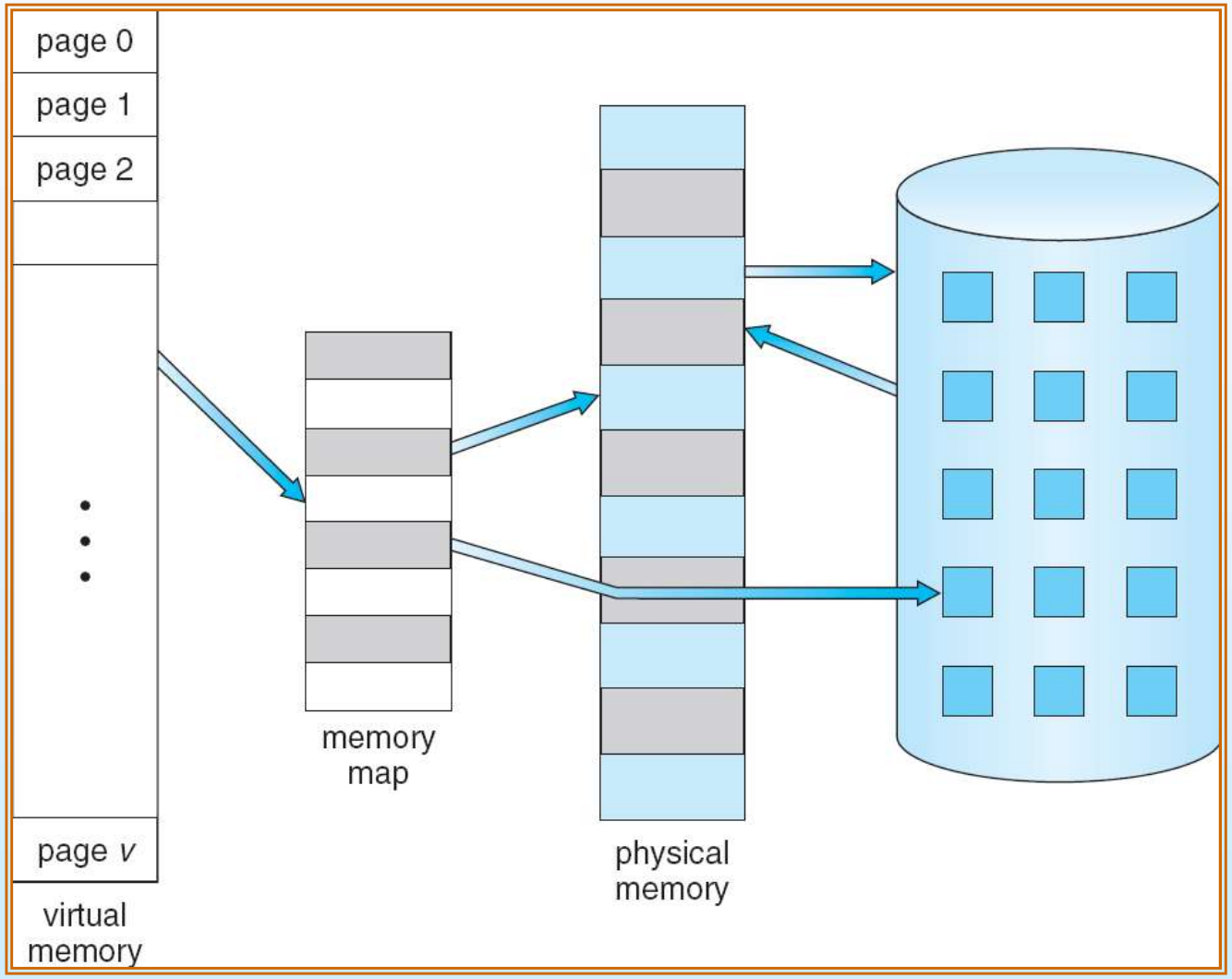
Virtual Memory

Dr. SUBHASIS DASH
SCHOLE OF COMPUTER ENGINEERING.
KIIT UNIVERSITY
BHUBANESWAR

Background

- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Virtual Memory That is Larger Than Physical Memory

