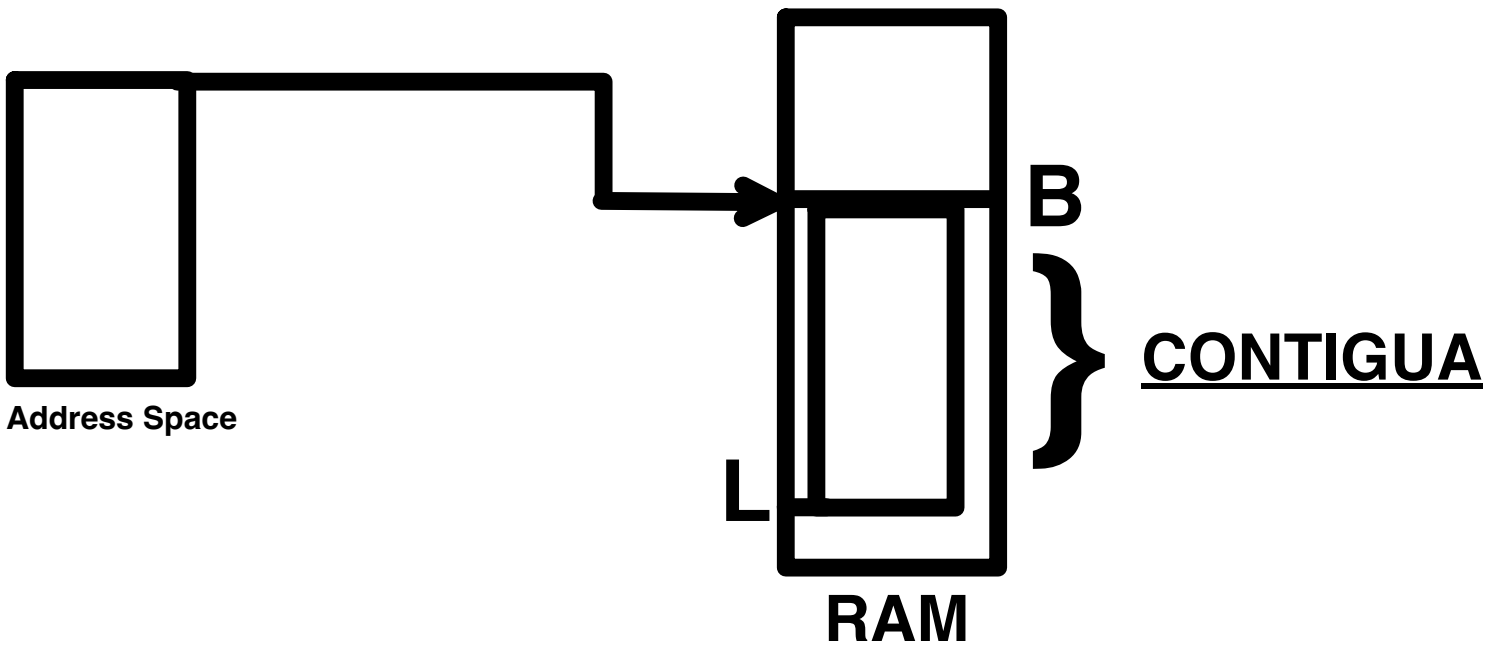
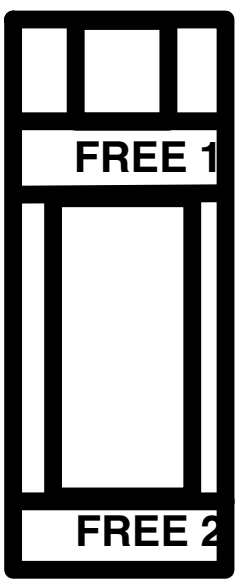
Paginazione

01:26:19

Stiamo parlando di uno schema della gestione della memoria, in particolare di "collocazione" di un address space logico all'interno della memoria di lavoro.
Questo è lo schema che stiamo utilizzando ora in tutte le tecnologie moderne.
Questo schema prevede di risolvere uno dei problemi fondamentali che noi avevamo con gli schemi sin ora presentati, il problema è il seguente:
Quando noi abbiamo uno schema basato su "partizioni" (siano esse statiche, siano esse dinamiche) quello che facciamo è la seguente cosa. Abbiamo un address space, data la mia RAM, il mio address space veniva ad essere caricato a partire da un punto, e in maniera CONTIGUA fino ad un limite.

