

I.-C. Lin, Assistant Professor. Textbook: Operating System
Principles 7ed

CHAPTER 9: VIRTUAL- MEMORY MANAGEMENT



Chapter 9: Virtual Memory

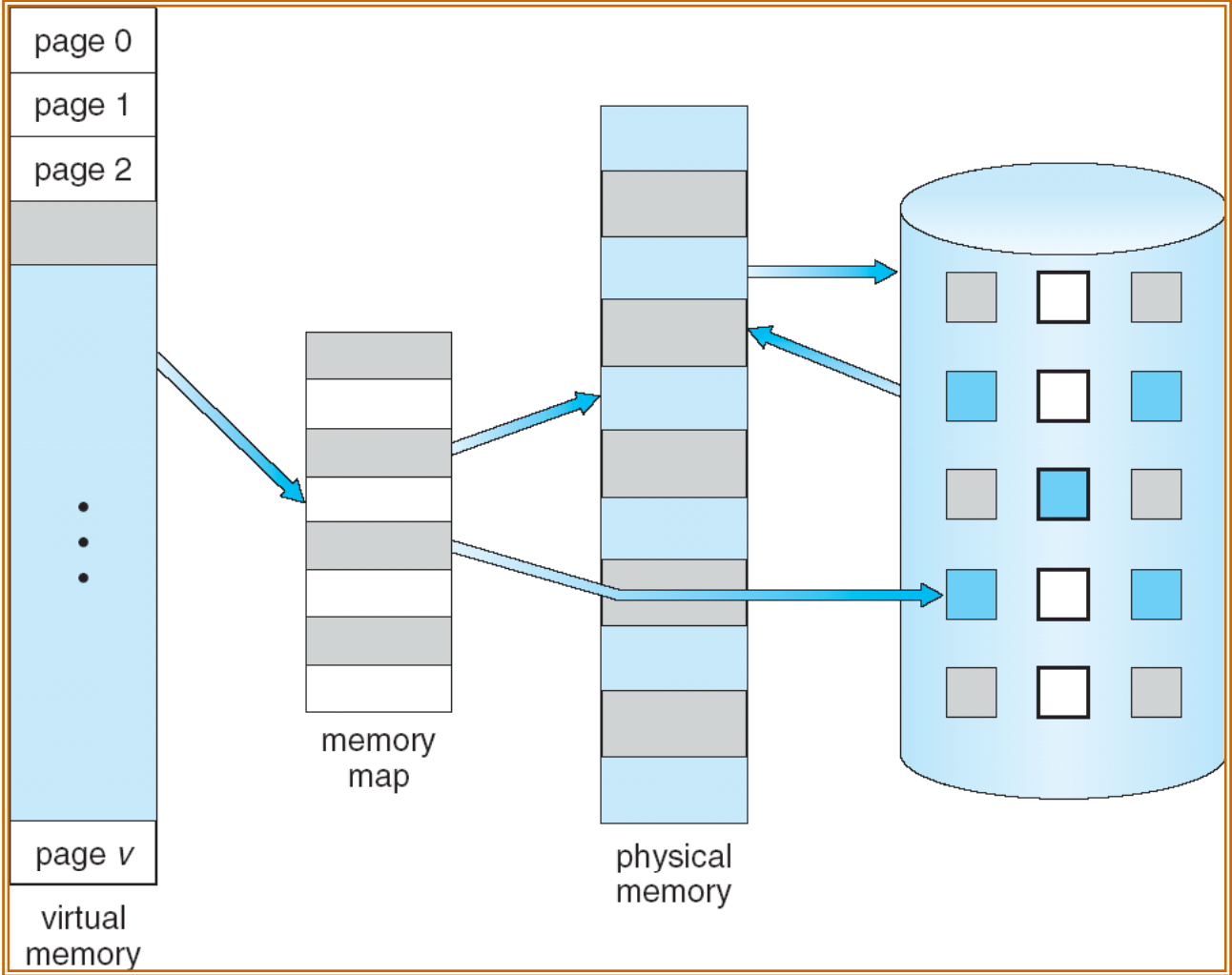
- Background
- Demand Paging
- Copy-on-Write
- Page Replacement
- Allocation of Frames
- Thrashing
- Memory-Mapped Files
- Allocation Kernel Memory
- Other Consideration
- Operating System Examples

Background

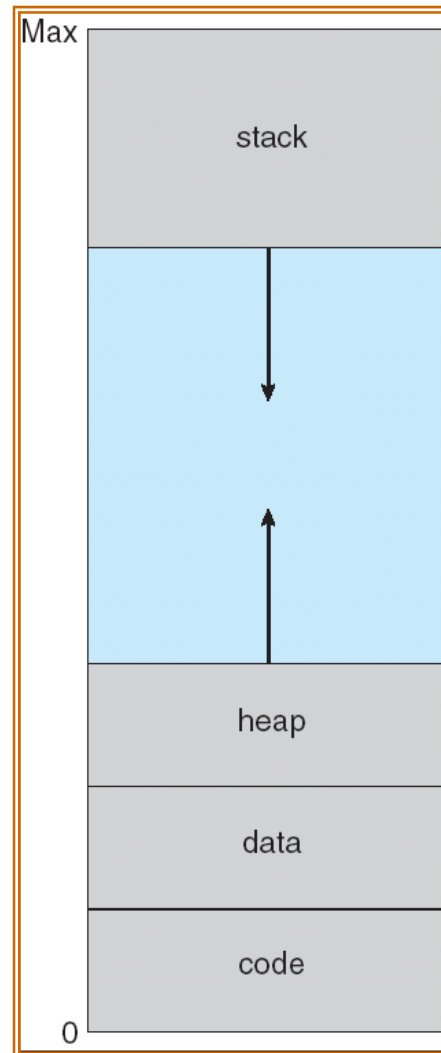
- **Virtual memory** – separation of user logical memory from physical memory.
 - Only part of the program needs to be in memory for execution.
 - Logical address space can therefore be much larger than physical address space.
 - Allows address spaces to be shared by several processes.
 - Allows for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

- Virtual memory can be implemented via:
 - Demand paging
 - Demand segmentation

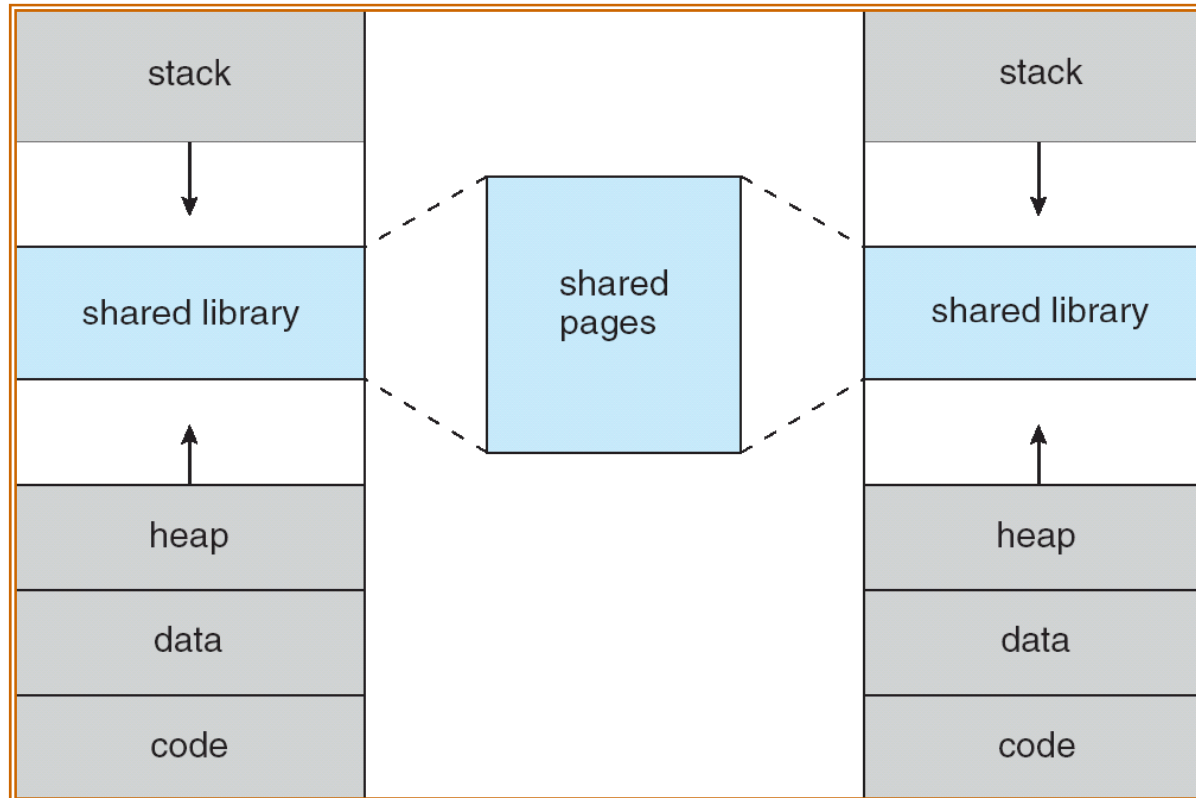
Virtual Memory That is Larger Than Physical Memory



