



Operating System Concepts

Che-Wei Chang

chewei@mail.cgu.edu.tw

Department of Computer Science and Information
Engineering, Chang Gung University

Contents

- 1. Introduction
- 2. System Structures
- 3. Process Concept
- 4. Multithreaded Programming
- 5. Process Scheduling
- 6. Synchronization
- 7. Deadlocks
- 8. Memory-Management Strategies
- 9. Virtual-Memory Management
- 10. File System
- 11. Implementing File Systems
- 12. Secondary-Storage Systems



Chapter 9. Virtual-Memory Management

Objectives

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

Background

- ▶ Virtual Memory
 - A technique that allows the execution of a process that may not be completely in memory
- ▶ Motivation
 - An entire program in execution may not all be needed at the same time
 - Error handling routines
 - A large array

Virtual Memory

▶ Potential Benefits

- Programs can be much larger than the amount of physical memory
 - Users can concentrate on their problem programming
- The level of multiprogramming increases because processes occupy less physical memory
- Each user program may run faster because less I/O is needed for loading or swapping user programs

▶ Implementation: demand paging

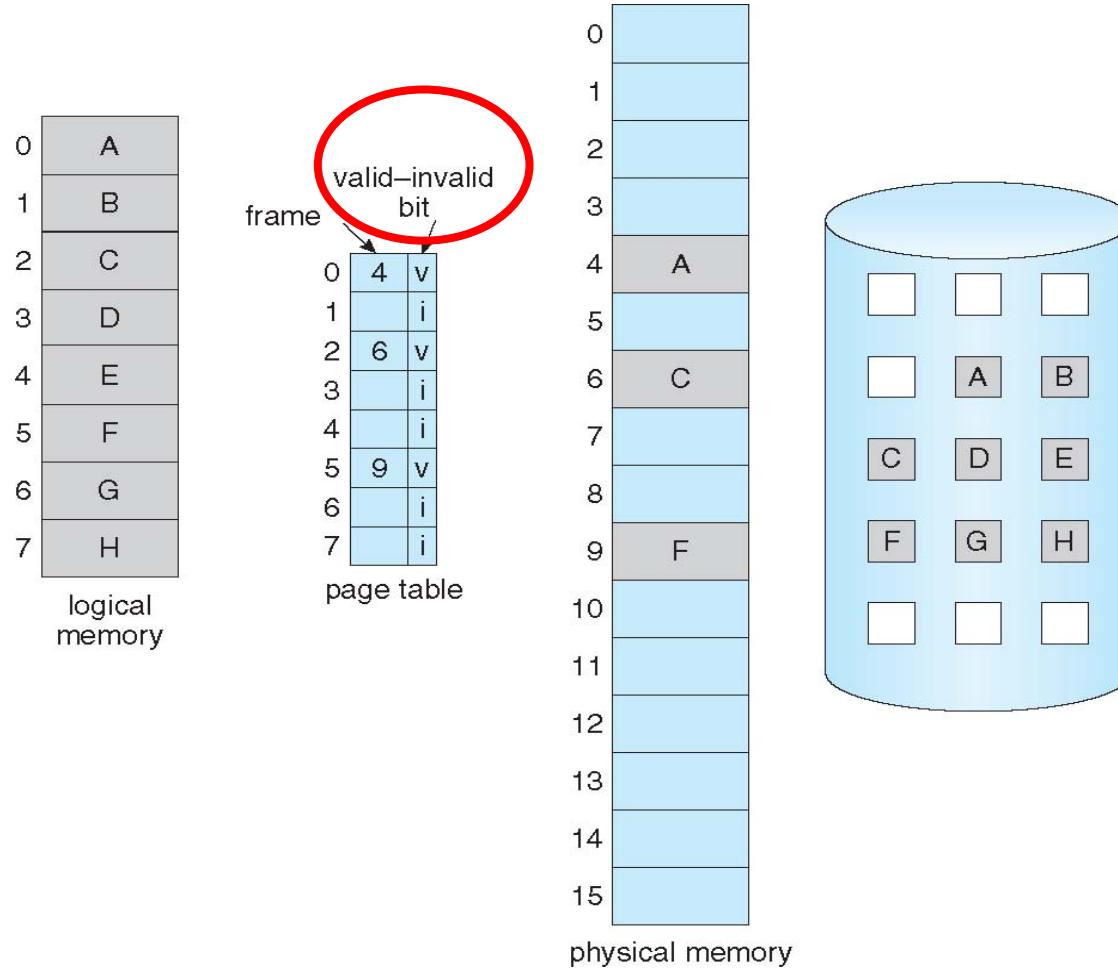
Demand Paging- Lazy Swapper

- ▶ Process image may reside on the backing store
 - Rather than swap in the entire process image into memory **Lazy Swapper** only swaps in a page when it is needed
- ▶ A mechanism is required to recover from the missing of non-resident referenced pages
 - A **Page Fault** occurs when a process references a non-memory-resident page

