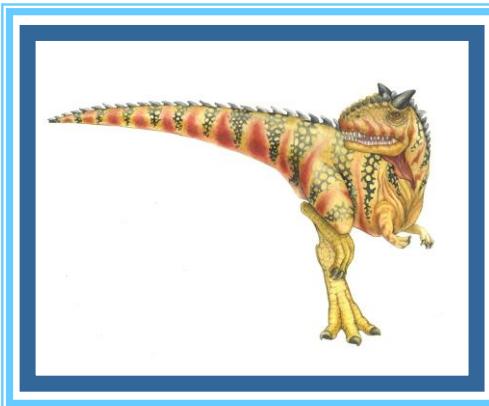
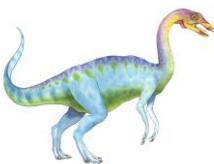


Memoria centrale

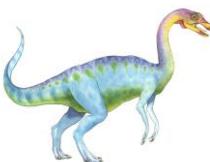




Background – 1

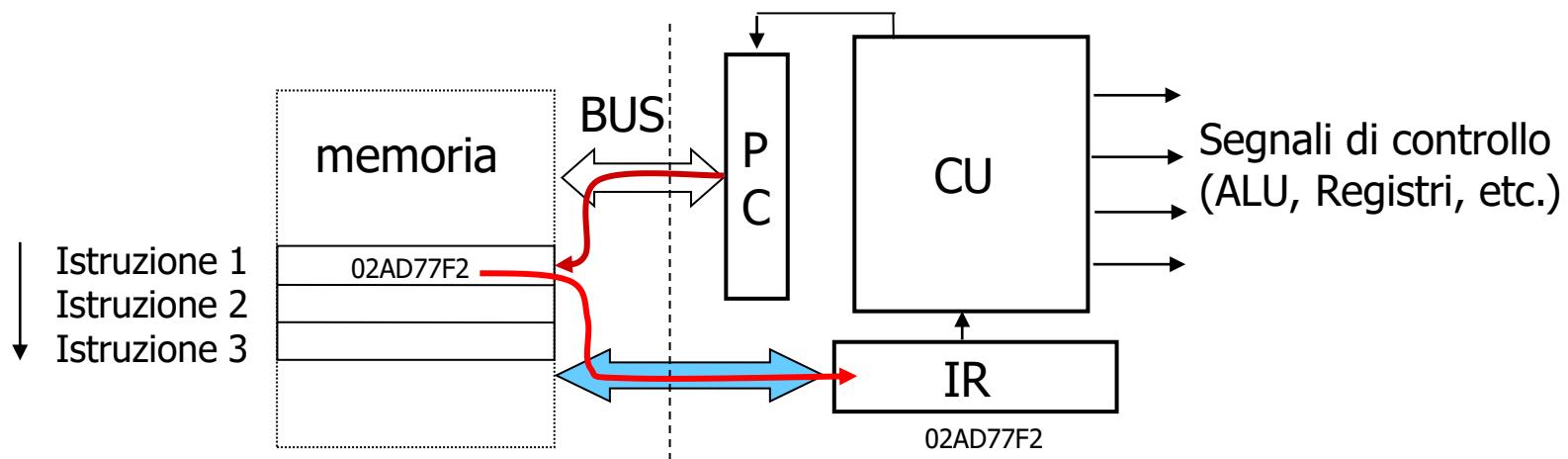
- ❖ Lo scheduling della CPU:
 - Migliora l'utilizzo della risorsa CPU
 - Migliora la "reattività" del sistema nei confronti degli utenti (minor tempo di risposta, maggiore throughput)
- ❖ Per poter essere eseguiti, i programmi devono essere copiati (almeno parzialmente) in memoria centrale, per "trasformarsi" 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



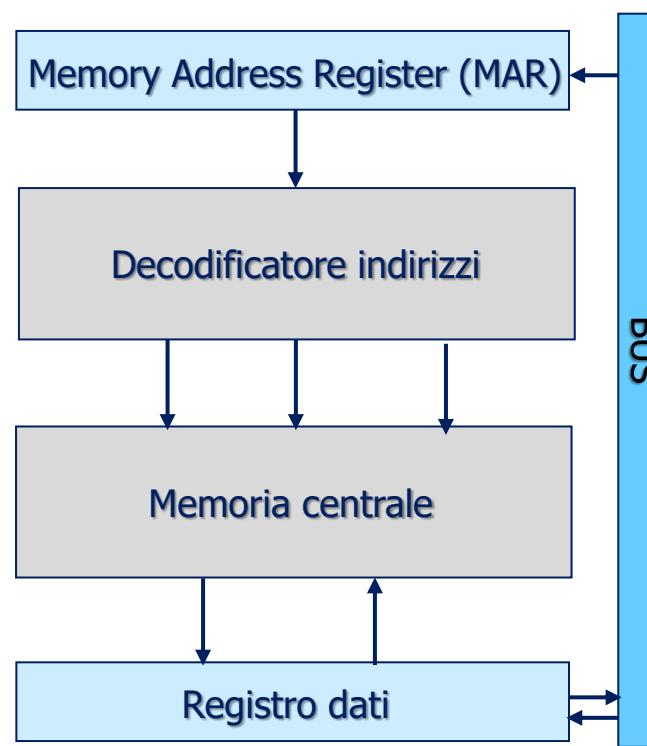


Background – 3

- ❖ 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

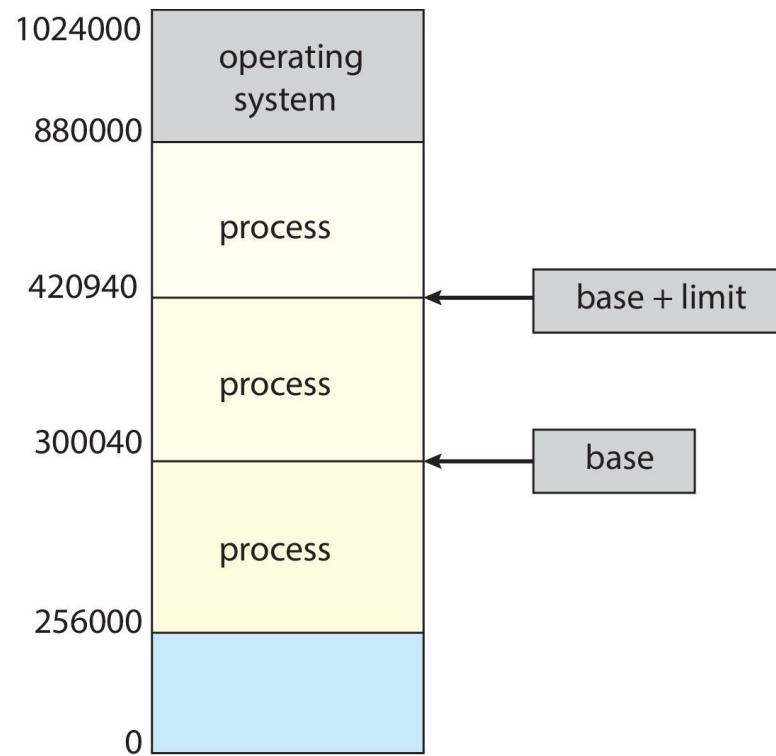
- ❖ La memoria principale e i registri sono gli unici dispositivi di memoria cui la CPU può accedere direttamente
- ❖ I registri vengono acceduti in un ciclo di clock (o meno)
- ❖ Un accesso alla memoria centrale può invece richiedere diversi cicli di clock (occorre “passare” dal bus)
 - ⇒ Possibile 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 ed 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