Virtual-address Space



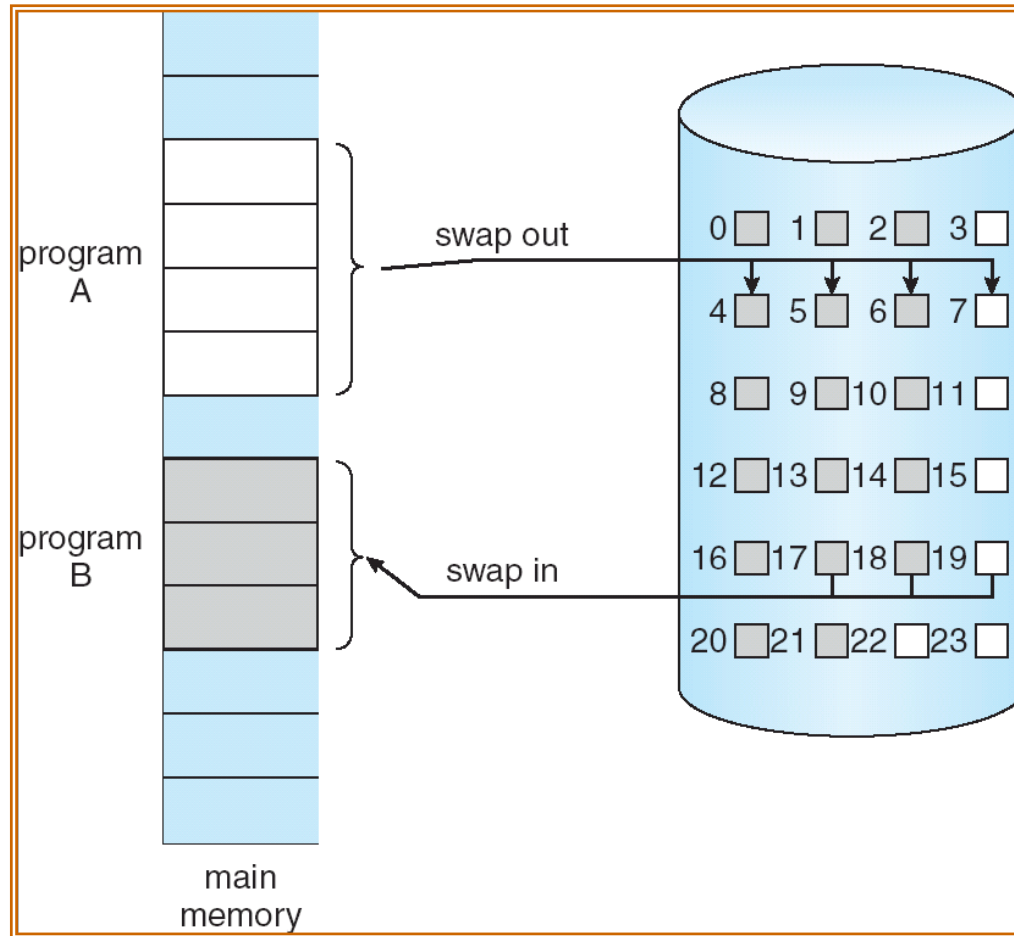
Shared Library Using Virtual Memory



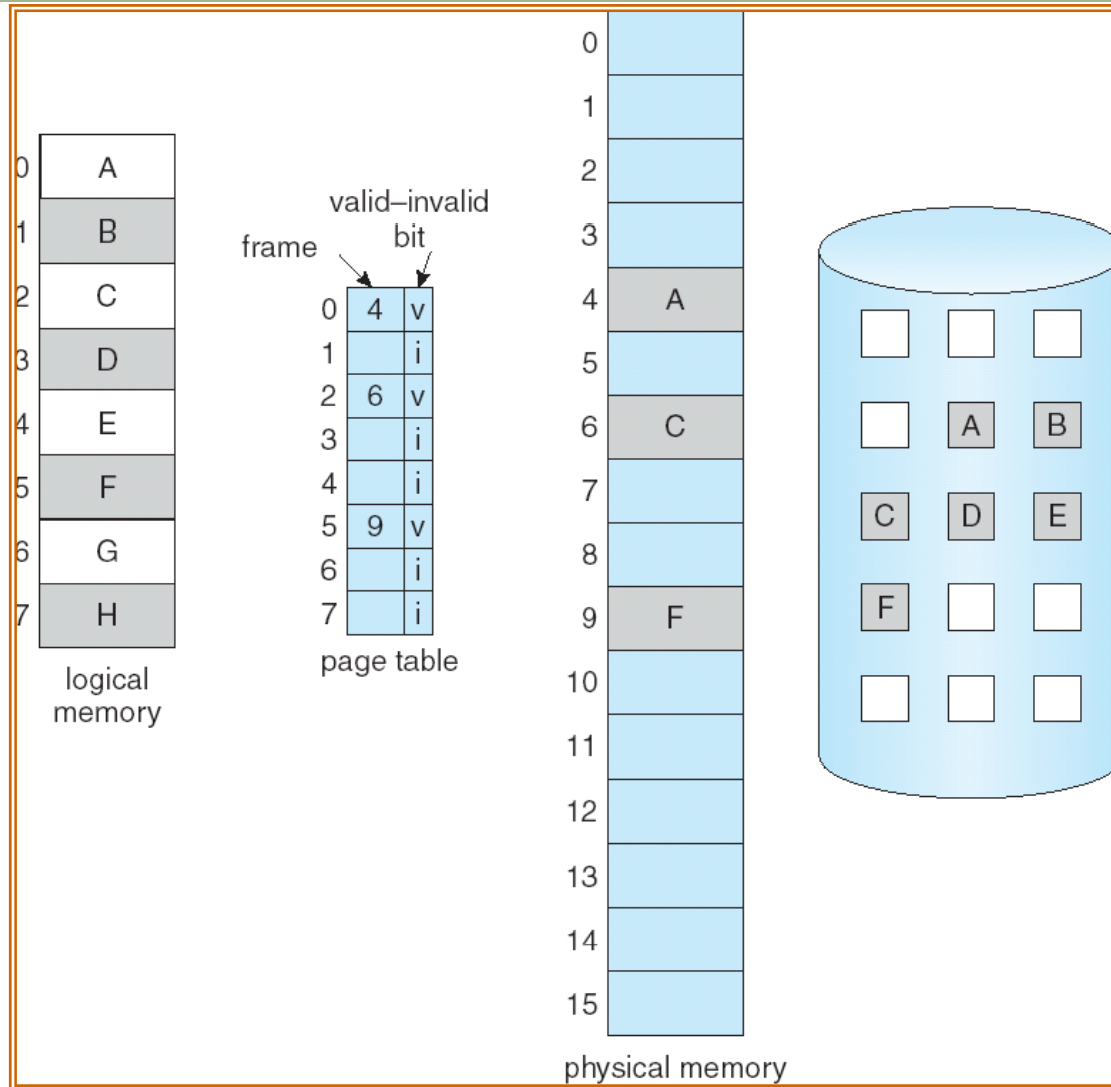
Demand Paging

- 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

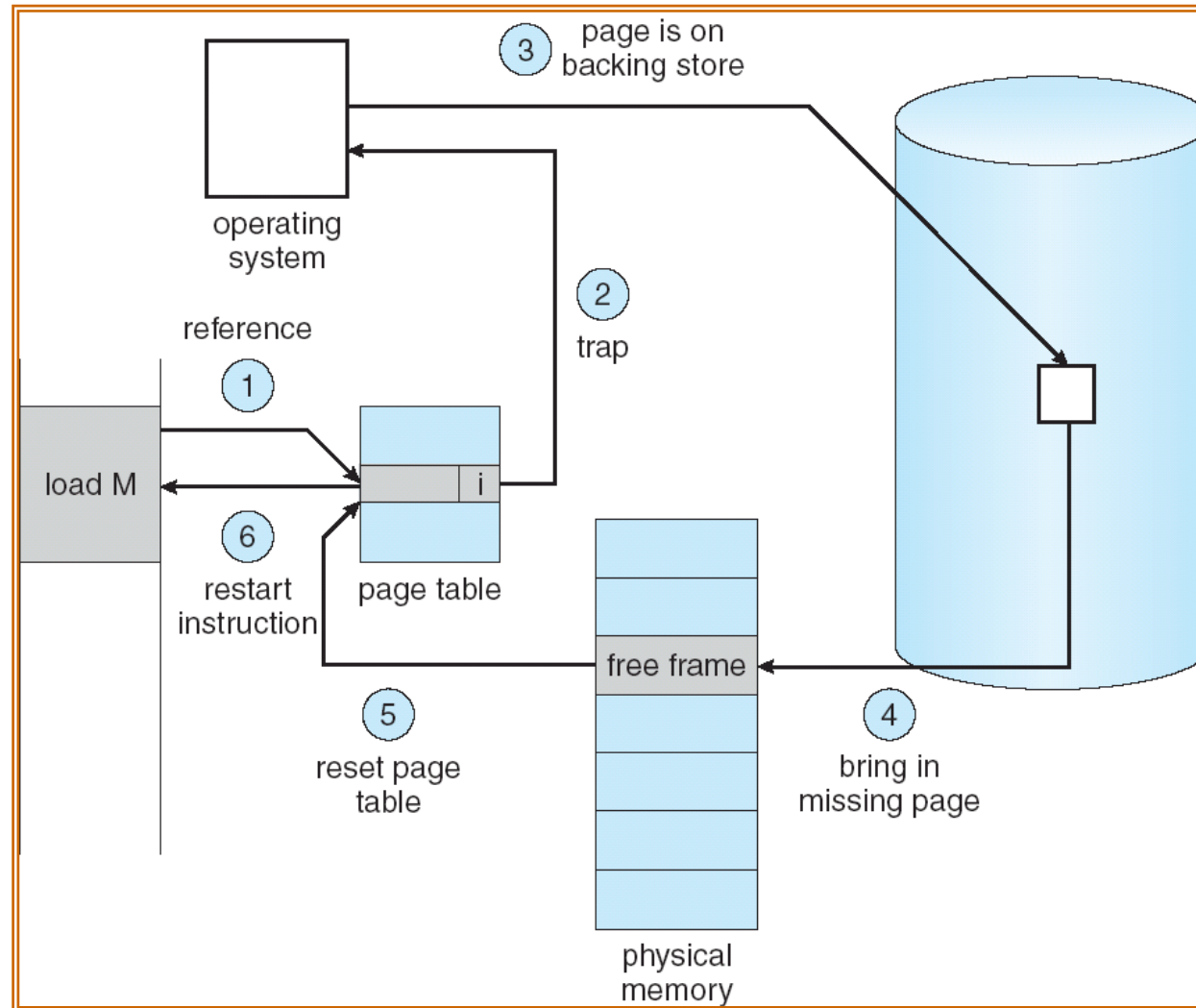
Transfer of a Paged Memory to Contiguous Disk Space



Page Table When Some Pages Are Not in Main Memory



Steps in Handling a Page Fault



Performance of Demand Paging

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

- Effective Access Time (EAT)

$$\begin{aligned} \text{EAT} = & (1 - p) \times \text{memory access} \\ & + p (\text{page fault overhead} \\ & + [\text{swap page out}] \\ & + \text{swap page in} \\ & + \text{restart overhead}) \end{aligned}$$

Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory

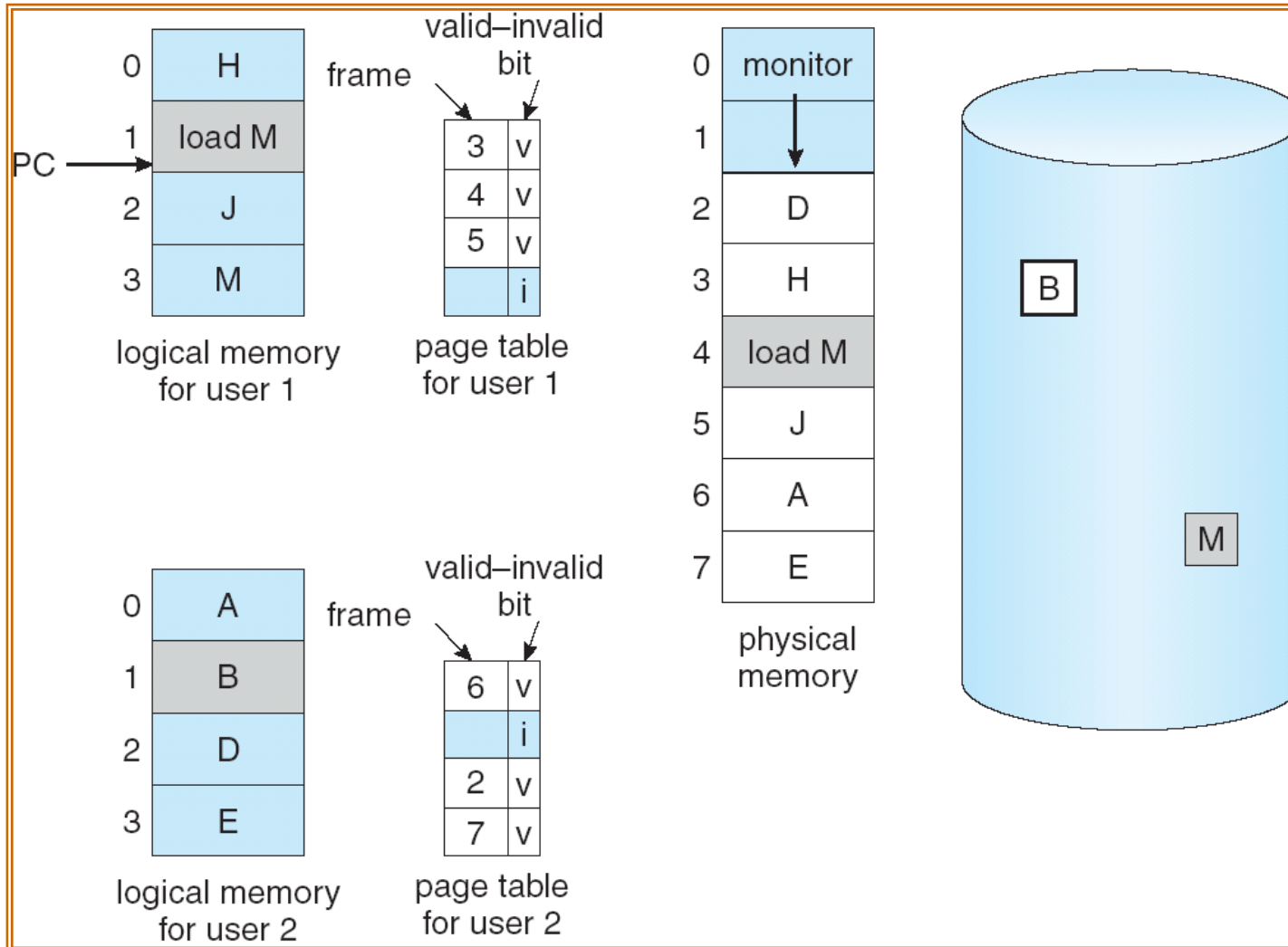
If either process modifies a shared page, only then is the page copied

- COW allows more efficient process creation as only modified pages are copied
- Free pages are allocated from a **pool** of zeroed-out pages

