LEZIONE 10

Memoria virtuale

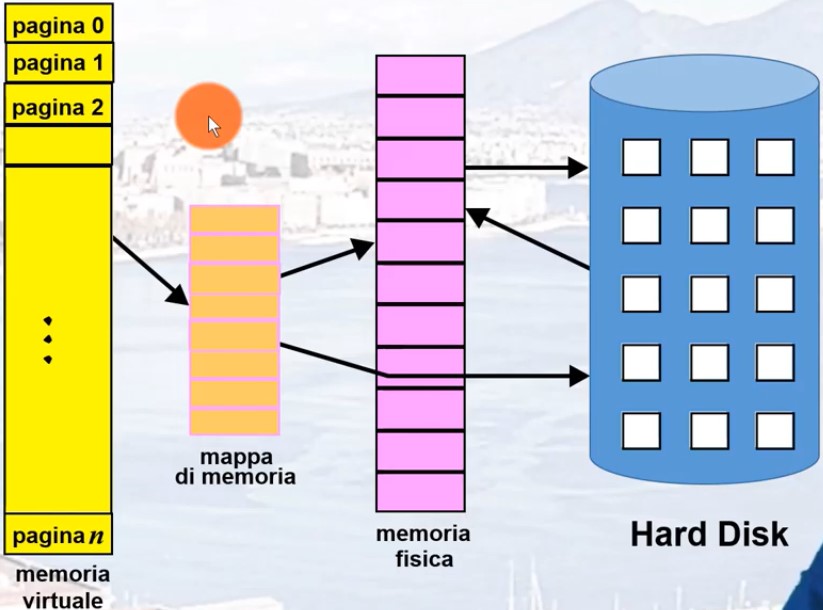
**Introduzione**

* **La memoria virtuale rappresenta la separazione della memoria logica, vista dall’utente, dalla memoria fisica.**
* **Solo parte del programma deve essere maggiore dello spazio degli indirizzi fisici**
* **Lo spazio degli indirizzi logici può essere maggiore dello spazio degli indirizzi fisici**
* Permette la condivisione di file e memoria tra diversi processi tramite la condivisione delle pagine.
* Consente anche un miglioramento delle prestazioni durante la creazione dei processi.

Dato che si carioca un pezzo del programma alla volta, man mano che servono

* La memoria virtuale generalmente si realizza nella forma di:
* **Paginazione su richiesta (demand paging)**
* **Segmentazione su richiesta (demand segmentation)**

**Schema di memoria virtuale più grande della memoria fisica**

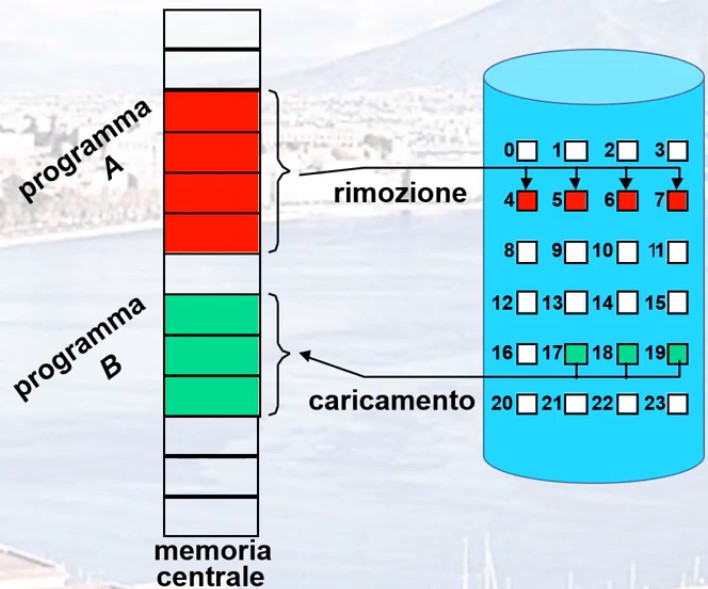
**Spiegazione**

la memoria virtuale è più grande di quella fisica. Dopo aver fatto la mappa di memoria (vista nella lezione 9), sono stati trasformati gli indirizzi logici in fisici. Se la memoria fisica è piena, per fare spazio ad un nuovo processo che ne necessita, si toglie qualcosa (ciò che serve di meno), e lo si posiziona nell’hard disk (memoria ausiliaria), e da quest’ultimo sposto ciò che mi serve nella memoria fisica. Infatti ci sta uno scambio continuo tra queste due.

