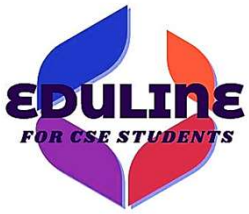
MODULE 4

CHAPTER 2 – VIRTUAL MEMORY MANAGEMENT



VIRTUAL MEMORY

- A computer can address more memory than the amount of physically installed on the system. This extra memory is actually called virtual memory and it is a section of hard disk that has setup to emulate the computers RAM
- Virtual memory is implemented using secondary storage to augment the main memory.
- Programs can be larger than the physical memory. Instead of holding the entire program , main memory hold only a portion of the program which is currently being executed.

- Virtual Memory enhances the CPU utilization and through put by executing as many programs as possible, simultaneously.

➢Virtual memory serves two purposes:

❖First it allows us to extend the use of physical memory by using Disk.

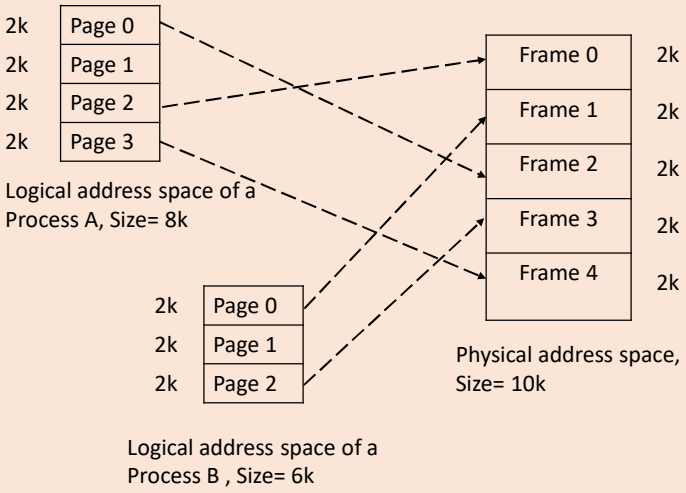
❖Second , it allows us to have memory protection, because each virtual address is translated to a physical address.

- Two most common methods of implementing virtual storage are **Paging** and **Segmentation**

PAGING

- Paging is a memory management technique.
- In this approach, physical memory is divided in to fixed sized block called frames and logical memory is also divided in to the fixed sized blocks called pages.
- The size of the page is same as that of frame.
- The key idea of this method is to place the pages of a process in to the available frames of memory , whenever this process is to be executed.

❖Paging Concept



❖ADDRESS TRANSLATION

- It is done by mapping. Logical address is mapped to physical address using a page table
- Every address generated by the CPU is divided into two parts
  page number (p) and **pageoffset (d)**.
- Physical address is represented by two parts:
  Frame number(f) and **page offset(d)**
- The page number is used as an index into a **page table**.
- The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

