Operating System Concepts
Ch. 12: Mass Storage Systems

Silberschatz, Galvin & Gagne

Universiteit Leiden
The Netherlands

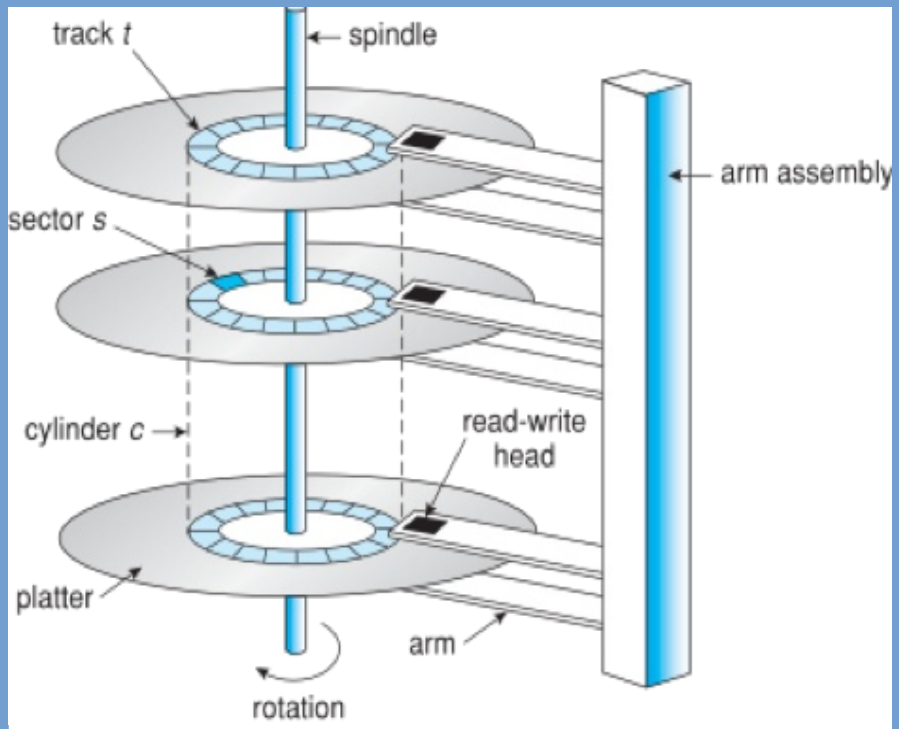
Aim

- We will conclude our discussion on file systems by investigating devices on which these file systems are stored.
- Two systems are in common use:
  - Flash / solid state drives (SSD); in use in laptops, tablets, phones, embedded devices.
  - Hard drives; in use in workstations, server systems.
- Past: magnetic tape
  - But still used in very large-scale back ups and archiving.

Hard Drives

- A hard drive consists of a spindle to which multiple magnetic platters are attached.
  - Bits are written in magnetic fashion on these platters.
- Main problem of hard drives: moving parts!
  - The spindle with platters spins. Speed from 5400 RPM (cheap laptops) to 15000 RPM (enterprise hard disk).
  - A disk arm moves the read/write heads over the disk.
  - When the disk is turned off, the head must be parked. Imagine a G-shock when the head is not parked, the head will hit **and damage** the disk (*head crash*).

Hard Drives (2)



Source: Silberschatz et al., Operating System Concepts, 9th Edition

Hard Drives (3)

- Hard drives are connected to a host controller through a certain interface.
  - Current: SATA, SAS, FibreChannel, USB
  - Past: (E)IDE, SCSI, Firewire
  - Type of bus determines maximum achievable transfer speed.
- *Transfer rate*: rate at which data is transferred between device and computer.
- For hard drives, the transfer rate is highly influenced by the *positioning time* (or *random-access time*) which consists of two components:
  - *Seek time*: the time it takes to move the disk arm to the desired (cylinder) location. Order of magnitude: milliseconds (!).
  - *Rotational latency*: time we have to wait for the desired disk sector to appear below the read/write head.

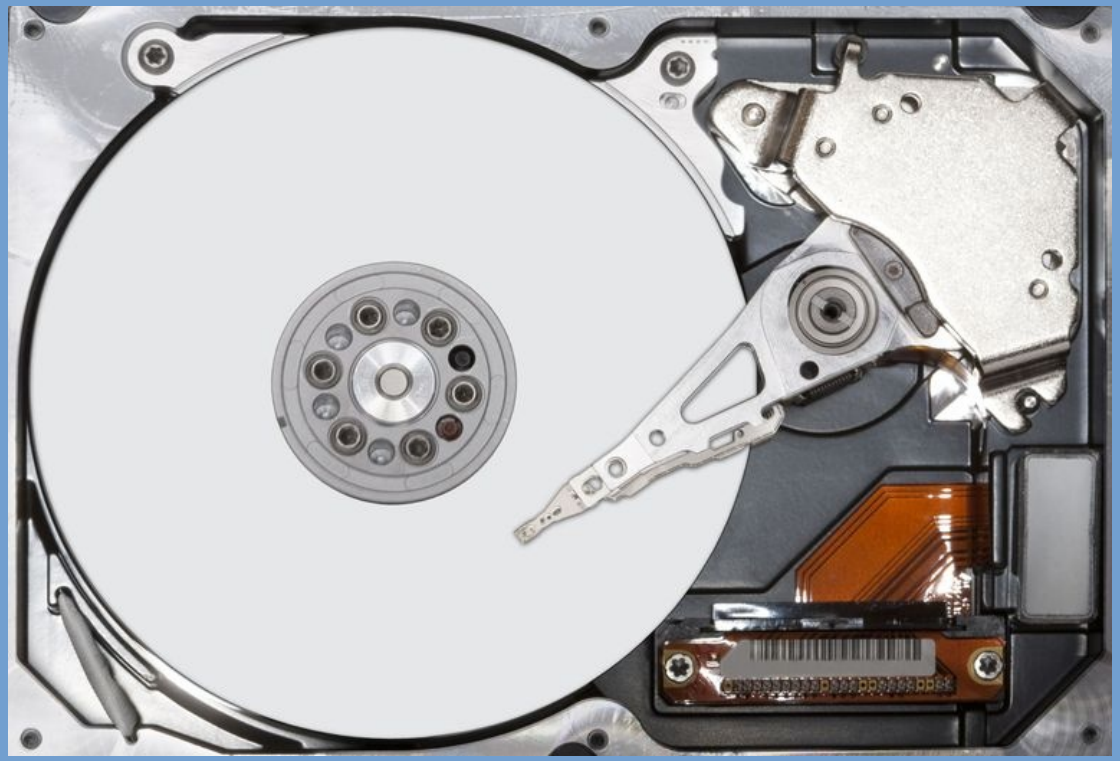
Hard Drives (4)

- We have come a long way.
  - 1956: first commercial hard drive, IBM Model 350 disk storage system.
  - 5M (7 bit) characters

Source: Silberschatz et al., Operating System Concepts, 9th Edition

Hard Drives (5)

- Modern hard drives available in 2.5” or 3.5” package.
- Available up to ~10 TB.