Paginazione su richiesta

* **Non si carica mai nella memoria una pagina che non sia necessaria.**
* Minore tempo di avvicendamento (dovuto al fatto che si carica meno roba)
* Minore quantità di memoria fisica (a parità di processi, si ha bisogno di un minor quantitativo di memoria, dato che si caricano piccoli pezzetti alla volta)
* Risposta più veloce
* Più utenti (dato che a ogni utente si carica poca memoria, si potrà sfruttare un più alto grado non solo di multiprogrammazione, ma anche di multiutenza

**Esempio di trasferimento di una memoria paginata nello spazio contiguo di un disco**

**spiegazione**

ci sono 2 programmi A e B. Il primo deve essere rimosso, e quindi smontato e messo in hard disk, per fare spazio al programma B che dall’hard disk deve essere messo in memoria. Quindi si smonta un oggetto in tanti pezzi, e questo smontaggio mi permette in maniera dinamica di prendere solo pezzi che servono effettivamente, da e verso la memoria esterna (hard disk)

Bit di validità

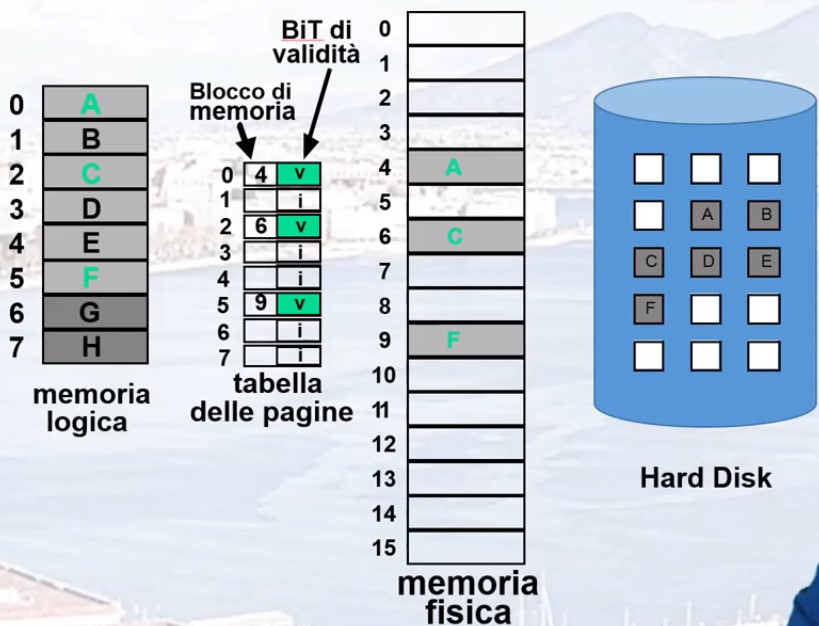
Con la memoria virtuale si può ipotizzare l’uso del bit di validità, il quale non serve solo a segnalare se un indirizzo è giusto o no, ma si potrebbe pensare che tale bit può essere usato per segnalare se la pagina è presente o meno in memoria.

* **A ciascun elemento della tabella delle pagine è associato un bit di validità (1 se l’elemento è in memoria, 0 non è in memoria)**
* Inizialmente il bit di validità è impostato a 0. Il processo non è stato caricato in memoria, quindi il SO si accorge di questa cosa (grazie al bit di validità) e carica la prima pagina
* istantanea di una tabella delle pagine.

