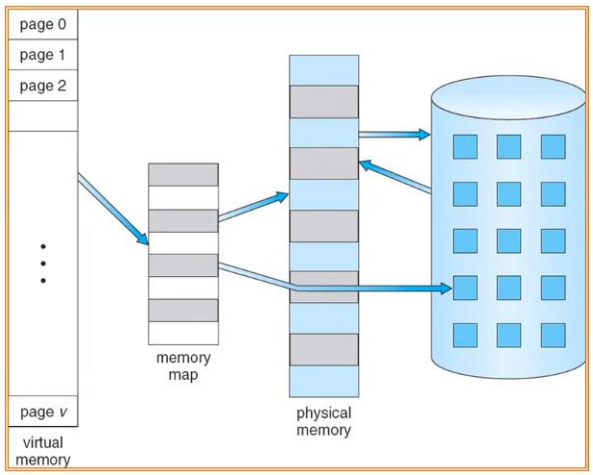
10. MEMORIA VIRTUALE

Quando un processo viene mandato in esecuzione gli si deve assegnare dello spazio (o contiguo o in frame) nella memoria principale. L'idea della memoria virtuale è che solo **una parte** del programma necessita di essere in memoria per l'esecuzione.

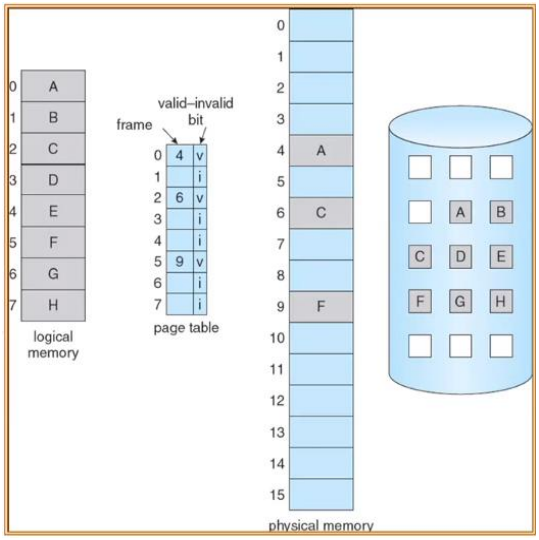


Supponiamo di avere un processo suddiviso in tante pagine (memoria virtuale), una memory map (page table), memoria fisica e poi il disco.
L'idea è caricare nella memoria principale solo un numero limitato di pagine del processo. Durante l'esecuzione, se c'è necessità di una nuova pagina la si prende dal disco, la si porta in memoria principale e si aggiorna la page table del processo, ovviamente bisogna avere un frame libero nella memoria principale per poter caricare la nuova pagina richiesta dal processo.

La memoria virtuale può essere implementata attraverso la **paginazione su richiesta** (**demand paging**) e **segmentazione su richiesta** (**demand segmentation**).

10.1 PAGINAZIONE SU RICHIESTA (DEMAND PAGING)

Un processo quando necessita di una pagina la richiede e il SO trovare il modo di prende la pagina, ancora sul disco, e portarla in memoria principale.



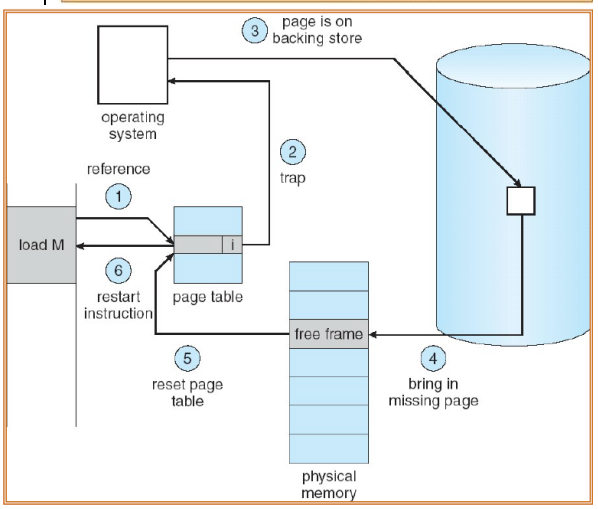
Per poter gestire quali pagine del processo sono realmente in memoria principale o meno, si utilizza il **bit di validità**.
Ad ogni elemento della page table è associato un bit (**v** → in-memoria, **i** → non-in-memoria). Inizialmente il bit è impostato a **i** per tutti gli elementi. Durante la traduzione dell'indirizzo, se il bit è **i** nell'elemento della tabella delle pagine, significa una mancanza di pagina e viene a crearsi un **page fault**.

