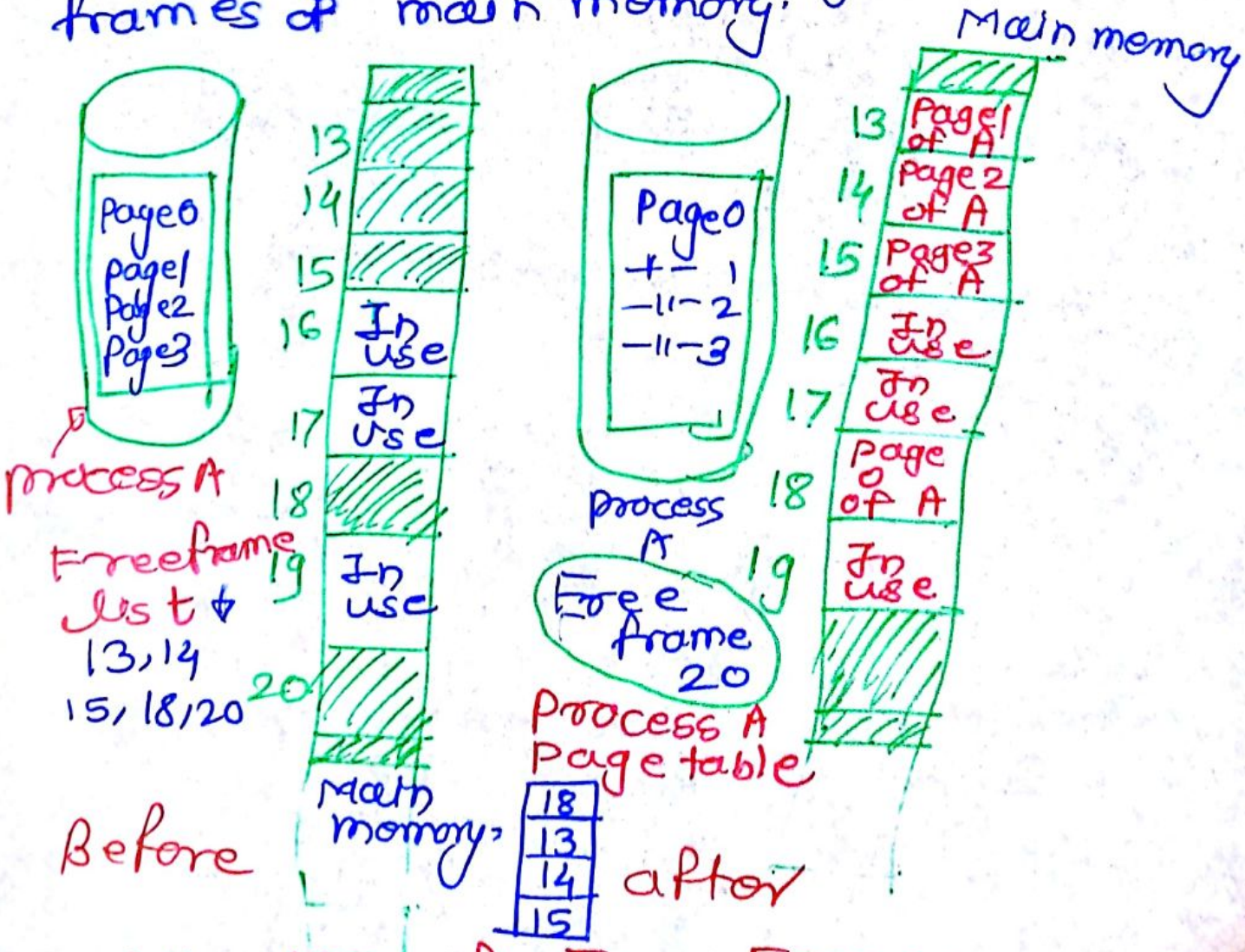
Paging

Both unequal fixed size and variable size partitions are insufficient in the use of memory. Suppose, however, that memory is partitioned into ~~#~~ equal fixed-size chunks that are relatively small called frames.

Each process is also divided into small fixed size chunks of some size. These chunks are called as pages.

These pages could be assigned to available frames of main memory.