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

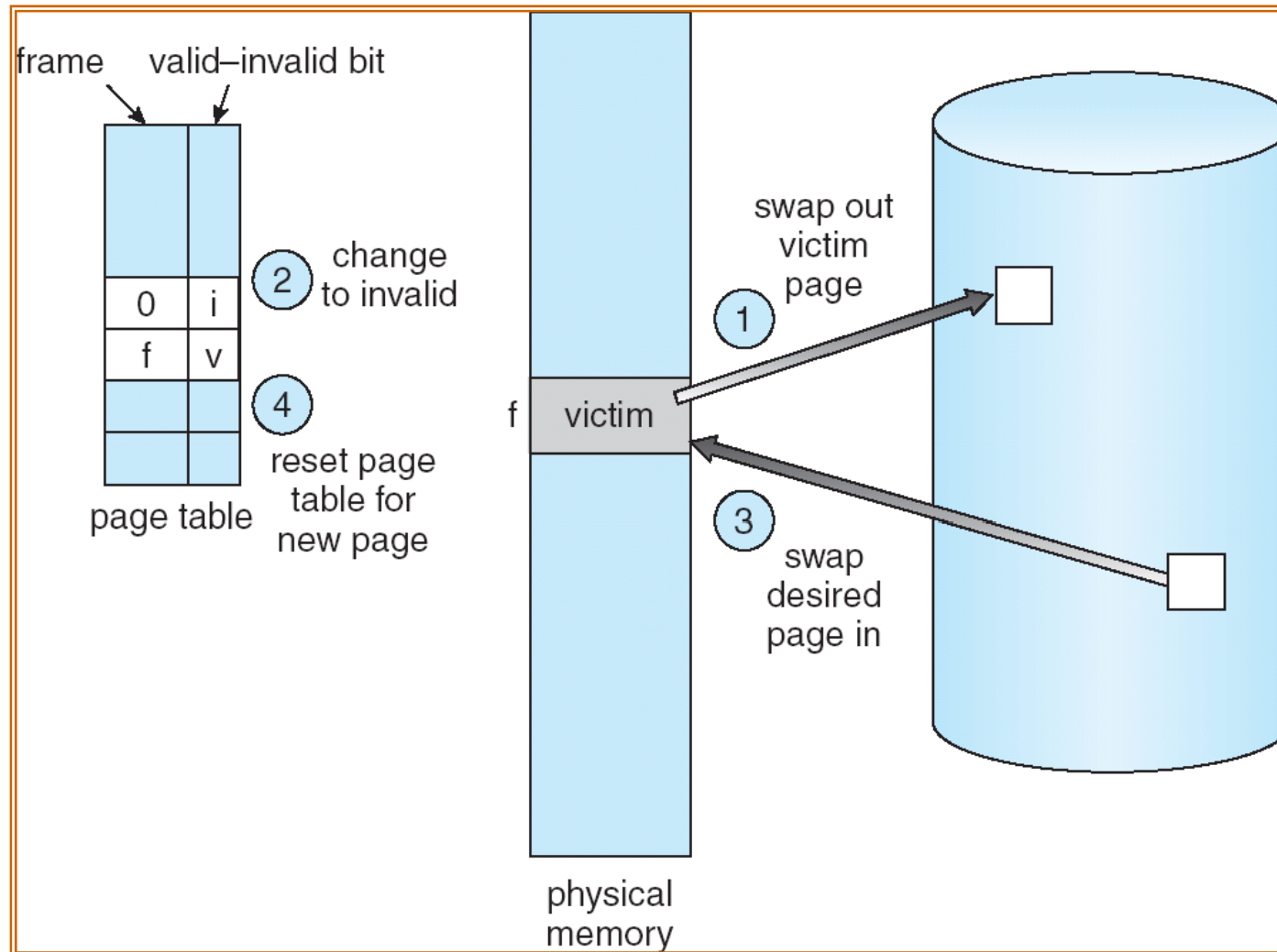
Need For Page Replacement



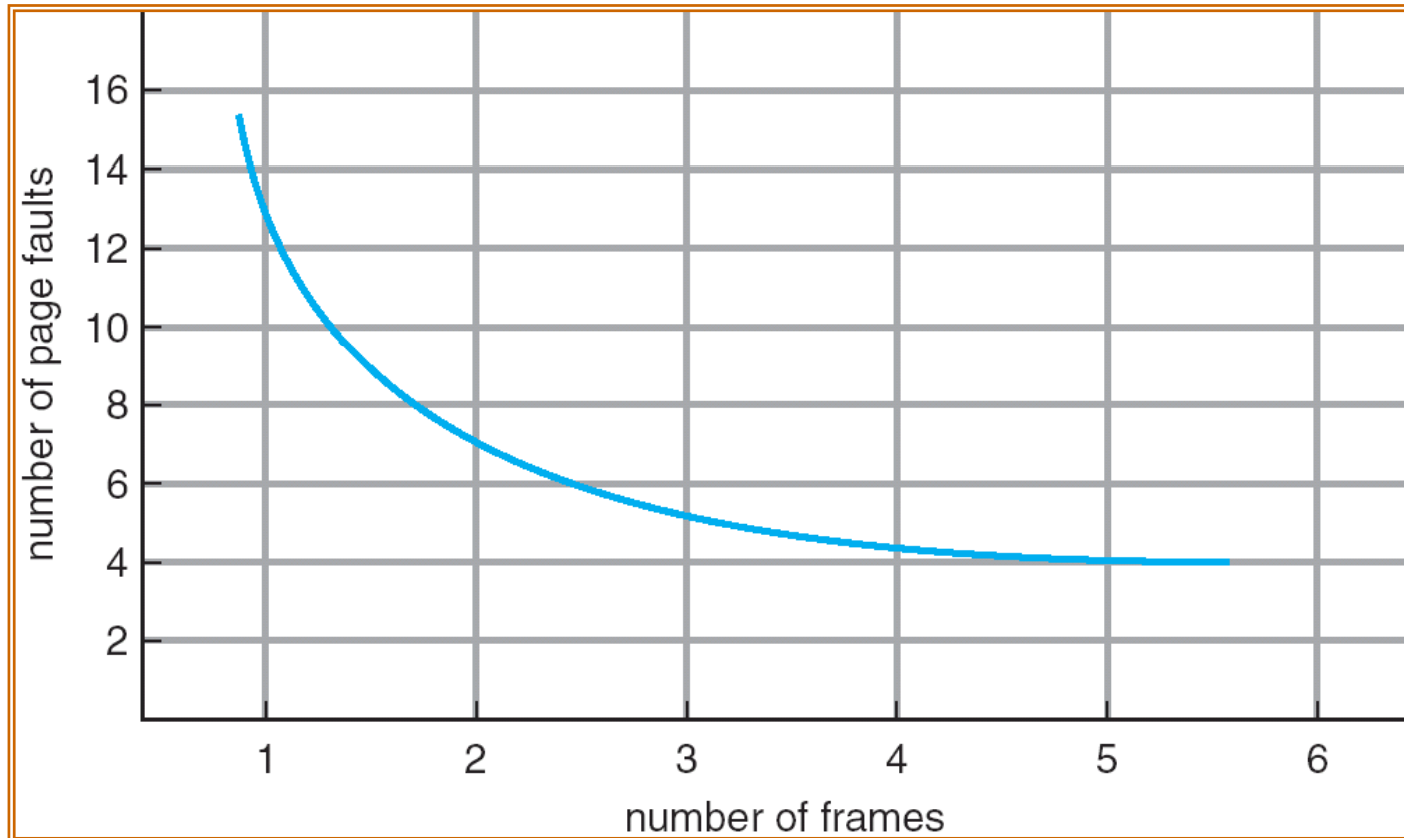
Basic Page Replacement

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. Read the desired page into the (newly) free frame. Update the page and frame tables.
4. Restart the process

