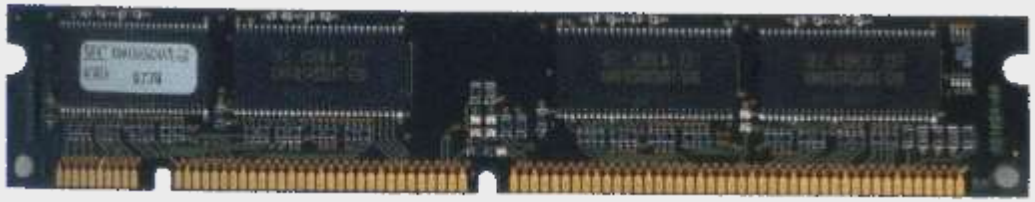
Gestione della Memoria Centrale



Un programma e i suoi dati al fine di essere eseguiti (run) devono essere, almeno parzialmente, presenti in memoria centrale

Il gestore della memoria:

- si occupa di allocare memoria fisica ai processi,
- in un ambiente multiprogrammato deve garantire la protezione dei dati, impedendo ai processi attivi di sconfinare nello spazio di indirizzamento di altri processi,
- In un ambiente multiprogrammato deve permettere meccanismi di condivisione affinché i processi cooperanti possano accedere ad aree comuni di memoria.

L'utilizzo globale delle risorse e le prestazioni di un calcolatore vengono influenzate dalle prestazioni del *modulo di gestione della memoria* sia in funzione della sua efficienza nell'allocare memoria, sia per l'influenza che può avere sullo *scheduler.*

Introduzione

La cpu preleva le istruzioni dalla memoria centrale sulla base del contenuto del PC.

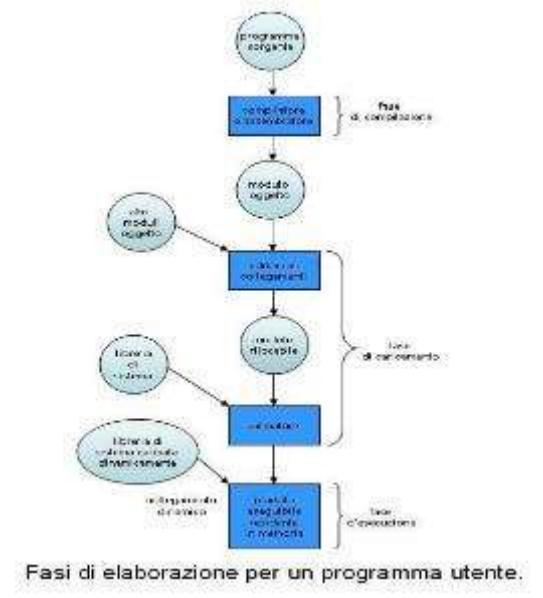
Il programma prima di essere eseguito attraversa diversi stadi (alcuni opzionali):

- programma sorgente con indirizzi simbolici (es.: *a, x, somma, contatore*)
- Compilazione: generazione di indirizzi rilocabili,
- Caricatore (loader) fa corrispondere a questi indirizzi gli indirizzi assoluti, dopo l'intervento dell'editor dei collegamenti "linkage editor"

Modulo sorgente - indirizzi simbolici (es. : x)

Modulo compilato - indirizzo rilocabile (es.: +14 bytes dall'inizio di questo modulo)

Modulo Eseguibile- indirizzo assoluto (es.: 74000+14= 74014)



Fasi di elaborazione per un programma utente.

GENERAZIONE DI INDIRIZZI DI MEMORIA (ASSOLUTI) A TEMPO DI:

COMPILAZIONE Si genera codice assoluto che deve risiedere in memoria (es. nell'MS-DOS file .COM).

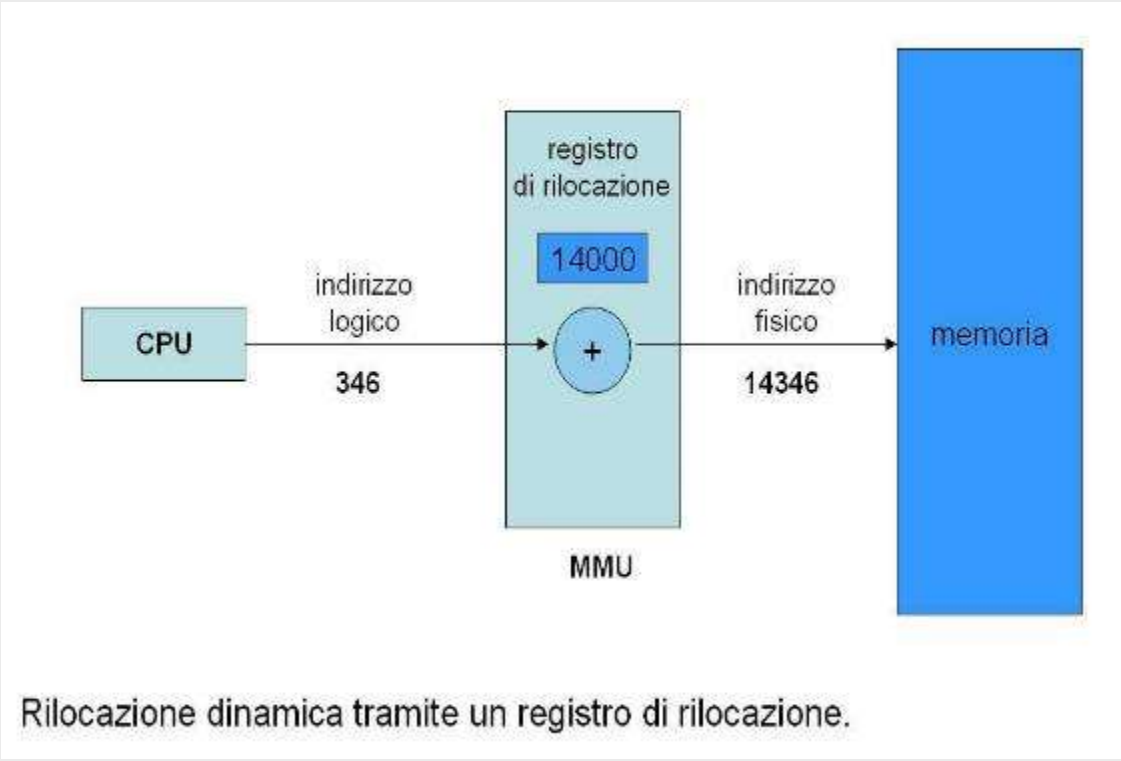
CARICAMENTO Se non si sa dove deve un programma deve risiedere viene generato codice rilocabile, in fase di caricamento si fa l'associazione agli indirizzi assoluti.

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

Introduzione

- **Indirizzo Logico (virtuali): indirizzo generato dalla cpu**
- **Indirizzo Fisico: indirizzo reale (MAR) visto dalla memoria**

MMU (Memory Management Unit): modulo che si occupa della traduzione.



Rilocazione dinamica tramite un registro di rilocazione.

Allocazione della Memoria

I moduli di gestione della memoria si differenziano in base al tipo di allocazione della stessa, che può essere:

CONTIGUA ***NON CONTIGUA***

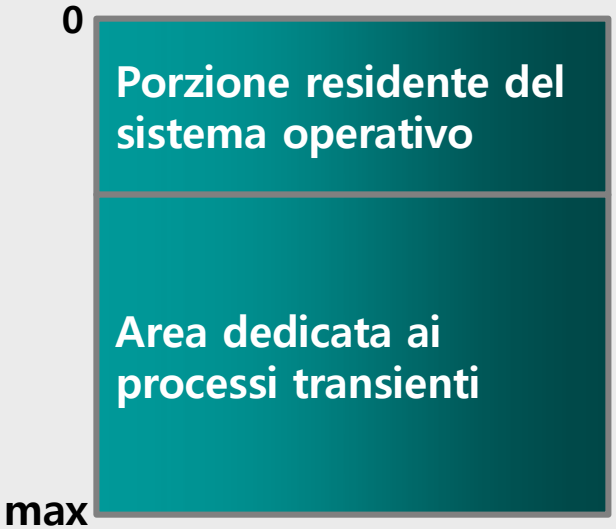
Allocazione Contigua: la memoria viene allocata in modo tale che ciascun oggetto occupa un insieme di locazioni i cui indirizzi sono strettamente consecutivi. Esistono vari tipi di allocazione contigua:

- ***MONOALLOCAZIONE***
- ***PARTIZIONAMENTO STATICO***
- ***PARTIZIONAMENTO DINAMICO***
- ***SEGMENTAZIONE***

Monoallocazione (Monitor Monoprocesso)

La **memoria** viene **suddivisa in due aree contigue**:

- la prima allocata permanentemente ad una porzione residente del sistema operativo (Monitor)
- la seconda assegnata ai processi in esecuzione, processi utente o porzioni non residenti del sistema operativo, per il tempo necessario al completamento.



Il sistema operativo:

- tiene traccia della prima e dell'ultima locazione disponibile per i processi transienti;
- viene posto ad uno dei due estremi della memoria;
- ingloba i vettori di interrupt;

Schema viene generalmente utilizzato da sistemi operativi monoprocesso per microcomputer.

Monoallocazione (Monitor Monoprocesso)

Alla richiesta di esecuzione di un programma, il sistema operativo:

- si assicura che le dimensioni del processo siano compatibili con la memoria disponibile;
- conferisce il controllo al processo utente fino al suo completamento o ad eventuali condizioni di errore;
- al termine del processo la memoria viene liberata e può essere assegnata ad un altro processo in attesa

Raramente un monitor monoprocesso supporta meccanismi di protezione tra processi utente in quanto in ogni istante vi può essere al massimo un solo processo residente in memoria. Gli eventuali meccanismi si riferiscono alla protezione del codice del sistema operativo da eventuali sconfinamenti del processo transiente in esecuzione.

La protezione del sistema operativo dai processi utente è spesso effettuata mediante supporti hardware:

- ***Registri Barriera*;**
- ***Diritti di accesso mediante bit di protezione*;**
- ***Sistema operativo in memoria a sola lettura*.**

Monoallocazione
Meccanismi di protezione

Registri barriera: usati per tracciare un confine tra le aree dei processi di sistema e dei processi transienti. Nel registro viene memorizzata la prima locazione disponibile al processo e il sistema operativo effettua i controlli di sconfinamento.

Diritti di accesso mediante bit di protezione: Ad ogni parola di memoria viene associato un bit di protezione (1 nelle zone che contengono il sistema operativo, 0 nelle restanti). I processi utente accedono solo a parole con bit 0, mentre il sistema operativo ha accesso illimitato.

Sistema operativo in memoria a sola lettura: non è solitamente utilizzato per la sua scarsa flessibilità e impossibilità di correggere o aggiornare il codice del sistema operativo.

***Condivisione del codice e dei dati*:** non ha molto senso negli ambienti monoprocesso e raramente viene supportata. Tuttavia programmi utente possono condividere dati mediante accordi interni, ponendoli in locazioni di memoria che non vengono sovrascritte durante l'esecuzione di processi partecipanti, o mediante file.

Monoallocazione
Prestazioni

Difetti:

- Assenza di multi-programmazione: abbassamento dell'efficienza della memoria e della CPU;
- La memoria può risultare sovradimensionata per la maggioranza dei processi;
- Processi di dimensione più grande della memoria non possono essere eseguiti, oppure richiedono particolari suddivisioni del codice (overlay);
- I programmi tendono ad essere ottimizzati rispetto alla dimensione della memoria, comportando spesso sacrifici di funzionalità e velocità.

Pregi:

- Basso costo di progettazione del modulo di gestione della memoria;
- Contenuti supporti hardware specifici per la gestione della memoria;

Utilizzato per micro-computer dedicati permanentemente a specifiche applicazioni.

Partizionamento Statico

Supportare la multiprogrammazione:

dividere la memoria in diverse partizioni, ciascuna delle quali può essere allocata a un processo diverso.

Partizionamento statico:

la suddivisione della memoria viene fatta “fuori linea”

le partizioni hanno dimensioni fisse

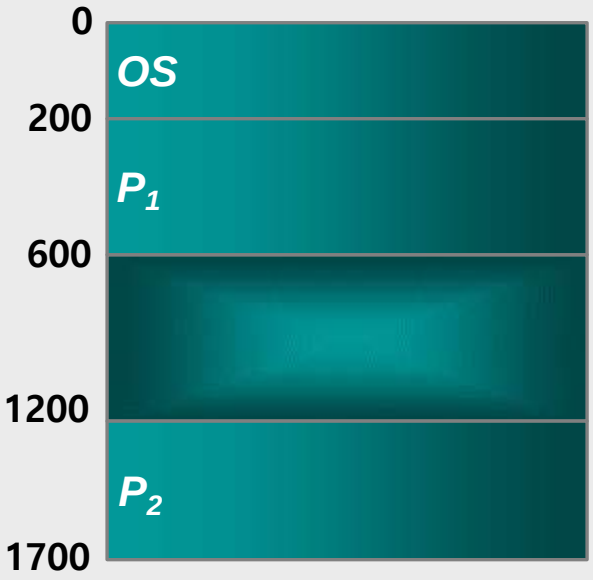
Il numero e la dimensione di ciascuna partizione viene determinato durante la generazione del sistema considerando:

- capacità della memoria fisica disponibile;
- grado desiderato di multiprogrammazione;
- dimensioni dei processi più frequentemente eseguiti.

