
University of Mosul

College of Computer Sciences and Mathematics

Software Department - Third Class - First course

Subject : **File Processing**

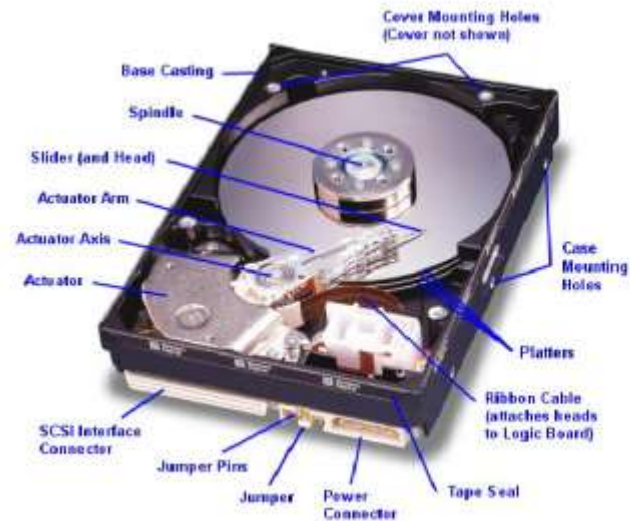
Lecture : Dr. Alyaa Qusay

Objective of File system

- To explain the function of file systems
- To describe the interfaces to file systems
- To describe storage management
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection and security

Lecture 1: Mass-Storage Systems

- Overview of Mass Storage Structure
- Disk Structure
- Disk Attachment
- Disk Scheduling
- Disk Management
- Swap-Space Management
- RAID Structure
- Disk Attachment
- Stable-Storage Implementation
- Tertiary Storage Devices
- Operating System Support
- Performance Issues



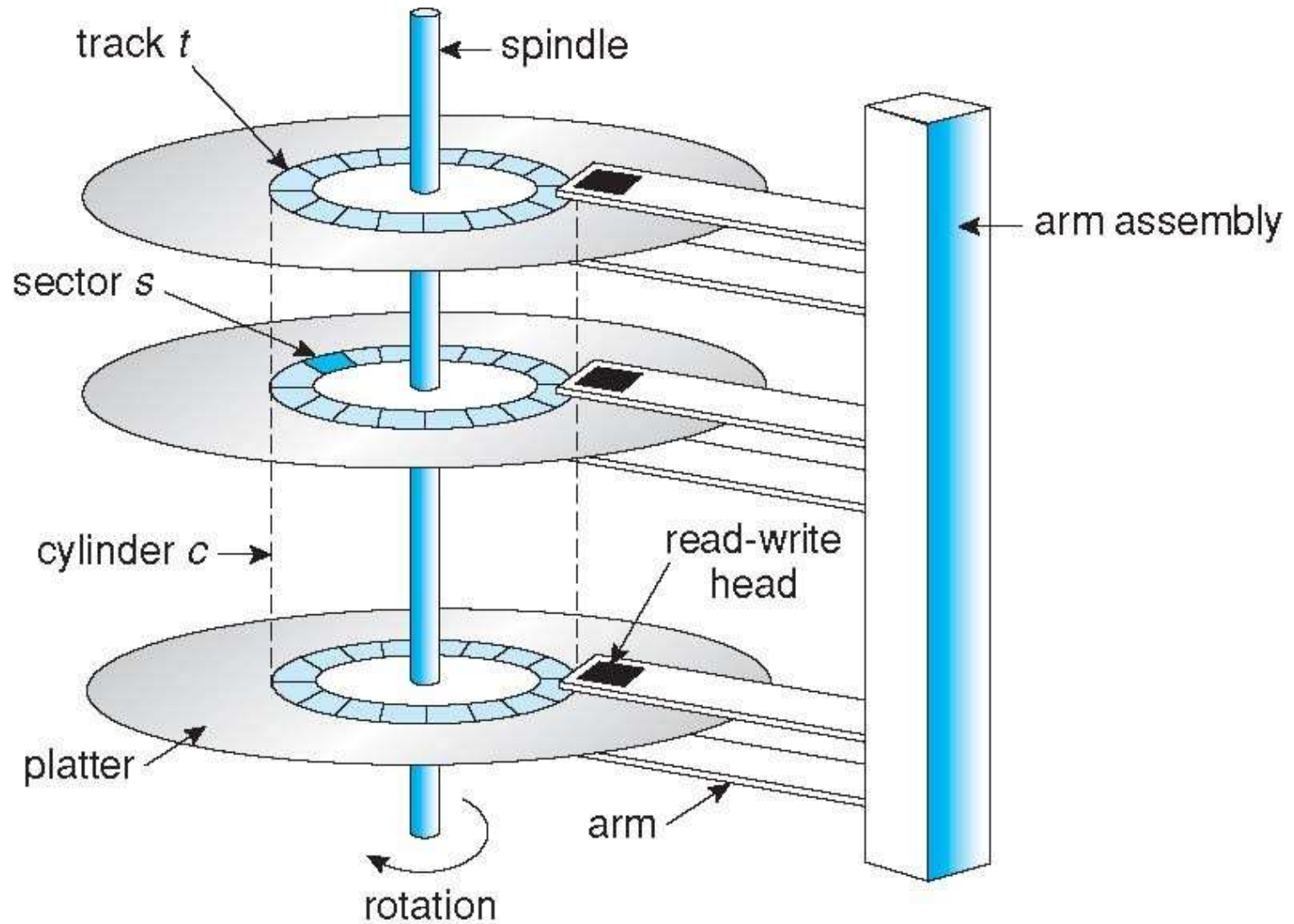
Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
 - **Sector 0** is the first sector of the first track on the **outermost cylinder**
 - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost



Overview of Mass Storage Structure

- **Magnetic disks** provide bulk of secondary storage of modern computers
 - Drives rotate at 60 to 200 times per second
 - **Transfer rate** is rate at which data flow between drive and computer
 - **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
 - **Head crash** results from disk head making contact with the disk surface
 - 🏠 That's bad
- Disks can be removable
- Drive attached to computer via **I/O bus**
 - Busses vary, including **EIDE**, **ATA**, **SATA**, **USB**, **Fibre Channel**, **SCSI**
 - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

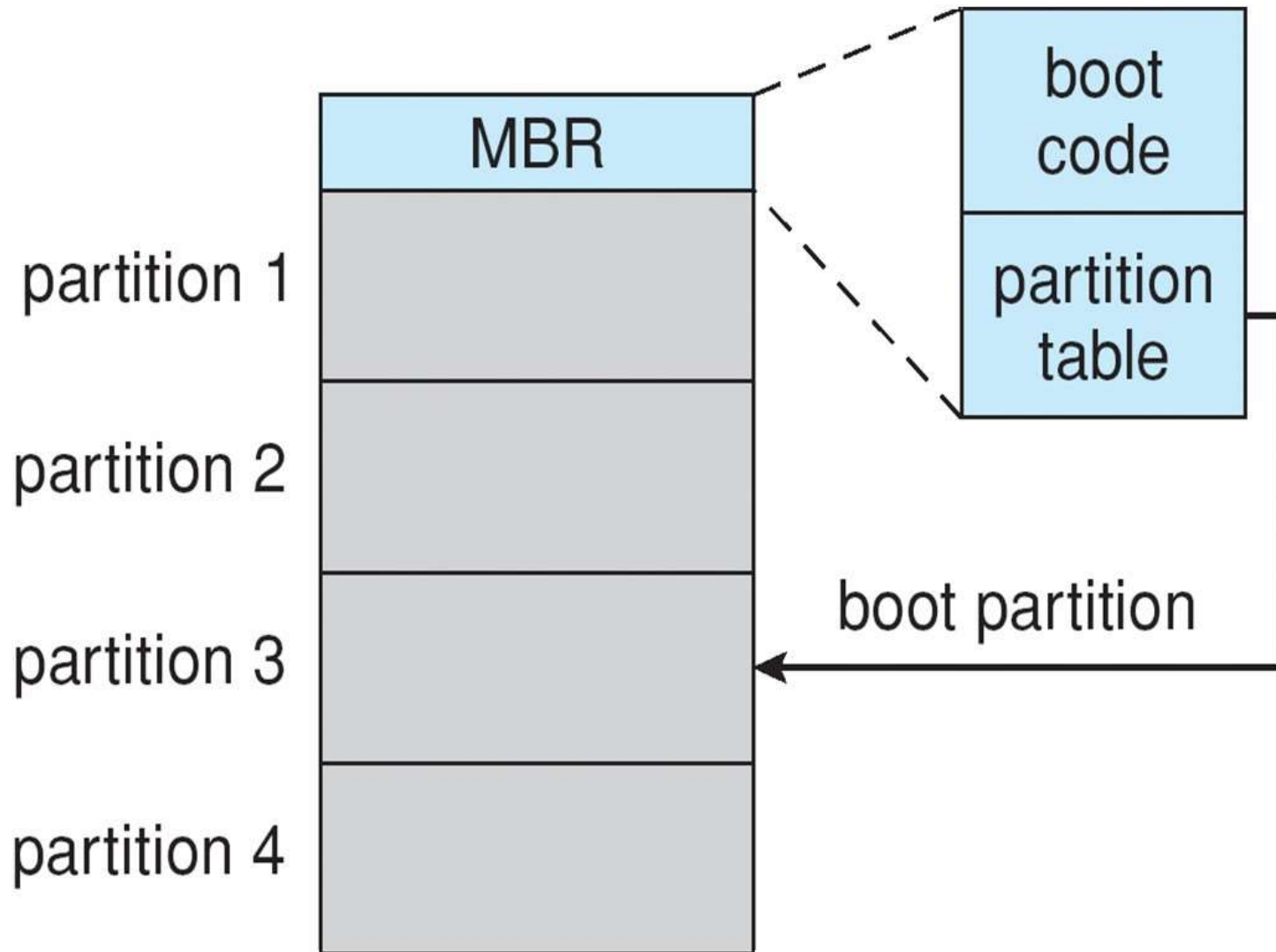
Moving-head Disk Mechanism



Disk Management

- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk
 - **Partition** the disk into one or more groups of cylinders
 - **Logical formatting** or “making a file system”
 - To increase efficiency most file systems group blocks into **clusters**
 -  Disk I/O done in blocks
 -  File I/O done in clusters
- Boot block initializes system
 - The bootstrap is stored in ROM
 - **Bootstrap loader** program
- Methods such as **sector sparing** used to handle bad blocks

Booting from a Disk in Windows 2000



Removable Disks

- Floppy disk — thin flexible disk coated with magnetic material, enclosed in a protective plastic case
 - Most floppies hold about 1 MB; similar technology is used for removable disks that hold more than 1 GB
 - Removable magnetic disks can be nearly as fast as hard disks, but they are at a greater risk of damage from exposure
- A magneto-optic disk records data on a rigid platter coated with magnetic material
 - Laser heat is used to amplify a large, weak magnetic field to record a bit
 - Laser light is also used to read data (Kerr effect)
 - The magneto-optic head flies much farther from the disk surface than a magnetic disk head, and the magnetic material is covered with a protective layer of plastic or glass; resistant to head crashes
- Optical disks do not use magnetism; they employ special materials that are altered by laser light

WORM Disks

- The data on read-write disks can be modified over and over
- **WORM** (“Write Once, Read Many Times”) disks can be written only once
- Thin aluminum film sandwiched between two glass or plastic platters
- To write a bit, the drive uses a laser light to burn a small hole through the aluminum; information can be destroyed by not altered
- Very durable and reliable
- **Read-only disks**, such as CD-ROM and DVD, come from the factory with the data pre-recorded

Lecture 2 :Overview of Mass Storage Structure

■ Magnetic tape

- Was early secondary-storage medium
- Relatively permanent and holds large quantities of data
- Access time slow
- Random access ~1000 times slower than disk
- Mainly used for backup, storage of infrequently-used data, transfer medium between systems
- Kept in spool and wound or rewound past read-write head
- Once data under head, transfer rates comparable to disk
- 20-200GB typical storage
- Common technologies are 4mm, 8mm, 19mm, LTO-2 and SDLT
- A disk-resident file can be **archived** to tape for low cost storage; the computer can **stage** it back into disk storage for active use.

RAID Structure

- RAID – multiple disk drives provides reliability via **redundancy**
- Increases the **mean time to failure**
- Frequently combined with **NVRAM** to improve write performance
- RAID is arranged into six different levels

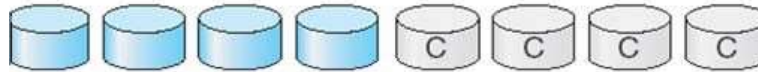
RAID (Cont.)

- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively
- Disk **striping** uses a group of disks as one storage unit
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
 - **Mirroring** or **shadowing** (**RAID 1**) keeps duplicate of each disk
 - Striped mirrors (**RAID 1+0**) or mirrored stripes (**RAID 0+1**) provides high performance and high reliability
 - **Block interleaved parity** (**RAID 4, 5, 6**) uses much less redundancy
- RAID within a storage array can still fail if the array fails, so automatic **replication** of the data between arrays is common
- Frequently, a small number of **hot-spare** disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them

RAID Levels



(a) RAID 0: non-redundant striping.



(b) RAID 1: mirrored disks.



(c) RAID 2: memory-style error-correcting codes.



(d) RAID 3: bit-interleaved parity.



(e) RAID 4: block-interleaved parity.



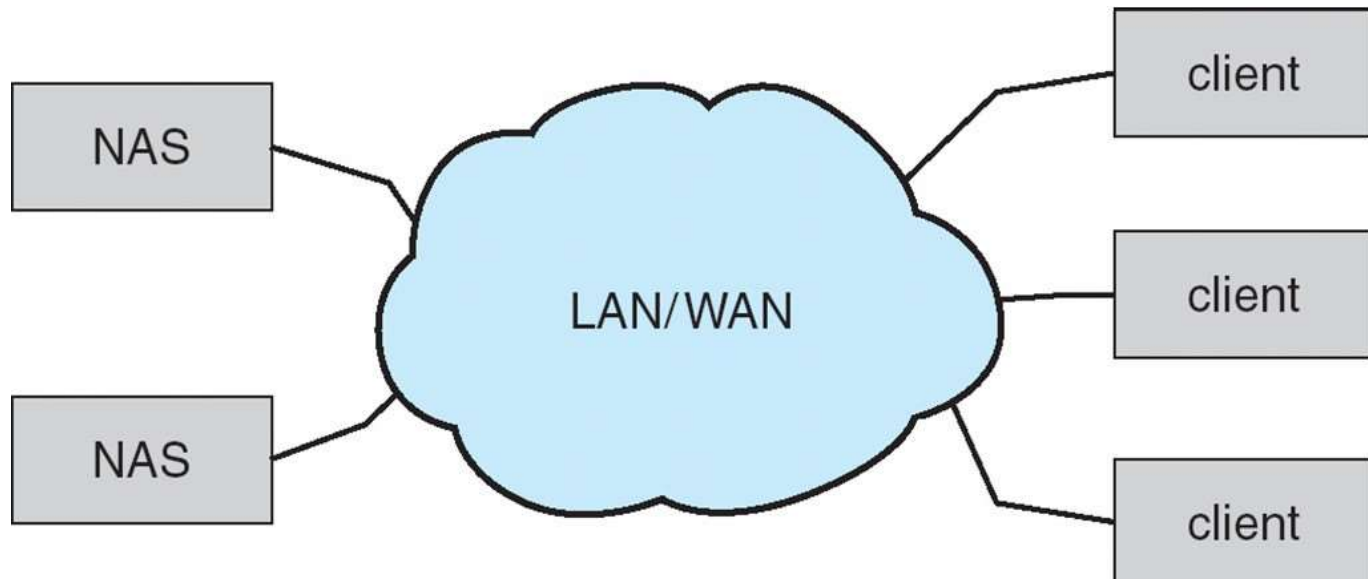
(f) RAID 5: block-interleaved distributed parity.



