SISTEMI OPERATIVI

(MODULO DI INFORMATICA II)

Gestione della memoria centrale

Prof. Luca Gherardi

Prof.ssa Patrizia Scandurra (anni precedenti)

Università degli Studi di Bergamo

a.a. 2012-13

Sommario

- Introduzione del problema
- Il gestore della memoria
- Concetti generali
- Swapping
- Allocazione contigua di memoria
- Paginazione
- Segmentazione
- Segmentazione con paginazione

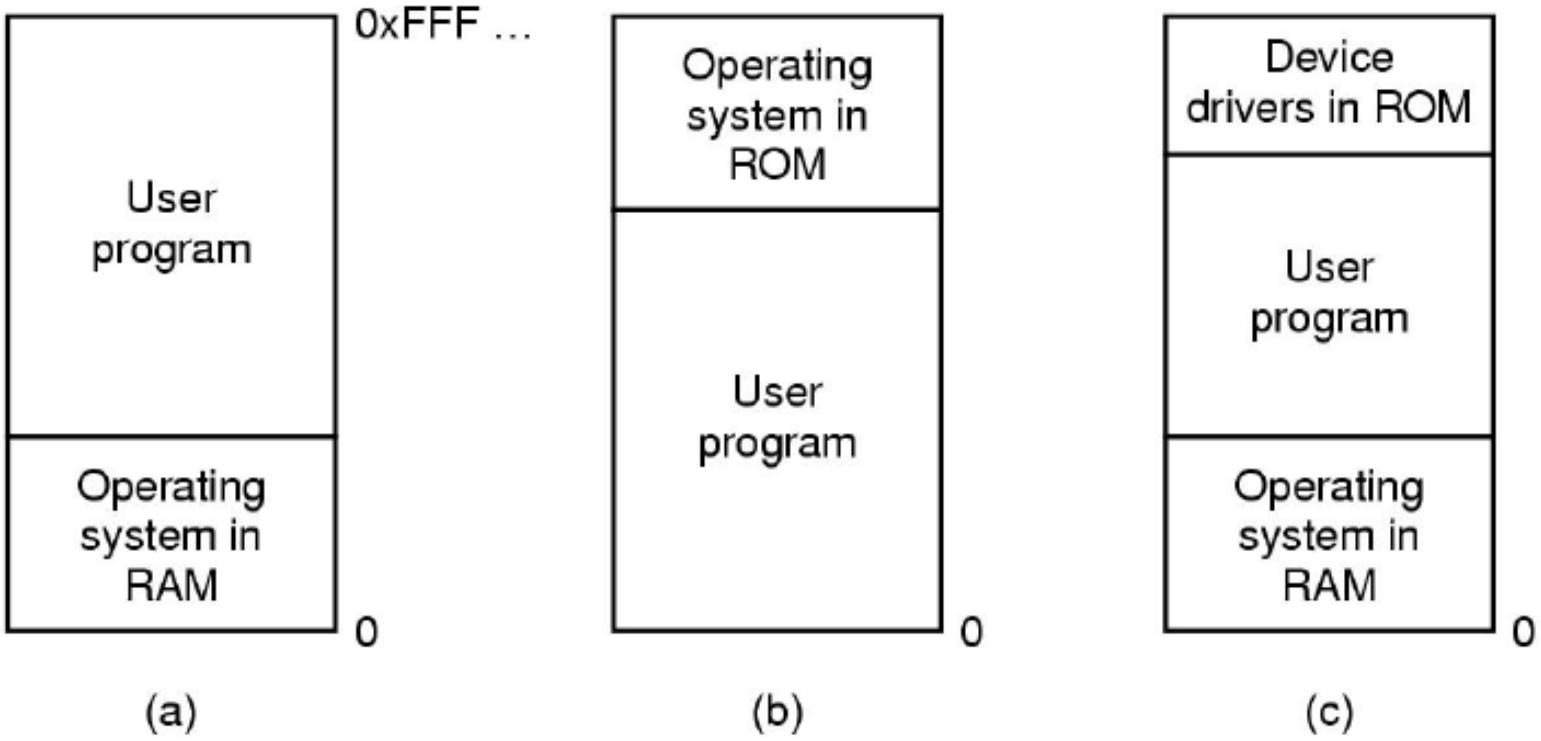
Attivazione di un programma

- 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** 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)



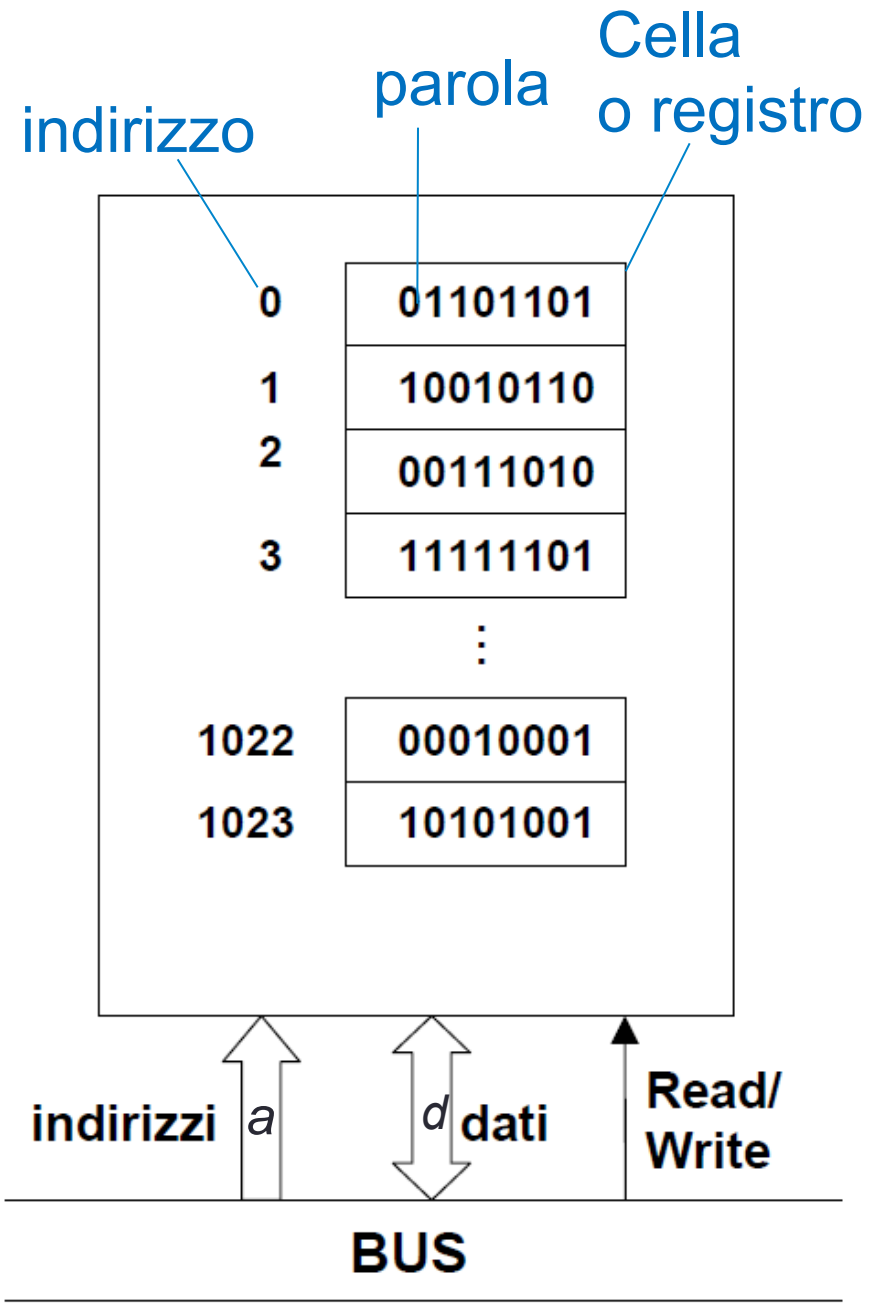
Sistemi multiprogrammati

- 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

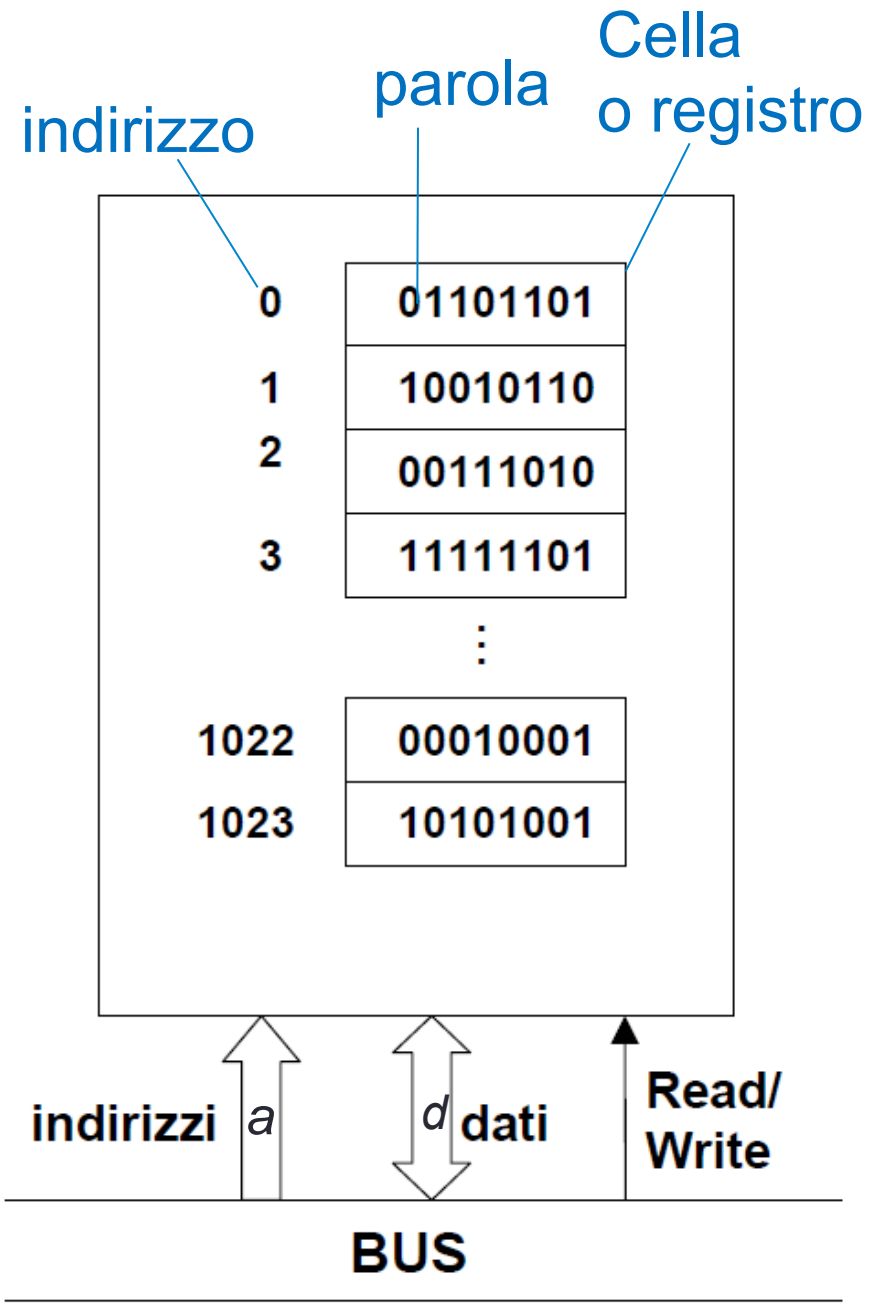
Memoria centrale

- Consiste in un ampio vettore di **parole**, ciascuna con il proprio **indirizzo**
- Una istruzione o un dato possono occupare più celle consecutive
- Tipico flusso di esecuzione di un'istruzione: prelevamento dalla memoria dell'istruzione, decodifica (eventuale prelevamento di altre istruzioni), esecuzione, eventuale salvataggio dei risultati
- Contenuto delle celle non riconoscibile
  - La memoria vedo solo parole e indirizzi ma non sa come essi siano generati (nemmeno se siano dati o istruzioni)



Memoria centrale

- 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
- *a* è la dimensione in bit degli indirizzi



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 processi

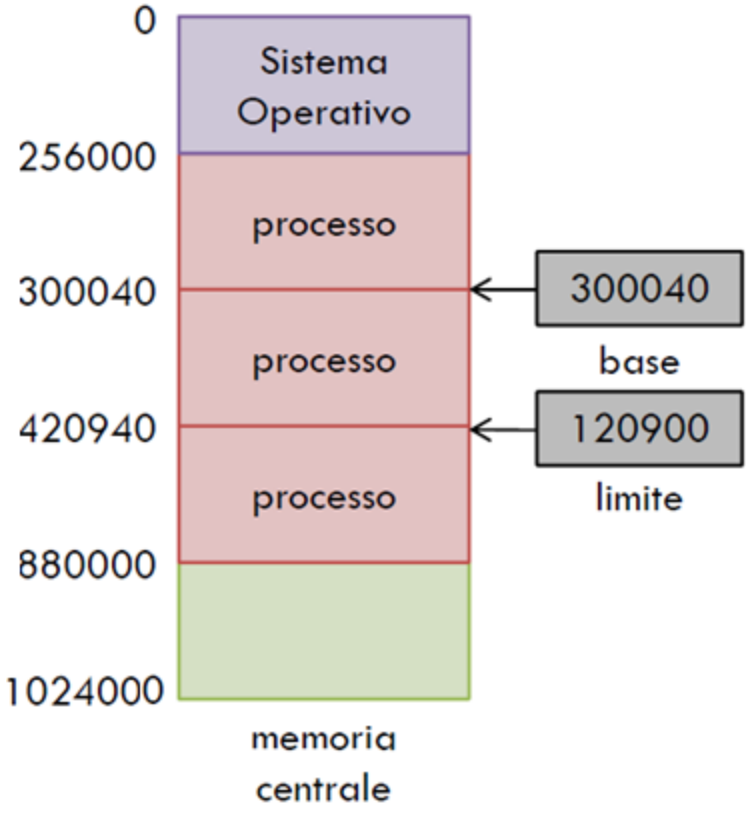
- Funzioni principali:
  - Allocazione
  - Protezione
  - Condivisione controllata
  - Sfruttamento delle *gerarchie di memoria*

Concetti generali

- Concetti generali che verranno affrontati:
  - 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

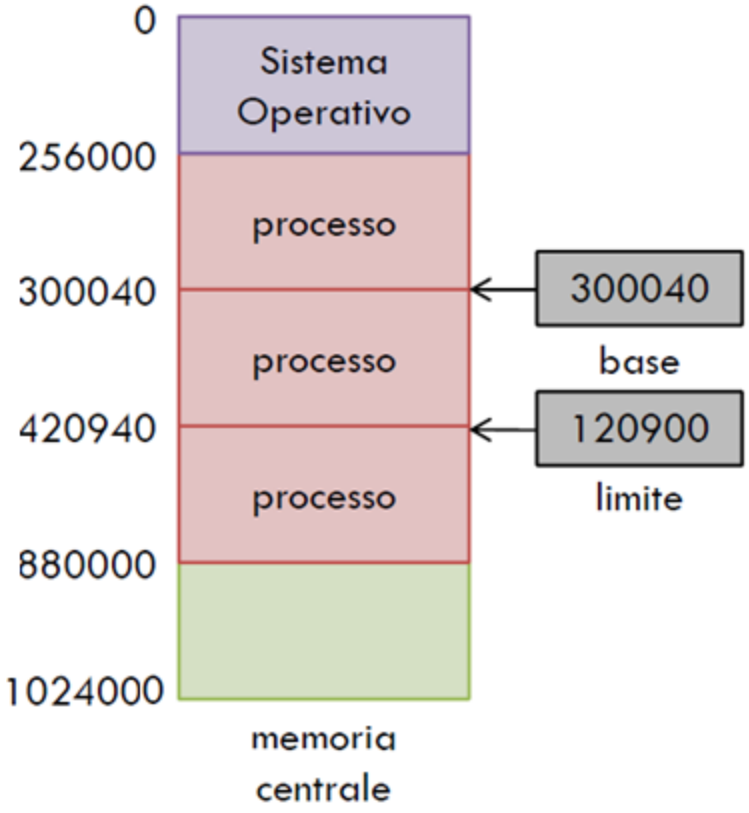
Separazione degli spazi di indirizzamento

- È necessario garantire che ogni processo acceda solo alla sua area di memoria (a meno di condivisioni volute)
- Si utilizzano 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



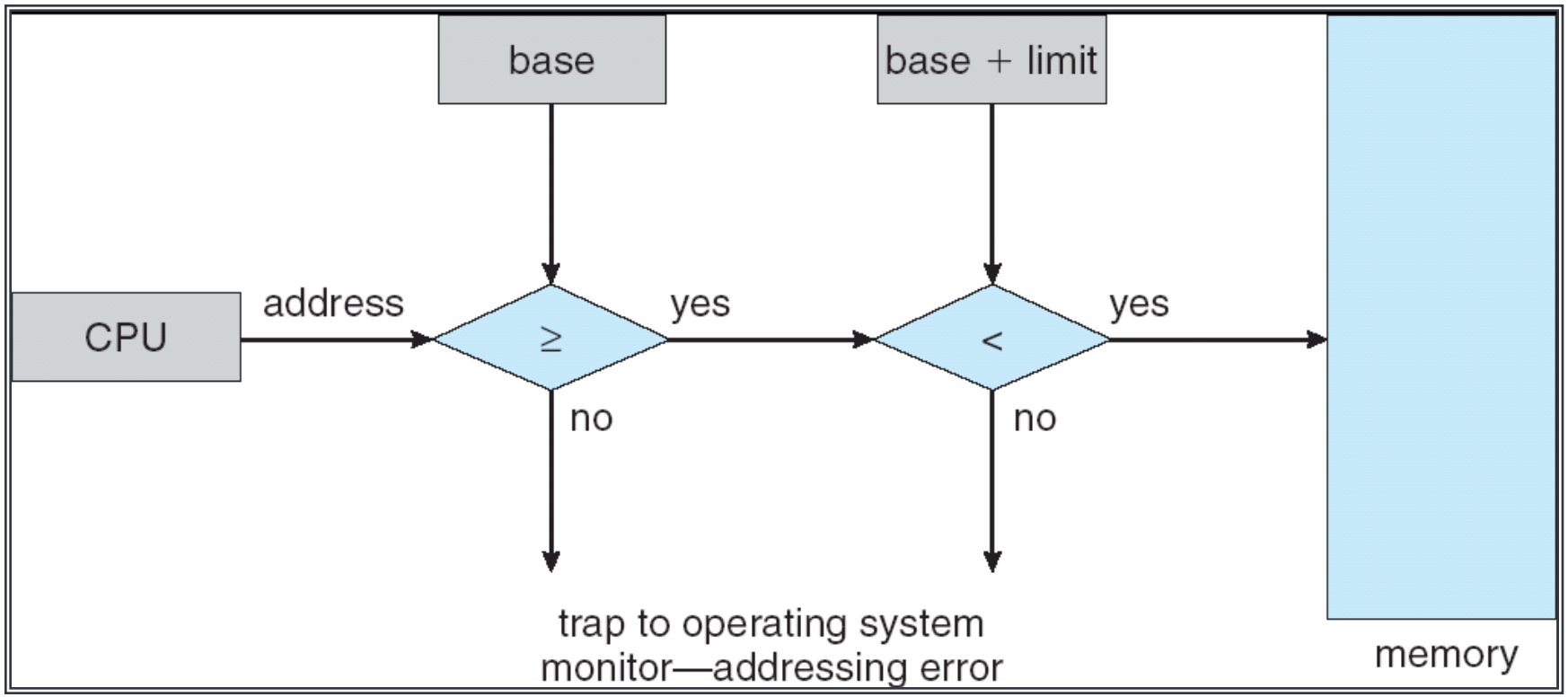
Separazione degli spazi di indirizzamento

