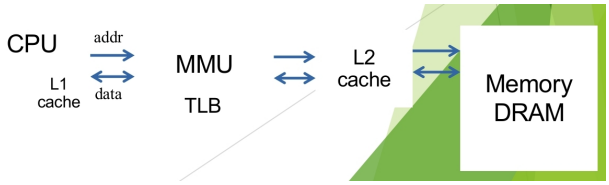
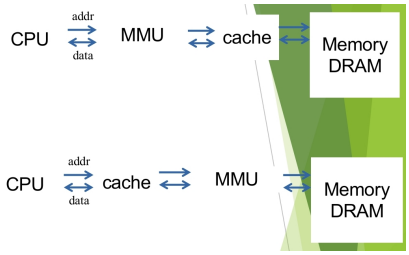
Cache della memoria vs Memoria virtuale

Cache basata su:

- ***INDIRIZZI FISICI***:
  - 

  - ***vantaggio***: **Non è necessario invalidare la cache** ad ogni cambio di contesto (ad ogni nuova tabella delle pagine) perchè, visto che si lavora su indirizzi fisici, non importa quale sia il processo che sta usando quell'indirizzo
  - ***svantaggio***: sil ha che **MMU si trova nel mezzo** fra *CPU* e *CACHE* ed è uno svantaggio perchè l'idea di usare la cache serviva per velocizzare. Se c'è MMU nel mezzo, le operazioni sono più lente
  - L1 è interna alla CPU e L2 è quella vicinia e in mezzo alle due si trova MMU e L1 non contiene MAI indirizzi fisici mentre quelle di livello superiore sì
- ***INDIRIZZI VIRTUALI***:
  - 

  - ***svantaggio***: **E' necessario invalidare la cache** per ogni *context switch* (`operazione lenta`) perchè l'indirizzo virtuale potrebbe indiicare **indirizzi di memoria fisici appartenenti a *DIVERSI PROCESSI*** e ciò potrebbe creare dei problemi per quanto riguarda *l'esecuzione*.
  - *Il SO coopera con la TLB affinchè ogni singolo processo trovi la sua corretta memoria assegnata*
  - MMU si trova dopo la cache e quindi prima si va a ricercare nella cache ecc...che velocizza gli algoritmi
- Generalmente L1 è basata su indirizzi virtuali mentre L2 e successive cache sono basate su indirizzi fisici

Page fault

Vuol dire che la pagina non è all'interno della TLB. In questo caso si deve sostituire un frame associato a una pagina per associare la nuova pagina se la TLB è piena

Quale pagina si rimuove? SI sceglie una ***VITTIMA*** da rimuovere per dare spazio alla nuova pagina

La tabella delle pagine ESTERNA è realizzata per ogni processo e contiene tutte le informazioni sulla posizione di ciascuna pagina virtuale. Viene realizzata solo in caso di *PAGE FAULT*.

> L'**obiettivo**, comunque, è quello di minimizzare il numero di page fault in futuro
> *Basta capire e individuare quali sono le **pagine più utilizzate***

- *Può capitare anche che i frame disponibili sono FULL e per tale motivo, allora, si deve disaccoppiare l'indirizzo del frame con il relativo indirizzo virtuale*
- Se il **bit di modifica della pagina** da rimuovere è **1**, allora **NON** si può rimuovere.
- In base a quale pagina si sta togliendo, si possono generare successivi page fault concatenato
- *IN OGNI CASO*, una pagina **deve essere assegnata** a un processo

Algoritmo della soluzione ottimale

Allora la **SOLUZIONE OTTIMALE** (*IDEALE*) e teorica è l'uso di un **algoritmo ottimale** (***OPT***):

- si deve scegliere la pagina da rimuovere che verrà referenziata in un furuto più lontano
- è **ottimale** ma **difficile da realizzare**
- dà comunque l'idea ottimale da seguire e un *criterio di paragone* con le altre tipologie di algoritmi
- L'**idea** è: Scegliere la pagina che verrà referenziata nel futuro più lontano possibile analizzando il numero di istruzioni che contiene (quindi più istruzioni ha) più tempo ci metterà per chiamare in causa tale pagina. Quindi creerà un **page fault nel lontano futuro**

Le ***DUE NOZIONI CHIAVE*** per gli algoritmi di sostituzione (*che hanno lo stesso obiettivo: scegliere la pagina che, con MOLTA probabilità, **evita i page fault nel breve futuro***, quindi lo ritarda) sono:

- la pagina da rimuovere è quella che ***non è stata usata recentemente***
- **fra 2 pagine** non usate recentemente, sostituire quella ***più anziana*** fra le due (confrontando il "*quando è stata usata l'ultima volta*", quindi referenziata) controllando, questa volta, il `bit di modifica`

Come si identifica se è stata referenziata recentemente o no? Si usano i 2 bit: `referenziamento` *e di* `modifica`.

