# CMPS 455
# Operating Systems

# Lecture: Virtual Memory-I

Dr. Ashok Kumar

*Adapted from:

Operating Systems Concepts, by A. Silberschatz, P. Galvin & G. Gagne

Prof. John Kubiatowicz, UC Berkeley

Nachos, UC Berkeley and Univ. of Washington

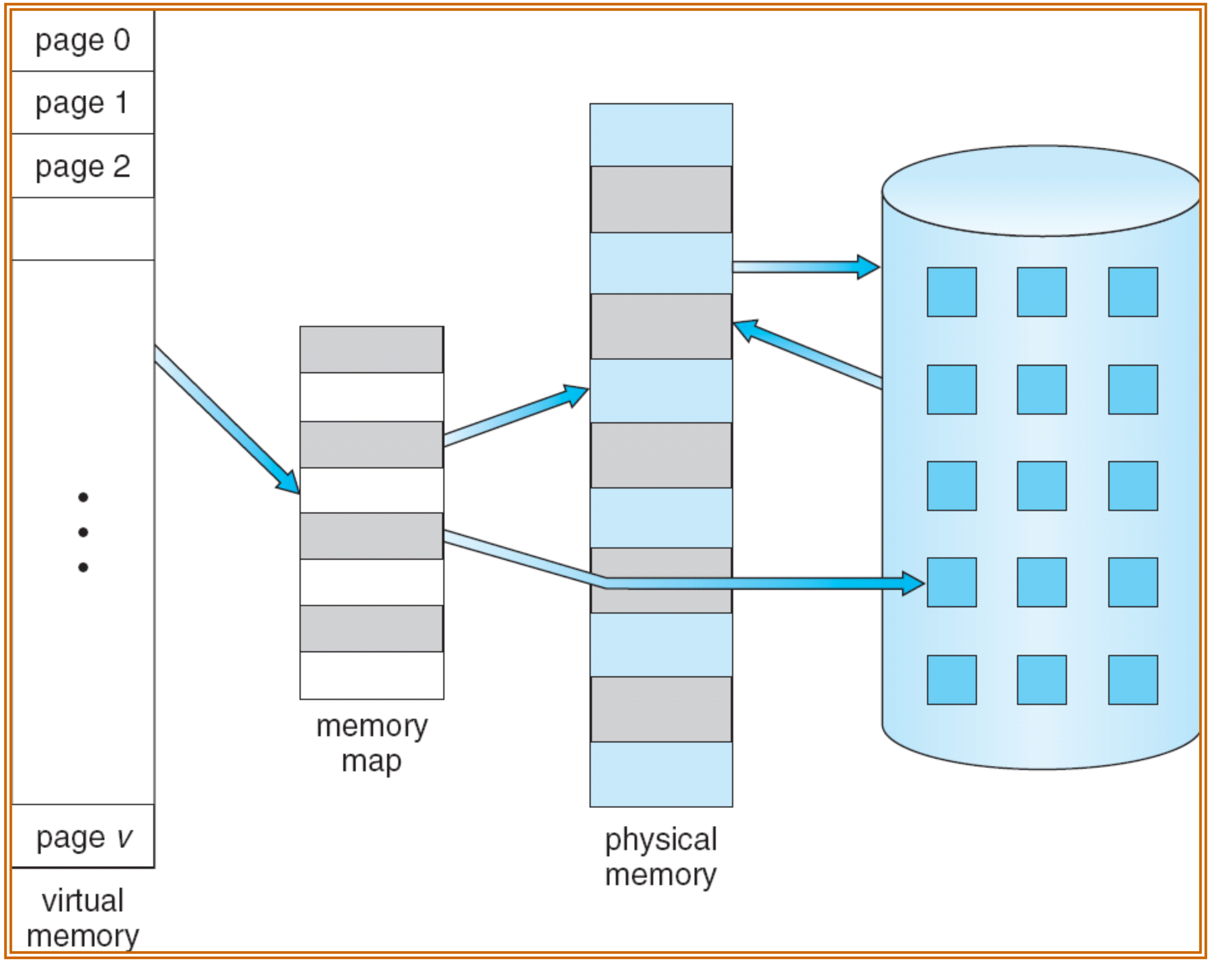