Hard Drives (6)

- We address hard drives as a large 1-d array of logical blocks.
  - But note that this differs from the physical layout of the disk (multiple platters).
- So a translation from a logical 1-d address to physical address is needed.
  - The sectors on the disk are numbered sequentially.
  - We start at the first sector of the first track on the outermost cylinder: sector 0.
  - From there we continue numbering through the same track, then the tracks in that cylinder and then the cylinders from outside in.
  - Most hard disks have some spare sectors that can take over in case of bad blocks. This is all handled in the hard drive internally by the firmware, so transparent to OS.

Solid State Drives

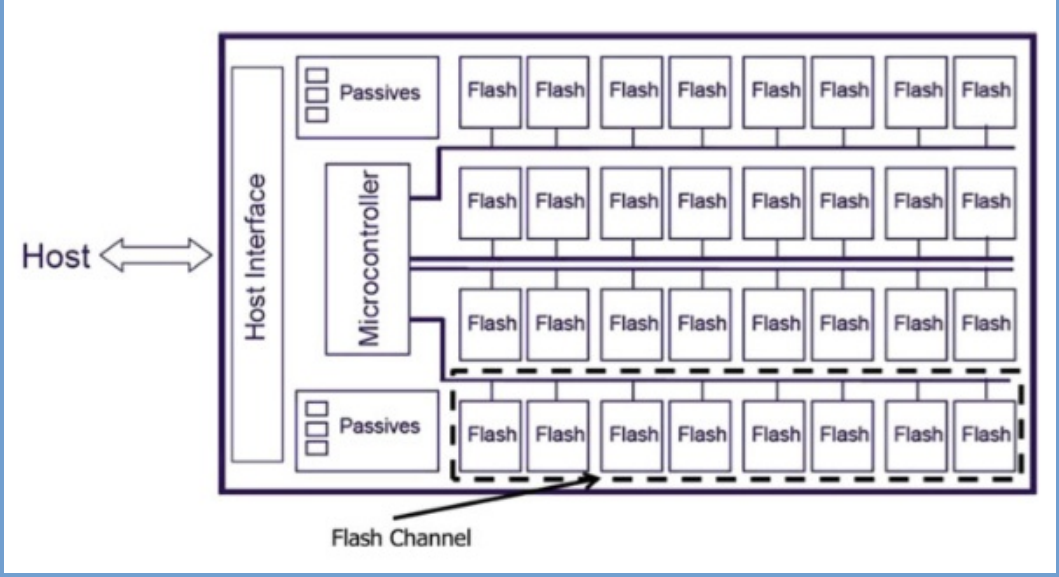
- In the last 5 – 8 years, solid state drives have greatly increased in popularity.
- These drives are based on non-volatile memory.
  - Advantage: no moving parts!
    - Less prone to mechanical failure.
    - No seek time, rotational latency.
  - Disadvantage: type of memory used wears out.
  - Disadvantage: more susceptible to controller failures.
- Started out as flash chips packaged in a 2.5” hard drive package with SATA bus.
- Larger capacity SSD (> 1 TB) are significantly more expensive than hard drives.
  - In large capacity systems, SSDs are used as cache.
  - Can also construct “Fusion Drives”

Solid State Disks

- These days available in different form factors, different buses (SATA, SAS, NVMe PCI).

SSD architecture

- SSDs are built from collections of flash memories.
  - The chips are organized into separate channels.
  - Communication on a channel can be interleaved, when one chip is busy, we can continue communicating with another chip on that channel.
- A microcontroller manages the flash memories and communication with the host.



Source: K. Eshghi and R. Micheloni, SSD Architecture and PCI Express Interface, Springer 2013.

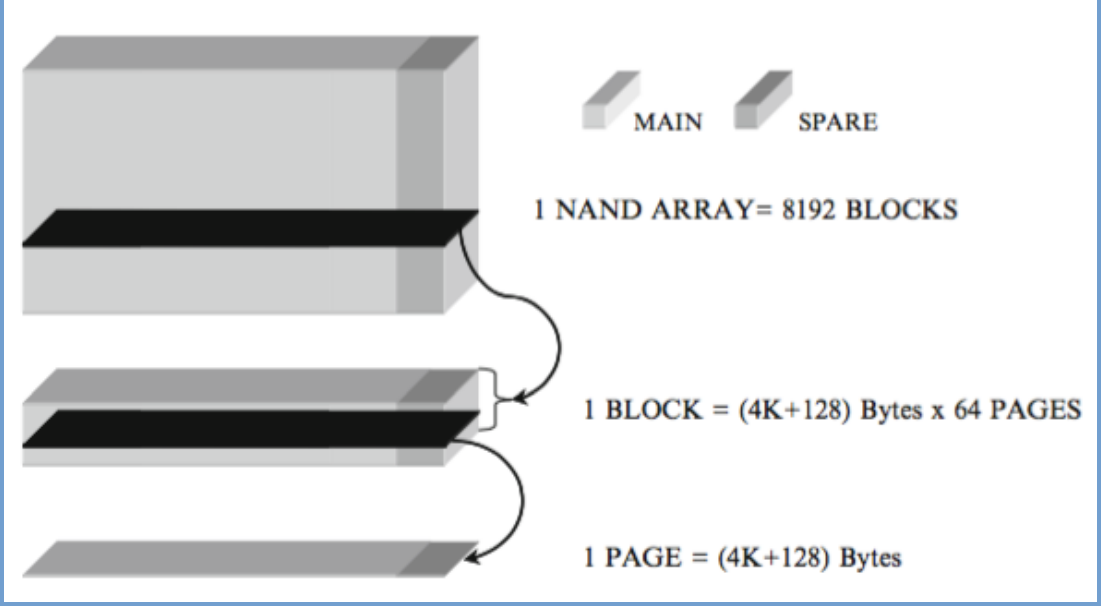
SSD architecture (2)

What is there to manage?

- Translation of host drive geometry (in particular for SATA SSD) to SSD architecture.
  - In fact, although the host thinks it is writing to contiguous blocks, this may not be the case as blocks are dynamically remapped by the SSD microcontroller.
- Wear leveling
  - Flash memory has a limited number of rewrite cycles before it breaks; so distribute the load uniformly.
- Bad block management
  - Maintain a list of bad blocks; already initialized during testing at the factory.

