Open Source OS

Module 3: Storage Management

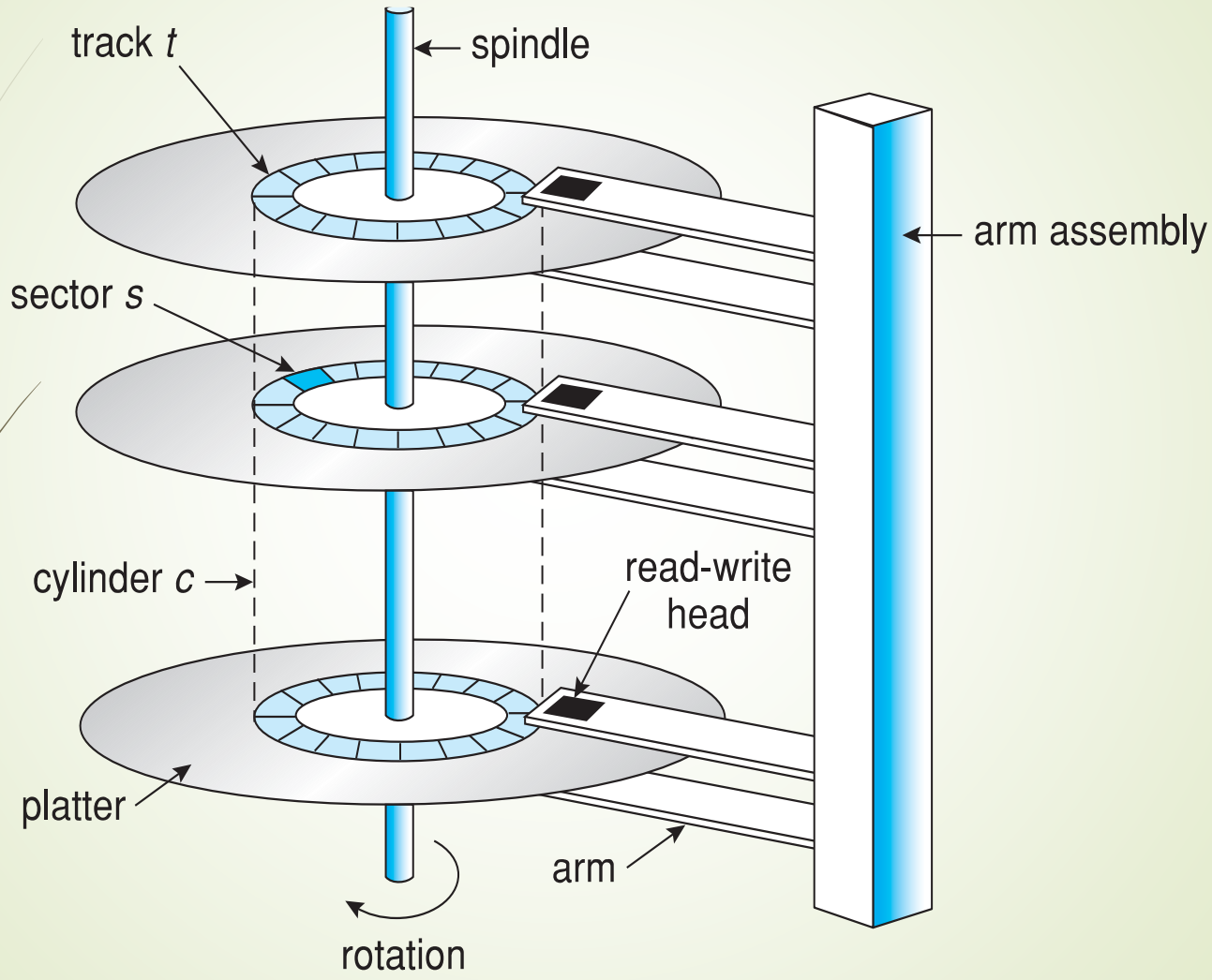
Module 3: Storage Management

- Mass Storage Structure
  - HDD Scheduling
  - NVM Scheduling
- Error Detection and Correction
- RAID Structure
- File Management
  - Disk and Directory Structure
  - File-System Operations
  - Allocation Methods
  - Efficiency and Performance
  - Recovery
- Virtual File Systems
- Remote File Systems
  - NFS

Overview of Mass Storage Structure

- Major secondary storage for modern computers is **hard disk drives** (**HDDs**) and **nonvolatile memory** (**NVM**) devices
- **HDDs** spin platters of magnetically-coated material under moving read-write heads
  - Drives rotate at 60 to 250 times per second
  - **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**)
  - **Transfer rate** is rate at which data flow between drive and computer
  - **Head crash** results from disk head making contact with the disk surface
- Disks can be removable

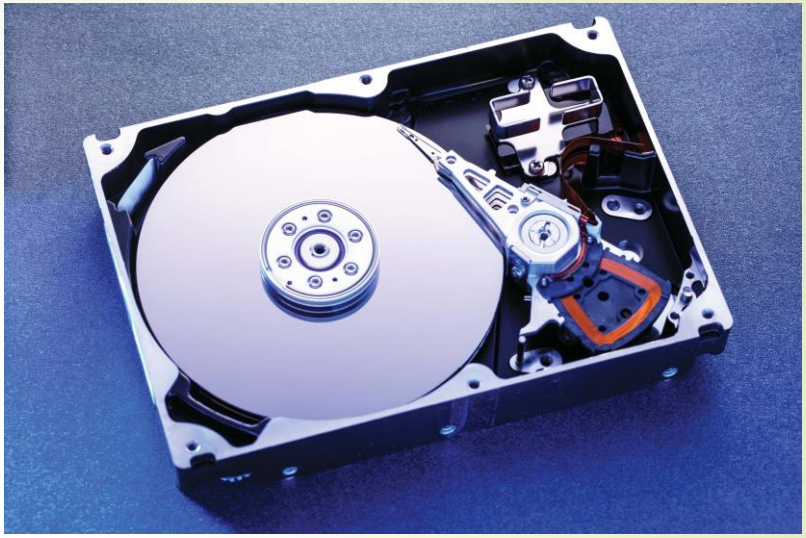
Moving-head Disk Mechanism



[Source: Operating Systems Concepts, 10th ed.]

Hard Disk Drives

- Platters range from .85″ to 14″ (historically)
  - Commonly 3.5″, 2.5″, and 1.8″
- Range from 30GB to 3TB per drive



[Source: Operating Systems Concepts, 10th ed.]

- Performance
  - Transfer Rate – theoretical – 6 Gb/sec
    - Effective Transfer Rate – real – 1Gb/sec
  - Seek time from 3ms to 12ms – 9ms common for desktop drives
    - Average seek time measured or calculated based on 1/3 of tracks
  - Latency based on spindle speed
    - 1 / (RPM / 60) = 60 / RPM
    - Average latency = ½ latency

Hard Disk Performance

- **Access Latency** = **Average access time** = average seek time + average latency
  - For fastest disk: 3ms + 2ms = 5ms
  - For slow disk: 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead
  - For example, to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead, average I/O time = 5ms + 4.17ms + 0.1ms + transfer time
  - Transfer time = 4KB / 1Gb/s * 8Gb / GB * 1GB / 1024^2KB = 32 / (1024^2) = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

The First Commercial Disk Drive

1956
IBM RAMDAC computer included the IBM Model 350 disk storage system

5M (7 bit) characters
50 x 24” platters
Access time = < 1 second

[Source: Operating Systems Concepts, 10th ed.]

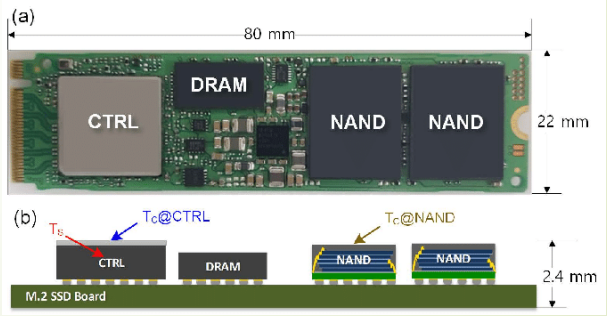
Nonvolatile Memory Devices

- If disk-drive like, then called **solid-state disks** (**SSDs**)
  - Other forms include **USB drives** (thumb drive, flash drive), DRAM disk replacements, surface-mounted on motherboards, and main storage in devices like smartphones
- Characteristics:
  - Can be more reliable than HDDs
  - Maybe have shorter life span – need careful management
  - More expensive per MB
  - Less capacity
  - But much faster
    - No moving parts, so no seek time or rotational latency
    - Buses can be too slow -> connect directly to PCI for example

Nonvolatile Memory Devices

- Have characteristics that present challenges
- Read and written in "page" increments (think sector) but can't overwrite in place
  - Must first be erased, and erases happen in larger "block" increments
  - Can only be erased a limited number of times before worn out – ~ 100,000
  - Life span measured in **drive writes per day** (**DWPD**)
    - A 1TB NAND drive with rating of 5DWPD is expected to have 5TB per day written within warrantee period without failing

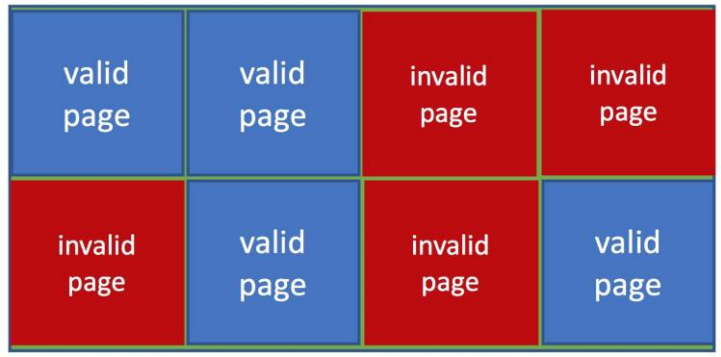
[Source: Operating Systems Concepts, 10th ed.]



[Source: ResearchGate]

NAND Flash Controller Algorithms

- With no overwrite, pages end up with mix of valid and invalid data
- To track which logical blocks are valid, controller maintains **flash translation layer** (**FTL**) table
- Also implements **garbage collection** to free invalid page space
- Allocates **overprovisioning** to provide working space for GC
- Each cell has lifespan, so **wear leveling** needed to write equally to all cells



[Source: Operating Systems Concepts, 10th ed.]

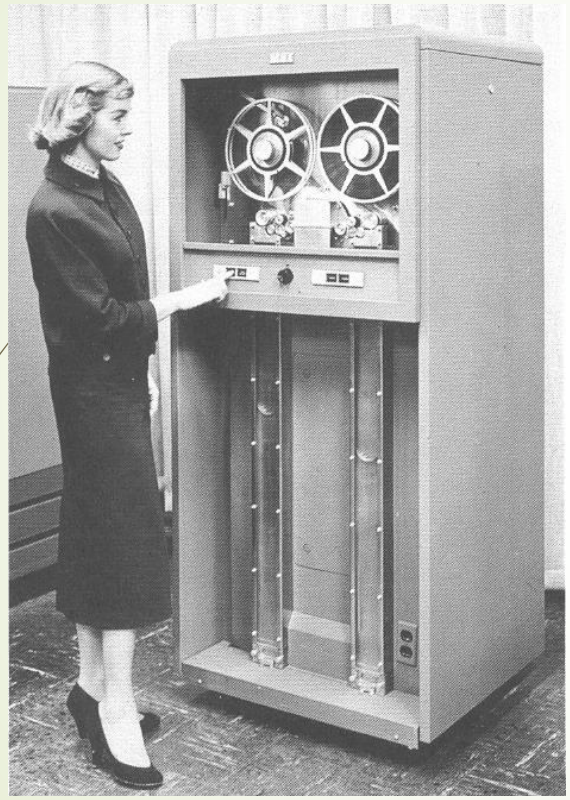
NAND block with valid and invalid pages

Volatile Memory

- DRAM frequently used as mass-storage device
  - Not technically secondary storage because volatile, but can have file systems, be used like very fast secondary storage
- **RAM drives** (with many names, including RAM disks) present as raw block devices, commonly file system formatted
- Used as high speed temporary storage
  - Programs could share bulk data, quickly, by reading/writing to RAM drive

- Computers have buffering, caching via RAM, so why RAM drives?
  - Caches / buffers allocated / managed by programmer, OS, hardware
  - RAM drives under user control
  - Found in all major OS
    - Linux `/dev/ram`
    - macOS `diskutil` to create them
    - Linux `/tmp` of file system type `tmpfs`

Magnetic Tape



[Source: Columbia University]

[Source: DataCenters.com]

Disk Attachment

- Host-attached storage accessed through I/O ports talking to **I/O buses**
  - Several busses available, including **advanced technology attachment** (**ATA**), **serial ATA** (**SATA**), **eSATA**, **serial attached SCSI** (**SAS**), **universal serial bus** (**USB**), and **fibre channel** (**FC**)
  - Most common is SATA
- Because NVM much faster than HDD, new fast interface for NVM called **NVM express** (**NVMe**), connecting directly to PCI bus

- Data transfers on a bus carried out by special electronic processors called **controllers** (or **host-bus adapters**, **HBAs**)
  - Host controller on the computer end of the bus, device controller on device end
  - Computer places command on host controller, using memory-mapped I/O ports
    - Host controller sends messages to device controller
    - Data transferred via DMA between device and computer DRAM

HDD Scheduling

- The OS is responsible for using hardware efficiently
  - for the disk drives, this means having a fast access time and disk bandwidth
- 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.)

- There are many sources of disk I/O requests
  - OS
  - System processes
  - Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device

- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms only make sense when a queue exists
- In the past, OS responsible for queue management, disk drive head scheduling
  - Now, built into the storage device controllers
  - Just provide LBAs, handle sorting of requests
    - Some of the algorithms they use described next

Disk Scheduling (Cont.)

