Chapter 9: Virtual Memory

Chapter 9: Virtual Memory

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

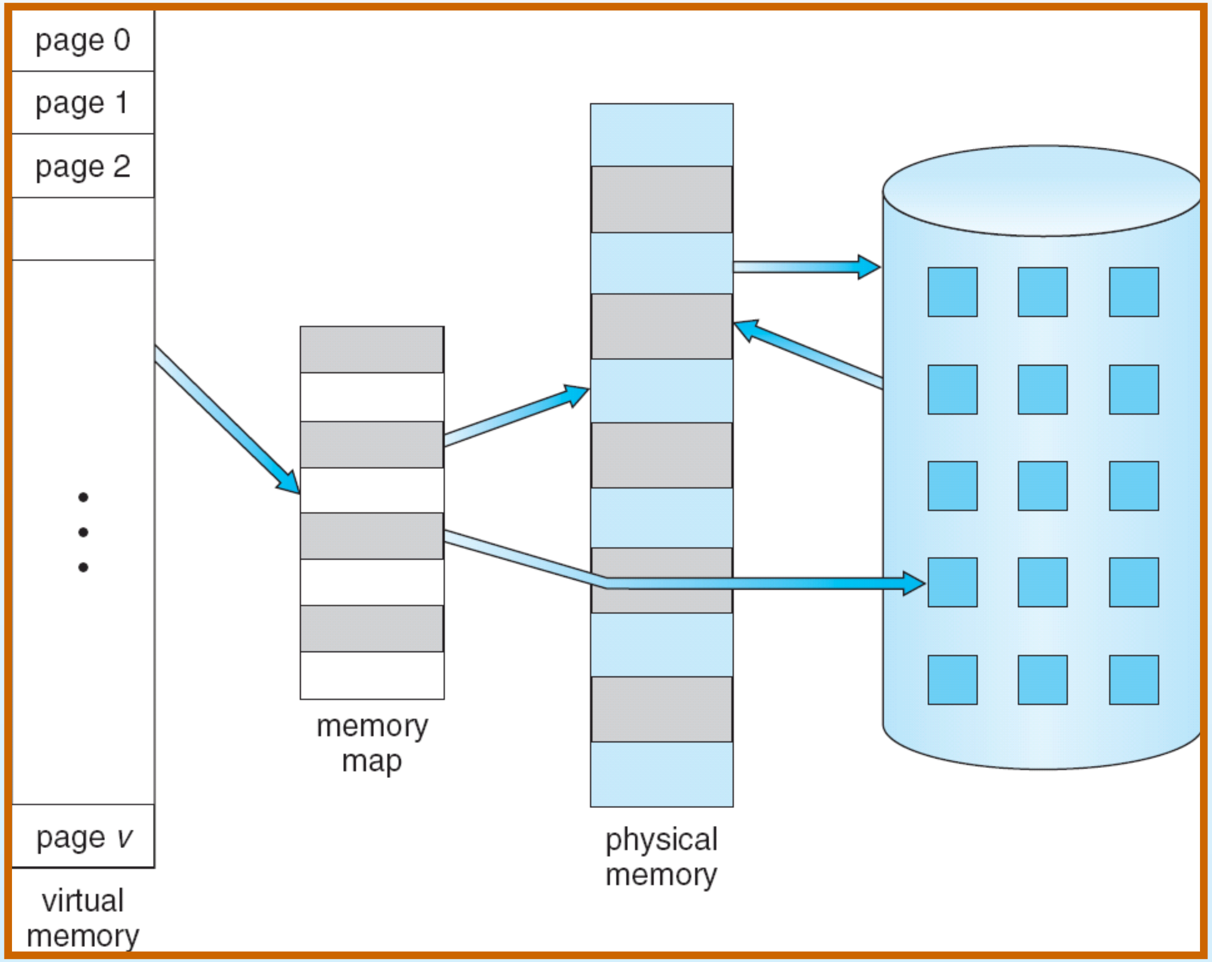
Objectives

- To describe the benefits of a virtual memory system
- To explain the concepts of
  - demand paging
  - page-replacement algorithms
  - allocation of page frames
- To discuss the principle of the working-set model

Background

- Virtual Memory
  - Only part of the program needs to be in memory
  - Logical address space can be much larger than physical address space
    - 0 to 0xFFFFFFFF...
  - Increase CPU utilization and throughput
    - No affect on response time and turnaround time
  - Free the programmer!
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Virtual Memory That is Larger Than Physical Memory

