LOGISTICS

Exercise 6 released

One more exercise for file & storage system as bonus due during exam ban (2pt)

Final exam review April 7 – last class

STORAGE AND FILE SYSTEMS

RONG ZHENG

OUTLINE

Overview of file system & storage system

I/O, storage system

File systems

FILE ABSTRACTION

File System

I/O Sub system

Device Controller

Create, Open
Read, Write …
Ownership,
Access control

Read/write a
particular
location (sector)

FILE SYSTEMS – BROADLY DEFINED

File systems interface with storage devices (SSD, flash drive, magnetic disks, magnetic tapes …)

I/O control consists of device drivers and interrupt handler

- Device driver translates “retrieve block 123” into low-level hardware specific instructions

Basic file system: issue command to the device driver to read and write physical blocks on the disk

- Also manages memory buffers and caches
    - Buffers hold data in transit
    - Caches hold frequently used data (e.g. directory)

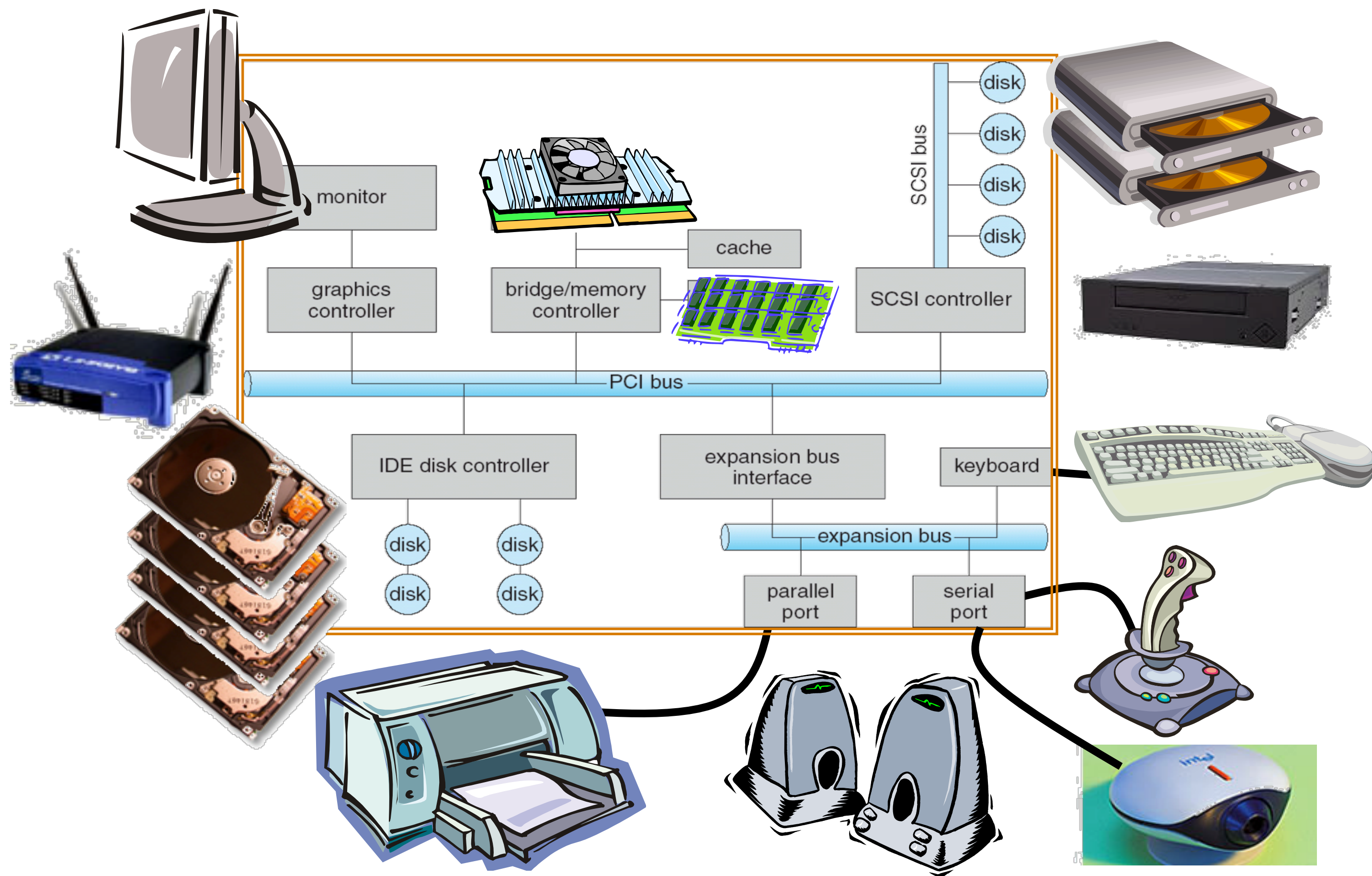
File organization module understands files, logical address, and how to map to physical blocks

- Translates logical block # to physical block #
- Manages free space, disk allocation

File management module: operations on files

- Naming: interface to find files by name, not by blocks; directories
- Protection: access control
- Reliability/Durability: keeping of files durable despite crashes, media failures, attacks, etc

I/O SYSTEM



ROLE OF I/O?

Without I/O, computers are useless (disembodied brains?)

But… thousands of devices, each slightly different

- How can we standardize the interfaces to these devices?

Devices unreliable: media failures and transmission errors

- How can we make them reliable???

Devices unpredictable and/or slow

- How can we manage them if we don't know what they will do or how they will perform?

Answer: abstraction, abstraction, abstraction!

OPERATIONAL PARAMETERS FOR I/O

Data granularity: Byte vs. Block

- Some devices provide single byte at a time (*e.g.,* keyboard)
- Others provide whole blocks (*e.g.,* disks, networks, etc.)

Access pattern: Sequential vs. Random

- Some devices must be accessed sequentially (*e.g.,* tape)
- Others can be accessed randomly (*e.g.,* disk, cd, etc.)

Transfer mechanism: Polling vs. Interrupts

- Some devices require continual monitoring
- Others generate interrupts when they need service

THE GOAL OF THE I/O SUBSYSTEM

Provide uniform interfaces, despite wide range of different devices

- This code works on many different devices:

```
FILE fd = fopen("/dev/something","rw");
for (int i = 0; i < 10; i++) {
  fprintf(fd, "Count %d\n",i);
}
close(fd);
```

- Why? Because code that controls devices ("device driver") implements standard interface

WANT STANDARD INTERFACES TO DEVICES

Block Devices: *e.g.,* disk drives, tape drives, DVD-ROM

- Access blocks of data
- Commands include `open()`, `read()`, `write()`, `seek()`
- Raw I/O or file-system access
- Memory-mapped file access possible

Character/Byte Devices: *e.g.,* keyboards, mice, serial ports, some USB devices

- Single characters at a time
- Commands include `get()`, `put()`
- Libraries layered on top allow line editing

Network Devices: *e.g.,* Ethernet, Wireless, Bluetooth

- Different enough from block/character to have own interface
- Unix and Windows include socket interface
  - Separates network protocol from network operation
  - Includes `select()` functionality

HOW DOES USER DEAL WITH TIMING?

Blocking Interface: "Wait"

- When request data (*e.g.*, `read()` system call), put process to sleep until data is ready
- When write data (*e.g.*, `write()` system call), put process to sleep until device is ready for data

Non-blocking Interface: "Don't Wait"

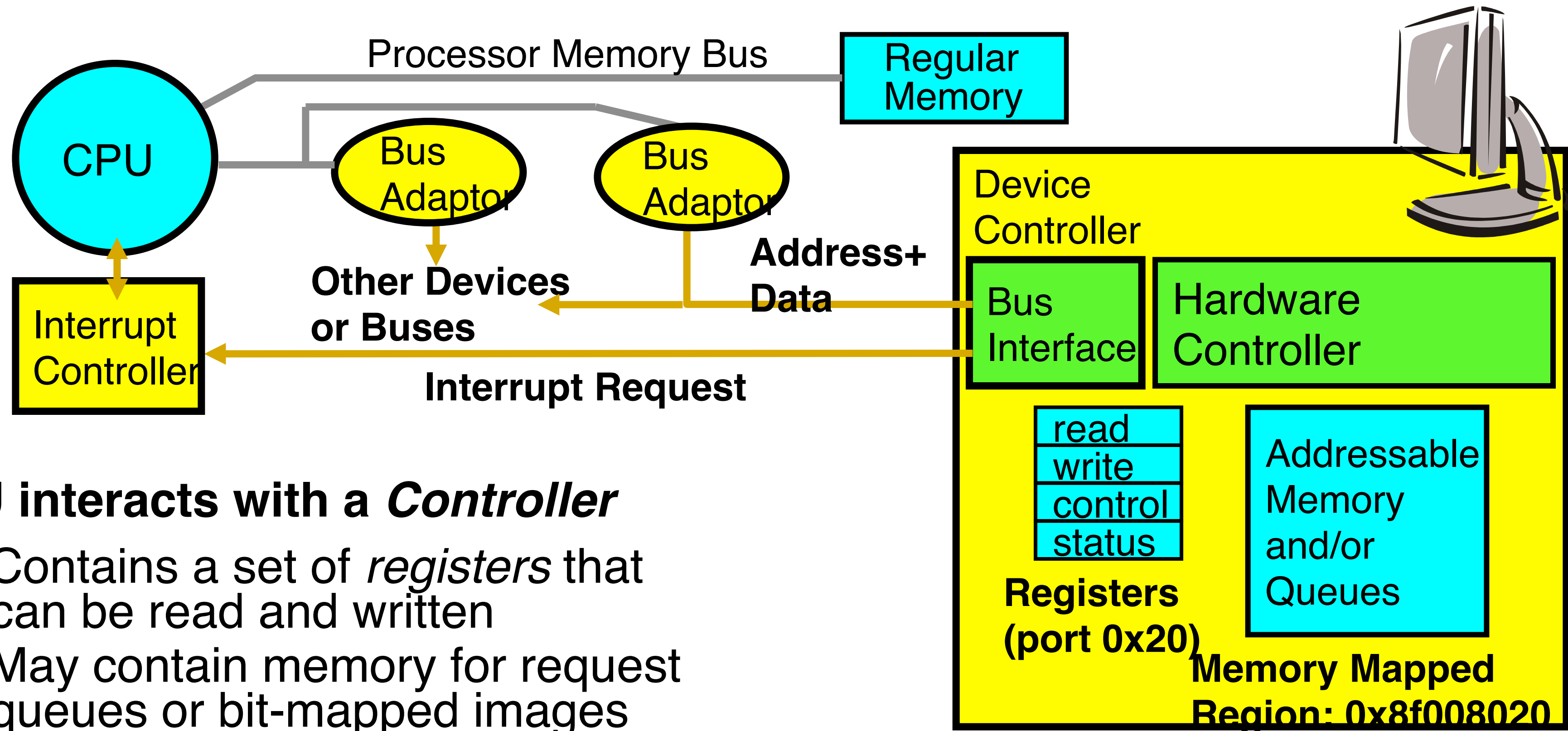
- Returns quickly from read or write request with count of bytes successfully transferred to kernel
- Read may return nothing, write may write nothing

