La gestione della memoria

Lezione 8
Sistemi Operativi

Background

- Un programma per poter essere eseguito deve essere caricato in memoria
- Per questioni prestazionali devono essere caricati in memoria contemporaneamente più processi
- L'accesso a memoria centrale è una delle operazioni più frequenti effettuate da un processo
  - Accesso ai dati e al codice
- La tecnica di gestione della memoria ha quindi un effetto determinante sulle prestazioni del sistema

Prestazioni

- Calcoliamo l'utilizzo della CPU in un sistema monoprogrammato
- Se l'unico processo spende in media 40% del tempo di esecuzione in attesa di I/O, l'utilizzo complessivo della CPU è

$$0.6 = 1 - 0.4$$

- Per migliorare questo parametro dobbiamo ridurre il tempo in cui la CPU è inattiva
  - Fare in modo che quando un processo esegue I/O il microprocessore venga usato per altre attività

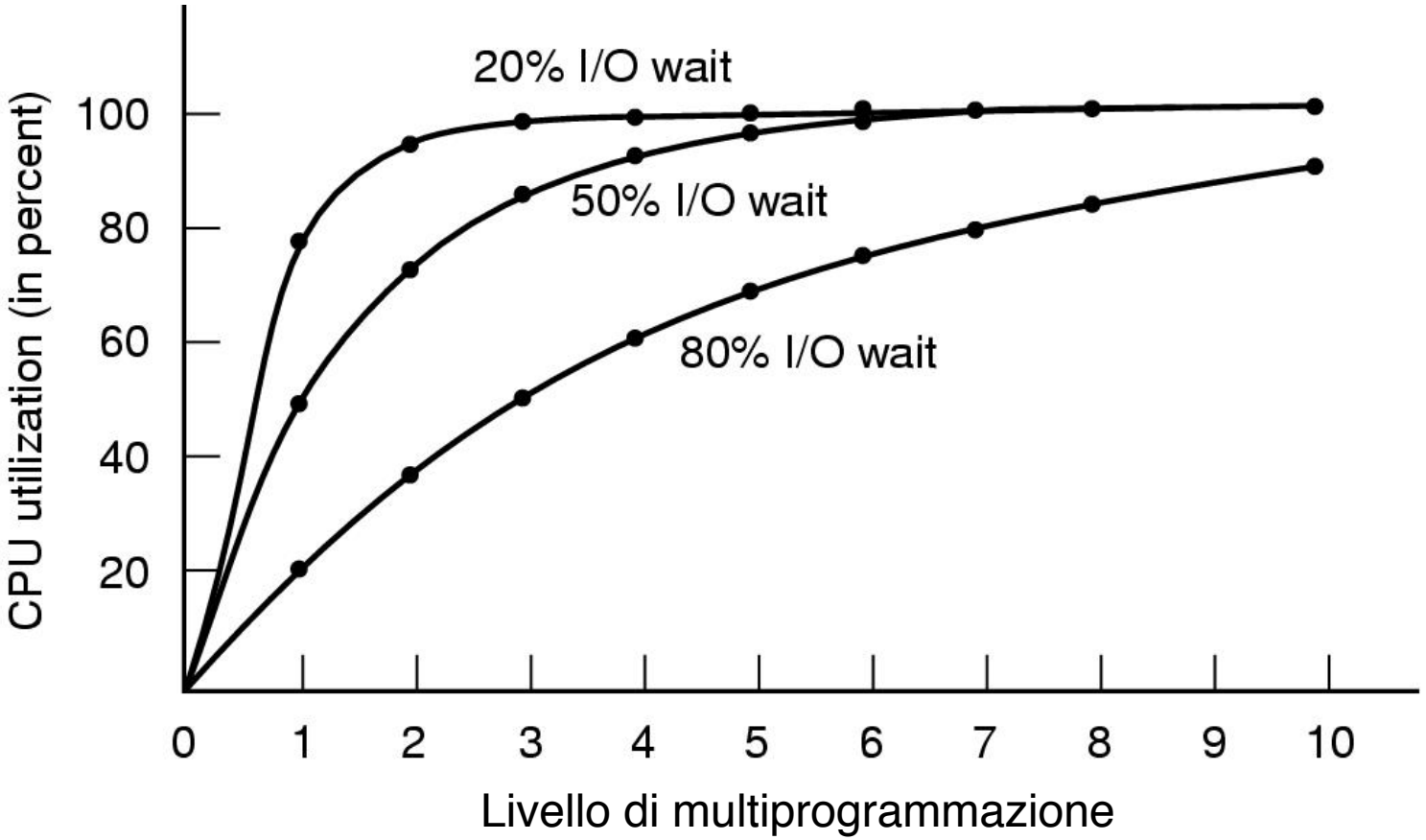
Prestazioni

- Con due processi in esecuzione, ciascuno con percentuale di I/O pari al 40%, l'utilizzo della CPU diventa

$$0.84 = 1 - 0.16$$

- Generalizzando
  - Siano dati n processi indipendenti
  - Ciascun processo spende una frazione p del proprio tempo di esecuzione in attesa di I/O
    - Non usa la CPU
  - La probabilità che tutti gli n processi siano in attesa è p^n
  - L'utilizzazione della CPU è $1 - p^n$
    - Quando tutti i processi sono in attesa, la CPU è idle

