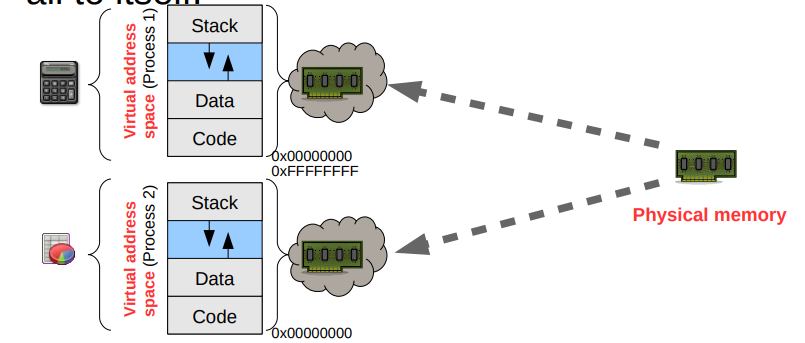
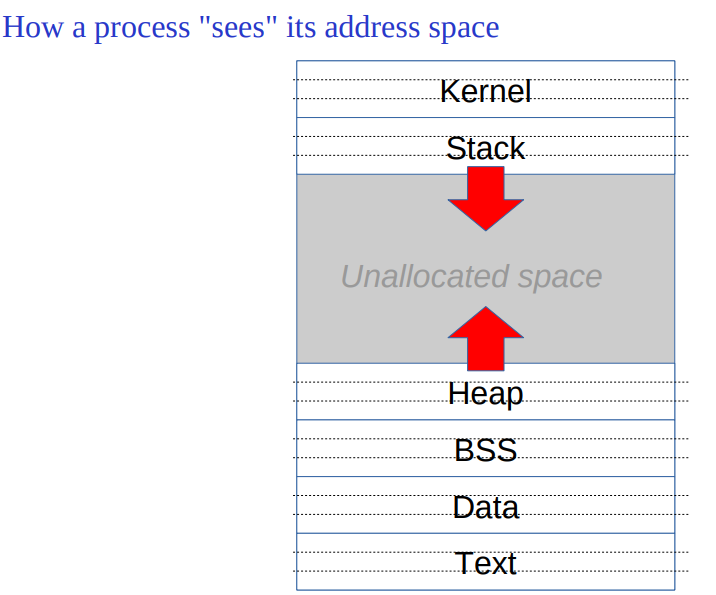
**Gestione della memoria**

Ogni processo deve ottenere un quantitativo di memoria sufficiente per l’esecuzione del codice

Il sistema operativo visualizza la memoria dando ad ogni processo uno spazio di indirizzamento virtuale che dà l’impressione che la memoria sia tutta sua.

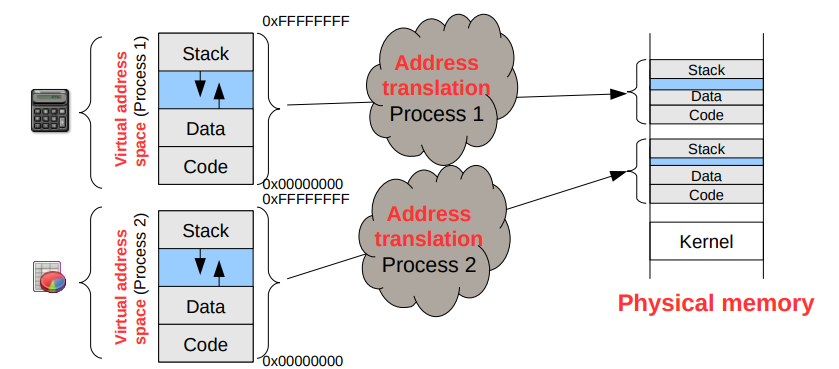




Il kernel consente di velocizzare le chiamate di sistema

**Virtualizzazione**

La virtualizzazione consiste nella traduzione degli indirizzi della memoria fisica in indirizzi della memoria virtuale. La virtualizzazione consente di isolare i processi



Nei sistemi moderni si usa la **Paginazione**

La paginazione usa il principio della località.