SSD architecture (3)

- The memories are grouped into *pages*.
- Pages are again combined into *blocks*.
- A collection of blocks forms the entire memory array.



Source: K. Eshghi and R. Micheloni, SSD Architecture and PCI Express Interface, Springer 2013.

SSD architecture (4)

- Read and write operations can be done at a page level.
  - (So not at the level of individual memories!).
- Now here's an interesting catch:
  - Before you can write a page, it MUST be empty.
  - If it's not empty, it must first be erased.
  - But: you can only erase entire blocks (!). (Hardware limitation)
- So, page overwrite implies:
  - Read the entire block into a buffer.
  - Erase the block.
  - Write all pages back.

TRIM

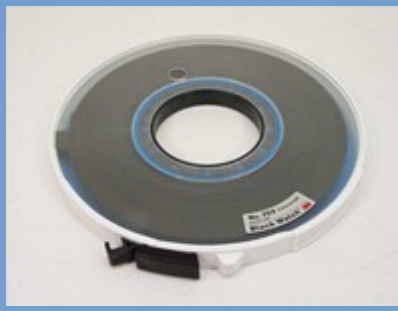
- For each overwrite, many more write operations have to be done (write amplification).
- This is not a very good idea, considering the limited number of cycles.
- What happens if a file is deleted?
  - The OS only updates the file system data structures.
  - It can rewrite the blocks when a new file is allocated there, not a problem for magnetic disks.
  - What does this mean for SSD?

TRIM (2)

- Say we have a block of which all pages, except one, contain data of **deleted** files.
- If we overwrite a page on this block, all pages are rewritten, also these pages that contain **deleted** data.
- To alleviate this, the TRIM command was added to the ATA command set (UNMAP in SCSI).
  - OS support is needed!
  - On file delete, the OS tells the SSD what blocks (pages on the SSD) were deleted. The SSD can then decide to no longer preserve this data in future overwrites of neighboring pages.

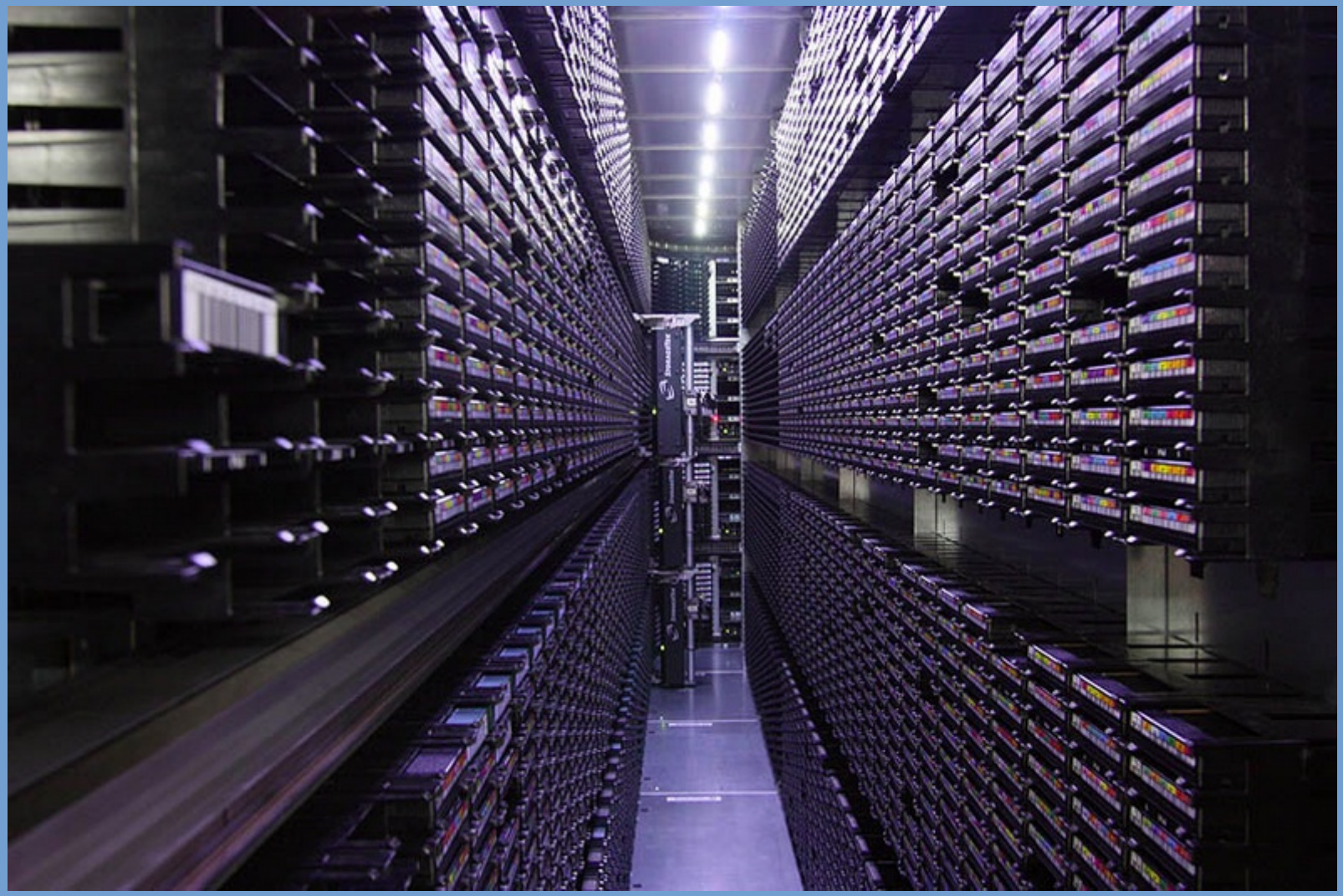
Magnetic Tape

- Magnetic Tape was the secondary storage medium of choice before hard drives became wide spread.
  - Started out as open tape spool (image right)
  - Later evolved to cartridges.



- A modern LTO-6 cartridge can store 2.5 TB of data.
  - Advances in technology still being made (density increase).
  - Now mainly in use for backups & archiving.
- Random access is slow, you need to wind the tape.
- Once the data is found, read/write speeds are quite fast, ~150 MB/sec.

Tape silos



Networked Disks

- In this age, we want networked access to disks.
- Two main methods:
  - *NAS: Network Attached Storage*

    A disk is made accessible over a regular LAN.

  - *SAN: Storage Area Network*

    A special network dedicated to storage.

