OPERATING SYSTEMS

Storage Management

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Mass-Storage Structure

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

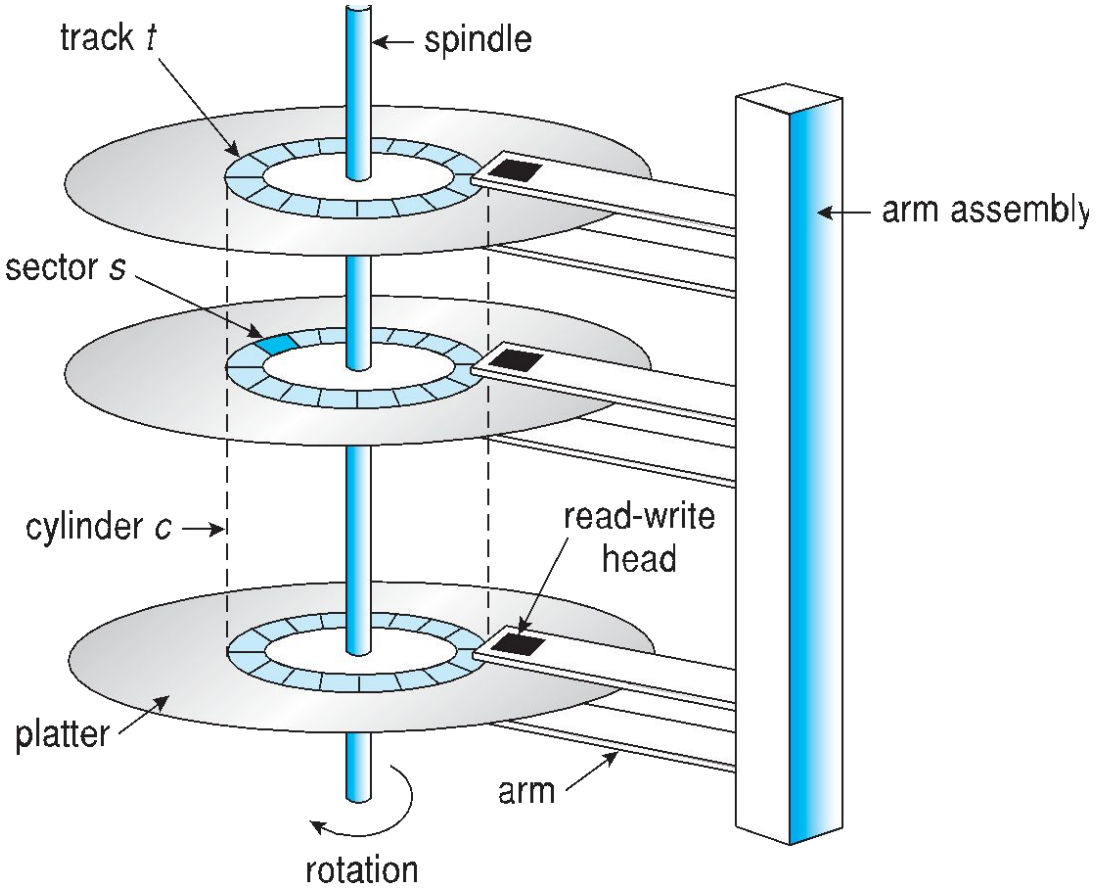
1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

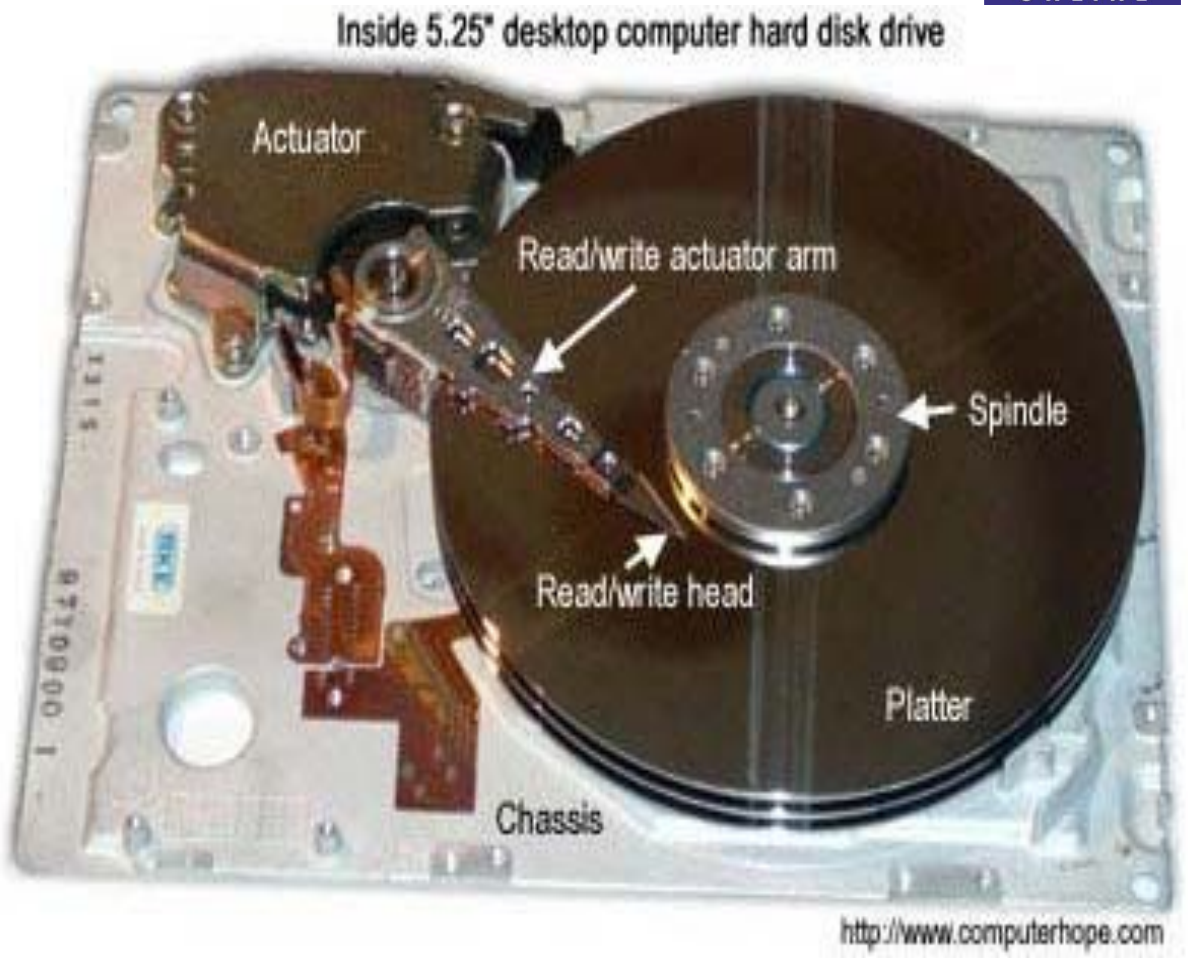
Overview of Mass Storage Structure

- **Magnetic disks** provide bulk of secondary storage of modern computers.
- Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 3.5 inches.
- The two surfaces of a platter are covered with a magnetic material.
- The heads are attached to a disk arm that moves all the heads as a unit.
- The surface of a platter is logically divided into circular tracks, which are subdivided into sectors.
- The set of tracks that are at one arm position makes up a cylinder.

OPERATING SYSTEMS

Moving-head Disk Mechanism





Overview of Mass Storage Structure

- When the disk is in use, a drive motor spins it at high speed. Most drives rotate 60 to 250 times per second.

Disk speed has two parts.

- The transfer rate is the rate at which data flow between the drive and the computer.
- The positioning time, or random-access time, consists of two parts
  - The time necessary to move the disk arm to the desired cylinder, called the **seek time**.
  - the time necessary for the desired sector to rotate to the disk head, called the **rotational latency**.

Disk Latency = Seek time + Rotational latency + Transfer time

OPERATING SYSTEMS

Magnetic Disk Specifications – Reference: www.seagate.com

Specifications	Barracuda®	Constellation®	Constellation ES	SV35 Series™
Primary Applications	Optimised for PC and personal external storage	Optimised for 2.5-inch business server and external storage arrays	Optimised for 3.5-inch business server and external storage arrays	Optimised for video surveillance applications
Capacity (GB)	250, 320, 500, 750, 1,000 1,500, 2,000, 3,000	250, 500, 1,000	500, 1,000, 2,000, 3,000	1,000, 2,000, 3,000
Spin Speed (RPM)	7,200	7,200	7,200	7,200
SATA interface (Gb/s)	1.5/3.0/6.0	1.5/3.0/6.0	1.5/3.0/6.0	1.5/3.0/6.0
SAS interface (Gb/s)	—	3.0/6.0	3.0/6.0	—
Rotational vibration (RV) (radians/s/s)	5.5 narrow spectrum up to 300Hz	16 broad spectrum up to 1,800Hz	12.5 broad spectrum up to 1,500Hz	5.5 narrow spectrum up to 300Hz
Seek time, average read/write (ms)	<8.5/<9.5	8.5/9.5	8.5/9.5	<8.5/<9.5
Cache (MB)[1]	16, 64	Up to 64	Up to 64	64
Non-recoverable read errors per bits read	1 sector per 10^{14}	1 sector per 10^{15}	1 sector per 10^{15}	1 sector per 10^{14}
Power-on hours (POH)	2,400 – 8x5	8,760 - 24x7	8,760 - 24x7	8,760- 24x7
Streaming capabilities	—	Multiple sequential streams	Multiple sequential streams	Up to 20 simultaneous HD streams[3]
POH usage profile	8x5 – On as needed	24x7 – Always on	24x7 – Always on	Up to 64 cameras 24x7 – Always on
MTBF (hours)	700,000	1.4 million	1.2 million	1 million
Power, average – idle (W)[2]	4.6	2.25 to 3.85	>3.74	—
Power, average – Idle2 (W)[2]	3.4 to 5.4	—	—	3.4 to 5.4
Acoustics, typical – idling (bels)	2.2 to 2.4	2.2	1.9 to 2.7	2.2 to 2.4
Shock, operating/non-operating (Gs)	70 to 80/300 to 350	70/400	40 to 70/300	80/300 to 350
Ambient temperature, operating/ non-operarting (°C)	0 to 60/–40 to 70	5 to 60/–40 to 70	5 to 60/–40 to 70	0 to 70/–40 to 70
RAID support	0, 1	0, 1, 3, 4, 5, 6, 10	0, 1, 3, 4, 5, 6, 10	0, 1, 3, 4, 5, 6, 10
RAID Rebuild™ support		•	•	
Enterprise expert support		•	•	•

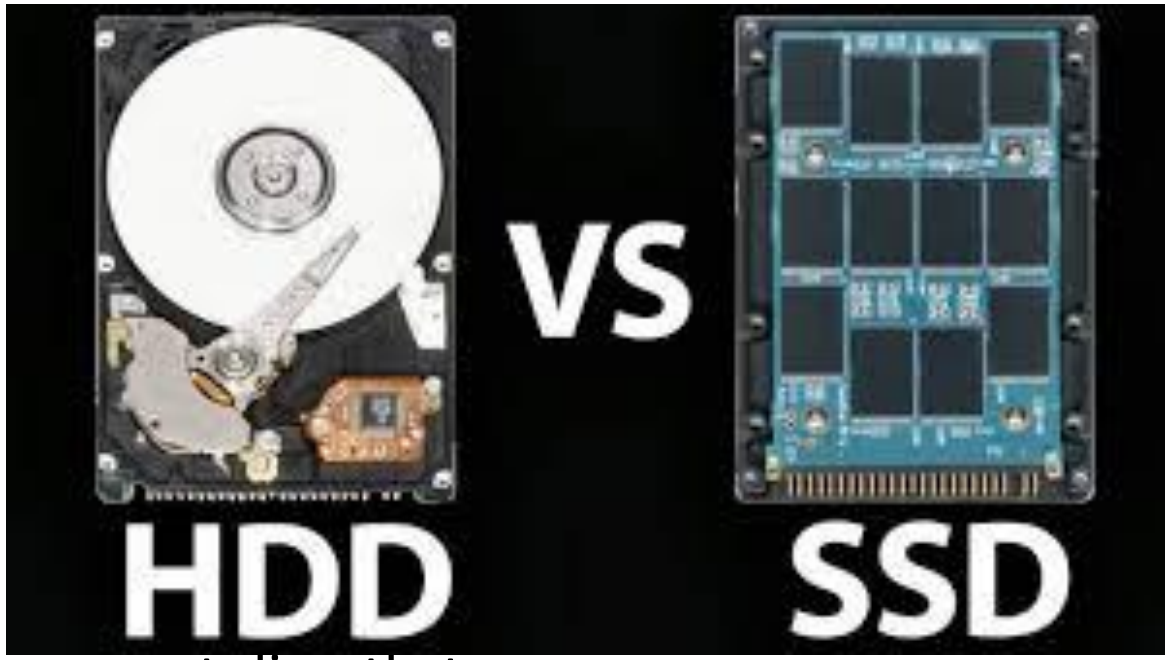
Overview of Mass Storage Structure

- Disk head flies on an extremely thin cushion of air.
- head will sometimes damage the magnetic surface when it makes contact with it.
  - Head crash
  - Not recoverable
- A disk can be **removable.**
- Removable magnetic disks generally consist of one platte
- Examples CDs, DVDs, and Blu-ray discs as well as removable flash-memory

Mass Storage Structure (Cont.)