- Quando un processo utente cerca di accedere ad un indirizzo la CPU lo confronta con i valori dei due registri
- Ogni tentativo da parte di un processo utente di accedere ad un'area non concessa della memoria porta ad un'eccezione
  - Il controllo viene restituito al SO
  - Passaggio da modalità utente a modalità Kernel
- Il SO non ha limiti sulle aree di memoria a cui può accedere



Protezione della memoria

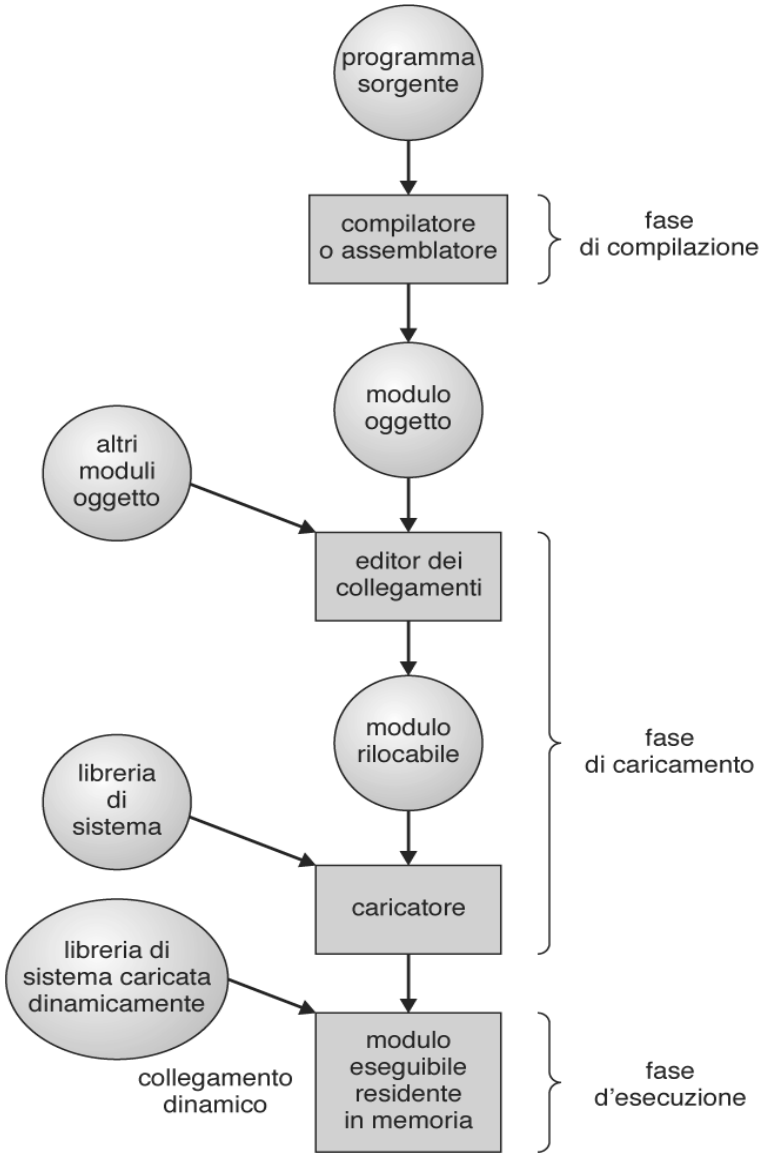
- Per assicurare che non ci siano accessi illegali in memoria, la CPU confronta ogni indirizzo generato dal processo con i valori contenuti nel registro base e nel registro limite



Indirizzamento (1)

- La CPU esegue programmi utente che elaborano insiemi di dati in base ad una sequenza di istruzioni
- I programmi utente passano attraverso più **stadi** prima di essere eseguiti, e in tali stadi gli indirizzi cambiano la loro **rappresentazione**
- Esisto diversi spazi degli indirizzi:
  - **Implementazione**: gli indirizzi sono simbolici (e.g. contatori)
  - **Compilazione**: il compilatore associa gli indirizzi simbolici ad indirizzi relativi (binding)
    - Esempio di indirizzo relativo: *n* byte dall'inizio di un determinato modulo
  - **Caricamento**: il linkage editor o il loader trasformano gli indirizzi relativi in indirizzi assoluti

Indirizzamento (2)



Indirizzamento (3)

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

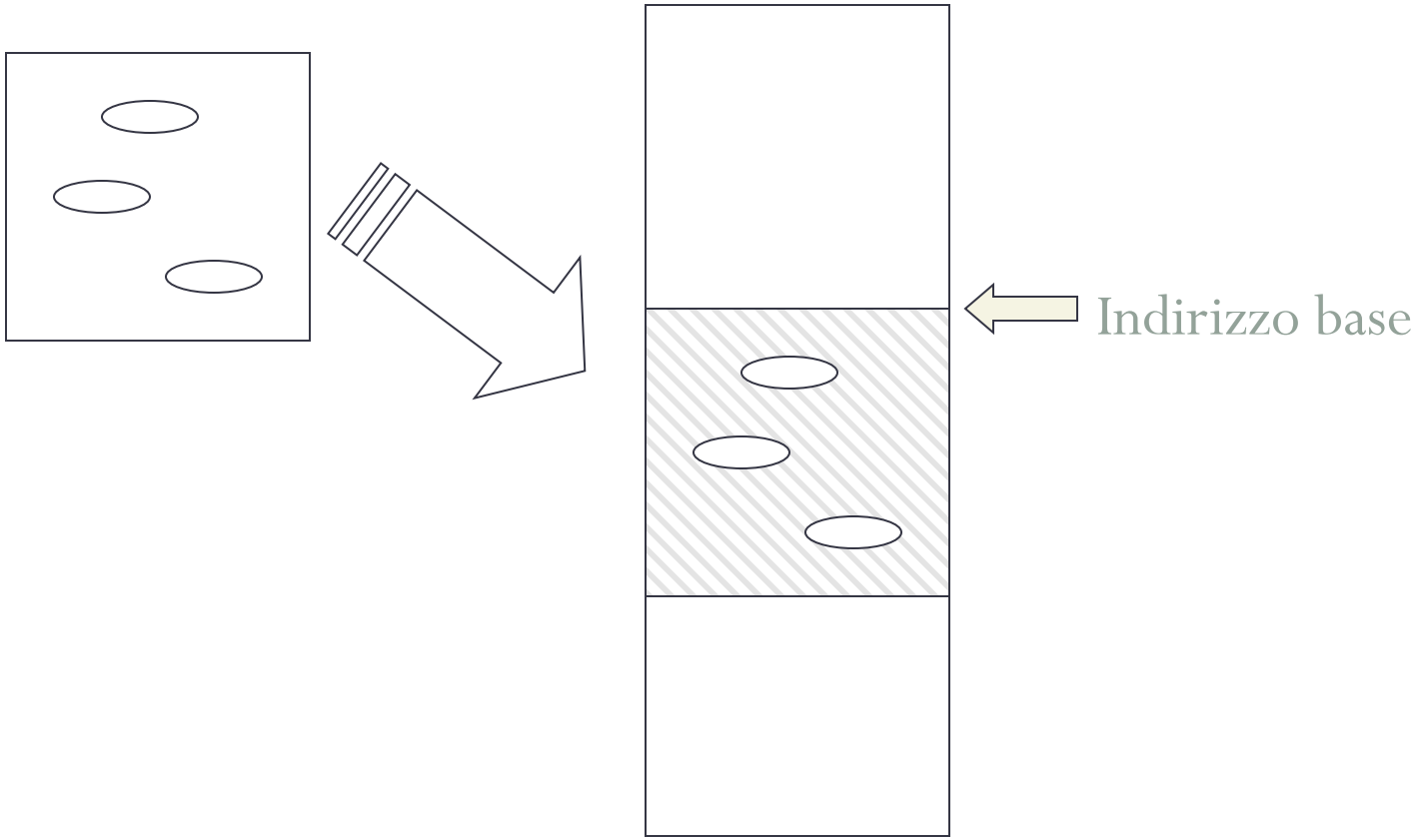
- **Fase di compilazione**: richiede di conoscere dove il processo risiederà in memoria
  - vengono generati solo indirizzi con riferimento a dove esattamente il codice dovrà risiedere in memoria durante l'esecuzione; **codice assoluto**
  - se la locazione di partenza cambia bisogna compilare di nuovo
  - se invece non si sa dove il processo risiederà è necessario generare del **codice rilocabile**

Indirizzamento (4)

- **Fase di caricamento**: gli eventuali indirizzi rilocabili (gli indirizzi fanno riferimento ad un indirizzo base non specificato) vengono associati ad indirizzi assoluti

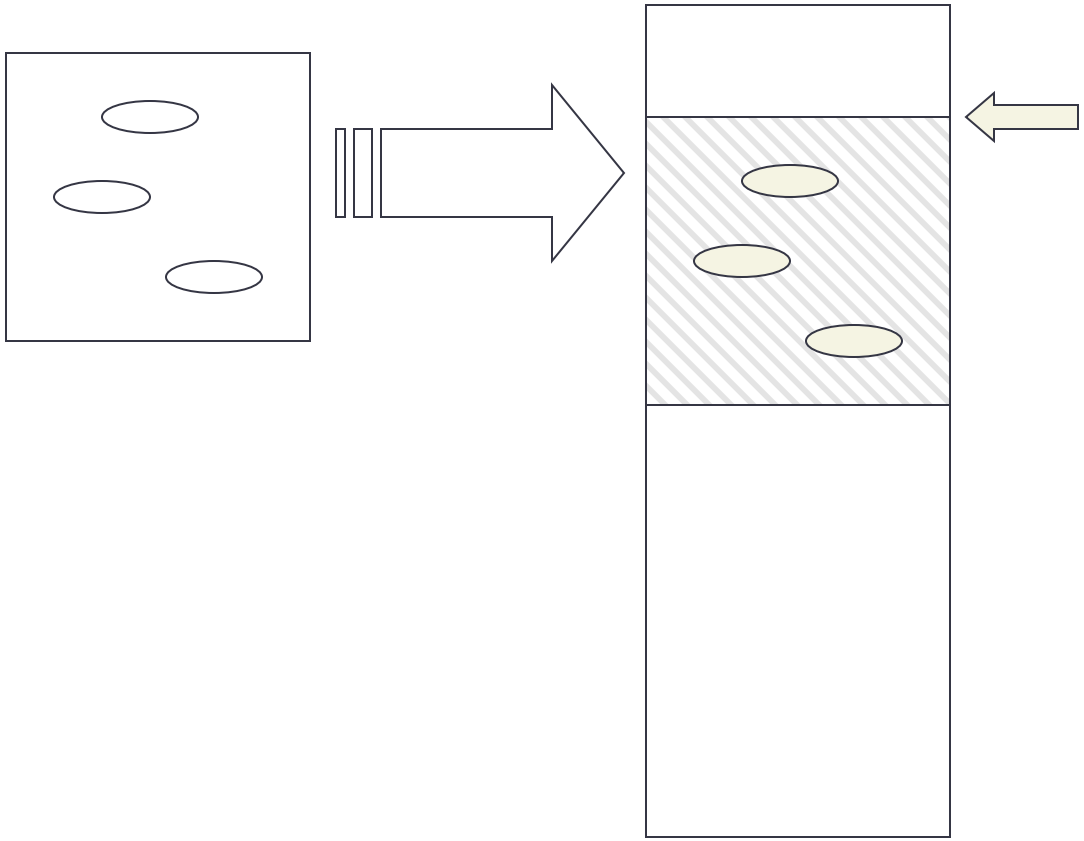
- **Fase di esecuzione**: se il processo può venire spostato in memoria durante l'esecuzione, allora il collegamento deve essere ritardato fino al momento dell'esecuzione; **codice dinamicamente rilocabile**
  - Contiene solo riferimenti relativi a se stesso
  - Necessita di un architettura hardware che consenta questa modalità
  - Metodo più utilizzato

Collegamento in compilazione



Caricamento statico in posizione fissa (codice assoluto)

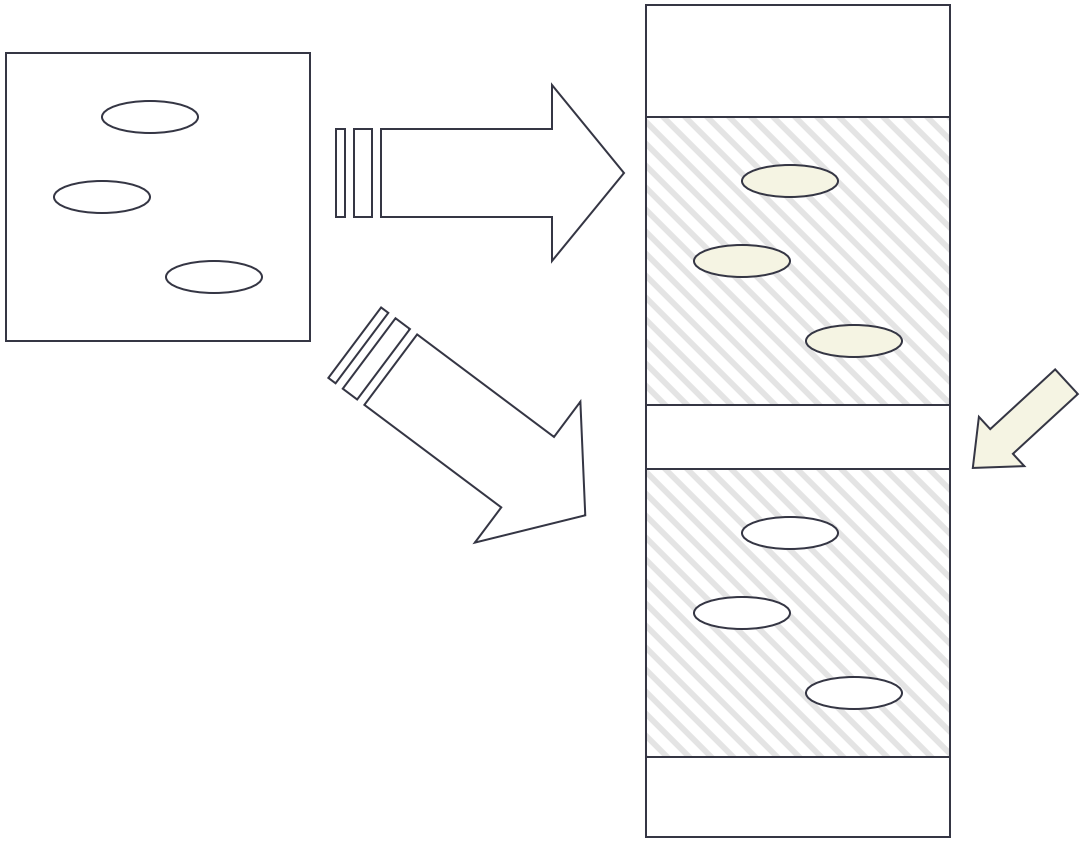
Collegamento in caricamento



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

Caricamento statico
con rilocazione del codice durante il caricamento

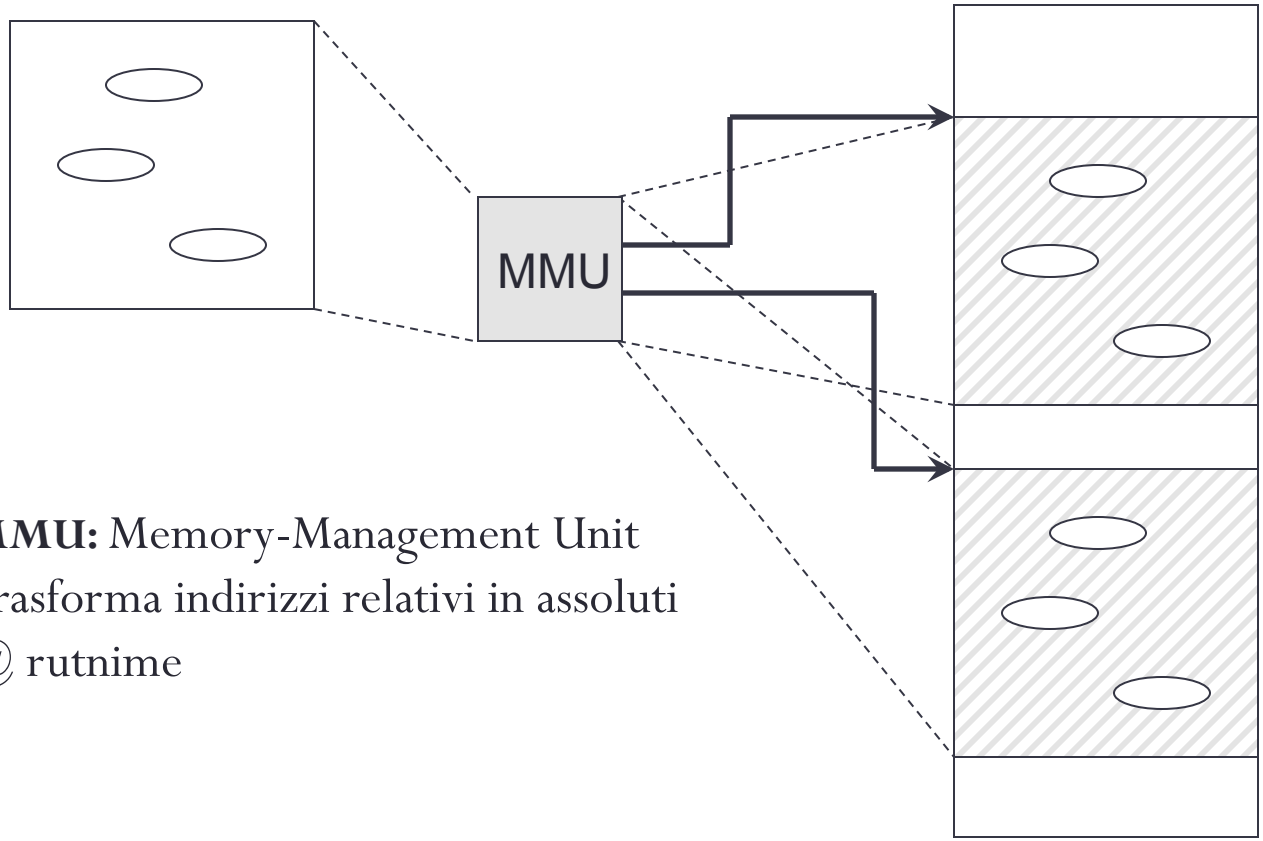
Collegamento in caricamento



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

Caricamento statico
con rilocazione del codice durante il caricamento

Collegamento in esecuzione



MMU: Memory-Management Unit
Trasforma indirizzi relativi in assoluti
@ rutnime

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

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 o virtuale**: indirizzo generato dalla CPU; anche definito come *indirizzo virtuale*
  - **Indirizzo fisico**: indirizzo visto dalla memoria

- Gli indirizzi logici sono trattati dai programmi utente, gli indirizzi fisici fanno riferimento alla effettiva posizione del dato nella 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 diversi
  - Gli indirizzi logici vengono chiamati in questo caso virtuali

