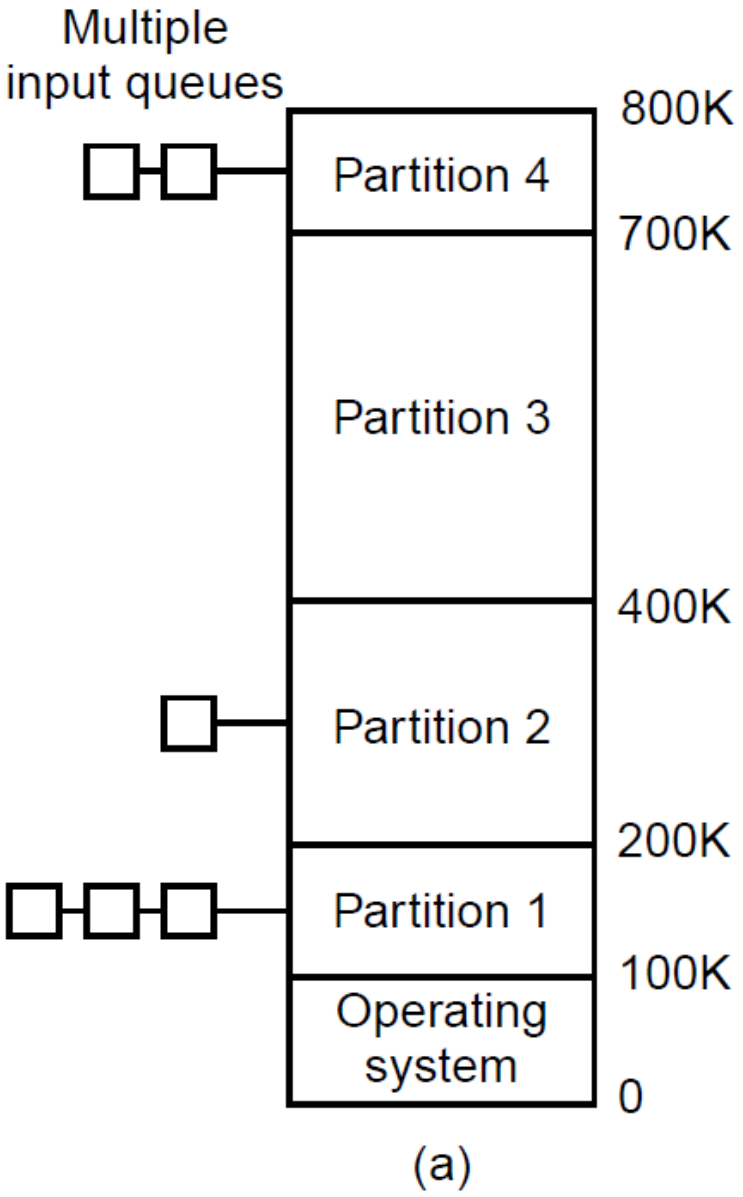




Allocazione a Partizioni Multiple Fisse (4)

Se si utilizza **una coda per ogni partizione**:

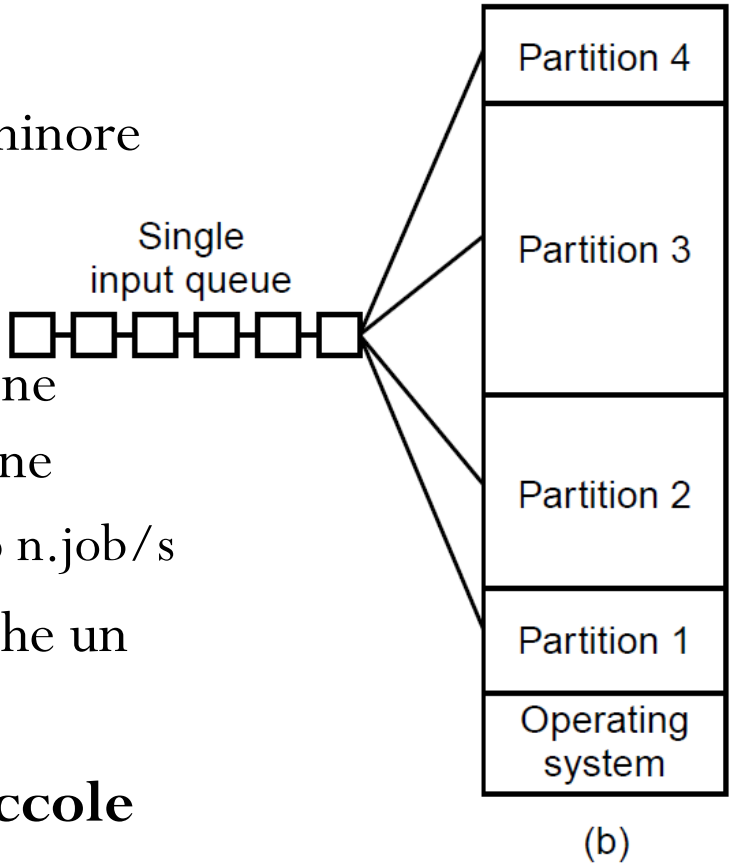
- **Un processo in coda su una partizione minima adeguata**
- Rischio di sottoutilizzare memoria, con code troppo lunghe rispetto ad altre



Allocazione a Partizioni Multiple Fisse (5)

Se si utilizza **una singola coda di attesa**: appena si libera una partizione, tra i processi in attesa di entrarvi si può scegliere con:

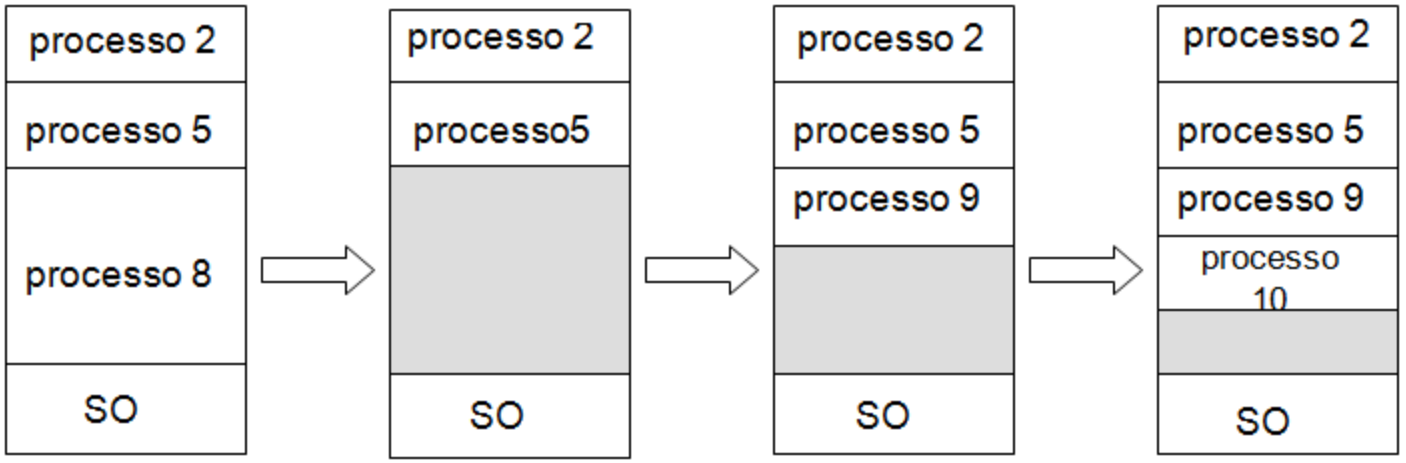
- **First-fit:** il primo processo con dimensione minore uguale della dimensione della partizione
  - non ottimizza tasso di occupazione
- **Best-fit:** il processo più grande con dimensione minore uguale della dimensione della partizione
  - Non equo per processi piccoli, peggiora il tasso n.job/s
- **Best-fit con upper bound** sul n. di volte che un processo è scartato
- **Best-fit definendo alcune partizioni piccole**



(b)

Allocazione a Partizioni Multiple Variabili (1)

- Il SO tiene traccia in una tabella di quali parti della memoria sono occupate e quali no:
  - ***Hole*** – blocco di memoria centrale disponibile
  - **Partizioni allocate**: il numero, la dimensione e la posizione variano dinamicamente
- Quando un processo arriva, il gestore di memoria cerca nell'insieme una partizione libera abbastanza grande per contenerlo completamente e la "ritaglia" a misura del processo



Allocazione a Partizioni Multiple Variabili (2)

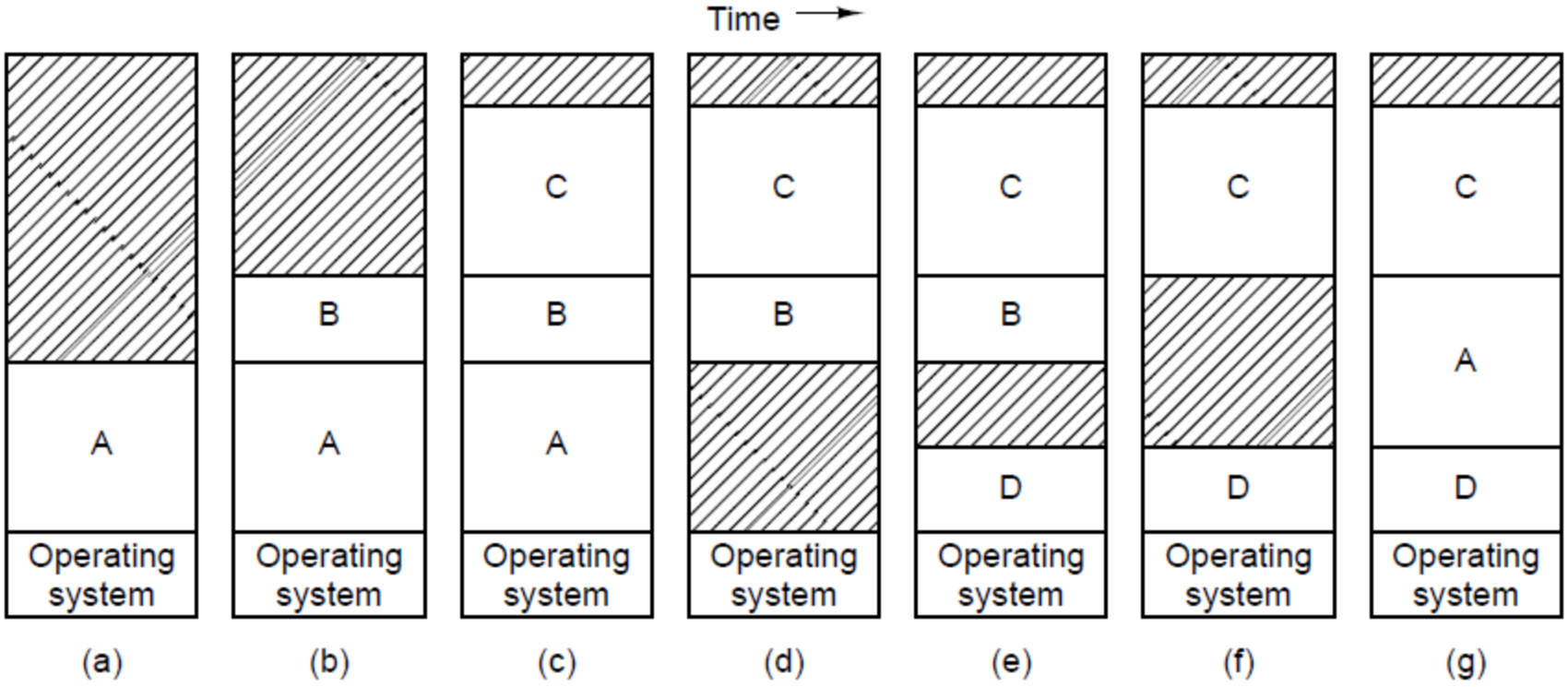
Come soddisfare una richiesta di dimensione *n* da una lista di blocchi liberi?

- **First-fit**: assegna il *primo* blocco libero abbastanza grande per contenere lo spazio richiesto
- **Best-fit**: assegna *il più piccolo* blocco libero *abbastanza grande*. Bisogna cercare nell'intera lista, a meno che la lista non sia ordinata in base alla dimensione
- **Worst-fit**: assegna il *più grande* blocco libero. Si deve nuovamente cercare nell'intera lista, a meno che non sia ordinata in base alla dimensione

Sia il metodo first-fit sia quello best-fit sono migliori del metodo worst-fit in termini di tempo e di utilizzo della memoria centrale

Allocazione a Partizioni Multiple Variabili (4)

ESEMPIO:



Alla lunga lo spazio libero appare suddiviso in piccole aree…è il fenomeno della frammentazione esterna

