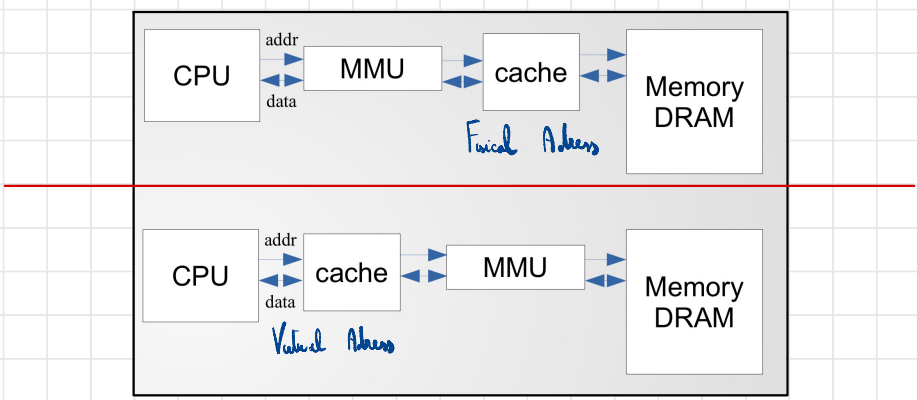
Memoria Cache

Si tratta di una memoria interposta tra CPU e memoria RAM. Nasce una possibilità di decisione su dove mettere la cache in relazione alla posizione dell'MMU, anch'essa interposta tra CPU e RAM.

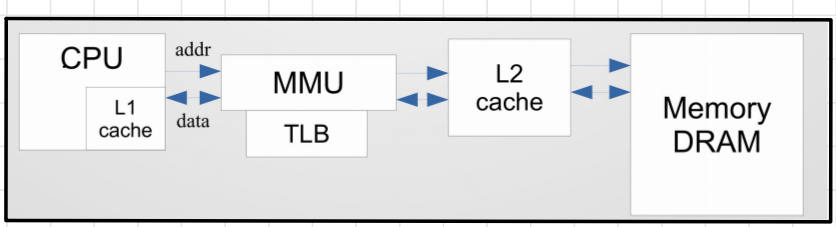
La cache utilizza come chiavi di ricerca indirizzi di natura differente sulla base della posizione rispetto all'MMU, in particolare interporla prima implica che essa lavorerà con indirizzi virtuali, mentre interporla tra MMU e RAM implica che la traduzione dell'indirizzo sia già stato tradotto e quindi la ricerca avviene con indirizzi fisici.

Cache basata su indirizzi fisici : all'interno della cache vengono mischiate informazioni riguardanti diversi processi senza problemi di conflitto, in quanto l'informazione viene ricercata sfruttando l'indirizzo fisico e quindi non si presentano ambiguità. Questa cache si occupa solo di rendere accessibili delle informazioni alla CPU senza dover necessariamente accedere alla RAM, diminuendo il costo di accesso.

Cache basata su indirizzo virtuale : in questo caso viene ricercata una voce nella cache sfruttando l'indirizzo virtuale. Questo tipo di ricerca causa problemi di univocità in caso di context switch, al pari della TLB. Il processo che ha sostituito il precedente se trova riscontro in una ricerca fatta in cache, farà riferimento ad informazioni relative al processo precedente. Anche qui il problema si risolve (come in TLB) facendo cache flush. Azzerando la cache si hanno problemi di cache miss e quindi anche qui una soluzione è data dall'arricchire la chiave di ricerca aggiungendo l'ASID (#PV, PID) del processo cui appartiene una determinata linea di cache.



In termini di efficienza e costi la cache basata su indirizzi virtuali scala male al crescere delle dimensioni richieste.
La soluzione risiede nell'interporre una cache di primo livello tra CPU ed MMU e quindi farla lavorare con indirizzi virtuali, ed una cache di secondo livello tra MMU e RAM per consentirle di lavorare con indirizzi fisici.
Si pensa di rendere le cache grosse basate su indirizzi fisici, in quanto prevedono minori costi e le cache che prevedono dimensioni minori le si interpongono tra CPU ed MMU e quindi basate su indirizzi virtuali.