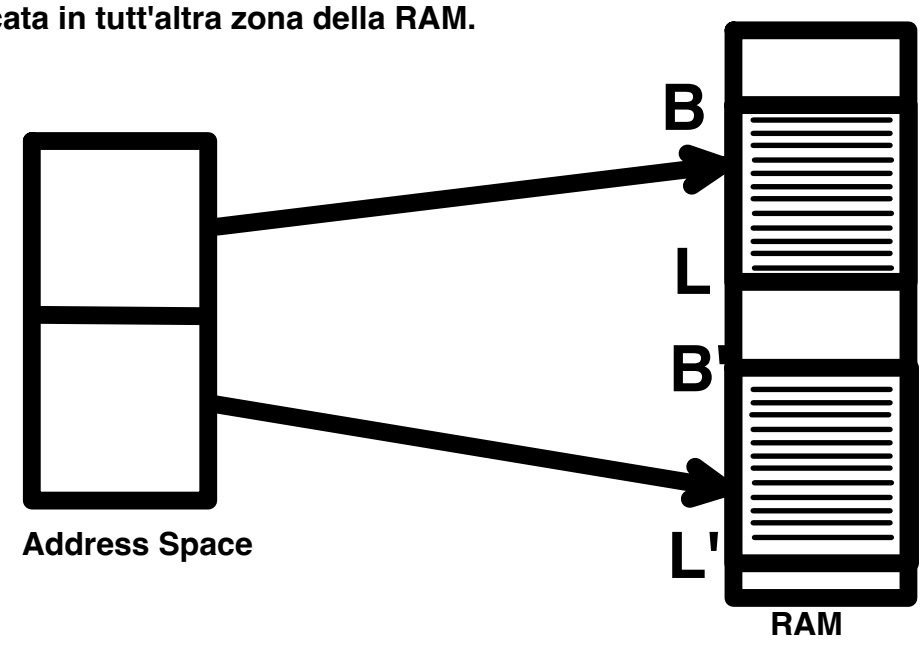


Quindi veniva essere caricato a partire da una BASE sino ad un LIMITE.
***Questo è un grossissimo problema*.**
Soprattutto quando noi consideriamo la frammentazione. Potremmo avere zone frammentate così:



E poi magari dobbiamo lanciare un altro address space, ossia un'altra applicazione che entrerebbe nella zona FREE 1 + FREE 2, ma queste due zone non sono contigue.
Quindi non possiamo effettuare il caricamento contiguo, di questo address space, nella memoria.

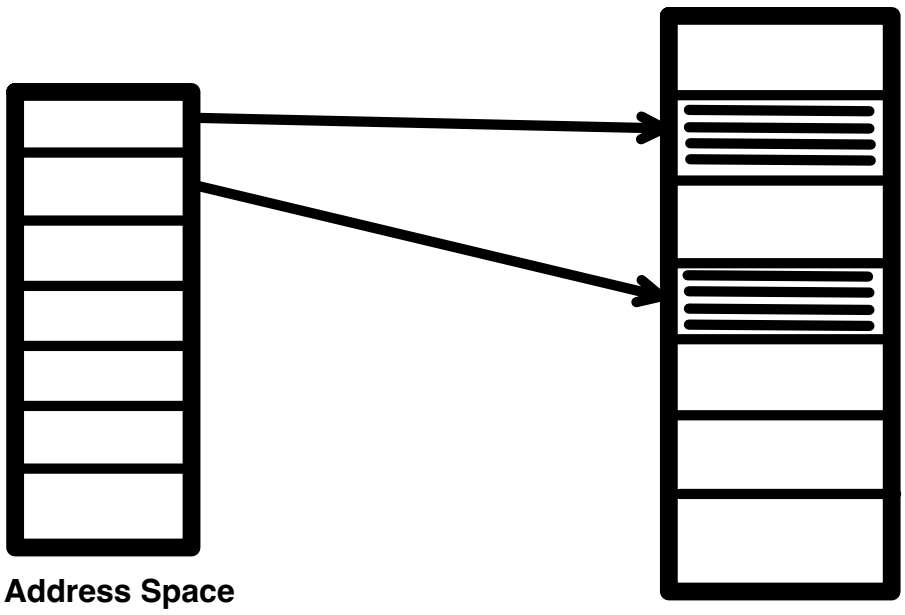
La paginazione ci permette di far esattamente questa cosa qua. Dato un address space, ci permette di avere una parte dell'address space caricata all'interno di una zona della RAM, e un'altra parte dell'address space caricata in tutt'altra zona della RAM.



Queste due zone non sono contigue.
Quindi abbiamo eliminato questo vincolo di caricare in maniera contigua l'address space all'interno della RAM.
In questo scenario non ci basta più una MMU basica: nella MMU noi avevamo soltanto la BASE e il LIMITE.
Qui abbiamo due basi e due limiti.
Una MMU basica tradizionale, in cui mantenevamo solo la base e il limite, chiaramente non era sufficiente.