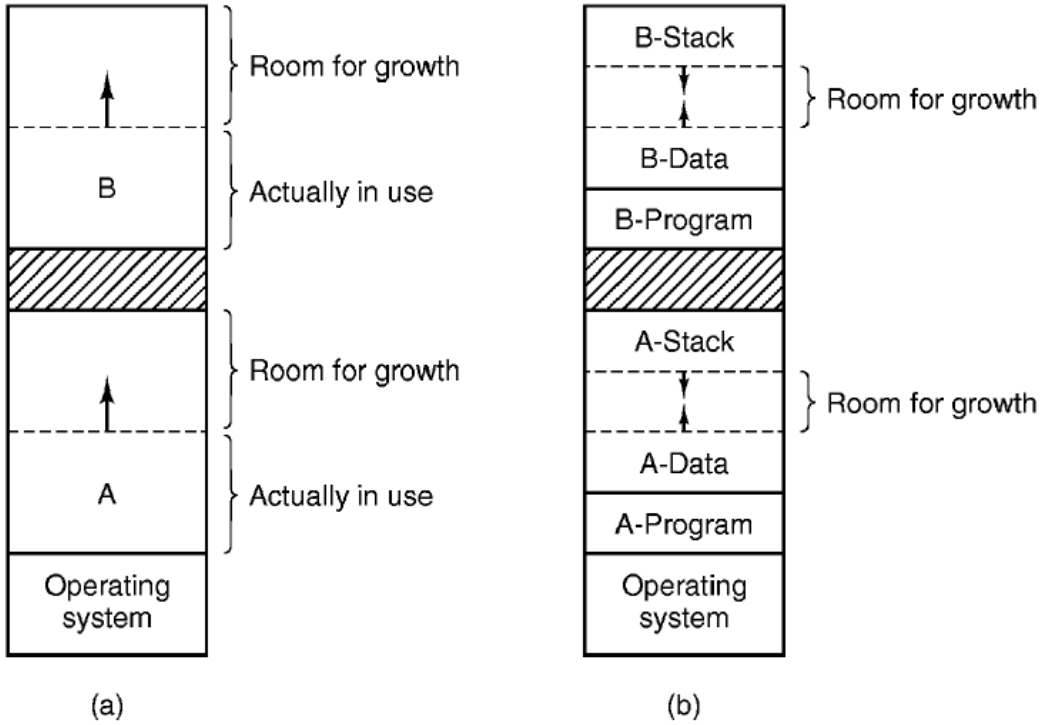
Compattazione

- **Frammentazione esterna** – fenomeno tipico della allocazione dinamica

- Ridurre la frammentazione esterna attraverso la **compattazione**:
  - Spostare in memoria tutti i programmi in modo da **"fondere" tutte le aree inutilizzate** per avere tutta la memoria centrale libera in un grande blocco
- *Svantaggi*:
  - La compattazione è possibilie solo se la rilocazione è dinamica ed è fatta al momento dell'esecuzione
  - E' un operazione molto onerosa: occorre copiare (fisicamente) in memoria grandi quantità di dati
  - Non utilizzabile in sistemi interattivi: i processi devono essere fermi durante la compattazione

Allocazione a Partizioni Multiple Variabili (5)

- Se durante l'esecuzione i processi "crescono", è buona idea allocare una piccola quantità di *memoria extra*
  - Tecnica usata per ridurre l'overhead dovuto allo spostamento del processo in memoria quando il processo non entra più nello spazio che gli è stato assegnato o allo scaricamento su disco (se combinata con SWAPPING)
- Strategie:
  a) Hole tra processi vicini
  b) Crescita tra stack e dati



Allocazione dinamica – strutture dati (1)

- In generale, ogni tecnica basata su "allocazione dinamica" ha bisogno di una struttura dati per mantenere informazioni sulle zone libere e sulle zone occupate

- Strutture dati possibili:
  - **mappa di bit**
  - **lista concatenata bidirezionale**

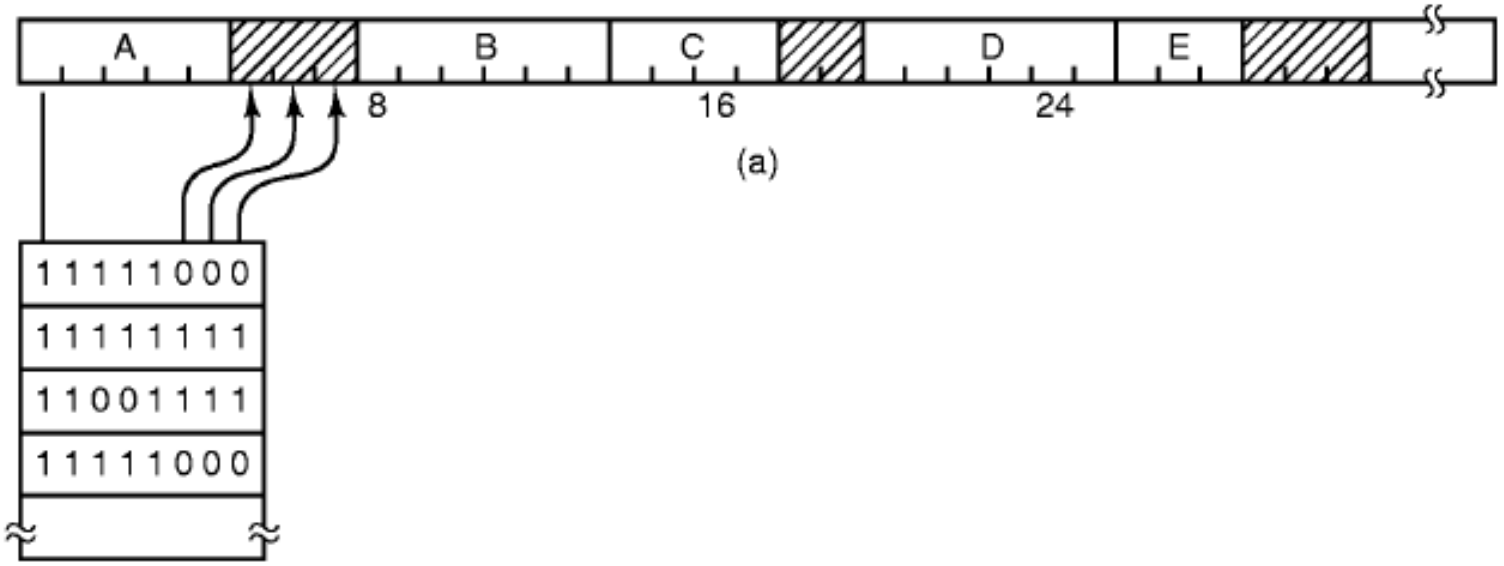
Allocazione dinamica – strutture dati (2)

Mappa di bit: la memoria viene suddivisa in *unità di allocazione*, ad ogni unità di allocazione corrisponde un bit in una bitmap: valore 0 (unità libera), valore 1 (unità occupata)

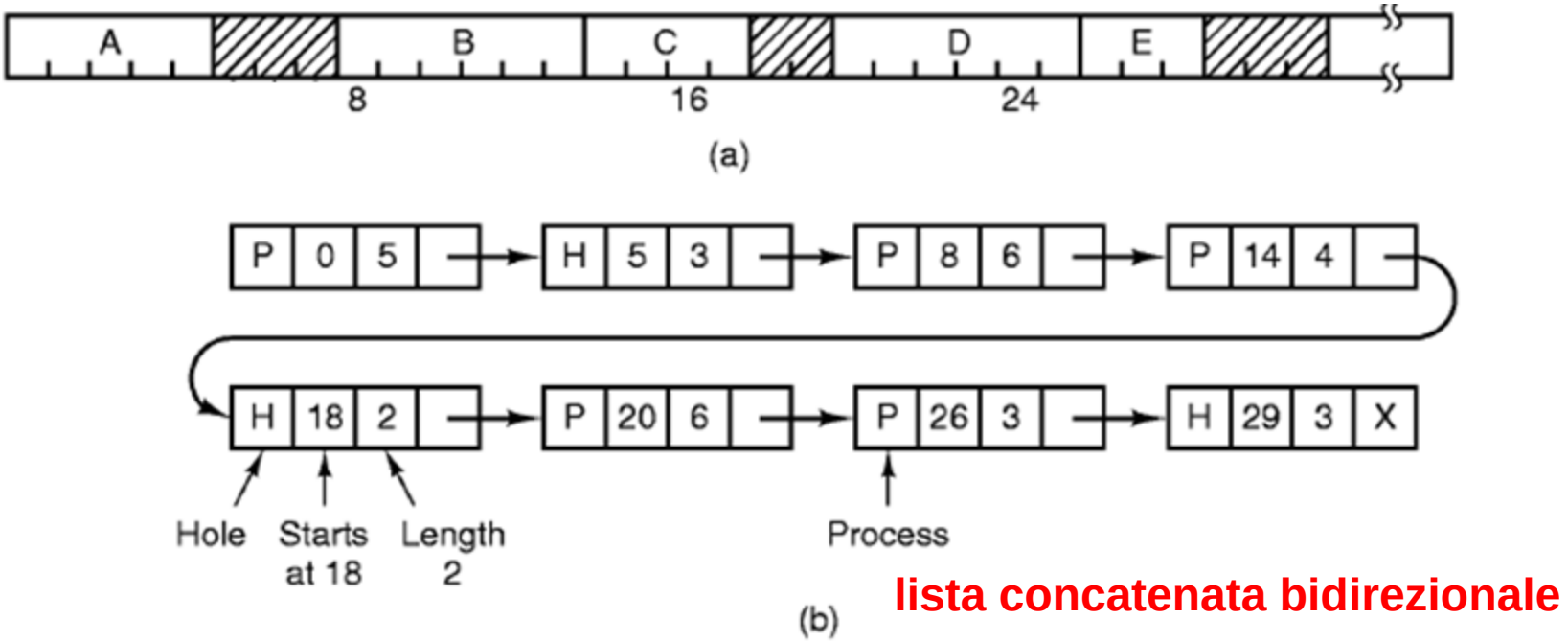
- La mappa di bit ha una dimensione fissa m e calcolabile a priori

Allocazione: per individuare in genere uno spazio di memoria di dimensione di k unità, è necessario cercare una sequenza di k bit 0 consecutivi

- tale operazione è O(m)



Allocazione dinamica – strutture dati (3)



lista concatenata bidirezionale

- Allocazione: *ricerca hole su lista, con diverse strategie*
  - Quando un blocco libero viene selezionato viene suddiviso in due parti: un *blocco processo P* della dimensione desiderata e un *blocco libero H* con quanto rimane del blocco iniziale
- Deallocazione: A seconda dei blocchi vicini, lo spazio liberato può creare un nuovo blocco libero, oppure essere accorpato ai blocchi vicini

Gestione dell'area di swap

- Concettualmente non è diverso dalla RAM, ma sta su disco (quindi le unità di allocazione sono blocchi, non byte)
- Per il resto, le tecniche di gestione per RAM sono valide anche per lo swap space
- ***Varianti***
  - swap space fisso, allocato alla nascita del processo, usato per tutta la durata del processo
  - swap space nuovo, allocato ad ogni swap-out
- In entrambi i casi **l'allocazione dello swap space (unica o ripetuta) può sfruttare tecniche e algoritmi per RAM**

Tecniche di gestione della memoria

Il gestore della memoria si può basare su diversi meccanismi utilizzandoli in base a opportune politiche:

Allocazione contigua (statica e dinamica):

- **Swapping**
- **A Partizioni: (a) singola (b) partizioni multiple fisse (c) partizioni multiple variabili**

Allocazione non contigua:

- **Paginazione**
- **Segmentazione**
- **Segmentazione con paginazione**

Paginazione

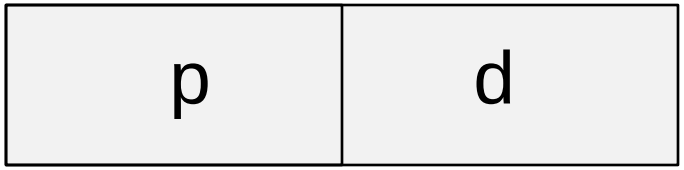
- I meccanismi a partizionamento fisso/dinamico non sono efficienti nell'uso della memoria (frammentazione interna/esterna)
- La paginazione è l'approccio utilizzato nei SO moderni per:
  - **ridurre il fenomeno di frammentazione esterna allocando ai processi spazio di memoria non contiguo**
  - si tiene **in memoria solo una porzione del programma**
    - aumento del numero dei processi che possono essere contemporaneamente presenti in memoria
  - La possibilità di **eseguire un processo più grande della memoria disponibile** (**memoria virtuale**)
- Attenzione però: **necessita di hardware adeguato**

Paginazione: metodo base

- Suddivide la memoria fisica in blocchi di **frame** della stessa dimensione (una potenza di 2, fra 512 byte e16 MB)
  - Lo spazio degli indirizzi fisici può essere non contiguo
- Divide la memoria logica in blocchi delle stesse dimensioni dei frame chiamati **pagine**
  - Il processo è sempre allocato all'interno della memoria logica in uno spazio contiguo
- Per eseguire un processo di dimensione di *n pagine*, bisogna trovare *n frame liberi* e caricare il programma
- Il SO imposta una **tabella delle pagine** per tradurre gli indirizzi logici in indirizzi fisici
  - Una per processo!

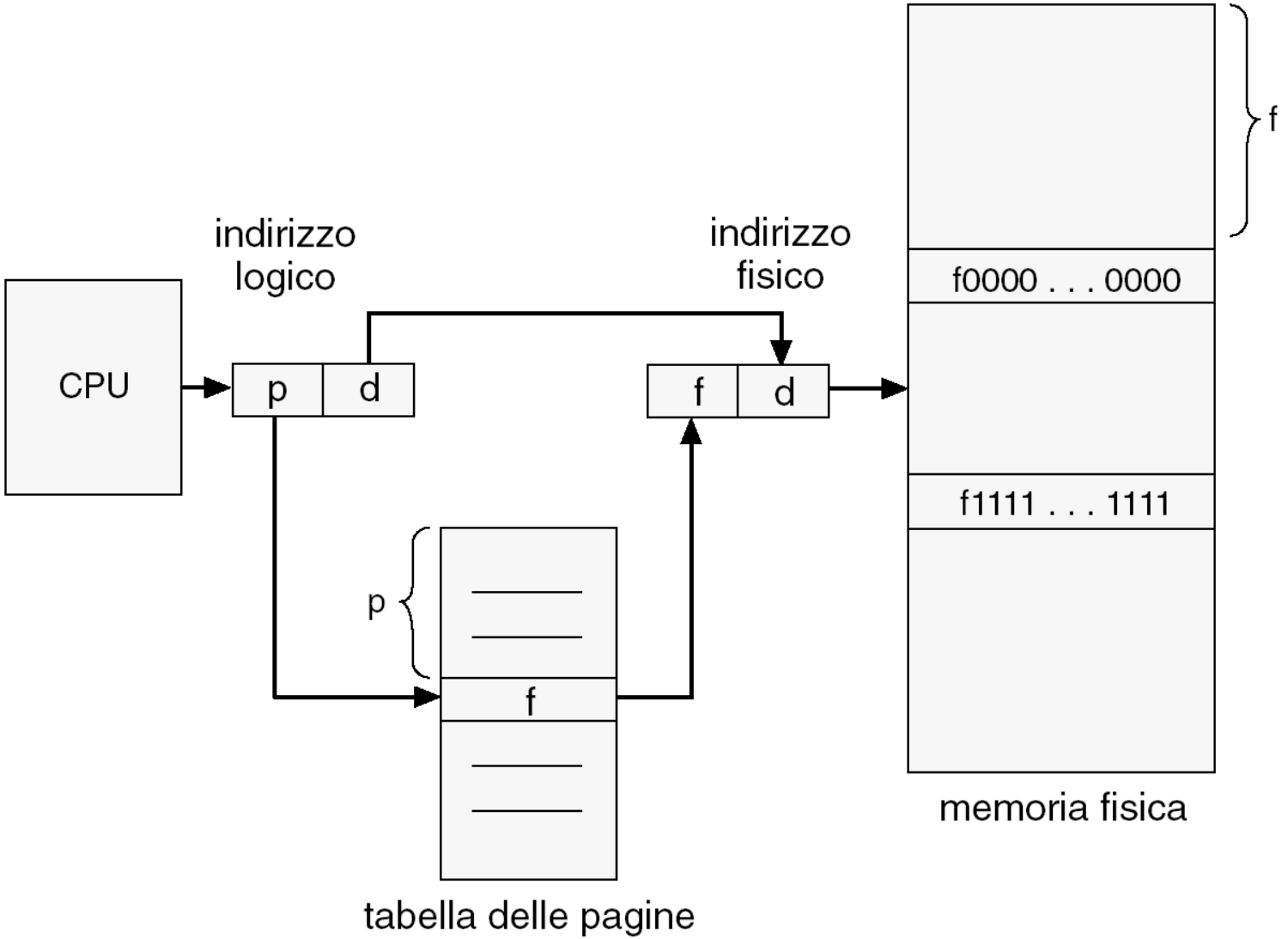
Paginazione: schema di traduzione dell'indirizzo (1)