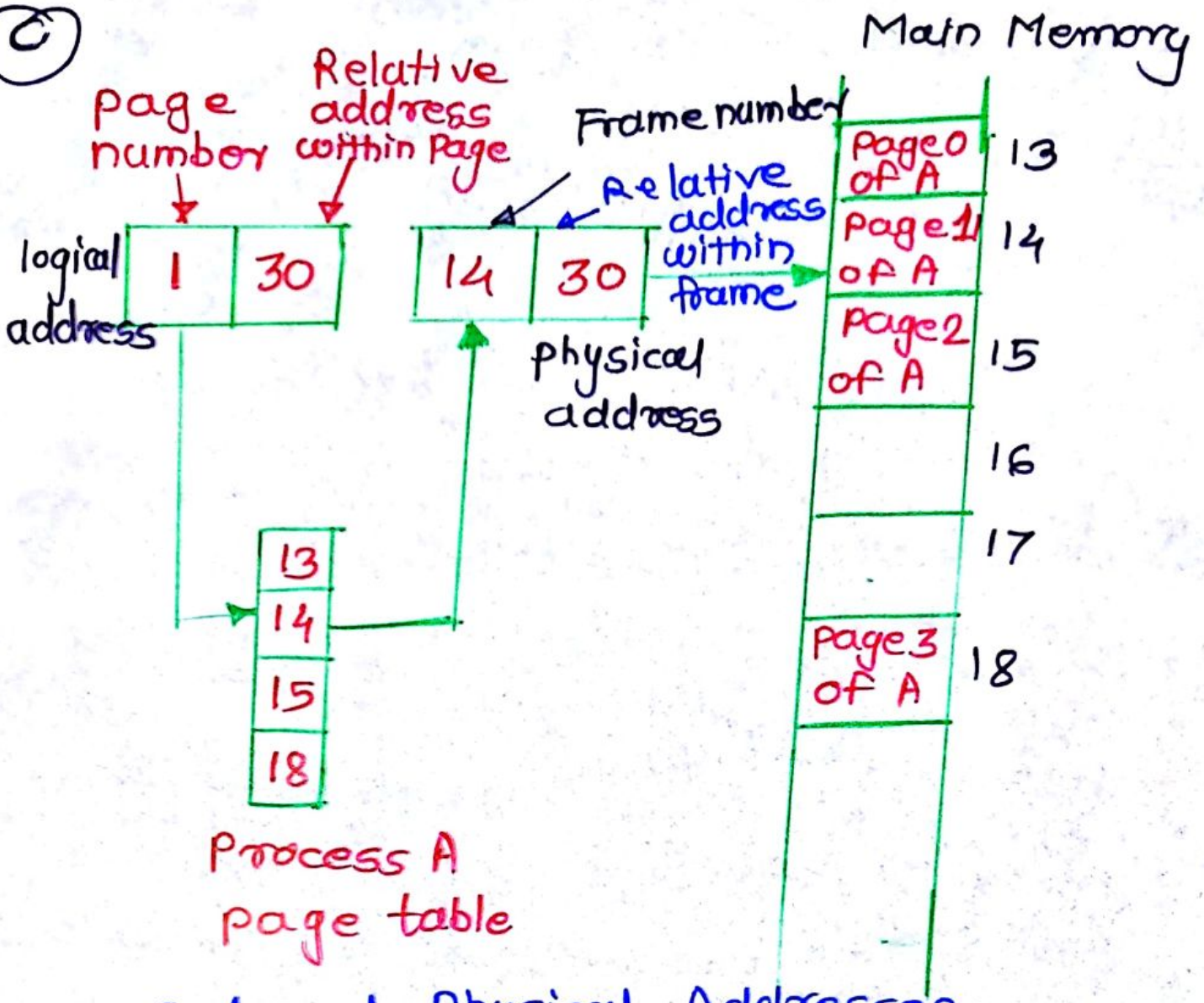
Allocation of Free Frames.

Fig shows an example of the use of pages and frames. At a given point in time, some of the frames in memory are in use and some are free. The list of free frames is maintained by the OS.

Process A stored on disk, consists of four pages. When it comes time to load this process, the OS finds 4 free frames and loads the 4 pages of the process A into four frames.

Now suppose, there are not sufficient unused contiguous frames to hold the process. Does this prevent the operating system from loading A? The answer is no, because once again we use the concept of logical address. Rather than base address OS maintains a page table for each process.

Within the program, each logical address consists of a page number and relative address within the page. Logical to physical address translation is done by processor hardware. The processor must know how to access the page table of the current process. Presented with a logical address (page no, relative address) the processor uses page table to produce a physical address (frame no, relative address).



Logical and Physical Addresses.

This approach solves the problems raised earlier. Main memory is divided into many small equal-size frames. Each process is divided into frame-size pages. Smaller processes require fewer pages. Larger processes require more. When a process is brought in, its pages are loaded into available frames and a page table is set up.

(5) **Demand Paging**: Each page of process is brought in only when it is needed, that is on demand.

simple tactic of breaking a process up into pages led to the development of important concept: virtual memory.

Consider a large process, consisting of a long program plus a number of arrays of data. Over any short period of time execution may be confined to a small section of program e.g a subroutine and perhaps only one or two arrays of data are being used.

It would be clearly wasteful to load in dozens of pages for that process when only a few pages will be used before the program is suspended.

We can make better use of memory by loading in just a few pages.

Then if program branches to an instruction in ~~memory~~, ~~a page fault is triggered~~. on a page not in main memory or if program references data on a page not in a memory, a page fault is triggered. This tells the OS to bring in the desired pages.

Thus at any one time, only few pages of any given process are in memory. Furthermore, time is saved because unused pages are not swapped in and out of memory. However the operating system must be clever about how it manages this scheme.