# Chapter 9: Virtual Memory

- Background
- Demand Paging
- Copy-on-Write
- Page Replacement

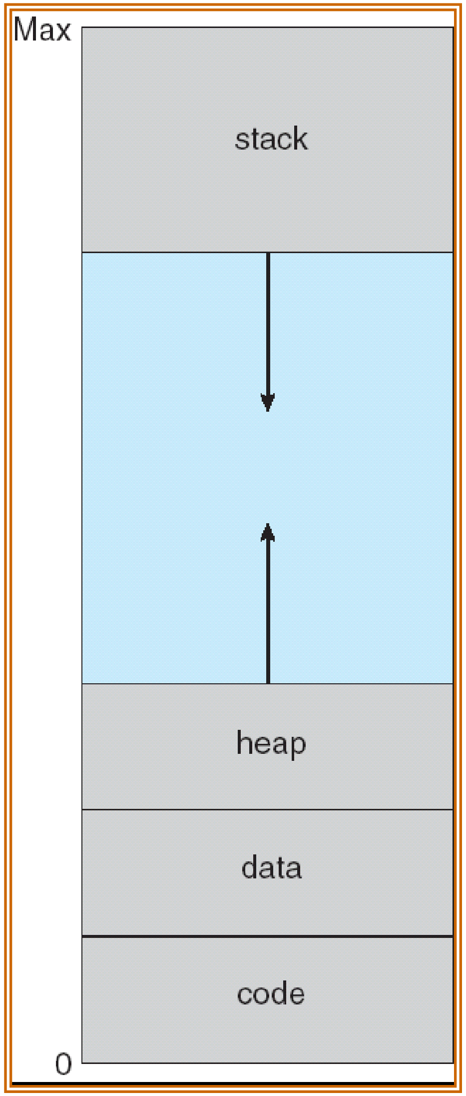
# Background

- **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

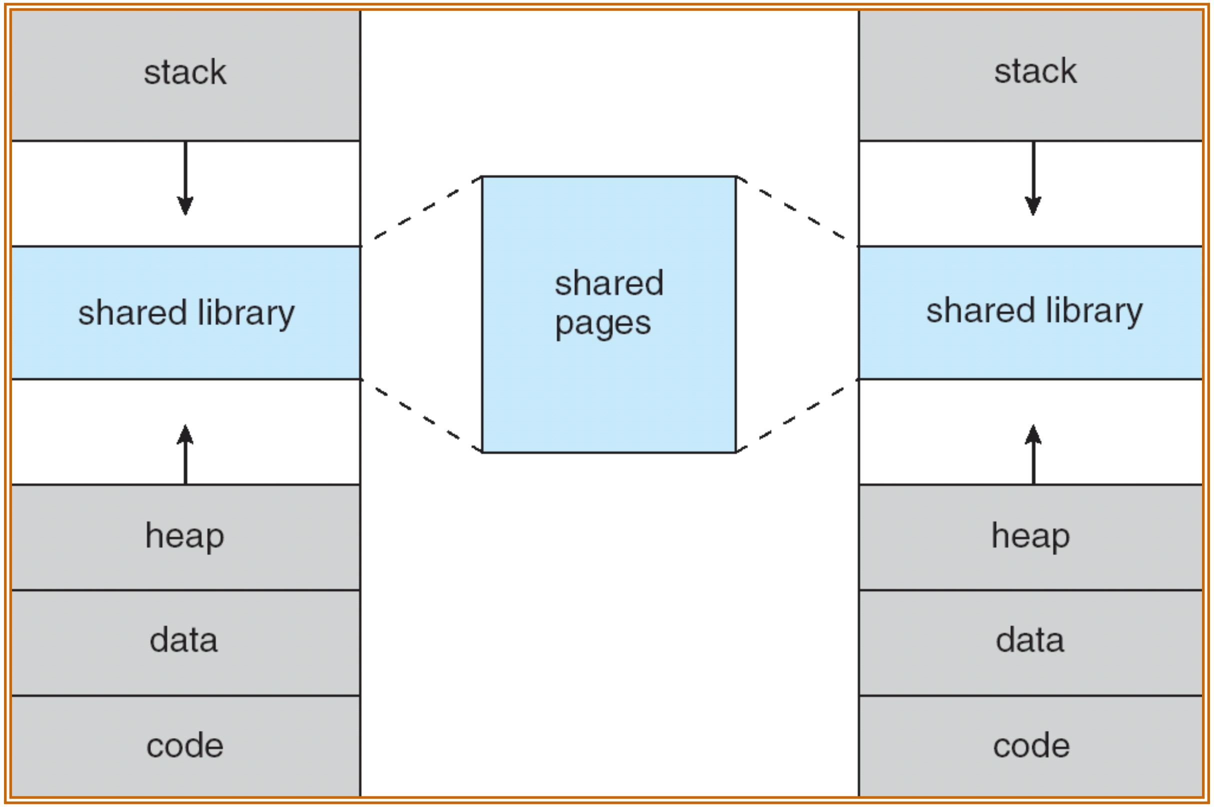
# Virtual Memory That is Larger Than Physical Memory



