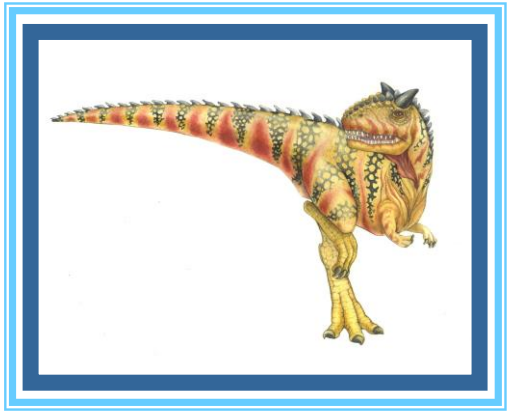
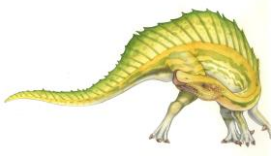
Memoria centrale



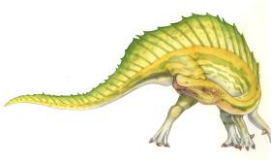
Obiettivi

- Spiegare la differenza fra indirizzi logici e fisici e il ruolo dell'unità di gestione della memoria (MMU) nella traduzione degli indirizzi
- Descrivere la realizzazione della MMU sulla base dell'architettura hardware
- Mostrare come si può produrre frammentazione della memoria e come intervenire per risolvere il problema utilizzando la paginazione
- Appendice: Discutere vari metodi di gestione della memoria, con particolare riferimento alle diverse architetture



Sommario

- Background
- Allocazione contigua della memoria
- Paginazione
- Struttura della tabella delle pagine
- Swapping
- Appendice
  - Segmentazione paginata nelle architetture Intel a 64 bit
  - Paginazione gerarchica nel microprocessore ARMv8

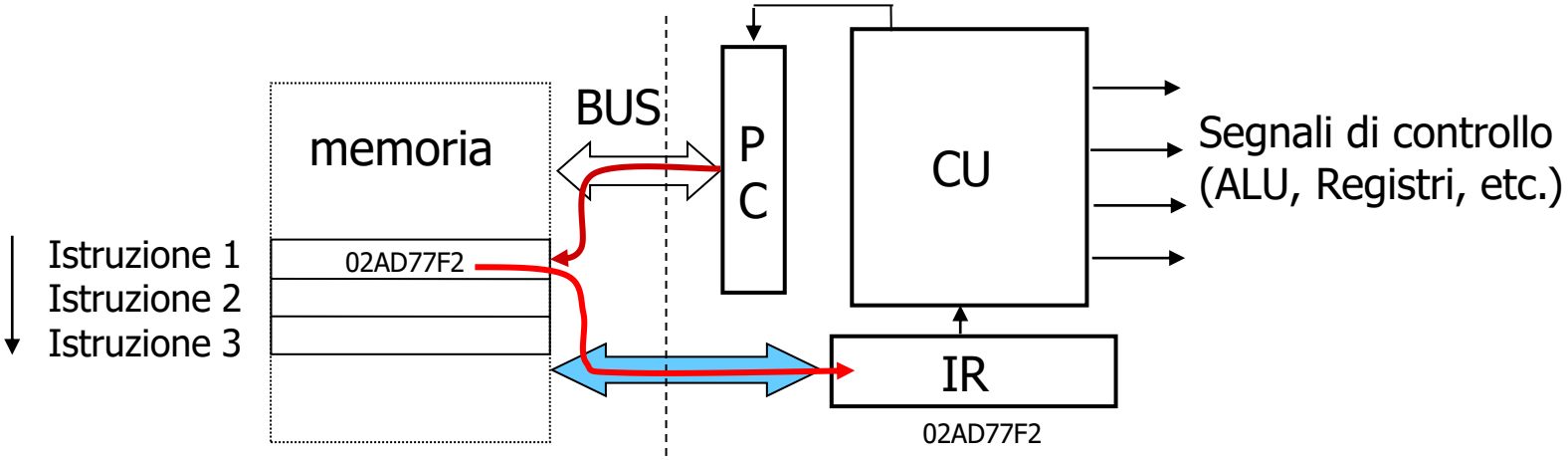


Background – 1

- Lo scheduling della CPU:
  - Migliora l'utilizzo della risorsa CPU
  - Migliora la "reattività" del sistema nei confronti degli utenti (minor tempo di risposta, maggiore throughput)
- Per poter essere eseguiti, i programmi devono essere copiati (almeno parzialmente) in memoria centrale, per "trasformarsi" in processi
  - ⇨Per ottenere l'aumento di prestazioni garantito dallo scheduling della CPU, nella memoria devono essere presenti contemporaneamente più processi
- La scelta di un metodo di gestione della memoria dipende dall'architettura del sistema di calcolo
  - Ogni algoritmo dipende dallo specifico supporto hardware

Background – 2

- Nella fase di *fetch*, la CPU preleva le istruzioni dalla memoria, in base al contenuto del contatore di programma (PC)
  - Le istruzioni possono determinare ulteriori letture e scritture di dati in specifici indirizzi di memoria

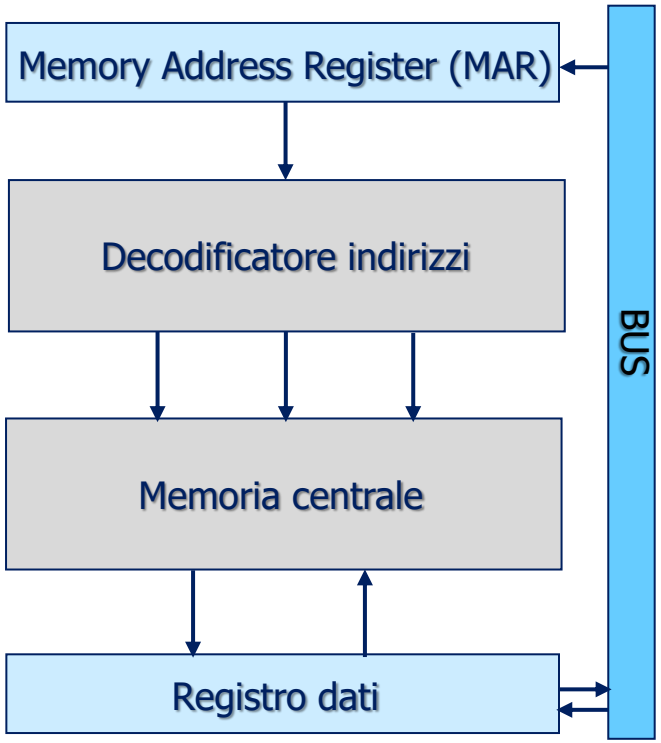


Background – 3

- La memoria "vede" soltanto un flusso di indirizzi e non conosce la modalità con cui sono stati generati o a cosa si riferiscano (istruzioni o dati)

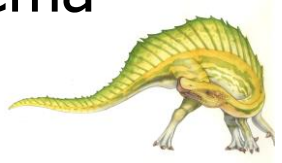
indirizzo +
richiesta di lettura

indirizzo + dati +
richiesta di scrittura



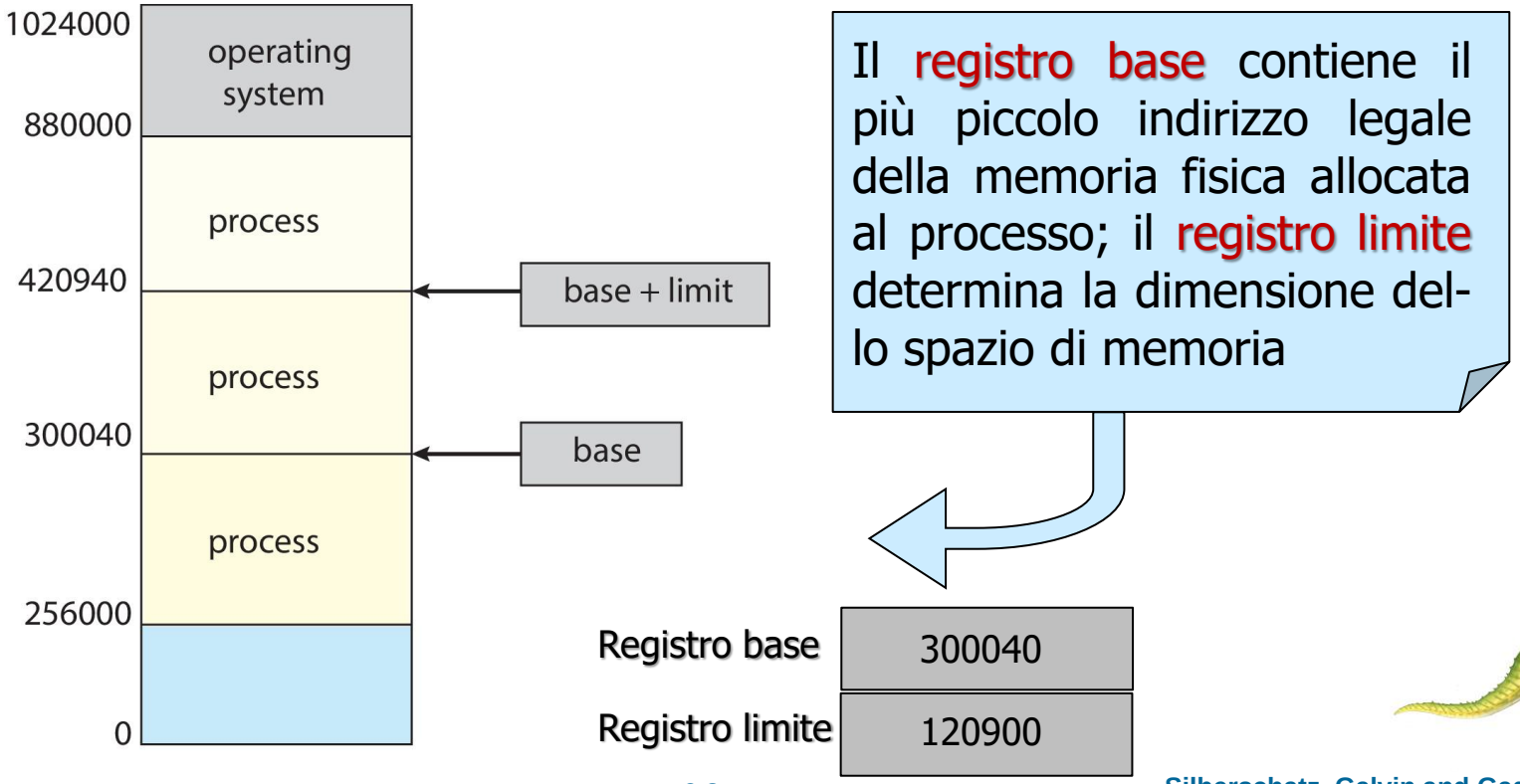
Background – 4

- La memoria principale e i registri sono gli unici dispositivi di memoria cui la CPU può accedere direttamente
- I registri vengono acceduti in un ciclo di clock (o meno)
- Un accesso alla memoria centrale può invece richiedere diversi cicli di clock (occorre "passare" dal bus)
  - ⇨Possibile stallo del processore in attesa dei dati
  - ⇨Soluzione 1: la memoria cache (on–chip) che si situa, nella gerarchia delle memorie, tra la memoria principale e i registri della CPU
  - ⇨Soluzione 2: multithreading hardware
- La protezione della memoria (a livello hardware – il SO non interviene negli accessi della CPU alla RAM) è fondamentale per la corretta operatività del sistema



Esempio: registri base e limite – 1

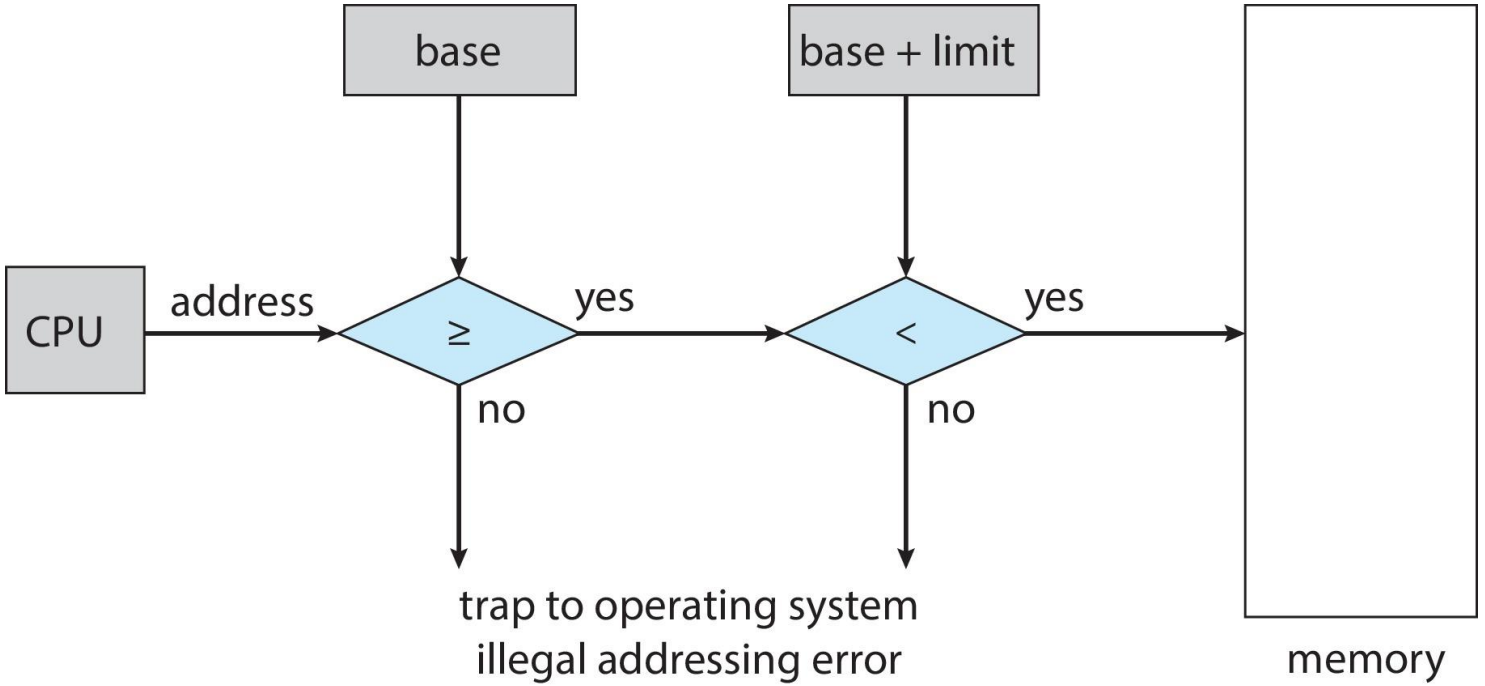
- Ogni processo deve avere uno spazio di memoria separato
- Possibile soluzione: i registri base e limite definiscono lo spazio degli indirizzi fisici di ciascun processo



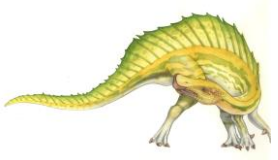
Esempio: registri base e limite – 2

- Per mettere in atto il meccanismo di protezione, la CPU confronta ogni indirizzo generato dal processo utente con i valori contenuti nei due registri
  - indirizzo $\geq$ base
  - indirizzo $<$ base+limite
- Solo il SO può caricare i registri base e limite tramite istruzioni privilegiate
- In modalità utente, un tentativo di accesso ad aree di memoria riservate al SO, o ad altri processi utente, provoca un segnale di eccezione, che restituisce il controllo al SO, e l'emissione di un messaggio di errore

Esempio: registri base e limite – 3

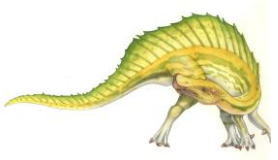


Supporto hardware alla protezione della memoria mediante registri base e limite



Associazione degli indirizzi – 1

- I programmi risiedono su disco sotto forma di file binari eseguibili
- Per essere eseguito, il programma va caricato in memoria e inserito nel contesto di un processo, dove diventa idoneo per l'esecuzione su una CPU disponibile
  - Durante l'esecuzione, si può accedere alle istruzioni del processo ed ai suoi dati residenti in memoria centrale
  - Quando termina, il processo rilascia lo spazio ad esso allocato



Associazione degli indirizzi – 2

- Prima di essere eseguiti, i programmi utente passano attraverso stadi diversi, con diverse rappresentazioni degli indirizzi
  - Gli indirizzi nel programma sorgente sono simbolici
    - e.g. "indice", un nome di variabile
  - Il compilatore associa gli indirizzi simbolici a indirizzi rilocabili (per esempio, calcolati a partire dalla prima istruzione del programma)
    - e.g. "14 byte dall'inizio del modulo corrente"
  - Il linker o il loader fa corrispondere gli indirizzi rilocabili a indirizzi assoluti (nella memoria fisica)
    - e.g. 74014
- Ogni associazione stabilisce una corrispondenza fra spazi di indirizzi (mappa uno spazio di indirizzi in un altro)

Associazione degli indirizzi – 3

- L'associazione – *binding* – di istruzioni e dati a indirizzi di memoria può avvenire durante la fase di…
  - Compilazione: se la posizione in memoria del processo è nota a priori, può essere generato codice assoluto; se la locazione iniziale cambia, è necessario ricompilare il codice (esempio: programmi MS–DOS nel formato .COM)
  - Caricamento: se la posizione in memoria non è nota in fase di compilazione, è necessario generare codice rilocabile; il compilatore genera indirizzi relativi che vengono convertiti in indirizzi assoluti dal loader; se l'indirizzo iniziale cambia, il codice deve essere ricaricato in memoria
    - In altre parole… durante la compilazione, vengono generati indirizzi che fanno riferimento ad un indirizzo base non specificato, e che verrà poi fissato dal loader durante il caricamento

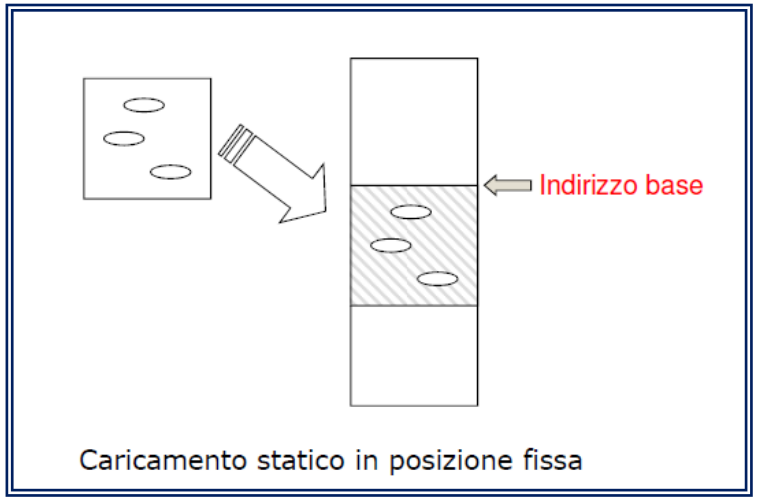
Associazione degli indirizzi – 4

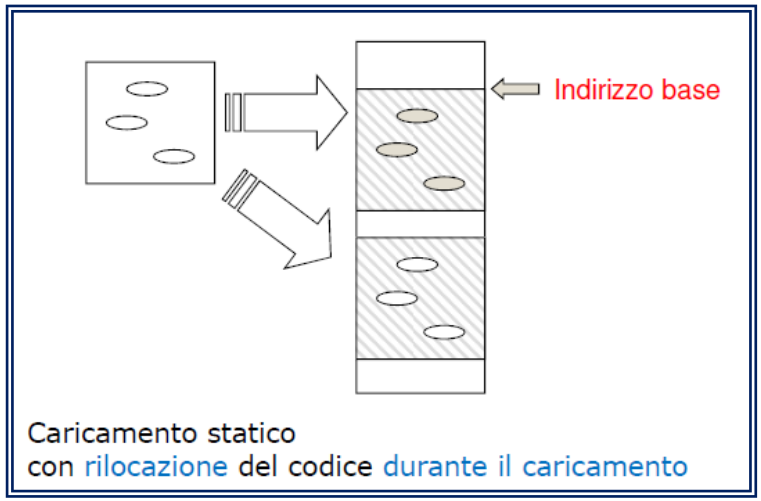
- Esecuzione: se il processo può essere spostato *a run–time* da un segmento di memoria all'altro, il binding viene rimandato fino al momento dell'esecuzione – codice dinamicamente rilocabile
  - Contiene solo riferimenti relativi a se stesso
  - La locazione finale di un riferimento ad un indirizzo di memoria non è determinata finché non si compie effettivamente il riferimento
  - Necessita di un opportuno supporto hardware per il mapping degli indirizzi (ad esempio attraverso registri base e limite o tabella delle pagine)
- La maggior parte dei SO general purpose impiega l'associazione a run–time