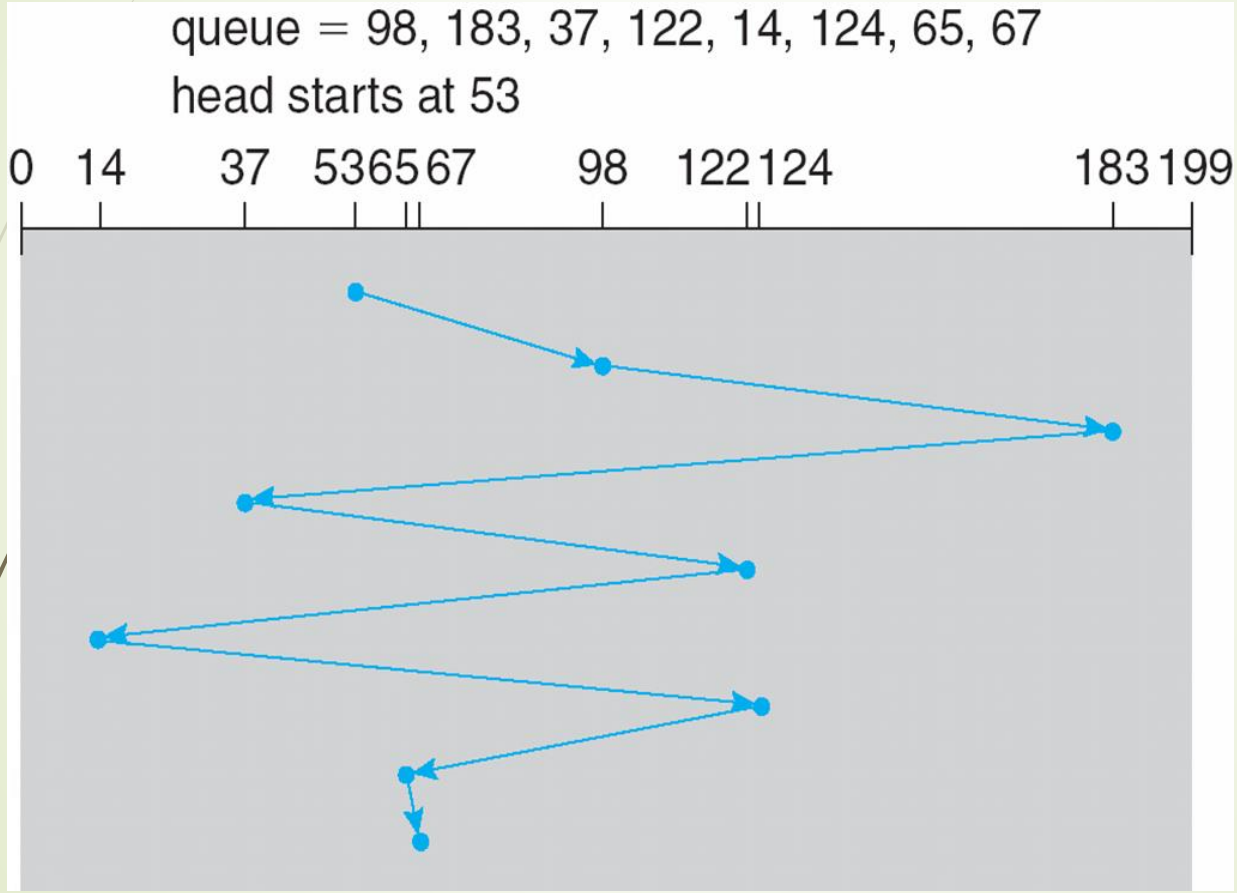
- Drive controllers have small buffers and can manage a queue of I/O requests (of varying sizes)
- Several algorithms exist to schedule the servicing of disk I/O requests
  - The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)

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

Head pointer 53

FCFS

It shows total head movement of 640 cylinders

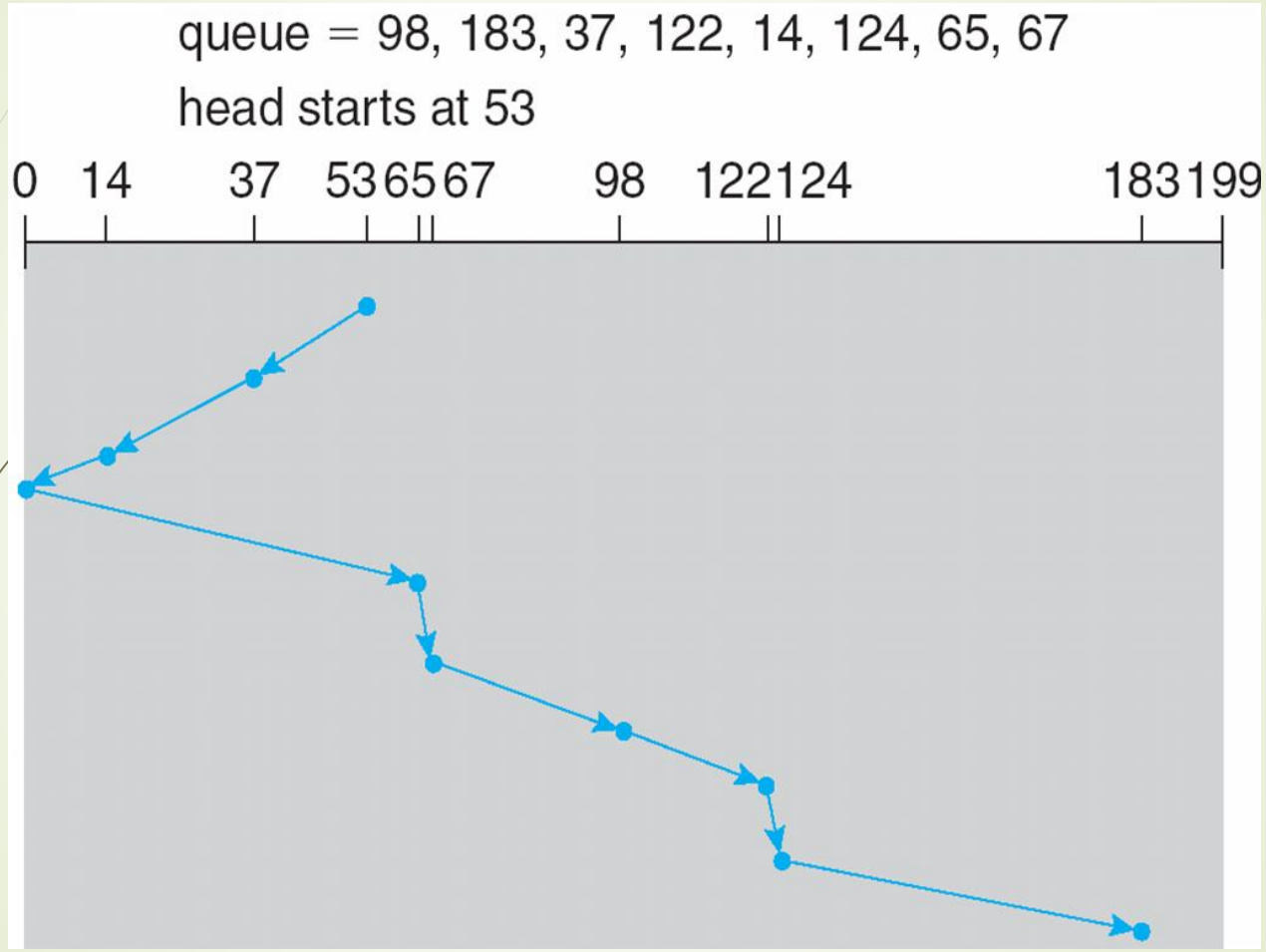


[Source: Operating Systems Concepts, 10th ed.]

SCAN

- **SCAN algorithm** Sometimes called the **elevator algorithm**
  - 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
  - The example shows total head movement of 208 cylinders
- But note that if requests are uniformly dense, largest density at other end of disk which wait the longest

SCAN (Cont.)

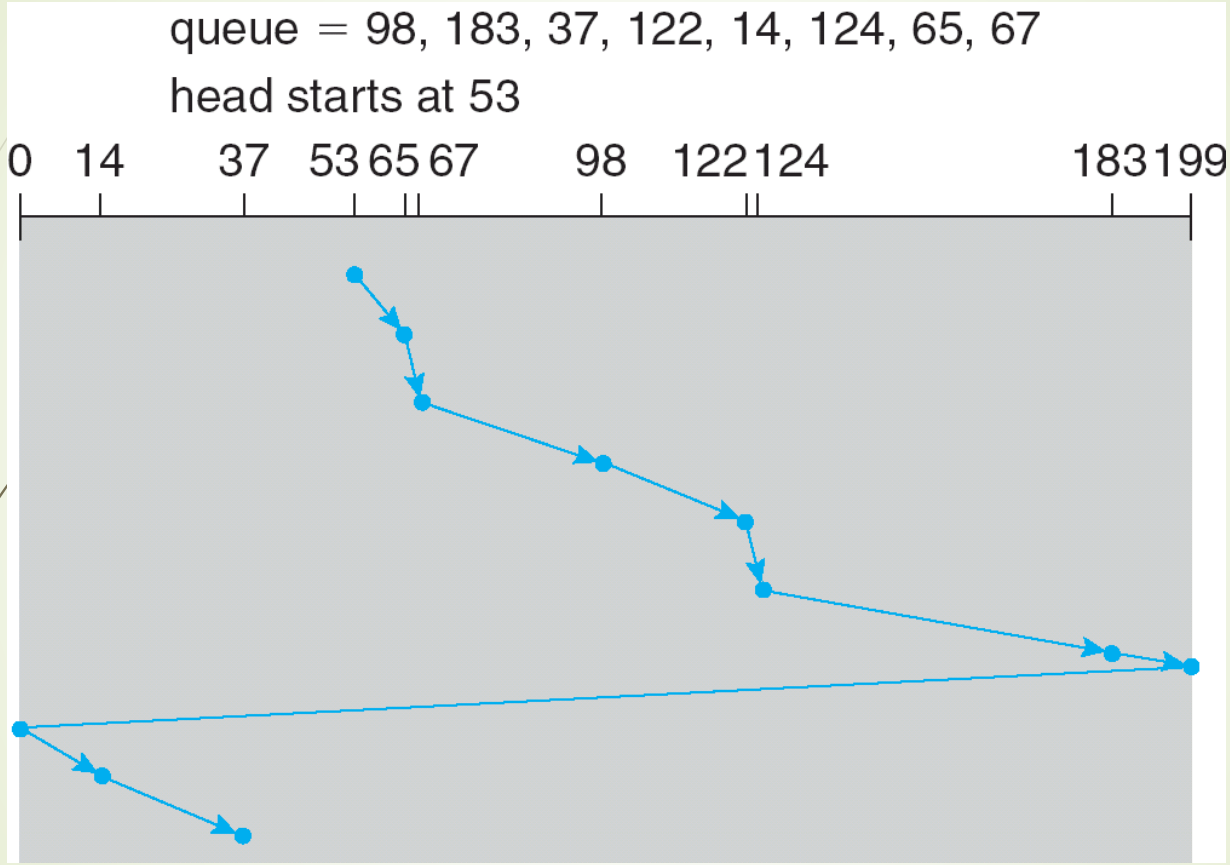


[Source: Operating Systems Concepts, 10th ed.]

C-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
- Provides a more uniform wait time than SCAN
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

C-SCAN (Cont.)



[Source: Operating Systems Concepts, 10th ed.]

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
  - Less starvation, but still possible
- To avoid starvation Linux implements **deadline** scheduler
  - Maintains separate read and write queues, gives read priority
    - Because processes more likely to block on read than write

- Implements four queues: 2 x read and 2 x write
  - 1 read and 1 write queue sorted in LBA order, essentially implementing C-SCAN
  - 1 read and 1 write queue sorted in FCFS order
  - All I/O requests sent in batch sorted in that queue's order
  - After each batch, checks if any requests in FCFS older than configured age (default 500ms)
    - If so, LBA queue containing that request is selected for next batch of I/O
- In RHEL 7 also **NOOP** and **completely fair queueing** scheduler (**CFQ**) also available, defaults vary by storage device

NVM Scheduling

- No disk heads or rotational latency but still room for optimization
- In RHEL 7 **NOOP** (no scheduling) is used but adjacent LBA requests are combined
  - NVM best at random I/O, HDD at sequential
  - Throughput can be similar
  - **Input/Output operations per second** (**IOPS**) much higher with NVM (hundreds of thousands vs hundreds)
  - But **write amplification** (one write, causing garbage collection and many read/writes) can decrease the performance advantage

Error Detection and Correction

- **Error detection** determines if a problem has occurred (for example a bit flipping)
  - If detected, can halt the operation
  - Detection frequently done via parity bit
- Parity is one form of **checksum** – uses modular arithmetic to compute, store, compare values of fixed-length words
  - Another error-detection method common in networking is **cyclic redundancy check** (**CRC**) which uses hash function to detect multiple-bit errors
- **Error-correction code** (**ECC**) not only detects, but can correct some errors
  - Soft errors correctable, hard errors detected but not corrected
- Fundamental aspect of many parts of computing (memory, networking, storage)

Storage Device Management

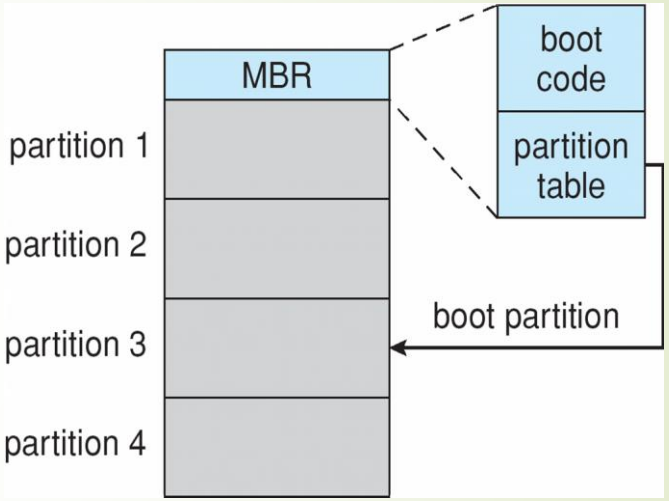
- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, plus data, plus error correction code (**ECC**)
  - Usually 512 bytes of data but can be selectable
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk
  - **Partition** the disk into one or more groups of cylinders, each treated as a logical disk
  - **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

Storage Device Management (cont.)

- **Root partition** contains the OS, other partitions can hold other Oses, other file systems, or be raw
  - **Mounted** at boot time
  - Other partitions can mount automatically or manually
- At mount time, file system consistency checked
  - Is all metadata correct?
    - If not, fix it, try again
    - If yes, add to mount table, allow access
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
  - Or a boot management program for multi-OS booting

Storage Device Management (Cont.)

- Raw disk access for apps that want to do their own block management, keep OS out of the way (databases for example)
- Boot block initializes system
  - The bootstrap is stored in ROM, firmware
  - **Bootstrap loader** program stored in boot blocks of boot partition
- Methods such as **sector sparing** used to handle bad blocks

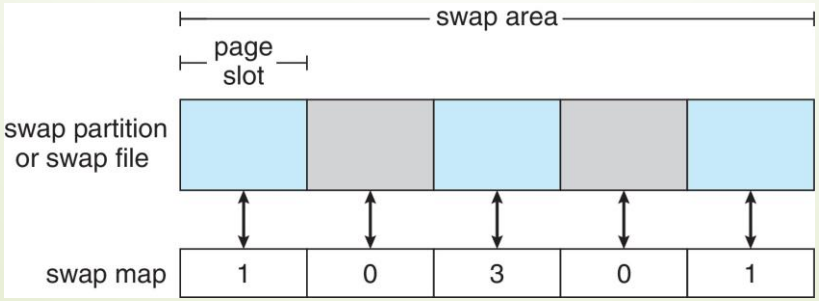


Booting from secondary storage in Windows

[Source: Operating Systems Concepts, 10th ed.]