# Virtual-address Space



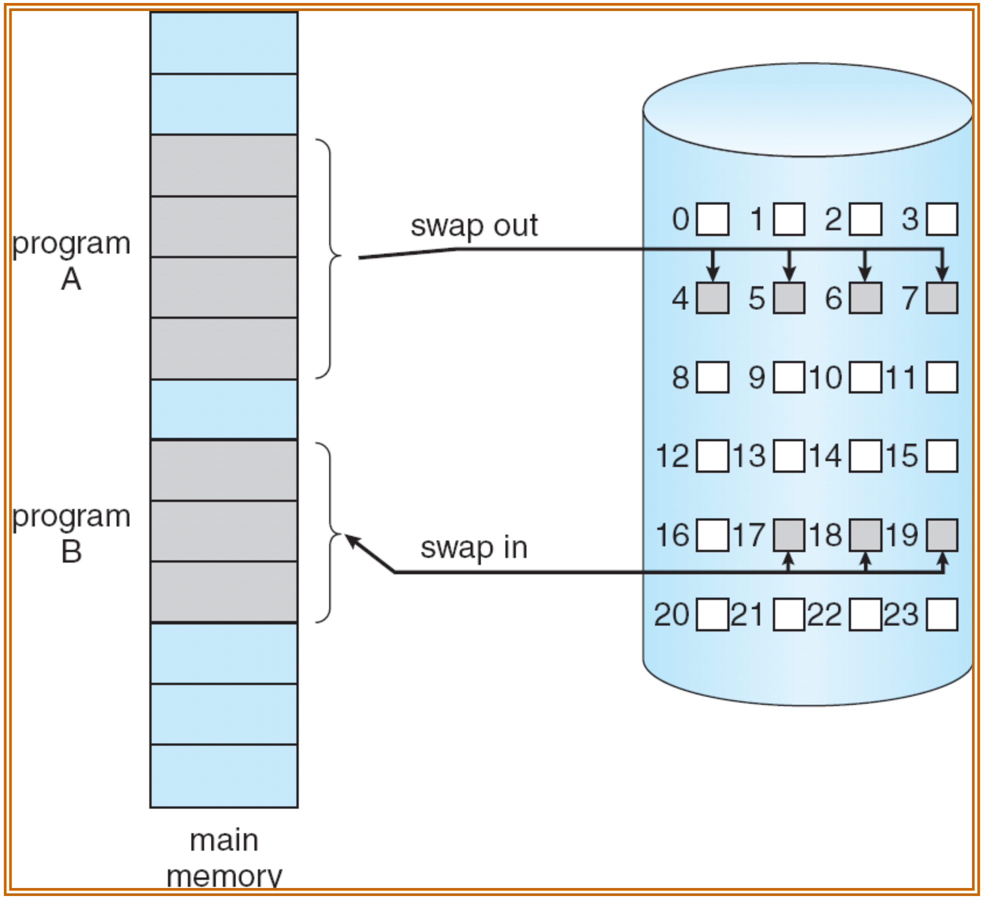
# Shared Library Using Virtual Memory



# 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 $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ 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



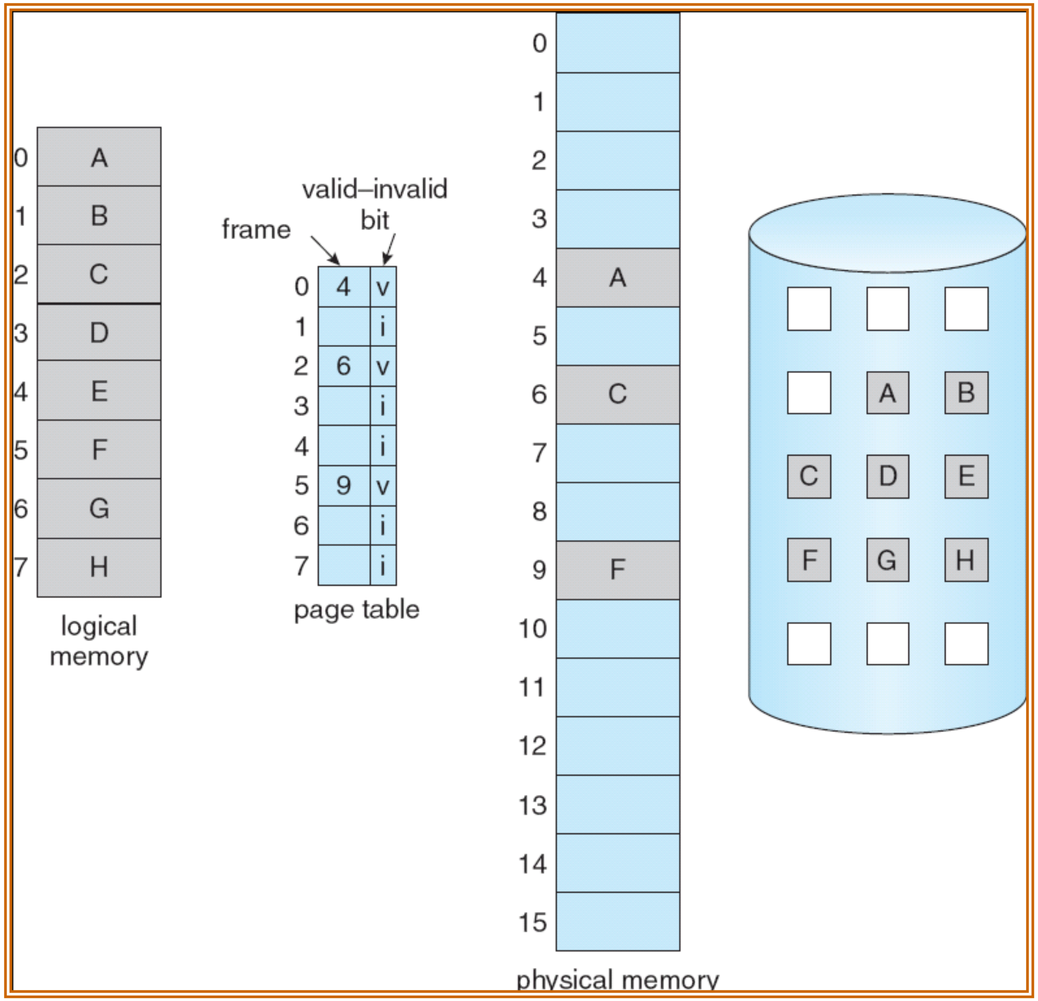
# Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (**v** $\Rightarrow$ in-memory, **i** $\Rightarrow$ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:

| Frame # | valid-invalid bit |
|---|---|
| | v |
| | v |
| | v |
| | v |
| | i |
| …. | |
| | i |
| | i |

page table

# Page Table When Some Pages Are Not in Main Memory



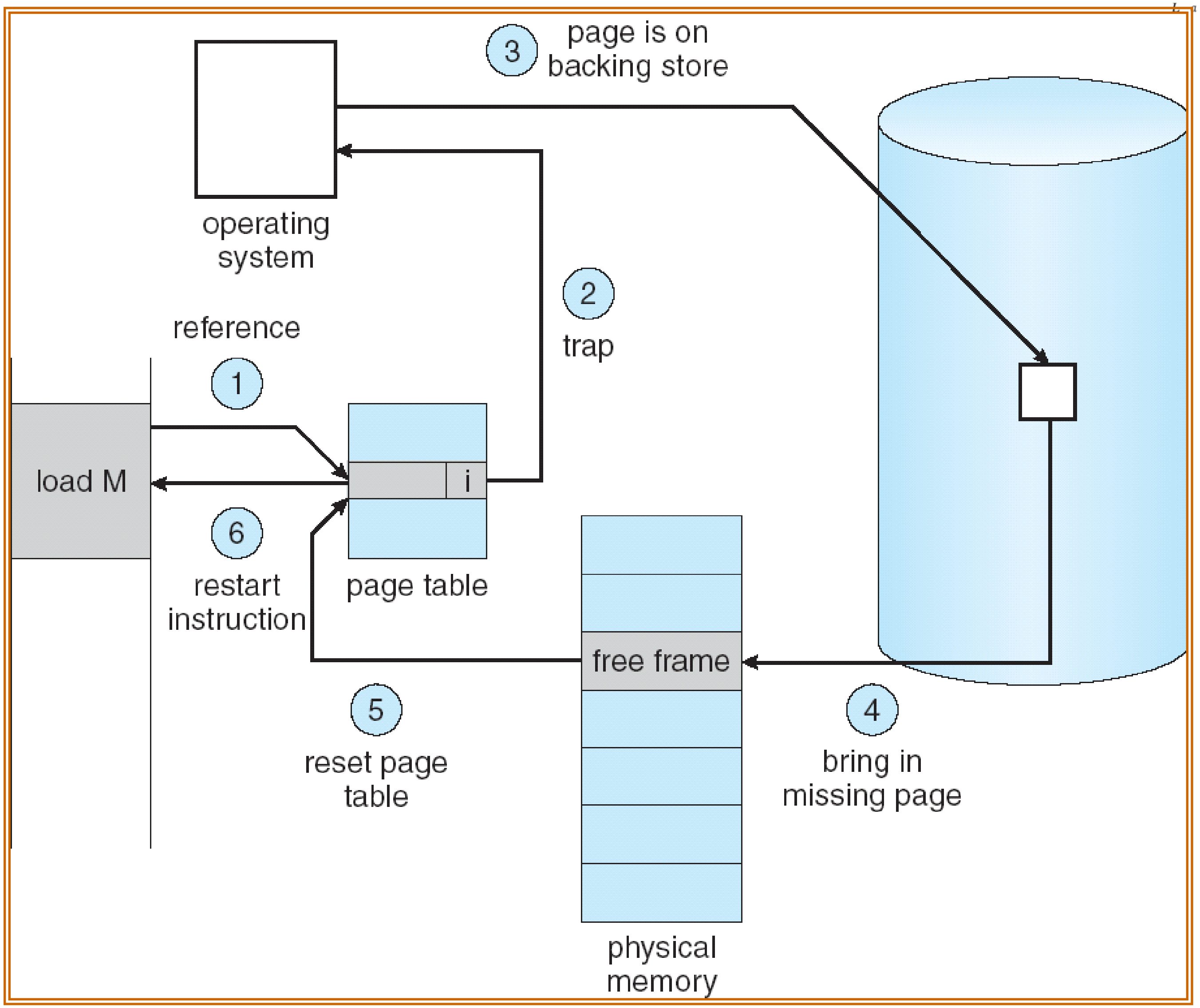
# Page Fault

- If there is a reference to a page, first reference to that page will 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
3. Swap page into frame
4. Reset tables
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

# Steps in Handling a Page Fault



# Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- Effective Access Time (EAT)

EAT = (1 – *p*) x memory access

+ *p* (page fault overhead

+ swap page out

+ swap page in

+ restart overhead

)

# Demand Paging Example

- Memory access time = 200 nanoseconds

- Average page-fault service time = 8 milliseconds

- EAT = (1 – p) x 200 + p (8 milliseconds)

  = (1 – p  x 200 + p x 8,000,000

  = 200 + p x 7,999,800

- If one access out of 1,000 causes a page fault, then

  EAT = 8.2 microseconds.

  This is a slowdown by a factor of 40!!