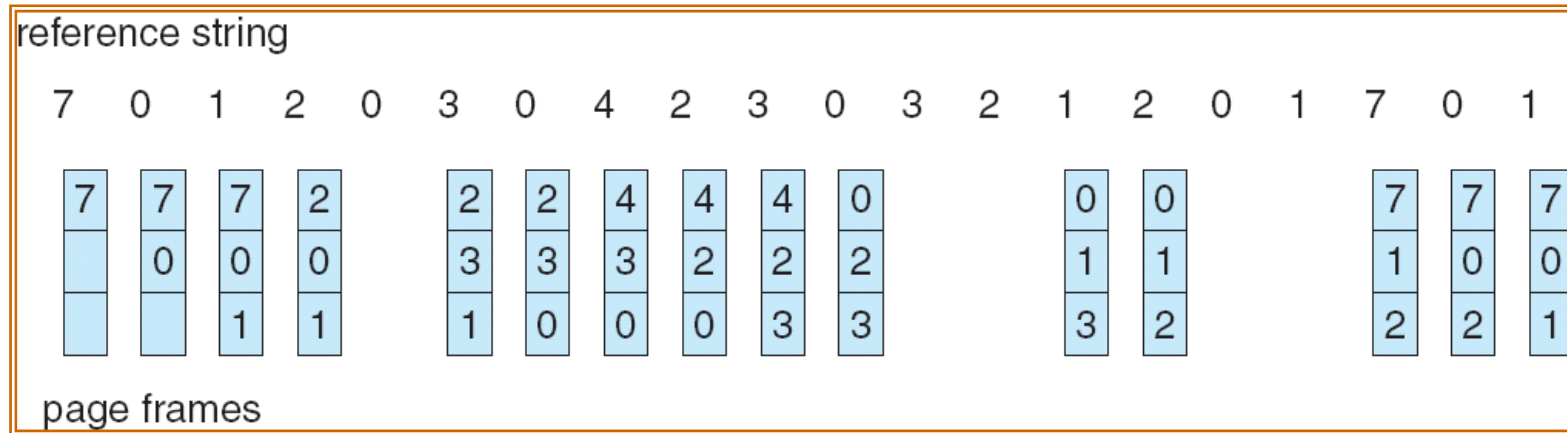
Page Replacement



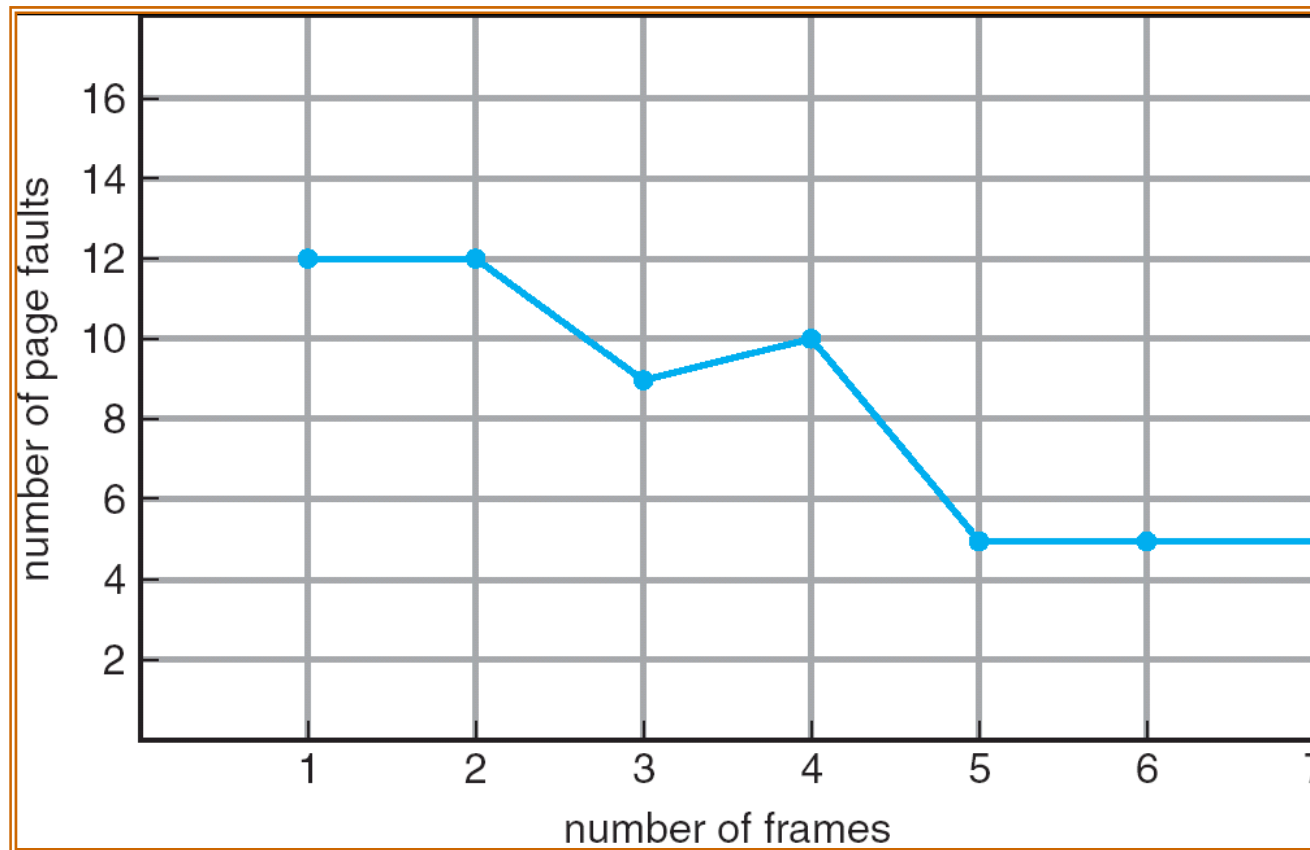
Graph of Page Faults Versus The Number of Frames