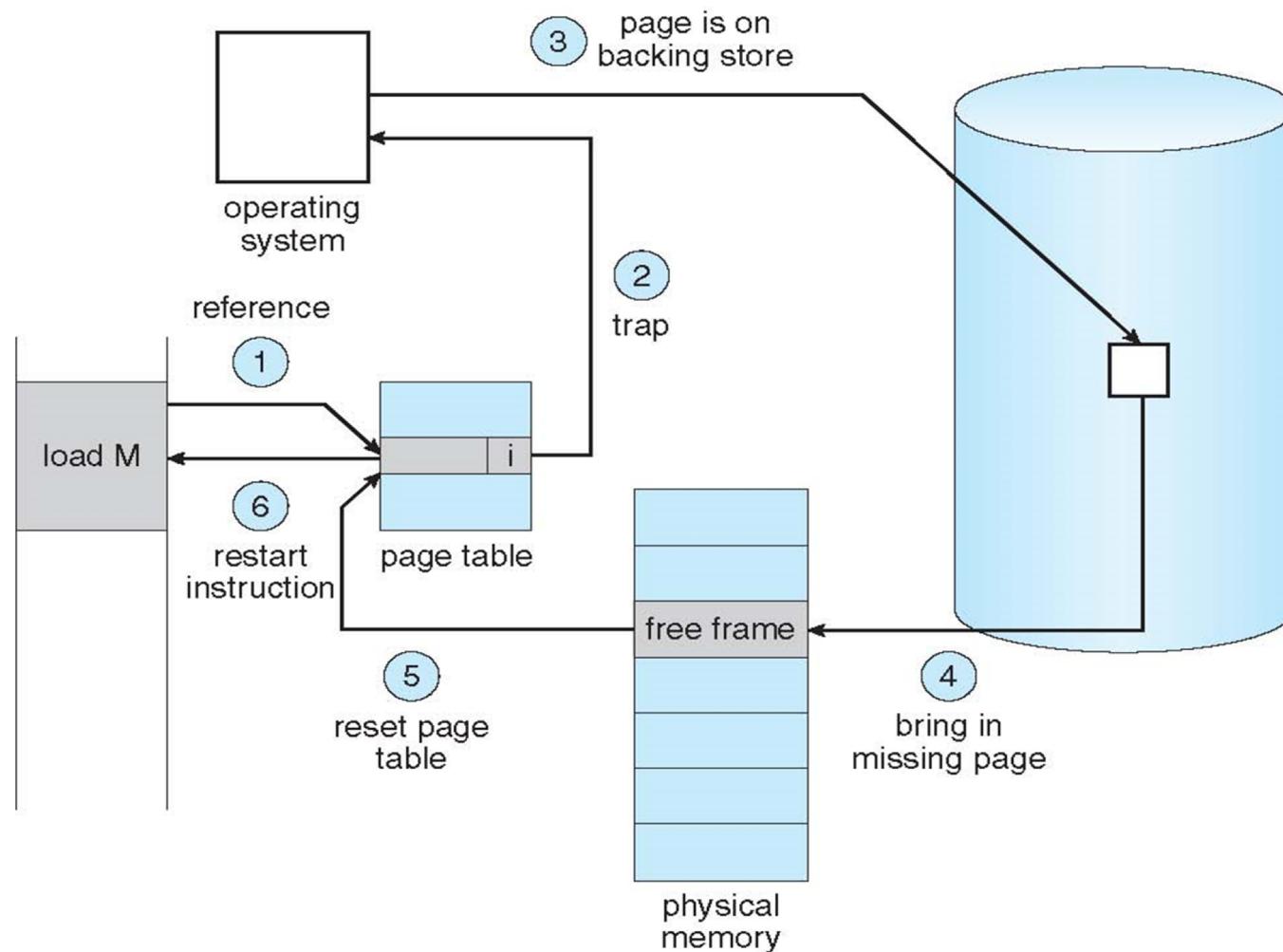
Hardware Support for Demand Paging

- ▶ New bits in the page table
 - To indicate that a page is now in memory or not
- ▶ Secondary storage management
 - Swap space in the backing store
 - A continuous section of space in the secondary storage for better performance

Valid–Invalid Bits



Steps in Handling a Page Fault



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, then the page is copied
- ▶ COW allows more efficient process creation as only modified pages are copied
- ▶ In general, free pages are allocated from a **pool** of **zero-fill-on-demand** pages

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)

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

An Example of Demand Paging

- ▶ Memory access time = 200 nanoseconds
- ▶ Average page-fault service time = 8 milliseconds
- ▶
$$\begin{aligned} \text{EAT} &= (1 - p) \times 200 + p \times 8,000,000 \\ &= 200 + p \times 7,999,800 \end{aligned}$$
- ▶ If one access out of 1,000 causes a page fault, then
 $\text{EAT} = 8.2 \text{ microseconds!}$
- ▶ If we want performance degradation < 10 percent
 - $220 > 200 + 7,999,800 \times p$
 - $p < 0.0000025$

Performance Improvement of Demand Paging

- ▶ Preload processes into the swap space before they start up
- ▶ Preload pages into the main memory before the pages are used
- ▶ Design a good page replacement algorithm

Algorithms for Demand Paging

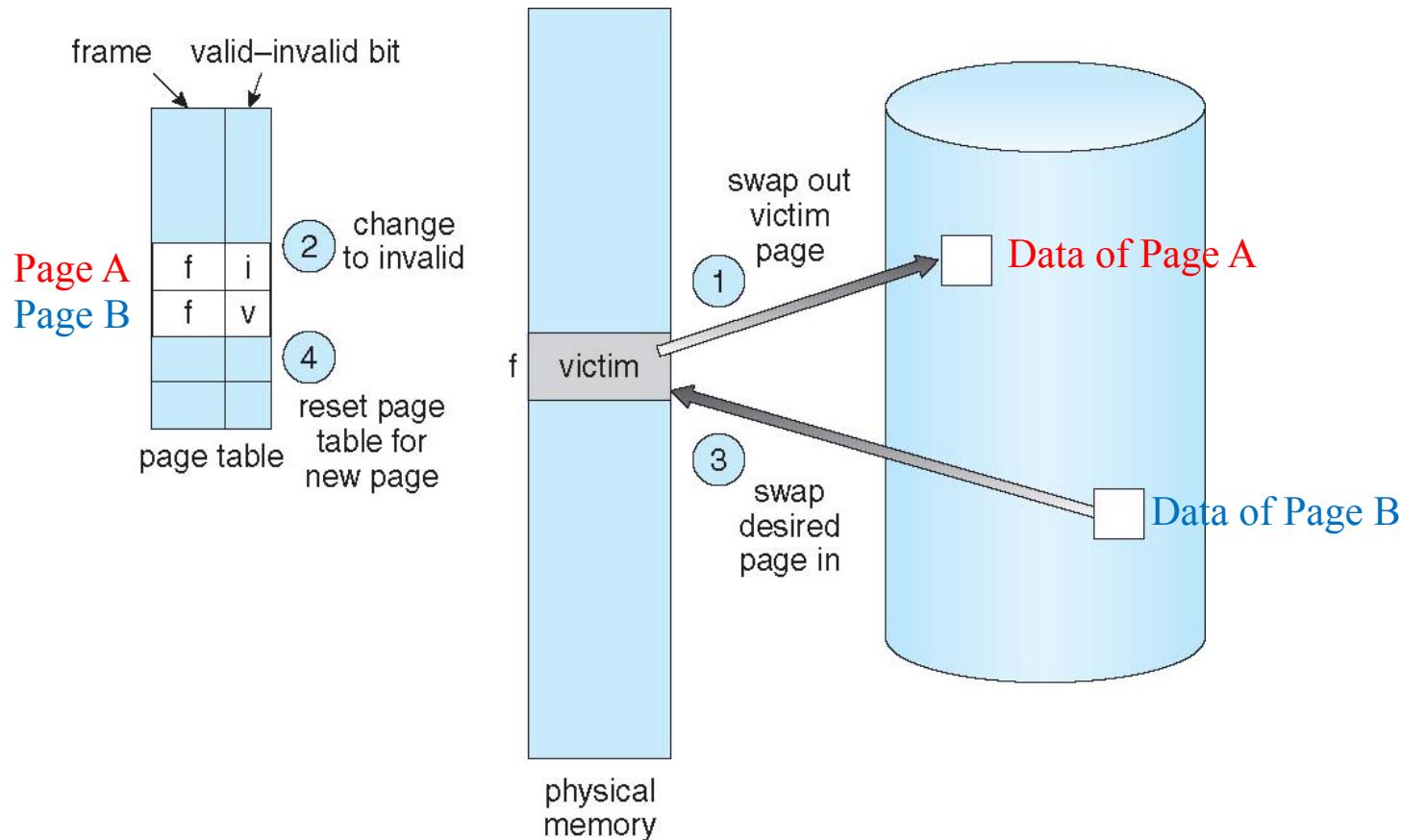
- ▶ Frame Allocation Algorithms
 - How many frames are allocated to a process?
- ▶ Page Replacement Algorithms
 - When page replacement is required, select the frame that is to be replaced!
- ▶ Goal: A low page fault rate!

Page Replacement

- ▶ Demand paging increases the multiprogramming level of a system by “potentially” over-allocating memory
 - Total physical memory = 40 frames
 - Run six processes of size equal to 10 frames
 - Each process currently uses only 5 frames
→ 10 spare frames
- ▶ Most of the time, the average memory usage is close to the physical memory size if we increase a system’s multiprogramming level

Victim Pages

What happens if there is no free frame?



