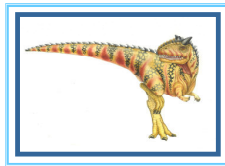


Chapter 11: Implementing File Systems



Chapter 11: Implementing File Systems

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- NFS
- Example: WAFL File System



Objectives

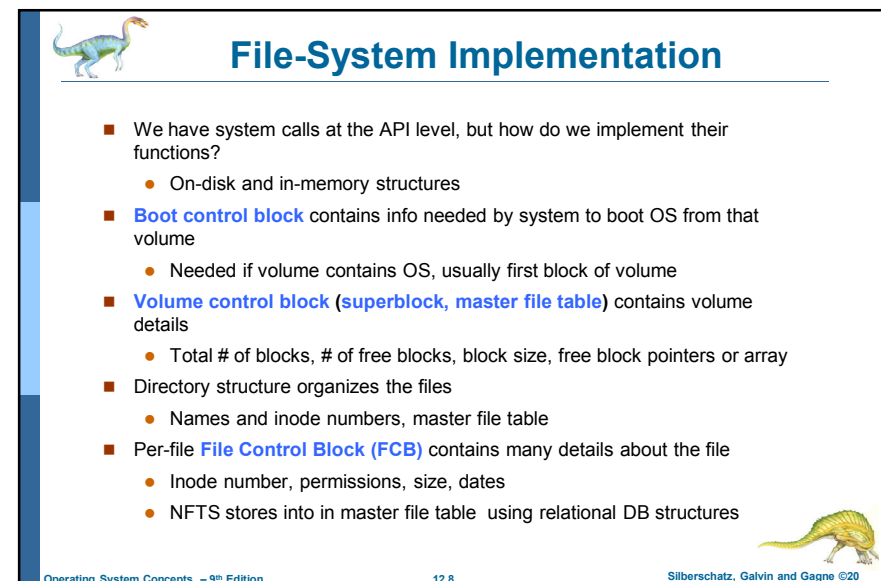
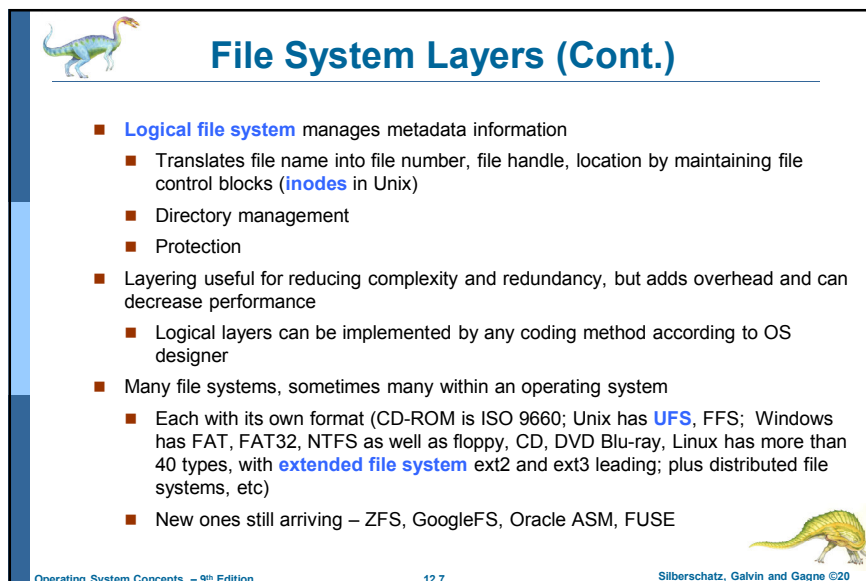
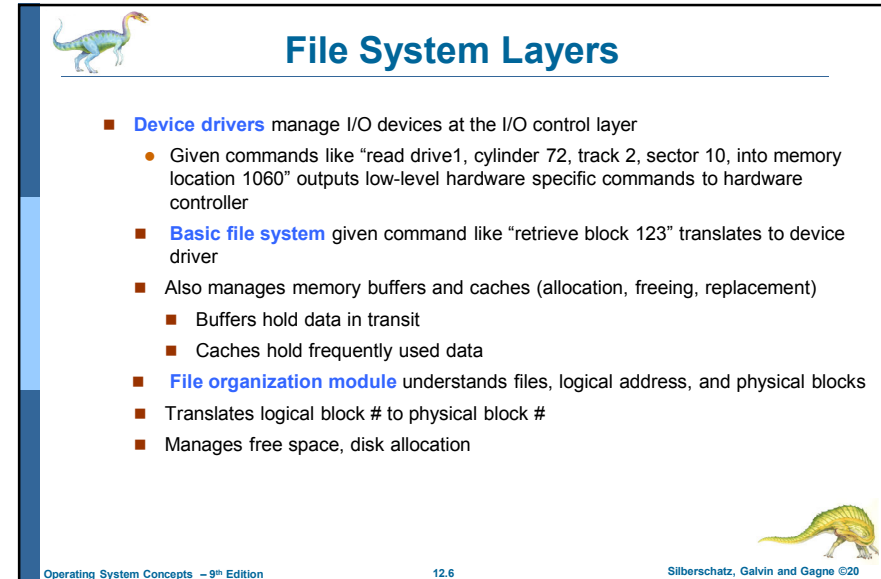
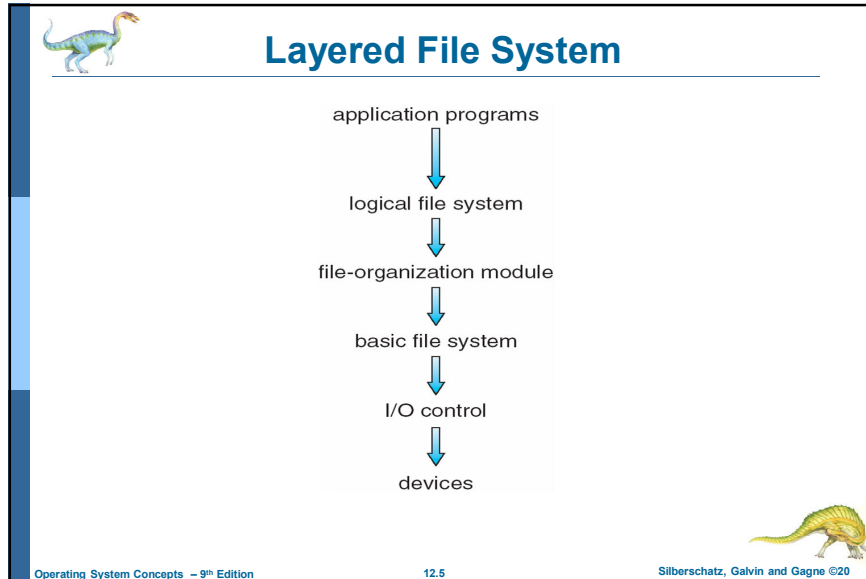
- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems
- To discuss block allocation and free-block algorithms and trade-offs