Asynchronous Interface: "Tell Me Later"

- When requesting data, take pointer to user's buffer, return immediately; later kernel fills buffer and notifies user
- When sending data, take pointer to user's buffer, return immediately; later kernel takes data and notifies user

Be aware, in some context, non-blocking & asynchronous interface are used interchangeably

HOW DOES THE PROCESSOR TALK TO DEVICES?



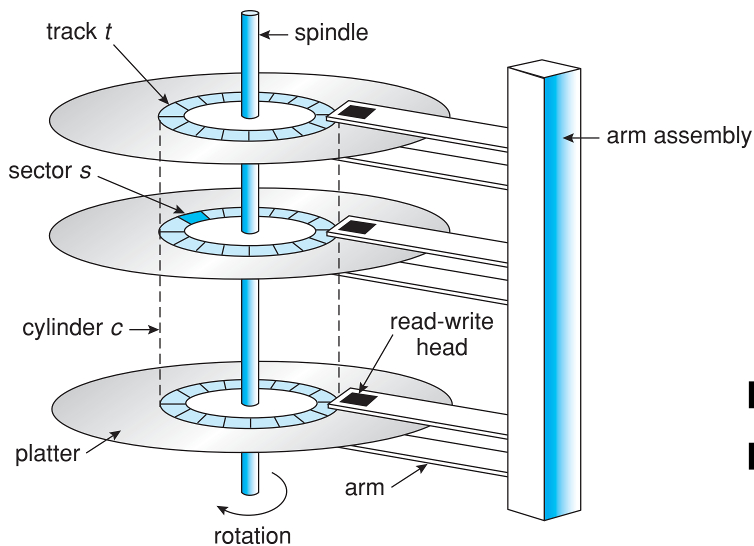
CPU interacts with a *Controller*

- Contains a set of *registers* that can be read and written
- May contain memory for request queues or bit-mapped images

Regardless of the complexity of the connections and buses, processor accesses registers in two ways:

- Port-mapped I/O: in/out instructions (e.g., Intel's 0x21,AL)
- Memory mapped I/O: load/store instructions
  - Registers/memory appear in physical address space
  - I/O accomplished with load and store instructions

SECONDARY STORAGE



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

MAGNETIC DISK PERFORMANCE

Capacity: GB – a few TB per drive

Transfer Rate – theoretical – 6 Gb/sec

Effective Transfer Rate – real – 1Gb/sec

Access Latency = Average access time = average seek time + average (rotational) 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 + 4.16ms + 0.1ms + transfer time
- Transfer time = 4KB / 1Gb/s * 8Gb / GB * 1GB / 10242KB = 32 / (10242) = 0.031 ms
- Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

Rotational speed [rpm]	Average latency [ms]
15,000	2
10,000	3
7,200	4.16
5,400	5.55
4,800	6.25

Seek time is large!

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

No moving parts, so *no seek time or rotational latency*

DISK ADDRESSING

Access disk as linear array of blocks. Two Options:

- Identify blocks as vectors [cylinder, surface, sector]. Sort in cylinder-major order. Not used much anymore.
- Logical Block Addressing (LBA). Every block has integer address from zero up to the max number.
    - A block of fixed size; typically consists of multiple sectors
- Disk controller translates from address $\Rightarrow$ physical position
    - First case: OS/BIOS must deal with bad blocks
    - Second case: hardware shields OS from structure of disk

Need way to track free disk blocks

- Link free blocks together $\Rightarrow$ too slow today
- Use bitmap to represent free space on disk

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

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 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

