LEZIONE 9

**La gestione della Memoria** **Introduzione**

* Per essere eseguito, un programma deve essere caricato in memoria e inserito all’interno di un processo ù
* Coda d’ingresso (input queue): l’insieme dei processi presenti nei dischi che attendono d’essere trasferiti nella memoria per essere eseguiti
* Prima di essere eseguito, un programma passa attraverso numerose fasi

**Associazione di istruzioni e dati a indirizzi di memoria**  Generalmente, l’associazione di istruzioni e dati a indirizzi di memoria si può compiere in qualsiasi fase del seguente percorso:

* **Compilazione:** si ha un certo programma, e si vuole istanziare tutti i sui indirizzi; si compila e si ottiene un certo codice che poi deve essere eseguito. Quindi se si ottengono in questa fase tutti i codici definitivi/assoluti **(absolute code)**, si ha un vantaggio, ovvero, avendoli già generati tutti in questa fase, non bisogna farlo più; oltre ciò c’è anche un grande svantaggio, ovvero, l’oggetto così generato, una volta caricato in memoria centrale, non potrà essere più spostato, in quanto dovrà essere incastrato in memoria a indirizzi compatibili, che sono stati già definiti nella fase di compilazione
* **Caricamento:** se nella fase di compilazione non è possibile sapere in che punto della memoria risiederà il processo, il compilatore deve generare codice rilocabile **(relocable code)**. Rispetto alla compilazione, questo metodo ah un vantaggio, in quanto prima che il programma venga mandato in memoria centrale, c’è ancora tempo per trovare una locazione migliore, quindi di reindirizzarlo
* **Esecuzione:** Si carica il programma in memoria centrale, ma mantengo ancora in qualche modo un livello astratto/ simboli astratti, che mi permettono di istanziare nel momento dell’esecuzione; ma solo in quell’istante si va a sostituire queste variabili con un indirizzo fisico reale. Il vantaggio è enorme in quanto pur essendo stato caricato in memoria, **ma non eseguito,** possono capitare diverse situazioni come: si ha la possibilità di fare spazio ad un altro processo, oppure il processo ha aumentato le richieste di risorse; cioè possono decidere **dinamicamente** **di avere maggior o minor spazio**. **Quindi fin quando non si esegue l’esecuzione, si può mantenere un livello d’astrazione che permette di far scivolare su e giù processi che si trovano nella RAM e questo è un bel vantaggio**

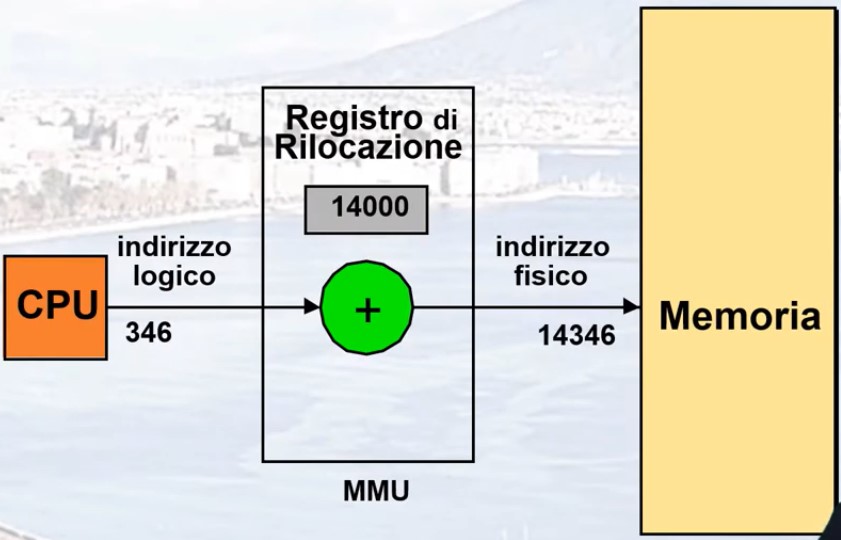
Spazio di indirizzi logici e fisici

* Il concetto di spazio di indirizzi logici associato a uno spazio degli indirizzi fisici separato è fondamentale per una corretta gestione della memoria
* **Indirizzo logico (logical address):** indirizzo generato dalla CPU; detto anche **indirizzi virtuale (virtual address)**
* **Indirizzo fisico** **(physical address):** indirizzo visto dall’unità di memoria
* I metodi di associazione degli indirizzi nelle fasi di compilazione e di caricamento producono indirizzi logici e fisici identici. Con i metodi di associazione nella fase d’esecuzione, invece, gli indirizzi logici non coincidono con gli indirizzi fisici

Unità di gestione della memoria (MMU)

* La MMU è un dispositivo per l’associazione, nella fase di esecuzione, degli indirizzi virtuali agli indirizzi fisici
* Nello schema MMU, quando un processo utente genera un indirizzo, prima dell’invio all’unità di memoria, **si somma a tale indirizzo il valore contenuto nel registro di rilocazione**
* **Il programma utente tratta con gli indirizzi logici; non considera mai gli indirizzi reali**

**Rilocazione dinamica tramite un registro di rilocazione**

****

**Spiegazione**

A sinistra c’è una CPU che gestisce un indirizzo logico 346(indirizzo astratto; è quello che vede l’utente). Poi c’è un indirizzo base contenuto nel registro di rilocazione (in questo caso 14000). La MMU non fa altro che sommare l’indirizzo logico (346) con tale base (14000), ottenendo cosi l’indirizzo reale che si troverà nella memoria, ovvero 14346.