Partizionamento Statico

Il sistema operativo deve tener traccia delle partizioni definite. Lo stato corrente delle partizioni ed i loro attributi vengono raccolti in una struttura dati chiamata ***tabella di descrizione delle partizioni*** (TDP).

L'unico campo variabile nella TDP è lo stato della partizione che indica in ogni istante se la partizione è allocata o no. Gli altri campi contengono valori definiti al momento del partizionamento.



Numero partizione	Base partizione	Dimensione partizione	Stato partizione
0	0 kB	200 kB	Allocata
1	200 kB	400 kB	Allocata
2	600 kB	600 kB	Libera
3	1200 kB	500 kB	Allocata

Esempio di TDP

Partizionamento Statico

Caricamento di un processo:

SO consulta la TDP. Se il test è positivo, il campo "stato della partizione" viene impostato ad "Allocata" e l'immagine del processo viene caricata nella corrispondente partizione. Viene aggiornato il campo nel PCB.

Metodi più comuni per la ricerca di una partizione libera:

- ***First fit*** **:** allocare la prima partizione libera sufficientemente grande per contenere il processo; molto veloce.
- ***Best fit*** **:** allocare la più piccola delle partizioni libere che contenga il processo; prima dell'allocazione devono essere controllate tutte le locazioni libere; ottimizzazione dello spazio di memoria, l'algoritmo produce la più piccola frammentazione interna.
- ***Worst fit***

Terminazione di un processo:

Deallocazione della partizione.

Aggiornamento della TDP.

Il funzionamento del gestore della memoria e dello scheduler sono strettamente correlati:

- lo scheduler determina quali processi sono pronti, e quindi residenti in memoria, in attesa della CPU;
- il gestore della memoria può decidere di "sfrattare" un processo in memoria per liberare una partizione.

Partizionamento Statico

Cause di mancata disponibilità di una partizione:

1) non vi sono partizioni di memoria sufficientemente grandi per contenere il processo entrante;

   Cause:

   - errore di scelta della dimensione delle partizioni;
   - Processo eccezionalmente più grande della norma.

   Soluzione:

   la dimensione delle partizioni deve essere ridefinita, oppure il programma di origine deve essere progettato utilizzando uno schema di *overlay*.

2) tutte le partizioni sono allocate;

   Soluzioni:

   - attesa di una partizione libera;
   - *swapping* di un processo.

Partizionamento Statico

Swapping: rimozione dalla memoria di un processo sospeso e caricamento di un nuovo processo.

1. Lo scheduler decide l'introduzione in memoria di un nuovo processo
2. Il gestore dello swapping ne sceglie uno da rimuovere in base a:
   - dimensione della partizione richiesta;
   - priorità del processo;
   - tipo di evento atteso dal processo;
   - tempo già trascorso in memoria.

Un processo residente in memoria, già parzialmente eseguito, comprende:

- codice eseguibile;
- dati già elaborati;
- stack;
- registri di stato;
- file aperti;
- descrittore del processo.

Al momento dello swapping questi oggetti vengono registrati in un file su hard disk detto *file di swapping*.

Partizionamento Statico

Tutti i processi che hanno subito uno swapping possono essere mantenuti:

- in un file unico per tutto il sistema
  - -creato al momento dell'inizializzazione del sistema
  - - collocato su una periferica veloce di memorizzazione secondaria
  - -L'indirizzo e la dimensione sono solitamente statici in modo da beneficiare di un indirizzamento diretto su disco
  - - Parametro critico: dimensione:
    - troppo grande spreca spazio su dischi veloci
    - troppo piccolo può rendere indisponibile l'operazione di swapping.

- in file separati, ciascuno associato ad un singolo processo.
  - - creato dinamicamente al momento della creazione del processo o staticamente al momento della preparazione del programma.

  Vantaggi:
  - Eliminazione del problema del dimensionamento del file unico;
  - Nessuna restrizione sul numero di processi attivi.

  Svantaggi:
  - Maggior spreco di spazio;
  - Maggiori tempi di accesso ai file distribuiti sul disco.

Partizionamento Statico

Uno swapping efficiente richiede che i processi siano *rilocabili dinamicamente*:

- il processo può iniziare l'esecuzione in qualunque area di memoria;
- può essere spostato in un'altra area di memoria;
- il calcolo degli indirizzi viene effettuato dinamicamente ad esempio con l'uso di un *registro base*.

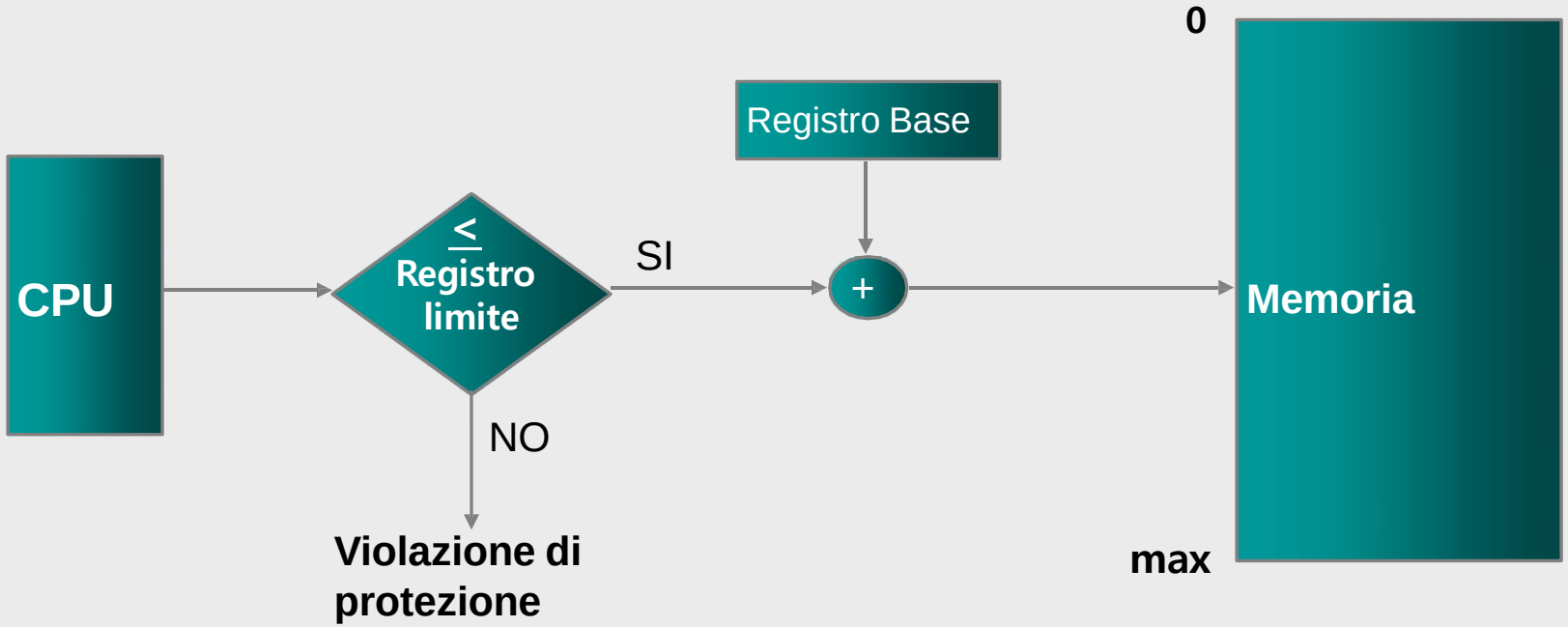
Processi non rilocabili dinamicamente sono legati alle partizioni in cui iniziano l'esecuzioni e rendono inefficiente lo swapping.

L'integrità di un sistema multiprogrammato dipende anche dalla sua capacità di garantire l'isolamento tra spazi di indirizzi separati.

-Registro base (per la rilocazione)
-Registro limite (per la protezione)

Partizionamento Statico: protezione

Il registro base contiene l'indirizzo più basso del processo.
Il registro limite contiene il valore dell'indirizzo virtuale più alto contenuto dal programma.



Partizionamento Statico: Condivisione

Condivisione controllata di dati e codice tra processi cooperanti.
I sistemi a partizioni fisse basano i propri meccanismi sull'isolamento degli spazi di indirizzamento => difficoltà di condivisione

Metodi di condivisione:

1. Affidamento degli oggetti condivisi al sistema operativo.
   Svantaggi:
   - l'area del sistema operativo dovrebbe poter variare dinamicamente;
   - le routine dei processi utente devono poter essere "linkate" al sistema operativo dinamicamente.

2. Ogni processo possiede una copia identica dell'oggetto condiviso, che usa e diffonde gli aggiornamenti. Svantaggi:
   - Se il sistema supporta lo swapping, uno o più processi potrebbero non essere in memoria e quindi non ricevere gli aggiornamenti

3. Collocare i dati in una partizione comune dedicata. Il sistema operativo considera come violazione i tentativi dei processi partecipanti alla condivisione di accedere a zone di memoria esterne alle rispettive partizioni. Se il sistema usa registri base e limite sono necessari accorgimenti per indirizzare partizioni che potrebbero non essere contigue.

Partizionamento Statico: Conclusioni

Partizionamento statico:

- Metodo semplice di gestione della memoria
- Supporta la multiprorammazione;
- Supporto hardware modesto;
- Si adegua ad ambienti statici con carico predicibile di lavoro come ambienti in cui vengono eseguiti solo applicativi, controllo di processi e sistemi di tipo bancario.

Svantaggi:

- *frammentazione interna*;
- può richiedere progettazione dei programmi con schemi ad overlay;
- non si adatta a contenere oggetti come *stack* che possono crescere;
- il numero di partizioni fissato limita il grato di multiprogrammazione.

Partizionamento Dinamico

- Sistema delle partizioni analogo a quello del partizionamento statico.
- Il gestore della memoria crea ed assegna partizioni di *dimensione variabile dinamicamente* in base alle richieste dei processi.

Richiesta di partizione:

- Il gestore della memoria ricerca una zona di memoria libera contigua di dimensione sufficiente.
- La partizione viene creata registrando la sua base, la sua dimensione e il suo stato (Allocata) nella TDP o in una tabella equivalente.
- L'eventuale parte rimanente di memoria libera viene restituita all'insieme della memoria libera e posta a disposizione del modulo di allocazione.

Terminazione di un processo o swapping:

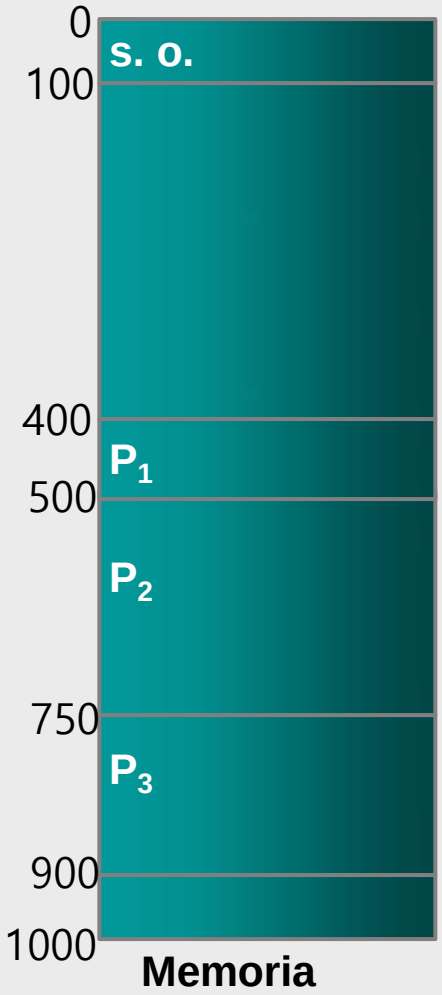
- il gestore della memoria restituisce lo spazio liberato all'insieme di aree di memoria libere e cancella la riga corrispondente nella TDP.

Il gestore della memoria può creare e allocare partizioni finché non viene esaurita la memoria fisica o finché non viene raggiunto il massimo grado di multiprogrammazione.