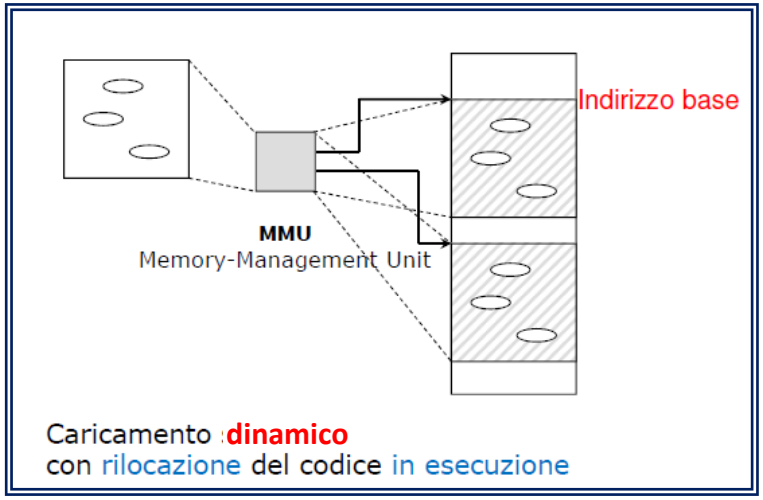
Associazione degli indirizzi – 5

- La CPU genera indirizzi logici, ma l'unità di memoria "vede" indirizzi fisici
- Il binding mappa indirizzi logici in indirizzi fisici
  - Binding statico (a tempo di compilazione o di caricamento)
    - ⇨indirizzi logici = indirizzi fisici
  - Binding dinamico (a run–time)
    - ⇨indirizzi logici (o virtuali) ≠ indirizzi fisici
    - ⇨Possibilità di spostare processi in memoria all'atto dell'esecuzione
    - ⇨Supporto fondamentale a memoria virtuale e swapping

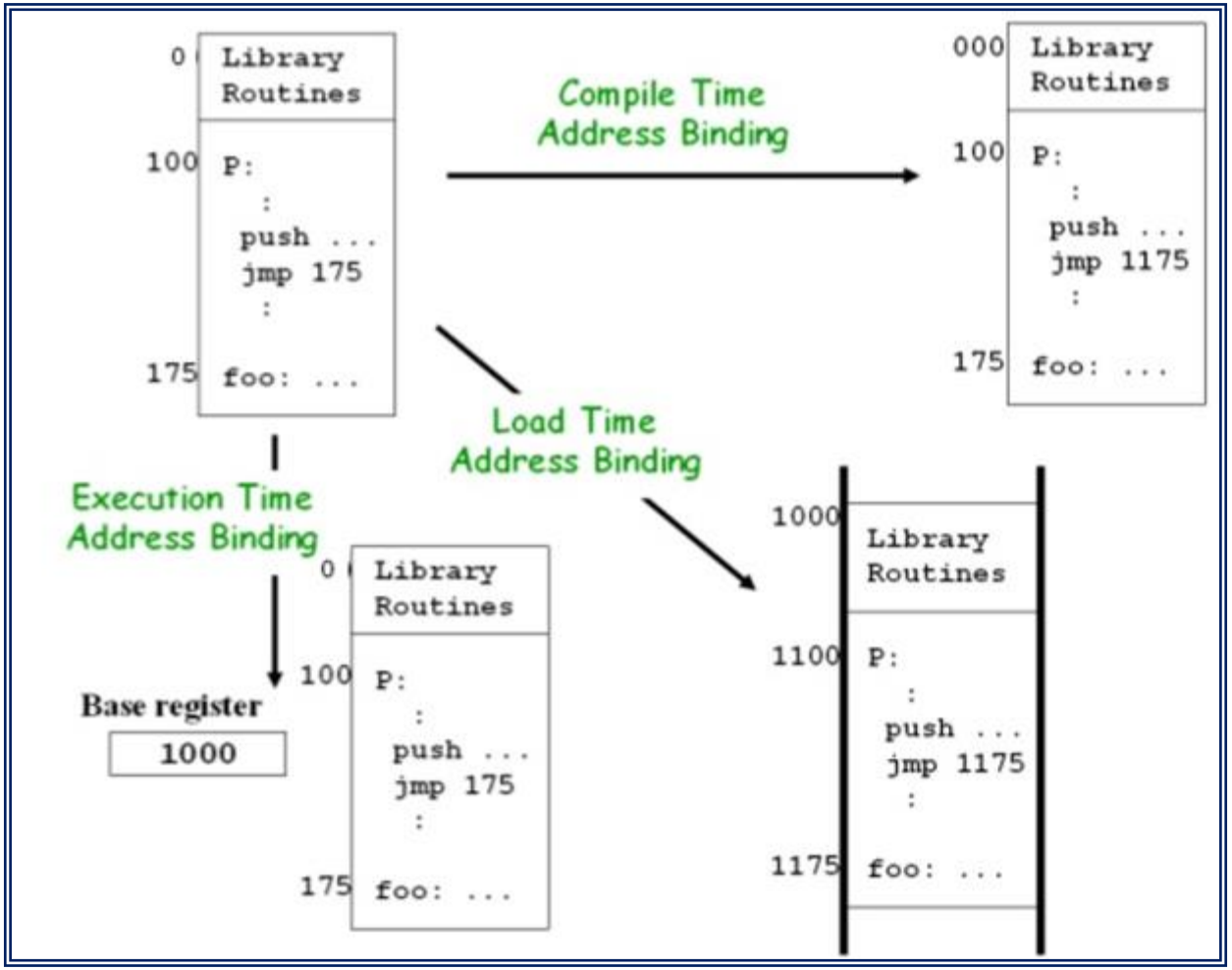
Associazione degli indirizzi – 6



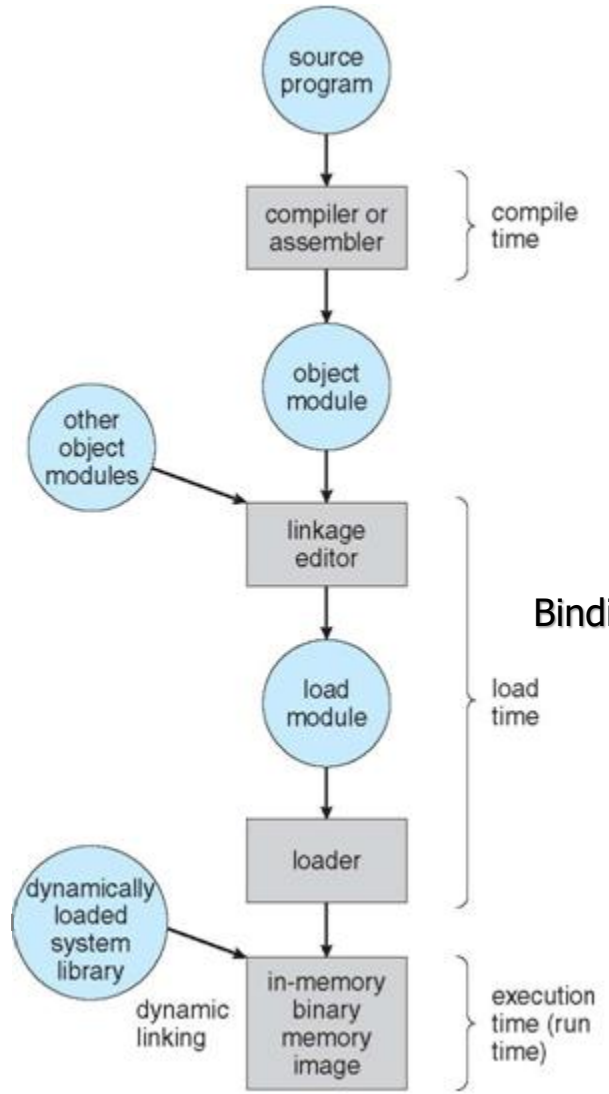




Associazione degli indirizzi – 7



Elaborazione multistep dei programmi utente



Indirizzi **simbolici**, ad esempio *count*

Indirizzi **rilocabili**, ad esempio "*4 byte a partire dall'inizio di questo modulo*"

Binding statico

Indirizzi **assoluti**, ad esempio *26800*

Caricamento dinamico: i sottoprogrammi vengono caricati in memoria quando vengono richiamati

Collegamento dinamico: i sottoprogrammi vengono collegati quando vengono richiamati

Binding dinamico

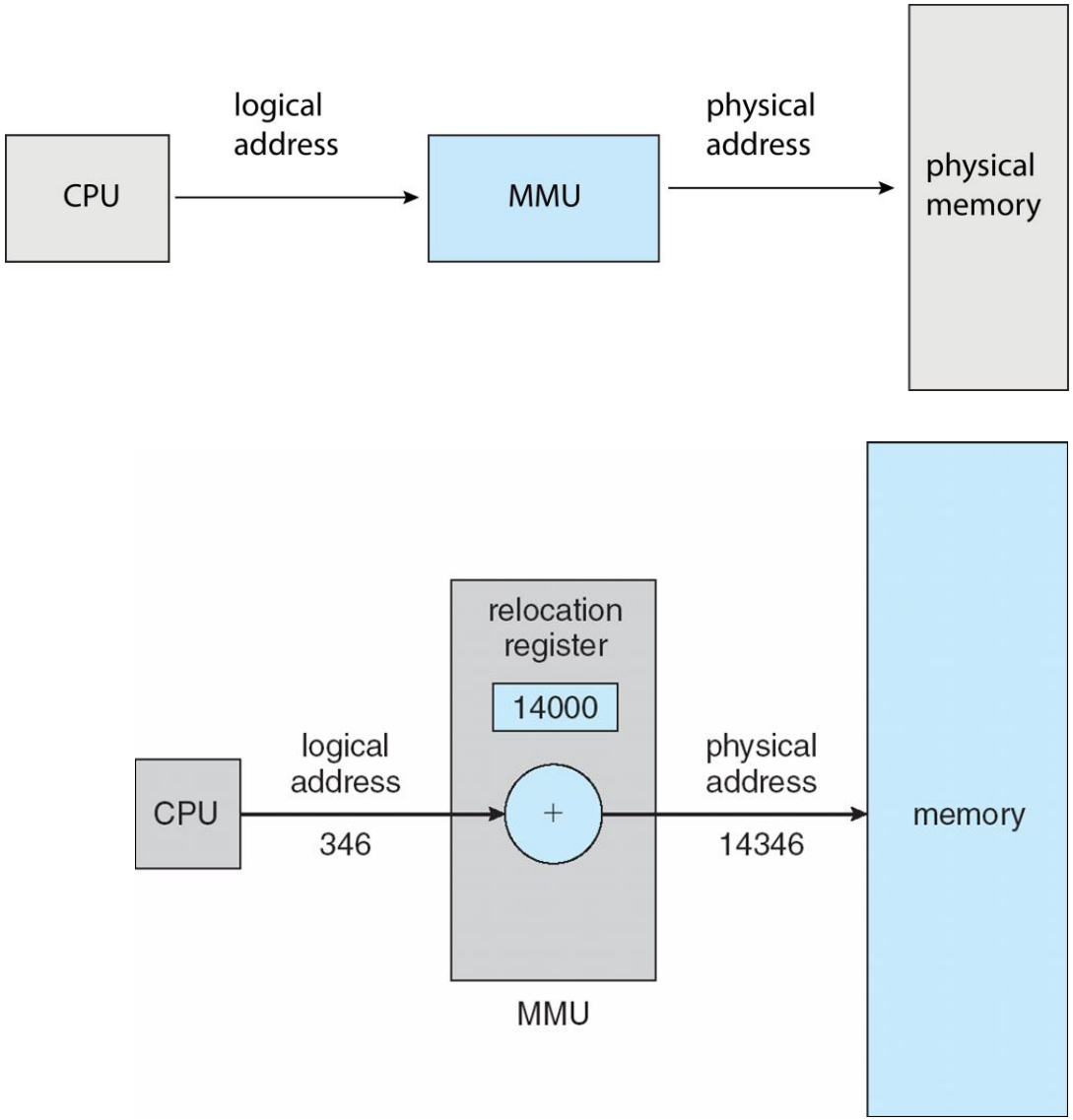
Spazi di indirizzi logici e fisici

- Il concetto di uno *spazio di indirizzi logici* nettamente distinto dallo *spazio degli indirizzi fisici* è centrale per la gestione della memoria
  - Indirizzi logici — generati dalla CPU, altrimenti chiamati indirizzi virtuali
  - Indirizzi fisici — indirizzi contenuti nel *memory address register* (MAR) e utilizzati dall'unità di memoria
- Gli indirizzi logici corrispondono agli indirizzi fisici negli schemi di binding in fase di compilazione e di caricamento, mentre differiscono per il binding in fase di esecuzione
- Lo spazio degli indirizzi logici è costituito dall'insieme di tutti gli indirizzi logici generati da un programma; lo spazio degli indirizzi fisici è l'insieme di tutti gli indirizzi fisici "visti dalla memoria" per quel programma

Memory Management Unit (MMU)

- Dispositivo hardware che mappa indirizzi virtuali su indirizzi fisici a run–time
- Possibili diverse soluzioni per effettuare il mapping
- La più semplice: il valore contenuto nel registro di rilocazione (il registro base) viene sommato ad ogni indirizzo generato dal processo utente nel momento stesso in cui l'indirizzo viene inviato alla memoria
- Il programma utente "ragiona" in termini di indirizzi virtuali, né mai è (deve essere) conscio del loro mapping fisico
  - Il binding a run–time si effettua quando si genera il riferimento ad una particolare posizione di memoria

Rilocazione dinamica tramite MMU



Binding a run–time effettuato tramite registro di rilocazione

Caricamento dinamico

- L'intero programma non deve risiedere in memoria per essere eseguito
- Le routine risiedono su disco in un formato rilocabile e non vengono caricate fino a quando non vengono richiamate
- Miglior impiego della memoria: routine non utilizzate non vengono mai caricate
- Utile quando si richiedono grandi quantità di codice per gestire situazioni che avvengono raramente (condizioni di eccezione, errori)
- Al SO non è richiesto alcun supporto speciale
  - Il caricamento dinamico viene implementato mediante software opportunamente codificato (modulare)
  - Tuttavia, il SO fornisce librerie di procedure che realizzano il caricamento dinamico

Link dinamico delle librerie – 1

- Link statico – librerie di sistema e codice del programma combinati dal caricatore nell'immagine binaria
- Link dinamico – il link viene rinviato fino al momento dell'esecuzione
- Particolarmente utile per librerie di sistema e librerie di procedure del linguaggio (librerie condivise, .so e .dll)
  - Altrimenti: ogni programma dovrebbe disporre all'interno della propria immagine eseguibile di una copia della libreria del linguaggio (o almeno delle routine utilizzate)
- Piccole porzioni di codice, dette *stub*, vengono impiegate per localizzare la routine appropriata nella libreria residente in memoria
- Lo stub rimpiazza se stesso con l'indirizzo della routine e la esegue

Link dinamico delle librerie – 2

- Il SO deve verificare se la routine si trova già nello spazio degli indirizzi del processo o, eventualmente, deve renderla accessibile se è presente in memoria nello spazio di indirizzi di un altro processo
- Modalità operativa a librerie condivise
- Supporto del sistema operativo per superare le barriere di protezione fra processi ed accedere a (aree di memoria "comuni" contenenti) segmenti di codice condiviso

Allocazione contigua

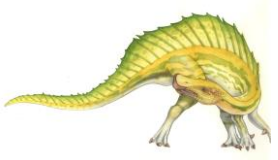
- La memoria principale è una risorsa limitata, che deve essere allocata in modo efficiente
- Deve contenere sia i processi utente che di sistema; viene pertanto suddivisa in due parti:
  - La parte residente del SO è generalmente memorizzata nella memoria alta (nelle architetture moderne), insieme al *vettore degli interrupt*
  - I processi utente sono memorizzati nella memoria bassa
- Allocazione contigua
  - a partizione singola
    - La parte di memoria disponibile per l'allocazione dei processi non è partizionata
    - Un solo processo alla volta può essere allocato in memoria: non c'è multiprogrammazione (MS–DOS)
  - a partizioni multiple

Partizioni multiple – 1

- Partizioni fisse (MFT – *Multiprogramming with a Fixed number of Tasks*): la dimensione di ogni partizione è fissata a priori
  - Quando un processo viene schedulato, il SO cerca una partizione libera di dimensioni sufficienti ad accoglierlo
  - Esempio: IBM OS/360 (1966)
  - Problemi
    - Frammentazione interna: sottoutilizzo della partizione
    - Grado di multiprogrammazione limitato dal numero di partizioni
    - Dimensione massima dei processi limitata dalla dimensione della partizione più estesa

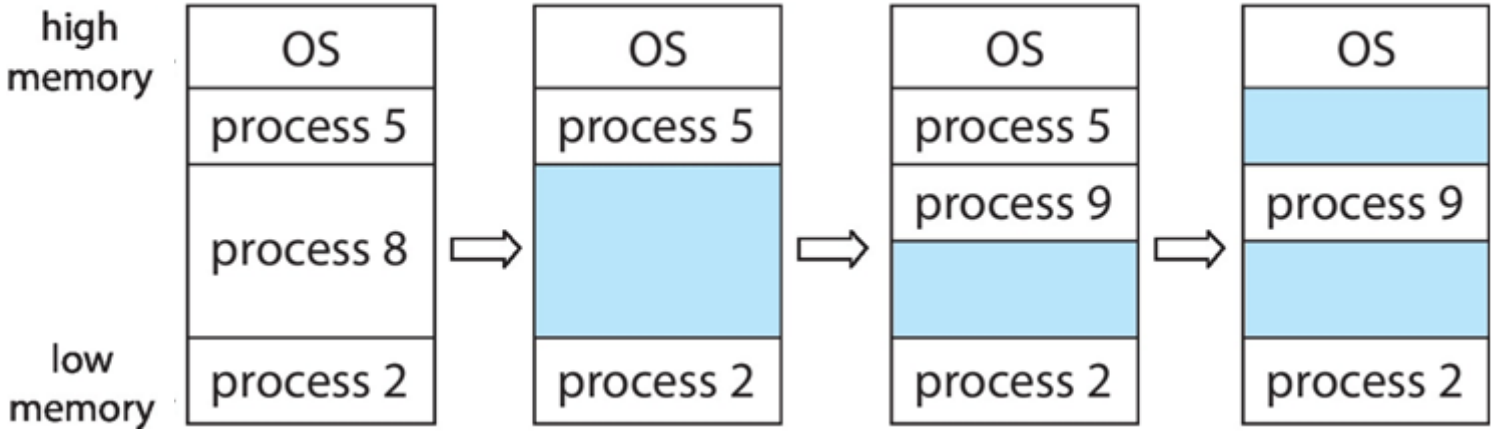
Partizioni multiple – 2

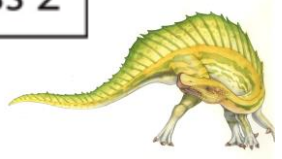
- Partizioni variabili (MVT – *Multiprogramming with a Variable number of Tasks*): ogni partizione è allocata dinamicamente in base alla dimensione del processo da allocare
  - Utilizzato in ambienti batch o time–sharing con gestione della memoria tramite segmentazione semplice
  - Vantaggi (vs MFT)
    - Si elimina la frammentazione interna: ogni partizione è dell'esatta dimensione del processo
    - Il grado di multiprogrammazione è variabile
    - La dimensione massima dei processi è limitata dalla disponibilità di spazio fisico
  - Problemi
    - Scelta dell'area da allocare
    - Frammentazione esterna



