PAGE REPLACEMENTNT

Page Replacement

Page Replacement is the technique of swapping out pages from physical memory when there are no more free frames available, in order to make room for other pages which has to be loaded to the physical memory.

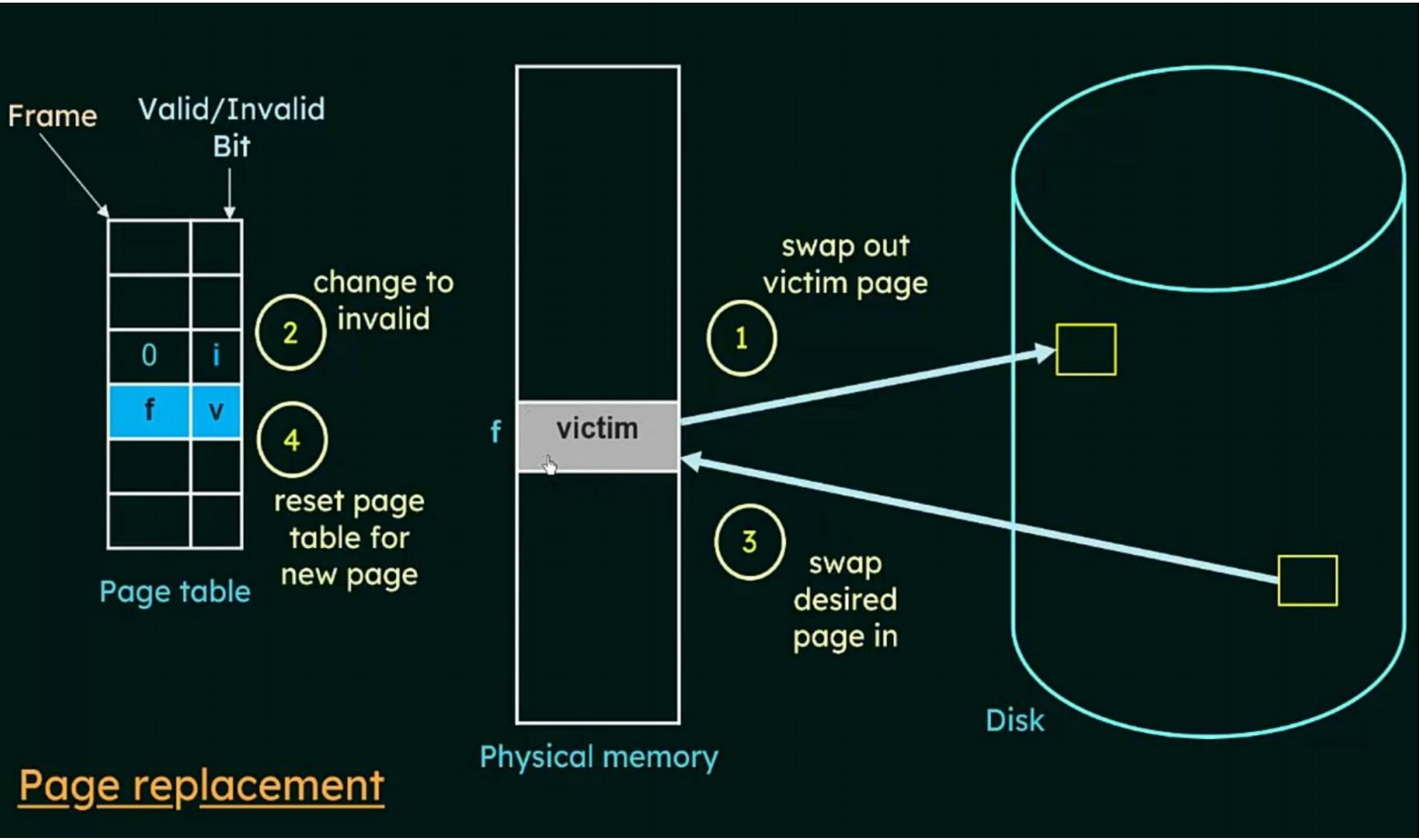
If a process wants to use/access a page which is not present in physical memory, it will cause a page fault.
So, that page has to be now loaded into memory.

But if there are no free frames available to load that page, what should we do then?

- We look for an occupied frame that is not being used currently.
- We free that frame by writing its contents to the swap space.
- We update the page tables (and other tables) to indicate that the page is now no more in memory.
- We load the page of the process that caused the page fault to occur into the freed frame.

Steps:

1. Find the location of the page to be loaded on the disk.
2. If there is a free frame, use that frame and load the page into that frame.
3. If there are no free frames, use a page-replacement algorithm to select a victim frame.
4. Write the victim frame to the disk and change the page and frame tables accordingly.
5. Read the desired page into the newly freed frame and change the page and frame tables.
6. Restart the user process.


Frame
Valid/Invalid Bit
0
i
f
v
Page table
2
change to invalid
4
reset page table for new page
f
victim
Physical memory
1
swap out victim page
3
swap desired page in
Disk
Page replacement

Page-fault service time when no frames are free

1. If no frames are free, two page transfers (one out and one in) are required.
2. This doubles the page-fault service time and increases the effective access time accordingly.

How can we reduce this?

We can make use of **the Modify (Dirty) Bit.**

The modify bit is set when a page has been modified.

So, when page replacement has to be done on a page which is present in memory:

- If modify bit is set – That means the page has been modified and we need to write that to the disk.
- If the modify bit is not set – That means the page has not been modified and it is the same as the copy which is already present in the disk. In this case, we don't have to overwrite this page on the disk, thus reducing the overhead and the effective access time accordingly.

FIFO PAGE REPLACEMENT

FIFO Page Replacement

The first-in, first-out (FIFO) algorithm is the simplest page replacement algorithm.

When page replacement has to be done, the oldest page in memory is chosen to be swapped out.

How to maintain a record of the time that pages were brought into memory in order to figure out which is the oldest page?

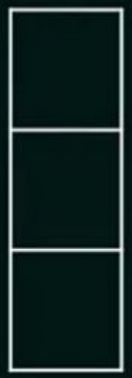
- We don't have to strictly record the times.
- Instead we can use a FIFO queue that can hold all the pages that are in memory.
- When replacement has to be done, we choose the page at the head of the queue to be swapped out of memory.
- When a new page is loaded into memory, we add it at the tail of the queue.

Consider the reference string

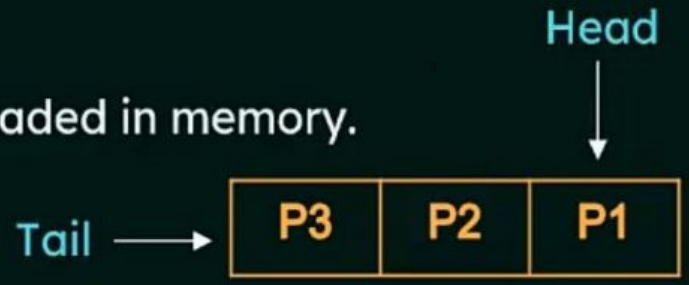
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

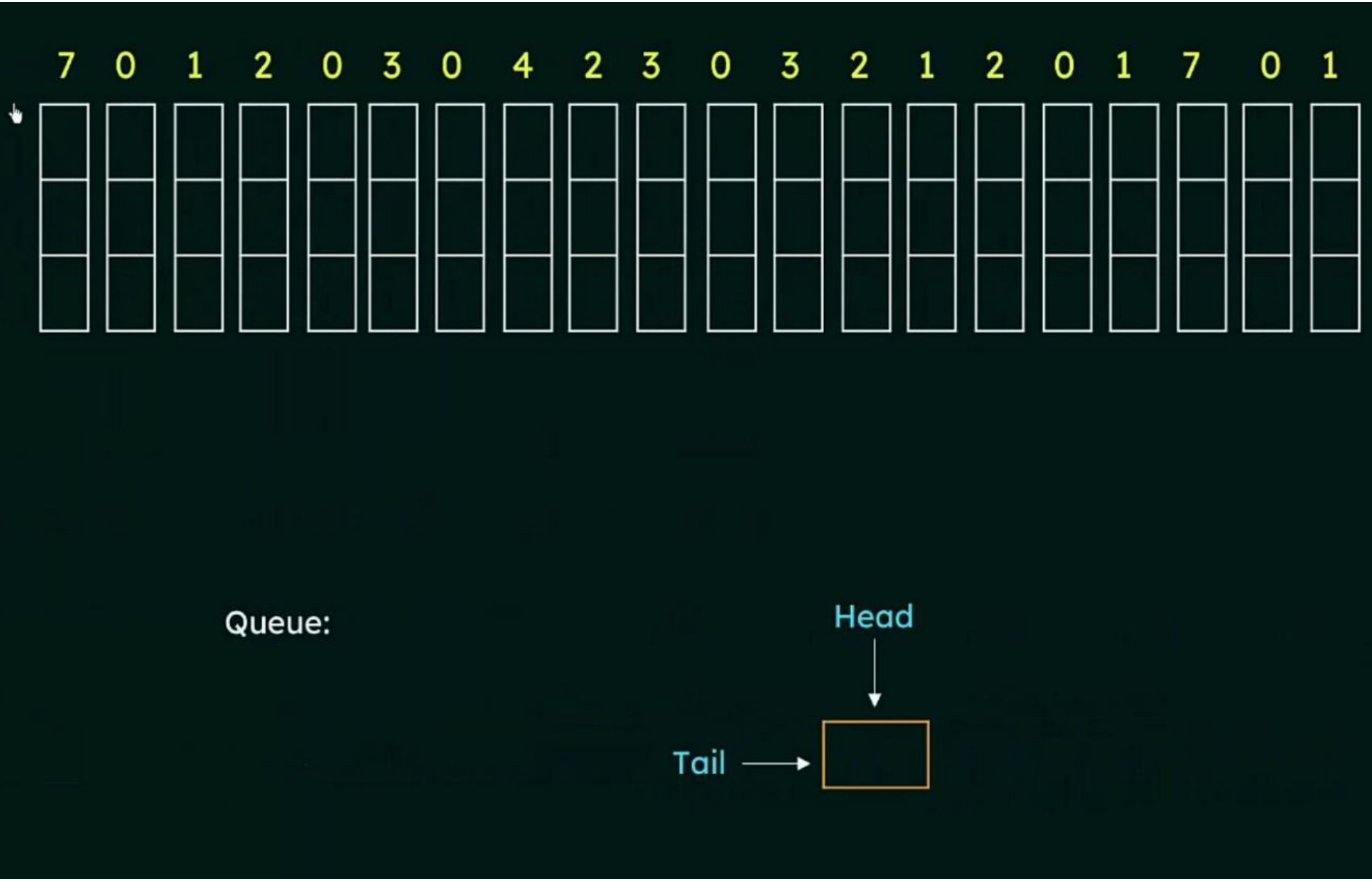
where each value represents a page.

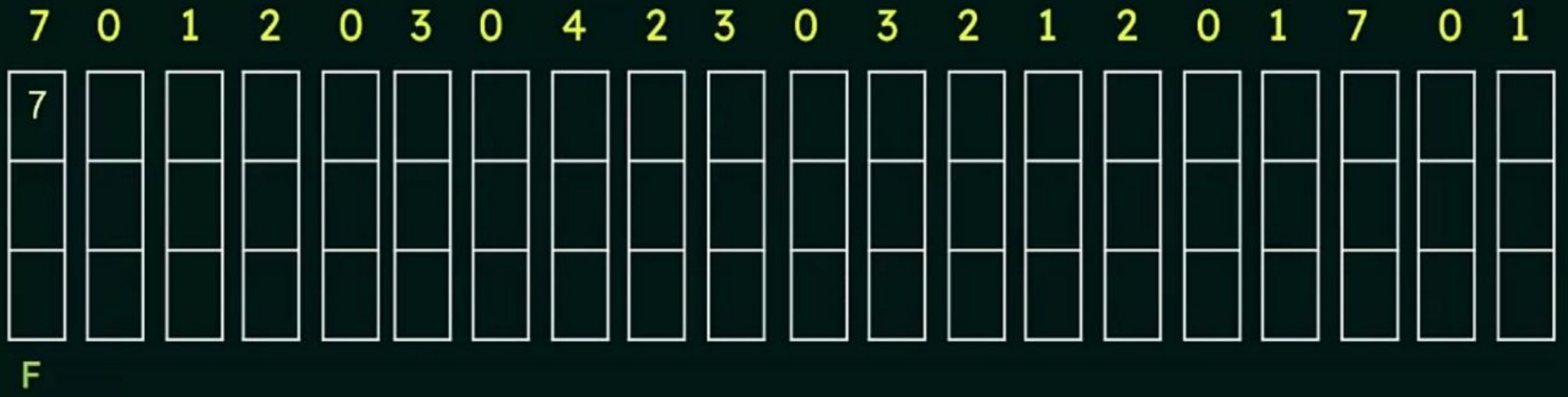
Let's say we have 3 frames in memory.



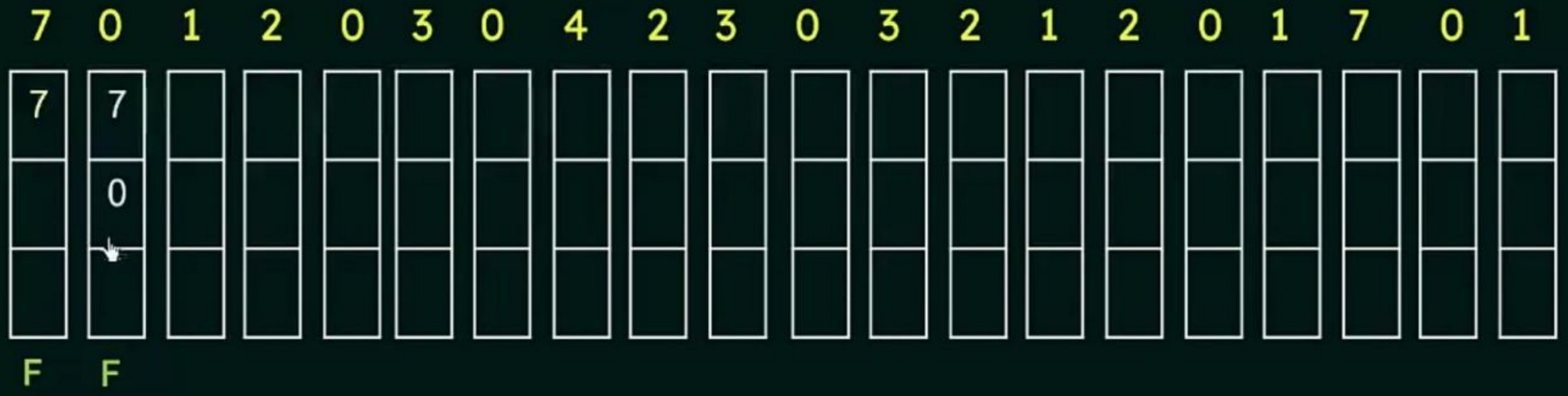
Also, let's consider a Queue that can hold all the pages that are loaded in memory.



