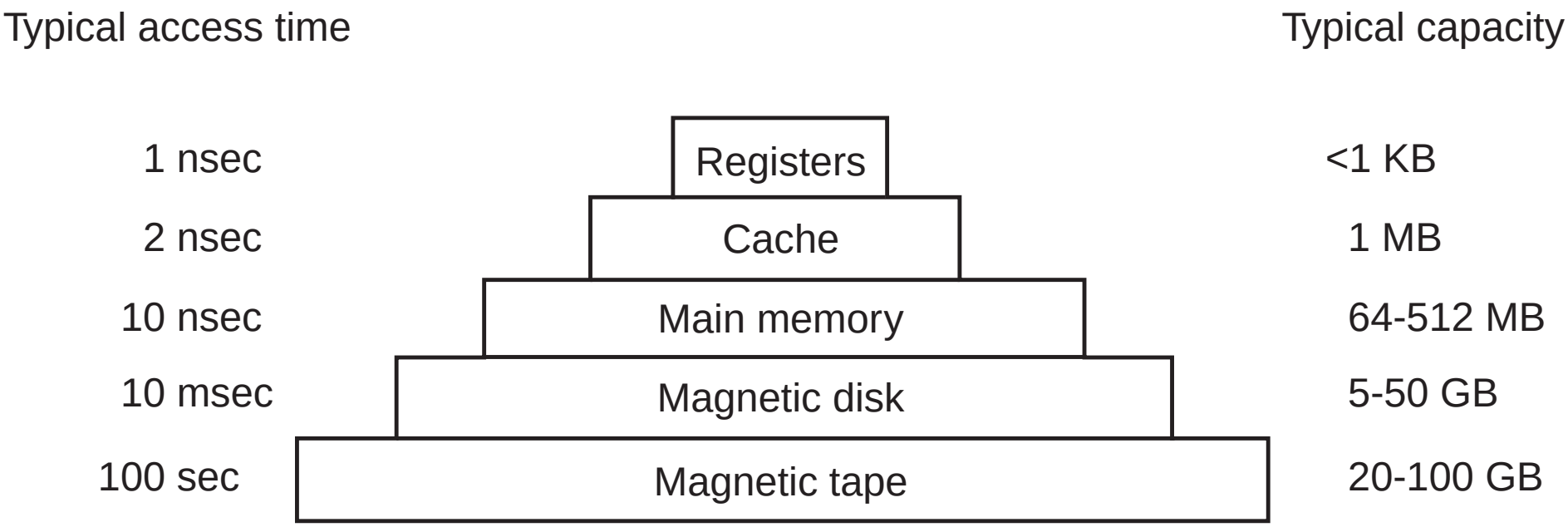
Gestione della Memoria Principale

- Fondamenti
- Binding, Loading, Linking
- Spazio indirizzi logico vs. fisico
- Allocazione contigua
- Paginazione
- Segmentazione

Fondamenti

- La memoria di un elaboratore e' organizzata nella seguente gerarchia



- La parte del sistema operativo che gestisce la memoria principale si chiama **memory manager**
- In alcuni casi, il memory manager può gestire anche parte della memoria secondaria, al fine di emulare memoria principale

Fondamenti

La gestione della memoria mira a soddisfare i seguenti requisiti:

- **Organizzazione logica:**
  allocare e deallocare memoria ai processi su richiesta trattando i vari tipi di memoria (cache, main, dischi) come un'unica entita'

- **Organizzazione fisica:**
  tenere traccia della memoria libera e occupata

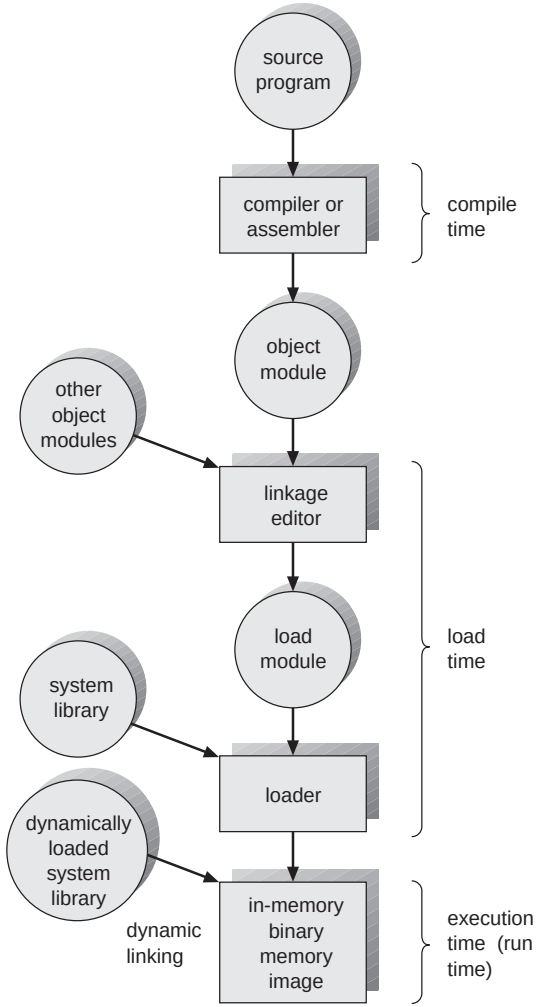
- **Binding**:
  mappare indirizzi logici in indirizzi fisici

- **Protezione**:
  coordinare l'accesso alla memoria da parte dei processi utente e di sistema

- **Condivisione**:
  aumentare l'efficienza nella gestione della risorsa memoria

Dal codice sorgente al codice eseguibile

Per capire meglio il funzionamento del MM rivediamo le fasi di gestione del codice



Dal codice sorgente al codice eseguibile

- Compile-time
  - Compilazione: il codice scritto nel linguaggio ad alto livello viene tradotto in codice in linguaggio macchina
  - Linking: il codice delle librerie importate viene incluso nel codice oggetto
- Load-time
  - il codice binario viene caricato in memoria principale
- Execution-time
  - il codice oggetto viene eseguito dalla CPU (che puo' inviare richieste di lettura scrittura alla memoria principale)

Indirizzi logici e indirizzi fisici

- Spazio di indirizzamento logico
  - ogni processo associato ad uno spazio di indirizzamento logico
  - gli indirizzi usati in un processo sono indirizzi logici, ovvero riferimenti a questo spazio di indirizzamento

    (in generale indirizzi logici = indirizzi di memoria generati dalla CPU)

- Spazio di indirizzamento fisico
  - ad ogni indirizzo logico corrisponde un indirizzo fisico
  - il sistema operativo (e/o hardware) si occupa della traduzione da indirizzi logici a indirizzi fisici

Binding

- Con il termine binding si indica l'associazione di indirizzi di memoria ai dati e alle istruzioni di un programma

- Il binding può avvenire
  - durante la compilazione
  - durante il caricamento
  - durante l'esecuzione

Binding durante la compilazione

- Gli indirizzi vengono calcolati al momento della compilazione e resteranno gli stessi ad ogni esecuzione del programma
- Il codice generato viene detto codice assoluto
- Il codice deve essere ricompilato ogni volta che si cambia locazione di esecuzione
- Ad es. codice per microcontrollori o per il kernel
- É semplice e veloce e non richiede hardware speciale ma non funziona con la multiprogrammazione

Binding durante il caricamento

- Il codice generato dal compilatore non contiene indirizzi assoluti ma relativi (solitamente rispetto ad un indirizzo base)

- Questo tipo di codice viene detto **rilocabile**

- Il programma **loader** si preoccupa di aggiornare tutti i riferimenti agli indirizzi di memoria coerentemente al punto iniziale di caricamento

- Permette di gestire multiprogrammazione e non richiede uso di hardware particolare

- Richiede una traduzione degli indirizzi da parte del loader, e quindi un formato particolare dei file eseguibili

Binding durante l'esecuzione

- l'individuazione dell'indirizzo di memoria effettivo viene effettuata durante l'esecuzione da un componente hardware apposito:

  la memory management unit (MMU)

- Il programma può essere spostato da una zona all'altra della memoria durante l'esecuzione.

Esempi di MMU

L'esempio piu' semplice di MMU si basa sul **registro di rilocazione**

- Se il valore del registro di rilocazione R, uno spazio logico 0...Max viene tradotto in uno spazio fisico R...R+MAX

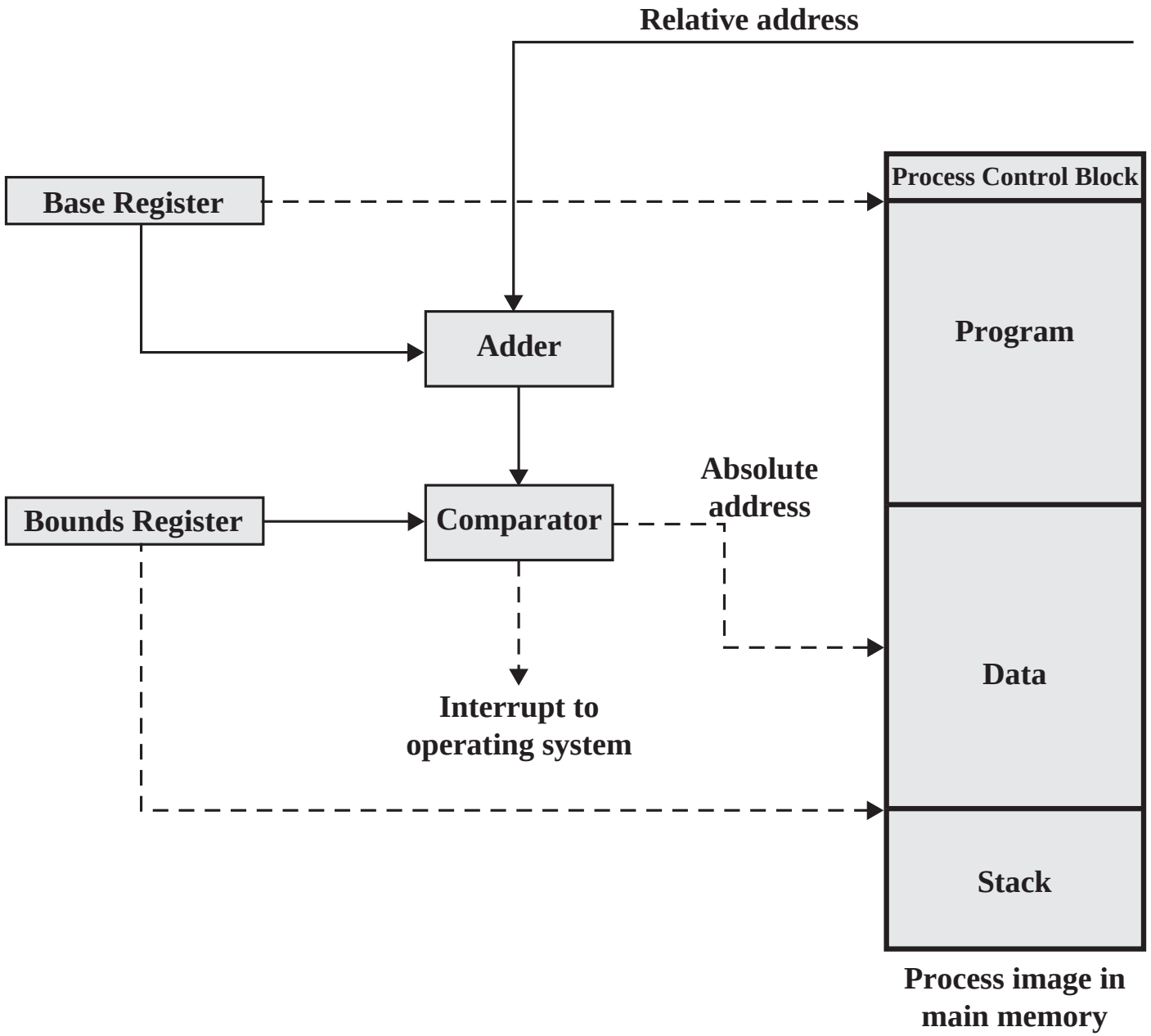
- Esempio: processori Intel 80x86

Esempi di MMU

Un'altro esempio di MMU si basa sui **registri di rilocazione e di limite**

- il registro limite viene utilizzato per implementare meccanismi di protezione della memoria
- es. se l'indirizzo logico e' $>$ del valore del registro limite si genera un errore
- se si passa il test con il registro limite si utilizza il registro di rilocazione per calcolare l'indirizzo fisico

Struttura generale della MMU



Loading (caricamento) dinamico

- consente di poter caricare alcune routine di libreria solo quando vengono richiamate
- Tutte le routine a caricamento dinamico risiedono su un disco (codice rilocabile), quando servono vengono caricate
- Le routine poco utili (e.g., casi di errore rari...) non vengono caricate in memoria al caricamento dell'applicazione
- Spetta al programmatore utilizzare questa possibilita'
- Il sistema operativo fornisce semplicemente una libreria che implementa le funzioni di caricamento dinamico

Linking (collegamento) dinamico

- Linking statico
  - se il linker collega e risolve tutti i riferimenti dei programmi...
  - le routine di libreria vengono copiate in ogni programma che le usa (e.g. printf in tutti i programmi C)
- Linking dinamico
  - e' possibile posticipare il linking delle routine di libreria al momento del primo riferimento durante l'esecuzione
  - consente di avere eseguibili pi compatti
  - le librerie vengono implementate come codice reentrant:

    esiste una sola istanza della libreria in memoria e tutti i programmi eseguono il codice di questa istanza
  - Esempi: le .so su Unix, le .DLL su Windows.