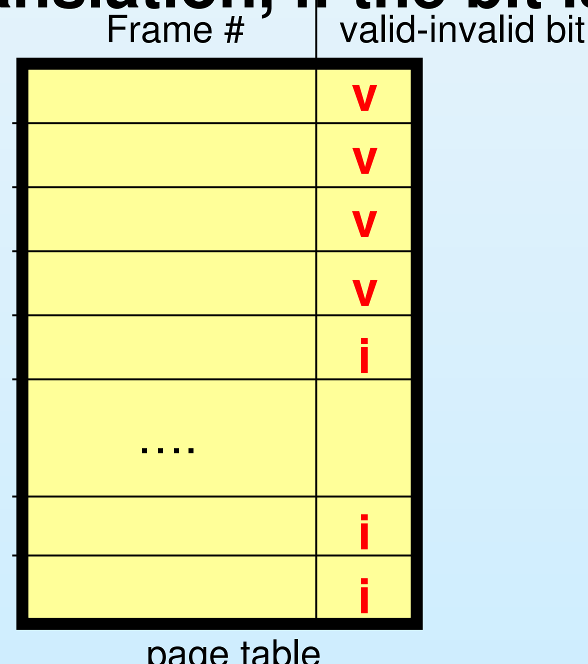


Demand Paging

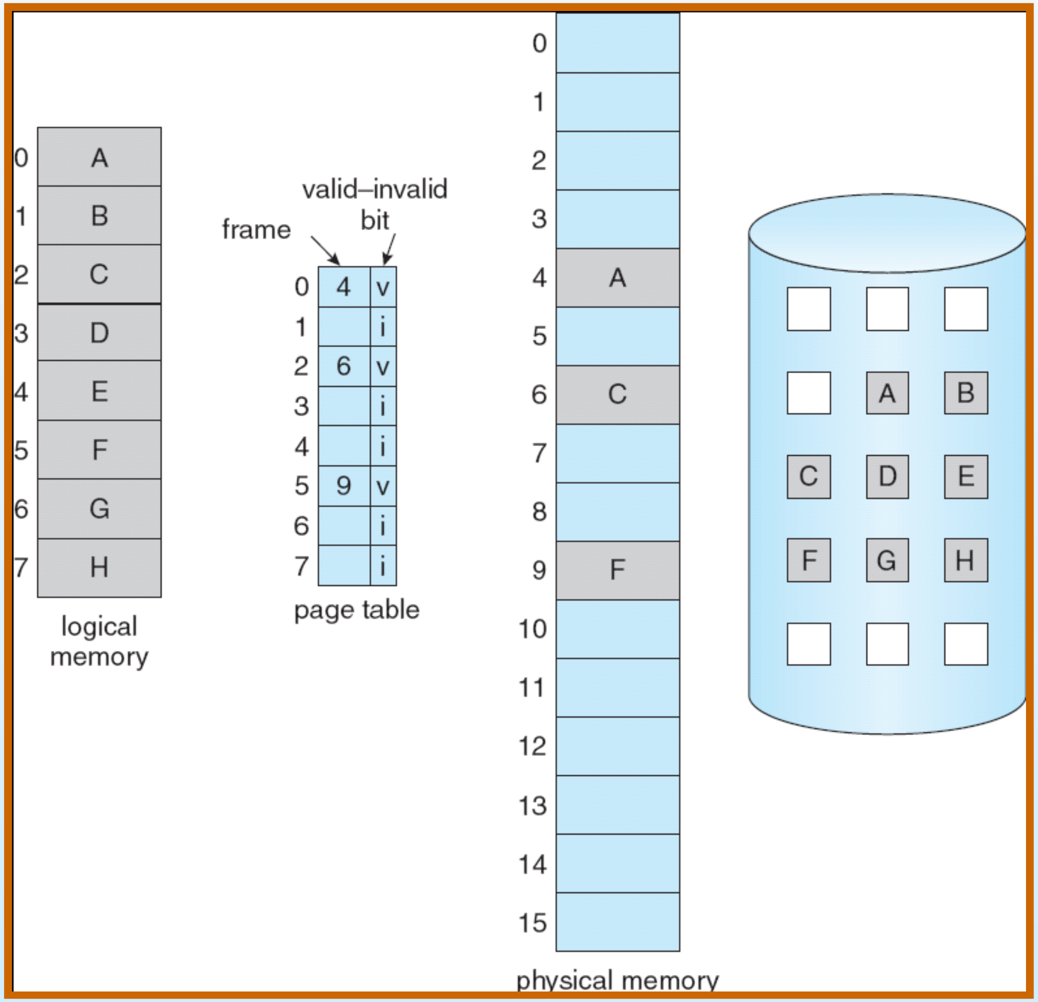
- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More processes
- Page is needed $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory
- Lazy swapper
  - Never swaps page into memory unless it'll be needed
  - Swapper that deals with pages is a pager

Valid-Invalid Bit

- A valid–invalid bit is included in each page table entry
  - v – in-memory
  - i – not-in-memory
- Initially the bit is set to i on all entries
- During address translation, if the bit is i, PAGE FAULT!



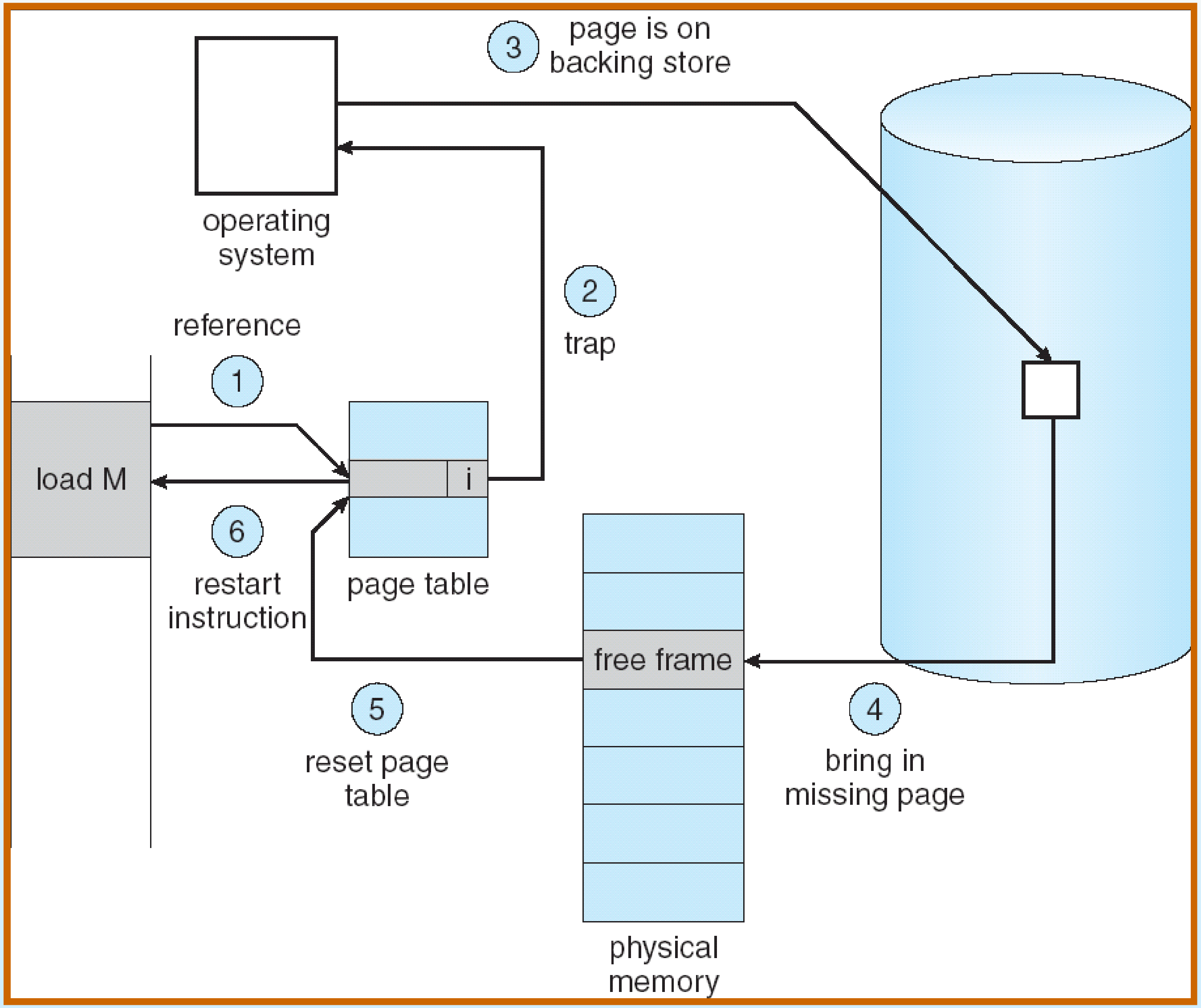
Page Table When Some Pages Are Not in Main Memory



Page Fault

- First reference to an invalid page will trap to OS
  - page fault
- OS looks at another table to decide:
  - Invalid reference ⇒ abort
  - Just not in memory
- Get empty frame
- Swap page into frame
- Reset tables
- Set validation bit = v
- 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

