Memoria virtuale

> ⚠ **Problema**
>
> La grandezza del software cresce troppo velocemente rispetto alla grandezza delle memorie, quindi è necessario poter gestire programmi più grandi della memoria stessa.

La prima soluzione, sviluppata negli anni '60 ci chiama **overlay**, semplicemente il programma veniva suddiviso dal programmatore su livelli diversi che venivano caricati uno sopra l'altro in memoria in maniera sequenziale, sovrascrivendo quello precedente.

L'overlay era noioso e complicato da fare, quindi venne creata una nuova soluzione nota come **memoria virtuale**.

> ⓘ **Memoria VIrtuale**
>
> Ogni Programma ha uno spazio di indirizzi suddiviso in pagine, una **pagina** è uno spazio di indirizzi contigui, sono mappate sulla memoria fisica, ma per eseguire il programma non abbiamo bisogno di mantenerle tutte nella memoria fisica.
> Se il programma fà riferimento ad una parte del suo spazio di indirizzi che è nelle memoria fisica, l'hardware esegue la mappatura necessaria, altrimenti il SO viene allertato e va a prelevare la parte mancante per poi eseguire nuovamente il programma.
> La parte mancante chee andiamo a prelevare è una pagina.

Paginazione

La maggior parte dei sistemi con memoria virtuale usano una tecninca chiamata **paginazione**.

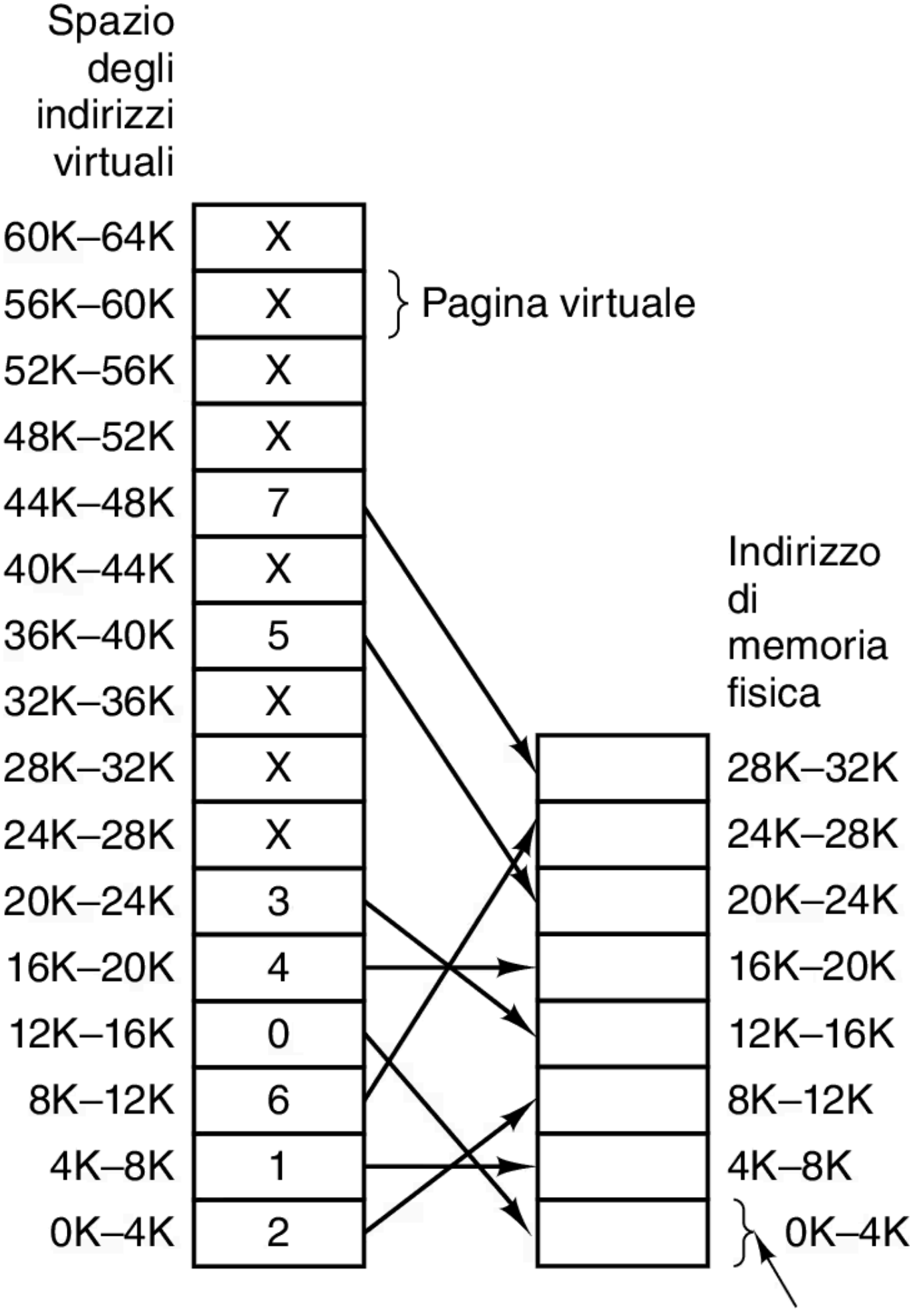
Gli indirizzi generati da un programma sono **indirizzi virtuali** e formano uno **spazio degli indirizzi virtuali.**

- Computer senza memoria virtuale -> Indirizzi posti sul bus di memoria che provoca la lettura o scrittura dell'indirizzo fisico.
- Computer con meoria virtuale -> Gli indirizzi prima di andare sul bus passano per la **MMU**.

