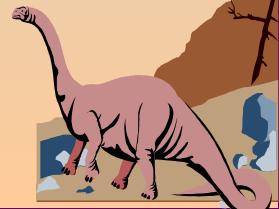
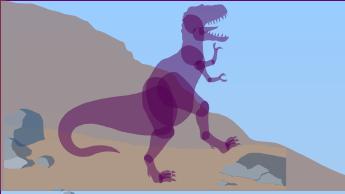




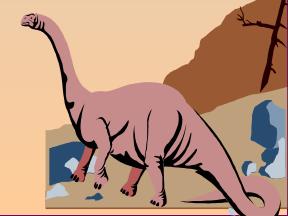
# Capitolo 8: Gestione della memoria

- ✓ Introduzione
- ✓ Avvicendamento dei processi (swapping)
- ✓ Allocazione contigua della memoria
- ✓ Paginazione
- ✓ Struttura della tabella delle pagine
- ✓ Segmentazione
- ✓ Un esempio: Pentium Intel



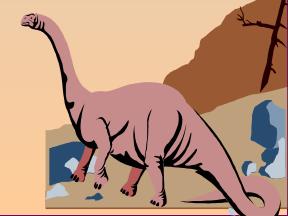
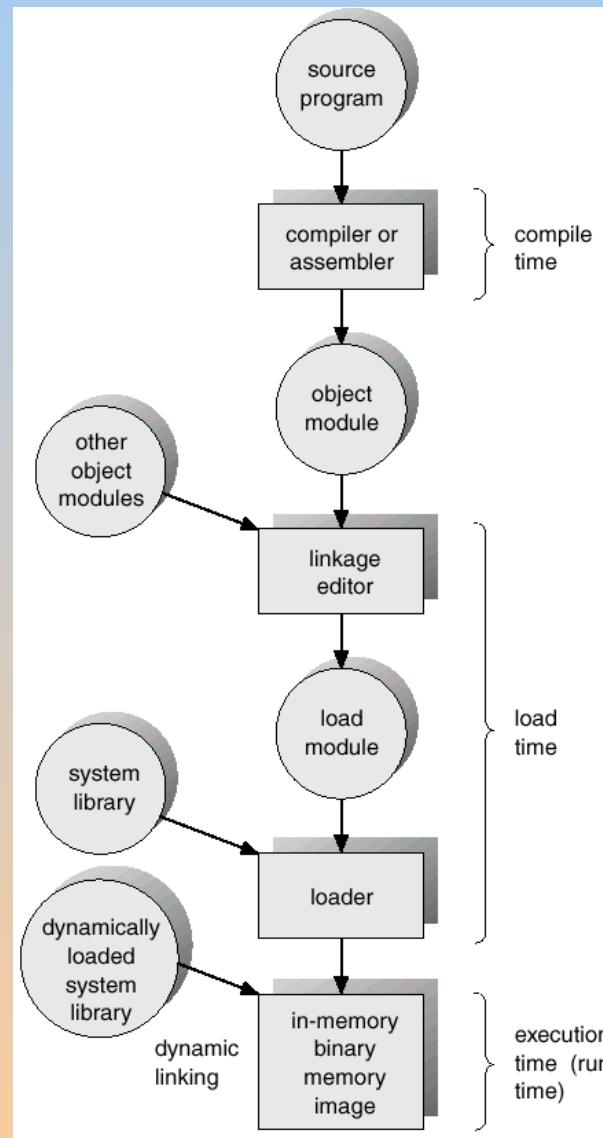


# Introduzione

- ✓ Un programma per essere eseguito deve essere caricato nella memoria ed inserito all'interno di un processo.
  - ✓ Durante la propria esecuzione un processo deve risiedere, almeno parzialmente, in memoria centrale.
  - ✓ Un computer moderno deve essere in grado di mantenere contemporaneamente più processi in memoria.
  - ✓ *Coda d'ingresso* – l'insieme dei processi presenti nei dischi e che attendono di essere trasferiti nella memoria per essere eseguiti
  - ✓ Un programma utente, prima di essere eseguito, passa attraverso vari stati in cui gli indirizzi possono essere rappresentati in modi diversi.
- 



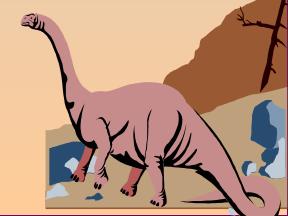
# Fasi di elaborazione per un programma utente





# Associazione di istruzioni e dati ad indirizzi di memoria

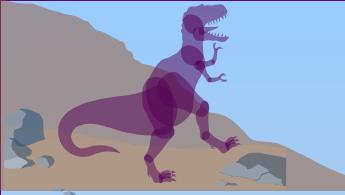
- ✓ La procedura normale consiste nel selezionare uno dei processi dalla coda di input e caricarlo in memoria.
- ✓ Durante la propria esecuzione il processo può accedere a istruzioni e dati in memoria.
- ✓ Quando il processo termina la memoria, che gli era stata allocata, viene deallocata.
- ✓ La maggior parte dei sistemi consente ai processi utenti di risiedere in qualsiasi parte della memoria fisica.
- ✓ Generalmente gli indirizzi del programma sorgente sono simbolici.
- ✓ Un compilatore di solito associa (**bind**) questi indirizzi simbolici a indirizzi rilocabili. Il linkage editor o **loader** collega questi indirizzi simbolici ad indirizzi assoluti.





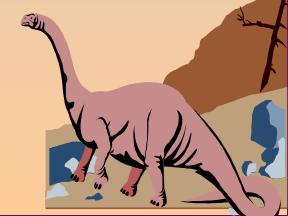
# Associazione di istruzioni e dati ad indirizzi di memoria (II)

- ✓ L'associazione di istruzioni e dati ad indirizzi di memoria si puo' compiere in fase di:
  - Φ **Compilazione:** Se al momento della compilazione è possibile sapere dove il processo risiederà in memoria, può essere generato un codice assoluto. Se, in un momento successivo, la locazione iniziale dovesse cambiare allora sarebbe necessario ricompilare il codice.
  - Φ **Caricamento:** Se al momento della compilazione non è possibile sapere in che punto della memoria risiederà il processo, il compilatore deve generare un codice rilocabile. Così il collegamento finale viene ritardato fino al momento del caricamento e in caso l' indirizzo iniziale cambiasse sarebbe sufficiente caricare solo il codice utente ed incorporare il valore modificato.
  - Φ **Esecuzione:** Il processo, in questa fase, può essere spostato da un segmento all'altro della memoria a condizione che il collegamento venga ritardato fino al momento dell' esecuzione del programma.



# Spazi di indirizzi logici e fisici

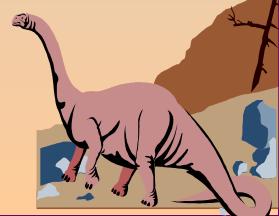
- ✓ Il concetto di *spazio degli indirizzi logici* associato a uno *spazio degli indirizzi fisici* separato è fondamentale per una corretta gestione della memoria.
  - Φ *Indirizzo logico* – generato dalla CPU; detto anche *indirizzo virtuale*.
  - Φ *Indirizzo fisico* – indirizzo visto dall’unità di memoria, cioè caricato nel *registro dell’indirizzo di memoria* (memory address register - MAR).
- ✓ Lo spazio degli indirizzi logico e fisico
  - Φ coincidono nell’ associazione di istruzioni e dati ad indirizzi di memoria in fase di compilazione o caricamento;
  - Φ differiscono invece per quanto riguarda lo schema di associazione di istruzioni e dati ad indirizzi di memoria in fase di esecuzione

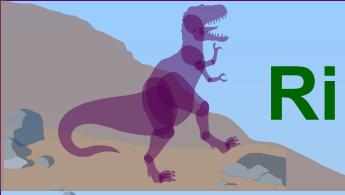




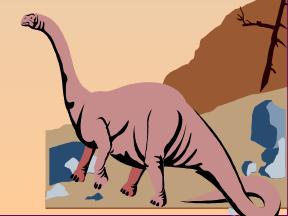
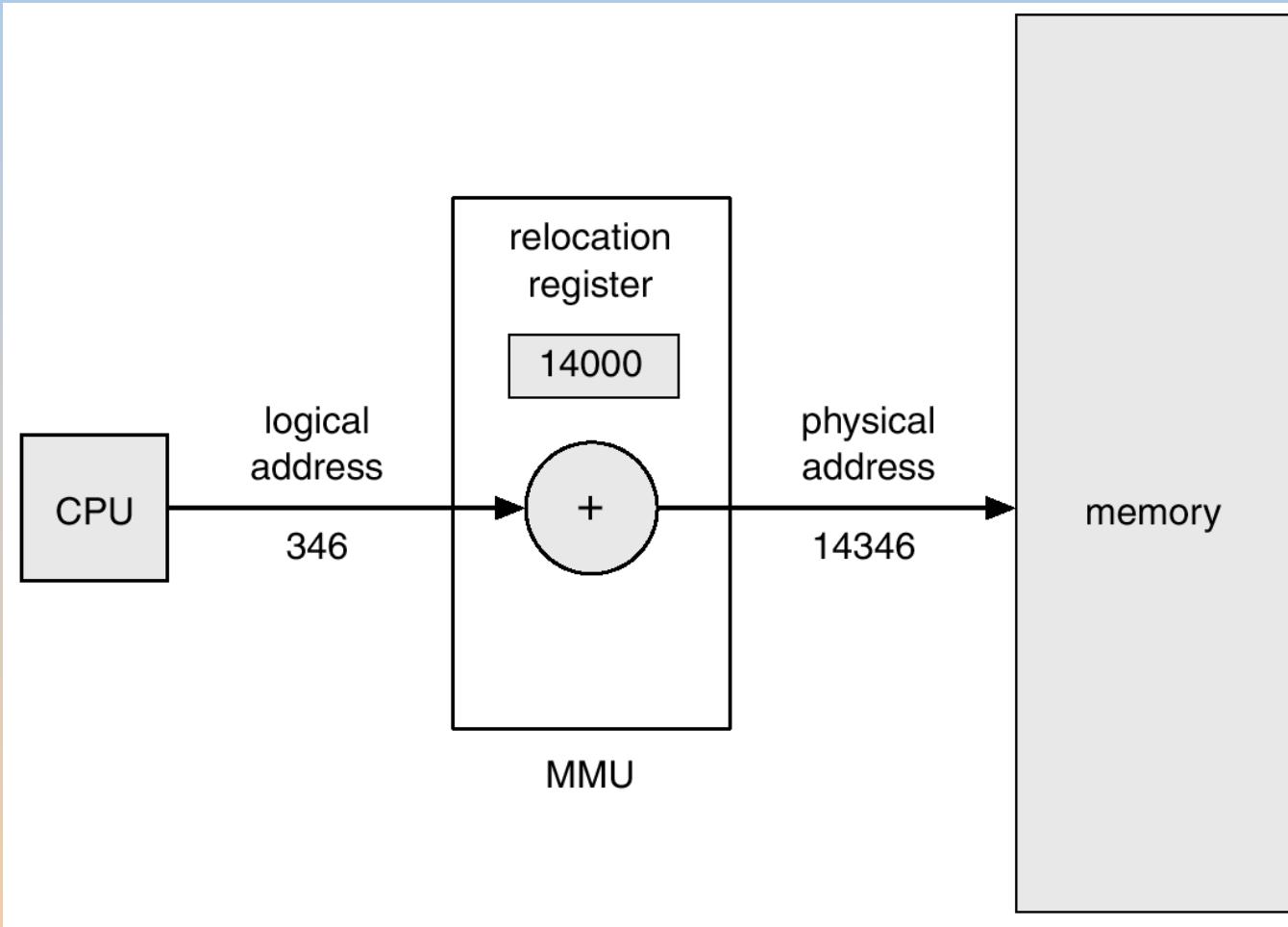
# Unità di gestione della memoria (Memory-Management Unit - MMU)

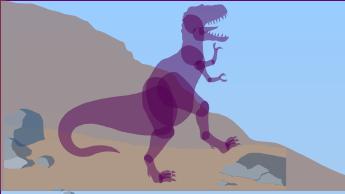
- ✓ Dispositivo hardware che associa in fase di esecuzione gli indirizzi virtuali agli indirizzi fisici.
- ✓ In uno schema basato su MMU, il valore del **registro di rilocazione** viene aggiunto ad ogni indirizzo generato da un processo utente, prima dell'invio all'unità di memoria.
- ✓ Il programma utente tratta esclusivamente indirizzi logici, non "vede" mai gli indirizzi fisici .