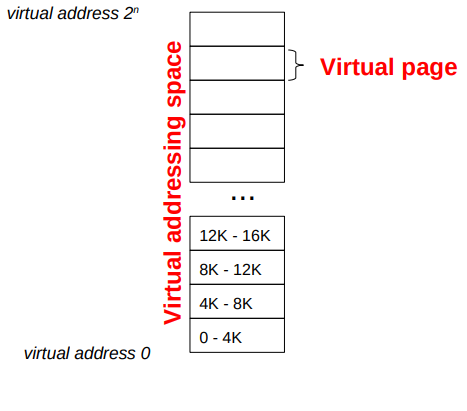
I programmi non usano la memoria in modo totale, ma usa segmenti sfruttando la località (se uso qualcosa probabilmente mi servirà qualcosa intorno a ciò che uso).

**Funzionamento**

La paginazione ha bisogno di un supporto hardware,

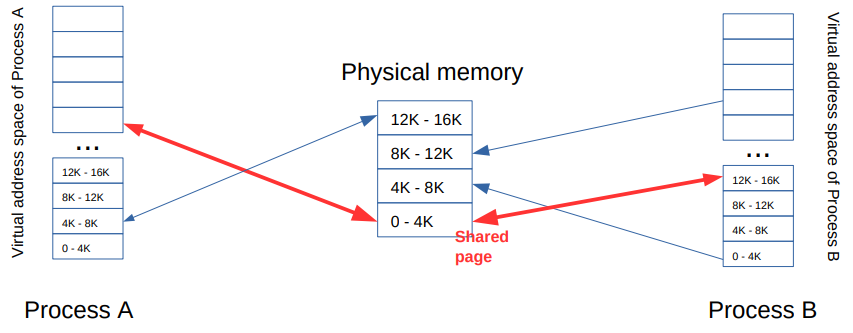
Si divide la memoria in segmenti più piccoli di dimensione fissa (4KB ad esempio)

*Esiste uno spazio di indirizzamento per ogni processo*



Bisogna dividere la RAM in segmenti in frame di pagina, divisione in segmenti più piccoli.

In RAM si spostano gli indirizzi tradotti.



Si possono avere delle pagine condivise tra processi

**Traduzione Virtuale-Fisica**

Gli indirizzi dello spazio di indirizzamento virtuale sono validi solo all’interno del processo

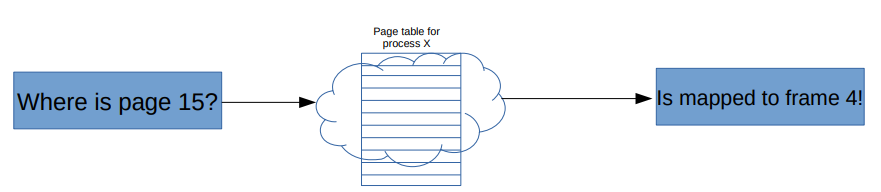
Si usa una tabella di traduzione

Input: numero di pagina

Output: numero di frame

Prendo l’indirizzo virtuale e lo divido per il numero di pagine

La CPU effettua questa operazione



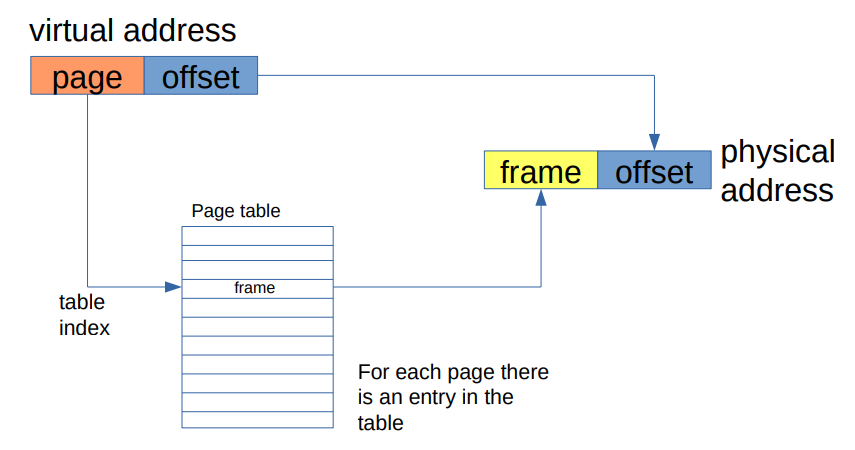
Fin quando c’è spazio nella pagina non alloco una nuova pagina.