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

FCFS

First come first serve

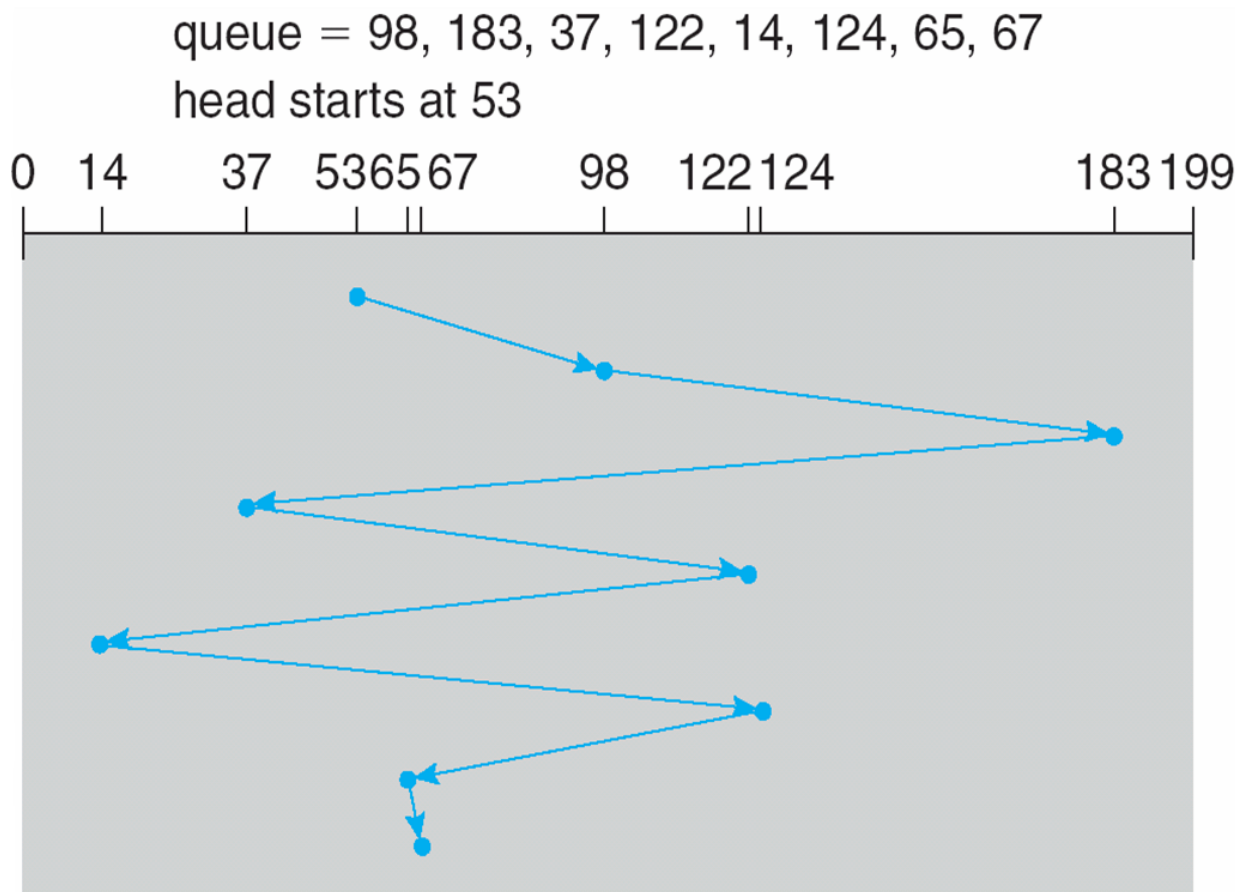


Illustration shows total head movement of 640 cylinders

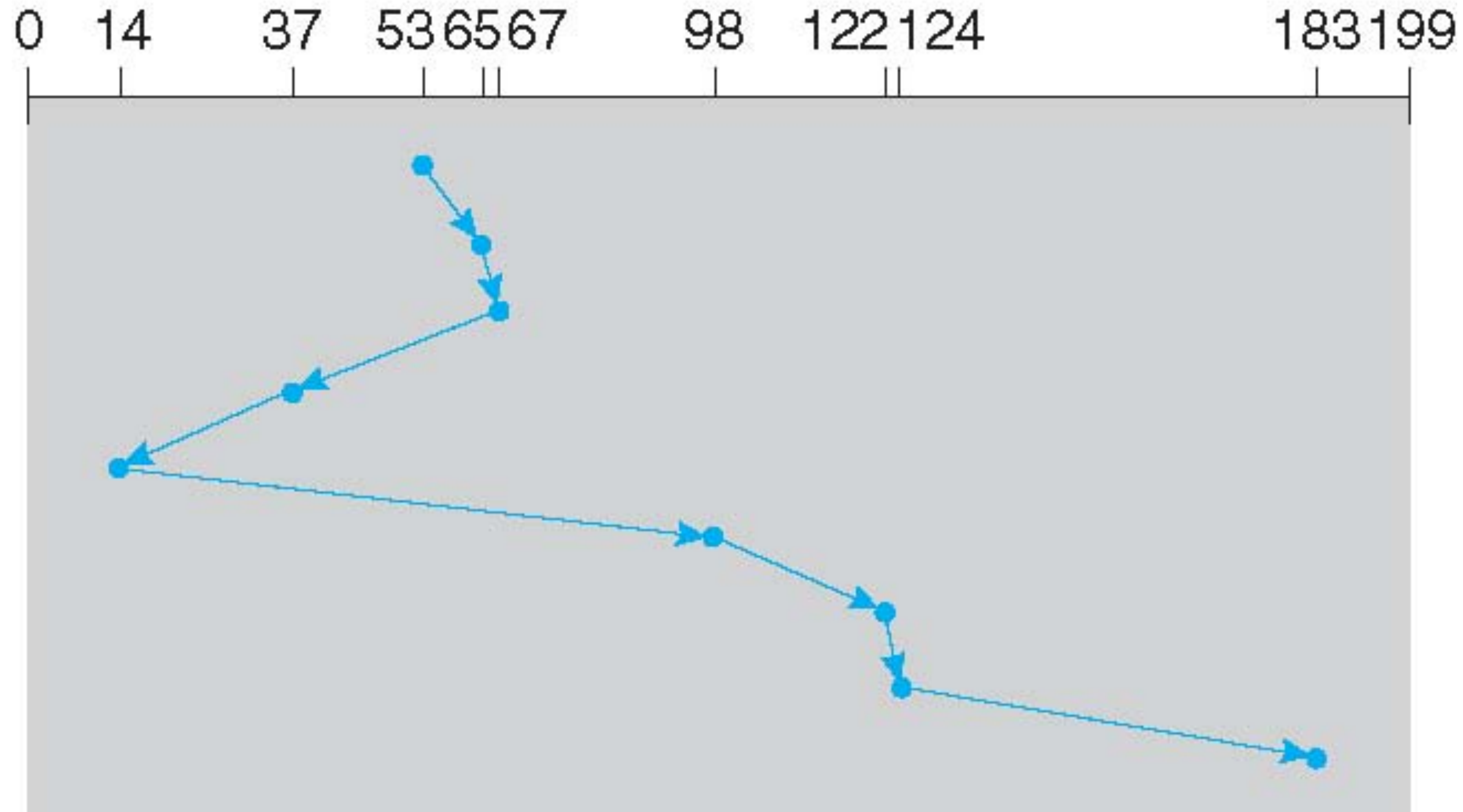
SSTF

Shortest Seek Time First selects the request with the minimum seek time from the current head position

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

Illustration shows total head movement of 236 cylinders

SSTF (CONT.)



SCAN

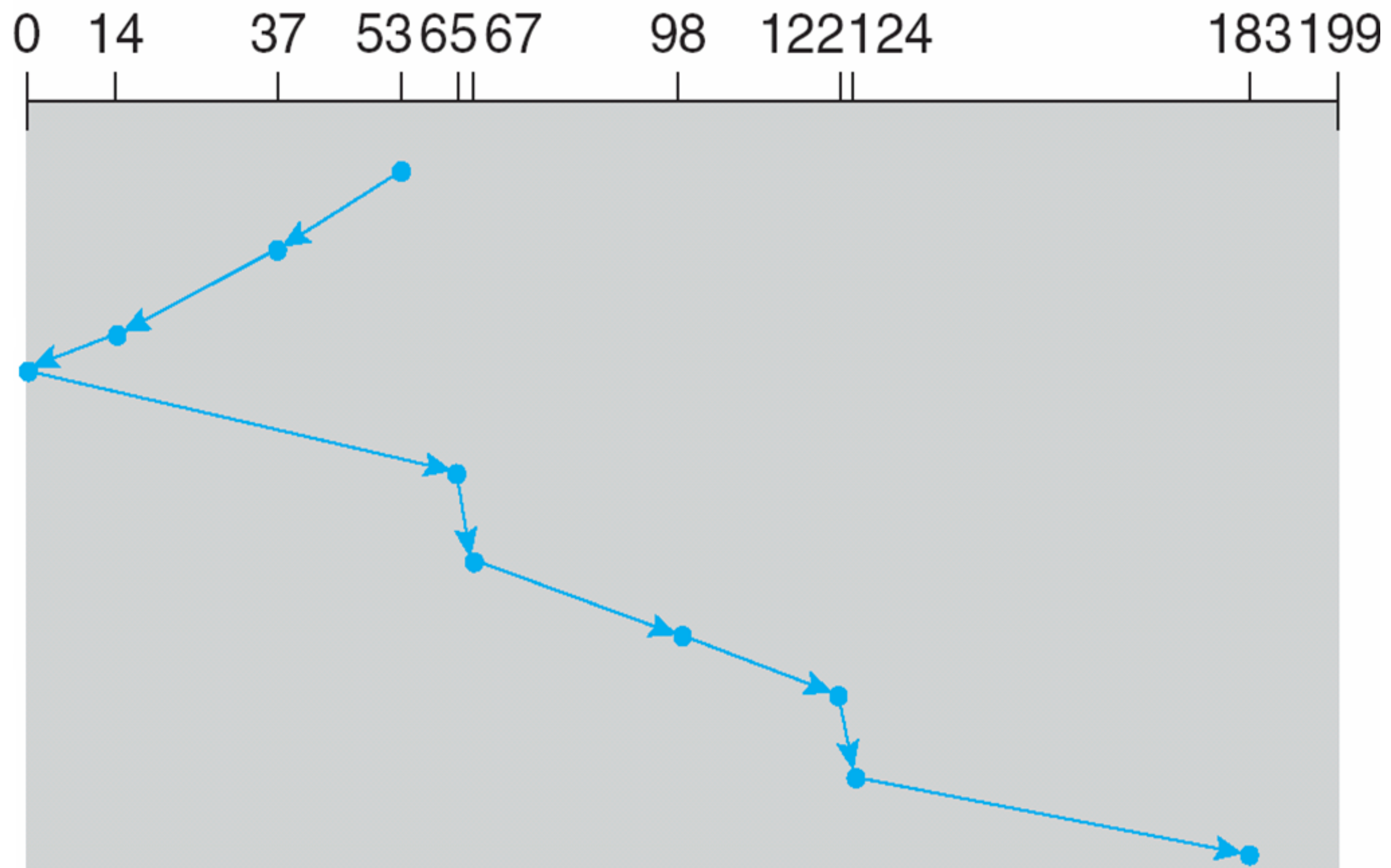
The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

SCAN algorithm Sometimes called the elevator algorithm

Illustration shows total head movement of 208 cylinders

But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

SCAN (CONT.)



C-SCAN

Provides a more uniform wait time than SCAN

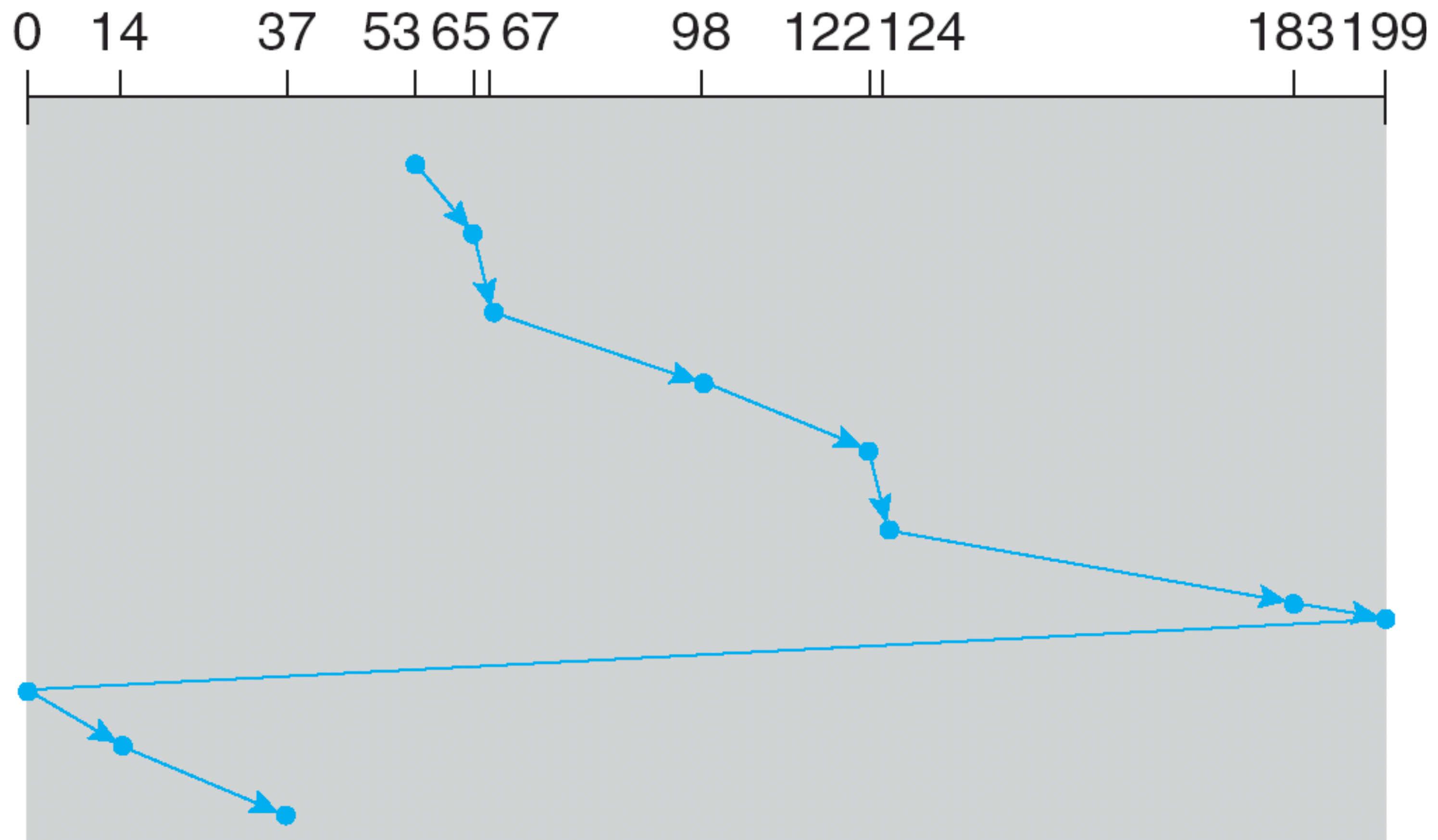
The head moves from one end of the disk to the other, servicing requests as it goes