(g) RAID 6: P + Q redundancy.

Network-Attached Storage

- Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)
- NFS and CIFS are common protocols
- Implemented via remote procedure calls (RPCs) between host and storage
- New **iSCSI** protocol uses IP network to carry the SCSI protocol



Chapter 10: File-System Interface

Lecture 3

- File Concept
- Access Methods
- Directory Structure
- File-System Mounting
- File Sharing
- Protection

- File definition Contiguous logical address space

- Types:
 - Data
 - numeric ■ character ■ binary
 - Program

File Structure

- None - sequence of words, bytes
- Simple record structure
 - Lines
 - Fixed length
 - Variable length
- Complex Structures
 - Formatted document
 - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
 - Operating system
 - Program

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

File Operations

- File is an **abstract data type**
- **Create**
- **Write**
- **Read**
- **Reposition within file**
- **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

Open Files

- Several pieces of data are needed to manage open files:
 - File pointer: pointer to last read/write location, per process that has the file open
 - File-open count: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
 - Disk location of the file: cache of data access information
 - Access rights: per-process access mode information
- **Open File Locking**
- Provided by some operating systems and file systems
- Mediates access to a file
- Mandatory or advisory:
 - **Mandatory** – access is denied depending on locks held and requested
 - **Advisory** – processes can find status of locks and decide what to do

File Types – Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine- language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com- pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

■ Sequential Access

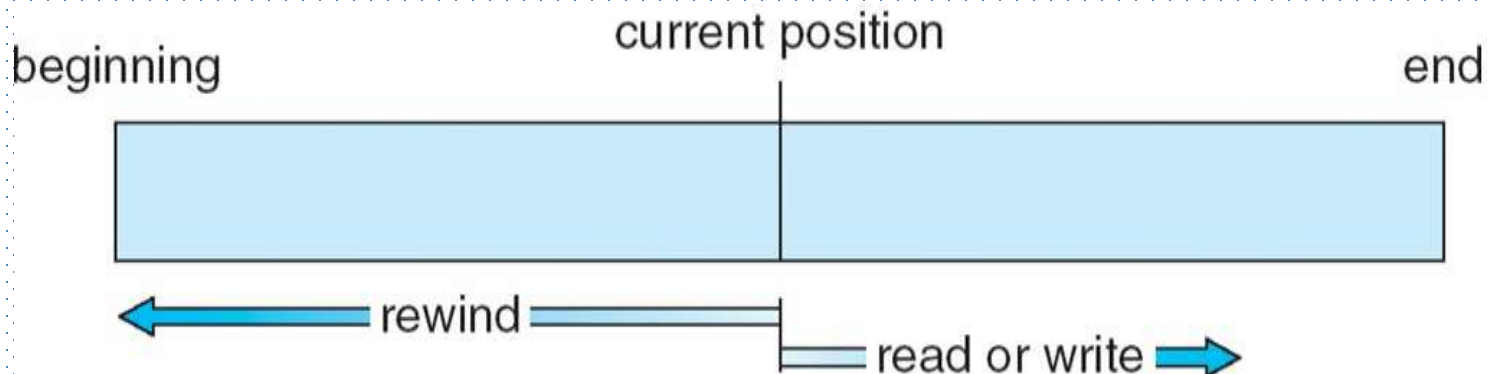
read next write next reset
no read after last write (rewrite)

■ Direct Access

read n
write n
position to n read next write next rewrite n

n = relative block number

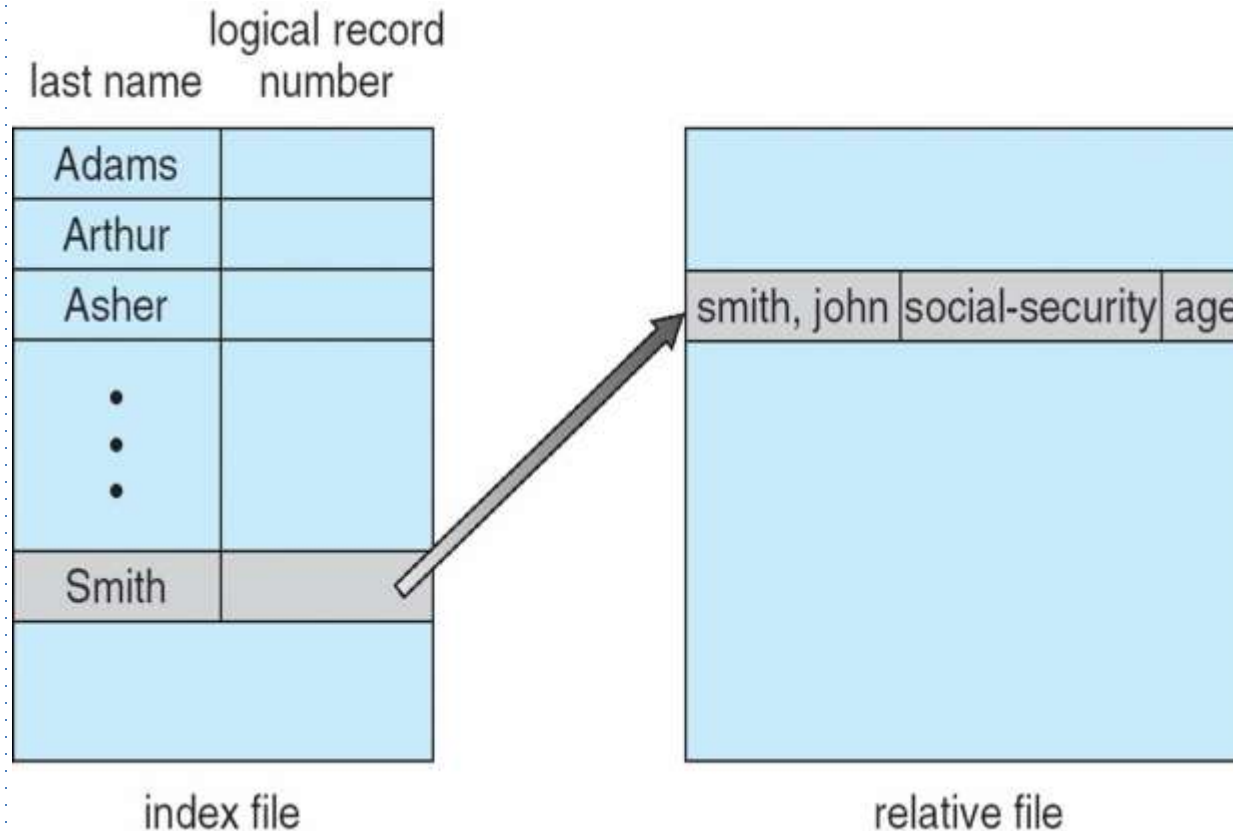
Sequential Access



Simulation of Sequential Access on Direct-access File

sequential access	implementation for direct access
<i>reset</i>	<i>cp = 0;</i>
<i>read next</i>	<i>read cp;</i> <i>cp = cp + 1;</i>
<i>write next</i>	<i>write cp;</i> <i>cp = cp + 1;</i>

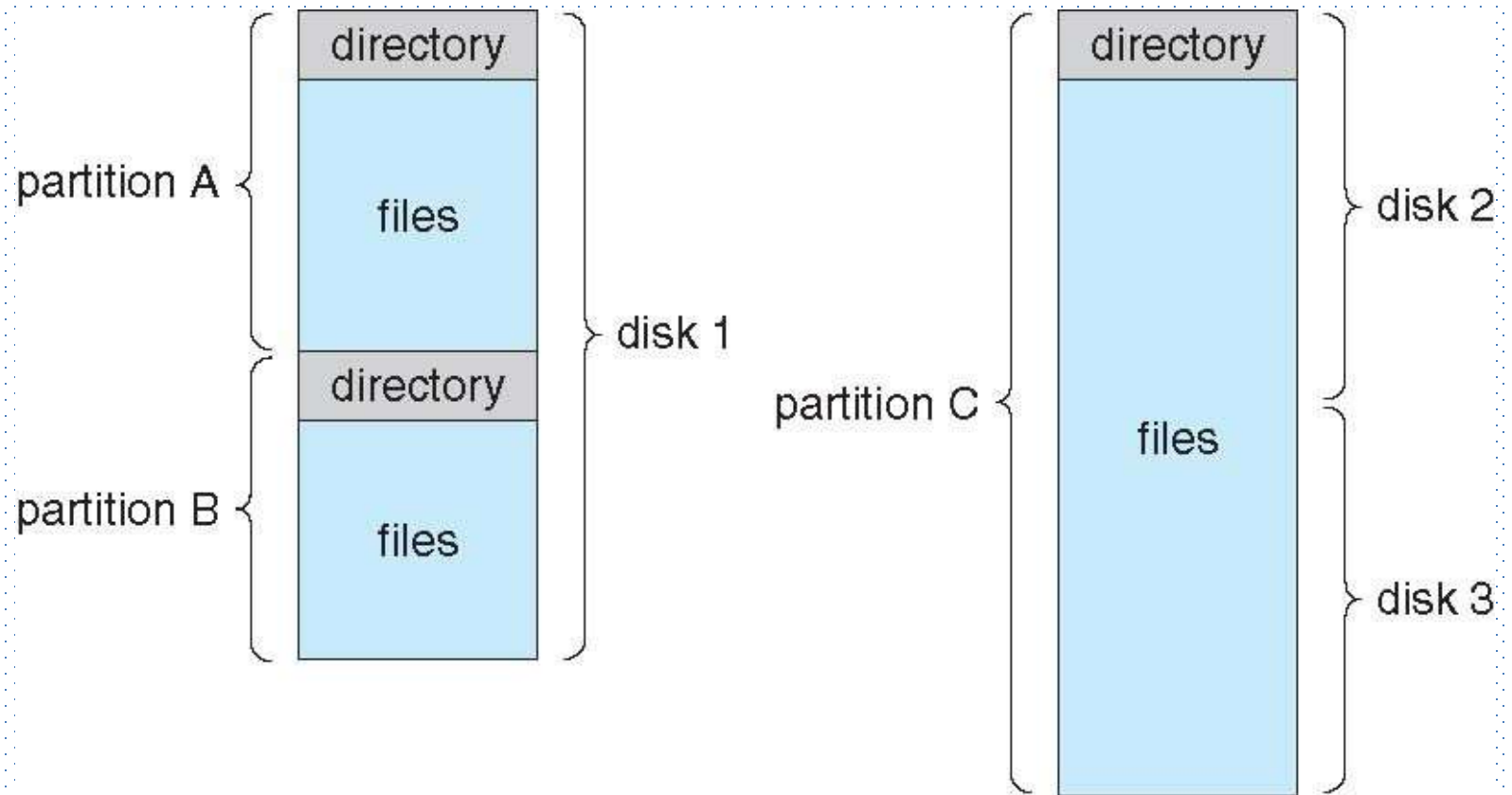
Example of Index and Relative Files



Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

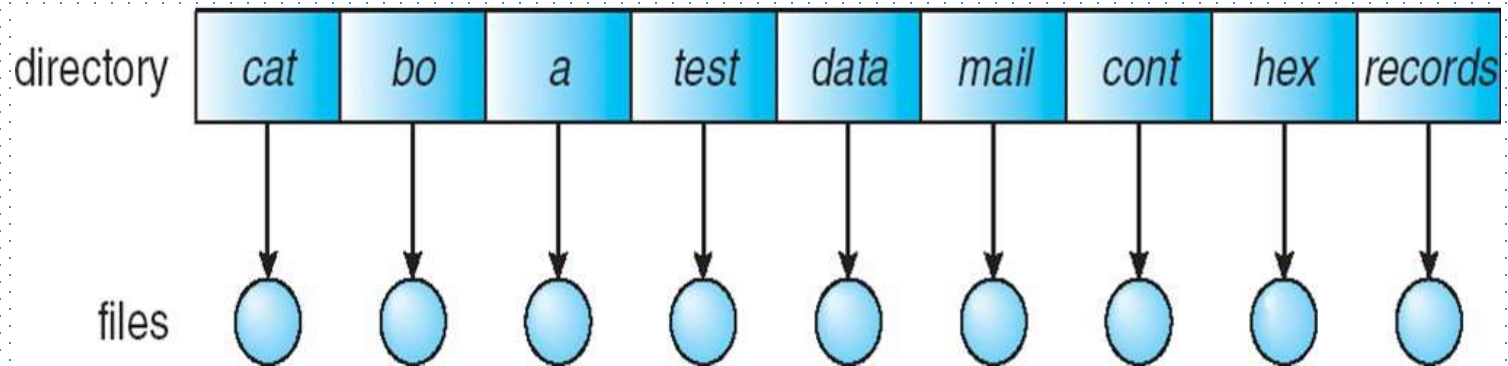
A Typical File-system Organization



- Efficiency – locating a file quickly
- Naming – convenient to users
 - Two users can have same name for different files
 - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, ...)