When it brings one page in, it must throw another page out. If it throws out a page just before it is about to be used, then it will just have to get that page again almost immediately. Too much of this leads to a condition known as thrashing The processor spends most of its time in swapping pages rather than executing instructions.

Operating system tries to guess, based on recent history, which pages are least likely to be used in the near future.

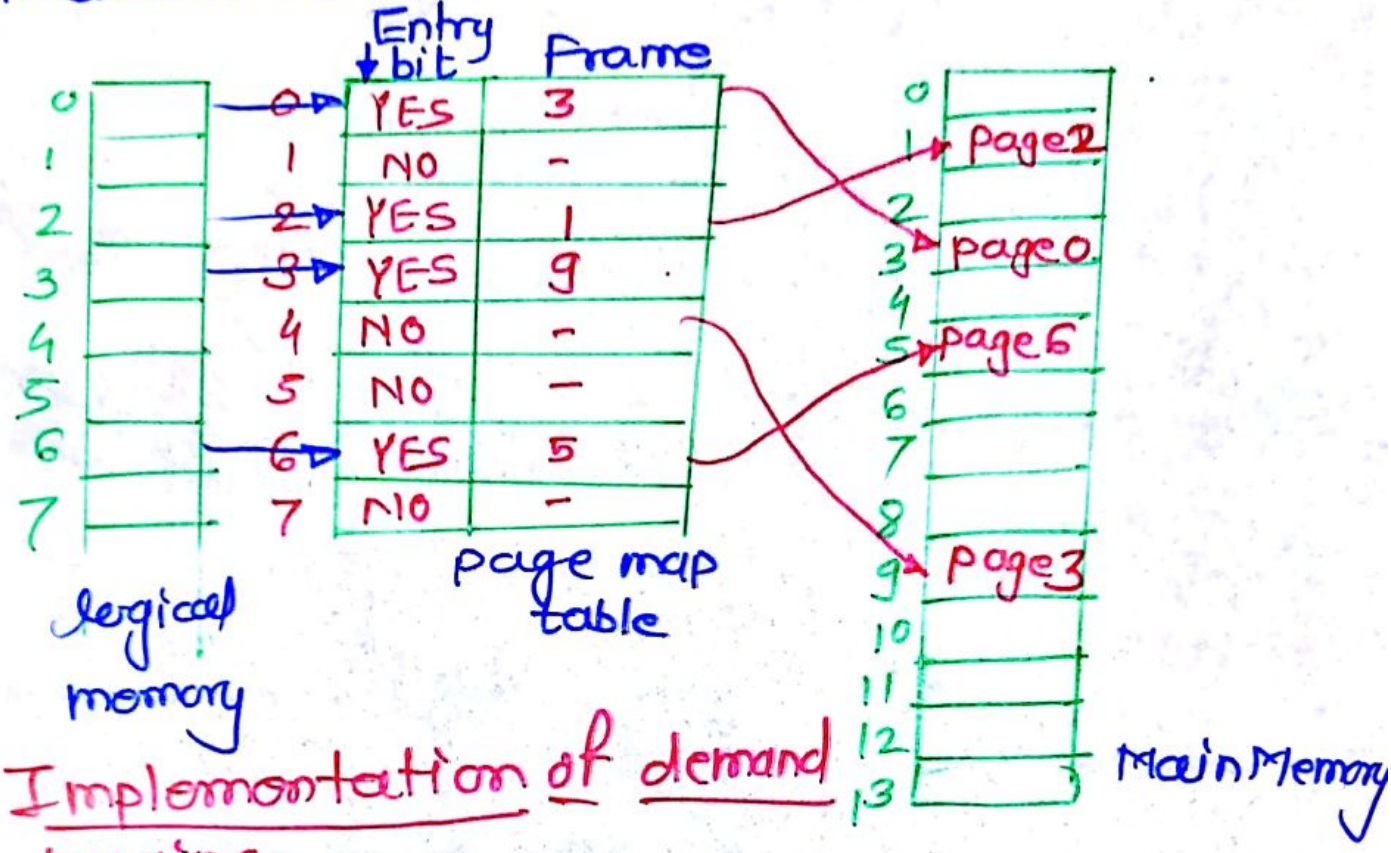
Implementation of demand paging

(1) To implement demand paging, operating system keeps track of which pages are in memory.

(2) Page map table contains an entry bit for each virtual page of the related process

(3) For each page in memory, pagemap table points to actual locations that contains the corresponding page frame and entry bit is set and marked as YES

④ If a particular page is not in memory, then the corresponding bit is reset and marked as NO.



Implementation of demand paging

If the program tries to access a page that is not in memory, page fault occurs, In response to a page fault, operating system brings in the page in demand from secondary storage. Sequence of steps in handling of page fault.

① If the process refers to a page which is not in the physical memory then an internal table kept with a process control block is checked to verify whether a memory reference to a page was valid or invalid.