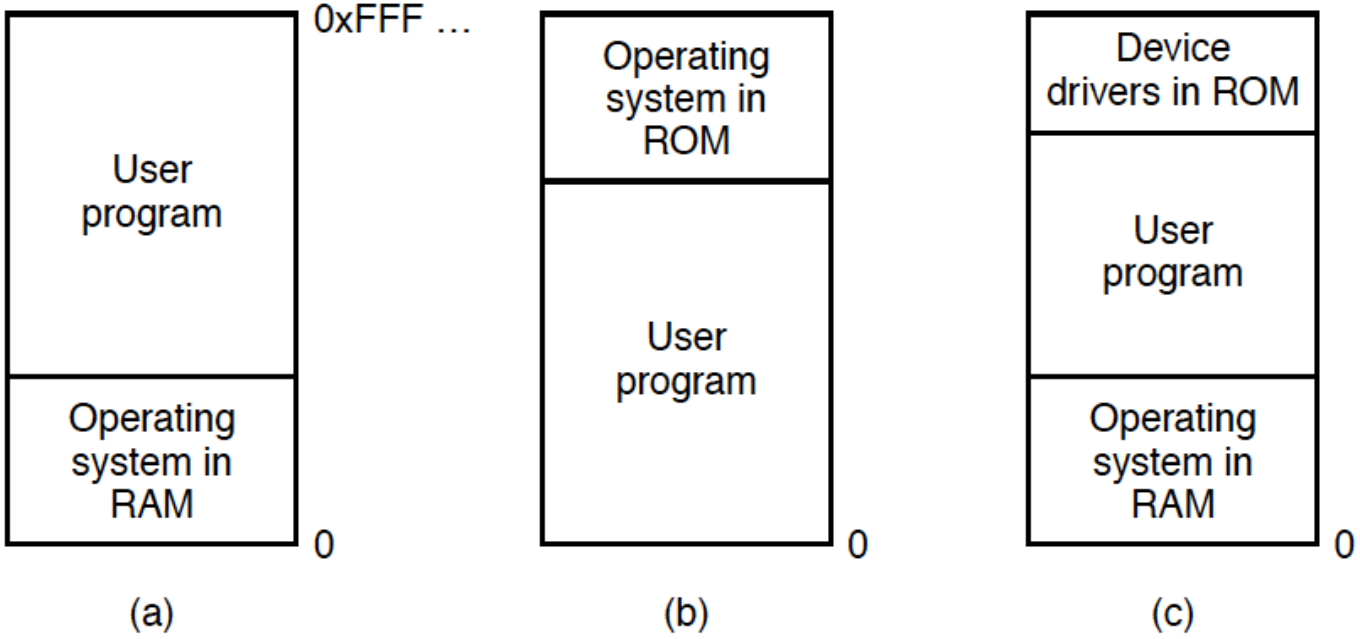
Memoria e prestazioni



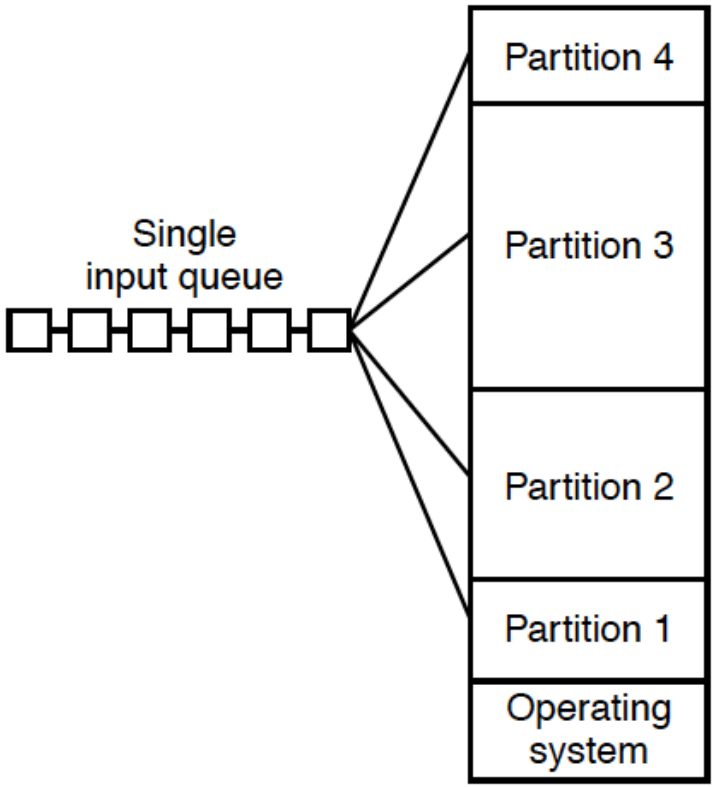
Osservazioni

- Se i programmi sono I/O bound, molti programmi devono essere in esecuzione per raggiungere un elevato utilizzo di CPU
- Aumentare la memoria centrale di un sistema non fa aumentare l'utilizzo di CPU in misura proporzionale all'incremento del livello di multiprogrammazione
  - L'efficacia dipende dal livello corrente di utilizzo
    - Pendenza della curva di utilizzo

Sistemi monoprogrammati



Sistemi multiprogrammati



Requisiti Memory Manager

- La presenza contemporanea in memoria di più processi e la loro esecuzione "concorrente" è possibile solo se il MM è in grado di garantire la loro coesistenza evitando ogni tipo di interferenza non espressamente richiesta (vedi sezioni critiche)

Il problema

Due utenti eseguono due copie indipendenti di questo programma.

```
int a[20];
…
int i, sum=0;
for (i=0; i<20; i++)
      sum += a[i];
```

```
    movl   $0, %eax
    movl   $0, %ebx
sumloop:
    movl   1000(,%eax,4),%ecx
    addl   %ecx, %ebx
    incl   %eax
    cmpl   $19, %eax
    jle    sumloop
```

Programma 2

```
int a[20];
…
int i, sum=0;
for (i=0; i<20; i++)
   a[i] = a[i] + 10;
```

```
    movl   $0, %eax
    movl   $0, %ebx
sumloop:
    movl   1000(,%eax,4),%ecx
    addl   $10, %ecx
    movl   %ecx,1000(,%eax,4)
    incl   %eax
    cmpl   $19, %eax
    jle    sumloop
```

Il problema

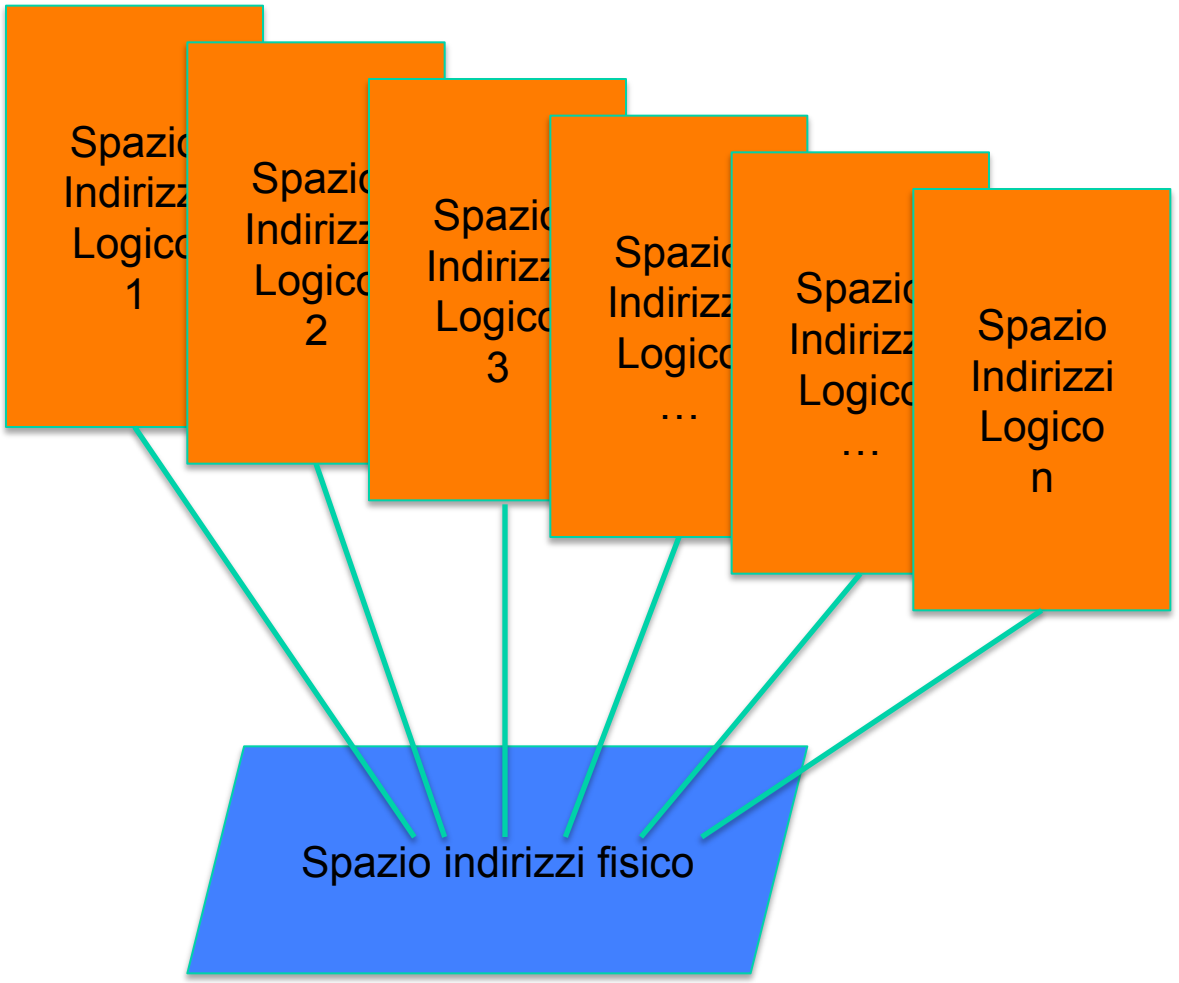
Spazio degli Indirizzi: un insieme di indirizzi di memoria

- Un sistema è caratterizzato da:
  - Un unico spazio di indirizzi fisici, in corrispondenza di ogni indirizzo vi è una locazione di memoria presente nella RAM
- Un processo è caratterizzato da
  - uno spazio di indirizzi virtuali (o logici), ad ogni indirizzo logico è associato un dato o un'istruzione
- Un programma (sorgente) è caratterizzato da
  - Uno spazio di indirizzi simbolici, costituiti dai nomi che il programmatore ha usato per denotare variabili e istruzioni

Separazione spazio degli indirizzi

- Uno degli obiettivi del MM è: mappare spazi di indirizzi logici sullo spazio degli indirizzi fisici evitando sovrapposizioni
- ad ogni processo deve essere assegnato un proprio spazio di indirizzi fisici separato da quello degli altri processi, cioè senza alcun indirizzo fisico in comune
- La trasformazione dallo spazio dei nomi simbolici allo spazio logico è effettuata dal compilatore

Il problema



L'approccio

- Il Memory Manager svolge questo compito (con il supporto dell'hw) espletando le seguenti funzioni:
  - Rilocazione o binding: operazione che definisce la corrispondenza tra elementi di due diversi spazi degli indirizzi
  - Protezione: un processo non può accedere (se non autorizzato) allo spazio degli indirizzi (fisico o logicio) di un altro processo
  - Sharing: alcuni spazi di memoria fisica possono essere condivisi tra processi
- Rilocazione e binding possono essere svolte anche dal compilatore

Formato sorgente e rilocabile

- Symbolic names → Logical addresses → Physical addressess
- Inizialmente, lo spazio di indirizzamento di un programma è costituito da nomi simbolici, ed in questo caso parliamo di formato sorgente del codice
- Lo spazio degli indirizzi simbolici viene trasformato in spazio logico dal compilatore/ linker, trasformando il formato del programma in rilocabile
  - Lo spazio logico è uguale per tutti i programmi e solitamente coincide in sistemi a 32 bit con lo spazio 0x00000000 – 0xFFFFFFFF

Formato assoluto

- Lo spazio logico deve poi essere mappato sullo spazio fisico, cioè sulle effettive locazioni di memoria che conterranno dati e istruzioni
- In questo caso parliamo di formato assoluto del codice