- 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.)



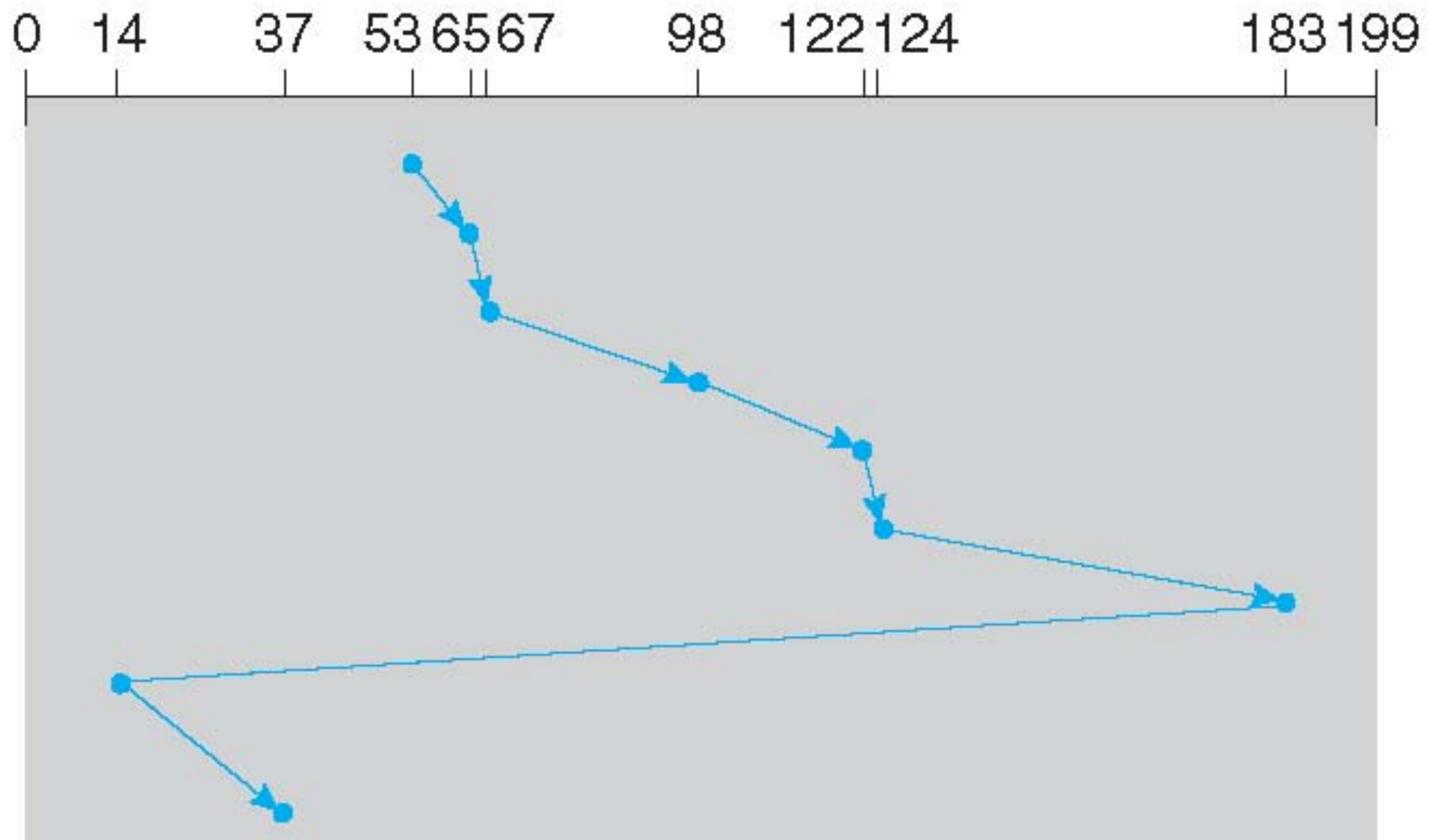
C-LOOK

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.)



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

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)
- ECC allows error detection and error recovery for small number of bit errors
- Usually 512 bytes of data but can be selectable: 256, 512, 1024

DISK MANAGEMENT (CONT'D)

To use a disk to hold files, the operating system still needs to record its own data structures on the disk

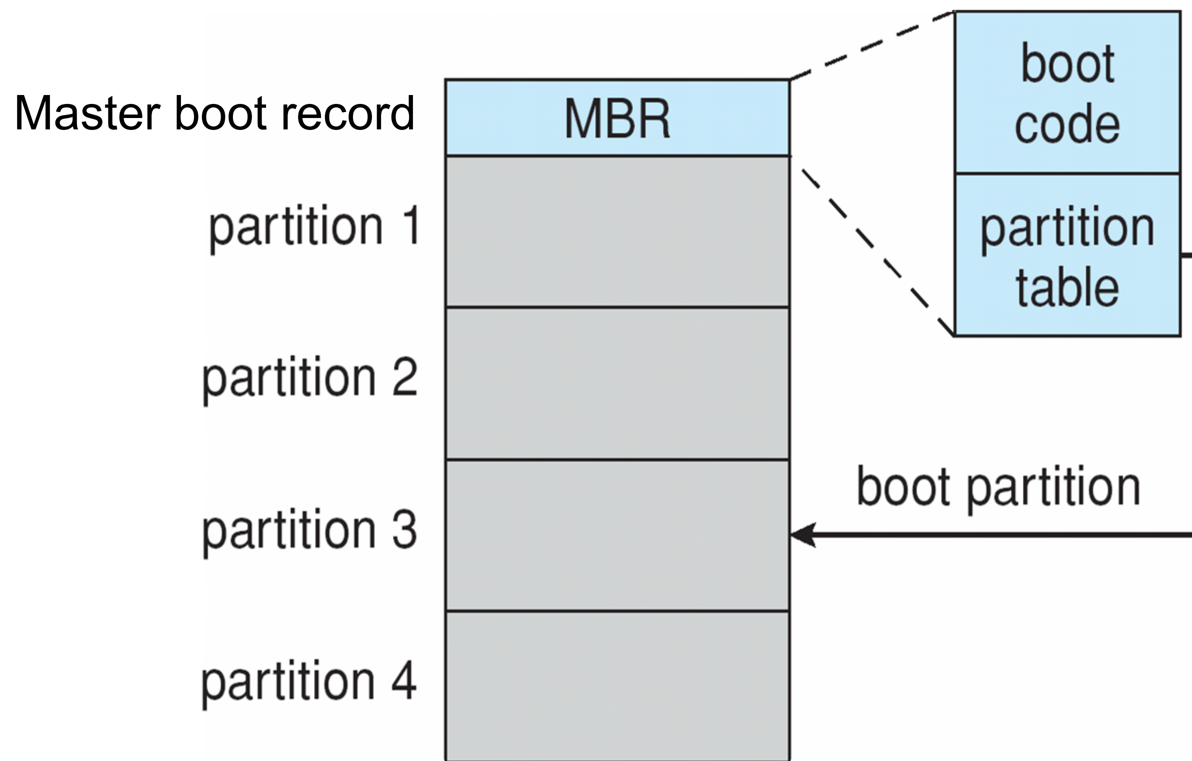
- Partition the disk into one or more groups of cylinders, each treated as a logical disk
- Logical formatting or "making a file system"
  - e.g., store maps of free and allocated space and an initial empty directory

Raw disk access for apps that want to do their own block management, keep OS out of the way (e.g., databases)

BOOTING FROM A DISK IN WINDOWS

Boot block initializes system

- The bootstrap is stored in ROM
- Bootstrap loader program stored in boot blocks (e.g., first section) of the hard disk



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 (as raw disk)

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

What if a system runs out of swap space?

