



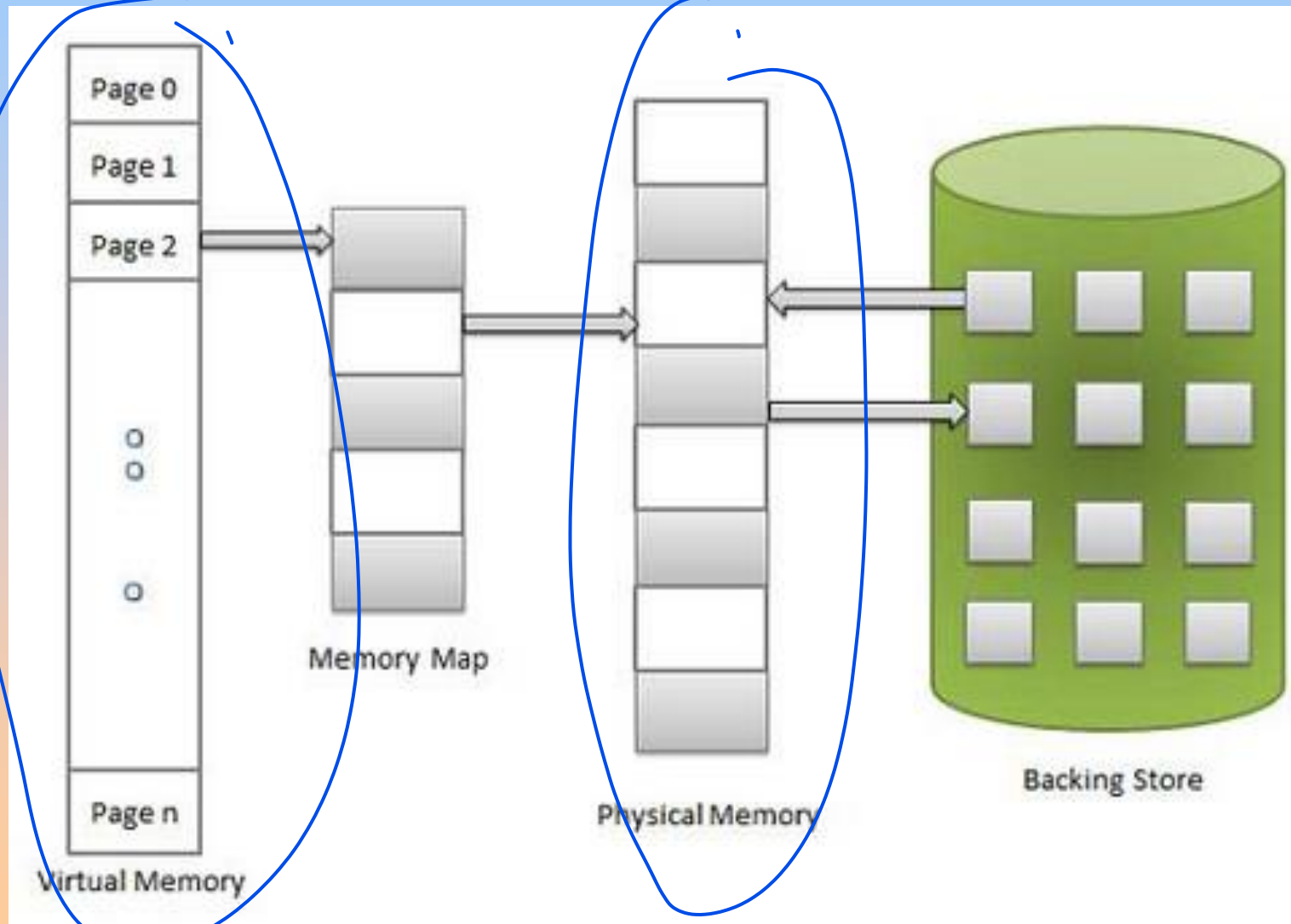
Virtual Memory

- ❑ Virtual memory is a technique that allows the execution of processes that may not be completely in memory
- ❑ Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory.
- ❑ Virtual memory is the separation of user logical memory from physical memory
- ❑ Helps to provide an extremely large virtual memory to the programmers when only a smaller physical memory is available
- ❑ **Benefits of having Virtual Memory :**
 - ❑ Large programs can be written, as virtual space available is huge compared to physical memory.
 - ❑ Less I/O required, leads to faster and easy swapping of processes.
 - ❑ More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.
- ❑ Virtual memory is commonly implemented by **Demand Paging** and **Segmentation** system





Diagram showing virtual memory that is larger than physical memory





SO 2: Exemplify the steps in handling a page fault





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 \Rightarrow reference to it
 - invalid reference \Rightarrow abort
 - not-in-memory \Rightarrow bring to memory



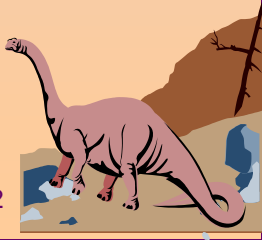


Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated
(1 \Rightarrow in-memory, 0 \Rightarrow not-in-memory)
- Initially valid–invalid but is set to 0 on all entries.
- Example of a page table snapshot.

Frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

page table





Demand Paging

- ❑ The basic idea behind *demand paging* is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them. (on demand.)
- ❑ A swapper manipulates entire processes whereas a *pager* is concerned with individual pages of a process
- ❑ *pager* is a more accurate than swapper in demand paging





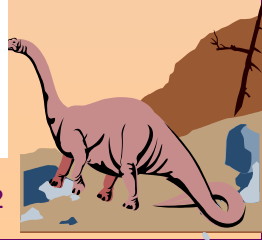
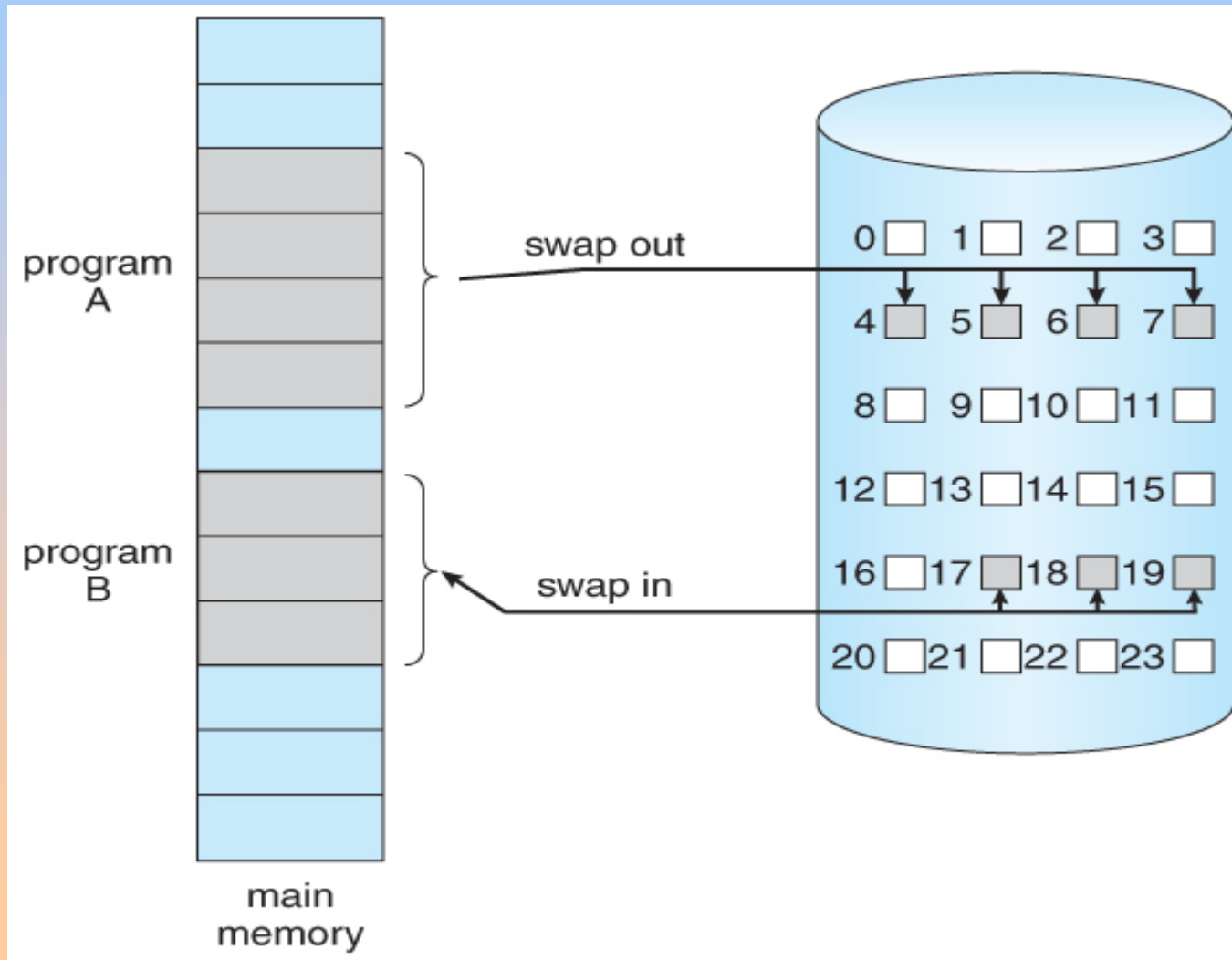
Basic Concepts

- The basic idea behind paging is that when a process is swapped in, the pager only loads into memory those pages that it expects the process to need (right away.)
- Pages that are not loaded into memory are marked as invalid in the page table, using the invalid bit. (The rest of the page table entry may either be blank or contain information about where to find the swapped-out page on the hard drive.)
- If the process only ever accesses pages that are loaded in memory (*memory resident* pages), then the process runs exactly as if all the pages were loaded in to memory.