Binding sullo spazio fisico

- Se in fase di compilazione è già noto lo spazio di memoria fisica che dovrà contenere il processo
  - è possibile generare codice assoluto a compile time; qualora lo spazio fisico dovesse essere modificato sarà necessario ricompilare il codice
- In caso contrario il compilatore associa al programma uno spazio logico (rilocabile) che potrà essere mappato sullo spazio fisico:
  - Load time: durante il caricamento del programma da memoria di massa a memoria centrale
  - Execution time: durante la fase di esecuzione di un'istruzione si provvede a rilocare opportunamente gli indirizzi in essa contenuti. In questo caso è necessario un supporto hw per lo svolgimento dell'operazione

Binding time tradeoff

- Early binding (compilatore)
  - Esecuzione codice efficiente
  - Consente di anticipare in fase di compilazione una serie di verifiche
- Delayed binding (linker e Loader)
  - Esecuzione codice efficiente
  - Consente compilazioni separate
  - Facilita portabilità e condivisione del codice
- Late binding (VM, linker/loader dinamici, overlay)
  - Esecuzione codice meno efficiente
  - Check a runtime
  - Molto flessibile, consente riconfigurazioni a run time

Memory Management Unit (MMU)

- Dispositivo hardware che mappa indirizzi logici in indirizzi fisici a run-time (execution-time)
  - La trasformazione degli indirizzi viene effettuata ogni volta che un indirizzo viene prelevato dalla CPU dall'istruzione e inviato sul bus indirizzi
- La CPU elabora solo indirizzi logici che sono poi trasformati dalla MMU

Il problema

Programma 1

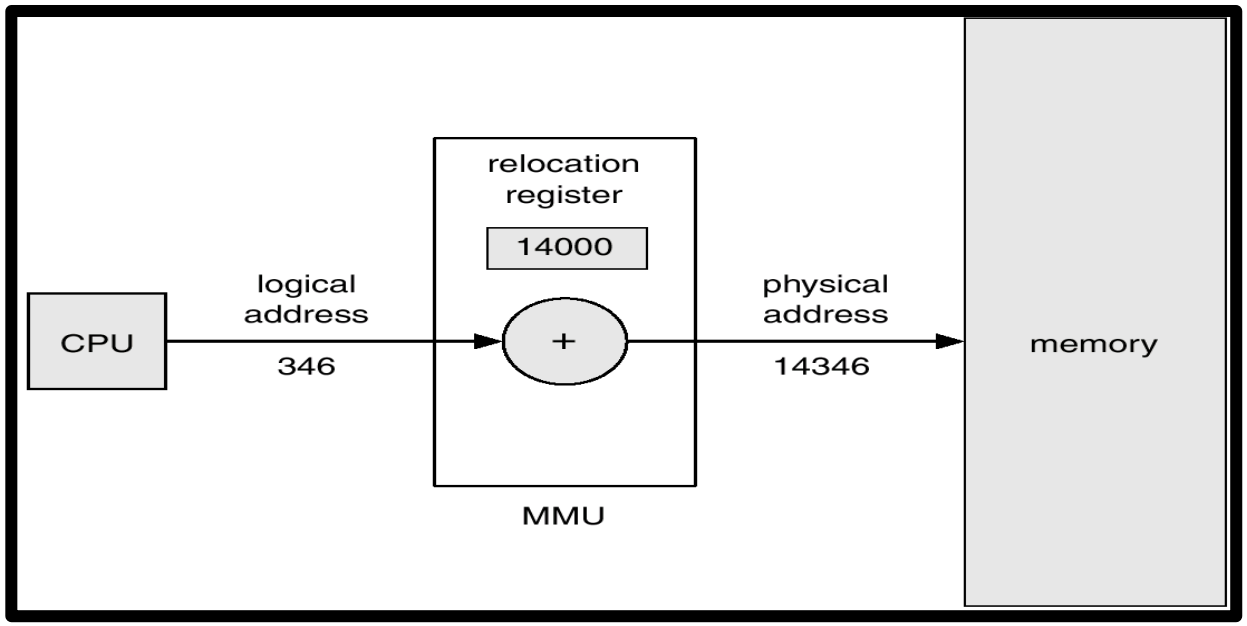
```
    movl   $0,%eax
    movl   $0,%ebx
sumloop:
    movl   1000(,%eax,4), %ecx
    addl   %ecx, %ebx
    incl   %eax
    cmpl   $19, %eax
    jle    sumloop
```

Programma 2

```
        movl   $0, %ebx
        movl   $0, %ebx
sumloop:
        movl   1000(,%eax,4),%ecx
        addl   $10, %ecx
        mov    %ecx,1000(,%eax,4)
        incl   %eax
        cmpl   $19, %eax
        jle sumloop
```

Rilocazione dinamica

Se voglio eseguire i programmi sopra indicati in multiprogrammazione posso ricorrere alla rilocazione dinamica, che agli spazi di indirizzamento logici dei due processi, associa porzioni disgiunte dello spazio fisico



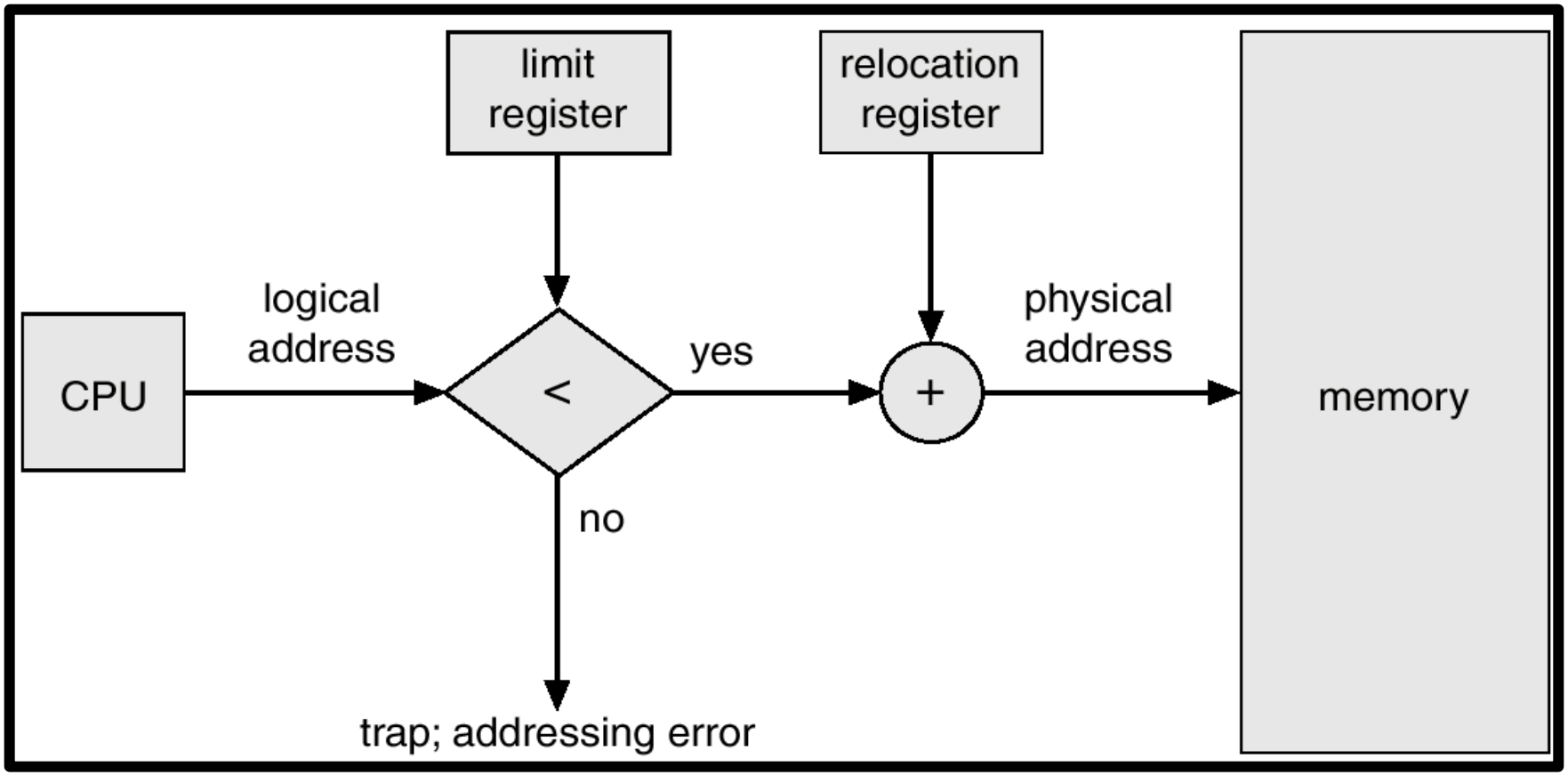
Protezione

- I processi non devono poter accedere ad aree di memoria di altri processi senza permesso