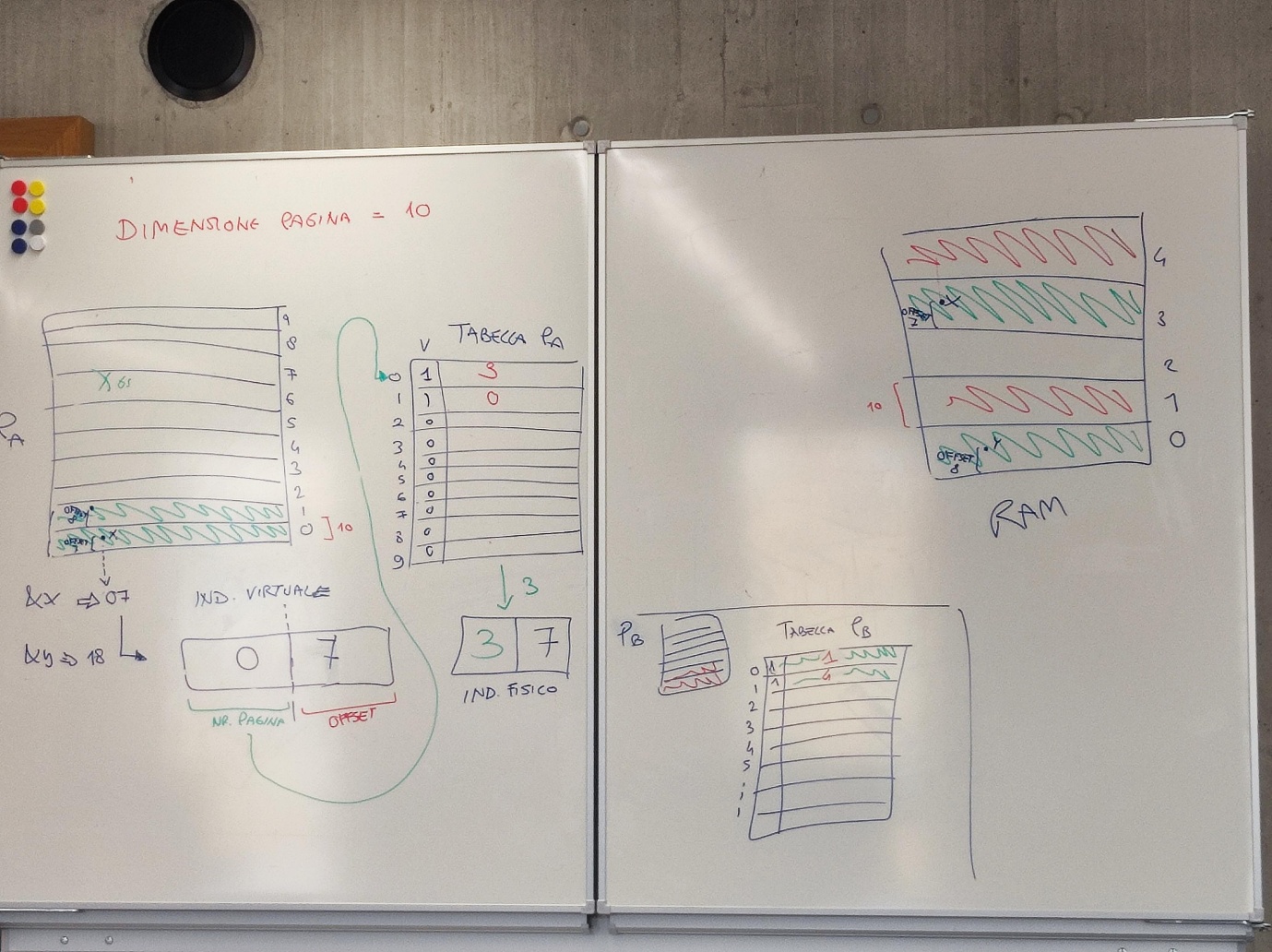
Quando un programma in esecuzione alloca della memoria il SO controlla prima se può esaudire la richiesta in una pagina esistente, altrimenti mappa una nuova pagina nella tabella corrispondente.

