Gestione della memoria centrale

Programma – Sistemi Operativi

- Introduzione ai sistemi operativi
- Gestione dei processi
- Sincronizzazione dei processi
- Gestione della memoria centrale
- Gestione della memoria di massa
- File system
- Sicurezza e protezione

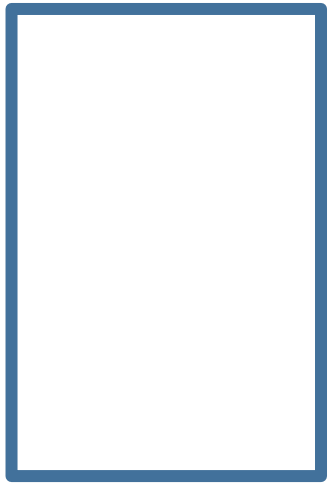
Esecuzione di un programma

Durante l'esecuzione, i programmi e i dati cui essi accedono devono trovarsi, almeno parzialmente, in memoria centrale.

La scelta di un metodo di gestione della memoria, per un sistema specifico, dipende da molteplici fattori, in particolar modo dall'*architettura hardware*.

Memoria centrale

- ROM (Read-Only Memory - memoria a sola lettura) è una memoria non volatile in grado di mantenere memorizzati i dati anche in assenza di alim
- RAM (Random-Access Memo è una memoria volatile, cioè allo spegnimento del compu
- CACHE (dal termine francese un tipo di memoria volatile c più veloce e più costosa di qu



Spazio di memoria separato

- Ciascun processo deve avere uno spazio di memoria separato, in modo da proteggere i processi l'uno dall'altro.

- Questo è fondamentale per avere più processi caricati in memoria per l'esecuzione concorrente.

Spazio di memoria protetto

- Bisogna proteggere il sistema operativo dall'accesso dei processi utenti e, in sistemi multiutente, salvaguardare i processi utenti uno dall'altro.

- Tale protezione è implementata a livello hardware poiché il sistema operativo (per questioni di prestazioni) non interviene negli accessi della CPU alla memoria.

Registro base e registro limite

Si può implementare il meccanismo di protezione tramite due registri, detti **registro base** e **registro limite**

Registro base contiene il più piccolo indirizzo legale della memoria fisica (per es. 300040)

Registro limite determina la dimensione dell'intervallo ammesso (per es. 120900)

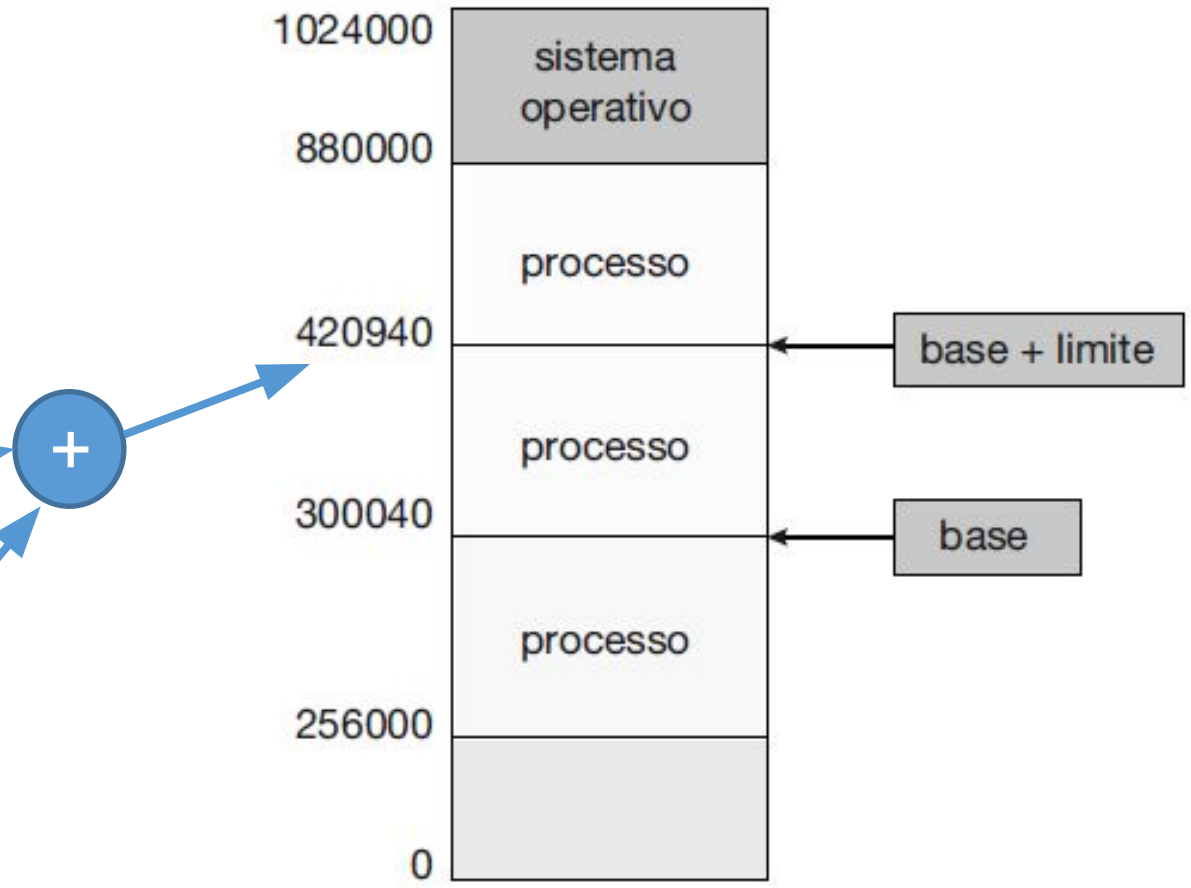


Figura 9.1 I registri base e limite definiscono lo spazio degli indirizzi logici.

Protezione delle aree di memoria

Qualsiasi tentativo da parte di un programma eseguito in modalità utente di accedere alle aree di memoria riservate al sistema operativo o a una qualsiasi area di memoria riservata ad altri utenti comporta l'invio di una **eccezione** (*trap*) che restituisce il controllo al sistema operativo che, a sua volta, interpreta l'evento come un errore fatale.

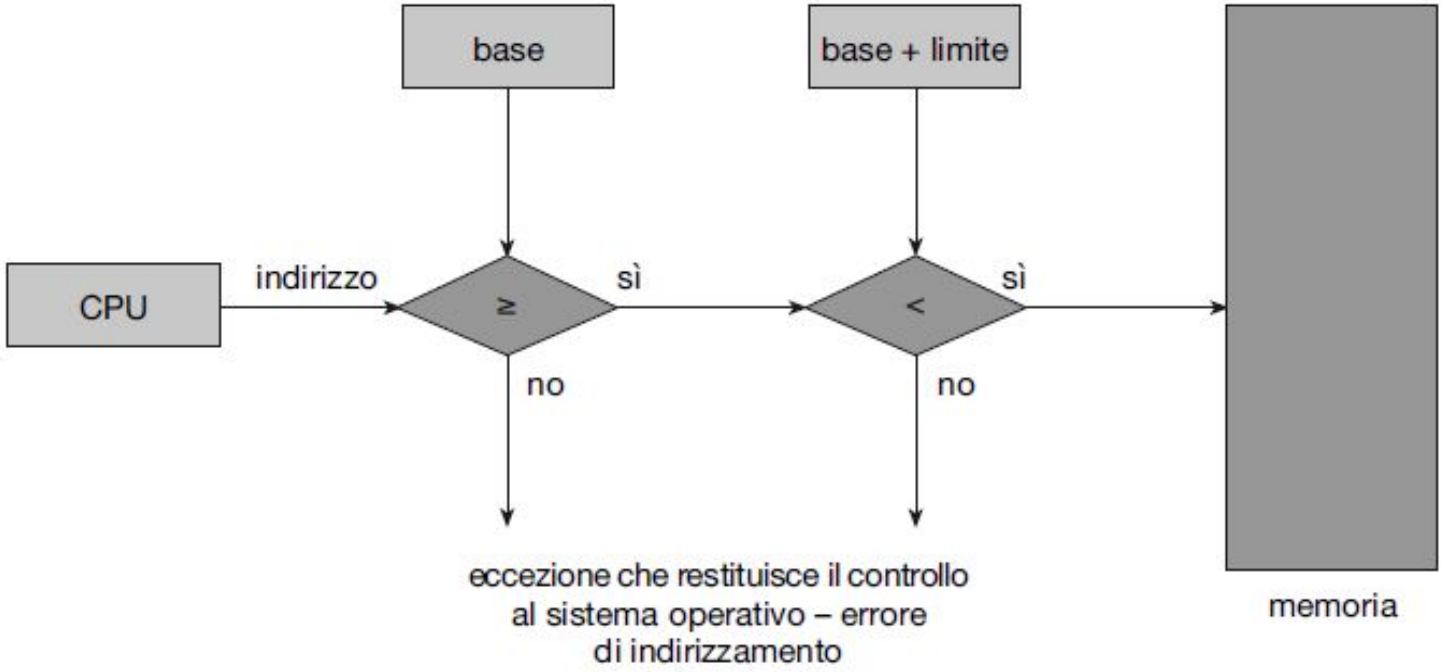


Figura 9.2 Protezione hardware degli indirizzi tramite registri base e limite.

Associazione degli indirizzi

Nella maggior parte dei casi un programma utente, prima di essere eseguito, deve passare attraverso *varie fasi*, alcune delle quali possono essere facoltative

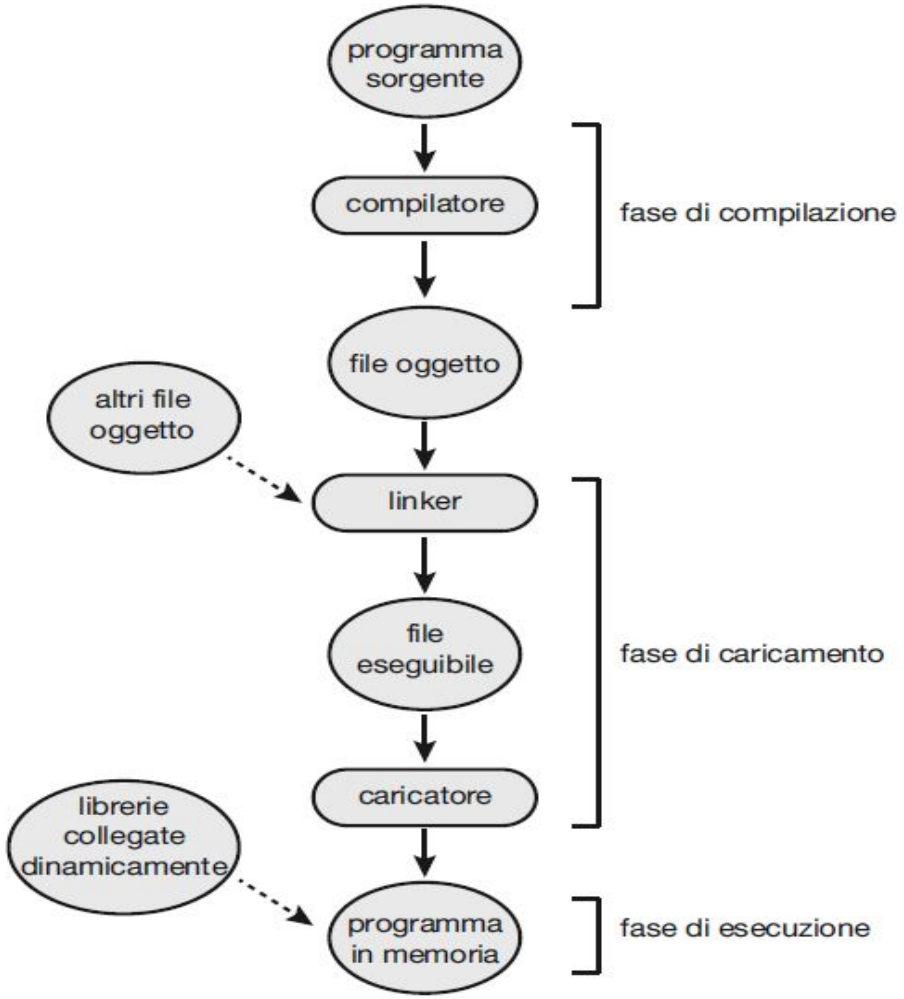


Figura 9.3 Fasi di elaborazione di un programma utente.

Associazione degli indirizzi

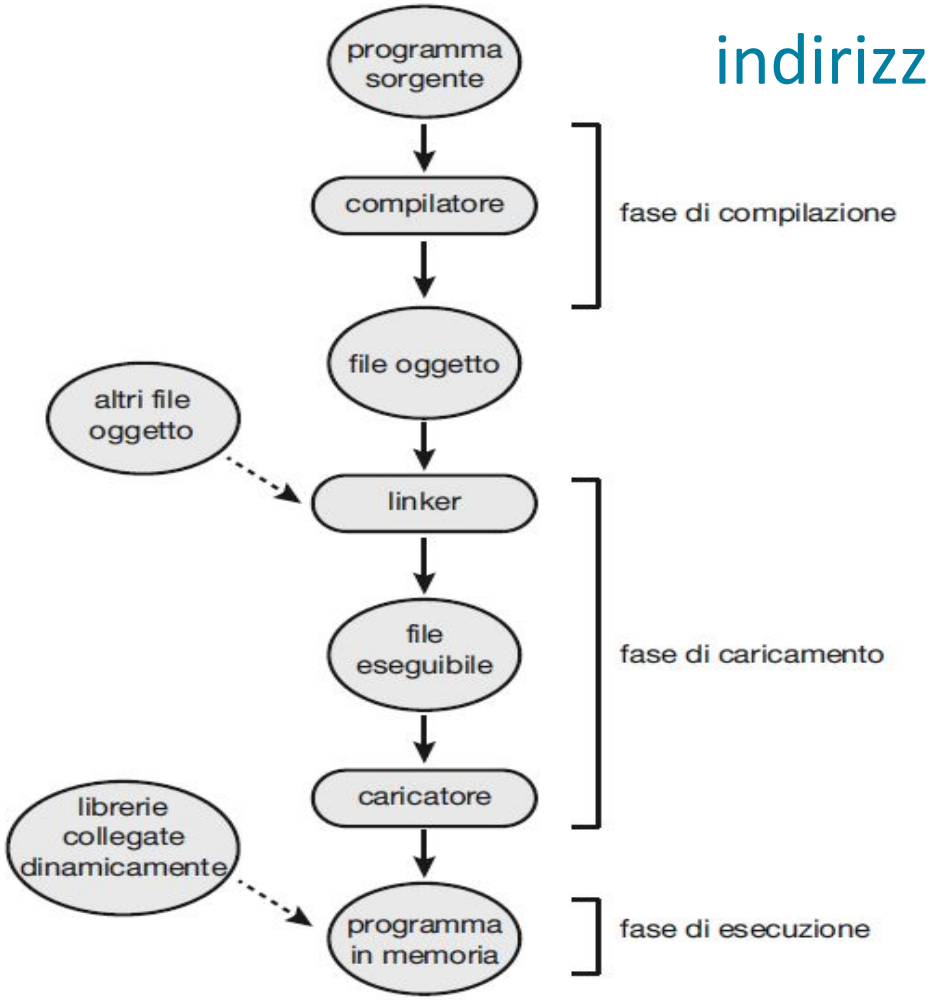


Figura 9.3 Fasi di elaborazione di un programma utente.

indirizzi simbolici (es. variabile `count`)

binding

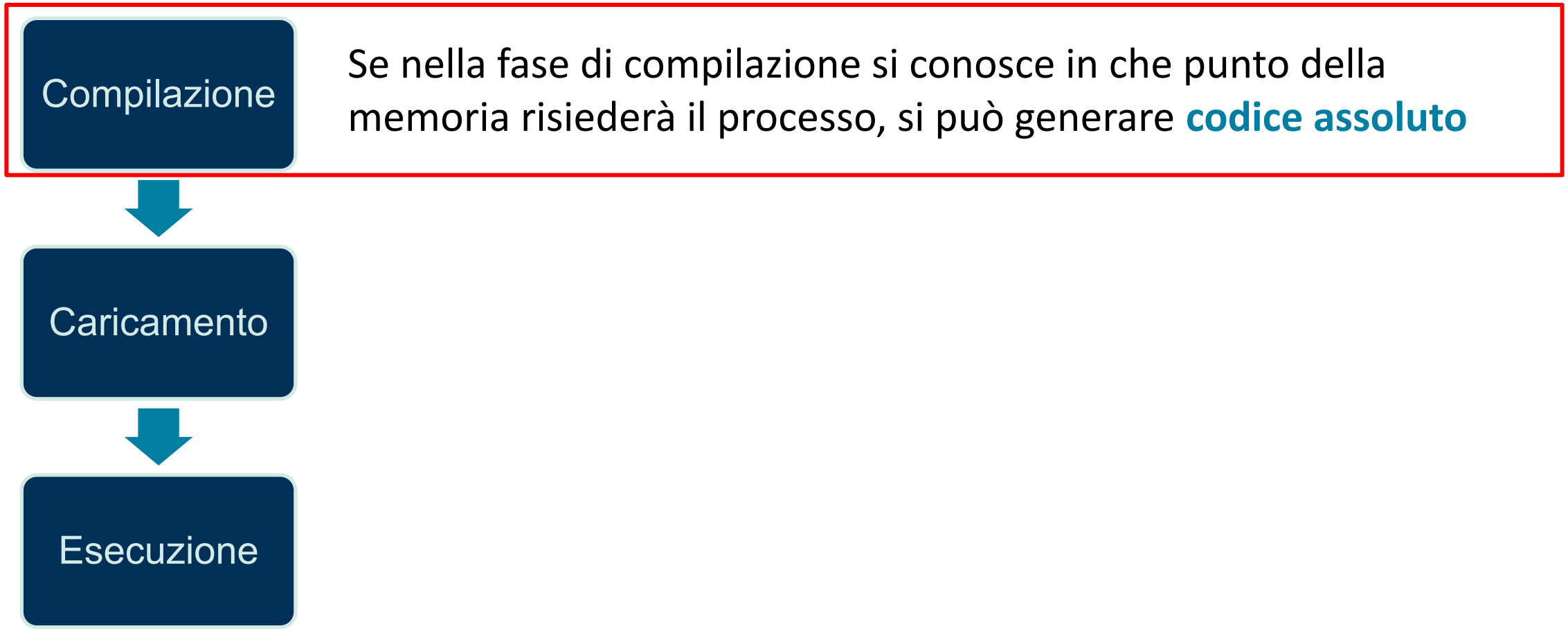
indirizzi rilocabili

indirizzi assoluti

Associazione degli indirizzi

Generalmente, l'associazione di istruzioni e dati a indirizzi di memoria si può compiere in qualsiasi *fase* del seguente percorso.

Codice assoluto



Se nella fase di compilazione si conosce in che punto della memoria risiederà il processo, si può generare **codice assoluto**

Codice rilocabile

Se nella fase di compilazione non è possibile conoscere in che punto della memoria risiederà il processo, il compilatore deve generare **codice rilocabile**

Associazione a runtime

Se durante l'esecuzione il processo può essere spostato da un segmento di memoria ad un altro, si deve **ritardare** l'associazione degli indirizzi fino alla fase di esecuzione

Spazio di indirizzi

indirizzo logico → indirizzo generato dalla CPU

indirizzo fisico → indirizzo caricato nel **registro dell'indirizzo di memoria**

indirizzi virtuali → perché con il metodo di associazione nella fase di esecuzione gli indirizzi logici non coincidono con gli indirizzi fisici

Unità di gestione della memoria

L'**unità di gestione della memoria** (*memory management unit, **MMU***) svolge l'associazione nella fase d'esecuzione dagli indirizzi virtuali agli indirizzi fisici.

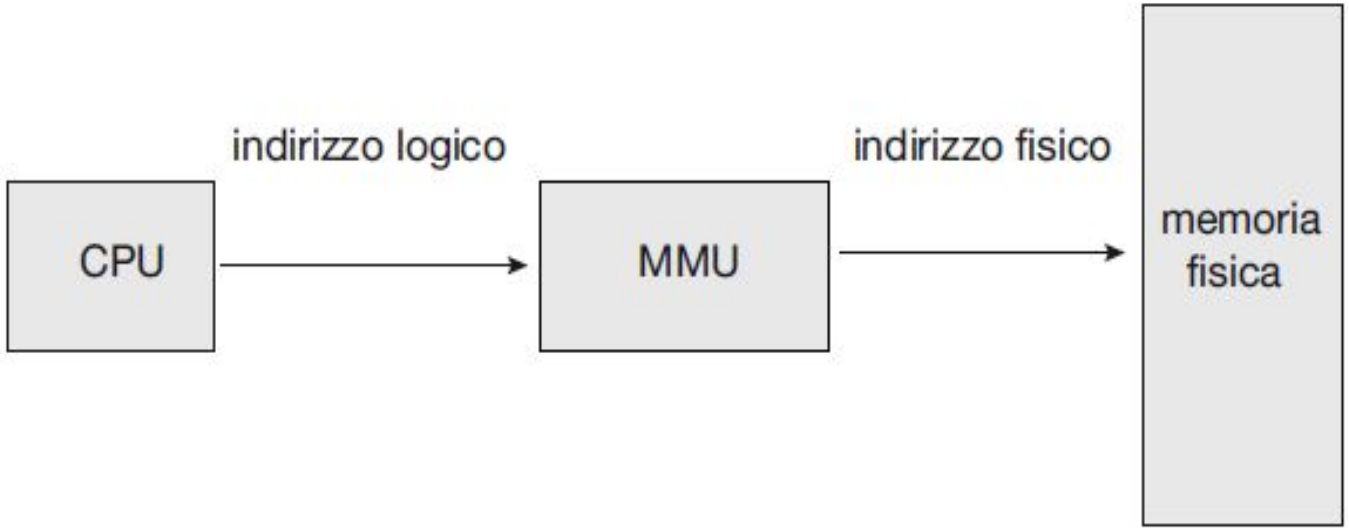


Figura 9.4 Unità di gestione della memoria (MMU).

Registro di rilocazione

Quando un processo utente genera un indirizzo, prima dell'invio all'unità di memoria, si somma a tale indirizzo il valore contenuto nel registro di rilocazione.

il registro di base è ora denominato **registro di rilocazione**

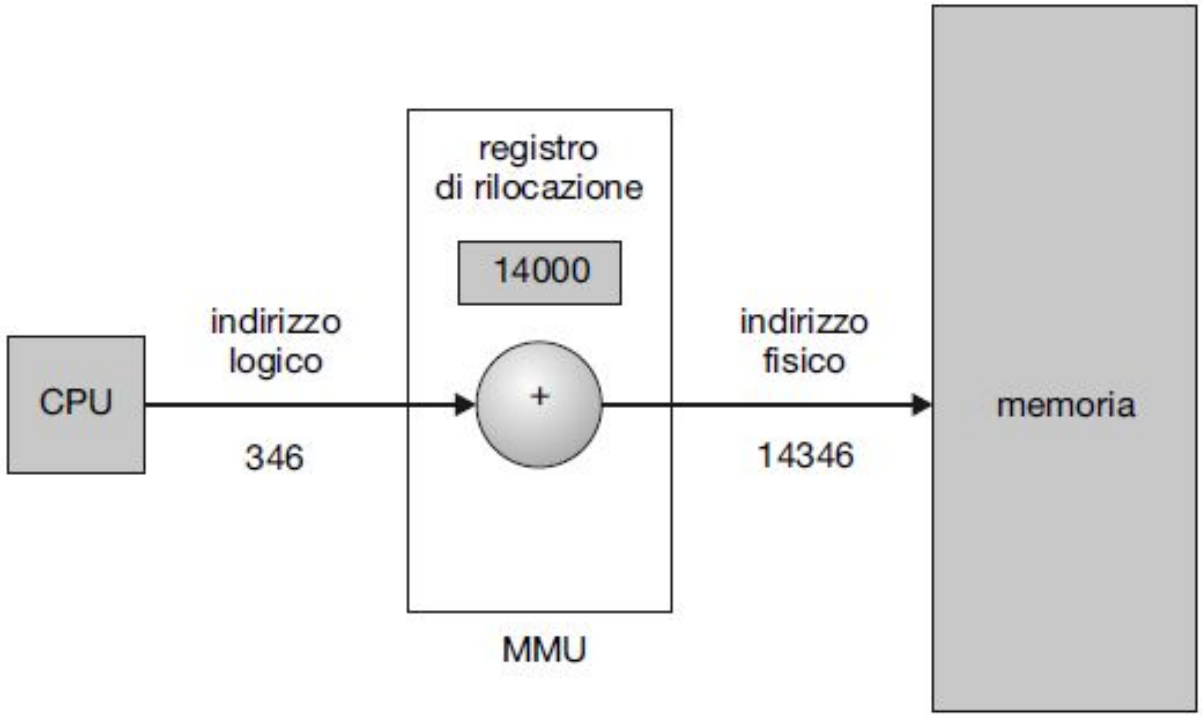


Figura 9.5 Rilocazione dinamica tramite un registro di rilocazione.