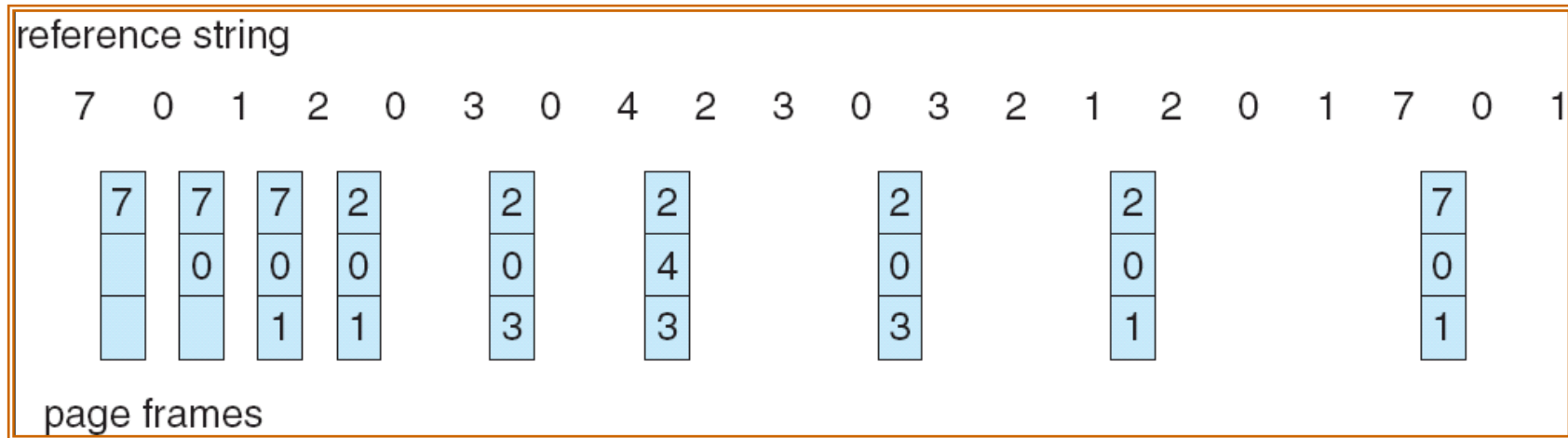
FIFO Page Replacement



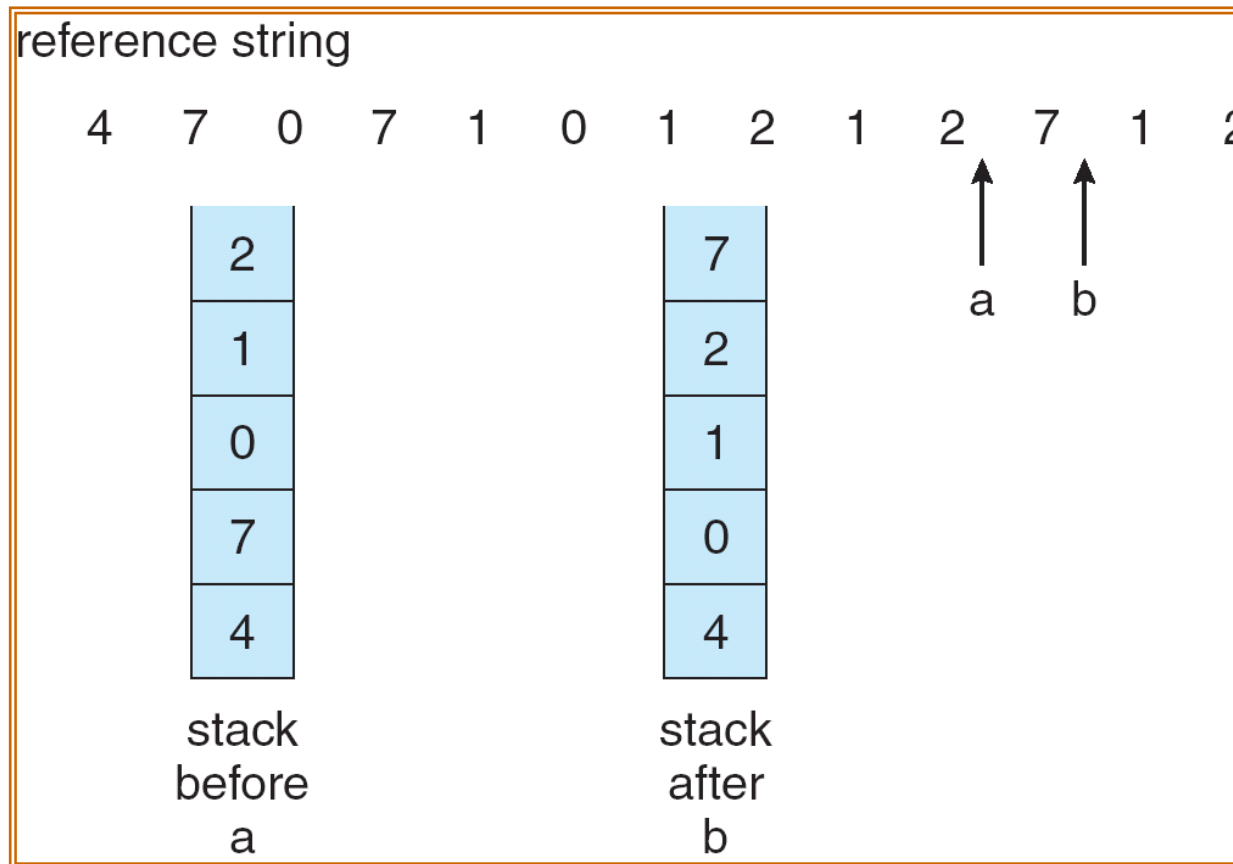
FIFO Illustrating Belady's Anomaly



Optimal Page Replacement



Use Of A Stack to Record The Most Recent Page References

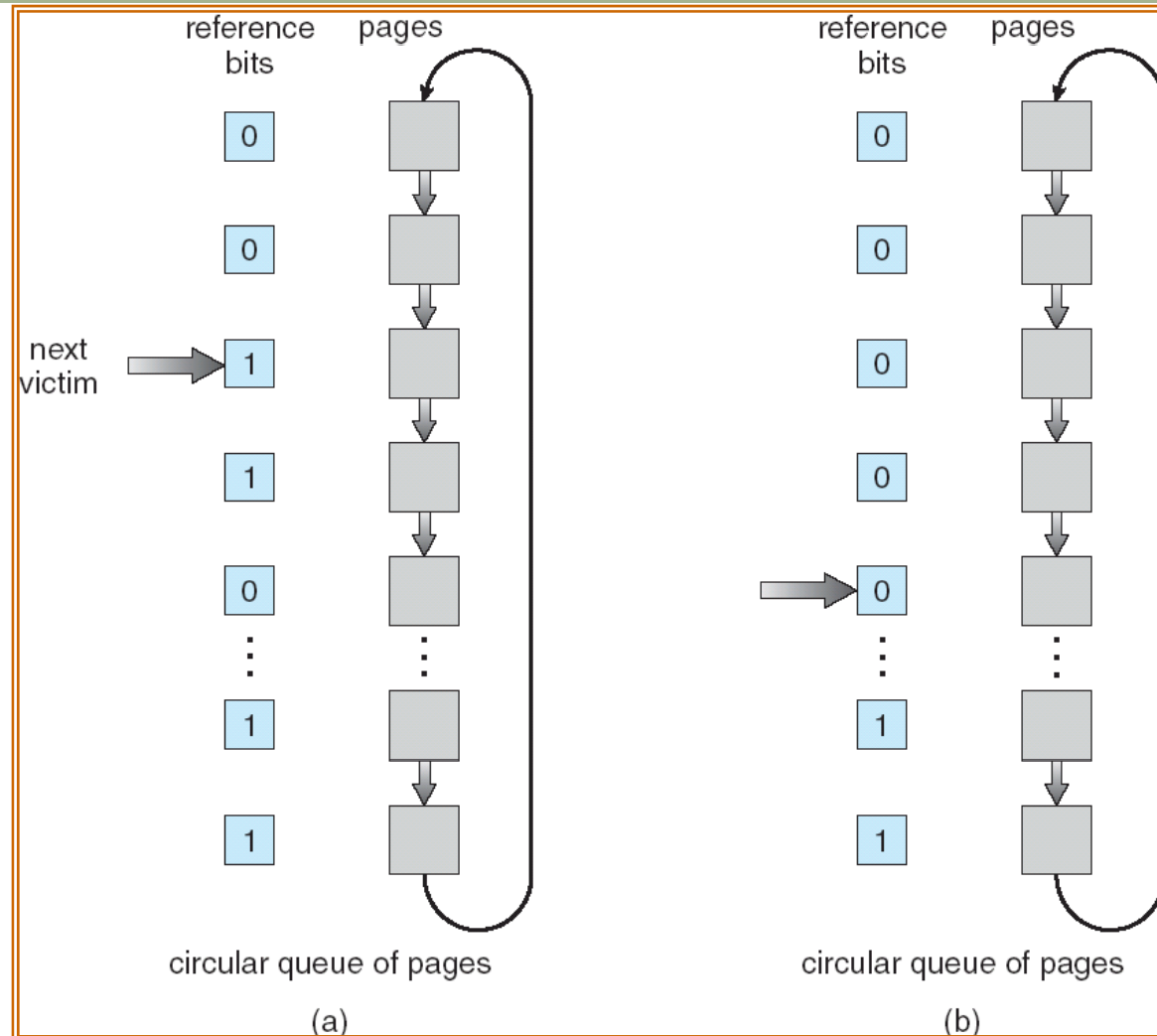


LRU Approximation Algorithms

- Reference bit
 - ▣ With each page associate a bit, initially = 0
 - ▣ When page is referenced bit set to 1
 - ▣ Replace the one which is 0 (if one exists). We do not know the order, however.

- Second chance
 - ▣ Need reference bit
 - ▣ Clock replacement
 - ▣ If page to be replaced (in clock order) has reference bit = 1 then:
 - set reference bit 0
 - leave page in memory
 - replace next page (in clock order), subject to same rules

Second-Chance (clock) Page-Replacement Algorithm



Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- **LFU Algorithm:** replaces page with smallest count
- **MFU Algorithm:** based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Allocation of Frames

- Each process needs *minimum* number of pages

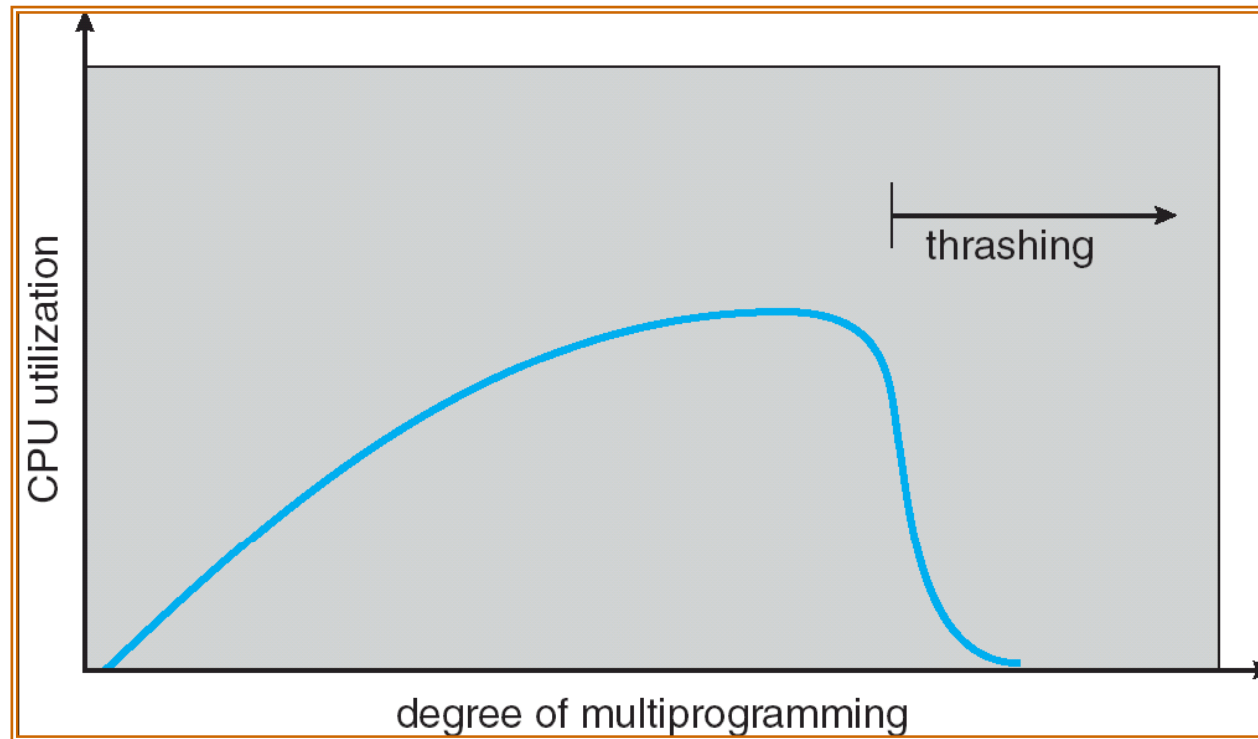
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
 - ▣ instruction is 6 bytes, might span 2 pages
 - ▣ 2 pages to handle *from*
 - ▣ 2 pages to handle *to*

- Two major allocation schemes
 - ▣ fixed allocation
 - ▣ priority allocation

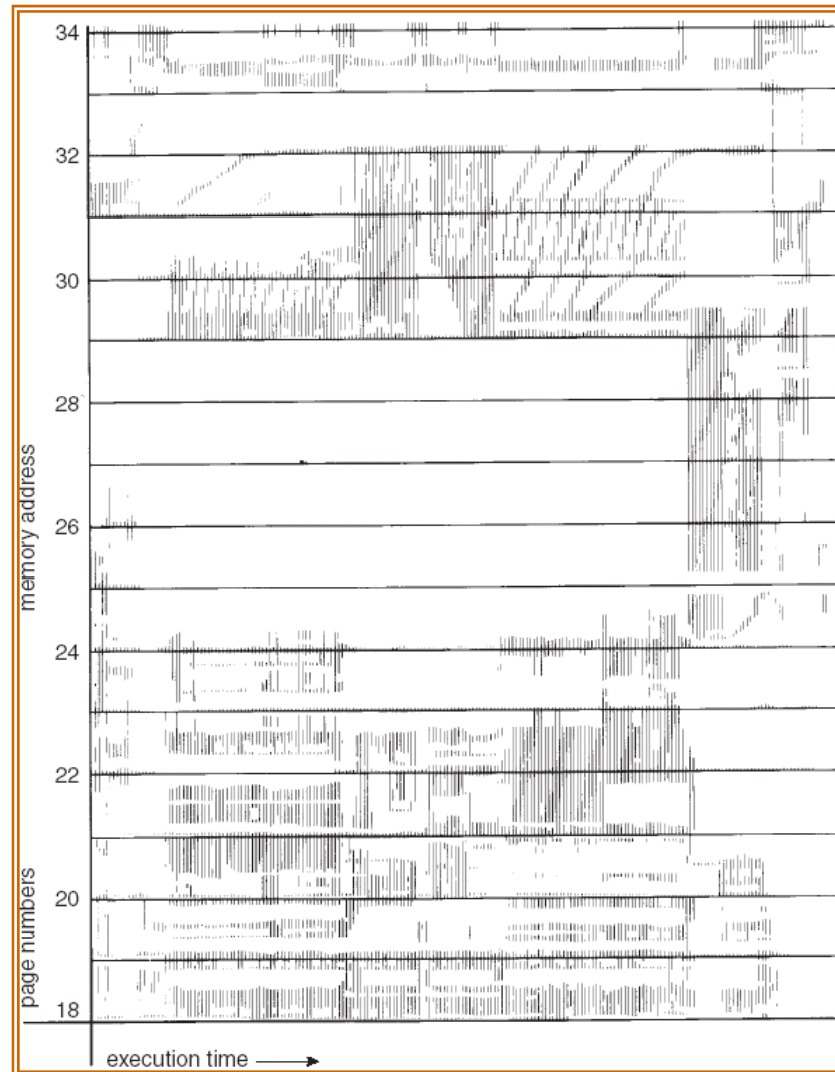
Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
 - low CPU utilization
 - operating system thinks that it needs to increase the degree of multiprogramming
 - another process added to the system
- **Thrashing** \equiv a process is busy swapping pages in and out

Thrashing (Cont.)