- Drive attached to computer via **I/O bus**
  - Busses vary, including **EIDE**, **ATA**, **SATA**, **USB**, **Fiber Channel**, **SCSI, SAS, Firewire**
    - SCSI is a set of parallel interface standards developed by ANSI
    - allows more hard disks per computer compared to IDE
    - SCSI is harder to configure compared to SATA and IDE
    - EIDE has a 133 megabytes per second speed rate, while SATA has up to a 150 megabytes per second speed rate.
  - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

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
- Standard Bus interfaces can be too slow -> connect directly to the system bus (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

Magnetic Tape (Cont.)

- Kept in spool and wound or rewound past read-write head
- Moving to the correct spot on a tape can take minutes, but once positioned, tape drives can write data at speeds comparable to disk drives.
- Tape capacities vary greatly, depending on the particular kind of tape drive, with current capacities exceeding several terabytes.
- Some tapes have built-in compression that can more than double the effective storage.
- Tapes and their drivers are usually categorized by width, including 4, 8, and 19 millimeters and 1/4 and 1/2 inch.
- Some are named according to technology, such as LTO-5 and SDLT.

THANK YOU

Chandravva Hebbi
Department of Computer Science Engineering

chandravvahebbi@pes.edu

OPERATING SYSTEMS

Storage Management

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Mass-Storage Structure – Disk Scheduling

FCFS, SSTF, SCAN, C-SCAN, LOOK

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

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

Disk Scheduling (Cont.)

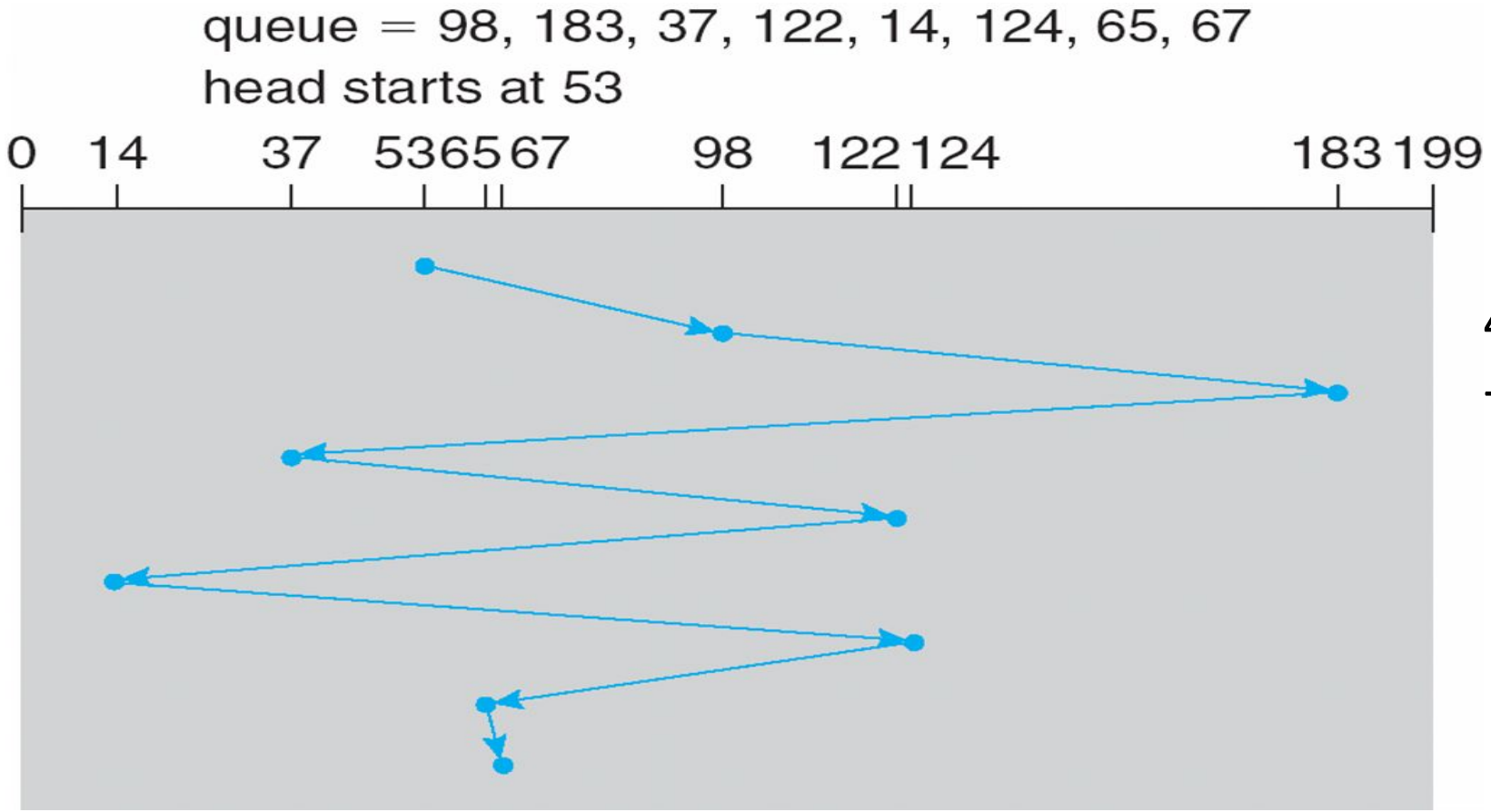
- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)

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

Head pointer 53

FCFS

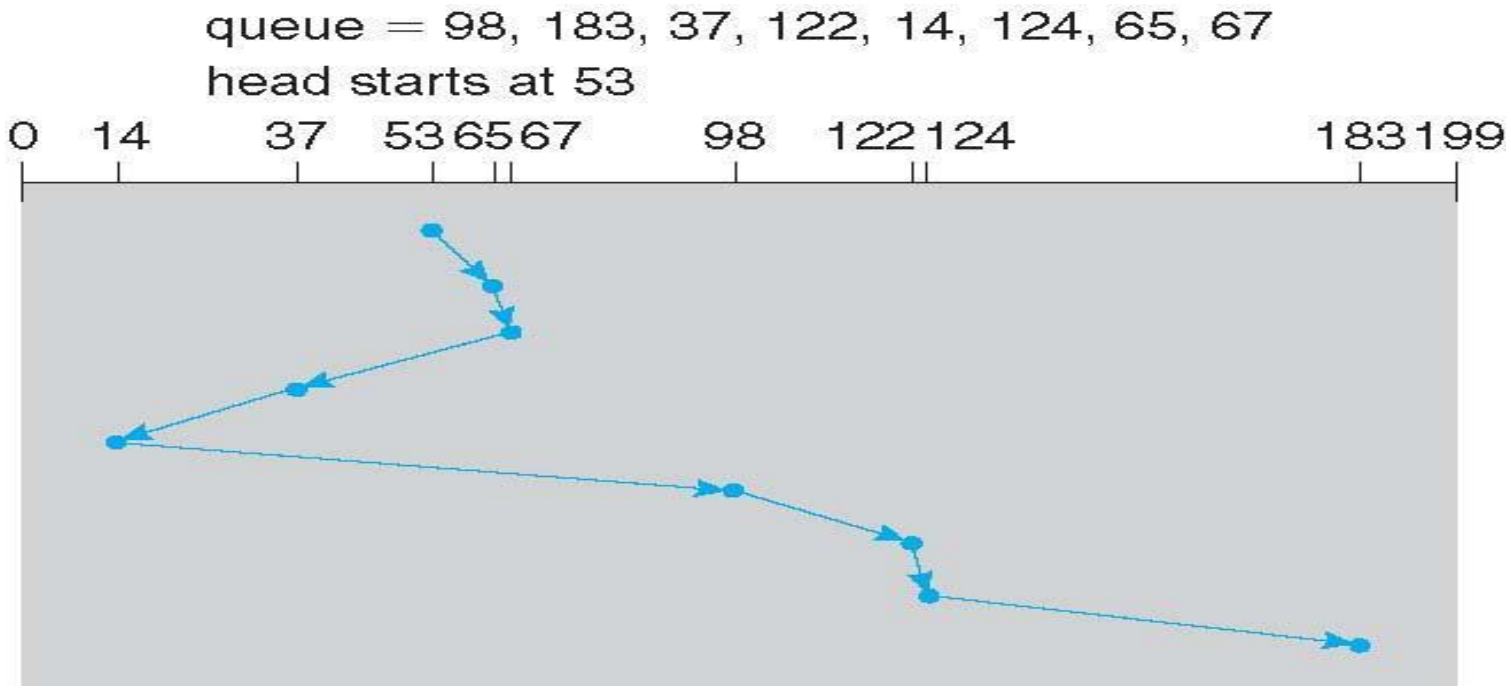
Illustration shows total head movement of 640 cylinders



45+85+146+85+108+110+59
+2 = 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

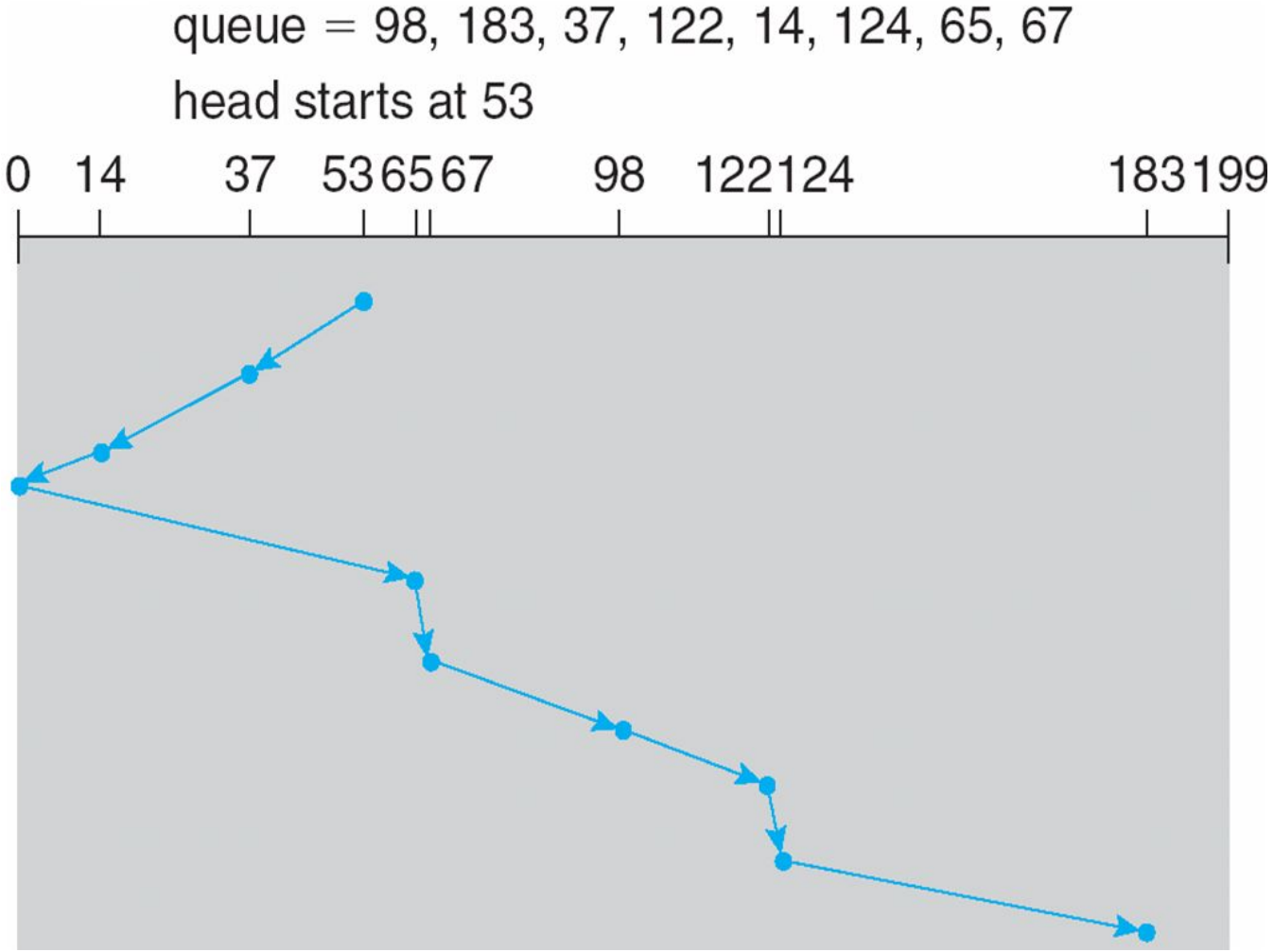


12+2+30+23+84+24+2+59 = 236 cylinders

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

SCAN (Cont.)



