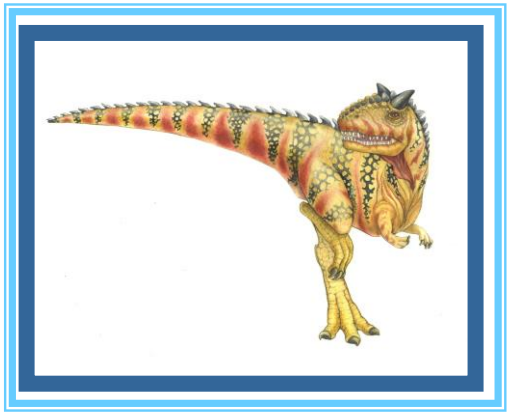
Chapter 10
File-System



Chapter 10: File System

File Concept

Access Methods

Directory Structure

File-System Mounting

File Sharing

Protection

Objectives

To explain the function of file systems

To describe the interfaces to file systems

To discuss file-system design tradeoffs, including

- access methods,
- file sharing,
- file locking, and
- directory structures

To explore file-system protection

File Concept

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 methods with first method by inserting appropriate control characters (CR, LF)

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 process 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

Shared Lock: several processes can acquire the lock concurrently (like a reader lock)

Exclusive Lock: Only one process at a time can acquire such a lock (like a writer lock)

Mandatory or advisory file locking mechanisms:

Mandatory – Once a process acquires an exclusive lock, the OS will prevent any other process from accessing the locked file. (Windows)

Advisory – The OS will not prevent a process from acquiring access to a locked file. Rather, the process must be written so that it manually acquiring the lock before accessing the file. (UNIX)

File Locking Example – Java API

```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
    public static final boolean EXCLUSIVE = false;
    public static final boolean SHARED = true;
    public static void main(String arsg[]) throws IOException {
        FileLock sharedLock = null;
        FileLock exclusiveLock = null;
        try {
            RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");
            // get the channel for the file
            FileChannel ch = raf.getChannel();
            // this locks the first half of the file - exclusive
            exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);

            /** Now modify the data . . . */

            // release the lock
            exclusiveLock.release();
```

File Locking Example – Java API (cont)

```
            // this locks the second half of the file - shared
            sharedLock = ch.lock(raf.length()/2+1, raf.length(), SHARED);

            /** Now read the data . . . */

            // release the lock
            sharedLock.release();
        } catch (java.io.IOException ioe) {
            System.err.println(ioe);
        }finally {
            if (exclusiveLock != null)
            exclusiveLock.release();
            if (sharedLock != null)
            sharedLock.release();
        }
    }
}
```

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 compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Access Methods

Sequential Access

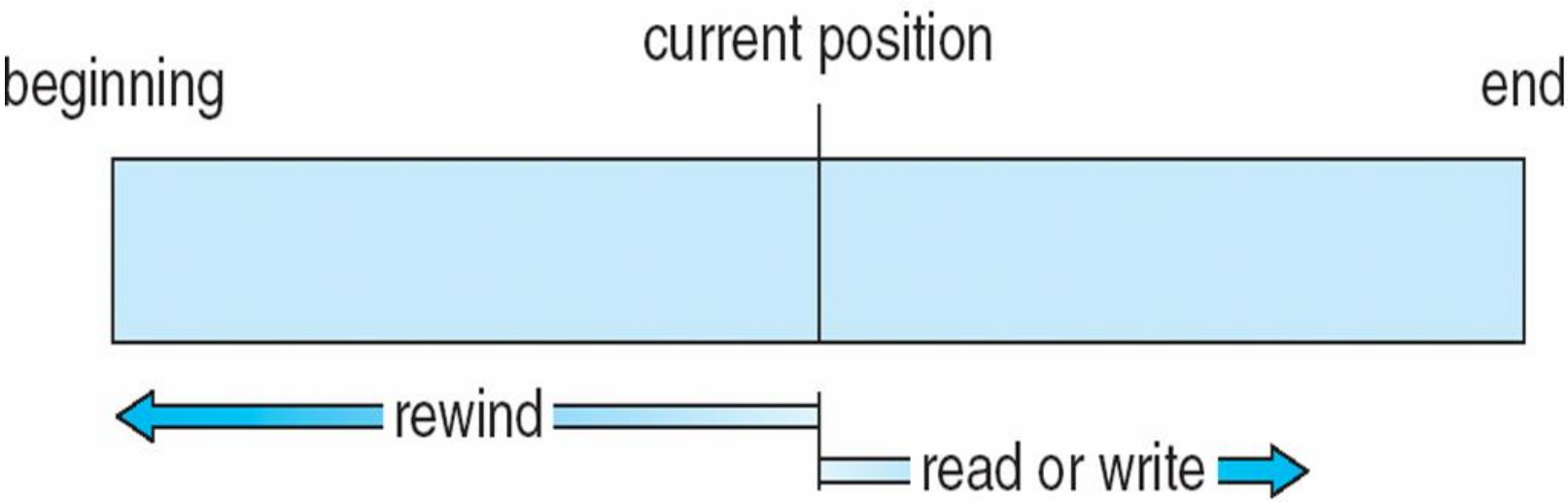
read next
write next
reset

Direct Access

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

n = relative block number

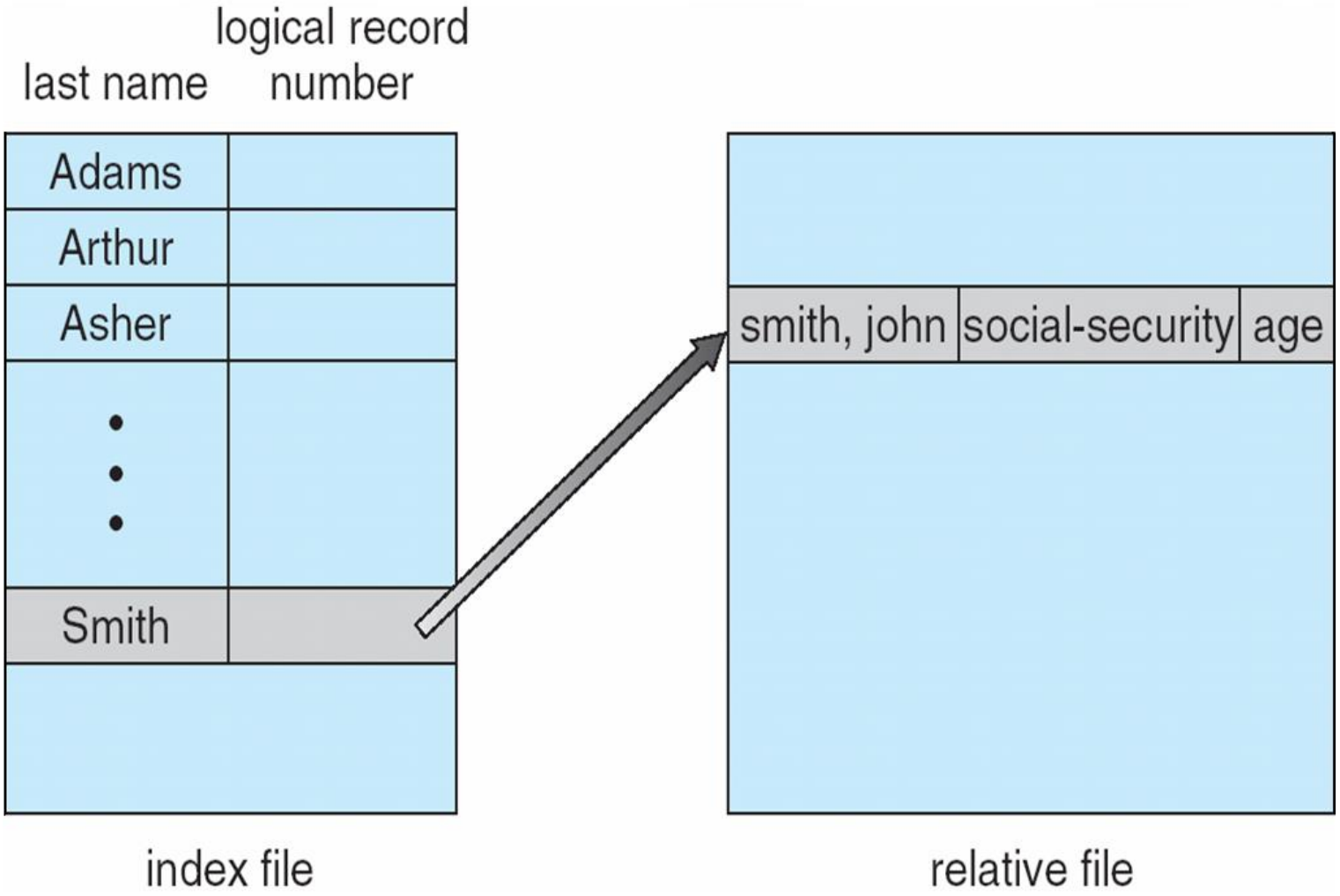
Sequential-access File



Simulation of Sequential Access on Direct-access File

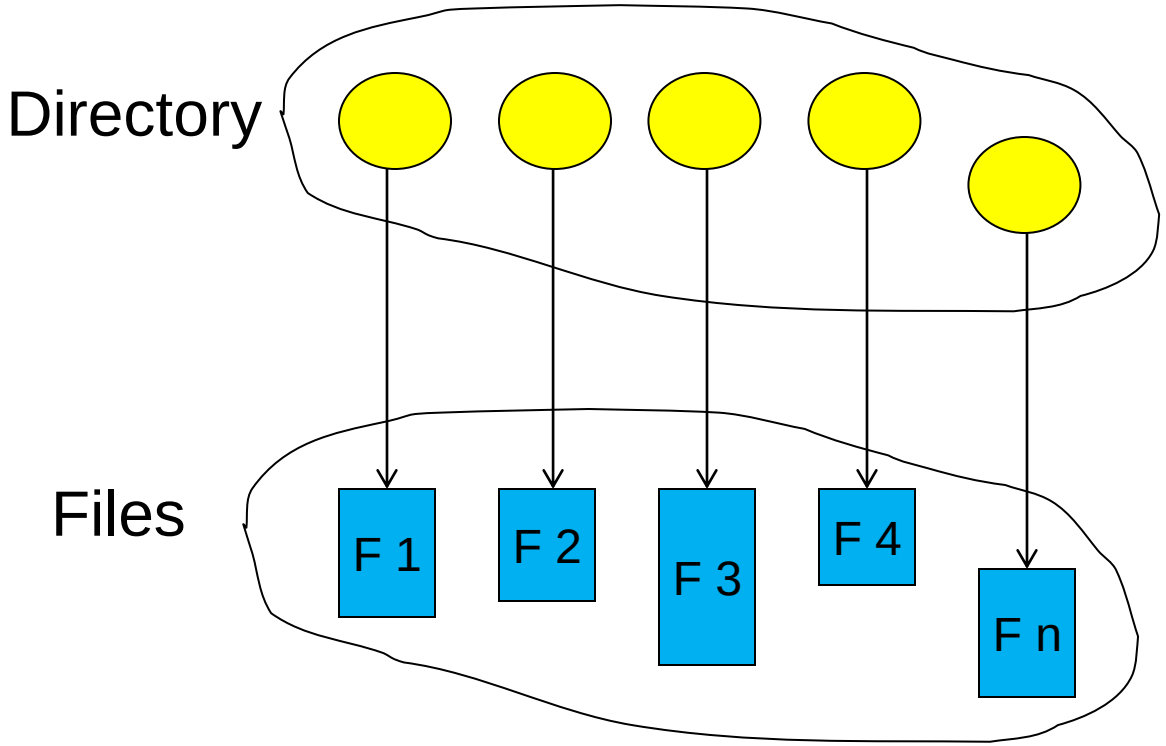
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

Example of Index and Relative Files



Directory Structure

A collection of nodes containing information about all files



Both the directory structure and the files reside on disk
Backups of these two structures are kept on tapes

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

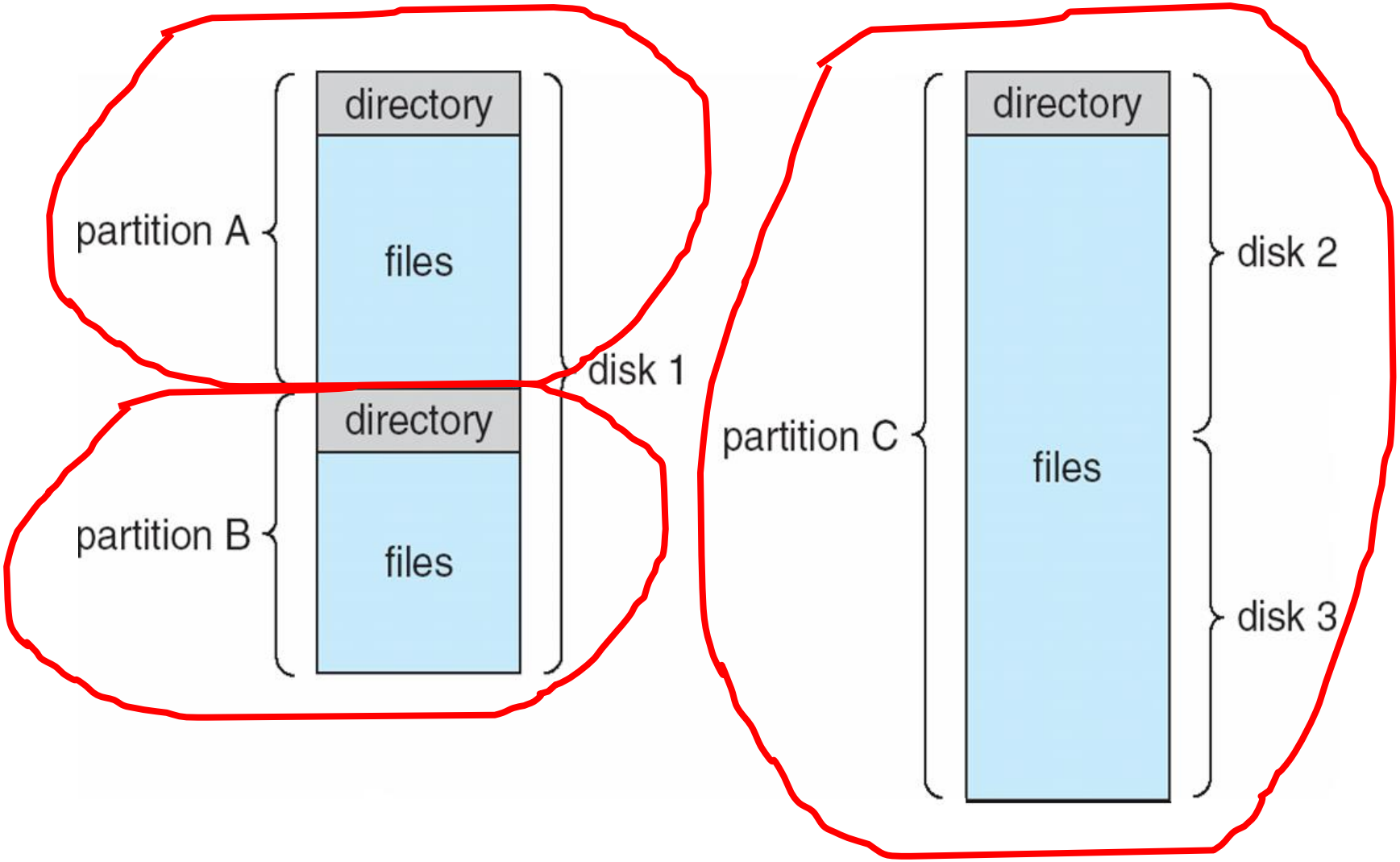
Partitions also known as minidisks, slices

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 (Solaris)

A Typical File-system Organization



Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Organize the Directory (Logically) to Obtain

Efficiency – locating a file quickly

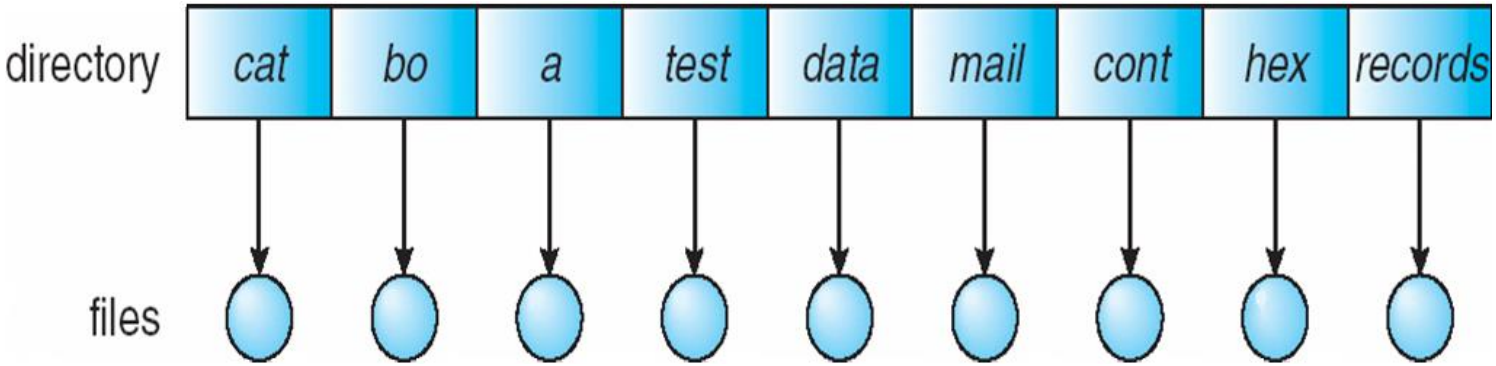
Naming – convenient to users

- 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

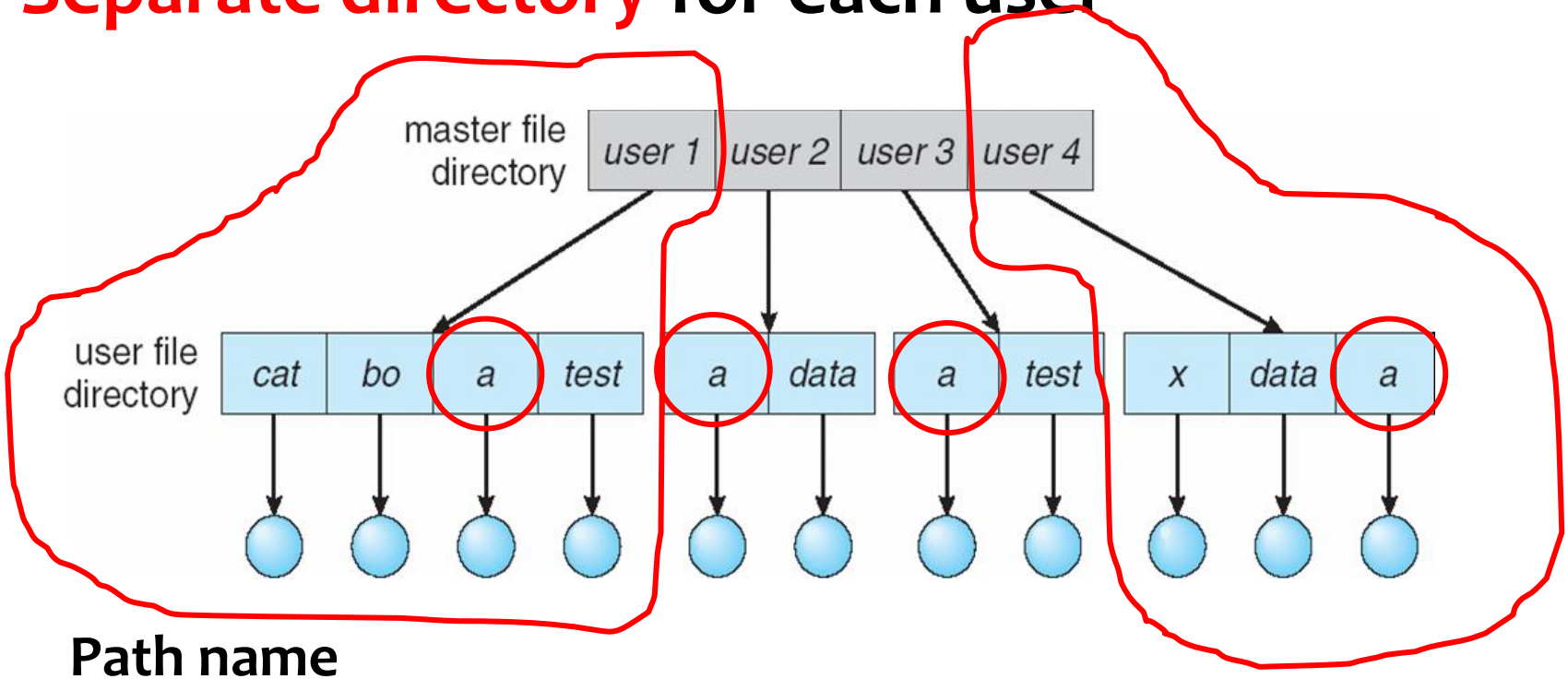
A single directory for all users



- Naming problem
- Grouping problem

Two-Level Directory

Separate directory for each user



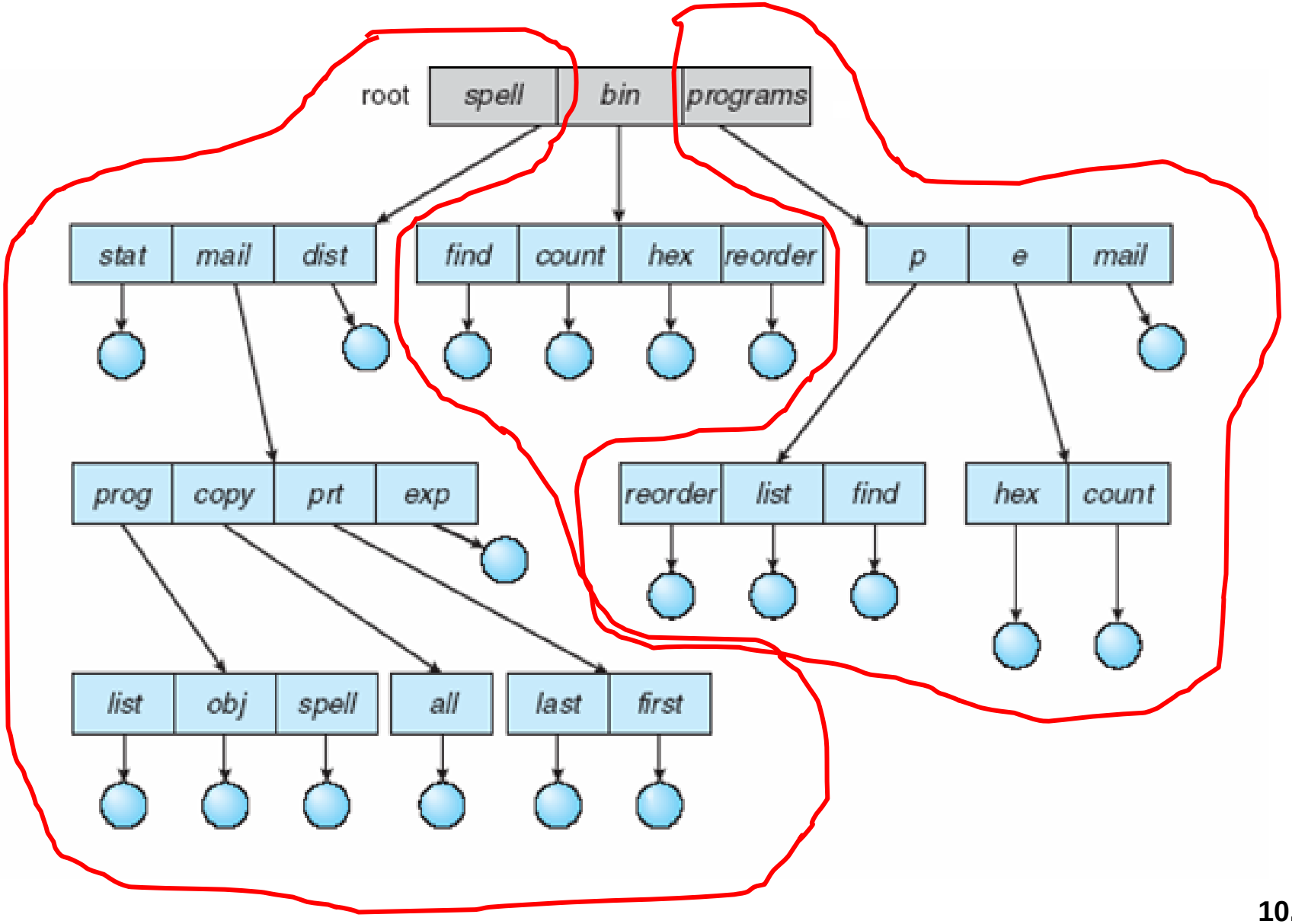
Path name

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

Current directory (working directory)

cd /spell/mail/prog

type list

Tree-Structured Directories (Cont)