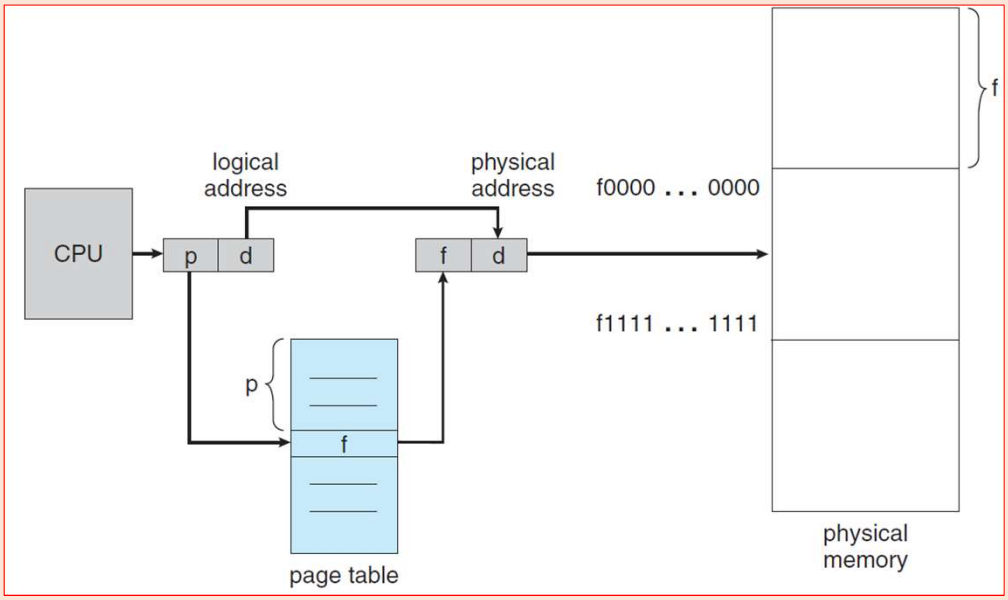
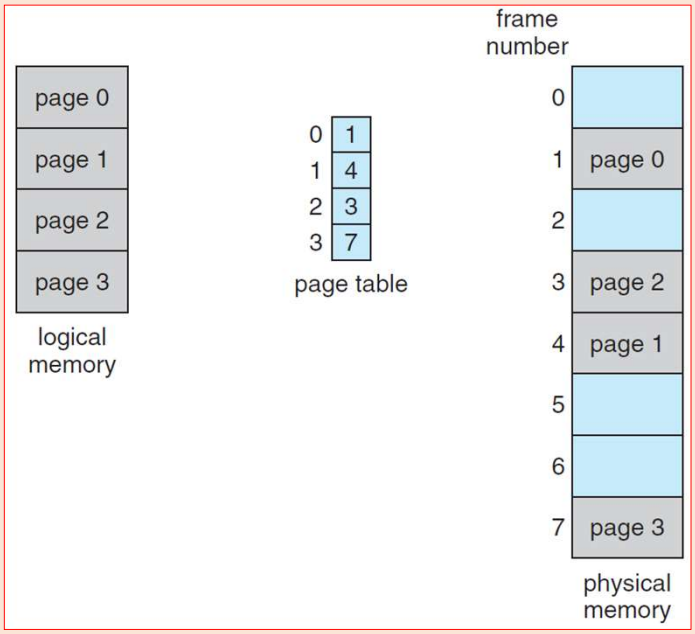


Fig: Paging hardware

- Paging model of logical and physical memory



- Page size is defined by the hardware. The size of the page is typically a power of 2.
- At physical level addresses are represented in binary form
- In binary system it is easy to handle various activities if everything is expressed in power of 2 and this is to make the division of a logical address into page number and page offset easy.
- To perform multiplication and division in power of 2, we only need to shift left or right the value to be multiplied. Thus the address computation becomes easy.

❖Advantages and Disadvantages of Paging

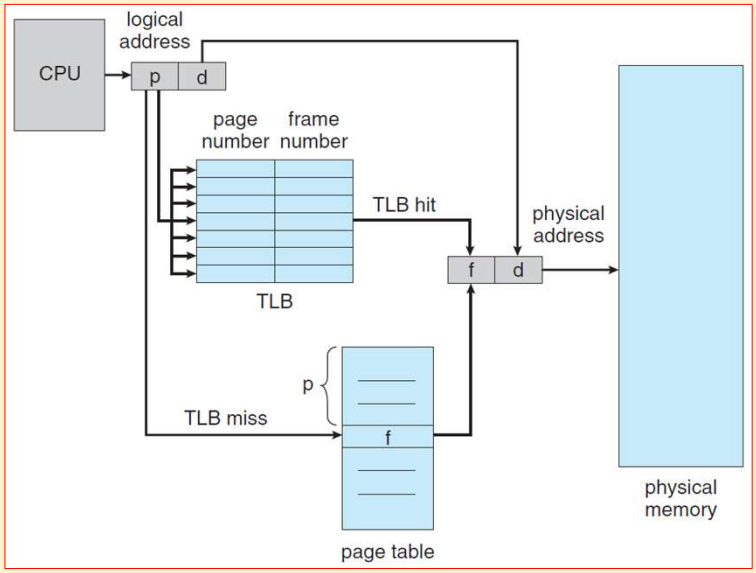
- ➢There is never a possibility of external fragmentation, but still suffer from internal fragmentation
- ➢Paging is simple to implement and assumed as an efficient memory management technique
- ➢Due to equal size of the pages and frames , swapping becomes very easy
- ➢Page table requires extra memory space, so may not be good for a system having small RAM