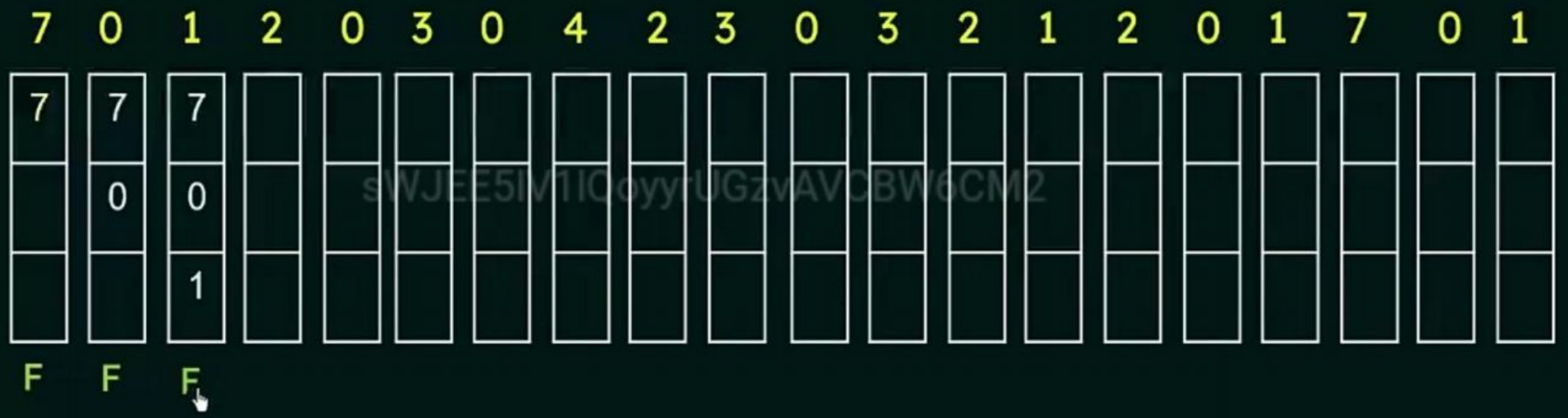

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Queue:
Head
Tail



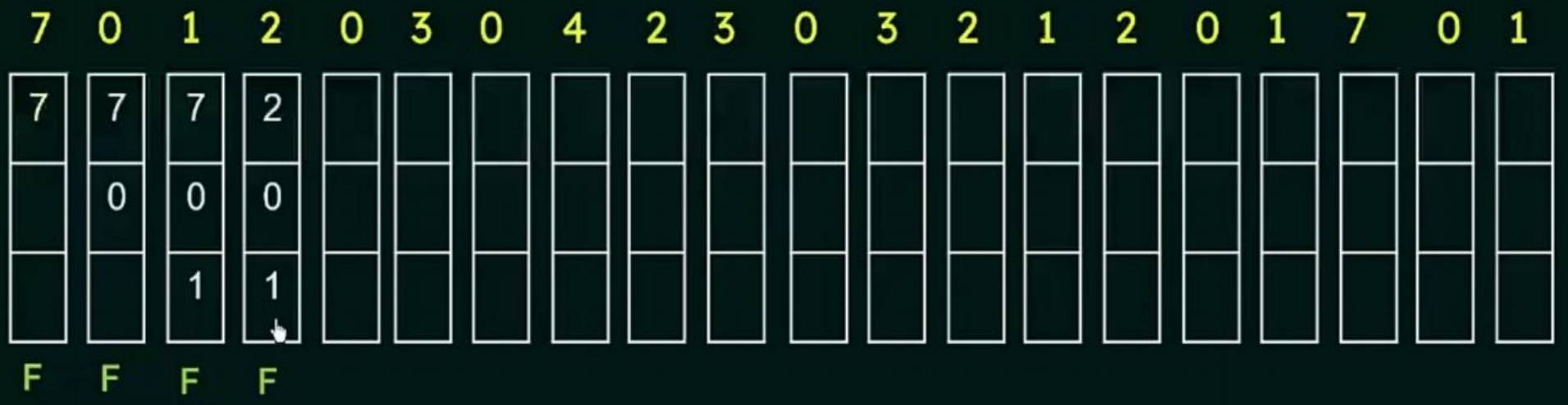
Queue:



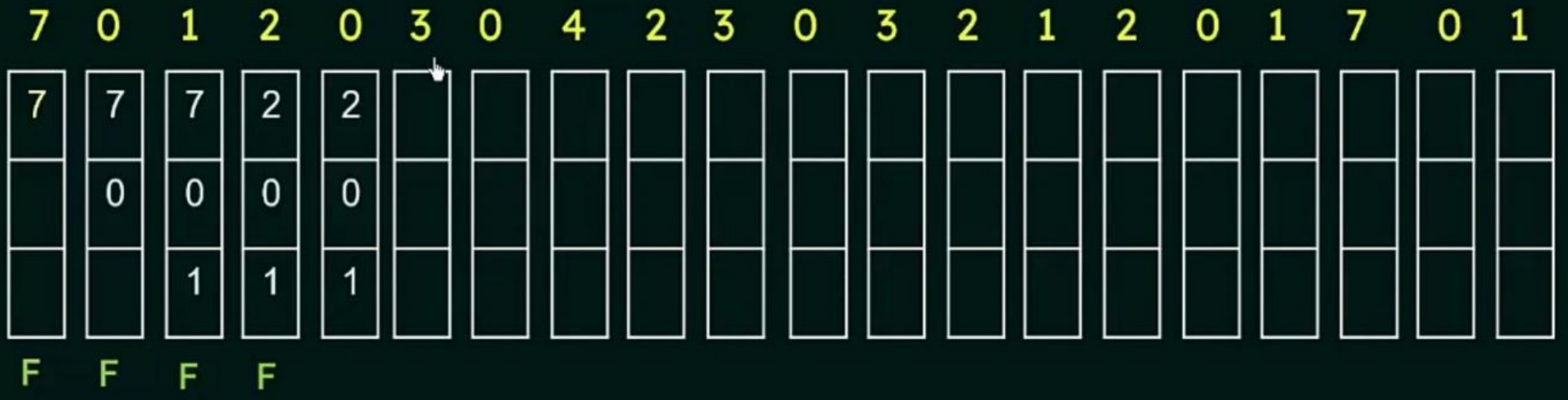
Queue:



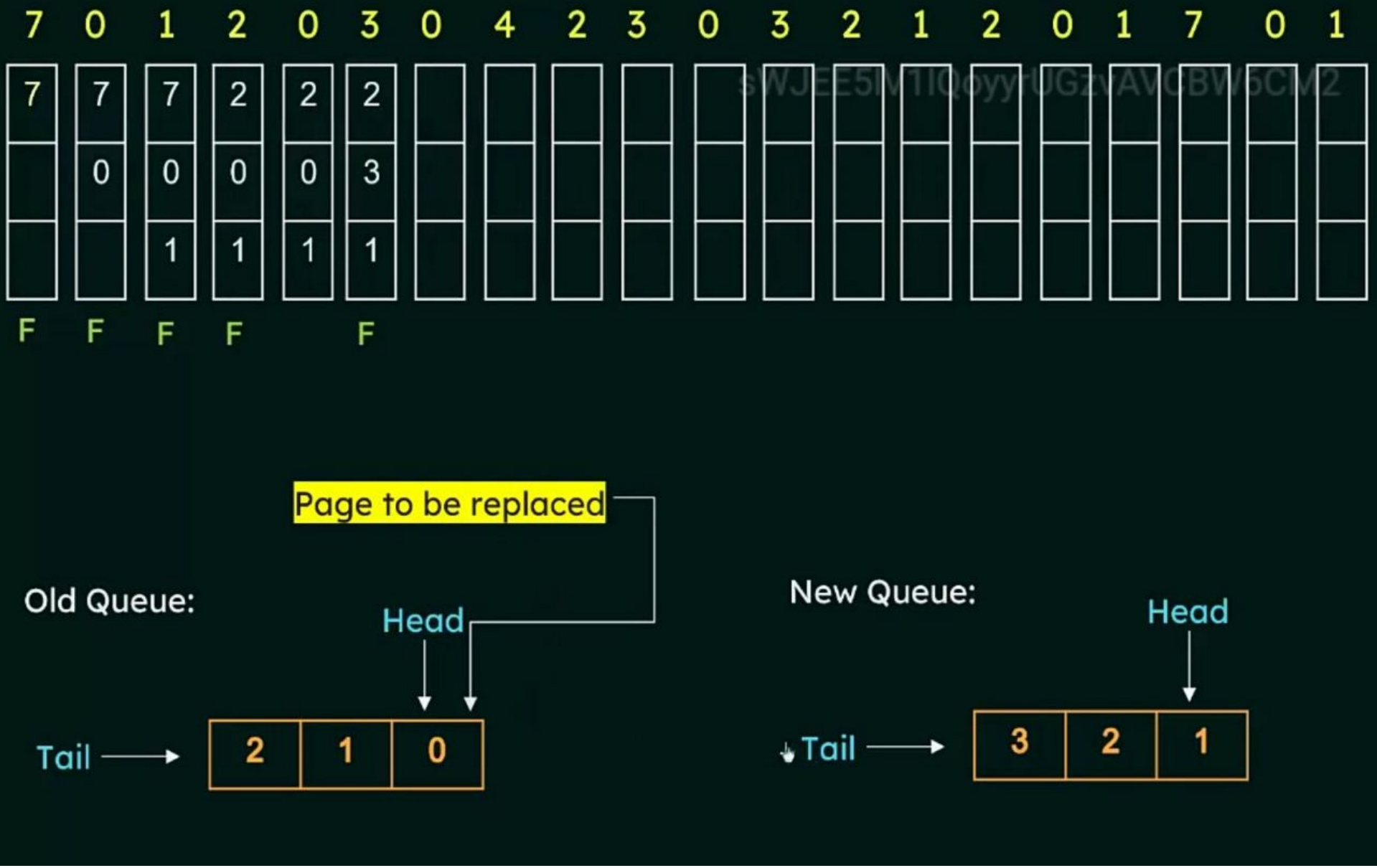
Queue:

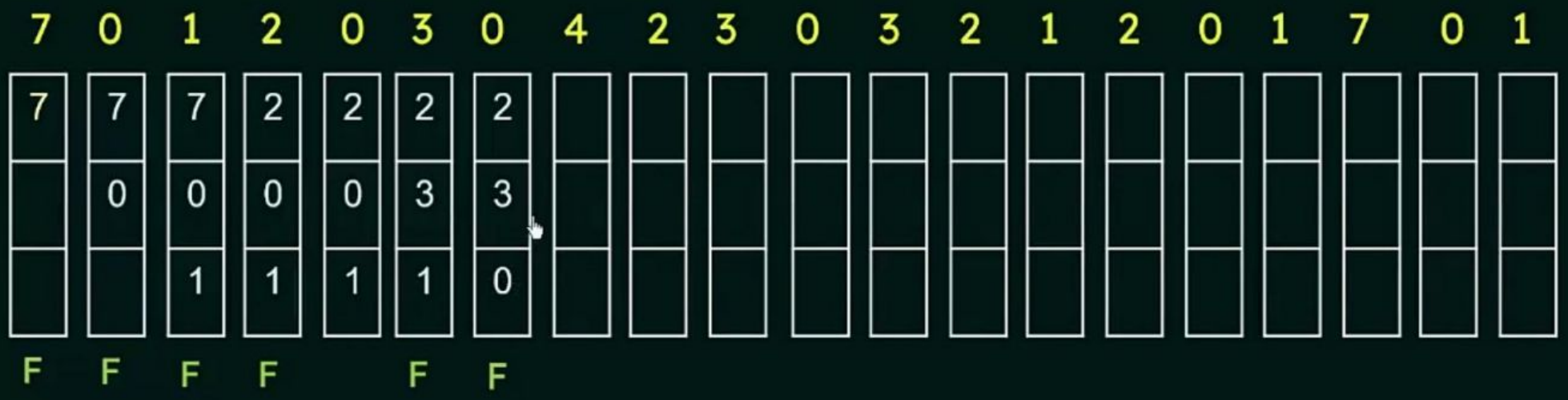

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2
0 0 0
1 1
F F F F

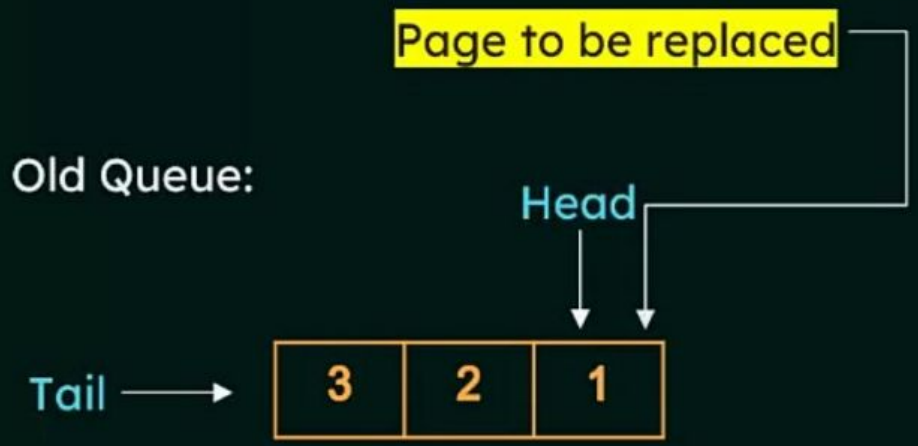

Page to be replaced
Old Queue:
Head
Tail
1 0 7
New Queue:
Head
Tail
2 1 0

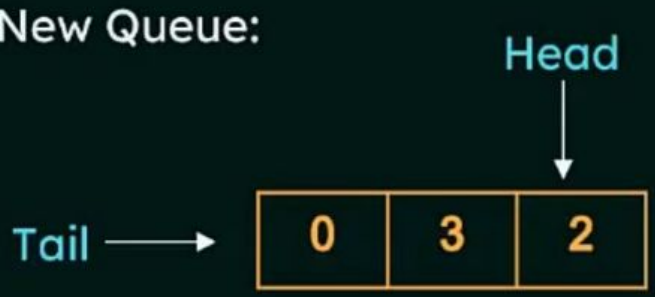


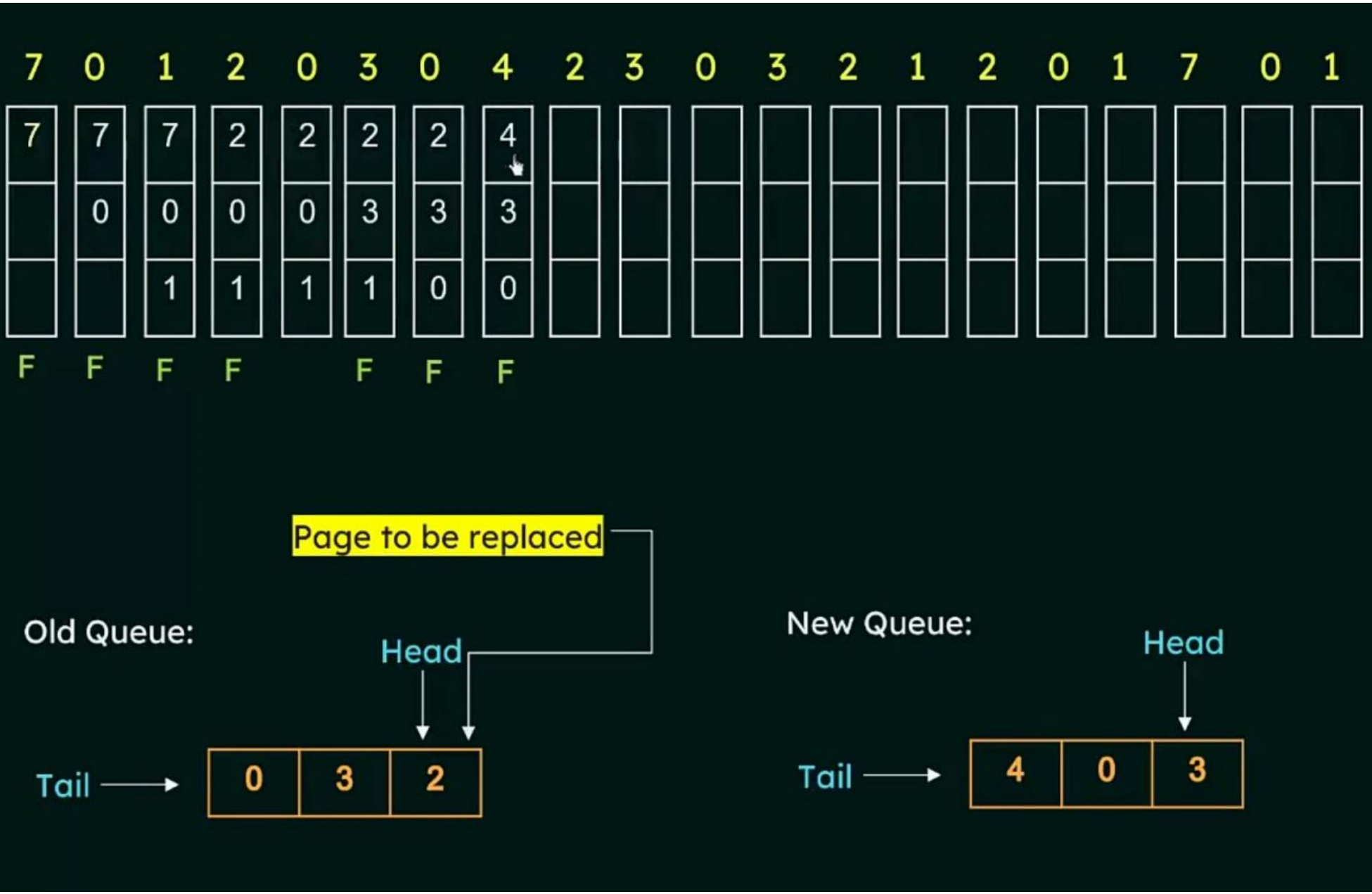
Queue:

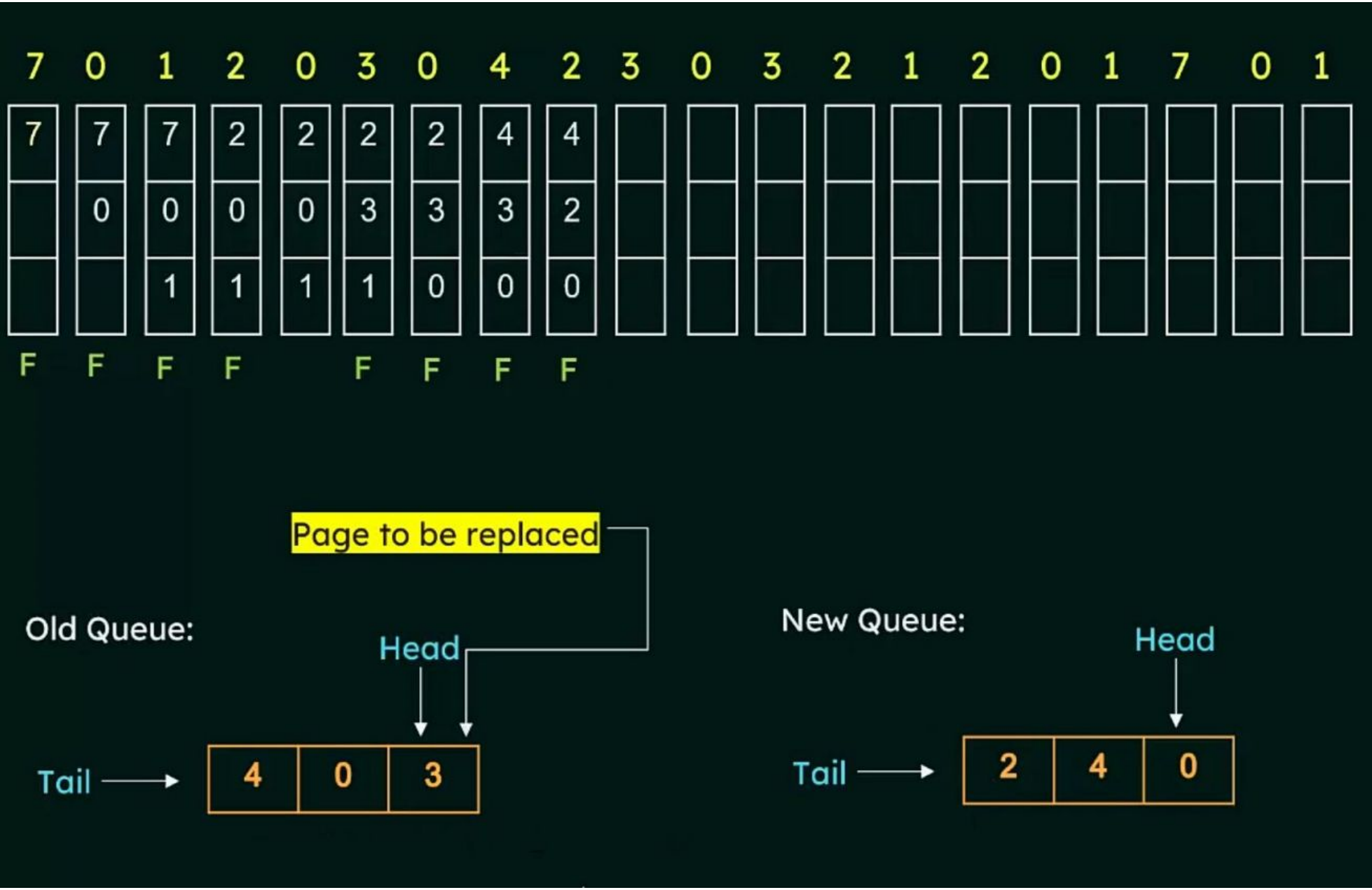

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2
0 0 0 0 3
1 1 1 1
F F F F F
Page to be replaced
Old Queue:
Head
Tail
2 1 0
New Queue:
Head
Tail
3 2 1

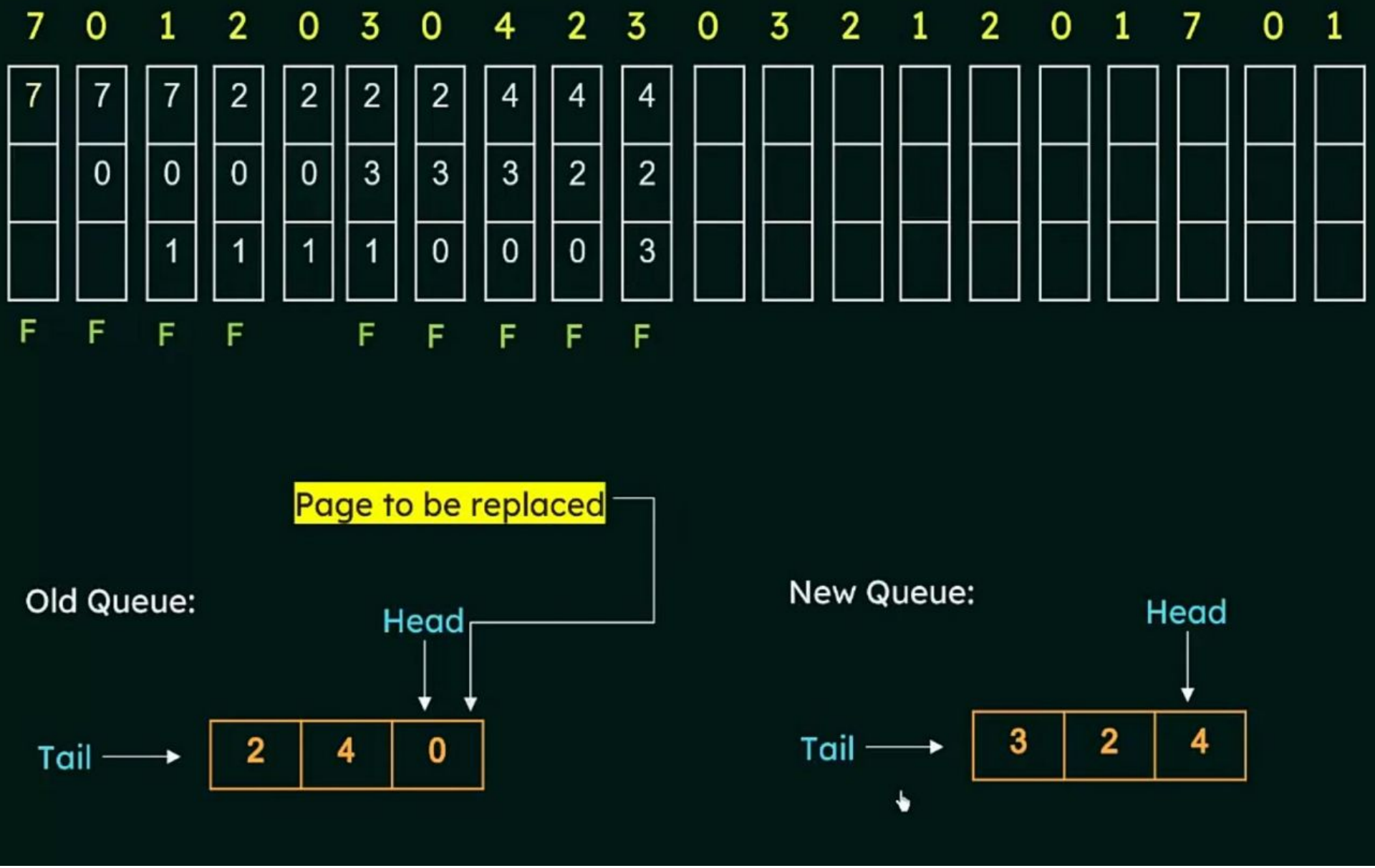

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2
0 0 0 0 3 3
1 1 1 1 0
F F F F F F

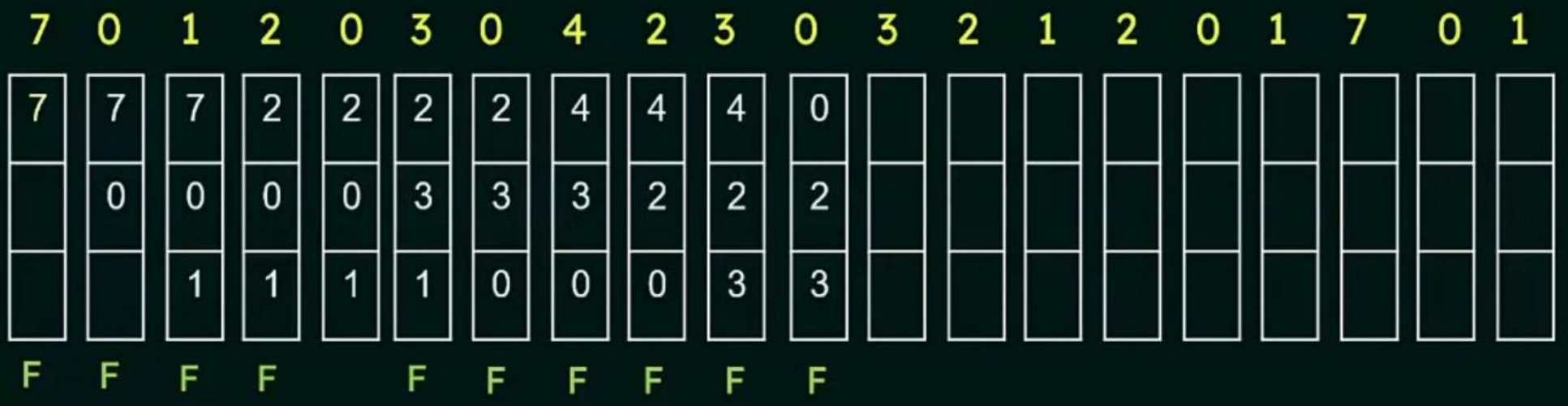

Page to be replaced
Old Queue:
Head
Tail
3 2 1

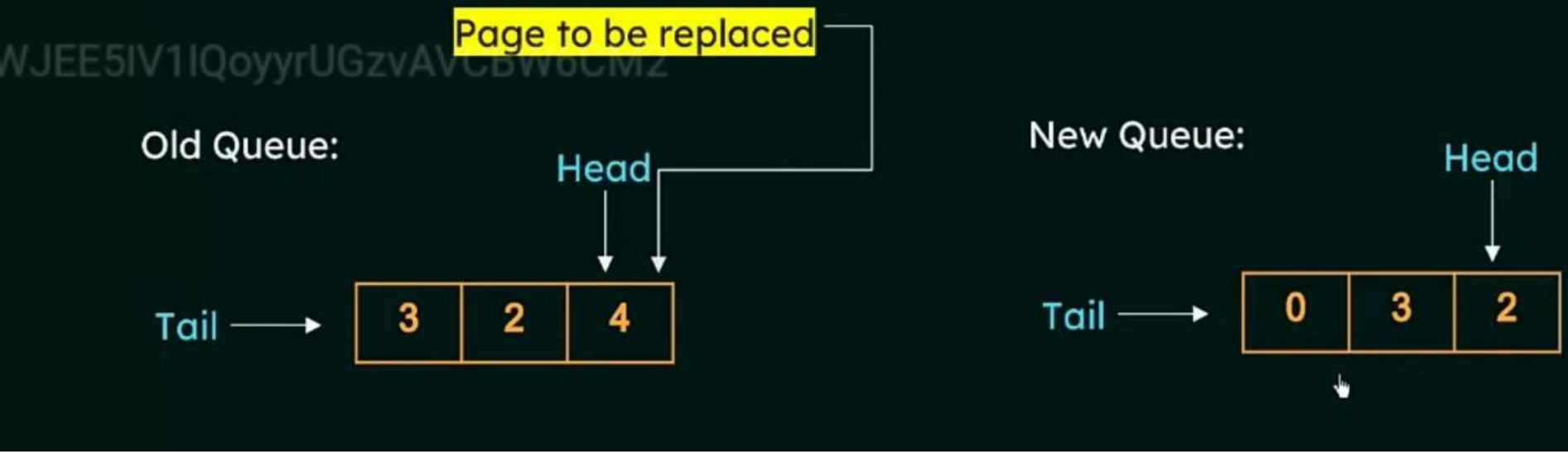

New Queue:
Head
Tail
0 3 2

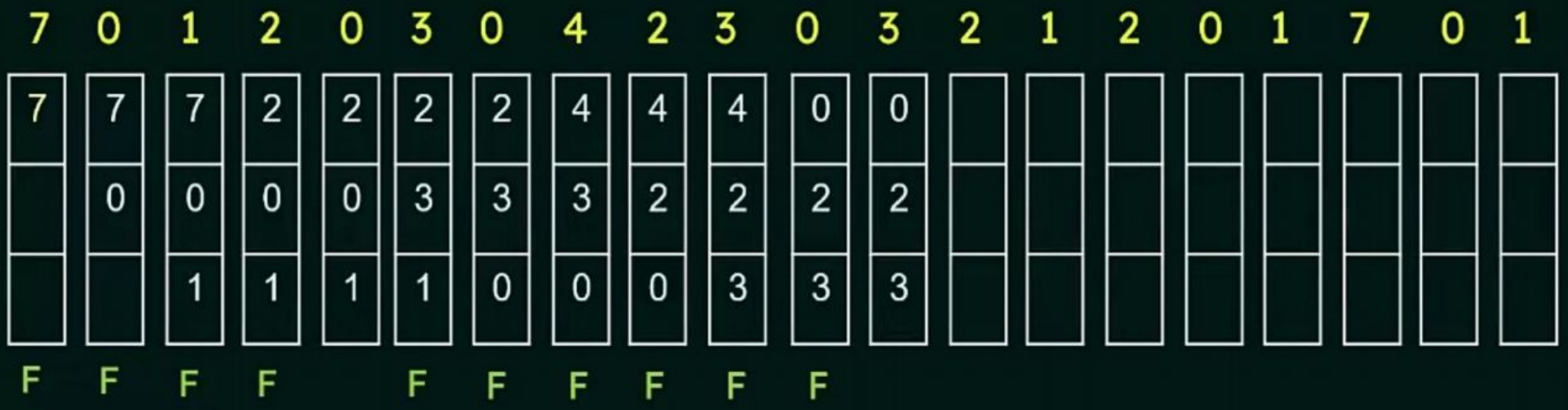

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4
0 0 0 0 3 3 3
1 1 1 1 0 0
F F F F F F F
Page to be replaced
Old Queue:
Head
Tail
0 3 2
New Queue:
Head
Tail
4 0 3


7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4
0 0 0 0 3 3 3 2
1 1 1 1 0 0 0
F F F F F F F F
Page to be replaced
Old Queue:
Head
Tail
4 0 3
New Queue:
Head
Tail
2 4 0


7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4
0 0 0 0 3 3 3 2 2
1 1 1 1 0 0 0 3
F F F F F F F F F
Page to be replaced
Old Queue:
Head
Tail
2 4 0
New Queue:
Head
Tail
3 2 4