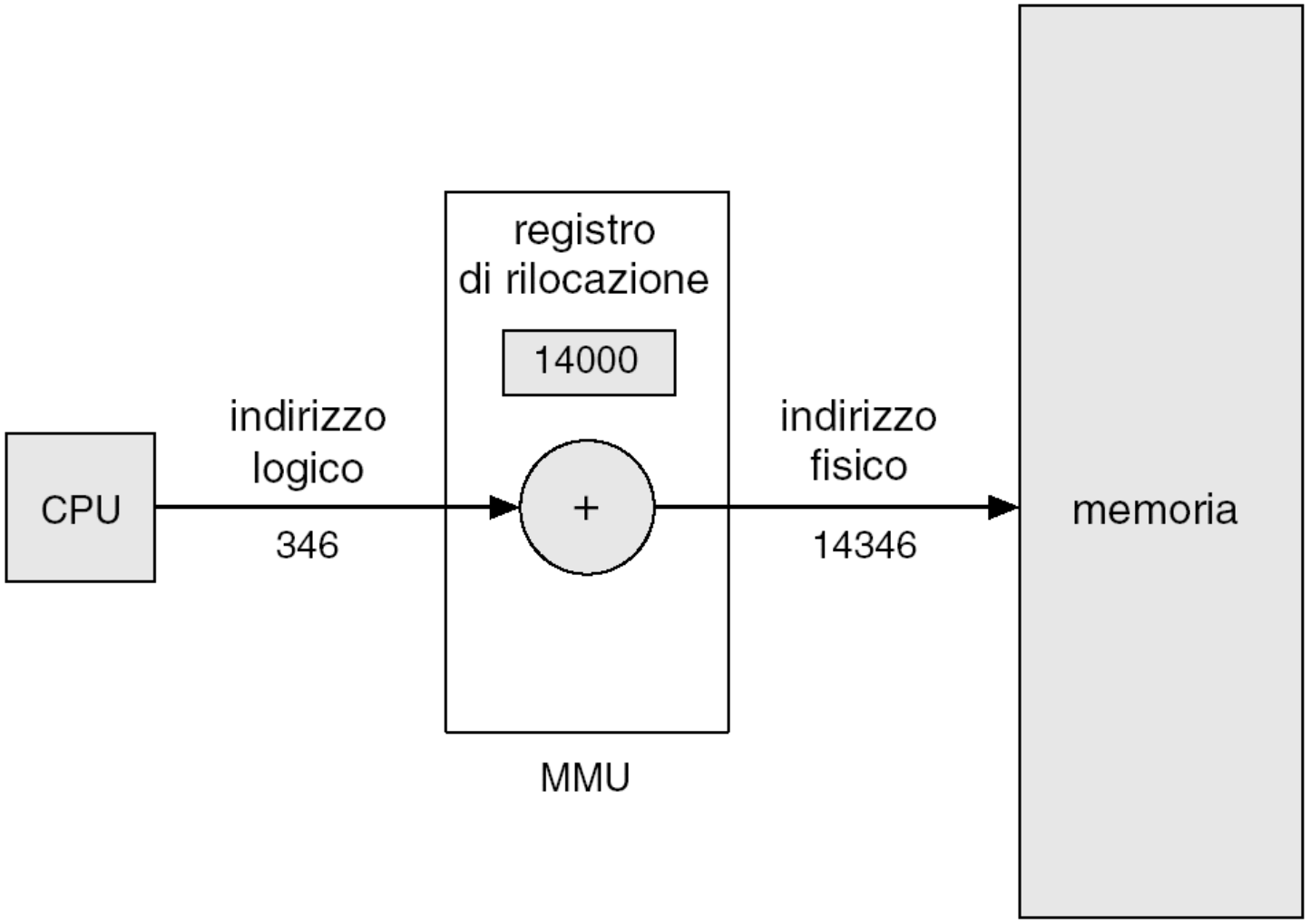
Unità di gestione della memoria centrale (MMU)

- **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 (r) è aggiunto ad ogni indirizzo logico generato da un processo nel momento in cui è trasmesso alla memoria

- Il programma utente interagisce con gli indirizzi logici (da `0` a `max`); non vede mai gli indirizzi fisici reali (da `r` a `r+max`)
- In questo modo il programma è indipendente da informazioni specifiche della memoria (valore di rilocazione, capacità della memoria, …)

Rilocazione dinamica mediante un registro di rilocazione



Fase di caricamento del programma

- **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
  - Si evita di occupare memoria caricando tutto il programma
  - Lo vedremo nella prossima lezione

Caricamento dinamico

- **Una procedura non è caricata finchè non è chiamata**
- Tutte le procedure risiedono in memoria secondaria, al momento del richiamo:
  - Si verifica se una procedura è già stata caricata,
  - E in caso negativo la si carica in memoria

- 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

- I programmi utente devono poter utilizzare librerie esterne (e.g. librerie di sistema)
- La fase di collegamento linka il codice utente a queste librerie
- Il caricamento può essere
  - Statico: le librerie esterne vengono incorporate nel file binario di esecuzione (file binari di dimensione molto grande)
  - Dinamico: le librerie vengono collegate al codice **utente solo in fase di esecuzione**

Collegamento (linking) dinamico

- Una piccola parte di codice (***stub***) indica come individuare la procedura di libreria desiderata residente in memoria o come caricarla se non è già presente
- L'immagine rimpiazza se stessa **con l'indirizzo della procedura** e la esegue
  - Si velocizzano eventuali chiamate successive
- 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

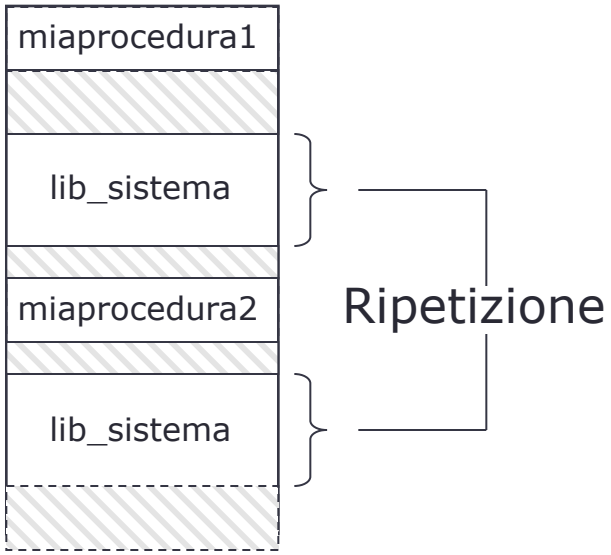
SENZA
COLLEGAMENTO DINAMICO
(COLLEGAMENTO STATICO)

CALL miaprocedura1

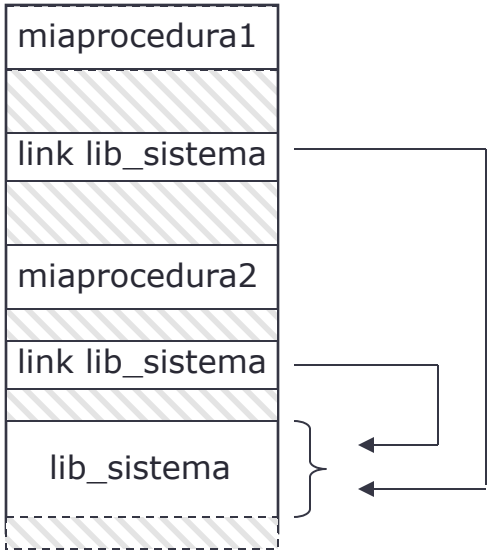
CALL lib_sistema

CALL miaprocedura2

CALL lib_sistema



CON
COLLEGAMENTO DINAMICO



Allocazione della memoria

- Nei prossimi lucidi ci concentreremo sull'allocazione della memoria
- Esistono due macro-approcci:
  - **Allocazione contigua**: tutto lo spazio assegnato ad un programma deve essere formato da celle consecutive
  - **Allocazione non contigua**: è 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

- Inoltre si può decidere la dinamicità con cui la memoria viene allocata:
  - **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

- Il gestore della memoria si può basare su diversi meccanismi utilizzandoli in base a opportune politiche
  - **Allocazione contigua, statica e dinamica**
    - Swapping
    - A Partizioni: singola, partizioni multiple fisse e variabili
  - **Allocazione non contigua**
    - Paginazione
    - Segmentazione
    - Segmentazione con paginazione

Tecniche di gestione della memoria: problema ricorrente

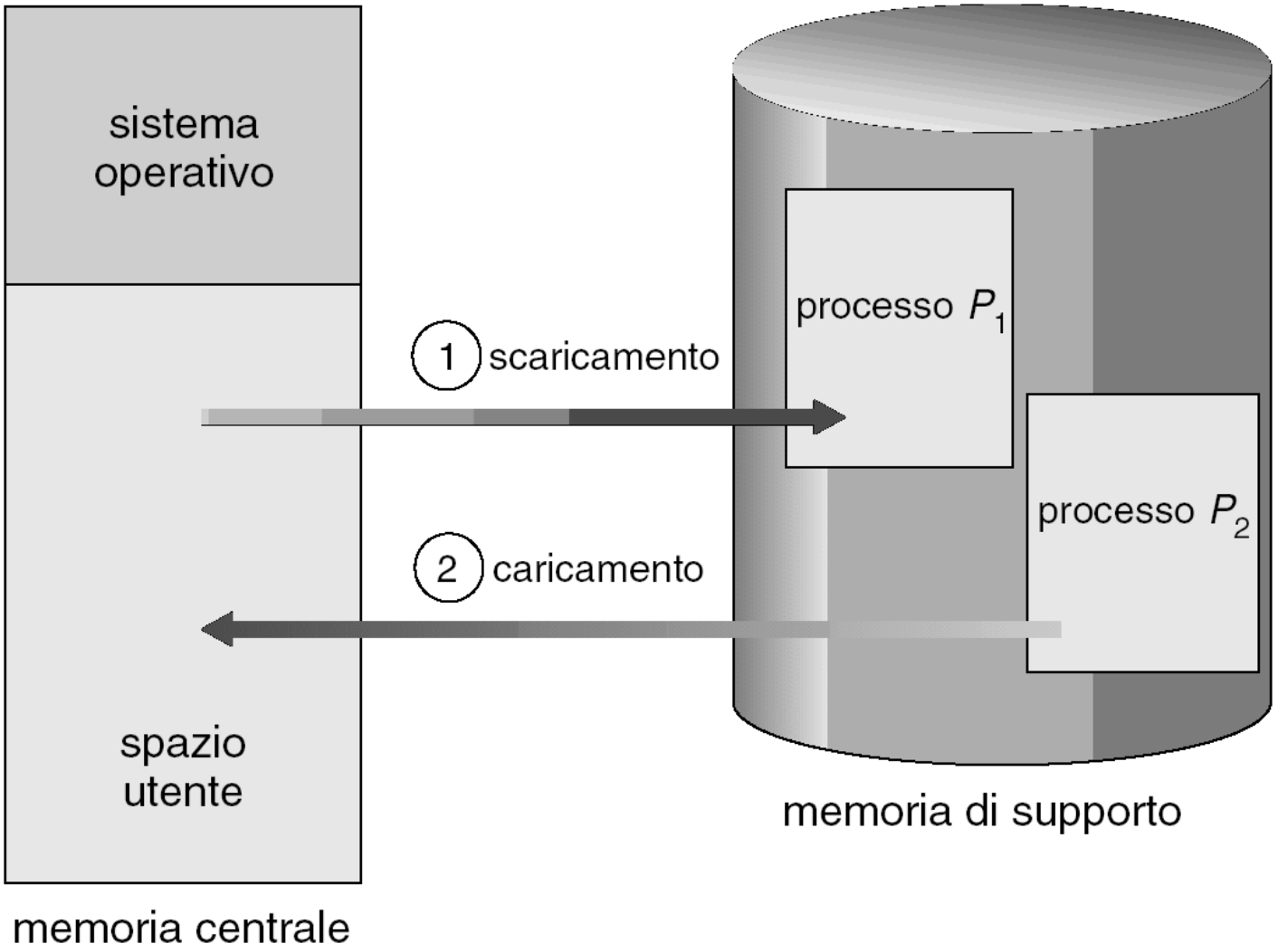
- L'allocazione della memoria porta al problema della **frammentazione**
  - **Interna**: lo spazio allocato in eccesso rispetto alle esigenze dei processi, inutilizzabile perché allocato
  - **Esterna**: lo spazio libero in aree troppo piccole per essere utili
- Tutte le tecniche di gestione della memoria 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

Swapping (1)

- Un processo può essere temporaneamente scambiato (**swapped**)
  - spostandolo dalla memoria **centrale** ad una memoria **secondaria** (**area di swap**)
  - e poi in seguito riportato **interamente** in memoria centrale per continuarne l'esecuzione
- È detto **avvicendamento semplice**, o **swapping**

- Memoria secondaria: 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)

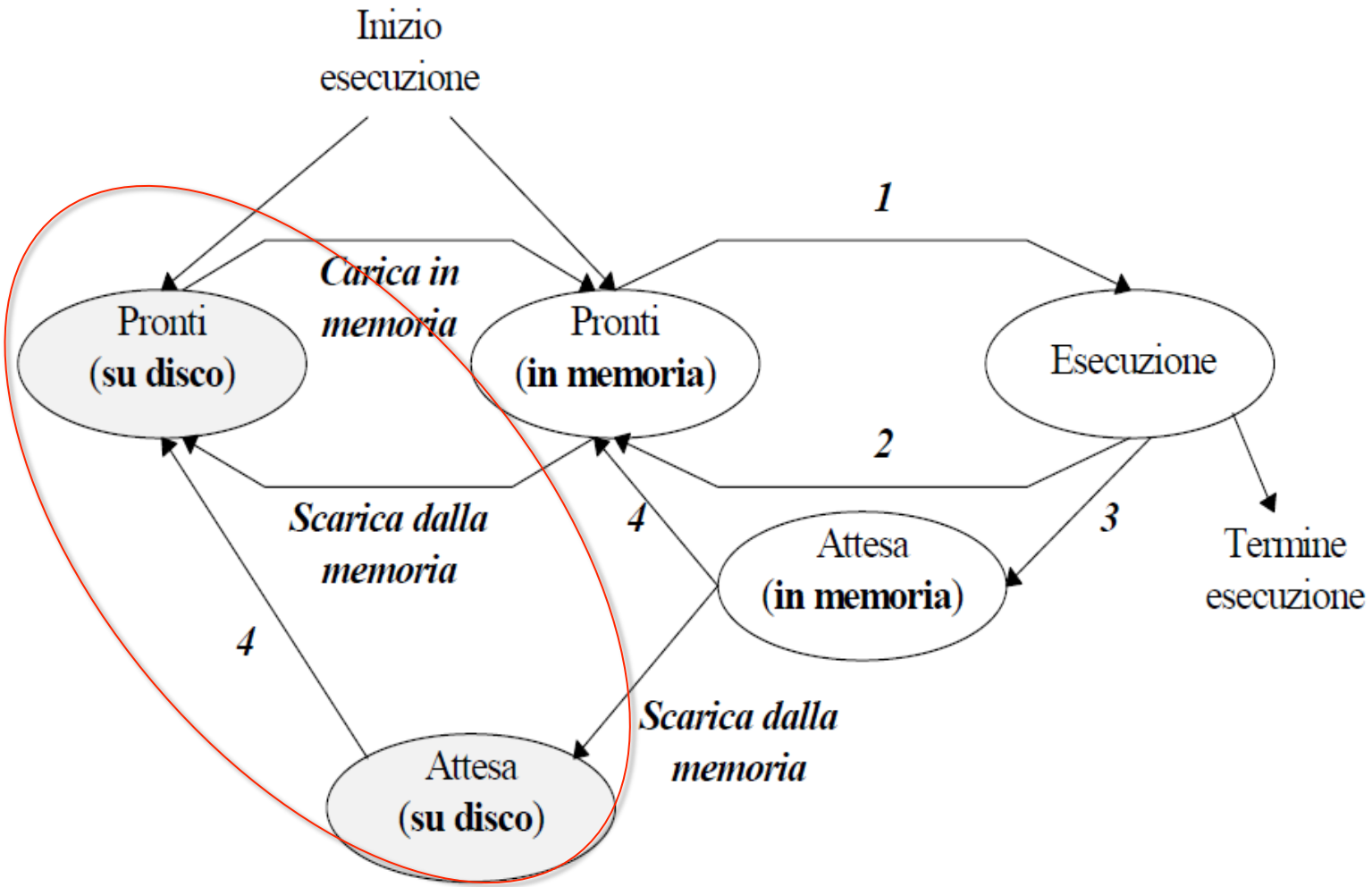
- Permette di gestire più processi di quelli che **fisicamente** sarebbero caricabili in memoria

- Il periodico scambio tra processi in memoria centrale e secondaria (swapping) è controllato dallo **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)

- Lo swapping è molto comune nei sistemi con schedulatore Round Robin
  - Il processo che finisce il quanto di tempo subisce lo swap-out
- Roll out, roll in è una 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) combinate con altre tecniche

Swapping (5)

- La zona di memoria in cui i processi che subiscono lo swap-in vengono spostati dipende dalla fase in cui gli indirizzi logici vengono associati a quelli fisici
  - **Fasi di assemblaggio o caricamento**: i processi vengono spostati nello stesso punto in cui sono stati originariamente caricati
  - **Fase di esecuzione**: i processi vengono spostati in uno spazio qualunque della memoria
    - Gli indirizzi vengono infatti ricalcolati al momento del passaggio in esecuzione

Swapping (6)

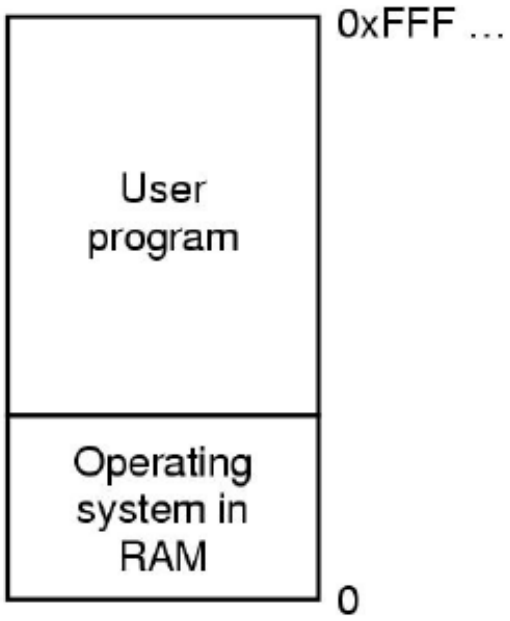
- La maggior parte del context switch è dovuto al tempo di trasferimento
- Il tempo totale di trasferimento è direttamente proporzionale alla quantità di memoria spostata
- Esempio:
  - Supponiamo che un programma occupi 100 MB in memoria centrale ed il sistema abbia una velocità di trasferimento della memoria di 50 MB/s
  - $T_{trasferimento} = 100/50 = 2$ s $= 2000$ ms
  - $T_{latenza} = 8$ms (tempo in media necessario per fermare un processo e avviarne un altro)
  - $T_{context\text{-}switch} = (2000 + 8) * 2 = 4016$ ms
- In conclusione è bene sapere sia quanta memoria i processi occupano effettivamente che quanta potrebbero richiederne in modo da alternarli in modo **più veloce possibile**

Swapping (7)

- Lo swapping deve tenere conto anche di possibili I/O
  - Se i processi sono impegnati in un I/O asincrono non possono essere avvicendati
- In conclusione l'avvicendamento semplice è oggi **poco usato**
  - Richiede un elevato tempo di gestione
  - Consente un tempo di esecuzione troppo breve per i processi
- Si preferiscono versioni modificate