Supponiamo un processo che richiede 8 pagine, con la sua page table, successivamente il processo quando parte ha caricato in memoria solo 3 pagine.

PAGE FAULT:
Quando avviene un ***riferimento*** ad una pagina non presente in memoria (cioè una pagina contrassegnata come non valida), si provoca una ***trap*** (segnale) al SO.
Il SO esamina una sua tabella interna per decidere cosa fare:

- Se si tratta di una pagina che non fa parte di quel processo, allora tale processo termina;
- Se la pagina è di quel processo ma c'è il bit di validità ad **i**, vuol dire che è in **page fault**.

In quest'ultimo caso, il SO deve porsi il problema di prendere quella pagina dal disco e portarla in memoria principale, così cerca un frame libero e sposta la pagina nel frame, dopodiché modifica la page table (scrivendo il numero del frame e mettere a **v** il bit di validità), fatto tutto ciò bisogna riavviare l'istruzione che ha causato tale **page fault**.



PRESTAZIONE DELLA PAGINAZIONE SU RICHIESTA:
Probabilità di page-fault $0 \leq p \leq 1$:

- se p = 0 non ci sono mancanze di pagine
- se p = 1, ogni riferimento è una mancanza di pagina

Tempo di accesso effettivo (**EAT**):

EAT = (1 – p) x accesso alla memoria + p (page fault overhead + lettura della pagina + overhead ripresa)

Esempio:

Tempo di accesso alla memoria = 200 nanosecondi

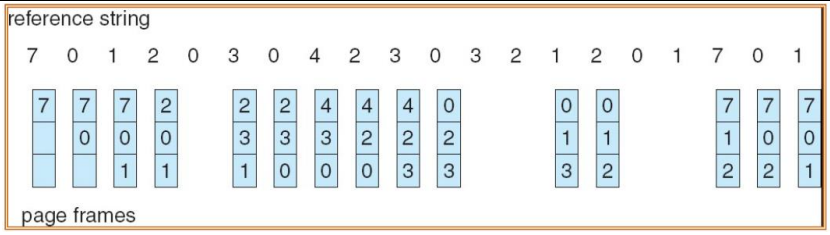
Tempo medio di gestione di un page fault = 8 millisecondi = 8,000,000 nanosecondi

EAT = (1 – p) * (2*200) + p * 8,000,000 = (2*200) + p x 7,999,600 = 8199,8 nanosecondi

SOVRALLOCAZIONE:
Questa situazione si verifica quando non sono più disponibili frame liberi in memoria. L'unica soluzione è sostituire delle pagine già presenti in memoria principale, in base a determinati criteri. Un algoritmo che implementa questa soluzione: individua la posizione della pagina desiderata sul disco, se non c'è un frame libero si usa un algoritmo di sostituzione (altrimenti si procede normalmente) che sceglie il **frame vittima**, carica la pagina desiderata nel frame libero (aggiornando la page table) e si riprende il processo.
Se bisogna sostituire una pagina, in teoria, si prende questa pagina, la si porta sul disco e si inserisce la pagina nuova, ma questo è necessario solo quando tale pagina viene modificata, così si aggiunge alla page table il **bit di modifica**, oltre al bit di validità, che indica se la pagina è stata modificata.

10.1.1 SOSTITUZIONE FIFO DELLA PAGINA

Avendo un numero fissato di frame, se si verifica un page fault, Il SO sceglie il frame vittima seguendo il criterio FIFO, la prima pagina inserita sarà la prima ad essere sostituita, e così via…



Avendo la seguente stringa di riferimento: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Se ho 3 frames (3 pagine possono essere in memoria per ogni processo):

1	1	1	4	4	4	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3
		3	3	3	2	2	2	2	2	4

In questo casi sia ha 9 page fault.

Se ho 4 frames:

1	1	1	1	1	1	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	5
		3	3	3	3	3	3	2	2	2
			4	4	4	4	4	4	3	3

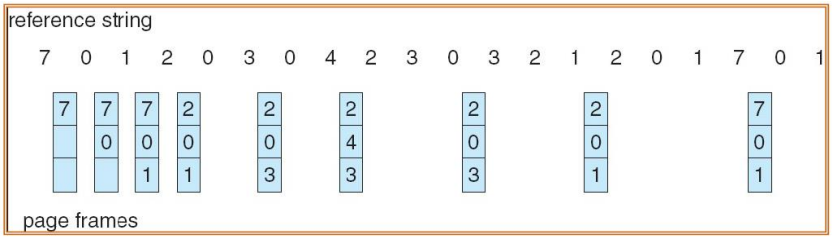
Qui si hanno 10 page fault.