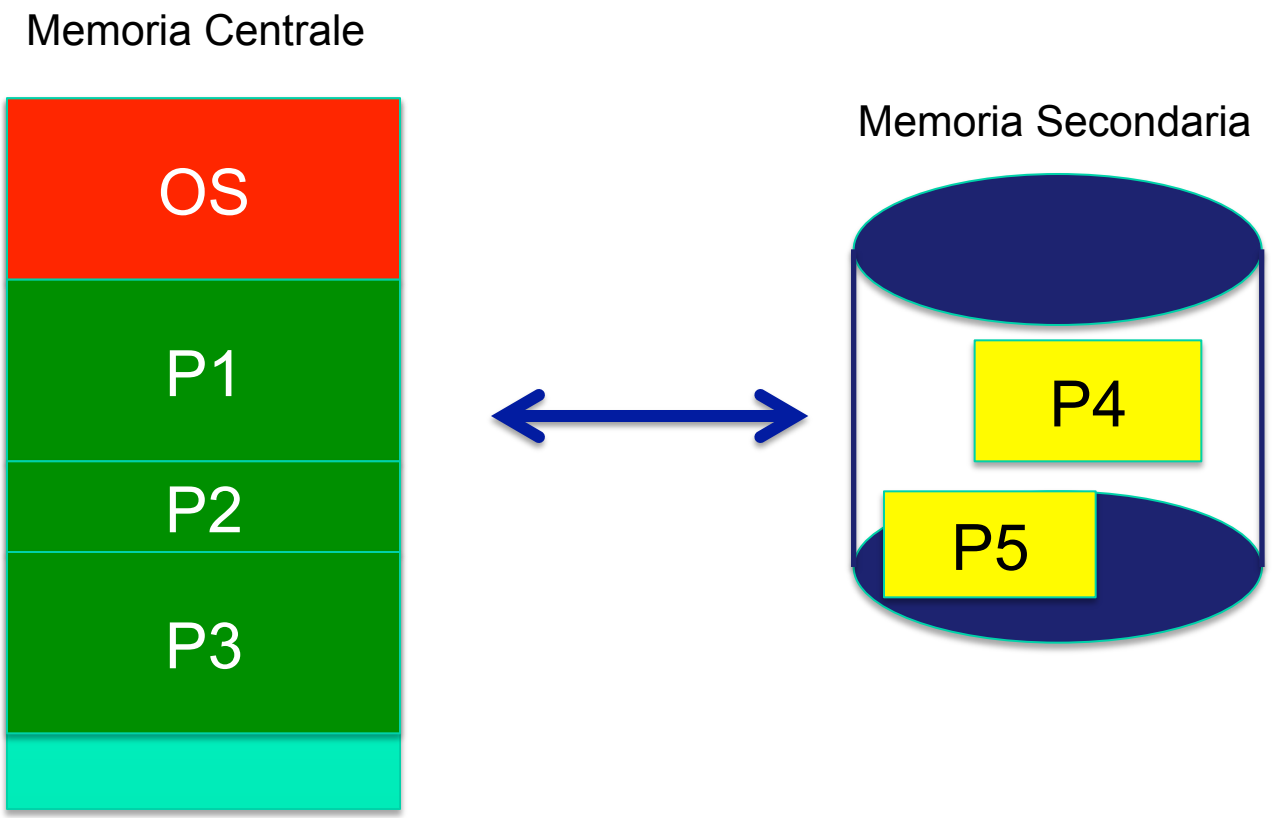
```
        mov end,%eax
loop:   mov 0,(%eax)
        add 4,%eax
        jmp loop
end:    ret
```

- Poiché è impossibile controllare a priori quali indirizzi di memoria un programma referenzierà i riferimenti alla memoria devono essere verificati durante l'esecuzione
- La protezione dei dati in memoria è una funzione che viene svolta dall'hardware