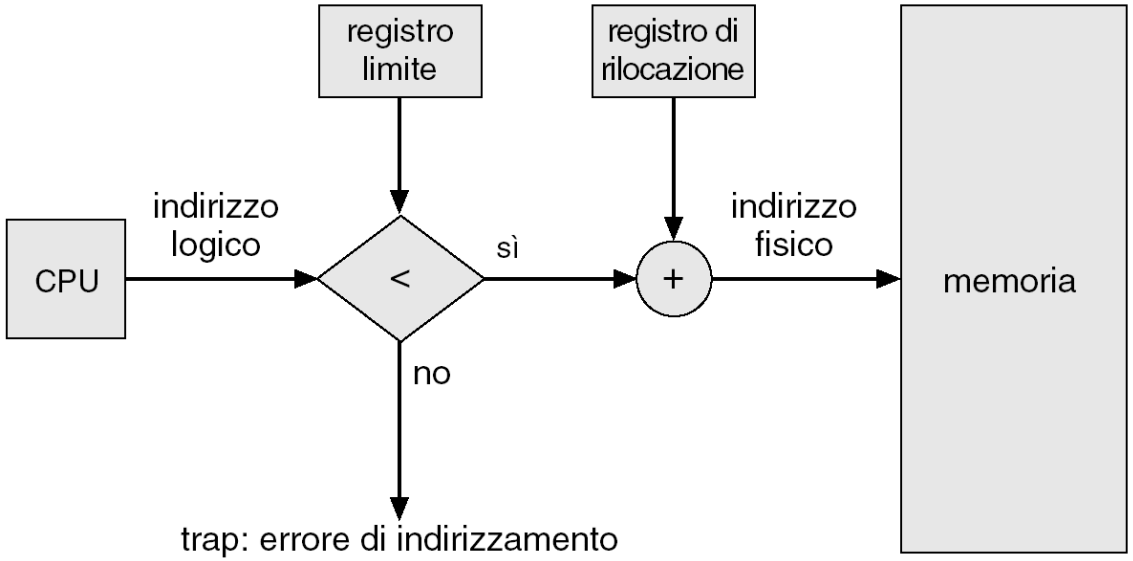
Allocazione contigua della memoria

- La memoria centrale è divisa in **due partizioni**:
  - Una per il SO (di solito collocato nella memoria bassa, vicino al vettore delle interruzioni)
  - E una per un processo utente (solitamente collocato nella memoria alta)
- È uno schema di **allocazione contigua e statica**



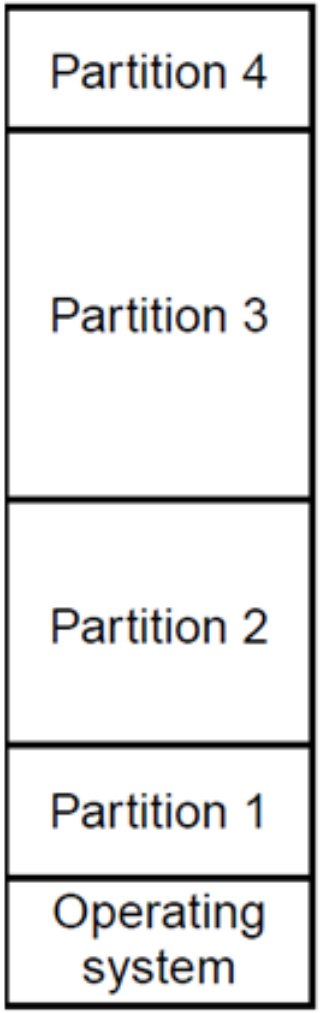
Rilocazione e protezione della memoria

- Le tecniche di allocazione basate su memoria contigua usano anche delle tecniche per **proteggere** la memoria del SO e dei processi utenti
- Il dispatcher sceglie uno dei processi presenti nella coda dei pronti e carica i corrispondenti **registri di rilocazione e limite**
- Confrontando ogni indirizzo prodotto dalla CPU con i valori di questi registri la memoria risulta protetta



Allocazione a Partizioni Multiple Fisse (1)

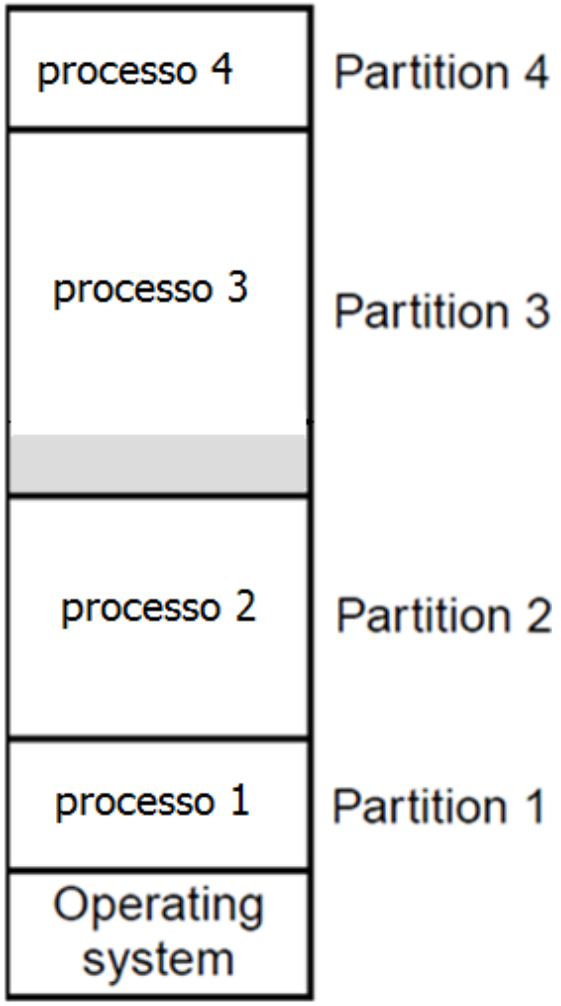
- La memoria è divisa in un **numero fisso *n* di aree** dette **partizioni** di dimensioni possibilmente **diverse**
  - Ogni partizione contiene un processo ed è identificata da coppia di registri *base-limite*
  - Quando c'è una partizione libera, un processo viene caricato in essa ed è pronto per essere schedulato per l'esecuzione
  - n definisce il livello di multiprogrammazione
- È necessario conoscere la *dimensione del processo* prima di attivarlo
- In fase di contex 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)

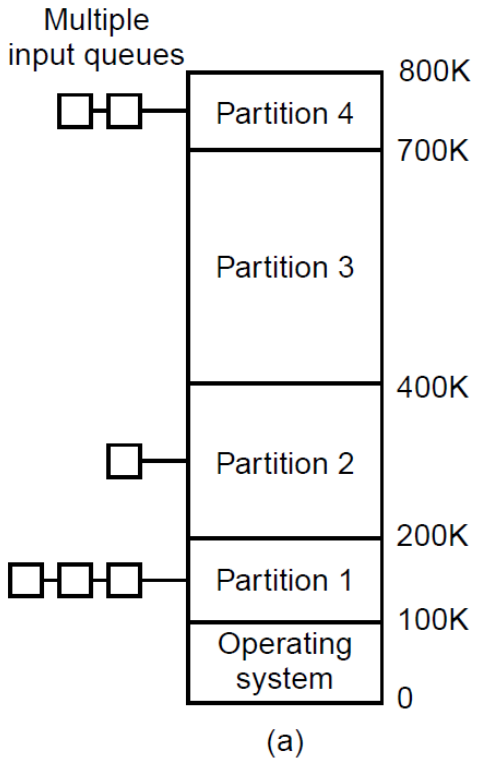
- **Processi piccoli**: causano spreco di spazio di memoria (frammentazione interna)

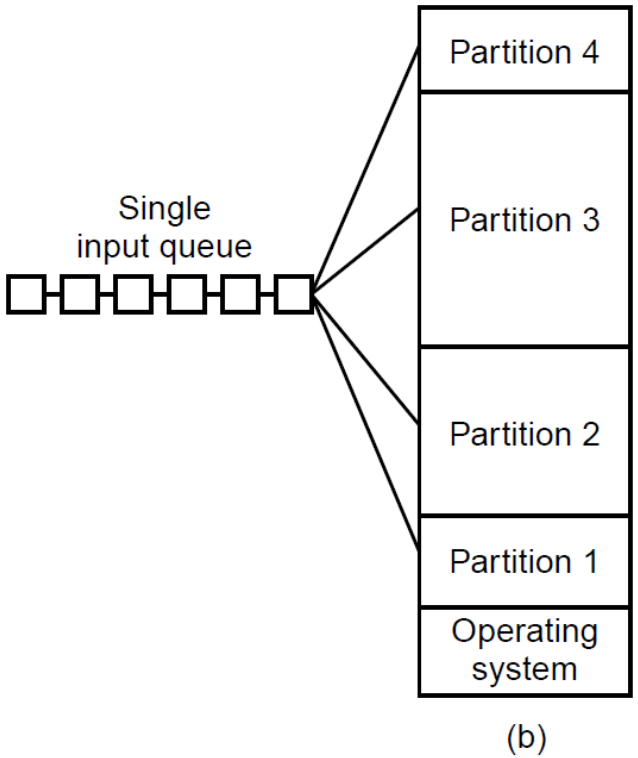
- **Processi grandi**: più grandi della più grande partizione non possono essere eseguiti
  - Aumentando la dimensione delle partizioni diminuisce il grado di multiprogrammazione,
  - e aumenta la frammentazione interna…



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**

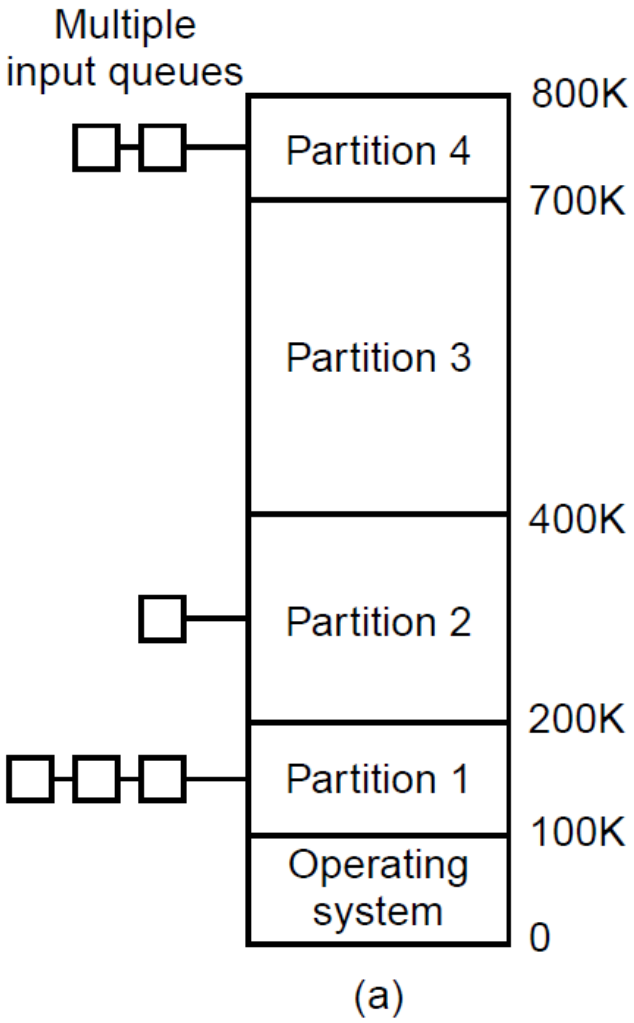




Allocazione a Partizioni Multiple Fisse (4)

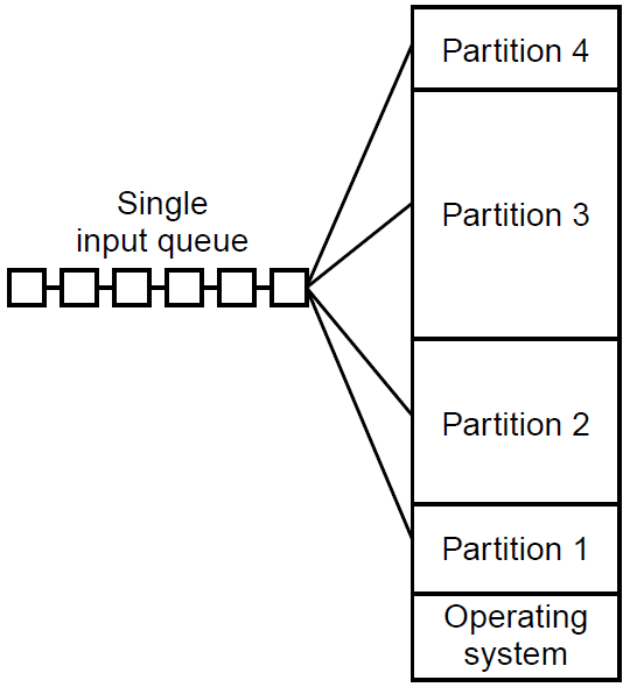
Se si utilizza **una coda per ogni partizione**:

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



Allocazione a Partizioni Multiple Fisse (5)

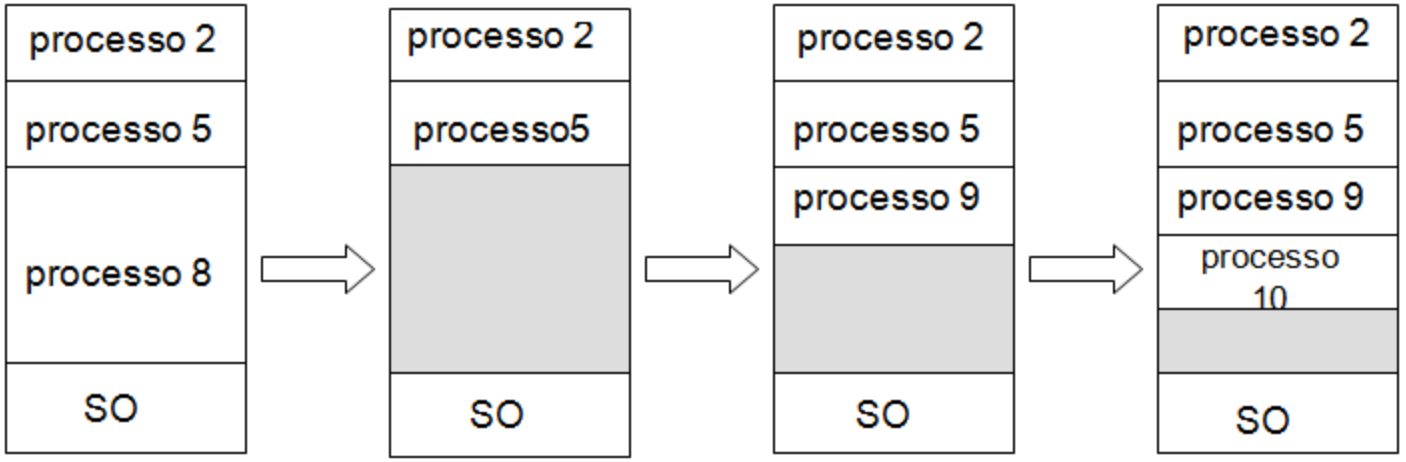
- Se si utilizza **una singola coda di attesa**: appena si libera una partizione si sceglie il processo secondo un algoritmo spazio per il primo processo
  - **First-fit**: il primo processo con dimensione minore uguale alla partizione
  - **Best-fit**: il processo più grande fra quelli con dimensioni minore uguale alla partizione
  - **Best-fit con upper bound** sul numero di volte in cui un processo può essere scartato



(b)

Allocazione a Partizioni Multiple Variabili (1)

- Il SO tiene traccia in una tabella di quali parti della memoria sono occupate e quali no:
  - Il numero, la dimensione e la posizione delle partizioni allocate 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



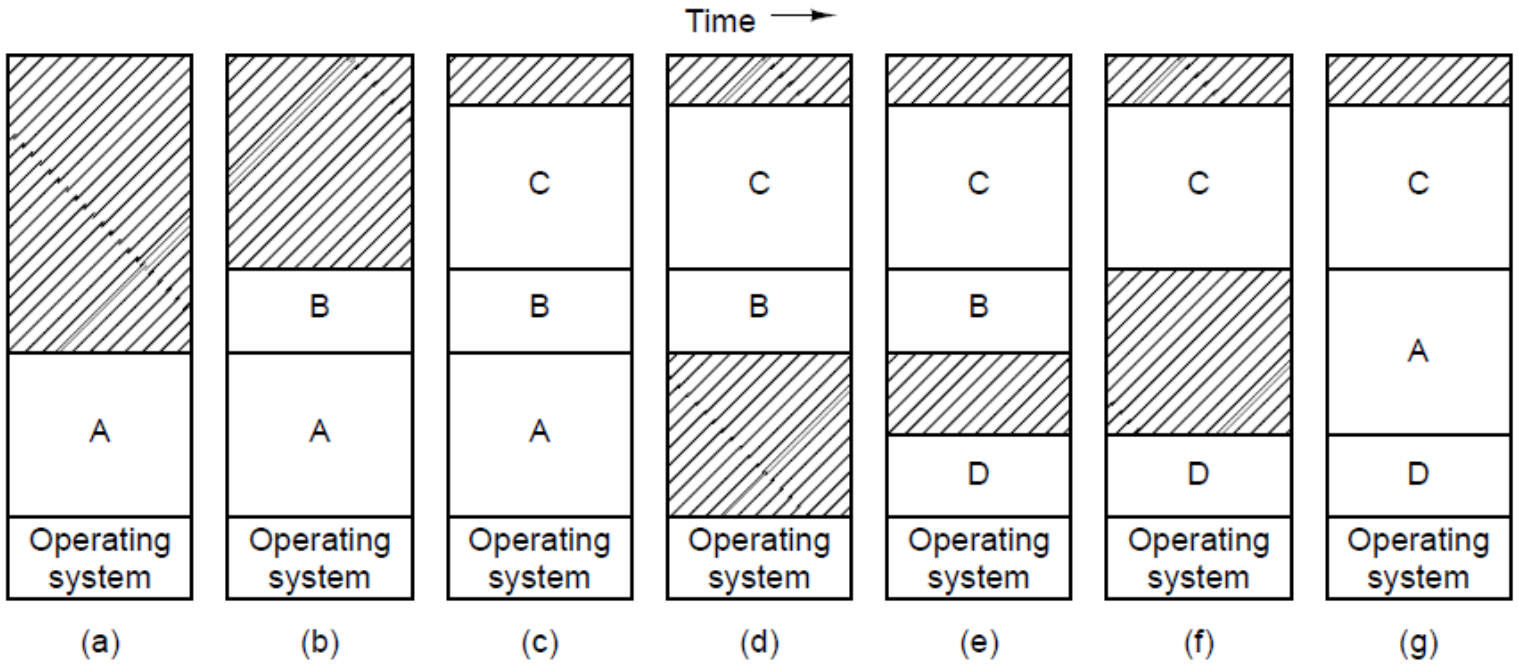
Allocazione a Partizioni Multiple Variabili (2)

- Il SO ha una coda dei processi pronti
  - La memoria è assegnata al primo processo a patto che ci sia una partizione abbastanza grande
  - Altrimenti il SO può attendere o passare al processo successivo
- La partizione da usare viene scelta fra quelle abbastanza grandi secondo un algoritmo:
  - **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
- Wors-fit è il peggiore per tempi e uso della memoria. First-fit e best-fit sono paragonabili per uso della memoria ma il primo è più veloce.