Locality In A Memory-Reference Pattern



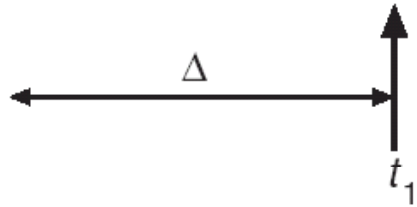
Working-Set Model

- $\Delta \equiv$ working-set window \equiv a fixed number of page references
Example: 10,000 instruction
- WSS_i (working set of Process F_i) =
total number of pages referenced in the most recent Δ (varies in time)
 - ▣ if Δ too small will not encompass entire locality
 - ▣ if Δ too large will encompass several localities
 - ▣ if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \sum WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if $D > m$, then suspend one of the processes

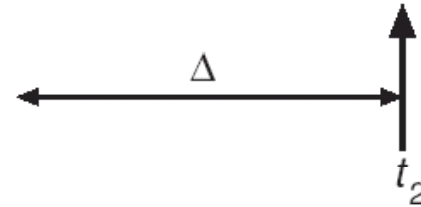
Working-set model

page reference table

... 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



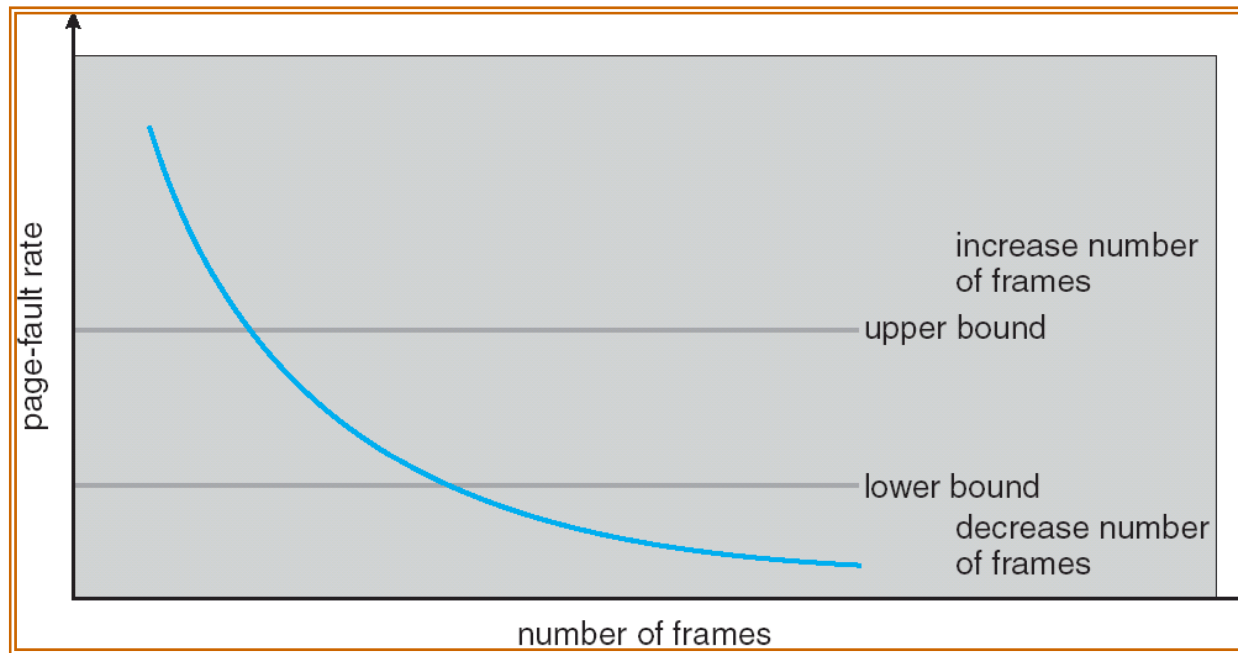
$$WS(t_1) = \{1, 2, 5, 6, 7\}$$



$$WS(t_2) = \{3, 4\}$$

Page-Fault Frequency Scheme

- Establish “acceptable” page-fault rate
 - ▣ If actual rate too low, process loses frame
 - ▣ If actual rate too high, process gains frame



Memory-Mapped Files

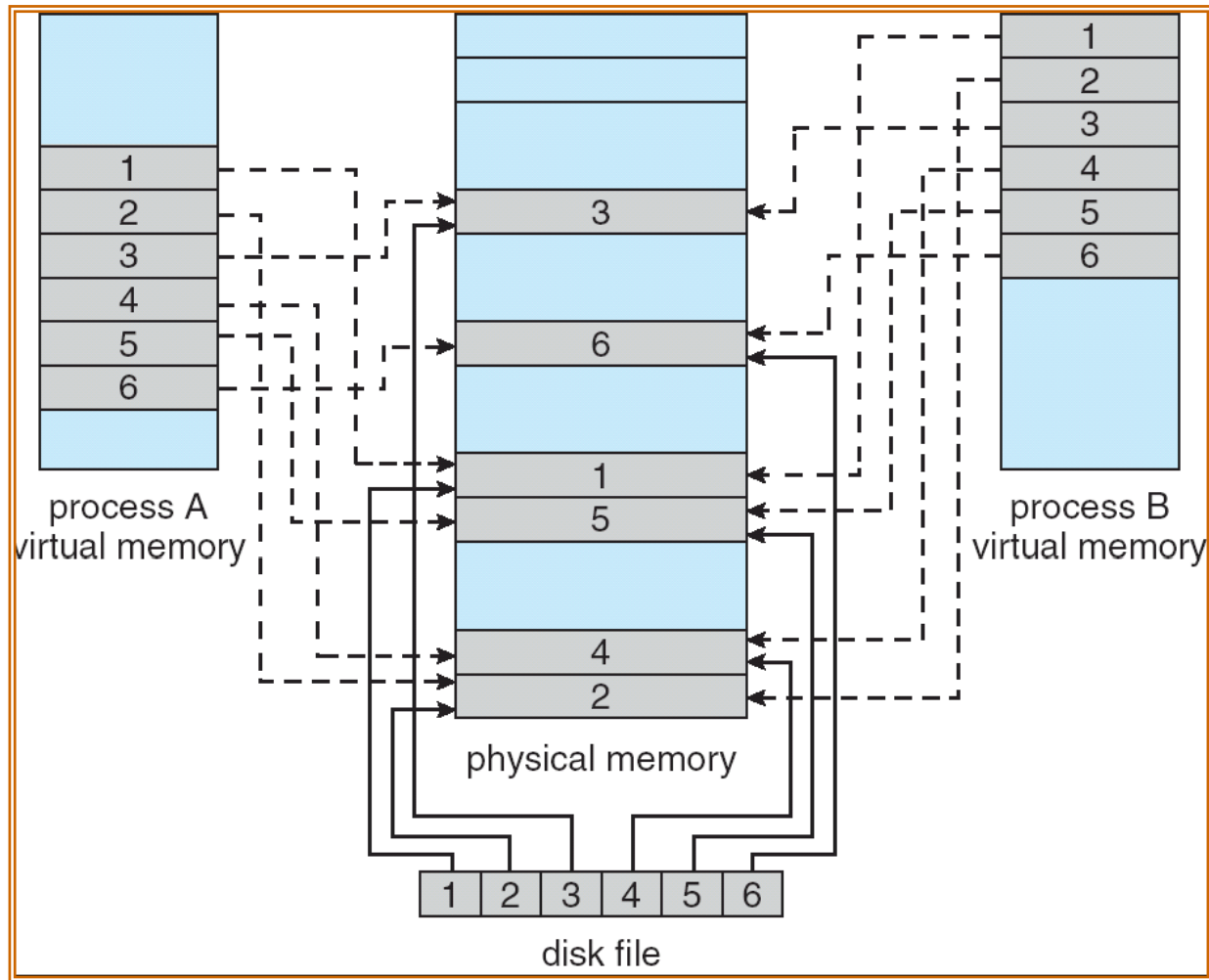
- Memory-mapped file I/O allows file I/O to be treated as routine memory access by **mapping** a disk block to a page in memory
- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses.