Transfer of a paged memory to contiguous disk space





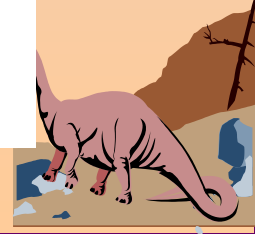
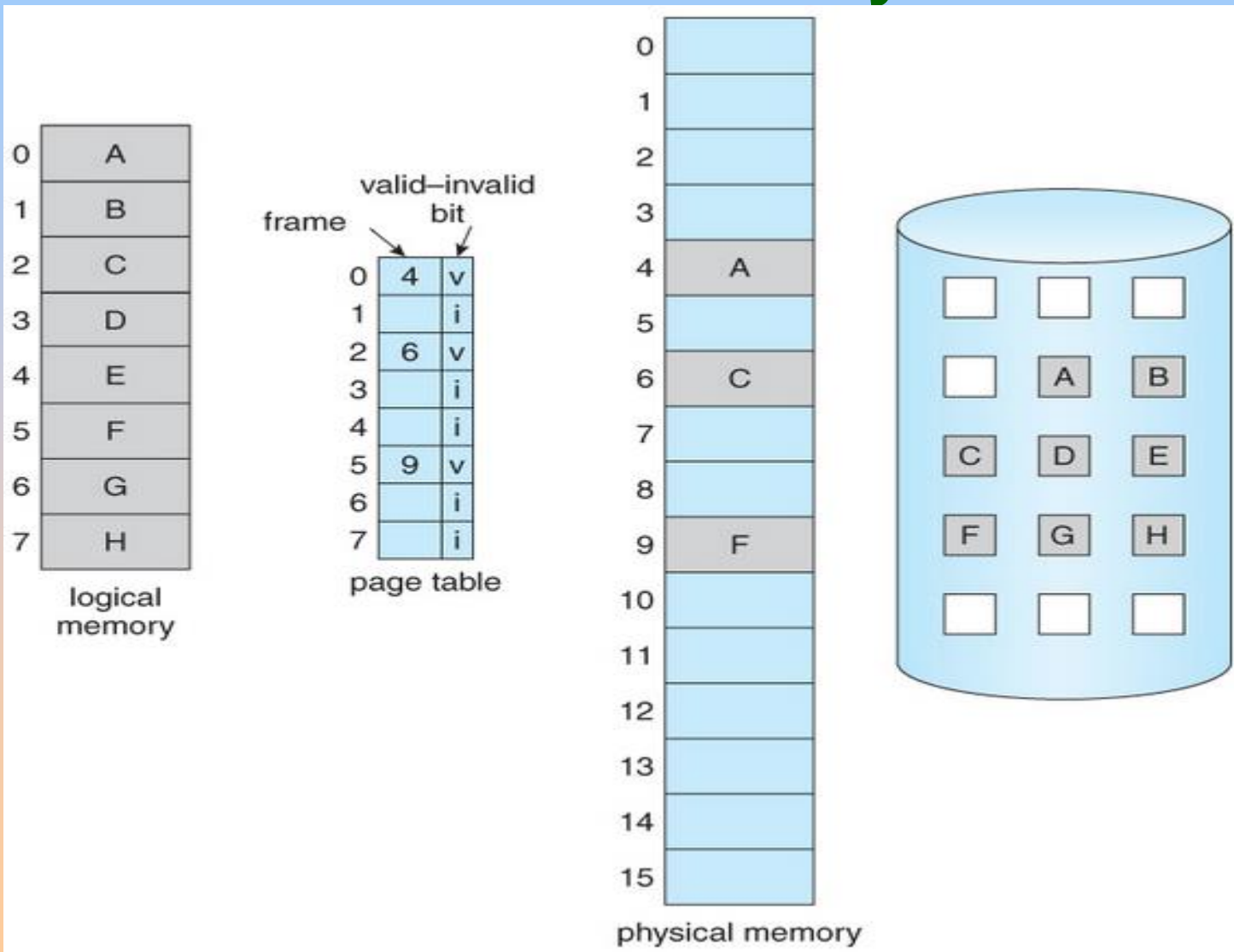
Basic Concepts

- On the other hand, if a page is needed that was not originally loaded up, then a **page fault trap** is generated, which must be handled in a series of steps:
 - The memory address requested is first checked, to make sure it was a valid memory request.
 - If the reference was invalid, the process is terminated. Otherwise, the page must be paged in.
 - A free frame is located, possibly from a free-frame list.
 - A disk operation is scheduled to bring in the necessary page from disk. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)





