Gestione della Memoria

Multiprogrammazione e gestione della memoria

- L'obiettivo primario della **multiprogrammazione** e` l'uso efficiente delle risorse computazionali:
  - Efficienza nell'uso della CPU
  - Velocita` di risposta dei processi
  - …
- **Necessita` di mantenere piu` processi in memoria centrale**: il S.O. deve quindi gestire la memoria in modo da consentire la presenza contemporanea di piu` processi;
- **Caratteristiche importanti**:
  - Velocita`
  - Grado di multiprogrammazione
  - Utilizzo della memoria
  - Protezione

Gestione della Memoria Centrale

A livello hardware:

ogni sistema di elaborazione è equipaggiato con un'unico spazio di memoria accessibile direttamente da CPU e dispositivi.

Compiti del Sistema Operativo:

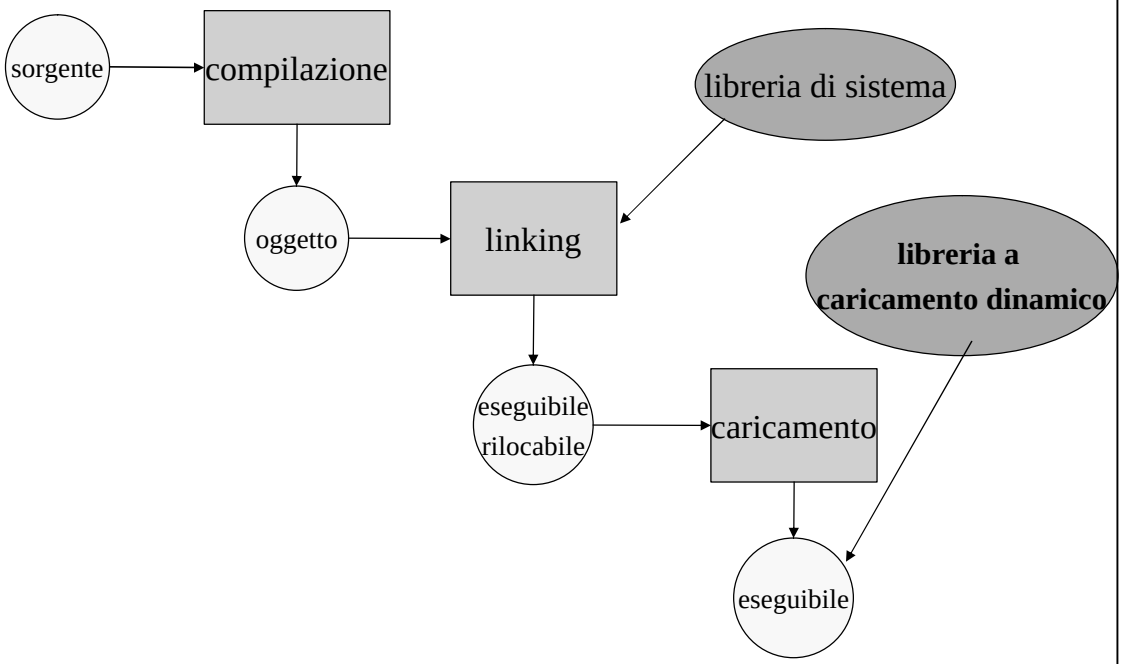
- **allocare memoria ai processi**
- **deallocare memoria**
- **separare gli spazi di indirizzi associati ai processi (*protezione*)**
- **realizzare i collegamenti (*binding*) tra memoria logica e memoria fisica**
- **memoria virtuale: gestire spazi logici di indirizzi di dimensioni complessivamente superiori allo spazio fisico**

Accesso alla memoria

- **Memoria centrale:**
  - vettore di celle (parole), ognuna univocamente individuata da un indirizzo.
  - Operazioni fondamentali sulla memoria: **load**, **store** di dati e istruzioni
  - **Indirizzi:**
    - indirizzi simbolici (riferimenti a celle di memoria nei programmi in forma sorgente)
    - indirizzi logici: riferimenti a celle nello spazio logico di indirizzamento del processo
    - indirizzi fisici: indirizzi assoluti delle celle in memoria a livello HW

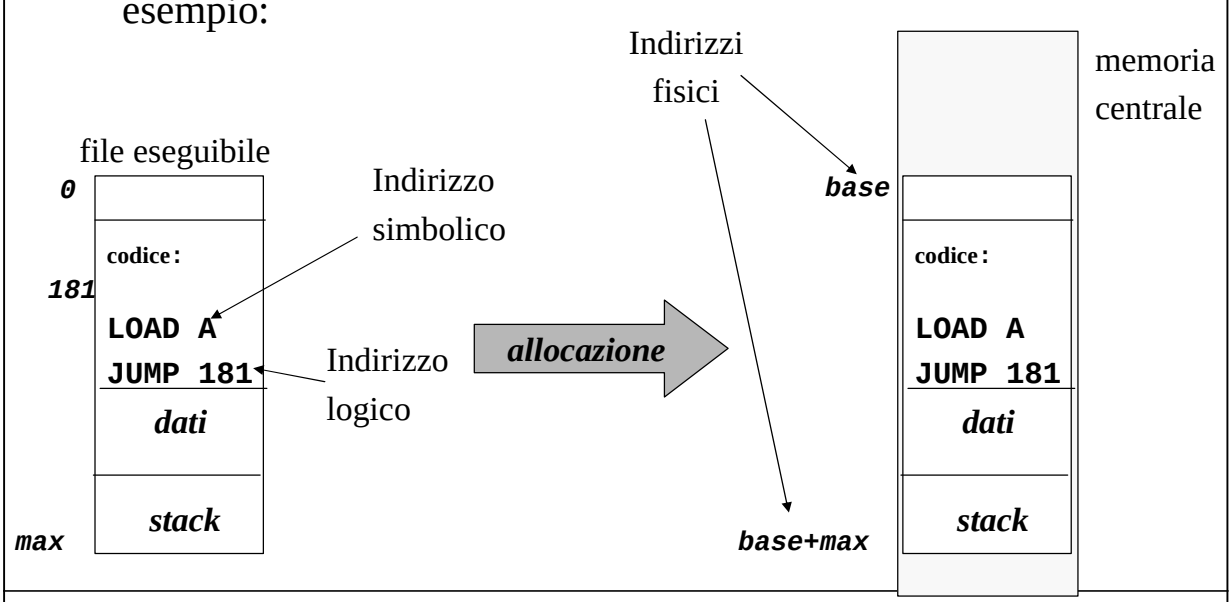
Qual'e` la relazione tra i diversi tipi di indirizzo?

Fasi di Sviluppo di un programma



Indirizzi simbolici, logici e fisici

- Ogni processo dispone di un proprio spazio di indirizzamento "logico" [0,max], che viene allocato nella memoria fisica. Ad esempio:



Binding degli indirizzi

- Ad ogni indirizzo **logico/simbolico** viene fatto corrispondere un indirizzo **fisico**: l'associazione tra indirizzi simbolici ed indirizzi assoluti viene detta ***binding***.
- **Il binding puo` essere effettuato:**
  - **staticamente:**
    - **a tempo di compilazione**: in questo caso il compilatore genera degli indirizzi assoluti (es: file .com del DOS)
    - **a tempo di caricamento**: il compilatore genera degli indirizzi relativi che vengono convertiti in indirizzi assoluti dal loader (codice *rilocabile*)
  - **dinamicamente:**
    - **a tempo di esecuzione**: durante l'esecuzione un processo puo` essere spostato da un'area all'altra

Caricamento/Collegamento dinamico

Obiettivo: ottimizzazione della memoria:

- **Caricamento dinamico**
  - in alcuni casi e` possibile caricare in memoria una funzione/procedura a tempo di esecuzione solo quando essa viene chiamata
  - **loader di collegamento rilocabile**: provvede a caricare e collegare dinamicamente la funzione al programma che la usa.

- **Collegamento dinamico**
  - una funzione/procedura viene **collegata** a un programma a tempo di esecuzione solo quando essa viene chiamata
  - la funzione puo`essere condivisa da piu` processi: problema di **visibilita`** -> compito del S.O. e` concedere/controllare:
    - ✓ l'accesso di un processo allo spazio di un altro processo
    - ✓ l'accesso di piu` processi agli stessi indirizzi

Overlay

In generale, la memoria disponibile puo` non essere sufficiente ad accogliere codice e dati di un processo.

- Una possibile soluzione a questo problema e` data dall'**overlay**: vengono mantenuti in memoria le istruzioni ed i dati:
  - che vengono utilizzati piu` frequentemente
  - che sono necessari nella fase corrente

  ❒ codice e dati di un processo vengono suddivisi (dal programmatore!) in ***overlay*** che vengono caricati e scaricati dinamicamente (dal *gestore di overlay*, di solito esterno al S.O.)

Overlay: esempio

Assembler a 2 passi: e` un programma che produce l'eseguibile di un programma assembler, mediante 2 fasi sequenziali:

1. Creazione della tabella dei simboli (Passo 1)
2. Generazione dell'eseguibile (Passo 2)

Possiamo individuare 4 componenti distinte nel codice dell'assembler:

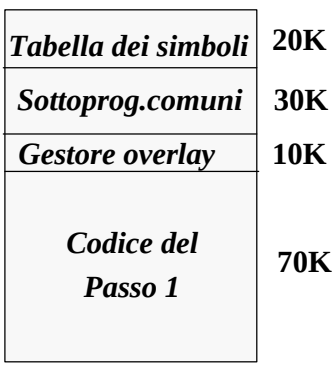
- Tabella dei simboli (ad esempio, dim. 20K)
- Sottoprogrammi comuni ai due passi (ad es., 30K)
- Codice del Passo 1 (ad es., 70K)
- Codice del passo 2 (ad es. 80K)

- Lo spazio richiesto per l'allocazione integrale dell'assembler e` quindi di 200K.

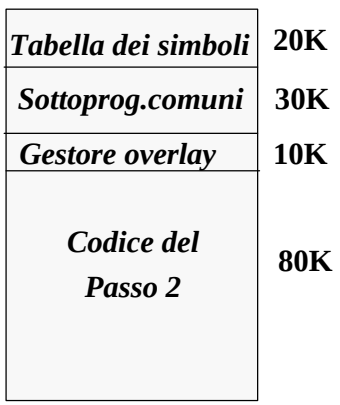
Overlay: esempio

Hp: Lo spazio libero in memoria e` di 150 K.