> ⓘ **MMU (Memory Managment Unit)**

Mappa gli indirizzi virtuali sugli indirizzi della memoria fisica.

Con la paginazione preticamente carichiamo nella memoria solo una parte del programma, poiché nella sua interezza non potrbbe entrarci.


Spazio degli indirizzi virtuali
60K–64K
56K–60K
52K–56K
48K–52K
44K–48K
40K–44K
36K–40K
32K–36K
28K–32K
24K–28K
20K–24K
16K–20K
12K–16K
8K–12K
4K–8K
0K–4K
X
X
X
X
7
X
5
X
X
X
3
4
0
6
1
2
Pagina virtuale
Indirizzo di memoria fisica
28K–32K
24K–28K
20K–24K
16K–20K
12K–16K
8K–12K
4K–8K
0K–4K

Frame

Lo spazio degli indirizzi virtuali è suddiviso in unità di dimensione fissa, cioè le **pagine**.
Le unità corrispondenti nella memoria fisica sono i **frame**.

> **Funzionamento**
>
> La MMU riceve la pagina da cui prelevare l'istruzione:
> `MOV REG, 0`
> L'indirizzo virtuale cade nella pagina 0, cioè da 0 a 4095 --> il frame 2(8k a 12k).
> La MMU mappa quindi gli indirizzi virtuali fra 0 e 4095 su 8192 a 12287.
> Quindi quando la memoria riceve una richiesta di lettura/scrittura vedrà 8192.

Il problema nasce quando la pagina nello spazio degli indirizzi virtuali non è mappata.
In uqesto caso viene causata una **trap** e ciene chiamato il **page fault**.

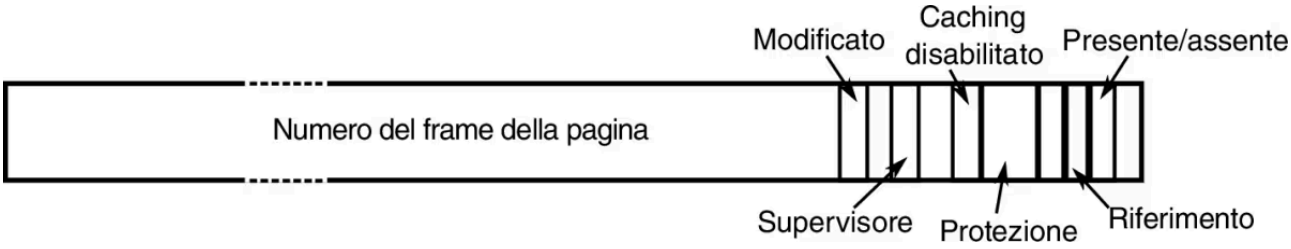
> **Page Fault**
>
> Il SO prelevare un frame poco usato e ci sovrascrive la pagina che è stata appena referenziata, poi riavvia l'istruzione che era in trap.

> **Funzionamento della MMU**
>
> L'indirizzo virtuale in ingresso (16 bit) è suddiciso in un numero di pagina di 4 bit e un offset di 12 bit.
> Con 4 bit per il numero di pagina possiamo avere 16 pagine e con un offset di 12 bit possiamo indirizzare 4096 byte per pagine.
> Il numero di pagine è usato come indice nella **tabella delle pagine**, che porta al numero del frame corrispondente alla pagina virtuale.
> Nei 4 bit il meno significativo è usato come bit di **assenza/presenza**, se è 0 avviene una trap al SO, se è 1 il numero di frame trovato nella tabella delle pagine viene copiato nei 3 bit più significativi del registro di output, insieme ai 12 bit di offset.
> Insieme compongono un indirizzo di 15 bit che è posto sul bus di memoria come indirizzo fisico.

Tabelle delle pagine

La struttura di una delle voci all'interno delle tabelle è composta da diverse informazioni:

- **Il numero del frame**
- **Bit presenza/assenza** -> Indica se la pagina è in memoria
- **Bit supervisor** -> Specifica i tipi di accessi consentiti
- **Bit Modificato (M) e Riferimento (R)** -> Registrano l'uso della pagina, il primo si attiva quando la pagina viene scritta, il secondo ogni volta che si accede alla pagina.



Velocizzare la paginazione

Ogni sistema di paginazione abbiamo due problemi principali:

- **La mappatura dall'indirizzo virtuale all'indirizzo fisico deve essere veloce** -> AVviene ad ogni riferiemento in memoria, potrebbe diventare un collo di bottiglia.
- **La tabella delle pagine non deve essere troppo grande** -> Tutti i computer moderni usano al massimo 48 bit per l'indirizzamento delle pagine, se usassimo 4kb per pagina occuperemo 100Gb di memoria solo per la tabella.