Page table when some pages are not in main memory



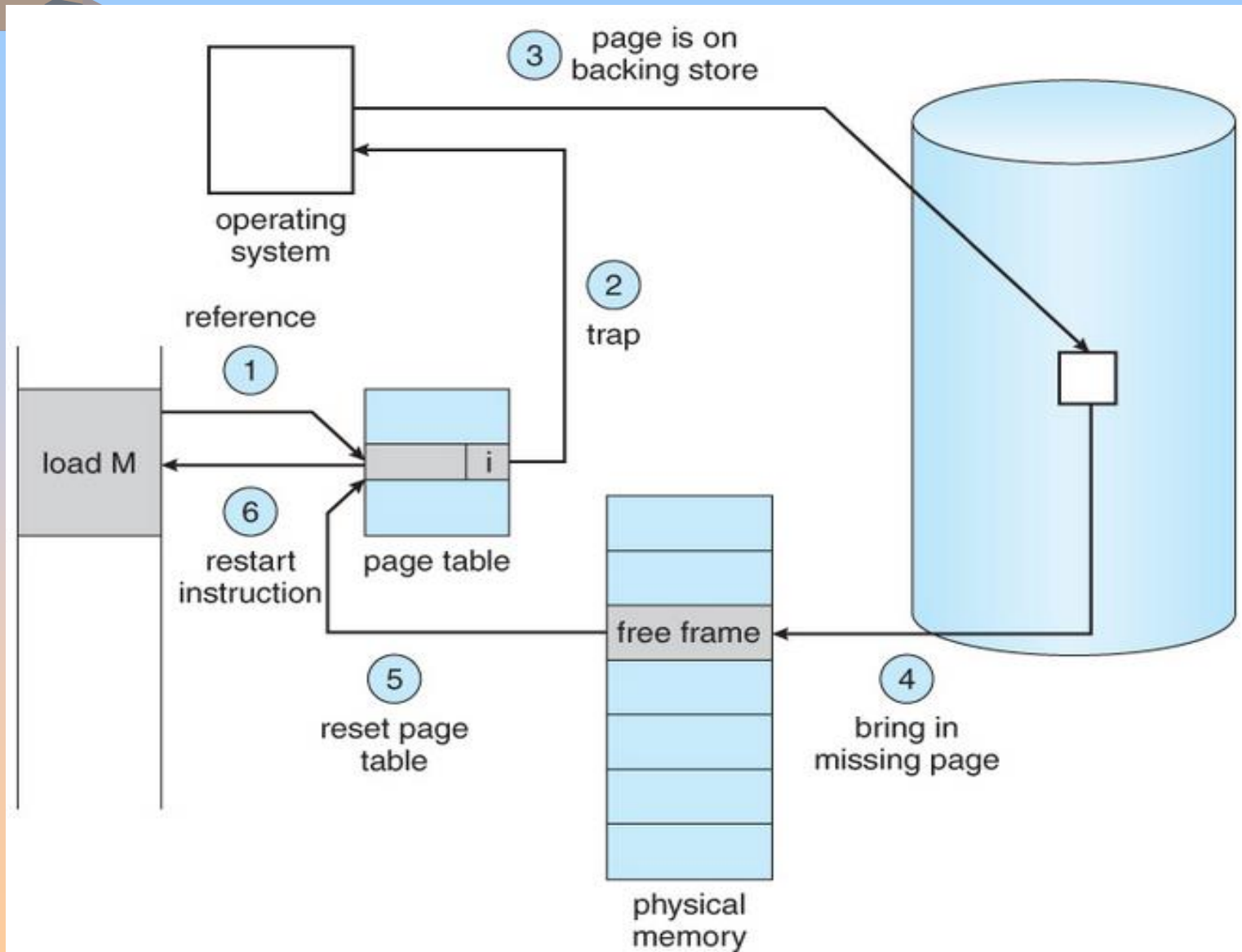


Basic Concepts

- When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.
- The instruction that caused the page fault must now be restarted from the beginning, (as soon as this process gets another turn on the CPU.)



Steps in handling a page fault





Page Fault

- ❑ A page fault occurs when a program attempts to access a block of memory that is not stored in the physical memory, or RAM.
- ❑ The fault notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM.