Caricamento dinamico

* **Una procedura non viene caricata fino a quando non è richiamata (cioè carico solo ciò che deve essere realmente eseguito)**
* Migliore utilizzo dello spazio di memoria: una procedura che non si adopera non viene caricata
* Utile quando servono grandi quantità di codice per gestire casi non frequenti (es. procedure di gestione degli errori)
* Non richiede un intervento particolare da parte del SO

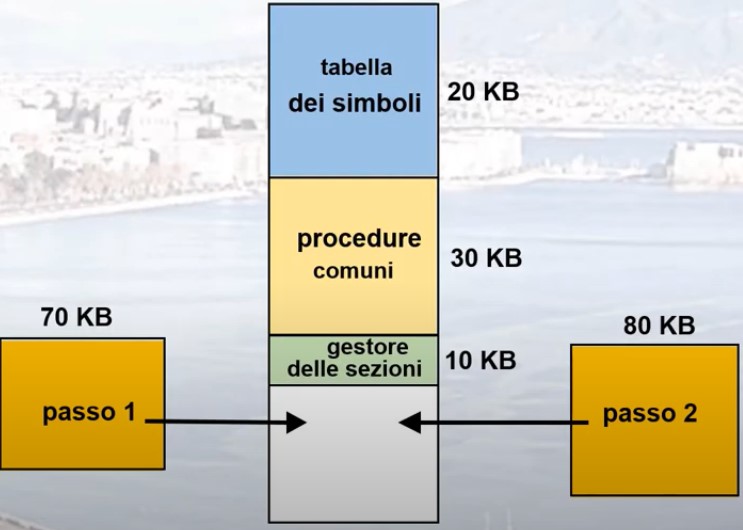
Collegamento dinamico

* **Il collegamento viene differito fino al momento dell’esecuzione**
* Una piccola porzione di codice di riferimento (stub) indica come localizzare la giusta procedura di libreria residente nella memoria
* Il codice (stub) sostituisce se stesso con l’indirizzo della procedura, che viene poi eseguita
* Richiede generalmente l’assistenza del SO per controllare se la procedura richiesta da un processo è nello spazio di memoria di un altro processo
* **Il collegamento dinamico si rivela molto utile per le librerie**

Sovrapposizione di servizi (overlay)

Si potrebbe decidere di scambiare delle parti nella memoria a seconda di eventi che avvengono

* Mantiene nella memoria soltanto **le istruzioni** **e i dati che si usano con maggior frequenza**
* Necessario per consentire a un processo di essere **più grande della memoria che gli si assegna**
* Questa tecnica non richiede alcun intervento speciale del SO, e po' essere realizzata direttamente dall’utente; molto complessa è invece la sua progettazione e programmazione

**Sovrapposizione di servizi per un assemblatore a due passi**

**Spiegazione**

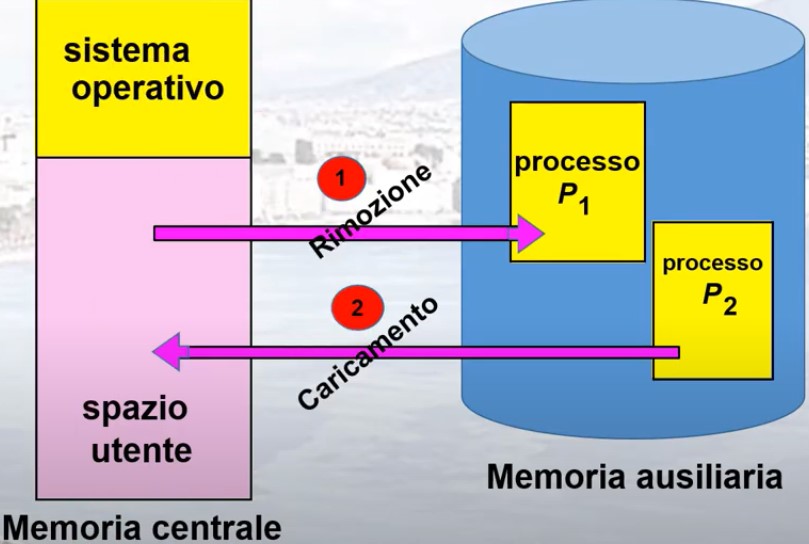
Una volta caricato una certa applicazione nella RAM, si potrebbe decidere di sostituire una parte in questa sezione (in questo esempio la parte da sostituire è l’ultimo quadrato dove stanno le frecce) con un primo passo, e poi con un secondo passo, a seconda di quelli che saranno gli eventi del sistema

Avvicinamento dei processi (swapping)

Con lo swap si intende scambiare l’ordine di presenza nella RAM di un certo codice. Questo scambio avviene perché si ha una memoria di scambio **(memoria ausiliaria chiamata backing store),** che attraverso il processo di swapping, permette di sostituire con altre aree

* **Memoria ausiliaria:** disco veloce sufficientemente capiente da contenere le copie di tutte le immagini di memoria di tutti i processi; deve permettere un accesso diretto a queste immagini di memoria
* **Roll OUT, Roll IN:** variante impiegata per gli algoritmi di scheduling basati sulle priorità: il processo con priorità inferiore viene scaricato dalla RAM per fare spazio all’esecuzione del processo con priorità maggiore
* La maggior parte del tempo d’avvicendamento è data dal tempo di trasferimento. Il tempo di trasferimento totale è direttamente proporzionale alla quantità di memoria interessata
* Si possono trovare versioni modificate di tecniche di avvicendamento su molti sistemi, es. UNIX, Linux e Windows

**Visione schematica dell’avvicendamento**

 **Spiegazione**

Nella RAM c’è una parte deve si trova il SO, e dall’altra parte lo spazio utente. Quindi per esempio una procedura viene rimossa (P1) e al suo posto viene caricato P2 (questo è lo swap), e ciò ottimizza l’uso della memoria, dato che si sostituisce un processo con un altro anziché richiedere altra memoria