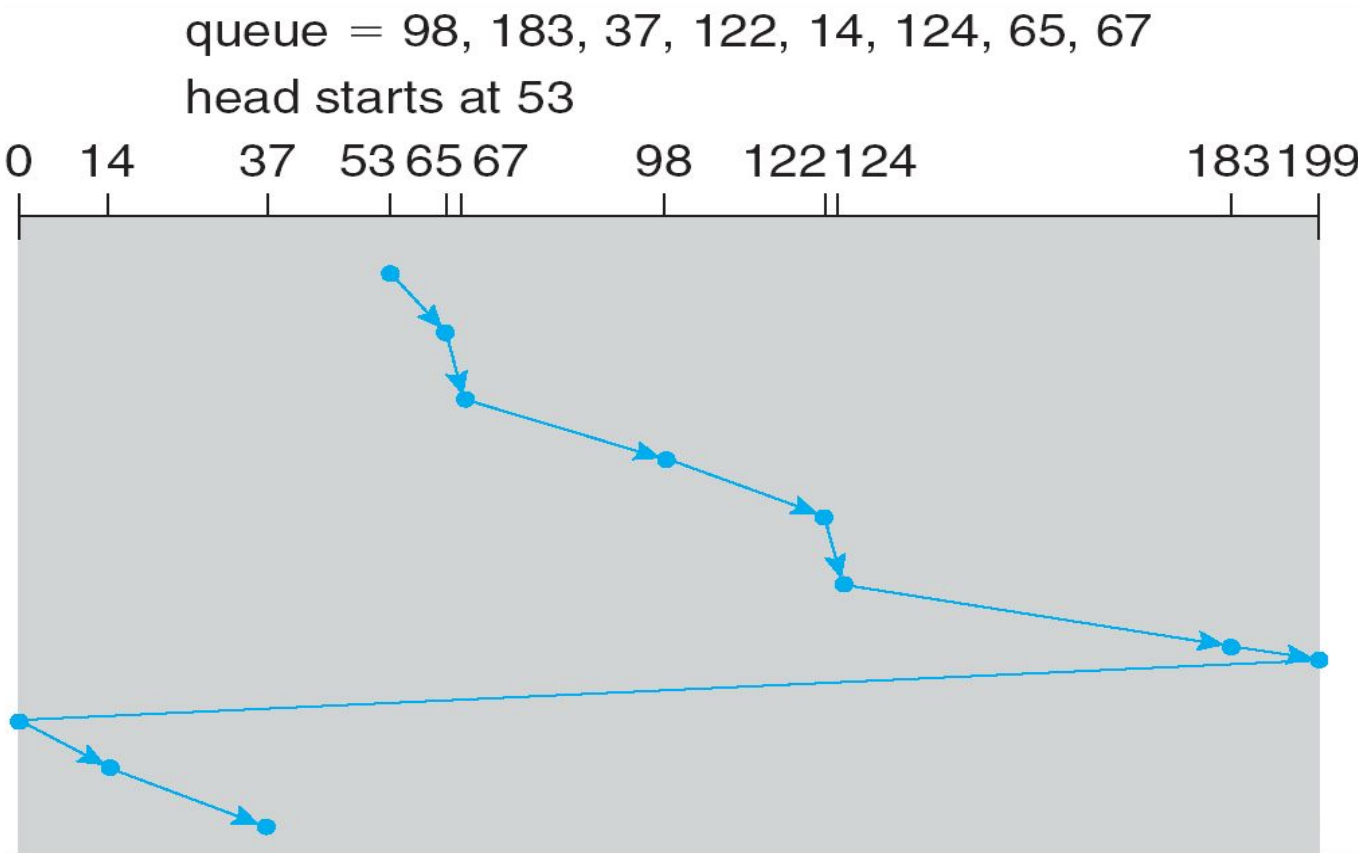
Number of cylinder moves=
16+23+79+2+31+24+2+59=236

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?

OPERATING SYSTEMS

C-SCAN (Cont.)



= (65-53)+(67-65)+(98-67)+(122-98)+(124-122)+(183-124)+(199-183)+(199-0)+(14-0)+(37-14)

=12+2+31+24+2+59+16+199+14+23

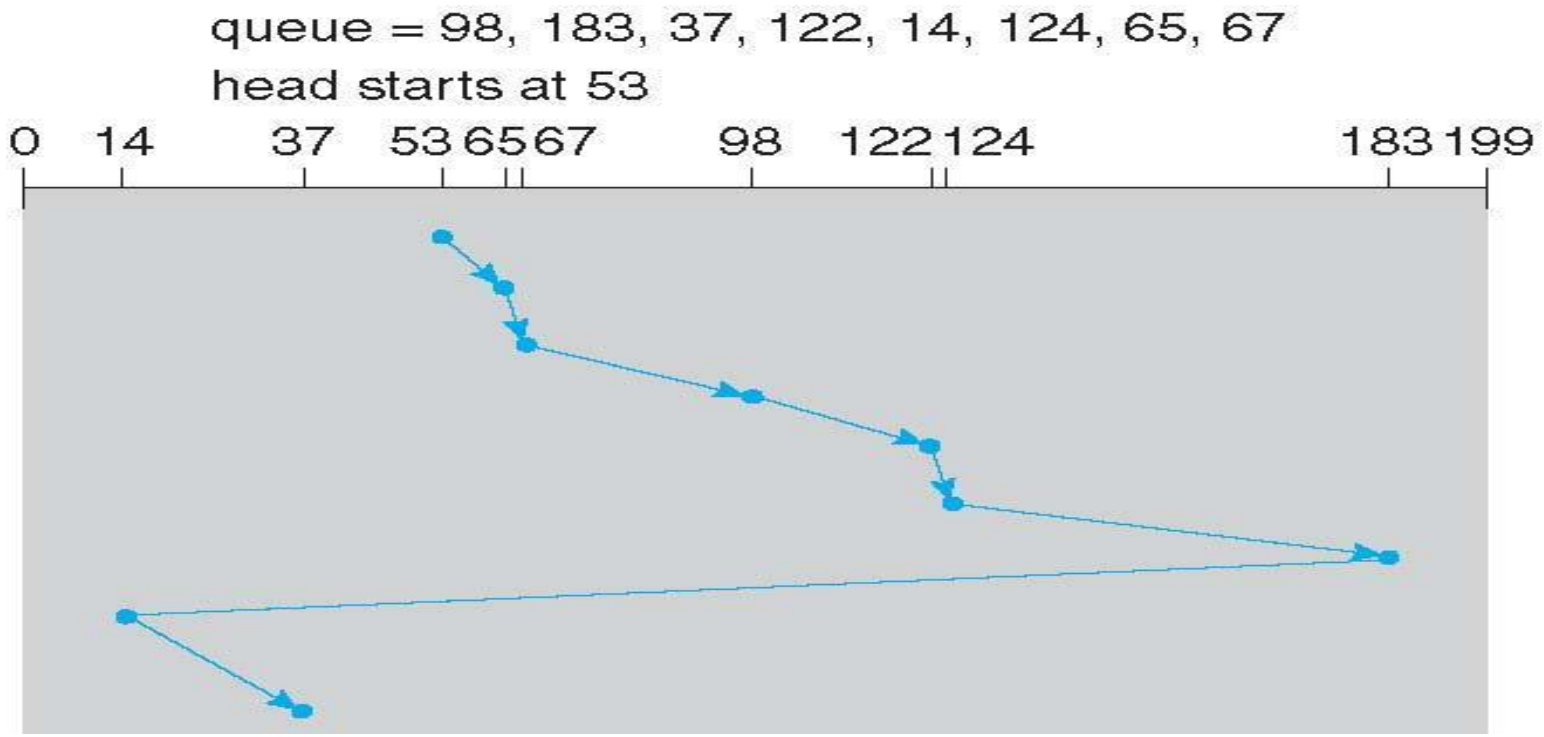
= 382

LOOK & 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?

OPERATING SYSTEMS

C-LOOK (Cont.)



Total head movements incurred while servicing these requests

= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (183 – 14) + (37 – 14)
= 12 + 2 + 31 + 24 + 2 + 59 + 169 + 23
= 322

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

Selecting a Disk-Scheduling Algorithm (Cont.)

- What about rotational latency?
  - Difficult for OS to calculate
  - The rotational latency can be nearly as large as the average seek time.
  - It is difficult for the operating system to schedule for improved rotational latency, though, because modern disks do not disclose the physical location of logical blocks.
  - Disk manufacturers have been alleviating this problem by implementing disk-scheduling algorithms in the controller hardware built into the disk drive.
- How does disk-based queueing effect OS queue ordering efforts?
  - If the OS sends a batch of requests to the controller, the controller can queue them and then schedule them to improve both the seek time and the rotational latency.

THANK YOU

Chandravva Hebbi
Department of Computer Science Engineering

chandravvahebbi@pes.edu

OPERATING SYSTEMS

Storage Management

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Mass-Storage Structure – Swap Space and RAID

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Swap-Space Management

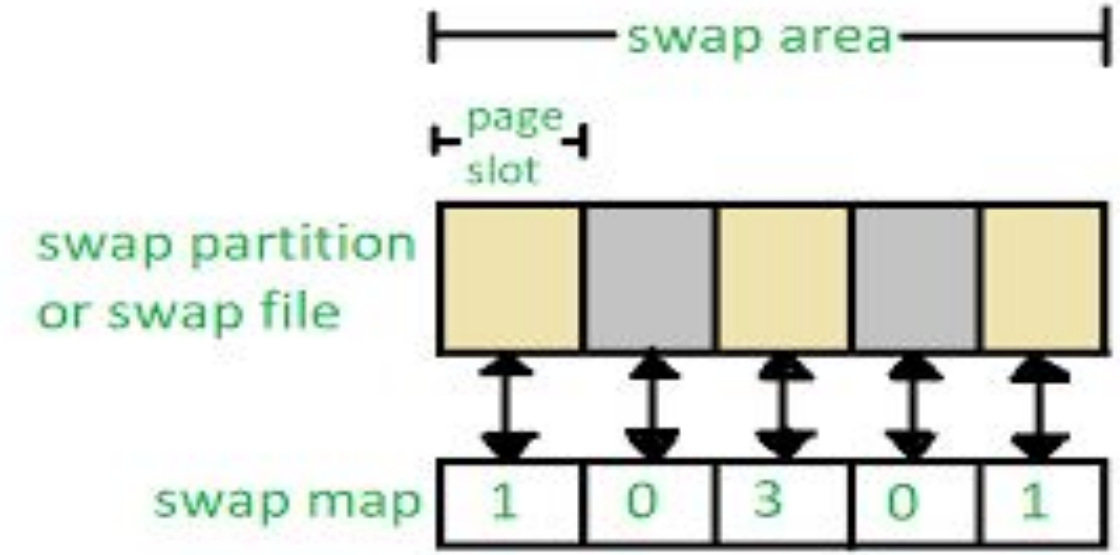
- Swap-space — Virtual memory uses disk space as an extension of main memory
  - Less common now due to memory capacity increases
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition (raw)
- Swap-space management
  - 4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment
  - Kernel uses **swap maps** to track swap-space use
  - Some systems allow the use of multiple swap spaces – both files and dedicated swap partitions

Swap-Space Management (Cont.)

- 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

- Linux allows one or more swap areas to be established.
- A swap area may be in either a swap file on a regular file system o a dedicated swap partition.
- Each swap area consists of a series of 4-KB **page slots**, which are used to hold swapped pages.
- Associated with each swap area is a **swap map**—an array of integer counters, each corresponding to a page slot in the swap area.
  - 0 => page slot is available
  - 3 => swapped page is mapped to 3 different processes



RAID Basics

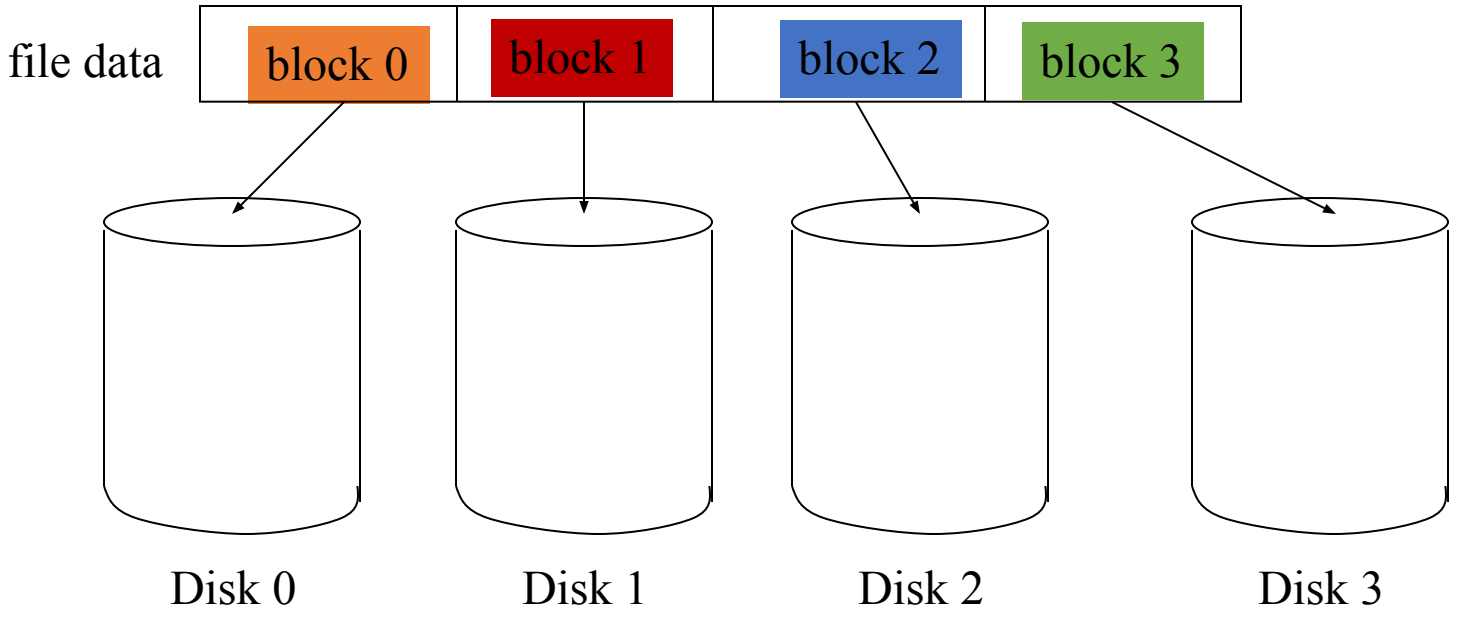
- RAID is a Disk Organization technique
  - Addresses performance and reliability issues
  - Multiple disk drives provide (or improve) reliability via **redundancy**
- Many systems today need to store many terabytes of data
- Don't want to use single, large disk
  - too expensive
  - failures could be catastrophic
- Would prefer to use many smaller disks
- **Redundant Array of Independent (inexpensive) Disks**

RAID Basics (Cont.)