- Some systems allow multiple swap spaces

RAID STRUCTURE

RAID – redundant array of inexpensive (independent) disks

- multiple disk drives provides reliability via redundancy

Terminology:

- Mean time to failure (MTF): the average time to failure (w/o repair)
- Mean time to repair (MTR): the average time to repair a failed component
- Mean time to data loss (MTDL): if failures occur during the repairing, data losses may occur
- Data mirroring: duplicating the same data on multiple disks
- Data striping: segmenting logically sequential data so that consecutive segments are stored on different physical storage devices: bit striping, block striping …

EXAMPLE

MTF of a single disk is 100,000 hrs

What about MTF for some disk among 100 disks?

- 1000 hrs ~ 41.66 days
- Needs redundancy!

For a single disk, MTDL = MTF

Consider 2-mirroring, the MTF of a single disk fail is thus 50,000 hrs (why?)

Let the MTR be 10 hrs.

- If 1 disk fails, while it is being repaired, another failure occurs to the remaining disk → data loss
- What is the MTDL for 2-mirroring?

MTDL FOR 2-MIRRORING

the probability that the second disk fails while the first disk is being repaired is 10/100,000

⇒ The probability of failure of both disks 1/50000x10/100,000

⇒ The MTDL = $100{,}000^2/2/10 = 5 \times 10^8$ hrs = 57,000 yrs!

(holds only for independent failures and MTR << MTF)

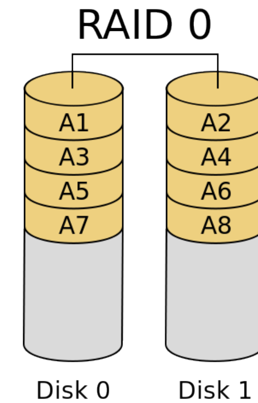
Generally, for two-mirroring

MTDL = $MTF^2/2/MTR$

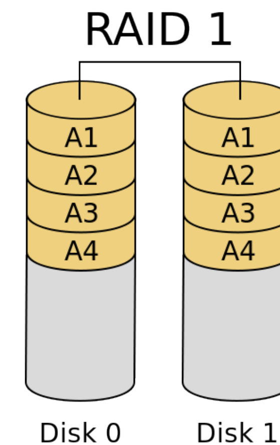
RAID LEVELS

RAID level 0: disk arrays with striping at the level of blocks but without any redundancy (such as mirroring or parity bits

- Increase the performance of disk I/O



RAID level 1: disk mirroring



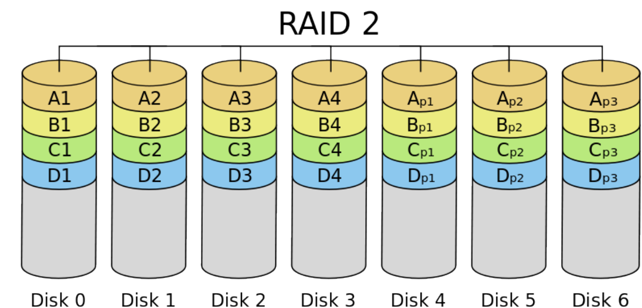
RAID level 2: striping at bit level; using hamming code for error correction

- **(7, 4) in the example**

(1 0 1 1) → (p1 p2 1 p3 0 1 1) → (0 1 1 0 0 1 1)

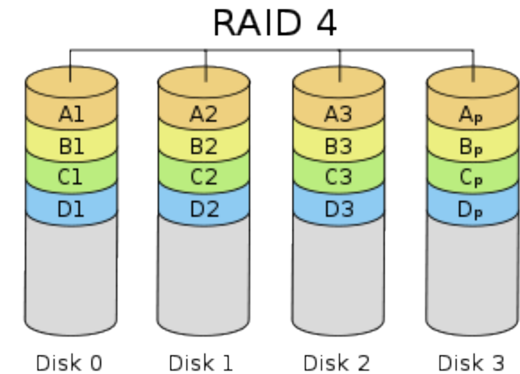
What happens in case of one bit error?

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

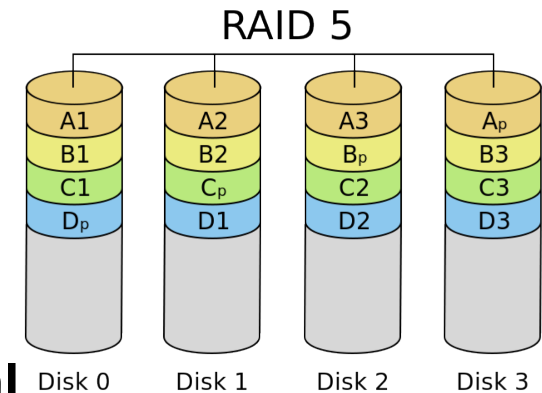


RAID LEVELS (CONT'D)

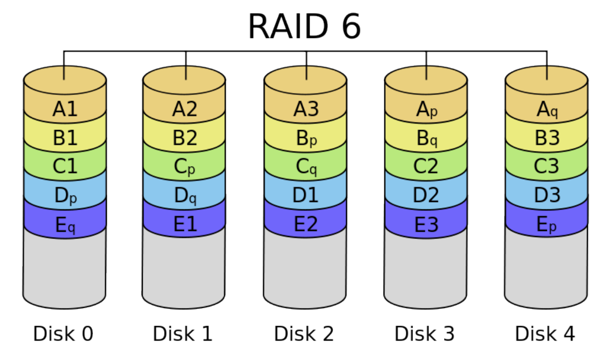
RAID 4: block level striping with parity disk



RAID 5: block level striping with distributed parity



RAID 6: extends RAID 5 by adding an additional parity block; thus it uses block-level striping with two parity blocks distributed across all member disks.



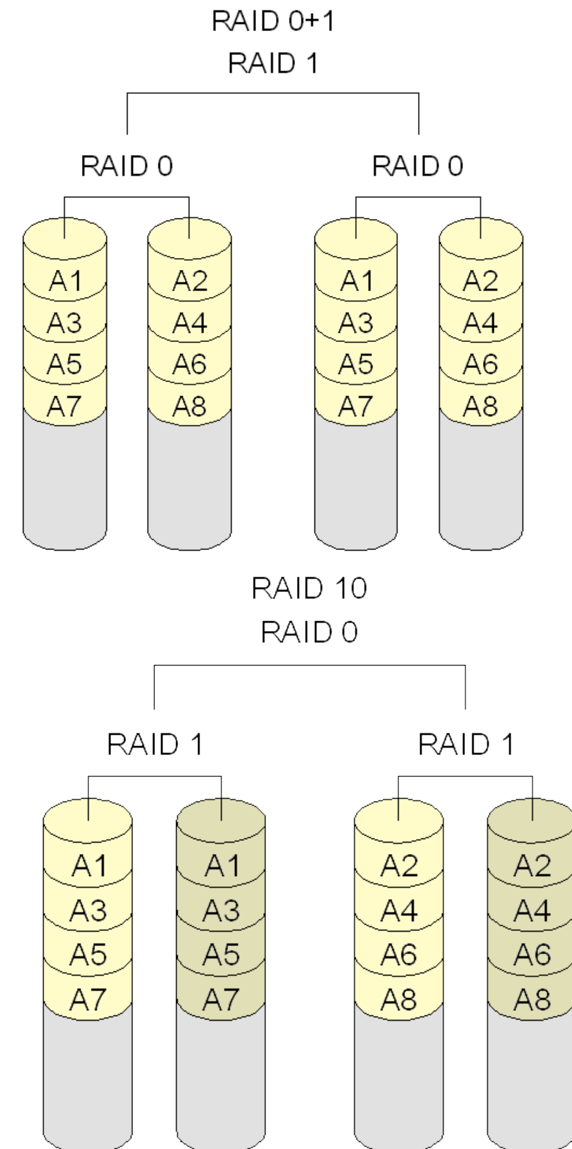
NESTED RAID

RAID 0 + 1: striping then mirroring

RAID 1 + 0: mirroring then striping

Any difference?

- Both can tolerate single disk failures
- Same capacity requirements
- In practice, may have different fault tolerance capabilities: RAID10 preferable



RAID (CONT.)

Trade-off in reliability and redundancy (cost)

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

FILE

Contiguous logical address space
Types:

- Data
  - numeric
  - character
  - binary
- Program

Contents defined by file’s creator

Analogous to processes

- Contiguous logical space
- PCB
- Page table

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 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(Fi) – search the directory structure on disk for entry Fi, and move the content of entry to memory

Close (Fi) – move the content of entry Fi 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

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

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

FILE-CONTROL BLOCK

File control block stores file meta data

- **Stored on disks when a file is not open**
- **Load into memory for open files**

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

HOW ARE FILES STORED?