Absolute or relative path name

Creating a new file is done in current directory

Delete a file

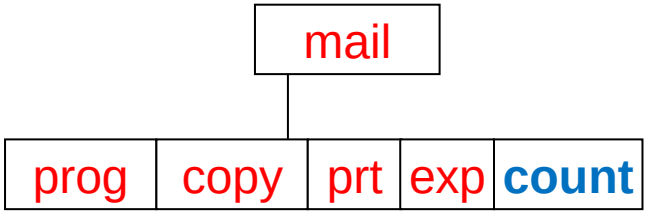
rm <file-name>

Creating a new subdirectory is done in current directory

mkdir <dir-name>

Example: if in current directory /mail

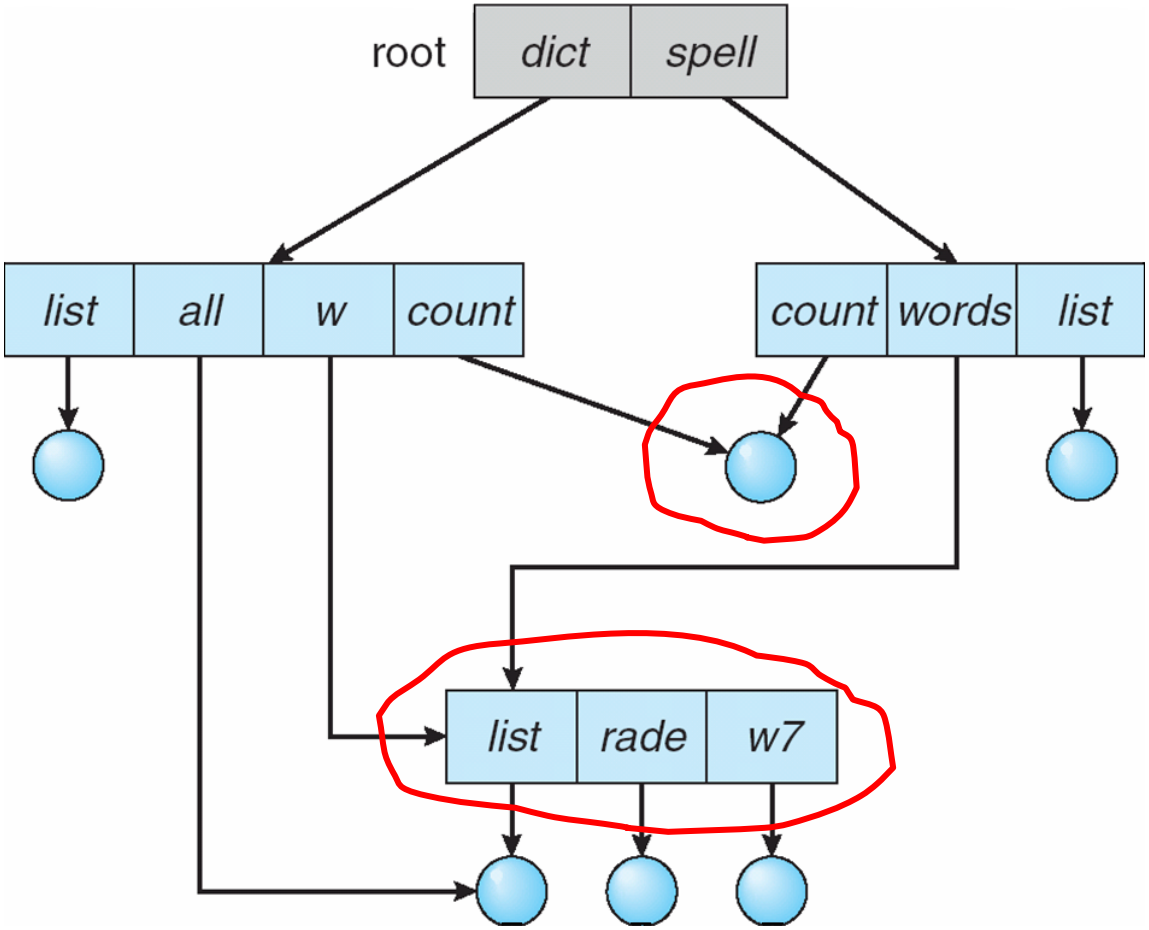
mkdir count



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

Acyclic-Graph Directories

Have shared subdirectories and files, for joined project, for example



Acyclic-Graph Directories (Cont.)

Allows directories to share subdirectories and files. The same file or subdirectory may be in two different directories

Shared files and subdirectories can be implemented in several ways.

Create a new directory entry - Link

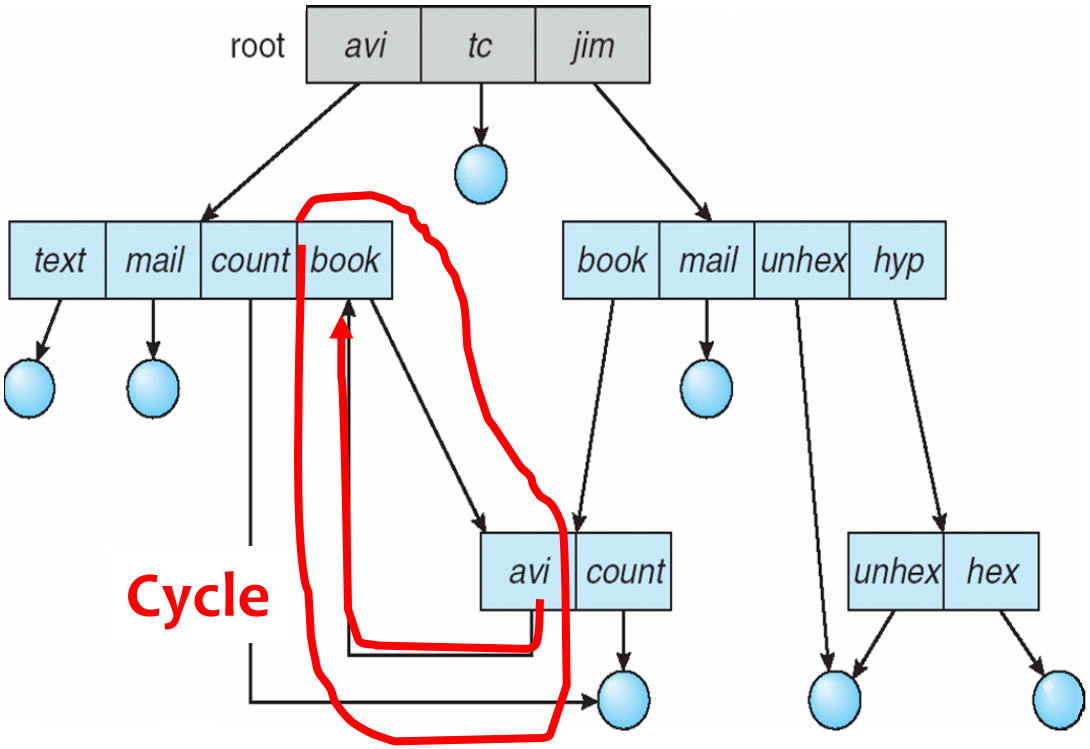
Link – a pointer to another file or subdirectory. A link may be implemented as an absolute or a relative path name.

Resolve the link – using that path name to locate the real file. Links are easily identified by their format in the directory entry and are effectively indirect pointers.

General Graph Directory

A serious problem with using acyclic-graph structure is ensuring that there is no cycles.

However, when we add links, the tree structure is destroyed, resulting in a simple graph structure.



General Graph Directory (Cont.)

If cycles are allowed to exist in the directory

An infinite loop continually searching through the cycle

When a file can be deleted ?

- A Garbage collection scheme is used to determine when the last reference has been deleted and the disk space can be reallocated.

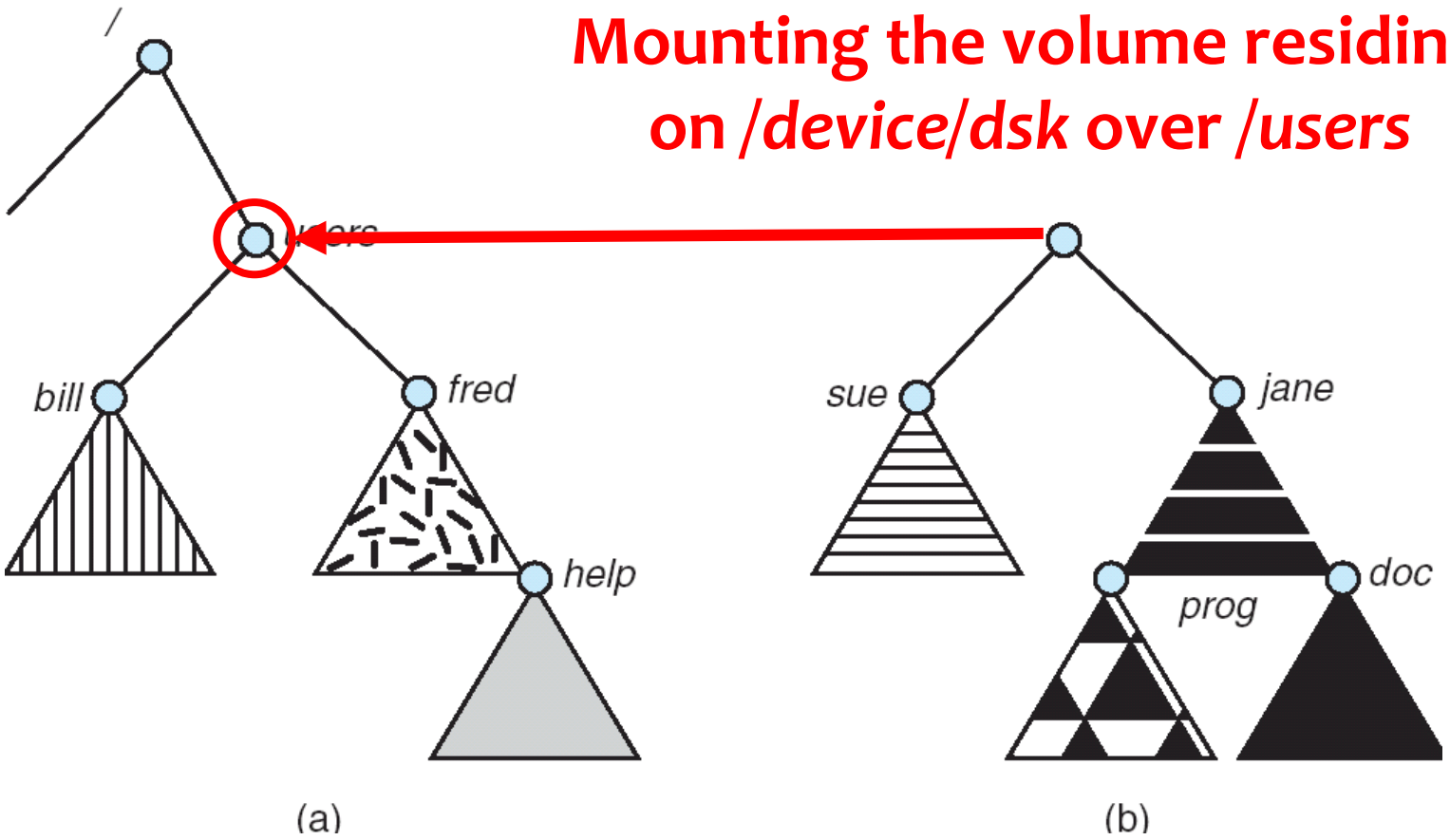
Every time a new link is added use a cycle detection algorithm to determine whether it is OK

File System Mounting

A file system must be mounted before it can be accessed

A unmounted file system is mounted at a mount point

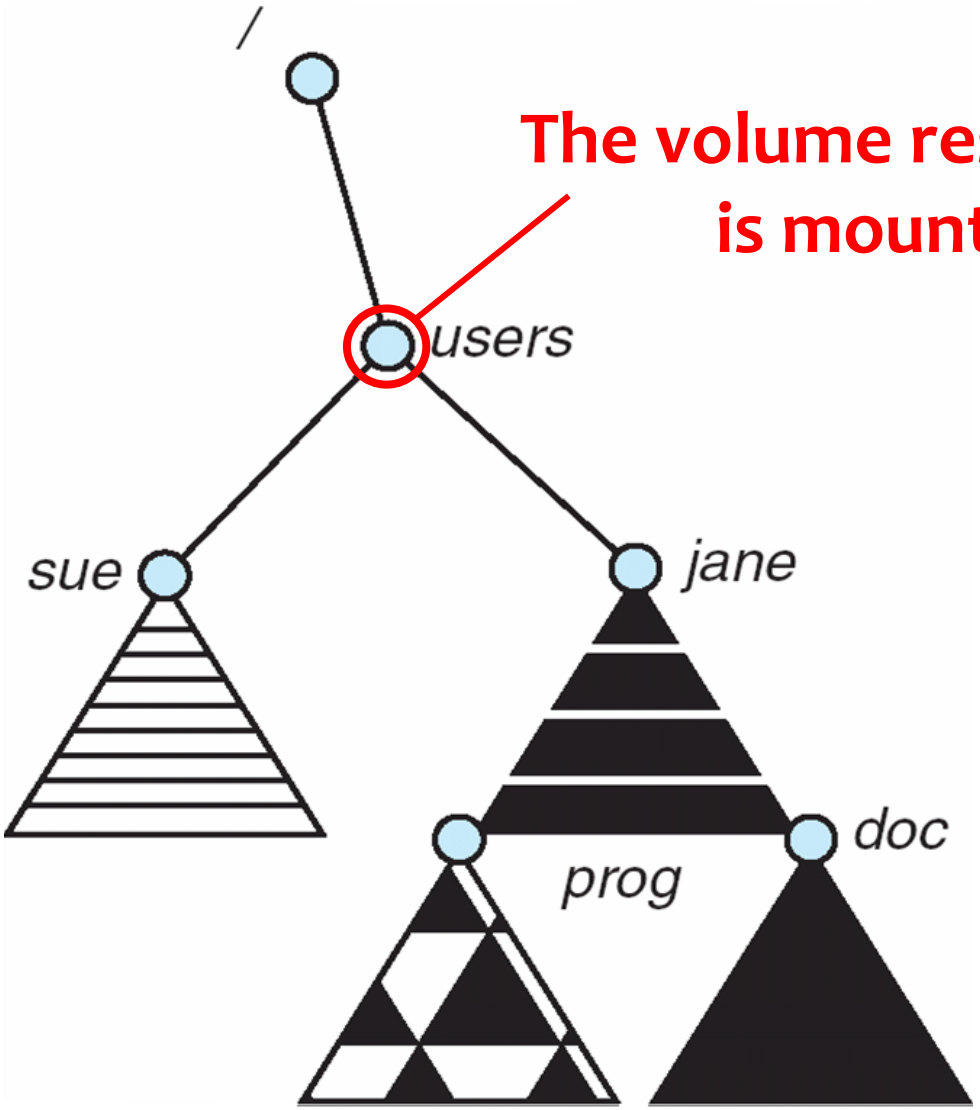
File System Mounting



(a) Existing.

(b) Unmounted Partition

Mount Point



The volume residing on */device/dsk* is mounted over */users*

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

Uses networking to allow file system access between systems

- **Manually** via programs like FTP
- **Automatically,** seamlessly using distributed file systems
- **Semi automatically** via the WWW

File Sharing – Remote File Systems

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 (Common Internet File System) is standard Windows protocol
- Standard OS file calls are translated into remote calls

Distributed Information Systems (distributed naming services) such as LDAP (lightweight directory access protocol), DNS, NIS, Active Directory (Windows XP and Windows 2000) 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 6 process synchronization algorithms

- Tend to be less complex due to disk I/O and network latency (for remote file systems)

Andrew File System (AFS, Chapter 17) implemented complex remote file sharing semantics

Unix file system (UFS, Chapter 17) 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

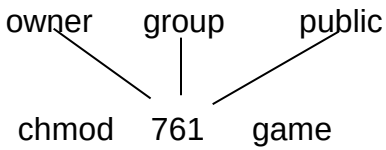
Mode of access: read, write, execute

Three classes of users

			RWX
a) owner access	7	⇒	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.

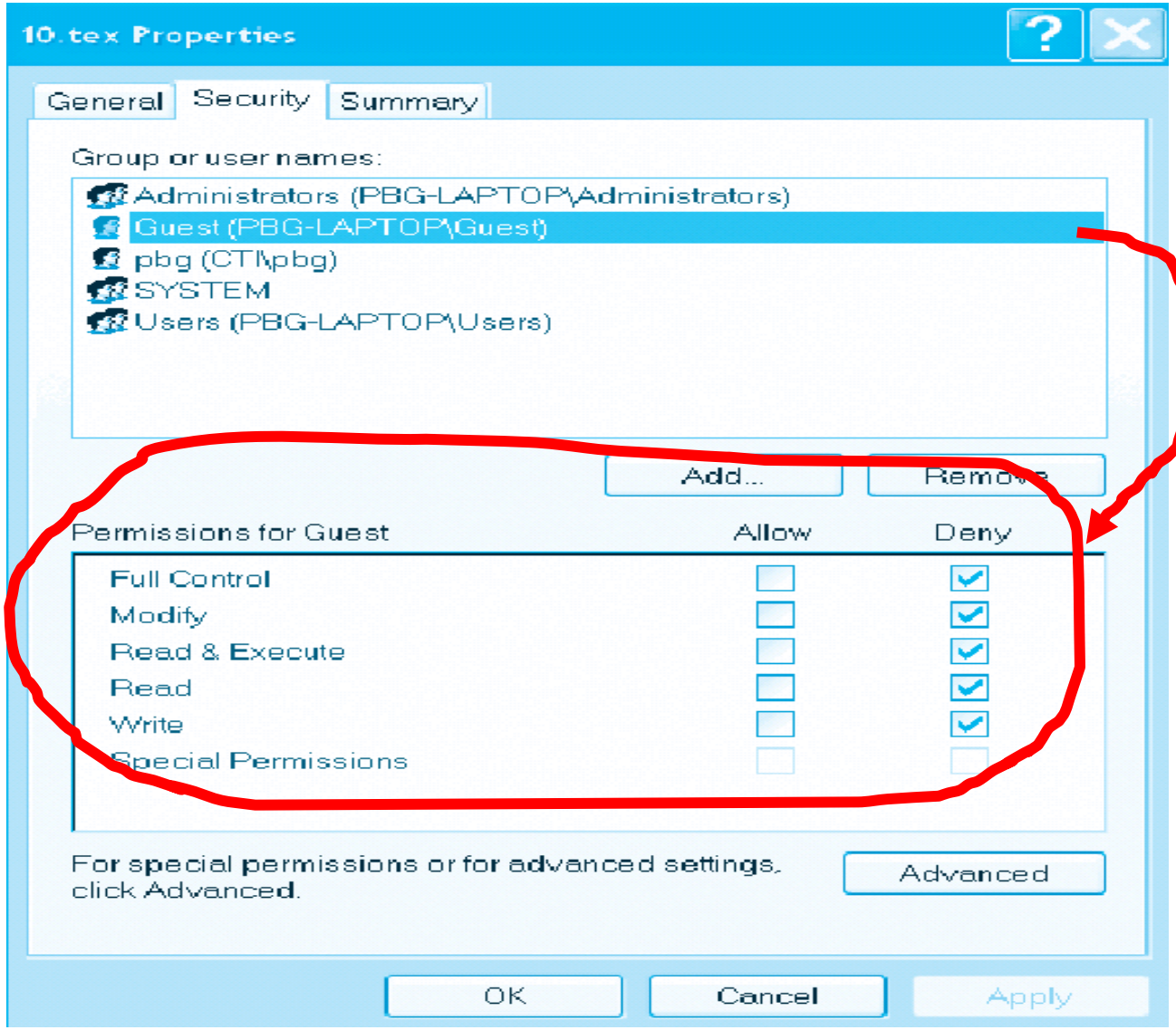
For a particular file (say *game*) or subdirectory, define an appropriate access.



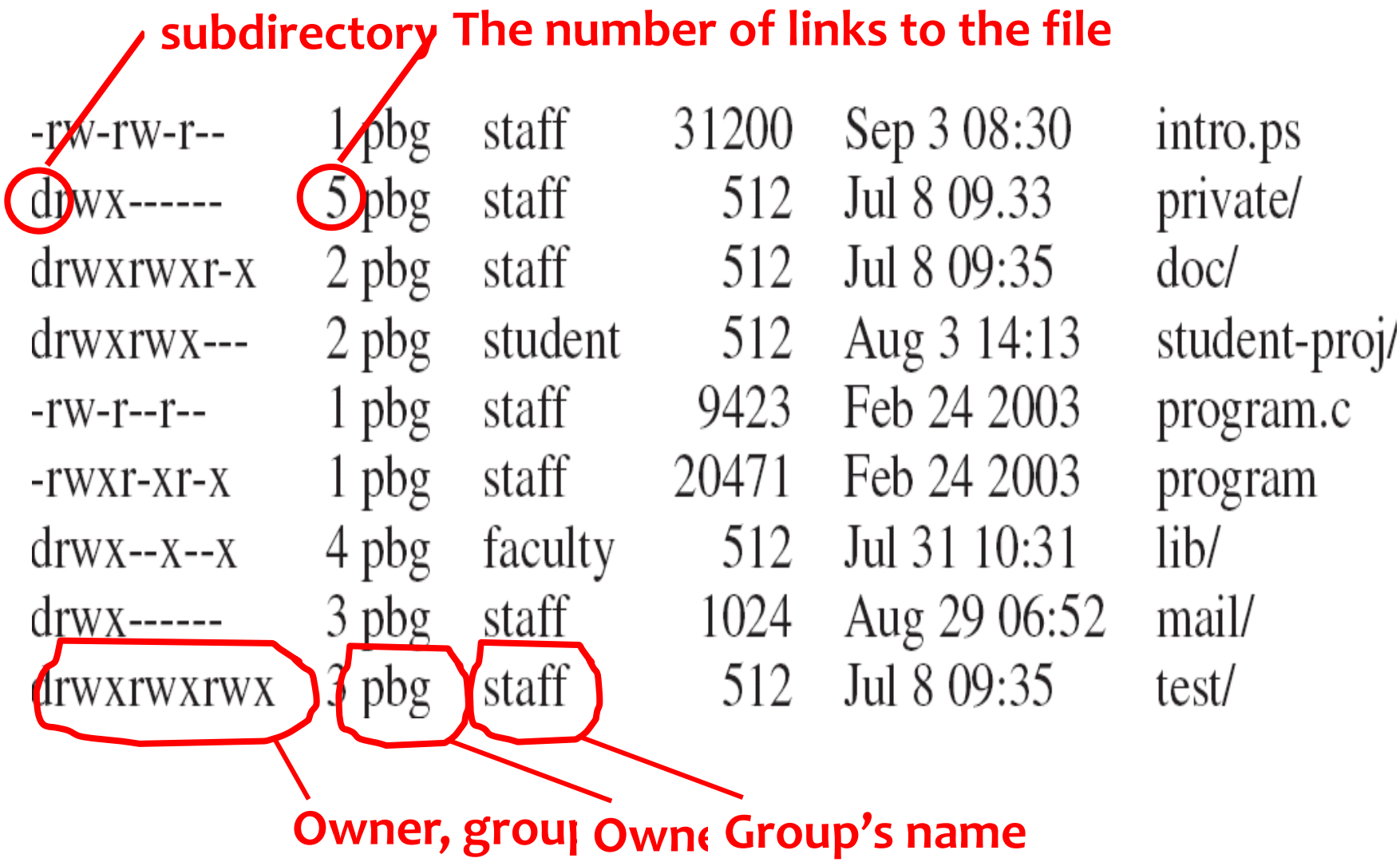
Attach a group to a file

chgrp G game

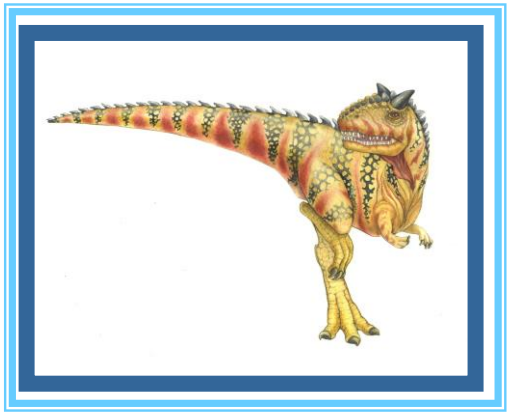
Windows XP Access-control List Management