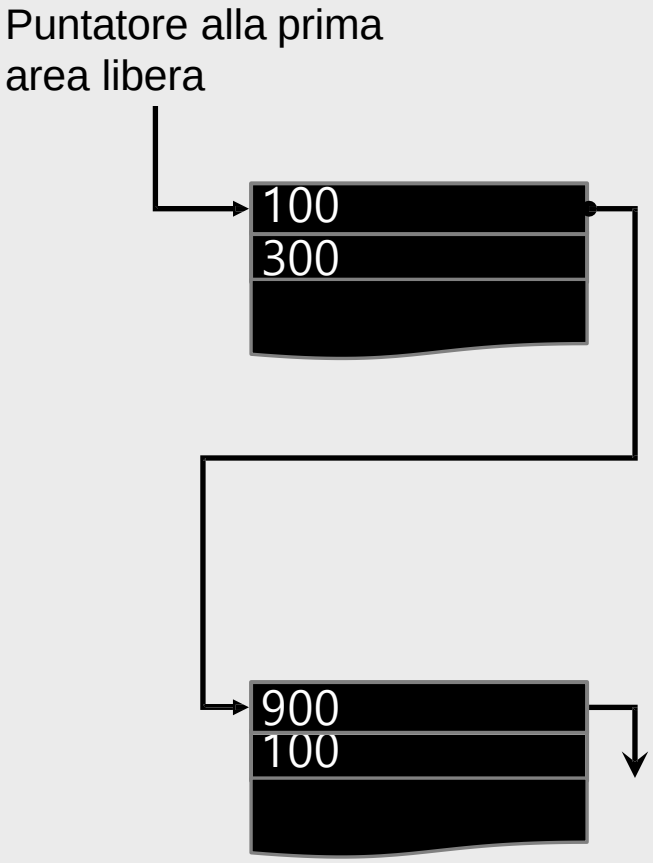
Partizionamento Dinamico: Esempi di partizioni

0	0	100	Allocata
1	-	-	-
2	400	100	Allocata
3	500	250	Allocata
4	750	150	Allocata
5	-	-	-
6	-	-	-
7	-	-	-

TDP



Memoria



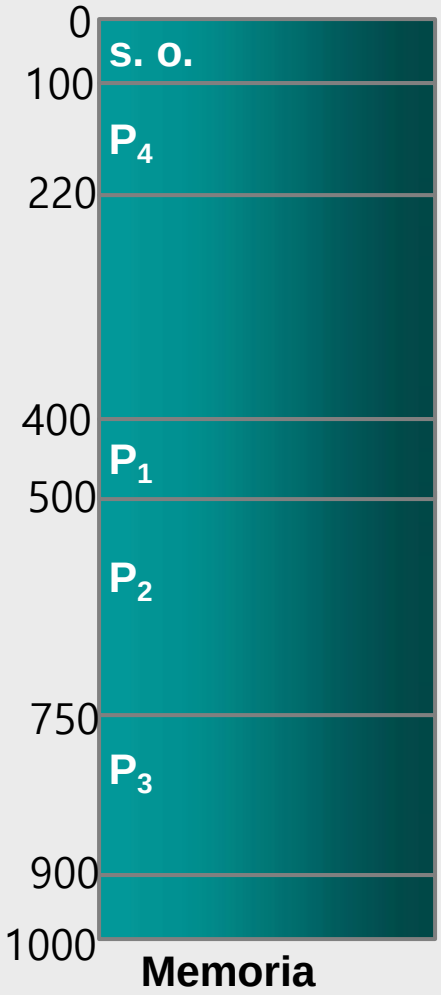
Lista aree libere

Partizionamento Dinamico: Esempi di partizioni

Richiesta di un'area pari a 120KB per il P4

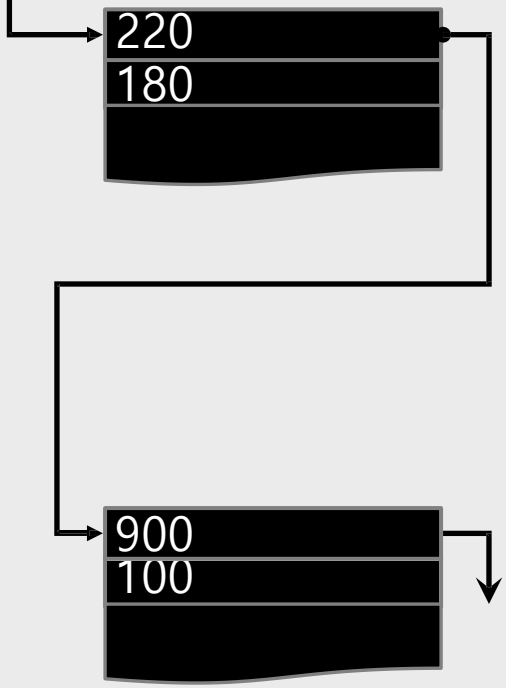
0	0	100	Allocata
1	100	120	Allocata
2	400	100	Allocata
3	500	250	Allocata
4	750	150	Allocata
5	-	-	-
6	-	-	-
7	-	-	-

TDP



Memoria

Puntatore alla prima area libera



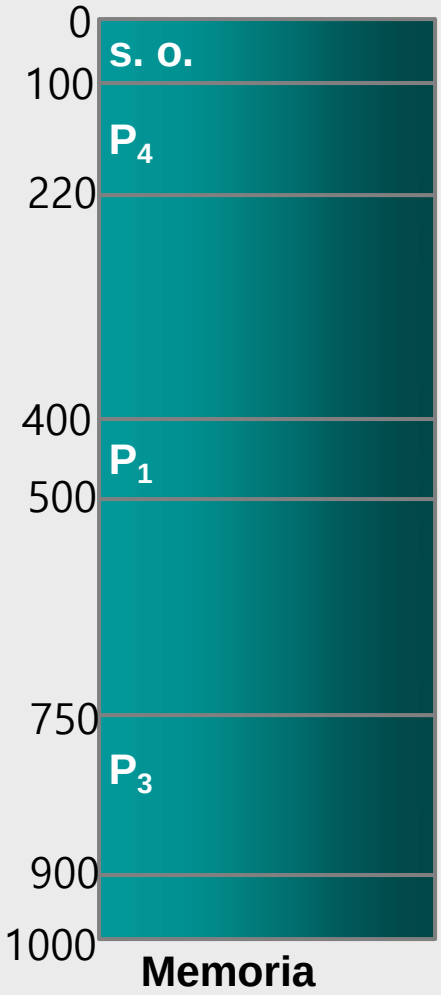
Lista aree libere

Partizionamento Dinamico: Esempi di partizioni

La partizione occupata da P2 viene liberata

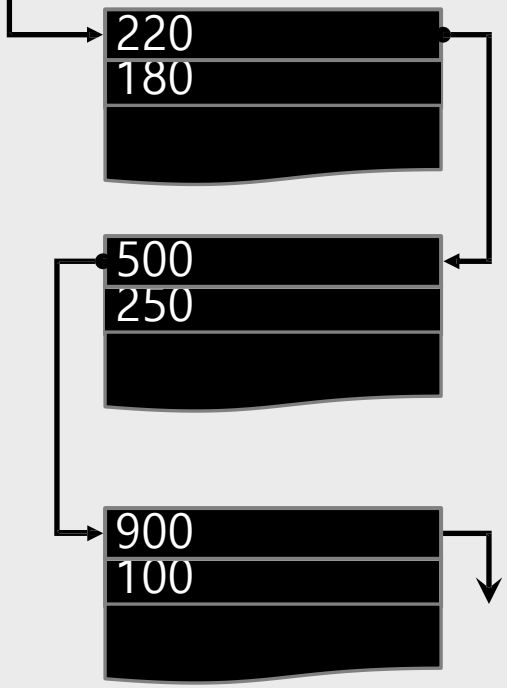
0	0	100	Allocata
1	100	120	Allocata
2	400	100	Allocata
3	-	-	-
4	750	150	Allocata
5	-	-	-
6	-	-	-
7	-	-	-

TDP



Memoria

Puntatore alla prima area libera



Lista aree libere

Partizionamento Dinamico

Algoritmi per la selezionare un'aria di memoria:

- ***First-fit***: termina la ricerca quando viene individuato il primo blocco di memoria sufficientemente grande da contenere il processo.
  - tempo di ricerca molto veloce
- ***Next-fit***: simile al first fit, ma il puntatore alla lista delle aree libere, dopo avere effettuato un'allocazione, viene salvato cosicché la ricerca successiva continua da dove si era fermata la precedente
  - tempo di ricerca molto veloce
- ***Best-fit***: viene utilizzato il blocco di memoria libero più piccolo che può contenere il processo. Vengono prodotte le parti di "buco" inutilizzate più piccole.
- ***Worst-fit***: viene utilizzato il blocco di memoria libero più grande. Vengono prodotti "buchi inutilizzati" grandi.

Partizionamento Dinamico

Qualsiasi algoritmo di selezione delle aree crea “buchi” di memoria inutilizzabile.

FRAMMENTAZIONE ESTERNA: creazione di aree di memoria libere di piccole dimensioni non contigue e impossibilità di allocazione di memoria richiesta da un processo, pur contenendo la memoria globalmente un’area di dimensione sufficiente.

COMPATTAZIONE DELLA MEMORIA: spostare i processi residenti in memoria in modo da creare partizioni libere più grandi. Questa operazione generalmente richiede un grande costo di CPU. La compattazione può essere effettuata:

1. Continuamente:
   - la memoria viene compattata ogni volta che un processo libera un’area;
   - la frammentazione è ridotta al minimo;
   - grande costo di CPU.

2. Su necessità:
   - la compattazione viene effettuata quando non si riesce ad allocare memoria ad un processo richiedente;
   - un controllo preliminare verifica se il totale dell’area liberabile è sufficiente a contenere il processo.

Partizionamento Dinamico

La compattazione può essere effettuata in due modi:

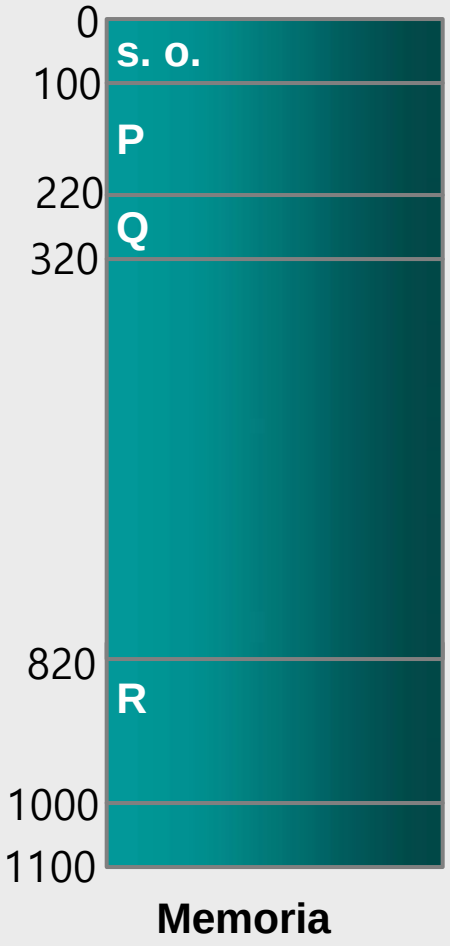
1. Spostamento selettivo incrementale:
   - ricerca di una strategia ottima di movimento per compattare al meglio la memoria;
   - consente di risparmiare tempo negli spostamenti;
   - Poco utilizzata perché richiede un sovraccarico notevole.

2. Spostamento globale:
   - tutte le partizioni vengono rilocate ad uno dei due estremi della memoria;
   - il tempo di trasferimento delle partizioni non è ottimizzabile;
   - non richiede strategie particolari.

NB: compattazione possibile solo se la rilocazione è dinamica (run time) non possibile se la rilocazione è statica o fatta a tempo di compilazione o caricamento

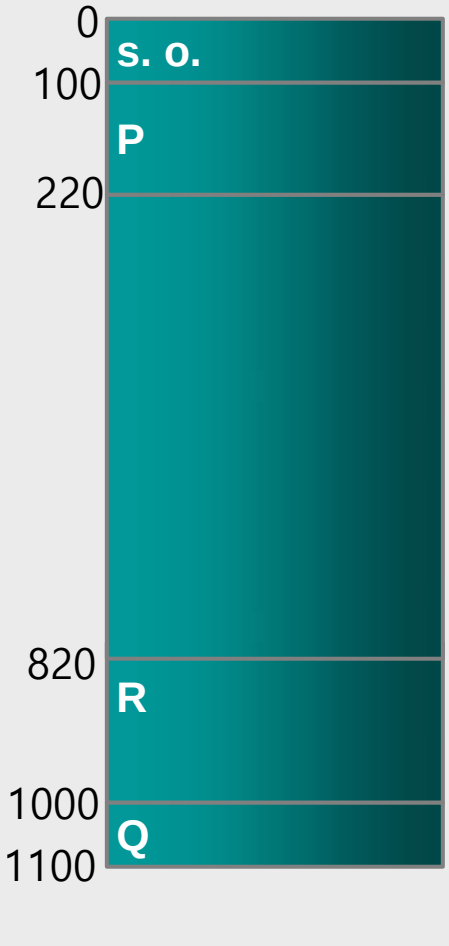
Partizionamento Dinamico
Esempi di compattazione

Situazione iniziale:

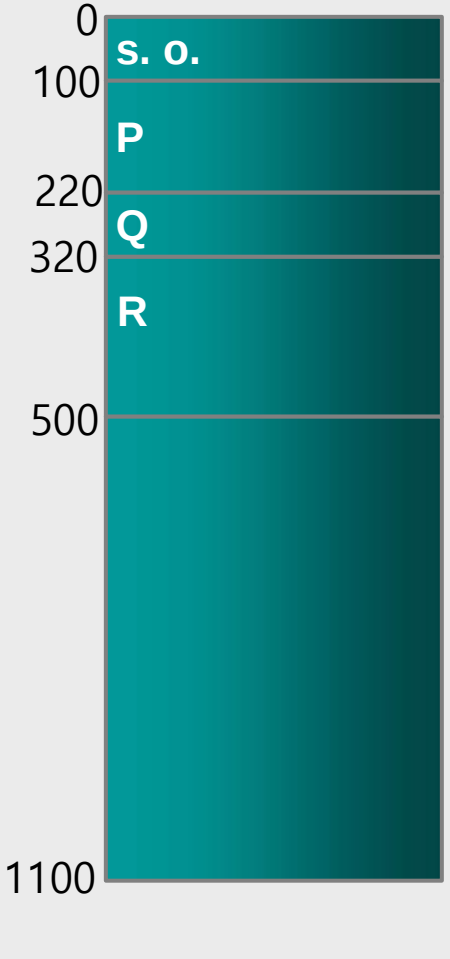


Memoria

Spostamento selettivo



Spostamento globale



Partizionamento Dinamico

Protezione e condivisione analoghe al partizionamento statico.

