VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SWE3001-Operating Systems

Prepared By
Dr. L. Mary Shamala
Assistant Professor
SCOPE/VIT

Module 7: Mass Storage Structure

- Overview of Mass Storage Structure
- Disk Structure
- Disk Attachment
- Disk Scheduling
- Disk Management
- Swap-Space Management
- RAID Structure

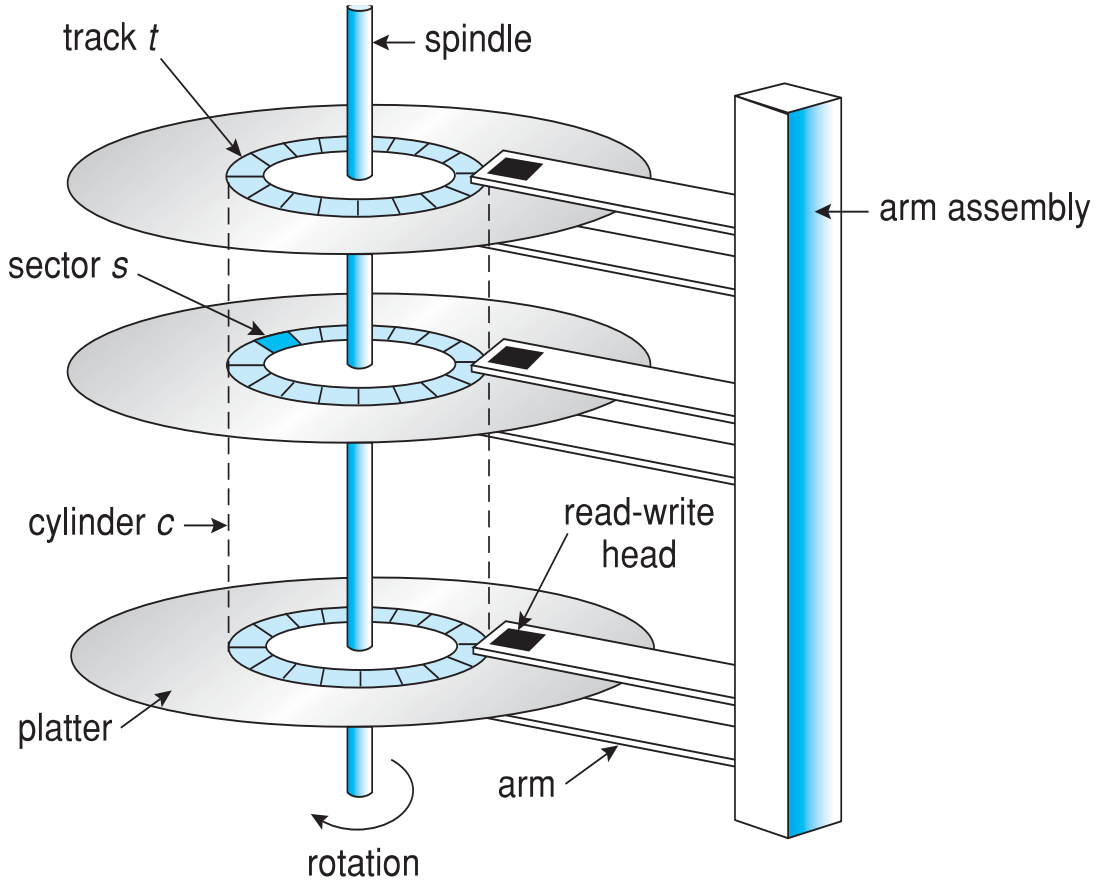
Objectives

- To describe the physical structure of secondary storage devices and its effects on the uses of the devices
- To explain the performance characteristics of mass-storage devices
- To evaluate disk scheduling algorithms
- To discuss operating-system services provided for mass storage, including RAID

Overview of Mass Storage Structure

- **Magnetic disks** provide bulk of secondary storage of modern computers
  - Drives rotate at 60 to 250 times per second
  - **Transfer rate** is rate at which data flow between drive and computer
  - **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
  - **Head crash** results from disk head making contact with the disk surface -- That's bad
- Disks can be removable
- Drive attached to computer via **I/O bus**
  - Busses vary, including **EIDE**, **ATA**, **SATA**, **USB**, **Fibre Channel**, **SCSI, SAS, Firewire**
  - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

Moving-head Disk Mechanism



Hard Disks

- Platters range from .85” to 14” (historically)
  - Commonly 3.5”, 2.5”, and 1.8”
- Range from 30GB to 3TB per drive
- Performance
  - Transfer Rate – theoretical – 6 Gb/sec
  - Effective Transfer Rate – real – 1Gb/sec
  - Seek time from 3ms to 12ms – 9ms common for d
  - Average seek time measured or calculated based
  - Latency based on spindle speed
    - 1 / (RPM / 60) = 60 / RPM
  - Average latency = ½ latency

Spindle [rpm]	Average latency [ms]
4200	7.14
5400	5.56
7200	4.17
10000	3
15000	2

(From Wikipedia)

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 =
  - 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

Solid-State Disks

- Nonvolatile memory used like a hard drive
  - Many technology variations
- Can be more reliable than HDDs
- More expensive per MB
- Maybe have shorter life span
- Less capacity
- But much faster
- Busses can be too slow -> connect directly to PCI for example
- No moving parts, so no seek time or rotational latency

Magnetic Tape

- Was early secondary-storage medium
  - Evolved from open spools to cartridges
- 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
  - 140MB/sec and greater
- 200GB to 1.5TB typical storage
- Common technologies are LTO-{3,4,5} and T10000

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer
  - Low-level formatting creates **logical blocks** on physical media
- 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
  - Logical to physical address should be easy
    - Except for bad sectors
    - Non-constant # of sectors per track via constant angular velocity

Disk Attachment

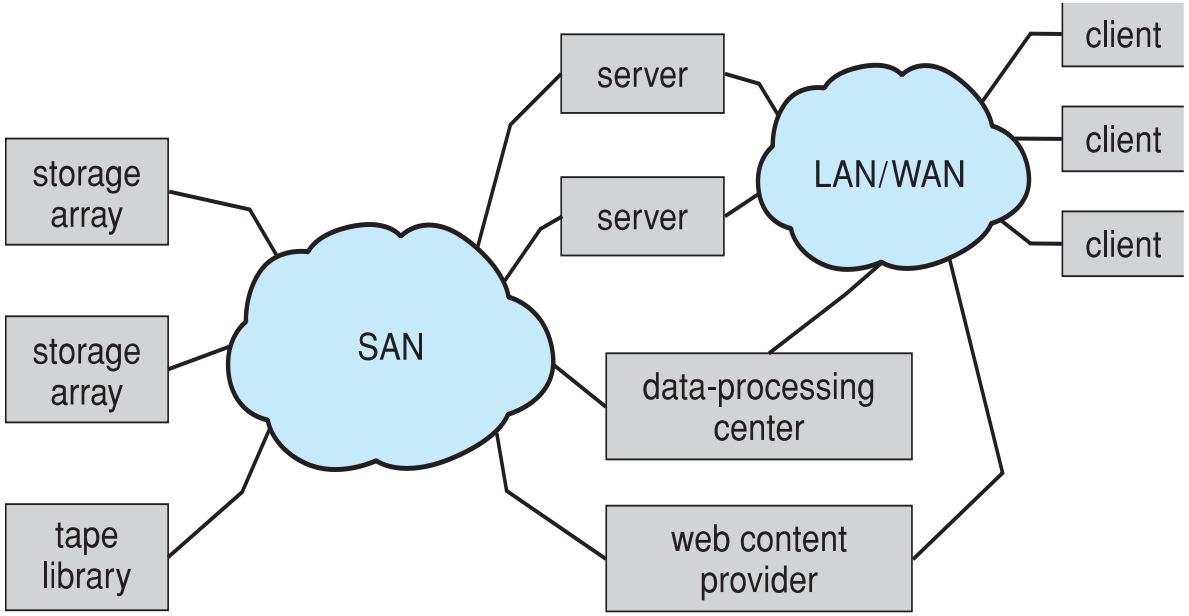
- Host-attached storage accessed through I/O ports talking to I/O busses
- SCSI itself is a bus, up to 16 devices on one cable, **SCSI initiator** requests operation and **SCSI targets** perform tasks
  - Each target can have up to 8 **logical units** (disks attached to device controller)
- FC is high-speed serial architecture
  - Can be switched fabric with 24-bit address space – the basis of **storage area networks (SAN**s) in which many hosts attach to many storage units
- I/O directed to bus ID, device ID, logical unit (LUN)

Storage Array

- Can just attach disks, or arrays of disks
- 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

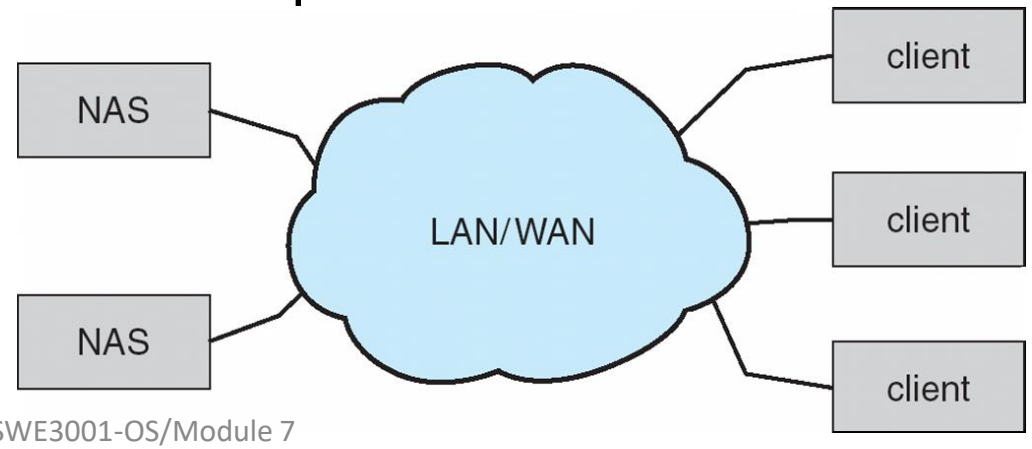


Storage Area Network (Cont.)