Partizioni multiple – 3

- Un buco – *hole* – è un blocco di memoria disponibile; nella memoria sono sparsi buchi di varie dimensioni
- Quando viene caricato un nuovo processo, gli viene allocato un buco grande abbastanza da contenerlo
- Quando un processo termina, libera la memoria allocata; partizioni libere contigue vengono "riassemblate"
- Il SO conserva informazioni su:
  - Partizioni allocate
  - Partizioni libere

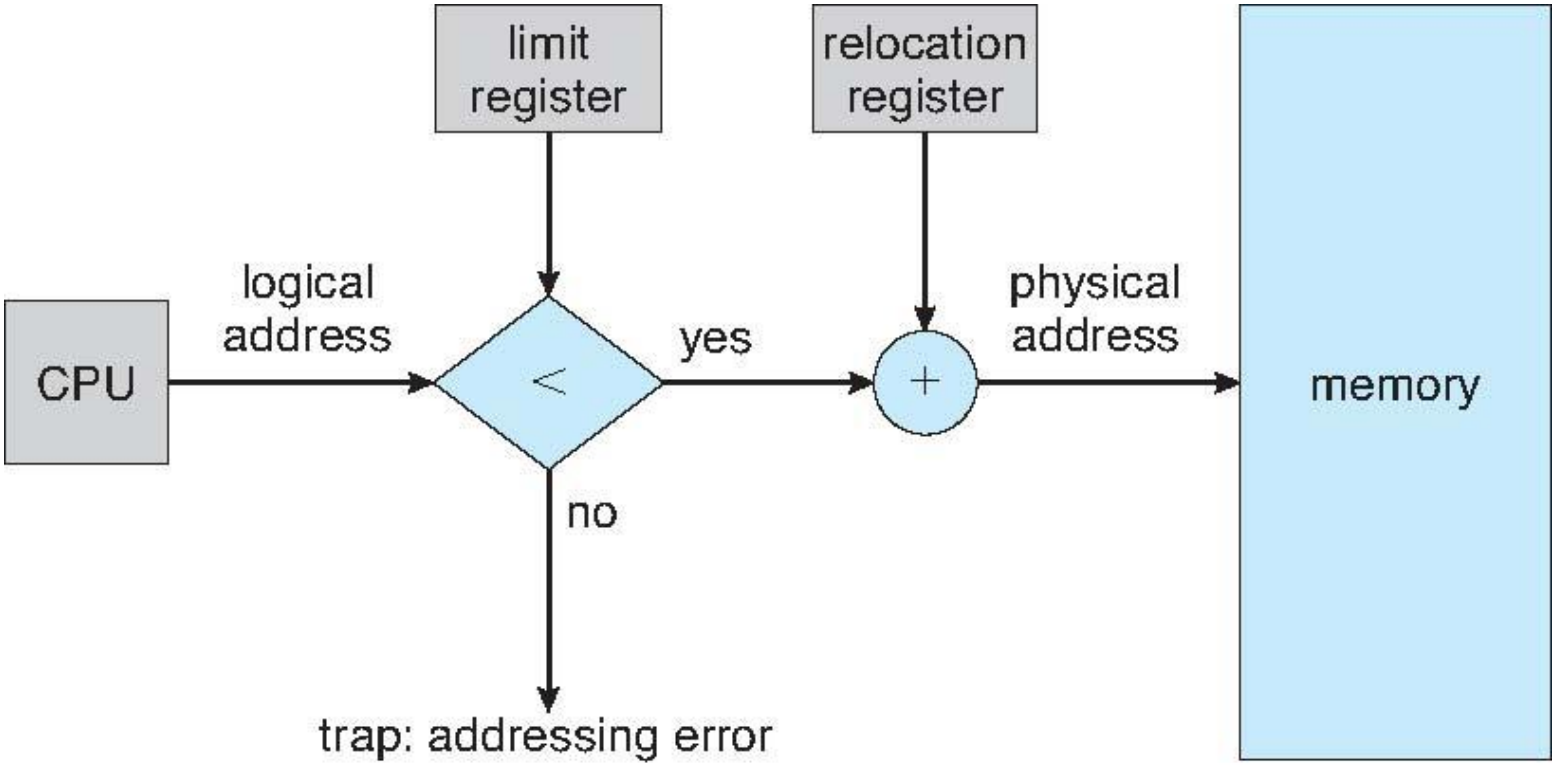




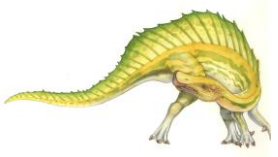
Allocazione contigua (cont.)

- In caso di partizioni multiple variabili, ad ogni processo viene associata un'area di memoria distinta
- I registri di rilocazione e limite vengono utilizzati per proteggere reciprocamente i processi utente e per prevenirne eventuali accessi a codice e dati di sistema
  - Il registro di rilocazione contiene il valore del più piccolo indirizzo fisico dell'area di memoria allocata ad un processo
  - Il registro limite definisce l'intervallo di variabilità degli indirizzi logici (ciascun indirizzo logico deve essere minore del valore contenuto nel registro)
  - La MMU mappa dinamicamente gli indirizzi logici negli indirizzi fisici
  - Il codice del kernel può essere in parte transiente ed il SO può cambiare dinamicamente le proprie dimensioni

Allocazione contigua (cont.)



Registro di rilocazione e registro limite

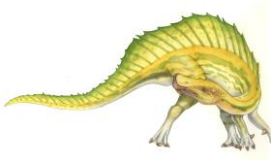


Come allocare memoria dinamicamente?

- In ogni momento è presente un insieme di buchi di diverse dimensioni sparsi per la memoria
- Come soddisfare una richiesta di dimensione n a partire da un insieme di buchi?
  - First–fit: Viene allocato il *primo* buco grande abbastanza
  - Best–fit: Viene allocato il buco *più piccolo* capace di contenere il processo; è necessario scandire tutta la lista dei buchi (se non è ordinata)
    - Si produce il più piccolo buco residuo
  - Worst–fit: Viene allocato il buco *più grande*; è ancora necessario ricercare in tutta la lista
    - Si produce il più grande buco residuo
- First–fit e Best–fit sono migliori di Worst–fit in termini di velocità e di impiego di memoria, rispettivamente

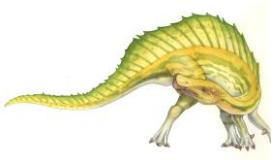
Frammentazione

- Frammentazione interna — La memoria allocata può essere leggermente maggiore della memoria effettivamente richiesta (pochi byte di differenza); la differenza di dimensioni è memoria interna ad una partizione che non viene impiegata completamente
- Frammentazione esterna — È disponibile lo spazio necessario a soddisfare una richiesta, ma non è contiguo
- Si può ridurre la frammentazione esterna con la compattazione
  - Si spostano i contenuti della memoria per avere tutta la memoria libera contigua a formare un grande blocco
  - La compattazione è possibile solo con la rilocazione dinamica perché viene effettuata a run–time
  - I processi con I/O pendenti non possono essere spostati o si deve garantire che l'I/O avvenga solo nello spazio kernel



Frammentazione esterna

- La gravità del problema della frammentazione esterna per allocazione contigua dipende dalla quantità totale di memoria e dalla dimensione media dei processi
- **Regola del 50%:** con First–fit, per n blocchi assegnati, $0.5n$ blocchi possono andare "persi" per frammentazione
  - Un terzo della memoria diventa inutilizzabile!
- Anche la memoria di massa, se allocata in modo contiguo, soffre del problema della frammentazione



Esempio

- Si supponga che la lista delle partizioni libere di memoria, mantenuta in ordine di indirizzo, consista di cinque nodi N_1, N_2, N_3, N_4, N_5, le cui dimensioni e gli indirizzi sono riportati nella tabella seguente:

Nodi	Dimensioni	Indirizzo
N_1	100K	1K
N_2	500K	610K
N_3	200K	1200K
N_4	300K	1520K
N_5	600K	2000K

Dovendo inserire P_1, P_2, P_3, P_4, rispettivamente di 212K, 417K, 112K, 426K (in questa successione), come e dove saranno memorizzati usando First–Fit (facendo partire la ricerca dalla partizione successiva a quella a cui si era arrivati, quand'anche fosse rimasto nella stessa un buco di dimensione sufficiente, ed iniziando dal nodo N_1) e Best–Fit? Calcolare la frammentazione in entrambi i casi.

Esempio (cont.)

- Soluzione

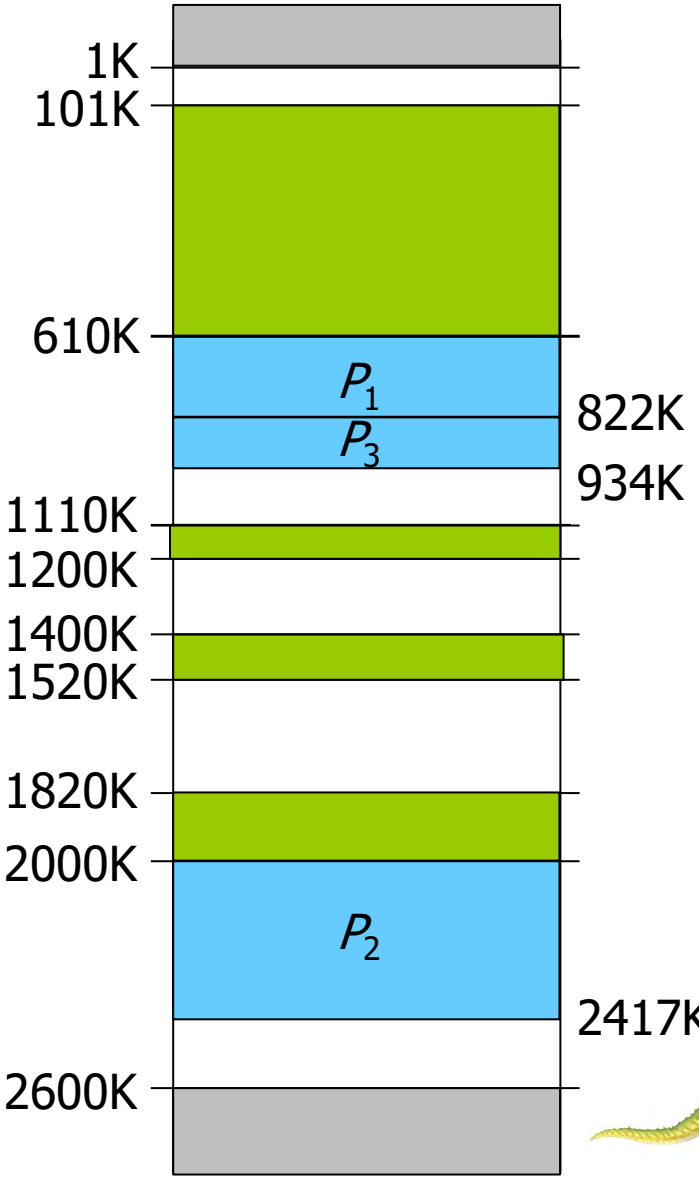
Nodi	Dimensioni	Indirizzo
N_1	100K	1K
N_2	500K	610K
N_3	200K	1200K
N_4	300K	1520K
N_5	600K	2000K

FIRST-FIT

P_1, P_2, P_3, P_4,
di 212K, 417K, 112K, 426K

Frammentazione esterna:

100K+176K+200K+300K+183K=959K



Esempio (cont.)

- Soluzione

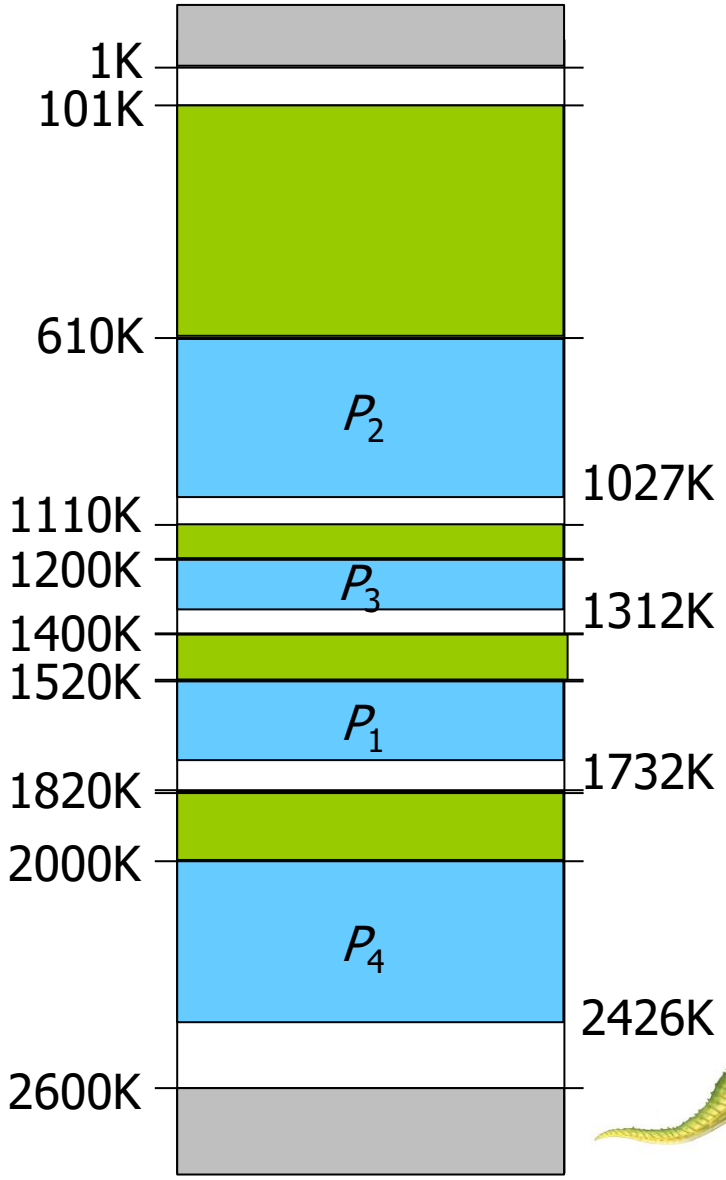
Nodi	Dimensioni	Indirizzo
N_1	100K	1K
N_2	500K	610K
N_3	200K	1200K
N_4	300K	1520K
N_5	600K	2000K

BEST-FIT

P_1, P_2, P_3, P_4,
di 212K, 417K, 112K, 426K

Frammentazione esterna:

100K+83K+88K+88K+174K=533K

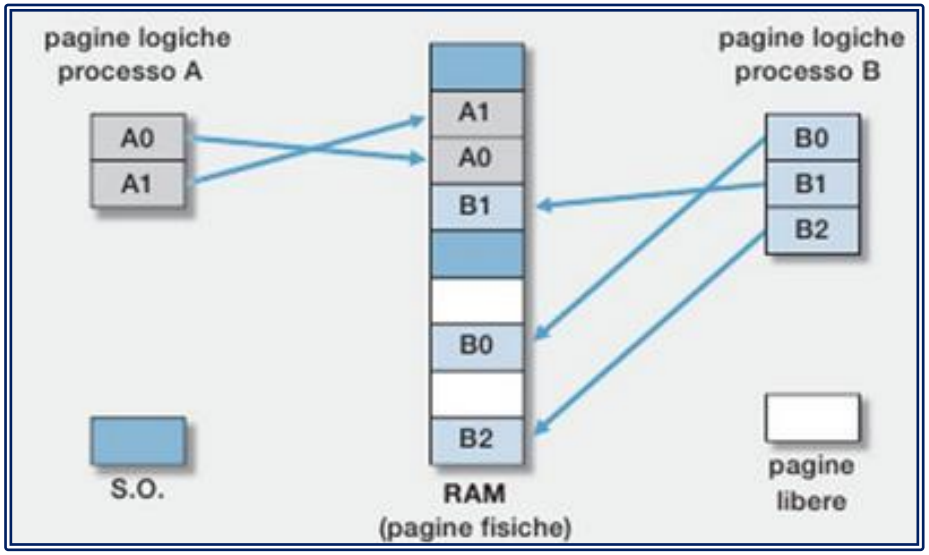


Paginazione – 1

- Con la paginazione, lo spazio degli indirizzi fisici di un processo può essere non contiguo
  - Al processo è allocata memoria fisica ogni volta che quest'ultima si rende disponibile
  - Evita la frammentazione esterna
  - Evita il problema di blocchi di memoria di dimensioni variabili
- In effetti, la paginazione costituisce una soluzione al problema della frammentazione esterna perché consente l'allocazione di memoria fisica solo per blocchi di dimensione prefissata
- Nelle sue varie forme, utilizzata nella maggior parte dei SO, da quelli per mainframe ai sistemi mobili

Paginazione – 2

- I blocchi di memoria fisica, chiamati frame o pagine fisiche, hanno dimensione pari a una potenza del 2 — valori tipici negli elaboratori attuali sono compresi nell'intervallo 4KB–1GB
- La memoria logica viene suddivisa in blocchi della stessa dimensione, chiamati pagine logiche
- Occorre tenere traccia di tutti i frame liberi

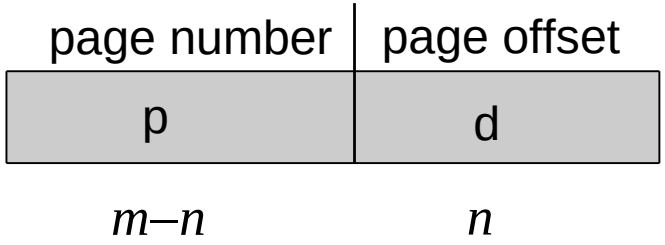


Paginazione – 3

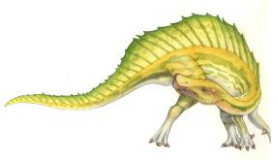
- Si ottiene uno spazio degli indirizzi logici totalmente separato dallo spazio degli indirizzi fisici (e in generale di dimensione diversa)
- Per eseguire un programma di dimensione n pagine, è necessario trovare n frame liberi prima di caricare il programma
- Si impiega una tabella delle pagine per tradurre gli indirizzi logici negli indirizzi fisici corrispondenti
- Si ha solo frammentazione interna (relativa all'ultimo frame)
- Si elimina anche il problema dell'allocazione dinamica nella backing store, perché anch'essa viene divisa in pagine delle stesse dimensioni (o multipli)