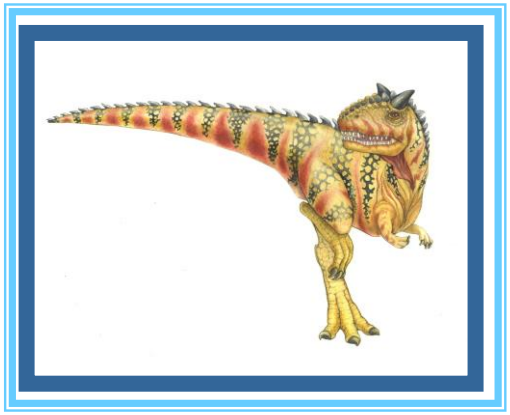
A Sample UNIX Directory Listing



End of Chapter 10



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

Log-Structured File Systems

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

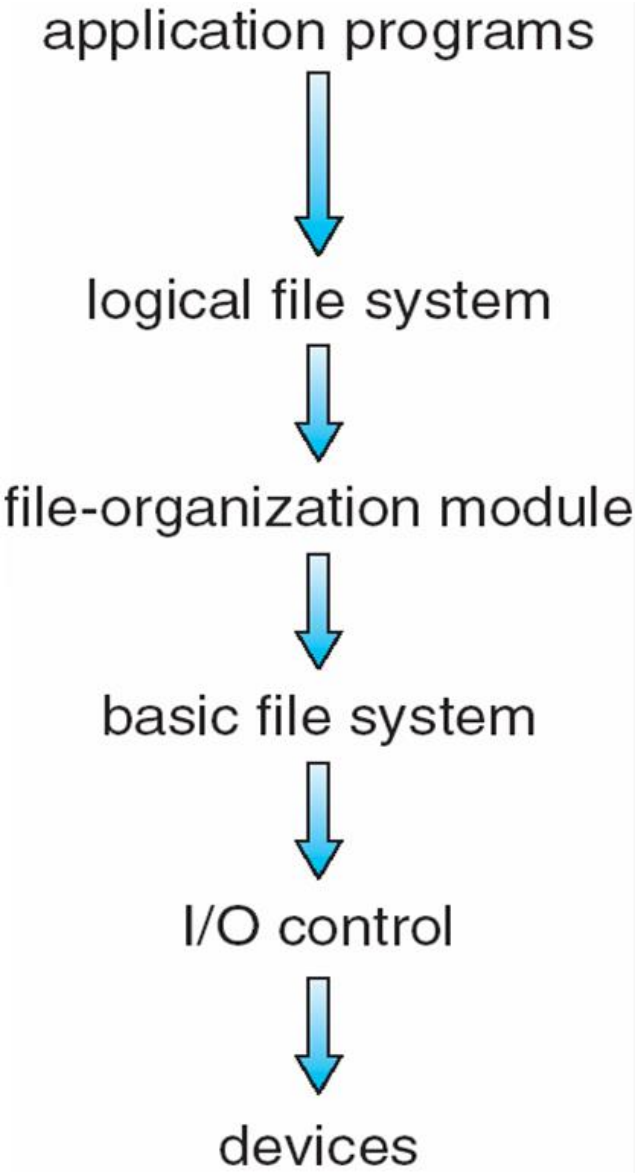
- Logical storage unit
- Collection of related information

File system resides on secondary storage (disks)

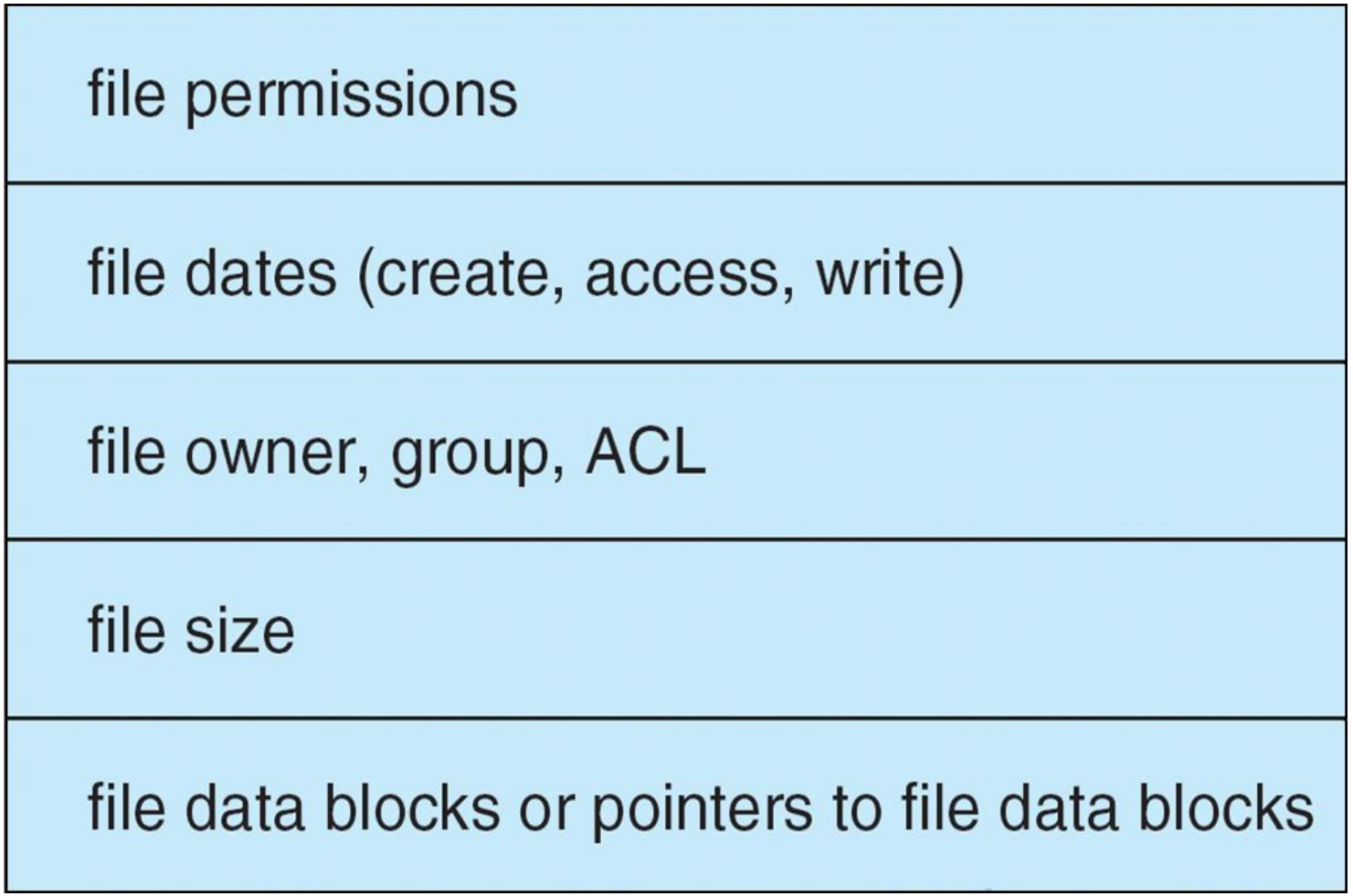
File system organized into layers

File control block – storage structure consisting of information about a file

Layered File System

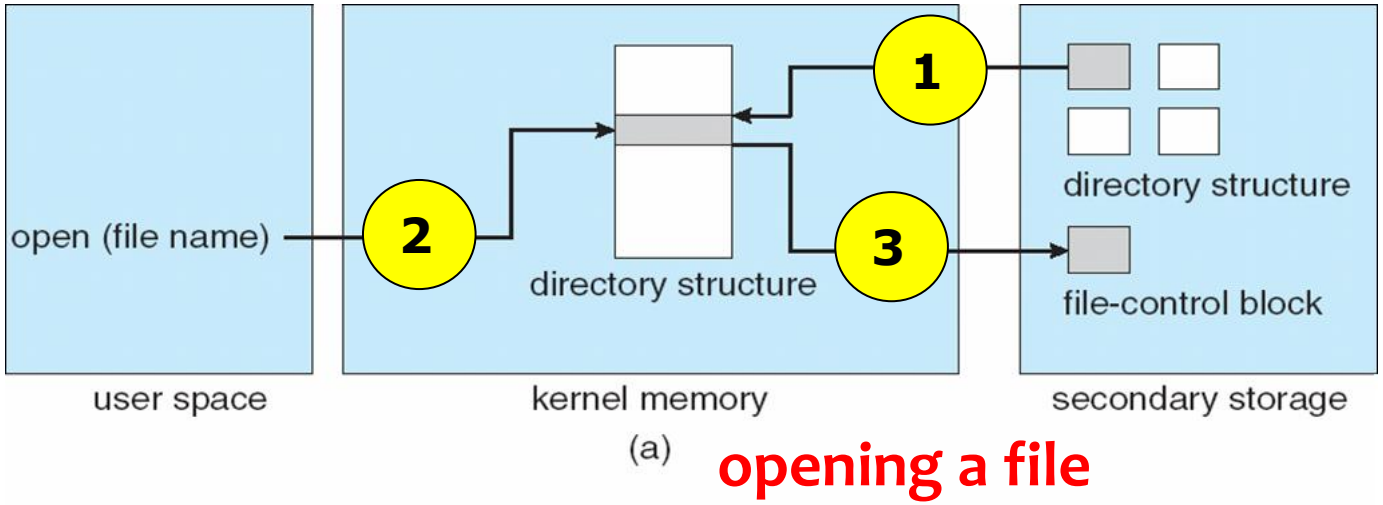


A Typical File Control Block

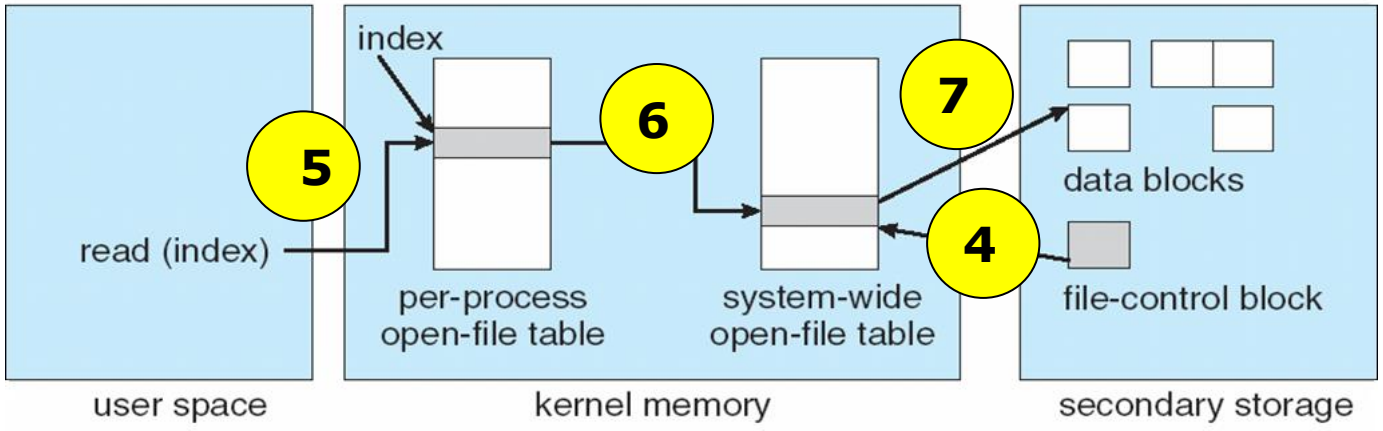


In-Memory File System Structures

The necessary file system structures provided by the OS.



(a) opening a file



(b) reading a file

Virtual File Systems

Modern OS must concurrently support multiple types of file systems.

How does an OS allow multiple types of file systems to be integrated into a directory structure ?

To write directory and file routines for each type.

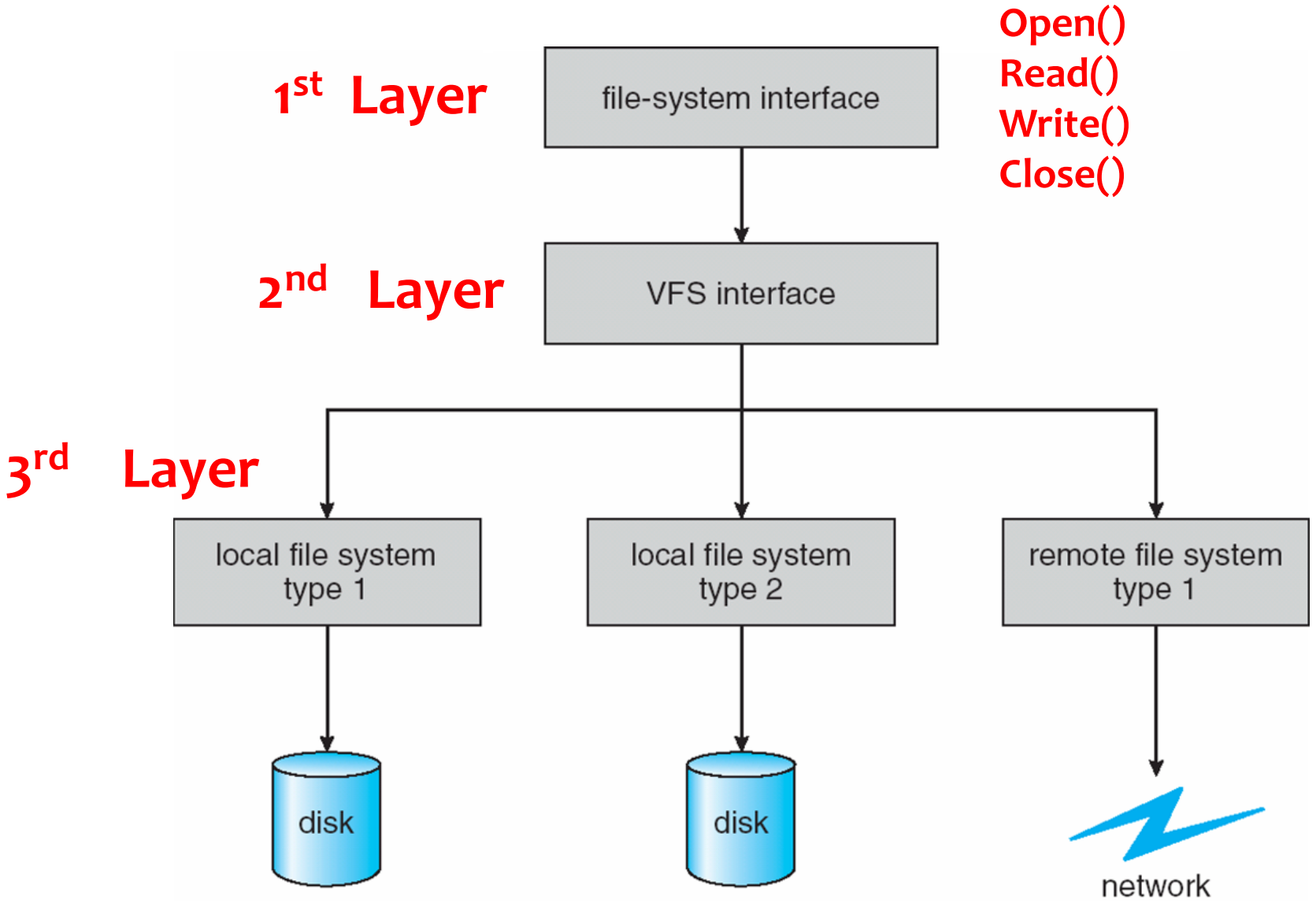
Most Operating systems provide an object-oriented way of implementing file systems, including UNIX.

The file-system implementation consists of three major layers (Figure 11.4)

The first layer is the file-system interface, based on open(), read(), write(), and close() calls

The 2nd layer is called the virtual file system (VFS) layer

Schematic View of a Virtual File System



Virtual File Systems

The VFS serves two important functions

It separates file-system-generic operations from their implementation by defining a clean VFS interface.

It provides a mechanism for uniquely representing a file throughout a network.

- The VFS is based on a file-representation structure, called a vnode, that contains numerical designator for a network-wild unique file.

Virtual File Systems

- Unix inodes are unique only within a single file system
- The kernel maintains one vnode structure for each active node (file or directory)

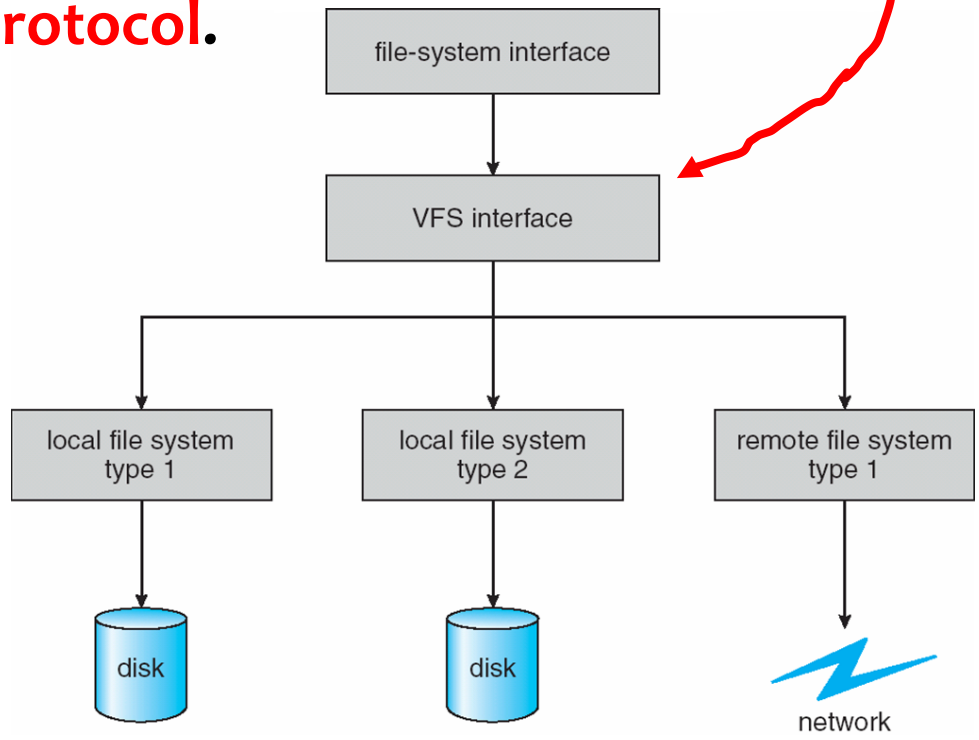
The VFS distinguishes local files from remote files, and local files are further distinguished according to their file-system types.

Virtual 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.

The 3rd layer implements the file-system type or the remote-file-system protocol.



Virtual File Systems

The VFS architecture in Linux defines four major object types

- The inode object, represents an individual file
- The file object, represents an open file
- The Superblock object, represents an entire file system
- The dentry object, represents an individual directory entry.

Directory Implementation

The selection of directory-allocation and directory-management algorithms affects the efficiency, performance, and reliability of the file system

Linear list of file names with pointer to the data blocks.

- simple to program
- time-consuming to execute

Hash Table – linear list stores the directory entries, but a hash data structure is also used.

- decreases directory search time
- collisions – situations where two file names hash to the same location
- fixed size

Allocation Methods

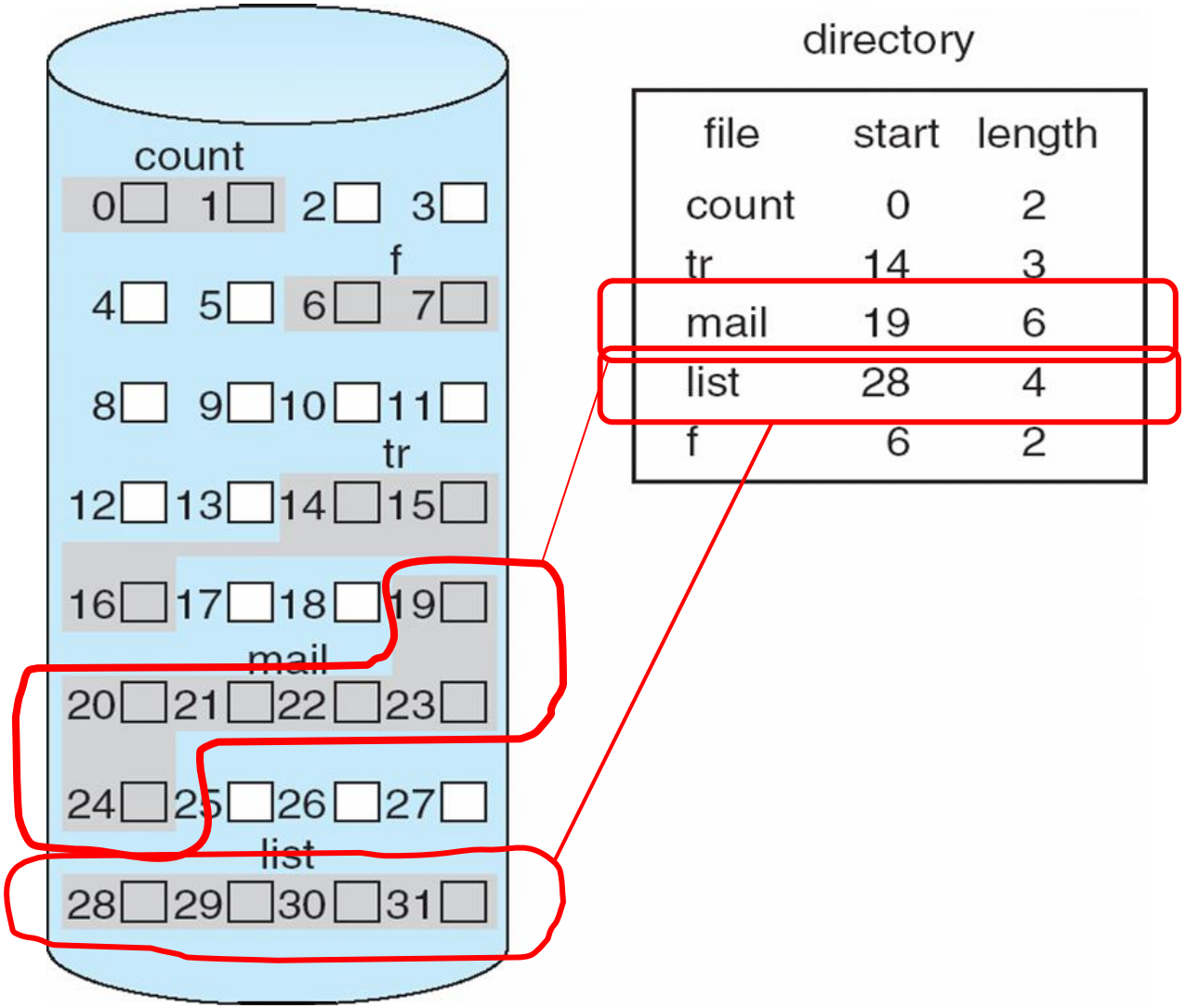
An allocation method refers to how disk blocks are allocated for files:

- 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), external fragmentation
- Files cannot grow

Contiguous Allocation of Disk Space



Extent-Based Systems

Many newer file systems use a modified contiguous allocation scheme

Extent-based file systems allocate disk blocks in extents

An extent is a contiguous block of disks

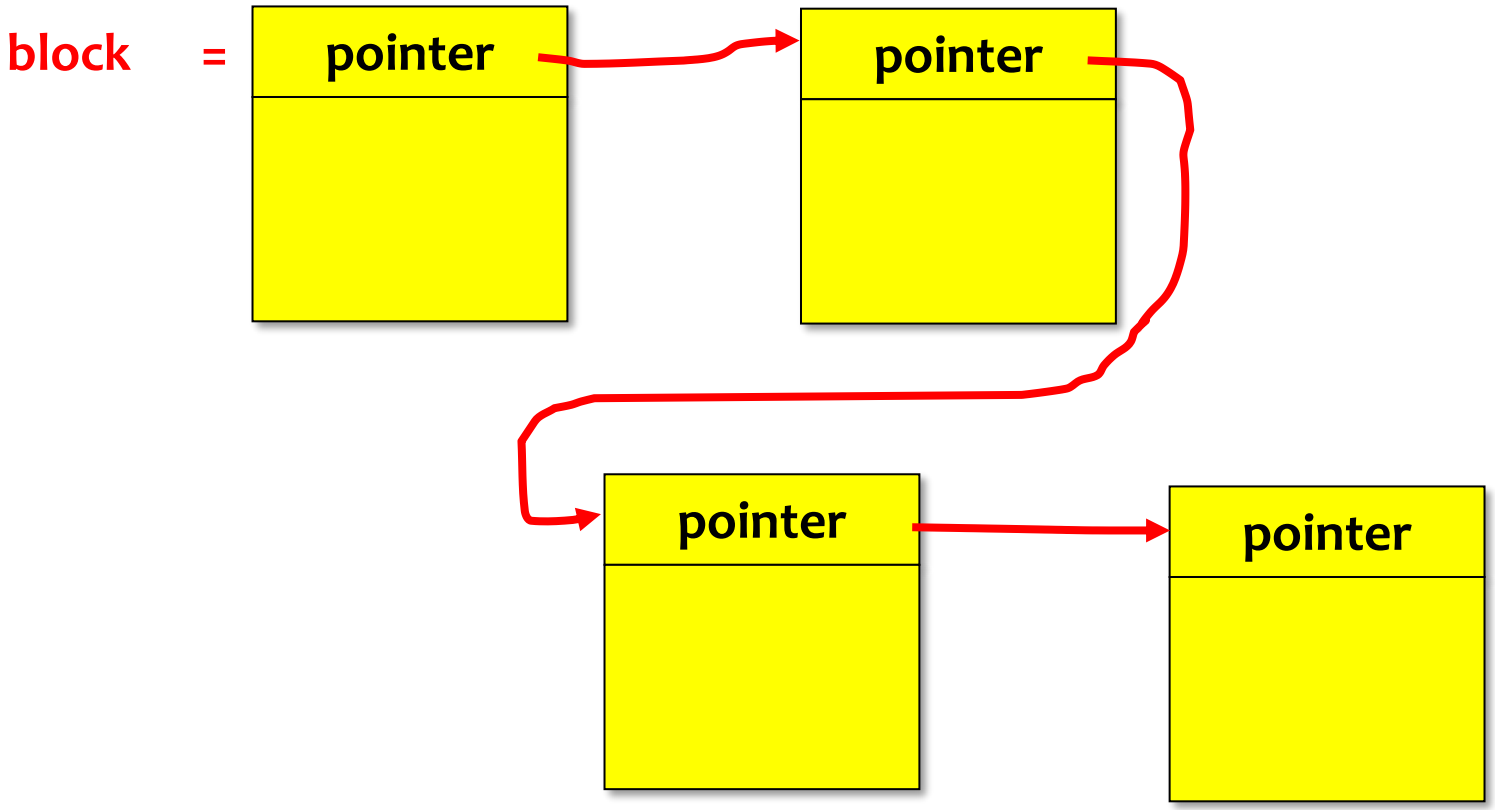
- Extents are allocated for file allocation
- A file consists of one or more extents.