Linking dinamico

- Vantaggi
  - risparmio di memoria
  - consente l'aggiornamento automatico delle versioni delle librerie (le librerie aggiornate sono caricate alla successiva attivazione dei programmi)
- Svantaggi
  - bisogna fare attenzione a tener traccia delle versioni
  - richiede un supporto da parte del sistema operativo per far condividere segmenti di codice tra più processi.

Allocazione dei processi in memoria

- E' una delle funzioni principali del gestore di memoria
- Consiste nel reperire ed assegnare uno spazio di memoria fisica a un programma che viene attivato
- oppure per soddisfare ulteriori richieste effettuate dai programmi durante la loro esecuzione

Allocazione - Definzioni

- Allocazione contigua
  - tutto lo spazio assegnato ad un programma deve essere formato da celle consecutive
- Allocazione non contigua
  - e' possibile assegnare a un programma aree di memorie separate
- La MMU deve essere in grado di gestire la conversione degli indirizzi in modo coerente
- Esempio: la MMU basata su rilocazione gestisce solo l'allocazione contigua

Allocazione statica e dinamica

- Allocazione statica
  - un programma deve mantenere la propria aerea di memoria dal caricamento alla terminazione
  - non e' possibile rilocare il programma durante l'esecuzione
- Allocazione dinamica
  - durante l'esecuzione, un programma puo' essere spostato all'interno della memoria

Allocazione contigua a Partizionamento Fissato

- La memoria disponibile (quella non occupata dal s.o.) viene suddivisa in partizioni (di dimensione fissata)

- Ogni processo viene caricato in una delle partizioni libere che ha dimensione sufficiente a contenerlo

- E' uno schema di allocazione contigua e statica

- Esempio:
  - Spazio di indizzi del sistema operativo: [0-R] (parte bassa della memoria)
  - Processi utente: [R+1-Max]

Frammentazione Interna

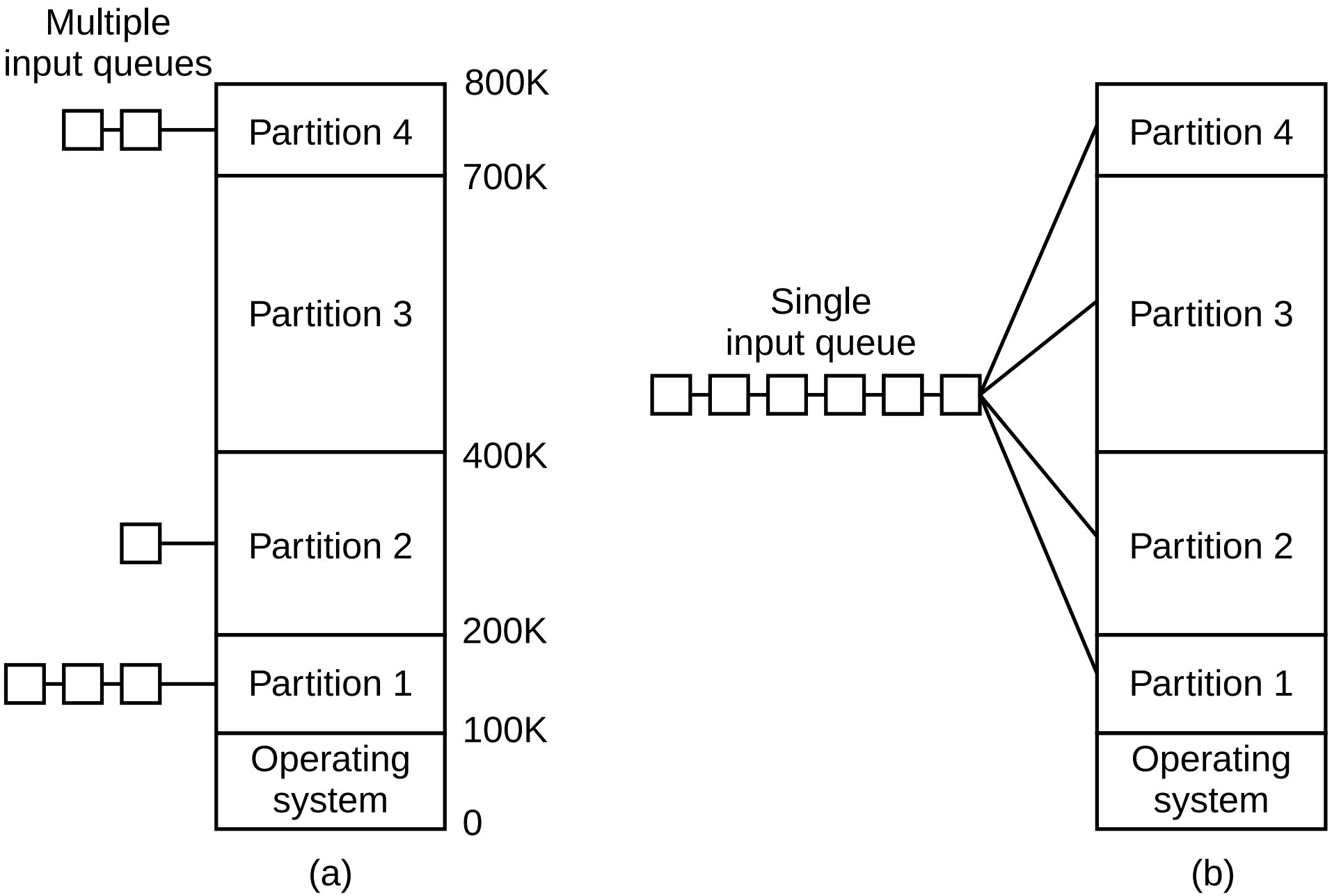
- Se la dimensione della partizione allocata ad un processo è superiore a quella necessaria al processo la parte di memoria rimanente viene sprecato

- Questo fenomeno viene chiamato **frammentazione interna**

- A causa di questo problema questo tipo di allocazione viene utilizzato solo per sistemi embedded

Gestione Partizioni - Code di input

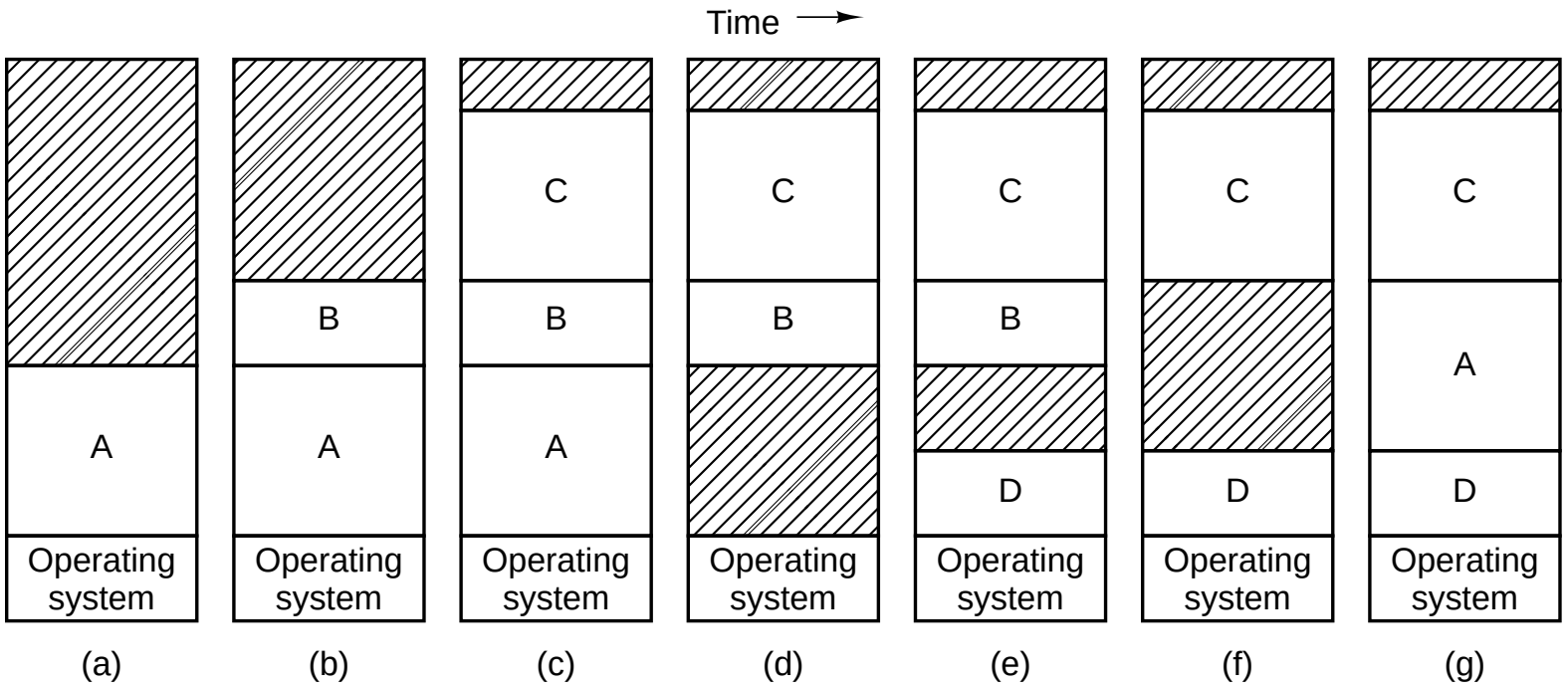
- Il sistema operativo puo' utilizzare delle code di input per scegliere come allocare le partizioni ai processi
- Quando arriva un processo, gli viene allocata completamente una partizione tra quelle libere
- Se si utilizza una coda per ogni partizione si corre il rischio si sottoutilizzare memoria
- Se si mantiene una sola coda per tutte le partizioni si pone il problema della mappatura processo $\rightarrow$ partizione
- Ci sono varie strategie:
  - **First-fit**: per ogni partizione si seleziona il primo processo nella coda con dimensione minore uguale della dimensione della partizione
  - **Best-fit**: per ogni partizione si seleziona il processo piu' grande tra i processi nella coda che hanno dimensione minore uguale della dimensione della partizione


Multiple
input queues
Partition 4
Partition 3
Partition 2
Partition 1
Operating
system
800K
700K
400K
200K
100K
0
(a)
Single
input queue
Partition 4
Partition 3
Partition 2
Partition 1
Operating
system
(b)

Allocazione contigua a Partizionamento Dinamico

- I processi vengono sempre allocati in uno spazio contiguo di celle
- La memoria disponibile (nella quantita' richiesta) viene assegnata ai processi che ne fanno richiesta
- Nella memoria possono essere presenti diverse zone inutilizzate: per effetto della terminazione di processi oppure per non completo utilizzo dell'area disponibile da parte dei processi attivi
- Il sistema operativo deve mantenere traccia delle partizioni allocate e degli spazi liberi

Esempio



Frammentazione Esterna

- Dopo un certo numero di allocazioni e deallocazioni di memoria dovute all'attivazione e alla terminazione dei processi lo spazio libero appare suddiviso in piccole aree

- E' il fenomeno della **frammentazione esterna**

Compattazione

- Se e' possibile rilocare i programmi durante la loro esecuzione, e' allora possibile procedere alla compattazione della memoria

- Compattare la memoria significa spostare in memoria tutti i programmi in modo da riunire tutte le aree inutilizzate

- E' un operazione volta a risolvere il problema della frammentazione esterna

- E' un operazione molto onerosa: occorre copiare (fisicamente) in memoria grandi quantit di dati

- Non puo' essere utilizzata in sistemi interattivi: i processi devono essere fermi durante la compattazione

Allocazione dinamica - Strutture dati

- Quando la memoria assegnata dinamicamente abbiamo bisogno di una struttura dati per mantenere informazioni sulle zone libere e sulle zone occupate