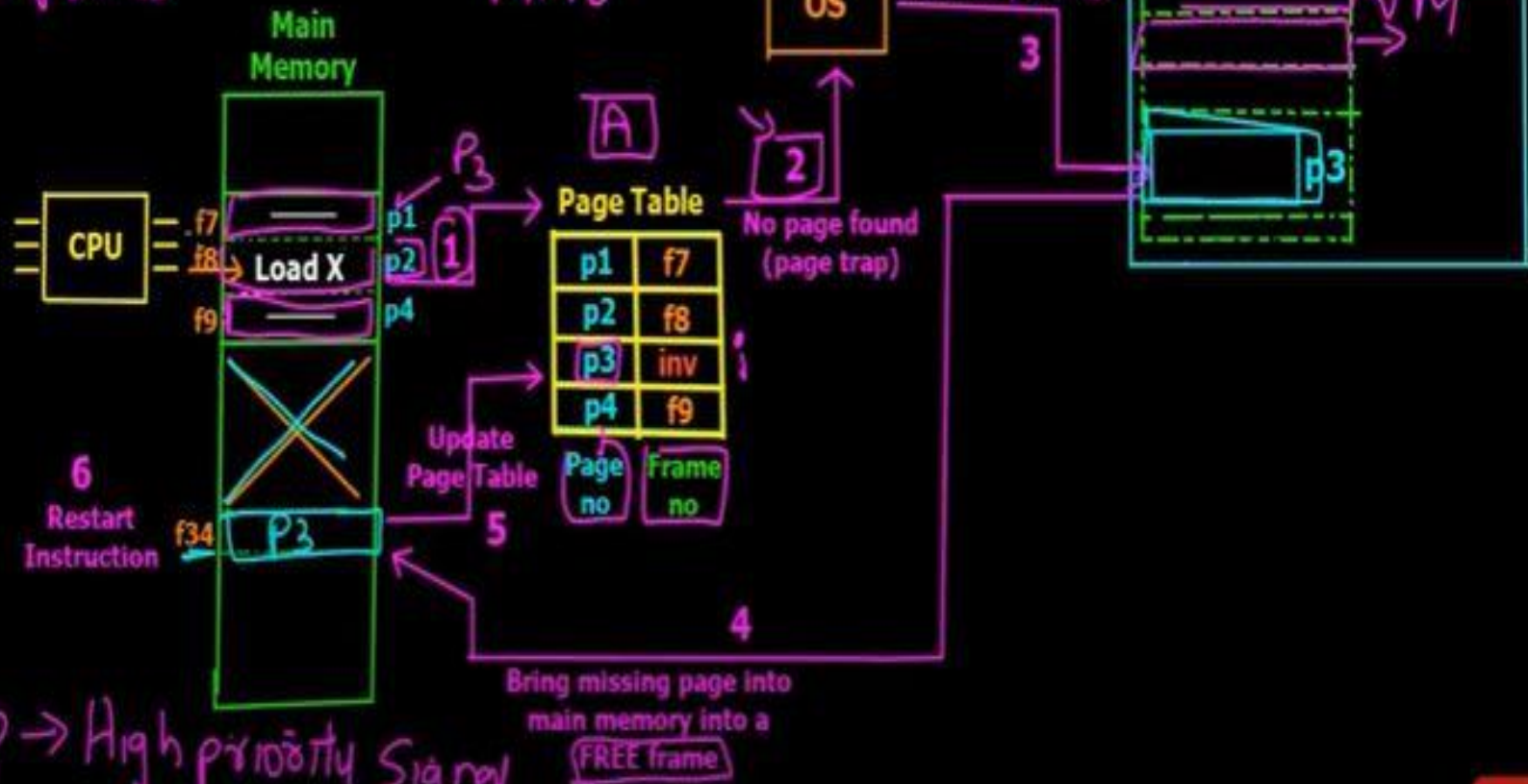


Page Faults & Page Fault Handling | Thrashing

Consider a Process A divided into 4 pages - p1, p2, p3, p4


Page = frames

MMU



Trap → High priority Signal in OS



- 
1. $0 \leq \text{Page Fault Probability}(P) \leq 1$
 2. Effective Access Time (EAT) -

$$\text{EAT} = (1-P) \times \text{Memory Access} + P \times \text{page fault overload}$$

Q) Memory access time = 100ms
Page fault service overhead = 10ms
1 access causes page fault out of 10
Calculate EAT ?





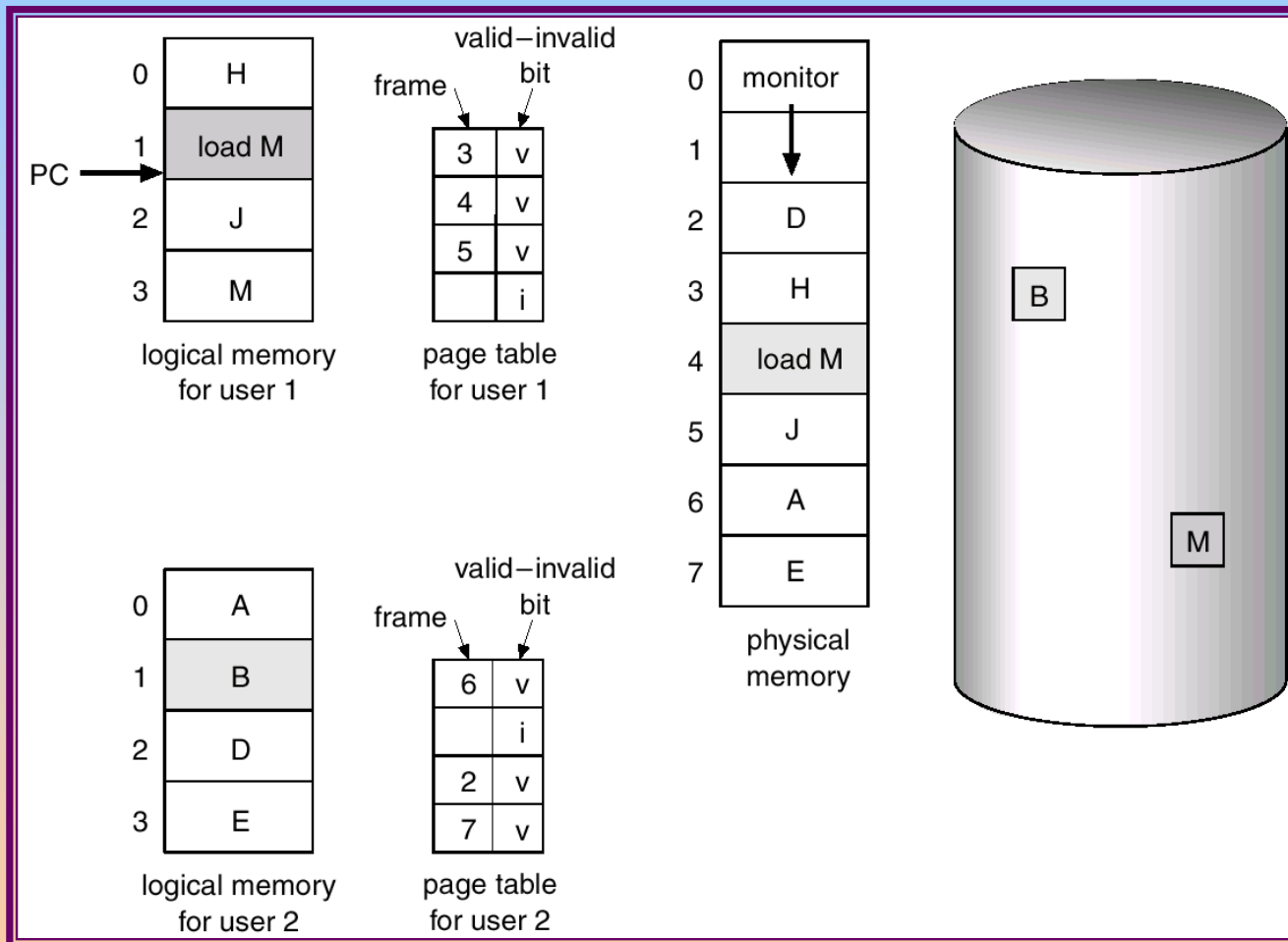
Q) Memory access time = 100ms
→ Page fault service overhead = 10ms
→ 1 access causes page fault out of 10
Calculate EAT ?