Nel partizionamento dinamico è possibile una particolare forma di condivisione:

- a due partizioni contigue è consentito sovrapporsi, mettendo un'area in comune;
- forma di condivisione molto restrittiva, può avvenire solo tra due processi.

CONCLUSIONI

Il partizionamento dinamico:

- richiede un supporto hardware modesto, analogo al partizionamento statico;
- la differenza tra i due schemi risiede sostanzialmente nel software;
- elimina la frammentazione interna ma produce quella esterna che può essere eliminata con la compattazione;
- si adegua ad ambienti con carico non predicibile di lavoro come ad esempio sviluppo di software.

Segmentazione

Schema di gestione della memoria che supporta la visione che l'utente ha della memoria

Un programma è una collezione di segmenti.
Un segmento è una unità logica come:

main program,
procedure,
function,
method,
object,
local variables, global variables,
common block,
stack,
symbol table, arrays

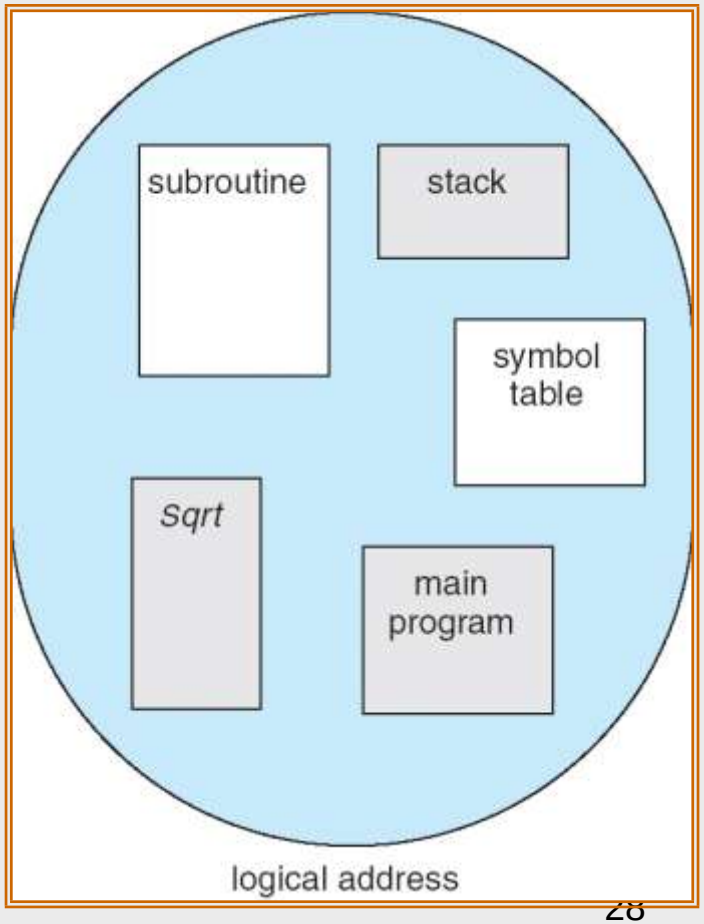
Ciascuno di questi oggetti può essere posto in un segmento diverso di dimensioni differenti.

Ogni entità (segmento) è identificata da un nome.

Le entità possono essere presenti in memoria anche in maniera non contigua.

L'indirizzo logico si compone di due parti:

1. **il nome dell'entità identificato dal suo numero**
2. **lo scostamento all'interno del segmento**



Segmentazione

La segmentazione condivide alcune proprietà:

- **degli schemi di allocazione contigua** relativamente ad un singolo segmento: i dati di ogni singola entità logica devono essere posti in un'area contigua di memoria;
- **degli schemi di allocazione non contigua** relativamente all'intero spazio di indirizzamento del processo: blocchi logici diversi possono essere messi in segmenti non contigui.

- La raccolta degli oggetti in segmenti viene predisposta dal programmatore;
- Ciascun segmento inizia all'indirizzo virtuale zero;
- Ogni segmento ha un nome che viene poi tradotto in un indirizzo in fase di caricamento in memoria;
- Un singolo dato all'interno del segmento viene identificato dallo "spiazzamento" relativo all'inizio del segmento cui appartiene.

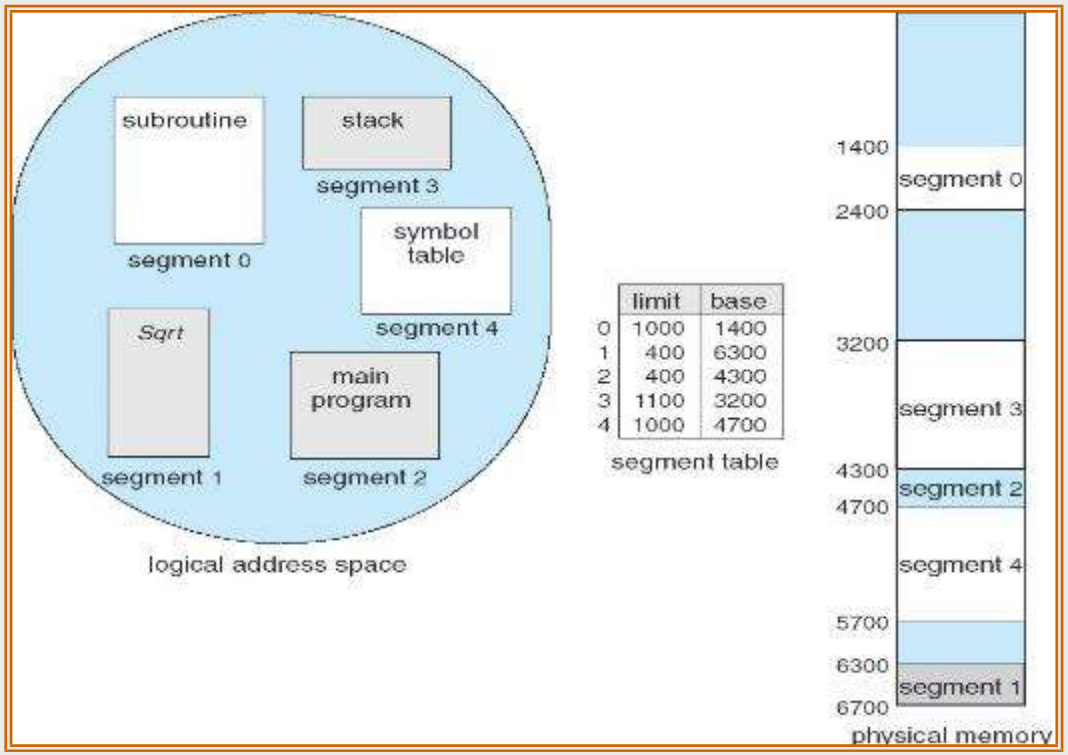
L'**indirizzamento** nella segmentazione è di tipo **bidimensionale**, la designazione univoca di un dato o di una istruzione richiede il nome del segmento e lo spiazzamento all'interno del segmento.

La memoria fisica mantiene invece l'indirizzamento lineare: è necessario un meccanismo per la traduzione degli indirizzi bidimensionali di segmenti virtuali in indirizzi fisici.

Segmentazione

Caricamento di un processo segmentato:

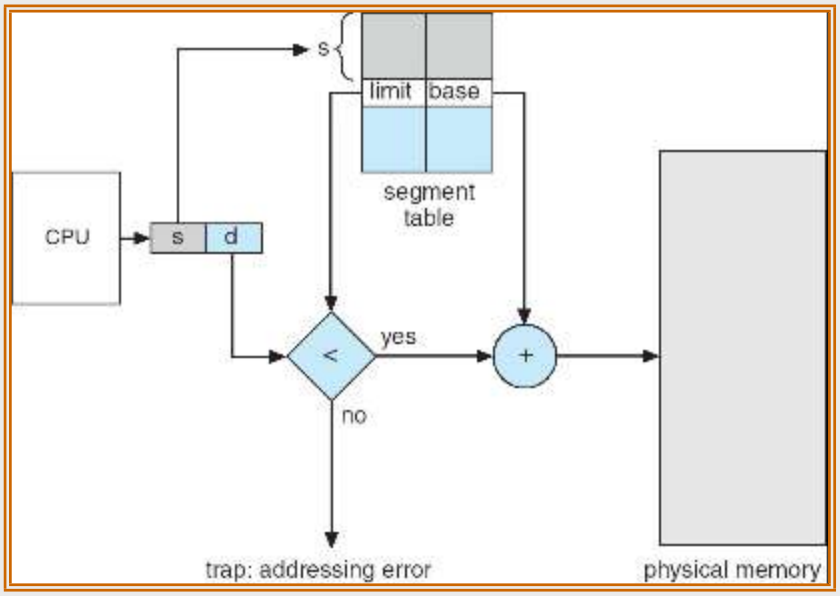
- il S.O. cerca di allocare N partizioni adatte per gli N segmenti in cui è suddiviso il processo;
- il S.O. crea un *descrittore di segmento* registrandovi l'indirizzo fisico in cui è stato posto il segmento e la sua dimensione;
- l'insieme dei descrittori di segmento di un processo viene raccolto nella *tabella dei descrittori di segmento* (segment table).



- ***base*** *– indirizzo fisico di inizio del segmento in memoria*
- ***limit*** *– specifica la lunghezza del segmento*

Segmentazione

TRADUZIONE DEGLI INDIRIZZI



- il *segment* (s) viene utilizzato per ritrovare l'indirizzo fisico del segmento nella segment table (base)
- Il *displacement* (d) viene sommato all'indirizzo fisico del segmento per ottenere l'indirizzo fisico richiesto

Due ulteriori registri:

- **Segment-table base register (STBR)** punta alla segment table in memoria
- **Segment-table length register (STLR)** indica il numero di segmenti utilizzati da un programma;

Possibilità di controllo di accesso a segmenti non assegnati ad un processo

NB: Quando un processo subisce uno swapping, al ritorno in memoria va aggiornata la sua segment table.

Segmentazione

Considerazioni:

La **traduzione** di ciascun indirizzo bidimensionale virtuale richiede **due accessi in memoria**:

1. alla segment table per tradurre l'indirizzo del segmento da virtuale a fisico;
2. per accedere fisicamente alla locazione richiesta.

⇒raddoppia il tempo per l'accesso ad una locazione di memoria.

Soluzione:

Se i descrittori di segmento sono pochi, o alcuni sono utilizzati più frequentemente, conviene specificare il loro caricamento in opportuni registri hardware.

Famiglia INTEL : CS: Code Segment; DS: Data Segment; SS: Stack Segment; ES: Extra Segment.

E' possibile definire **diritti e modalità di accesso per ogni "tipo" di segmento**. Esempio:

- accesso in modalità "execute" (o anche "read-only") al segmento codice;
- accesso in modalità "read/write" al segmento stack;
- accesso "read-only" o "read/write" al segmento dati.

I diritti di accesso possono essere registrati in un campo della TDS.

Segmentazione

CONDIVISIONE:

- Gli oggetti condivisi sono collocati in segmenti dedicati e separati.
- Il segmento condiviso può essere mappato, attraverso la segment table, nello spazio di indirizzamento virtuale di tutti i processi autorizzati ad accedervi.
- Lo spiazzamento interno di un dato risulta identico per tutti i processi che lo condividono.

COLLEGAMENTO DINAMICO:

- Caricamento di una procedura, ad esempio una libreria, in fase di esecuzione di un processo e solo su sua richiesta.
- Lo spazio in memoria per una procedura viene occupato solo se e quando tale procedura viene richiesta.
- La segmentazione consente di aggiungere nuovi segmenti al processo richiedente, aggiornando la segment table e i registri Segment-table base register (STBR) Segment-table length register (STLR)

Segmentazione: Conclusioni

- Eliminando la necessità di allocare processi in aree contigue si rende più efficiente la gestione della memoria fisica;
- no frammentazione interna;
- Supporto efficiente di protezione e condivisione;
- Supporto del collegamento dinamico.

SVANTAGGI:

- Frammentazione esterna (buchi troppo piccoli per ospitare un intero segmento), necessità di effettuare comunque la compattazione (resa anche più complessa);
- il duplice accesso alla memoria deve essere supportato da hardware apposito;
- la gestione della memoria da parte del sistema operativo è più complessa rispetto a quella per il partizionamento statico e dinamico;
- non rende ancora possibile l'esecuzione di processi più grandi delle dimensioni fisiche della memoria.

Allocazione non contigua

La memoria viene allocata in modo tale che un unico oggetto logico viene posto in aree separate e non adiacenti.

