# CAPITOLO 8: MEMORIA CENTRALE

Uno dei risultati dello scheduling della CPU consiste nella possibilità di migliorare sia l’utilizzo della CPU sia la rapidità con cui il calcolatore risponde ai propri utenti. Per ottenere questo aumento delle prestazioni occorre tener la memoria parecchi processi: la memoria deve essere condivisa

## 8.1 INTRODUZIONE

La memoria è fondamentale nelle operazioni di un moderno sistema di calcolo; consiste in un ampio vettore di parole o byte, ciascuno con il proprio indirizzo. La CPU preleva le istruzioni dalla memoria sulla base del contenuto del program counter; tali istruzioni possono determinare letture (load) o scritture (store) in specifici indirizzi di memoria.

## 8.1.1 Dispositivi essenziali

La **memoria centrale** e i **registri** incorporati nel processore sono le sole aree di memorizzazione alle quali la CPU può accedere direttamente. Vi sono istruzioni macchina che accettano gli indirizzi di memoria come argomenti, ma nessuna accetta gli indirizzi del disco. Pertanto, qualsiasi istruzione in esecuzione, e tutti i dati utilizzati dalle istruzioni, devono richiedere in uno di questi dispositivi per la memorizzazione ad accesso diretto. I dati che non sono in memoria devono essere caricati prima che la CPU posso operare su di loro.

-I registri incorporati nella CPU sono accessibili, in genere, nell’arco di un ciclo dell’orologio di sistema. Molte CPU sono capaci di decodificare istruzioni ed effettuare semplici operazioni sui contenuti dei registri alla velocità di una o più operazioni per ciclo. Ciò non vale per la memoria centrale, Cui si accede attraverso una transizione sul bus della memoria.

Nei casi in cui, l'accesso alla memoria sia molto lento e richieda molti cicli di orologio, il processore entra in stallo perché manca dei dati richiesti per completare l'operazione che sta eseguendo.

Per rimediare a queste situazioni si fa uso di **una memoria veloce (CACHE) interposta tra la CPU e la memoria centrale.**

È importante proteggere il sistema dall’accesso degli utenti e salvaguardare i processi utenti l’uno dall’altro. Innanzitutto, bisogna assicurarsi che ciascun processo abbia uno spazio di memoria separato. A Tal fine, occorre poter determinare l'intervallo di indirizzi a cui un processo può accedere legalmente, e garantire che possa accedere soltanto a questi indirizzi. Si può implementare il **meccanismo di protezione** tramite **due registri**, detti **registri base e registri limite**. Il **registro di base** contiene il più piccolo indirizzo legale della memoria fisica, il **registro limite** determina la dimensione dell’intervallo ammesso. Per mettere in atto il meccanismo della protezione la CPU confronta ciascun indirizzo generato in modalità utente con i valori contenuti nei due registri. Un tentativo da parte di un programma eseguito in modalità utente di accedere ad aree riservate di memoria (anche memoria riservata al SO) causa un segnale di eccezione. Solo il SO può caricare i registri base e limite grazie a una istruzione privilegiata. Grazie all’esecuzione nella modalità di sistema il SO ha la possibilità di accedere sia alla memoria a esso riservata sia a quella riservata agli utenti. Questo privilegio consente al sistema di caricare i programmi utenti nelle aree di memoria loro riservate; Di generare copie del contenuto di queste regioni di memoria (*dump)* a scopi diagnostici.

## 8.1.2 Associazione degli indirizzi

In generale un programma risiede in un disco sotto forma di un file binario eseguibile.

Per essere eseguito, il programma va caricato in memoria e inserito all’interno di un processo. Durante la sua esecuzione il processo può essere trasferito dalla memoria al disco e viceversa.

L’insieme dei processi presenti nei dischi e che attendono d’essere trasferiti in memoria per essere eseguiti forma la coda di ingresso *(input queue)*.

La procedura normale consiste nello sceglier uno dei processi appartenenti alla coda di ingresso e caricarlo in memoria. Il processo durante l'esecuzione può accedere alle istruzioni e dati in memoria. Quando il processo termina si dichiara disponibile il suo spazio di memoria

Nella maggior parte dei casi un programma utente, prima di essere seguito, deve passare attraverso vari stati, alcuni dei quali facoltativi, In cui gli indirizzi sono rappresentabili in modi diversi. Generalmente gli indirizzi del programma sorgente sono simbolici. Un compilatore di solito **associa** (bind) questi indirizzi simbolici a indirizzi rilocabili. L'editor dei collegamenti *(linkage editor),* o il caricatore *(loader),* fa corrisponde a sua volta questi indirizzi di rilocabili a indirizzi assoluti.

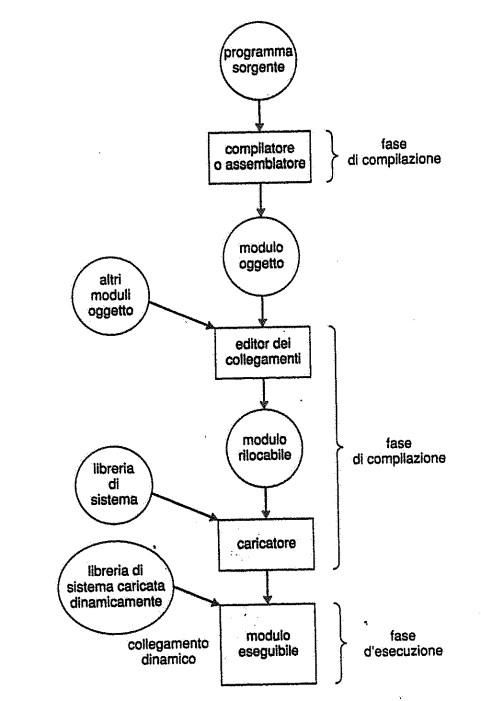
Generalmente l’associazione di istruzione e dati a indirizzi di memoria si può compiere in qualsiasi fase del seguente percorso.

* **Compilazione:** Se nella fase di compilazione si sa dove il processo risiederà in memoria, si può generare codice assoluto. Se, per esempio, e noto a priori che un processo

utente inizia alla locazione r, anche il codice generato dal compilatore comincia da

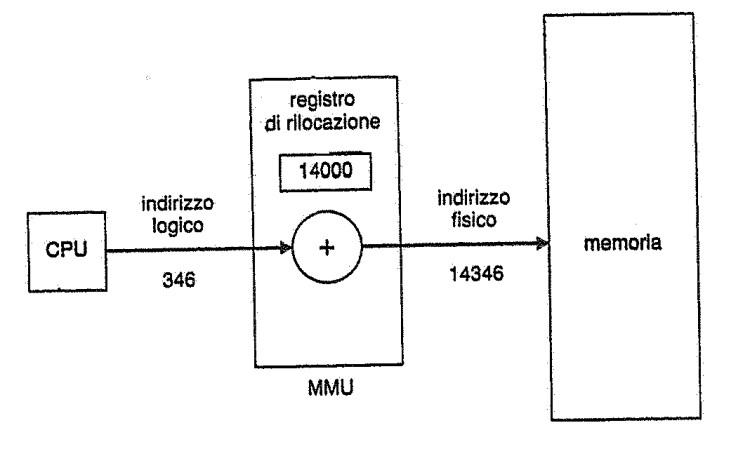
quella locazione. Se per caso, la locazione iniziale cambiasse, sarebbe necessario ricompilare il codice.