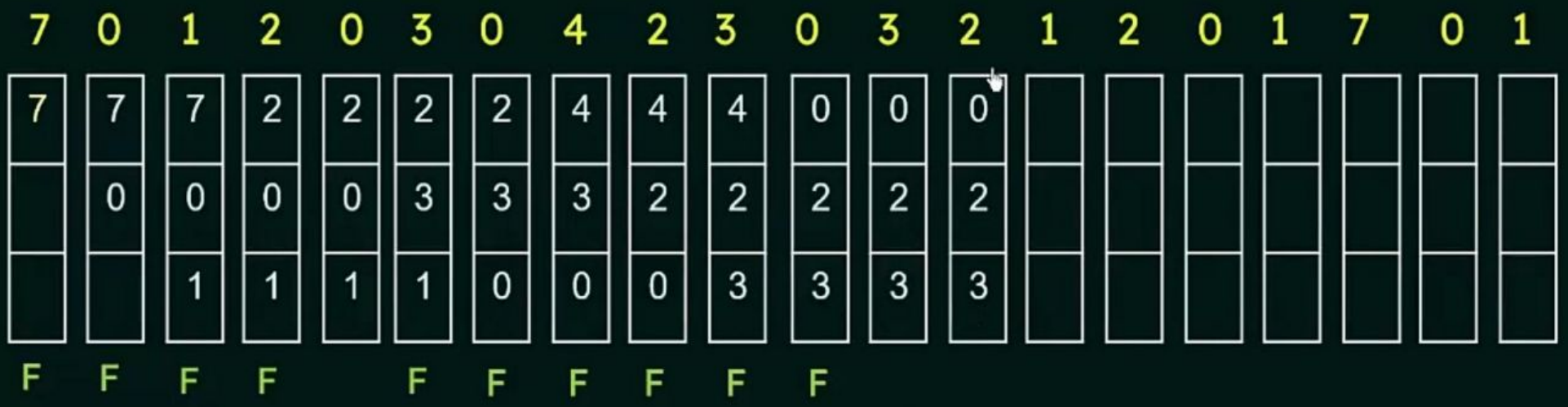

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0
0 0 0 0 3 3 3 2 2 2
1 1 1 1 0 0 0 3 3
F F F F F F F F F


Page to be replaced
Old Queue:
Head
Tail
3 2 4
New Queue:
Head
Tail
0 3 2


7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0
0 0 0 0 3 3 3 2 2 2 2
1 1 1 1 0 0 0 3 3 3
F F F F F F F F F F

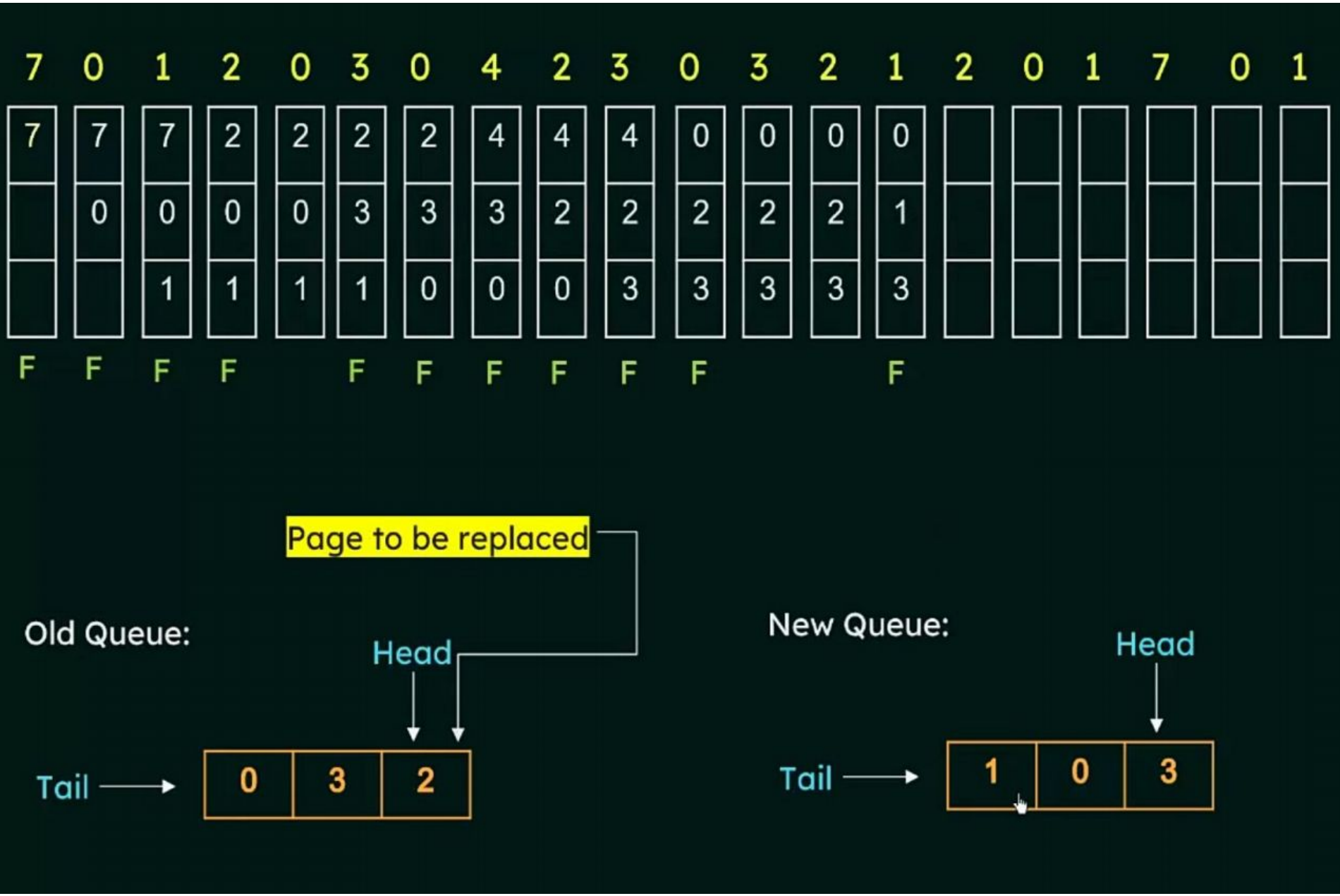
Queue:

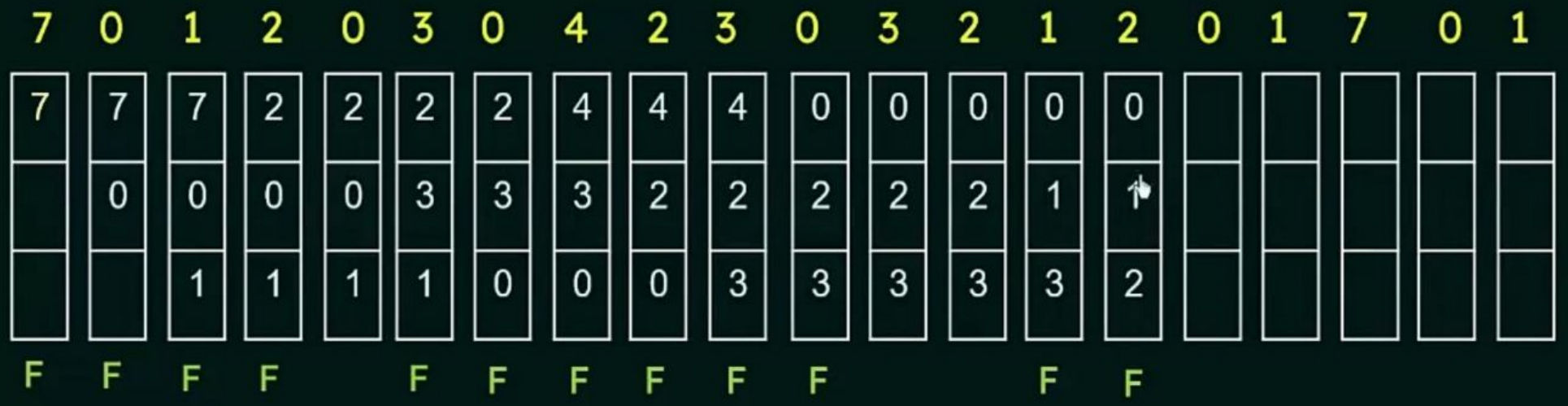

Head
Tail
0
3
2

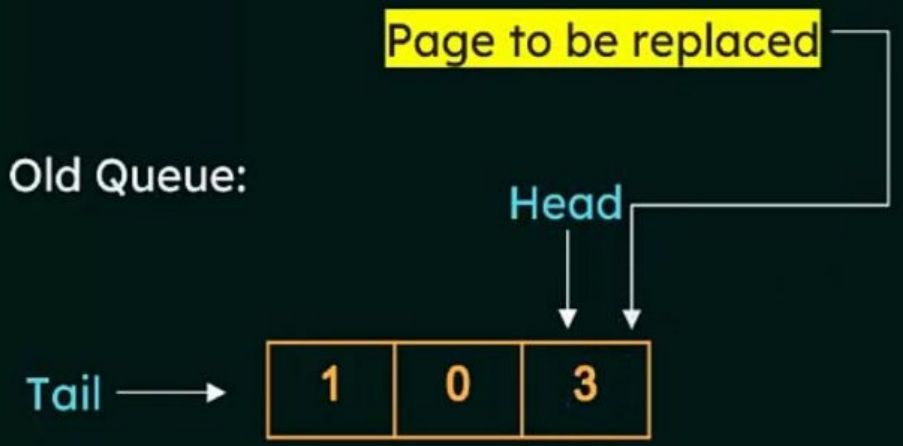

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0 0
0 0 0 0 3 3 3 2 2 2 2 2
1 1 1 1 0 0 0 3 3 3 3
F F F F F F F F F

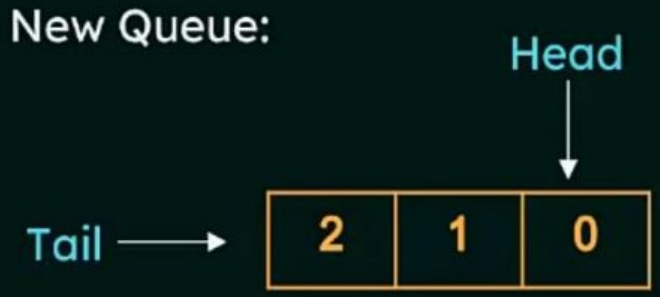
Queue:

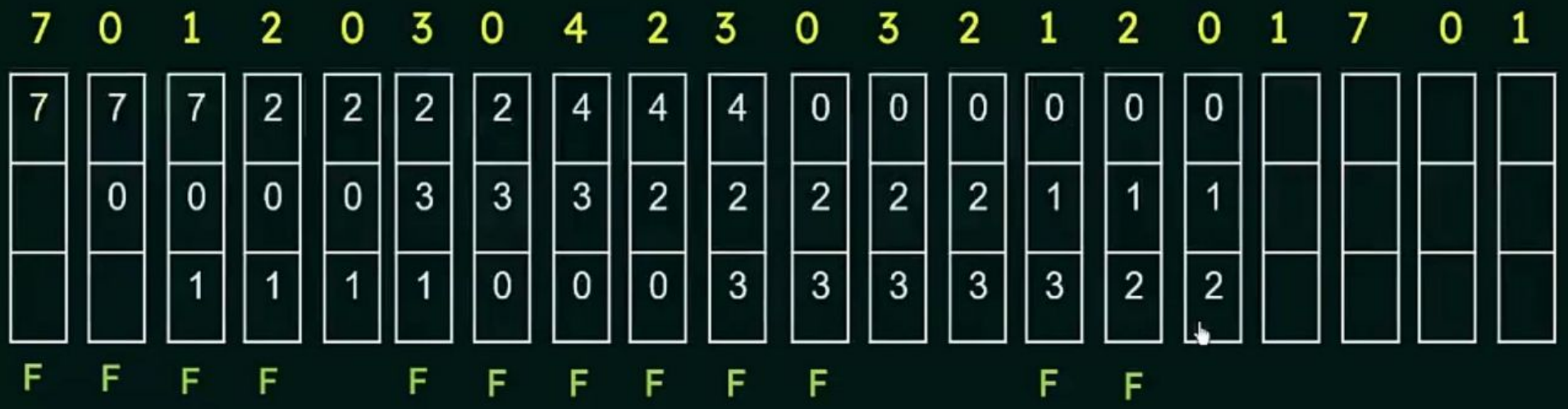

Head
Tail
0
3
2


7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0 0 0
0 0 0 0 3 3 3 2 2 2 2 2 1
1 1 1 1 0 0 0 3 3 3 3 3
F F F F F F F F F F F
Page to be replaced
Old Queue:
Head
Tail
0 3 2
New Queue:
Head
Tail
1 0 3

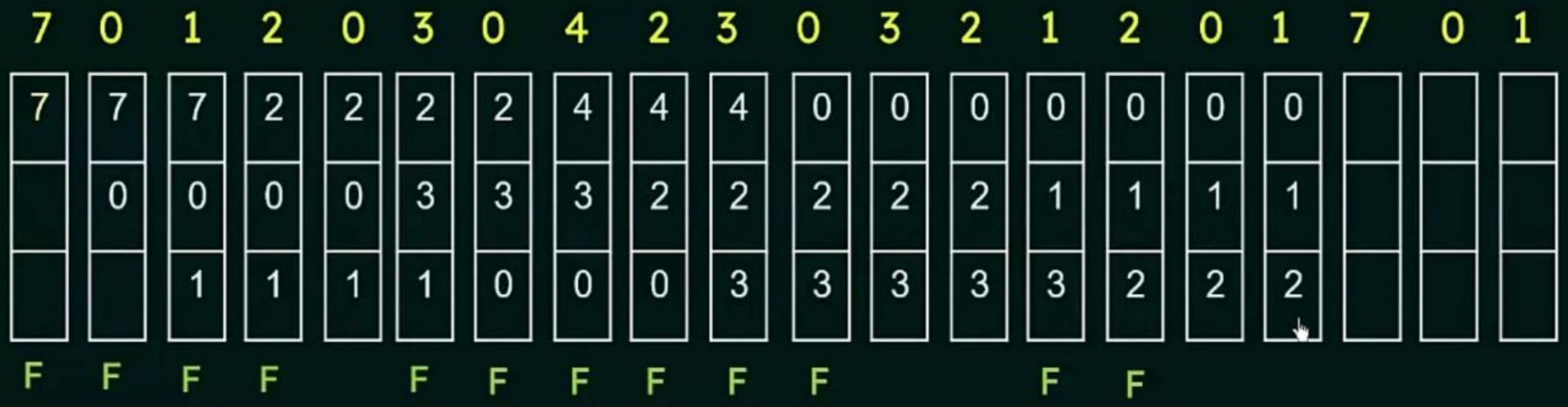

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0 0 0 0
0 0 0 0 3 3 3 2 2 2 2 2 1 1
1 1 1 1 0 0 0 3 3 3 3 3 2
F F F F F F F F F F F F