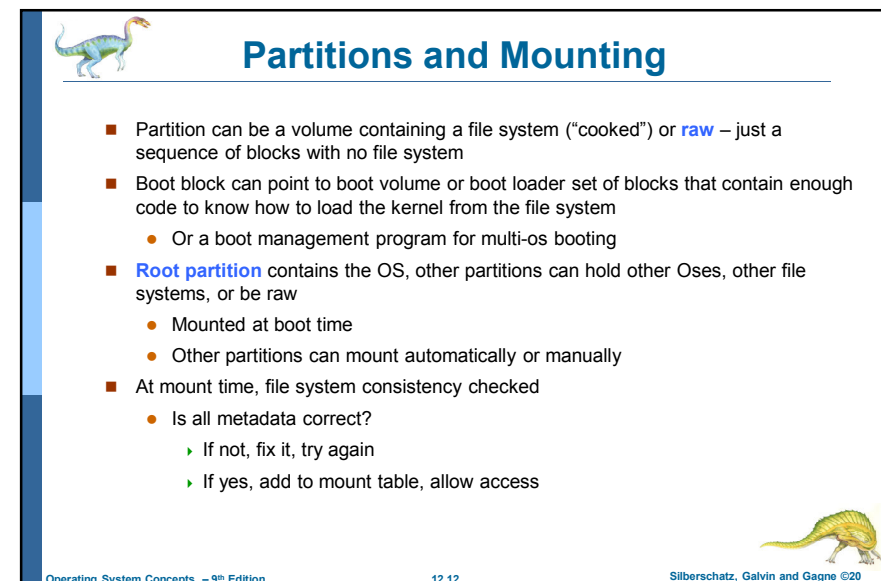
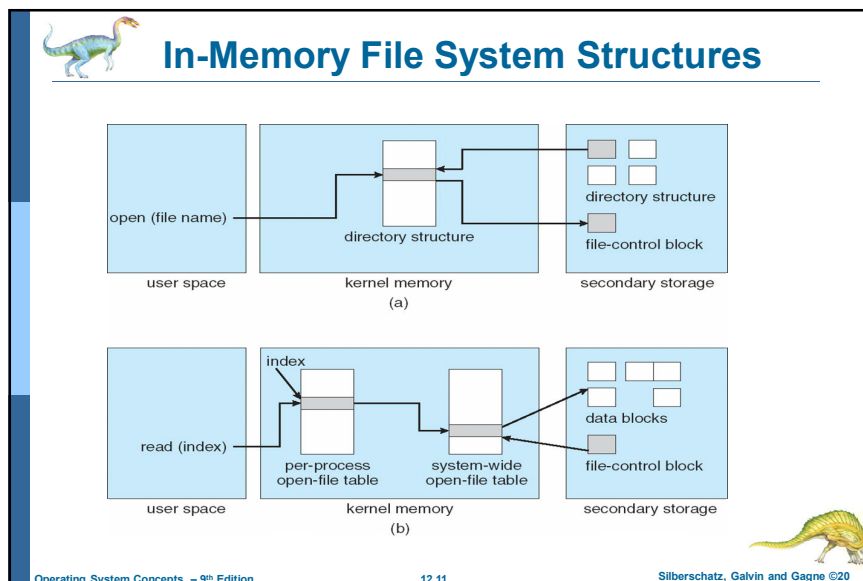
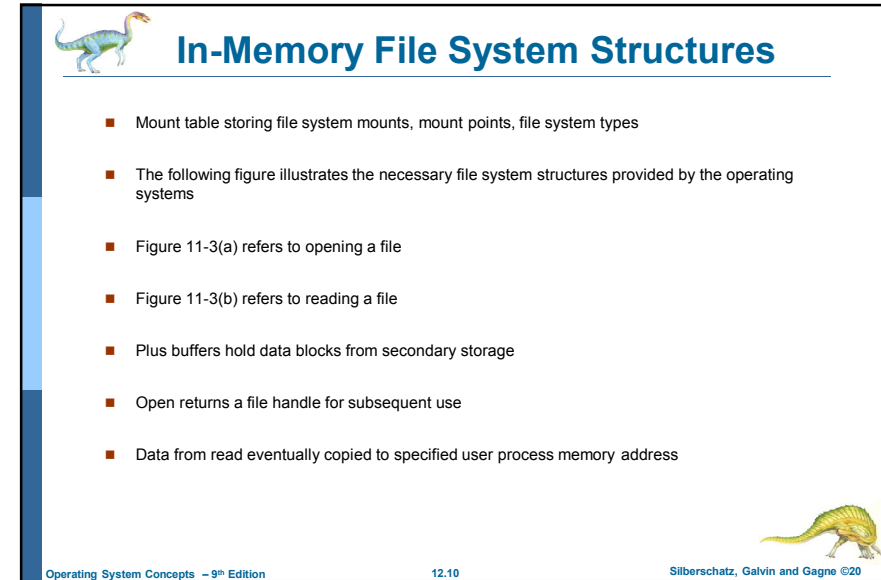
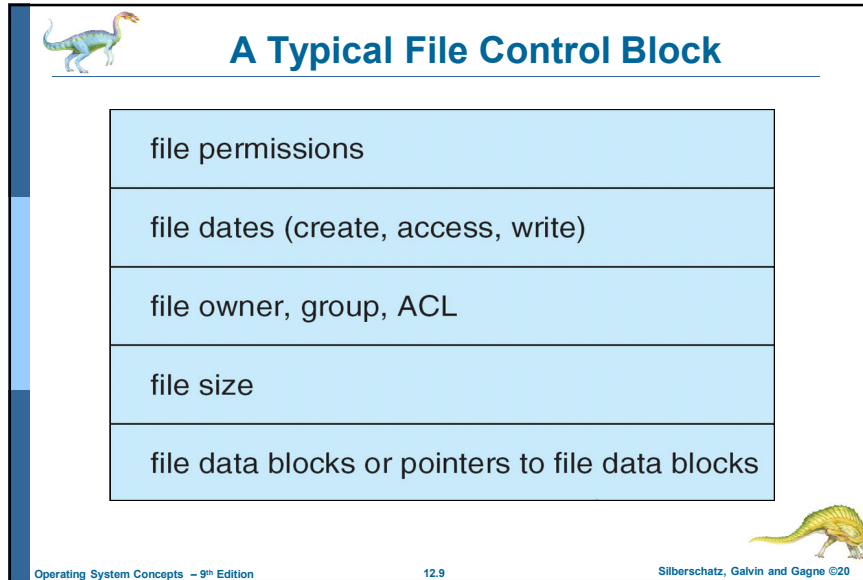