* **Caricamento.** Se nella fase di compilazione non è possibile sapere in che punto della memoria risiedere il processo, il compilatore deve generare codice rilocabile. Se l’indirizzo iniziale cambia, è sufficiente ricaricare il codice utente per incorporare il valore modificato
* **Esecuzione.** Se durante l’esecuzione il processo può essere spostato da un segmento di memoria a un altro, si deve ritardare l’associazione degli indirizzi fino alla fase d’esecuzione.



## 8.1.3 Spazi di indirizzi logici e fisici a confronto

Un indirizzo generato dalla CPU di solito si indica come indirizzo logico, mentre un indirizzo visto dall’unita di memoria, cioè caricato nel registro dell’indirizzo di memoria (MAR) di solito si indica come indirizzo fisico. **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**, i**nvece, gli indirizzi logici non coincidono con gli indirizzi fisici**. In questo caso si riferisce di solito agli **indirizzi logici** col termine di **indirizzi virtuali**. L’insieme di tutti gli indirizzi logici generati da un programma è lo **spazio degli indirizzi logici**, quelli degli indirizzi fisici, corrispondenti a tali indirizzi logici è **detto spazio degli indirizzi fisici**. L’associazione nella fase d’esecuzione dagli indirizzi virtuali agli indirizzi fisici è svolta da un dispositivo detto **unita di gestione della memoria MMU**. Il registro base è ora denominato registro di rilocazione: quando un processo utente genera un indirizzo (indirizzo virtuale) , prima dell’invio all’unità di memoria, si somma tale indirizzo al valore contenuto nel registro di rilocazione.



Il programma utente non considera mai gli indirizzi fisici reali. Il programma crea un puntatore alla locazione 346, lo memorizza, lo modifica e lo confronta con gli altri indirizzi, tutto ciò semplicemente come un numero. Solo quando assume il ruolo di un indirizzo di memoria si riloca il numero sulla base del contenuto del registro di rilocazione. Il programma utente tratta indirizzi logici, l’architettura del sistema converte gli indirizzi logici in indirizzi fisici.

{Esistono 2 tipi di indirizzi: gli indirizzi logici (nell’intervallo da 0 a max) e indirizzi fisici (nell’intervallo da r+0 a r + max dove r è il registro base). L’utente genera solo indirizzi logici e pensa che il processo eseguito nelle posizioni da 0 a max. Il programma utente fornisce indirizzi logici che prima d’essere usati si devono far corrispondere a indirizzi fisici. Il concetto di spazio di indirizzi logici associato uno spazio indirizzi fisici separato è fondamentale per una corretta gestione della memoria.}

## 8.1.4 Caricamento dinamico

Per migliorare l’utilizzo della memoria si può ricorrere al **caricamento dinamico** mediante il quale si carica una procedura solo quando viene richiamata; tutte le procedure si tengono in memoria secondaria in un formato di caricamento rilocabile.

Si carica il programma principale in memoria e quando, durante l'esecuzione, una procedura deve richiamare un'altra, si controlla innanzitutto che sia stata caricata, altrimenti si richiama il caricatore di collegamento rilocabile per caricare memoria la procedura richiesta e aggiornare le tabelle degli indirizzi del programma in modo che registrano questo cambiamento.

Il vantaggio dato dal caricamento dinamico consiste nel fatto che una procedura che non si adopera non viene caricata. Questo metodo e utile soprattutto quando servono grandi quantità di codice per gestire casi non frequenti, per esempio come le procedure di gestione degli errori.

## 8.1.5 Collegamento dinamico e librerie condivise

Alcuni SO consentono solo il collegamento statico, in cui le librerie di sistema del linguaggio sono trattate come qualsiasi altro modulo oggetto e combinate dal caricatore nell’immagine binaria del programma.

-In questo caso invece di differire il caricamento (CARICAMENTO DINAMICO) differisce il collegamento di una procedura fino al momento dell'esecuzione.

-Senza questo strumento tutti i programmi di sistema dovrebbero disporre all'interno dell'immagine eseguibile di una copia della libreria di linguaggio e tutto ciò richiede spazio nei dischi e in memoria centrale.

Con il collegamento dinamico invece per ogni riferimento a una procedura di libreria si inserisce all’interno dell’immagine eseguibile una piccola porzione di codice di riferimento che indica come localizzare la giusta procedura di libreria residente in memoria o come caricare la procedura se non è presente.

Durante l’esecuzione il codice di riferimento controlla se la procedura richiesta è in memoria altrimenti provvede a caricarla. in ogni caso tale codice sostituisce sé stesso con l’indirizzo della procedura, che

viene poi eseguita.

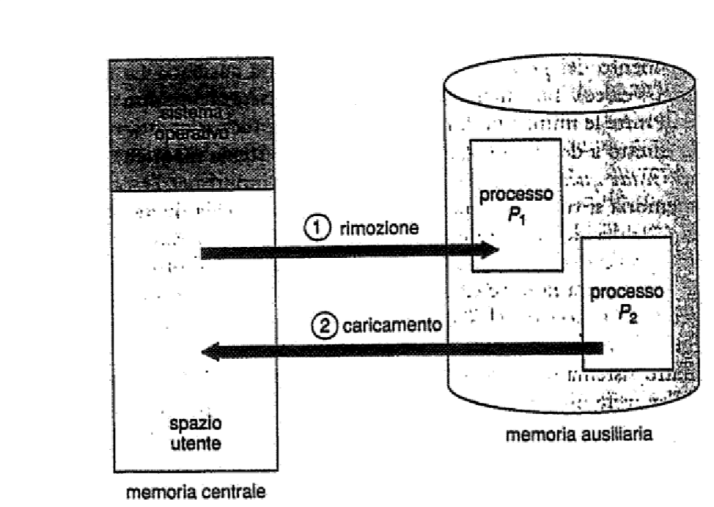
In questo modo, quando si raggiunge nuovamente quel segmento del codice, si esegue direttamente la procedura di libreria, senza costi aggiuntivi per il collegamento dinamico. Questa caratteristica si può estendere anche gli aggiornamenti delle librerie, una libreria si può sostituire con una nuova versione, e tutti i programmi che fanno riferimento a quella libreria usano automaticamente la nuova. È possibile caricare memoria più di una versione nella stessa libreria, così ciascun programma si serve dell’informazione sulla versione per decidere quale coppia debba usare. I programmi collegati prima dell’installazione della nuova libreria continua avvalersi della vecchia libreria, questo sistema è noto anche con il nome di **librerie condivise.**

## 8.2 AVVICENDAMENTO DEI PROCESSI (SWAPPING)

Per essere eseguito un processo deve trovarsi in memoria centrale ma si può trasferire temporaneamente in memoria ausiliare da cui riporta la memoria centrale nel momento in cui si deve riprenderne l’esecuzione.

Si consideri, per esempio, un ambiente di multiprogrammazione con un algoritmo circolare (*round-Robin*) per lo scheduling della CPU. Trascorso un quanto di tempo, il gestore di memoria scarica dalla memoria il processo appena terminato e carica un altro processo nello spazio di memoria appena liberato; questo procedimento si chiama **avvicendamento dei processi in memoria** o, più brevemente, **avvicendamento** o **scambio** *(swapping).*

Nel frattempo lo scheduler della CPU assegna un quanto di tempo un altro processo presente in memoria.