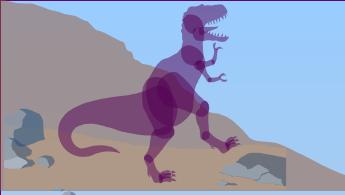
# Rilocazione dinamica tramite un registro di rilocazione





# Caricamento dinamico

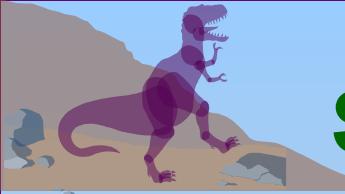
- ✓ Per migliorare l'utilizzo della memoria è possibile adoperare il caricamento dinamico, con il quale una routine viene caricata solo quando viene richiamata.
  - ✓ Migliore utilizzo dello spazio di memoria: le routine non utilizzate non verranno caricate.
  - ✓ Quando bisogna caricare una routine, viene chiamato il loader di collegamento rilocabile che controlla se la routine è già stata caricata o meno.
  - ✓ In caso negativo il loader carica la routine ed aggiorna la tabella degli indirizzi del programma.
  - ✓ Quindi il controllo passa alla routine.
  - ✓ Utile quando sono presenti grandi quantità di codice che sono necessarie per la gestione di casi non frequenti.
  - ✓ Non richiede un intervento particolare del sistema operativo ma va progettato dall'utente in fase di progettazione di programma.
- 



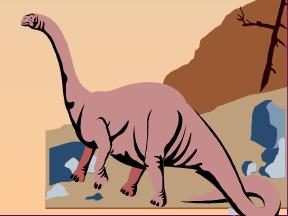
# Collegamento dinamico a librerie condivise

- ✓ In un programma vengono spesso adoperate delle funzioni di libreria
- ✓ Alcuni sistemi operativi supportano solo un *collegamento statico*,
  - Φ nel quale le librerie sono trattate come qualsiasi altra routine e vengono combinate dal loader nell' immagine binaria del programma.
- ✓ Con il *collegamento dinamico*, invece che posticipare il caricamento, viene posticipato il collegamento alle librerie fino al momento dell'esecuzione.
- ✓ Piccole porzioni di codice di riferimento, dette *stub*, indicano come localizzare la giusta procedura residente nella memoria o come caricare la libreria se non è presente.
- ✓ Lo stub poi rimpiazza se stesso con l'indirizzo della procedura per permetterne l'esecuzione.
- ✓ Il collegamento dinamico, a differenza del caricamento dinamico, necessita di un supporto del sistema operativo.
- ✓ Infatti questo è l' unico che può controllare se la routine richiesta da un processo è nella memoria di un altro processo o se più processi possono accedervi.



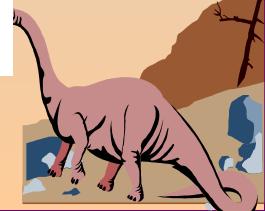
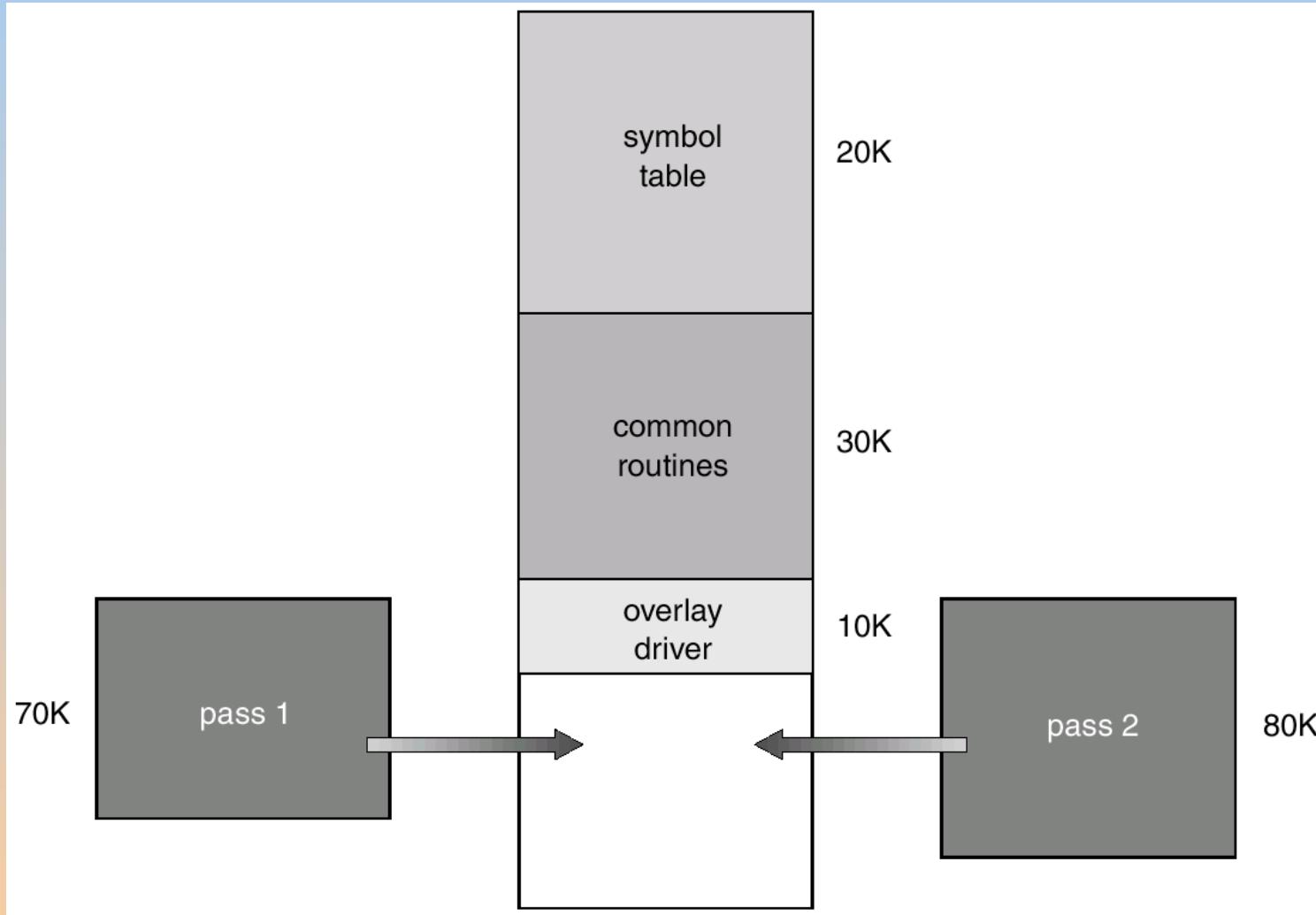


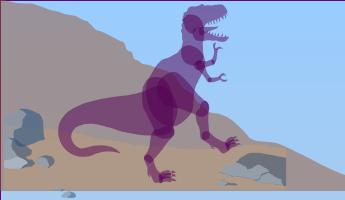
# Sovrapposizione di sezioni (overlay)

- ✓ Possibilità di mantenere in memoria soltanto le istruzioni e i dati dati che si usano con maggior frequenza.
  - ✓ Quando sono necessarie altre istruzioni queste si caricano nello spazio precedentemente occupato dalle istruzioni che non sono più in uso.
  - ✓ Necessario quando il processo è più grande della memoria che gli può essere assegnata.
  - ✓ Implementato in genere dall'utente.
  - ✓ Non vi è supporto da parte del S.O.
  - ✓ Utilizzato nei microcalcolatori e in sistemi dotati di memoria limitata.
- 



# Sovrapposizione di sezioni per un assemblatore a due passi



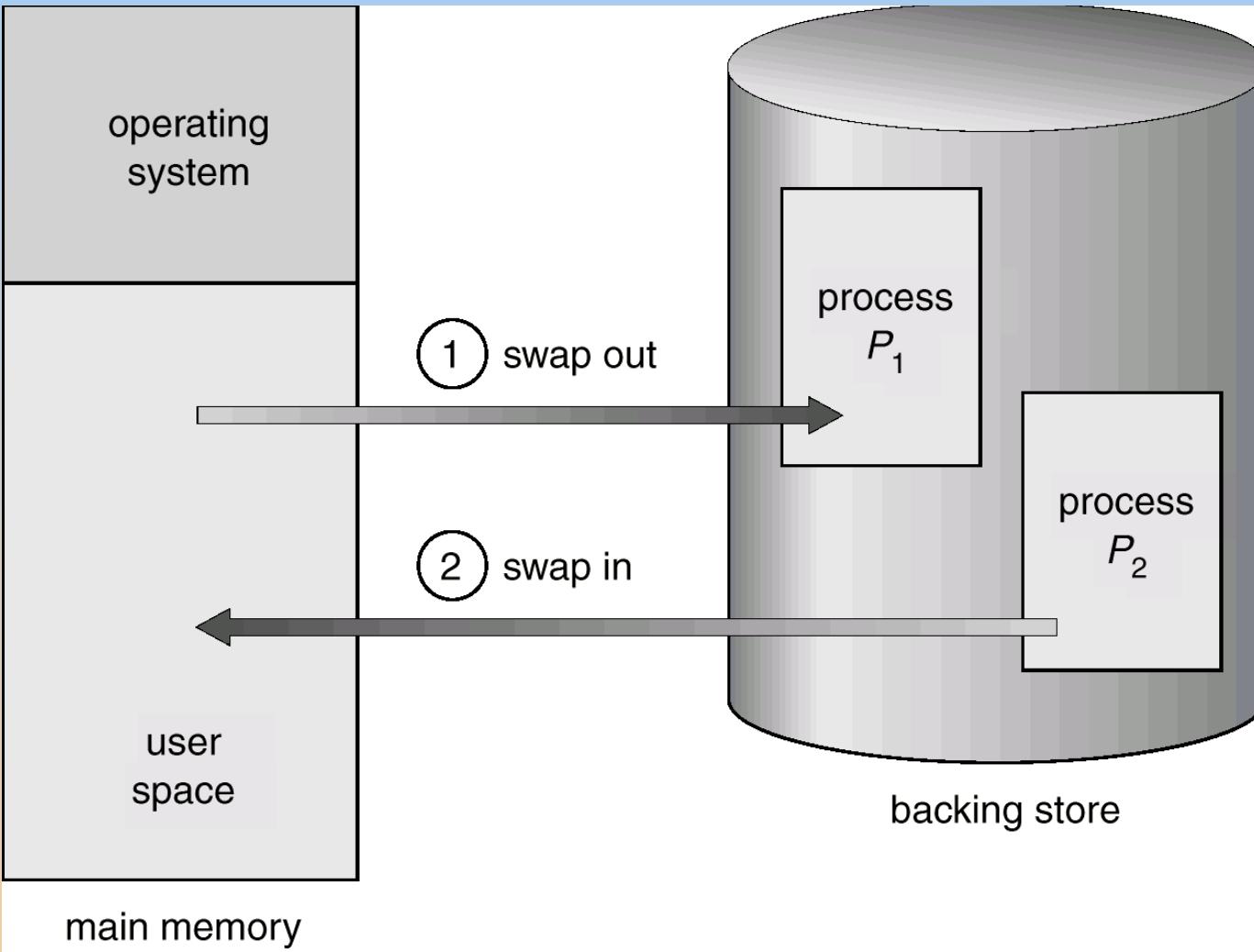


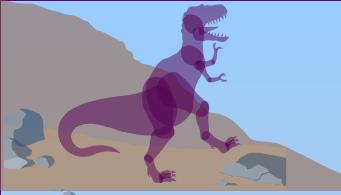
# Avvicendamento di processi (swapping)

- ✓ Per essere eseguito un processo deve trovarsi nella memoria centrale.
- ✓ Un processo può essere trasferito temporaneamente nella *memoria ausiliaria (backing store)*,
  - Φ da cui si riporta nella memoria centrale al momento di riprendere l'esecuzione.
  - Φ Ad es. in una situazione di timesharing ci sono normalmente molti utenti e la memoria centrale non è sempre sufficiente a contenere tutti i loro processi.
- ✓ Diventa necessario memorizzare i processi in eccesso nella memoria secondaria di supporto.
- ✓ Per eseguire questi processi bisogna spostarli dal disco alla memoria e viceversa.
- ✓ Questo metodo viene detto *avvicendamento dei processi nella memoria (swapping)*
- ✓ *Memoria ausiliaria di supporto (backing store)*:
  - Φ un disco veloce, grande abbastanza da memorizzare le copie delle immagini di memoria per tutti i processi in esecuzione nel sistema.
  - Φ Deve poter provvedere accesso diretto a queste immagini di memoria.



# Swapping

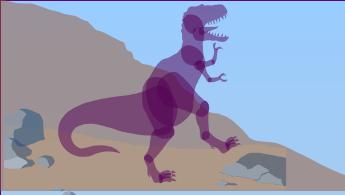




# Avvicendamento di processi (II)

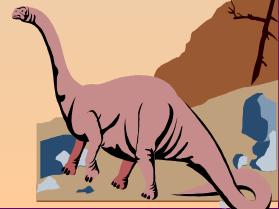
- ✓ Una variante di questa politica di swapping viene utilizzata per gli algoritmi di scheduling basati su priorità.
- ✓ Quando si presenta la necessità di eseguire un processo con priorità maggiore,
  - Φ il gestore della memoria effettua lo swap out di un altro con priorità inferiore, che verrà ripreso in memoria centrale appena quello con priorità maggiore termina.
- ✓ Questo tipo di swapping a volte viene detto **roll out, roll in**.
- ✓ Se l'associazione degli indirizzi logici agli indirizzi fisici della memoria (binding) viene effettuata al momento dell'assemblaggio o del caricamento
  - Φ allora il processo viene ricaricato nello stesso spazio di memoria occupato precedentemente.
- ✓ Se il binding viene fatto a tempo di esecuzione,
  - Φ allora il processo potrebbe essere caricato in uno spazio di memoria diverso.
- ✓ Attualmente lo swapping “semplice” è usato in pochi sistemi, richiede infatti un elevato tempo di gestione.
- ✓ Versioni modificate di swapping si trovano su molti sistemi, quali **UNIX, Linux, and Windows**.