Abbiamo diversi schemi per gestire tutto questo:

- Unica tabella delle pagine che consiste in un array di registri hardware veloci, registro hardware per ogni pagina, caricato all'avvio del processo.

> ⚠ **Svantaggio**
>
> Costoso con tabelle delle pagine grandi, ricaricare ll'intera tabella ad ogni scambio di contesto è inefficiente.

- Inseriamo le tabelle delle pagine nella memoria principale, adesso abbiamo bisogno di un solo registro che punti all'inizio della tabella.

> ⚠ **Svantaggio**
>
> Richiede accessi frequenti alla memoria, rendendo la mappatura lenta.

Translation Lookaside Buffer (TLB)

Questi svantaggi rendono l'intero processo molto lento, quello che possiamo fare è aggiungere una componente hardware che rende l'indirizzo fisico un indirizzo virtuale e si trova all'interno della CPU.

Praticamente ci permette di mappare l'indirizzo delle pagine senza passare per la tabella e quindi ridurre gli accessi alla memoria durante la paginazione.

Questa componente si chiama **Translation Lookaside Buffer** -> è una cache.
Quando la CPU vuole tradurre un indirizzo fisico chiede l'indirizzo virtuale al TLB:

- Se ci sta l'indirizzo all'interno della TLB -> Veloce (Meno di un ciclo di clock).
- Se non ci sta l'indirizzo all'interno della TLB -> Carichiamo l'indirizzo dalla tabella delle pagine. (lento).

Le CPU moderne hanno solitamente due TLB per:

- **Istruzioni**
- **Dati**

Vengono costantemente aggiornate e sono **molto piccole** (Contengono poche voci con: Pagina virtuale, bit modificato, codice di protezione e frame fisico).

> **Funzionamento**
>
> 1. Richiesta indirizzo virtuale da parte della CPU
> 2. MMU controlla nel TLB
> 3. Se trovato, il frame è prelevato direttamente dal TLB
> 4. Altrimenti ricerca normale nella tabella della pagina che poi rimpiazza un'altra voce all'interno della TLB

Su alcune CPU RISC si possono gestire le pagine tramite software, su queste macchine le voci del TLB sono caricate dal SO.
In caso di TLB miss, il SO prende il controllo che sostituisce la pagina e riavvia l'istruzione.

- Page Fault -> Rari
- TLB miss -> Comuni perché è una memoria piccola.

Per risolvere il problemas non possiamo aumentare le dimensione del TLB per due motivi:

- Costoso
- Occuperebbe spazio per altri elementi

Tramite la gestione software dei TLB otteniamo un'efficienza accettabile mantenendo la semplicità del TLB e della MMU, lasciando molto spazio libero sulla CPU.
Esistono due tipi di miss:

- **Soft miss** -> La pagina di riferimento non è nel TLB, ma in memoria, non serve nessun aiuto esterno, per risovere ci voglio 10-20 istruzioni.
- **Hard miss** -> La pagina non è in memoria, dobbiamo accedere al disco per prelevare la pagina, è un'operazione **milioni di volte** più lenta rispetto al soft miss.

Cercare la mappatura nella gerarchia delle tabelle è un' operazione che prende il nome di **page table walk**

Non tutti i miss sono uguali, alcuni sono più leggeri/gravi di altri, infatti possiamo avere un:

- **minor page fault** -> La pagina è in memoria, ma non nella tabella delle pagine del processo, quindi accediamo alla emmoria e facciamo la mappatura.
- **major page fault** -> La pagina non è in memoria e dev'essere caricata dalla memoria non volatile.
- **Segmentatio fault** -> Tutto il programma si comporta in modo sbagliato, perché sta provando ad accedere ad una parte della memoria che non gli è destinata.

Page Table Sizes

Un problema della tabella delle pagine è che potrebbe occupare molto spazio all'interno della memoria, infatti uno spazio degli indirizzi molto grande porterebbe a una tabella di pagine molto grande.

Una delle possibili soluzioni è il **multi-level page table**

> ⓘ **Multi-Level Page Table**
>
> Tecnica utilizzata nella gestione della memoria virtuale per ridurre le dimensioni delle tabelle delle pagine in sistemi con spazi di indirizzamento virtuale molto grandi.

Invece di avere un'unica grande tabella, possiamo **dividere l'indirizzo virtuale in più parti**, ciascuna delle quali punta a una tabella di livello inferiore. Ogni tabella contiene riferimenti ad altre tabelle finché non si raggiunge l'indirizzo fisico della pagina.

Ad esempio, in un sistema a 32 bit con pagine da 4 KB, un indirizzo virtuale può essere diviso in più segmenti:

- Un primo segmento può indicare la **Page Global Directory (PGD)**.

- Il secondo segmento può puntare alla **Page Upper Directory (PUD)**.
- E così via, fino ad arrivare alla **Page Table Entry (PTE)**, che contiene l'indirizzo fisico della pagina.