- SAN is one or more storage arrays
  - Connected to one or more Fibre Channel switches
- 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
  - Over low-latency Fibre Channel fabric
- Why have separate storage networks and communications networks?
  - Consider iSCSI, FCOE

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)



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
- Minimize seek time
- The **seek time** is the time for the disk arm to move the heads to the cylinder containing the desired sector.
- The **rotational latency** is the additional time for the disk to rotate the desired sector to the disk head.
- 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 request
  - 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

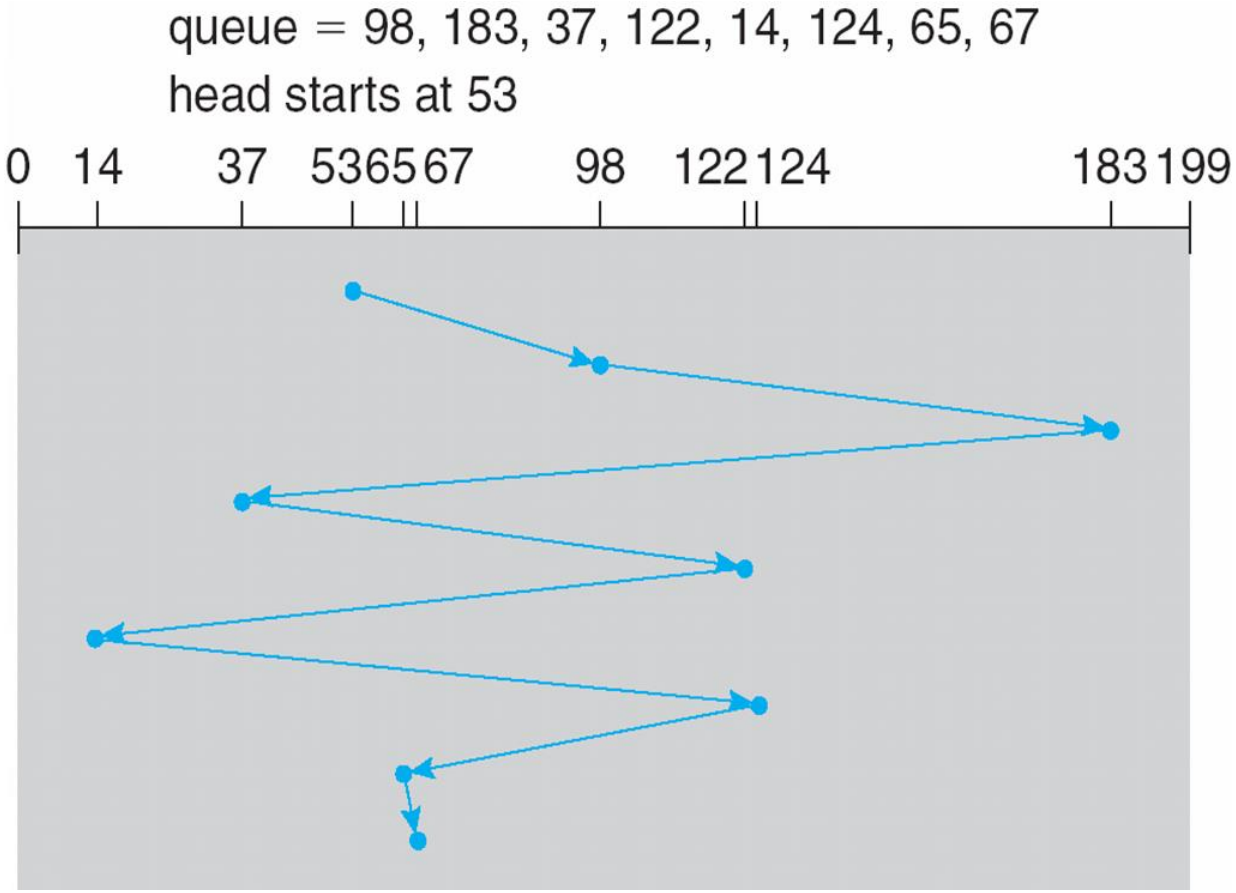
Disk Scheduling (Cont.)

- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- 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

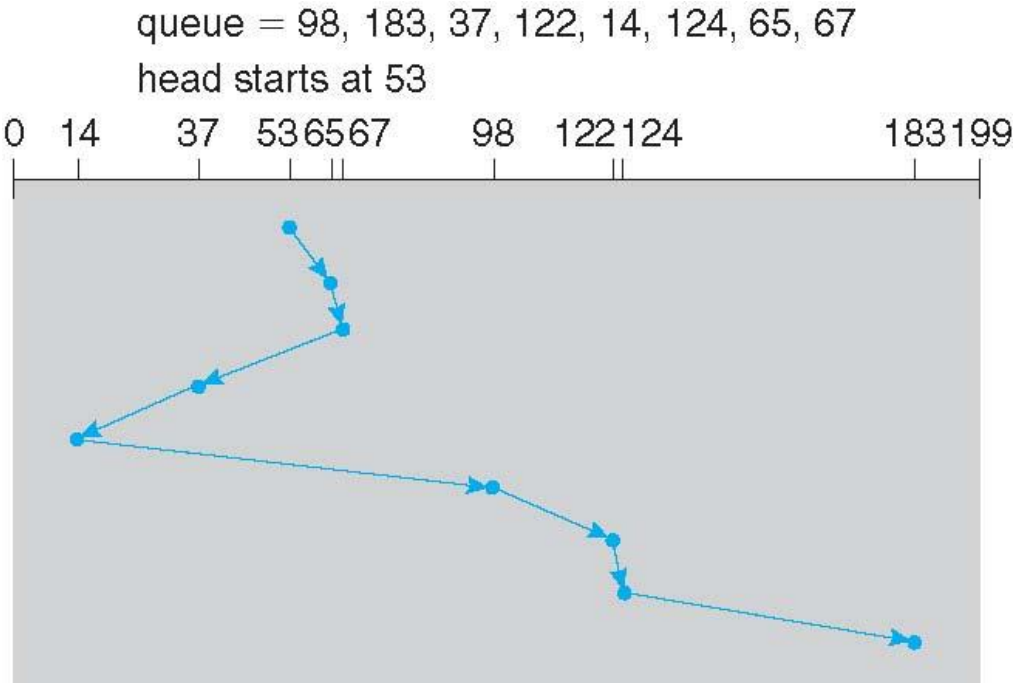
FCFS



Total head movement = 640 cylinders

SSTF

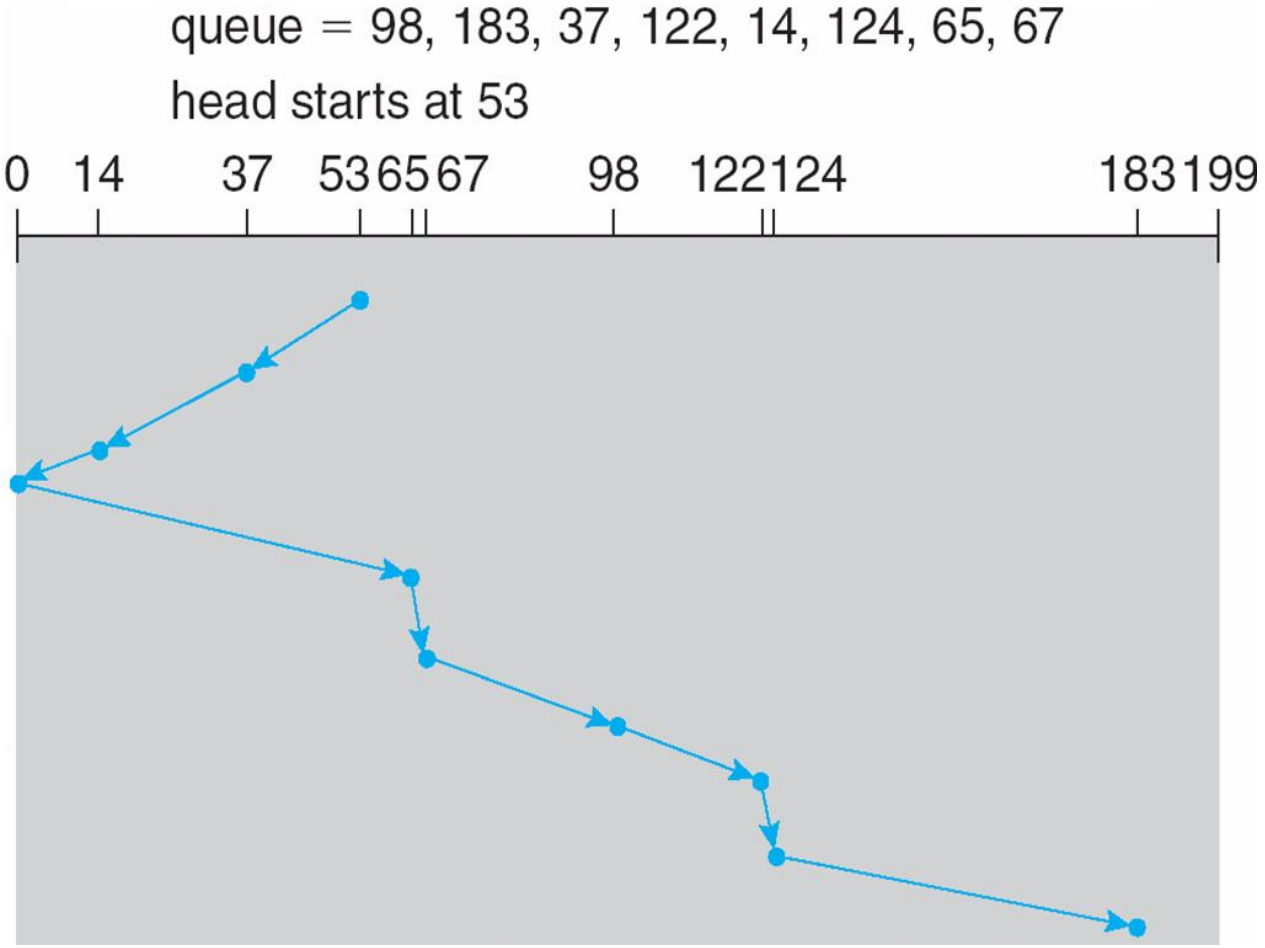
- Shortest Seek Time First selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- Total head movement =236 cylinders



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.
- **SCAN algorithm** Sometimes called the **elevator algorithm**
- Illustration shows total head movement of 236 cylinders
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

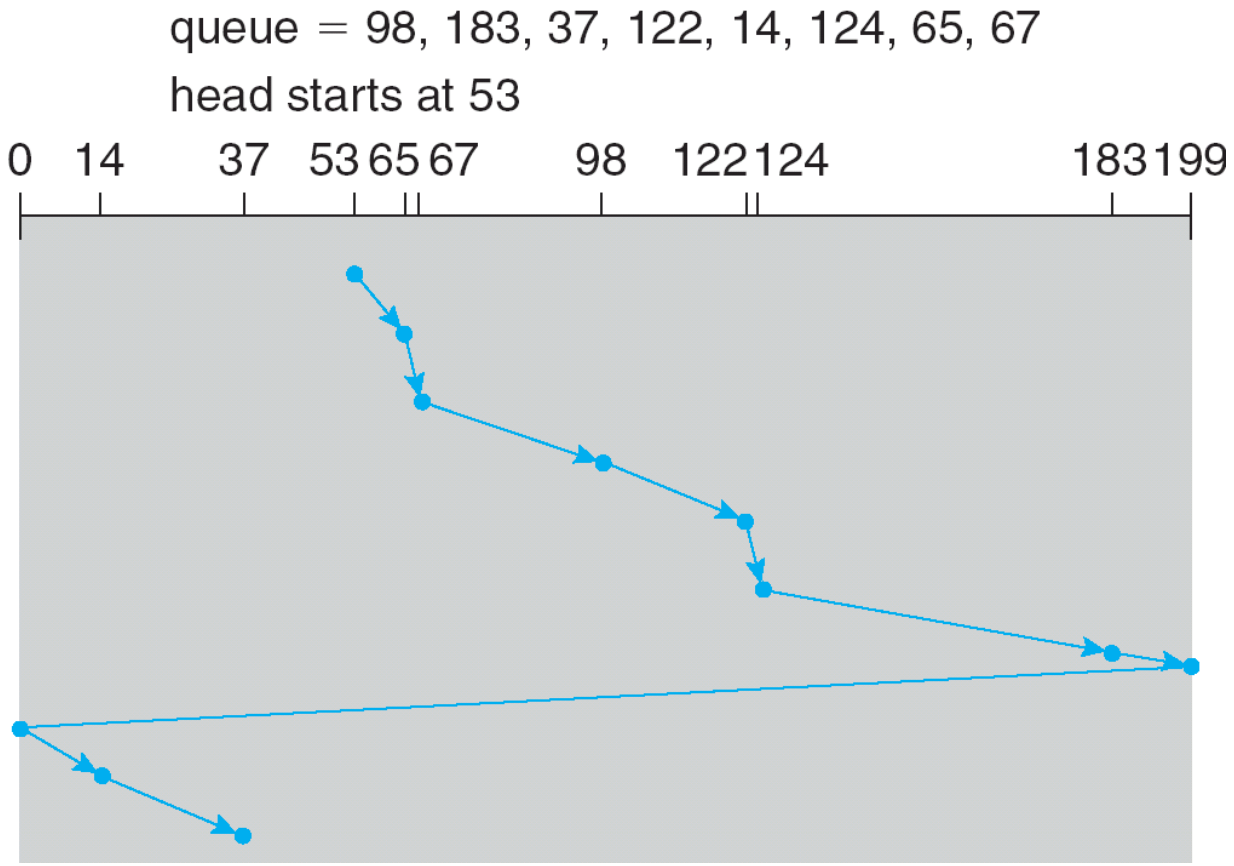
SCAN (Cont.)