$$P = \frac{1}{10} = 0.1$$

$$\begin{aligned} \text{EAT} &= (1 - 0.1) \times 100 + 0.1 \times 10 \\ &= 0.9 \times 100 + 1 = \boxed{91 \text{ms}} \end{aligned}$$



Need For Page Replacement



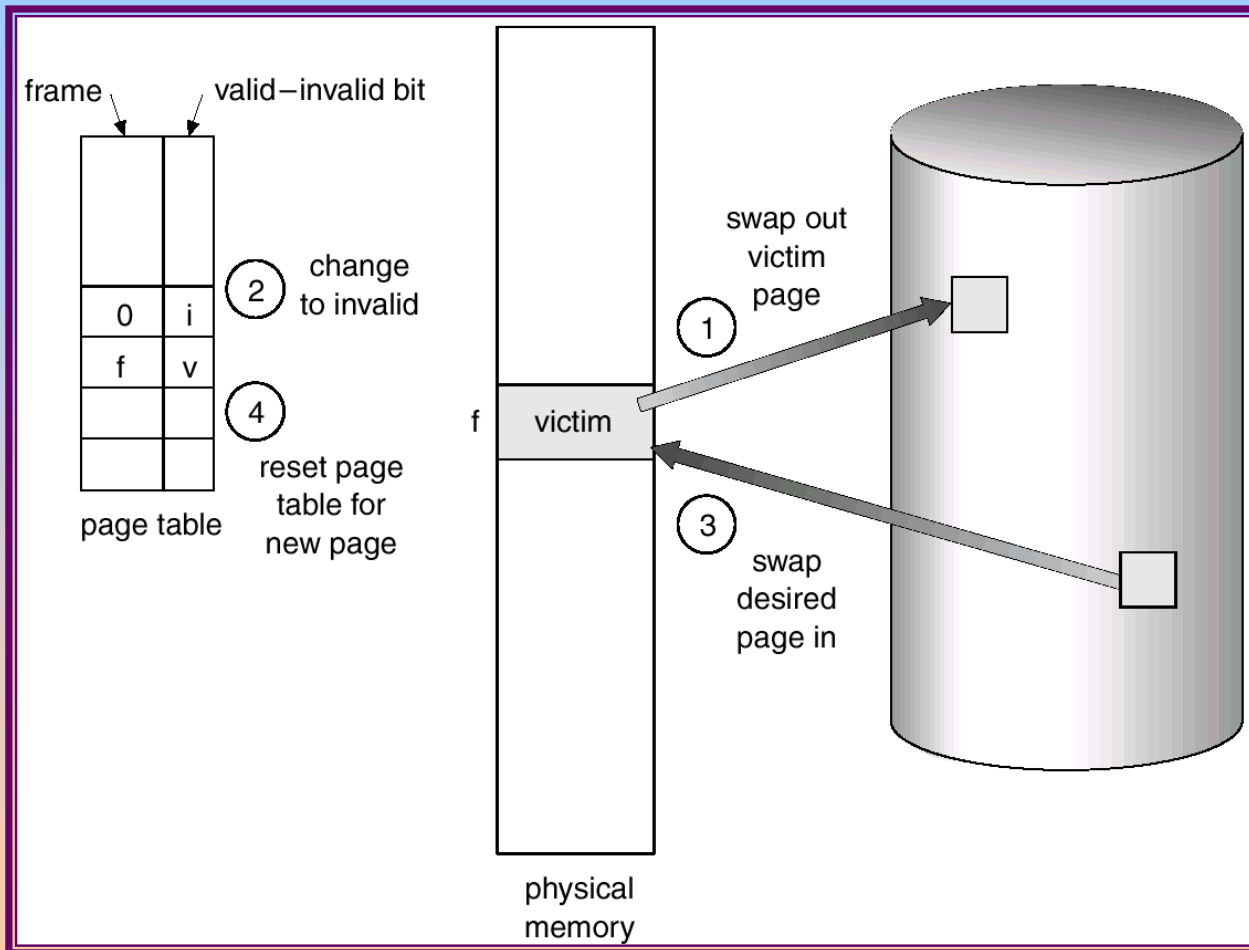


Basic Page Replacement

1. Find the location of the desired page on disk.
2. Find a free frame:
 - If there is a free frame, use it.
 - If there is no free frame, use a page replacement algorithm to select a *victim* frame.
3. Read the desired page into the (newly) free frame.