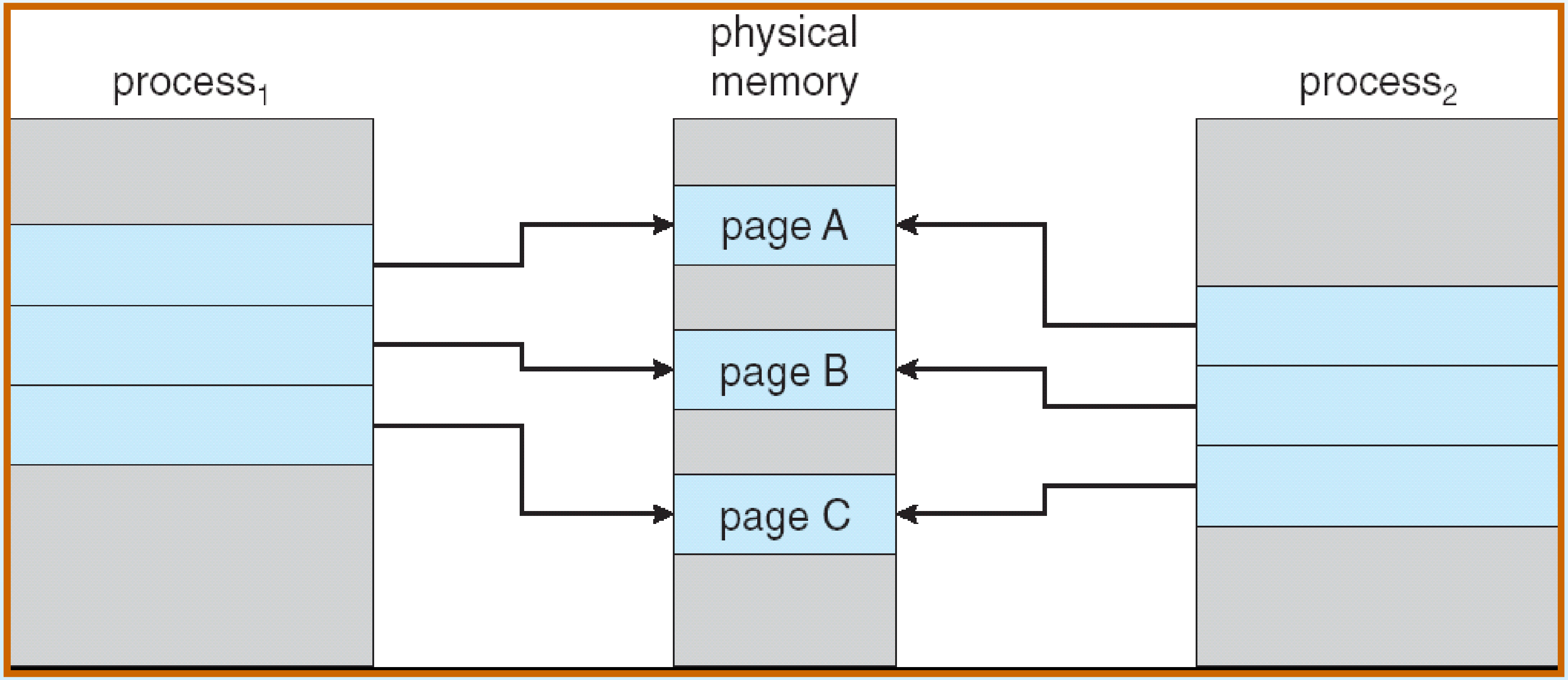
- If page fault rate = 1/1000, then EAT = 8.2 ms

  This is a slowdown by a factor of 40!!

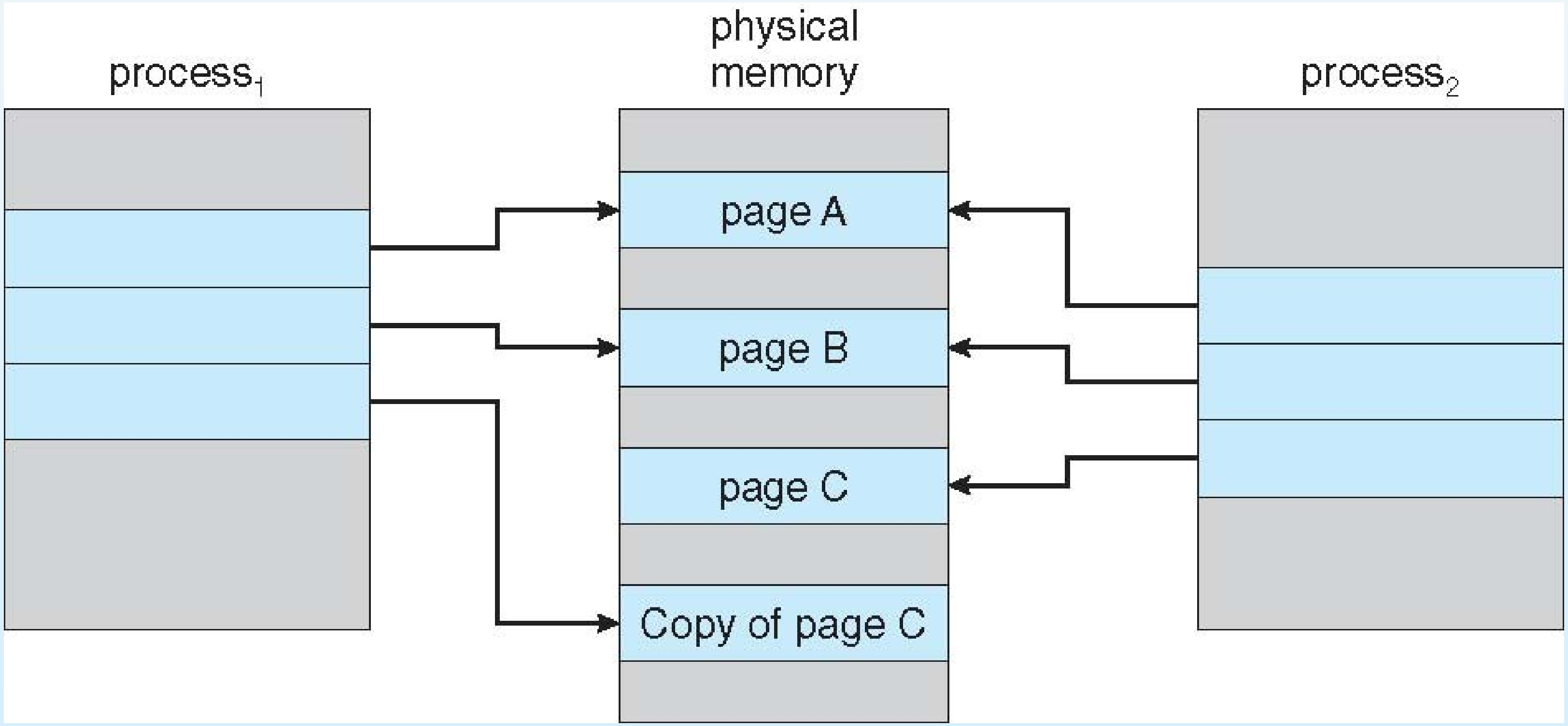
Copy-on-Write

- Copy-on-Write (COW)
  - Both parent and child processes initially *share* the same pages in memory
  - If either process modifies a page, the page is copied
- COW allows more efficient process creation
- Free pages are allocated from a pool of zeroed-out pages

Before Process 1 Modifies Page C



After Process 1 Modifies Page C



What happens if there is no free frame?