Durante la traduzione da indirizzo virtuale a fisico di necessita di consultare la tabella delle pagine e se il bit di assenza indica che la pagina richiesta non si trova in memoria, bisogna prelevare la word dal disco.
In caso di page fault si ricaricano le informazioni mancanti in RAM e si necessita di reiterare l'istruzione interrotta che stavolta riuscirà a terminare la propria esecuzione. Si nota che durante l'esecuzione di un'istruzione si potrebbero verificare diversi page fault, per esempio a causa della presenza di diversi operandi nell'istruzione. Infatti una sola istruzione potrebbe comportare vari fetch e quindi potenziali page fault.

L'MMU si accorge che avviene un page fault in quanto non riesce ad effettuare la traduzione con successo, mentre la gestione del page fault è a carico del sistema operativo, in quanto a livello kernel viene svolta un operazione che legge dal disco e carica in memoria la pagina che ha causato il page fault. Se la **tabella dei frame** non ha spazio per ospitare il nuovo frame si necessita di spostare sul disco un frame dalla tabella dei frame, quindi il processo proprietario del frame spostato su disco si vedrà penalizzato. Se l'algoritmo di sostituzione ha fatto una scelta pessima allora l'informazione viene richiesta subito dopo, ma anche qui l'evento viene gestito come prima.

Algoritmi di sostituzione delle pagine

La differenza principale con gli algoritmi di scheduling risiede nel fatto che gli algoritmi di sostituzione delle pagine sono gestiti a livello software. Bisogna effettuare una scelta sulla pagina che verrà spostata su disco nel caso in cui la tabella dei frame non possa ospitare la pagina dal disco. Una scelta pessima comporta che poco dopo una istruzione prelievi la stessa pagina che è stata appena spostata sul disco e quindi si dovrà ripagare il costo dovuto all'accesso ad una memoria lenta. La scelta dovrà **minimizzare il numero di page fault futuri** dovuti alla pagina spostata, ad esempio si potrebbe pensare di individuare una pagina che non verrà più referenziata.

Algoritmo OPT (soluzione ottimale)

L'idea alla base dell'algoritmo OPT è di sostituire la pagina che non sarà utilizzata per il periodo più lungo di tempo futuro. Ad esempio si può pensare di assegnare un etichetta ai processi che indicano il numero di istruzioni che eseguirà la CPU prima di referenziare la pagina. L'algoritmo OPT richiede di conoscere in anticipo **l'elenco completo delle richieste** di pagina future, il che ovviamente non è possibile nella realtà. Tuttavia l'algoritmo OPT può essere utilizzato come **termine di paragone** per valutare le prestazioni di altri algoritmi di sostituzione delle pagine. I risultati di questo algoritmo forniscono una sorta di limite che consente di valutare se un algoritmo di sostituzione sia più o meno lontano dall'ottimo.

L'algoritmo OPT analizza l'elenco delle richieste di pagina future per determinare quale pagina non sarà utilizzata per il periodo più lungo di tempo futuro e sostituisce la pagina identificata al passo precedente con la nuova pagina richiesta.

Algoritmo NRU(Not Recently Used)

L'algoritmo NRU (Not Recently Used) è un algoritmo di sostituzione delle pagine che tiene traccia delle pagine usate meno di recente in modo farà ricadere su queste la scelta di "sacrificio". NRU assegna a ogni pagina una classe di utilizzo, valutando lo stato del bit di **referenziamento** (R) e del bit di **modifica** o dirty bit (W) :

- classe 0 : non referenziato, non modificato R = 0 W = 0
- classe 1 : non referenziato, modificato R = 0 W = 1
- classe 2 : referenziato, non modificato R = 1 W = 0
- classe 3 : referenziato, modificato R = 1 W = 1

L'algoritmo va ad individuare la classe più alta non vuota : vengono considerate più sacrificabili le pagine di classi con numero più basso. Le pagine non referenziate di recente trovano il proprio bit a zero (azzerato ciclicamente dall'MMU) si pensa non verranno referenziate nel breve termine e quindi vengono predilette per essere scartate. Si preferisce in oltre, a parità di stato di referenziamento, scartare quelle non modificate in quanto la copia su disco è aggiornata e quindi si può semplicemente sovrascrivere la pagina dalla RAM senza doverla ricopiare prima su disco.
Se nella classe considerata si trovano più pagine si necessita di effettuare un altra scelta, solitamente di tipo FIFO che prevede lo scarto della pagina più vecchia, perchè per il principio di localizzazione si pesano più le informazioni recenti per fare previsioni future. Si tiene conto del momento di arrivo di una pagina tramite un di timestamp che da informazioni su quando la pagina è stata caricata in memoria RAM. Se tutti i bit hanno lo stesso stato l'algoritmo manifesta un comportamento FIFO.