Page to be replaced
Old Queue:
Head
Tail
1 0 3

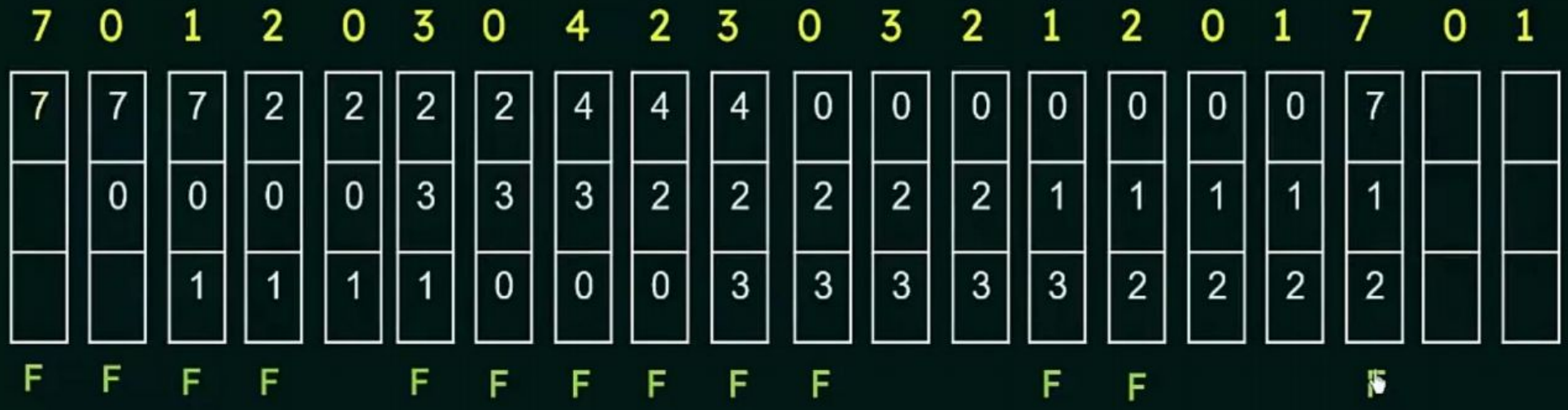

New Queue:
Head
Tail
2 1 0

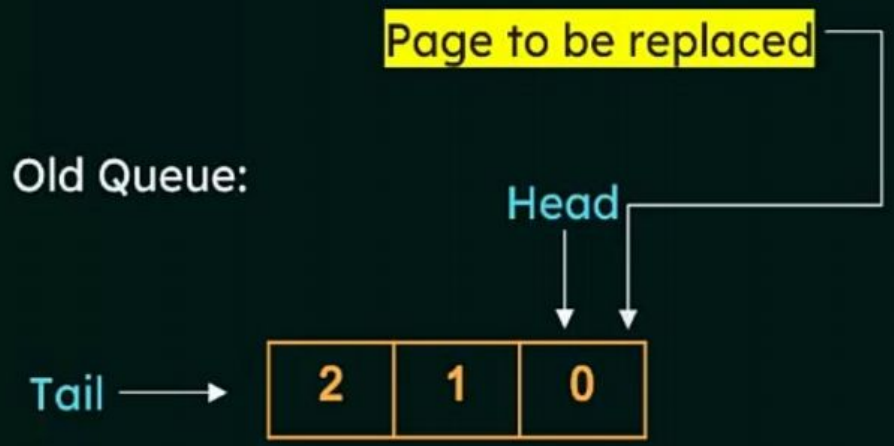

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0 0 0 0 0
0 0 0 0 3 3 3 2 2 2 2 2 1 1 1
1 1 1 1 0 0 0 3 3 3 3 3 2 2
F F F F F F F F F F F F

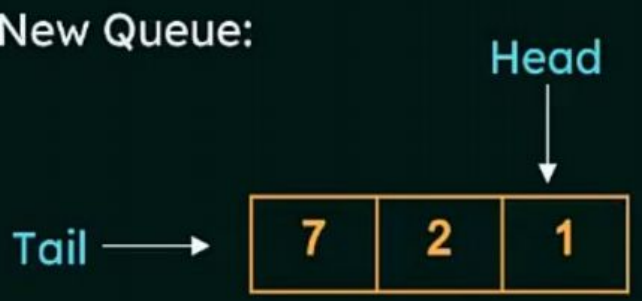

Queue:
Head
Tail
2
1
0

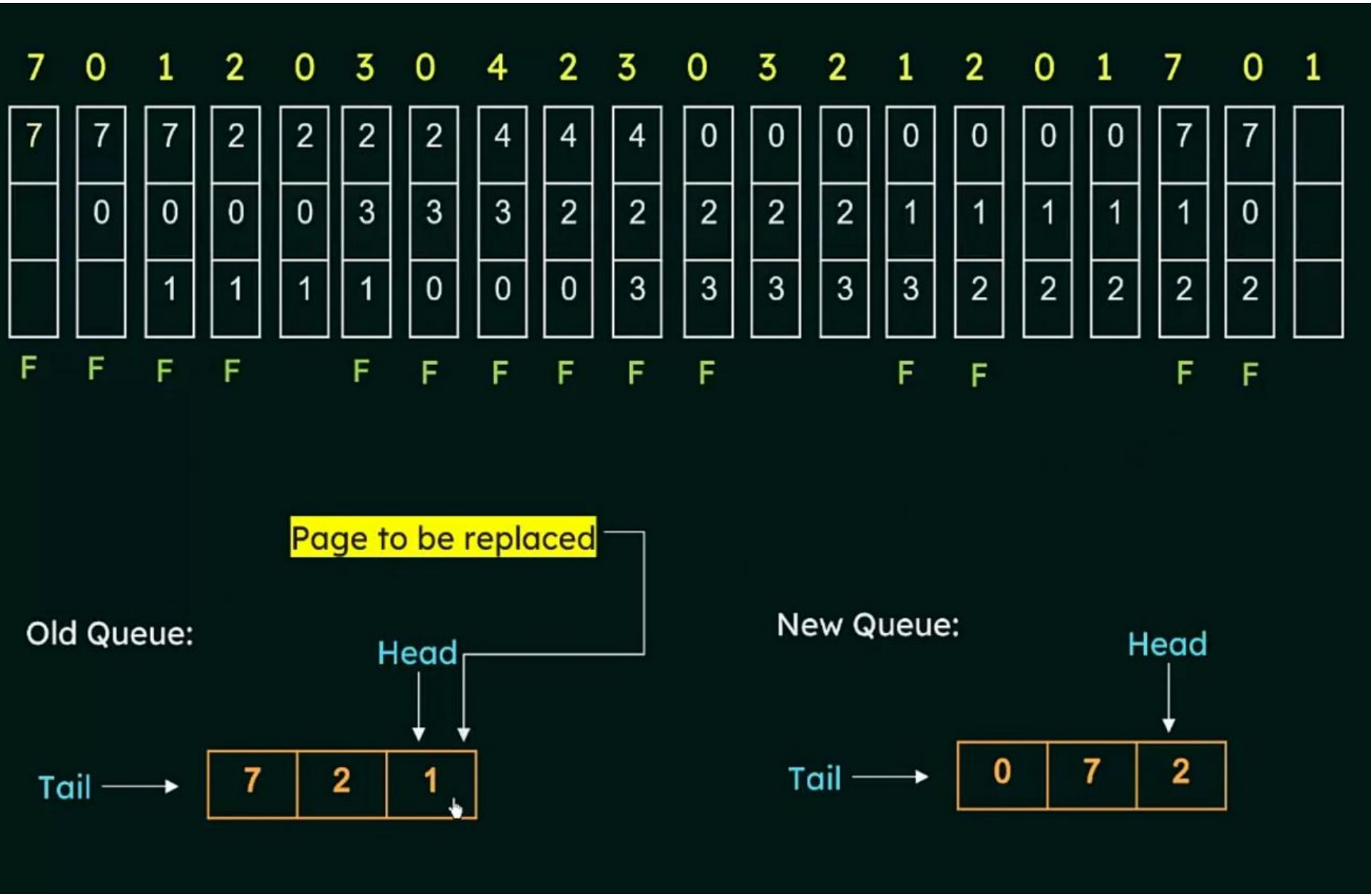

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0 0 0 0 0 0
0 0 0 0 3 3 3 2 2 2 2 2 1 1 1 1
1 1 1 0 0 0 3 3 3 3 3 2 2 2
F F F F F F F F F F F F

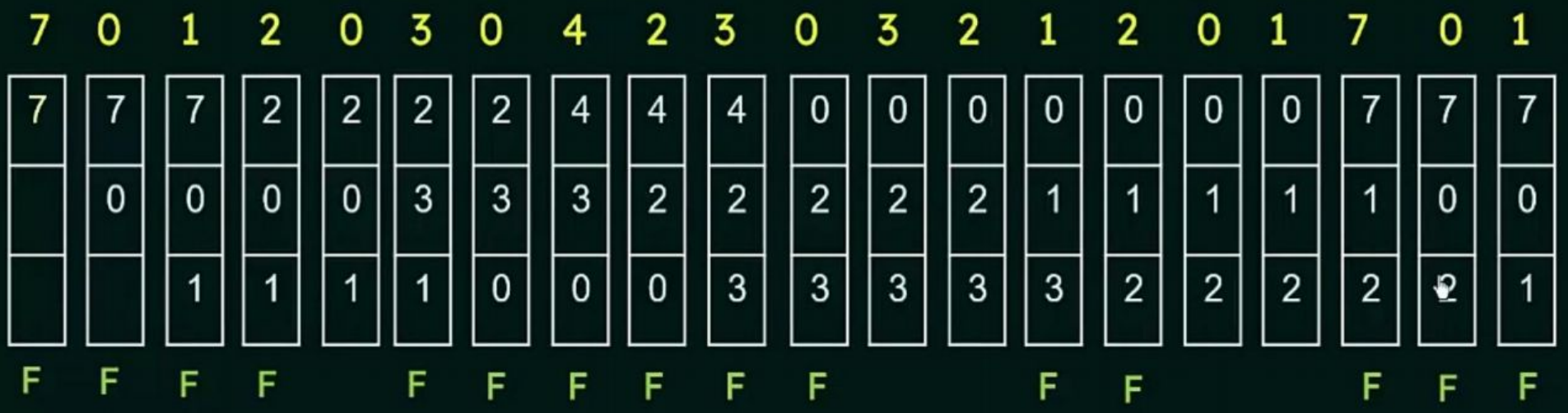

Queue:
Head
Tail
2
1
0


7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0 0 0 0 0 0 7
0 0 0 0 3 3 3 2 2 2 2 2 1 1 1 1 1
1 1 1 1 0 0 0 3 3 3 3 3 2 2 2 2
F F F F F F F F F F F F F

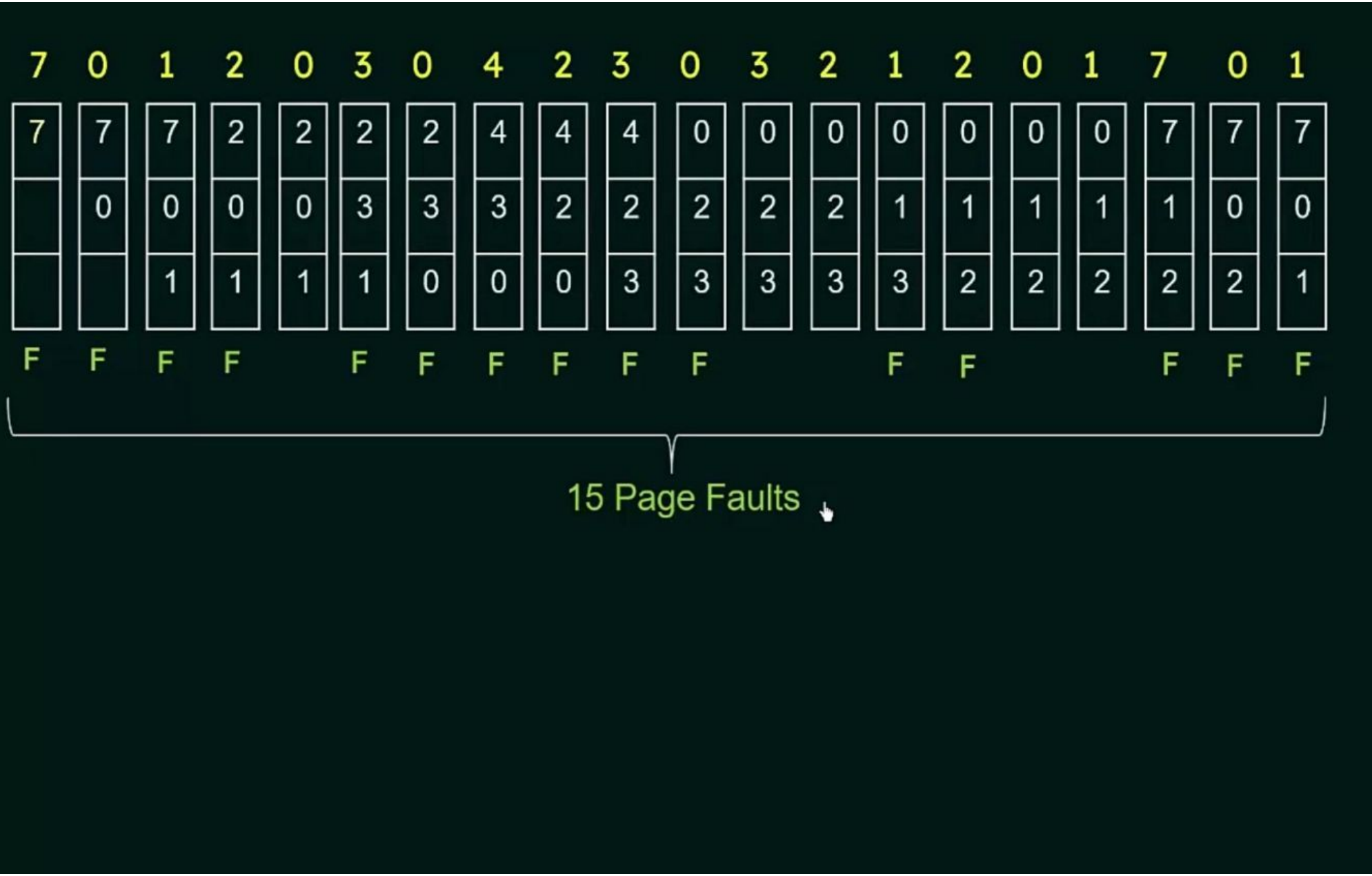

Page to be replaced
Old Queue:
Head
Tail
2 1 0


New Queue:
Head
Tail
7 2 1


7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0 0 0 0 0 0 7 7
0 0 0 0 3 3 3 2 2 2 2 2 1 1 1 1 1 0
1 1 1 1 0 0 0 3 3 3 3 3 2 2 2 2 2
F F F F F F F F F F F F F
Page to be replaced
Old Queue:
Head
Tail
7 2 1
New Queue:
Head
Tail
0 7 2


7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0 0 0 0 0 0 7 7 7
0 0 0 0 3 3 3 2 2 2 2 2 1 1 1 1 1 0 0
1 1 1 1 0 0 0 3 3 3 3 3 2 2 2 2 2 1
F F F F F F F F F F F F F


Page to be replaced
Old Queue:
Head
Tail
0 7 2
New Queue:
Head
Tail
1 0 7


7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0 0 0 0 0 0 7 7 7
0 0 0 0 3 3 3 2 2 2 2 2 1 1 1 1 1 0 0
1 1 1 1 0 0 0 3 3 3 3 3 2 2 2 2 2 1
F F F F F F F F F F F F F F F
15 Page Faults

Belady's Anomaly

General Expectation about the relation between number of frames in memory and number of page faults:

With more number of page frames there will be lesser page faults.

- As the number of frames increases, the number of page faults drops to some minimal level.
- Adding physical memory increases the number of frames.

Consider the reference string

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

where each value represents a page.

Let's say we have 3 frames in memory.

Let's use FIFO page replacement and find the number of page faults that would occur in this case.