Allocazione contigua della memoria

Con l’**allocazione contigua della memoria** ciascun processo è contenuto in una singola sezione di memoria contigua a quella che contiene il processo successivo.

Protezione della memoria

Ogni indirizzo logico deve cadere nell'intervallo specificato dal registro limite

La **MMU** fa corrispondere *dinamicamente* l'indirizzo fisico all'indirizzo logico sommando a quest'ultimo il valore contenuto nel registro di rilocazione (Figura 9.6) e invia l'indirizzo risultante alla memoria

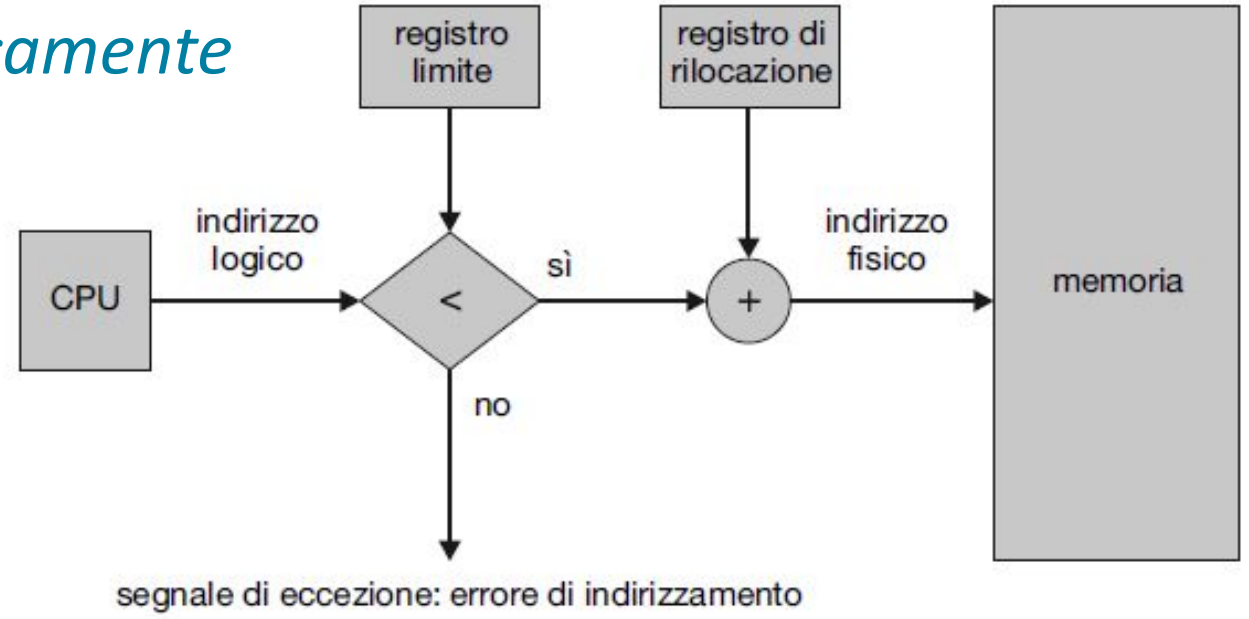


Figura 9.6 Registri di rilocazione e limite.

Vantaggi del registro di rilocazione

Lo schema con registro di rilocazione consente al sistema operativo di cambiare dinamicamente le proprie dimensioni.

Tale flessibilità è utile in molte situazioni. Per esempio, se un driver di periferica non è attualmente in uso, può essere rimosso dalla memoria.

Metodi per l’allocazione della memoria

Uno dei metodi più semplici per l’allocazione della memoria consiste nel suddividere la stessa in **partizioni di dimensione variabile**, dove ciascuna partizione può contenere esattamente un processo.

Schema a partizione variabile

Nello **schema a partizione variabile** il sistema operativo conserva una tabella in cui sono indicate le partizioni di memoria disponibili e quelle occupate.

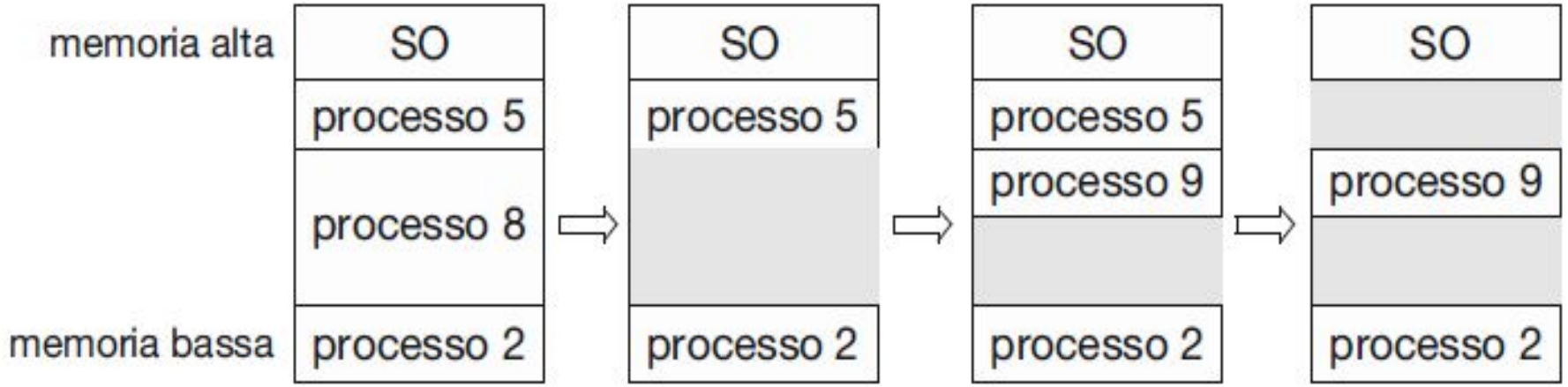


Figura 9.7 Schema a partizione variabile.

Allocazione dinamica

L'**allocazione dinamica della memoria** consente di soddisfare una richiesta di dimensione *n* data una lista di **buchi** liberi

I criteri più usati per scegliere un buco libero tra quelli disponibili nell'insieme sono i seguenti:

First-fit

Si assegna il primo buco abbastanza grande

Best-fit

Si assegna il più piccolo buco in grado di contenere il processo

Worst-fit

Si assegna il buco più grande

Problema della frammentazione

First-fit

Si assegna il primo buco abbastanza grande

Best-fit

Si assegna il più piccolo buco in grado di contenere il processo

Worst-fit

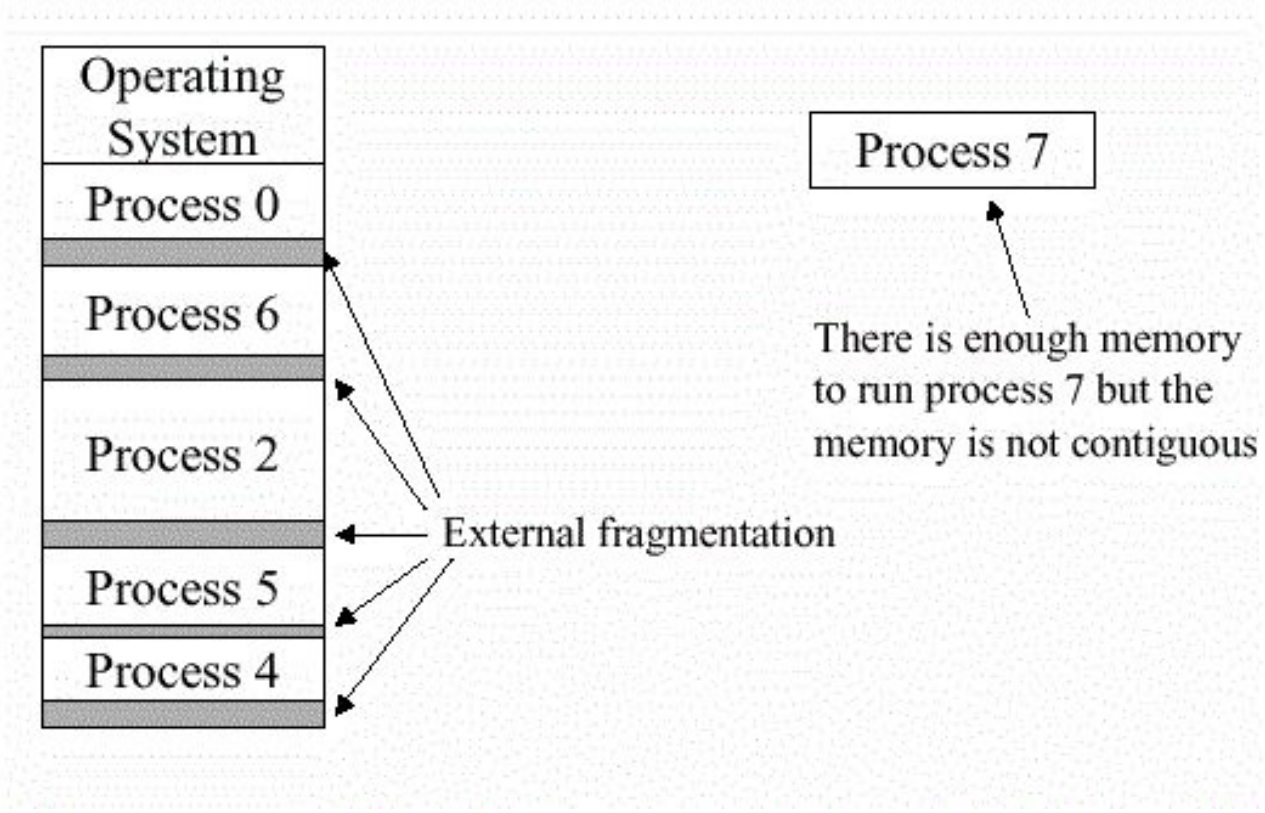
Si assegna il buco più grande

I criteri *first-fit* e *best-fit* di allocazione della memoria soffrono di **frammentazione esterna**

Frammentazione esterna

External fragmentation

Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous so it can not be used.



https://stackoverflow.com/questions/1200694/internal-and-external-fragmentation/43870017

Regola del 50%

L'analisi statistica dell'algoritmo first-fit rileva che per *n* blocchi assegnati si perdono altri *0,5n* blocchi a causa della frammentazione esterna.

Ciò significa che potrebbe essere inutilizzabile ben un terzo della memoria.

Questa caratteristica è nota come regola del 50%

Compattazione

Una soluzione al problema della frammentazione esterna è data dalla compattazione.

Si riordina il contenuto della memoria per riunire le porzioni di memoria libera in un unico grosso blocco.

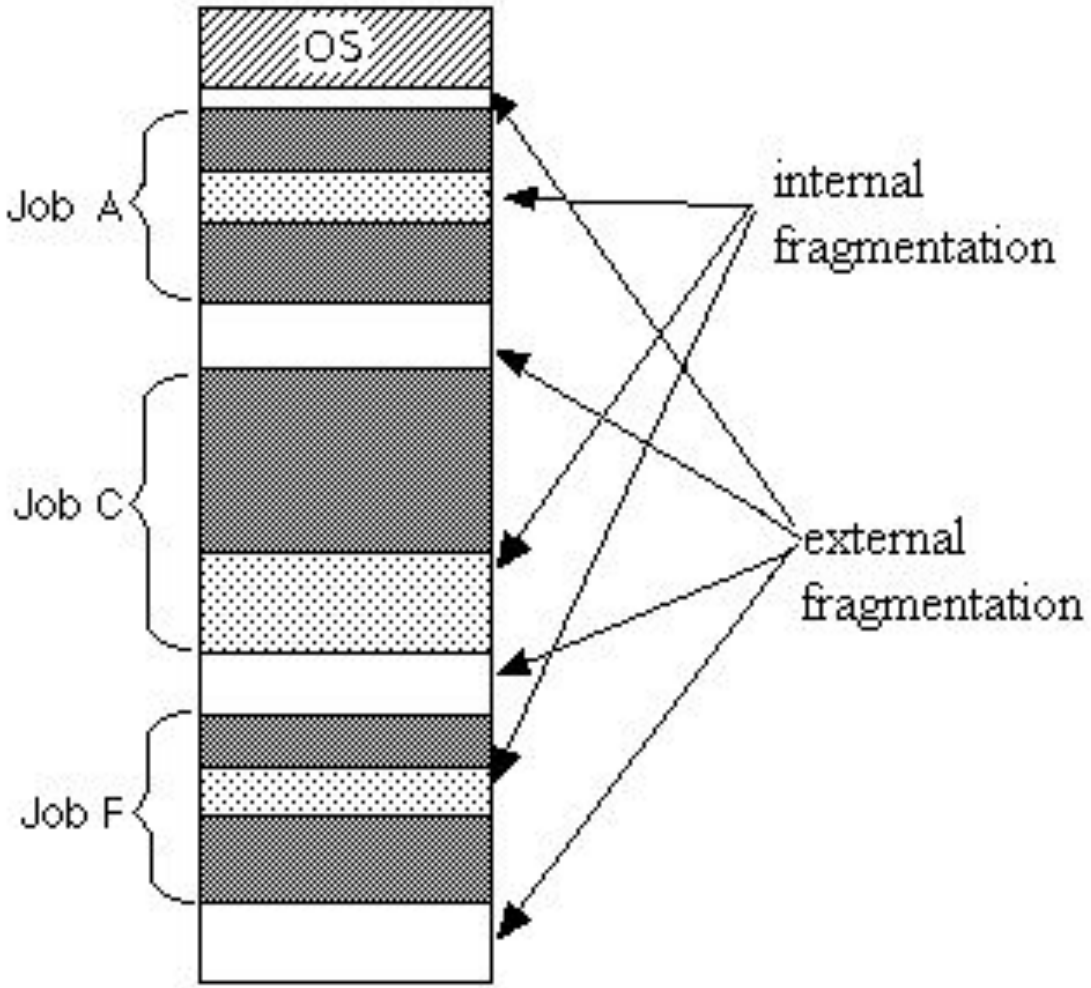
La compattazione è possibile solo se la rilocazione è dinamica e si effettua nella fase di esecuzione.

Frammentazione interna

Internal fragmentation

Memory block assigned to process is bigger.

Some portion of memory is left unused as it cannot be used by another process.



https://stackoverflow.com/questions/1200694/internal-and-external-fragmentation/43870017

Paginazione

Paginazione → schema di gestione della memoria che consente allo spazio di indirizzamento fisico di un processo di essere non contiguo

La paginazione è implementata sfruttando la cooperazione tra sistema operativo e hardware del computer

Paginazione

Paginazione → schema di gestione della memoria che consente allo spazio di indirizzamento fisico di un processo di essere non contiguo

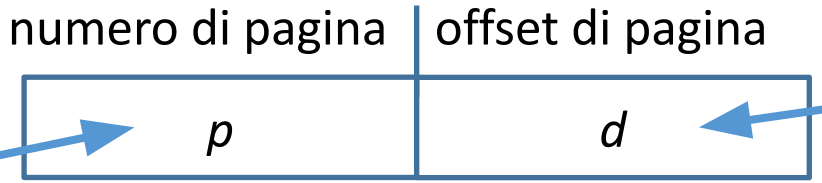
La paginazione è implementata sfruttando la cooperazione tra sistema operativo e hardware del computer

Pagine vs. Frame

Pagine
suddivisione della
memoria logica in blocchi di
pari dimensione

Frame
suddivisione della
memoria fisica in blocchi
di dimensione fissa

Ogni indirizzo generato dalla CPU è diviso in:



Indice per la tabella
delle pagine

Posizione nel frame
a cui viene fatto
riferimento

Paginazione

- La **tabella delle pagine** contiene l'indirizzo di base di ciascun frame nella memoria fisica
- l'offset è la posizione nel frame a cui viene fatto riferimento

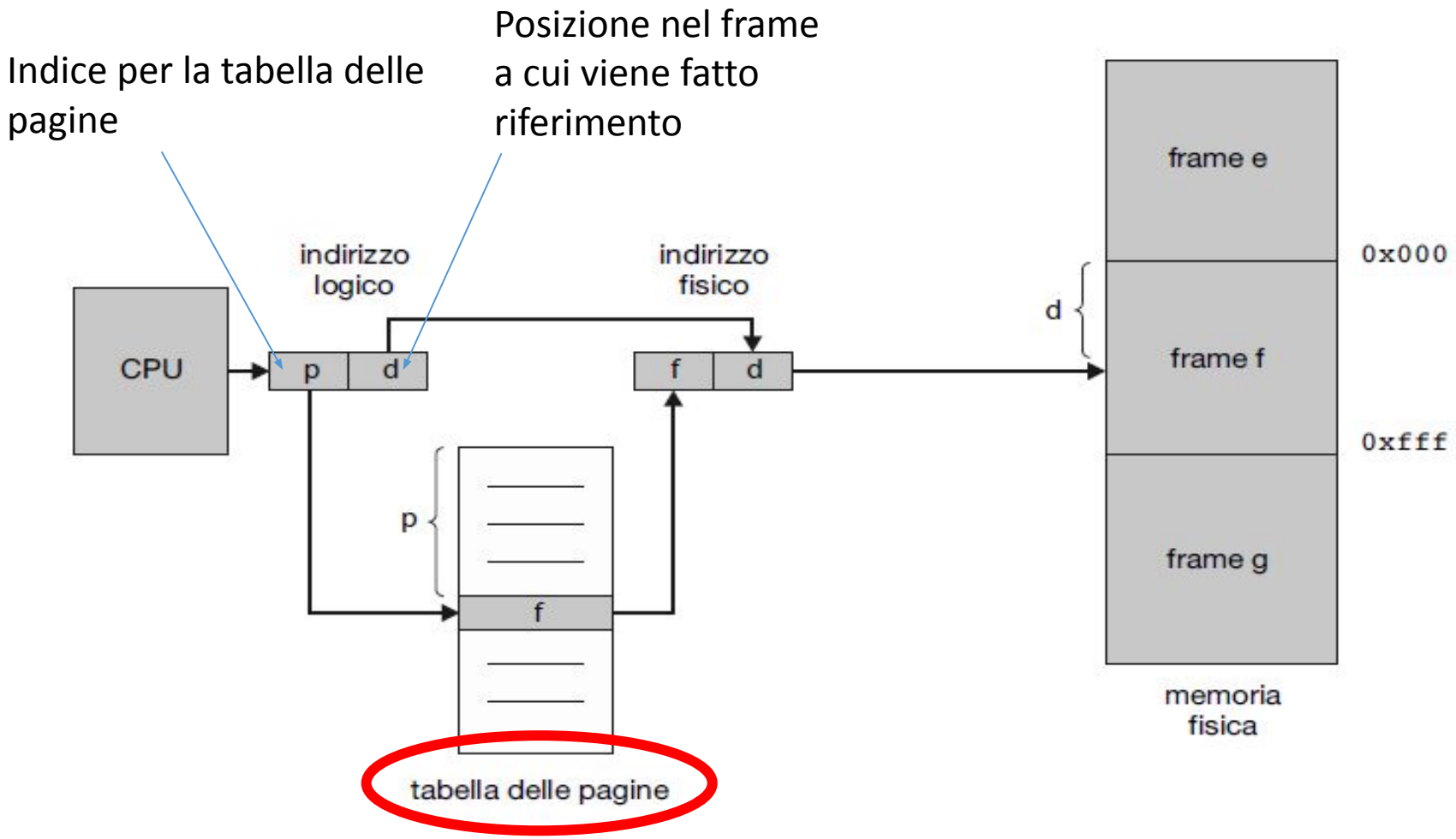


Figura 9.8 Hardware di paginazione.

Modello di paginazione della memoria

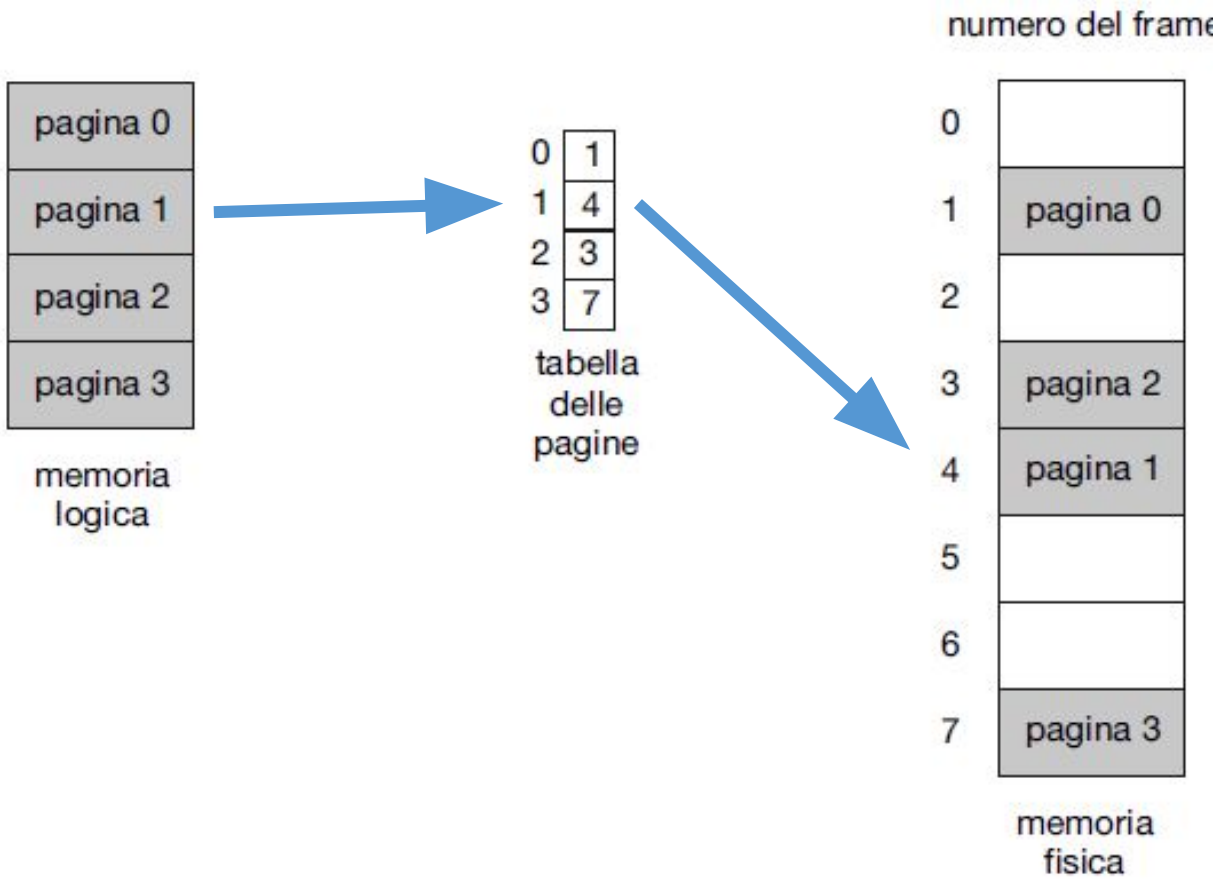


Figura 9.9 Modello di paginazione di memoria logica e memoria fisica.

Traduzione degli indirizzi

indirizzo logico di partenza 

Per tradurre un indirizzo logico in un indirizzo fisico, la MMU compie i seguenti passi:

1. Estrae il numero di pagina *p* e lo utilizza come indice nella tabella delle pagine
2. Estrae il numero di frame *f* corrispondente dalla tabella delle pagine
3. Sostituisce il numero di pagina *p* nell'indirizzo logico con il numero di frame *f*

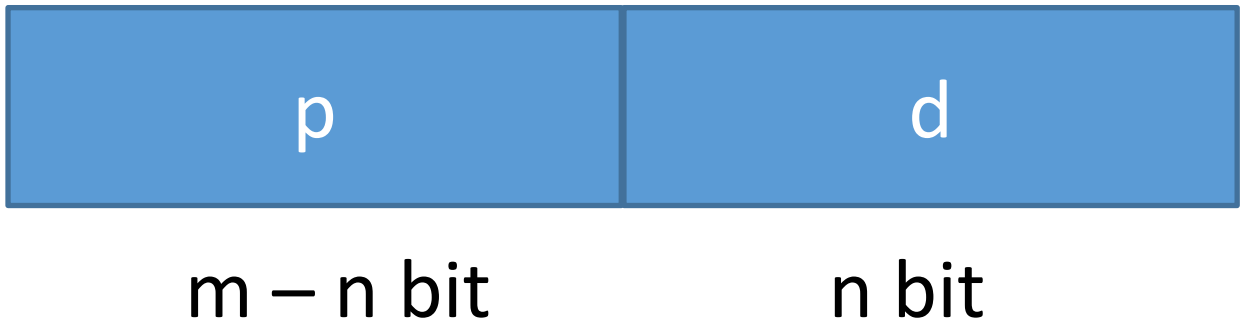
indirizzo fisico ottenuto 

Dimensione degli indirizzi

Dati

- dimensione dello spazio degli indirizzi logici: 2^m
- dimensione di una pagina: 2^n byte

Si avrà il seguente indirizzo logico



Ricapitolando: paginazione

La **paginazione** non è altro che una forma di rilocazione dinamica:
a ogni indirizzo logico l'architettura di paginazione fa corrispondere
un indirizzo fisico.