Assegnazione contigua della memoria

È un’assegnazione che ha luogo quando si riesce a caricare, con spazi contigui di memoria centrale, l’oggetto che si vuole caricare dall’esterno (cioè se si ha un oggetto di grandezza 1000, lo si può caricare solo se in memoria si ha uno spazio di grandezza pari o maggiore all’oggetto). Si ha però bisogno di proteggere la memoria, cioè pur essendo contigua, si deve sapere dove finisce un processo/oggetto, e dove comincia il successivo. Tale schema lo si può ottenere mediante:

* **Un registro di rilocazione,** il quale contiene il valore dell’indirizzo fisico minore
* **Un registro limite,** che contiene l’intervallo di indirizzi logici (cioè dice quanto può essere lungo il segmento di codice al quale si deve fare riferimento)

(ciascun indirizzo logico deve essere minore del contenuto de registri limite)

**Registri di rilocazione e di limite**

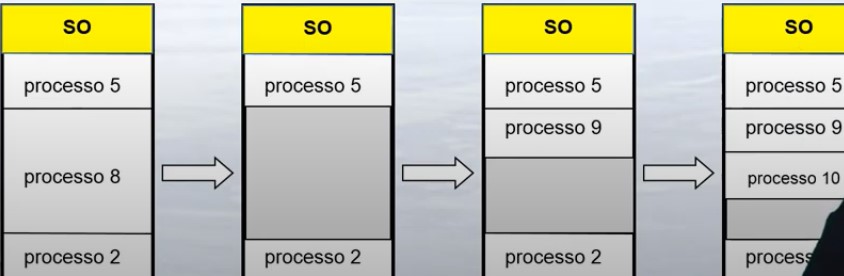
**Spiegazione**

Dalla CPU si elabora un indirizzo logico che bisogna andare a mappare sulla memoria fisica. Ciò viene fatto in tale modo: si confronta l’indirizzo logico, il quale deve essere ≥ di una certa base; se ciò non è vero allora deve segnalare un’eccezione; se invece è vera, e risulta compreso tra la base e la base+limite, si ottiene l’indirizzi fisico che si andrà a mappare poi nella memoria fisica

**Assegnazioni su più partizioni**

* **Buco (hole):** blocco di memoria disponibile; ve ne sono varie dimensioni, sparsi all’interno della memoria
* Quando si carica un processo che necessita di memoria, **occorre cercare un buco sufficientemente grande da contenerlo**
* Il SO tiene traccia di:

1. **Partizioni assegnate b) partizioni libere (buchi)**

**spiegazione**

inizialmente si hanno i processi 5, 8, 2. Poi l’8 finisce e lascia un buco libero all’interno del quale si può inserire qualche altro processo per esempio il 9 e cosi via

Assegnazione dinamica della memoria

La domanda da porsi per stabilire quale punto della memoria andare ad occupare per inserire un processo che dovrà poi essere elaborato è: in quale, dei buchi disponibili nella memoria, bisogna inserire il processo? Ci sono diverse regole per stabilire ciò:

1. **First-fit:** **si assegna il primo buco sufficientemente grande per contenere il processo**
2. **Best-fit: si assegna il buco più piccolo in grado di contenere il processo**; occorre compiere la ricerca su tutta la lista, sempre che non sia ordinata per dimensione. **Produce le parti di buco inutilizzate più piccole,** ed è pertanto noto come best-fit. Tale metodo è vantaggioso da un lato, in quanto il disavanzo (cioè l’avanzo di memoria) è poco; dall’altro lato invece è svantaggioso perché il disavanzo prodotto, essendo molto piccolo non servirà a nessuno, e quindi potrebbe andare perso. Allora si può pensare di usare la terza tecnica
3. **Worst-fit: si assegna il buco più grande**; anche in questo caso occorre esaminare tutta la lista. **Produce le parti di buco inutilizzate più grandi**

**First-fit e bes-fit funzionano meglio di worst-fit poiché riducono il tempo e l’utilizzo della memoria**

Frammentazione

È un indice che misura quanto lo spazio libero in memoria sia sparpagliato

* **Frammentazione esterna:** lo spazio di memoria totale è sufficiente per soddisfare una richiesta, ma non è contiguo.
* una soluzione al problema della frammentazione esterna è data dalla **compattazione:**
* **riordina il contenuto della memoria della memoria per riunire la memoria libera in un unico grosso blocco**
* **è possibile solo se la rilocazione è dinamica, e si compie nella fase di esecuzione**
* **è una soluzione fattibile, ma complicata da realizzare**
* **frammentazione interna:** la memoria assegnata può essere leggermente maggiore della memoria richiesta; la memoria è interna a una partizione, ma non è in uso. Significa che il SO ragiona per blocchi discreti di lunghezza prefissata; quindi questa misura la differenza tra ciò che si chiede e ciò che si ottiene.

Paginazione

* **Metodo di gestione della memoria il quale permette che lo spazio degli indirizzi fisici di un processo non sia contiguo**
* Suddivide la memoria fisica in **blocchi di memoria** di dimensione fisse (noti anche come **pagine fisiche o frame**). La dimensione di una pagina è tipicamente una potenza di 2, compresa tra 512 byte e 16 MB
* Suddivide la memoria logica in blocchi di uguale dimensione detti pagine (pages)
* Tiene traccia di tutti i blocchi liberi
* Utilizzauna tabella delle pagine per tradurre gli indirizzi logici in indirizzi fisici
* **Può evitare la frammentazione esterna**

A questo punto si deve trovare uno **schema di traduzione di questi indirizzi.**

il SO deve essere capace di puntare alla pagina intera (quindi il blocco), e poi per puntare ad una parte della pagina, bisogna avere una variabile chiamata **scostamento di pagina**; cioè di quanto ci si deve spostare a partire dall’inizio della pagina, per trovare la riga desiderata.