HARDWARE SUPPORT

- Each operating system has its own methods for storing page tables.
- The hardware implementation of the page table can be done in several ways.
- In the simplest case, the page table is implemented as a set of dedicated **registers**. The use of registers for the page table is satisfactory if the page table is reasonably small.
- If the page size is high, we store the page table in the primary memory , therefore memory access time is increased.
- That is we need 2 memory access to fetch instruction from the memory. One is to access the page table and the other is to fetch actual instruction.

- The standard solution to this problem is to use a special, small, fast lookup hardware cache called a **translation look-aside buffer (TLB)**.
- The TLB is associative, high-speed memory.
- Each entry in the TLB consists of two parts : a key (or tag) and a value.
- When the associative memory is presented with an item, the item is compared with all keys simultaneously.
- If the item is found, the corresponding value field is returned.
- The search is fast; the hardware, however , is expensive. Therefore the number of entries in a TLB is small.

Paging Hardware with TLB



- The TLB is used with page tables in the following way.
- The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB.
- If the page number is found, its frame number is immediately available and is used to access memory.
- If the page number is not in the TLB (known as a **TLB miss**), a memory reference to the page table must be made.
- When the frame number is obtained, corresponding changes are made in the TLB , so that they will be found next time very quickly.
- If the TLB is already full of entries, an existing entry must be selected for replacement. Replacement policies range from least recently used (LRU) through round-robin to random.

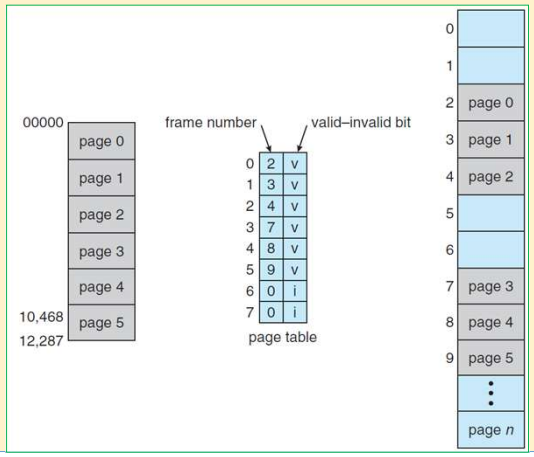
❖PROTECTION

- Memory protection in a paged environment is accomplished by protection bits associated with each frame. Normally, these bits are kept in the page table.
- One additional bit is generally attached to each entry in the page table: a **valid–invalid** bit.
- When this bit is set to *valid,* the associated page is in the process's logical address space and is thus a legal (or valid) page.
- When the bit is set to *invalid,* the page is not in the process's logical address space.
- Illegal addresses are trapped by use of the valid–invalid bit.

❖If the page is invalid, one of the following assumptions may occur:

➢The page is invalid

➢The page is valid, but currently not available in the primary memory. This is also called page fault. The page fault services are done.



Valid bit (v)

Invalid bit (i)

SEGMENTATION

- **Segmentation** is a memory-management scheme that supports this programmer view of memory.
- Here each job is divided in to several segments of different sizes.
- Segments are variable size.
- So each segment has a base and limit.
- Limit is provided for avoiding segment overlapping.
- A program segment contains the program's main function, utility functions, data structures and so on.