Algoritmo FIFO, della seconda chance e Clock

- **FIFO** : si usa una struttura dati di supporto alla scelta con tanti nodi quanti sono i frame in memoria. La scelta viene effettuata sulla base dell'età, ovvero il tempo in cui una pagina è stata caricata in memoria. Una coda FIFO tiene conto dell'inserimento di pagine più recenti ed il funzionamento FIFO della coda permette l'estrazione di pagine vecchie in memoria. Una scelta del genere potrebbe scartare una pagina vecchia ma molto referenziata, per tanto si effettua una miglioria: la seconda chance.

- **della Seconda Chance** : un adattamento della versione appena vista che tiene conto del bit di referenziamento, il quale viene considerato andando a pesare sulla scelta. Questo algoritmo di base guarda alla testa della coda ma valutando il bit di **referenziamento** : se è a zero la pagina viene scartata ma se si trova ad 1 esso viene azzerato e la pagina viene reinserita in coda e si riguarda alla testa. Questo algoritmo è ben posto e manifesta lo stesso comportamento di quello FIFO nel caso in cui tutte le pagine si trovino nello stato di referenziamento.

- **algoritmo Clock :** questo algoritmo rappresenta un miglioramento dell'algoritmo della seconda chance più efficiente : si implementa la struttura come una coda circolare. Si ha un puntatore "NEXT" fisso alla testa della coda, quindi alla prossima potenziale pagina da scartare. L'implementazione circolare prevede un netto miglioramento delle prestazioni in quanto nel caso in cui si debba inserire l'elemento in coda non si scorre l'intera coda ma si aggiorna semplicemente la posizione il puntatore NEXT alla prossima pagina da scartare.

Algoritmo LRU(last recently used)

Esiste un'approssimazione dell'algoritmo OPT ottimale ovvero l'algoritmo LRU, esso prevede lo scarto della pagina utilizzata meno di recente. Si tiene conto dell'istante di tempo in cui è stata referenziata per ultima la pagina e non più dell'istante di caricamento in memoria come in FIFO (che scartava quella più vecchia in memoria). L'algoritmo OPT applica lo stesso principio sulle pagine future, infatti scartava la pagina che viene utilizzata nel lontano futuro. LRU guarda il passato più recente per fare supposizioni sui riferimenti che verranno fatti nel futuro prossimo e **scarta la pagina usata meno di recente** in quanto si suppone che non sarà riutilizzata a breve.

- **implementazione con contatore:**

Un'implementazione algoritmica si basa sull'incremento di un contatore monotono crescente ogni volta che si effettua un referenziamento ad un certa pagina. L'algoritmo LRU utilizza un contatore monotono crescente per tenere traccia dell'ultimo accesso a ogni pagina. Questo contatore viene riportato nella tabella dei frame dato che si ha un numero di contatori al più pari al numero di frame che devono essere gestiti ed ogni contatore viene aggiornato con riferimento ad una esatta pagina di memoria. Quando una pagina viene referenziata, la CPU aggiorna il contatore nella tabella dei frame associato alla pagina referenziata. In questo modo, LRU tiene traccia dell'ordine in cui le pagine sono state referenziate e può scegliere la pagina utilizzata meno di recente per la sostituzione.

L'idea è buona ma l'implementazione è dispendiosa in quanto necessita di un'azione svolta a livello hardware (MMU) perchè una soluzione software sarebbe impensabile data la frequenza con cui avviene l'aggiornamento dei contatori.

- **implementazione con matrice di bit:**

Un altra implementazione, che non costringe ad aggiornare sempre i contatori, facendo un utilizzo meno intensivo dell'hardware, prevede che sia implementata una matrice sull'MMU che contiene una riga e colonna per ogni frame. All'inizio la matrice di bit è tutta nulla, se viene referenziata la pagina contenuta nell' i-esimo frame pongo ad 1 tutti i bit nella riga i-esima ed azzero tutti i bit nella i-esima colonna. L'operazione pone un **peso di hamming** pari ad n-1 all'ultima pagina usata (n = numero frame) e le altre pagine si vedono abbassare il proprio peso. La scelta su chi scartare ricade sulla pagina la cui riga contiene il peso di hamming più basso, poichè la meno recentemente utilizzata. Questo modello implementativo tiene comunque conto di un supporto hardware per la matrice anche se ne fa un uso molto meno intensivo.