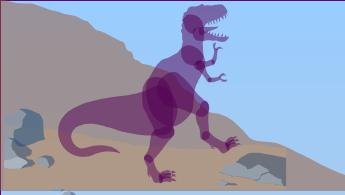




# Avvicendamento di processi (III)

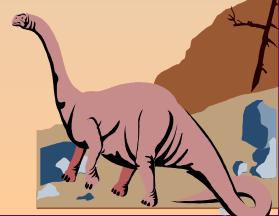
- ✓ Quando lo scheduler della CPU deve eseguire un processo, richiama il dispatcher, che controlla se il primo processo della coda è in memoria.
- ✓ Se il processo non è in memoria e non c'è spazio libero,
  - Φ allora il dispatcher esegue lo swap out di un processo presente in memoria e lo swap in del processo da eseguire.
  - Φ Quindi vengono caricati i registri e trasferito il controllo al processo che deve essere eseguito.
- ✓ Per utilizzare efficacemente la CPU è necessario che il tempo di esecuzione di ogni processo sia lungo rispetto al tempo di swap.

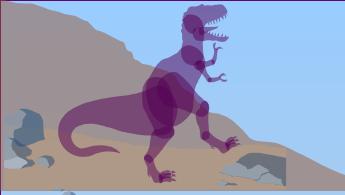




# Avvicendamento di processi (IV)

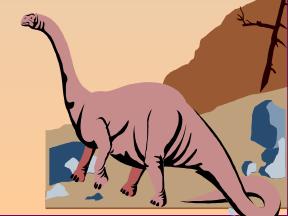
- ✓ La maggior parte del tempo di swap è dato dal tempo di trasferimento,
  - Φ che è direttamente proporzionale alla quantità di memoria sottoposta a swap.
- ✓ Lo swapping presenta diversi svantaggi.
  - Φ Sarebbe utile sapere quanta memoria viene effettivamente utilizzata per effettuare lo swap out riducendo così il tempo.
  - Φ Quindi il sistema dovrebbe essere informato di quanta memoria un processo necessita tramite una system call effettuata dal processo.
- ✓ Inoltre per sottoporre a swap un processo è necessario che il processo sia inattivo.





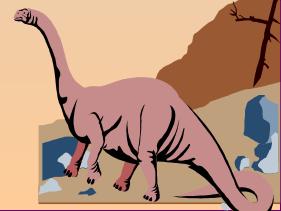
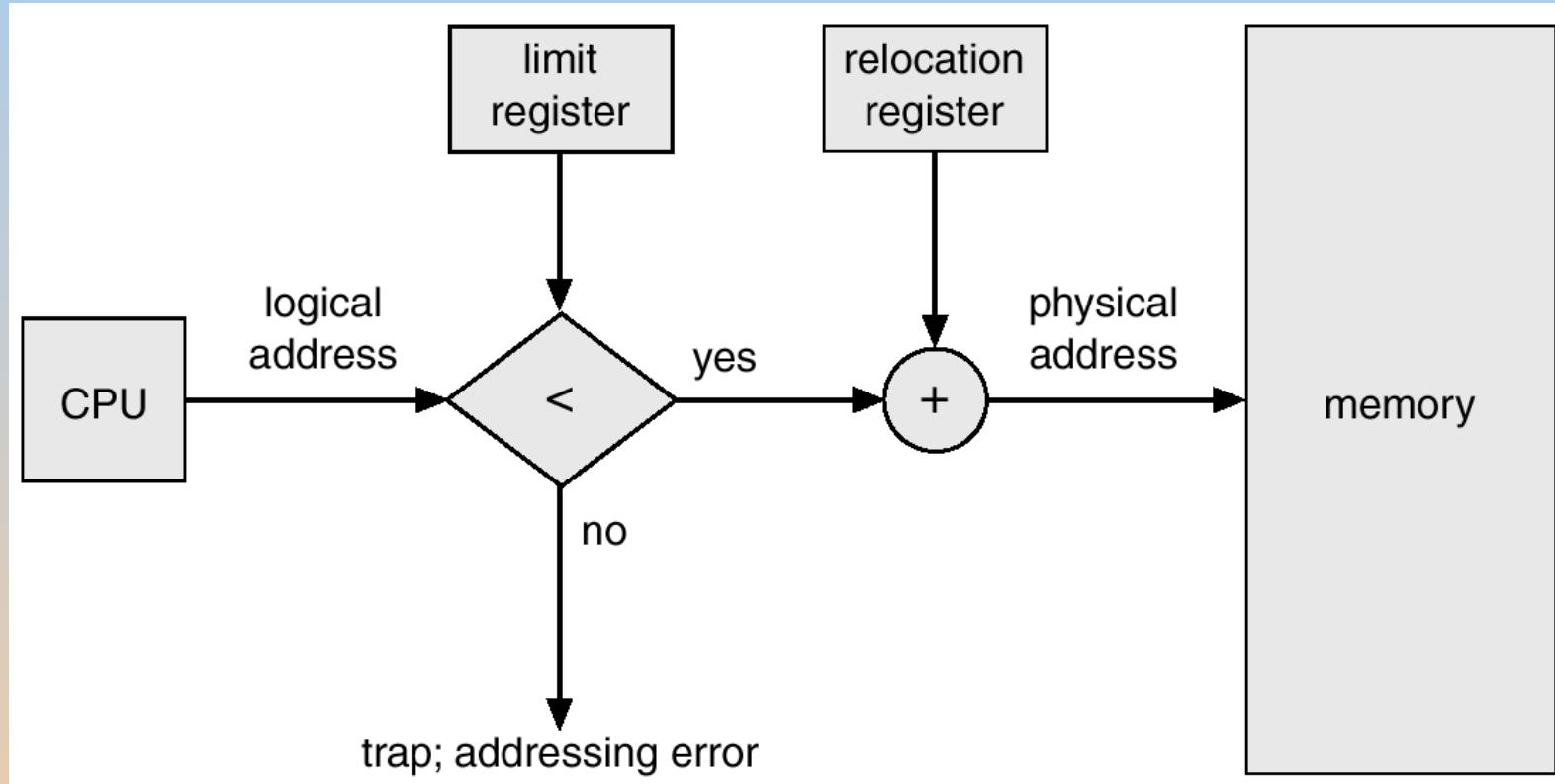
# Allocazione della memoria

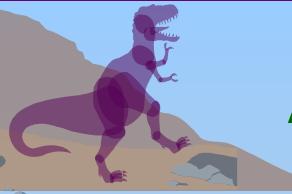
- ✓ La memoria centrale si divide di solito in due partizioni:
  - Φ *Sistema operativo residente*, generalmente mantenuto nella parte bassa della memoria, insieme al vettore degli interrupt.
  - Φ *Processi utenti*, generalmente mantenuti nella parte alta della memoria.
- ✓ Per quanto riguarda la memoria, la protezione del S.O. dai processi utenti e la protezione dei processi utenti dagli altri processi utenti si può realizzare usando un registro di rilocazione insieme con un registro limite.
- ✓ Il registro di rilocazione contiene il valore dell'indirizzo fisico minore.
- ✓ Il registro limite contiene il range di indirizzi logici:
  - Φ ciascun indirizzo logico deve essere minore del contenuto del registro limite.





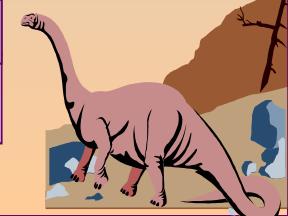
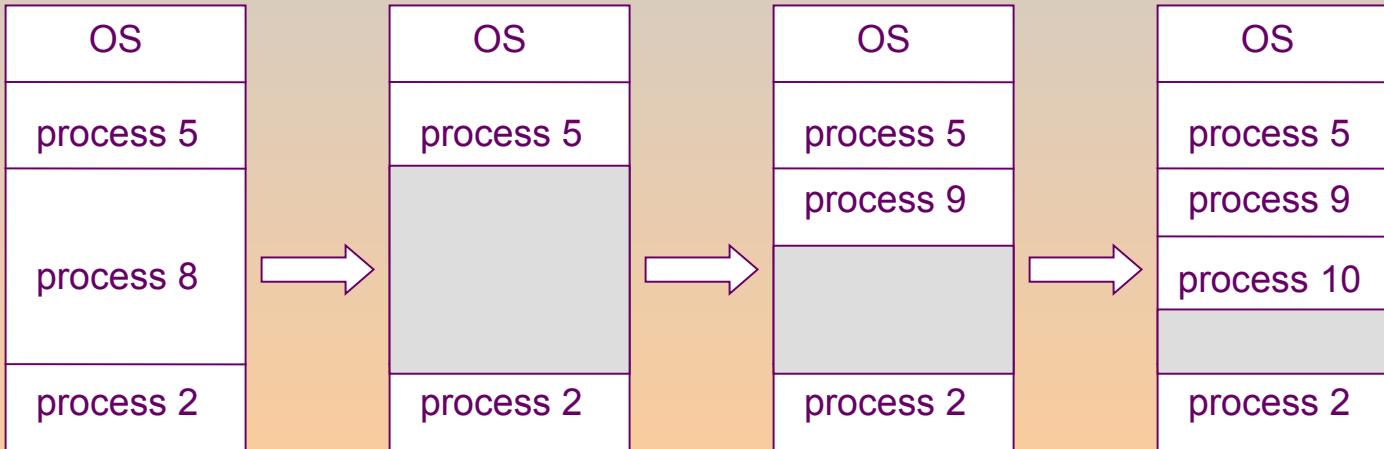
# Registri di rilocazione e di limite

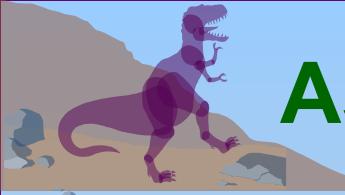




# Allocazione contigua della memoria

- ✓ Uno dei metodi più semplici di gestione della memoria consiste nel dividerla in *partizioni* di dimensione fissa.
  - Φ Ogni partizione deve contenere esattamente un processo.
  - Φ *Buco* – blocco di memoria disponibile.
    - 4 Buchi di dimensioni diverse sono in genere presenti in memoria.
  - Φ Quando un processo viene caricato in memoria, gli viene assegnata la memoria di un *buco* sufficientemente grande da contenerlo.
  - Φ Il sistema operativo deve mantenere informazioni in una tabella circa le partizioni allocate e quelle libere (*buchi*).
  - Φ *Frammentazione esterna* : quando si caricano e si rimuovono i processi dalla memoria si frammenta lo spazio libero in tante piccole parti.



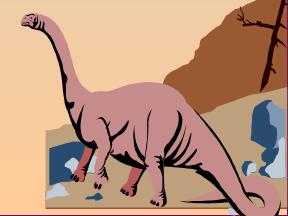


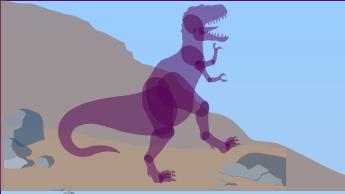
# Assegnazione dinamica della memoria

Come si può soddisfare una richiesta di  $n$  blocchi di memoria?

- ✓ **First-fit:** Allocare il primo buco abbastanza grande.
- ✓ **Best-fit:** Allocare il più piccolo buco abbastanza grande, si deve compiere la sua ricerca in tutta la lista dei buchi liberi. Produce il più piccolo buco residuo.
- ✓ **Worst-fit:** Allocare il più grande buco disponibile; si deve compiere la sua ricerca in tutta la lista dei buchi liberi. Produce il più grande buco residuo.

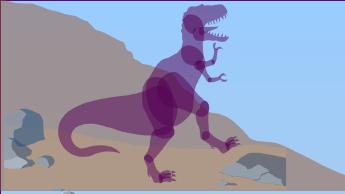
First-fit e best-fit sono più efficienti di worst-fit in termini di velocità e utilizzo della memoria.





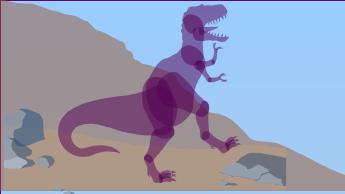
# Frammentazione

- ✓ **Frammentazione esterna** – lo spazio complessivo di memoria disponibile necessario per soddisfare la richiesta esiste, ma non è contiguo.
  - ✓ **Frammentazione interna** – la memoria allocata è leggermente più grande di quella richiesta, questa memoria residua, interna alla partizione, non viene utilizzata.
  - ✓ La frammentazione esterna può essere ridotta con un'operazione di compattamento.
    - Φ Il contenuto della memoria viene riordinato per riunire la memoria libera in un solo grosso blocco.
    - Φ Il compattamento è possibile solo se la rilocazione è dinamica e si compie in fase di esecuzione.
    - Φ Problemi realtivi all' I/O:
      - 4 Un job non può essere spostato in un'altra porzione di memoria mentre esegue operazioni di I/O.
      - 4 Possibile soluzione: fare I/O solo usando buffer nella porzione di memoria del S.O..
- 

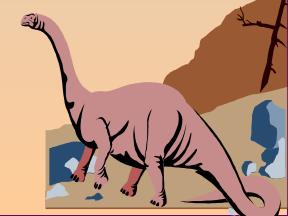


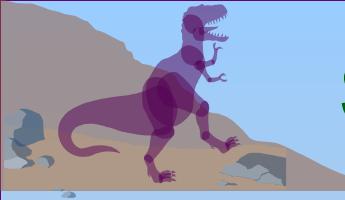
# Paginazione

- ✓ Con il metodo della paginazione la memoria viene suddivisa in blocchi di dimensione fissa.
  - ✓ Lo spazio indirizzi logico di un processo può essere allocato in blocchi anche non contigui di memoria fisica.
  - ✓ I blocchi della memoria fisica sono detti **frame**, mentre i blocchi della memoria logica sono detti **pagine**.
  - ✓ Quando un processo è pronto per essere eseguito si caricano le sue pagine nei blocchi di memoria disponibile, prendendole dalla memoria ausiliaria, che è divisa in blocchi di dimensione fissa uguale a quella dei frame.
  - ✓ Si risolve il problema della frammentazione esterna (si potrà avere invece frammentazione interna).
  - ✓ Bisogna mantenere traccia dei frame liberi: per eseguire un processo di  $n$  pagine ci sarà bisogno di  $n$  frame liberi in cui caricare il processo.
  - ✓ Una *tabella delle pagine* verrà utilizzata per tradurre l'indirizzo logico in un indirizzo fisico.
- 