Swap-Space Management

- Used for moving entire processes (swapping), or pages (paging), from DRAM to secondary storage when DRAM not large enough for all processes
- OS provides **swap space management**
  - Secondary storage slower than DRAM, so important to optimize performance
  - Usually multiple swap spaces possible – decreasing I/O load on any given device
  - Best to have dedicated devices
  - Can be in raw partition or a file within a file system (for convenience of adding)
  - Data structures for swapping on Linux systems:



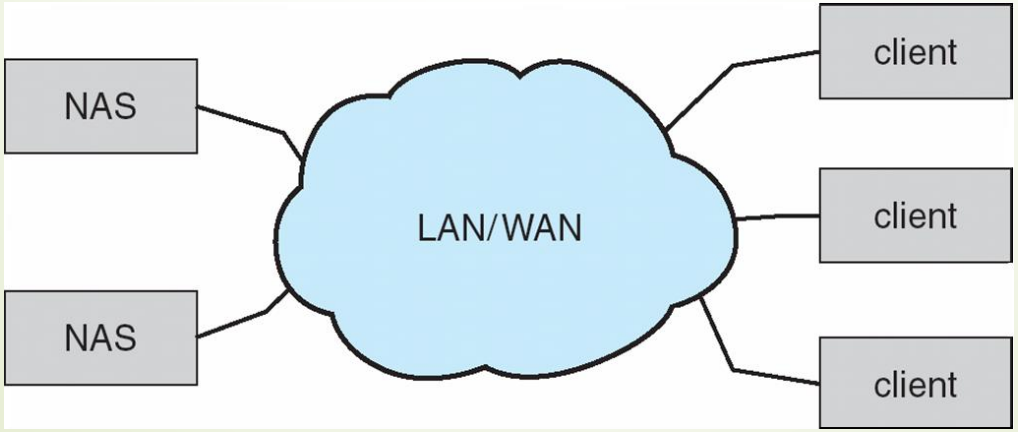
[Source: Operating Systems Concepts, 10th ed.]

Storage Attachment

- Computers access storage in three ways
  - host-attached
  - network-attached
  - cloud
- Host attached access through local I/O ports, using one of several technologies
  - To attach many devices, use storage busses such as USB, firewire, thunderbolt
  - High-end systems use **fibre channel** (**FC**)
    - High-speed serial architecture using fibre or copper cables
    - Multiple hosts and storage devices can connect to the FC fabric

Network-Attached Storage

- Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)
  - Remotely attaching to file systems
  - NFS and CIFS are common protocols
  - Implemented via remote procedure calls (RPCs) between host and storage over typically TCP or UDP on IP network
- **iSCSI** protocol uses IP network to carry the SCSI protocol
  - Remotely attaching to devices (blocks)



[Source: Operating Systems Concepts, 10th ed.]

Cloud Storage

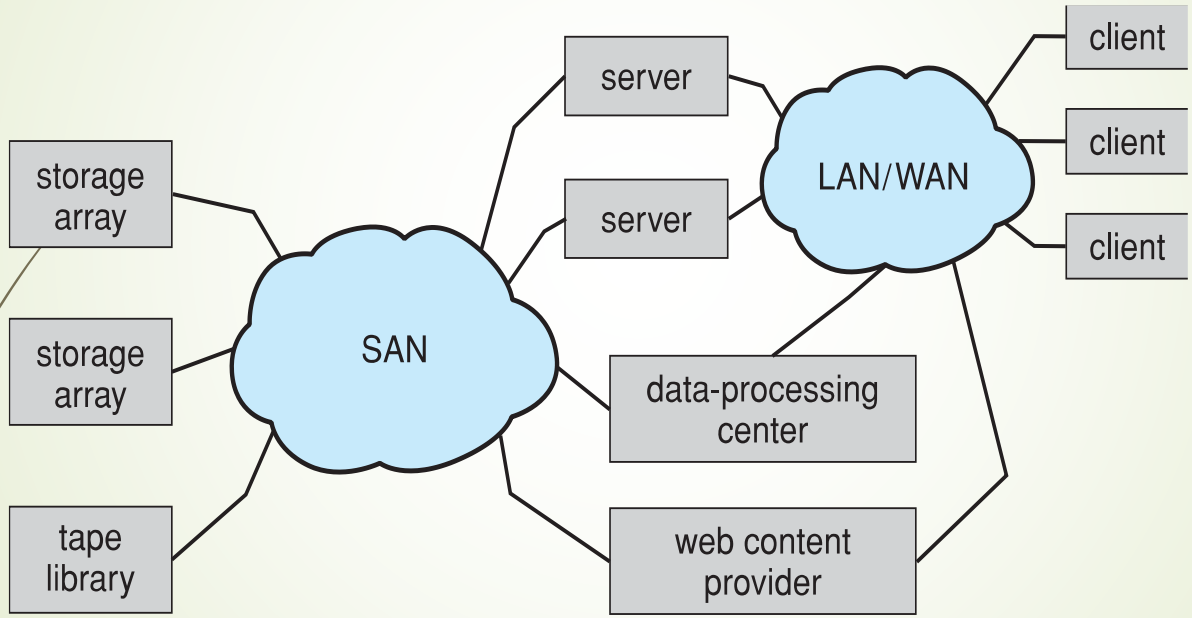
- Similar to NAS, provides access to storage across a network
  - Unlike NAS, accessed over the Internet or a WAN to remote data center
- NAS presented as just another file system, while cloud storage is API based, with programs using the APIs to provide access
  - Examples include Dropbox, Amazon S3, Microsoft OneDrive, Apple iCloud
  - Use APIs because of latency and failure scenarios (NAS protocols wouldn't work well)

Storage Array

- Can just attach disks, or arrays of disks
- Avoids the NAS drawback of using network bandwidth
- Storage Array has controller(s), provides features to attached host(s)
  - Ports to connect hosts to array
  - Memory, controlling software (sometimes NVRAM, etc)
  - A few to thousands of disks
  - RAID, hot spares, hot swap (discussed later)
  - Shared storage -> more efficiency
  - Features found in some file systems
    - Snaphots, clones, thin provisioning, replication, deduplication, etc

Storage Area Network

- Common in large storage environments
- Multiple hosts attached to multiple storage arrays – flexible



[Source: Operating Systems Concepts, 10th ed.]

Storage Area Network (Cont.)

- SAN is one or more storage arrays
  - Connected to one or more Fibre Channel switches or **InfiniBand** (**IB**) network
- Hosts also attach to the switches
- Storage made available via **LUN Masking** from specific arrays to specific servers
- Easy to add or remove storage, add new host and allocate it storage
- Why have separate storage networks and communications networks?
  - Consider iSCSI, FCOE

A Storage Array

[Source: Operating Systems Concepts, 10th ed.]

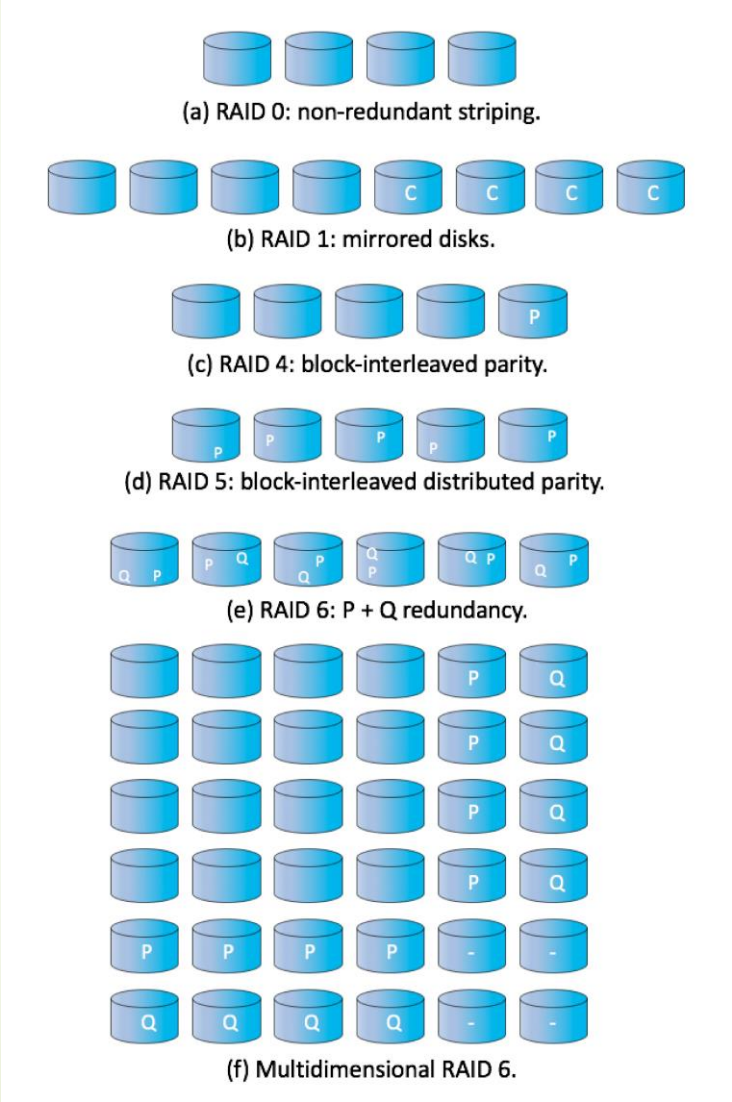
RAID Structure

- **RAID** – **redundant array of inexpensive disks**
  - multiple disk drives provides reliability via **redundancy**
- Increases the **mean time to failure**, by mirroring
  - If mirrored disks fail independently, consider disk with 100,000 hour **mean time to failure** and 10 hour mean time to repair
  - **Mean time to repair –** exposure time when another failure could cause data loss
  - **Mean time to data loss** is $100,000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years!
- Frequently combined with **NVRAM** to improve write performance
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively

RAID (Cont.)

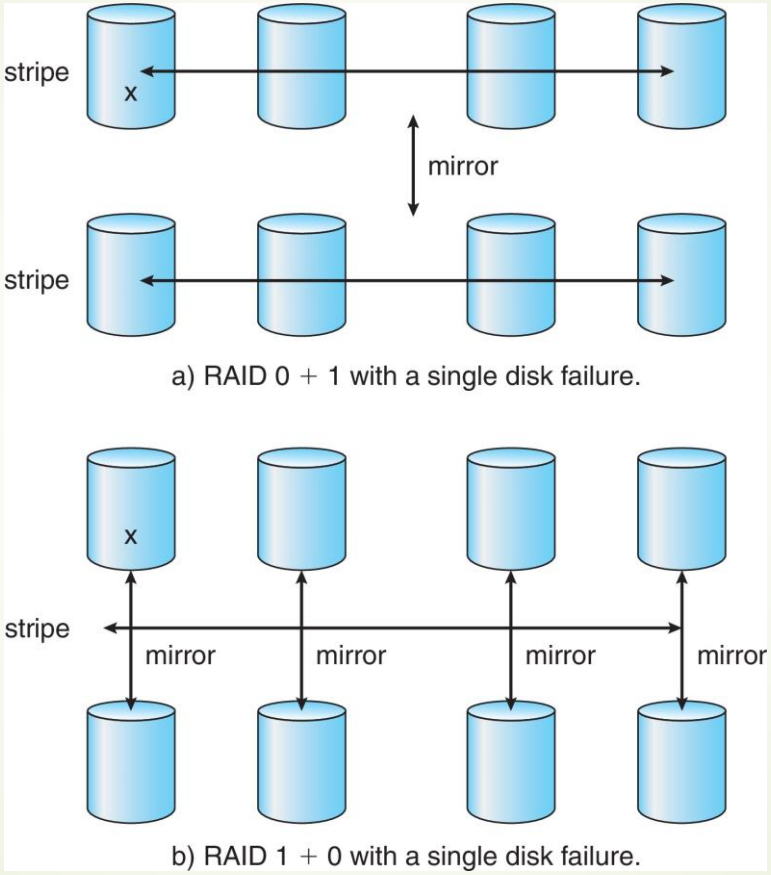
- RAID is arranged into six different levels
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
  - Disk **striping** uses a group of disks as one storage unit
  - **Mirroring** or **shadowing** (**RAID 1**) keeps duplicate of each disk
  - 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



[Source: Operating Systems Concepts, 10th ed.]

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



[Source: Operating Systems Concepts, 10th ed.]

Other Features

- **Snapshot** is a view of file system before a set of changes take place (i.e. at a point in time)
  - More in Ch. 12
- Replication is automatic duplication of writes between separate sites
  - For redundancy and disaster recovery
  - Can be synchronous or asynchronous
- Hot spare disk is unused, automatically used by RAID production if a disk fails to replace the failed disk and rebuild the RAID set if possible
  - Decreases mean time to repair

File Systems

File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- Contents defined by file's creator
  - Many types
    - Consider **text file, source file, executable file**

File Attributes

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

File info Window on Mac OS X

[Source: Operating Systems Concepts, 10th ed.]

File Operations

- File is an **abstract data type**
- 6 basic operations
  - **Create**
  - **Write –** at **write pointer** location
  - **Read –** at **read pointer** location
  - **Reposition within file - seek**
  - **Delete**
  - **Truncate**
- To avoid constant searching of the file entry in the directory,
  - ***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:
  - **Open-file table**: tracks 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 the last process closes it
  - Disk location of the file: cache of data access information
  - Access rights: per-process access mode information

File Types – Filename 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

[Source: Operating Systems Concepts, 10th ed.]