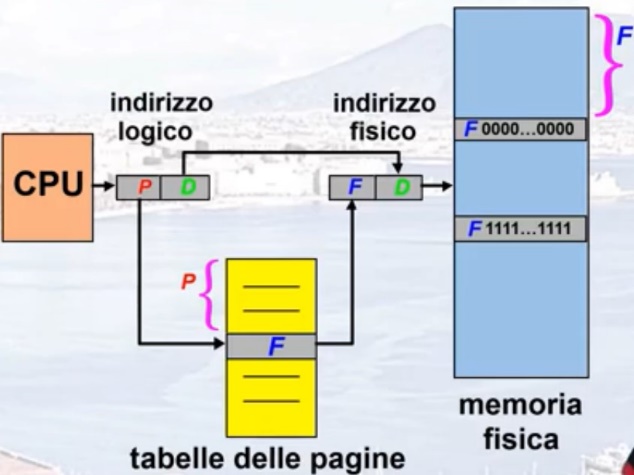
Ogni indirizzo generato dalla CPU è diviso in:

* **Numero di pagina (p):** serve come indice per la tabella delle pagine, che contiene **l’indirizzo di base nella memoria fisica di ogni pagina**
* **Scostamento di pagina (d): combinato con l’indirizzo di base definisce l’indirizzo della memoria fisica,** che si invia all’unità di memoria

**Ricorda:**

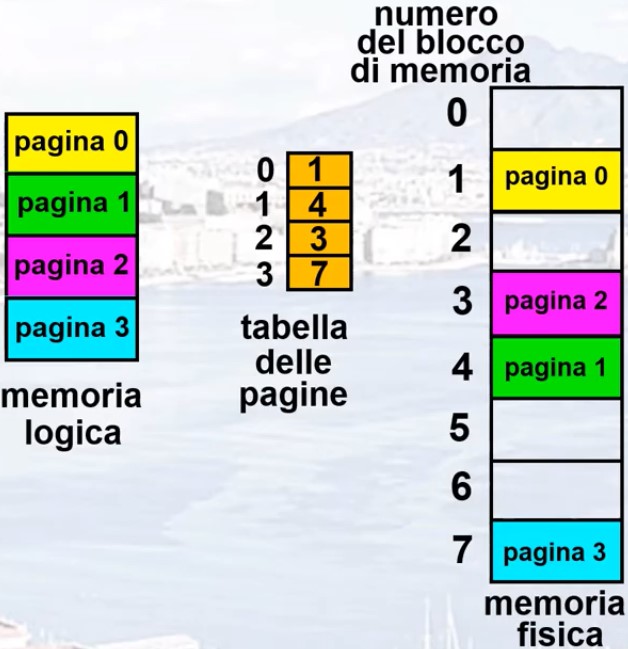
le pagine, e la tabella delle pagine permette di spezzare l’oggetto in tutti oggetti uguali che si andranno a posizione nella memoria centrale in vari punti diversi. Quindi la tabella delle pagine serve proprio per ricomporre l’oggetto quando poi servirà.

L’utente non vede questa discontinuità, ma vede tutto compatto grazie al SO

**Architettura di paginazione**

**Spiegazione**

Dalla CPU esce un indirizzo logico composto da una P e una D. La P indica un indirizzo che punta alla tabella delle pagine, che scorre di p posizioni, e alla p-esima posizione trova in questo caso la F che va a comporre l’indirizzo fisico in memoria. La D invece rimane uguale e corrisponde allo spiazzamento

**Esempio di paginazione**

**Spiegazione**

ci sta una memoria logica di un certo processo che è composta da 4 pagine; poi ci sta una tabella delle pagine che permette di effettuare al corrispondenza tra pagine logiche e fisiche. Quindi in questo caso indica che la pagina logica 0 corrisponde alla pagina 1 fisica, e così via.

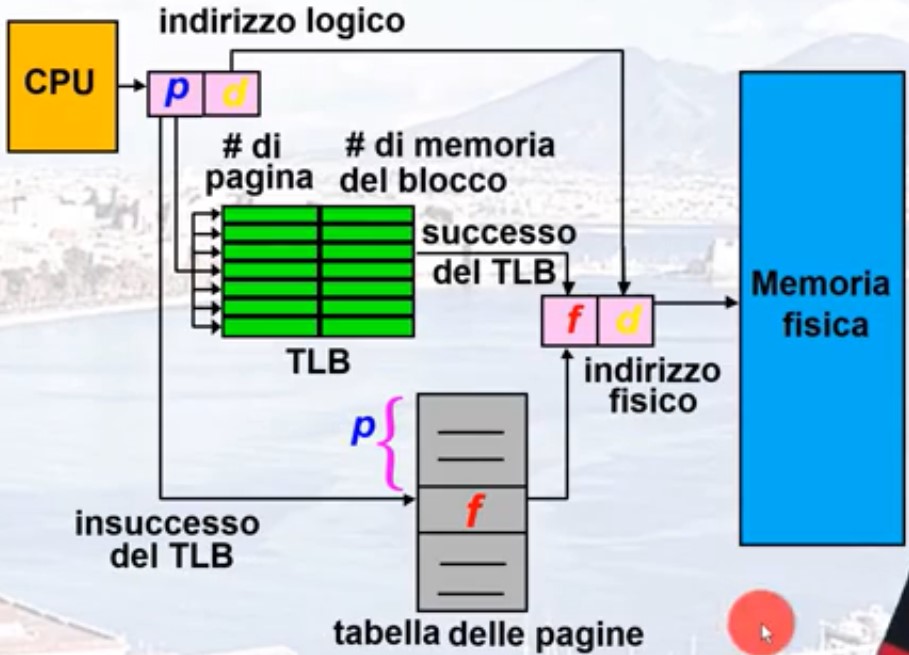
Architettura di paginazione

* La tabella delle pagine è mantenuta nella memoria principale
* Un **registro di base della tabella delle pagine (page-table base register, PTBR)** punta alla tabella stessa
* Un **registro di lunghezza della tabella delle pagine (page-table lenghtregister, PRLR)** indica la dimensione della tabella delle pagine
* Con questo metodo, per accedere a un byte occorrono due accessi alla memoria, uno per l’elemento della tabella delle pagine e uno per il byte stesso
* La soluzione tipica a questo problema consiste nell’impiego di una speciale, piccola cache di memoria veloce detta **TLB (translation look-aside buffer)**, una memoria associativa ad alta velocità