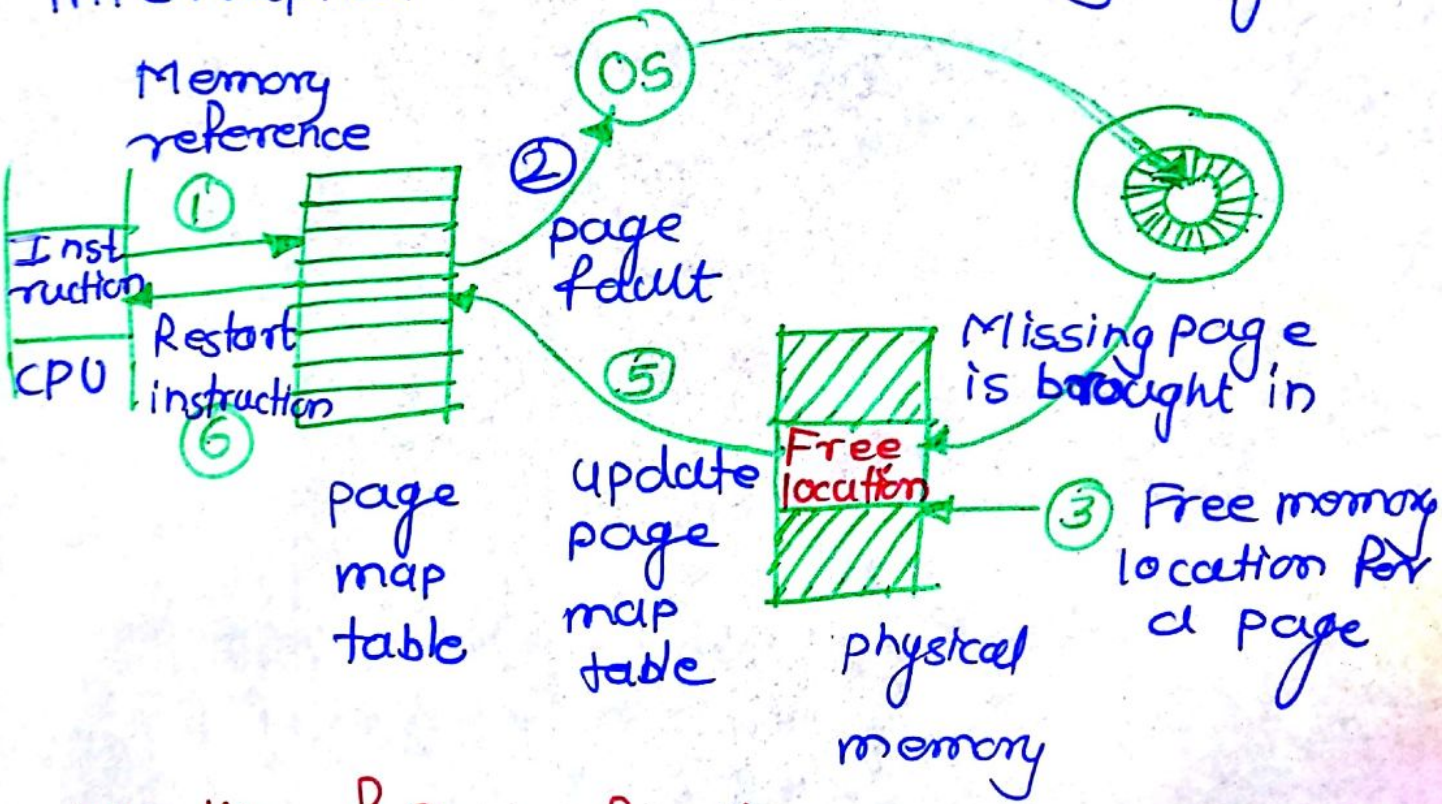
② If the memory reference to a page is valid, but that page is not in main memory the process of page transfer starts.

③ Free memory location is identified to bring the missing page.

④ Required page is brought back into the free memory location.

⑤ Once the page is in physical memory, the identical table kept with the process and page map table is updated to indicate that the page is now in memory.

⑥ Restart the instruction that was interrupted due to the missing page.



Handling of Page Fault

Advantages of demand paging

- With demand paging, it is not necessary to load an entire process into main memory.
- It is possible for a process to be larger than the physical size of main memory.
- With demand paging, only a few pages of any process are in memory, and therefore more processes can be maintained in memory.
- There is time saving because unused pages are not swapped in and out of memory.