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
- Total number of cylinders?

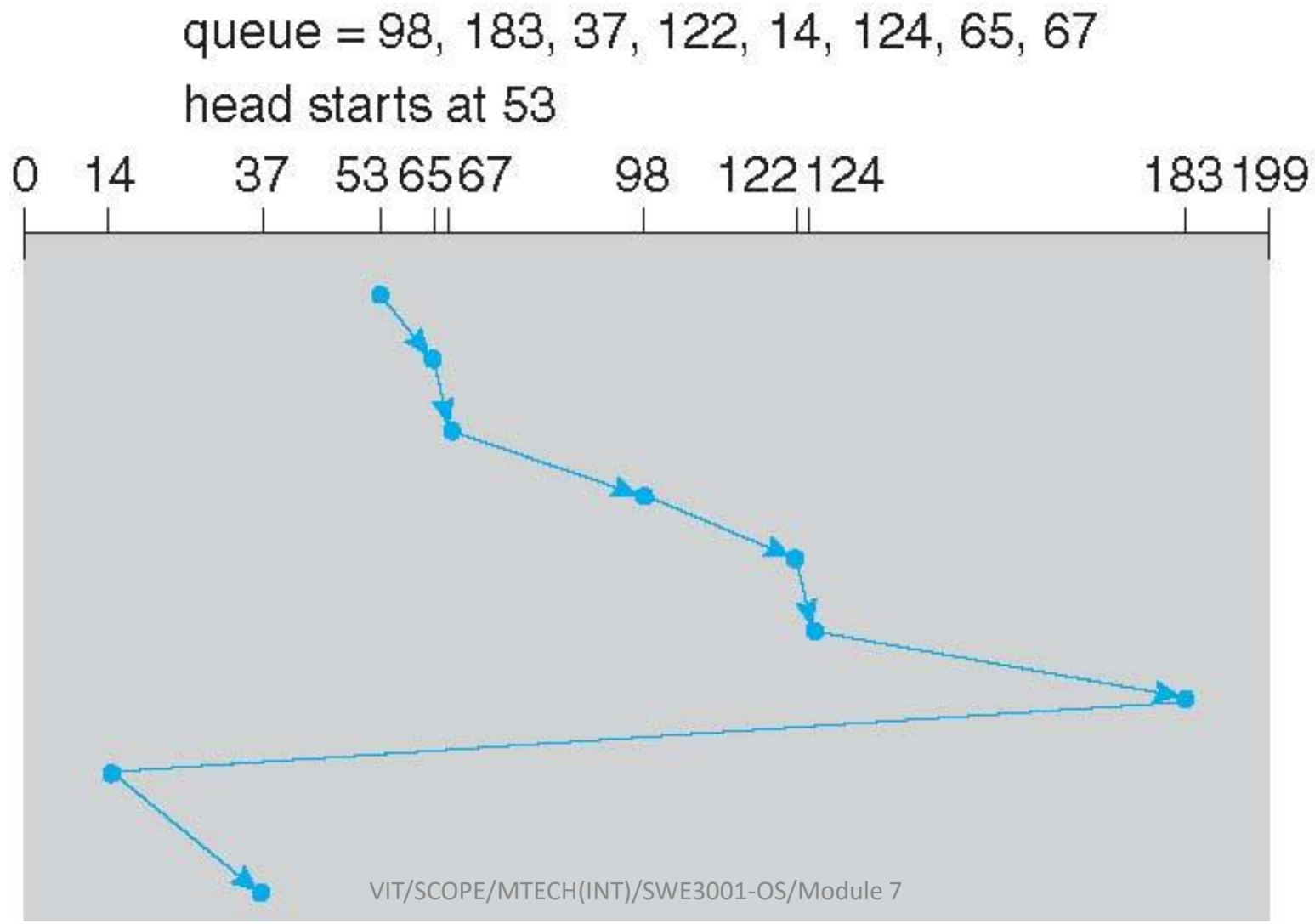
C-SCAN (Cont.)


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

C-LOOK

- LOOK a version of SCAN, C-LOOK a version of 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
- Total number of cylinders?

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
  - Less starvation
- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method
  - And metadata layout
- 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
- What about rotational latency?
  - Difficult for OS to calculate
- How does disk-based queueing effect OS queue ordering efforts?

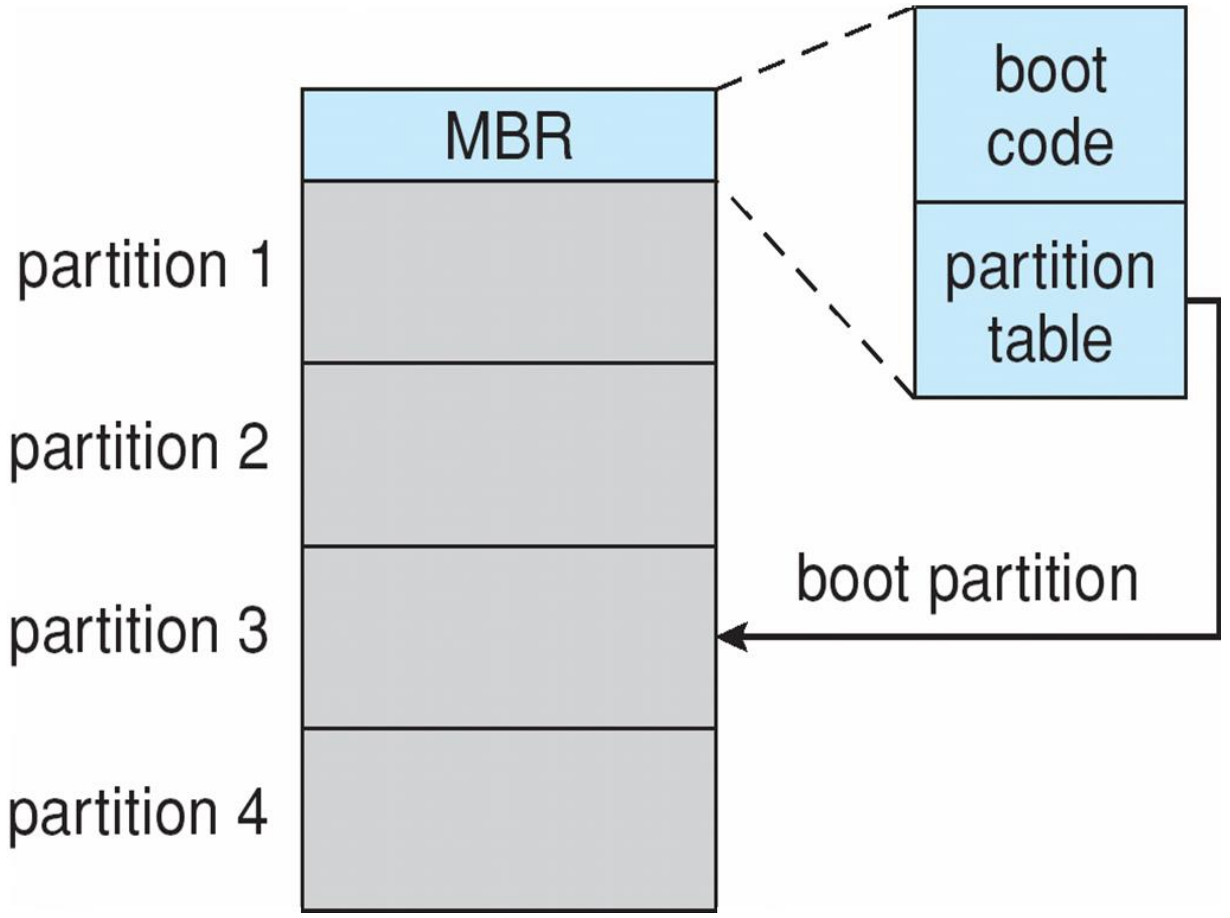
Disk Management

- **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

Disk 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
  - **Bootstrap loader** program stored in boot blocks of boot partition
- Methods such as **sector sparing** used to handle bad blocks

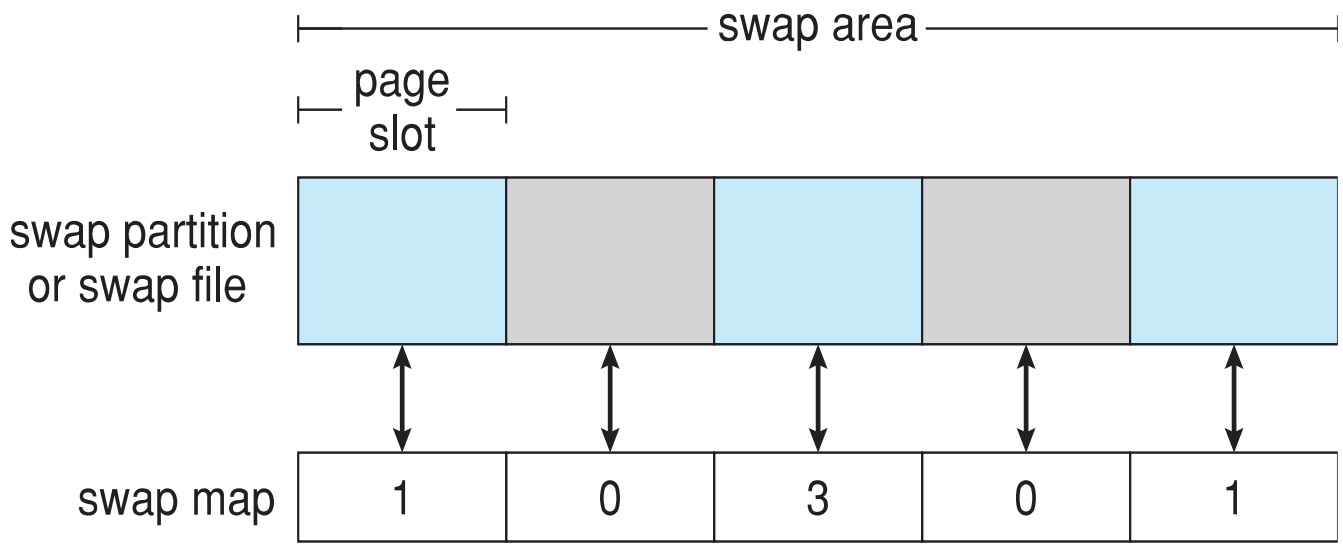
Booting from a Disk in Windows



Swap-Space Management

- Swap-space — Virtual memory uses disk space as an extension of main memory
  - Less common now due to memory capacity increases
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition (raw)
- 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 dirty page is forced out of physical memory, not when the virtual memory page is first created
    - File data written to swap space until write to file system requested
    - Other dirty pages go to swap space due to no other home
    - Text segment pages thrown out and reread from the file system as needed
- What if a system runs out of swap space?
- Some systems allow multiple swap spaces

Data Structures for Swapping on Linux Systems



RAID Structure

- RAID – redundant array of inexpensive disks
  - multiple disk drives provides reliability via **redundancy**
- RAID s are used for their higher reliability and higher data-transfer rate.
- Increases the **mean time to failure**

Improvement of Reliability via Redundancy

- Suppose that the mean time to failure of a single disk is 100,000 hours.
- Then the mean time to failure of some disk in an array of 100 disks will be 100,000/100 = 1,000 hours, or 41.66 days.
- If we store only one copy of the data, then each disk failure will result in loss of a significant amount of data; such a high rate of data loss is unacceptable.
- The solution to the problem of reliability is to introduce redundancy
- The simplest approach is **Mirroring**: duplicate every disk
  - logical disk consists of two physical disks, and every write is carried out on both disks.
  - The result is called a mirrored volume.