Allocazione a Partizioni Multiple Variabili (3)

Esempio

- Alla lunga lo spazio libero appare suddiviso in piccole aree…è il fenomeno della **frammentazione esterna**
- Si stima che l'algoritmo worst-fit abbia 0.5 unità di frammentazione per ogni unità di allocazione



Allocazione a Partizioni Multiple Variabili (4) Compattazione

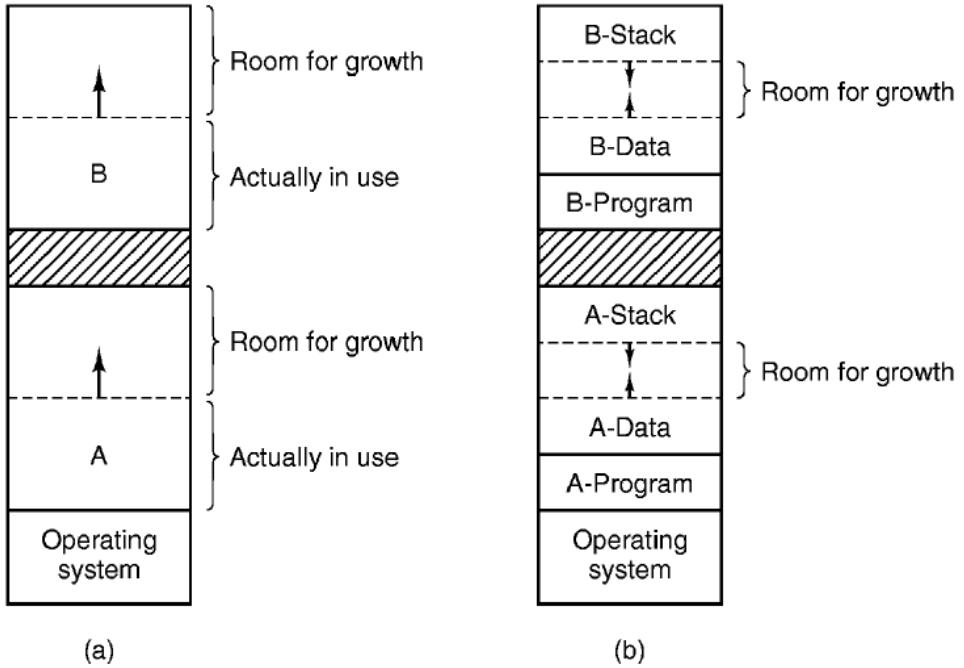
- La frammentazione esterna può essere ridotta attraverso la **compattazione**:
  - Spostare in memoria tutti i programmi in modo da **"fondere" tutte le aree inutilizzate** per avere tutta la memoria centrale libera in un grande blocco

- Svantaggi:
  - La compattazione è possibile **solo se la l'associazione** tra indirizzi virtuali e fisici è fatta dinamica al **momento dell'esecuzione**
  - E' un operazione **molto onerosa**: occorre copiare (fisicamente) in memoria grandi quantità di dati

- Una soluzione alternativa è quella di consentire una allocazione non contigua: tecniche di paginazione, segmentazione e combinazione delle due

Allocazione a Partizioni Multiple Variabili (5)

- Se durante l'esecuzione i processi possono "crescere", 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)

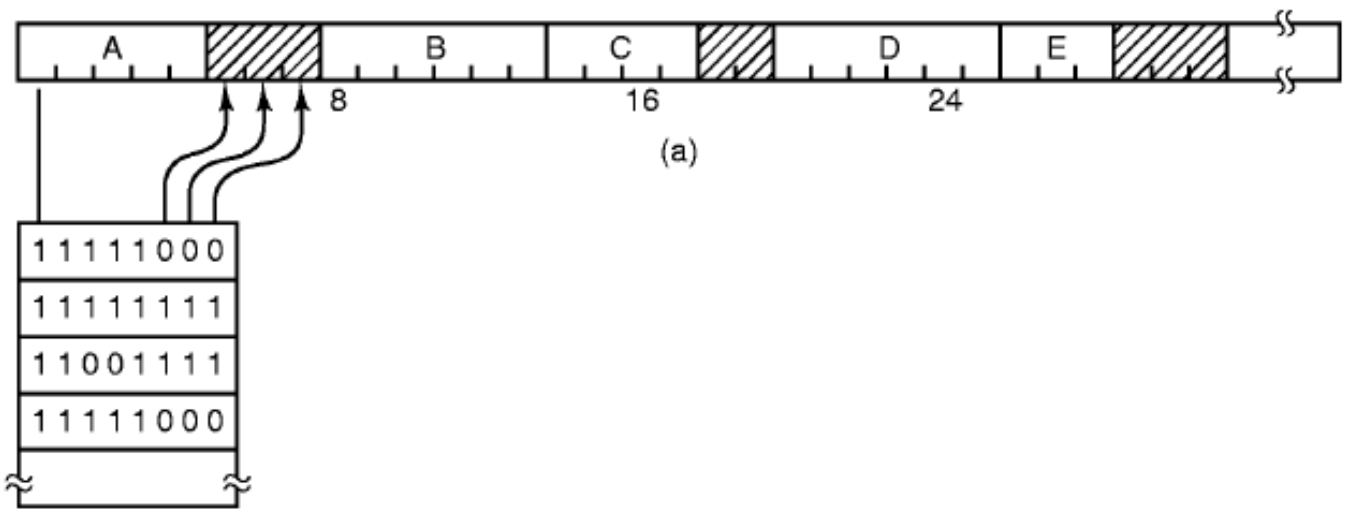
- In generale, ogni tecnica basata su **allocazione dinamica** ha bisogno di una struttura dati per mantenere informazioni sulle zone libere e sulle zone occupate

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

Allocazione dinamica – strutture dati (2)

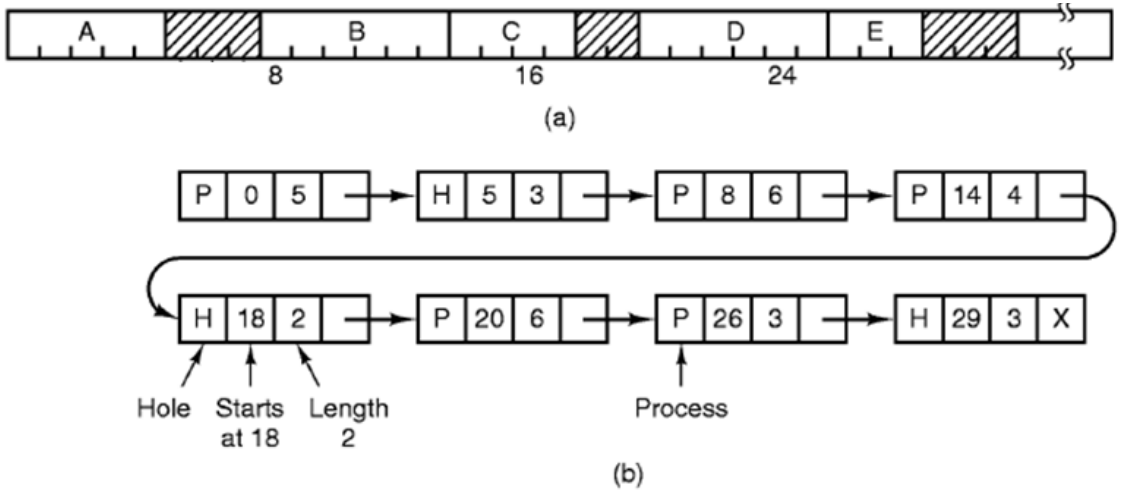
Mappa di bit: la memoria viene suddivisa in *unità di allocazione*, ad ogni unità di allocazione corrisponde un bit in una bitmap: valore 0 (unità libera), valore 1 (unità occupata)

- 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



Allocazione dinamica – strutture dati (3)

- Lista concatenata bidirezionale: possibili elementi sono processi e holes



- Allocazione: *ricerca spazi 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

- Cosi come la RAM anche lo spazio di SWAP deve essere gestito
- 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

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

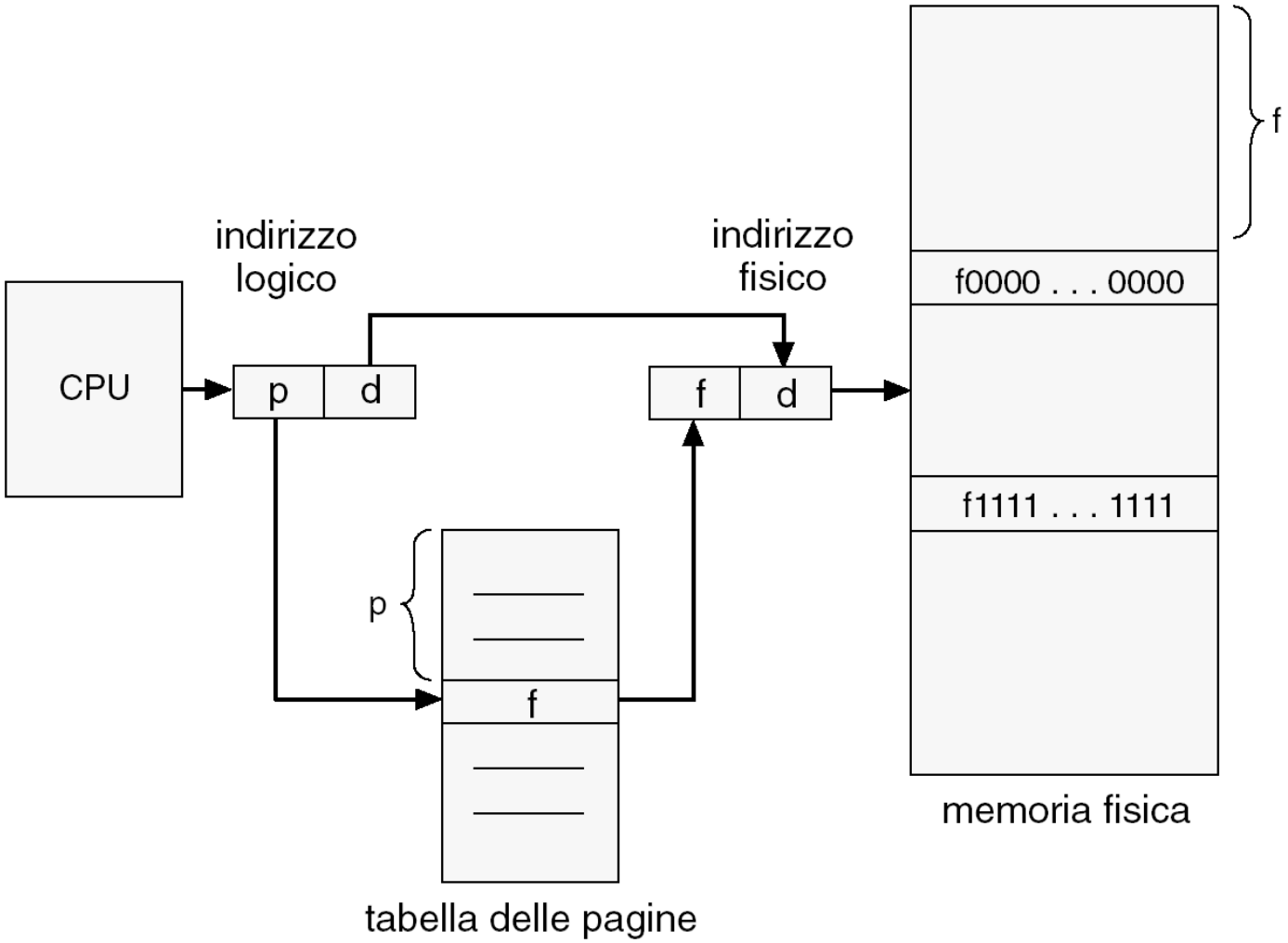
- Suddivide la **memoria fisica** (memoria centrale) in blocchi di **frame** della stessa dimensione (una potenza di 2, fra 512 byte e 16 MB)
  - Lo spazio degli indirizzi fisici può essere non contiguo
- Divide la **memoria logica** in blocchi delle stesse dimensioni dei frame chiamati **pagine**
  - 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

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

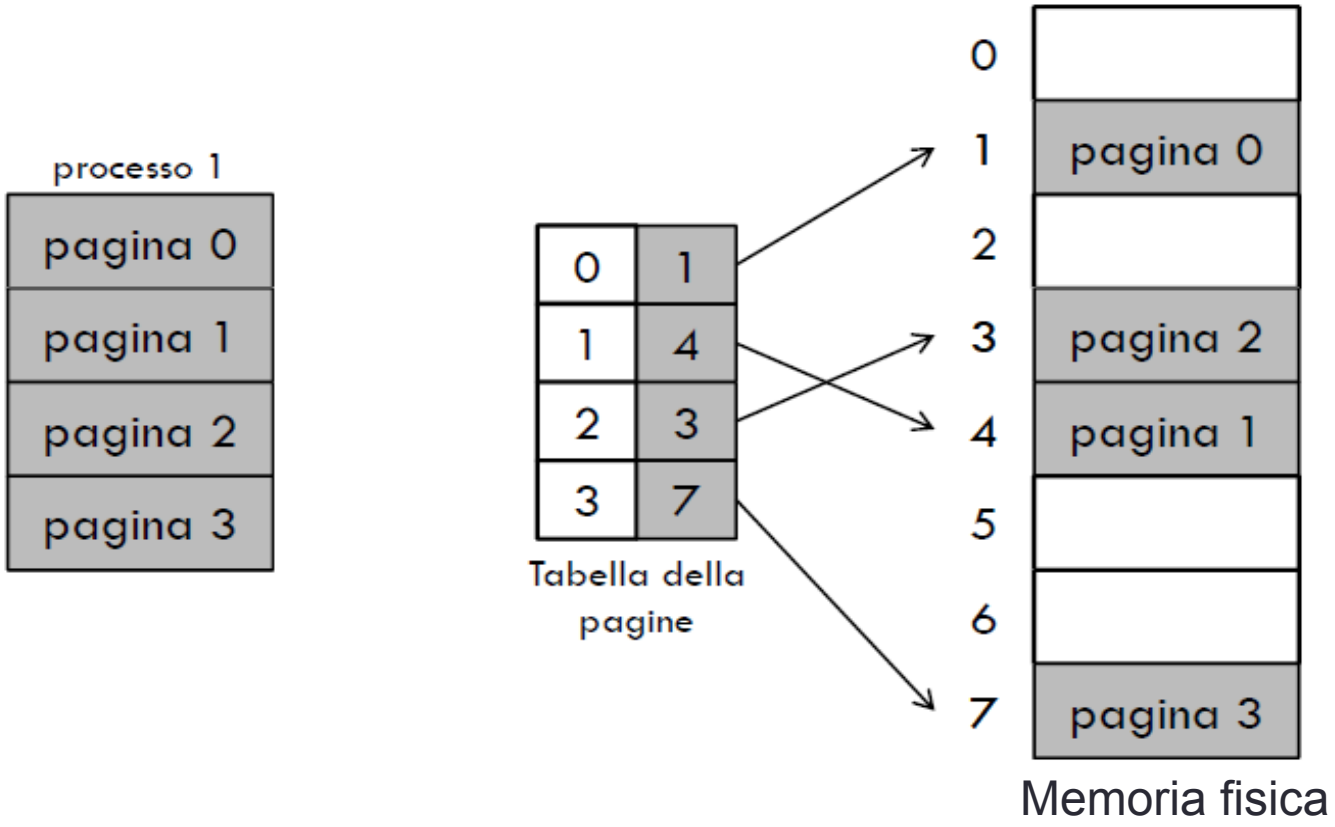
Indirizzo logico

Paginazione: schema di traduzione dell'indirizzo (2)



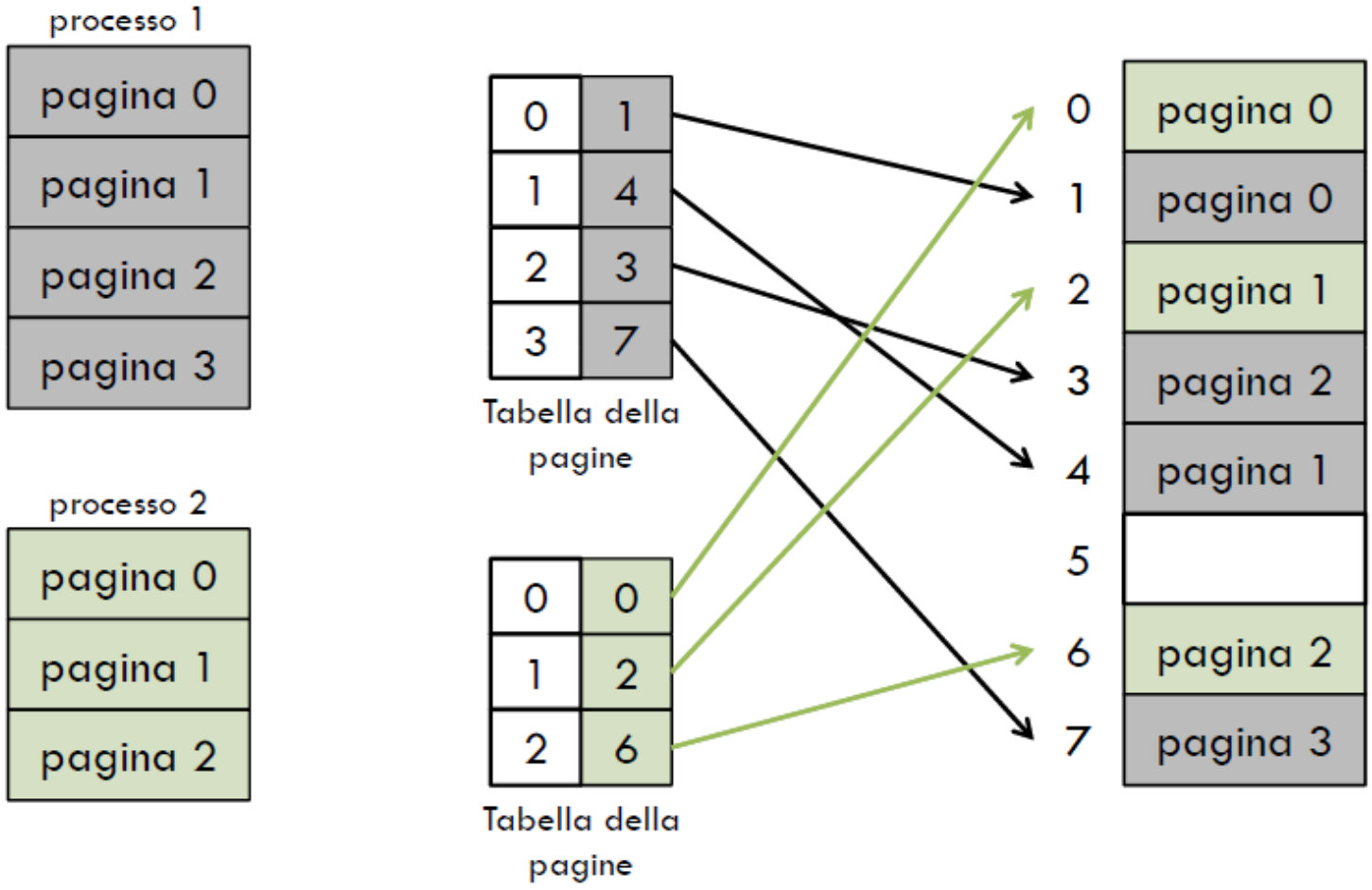
Paginazione: un esempio

- La tabella delle pagine associa l'indirizzo di una pagina logica al corrispettivo indirizzo della pagina nella memoria fisica



Paginazione: un altro esempio

- Le aree di memoria dei processi possono essere interfogliate

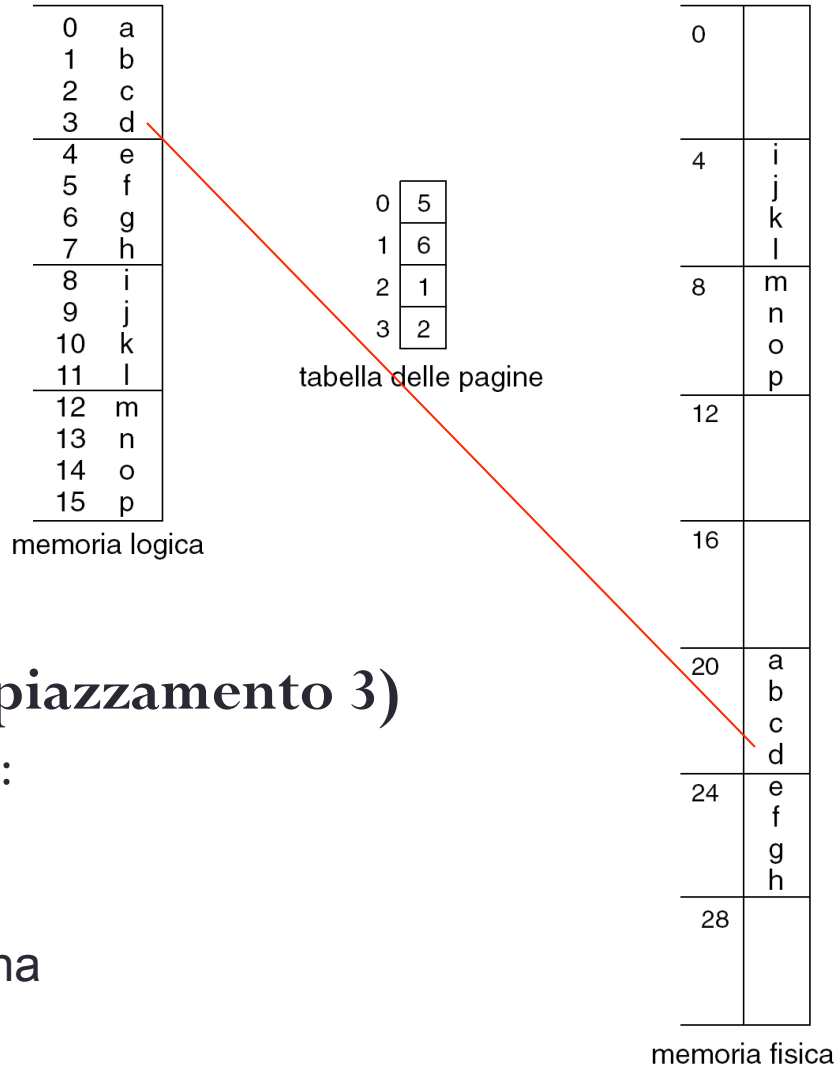


Paginazione: dimensione delle pagine

- La dimensione di una pagina **varia dai 512 byte ai 16MB**
- La dimensione della pagina è definita dall'architettura del sistema ed **è in genere una potenza di 2**
  - perché semplifica la traduzione degli indirizzi
- Infatti, 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 lo scostamento di pagina d**