Con tale algoritmo di sostituzione si ha la **Belady's Anomaly**, ovvero più frame più page fault.

10.1.2 SOSTITUZIONE OTTIMALE DELLE PAGINE

Sostituisce la pagina che non si userà per il periodo di tempo più lungo.

Questo algoritmo richiede la conoscenza della sequenza di riferimenti futuri. Usato per misurare quanto sono buone le prestazioni di un algoritmo.



Avendo la seguente stringa di riferimento: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

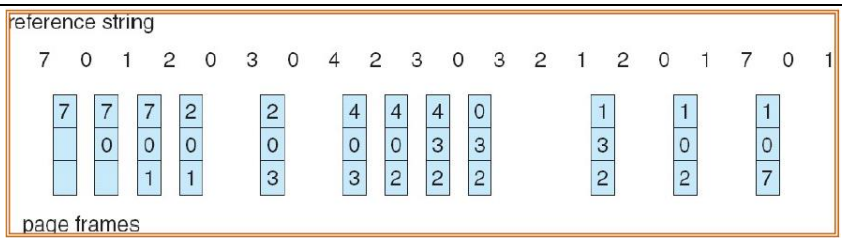
Se ho 4 frames:

1	1	1	1	1	1	1	1	1	4	4
	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3
			4	4	4	5	5	5	5	5

In questo caso ho 6 page fault.

10.1.3 SOSTITUZIONE LRU (LAST RECENTLY USED) DELLE PAGINE

Sostituisce la pagina che non è stata usata per il periodo di tempo più lungo.



Avendo la seguente stringa di riferimento: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Se ho 4 frames:

1	1	1	1	5
2	2	2	2	2
3	5	5	4	4
4	4	3	3	3

IMPLEMENTAZIONE LRU:

L'implementazione di tale algoritmo, esistono diverse tecniche, la prima è attraverso un **contatore**:

- Si assegna alla CPU un contatore che si incrementa ad ogni riferimento alla memoria;
- Si assegna ad ogni entry della tabella delle pagine un ulteriore campo;
- Ogni volta che si fa riferimento ad una pagina viene copiato il valore del contatore della CPU nel campo corrispondente per quella pagina nella tabella delle pagine;
- Quando una pagina deve essere sostituita si guardano i valori dei contatori, scegliendo la *pagina con il valore del contatore più basso*.

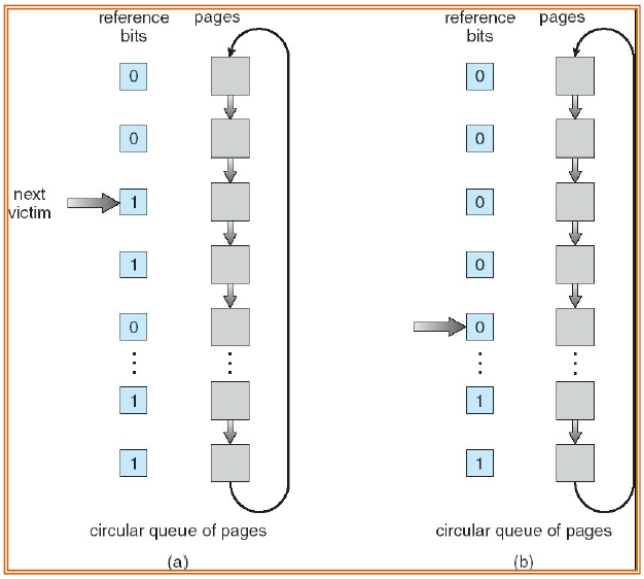
La seconda implementazione è attraverso uno **stack**:

- Mantenere uno stack dei numeri di pagina in una lista a doppio collegamento;
- Riferimento ad una pagina: se è una pagina non presente nei frame la si mette in cima allo stack, ogni volta che si fa riferimento ad una pagina già presente nello stack bisogna prelevare tale riferimento dalla sua posizione e metterla in cima, nessuna ricerca per la sostituzione: basta *prelevare la pagina dal fondo*;
- Richiede di cambiare al più 6 puntatori.