Lo spazio degli indirizzi fisici non’è contiguo.

Durante l’esecuzione di un processo viene eseguita la traduzione degli indirizzi per ristabilire la corrispondenza tra lo spazio virtuale di indirizzamento (contiguo) e gli indirizzi fisici.

PAGINAZIONE

MEMORIA VIRTUALE

Paginazione

La memoria fisica viene suddivisa in pagine di dimensione fissa chiamate *pagine fisiche* o *frame*.
La memoria logica viene suddivisa in *pagine logiche* o *virtuali* della stassa dimensione delle pagine fisiche.

Caricamento in memoria di un processo:
- si prelevano le pagine logiche dalla memoria ausiliaria
- si caricano le pagine logiche nelle pagine fisiche (blocchi) della memoria centrale

Traduzione degli indirizzi: ogni indirizzo generato dalla CPU è diviso in due parti:
- bit più significativi → P - numero di pagina (virtuale);
- bit meno significativi → D - spiazzamento (reale);

Il sistema operativo tiene traccia della corrispondenza tra pagine virtuali e pagine fisiche mediante la *tabella delle pagine* (TDP) o *page table*.
- la TDP viene costruita al momento del caricamento del processo;
- ogni riga della TDP contiene l'indirizzo di partenza della pagina fisica in cui è allocata la corrispondente pagina virtuale
- la tabella è costituita da un numero di righe pari al numero di pagine occupate.

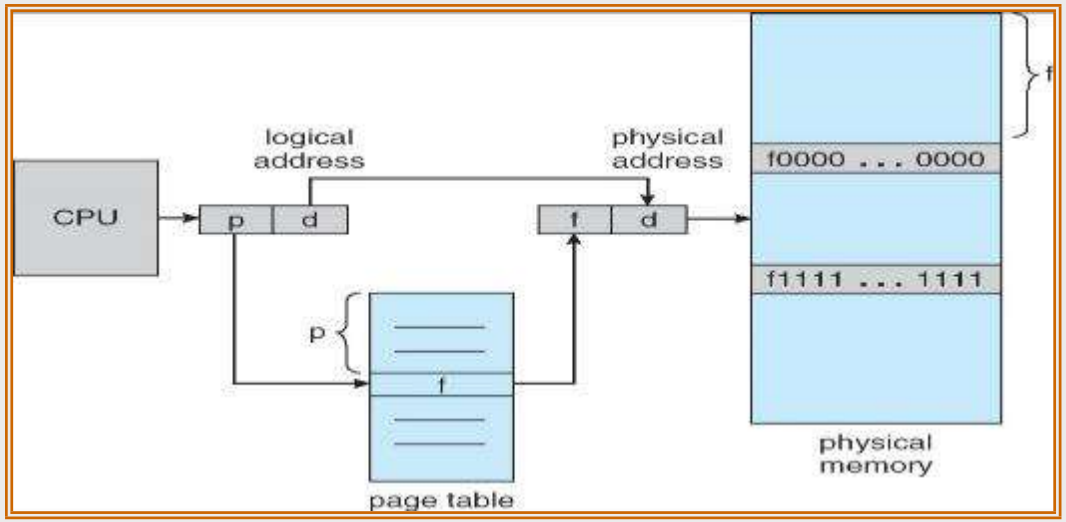
Paginazione esempio

ES:

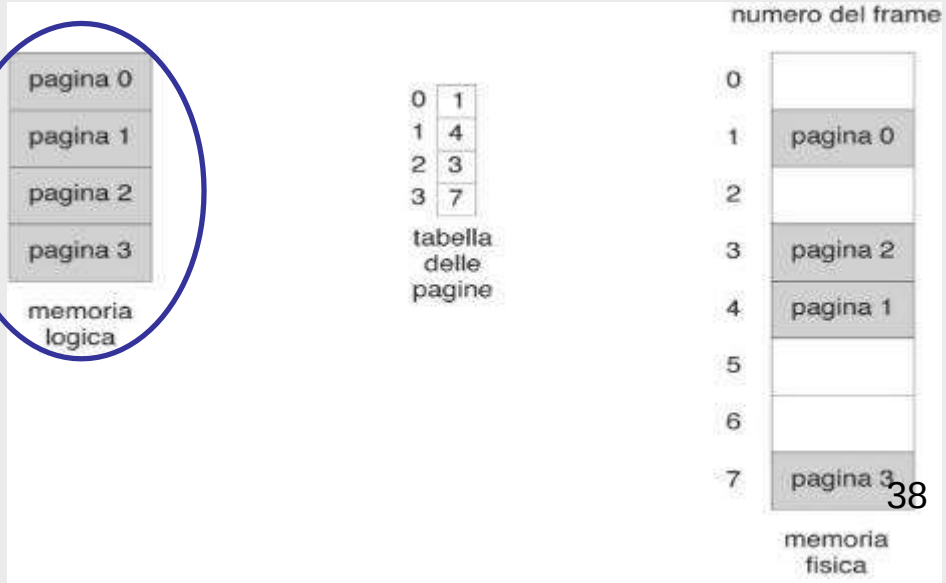
Memoria:	1 Mb
Dimensione Pagina:	256 byte (se la memoria è indirizzabile al byte occorrono 8bit per il displacement)
Numero di pagine:	4096 (12 bit per indirizzare la pagina)
Lunghezza dell'indirizzo:	20 bit (12+8)

Lo spazio di indirizzamento di un ipotetico processo utente che occupi 1008 byte viene diviso in quattro pagine virtuali numerate da 0 a 3.

Paginazione
Meccanismo di traduzione



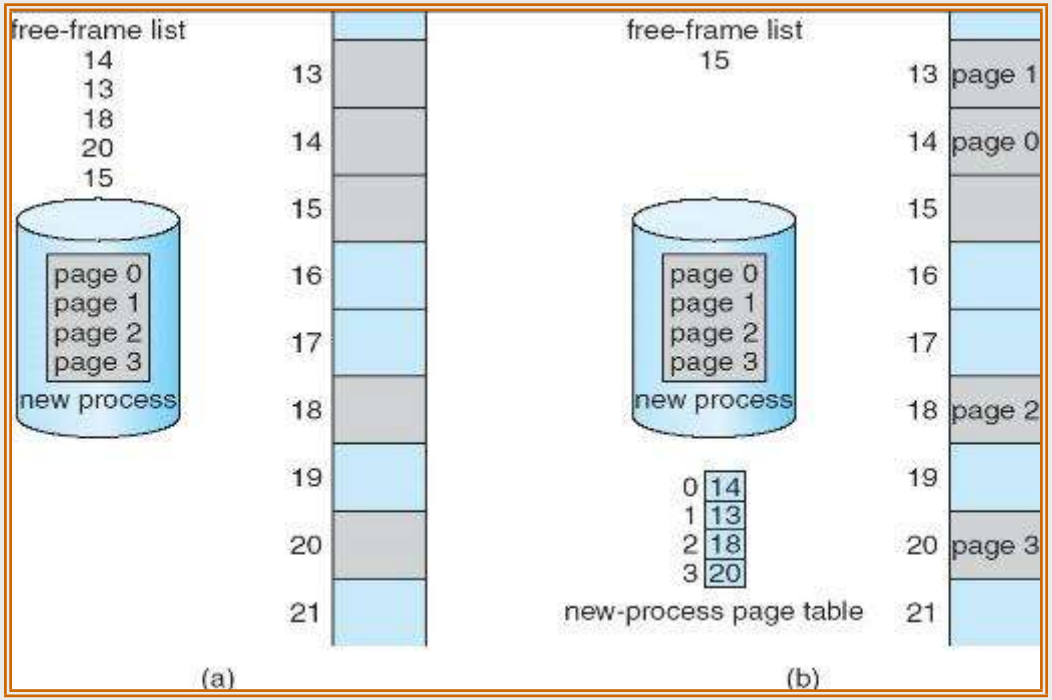
Memoria vista dall'utente



Paginazione

Il sistema operativo tiene traccia dello stato di tutte le pagine fisiche mediante la *tabella di memoria* (TDM):

- in ogni riga della TDM è indicato lo stato della relativa pagina: “Libera” o “Allocata”;
- il numero di righe della tabella è uguale al numero di pagine fisiche.
- esiste la tabella delle pagine (frame) libere



Quando si deve eseguire un processo si esamina la sua grandezza in pagine e se ci sono sufficienti frame disponibili, ad ogni pagina si assegna un frame. Si carica la prima pagina in un frame e si inserisce il numero del frame ad essa assegnato nella tabella delle pagine, la successiva in un altro frame e così via.

OSSERVAZIONI:

1. Un processo non può accedere alle pagine di un altro processo poiché queste non sono contenute nella sua tabella delle pagine
2. Ad ogni processo è associata una tabella delle pagine => ulteriori dati da gestire nel context switching

Paginazione

Frammentazione interna: l'ultima pagina assegnata ad un programma non sarà completamente piena

Pagine molto piccole:
- minore frammentazione
- Carico eccessivo nell'accesso al meccanismo di traduzione degli indirizzi
- Numerosi accessi alla memoria di massa,

ATTUALMENTE LA DIMENSIONE DELLE PAGINE E' COMPRESA TRA l 4KB E 8 KB

Paginazione: Memorizzazione di aree libere

La consultazione della *Tabella di Memoria* statica per trovare *n* pagine fisiche libere richiede una ricerca su un numero di righe che in media è $x = n / q$.

q è la probabilità che una pagina sia libera e si lega alla percentuale di memoria libera *u*: $q = u / 100$ $0 \leq q \leq 1$

x aumenta all'aumentare dell'occupazione di memoria

Per questo motivo si preferisce sostituire la TDM con una lista a puntatori contenente i numeri delle pagine fisiche:

- *n* pagine da allocare si individuano nei primi *n* nodi della lista
- in fase di disallocazione vengono inseriti in testa alla lista gli *n* nodi relativi alle pagine liberate.

Vantaggi:

- il tempo di allocazione e disallocazione non dipende dalla percentuale di occupazione della memoria.

Svantaggi:

- per il singolo elemento vi è un lieve spreco di memoria e di tempo rispetto alla gestione di una tabella statica.

Architettura di paginazione

Se per ogni processo c'è una Page Table, il PCB deve contenere un puntatore a tale tabella.

⇒Per un meccanismo veloce di traduzione l'intera tabella deve essere posizionata in opportuni registri del processore

⇒Limitazione alla dimensione di tale tabella… meccanismo non idoneo per i moderni SO

I meccanismi hardware per la paginazione vengono utilizzati per due scopi:

1. Risparmiare la memoria necessaria per le TDP

2. Velocizzare la traduzione da indirizzi virtuali a fisici

E' ragionevole costruire tabelle con un numero di righe uguale alle pagine effettivamente utilizzate.

a) La tabella delle pagine viene mantenuta in memoria
b) registro **base** della tabella delle pagine (PTBR): contiene l'indirizzo di partenza della tabella. Il cambio di tabelle implica il cambio di questo registro. L*a velocità si dimezza, doppio accesso alla memoria (TDP e pagina)…*

Architettura di paginazione

Velocizzazione del tempo di traduzione

La traduzione di ciascun indirizzo virtuale richiede due accessi in memoria:

- uno alla TDP per prelevare il numero di pagina fisica;
- l'altro per accedere fisicamente alla locazione richiesta.