Indirizzo logico

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

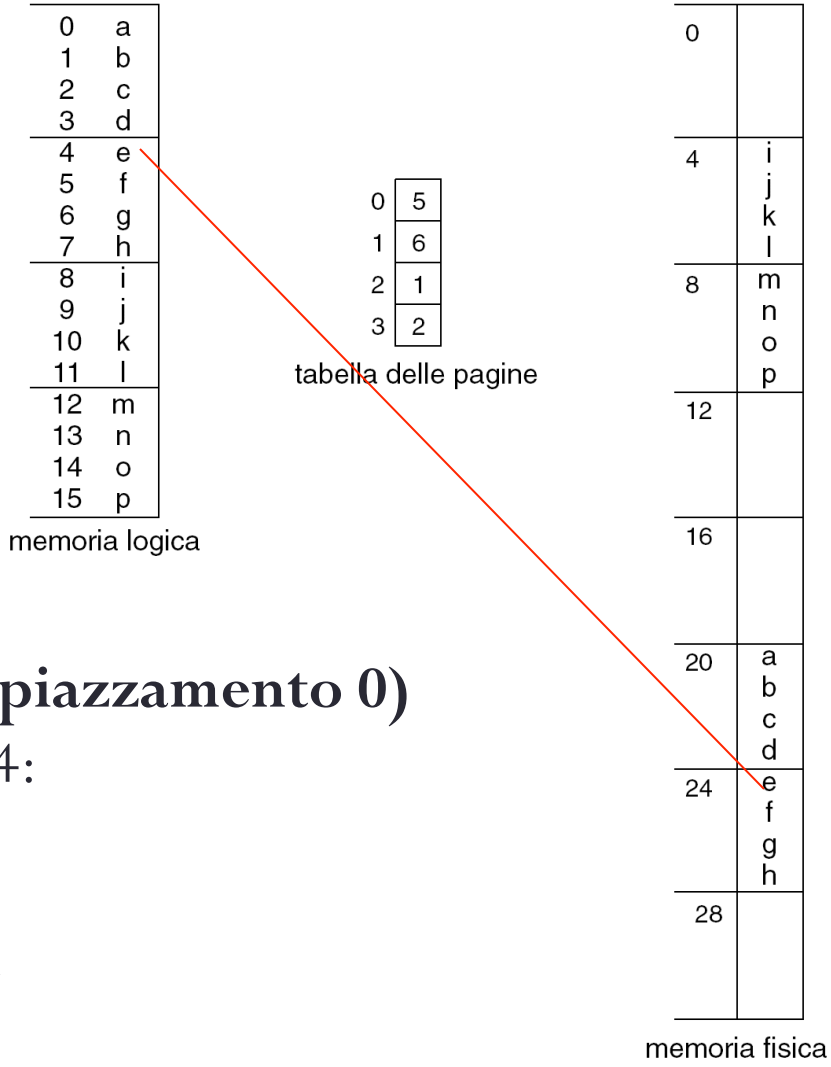


L'indirizzo logico **3 (pagina 0, spiazzamento 3)** corrisponde all'indirizzo fisico 23:

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

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, spiazzamento 0)** corrisponde all'indirizzo fisico 24:

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

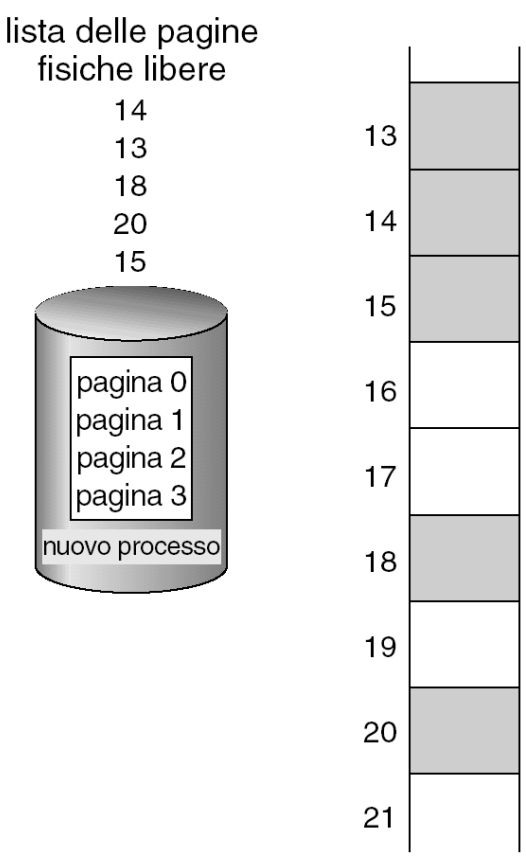
frame — dimensione pagina

Paginazione: frame liberi

Grigi = liberi

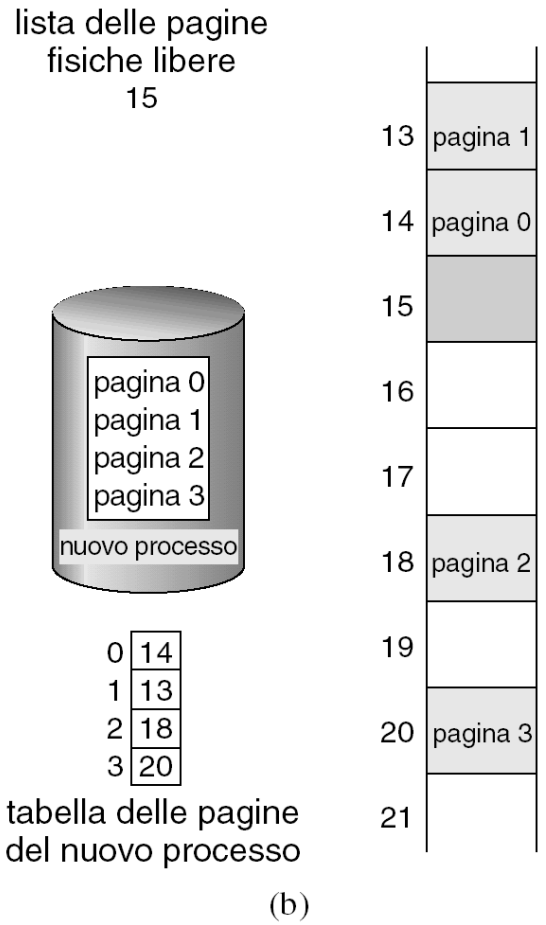
Dato un processo con n pagine si verifica la presenza di n frame liberi.

Se ci sono:
Si cerca un frame per la prima pagina,
la si trasferisce e
si procede con gli altri



(a)

Prima dell'allocazione



(b)

Dopo l'allocazione

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

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) e I/O più efficiente

Implementazione della tabella delle pagine Uso dei registri

- La maggior parte dei sistemi usa una tabella delle pagine per ogni processo

- **Prima soluzione**
  - Si usa uno specifico insieme di registri per salvare la tabella
  - Durante il context switch i valori della tabella delle pagine del processo vengono caricate nei registri
  - Molto veloce
  - Ma va bene solo per tabelle con pochi elementi (256 circa, quando tipicamente ne servono 1.000.000)

Implementazione della tabella delle pagine RAM + PTBR

- **Seconda soluzione**
  - La tabella delle pagine viene mantenuta in memoria centrale
  - Un registro chiamato page-table base register (PTBR) punta alla tabella del processo corrente
  - Cambiare tabella delle pagine significa cambiare valore del registro
  - Problema: per accedere ad un dato occorrono due accessi alla memoria centrale
    - 1 per la tabella delle pagine
    - 1 per il byte stesso
  - L'accesso alla memoria è rallentato di un fattore 2 (sarebbe preferibile usare lo swapping)

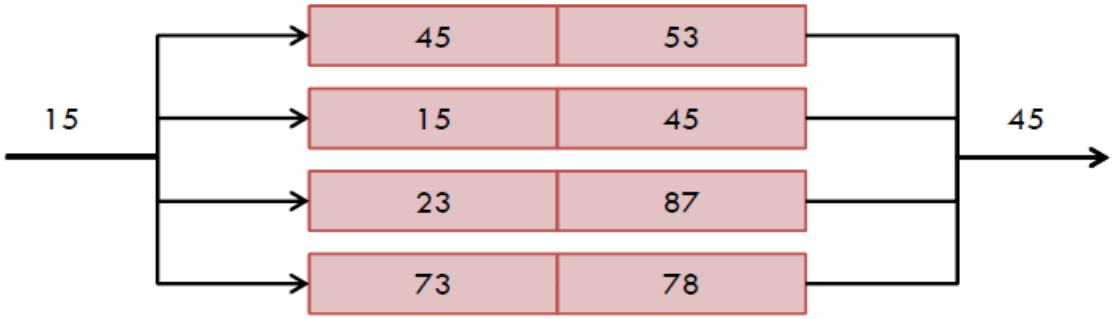
Implementazione della tabella delle pagine RAM + TLB

- **Terza soluzione**
  - Si utilizza una cache associativa: translation look-aside buffer (TLB)
  - Associa ad una chiave un valore
    - Chiave: indirizzo logico - Valore: indirizzo fisico
    - Molto rapida ma anche costosa e piccola (max 1024 elementi)
  - I record possono essere scritti e sovrascritti
  - Ma possono anche essere vincolati (non modificabili)
  - Utilizzo:
    - La TLB contiene una piccola parte delle righe della tabella delle pagine

Implementazione della tabella delle pagine RAM + TLB

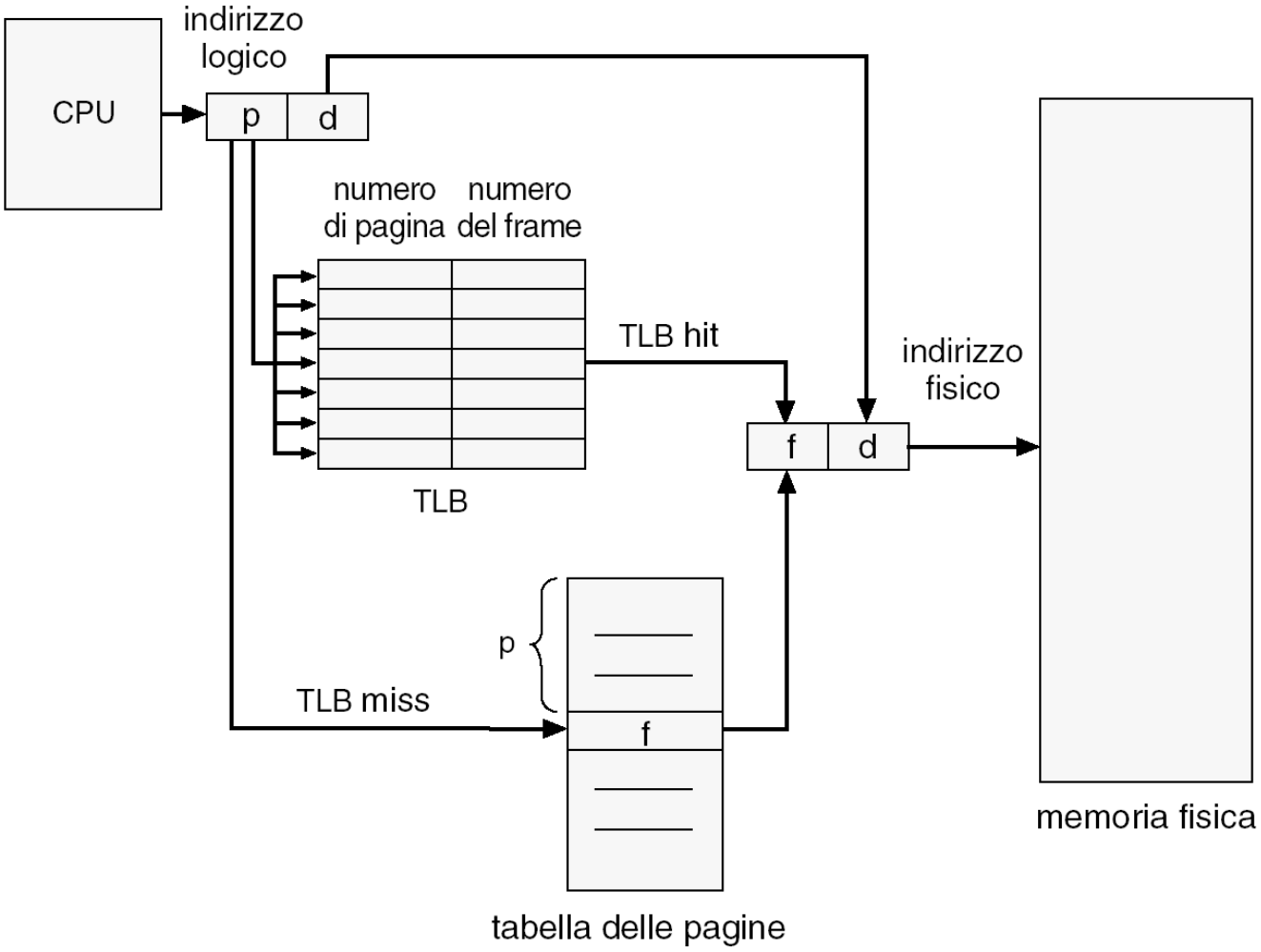
- Memoria associativa

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



- Dato un indirizzo logico lo si passa alla TLB, se essa contiene tale chiave restituisce l'indirizzo fisico
- Altrimenti (TLB Miss) si cerca nella tabella residente in memoria centrale
  - Si copiano poi i due indirizzi nella TLB in modo da velocizzare accessi futuri
  - Se la tabella è piena si sceglie quale record togliere secondo diversi criteri (e.g. elemento usato meno di recente, casuale)

Implementazione della tabella delle pagine RAM + TLB



Tempo di accesso effettivo

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

- **Tempo di accesso effettivo (EAT)**

$$\begin{aligned} \text{EAT} &= (c + \varepsilon)\,\alpha + (2c + \varepsilon)(1 - \alpha) \\ &= 2c + \varepsilon - c\,\alpha \end{aligned}$$

Tempo di accesso effettivo: un esempio

- Tempo di ricerca nella TLB $= \varepsilon$ = 20 ns
- Tempo di accesso alla memoria = c = 100 ns
- Tasso di accesso con successo $\alpha = 0.80$ (80%)

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

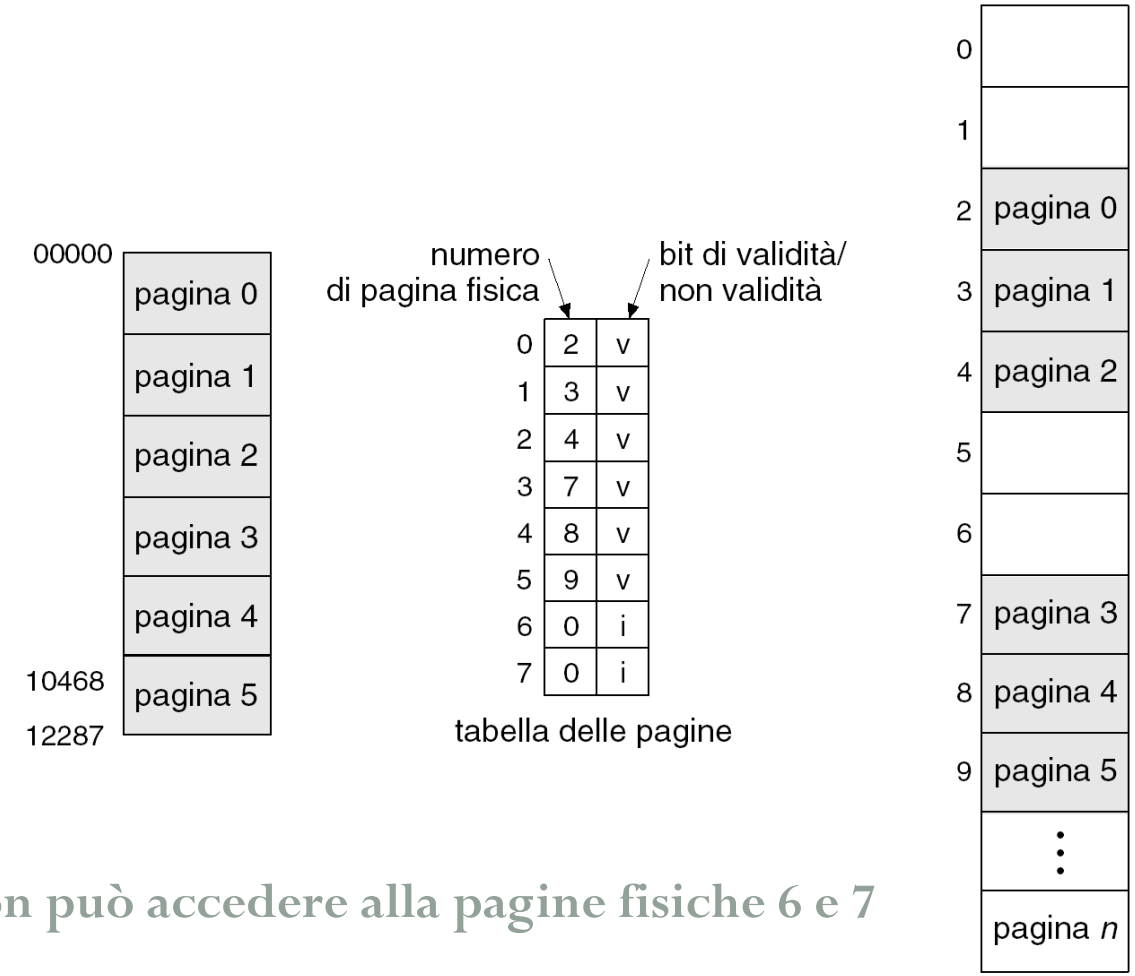
- Si ha un rallentamento del 40% del tempo di accesso alla memoria centrale (da 100 a 140 ns)
  - più alto è l'hit ratio α , più piccolo è il rallentamento

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 viene associato:
  - 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 (può essere esteso per definire diversi permessi: 3 bit per lettura, scrittura, esecuzione,…)