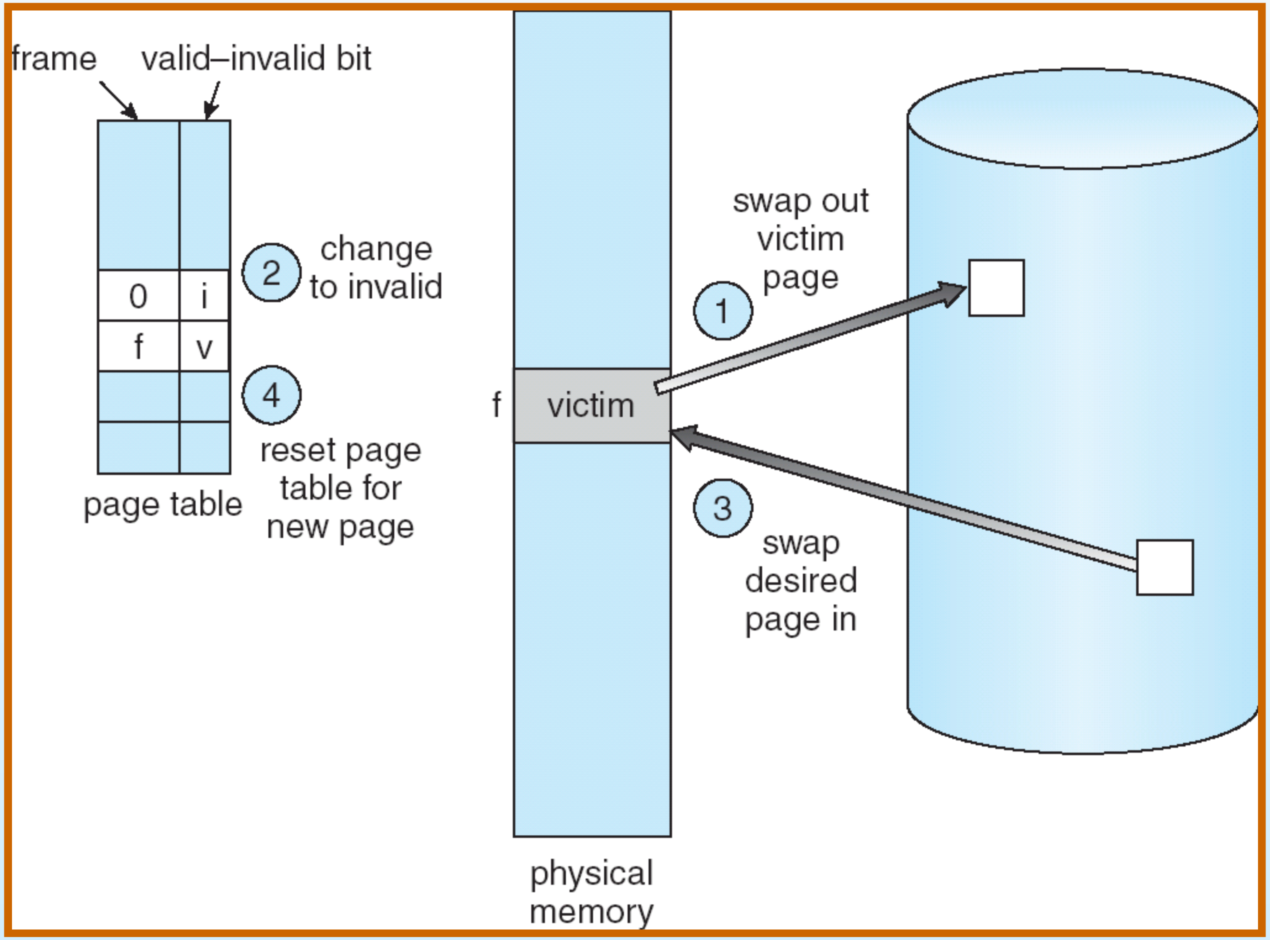
- Page replacement
  - Find some page in memory, but not really in use, swap it out
  - Algorithm
  - Performance – minimize number of page faults

- Same page may be brought into memory several times

Basic Page Replacement

- Find the location of the desired page on disk
- 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
  - Swap out the victim if its modified (dirty) bit is set
- Bring the desired page into the (newly) free frame
- Update the page and frame tables
- Continue to run the process

Page Replacement

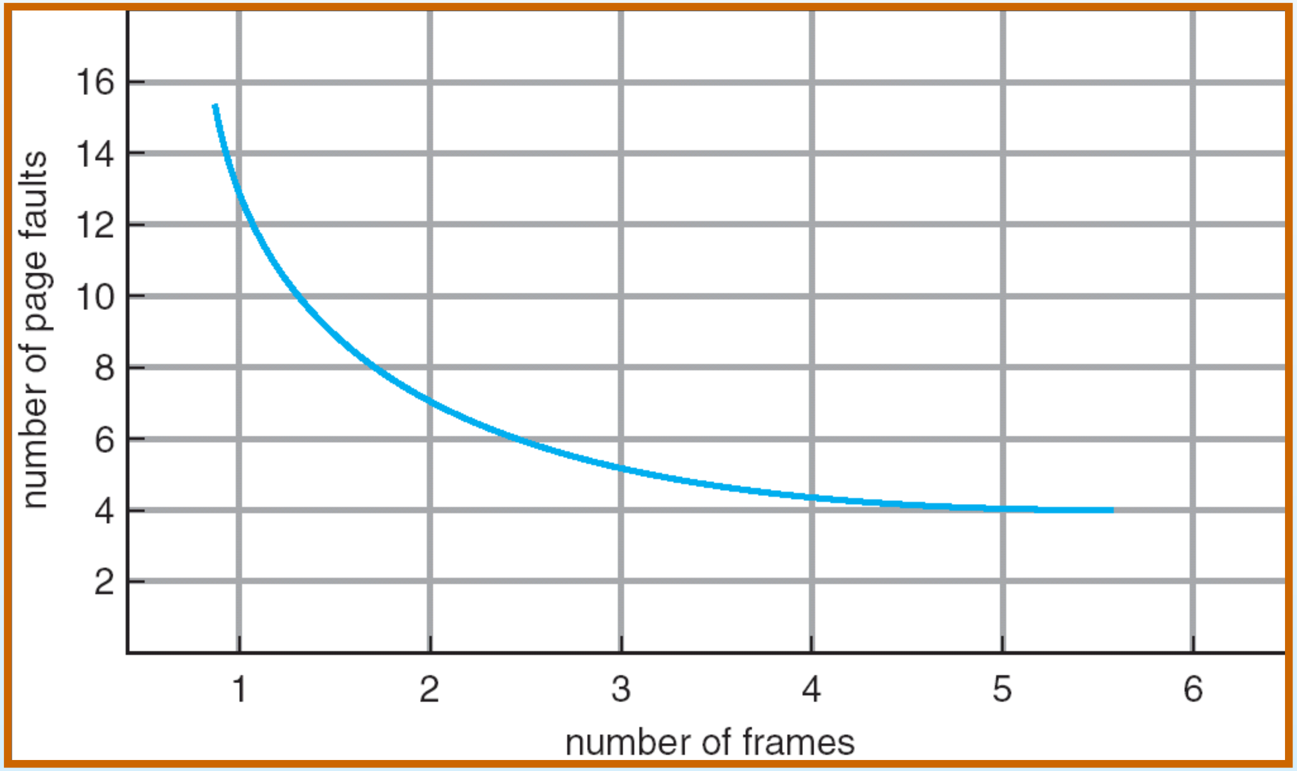


Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm
  - 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