- Ogni **indirizzo logico** generato dalla CPU è diviso in due parti:
  - *Numero di pagina (p)* – usato come indice nella *tabella delle pagine* che contiene l'indirizzo di base di ogni frame nella memoria fisica
  - *Spiazzamento nella pagina (d)* – combinato con l'indirizzo di base per calcolare l'indirizzo di memoria fisica che viene mandato all'unità di memoria centrale

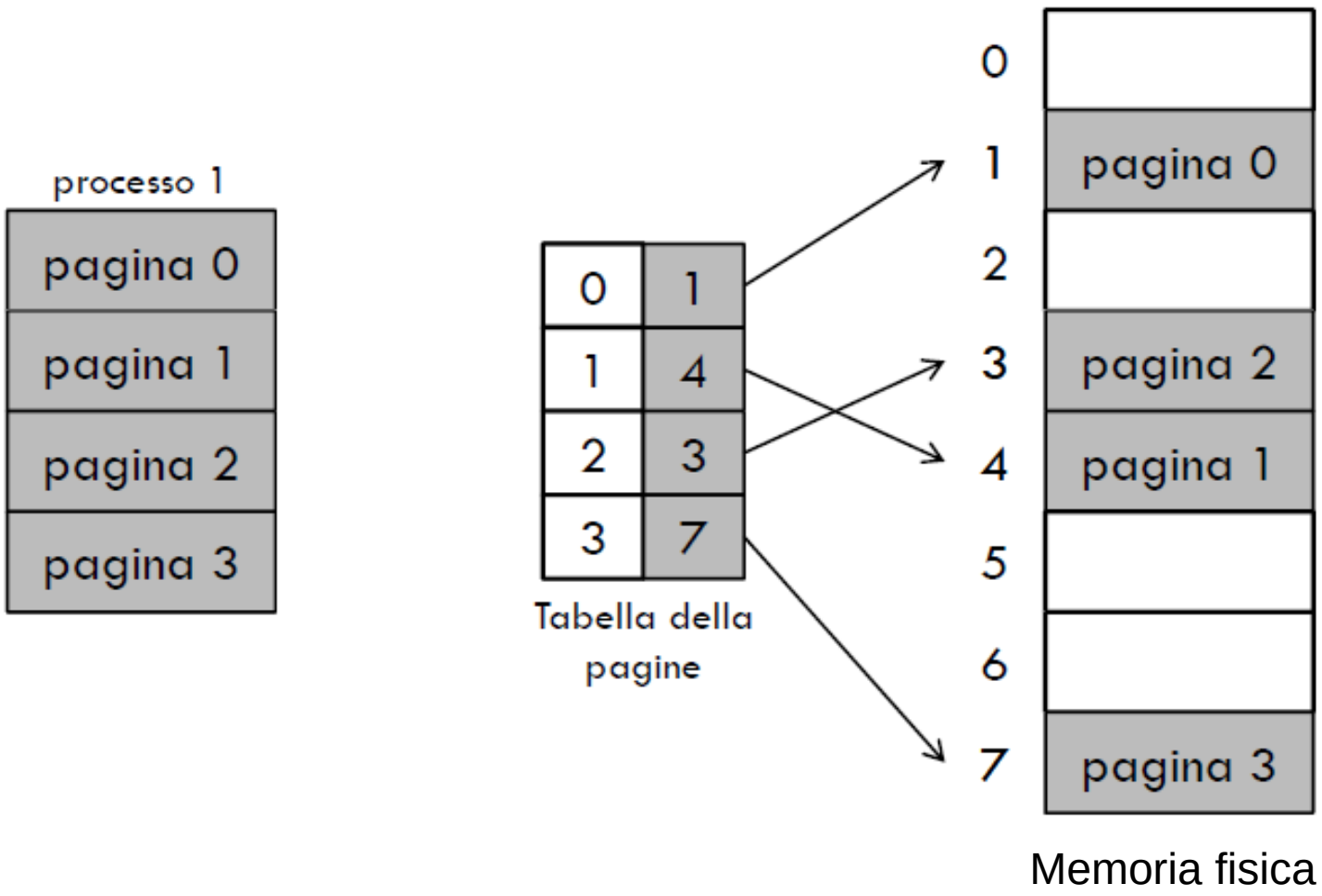


Indirizzo logico

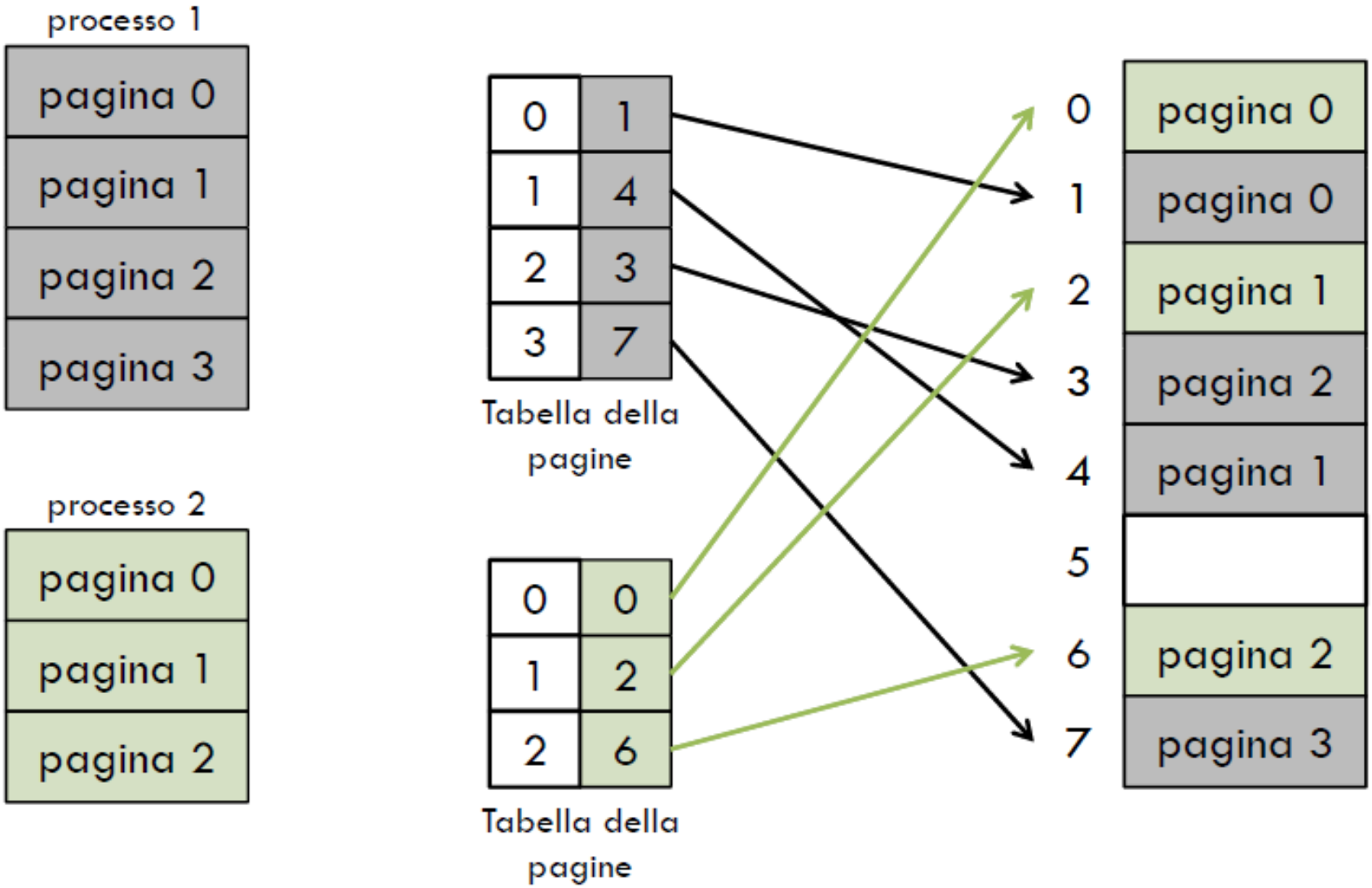
Paginazione: schema di traduzione dell'indirizzo (2)



Paginazione: un esempio

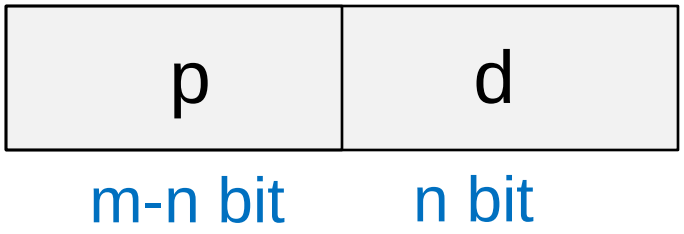


Paginazione: un altro esempio



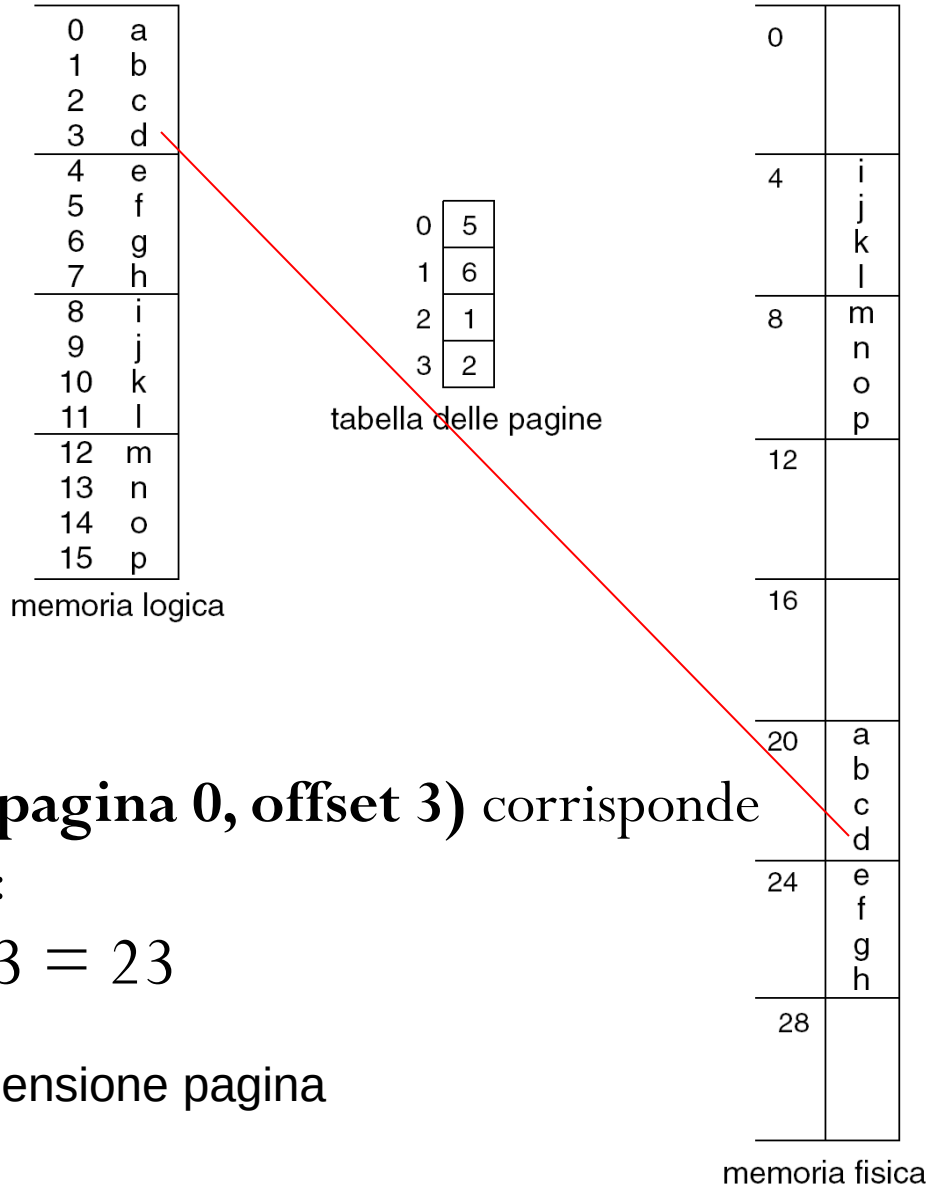
Paginazione: dimensione delle pagine

- La dimensione della pagina è definita dall'architettura del sistema ed **è in genere una potenza di 2**
  - perché semplifica la traduzione degli indirizzi
- La dimensione di una pagina **varia dai 512 byte ai 16MB**
- Se lo **spazio logico di indirizzamento è 2^m** e la **dimensione di pagina è 2^n unità**, allora:
  - m-n bit più significativi dell'ind. logico **indicano la pagina p**
  - n bit meno significativi dell'ind. logico **indicano l'offset di pagina d**