The location of a file's blocks is then recorded as a location and a block count, plus a link to the first block of the next extent.

The commercial Veritas File System uses extents to optimize performance. It is a high-performance replacement for standard UNIX UFS.

Linked Allocation

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



Linked Allocation (Cont.)

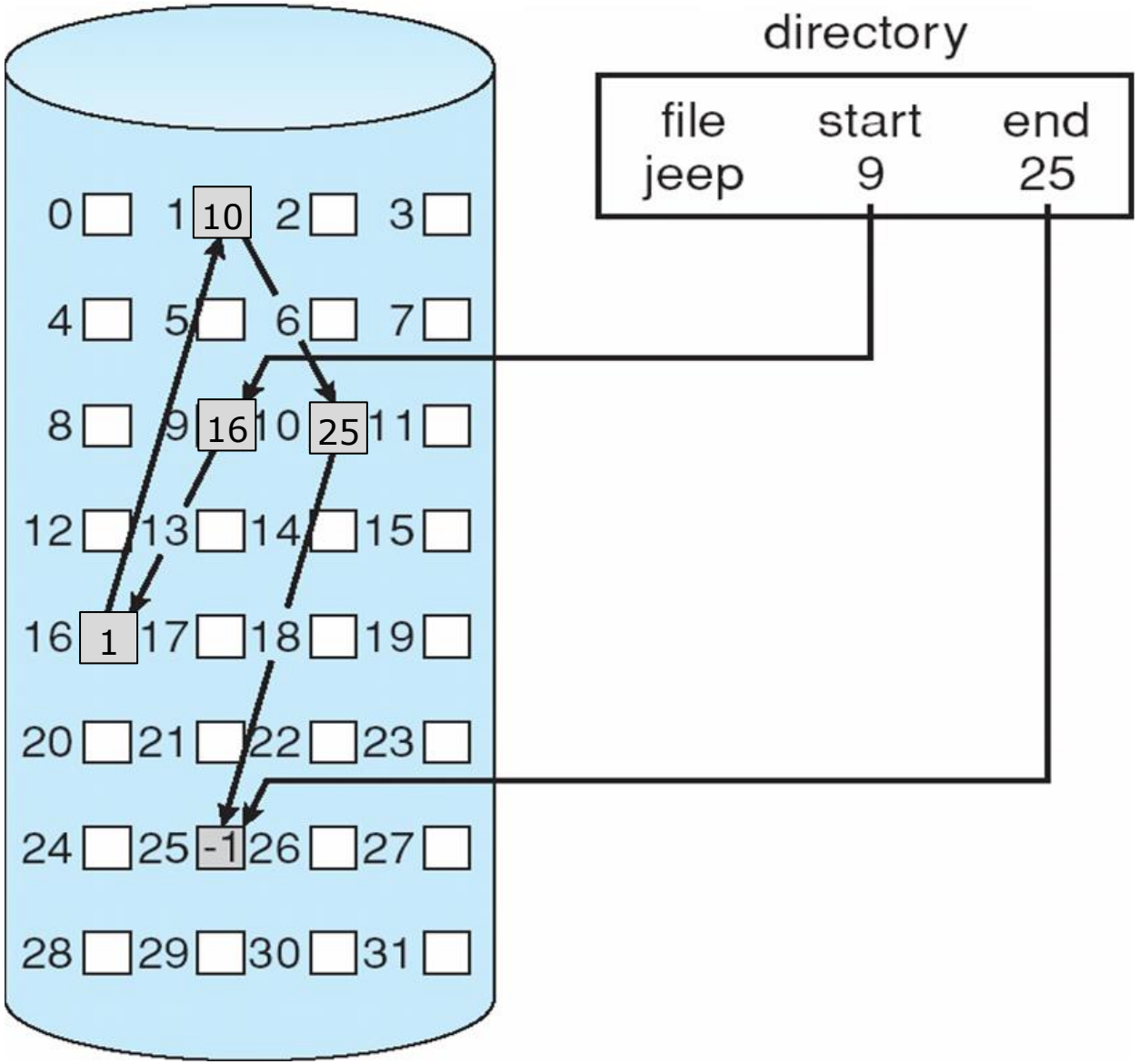
Simple – need only starting address

Free-space management system – no waste of space

Some disadvantages

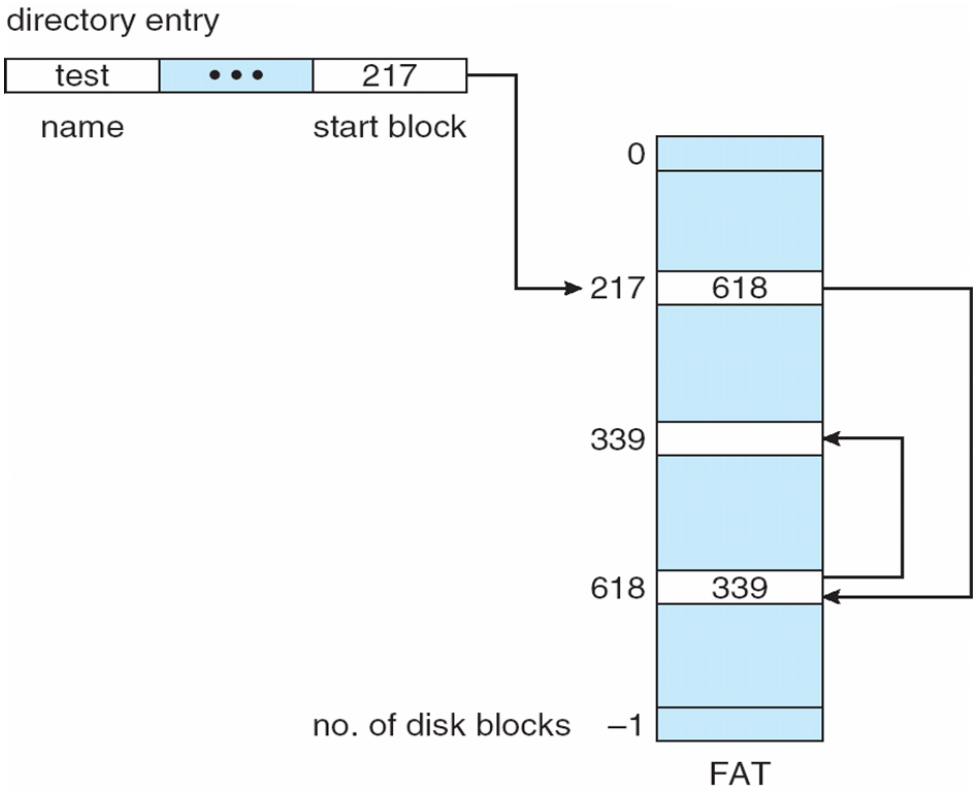
- No random access (only for sequential access files)
- Space required for pointers, If a pointer requires 4 bytes out of a 512 bytes block, 0.78 percent is used for pointers
  - Clusters (k blocks)
- Reliability Issue

Linked Allocation



File-Allocation Table

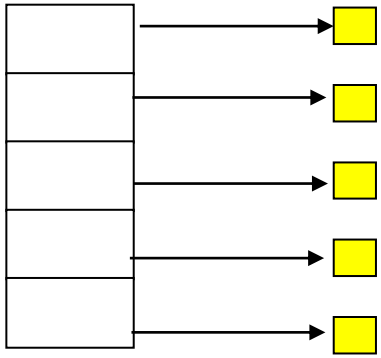
- An important variation on linked allocation is the use of a file-allocation table (FAT)
- The FAT allocation scheme can result in a significant number of disk head seeks, unless the FAT is cached.



Indexed Allocation

Brings all pointers together into the *index block.*

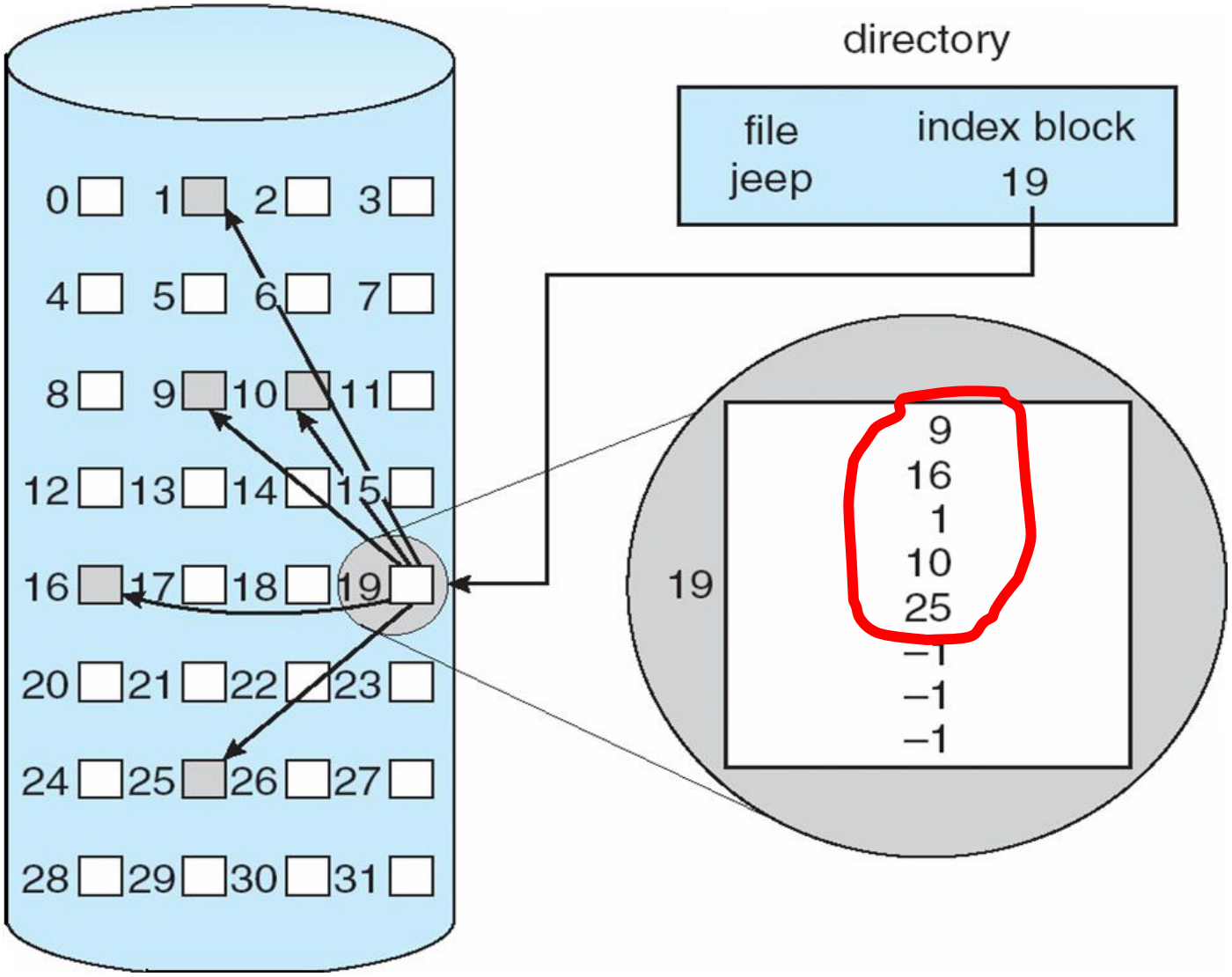
Logical view.



blocks

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.

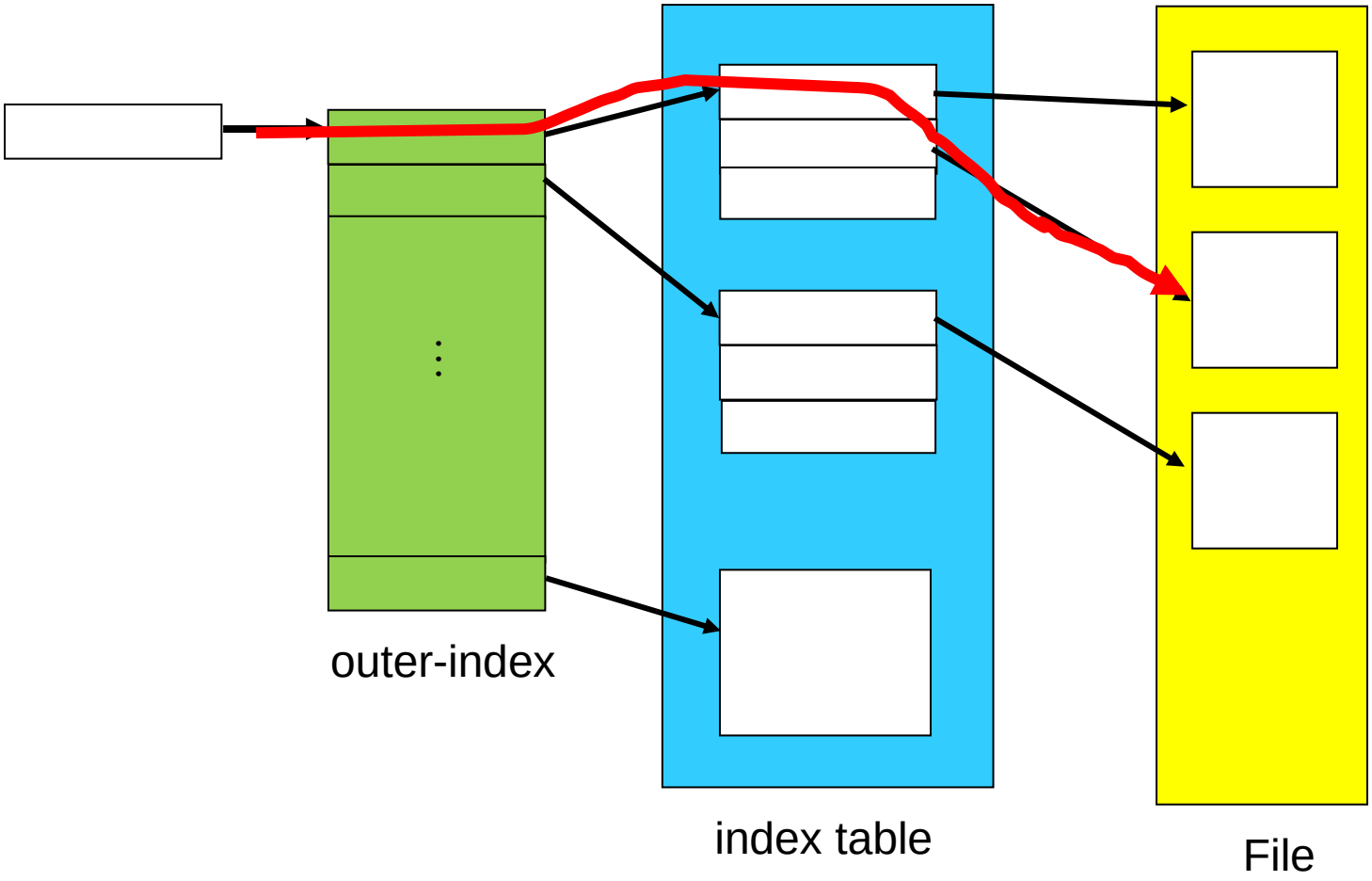
Only 1 block is needed for index table.
256K/0.5K = 512 blocks

Indexed Allocation – Mapping (Cont.)

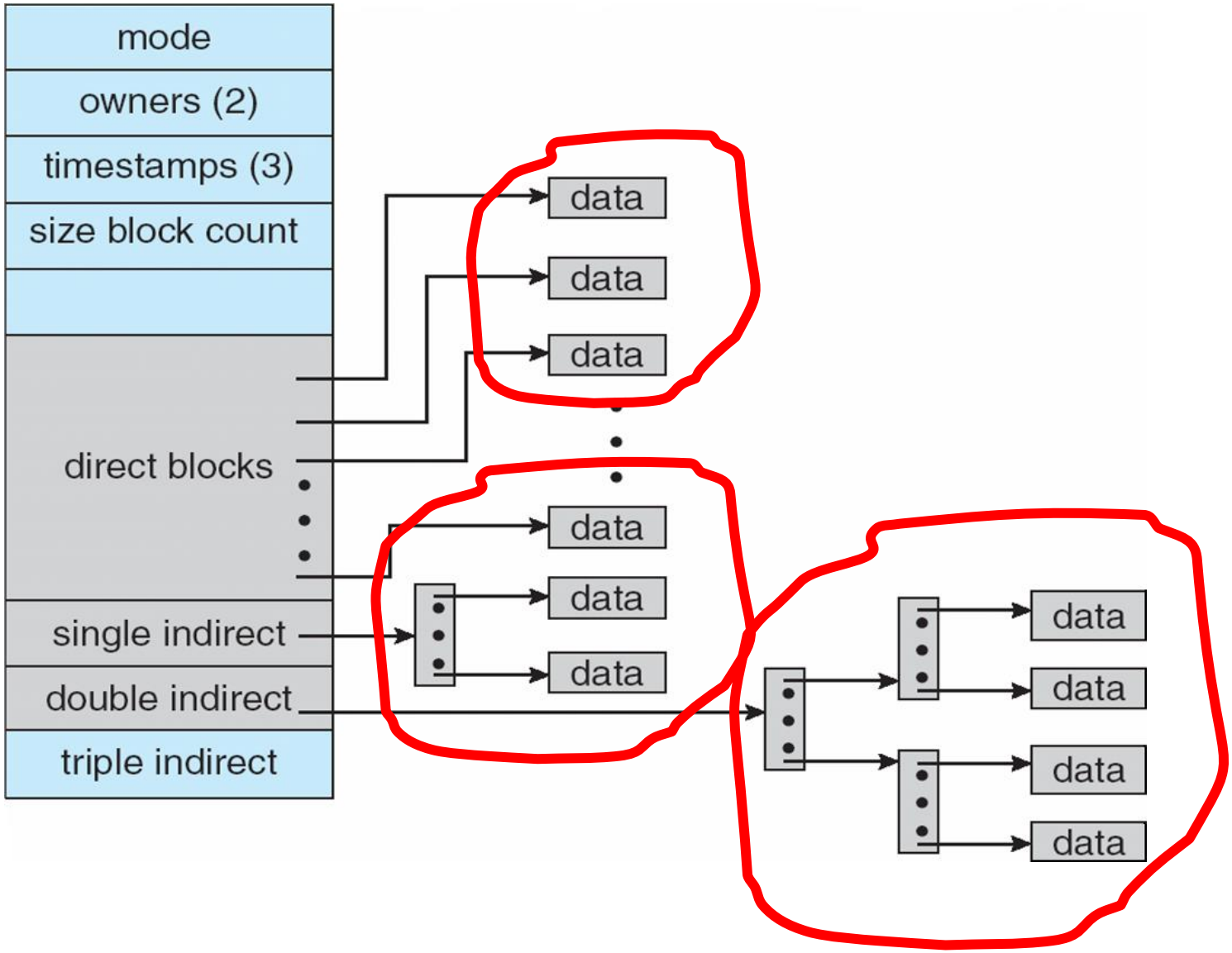
Mapping from logical to physical in a file of unbounded length (block size of 512 words).

Linked scheme – Link blocks of index table (no limit on size).

Indexed Allocation – Mapping (Cont.)

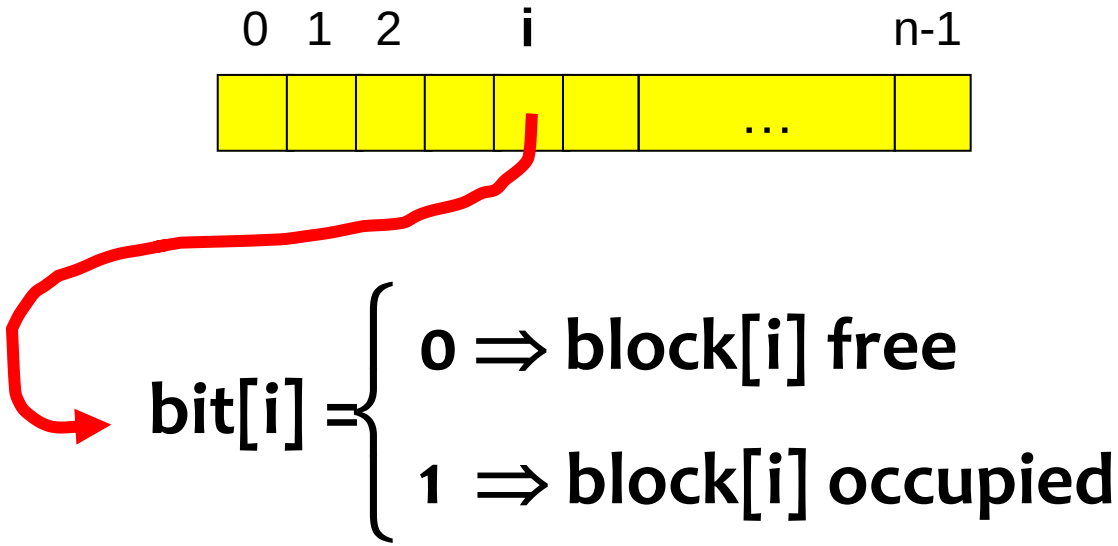


Combined Scheme: UNIX (4K bytes per block)



Free-Space Management

Bit vector (*n* blocks)



Free-Space Management (Cont.)

Bit map requires extra space

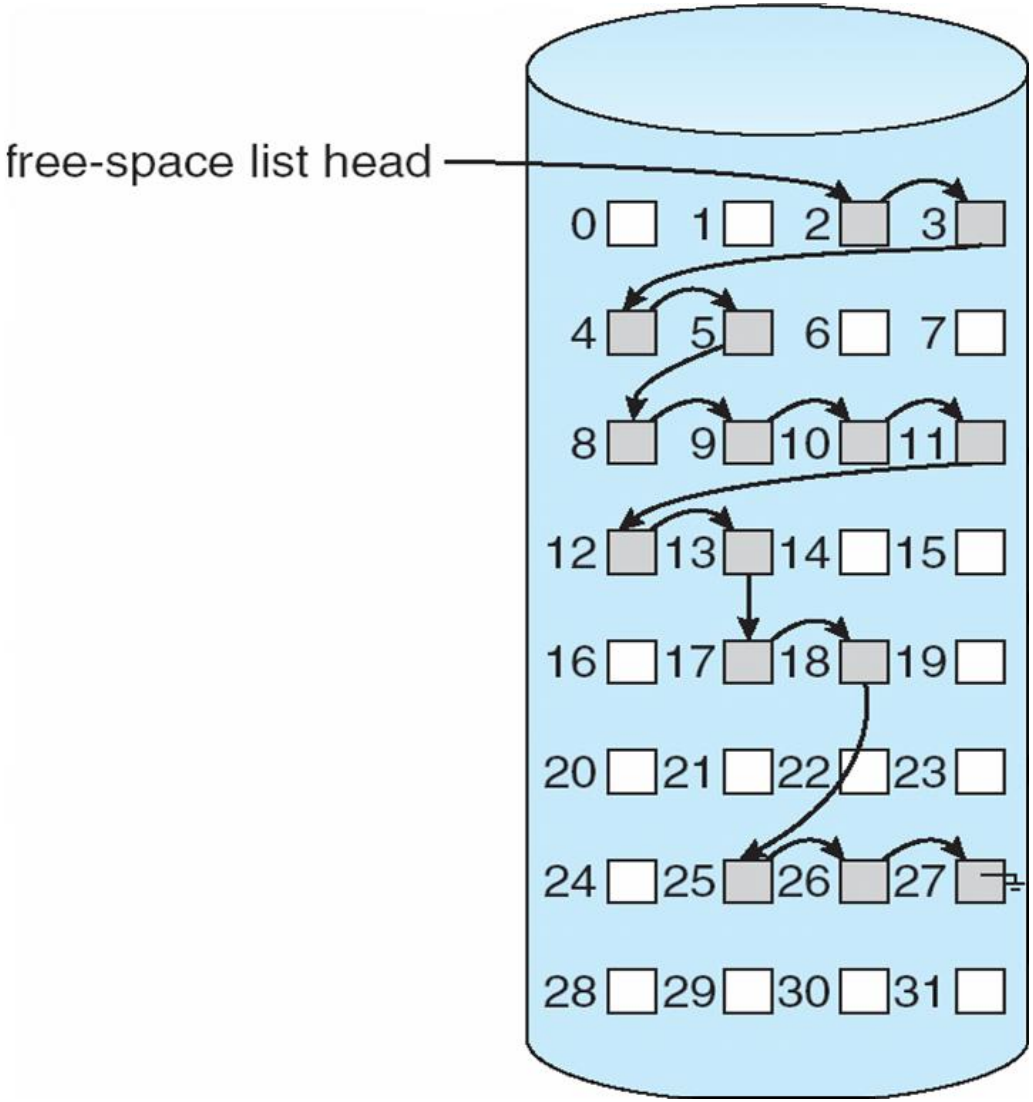
- Example:
  - block size = 2^{12} bytes
  - disk size = 2^{30} bytes (1 gigabyte)
  - $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

Easy to get contiguous files …. 000**11111111**000….

Linked list (free list)

- Cannot get contiguous space easily
- No waste of space

Linked Free Space List on Disk



Free-Space Management (Cont.)