Le implementazioni con contatore e stack richiedono molte risorse, una soluzione più semplice è quella di assegnare alla page table, oltre al bit di validità e modifica, un **bit di riferimento** che inizialmente è 0 quando la pagina viene caricata la prima volta in memoria, quando viene fatto riferimento nuovamente a questa pagina, viene messo il bit ad 1 e nel momento in cui si deve scegliere una pagina da eliminare si va a scegliere la pagina che ha questo bit di riferimento a 0, perché significa che oltre al momento iniziale che si setta a 0, tale pagina non è stata proprio usata.

Questa soluzione semplice è una approssimazione, perché non è detto che si trova solo un bit posto a 0, ma possono essere molteplici, e nasce il problema di quale scegliere. Tale approssimazione potrebbe essere migliorata con l'algoritmo della seconda chance.

ALGORITMO DELLA SECONDA CHANCE:
Le pagine vengono predisposte in una sorta di lista circolare, e quando si seleziona una pagina vittima si inizia la scansione della lista.
Si parte da una certa posizione, quando si cerca una pagina vittima si cerca la pagina che ha il bit di riferimento ad 1, lo si fa diventare 0 (seconda chance) e si prosegue con la scansione, così con tutti gli altri bit ad 1.
Alla prima pagina che ha bit 0, questa sarà la vittima.

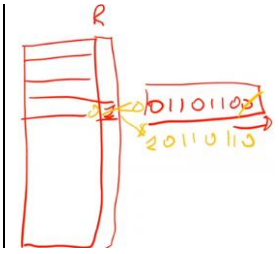


RAFFINAMENTO DELL'ALGORITMO DELLA SECONDA CHANCE:
Invece di considerare solo il *bit di riferimento*, si valuta anche il **bit di modifica** in una coppia (**bit riferimento**, **bit modifica**):

- (0,0) non usato di recente, non modificato
- (0,1) non usato di recente, modificato
- (1,0) usato di recente, non modificato
- (1,1) usato di recente, modificato

La pagina che verrà eliminata sarà quella con la **coppia di valore minore**, chiaramente la pagina che non è stata modificata costa meno perché consente di avere un accesso in meno al disco durante la sostituzione.

Un altro raffinamento potrebbe essere quello di avere più *bit di riferimento* (tenuti in un **registro a 8 bit**).
In corrispondenza ad una pagina p si ha il bit di riferimento ed un registro a 8 bit, associato alla pagina.
Ad intervalli regolari, si prende il contenuto del registro e lo si shifta a destra di una posizione, shiftando anche il bit di riferimento. La pagina da eliminare, in questo caso, sarà quella col valore di registro più basso.



ALGORITMO DI CONTEGGIO:
Esistono altri tipi di algoritmi, a parte LRU, che tengono un contatore del numero di riferimenti che sono stati fatti ad ogni pagina:

- **Algoritmo LFU** (**Least Frequently Used**): sostituisce la pagina con il più basso conteggio;
- **Algoritmo MFU** (**Mostl Frequently Used**): sostituisce la pagina con il conteggio più alto, è basato sulla assunzione che la pagina col valore più basso è stata spostata recentemente in memoria e quindi deve ancora essere impiegata.

BUFFERIZZAZIONE DELLE PAGINE:
Per semplificare la scelta delle *pagine vittima* si utilizza un buffer, ovvero si considera un **pool di frame liberi** per soddisfare le richieste velocemente.
Quando si sceglie una pagina vittima, invece di inserirla nel disco, la si trasferisce nel pool dei frame liberi, in questo modo se si fa riferimento a tale pagina successivamente il SO va vedere se è presente nel pool, in caso affermativo la prende e la porta in memoria evitando l'accesso al disco.
Il processo può essere riattivato rapidamente, senza attendere la fine dell'operazione di salvataggio del frame vittima sulla memoria di massa.
Le pagine modificate del pool dei frame liberi sono scritte sul disco periodicamente in background.

10.2 ALLOCAZIONE DEI FRAME

Ogni processo ha bisogno di un numero **minimo** di pagine.
Esempio: IBM 370 – 6 pagine per gestire l'istruzione MVC:

- L'istruzione richiede 6 byte, che possono estendersi su 2 pagine.
- 2 pagine per gestire il "from" dell'istruzione.
- 2 pagine per gestire il "to" dell'istruzione.

Quindi significa che per gestire tale operazione servono almeno 2 pagine.
Però bisogna stabilire i criteri per poter assegnare i frame ai vari processi, i principali sono **allocazione fissa** e **allocazione a priorità**.