Indirizzo logico

Esempio di paginazione per una memoria centrale di 32 byte con pagine di 4 byte (8 pagine in totale)

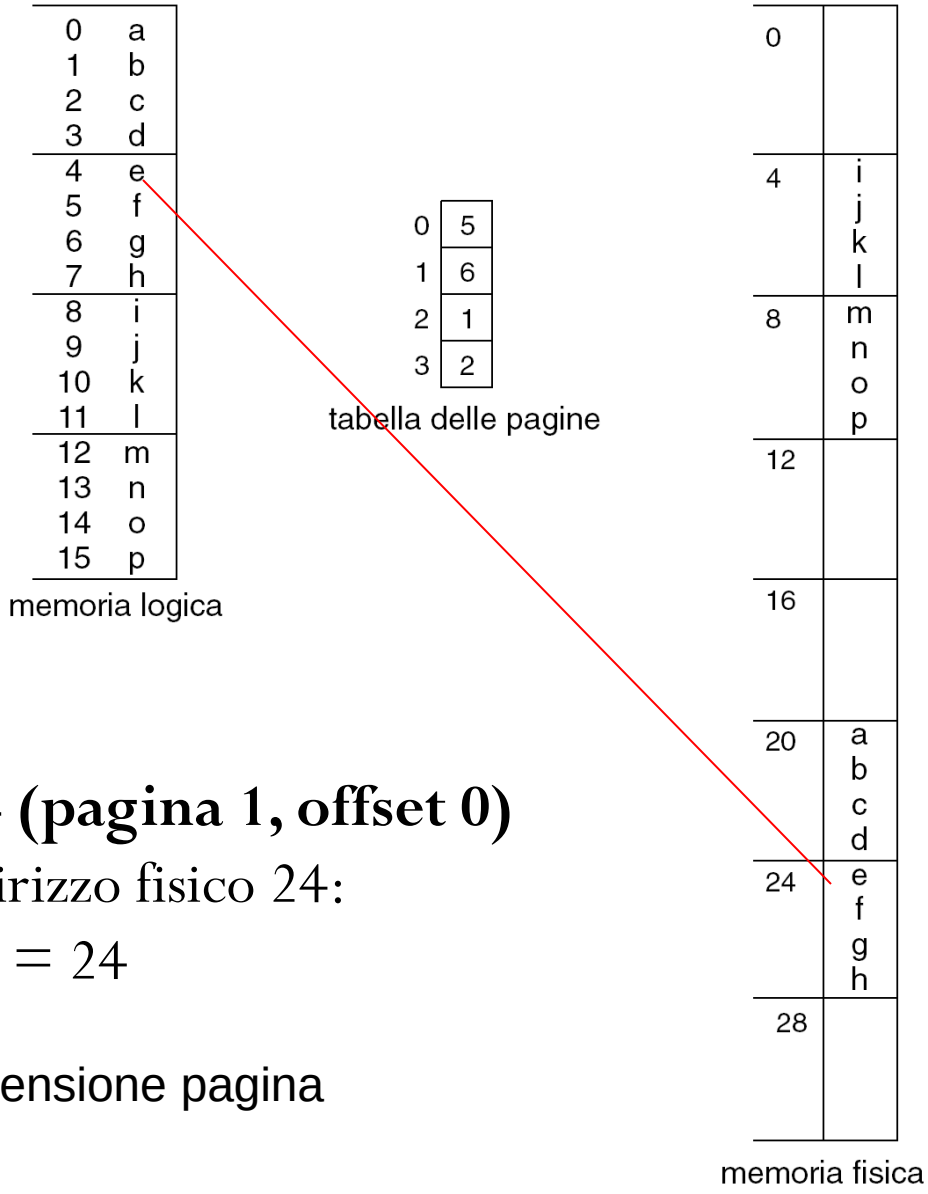


L'indirizzo logico **3 (pagina 0, offset 3)** corrisponde all'indirizzo fisico 23:

$$(5 \times 4) + 3 = 23$$

frame — dimensione pagina

Esempio di paginazione per una memoria centrale di 32 byte con pagine di 4 byte (8 pagine in totale)



L'indirizzo logico **4 (pagina 1, offset 0)** corrisponde all'indirizzo fisico 24:

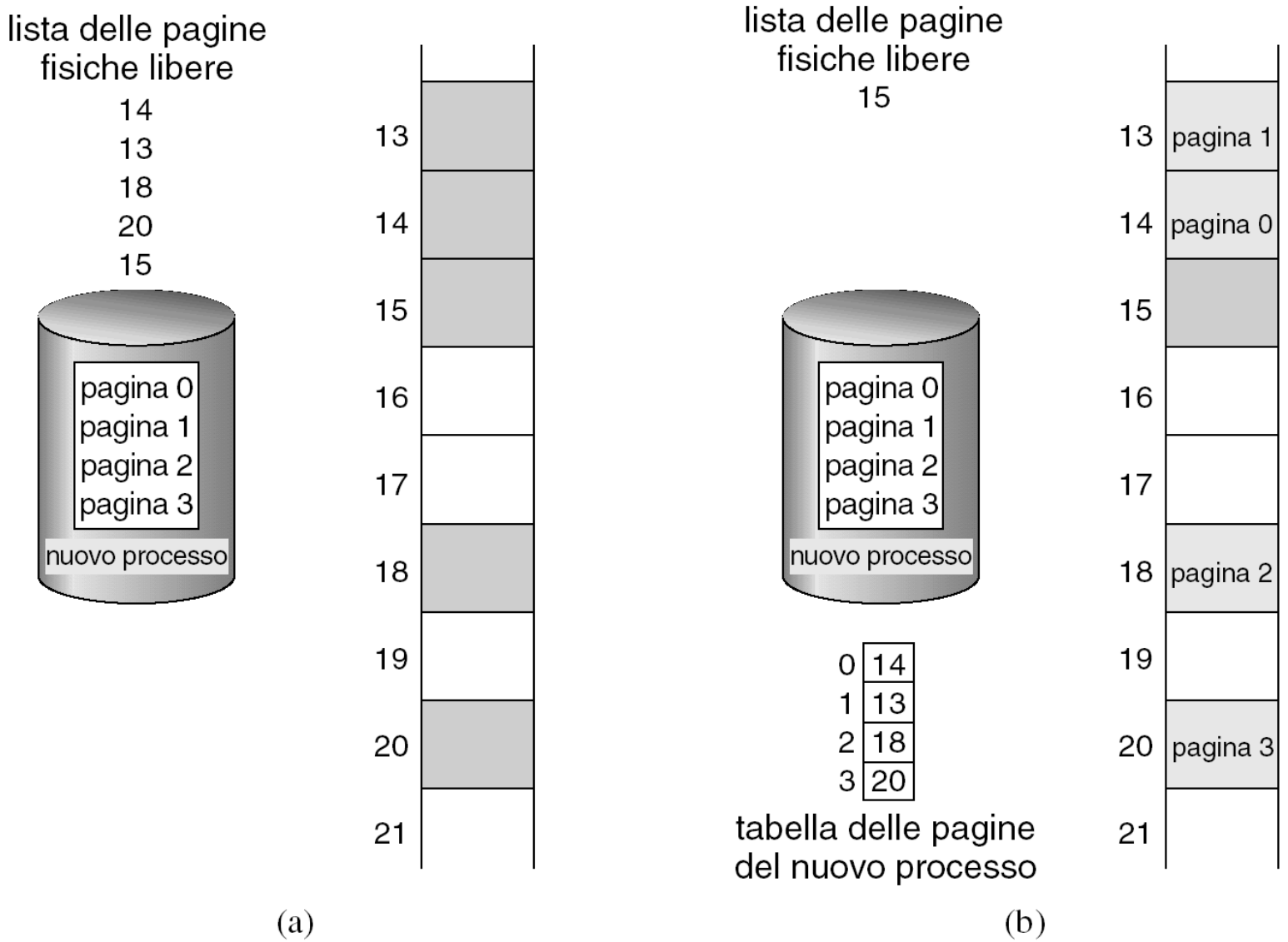
$$(6 \times 4) + 0 = 24$$

frame — dimensione pagina

Tabella dei frame

- Oltre alla tabella delle pagine (logiche) di ciascun processo, il SO mantiene un'unica **tabella dei frame** (pagine fisiche):
  - contiene **un elemento per ogni frame**, che indica se questo è libero o allocato,
  - e (se allocato) a quale pagina di quale processo o processi
- Quando un processo "entra" nel sistema, viene esaminata la sua dimensione espressa in numero di pagine! Se il processo richiede n pagine:
  - occorre trovare n frame liberi
    - anche sparpagliati in memoria
  - ed allocarli al processo aggiornando la funzione di mappatura nella tabella delle pagine del processo

Paginazione: frame liberi



Prima dell'allocazione

Dopo l'allocazione

Paginazione: vantaggi

- La paginazione implementa automaticamente una forma di **protezione dello spazio di indirizzamento**
  - Un programma può indirizzare solo i frame contenuti nella sua tabella delle pagine
- **No frammentazione esterna**
  - L'ultimo frame assegnato però potrebbe non essere completamente occupato causando **frammentazione interna**
    - Il caso peggiore si verifica quando un processo è un multiplo della dimensione della pagina + 1 byte (un frame è completamente sprecato per l'ultimo byte)
- **Pagine più grandi** causano maggiore frammentazione interna, ma permettono di avere **tabelle delle pagine più piccole** (meno occupazione di memoria)

Implementazione della tabella delle pagine

- Tipicamente, la tabella delle pagine è mantenuta nella memoria centrale
- **Il tempo per il context switch con la paginazione può aumentare in maniera considerevole**
  - In fase di context switch, il SO "carica" anche la tabella delle pagine per il processo a cui è stata assegnata la CPU
- Il vero **collo di bottiglia potrebbe essere l'accesso alla tabella della pagine** (PT)
  - Ogni indirizzo logico deve essere tradotto in un indirizzo fisico con un accesso alla PT – **accesso alla memoria rallentato di un fattore 2!**
- Il meccanismo di traduzione degli indirizzi DEVE essere reso molto efficiente

Tabella delle pagine: supporto HW (1)

- **Prima soluzione**: nel caso in cui il numero di pagine per ogni processo è piccolo, la PT può essere memorizzata all'interno di **appositi registri della CPU**
  - Ad *esempio*, il *PDP-11* usava 8 registri per la PT
  - Gli indirizzi fisici erano a 16 bit, per cui la memoria fisica aveva una dimensione di 64 KB (divisi in 8 frame da 8 KB)

- Negli attuali computer la tabella della pagine può avere un numero di elementi troppo grande (ad es. di 1 milione) per essere contenuta all'interno dei registri della CPU

Tabella delle pagine: supporto HW (2)

- **Due altre possibili soluzioni**:
  - **Registro di base** della tabella delle pagine *(pagetable base register, PTBR)* e **tabella delle pagine in RAM**
  - **Memoria associativa** (***Translation look-aside buffer,*** TLB) e **tabella delle pagine in RAM**

Tabella delle pagine: Registro di base della tabella delle pagine

In questa soluzione:

- Un ***registro base (*PTBR)** punta alla tabella delle pagine in memoria
  - Il cambio di contesto richiede così di cambiare solo questo registro
- **L'accesso alla memoria fisica è rallentato di un fattore 2**
  - In questo schema ogni accesso alla memoria (a dati/istruzione) richiede in verità due accessi alla memoria: uno per la tabella delle pagine e uno per i dati/istruzione

Tabella delle pagine: memoria associativa (TLB)

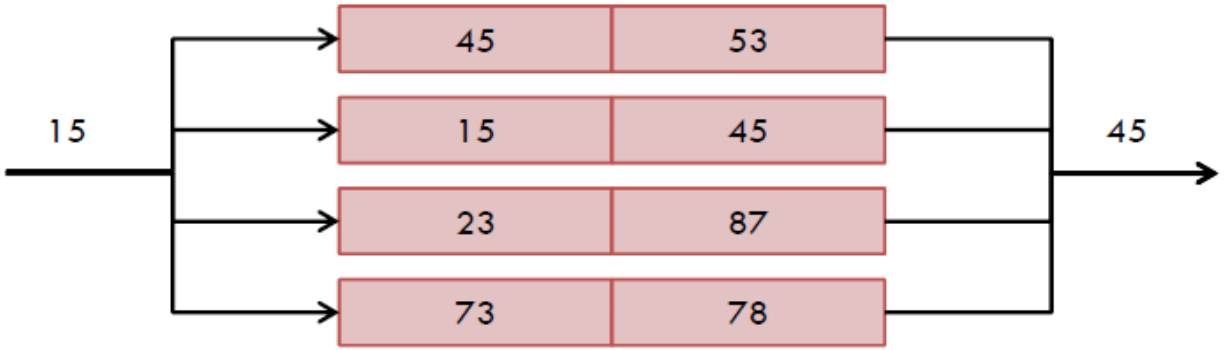
- I due problemi di accesso alla memoria della soluzione precedente possono essere risolti attraverso l'uso di **una speciale piccola cache per l'indicizzazione veloce** detta **memoria associativa** (o *translation look-aside buffers* – ***TLB***)

- Il dispositivo è molto veloce e costoso, e le dimensioni di una TLB sono ridotte (dai 64 ai 1024 elementi)

Tabelle delle pagine: Memoria associativa (TLB)