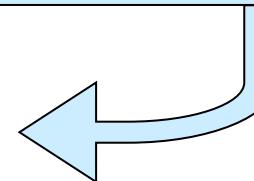


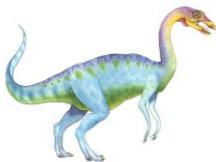
Esempio: registri base e limite – 1

- ❖ 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



Il **registro base** contiene il più piccolo indirizzo legale della memoria fisica allocata al processo; il **registro limite** determina la dimensione dello spazio di memoria





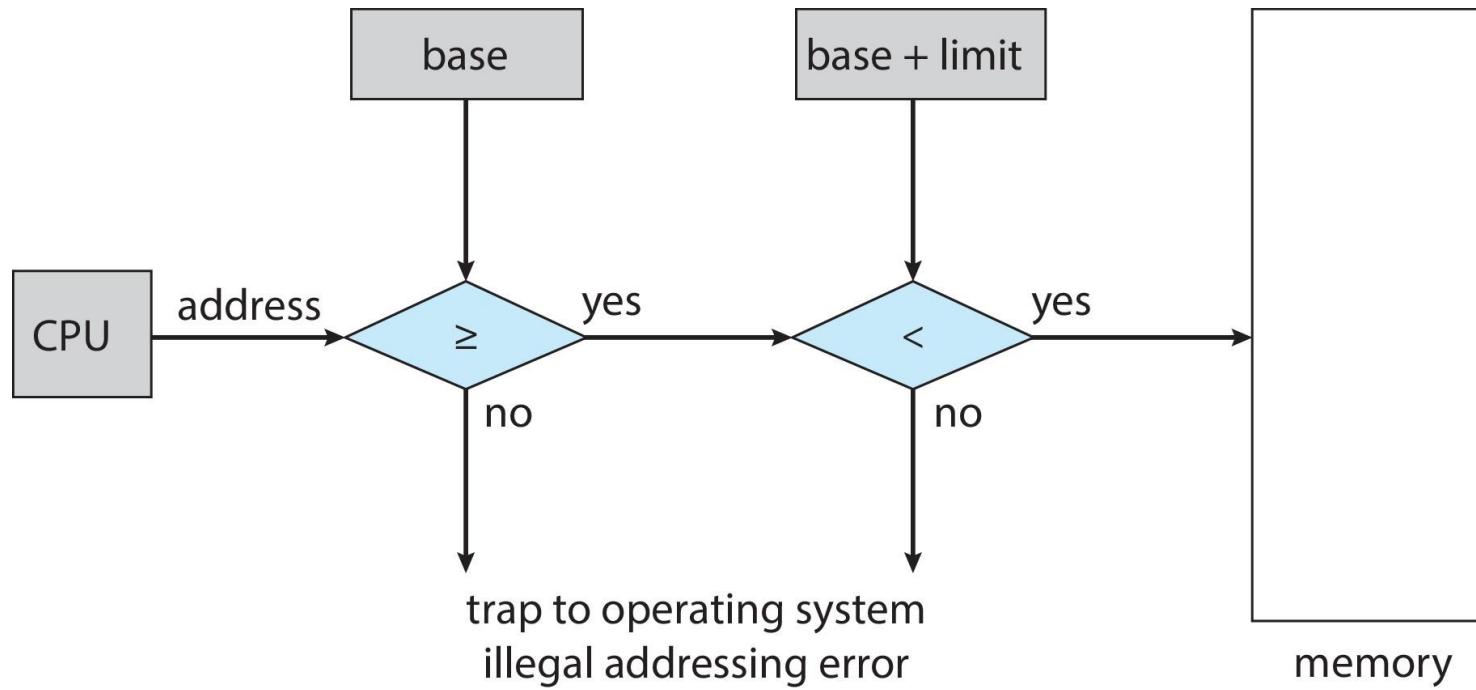
Esempio: registri base e limite – 2

- ❖ Per mettere in atto il meccanismo di protezione, la CPU confronta ogni indirizzo generato dal processo utente con i valori contenuti nei due registri
 - $\text{indirizzo} \geq \text{base}$
 - $\text{indirizzo} < \text{base+limite}$
- ❖ 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



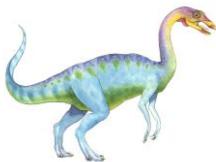


Esempio: registri base e limite – 3



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

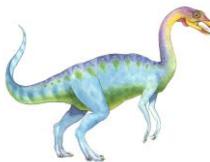




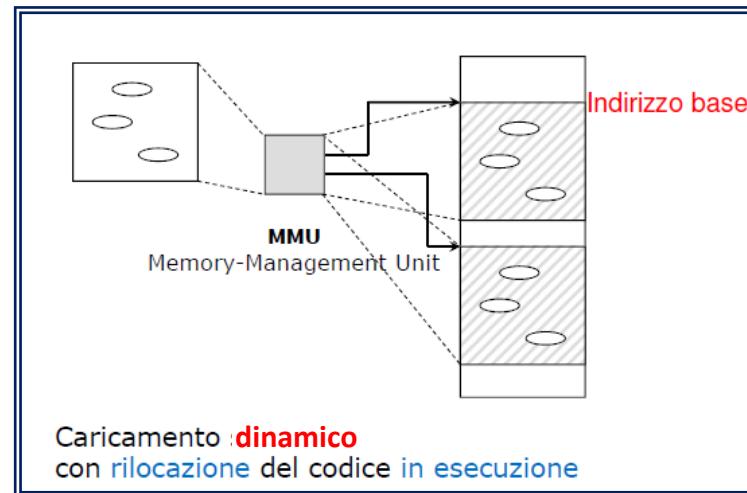
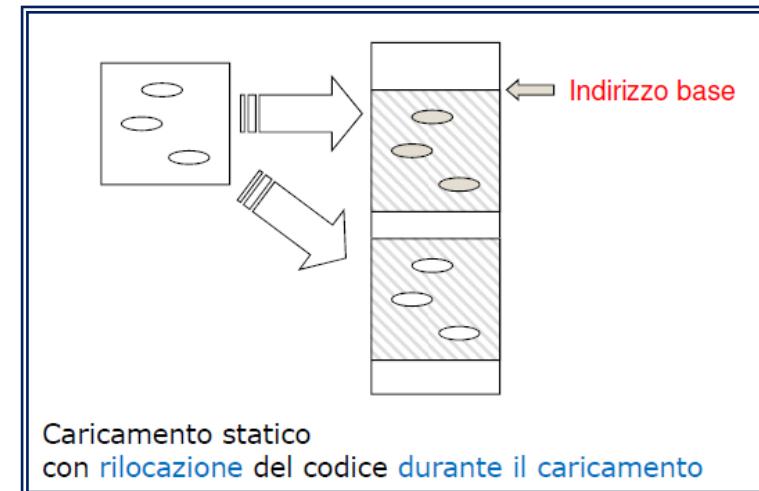
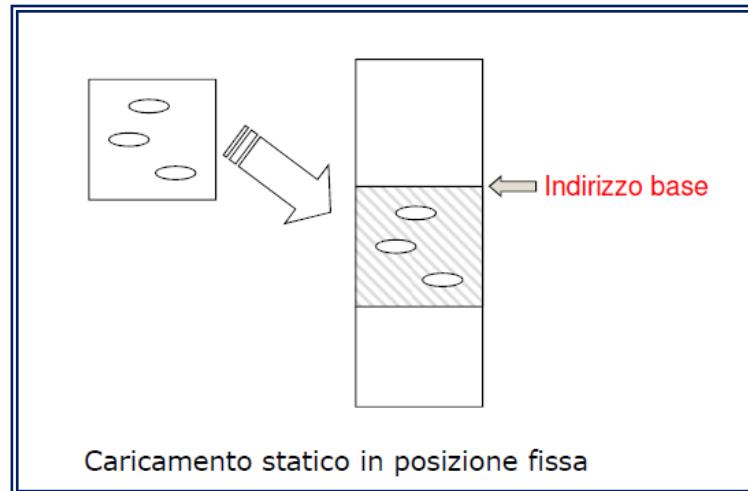
Associazione degli indirizzi – 5