Memoria associativa (TLB)

Memoria ad alta velocità che nel momento in cui si dà in input un certo numero di pagine, in automatico, fa corrispondere subito un blocco di memoria (quindi in un unico ciclo senza dover fare la ricerca sequenziale, dice subito che la pagina che si cerca sta in un determinato blocco). Questo TLB viene concretizzato attraverso una memoria associativa che effettua una ricerca parallela, quindi è un modulo più costoso proprio perché è fatto in modo tale da avere un accesso parallelo, dal quale si hanno notevoli vantaggi.

**Architettura di paginazione con TLB**

**Spiegazione**

Dalla CPU esce l’indirizzo logico composto da P e D. Se P è presente nel TLB, il corrispondente numero di blocco di memoria è immediatamente disponibile e si usa per accedere alla memoria fisica. Se invece non è presente nel TLB di deve consultare la tabella delle pagine.

Tempo effettivo d’accesso

* Lookup associativo = ɛ unità di tempo
* Si assuma un tempo d’accesso alla memoria di 1 microsecondo
* **Tasso di successi (hit ratio):** percentuale di volte che un numero di pagine si trova nel TLB; relativo al numero di registri associativi
* Tasso di successi = α
* Tempo effettivo d’accesso (EAT,effective access time)

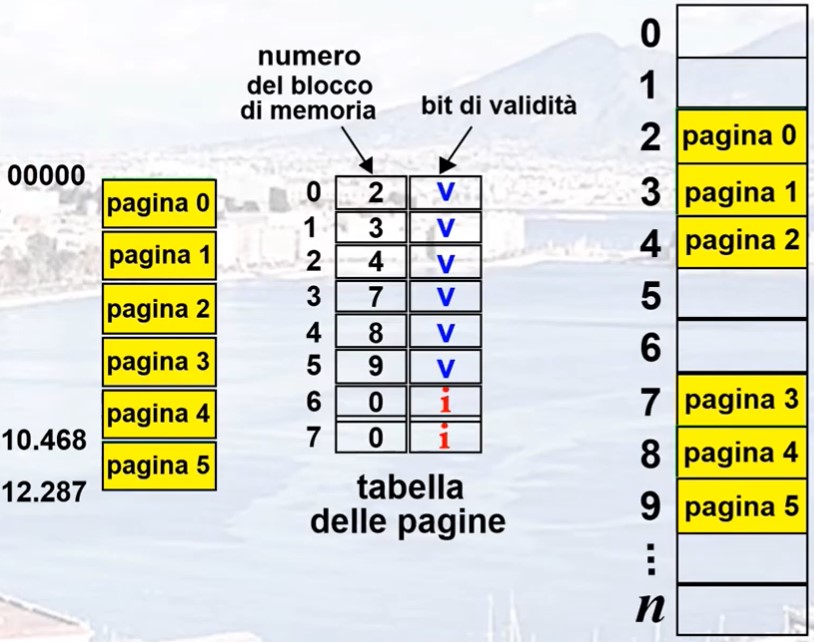
**EAT = (1 +ɛ) α + (2 + ɛ)(1-α) = 2 + ɛ - α**

Protezione della memoria

* In un ambiente paginato, la protezione della memoria è assicurata dai bit di protezione associati a ogni blocco di memoria
* Di solito si associa un **bit di validità** a ciascun elemento della tabella delle pagine:
* **“bit valido”** indica che la pagina corrispondente è nello spazio d’indirizzi logici del processo, quindi è una pagina valida
* **“bit non valido”** indica che la pagina non è nello spazio d’indirizzi logici del processo

Il bit di validità assume due valori 0-1

**Esempio**

**Spiegazione**

A sx ci sta il processo contiguo, e a dx quello paginato; poi c’è la tabella delle pagine. A sx di esse ci sta il numero del blocco di memoria e si va a verificare (con il bit di validità) se questo è valido o no

Struttura della tabella delle pagine

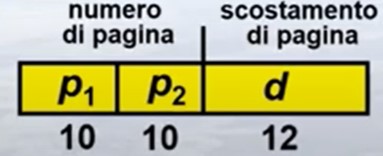
Nella tabella delle pagine vista fin ora c’è un problema. La tabella delle pagine descritta prima è praticamente un libretto delle istruzioni su come montare un processo. Tale tabella però potrebbe essere molto lunga; quindi non è detto che essa possa essere contenuta in memoria contigua. Di conseguenza per processi molto lunghi, le stesse istruzioni devono essere frammentate. Perciò occorre fare un passo in avanti; esistono vari tipi di paginazione:

1. **Paginazione gerarchica:** è un tipo di paginazione con la quale si accede alla pagina effettiva del blocco di memoria fisica, partendo da quella logica, non in modo diretto; ma essendo la tabella delle pagine anch’essa molto grande, si deve frammentare, si crea quindi un ulteriore indice, ovvero, una nuova tabella delle pagine che punta alle tabelle delle pagine frammentate, la quale a loro volta puntano alla memoria fisica.

