

## 9 - Gestione della memoria

domenica 1 maggio 2022 10:02

- Uno dei compiti più importanti del SO è la gestione della memoria, che comprende la **condivisione della memoria da parte di più processi**

- I processi accedono alla memoria tramite **decodifica degli indirizzi**: un indirizzo logico (che può essere anche solo una variabile) viene tradotto in un indirizzo fisico (un indirizzo vero e proprio a una parte della memoria)

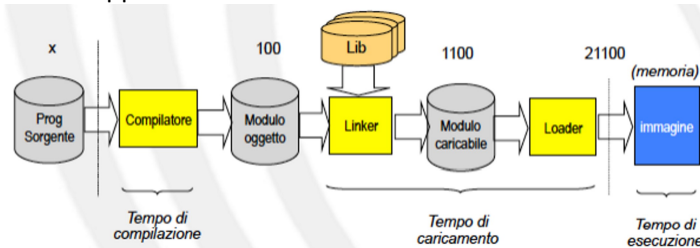
- **IMPORTANTE**: ogni programma deve essere portato in memoria ed essere trasformato in processo per essere eseguito

- il programma diventa un processo SOLO quando raggiunge la memoria

- al termine del processo, la memoria viene rilasciato

- La trasformazione da programma a processo avviene in diverse fasi

- ad ogni fase gli indirizzi hanno una rappresentazione diversa



### BINDING

**Binding**: traduzione tra indirizzi simbolici e indirizzi fisici

Il binding può avvenire in 3 momenti distinti:

- **a tempo di compilazione:**

- deve essere nota la locazione del programma: gli indirizzi fisici sono scritti direttamente nel codice

- **statico**: indirizzi fisici e logici sono uguali

- **a tempo di caricamento:**

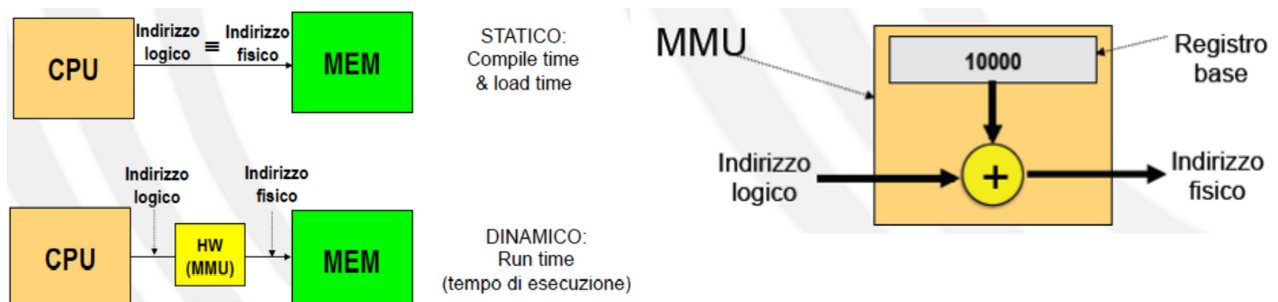
- si specifica un indirizzo fisico come primo indirizzo e gli altri sono scritti in relazione a quest'ultimo tramite offset

- **statico**

- **a tempo di esecuzione:**

- non ci sono indirizzi fisici perché la locazione può cambiare costantemente durante l'esecuzione

- **dinamico**: richiede HW apposito (MMU)



### COLLEGAMENTO (LINKING) E CARICAMENTO (LOADING)

Le operazioni di linking e loading possono avvenire in modo **statico o dinamico**:

- **Linking:**

- **statico**: viene creata una copia delle librerie usate

- **dinamico**: vengono creati dei riferimenti alle librerie (si risparmia codice)

- **Loading:**

- **statico**: tutto il codice viene caricato in memoria

- **dinamico**: viene caricato in memoria solo il codice che viene effettivamente utilizzato

## SCHEMI DI GESTIONE DELLA MEMORIA

- 4 schemi:
  - Allocazione contigua
  - Paginazione
  - Segmentazione
  - Segmentazione paginata

