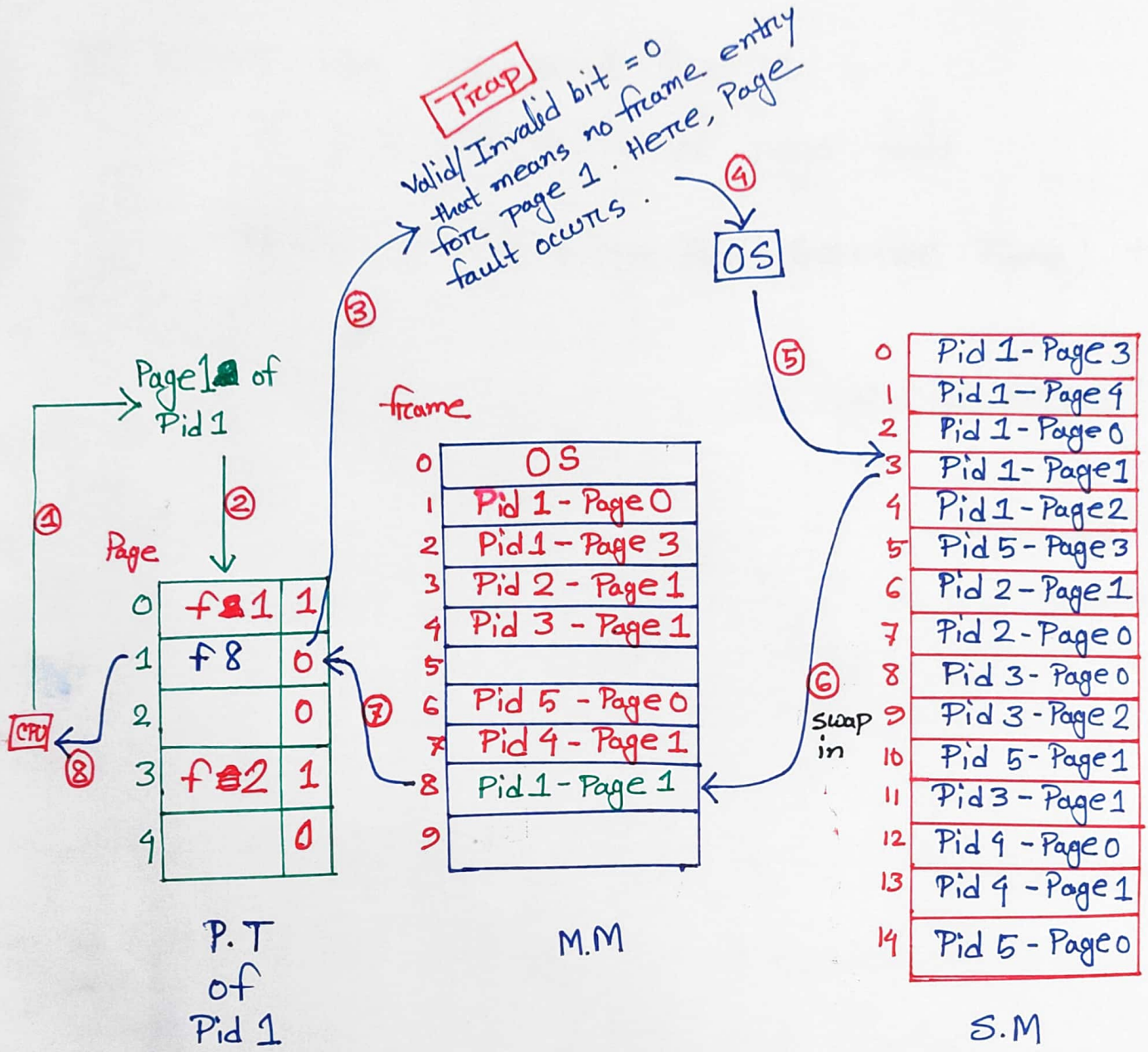
Demand Paging

- In demand paging, programs are not entirely loaded into memory when they start execution. Instead, only the essential portions means some pages are loaded initially. Additional pages are loaded into memory when they are requested or demanded by the CPU.

- When a requested page is not in the M.M, a page fault occurs. The OS responds to the page fault by fetching the required page from S.M to M.M. This process is known as page swapping or page replacement.



When Page Fault occurs, a trap will be generated When trap is generated, system control will be transferred from User to OS.