Schema di traduzione degli indirizzi

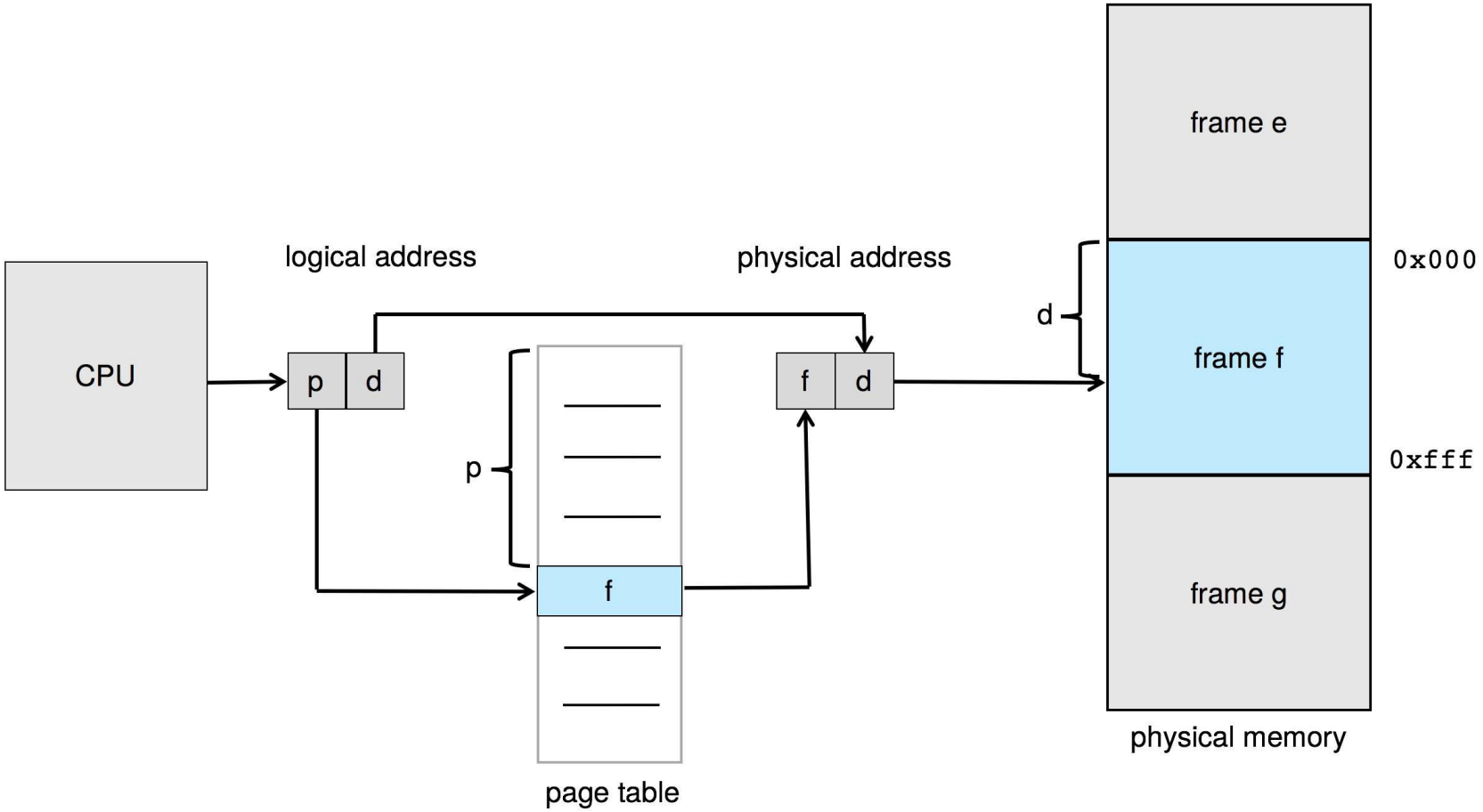
- L'indirizzo logico generato dalla CPU viene suddiviso in:
  - Numero di pagina (p) — impiegato come indice in una tabella delle pagine che contiene l'indirizzo base di ciascun frame nella memoria fisica (o il numero del frame)
  - Offset nella pagina (d) — combinato con l'indirizzo base per definire l'indirizzo fisico che viene inviato all'unità di memoria



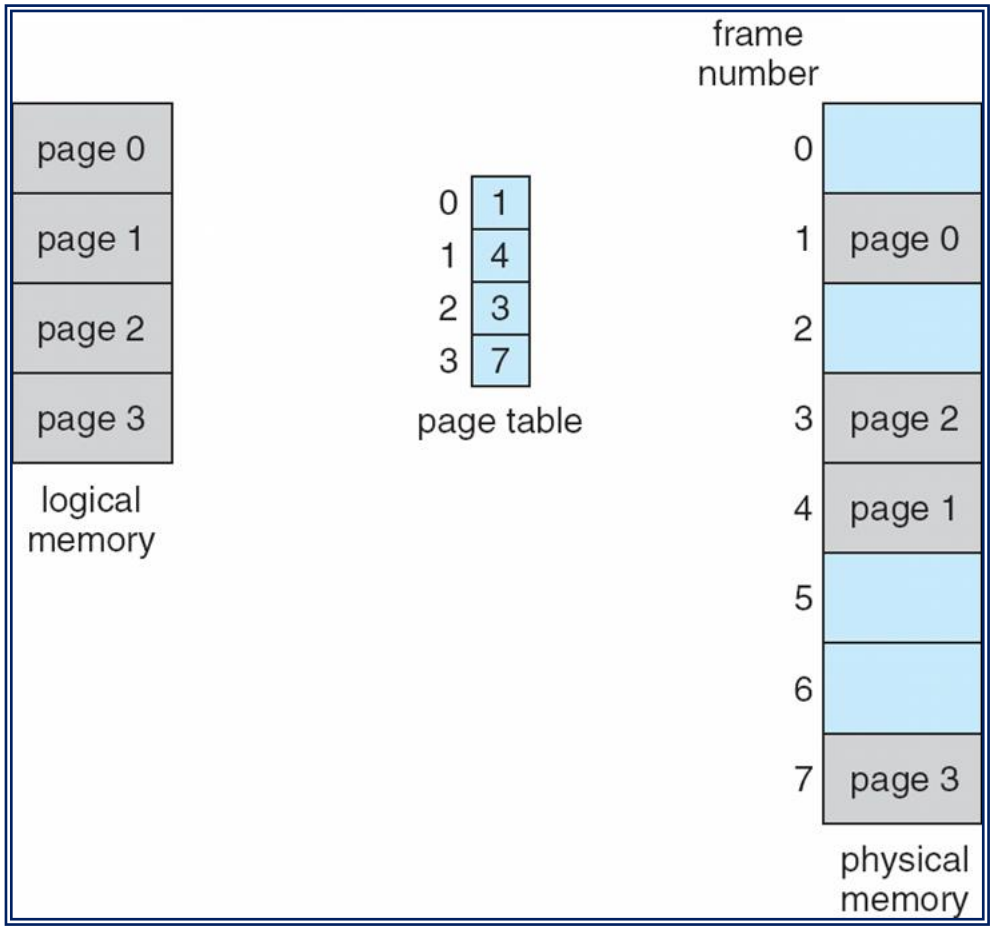
Spazio logico di 2^m indirizzi con 2^{m-n} pagine di dimensione 2^n



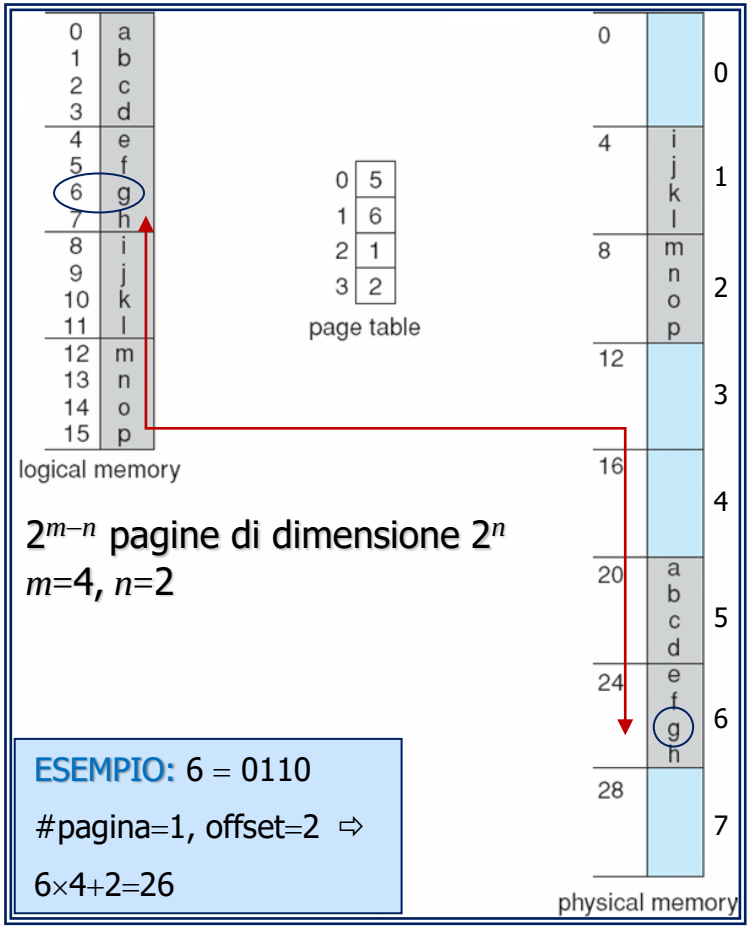
Supporto hardware alla paginazione



Esempi di paginazione



Modello di paginazione di memoria logica e memoria fisica



Memoria fisica da 32 byte con pagine da 4 byte

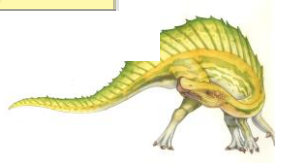
Considerazioni pratiche sulla paginazione

- Esempio
  - Dimensione della pagina: 2048 byte
  - Dimensione del processo: 72766 byte
    - 35 pagine + 1086 byte ⇨ 36 frame
    - Frammentazione interna: 2048−1086=962 byte
- Frammentazione nel caso peggiore: 1 frame – 1 byte; nel caso medio pari a mezzo frame
- Dunque frame piccoli sono preferibili?
  - Non necessariamente, dato che occorre memoria per tener traccia di ogni elemento della tabella delle pagine; inoltre, l'I/O è più efficiente per pagine grandi
  - La dimensione delle pagine cresce nel tempo
    - Windows 11 supporta pagine da 4 KB e 2 MB
    - Linux supporta pagine da 4 KB e *huge page* più grandi, di dimensioni dipendenti dall'architettura (**`getpagesize()`** o, da linea di comando, **`getconf PAGESIZE`**)
- La memoria "vista" dai processi e la memoria fisica differiscono significativamente
- I processi sono "relegati" nel loro spazio di memoria grazie all'implementazione del paging

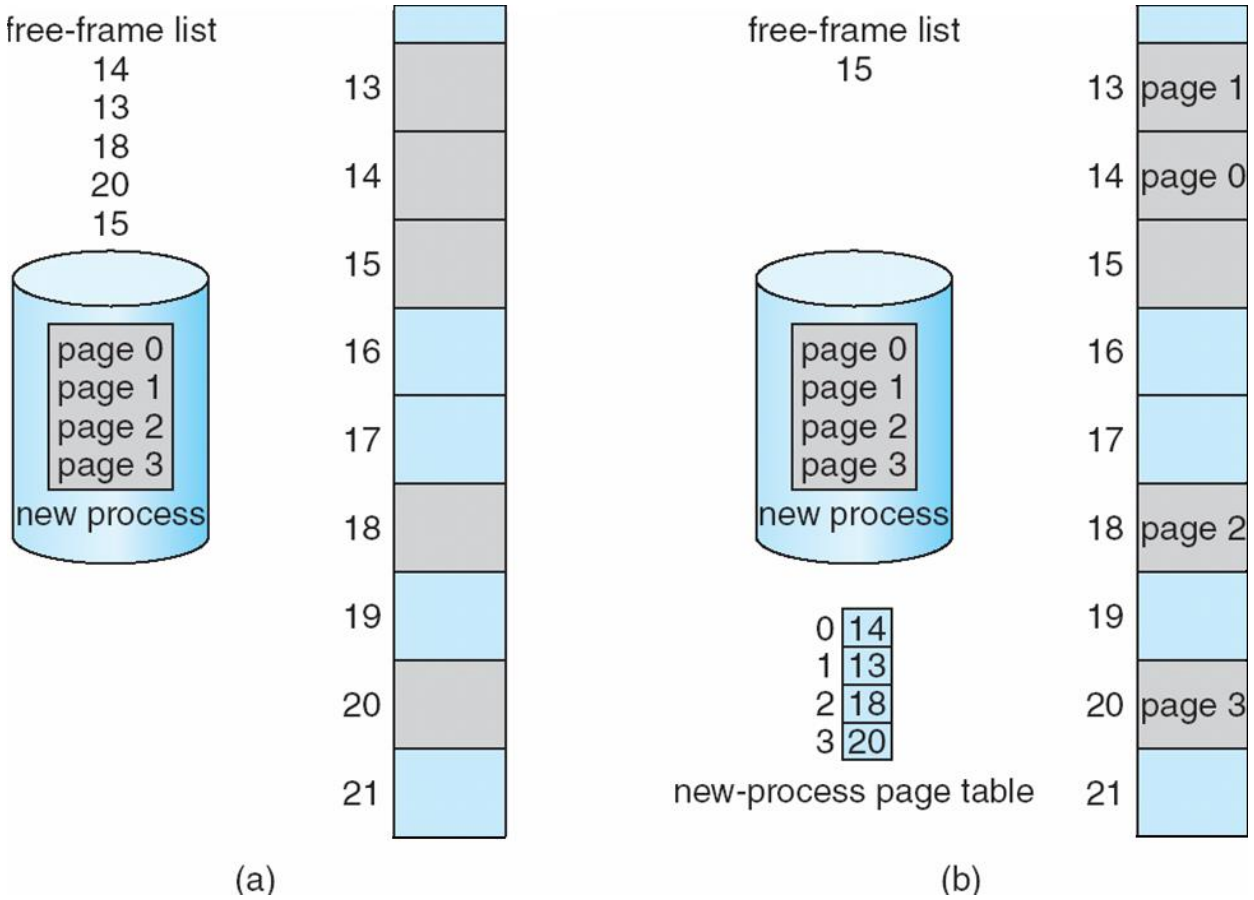
Tabella dei frame

- Poiché il SO gestisce la memoria fisica, deve essere informato su:
  - quali frame sono assegnati e a chi
  - quali e quanti frame sono liberi
- Le informazioni sono contenute nella tabella dei frame
  - Una entry per ciascun frame, per definire se la pagina fisica è libera o assegnata e, nel secondo caso, a quale pagina di quale(/i) processo(/i)

0	P_1 pagina 0
1	P_1 pagina 1
2	P_1 pagina 2
3	P_1 pagina 3
4	
5	P_2 pagina 3
6	
7	P_2 pagina 1
8	
9	P_2 pagina 2
10	P_2 pagina 0
11	P_3 pagina 0
12	
13	
14	
15	P_3 pagina 1



Frame liberi

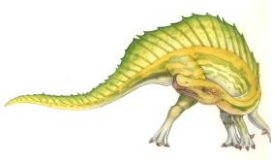


Prima dell'allocazione

Dopo l'allocazione

Ancora sulla paginazione

- La paginazione impone al SO la gestione di strutture dati aggiuntive, sia a livello globale (tabella dei frame, lista dei frame liberi) sia a livello dei singoli task (tabella delle pagine), che occupano spazio di memoria e aggiungono overhead alla computazione
- Per esempio, la tabella delle pagine è usata dal dispatcher per impostare l'hardware di paginazione quando a un processo sta per essere assegnata la CPU ⇨ aumento del tempo di context–switch
- Tuttavia, la paginazione consente di utilizzare al meglio lo spazio di memoria e di indirizzare una memoria fisica che è decisamente più grande di quella (logica) indirizzabile direttamente dall'architettura hardware
  - Esempio: per una CPU a 32 bit e pagine da 4KB
    - Spazio logico: 2^{32} Byte = 4GB
    - Spazio fisico: 2^{32} frame da 4KB = $2^{32} \times 2^{12}$ Byte = 16TB (se ogni elemento della tabella delle pagine è costituito da 4 byte)

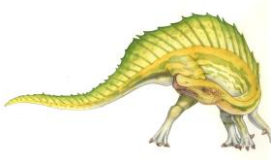


Implementazione della tabella delle pagine – 1

- Originariamente implementata con un insieme di registri dedicati
  - Traduzione dell'indirizzo molto efficiente
  - Cambio di contesto molto lungo
  - Non più di 256 elementi (ma i processi attuali sono costituiti anche da un milione di pagine…)
- Attualmente, la tabella delle pagine risiede in memoria centrale; in generale, si ha una tabella per ogni processo
- Il registro Page–Table Base Register (PTBR) punta all'inizio della tabella
- Il registro Page–Table Length Register (PTLR) indica la dimensione della tabella (il numero di elementi)
- Con questo schema, ogni accesso a dati o istruzioni richiede di fatto due accessi alla memoria: uno per la tabella e uno per le istruzioni/dati

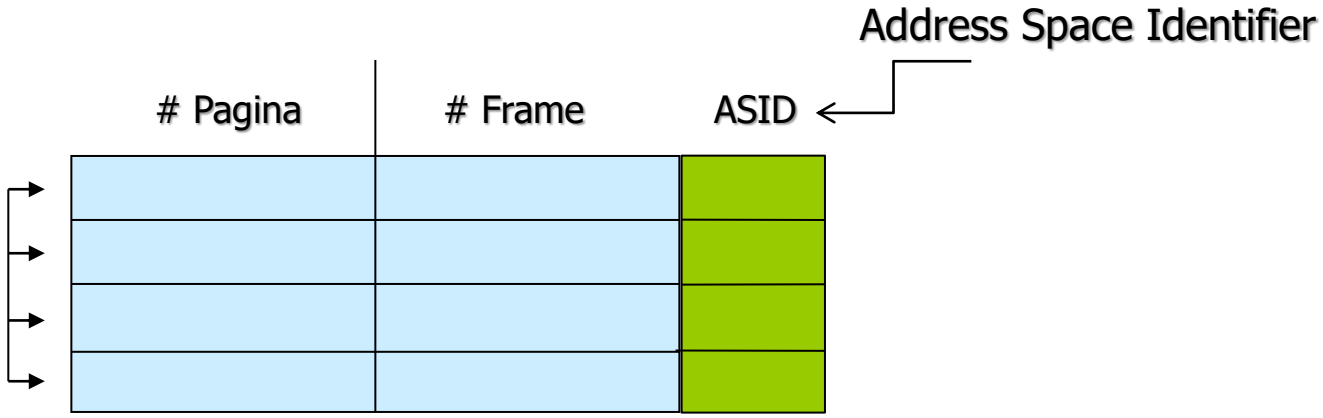
Implementazione della tabella delle pagine – 2

- Il doppio accesso alla memoria può essere evitato con l'uso di registri associativi (altrimenti detti *translation look–aside buffer*, TLB), attraverso i quali si effettua una ricerca parallela veloce su una piccola tabella (64–1024 elementi)
- Alcune TLB memorizzano un *address space identifier* (ASID) per ciascun elemento, così da identificare univocamente il processo e garantire la protezione del suo spazio di indirizzi
  - Prima di calcolare l'indirizzo fisico, si controlla la corrispondenza fra l'ASID del processo in esecuzione e quello associato alla particolare pagina virtuale: la mancata corrispondenza è un *TLB miss*
  - L'ASID consente alla TLB di contenere, nello stesso istante, elementi di diversi processi
  - Senza l'ASID, ad ogni cambio di contesto, la TLB deve essere svuotata (*flush*), per evitare l'uso di informazioni errate da parte del processo attualmente in esecuzione