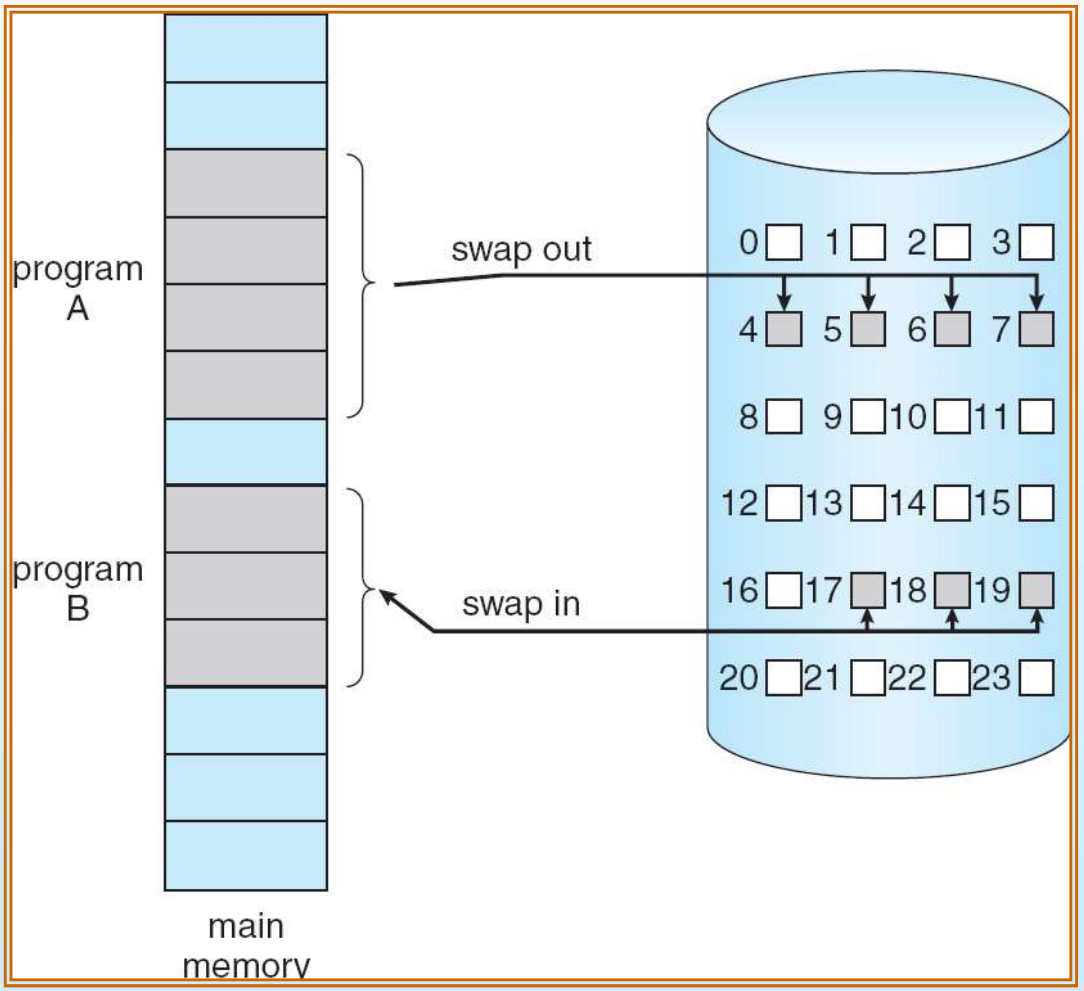


Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users

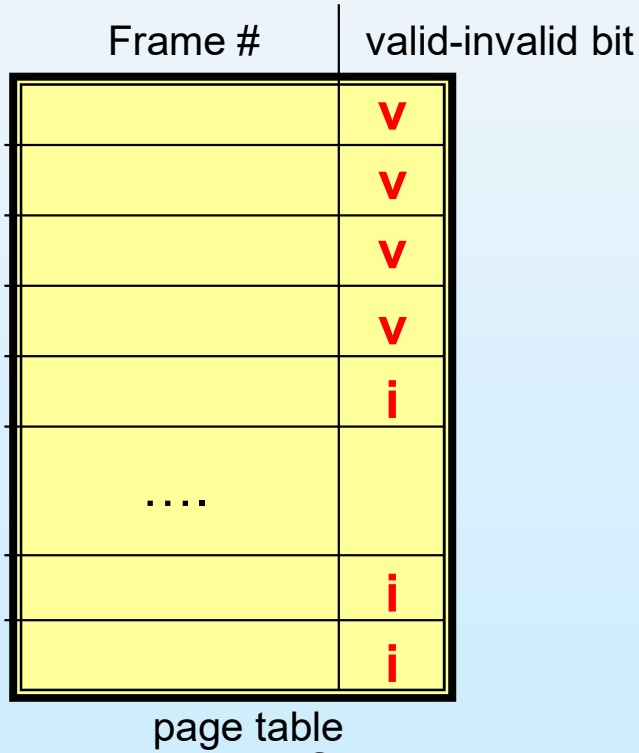
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**

Transfer of a Paged Memory to Contiguous Disk Space



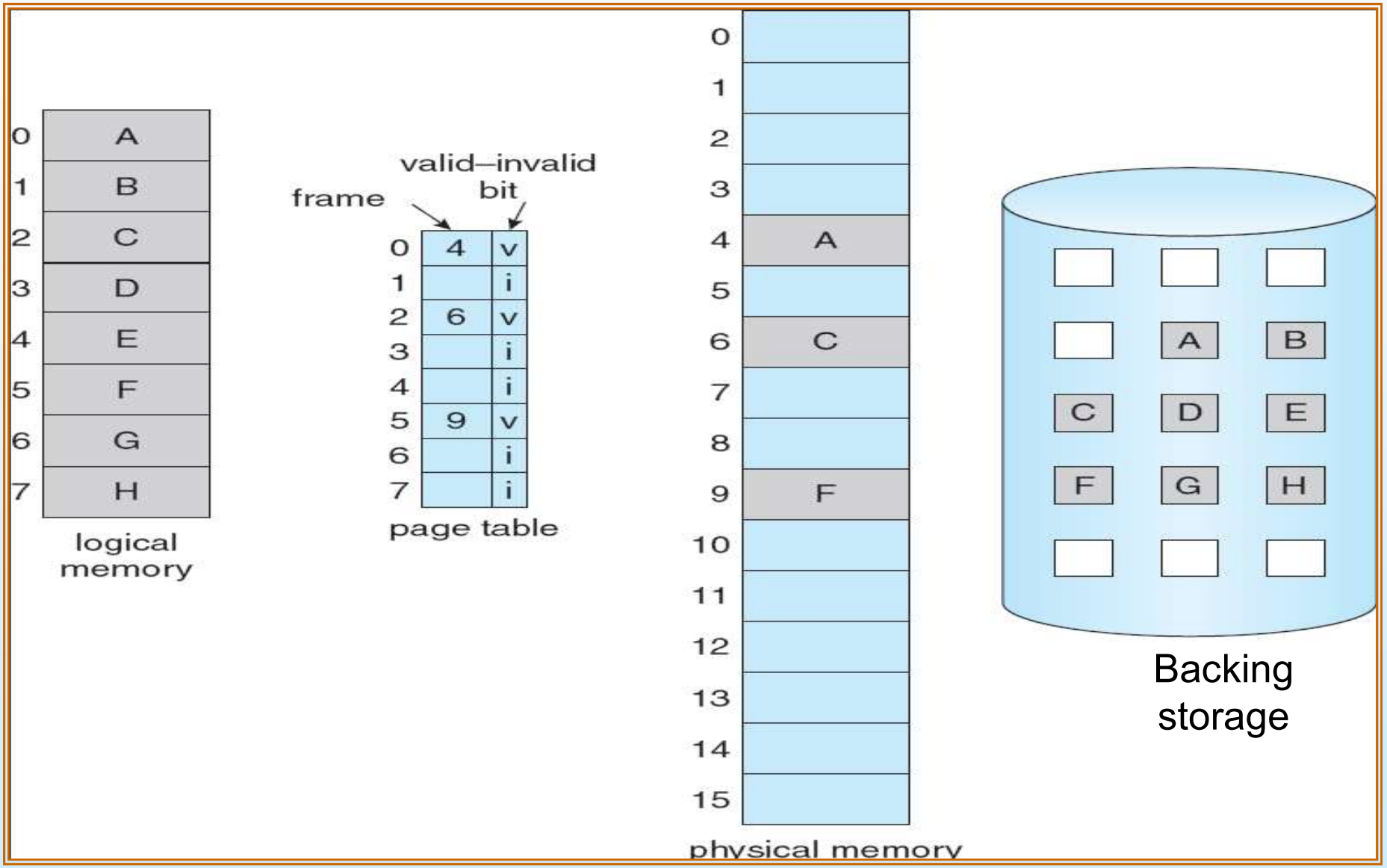
Valid-Invalid Bit

- Valid / invalid bit is used to know which page is legal & needed.
- With each page table entry a valid–invalid bit is associated (**v** ⇒ Legal page & in-memory, **i** ⇒ Illegal page & not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries then it has to updated.
- Example of a page table snapshot:



During address translation, if valid–invalid bit in page table entry is **I** ⇒ page fault

Page Table When Some Pages Are Not in Main Memory

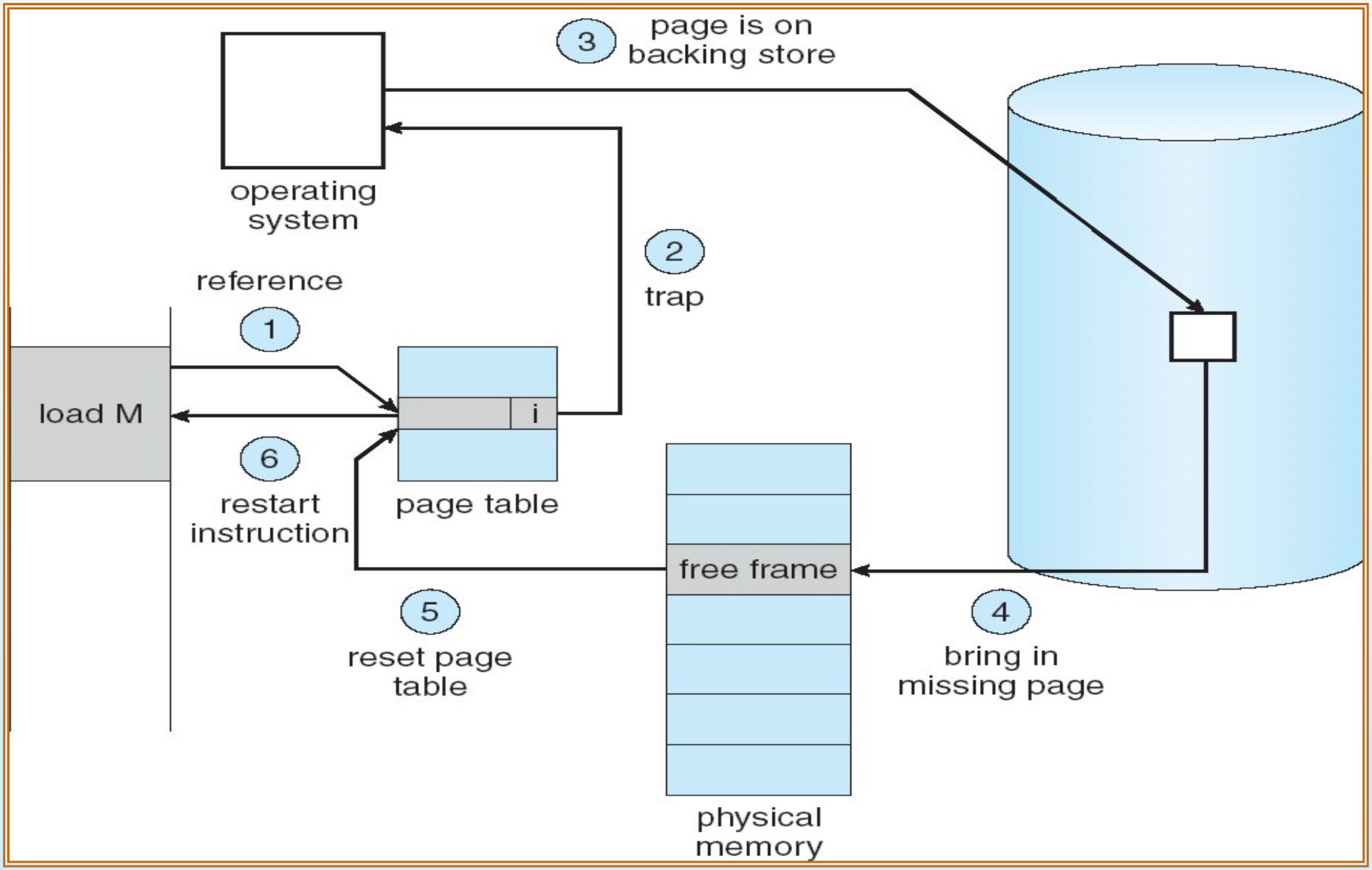


Page Fault

- If the process is available on main memory in terms of pages then a legal page (V)access is possible & execution starts.
- If there is a reference to a page for access & the page is not available on main memory then it is an illegal page (I) which will be trap to operating system: **PAGE FAULT**

1. Operating system looks at another table to decide:
   - Invalid reference $\Rightarrow$ abort
   - Just not in memory
2. Get empty frame on main memory to load the illegal page.
3. Swap page into frame.
4. Reset tables entries [I bit converts to V]
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

Steps in Handling a Page Fault



Page Fault Service Time

1. Trap to the operating system.
2. Save the user register & process state.
3. Determine the interrupt was a page fault.
4. Determine the page location on disk.
5. Issue a read from disk to free frame (device queue, disk latency time & page transfer time)
6. While waiting allocate the CPU to some other user.
7. Interrupt from disk (I/O completion).
8. Save the process state for the other process.
9. Determine that the interrupt was from disk.
10. Correct the page table.
11. Wait for CPU to resume back to the same process.
12. Restore the user register, process state & new page table then resume the interrupted instruction.

Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

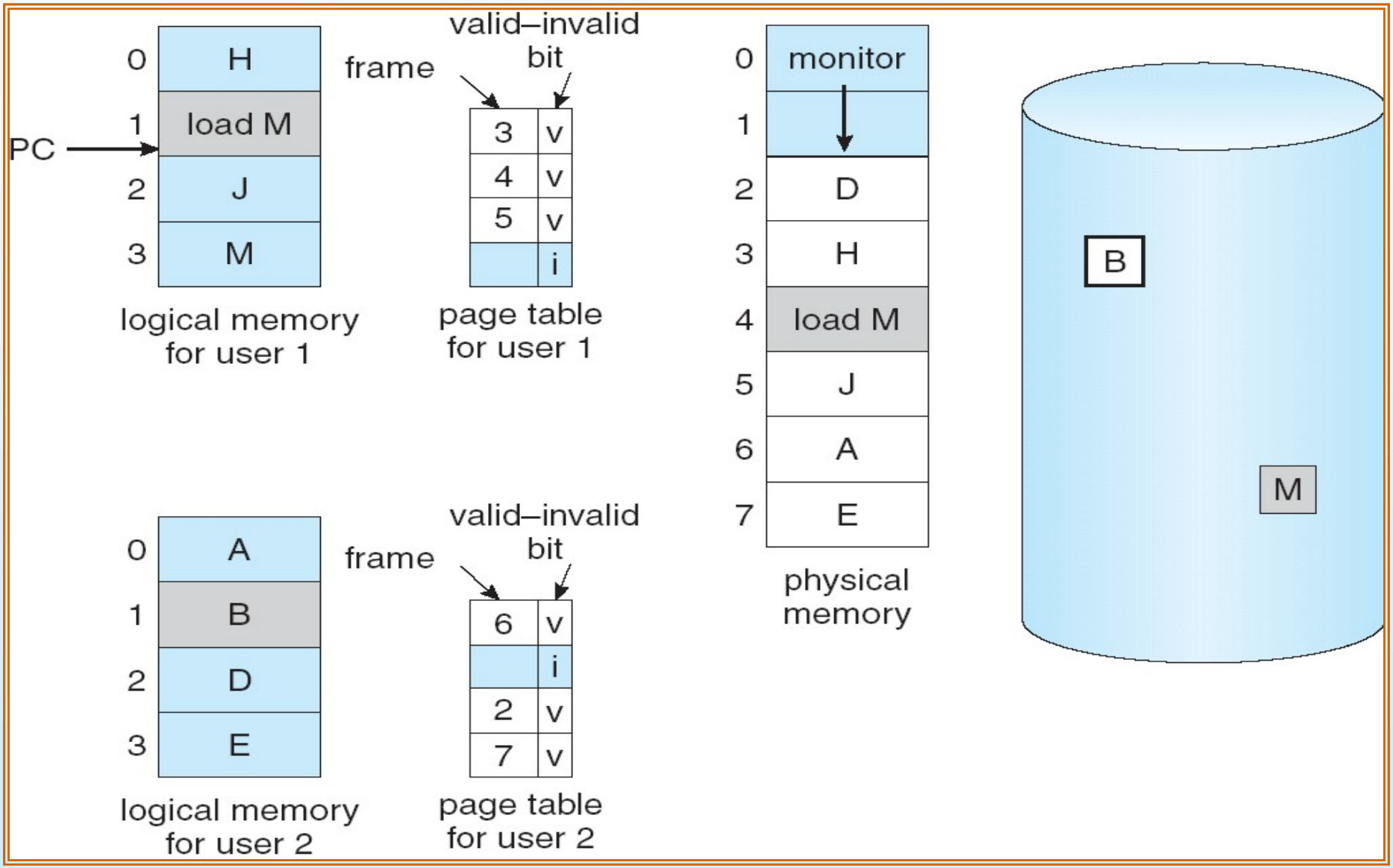
- Effective Access Time (EAT)
  - EAT = (1 - p) x ma + p x page fault service time

EAT = (1 – p) x memory access
+ p x (page fault overhead + swap page out + swap page in + restart overhead)

What happens if there is no free frame?

- When page fault service finds the desired page on disk and wants to transfer it to a free frame on main memory then there may be two possibilities →
  - Free frame found then
  - Free frame not found then
- Process may abort or Process may swap.
- Intended to have dynamic swapping instead of the whole process to swap with another process.
- Page replacement – find some free frame/page in memory, but not really in use, swap it out, so that desired page can swap in.
  - Page replacement algorithm
  - performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

Need For Page Replacement



Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

Basic Page Replacement

1. Find the location of the desired page on 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 page to the disk; change the page table and frame table.
3. Bring the desired page into the (newly) free frame; update the page and frame table.
4. Restart the user process.

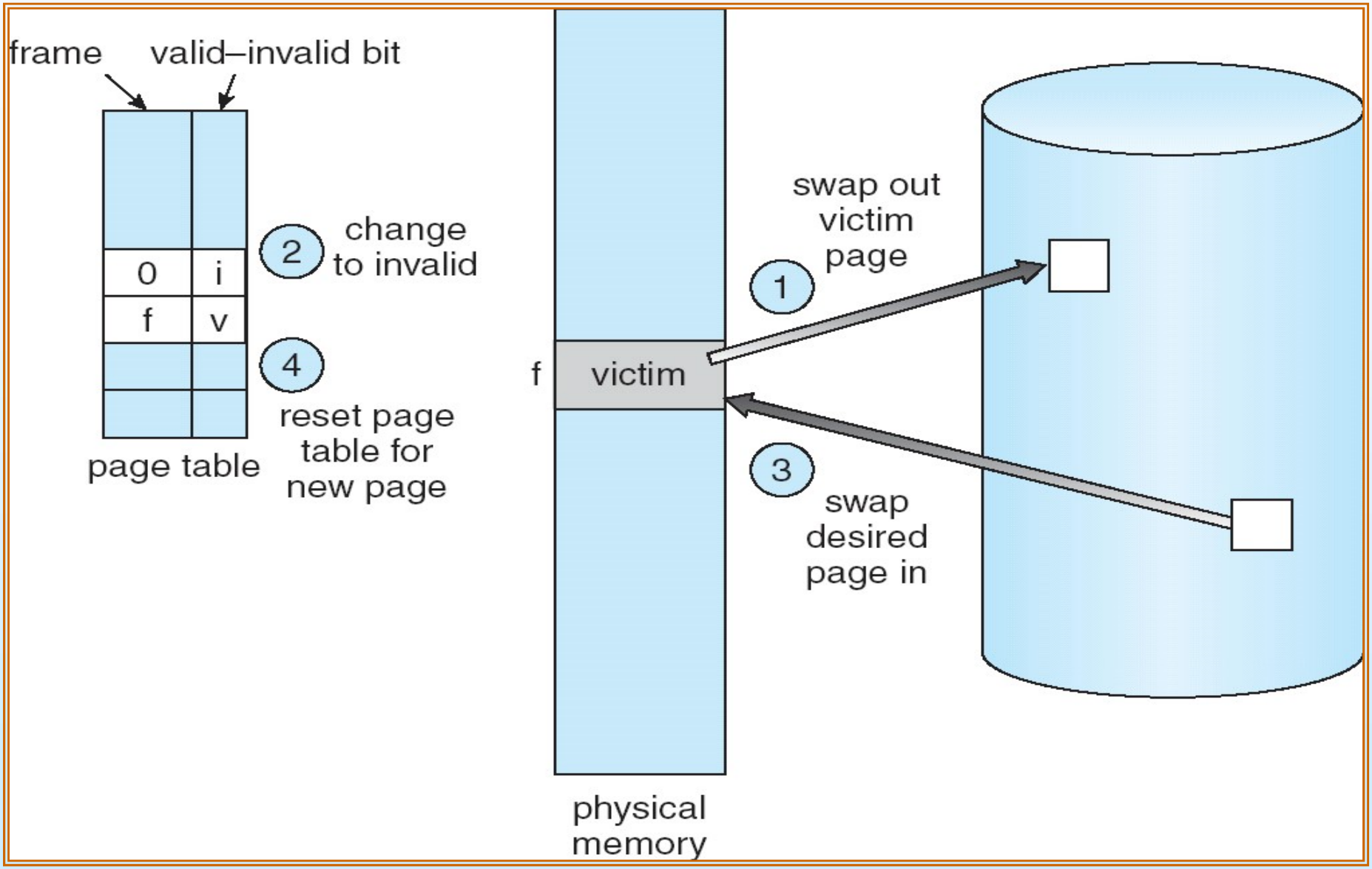
   For each case where no free frames available effectively doubles the page fault service time & increase the EAT.