During step 8, we will let the CPU know that Page Fault service Time is over and requested page is brought to the M.M from S.M by OS. After this step system control will be transferred from OS to User again.

EMAT For Demand Paging:

if P = probability of page fault

EMAT = (P * Page Fault Service Time) +

(1-P)*(M.M. Access Time)

if no page fault, then we found the requested page in M.M. So here is M.M Access Time.

→ Page Fault Service Time is in milli-second.

→ M.M Access Time is in nano-second.

→ milli is 10^6 times larger than nano.

→ So Time taken for accessing S.M is significantly higher than M.M access time.

FIFO Page Replacement Algo

1) Maintain a FIFO queue to keep track of the order in which pages are brough into memory.

2) When a page is demanded

→ Check if the page is already in M.M

→ If it is present, no action is needed

→ If it is not present means a page fault occures, check if there is space available in M.M

↳ If there is space available in M.M, bring the page into M.M and add it to the end of the queue.

↳ If there is no space available, remove the page at the front of the queue (the oldest page) to make space and then bring the new page into M.M, adding it to the end of the queue.

3) Continue these processes as new page is demanded

Reference string:

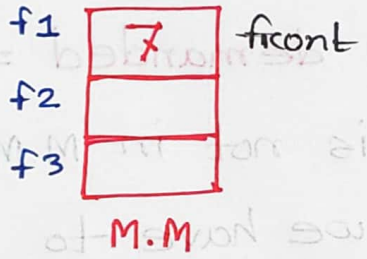
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0

Let, M.M has 3 frames.

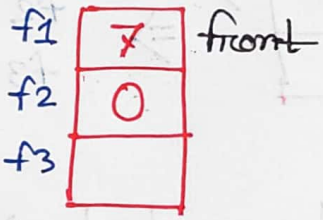
⇒

Ref. string means 7th page is demanded first, then page 0, then page 1, ...

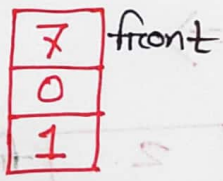
7 is demanded ⇒
initially all frames are empty. so frame 1 stores page 7.



then page 0 is demanded ⇒
f2 and f3 is empty. so f2 contains page 0.

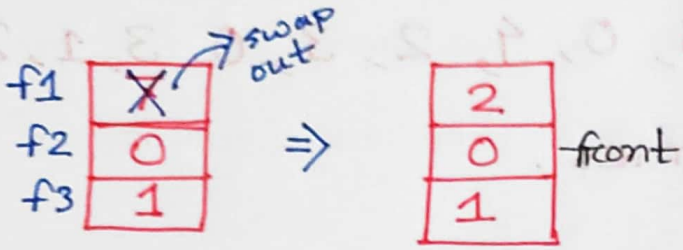


then 1 is demanded ⇒



then 2 is demanded ⇒
no frame is free. so now we have to swap out page from M.M

As FIFO, we will swap out that page which comes first (oldest). Front indicates that.



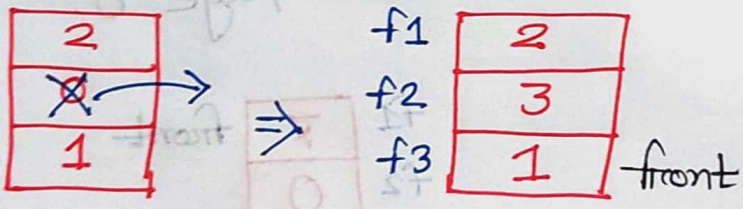
when we remove page 7, front will be ~~4~~ front++

0 is demanded ⇒

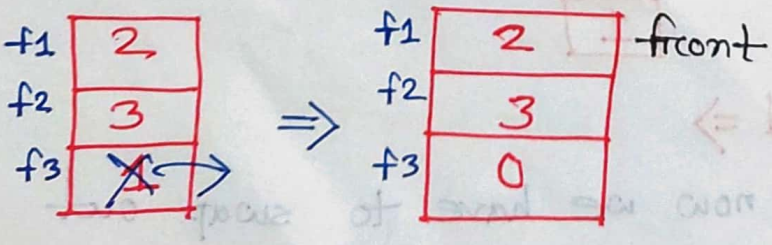
0 page is already in the M.M. So no operation is needed.