A Page-Fault Service

- ▶ Find the desired page on the disk
- ▶ Find a free frame
 - Select a victim and write the victim page out when there is no free frame
- ▶ Read the desired page into the selected frame
- ▶ Update the page and frame tables, and restart the user process

Page Replacement — FIFO Algorithm

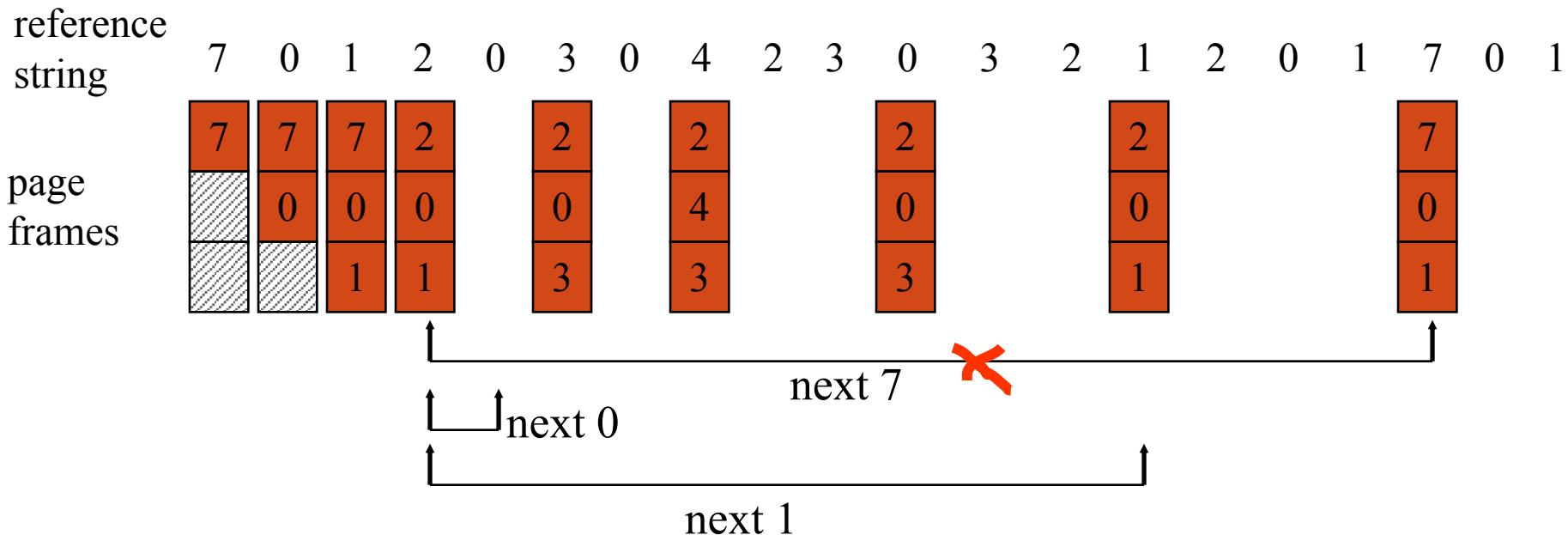
- ▶ First In First Out (FIFO) Implementation
 1. Each page is given a time stamp when it is brought into memory
 2. Select the oldest page for replacement

| reference string | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| page frames | <table border="1"><tr><td>7</td><td>7</td><td>7</td><td>2</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td></td><td></td></tr></table> | 7 | 7 | 7 | 2 | 0 | 0 | 0 | 0 | 1 | 1 | | | <table border="1"><tr><td>2</td><td>2</td><td>4</td><td>4</td><td>4</td><td>0</td></tr><tr><td>3</td><td>3</td><td>3</td><td>2</td><td>2</td><td>2</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>3</td><td>3</td></tr></table> | 2 | 2 | 4 | 4 | 4 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 0 | 0 | 0 | 3 | 3 | <table border="1"><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr><tr><td>3</td><td>2</td></tr></table> | 0 | 0 | 1 | 1 | 3 | 2 | <table border="1"><tr><td>7</td><td>7</td><td>7</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>2</td><td>2</td><td>1</td></tr></table> | 7 | 7 | 7 | 1 | 0 | 0 | 2 | 2 | 1 |
| 7 | 7 | 7 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 2 | 4 | 4 | 4 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 3 | 3 | 2 | 2 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | 0 | 3 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 7 | 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| FIFO queue | 7 | 7 | 7 | 0 | 1 | 2 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 0 | 1 | 2 | 7 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | 0 | 0 | 1 | | 2 | 3 | 0 | 4 | 2 | 3 | | | 0 | 1 | 2 | 7 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |



Page Replacement — Optimal Algorithm

- ▶ Optimality
 - One with the lowest page fault rate
- ▶ Replace the page that will not be used for the longest period of time → It needs future prediction



Page Replacement — Least-Recently-Used Algorithm

- ▶ Least-Recently-Used Algorithm (LRU)
 - We don't have knowledge about the future
 - Thus, we use the history of page referencing in the past to predict the future
- However, it is too expensive to update the time stamp for each memory access!

| reference string | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|------------------|---|---|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| page frames |  7  0 0 0 0 2 0 0 0 3 4 0 0 3 3 0 2 2 3 2 1 3 0 2 7 |  0 0 1 1 3 3 2 2 2 3 3 2 2 2 3 3 2 2 3 3 1 2 0 0 7 |  1 1 0 0 3 3 2 2 2 2 1 2 0 0 1 1 2 2 0 0 1 1 0 0 7 | | | | | | | | | | | | | | | | | |
| LRU queue | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
| | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 0 |
| | 7 | 0 | 1 | 2 | 1 | 2 | 2 | 3 | 0 | 4 | 2 | 2 | 0 | 3 | 3 | 1 | 2 | 0 | 1 | 7 |

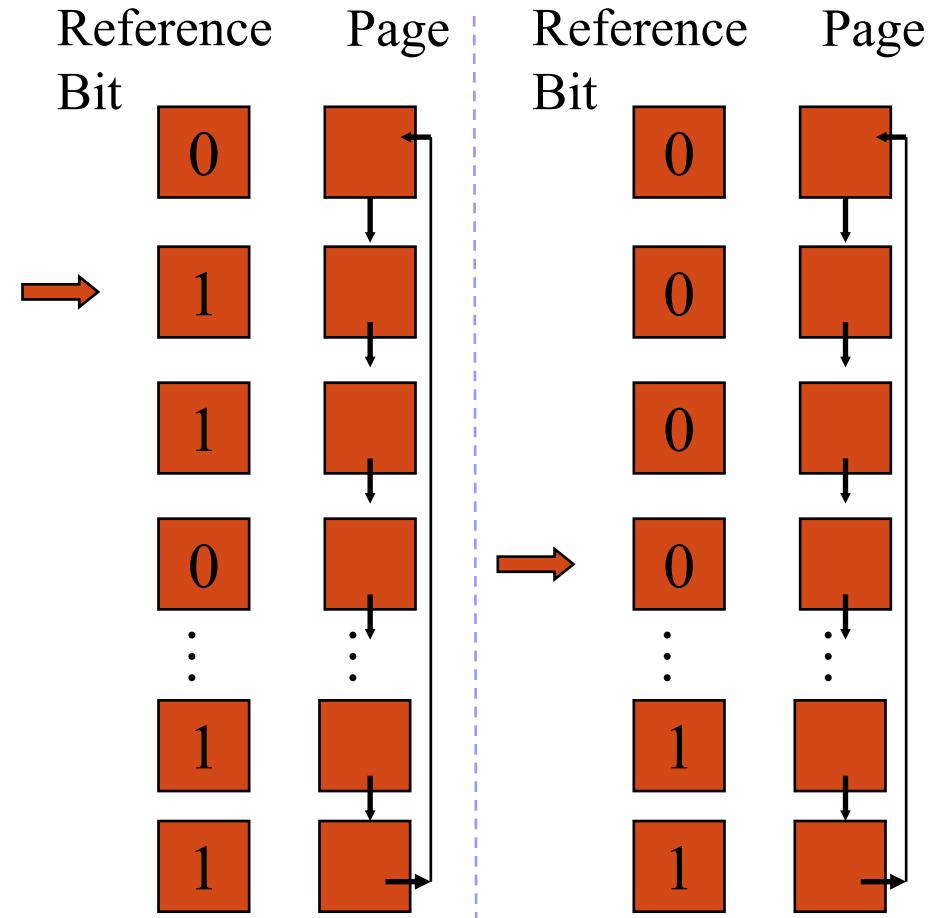
Page Replacement — LRU Approximation Algorithms