Single-Level Directory

- A single directory for all users

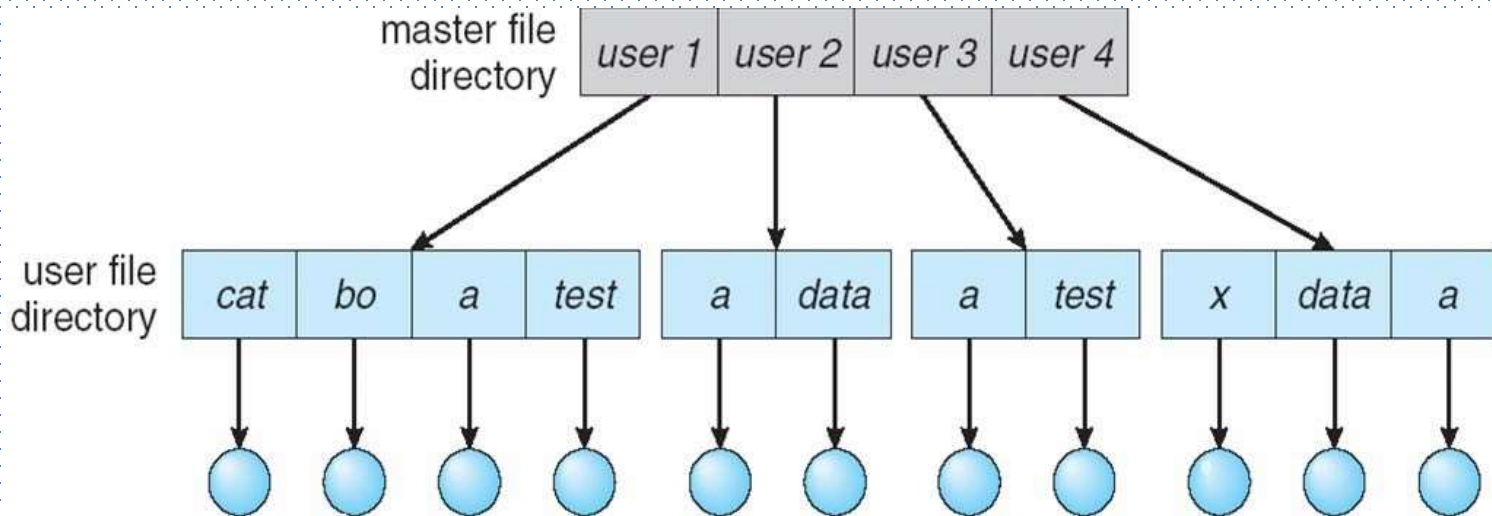


Naming problem

Grouping problem

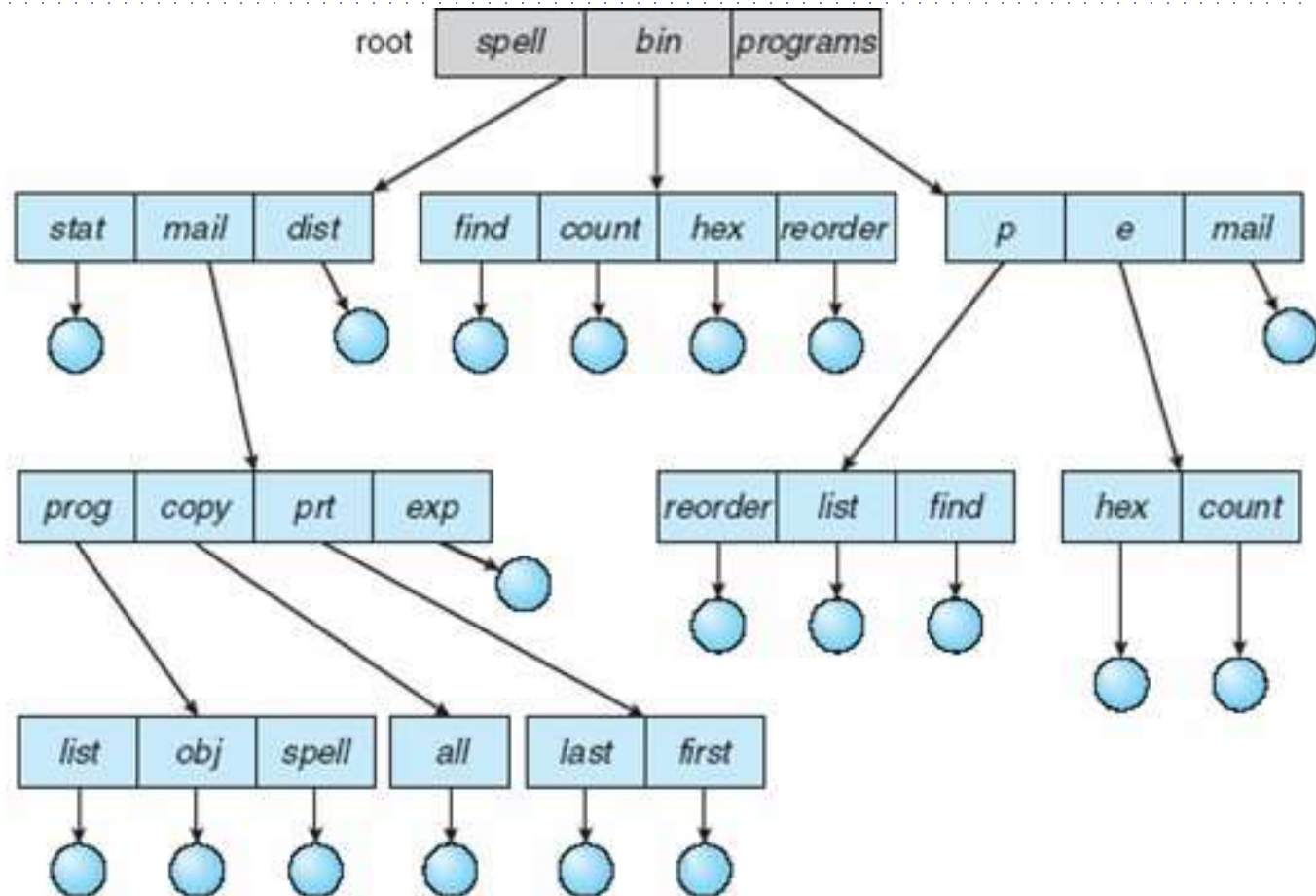
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



Tree-Structured Directories (Cont)

- Efficient searching
- Grouping Capability
- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

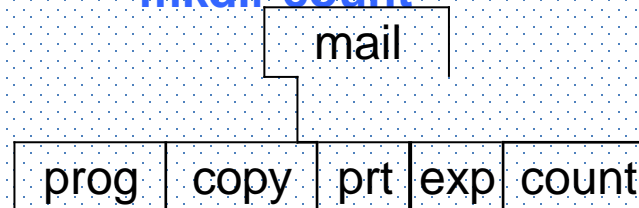
`rm <file-name>`

- Creating a new subdirectory is done in current directory

`mkdir <dir-name>`

Example: if in current directory `/mail`

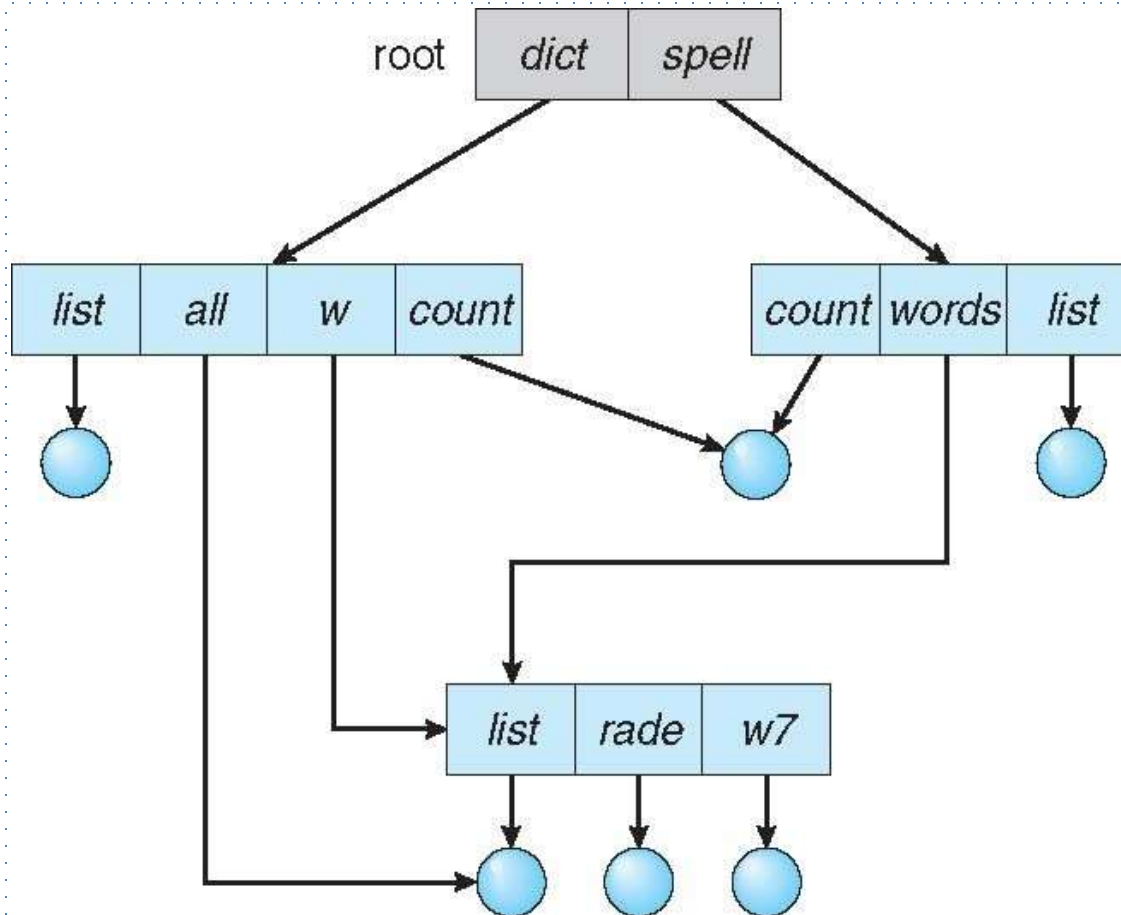
`mkdir count`



Deleting “mail” \Rightarrow deleting the entire subtree rooted by “mail”

Acyclic-Graph Directories

- Have shared subdirectories and files



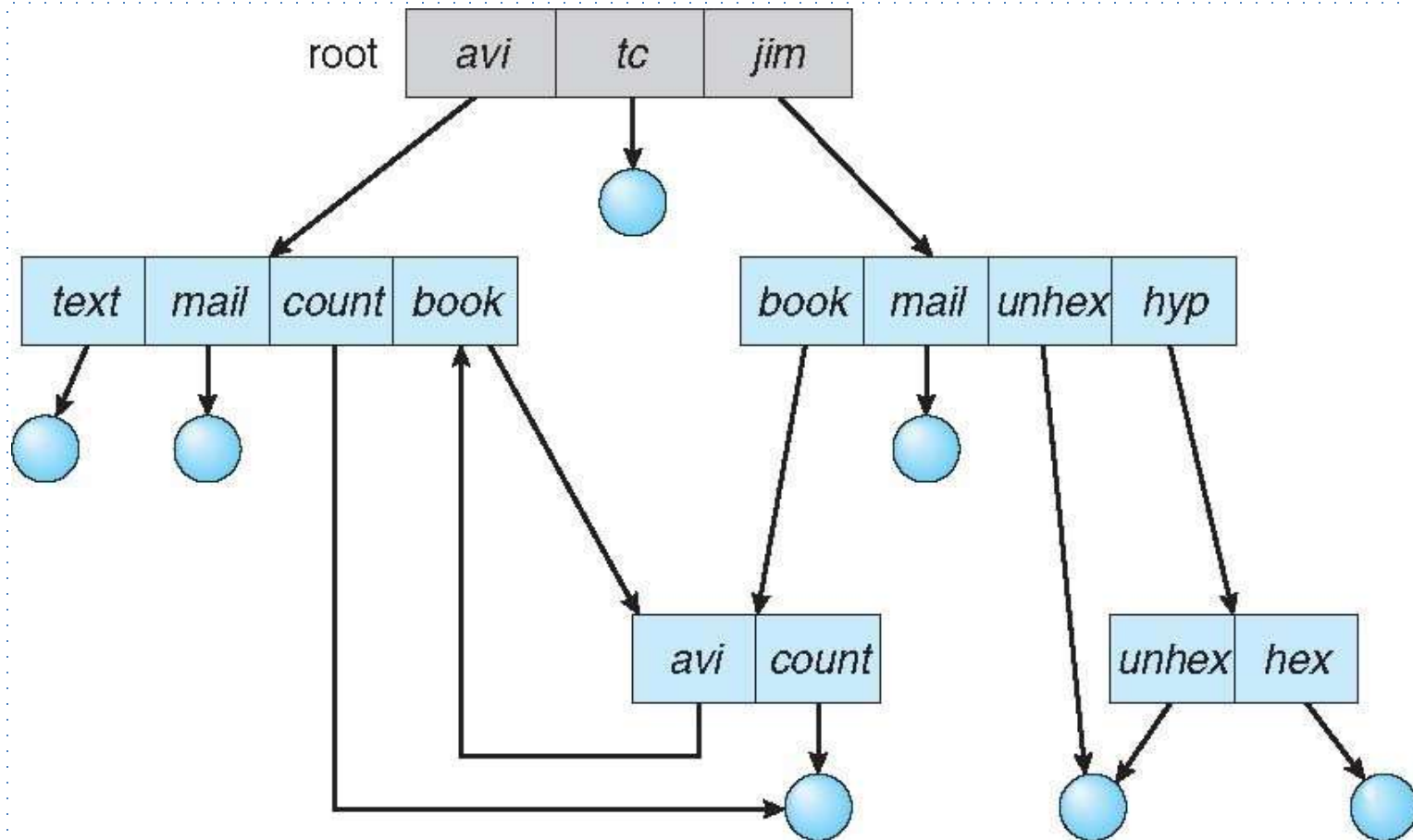
Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If *dict* deletes *list* \Rightarrow dangling pointer

Solutions:

- Backpointers, so we can delete all pointers
Variable size records a problem
 - Backpointers using a daisy chain organization
 - Entry-hold-count solution
-
- New directory entry type
 - **Link** – another name (pointer) to an existing file
 - **Resolve the link** – follow pointer to locate the file

General Graph Directory



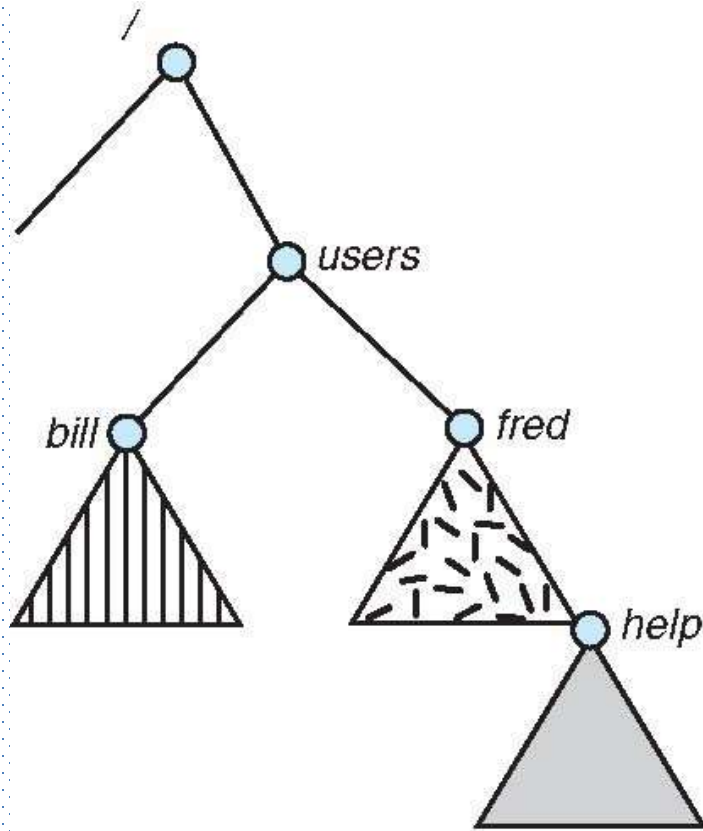
General Graph Directory (Cont.)