L'uso della **tabella delle pagine** è simile all'uso di una tabella di
registri base (o di rilocazione), uno per ciascun frame.

Paginazione e frammentazione

- La **paginazione** evita la frammentazione esterna. Qualsiasi frame libero può essere assegnato a un processo che ne abbia bisogno.

- La **paginazione** non evita la frammentazione interna. Tipicamente parte dello spazio nell'ultimo frame assegnato a un processo sarà sprecato.

Frame liberi

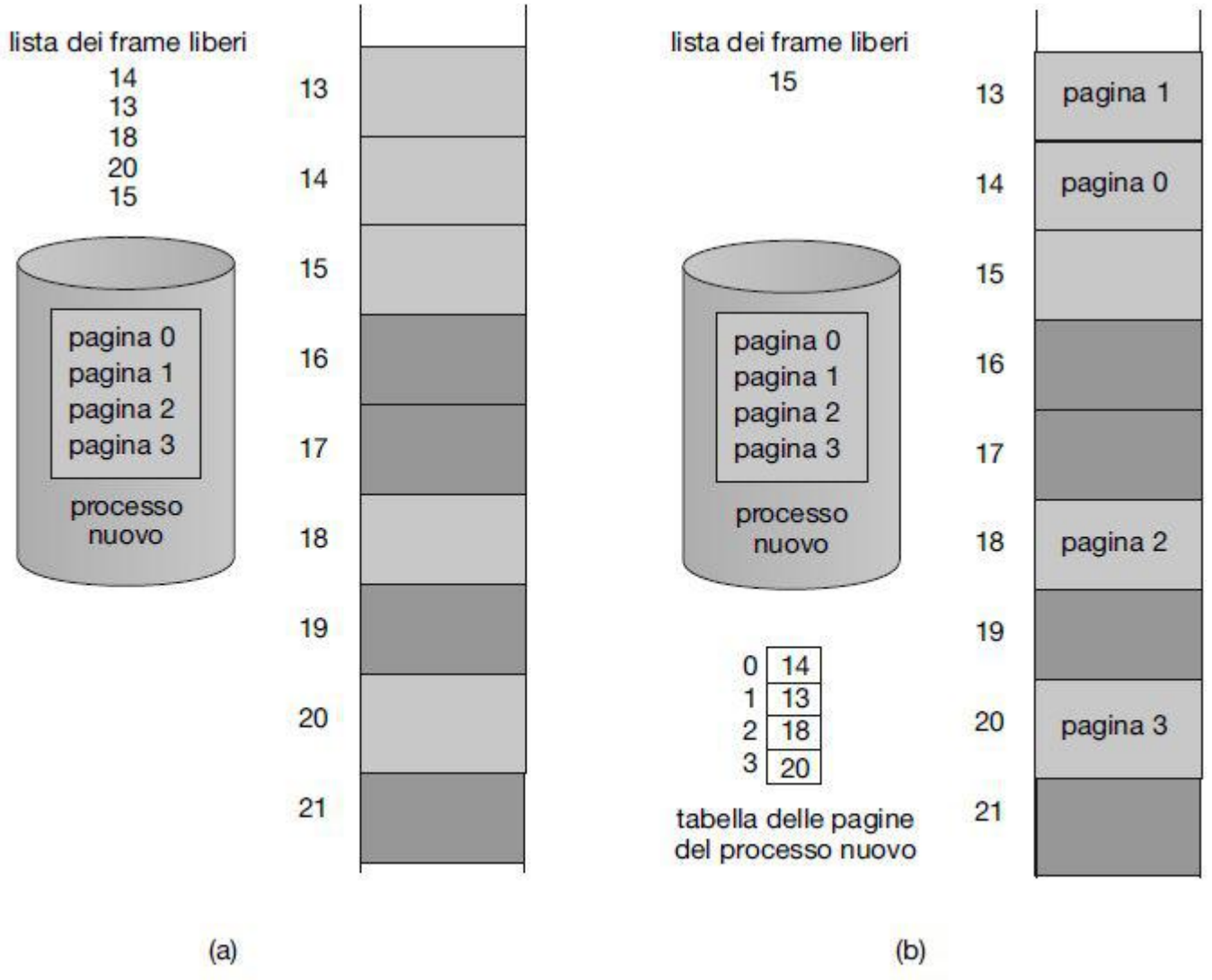


Figura 9.11 Frame liberi; (a) prima e (b) dopo l'allocazione.

Ogni processo ha la sua tabella delle pagine

TLB (supporto hardware alla paginazione)

TLB (*translation look-aside buffer)* → speciale, piccola cache hardware.
Il TLB è una memoria associativa ad alta velocità in cui ogni elemento consiste di due parti: una chiave (o *tag)* e un valore.

- il TLB contiene una piccola parte degli elementi della tabella delle pagine
- quando la CPU genera un indirizzo logico, si presenta il suo numero di pagina al TLB
- se tale numero è presente, il corrispondente numero del frame è immediatamente disponibile e si usa per accedere alla memoria

...

TLB

…

- se tale numero è presente, il corrispondente numero del frame è immediatamente disponibile e si usa per accedere alla memoria

Altrimenti

↓

Insuccesso del TLB (*TLB miss)*

↓

si deve consultare la tabella delle pagine in memoria

Paginazione con TLB

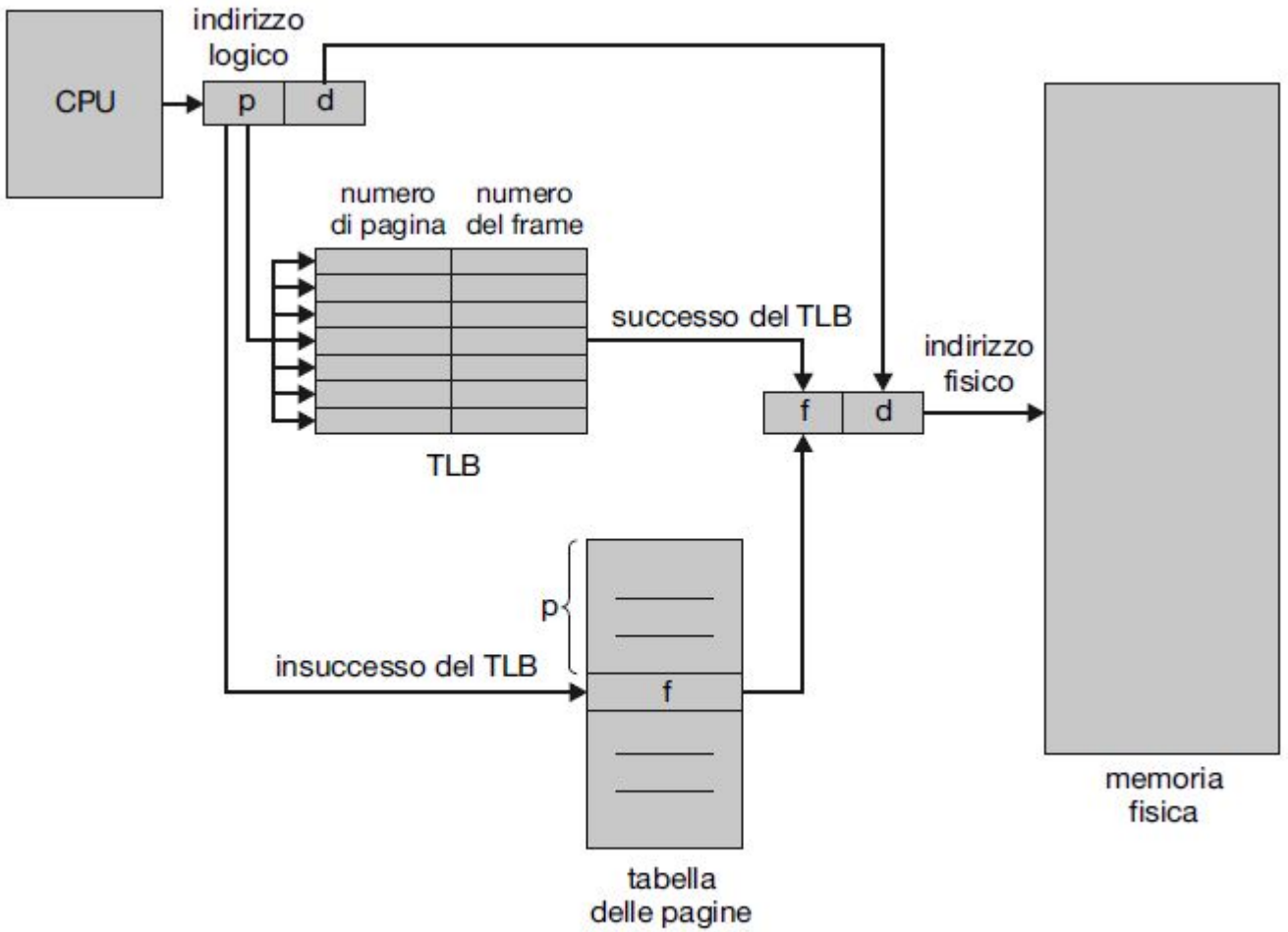


Figura 9.12 Hardware di paginazione con TLB.

ASID

Alcuni TLB memorizzano gli **identificatori dello spazio d'indirizzi** (*address-space identifier, ASID*) in ciascun elemento del TLB.

Un **ASID** identifica in modo univoco ciascun processo e si usa per fornire al processo corrispondente la protezione del suo spazio d'indirizzi.

Hit ratio

tasso di successi

percentuale di volte che il numero di pagina di interesse si trova nel TLB

tempo effettivo d’accesso alla memoria

Bit di validità

bit di validità □ ulteriore bit che si associa a ciascun elemento della tabella delle pagine

impostato a *valido*

↓

la pagina corrispondente *è* nello spazio d'indirizzi logici del processo

impostato a *non valido*

↓

la pagina *non è* nello spazio d'indirizzi logici del processo

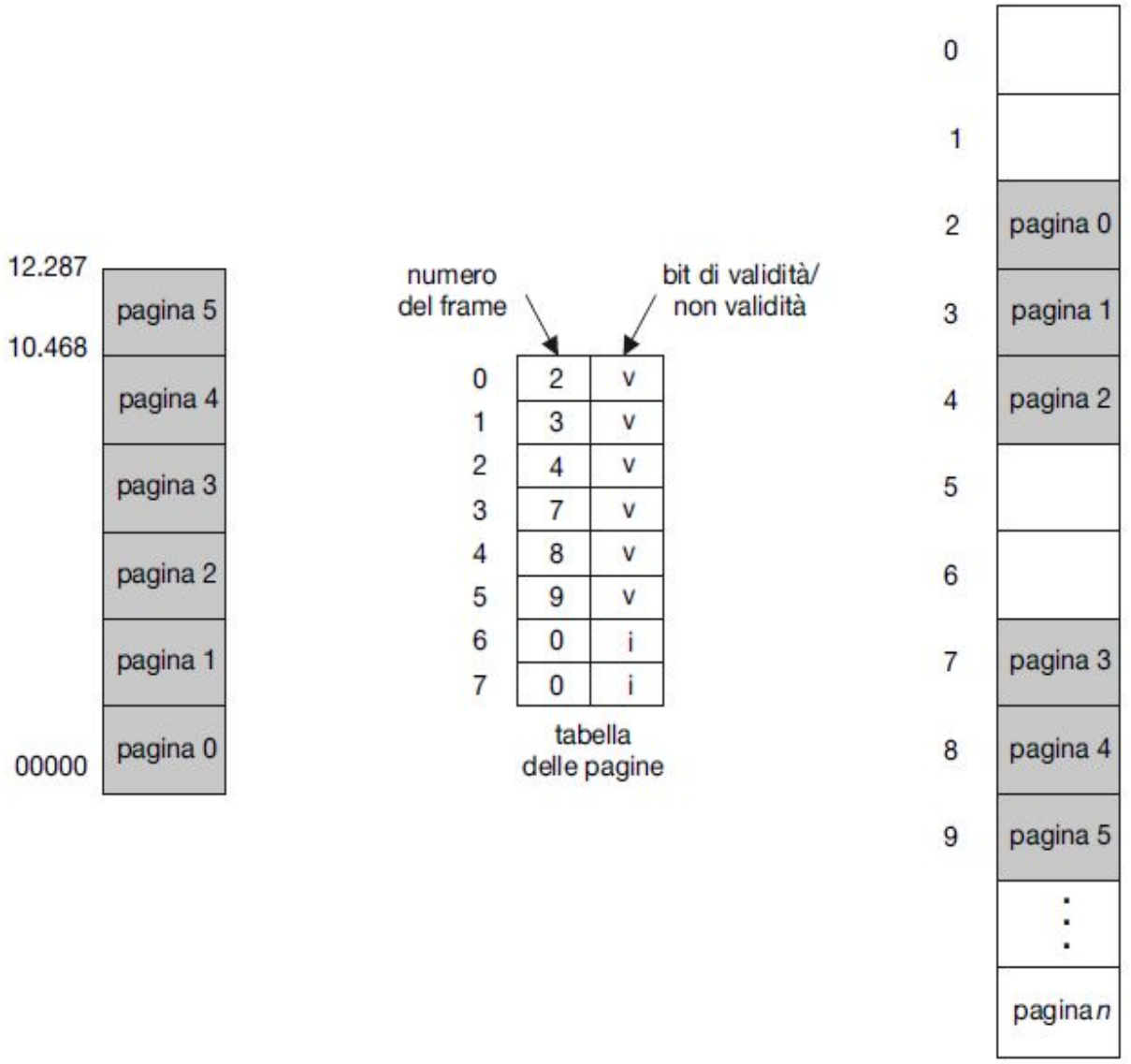


Figura 9.13 Bit di validità (v) o non validità (i) in una tabella delle pagine.

Condivisione

Un vantaggio della paginazione risiede nella possibilità di ***condividere codice comune***, il che è particolarmente importante in un ambiente con più processi.

La **condivisione della memoria tra processi** di un sistema è simile al modo in cui i thread condividono lo spazio d'indirizzi di un task.

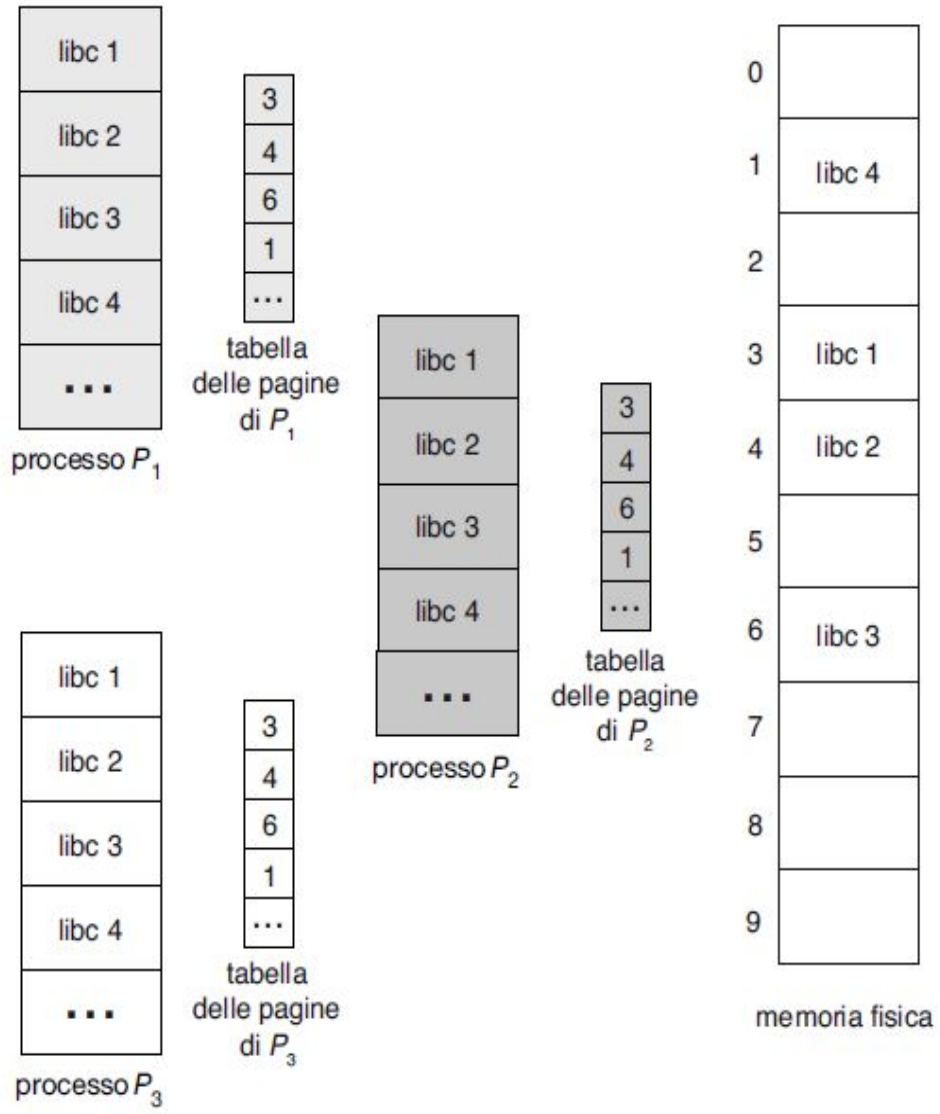


Figura 9.14 Condivisione della libreria standard del C in un ambiente paginato.

Struttura della tabella delle pagine

tecniche più comuni per *strutturare*
la tabella delle pagine

- paginazione gerarchica
- tabella delle pagine di tipo hash
- tabella delle pagine invertita

Paginazione gerarchica

Lo spazio degli indirizzi logici in un moderno calcolatore è molto grande → la stessa tabella delle pagine diventa eccessivamente grande.

suddividere la tabella
delle pagine in parti più piccole

algoritmo di **paginazione a due livelli**

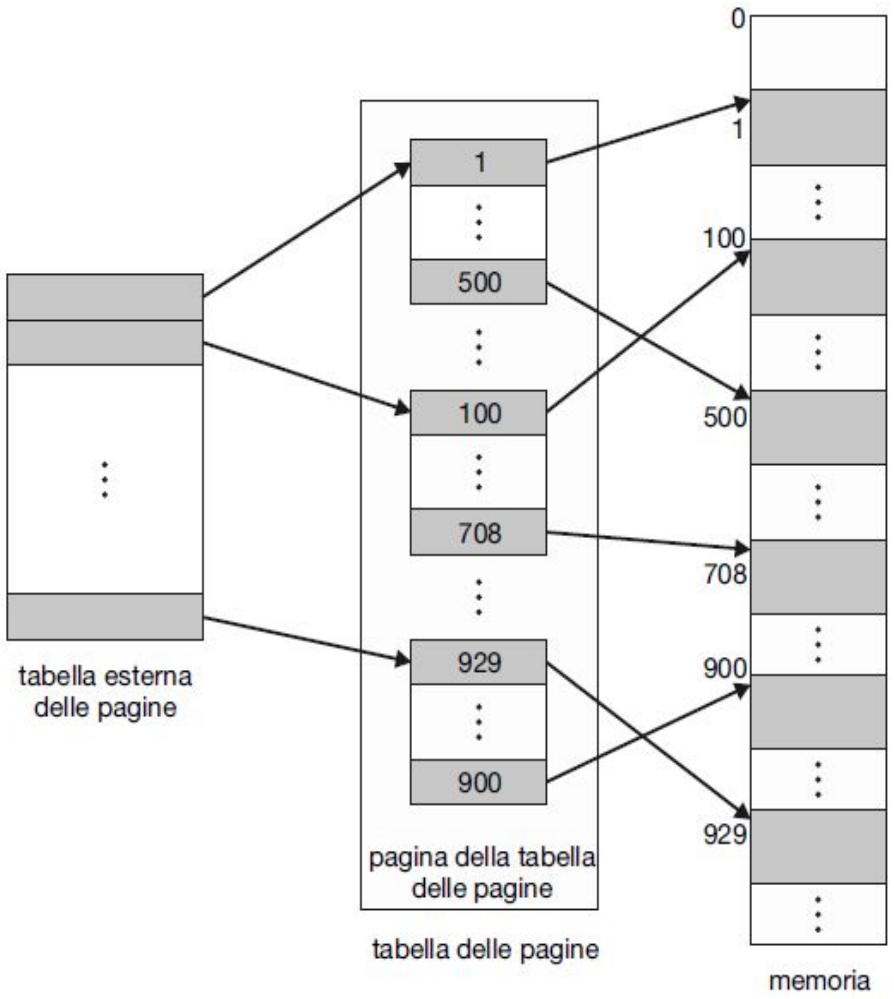


Figura 9.15 Schema di una tabella delle pagine a due livelli.

Associazione diretta

Poiché la traduzione degli indirizzi si svolge dalla tabella esterna delle pagine verso l'interno, questo metodo è anche noto come **tabella delle pagine ad associazione diretta** (*forward-mapped page table*).

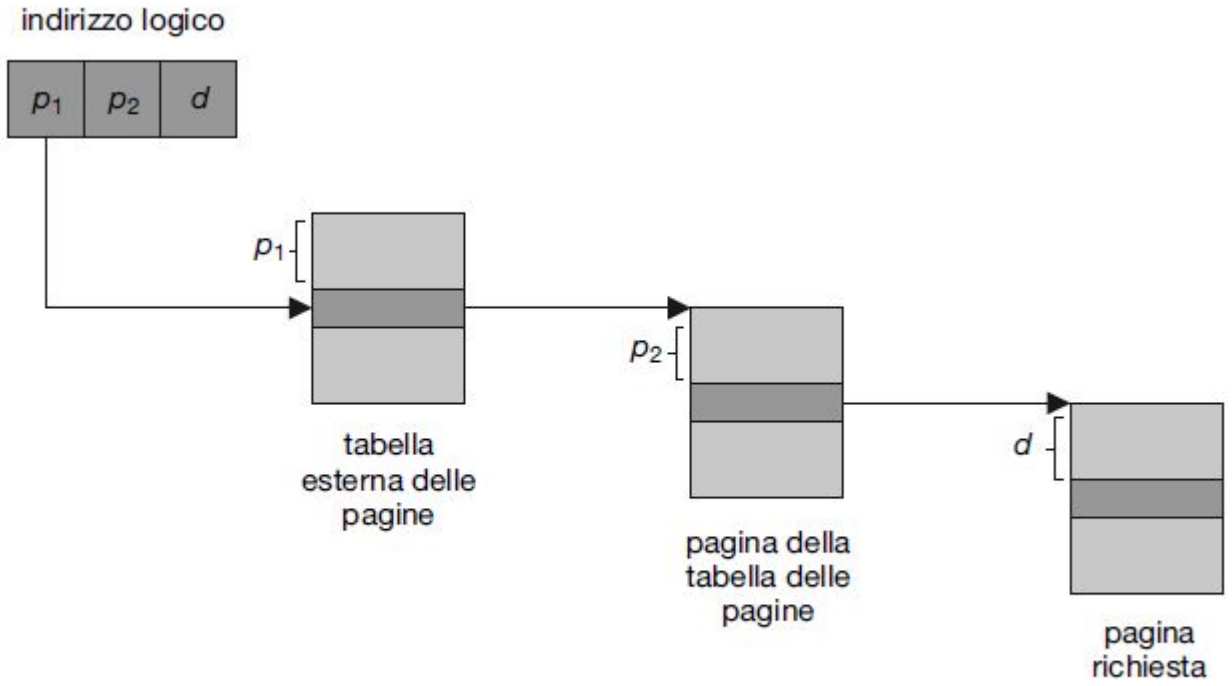


Figura 9.16 Traduzione degli indirizzi per un'architettura a 32 bit con paginazione a due livelli.