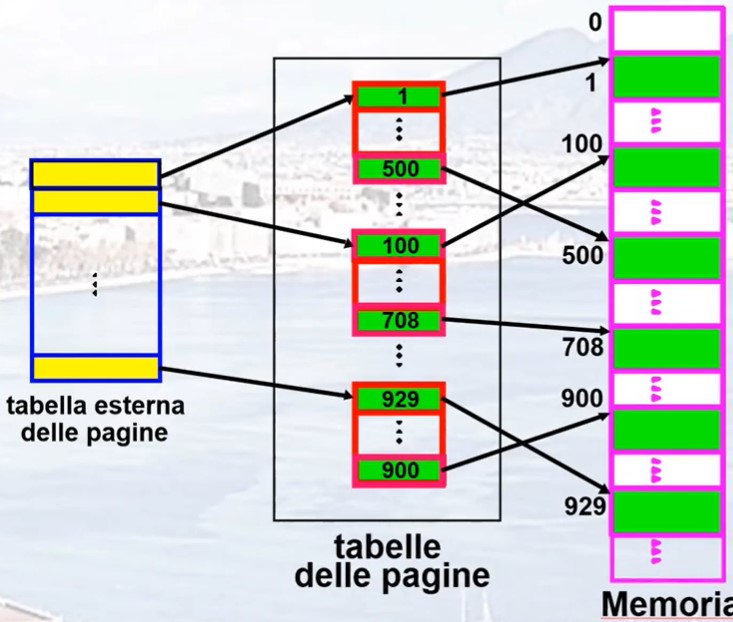
Dal punto di vista implementativo, prima c’era un indirizzo unico che per convenzione era diviso in 2 parti (pagina logica e scostamento). Con lo scostamento si scende fino al punto desiderato.

P D

Ora invece dato che la tabella delle pagine non riesce a contenere, si divide ulteriormente la parte sx in due parti P1 e P2, dove P1 è un indice della tabella delle pagine, e P2 è lo scostamento all’interno della pagina indicata dalla tabella esterna delle pagine



**Esempio**

****

1. **Tabelle delle pagine di tipo hash**

* Comune negli spazi d’indirizzi relativi ad architetture > 32bit
* L’argomento della funzione di hash è il numero della pagina virtuale. Ogni elemento della tabella hash contiene una lista concatenata di elementi che la funzione hash fa corrispondere alla stessa locazione
* I numeri di pagina virtuali vengono confrontati e, se coincidono, si usa l’indirizzo del relativo blocco di memoria per generare l’indirizzo fisico desiderato

**Funzione hash**

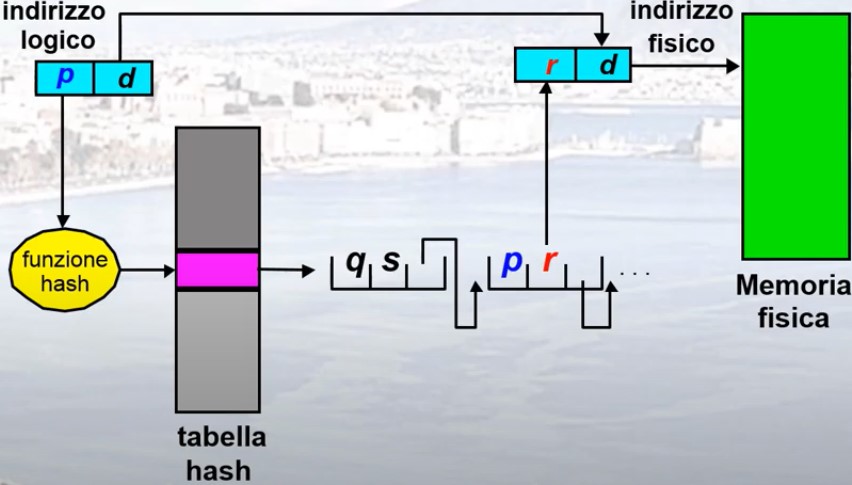
Partendo da una parola, restituisce un valore.

Ad esempio se si prende la parola CASA e si vuole sapere in che punto della memoria si trova, invece di andare a scorrere tutta la memoria, si applica il criterio hash. Tale criterio funziona così: prende la parola casa, e la tratta non come caratteri, ma bensì come byte numerici; vuol dire che si sommano le lettere di casa, le quale sono viste come interi (si è scelto di fare la somma, ma si potrebbe applicare infiniti criteri tipo sottrazione, quadrati ecc); quindi si sommano i byte delle 4 lettere, ed esce un certo valore. Questo sarà il valore che punta alla RAM.

Questo criterio è molto potente, in quanto permette di fare accesso alla memoria senza alcun tipo di indice, questo perché l’indice è generale, dato che **si prende la chiave, e si genera con la funzione hash, quindi non è più una ricerca, ma una funzione che prende un indice dalla memoria logica, e lo trasforma in un indice della memoria fisica, facendo soltanto una traduzione funzionale.**

**Svantaggi**

La funzione hash, soffre del problema delle collisioni, ovvero, se si fa una permutazione della parola CASA, tipo SACA, e si fa la somma delle lettere come visto prima, si crea una chiave identica a quella di casa.

**Esempio**

**Spiegazione**

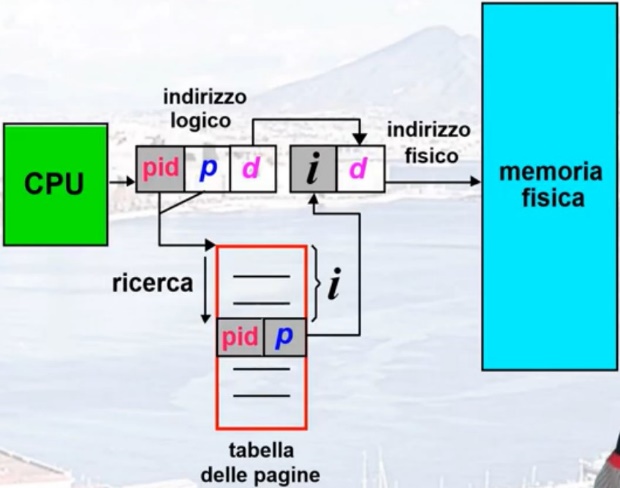
Ci sta sempre indirizzo logico della pagina e scostamento (P, D). La D si riporta così come è nella parte destra. Invece alla P si applica la funzione hash, la quale restituisce un certo valore, che corrisponde ad una lincked list, si percorrono nodi fin quando non si trova il nodo che contiene la P e in questo caso la r (la corrispondenza fisica di P) (sono nello stesso nodo uno dopo l’altro); dopo di che si compone l’indirizzo fisico sostituendo la P con la r che permette l’accesso alla memoria fisica