- Ogni volta che una pagina è **chiamata in causa** (*lettura*), allora il **bit di referenziamento** viene posto a `1`

- E' *impossibile* che una pagina venga modificata senza che sia mai stata referenziata, pertanto il bit di modifica viene messo a 1 se la pagina è modificata

> *I **due bit di stato** sopra descritti servono per capire se una pagina è stata **UTILIZZATA** o meno.*

Algoritmo Not Recently Used (*NRU*)

Ad ogni clock, tutto ciò che non è stato chiamato in causa avrà bit di referenziamento a 0. in questo caso si può differenziare fra una pagina referenziata da poco o da molto:

- Se ***refernziata nell'ultimo clock*** allora il suo bit di referenziamento sarà a 1
- Se ***NON refernziata nell'ultimo clock*** allora il suo bit di referenziamento sarà a 0

Le possibili casistiche di una pagina per quanto riguarda la referenziazione sono:

- referenziata un ***"sacco"** di tempo fa*: bit referenziamento = 0
- referenziata in ***quell'istante***: bit referenziamento = 1

Vengono distinte ***4 CLASSI***, in base ai 2 bit citati sopra:

- **classe 0**: non referenziato, non modificato;
- **classe 1**: non referenziato, modificato; (*modificato ma referenziato molto tempo fa*)
- **classe 2**: referenziato, non modificato;
- **classe 3**: referenziato, modificato
- La pagina che l'algoritmo **sostituisce**, di base, è quella di **classe 0**. Se non è presente la classe 0, allora viene presa in considerazione la classe 1 e così via.
- Le classi con "indice più alto(3)" devono essere salvaguardate rispetto a quelle con "indice più basso (0)".
- In caso di ***PARITA' di classe fra 2 pagine***, viene scelta **UNA PAGINA *CASUALE*** da scartare visto che, per l'algoritmo, le pagine sono esattamente **EQUIVALENTI**
- Se la pagina è modificata, allora, **prima di essere rimossa**, deve essere copiata in memoria. Di conseguenza si ha un *accesso in memoria* e quindi una "*perdita di tempo*"

> *Formalmente*: Viene scelta una pagina dalla classe non vuota di numero più basso
> Questo algoritmo è semplice da implementare e dà dei risultati abbastanza accettabili

Algoritmo FIFO

Si usa una normalissima ***CODA***. Le nuove pagine vengono messe in coda mentre quelle da rimuovere sono le prime arrivate e meno usate, quindi in testa.

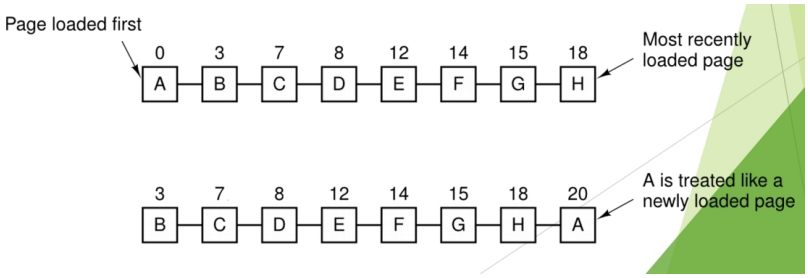
- Si va a rimuovere la pagina chiamata molto tempo fa senza tenere conto se è stata richiamata recentemente oppure no

> *Questo approccio non fa distinzione: la testa indica la pagina chiamata per prima rispetto alle altre ma non dice quando è stata richiamata l'ultima volta.*

- FIFO rimuove la pagina più vecchia e ***non tiene conto della referenziazione nell'ultimo intervallo di tempo*** (quindi magari ***MOLTO USATE***) e per questo motivo l'agoritmo, nonostante sia semplice, non è da implementare