Hashing

Tabella delle pagine di tipo hash metodo di gestione molto comune degli spazi d'indirizzi oltre i 32 bit

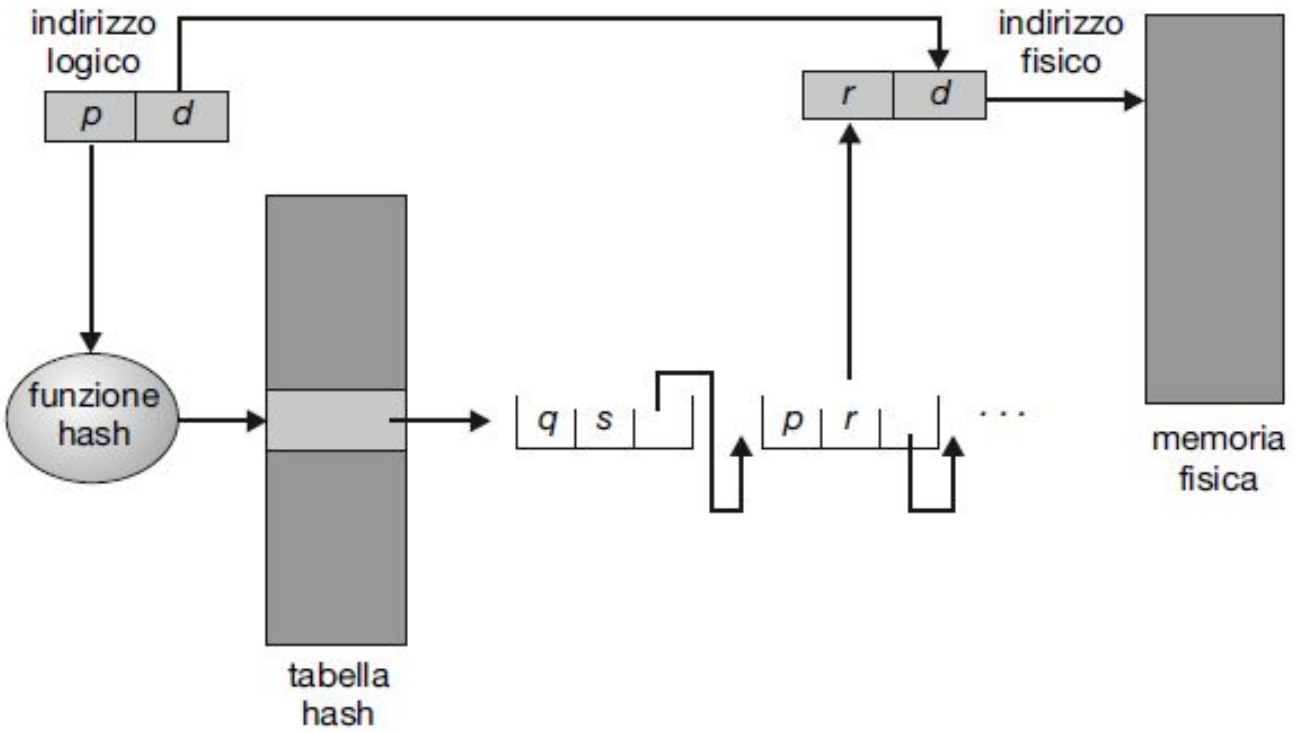


Figura 9.17 Tabella delle pagine di tipo hash.

Tabella delle pagine invertita

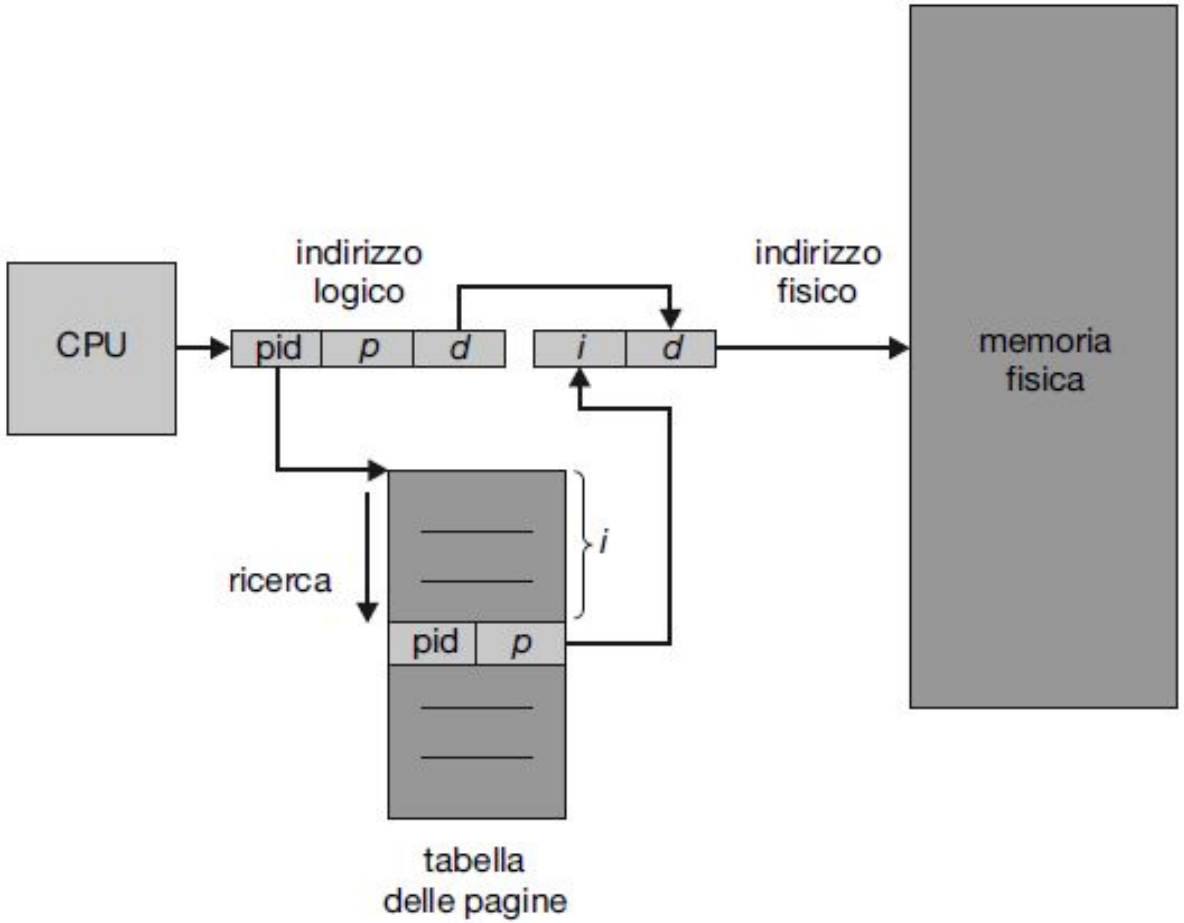


Figura 9.18 Tabella delle pagine invertita.

Swapping

Un processo, o una sua parte, può essere rimosso temporaneamente dalla memoria centrale (mediante un'operazione detta di ***swap out, scaricamento***), spostato in una **memoria ausiliaria** (*backing store)* e in seguito riportato in memoria (***swap in, caricamento***) per continuare la sua esecuzione

Swapping (avvicendamento)

Esistono tre procedure di avvicendamento o swapping

- Avvicendamento standard
- Avvicendamento con paginazione
- Avvicendamento di processi nei sistemi mobili

Swapping standard

- Lo swapping standard tra memoria centrale e memoria secondaria riguarda l'intero processo
- Vantaggio: si può sovrascrivere la memoria fisica

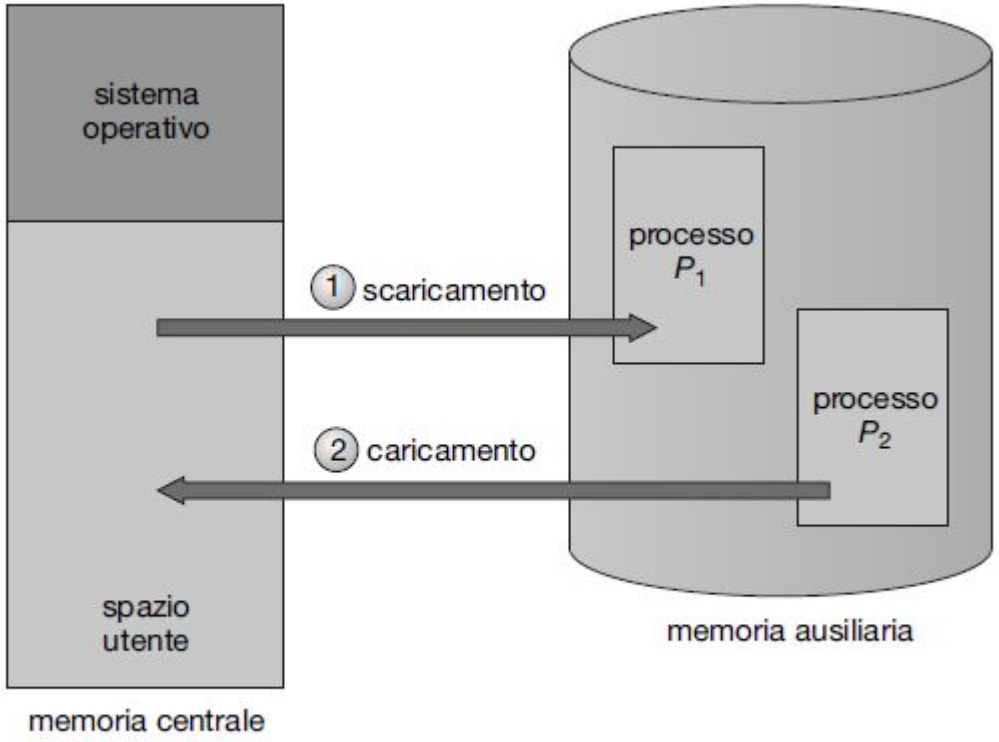


Figura 9.19 Avvicendamento standard di due processi con un disco come memoria ausiliaria.

Swapping con paginazione

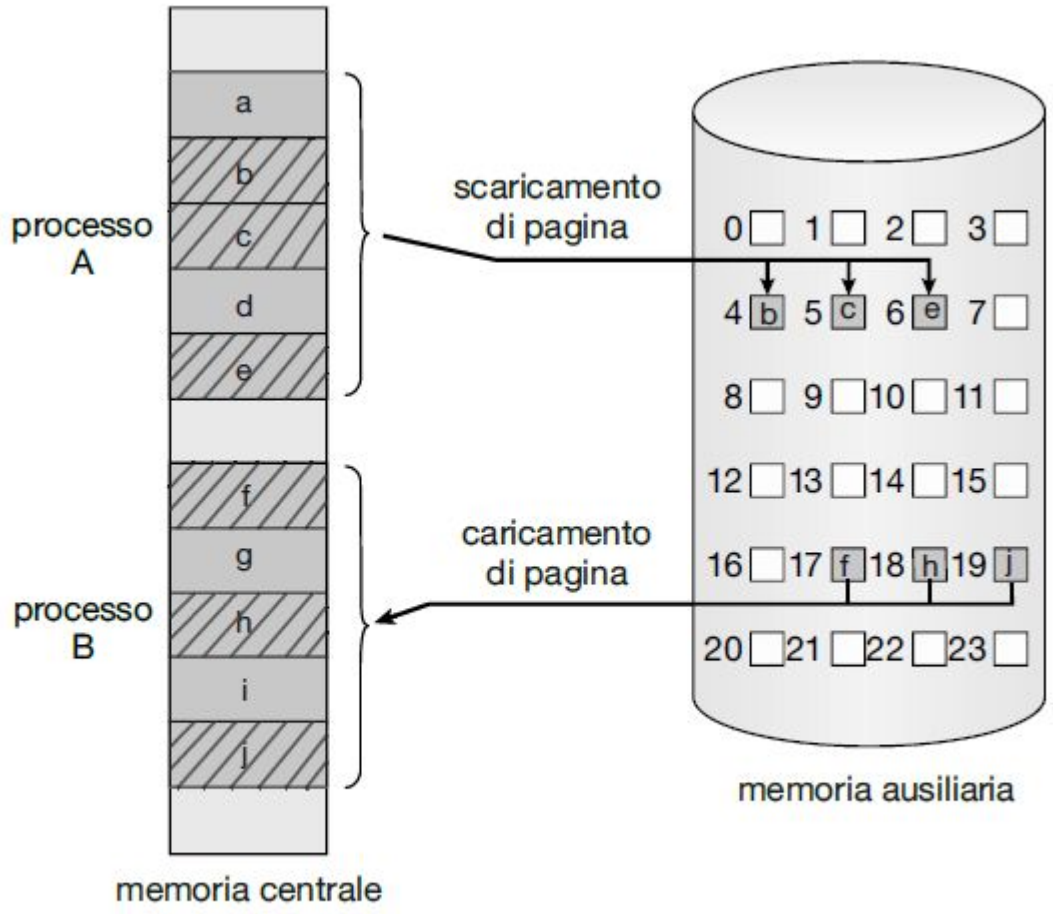


Figura 9.20 Avvicendamento con paginazione.

Swapping nei sistemi mobili

Lo swapping nei sistemi mobili non viene generalmente supportato:

- Le memorie flash a disposizione per l'archiviazione secondaria non sono molto capienti
- la memoria flash può tollerare un numero limitato di scritture prima di diventare inaffidabile
- scarsa velocità di trasferimento tra la memoria centrale e la memoria flash

Architettura Intel IA-32

La gestione della memoria nei sistemi IA-32 è suddivisa in due componenti: **segmentazione** e **paginazione**

L'unità di segmentazione e l'unità di paginazione formano insieme l'equivalente dell'unità di gestione della memoria (MMU)

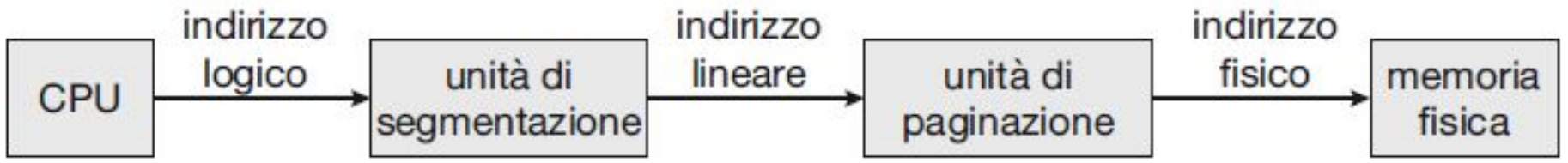


Figura 9.21 Traduzione degli indirizzi logici in indirizzi fisici in IA-32.

Segmentazione in IA-32

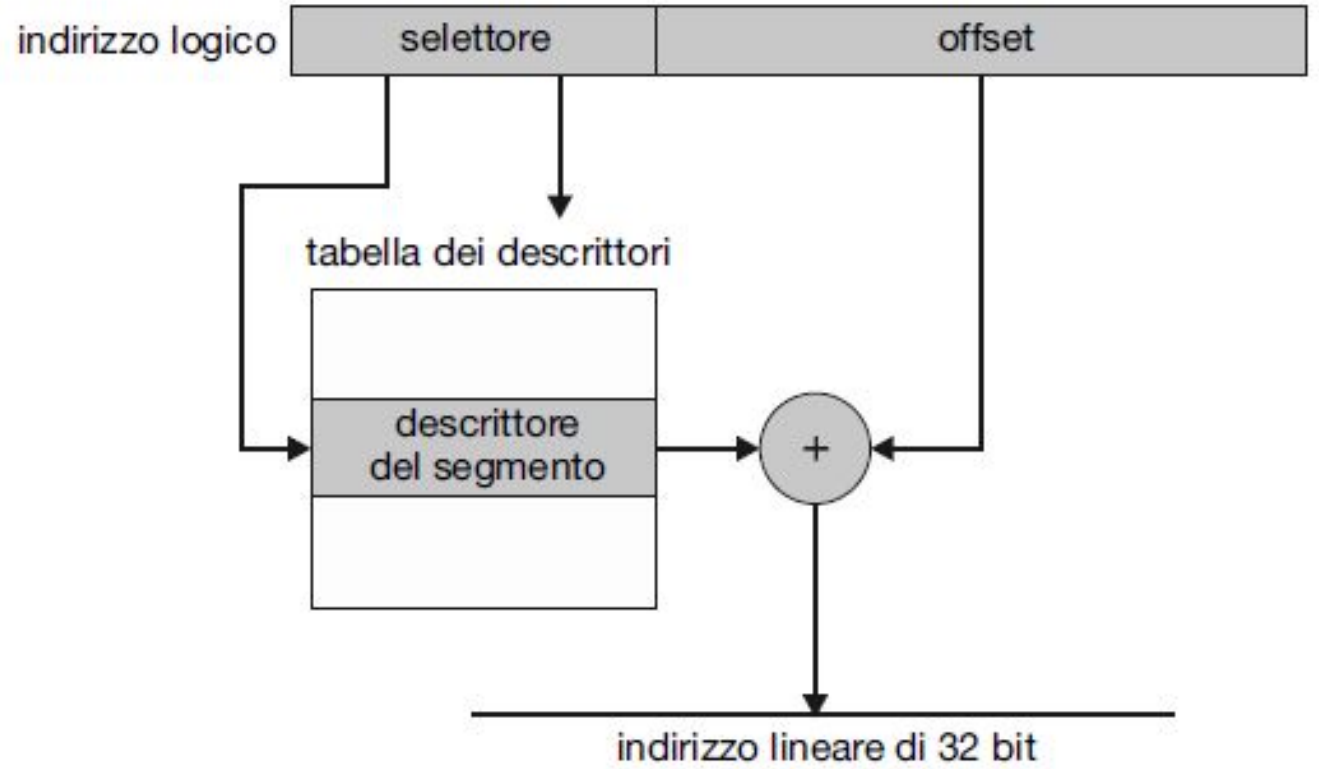


Figura 9.22 Segmentazione in IA-32.

Paginazione in IA-32

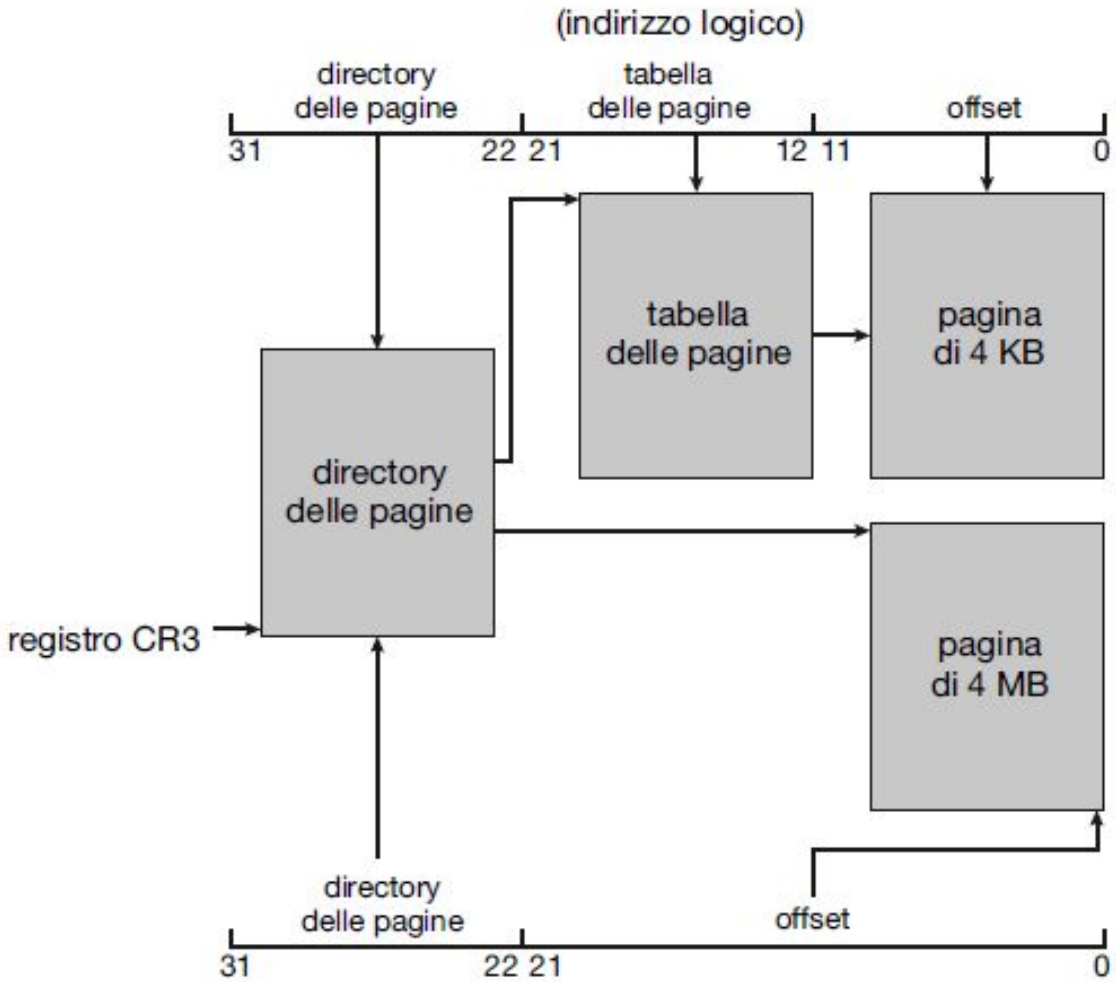


Figura 9.23 Paginazione nell'architettura IA-32.

Estensione PAE

L'**estensione di indirizzo della pagina** (**PAE**, *page address extension),* consente ai processori a 32 bit di accedere a uno spazio di indirizzamento fisico più grande di 4 GB.

Con PAE è stata inoltre aumentata la dimensione degli elementi della directory delle pagine e della tabella delle pagine, che passa da 32 a 64 bit, permettendo di estendere l'indirizzo di base delle tabelle delle pagine e dei frame da 20 a 24 bit.

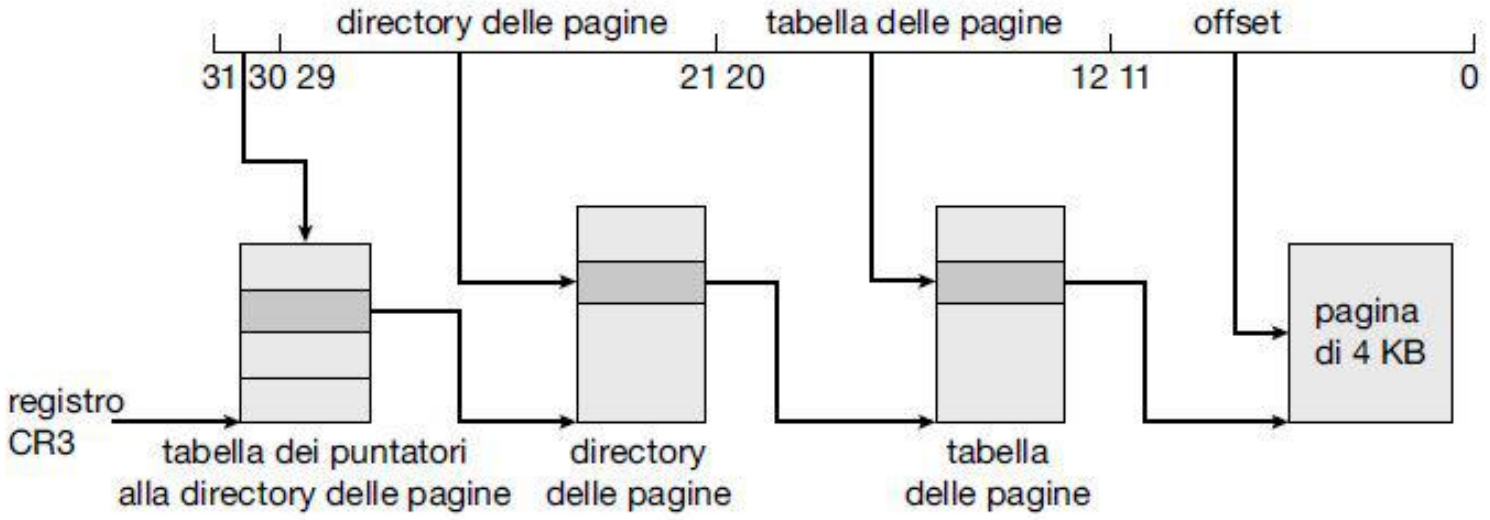


Figura 9.24 Estensione degli indirizzi di pagina.