▶ Second-Chance Algorithm

- When a page is selected
 - Take it as a victim if its reference bit = 0
 - Otherwise, clear the bit and advance to the next page

▶ Basic Data Structure

- Use a reference bit for each page in memory
- Define a circular FIFO queue of pages



Enhanced Second-Chance Algorithm

- ▶ Considering the reference bit and the modify bit as an ordered pair

| | |
|---------------|---|
| Low Priority | <ul style="list-style-type: none">◦ (0, 0) neither recently used nor modified – best page to replace◦ (0, 1) not recently used but modified – the page will need to be written out before replacement◦ (1, 0) recently used but clean – probably will be used again soon◦ (1, 1) recently used and modified - probably will be used again soon, and the page will need to be written out to disk before it can be replaced |
| High Priority | |

- ▶ We replace the first page encountered in the lowest nonempty class

Counting-Based Algorithms

- ▶ Motivation:
 - Count the number of references made to each page, instead of their referencing times
- ▶ Least Frequently Used Algorithm (LFU)
 - LFU pages are less actively used pages
 - Hazard: Some heavily used pages may no longer be used
 - A Solution – Aging
 - Pages with the smallest number of references are probably just brought in and has yet to be used
- ▶ Most Frequently Used Algorithm (MFU)
- ▶ LFU & MFU replacement schemes can be fairly expensive
- ▶ They do not approximate OPT very well

Page Buffering

- ▶ Basic Idea: to reduce the latency for writing victims out
 - Systems keep a pool of free frames
 - Desired pages are first “swapped in” some frames in the pool
 - When the selected page (victim) is later written out, its frame is returned to the pool
- ▶ Basic Approach
 - Maintain a list of modified pages
 - Whenever the paging device is idle, a modified page is written out and reset its “modify bit”
 - The clean pages can be included in the pool

Allocation of Frames (1 / 2)

- ▶ Each process needs minimum number of frames
- ▶ 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*
- ▶ Maximum of course is total frames in the system
- ▶ Fixed allocation
 - Use a formula to derive the number of required frames for each application
- ▶ Dynamic allocation
 - Measure some behavior, e.g. page fault rated, to know the needs of applications

Allocation of Frames (2/2)

- ▶ Global Allocation
 - Processes can take frames from others
 - For example, high-priority processes can increase its frame allocation at the expense of the low-priority processes
- ▶ Local Allocation
 - Processes can only select frames from their own allocated frames
 - The set of pages in memory for a process is affected by the paging behavior of only that process

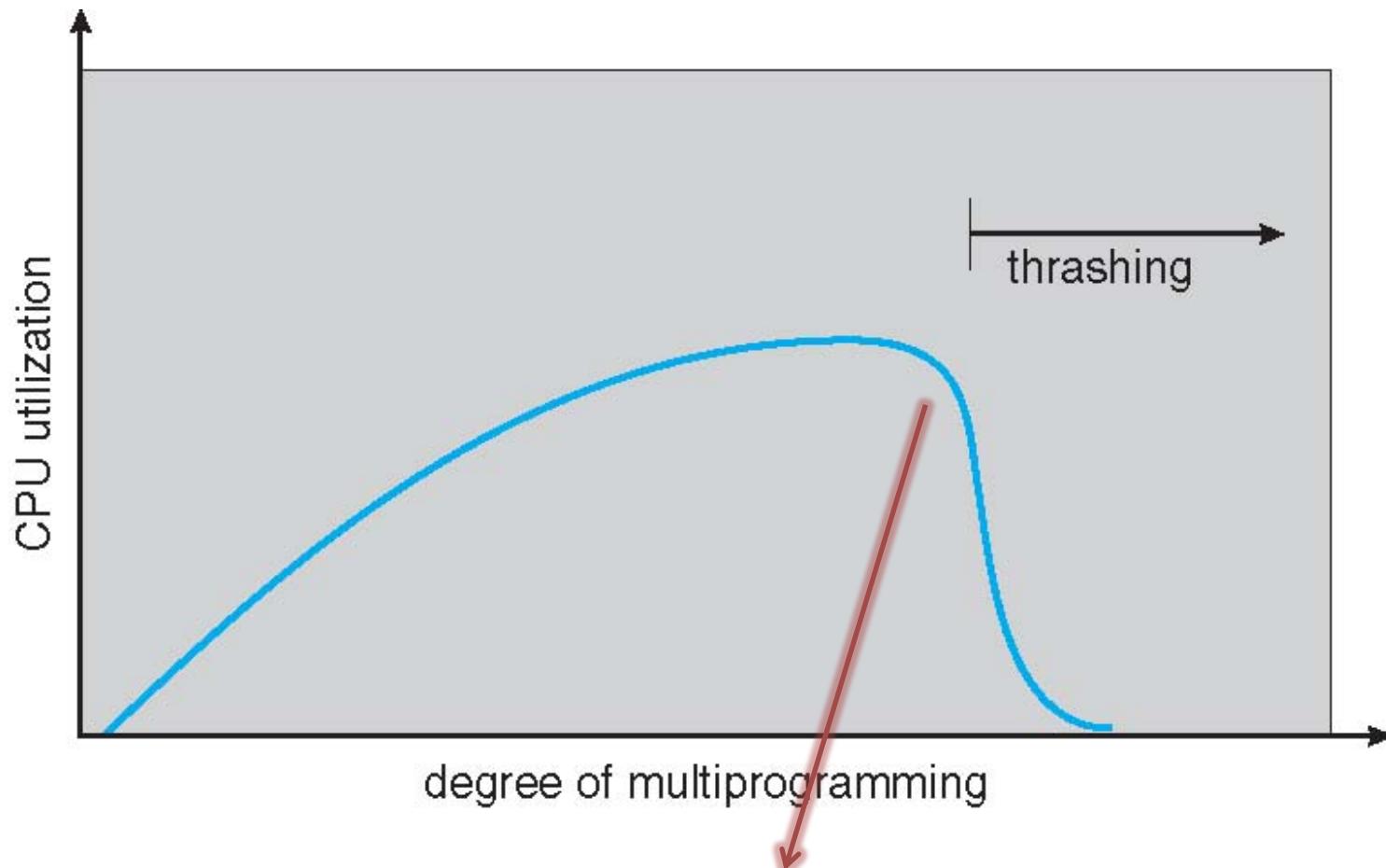
Non-Uniform Memory Access

- ▶ Many systems are NUMA – speed of access to memory varies
 - Consider system boards containing CPUs and memory, interconnected over a system bus
- ▶ Optimal performance comes from allocating memory “close to” the CPU on which the thread is scheduled
 - Modifying the scheduler to schedule the thread on the same CPU when possible

Thrashing (1 / 2)