Quando la pagina non serve più si libera memoria facendo unmap (lo fa il SO) invalidando la pagina nella tabella delle pagine.

Quando un processo deve accedere alla memoria tramite l’indirizzo virtuale ha bisogno di sapere a che pagina appartiene.

Le pagine hanno dimensione fissa ed è potenza di due, quindi facilmente trattabile per operazioni di divisione tramite la divisione del numero di indirizzo, dove la parte in bit più significativa indica il numero di pagina la meno significativa indica l’offset della pagina





Ogni processo ha una sua tabella di traduzione

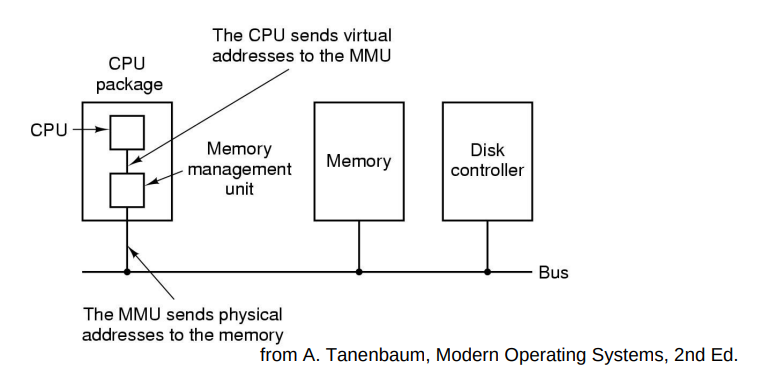
Nella tabella c’è un bit di validità.

L'assegnamento delle page number lo fa il sistema operativo

Se il dato è più grande della pagina, si allocano più pagine e il processo di accesso coinvolge tutto il sistema di traduzione ogni volta.

In ram si può avere il dato sparso, ma tramite la traduzione si possono ottener egli indirizzi corretti.

Tutto questo processo si effettua a livello hardware tramite l’MMU(Memory managment Unit)



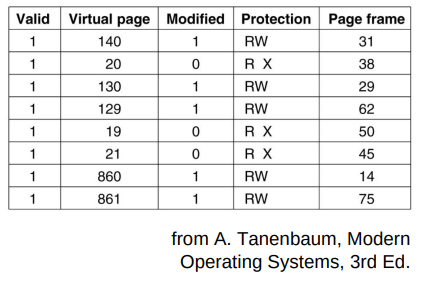
Un accesso invalido si ottiene una Segmentation fault

**MMU**

Per evitare sovra-accessi al componente, c’è una cache chiamata TLB (translation lookaside buffe)

La tabella TLB è valida **SOLO** per il processo che è in esecuzione, con un context switch si butta via il contenuto, TLBflush.