- How do we guarantee no cycles?
 - Allow only links to file not subdirectories
 - Garbage collection
 - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

Lecture : 6

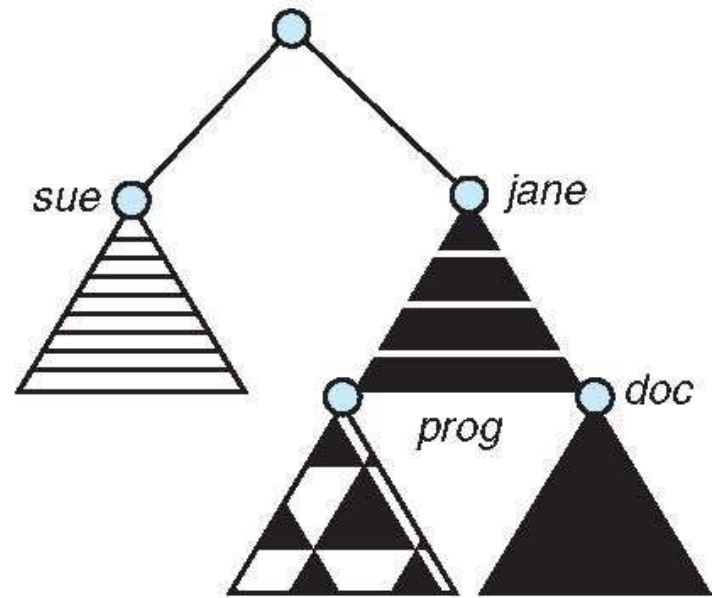
File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e., Fig. 11-11(b)) is mounted at a **mount point**



(a)

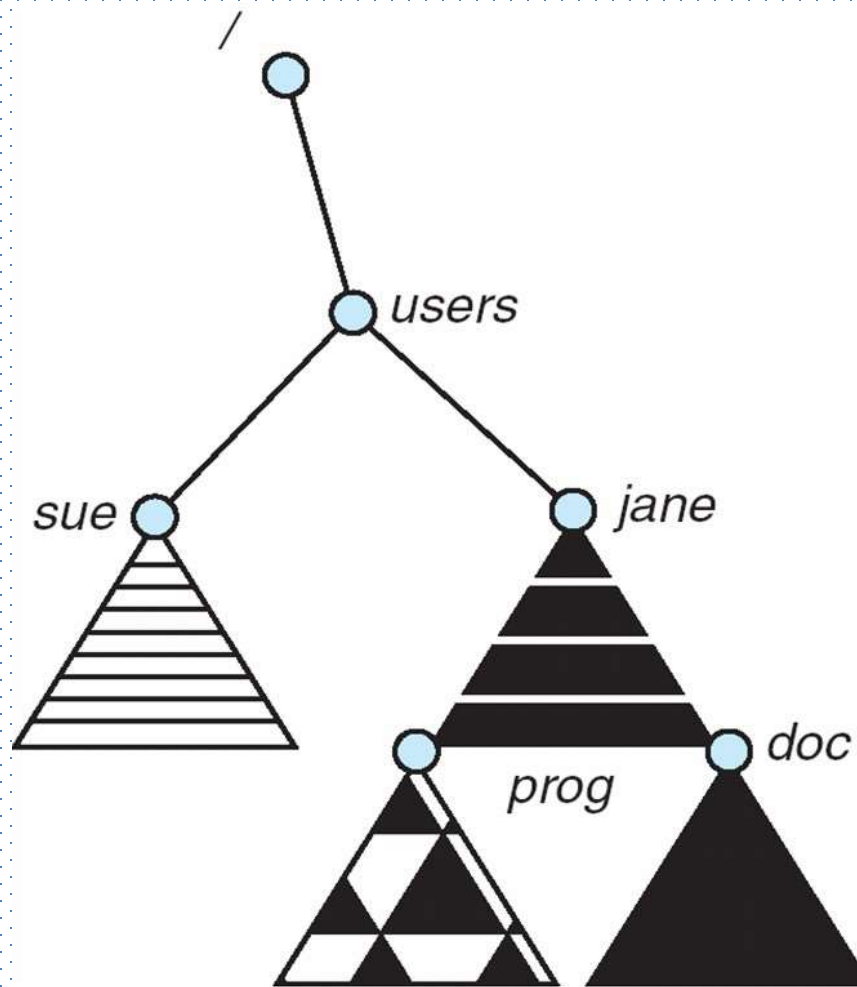
(a) Existing



(b)

(b) Unmounted Partition

Mount Point



File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method

- File Sharing – Multiple Users

- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** allow users to be in groups, permitting group access rights

File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
 - Manually via programs like FTP
 - Automatically, seamlessly using **distributed file systems**
 - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
 - Server can serve multiple clients
 - Client and user-on-client identification is insecure or complicated
 - **NFS** is standard UNIX client-server file sharing protocol
 - **CIFS** is standard Windows protocol
 - Standard operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

File Sharing – Failure Modes

- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security

File Sharing – Consistency Semantics

- **Consistency semantics** specify how multiple users are to access a shared file simultaneously
 - Similar to Ch 7 process synchronization algorithms
 - 🌲 Tend to be less complex due to disk I/O and network latency (for remote file systems)
 - Andrew File System (AFS) implemented complex remote file sharing semantics
 - Unix file system (UFS) implements:
 - 🌲 Writes to an open file visible immediately to other users of the same open file
 - 🌲 Sharing file pointer to allow multiple users to read and write concurrently
 - AFS has session semantics
 - 🌲 Writes only visible to sessions starting after the file is closed

Protection

- File owner/creator should be able to control:
 - what can be done
 - by whom

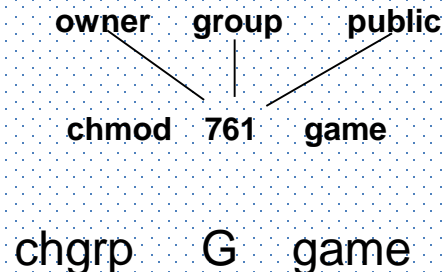
- Types of access
 - **Read**
 - **Write**
 - **Execute**
 - **Append**
 - **Delete**
 - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

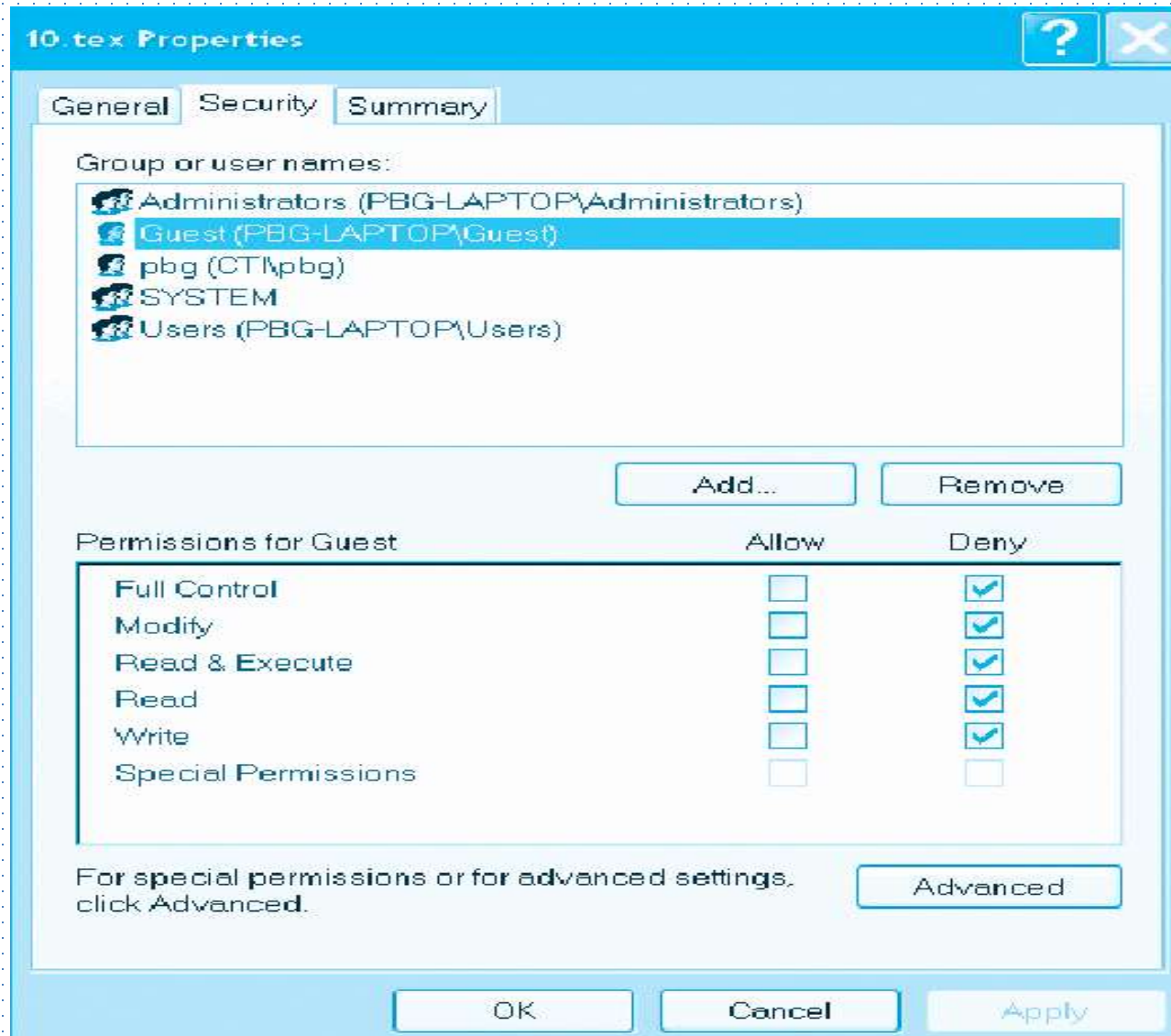
a) owner access	7	⇒	RWX 1 1 1
b) group access	6	⇒	RWX 1 1 0
c) public access	1	⇒	RWX 0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

Windows XP Access-Control List Management



A Sample UNIX Directory Listing

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx-----	5 pbg	staff	512	Jul 8 09:33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx-----	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

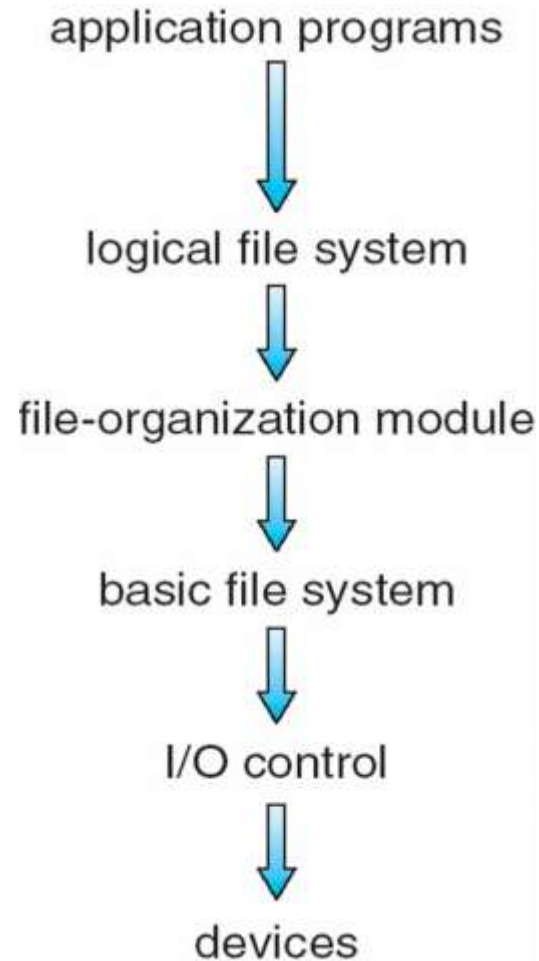
Chapter 11: File System Implementation

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

File-System Structure

- File structure
 - Logical storage unit
 - Collection of related information
- File system organized into layers
- **File system** resides on secondary storage (disks)
 - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- **File control block** – storage structure consisting of information about a file
- **Device driver** controls the physical device

Layered File System



File-System Implementation

- **Boot control block** contains info needed by system to boot OS from that volume
- **Volume control block** contains volume details
- Directory structure organizes the files
- Per-file **File Control Block (FCB)** contains many details about the file

A Typical File Control Block

file permissions

file dates (create, access, write)