- ▶ If a process does not have “enough” memory frames, the page-fault rate is very high
 - Page fault to get pages into memory frames
 - Replace existing pages in frames
 - But soon need to get the replaced pages back
 - This leads to:
 - Low CPU utilization
 - Operating system is then thinking that it needs to increase the degree of multiprogramming
 - Another processes are added to the system
 - More page faults
- ▶ **Thrashing** → Process is busy swapping pages in and out

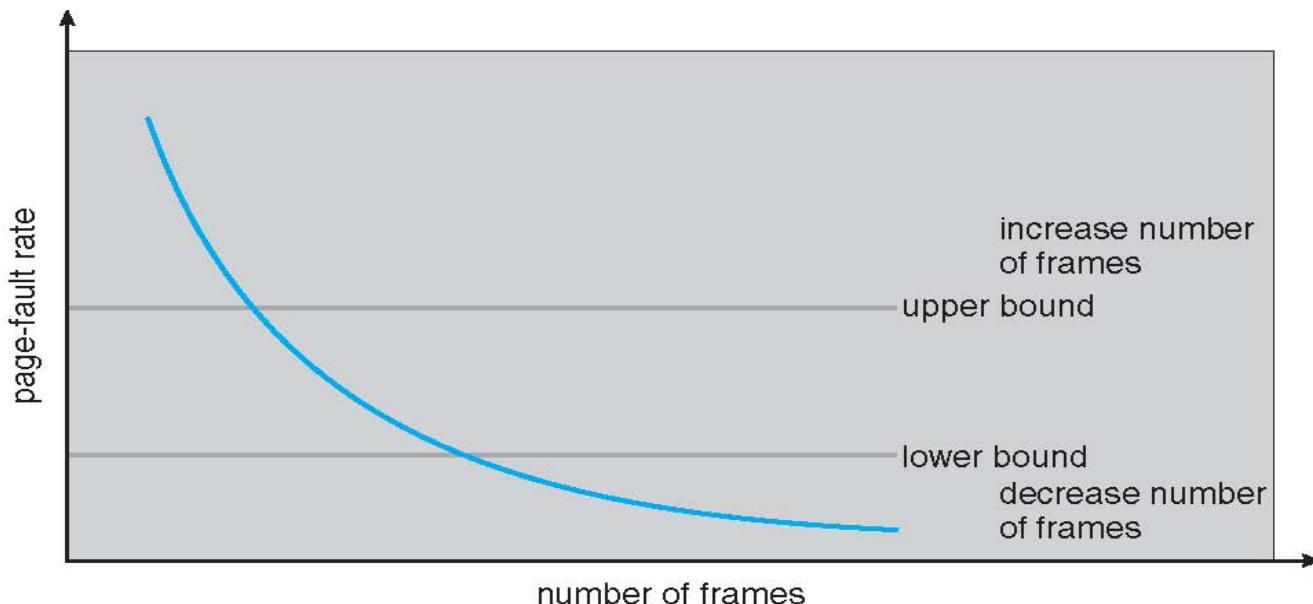
Thrashing (2/2)



Be careful of the page fault rate

Page-Fault Frequency

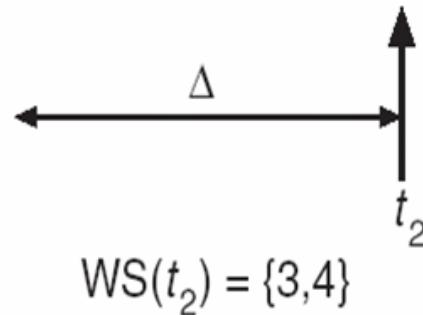
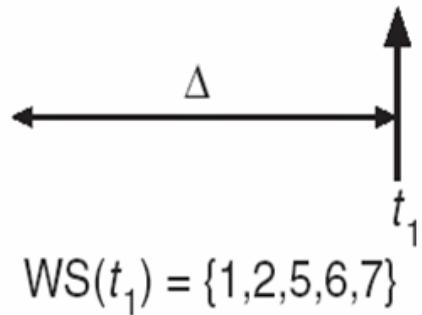
- ▶ Establish “acceptable” page-fault frequency rate and use local replacement policy
 - Control thrashing directly through the observation on the page-fault rate
 - If actual rate too low, process loses frame
 - If actual rate too high, process gains frame



Working-Set Model (1 / 2)

page reference table

... 2 6 1 5 7 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 ...

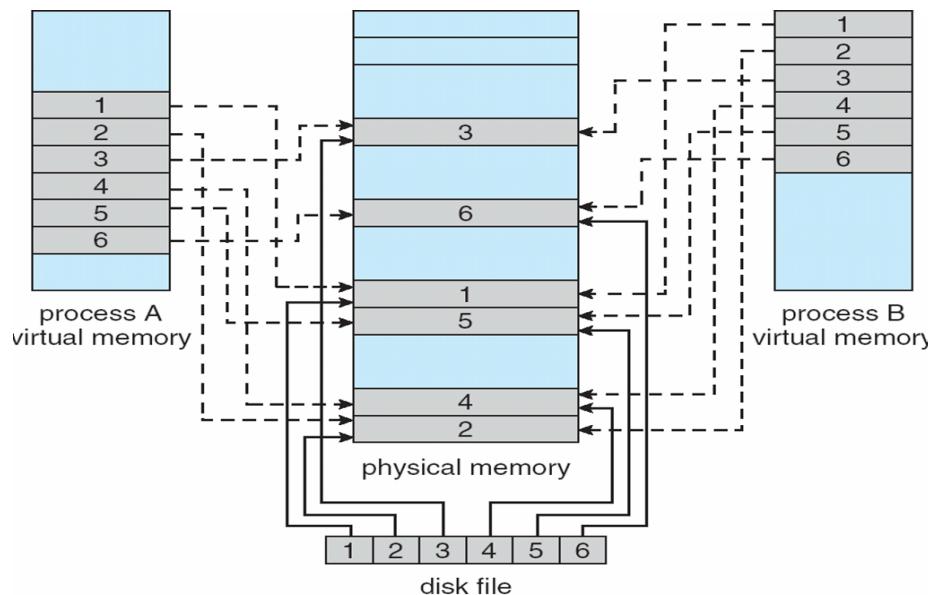


Working-Set Model (2/2)

- ▶ $\Delta \equiv$ a working-set window \equiv a fixed number of page references
 - Example: 10,000 instructions
- ▶ WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ
 - if Δ is too small: will not encompass entire locality
 - if Δ is too large: will encompass several localities
 - if $\Delta = \infty$: will encompass entire program
- ▶ $D = \sum WSS_i \equiv$ total demand frames
 - Approximation of locality
- ▶ if $D > \text{the number of frames} \rightarrow$ Thrashing

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
 - But when does written data make it to disk?
 - Periodically and/or at file `close()` time