Seconda Chance

- Si prende l'idea della FIFO e si ***prende in considerazione il bit di referenziazione***.
- Si prende la **pagina in testa** e si **controlla** tale bit.
  - Se è 0 si può rimuovere
  - altrimenti non si può rimuovere e quindi si mette la pagina in coda (e resettato il bit di referenziazione) e si va avanti finchè non si trova il bit di referenziamento a 0
- Viene scartata la ***pagina più vecchia*** fra quelle non referenziate di recente



Si devono gestire tutti i possibili casi, ma in questo algoritmo ci possono essere delle condizioni che riportano alla FIFO:

- Se ***tutte le pagine sono referenziate***, si torna alla FIFO dove ogni pagina ha il bit `R` (referenziazione) a 0
- In questo caso si ha lo stesso problema di prima
- *Ciò potrebbe essere possibile se i processi in esecuzione sono molto veloci*

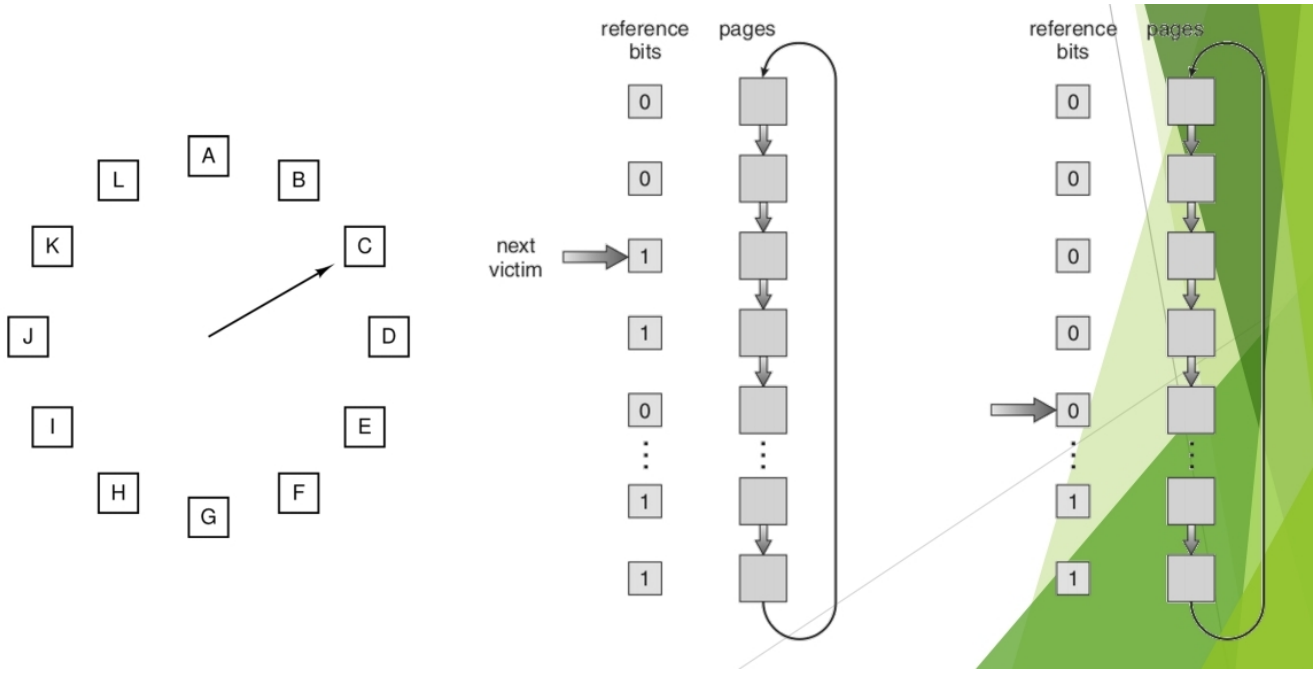
In alcuni casi, questo algoritmo degenera al limite di FIFO (quindi l'algoritmo diventa proprio *FIFO*):

- Si usa invece l'algoritmo ***SECONDA CHANCHE*** una **CODA CIRCOLARE**, chiamato ***ALGORITMO CLOCK***
- In questo caso non si risolve il problema FIFO ma lo attenua

Algoritmo Clock

Non risolve il problema che può emergere dall'algoritmo *Seconda Chance* ma lo migliora a ***livello di tempistica e di algoritmo in di per sé***.