**RICORDA**

**La pagina 0 (oppure le prime pagine) sono più importanti riaspetto le altre pagine, in quanto al suo interno ci saranno sicuramente elementi/istruzioni che serviranno e saranno referenziate da tutto il programma.**

**Tabella delle pagine quando alcune pagine non si trovano nella memoria centrale**

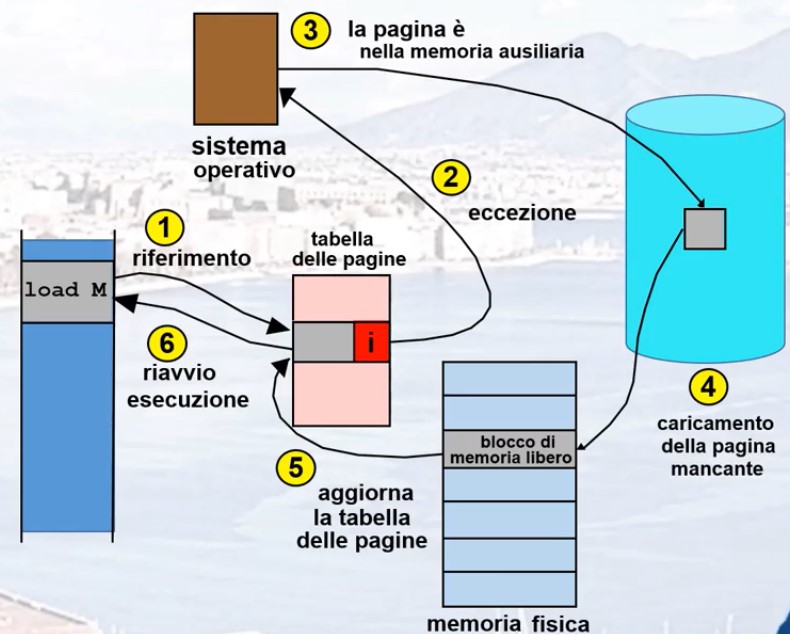
**spiegazione**

Ci sta un programma in memoria logica, lo si smonta e si cerca di caricare solo i pezzi che servono. La tabella delle pagine infatti non alloca per tutti gli elementi, ma solo ciò che serve. Nell’hard disk invece ci sta tutto il programma, e se per qualsiasi motivo serve un pezzo, tipo B, il quale non è caricato in memoria fisica; allora il SO dovrà adoperare per prenderlo dall’hard disk e caricarlo in memoria fisica, ma se la memoria fisica è piena bisognerà togliere qualcosa dal suo interno, e spostarlo in hard disk.

Eccezione di pagina mancante

* Se il processo tenta di accedere a una pagina che non era stata caricata nella memoria, si verifica un’eccezione di pagina mancante **(page fault trap).**
* **L’architettura di paginazione, traducono l’indirizzo, nota che il bit non è valido e invia un segnale d’eccezione al SO.**
* Il SO deve decidere:
* **Errore di indirizzo non valido => abort.**
* **Insuccesso del SO nella scelta delle pagine da caricare in memoria.**
* **Si individua un blocco di memoria libero.**
* **Si trasferisce la pagina desiderata nel blocco di memoria libero**
* **Si aggiornano le tabelle, bit di validità = 1**
* **Si riavvia l’istruzione che era stata interrotta dal segnale di eccezione**

**Fasi di gestione di un’eccezione di pagina mancante**

**spiegazione**

C’è un programma logico, il quale ha un modulo che fa una load m (istruzione generica). Dato che serve la sua corrispondenza fisica, si va nella tabella delle pagine; se vicino a lui ci sta invalid (come in questo caso), si segnala tale evento al SO, il quale decide a cosa è dovuta questa invalidità (se è uno sconfinamento e quindi un errore, oppure, semplicemente non è stata caricata in memoria). Se non è stata caricata, il SO deve andarla a prendere dall’hard disk, dopo di che viene caricata in memoria centrale, dove c’è spazio (se non c’è si butta fuori qualcosa che non serve e si fa spazio); in fine si riaggiorna la tabella delle pagine a valid, e si riavvia l’esecuzione del processo dal punto in cui era stata interrotta.