Architettura x86-64

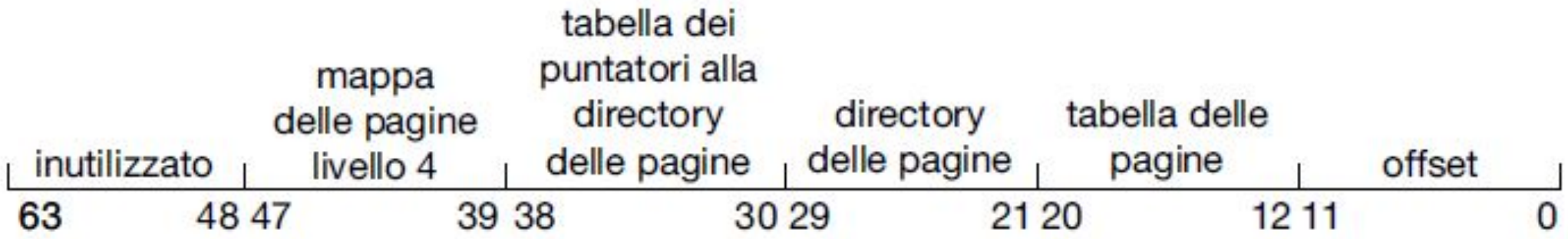


Figura 9.25 Indirizzo lineare in x86-64.

Architettura ARMv8

ARMv8 supporta tre diverse **granularità di traduzione** (*translation granule):*

4 KB, 16 KB e 64 KB.

La Figura 9.26 illustra la struttura degli indirizzi ARMv8 con una granularità di 4 KB con un massimo di quattro livelli di paginazione.

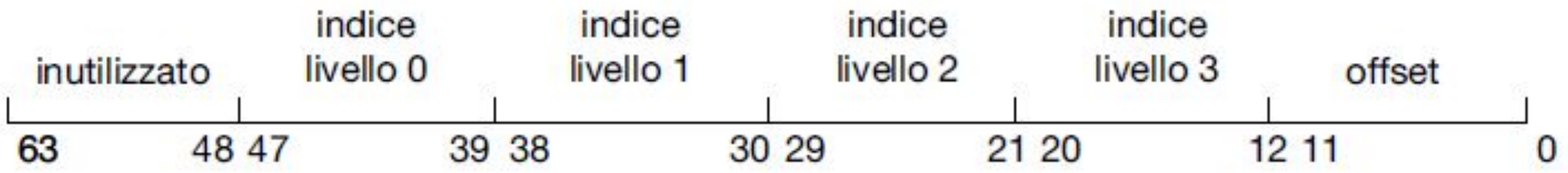


Figura 9.26 Indirizzo in ARM, con granularità di 4 KB.

Paginazione ARMv8

registro TTBR ⟶ è il registro di base della tabella di traduzione e punta alla tabella del livello 0 per il thread corrente

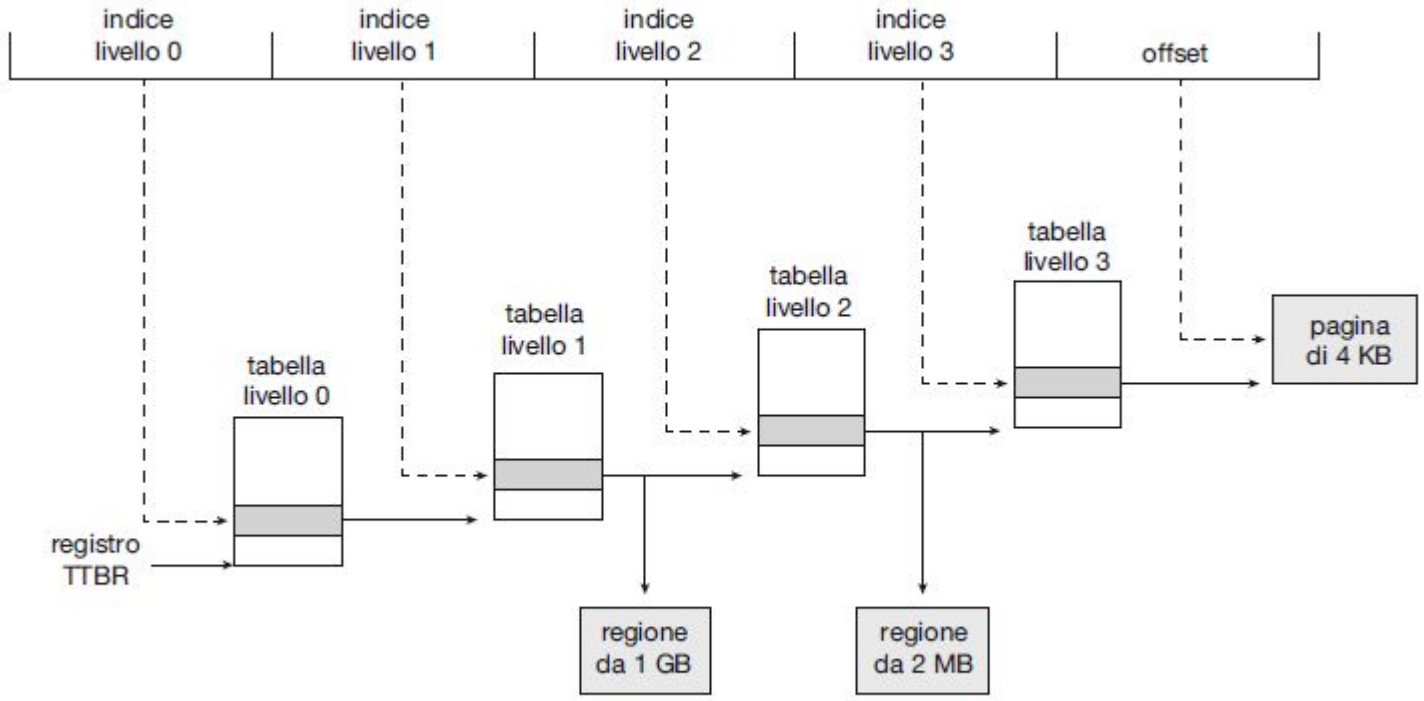


Figura 9.27 Paginazione gerarchica su 4 livelli in ARM.

Memoria virtuale

- Le istruzioni di un processo, per essere eseguite, devono risiedere in memoria fisica
- In sistemi multiprogrammati più processi risiedono contemporaneamente in memoria
- La memoria fisica potrebbe non essere sufficientemente capiente
- La **memoria virtuale** è una tecnica che permette di eseguire processi che possono anche non essere completamente contenuti in memoria fisica

Memoria virtuale

La **memoria virtuale** facilita la programmazione, poiché il programmatore non deve preoccuparsi della quantità di memoria fisica disponibile, ma può concentrarsi sul problema da risolvere con il programma

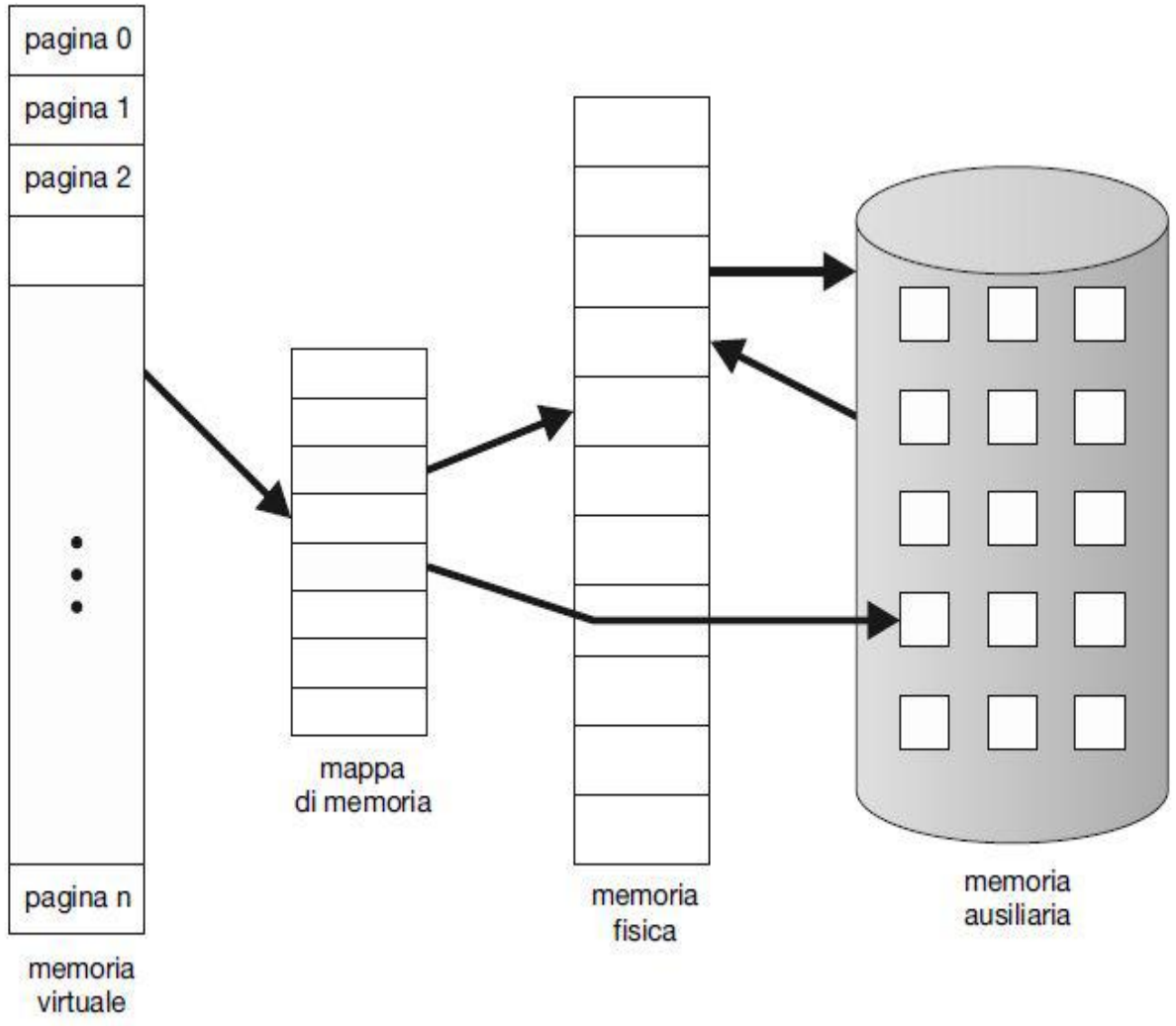


Figura 10.1 Schema che mostra una memoria virtuale più grande di quella fisica.

Vantaggi della memoria virtuale

1. I programmi possono essere più grandi della memoria fisica
2. La memoria centrale viene astratta in un vettore di memorizzazione molto grande e uniforme
3. I processi possono condividere facilmente file
4. Possono essere realizzate memorie condivise

Svantaggi della memoria virtuale

1. Realizzazione complessa

2. Se usata scorrettamente, riduce di molto le prestazioni del sistema

Spazio degli indirizzi virtuali

Spazio degli indirizzi virtuali

collocazione dei processi in memoria dal punto di vista logico (o virtuale)

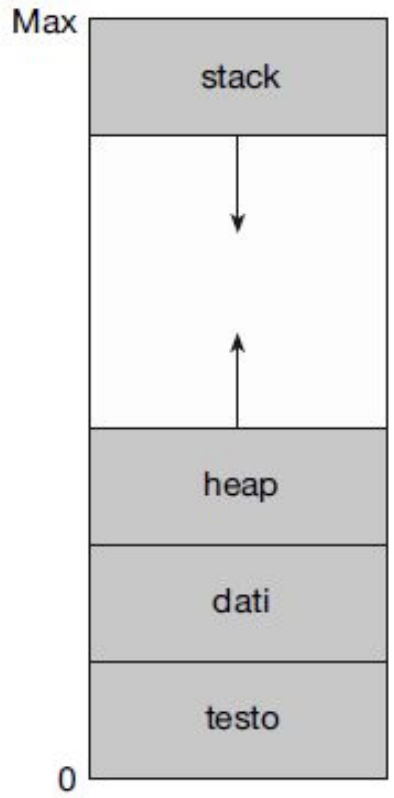


Figura 10.2 Spazio degli indirizzi virtuali di un processo in memoria.

Uno spazio degli indirizzi virtuali che contiene buchi si definisce sparso

Condivisione delle pagine

Oltre a separare la memoria logica da quella fisica, la **memoria virtuale** offre il vantaggio di condividere i file e la memoria fra due o più processi, mediante la condivisione delle pagine.

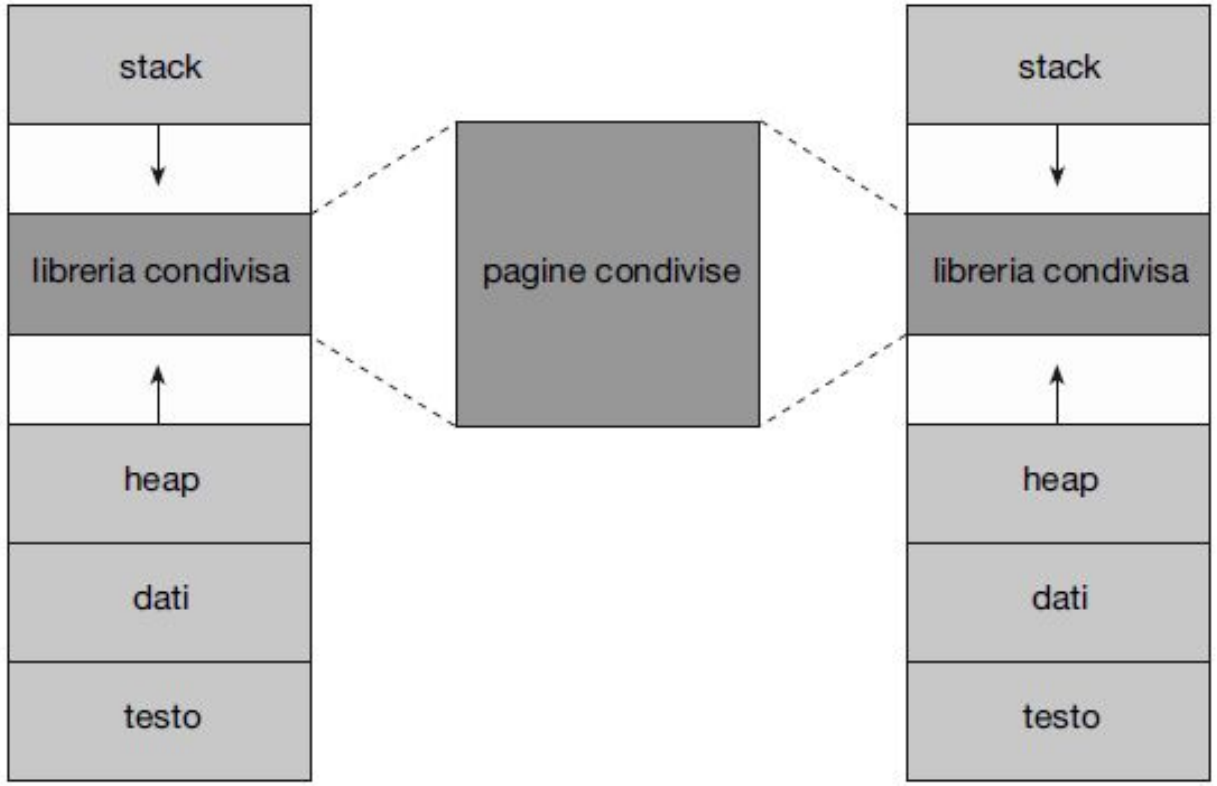


Figura 10.3 Condivisione delle librerie tramite la memoria virtuale.

Paginazione su richiesta

Paginazione su richiesta → strategia di allocazione di memoria che consiste nel caricare le pagine nel momento in cui servono realmente

le pagine sono caricate in memoria solo quando richieste durante l'esecuzione del programma

La **paginazione su richiesta** mostra uno dei principali vantaggi della memoria virtuale: caricando solo le parti necessarie dei programmi la memoria viene utilizzata in modo *più efficiente*.

Paginazione su richiesta

1. Mentre un processo è in esecuzione, alcune pagin saranno in memoria, altr saranno in memoria secondaria (es. disco)
2. È necessario un supporto hardware per tenere traccia di questa division
3. Possiamo utilizzare lo schema con bit di validit a tale scopo

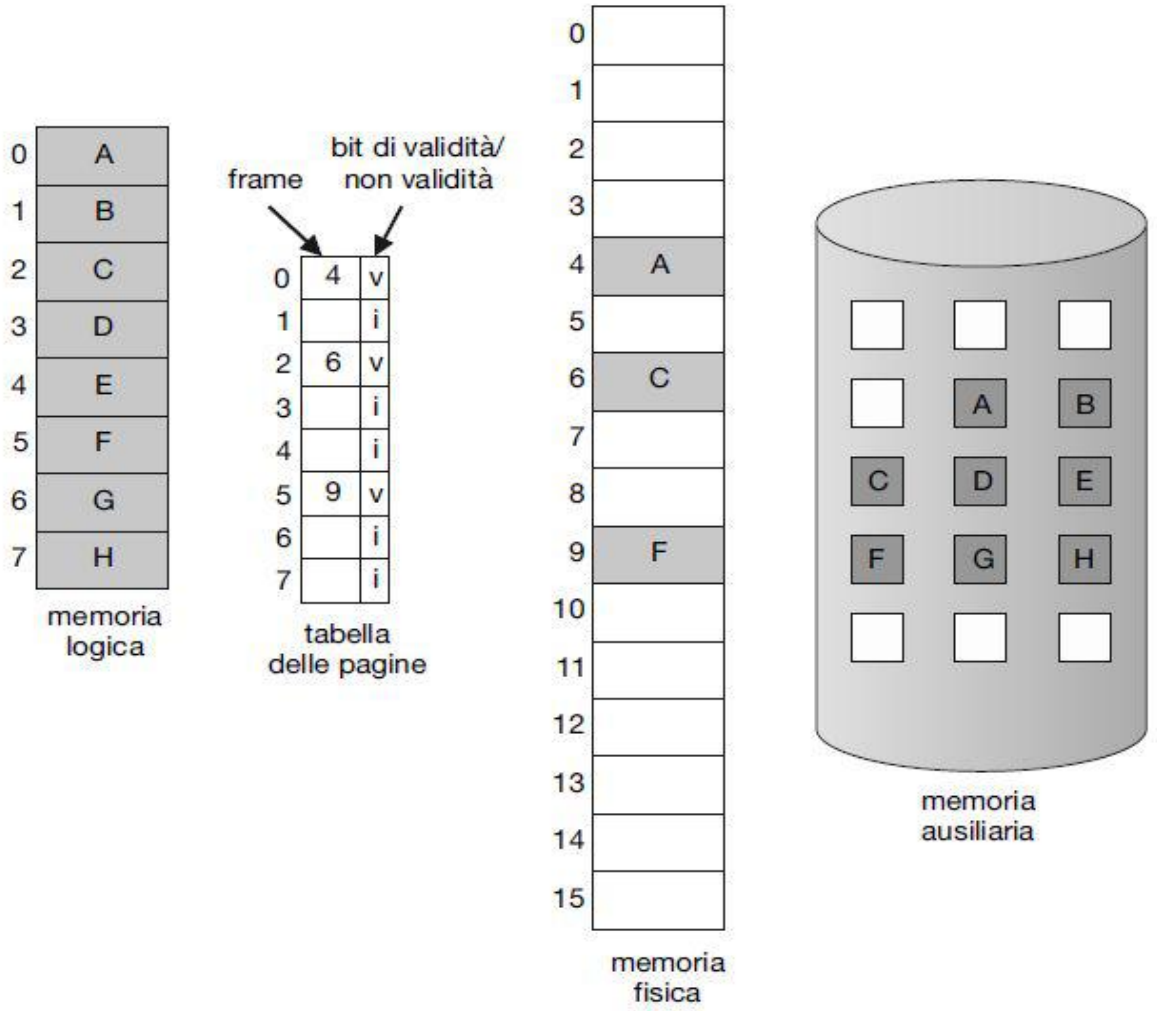


Figura 10.4 Tabella delle pagine quando alcune pagine non si trovano nella memoria centrale.

Page fault

L'accesso a una pagina contrassegnata come non valida causa un evento o eccezione di **page fault** (*pagina mancante*)

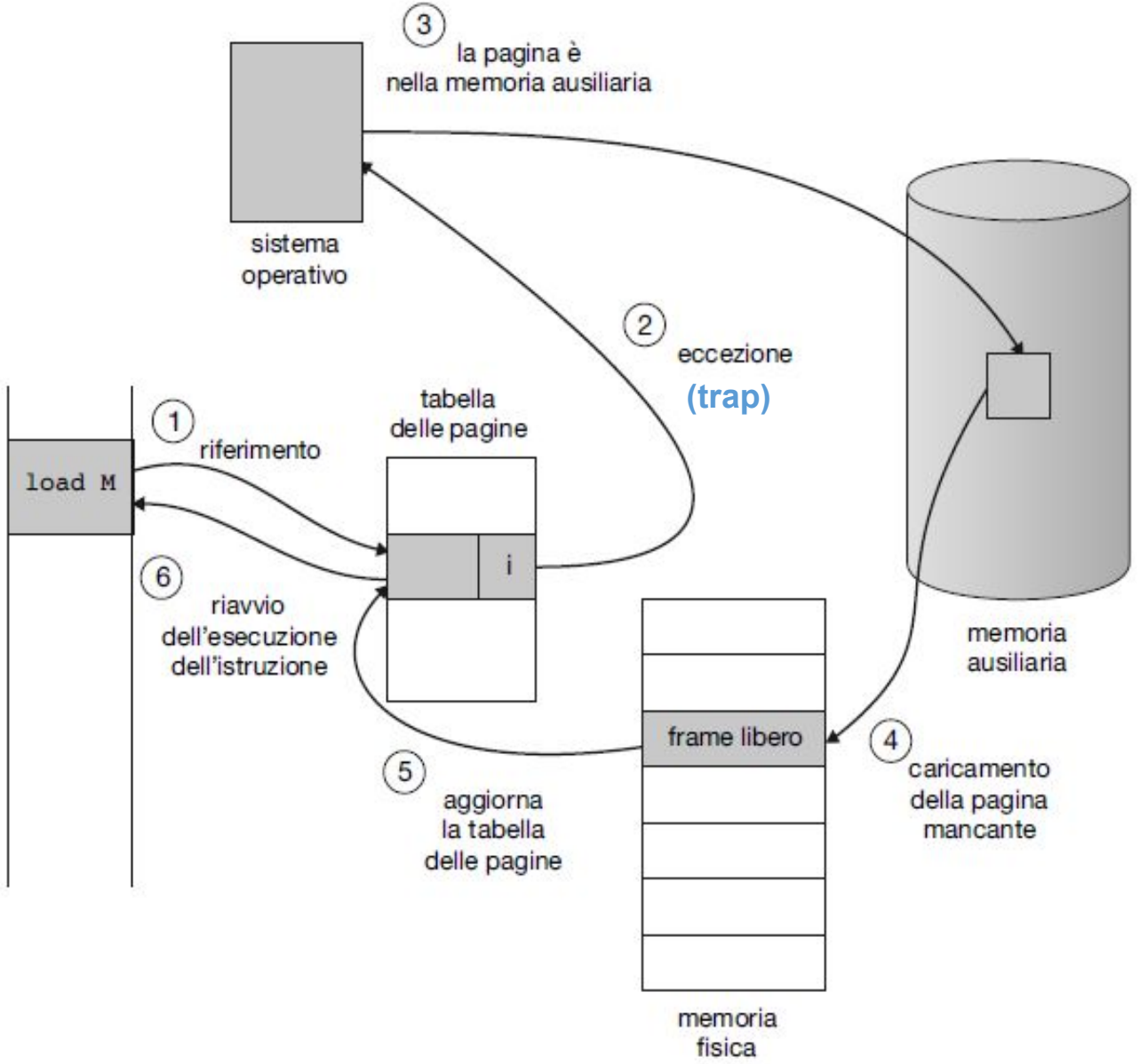


Figura 10.5 Fasi di gestione di un page fault.

Hardware per la paginazione su richiesta

1. Tabella delle pagine con bit di validità

2. Memoria secondaria costituita da un disco ad alta velocità (dispositivo di swap). La porzione di memoria dedicata per lo swapping si chiama swap space

Lista dei frame liberi

Lista dei frame liberi → insieme di frame disponibili e utilizzabili per soddisfare le richieste

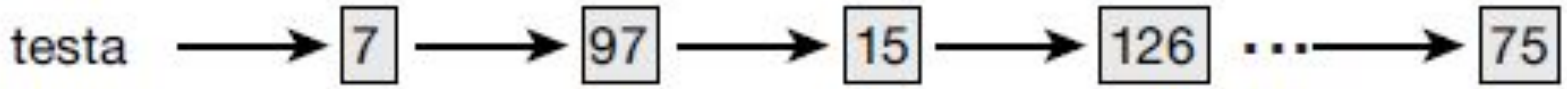


Figura 10.6 Lista dei frame liberi.