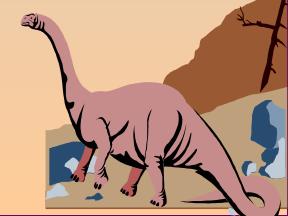
# Paginazione (II)

- ✓ La paginazione non è altro che una forma di rilocazione dinamica.
  - ✓ Ogni indirizzo logico è associato a un indirizzo fisico dall' hardware di paginazione.
  - ✓ La paginazione è simile all' utilizzo di una tabella di registri base (rilocazione), uno per ciascun frame di memoria.
  - ✓ Con la paginazione ciascun frame libero può essere utilizzato ed allocato ad un processo che deve essere eseguito.
  - ✓ I frame vengono allocati come unità. Se i requisiti di memoria non combaciano con i limiti di pagina, l' ultimo frame allocato può non essere completamente pieno.
  - ✓ Il caso peggiore si presenta quando un processo necessita di  $n$  pagine più un byte. Anche in questo caso bisogna allocare  $n+1$  pagine.
  - ✓ Di solito, in media, con la paginazione si ha una frammentazione interna di mezza pagina per processo.
- 



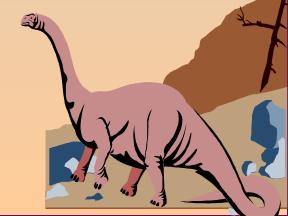
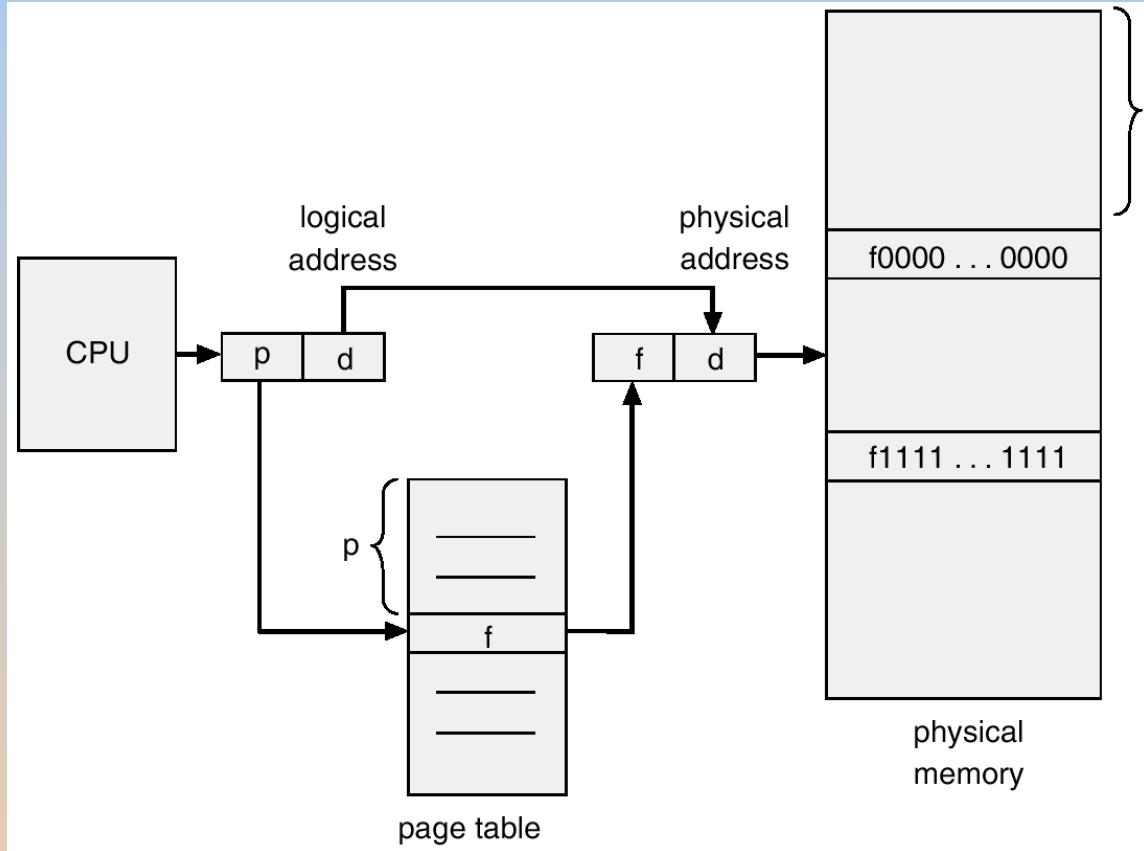
# Schema di traduzione degli indirizzi

- ✓ Ogni indirizzo generato dalla CPU è diviso in due parti:
  - Φ *Numero di pagina (p)* – usato come indice per la *tabella delle pagine* che contiene l' indirizzo base di ogni pagina nella memoria fisica.
  - Φ *Scostamento di pagina (offset) (d)* – viene combinato con l'indirizzo di base per definire l'indirizzo della memoria fisica che viene inviato all'unità di memoria.
- ✓ La dimensione di una pagina, che è la stessa di un frame, può variare da un compilatore all' altro a seconda del tipo di hardware. Di solito, comunque, si preferisce una potenza di 2 compresa fra 512 e 16MB.
- ✓ Se la dimensione dello spazio degli indirizzi logici è  $2^m$  e la dimensione di una pagina è di  $2^n$  unità di indirizzamento, allora gli  $m-n$  bit più significativi di un indirizzo logico indicano il numero di pagina, e gli  $n$  bit meno significativi indicano l'offset di pagina.



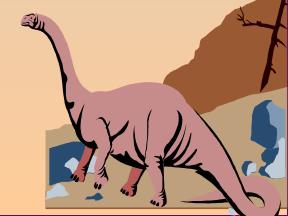
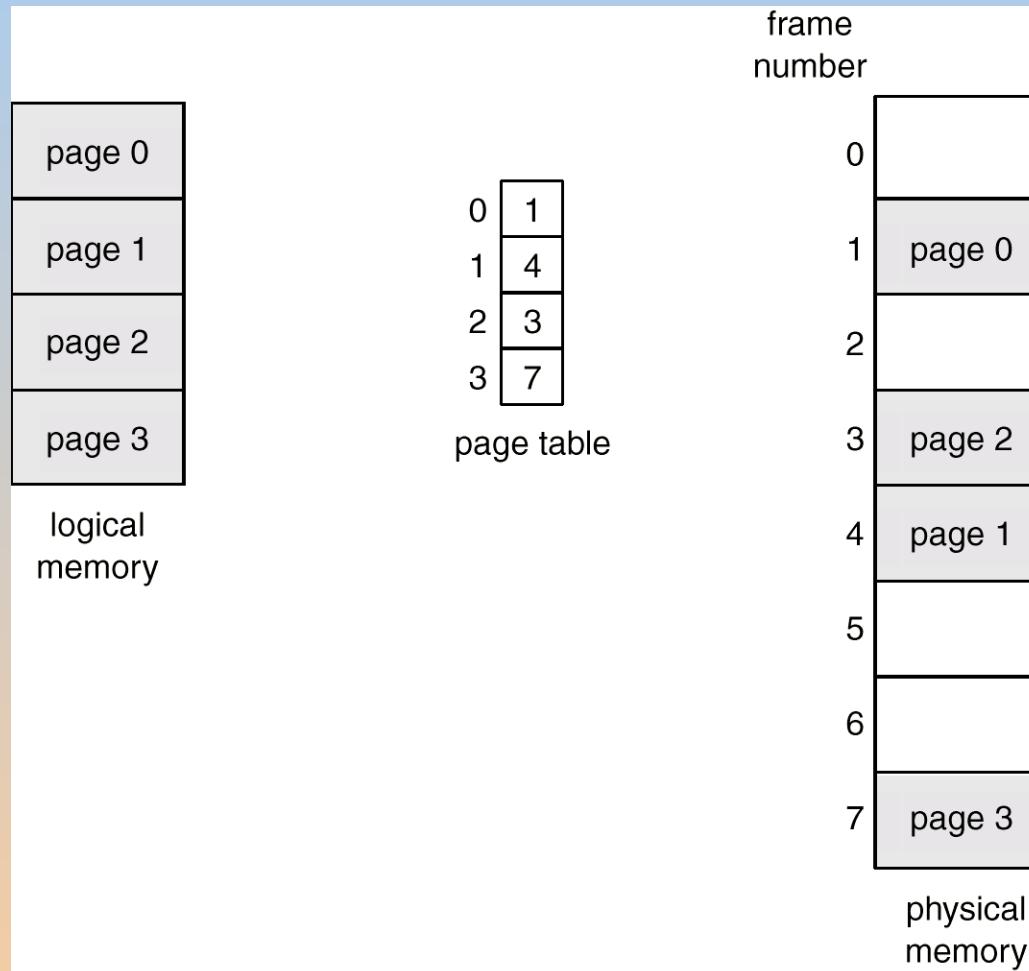


# Architettura di paginazione





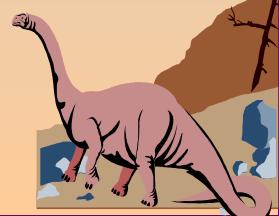
# Modello di paginazione di memoria logica e memoria fisica

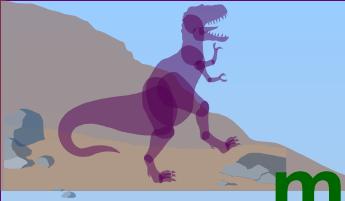




# Esempio di paginazione per una memoria di 32 byte con pagina di 4 byte

- ✓ Supponiamo di avere una memoria fisica di 32 byte con ogni pagina di 4 byte (ossia 8 pagine) .
- ✓ L' indirizzo logico 0 è pagina 0, offset 0. Nella tabella delle pagine, la pagina 0 si trova nel frame 5.
- ✓ Quindi l' indirizzo logico 0 viene mappato nell' indirizzo fisico 20 ( $20=(5 \times 4)+0$ ).
- ✓ L' indirizzo logico 3 (pagina 0, offset 3) viene mappato nell' indirizzo fisico 23 ( $23=(5 \times 4)+3$ ).
- ✓ L' indirizzo logico 4 è pagina 1, offset 0; secondo la tabella delle pagine, la pagina 1 è mappata nel frame 6. L' indirizzo 4 viene mappato nell' indirizzo fisico 24 ( $24 = (6 \times 4) + 0$ ).
- ✓ L' indirizzo logico 13 viene mappato nell' indirizzo fisico 9.





# Esempio di paginazione per una memoria di 3 byte con pagina di 4 byte

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

logical memory

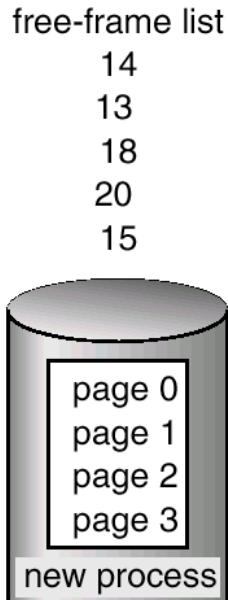
0	5
1	6
2	1
3	2

page table

0	
4	i
	j
	k
	l
8	m
	n
	o
	p
12	
16	
20	a
	b
	c
	d
24	e
	f
	g
	h
28	

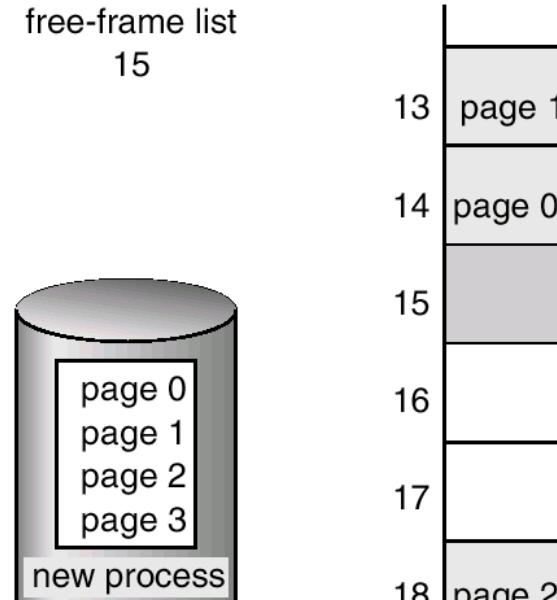
physical memory

# Blocchi di memoria liberi



(a)

Prima dell'assegnazione



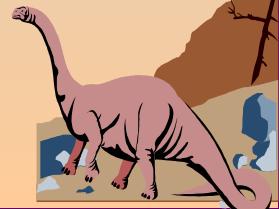
(b)