- Basic idea is to connect multiple disks together to provide
  - large storage capacity
  - faster access to reading data
  - redundant data
- Many different levels of RAID systems
  - differing levels of redundancy, error checking, capacity, and cost

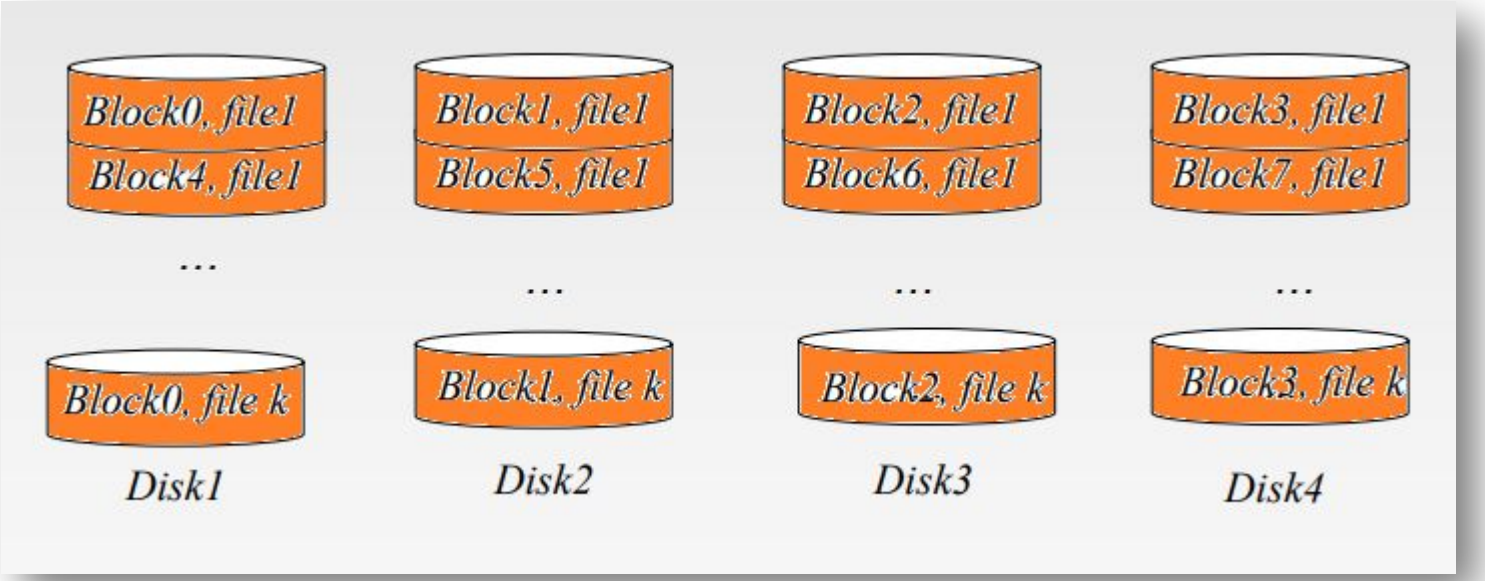
Striping

- Take file data and map it to different disks
- Allows for reading data in parallel
- Transfer rate is improved by striping data across the disks
  - **Bit-level striping and block-level striping (most common)**



Block-level Striping

- With n disks, block i of a file goes to disk (i mod n) + 1
  - Requests for different blocks can run in parallel if the blocks reside on different disks
  - A request for a long sequence of blocks can utilize all disks in parallel



Mirroring

- Keep two copies of data on two separate disks
- Gives good error recovery
  - if some data is lost, get it from the other source
- Expensive
  - requires twice as many disks
- Write performance can be slow
  - have to write data to two different spots
- Read performance is enhanced
  - can read data from file in parallel

RAID Structure

- Increases the **mean time to failure**
  - If MTTF of a single disk is 100,000 hours, MTTF of some disk in an array of 100 disks = 100000/100 = 1000 hours or 41.66 days
- **Mean time to repair –** time taken to replace a failed disk and to restore the data on it
- **Mean time to data loss** based on above factors
  - If mirrored disks fail independently (i.e., not related to power failures and natural disasters), consider disk with MTTF of 100,000 hours and MTTR is 10 hours
  - Mean time to data loss of a mirrored disk system is $100,000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years!

RAID (Cont.)

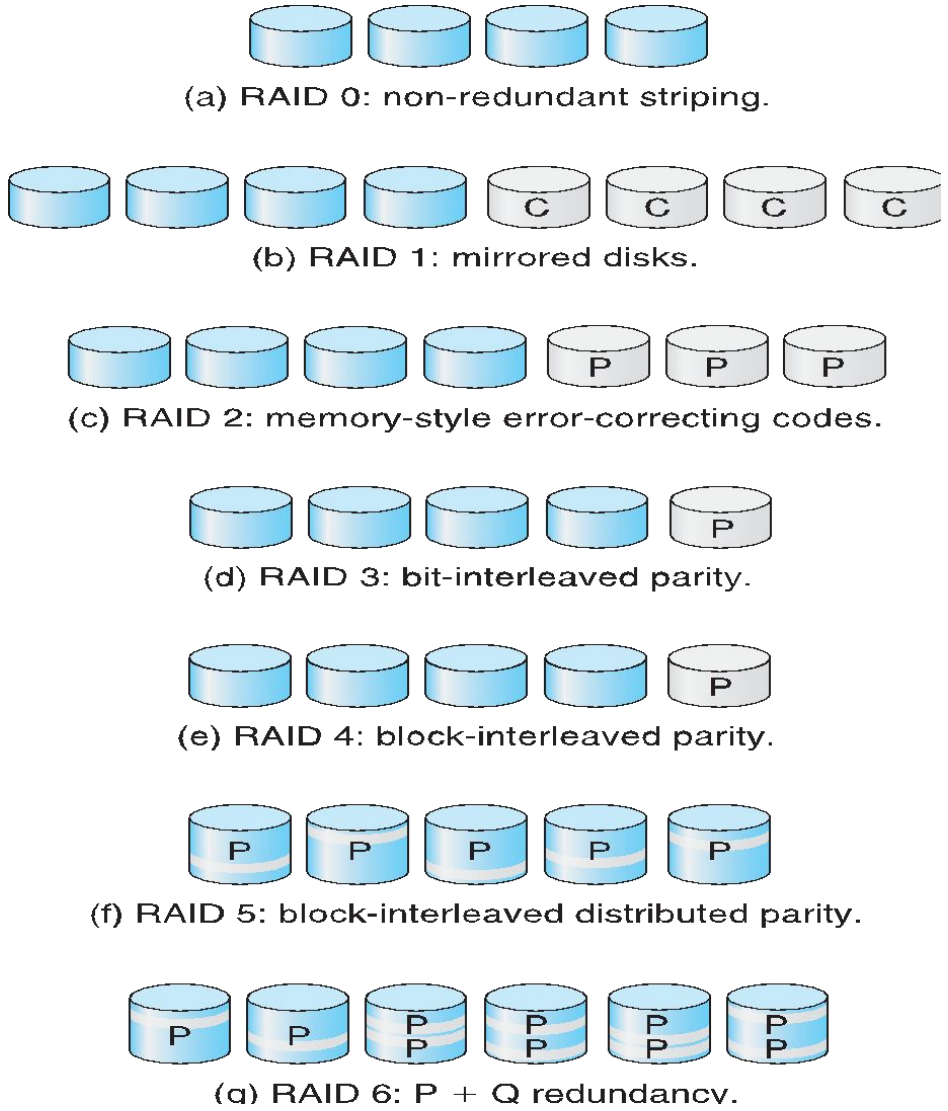
- Frequently combined with **NVRAM** to improve write performance
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively
- Disk **striping** uses a group of disks as one storage unit
- RAID is arranged into six different levels
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data
  - **Mirroring** or **shadowing** (**RAID 1**) keeps duplicate of each disk
  - Striped mirrors (**RAID 1+0**) or mirrored stripes (**RAID 0+1**) provides high performance and high reliability
  - **Block interleaved parity** (**RAID 4, 5, 6**) uses much less redundancy

RAID (Cont.)

- 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

OPERATING SYSTEMS

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.

P indicates error-correcting bits and C indicates a second copy of the data

P + Q redundancy scheme uses 2 parity values P and Q

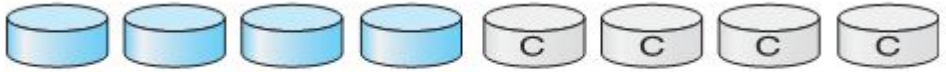
OPERATING SYSTEMS

RAID Levels 0 and 1

- **RAID level 0** refers to disk arrays with striping at the level of blocks
  - No redundancy (such as mirroring or parity bits)
  - lots of disks means low Mean Time To Failure (MTTF)

(a) RAID 0: non-redundant striping.

- **RAID level 1** refers to disk mirroring.
  - A complete file is stored on a single disk
  - A second disk contains an exact copy of the file
  - Provides complete redundancy of data
  - Read performance can be improved
  - file data can be read in parallel
  - Write performance suffers
  - must write the data out twice
  - Most expensive RAID implementation
  - requires twice as much storage space



(b) RAID 1: mirrored disks.

RAID Level-2

- **RAID level 2** is also known as memory-style error correcting code (ECC) organization.

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

  - Stripes data across disks similar to Level-0
    - difference is data is **bit** interleaved instead of **block** interleaved
    - For ex, the first bit of each byte can be stored in disk 1, the second bit in disk 2, and so on until the eighth bit is stored in disk 8; the error-correction bits are stored in further disks.
  - Uses ECC to monitor correctness of information on disk
    - Multiple disks record the ECC information to determine which disk is in fault
    - A parity disk is then used to reconstruct corrupted or lost data
  - RAID level 2 requires only 3 disks' overhead for 4 disks of data, unlike RAID level 1, which requires 4 disks' overhead.

RAID Level-3

- **RAID level 3**, or bit-interleaved parity organization;
  - One big problem with Level-2 is the number of extra disks needed to detect which disk had an error
  - Modern disks can already determine if there is an error
    - using ECC codes with each sector
  - So just need to include a parity disk
    - if a sector is bad, the disk itself tells us, and use the parity disk to correct it
  - Transfer rate for reading or writing a single block is faster than RAID level 1.
  - But supports fewer I/Os per second, since every disk has to participate in every I/O request.
  - Has performance problem due to the expense of computing and writing the parity.

(d) RAID 3: bit-interleaved parity.

RAID Level-4

- **RAID level 4** interleaves file blocks
  - allows multiple small I/O's to be done at once

(e) RAID 4: block-interleaved parity.

- Consists of block-level striping with dedicated parity.
- Still use a single disk for parity
- Now the parity is calculated over data from multiple blocks
  - Level-2,3 calculate it over a single block
- If an error detected, need to read other blocks on other disks to reconstruct data

- Doing multiple small reads is now faster than before
- However, writes are still very slow
  - this is because of calculating and writing the parity blocks
- Also, only one write is allowed at a time
  - all writes must access the check disk so other writes have to wait

OPERATING SYSTEMS

RAID Level-5

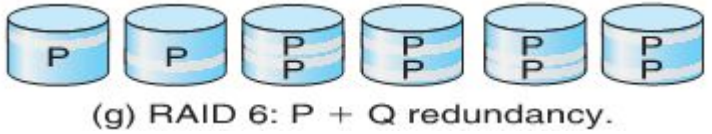
- **RAID level 5** stripes file data and checks data over all the disks
  - no longer a single check disk
  - no more write bottleneck

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

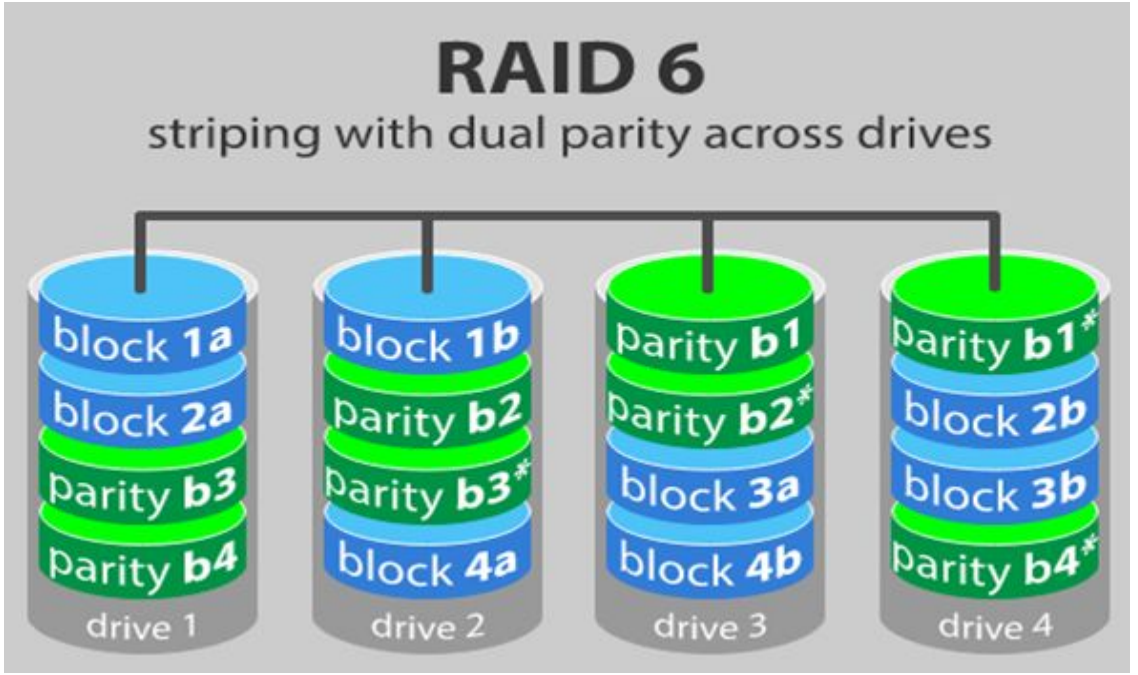
- Consists of block-level striping with distributed parity.
  - Unlike RAID 4, parity information is distributed among the drives
- Drastically improves the performance of multiple writes
  - they can now be done in parallel
- Slightly improves reads
  - one more disk to use for reading
- read and write performance close to that of RAID Level-1
- requires as much disk space as Levels-3,4

RAID Level-6

- RAID 6 is like RAID 5, but the parity data are written to two drives.
  - That means it requires at least 4 drives and can withstand 2 drives dying simultaneously.
- Write data transactions are slower than RAID 5 due to the additional parity data that have to be calculated.