Solution → Maintain a MODIFY BIT.

Modify Bit

- To reduce the page fault service time use a modify bit which is associated with each page.
- If modify bit is set to 1 then
  - Page has been modified, write the page to disk, so that it can be modified over main memory.
- If modify bit is set to 0 then
  - Page has not been modified, since it was read into main memory.
- Victim Page Selection
  - Leads to lowest page fault
  - Reference string
  - Free frame increases ➔ the page fault decrease.

Page Replacement



Page Replacement Algorithms

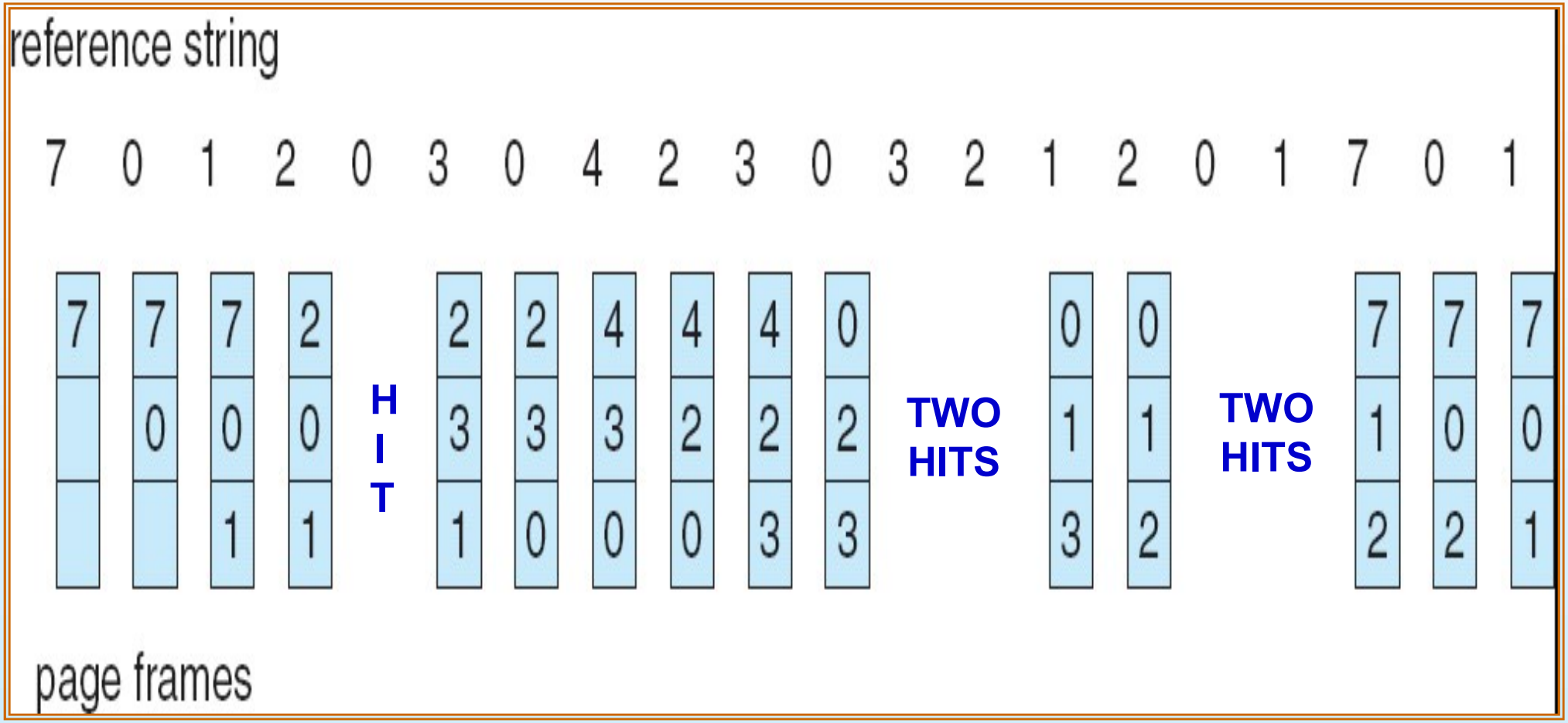
- Want lowest page-fault rate

- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string

First-In-First-Out (FIFO) Algorithm

- Replacement is depends upon the arrival time of a page to memory.
- A page is replaced when it is oldest (in the ascending order of page arrival time to memory).
- As it is a FIFO queue no need to record the arrival time of a page & the page at the head of the queue is replaced.
- Performance is not good always
- When a active page is replaced to bring a new page, then a page fault occurs immediately to retrieve the active page.
- To get the active page back some other page has to be replaced. Hence the page fault increases.