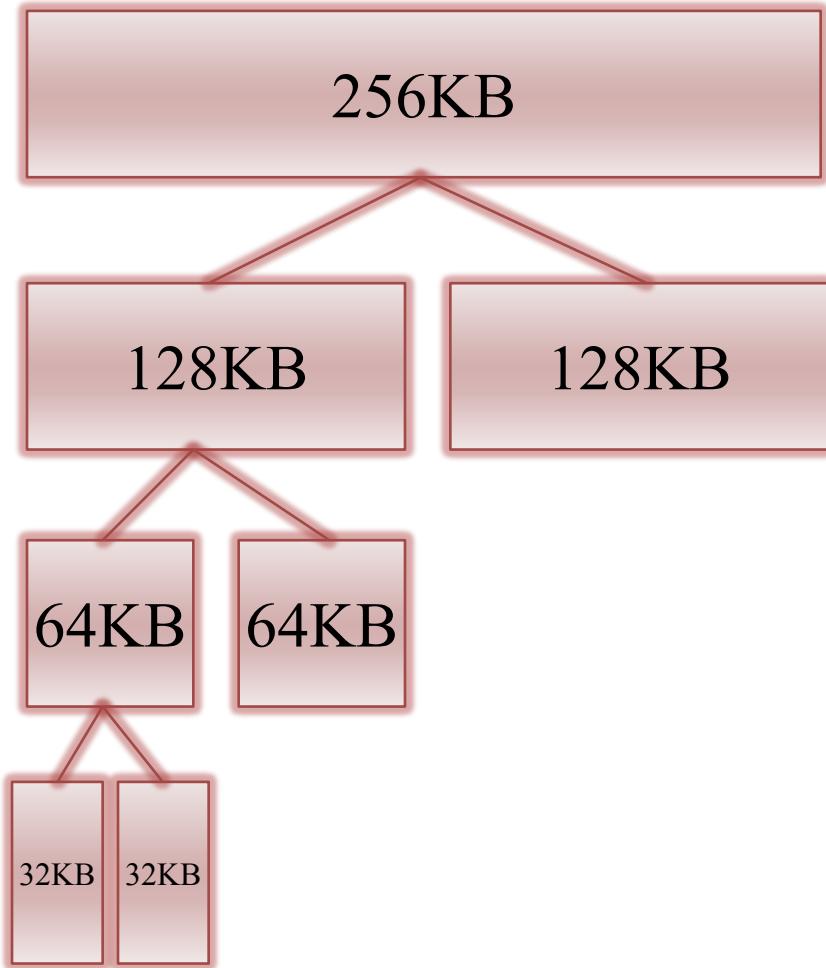


Memory-Mapped I/O

- ▶ Processor can have direct access
- ▶ Memory-Mapped I/O
 - (1) Frequently used devices
 - (2) Devices must be fast, such as video controller, or special I/O instructions are used to move data between memory & device controller registers
- ▶ Programmed I/O – polling
 - or interrupt-driven handling

Kernel Memory Allocation (1 / 2)

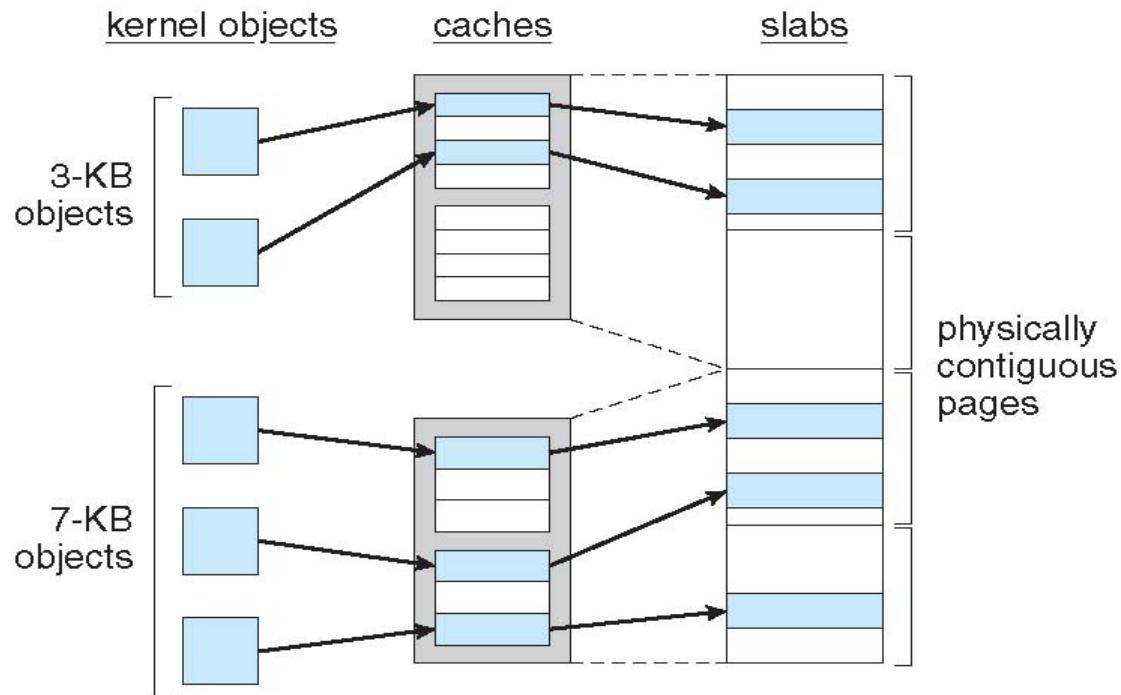
- ▶ The Buddy System
 - A fixed-size segment of physically contiguous pages
 - A power-of-2 allocator
 - Advantage: quick coalescing algorithms
 - Disadvantage: internal fragmentation



Kernel Memory Allocation (2/2)

▶ Slab Allocation

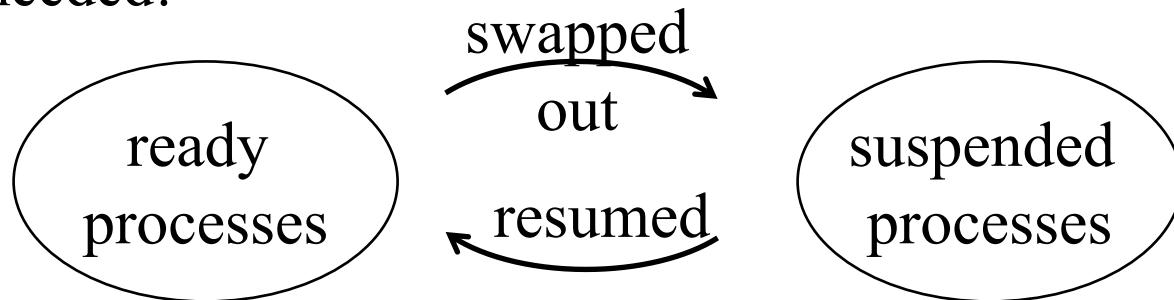
- Slab: one or more physically contiguous pages
- Cache: one or more slabs with the same size



Other Considerations: Pre-Paging

▶ Pre-Paging

- Bring into memory at one time all the pages that will be needed!



Do pre-paging if the working set is known!

▶ Issue

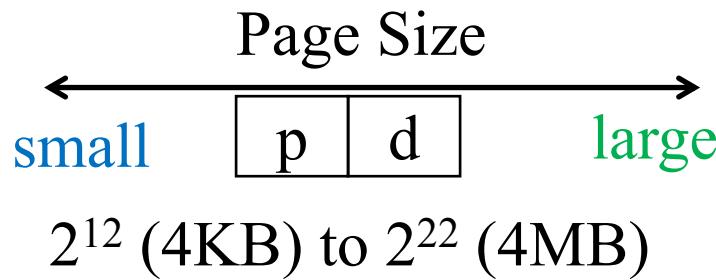
Pre-Paging Cost \longleftrightarrow Cost of Page Fault Services

Not every page in the working set will be used!

Other Considerations: Page Size

▶ Page Size

Better
Resolution
for Locality &
Internal
Fragmentation



Smaller Page
Table Size &
Better I/O
Efficiency

- Trends: Large Page Size
 - ∴ The CPU speed and the memory capacity grow much faster than the disk speed!