Una variante di questo criterio d’avvicendamento dei processi s’impiega per gli algoritmi di scheduling basati sulle priorità. Se si presenta un processo con priorità maggiore, il gestore della memoria può scaricare dalla memoria centrale il processo con priorità inferiore per far spazio all’esecuzione del processo con priorità maggiore. Quando il processo con priorità maggior termina, si può ricaricar in memoria quello con priorità minore e continuare la sua esecuzione. Normalmente un processo quando viene scaricato deve essere ricaricato nello stesso spazio di memoria occupato prima. Questa limitazione è dovuta al metodo di associazione di indirizzi. Se l’associazione degli indirizzi logici a quelli fisici si effettua nella fase di assemblaggio o caricamento, il processo non può essere caricato altrove mentre se avviene in fase d’esecuzione può essere riversato in uno spazio di memoria diverso, poiché gli indirizzi fisici si calcolano nella fase di esecuzione.

L’avvicendamento dei processi richiede una memoria ausiliaria. Tale memoria ausiliaria deve essere abbastanza ampia da contenere le copie di tutte le immagini di memoria di tutti i processi utenti e deve permettere un accesso diretto a tali immagini in memoria.

Il sistema mantiene una **coda dei processi pronti (ready queue)** formati da tutti i processi pronti per l’esecuzione le cui immagini di memoria si trovano in memoria ausiliaria o in memoria.

Quando lo scheduler della CPU decide di seguire un processo, richiama il dispatcher, che controlla se il primo processo della coda si trova in memoria. se non si trova in memoria, e in questo non c'è spazio, il dispatcher scarica un processo della memoria e vi carica il processo richiesto dallo scheduler della CPU (tempo del cambio di contesto elevato).

Un processo con requisiti di memoria dinamici deve impiegare chiamate di sistema (**request memory** e **release memory**) per informare il sistema operativo delle modifiche da apportare alla memoria.

Per scaricare un processo dalla memoria è necessario essere certi che sia completamente inattivo, soprattutto che non abbia operazioni di I/O pendenti

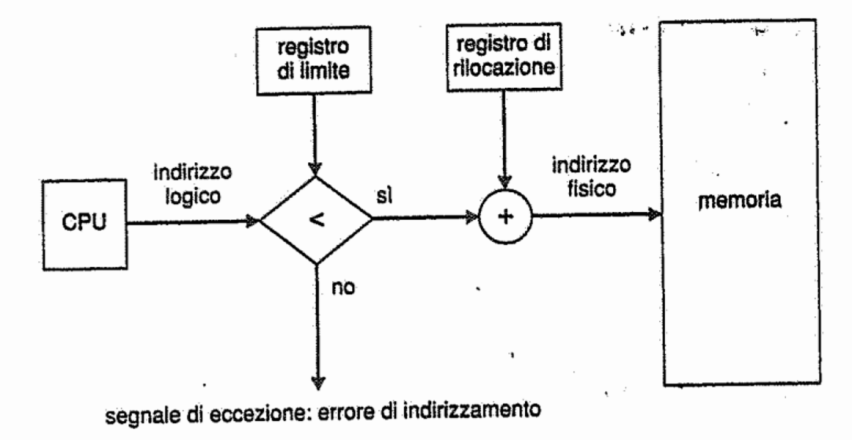
Per utilizzare al meglio la CPU è necessario che il tempo d’esecuzione di ciascun processo sia lungo rispetto al tempo d’avvicendamento. L'avvicendamento semplice si usa in pochi sistemi; richiede infatti un elevato tempo di gestione e consente un tempo di esecuzione troppo breve per essere considerato una soluzione ragionevole al problema di gestione della memoria. Versioni modificate dell'avvicendamento dei processi si trovano comunque in molti sistemi.

**8.3 ALLOCAZIONE CONTIGUA DELLA MEMORIA**

La memoria centrale deve contener sia il SO che i vari processi che si vogliono eseguire. Di solito si divide in 2 partizioni, una per il SO e l’altra per i processi utenti. Il SO si può collocare sia in memoria bassa che in quella alta ma normalmente viene collocato in memoria bassa vicino al vettore delle interruzioni. Con **l’allocazione contigua della memoria**, ciascun processo è contenuto in una singola sezione contigua di memoria.

**8.3.1 Rilocazione e protezione della memoria**

La protezione della memoria si può realizzare usando un registro di rilocazione che contiene il valore dell’indirizzo fisico minore, in registro limite contiene l’intervallo di indirizzi logici. Con i registri di rilocazione e limite, ogni indirizzo logico deve essere minore del contenuto del registro limite, la **MMU** fa corrispondere *dinamicamente* l’indirizzo fisico all’indirizzo logico sommando a quest’ultimo il valore contenuto nel registro di rilocazione.



Quando lo scheduler della CPU seleziona un processo per l’esecuzione il dispatcher durante l’esecuzione del cambio di contesto carica il registro di rilocazione e il registro limite con i valori corretti. Siccome si confronta ogni indirizzo generato dalla CPU con i valori contenuti in questi registri, si possono proteggere il SO, i programmi e i dati di altri utenti Da tentativi di modifiche da parte del processo in esecuzione.

Lo schema con registro di rilocazione consente al SO di cambiar dinamicamente le proprie dimensioni: se per esempio il SO, contiene codice e spazio di memoria per i driver dei dispositivi e nessuno di questi viene quasi mai usato, è inutile tenere in memoria codice e dati!

Pertanto, lo spazio potrebbe essere usato per altri scopi. Questo codice si chiama transiente del SO perché si inserisce secondo le necessità; l’uso di questo codice cambia le dimensioni del SO durante l’esecuzione del programma.

**8.3.2 Allocazione della memoria**

Uno dei metodi più semplici per l’allocazione della memoria consiste nel suddividere la stessa in partizioni di dimensione fissa. Ogni partizione deve contenere esattamente un processo quindi il grado di multiprogrammazione è limitato dal numero di partizioni.

Con il metodo delle partizioni multiple quando una partizione è libera può esser occupata da un processo presente nella coda d'ingresso; terminato il processo la partizione diviene nuovamente disponibile per un altro processo.

Nello schema a partizione fissa il SO conserva una tabella in cui sono indicate le porzioni di memoria disponibili e quelle occupate. Inizialmente tutta la memoria è a disposizione dei processi utenti, si tratta di un grande blocco di memoria disponibile, un buco.

Quando entrano nel sistema, i processi vengono inseriti in una coda d’ingresso. Per determinare a quali processi si debba assegnare la memoria, il sistema operativo tiene conto dei requisiti di memoria di ciascun processo e della quantità di spazio di memoria disponibile.

In ogni dato istante è sempre disponibile una lista delle dimensioni dei blocchi liberi e della coda d’ingresso. Il sistema operativo può ordinare la coda d’ingresso secondo un algoritmo di scheduling. La memoria si assegna ai processi della coda finché si possono soddisfare i requisiti di memoria del processo successivo, cioè finché esiste un blocco di memoria (o buco) disponibile, sufficientemente grande da accogliere quel processo.

Il sistema operativo può quindi attendere che si renda disponibile un blocco sufficientemente grande, oppure può scorrere la coda d’ingresso per verificare se sia possibile soddisfare le richieste di memoria più limitate di qualche altro processo.