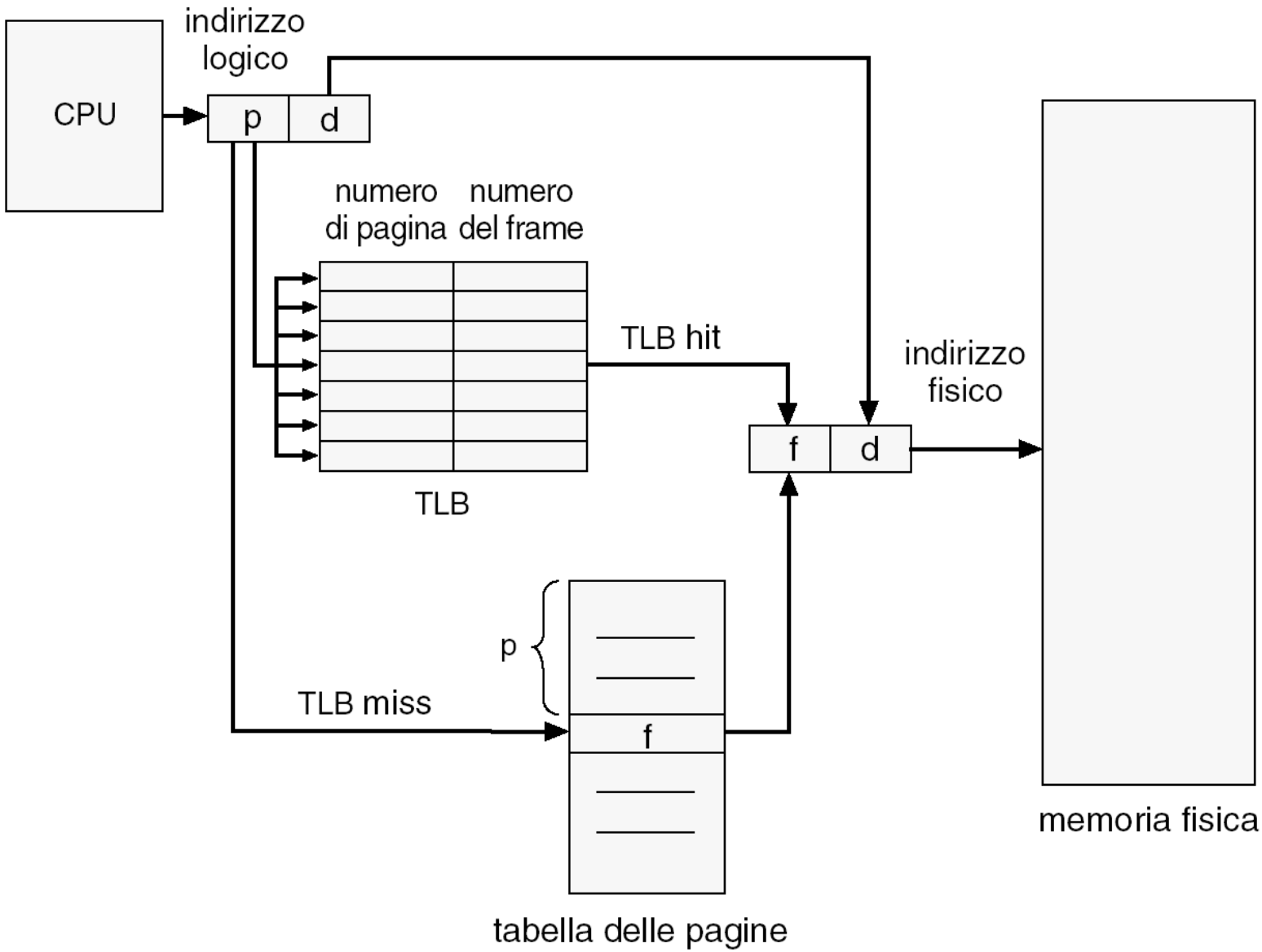
- Memoria associativa – ricerca parallela

coppia del tipo (*chiave = pagina #, valore = frame #)*



- Traduzione dell'indirizzo (A, A′)
  - Se A si trova nella memoria associativa, si ottiene il frame #: ricerca con successo (*TLB HIT*)
  - Altrimenti, ottiene il frame # dalla tabella delle pagine in RAM: mancato ritrovamento nel TLB (*TLB MISS*)

Hardware per la paginazione con la TLB



Tempo di accesso effettivo

- **Associative Lookup** (tempo di ricerca nella TLB) = **ε** unità di tempo
- **Ciclo di memoria** (tempo di accesso alla memoria) = **c** unità di tempo
- **Tasso di accesso con successo** (***hit ratio***) = **α** percentuale delle volte che un numero di pagina si trova nella TLB
- **Tempo di accesso effettivo (EAT)**

$$EAT = (c + \varepsilon)\,\alpha + (2c + \varepsilon)(1 - \alpha)$$

$$= 2c + \varepsilon - c\alpha$$

(pesatura con il calcolo delle probabilità)

Tempo di accesso effettivo: un esempio

- Associative Lookup ε = 20 ns
- Ciclo di memoria c = 100 ns
- Tasso di accesso con successo $\alpha = 0.80$ (80%)

$$\textbf{EAT} = 2c + \varepsilon - c\alpha = (200 + 20 - 80)\ \text{ns} = 140\ \text{ns}$$

Si ha un rallentamento del 40% del tempo di accesso alla memoria centrale (da 100 a 140 ns)

- **più alto è l'hit ratio α, più piccolo è il rallentamento**

Paginazione: protezione della memoria centrale

- La protezione della memoria centrale in ambiente paginato è ottenuta **mediante bit di protezione mantenuti nella tabella delle pagine**

- **Ad ogni elemento della tabella delle pagine**:
  - Un **bit di validità/non validità**
    - "valido" indica che il frame associato è nello spazio degli indirizzi logici del processo ed è quindi una pagina legale
    - "non valido" indica che il frame non è nello spazio degli indirizzi logici del processo
  - Un **bit di lettura-scrittura o sola lettura**
    - Indica se la pagina è in sola lettura

Bit di validità (v) o non validità (i) in una tabella delle pagine

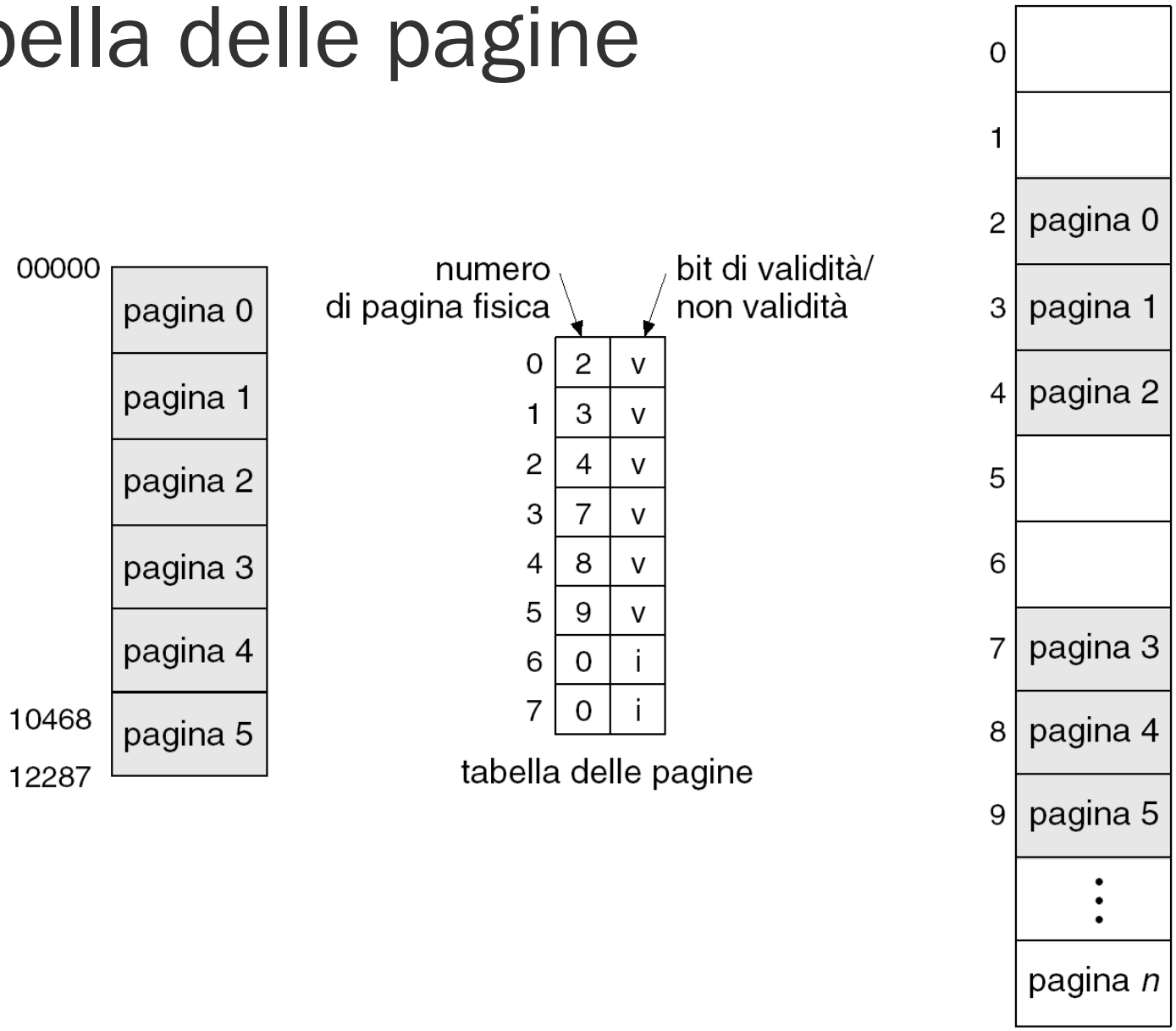


Tabelle delle pagine: altri bit

- **Bit presente/assente**
  - Indica se la pagina è in memoria centrale
- **Bit usata/non usata**
  - Serve nelle politiche di rimpiazzamento delle pagine
- **Bit modificata**
  - Utile quando la pagina deve essere eliminata dalla memoria centrale

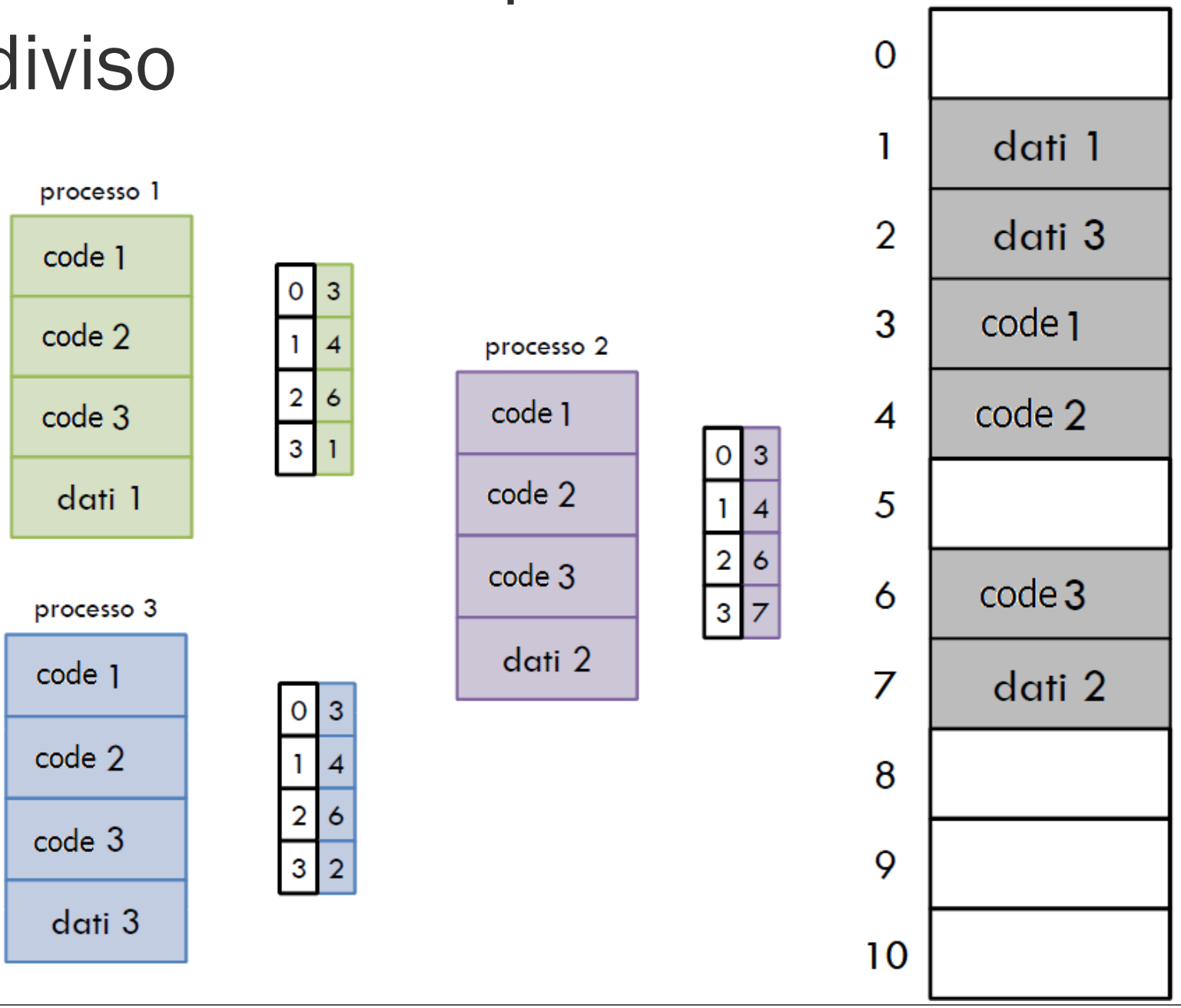
Entry della PT

P	U	M	Altri bit protezione	Frame #

Paginazione: pagine condivise

- **La paginazione facilita la condivisione di memoria tra processi**
- **Non tutte le pagine sono condivisibili!**
  - Le pagine dati tipicamente non sono condivisibili
- **Codice condivisibile:**
  - E' condivisibile il codice *read-only* o in genere *rientrante* (detto anche *puro*)
    - *Esempi*: una libreria oppure un editor di testi, un compitalote, ecc.. in un ambiente multiutente
- **Codice e dati private:**
  - Ogni processo possiede una copia separata del codice e dei dati

Paginazione: esempio di codice condiviso



Struttura dati per la tabella delle pagine

- **Paginazione gerarchica**: la tabella stessa è paginata, ovvero è in memoria virtuale e non in memoria reale!
  - e quindi può risiedere su memoria di massa