File-System Structure

- File structure
 - Logical storage unit
 - Collection of related information
- **File system** resides on secondary storage (disks)
 - Provided user interface to storage, mapping logical to physical
 - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
 - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers







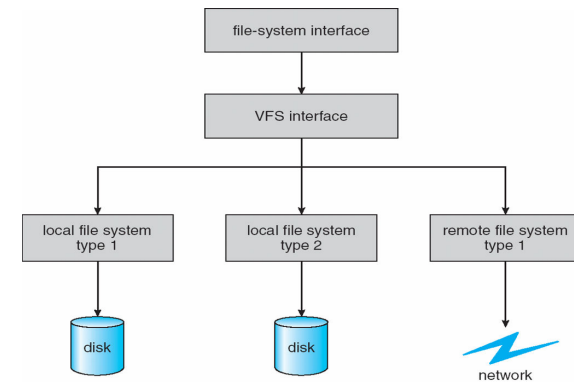


Virtual File Systems

- Virtual File Systems (VFS) on Unix 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
 - Implements vnodes which hold inodes or network file details
 - Then dispatches operation to appropriate file system implementation routines
- The API is to the VFS interface, rather than any specific type of file system



Schematic View of Virtual File System



Virtual File System Implementation

- For example, Linux has four object types:
 - inode, file, superblock, dentry
- VFS defines set of operations on the objects that must be implemented
 - Every object has a pointer to a function table
 - Function table has addresses of routines to implement that function on that object



Directory Implementation

- **Linear list** of file names with pointer to the data blocks
 - Simple to program
 - Time-consuming to execute
 - Linear search time
 - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
 - Decreases directory search time
 - **Collisions** – situations where two file names hash to the same location
 - Only good if entries are fixed size, or use chained-overflow method