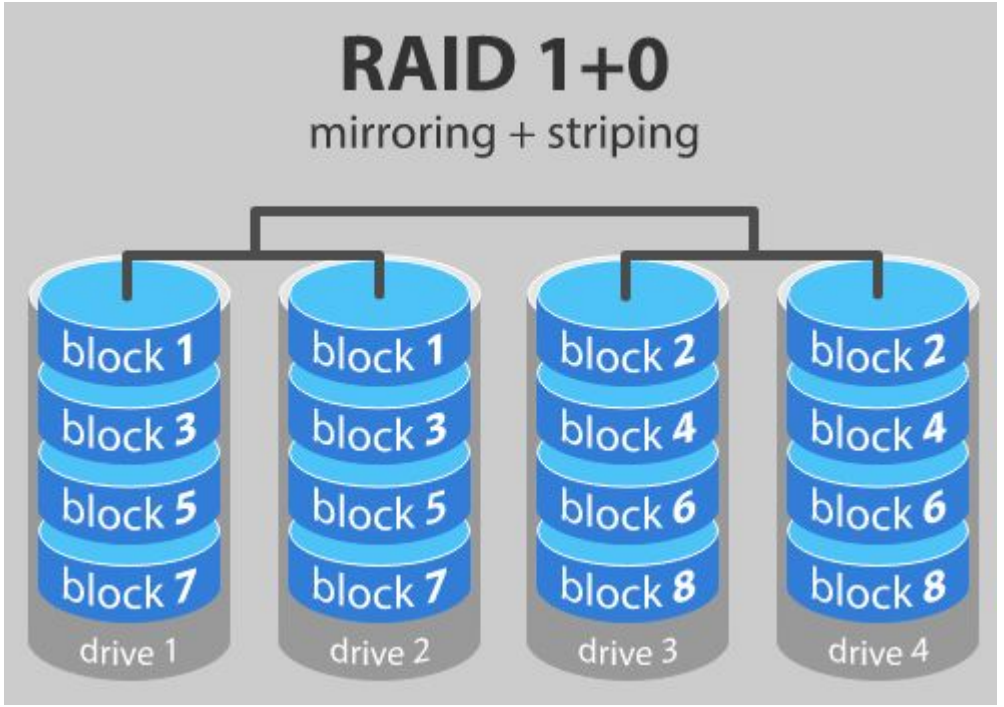


(g) RAID 6: P + Q redundancy.



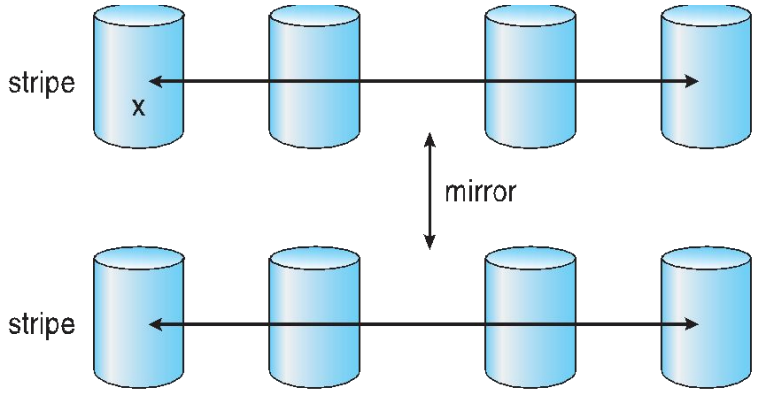
RAID level 10 - combining RAID 1 and RAID 0

- Provides security by mirroring all data on secondary drives while using striping across each set of drives to speed up data transfers.
- Rebuild time is very fast
- Half of the storage capacity goes to mirroring
  - so compared to large RAID 5 or RAID 6 arrays, this is an expensive way to have redundancy.



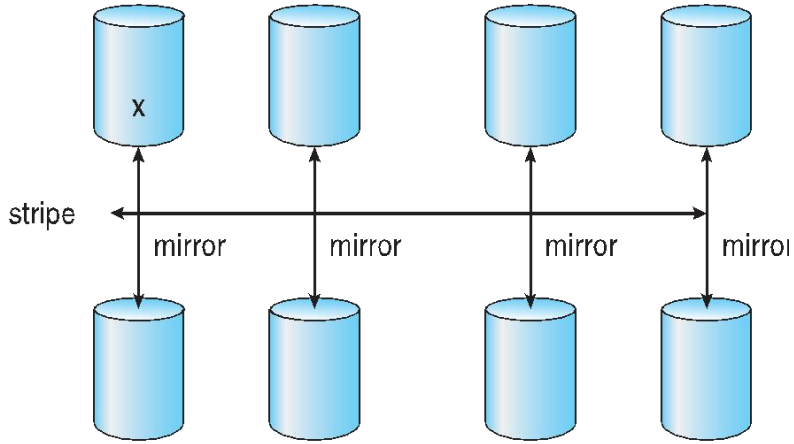
OPERATING SYSTEMS

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



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

Disks are striped and then the stripe is mirrored to another, equivalent stripe. If a single disk fails, an entire stripe is inaccessible.



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

Disks are mirrored in pairs and then the resulting mirrored pairs are striped. This is better than 0+1 when a single disk fails

Selecting a RAID level

- One consideration is rebuild performance.
  - If a disk fails, the time needed to rebuild its data can be significant.
  - This may be an important factor if a continuous supply of data is required, as it is in high-performance or interactive database systems.
  - Furthermore, rebuild performance influences the mean time to failure.
  - Rebuild performance varies with the RAID level used.
    4. Rebuilding is easiest for RAID level 1, since data can be copied from another disk.

Selecting a RAID level (Cont.)

- For the other levels, we need to access all the other disks in the array to rebuild data in a failed disk.
- Rebuild times can be hours for RAID 5 rebuilds of large disk sets.

- 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 example, for small databases.
- Due to RAID 1's high space overhead, RAID 5 is often preferred for storing large volumes of data.

Selecting a RAID level (Cont.)

- RAID system designers and administrators of storage have to make several other decisions as well.
  - How many disks should be in a given RAID set?
  - How many bits should be protected by each parity bit?
  - If more disks are in an array, data-transfer rates are higher, but the system is more expensive.
  - If more bits are protected by a parity bit, the space overhead due to parity bits is lower, but the chance that a second disk will fail before the first failed disk is repaired is greater, and that will result in data loss.

THANK YOU

Chandravva Hebbi
Department of Computer Science Engineering

chandravvahebbi@pes.edu

OPERATING SYSTEMS

I/O Management, System Protection and Security

Arya S S
Department of Computer Science

OPERATING SYSTEMS

System Protection - Goals, Principles and Domain of Protection

Arya S S
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Goals of Protection

- In one protection model, computer consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so
- Protection provides a mechanism for the enforcement of the policies governing resource use
  - Some policies are fixed in the design of the system, while others are formulated by the management of a system and by the individual users as well (to protect their own files and programs).

Principles of Protection

- Guiding principle – **principle of least privilege**
  - Programs, users and systems should be given just enough **privileges** to perform their tasks
  - Failure of a component does the minimum damage and allows the minimum damage to be done.
  - OS follows this principle for its features, programs, system calls and data structures
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**

Principles of Protection (Cont.)

- Must consider “grain” aspect
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective
    - Provide/disable privileges as needed
    - Create audit trail for privileged function access
    - File ACL lists, RBAC

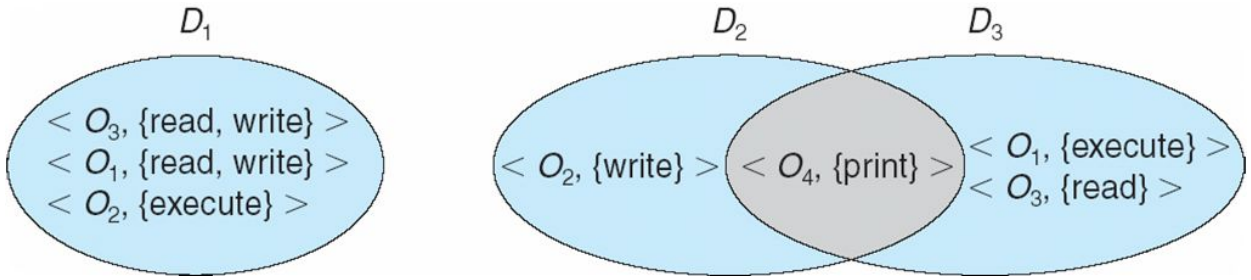
Domain of Protection

- A computer system is a collection of processes and objects.
  - Hardware objects (CPU, memory segments, printers, disks, and tape drives)
  - Software objects (files, programs, and semaphores)
- Each object has a unique name and can be accessed only through well-defined and meaningful operations.
- A process should be allowed to access only those resources for which it has authorization.
- A process should be able to access only those resources that it currently requires to complete its task.(**need-to-know principle**)
- Need-to-know principle, is useful in limiting the amount of damage a faulty process can cause in the system

OPERATING SYSTEMS

Domain Structure

- A process operates within a **protection domain**, which specifies the resources that the process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- The ability to execute an operation on an object is an **access right**
- **Access-right=<object-name, rights-set>**
  where rights-set is a subset of all valid operations that can be performed on the object.



System with 3 protected domains

Domain Structure(Cont.)

- The association between a process and a domain may be either static or dynamic.
- **Static**: the set of resources available to the process is fixed throughout the process's lifetime.
  - But a process may execute in two different phases, for eg ,need read access in one phase and write access in another phase.
  - So the fixed domain should contain both read and write access for that process.
  - This violates need to know principle.
  - So we must allow the contents of domain to be modified during each phase of the process.
- If the association is **dynamic**, a mechanism is available to allow domain switching.

Domain Structure (Cont.)

- A domain can be realized in a variety of ways:
  - User
  - Process
  - Procedure
- Standard dual-mode (monitor-user mode) model of operating-system execution is followed in system.
- But in a multi programming environment these two protection domains are insufficient, since users want to be protected from one another.
- So a more elaborate scheme is used in UNIX and MULTICS.

THANK YOU

Arya S S

Department of Computer Science Engineering

aryadeep@pes.edu

OPERATING SYSTEMS

I/O Management, System Protection and Security

Arya S S
Department of Computer Science

OPERATING SYSTEMS

Domain of Protection: Unix, MULTICS examples

Arya S S
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Domain Implementation (UNIX)

- Domain = user-id
- Domain switch accomplished via file system
  - Each file has associated with it a domain bit (setuid bit)
  - When file is executed and setuid = on, then user-id is set to owner of the file being executed
  - When execution completes user-id is reset
- Domain switch accomplished via passwords
  - su command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
  - sudo command prefix executes specified command in another domain (if original domain has privilege or password given)

OPERATING SYSTEMS

Domain Implementation (MULTICS)

- The protection domains are organized hierarchically into a ring structure.
- Each ring corresponds to a single domain
- The rings are numbered from 0 to 7.
- A process executing in domain Do has the most privileges.
- MULTICS has a segmented address space.
- Each segment is a file, and each segment is associated with one of the rings.
- It includes three access bits to control reading, writing, and execution

Multiplexed Information and Computing Service

- was a cooperative project led by MIT along with General Electric and Bell Labs.