- All'avvio di un sistema tutta la memoria disponibile viene inserita nella lista dei frame liberi.

- A un certo punto la lista diventa vuota → *deve essere ripopolata*

Allocazione dei frame liberi

I sistemi operativi allocano generalmente i frame liberi usando una tecnica nota come zero-fill-on-demand:

- i frame liberi vengono azzerati su richiesta prima di essere allocati, cancellando così il loro precedente contenuto

Prestazioni della paginazione su richiesta

La **paginazione su richiesta** può avere un effetto rilevante sulle prestazioni di un calcolatore.

Il motivo si può comprendere calcolando il **tempo di accesso effettivo** per una memoria con paginazione su richiesta.

Tempo di accesso effettivo EAT

$$T_{EAT} = p \times t_{pf} + (1 - p) \times t_{ma}$$

dove

p è la probabilità di page fault, con $0 \leq p \leq 1$

t_{ma} è il tempo di accesso in memoria (tipicamente nanosecondi)

t_{pf} è il tempo di gestione del page fault, che comprende:

- Gestione dell'eccezione (trap) di page fault (tipicamente microsecondi)
- Lettura della pagina mancante da disco (tipicamente millisecondi)
- Riavvio del processo (tipicamente microsecondi)

Esempio calcolo T_{EAT}

Si consideri una situazione in cui ci siano stati 15 riferimenti in memoria con 8 page fault

Denotando con t_{ma} il tempo di accesso in memoria e con t_{pf} il tempo necessario alla gestione del page fault, si ottiene:

tempo effettivo di accesso (Effective Access Time)

$T_{EAT} = 7 \times t_{ma} + 8 \times t_{pf}$

La probabilità di page fault sarà $p_{pf} = 8 / 15 = 0.53$

pari al 53%

Copiatura su scrittura

In caso di fork, le pagine del processo genitore sono inizialmente condivise con il processo figlio

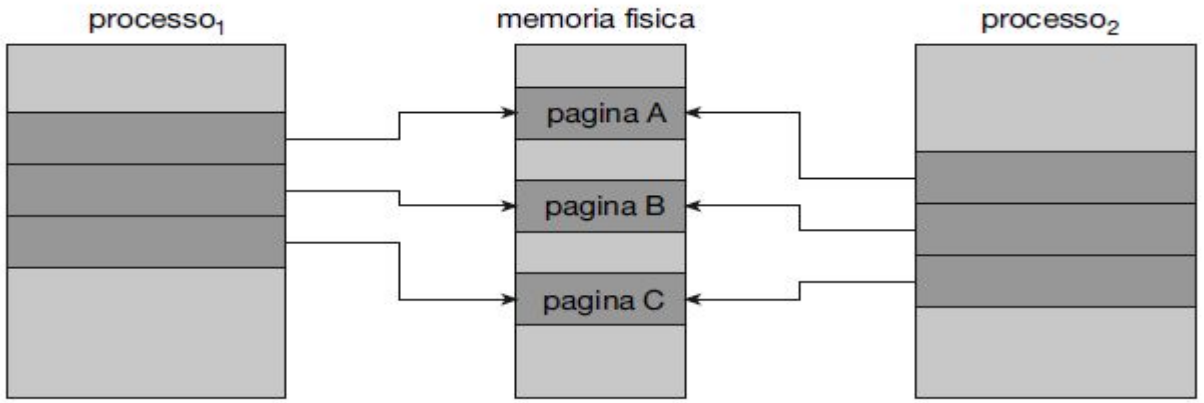


Figura 10.7 Prima della modifica alla pagina C da parte del processo 1.

Le pagine condivise si contrassegnano come pagine da copiare su scrittura

- se un processo (genitore o figlio) scrive su una pagina condivisa, il sistema deve creare una copia di tale pagina

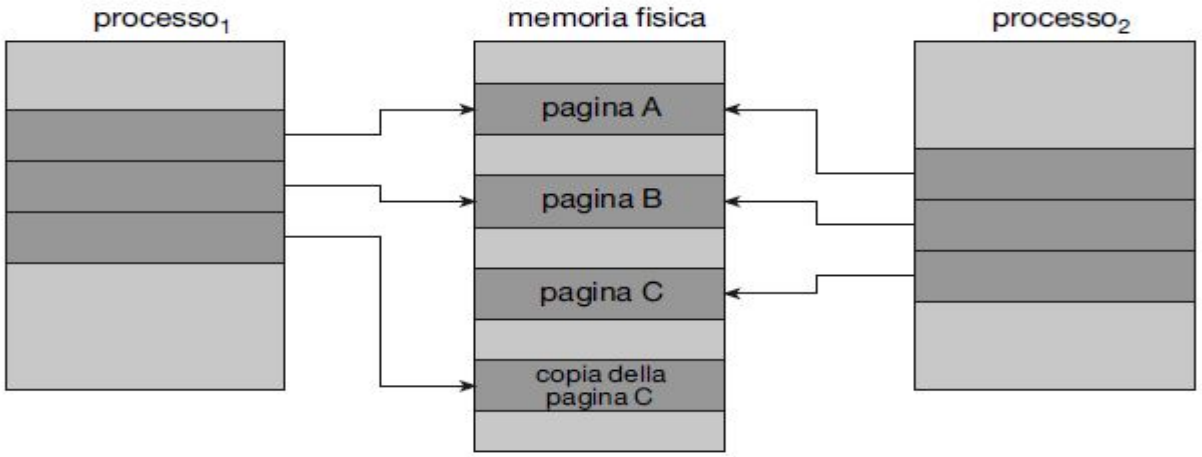


Figura 10.8 Dopo la modifica alla pagina C da parte del processo 1.

Sovrallocazione

Se un processo di 10 pagine ne impiega effettivamente solo la metà, la paginazione su richiesta fa risparmiare il tempo di I/O necessario a caricare le 5 pagine mai usate

Se la memoria dispone di 40 frame, possiamo allocare fino a 8 processi in memoria invece dei 4 allocabili se ciascun processo richiedesse 10 pagine impiegandone effettivamente solo la metà

Cosa succede se un processo ha improvvisamente necessità di caricare tutte e dieci le sue pagine?

Necessità di sostituzione di pagine

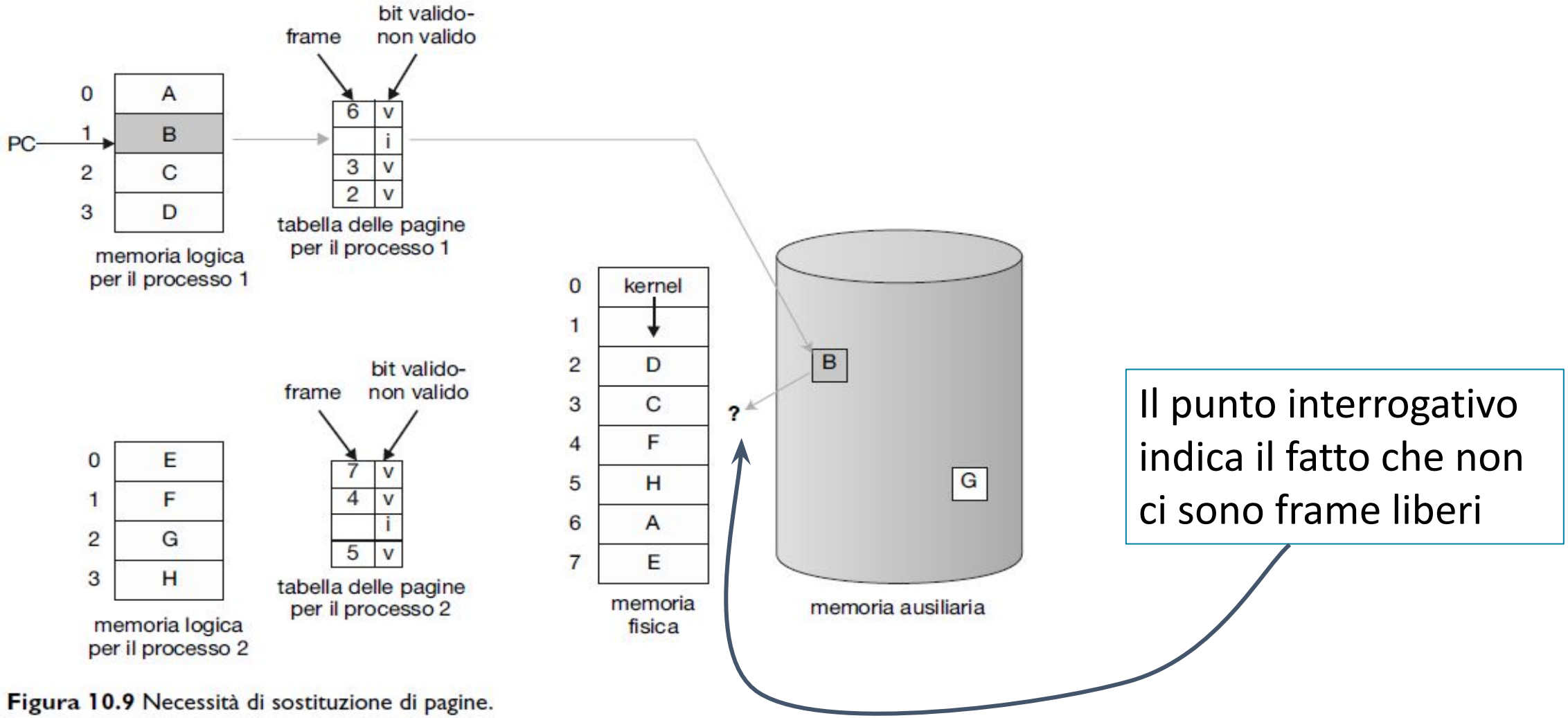


Figura 10.9 Necessità di sostituzione di pagine.

Il punto interrogativo indica il fatto che non ci sono frame liberi

Sostituzione di pagina

Se nessun frame è libero ne viene liberato uno attualmente inutilizzato (vittima)

La vittima viene copiata nello spazio di swap e la tabella delle pagine viene modificata

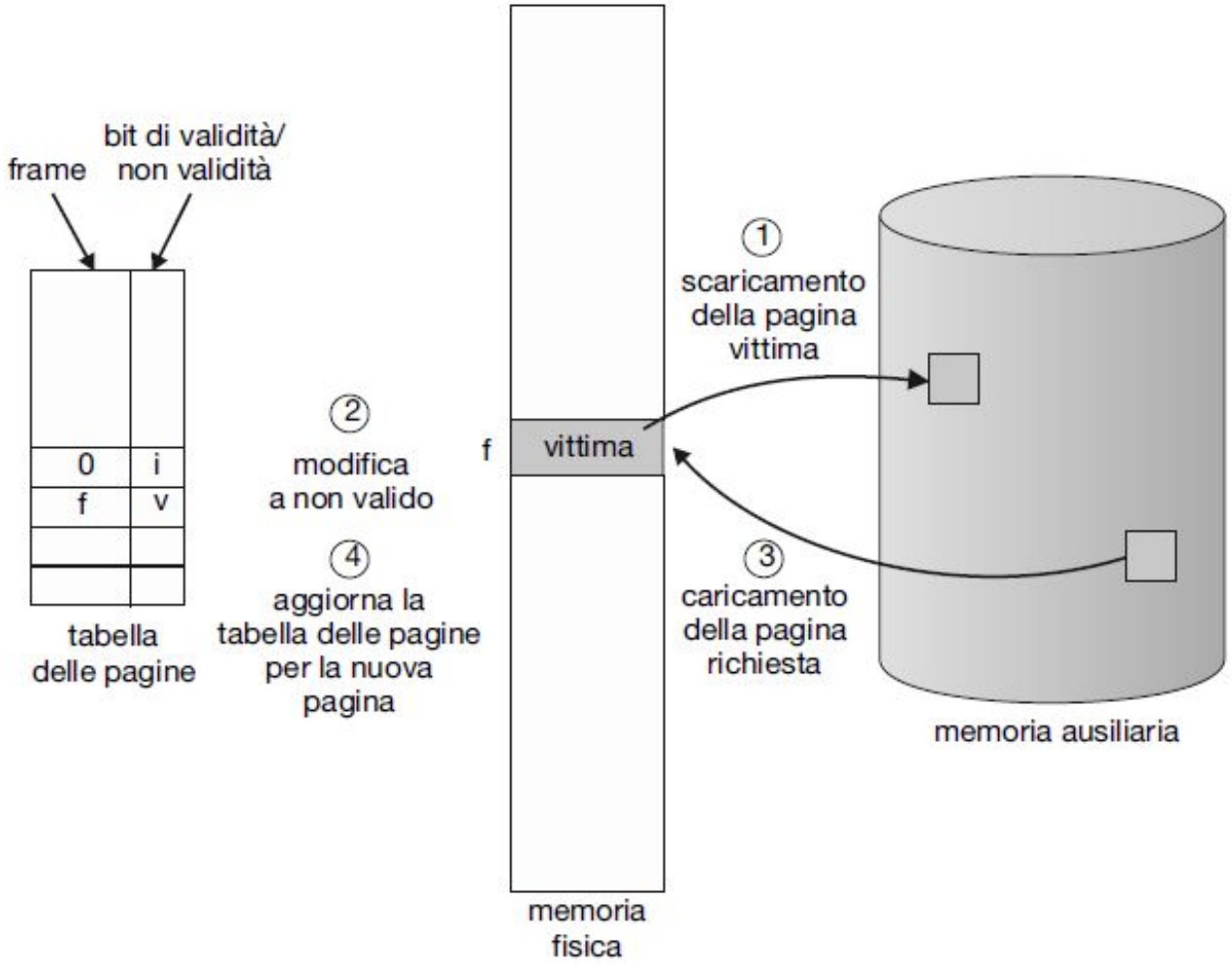


Figura 10.10 Sostituzione di una pagina.

Dirty bit

Se non esiste alcun frame libero, sono necessari due trasferimenti di pagina

1. da memoria principale a memoria secondaria
2. da memoria secondaria a memoria principale

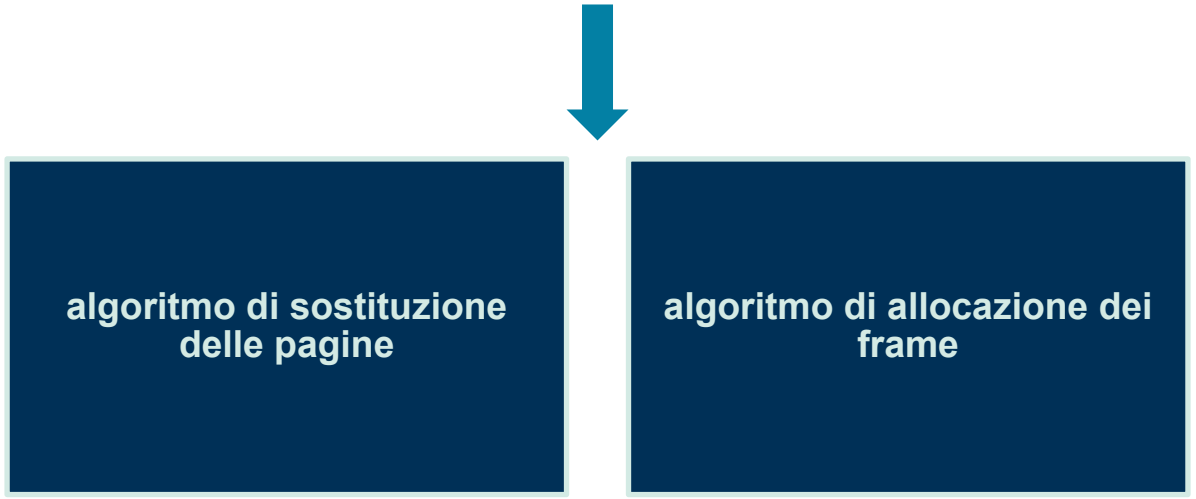
Per limitare il tempo del page fault, l'hardware del calcolatore dispone di un bit di modifica (dirty bit) , che viene posto a 1 quando si modifica una pagina

Quando si sceglie una pagina da sostituire, si legge il suo dirty bit:

- Se vale 1, la pagina deve essere scritta su disco
- Se vale 0, non è necessario scrivere su disco, la pagina già c'è.

Paginazione su richiesta

Per realizzare la paginazione su richiesta è necessario sviluppare



Quali frame sostituire in caso di page fault

Quanti frame allocare ad ogni singolo processo

Valutazione degli algoritmi di sostituzione

Per valutare un **algoritmo di sostituzione delle pagine** si conta il numero di page fault data successione di riferimenti in memoria, la quale contiene i numeri di pagina da accedere, e il numero di frame disponibili

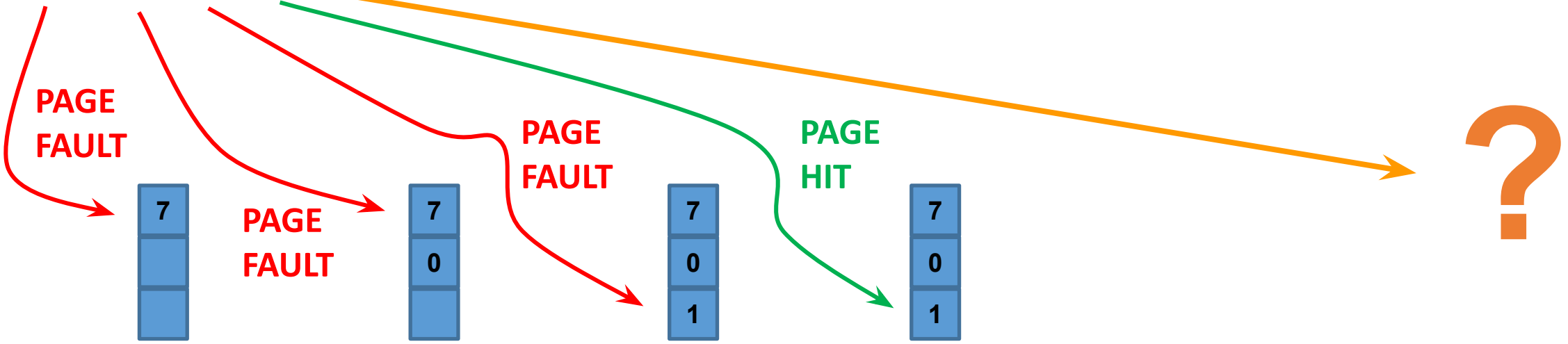
Sostituzione delle pagine

Esempio di successione di riferimenti in memoria:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Avendo 3 frame di memoria disponibili:

7, 0, 1, 0, 2, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Valutazione degli algoritmi di sostituzione

Aumentando il numero dei frame, il numero di page fault diminuisce fino al livello minimo

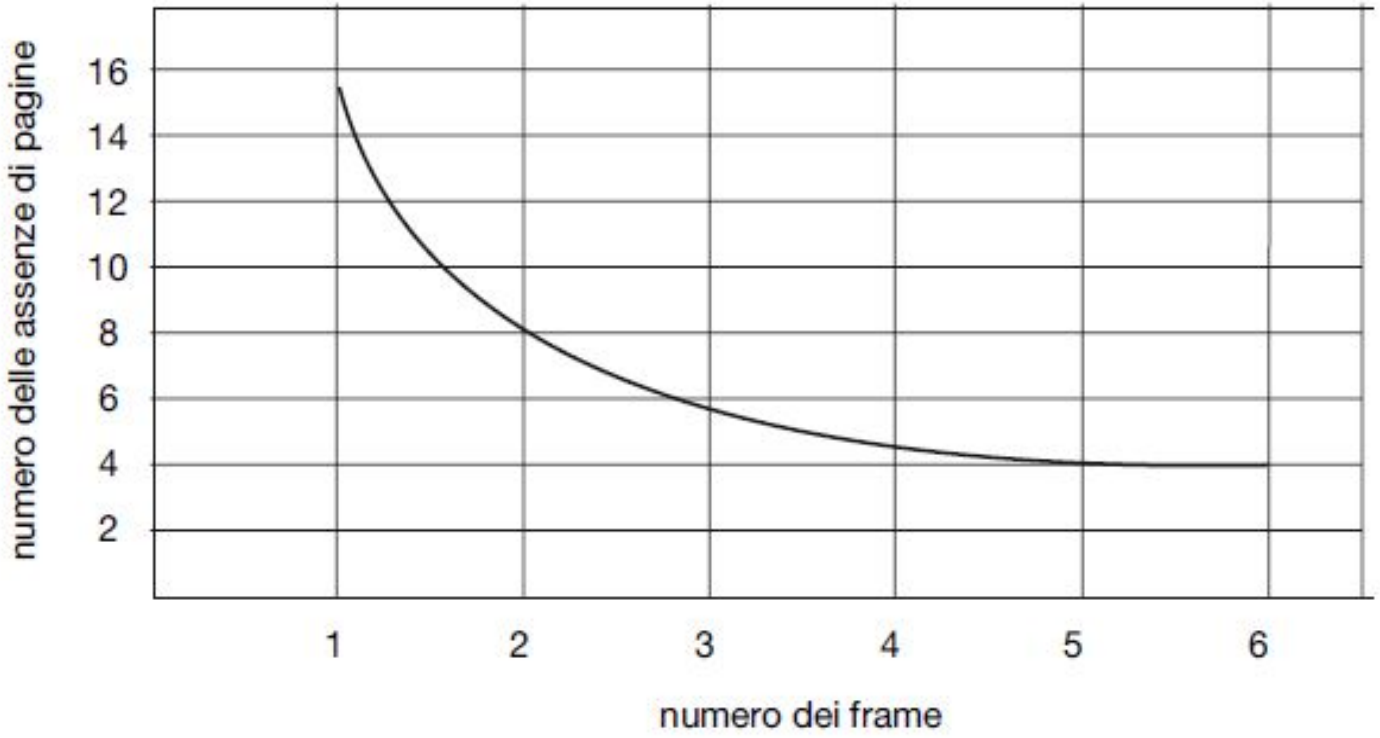


Figura 10.11 Grafico che illustra il numero di page fault rispetto al numero dei frame.

Algoritmi di sostituzione

Esistono molti **algoritmi di sostituzione delle pagine**: in genere si sceglie quello con il minimo tasso di page fault. I principali sono:

- Sostituzione delle pagine secondo l'ordine d'arrivo (FIFO)
- Sostituzione ottimale delle pagine (OPT)
- Sostituzione delle pagine usate meno recentemente (LRU)

Algoritmo di sostituzione delle pagine FIFO

algoritmo FIFO $\longrightarrow$ algoritmo di sostituzione delle pagine più semplice. Associa a ogni pagina l'istante di tempo in cui quella pagina è stata portata in memoria.

Se si deve sostituire una pagina, si seleziona quella presente in memoria da più tempo.

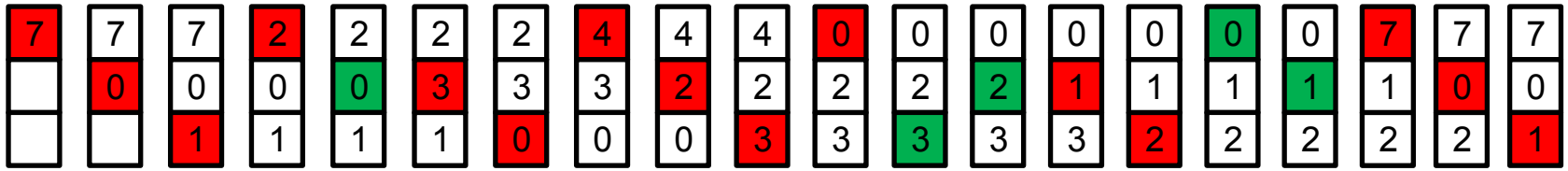
Algoritmo di sostituzione delle pagine FIFO

Esempio

Sia data la seguente successione di riferimenti in memoria con 3 frame disponibili e un algoritmo di sostituzione FIFO.

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Quanti page fault si verificheranno?