In generale, è sempre presente un *insieme* di buchi di diverse dimensioni sparsi per la memoria. Quando si presenta un processo che necessita di memoria, il sistema cerca nel gruppo un buco di dimensioni sufficienti per contenerlo. Se è troppo grande, il buco viene diviso in due parti: si assegna una parte al processo in arrivo e si riporta l’altra nell’insieme dei buchi. Quando termina, un processo rilascia il blocco di memoria, che si reinserisce nell’insieme dei buchi; se si trova accanto ad altri buchi, si uniscono tutti i buchi adiacenti per formarne uno più grande. A questo punto il sistema deve controllare se vi siano processi nell’attesa di spazio di memoria, e se la memoria appena liberata e ricombinata possa soddisfare le richieste di qualcuno fra tali processi. Questa procedura è una particolare istanza del più generale problema **di allocazione dinamica della memoria**, che consiste nel soddisfare una richiesta di dimensione n data una lista libera di buchi.

I criteri più usati per scegliere un buco libero tra quelli disponibili sono i seguenti:

* **First-fit :** si assegna il *primo* buco abbastanza grande. La ricerca può cominciare sia dall’inizio

dell'insieme di buchi sia dal punto in cui era terminata la ricerca precedente. Si può fermare la ricerca non appena s’individua un buco libero di dimensioni sufficientemente grandi.

* **Best-fit :**si assegna il *più piccolo* buco in grado di contenere il processo. Si deve compiere la ricerca in tutta la lista, sempre che questa non sia ordinata per dimensione. Tale criterio produce le parti di buco inutilizzate piu piccole.
* **Worst-fit** : si assegna il buco *più grande.* Anche in questo caso si deve esaminare tutta la lista, sempre che non sia ordinata per dimensione. Tale criterio produce le parti di buco inutilizzate più grandi, che possono essere più utili delle parti più piccole ottenute col criterio precedente.

Con l'uso di simulazioni si è dimostrato che sia first-fit sia best-fit sono migliori rispetto a worst-fit in termini di risparmio di tempo e di utilizzo di memoria. In generale first-fit è più veloce.

**8.3.3 Frammentazione**

Entrambi i criteri first-fit e best-fit soffrono della **frammentazione esterna**: quando si caricano e si rimuovono i processi dalla memoria, si frammenta lo spazio libero della memoria in tante piccole parti.

-Si ha la **frammentazione esterna** se lo spazio di memoria totale è sufficiente per soddisfare una richiesta, ma non è contiguo; la memoria è frammentata in tanti piccoli buchi. Questo problema di frammentazione può essere molto grave; nel caso peggiore può verificarsi un blocco di memoria libera, sprecata, tra ogni coppia di processi. Se tutti questi piccoli pezzi di memoria costituissero in un unico blocco libero di grandi dimensioni, si potrebbero eseguire molti più processi.

Sia che si adotti l’uno o l’altro, l’impiego di un determinato criterio può influire sulla quantità di frammentazione.

**Esempio:** La frammentazione può essere sia interna sia esterna. Si consideri il metodo d’allocazione con più partizioni con un buco di 18.464 byte. Supponendo che il processo successivo richieda 18.462 byte, assegnando esattamente il blocco richiesto rimane un buco di 2 byte. Il carico necessario per tener traccia di questo buco è sostanzialmente più grande del buco stesso. Il metodo generale prevede di suddividere la memoria fisica in blocchi di dimensione fissa, che costituiscono le unità d’allocazione. Con questo metodo la memoria assegnata può essere leggermente maggiore della memoria richiesta.

-la **frammentazione interna** consiste nella differenza tra la memoria assegnata e la memoria richiesta: la memoria è interna a una partizione ma non è in uso.

Una soluzione al problema della **frammentazione esterna** è data dalla **compattazione**.

Lo scopo è quello di riordinare il contenuto della memoria per riunire la memoria libera in un unico grosso blocco. La compattazione, tuttavia, non è sempre possibile: è possibile solo se la rilocazione è dinamica e si compie nella fase d’esecuzione.

Quando è possibile eseguire la compattazione, è necessario determinarne il costo. Il più semplice algoritmo di compattazione consiste nello spostare tutti i processi verso un’estremità della memoria, mentre tutti i buchi vengono spostati nell’altra direzione formando un grosso buco di memoria. Questo metodo può essere assai oneroso.

**Un’altra possibile soluzione** del problema della **frammentazione esterna** è data dal consentire la non contiguità dello spazio degli indirizzi logici di un processo, permettendo così di assegnare la memoria fisica ai processi dovunque essa sia disponibile. Due tecniche complementari conseguono questo risultato: la **paginazione** e la segmentazione

**8.4 PAGINAZIONE**

La paginazione è un metodo di gestione della memoria che permette che lo spazio degli indirizzi fisici di un processo non sia contiguo. Elimina il gravoso problema della sistemazione di blocchi di memoria di diverse dimensioni in memoria ausiliaria.

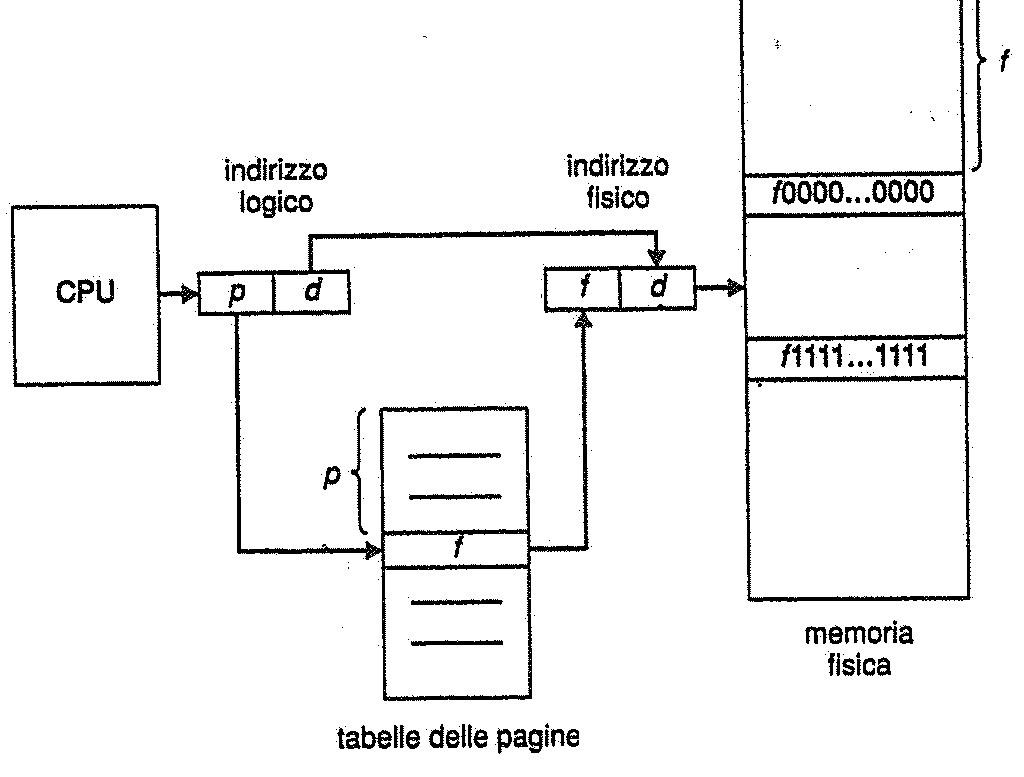
Il problema insorge perché quando alcuni frammenti di codice o dati residenti in memoria centrale devono essere scaricati, si deve trovare lo spazio necessario in memoria ausiliaria.