Soluzione: Definiamo 2 overlay da caricare in sequenza (passo1 e passo 2); caricamento/scaricamento vengono effettuati da una parte aggiuntiva di codice (gestore di overlay, dimensione 10K) aggiunta al codice dell'assembler.



Occupazione complessiva: 130K



Occupazione complessiva: 140K

Spazi di Indirizzi Logici e Fisici

La CPU genera degli indirizzi che, in generale, possono essere diversi dagli indirizzi fisici:

- indirizzi **logici**: indirizzi generati dalla CPU, riferiti allo spazio logico di indirizzamento.
- indirizzi **fisici**: indirizzi riferiti alla memoria fisica.

Binding:

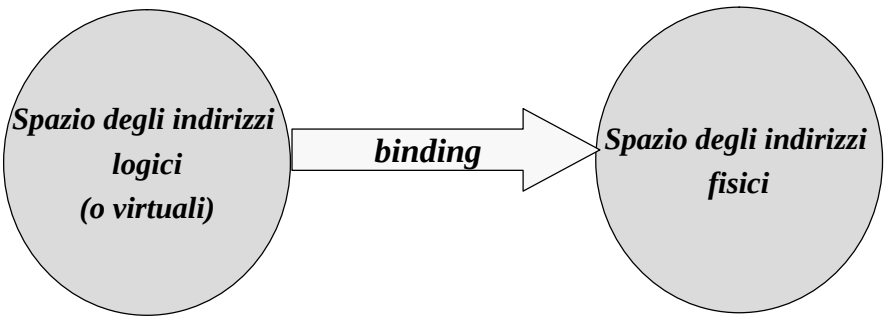
mappa indirizzi *logici* in indirizzi *fisici*

- **Binding statico** (a tempo di compilazione o di caricamento) ➔ **indirizzi logici o indirizzi fisici**
- **Binding dinamico** (a tempo di esecuzione)

  ➔ **indirizzi logici (*virtuali*) 1 indirizzi fisici**

Binding dinamico

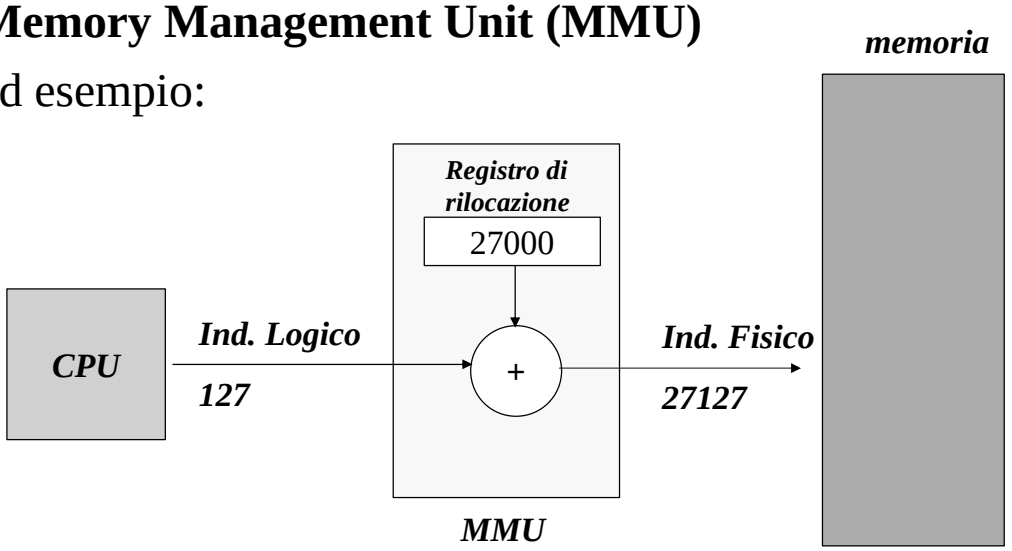
- L’associazione tra indirizzi logici e fisici viene effettuata a **run-time**:
  - possibilita` di spostare processi in memoria
  - supporto allo swapping



Binding dinamico

- Il meccanismo di binding dinamico trova supporto nell'architettura HW del sistema:
- **Memory Management Unit (MMU)**

  ad esempio:



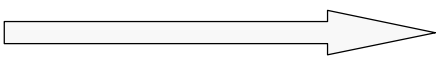
MMU

- **Registro di rilocazione**: esprime la base RL a cui riferire gli indirizzi logici

  Indirizzo fisico= RL+Indirizzo logico

- Relazione tra **spazio logico e spazio fisico** degli indirizzi:

spazio ind.logici **spazio ind. fisici**

[0, maxind]  [RL, RL+maxind]

Tecniche di allocazione della memoria centrale

Come vengono allocati codice e dati dei processi in memoria centrale ?

Varie tecniche:

- **Allocazione Contigua**
  - a partizione singola
  - a partizioni multiple
- **Allocazione non contigua**
  - paginazione
  - segmentazione

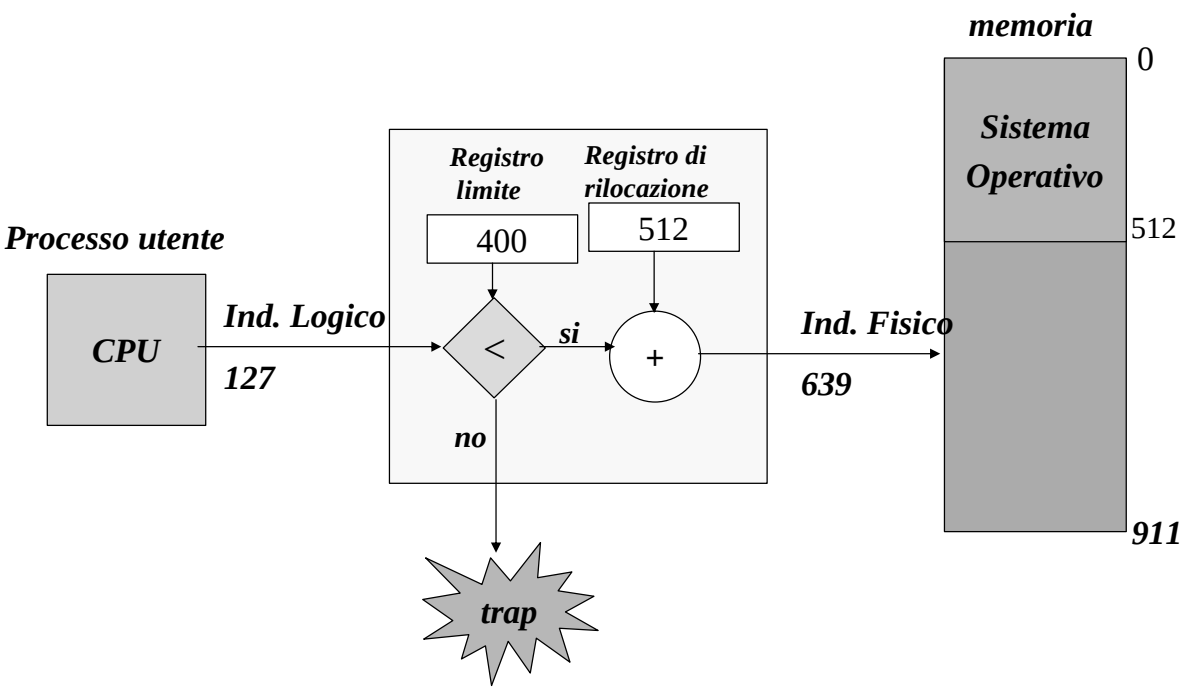
Allocazione Contigua a partizione singola

La parte di memoria disponibile per l'allocazione dei processi di utente **non e` partizionata:**

➔ un solo processo alla volta puo` essere allocato in memoria: **non c'e` multiprogrammazione**.

Di solito:

- il sistema operativo risiede nella **memoria bassa** [0, max]
- necessita` di **proteggere** codice e dati del S.O. da accessi di processi utente:
  - uso del **registro di rilocazione** (RL=max+1) per garantire la correttezza degli accessi.


memoria
0
Sistema Operativo
512
911
Registro limite
Registro di rilocazione
400
512
Processo utente
CPU
Ind. Logico
127
<
si
+
Ind. Fisico
639
no
trap

Allocazione Contigua: partizioni multiple

Multiprogrammazione ➔ necessita` di **proteggere** codice e dati di ogni processo.

- **Partizioni multiple:** ad ogni processo caricato viene associata un'area di memoria distinta (*partizione*):
  - partizioni **fisse**
  - partizioni **variabili**

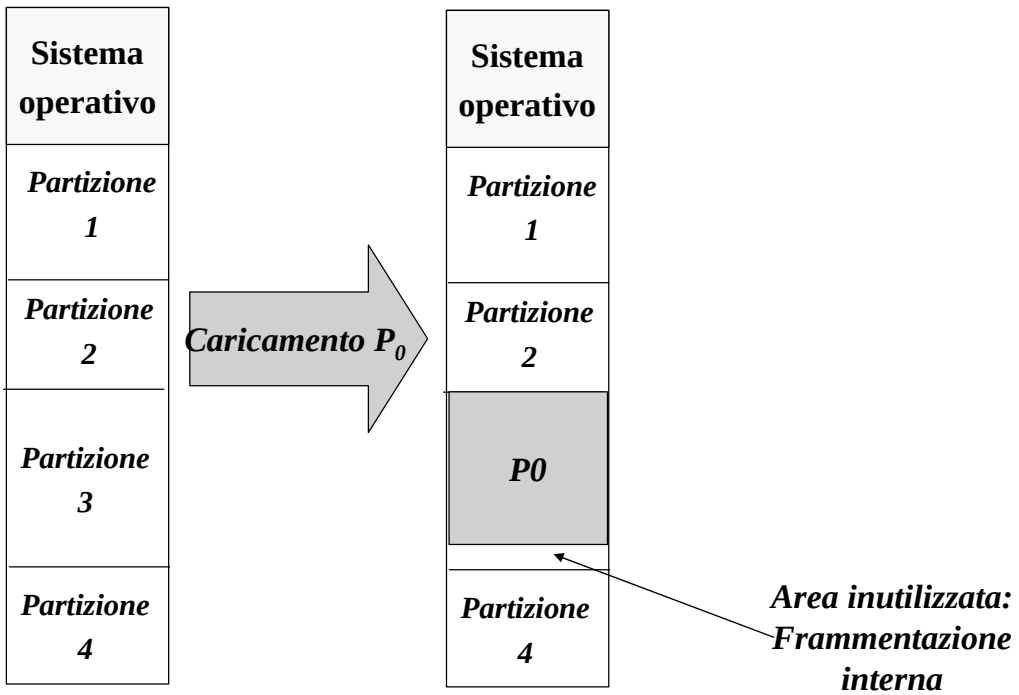
❒ **Partizioni fisse (MFT, Multiprogramming with Fixed number of Tasks):** la dimensione di ogni partizione e`fissata a priori:

- quando un processo viene schedulato, il S.O. cerca una partizione libera di dimensione sufficiente ad accoglierlo.