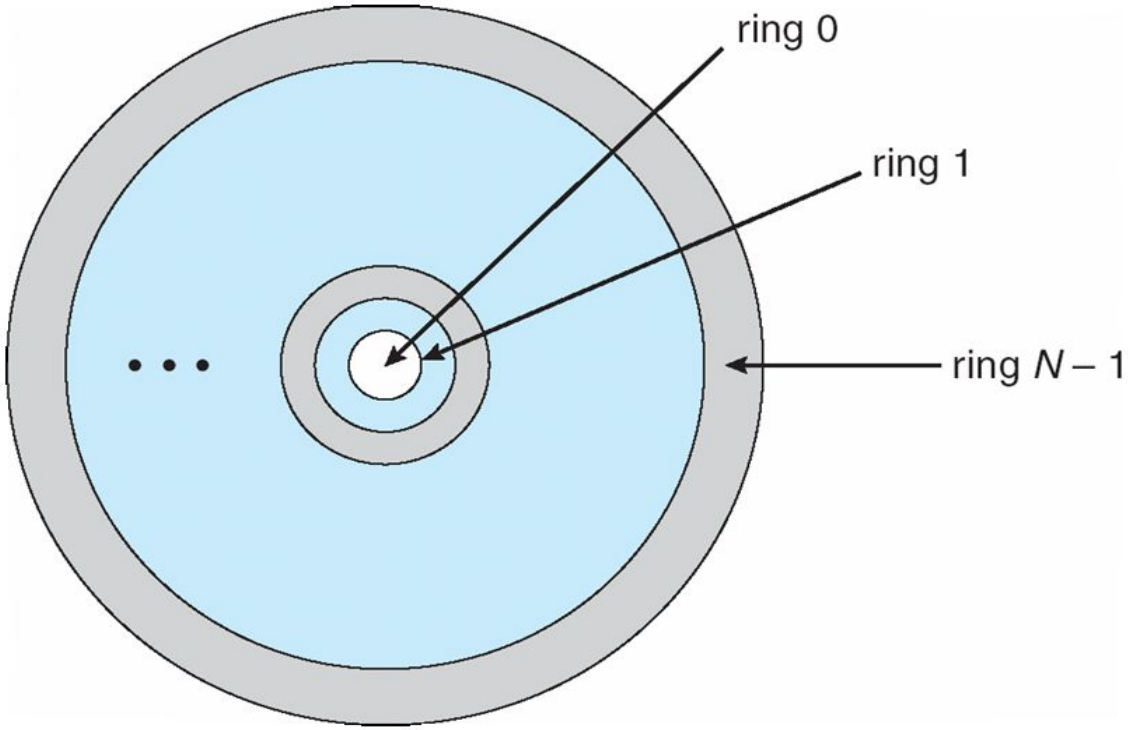
-was started in 1964 and has influenced all modern operating systems from microcomputers to mainframes

--was the first major OS to be designed as a secure system

-- had hardware support for ring-oriented security

Multics ring Structure

- Let Di and Dj be any two domain rings.
- If $j < i \Rightarrow Di \subseteq Dj$



Multics Domain Implementation

- A current-ring-number counter is associated with each process, identifying the ring in which the process is executing currently.
- When a process is executing in ring i, it cannot access a segment associated, with ring j (j < i).
- It can access a segment associated with ring k (k > i).
- Domain switching in MULTICS occurs when a process crosses from one ring to another by calling a procedure in a different ring.

Multics Domain Switching

Ring field of the segment descriptor include the following:

1. Access bracket. A pair of integers, b1 and b2, such that b1<= b2.
2. Limit. An integer b3 such that b3 > b2.
3. List of gates. Identifies the entry points (or gates) at which the segments may be called.

- If a process operating in ring i calls a segment whose bracket is such that b1 <= i <= b2, then the call succeeds and the current ring no of process remains i.
- Otherwise a trap to the OS occurs, and is handled as follows:
  - If i < b1, then the call is allowed, because we are transferring to a procedure with fewer privileges.
  - If i > b2, then the call is allowed only if i <= b3 and the call is directed to one of the entries on the list of gates.

THANK YOU

Arya S S

Department of Computer Science Engineering

aryadeep@pes.edu

OPERATING SYSTEMS

I/O Management, System Protection and Security

Arya S S
Department of Computer Science

OPERATING SYSTEMS

System Protection- Access Matrix

Arya S S
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Access Matrix

- View protection abstractly as a matrix (**access matrix**)
- Rows represent domains
- Columns represent objects
- **Access(i, j)** is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- User who creates object can define access column for that object
- Can be expanded to dynamic protection
  - Operations to add, delete access rights
  - Special access rights:
    - *owner of* O_i
    - *copy op from* O_i *to* O_j *(denoted by "*")*
    - *control –* D_i *can modify* D_j *access rights*
    - *transfer – switch from domain* D_i *to* D_j
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object

Use of Access Matrix (Cont.)

- **Access matrix** design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - It ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy
    - User dictates policy
    - Who can access what object and in what mode
- But doesn't solve the general confinement problem i.e. preventing a process from taking disallowed actions
  - Ex: In a client/server situation, preventing a server from leaking information that the client considers confidential

Access Matrix with Domains as Objects

Processes should be able to switch from one domain to another.

Switching from domain *Di* to domain *Dj* is allowed if and only if the access right switch $\in$ access(*i*, *j*).

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access Matrix with Copy Rights

- The ability to copy an access right from one domain (or row) of the access matrix to another is denoted by an asterisk (*) appended to the access right.

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix with Copy Rights

The copy scheme has 3 variants:

1. A right is copied from access(i,j) to access(k,j) is not limited: This action is called **copy**.
    - When the right Read* is copied from access(i,j) to access(k,j), the Read* is created.
    - So, a process executing in Dk can further copy the right Read*.
2. Propagation of the copy right may be limited: This action is called **limited copy**.
    - When the right Read* is copied from access(i,j) to access(k,j), only the Read (not Read*) is created.
    - So, a process executing in Dk cannot further copy the right Read.
3. A right is copied from access(i,j) to access(k,j); it is then removed from access(i,j).
    - This action is called a **transfer of a right**, rather than a copy

Access Matrix with Owner Rights

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

- **Owner** right controls addition of new rights and removal of some rights.
- Domain D1 is the owner of F1 and can add /delete any valid right in column F1
- Owner rights allow a process to change the entries in a column

Access Matrix with Control Rights

- The copy and owner rights allow a process to change the entries in a column.
- A mechanism is now needed to change the entries in a row.
- The control right is applicable only to domain objects (rows).
- If access(i,j) includes the control right, then a process executing in Di can remove any access right from row j.
- If we include the *control* right in access(D2, D4), then, a process executing in domain D2 could modify domain D4.

Modified Access Matrix with Domains as Objects

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Fig A

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Fig B: Modified access matrix of fig A

Add **control** right in access(D2, D4).

Then, a process executing in D2 (row) could modify D4 (row).

THANK YOU

Arya S S

Department of Computer Science Engineering

aryadeep@pes.edu

OPERATING SYSTEMS

I/O Management, System Protection and Security

Arya S S
Department of Computer Science

OPERATING SYSTEMS

System Protection – Implementation of Access Matrix, Access control, Access rights

Arya S S
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

Implementation of Access Matrix

- Generally, a sparse matrix (i.e. most of the entries will be empty)
- **Option 1 – Global table**
  - Store ordered triples **<domain, object, rights-set>** in table
  - A requested operation M on object O_j within domain D_i -> search table for a triple $< D_i, O_j, R_k >$
    - with $M \in R_k$
    - If triple found, operation is allowed to continue; otherwise an exception or condition is raised
  - But table could be large -> won't fit in main memory
    - Virtual memory techniques can be used for managing this table
  - Difficult to group objects
    - consider an object that all domains can read, this object must have a separate entry in every domain)

Implementation of Access Matrix (Cont.)

- **Option 2 – Access lists for objects**
  - Each **column** implemented as an access list for one object i.e. specifying user names and the types of access allowed for each user (empty entries can be discarded)
  - Resulting per-object list consists of ordered pairs **<domain, rights-set>** defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If M ∈ default set, also allow access
    - For efficiency, check the default set first and then search the access list

Implementation of Access Matrix (Cont.)

- Each column = Access-control list for one object
  Defines who can perform what operation

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- Each Row = Capability List (like a key)
  For each domain, what operations allowed on what objects

  Object F1 – Read

  Object F4 – Read, Write, Execute

  Object F5 – Read, Write, Delete, Copy

Implementation of Access Matrix (Cont.)

- **Option 3 – Capability list for domains**
  - Instead of object-based (i.e column wise), list is domain based (i.e row wise)
  - **Capability list** for domain is list of objects together with operations allowed on them
  - Object represented by its name or address, called a **capability**
  - Execute operation M on object O_j, process requests operation and specifies capability as parameter
    - **Possession of capability** means access is allowed
  - Capability list associated with domain but never directly accessible to a process executing in that domain
    - Rather, protected object, maintained by OS and accessed by the user indirectly
    - Like a “secure pointer”
    - Idea can be extended up to the application level

Implementation of Access Matrix (Cont.)

- **Option 4 – Lock-key**
  - Compromise between access lists and capability lists
  - Each object has a list of unique bit patterns, called **locks**
  - Each domain has a list of unique bit patterns called **keys** (managed by the OS)
  - Process in a domain can only access object if domain has a key that matches one of the locks

Comparison of Implementations

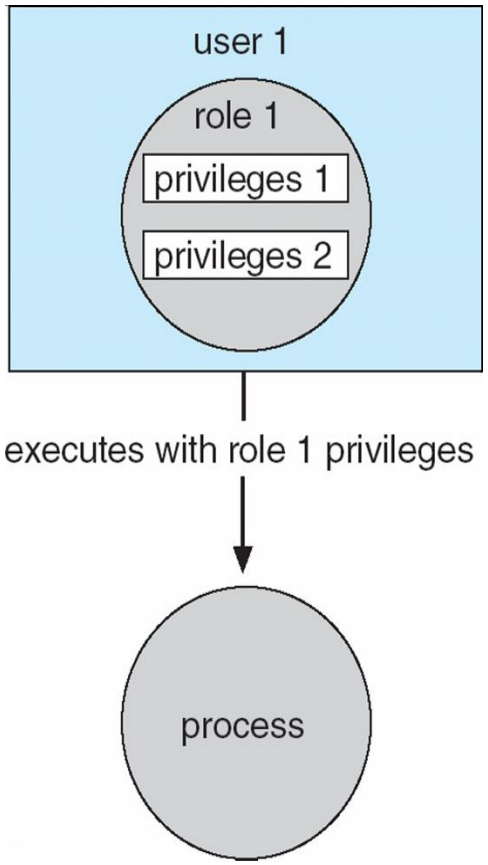
- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Access rights for a particular domain is non-localized, so difficult to determine the set of access rights for each domain
    - Every access to an object must be checked
      - Many objects and access rights -> slow (i.e not suitable for large system with long access lists)
  - Capability lists useful for localizing information for a given process
    - But revocation capabilities can be inefficient
  - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

Comparison of Implementations (Cont.)

- Most systems use combination of access lists and capabilities
  - First access to an object -> access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed
    - Consider file system with ACLs per file recorded in a new entry in a file table (file table maintained by the OS such as UNIX and protection is ensured)

Access Control

- Protection can be applied to non-file resources
- Oracle Solaris 10 provides **role-based access control** (**RBAC**) to implement least privilege
  - ***Privilege*** is right to execute system call or use an option (ex: write access for a file) within a system call
  - Can be assigned to processes
  - Users assigned ***roles*** granting access to privileges and programs
    - 4 Enable role via password to gain its privileges
  - Similar to access matrix



Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - **Immediate vs. delayed** (i.e. when revocation will occur)
  - **Selective vs. general** (i.e. select group of users or all the users)
  - **Partial vs. total** (i.e. subset of the rights or all the rights)
  - **Temporary vs. permanent** (can access right be revoked and obtained later?)
- Access List – Delete access rights from access list
  - **Simple** – search access list and remove entry, revocation is easy
  - **Immediate**, **general** or **selective**, **total** or **partial**, **permanent** or **temporary**

Revocation of Access Rights (Cont.)

- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - **Reacquisition** – periodic delete from each domain, with reacquire and denial if revoked by a process
  - **Back-pointers** – set of pointers from each object to all capabilities of that object , follow these pointers for revocation (adopted in Multics)
  - **Indirection** – capability points to global table entry which in turn points to object – delete entry from global table, selective revocation not allowed
  - **Keys** – unique bit pattern associated with a capability, generated when capability is created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key (with a new value)
    - Policy decision of who can create and modify keys – object owner or others?

THANK YOU

Arya S S

Department of Computer Science Engineering

aryadeep@pes.edu

OPERATING SYSTEMS

Storage Management

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Storage Management – System calls

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

File access permission

- There are nine permission bits for each file, divided into three categories
- The term user in the first three rows refers to the owner of the file
- The first rule is that to open any type of file by name, user must have execute permission in each directory mentioned in the name, including the current directory.
- The execute permission bit for a directory is often called the search bit
- For example, to open the file /usr/include/stdio.h, user would need execute permission in the directory /, execute permission in the directory /usr, and execute permission in the directory /usr/include and appropriate permission for the file stdio.h

st_mode mask	Meaning
S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute

OPERATING SYSTEMS

File access permission

- The read permission for a file determines whether user can open an existing file for reading: the O_RDONLY and O_RDWR flags for the open function.
- The write permission for a file determines whether user can open an existing file for writing: the O_WRONLY and O_RDWR flags for the open function.
- User must have write permission for a file to specify the O_TRUNC flag in the open function.
- User cannot create a new file in a directory unless they have write permission and execute permission in the directory.
- To delete an existing file, user needs write permission and execute permission in the directory containing the file.
- Users do not need read permission or write permission for the file itself

Set-User-ID and Set-Group-ID

real user ID real group ID	who we really are
effective user ID effective group ID supplementary group IDs	used for file access permission checks
saved set-user-ID saved set-group-ID	saved by exec functions

- Every process has six or more IDs associated with it
- The real user ID and real group ID identify who the user really is. These two fields are taken from an entry in the password file when the user logs in
- The effective user ID, effective group ID, and supplementary group IDs determine file access permissions for the user
- The saved set-user-ID and saved set-group-ID contain copies of the effective user ID and the effective group ID, respectively, when a program is executed.