Prestazioni della paginazione su richiesta

* Frequenza di assenza di pagina 0 ≤ p ≤ 1.0
* Se p = 0 non si verifica assenza di pagina (page fault)
* Se p = 1, ogni riferimento è un’assenza di pagina
* Tempo d’accesso effettivo (EAT, effective access time)

EAT = (1-p) x accesso alla memoria

+ p (page fault overhead)

+ [swap page out]

+ swap page in

+ (restart overhead)

Creazione dei processi

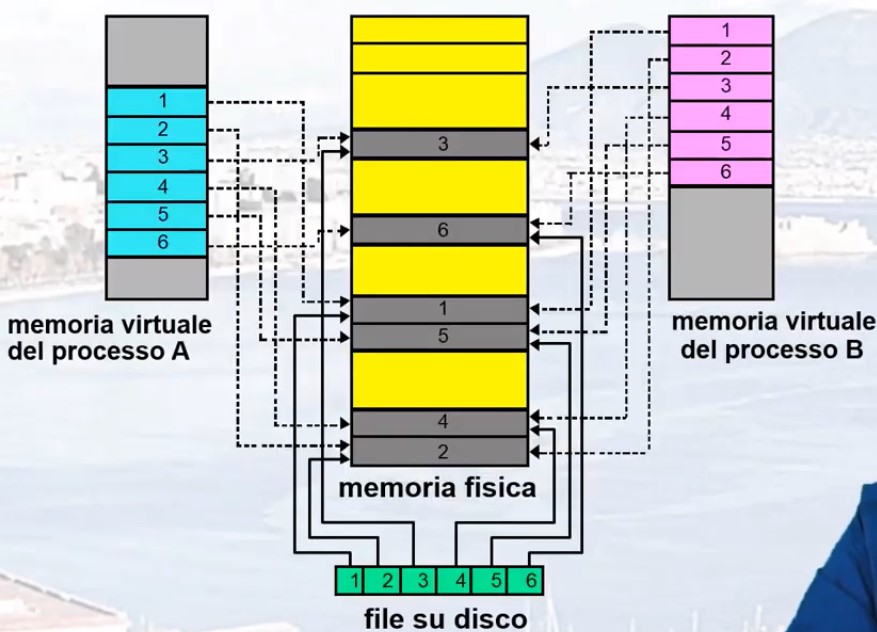
* La memoria virtuale offre altri vantaggi durante la creazione dei processi:

1. **Copia in scrittura (copy-on-write):** nel caso di un thread, il quale si sa che è un processo leggero che condivide molto del processo chiamante(padre), per es. il codice. Se per qualche motivo il processo figlio deve cambiare qualcosa, soltanto nel momento in cui si deve effettuare un cambiamento di una certa pagina, **si fa una ricopia su scrittura della pagina,** perché la lettura a differenza della scrittura non modifica la pagina, ergo se bisogna fare una modifica, quindi una scrittura, si effettua quella che si chiama copy-on-write.

* **Questa tecnica consente una maggiore efficienza poiché solo le pagine modificate vengono copiate.**
* Le pagine libere si trovano in un gruppo (pool) di pagine libere che di solito si assegnano quando la pila di un processo deve espandersi oppure proprio per gestire pagine da copiare su scrittura.

1. **Associazione dei file alla memoria:** per rendere una serie di funzioni che in qualche modo permettano di gestire questo scambio di informazioni da e verso il disco, la RAM, la memoria logica. Per uniformare tale via-vai di scambi, si è pensato di associare/di far finta che la memoria fisica si vada ad estendere sull’hard disk. Dato che ci sono delle system call che agiscono sulla RAM, ed altre che agiscono sull’hard disk; se in qualche modo il SO stabilisce un’associazione, quindi maschera la differenza tra memoria centrale e disco.