Problemi:

- frammentazione **interna**; **sottoutilizzo** della partizione
- Il grado di multiprogrammazione e` limitato al numero di partizioni.
- La dimensione massima dello spazio di indirizzamento di un processo e` limitata dalla dimensione della partizione piu` estesa.

Partizioni fisse



Partizioni Variabili

- **Partizioni variabili (MVT, Multiprogramming with Variable number of Tasks):** ogni partizione e` allocata **dinamicamente**, e dimensionata in base alla dimensione del processo da allocare.
  - quando un processo viene schedulato, il S.O. cerca un'area sufficientemente grande per allocarvi dinamicamente la partizione destinata ad accoglierlo.

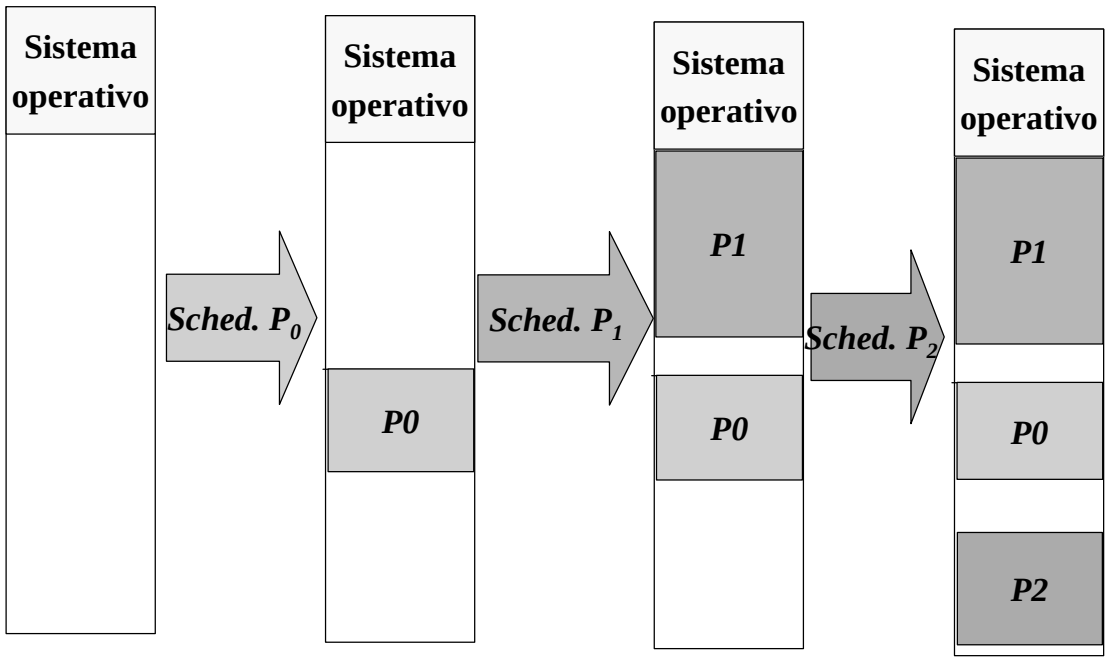
Vantaggi (rispetto a MFT):

- Si elimina la frammentazione interna (ogni partizione e` della esatta dimensione del processo)
- il grado di multiprogrammazione e` variabile
- La dimensione massima dello spazio di indirizzamento di ogni processo e` limitata dalla dimensione dello spazio fisico.

Problemi:

- scelta dell'area in cui allocare: ***best fit***, ***worst fit***, ***first fit***, *etc*.
- **frammentazione esterna;** man mano che si allocano nuove partizioni, la memoria libera e` sempre piu` frammentata
  - ➢ necessita` di **compattazione**

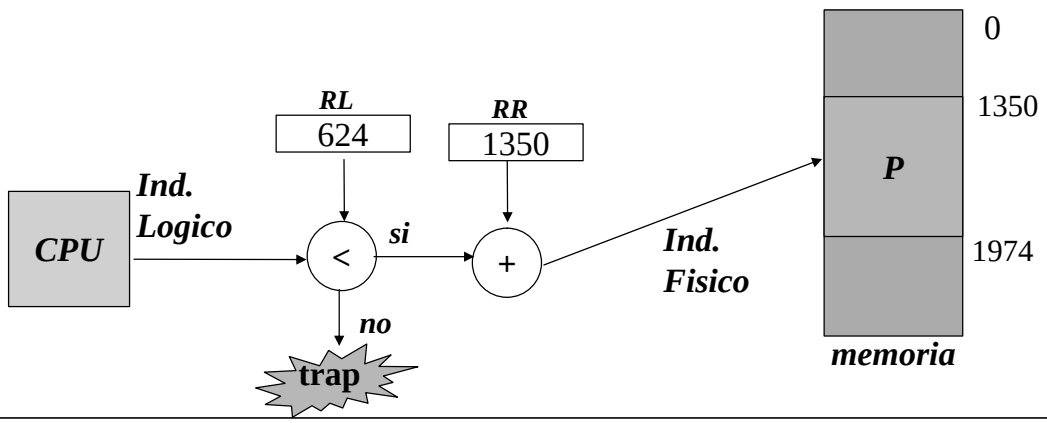
Partizioni variabili



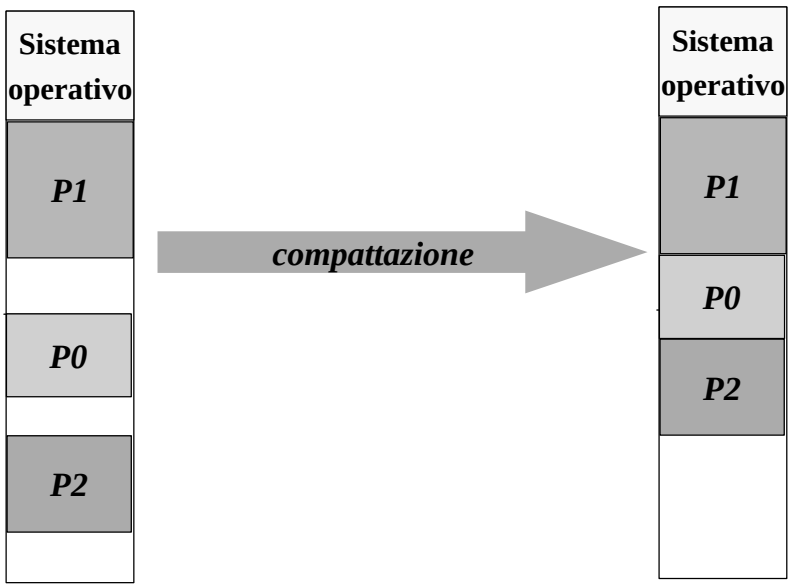
Partizioni & Protezione

- La protezione viene realizzata a livello HW mediante:
  - registro di **rilocazione** RR
  - registro **limite** RL

  ad ogni processo e` associata una coppia di valori <V_{RR}, V_{RL}>.
- Quando un processo P viene schedulato: il *dispatcher* carica RR e RL con i valori associati al processo <V_{RR}, V_{RL} >.



Compattazione



Problema: possibile crescita dinamica dei processi **-> mantenimento dello *spazio di crescita***

Paginazione

- **Allocazione contigua a partizioni multiple:** il problema principale e` la frammentazione (interna e esterna).

Paginazione -> allocazione **non contigua**:

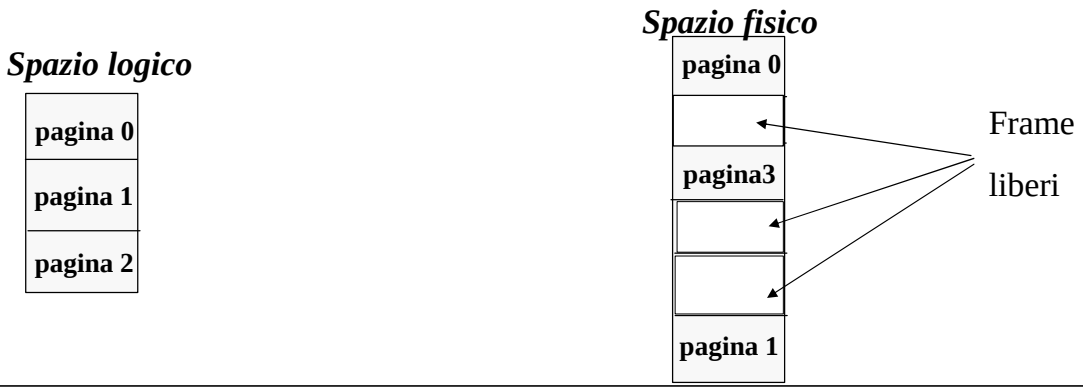
- eliminazione della frammentazione esterna
- possibilita` di riduzione della frammentazione interna a valori trascurabili.

Idea di base: partizionamento dello spazio fisico di memoria in **pagine** (***frame***) di dimensione costante e limitata (ad es. 4 K) sulle quali mappare porzioni dei processi da allocare.

Paginazione

- **Spazio Fisico:** insieme di *frame* di dimensione D_f costante prefissata
- **Spazio logico:** insieme di *pagine* di dimensione uguale a D_f

ogni *pagina logica* di un processo caricato in memoria viene mappata su una pagina fisica.



Paginazione

Vantaggi:

- Pagine logiche contigue possono essere allocate su pagine fisiche non contigue: **non c'e` frammentazione esterna**
- Le pagine sono di dimensione limitata: la frammentazione interna, per ogni processo e` **limitata dalla dimensione del frame**.
- E` possibile caricare in memoria un sottoinsieme delle pagine logiche di un processo (v. memoria virtuale).