I problemi di frammentazione relativi alla memoria centrale valgono anche per la memoria secondaria(ausiliaria) con la differenza che in questo caso l’accesso è molto più lento e quindi è impossibile eseguire la compattazione.

**8.4.1 Metodo di base**

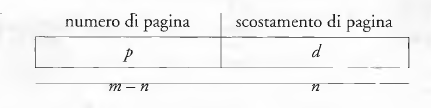
Il metodo di base per implementare la paginazione consiste nel suddividere la memoria fisica in blocchi di dimensione costante (detti anche frame o pagine fisiche) e nel suddividere la memoria logica in blocchi di pari dimensioni (pagine).

Quando si deve eseguire un processo si caricano le sue pagine nei frame disponibili prendendole dalla memoria ausiliaria, divisa in blocchi di dimensione fissa, uguale a quella dei frame della memoria.



Ogni indirizzo generato dalla CPU è diviso in due parti: numero di pagina (p) e uno scostamento (Offset) di pagina (d). Il numero di pagina serve come indice per la tabella delle pagine contenente l’indirizzo base in memoria fisica di ogni pagina. Questo indirizzo si combina con lo scostamento di pagina (d) per definire l’indirizzo della memoria fisica, che si invia all'unità di memoria.

La dimensione di una pagina, così come quella di un frame, è definita dall’architettura e varia tra i 512 byte a 16 MB. Se la dimensione dello spazio degli indirizzi logici è 2^m e la dimensione di una pagina è di 2^n unità di indirizzamento (byte o parole), allora gli m-n bit più significativi Di un indirizzo logico indicano il numero di pagina, Ehi n bit meno significativi indicano lo scostamento di pagina.



Dove ***p*** è un indice della tabella delle pagine e ***d*** è lo scostamento all’interno della pagina indicata da p.

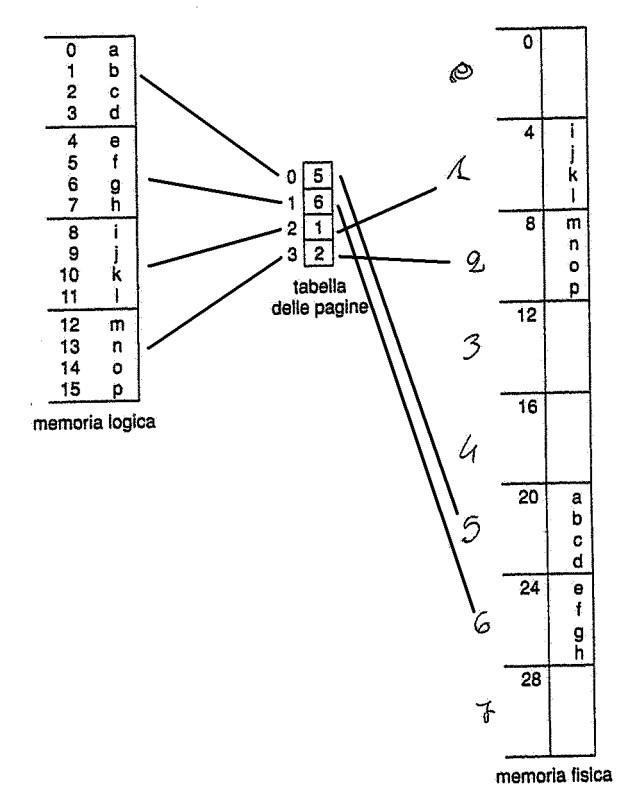
La paginazione non è altro che una forma di rilocazione dinamica: a ogni indirizzo logico l’architettura di paginazione fa corrispondere un indirizzo fisico. L’uso della tabella delle pagine è simile all’uso di una tabella di registri base uno per ciascun frame.

Con la paginazione si può evitare la frammentazione esterna: qualsiasi frame libero si può assegnare a un processo che ne abbia bisogno. Però c’è il problema della frammentazione interna nel caso peggiore in cui si ha un processo che necessita di n pagine + 1 byte e si devono allocare n+1 pagine con una frammentazione interna media di mezza pagina per processo. Questo fa si che si convenga usare pagine di piccole dimensioni.

Attualmente la dimensione tipica delle pagine è compresa tra 4kb e 8kb: in alcuni sistemi può essere anche maggiore; alcune CPU e Kernel di SO gestiscono pagine di dimensioni diverse.

Quando si deve eseguire un processo, si esamina la sua dimensione espressa in pagine.

Poiché ogni pagina del processo necessita di un frame, se il processo richiede *n* pagine, devono essere disponibili almeno *n* frame che, se ci sono, si assegnano al processo stesso. Si carica la prima pagina del processo in uno dei frame assegnati e s’inserisce il numero del frame nella tabella delle pagine relativa al processo in questione.



Un aspetto importante della paginazione èla netta distinzione tra la memoria vista dall’utente

e l’effettiva memoria fisica: il programma utente *vede* la memoria come un unico spazio contiguo, contenente solo il programma stesso; in realtà, il programma utente èsparso in una memoria fisica contenente anche altri programmi.

La differenza tra la memoria vista dall’utente e la memoria fisica è colmata dall’architettura di traduzione degli indirizzi, che fa corrispondere gli indirizzi fisici agli indirizzi logici generati dai processi utenti. Queste trasformazioni non sono visibili agli utenti e sono controllate dal sistema operativo.

Un processo utente, per definizione, non può accedere alle zone di memoria che non gli appartengono.

Non ha modo di accedere alla memoria oltre a quello che è previsto dalla sua tabella delle pagine. Tale tabella riguarda solo soltanto le pagine che appartengono al processo.

Siccome il SO gestisce la memoria fisica, deve essere informato dei relativi particolari di allocazione: quali frame sono assegnati, quali sono disponibili, il loro numero totale e così via. In genere queste informazioni sono contenute in una struttura dati chiamata **tabella dei frame** contenenti un elemento per ogni frame, Indicante se sia libero oppure è segnato e, se è segnato, a quale pagina di quale processo o di quali processi.

Inoltre, il sistema operativo deve sapere che i processi utenti operino nello spazio utente, e tutti gli indirizzi logici si devono far corrispondere a indirizzi fisici. Il sistema operativo conserva una copia della tabella delle pagine per ciascun processo, così come conserva una copia dei valori contenuti nel contatore di programma e nei registri. Questa copia si usa per tradurre gli indirizzi logici in indirizzi fisici ogni volta che il sistema operativo deve associare esplicitamente un indirizzo fisico un indirizzo logico. La stessa coppia è usata anche dal dispatcher della CPU per impostare l'architettura di paginazione quando un processo sta per essere assegnato la CPU. La paginazione, quindi, fa aumentare la durata dei cambi di contesto.

**8.4.2 Architettura di paginazione**

Ogni SO segue metodi propri per memorizzare le tabelle delle pagine. La maggior parte dei sistemi impiega una tabella delle pagine per ciascun processo. Il PCB, oltre a contenere le informazioni del processo, contiene un puntatore della tabella delle pagine. Per avviare un processo, il dispatcher ricariche registra utente è imposta i corretti valori della tabella delle pagine fisiche, usando la tabella delle pagine presenti in memoria e relativa al processo.

L’architettura d’ausilio alla tabella delle pagine può essere realizzata in modi diversi, nel caso più semplice si usa uno specifico insieme di registri. Per garantire un’efficiente traduzione degli indirizzi di paginazione, questi registri devono essere costruiti in modo da operare a una velocità molto elevata. Tale efficienza è determinante, poiché ogni accesso alla memoria passa attraverso il sistema di paginazione.