Memory-Mapped Files (cont.)



- Simplifies file access by treating file I/O through memory rather than `read()` `write()` system calls
- Also allows several processes to map the same file allowing the pages in memory to be shared

Memory Mapped Files



Prepaging

□ Prepaging

- To reduce the large number of page faults that occurs at process startup
- Prepage all or some of the pages a process will need, before they are referenced
- But if prepaged pages are unused, I/O and memory was wasted
- Assume s pages are prepaged and α of the pages is used
 - Is cost of $s * \alpha$ save pages faults $>$ or $<$ than the cost of prepaging
 $s * (1 - \alpha)$ unnecessary pages?
 - α near zero \Rightarrow prepaging loses

Page Size



- Page size selection must take into consideration:
 - Fragmentation
 - table size
 - I/O overhead
 - locality

TLB Reach

- TLB Reach - The amount of memory accessible from the TLB
- $TLB\ Reach = (TLB\ Size) \times (Page\ Size)$
- Ideally, the working set of each process is stored in the TLB. Otherwise there is a high degree of page faults.
- Increase the Page Size. This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes. This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation.

Program Structure

- Program structure

- `Int[128,128] data;`
- Each row is stored in one page
- Program 1

```
for (j = 0; j < 128; j++)  
  for (i = 0; i < 128; i++)  
    data[i,j] = 0;
```

128 x 128 = 16,384 page faults

- Program 2

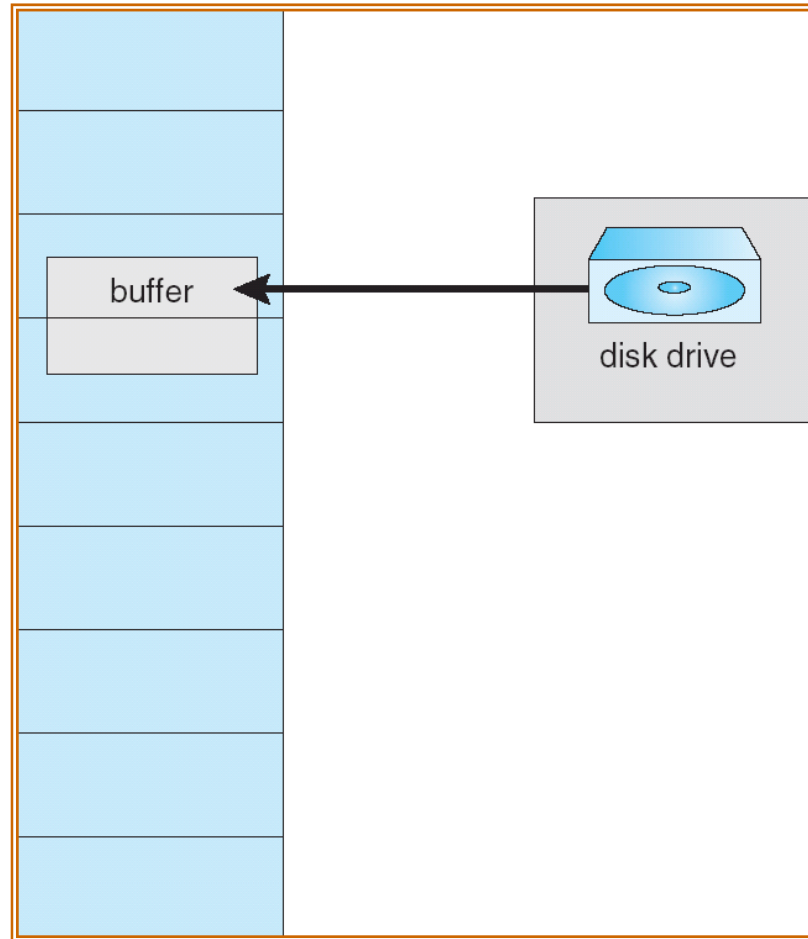
```
for (i = 0; i < 128; i++)  
  for (j = 0; j < 128; j++)  
    data[i,j] = 0;
```

128 page faults

I/O interlock

- I/O Interlock – Pages must sometimes be locked into memory
- Consider I/O. Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm.

Reason Why Frames Used For I/O Must Be In Memory



Operating System Examples



- Windows XP
- Solaris

Windows XP

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page.
- Processes are assigned **working set minimum** and **working set maximum**
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory

Windows XP (cont.)



- A process may be assigned as many pages up to its working set maximum.
- When the amount of free memory in the system falls below a threshold, **automatic working set trimming** is performed to restore the amount of free memory.
- Working set trimming removes pages from processes that have pages in excess of their working set minimum.

Solaris

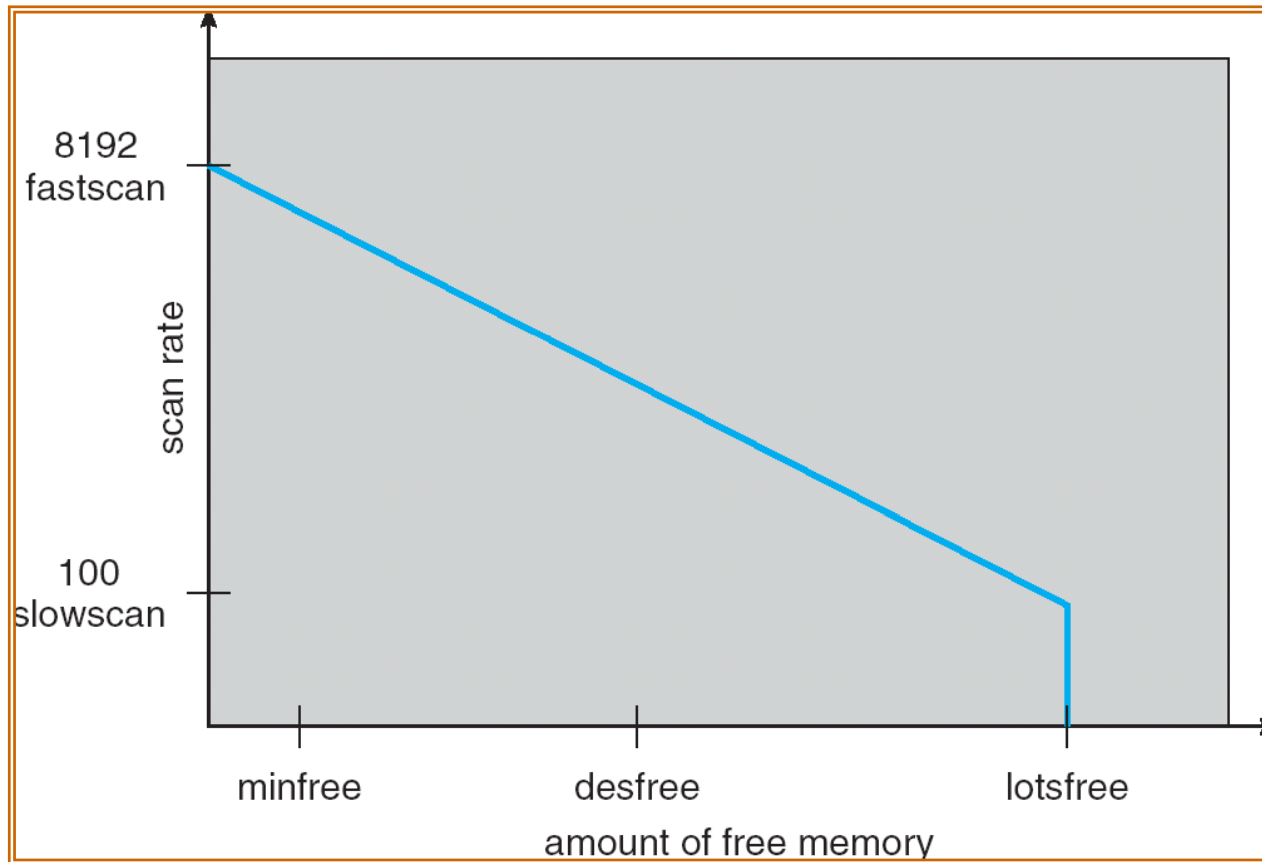
- Maintains a list of free pages to assign faulting processes
- *Lotsfree* – threshold parameter (amount of free memory) to begin paging
- *Desfree* – threshold parameter to increasing paging
- *Minfree* – threshold parameter to being swapping
- Paging is performed by *pageout* process

Solaris (cont.)



- Pageout scans pages using modified clock algorithm
- *Scanrate* is the rate at which pages are scanned. This ranges from *slowscan* to *fastscan*
- Pageout is called more frequently depending upon the amount of free memory available

Solaris 2 Page Scanner



END OF CHAPTER 9