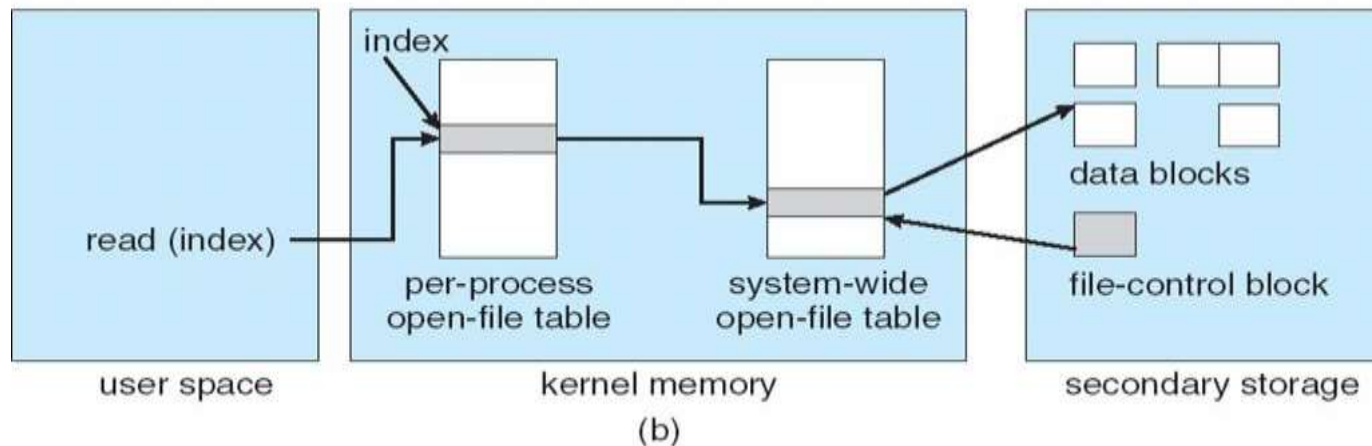
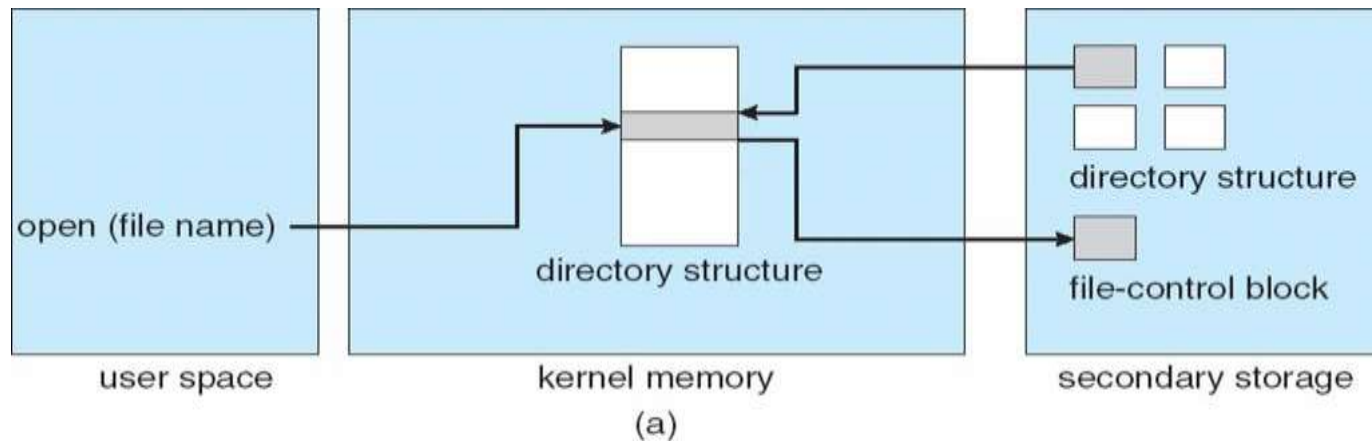
file owner, group, ACL

file size

file data blocks or pointers to file data blocks

In-Memory File System Structures

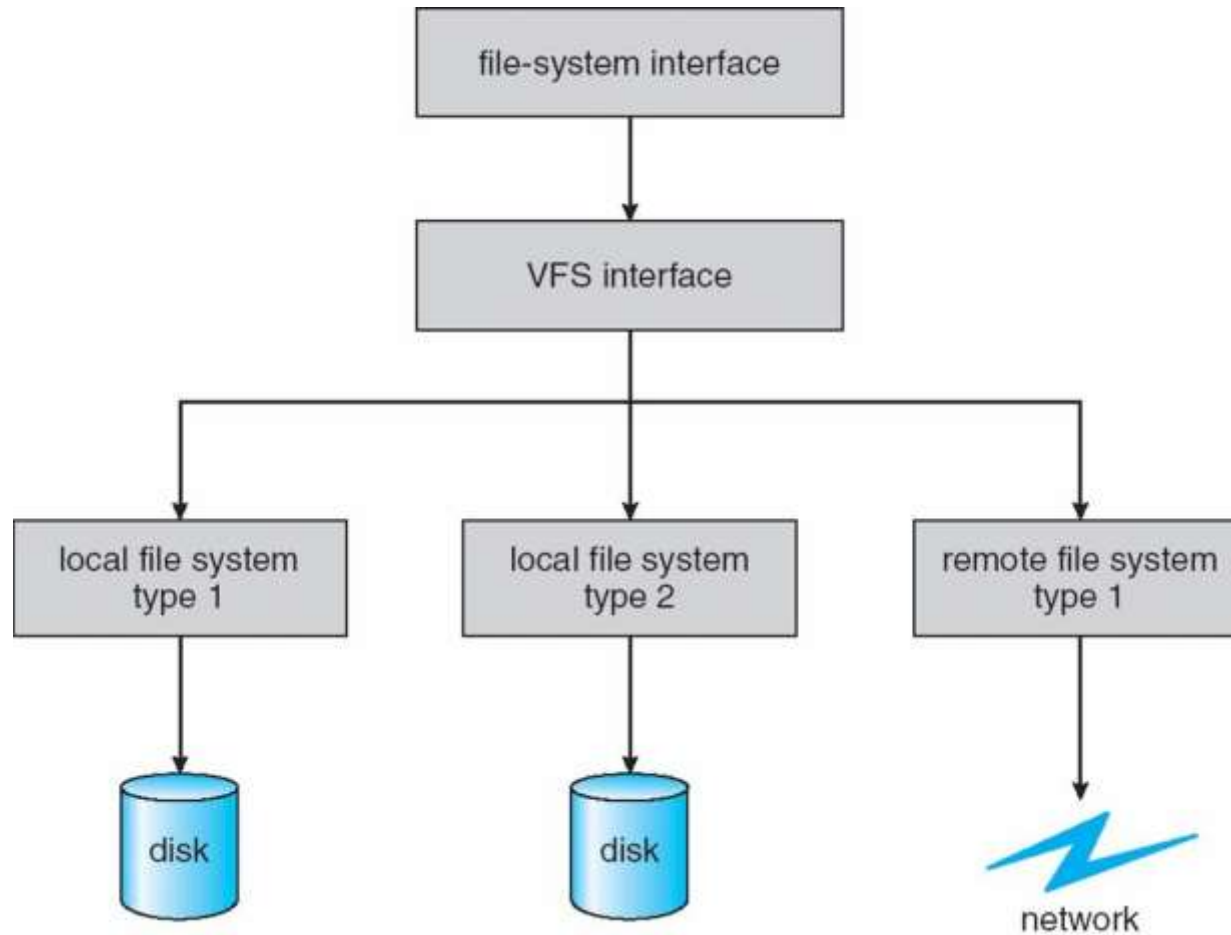
- The following figure illustrates the necessary file system structures provided by the operating systems.
- Figure 12-3(a) refers to opening a file. Figure 12-3(b) refers to reading a file.



Virtual File Systems

- Virtual File Systems (VFS) 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.
- The API is to the VFS interface, rather than any specific type of file system.

Schematic View of Virtual File System



Directory Implementation

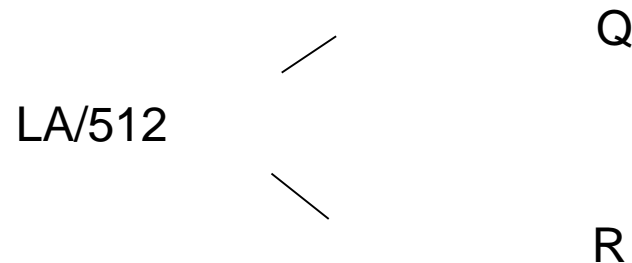
- **Linear list** of file names with pointer to the data blocks.
 - simple to program
 - time-consuming to execute
- **Hash Table** – linear list with hash data structure.
 - decreases directory search time
 - **collisions** – situations where two file names hash to the same location
 - fixed size

Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

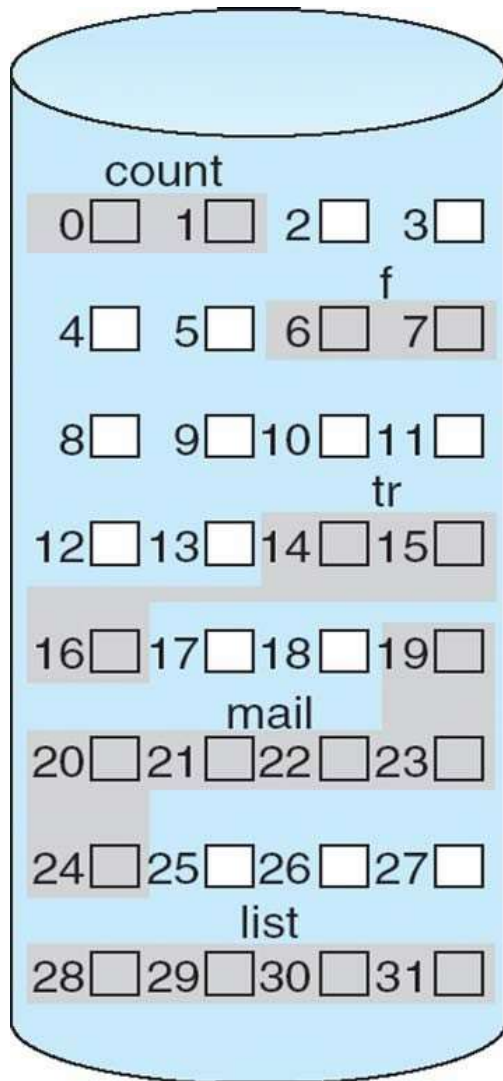
Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Random access
- Wasteful of space (dynamic storage-allocation problem)
- Files cannot grow
- Mapping from logical to physical



Block to be accessed = ! + starting address Displacement into block = R

Contiguous Allocation of Disk Space



directory

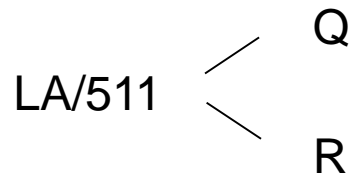
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

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

Linked Allocation (Cont.)

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.
- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping

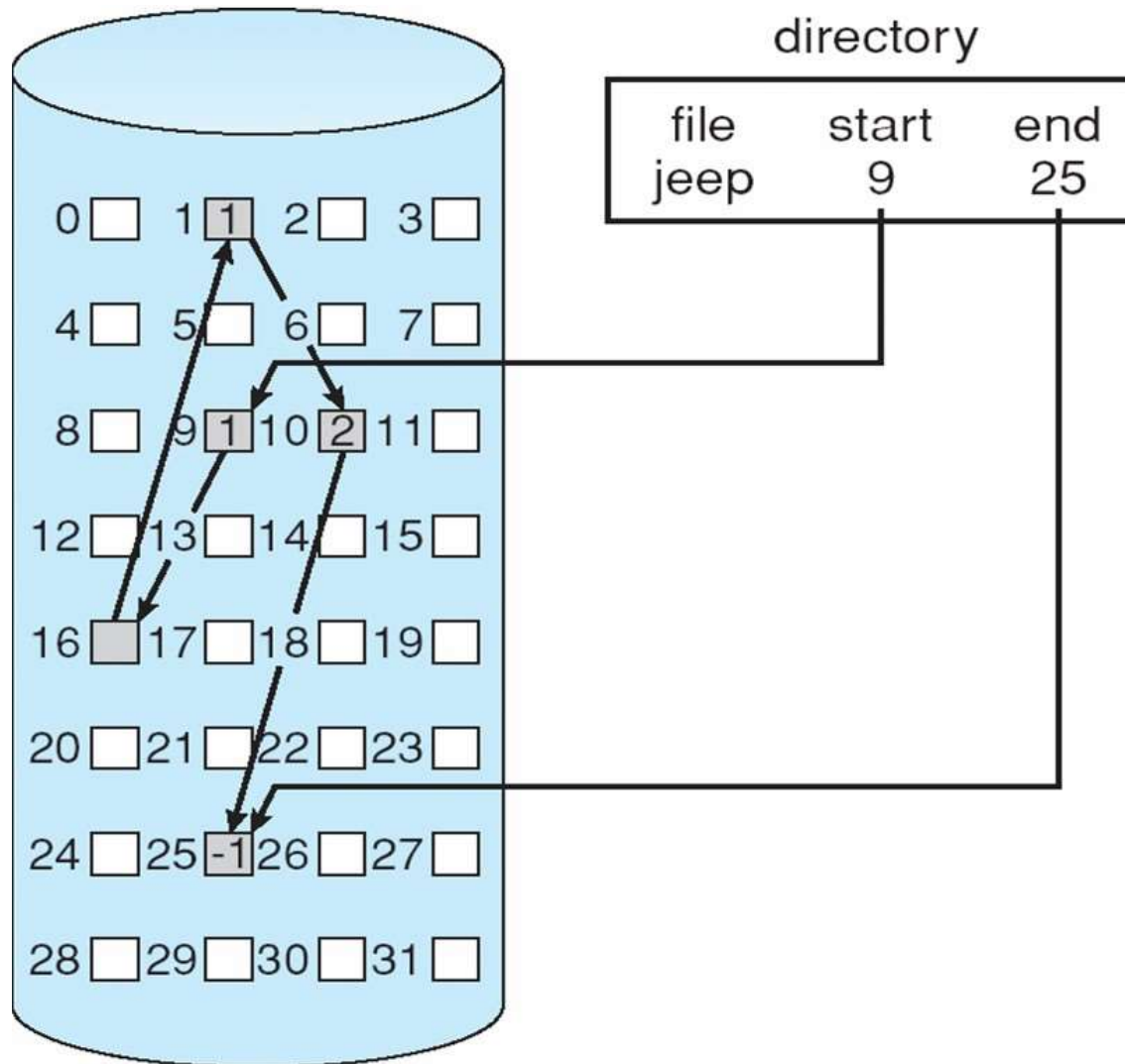


Block to be accessed is the Qth block in the linked chain of blocks representing the file.