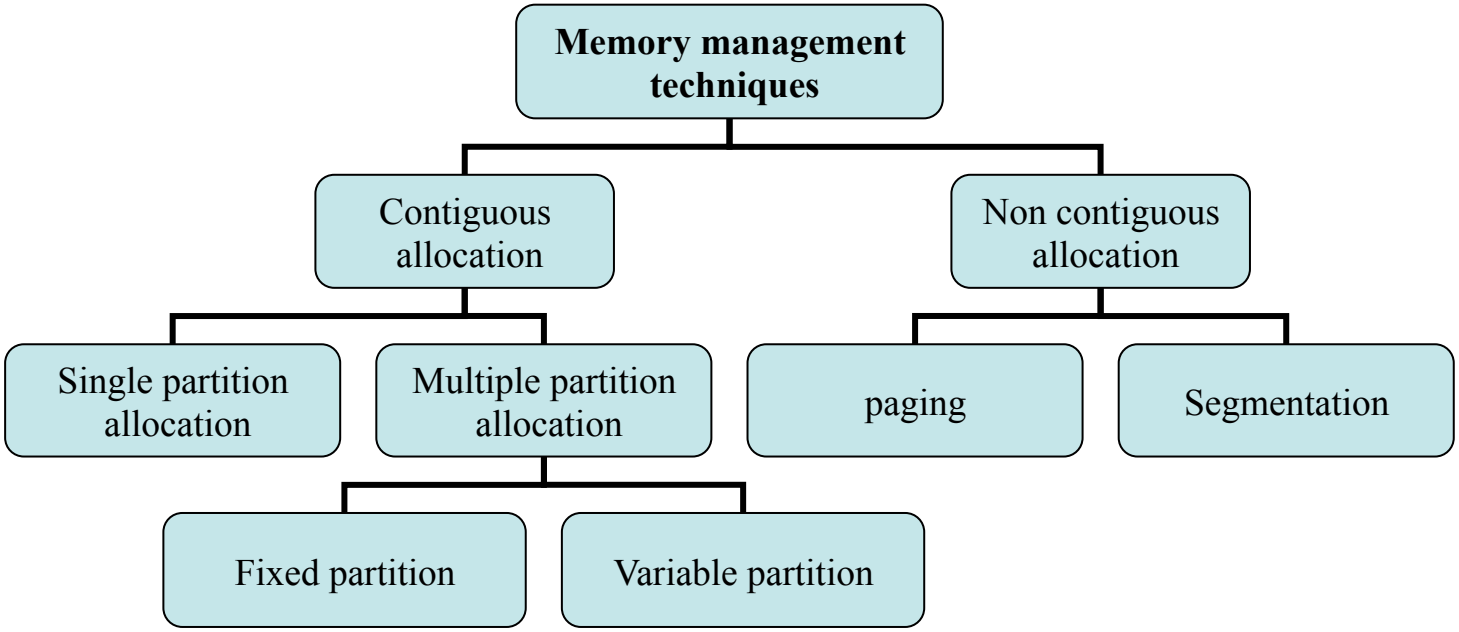
Protezione



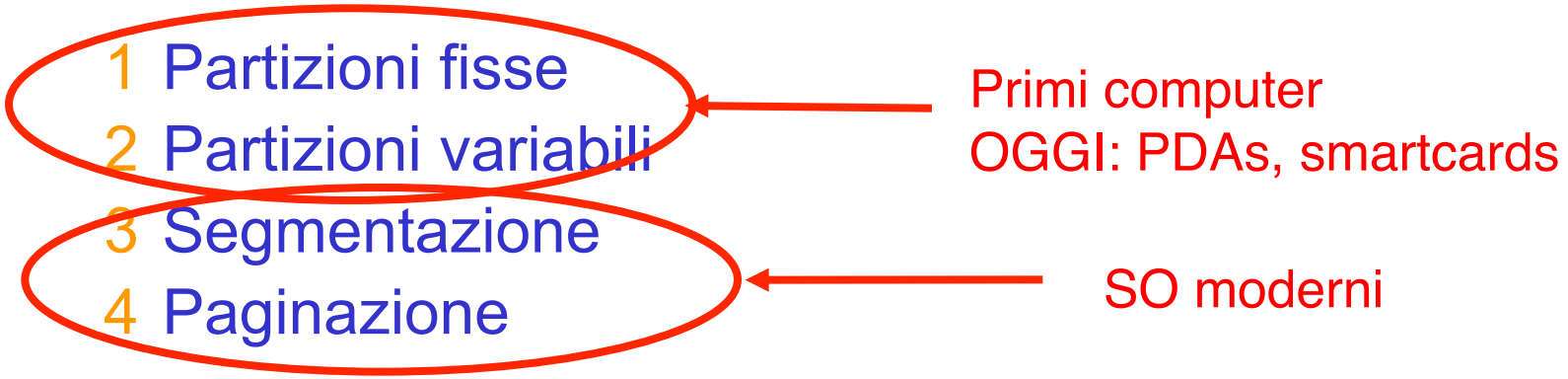
Swapping



Tecniche di allocazione della memoria

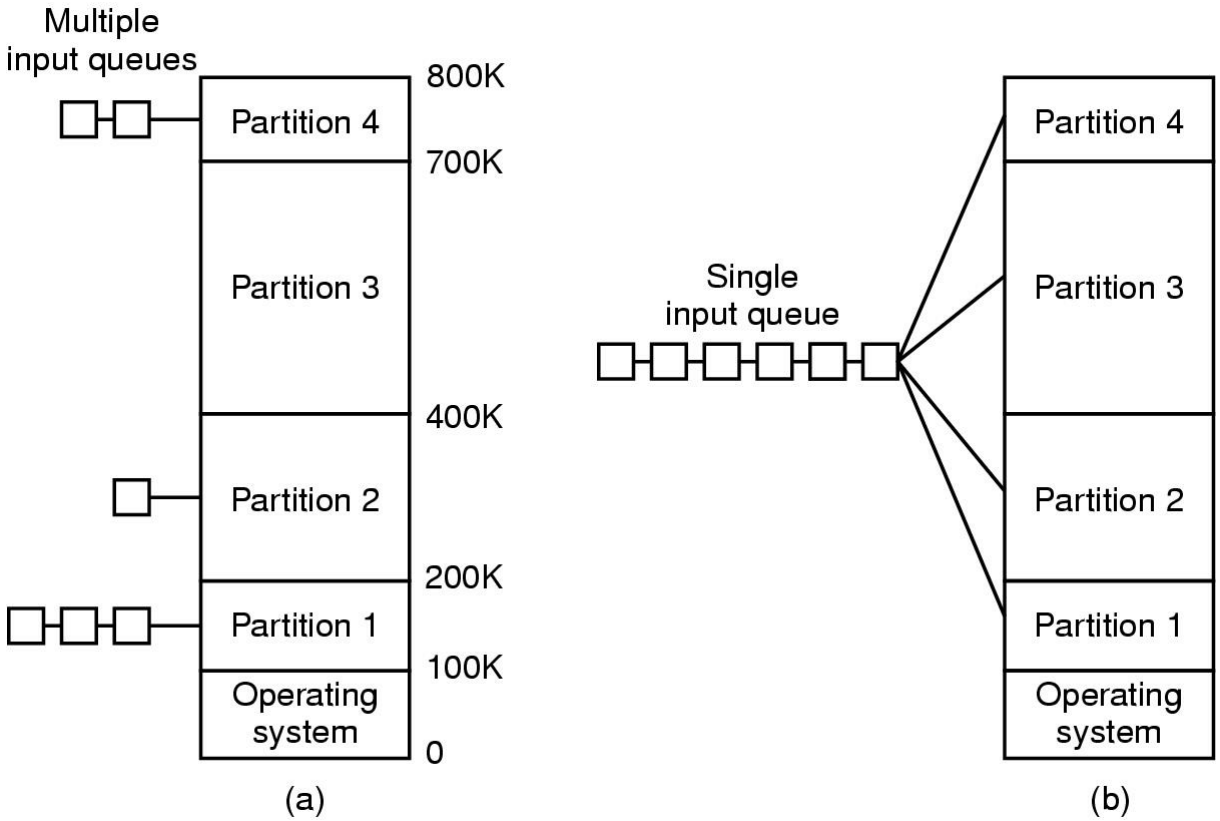


Gestione della memoria



- Criteri di riferimento
  - Efficienza implementativa
  - Sfruttamento della risorsa (spazio lasciato inutilizzato)

Partizioni multiple fisse

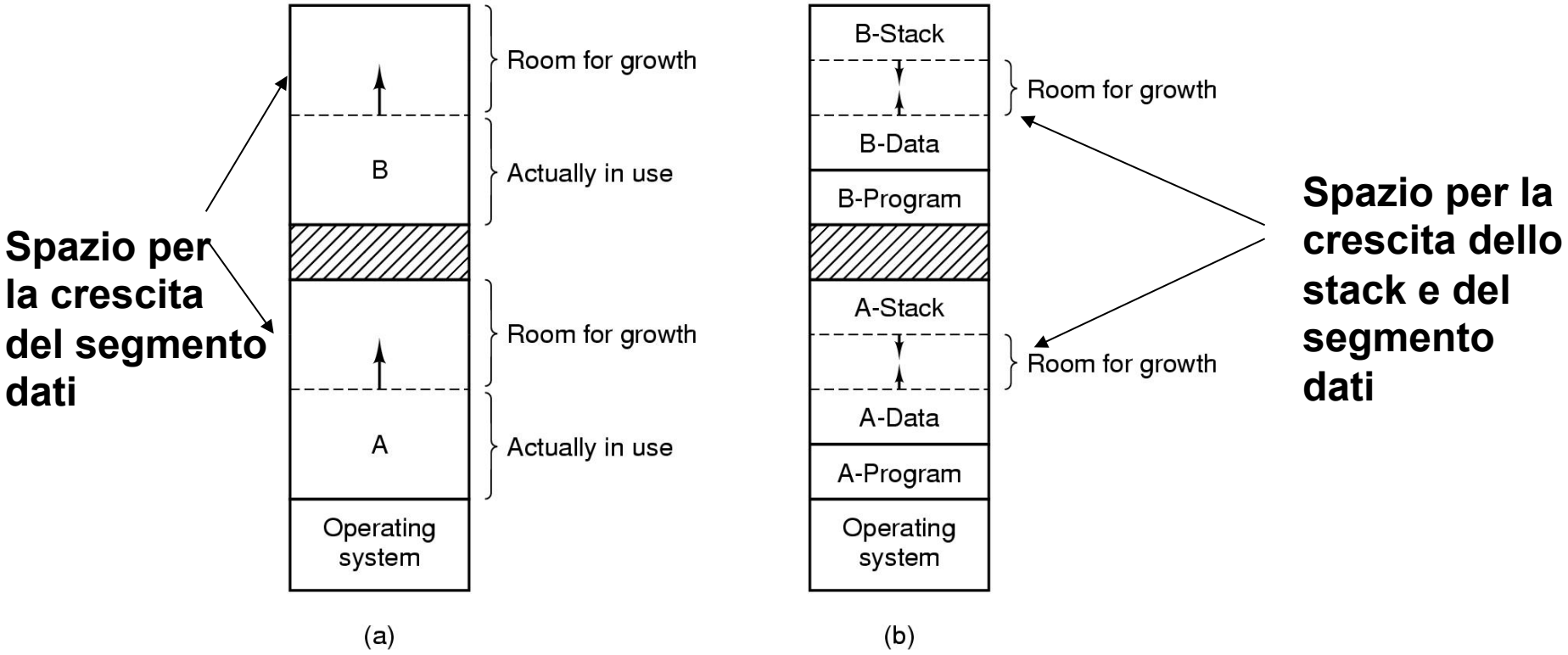


- Memoria a partizioni fisse
  - Code separate per ogni partizione
    - possibile sotto utilizzo del sistema
  - Singola coda di input