File is an abstract data type

- FCB
- data blocks

Allocation methods decide how disk blocks are allocated for files

- Analogous to memory address translation
- Contiguous allocation – each file occupies set of contiguous blocks
- Linked allocation – each file a linked list of blocks (FAT)
- Indexed allocation – each file has its own index block(s) of pointers to its data blocks

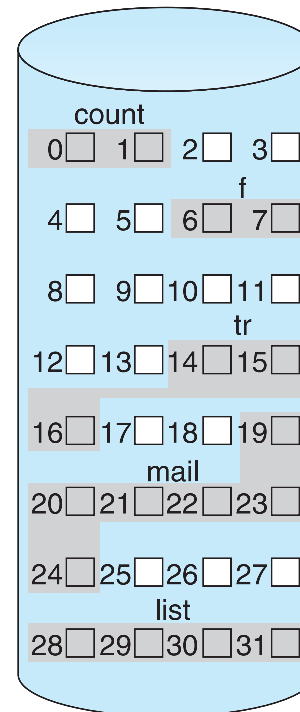
CONTIGUOUS ALLOCATION

A file of size n is stored in block b, b+1, b+2, …, b+n

Advantage:

- reduce access time for sequential accesses
- Easy to support direct access

Disadvantage: external fragmentation



directory

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

LINKED ALLOCATION

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

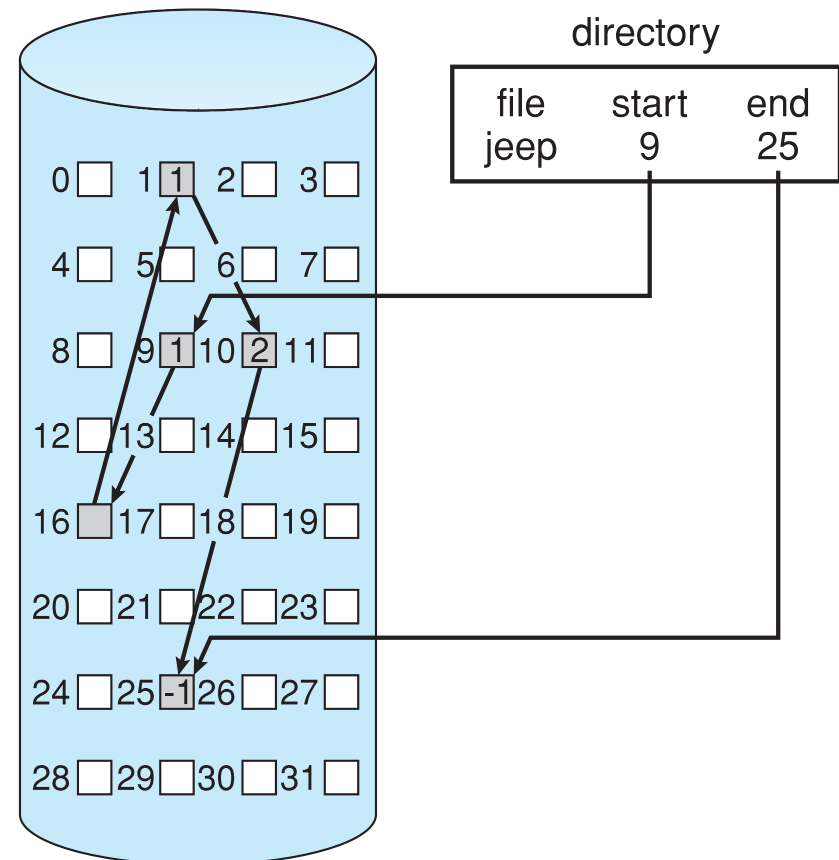
Directory contains the pointer to the first and last blocks

Each block contains a point to the next block

Advantage: no external fragmentation

Disadvantage:

- Not suitable for direct access
- Additional space required for storing the pointers



FILE ALLOCATION TABLE (FAT)

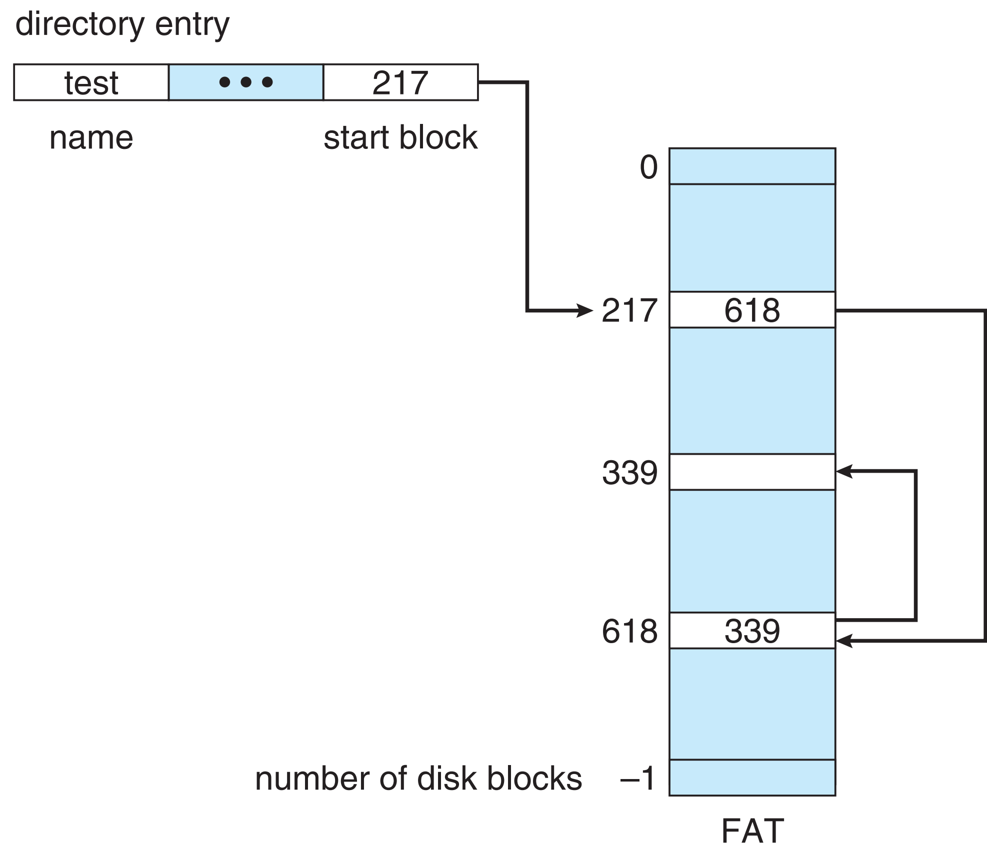
Store the linked list in a table

A section of disk at the beginning of each volume is set aside to contain the table.

The table has one entry for each disk block and is indexed by block number.

The table entry indexed by that block number contains the block number of the next block in the file

An unused block is indicated by a table value of 0



INDEXED ALLOCATION

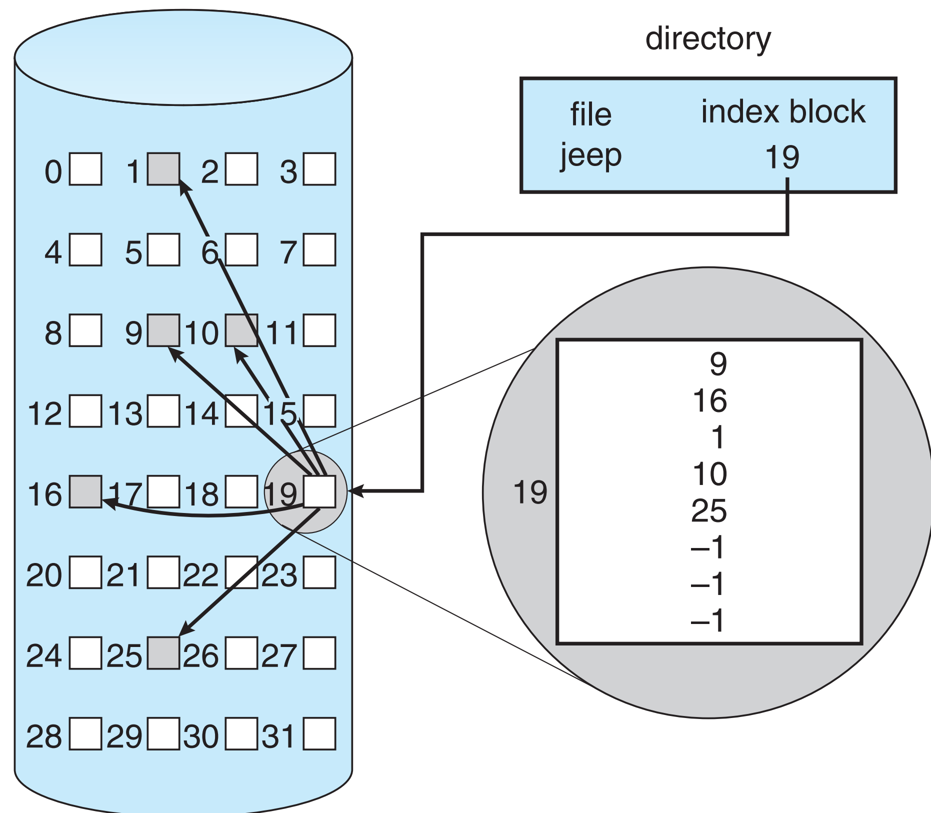
Each file has its own **index block(s)** of pointers to its data blocks

The index blocks are stored on disk

What does this remind you of?

- Page table

File size limit?