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Who decides:
  - OS
  - Program

Access Methods

- **Sequential Access**

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

- **Direct Access –** file is fixed length **logical records**

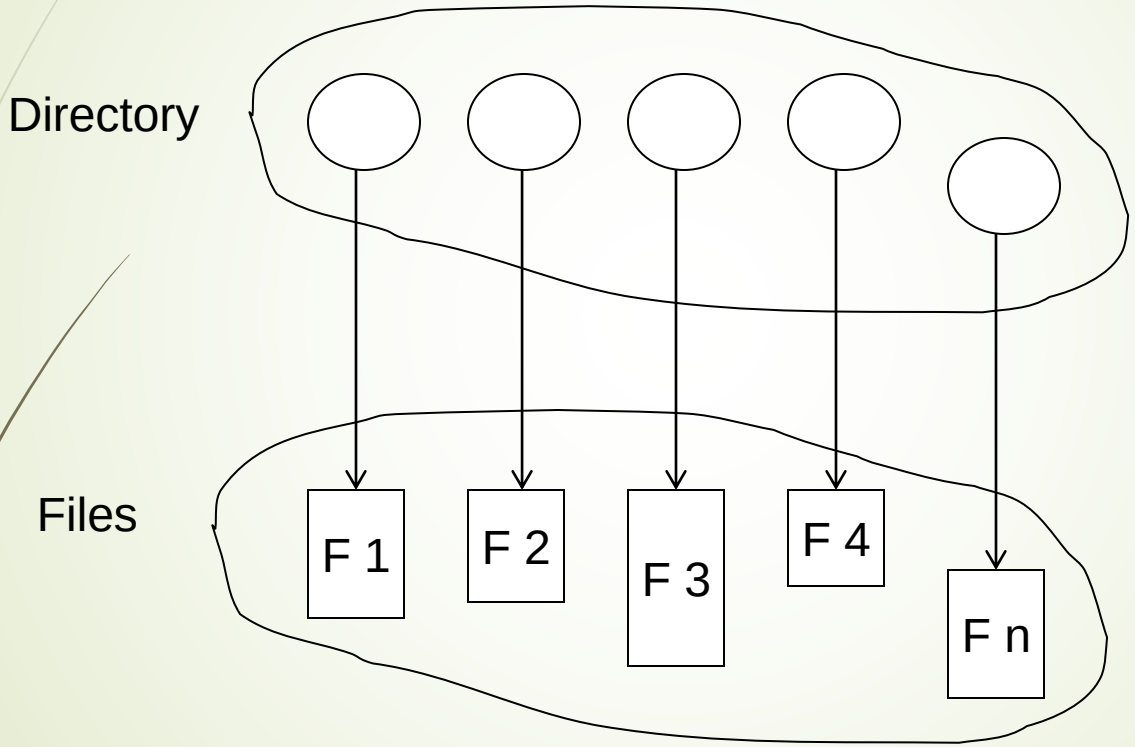
```
read n
write n
position to n
        read next
        write next
rewrite n
```

n = **relative block number**

- Relative block numbers allow OS to decide where file should be placed
  - See **allocation problem** in Ch.14

Directory Structure

- A collection of nodes containing information about all files



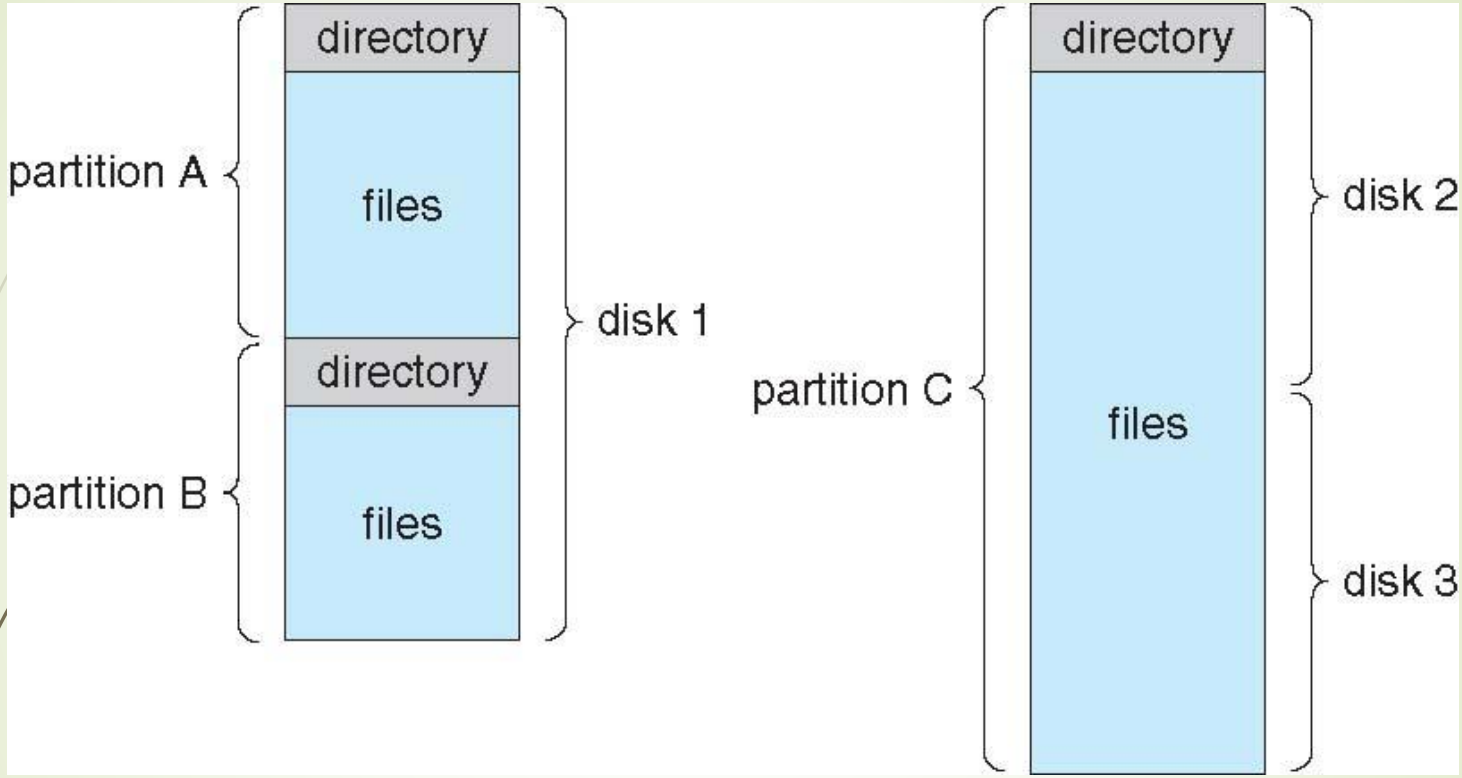
Both the directory structure and the files reside on disk

[Source: Operating Systems Concepts, 10th ed.]

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 OS or computer

A Typical File-system Organization



[Source: Operating Systems Concepts, 10th ed.]

Types of File Systems

- We mostly talk of general-purpose file systems
  - But systems frequently have may file systems, some general- and some special- purpose
- Consider Solaris has
  - tmpfs – memory-based volatile FS for fast, temporary I/O
  - objfs – interface into kernel memory to get kernel symbols for debugging
  - ctfs – contract file system for managing daemons
  - lofs – loopback file system allows one FS to be accessed in place of another
  - procfs – kernel interface to process structures
  - ufs, zfs – general purpose file systems

Operations Performed on Directory

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

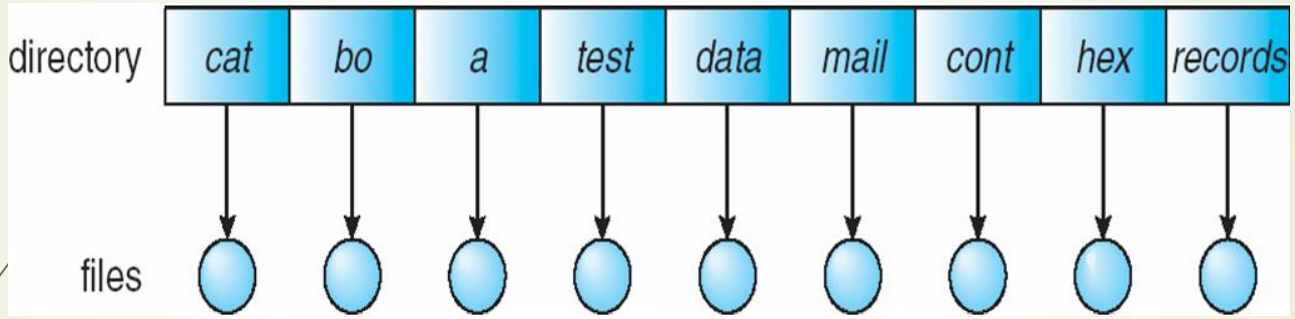
Directory Organization

The directory is organized logically to obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have the 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

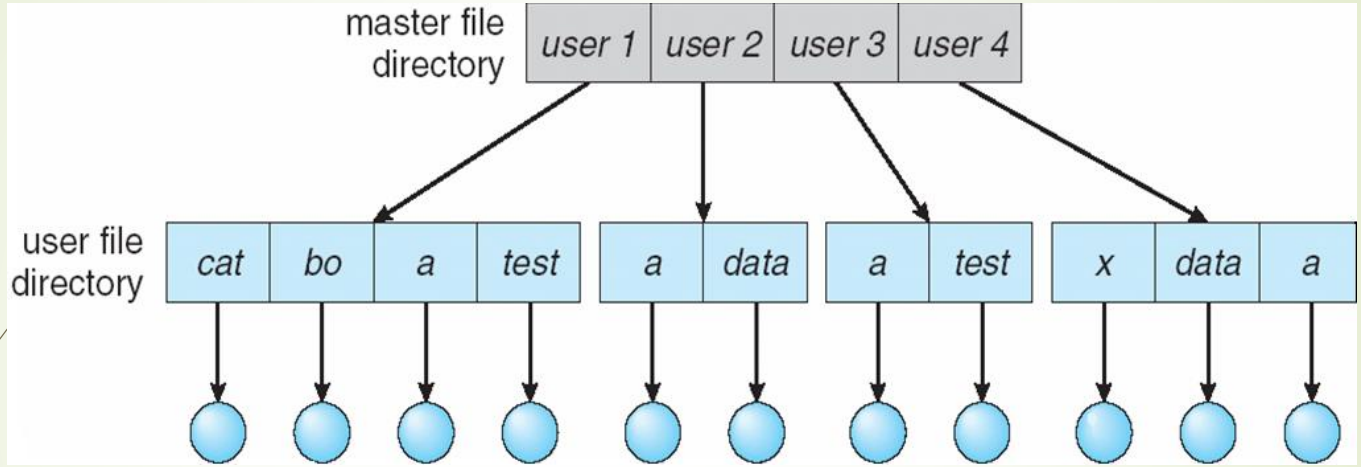
- A single directory for all users



- Naming problem
- Grouping problem

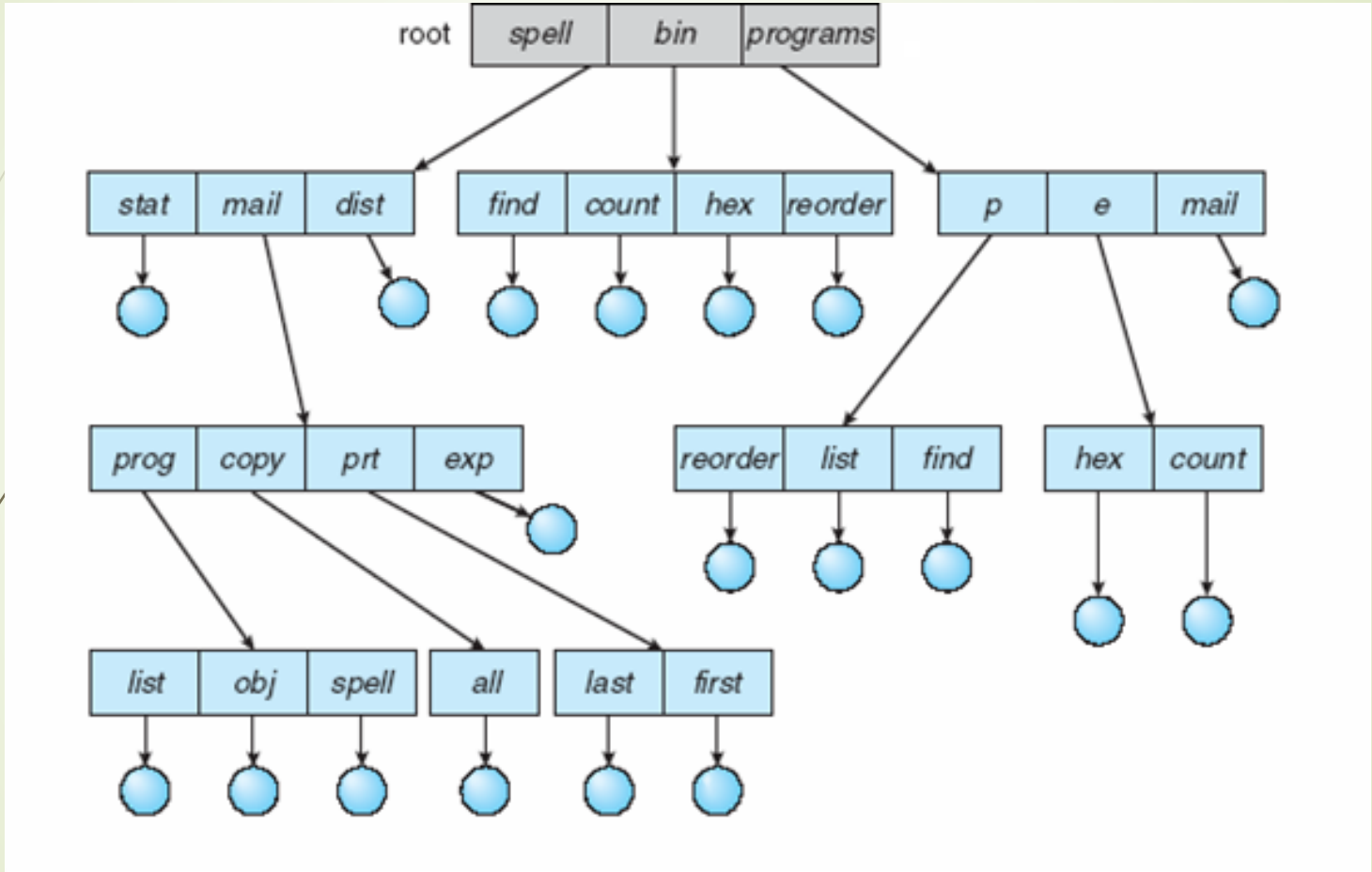
Two-Level Directory

- Separate directory for each user



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

Tree-Structured Directories



[Source: Operating Systems Concepts, 10th ed.]