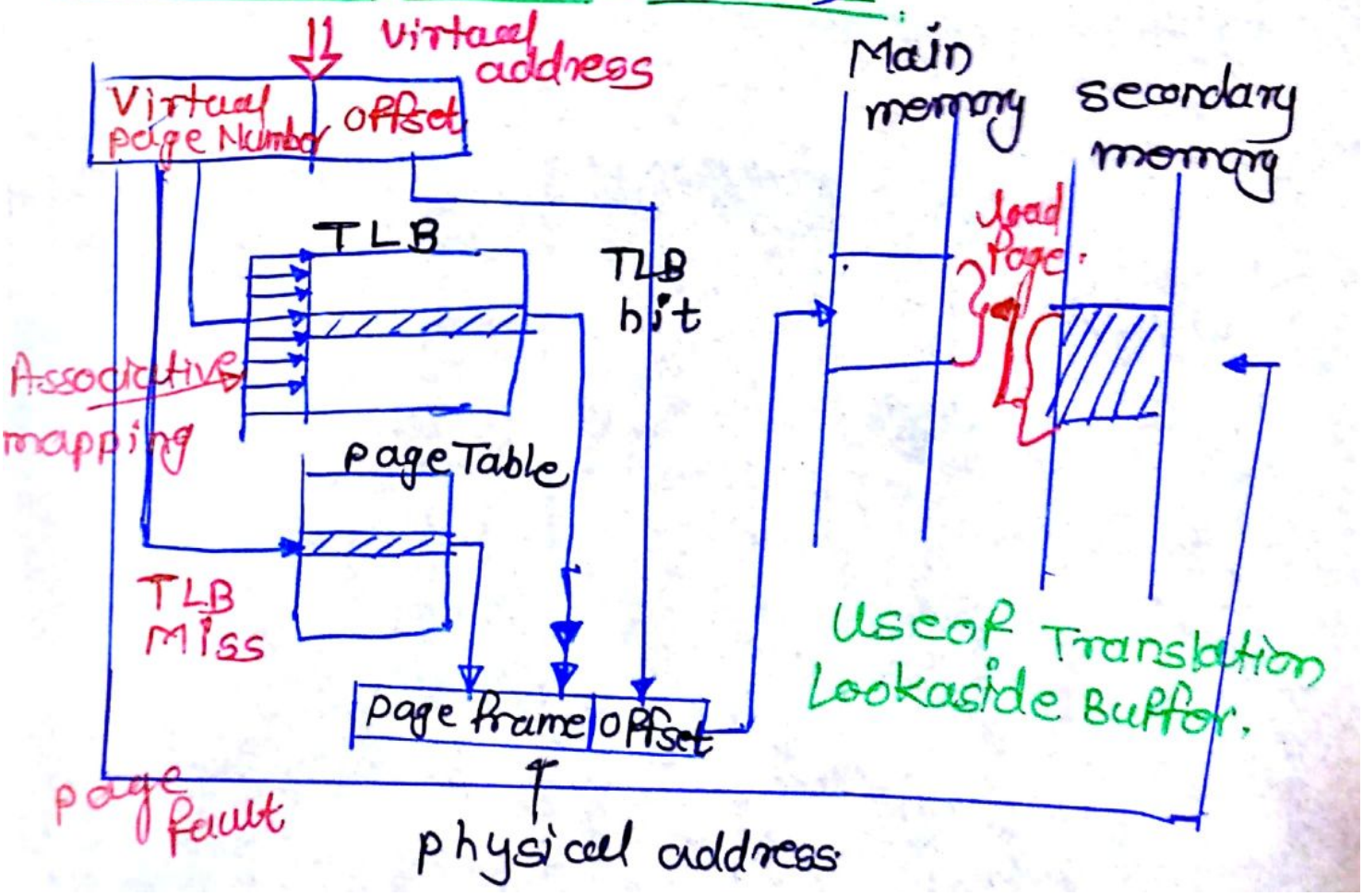
Translation lookaside Buffer. (1)

Every virtual memory reference can cause two physical memory accesses

(1) One to fetch the appropriate page table entry.

(2) One to fetch the desired data

Thus the virtual memory scheme would have the effect of doubling the memory access time. To overcome this problem most virtual memory management schemes make use of a special high speed cache for page table entries, usually called a Translation Lookaside Buffer. (TLB)



The organization of resulting paging hardware is shown. Here associative mapping is used.

- TLB contains those page entries that have been most recently used.
- An entry of TLB contains.
  1) Virtual page number
  2) control bits
  3) Page frame number in memory.

- Address translation proceeds as follows Given a virtual address, the processor looks in TLB for referenced page. If the page table entry for this page is found in the TLB, the physical address is obtained immediately. If there is a miss in the TLB, then the required entry is obtained from the page table in main memory, and the TLB is updated.
- When a program generates an access request to a page that is not in the main memory, a page fault is said to have occured. The whole page must be brought from the disk into the memory before access can proceed.

Segmentation

There is another way in which addressable memory can be subdivided known as Segmentation.

Whereas paging is invisible to the programmer and serves the purpose of providing the programmer with larger address space, segmentation is usually visible to the programmer and is provided as convenience for organizing programs and data

Segmentation allows the programmer to view memory as consisting of multiple address spaces or segments. Segments are of variable, indeed dynamic size. Typically, the programmer or operating system will assign programs and data to different segments.

Segmented organization has a number of advantages:

- concept of natural division of program in terms of code, data and stack to Handle function call is preserved in segmentation. A program can be divided into code segment stack segment and data segment.
- It is easy to share segments by different programs.