3 is demanded ⇒

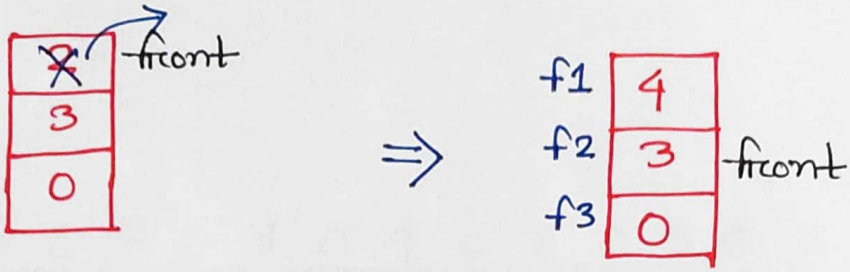
3 is not in M.M & M.M is not free.
So we have to swap out the comparatively oldest page (front).



0 is demanded ⇒



4 is demanded ⇒



2 is demanded ⇒

3 is demanded ⇒

0 ⇒

.
.
.

finally

					Hit						Hit			Hit
	7	0	1	2	0	3	0	4	2	3	0	3	1	2	0
frame 1	7 front	7 front	7 front	2	2	2	2 front	4	4	4 front	0	0	0	0 front	0 front
frame 2		0	0	0 front	0 front	3	3	3 front	2	2	2 front	2 front	1	1	1
frame 3			1	1	1	1 front	0	0	0 front	3	3	3	3 front	2	2

- Adding more frame for a process should theoretically decrease the number of page fault. Because it provides more space for storing frequently accessed data, reducing need to swap data betweeen S.M & M.M.

Right ?

Ref: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

⇒ Apply FIFO Page Replacement algorithm for 3 frames

					Hit		Hit	Hit			Hit
	1	2	3	4	1	2	5	1	2	3	4	5
frame 1	1 front	1	1	4	4	4 front	5	5	5	5	5 front	5
frame 2		2	2	2 front	1	1	1 front	1	1	3	3	3
frame 3			3	3	3 front	2	2	2	2	2 front	4	4

Here, Hit = 3
Page Fault = 9

⇒ Apply FIFO for 4 frames

					Hit	Hit						
	1	2	3	4	1	2	5	1	2	3	4	5
frame 1	1 front	1	1	1	1	1 front	5	5	5	5 front	4	4
frame 2		2	2	2	2	2	2 front	1	1	1	1 front	5
frame 3			3	3	3	3	3	3 front	2	2	2	2
frame 4				4	4	4	4	4	4 front	3	3	3

Here, Hit = 2
Page Fault = 10

So, here increasing the number of frames leads to more page faults. This paradoxical situation is called Belady's anamoly.

Optimal Page Replacement Algo:

↳ theoretical algo

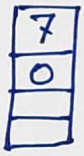
↳ replace the page that will not be used for the longest period of time in future

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

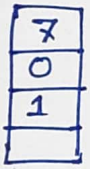
Let's say, we use 4 frames

→ Page 7 is demanded:

→ 0 is demanded:



→ 1 is demanded:



→ 2 is demanded:

→ 0 is demanded: Hit

→ 3 is demanded:

Page 3 occurs page fault. From 4 frames, we will swap out that page which is furthest from this demanded page.

CPU will demand 0 immediately after page 3.
Then among 7, 1, 2; CPU will demand 2
Then among 7, 1; CPU will demand 1
And 7 will be demanded in the furthest future.

so 7 will be swaped out and 3 will be swap in.

				Hit		Hit		Hit	Hit	Hit	Hit	Hit		Hit	Hit	Hit		Hit	Hit
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	7	7	7
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

⇓ when 4 is demanded

Among 3, 0, 1, 2 ; 1 is in the furthest from 4. so 1 will be swap out.

⇓

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, ...