- Strutture dati possibili: mappe di bit, liste con puntatori

Mappa di bit

- Struttura dati
  - La memoria viene suddivisa in unit di allocazione ad ogni unit di allocazione corrisponde un bit in una bitmap
  - Le unit libere sono associate ad un bit di valore 0, le unit occupate sono associate ad un bit di valore 1
- La mappa di bit ha una dimensione fissa e calcolabile a priori
- per individuare uno spazio di memoria di dimensione k unita', e' necessario cercare una sequenza di k bit 0 consecutivi
- in generale, tale operazione e' O(m), dove m rappresenta il numero di unit di allocazione

Liste di puntatori

- Struttura dati
  - Si mantiene una lista dei blocchi allocati e liberi di memoria
  - Ogni elemento della lista specifica
    * Se si tratta di un processo (P) o di un blocco libero (hole, H)
    * la dimensione (inizio/fine) del segmento

- Allocazione: Quando un blocco libero viene selezionato viene suddiviso in due parti: un blocco processo della dimensione desiderata e un blocco libero con quanto rimane del blocco iniziale

- Deallocazione: A seconda dei blocchi vicini, lo spazio liberato puo' creare un nuovo blocco libero, oppure essere accorpato ai blocchi vicini

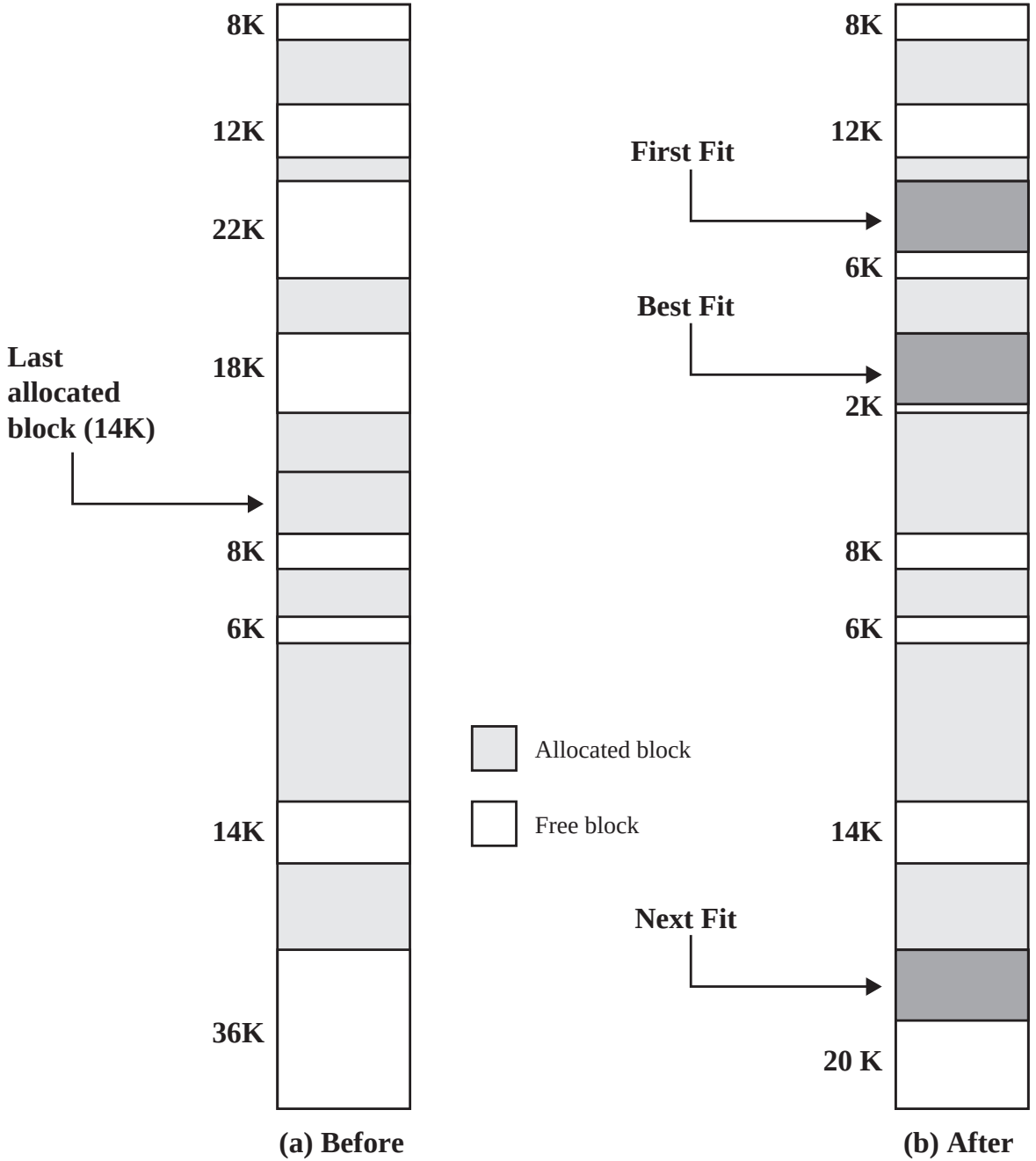
Allocazione dinamica - Selezione blocco libero

L'operazione di selezione di un blocco libero concettualmente indipendente dalla struttura dati

- **First-fit:** Alloca il *primo* spazio libero sufficientemente grande
- **Next-fit:** Alloca il *primo* spazio libero sufficientemente grande a partire dall'ultimo usato.
- **Best-fit:** Alloca il *più piccolo* spazio libero sufficientemente grande. Deve scandire l'intera lista (a meno che non sia ordinata). Produce il più piccolo spazio libero di scarto.
- **Worst-fit:** Alloca il *più grande* spazio libero sufficientemente grande. Deve scandire l'intera lista (a meno che non sia ordinata). Produce il più grande spazio libero di scarto.

In generale, gli algoritmi migliori sono il first-fit e il next-fit. Best-fit tende a frammentare molto. Worst-fit è più lento.

Allocazione contigua: esempi di allocazione



Allocazione non contigua: Paginazione

Paginazione

- I meccanismi visti fino ad ora (partizionamento fisso/dinamico) non sono efficienti nell'uso della memoria (frammentazione interna/esterna)

- La paginazione e' l'approccio utilizzato nei S.O. moderni per ridurre il fenomeno di frammentazione interna e minimizzare il fenomeno della frammentazione esterna

- Attenzione pero': necessita di hardware adeguato

Paginazione - Definizione

- Lo **spazio di indirizzamento logico** di un processo viene suddiviso in un insieme di blocchi di dimensione fissa chiamati **pagine**

- La **memoria fisica** . viene suddivisa in un insieme di blocchi della stessa dimensione delle pagine, chiamati **frame**

- Quando un **processo viene allocato in memoria**:

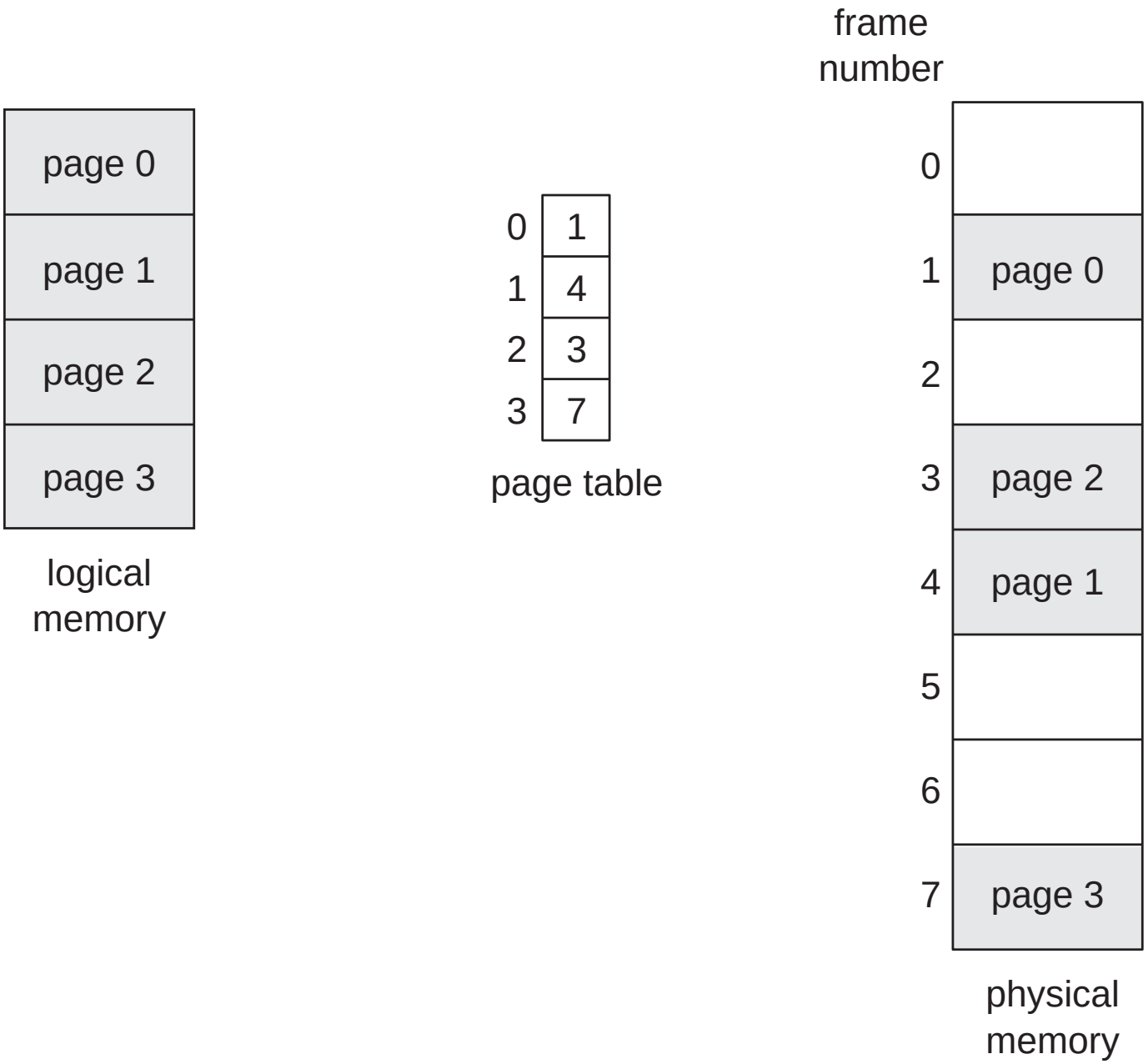
- vengono reperiti ovunque in memoria un numero sufficiente di frame per contenere le pagine del processo

- Il S.O. tiene traccia dei frame liberi

- Per eseguire un programma di n pagine, servono n frame liberi in cui caricare il programma

- Il S.O. mantiene quindi anche una **Page Table** (tabella delle pagine) per tradurre indirizzi logici in indirizzi fisici.