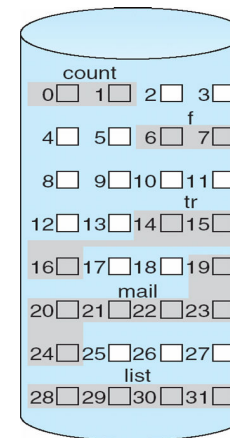


Allocation Methods - Contiguous

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
 - Best performance in most cases
 - Simple – only starting location (block #) and length (number of blocks) are required
 - Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line (downtime)** or **on-line**



Contiguous Allocation of Disk Space



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2



Extent-Based Systems

- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
 - Extents are allocated for file allocation
 - A file consists of one or more extents



Allocation Methods - Linked

- **Linked allocation** – each file a linked list of blocks
 - File ends at nil pointer
 - No external fragmentation
 - Each block contains pointer to next block
 - No compaction, external fragmentation
 - Free space management system called when new block needed
 - Improve efficiency by clustering blocks into groups but increases internal fragmentation
 - Reliability can be a problem
 - Locating a block can take many I/Os and disk seeks
- FAT (File Allocation Table) variation
 - Beginning of volume has table, indexed by block number
 - Much like a linked list, but faster on disk and cacheable
 - New block allocation simple




Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk

block =

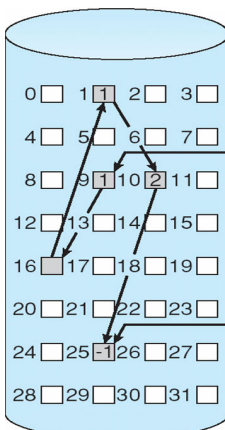
pointer




Operating System Concepts – 9th Edition 12.21 Silberschatz, Galvin and Gagne ©20

Linked Allocation

file	start	end
jeep	9	25





Operating System Concepts – 9th Edition 12.22 Silberschatz, Galvin and Gagne ©20


File-Allocation Table

directory entry

test	...	217
name		start block

0	
217	618
339	
618	339
no. of disk blocks	-1

FAT




Operating System Concepts – 9th Edition 12.23 Silberschatz, Galvin and Gagne ©20

Allocation Methods - Indexed

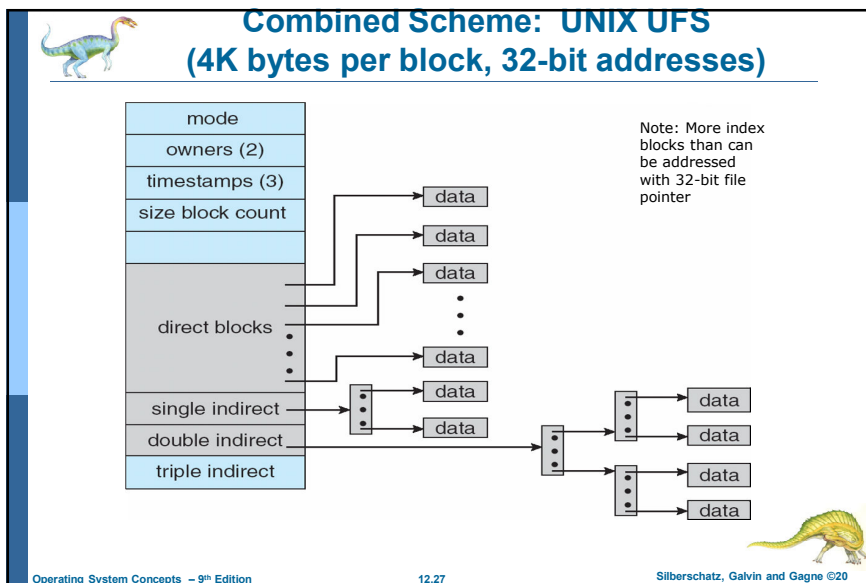
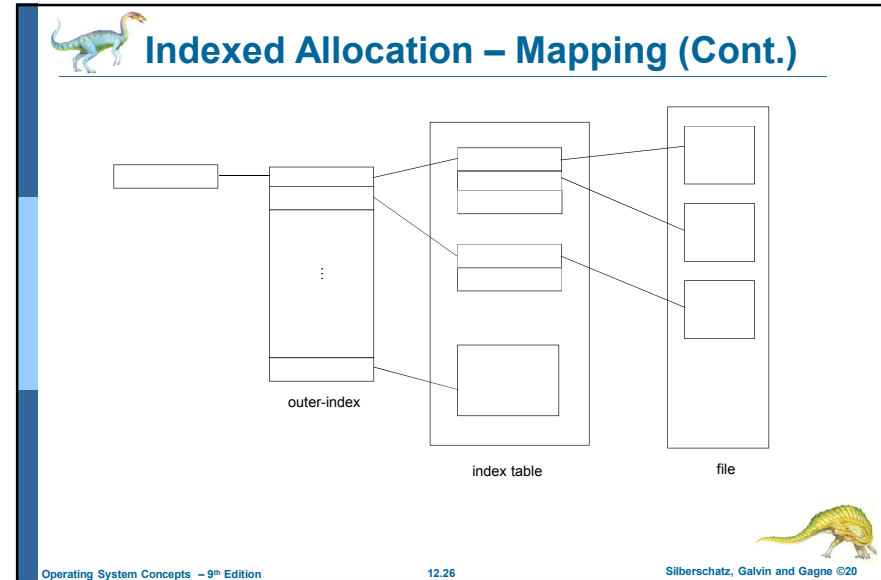
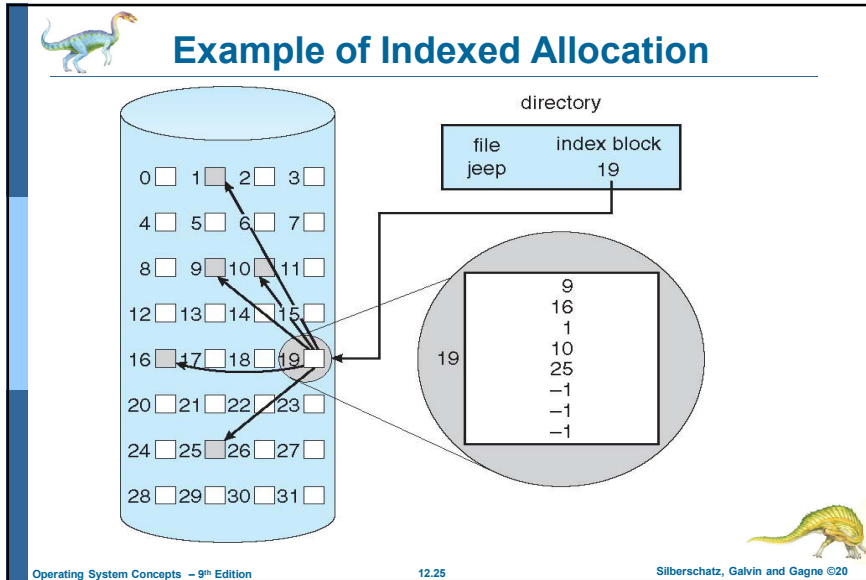
- Indexed allocation**
 - Each file has its own **index block**(s) of pointers to its data blocks
- Logical view