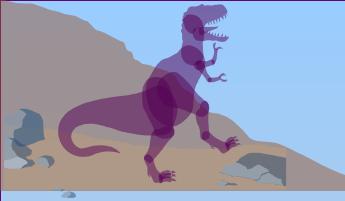
Dopo l'assegnazione



# Implementazione della tabella delle pagine

- ✓ La tabella delle pagine è mantenuta nella memoria centrale.
- ✓ Il *Registro di base della tabella delle pagine* (*Page-table base register* - PTBR) punta alla tabella delle pagine.
- ✓ Il *Registro di lunghezza della tabella delle pagine* (*Page-table length register* - PTLR) indica le dimensioni della tabella delle pagine.
- ✓ In questo schema l'accesso ad ogni dato o istruzione necessita di due accessi alla memoria, uno per la tabella delle pagine ed uno per il dato o istruzione.
- ✓ Il problema dei due accessi alla memoria può essere risolto attraverso l'utilizzo di una memoria cache speciale, molto veloce detta *memoria associativa* (*associative memory or translation look-aside buffers* - TLBs)



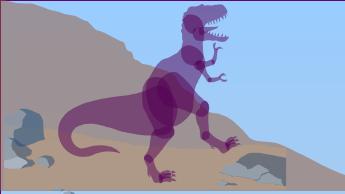


# Memoria Associativa

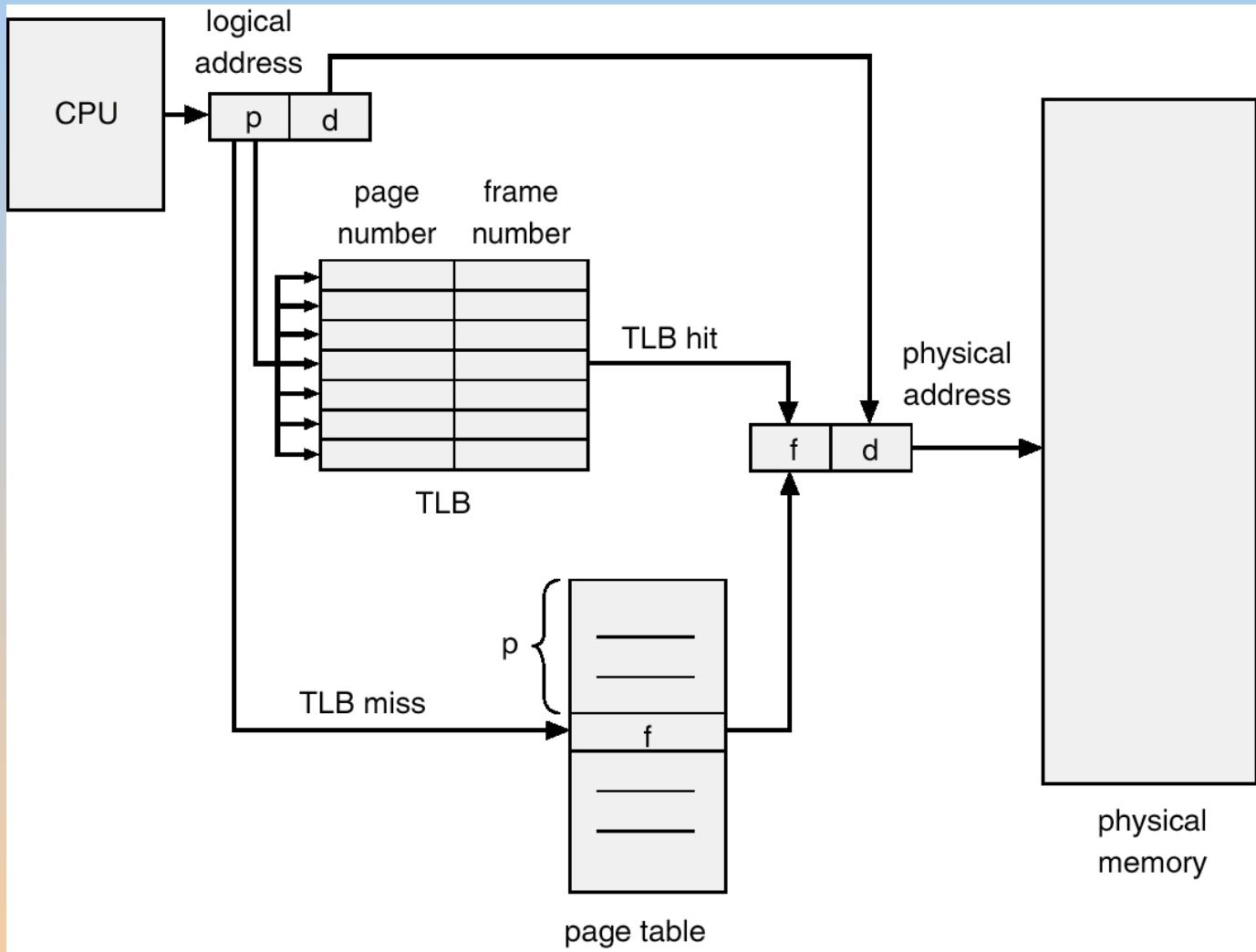
- ✓ Ogni registro è formato da una chiave e un valore. Quando ai registri associativi si presenta un elemento, viene confrontato con tutte le chiavi contemporaneamente, riducendo il tempo di ricerca; (questa soluzione presenta uno svantaggio legato al tipo di hardware molto costoso).
- ✓ Memoria associativa: ricerca parallela

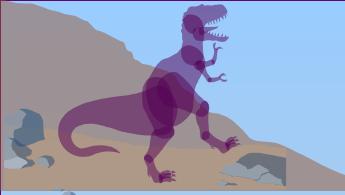
Page #	Frame #

- ✓ Traduzione dell'indirizzo ( $A'$ ,  $A''$ )
  - Φ Se  $A'$  è in un registro associativo, il numero di frame si ottiene tramite memoria associativa con una ricerca parallela.
  - Φ Altrimenti il numero di frame si ottiene dalla tabella delle pagine in memoria.
  - Φ *Tasso di successi (hit ratio)* – percentuale di volte che un numero di pagina si trova nel TLB: dipende anche dal numero di registri associativi.



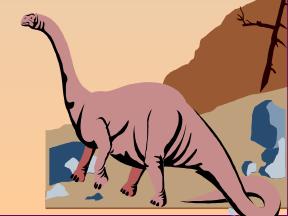
# Architettura di paginazione con memoria associativa (TLB)

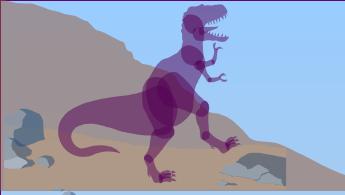




# Protezione della memoria

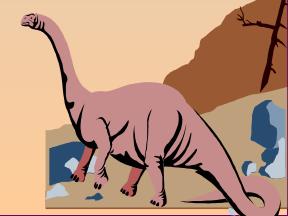
- ✓ Associato ad ogni frame vi è un **bit di protezione**, che normalmente si trova nella tabella delle pagine.
- ✓ Questo bit viene utilizzato per la protezione della memoria e determina se una pagina è di lettura e scrittura o di sola lettura.
  - Φ In questo modo si evita che si vada a scrivere su pagine di sola lettura.
- ✓ Nel caso si tenti di scrivere su una pagina di sola lettura viene generato un **trap** dell'hardware al sistema operativo.
- ✓ Un ulteriore bit, detto **bit di validità** viene associato a ciascun elemento della tabella delle pagine.
- ✓ Quando è posto a *valido* indica che la pagina associata è nello spazio di indirizzi logici del processo e pertanto è una pagina a cui il processo può legalmente accedere.
- ✓ In caso è posto a *invalido* allora la pagina non è nello spazio degli indirizzi logici del processo.





# Protezione della memoria (II)

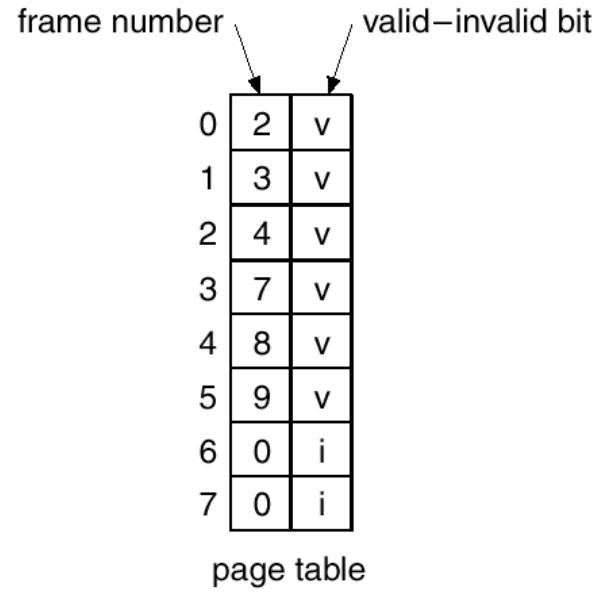
- ✓ Alcuni sistemi dispongono di registri hardware contenenti la lunghezza della tabella delle pagine
  - Φ detti ***registri di lunghezza della tabella delle pagine*** per indicare la dimensione della tabella.
  - Φ Questo valore viene confrontato ad ogni indirizzo logico per verificare che si trovi nell'intervallo valido per il processo.



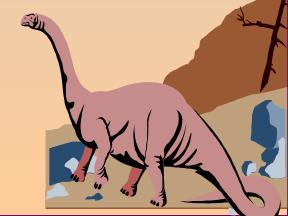


# Bit di validità in una tabella delle pagine

00000	page 0
	page 1
	page 2
	page 3
	page 4
10,468	page 5
12,287	



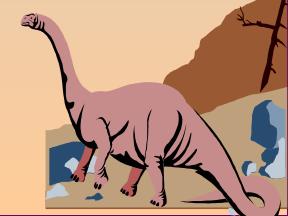
0	
1	
2	page 0
3	page 1
4	page 2
5	
6	
7	page 3
8	page 4
9	page 5
⋮	
	page <i>n</i>

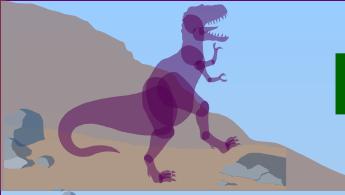




# Struttura della tabella delle pagine

- ✓ La maggior parte dei moderni computer supporta uno spazio di indirizzi logici estremamente grande.
- ✓ In un ambiente di questo tipo la stessa tabella delle pagine finirebbe per diventare eccessivamente grande.
- ✓ Chiaramente, sarebbe meglio evitare di allocare la tabella delle pagine in modo contiguo in memoria centrale.
- ✓ Una soluzione semplice consiste nel dividere la tabella delle pagine in parti più piccole; questo risultato può essere ottenuto in molti modi differenti.
  - Φ Paginazione gerarchica.
  - Φ Tabella delle pagine di tipo hash.
  - Φ Tabella delle pagine invertita.





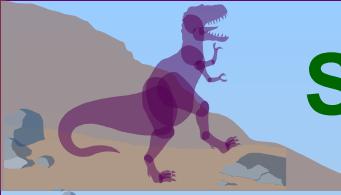
# Paginazione gerarchica: esempio di paginazione a due livelli

- ✓ Il metodo consiste nell' adottare uno schema di paginazione a più livelli, nel quale la stessa tabella delle pagine viene paginata.
- ✓ Un indirizzo logico (su una macchina a 32 bit con pagina di 4K) viene diviso in:
  - Φ Un numero di pagina di 20 bit.
  - Φ Uno scostamento di pagina di 12 bit.
- ✓ Giacché la tabella delle pagine è paginata, il numero di pagina è ulteriormente diviso in:
  - Φ Un numero di pagina di 10 bit.
  - Φ Uno scostamento di pagina di 10 bit.
- ✓ Quindi un indirizzo logico può essere così schematizzato:

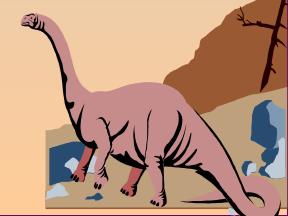
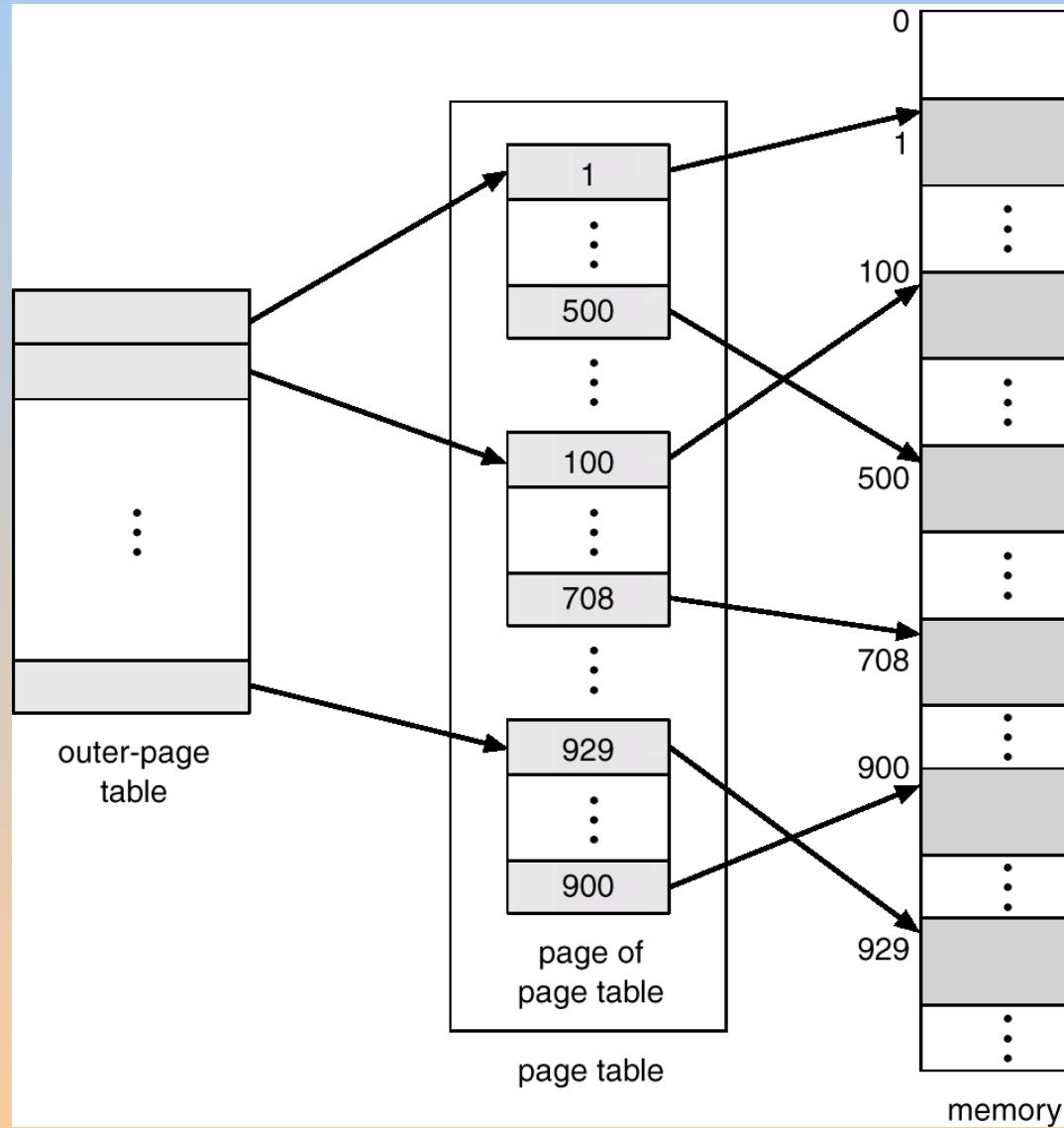
numero di pagina		scostamento di pagina
$p_i$	$p_2$	$d$
10	10	12