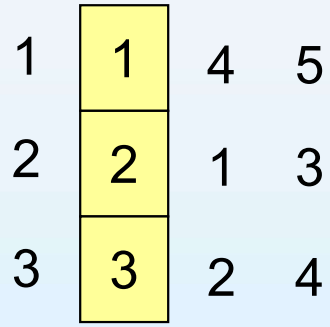
FIFO Page Replacement



15 PAGE FAULTS

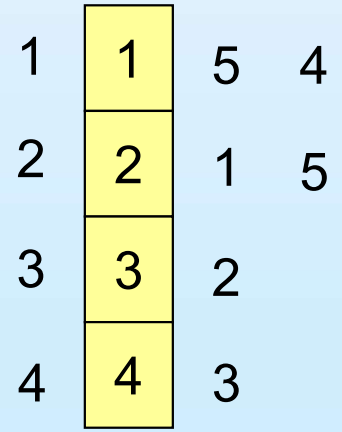
Problem with FIFO Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

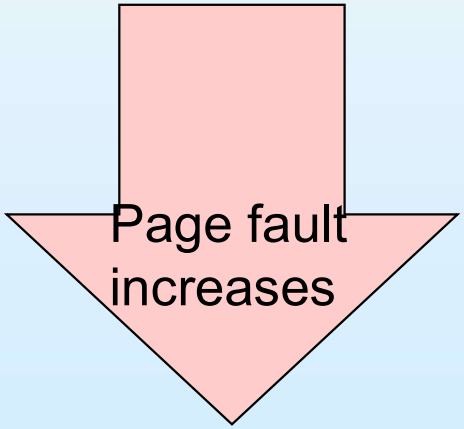


9 page faults

- 4 frames



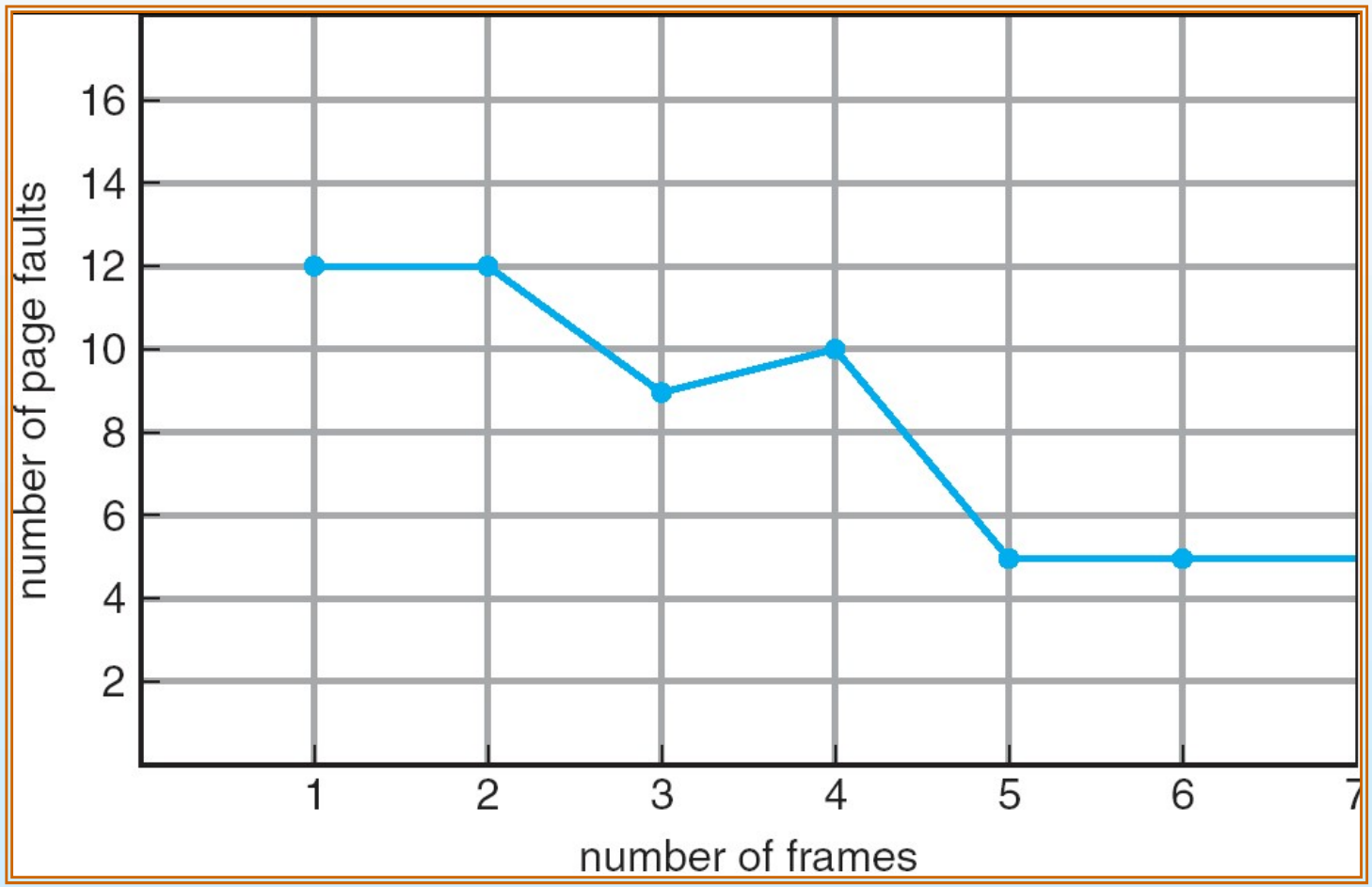
10 page faults



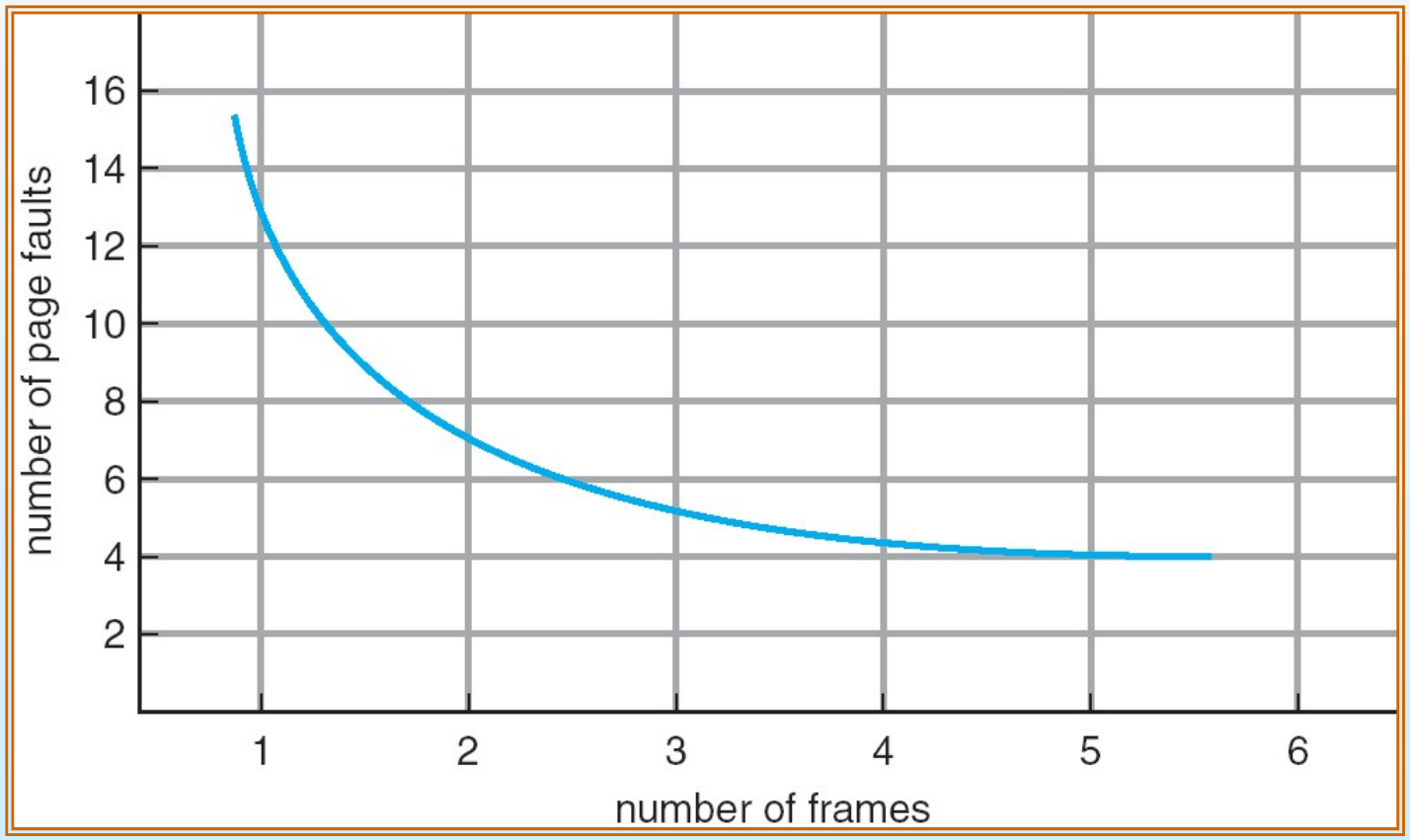
UNEXPECTED

- Belady's Anomaly:→ more frames ⇒ more page faults

FIFO Illustrating Belady's Anomaly



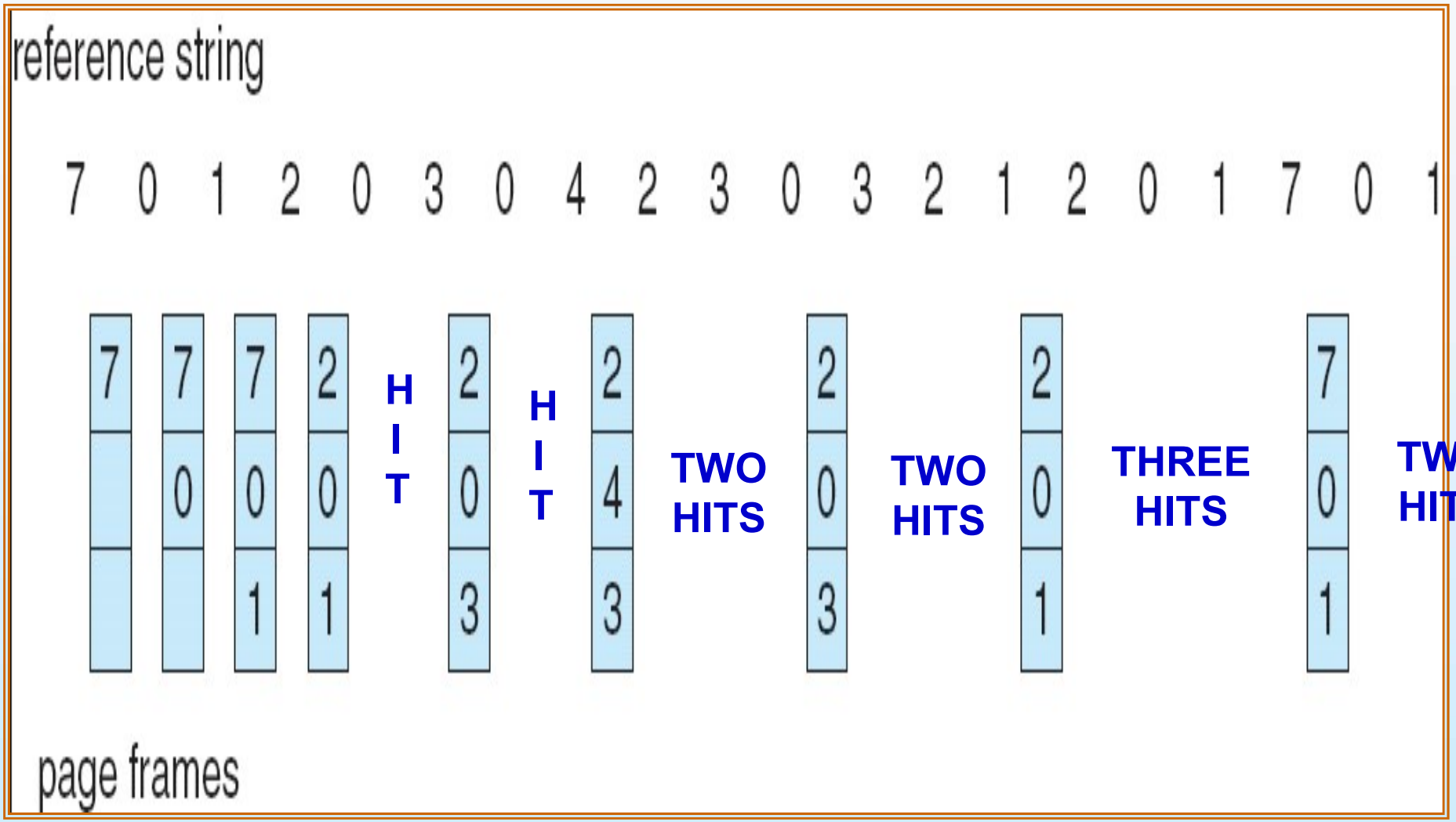
Graph of Page Faults Versus The Number of Frames



Optimal Algorithm

- To recover from belady's anomaly problem :→ **Use Optimal page replacement algorithm**
- Replace the page that will not be used for longest period of time.
- This guarantees lowest possible page fault rate for a fixed number of frames.
- Example :→
  - First we found 3 page faults to fill the frames.
  - Then replace page 7 with page 2 because it will not needed upto the 18^{th} place in reference string.
  - Finally there are 09 page faults.
  - Hence it is better than FIFO algorithm (15 page Faults).