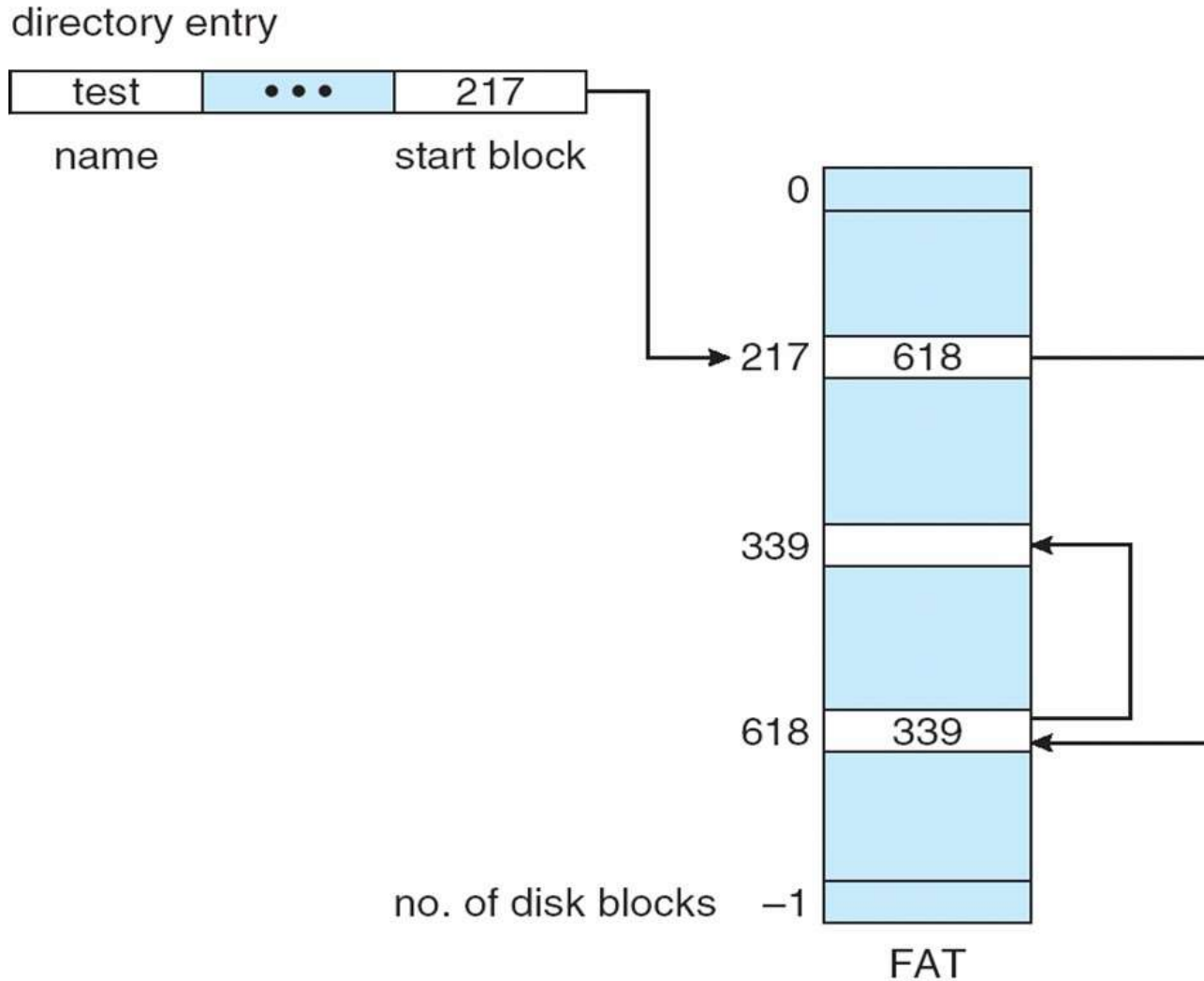
Displacement into block = $R + 1$

File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

Linked Allocation

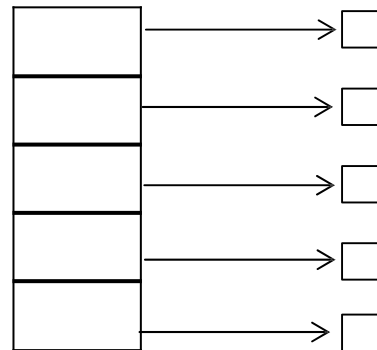


File-Allocation Table



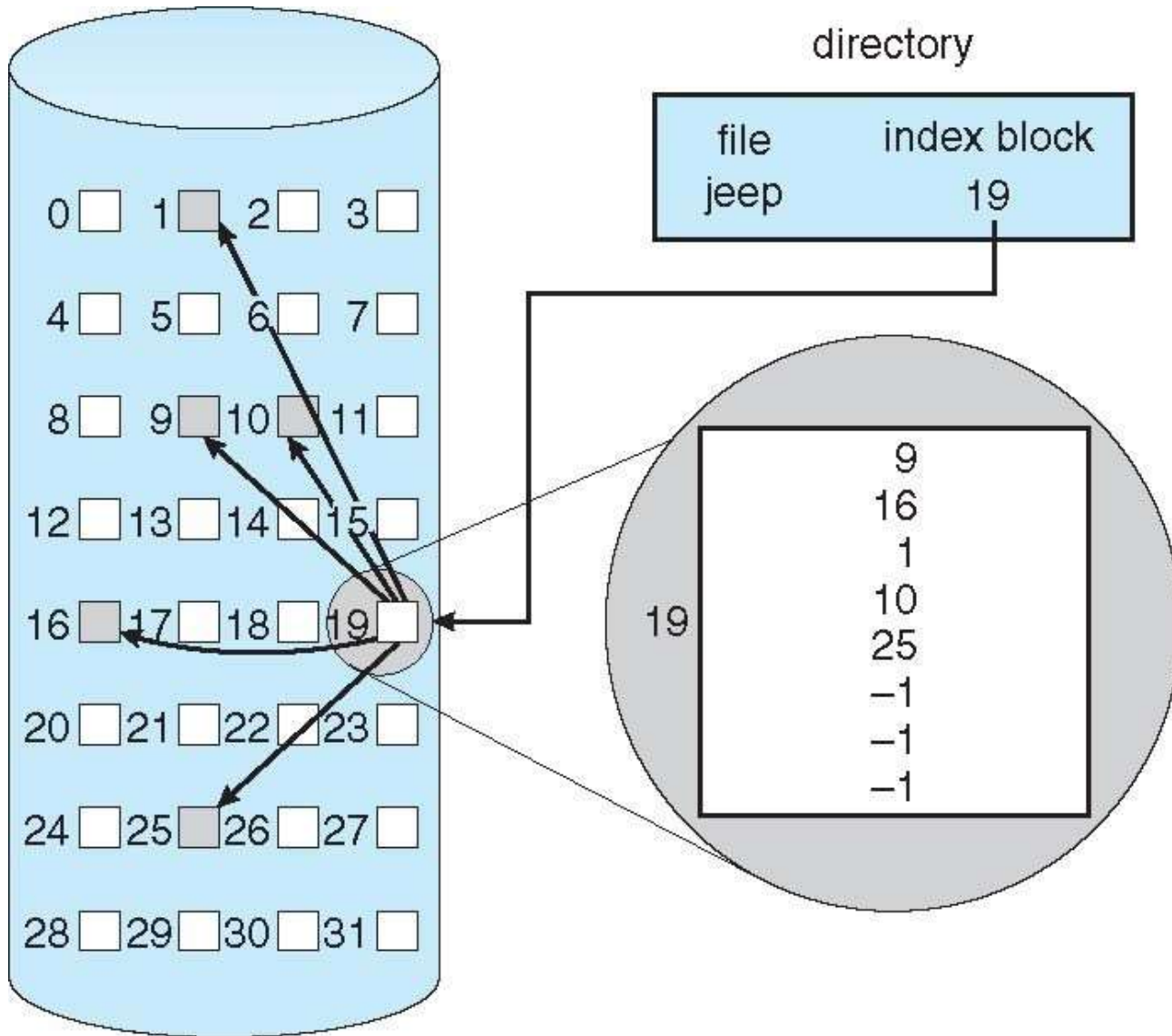
Indexed Allocation

- Brings all pointers together into the **index block**
- Logical view



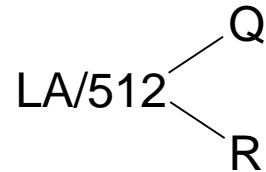
index table

Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table

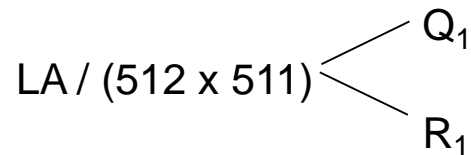


Q = displacement into index table

R = displacement into block

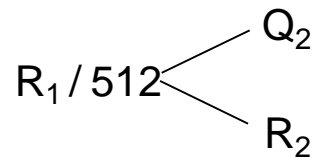
Indexed Allocation – Mapping (Cont.)

- Mapping from logical to physical in a file of unbounded length (block size of 512 words)
- Linked scheme – Link blocks of index table (no limit on size)



Q_1 = block of index table

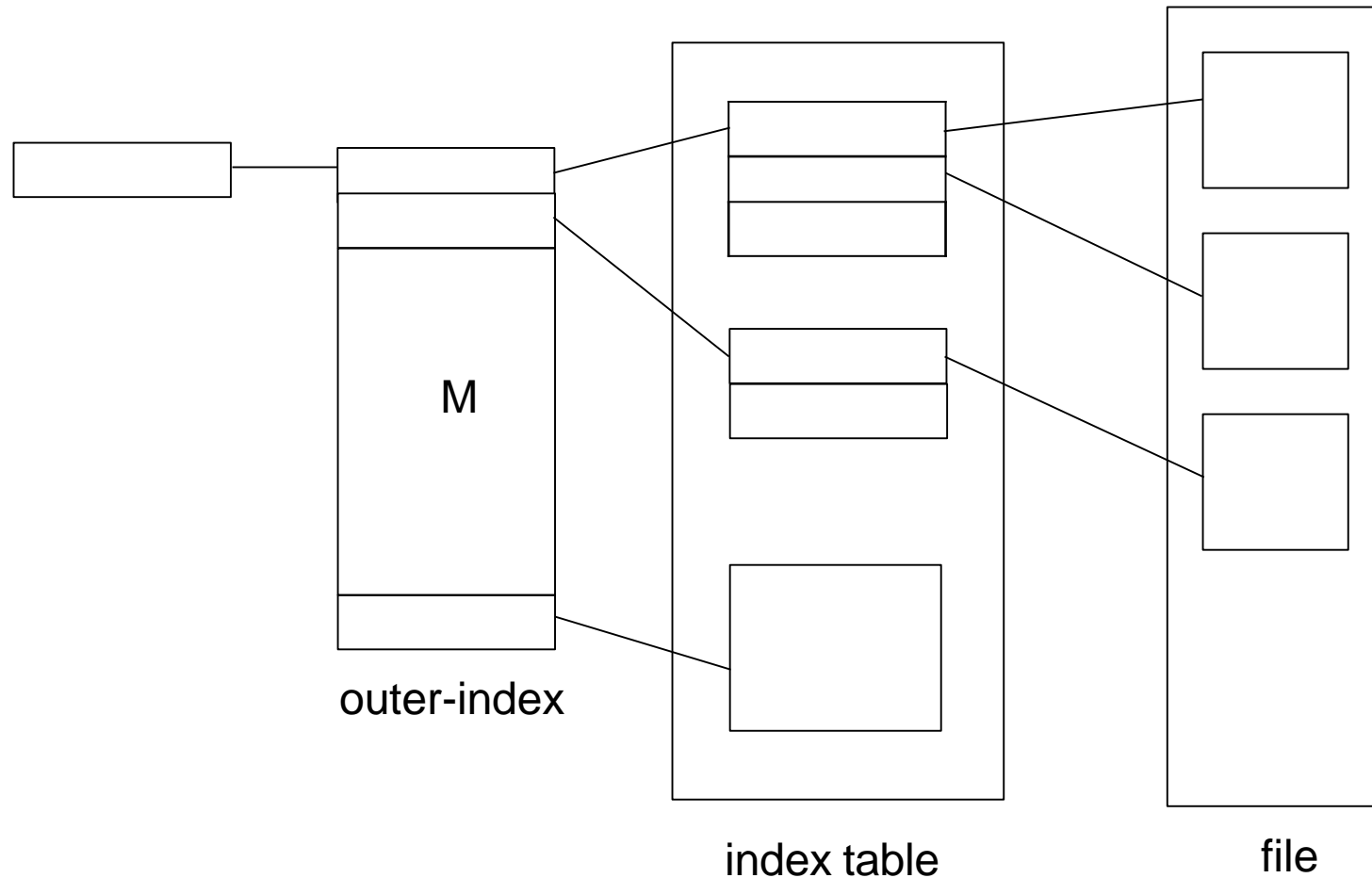
R_1 is used as follows:



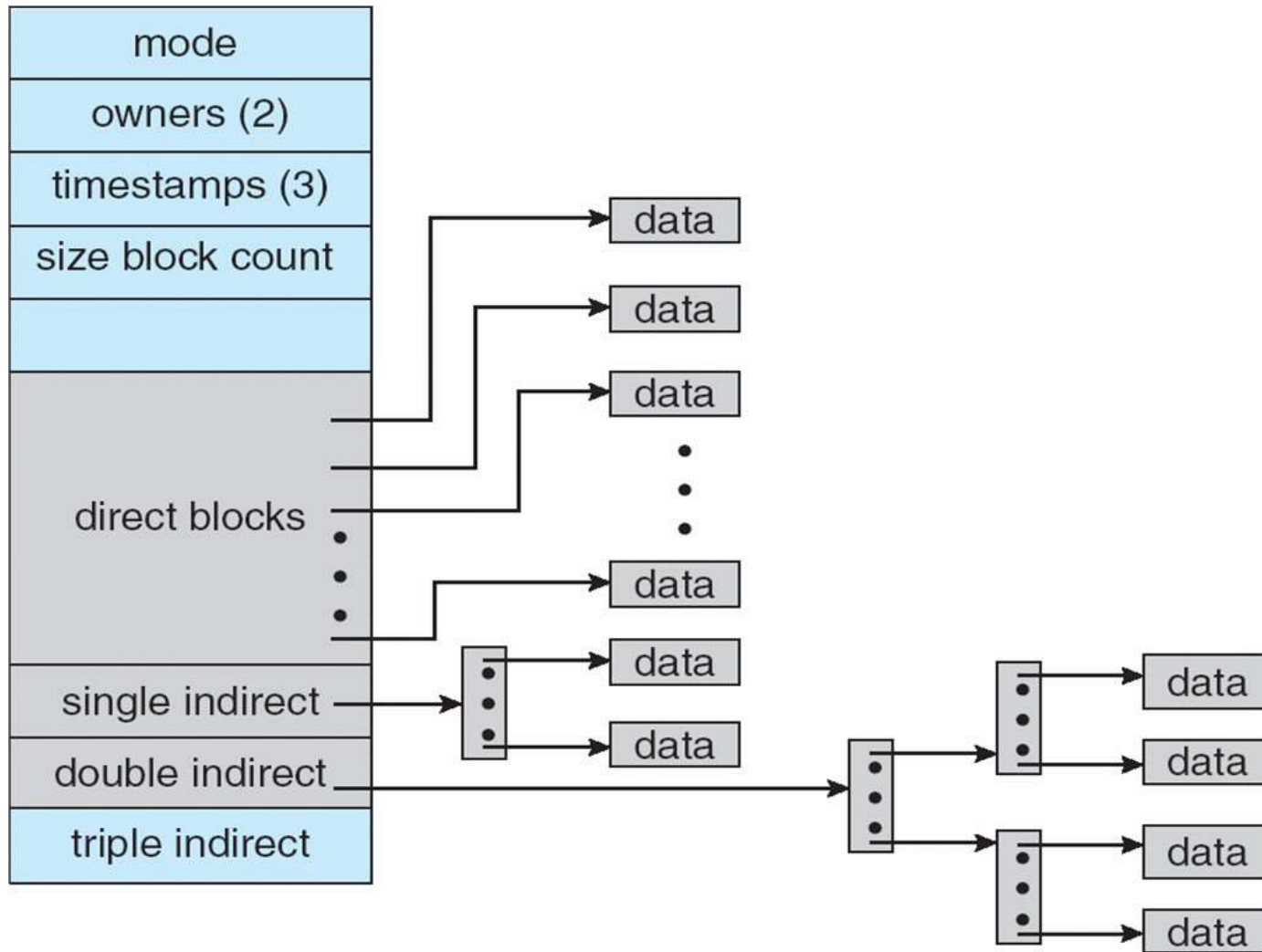
Q_2 = displacement into block of index table

R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)

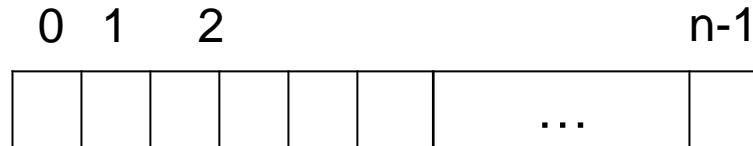


Combined Scheme: UNIX UFS (4K bytes per block)



Free-Space Management

- Bit vector (n blocks)



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

Block number calculation

(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

Free-Space Management (Cont.)