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

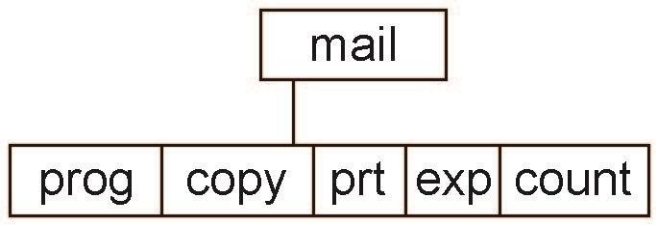
```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

```
mkdir <dir-name>
```

Example: if in current directory **/mail**

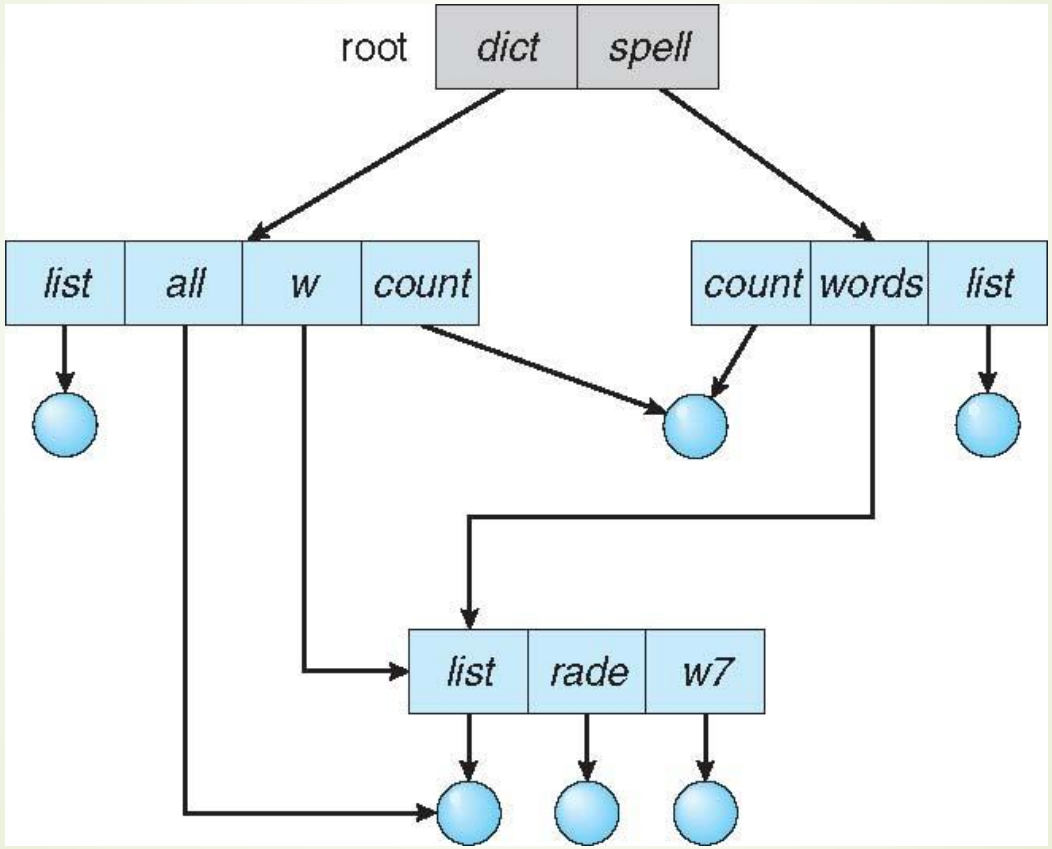
```
mkdir count
```



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

Acyclic-Graph Directories

- Have shared subdirectories and files



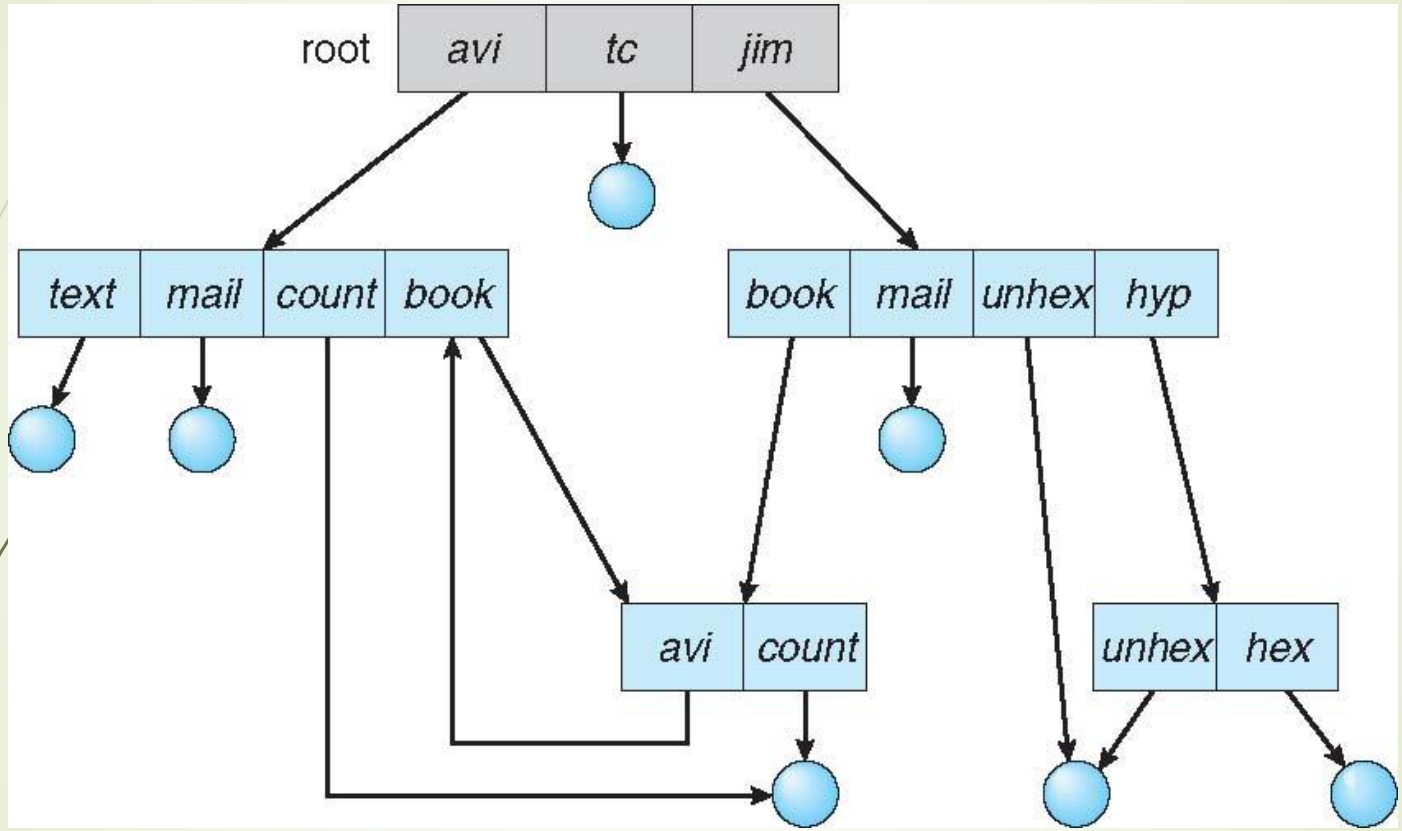
[Source: Operating Systems Concepts, 10th ed.]

Acyclic-Graph Directories (Cont.)

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

  Solutions:
  - Backpointers, so we can delete all pointers
    - Variable size records 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



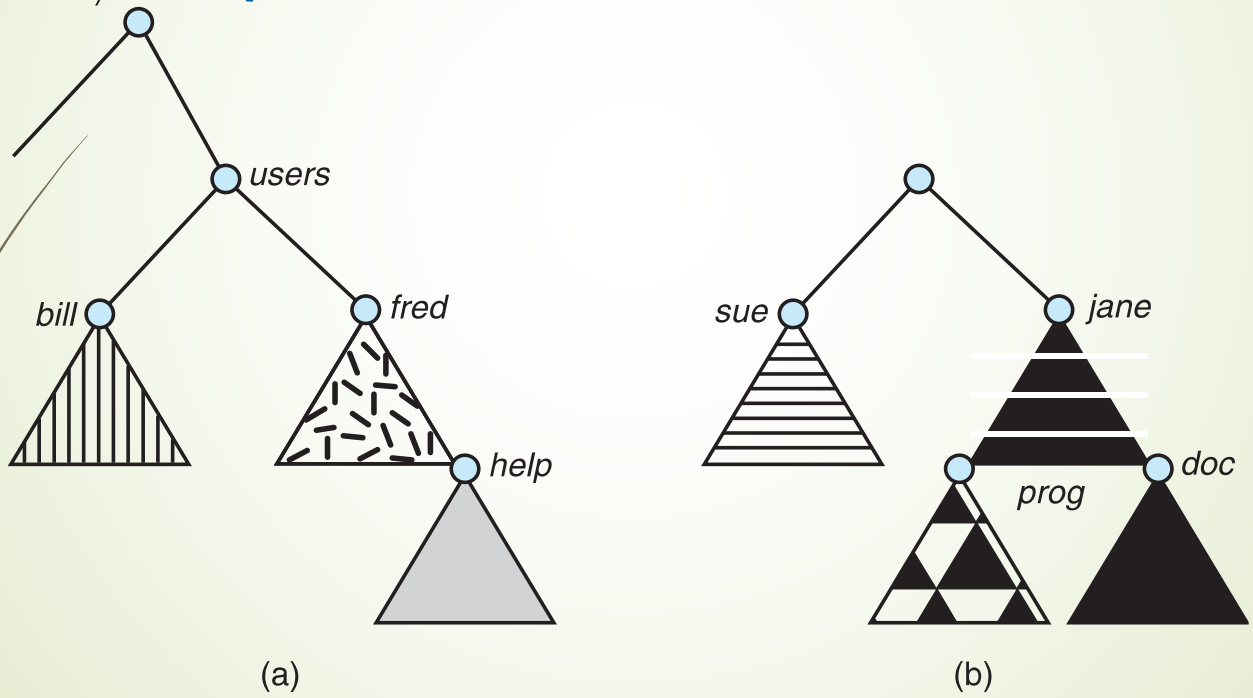
[Source: Operating Systems Concepts, 10th ed.]

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

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**



[Source: Operating Systems Concepts, 10th ed.]

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
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

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 OS file system 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

- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called **metadata**
- 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 v3 include all information in each request, allowing easy recovery but less security

File Sharing – 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)
  - Unix file system (UFS) implements UNIX semantics:
    - 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
  - Andrew File System (AFS) implemented complex remote file sharing semantics - session semantics
    - Writes only visible to sessions starting after the file is closed

Protection

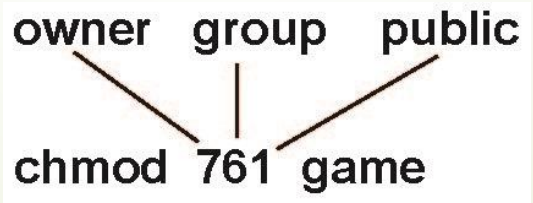
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

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

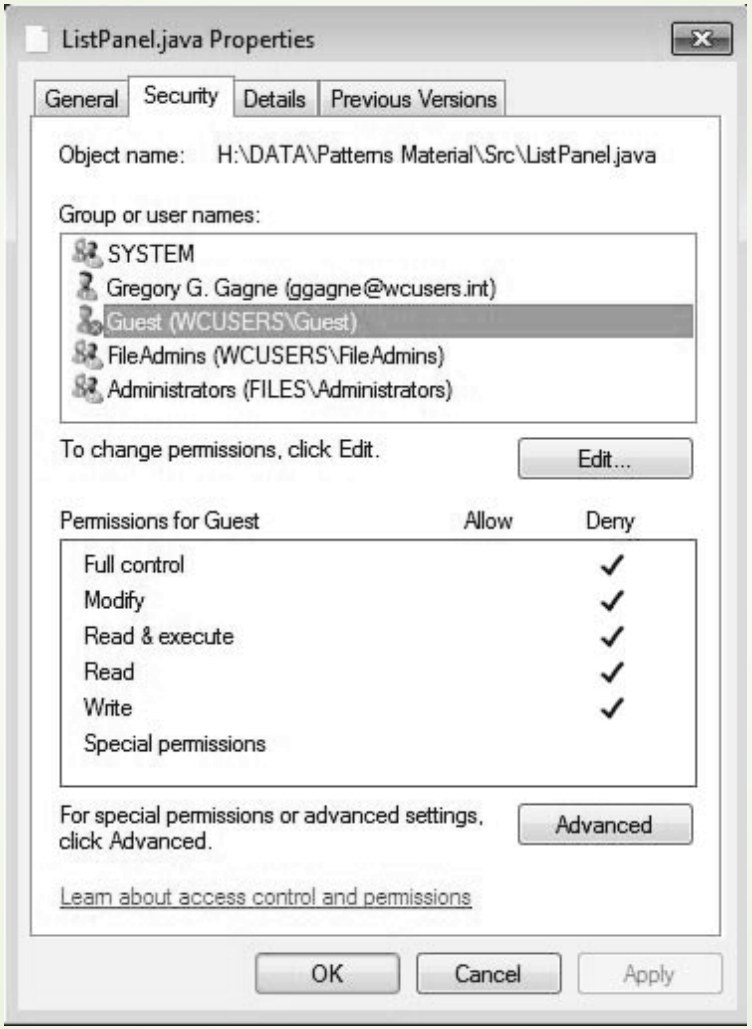
- Ask manager to create a group (unique name), say G, and add some users to the group
- For a particular file (say *game*) or subdirectory, define an appropriate access



Attach a group to a file