**funzionamento dell’associazione dei file alla memoria**

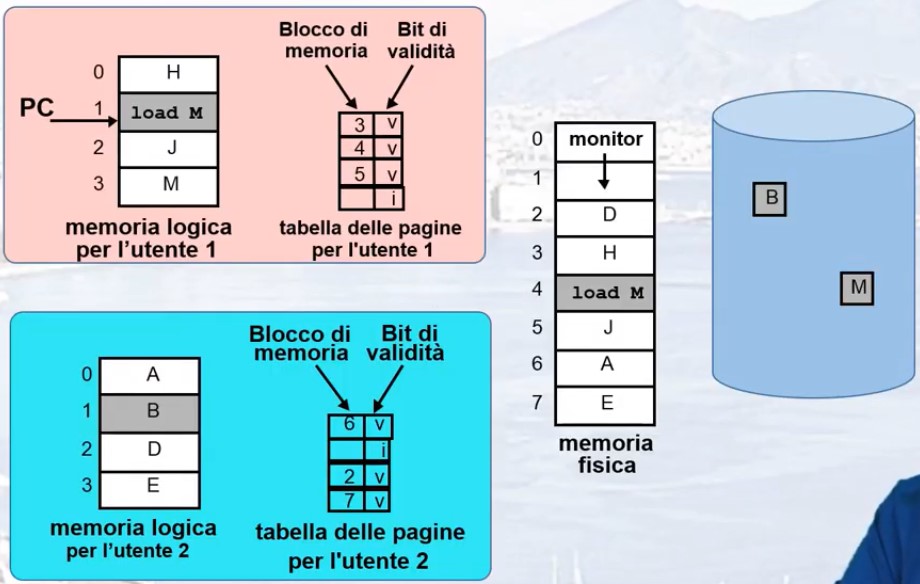
 **spiegazione**

ci sono 2 memorie virtuali di 2 processi A e B. Entrambi incidono sulla memoria fisica, ed è possibile che esse vadano ad incidere anche sul disco (file su disco). Infatti i file su disco (che ho in questo caso), che sono associati a locazioni della memoria fisica, e quando dalla memoria virtuale si va ad incidere sulla memoria fisica, in realtà grazie a tale associazione effettuata dal SO, la memoria fisica automaticamente va ad incidere sul disco.

Il vantaggio che si ha è che si utilizzano le primitive di scrittura ridotte, quindi si va ad utilizzare tutto il set di primitive della scrittura sulla RAM, e poi eventualmente il set di operazioni system call sul disco fisso, cioè non si avranno 2 classi distinte e separate, ma si utilizzerà un numero ridotto di system call, questo grazie a tale associazione.

Sostituzione delle pagine

Quando bisogna scegliere una pagina che sta nella memoria centrale, per buttarla fuori da lì e fare spazio ad un’altra pagina che deve essere inserita nella RAM, si deve studiare una serie di strategie che permettono di individuare la pagina meno importante. Tutto ciò viene rappresentato attraverso **una necessità di sostituzione delle pagine:**