1 2 3 4 1 2 5 1 2 3 4 5
1 1 1 4 4 4 5 5 5 5 5 5
2 2 2 1 1 1 1 1 3 3 3
3 3 3 2 2 2 2 2 4 4
F F F F F F F F F
9 Page Faults

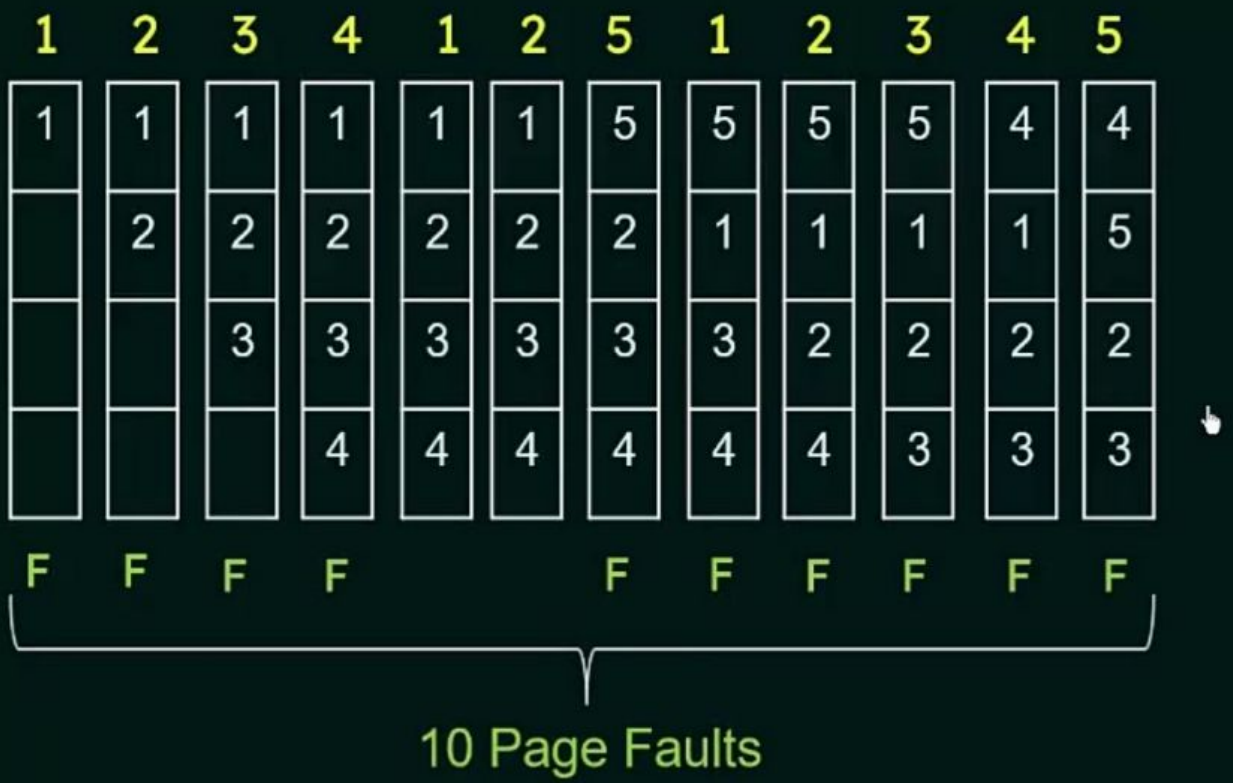
Consider the reference string

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

where each value represents a page.

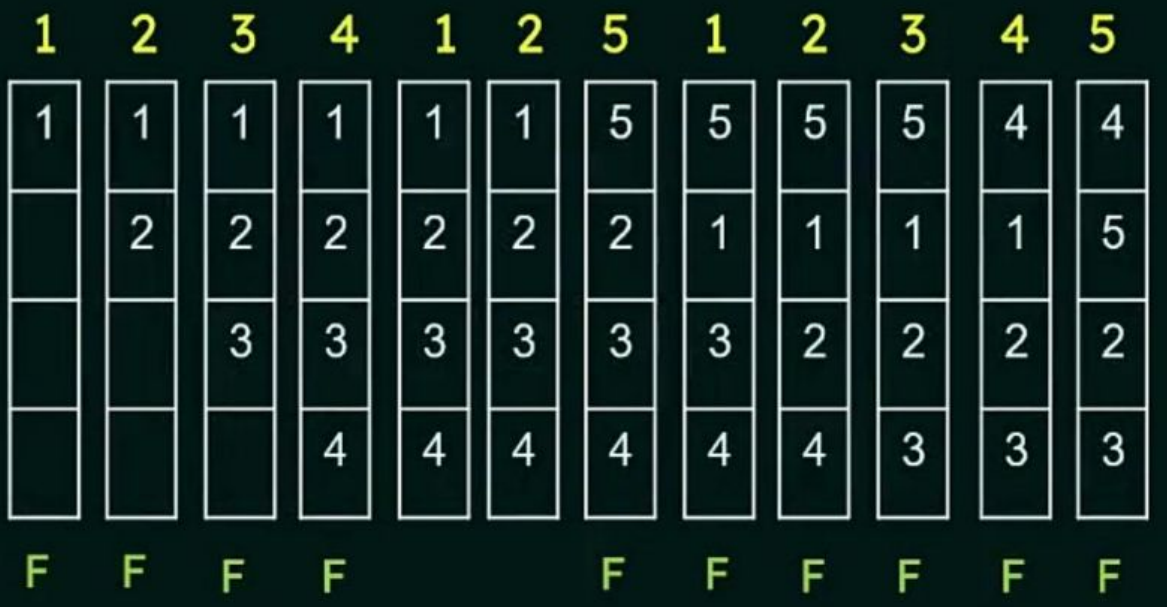
Let's say we have 4 frames in memory.

Let's use FIFO page replacement and find the number of page faults that would occur in this case.


1 2 3 4 1 2 5 1 2 3 4 5
1 1 1 1 1 1 5 5 5 5 4 4
2 2 2 2 2 1 1 1 1 5
3 3 3 3 3 2 2 2 2
4 4 4 4 4 4 3 3 3
F F F F F F F F F F
10 Page Faults

← 9 page faults when there were 3 frames

10 page faults when there were 4 frames →



We see that as the number of frames were increased, the number of page faults also increased.

This unexpected result is known as Belady's anomaly.

OPTIMAL PAGE REPLACEMENT

Optimal Page Replacement

An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms.

will never suffer from Belady's anomaly.

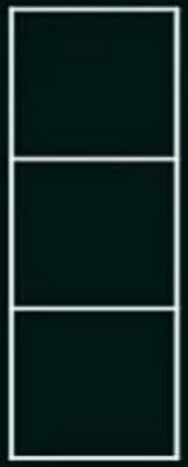
guarantees the lowest possible page fault rate for a fixed number of frames.

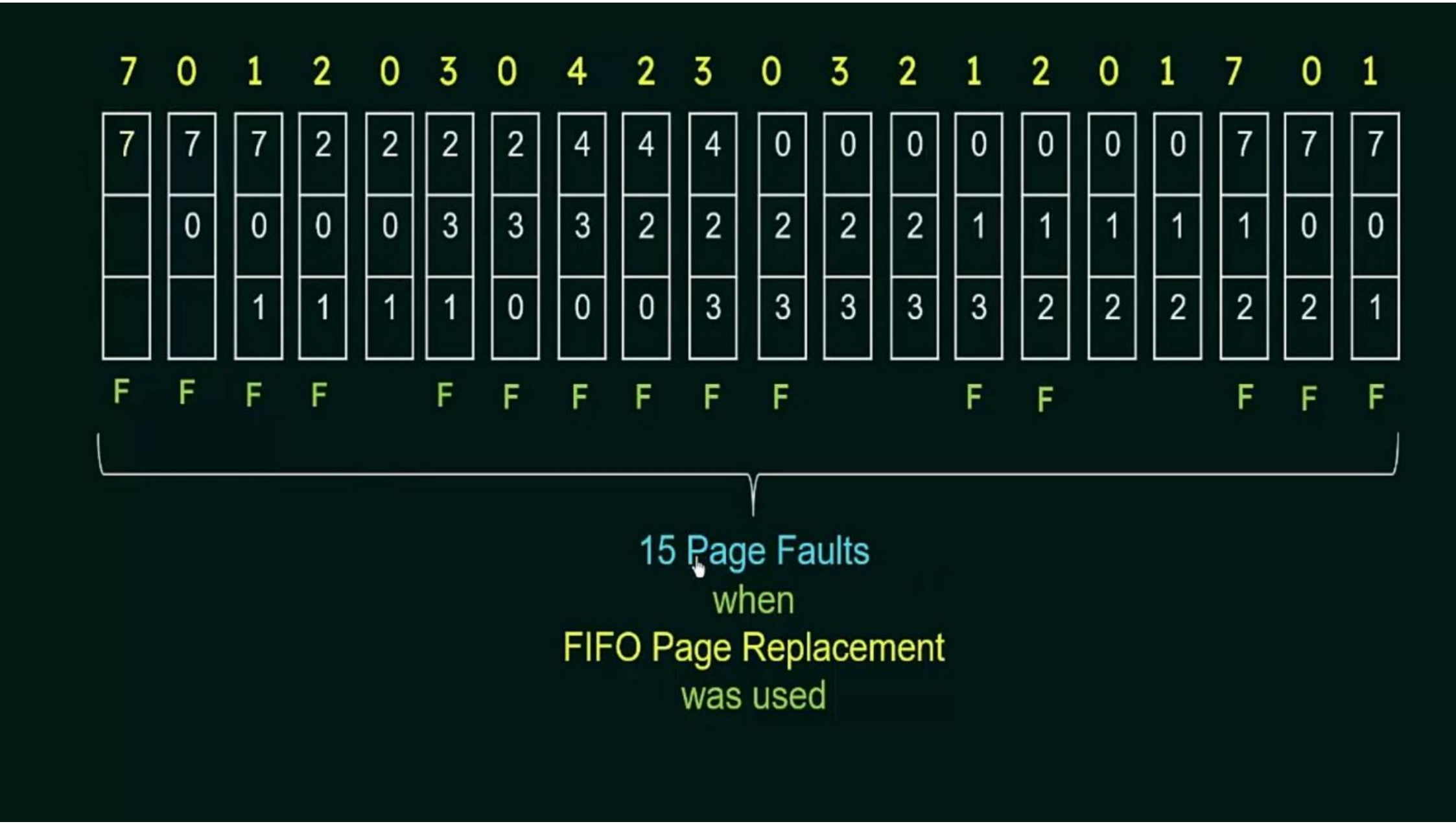
Consider the reference string

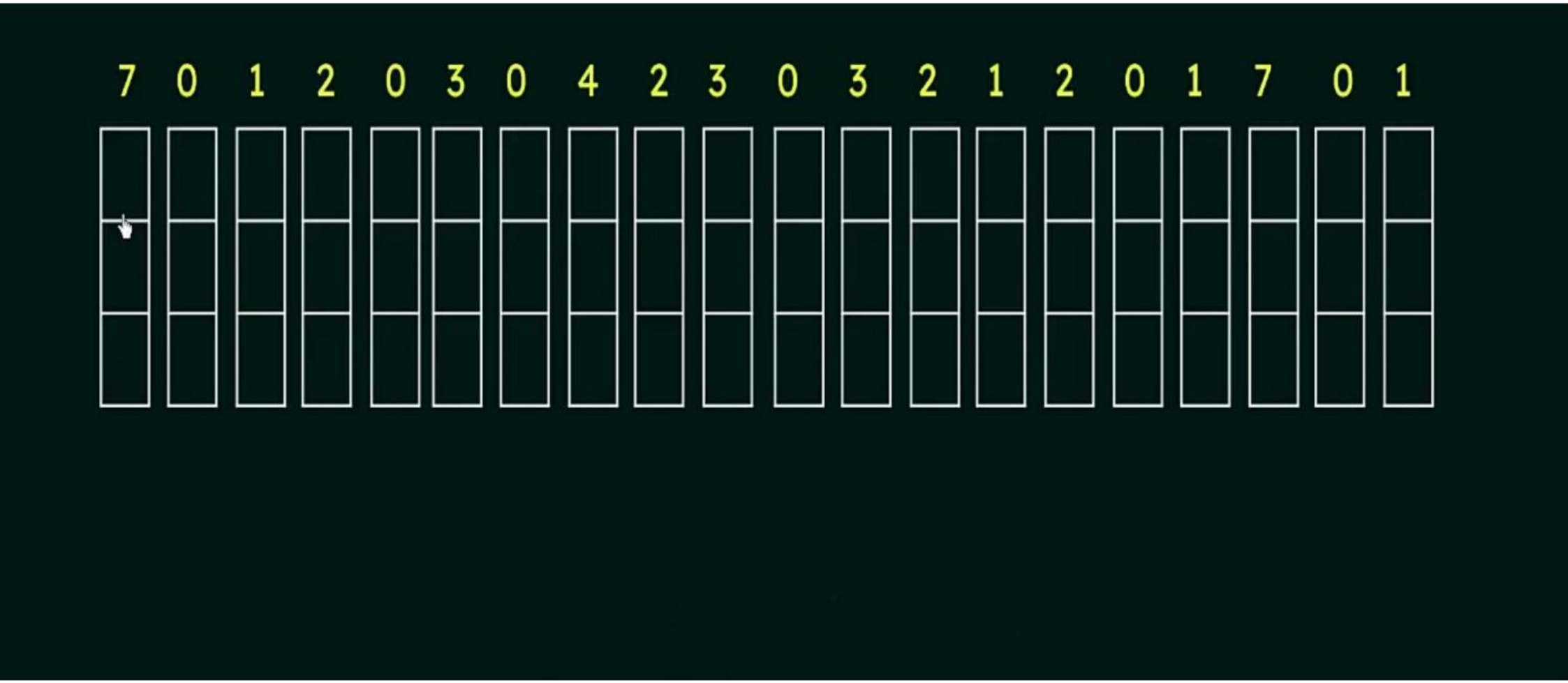
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

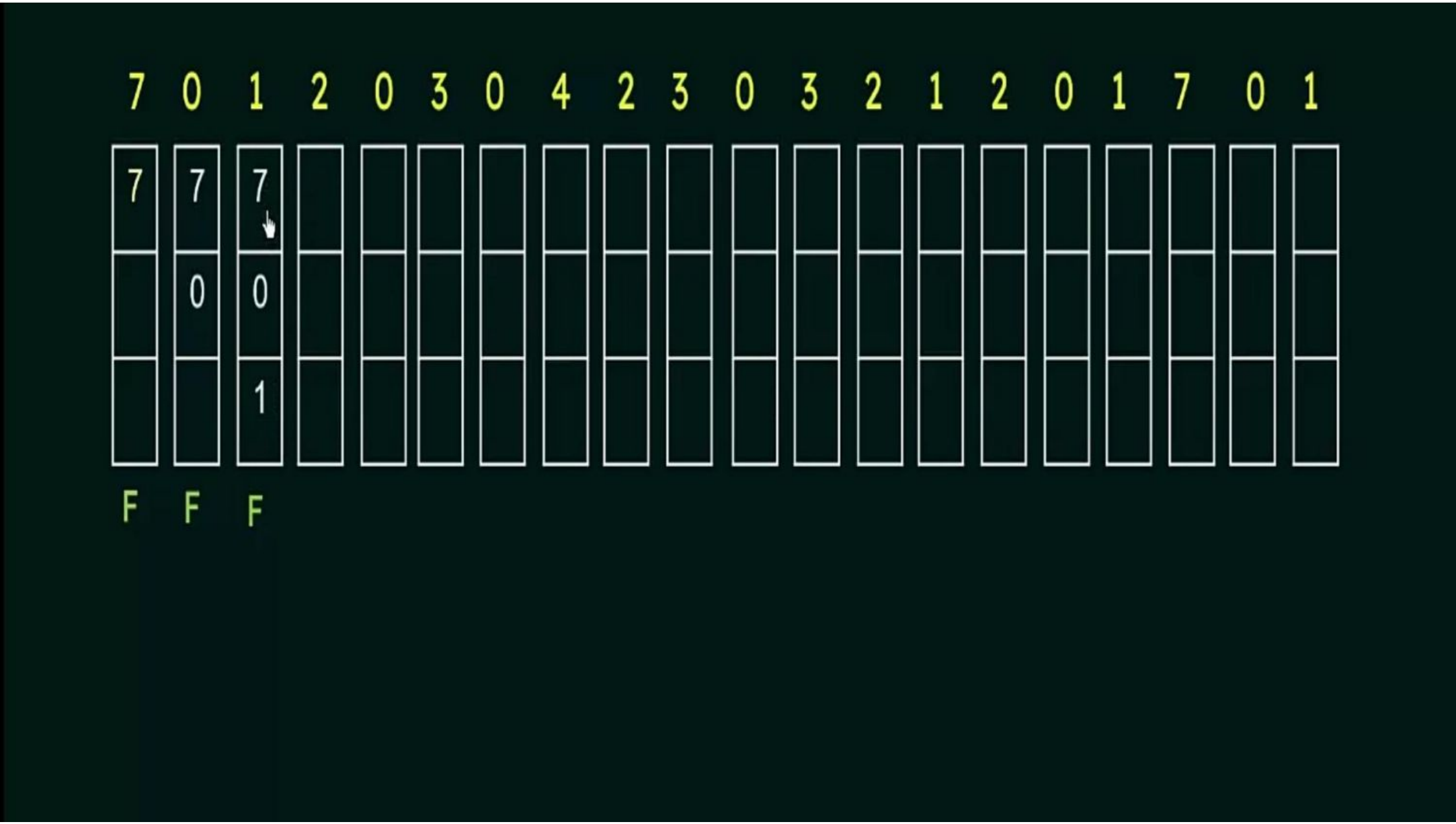
where each value represents a page.