Network Attached Storage

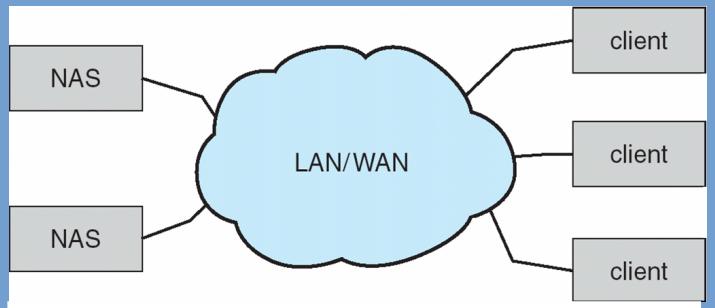
- Export a disk over a local area network (LAN)
- Can be done in two ways:
  - ***File-based***, with protocols such as NFS, CIFS.

    Remote host is accessed using requests for particular files. Client has no knowledge about underlying file system.

    Multiple clients may access the exported files at the same time.
  - ***Block-based***, with iSCSI.

    Remote host is accessed using requests for particular disk blocks. Client determines file system.

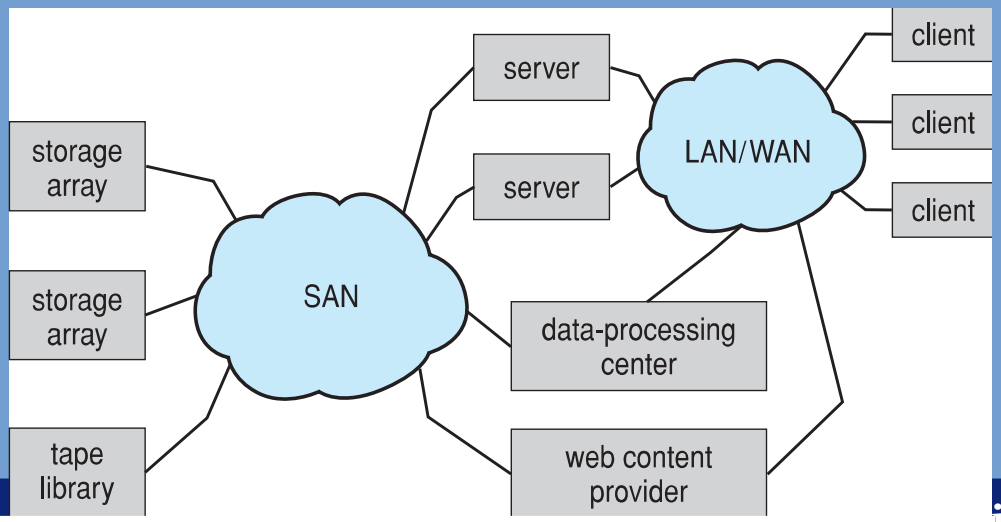
    Only one client can access the disk at a time (just with 'regular' disks).



Source: Silberschatz et al., Operating System Concepts, 9th Edition

Storage Area Networks

- A SAN is only seen in large (enterprise) deployments.
- Storage devices (arrays of hard drives, tape robots) are connected to a dedicated network.
- Servers are connected to this network as well.
- Mappings between server and storage device can be made dynamically.
  - Example: when a server fails another server can be started and be configured to access the same storage device. This other server can now take over duties.
- Use of fiber allows long distances between server and storage device.



Source: Silberschatz et al., Operating System Concepts, 9th Edition

Improving Disk I/O Performance

- The OS is responsible for efficient and effective use of the available disks.
- What aspects of disk I/O performance can we measure and improve?
  - *Access time*: how much time it takes to access certain data. Let's assume time from when request for data was posted until first block of the data becomes available.

    Closely related to *seek time*. *Seek time* is proportional to *seek distance*.
  - *Disk bandwidth*: the total number of bytes transferred divided by the first request for I/O service and completion of last I/O transfer.

Improving Disk I/O Performance (2)

- Since access time is related to seek time, which is in turn proportional to seek distance, we try to improve access time by minimizing seek time.
  - So, try to minimize seek distance.
- As for disk bandwidth, note that the time span between first request and completion of last request is included.
  - By shortening this time span, we improve disk bandwidth.
  - Time span can be shortened by performing less disk seeks, or decreasing seek distance covered.

Improving Disk I/O Performance (3)

- Laptops, desktops, workstations often have an *idle* disk.
  - A lot of the time the disk is sitting there waiting (unless when launching a new application).
  - As soon as a disk request comes in, it can be directly sent to the disk.
  - Not much room for optimization in this case, apart from optimizing file I/O of application launches (these files could be organized on contiguous blocks on a disk).
- What about server systems?
  - Many server systems are I/O bound: webservers, mailservers, database servers ...
  - Different transactions are processed in parallel. Different transactions need different data.
  - We get a list (rather a queue) of disk requests that compete to get access to the disk.

Disk Scheduling

- Given such a queue of I/O requests for a given disk, can we reorder the requests in order to improve disk performance?
  - So: shorten access time, improve disk bandwidth?
  - Main idea: minimize seek time/distance.
  - This is known as *disk scheduling*.
  - Several disk scheduling algorithms exist.
- Notes:
  - The analysis is true for one or multiple platters (since the heads always move to the same cylinder).
  - Many hard drives have internal buffers and request queues. We ignore the scheduling done by the hard drives themselves for now.