Memoria associativa – 1

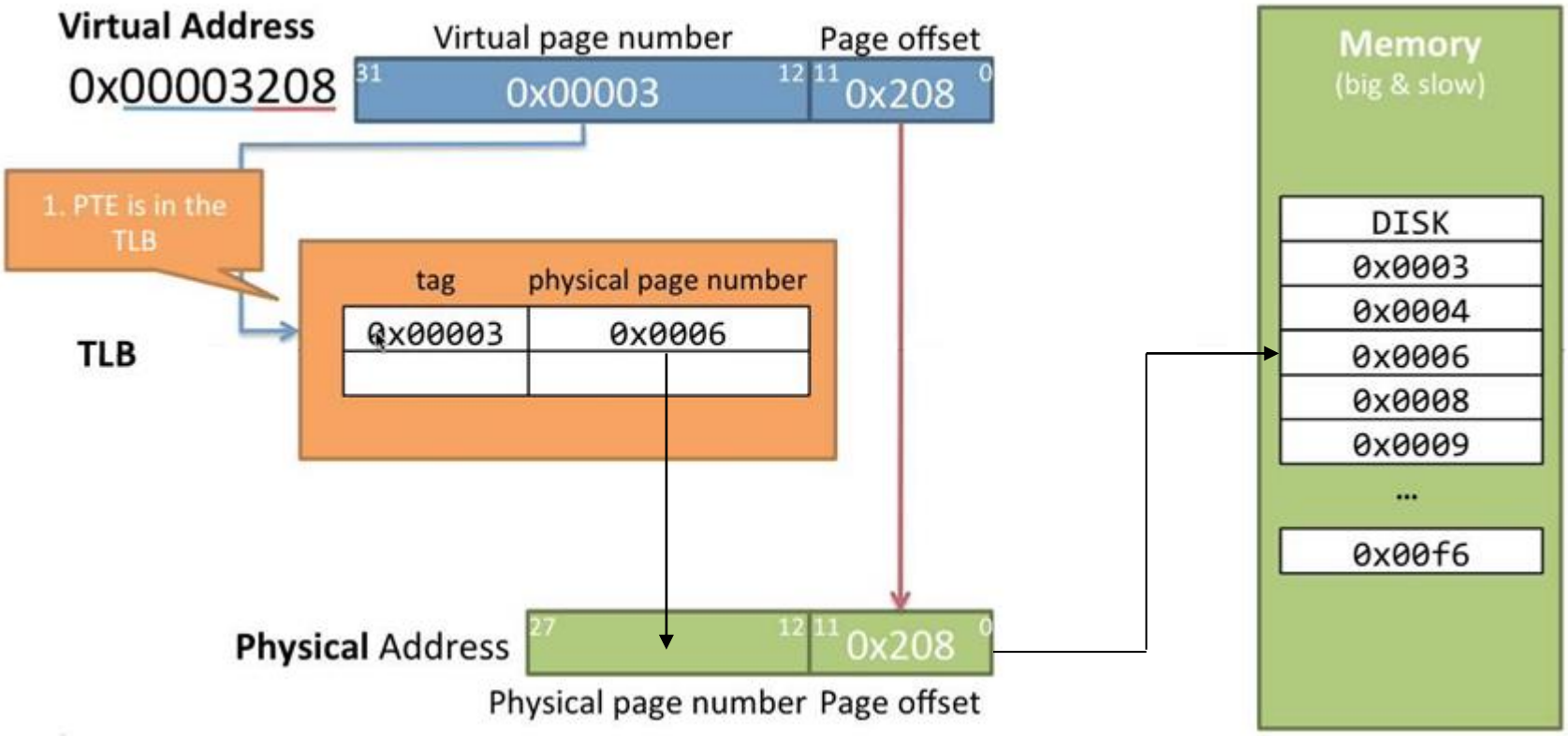
- Memoria associativa – ricerca parallela



Traduzione dell'indirizzo (p, d)

- Se p è nel registro associativo, si reperisce il numero del frame
- Altrimenti, si recupera il numero di frame dalla tabella delle pagine in memoria

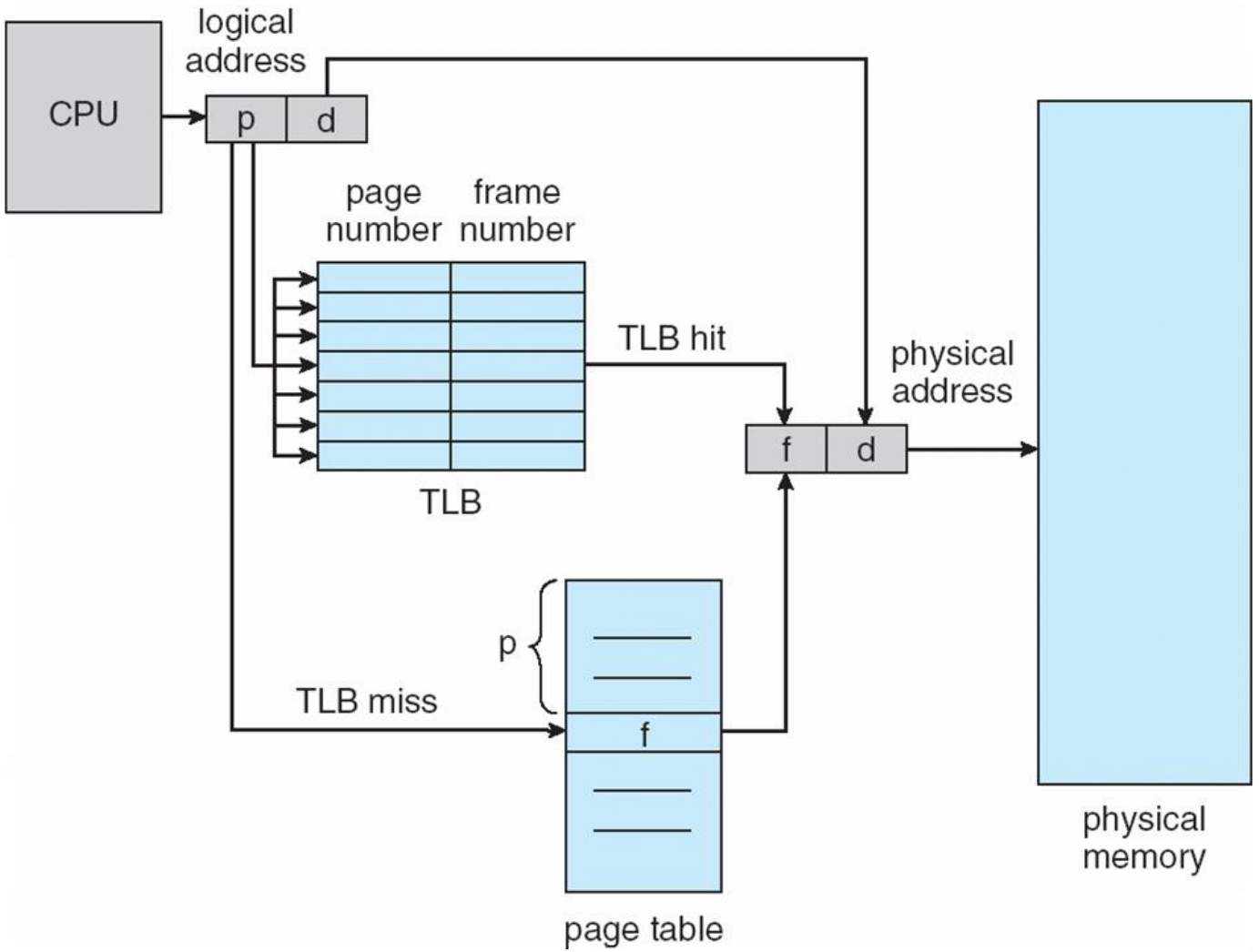
Memoria associativa – 2

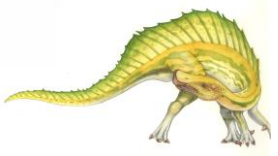


Memoria associativa – 3

- A fronte di un TLB–miss, l'informazione deve essere spostata nella TLB per poter essere poi acceduta rapidamente
  - Dovranno essere attuate politiche di sostituzione degli elementi della TLB (LRU, RR, scelta casuale – il SO può intervenire, ma più spesso la scelta avviene a livello hardware)
  - Alcuni elementi potranno essere "vincolati" (*wired down*) per ottenerne un accesso rapido permanente
    - Normalmente elementi relativi al codice del kernel

Hardware di paginazione con TLB





Tempo di accesso effettivo – 1

- *Lookup associativo* = ε unità di tempo
- *Hit ratio* α — percentuale di volte che un numero di pagina viene reperito nei registri associativi; è correlata al numero di registri associativi
- Tempo di accesso effettivo (EAT, *Effective Access Time*)

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

dove si suppone che un accesso alla memoria abbia la durata $t=10\nu sec$ e $\varepsilon \cong 1\nu sec$

- Si consideri α=80%; conseguentemente…

$$EAT = 0.80\times 11 + 0.20\times 21 = 13\nu sec$$

- Con una hit ratio più realistica $\rightarrow$ α=99%:

$$EAT = 0.99\times 11 + 0.01\times 21 = 11.1\nu sec$$

Tempo di accesso effettivo – 2

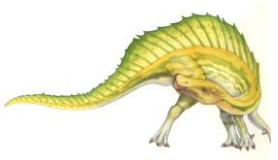
- Nelle architetture hardware attuali, la ricerca nella TLB è parte della pipeline delle istruzioni
  - ⇨ Nessuna penalizzazione in termini di prestazioni
  - ⇨ La TLB deve però essere piccola (64–1024 voci)
  - ⇨ TLB separate per istruzioni e dati, per aumentare il numero di "elementi" disponibili, poiché le due ricerche vengono effettuate in diversi stadi della pipeline
- Si consideri α=80% e sia 10νsec il tempo di accesso alla memoria

$$EAT = 0.80 \times 10 + 0.20 \times 20 = 12\nu\text{sec}$$

  - ⇨ Rallentamento del tempo d'accesso alla memoria del 20%
- Con una hit ratio α=99%, invece, si ha:

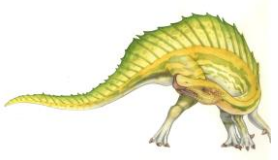
$$EAT = 0.99 \times 10 + 0.01 \times 20 = 10.1\nu\text{sec}$$

  - ⇨ Rallentamento di un solo punto percentuale



Tempo di accesso effettivo – 3

- EAT difficile da calcolare nelle architetture moderne dove possono essere presenti più livelli di TLB
- Esempio: Intel Core i7
  - Due TLB L1, da 128 elementi per le istruzioni e da 64 per i dati (tempo di accesso incluso nella pipeline)
  - Una TLB L2, da 512 elementi comune ad entrambi (6 cicli di CPU per l'accesso)
  - Centinaia di cicli di CPU per accedere alla RAM (direttamente – via hardware – o con interrupt al SO)



Esempio

- Si consideri un sistema di paginazione con la tabella delle pagine conservata in memoria
  - Se un riferimento alla memoria necessita di 20νsec per essere servito, di quanto necessiterà un riferimento alla memoria paginata?
  - Se si aggiunge un TLB con hit ratio del 75%, quale sarà il tempo effettivo di riferimento alla memoria? Si ipotizzi che un accesso al TLB richieda un tempo pari a 2νsec
- Soluzione
  - Un riferimento alla memoria paginata necessita di 40νsec
  - Con l'aggiunta della TLB

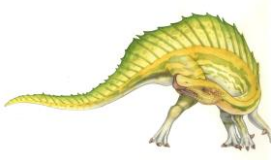
$$EAT = 0.75 \times 22 + 0.25 \times 42 = 27\nu\text{sec}$$

Protezione della memoria – 1

- La protezione della memoria è implementata associando uno o più bit di protezione a ciascun frame nella tabella delle pagine
  - I bit determinano se una certa pagina può essere acceduta in sola lettura, lettura/scrittura e/o esecuzione
- Tutti i riferimenti alla memoria passano attraverso la tabella delle pagine
  - ⇨Mentre si effettua il mapping da indirizzi logici ad indirizzi fisici, si verificano le modalità di accesso al dato frame
  - ⇨Un tentativo di accesso che viola quanto codificato nei bit di protezione provoca una trap al SO

Protezione della memoria – 2

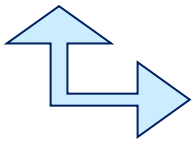
- Nelle architetture più datate, in alternativa all'utilizzo del Page–Table Length Register, un ulteriore bit di validità veniva associato ad ogni elemento della tabella delle pagine:
  - Un valore "valido" indicava che la pagina era nello spazio degli indirizzi logici del processo, e quindi era legale
  - Un valore "non valido" indicava che la pagina non si trovava nello spazio degli indirizzi logici del processo
  - ⇨ Il bit di validità consentiva di riconoscere gli indirizzi illegali e di notificare l'anomalia con un segnale di eccezione
- È il sistema operativo che concede o revoca il diritto di accesso ad una data pagina, impostando opportunamente il bit di validità o verificando la legittimità dell'indirizzo logico (in base al contenuto del PTLR)



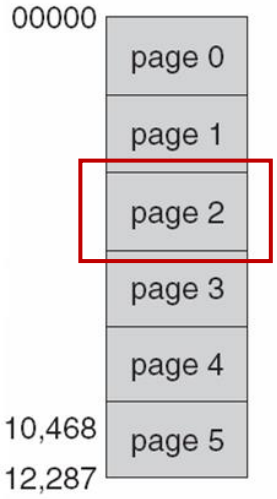
Bit di validità nella tabella delle pagine

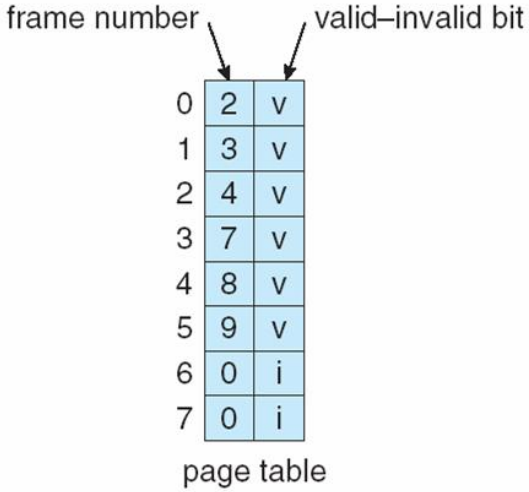
Spazio degli indirizzi logici per ogni processo costituito da 14 bit (16384 byte)

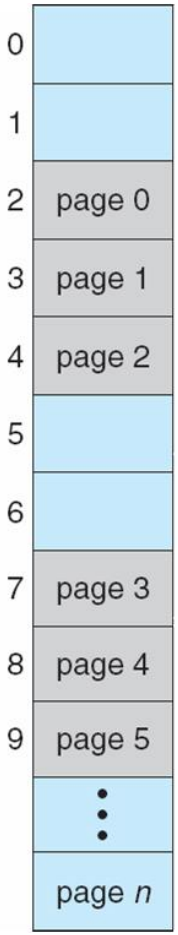
Pagine/frame di 2KB



Processo che occupa 10468 byte





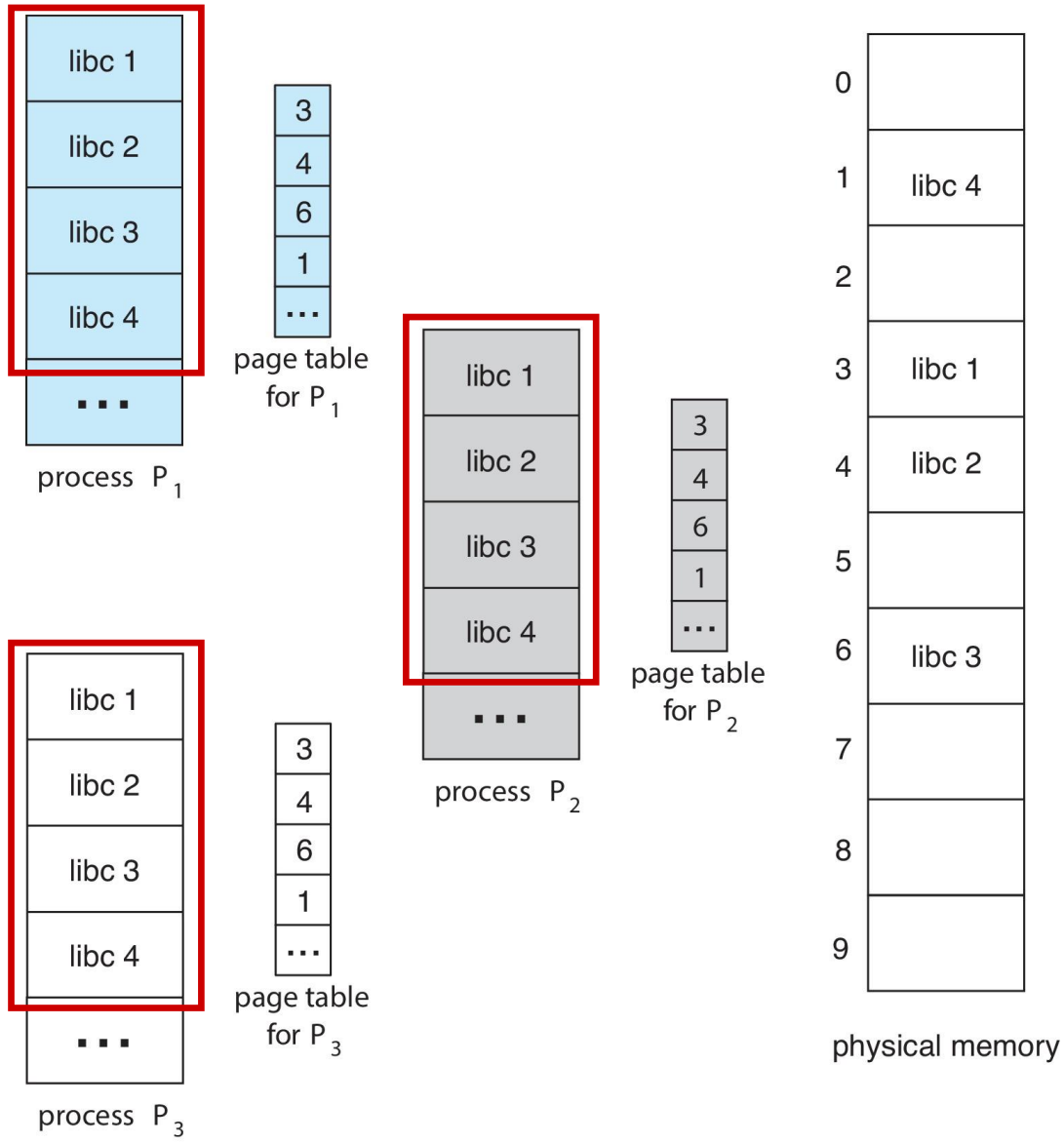


Gli indirizzi da 10469 a 12287 sono di fatto non utilizzati ma vengono considerati come validi a causa della frammentazione interna