ALLOCAZIONE FISSA:

- **Allocazione uniforme**, avendo *m* frame ed *n* processi, si assegnano *m/n* frame a ciascun processo.
  Esempio: se 100 frame e 5 processi, ognuno prende 20 frame.
- **Allocazione proporzionale**, si assegna la memoria disponibile ad ogni processo in base alle dimensioni di quest'ultimo, ovvero più un processo richiede spazio e più sarà lo spazio assegnatogli.

$s_i = \text{size del processo } p_i$
$S = \sum s_i$
$m = \text{numero totale di frames}$
$a_i = \text{numero di frame allocati per } p_i = \frac{s_i}{S} \times m$

Esempio:
$m = 64$
$s_1 = 10$
$s_2 = 127$
$a_1 = \frac{10}{137} \times 64 \approx 5$
$a_2 = \frac{127}{137} \times 64 \approx 59$

ALLOCAZIONE A PRIORITÀ:

Usare uno schema di allocazione proporzionale basato sui valori delle priorità piuttosto che delle dimensioni, esempio processo di sistema, più frame si assegna al processo e maggiore sarà la su velocità di esecuzione.

SOSTITUZIONE GLOBALE E LOCALE:

Quando si sceglie la pagina vittima la si può scegliere in due ambiti:

- **Sostituzione locale**, vengono considerati solo i frame allocati al processo.
- **Sostituzione globale**, permette di selezionare un frame di sostituzione a partire dall'insieme di tutti i frame, anche se quel frame è correntemente allocato a qualche altro processo, cioè un processo può prendere un frame da un altro processo.

In quest'ultimo caso, un processo non può controllare la propria frequenza di page fault ed il tempo di esecuzione di ciascun processo può variare in modo significativo.

10.3 THRASHING

Se un processo non ha abbastanza pagine, il tasso di page fault aumenta. Questo comporta:

- riduzione utilizzo della CPU;
- il sistema operativo ritiene che sia necessario aumentare il livello di multiprogrammazione;
- un altro processo aggiunto al sistema.

Thrashing ≡ si spende più tempo nella gestione della paginazione che nella esecuzione dei processi.

Per evitare il thrashing occorrerebbe sapere quali, oltre che quante pagine, servono.

Modello di località: Una località è un insieme di pagine che vengono accedute insieme, che sono contemporaneamente in uso attivo, il processo si muove da una località all'altra e le località possono sovrapporsi.

Perchè si verifica il thrashing? **dimensione della località > numero di frame assegnati**

Esempio:

Quando viene invocato un sottoprogramma, si definisce una nuova località: vengono fatti riferimenti alle sue istruzioni, alle sue variabili locali ed a un sottoinsieme delle variabili globali. Quando il sottoprogramma termina, il processo lascia la località corrispondente.

Le località sono definite dalla struttura del programma e dei dati.

MODELLO DEL WORKING-SET:

Per tentare di gestire il thrashing è utilizzare il **modello del working-set**:

- **Δ**: Si considera una finestra temporale **Δ** chiamata *finestra working-set*, fondamentalmente è un numero fisso di riferimenti di pagina, ad esempio 10.000 istruzione, cioè ogni 10.000 riferimenti a pagina viene eseguita un'istruzione;
- **WSS_i**: Si individua il numero totale di pagine cui un processo **P_i** fa riferimento nell'intervallo Δ, cioè si considera il tempo partizionato in queste finestre (intervalli di 10.000 istruzioni) e si va a contare il numero delle pagine che ciascun processo P_i fa riferimento.
  - Se Δ è troppo piccolo non comprenderà l'intera località;
  - Se Δ è troppo grande può sovrapporre parecchie località;
  - Se Δ = ∞ → il working set è l'insieme delle pagine toccate durante l'esecuzione del processo.

A questo punto, si prende il WSS di ciascun processo e li si somma: **$D = \sum WSS_i \equiv$ richiesta globale dei frame**.

Dopo si va a confrontare D con il numero reale di frame nel disco : **m = numero di frame**.

- Se D > m → Thrashing, cioè il numero totale di pagine che sono state richieste dai vari processi è maggiore rispetto al numero di frame disponibili;
- Se D > m, allora occorre sospendere uno dei processi.

Per individuare quali sono le pagine che sono state utilizzate nella finestra di tempo Δ:

Man mano che la finestra scorre può aumentare o diminuire il numero di frame dati al processo.
Nell'esempio in t_1+1 la pagina 2 scompare, quindi il frame ad essa allocato viene considerato libero (può eventualmente essere usato da un altro processo il cui working set invece cresce).