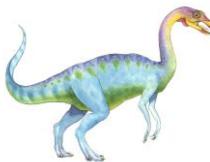
- ❖ La CPU genera **indirizzi logici**, ma l'unità di memoria "vede" **indirizzi fisici**
- ❖ Il binding mappa indirizzi logici in indirizzi fisici
 - **Binding statico** (a tempo di compilazione o di caricamento)
 - ⇒ **indirizzi logici = indirizzi fisici**
 - **Binding dinamico** (a run-time)
 - ⇒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



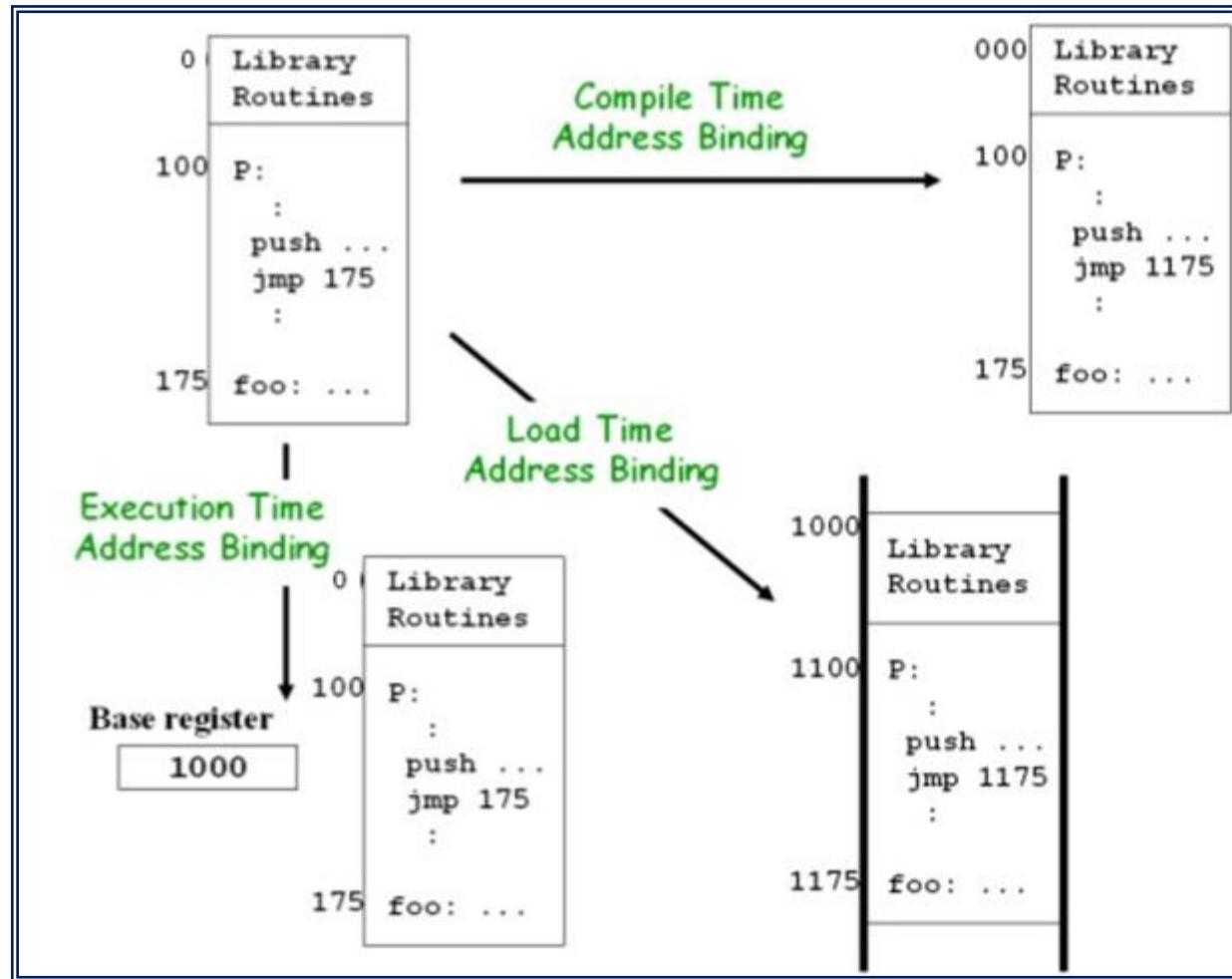


Associazione degli indirizzi – 6





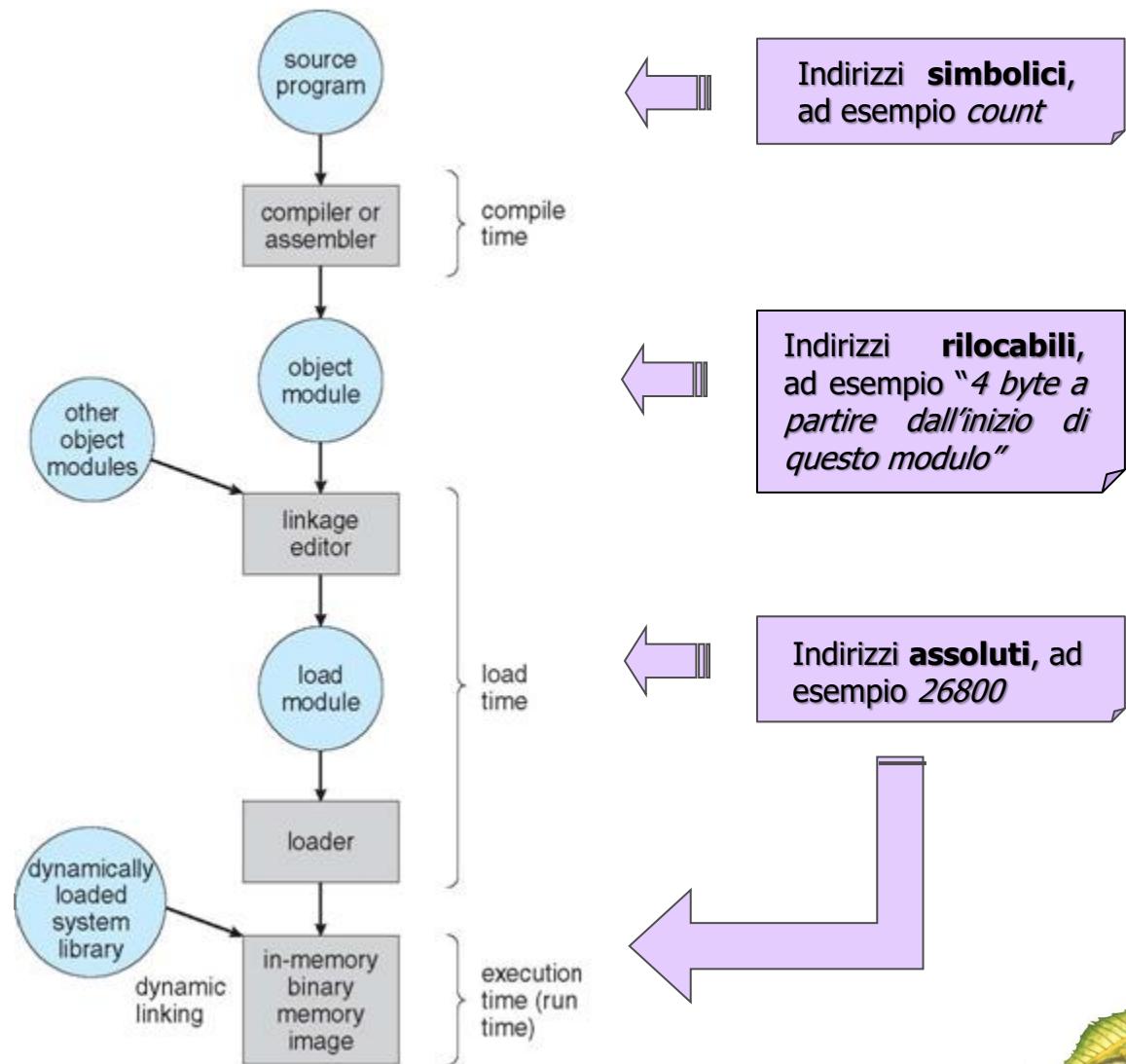
Associazione degli indirizzi – 7





Elaborazione multistep dei programmi utente

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

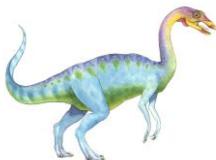




Allocazione contigua

- ❖ 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

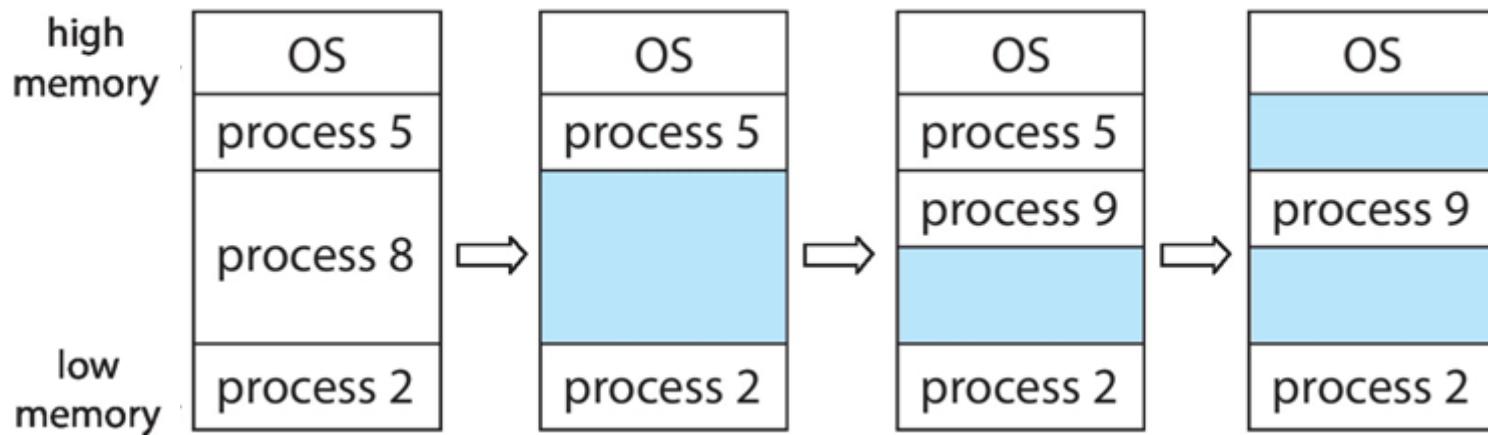
- ❖ **Partizioni variabili** (MVT – *Multiprogramming with a Variable number of Tasks*): ogni partizione è allocata dinamicamente in base alla dimensione del processo da allocare
 - Utilizzato in ambienti batch o time-sharing con gestione della memoria tramite segmentazione semplice
 - **Vantaggi** (vs 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**

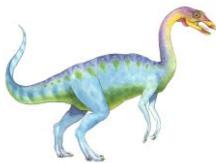




Partizioni multiple – 3

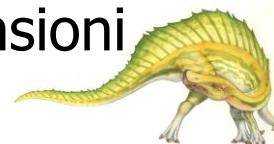
- ❖ Un buco – *hole* – è un blocco di memoria disponibile; nella memoria sono sparsi buchi di varie dimensioni
- ❖ Quando viene caricato un nuovo processo, gli viene allocato un buco grande abbastanza da contenerlo
- ❖ Quando un processo termina, libera la memoria allocata; partizioni libere contigue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