L'implementazione di LRU richiede in ogni caso un costoso supporto hardware e quindi l'ideale sarebbe pensare ad un'approssimazione via software.

Algoritmo NFU (not frequently used)

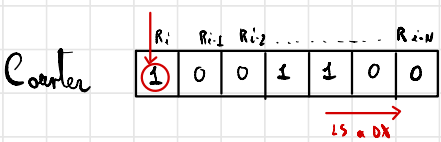
L' algoritmo NFU è un'approssimazione via software dell'algoritmo LRU. Si ha un contatore per ogni pagina presente memoria, ovvero una per ogni frame occupato. Ogni contatore viene aggiornato periodicamente attraverso un clock periodico che innesca una procedura che, prima di azzerare il bit di referenziamento, lo somma al contatore. Come risultato ogni contatore sarà relativo alla frequenza con cui viene referenziata una pagina, dal momento in cui essa è stata portata in memoria. L'algoritmo di sostituzione prevede il sacrificio della pagina col contatore più piccolo, poichè referenziata di meno.

La pecca di questo algoritmo è che pesa in maniera equivalente le pagine referenziate in momenti differenti, infatti le pagine recenti hanno lo stesso peso di quelle passate.

Algoritmo di aging

Una proposta implementativa migliore di NFU è quella dell'algoritmo di aging. Si effettua, alla scadenza di ogni ciclo di clock, uno shift a destra di una posizione della rappresentazione binaria del contatore e si inserisce il bit di referenziamento dell'ultima pagina referenziata nella posizione più significativa del contatore. L'algoritmo di selezione prevede di sostituire ancora la pagina con il valore del contatore minore, in quanto quella pagina sarà anche la referenziata meno di recente rispetto alle altre.

L'algoritmo approssima meglio l'idea dell'LRU poichè, avendo un contatore che tiene uno storico di diversi cicli di clock ed inserento come cifra più significativa il bit R, attribuisce un peso maggiore alle pagine referenziate nei cicli di clock più recenti. Le informazioni già presenti nel contatore al momento del referenziamento della pagina invecchiano e le più recenti sono inserite nella posizione che ha un peso maggiore (la più significativa).



Anomalia di Belady

Si potrebbe intuire che all'aumentare delle dimensioni della memoria RAM si verifichino meno eventi di page fault in quanto la RAM può ospitare più pagine. L'anomalia di Belady è un fenomeno che entra in contraddizione con questa affermazione in quanto all'aumentare della RAM possono verificarsi più page fault.
Gli algoritmi che soffrono di quest'anomalia sono FIFO, Seconda chance, Clock ed NRU, in quanto non sono algoritmi a stack e in casi specifici questi algoritmi riducono al FIFO in alcuni casi particolari.

Proprietà di inclusione

Altri algoritmi di sostituzione come LRU non soffrono dell'anomalia di Belady, in quanto manifesta la proprietà di inclusione, ovvero :

- Per qualunque istante di tempo e per qualunque numero di frame, l'insieme di pagine contenute in memoria avendo n frame è incluso in quello che si avrebbe usando n+1 frame.

Questa proprietà, tipica degli algoritmi a stack, non manifesta l'anomalia perchè le pagine che si hanno in memoria con n frame continuano a restare in memoria anche con n+1 frame. Ovviamente non cambiando i frame che sono presenti in memoria non si influisce sulla scelta dell'algoritmo.

Riepilogo sugli algoritmi

- **OPT**: non implementabile, ma utile come termine di paragone;
- **NRU***: approssimazione rozza dell'LRU;
- **FIFO***: può portare all'eliminazione di pagine importanti;
- **Seconda chance***: un netto miglioramento rispetto a FIFO;
- **Clock***: come S.C. ma più efficiente;
- **LRU**: eccellente idea (vicina a quella ottima) ma difficilmente realizzabile se non in hardware;
- **NFU**: approssimazione software abbastanza rozza dell'LRU;
- **Aging**: buona approssimazione di LRU con implementazione software efficiente.

* soffre dell'anomalia di Belady