```
chgrp      G      game
```

Windows Access-Control List Management



[Source: Operating Systems Concepts, 10th ed.]

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/

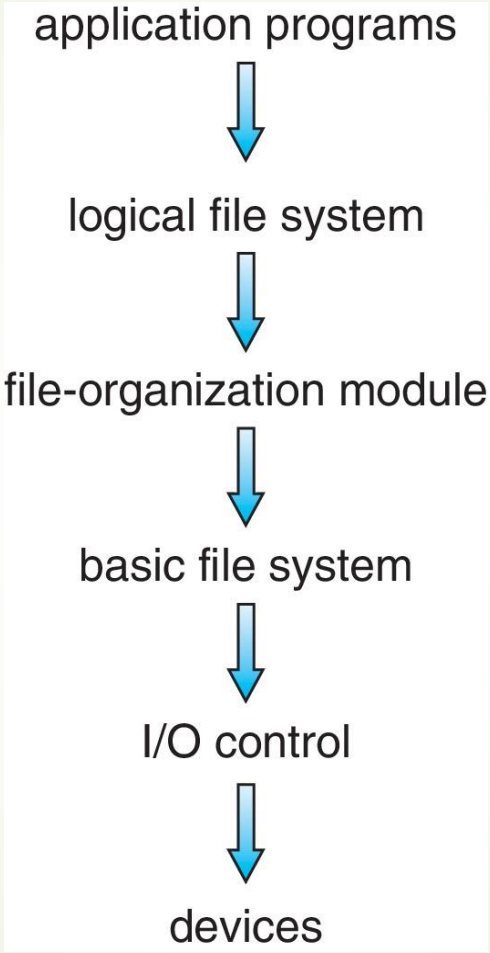
[Source: Operating Systems Concepts, 10th ed.]

File System Structures

File-System Structure

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

Layered File System



[Source: Operating Systems Concepts, 10th ed.]

File System Layers

- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like "read drive1, cylinder 72, track 2, sector 10, into memory location 1060" outputs low-level hardware specific commands to hardware controller (Ch.12)
- **Basic file system** given command like "retrieve block 123" translates to device driver
  - Also manages memory buffers and caches (allocation, freeing, replacement)
    - Buffers hold data in transit
    - Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
  - Translates logical block # to physical block #
  - Manages free space, disk allocation

File System Layers (Cont.)

- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
  - Directory management
  - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance
- Logical layers can be implemented by any coding method according to OS designer

File System Layers (Cont.)

- Many file systems, sometimes many are supported within an OS
  - Each with its own format
    - CD-ROM is ISO 9660
    - Unix has **UFS**, FFS
    - Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray
    - Linux has more than 130 types, with **extended file system** ext3 and ext4 leading; plus distributed file systems, etc.
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE

File-System Operations

- We have system calls at the API level, but how do we implement their functions?
  - On-disk and in-memory structures
- On-disk structures
  - Boot control block (per volume)
  - Volume control block (per volume)
  - Directory structure (per file system)
  - Per-file File-control block (FCB)

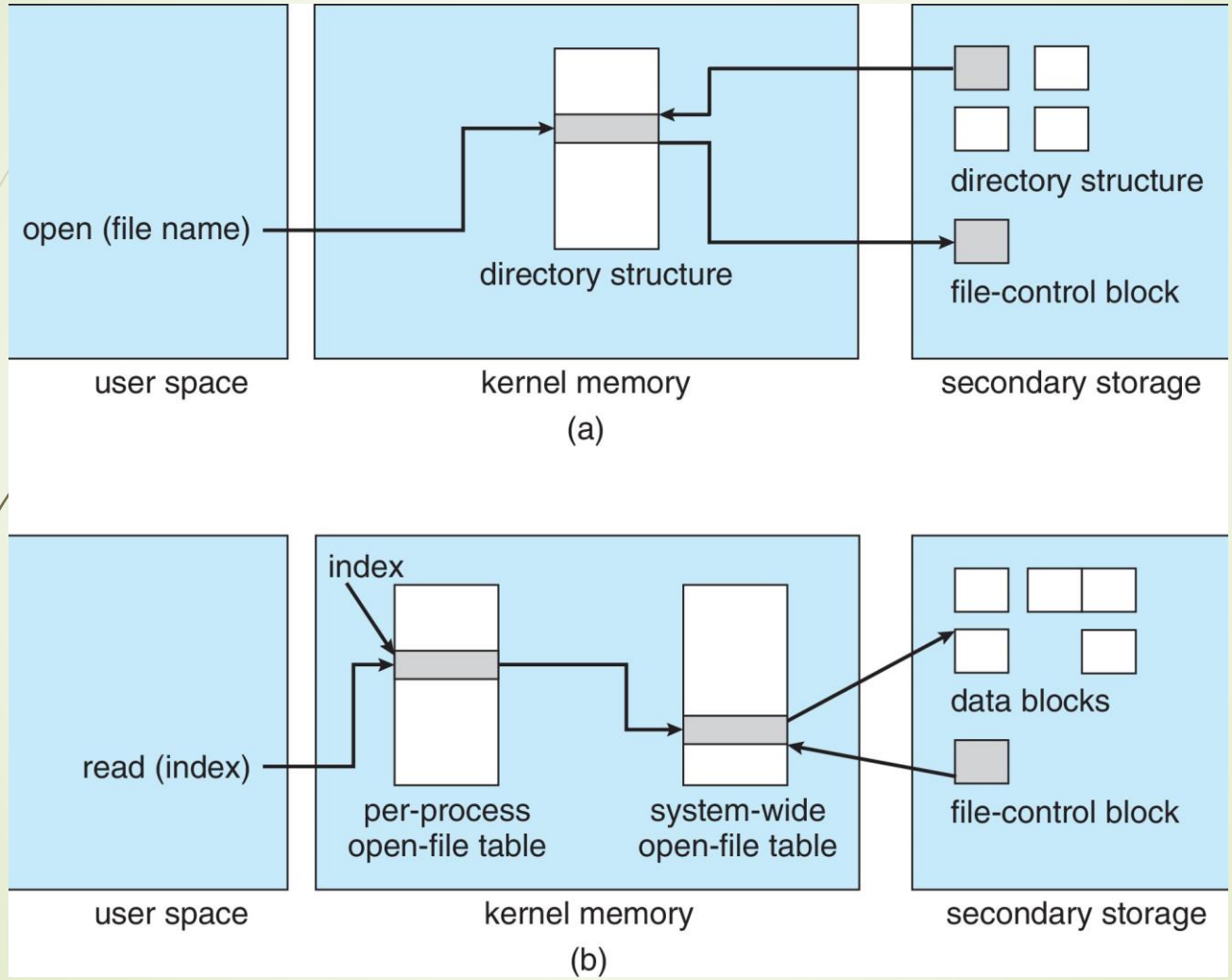
On-disk Structures

- **Boot control block**: info needed to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
  - UNIX boot block, NTFS partition boot sector
- **Volume control block (superblock, master file table)** contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
  - UNIX superblock, NTFS master file table
- Directory structure organizes the files
  - UNIX filenames and inode numbers, NTFS master file table
- Per-file **File Control Block** (**FCB**) contains many details about the file
  - typically inode number, permissions, size, dates
  - NTFS stores info in master file table using relational DB structures

In-Memory File System Structures

- **Mount table** storing file system mounts, mount points, file system types
- **system-wide open-file table** contains a copy of the FCB of each file and other info
- **per-process open-file table** contains pointers to appropriate entries in system-wide open-file table as well as other info
- Directory-structure cache holds directory information of recently accessed directories
- Plus buffers hold data blocks from secondary storage

In-Memory File System Structures



[Source: Operating Systems Concepts, 10th ed.]

Directory Implementation

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

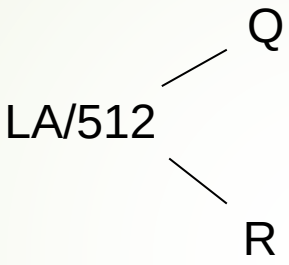
File Allocation Methods

Allocation Methods - Contiguous

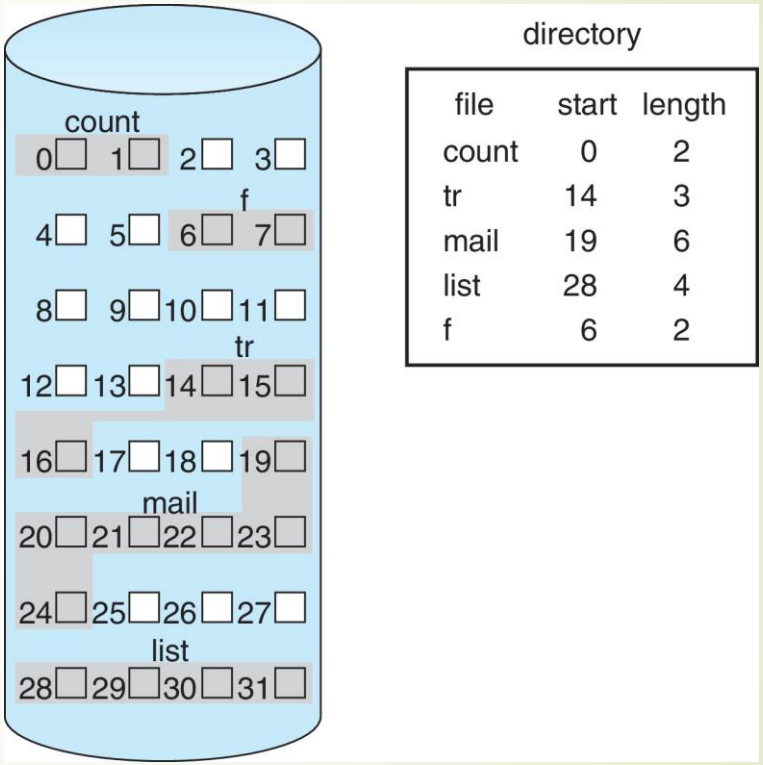
- Allocation method: how disk blocks are allocated for files
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding space for file,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

Contiguous Allocation

- Mapping from logical to physical



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



[Source: Operating Systems Concepts, 10th ed.]

Extent-Based Systems

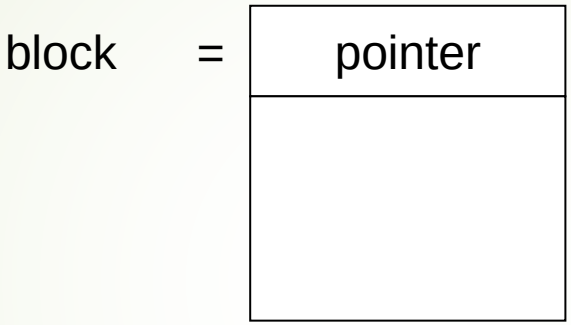
- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is contiguous blocks on disk
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Allocation Methods - Linked

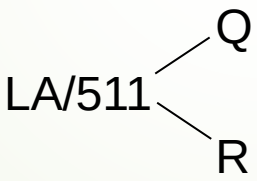
- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - Each block contains pointer to next block
  - Free space management system called when new block needed
- Pros and cons
  - No compaction, external fragmentation
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks
    - Improve efficiency by clustering blocks into groups but increases internal fragmentation

Linked Allocation

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



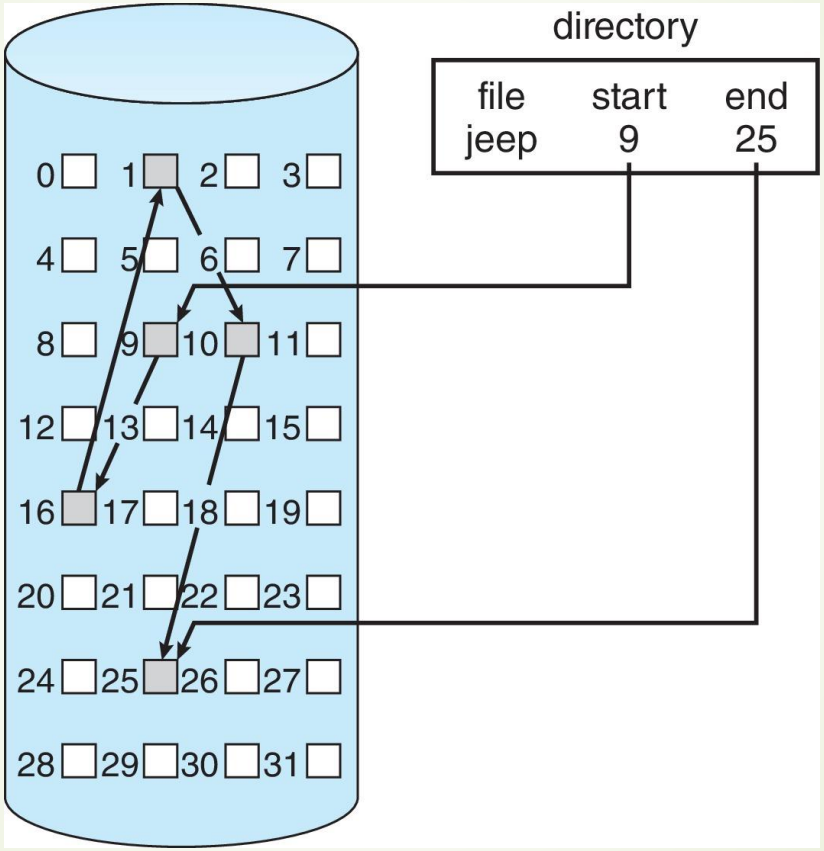
- Mapping



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

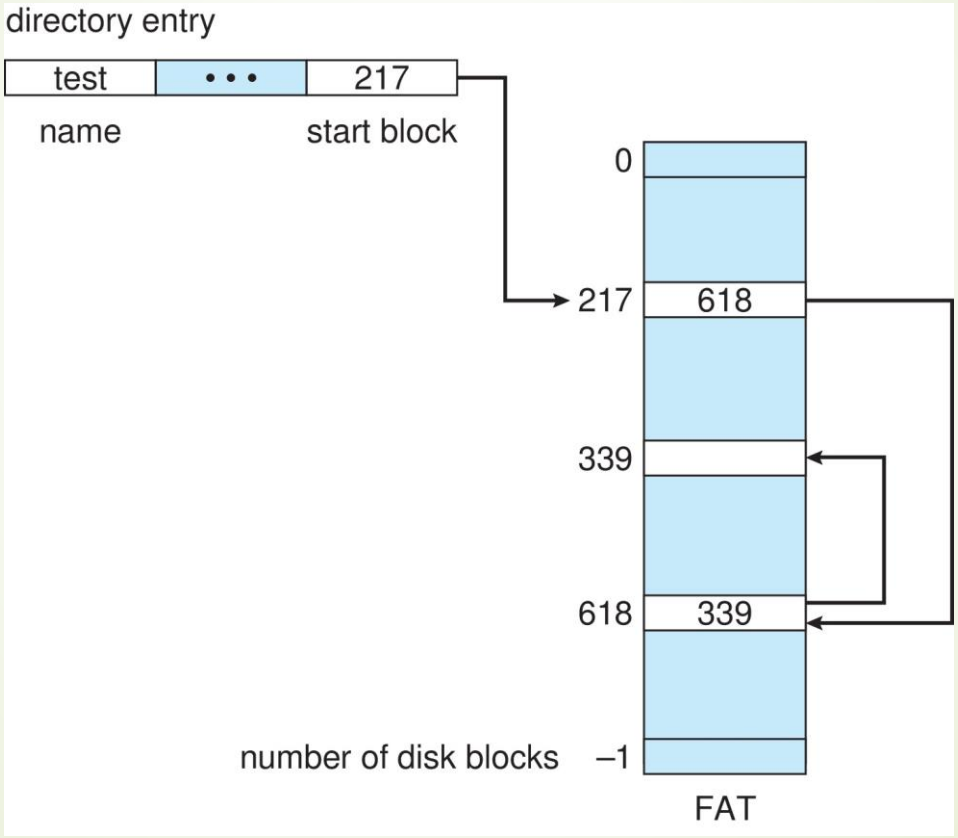
Displacement into block = R + 1

Linked Allocation



[Source: Operating Systems Concepts, 10th ed.]

File-Allocation Table

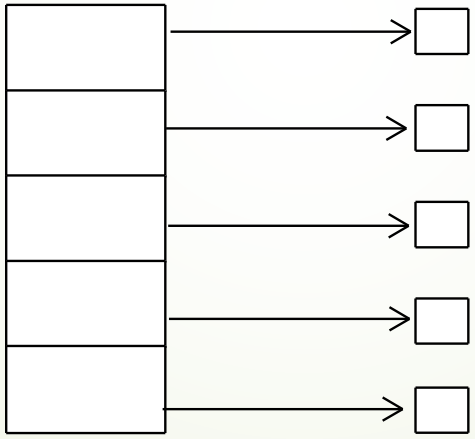


[Source: Operating Systems Concepts, 10th ed.]

Allocation Methods - Indexed

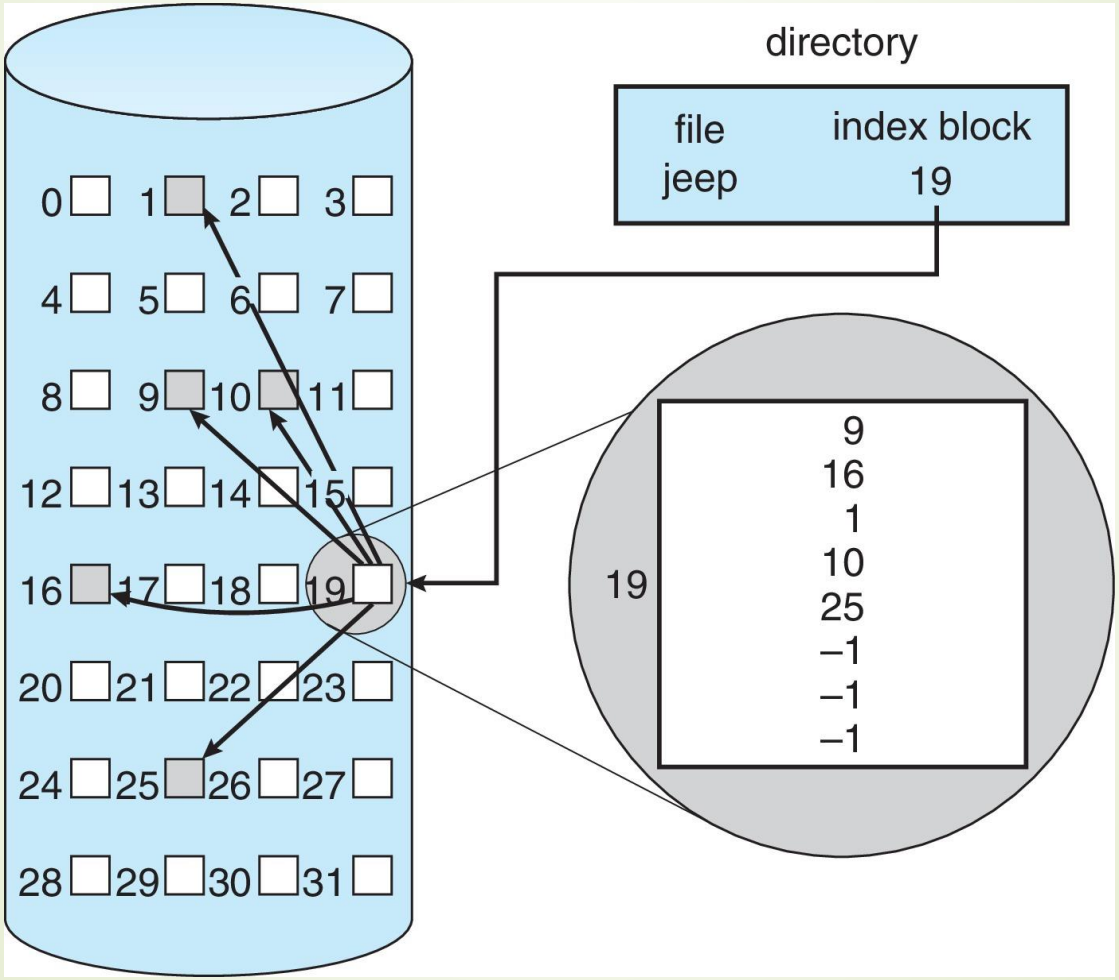
- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks

- Logical view



[Source: Operating Systems Concepts, 10th ed.]

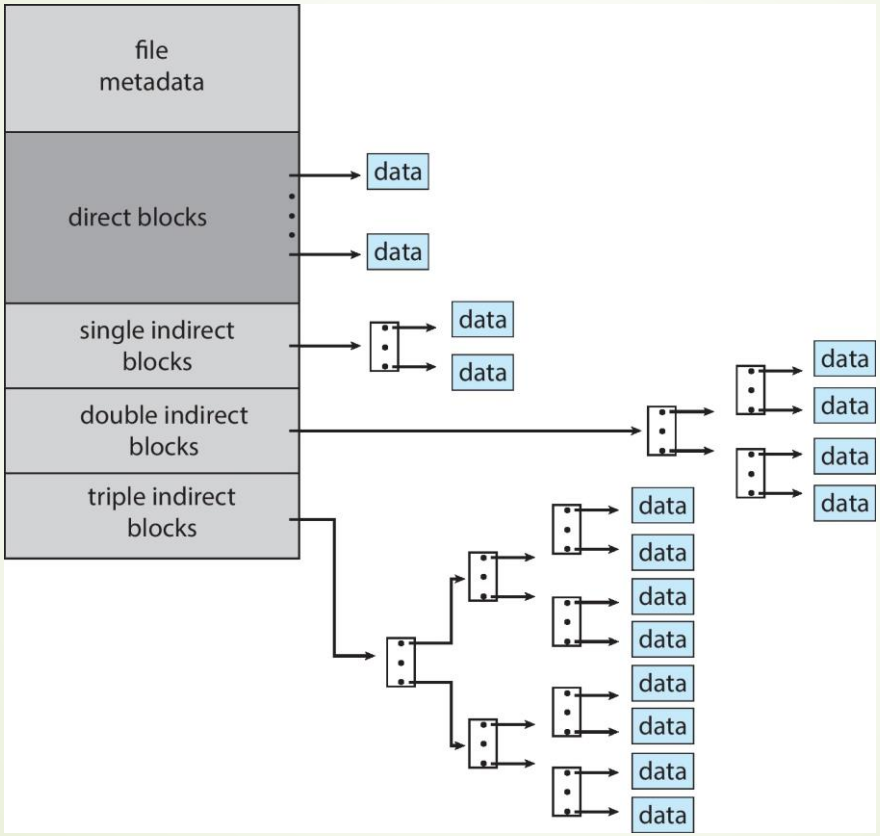
Example of Indexed Allocation



[Source: Operating Systems Concepts, 10th ed.]

Combined Scheme: UNIX UFS

4K bytes per block, 32-bit addresses