Update the page and frame tables.
4. Restart the process.



Page Replacement





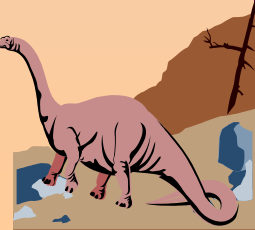
SO 2: Demonstrate the page replacement algorithms





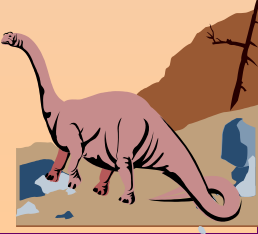
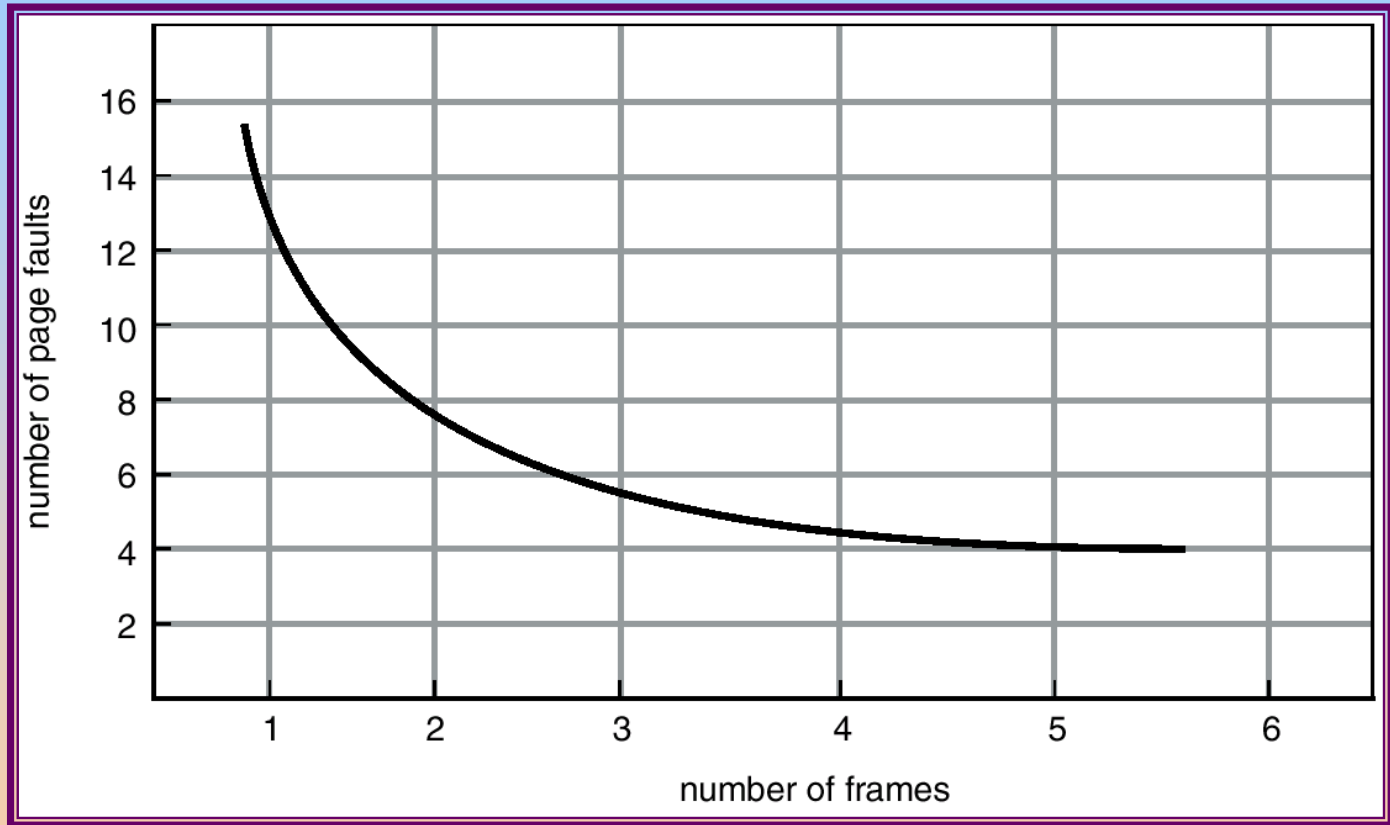
Page Replacement Algorithms

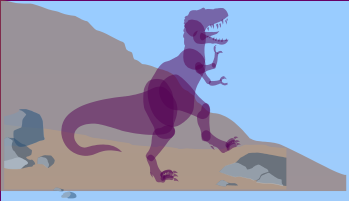
- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In all our examples, the reference string is
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.