- **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

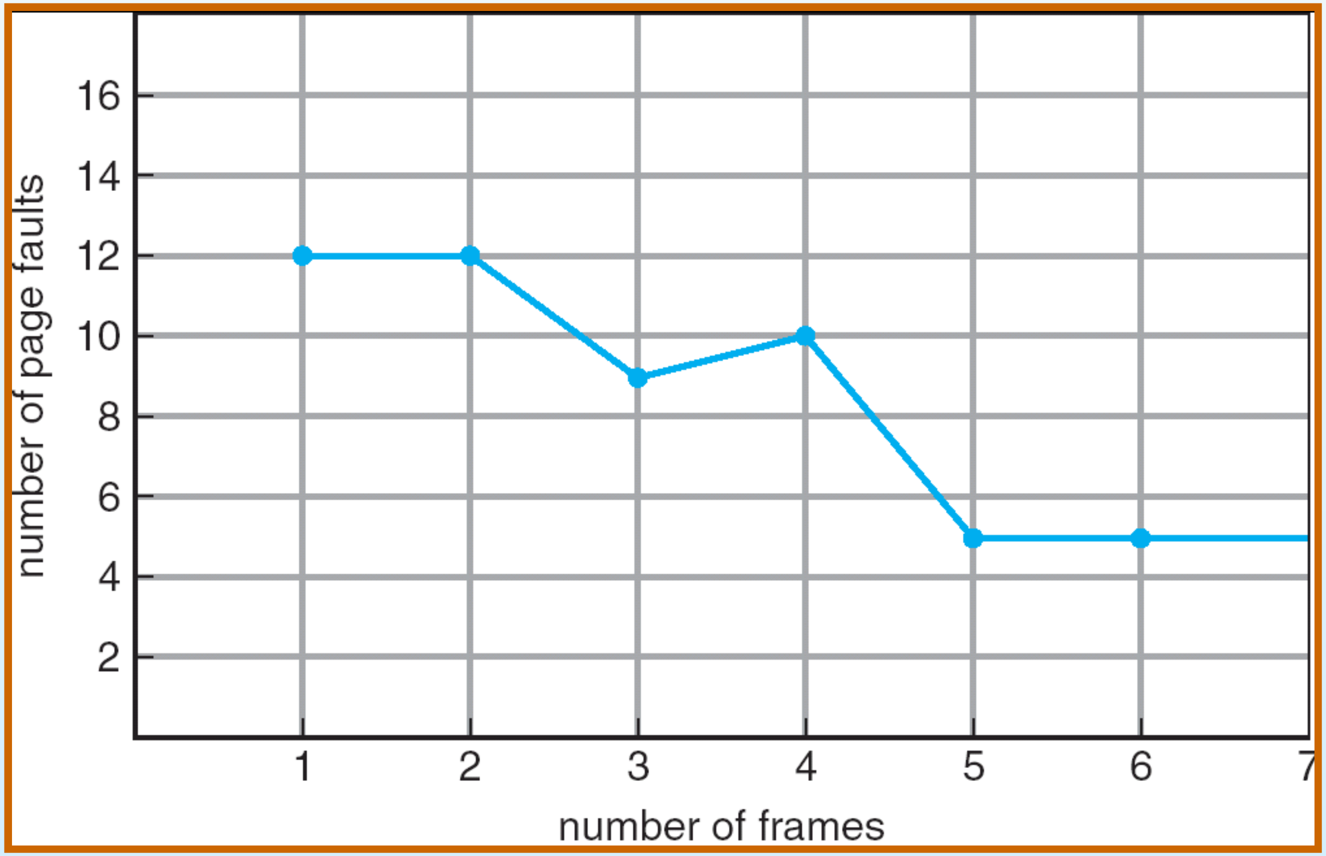
- **4 frames**

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

10 page faults

- **Belady's Anomaly: more frames -> more page faults**

FIFO Illustrating Belady's Anomaly



Optimal Algorithm

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

1	4	
2		6 page faults
3		
4	5	

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

Least Recently Used (LRU) Algorithm

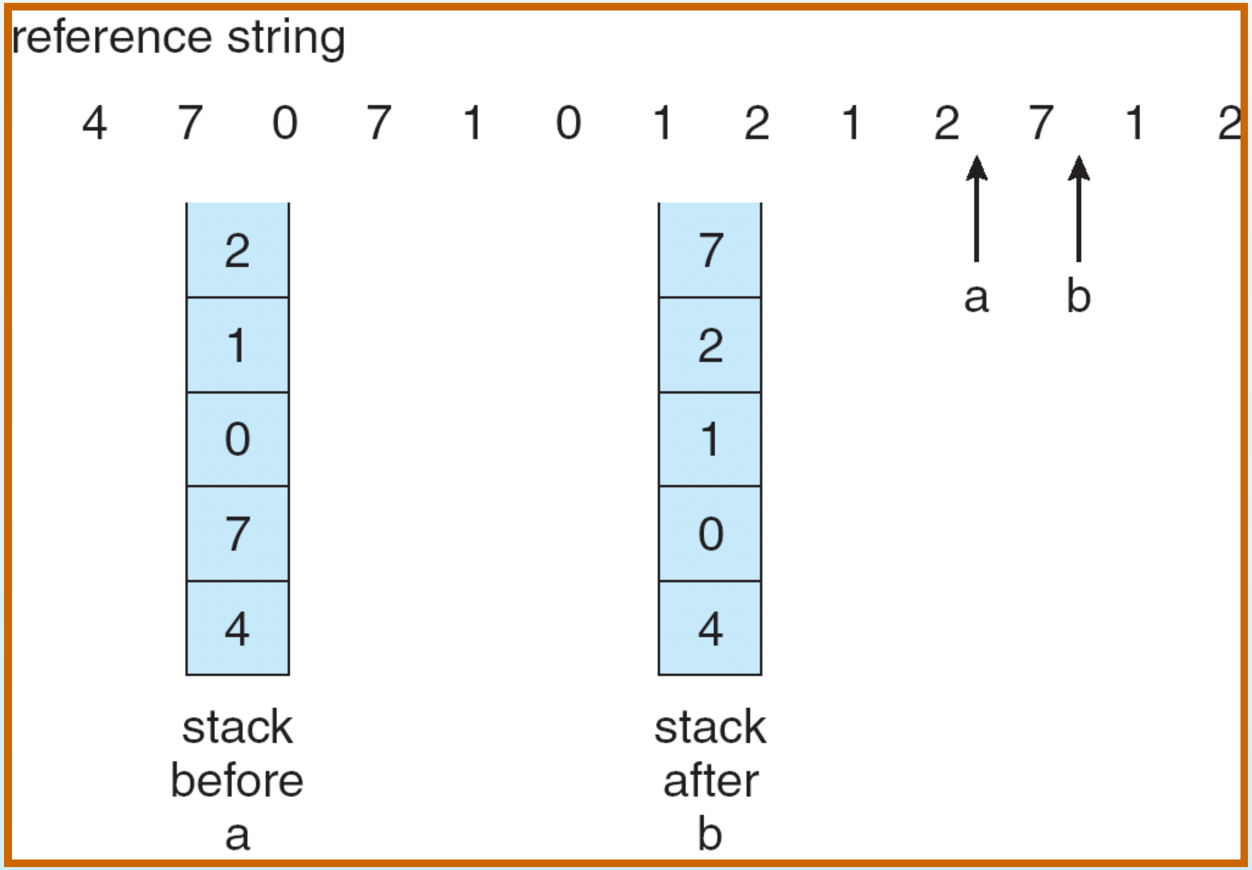
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