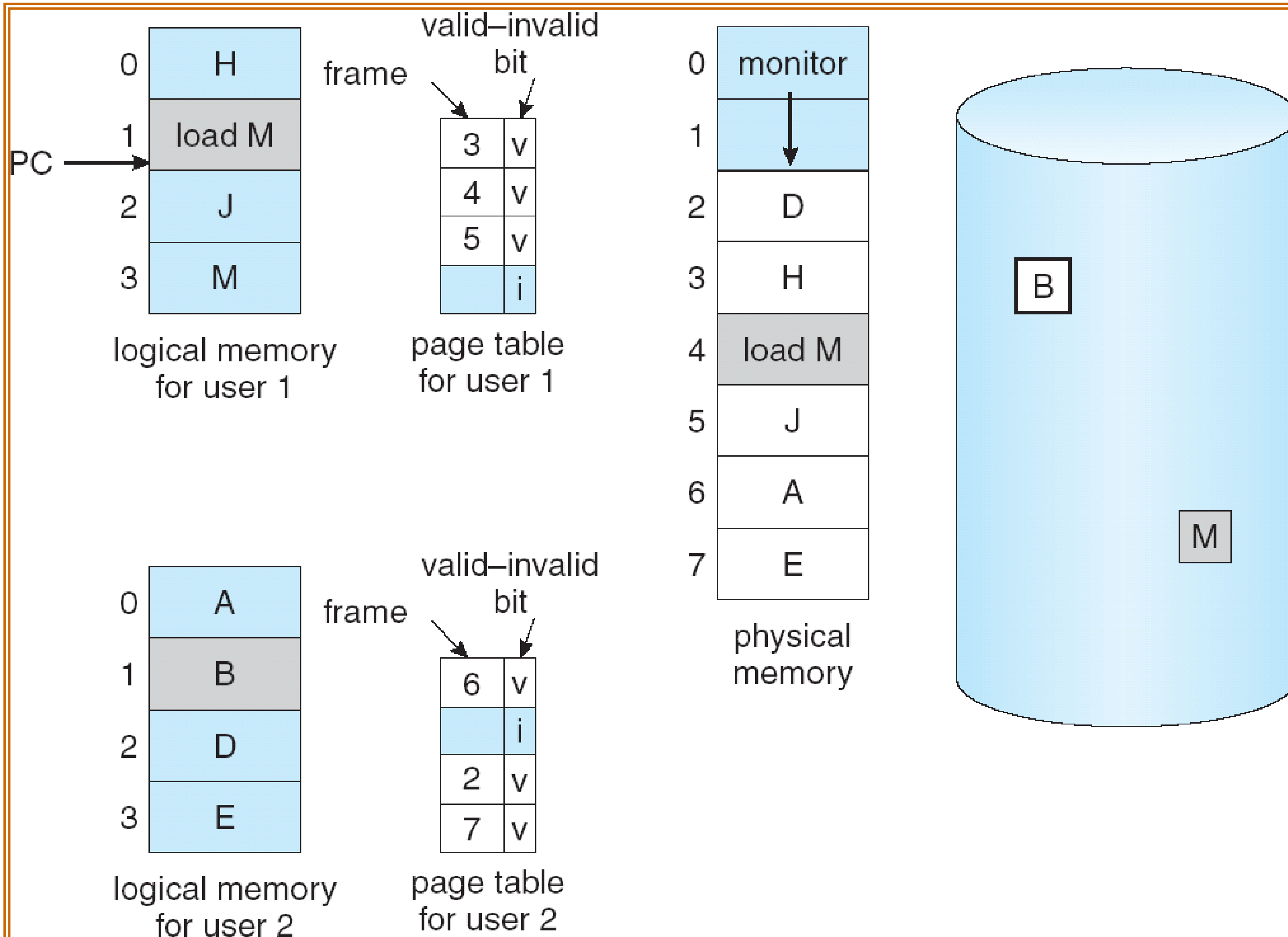
# What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