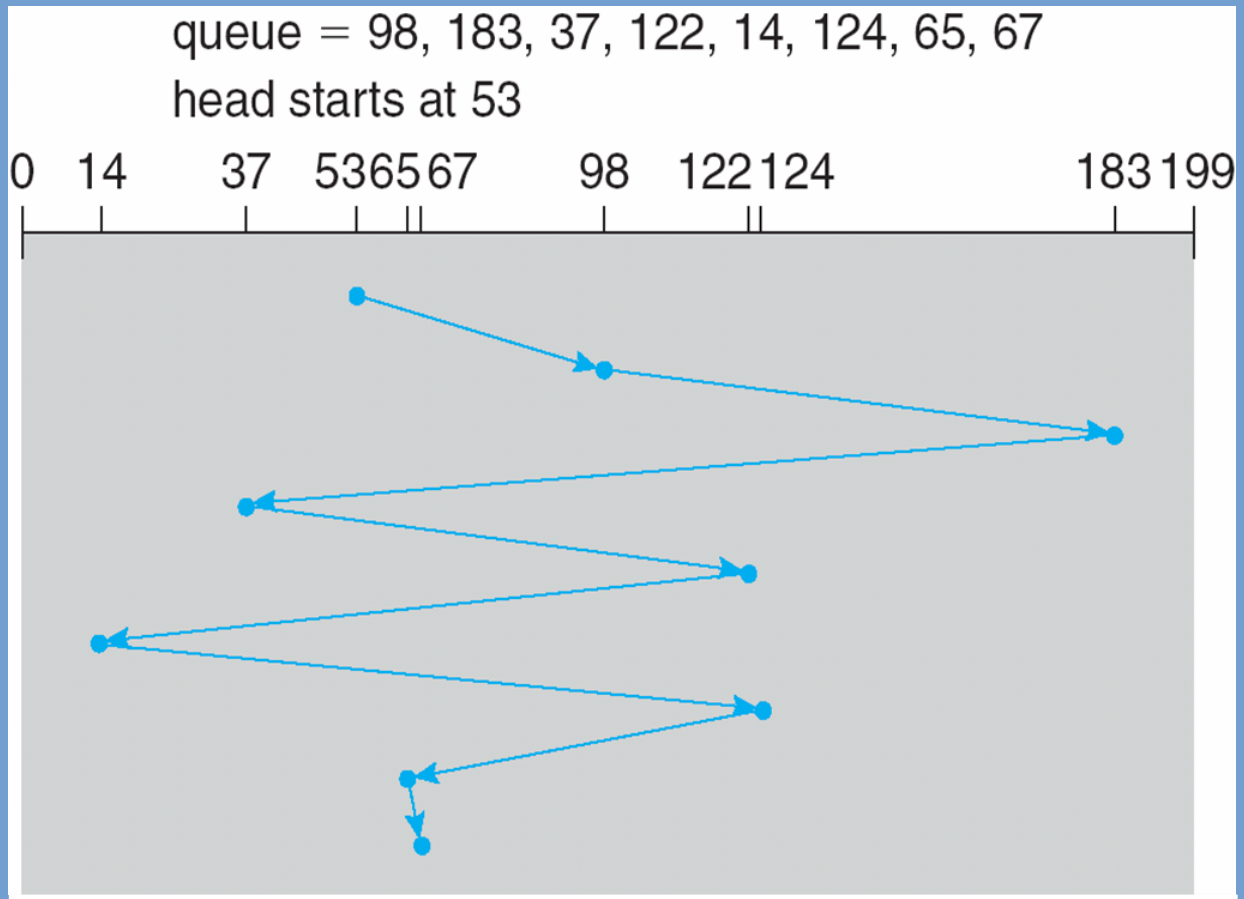
Disk Scheduling (2)

- For the examples that will follow we consider:
  - A disk with cylinders 0 – 199.
  - A request queue consisting of requests for

    98, 183, 37, 122, 14, 124, 65, 67
  - The initial position of the head is 53.

FCFS: First Come First Serve

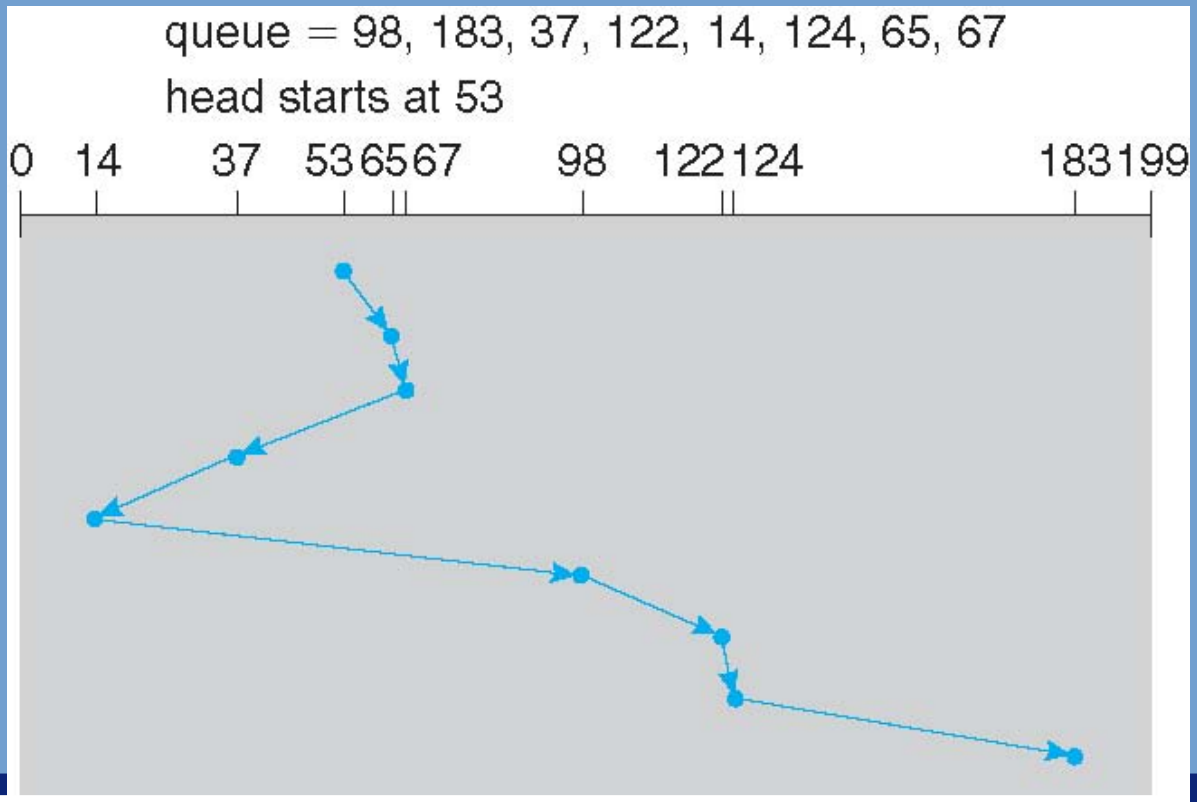
- The total distance covered by the head (head movement) is 640 cylinders.