Partizioni multiple fisse: criticità

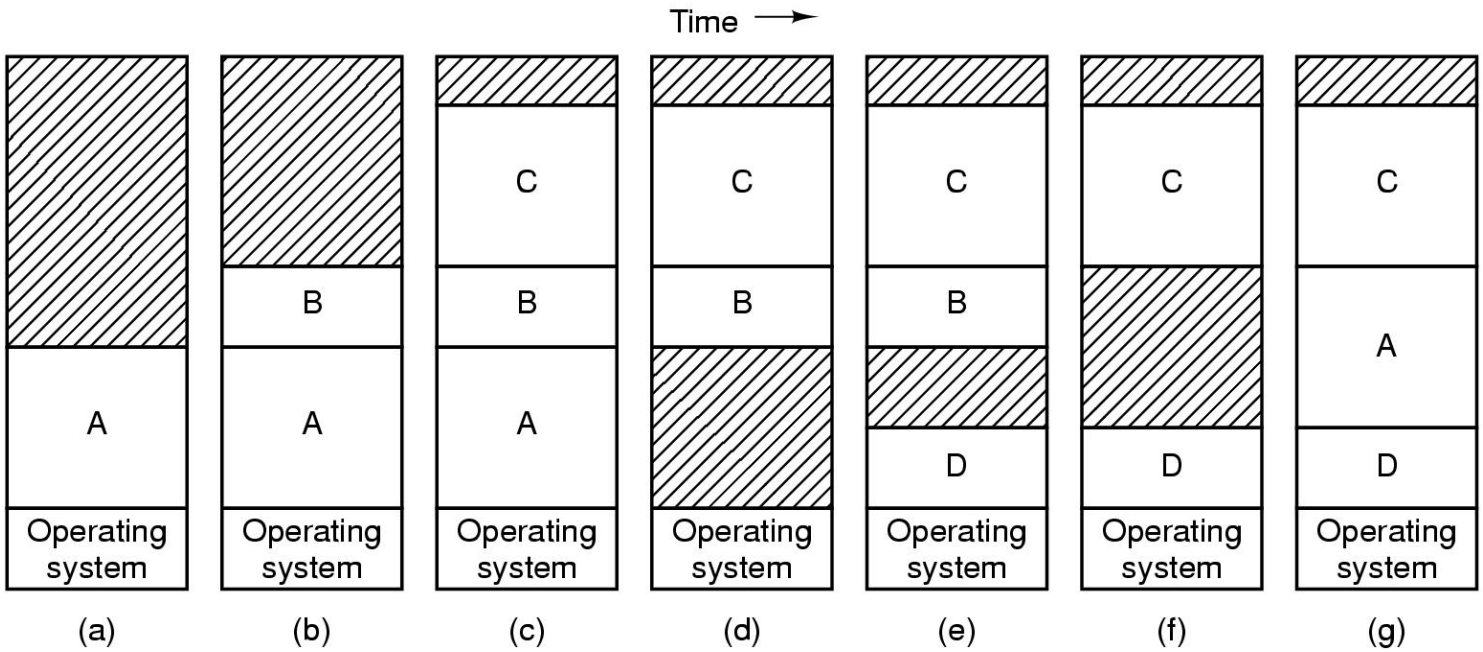
- Le partizioni fisse hanno avuto un certo successo in sistemi di tipo batch
  - Relativa staticità dei programmi eseguiti su un sistema
- Con l'avvento dei sistemi time sharing, i problemi principali da risolvere sono diventati:
  - dover gestire processi la cui dimensione, nel complesso, era di gran lunga superiore di quella della memoria centrale
  - Dover gestire processi la cui dimensione era fortemente variabile

Partizionamento guidato dai processi



- Le dimensioni di un processo cambiano durante l'esecuzione va quindi prevista la possibilità di espansione

Partizioni multiple variabili



- Numero, dimensione e posizione delle partizioni cambiano col variare dei processi in memoria
- Lo stato della memoria si modifica continuamente
  - I processi accedono alla memoria centrale
  - Lasciano la memoria per poi rientrarvi, non necessariamente nello stesso posto occupato in precedenza

Partizioni multiple variabili: criticità

- Gestione delle zone libere e occupate della memoria centrale
- Frammentazione esterna
  - Degli spazi liberi

Frammentazione esterna

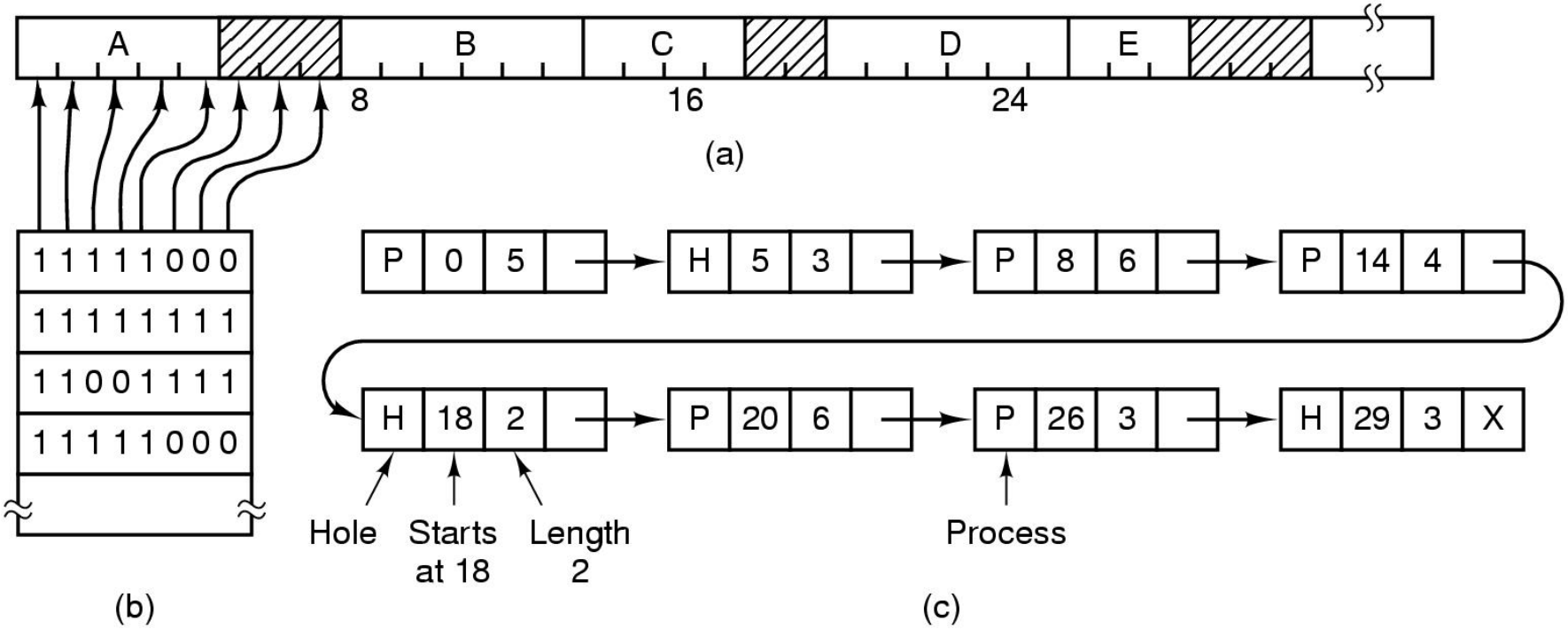
- Frazionamento della memoria libera in piccoli pezzi inutilizzabili
- Una possibile soluzione è l'adozione di strategie di compressione della memoria centrale
  - Operazione molto costosa in termini di efficienza del sistema
  - Non esistono strategie ottimali

GESTIONE MEMORIA PER ALLOCAZIONI CONTIGUE

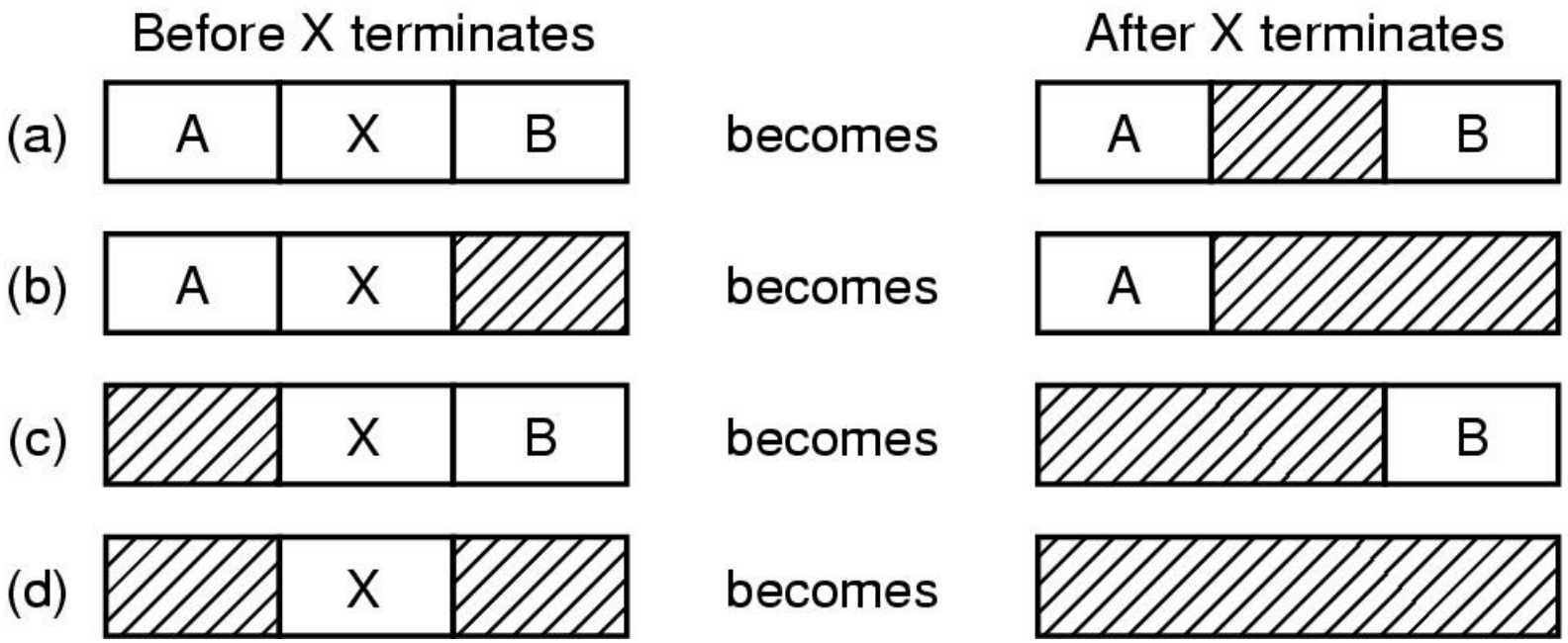
Gestione della memoria

- Si suddivide la memoria in unità minime allocabili di k byte bit
- A ciascuna unità è associato un bit
  - Libera/occupata
- Un array di dimensione h sarà allora sufficiente per descrivere lo stato di occupazione della memoria
  - h = Dimensione memoria in byte / k
- Ricerca di successioni di n bit a zero nella bitmap per trovare aree libere di dimensioni adeguate al programma da caricare
  - Semplice ma inefficiente