Graph of Page Faults Versus The Number of Frames





First-In-First-Out (FIFO) Algorithm

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

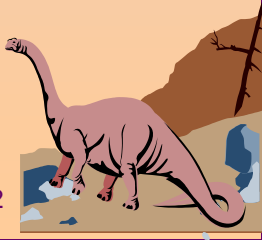
3 frames (3 pages can be in memory at a time per process)

4 frames

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

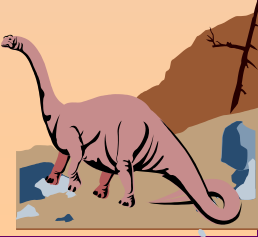
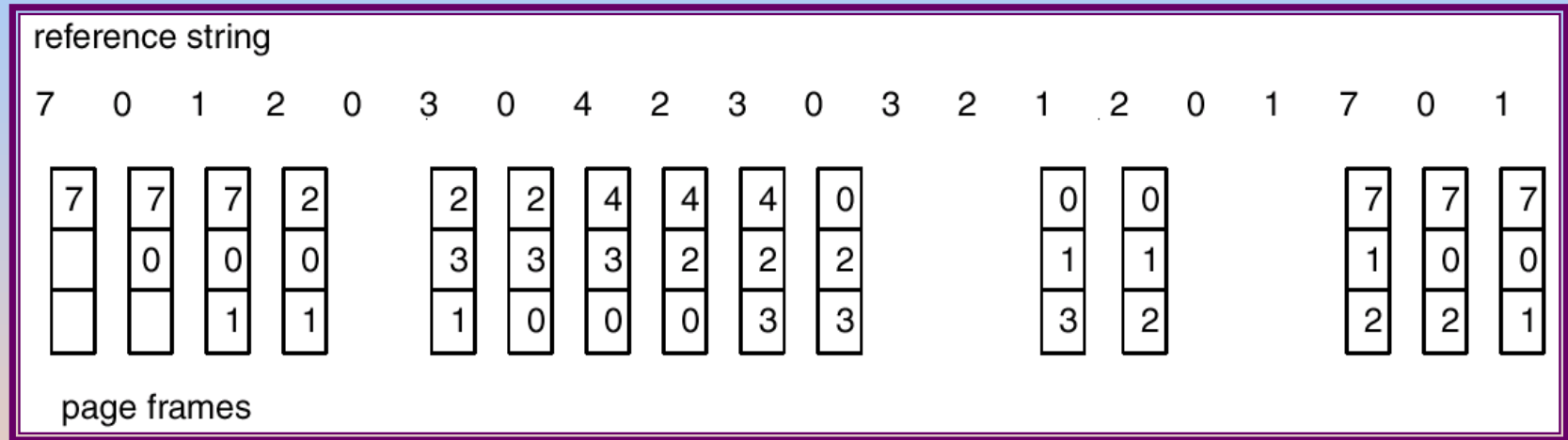
1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

FIFO Replacement – Belady's Anomaly
more frames \Rightarrow less page faults



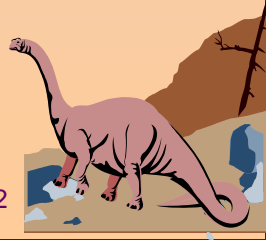


FIFO Page Replacement





FIFO Illustrating Belady's Anamoly





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

1	4
2	
3	
4	5

6 page faults

- ❑ How do you know this?
- ❑ Used for measuring how well your algorithm performs.





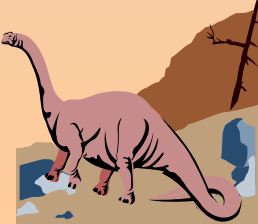
Optimal Page Replacement

reference string

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

page frames





Least Recently Used (LRU) Algorithm

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

1	5
2	
3	5 4
4	3

- 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.





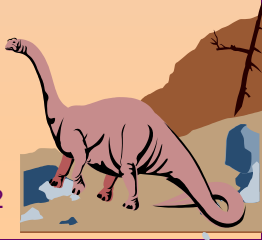
LRU Page Replacement

reference string

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

page frames





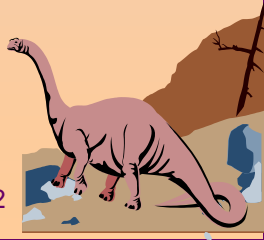
□ Reference strings:

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

Frame size : 4

2, 3, 4, 8, 7, 6, 8, 4, 4, 5, 6, 7, 8, 4, 4, 3, 3, 7, 8, 9

Frame size : 3





SUMMARY





Stimulating Question

- How to avoid external fragmentation in segmentation with paging? Justify your answer