**spiegazione**

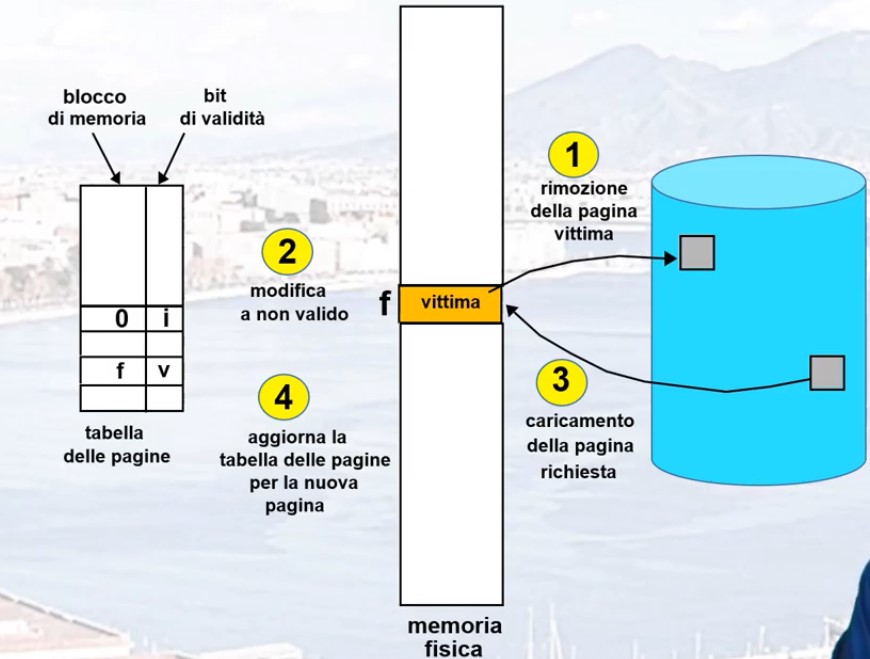
in alto a sx ci sta un processo composto da un program counter, delle istruzioni nella memoria logica e una tabella delle pagine con bit di validità. Sotto a sx ci sta un altro processo composto allo stesso modo. A dx ci sta una memoria fisica, che in qualche modo deve gestire l’uso delle pagine richieste, sapendo: quali sono le richieste dei rispettivi processi, quali sono le pagine che sono caricate effettivamente in memoria, e quali invece sono in memoria ausiliaria e devono essere caricate in memoria fisica.

**Schema di base**

1. Si individua la locazione nel disco della pagina richiesta.
2. Si cerca un blocco di memoria libero:

* Se esiste, lo si usa;
* Altrimenti, si impiega un algoritmo di sostituzione delle pagine per scegliere un blocco di memoria “vittima”

1. Si scrive la pagina richiesta nel blocco di memoria appena liberato; si modificano le tabelle delle pagine e dei blocchi di memoria.
2. Si riavvia il processo utente

**Sostituzione di una pagina**

**Spiegazione**

C’è una tabella delle pagine, con relativo bit di validità. Poi si individua una vittima nella memoria centrale, si rimuove e la si posiziona in hard disk; da quest’ultimo poi si carica la pagina richiesta in memoria centrale. In fine si riaggiorna la tabella delle pagine e si riavvia il processo.

Algoritmi di sostituzione delle pagine

* Si desidera la minor frequenza di assenza di pagine.
* Si valuta l’algoritmo su una particolare successione (stringa) di riferimenti (reference string) e si calcola il numero di assenze di pagine su quella successione.

**Reference string:** sta a significare il fatto che nel momento in cui si sta eseguendo un processo Pi, la sua successione non è in ordine di pagine (1, 2, 3 …), ma è stabilita da questa string reference

**Numero delle assenze di pagine rispetto al numero dei blocchi di memoria**

**spiegazione**

esiste un certo rapporto tra numero si assenze di pagine **(page fault),** rispetto all’ampiezza dello spazio disponibile, ovvero, il numero dei blocchi di memoria. Più grande è la memoria centrale, più difficile è avere un page fault, e quindi aumenta la performance del SO ( non deve fare molte operazioni per liberare spazio per caricare e scaricare oggetti)

1. Algoritmo FIFO

Si butta fuori la prima pagina che entra. Questo algoritmo è facile da implementare, ma è tutta qui la sua forza.

**Spiegazione**

**Primo caso:** la RAM ha messo a disposizione solo 3 blocchi di memoria. Si fanno scambi continui; infatti inizialmente i blocchi sono vuoti, poi quando vengono richiamate si inseriscono le prime tre chiamate (1, 2, 3). Ora lo spazio è pieno, ma se il codice richiede una pagina che non è contenuta in nessun blocco (come in questo caso il 4), si va a togliere il primo ad essere entrato, quindi il più ansiano (in questo caso 1), e cosi via fino alla fine.

In questo caso, il numero si assenze è 9 (si contano pure le istanze iniziali, blocchi vuoti).

**Secondo caso:** Ci sono più blocchi (in questo caso 4), in teoria in numero di assenze dovrebbe essere diminuito, **ma si ottiene una cosa strana**, ovvero, **10 assenze.** Tale stranezza si chiama **ANOMALIA DI BELLADY,** di cui soffre questo algoritmo**. Questa anomalia consiste nel fatto che anche assegnando più memoria, si peggiorano le prestazioni;** infatti in questo esempio con 4 blocchi, si ottengono 10 assenze di pagina, il che non rispecchia il grafico di prima.