dove  $p_i$  è un'indice della tabella delle pagine di primo livello (tabella esterna delle pagine), e  $p_2$  è lo scostamento all'interno della pagina indicata da questa tabella.

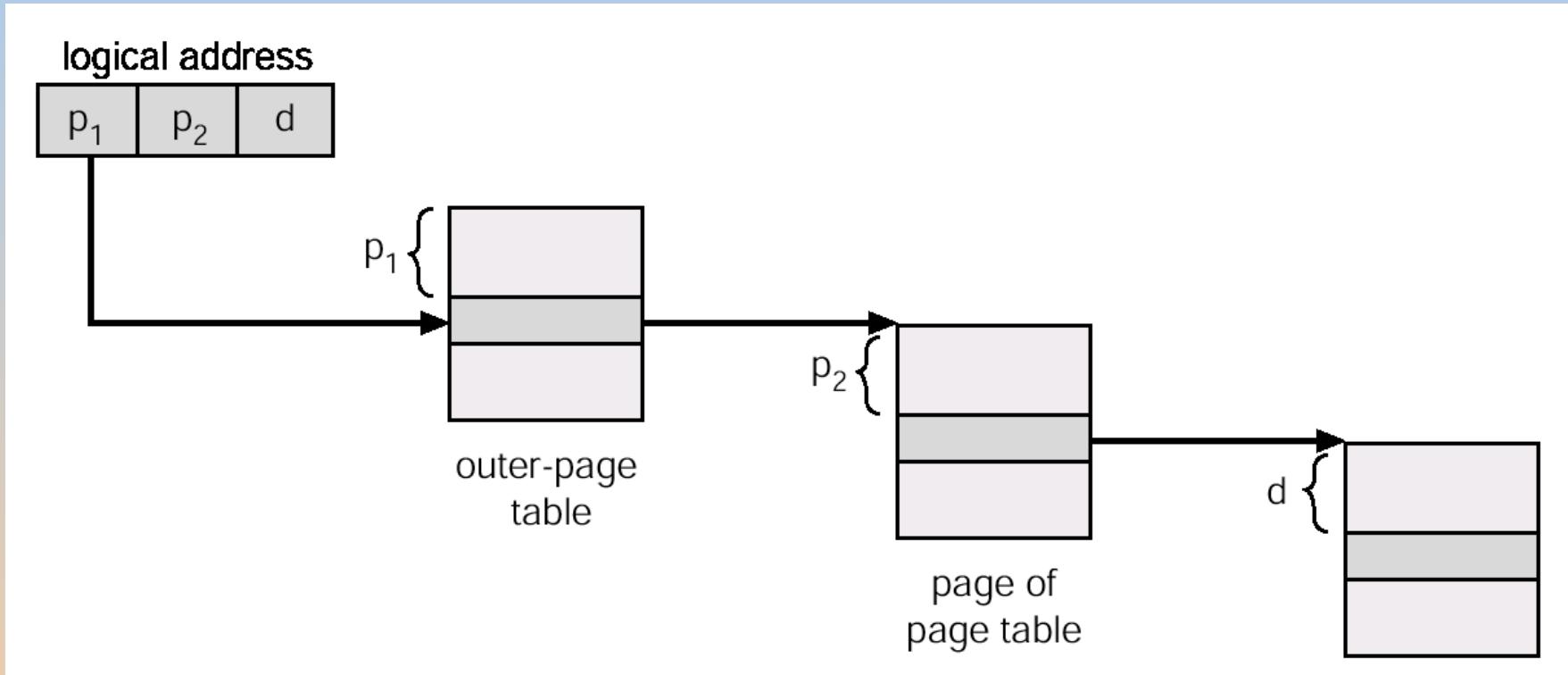


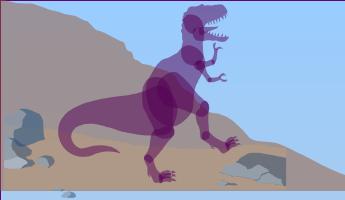


# Schema di una tabella delle pagine a due livelli



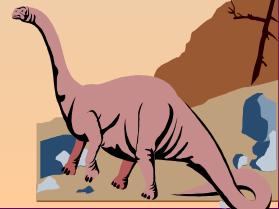
# Traduzione degli indirizzi per un'architettura a 32 bit con paginazione a 2 livelli





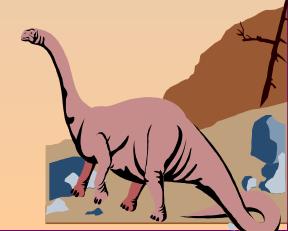
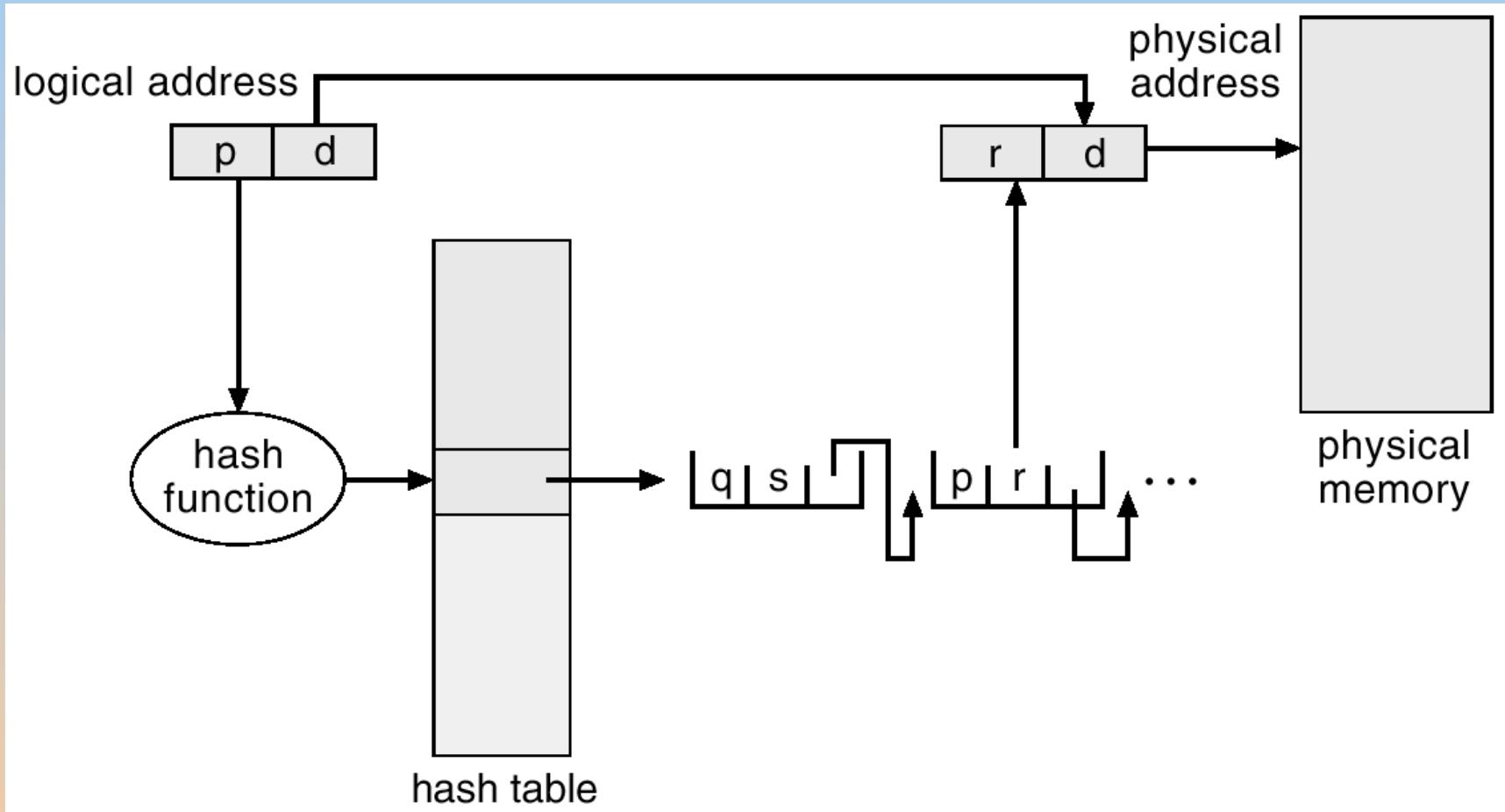
# Tabella delle pagine di tipo hash

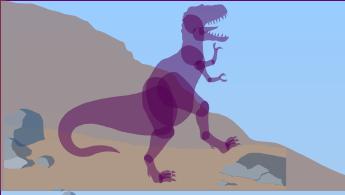
- ✓ Comune per gli spazi di indirizzi relativi ad architetture oltre i 32 bit.
- ✓ Consiste nell'impiego di una *tabella delle pagine di tipo hash* in cui l'argomento della funzione di hashing è il numero della pagina virtuale.
- ✓ Per la gestione delle collisioni, ogni elemento della tabella hash contiene una lista concatenata degli elementi (numero di pagina virtuale, indirizzo di pagina fisica, puntatore next) che la tabella hash fa corrispondere alla stessa locazione.
- ✓ Quindi si applica la funzione hash al numero della pagina virtuale, e si scorre la lista relativa all'elemento della tabella hash ottenuto, fino a che non si identifica la corrispondente pagina fisica.





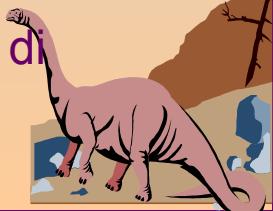
# Tabella delle pagine di tipo hash

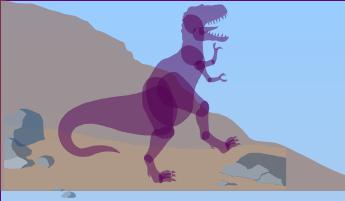




# Tabella delle pagine invertita

- ✓ Generalmente, si associa una tabella delle pagine ad ogni processo
- ✓ Tale tabella ha un elemento per ogni pagina virtuale che il processo sta utilizzando, oppure un elemento per ogni indirizzo virtuale.
- ✓ Questa è una rappresentazione naturale della tabella,
  - Φ poichè i processi fanno riferimento alle pagine tramite gli indirizzi virtuali delle pagine stesse.
- ✓ Il sistema operativo deve poi tradurre questo riferimento in un indirizzo di memoria fisica.
- ✓ Poichè la tabella è ordinata per indirizzo virtuale,
  - Φ il sistema operativo può calcolare in che punto della tabella si trovi l'elemento dell' indirizzo fisico associato, e utilizzare direttamente tale valore.
- ✓ Uno degli inconvenienti ,in questo caso, è costituito dalla dimensione di ciascuna tabella delle pagine,
  - Φ che può avere milioni di elementi e consumare grandi quantità di memoria fisica.