Optimal Page Replacement

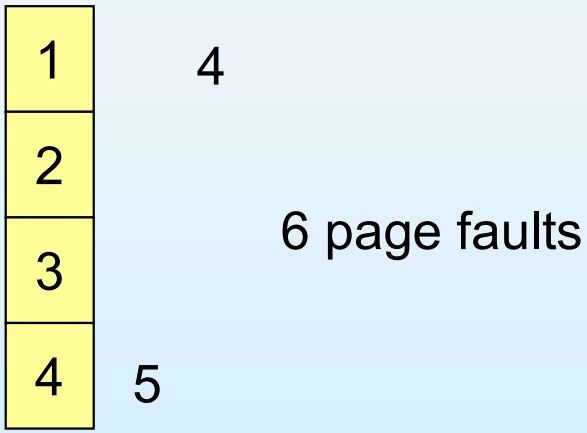


09 PAGE FAULTS

Difficulty with Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example

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

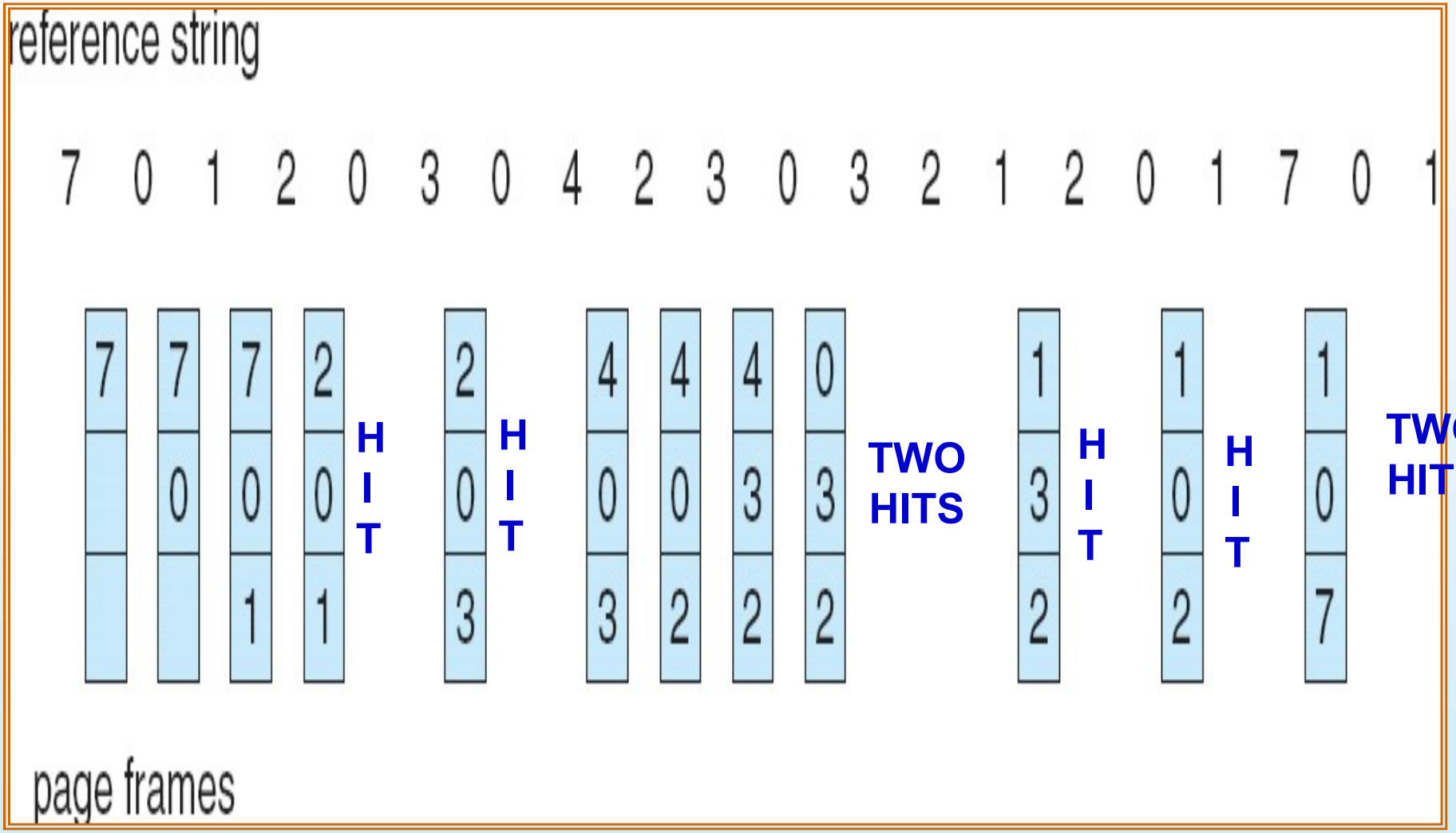


- Used for measuring how well your algorithm performs.
- **It always needs future knowledge of reference string.**

Least Recently Used (LRU) Algorithm

- LRU algorithm lies between FIFO & Optimal algorithm (in terms of page faults).
- FIFO :→ time when page brought into memory.
- OPTIMAL :→ time when a page will used.
- LRU :→ Use the recent past as an approximation of near future (so it cant be replaced), then we will replace that page which has not been used for longest period of time. Hence it is least recently used algorithm.
- Example :→
  - Up to 5th page fault it is same as optimal algorithm.
  - When page 4 occur LRU chose page 2 for replacement.
  - Here we find only 12 page faults.

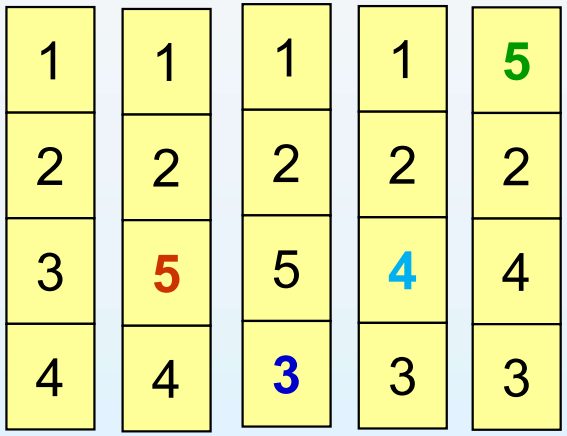
LRU Page Replacement



12 PAGE FAULTS

Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**



- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

End of Chapter 9