- As segments are logically independent, a program segment can be changed or recompiled at any time without affecting other segments.
- Properties of a program such as scope of a variable and access rights are naturally specified by segment. These properties can be used to protect a program from unauthorized access.
- Segments can grow dynamically, certain segment types - stacks and queues grow during runtime.
- Each process (program) has its own segment table.

Segmentation	Paging
• Program is divided into variable size segments	Program is divided into fixed size pages.
• User (or compiler) is responsible for dividing the program in segments	Division into pages is performed by the operating system.
• Segmentation is slower than paging	Paging is faster than segmentation
• Segmentation is visible to user	Paging is invisible to user
• Segmentation suffers from external fragmentation	There is no external fragmentation
• Processor uses segment no, offset to calculate absolute address	Processor uses page no, offset to calculate absolute address
• OS maintains a list of free holes in main memory	• OS must maintain a free frame list.

Replacement Policy

When a page fault (page being accessed is not in main memory) occurs, the OS has to choose a page to remove from memory to make room for the page that has to be brought in. If the page to be removed has been modified while in memory, it must be rewritten to the disk to bring the disk copy up to date. If however, the page has not been changed no rewrite is needed.

While it would be possible to pick a random page to replace at each page fault, system performance is much better if a page that is not heavily used is chosen. The criteria for selecting a page for replacement constitute a replacement policy.

The main goal in choosing a replacement policy is to maximize the hit ratio. It should minimize the number of times a referenced block is not in memory. (Fault condition) More popular replacement strategies are

1. First In First Out (FIFO)
2. Least Recently Used (LRU)
3. Optimal (OPT)

First-in First Out (FIFO)

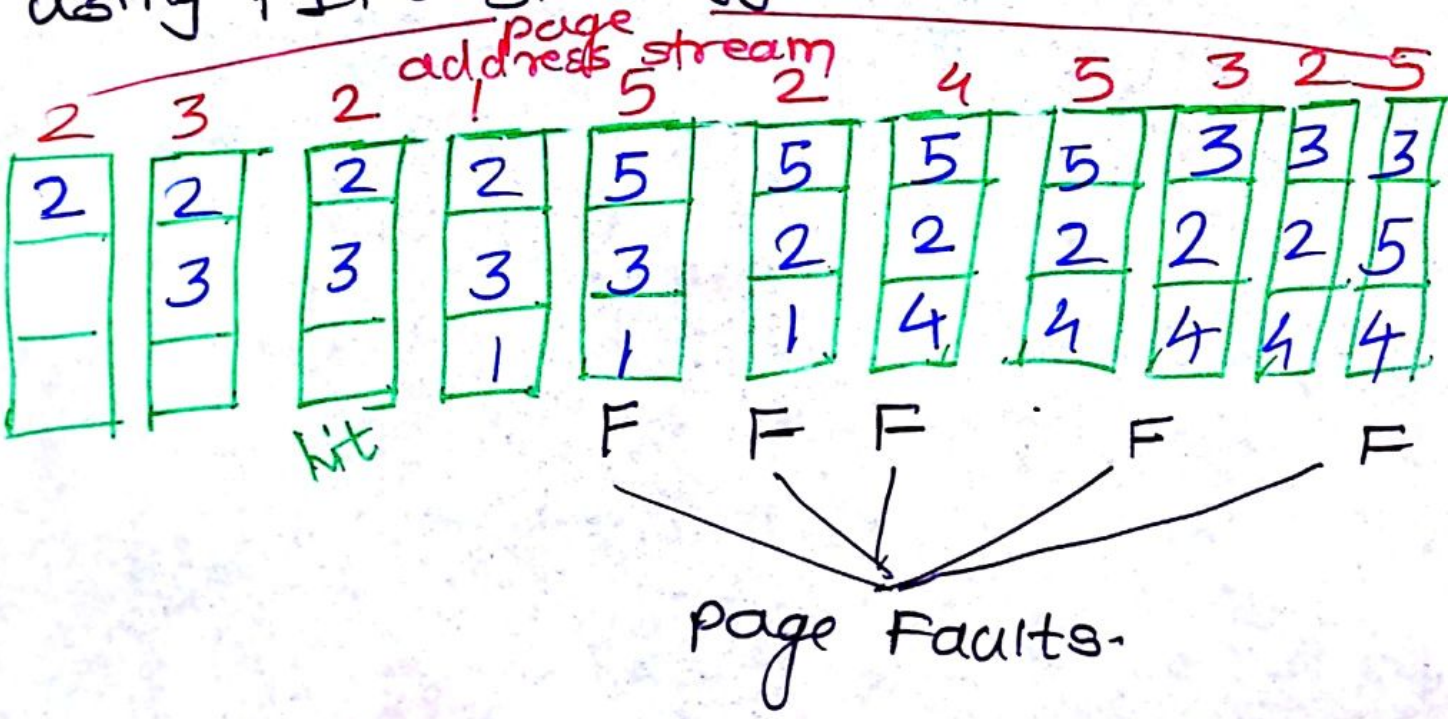
FIFO selects for replacement the page least recently loaded in memory.
It is very easy to implement.
A loading sequence number is associated with each page in memory. Memory management unit can easily find the oldest page (First-in) for replacement.