Questo è quello più **robusto** ed **efficiente**



Se si hanno più pagine con bit `R=0` appartenenti a una specifica classe:

- Alcune pagine sono state referenziate in un tempo recente e alcune in un tempo più lontano
- Fra 2 pagine referenziate nel lontano passato, si deve scegliere quella ***PIU' ANZIANA*** fra entrambe
- *Non si hanno informazioni sull'esatto momento di chiamata in causa*
- Con l'algoritmo che si basa solo su bit R non permette di distinguere fra una pagina referenziata 2 clock fa e una 4 clock fa (*per esempio*)

Si usa Least Recently Used

Algoritmo Least Recently Used (*LRU*)

Fra 2 pagine non usate di recente, si cerca di eliminare la pagina usata nel tempo più lontano fra le due.

> *Si basa sull'idea: le pagine più recentemete usate hanno maggior probabilità di essere chiamate in causa, e quindi referenziate.*

Un ***PRIMO METODO*** è quello di usare un contatore:

- Si basa su un ***CONTATORE*** (*nella **CPU***) usualmente di una lunghezza di 64 bit: ogni volta che si esegue l'istruzione, **questo contatore viene incrementato**. Quando una pagina viene referenziata, il contatore viene copiato in un apposito campo della pagina.
- Ogni volta che referenzio una pagina, si copia il contatore di che è stato calcolato in quell'istante su quella pagina
- In caso di PAGE FAULT:

- Si vanno a **confrontare i contatori** alle pagine e il ***CONTATORE PIU' BASSO*** numericamente rappresenta la pagina più vecchia ***DA RIMUOVERE*** perchè vuol dire che quella *pagina è quella che è stata chiamata in causa di meno rispetto a tutte le altre* (*parlando di referenziamento*)

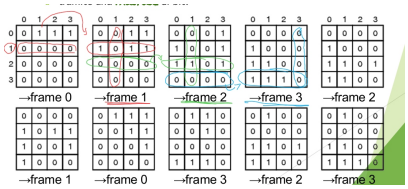
Un ***SECONDO METODO*** è quello di usare una ***MATRICE DI BIT***:

- E' grande NxN dove N = *numero di frame*
- L'idea è: inizialmente le matrici sono tutte inizializzate a 0
  - ogni volta che una pagina è referenziata (corrispondente a un frame `k`), il frame corrispondente viene così settato:
    - la k-esima **riga** di quel frame viene settata a **1**
    - la k-esima **colonna** viene settata a **0**
  - In caso di page-fault, la riga più bassa sarà ***quella meno utilizzata***, cioè ***quella riga dove saranno presenti più zeri***

> *Se più zeri sono presenti nella riga 2, allora la pagina di indice 2 è quella da rimuovere*

Per gestire i contatori serve un **supporto hardware opportuno** e si tratta di **operazioni dispendiose** (soprattutto per le ***matrici utilizzate***, che risultano piuttosto ***enormi***)

Esempio



Algoritmo Not Frequently Used (*NFU*)

Si usa l'idea di LRU(approssimazione di LRU) usando una ***VERSIONE PIU' SOFTWARE*** LRU: è gestito di più dal punto di vista software e usa proprio un contatore ***SOFTWARE***

- A ogni pagina viene **sommato al contatore** il valore del bit di referenziamento
- In questo modo, periodicamente, si ha il bit `R` il quale viene sommato al contatore
- Quando si ha un page-fault, la pagina da sostituire è quella che avrà il ***CONTATORE PIU' BASSO*** numericamente