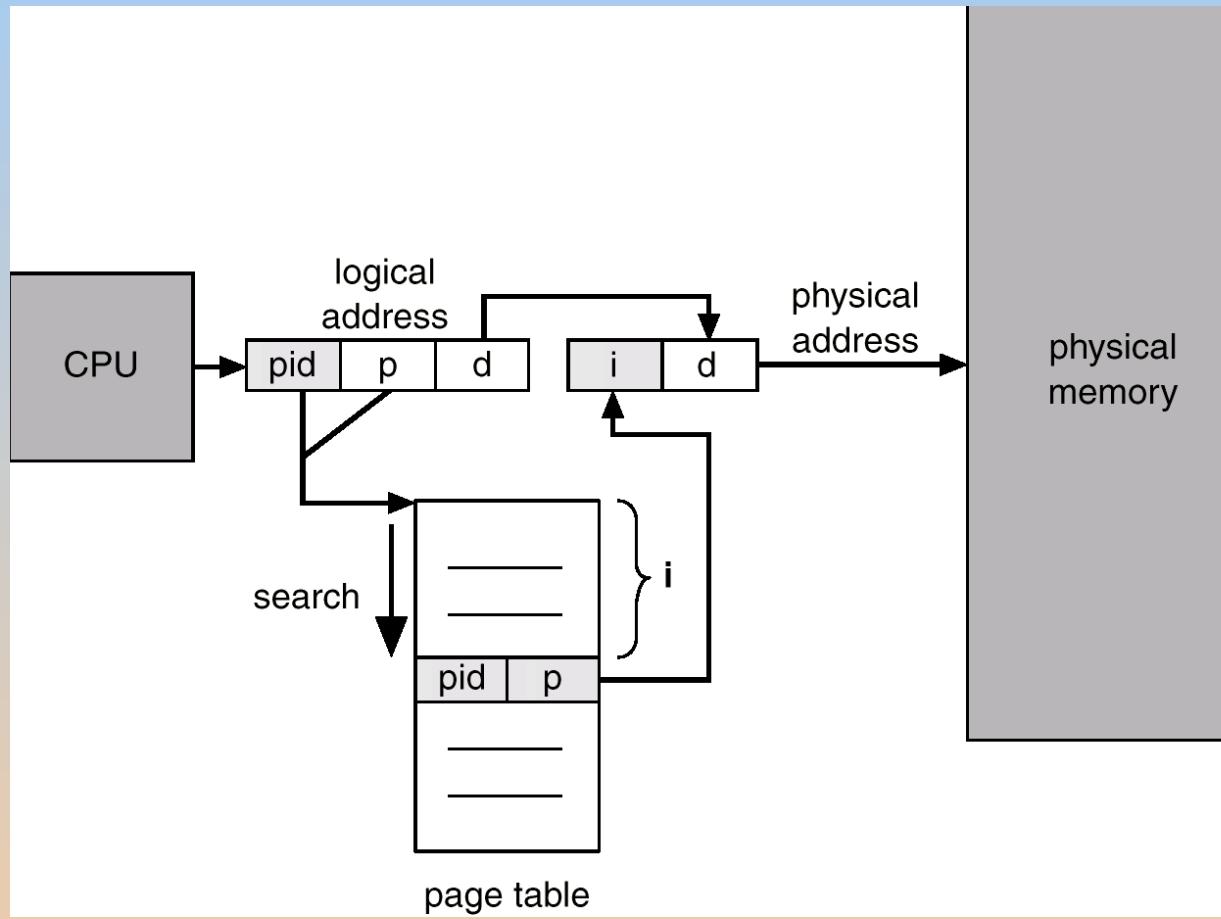


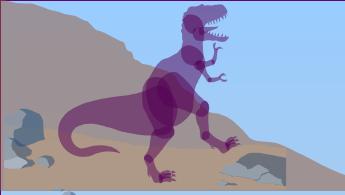


# Tabella delle pagine invertita (II)

- ✓ Per risolvere questo problema si può fare uso della *tabella delle pagine invertita* che ha un elemento per ogni pagina reale (frame) di memoria.
  - ✓ Ciascun elemento è quindi costituito dall' indirizzo virtuale della pagina memorizzata in quella reale locazione di memoria, con informazioni sul processo che possiede tale pagina.
  - ✓ Quindi, nel sistema esiste solo una tabella delle pagine che ha un solo elemento per ciascuna pagina di memoria fisica.
  - ✓ Ciascun indirizzo virtuale è formato dalla seguente tripla:  
 $\langle \text{id-processo}, \text{numero di pagina}, \text{offset} \rangle$
  - ✓ Ogni elemento della tabella delle pagine invertita è una coppia  $\langle \text{id-processo}, \text{numero di pagina} \rangle$ .
  - ✓ Quando viene effettuato un riferimento alla memoria, parte dell' indirizzo virtuale, formato da  $\langle \text{id-processo}, \text{numero di pagina} \rangle$ , viene presentato al sottosistema di memoria.
  - ✓ Quindi viene cercata una corrispondenza nella tabella delle pagine invertita.
  - ✓ Se tale corrispondenza viene trovata, ad esempio sull' elemento i, viene generato l' indirizzo fisico  $\langle i, \text{offset} \rangle$ , altrimenti è stato tentato un accesso illegale a un indirizzo.
- 

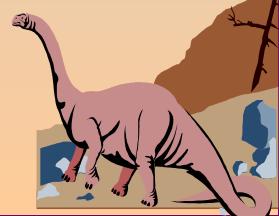
# Esempio di tabella delle pagine invertita

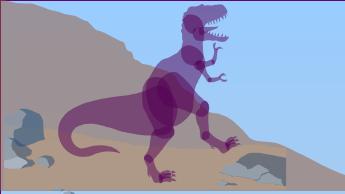




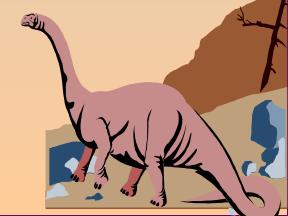
# Tabella delle pagine invertita (III)

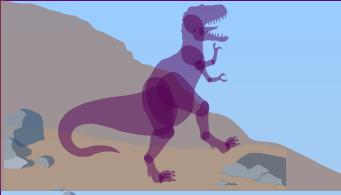
- ✓ Sebbene questo schema riduca la quantità di memoria necessaria per memorizzare ogni tabella delle pagine,
  - ∅ aumenta però la quantità di tempo necessaria per cercare la tabella quando viene fatto riferimento a una pagina.
- ✓ La tabella delle pagine invertita è ordinata per indirizzo fisico, mentre le ricerche vengono effettuate su indirizzi virtuali.
- ✓ Per trovare una corrispondenza occorre cercare in tutta la tabella, e questa ricerca richiede molto tempo.
- ✓ Per alleviare il problema può essere utilizzata una tabella hash per limitare la ricerca a un solo, o a pochi, elementi della tabella delle pagine.



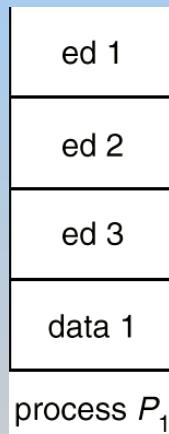


# Pagine condivise

- ✓ Un altro vantaggio della paginazione consiste nella possibilità di *condividere* codice comune,
    - Φ cosa importante soprattutto in un ambiente con time-sharing.
  - ✓ Un codice eseguibile è detto *rientrante* (o *puro*) se è un codice non automodificante, poichè non cambia mai durante l' esecuzione.
  - ✓ Quindi, due o più processi possono eseguire lo stesso codice nello stesso momento.
  - ✓ Ciascun processo dispone di una propria copia dei registri e di una memoria dove conserva i dati necessari alla propria esecuzione.
    - Φ Ad es., nella memoria fisica è presente solo una copia dell' editor.
    - Φ La tabella delle pagine di ogni utente fa corrispondere gli stessi blocchi di memoria contenenti l'editor, mentre le pagine dei dati sono mappate su frame diversi.
  - ✓ Possono essere condivisi anche altri programmi di utilizzo frequente: compilatori, sistemi di finestre, database e così via.
  - ✓ Per essere condivisibile, il codice deve essere rientrante.
- 



# Esempio di pagine condivise



process  $P_1$

3
4
6
1

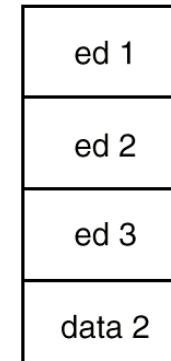
page table  
for  $P_1$



process  $P_3$

3
4
6
2

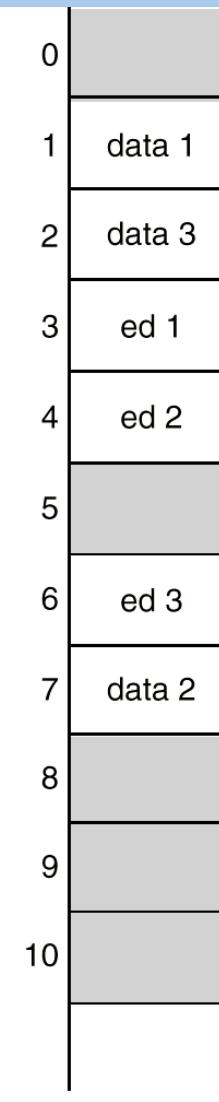
page table  
for  $P_3$

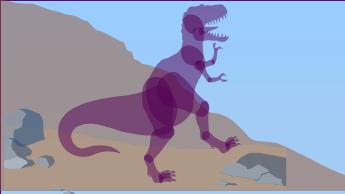


process  $P_2$

3
4
6
7

page table  
for  $P_2$



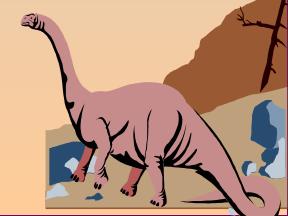
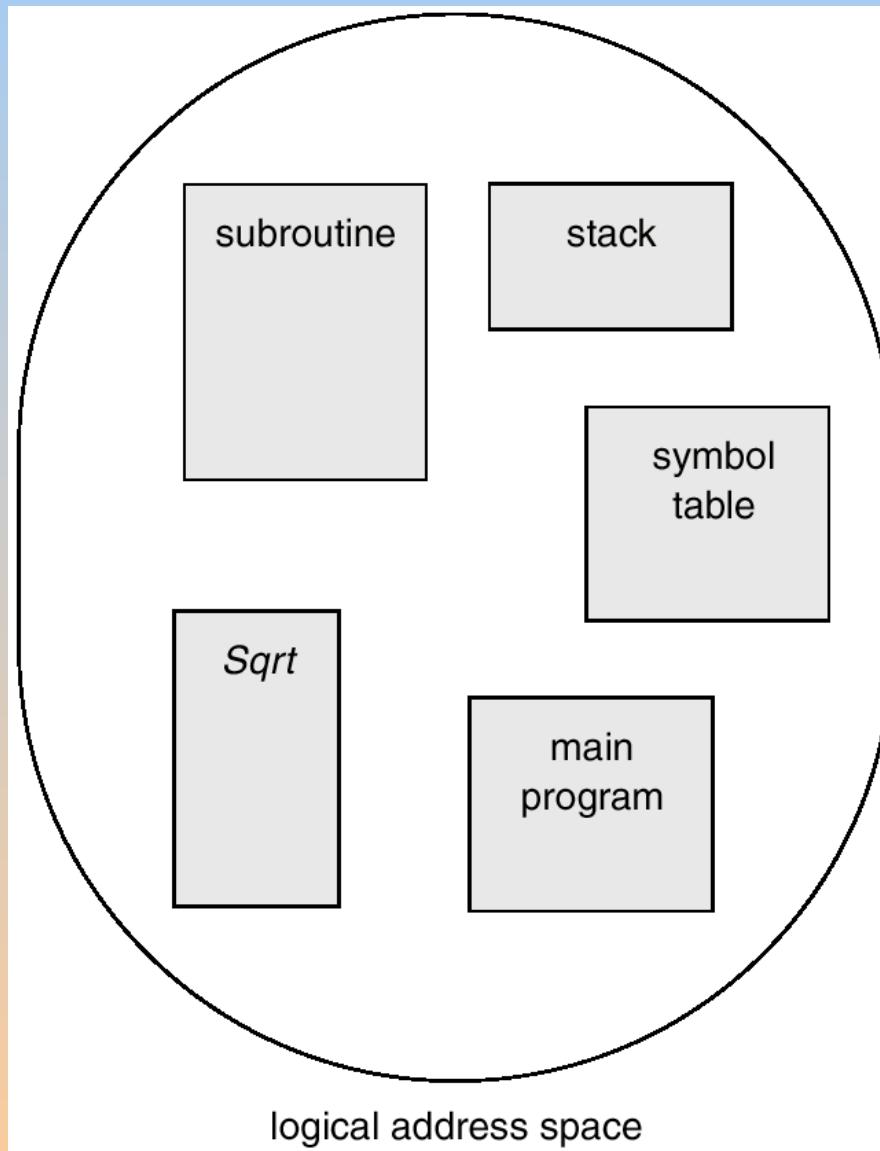


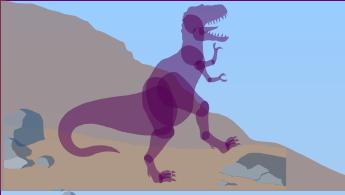
# Segmentazione

- ✓ E' uno schema di gestione della memoria che imita la visione che l'utente ha della memoria.
- ✓ Un programma è costituito da un insieme di *segmenti*.
- ✓ Un segmento è un'unità logica, come:
  - programma principale,
  - procedura,
  - funzione,
  - metodo,
  - oggetto,
  - variabili locali, variabili globali,
  - blocchi comuni,
  - stack,
  - symbol table, array,
  - etc.