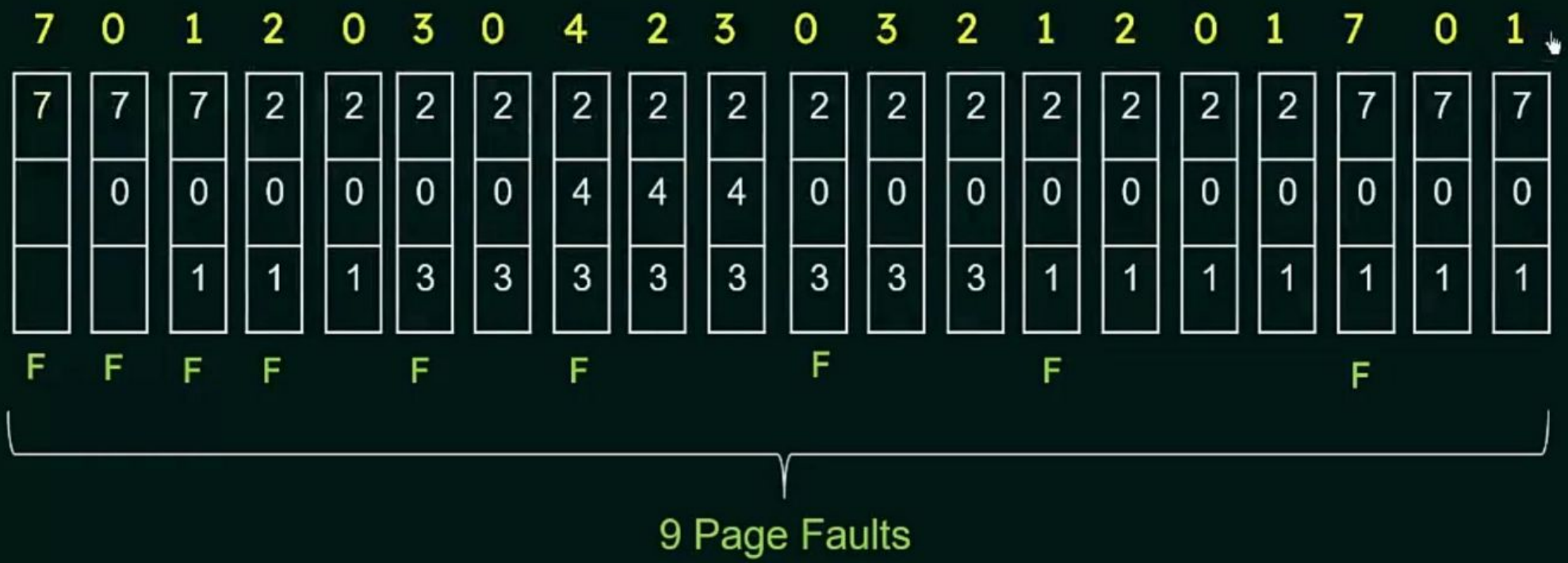
Let's say we have 3 frames in memory.




7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 2 2 4 4 4 0 0 0 0 0 0 0 7 7 7
0 0 0 0 3 3 3 2 2 2 2 2 1 1 1 1 1 0 0
1 1 1 1 0 0 0 3 3 3 3 3 2 2 2 2 2 1
F F F F F F F F F F F F F F F
15 Page Faults
when
FIFO Page Replacement
was used


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


7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7
0 0
1
F F F



Problem:

Optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

LRU PAGE REPLACEMENT

LRU Page Replacement

Least Recently Used Algorithm

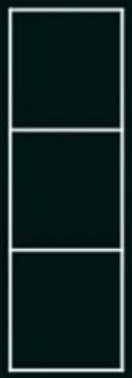
- It associates with each page the time of that page's last use.
- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
- It is almost like the Optimal Page-Replacement Algorithm looking backward in time, rather than forward.

Consider the reference string

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

where each value represents a page.

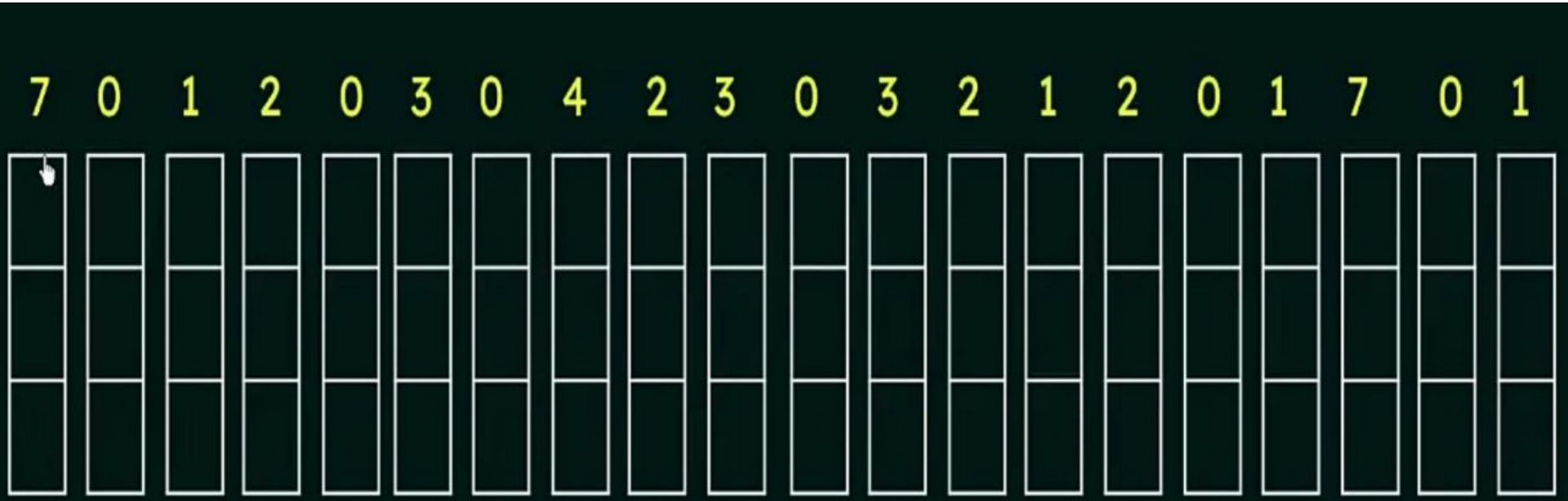
Let's say we have 3 frames in memory.

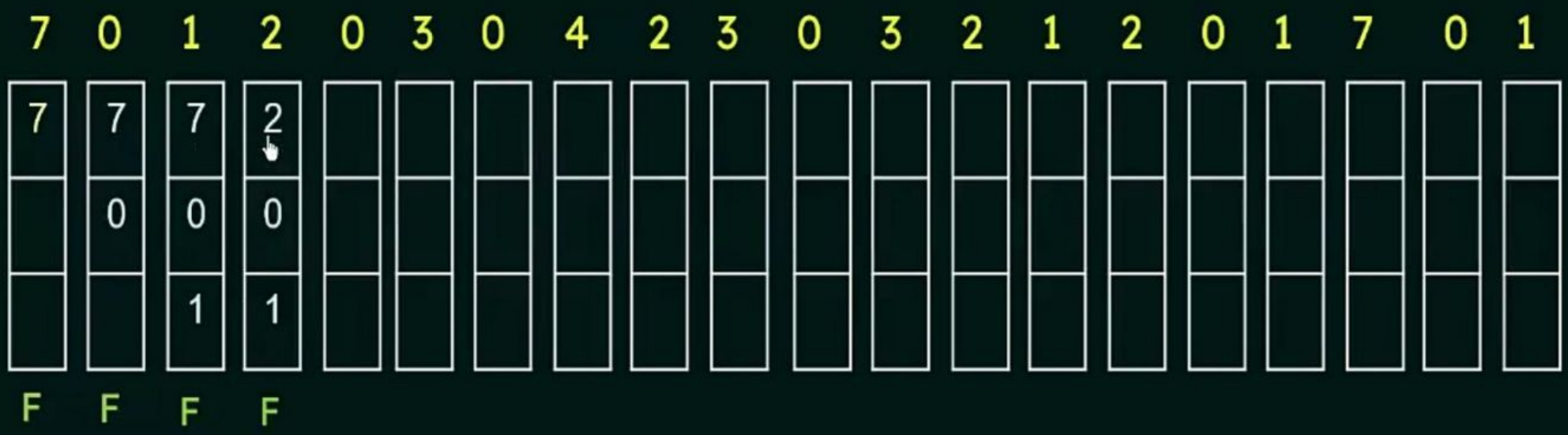


Let's apply

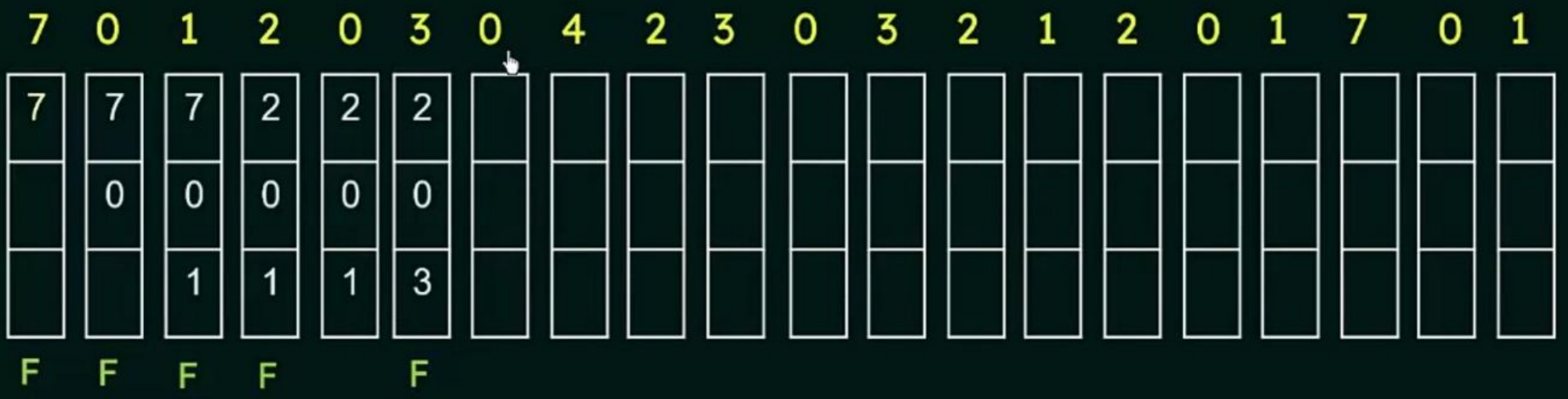
Least Recently Used Replacement Algorithm