**NOTA:** si assume che tutto il programma sia stato caricato in memoria

### SCHEMA 1 - ALLOCAZIONE CONTIGUA

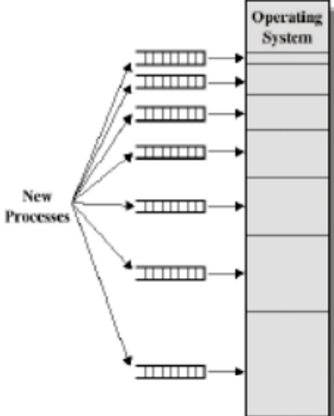
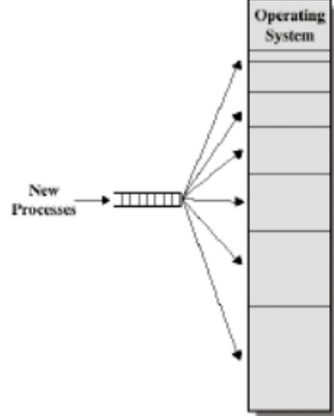
- La memoria è divisa in **partizioni**
  - partizioni fisse
  - partizioni variabili
- I processi vengono allocati in **posizioni contigue di una partizione**

#### TECNICA DELLE PARTIZIONI FISSE

- La RAM è divisa in partizioni di **diverse dimensioni**
  - i processi potrebbero richiedere più o meno spazio
  - es. se la ram ha 20 partizioni allora può contenere max 20 processi

#### ASSEGNAZIONE DELLA MEMORIA

- Viene effettuata dallo scheduling a lungo termine

Metodo 1 - Una coda per ogni partizione	Metodo 2 - Coda singola
<ul style="list-style-type: none"><li>•Il processo viene assegnato alla partizione più piccola in grado di contenerlo</li><li>•Possono esserci partizioni vuote</li></ul>	<p>Coda gestita con un'unica politica:</p> <ul style="list-style-type: none"><li>•<b>FCFS</b>: viene ammesso il primo programma che arriva nella coda</li><li>•<b>Analisi della coda</b>:<ul style="list-style-type: none"><li>-<b>Best-fit-only</b>: viene ammesso il processo con le dimensioni più simili alla partizione</li><li>-<b>First-avaible-fit</b>: viene ammesso il primo processo che può stare nella partizione</li></ul></li></ul>
	

- Vantaggi**:
  - molto semplice
- Svantaggi**:
  - problemi di **frammentazione**: spreco di memoria
  - frammentazione interna**: il processo viene messo in una partizione troppo grande sprecando memoria
  - frammentazione esterna**: si creano dei buchi (memoria non utilizzata) troppo piccoli per farci entrare dei processi

## TECNICA DELLE PARTIZIONI VARIABILI

- Il SO crea partizioni di dimensioni identiche a quelle dei processi
  - elimina la frammentazione interna
- Quando arriva un processo, **viene messo nella prima buca che può contenerlo**

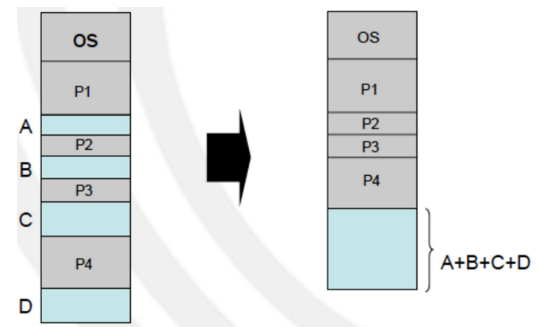
OS	OS	OS	OS	OS
Processo 5	Processo 5	Processo 5	Processo 5	Processo 5
Processo 8		Processo 9	Processo 9	
Processo 2	Processo 2	Processo 2	Processo 2	Processo 2
			Processo 10	Processo 10

## ASSEGNAZIONE DELLA MEMORIA

- Vengono usate 3 strategie:
  - **First-fit:** processo allocato nella prima buca grande a sufficienza (il migliore)
  - **Best-fit:** processo allocato nella buca più piccola che può contenerlo (troppo lento)
  - **Worst-fit:** processo allocato nella buca più grande trovata (troppo lento e spreca memoria)
- **Vantaggi:**
  - risolve il problema della frammentazione interna
- **Svantaggi:**
  - persiste il problema della frammentazione esterna