More index blocks than can be addressed with 32-bit file pointer

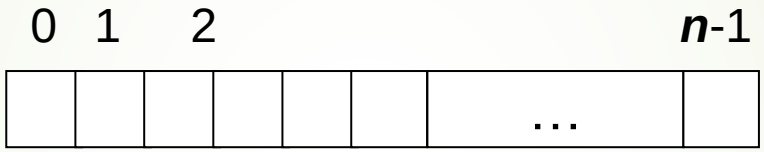
[Source: Operating Systems Concepts, 10th ed.]

Performance

- Best method depends on file access type
  - Contiguous great for sequential and random
  - Linked good for sequential, not random
  - Declare access type at creation -> select either contiguous or linked
- Indexed more complex
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead
- For NVM, no disk head so different optimizations needed
  - Using old algorithm takes many CPU cycles trying to avoid non-existent head movement
  - With NVM goal is to reduce CPU cycles and overall path needed for I/O

Free-Space Management

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



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

Block number calculation

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

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

Free-Space Management (Cont.)

- Bit map requires extra space
  - Example:

    block size = 4KB = 2^{12} bytes

    disk size = 2^{40} bytes (1 terabyte)

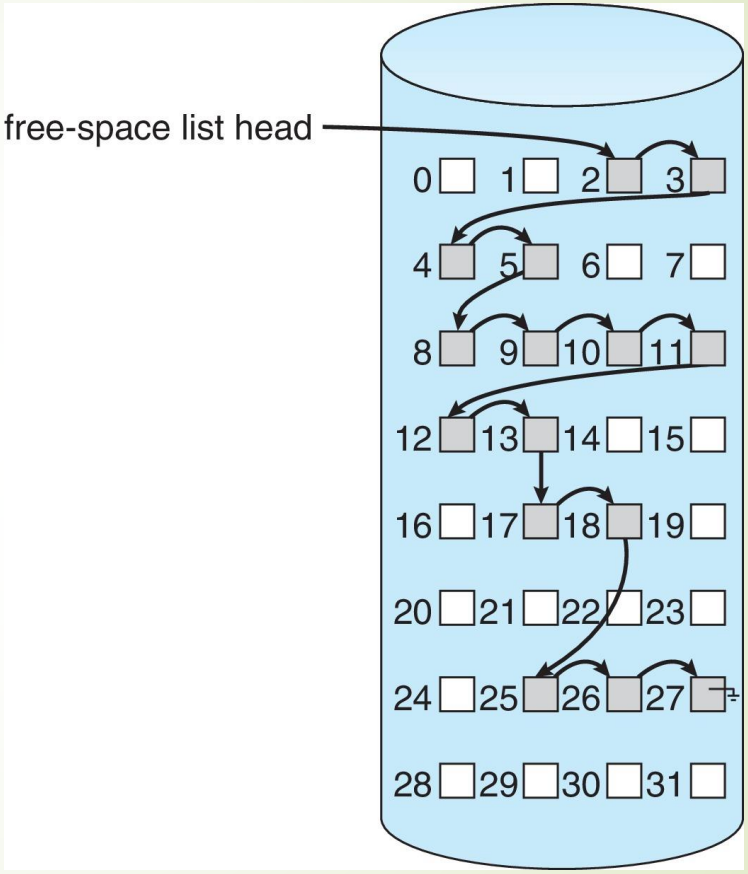
    n = $2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

    if clusters of 4 blocks -> 8MB of memory

- Easy to get contiguous files

Linked Free Space List on Disk

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)



[Source: Operating Systems Concepts, 10th ed.]

Free-Space Management (Cont.)

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

TRIMing Unused Blocks

- HDDS overwrite in place so need only free list
- Blocks not treated specially when freed
  - Keeps its data but without any file pointers to it, until overwritten
- Storage devices not allowing overwrite (like NVM) suffer badly with same algorithm
  - Must be erased before written, erases made in large chunks (blocks, composed of pages) and are slow
  - TRIM is a newer mechanism for the file system to inform the NVM storage device that a page is free
    - Can be garbage collected or if block is free, now block can be erased

Efficiency and Performance

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

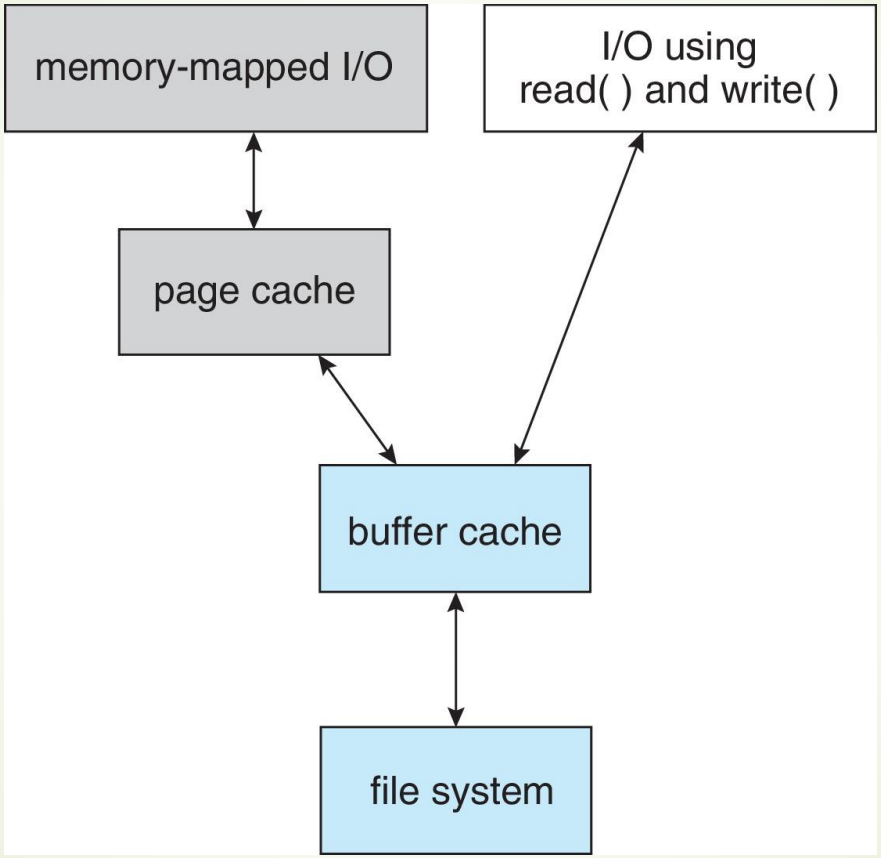
Efficiency and Performance (Cont.)

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

Page Cache

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

I/O Without a Unified Buffer Cache

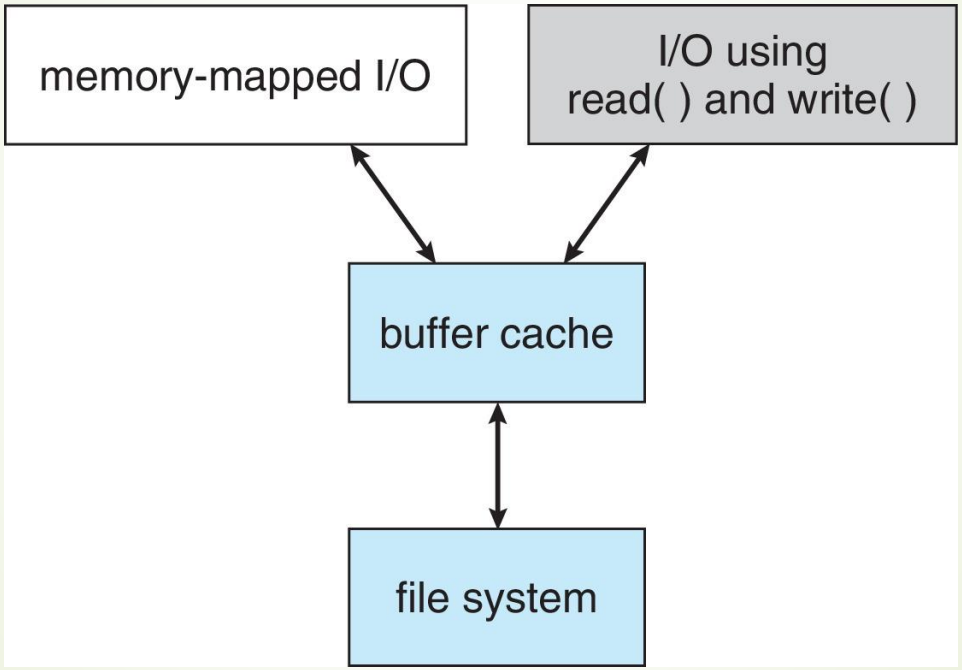


[Source: Operating Systems Concepts, 10th ed.]

Unified Buffer Cache

- A **unified buffer cache** uses the same page cache to cache both memory-mapped pages and ordinary file system I/O to avoid **double caching**
- But which caches get priority, and what replacement algorithms to use?

I/O Using a Unified Buffer Cache



[Source: Operating Systems Concepts, 10th ed.]

Recovery

- **Consistency checking** – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
  - Can be slow and sometimes fails
- 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 metadata update to the file system as a **transaction**
- All transactions are written to a log
  - A transaction is considered committed once it is written to the log (sequentially)
  - Sometimes to a separate device or section of disk
  - However, the file system may not yet be updated
- The transactions in the log are asynchronously written to the file system structures
  - When the file system structures are modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed
- Faster recovery from crash, removes chance of inconsistency of metadata

Example: The Apple File System

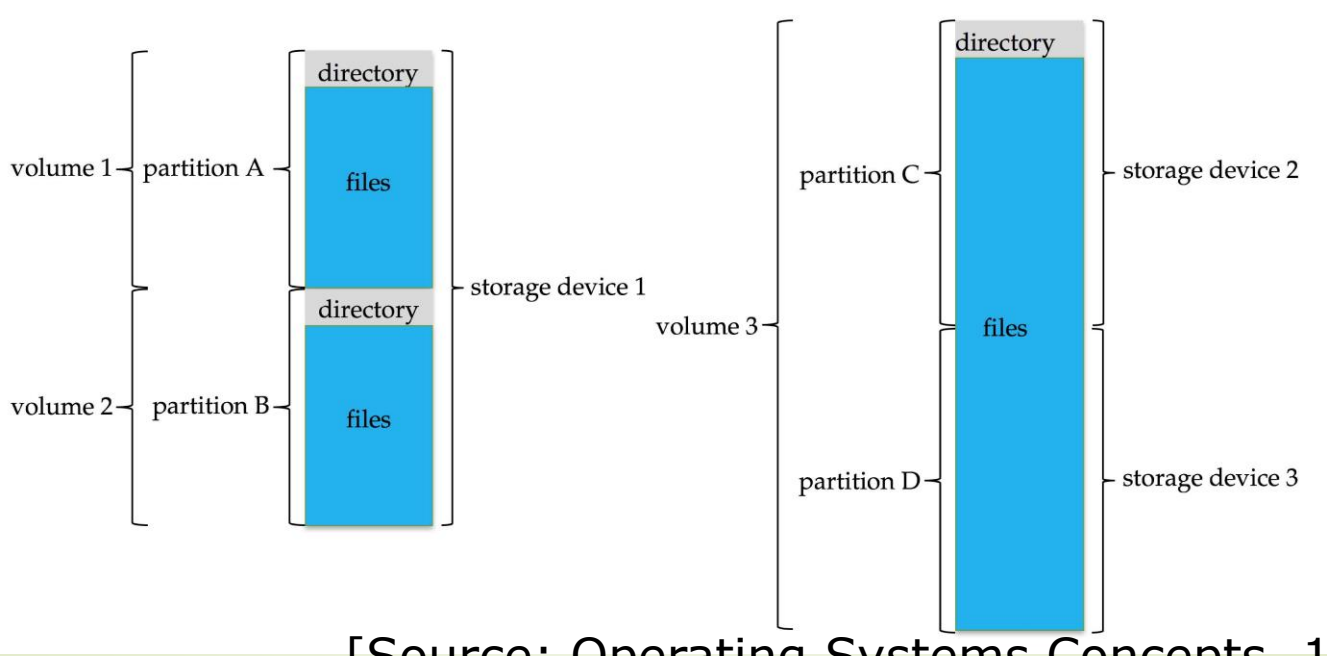
- Apple released a new file system in 2017 called APFS to replace its 30-year-old HFS+
- The goal is to run on all current Apple devices
  - From Apple Watch through the iphone to the Mac computers
  - watchOS, iOS, tvOS, macOS
- Features include
  - 64-bit pointers, clones for files and directories, snapshots, copy-on-write design, encryption
  - Space sharing: storage is available as one or more large free spaces (containers) from which file systems can draw allocations
  - Fast directory sizing: provides quick used space calculation and updating
  - Atomic safe-save primitives: perform renames of files, bundles of files, and directories as single atomic operations

File System Internals

File System

- General-purpose computers can have multiple storage devices
  - Devices can be sliced into partitions, which hold volumes
  - Volumes can span multiple partitions
  - Each volume usually formatted into a file system
  - # of file systems varies, typically dozens available to choose from

Typical storage device organization:



[Source: Operating Systems Concepts, 10th ed.]

Example Mount Points and File Systems - Solaris

/	ufs
/devices	devfs
/dev	dev
/system/contract	ctfs
/proc	proc
/etc/mnttab	mntfs
/etc/svc/volatile	tmpfs
/system/object	objfs
/lib/libc.so.1	lofs
/dev/fd	fd
/var	ufs
/tmp	tmpfs
/var/run	tmpfs
/opt	ufs
/zpbge	zfs
/zpbge/backup	zfs
/export/home	zfs
/var/mail	zfs
/var/spool/mqueue	zfs
/zpbg	zfs
/zpbg/zones	zfs

Partitions and Mounting

- Partition can be a volume containing a file system ("cooked") or **raw** – just a sequence of blocks with no file system
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
  - Or a boot management program for multi-OS booting