- Bit map requires extra space

- Example:

- block size = 2^{12} bytes

- disk size = 2^{30} bytes (1 gigabyte)

- $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

- Easy to get contiguous files

- Linked list (free list)

- Cannot get contiguous space easily
 - No waste of space

- Grouping

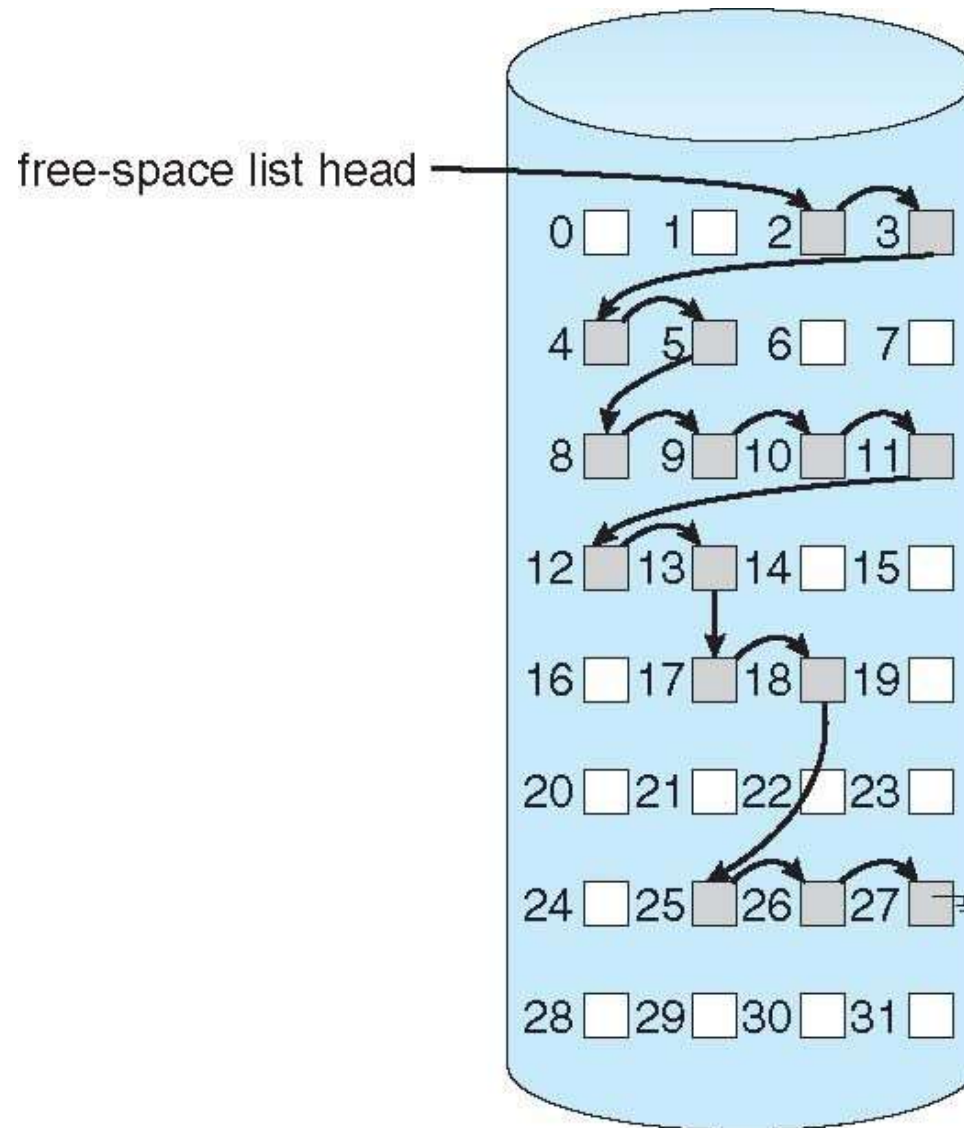
- Counting

Free-Space Management (Cont.)

■ Need to protect:

- Pointer to free list
- Bit map
 - Must be kept on disk
 - Copy in memory and disk may differ
 - Cannot allow for block[i] to have a situation where $\text{bit}[i] = 1$ in memory and $\text{bit}[i] = 0$ on disk
- Solution:
 - Set $\text{bit}[i] = 1$ in disk
 - Allocate block[i]
 - Set $\text{bit}[i] = 1$ in memory

Linked Free Space List on Disk



Directory Implementation

- Linear list of file names with pointer to the data blocks
 - simple to program
 - time-consuming to execute
- Hash Table – linear list with hash data structure
 - decreases directory search time
 - **collisions** – situations where two file names hash to the same location
 - fixed size

Efficiency and Performance

- Efficiency dependent on:
 - disk allocation and directory algorithms
 - types of data kept in file's directory entry

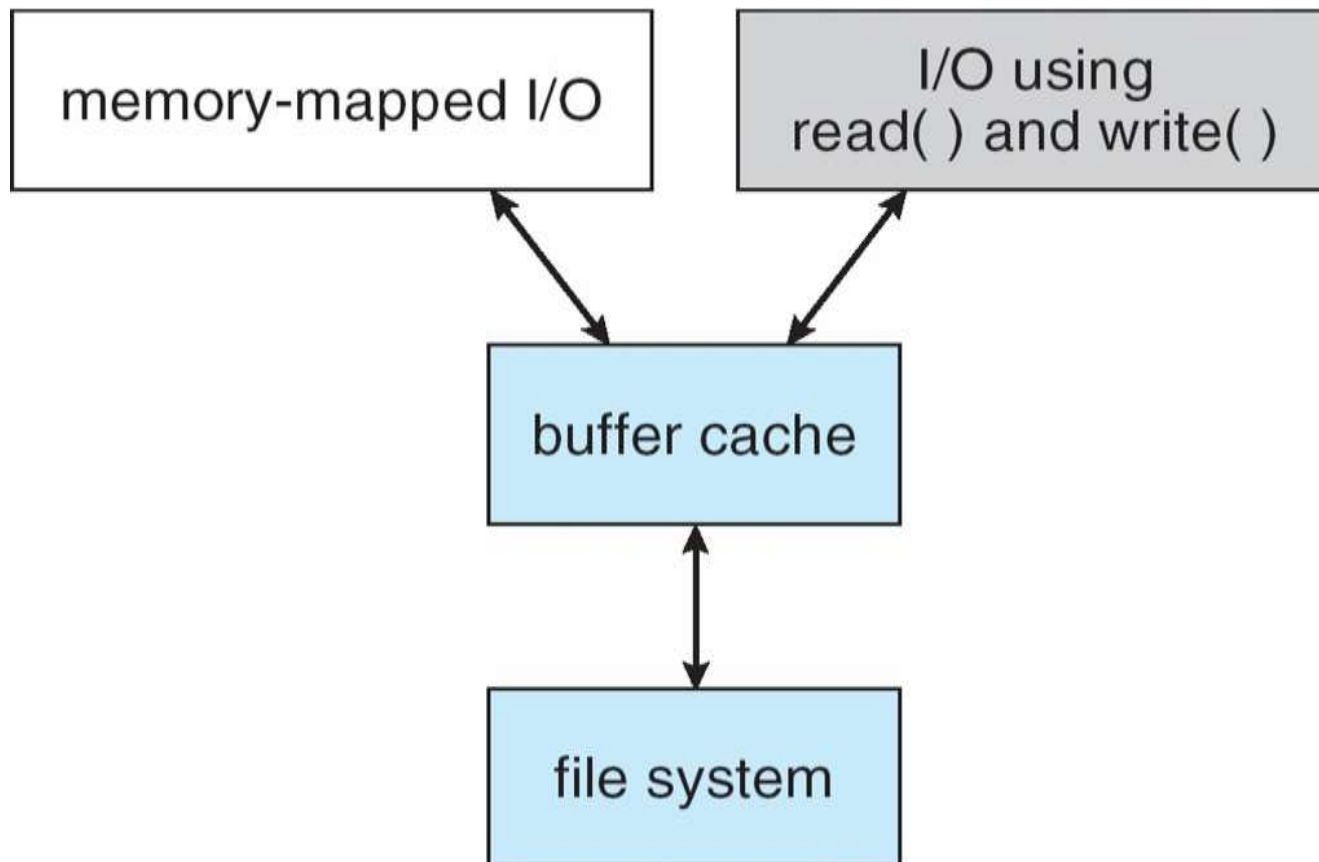
- Performance
 - disk cache – separate section of main memory for frequently used blocks
 - free-behind and read-ahead – techniques to optimize sequential access
 - improve PC performance by dedicating section of memory as virtual disk, or RAM disk

Page Cache

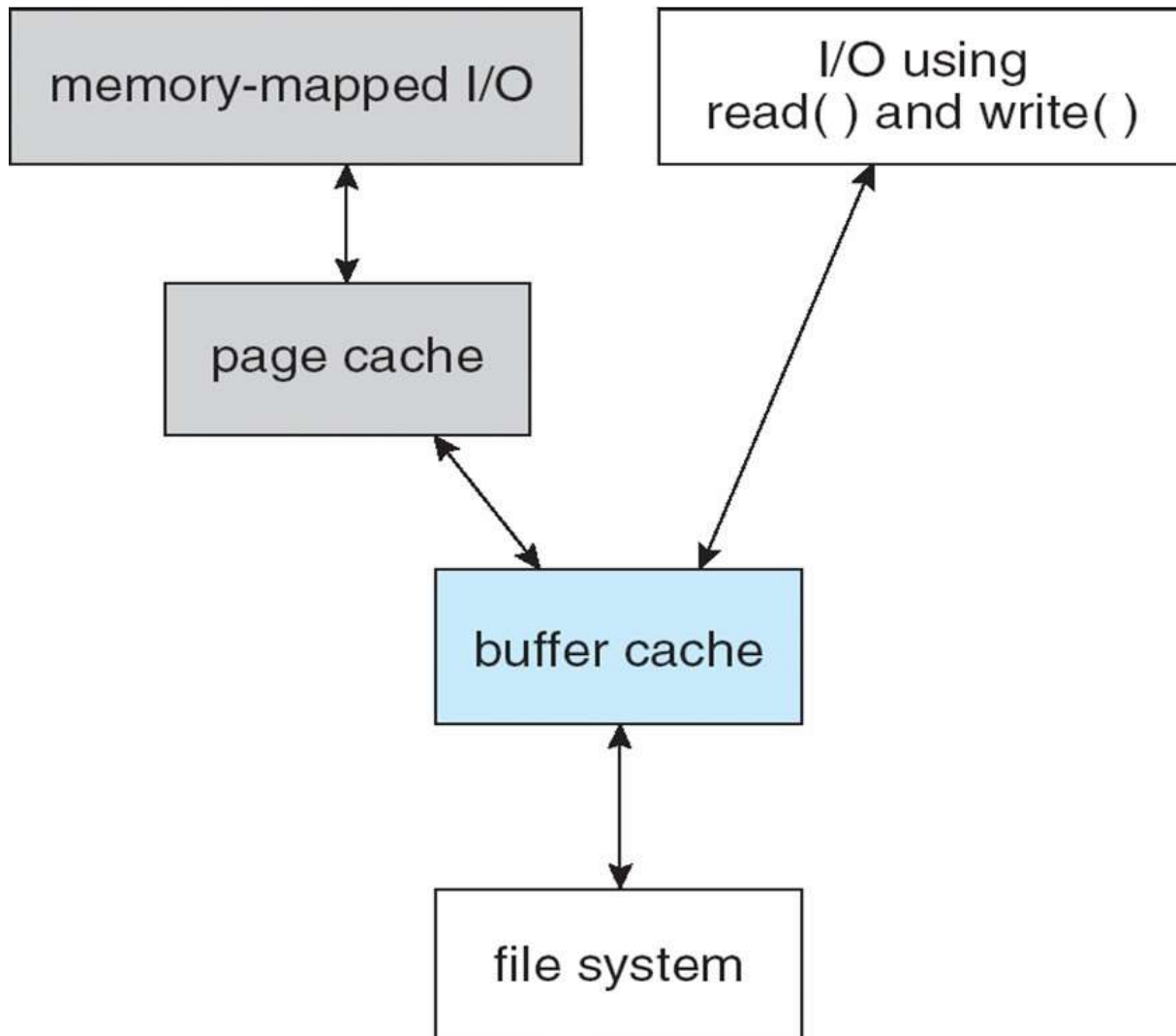
- A **page cache** caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure

I/O Using a Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O



I/O Without a Unified Buffer Cache



Recovery

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- Use system programs to **back up** data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup

Log Structured File Systems

- **Log structured** (or **journaling**) file systems record each 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
 - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system
 - When the file system is modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed

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)