Source: Silberschatz et al., Operating System Concepts, 9th Edition

SSTF: Shortest Seek Time First

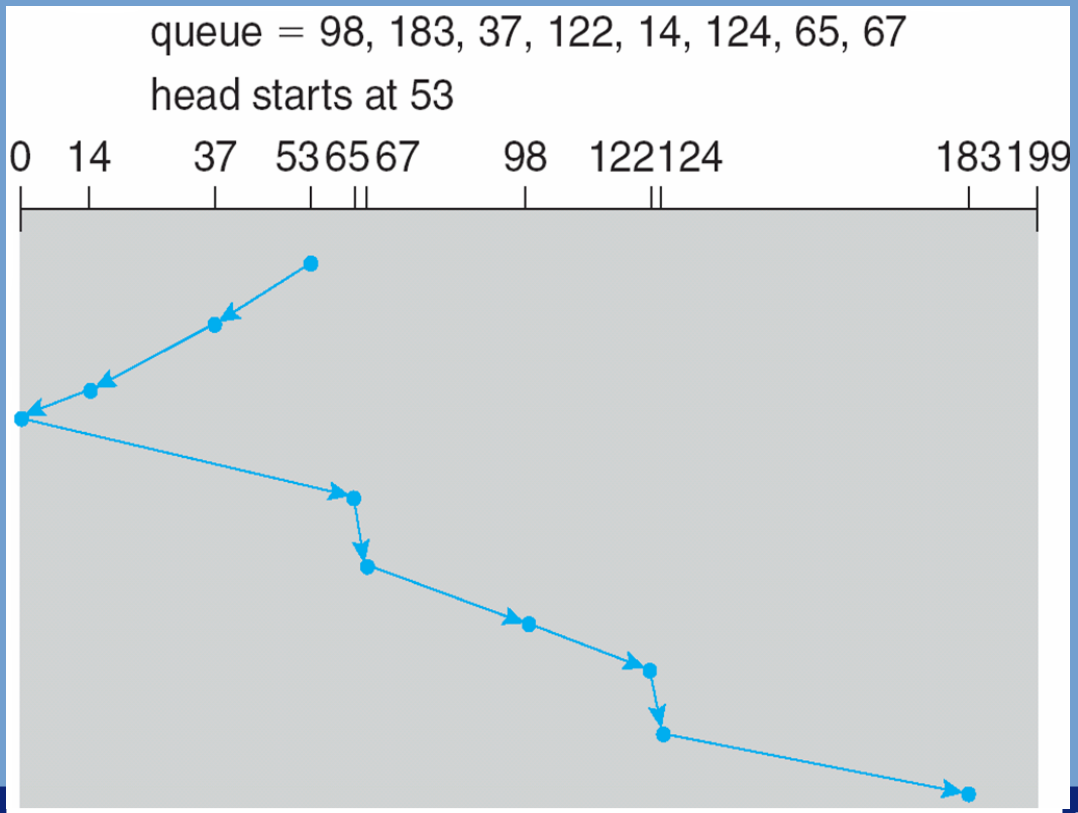
- Scan queue each time for request at shortest distance from current head position.
  - Important: potential of *starvation*!
- Distance covered in this example: 236 cylinders.



Source: Silberschatz et al., Operating System Concepts, 9th Edition

SCAN

- Handle all requests in one direction until you reach the end of the disk.
- At that point the disk arm reserves direction and moves to the other end of the disk, handling requests along the way.
- Also known as the *elevator* algorithm.



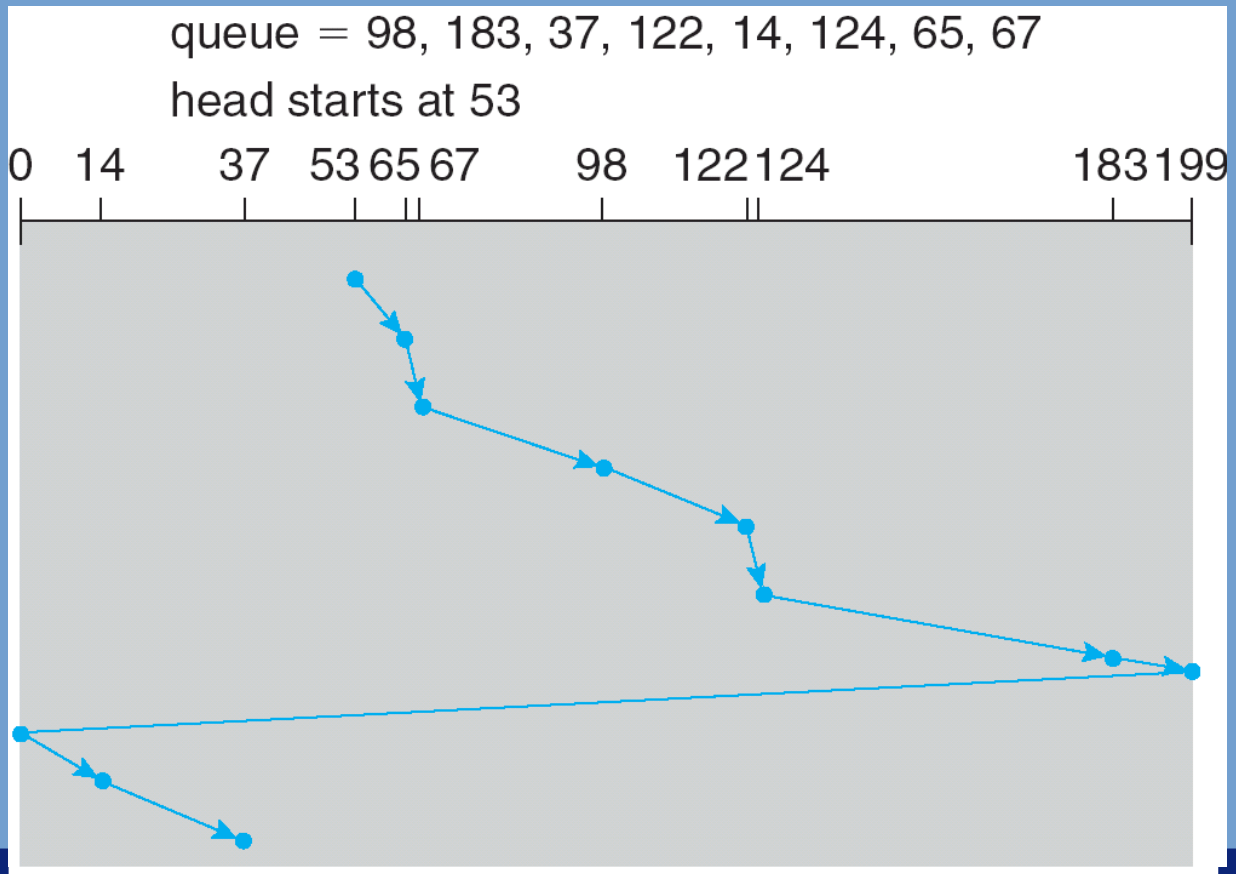
Source: Silberschatz et al., Operating System Concepts, 9th Edition

C-SCAN

- A problem with SCAN is the following:
  - Consider the arm is moving in the upwards direction, currently at location 53.
  - A new request comes in for cylinder 45.
  - This request will not be serviced until the arm has reached the end of the disk, reversed and handled all requests on the way back.
  - E.g. a request at cylinder 180, in case the arm has just passed that cylinder in the upwards direction, will be serviced earlier in time (quickly after head reversal).
- C-SCAN solves this by not handling requests in the downwards direction. When the end of the disk is reached, the disk arm is immediately moved to the beginning of the disk again.
  - This results in more uniform wait times.
  - ***Circular*** SCAN.