	→	□
	→	□
	→	□
	→	□
	→	□
	→	□


index table



Operating System Concepts – 9th Edition 12.24 Silberschatz, Galvin and Gagne ©20




- ### Performance
- Best method depends on file access type
 - Contiguous great for sequential and random
 - Linked good for sequential, not random
 - Declare access type at creation -> select either contiguous or linked
 - Indexed more complex
 - Single block access could require 2 index block reads then data block read
 - Clustering can help improve throughput, reduce CPU overhead
- Operating System Concepts – 9th Edition 12.28 Silberschatz, Galvin and Gagne ©20




Performance (Cont.)

- Adding instructions to the execution path to save one disk I/O is reasonable
 - Intel Core i7 Extreme Edition 990x (2011) at 3.46Ghz = 159,000 MIPS
 - ▶ http://en.wikipedia.org/wiki/Instructions_per_second
 - Typical disk drive at 250 I/Os per second
 - ▶ $159,000 \text{ MIPS} / 250 = 630$ million instructions during one disk I/O
 - Fast SSD drives provide 60,000 IOPS
 - ▶ $159,000 \text{ MIPS} / 60,000 = 2.65$ millions instructions during one disk I/O



Operating System Concepts – 9th Edition
12.29
Silberschatz, Galvin and Gagne ©20



Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
 - (Using term "block" for simplicity)
- **Bit vector** or **bit map** (n blocks)


0	1	2	...	$n-1$

$$\text{bit}[j] = \begin{cases} 1 \Rightarrow \text{block}[j] \text{ free} \\ 0 \Rightarrow \text{block}[j] \text{ occupied} \end{cases}$$


Block number calculation

$$\begin{aligned} &(\text{number of bits per word}) * \\ &(\text{number of 0-value words}) + \\ &\text{offset of first 1 bit} \end{aligned}$$

CPUs have instructions to return offset within word of first "1" bit




Operating System Concepts – 9th Edition
12.30
Silberschatz, Galvin and Gagne ©20




Free-Space Management (Cont.)

- Bit map requires extra space
 - Example:
 - block size = 4KB = 2^{12} bytes
 - disk size = 2^{40} bytes (1 terabyte)
 - $n = 2^{40}/2^{12} = 2^{28}$ bits (or 256 MB)
 - if clusters of 4 blocks -> 64MB of memory
- Easy to get contiguous files
- Linked list (free list)
 - Cannot get contiguous space easily
 - No waste of space
 - No need to traverse the entire list (if # free blocks recorded)




Operating System Concepts – 9th Edition
12.31
Silberschatz, Galvin and Gagne ©20



Linked Free Space List on Disk

free-space list head



Operating System Concepts – 9th Edition
12.32
Silberschatz, Galvin and Gagne ©20



Free-Space Management (Cont.)