# 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

# Need For Page Replacement



# Basic Page Replacement

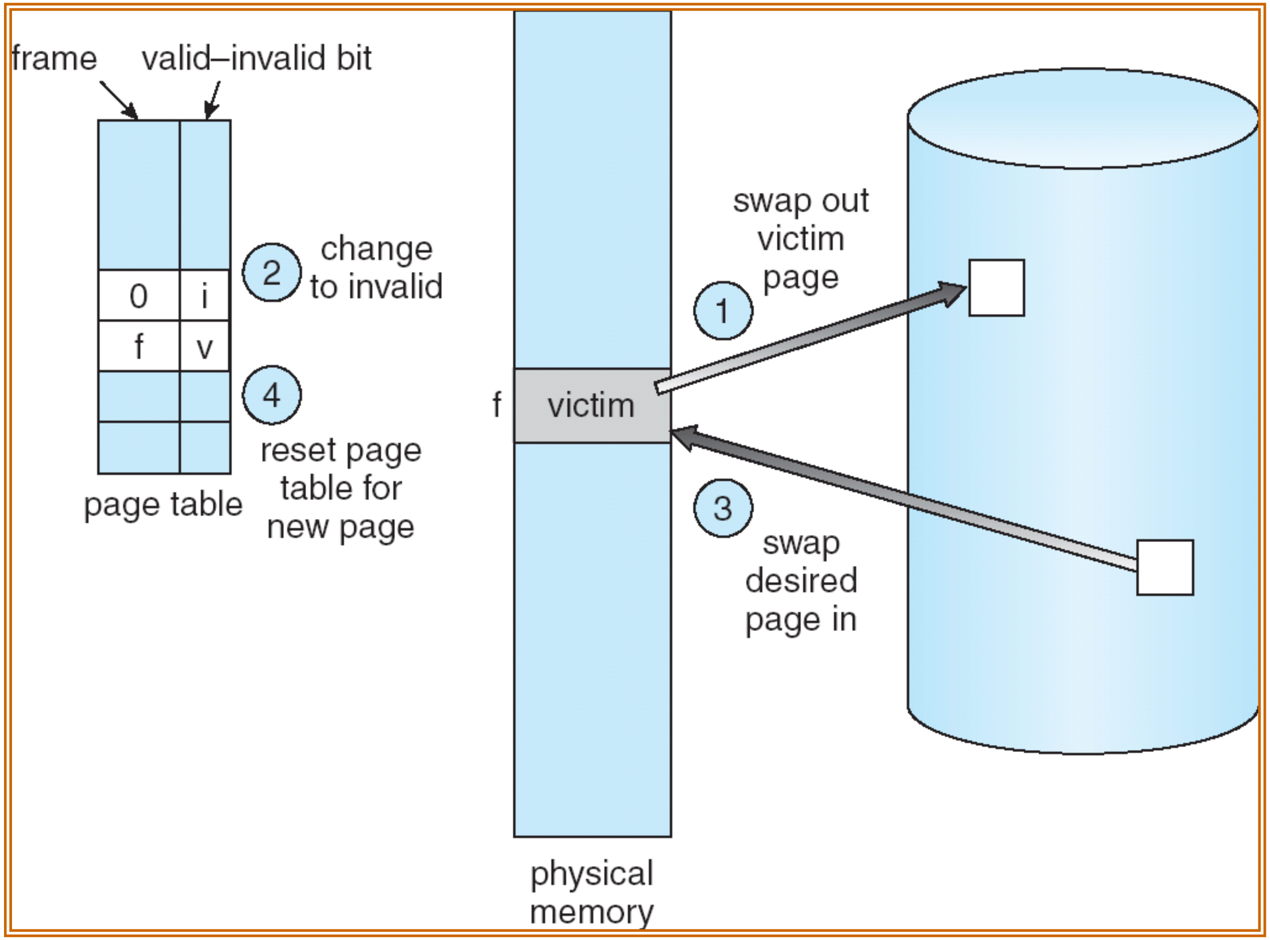
1. Find the location of the desired page on disk

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. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Restart the process