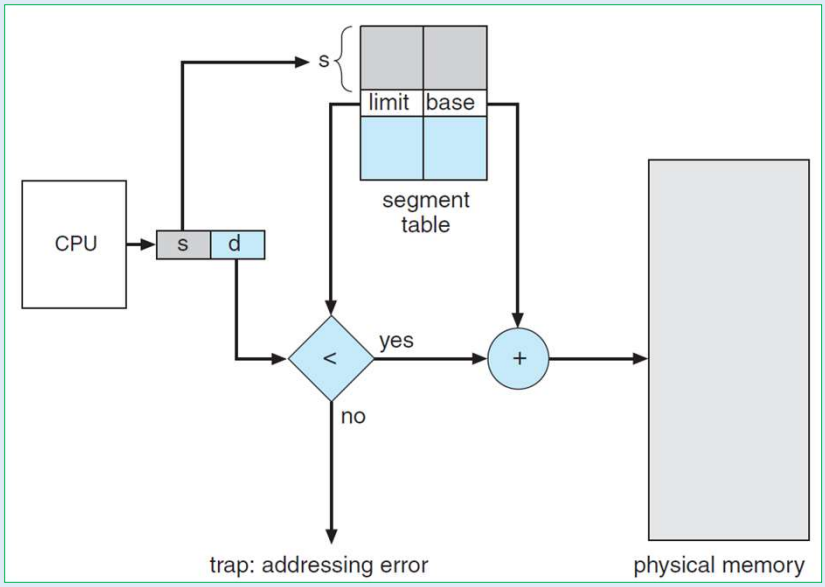
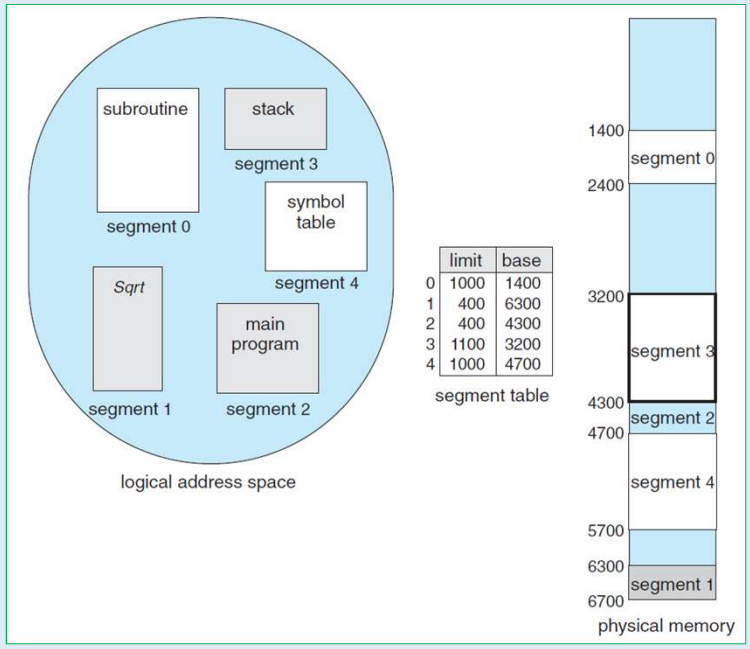


Fig: Segmentation Hardware

- A segment table is provided. Each entry of the segment table has a segment base and a segment limit.
- Segment base contains the starting physical address where the segment resides in memory, whereas segment limit specifies the length of the segment.
- A logical addressconsists of two parts: a segment number ***s***, and an offset into that segment ***d***.
- The segment number is used as an index to the segment table.
- The offset *d* of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system.
- When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base–limit register pairs



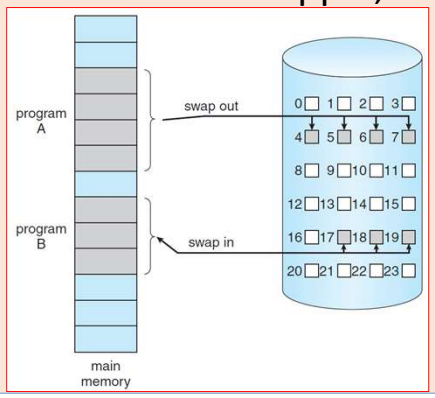
Example of Segmentation.