- Grouping
 - Modify linked list to store address of next $n-1$ free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)
- Counting
 - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
 - ▶ Keep address of first free block and count of following free blocks
 - ▶ Free space list then has entries containing addresses and counts



Efficiency and Performance

- Efficiency dependent on:
 - Disk allocation and directory algorithms
 - Types of data kept in file's directory entry
 - Pre-allocation or as-needed allocation of metadata structures
 - Fixed-size or varying-size data structures



Efficiency and Performance (Cont.)

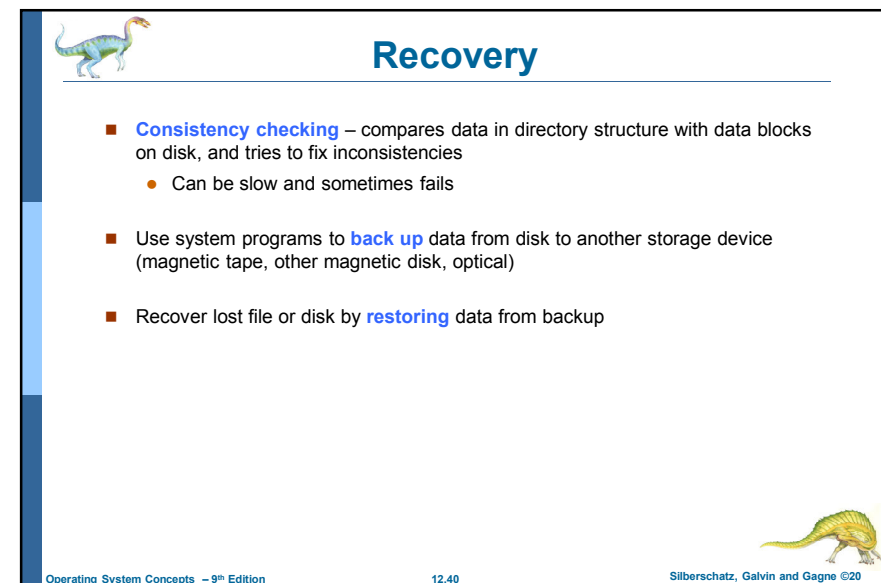
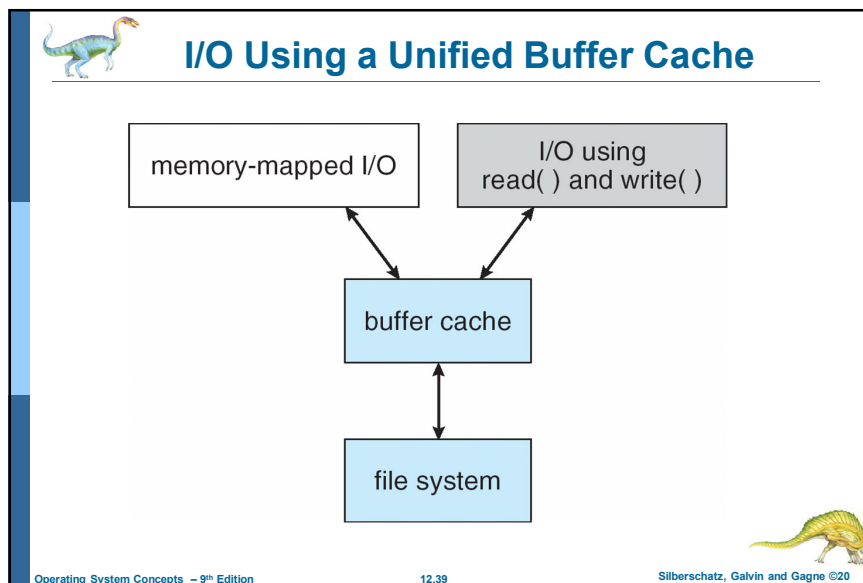
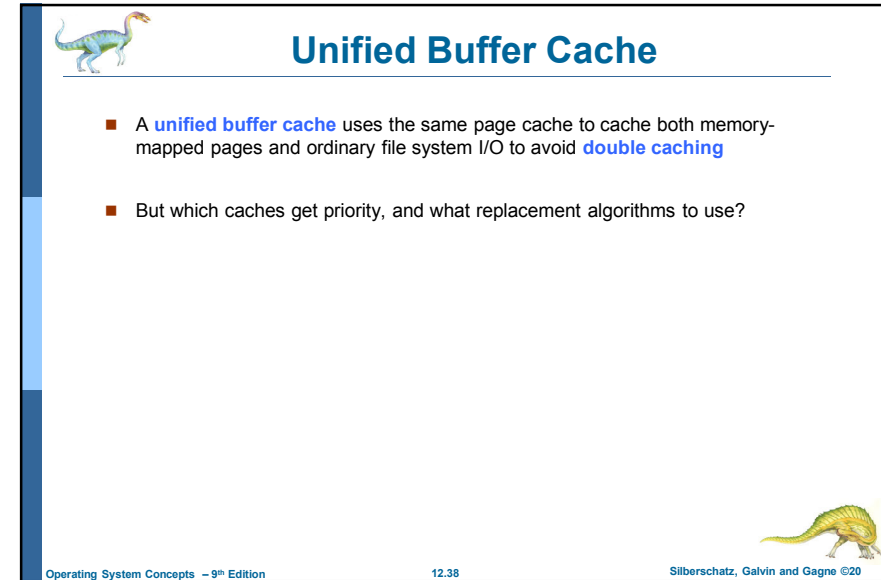
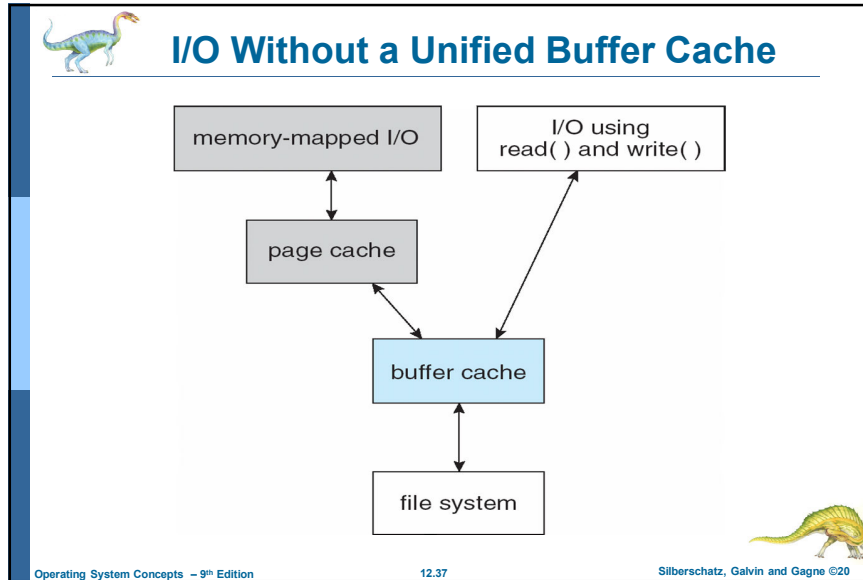
- Performance
 - Keeping data and metadata close together
 - **Buffer cache** – separate section of main memory for frequently used blocks
 - **Synchronous** writes sometimes requested by apps or needed by OS
 - ▶ No buffering / caching – writes must hit disk before acknowledgement
 - ▶ **Asynchronous** writes more common, buffer-able, faster
 - **Free-behind** and **read-ahead** – techniques to optimize sequential access
 - Reads frequently slower than writes