on this reference string

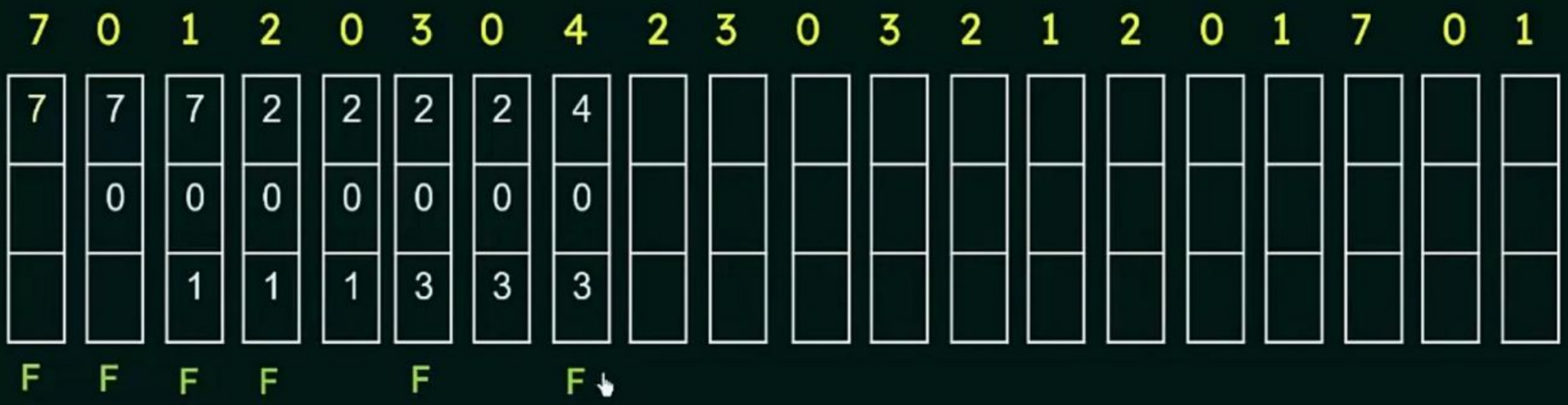

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



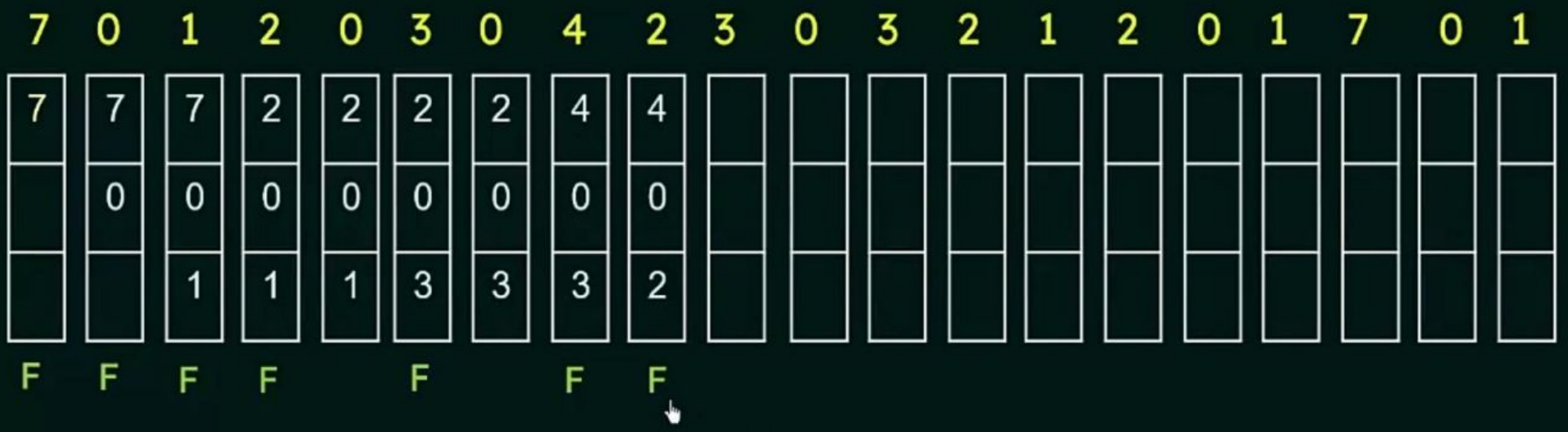
Least Recently Used Page - 7



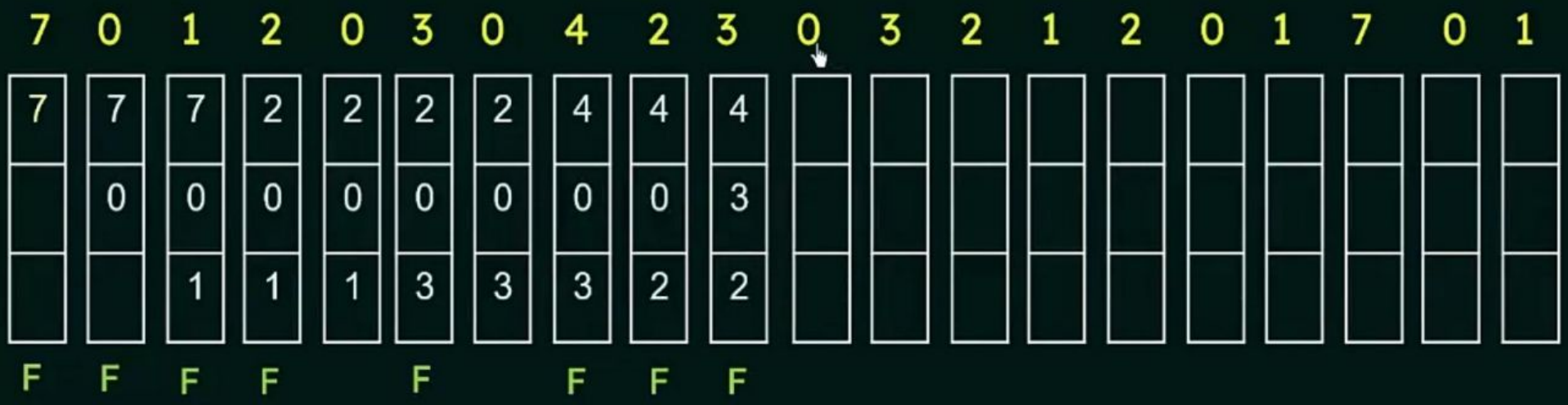
Least Recently Used Page - 1



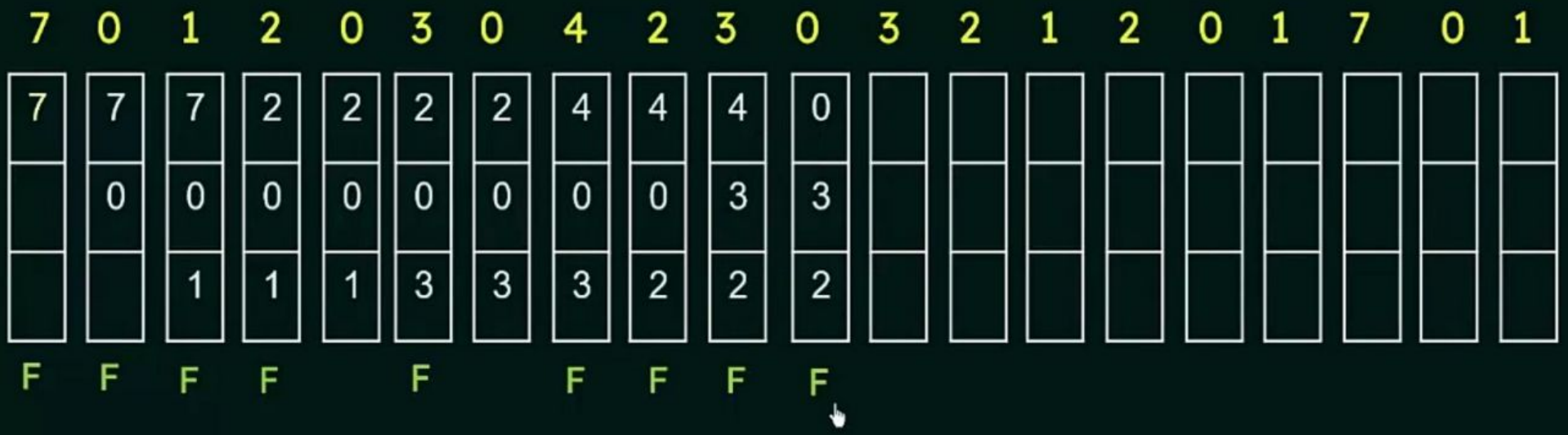
Least Recently Used Page - 2



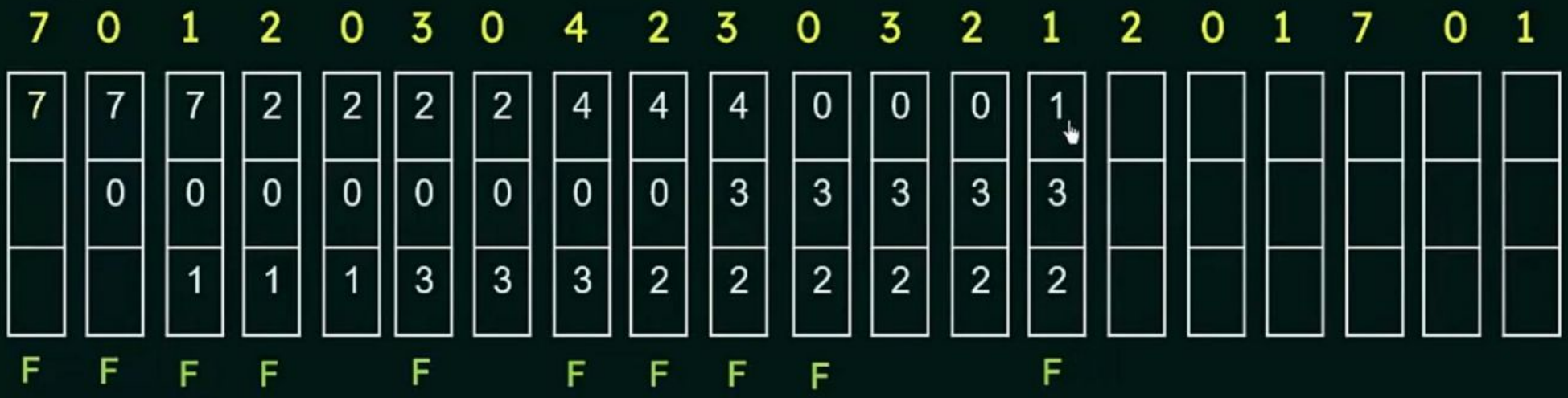
Least Recently Used Page - 3



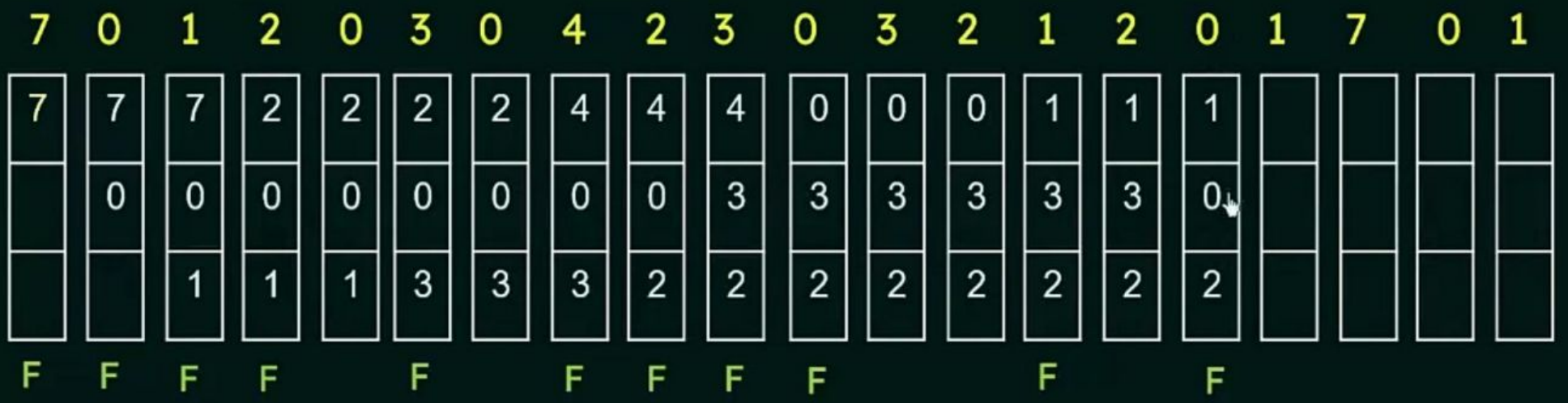
Least Recently Used Page - 0



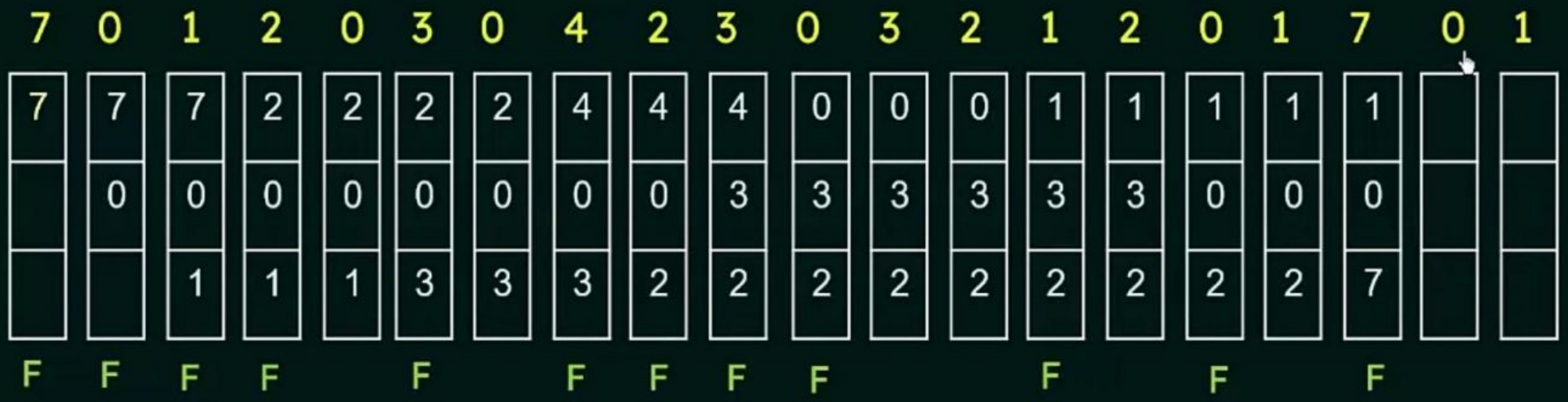
Least Recently Used Page - **4**



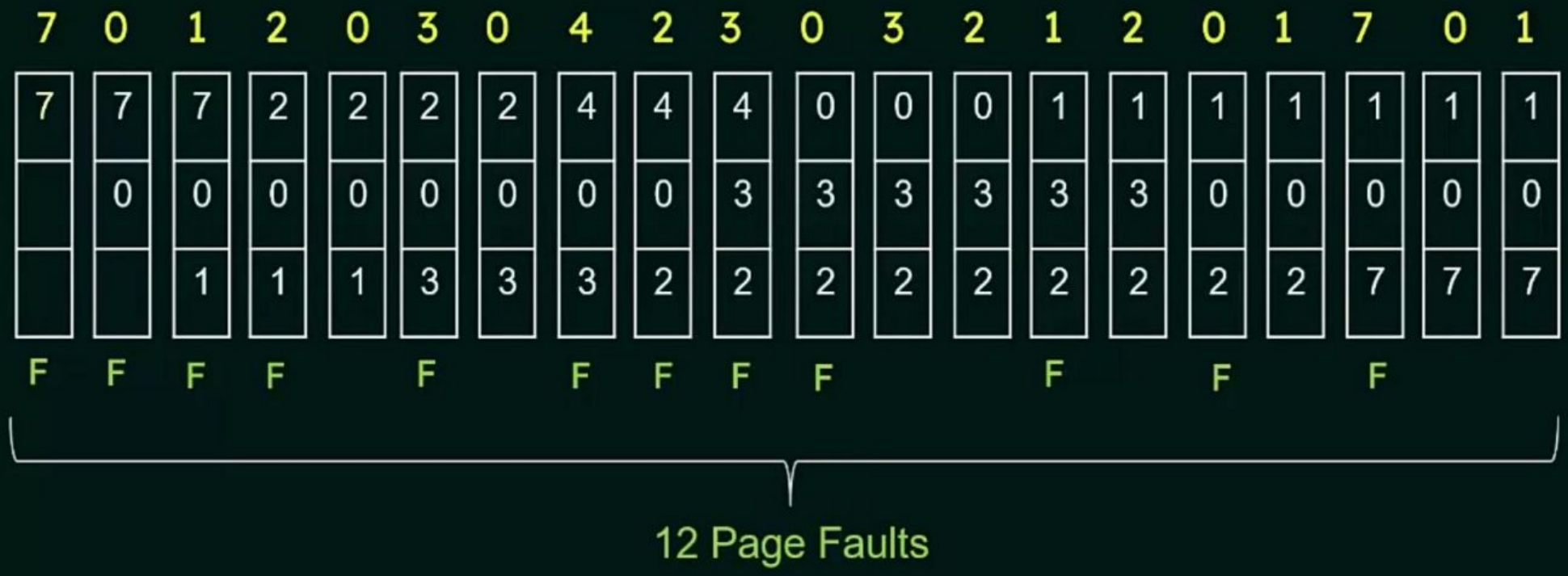
Least Recently Used Page - 0



Least Recently Used Page - 3



Least Recently Used Page - 2



The LRU page-replacement algorithm is generally considered a good algorithm and is used frequently.

Problem: How to implement it ?

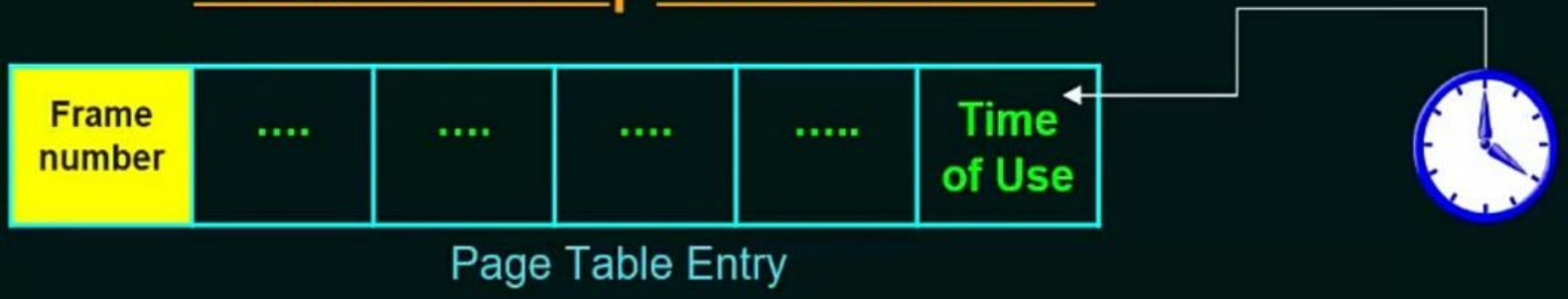
Implementation of LRU Page Replacement

- An LRU page-replacement algorithm may require substantial hardware assistance.
- The problem is – How to keep track of the pages that were least recently used?

There are two ways we can implement this:

1. By using Counters.
2. By using a Stack.

Counter Implementation



Page Table Entry

- With each Page Table Entry, we associate a time-of-use field and add a logical clock or counter to the CPU.
- The clock is incremented for every memory reference.
- Whenever a reference to a page is made, the contents of the clock register are copied to the time-of-use field in the page-table entry for that page.
- Hence, we always have the "time" of the last reference to each page.
- We replace the page with the smallest time value.

Stack Implementation

- Maintain a Stack of Page Numbers.
- Whenever a page is referenced, it is removed from the stack and put on the top.
- Hence, the most recently used page is always at the top of the stack and the least recently used page is always at the bottom.
- Since, entries must be removed from the middle of the stack, it is best to implement this approach by using a doubly linked list with a head and tail pointer.

Stack Implementation

Reference string : 4 7 0 7 1 0 1 2 1 2 7 1 2