Pagine condivise

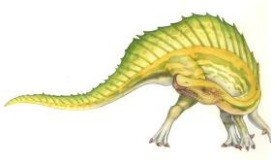
- Codice condiviso
  - Una copia di codice di sola lettura (*rientrante*) viene condivisa fra processi (ad es.: librerie, text editor, compilatori, sistemi a finestre)
  - Simile ai thread che condividono lo stesso "spazio di indirizzi del task" di appartenenza
  - Il codice condiviso deve apparire nella stessa locazione nello spazio degli indirizzi logici di tutti i processi
- Codice e dati privati
  - Ciascun processo mantiene una copia separata dei dati e del codice
  - Le pagine di codice e dati privati possono apparire ovunque nello spazio degli indirizzi logici
- Pagine condivise utili per l'IPC quando la memoria è accessibile sia in lettura che in scrittura

Esempio di pagine condivise



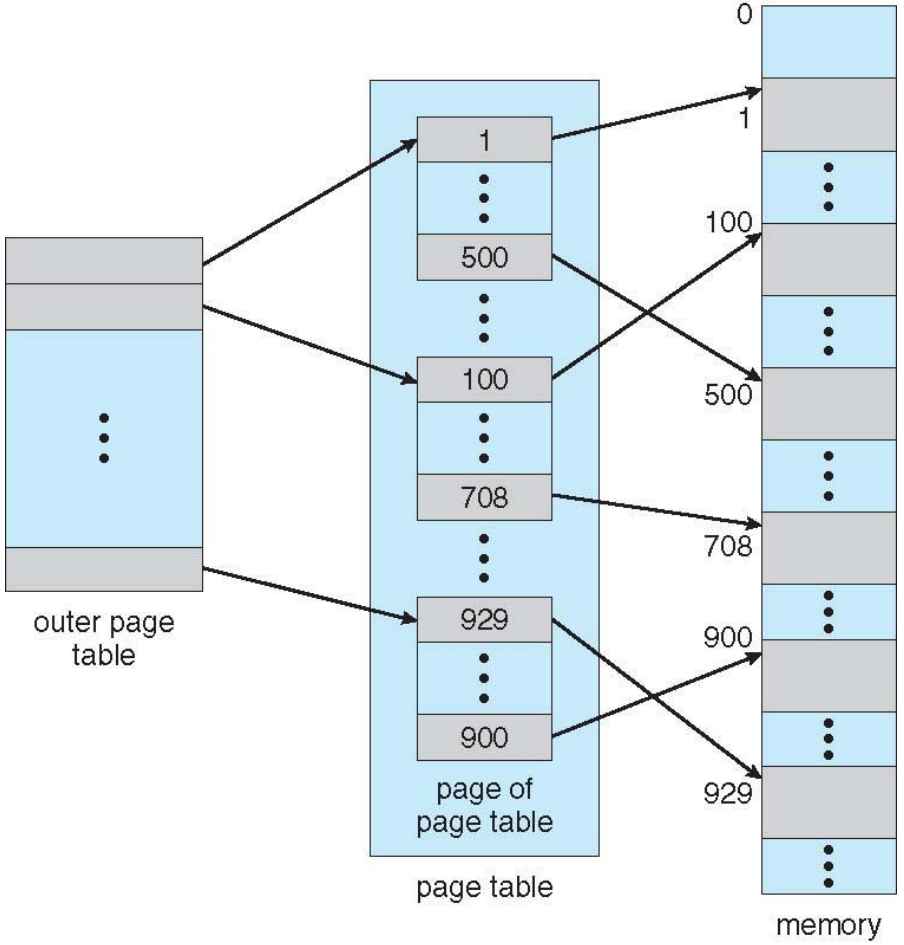
Struttura della tabella delle pagine

- L'occupazione di memoria dovuta alle strutture dati necessarie per gestire la paginazione può divenire eccessivamente grande; si consideri:
  - Uno spazio di indirizzi logici a 32 bit, con pagine di dimensione pari a 4KB (2^{12} byte)
    - ⇨La tabella delle pagine potrebbe essere costituita da più di un milione di elementi ($2^{32}/2^{12}$)
  - Se ciascun elemento occupasse 4 byte
    - ⇨4 MB di occupazione di memoria per la sola tabella delle pagine
    - ⇨Meglio evitare di collocare la tabella delle pagine in modo contiguo in memoria centrale
- Soluzione: ottimizzare l'occupazione di memoria e/o l'accesso all'informazione contenuta
  - Paginazione gerarchica
  - Tabella delle pagine hash
  - Tabella delle pagine invertita



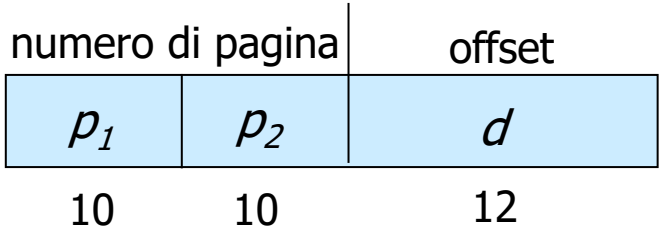
Paginazione gerarchica

- Dividere lo spazio degli indirizzi logici in più tabelle delle pagine
- Un metodo semplice consiste nell'adottare un algoritmo di paginazione a due livelli
- La tabella delle pagine viene essa stessa paginata



Esempio di paginazione gerarchica

- Un indirizzo logico, in architetture a 32 bit con dimensione della pagina di 4 KB, viene suddiviso in:
  - un numero di pagina a 20 bit
  - un offset all'interno della pagina di 12 bit
- Dato che la tabella delle pagine è paginata, il numero di pagina viene ulteriormente suddiviso in:
  - un numero di pagina di 10 bit (tabella esterna)
  - un offset di 10 bit (tabella delle pagine)
- Pertanto, un indirizzo logico si rappresenta come:

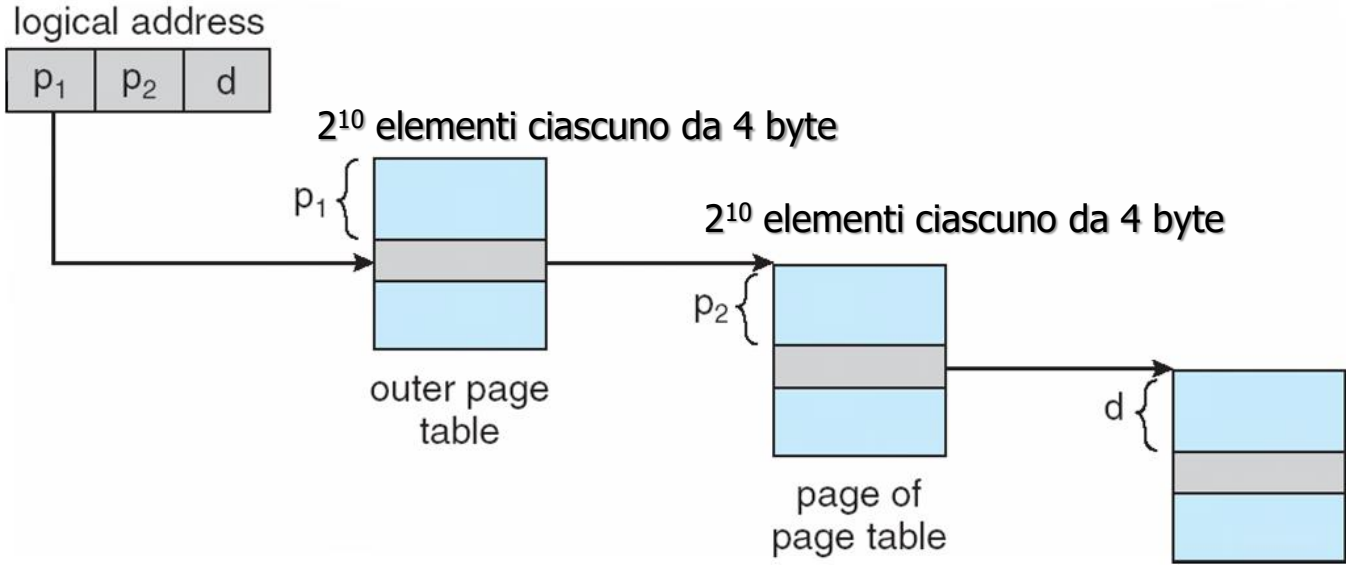


dove p_1 è un indice nella tabella delle pagine esterna e p_2 rappresenta lo spostamento all'interno della pagina indicata dalla tabella esterna (4 byte per ogni elemento)

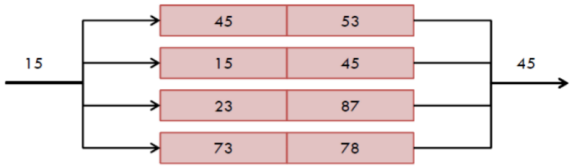
- Metodo noto anche come tabella della pagine ad associazione diretta (*forward–mapped page table*)

Schema di traduzione degli indirizzi

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

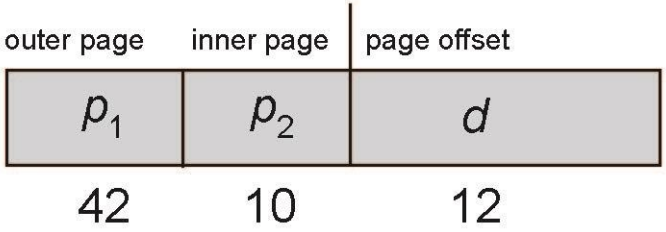


- Dato che ciascun livello viene memorizzato come una tabella separata in memoria, la traduzione di un indirizzo logico in uno fisico può richiedere tre accessi alla memoria
- Anche se il tempo richiesto per un accesso alla memoria è triplicato, la presenza di TLB consente di mantenere prestazioni ragionevoli



Paginazione gerarchica (cont.)

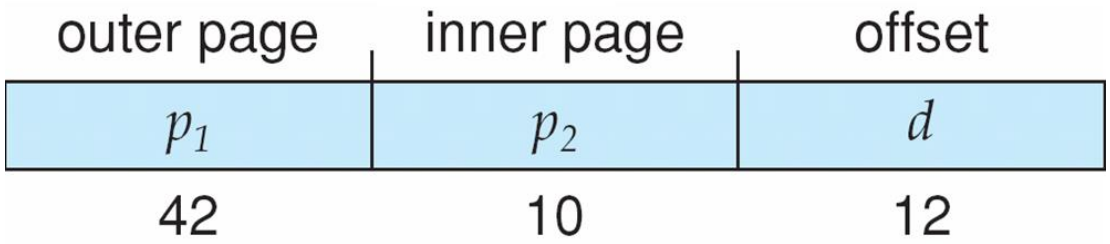
- Lo schema di paginazione a due livelli non è più adeguato nel caso di sistemi con spazio di indirizzi a 64 bit
  - Per pagine di 4 KB, la tabella delle pagine conterrebbe 2^{52} elementi
  - Con una struttura a due livelli la tabella interna delle pagine potrebbe essere costituita da 2^{10} elementi da 4 byte ciascuno
  - L'indirizzo avrebbe la forma

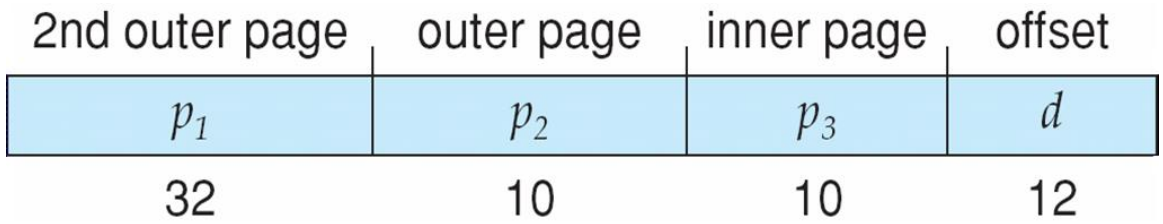


  - La tabella delle pagine più esterna ha 2^{42} elementi e consta di 2^{44} byte
  - Si opta per una soluzione a più livelli di paginazione (da 3 a 7 nelle architetture a 32/64 bit esistenti)

Paginazione gerarchica (cont.)

- Nell'esempio seguente, la tabella delle pagine esterna di secondo livello ha ancora una dimensione pari a 2^{34} byte





  - ⇨ Necessari 4 accessi alla memoria per reperire un dato
- ⇨ Per le architetture a 64 bit la paginazione gerarchica è da considerarsi inadeguata a causa dei costi proibitivi di accesso alla memoria in caso di TLB miss

Tabella delle pagine hash – 1

- Comune per spazi di indirizzi a più di 32 bit
- L'argomento della funzione hash è il numero della pagina virtuale
  - Per la gestione delle collisioni, ogni elemento della tabella hash contiene una lista concatenata di elementi che la funzione hash fa corrispondere alla stessa posizione nella tabella
- Ciascun elemento della tabella è composto da tre campi
  - Numero della pagina virtuale
  - Indirizzo del frame corrispondente
  - Puntatore al successivo elemento nella lista
- I numeri di pagina virtuale vengono confrontati con tutti gli elementi della catena
  - A fronte di una ricerca positiva, si estrae il corrispondente numero di frame

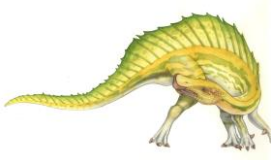


Tabella delle pagine hash – 2

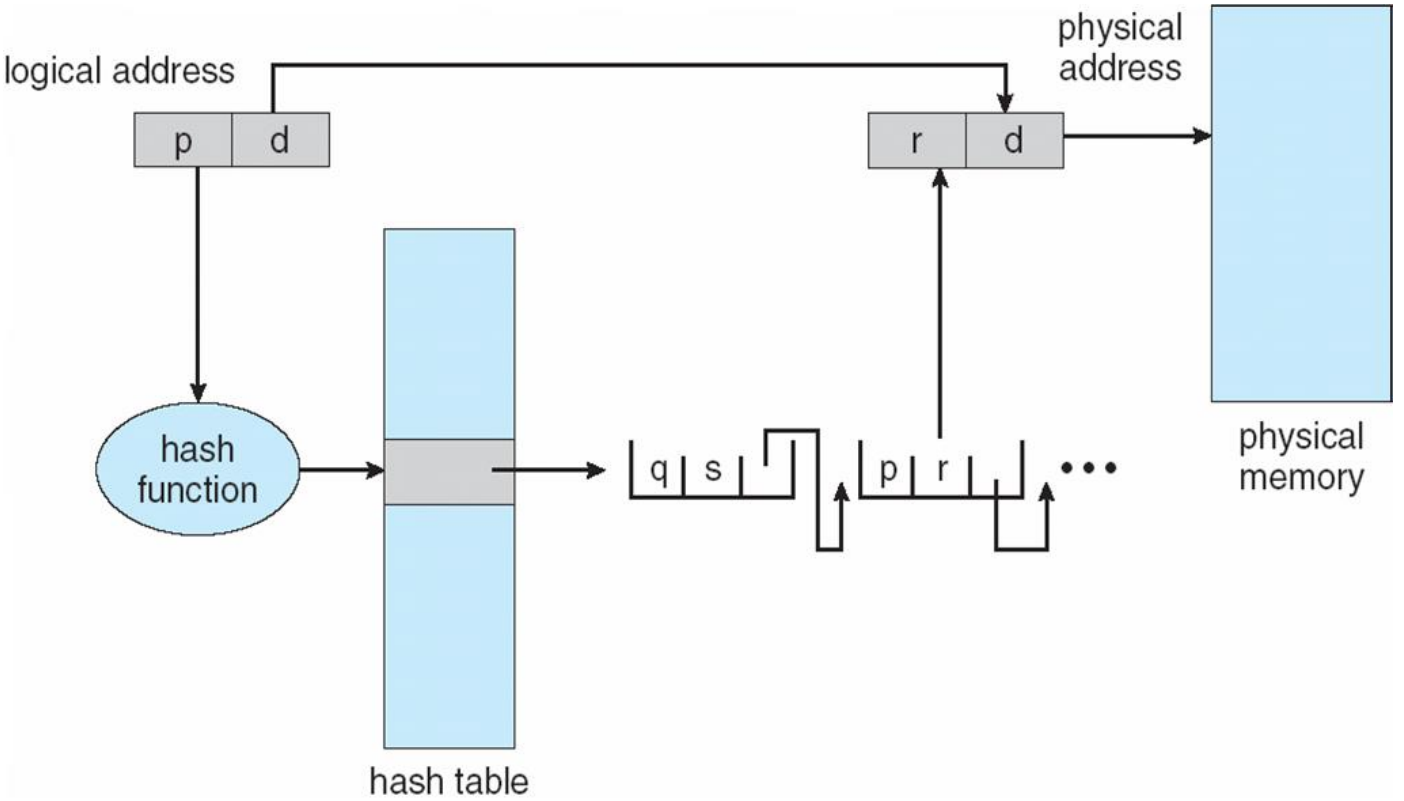


Tabella delle pagine hash – 3

- Le tabelle delle pagine hash sono particolarmente utili per spazi di indirizzi logici sparsi, in cui riferimenti alla memoria non sono contigui (situazione un po' irrealistica)
- Per spazi a 64 bit si utilizza la tabella delle pagine a gruppi (clustered page table)
  - Simile ad una tabella hash, ma ciascun elemento della tabella delle pagine contiene riferimenti alle pagine fisiche corrispondenti ad un gruppo di pagine virtuali contigue (per esempio 16)
    - ⇨Si riduce lo spazio di memoria richiesto
    - ⇨Dato che lo spazio di indirizzi dei programmi è naturalmente "contiguo in tempo e spazio" (località negli accessi alla memoria), aumentano le prestazioni del sistema

Tabella delle pagine invertita – 1

- Generalmente, si associa una tabella delle pagine ad ogni processo, che contiene un elemento per ogni pagina virtuale che il processo sta utilizzando (o un elemento per ogni possibile indirizzo virtuale, a prescindere dalla validità)
- Rappresentazione naturale, dato che i processi si riferiscono alle pagine per mezzo di indirizzi virtuali ed il SO si occupa della traduzione in indirizzi fisici
- Problema: ogni tabella delle pagine può contenere milioni di elementi ed occupare molta memoria

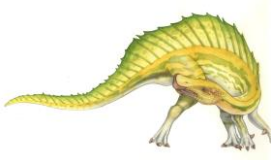
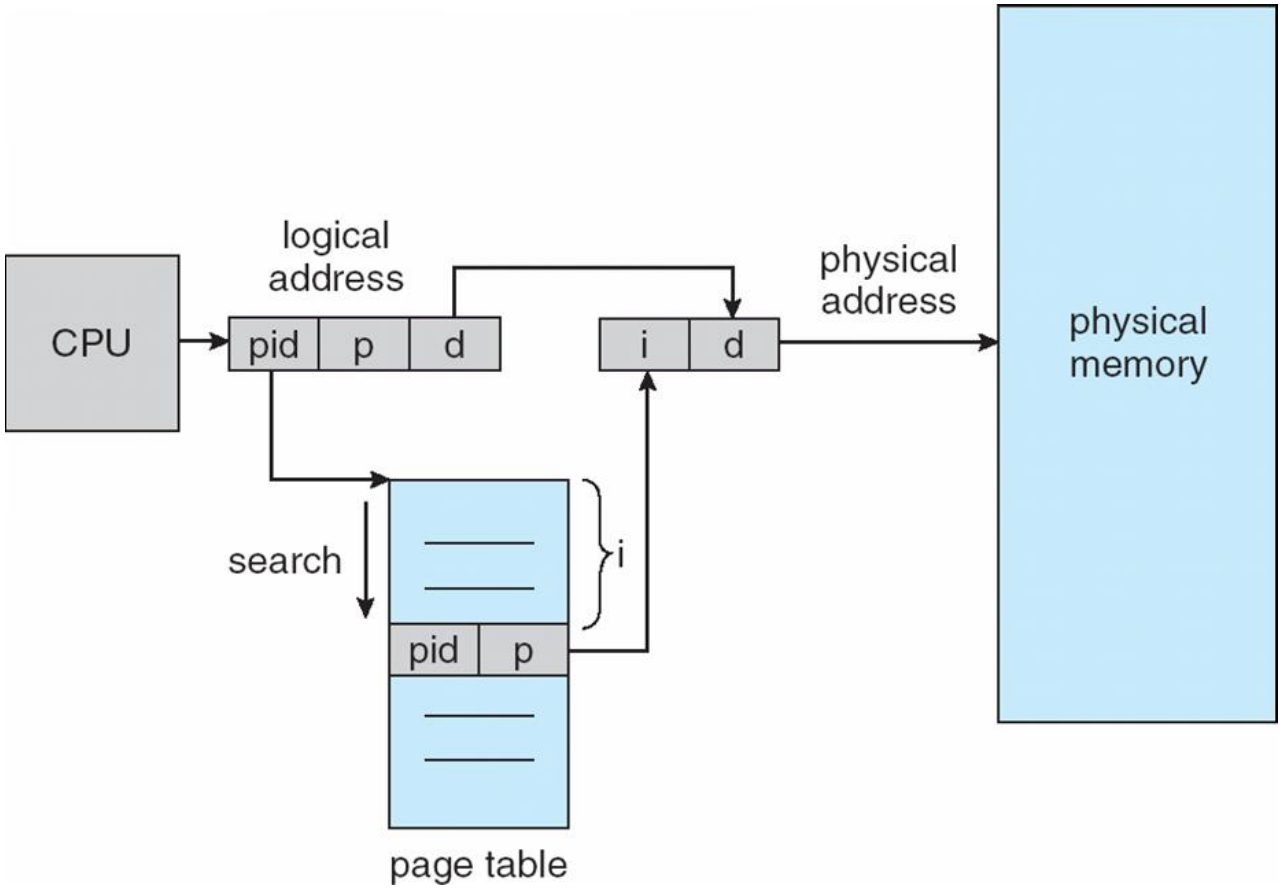


Tabella delle pagine invertita – 2

- Soluzione: tabella delle pagine invertita
  - Un elemento della tabella per ogni frame
  - Gli elementi della tabella contengono il numero della pagina virtuale memorizzata nel dato frame e l'identificativo del processo che possiede tale pagina
  - Riduzione della memoria richiesta per realizzare il mapping indirizzi logici/fisici
  - Incremento del tempo necessario per ricercare nella tabella quando si fa un riferimento a pagina

  ⇨ Notare che è necessario ricercare su tutta la tabella!