Grouping: stores the **addresses of n free blocks** in the first free block. A block has **n** entries.

The first n-1 of these blocks are actually free.

The last block contains the addresses of another n free blocks, and so on.

Counting: keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block.

Address 1, n1 (2,3)

Address 2, n2, (8,5)

Address 3, n3, (17,2) …

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 bit[*i*] = 1 (occupied) in memory and bit[*i*] = 0 (free) on disk
- Solution:
  - Set bit[*i*] = 1 in disk
  - Allocate block[*i*]
  - Set bit[*i*] = 1 in memory

Efficiency and Performance

Efficiency

The Efficient use of disk space depends heavily on the disk allocation and directory algorithms

- UNIX inodes are preallocated on a volume. Even a "empty" disk has a percentage of it space lost to inodes.
- By preallocating, the inodes and spreading them across the volume, we improve the file system's performance. --- Try to keep a file's data block near that file's inode block to reduce the seek time.

The type of data kept in file's directory (or inode) entry also require consideration

- "Last write date"
- "Last access date"

Efficiency and Performance

Performance

disk cache – separate section of main memory for frequently used blocks

free-behind and read-ahead – techniques to optimize sequential access

- Free-behind removes a page from the buffer as soon as the next page is requested.
- With read-ahead, a requested page and several subsequent pages are read and cached.

improve PC performance by dedicating section of memory as virtual disk, or RAM disk

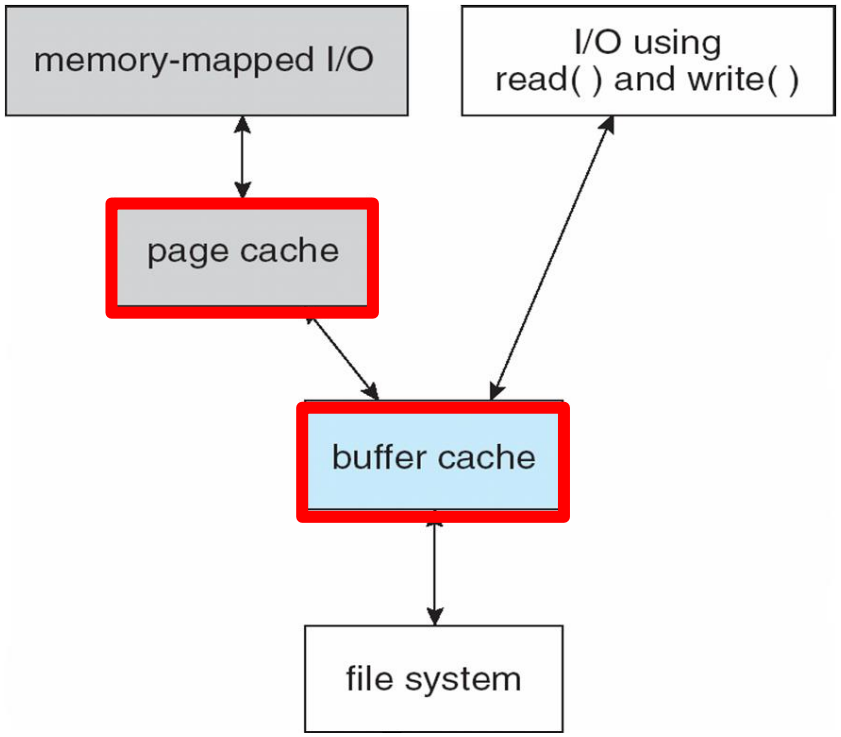
Page Cache

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

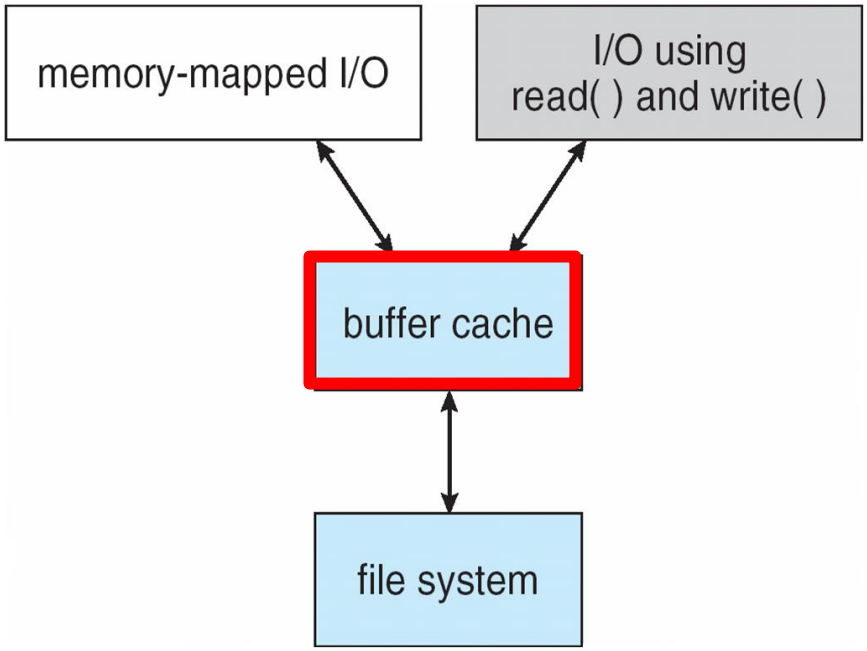
Double caching



I/O Without a Unified Buffer Cache

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 Using 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 (floppy disk, magnetic tape, other magnetic disk, optical)

Recover lost file or disk by restoring data from backup

Log Structured File Systems

Log structured 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)

Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner

NFS (Cont.)

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

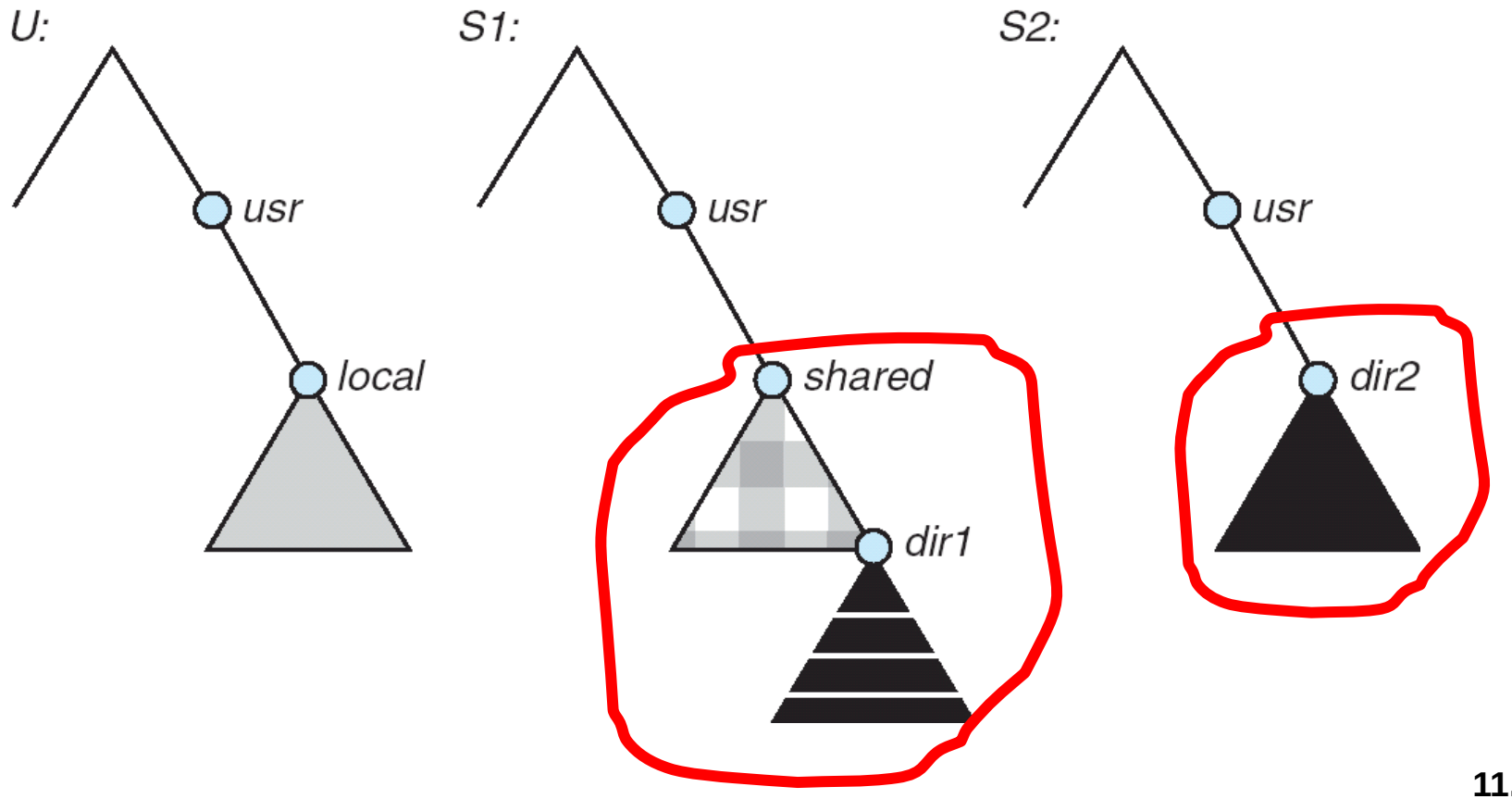
- Files in the remote directory can then be accessed in a transparent manner

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.)

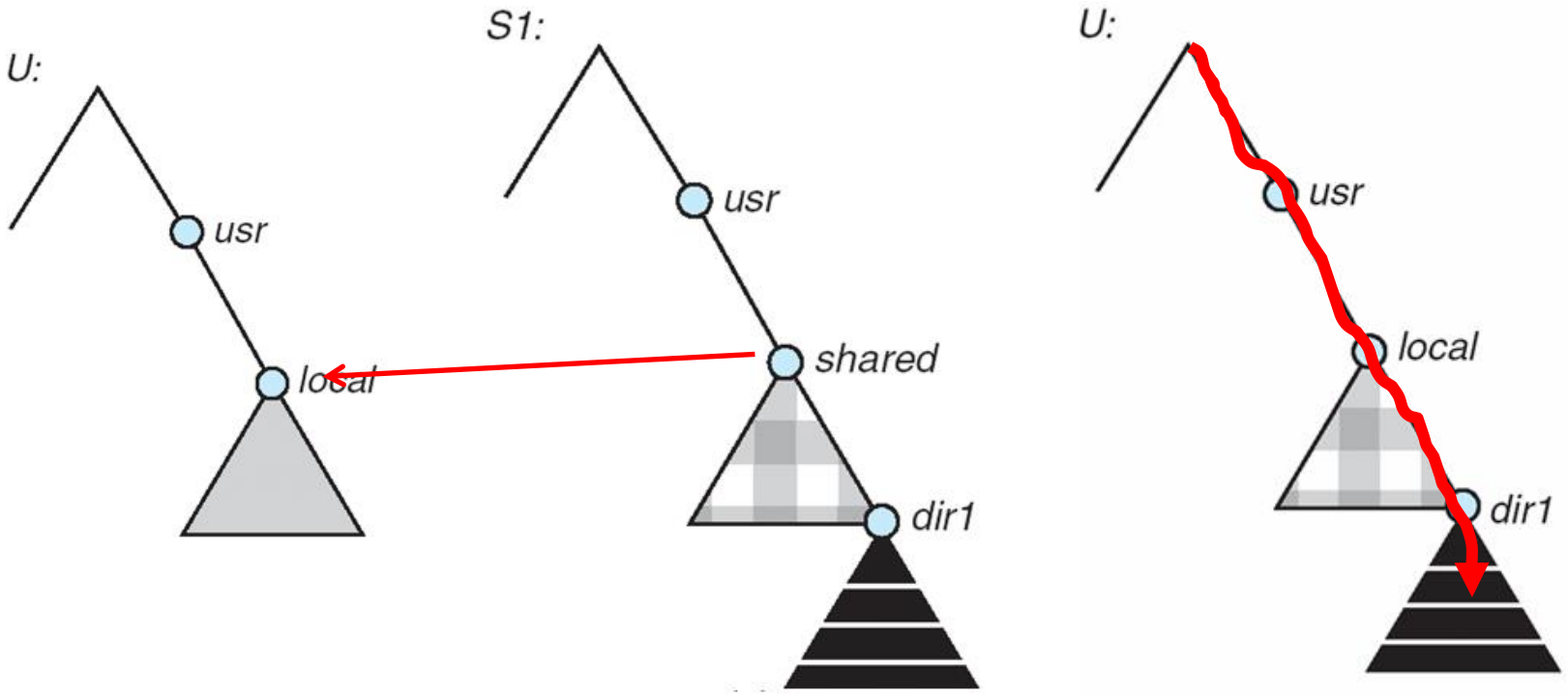
Consider the following file system where the triangles represent subtrees of directories that are of interest.

Three independent file systems of machines: U, S1, and S2



NFS (Cont.)

The effect of mounting S1:/user/shared over U:/user/local



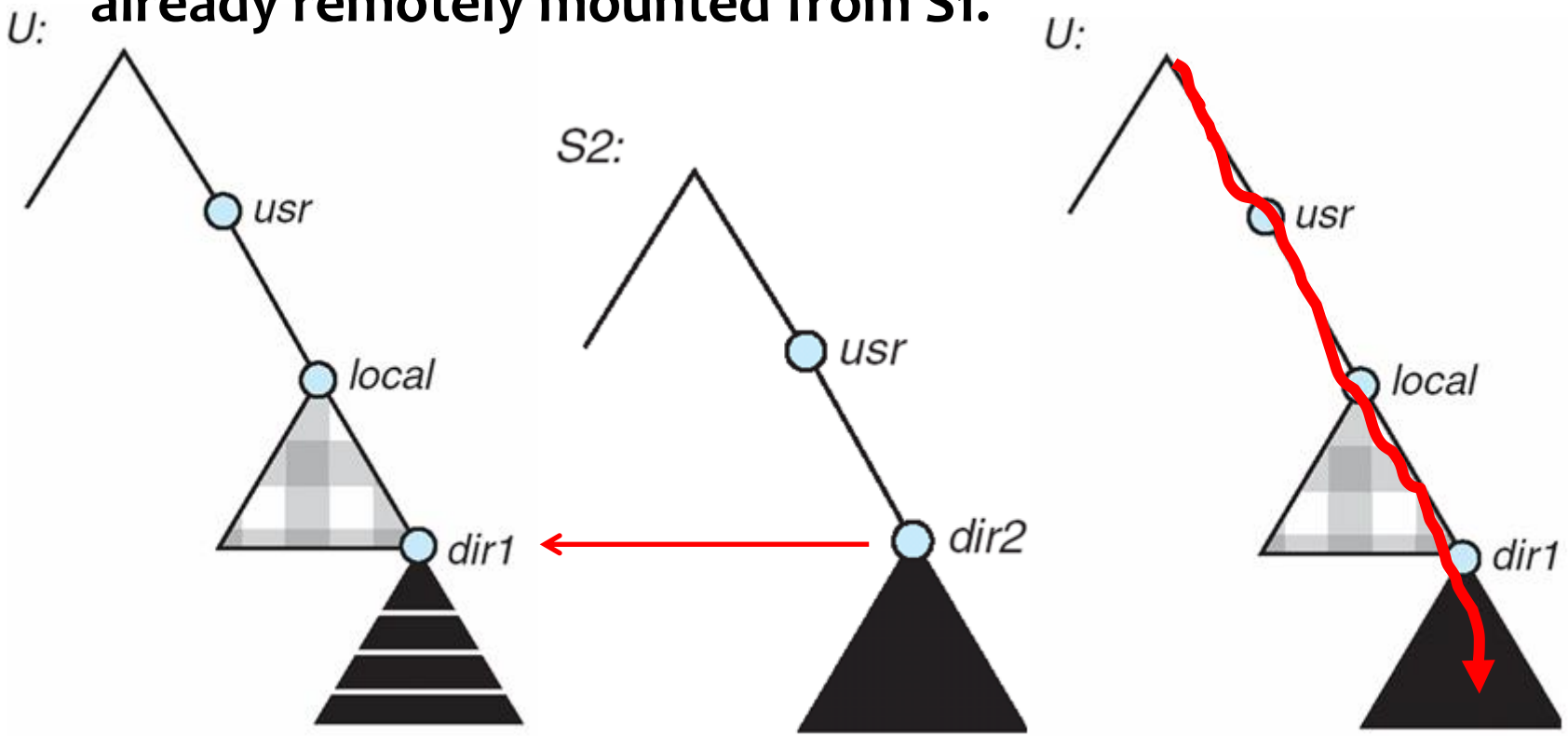
The machine U can access any file within the dir1 directory using prefix /usr/local/dir1.

The original /usr/local on that machine is no longer visible.

NFS (Cont.)

Cascading mounts.

mounting S2:/user/dir2 over U:/user/local/dir1 which is already remotely mounted from S1.



Users can access files within the dir2 on U using prefix /usr/local/dir1.

NFS (Cont.)

User mobility

If a shared file system is mounted over a user's home directories on all machines in a network, the user can log into any workstation and get his home environment.

NFS (Cont.)

One of the design goals of NFS was 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

NFS (Cont.)

The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services.

Accordingly, two separate protocols are specified for these services:

- a mount protocol and
- an NFS protocol for remote file accesses.

The protocols are specified as sets of RPCs.

- These RPCs are the building blocks used to implement transparent remote file access.

NFS Mount Protocol

The mount protocol establishes initial logical connection between a server and a client.

A mount operation includes

name of remote directory to be mounted and

name of server machine storing it

Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine

The server maintains an export list – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them

NFS Mount Protocol

In Solaris, this list is the /etc/dfs/dfstab, which can be edited only by a supervisor.

Any directory within an exported file system can be mounted by an accredited machine.

A component unit is such a directory.

Following a mount request that conforms to its export list, the server returns a file handle — a key for further accesses

In UNIX terms, the file handle consists of a file-system identifier, and an inode number to identify the mounted directory within the exported file system

The mount operation changes only the user's view and does not affect the server side

NFS Protocol

The NFS protocol provides a set of RPCs 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

These procedures can be invoked only after a file handle for the remotely mounted directory has been established.

NFS Protocol

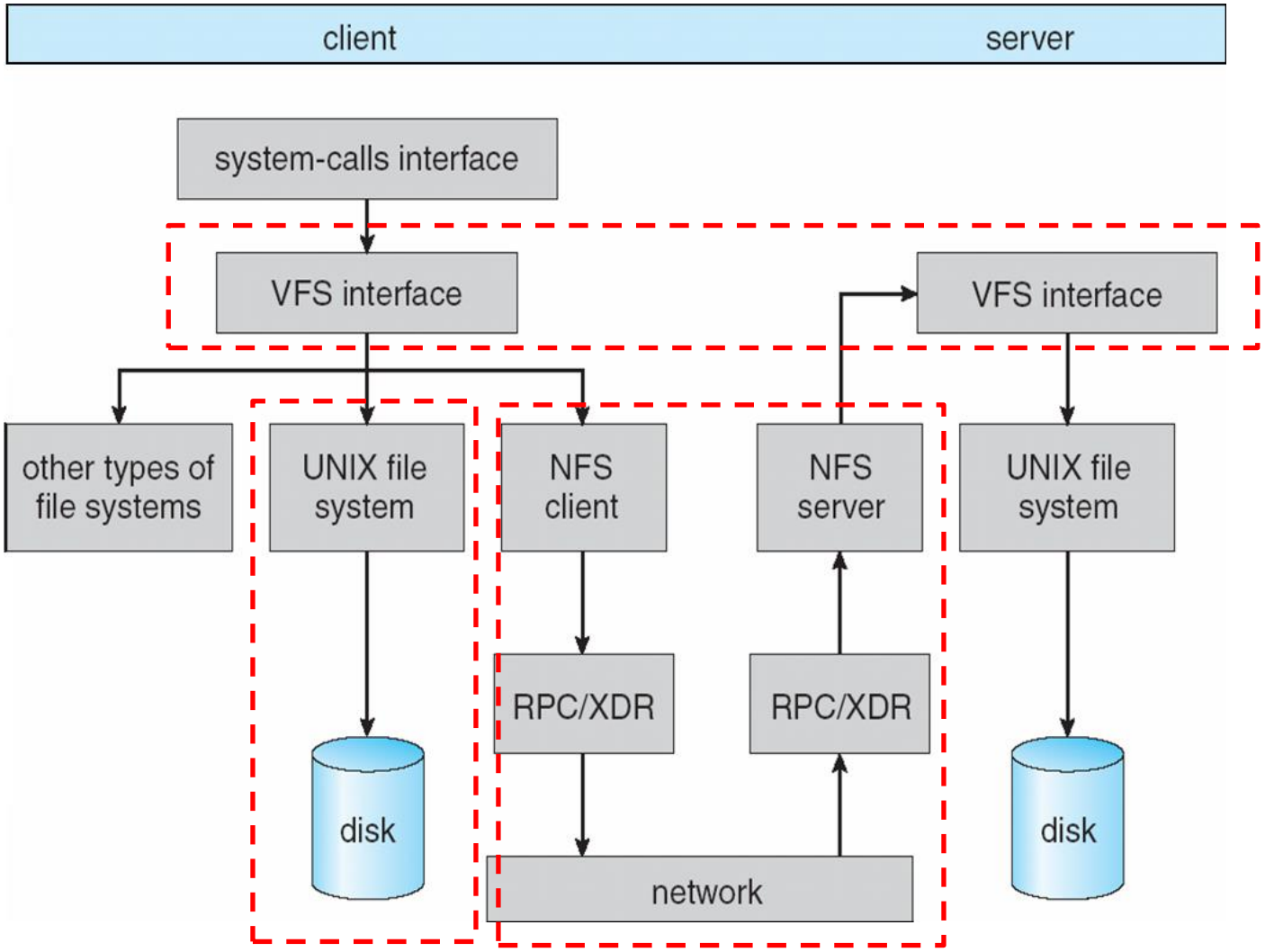
NFS servers are stateless; Servers do not maintain information about their clients from one access to another. Each request has to provide a full set of arguments.

Every NFS request has a sequence number, allowing the server to determine if a request is duplicated or missed.

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

Schematic View of NFS Architecture



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

An illustration of NFS Architecture

NFS is integrated into the OS via a VFS.

Let's trace how the operation on an already open remote file is handled.

The client initiates the operation with a regular system call.

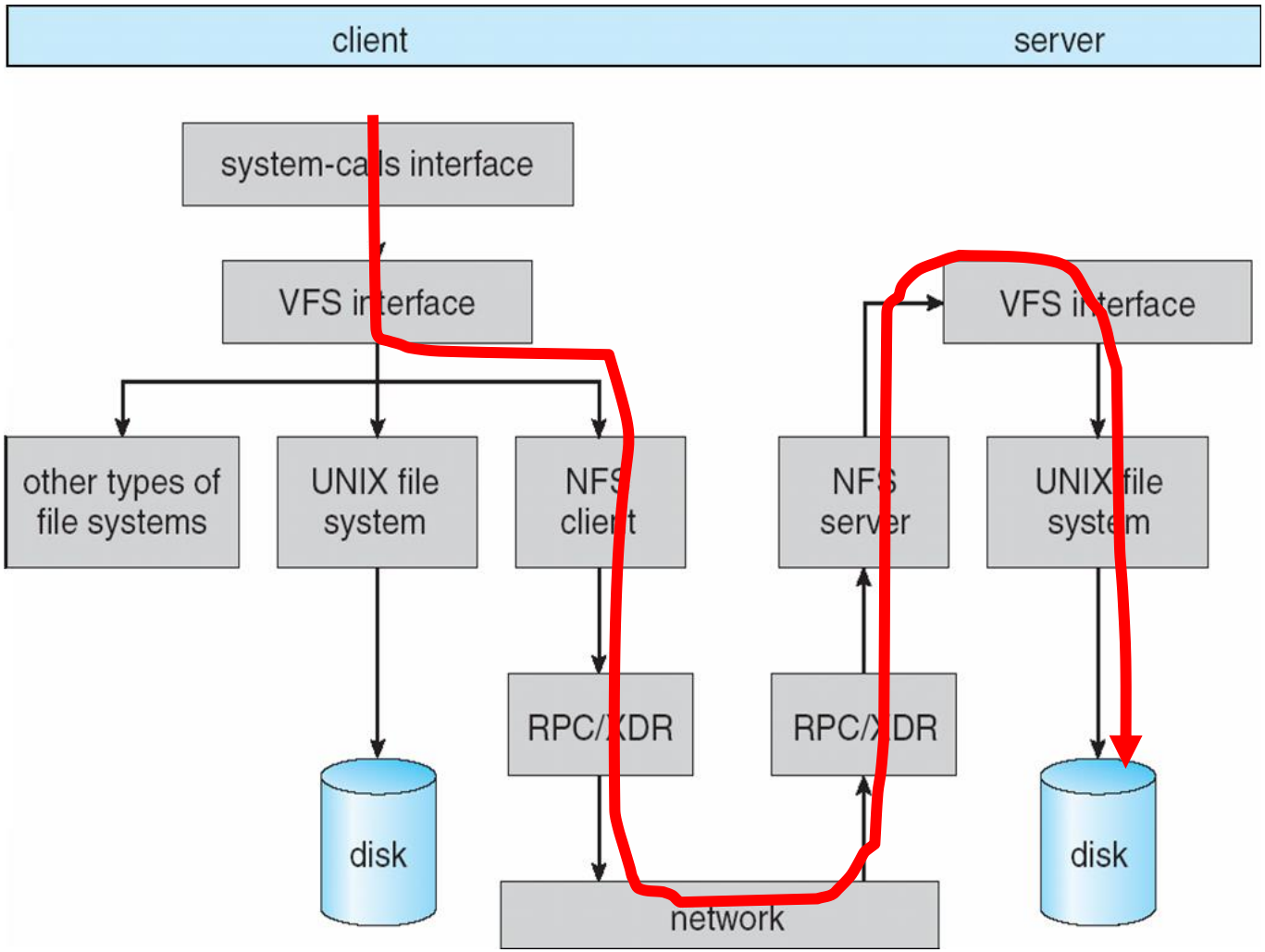
The OS layer maps this call to a VFS operation on the appropriate vnode.