Esempio di paginazione

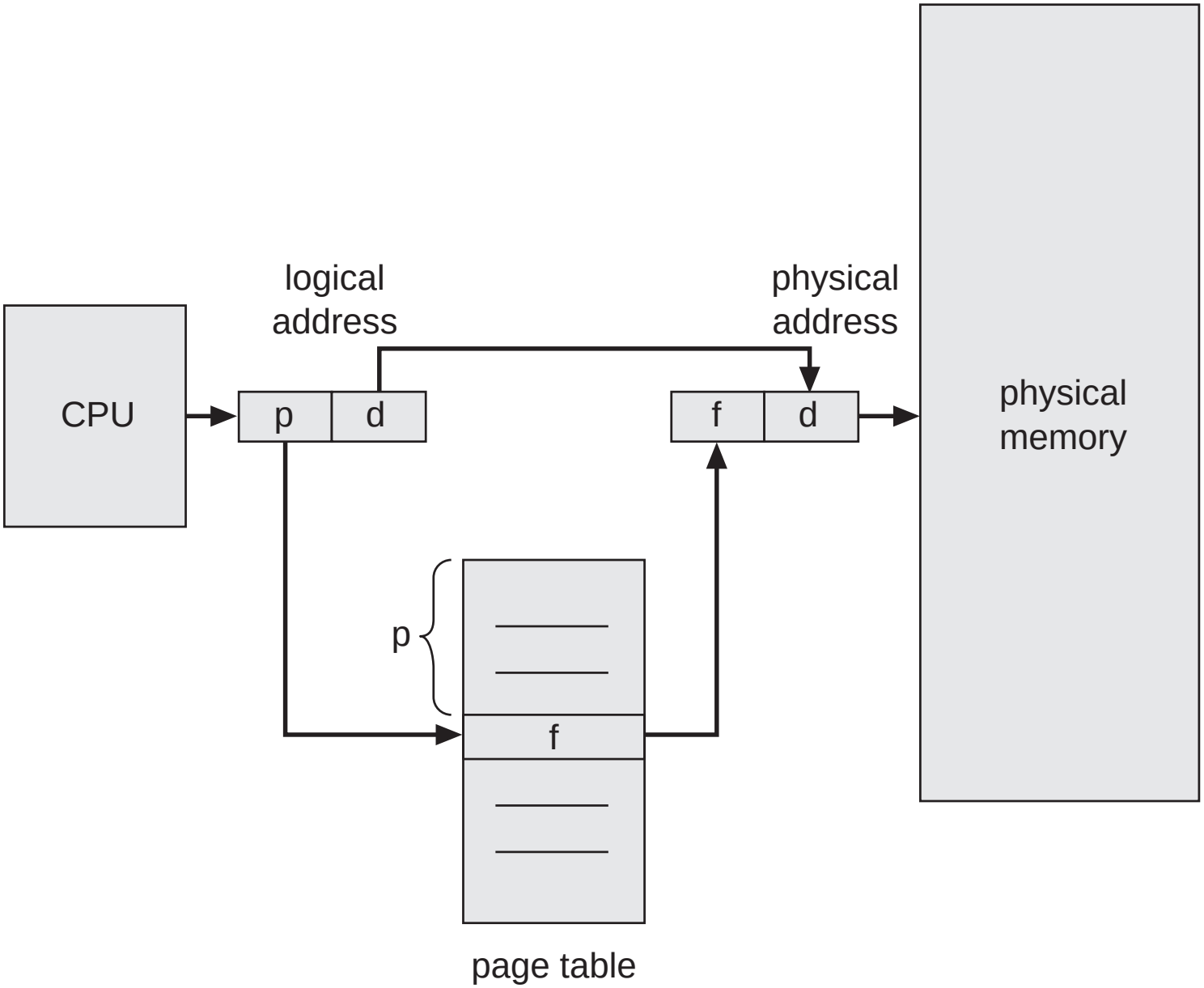


Schema di traduzione degli indirizzi

L'indirizzo generato dalla CPU viene diviso in

- *Numero di pagina* p*:* usato come indice in una *page table* che contiene il numero del frame contenente la pagina p.

- *Offset di pagina* d*:* combinato con il numero di frame fornisce l'indirizzo fisico da inviare alla memoria.


CPU
logical address
p
d
physical address
f
d
physical memory
p
f
page table

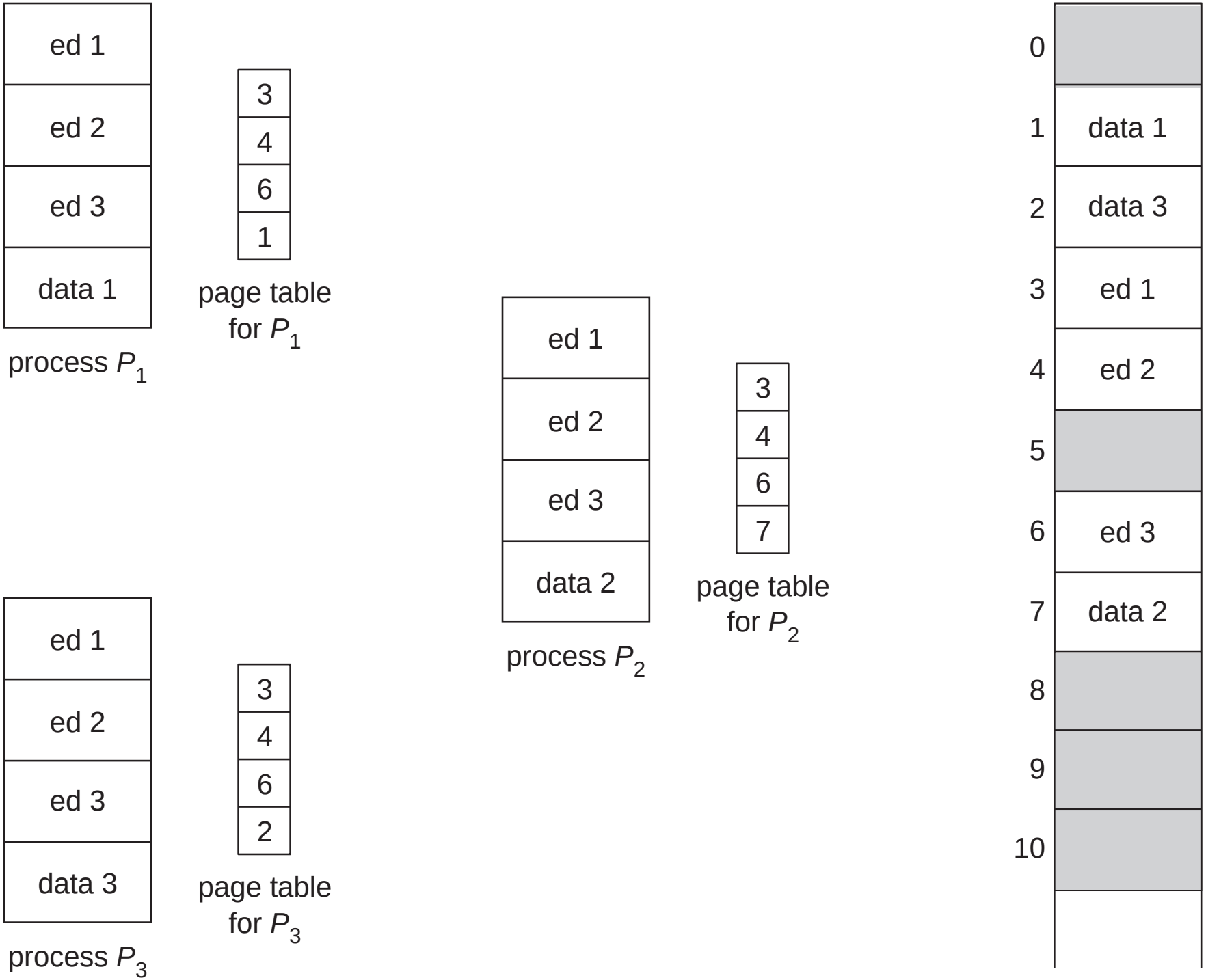
Paginazione: condivisione

La paginazione permette la condivisione del codice

- Una sola copia di codice read-only può essere condivisa tra più processi. Il codice deve essere *rientrante* (separare codice eseguibile da record di attivazione). Es.: editors, shell, compilatori, . . .

- Il codice condiviso appare nelle stesse locazioni logiche per tutti i processi che vi accedono

- Ogni processo mantiene una copia separata dei proprî dati


ed 1
ed 2
ed 3
data 1
process P_1
3
4
6
1
page table for P_1
ed 1
ed 2
ed 3
data 2
process P_2
3
4
6
7
page table for P_2
ed 1
ed 2
ed 3
data 3
process P_3
3
4
6
2
page table for P_3
0
1
data 1
2
data 3
3
ed 1
4
ed 2
5
6
ed 3
7
data 2
8
9
10

Paginazione: protezione

- La protezione della memoria è implementata associando bit di protezione ad ogni frame.

- *Valid* bit collegato ad ogni entry nella page table

  - "valid" = indica che la pagina associata è nello spazio logico del processo, e quindi è legale accedervi

  - "invalid" = indica che la pagina non è nello spazio logico del processo $\Rightarrow$ violazione di indirizzi (Segment violation)

Allocazione non contigua: Segmentazione

Segmentazione

In un sistema con segmentazione la memoria associata ad un programma suddivisa in aree differenti (**segmenti**) dal punto di vista funzionale

- aree text:
  - contengono il codice eseguibile e sono normalmente in sola lettura (solo i virus cambiano il codice)
  - possono essere condivise tra piu' processi (codice reentrant)
- aree dati: possono essere condivise oppure no
- area stack: non puo' assolutamente essere condivisa

Segmentazione

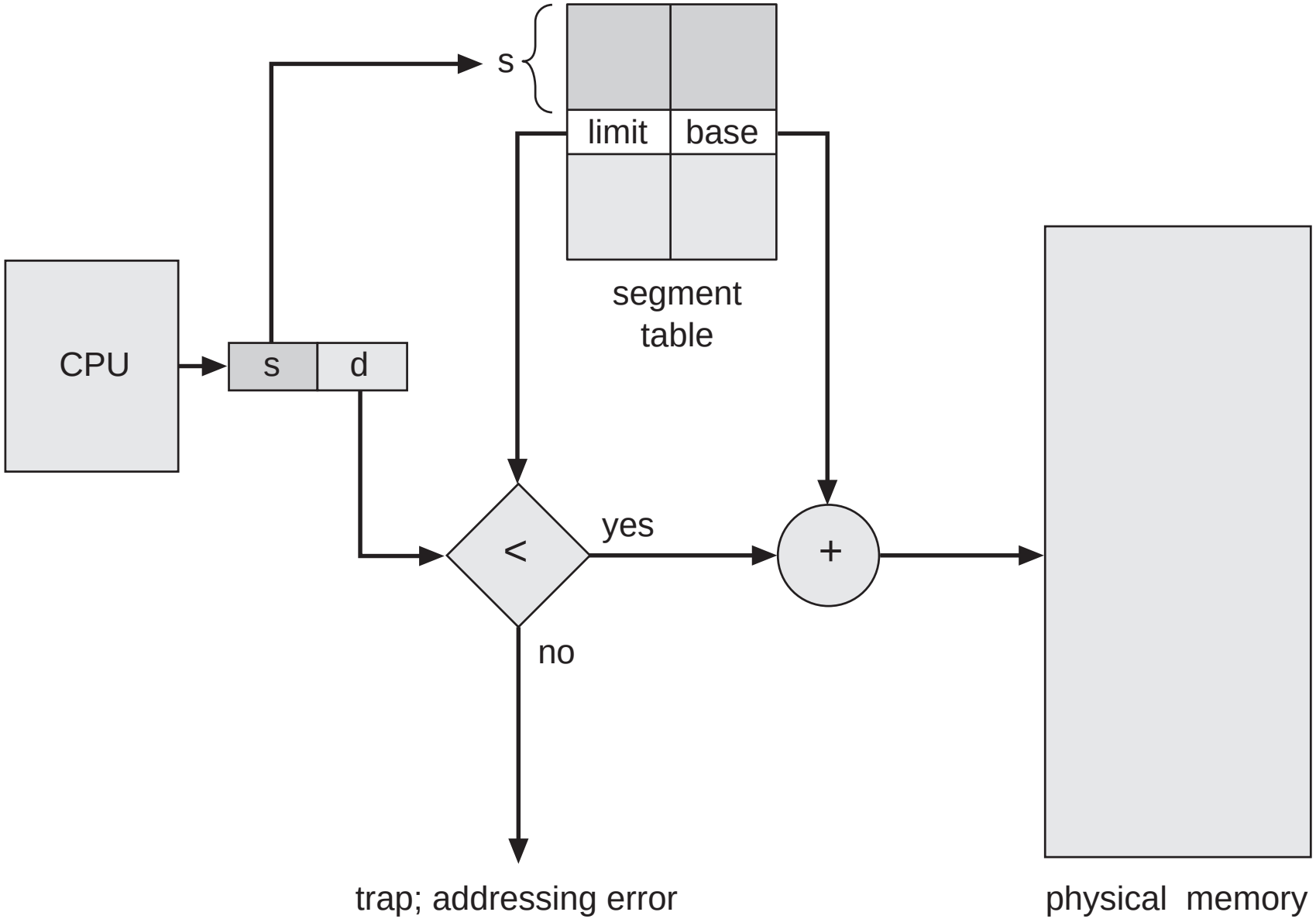
In un sistema basato su segmentazione

- uno spazio di indirizzamento logico e' dato da un insieme di segmenti
- un segmento e' un'area di memoria (logicamente continua) contenente elementi tra loro affini
- ogni segmento e' caratterizzato da un nome (normalmente un indice) e da una lunghezza
- ogni riferimento di memoria e' dato da una coppia $\langle$nome segmento, offset$\rangle$
- Spetta al programmatore o al compilatore la suddivisione di un programma in segmenti

Architettura della Segmentazione

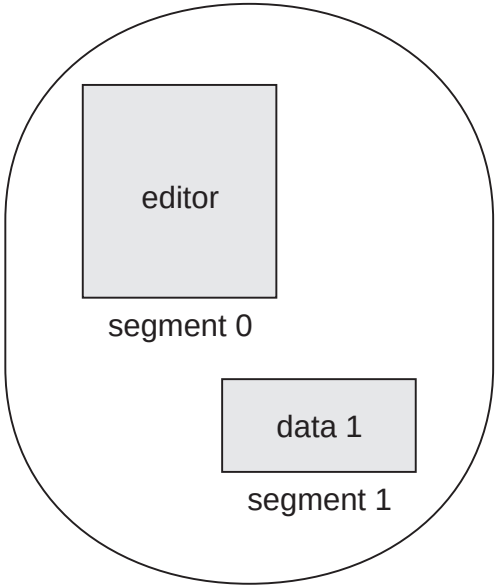
- Un indirizzo logico consiste in un coppia ⟨nome-segmento, offset⟩.
- La *segment table* mappa gli indirizzi bidimensionali dell'utente negli indirizzi fisici unidimensionali. Ogni entry ha
  - *base:* indirizzo fisico di inizio del segmento
  - *limit:* lunghezza del segmento
- Il *Segment-table base register (STBR)* punta all'inizio della tabella dei segmenti
- Il *Segment-table length register (STLR)* indica il numero di segmenti usati dal programma
- Il nome-segmento s è legale solo se $s <$ STLR.