# Page Replacement

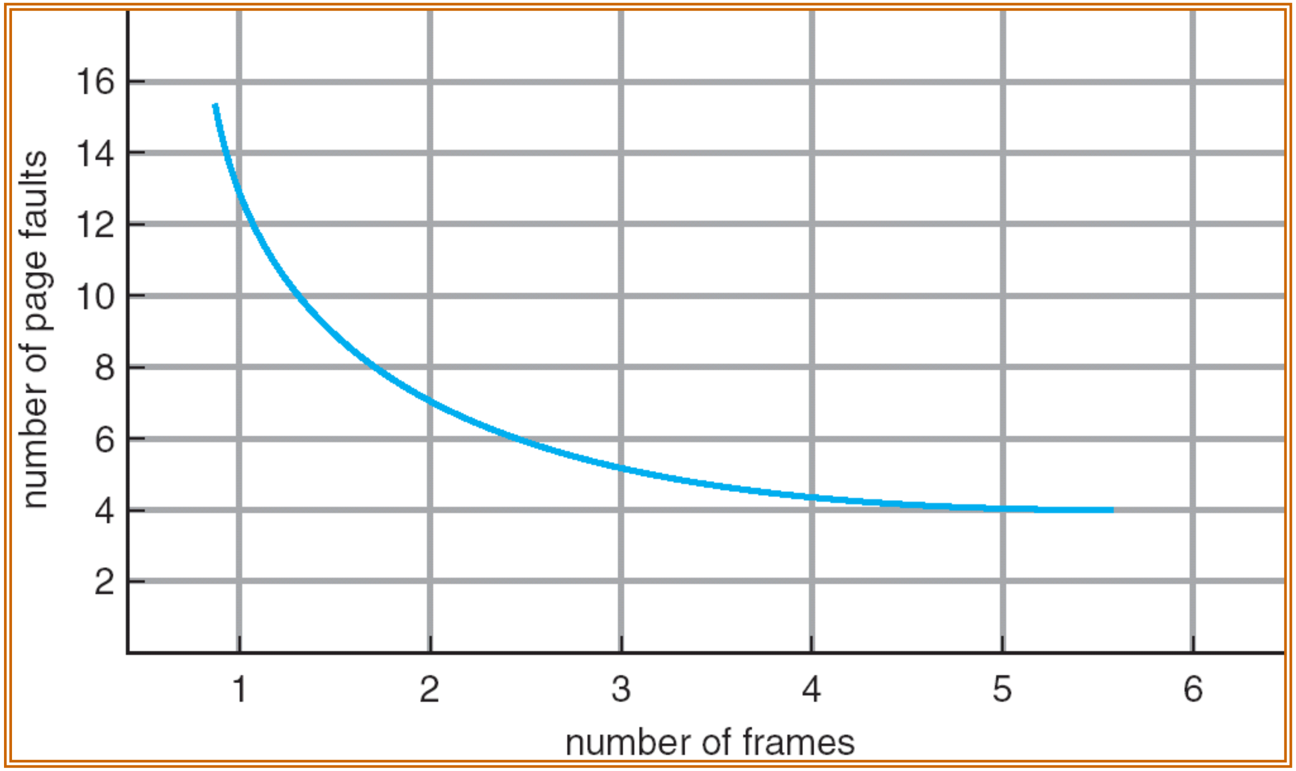


# Page Replacement Algorithms

- 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
- 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

THE UNIVERSITY OF LOUISIANA Lafayette

- 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)

| | | | |
|---|---|---|---|
| 1 | 1 | 4 | 5 |
| 2 | 2 | 1 | 3 |
| 3 | 3 | 2 | 4 |

9 page faults

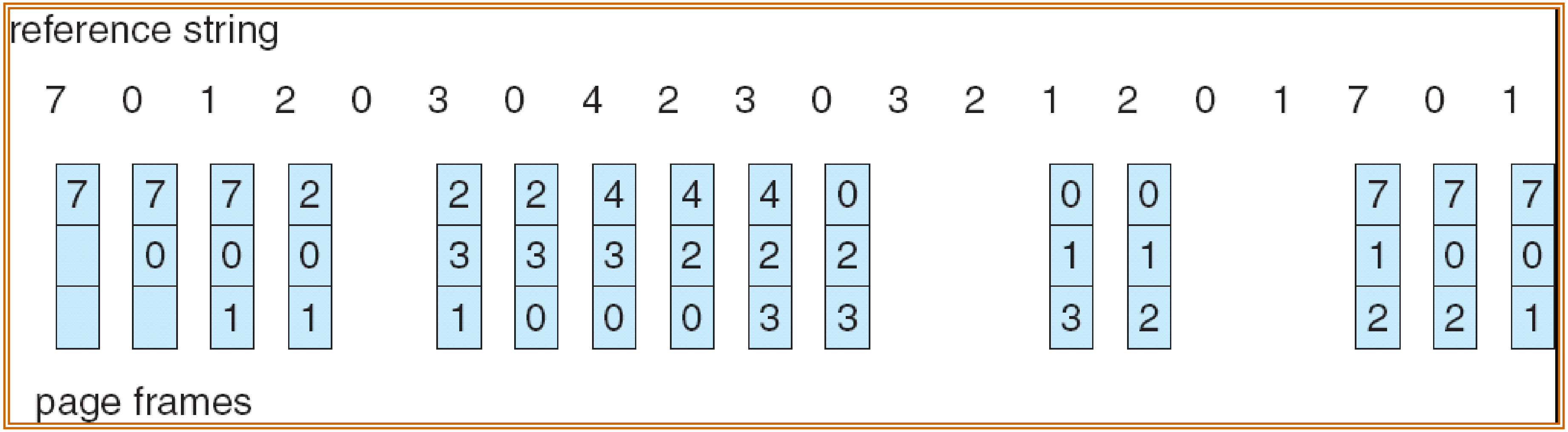
- 4 frames

| | | | |
|---|---|---|---|
| 1 | 1 | 5 | 4 |
| 2 | 2 | 1 | 5 |
| 3 | 3 | 2 | |
| 4 | 4 | 3 | |