The VFS layer identifies the file as a remote one and invokes the appropriate NFS procedure.

An RPC call is made to the NFS service layer at the remote server. This call is reinjected to the VFS layer on the remote system, which finds that it is local and invokes the appropriate file system operation.

This path is retraced to return the results.

Schematic View of NFS Architecture



NFS Path-Name Translation

Path-name translation in NFS involves the parsing of a path name such as /usr/local/dir1/file.txt into separate directory entries, or components: (1) usr, (2) local, and (3) dir1.

Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode.

Once a mount point is crossed, every component lookup causes a separate RPC to the server.

To make lookup faster, a directory name lookup cache on the client's side holds the vnodes for remote directory names

NFS Remote Operations

Nearly one-to-one correspondence between regular UNIX system calls for file operations and the NFS protocol RPCs (except opening and closing files).

A remote file operation can be translated directly to the corresponding RPC.

NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance.

There are two caches: the file-attribute cache and the file-blocks cache.

NFS Remote Operations

When a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes.

The cached file blocks are used only if the corresponding cached attributes are up to date.

The attribute cache is updated whenever new attributes arrive from the server.

Cached attributes are discarded after 60 seconds.

Both read-ahead and delayed-write techniques are used between the server and the client.

Clients do not free delayed-write blocks until the server confirms that the data have been written to disk.

Example: WAFL File System

Disk I/O has a huge impact on system performance.

Some file systems are general purpose, others are optimized for specific tasks,

The WAFL ("Write-anywhere file layout") file system from Network Appliance, is a powerful file system optimized for random writes.

Used on Network Appliance "Files" – distributed file system appliances

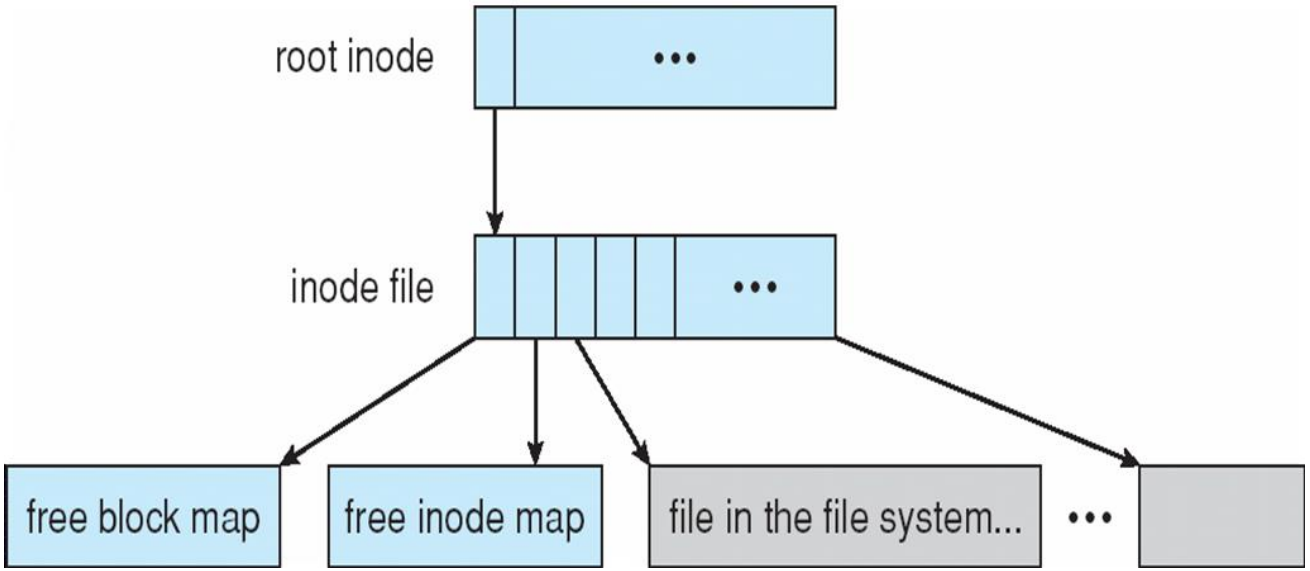
Serves up NFS, CIFS, http, ftp protocols

Random I/O optimized, write optimized

Example: WAFL File System

Similar to Berkeley Fast File System, with extensive modifications.

It is block-based and uses inodes to describe files. Each inode contains 16 pointers to blocks (or indirect blocks) belonging to the file described by the inode.



The WAFL File Layout

Example: WAFL File System

Each file system has a root inode.

A WAFL file system is a tree of blocks with the root inode as its base.

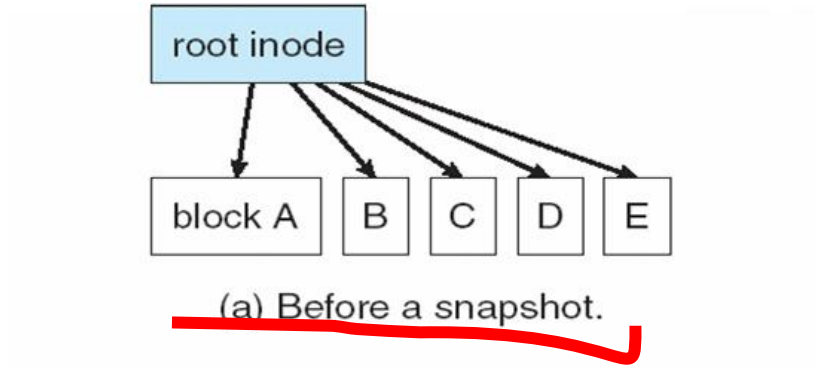
To take a snapshot, WAFL creates a copy of the root inode.

Any file or metadata updates after that go to new blocks rather than overwriting their existing blocks.

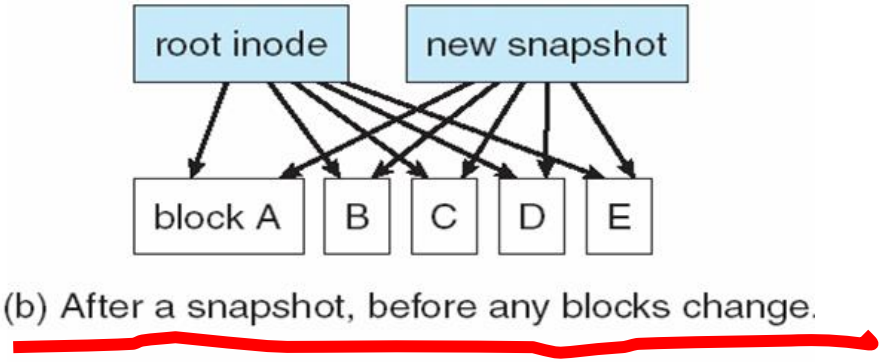
Used blocks are never overwritten, so writes are very fast, because a write can occur at the free block nearest the current head location.

The snapshot facility is also useful for backups, testing, versioning, and so on.

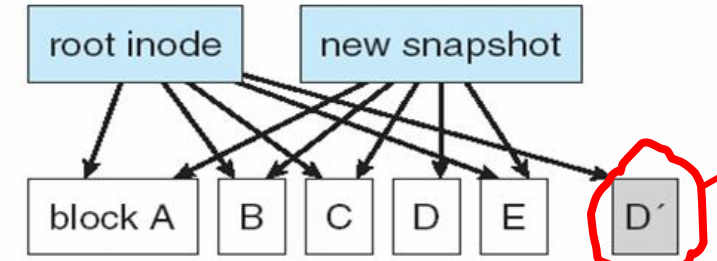
Snapshots in WAFL



(a) Before a snapshot.



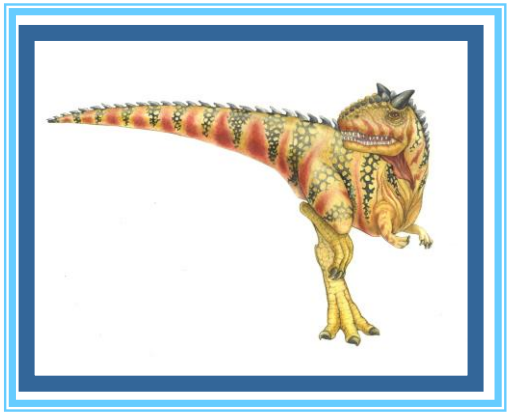
(b) After a snapshot, before any blocks change.



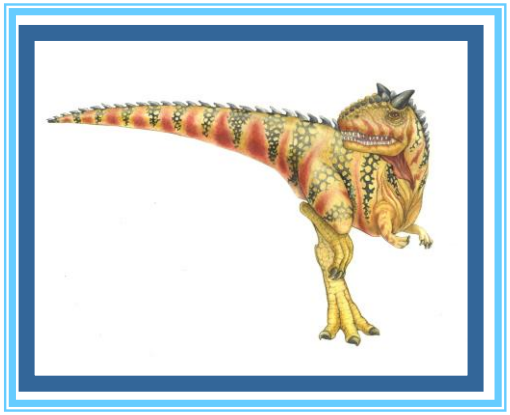
(c) After block D has changed to D´.

A write can occur at the free block nearest the current head location

End of Chapter 11



Chapter 12
Secondary-Storage Structure



Chapter 12: Secondary-Storage Structure

- Overview of Mass Storage Structure
- Disk Structure
- Disk Attachment
- Disk Scheduling
- Disk Management
- Swap-Space Management
- RAID Structure
- Stable-Storage Implementation
- Tertiary Storage Devices

Objectives

Describe the physical structure of secondary and tertiary storage devices and the resulting effects on the uses of the devices

Explain the performance characteristics of mass-storage devices

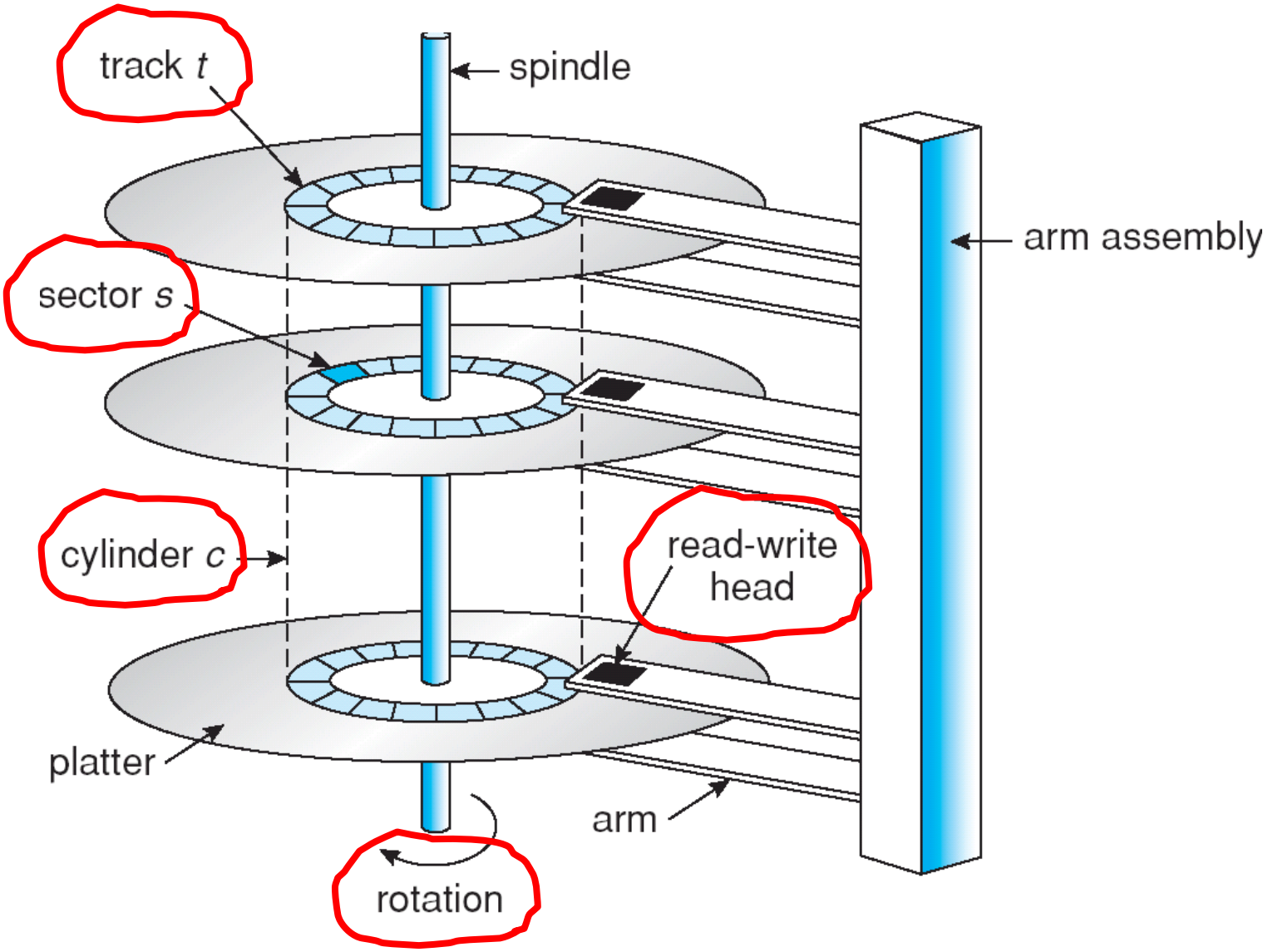
Discuss operating-system services provided for mass storage, including RAID and HSM (Hierarchical Storage Management)

12.1 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

Moving-head Disk Mechanism



Overview of Mass Storage Structure

- Disks can be removable
- Drive attached to computer via I/O bus
  - Busses vary, including EIDE, ATA, SATA, USB, Fibre Channel (FC), SCSI
  - Host controller in computer uses bus to talk to disk controller built into drive or storage array

Overview of Mass Storage Structure (Cont.)

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

12.2 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.

12.3 Disk Attachment

Host-attached storage accessed through I/O ports talking to I/O busses

SCSI itself is a bus, up to 16 devices on one cable, SCSI initiator requests operation and SCSI targets perform tasks

- Each target can have up to 8 logical units (disks attached to device controller)

FC (Fiber Channel) is high-speed serial architecture

- Can be switched fabric with 24-bit address space – the basis of storage area networks (SANs) in which many hosts attach to many storage units
- Can be arbitrated loop (FC-AL) of 126 devices

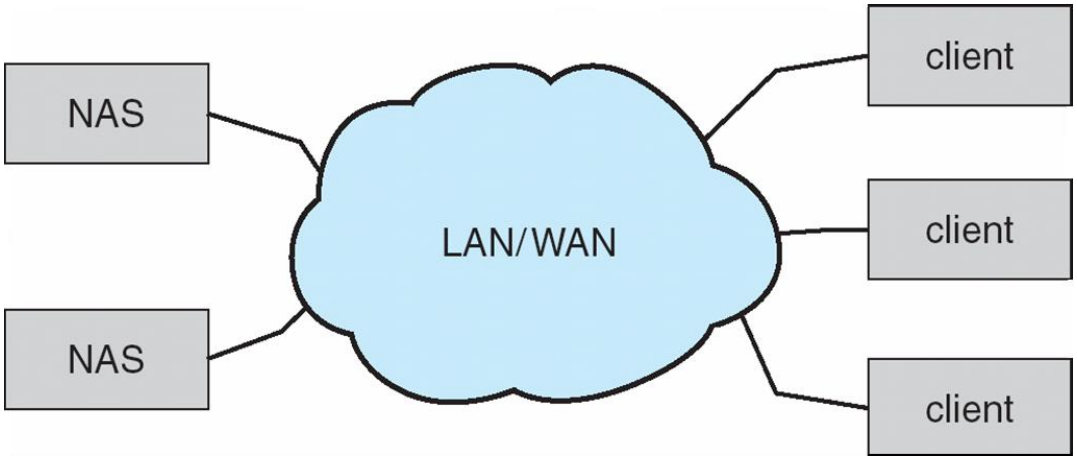
Network-Attached Storage (NAS)

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

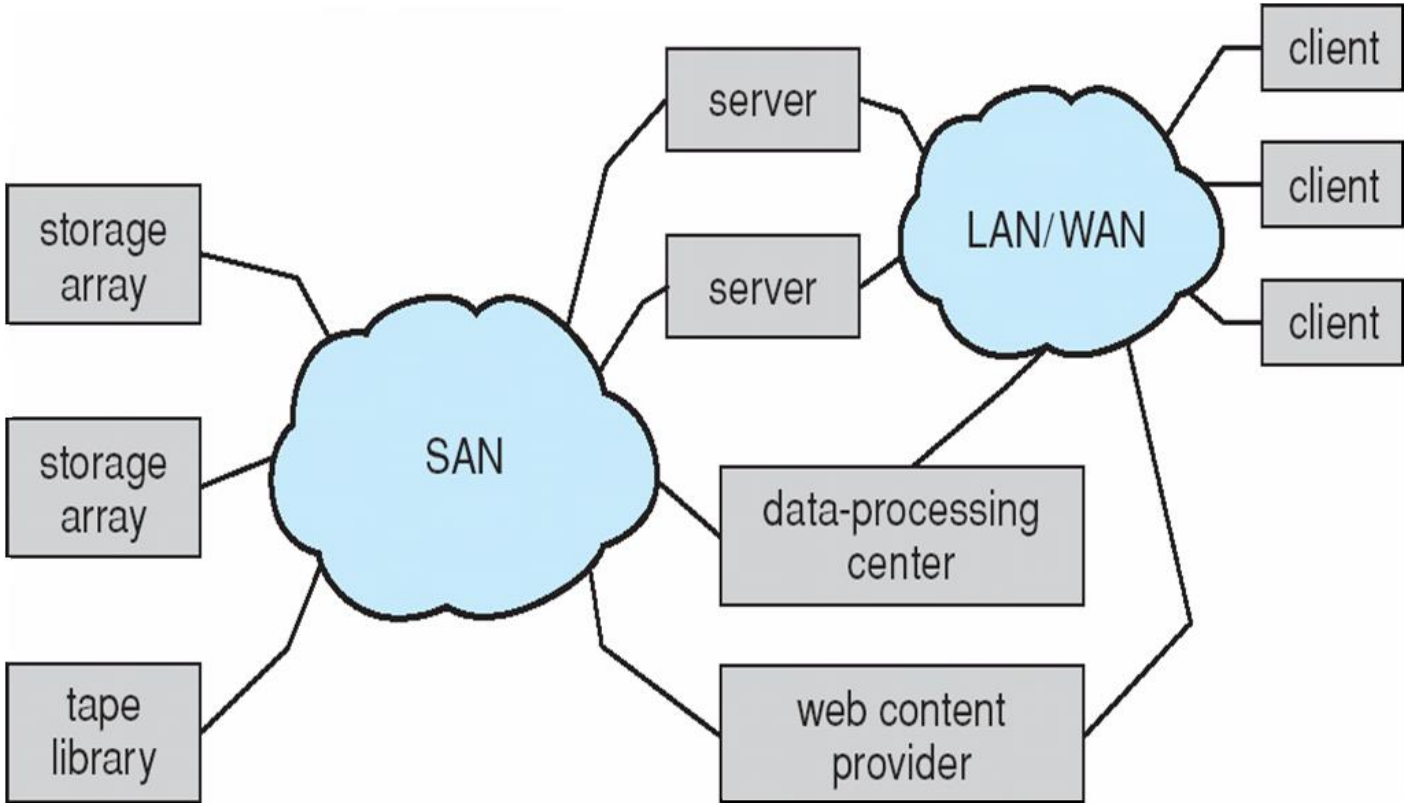
New iSCSI protocol uses IP network to carry the SCSI protocol



Storage Area Network (SAN)

Common in large storage environments (and becoming more common)

Multiple hosts attached to multiple storage arrays - flexible



12.4 Disk Scheduling

The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.

Access time has two major components

Seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector.

Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head.

Minimize seek time

Seek time ≈ seek distance

Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

Disk Scheduling (Cont.)

Several algorithms exist to schedule the servicing of disk I/O requests.

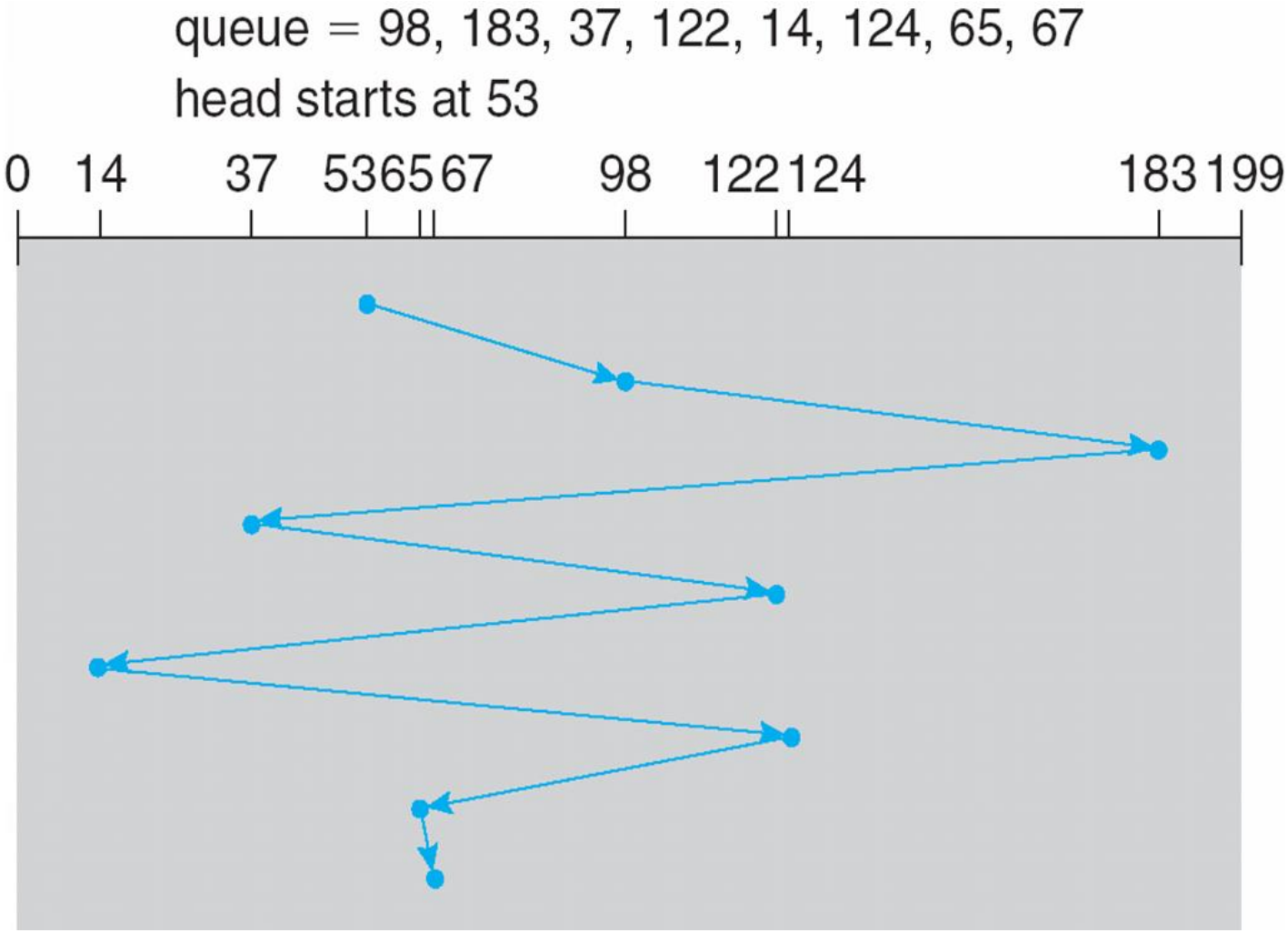
We illustrate them with a request queue (0-199 cylinders).

98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

FCFS (First Come First Service)

Illustration shows total head movement of 640 cylinders.