C-SCAN (2)

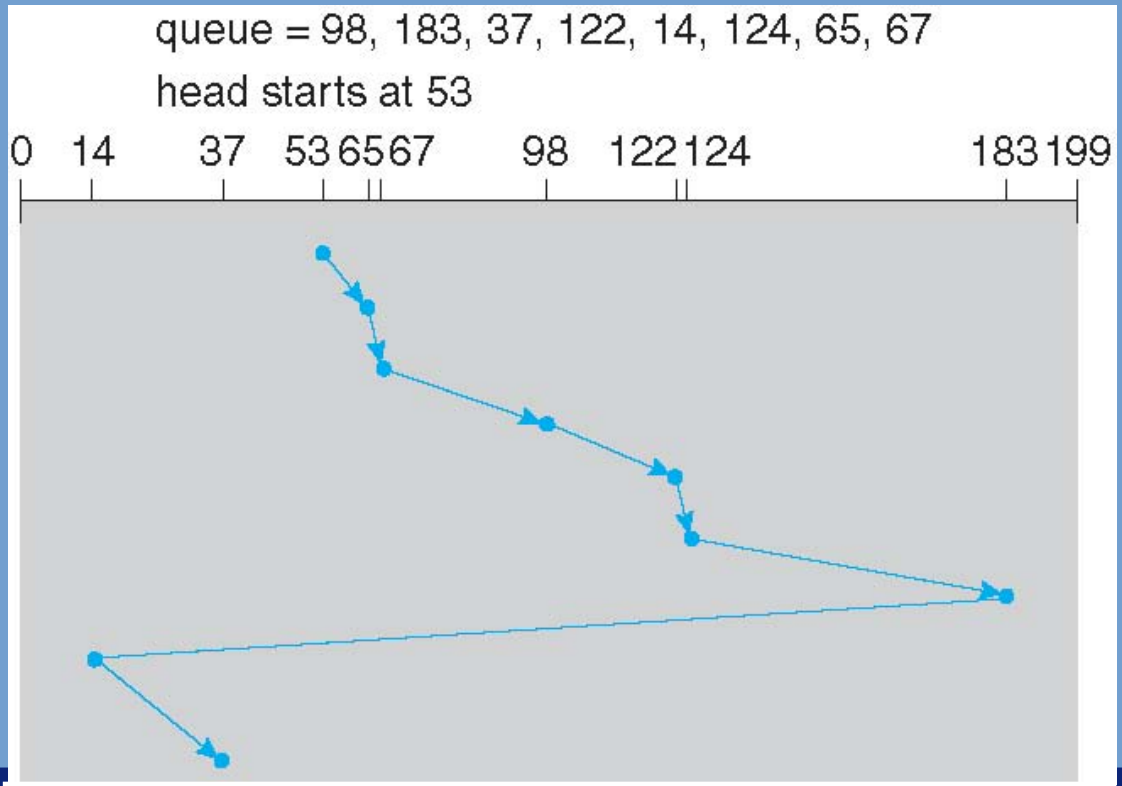
- C-SCAN example:
  - Note move of disk arm from end to beginning of disk.



Source: Silberschatz et al., Operating System Concepts, 9th Edition

LOOK, C-LOOK

- LOOK: do not visit beginning and end of disk, instead reverse direction at maximum/minimum request in the queue.
- C-LOOK: circular version (cf. C-SCAN).
- C-LOOK example:



Source: Silberschatz et al., Operating System Concepts, 9th Edition

Which algorithm to use?

- SSTF; commonly used but could cause starvation.
- For systems with heavy disk loads, SCAN and C-SCAN perform better.
- Disk scheduling does not make much sense for SSDs, often disabled in that case.
- Many systems allow you to choose an algorithm for each disk in the system.
  - Some systems also allow you to insert new algorithms in the form of kernel modules.

Which algorithm to use? (2)

- Also consider that file system design has a big impact on disk performance!
  - For instance the way blocks for files are allocated. Contiguous allocation: less disk seeks when reading.
  - Where do you store file metadata? Scattered across the disk? Centered around a single location?
- Other issues:
  - It is hard to optimize for rotational latency. Difficult to calculate. Does OS know current position of disk head at any time?
  - Influence of queuing that is performed in the disks itself. Does this negate the scheduling performed by the OS?

RAID Systems

- RAID: Redundant Array of Inexpensive Disks
  - Sometimes “Inexpensive” is replaced with “Independent”
- Why?
  - Build larger drives by combining separate drives into one.
  - Store data redundantly, a hard drive may then fail without causing data loss.
  - Performance, we can read/write from/to multiple disks in parallel.

RAID Systems (2)

- RAID systems can be fully implemented in software.
- However, for server systems separate hardware RAID cards are common:
  - RAID configuration to be entered into BIOS of this RAID card.
  - OS will not see individual drives, but rather the “combined” drives that are configured in this BIOS.
  - RAID card is equipped with small CPU to offload parity calculations (see later) from host CPU.
  - RAID card optionally equipped with battery-backed DRAM or flash (NVRAM) to store intermediate buffers.
    - This allows a write-back mode to be enabled, in which case the OS is informed that a write has been completed before the write actually made it to the physical disks.
    - Buffer remains in DRAM or NVRAM. In case of power failure this data is not lost and written to disks as soon as the power comes back online.

RAID Systems (3)

- Different RAID “levels” are distinguished.
- Most are based on block striping. The data is striped in blocks of, for example, 64 KB (often configurable).
- Levels:
  - *RAID0*: data is striped by alternating over the disks in the RAID0 set. Important: NOT REDUNDANT!

    Why is this used? For speed, e.g. video editing setups.
  - *RAID1*: each stripe is stored on all disks in the RAID1 set. Mirrored stripes. If a disk fails, no data is lost. RAID1 sets are often pairs of disks.

RAID Systems (4)

- Levels (continued):
  - *RAID2*: bit-level striping, redundancy through ECC as also used in memory (RAM) systems. Not commonly used.
  - *RAID3*: byte-level striping. Disks have to spin in sync. Not commonly used.
  - *RAID4*: striping with separated & dedicated parity disk.

    Any disk may be lost (but only one) and contents may be rebuilt using the other disks.

    Parity disk is more often used than other disks and therefore more likely to fail.