Tabella delle pagine invertita – 3



Ciascun indirizzo virtuale è formato dalla tripla: <#processo, #pagina, offset>

Tabella delle pagine invertita – 4

- Difficile la realizzazione della memoria condivisa, dovuta alla presenza di un solo elemento indicante la pagina virtuale corrispondente ad ogni pagina fisica
  - Un riferimento da parte di un altro processo che condivide la memoria provocherà un errore di pagina e sostituirà la mappatura con un indirizzo virtuale diverso

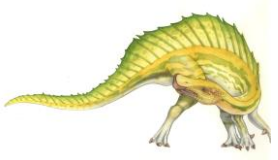
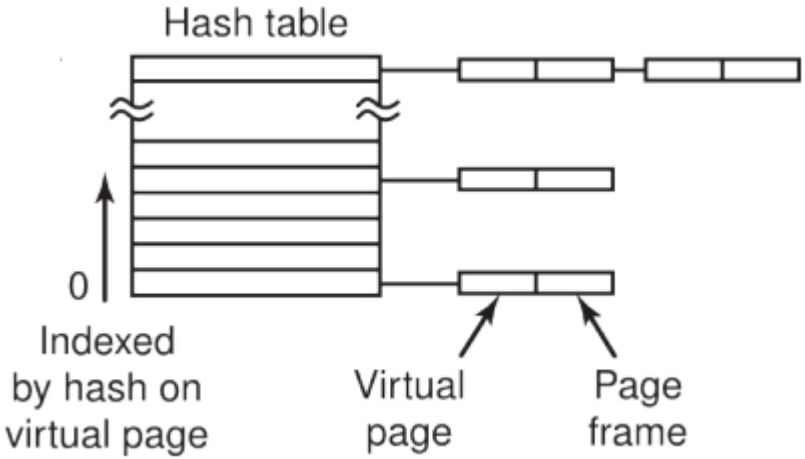


Tabella delle pagine invertita – 5

- Da un punto di vista implementativo, si possono impiegare tabelle hash per limitare la ricerca ad uno — o al più pochi — elementi della tabella
- Ogni accesso alla tabella hash aggiunge comunque un riferimento alla memoria
  - Per migliorare le prestazioni, si usa una TLB



Esempio 1

- Si consideri un sistema di traduzione da indirizzamento logico a indirizzamento fisico basato su paginazione. Lo spazio logico di un programma è costituito da un massimo di 64 byte, suddivisi in pagine da 4 byte. La memoria fisica è costituita da 256 byte. Calcolare:
  - Da quanti bit sono costituiti gli indirizzi logici e gli indirizzi fisici;
  - Da quanti bit sono costituiti i numeri di pagina e di frame.

  Ad un dato istante, la tabella delle pagine di un processo (costituito da 8 pagine logiche) è la seguente:

# pagina logica	# pagina fisica
0	12
1	1
2	17
3	62
4	11
5	16
6	61
7	12

  Tradurre in indirizzi fisici i seguenti indirizzi logici: 0, 2, 4, 9, 19, 11, 22, 30, 26, 23, 14.

Esempio 1 (cont.)

- Soluzione
  - Gli indirizzi logici sono rappresentati su 6 bit, 4 per il numero di pagina e 2 per l'offset
  - Gli indirizzi fisici sono rappresentati su 8 bit, 6 per il numero di frame e 2 per l'offset
  - Mapping indirizzi logici → indirizzi fisici

$0 \rightarrow 0000|00 \rightarrow 001100|00$
$2 \rightarrow 0000|10 \rightarrow 001100|10$
$4 \rightarrow 0001|00 \rightarrow 000001|00$
$19 \rightarrow 0100|11 \rightarrow 001011|11$
$11 \rightarrow 0010|11 \rightarrow 010001|11$
$22 \rightarrow 0101|10 \rightarrow 010000|10$
$30 \rightarrow 0111|10 \rightarrow 001100|10$
$26 \rightarrow 0110|10 \rightarrow 111101|10$
$23 \rightarrow 0101|11 \rightarrow 010000|11$
$14 \rightarrow 0011|10 \rightarrow 111110|10$

# pagina logica	# pagina fisica
0	12
1	1
2	17
3	62
4	11
5	16
6	61
7	12

Esempio 2

- Sia dato uno spazio di indirizzi logici a 32 bit. Un programma ha il segmento di testo di 200KB, quello dei dati di 100KB e uno stack di 50KB. Se la memoria è organizzata a pagine da 16KB, qual è dimensione in byte della tabella delle pagine del processo?
- Soluzione

  200KB/16KB=12.5 → 13 frame

  100KB/16KB=6.25 → 7 frame

  50KB/16KB=3.125 → 4 frame

  La tabella delle pagine è costituita da 24 elementi ognuno da 32 bit

  » 96 byte

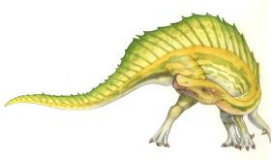
Esempio 3

- Si consideri uno spazio degli indirizzi logici costituito da 128 pagine da 2KB, mappato su una memoria fisica di 64 frame.
  - Quanti bit servono per descrivere lo spazio degli indirizzi logici? E quello degli indirizzi fisici?
  - Qual è la dimensione massima di un processo?
  - Qual è la dimensione della tabella delle pagine invertita supponendo che il sistema non supporti l'esecuzione contemporanea di più di 512 processi alla volta?
  - Calcolare numeri di pagina logica e offset relativi agli indirizzi seguenti (in decimale): 3315, 18363, 10000, 512, 16385.

Esempio 3 (cont.)

- Soluzione
  - Per descrivere lo spazio degli indirizzi logici servono 18 bit, per gli indirizzi fisici ne servono 17.
  - La dimensione massima di un processo è 256KB.
  - La tabella delle pagine invertita contiene 64 elementi ciascuno dei quali contiene un numero di pagina logica (7 bit) ed un identificativo di processo (9 bit)
    $\rightarrow$ 64×2byte = 128 byte
  - Pagina logica e offset

    3315 = 0000001|10011110011
    18363 = 0001000|11110111011
    10000 = 0000100|11100010000
    512 = 0000000|01000000000
    16385 = 0001000|00000000001

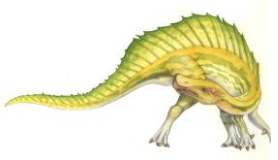


Swapping – 1

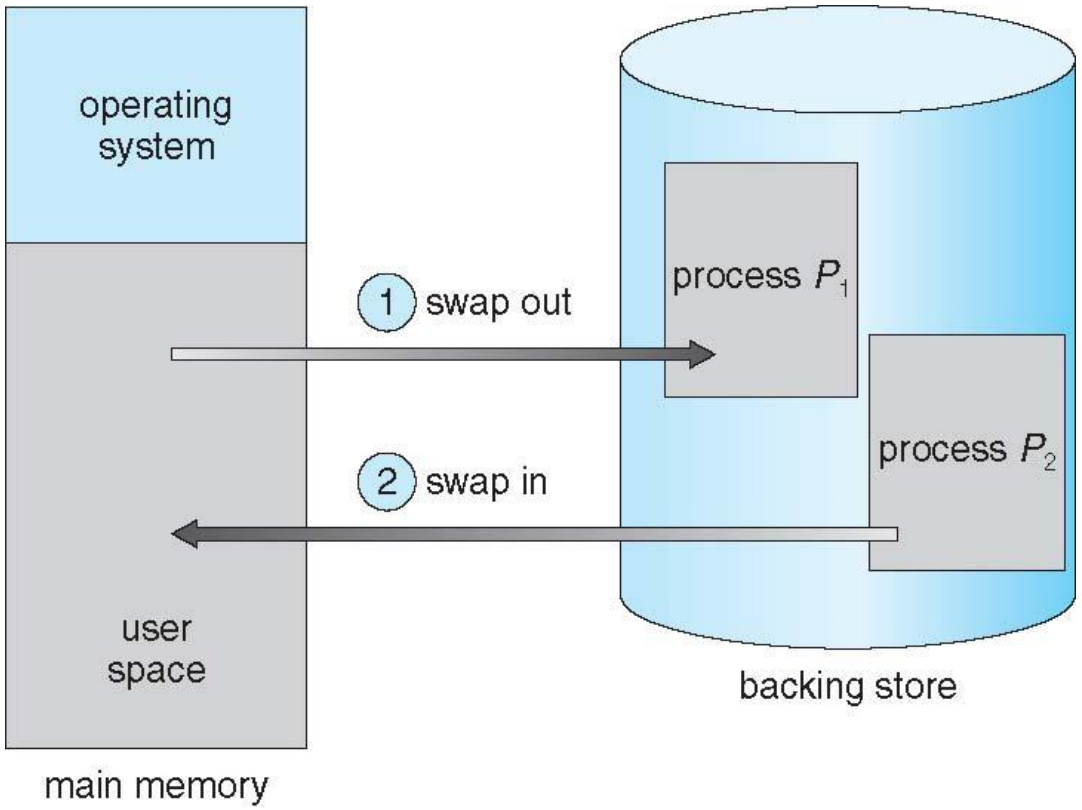
- Un processo può venire temporaneamente riversato – swapped out – dalla memoria centrale alla backing store, dalla quale, in seguito, viene portato nuovamente in memoria per proseguire l'esecuzione
  - Lo spazio fisico totale di memoria allocata ai processi può eccedere la memoria fisica
- Backing store — È una partizione del disco ad accesso rapido, sufficientemente capiente da accogliere copie di tutte le immagini di memoria per tutti i processi utente; deve garantire accesso diretto a tali immagini
- Roll out, roll in — È una variante dello swapping impiegata per algoritmi di scheduling basati su priorità; processi a bassa priorità vengono riversati sulla memoria di massa, in modo tale da permettere che processi a priorità maggiore vengano caricati ed eseguiti

Swapping – 2

- La maggior parte del tempo di swap è dovuta al trasferimento di dati; il tempo totale di trasferimento è direttamente proporzionale alla quantità di memoria riversata
- Il sistema operativo mantiene una ready queue dei processi la cui immagine è stata riversata su disco e dei processi attualmente presenti in memoria
- Se il binding viene effettuato in fase d'esecuzione, il processo sottoposto a swapping può successivamente essere riversato in uno spazio di memoria diverso
  - Inoltre, occorre fare attenzione allo swapping di processi con I/O pendenti
- Molti SO attuali (i.e., UNIX, Linux, e Windows) supportano differenti versioni di swapping
  - Swapping normalmente disabilitato
  - Attivato quando è allocata una quantità di memoria superiore ad una data soglia (e, comunque, effettuato su pagine); disabilitato quando l'allocazione rientra sotto soglia



Swapping – 3



Swapping di due processi su backing store (memoria di massa)

Swapping e tempo di context–switch – 1

- Se il prossimo processo da eseguire non è in memoria, e la memoria è piena, occorre scambiare un processo attualmente in memoria con il processo target
  - ⇨Il tempo di context–switch può essere molto elevato
- Esempio: si consideri un processo da 100MB che viene "swappato" sul disco fisso ad un tasso di 50MB/sec
  - Tempo di swap–out pari a 2sec
  - Uguale tempo di swap–in del processo target
  - Tempo totale dell'operazione di swap all'interno del context– switch pari a 4sec
  - Nelle architetture attuali, tempo di context–switch $<10\mu$sec

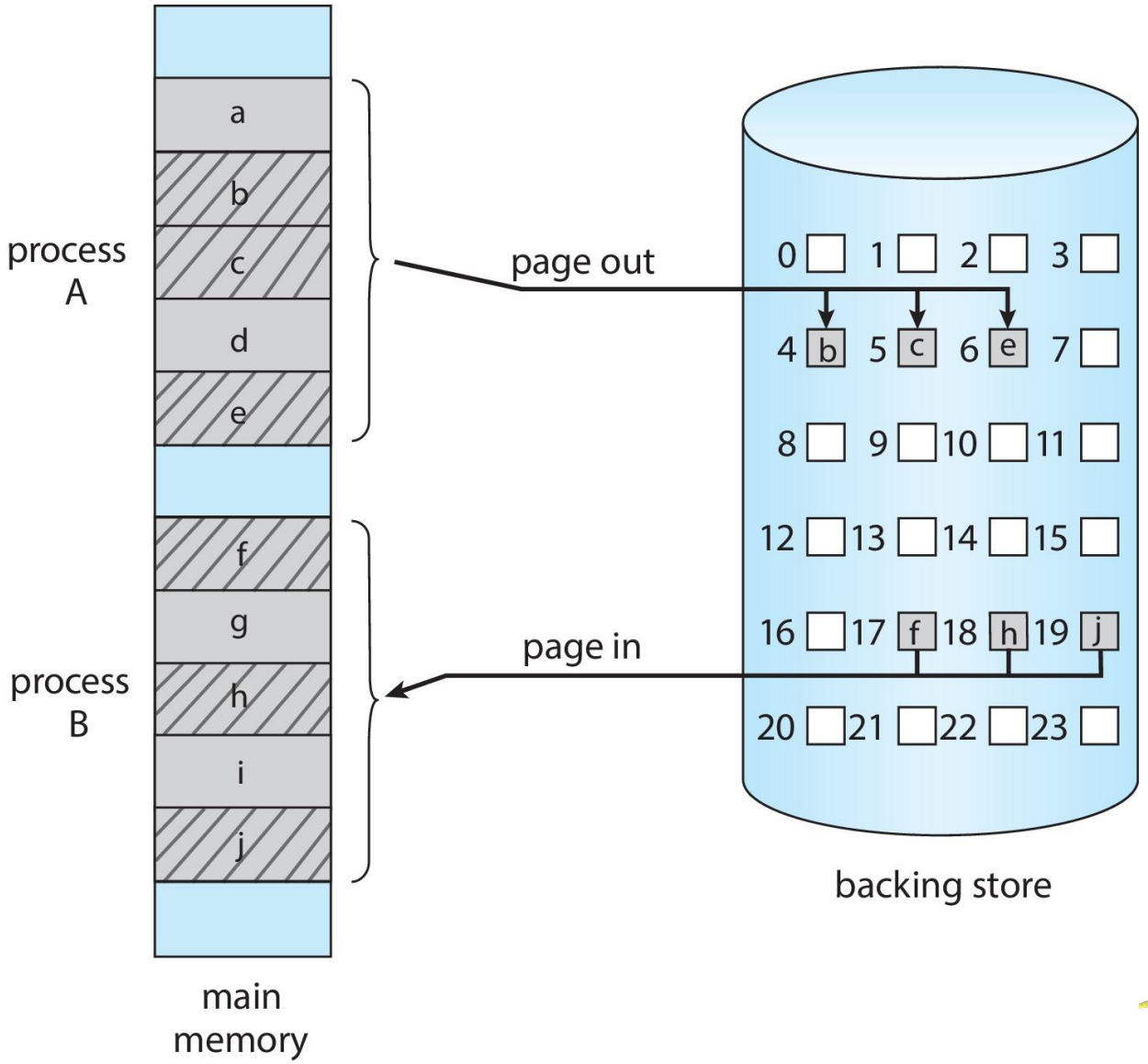
Swapping e tempo di context–switch – 2

- Inoltre, i processi con I/O pendenti non possono essere spostati su disco, salvo effettuare il trasferimento di dati dovuto all'I/O nello spazio kernel
  - Tecnica nota come double buffering, che aggiunge un overhead ulteriore al tempo di context–switch
- Le tecniche di swapping standard sono raramente implementate nei SO attuali; sono invece comuni realizzazioni customizzate
  - Si attua lo swapping solo quando la quantità di memoria disponibile è estremamente bassa

Swapping con paginazione – 1

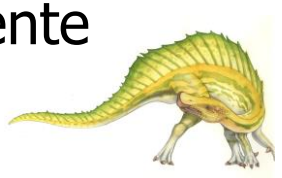
- La maggior parte dei SO, inclusi Linux e Windows, usa invece normalmente una variante dell'avvicendamento
  - Si spostano solo alcune pagine di un processo
    - Si possono sovrascrivere "pezzi" di memoria fisica
    - Costa meno in termini di tempo
  - Fondamentale per la realizzazione della memoria virtuale
- In questo caso non si parla di swapping, ma di paging
  - Un'operazione di *page out* (scaricamento della pagina) sposta una pagina dalla memoria centrale alla memoria ausiliaria
  - Il processo inverso è il *page in* (caricamento della pagina)