- **Root partition** contains the OS, other partitions can hold other OSes, other file systems, or be raw
  - Mounted at boot time
  - Other partitions can mount automatically or manually on **mount points** – location at which they can be accessed
- At mount time, file system consistency checked
  - Is all metadata correct?
    - If not, fix it, try again
    - If yes, add to mount table, allow access

File Sharing

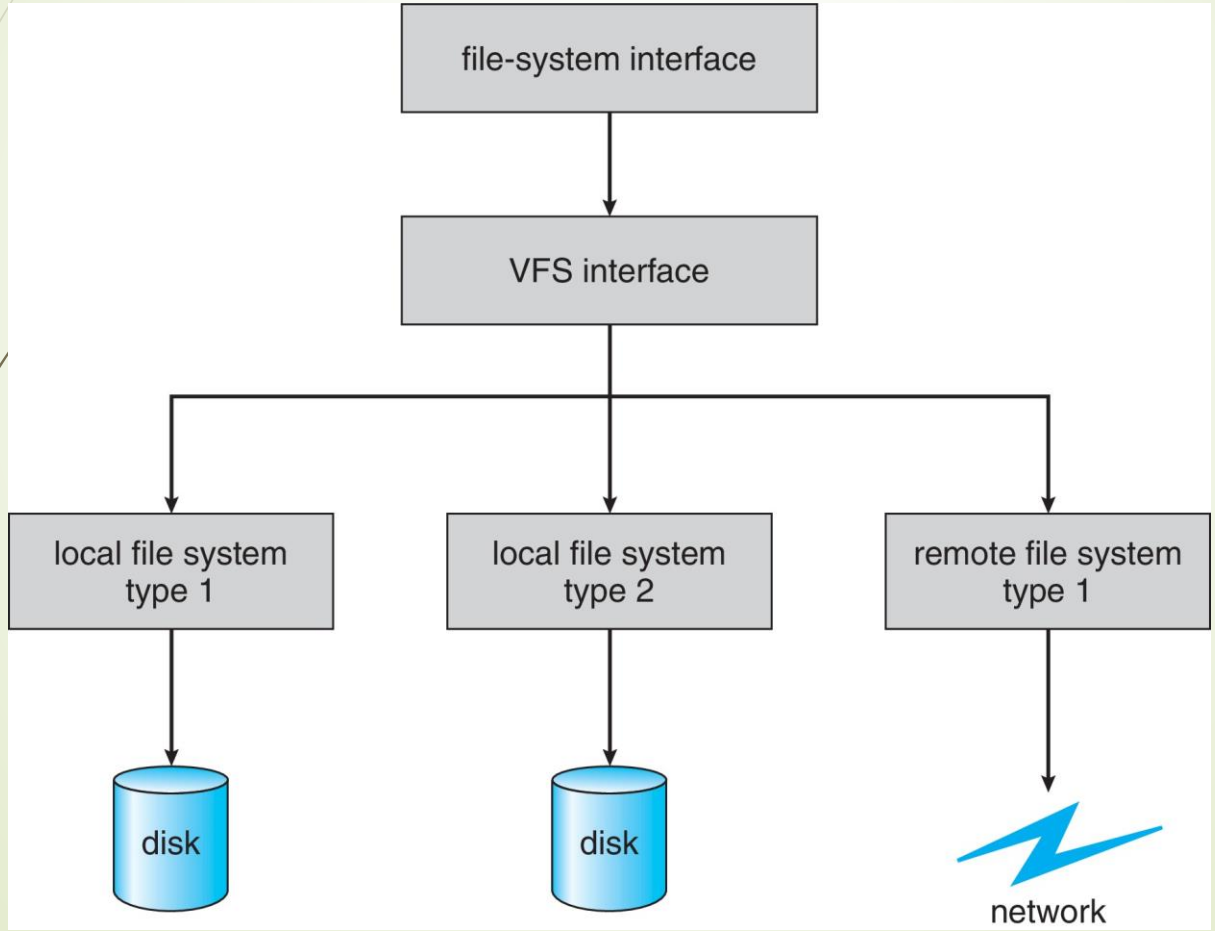
- Allows multiple users / systems access to the same files
- Permissions / protection must be implemented and accurate
  - Most systems provide concepts of owner, group member
  - Must have a way to apply these between systems

Virtual File Systems

- **Virtual File Systems** (**VFS**) on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Separates file-system generic operations from implementation details
  - Implementation can be one of many file systems types, or network file system
    - Implements **vnodes** which hold inodes or network file details
  - Then dispatches operation to appropriate file system implementation routines

Virtual File Systems (Cont.)

- The API is to the VFS interface, rather than any specific type of file system



[Source: Operating Systems Concepts, 10th ed.]

Virtual File System Implementation

- For example, Linux has four object types:
  - **inode, file, superblock, dentry**
- VFS defines set of operations on the objects that must be implemented
  - Every object has a pointer to a function table
    - Function table has addresses of routines to implement that function on that object
    - For example:
    - • **`int open(. . .)`**—Open a file
    - • **`int close(. . .)`**—Close an already-open file
    - • **`ssize t read(. . .)`**—Read from a file
    - • **`ssize t write(. . .)`**—Write to a file
    - • **`int mmap(. . .)`**—Memory-map a file

Remote File Systems

- Sharing of files across a network
- First method involved manually sharing each file – programs like `ftp`
- Second method uses a **distributed file system** (**DFS**)
  - Remote directories visible from local machine
- Third method – **World Wide Web**
  - A bit of a revision to the first method
  - Use browser to locate file/files and download /upload
  - **Anonymous** access doesn't require authentication

Client-Server Model

- Sharing between a server (providing access to a file system via a network protocol) and a client (using the protocol to access the remote file system)
- Identifying each other via network ID can be spoofed, encryption can be expensive
- NFS an example
  - User auth info on clients and servers must match (UserIDs for example)
  - Remote file system mounted, file operations sent on behalf of user across network to server
  - Server checks permissions, file handle returned
  - Handle used for reads and writes until file closed

Consistency Semantics

- Important criteria for evaluating file sharing-file systems
- Specify how multiple users are to access shared file simultaneously
  - When modifications of data will be observed by other users
  - Directly related to process synchronization algorithms, but atomicity across a network has high overhead (see Andrew File System)
- The series of accesses between file open and closed called **file session**
- UNIX semantics
  - Writes to open file immediately visible to others with file open
  - One mode of sharing allows users to share pointer to current I/O location in file
  - Single physical image, accessed exclusively, contention causes process delays
- Session semantics (Andrew file system (OpenAFS))
  - Writes to open file not visible during session, only at close
  - Can be several copies, each changed independently

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 originally part of SunOS, now industry standard / very common
  - Can use unreliable datagram protocol (UDP/IP) or TCP/IP, over Ethernet or other network

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

NFS Mount Protocol

- Establishes initial logical connection between server and client
- 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
  - Export list – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them
- Following a mount request that conforms to its export list, the server returns a file handle—a key for further accesses
  - File handle – 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

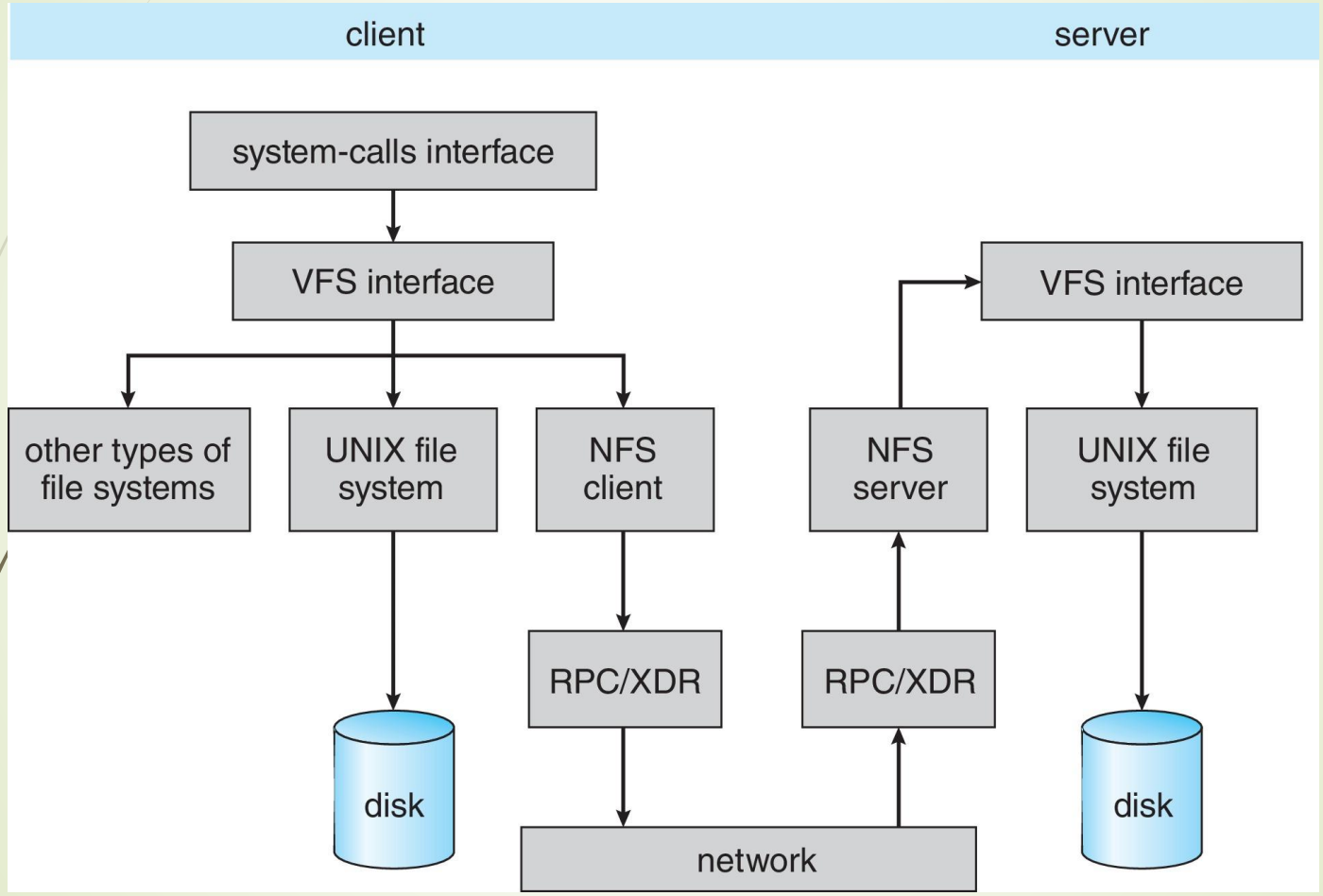
NFS Protocol

- Provides a set of remote procedure calls for remote file 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 newer, less used – 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

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



[Source: Operating Systems Concepts, 10th ed.]

NFS Path-Name Translation

- 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 side holds the vnodes for remote directory names

NFS Remote Operations

- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files)
- NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance
- File-blocks cache – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes
  - Cached file blocks are used only if the corresponding cached attributes are up to date
- File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk

End of Module 3