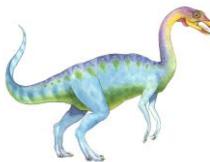




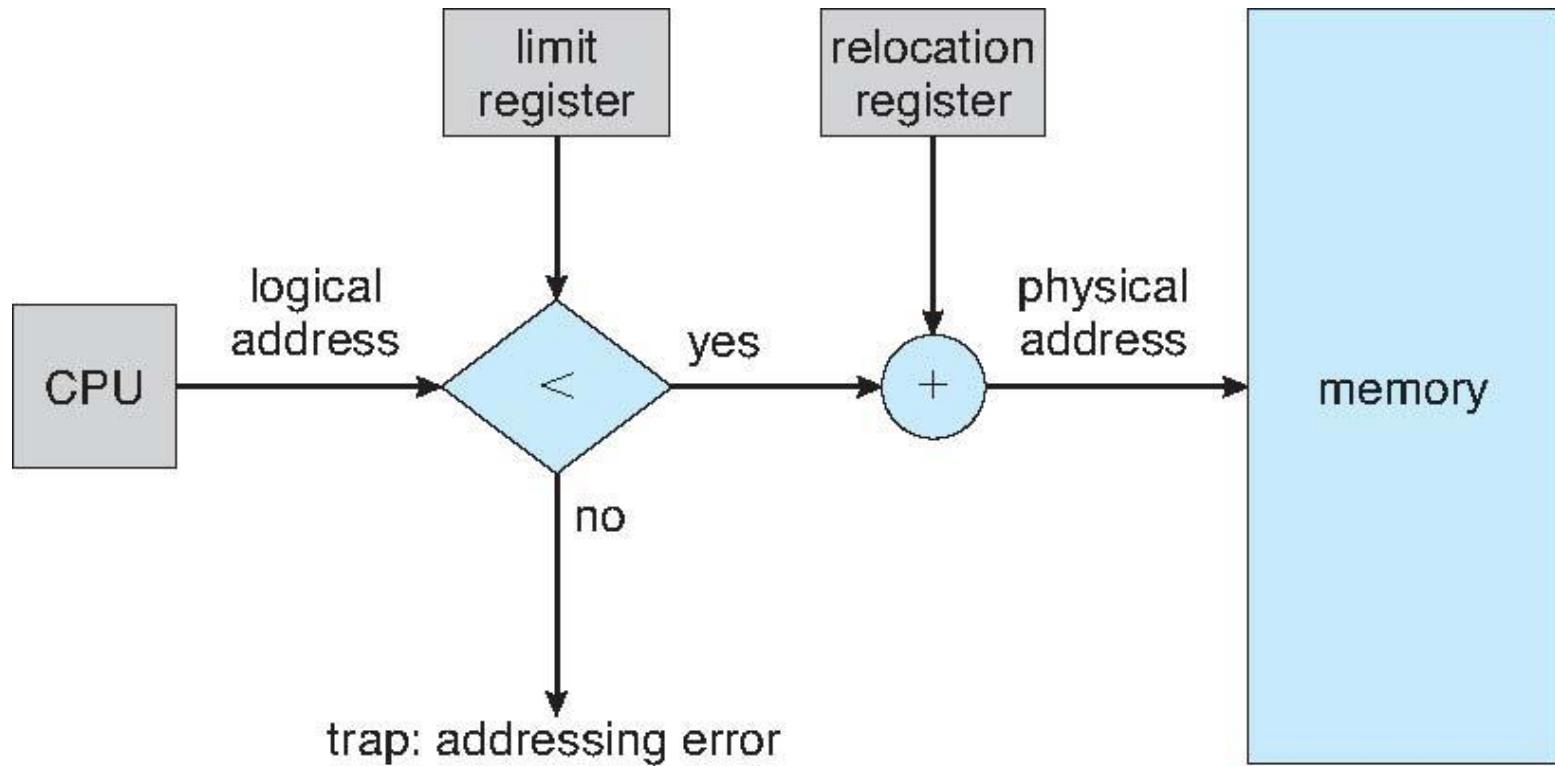
Allocazione contigua (cont.)

- ❖ 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





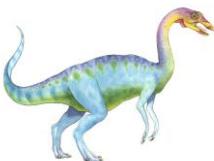
Esempio

- ❖ 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

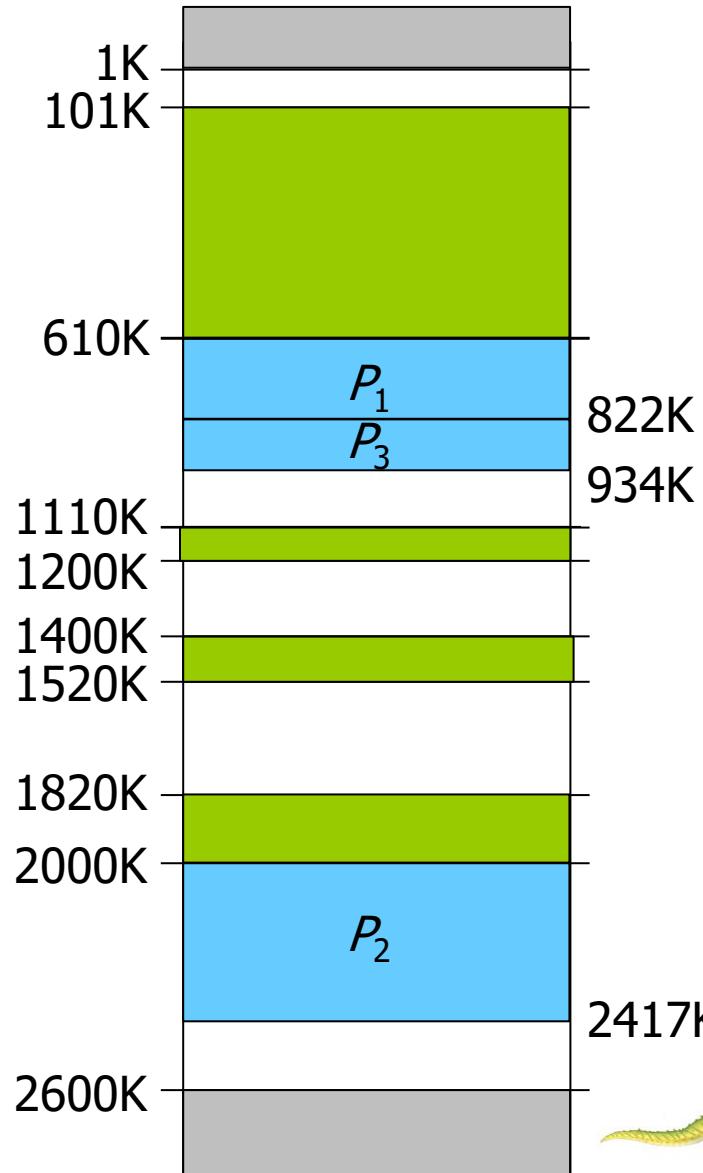
Nodi	Dimensioni	Indirizzo
N_1	100K	1K
N_2	500K	610K
N_3	200K	1200K
N_4	300K	1520K
N_5	600K	2000K