Il problema della frammentazione esterna può essere risolto tramite **compattazione**

- Il contenuto della memoria viene spostato in modo da rendere contigue le partizioni
- Fattibile solo se la rilocalizzazione è dinamica
- Può risultare costoso

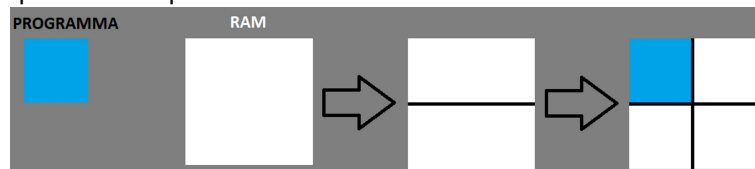


## TECNICA DEL BUDDY SYSTEM

- Via di mezzo tra partizioni fisse e variabili
- Si può allocare tutta la memoria
  - La memoria è vista come un insieme di blocchi di  $2^k$  dimensioni

### ESEMPIO

• Sostanzialmente si parte con tutta la memoria libera disponibile. La memoria verrà divisa in 2 parti uguali tante volte quanto serve pur di ottenere la Best-Fit



- **Vantaggi:**
  - molto veloce
- **Svantaggi:**
  - rimane il problema della frammentazione interna (seppur attenuata)

## SCHEMA 2 - PAGINAZIONE

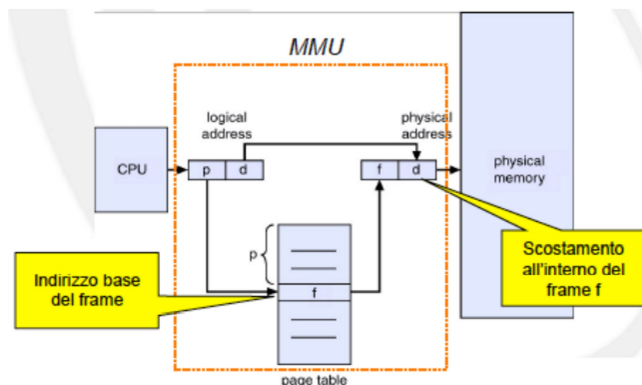
- Rimuove il problema della frammentazione esterna
- Permette l'allocazione dei processi **non contigua**
  - la memoria viene divisa in **frame**
- I programmi sono divisi in n **pagine** (a cui corrisponderanno n frame)
- Si utilizza una tabella delle pagine (**page table**) per mantenere traccia di **quale frame corrisponde a quale pagina**
  - viene salvata in memoria
  - non viene implementata nei registri perché non basterebbero
  - una tabella per ogni processo
  - serve a tradurre l'indirizzo logico in fisico

### Paginazione - esempio

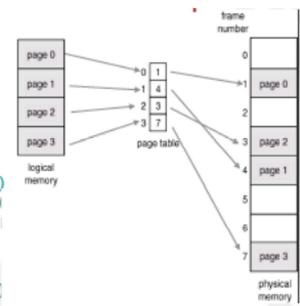
- Dimensione della pagina = 1KB
- Dimensione del programma = 2.3KB
- Necessarie 3 pagine
  - dell'ultima pagina si userà solo 0.3KB
- E' ancora possibile della frammentazione interna, ma solo nell'ultima pagina

## TRADUZIONE DELL'INDIRIZZO

- L'indirizzo generato dalla CPU viene diviso in 2 parti:
  - **numero di pagina (p)**: ci indica la pagina in sè
  - **offset (d)**: ci indica i bit che dobbiamo leggere di quella pagina



- Memoria da 8KB
- Pagine di 1KB
  - L'indirizzo logico 936 diventa l'indirizzo fisico 1960
    - pagina 0 (0...1023)
      - (inizio pagina = 0)
      - offset = 936
    - pagina 0 → frame 1 (1024...2047)
    - indirizzo fisico = 1024 + offset = 1024 + 936 = 1960
  - L'indirizzo logico 2049 diventa l'indirizzo fisico 3073
    - pagina 2 (2048...3071)
      - inizio pagina = 2048
      - offset = 1
    - pagina 2 → frame 3 (3072...4095)
    - indirizzo fisico = 3072 + 1 = 3073



