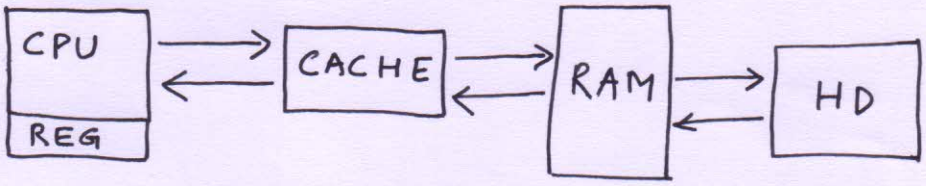
MEMORY HIERARCHY



(a) Data in Cache can be accessed much more quickly than RAM.

(b) The size of Cache is small and fixed.

(c) Sometimes the requested data is already in the cache (HIT).

(d) Sometimes the requested data is not in the cache (MISS)

- A cache maintenance algo decides what to keep and what to evict during a MISS.

<u>Sequence</u> : a b c b c a b

<u>Initial Cache Contents</u> = {a, b}

Ref	Cache Content	H/M	Fetch
a	{a, b}	H	N
b	{a, b}	H	N
c	{c, b}	M	Y
b	{c, b}	H	N
c	{c, b}	H	N
a	{a, b}	M	Y
b	{a, b}	H	N

Schedule S_1 Misses = 2
Fetches = 2

Sequence: a b c b c a b

Initial Cache Contents : {a, b}

Ref	Cache Content	H/M	Fetch
a	{a,b}	H	N
b	{a,b}	H	N
c	{a, c}	M	Y
b	{a, b}	M	Y
c	{b, c}	M	Y
a	{a, c}	M	Y
b	{b, c}	M	Y

Schedule S_2 Misses = 5

Fetches = 5

GOAL: Given a sequence of memory references, design an optimal eviction schedule.

OPTIMAL SCHEDULE:

A schedule is optimal if it incurs as few fetches from the main memory as possible.

GREEDY APPROACH

- Everytime evict the item that is needed farthest in future.

- This eviction schedule is called S_{FF}

- Check whether S_1 is ~~eqal~~ to S_{FF} or not.

- Now we have to prove that SFF is optimal.

- Before proving, we need to understand the difference between a Reduced Schedule and a non-Reduced Schedule.

Reduced Schedule

If # Fetches = # Misses.

eg S_1, S_2

Non-Reduced Schedule

If # Fetches > # Misses

eg as below: Initial Cache = {a, b}

Ref	Cache Contents	~~H~~/M	FH
a	{a,b}	H	N
b	{a,b}	H	N
c	{a,c}	M	Y
b	{b,c}	M	Y
c	{a,c}	H	Y
a	{a,c}	H	N
b	{b,c}	M	Y

Misses = 3

Fetches = 4

Observation

Any non reduced schedule can be converted into a reduced schedule without increasing the number of fetches.

Proof: Be Lazy ....

THEOREM

Let S be a reduced schedule that makes the same eviction decisions as SFF through the first j items in the sequence, for a number j.

Then there is a reduced schedule S' that makes the same eviction decisions as SFF through the the first $(j+1)$ items, and incurs no more misses than S does.

PROOF : S and SFF have same cache contents till j th point.

Say $(j+1)$ th request is for item d

- $d \in$ Cache — S and SFF agrees till point $(j+1)$. $\therefore S' = S$
- $d \notin$ Cache
  - CASE 1
  - CASE 2

CASE 1

Both S and S_{FF} evict the same item to make room for d. Again S and S_{FF} agree till point $(j+1)$ and

$\therefore S' = S$

CASE 2 S evicts f, but S_{FF} evicts e s.t $e \neq f$.

Now after $(j+1)$ th step:

S has Cache Contents $(e + X)$

S' or S_{FF} has Cache Contents $(f + X)$

From step $(j+2)$ onward S' should behave in such a way that it does not do more fetches than S

CASE 2 (Continued)

From step $(j+2)$ onward, S' will behave exactly like S Untill One of the following happens.

(a) There is a request for $g \neq e, f$ s.t $g \notin$ Cache

- If S evicts e, then S' evicts f.
- The Caches of S and S' becomes equal, and S' behaves like S there after.

(b) There is a request for f, and S evicts e to bring in f. In this case S' does nothing. S and S' have same Cache.

(c) There is a request for f and S' evicts e' (s.t $e' \neq e$) to bring in f.

- S' will evict e' to bring in e.
- S and S' both have same cache contents hereafter. However S' now is a non-reduced scheduled.
- Convert S' into its reduced form without increasing the # fetches.

(d) There is a request for e. This case is not possible, because S_{FF} has evicted e which means that it is requested after f.

THE PROOF ENDS.

COROLLARY: Suppose S^* is the optimal schedule which has ~~first~~ zero steps common with S_{FF}. We can construct S_1 which has first step common with S_{FF} and doesn't incur more fetches than S^*.
From S_1 we can construct S_2
From S_2 we can construct S_3 ...
From S_{n-1} we can construct S_n.