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

LRU Algorithm Implementation

- Counter implementation
  - Every page entry has a counter
  - Every time page is referenced, copy the clock into the counter
  - Search for the page with smallest counter
  - Replace it
- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

Use Of A Stack to Record The Most Recent Page References



LRU Approximation Algorithms

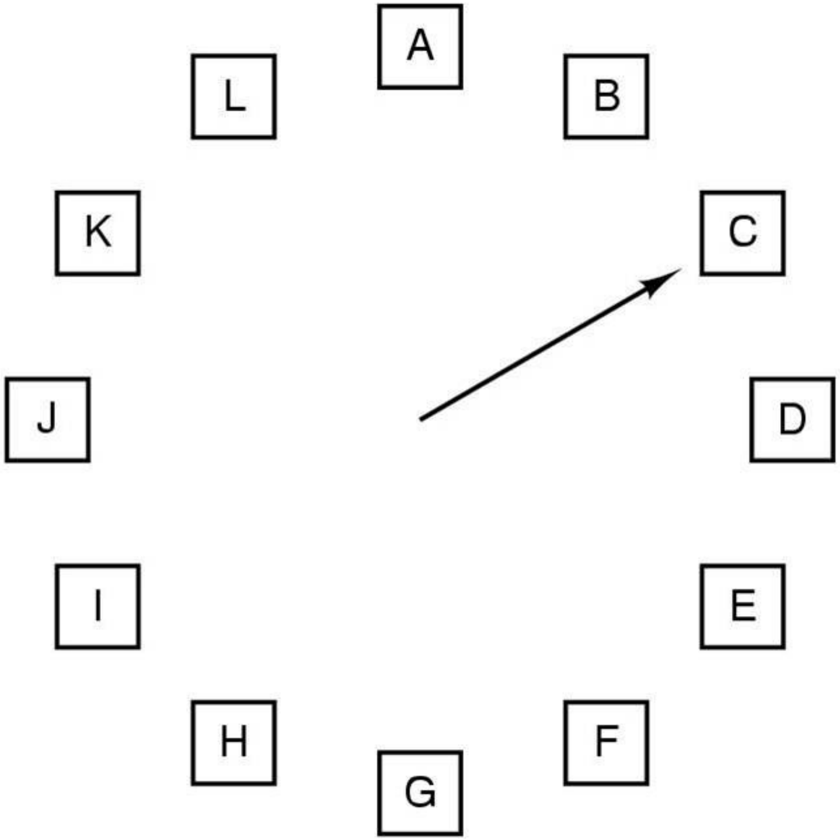
- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced, CPU set bit to 1
  - Replace the one which is 0 (if one exists)
  - We do not know the order, however

Additional-Reference-Bits Algorithm

	(a)	(b)	(c)	(d)	(e)
R bits for pages 0-5	clock tick 0	clock tick 1	clock tick 2	clock tick 3	clock tick 4
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0

Page	(a)	(b)	(c)	(d)	(e)
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00100000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000

Second-Chance (clock) Page-Replacement Algorithm



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

- R = 0: Evict the page
- R = 1: Clear R and advance hand

Enhanced Second-Chance Algorithm

- (reference bit, modify bit)
  - (0, 0) – neither recently used, nor modified
  - (0, 1) – not recently used, but modified
  - (1, 0) – recently used, but clean
  - (1, 1) – recently used and modified
- In which order to evict?

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- Least frequently used (LFU) Algorithm
  - Replaces page with smallest count
- Most frequently used (MFU) Algorithm
  - The page with the smallest count was probably just brought in and has yet to be used

Page Buffering Algorithms

- OS keeps a pool of free frames
  - Allocate from the pool
  - Swap out a frame lately and add it to the pool
- When system is idle, write back the modified pages
- If a requested page is in the pool, get it immediately

Global vs. Local Allocation

- Global replacement
  - Get a replacement frame from the set of all frames
  - One process can take a frame from another
- Local replacement
  - Each process selects from only its own set of allocated frames
- Thrashing
  - A process is busy swapping pages in and out

Locality Model

- To prevent thrashing, provide a process with as many frames as it needs.

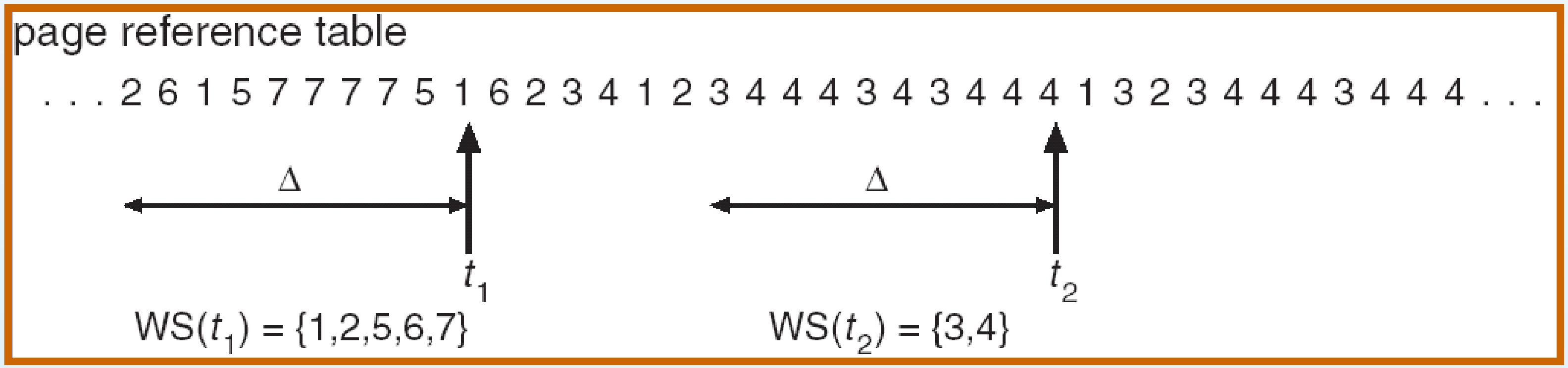
- A locality is a set of pages that are actively used together

- Processes migrates from one locality to another locality always

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 P_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
- $D = \Sigma\ WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if $D >$ m, then suspend one of the processes