- **Tabella delle pagine invertita**

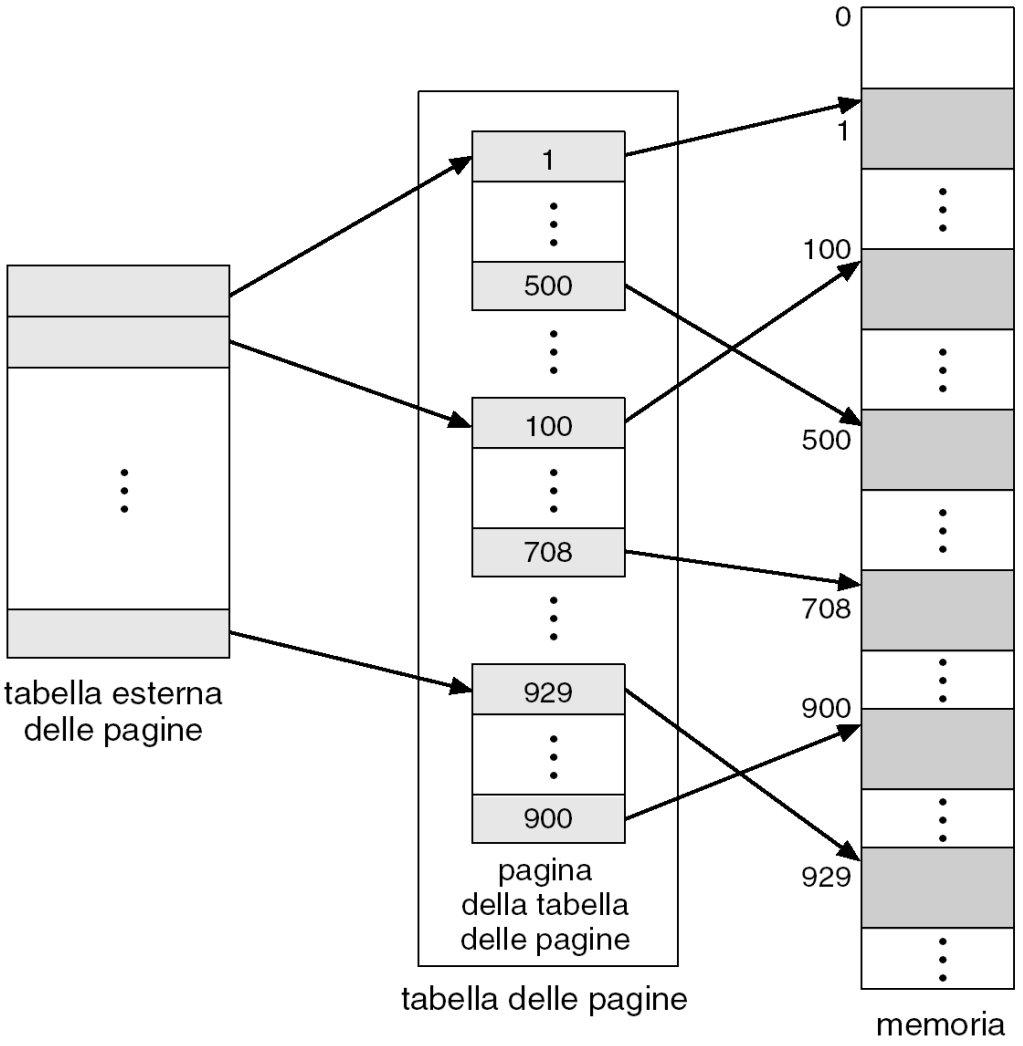
- **Tabelle delle pagine con hashing**: si prevede una *funzione hash* che prende come argomento la pagina e restituisce un frame *[non in programma]*

Paginazione gerarchica

- **Nei computer moderni** che supportano un vasto spazio di indirizzamento logico (da 2^{32} a 2^{64}), **la tabella delle pagine è eccessivamente grande**
  - Ad es. su una architettura a 32 bit con dimensione di pagina di 4 KB (2^{12}), la tabella può contenere fino a 1 milione di elementi ($2^{32} / 2^{12} = 2^{20}$ B = 1 MB = 1,048,576)

- **Una soluzione**: suddividere lo spazio degli indirizzi logici in più tabelle di pagine
- Una tecnica semplice è la **tabella delle pagine a due livelli**

Paginazione a due livelli



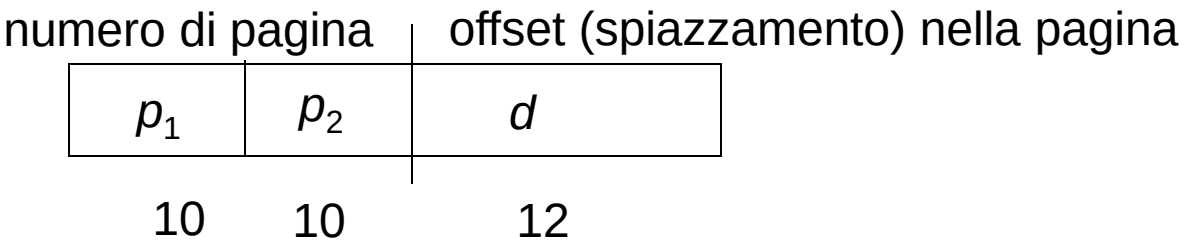
- Una directory delle pagine detta **tabella esterna**
  - ogni elemento punta ad una sottotabella delle pagine
- **Un insieme di sottotabelle delle pagine**
  - Tipicamente una sottotabella delle pagine è ampia quanto una pagina al fine di essere completamente contenuta in un frame

Esempio di paginazione a due livelli

- Un indirizzo logico su una macchina a 32-bit con pagine di 4K è diviso in: un numero di pagina di 20 bit e un offset di 12 bit

- Poichè **paginiamo la tabella in due livelli**, il numero di pagina è ulteriormente diviso in :
  - un numero di pagina p_1 da 10-bit
  - uno spiazzamento p_2 nella pagina da 10-bit

e un **indirizzo logico** è diviso come segue:



dove p_1 è un indice nella **tabella esterna**, e p_2 rappresenta lo spostamento all'interno della pagina della tabella esterna

Paginazione a due livelli: schema di traduzione dell'indirizzo

- Traduzione dell'indirizzo per un architettura di paginazione a due livelli a 32 bit (ad es. nel Pentium, VAX,ecc..)
- Nota anche come ***tabella delle pagine mappata in avanti*** (*forward-mapped page table*)

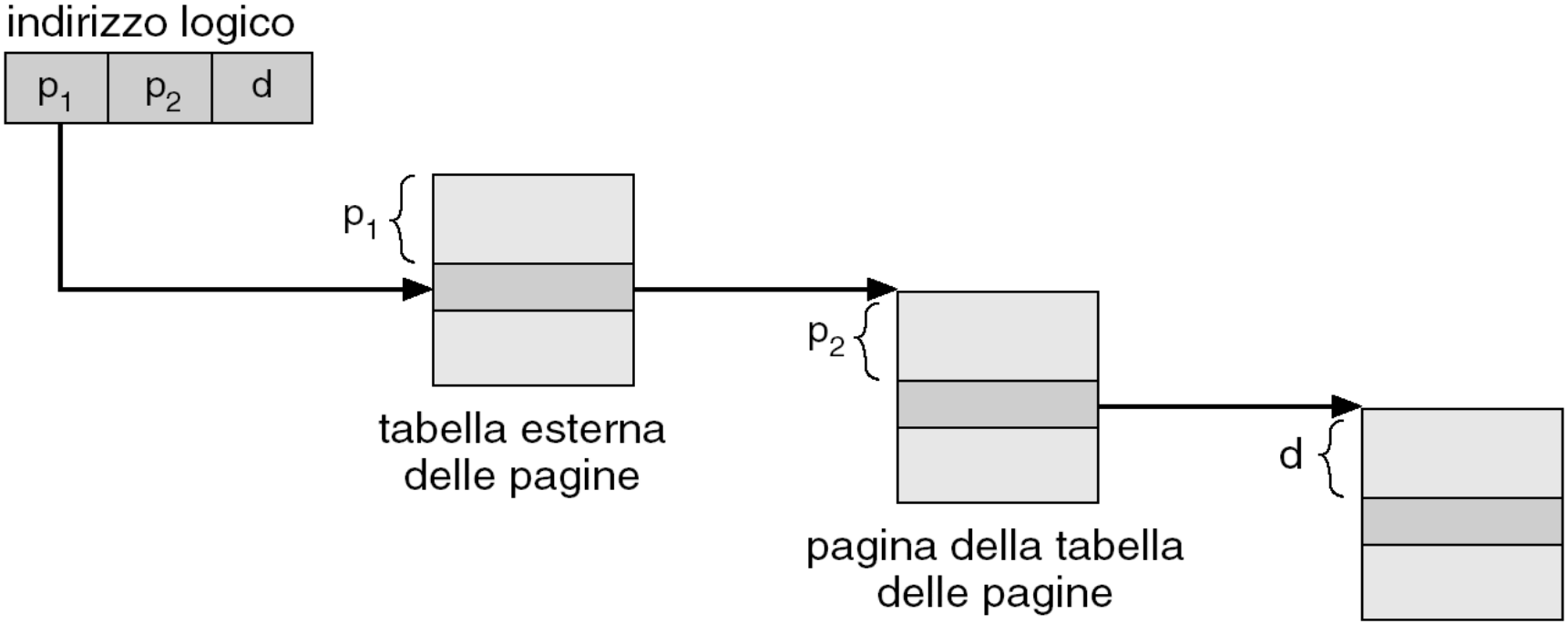


Tabelle a più livelli

- Se lo spazio logico è di 64 bit la paginazione a due livelli non è più sufficiente: **occorre paginare anche la tabella esterna**!
  - Nelle architetture SPARC a 32 bit si utilizza un schema a 3 livelli
  - La CPU a 32 bit Motorola 68030 ha una PT a 4 livelli

- **4 livelli non sono ancora sufficienti per architetture a 64 bit**
  - L'architettura UltraSPARC a 64 bit richiederebbe 7 livelli di paginazione
  - se la pagina cercata non è nel TLB, la traduzione impiega troppo tempo!

- Nei sistemi UltraSPARC a 64 bit si utilizza la tecnica della **tabella delle pagine invertite**

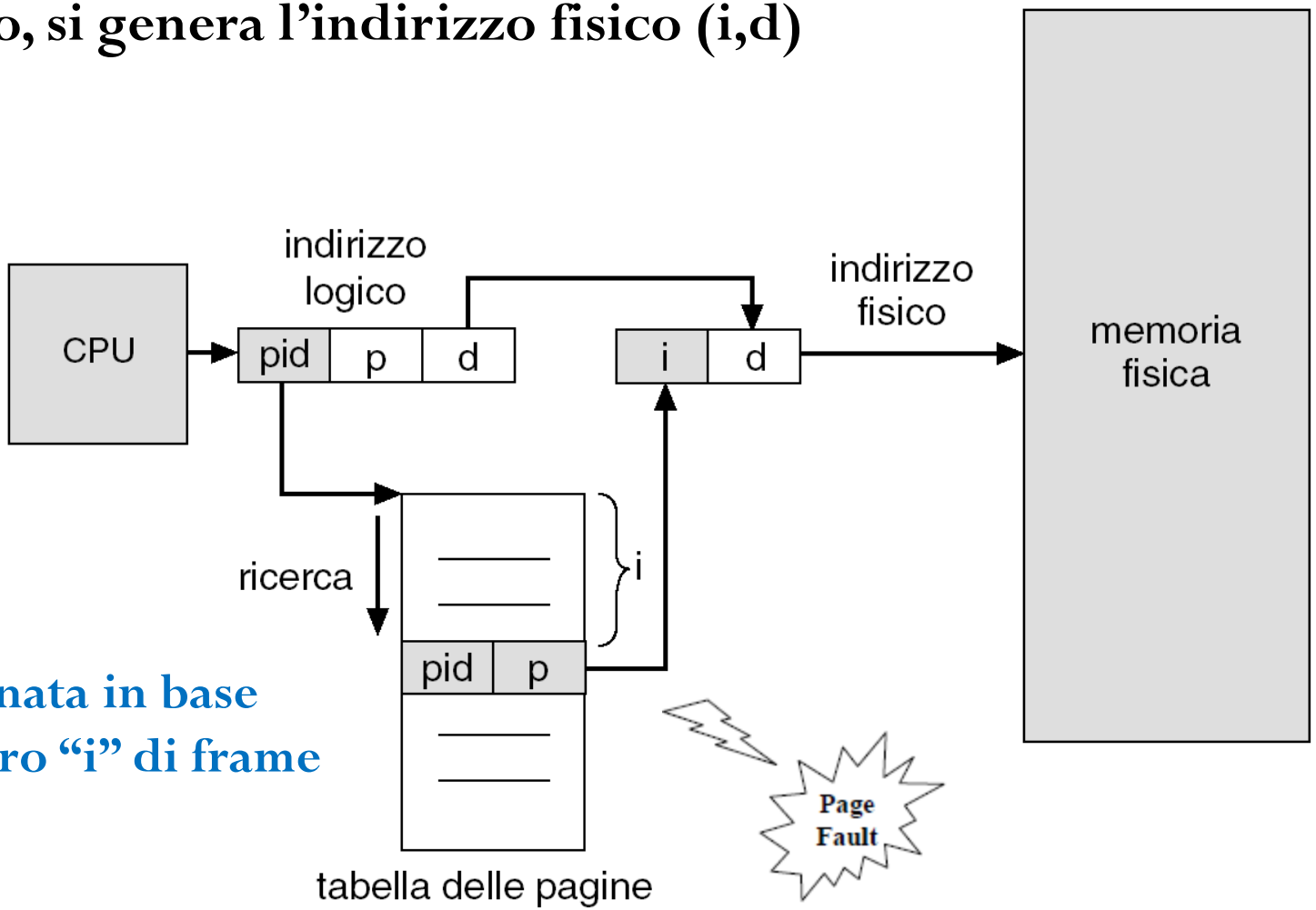
Tabella delle pagine invertita (IPT)