Supporto HW a Paginazione

Struttura dell'indirizzo logico:

p	d

- **p** e` il numero di pagina logica
- **d** e` l'offset della cella rispetto all'inizio della pagina

- **Hp:** indirizzi logici di m bit (n bit per offset, e m-n per la pagina)
  - dimensione spazio logico di indirizzamento => 2^m
  - dimensione della pagina => 2^n
  - numero di pagine => 2^{m-n}

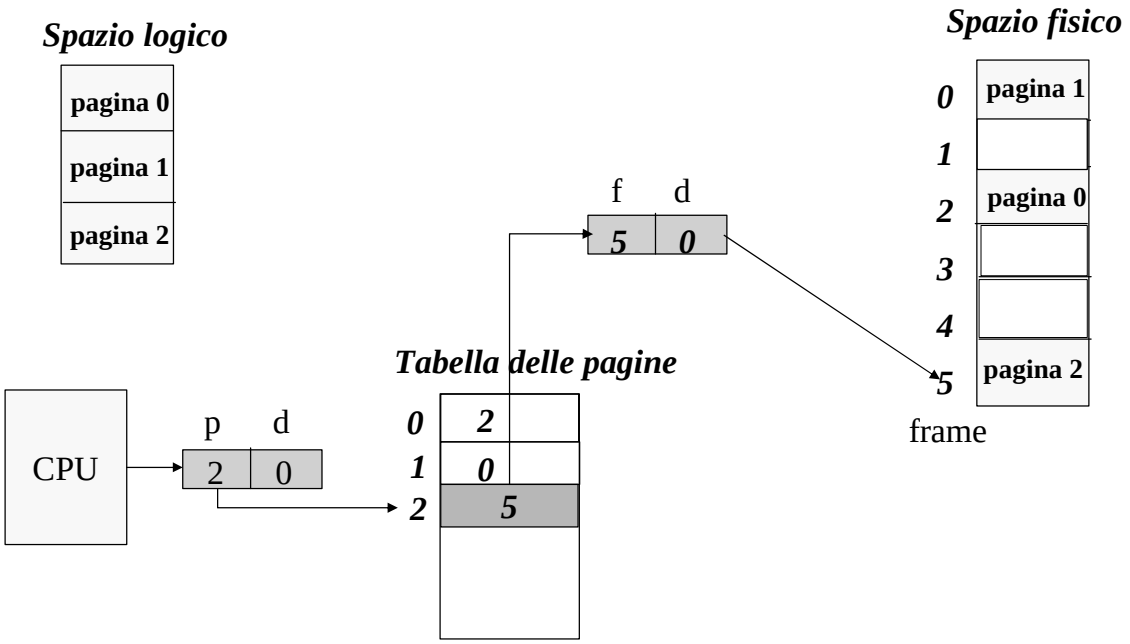
Supporto HW a Paginazione

Struttura dell'indirizzo fisico:

- **f** e` il numero di frame (pagina fisica)
- **d** e` l'offset della cella rispetto all'inizio del frame

- Il ***binding*** tra indirizzi logici e fisici puo` essere realizzato mediante la **tabella delle pagine** (associata al processo):
  - ad ogni pagina logica associa la pagina fisica sulla quale e` mappata

Supporto HW a paginazione: tabella delle pagine



Realizzazione della Tabella delle Pagine

Problemi da affrontare:

- la tabella può essere molto grande
- la traduzione (ind.logico ->ind. fisico) deve essere il più veloce possibile

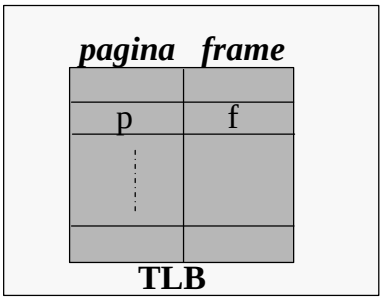
Varie soluzioni:

1. Su **registri** di CPU:
   - accesso veloce
   - cambio di contesto ***pesante***
   - dimensioni limitate della tabella
2. In **memoria centrale**: il registro *PageTableBaseRegister* (PTBR) memorizza la collocazione della tabella in memoria.
   - 2 accessi in memoria per ogni operazione (load, store)
3. Uso di **cache**: (***translation look aside buffers, TLB***) per velocizzare l'accesso.

TLB

• la tabella delle pagine è allocata in memoria centrale

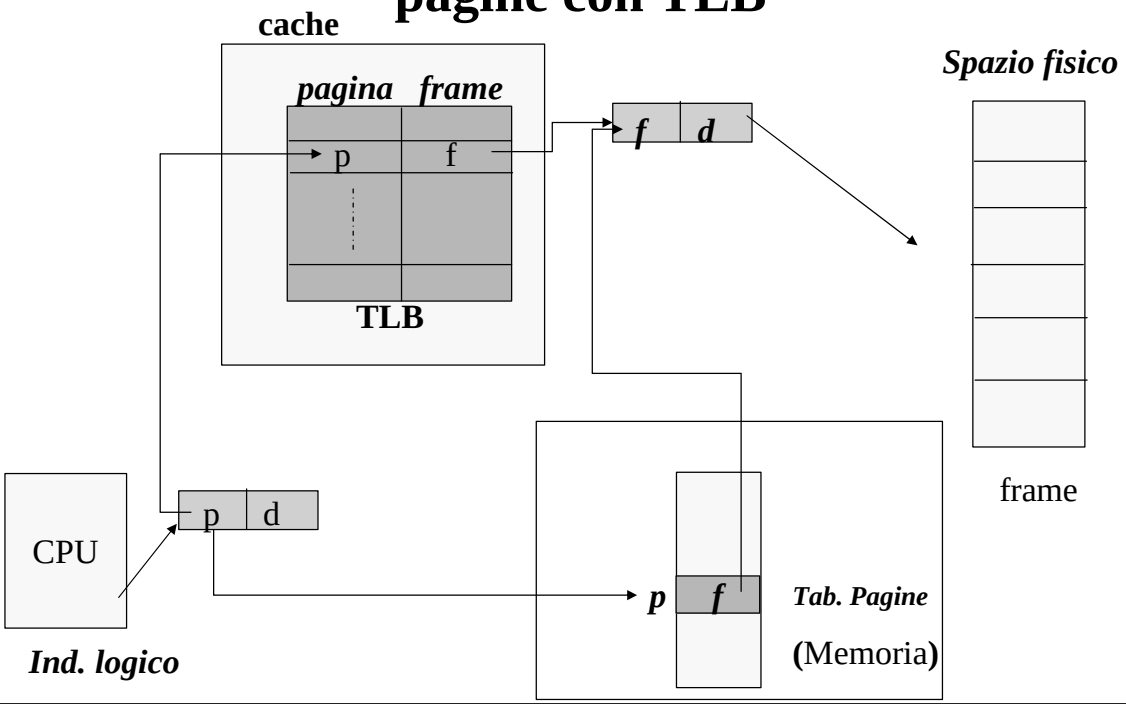
• **una parte** della tabella delle pagine (di solito, le pagine accedute piu` di frequente, o piu` di recente) è copiata **nella cache: *TLB***



cache

Se la coppia (p,f) è già presente nella cache, l'accesso è veloce, altrimenti il S.O. deve trasferire la coppia richiesta dall tabella delle pagine (in memoria centrale) alla TLB.

Supporto HW a paginazione: tabella delle pagine con TLB



Gestione della TLB

- La TLB è inizialmente vuota;
- mentre l'esecuzione procede, viene gradualmente riempita con gli indirizzi della pagine già accedute.

HIT-RATIO: percentuale di volte che una pagina viene trovata nella TLB.

- Dipende dalla dimensione della TLB (intel 486: **98**%)

Paginazione & Protezione

- La tabella delle pagine:
  - ha dimensione fissa
  - non sempre viene utilizzata completamente

Come distinguere gli elementi significativi da quelli non utilizzati?

- Bit di validità: ogni elemento contiene un bit:
  - se è settato: la entry è valida (la pagina appartiene allo spazio logico del processo)
  - se è 0: la entry non è valida
- Page Table Length Register: registro che contiene il numero degli elementi significativi nella tabella delle pagine

- In aggiunta, per ogni entry della tabella delle pagine, possono esserci uno o piu` bit di protezione che esprimono le modalita` di accesso alla pagina (es. Read-only, etc.)

Paginazione a più livelli

- Lo spazio logico di indirizzamento di un processo può essere molto esteso:
  - elevato numero di pagine
  - tabella delle pagine di grandi dimensioni

Ad esempio:

HP: indirizzi di 32 bit -> spazio logico di 4 GB
dimensione pagina 4Kb (2^{12})

- la tabella delle pagine dovrebbe contenere $2^{32}/2^{12}$ elementi-> 2^{20} elementi (circa 1 Milione)!

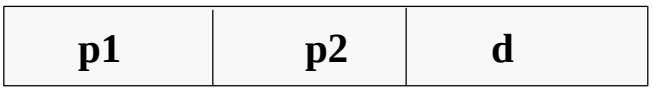
Paginazione a più livelli: allocazione **non contigua** della tabella delle pagine -> si applica ancora la **paginazione** alla **tabella della pagine**!

Esempio: paginazione a due livelli

Vengono utilizzati due livelli di tabelle delle pagine:

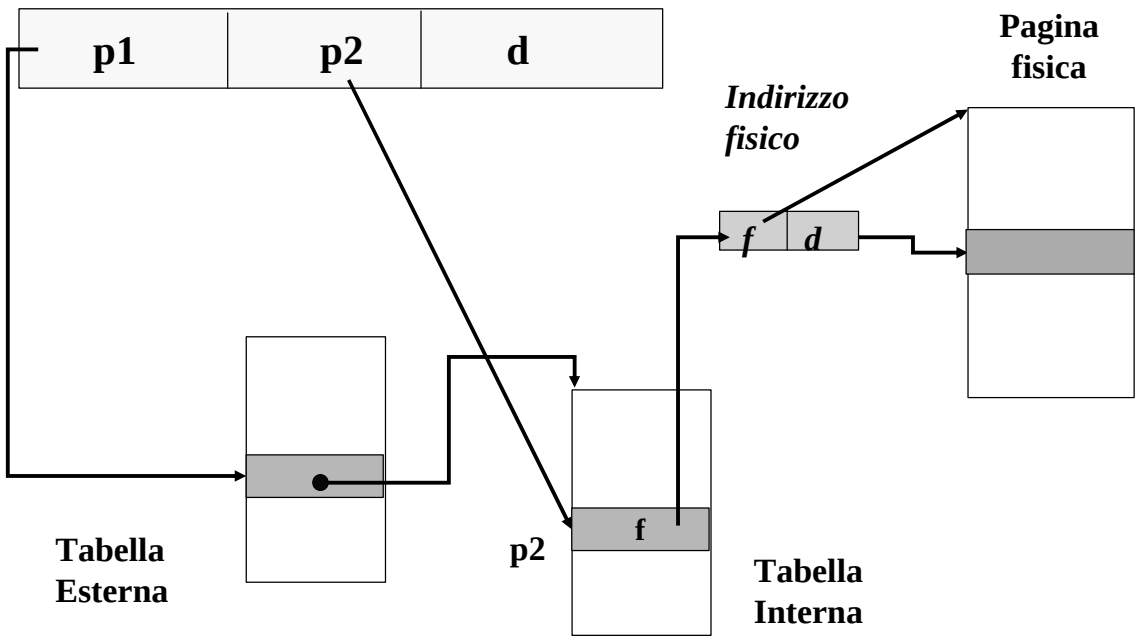
- primo livello(tabella esterna): contiene gli indirizzi delle tabelle delle pagine collocate al secondo livello (tabelle interne).