Le istruzioni di caricamento e modifica dei registri della tabella delle pagine sono privilegiate, quindi soltanto il sistema operativo può modificare la mappa della memoria.

L’uso di registri per la tabella delle pagine è efficiente se la tabella stessa è ragionevolmente

piccola, nell’ordine, per esempio, di 256 elementi. La maggior parte dei calcolatori contemporanei usa comunque tabelle molto grandi, quindi non si possono impiegare i registri veloci per realizzare la tabella delle pagine; quest’ultima si mantiene in memoria principale e un **registro di base della tabella delle pagine** (*page-table base register*, PTBR) punta alla tabella stessa. Il cambio delle tabelle delle pagine richiede soltanto di modificare questo registro, riducendo considerevolmente il tempo dei cambi di contesto.

Questo metodo presenta un problema connesso al tempo necessario di accesso a una locazione della memoria utente. Per accedere alla locazione *i*, occorre far riferimento alla tabella

delle pagine usando il valore contenuto nel PTBR aumentato del numero di pagina relativo a i, perciò si deve accedere alla memoria. Si ottiene il numero del frame che, associato allo scostamento di pagina, produce l’indirizzo cercato; a questo punto è possibile accedere alla posizione di memoria desiderata. 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), quindi l’accesso alla memoria è rallentato di un fattore 2. Nella maggior parte dei casi un tale ritardo e intollerabile;

La soluzione tipica a questo problema consiste nell’impiego di una speciale, piccola cache di ricerca veloce, detta TLB *(translation look-aside buffer)*.

La TLB è una memoria associativa ad alta velocità in cui ogni suo elemento consiste di due parti: una chiave, o un indicatore*(tag)* e un valore. Quando si presenta un elemento, la memoria associativa lo confronta

contemporaneamente con tutte le chiavi; se trova una corrispondenza, riporta il valore correlato.

La ricerca è molto rapida, ma le memorie associative sono molto costose. Il numero degli elementi in una TLB è piccolo, spesso e compreso tra 64 e 1024.

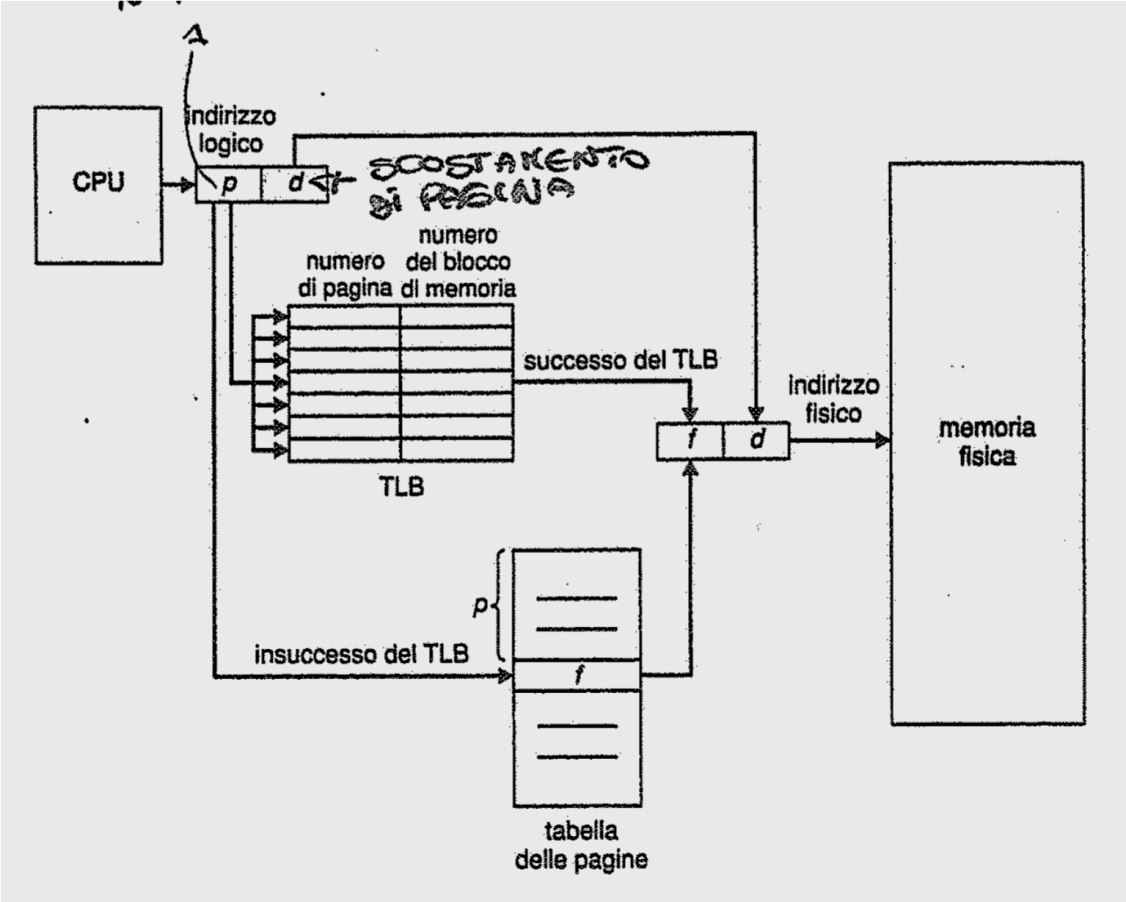
La TLB si usa insieme con le tabelle delle pagine nel modo seguente: la TLB contiene una piccola parte degli elementi della tabella delle pagine; quando la CPU genera un indirizzo logico, si presenta il suo numero di pagina alla TLB; se tale numero è presente, il corrispondente numero del frame è immediatamente disponibile e si usa per accedere alla memoria.

Se nella TLB non è presente il numero di pagina, situazione nota come **insuccesso della TLB *(TLB miss),*** si deve consultare la tabella delle pagine in memoria. Il numero del frame così ottenutosi può eventualmente usare per accedere alla memoria. Inoltre, inserendoi numeri della pagina e del frame nella TLB, al riferimento successivo la ricerca sarà molto più rapida.

Se la TLB è già piena d’elementi, il sistema operativo deve sceglierne uno per sostituirlo. I criteri di sostituzione variano dalla scelta dell’elemento usato meno recentemente (LRU) alla scelta casuale. Inoltre, alcune TLB consentono che certi elementi siano **vincolati,** ***cioè*** non si possano rimuovere dalla TLB; in genere si vincolano gli elementi per il codice del kernel.

Alcune TLB memorizzano gli identificatori dello spazio d’indirizzi (ASID) in ciascun elemento della TLB. Un’ASID identifica in modo univoco ciascun processo e si usa per fornire al processo la protezione del suo spazio di indirizzi.

La mancata corrispondenza dell’ASID viene trattata come un insuccesso della TLB. inoltre, per fornire la protezione dello spazio di indirizzi, l’ASID consente che la TLB contenga nello stesso istante elementi di diversi processi. La percentuale di volte con numero di pagina si trova nella TLB e detta tasso di successo **(hit ratio)**.