- As an example, consider the situation shown in above Figure . We have five segments numbered from 0 through 4.
- The segments are stored in physical memory as shown. The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the length of that segment (or limit).
- For example, segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 + 53 = 4353. A reference to segment 3, byte 852, is mapped to3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment0 would result in a trap to the operating system, as this segment is only 1,000bytes long.

DEMAND PAGING

- A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory (usually a disk).
- When we want to execute a process, we swap it into memory.
- Any page execution is started on a page fault.
- In demand paging firstly no programs are in memory.
- When CPU generate an address, a page fault will occur. When a page fault occurs, we can load the entire program in to main memory or we can load only the needed program.

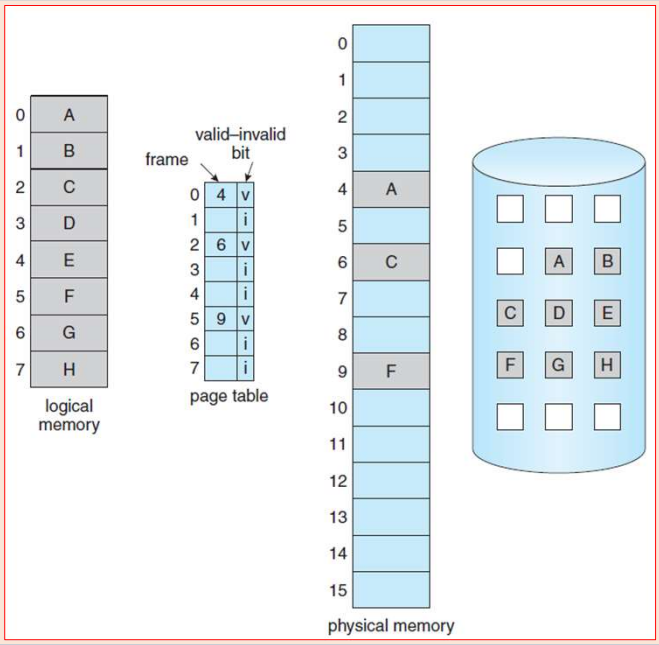
- A lazy swapper never swaps a page into memory unless that page will beneeded.
- A swapper manipulates entire processes, whereas a **pager** is concerned with the individual pages of a process.
- We thus use "**pager**," rather than "swapper," in connection with demand paging.



❖Basic Concepts

- With this scheme, we need some form of hardware support to distinguish between the pages that are in memory and the pages that are on the disk.
- The valid–invalid bit scheme can be used for this purpose.
- When this bit is set to "valid," the associated page is both legal and in memory.
- If the bit is set to "invalid," the page either is not valid (that is, not in the logical address space of the process) or is valid but is currently on the disk.

Page table when some pages are not in main memory



- Access to a page marked invalid causes a **page fault**.
- The paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system.
- The procedure for handling this page fault is straightforward

1. We check an internal table (usually kept with the process control block)for this process to determine whether the reference was a valid or aninvalid memory access.
2. If the reference was invalid, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in.
3. We find a free frame (by taking one from the free-frame list, for example).