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



Il processo non può accedere alla pagine fisiche 6 e 7

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

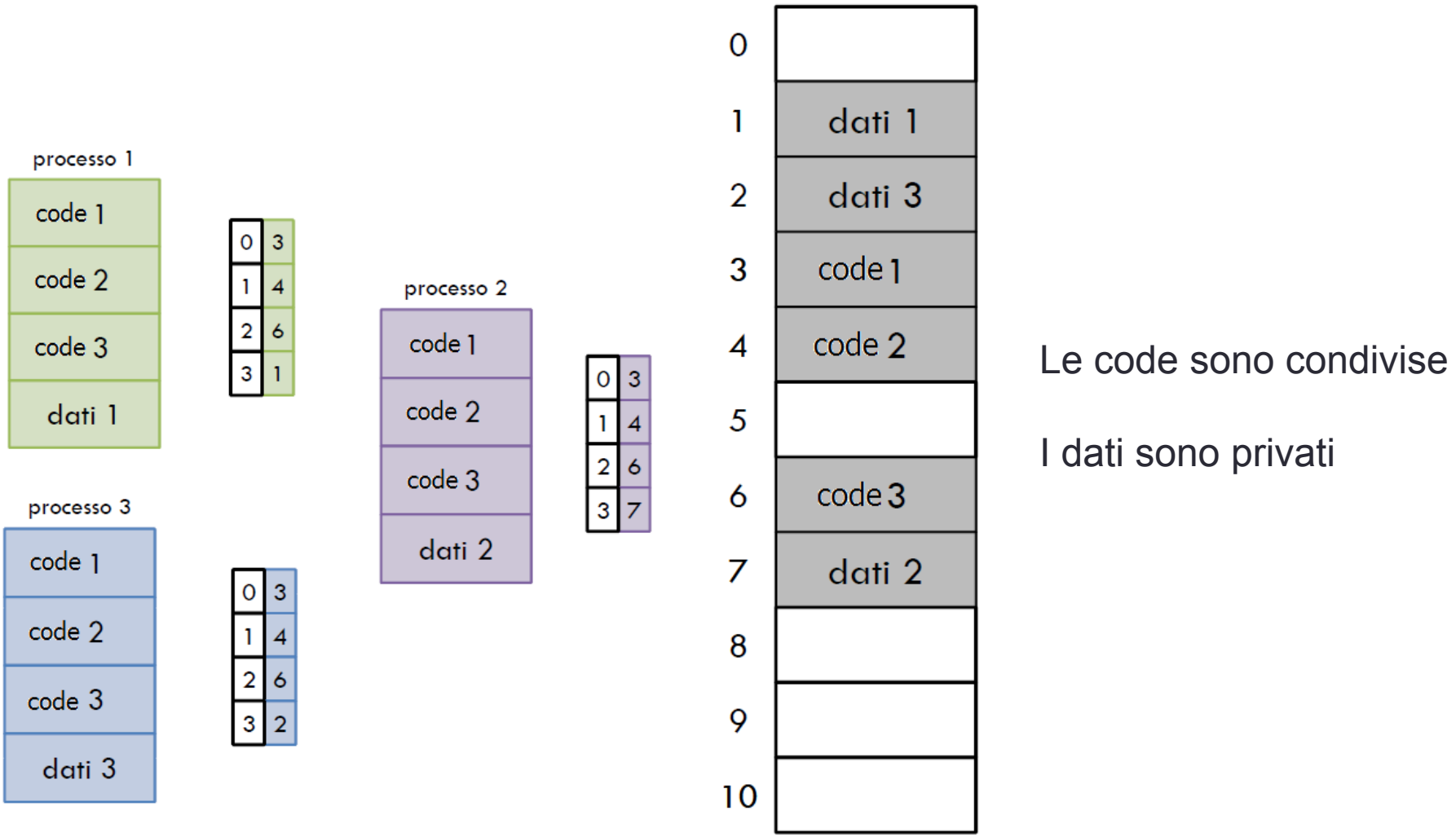
Paginazione: pagine condivise

- In un ambiente multiutente, due processi potrebbero eseguire lo stesso codice: ad esempio una libreria oppure un editor di testi
- La paginazione permette di **condividere** facilmente codice tra diversi processi (codice puro **o** rientrante)
  - Questa opzione è possibile perché il codice rientrante non cambia durante l'esecuzione del processo
  - Ad *esempio*, una pagina condivisa può essere usata per contenere il codice di una libreria dinamica usata contemporaneamente da più processi

Paginazione: codice condiviso/privato

- **Codice condiviso**
  - Una copia di sola lettura di codice rientrante condiviso fra processi (ad esempio compilatori, sistemi a finestre)
  - Il codice condiviso deve apparire nella stessa posizione dello spazio di indirizzamento logico di tutti i processi
- **Codice privato e dati**
  - Ogni processo possiede una copia separata del codice e dei dati
  - Le pagine per il codice privato ed i dati possono apparire ovunque nello spazio di indirizzamento logico

Paginazione: esempio di codice condiviso



Le code sono condivise

I dati sono privati

Struttura dati per la tabella delle pagine

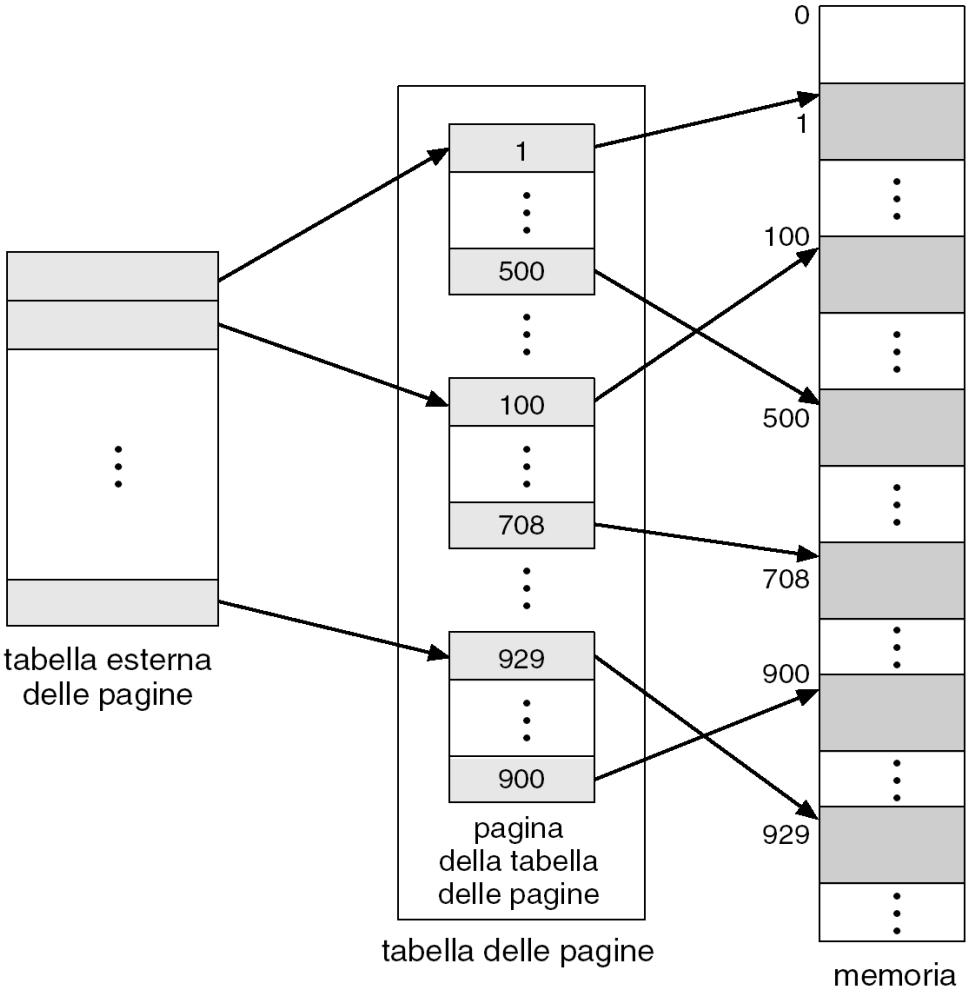
- Paginazione gerarchica
- Tabelle delle pagine con hashing
- Tabella delle pagine invertita

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 ($2^{32} / 2^{12} = 2^{20} = 1{,}048{,}576$)
  - Se ogni elemento occupa 4 Byte si ha una tabella da 4MB per ogni processo

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

Schema di tabella delle pagine a due livelli

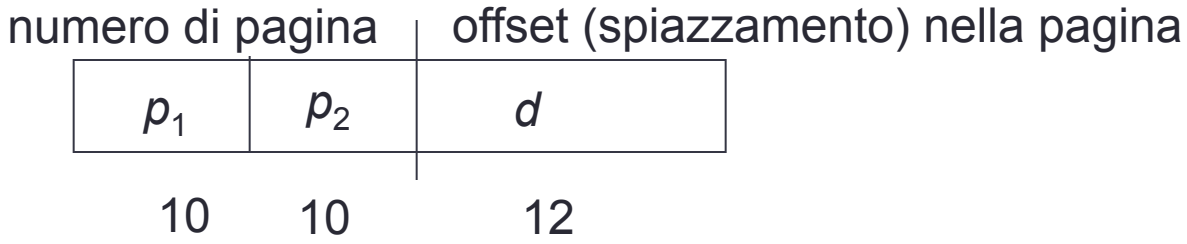


- Una directory delle pagine detta **tabella esterna**
  - ogni elemento punta ad una sotto-tabella delle pagine
- Un insieme **di sotto-tabelle delle pagine**
  - Tipicamente una sotto-tabella delle pagine è ampia quanto una pagina al fine di essere completamente contenuta in un frame

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

- Poiché 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
- Nota anche come **tabella delle pagine mappata in avanti** (forward-mapped page table)

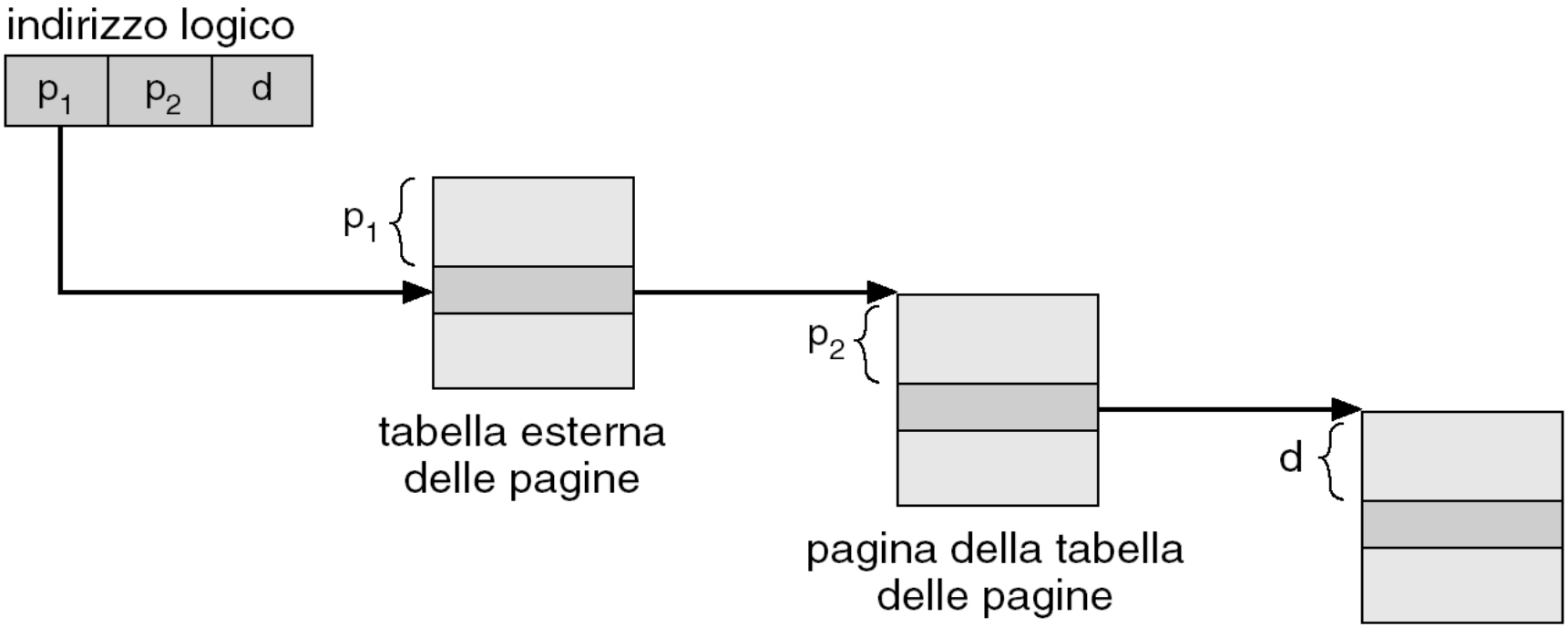


Tabelle a più livelli

- 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**

Tabelle delle pagine con hashing

- Comune per trattare gli spazi di indirizzamento più grandi di 32 bit
  - **Chiave**: hash dell'indirizzo virtuale
  - **Valore**: una lista di elementi formati da (a) indirizzo virtuale, (b) indirizzo della pagina fisica, (c) puntatore al prossimo elemento
- I numeri di pagina virtuali sono confrontati con il campo (a) degli elementi della lista. Se viene trovata una corrispondenza, il corrispondente frame fisico viene estratto.

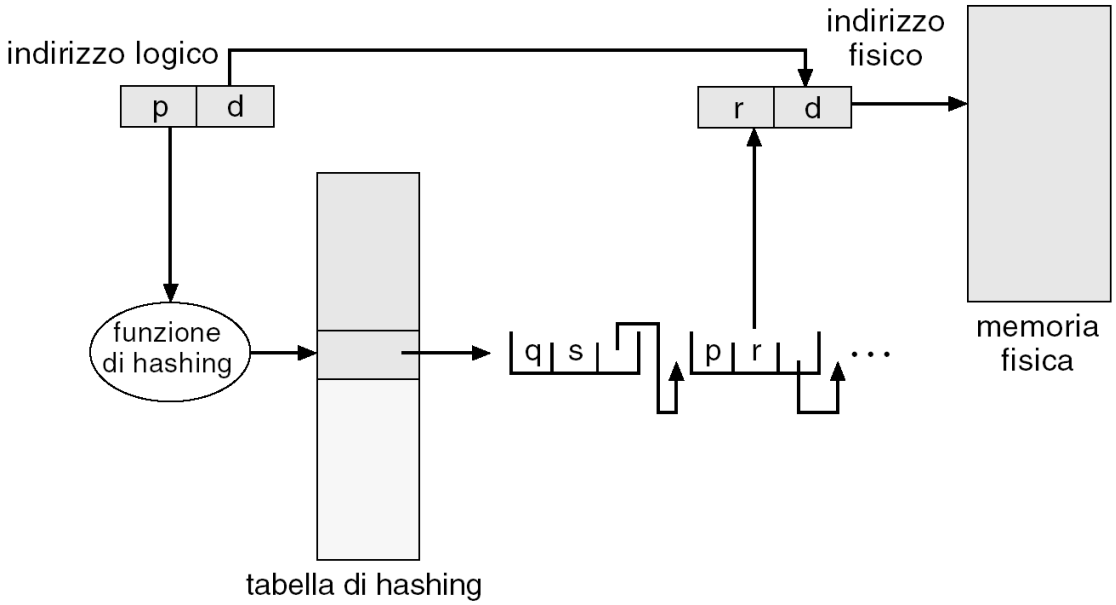
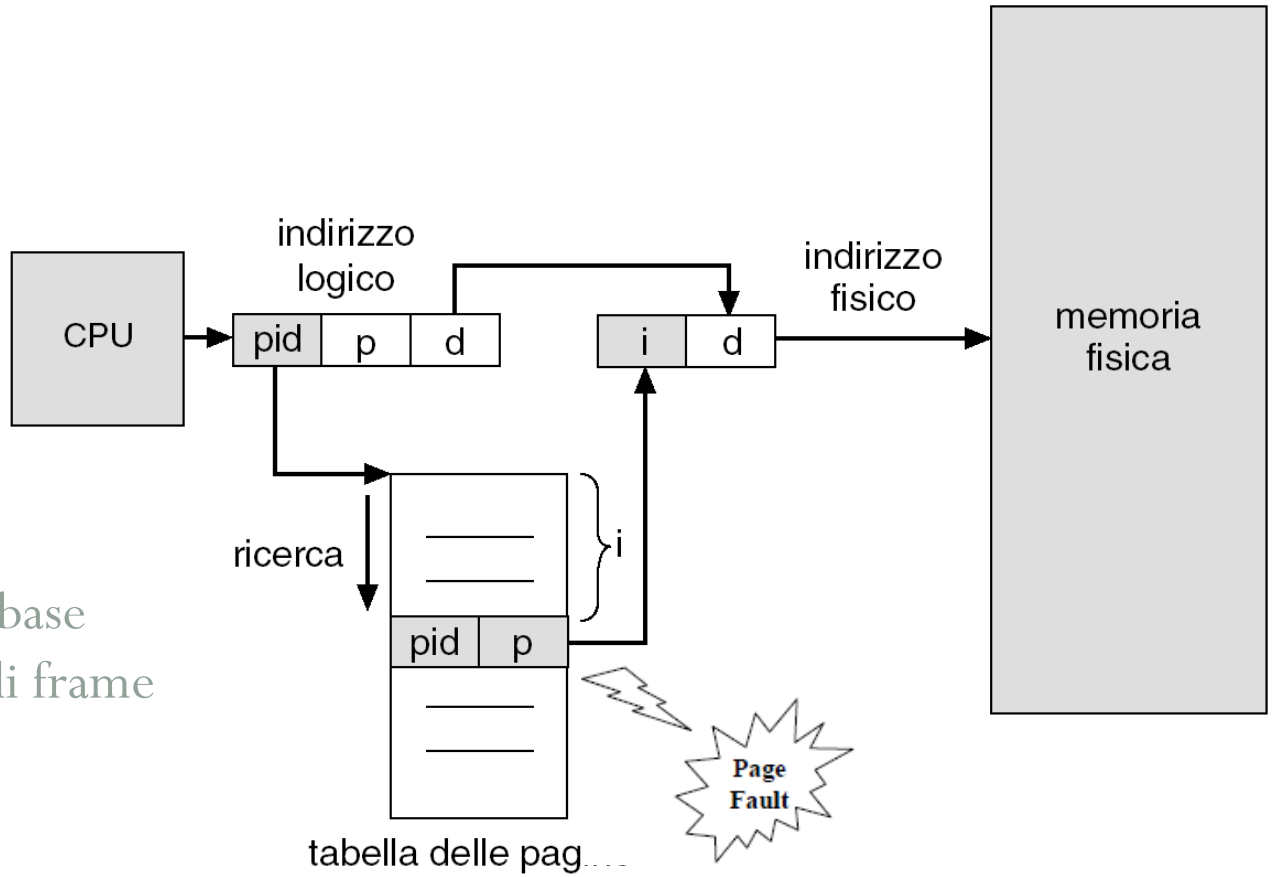


Tabella delle pagine invertita (IPT)

- 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), altrimenti un page-fault



IPT ordinata in base al numero "i" di frame

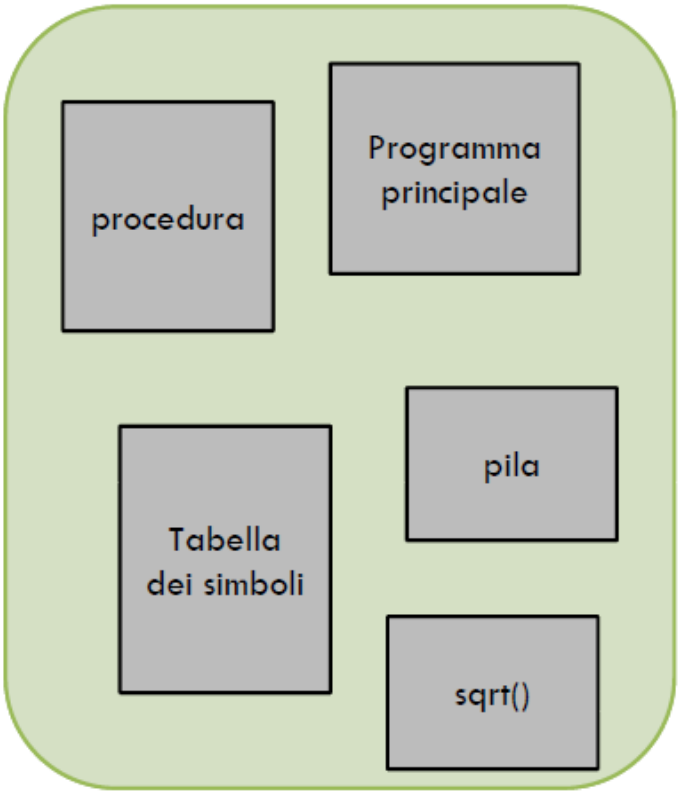
Tabella delle pagine invertita (IPT) Vantaggi e Svantaggi