Struttura dell'indirizzo logico:



- **p1** è l'indice di pagina nella tavola esterna
- **p2** è l'offset nella tavola interna
- **d** è l'offset della cella all'interno della pagina fisica

Esempio: paginazione a due livelli



Paginazione a più livelli

Vantaggi:

- possibilità di indirizzare spazi logici di dimensioni elevate riducendo i problemi di allocazione delle tabelle
- possibilità di mantenere in memoria soltanto le pagine della tabella che servono

Svantaggio

- tempo di accesso più elevato: per tradurre un indirizzo logico sono necessari più accessi in memoria (ad esempio, 2 livelli di paginazione -> 2 accessi)

Tabella delle pagine invertita

Paginazione: per ogni processo necessità di mantenere il *binding* tra pagine logiche e frames :

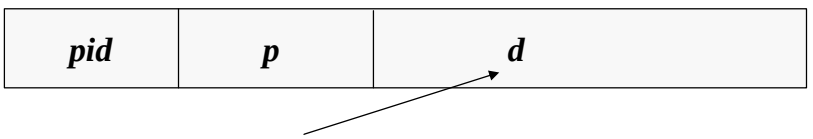
- una tabella delle pagine per ogni processo
- le dimensioni di ogni tabella possono essere elevate

➢ per limitare l'occupazione di memoria, in alcuni sistemi si usa un'unica struttura dati **globale** che ha un elemento per ogni frame :

tabella delle pagine invertita

Tabella delle pagine invertita

- Ogni elemento della tabella delle pagine invertita rappresenta un frame e, in caso di frame allocato, contiene:
  - ✓ **pid**: l'identificatore del processo al quale è assegnato il frame
  - ✓ **p**: il numero di pagina logica
- La struttura dell'indirizzo logico è, quindi:



d è l'offset all'interno della pagina.

Tabella delle pagine invertita

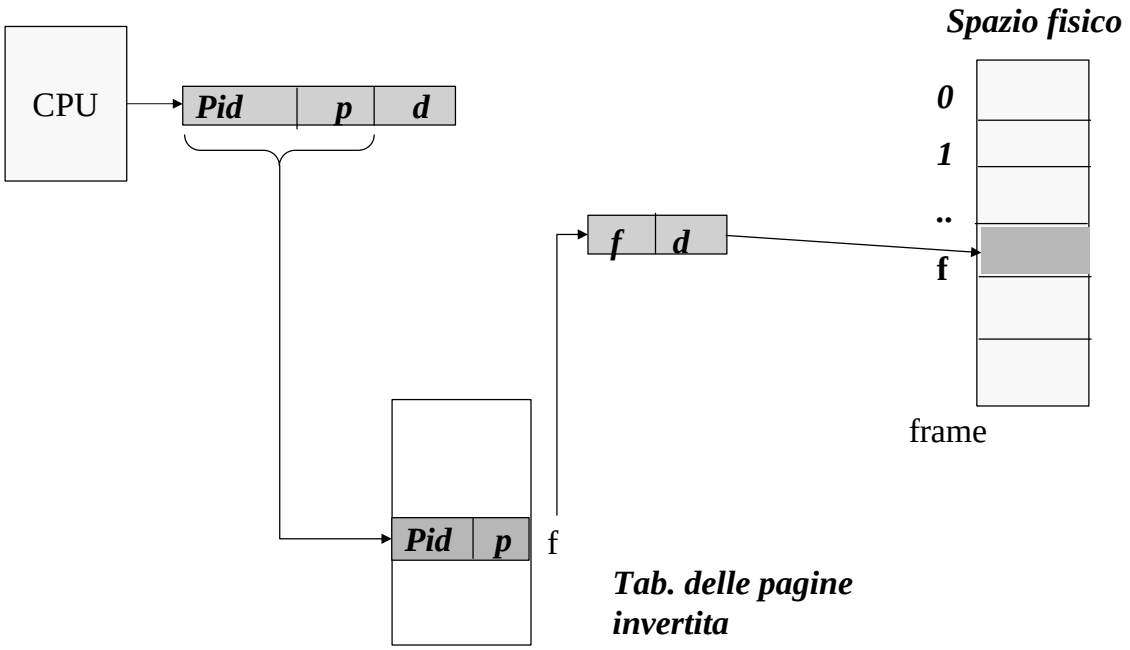


Tabella delle Pagine Invertita

Per tradurre un indirizzo logico <pid, p, d>:

- Ricerca nella tabella dell'elemento che contiene la coppia (pid,p) -> l'indice dell'elemento trovato rappresenta il **numero del frame** allocato alla pagina logica p.

- **Problemi:**
  - **tempo di ricerca** nella tabella invertita.
  - difficoltà di realizzazione della condivisione di codice tra processi (***rientranza***): come associare un frame a più pagine logiche di processi diversi?

Segmentazione

- La segmentazione si basa sul partizionamento dello spazio logico degli indirizzi di un processo in parti (***segmenti***), ognuna caratterizzata da un nome e una lunghezza.
- **Divisione semantica**: ad esempio
  - ✓ codice
  - ✓ dati
  - ✓ stack
  - ✓ heap
- Non è stabilito un ordine tra i segmenti.
- Ogni segmento viene allocato in memoria in modo contiguo.
- Ad ogni segmento il S.O. associa un intero attraverso il quale lo si può riferire.

Segmentazione

Struttura degli indirizzi logici: ogni indirizzo è costituito dalla coppia <***segmento, offset***>:

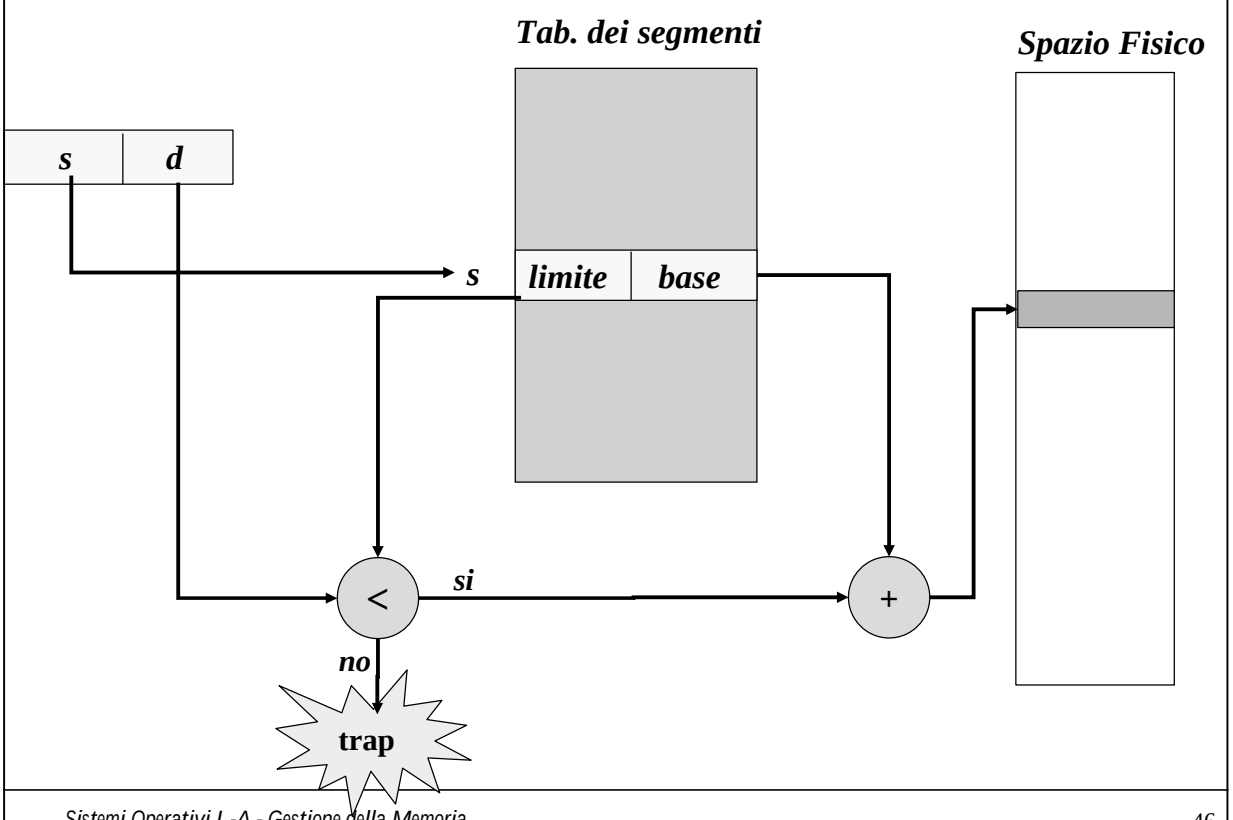
- segmento: è il numero che individua il segmento nel sistema
- Offset: posizione della cella all'interno del segmento

Supporto HW alla segmentazione:

Tabella dei segmenti: ha una entry per ogni segmento che ne descrive l'allocazione in memoria fisica mediante la coppia **<base, limite>:**

- **base**: indirizzo della prima cella del segmento nello spazio fisico
- **limite**: indica la dimensione del segmento

Segmentazione



Realizzazione della tabella dei segmenti

Tabella Globale: possibilità di dimensioni elevate

Realizzazione:

- su **registri** di CPU
- In **memoria**, mediante registri base (Segment Table Base Register, *STBR*) e limite (Segment table Length Register, *STLR*)
- Su **cache** (solo l'insieme dei segmenti usati più recentemente)

Segmentazione

È una estensione della tecnica di allocazione a partizioni variabili:

- **partizioni variabili: 1 *segmento*/processo**
- **segmentazione: più segmenti/processo**

Problema principale:

- **come nel caso delle partizioni variabili:**

  frammentazione esterna

Soluzione: allocazione dei segmenti con tecniche

- best fit
- worst fit,
- etc.