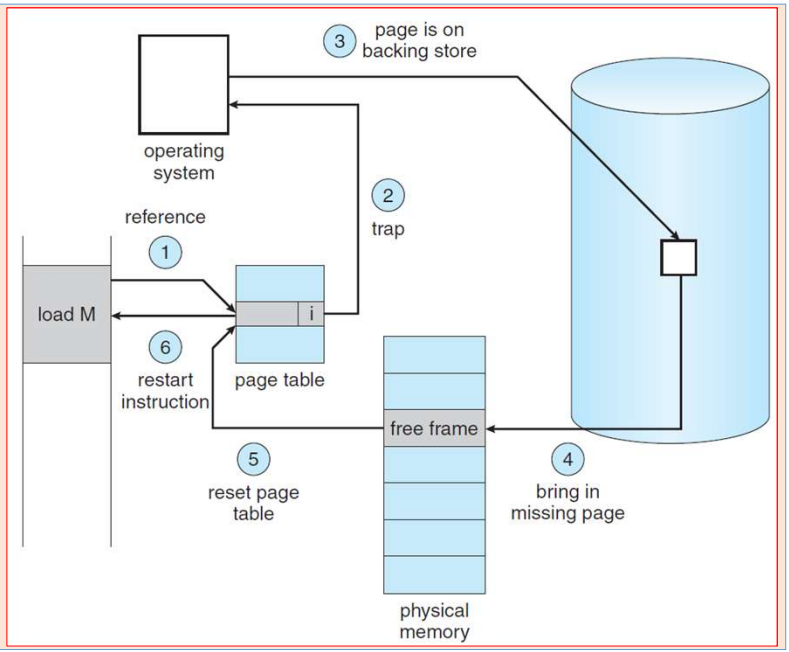
4. We schedule a disk operation to read the desired page into the newlyallocated frame.
5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

- The efficiency of demand paging is increased by using locality of reference, because continuous hit is occurred. Hardware support for demand paging is

✓The page table must have valid/invalid bit

✓A swap area must need for performing swap out/swap in

❖**Steps in handling a page fault**

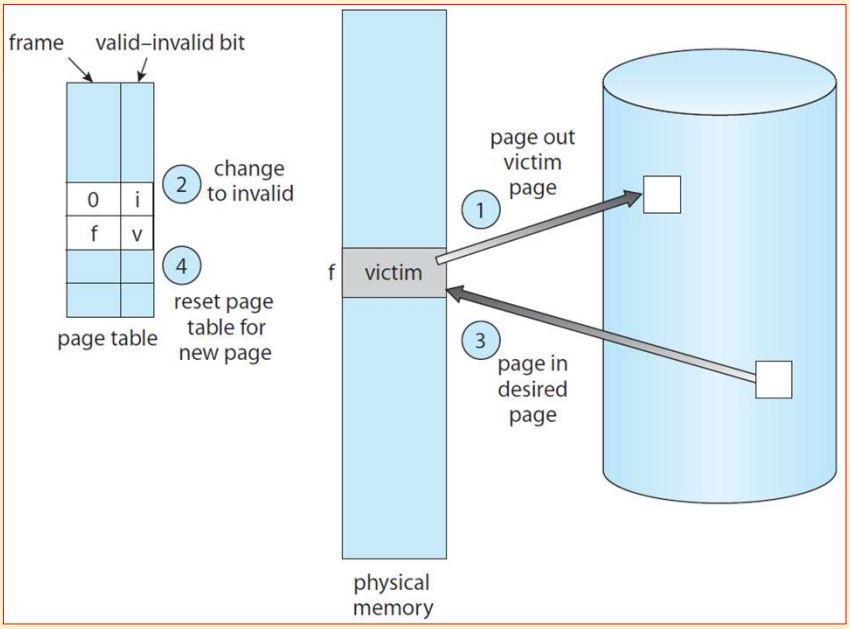


PAGE REPLACEMENT

• Page replacement takes the following approach.

1. Find the location of the desired page on the disk.
2. Find a free frame:
   - **a.** If there is a free frame, use it.
   - **b.** If there is no free frame, use a page-replacement algorithm to select a **victim frame**.
   - **c .** Write the victim frame to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Continue the user process from where the page fault occurred.