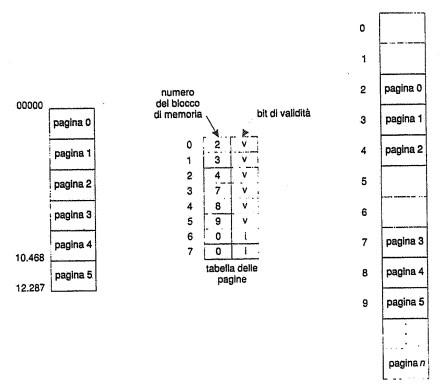


**8.4.3 Protezione**

In un ambiente paginato la protezione della memoria è assicurata dai bit di protezione associati a ogni frame. Normalmente tali bit si trovano nella tabella delle pagine. Un bit può determinare se una pagina si può leggere e scrivere oppure solo leggere.

Tutti i riferimenti alla memoria passano attraverso la tabella delle pagine per trovare il numero corretto del frame; quindi, mentre si calcola l'indirizzo fisico, si possono controllare bit di protezione per verificare che non si scrive una pagina di sola lettura. questo metodo si può facilmente estendere per fornire un livello di protezione maggiore. Si può progettare un’architettura che fornisca la protezione di sola lettura, di sola scrittura o di sola esecuzione. In alternativa con bit di protezione distinti per ogni tipo d’accesso, si può ottenere una qualsiasi combinazione di tali tipi d’accesso, i tentativi illegali causano l’invio di un segnale di eccezione al SO (tipo se si vuole scrivere in una pagina di sola lettura). Di solito si associa a ciascun elemento della tabella delle pagine un ulteriore bit, detto bit di validità. Tale bit se impostato a *valido*, indica che la pagina corrispondente è nello spazio degli indirizzi logici del processo, quindi è una pagina valida; impostato a *non valido,* indica che la pagina non è nello spazio degli indirizzi logici del processo. Quindi tale bit consente di riconoscere gli indirizzi illegali e di notificarne la presenza attraverso le eccezioni. Tutto questo è gestito dal SO che decide come impostare tale bit.

Alcune architetture dispongono di registri, detti registri di **lunghezza della tabella delle pagine** (PTRL) per indicare le dimensioni della tabella.



**8.4.4 Pagine condivise**

Un altro vantaggio della paginazione consiste nella possibilità di condividere codice comune. Questa considerazione è importante soprattutto in un ambiente a partizione di tempo.

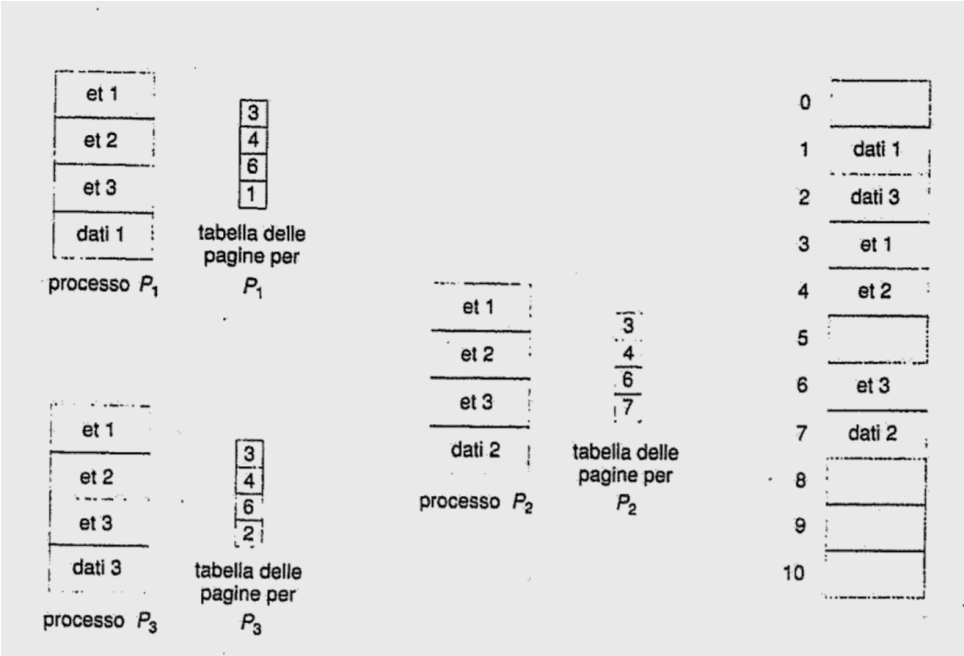
**Esempio:** Si consideri un sistema con 40 utenti, ciascuno dei quali usa un elaboratore di testi. Se tale

programma e formato da 150 KB di codice e 50 KB di spazio di dati, per gestire i 40 utenti sono necessari 8000 KB. Se però il codice è **rientrante**, può essere condiviso.

In memoria fisica è presente una sola copia dell’elaboratore di testi: la tabella delle pagine di ogni utente fa corrispondere gli stessi frame contenenti l’elaboratore di testi, mentre le pagine dei dati si fanno corrispondere a frame diversi. Quindi per gestire 40 utenti sono sufficienti una copia dell’elaboratore di testi (150 KB) e 40 copie dei 50 KB di spazio di dati per ciascun utente; per un totale di 2150 KB, invece di 8000 KB; il risparmio è notevole.

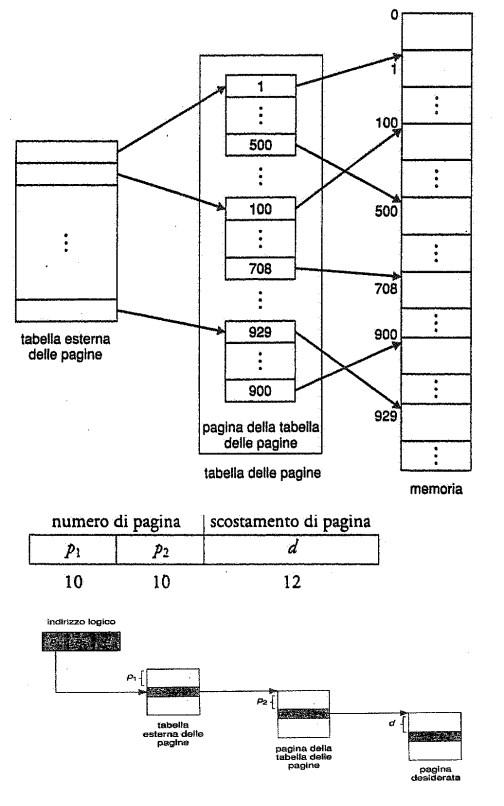
Il codice rientrante**\***, detto codice puro, è un codice non automodificante, non cambia durante l’esecuzione. Quindi due o più processi possono eseguire lo stesso codice nello stesso momento. Ciascun processo dispone di una propria copia dei registri ed una memoria dove conserva i dati necessari alla propria esecuzione punto i dati per due differenti processi variano, ovviamente, per ciascun processo.

**\*Rientrante:** condizione necessaria per la condivisione;



## 8.5 STRUTTURA DELLA TABELLA DELLE PAGINE

## 8.5.1 Paginazione gerarchica

La maggior parte dei moderni calcolatori dispone di uno spazio d’indirizzi logici molto grande (da 2^32 a 2^64 elementi). Chiaramente sarebbe meglio evitare di collocare la tabella delle pagine in modo contiguo in memoria centrale. Una semplice soluzione a questo problema consiste nel suddividere la tabella delle pagine in parti più piccole; questo risultato si può ottenere in molti modi. Un metodo consiste nell’adottare un algoritmo di paginazione a due livelli, in cui la tabella stessa è paginata. 