- Con questo schema **si risparmia spazio, ma si perde in efficienza per cercare nella IPT** l'entry che contiene la coppia (*pid, p)*
  - Poiché la tabella è ordinata per indirizzo fisico e può essere necessario scorrerla tutta per trovare l'indirizzo logico desiderato

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

Segmentazione

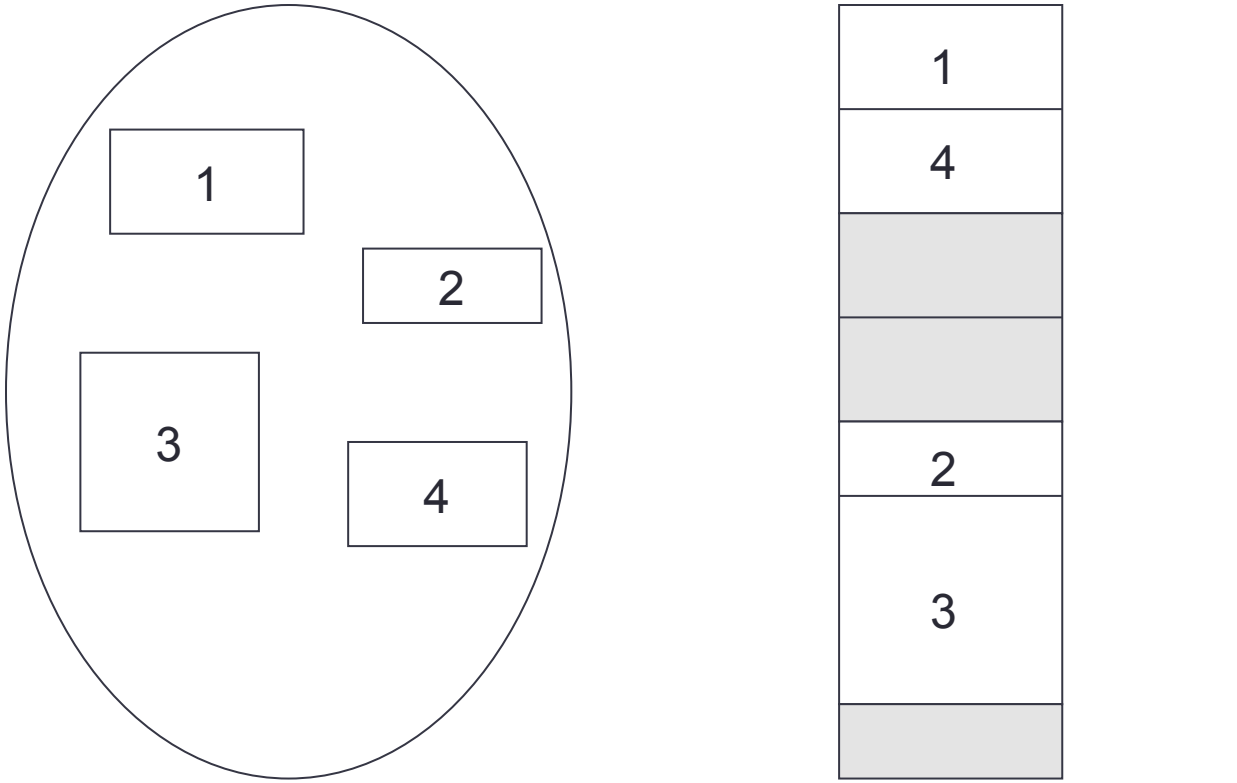
- Invece di dividere la memoria in blocchi anonimi (pagine)…
- Un programma può essere visto come una **collezione di diverse entità** così come percepite dall'utente
  - Codice, dati statici, dati locali alle procedure, stack…



- 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

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



Spazio dell'utente

Spazio della memoria fisica

Segmentazione pura

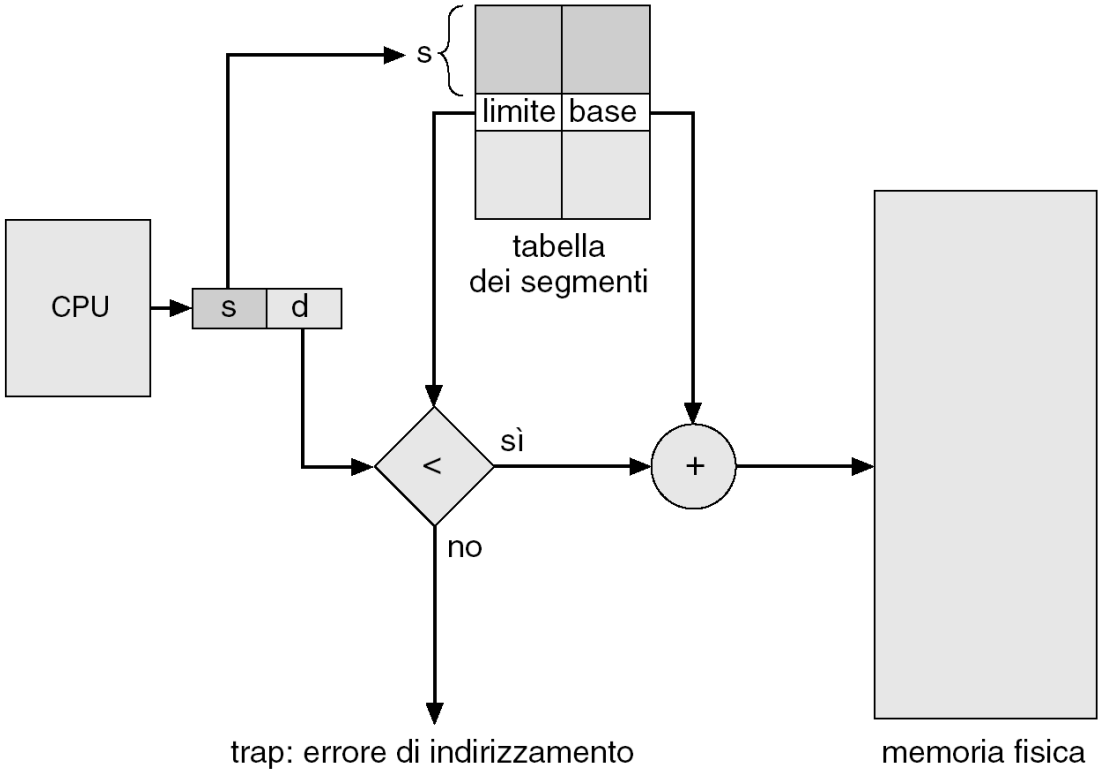
- Lo spazio di indirizzamento logico è **un insieme di segmenti di dimensione variabile**
- **I segmenti sono generati automaticamente dal compilatore**; ad es. un compilatore C crea i segmenti:
  - Il codice
  - Variabili globali
  - Heap, da cui si alloca la memoria
  - Variabili locali statiche di ogni funzione o procedura
  - Librerie standard del C
- Ogni indirizzo logico consiste di due parti
  - Numero del segmento
  - Offset nel segmento

Architettura della segmentazione pura (1)

- **Tabella dei segmenti**: mappa gli indirizzi logici in indirizzi fisici

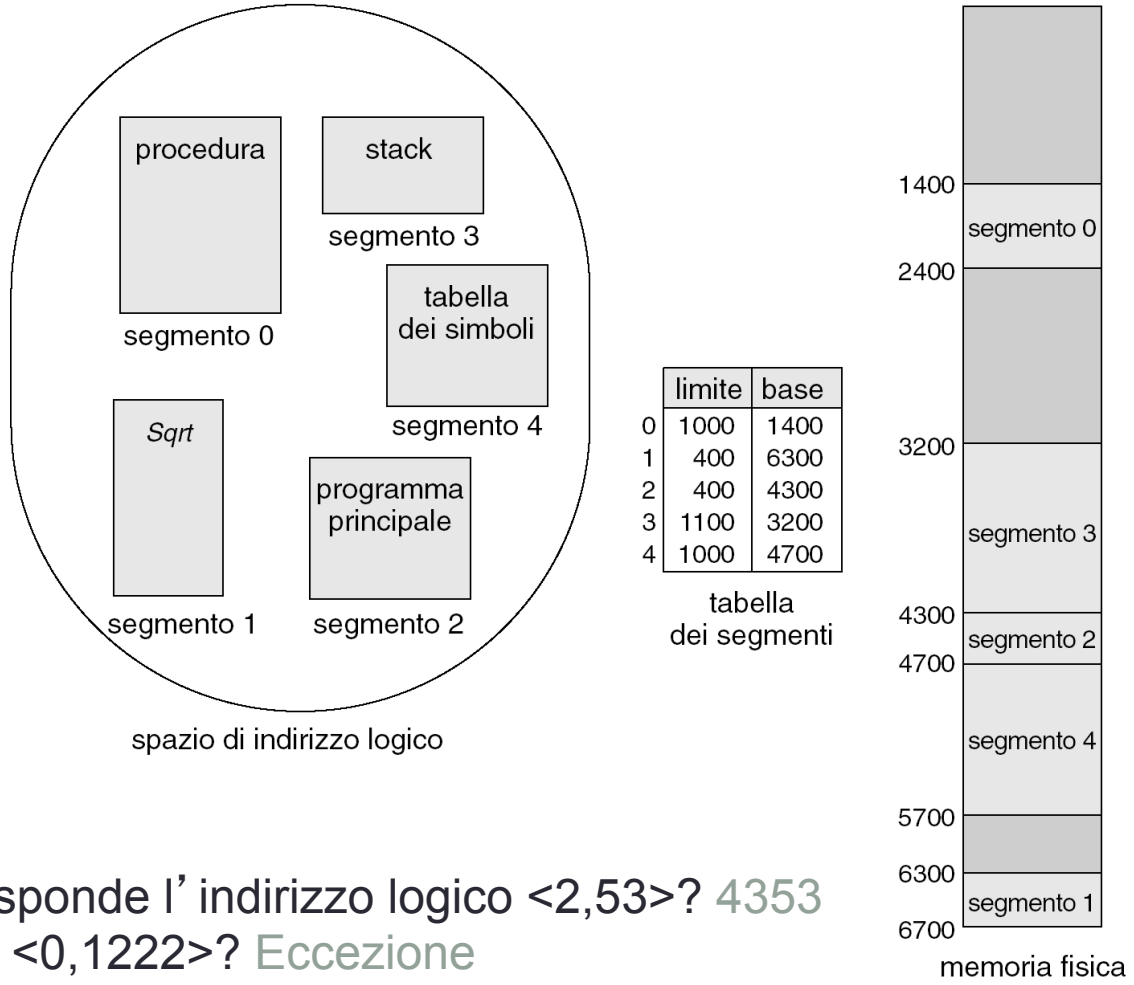
- Ogni elemento della tabella ha:
  - Una **base**: contiene l'indirizzo fisico di partenza in cui il segmento risiede in memoria centrale
  - Un **limite**: specifica la lunghezza del segmento stesso

Architettura della segmentazione pura (2)



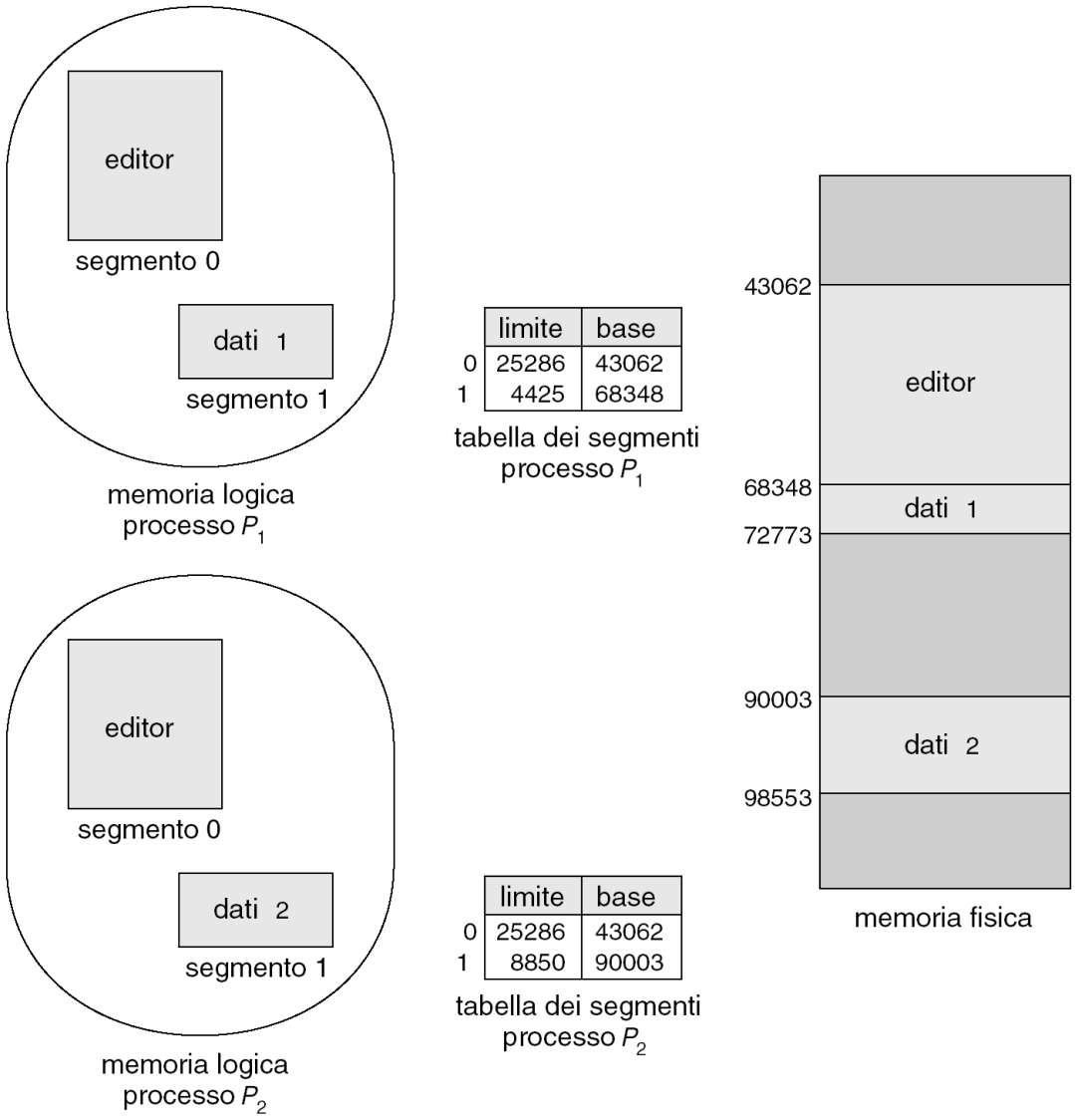
- Dato un indirizzo logico <s,d> (num. Seg, scostamento)
  - Si usa s come indice della tabella per ricavare l'indirizzo base
  - Si verifica che d < limite
    - Vero: indirizzo fisico = base + d
    - Falso: si solleva un'eccezione

Architettura della segmentazione pura (3)



A cosa corrisponde l'indirizzo logico <2,53>? 4353
E l'indirizzo <0,1222>? Eccezione

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**

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

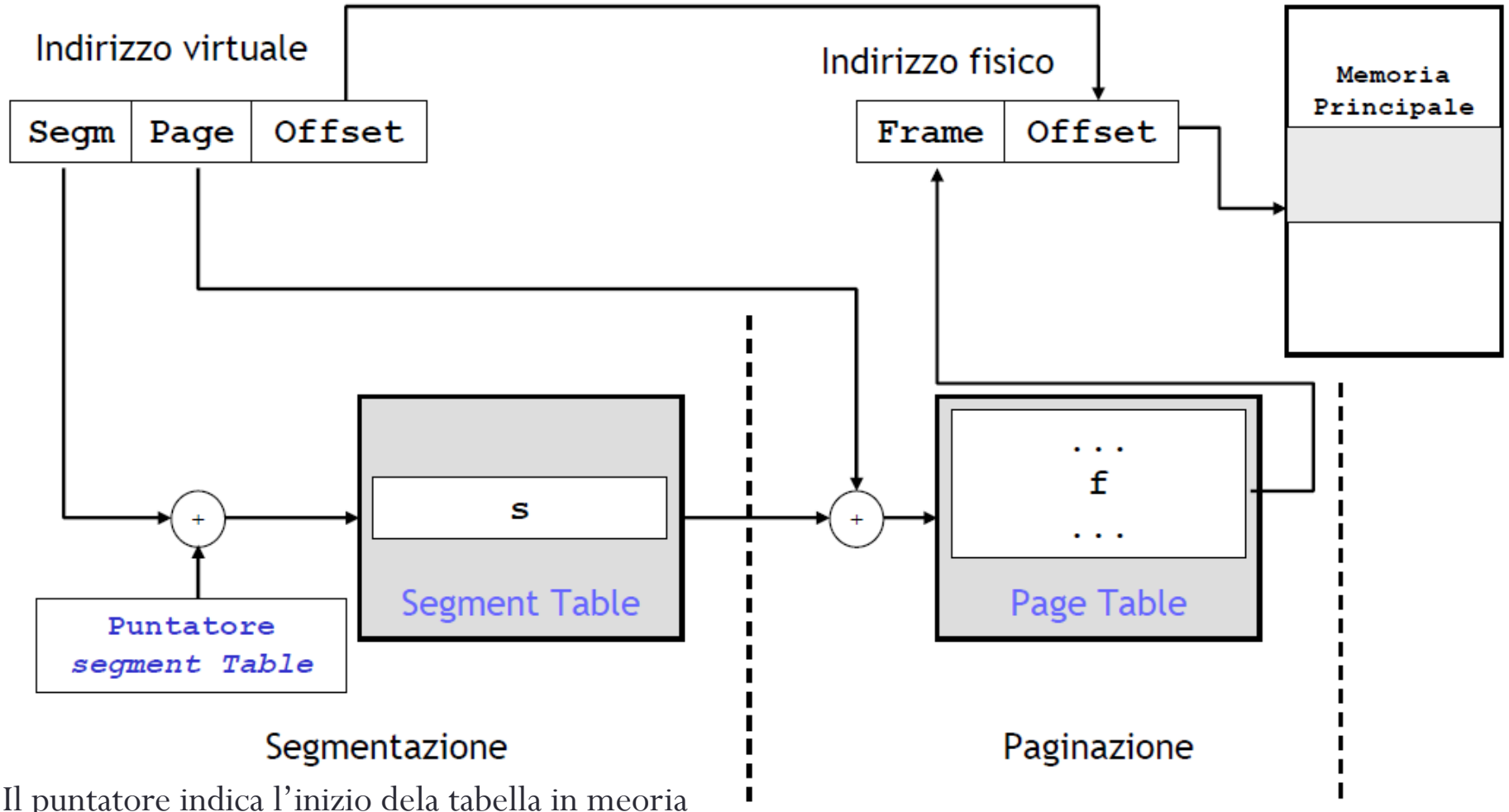
Segmentazione con paginazione

- Risolve i problemi della frammentazione esterna e della lunghezza dei tempi di ricerca attraverso la **paginazione dei segmenti**
  - le informazioni della tabella dei segmenti non contengono l'indirizzo di un segmento
  - quanto piuttosto *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: traduzione degli indirizzi



Il puntatore indica l'inizio dela tabella in meoria

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