## IMPLEMENTAZIONE IN MEMORIA

- La tabella risiede in memoria
- Vengono usati 2 registri:
  - **Page-table base register (PTBR)**: punta alla tabella delle pagine
  - **Page-table length register (PTLR)**: contiene la dimensione della tabella delle pagine (opzionale)

**Problema:** la paginazione in memoria richiede 2 accessi in memoria (uno per leggere la tabella e uno per l'indirizzo fisico)

**Soluzione:** si usa una cache veloce detta **translation look-aside buffers (TLB)**

- contiene le associazioni pagina-frame
- risulta comunque molto costosa
- memorizza solo un piccolo sottoinsieme delle entry nella tabella delle pagine

## PROTEZIONE

Al fine di evitare errori, vengono usati dei **bit di flag** in ogni frame:

- **bit di validità:** spesso un processo non usa tutti gli indirizzi logici e alcune pagine rimangono vuote. Questo bit segnala tali pagine in modo da bloccarne i tentativi di accesso
- **bit di accesso:** marca se una pagina è modificabile, eseguibile o read-only

## PAGINE CONDIVISE

- Usate per processi che **usano le stesse pagine**
  - **un'unica copia fisica ma più copie logiche** (una per processo)
  - es. se apro 3 processi di Word, 90% del codice viene caricato dalla stessa pagina

## PAGINAZIONE MULTILIVELLO

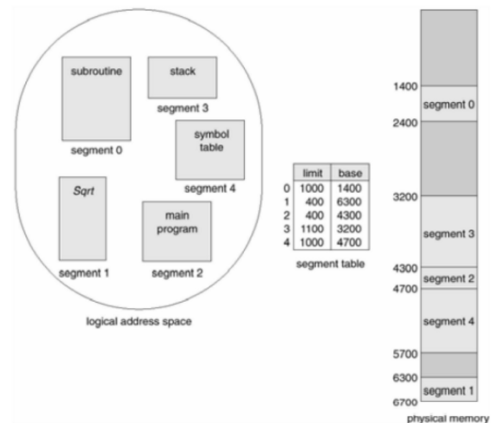
- Salvare tutte le pagine in un'unica tabella può non essere possibile: si usa la **paginazione multilivello**
  - sostanzialmente **le tabelle portano ad altre tabelle**
  - in pratica, solo alcune parti delle tabelle delle pagine sono memorizzate in memoria, le altre sono su disco
  - può portare a problemi di velocità

## TABELLA DELLE PAGINE INVERTITA

- Invece di usare una tabella per ogni processo, si usa **un'unica tabella per tutto il sistema**
  - contiene una entry per ogni frame
  - le associazioni sono invertite: frame → pagina
- Ogni entry contiene la coppia **<process-id, page-number>**:
  - **process-id**: id del processo che possiede la pagina
  - **page-number**: indirizzo logico della pagina contenuta nel frame
- Ogni indirizzo logico a sua volta è formato da una tripla coppia: **<process-id, page-number, offset>**
- L'unico svantaggio è che persiste il problema della frammentazione esterna

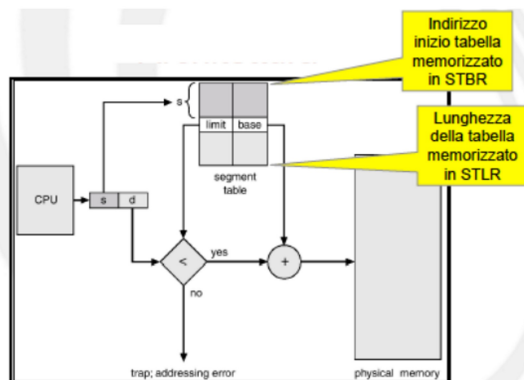
## SCHEMA 3 - SEGMENTAZIONE

- Il programma è diviso in **segmenti di dimensione variabile**
  - a crearli è il compilatore
- Un segmento può essere un'unità logica come:
  - Main
  - Procedure
  - Funzioni
  - variabili locali e globali
  - Stack
  - symbol table
  - vettori