4 → future

1 → furthest

⇓

Among 3, 0, 4, 2 ; 3 and 4 is not in demand that means 3 and 4 is the furthest. so 3 or 4 anyone can be replaced

Least Recently Used (LRU):

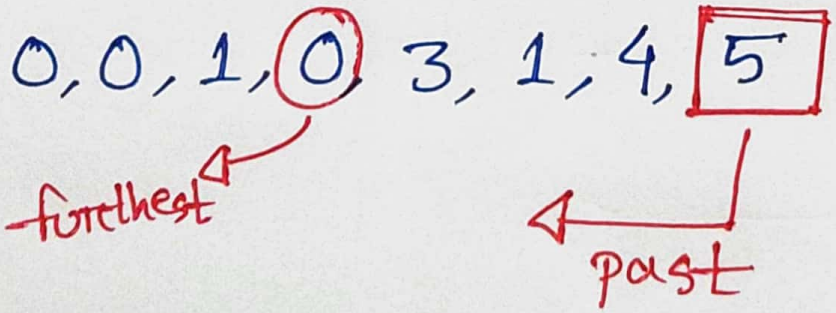
- remove the page from frames that has not been accessed for the longest period of time
- for a demanded page, we will find which page from the frames is furthest in past direction. Then we will replace that page.

- for example:

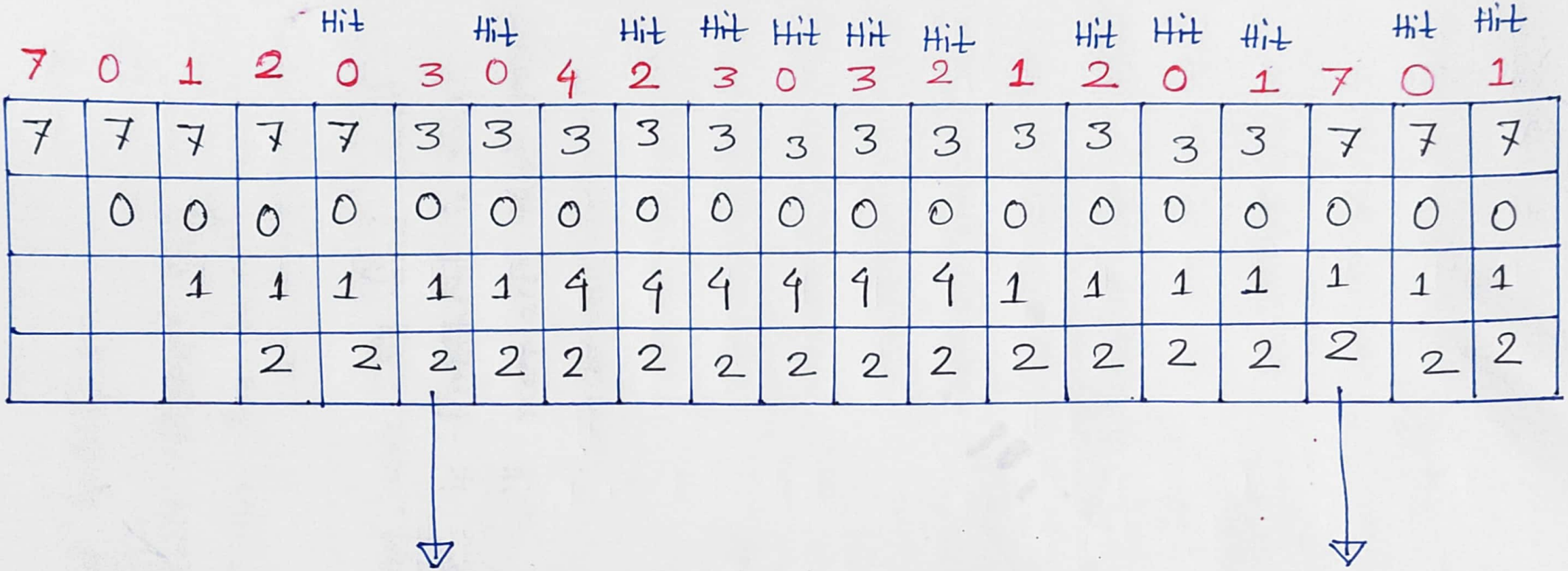
Ref: 0, 0, 1, 0, 3, 1, 4, 5

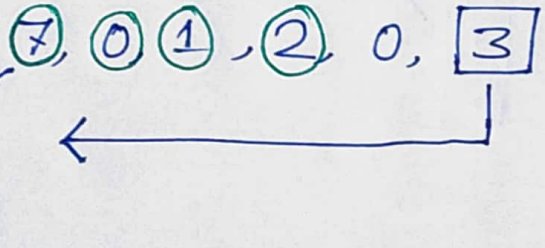
M.M:

then Page 5 is demanded.



so 0 will be replaced.





Least recently used page when page 3 is demanded. That means 7 is the forthest in past direction according to page 3. so 7 will be replaced.

LRU