FIFO strategy does not give high hit ratio. A frequently used page, for instance, one containing a program loop, may be replaced because it is the oldest block.

Assume that the memory consists of 3 frames and during execution of a program, following pages are referenced in the sequence

2 3 2 1 5 2 4 5 3 2 5

show the behaviour of the page replacement using FIFO strategy.



LRU (Least Recently Used) (7)

LRU policy replaces the page in memory that has not been referenced for the longest time. By the principle of locality this should be the page least likely to be referenced in the near future. This policy offers near optimal hit ratio. This algorithm suggests that when a page fault occurs, throw out the page that has been unused for the longest time.

Realization of LRU algorithm is not cheap. Algorithm is likely to maintain a linked list of all pages in memory, with the most recently used page at the front and least recently used page at the rear. The difficulty is that the list must be updated on every memory reference. Finding the page in the list, deleting it, and then moving it to the front is a very time consuming operation.

Find out page fault for the following string using LRU and FIFO method (8)

6 0 12 0 30 4 2 30 32 1 20 15

consider page frame size = 3

FIFO

	6	0	12	0	30	4	2	30	32	1	20	15
	6	6	6	6	30	30	30	30	32	32	32	15
		0	0	0	0	4	4	4	4	1	1	1
			12	12	12	12	2	2	2	2	20	20
				Hit	F	F	F	Hit	F	F	F	F

LRU

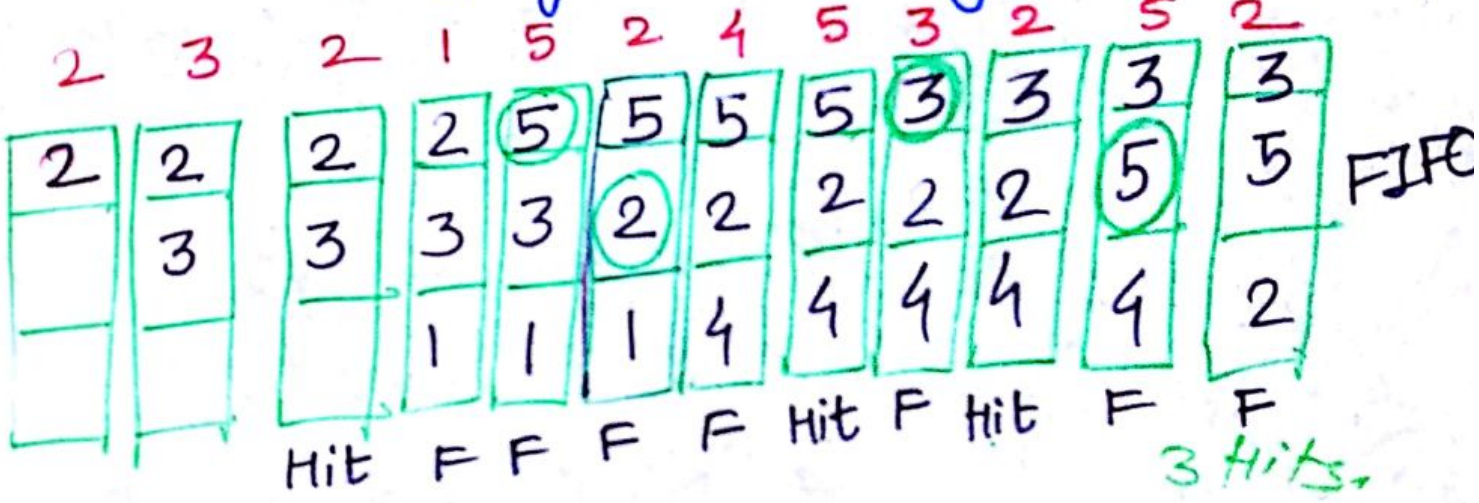
	6	0	12	0	30	4	2	30	32	1	20	15
	6	6	6	6	30	30	30	30	30	30	20	20
		0	0	0	0	0	2	2	2	1	1	1
			12	12	12	4	4	4	32	32	32	15
				Hit	F	F	F	Hit	F	F	F	F