Segmentazione Paginata

Segmentazione e paginazione possono essere combinate (ad esempio Intel x86**):**

- lo spazio logico è segmentato
- ogni segmento è suddiviso in pagine

Vantaggi:

- eliminazione della frammentazione esterna (ma introduzione di frammentazione interna..)
- non è necessario mantenere in memoria l'intero segmento, ma è possibile caricare soltanto le pagine necessarie (v. Memoria virtuale)

Strutture dati:

- ☒tabella dei segmenti
- ☒una tabella delle pagine per ogni segmento

Esempio: linux

Linux adotta una gestione della memoria basata su segmentazione paginata.

Vari tipi di segmento:

- code (kernel, user)
- data (kernel, user)
- task state segments (registri dei processi per il cambio di contesto)
- ...

- I segmenti sono paginati con **paginazione a tre livelli**.

Memoria Virtuale

La **dimensione della memoria** può rappresentare un vincolo importante, riguardo a:

- dimensione dei processi
- grado di multiprogrammazione

➢ **Può essere desiderabile** un sistema di gestione della memoria che:

- consenta la presenza di **più processi** in memoria (ad es. Partizioni multiple, paginazione e segmentazione), **indipendentemente dalla dimensione dello spazio disponibile.**
- Svincoli il **grado di multiprogrammazione** dalla dimensione effettiva della memoria.

➢ **Memoria Virtuale**

Memoria Virtuale

.

Con le tecniche viste finora:

- l'intero spazio logico di ogni processo è allocato in memoria

oppure

- **overlay**, **caricamento dinamico**: si possono allocare/deallocare parti dello spazio di indirizzi:
  - **a carico del programmatore** !

Memoria Virtuale:

È un metodo di gestione della memoria che consente l'esecuzione di processi non completamente allocati in memoria.

Memoria Virtuale

Vantaggi:

- dimensione dello spazio logico degli indirizzi **illimitata**
- **grado di multiprogrammazione indipendente dalla dimensione della memoria fisica**
- **efficienza**: il caricamento di un processo e lo swapping determinano un costo minore (meno I/O)
- **Astrazione**:il programmatore non deve preoccuparsi dei vincoli relativi alla dimensione della memoria, e può concentrarsi sul problema che deve risolvere.

Paginazione su Richiesta

Di solito: la memoria virtuale è realizzata mediante tecniche di **paginazione su richiesta:**

- Tutte le pagine di ogni processo risiedono in memoria secondaria (***backing store***); durante l'esecuzione alcune di esse vengono trasferite, all'occorrenza, in memoria centrale.

Pager: è un modulo del S.O. che realizza i trasferimenti delle pagine da/verso memoria secondaria/memoria centrale (e` uno "***swapper"*** di pagine).

- **Paginazione su richiesta**: il pager è ***pigro:*** trasferisce in memoria centrale una pagina soltanto se ritenuta necessaria

Paginazione su richiesta

Esecuzione di un processo: puo` richiedere lo *swap-in* del processo:

- **Swapper: gestisce i trasferimenti di *interi* processi (mem.centrale←→mem. secondaria)**
- Pager**: gestisce i trasferimenti di singole pagine**

Prima di eseguire lo *swap in* di un processo:

- il pager prevede le pagine di cui (probabilmente) il processo avrà bisogno inizialmente → caricamento.

HW necessario:

- tabella delle pagine (con PTBR, PTLR, e/o TLB etc.)
- **memoria secondaria** (*backing store*) e strutture necessarie per la sua gestione (uso di dischi veloci)

Paginazione su richiesta

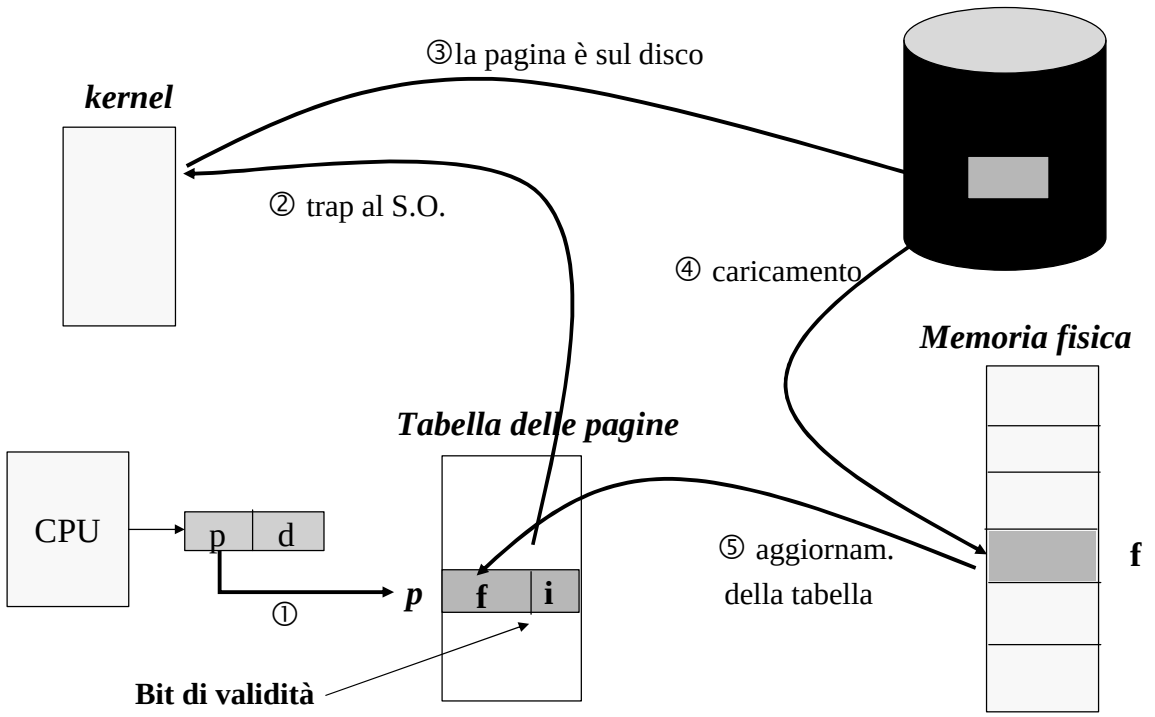
- In generale, una pagina dello spazio logico di un processo:
  - può essere allocata in **memoria centrale**
  - può essere in **memoria secondaria**

Come distinguere i due casi ?

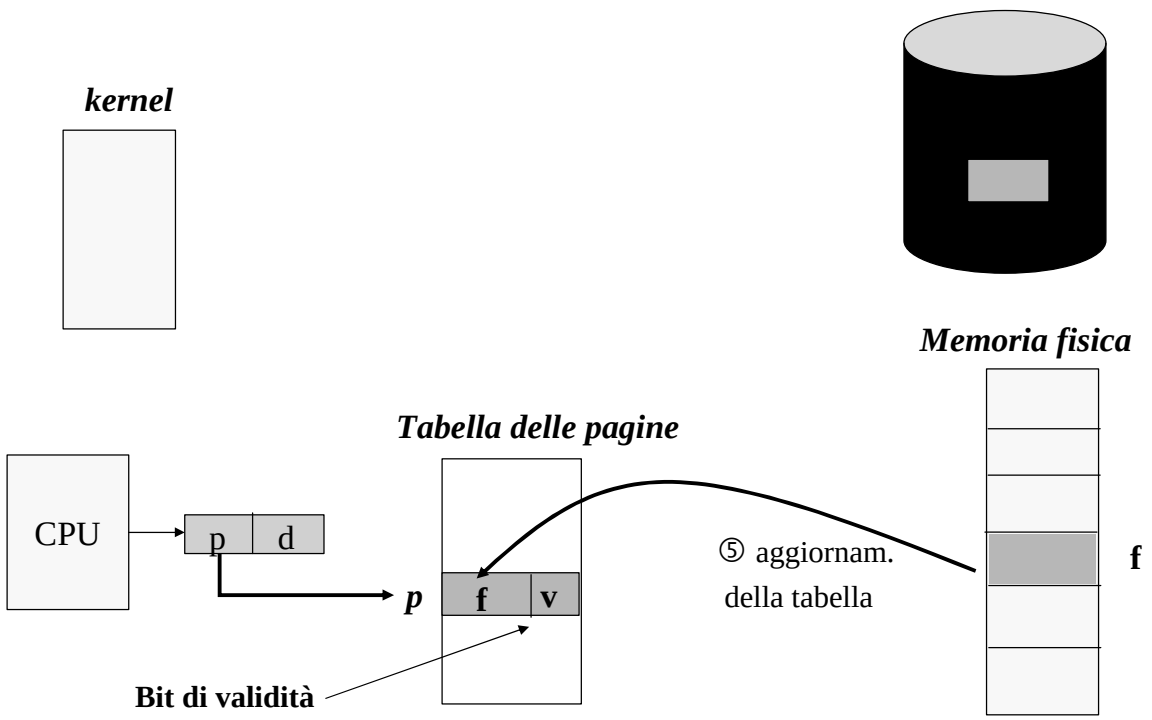
La tabella delle pagine contiene il **bit di validità**, per indicare:

- se la pagina è presente in memoria
- è in memoria secondaria, oppure è invalida ($\notin$ spazio logico del processo) ➔**interruzione** al S.O. (***page fault***)

Paginazione su Richiesta: page fault



page fault: situazione finale



Trattamento del page fault

Quando il kernel riceve l'interruzione dovuta al page fault:

0. Salvataggio del contesto di esecuzione del processo (registri, stato, tab. pagina)

1. Verifica del motivo del page fault (mediante una tabella interna al kernel):

- riferimento **illegale (violazione delle politiche di protezione):** ➔ terminazione del processo
- riferimento **legale**: la pagina è in memoria secondaria

2. Copia della pagina in un frame libero.

3. Aggiornamento della tabella delle pagine.

4. Ripristino del processo: esecuzione dell'istruzione interrotta dal page fault.

Paginazione su richiesta: sovrallocazione

- In seguito a un page fault:
  - se è necessario caricare una pagina in memoria centrale, può darsi che **non ci siano frames liberi**:

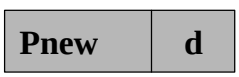
sovrallocazione.