//esempio sul libro pag 322

**Paginazione gerarchica slide prof**

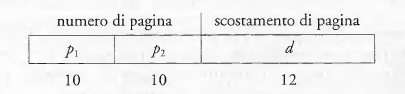
Il metodo consiste nell’adottare uno schema di paginazione più livelli, nel quale la stessa tabella delle pagine viene paginata. Un indirizzo logico (su una macchina a 32 bit compagine di 4K ) viene diviso in:

* Un numero di pagina di 20 bit
* Uno scostamento di pagina di 12 bit.

Giacché la tabella delle pagine e paginata, il numero di pagine e ulteriormente diviso in:

* Un numero di pagina di 10 bit
* Uno scostamento di pagina di 10 bit.

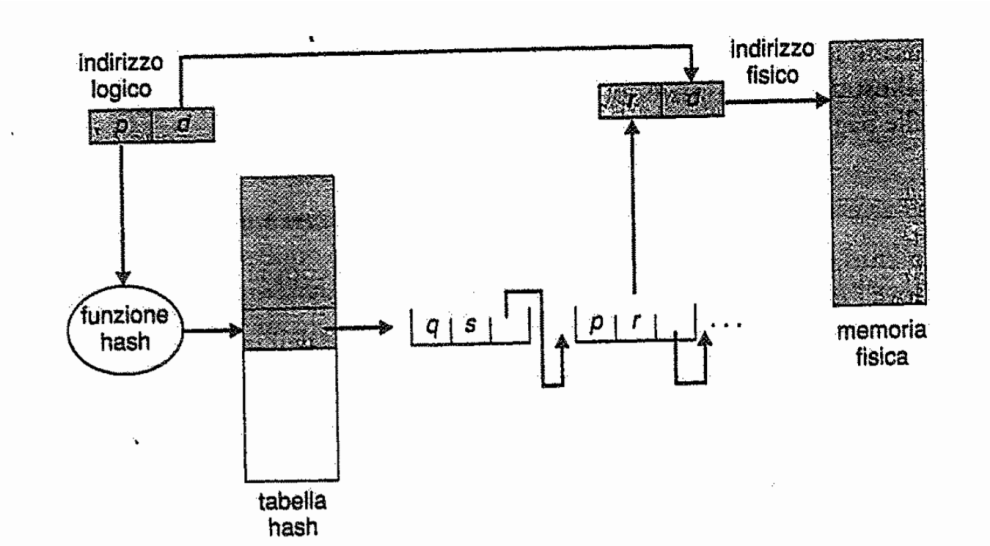
Quindi un indirizzo logico può essere così schematizzato:



## 8.5.2 Tabella delle pagine di tipo hash

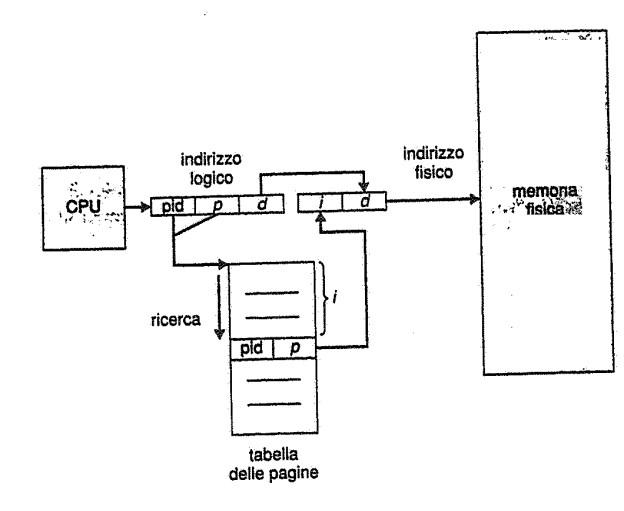
Un metodo di gestione molto comune degli spazi d’indirizzi relativi ad architetture oltre 32 bit consiste nell’impiego di una tabella delle pagine di tipo hash, in cui l’argomento della funzione hash è il numero della pagina virtuale.

Ciascun elemento è composto da 3 campi: il numero della pagina virtuale, l’indirizzo del frame corrispondente alla pagina virtuale, un puntatore al successivo elemento. Si applica la funzione hash al numero della pagina virtuale contenuto nell’indirizzo virtuale, identificando un elemento della tabella. Si confronta il numero di pagina con il primo campo degli elementi della lista e se coincidono si usa l’indirizzo relativo al frame per generare l’indirizzo fisico desiderato. altrimenti, l'algoritmo esami nello stesso modo gli elementi successivi della lista concatenata. Le tabelle delle pagine di tipo esce sono particolarmente utili degli spazi l'indirizzo sparsi virgola in cui riferimenti alla memoria non sono contigui ma distribuiti per tutto lo spazio di indirizzi.



## 8.5.3 Tabella delle pagine invertita

Generalmente si associa una tabella delle pagine a ogni processo e tale tabella contiene un elemento per ogni pagina virtuale che il processo sta utilizzando, oppure un elemento per ogni indirizzo virtuale a prescindere dalla validità di quest’ultimo. Poiché la tabella è ordinata per indirizzi virtuali il sistema operativo può calcolare in che punto della tabella si trova l’elemento dell’indirizzo fisico associato e usare direttamente tale valore. Uno degli inconvenienti insiti in questo metodo è costituito dalla dimensione di ciascuna tabella delle pagine, che può contenere milioni di elementi e occupare grandi quantità di memoria fisica, necessaria proprio per sapere com’è impiegata la rimanente memoria fisica. Per risolvere questo problema si può fare uso della tabella delle pagine invertita. Una tabella delle pagine invertita ha un elemento per ogni pagina reale o frame. Ciascun elemento è quindi costituito dell’indirizzo virtuale della pagina memorizzata in quella reale locazione di memoria con informazioni sul processo che possiede tale pagina. Ciascun indirizzo virtuale è una tripla del tipo seguente: <id-progesso, numero pagina, scostamento>. Viene ricercato tramite il pid il valore nella tabella delle pagine se viene trovato lo scostamento più il valore il numero i.



Sebbene riduca la quantità di memoria necessaria per memorizzar ogni tabella delle pagine, aumenta però il tempo di ricerca nella tabella. Poiché la tabella delle pagine invertita è ordinata per indirizzi fisici, mentre la ricerca si effettua per indirizzi virtuali, si deve effettuare la ricerca sull’intera tabella. Per limitare il problema si può usare una tabella hash. Naturalmente, ogni accesso alla tabella hash aggiungi al procedimento un riferimento alla memoria, quindi un riferimento alla memoria virtuale richiede almeno due letture della memoria reale: una per l'elemento della tabella hash e l'altro per la tabella delle pagine. Per migliorare le prestazioni, la ricerca si effettua prima nella TLB, quindi si Consulta la tabella hash. nei sistemi che adottano le tabelle delle pagine invertite, l'implementazione della memoria condivisa e difficoltosa. Difatti, la condivisione si realizza solitamente tramite indirizzi virtuali multipli associati a un unico indirizzo fisico. Il metodo però e inutilizzabili in presenza di tabella invertite, perché, essendo un solo elemento indicante la pagina virtuale corrispondente ogni pagina fisica, questa non può avere più di un indirizzo virtuale associata appunto una semplice tecnica per superare il problema consiste nel porre nella tabella delle pagine una sola associazione fra un indirizzo virtuale e l'indirizzo fisico condiviso; ciò comporta un errore dovuto all’assenza della pagina per ogni riferimento agli indirizzi virtuali non associati *(page fault).*