1. **Tabella delle pagine invertita**

* Un elemento per ogni pagina reale (blocco di memoria)
* Ciascun elemento è costituito dall’indirizzo virtuale della pagina memorizzata in quella reale locazione di memoria, con informazioni sul processo che possiede tale pagina
* Riduce la quantità di memoria necessaria per memorizzare ogni tabella delle pagine, ma aumenta il tempo di ricerca nella tabella quando si fa riferimento a una pagina
* Utilizza una tabella hash che riduce la ricerca a un solo – o a pochi – elementi della tabella delle pagine

Tutto ciò significa che il termine **inversione** sta a considerare il fatto che invece di utilizzare tante tabelle delle pagine almeno quanti sono i processi (dato che le tabelle potrebbero essere frammentate, e quindi di più), **si può ipotizzare di usare una tabella delle pagine che corrisponde alla mappatura della memoria centrale**

**Esempio**

**spiegazione**

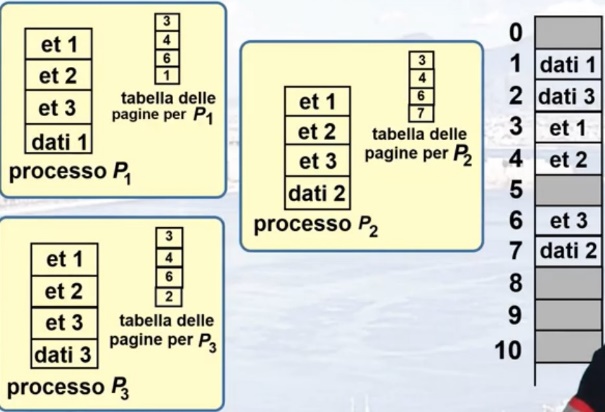
c’è una terna (PID, P, D). il PID (sarebbe carta d’identità del processo) e P serviranno insieme per cercare nella tabella unica (tabella delle pagine invertita), e laddove si trova un certo PID con una certa pagina, si va a vedere tale coppia a quale elemento corrisponde, in questo caso si prende la i, ovvero, quante righe si è dovuto leggere prima di arrivare a questa riga. Dopo di che tale i , va a comporre il blocco definitivo per poter accedere alla memoria fisica. Dove ci sta (PID, P), ci potrebbe essere anche (PID, Q) ecc, perché questo processo logico è composto da un certo numero di pagine P1, P2….; di conseguenza nella tabella delle pagine, si potrebbe trovare (PID, P1), (PID,P2) ecc. però la cosa importante è che in questa tabella delle pagine, si avrà la mappatura completa della memoria, nella quale si sta esattamente per ogni pagina, a quale processo è assegnata.

**Questo schema permette di gestire anche al meglio la condivisione, perché si sa che ci sono blocchi di informazioni che possono essere condivise tra processi, e questa tabella implementa facilmente anche tale concetto.**

Pagine condivise

* **Codice condiviso**
* Una copia di codice rientrante condiviso tra i processi (es. editor di testi, compilatori, interfacce a finestre)
* Il codice condiviso deve essere collocato nella stessa locazione nello spazio degli indirizzi logici di tutti i processi
* **Codice privato e dati**
* Ciascun processo dispone di una propria copia di codice e dati
* Le pagine per il codice e i dati possono essere collocate in un qualsiasi punto nello spazio degli indirizzi logici

**Esempio di pagine condivise**

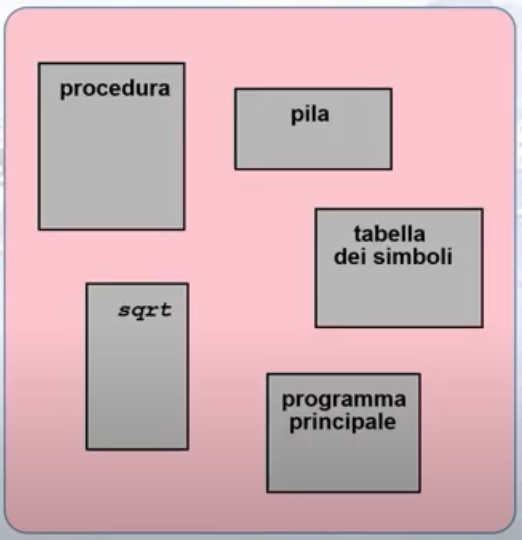
**spiegazione**

Ci sono 3 propcessi P1,P2,P3. Ognuno composto da blocchi di dati, e tablle delle pagine. I dati et1, et2, et3, sono condivisi tra i 3 processi. Quindi nella memoria centrale non si dovranno duplicare, e se si effettua una tabella delle pagine invertita, si ha il compito facilitato, in quanto si possono trovare diversi PID, quindi diversi processi che condividono

Le stesse celle di memoria

Segmentazione

È il criterio di divisione di un oggetto, ma non in parti uguali come la paginazione, bensì seguendo la natura dell’oggetto (quindi di grandezze diverse). Infatti dal punto di vista dell’utente si hanno diversi blocchi così:

quindi l’utente vede una contiguità logica, cioè non ha bisogno di fare una suddivisione in pagine o altro. Sicuramente la suddivisione in pagine permette di realizzare un ingresso nella memoria centrale, pur non essendo una contiguità; ma ciò che è stato fatto è unire i concetti di spazi diversi (segmenti), con quelli di frammentazione. Questa fusione è fatta in questo modo: si divide il processo in tanti pezzetti, quindi è ovvio che la gestione sarà più complessa, dato che quando si vogliono inserire gli oggetti con forme diverse (forme intese come lunghezze); questo significa conservare la natura dell’oggetto.