Hardware per la segmentazione



Architettura della Segmentazione (cont.)

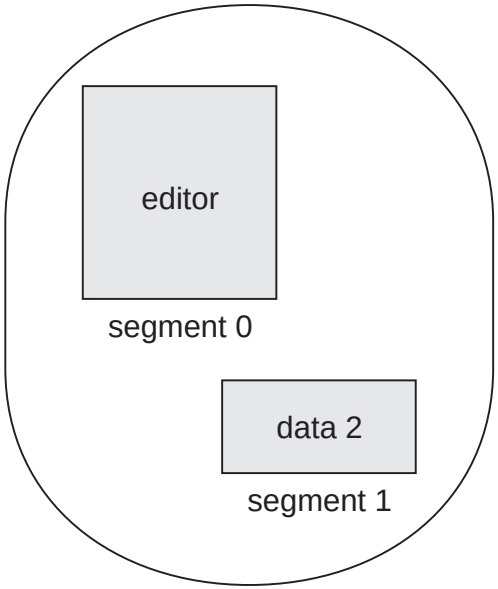
- Rilocazione
  - dinamica, attraverso tabella dei segmenti
- Condivisione
  - interi segmenti possono essere condivisi
- Allocazione
  - gli stessi algoritmi dell'allocazione contigua
  - frammentazione esterna; non c'è frammentazione interna
- Protezione: ad ogni entry nella segment table si associa
  - bit di validità: 0 $\Rightarrow$ segmento illegale
  - privilegi di read/write/execute
- I segmenti possono cambiare di lunghezza durante l'esecuzione (es. lo stack): problema di allocazione dinamica di memoria.



logical memory
process P_1

	limit	base
0	25286	43062
1	4425	68348

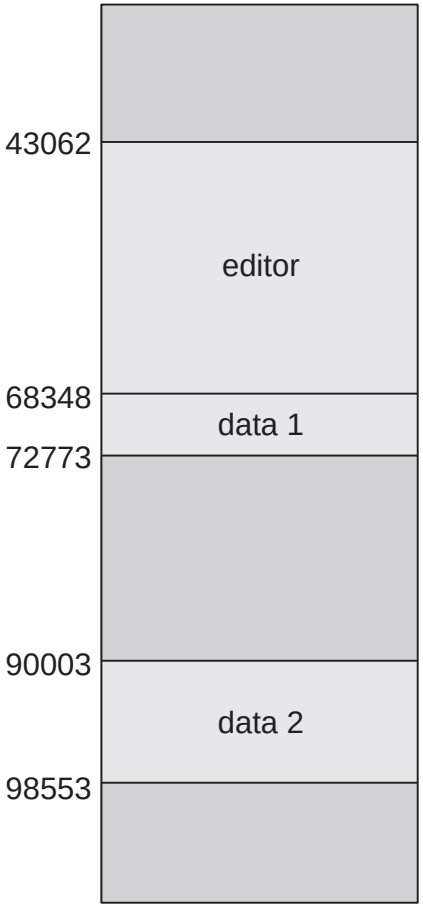
segment table
process P_1



logical memory
process P_2

	limit	base
0	25286	43062
1	8850	90003

segment table
process P_2



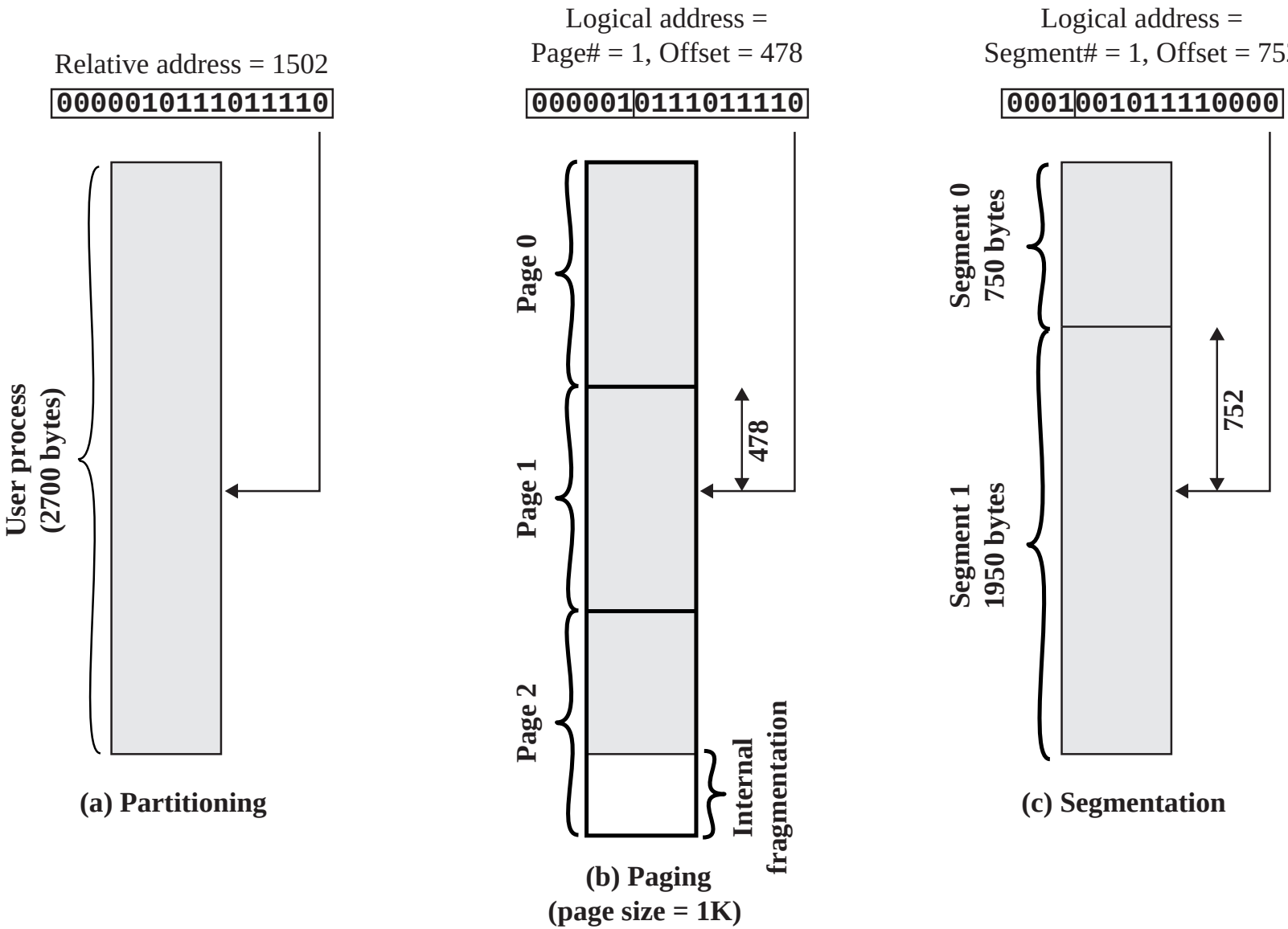
physical memory

Segmentazione vs Paginazione

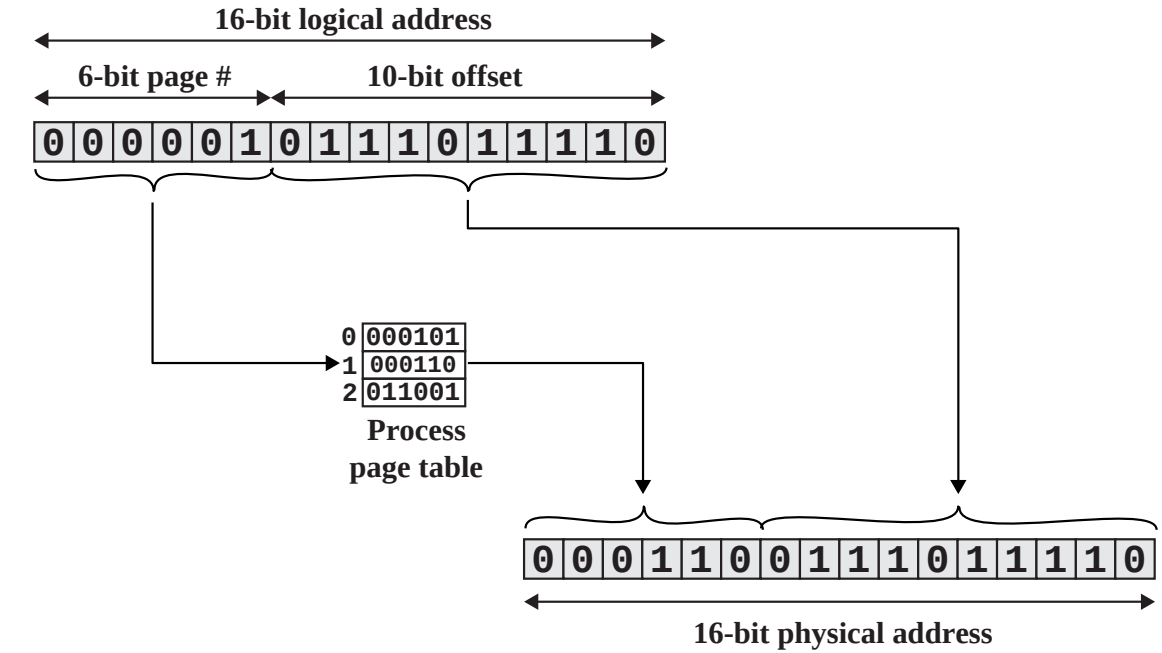
- Paginazione
  - la divisione in pagine automatica.
  - le pagine hanno dimensione fissa
  - le pagine possono contenere informazioni disomogenee (ad es. sia codice sia dati)
  - una pagina ha un indirizzo
  - dimensione tipica della pagina: 1-4 KB

- Segmentazione
  - la divisione in segmenti spetta al programmatore.
  - i segmenti hanno dimensione variabile
  - un segmento contiene informazioni omogenee per tipo di accesso e permessi di condivisione
  - un segmento ha un nome.
  - dimensione tipica di un segmento: 64KB - 1MB

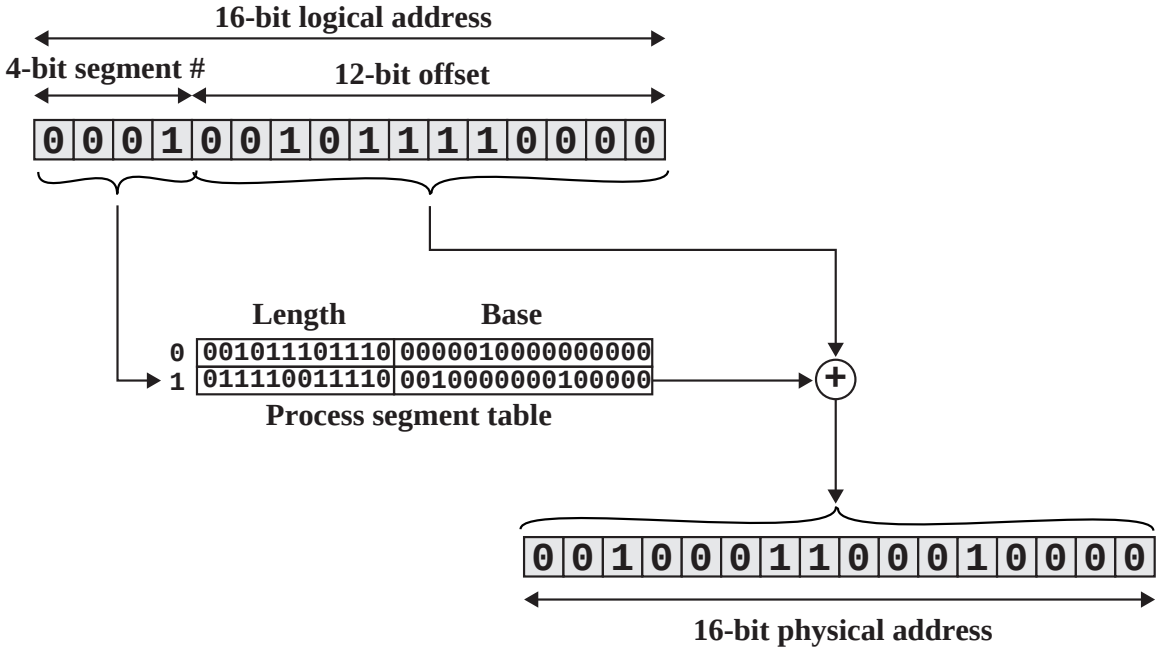
Partizionamento, Paginazione, Segmentazione