Gestione della Memoria



Liste per allocazione memoria



- Aree libere contigue vengono compattate
- Quando si usano le liste linkate, può essere più efficiente fare ricorso a liste doppiamente linkate

Strategie di allocazione spazi liberi

- First Fit: la prima porzione libera sufficientemente grande per contenere il processo
- Next fit: come first fit, ma riprende da dove sia era bloccato all'ultima richiesta esaudita
- Best Fit: la più piccola porzione libera sufficiente a contenere il processo
- Worst Fit: la più grande porzione libera che può contenere il processo

MEMORIA VIRTUALE

Introduzione

- La crescente dimensione (in termine di codice) delle applicazioni
- la diffusione di sistemi time-sharing con elevati numeri di utenti
- Imponevano la necessità di disporre di sistemi in grado di gestire in modo efficiente situazioni in cui la dimensione complessiva dei processi da eseguire era di gran lunga superiore alla memoria fisica
- Lo swapping era ritenuta una soluzione poco efficiente al problema

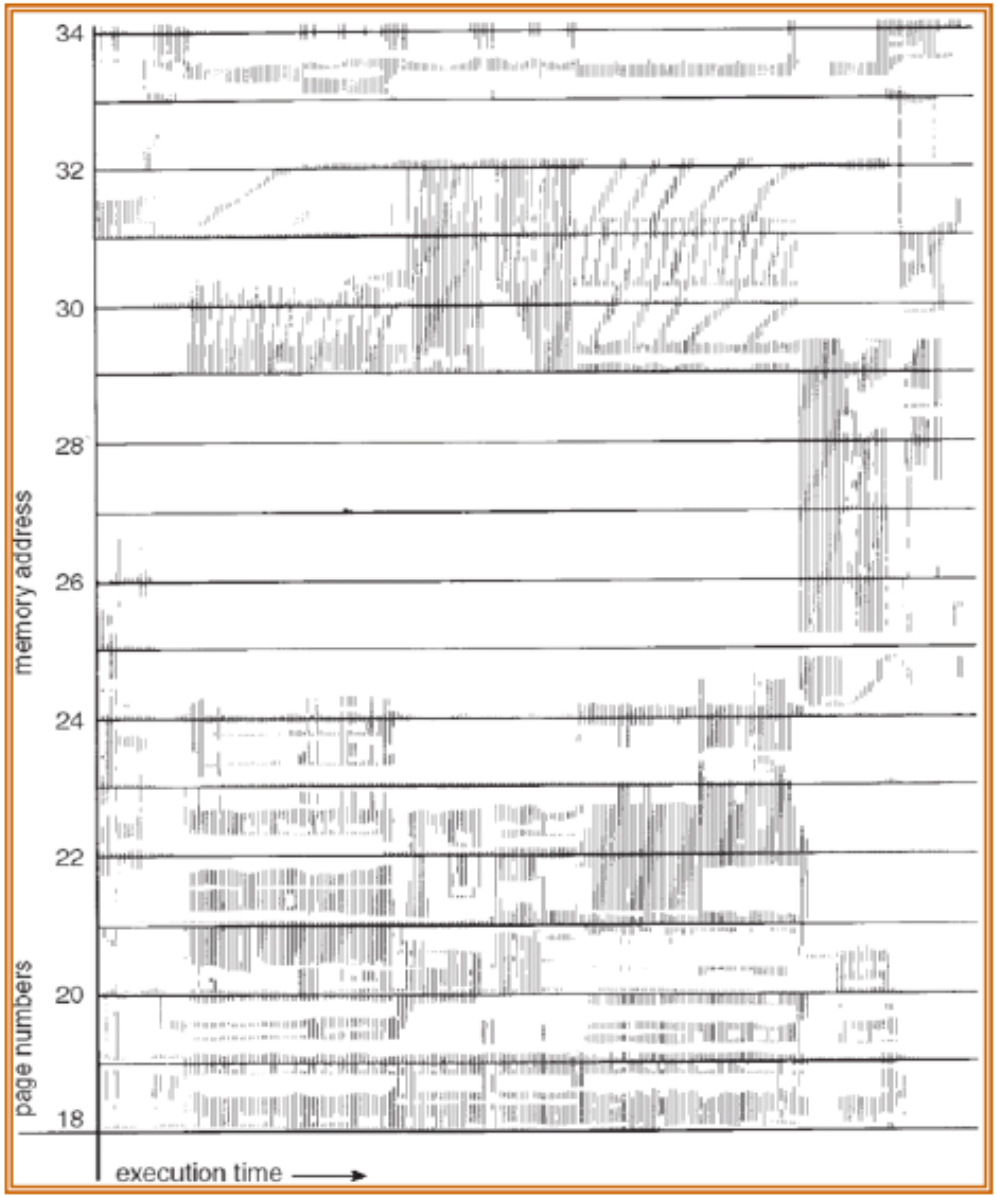
VM

- In risposta a queste esigenze viene prospettata nel 1961 una tecnica nota come Memoria Virtuale, che richiedeva però il superamento di due grossi vincoli:
  - La permanenze di un processo in memoria nella sua interezza;
  - La permanenza di un processo in locazioni di memoria contigue.

I Principi

- Sino alla metà degli anni ’70, un programma per poter essere eseguito veniva caricato interamente in memoria
- Ma ad un certo punto qualcuno ha incominciato a dubitare della necessità di questo requisito …

Località in spazio e tempo di un processo



Località

- Durante la loro esecuzione i programmi godono di due importanti proprietà:
  - Località temporale: se un elemento x viene referenziato all'istante t, la probabilità che x venga referenziato anche all'istante t+t' cresce al tendere di t'→0
  - Località spaziale: se un elemento x di posizione s viene referenziato all'istante t, la probabilità che venga referenziato un elemento x' di posizione s',

$$|s - s'| \leq \Delta$$

all'istante t+t' cresce al tendere di t'→0

Località

- I principi di località suggeriscono un importante accorgimento nella gestione della memoria:

  "*non è necessario caricare un programma interamente in memoria per poterlo eseguire, è sufficiente caricarlo località per località*"

Località: i vantaggi

- I vantaggi che deriverebbero dall'adozione del suddetto schema sono:
  - Svincolo della dimensione di un programma dalla dimensione della memoria centrale
  - Aumento del livello di programmazione, a parità di memoria centrale disponibile
  - Maggiore velocità nelle operazioni di swapping