Page Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques and addresses
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure







Log Structured File Systems

- **Log structured** (or **journaling**) file systems record each metadata update to the file system as a **transaction**
- All transactions are written to a log
 - A transaction is considered committed once it is written to the log (sequentially)
 - Sometimes to a separate device or section of disk
 - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system structures
 - When the file system structures are modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed
- Faster recovery from crash, removes chance of inconsistency of metadata



The Sun Network File System (NFS)

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs)
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations using an unreliable datagram protocol (UDP/IP protocol and Ethernet)



NFS (Cont.)

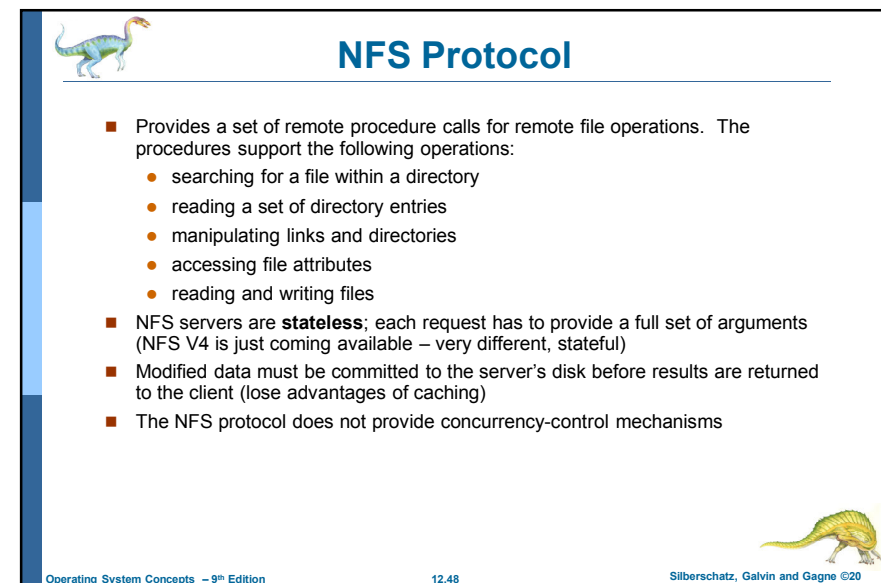
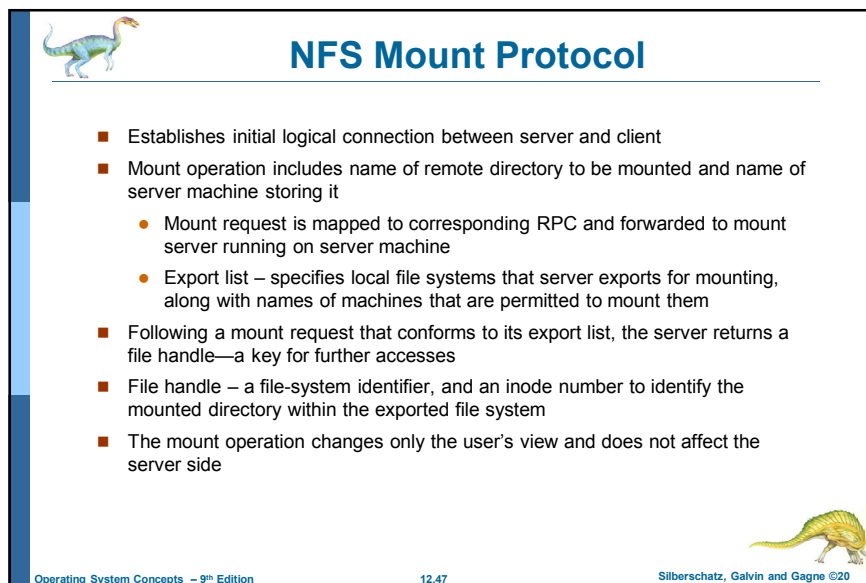
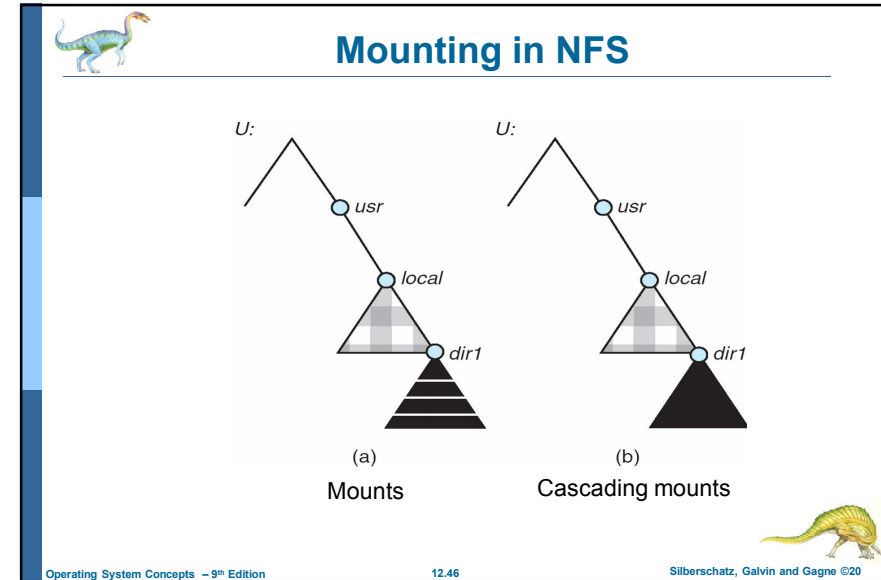