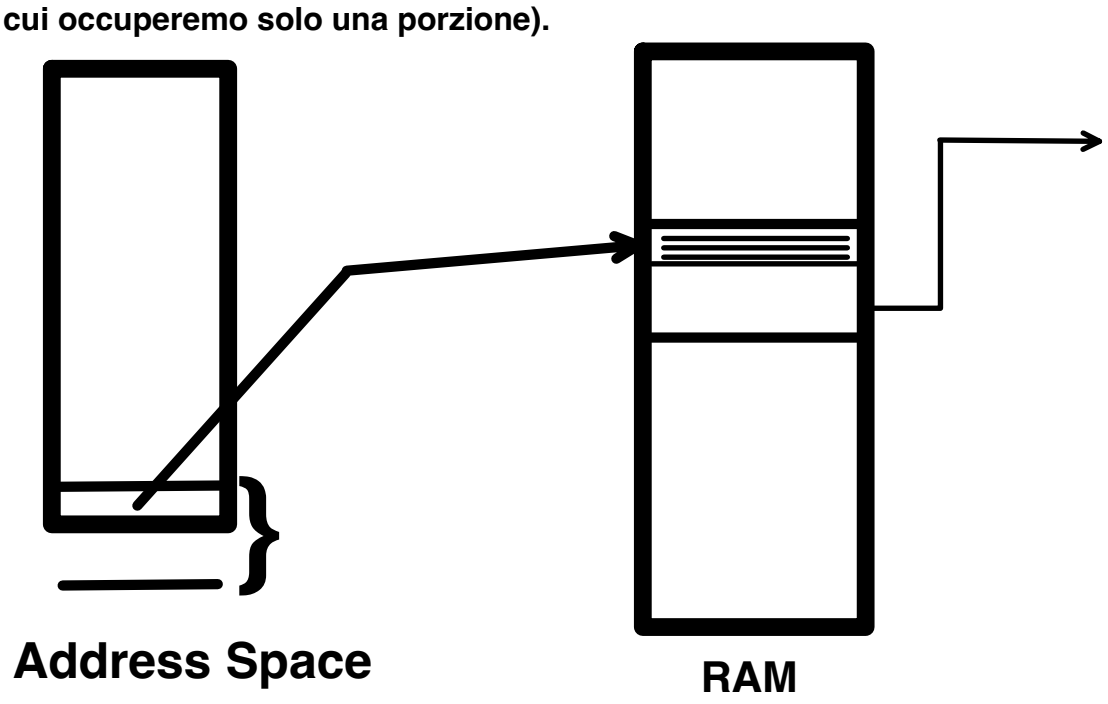
Vediamo cosa succede nella paginazione in maniera approfondita.

Supponiamo di avere un address space suddiviso in zone. Queste zone (interne all'address space) venivano chiamate pagine logiche che sono tutte uguali, come su un libro, abbiamo tutte pagine della stessa taglia.
L'ultima pagina potrebbe essere più piccola, come l'ultima pagina di un libro potrebbe contenere non tutta l'informazione.
La RAM veniva logicamente suddivisa in pagine, queste sono pagine fisiche.
Quello che noi abbiamo è che ciascuna di queste pagine logiche poteva andare a finire all'interno di una pagina fisica.
L'altra pagina logica poteva andare a finire più avanti in una pagina fisica.



E così via.
Quindi potevamo portare le informazioni associate all'address space, in maniera non contigua dentro la RAM.
Ovviamente ottimizzando anche il PLACEMENT, perché la taglia della pagina logica e la taglia della pagina fisica, erano uguali. Quindi qui non c'è frammentazione interna. Non vi è nemmeno una frammentazione esterna, perché ognuno dei buchi liberi è comunque utilizzabile per qualsiasi pagina di qualsiasi address space.
Quindi potevamo caricare lì anche della pagine logiche di altri processi.

Ovviamente l'unico vincolo è che se l'address space ha l'ultima zona che non casca all'interno di una pagina completa ma ci sono informazioni di taglia minore, questa va comunque ad impegnare un'intera pagina fisica (di cui occuperemo solo una porzione).



Questo è l'unico punto di frammentazione interna che abbiamo.

Tipicamente se un address space è formato da tante pagine logiche, quell'unica pagina logica frammentata contribuisce in maniera "marginale" alla frammentazione interna che abbiamo sulla pagina fisica all'interno della RAM.

Questo è uno schema estremamente flessibile con il quale possiamo collocare le informazioni che sono all'interno di un address space, verso la memoria fisica.
Quindi ricapitolando.
Un address space è un insieme di pagine con taglia fissa pari a X byte.
La RAM è un insieme di pagine anch'esse con taglia ciascuna di X byte.
Questi sono denominati frame, sono pagine fisiche.
Ogni pagina logica viene caricata in uno specifico frame di memoria.
I frame allocati possono anche essere "non contigui".

- lo spazio di indirizzamento di un processo viene visto come un insieme di pagine di taglia fissa pari ad X bytes
- la memoria di lavoro viene vista come partizionata in un insieme di pagine della stessa taglia (X bytes), denominate frames
- ogni pagina dello spazio di indirizzamento viene caricata in uno specifico frame di memoria
- i frame allocati per un dato processo possono anche essere non contigui (allocazione di processo non contigua in memoria)
- la frammentazione interna alle pagine è in media di X/2 bytes per ogni processo attivo

Il problema di ricompattare la memoria libera come facevamo nello schema di frammentazione dinamica, non lo abbiamo più.
Perché non è più vero che dobbiamo essere contigui in memoria fisica, anzi, possiamo avere la nostra rappresentazione dell'address space in memoria fisica, in maniera sparsa.

Vantaggio

- l'allocazione non contigua aumenta la flessibilità di allocazione dei processi in memoria
- con N bytes liberi in memoria (distribuiti su ⌈N/X⌉ frames) è sempre possibile caricare

in memoria un processo con taglia fino ad N byte