Parity

- **Odd parity**
  - 0 1 1 0 → 0 1 1 0 1
  - Added parity bit causes number to the odd.
  - We can lose any bit and still recover, why?
- **Even parity**
  - 0 1 1 0 → 0 1 1 0 0

RAID Systems (5)

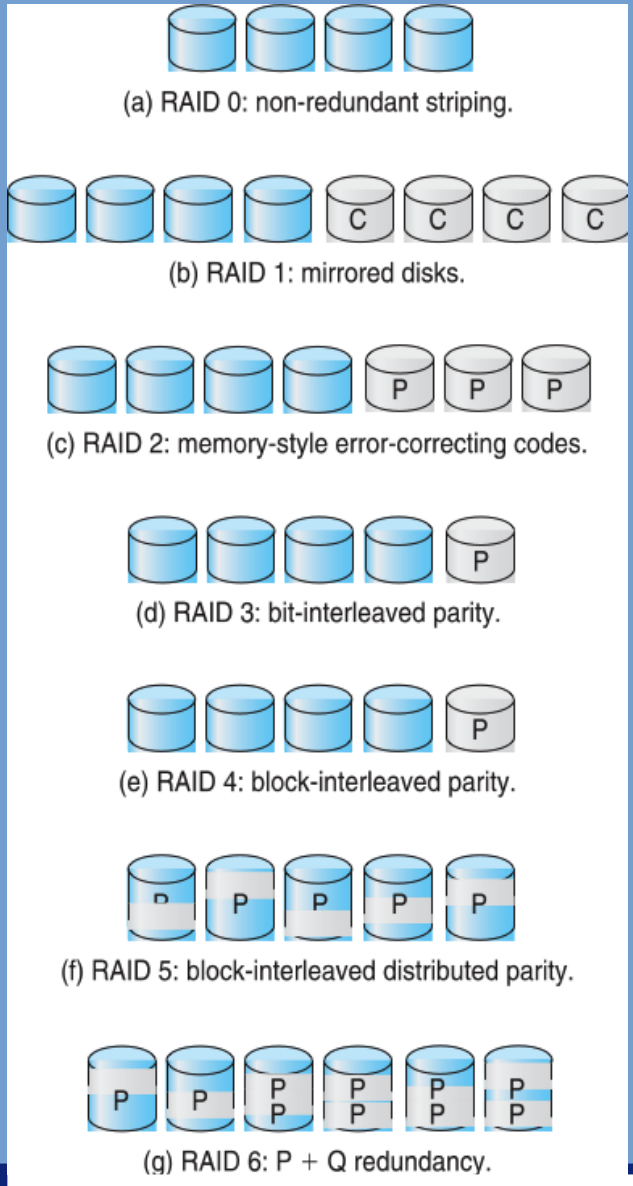
- Levels (continued):
  - *RAID5*: same as RAID4, but the parity is now distributed over all disks, instead of using a single parity disk.

    This evens out the load on the disks.
  - *RAID6*: two parity blocks are stored (instead of one).

    Now two disks may fail, and the contents may still be recovered.

RAID Systems (6)

- Schematic & comprehensive overview of all RAID levels.



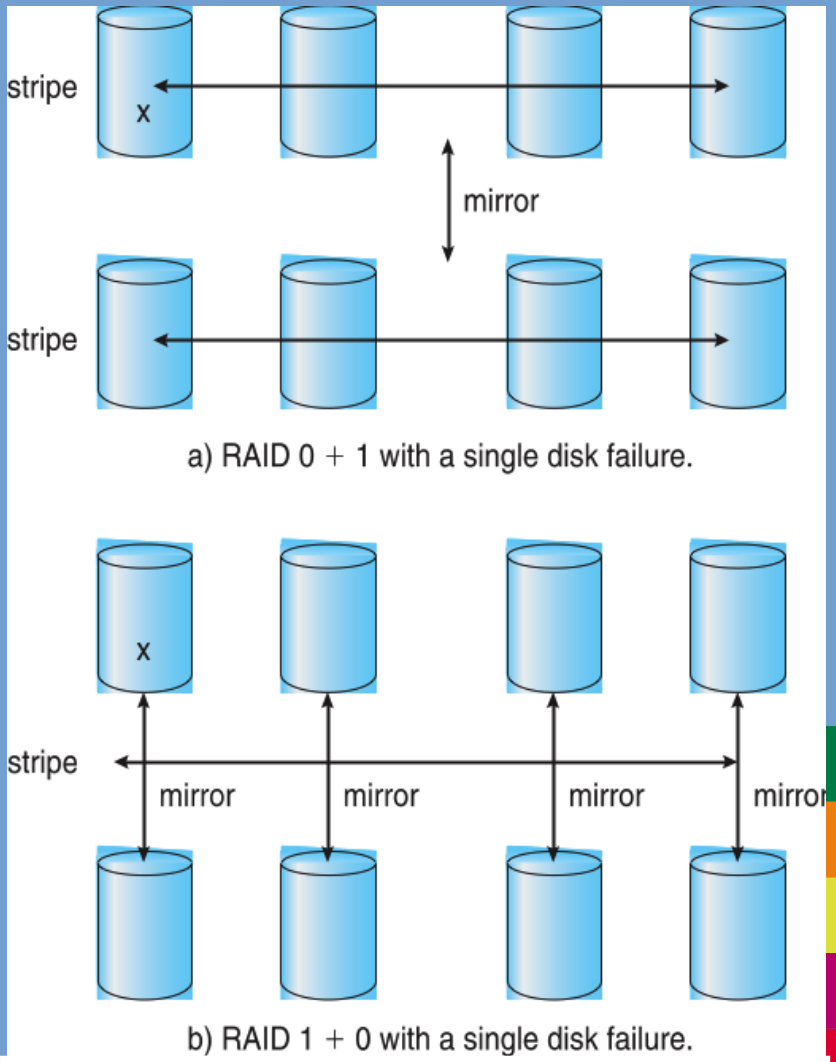
Source: Silberschatz et al., Operating System Concepts, 9th Edition

RAID Systems (7)

- What happens when a disk fails?
  - Controller detects failure, usually starts beeping and sends e-mail.
  - Broken disk must be replaced. As soon as this is done a rebuild is started automatically. The content of the old disk is reconstructed by reading the other disks and this is stored on the new disk.
  - What if another disk fails during the rebuild? In case of RAID5: all data will be lost. In case of RAID6: safe until another disk fails.
- You often want to start a rebuild as soon as possible.
  - To this end *hot spare disks* are used. This is a disk that is already installed in the system but is not used. As soon as a disk fails, the hot spare will automatically take over and a rebuild is initiated.
- Disclaimer: RAID is NOT a backup. An array may still fail. A building may still burn. File system may still be corrupted. Backups to other storage devices remain important.

RAID Systems (8)

- Two other often used levels are combinations of RAID0 and RAID1:
  - RAID01: mirror of striped sets.
  - RAID10: striping over mirrors.
- High performance & redundancy at the cost of more disks.
- RAID10 regularly advised for database servers (DBMS).



Source: Silberschatz et al., Operating System Concepts, 9th Edition

End of Chapter 12.