Swapping con paginazione – 2



Swapping nei dispositivi mobili

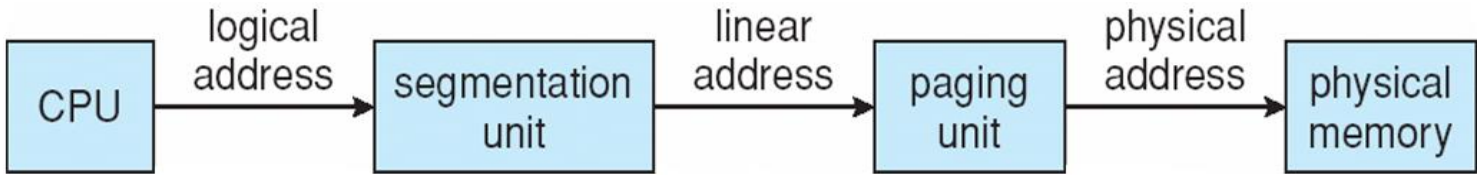
- Lo swapping non è solitamente supportato nei dispositivi mobili
  - La memoria di massa è realizzata tramite flash
    - Poco capienti
    - Accesso lento (bassa velocità di trasferimento fra memoria centrale e memoria flash) e rischio di "memory leak"
- Metodi diversi per liberare la memoria a fronte di sovraccarichi
  - iOS "chiede" alle applicazioni di rinunciare volontariamente alla memoria allocata
    - I dati di sola lettura vengono rimossi dal sistema e, se necessario, successivamente ricaricati dalla memoria flash
    - Terminazione forzata delle app che non riescono a liberare memoria
  - Android termina le app qualora la memoria libera disponibile non sia sufficiente, ma scrive lo stato dell'applicazione nella memoria flash, per riavviarle rapidamente
  - Entrambi i SO supportano invece la paginazione



Appendice

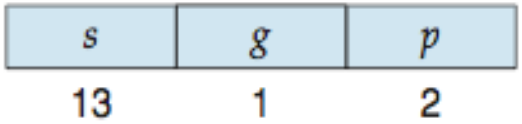
Case–study: le architetture Intel a 32/64 bit

- Le architetture Intel hanno dominato il mondo dei personal computer fino dagli anni ’70
  - Primo PC IBM: Intel 8088 (16 bit, memoria segmentata)
  - La CPU Pentium a 32 bit è chiamata anche IA–32 architecture
  - Le CPU attuali sono a 64 bit e sono chiamate X86–64 (AMD) o IA–64 (Intel)
- In particolare, il Pentium supporta la segmentazione mista a paginazione
  - La CPU genera **indirizzi logici**, che confluiscono nell’unità di segmentazione, la quale produce **indirizzi lineari**
  - L’**indirizzo lineare** passa all’unità di paginazione, che a sua volta genera l’**indirizzo fisico** relativo alla memoria centrale
  - Le unità di segmentazione e di paginazione formano un equivalente dell’unità di gestione della memoria (MMU)

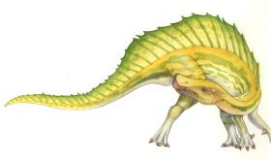


L'architettura Intel a 32 bit

- Segmenti lunghi al più 4GB; fino a 16K segmenti per ogni processo
- Spazio degli indirizzi logici diviso in due partizioni
  - 8K segmenti riservati (informazioni nella *Local Descriptor Table*, LDT)
  - 8K segmenti condivisi (informazioni nella *Global Descriptor Table*, GDT)
  - Ogni elemento (in LDT o GDT) è costituito da 8 byte e contiene informazioni sul segmento (oltre a indirizzo base e limite)
- Un indirizzo logico è una coppia <*selettore*,*offset*>, dove il selettore è un numero a 16 bit

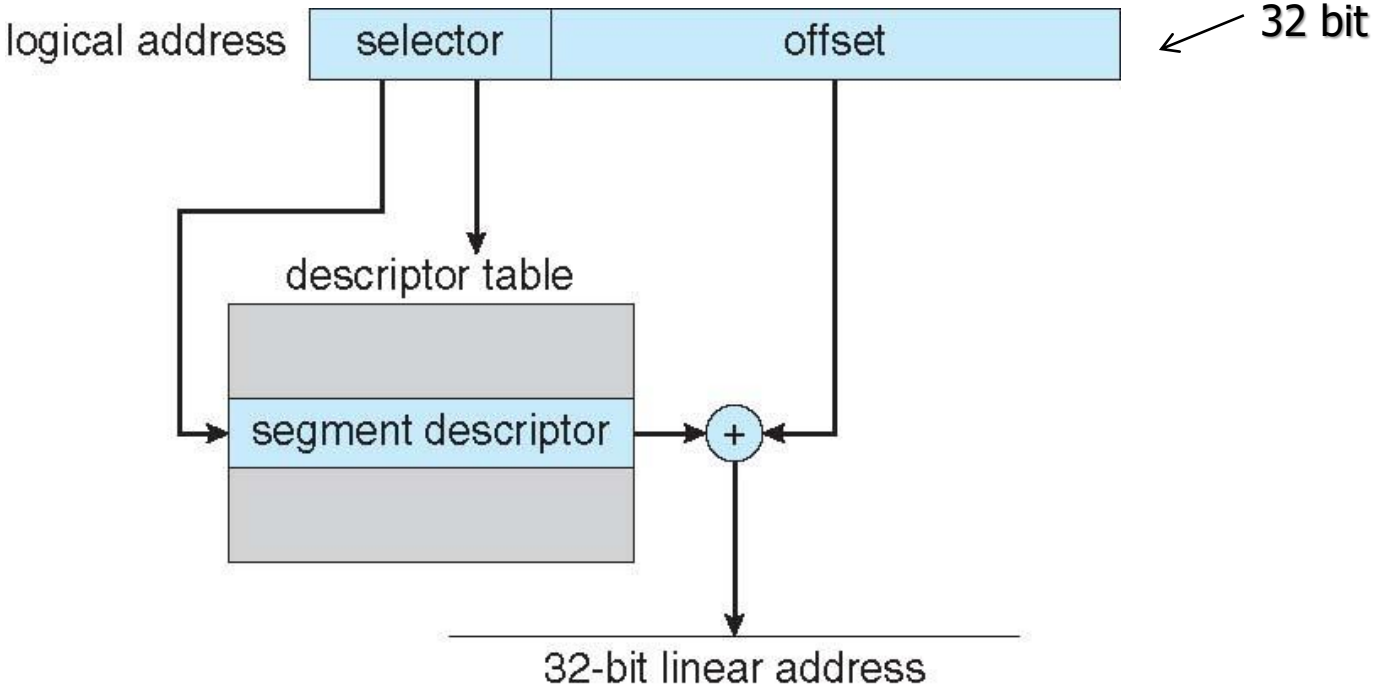


con s numero del segmento (13 bit), g indicatore di tipo (in LDT o GDT) e p contenente due bit di protezione



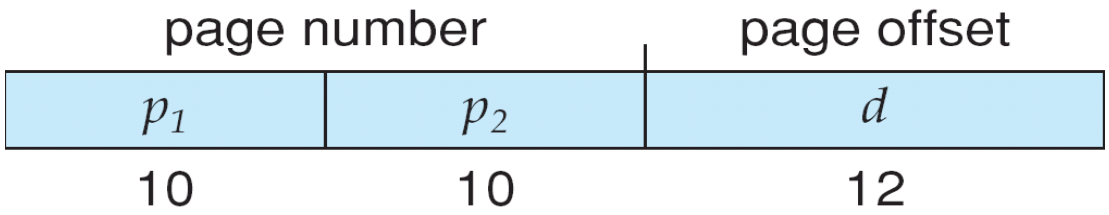
Segmentazione nell'IA–32

- Un indirizzo lineare del Pentium è lungo 32 bit e si genera come segue:
  - Il selettore punta all'elemento appropriato nella LDT o nella GDT
  - Le informazioni relative alla base ed al limite di tale segmento vengono utilizzate per generare l'indirizzo lineare (o, eventual-mente, per segnalare un errore)

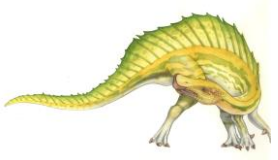


Paginazione nell'IA–32 – 1

- L'architettura Pentium prevede che le pagine abbiano una dimensione di 4KB o 4MB
  - Per le prime, vige uno schema di paginazione a due livelli:



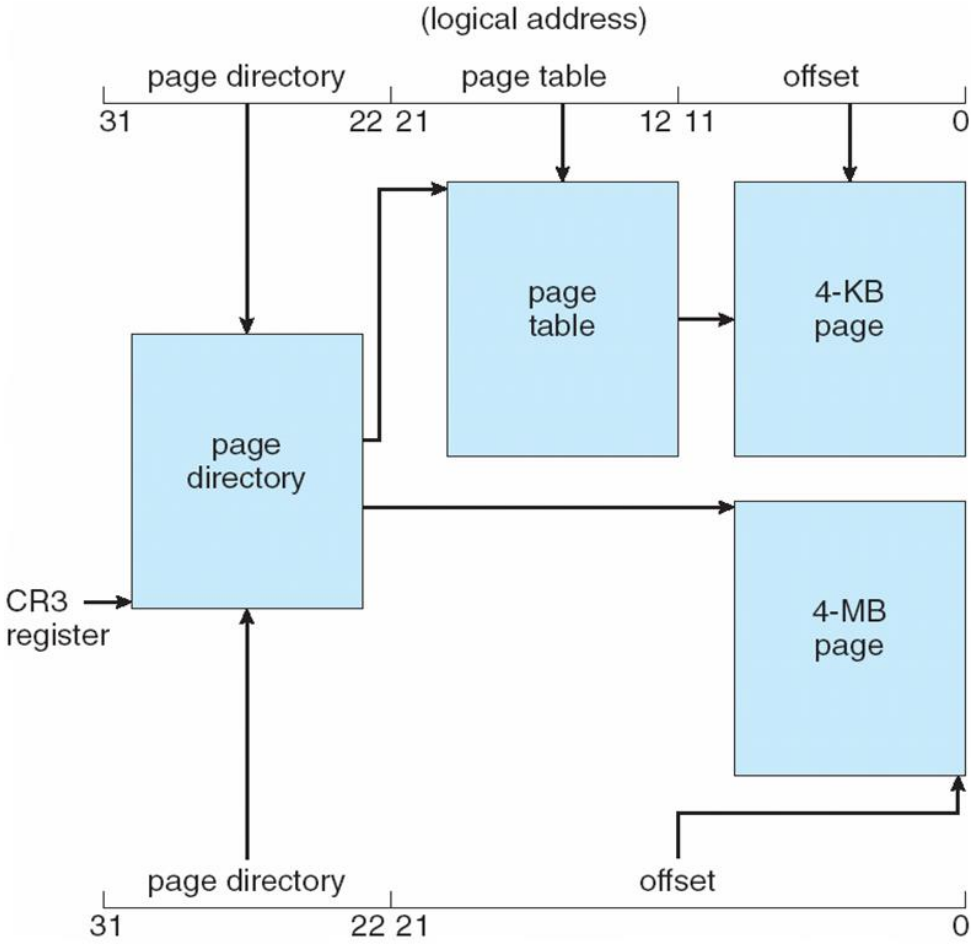
  - I 10 bit più significativi puntano alla tabella delle pagine esterna, detta directory delle pagine
- Se la pagina ha invece dimensione 4MB, la directory di pagina punta direttamente al frame corrispondente (senza considerare la tabella delle pagine interna – si utilizzano 22 bit per l'offset)



Paginazione nell'IA–32 – 2

- Le tabelle delle pagine possono essere trasferite su disco:
  - Invalid bit nella directory delle pagine
  - Gli altri 31 bit possono essere utilizzati per localizzare la tabella su disco

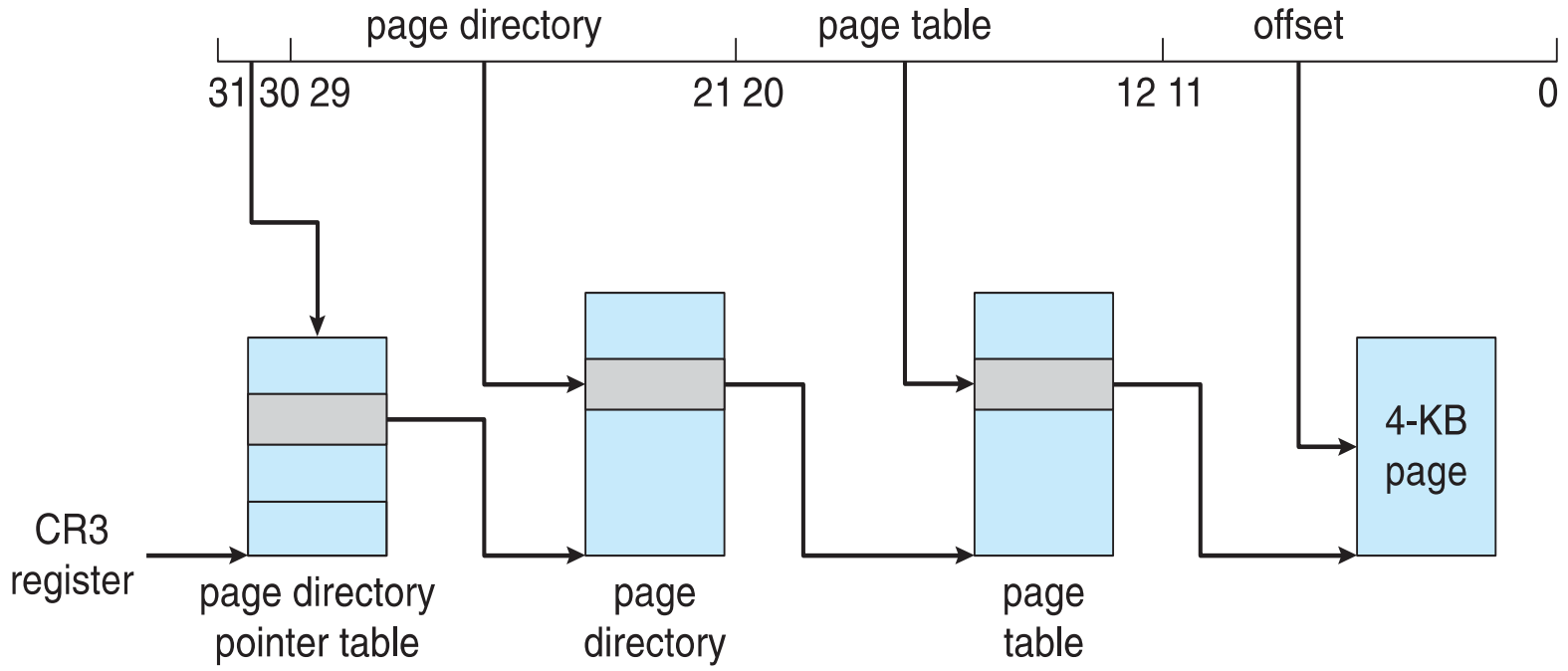
Il registro CR3 punta alla directory delle pagine del processo corrente



Paginazione nell'IA–32 – 3

- Il limite degli indirizzi a 32 bit portò l'Intel a sviluppare l'estensione di indirizzo della pagina (PAE, page address extension), per permettere alle applicazioni di accedere a più di 4GB di spazio di memoria
  - Schema di paginazione a 3 livelli
  - I due bit più significativi si riferiscono a una tabella dei puntatori alla directory delle pagine
  - Gli elementi della directory delle pagine e della tabella delle pagine sono a 64 bit (si utilizzano 24 bit per gli indirizzi base dei frame)
  - L'effetto finale è un aumento dello spazio degli indirizzi a 36 bit, garantendo il supporto massimo di 64 GB di memoria fisica

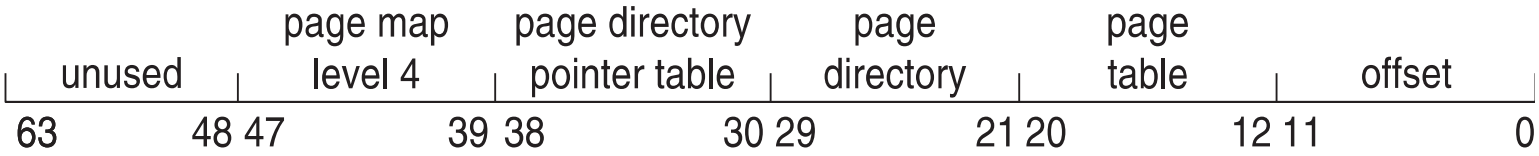
Paginazione nell'IA–32 – 4



Estensione degli indirizzi di pagina

Intel x86–64

- Le architetture a 64 bit rappresentano la generazione attuale dei microprocessori Intel (AMD)
- 64 bit è "ginormous" (16 exabyte)
- Nella pratica, implementano una modalità di indirizzamento a 48 bit
  - Dimensione delle pagine di 4KB, 2MB, 1GB
  - Paginazione gerarchica a 4 livelli
- Possono utilizzare anche la PAE, cosicché si ottengono indirizzi virtuali a 48 bit e indirizzi fisici a 52 bit

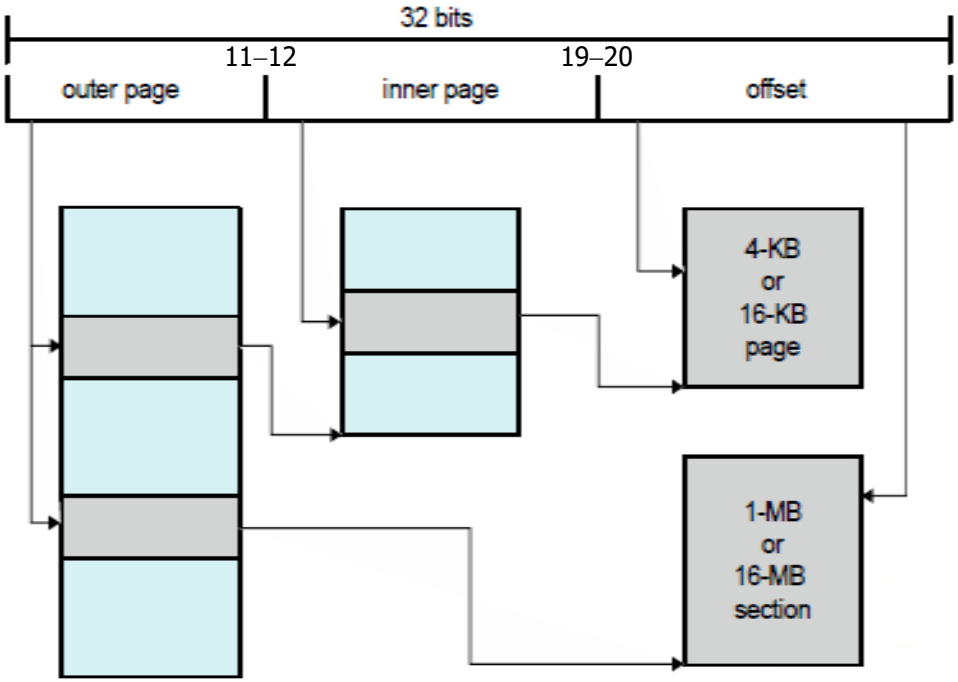


Case–study: l'architettura ARM – 1

- Tecnologia dominante (almeno fino a tempi recenti) per le piattaforme mobili — per esempio, nei dispositivi Apple iOS e Google Android, dove ha coperto fino al 95% del mercato
- CPU 64–bit di nuova generazione, particolarmente efficienti dal punto di vista del risparmio energetico
- Supportano diversi formati di pagina (pagine da 4 KB 16 KB e 64 KB, oppure pagine da 2 MB, 32 MB e 512 MB, chiamate sezioni)
- Paginazione a tre livelli per le sezioni, a quattro livelli per le pagine di dimensione minore

Case–study: l'architettura ARM – 2

- Due livelli di TLB
  - Due micro TLB interne, una per i dati ed una per le istruzioni (con ASID)
  - Una TLB principale
  - La traduzione di un indirizzo inizia a livello micro TLB; in caso di insuccesso, viene controllata la TLB principale; in caso di ulteriore insuccesso ci si rivolge alla tabella delle pagine



Fine del Capitolo 9