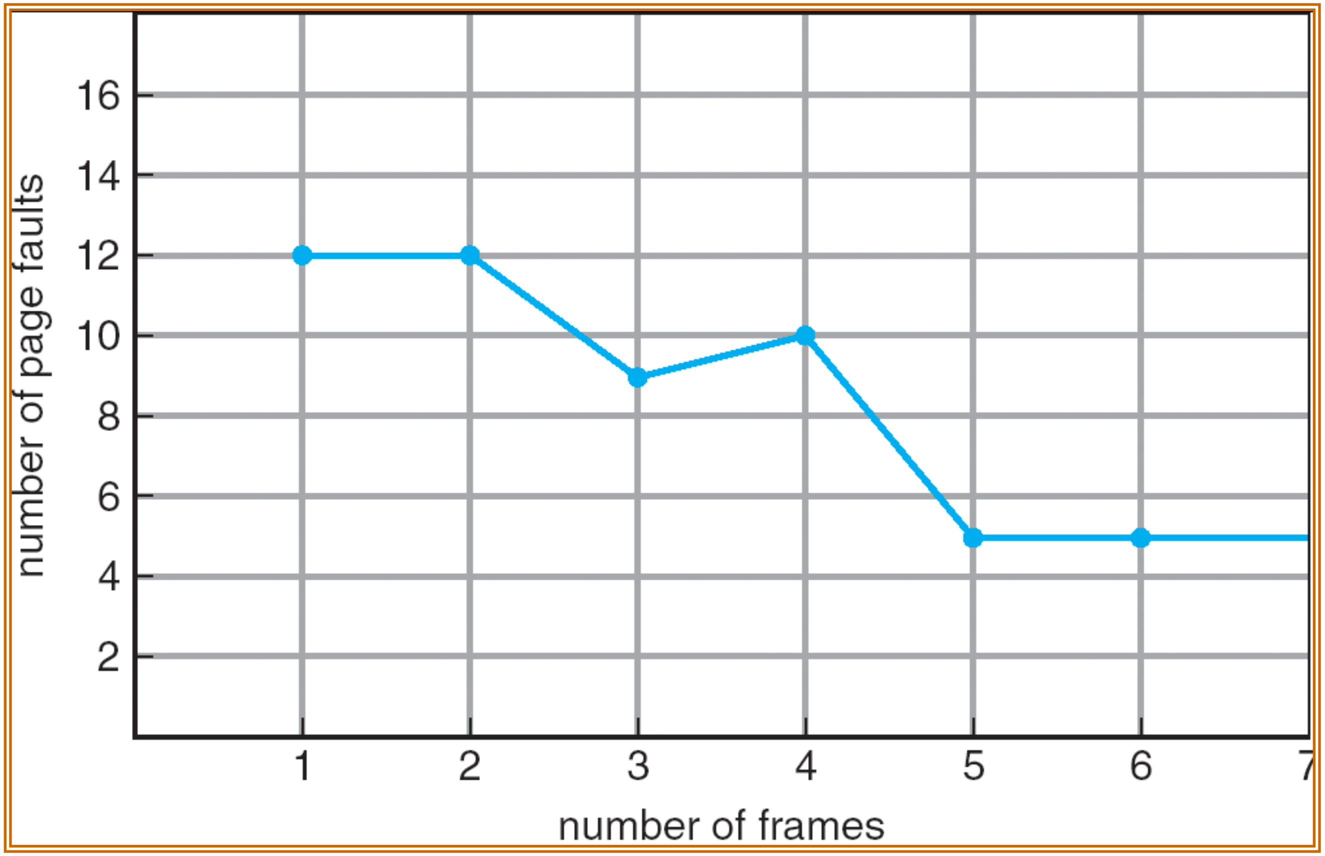
10 page faults

- Belady's Anomaly: more frames $\Rightarrow$ more page faults

# FIFO Page Replacement



# FIFO Illustrating Belady's Anomaly

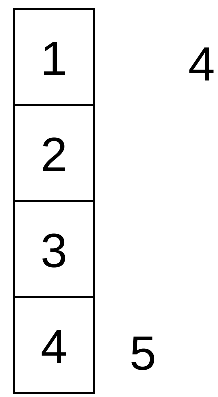


# Optimal Algorithm

- **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

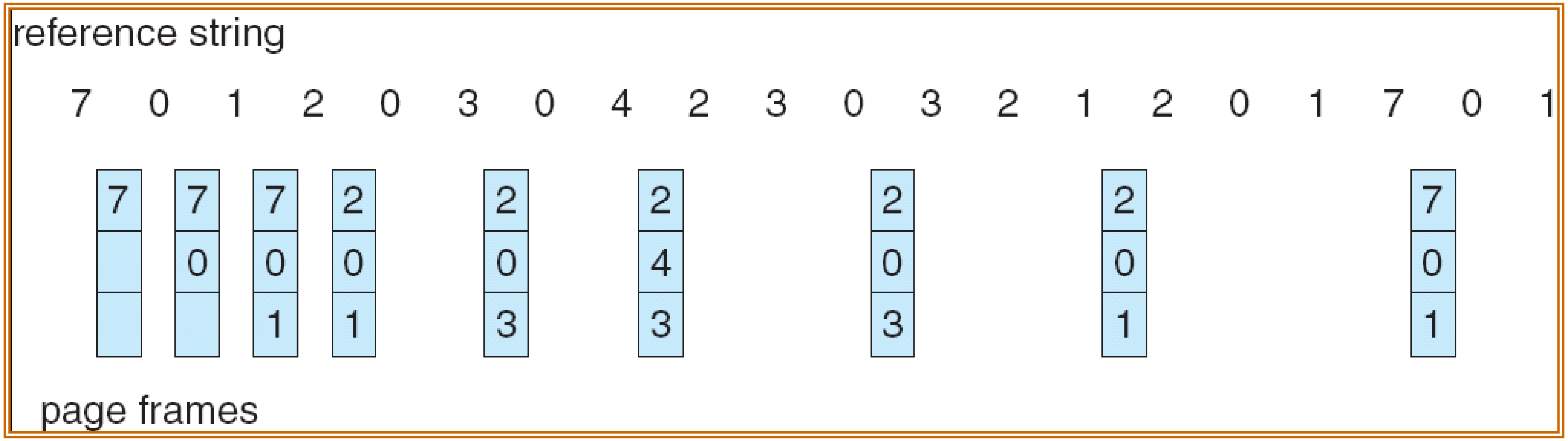



6 page faults

- How do you know this?

- Used for measuring how well your algorithm performs

# Optimal Page Replacement



# Least Recently Used (LRU) Algorithm

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

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | **5** |
| 2 | 2 | 2 | 2 | 2 |
| 3 | **5** | 5 | **4** | 4 |
| 4 | 4 | **3** | 3 | 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