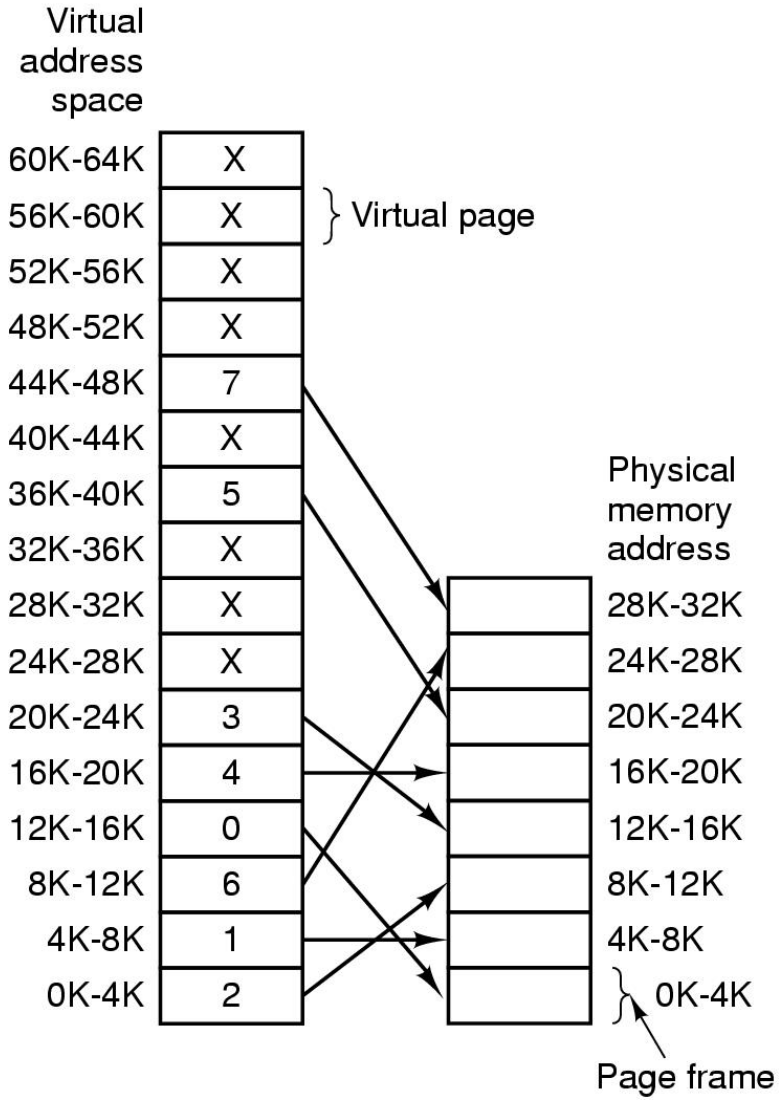
Overlay

- Meccanismo molto primitivo per realizzare la memoria virtuale
- Il programmatore suddivide il programma in porzioni la cui esecuzione non si sovrappone nel tempo
- Ogni porzione è chiamata overlay
- Gli overlay vengono caricati in istanti successivi su esplicito comando del programmatore ed eseguiti
- Ogni programma è costituito da almeno un overlay sempre residente

Paginazione

- Un meccanismo che consente di mappare automaticamente uno spazio di indirizzamento logico su porzioni diverse dello spazio di indirizzamento fisico
- Come?
  - Suddivido la memoria fisica e logica in blocchi di dimensioni uguali (4096 byte), chiamo i primi “frame” e gli altri “pagine”, carico un processo nei frame di memoria non assegnati ad altri processi

Demand Paging



Demand Paging

- Sistema completamente automatico per realizzare memoria virtuale
- Uno schema elementare può essere il seguente (lazy swapper)
  - Carico la prima pagina del programma da eseguire
  - Quando una nuova pagina viene referenziata la carico in memoria

Demand Paging

- La precedente strategia richiede però che sia possibile stabilire a priori la presenza o meno di una pagina in memoria
- Uso un bit aggiuntivo nella tabella delle pagine: bit di validità
- L'evento di pagina non trovata in memoria, è denominato page fault ed il suo verificarsi produce un'eccezione (#14), gestita da un apposito handler (page fault handler)
- Gestire il page fault significa preoccuparsi di recuperare la pagina da disco e portarla in memoria

Demand Paging

- In un sistema paginato il tempo medio di accesso a memoria peggiora e dipende da due fattori: il tempo di gestione del page fault e la frequenza con cui ciò si verifica
- se p è la probabilità che si verifichi un page fault:

$$\text{tempo medio di accesso} = \text{memory access} + p \cdot \text{tempo di page fault}$$

Paginazione

- In un sistema paginato gli indirizzi logici di memoria sono logicamente espressi da una coppia <p,d>, a cui va associato in fase di esecuzione un indirizzo fisico, che esprime la locazione esatta del dato o istruzione in memoria fisica

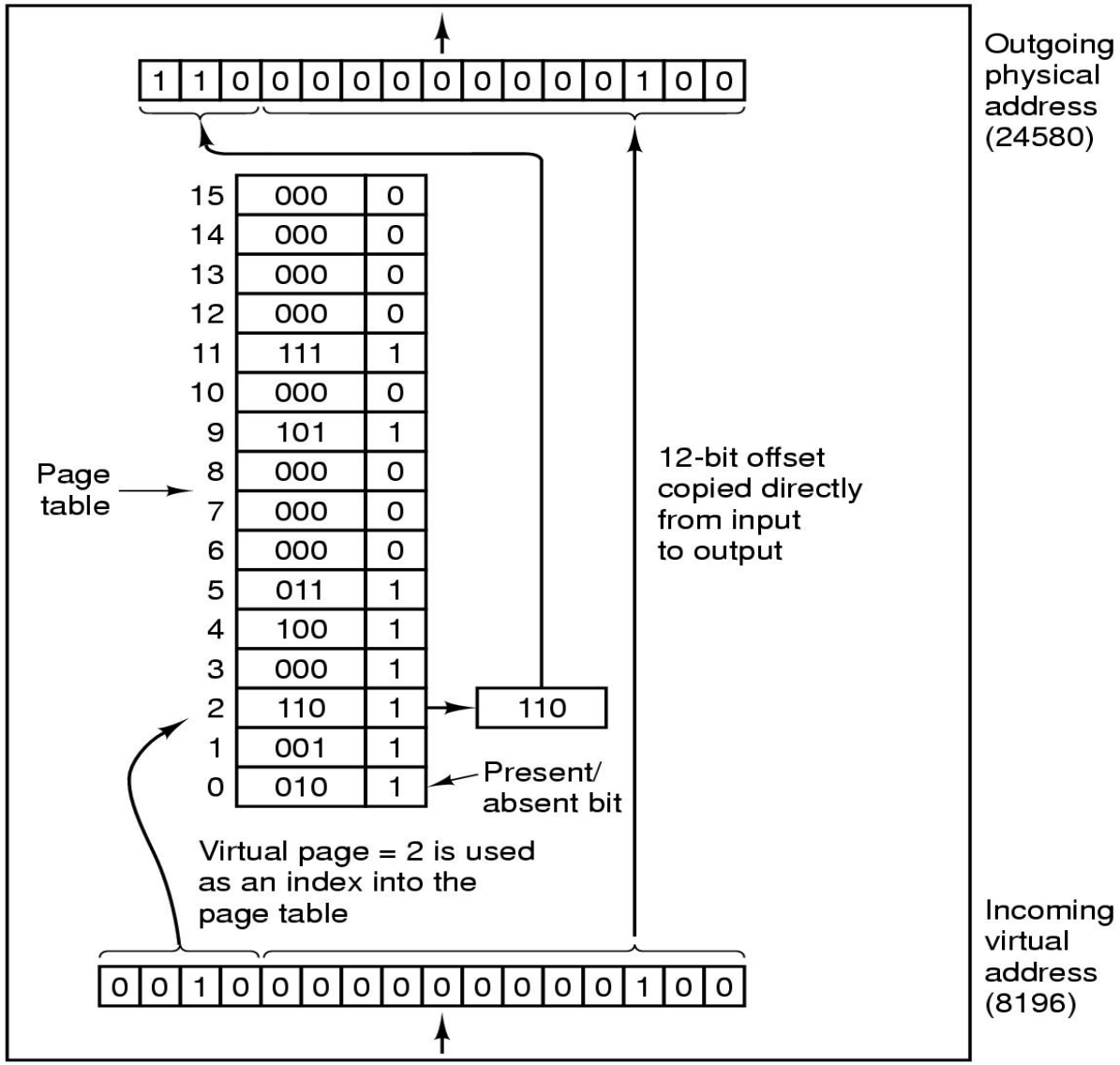
Calcolo degli indirizzi

- In un sistema paginato con pagine di 256 byte l'indirizzo 26251 sarà rappresentato dalla copia
  - p = 26251 DIV 256 = 102
  - d = 26252 MOD 256 = 139
- Per conoscere l'indirizzo fisico associato all'indirizzo logico <102, 139> dobbiamo conoscere a quale frame è stata assegnata la pagina 102
  - Indirizzo fisico = n.ro frame* 256 + d

Paginazione

- Con pagine di dimensione 2^k, per il calcolo degli indirizzi fisici si sfrutta l'indirizzo logico
  - I k bit meno significativi dell' indirizzo logico individuano lo spiazzamento all' interno della pagina
  - I restanti bit più significativi individuano il numero della pagina
  - Il numero di pagina si usa come indice nella tabella delle pagine
  - Il valore trovato in corrispondenza è il numero di page frame associato alla pagina
  - Se la pagina è presente in memoria, il numero di page frame è copiato nei bit alti del registro di output e concatenato ai k bit bassi dell' offset
- ES: 26251: 000001100110 10001011

Tabella delle pagine



Esercizio

Si consideri una memoria centrale formata da 16 pagine di 1Kbyte ciascuna. Qual è l'indirizzo fisico della locazione di indirizzo 1030 di un programma P, supponendo che la pagina 0 di P sia mappata nella XVI-ma pagina fisica, e la pagina 1 di P nella pagina fisica 9?

1. 1001000 0000110
2. 0001000 0000110
3. 0000000 0000110
4. 0110000 0001001