Soluzione:

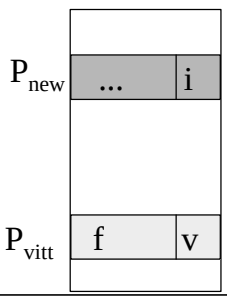
- **sostituzione** di una pagina P_{vitt}(***vittima***) allocata in memoria con la pagina P_{new} da caricare:
  1. Individuazione della vittima P_{vitt}
  2. Salvataggio di P_{vitt} su disco
  3. Caricamento di P_{new} nel frame liberato
  4. Aggiornamento tabelle
  5. Ripresa del processo

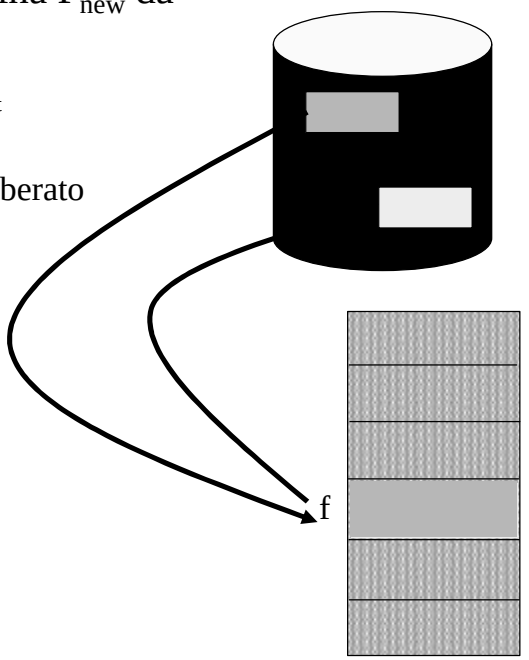
Sostituzione di pagine

- **Memoria completamente allocata: sostituisco** la pagina (***vittima***) P_{vitt} con la pagina P_{new} da caricare:
  1. Individuazione della vittima P_{vitt}
  2. Salvataggio di P_{vitt} su disco
  3. Caricamento di P_{new} nel frame liberato
  4. Aggiornamento tabelle
  5. Ripresa del processo



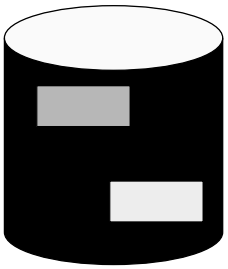
Ind. Logico



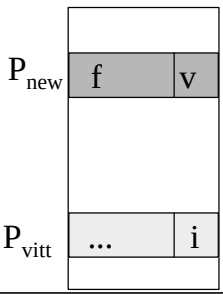


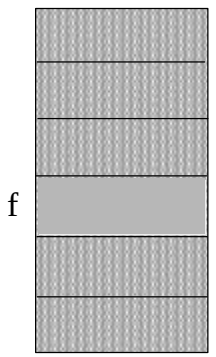
Sostituzione di pagine

Situazione finale



Ind. Logico





Sostituzione di pagine

In generale: la sostituzione di una pagina può richiedere **2 trasferimenti** da/verso il disco:

- per scaricare la vittima
- per caricare la pagina nuova

Però: è possibile che la vittima non sia stata modificata rispetto alla copia residente sul disco; ad esempio:

- pagine di codice (*read-only*)
- pagine contenenti dati che non sono stati modificati durante la permanenza in memoria

➢ In questo caso la copia della vittima sul disco può essere **evitata:**

➔ introduzione del ***bit di modifica* (*dirty bit*)**

Dirty bit

Per rendere **più efficiente** il trattamento del page fault in caso di **sovrallocazione**:

- si introduce in ogni elemento della tabella delle pagine un **bit di modifica** (***dirty bit***):
  - se è settato, significa che la pagina ha subito almeno un aggiornamento da quando è caricata in memoria
  - se è 0: la pagina non è stata modificata
- L'algoritmo di sostituzione esamina il bit di modifica della vittima:
  - esegue lo *swap-out* della vittima **solo se il *dirty bit* è settato**!

Sostituzione di pagine

Come individuare la *vittima*?

Esistono vari **algoritmi (*politiche*) di sostituzione** delle pagine; ad esempio:

- **FIFO**
- ***Least Recently Used***
- ***Least Frequently Used***
- **...**

Algoritmi di sostituzione

La finalità di ogni algoritmo di sostituzione delle pagine è sostituire quelle pagine la cui probabilità di essere accedute a breve termine è ***bassa***.

Algoritmi:

- ***LFU*** (Least Frequently Used): viene sostituita la pagina che è stata usata meno frequentemente (in un intervallo di tempo prefissato):
  - è necessario associare un **contatore** degli accessi ad ogni pagina:

  ➔La vittima e` quella con il minimo valore del contatore.

Algoritmi di sostituzione

- ***FIFO*** : viene sostituita la pagina che è da più tempo caricata in memoria (indipendentemente dal suo uso)
  - è necessario memorizzare la ***cronologia*** dei caricamenti in memoria

- ***LRU*** (Least Recently Used): viene sostituita la pagina che è stata usata meno recentemente:
  - è necessario registrare la **sequenza degli accessi** alle pagine in memoria
  - **overhead**, dovuto all'aggiornamento della sequenza degli accessi per ogni accesso in memoria

Algoritmi di sostituzione

- ***Implementazione LRU***: e` necessario registrare la sequenza temporale di accessi alle pagine. Soluzioni:
  - ***Time stamping*** : l'elemento della tabella delle pagine contiene un campo che rappresenta l'istante in cui e` avvenuto l'ultimo accesso alla pagina.
    - Costo della ricerca della vittima
  - ***"Stack":*** viene introdotta una struttura dati tipo stack in cui ogni elemento rappresenta una pagina;il riferimento a una pagina provoca lo spostamento dell'elemento che rappresenta la pagina al top dello stack => il ***bottom*** contiene la pagina LRU
    - La gestione puo` essere costosa, ma non c'e` ricerca.

Algoritmi di sostituzione:
LRU approssimato

Spesso si utilizzano versioni **semplificate** di questo algoritmo , introducendo, al posto della sequenza degli accessi, un **bit di uso** associato alla pagina, gestito come segue:

- al momento del caricamento è **inizializzato** a 0
- quando la pagina viene acceduta, viene **settato**
- ***periodicamente***, i bit di uso vengono resettati

- viene sostituita una pagina con bit di uso==0; il criterio di scelta, ad esempio, potrebbe inoltre considerare il ***dirty bit***:
  - tra tutte le pagine non usate di recente (**bit di uso==0**), ne viene scelta una che non è stata aggiornata (**dirty bit=0**).

Località dei programmi

Si è osservato che ogni programma, in una certa fase di esecuzione:

- usa solo un *sottoinsieme* delle sue pagine logiche;
- il sottoinsieme delle pagine effettivamente utilizzate varia *lentamente* nel tempo;

- **Località spaziale**:
  - alta probabilità di accedere a locazioni ***vicine*** (nello spazio logico/virtuale) a locazioni appena accedute (ad esempio, elementi di un vettore, codice sequenziale, etc.)
- **Località temporale**:
  - alta probabilità di accesso a locazioni accedute di recente (ad es. cicli) -> v. Algoritmo LRU

Working Set

La paginazione su domanda *pura*, carica una pagina soltanto se strettamente necessaria:

- possibilita` di ***trashing***: il processo impiega piu` tempo per la paginazione che per l'esecuzione.

In alternativa: tecniche di gestione della memoria che si basano su ***pre paginazione***:

- si prevede il set di pagine di cui il processo da caricare ha bisogno per la prossima fase di esecuzione:

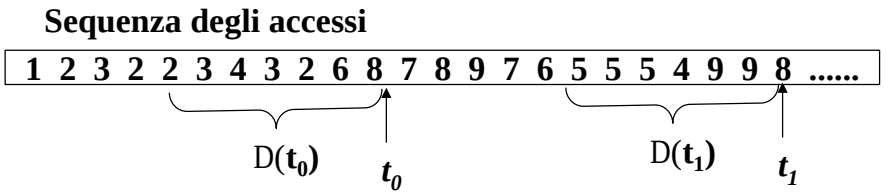
working set

- Il working set puo` essere individuato in base a criteri di

localita` temporale

Working Set

- **Working set**: dato un intero D, il working set di un processo P (nell'istante t) e` l'insieme di pagine Δ(t) indirizzate da P nei piu` recenti Δ riferimenti.
- Δ definisce la "finestra" del working set.



$D(t_0)=\{2,3,4,6,8\}$
$D(t_1)=\{5,4,9,8\}$

Working Set

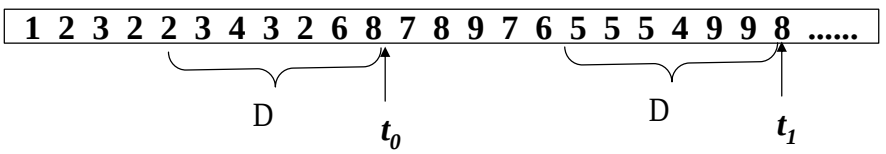
- Il caricamento di un processo consiste nel caricamento di un ***working set iniziale***
- il sistema mantiene il *working set* di ogni processo aggiornandolo dinamicamente, in base al principio di *località temporale*:
  - all'istante *t* vengono mantenute le pagine usate dal processo nell'ultima finestra $\Delta(t)$;
  - Le altre pagine (esterne al $\Delta(t)$) possono essere sostituite.

Vantaggio:

- **riduzione del numero di *page fault***

Working Set

Sequenza degli accessi



WorkingSet(t_0)={2,3,4,6,8}

WorkingSet(t_1)={5,4,9,8}

La dimensione del working set puo` variare nel tempo!