**Si ha questa anomalia perché il FIFO, non è un algoritmo intelligente dato che non butta fuori dai blocchi ciò che non serve, ma ciò che è più vecchio, ma si sa bene che le prime pagine contengono informazioni utili a tutto il codice, quindi sono le più importanti.**

**Esempio FIFO Curva delle assenze dell’algoritmo FIFO**

Questo grafico fa capire l’anomalia di Bellady, dato che la curva non scende sempre, ma ad un certo punto sale.

1. Algoritmo ottimale (OPT)

Questo è l’algoritmo migliore che si possa ipotizzare. Si sostituisce la pagina che non si userà per il periodo di tempo più lungo.

**spiegazione**

Ci sono 4 blocchi. Si caricano le pagine secondo la stringa di riferimento, 1, 2, 3, 4, poi sono richieste 1, 2, che già ci sono, poi la 5, dovendo togliere qualcosa, si toglie ciò che meno si utilizzerà, tra ciò che occupa i blocchi; in questo caso quello che si utilizza più lontano è la pagina 4, che sarà quindi la **“vittima”; e così via fino alla fine,** ma arrivati alla fine cioè all’ultima pagina, la 5, si può sostituire un blocco qualsiasi.

* **Il problema è che non è possibile conoscere la successione di riferimento in anticipo.** L’unica cosa che resta da fare è trovare una strategia che sfrutti tale criterio di utilizzo, questo perché la sequenza di riferimento serve solo per fare una statistica/conteggio, **dato che il criterio di ottimalità è solo un FATTO TEORICO, perché la** **sequenza non si conoscerà mai.**

**Esempio OPT**

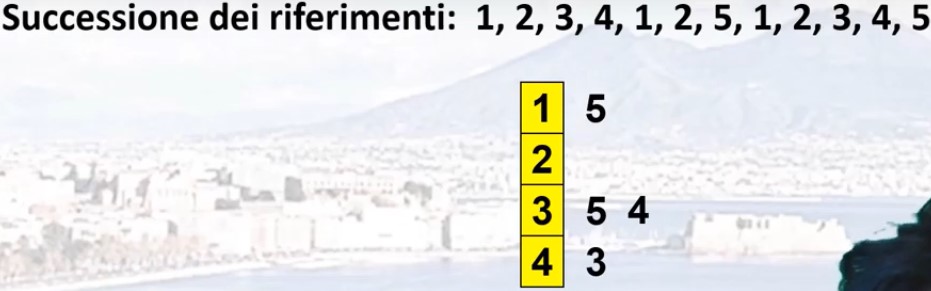
****

1. Algoritmo LRU (least recently used) Dato che l’OPT è un algoritmo teorico e quindi non può essere utilizzato, si usa un’approssimazione LRU.

Si prende come vittima la pagina che è stata utilizzata più lontanamente possibile.

Più volte si è parlato di **principio di località.**

* **Principio di località:** è un principio spazio-temporale, che sta a significare che se si ragiona in generale su uno spazio e su un tempo molto ristretto (a seconda di su cosa si sta lavorando), si potrebbe dire che **l’immediato futuro è simile all’immediato passato.** Questo è il principio che si usa qui, infatti se una pagina è stata utilizzata (vedendo i vari riferimenti che si hanno delle pagine) molto tempo prima, si presume che tale pagina, non la si userà nemmeno nel futuro immediato.
* Dal punto di vista algoritmico, rispetto all’OPT, cambia che anziché **guardare in avanti, come l’OPT, che guarda il futuro (molto avanti) in quanto guarda quello che in avanti sarà usata meno; LRU invece guarda all’indietro,** cioè cerca di indentificare la vittima e toglierla, guardando al passato, e quindi come detto prima si toglie quella che è stata usata più lontanamente nel passato.