Improvement of Reliability via Redundancy…

- The mean time to failure of a mirrored depends on two factors:
  1. Mean time to failure of the individual disks
  2. **Mean time to repair –** the time it takes to replace a failed disk and to restore the data on it
- **Mean time to data loss** is based on above factors
  - Suppose mirrored disks fail independently; Then if the mean time to failure of a single disk is 100,000 hours and the mean time to repair is 10 hours
  - Mean time to data loss is $100,000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years!
- Mirrored-disk systems offer much higher reliability than do single-disk systems
- Frequently combined with **NVRAM** to improve write performance
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively

Improvement in Performance via Parallelism

- With multiple disks, we can improve the transfer rate as well by striping data across the disks.
- Disk **striping** uses a group of disks as one storage unit
  - Bit-level striping: splitting the bits of each byte across multiple disks
  - Block-level striping: blocks of a file are striped across multiple disks
    - With n disks, block i of a file goes to disk (i mod n) + 1
- Other levels of striping, such as bytes of a sector or sectors of a block, also are possible.
- Block-level striping is the most common
- Parallelism by striping, has two main goals:
  - Increase the throughput of multiple small accesses (that is, page accesses) by load balancing.
  - Reduce the response time of large accesses.

RAID Levels

- Mirroring provides high reliability, but it is expensive.
- Striping provides high data-transfer rates, but it does not improve reliability.
- Numerous schemes provide redundancy at lower cost by using disk striping combined with “parity” bits.
- These schemes have different cost–performance trade-offs and are classified according to levels called RAID levels.
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data.
- **RAID level 0-** refers to disk arrays with striping at the level of blocks but without any redundancy

RAID Levels…

- **Mirroring** or **shadowing** (**RAID level 1**) keeps duplicate of each disk
- **RAID level 2** is also known as memory-style error-correcting- code (ECC) organization.
  - RAID level 2 requires only three disks' overhead for four disks of data, unlike RAID level 1, which requires four disks' overhead.
- **RAID level 3** or bit-interleaved parity organization, improves on level 2
  - a single parity bit can be used for error correction as well as for detection
  - less expensive in the number of extra disks required (it has only a one-disk overhead)
  - RAID level 3 has two advantages over level 1.
    - storage overhead is reduced
    - the transfer rate is faster
  - Limitations
    - Supports fewer I/Os per second
    - Significantly slower writes

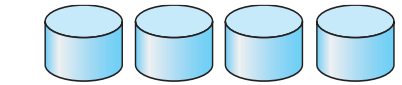
RAID Levels…

- **RAID level 4**, or block-interleaved parity organization, uses block-level striping, as in RAID 0; keeps a parity block on a separate disk for corresponding blocks from N other disk
  - Higher overall I/O rate.
  - Small independent writes cannot be performed in parallel
  - **Read-modify-write cycle**- A single write requires four disk accesses: two to read the two old blocks and two to write the two new blocks
- **RAID level 5**, or block-interleaved distributed parity, differs from level 4 in that it spreads 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.
  - Avoids potential overuse of a single parity disk
  - Most common parity RAID system.

RAID Levels…

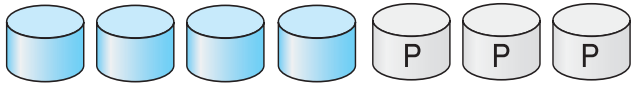
- **RAID level 6**, also called the **P + Q redundancy scheme**, is much like RAID level 5 but stores extra redundant information to guard against multiple disk failures.
  - Instead of parity, error-correcting codes such as the Reed–Solomon codes are used.

RAID Levels

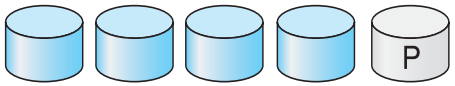


(a) RAID 0: non-redundant striping.

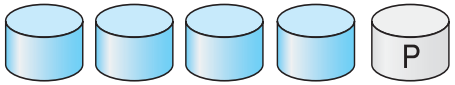
(b) RAID 1: mirrored disks.



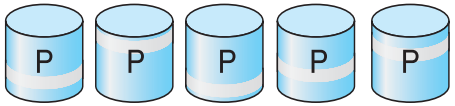
(c) RAID 2: memory-style error-correcting codes.



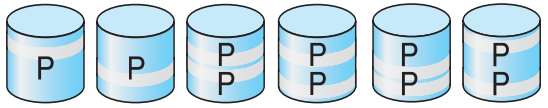
(d) RAID 3: bit-interleaved parity.



(e) RAID 4: block-interleaved parity.



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



(g) RAID 6: P + Q redundancy.

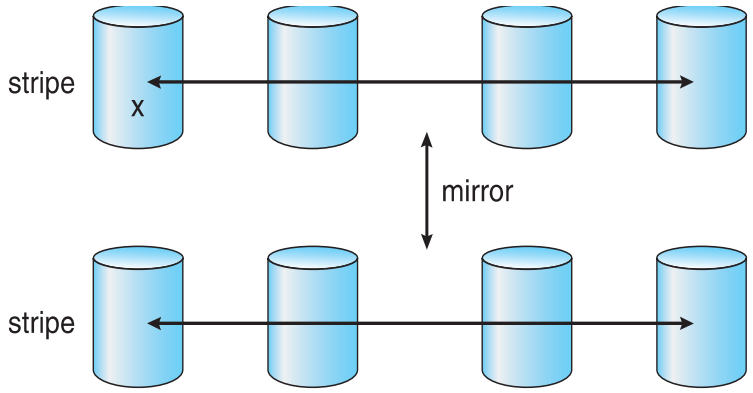
RAID 0+1 and RAID 1+0

- Striped mirrors (**RAID 1+0**) or mirrored stripes (**RAID 0+1**) provides high performance and high reliability
- RAID level 0 + 1 refers to a combination of RAID levels 0 and 1.
- This level provides better performance than RAID 5.
- It is common in environments where both performance and reliability are important.
- In RAID 0 + 1, a set of disks are striped, and then the stripe is mirrored to another, equivalent stripe.
- It doubles the number of disks needed for storage, so it is also relatively expensive

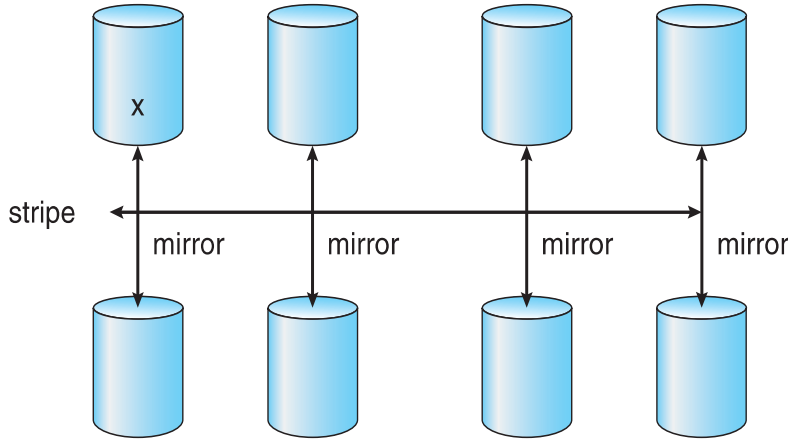
RAID 0+1 and RAID 1+0…

- In RAID 1 + 0, disks are mirrored in pairs and then the resulting mirrored pairs are striped.
- Advantage
  - 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
- Numerous variations have been proposed to the basic RAID schemes

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



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



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

RAID Implementations

- Volume-management software can implement RAID within the kernel or at the system software layer.
- Host bus-adapter (HBA) hardware.
- Hardware of the storage array.
- SAN interconnect layer by disk virtualization devices.

Other Features

- Regardless of where RAID implemented, other useful features can be added
- **Snapshot** is a view of the file system before a set of changes take place
- **Replication** is automatic duplication of writes between separate sites
  - For redundancy and disaster recovery
  - Can be synchronous or asynchronous
- The implementation of these features differs depending on the layer at which RAID is implemented.
- RAID within a storage array can still fail if the array fails, so automatic **replication** of the data between arrays is common
- A **hot spare** is not used for data but is configured to be used as a replacement in case of disk failure.
  - Decreases mean time to repair

Selecting a RAID Level

- Rebuild performance.
  - influences the mean time to failure
- Varies with the RAID level used.
- RAID level 0 is used in high-performance applications where data loss is not critical.
- RAID level 1 is popular for applications that require high reliability with fast recovery.
- RAID 0 + 1 and 1 + 0 are used where both performance and reliability are important (for small databases)
- RAID 5 is often preferred for storing large volumes of data.
- Level 6 for better reliability than level 5.
- Several other decisions
  - How many disks should be in a given RAID set?
  - How many bits should be protected by each parity bit?

Extensions

- The concepts of RAID have been generalized to other storage devices
  - arrays of tapes
  - broadcast of data over wireless systems.
- When applied to arrays of tapes, RAID structures are able to recover data even if one of the tapes in an array is damaged.
- When applied to broadcast of data,
  - A block of data is split into short units and is broadcast along with a parity unit.
  - If one of the units is not received for any reason, it can be reconstructed from the other units.
- Tape-drive robots containing multiple tape drives will stripe data across all the drives to increase throughput and decrease backup time.

Problems with RAID

- RAID protects against physical media errors, but not other hardware and software errors.
- Lack of flexibility

Stable-Storage Implementation

- Write-ahead log scheme requires stable storage
- Stable storage means data is never lost (due to failure, etc)
- 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
- Disk write has 1 of 3 outcomes
  1. **Successful completion -** The data were written correctly on disk
  2. **Partial failure -** A failure occurred in the midst of transfer, so only some of the sectors were written with the new data, and the sector being written during the failure may have been corrupted
  3. **Total failure -** The failure occurred before the disk write started, so the previous data values on the disk remain intact

Stable-Storage Implementation (Cont.)

- If failure occurs during block write, recovery procedure restores block to consistent state
  - System maintains 2 physical blocks per logical block and does the following:
    1. Write to 1st physical
    2. When successful, write to 2nd physical
    3. Declare complete only after second write completes successfully

  Systems frequently use NVRAM as one physical to accelerate

File-System Interface

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- File Sharing
- Directory Implementation
- Allocation Methods

File System

- It provides the mechanism for on-line storage of and access to both data and programs of the operating system and all the users of the computer system.
- The file system consists of two distinct parts:
  - a **collection of files**, each storing related data,
  - a **directory structure**, which organizes and provides information about all the files in the system.