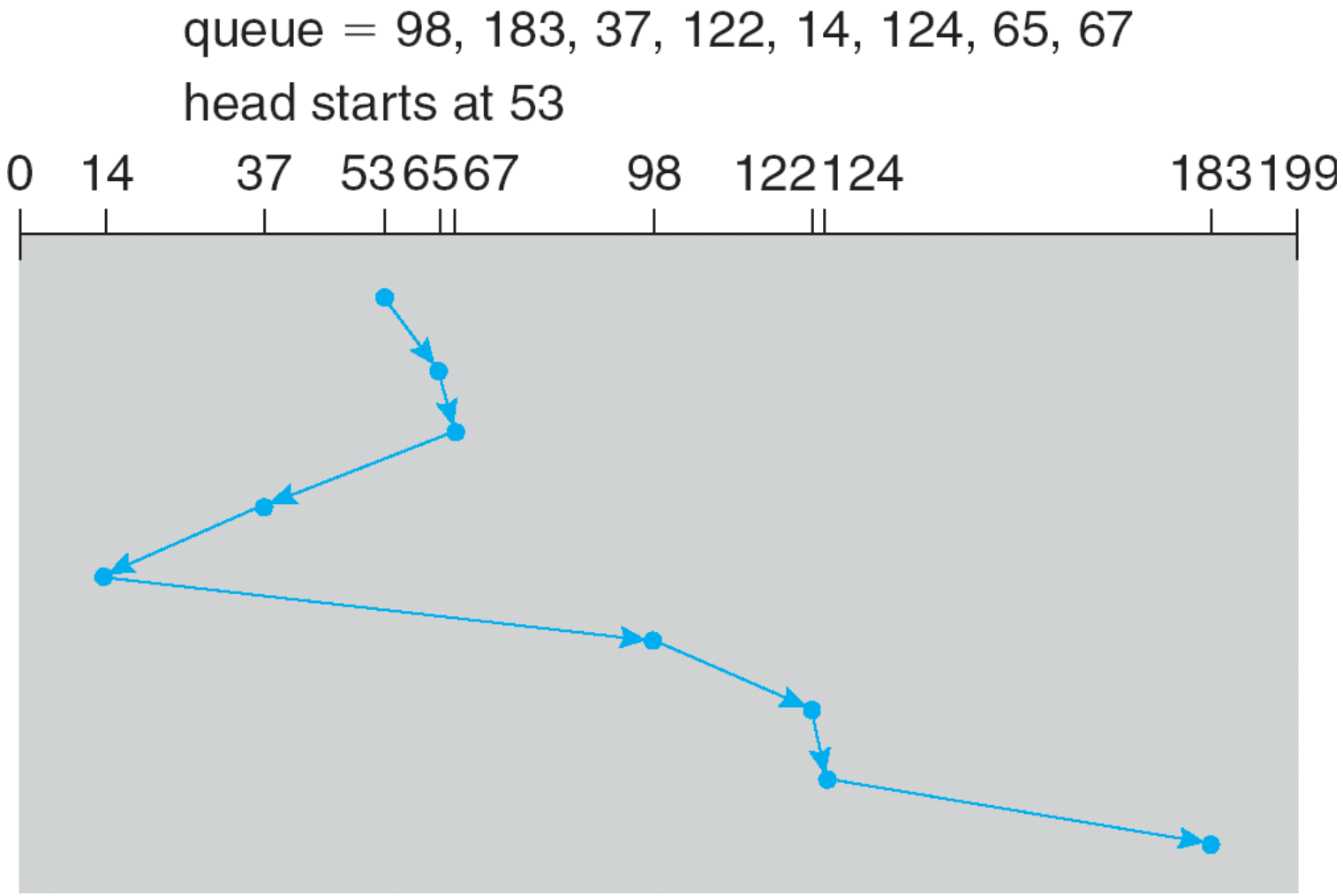
SSTF (Shortest Seek Time First)

Selects the request with the minimum seek time from the current head position.

SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

Illustration shows total head movement of 236 cylinders.

SSTF (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

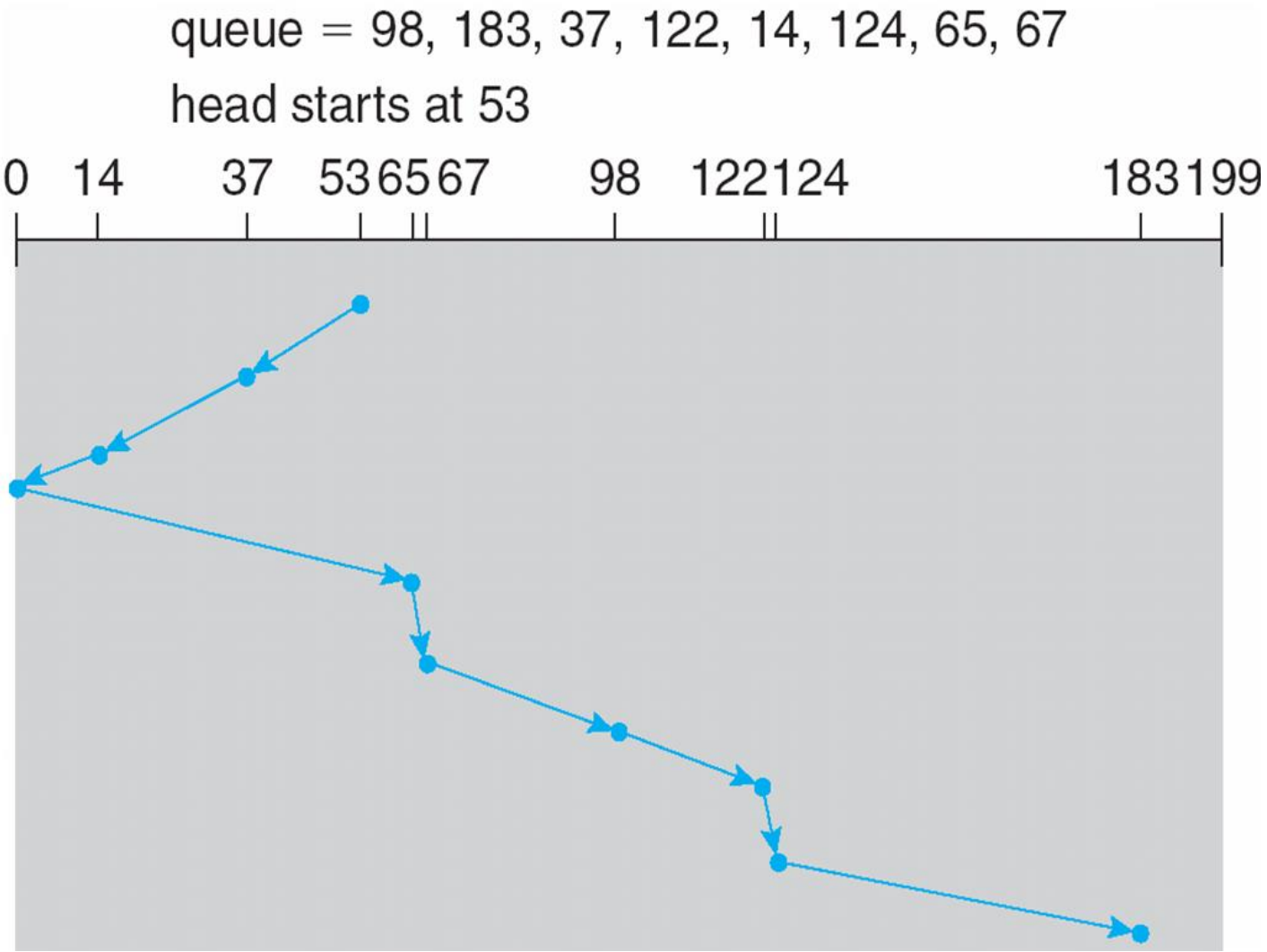
SCAN

The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

Sometimes called the *elevator algorithm*.

Illustration shows total head movement of 208 cylinders.

SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

C-SCAN

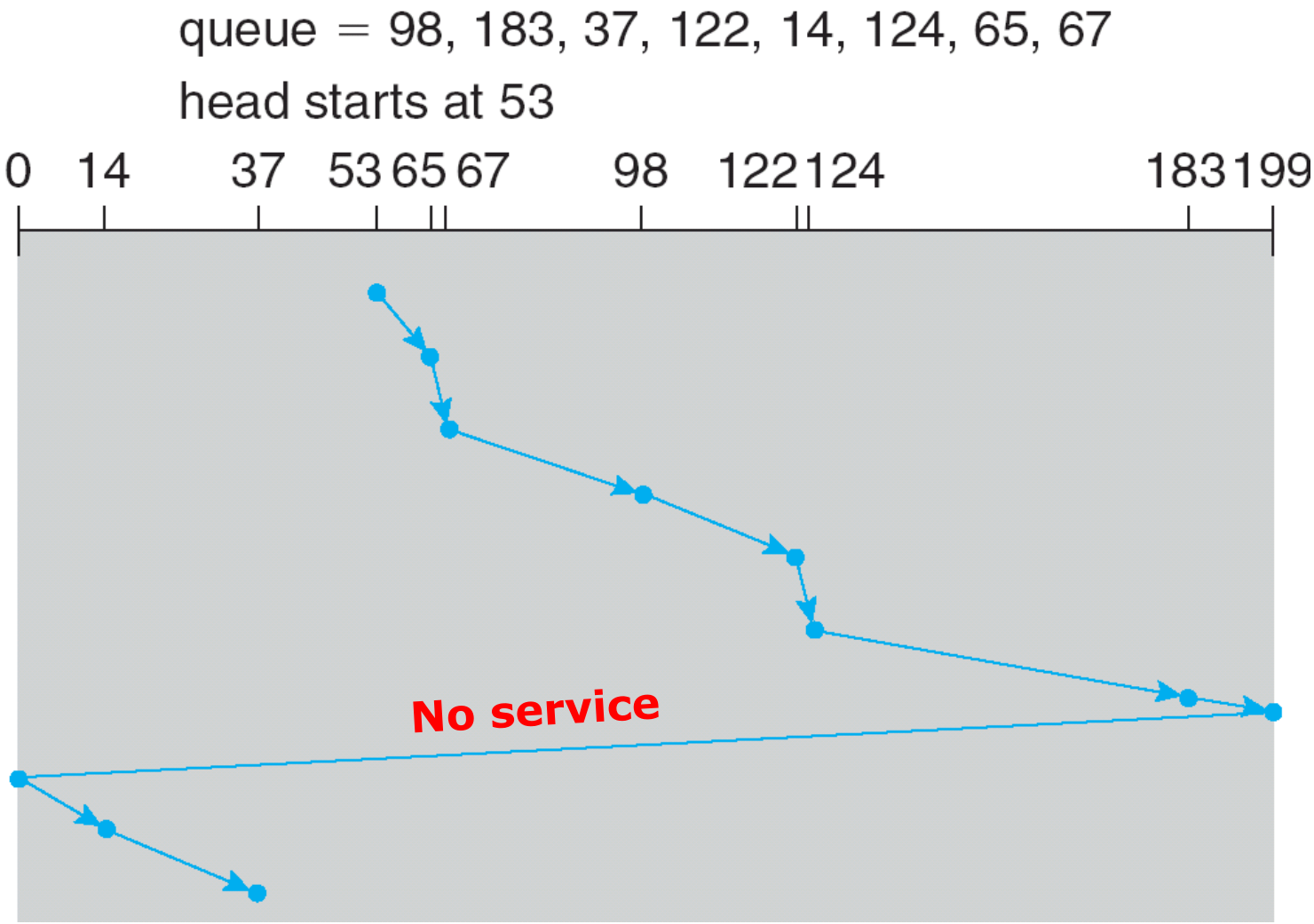
Provides a more uniform wait time than SCAN.

The head moves from one end of the disk to the other. servicing requests as it goes.

When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

C-SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199
No service

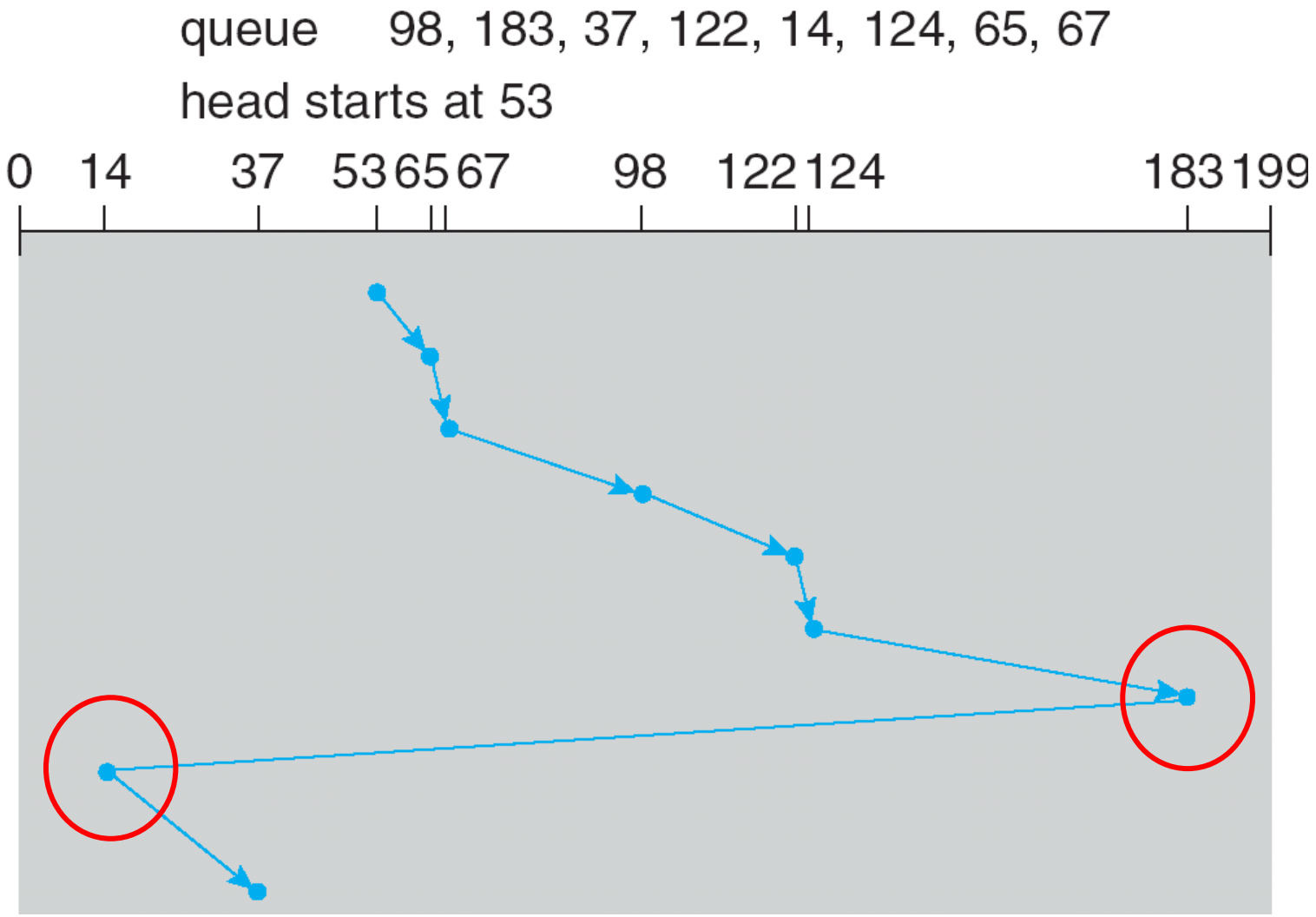
C-LOOK (or LOOK)

Versions of SACN and C-SCAN

Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

They look for a request before continuing to move in a given direction.

C-LOOK (Cont.)


queue 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

Selecting a Disk-Scheduling Algorithm

SSTF is common and has a natural appeal

SCAN and C-SCAN perform better for systems that place a heavy load on the disk.

Performance depends on the number and types of requests.

Requests for disk service can be influenced by the file-allocation method.

The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.

Either SSTF or LOOK is a reasonable choice for the default algorithm.

12.5 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".

Boot block initializes system.

- The bootstrap is stored in ROM.
- *Bootstrap loader* program.

Methods such as *sector sparing* used to handle bad blocks.

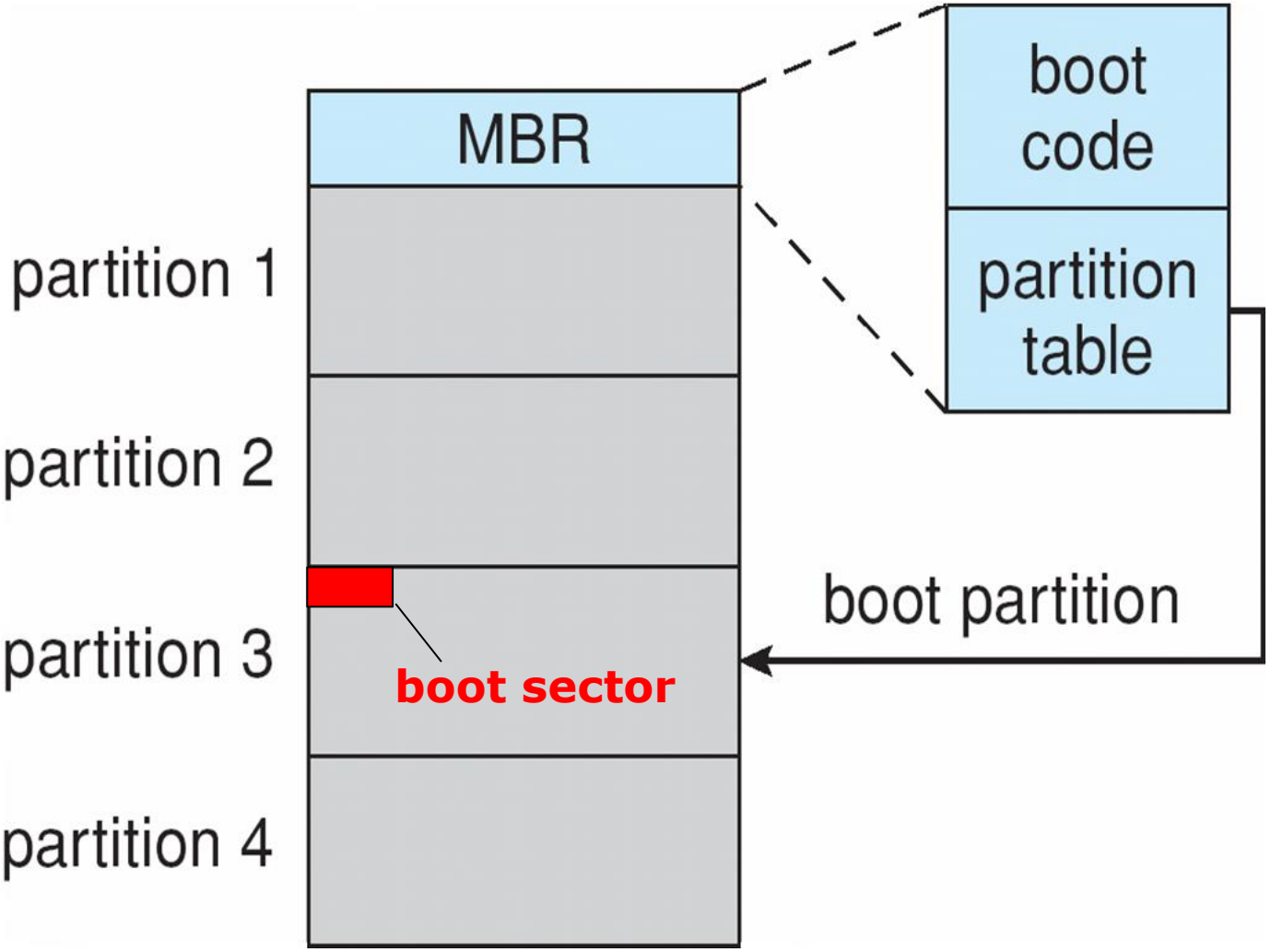
Booting from a Disk in Windows 2000

The Windows 2000 system places it boot code in the first sector on the hard disk (Master boot record, MBR).

Windows 2000 allows a hard disk to be divided into one or more partitions; one partition, identified as the boot partition, contains the OS ad device drivers.

Once the system identifies the boot partition, it reads the first sector from that partition (which is called the boot sector) and continues with the remainder of the boot process.

Booting from a Disk in Windows 2000



12.6 Swap-Space Management

Swap-space — Virtual memory uses disk space as an extension of main memory.

Swap-space can be carved out of the normal file system or, more commonly, it can be in a separate disk partition.

Swap-space management

- **4.3BSD allocates swap space when process starts; holds *text segment* (the program) and *data segment*.**
- **Kernel uses *swap maps* to track swap-space use.**
- **Solaris 2 allocates swap space only when a page is forced out of physical memory, not when the virtual memory page is first created.**

Swapping on Linux Systems

Linux allows one or more swap areas to be established.

A swap area may be either a swap file on a regular file system or a raw-swap-space partition.

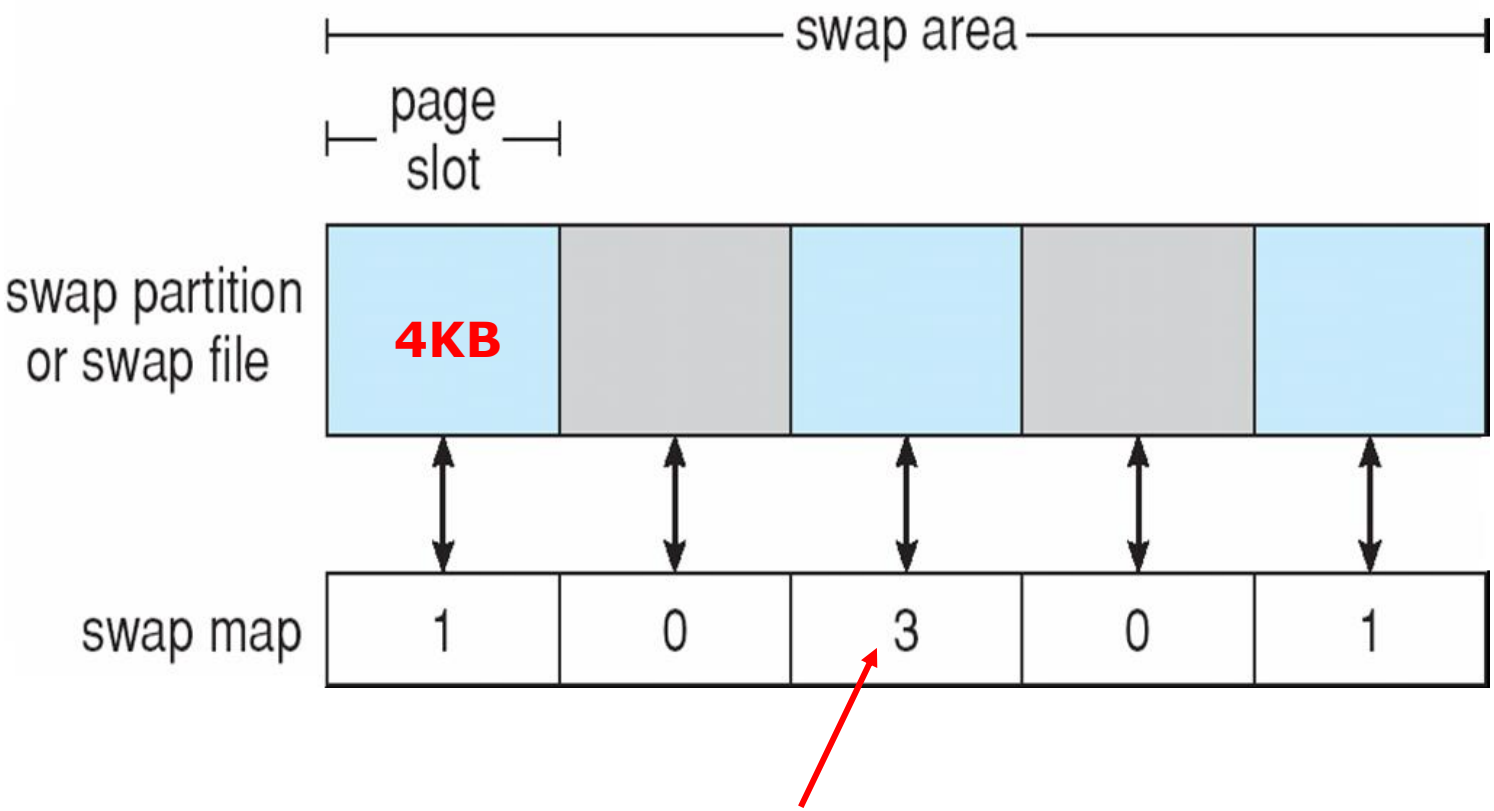
Each swap area consists of a series of 4KB page slots, which are used to hold swapped pages.

Each swap area is associated with a swap map.

Counter = 0, the corresponding page slot is available.

Counter >0, the page slot is occupied by a swapped page. The value of the counter indicates the number of mappings to the swapped page. Counter =3, the swapped paged is storing a region of memory shared by three processes.

Data Structures for Swapping on Linux Systems



The value of the counter indicates the number of mappings to the swapped page

12.7 RAID Structure

RAID (Redundant Arrays of Independent Disks) – multiple disk drives provides reliability via redundancy.

RAID is arranged into seven different levels.

With multiple disks, we can improve the transfer rate by striping data across the disks.

Data striping consists of splitting the bits of each byte across multiple disks – bit level striping

Block-level striping consists of splitting the blocks of each file across multiple disks.

RAID (cont)

Several improvements in disk-use techniques involve the use of multiple disks working cooperatively.

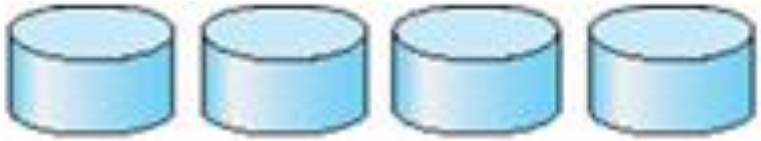
Disk striping uses a group of disks as one storage unit.

RAID schemes improve performance and improve the reliability of the storage system by storing redundant data.