SOLUZIONE:

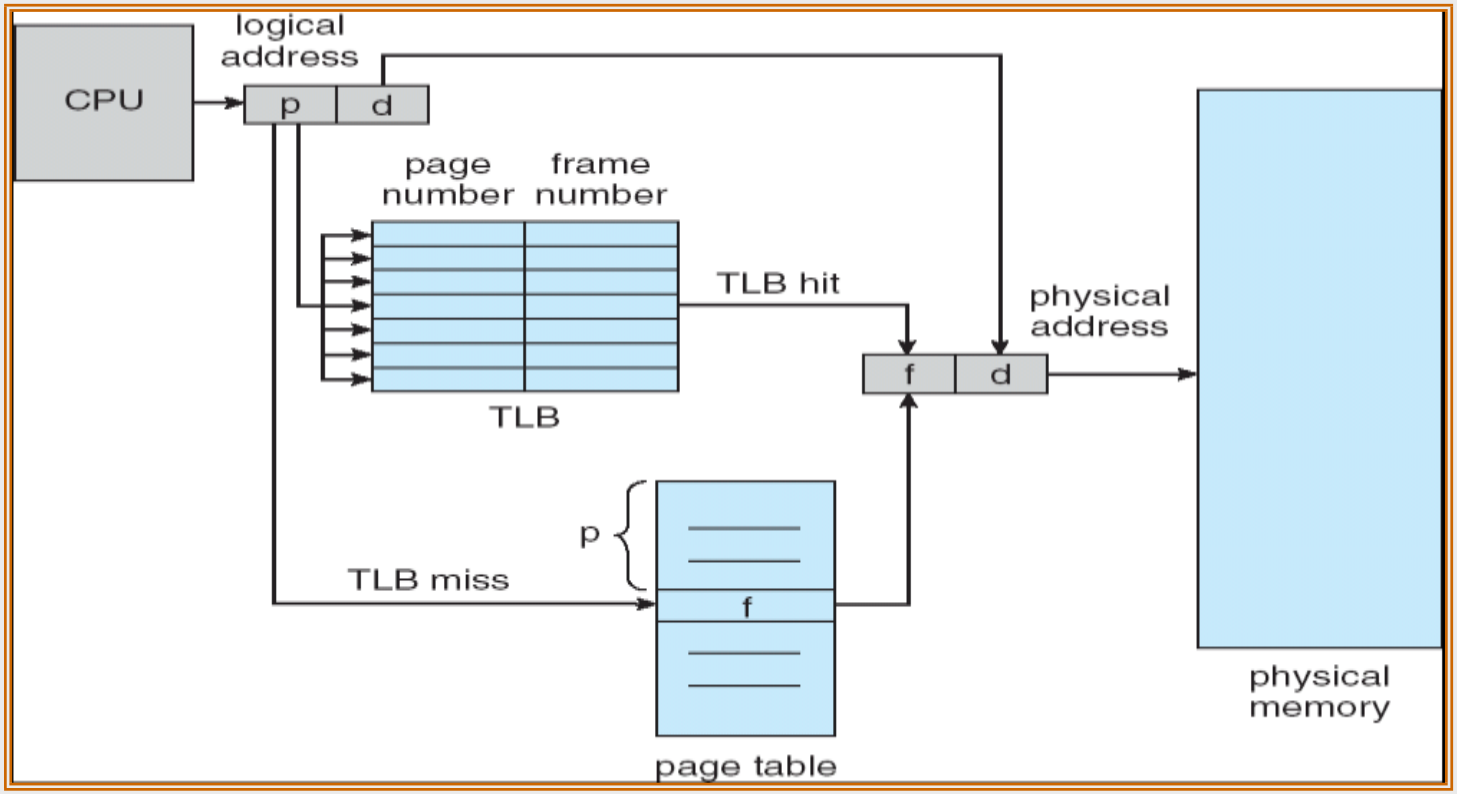
Utilizzo di una cache di traduzione (eventualmente di tipo associativo) "Translation Look-aside buffer" (**TLB**) in cui vengono mantenuti gli indirizzi delle pagine usate più frequentemente (porzione della tabella delle pagine).

Ogni elemento della $ è costituito da due parti: una chiave ed un valore.

- Si confronta la chiave generata (ad esempio tramite l'operazione mod se la $ è set associativa…) a partire dall'indirizzo logico della pagina con le chiavi presenti nella TLB:
  - -se presente il corrispondente valore (indirizzo della pagina fisica) è usato per accedere alla memoria
  - -se assente si è verificato un miss e si deve accedere alla page table in memoria

Aggiornando la $ otterremo un più rapido accesso al referenziamento successivo.

Architettura di paginazione



- *Se a seguito di un miss si deve inserire nella TLB il nuovo riferimento e non c'è spazio si può usare un algoritmo di sostituzione ad es. LRU…*
- *Alcune pagine si possono vincolare (wired-down) nella TLB per esempio quelle del Kernel.*
- *Una TLB può anche contenere un ASID (Address Space Identifier): viene usato per controllare se il processo che tenta di accedere è autorizzato altrimenti si genera un miss TLB. ASID associato ai processi…*

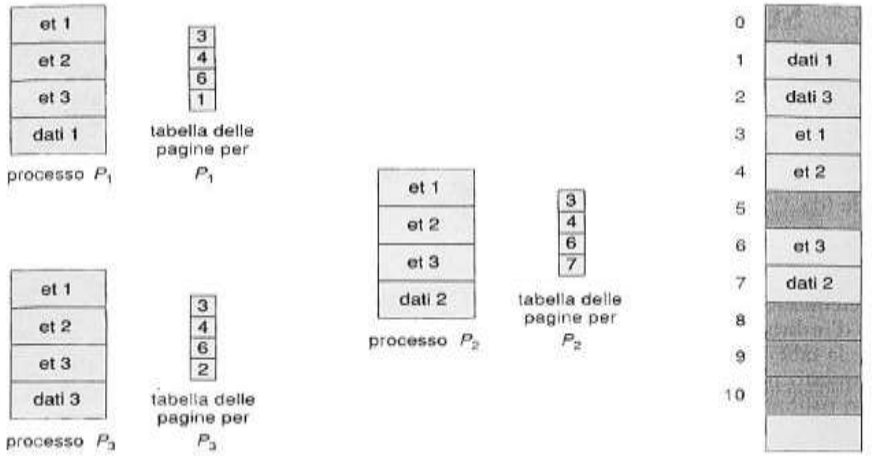
Paginazione: protezione

- i valori dei registri sono modificati esclusivamente da istruzioni privilegiate del sistema operativo;
- **l'aggiunta di alcuni bit alle righe delle TDP consente di definire e controllare il tipo di accesso alla pagina (r,rw,exe);**
- non ha senso la protezione interna allo spazio di indirizzamento del processo, poiché non prevede tipizzazione di dati.
- bit di validità: valido=pagina corrispondente è realmente utilizzata dal processo. Un processo può utilizzare solo un sottoinsieme delle pagine a sua disposizione
- Alcuni sistemi utilizzando il Page table length register (PTLR) come indicazione sul numero di pagine appartenenti ad un processo.

Paginazione: condivisione

- **una copia unica di una pagina fisica può essere mappata in molti spazi di indirizzamento;**
- **i diversi processi possono avere un tipo di accesso diverso alla pagina;**
- la condivisione viene riconosciuta ed effettuata da programmi di sistema, poiché la paginazione è trasparente all'utente;
- il codice condiviso deve essere eseguito in mutua esclusione.

- *Le pagine condivise vengono utilizzate da alcuni sistemi per inviare messaggi tra processi*.



ES.: pagine et1,et2, et3 editor di testo..

Paginazione Gerarchica

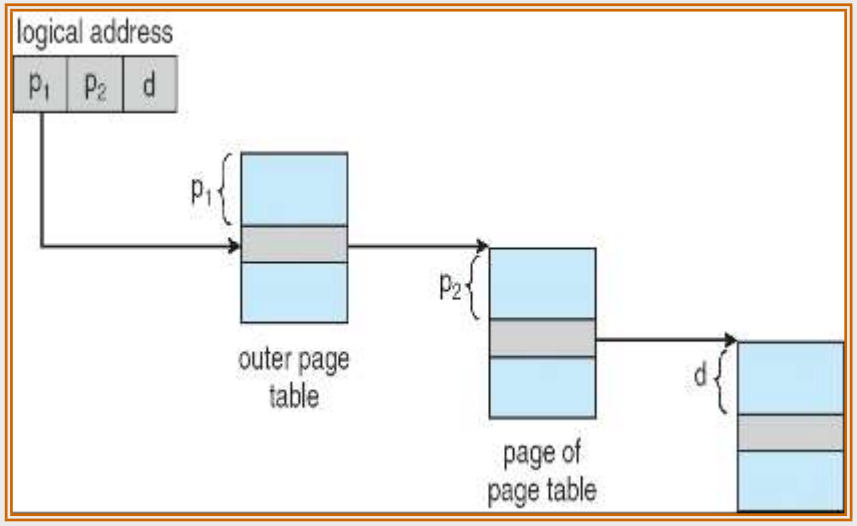
PROBLEMA: Nelle moderne architetture a 32 e 64 bit, la dimensione della tabella delle pagine può arrivare ad occupare alcuni MB per ogni processo

SOLUZIONE: paginazione a due livelli:

- Viene paginata anche la tabella delle pagine
- Viene generato l'indirizzo logico:

page number		*page offset*
p_1	p_2	d

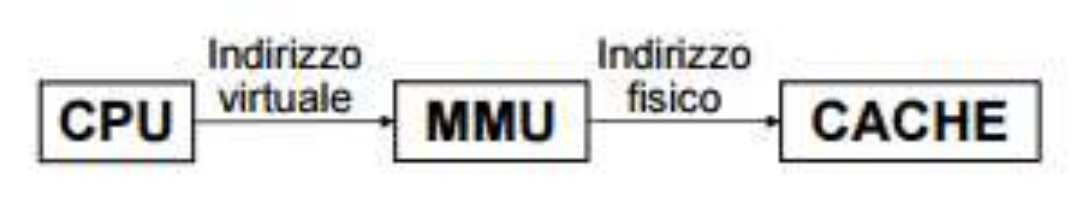
- P1:indirizzo logico della tabella di primo livello
- P2:scostamento all'interno della pagina puntata da P1



Cpu - $ - RAM .....

Cosa accade se tra CPU e RAM re-introduciamo il concetto di cache….????

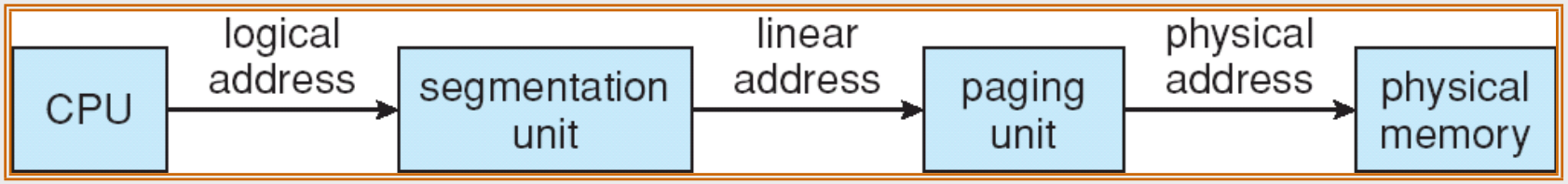
1. l'accesso alla cache può avvenire usando l'indirizzo fisico di memoria:
   - Prima dell'accesso in $ si traduce l'indirizzo virtuale in indirizzo fisico



2. Alcune architetture consentono l'accesso alla $ con indirizzi virtuali (raro)

INTEL PENTIUM

- **Supporta sia la segmentazione che la *segmentazione con paginazione***
- La CPU genera indirizzi logici
  - Dati all'unità di segmentazione
    - Produce indirizzi lineari
  - Gli indirizzi lineari vengono dati all'unità di paginazione
    - Genera indirizzi fisici per la memoria centrale



In sostanza I segmenti sono costituiti da pagine.

Memoria Virtuale

Schema di gestione della memoria in cui soltanto una parte dello spazio di indirizzamento virtuale di un processo “residente” viene effettivamete caricata in memoria.

Tecnica che consente di eseguire processi senza che essi siano completamente contenuti in memoria

- la somma di tutti gli spazi di indirizzamento virtuali dei processi attivi può superare la capacità della memoria fisica;
- La dimensione ammissibile per lo spazio di indirizzamento virtuale di un singolo processo può superare la capacità della memoria fisica disponibile in un sistema .

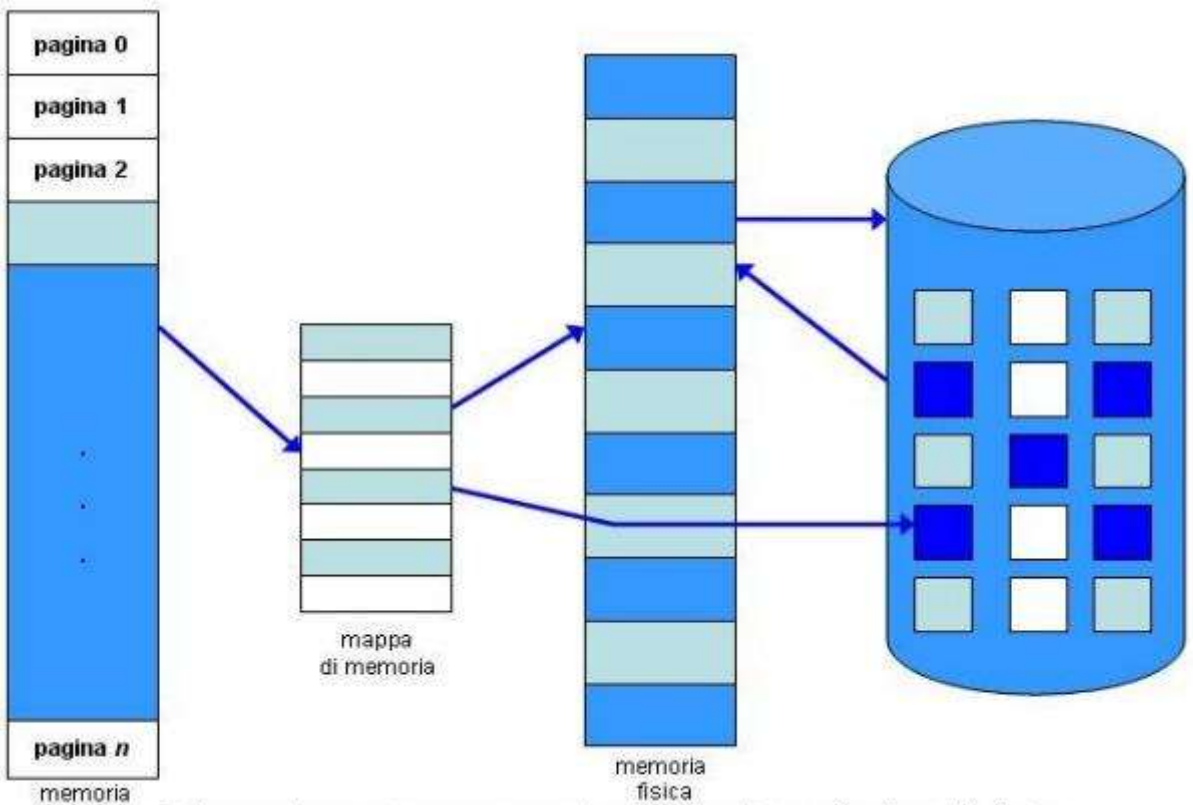
Molte parti di un processo vengono usate raramente.