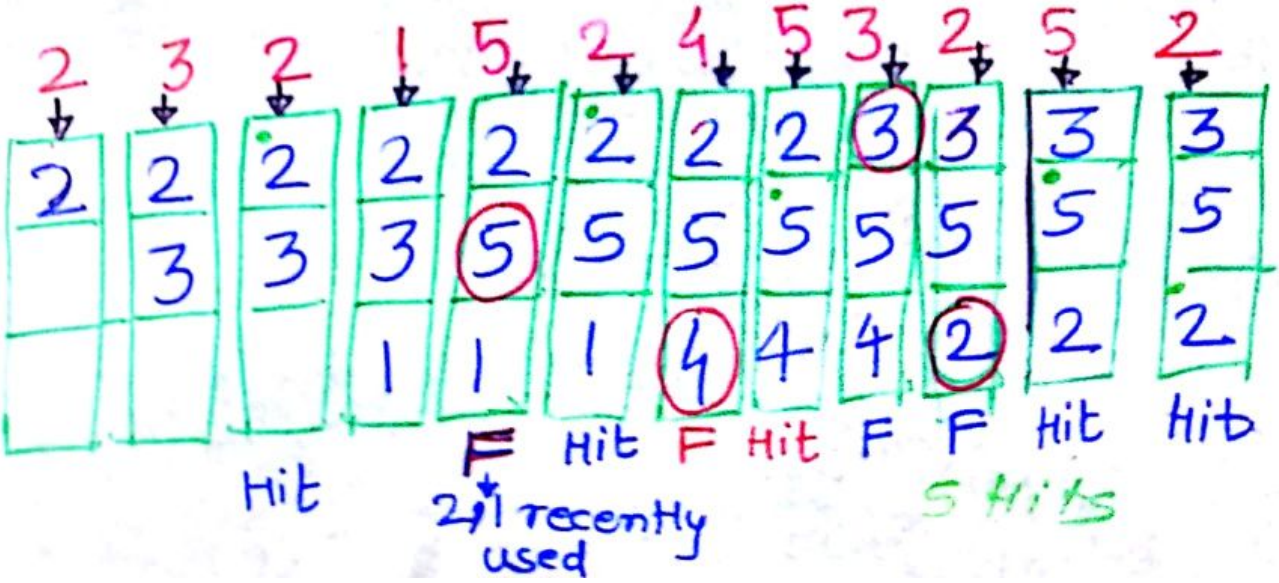
6 is used long back (not used for much time)

12 is used " " here 0 is used recently.

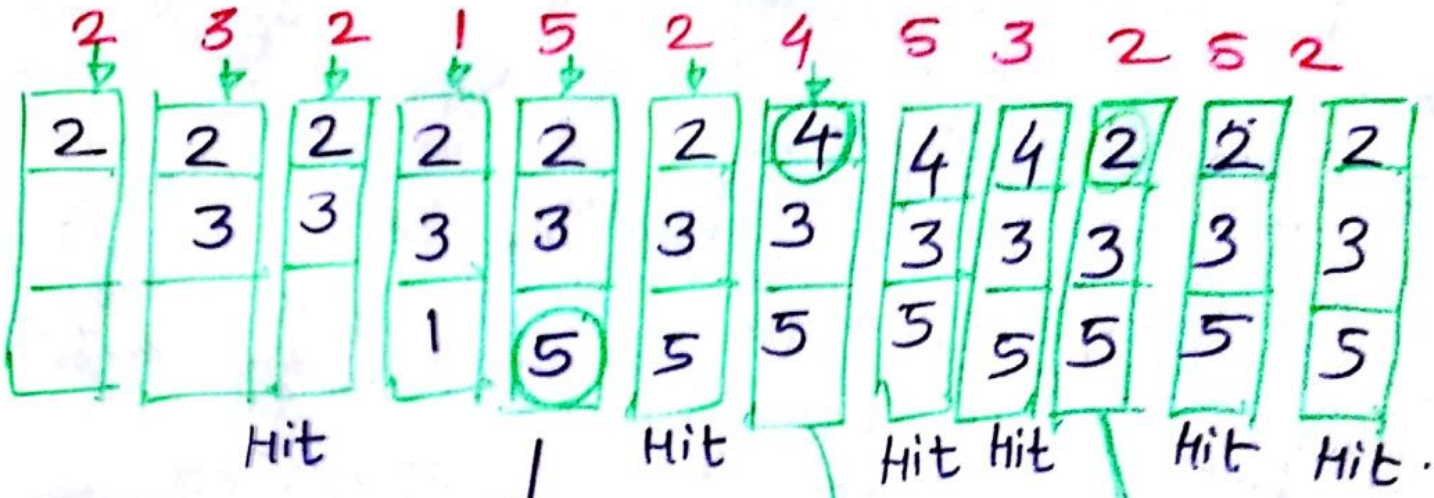
30 " " " .

Optimal : The optimal policy selects for replacement that page for which the time to the next reference is longest (Replace the page that will not be used for the longest period of time)
This algorithm results in fewest number of page fault. But this algorithm is impossible to implement. At the time of page fault, operating system has no way of knowing when each of pages will be referenced next. However it does serve as standard against which to judge other algorithms.





Optimal



we have to remove page that will not be used for longest time. see further sequence of pages for 2, 3 and 1 entries (2, 4, 5, 3, 2, 5, 2)
1 is not present (2 and 3 are present but not longest) hence remove page 1

For entry 4, look at sequence for entries of pages 2, 3 & 5.
2, 3, 5 are in sequence (5, 3, 2, 5, 2), 4 not present
2 is after long time as compared to 3 & 5. Hence remove page 2

sequence 5, 2

pg 4, 3, 5 pages, 5 is present, we can remove page 4.