- Indirizzi usati nella paginazione
  - Spazio di indirizzi logici = 2^m
  - Dimensione di pagina = 2^n
  - Numero di pagine = 2^{m-n}
  - $m - n$ bit più significativi di un indirizzo logico per il numero di pagina
  - n bit meno significativi per offset



(a) Paging



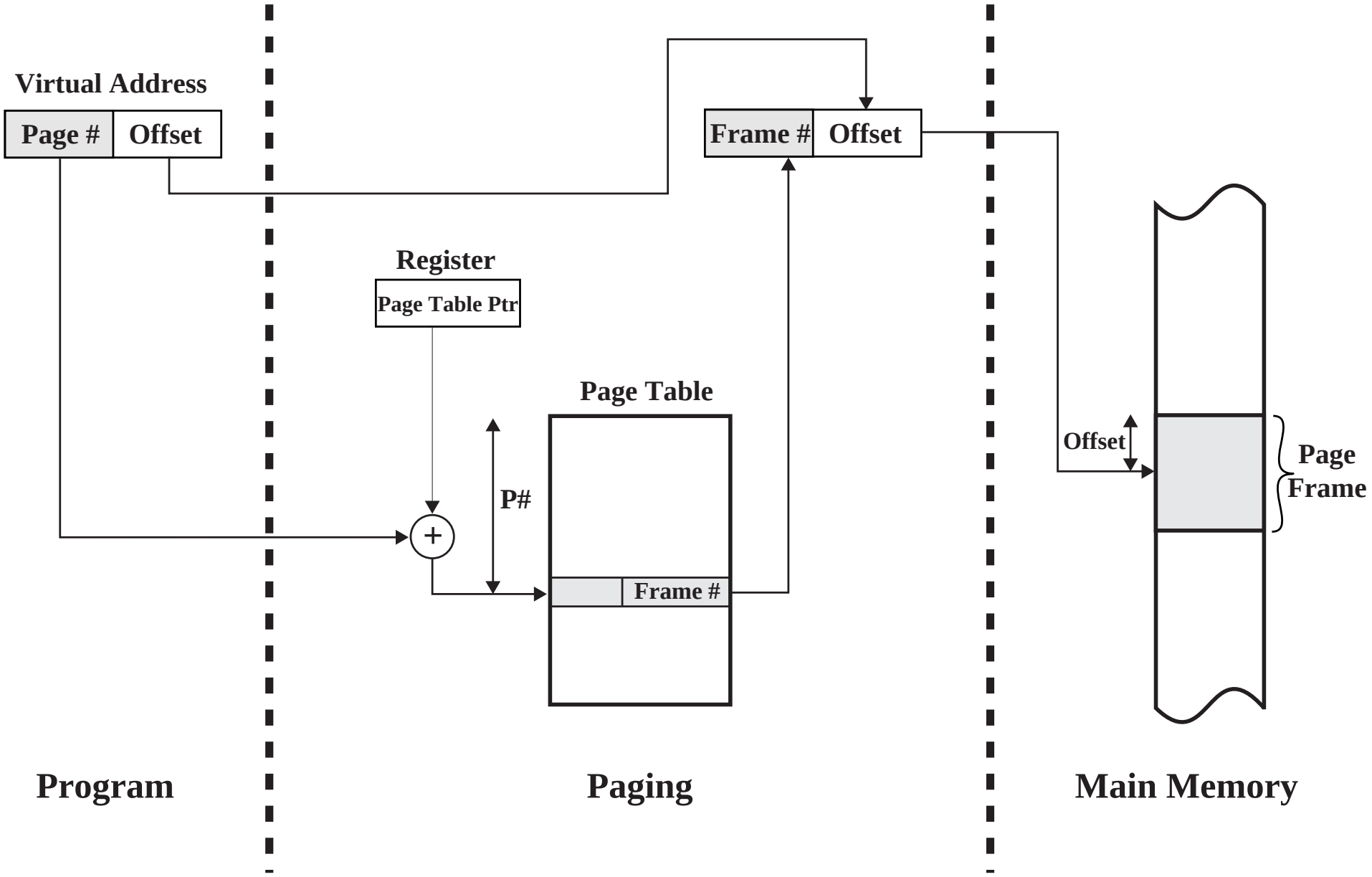
(b) Segmentation

Implementazione e Combinazione di Tecniche

Implementazione della Page Table

- Idealmente, la page table dovrebbe stare in registri veloci della MMU.
  - Costoso al context switch (salvataggio/ripristino di tutta la tabella)
  - Improponibile se il numero delle pagine è elevato. Es: indirizzi virtuali a 32 bit, pagine di 4K (2^{12}): ci sono $2^{20} > 10^6$ entry.
  - Se usiamo 2byte per ogni entry (max RAM = 256M) abbiamo bisogno di 2^{21} = 2M in registri.
- La page table viene tenuta in memoria principale
  - *Page-table base register (PTBR)* punta all'inizio della page table
  - *Page-table length register (PTLR)* indica il numero di entry della page table

Paginazione con page table in memoria



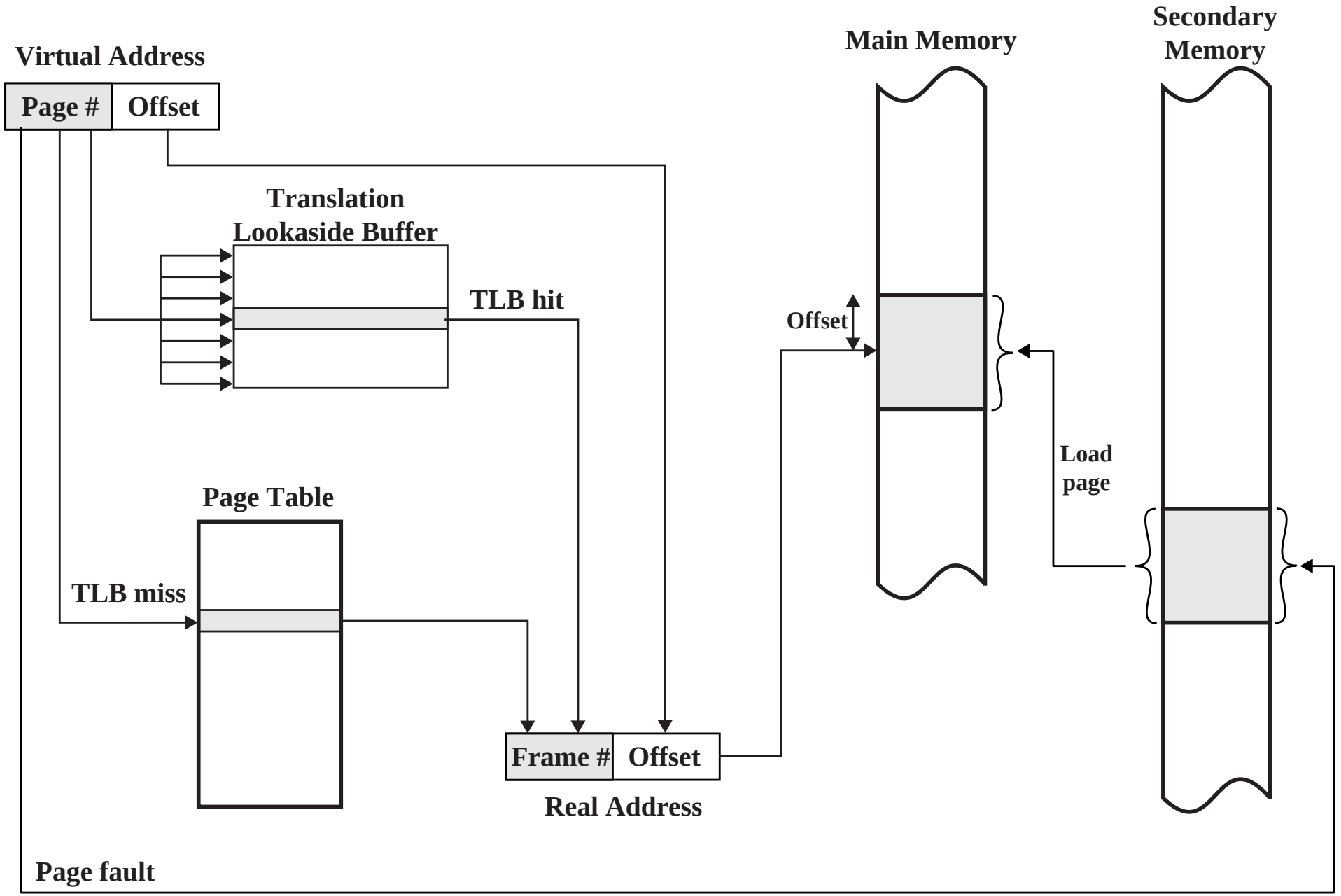
Paginazione con page table in memoria (cont.)

- Rimane comunque un grande consumo di memoria (1 page table per ogni processo). Nell'es. di prima: 100 processi $\Rightarrow$ 200M in page tables (su 256MB RAM complessivi).

- Ogni accesso a dati/istruzioni richiede 2 accessi alla memoria: uno per la page table e uno per i dati/istruzioni $\Rightarrow$ degrado del 100%.

- Il doppio accesso alla memoria si riduce con una cache dedicata per le entry delle page tables: *registri associativi* detti anche *translation look-aside buffer (TLB).*

Registri Associativi (TLB)



Traduzione indirizzo logico (A', A'') con TLB

- Il virtual page number A' viene confrontato con tutte le entry contemporaneamente.

- Se A' è nel TLB (TLB hit), si usa il frame # nel TLB

- Altrimenti, la MMU esegue un normale lookup nelle page table in memoria, e sostituisce una entry della TLB con quella appena trovata

- Il S.O. viene informato solo nel caso di un page fault

Variante: software TLB

I TLB miss vengono gestiti direttamente dal S.O.

- nel caso di una TLB miss, la MMU manda un interrupt al processore (*TLB fault*)

- si attiva una apposita routine del S.O., che gestisce le page table e la TLB esplicitamente

Abbastanza efficiente con TLB suff. grandi (≥ 64 entries)

MMU estremamente semplice $\Rightarrow$ lascia spazio sul chip per ulteriori cache

Molto usato (SPARC, MIPS, Alpha, PowerPC, HP-PA, Itanium. . .)

Tempo effettivo di accesso con TLB

- ϵ = tempo del lookup associativo
- t = tempo della memoria
- α = *Hit ratio*: percentuale dei page # reperiti nel TLB (dipende dalla grandezza del TLB, dalla natura del programma...)