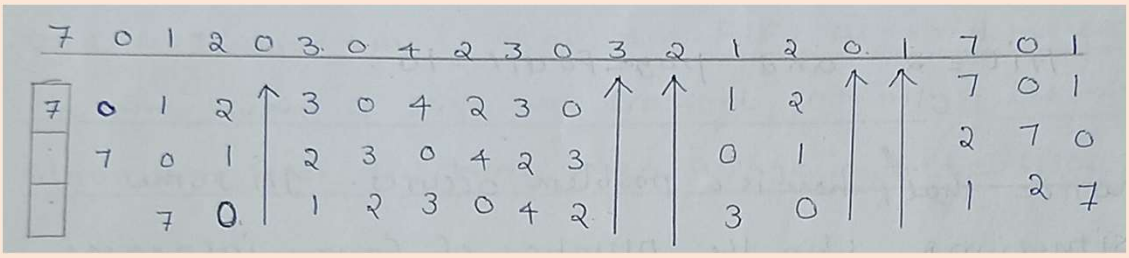
Page Replacement



PAGE REPLACEMENT ALGORITHM

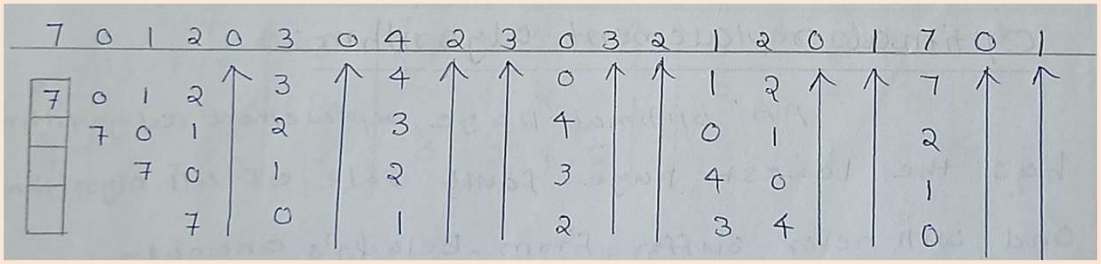
1. FIFO

- Simplest page replacement algorithm
- When a page wants to be replaced, the oldest page will be chosen.



- Hit = 5, Page fault=15 , Hit ratio= 5/(15+5) = 5/20

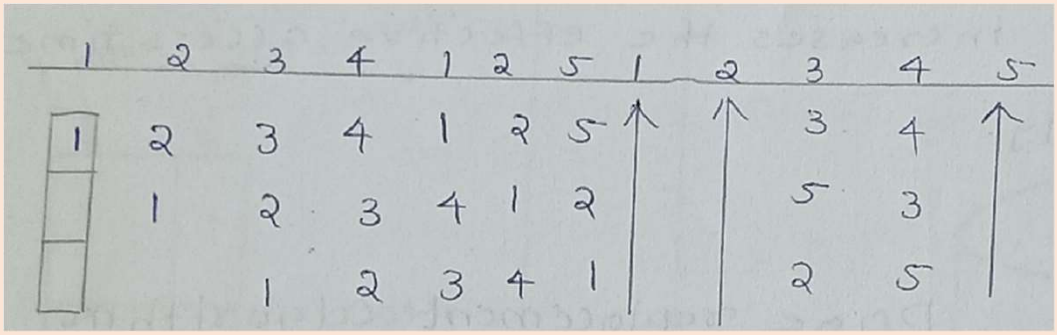
- When the frame number is 4 , it can be performed as



Hit=10 and page fault=10

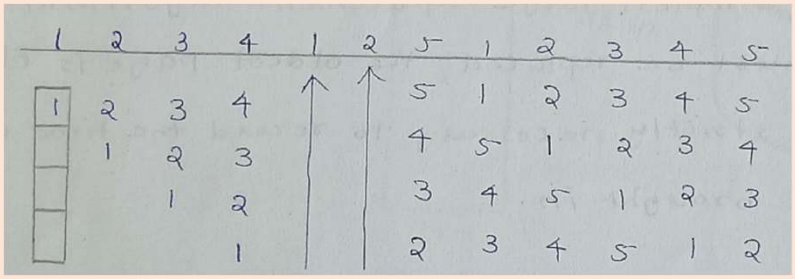
- That is, when number of frame increases, the page fault rate is decreased.