VARIANTS OF INDEXED ALLOCATION

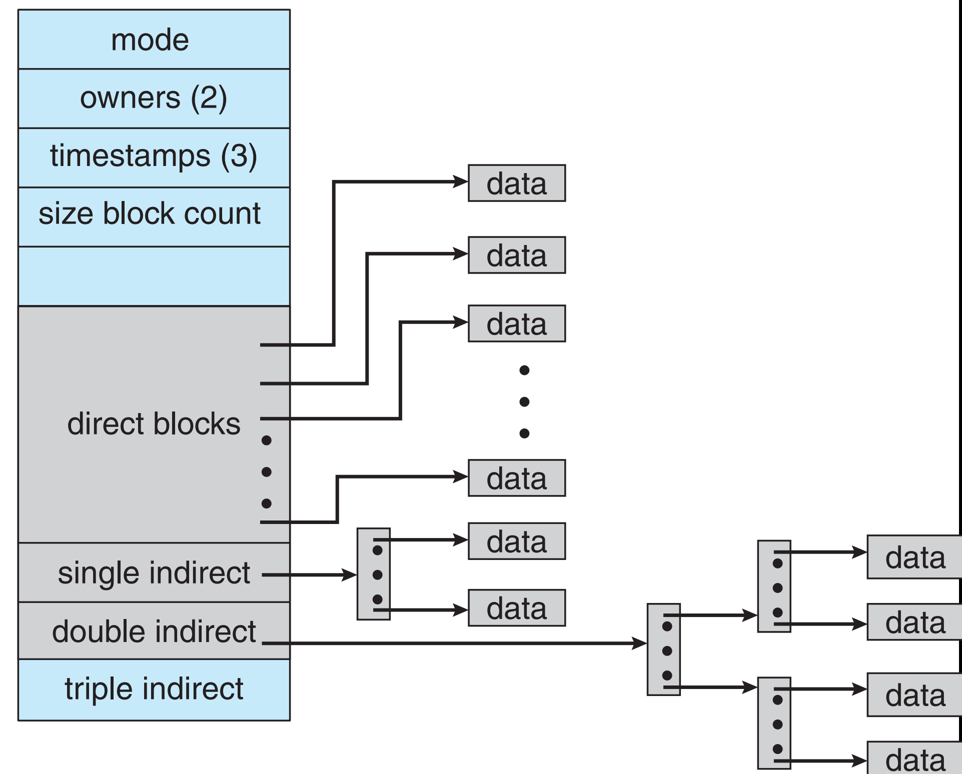
Linked index blocks

Multilevel index

- **1-level index: 4KB/block, 4-byte pointers → 4MB**
- **2-level index: 4KB/block, 4 type points → 1Kx1Kx4KB = 4GB file size**

Combined scheme

- **used in Unix**
- **In UFS: 12 direct pointers for blocks, 1 singly indirect, 1 double indirect and 1 triply indirect (15 altogether)**
- **4KB blocks: 12*4KB + 1KB*4KB + 1KB*1KB*4KB + 1KB*1KB*1KB*4KB ~ 4TB (assume 32-bit pointers)**
- **Using 32-bit pointer → maximally addressable # of blocks = 2^{32} → 16TB**



DIRECTORY

What is directory?

- Groups files into collections
- Can be implemented as a file (Unix)
- Store file related information (e.g., index block pointer)

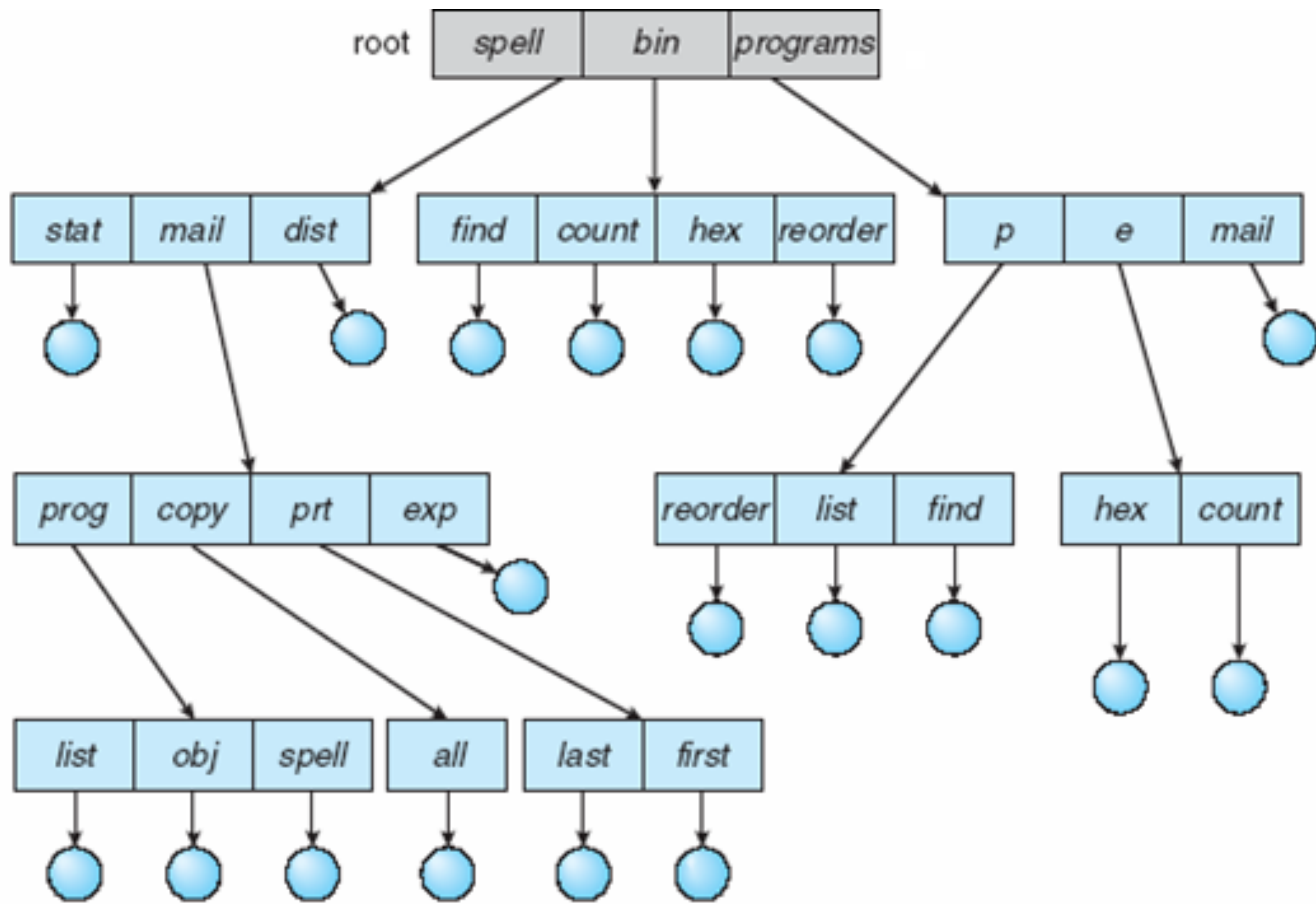
Operations performed on a directory

- Search for a file, create a file, delete a file, list a directory, rename a file, traverse the file system

Logical organization of directories

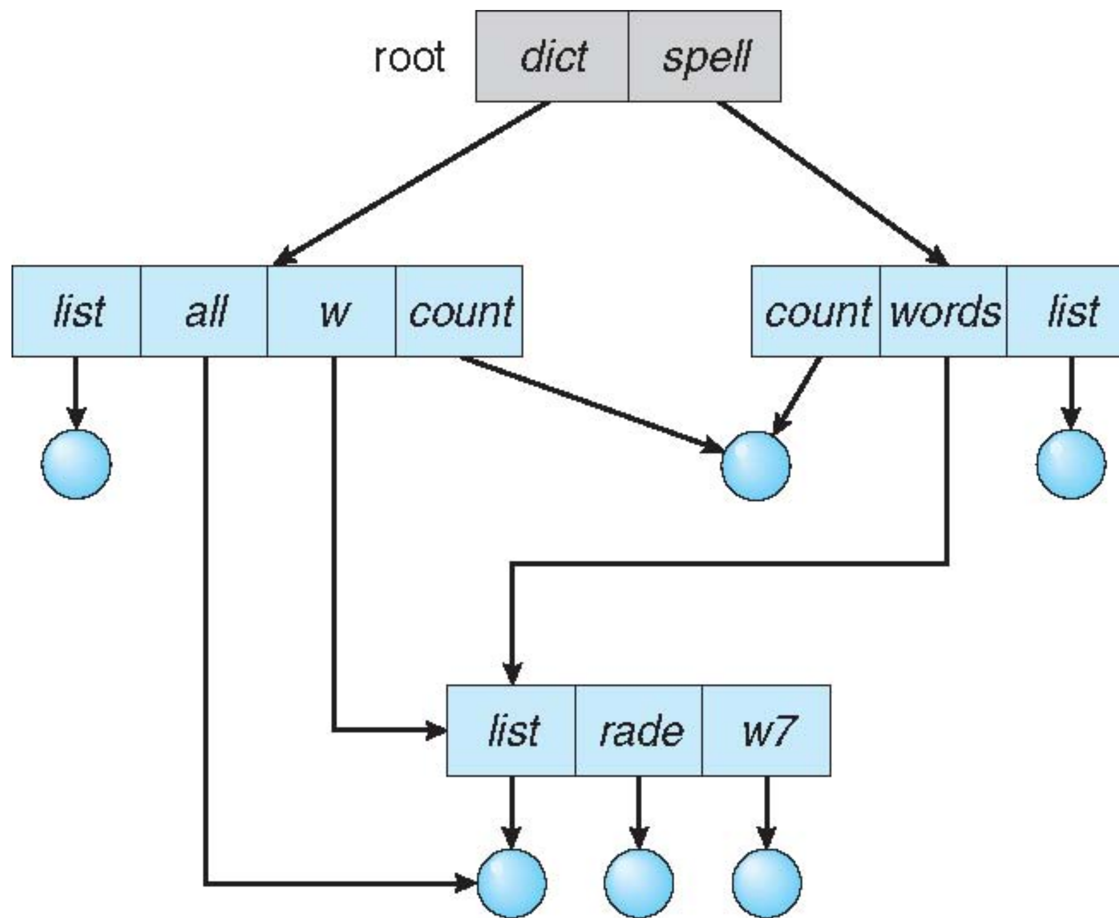
- **Single level (flat fs)**
- **Two level**
- **Tree-Structured directories**
- **Acyclic graph**