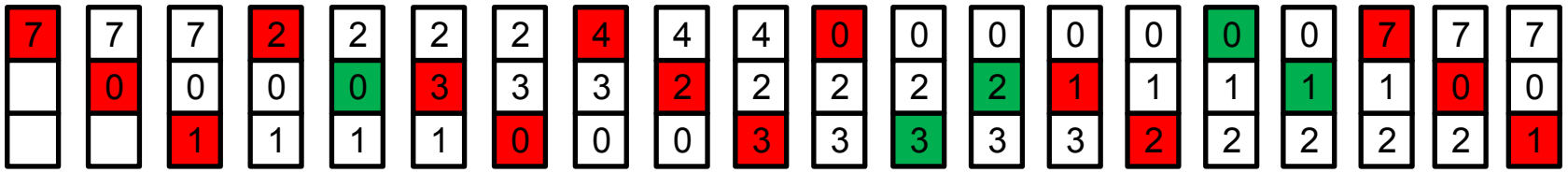


page fault

page hit

Tasso di page fault

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Il tasso di page fault per la successione di riferimenti dato e una memoria di tre frame è pari a

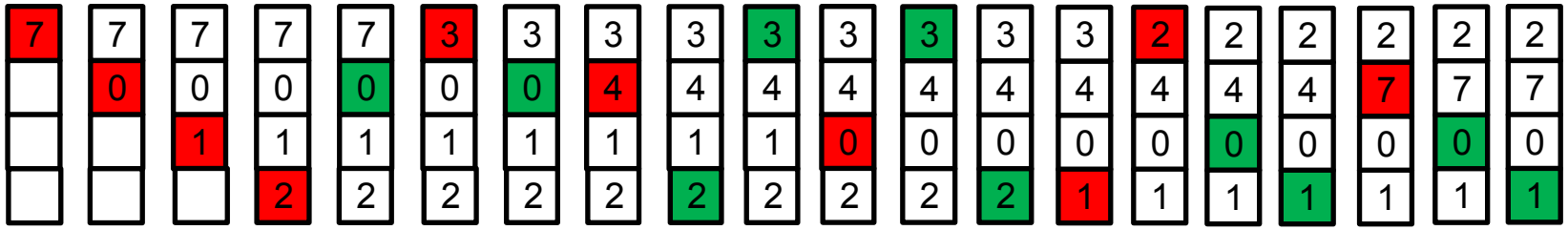
Tasso di page fault = 15 / 20 = 75%

Cosa succede aumentando la memoria?

Algoritmo di sostituzione delle pagine FIFO

Successione di riferimenti:
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Pagine in memoria:
4



Tasso di page fault = 10 / 20 = 50%

Anomalia di Belady

Gli algoritmi FIFO soffrono dell'**anomalia di Belady**: il tasso di page fault può *aumentare* con il numero dei frame assegnati ai processi.

Algoritmo di sostituzione delle pagine OPT

In seguito alla scoperta dell'anomalia di Belady si è ricercato un **algoritmo ottimale di sostituzione delle pagine**

Algoritmo di sostituzione OPT: *si sostituisce la pagina che non verrà usata per il periodo di tempo più lungo.*

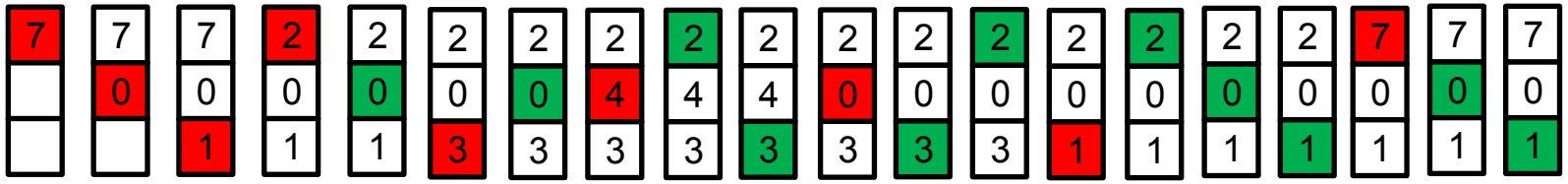
Algoritmo di sostituzione delle pagine OPT

Esempio

Sia data la seguente successione di riferimenti in memoria con 3 frame disponibili e un algoritmo di sostituzione OPT

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Quanti page fault si verificheranno?



page fault

page hit

Tasso di page fault = 9 / 20 = 45%

Algoritmo di sostituzione delle pagine OPT

L'algoritmo ottimale di sostituzione OPT delle pagine è difficile da realizzare, perché richiede la conoscenza futura della successione della successione dei riferimenti

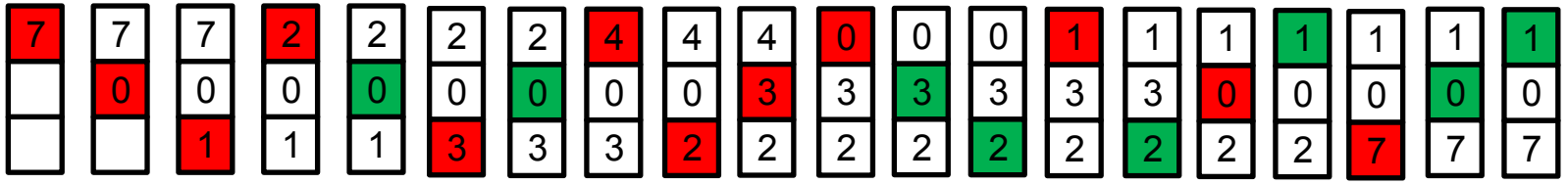
OPT assicura il tasso minimo di page fault per un dato numero di frame

Quindi, OPT si impiega soprattutto per studi comparativi

Algoritmo di sostituzione delle pagine LRU

La **sostituzione LRU** associa a ogni pagina l'istante in cui è stata usata per l'ultima volta. Quando occorre sostituire una pagina, l'algoritmo LRU sceglie quella che non è stata usata per il periodo più lungo.

Successione di riferimenti:
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
Pagine in memoria:
3



page fault
page hit

Tasso di page fault = 12 / 20 = 60%

Algoritmo di sostituzione delle pagine LRU

- Il criterio **LRU** si usa spesso come algoritmo di sostituzione delle pagine ed è considerato valido
- Un algoritmo di sostituzione delle pagine LRU può richiedere una notevole assistenza da parte dell'hardware

Algoritmo di sostituzione delle pagine LRU

Per la realizzazione della sostituzione delle pagine LRU si possono utilizzare due soluzioni:

1. utilizzo di un **contatore**
2. utilizzo di uno **stack** dei numeri delle pagine

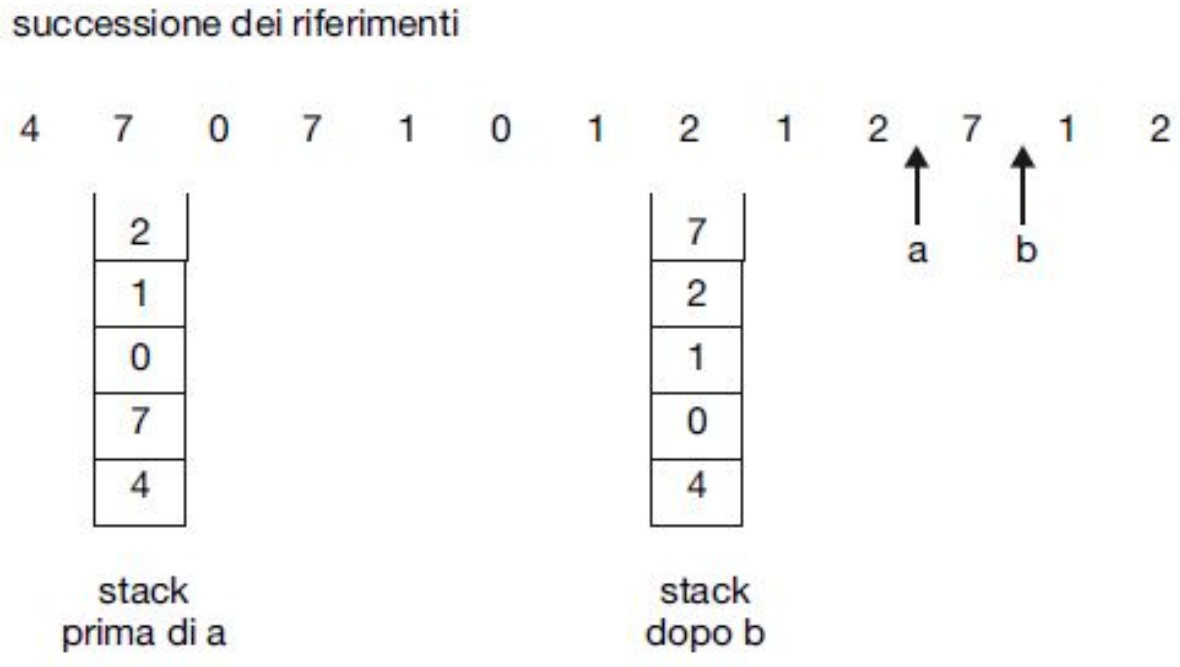


Figura 10.16 Uso di uno stack per registrare i più recenti riferimenti alle pagine.

Anomalia di Belady

Gli algoritmi OPT e LRU appartengono a una classe di algoritmi di sostituzione delle pagine chiamati **algoritmi a stack**

Gli algoritmi OPT e LRU non soffrono dell'**anomalia di Belady**

Altri algoritmi di sostituzione delle pagine

- Sostituzione delle pagine per approssimazione a LRU
- Algoritmo con bit supplementari di riferimento
- Algoritmo con seconda chance
- Algoritmo con seconda chance migliorato
- Sostituzione delle pagine basata su conteggio
- Algoritmi con buffering delle pagine

Bit di riferimento

Il bit di riferimento a una pagina è settato automaticamente dall'hardware del sistema ogni volta che si fa riferimento a quella pagina, che sia una lettura o una scrittura su qualsiasi byte della pagina.

I bit di riferimento sono associati a ciascun elemento della tabella della pagine

Algoritmo di sostituzione delle pagine con seconda chance

L’algoritmo **con seconda chance** è un algoritmo di tipo FIFO, con la seguente modifica:

Dopo aver selezionato la pagina si controlla il bit di riferimento:

- se il valore è 0, si sostituisce la pagina
- se il valore è 1, si dà una seconda chance alla pagina e si passa alla successiva pagina nella lista FIFO

Quando una pagina riceve la seconda chance, si azzera il suo bit di riferimento.

Algoritmo di sostituzione delle pagine con seconda chance

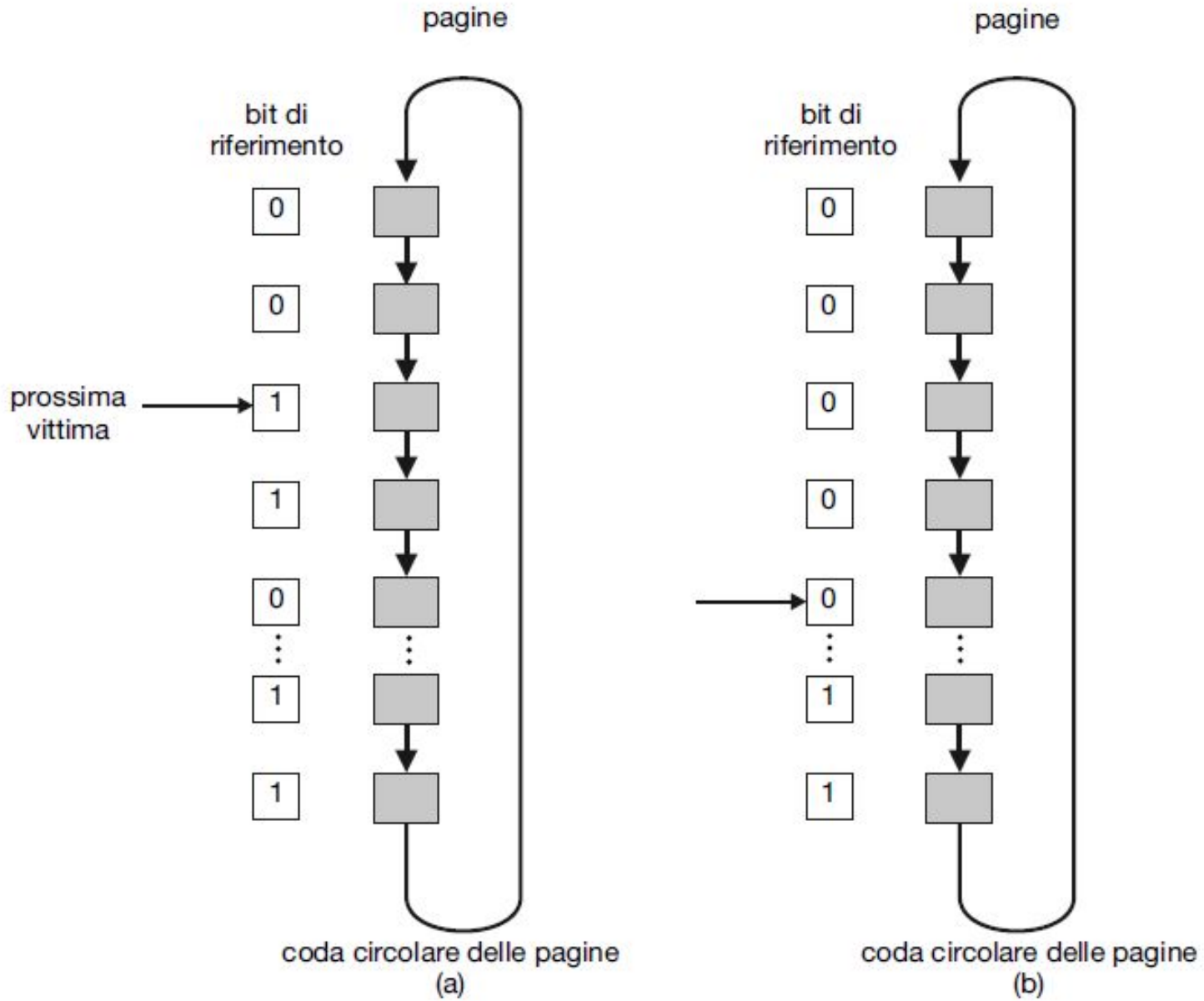


Figura 10.17 Algoritmo di sostituzione delle pagine con seconda chance (orologio).

Algoritmo di sostituzione delle pagine con seconda chance

Poiché l’algoritmo **con seconda chance** è un algoritmo di tipo FIFO, anch’esso soffre dell’anomalia di Belady

Allocazione dei frame

Problema dell'allocazione:

Quale criterio si deve utilizzare per assegnare la memoria libera ai diversi processi che devono essere eseguiti?

Vincoli per l’allocazione dei frame

1. Non si possono assegnare più frame di quanti siano disponibili

2. È necessario assegnare almeno un numero minimo di frame

   ↓

   prestazioni

Vincoli per l’allocazione dei frame

- il numero minimo di frame per ciascun processo è stabilito dell’architettura del sistema

- il numero massimo di frame per ciascun processo è definito dalla quantità di memoria fisica disponibile

Algoritmi di allocazione dei frame

allocazione uniforme

allocazione proporzionale

allocazione locale

allocazione globale

Allocazione uniforme

allocazione uniforme

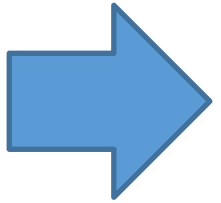
Assumendo di avere *m* frame liberi e *n* processi da servire, si assegnano *m*/*n* frame a ogni processo

Allocazione proporzionale

allocazione proporzionale

La memoria disponibile si assegna a ciascun processo secondo la propria dimensione

- Si assuma di avere *m* frame liberi
- Sia s_i la dimensione di memoria richiesta dal processo p_i
- Si definisce $S = \sum s_i$ e si assegna a p_i una quantità di memoria pari a $s_i / S \times m$



Allocazione proporzionale

allocazione proporzionale

È possibile utilizzare uno schema di allocazione proporzionale in cui il rapporto dei frame non dipende dalle dimensioni relative dei processi, bensì dalle priorità degli stessi (oppure da una combinazione di dimensioni e priorità)

Allocazione locale

allocazione locale

L’allocazione locale prevede che a un processo venga allocata una certa quantità di memoria e che si possa scegliere un frame da allocare solo all’interno di quell’insieme dei frame.

L’allocazione locale utilizza la sostituzione locale, nella quale il numero di blocchi di memoria associati ad un processo non cambia.

La sostituzione locale può penalizzare un processo, non rendendo disponibili pagine di memoria meno usate

Allocazione globale

allocazione globale

È la politica di allocazione più usata.
Viene implementata usando la sostituzione globale: un processo può scegliere un frame per la sostituzione dall'insieme di tutti frame.

Pro: un processo ad alta priorità può aumentare il proprio livello di allocazione dei frame a discapito dei processi a bassa priorità

Contro: Il tasso di page fault di ogni processo non è controllabile poiché dipende anche dal comportamento degli altri processi.

Recupero

La strategia per implementare una politica globale di sostituzione delle pagine è quella di *garantire che ci sia sempre sufficiente memoria libera per soddisfare nuove richieste*

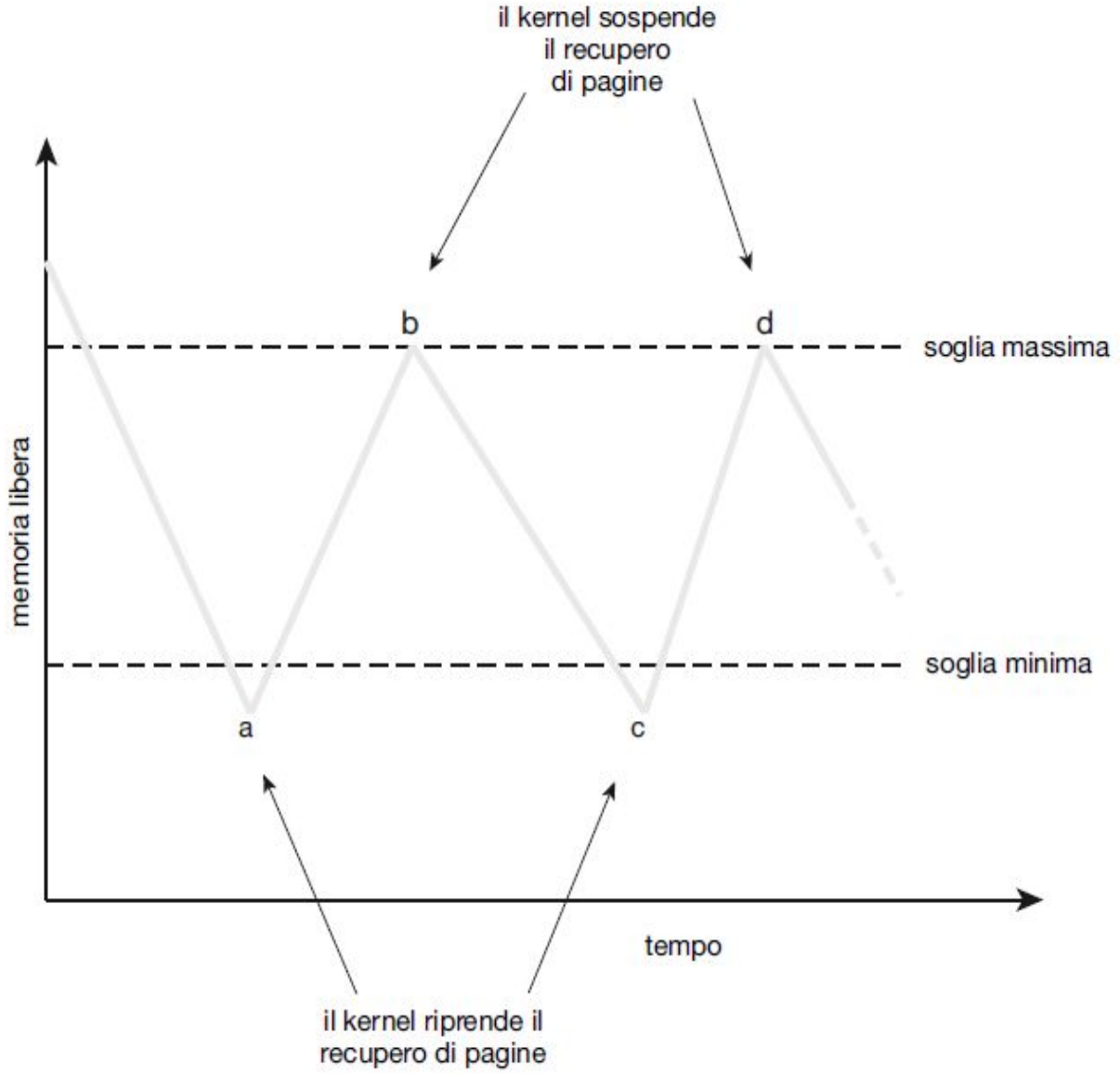


Figura 10.18 Recupero di pagine.

Sistemi con accesso non uniforme in memoria NUMA

Quando un processo incorre in un page fault in un sistema NUMA (Non Uniform Memory Access), se il sistema operativo è conscio dell'architettura NUMA, allora assegna a quel processo un frame il più possibile vicino alla CPU su cui è in esecuzione.

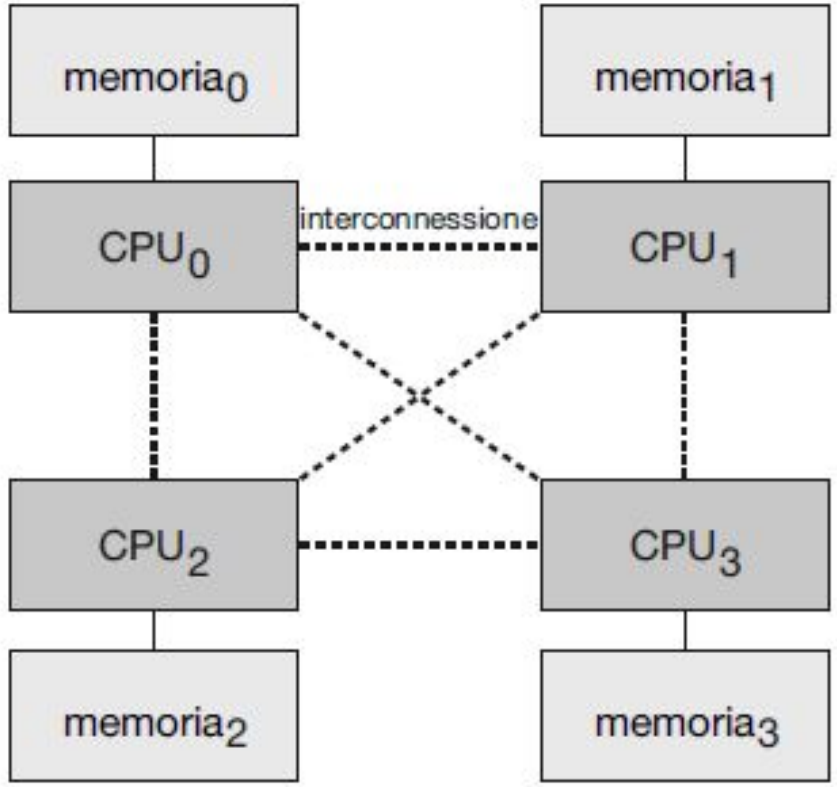


Figura 10.19 Architettura multiprocesso NUMA.

Trashing

Il **thrashing** si verifica quando un sistema spende più tempo per la paginazione rispetto al tempo destinato all'esecuzione.

1. Immaginiamo che un processo non disponga di un numero sufficiente di frame

2. Il processo incorrerà in un page fault e bisognerà sostituire una pagina del processo

3. Se tutte le pagine sono attive, si dovrà scegliere una pagina che molto probabilmente sarà di nuovo necessaria. Pertanto, si verificherà presto un nuovo page fault e si ripeterà il punto 2

Cause del trashing

Il thrashing causa notevoli problemi di prestazioni.