Other Considerations: TLB Reach

- ▶ TLB Reach - The amount of memory accessible from the TLB
- ▶ $\text{TLB Reach} = (\text{TLB Size}) \times (\text{Page Size})$
- ▶ 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

Other Considerations: Program Structures

▶ Program Structures:

- 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;
```

1024 x 1024 page faults

- Program 2

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

1024 page faults

Other Considerations: 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



File Concepts

File Attributes

- ▶ **Name** – only information kept in human-readable form
- ▶ **Identifier** – unique tag (number) identifies file within file system
- ▶ **Type** – needed for systems that support different types
- ▶ **Location** – pointer to file location on device
- ▶ **Size** – current file size
- ▶ **Protection** – controls who can do reading, writing, executing
- ▶ **Time, date, and user identification** – data for protection, security, and usage monitoring
- ▶ Information about files are kept in the directory structure, which is maintained on the disk
- ▶ Many variations, including extended file attributes such as file checksum

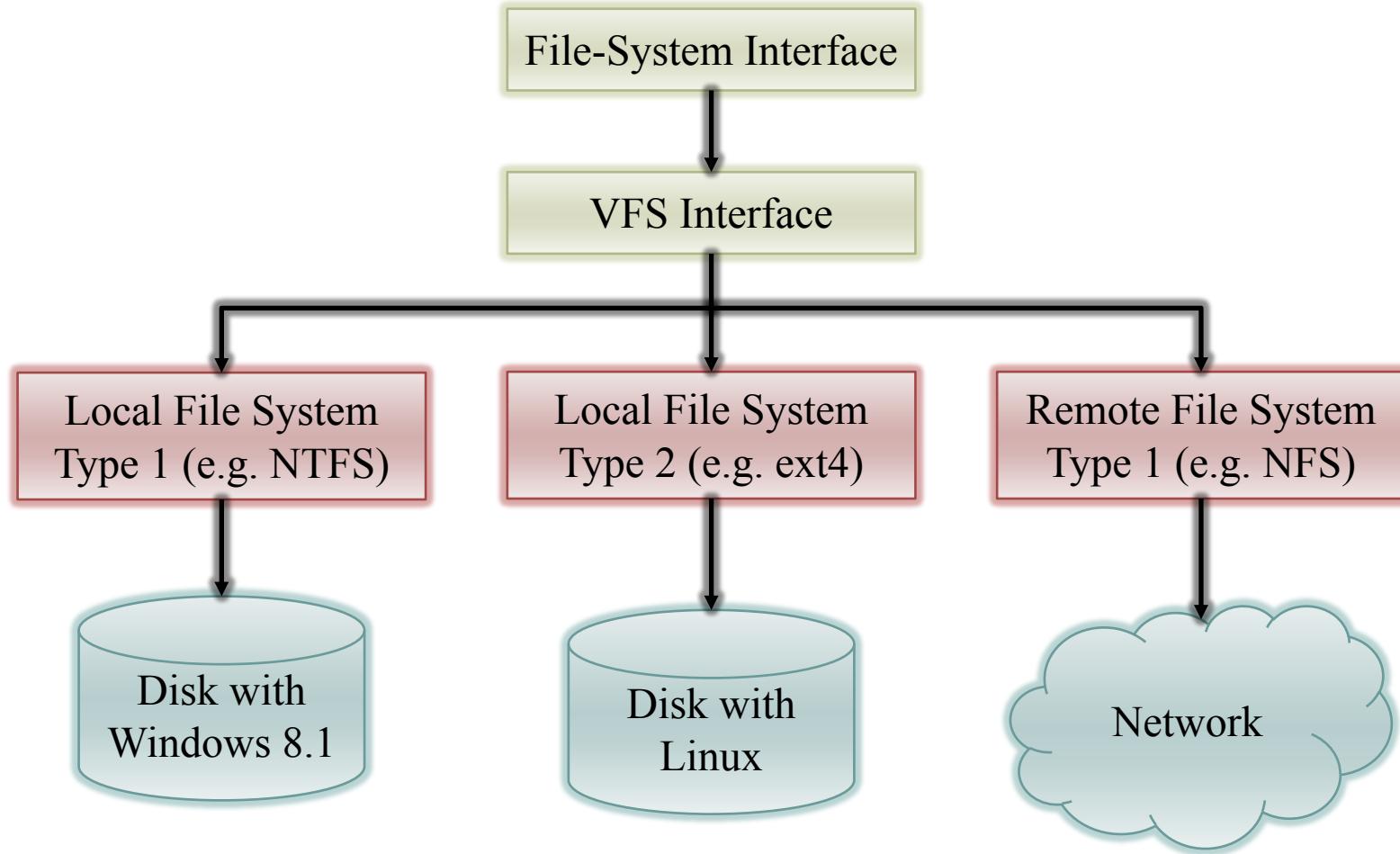
File Operations

- ▶ File is an **abstract data type**
- ▶ **Create**
- ▶ **Write** – at **write pointer** location
- ▶ **Read** – at **read pointer** location
- ▶ **Reposition within file - seek**
- ▶ **Delete**
- ▶ **Truncate**
- ▶ ***Open*(F_i)** – search the directory structure on disk for entry F_i , and move the content of entry to memory
- ▶ ***Close* (F_i)** – move the content of entry F_i in memory to directory structure on disk

File Systems

- ▶ Microsoft Windows File Systems
 - FAT
 - NTFS
 - exFAT
- ▶ Linux File Systems
 - ext2
 - ext3
 - ext4
 - JFFS → for Flash devices
- ▶ Network File Systems
 - NFS
 - Samba

Schematic View of Virtual File System



Virtual File System

- ▶ Virtual File Systems (VFS) on provide an object-oriented way of implementing file systems
- ▶ VFS allows the same system call interface (the API) to be used for different types of file systems
 - Separates file-system generic operations from implementation details
 - Implementation can be one of many file systems types, or network file system
 - Then dispatches operation to appropriate file system implementation routines