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

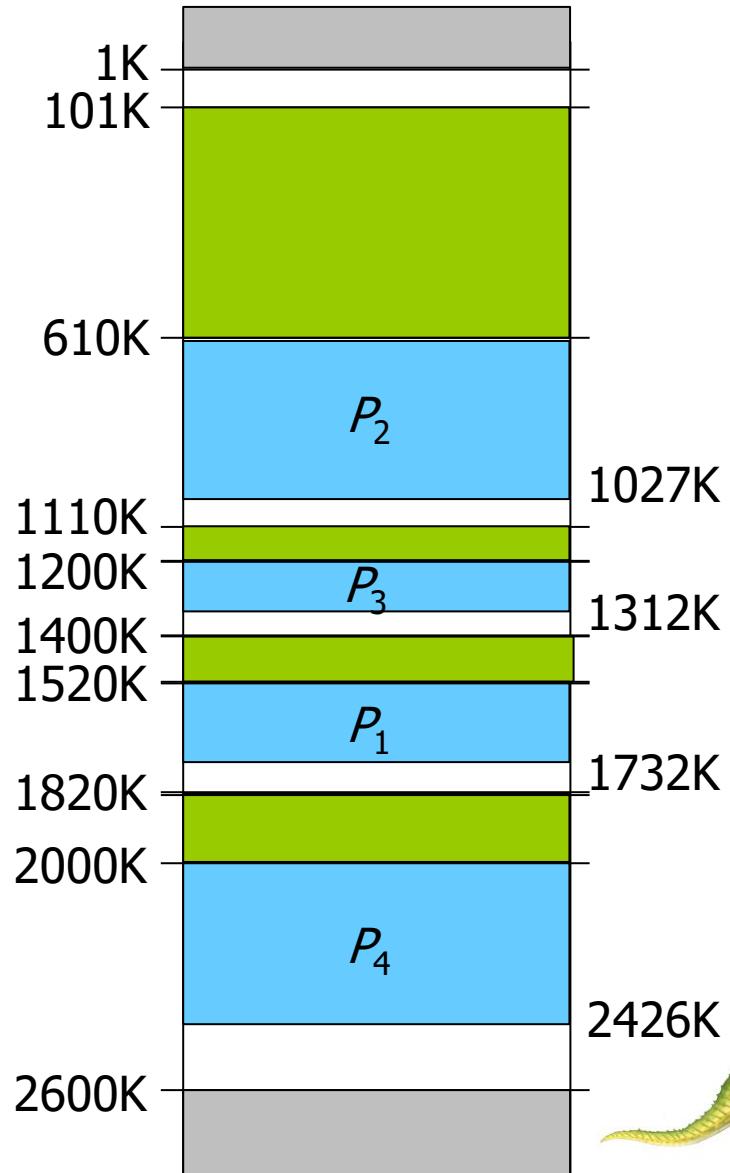
Nodi	Dimensioni	Indirizzo
N_1	100K	1K
N_2	500K	610K
N_3	200K	1200K
N_4	300K	1520K
N_5	600K	2000K

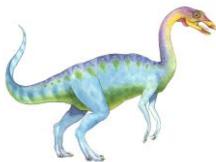
BEST-FIT

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

Frammentazione esterna:

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

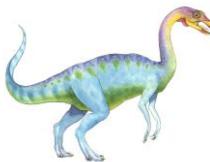




Paginazione – 1

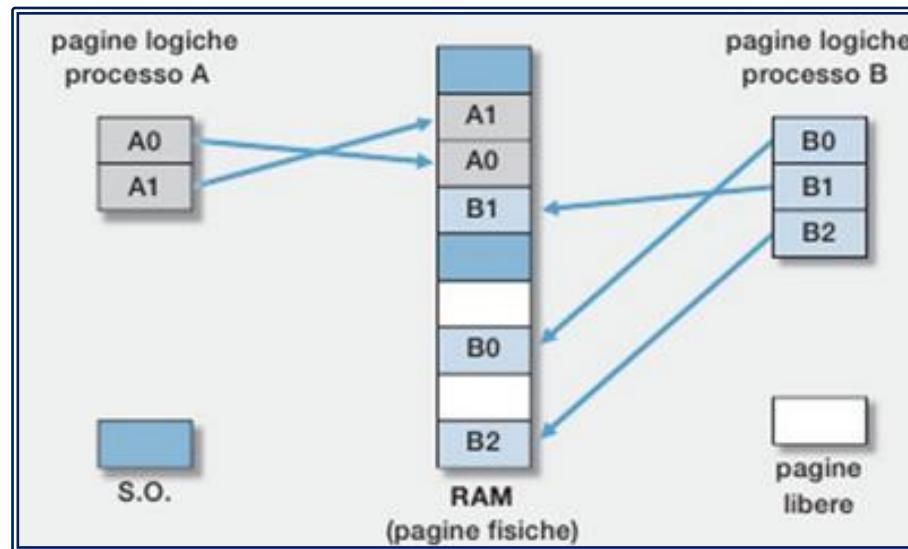
- ❖ Con la paginazione, lo spazio degli indirizzi fisici di un processo può essere non contiguo
 - Al processo è allocata memoria fisica ogni volta che quest'ultima si rende disponibile
 - Evita la frammentazione esterna
 - Evita il problema di blocchi di memoria di dimensioni variabili
- ❖ In effetti, 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

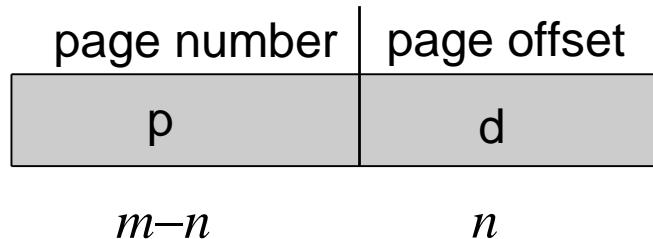
- ❖ Si ottiene uno spazio degli indirizzi logici totalmente separato dallo spazio degli indirizzi fisici (e in generale di dimensione diversa)
- ❖ Per eseguire un programma di dimensione n pagine, è necessario trovare n frame liberi prima di caricare il programma
- ❖ Si impiega una **tabella delle pagine** 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