TREE-STRUCTURED DIRECTORY



ACYCLIC-GRAPH DIRECTORIES

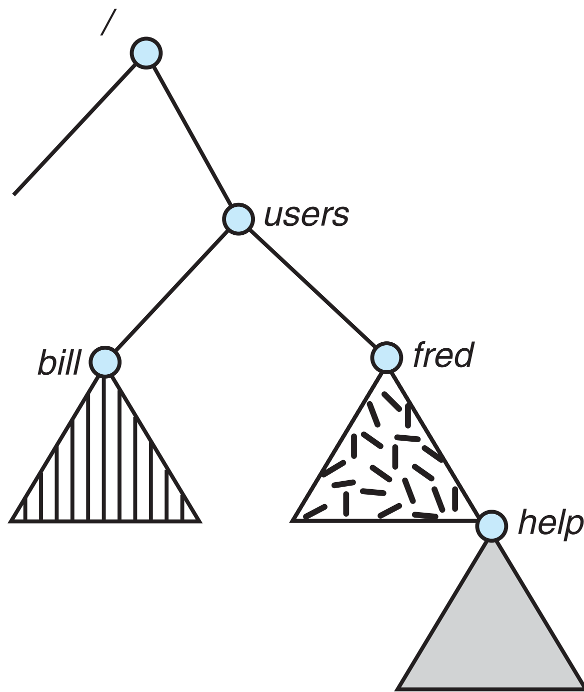
Have shared



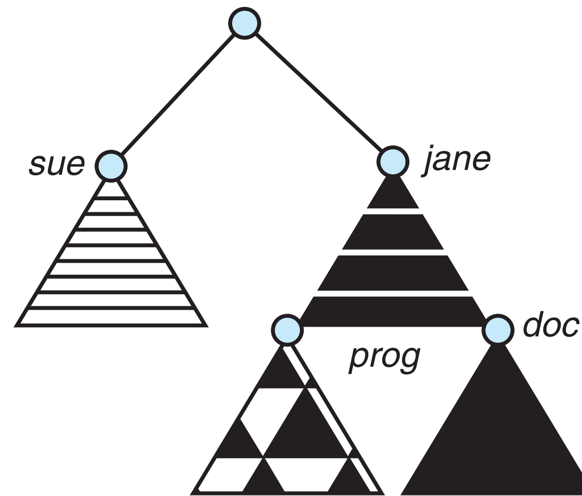
FILE SYSTEM MOUNTING

A file system must be mounted before it can be accessed

A unmounted file system is mounted at a mount point

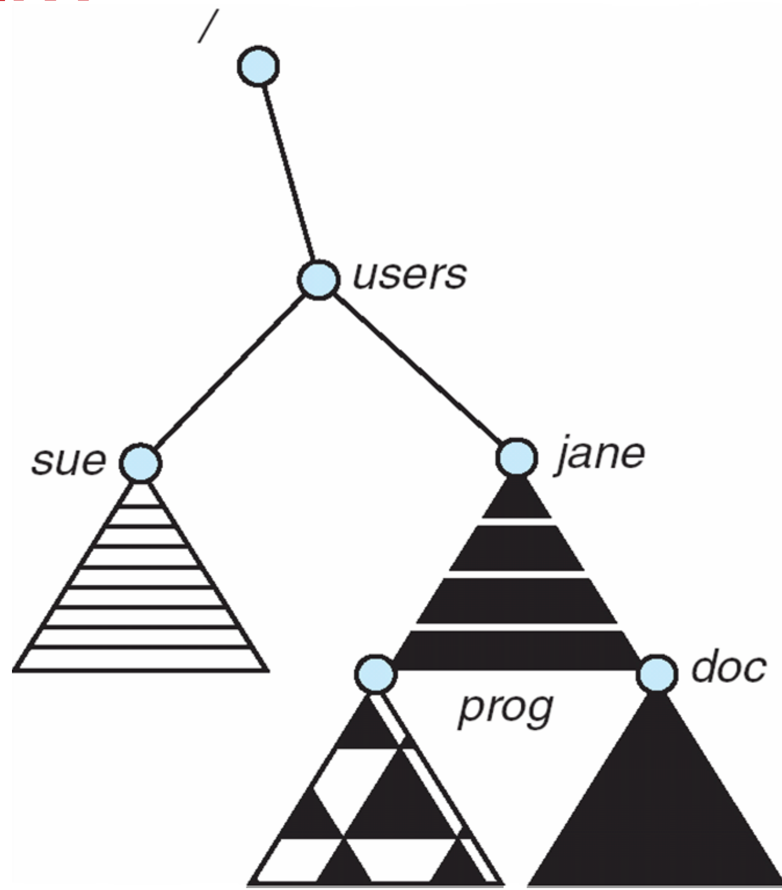


(a)



(b)

MOUNT POINT



IMPLEMENTATION OF DIRECTORY

Recently accessed directories can be cached in memory

1) Linear list of file names with pointers to the data blocks

- **File creation:**
  - 1) search no existing file with the same name [linear search]
  - 2) Add new entry at the end of the directory
- **File deletion:**
  - 1) Search for the respective entry
  - 2) Release the space (mark as unused, attach a free list, move the last entry over …)

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

DATA STRUCTURES OF FILE SYSTEMS

Both on-disk and in memory

On-disk structures

- Boot control block contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- Volume control block (superblock, master file table) contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table
- Per-file File Control Block (FCB) contains many details about the file
  - inode number, permissions, size, dates

IN-MEMORY FILE SYSTEM STRUCTURES

Mount table storing file system mounts, mount points, file system types

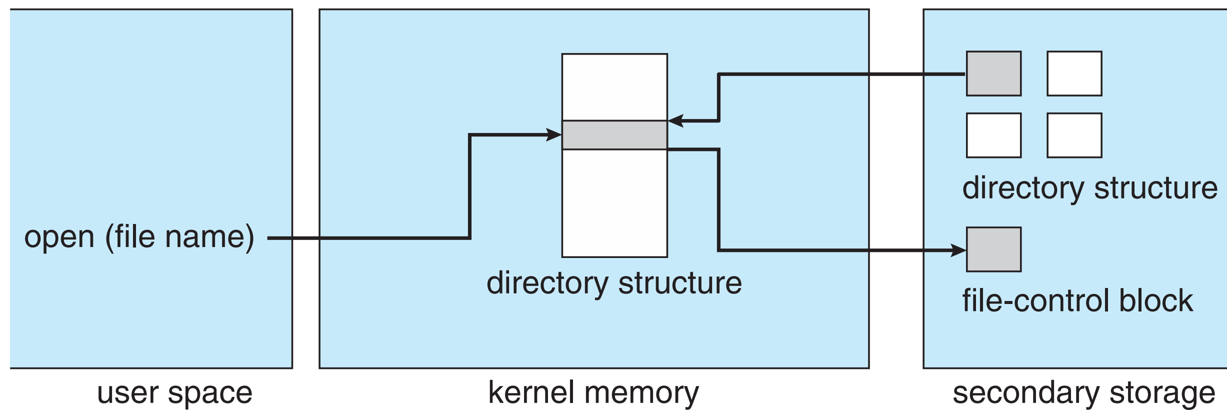
An in-memory directory-structure cache holds the directory information of recently accessed directories

The system-wide open-file table contains a copy of the FCB of each open file, as well as other information

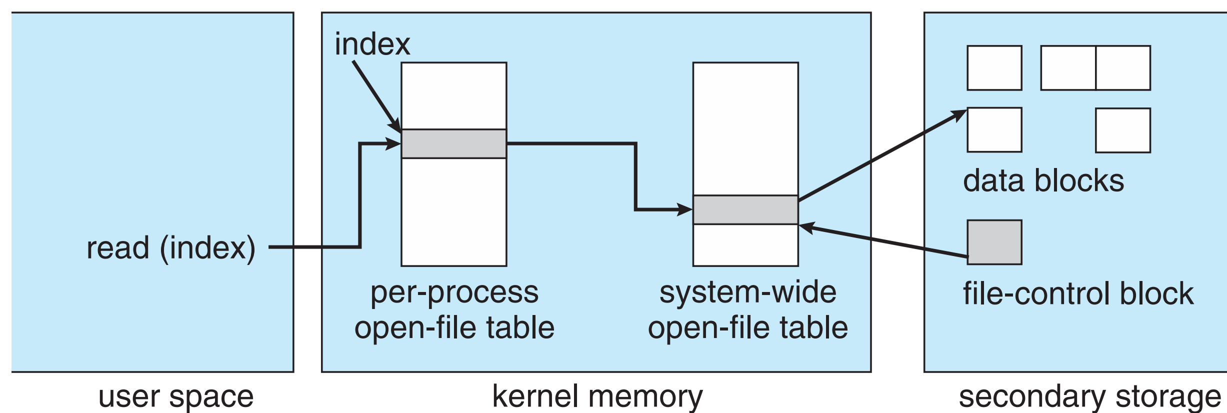
The per-process open file table contains a pointer to the appropriate entry in the system-wide open-file table, as well as other information

Buffers hold file-system blocks when they are being read from disk or written to disk

IN MEMORY FILE-SYSTEM STRUCTURES



File open



File read

SUMMARY