- ▶ The API is to the VFS interface, rather than any specific type of file system

Contents

- 1. Introduction
- 2. System Structures
- 3. Process Concept
- 4. Multithreaded Programming
- 5. Process Scheduling
- 6. Synchronization
- 7. Deadlocks
- 8. Memory-Management Strategies
- 9. Virtual-Memory Management
- 10. File System
- 11. Implementing File Systems
- 12. Secondary-Storage Systems

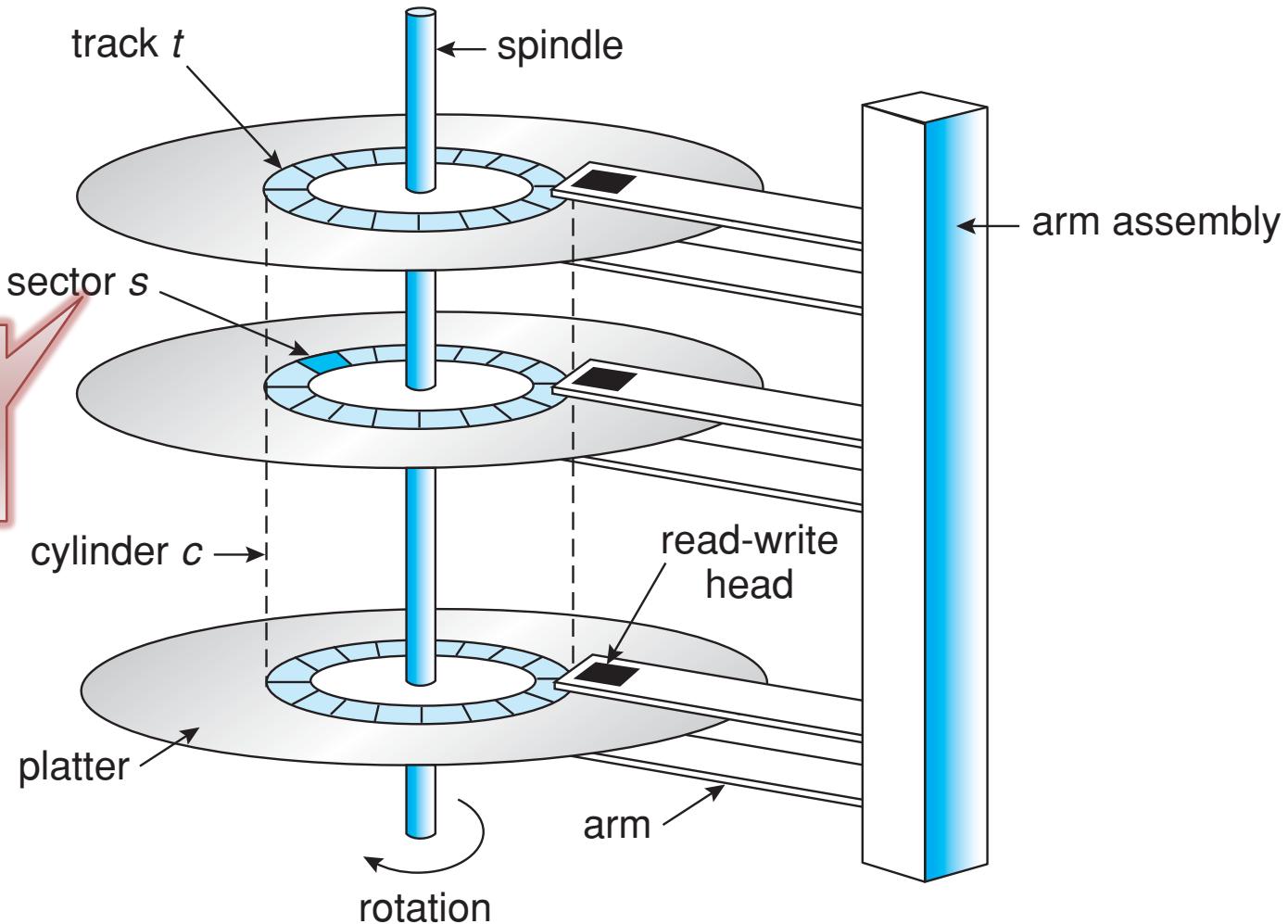




Mass-Storage Structure

Moving-Head Disk Mechanism

The size of a sector is
from 512B to 4KB



Disk Scheduling

- ▶ The disk I/O request specifies several pieces of information:
 - Whether this operation is input or output
 - What the disk address for the transfer is
 - What the memory address for the transfer is
 - What the number of sectors to be transferred is
- ▶ When there are multiple request pending, a good disk scheduling algorithm is required
 - Fairness: which request is the most urgent one
 - Performance: sequential access is preferred

| | | | | | | | |
|-----------|---|---|---|---|---|---|---|
| Cylinders | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Requests | 5 | 7 | 2 | 6 | 4 | 1 | 3 |



Magnetic Disk Performance

- ▶ Access Latency = Average access time = average seek time + average rotation latency
 - For fastest disk $3\text{ms} + 2\text{ms} = 5\text{ms}$
 - For slow disk $9\text{ms} + 5.56\text{ms} = 14.56\text{ms}$
- ▶ Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead



System Protection and Security

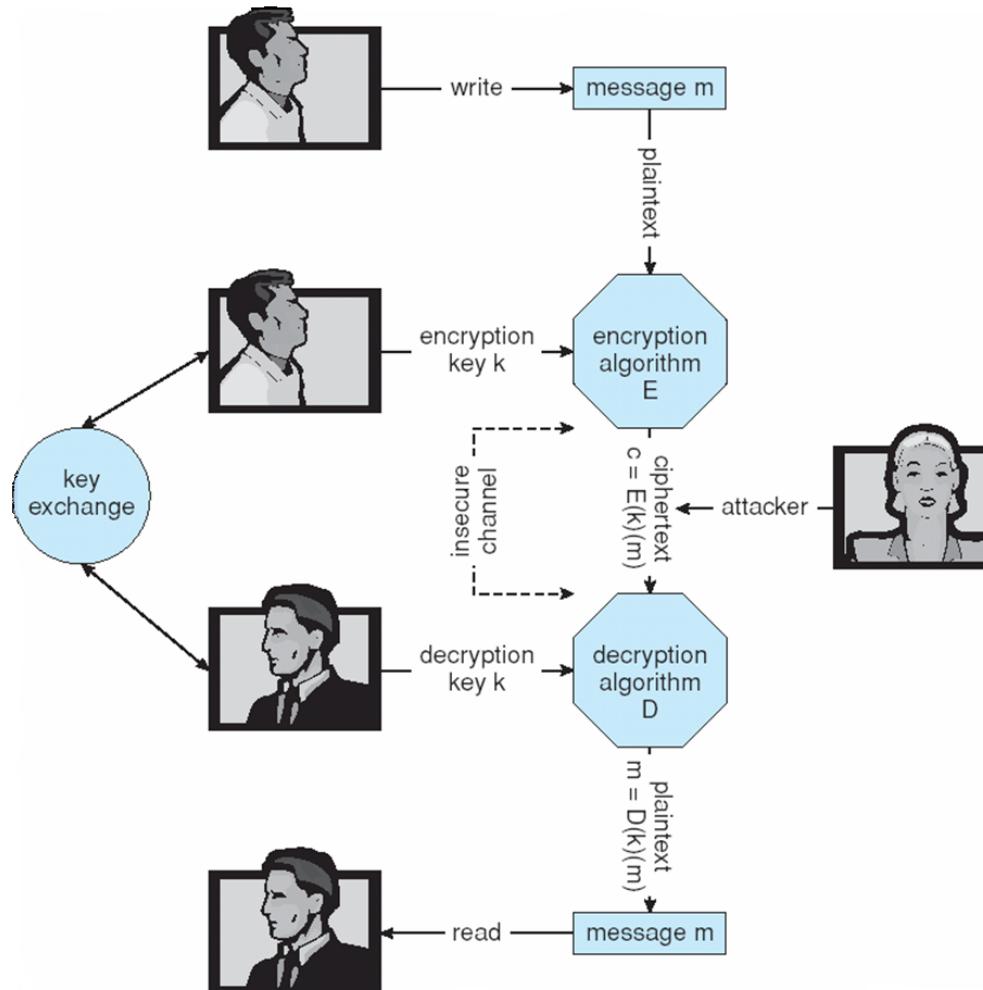
Principles of Protection

- ▶ Principle of Least Privilege
 - Programs, users and systems should be given just enough privileges to perform their tasks
 - Limits damage if entity has a bug or gets abused
- ▶ Principle of Need-to-Know
 - At any time, a process should be able to access only those resources that it currently requires to complete its task

Security Violation Categories

- ▶ Breach of confidentiality
 - Unauthorized reading of data
- ▶ Breach of integrity
 - Unauthorized modification of data
- ▶ Breach of availability
 - Unauthorized destruction of data
- ▶ Theft of service
 - Unauthorized use of resources
- ▶ Denial of service (DOS)
 - Prevention of legitimate use

Secure Communication over Insecure Medium



Scenario of Asymmetric Encryption