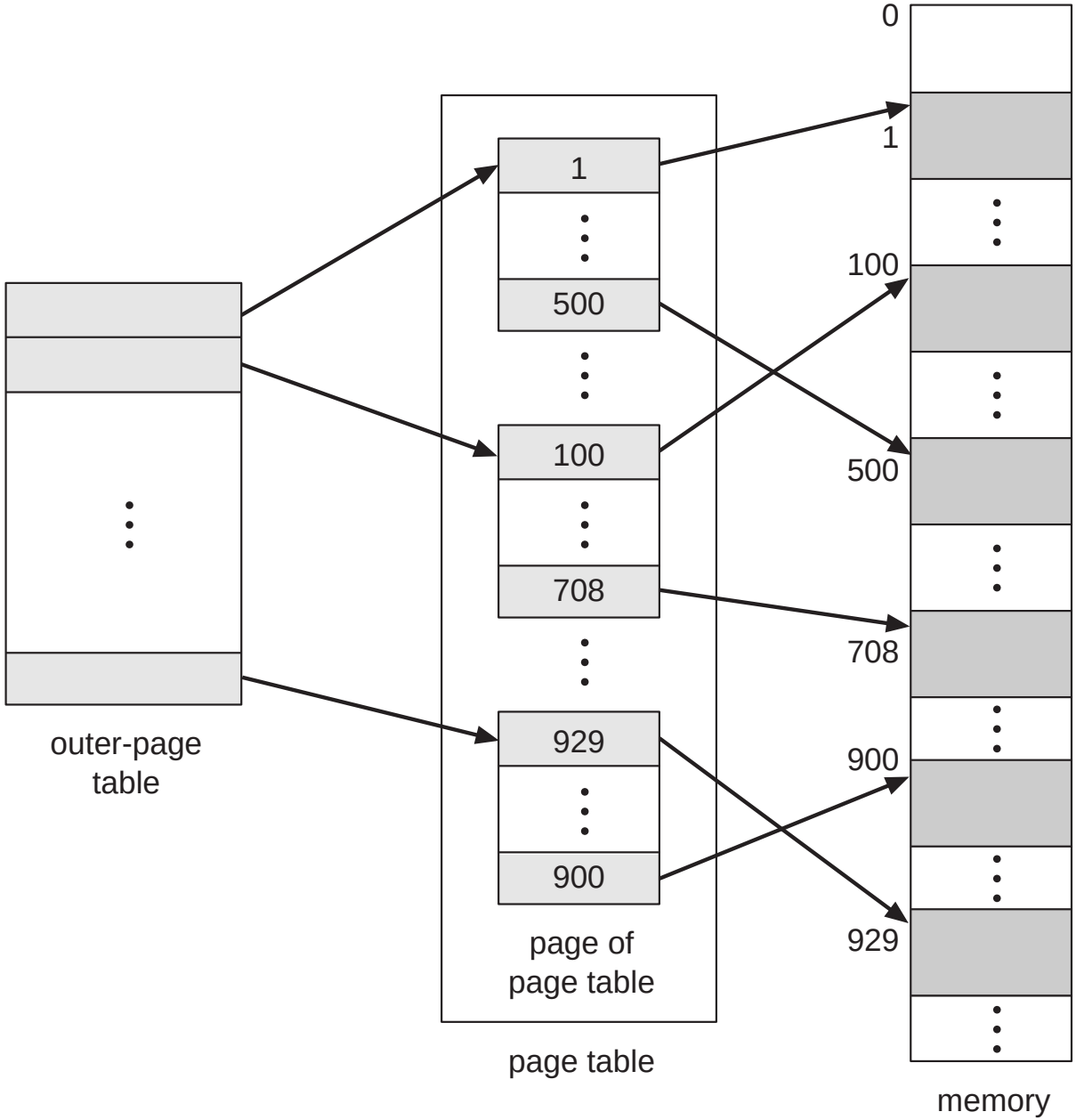
$$EAT = (t + \epsilon)\alpha + (2t + \epsilon)(1 - \alpha) = (2 - \alpha)t + \epsilon$$

- In virtù del *principio di località*, l'hit ratio è solitamente alto
- Con $t = 50ns, \epsilon = 1ns, \alpha = 0.98$ si ha $EAT/t = 1.04$

Paginazione a più livelli

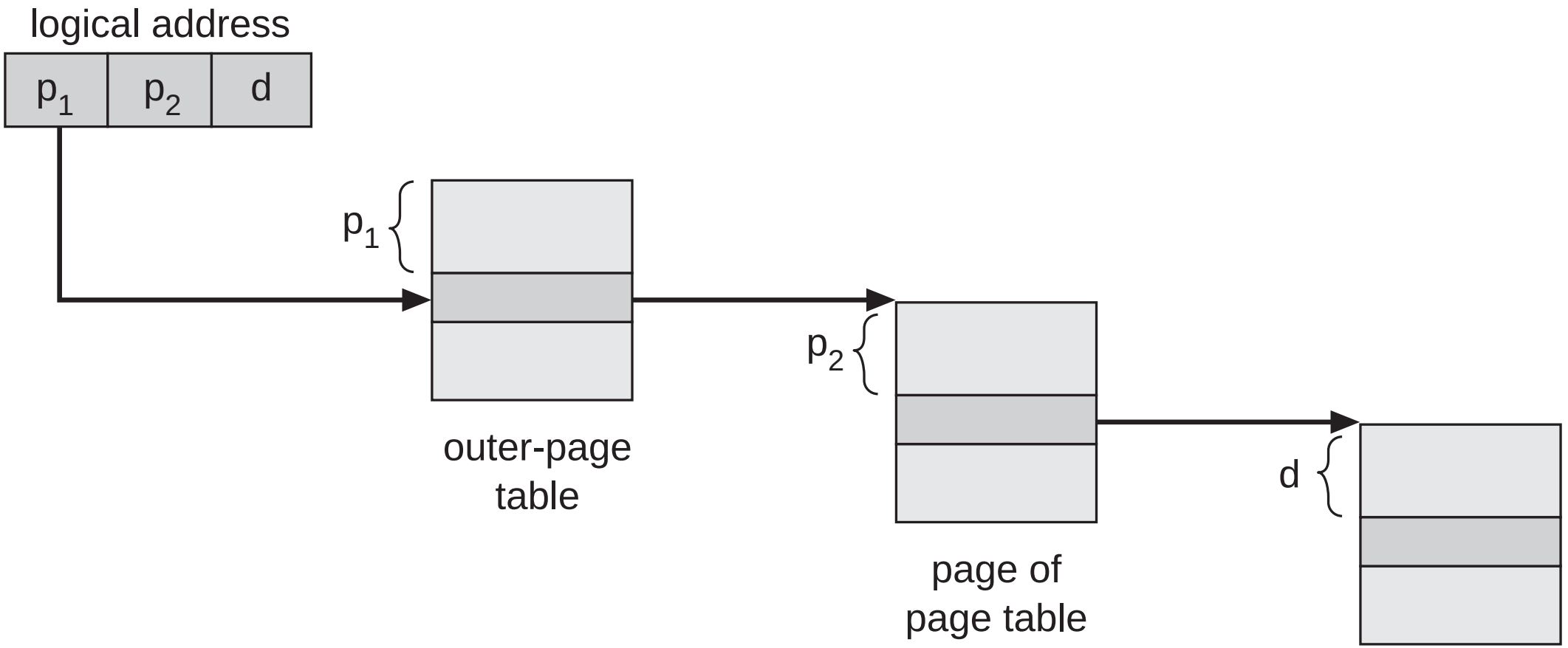
- Per ridurre l'occupazione della page table, si pagina la page table stessa.

- In questo modo solo le pagine effettivamente usate sono allocate in memoria RAM.


0
1
100
500
708
900
929
1
500
100
708
929
outer-page table
page of page table
page table
memory

Esempio di paginazione a due livelli

- Un indirizzo logico (a 32 bit con pagine da 4K) è diviso in
  - un numero di pagina consistente in 20 bit
  - un offset di 12 bit
- La page table è paginata, quindi il numero di pagina è diviso in
  - un *directory number* di 10 bit
  - un *page offset* di 10 bit.


logical address
p1
p2
d
p1
outer-page table
p2
page of page table
d

Performance della paginazione a più livelli

- Dato che ogni livello è memorizzato in RAM, la conversione dell'indirizzo logico in indirizzo fisico può necessitare di 4 accessi alla memoria (in caso di TLB miss: lookup in TLB, 2 accessi alla RAM per la rilocazione, 1 accesso per leggere la casella di memoria)

- Il caching degli indirizzi di pagina permette di ridurre drasticamente l'impatto degli accessi multipli. Ad esempio se in caso di TLB miss servono 5 accessi alla memoria (ad es. 4 livelli di page tabling)

$$EAT = \alpha(t + \epsilon) + (1 - \alpha)(5t + \epsilon) = \epsilon + (5 - 4\alpha)t$$

- Nell'esempio di prima, con un hit rate del 98%: $EAT/t = 1.1$: 10% di degrado

- Schema molto adottato da CPU a 32 bit (IA32 (Pentium), 68000, SPARC a 32 bit, . . .)

Tabella delle pagine invertita

- Una tabella con una entry per ogni *frame*, non per ogni page.
- Ogni entry consiste nel numero della pagina (virtuale) memorizzata in quel frame, con informazioni riguardo il processo che possiede la pagina.
- Diminuisce la memoria necessaria per memorizzare le page table, ma aumenta il tempo di accesso alla tabella.
- Questo schema è usato su diversi RISC a 32 bit (PowerPC), e tutti quelli a 64 bit (UltraSPARC, Alpha, HPPA, . . .), ove una page table occuperebbe petabytes (es: a pagine da 4k: $8 \times 2^{52} = 32$PB per ogni page table)

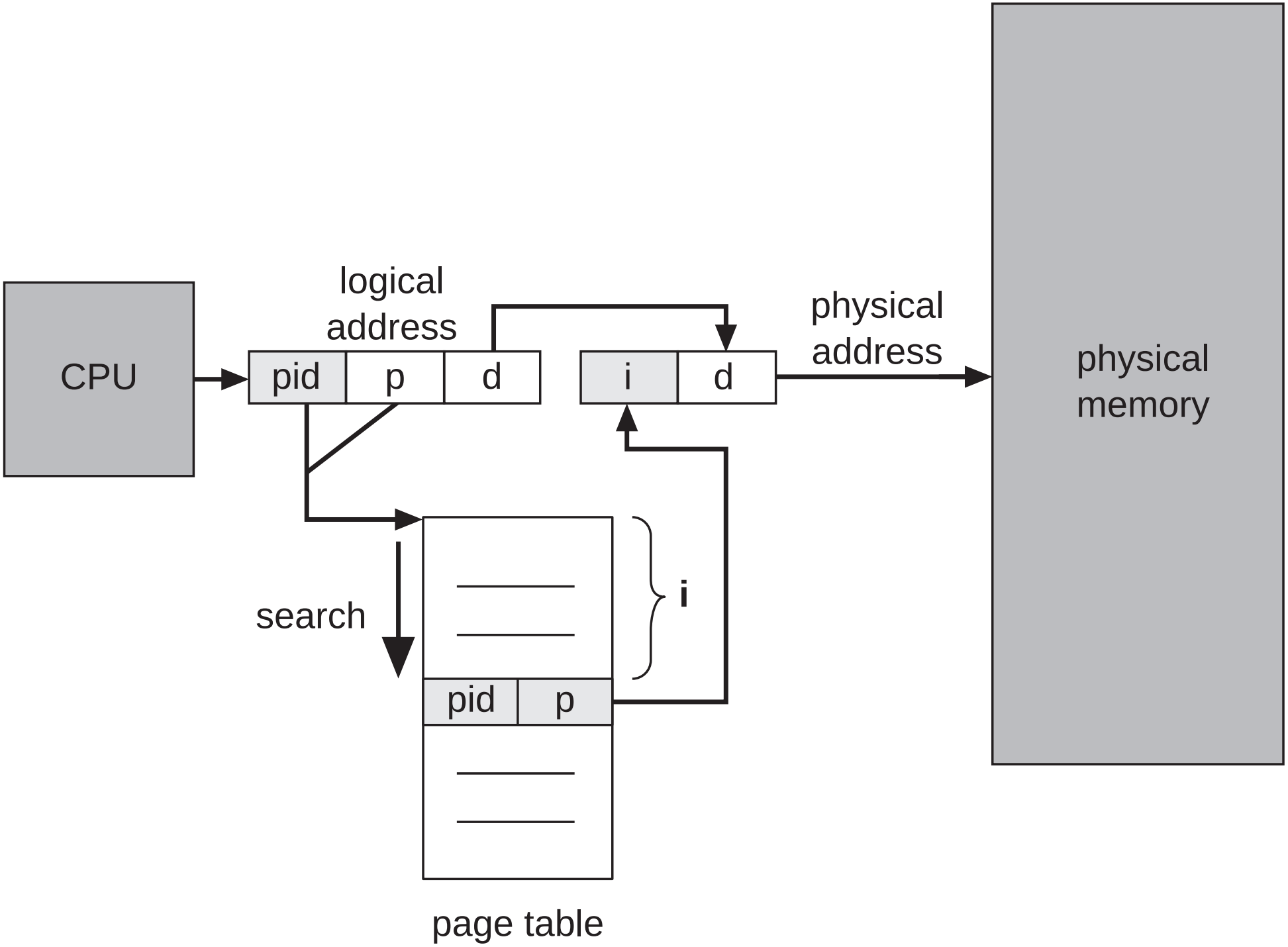
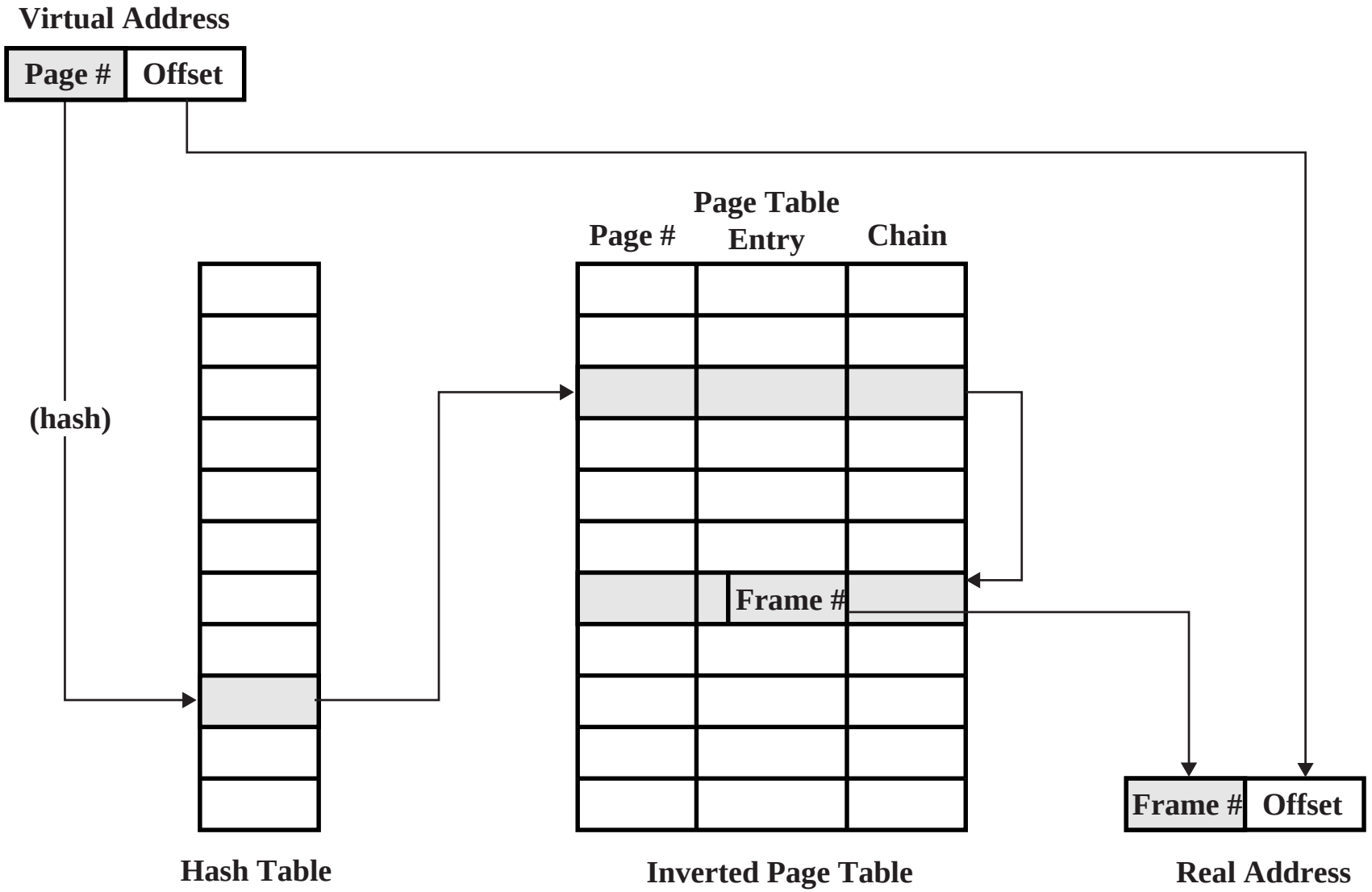