Set-User-ID and Set-Group-ID

```
#include <unistd.h>
int setuid(uid_t uid);
int setgid(gid_t gid);
```

Both return: 0 if OK, –1 on error

- We can set the real user ID and effective user ID with the setuid function.
- We can set the real group ID and the effective group ID with the setgid function.
- If the process has superuser privileges, the setuid function sets the real user ID, effective user ID, and saved set-user-ID to uid.
- If the process does not have superuser privileges, but uid equals either the real user ID or the saved set-user-ID, setuid sets only the effective user ID to uid. The real user ID and the saved set-user-ID are not changed.
- If neither of these two conditions is true, errno is set to EPERM and −1 is returned

access and faccessat

```
#include <unistd.h>
int access(const char *pathname, int mode);
int faccessat(int fd, const char *pathname, int mode, int flag);
                                        Both return: 0 if OK, –1 on error
```

mode	Description
R_OK	test for read permission
W_OK	test for write permission
X_OK	test for execute permission

- When a user tries to open a file, the kernel performs access tests based on the effective user and group IDs
- Sometimes, however, a process wants to test accessibility based on the real user and group IDs. This is useful when a process is running as someone else, using either the set-user-ID or the set-group-ID feature.
- The **access** and **faccessat** functions base their tests on the real user and group IDs.

umask

```
#include <sys/stat.h>
mode_t umask(mode_t cmask);
                                    Returns: previous file mode creation mask
```

- The umask function sets the file mode creation mask for the process and returns the previous value
- The cmask argument is formed as the bitwise OR of any of the nine constants
- The file mode creation mask is used whenever the process creates a new file or a new directory.

st_mode mask	Meaning
S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute

chmod, fchmod, and fchmodat

```
#include <sys/stat.h>
int chmod(const char *pathname, mode_t mode);
int fchmod(int fd, mode_t mode);
int fchmodat(int fd, const char *pathname, mode_t mode, int flag);
                                    All three return: 0 if OK, –1 on error
```

- The chmod, fchmod, and fchmodat functions allow us to change the file access permissions for an existing file
- The chmod function operates on the specified file, whereas the fchmod function operates on a file that has already been opened.
- The fchmodat function behaves like chmod when the pathname argument is absolute or when the fd argument has the value AT_FDCWD and the pathname argument is relative
- To change the permission bits of a file, the effective user ID of the process must be equal to the owner ID of the file, or the process must have superuser permissions

chown, fchown, fchownat, and lchown

```
#include <unistd.h>

int chown(const char *pathname, uid_t owner, gid_t group);

int fchown(int fd, uid_t owner, gid_t group);

int fchownat(int fd, const char *pathname, uid_t owner, gid_t group,
             int flag);

int lchown(const char *pathname, uid_t owner, gid_t group);
```

All four return: 0 if OK, –1 on error

- The chown functions allow users to change a file's user ID and group ID, but if either of the arguments owner or group is −1, the corresponding ID is left unchanged
- The fchown function changes the ownership of the open file referenced by the fd argument.
- lchown and fchownat (with the AT_SYMLINK_NOFOLLOW flag set) change the owners of the symbolic link itself, not the file pointed to by the symbolic link

OPERATING SYSTEMS

File truncation

```
#include <unistd.h>

int truncate(const char *pathname, off_t length);

int ftruncate(int fd, off_t length);
```

Both return: 0 if OK, –1 on error

- **truncate** and **ftruncate** functions truncate an existing file to *length* bytes.
- If the previous size of the file was greater than *length*, the data beyond *length* is no longer accessible.
- If the previous size was less than *length*, the file size will increase and the data between the old end of file and the new end of file will read as 0 (i.e., a hole is probably created in the file)

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu

OPERATING SYSTEMS

I/O Management, System Protection and Security and Case Study

Kakoli Bora
Department of Computer Science

OPERATING SYSTEMS

Case Study – Windows File System

Kakoli Bora
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau
5. Internet source

File System - Basics

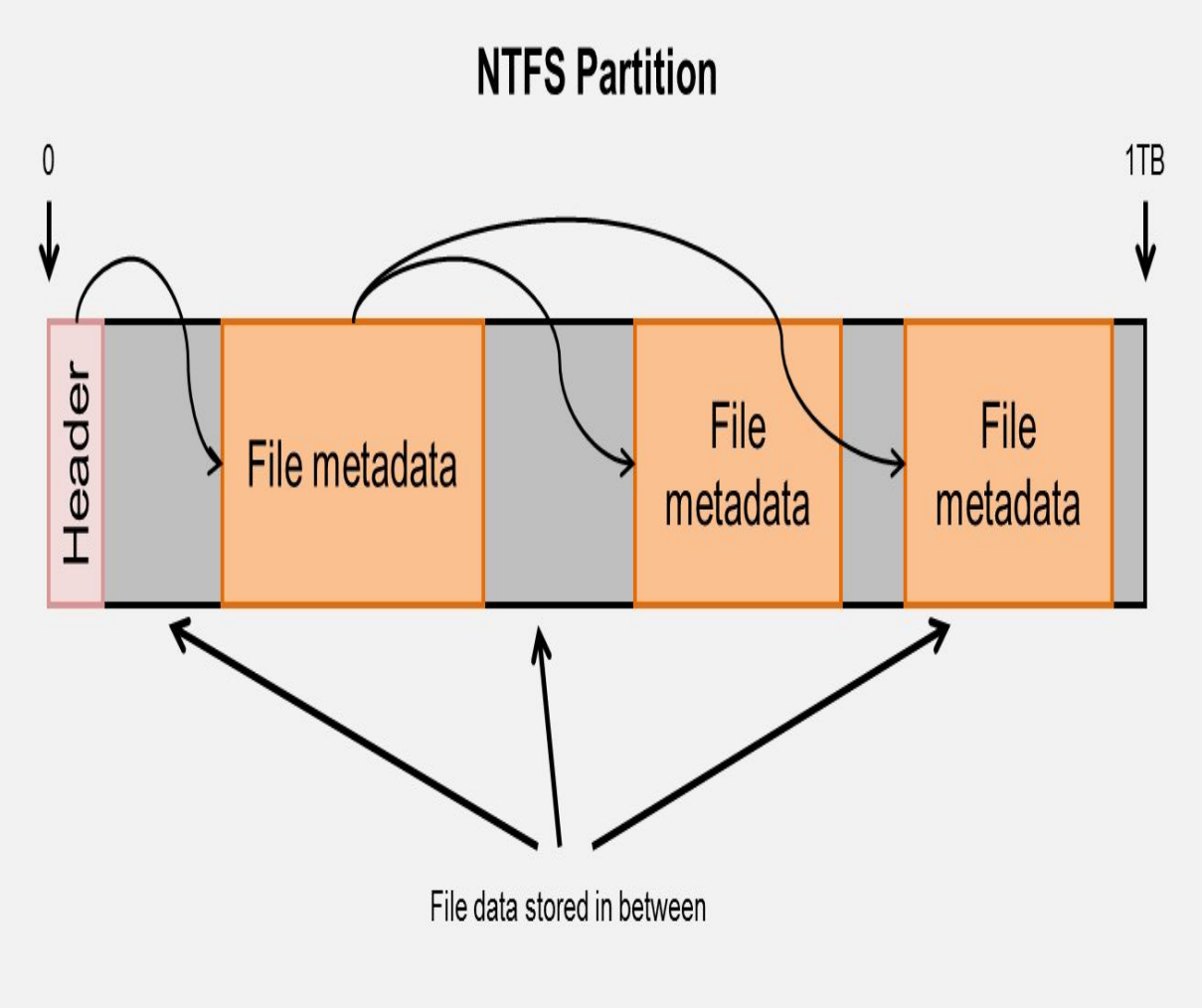
- FAT file system is still used for portability on other systems such as cameras, flash memory and external disks
- FAT file system does not restrict file access to authorized users.
- NTFS uses ACLs to control access to individual files and supports encryption.
- NTFS supports data recovery, fault tolerance, very large files and file systems, journaling, file compression, etc
- A file in NTFS is not a simple byte stream, as in MS-DOS or UNIX, rather, it is a structured object consisting of attributes
- Attributes like file name, creation time, descriptor, ACLs, etc

File System – Basics (Cont.)

- The fundamental structure of Windows file system (NTFS) is a *volume*
  - Created by the disk administrator utility
  - Based on a logical disk partition
  - May occupy a portions of a disk, an entire disk, or span across several disks
- All *metadata*, such as information about the volume, is stored in a regular file
- NTFS uses ***clusters*** as the underlying unit of disk allocation
  - A cluster is a number of disk sectors that is a power of two
  - The default cluster size is based on the volume size - 4 KB for volumes > 2 GB
  - Because the cluster size is smaller than for the 16-bit FAT file system, the amount of internal fragmentation is reduced

OPERATING SYSTEMS

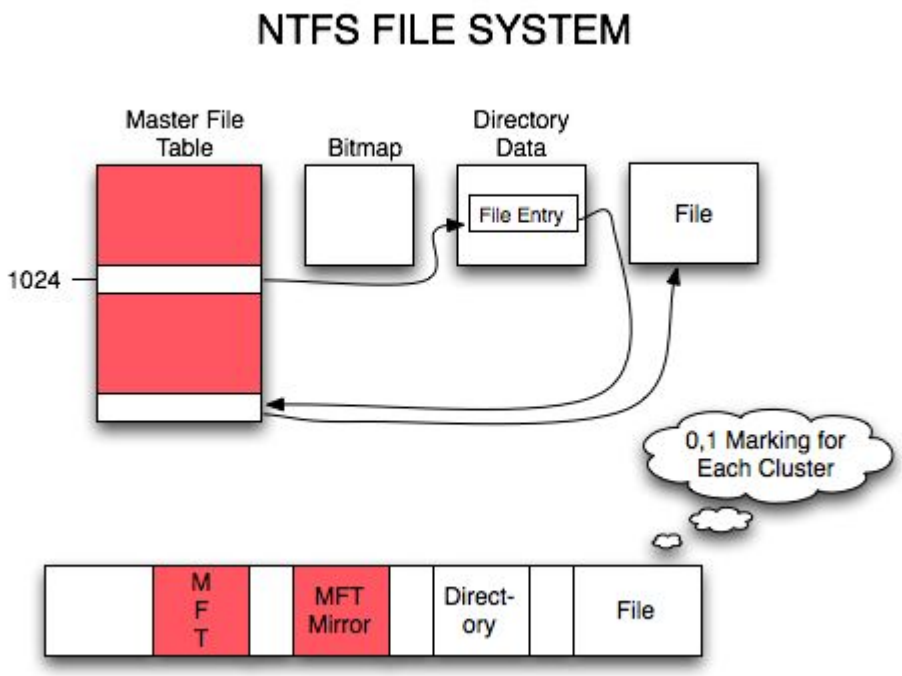
File System — NTFS (Cont.)



- NTFS, the filesystem used by all modern versions of Windows
- Instead of storing file metadata in tables scattered across the partition (usually to optimise hard disk seek times going from file metadata to actual file data), NTFS stores all the file metadata in a few large contiguous blocks.
- These blocks are collectively known as the MFT (master file table).
- Helps to quickly scan all the files in the partition to get each file's associated metadata. Instead of recursing through each directory manually and enumerating contents (like when computing summary statistics for directories)

OPERATING SYSTEMS

File System — NTFS (Cont.)



- Every file in NTFS is described by one or more records in an array stored in a special file called the Master File Table (MFT)
- Windows puts the Master File Table at the front of the drive, with a mirror file in the middle of the drive with only the first directory containing critical information.
- The Bitmap is used to determine allocation/non-allocation of clusters.

OPERATING SYSTEMS

File System — NTFS vs FAT

Features	NTFS	FAT32	FAT16	FAT12
Max Partition Size	2TB	32GB	4GB	16MB
Max File Size	16TB	4GB	2GB	Less than 16MB
Cluster Size	4KB	4KB to 32KB	2KB to 64KB	0.5KB to 4KB
Fault Tolerance	Auto Repair	No	No	No
Compression	Yes	No	No	No
Security	Local and Network	Only Network	Only Network	Only Network
Compatibility	Windows 10/8/7/XP/Vista/2000	Windows ME/2000/XP/7/8.1	Windows ME/2000/XP/7/8.1	Windows ME/2000/XP/7/8.1

File System — Internal Layout

- NTFS uses logical cluster numbers (LCNs) as disk addresses
- Physical disk offset in bytes = LCN x cluster size
- A file in NTFS is not a simple byte stream, as in MS-DOS or UNIX, rather, it is a structured object consisting of attributes
- Each file on an NTFS volume has a unique ID called a file reference.
  - 64-bit quantity that consists of a 48-bit file number and a 16-bit sequence number
  - Sequence number is incremented every time an MFT entry is reused, can be used to perform internal consistency checks (to catch a stale reference to a deleted file)
- The NTFS name space is organized by a hierarchy of directories; the index root contains the top level of the B+ tree
  - B+ tree is an extension of the B tree

OPERATING SYSTEMS

File System—B+ Tree vs. B Tree

B+ Tree	B Tree
Data is only saved on the leaf nodes.	Both leaf nodes and internal nodes can store data
Data stored on the leaf node makes the search more accurate and faster.	Searching is slow due to data stored on Leaf and internal nodes.
Deletion is not difficult as an element is only removed from a leaf node.	Deletion of elements is a complicated and time-consuming process.
Linked leaf nodes make the search efficient and quick.	You cannot link leaf nodes.

The drawback of B-tree used for indexing, however is that it stores the data pointer (a pointer to the disk file block containing the key value), corresponding to a particular key value, along with that key value in the node of a B-tree. This technique, greatly reduces the number of entries that can be packed into a node of a B-tree, thereby contributing to the increase in the number of levels in the B-tree, hence increasing the search time of a record.

B+ tree eliminates the above drawback by storing data pointers only at the leaf nodes of the tree.

File System - Recovery

- All file system data structure updates are performed inside transactions that are logged.
  - Before a data structure is altered, the transaction writes a log record that contains redo and undo information.
  - After the data structure has been changed, a commit record is written to the log to signify that the transaction succeeded.
  - After a crash, the file system data structures can be restored to a consistent state by processing the log records.