Es: Il codice per le correzione degli errori;
Opzioni usate quasi mai;
Risorse che sono state per prudenza sovradimensionate.

Apparente uso della memoria superiore al 100%

Memoria Virtuale

La memoria virtuale si realizza (solitamente) nella forma di paginazione su richiesta



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

In memoria secondaria si mantiene l'immagine dell'intero spazio virtuale di indirizzzamento del processo;

Si trasferiscono in memoria fisica le diverse parti solo al momento in cui sono necessarie (vengono referenziate);

Il sistema operativo sceglie tempi e modalità di trasferimento in memoria fisica tenendo conto di:

- *richieste dei processi attivi;*
- *priorità dei processi;*
- *disponibilità globali del sistema.*

Memoria Virtuale

Punto di vista del programmatore:

- i dettagli della gestione sono trasparenti;
- non è necessario l'adeguamento di un programma ad una memoria limitata;
- un programma può essere eseguito su sistemi con disponibilità di memoria fisica diversa, senza alcun adattamento.

Punto di vista del Sistema Operativo:

- un processo può essere caricato in uno spazio di dimensione arbitrarie;
- non è necessario modificare l'ordine previsto di esecuzione di un processo;
- il quantitativo di memoria assegnato ad un processo può variare durante la sua esecuzione;
- il sistema operativo può dinamicamente scegliere di velocizzare l'esecuzione di un processo allocando un quantitativo maggiore di memoria o, in alternativa, aumentare il grado di multiprogrammazione.
- si riduce la frammentazione esterna;

Memoria Virtuale

Osservazioni:

1. Una istruzione può essere completata soltanto se quanto gli è necessario (codice, stack, dati) è presente in memoria;
2. un processo che richiede un “oggetto” fuori dalla memoria viene sospeso (blocked) per un tempo “relativamente” lungo.

Importanza dell’uso dello schema di memoria virtuale:

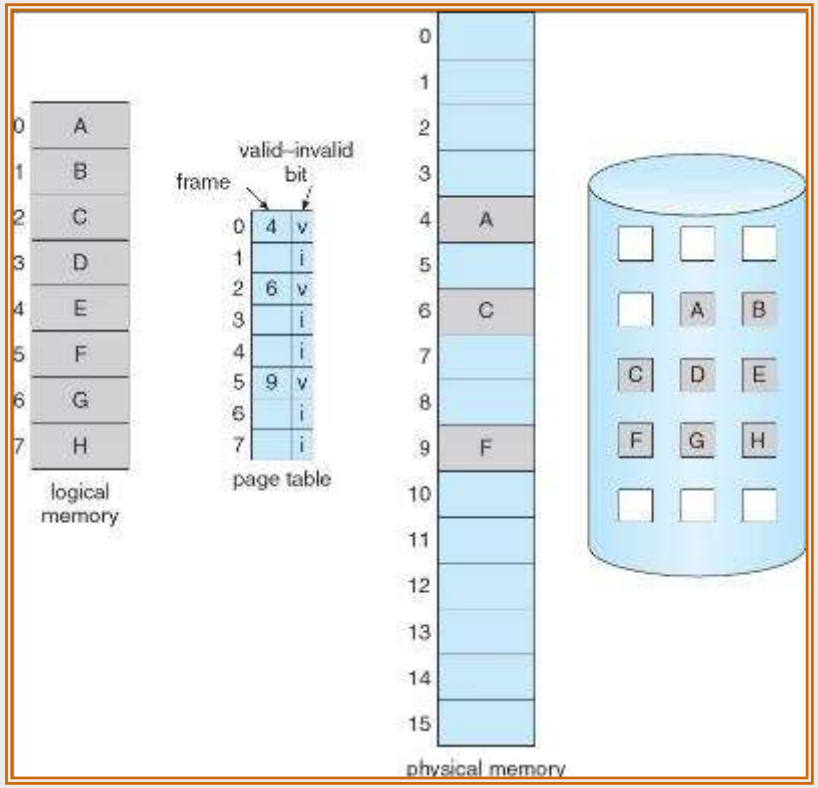
- durante una specifica esecuzione, alcune parti del programma non vengono indirizzate;
- le diverse condizioni interne o esterne causano diversi tracciati ad ogni esecuzione di un processo (if-else-if, switch-case);
- alcune parti di un programma vengono eseguite molto raramente (es. routine di gestione errori);

Bit di validità

- Al momento del caricamento del processo in memoria, il paginatore opera una predizione sulle pagine che saranno utilizzate
- Vengono trasferite in memoria solo tali pagine

➢ **Necessità di un meccanismo che indichi se una pagina è in memoria o su disco**

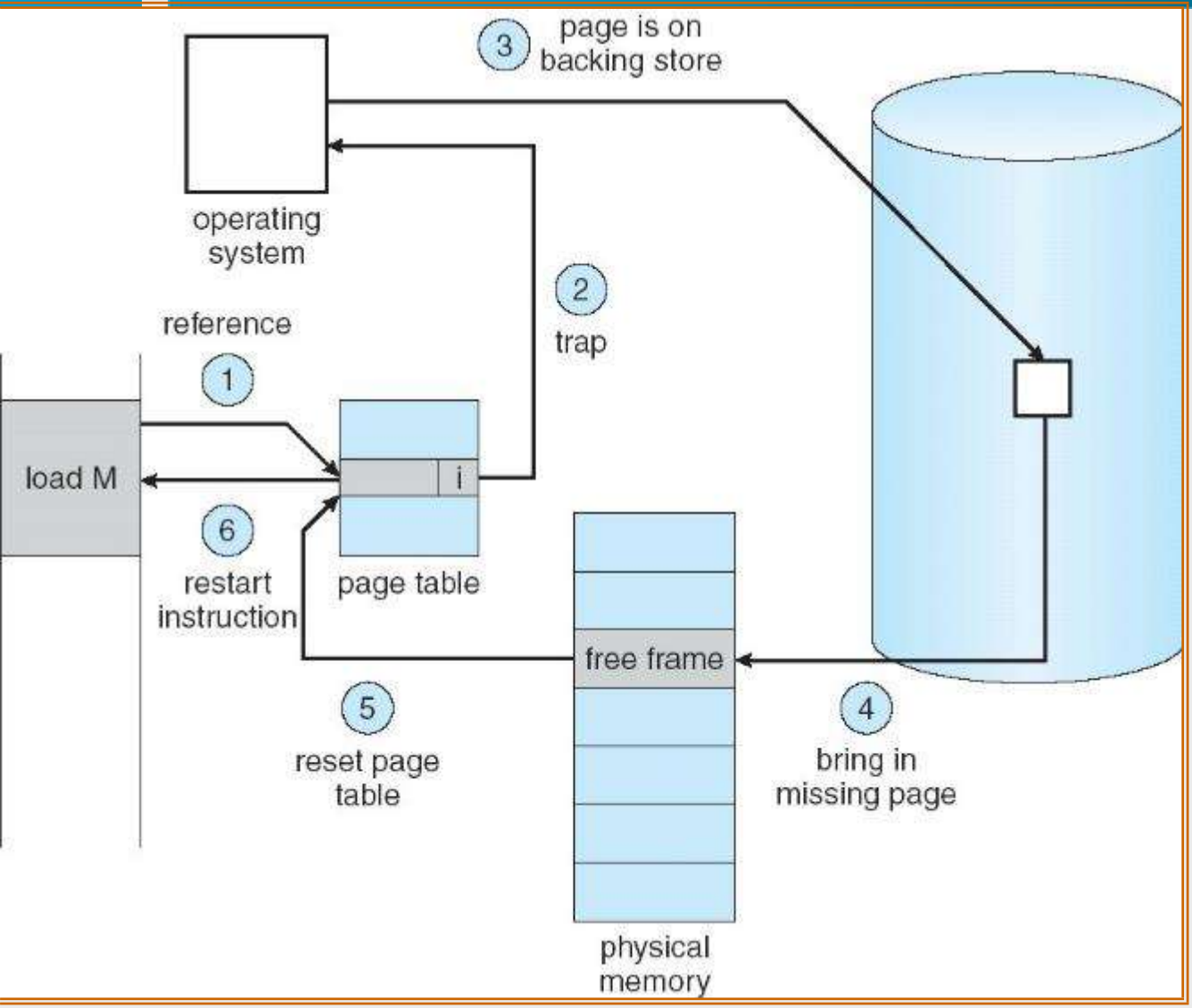
- **BIT di validità** nella tabella delle pagine: (**v** ⇒ in-memory, **i** ⇒ not-in-memory)



NB: i può anche indicare che la pagina referenziata non'è valida per il processo

PAGE FAULT TRAP: eccezione di pagina mancante – riferimento a una pagina non presente in memoria centrale

Gestione del PAGE FAULT



- **OS** (attraverso l'uso di una tabella esterna indicata nel PCB) determina se:
  - Invalid reference ⇒ abort
  - Pagina non in memoria
- Individuazione di un blocco libero
- Trasferimento della pagina nel blocco
- Aggiornamento della tabella delle pagine (include, validation bit = **v**)
- Riavviare l'istruzione che aveva cagionato il page fault…questa volta la pagina referenziata sarà in memoria…

Paginazione su richiesta

PAGINAZIONE SU RICHIESTA PURA

Una pagina non viene trasferita in memoria fino a quando non’è richiesta

- E’ possibile avviare l’esecuzione di un processo senza pagine in memoria
- Al primo riferimento si genera un page fault
- Una volta caricata la pagina l’esecuzione del processo prosegue fino al successivo page fault

Meccanismi di ausilio al SO:

1. Tabella delle pagine con bit di validità
2. Memoria ausiliaria

   La parte di memoria ausiliaria dedicata all’avvicendamento dei processi prende il nome di area di avvicendamento o di swap (scambio)

NB: sistema di paginazione su richiesta trasparente all’utente e si colloca tra CPU e memoria

Performance della Paginazione su richiesta

Fino a quando non si verificano page fault, il tempo di accesso effettivo è uguale al tempo di accesso alla memoria (nella ipotesi di assenza di $)

Sia p la probabilità che si verifichi un Page Fault ($0 \leq p \leq 1$)

- $p = 0$ no page faults
- $p = 1$, every reference is a fault

- Tempo di accesso effettivo EAT=
  = (1 – p) * memory access + p * (tempo gestione pagina mancante)

Gestione pagina mancante:

- Generazione trap
- Salvataggio contesto processo (run->blocked)
- Determinazione della natura della trap
- Controllo correttezza riferimento
- Lettura da disco e trasferimento
  - Attesa in blocked per lo specifico dispositivo (la CPU può essere assegnata ad altri processi)
  - Seektime+rotational latency+transfert time

continua…

Performance della Paginazione su richiesta

…continua

- Generazione interrupt (I/O completato)
- Salvataggio contesto processo nello stato di Run
- Gestione interruzione
- Aggiornamento tabella delle pagine per il processo che aveva generato il page fault, blocked->ready
- Attesa in ready
- Ripristino del contesto

Esempio:

•Memory access time = 100 nsec

•Average page-fault service time = 25 msec

•EAT = (1 – p) x 100 + p (25 milliseconds)

= (1 – p) x 100 + p x 25,000,000

= 100 + p x 24,999,900

•Se un accesso ogni 1000 genera un page fault: EAT = 25,000 nsec

RALLENTAMENTO DEL 250%!!!

Per avere un rallentamento del 10% il sistema deve garantire meno di una assenza ogni 2,500,000 di riferimenti

Tempo di accesso effettivo proporzionale alla frequenza di assenza delle pagine

Paginazione su richiesta

Come selezionare quali pagine portare in memoria:

CONCETTO DI LOCALITA': i programmi hanno una forte tendenza a favorire un sottoinsieme del loro spazio di indirizzamento durante l'esecuzione.

Una parte notevole degli accessi, in un periodo di tempo, vengono effettuati su un set ridotto delle pagine virtuali di un processo. Un processo si "muove" lentamente da una località ad un'altra nel corso della sua esecuzione.