File Concept

- The operating system provides a uniform logical view of stored information.
- Define a logical storage unit, the **file** (Contiguous logical address space)
- Files are mapped by the operating system onto physical devices.
- A file is a named collection of related information that is recorded on secondary storage.
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- The information in a file is defined by its creator
  - Many different types of information may be stored in a file— **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
- Many variations, including extended file attributes such as file checksum
- Information about files are kept in the directory structure, which is maintained on the disk
- A directory entry consists of the file's name and its unique identifier.

File info Window on Mac OS X

File Operations

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

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 last processes closes it
  - **Disk location of the file**: cache of data access information
  - **Access rights:** per-process access mode information

Open File Locking

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

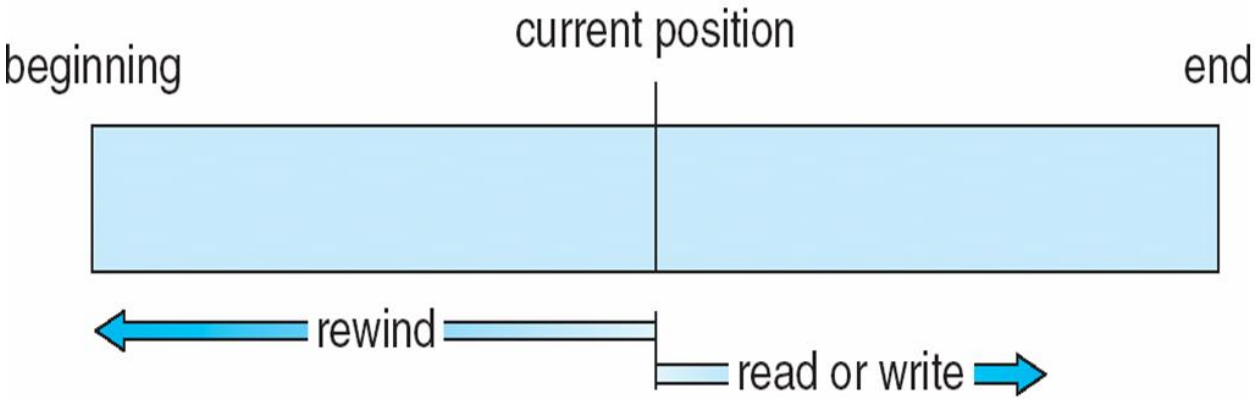
File Types – Name, Extension

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

File Structure

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

Sequential-access File



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

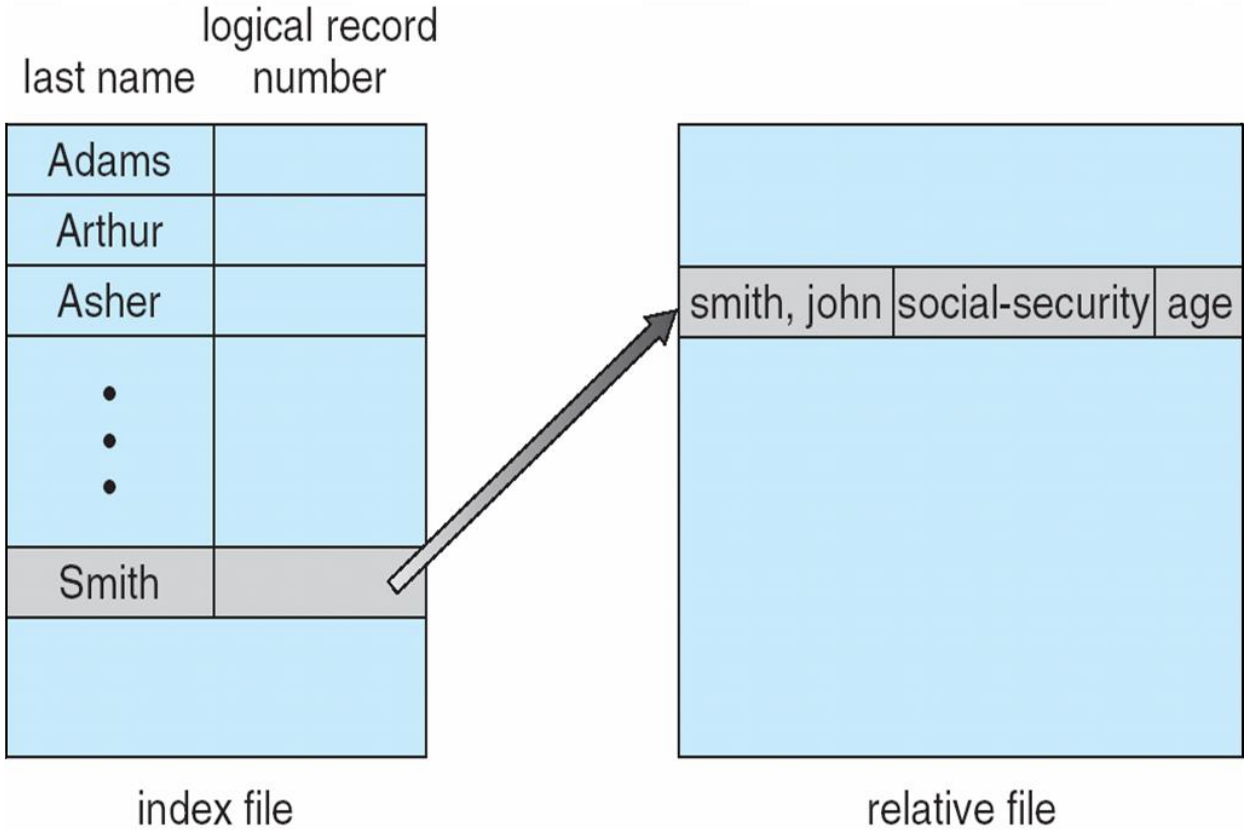
Simulation of Sequential Access on Direct-access File

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

Other Access Methods

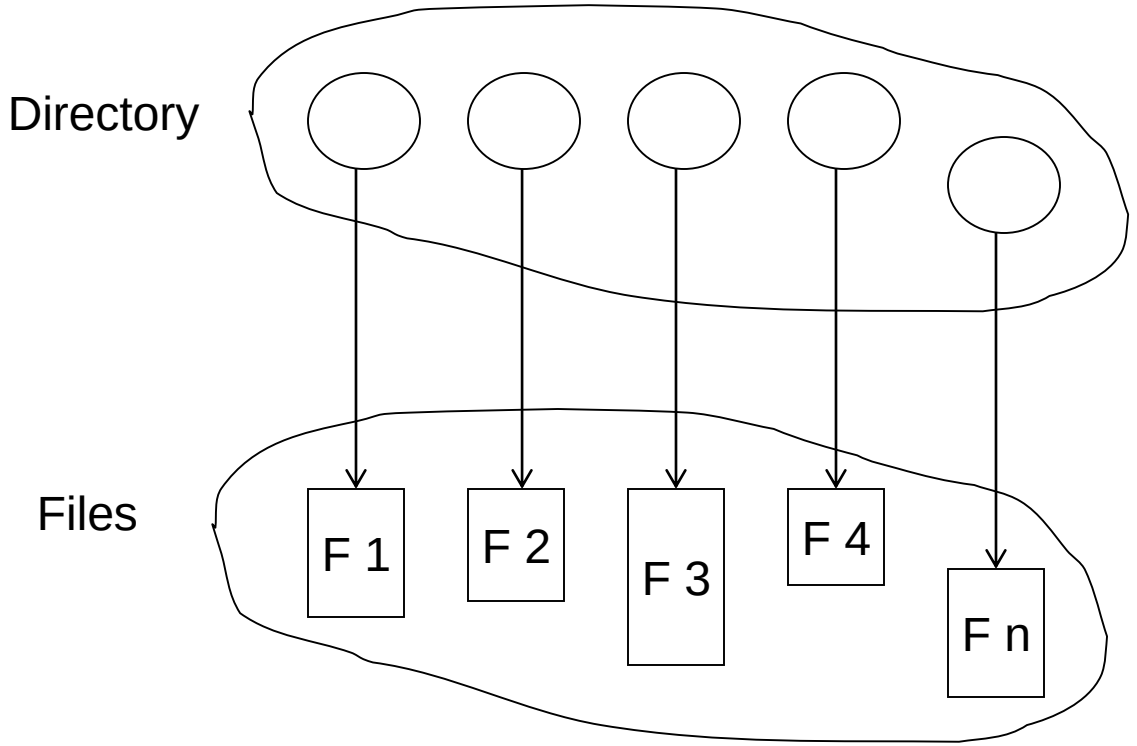
- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on
- If too large, index (in memory) of the index (on disk)
- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key
  - All done by the OS
- VMS operating system provides index and relative files as another example (see next slide)