- Una IPT descrive l'occupazione dei frame della memoria fisica con **una entry per ogni frame**, quindi:
  - **C'è una sola IPT per tutto il sistema** (anziché una PT per processo)
  - La dimensione della IPT dipende strettamente dalla dimensione della memoria primaria
  - **L'indice di ogni elemento della IPT corrisponde al numero del frame corrispondente**
- Ogni **entry della IPT** contiene **una coppia**

  (process-id, page-number)
  - process-id: identifica il processo che possiede la pagina
  - page-number: indirizzo logico della pagina contenuta nel frame corrispondente a quella entry

Architettura della tabella delle pagine invertita

Si cerca nella IPT la coppia (pid, p), se la si trova all'i-esimo elemento, si genera l'indirizzo fisico (i,d)



IPT ordinata in base al numero "i" di frame

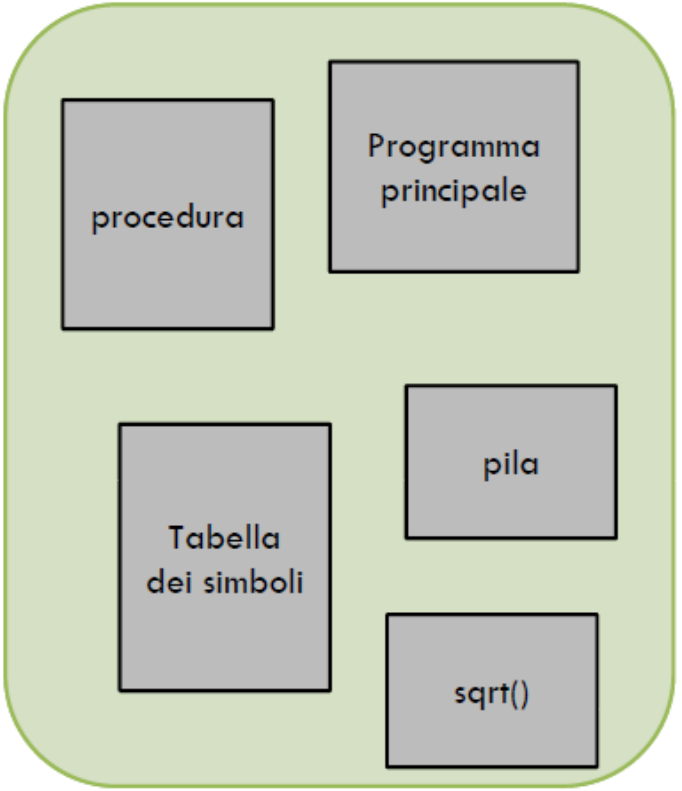
Tabella delle pagine invertita (IPT): Vantaggi e Svantaggi

- Con questo schema **si risparmia spazio, ma si perde in efficienza per cercare nella IPT** l'entry che contiene la coppia (*pid, p)*

- **Devono essere usate delle memorie associative** che contengono una porzione della IPT per velocizzare la maggior parte degli accessi

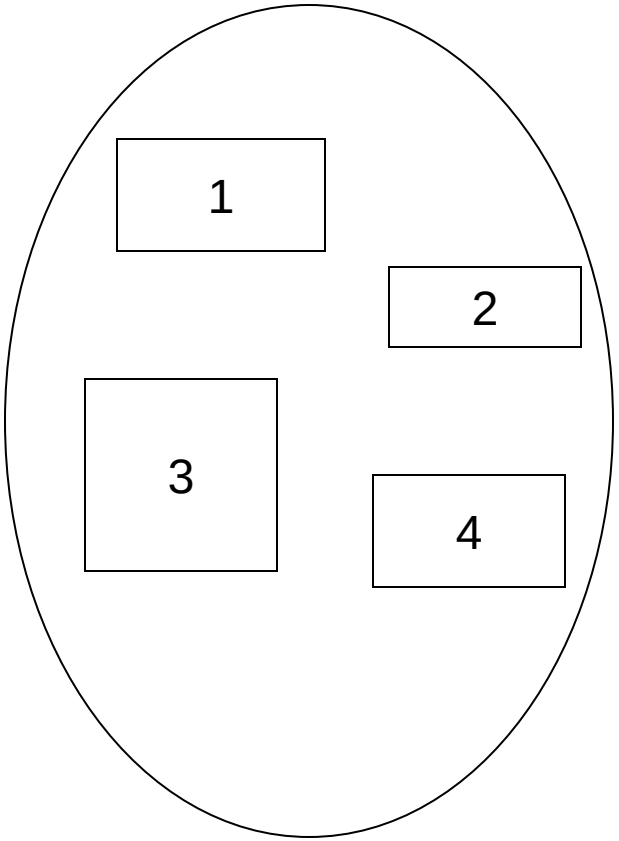
Segmentazione

- Invece di dividere la memoria in blocchi anonimi…
- **Un programma** può essere visto come **una collezione di diverse entità** così come percepite dall'utente
  - Codice, dati statici, dati locali alle procedure, stack…



- Il compilatore può costruire il codice oggetto in modo da rispecchiare questa ripartizione
- Ogni entità viene caricata dal loader separatamente in **aree di dimensione variabile** dette ***segmenti***

Vista logica della segmentazione



Spazio dell'utente

Spazio della memoria fisica

I segmenti hanno dimensione variabile e sono allocati (in modo non contiguo) all'interno della memoria fisica

Segmentazione pura

- Lo spazio di indirizzamento logico è **un insieme di segmenti di dimensione variabile**
- **Un segmento è un'unità logica con contenuti "omogenei"**
- **I segmenti sono generati automaticamnete dal compilatore**; ad es. un compilatore C crea i segmenti:
  - Il codice
  - Variabili globali
  - Heap, da cui si alloca la memoria
  - Variabili locali statiche di ogni funzione o procedura
  - Librerie standard del C
- **Ogni segmento è caratterizzato dalla sua dimensione**
- **Ogni indirizzo logico consiste di due parti**

(# segmento s, *offset d*)

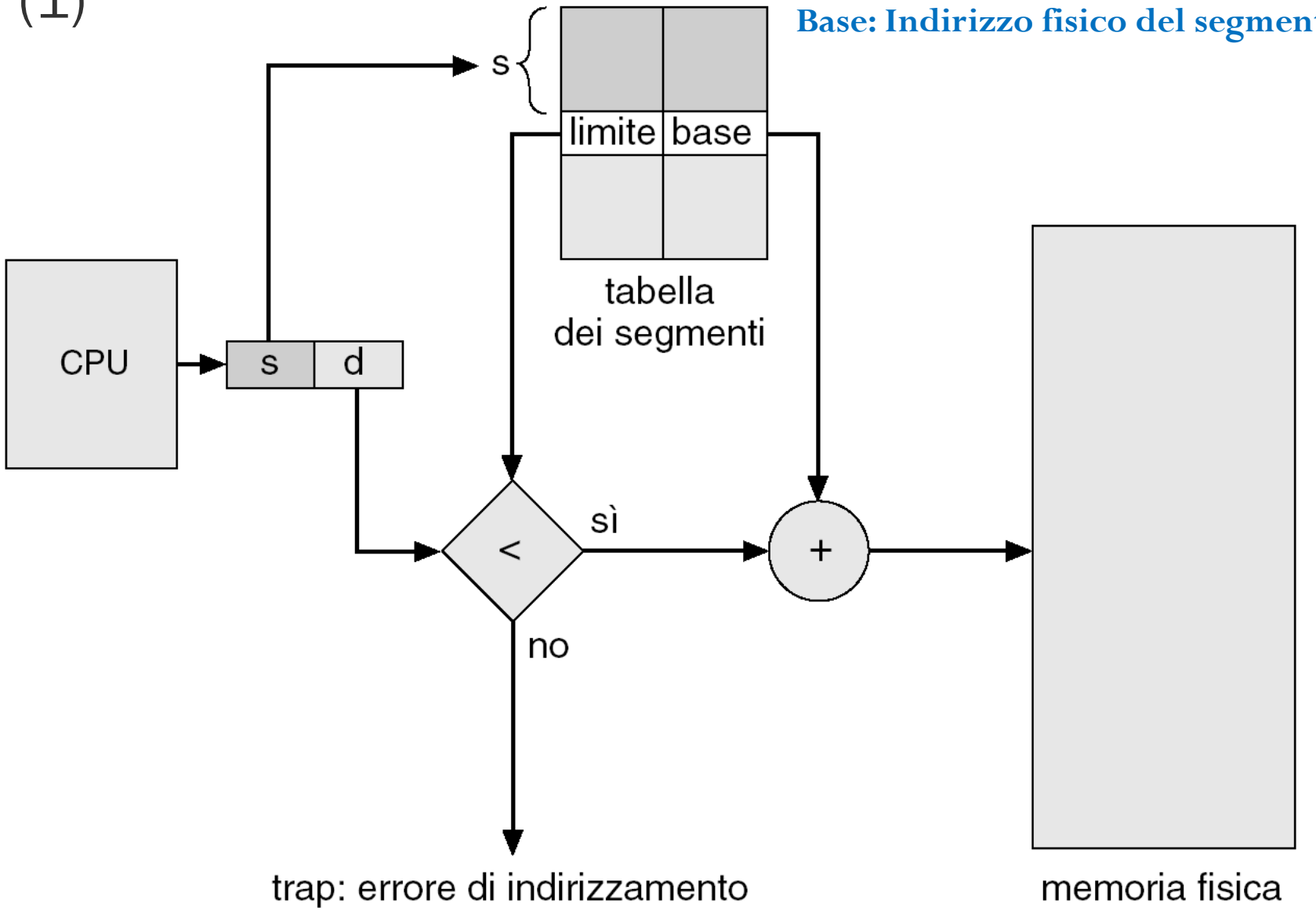
Architettura della segmentazione pura

- ***Tabella dei segmenti*** – mappa gli indirizzi logici in indirizzi fisici
- **Ogni elemento della tabella ha**:
  - Una ***base*** – contiene l'indirizzo fisico di partenza in cui il segmento risiede in memoria centrale
  - Un ***limite*** – specifica la lunghezza del segmento stesso

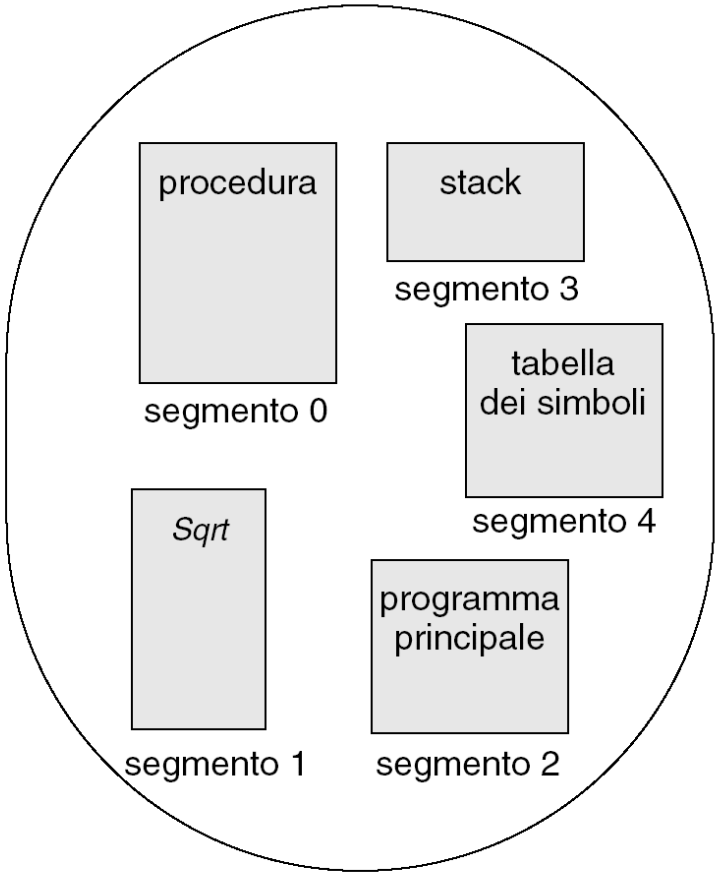
Hardware per la segmentazione pura (1)

Limite: Dimensione del segmento

Base: Indirizzo fisico del segmento

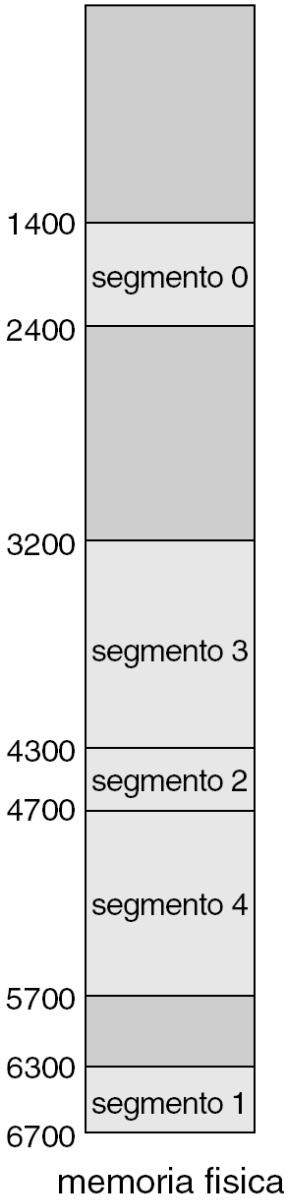


Esempio di segmentazione pura



	limite	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

tabella
dei segmenti



A cosa corrisponde l'indirizzo logico <2,53>?
E l'indirizzo <0,1222>?