Working-set model

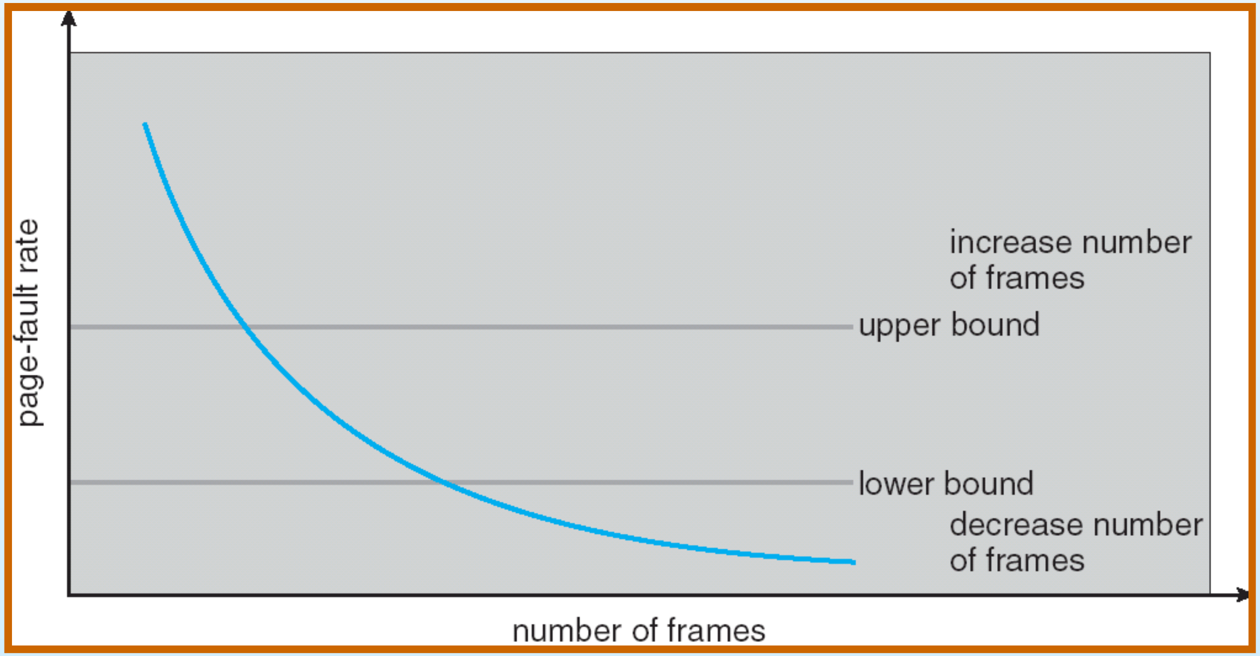


Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: $\Delta = 10{,}000$
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set
- Improvement = 10 bits and interrupt every 1000 time units

Page-Fault Frequency Scheme

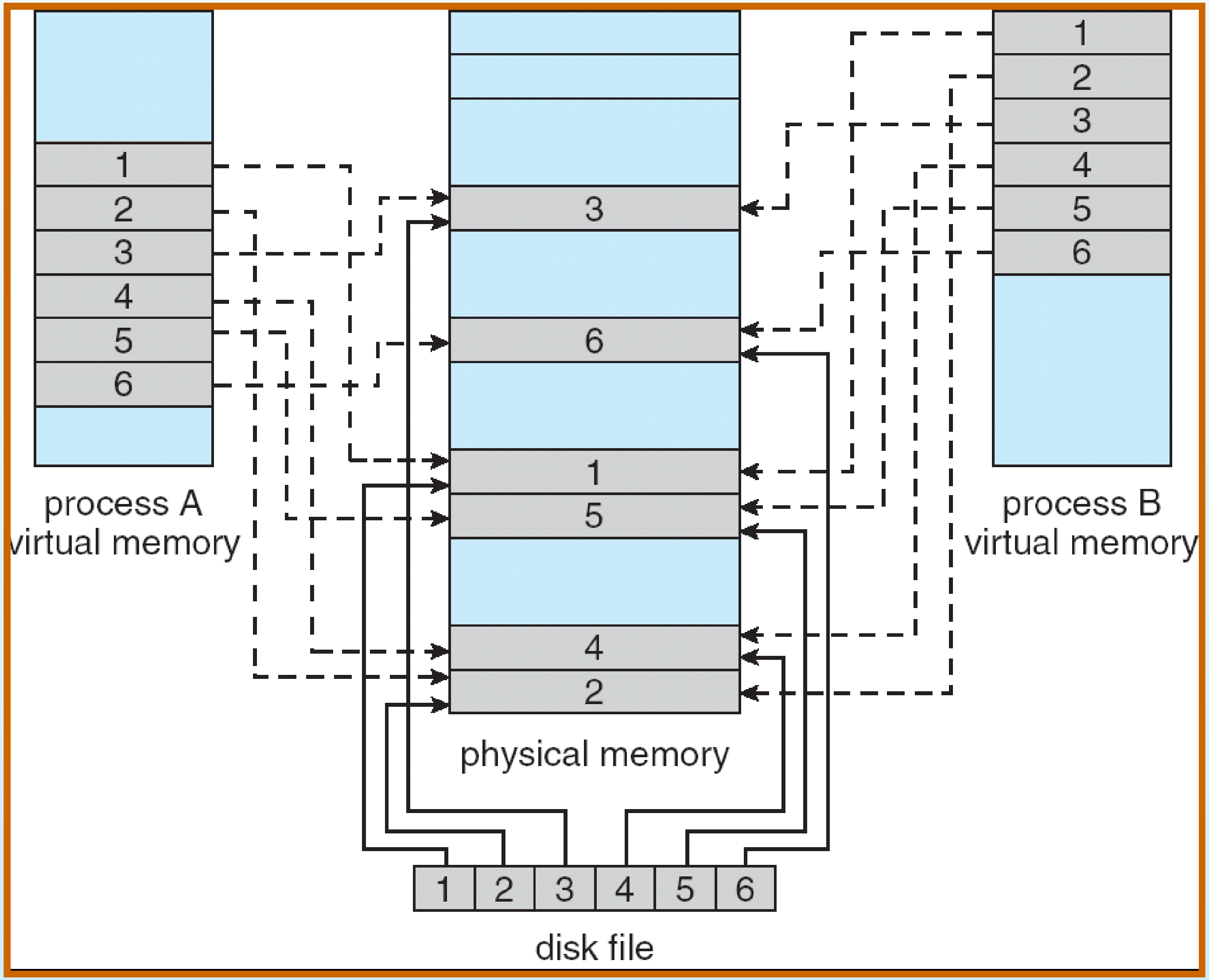
- Establish "acceptable" page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