Working Set

- Il parametro Δ caratterizza il working set, esprimendo l'estensione della finestra dei riferimenti:
  - D **piccolo**: il working set non e` sufficiente a garantire localita` (e a contenere il numero dei page fault)
  - D **grande**: allocazione di pagine non necessarie

- **Ad ogni istante**:

  per ogni processo Pi possiamo individuare la dimensione corrente del suo working set $\mathbf{WSS_i}$ **:**

  $\mathbf{D = \Sigma_i WSS_i}$ e` la richiesta totale di frame**.**

Working Set & Swapping

D= Σ_i**WSS$_i$** e` la richiesta totale di frame

Se *m* e` il numero totale di frame liberi:

- **D<m: puo` esserci spazio disponibile per l'allocazione di un nuovo processo**
- **D>m: uno (o piu`) processi devono essere sottoposti a swapping**

Un esempio: la gestione della memoria in Unix (prime versioni)

In Unix lo spazio logico è **segmentato**:

- nelle **prime versioni** (prima di BSD v.3), l'allocazione dei segmenti era contigua:
  - segmentazione pura
  - non c'era memoria virtuale
- in caso di difficoltà di allocazione dei processi:
  - swapping dell'intero spazio degli indirizzi
- Condivisione di codice:
  - possibilità di evitare trasferimenti di codice da memoria secondaria a memoria centrale ➔minor overhead di swapping

La gestione della memoria in Unix (prime versioni)

- Tecnica di allocazione (**contigua**) dei segmenti:
  - sia per l'allocazione in memoria centrale, che in memoria secondaria (swap-out): **first fit**

Problemi:

- frammentazione esterna
- stretta influenza della dimensione dello spazio fisico sulla gestione dei processi in multiprogrammazione.
- crescita dinamica dello spazio logico ➔possibilità di riallocazione di processi già caricati in memoria

Unix: Swapping

In assenza di memoria virtuale, lo ***swapper*** ricopre un ruolo chiave per la gestione delle contese di memoria da parte dei diversi processi:

- periodicamente (ad es, nelle prime versioni ogni 4 sec.) lo ***swapper*** viene attivato per provvedere (*eventualmente*) a **swap-in** e **swap-out** di processi:
  - **swap-out**:
    - processi inattivi (sleeping)
    - processi “ingombranti”
    - processi da più tempo in memoria
  - **swap-in**:
    - processi piccoli
    - processi da più tempo ***swappati***

La gestione della memoria in Unix (versioni moderne)

Da BSD v.3 in poi:

- **segmentazione paginata**
- **memoria virtuale** tramite **paginazione su richiesta**

➢ L'allocazione di ogni segmento **non è contigua**:

- si risolve il problema della frammentazione esterna.
- frammentazione interna trascurabile (pagine di dimensioni piccole)

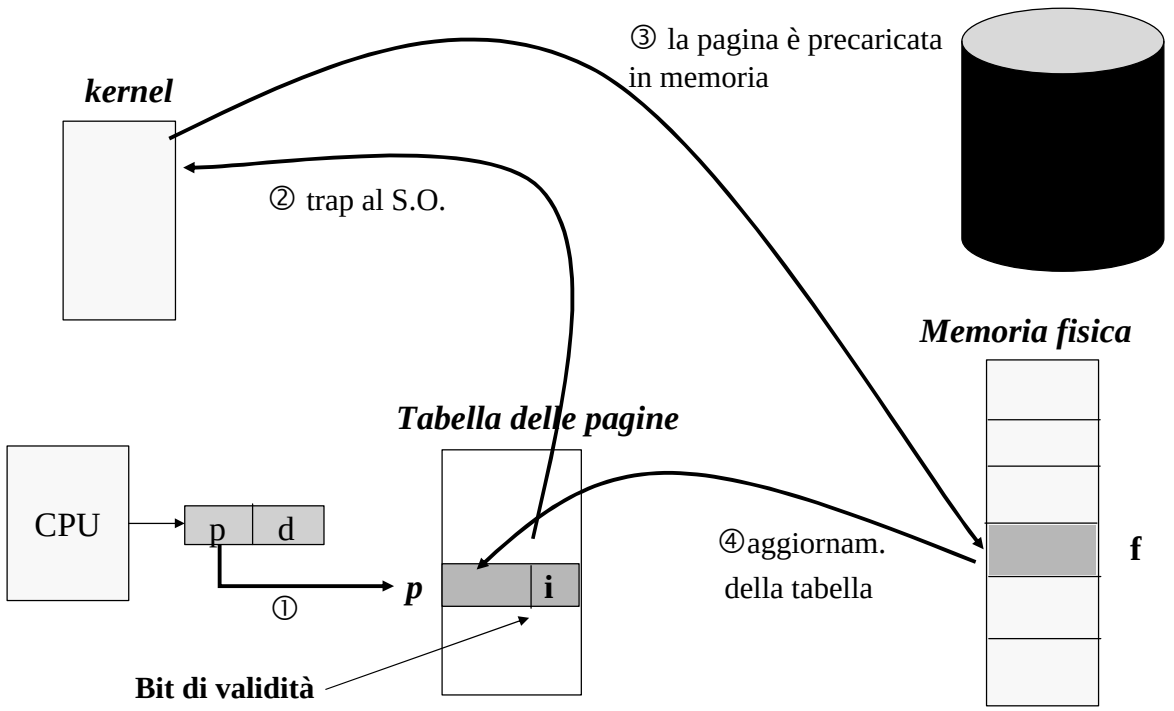
La gestione della memoria in Unix (versioni moderne)

Paginazione su richiesta:

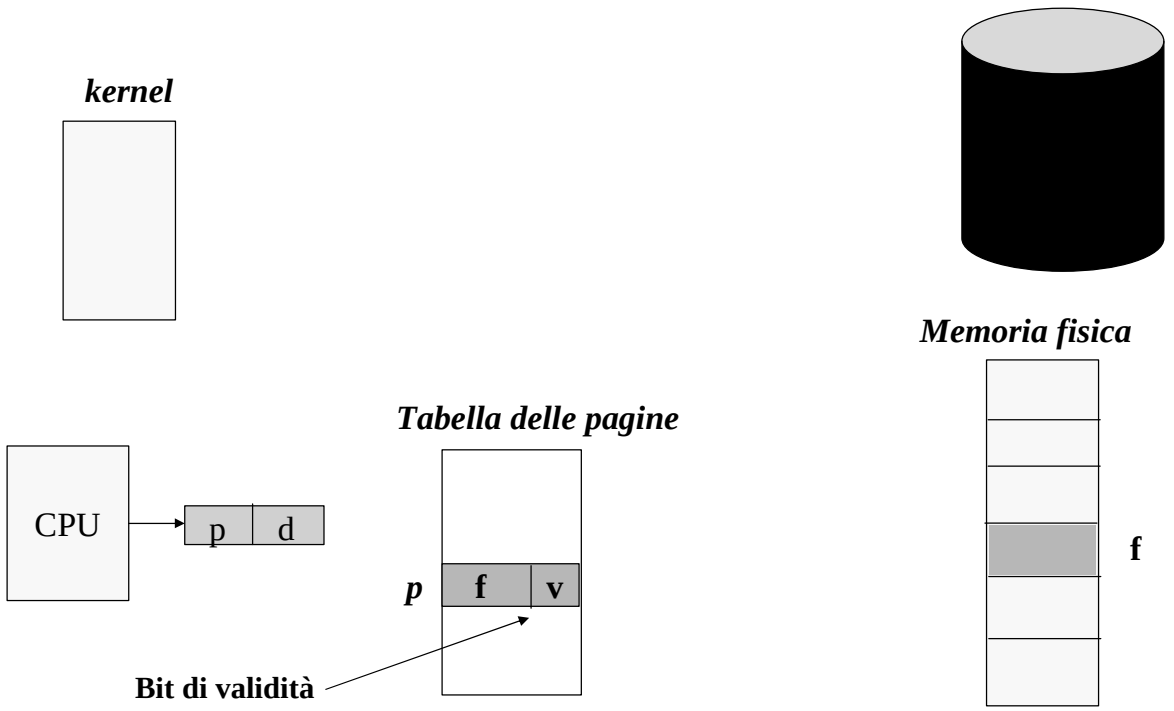
- **prepaginazione**: uso dei frame liberi per pre-caricare pagine non necessarie nei frame liberi:
  - quando avviene un page fault, se la pagina è già in un frame libero, basta soltanto modificare:
    - la tabella delle pagine
    - la lista dei frame liberi

- ***Core map***: struttura dati interna al kernel che descrive lo stato di allocazione dei frame e che viene consultata in caso di page fault.

Unix: page-fault in caso di pre-paginazione



Unix: page-fault in caso di pre-paginazione situazione finale



Unix: algoritmo di sostituzione

LRU modificato, o algoritmo di seconda chance

(*BSD v. 4.3 Tahoe*):

- ad ogni pagina viene associato un **bit di uso**, gestito come segue:
  - al momento del caricamento è **inizializzato** a 0;
  - quando la pagina viene acceduta, viene **settato;**
  - nella fase di **ricerca di una vittima**, vengono esaminati i bit di uso di tutte le pagine in memoria:
    - se una pagina ha il bit di uso a 1, viene posto a 0
    - se una pagina ha il bit di uso a 0, viene selezionata come vittima

Unix: algoritmo di sostituzione

Sostituzione della vittima:

- la pagina viene resa *invalida*
- il frame selezionato viene inserito nella lista dei frame liberi
  - se c'è il ***dirty bit***:
    - dirty bit=1 ➔la pagina va anche copiata in memoria secondaria
  - se non c'è il ***dirty bit*** ➔la pagina va comunque copiata in memoria secondaria

- L'algoritmo di sostituzione viene eseguito dal ***pagedaemon*** (pid=2).

Unix: sostituzione delle pagine

Viene attivata quando il numero totale di frame liberi è ritenuto insufficiente (minore del valore ***lotsfree***) :

- **Parametri**:
  - ***lotsfree***: numero minimo di frame liberi per evitare paginazione
  - ***minfree:*** numero minimo di frame liberi necessari per evitare lo swapping dei processi
  - ***desfree***: numero minimo di frame *desiderabili*

lotsfree > desfree > minfree

Scheduling,Paginazione e swapping

- Lo **scheduler** attiva l'algoritmo di sostituzione se:
  - il numero di frame liberi è minore di lotsfree
- Se il sistema di paginazione è **sovraccarico**, cioè:
  - carico elevato
  - numero di frame liberi <***minfree***
  - numero medio di frame nell'unità di tempo <***desfree***

➢ lo ***scheduler*** attiva lo ***swapper*** (al massimo ogni secondo).

- Il sistema evita che il ***pagedaemon*** usi più del 10% del tempo totale di CPU: attivazione (al massimo) ogni 250 ms.