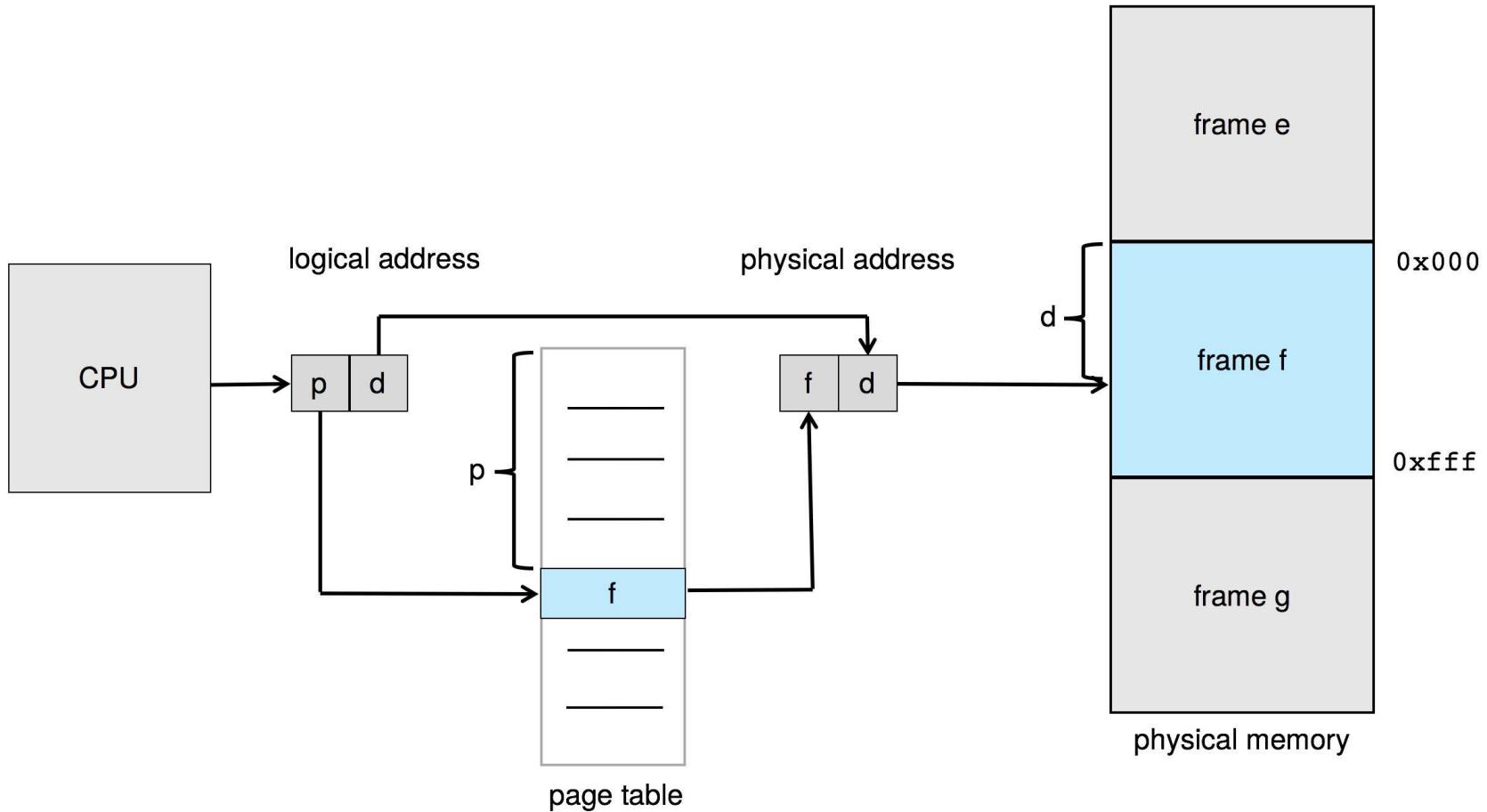


Spazio logico di 2^m indirizzi con 2^{m-n} pagine
di dimensione 2^n



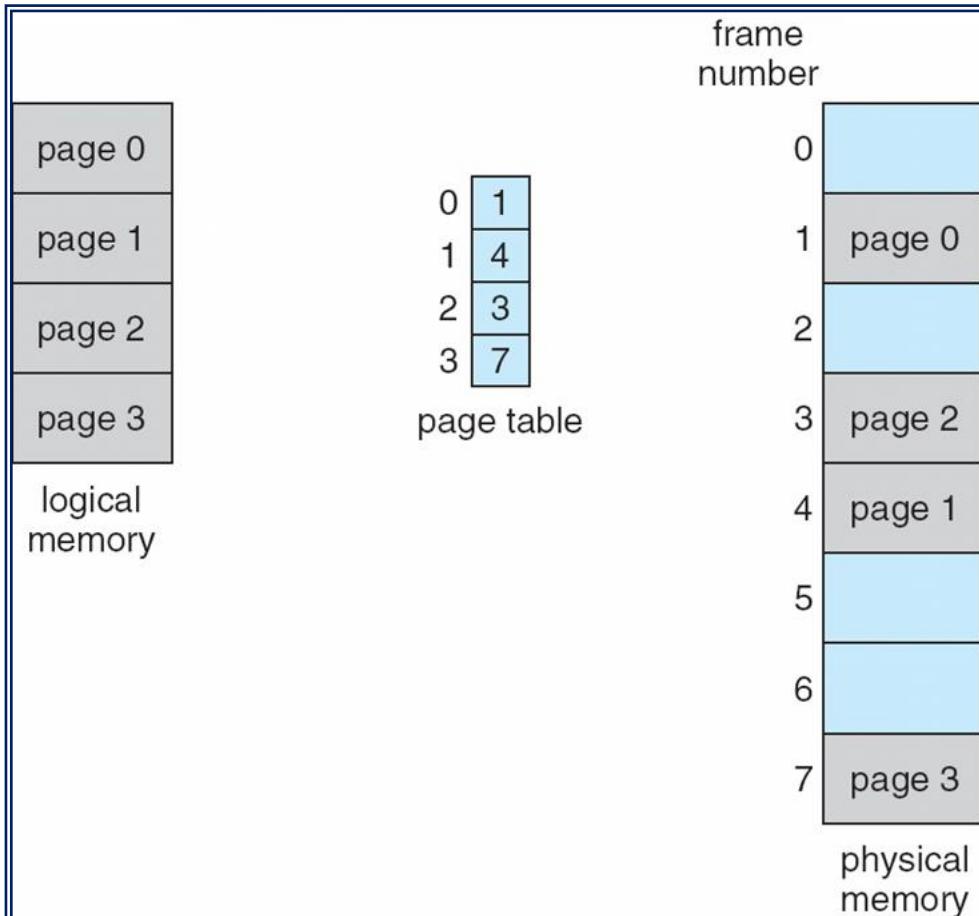


Supporto hardware alla paginazione

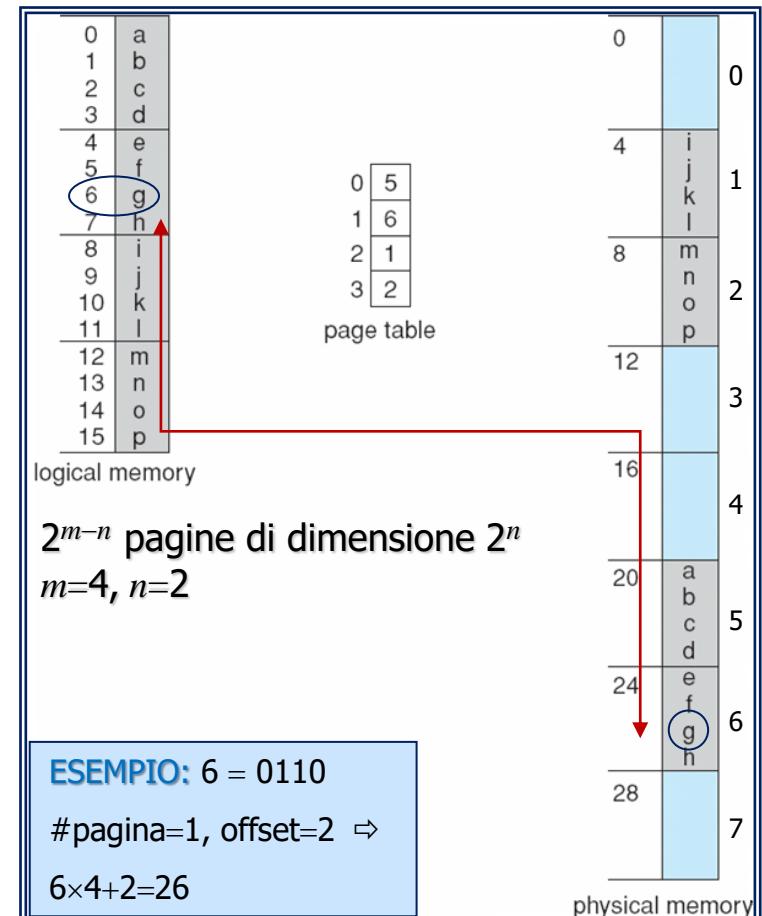




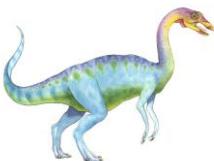
Esempi di paginazione



Modello di paginazione di memoria logica e memoria fisica



Memoria fisica da 32 byte con pagine da 4 byte



Ancora sulla paginazione

❖ Esempio

- Dimensione della pagina: 2048 byte
- Dimensione del processo: 72766 byte
 - ▶ 35 pagine + 1086 byte \Rightarrow 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 10 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