- La multiprogrammazione può portare a problemi di trashing
- Se si supera una certa soglia di multiprogrammazione, l'attività di paginazione degenera e fa crollare l'utilizzo della CPU

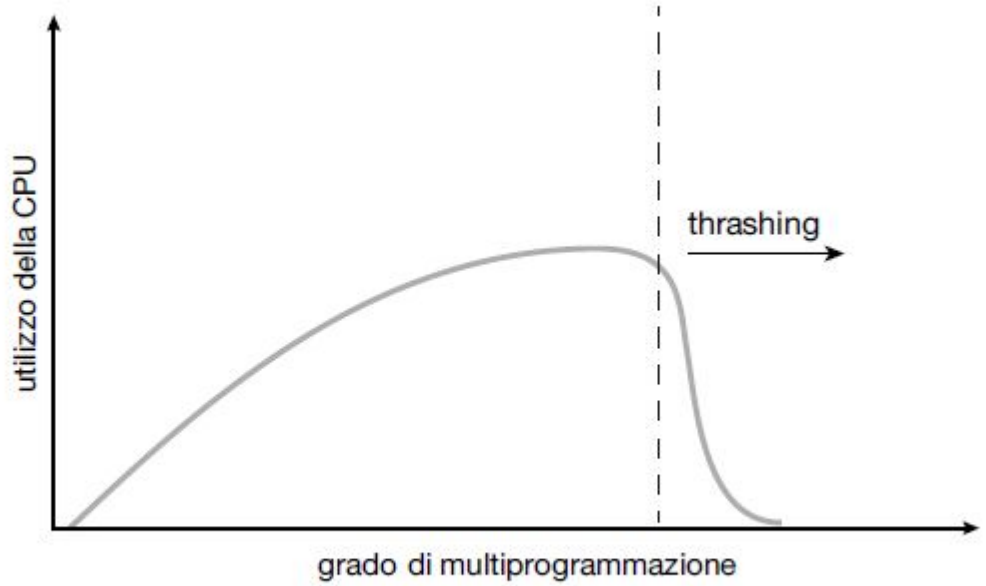


Figura 10.20 Thrashing.

Soluzione al trashing

La soluzione al trashing consiste nel fornire a un processo tutti i frame di cui necessita.

Ma come possiamo farlo?

Località

Una **località** è un set di pagine usate attivamente insieme

La Figura 10.21 illustra il concetto di località e come la località di un processo cambia nel tempo

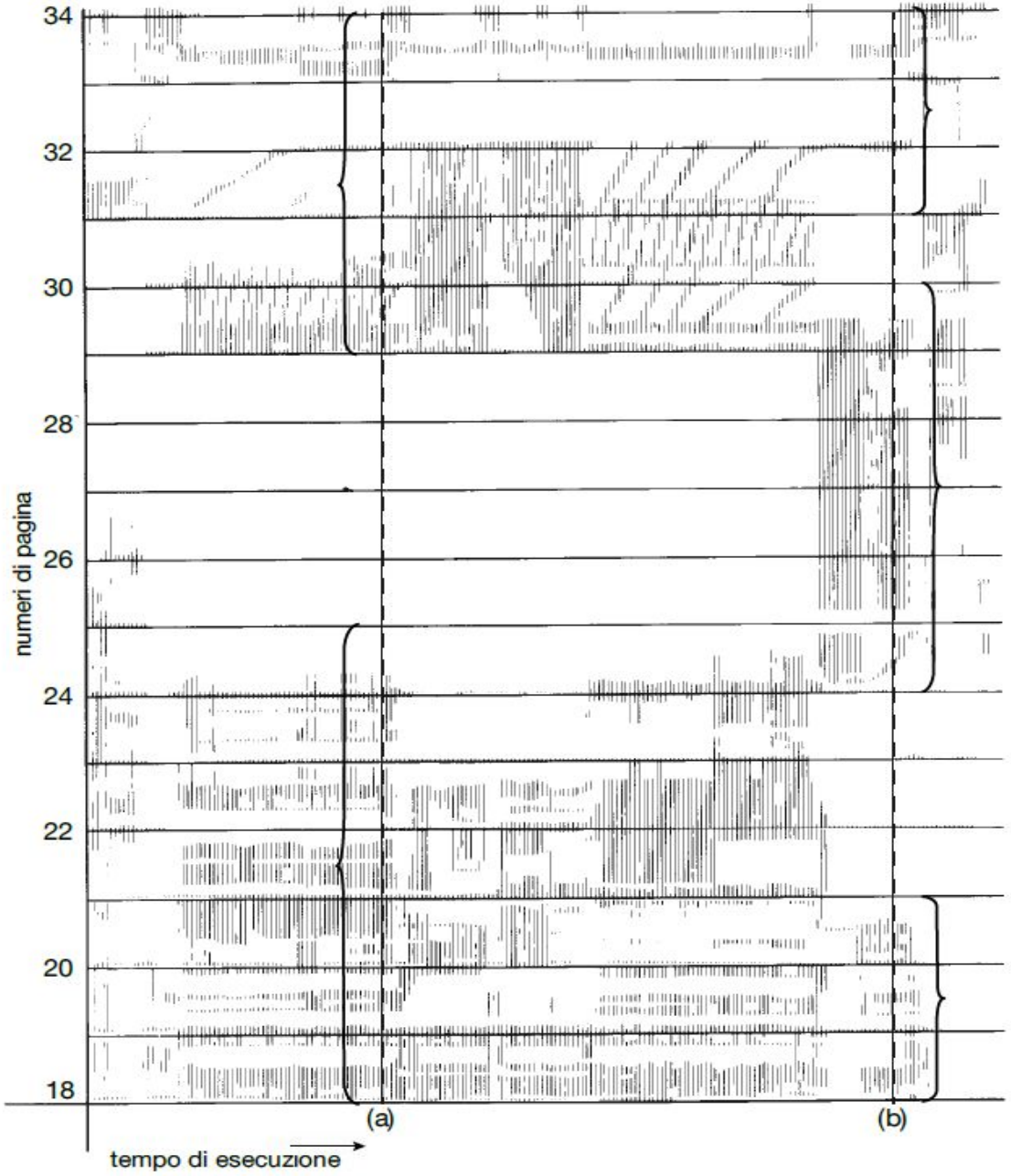


Figura 10.21 Località dei riferimenti alla memoria.

Modello del working set

L'insieme di pagine nei più recenti Δ riferimenti è il **working set** → l'*insieme di pagine utilizzate da un processo in un dato istante*.

Se una pagina è in uso attivo si trova nel working set; se non è più usata esce dal working set Δ unità di tempo dopo il suo ultimo riferimento. Quindi, il working set non è altro che un'approssimazione della località del programma.

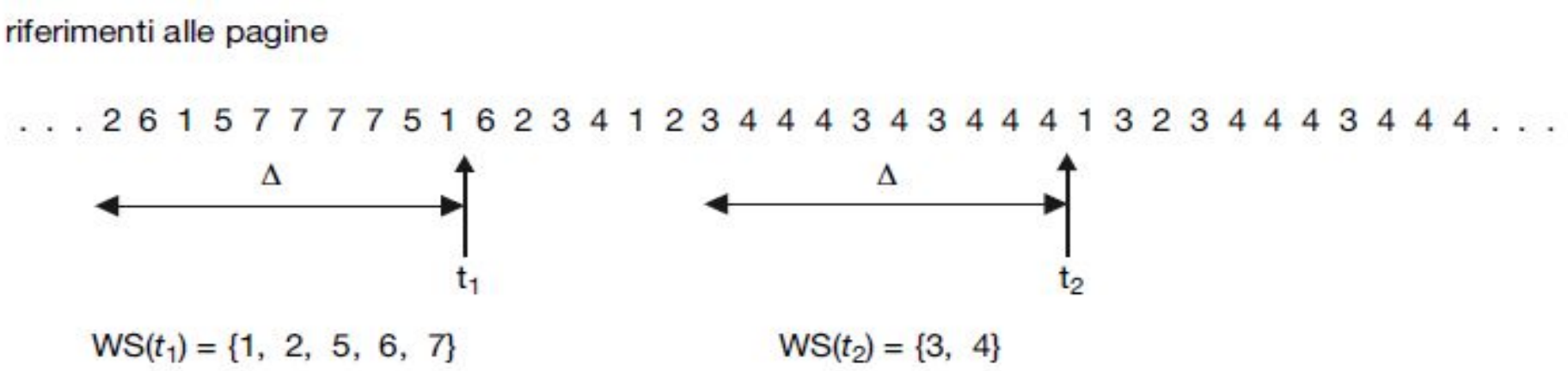


Figura 10.22 Modello del working set.

Frequenza dei page fault

Si può fissare un *limite inferiore* e un *limite superiore* per la frequenza desiderata dei page fault. Se la frequenza effettiva dei page fault per un processo oltrepassa il limite superiore, occorre allocare a quel processo un altro frame; se la frequenza scende sotto il limite inferiore, si sottrae un frame a quel processo → prevenire il thrashing

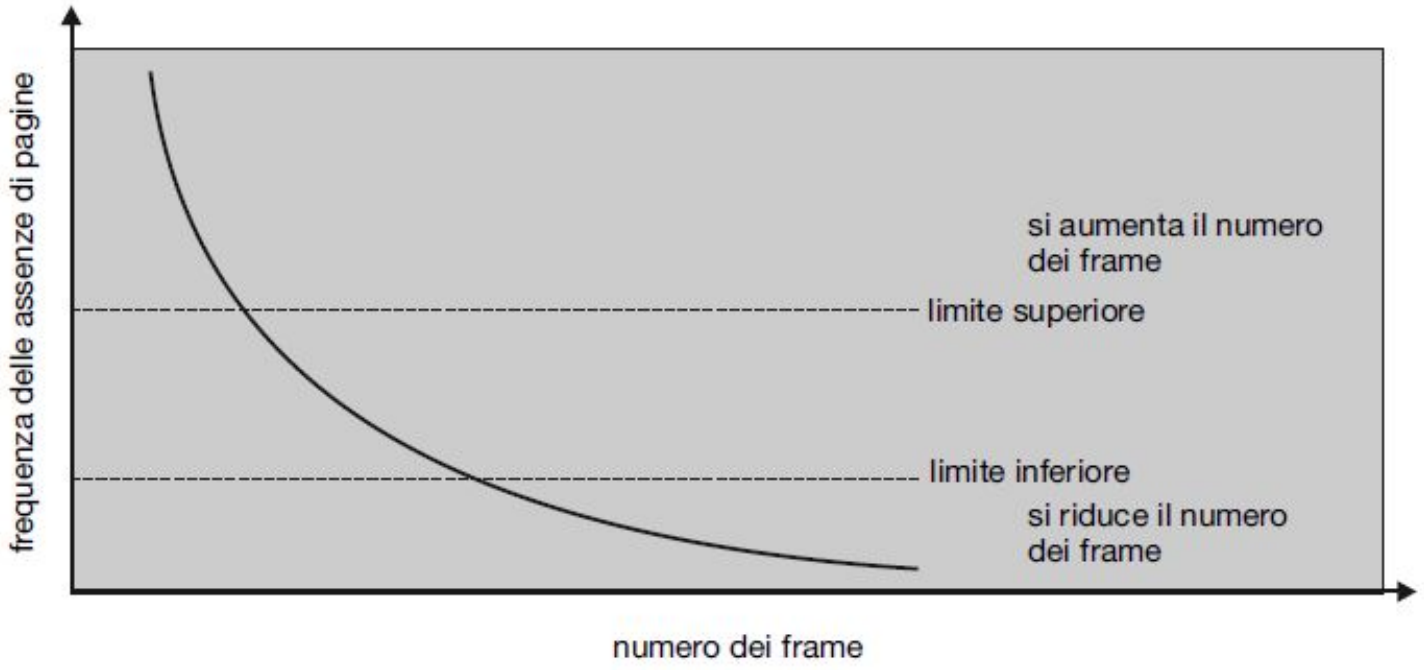


Figura 10.23 Frequenza dei page fault.

Compressione della memoria

La **compressione della memoria** è una tecnica di gestione della memoria che consiste nel comprimere un certo numero di pagine in una singola pagina

La **memoria compressa** è un'alternativa alla paginazione e viene utilizzata su sistemi mobili che non supportano la paginazione

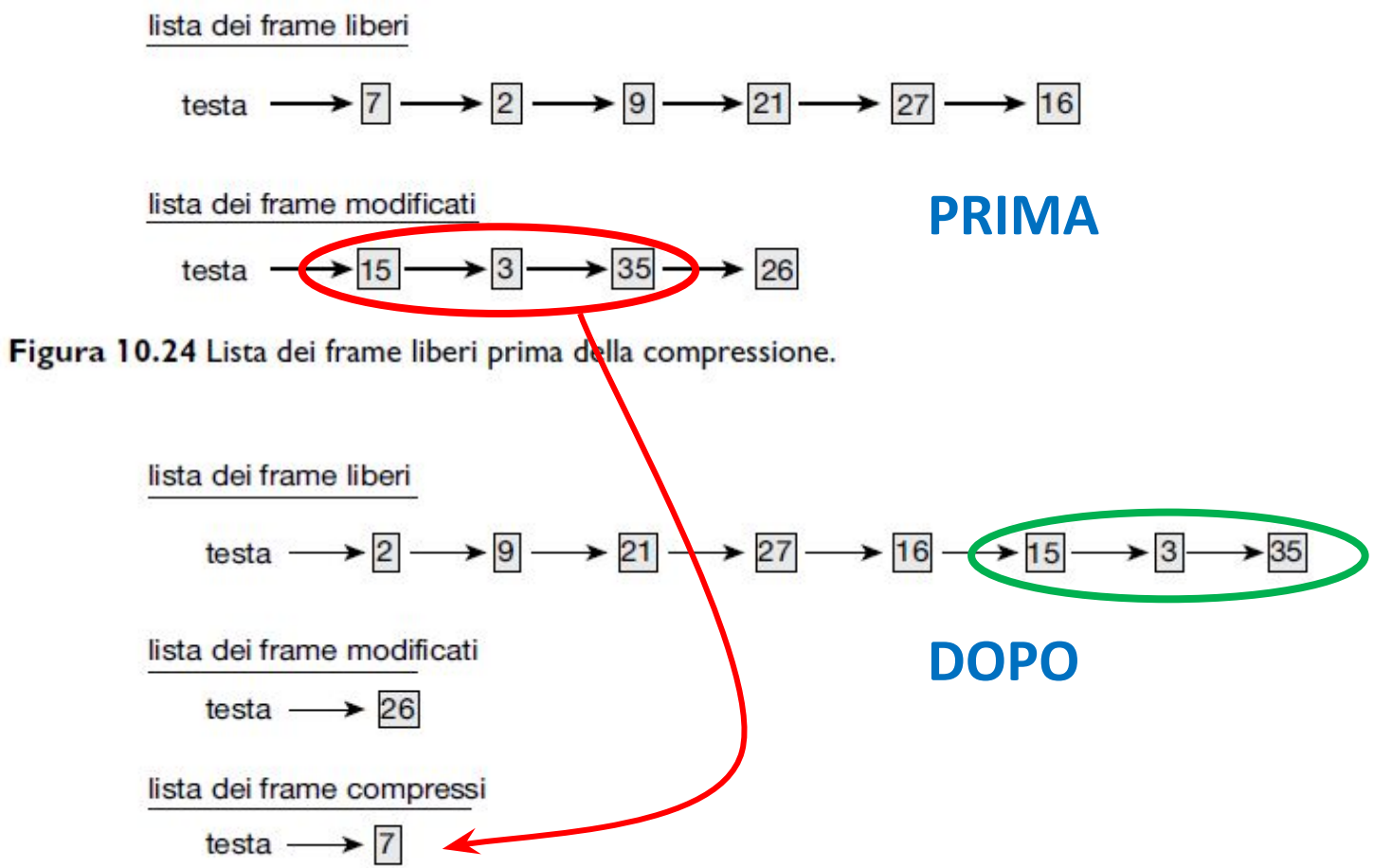


Figura 10.24 Lista dei frame liberi prima della compressione.

Figura 10.25 Lista dei frame liberi dopo la compressione.

Allocazione di memoria del kernel

La memoria del kernel è allocata in modo differente rispetto a quanto avviene per i processi in modalità utente, utilizzando

- blocchi contigui
- di dimensioni variabili

Due tecniche comuni per l'**allocazione della memoria del kernel** sono:

Sistema buddy

Allocazione a lastre (slab)

Sistema buddy

Sistema buddy

Esempio:
Richiesta di memoria pari a 21KB

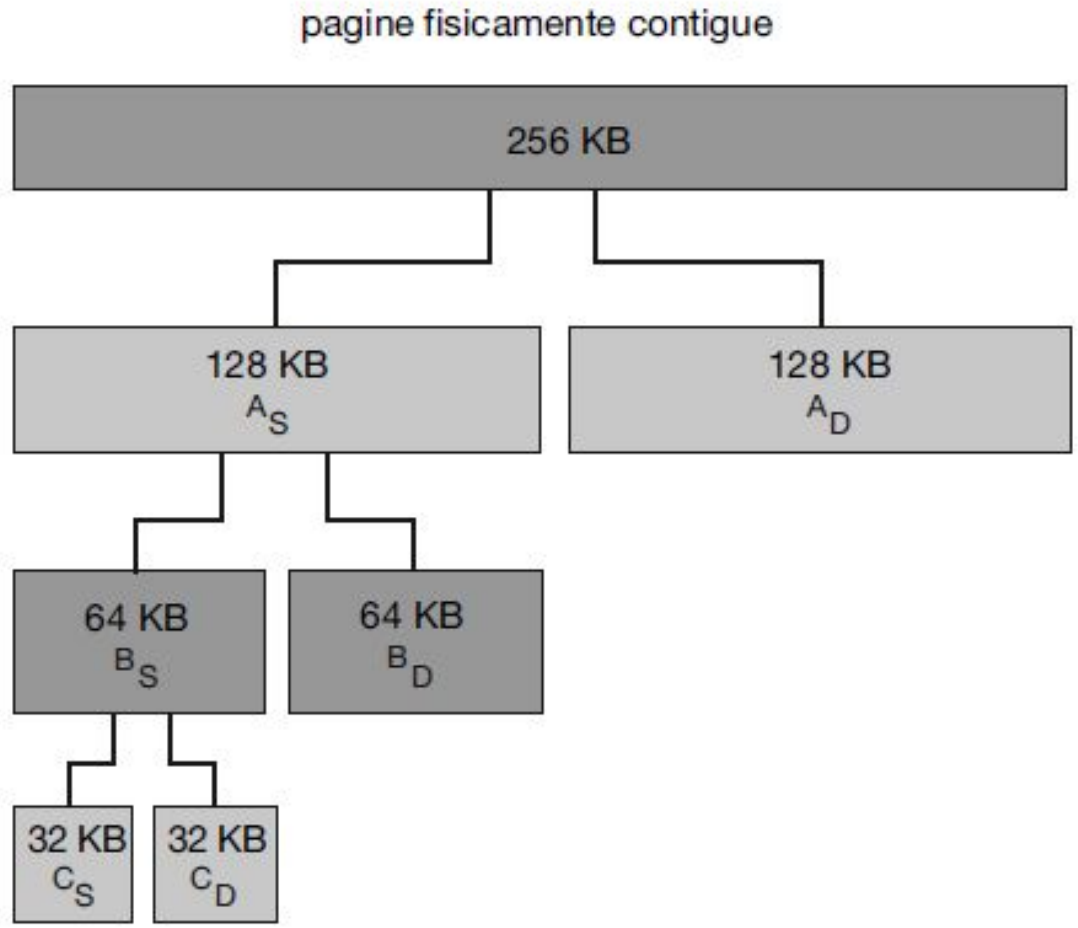


Figura 10.26 Sistema di allocazione buddy.

Allocazione buddy

- Buddy Allocator viene usato per allocare oggetti di dimensione variabile.
- Il buffer viene partizionato ricorsivamente in 2, creando di fatto un albero binario.
- La foglia più piccola che soddisfa la richiesta di memoria sarà utilizzata per soddisfare la richiesta.
- Il buddy associato a una foglia sarà l'altra regione ottenuta dalla divisione del parent.
- Se un oggetto è più piccolo della minima foglia che lo contiene, il restante spazio verrà sprecato.
- Quando un blocco viene rilasciato, esso verrà ricompattato con il suo buddy (se libero), risalendo fino al livello più grande non occupato.

https://sites.google.com/diag.uniroma1.it/sistemi-operativi-1819

Allocazione a lastre (SLAB)

Allocazione a lastre (SLAB)

- Una cache consiste di una o più lastre
- Ogni cache è popolata da oggetti
- Ogni oggetto è una istanza di una struttura dati del kernel

Allocazione SLAB

- Slab Allocator viene usato per allocare oggetti di dimensione fissa.
- Può allocarne fino ad un numero massimo fissato.

https://sites.google.com/diag.uniroma1.it/sistemi-operativi-1819

Hit ratio del TLB

Tasso di successi (*hit ratio)* di un TLB

- è la percentuale di traduzioni di indirizzi virtuali risolte dal TLB anziché dalla tabella delle pagine
- è proporzionale al numero di elementi del TLB

Portata del TLB

Portata del TLB

- è il numero di elementi moltiplicato per la dimensione delle pagine
- esprime la quantità di memoria accessibile dal TLB

Una tecnica per aumentare la portata del TLB è aumentare la dimensione delle pagine.

Esempi di sistemi operativi

Realizzazione della memoria virtuale in:

Linux, Windows e Solaris gestiscono la memoria virtuale in modo simile, utilizzando, tra l'altro, la paginazione su richiesta e la copia su scrittura.

Ogni sistema utilizza anche una variante per approssimazione di LRU nota come algoritmo a orologio.

Linux

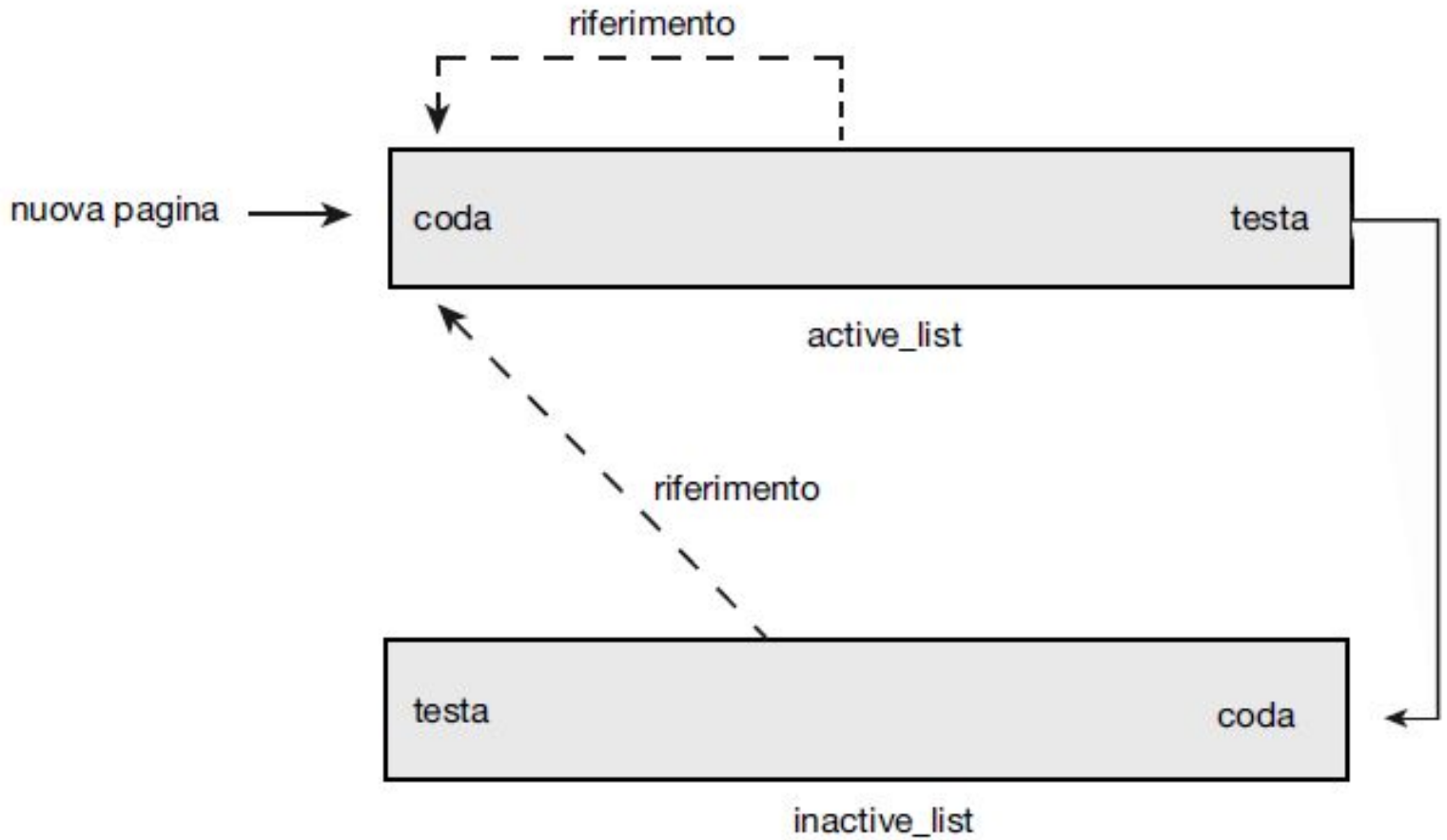


Figura 10.29 Le strutture `active_list` e `inactive_list` di Linux.