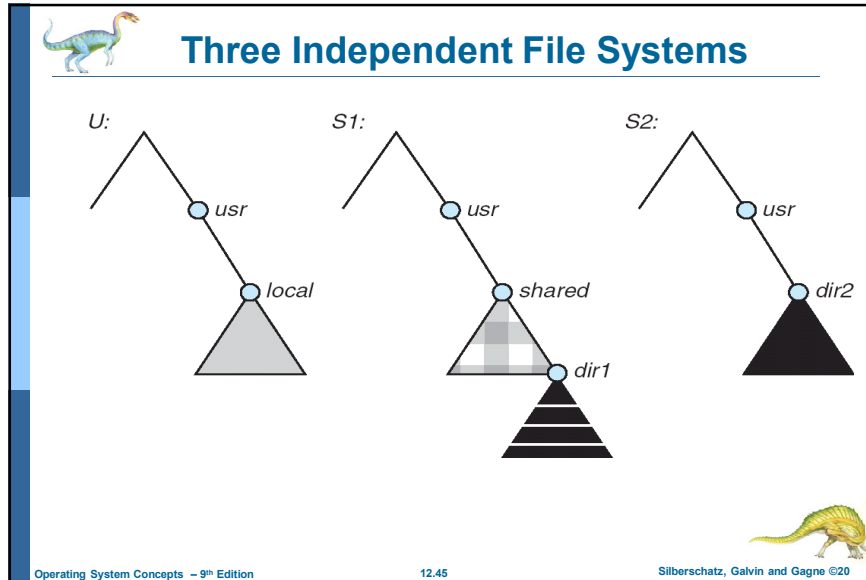
- Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
 - A remote directory is mounted over a local file system directory
 - ▶ The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory
 - Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
 - ▶ Files in the remote directory can then be accessed in a transparent manner
 - Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory



NFS (Cont.)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services







Three Major Layers of NFS Architecture

- UNIX file-system interface (based on the **open**, **read**, **write**, and **close** calls, and **file descriptors**)
- *Virtual File System (VFS)* layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
 - The VFS activates file-system-specific operations to handle local requests according to their file-system types
 - Calls the NFS protocol procedures for remote requests
- NFS service layer – bottom layer of the architecture
 - Implements the NFS protocol



Schematic View of NFS Architecture