Example of Index and Relative Files



Directory Structure

- A collection of nodes containing information about all files

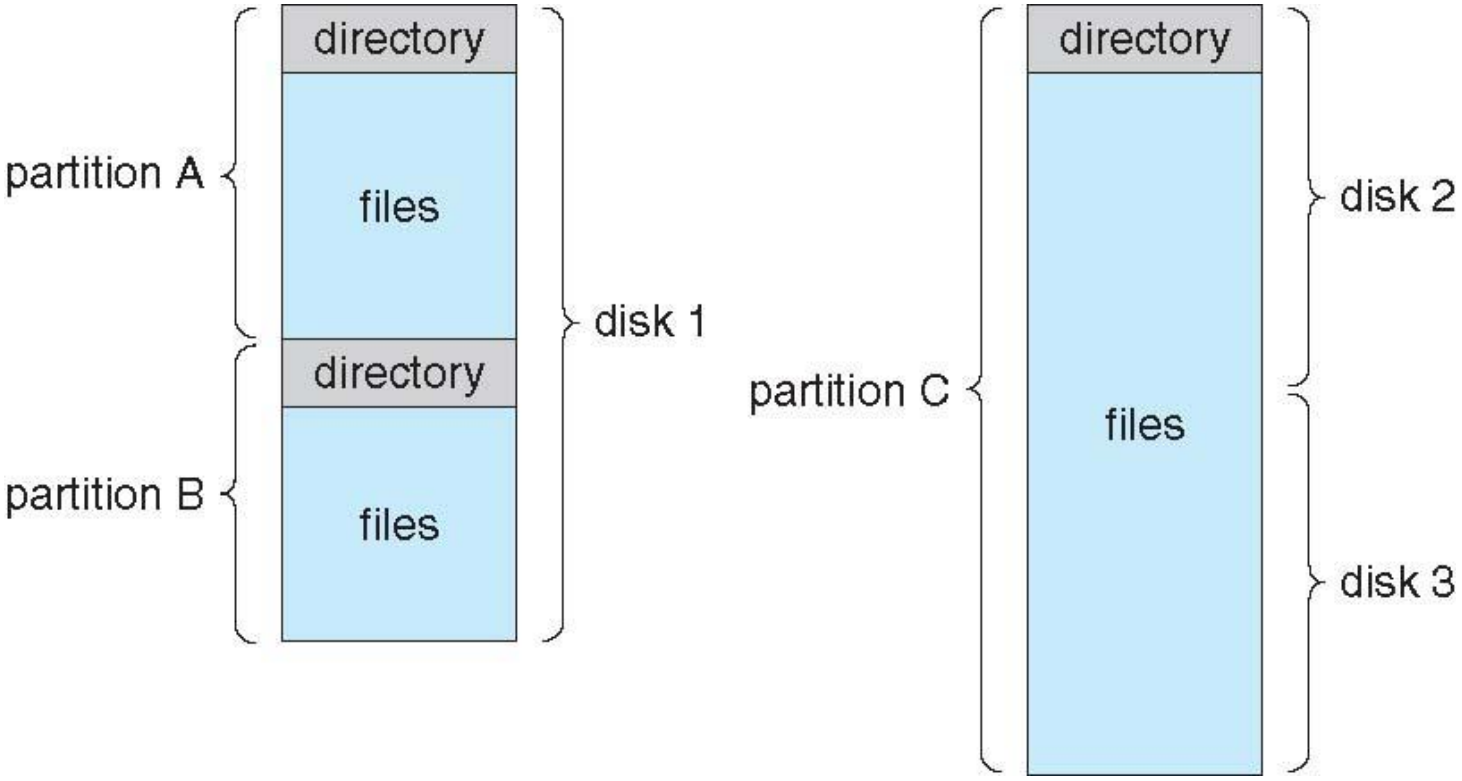


Both the directory structure and the files reside on disk

Disk Structure

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

A Typical File-system Organization



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

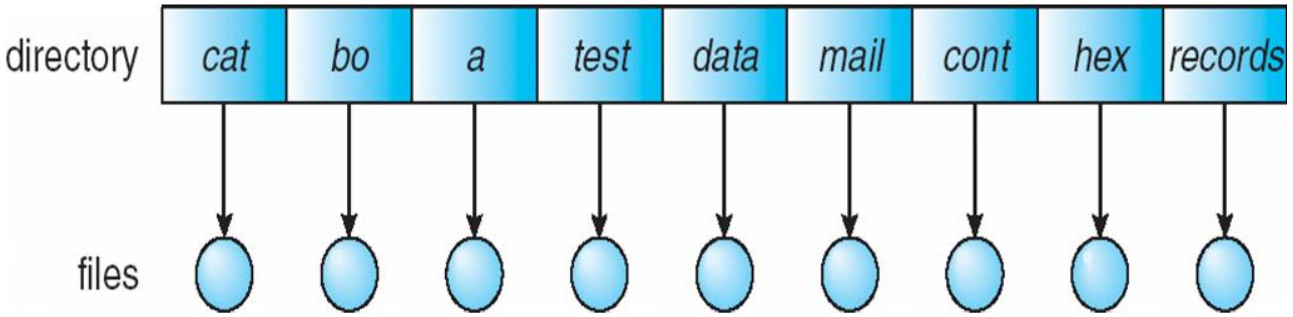
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Directory Organization

- The directory is organized logically to obtain:
- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have 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