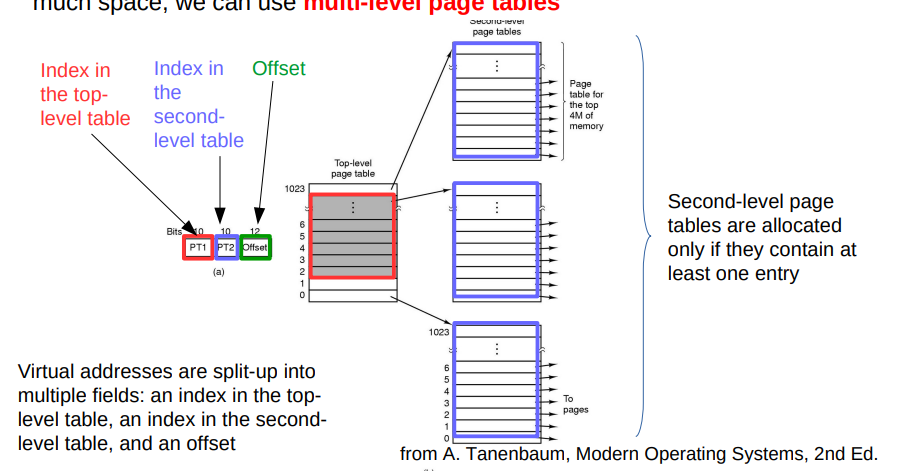
MMU prima di eseguire la traduzione interroga la TLB e se ottiene un TLBHit prende il dato dalla cache se ottiene un TLBMiss rieffettua il calcolo della traduzione dell’indirizzo, una volta calcolato il dato viene inserito nella TLB.



**Dimensione della memoria**

Il numero di pagine si ottiene con 2nBitsCPU / dimensione memoria

Il grande spazio di indirizzamento richiede spazio, per evitare sprechi di spazio si usa una tabella a più livelli.



Si ha una tabella di primo livello che contiene un indice per la tabella di secondo livello (gli indici in un processo a 32 sono da 10 bit)

PT1 | PT2 | Offset

Senza tabella delle pagine il processo non funziona

Il sistema operativo quando fa context switch aggiorna il registro CR3 (in x86) ed il valore che aggiorna indica dove la tabella è aggiornata in memoria.

**Esaurimento della memoria**

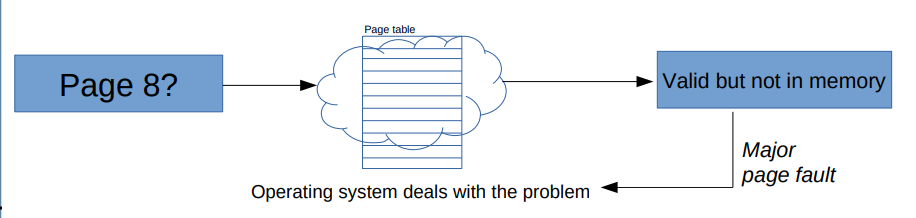
Quando la RAM è piena si spostano i dati non utili nella memoria swap che risiede sul disco (HHD, SSD)

Swap out: ram->swap

Swap-in: ram<-swap

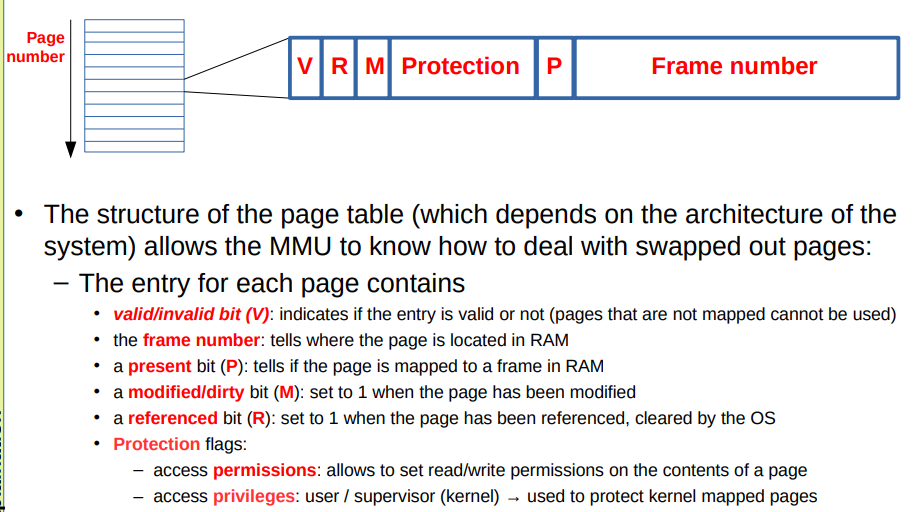
In linux si può creare una partizione dedicata, su windows è un file chiamato pagefile.sys