➢Consider the following example of frame number 3 and 4



Hit = 3 , Page fault = 9

When frame number =4

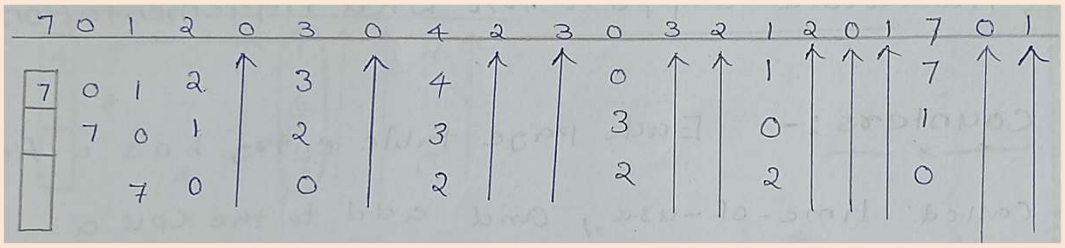


Hit=2 and page fault=10

- In some situations, when the number of frame increases, the page fault rate is also increases. It is only occur in FIFO algorithm. This is known as **Belady's anomaly**

2. Optimal Replacement algorithm

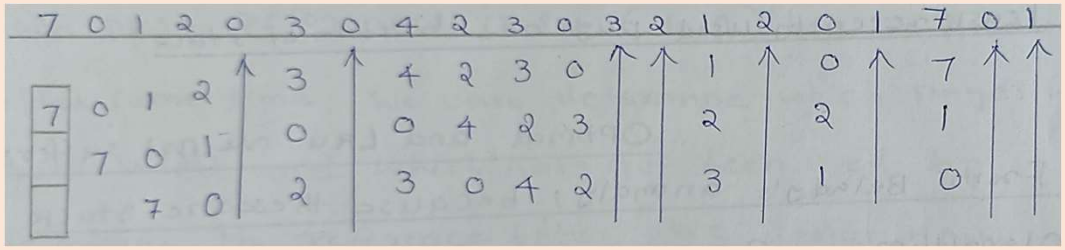
- Optimal page replacement algorithm has the lowest page fault rate of all algorithms, and will never suffer from Belady's anomaly.
- It is simply, replace the page that will not be used for the longest period of time.



- Hit=11, Page fault=9. Difficult to implement because it requires future knowledge of the reference string

3. LRU (Least Recently Used) Page Replacement

- LRU chooses the page that has not been used for the longest period of time.
- This strategy looking backward in time rather than forward.



Hit=8 and page fault=12

❖Hardware Support for LRU implementation

➢Counters

- we associate with each page-table entry a time-of-use field and add to the CPU a logical clock or counter.
- 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.
- We replace the page with the smallest time value.

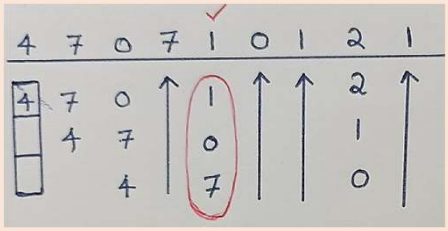
➢Stack

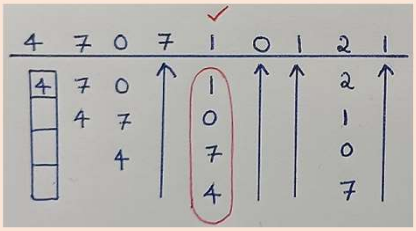
- Another approach to implementing LRU replacement is to keep a stack of page numbers.
- Whenever a page is referenced, it is removed from the stack and put on the top.
- In this way, the most recently used page is always at the top of the stack and the least recently used page isalways at the bottom.
- Here doubly linked list is used for implementing stack

  Most recently used page number – Top of stack

  Least recently used page number - Bottom of stack

- Like optimal replacement, LRU replacement does not suffer from Belady's anomaly. Both algorithms are called **stack algorithms.**
- A stack algorithm is an algorithm for which it can be shown that the set of pages in memory for *n* frames is always a ***subset*** of the set of pages that would be in memory with *n*+ 1 frames.





- Consider a particular instant. We can see that LRU with frame number 3 is a subset of LRU with frame number 4.