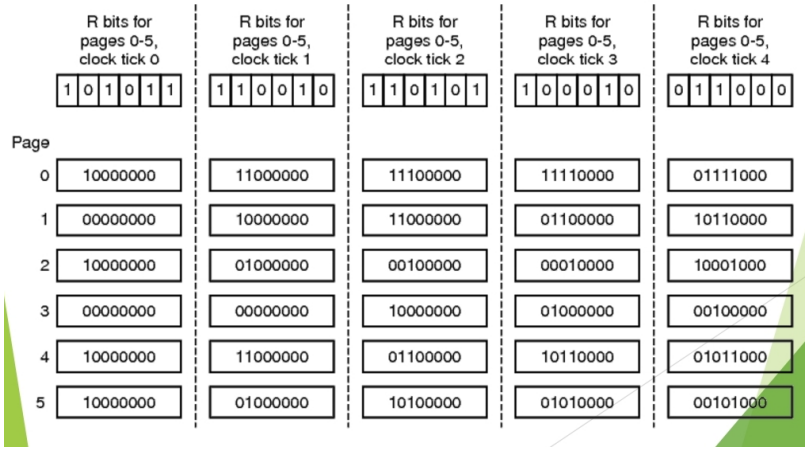
Problema di questo algoritmo: esso dipende dal tempo di esecuzione dei processi ed inoltre **può privilegiare** (*erroneamente*) **pagine molto usate nel passato** rispetto a quelle usate di recente e questo problema si può risolvere nel seguente modo:

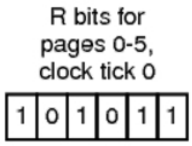
- SI usa l'algoritmo di AGING usando un meccanismo di contatori.
- Fa solo 2 modifiche:

- Si ha un contatore (*sequenza di bit*) e si **SHIFTANO** i suoi bit **verso destra** di una posizione
- Dopo lo shift **si libera il bit più significativo** e proprio in quella posizione **si copia il bit `R` di referenziamento**

- In questo modo si distinguono i tempi esatti di referenziamento di ogni singola pagina

Algoritmo di Aging



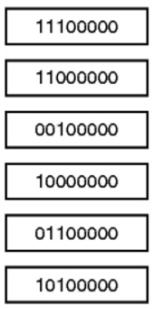


Guardando la sequenza di bit in alto a sinistra:

- nella posizione i-esima si ha la pagina i-esima che è stata referenziata (bit=1) oppure no (bit=0)

Leggendo *dall'alto verso il basso* l'immagine sopra:

- La sequenza **più in alto** viene **shiftata a destra** di una posizione
- nella posizione `0` si mette il `bit R` (*presente nella posizione i-esima della pagina* `i`)



- In caso di *page-fault* in tick 2 (*clock 2*) , allora **la pagina da rimuovere**, guardando quelle disponibili in colonna, ***è quella che ha una sequenza di bit che compone un numero decimale più basso***.

In caso di una pagina con sequenza di bit `0010000` vuol dire che è stata chiamata in causa *3 clock fa* e si capisce **dal numero di 0 più significativi** (*quindi a sinistra della sequenza*)

> *Fra 2 pagine non recentemete usate, si identifica quella ancora più vecchia fra entrambe*, cioè quella che ha più zeri a sinistra perchè, in questo caso, ***più il contatore è basso*** (confrontando i ***NUMERI BINARI***)

In particolar modo..
Clock 0:

- Le pagine referenziate sono quelle di indice 0,2,4,5. (*si legge dall'array più in alto di tutti*)
- Si prende l'array che è pieno di zeri (per ogni pagina)
- Considerando un singolo array nella pagina 0:
  - Si shifta a destra di una posizione
  - Si copia il valore i-esimo e, visto che è stata usata, allora si avrà `10000000`

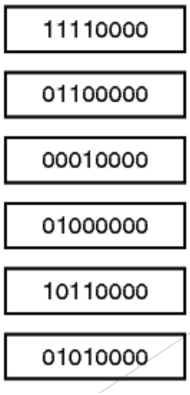
Clock 1:

- Le pagine referenziate sono quelle di indice 0,1,4;
- Si prende l'array di prima corrispondente per una singola pagina, nell'esempio è la pagina 0:
  - Si shifta a destra e si ottiene `01000000`
  - Nella posizione 0 dell'array sopra, si copia il bit di referenziamento per quella specifica pagina (che è 1)
  - Si ottiene `11000000`

Clock 2: *ecc....*

Clock 3 -> Page fault:

- Si ha la seguente situazione:



- In questo caso è da rimuovere quella con "*valore più basso*", quindi quella più vecchia
- Verrà rimossa `00010000`, quindi la pagina di *indice 2*

> ***Solo l'AGING tiene in considerazione l'ATTIVITA' PASSATA di ogni singola pagina***