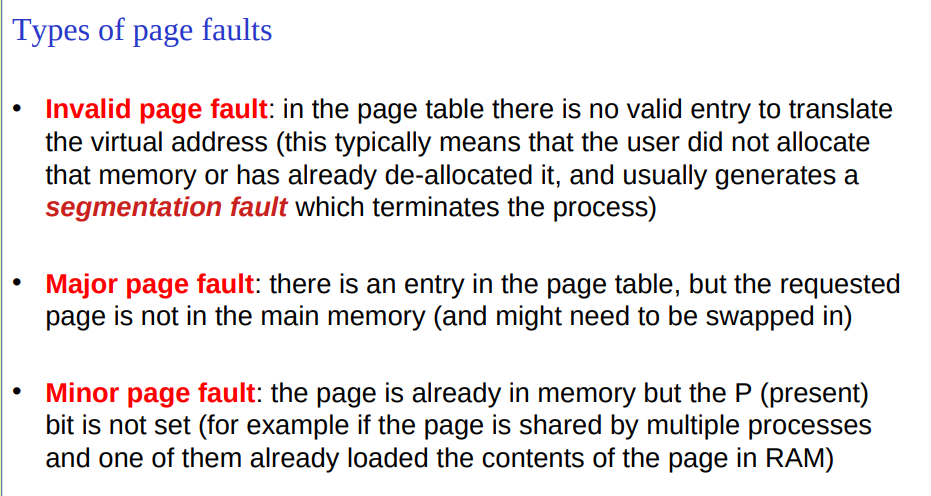
**Major page fault**



Se il page table walker algorithm incontra una entry di una pagina valida, ma si trova sullo swap, la cpu genera un major page fault exception, che viene catturata dal SO

Viene generata dal MMU





**Allocare e deallocare memoria**

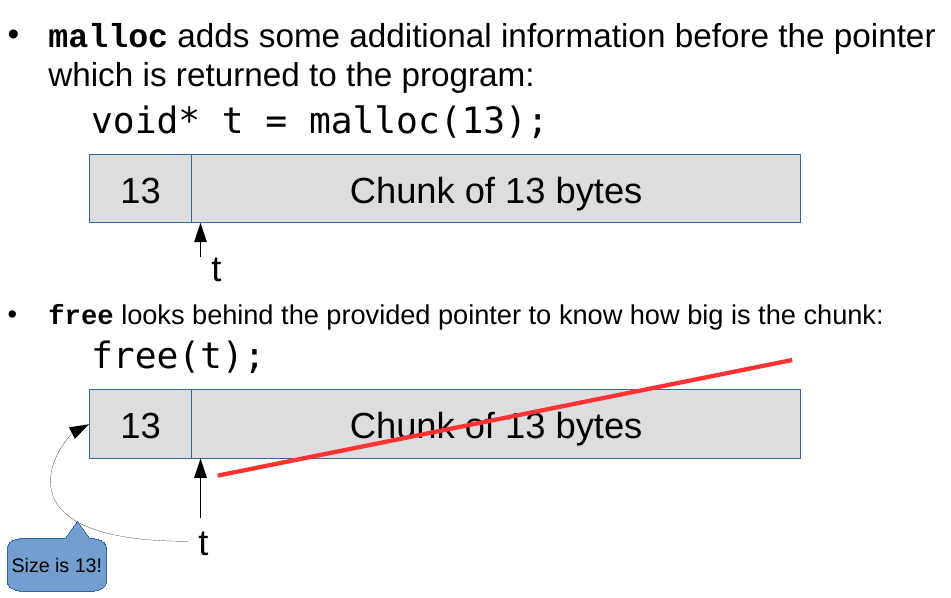
Esempio di malloc

Con malloc alloca lo spazio richiesto,

Il free non necessita lo spazio da liberare, ma lo capisce da puntatore.