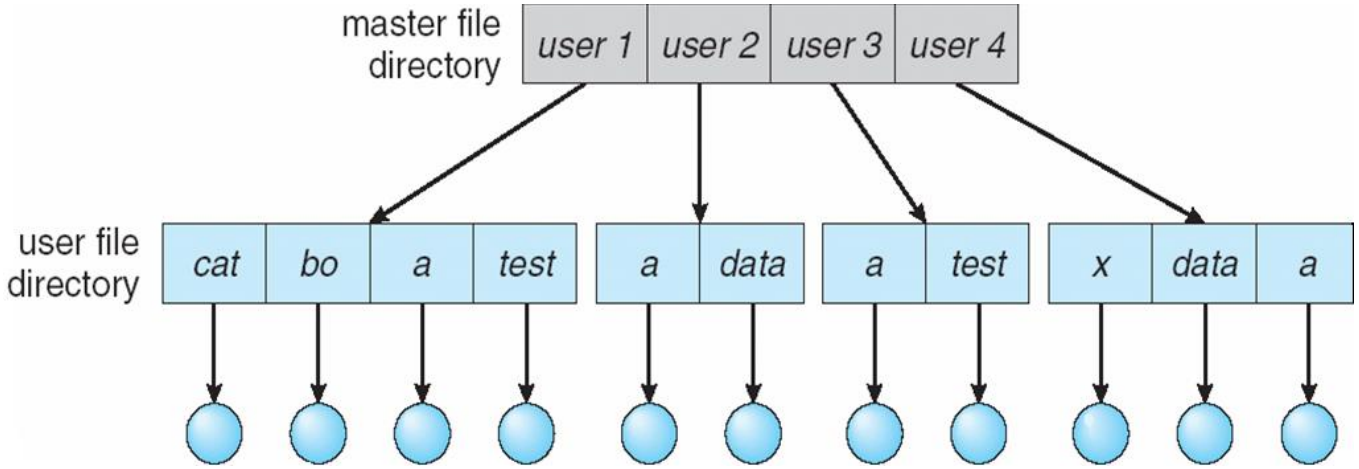
- A single directory for all users



- Naming problem
- Grouping problem

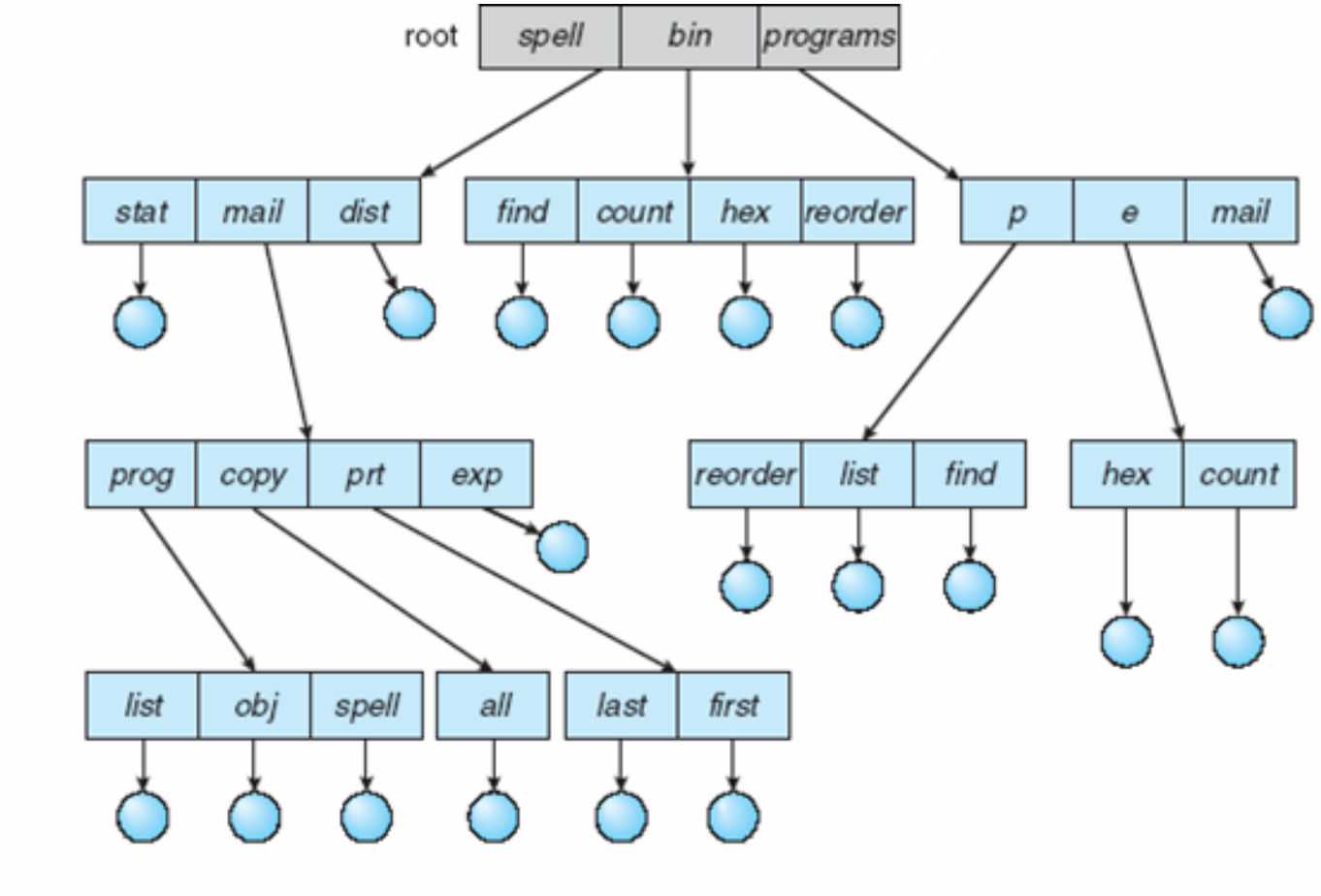
Two-Level Directory

- 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

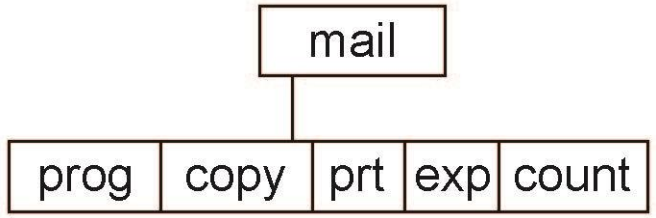
```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

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

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

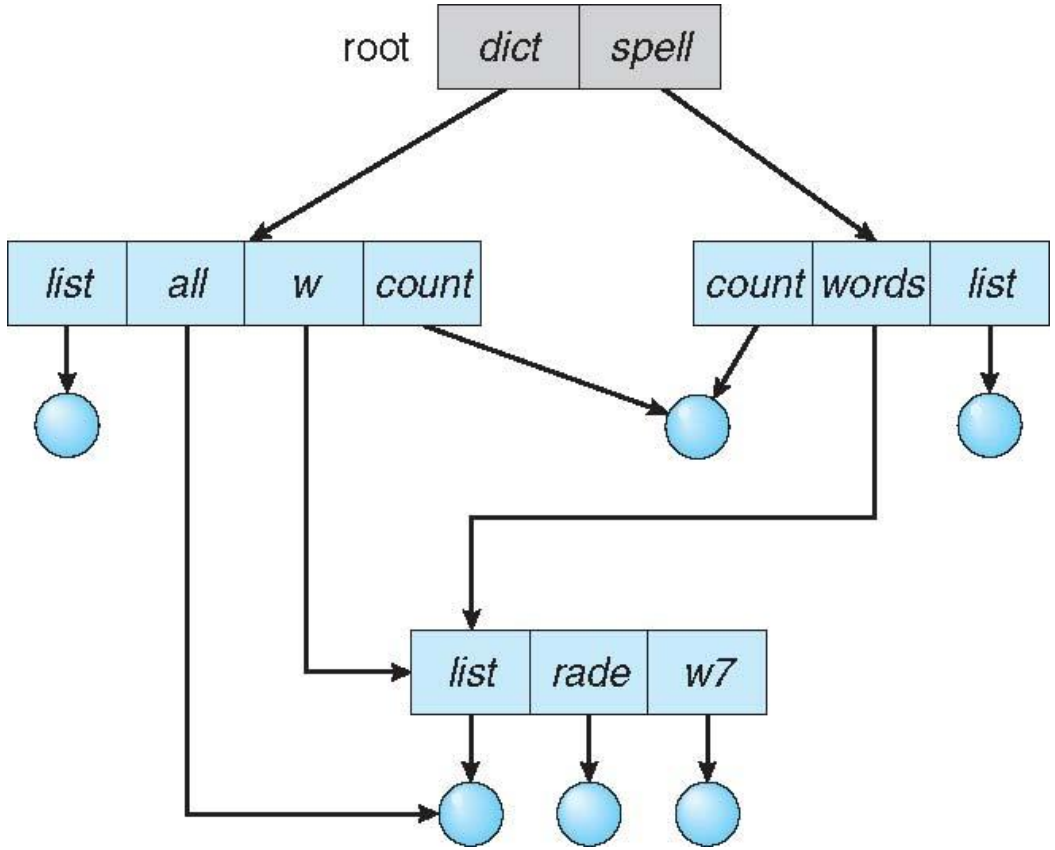
```
mkdir count
```



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

Acyclic-Graph Directories

- Have shared subdirectories and files

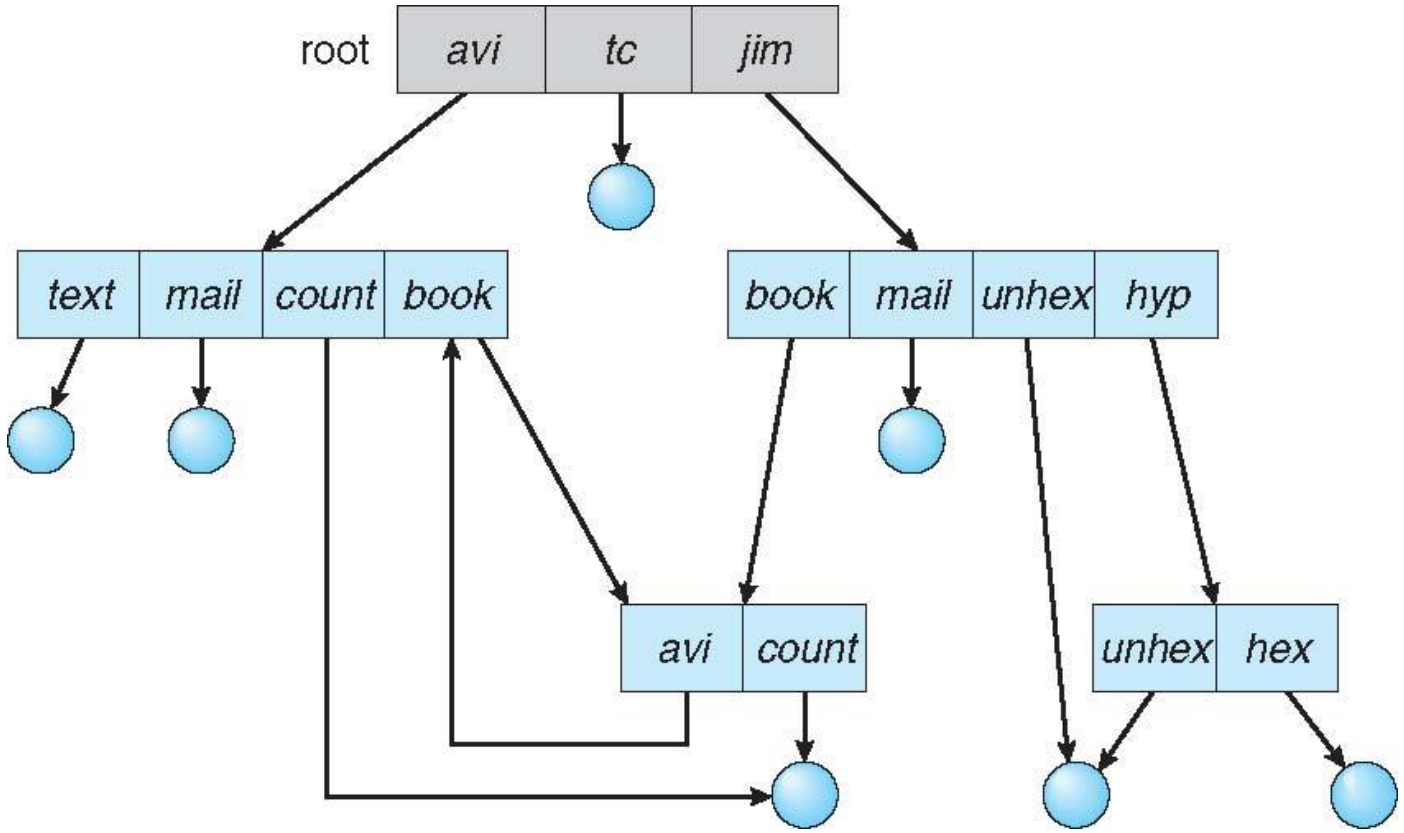


Acyclic-Graph Directories (Cont.)

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

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

General Graph Directory

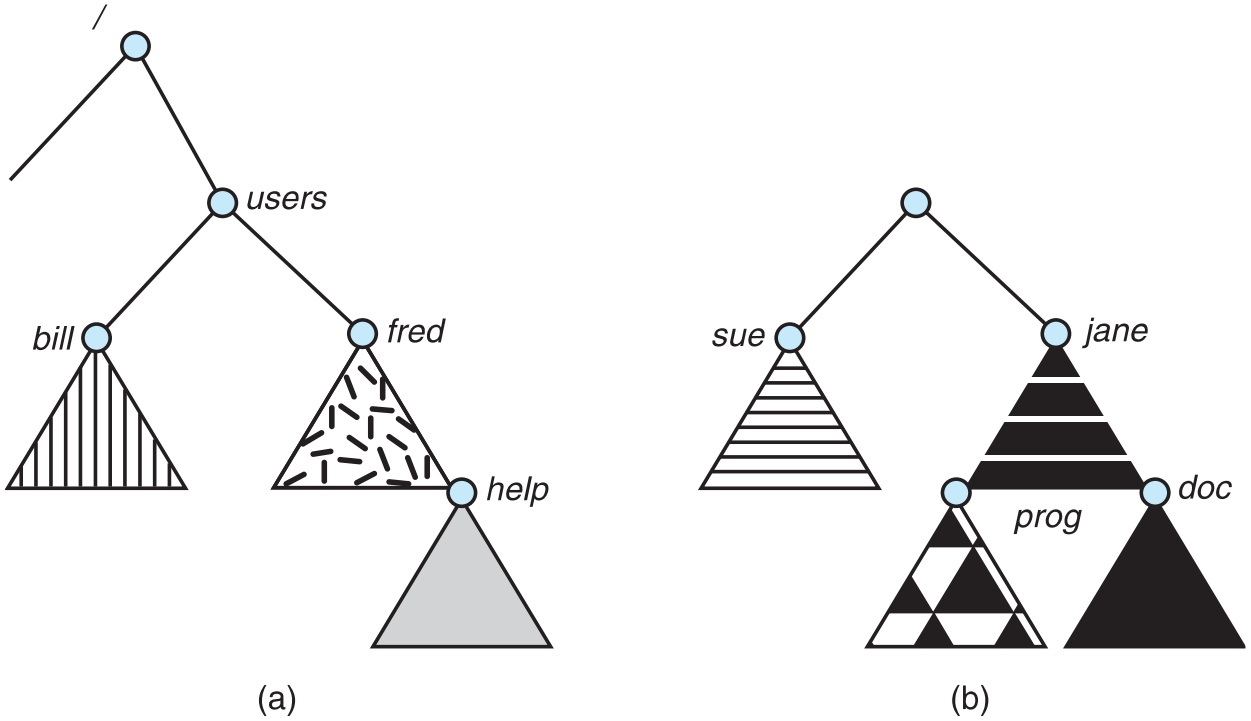


General Graph Directory (Cont.)

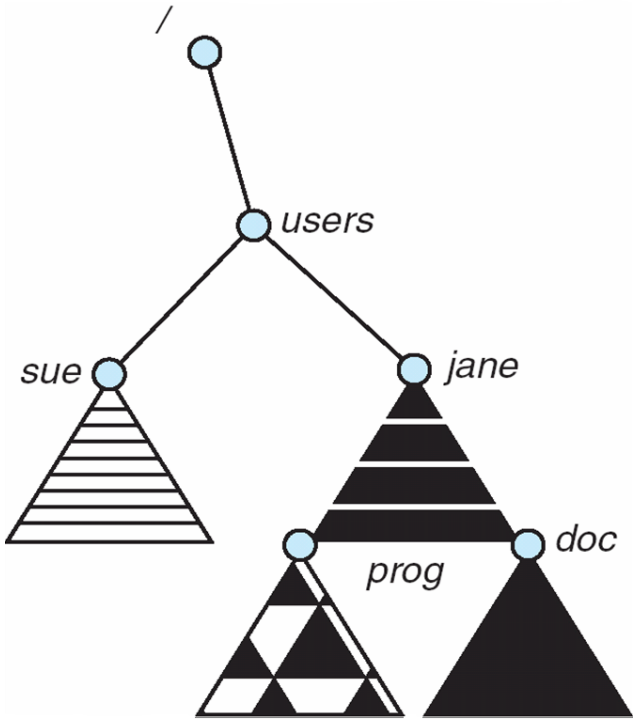
- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - **Garbage collection**
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

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**



Mount Point



File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- 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 operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

File Sharing – Failure Modes

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

Directory Implementation

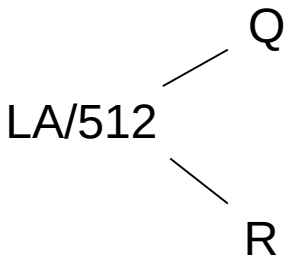
- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time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

Allocation Methods - Contiguous

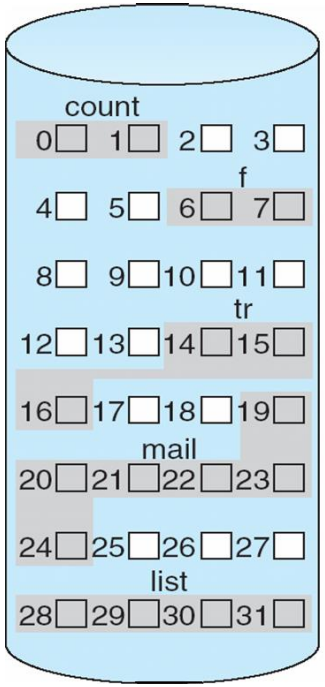
- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding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

Contiguous Allocation

- Mapping from logical to physical



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



directory

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

Extent-Based Systems

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