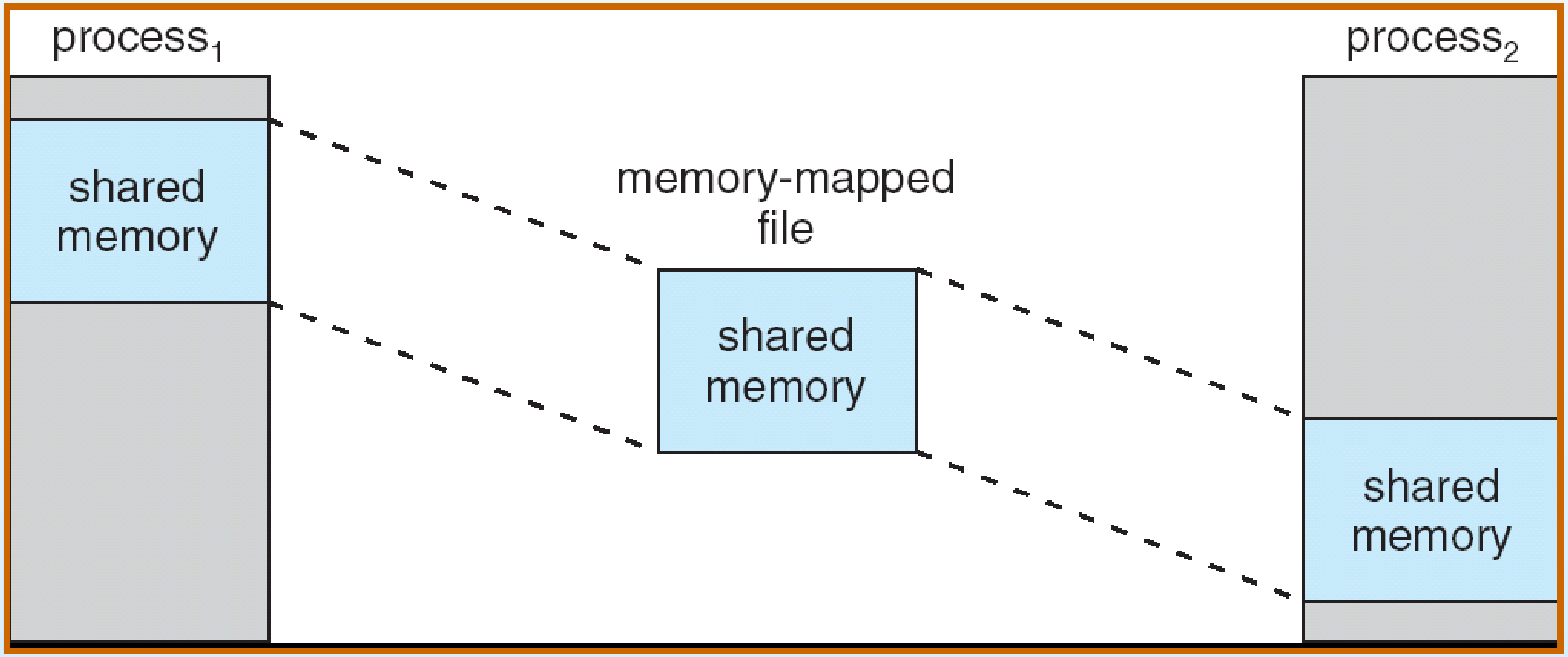
Memory-Mapped Files

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



Memory-Mapped Shared Memory in Windows



Other Issues – Program Structure

- Program structure
  - `int data[1024][1024];`
  - Each row is stored in one page
  - Program 1

```
for (j = 0; j < 1024; j++)
    for (i = 0; i < 1024; i++)
        data[i][j] = 0;
```

  - Program 2

```
for (i = 0; i < 1024; i++)
    for (j = 0; j < 1024; j++)
        data[i][j] = 0;
```

Other Issues – Program Structure(Cont.)

- Array vs. List/Hash table
- Local variable, global variable vs. heap variable
- Align to page edge
- Compiler and loader can affect locality

Other Issues – I/O interlock

- DMA from a device
  - DMA to kernel space, then copy to user space
  - lock the frame. It will not be replaced until unlock

Operating System Examples

- Windows XP
- Solaris

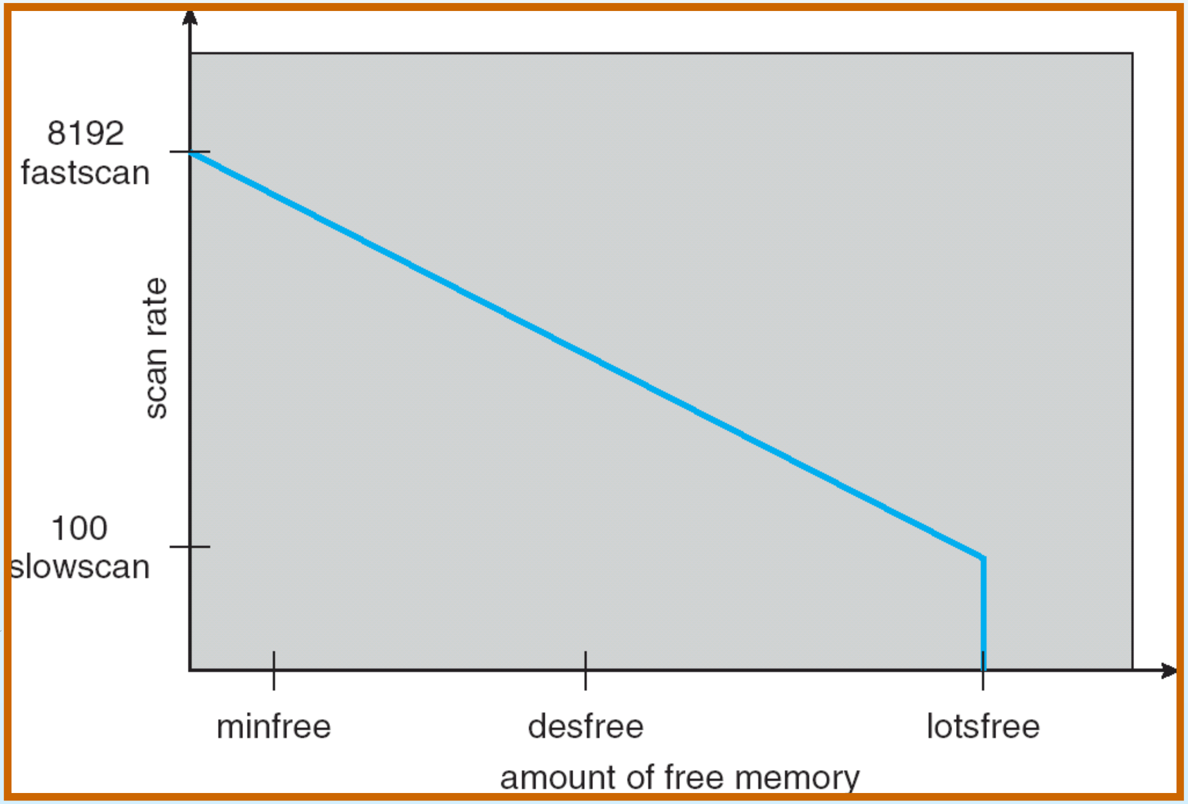
Windows XP

- Uses demand paging with clustering.
- Clustering brings in pages following the faulting page.
- Processes are assigned working set minimum and working set maximum
- A process may be assigned as many pages up to its working set maximum
- When the amount of free memory falls below a threshold, automatic working set trimming is performed
- Working set trimming removes pages from processes that have pages in excess of their working set minimum
  - Using clock algorithm

Solaris

- Maintains a list of free pages
- *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
- Pageout scans pages using modified clock algorithm
  - Two hands
- *Scanrate* is the rate at which pages are scanned. 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