**spiegazione**

si caricano le pagine 1,2,3,4 senza problemi, continuo e arrivo alla 5 che non è presente, quindi si sceglie la vittima, tra le prime 4 nei blocchi; si guarda quindi all’indietro e si vede che la pagina usata più vecchia è il 3, e cosi via fino alla fine della successione.

**Esempio LRU**

****

**Per implementare questo algoritmo ci sono diversi metodi:**

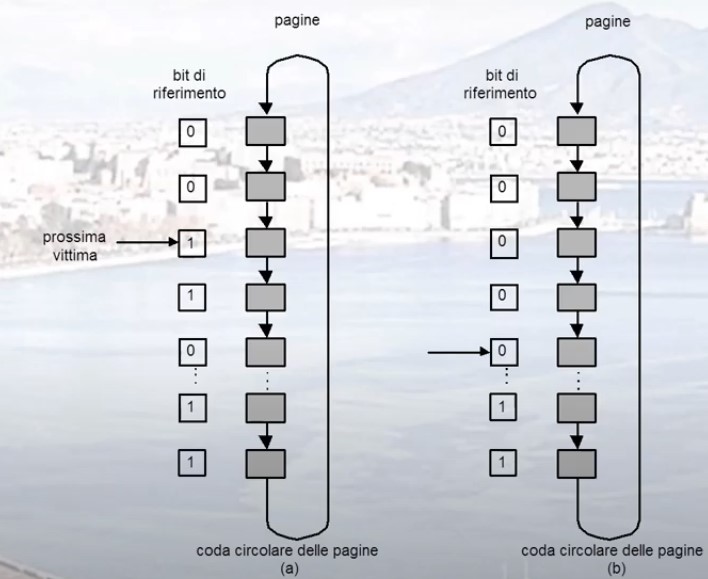
1. **CONTATORI:** ogni volta che la pagina viene eseguita si assegna un **contatore** che viene incrementato ogni qualvolta questa viene richiamata. Quando si deve decidere la vittima, basta vedere la pagina con il contatore più basso.
2. **STACK:** **si ottiene usando una lista doppiamente concatenata, con un puntatore all’elemento iniziale e uno a quello finale.** Ciò che avviene è: si estraggono le pagine man mano che sono referenziate e si mettono in cima allo stack. Nel caso peggiore è necessario modificare 6 puntatori. **Per la sostituzione non si deve compiere alcuna ricerca.**

**Esempio stack**

**spiegazione**

inizialmente lo stack è vuoto. Vedendo la successione si inseriscono la 4 poi la 7 e la 0. Dopo di che nella successione ci sta di nuovo la 7, ma quello che si deve fare non è nient’altro che prendere la 7 e spostarla in cima. **Così facendo in cima si avrà sempre l’ultima pagina usata e in fondo la pagina usata meno recentemente (la più vecchia).**

Sostituzione delle pagine per approssimazione a LRU

* **Algoritmo Bit di riferimento**
* A ciascuna pagina è associato un bit, inizialmente a 0
* Ogni volta che si fa riferimento a quella pagina, il bit è impostato a 1
* Si sostituisce quella che è a 0 (se esiste). Non è però possibile conoscere l’ordine d’uso.
* **Algoritmo seconda chance**
* Occorre il bot di riferimento
* Un puntatore indica qual è la prima pagina da sostituire.
* Se la pagina da sostituire (in ordine di clock) ha il bit di riferimento = 1, allora:
* Imposta il bit di riferimento a 0
* Lascia la pagina in memoria
* Sostituisce la pagina successiva (in ordine di clock), soggetta alle stesse regole.

Sostituzione delle pagine basata su conteggio

* Mantiene un contatore del numero di riferimenti che sono stati fatti a ciascuna pagina
* **Algoritmo LFU (Least frequently used):** sostituisce la pagina con il conteggio più basso
* **Algoritmo MFU (Most frequently used):** è basato sul fatto che, probabilmente, la pagina con il contatore più basso è stata appena inserita e non è stata ancora usata.