CPU
logical address
pid
p
d
i
d
physical address
physical memory
search
i
pid
p
page table

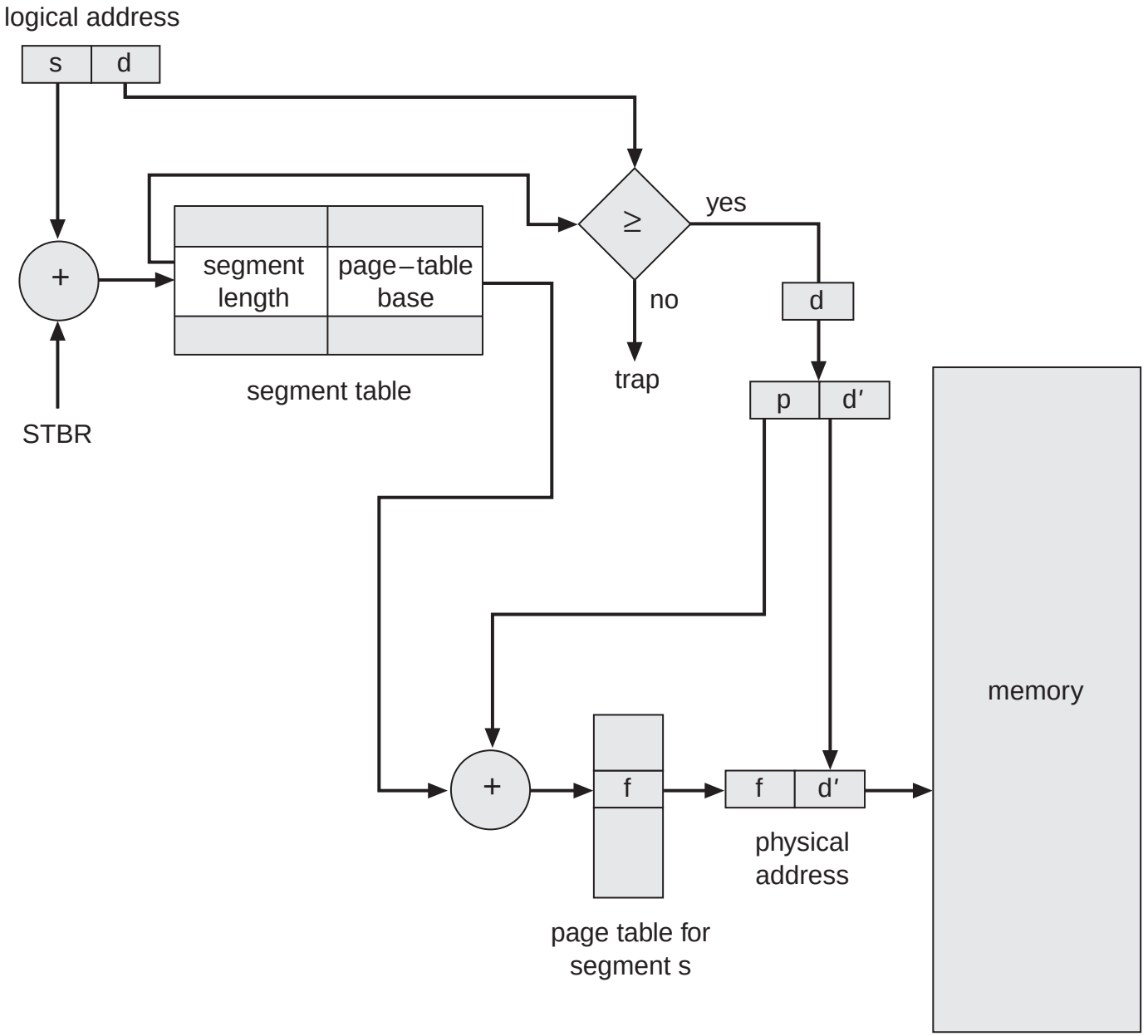
Tabella delle pagine invertita con hashing

Per ridurre i tempi di ricerca nella tabella invertita, si usa una funzione di hash (hash table) per limitare l'accesso a poche entry (1 o 2, solitamente).

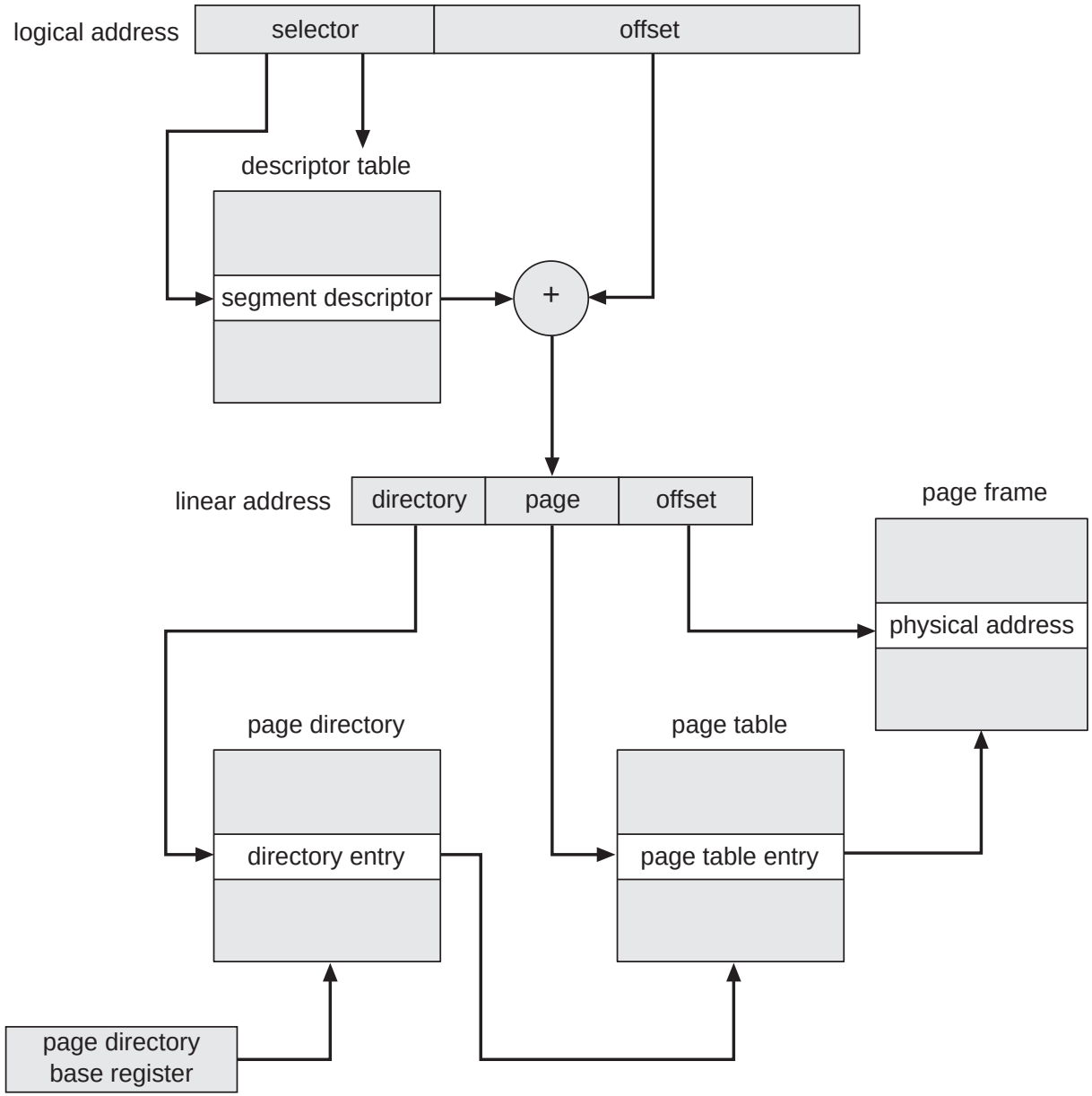


Segmentazione con paginazione: MULTICS

- Il MULTICS ha risolto il problema della frammentazione esterna paginando i segmenti
- Permette di combinare i vantaggi di entrambi gli approcci
- A differenza della pura segmentazione, nella segment table ci sono gli indirizzi base delle page table dei segmenti



Segmentazione con paginazione a 2 livelli: la IA32



Sommario sulle strategie della Gestione della Memoria

- Supporto Hardware: da registri per base-limite a tabelle di mappatura per segmentazione e paginazione

- Performance: maggiore compessità del sistema, maggiore tempo di traduzione. Un TLB può ridurre sensibilmente l'overhead.

- Frammentazione: la multiprogrammazione aumenta l'efficienza temporale. Massimizzare il num. di processi in memoria richiede ridurre spreco di memoria non allocabile. Due tipi di frammentazione.

- Rilocazione: la compattazione è impossibile con binding statico/al load time; serve la rilocazione dinamica.

Sommario sulle strategie della Gestione della Memoria (Cont)

- Condivisione: permette di ridurre lo spreco di memoria e quindi aumentare la multiprogrammazione. Generalmente, richiede paginazione e/o segmentazione. Altamente efficiente, ma complesso da gestire (dipendenze sulle versioni).

- Protezione: modalità di accesso associate a singole sezioni dello spazio del processo, in caso di segmentazione/paginazione. Permette la condivisione e l'identificazione di errori di programmazione.