File System – Recovery (Cont.)

- This scheme does not guarantee that all the user file data can be recovered after a crash, just that the file system data structures (the metadata files) are undamaged and reflect some consistent state prior to the crash.
- The log is stored in the third metadata file at the beginning of the volume.
- The logging functionality is provided by the *log-file service which keeps track of free space in the log file*
  - *halts new I/O operations in case free space gets too low*

File System - Security

- Security of an NTFS volume is derived from the Windows object model.
- Each file object has a security descriptor attribute stored in this MFT record.
- This attribute contains the access token of the owner of the file, and an access control list that states the access privileges that are granted to each user that has access to the file.

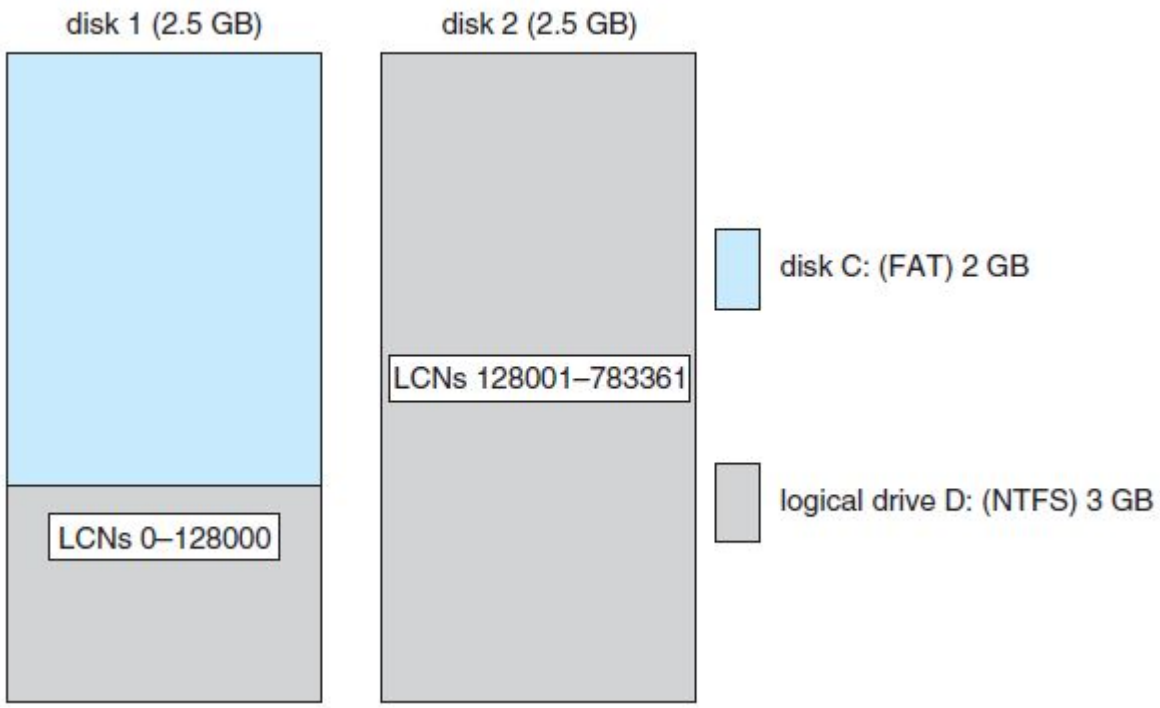
OPERATING SYSTEMS

Volume Management and Fault Tolerance

- **FtDisk**, the fault tolerant disk driver provides several ways to combine multiple SCSI disk drives into one logical volume
- Logically concatenate multiple disks to form a large logical volume, a *volume set*
- Interleave multiple physical partitions in round-robin fashion to form a ***stripe set*** (also called RAID level 0, or “disk striping”)
    - FtDisk uses a stripe size of 64 KB – 1st 64 KB of the logical volume are stored in the 1st physical partition, 2nd 64 KB in the 2nd physical partition, and so on
    - Windows also supports RAID level 1 (mirroring) and RAID level 5 (*stripe set with parity)*
- Disk mirroring, or RAID level 1, is a robust scheme that uses a ***mirror set*** — two equally sized partitions on tow disks with identical data contents
- To deal with disk sectors that go bad, FtDisk, uses a hardware technique called ***sector sparing*** *(i.e. formatting a disk drive leaves extra sectors unmapped as spares)* and NTFS uses a software technique called ***cluster remapping*** *(i.e changes the pointers in the MFT from bad block to unallocated block)*

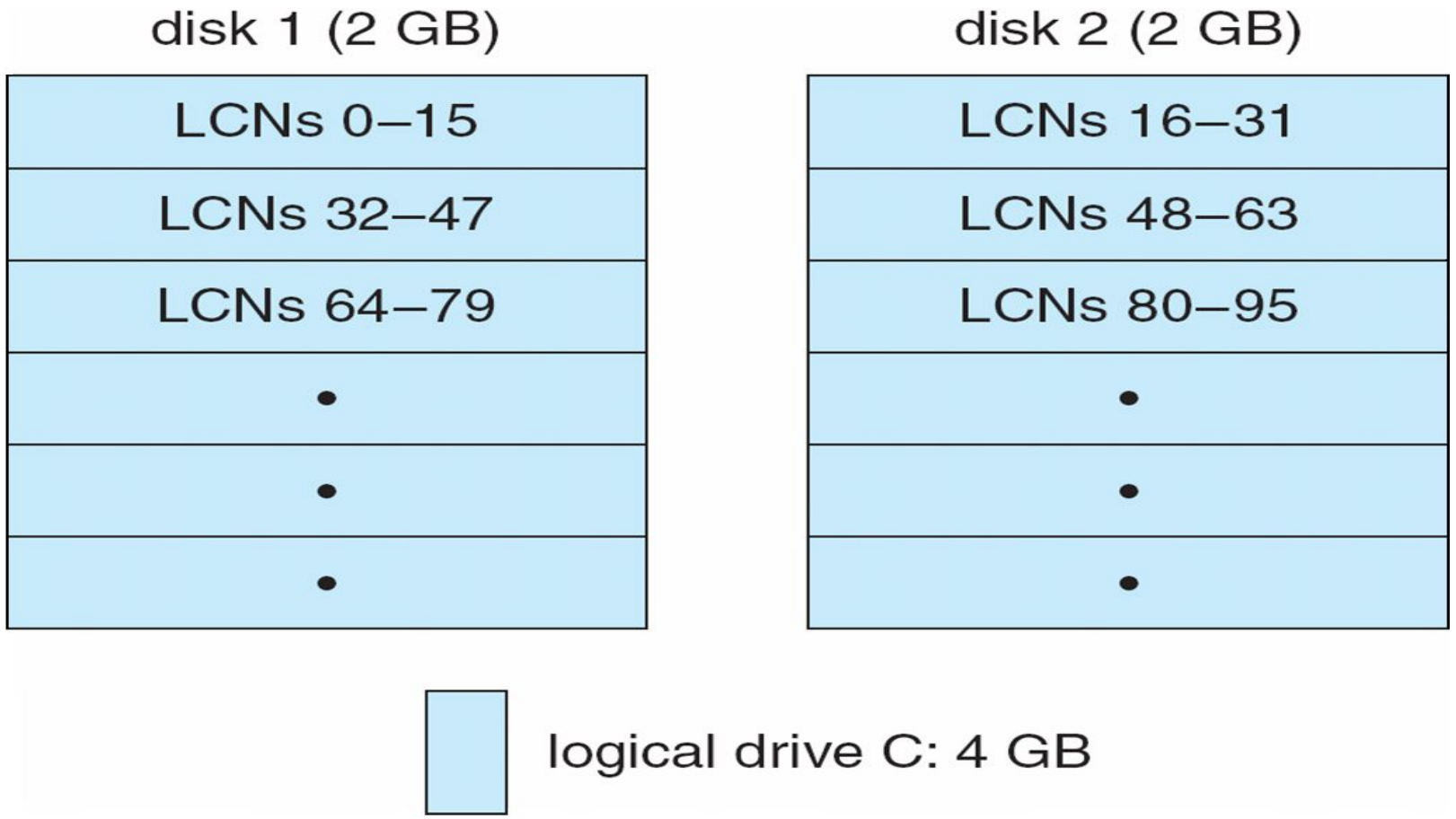
OPERATING SYSTEMS

Volume Set On Two Drives

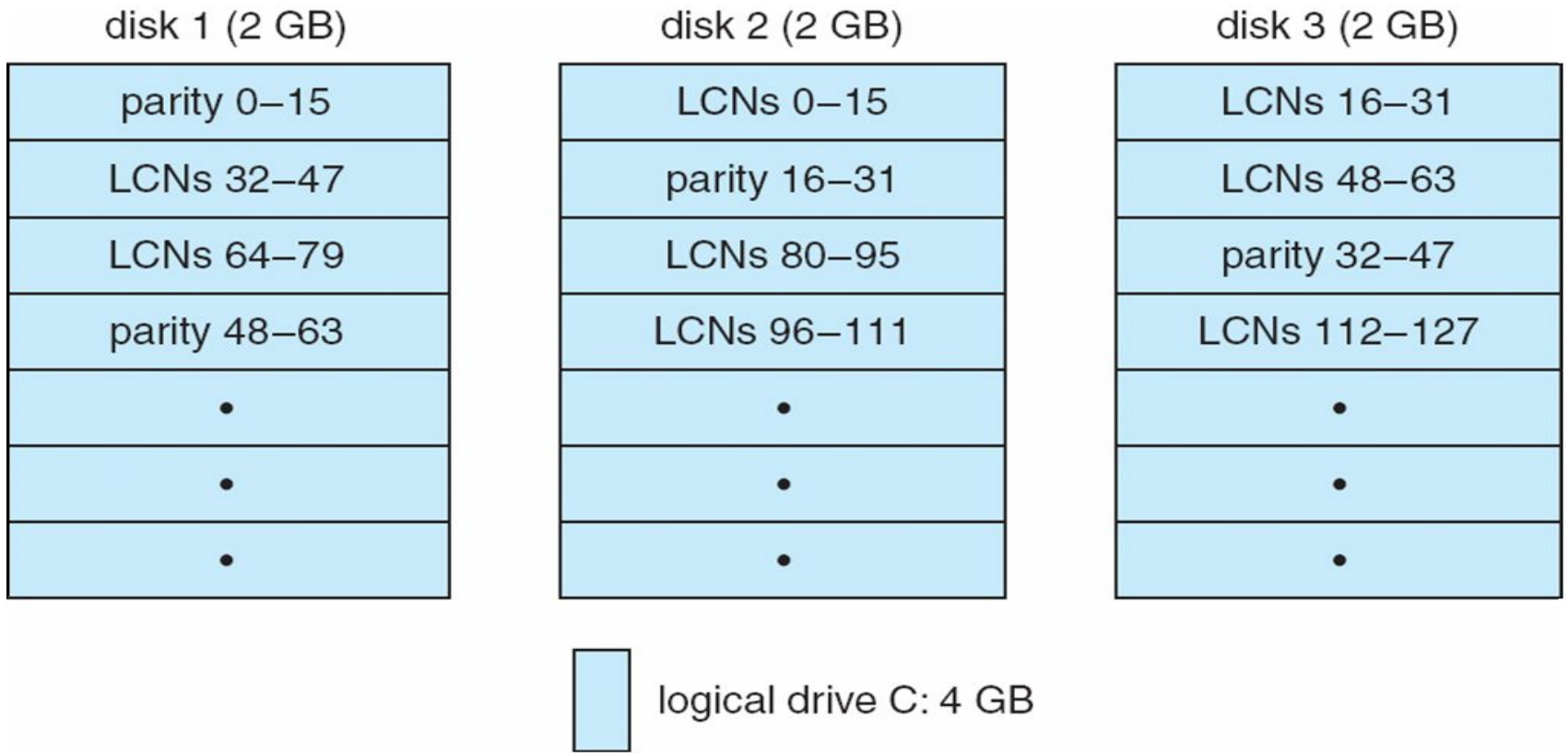


Logical volume or a *volume set can consist of up to 32 physical partitions*
A volume set can be extended by extending the bitmap metadata on the NTFS volume to cover the newly added space (bitmap contains information about free or used data blocks/clusters on a volume)
Fig 1==> 128000 x 4 KB = 0.5 GB
Fig 2 => 655360 x 4 KB = 2.5 GB

Stripe Set on Two Drives



Stripe Set With Parity on Three Drives



Mirror Set on Two Drives

File System — Compression

- To compress a file, NTFS divides the file's data into *compression units*, which are blocks of 16 contiguous clusters.
  - To improve performance when reading contiguous compression units, NTFS pre-fetches and decompresses ahead of the application requests.
- For sparse files, NTFS uses another technique to save space.
  - Clusters that contain all zeros (ie. never been written) are not actually allocated or stored on disk.
  - Instead, gaps are left in the sequence of virtual cluster numbers stored in the MFT entry for the file.
  - When reading a file, if a gap in the virtual cluster numbers is found, NTFS just zero-fills that portion of the caller's buffer.

File System (Contd.)

- Mount Points, Symbolic links, Hard Links
  - Hard links: A single file has an entry in more than one directory of the same volume
- NTFS Journal – describes all changes that have been made to the file system.
  - Used by services such as User-mode (to identify what files have changed), search indexer (to re-index files) and file-replication (to replicate files across the network)
- Snapshots to create a shadow copy of volume for backup (and recovery of files if accidentally deleted)

THANK YOU

Kakoli Bora

Department of Computer Science Engineering

k_bora@pes.edu