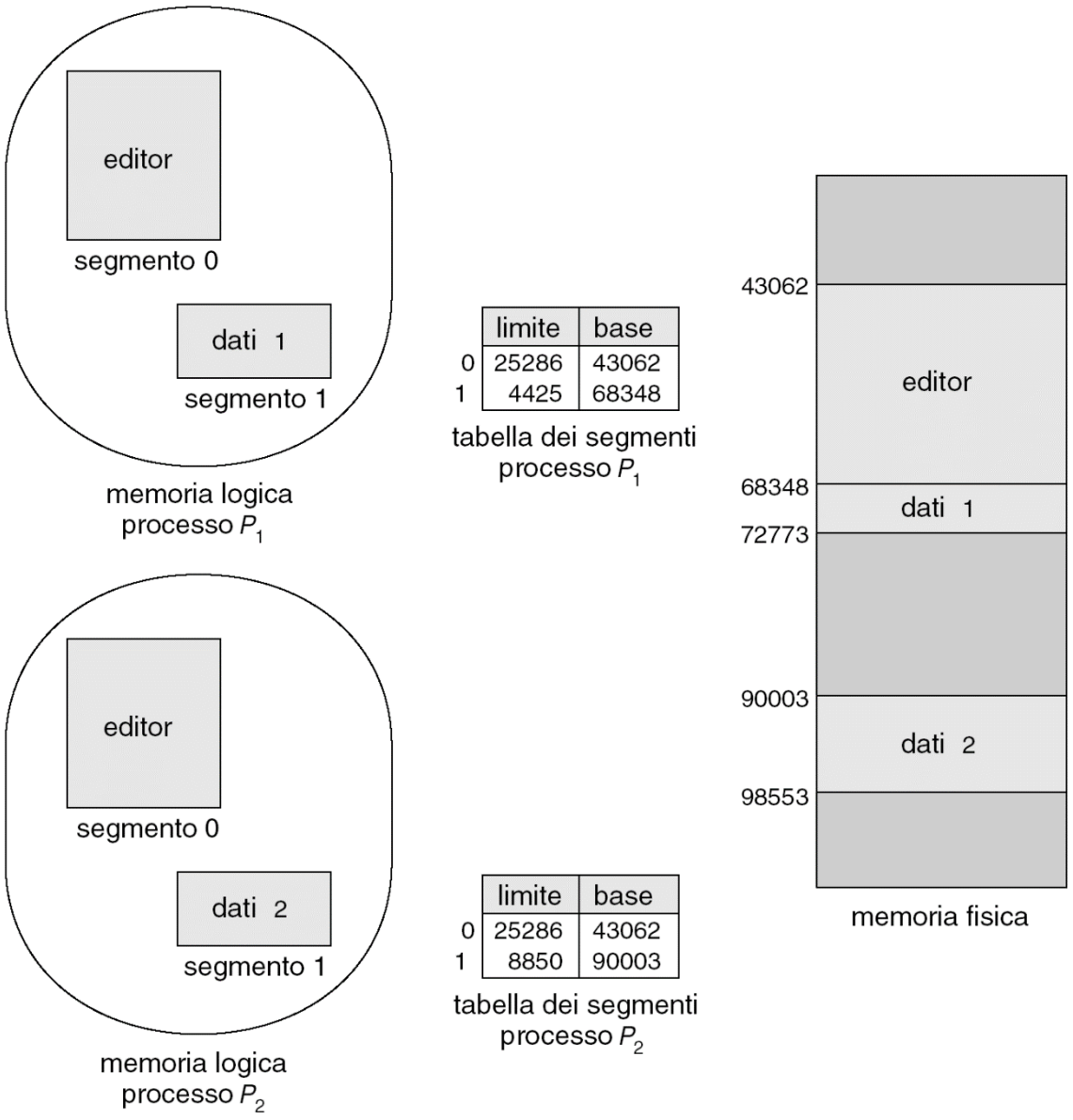
Hardware per la segmentazione pura
(2) – implementazione della tabella dei segmenti

- Se la tabella dei segmenti è piccola, può stare nei **registri della macchina**
- Solitamente, si utilizzano **due registri speciali**:
  - ***Registro base del segmento dalla pagina (STBR)*** punta alla tabella dei segmenti che si trova in memoria fisica
  - ***Registro della lunghezza del segmento della pagina (STLR)*** indica il numero dei segmenti usati dal programma; il numero del segmento s è legale se $s <$ STLR
- E' possibile utilizzare un **meccanismo a memoria associativa** per limitare l'overhead

Architettura della segmentazione pura (Cont.)

- Allocazione:
  - first fit/best fit
  - **frammentazione esterna** (un segmento potrebbe non trovare spazio in RAM, anche se lo spazio c'è…)
- Rilocazione:
  - dinamica
  - attraverso una tabella di segmenti
- Condivisione:
  - A livello di segmenti
- Protezione. Ogni bit di protezione è associato ad un segmento:
  - bit di validità = 0 $\Rightarrow$ segmento illegale
  - privilegi di lettura/scrittura/esecuzione

Condivisione dei segmenti



Segmentazione e paginazione

- La segmentazione è una soluzione più "naturale" della paginazione, ma soffre degli stessi problemi (seppure mitigati) dell'allocazione contigua a partizioni variabili
  - Il problema della **frammentazione esterna**

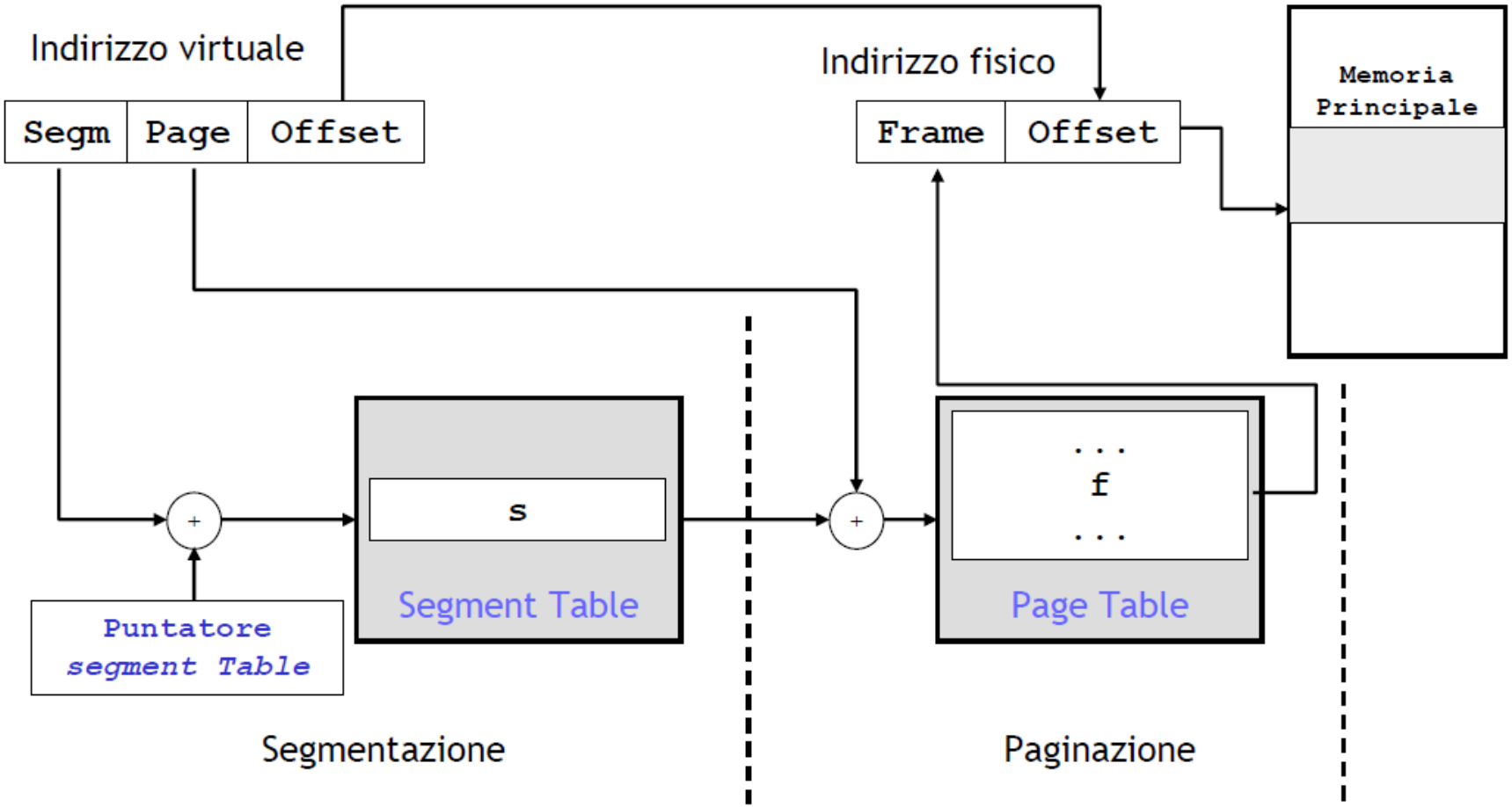
- **IDEA**: si possono paginare i segmenti mantenendo i vantaggi della segmentazione: **segmentazione con paginazione**

- Usata in origine nel sistema MULTICS

- Queste tecniche combinate vengono usate nella maggior parte dei sistemi operativi moderni

Segmentazione con paginazione

- Le informazioni della tabella dei segmenti non contengono l'indirizzo di un segmento,
- ma *l'indirizzo della tabella delle pagine per quel segmento!*



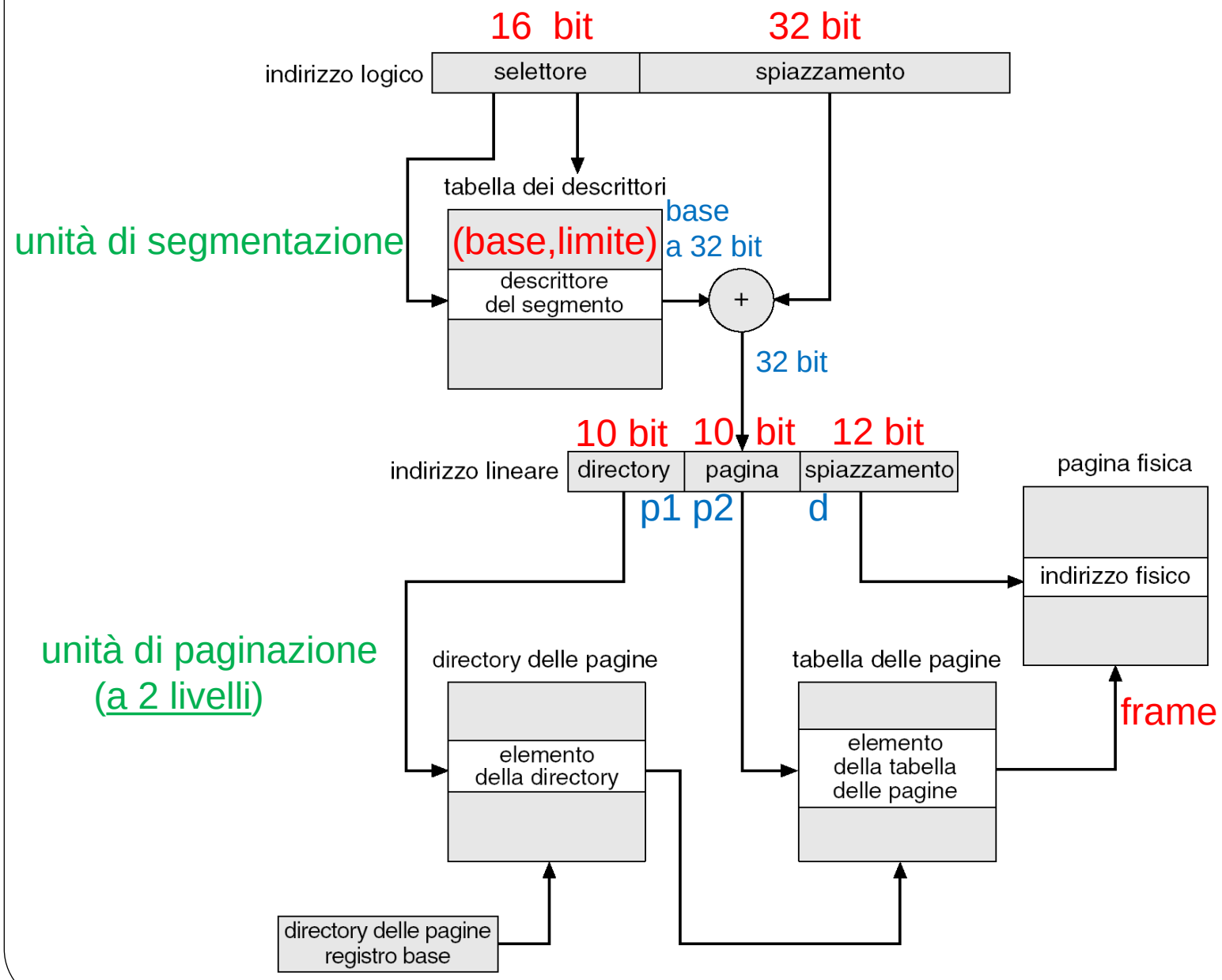
Segmentazione con paginazione: vantaggi

- La segmentazione con paginazione unisce i vantaggi dei due approcci
- **Vantaggi della paginazione**:
  - Trasparente al programmatore
  - Elimina la frammentazione esterna
- **Vantaggi della segmentazione**:
  - Modulare
  - Supporto per la condivisione e protezione

Segmentazione con paginazione – esempio Intel 80x86

- Lo spazio di indirizzamento fisico è a 32 bit, ed è quindi possibile indirizzare al **massimo 4GB di memoria** (*dimensione max di un segmento*)
- Il **numero max di segmenti per processo è di 16 KB**
  - e ogni segmento può arrivare a 4 GB
- **Ogni segmento è paginato**
- Il SO può scegliere tra **pagine da 4 Mbyte o da 4 Kbyte**
- **In caso di 4Kbyte**
  - Una tabella delle pagine può pertanto contenere fino a 1 milione CIRCA di elementi (4 GB / 4 KB = 1 MB)
  - Il Pentium adotta uno **schema di paginazione a due livelli**
    - **Tabella esterna** (**directory** delle pagine) + **tabella interna** delle pagine

Intel 80x86 traduzione dell'indirizzo



Alcune conclusioni

- Diverse sono le tecniche adoperate per la gestione della memoria centrale: alcune semplici altre complesse

- **Il supporto hardware è fondamentale** per:
  - determinare la classe di tecniche usabili
  - migliorare l'efficienza dei diversi approcci

- Le varie tecniche cercano di **aumentare il più possibile il livello di multiprogrammazione**
  - permettono lo swapping e la rilocazione dinamica del codice
  - limitano la frammentazione
  - favoriscono la condivisione del codice fra i diversi processi