Allocation Methods - Linked

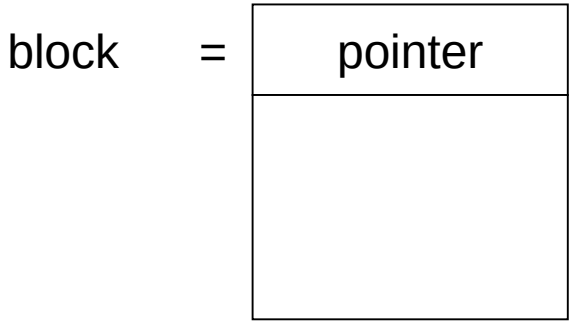
- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - No external fragmentation
  - Each block contains pointer to next block
  - No compaction, external fragmentation
  - Free space management system called when new block needed
  - Improve efficiency by clustering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

Allocation Methods – Linked (Cont.)

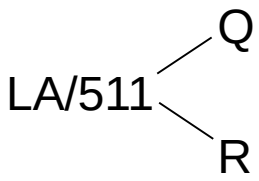
- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

Linked Allocation

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



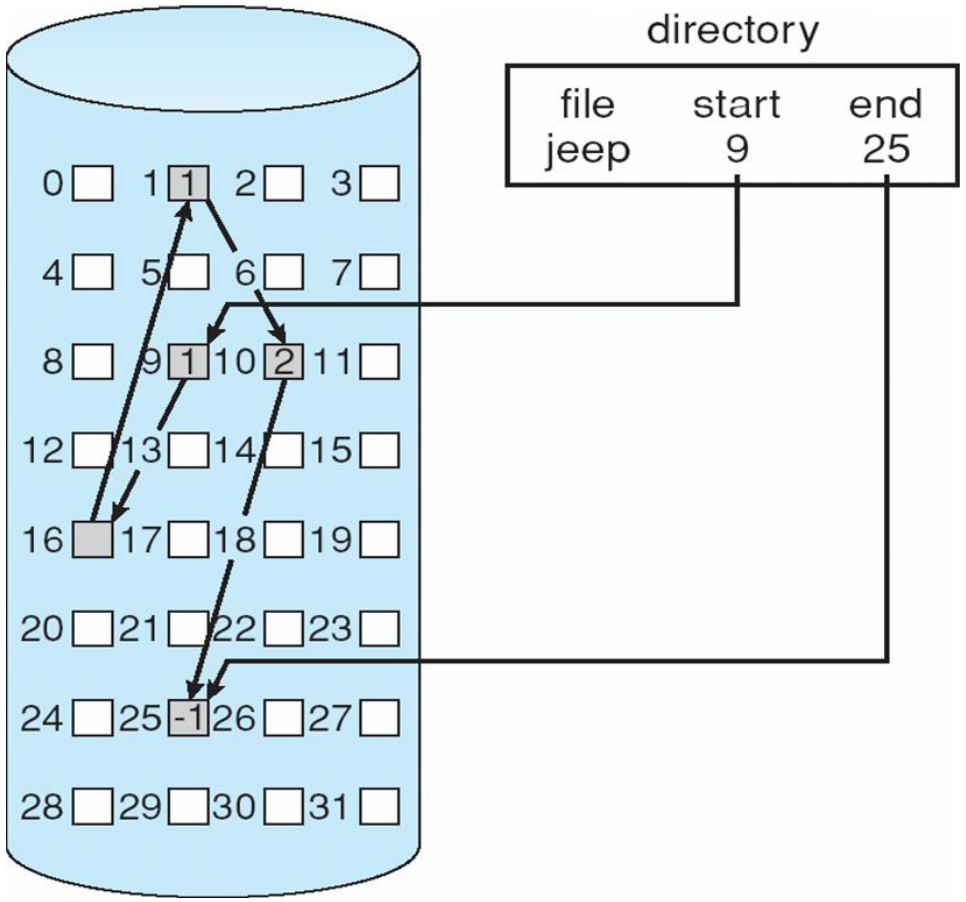
- Mapping



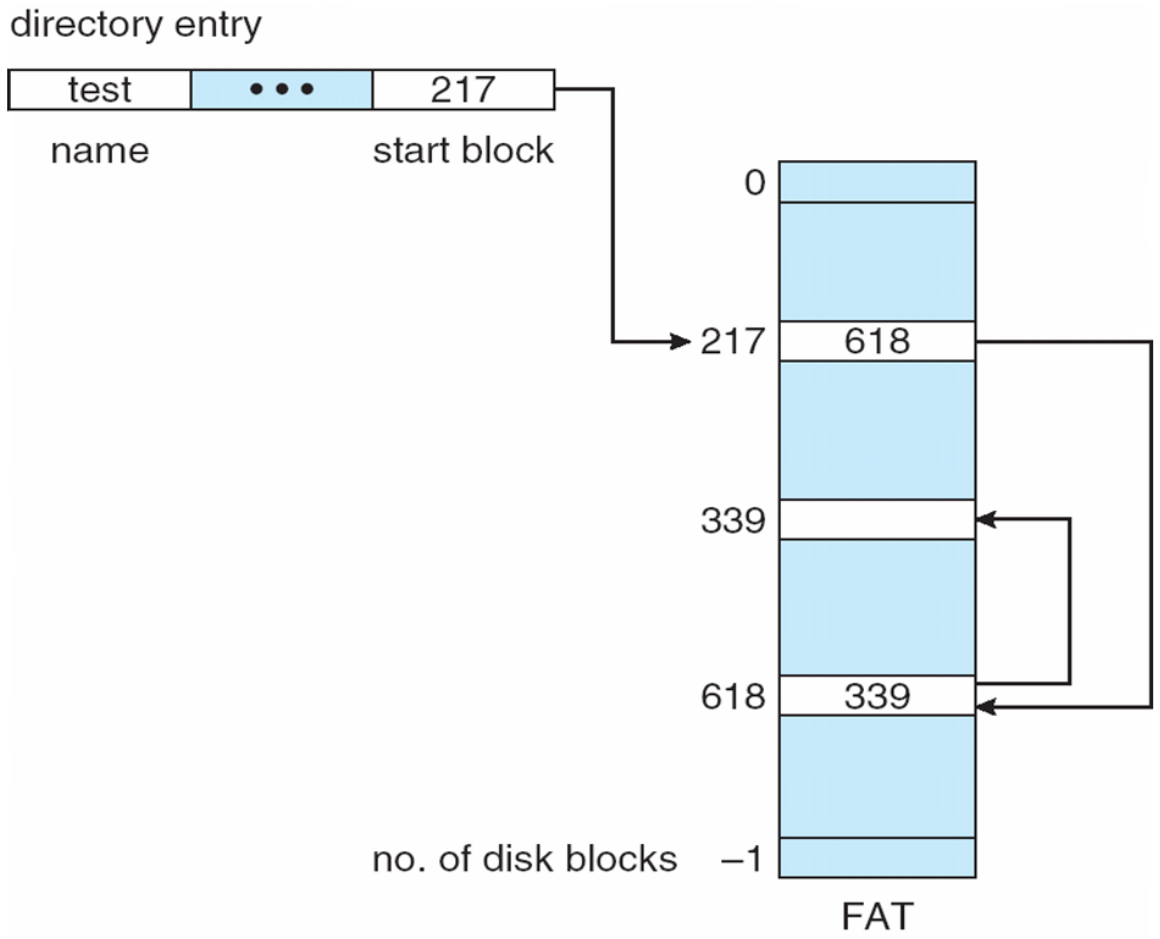
Block to be accessed is the Qth block in the linked chain of blocks representing the file.

Displacement into block = R + 1

Linked Allocation



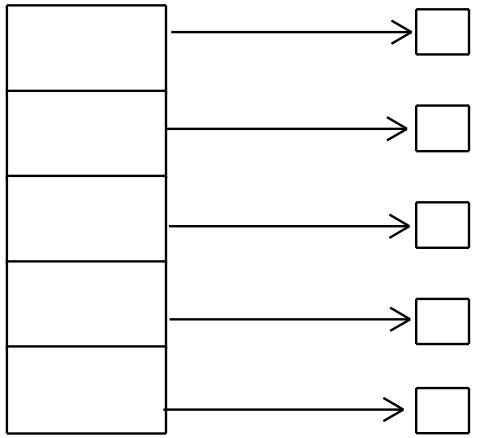
File-Allocation Table



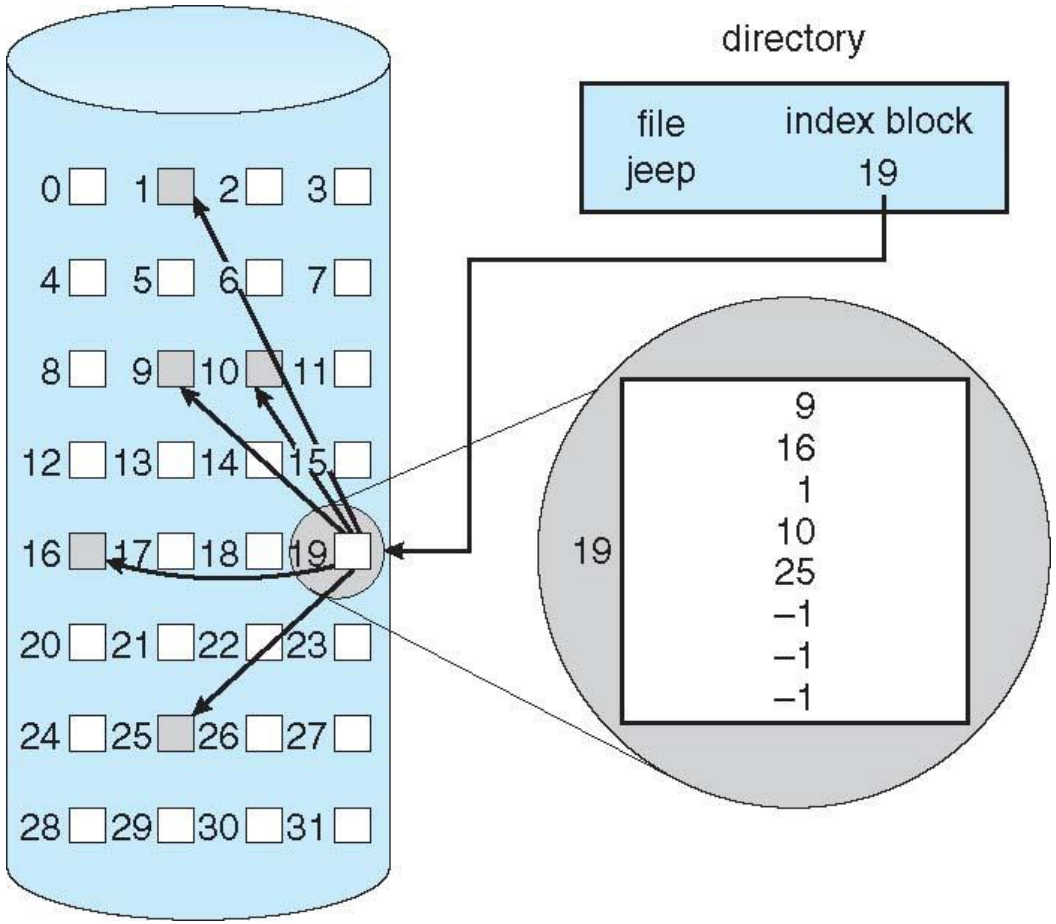
Allocation Methods - Indexed

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

- Logical view

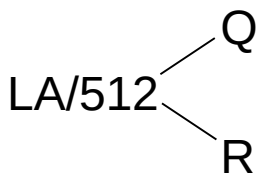


Example of Indexed Allocation



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 bytes and block size of 512 bytes. We need only 1 block for index table

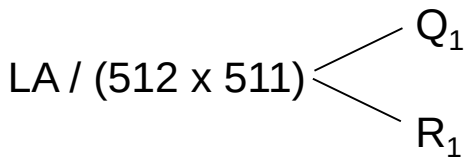


Q = displacement into index table
R = displacement into block

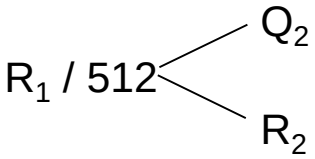
Indexed Allocation – Mapping (Cont.)

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

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



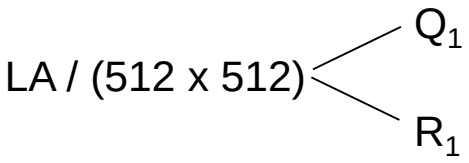
Q_1 = block of index table
R_1 is used as follows:



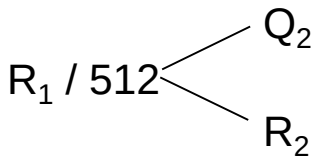
Q_2 = displacement into block of index table
R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)

- Two-level index (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)



Q_1 = displacement into outer-index
R_1 is used as follows:



Q_2 = displacement into block of index table
R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)