- L'indirizzo logico è formato da **<numero di segmento, offset>**
- La tabella dei segmenti (**segment table**) risiede in memoria
- Vengono usati 2 registri:
  - **Segment-table base register (STBR)**: punta alla tabella dei segmenti
  - **Segment-table length register (STLR)**: contiene il numero di segmenti usati (**non opzionale**)

### Traduzione indirizzi nella Segmentazione



### Traduzione indirizzi - esempio

Segmento	Limite	Base
0	600	219
1	14	2300
2	100	90
3	580	1327

- Indirizzi logici
  - **<0, 430>**
    - segmento 0, offset 430
    - $430 < 600$ ? OK
    - indirizzo fisico =  $430 + \text{base di } 0 = 430 + 219 = 649$
  - **<1, 20>**
    - segmento 1, offset 20
    - $20 < 14$  NO
    - !indirizzo non valido!

- Come nella paginazione, sono supportati la **protezione** (bit di validità e accesso) e la **condivisione**
- È presente la **cache TLB**

**Problema:** la segmentazione soffre di **frammentazione esterna** dovuta a 2 cause:

- I segmenti hanno lunghezza variabile
- Il SO deve allocare tutti i segmenti

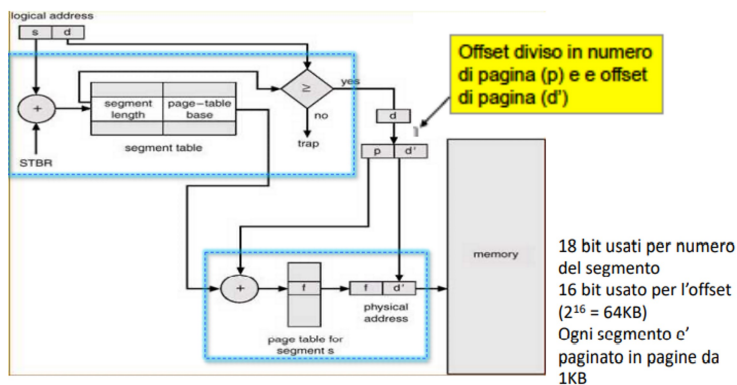
## PAGINAZIONE E SEGMENTAZIONE A CONFRONTO

	PAGINAZIONE	SEGMENTAZIONE
Programma	diviso in pagine	diviso in segmenti
Tabella	Page table	Segment table
Indirizzo logico	<numero di pagina, offset>	<numero di segmento, offset>
Registri	PTBR PTLR (opzionale)	STBR STLR (non opzionale)
Cache TLB	Sì	Sì
Protezione	Sì	Sì
Condivisione	Sì	Sì
Vantaggi	<ul style="list-style-type: none"> <li>• Non esiste frammentazione (minima interna)</li> <li>• Allocare i frame non richiede algoritmi specifici</li> </ul>	<ul style="list-style-type: none"> <li>• Consistenza tra vista utente e vista fisica della memoria</li> </ul>
Svantaggi	<ul style="list-style-type: none"> <li>• Separazione tra vista utente e vista fisica della memoria</li> </ul>	<ul style="list-style-type: none"> <li>• Potenziale frammentazione esterna</li> <li>• Richiede l'allocazione dinamica dei segmenti</li> </ul>

## SCHEMA 4 - PAGINAZIONE SEGMENTATA

- Unione tra la paginazione e la segmentazione
- **Ogni segmento ha delle pagine**
- Ogni segmento ha la sua tabella delle pagine
- La tabella dei segmenti non contiene l'indirizzo di ogni segmento, ma l'indirizzo delle tabelle delle pagine per ogni segmento
- **Risolve il problema dell'allocazione dei segmenti e della frammentazione esterna**

### Esempio (MULTICS)



### Traduzione indirizzi MULTICS

- In realtà, la ST è troppo grande ( $2^{18}$  = fino a più di 256 mila segmenti...), per cui viene paginata anch'essa, scomponendo il segment number in due parti...