In realta oltra al nostro spazio viene riempito un ulteriore spazio che contiene la dimensione allocata.

Questo è sfruttato dal free per liberare.

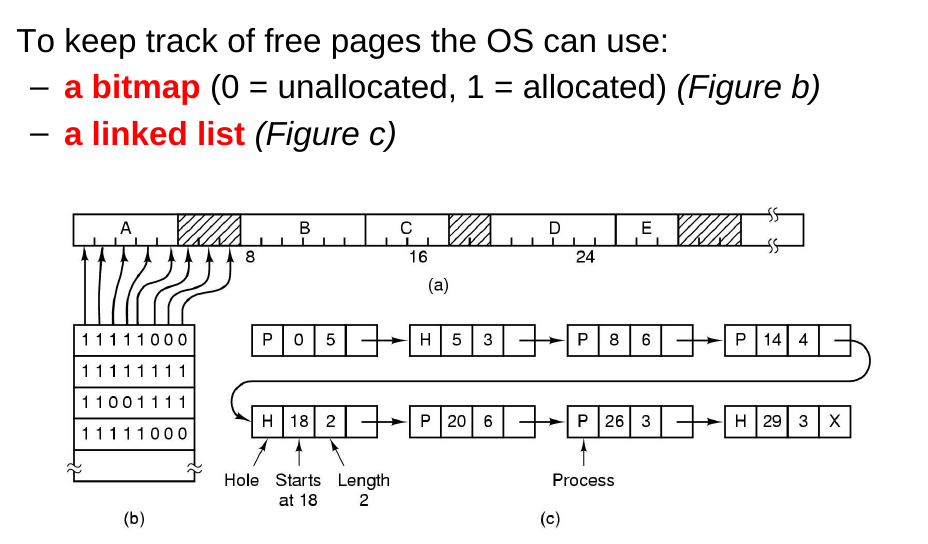


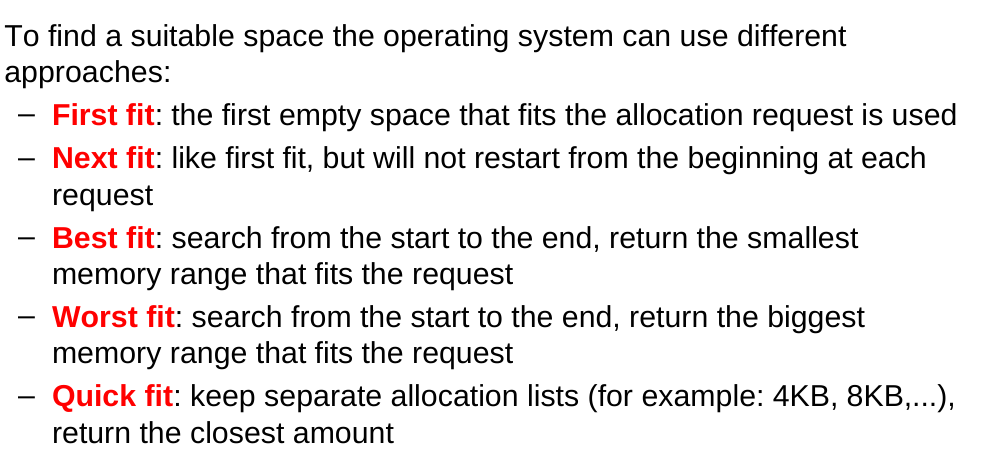
Solitamente si dovrebbe mantenere l’ordine di allocazione e deallocazione eseguendolo al contrario, per evitare buchi nello heap.

Dietro a malloc si trova mmap, metodo più flessibile e a basso livello.

**Gestione dello spazio in memoria**

Il sistema operativo deve mantenere le informazioni sulle pagine





Worst fit evita che spazi intermedi più piccoli vengano lasciati vuoti perché non sufficienti alle richieste, cosa che fa il best fit che cerca la locazione più piccola.