**Vantaggio**

Soddisfa l’utente, che vede le cose più simili alla sua visione (cioè segmentate come nella figura). Ha anche il vantaggio che se si vede un oggetto, lo si contiene in uno spazio contiguo, è ovvio che tutte le operazioni che si fanno in quello spazio sono più facilitate, perché non si devono cercare i vari pezzi e vedere dove sono stati collocati. Quindi

gli spazi nella memoria fisica non sono più uguali, ma rispecchiano un po' quelli che sono gli spazi dell’utente.

Architettura di segmentazione

* **Un indirizzo logico è una coppia:**

**<numero di segmento, scostamento>**

* Tabella dei segmenti (segment table): traduce gli indirizzi bidimensionali definiti dall’utente negli indirizzi fisici unidimensionali; ogni elemento è composto da:
* **Base:** contiene l’indirizzo fisico iniziale della memoria al quale il segmento risiede
* **Limite:** contiene la lunghezza del segmento
* **Registro di base della tabella dei segmenti**

(segment-table baseregister, STBR) punta alla tabella dei segmenti in memoria

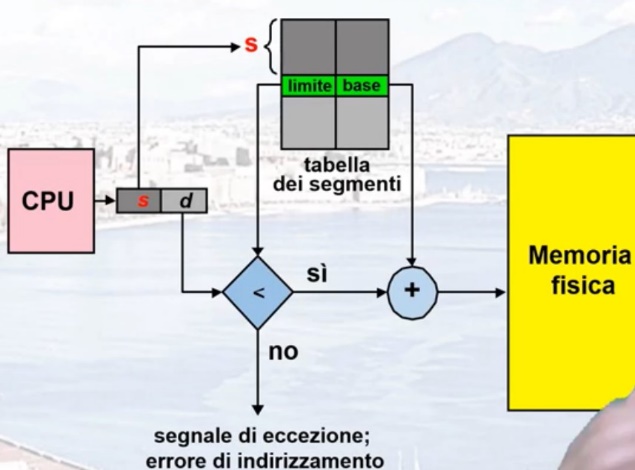
* **Registro di limite della tabella dei segmenti**

(segment-table lenght register, STLR) indica il numero di segmenti usati da un programma;

numero di segmenti s è valido se s < STLR

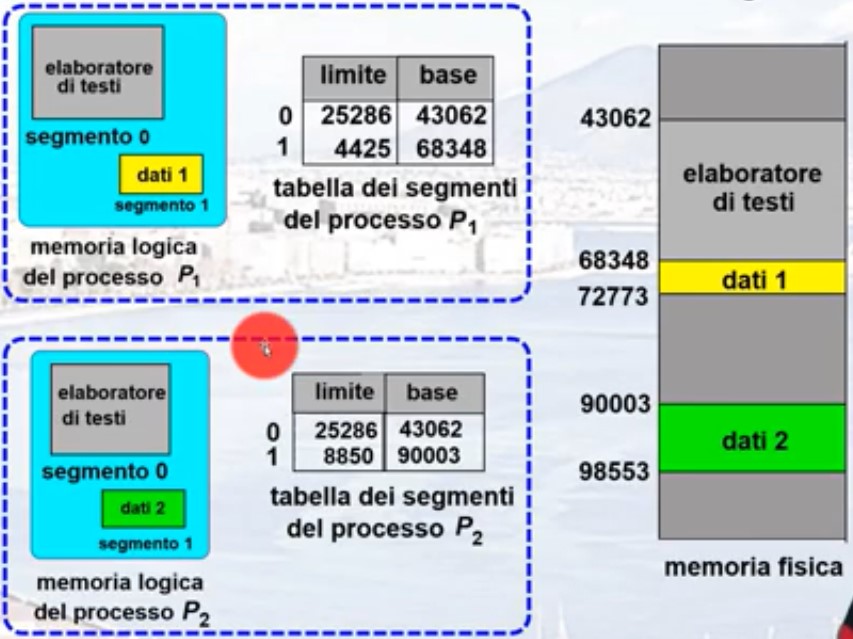
* **Rilocazione**
* Dinamica
* Tramite tabella dei segmenti
* **Condivisione**
* Segmenti condivisi
* Steso numero di segmenti
* **Assegnazione della memoria**
* First fit/best fit
* Frammentazione esterna
* Protezione. A ciascun elemento della tabella dei segmenti è associato:
* Bit di validazione 0 => segmento illegale
* Privilegi read/write/execute
* Bit di protezione associati ai segmenti; la condivisione di codice avviene a livello di segmento
* Poiché **i segmenti variano in lunghezza, l’assegnazione della memoria** è un problema di allocazione dinamica

**Esempio**

**spiegazione**

Dalla CPU esce un indirizzo (s,d) suddiviso in 2 pezzi spiazzamento d, e segmento s. Essendo il segmento variabile, bisogna sapere dove si trova, quanto è lungo e dove inizia. Lo si va a cercare nella tabella dei segmenti (dove trovo limite (indica quanto è lungo), e base (indica dove inizia). Una volta ricavati entrambi, si prende il limite e si confronta con l’indirizzo di spiazzamento, e tale d deve essere < del limite. Se ciò è vero, si va avanti e si somma la base allo spiazzamento e si ottiene in uscita l’indirizzo fisico di memoria.

**Anche con la segmentazione si può pensare di condividere:**

*  **dati 1 e dati 2 sono diversi**
* **Elaboratore di testo invece è ugualòe in entrabi i processi P1 e P2**