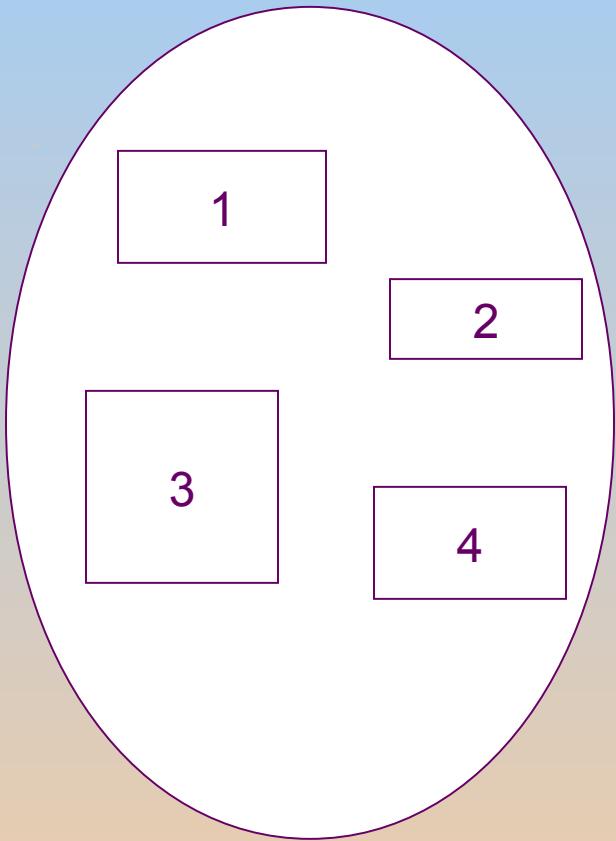


# Un programma dal punto di vista dell'utente





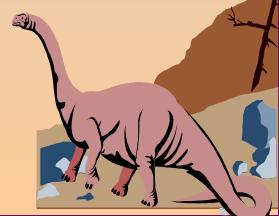
# Visione logica della segmentazione

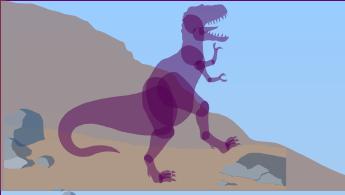


Spazio utente



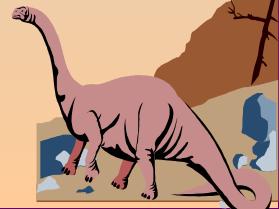
Spazio di memoria fisico





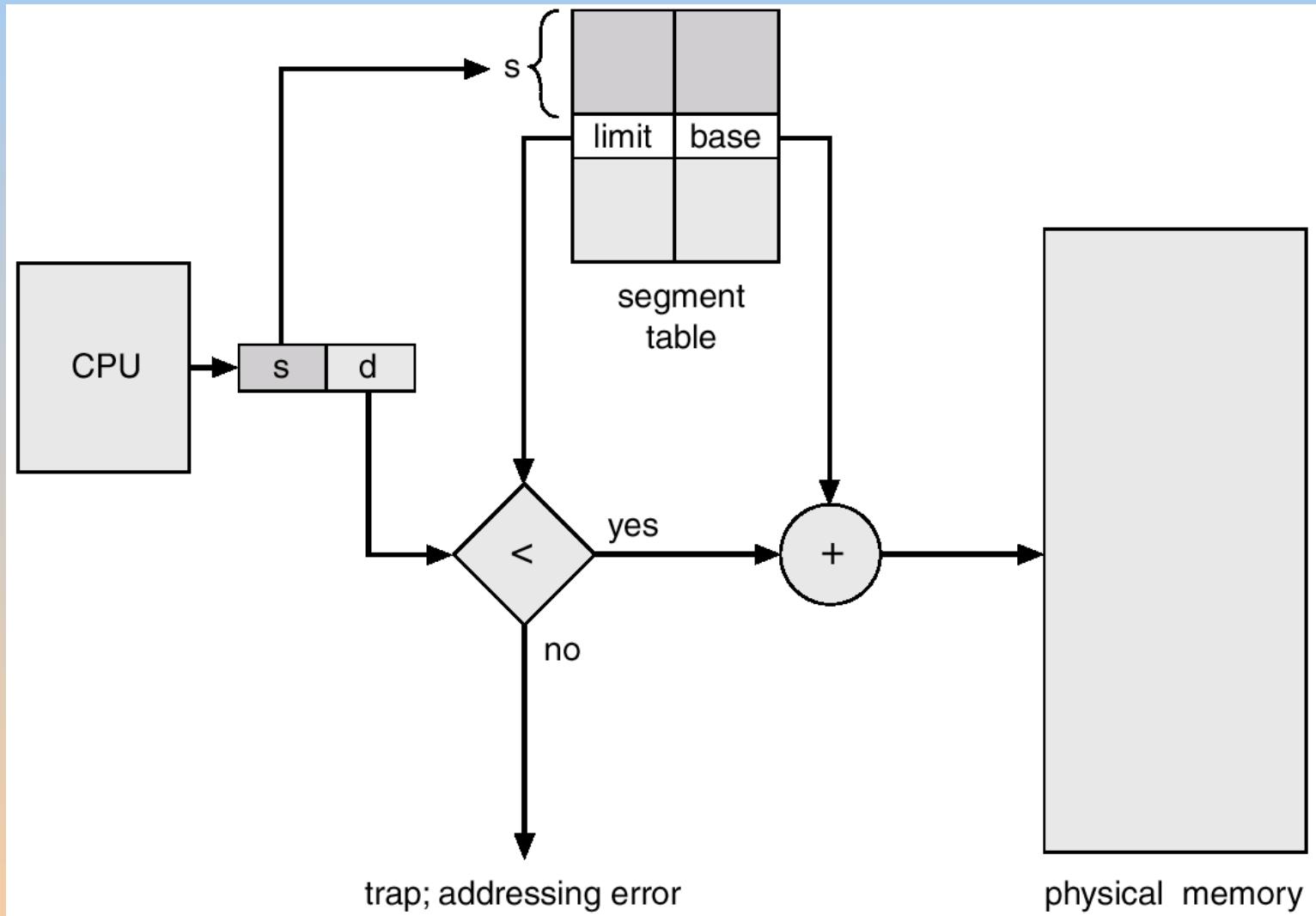
# Architettura di segmentazione

- ✓ L'indirizzo logico consiste di una tupla:
  - $\langle$ numero-di-segmento, scostamento $\rangle$ ,
- ✓ *Tabella dei segmenti* – mappa, in uno spazio ad una dimensione (memoria fisica), gli indirizzi logici.
- ✓ Ogni elemento della tabella è una coppia ordinata:
  - Φ base – contiene l'indirizzo fisico iniziale della memoria nel quale il segmento risiede.
  - Φ *limite* – contiene la lunghezza del segmento.
- ✓ *Segment-table base register (STBR)* - punta alla locazione in memoria della tabella dei segmenti
- ✓ *Segment-table length register (STLR)* - indica il numero di segmenti di un processo:
  - Il segmento numero s sarà legale se  $s < \text{STLR}$ .

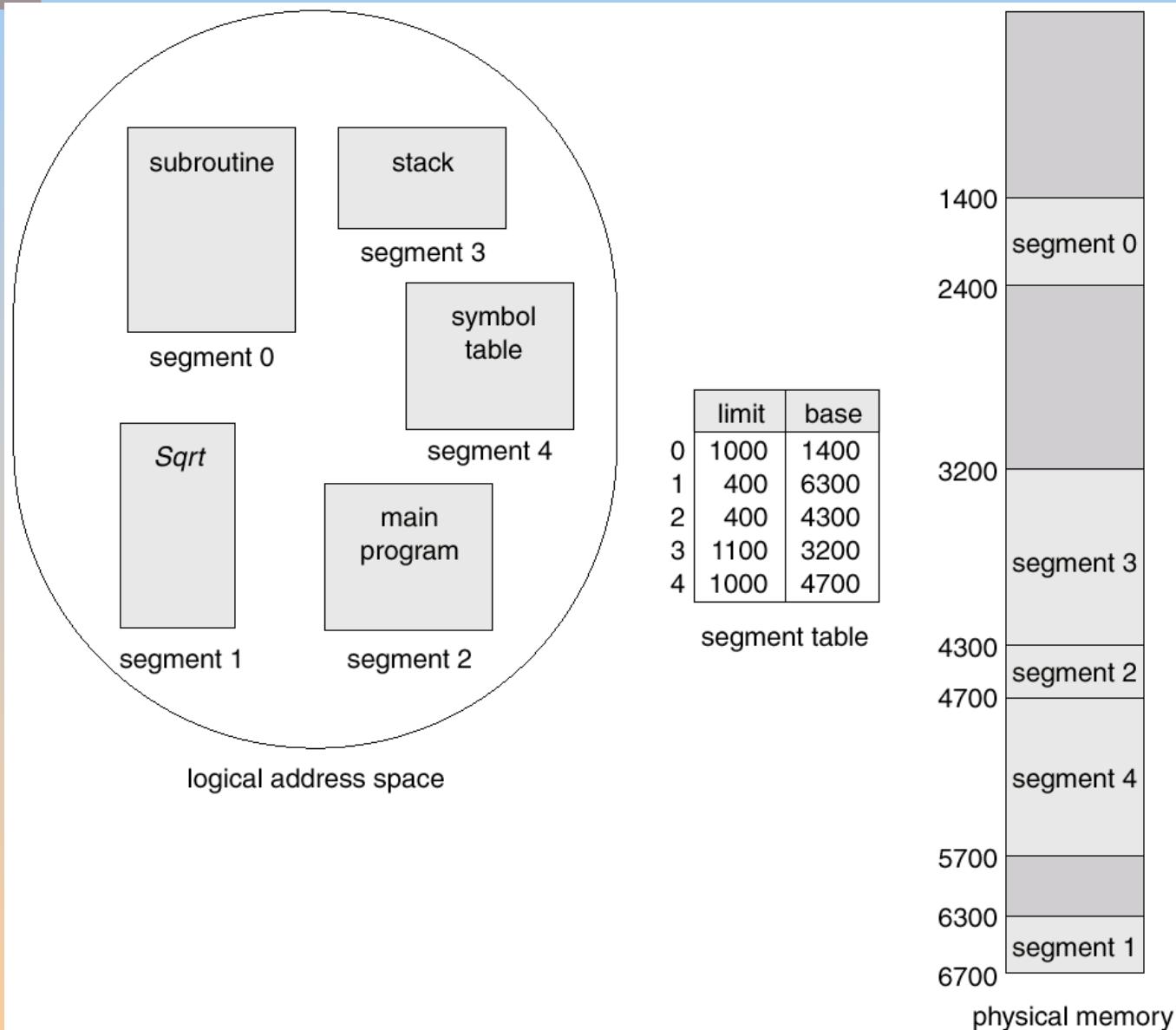




# Architettura di segmentazione (schema)



# Esempio di segmentazione





# Architettura di segmentazione (II)

- ✓ Rilocazione:

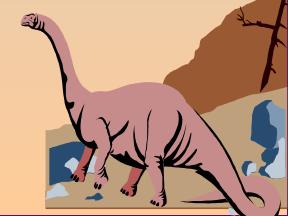
- Φ Dinamica
  - Φ Tramite la tabella dei segmenti

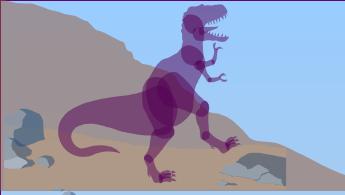
- ✓ Condivisione:

- Φ Segmenti condivisi
  - Φ Devono avere lo stesso *numero di segmento*

- ✓ Allocazione.

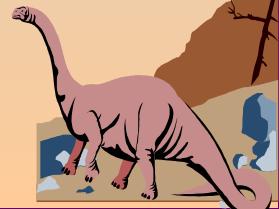
- Φ First fit / best fit
  - Φ Frammentazione esterna



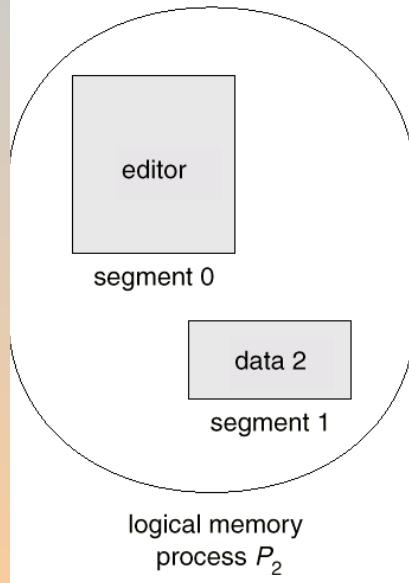
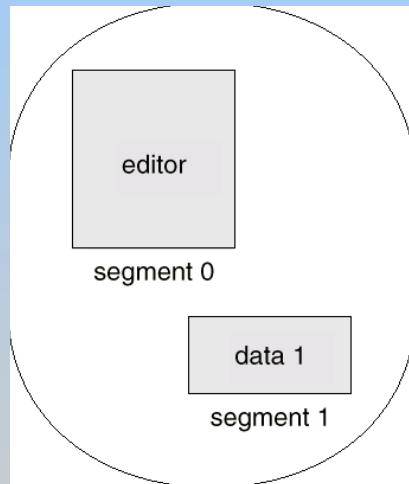


# Architettura di segmentazione (III)

- ✓ Protezione: Si può associare ad ogni elemento della tabella dei segmenti:
  - Φ bit di validità = 0  $\Rightarrow$  segmento illegale
  - Φ Privilegi di read/write/execute
- ✓ Quindi i bit di protezione sono associati ai segmenti.
- ✓ La condivisione del codice può essere effettuata al livello dei segmenti.
- ✓ Giacché la lunghezza dei segmenti può variare, l'allocazione con segmentazione della memoria è un problema di allocazione dinamica di memoria.



# Condivisione di segmenti



	limit	base
0	25286	43062
1	4425	68348

segment table  
process  $P_1$

	limit	base
0	25286	43062
1	8850	90003

segment table  
process  $P_2$