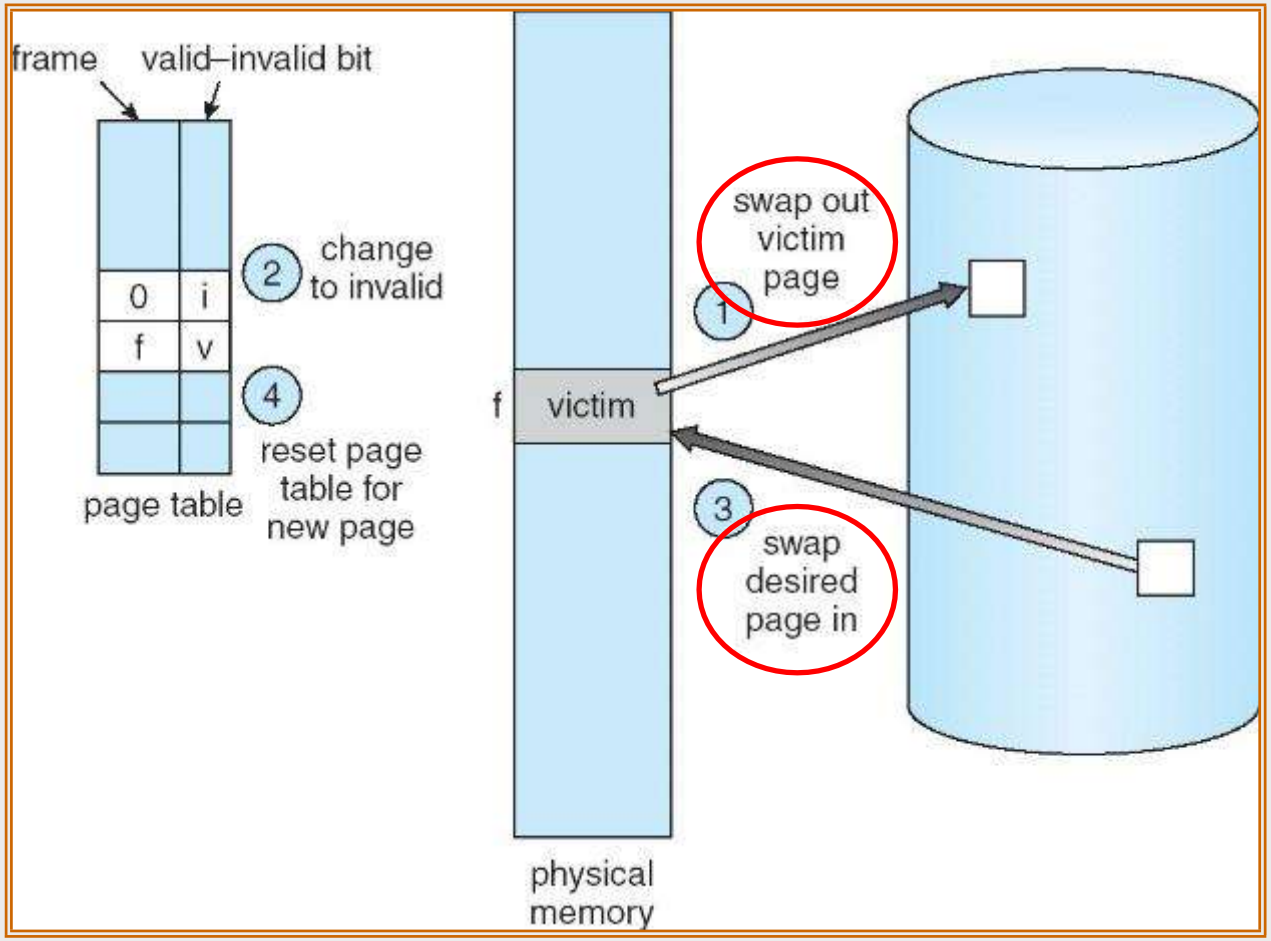
Il set di pagine più referenziate varia lentamente nel tempo. Questa proprietà è utilizzata per realizzare le strategie di allocazione e sostituzione delle pagine nello schema di gestione della memoria virtuale.

Come gestire il caso di:

MANCANZA DI SPAZIO LIBERO IN MEMORIA:

1. Terminare un processo utente…*scelta peggiore*
2. Scaricare dalla memoria l'intero processo e ridurre il grado di multiprogrammazione
3. **Sostituzione delle pagine**:
    1. Necessità di un algoritmo di selezione della pagina "vittima"
    2. Due trasferimenti da/a memoria (swap-in swap-out)
    3. Sovraccarico al 2. riducibile tramite *dirty bit* (blocco in memoria da portare su disco non modificato dal suo caricamento)

Page Replacement



Algoritmo di replacement:

- Minor page-fault possibile
- Valutare l'algoritmo su specigiche sequenze di accesso alla memoria
- Negli esempi la sequenza è:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Algoritmi di replacement: FIFO

Vengono sostituite le pagine che da più tempo risiedono in memoria.
Il gestore della memoria tiene traccia dell'ordine di caricamento delle pagine in memoria, ad esempio con una coda FIFO dei numeri di pagina.

Pregi:

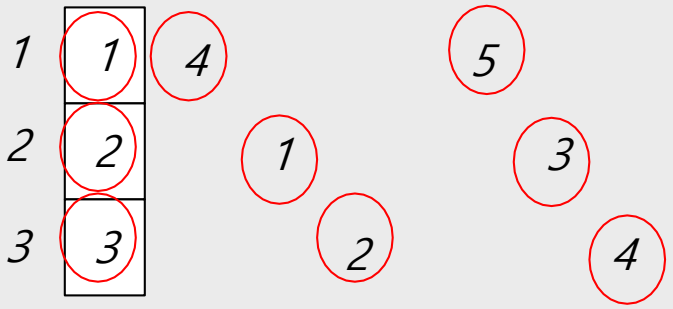
- semplice realizzazione;
- aggiornare la coda solo ad ogni page fault;
- non richiede supporti hardware specifici.

Difetti:

- swap out di pagine più frequentemente indirizzate, poiché per loro natura permangono più tempo in memoria;
- non tiene traccia dell'ordine di riferimento delle pagine.

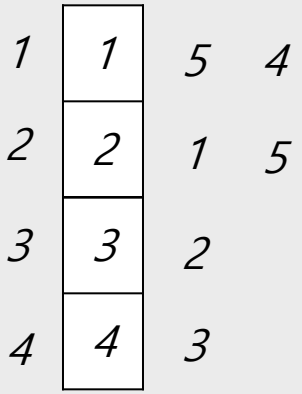
Algoritmi di replacement: FIFO

- Stringa di riferimento: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages in memoria)



9 page faults

- 4 frames



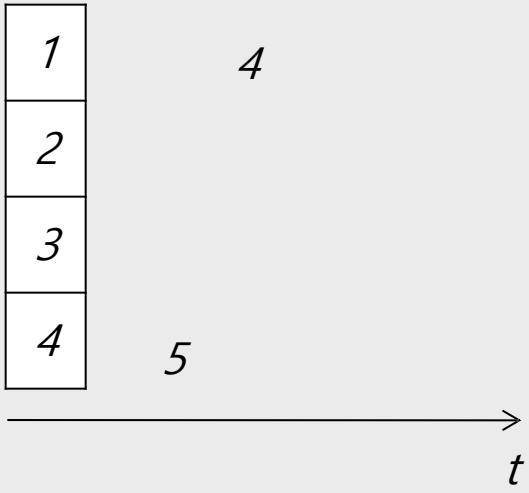
10 page faults

- Anomalia di Belady: con alcuni algoritmi di sostituzione delle pagine, più frames ⇒ più page faults

Algoritmi di replacement: Algoritmo Ottimale o di Belady

- **Sostituire le pagine che non saranno utilizzate per il periodo di tempo più lungo (pagine che saranno utilizzate solo dopo molto tempo)**
- Algoritmo ottimale: minimizza il numero di page faults
- 4 frames, es:

1, 2, 3, 4, 1, 2, **5**, 1, 2, 3, **4**, 5

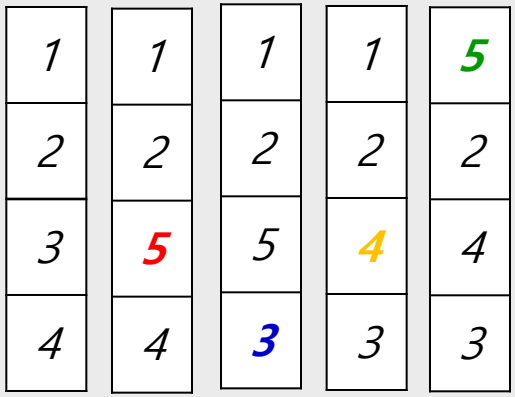


6 page faults

- Non realizzabile: richiede la conoscenza futura della successione dei riferimenti
- Consente di confrontare le prestazioni degli altri algoritmi.
- NB: non soggetto ad anomalia di Belady

Algoritmi di replacement: Least Recently Used (LRU)

- **Si sostituiscono le pagine meno recentemente utilizzate**
  - Ipotesi: la pagina meno utilizzata recentemente è quella che ha la minore probabilità di essere referenziata in futuro. Si approssima il futuro prossimo al passato recente
- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

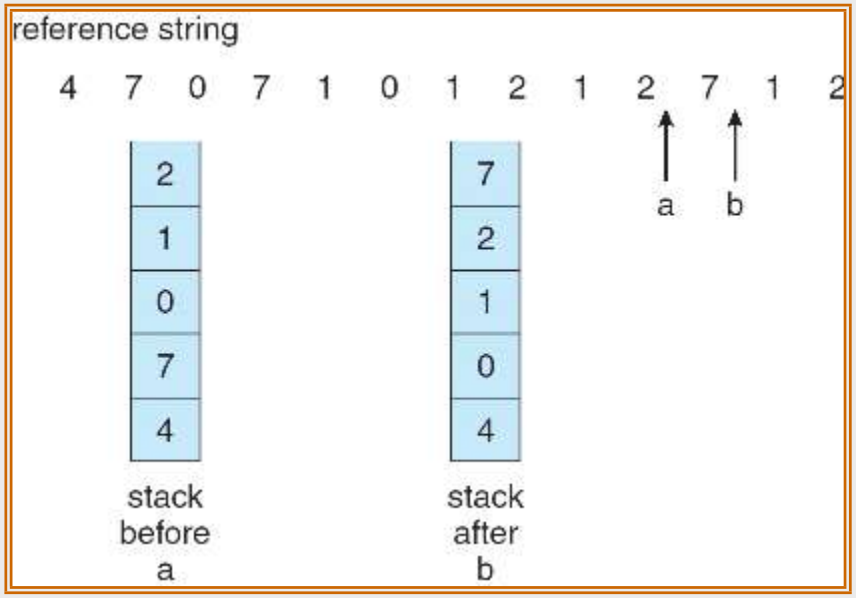


8 page faults

- Non soggetto ad anomalia di Belady
- **Strategie implementative (forte assistenza del SO)**
  - Associare ad ogni pagina l'istante in cui è stata usata per l'ultima volta
    - Campo aggiuntivo nella page table
    - Necessità di dover ricercare la pagina con il più vecchio marcatore temporale
  - Memorizzare l'uso delle pagine in una struttura di tipo stack:
    - Ogni volta che una pagina viene referenziata viene spostata in cima allo stack
    - La pagina da swap out viene presa dalla base dello stack
    - Necessità di dover costantemente aggiornare lo stack vs. semplice selezione della pagina vittima

Algoritmi di replacement: Least Recently Used (LRU)

Utilizzo dello stack…



Pregi:

- in media si comporta meglio dell'algoritmo FIFO.

Difetti:

- l'aggiornamento dello stack va effettuato ad ogni riferimento alla pagina (praticamente ad ogni accesso in memoria)
- l'organizzazione dell'LRU è tale da richiedere un supporto hardware sofisticato e dedicato alle operazioni relative all'aggiornamento dello stack.

Algoritmi di replacement: Approssimazione a LRU

- Utilizzo di un contatore per numerare il numero di accessi alle pagine

- **LFU Algorithm – Least Frequently Used**: sostituzione delle pagine con il valore di contatore più basso
  - Filosofia: le pagine utilizzate frequentemente hanno valori contatori elevati
  - Alcune pagine potrebbero essere referenziate frequenteme all'avvio di un processo e poi non essere più necessarie. Soluzione: azzerare il contatore periodicamente…

- **MFU Algorithm – Most Frequently Used**: sostituzione delle pagine con il più alto valore di contatore
  - Filosofia: la pagina con valore basso di contatore è stata appena portata in memoria e sarà utilizzata in futuro

Algoritmi poco utilizzati: male approssimano l'algoritmo ottimale.

Algoritmi di replacement: Approssimazione a LRU

- **Reference bit (bit di riferimento)**
  - Ad ogni pagina (nella tabella delle pagine) è associato un bit, inizialmente = 0
  - Quando una pagina viene referenziata (sia in lettura che in scrittura) si setta il bit a 1
  - Sostituire le pagine che hanno il bit a 0
    - NB: non si conosce l'ordine di utilizzo
    - I bit possono essere azzerati ad intervalli regolari

- **Seconda chance (basato su logica FIFO)**
  - Utilizzo del reference bit
  - Clock replacement (per la generazione della coda FIFO)
  - Dopo aver selezionato una pagina dalla coda FIFO si controlla il bit di riferimento:
    - 0 => sostituzione della pagina
    - 1 => si dà una seconda chance alla pagina (non la si sostituisce)
      - Si pone il bit a 0
      - si passa ad esaminare un'altra pagina nella coda FIFO

- **Seconda chance migliorato**
  - Utilizzo di reference e dirty bit:
    - (0,0) – pagina non referenziata nè modificata -> migliore pagina da sostituire
    - (0,1) – non referenziata ma modificata -> prima di sostituire la pagina la si deve salvare
    - (1,0) – usato recentemente ma non modificato -> potrebbe essere nuovamente referenziata
    - (1,1) – usato recentemente e referenziato ->…
    - *MACH OS*