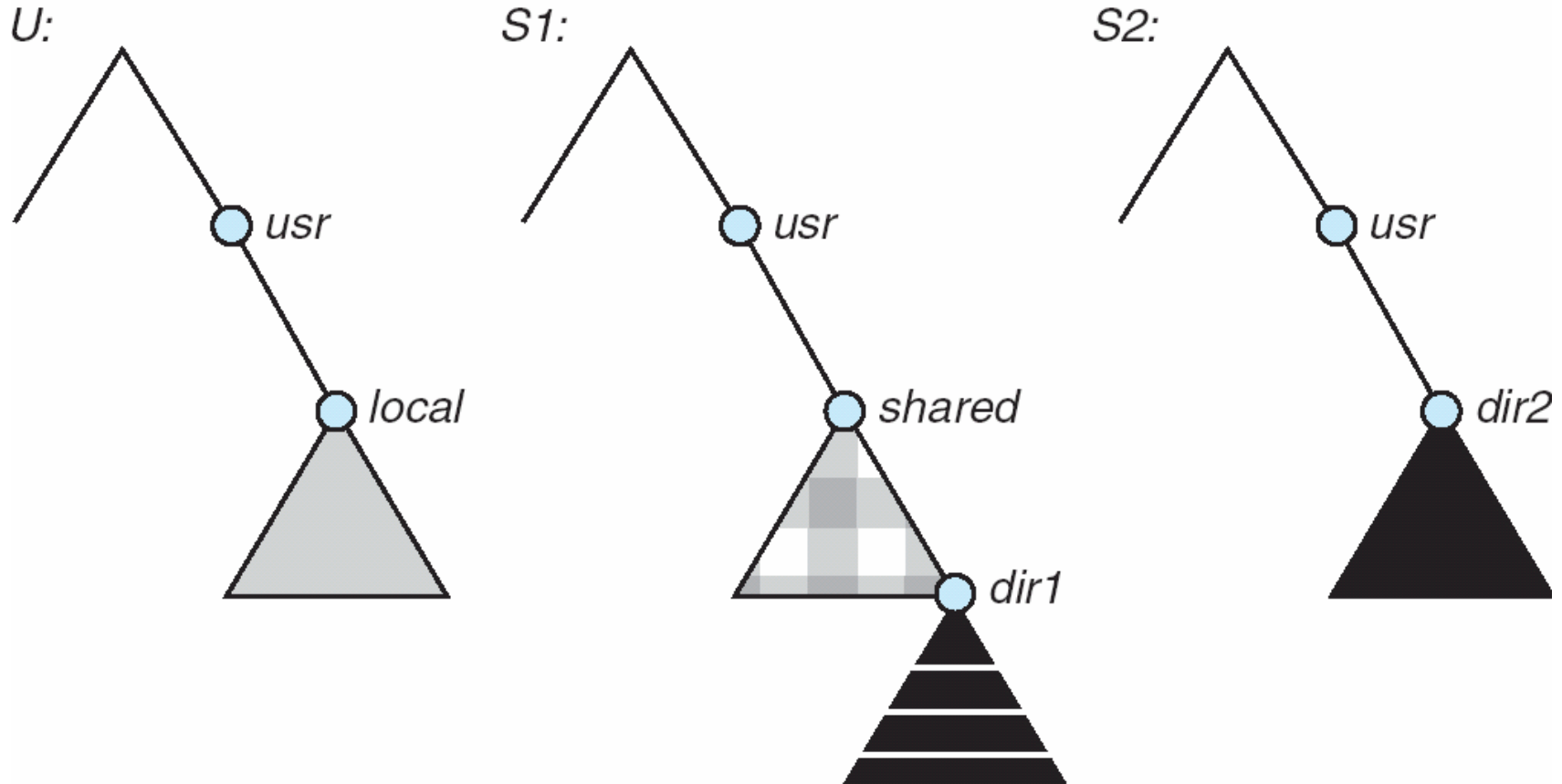
NFS (Cont.)

- 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

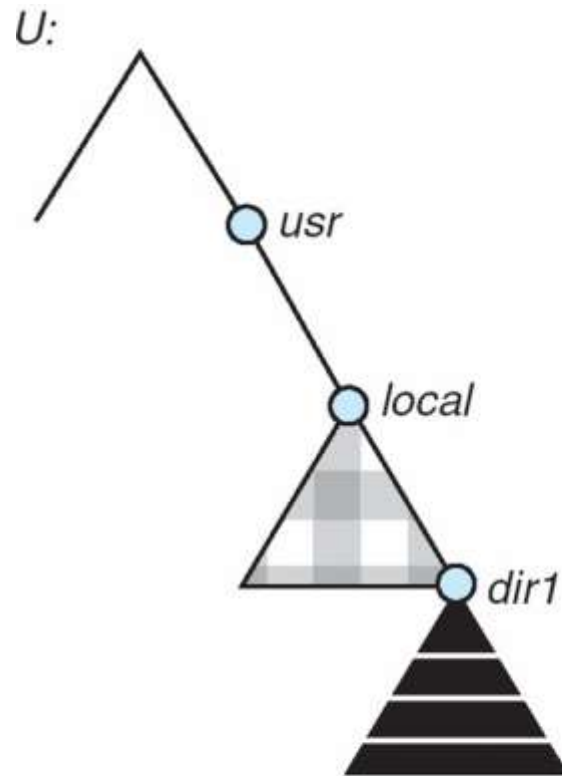
NFS (Cont.)

- 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 Independent File Systems

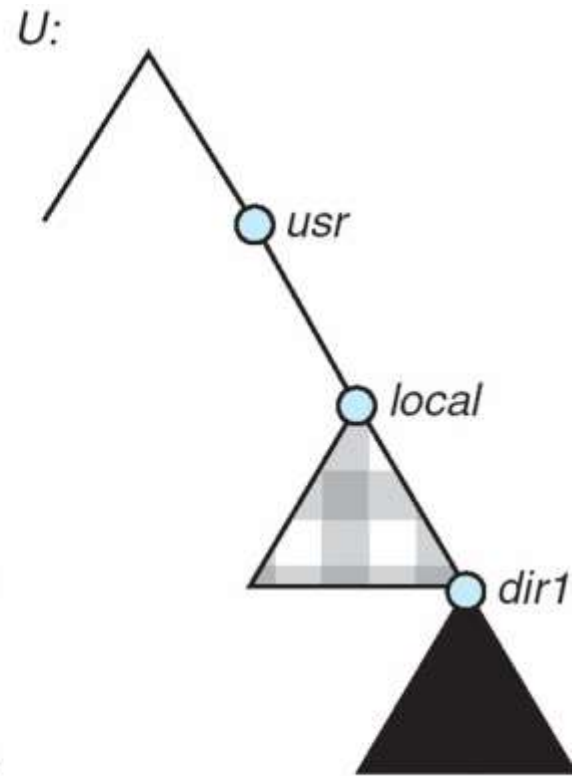


Mounting in NFS



(a)

Mounts



(b)

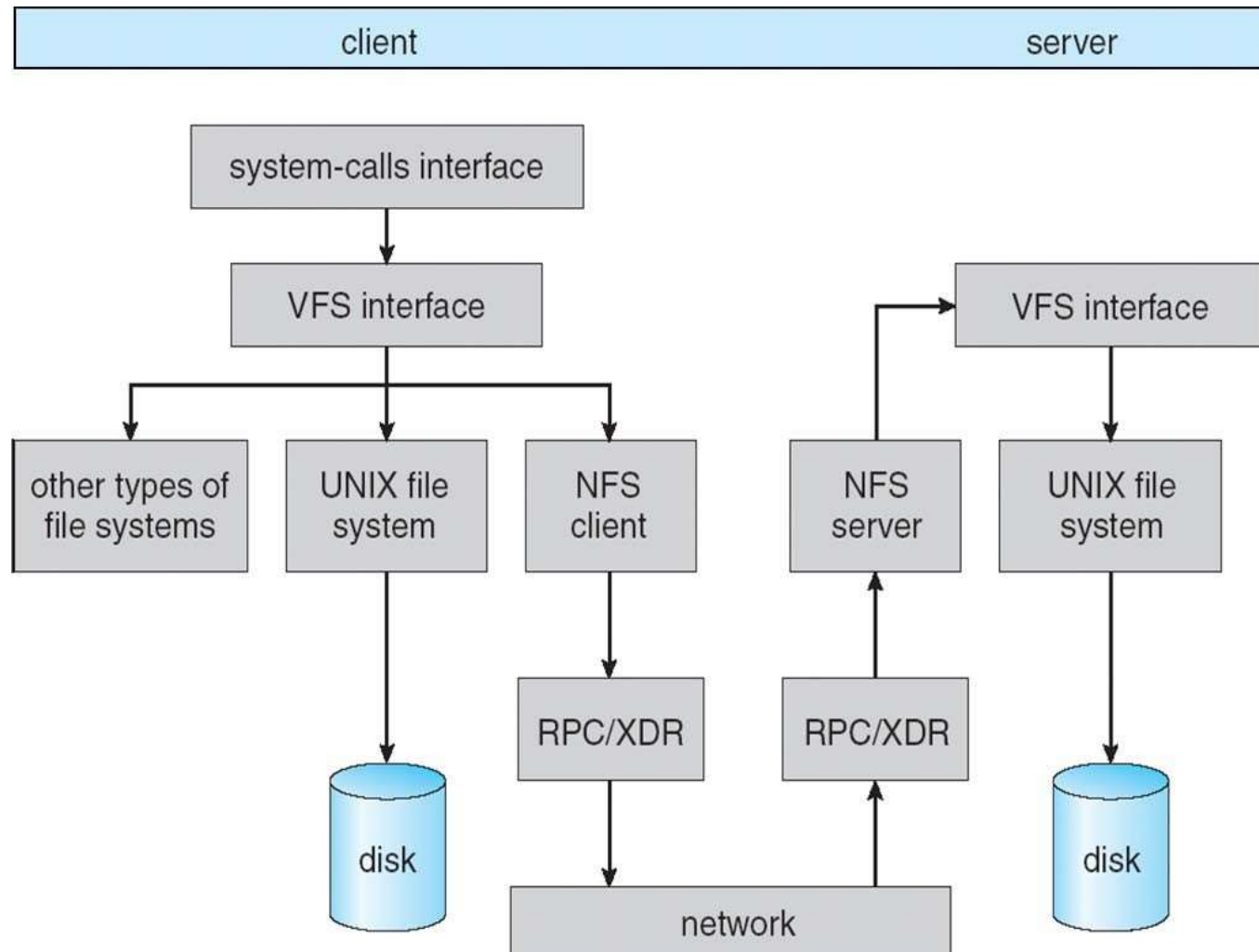
Cascading mounts

- Provides a set of remote procedure calls for remote file operations. The procedures support the following operations:
 - searching for a file within a directory
 - reading a set of directory entries
 - manipulating links and directories
 - accessing file attributes
 - reading and writing files
- NFS servers are **stateless**; each request has to provide a full set of arguments (NFS V4 is just coming available – very different, stateful)
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- The NFS protocol does not provide concurrency-control mechanisms
- Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode
- To make lookup faster, a directory name lookup cache on the client's side holds the vnodes for remote directory names

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

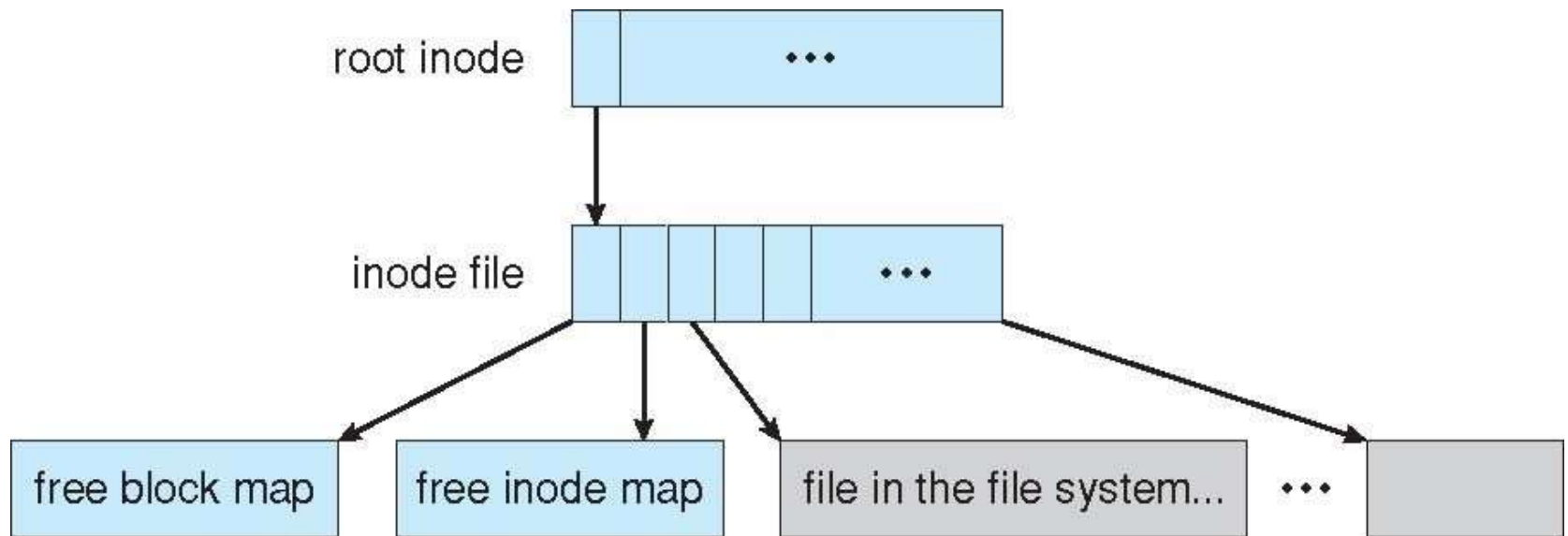
Schematic View of NFS Architecture



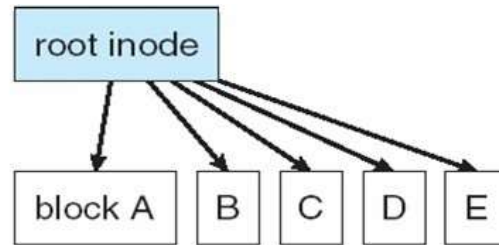
Example: WAFL File System

- Used on Network Appliance “Filers” – distributed file system appliances
- “Write-anywhere file layout”
- Serves up NFS, CIFS, http, ftp
- Random I/O optimized, write optimized
 - NVRAM for write caching
- Similar to Berkeley Fast File System, with extensive modifications

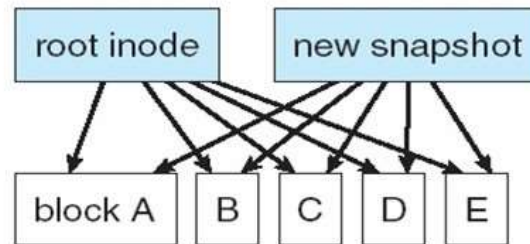
The WAFL File Layout



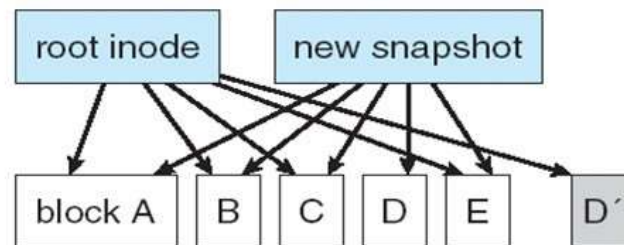
Snapshots in WAFL



(a) Before a snapshot.



(b) After a snapshot, before any blocks change.



(c) After block D has changed to D'.