- *Mirroring* or *shadowing* keeps duplicate of each disk.
- *Block interleaved parity* uses much less redundancy.

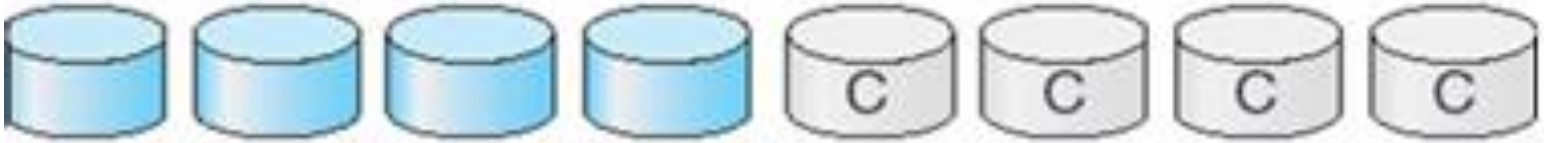
RAID (cont)

RAID level 0: RAID level 0 refers to disk arrays with striping at the level of blocks but without any redundancy.



(a) RAID 0: non-redundant striping.

RAID level 1: RAID level 1 refers to disk mirroring.



(b) RAID 1: mirrored disks.

RAID (cont)

RAID level 2: RAID level 2 also known as memory-style error-correcting-code (ECC) organization. The parity bits are used. The disks labeled P store the error-correction bits.

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

RAID (cont)

RAID level 3: RAID level 3 refers to bit-interleaved parity organization improves on level 2 by taking into account the fact that, unlike memory systems, disk controllers can detect whether a sector has been read correctly, so a single parity bit can be used for error correction and for detection.

(d) RAID 3: bit-interleaved parity.

RAID (cont)

RAID level 4: RAID level 0 refers to block-interleaved parity organization, uses block-level striping, as in RAID 0, and also keeps a parity block on a separate disk for corresponding blocks from N other disks.

(e) RAID 4: block-interleaved parity.

RAID (cont)

RAID level 5: RAID level 5 refers to block-interleaved distributed parity, differs from level 4 by spreading data and parity among all N+1 disks, rather than storing data in N disks and parity in one disk.

For each block, one of the disks stores the parity, and the others store data.

With an array of five disks, the parity for the *n*th block is stored in disk (*n* mod 5)+1; the *n*th blocks of the other four disks store actual data for the block.

A parity block cannot store parity for the blocks in the same disk.

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

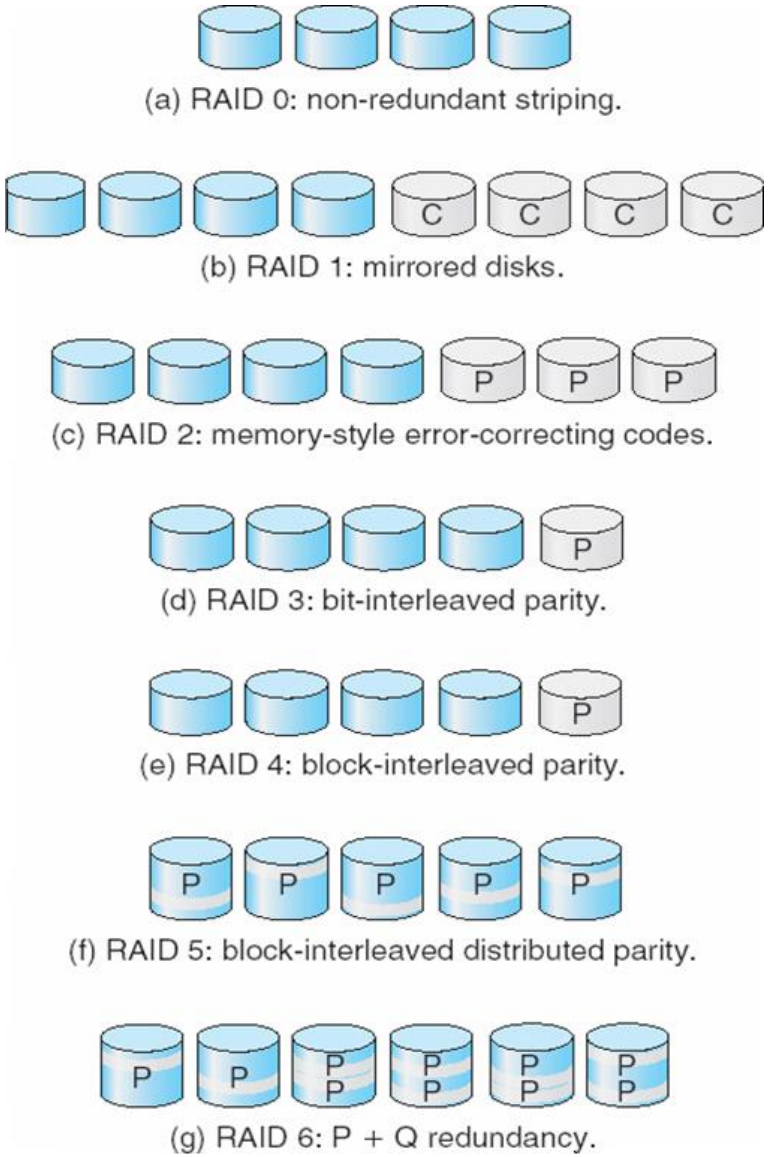
RAID (cont)

RAID level 6, P+Q redundancy scheme, is much like RAID 5 but stores extra redundant information to guard against multiple disk failures.

Instead of parity, error-correcting codes such as Reed-Solomon codes are used. 2 bits of redundant data are stored for every 4 bits of data – compared with 1 parity bit in level 5 – and the system can tolerate two disk failures.

(g) RAID 6: P + Q redundancy.

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.

RAID (cont)

RAID 0 provides the performance,

RAID 1 provides the reliability.

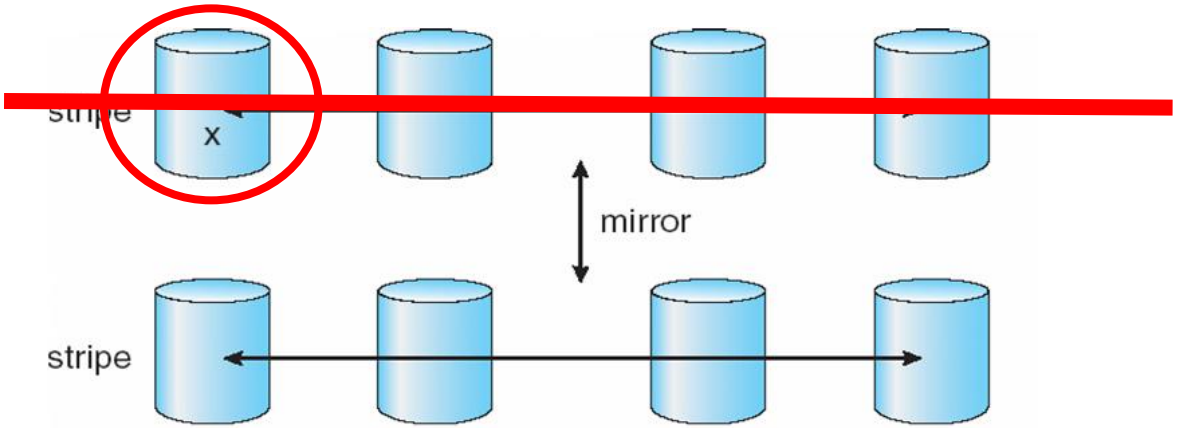
RAID 0 +1 : A set of disks are striped, and then the stripe is mirrored to another, equivalent strip

RAID 1+0: Disks are mirrored in pairs and then the resulting mirrored pairs are striped.

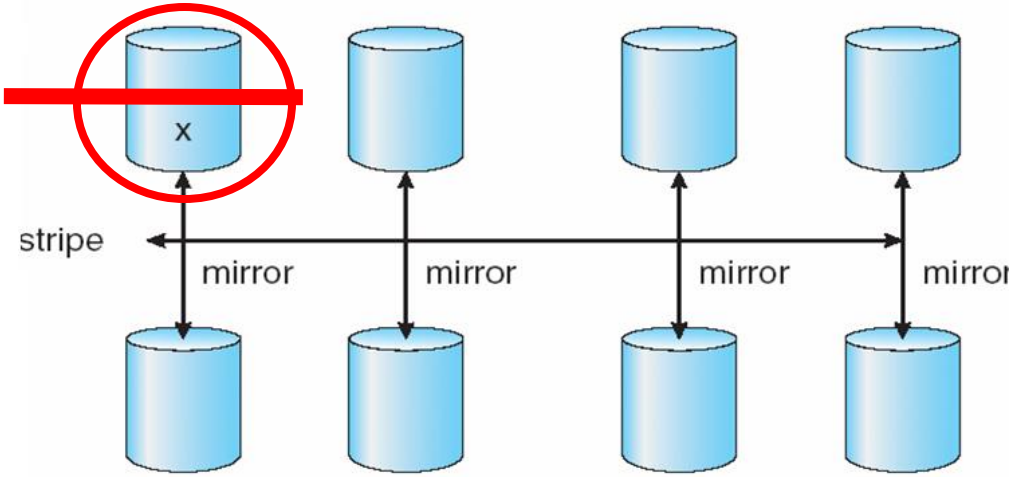
RAID 1+0 has some theoretical advantages over RAID 0+1.

For example, if a single disk fails in RAID 0+1, an entire strip is inaccessible, leaving only the other strip available. With a failure in RAID 1+0, a single disk is unavailable, but the disk that mirrors it is still available, as are all the rest of the disks.

RAID (0 + 1) and (1 + 0)



a) RAID 0 + 1 with a single disk failure.



b) RAID 1 + 0 with a single disk failure.

12.8 Stable-Storage Implementation

Write-ahead log scheme requires stable storage.

To implement stable storage:

Replicate information on more than one nonvolatile storage media with independent failure modes.

Update information in a controlled manner to ensure that we can recover the stable data after any failure during data transfer or recovery.

12.9 Tertiary Storage Structure

Low cost is the defining characteristic of tertiary storage.

Generally, tertiary storage is built using *removable media*

Common examples of removable media are floppy disks and CD-ROMs; other types are available.

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.**

Removable Disks (Cont.)

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.

Tapes

Compared to a disk, a tape is less expensive and holds more data, but random access is much slower.

Tape is an economical medium for purposes that do not require fast random access, e.g., backup copies of disk data, holding huge volumes of data.

Large tape installations typically use robotic tape changers that move tapes between tape drives and storage slots in a tape library.

stacker – library that holds a few tapes

silo – library that holds thousands of tapes

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.

Operating System Support

Major OS jobs are to manage physical devices and to present a virtual machine abstraction to applications

For hard disks, the OS provides two abstraction:

Raw device – an array of data blocks.

File system – the OS queues and schedules the interleaved requests from several applications.

Application Interface

Most OSs handle removable disks almost exactly like fixed disks — a new cartridge is formatted and an empty file system is generated on the disk.

Tapes are presented as a raw storage medium, i.e., and application does not open a file on the tape, it opens the whole tape drive as a raw device.

Usually the tape drive is reserved for the exclusive use of that application.

Since the OS does not provide file system services, the application must decide how to use the array of blocks.

Since every application makes up its own rules for how to organize a tape, a tape full of data can generally only be used by the program that created it.

Tape Drives

The basic operations for a tape drive differ from those of a disk drive.

locate positions the tape to a specific logical block, not an entire track (corresponds to seek).

The read position operation returns the logical block number where the tape head is.

The space operation enables relative motion.

Tape drives are "append-only" devices; updating a block in the middle of the tape also effectively erases everything beyond that block.

An EOT mark is placed after a block that is written.

File Naming

The issue of naming files on removable media is especially difficult when we want to write data on a removable cartridge on one computer, and then use the cartridge in another computer.

Contemporary OSs generally leave the name space problem unsolved for removable media, and depend on applications and users to figure out how to access and interpret the data.

Some kinds of removable media (e.g., CDs) are so well standardized that all computers use them the same way.

Hierarchical Storage Management (HSM)

A *hierarchical storage system* extends the storage hierarchy beyond primary memory and secondary storage to incorporate tertiary storage — usually implemented as a jukebox of tapes or removable disks.

Usually incorporate tertiary storage by extending the file system.

- Small and frequently used files remain on disk.
- Large, old, inactive files are archived to the jukebox.

HSM is usually found in supercomputing centers and other large installations that have enormous volumes of data.

Speed

Two aspects of speed in tertiary storage are bandwidth and latency.

Bandwidth is measured in bytes per second.

Sustained bandwidth – average data rate during a large transfer; # of bytes/transfer time.
Data rate when the data stream is actually flowing.

Effective bandwidth – average over the entire I/O time, including seek or locate, and cartridge switching.
Drive's overall data rate.

Speed (Cont.)

Access latency – amount of time needed to locate data.

- Access time for a **disk** – move the arm to the selected cylinder and wait for the rotational latency; **< 35 milliseconds**.
- Access on **tape** requires winding the tape reels until the selected block reaches the tape head; **tens or hundreds of seconds.**
- Generally say that random access within a tape cartridge is about **a thousand times slower than random access on disk**.

The low cost of tertiary storage is a result of having **many cheap cartridges share a few expensive drives**.

A removable library is best devoted to the storage of infrequently used data, because the library can only satisfy a relatively small number of I/O requests per hour.

Reliability

A fixed disk drive is likely to be more reliable than a removable disk or tape drive.

An optical cartridge is likely to be more reliable than a magnetic disk or tape.

A head crash in a fixed hard disk generally destroys the data, whereas the failure of a tape drive or optical disk drive often leaves the data cartridge unharmed.

Cost

Main memory is much more expensive than disk storage

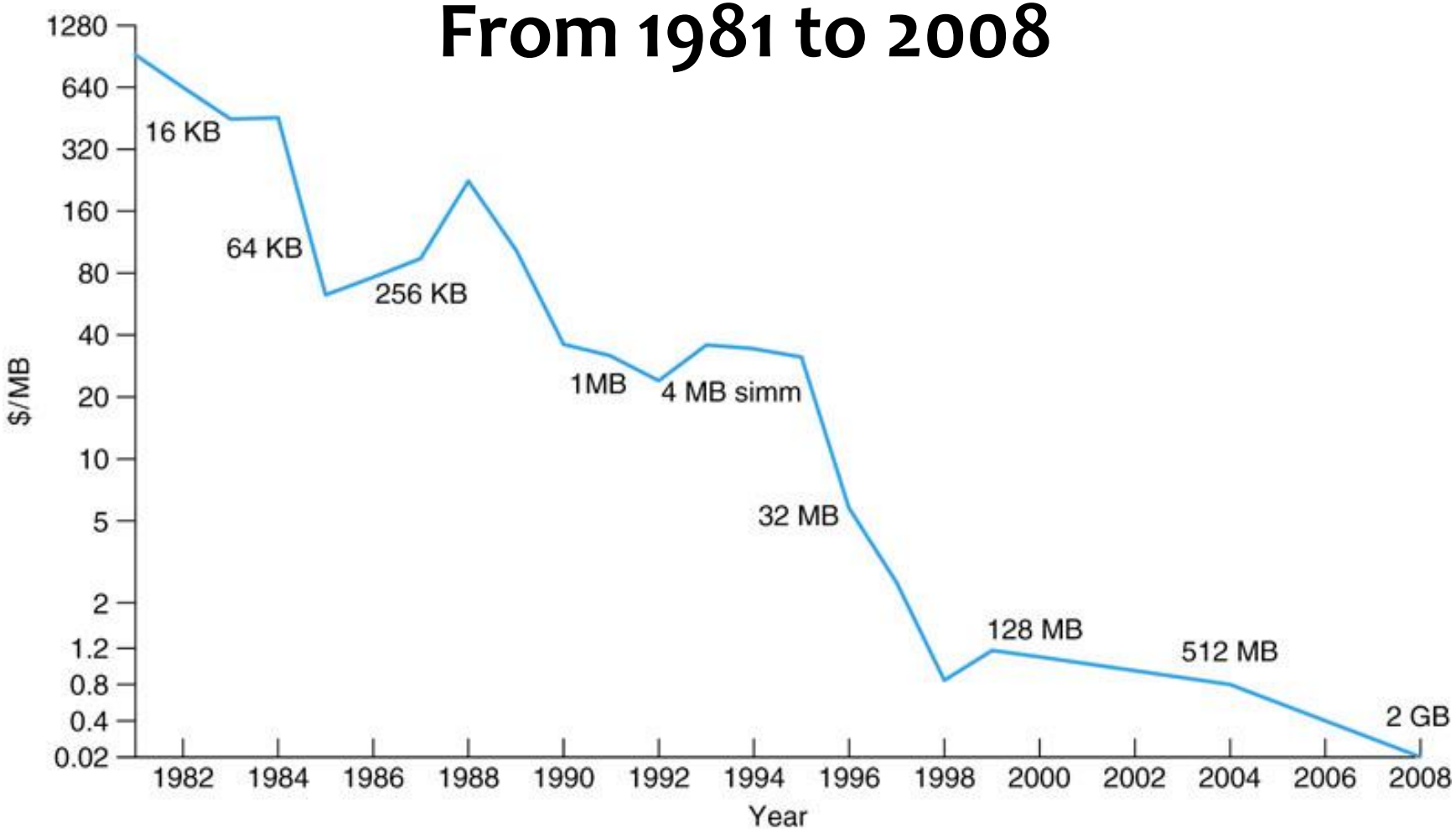
The cost per megabyte of hard disk storage is competitive with magnetic tape if only one tape is used per drive.

The cheapest tape drives and the cheapest disk drives have had about the same storage capacity over the years.

Tertiary storage gives a cost savings only when the number of cartridges is considerably larger than the number of drives.

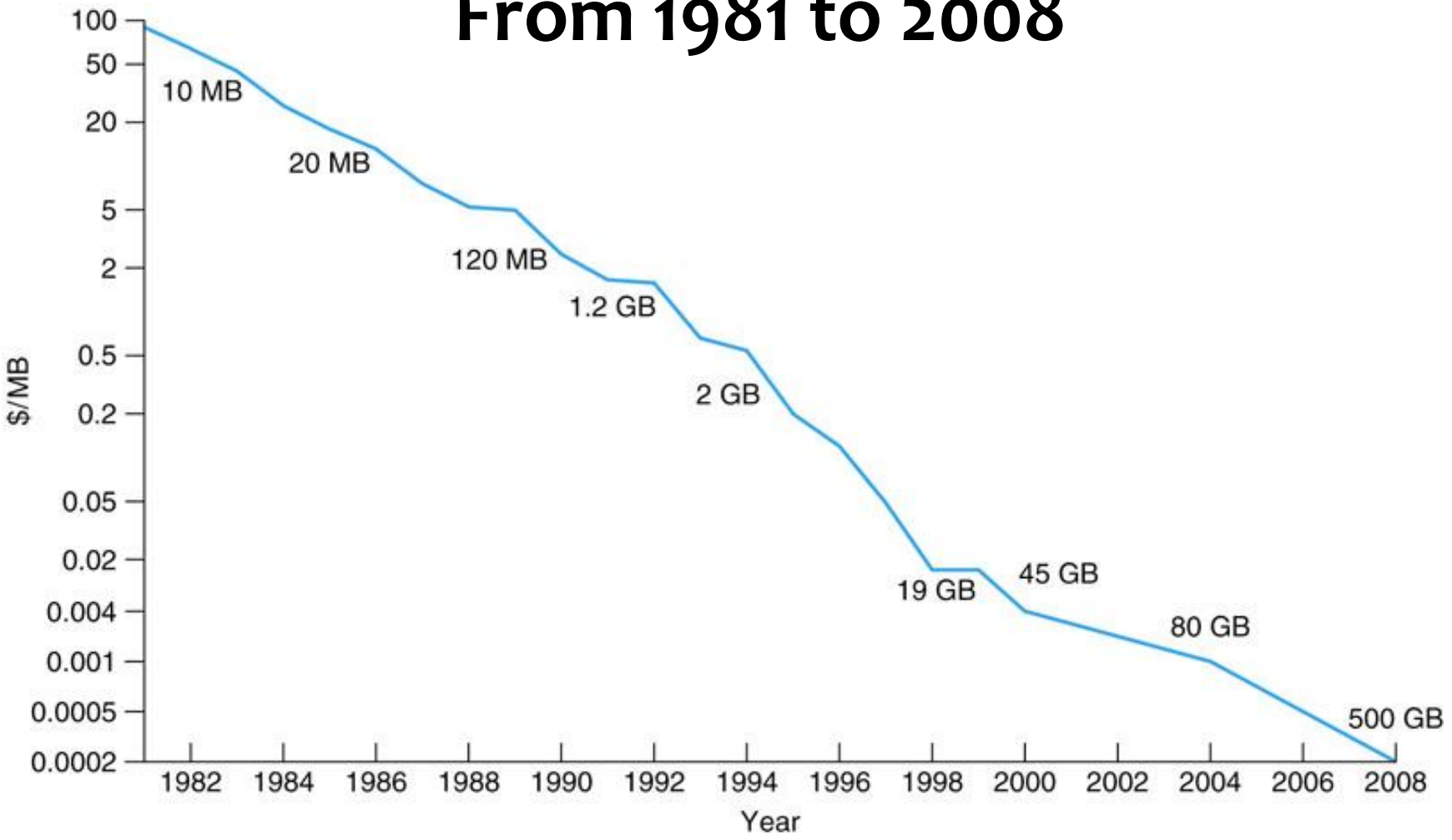
Price per Megabyte of DRAM

From 1981 to 2008



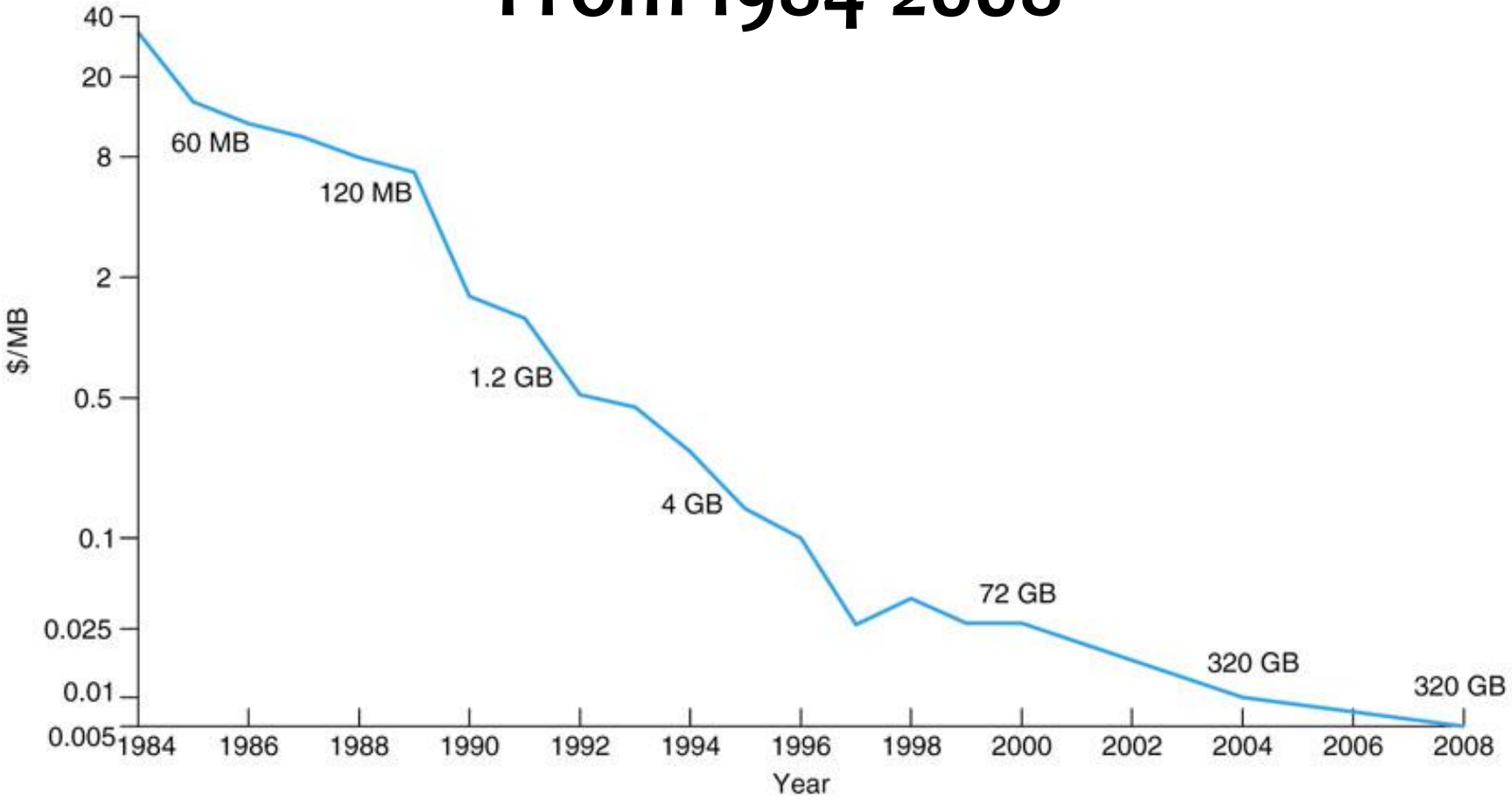
Price per Megabyte of Magnetic Hard Disk

From 1981 to 2008



Price per Megabyte of a Tape Drive

From 1984-2008



End of Chapter 12