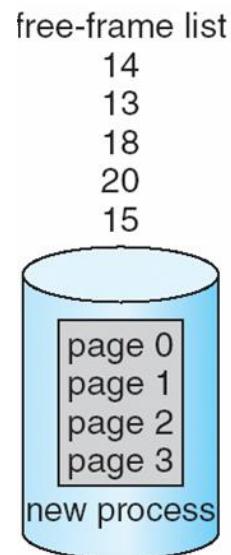
Tabella dei frame

- ❖ 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)



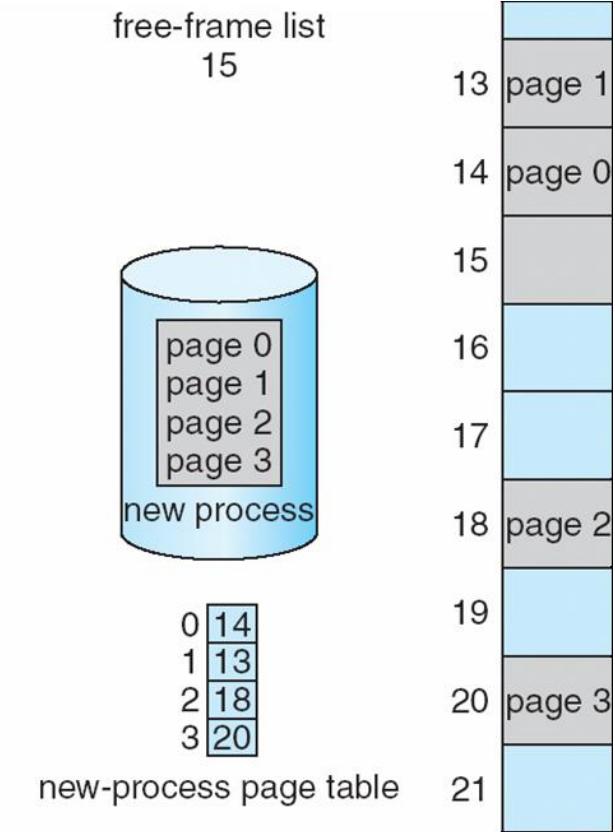
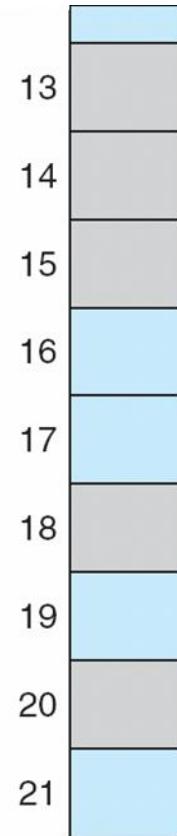


Frame liberi



(a)

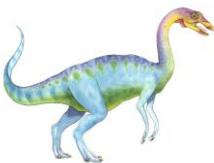
Prima dell'allocazione



(b)

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 della CPU per impostare l'hardware di paginazione quando a un processo sta per essere assegnata la CPU \Rightarrow 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

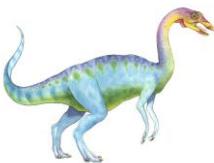




Implementazione della tabella delle pagine – 1

- ❖ 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
- ❖ 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

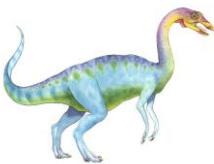




Swapping – 1

- ❖ 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





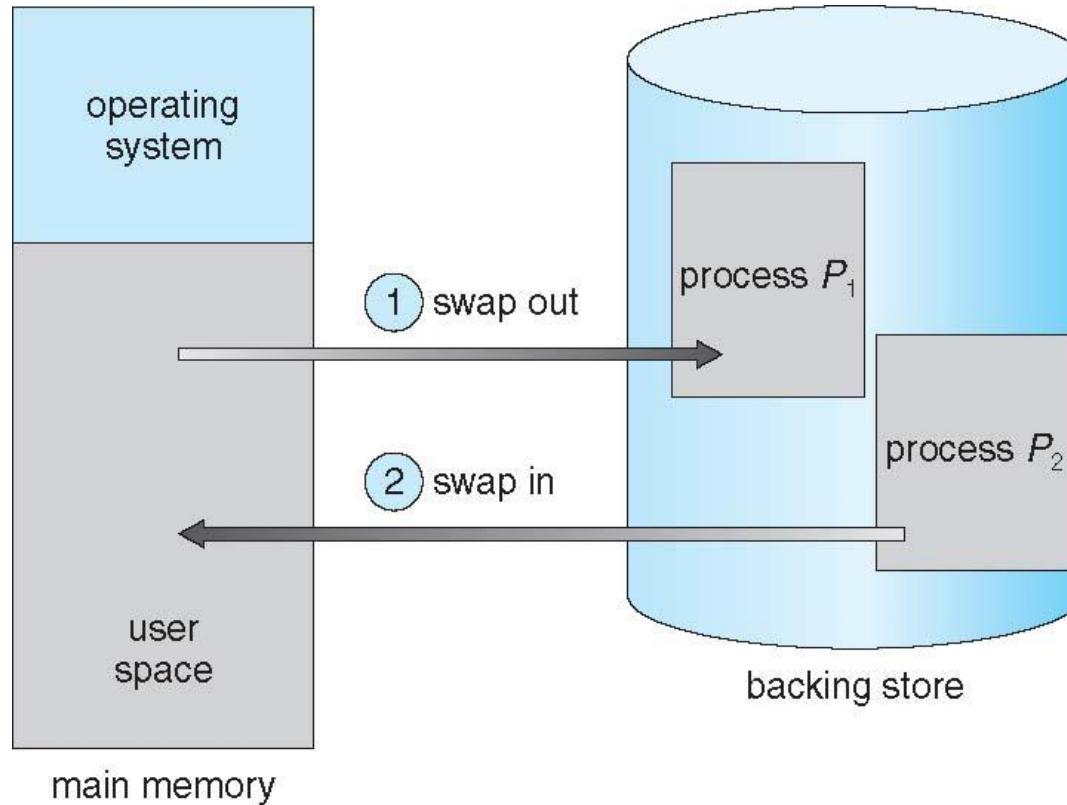
Swapping – 2

- ❖ 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; disabilitato quando l'allocazione rientra sotto soglia



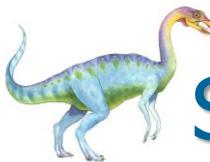


Swapping – 3



Swapping di due processi su backing store (disco)

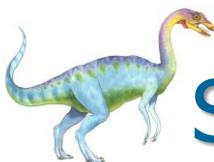




Swapping e tempo di context-switch – 1

- ❖ 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 μ 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

