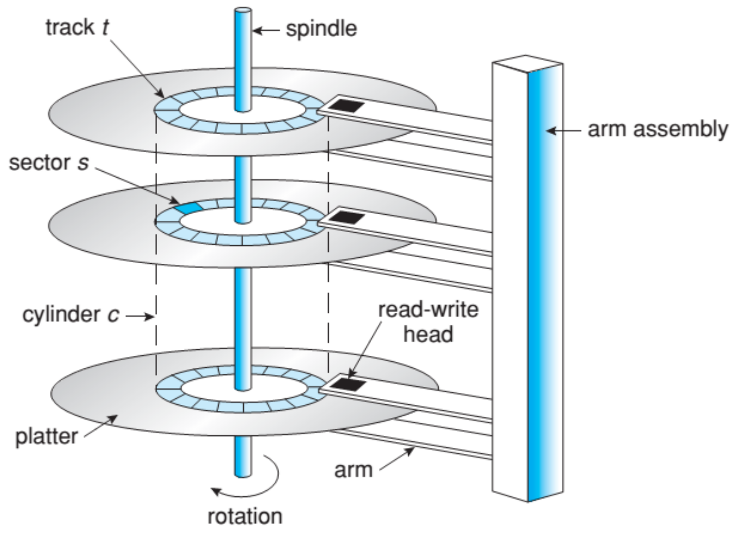
UNIT-V

MASS-STORAGE STRUCTURE

MAGNETIC DISK

Magnetic disks provide the bulk of secondary storage for modern computer systems.

- 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.
- We store information by recording it magnetically on the platters.
- A read–write head "flies" just above each surface of every platter.
- 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**.
- The tracks are subdivided into **Sectors**. Each track may contain hundreds of sectors.
- The set of tracks that are at one arm position makes up a **Cylinder**.
- There may be thousands of concentric cylinders in a disk drive.
- The storage capacity of common disk drives is measured in Giga-bytes.
- When the disk is in use, a drive motor spins it at high speed. Common drives spin at 5,400, 7,200, 10,000 and 15,000 RPM.



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:

- Seek time: It is the time necessary to move the disk arm to the desired cylinder.
- Rotational Latency: It is the time necessary for the desired sector to rotate to the disk head.

Typical disks can transfer several megabytes of data per second and they have seek times and rotational latencies of several milliseconds.

Head crash

The disk head flies on an extremely thin cushion of air, there is a danger that the head will make contact with the disk surface and damage the magnetic surface is called Head Crash.

A head crash cannot be repaired. The entire disk must be replaced.

Removable Disks

- A disk can be **removable**, allowing different disks to be mounted as needed.
- Removable magnetic disks consist of one platter, held in a plastic case to prevent damage while not in the disk drive.
- Other forms of removable disks include CDs, DVDs and Blu-ray discs removable flash-memory devices known as **flash drives.**

A disk drive is attached to a computer by a set of wires called an **I/O bus**. Several kinds of buses are available, including **advanced technology attachment (ATA)**, **serial ATA (SATA)**, **eSATA**, **universal serial bus (USB)** and **fibre channel (FC)**.

SOLID-STATE DISKS (SSD)

- SSD is nonvolatile memory that is used like a hard drive.
- SSDs are more reliable because they have no moving parts.
- SSDs are faster because they have no seek time or latency.
- SSDs can be much faster than magnetic disk drives.
- By comparing Hard disk, SSDs consumes less power but SSDs are more expensive per megabyte, have less capacity and may have shorter life spans than hard disks.

MAGNETIC TAPES

- **Magnetic tape** was used as an early secondary-storage medium.
- Tapes relatively permanent and can hold large quantities of data.
- Magnetic Tape access time is slow compared with that of main memory and magnetic disk. Random access to magnetic tape is about a thousand times slower than random access to magnetic disk, so tapes are not very useful for secondary storage.
- Tapes are used mainly for backup, for storage of infrequently used information and as a medium for transferring information from one system to another system.
- A tape is kept in a spool and it is wound or rewound past a 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 capacity is up to several terabytes.

DISK STRUCTURE

Magnetic disk drives are addressed as large one-dimensional arrays of **logical blocks**.

- The Logical block is the smallest unit of transfer.
- Size of the Logical block is 512 Bytes or 1024 Bytes.
- The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outermost cylinder.
- The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder and then through the rest of the cylinders from outermost to innermost.
- By using this mapping, we can—at least in theory—convert a logical block number into an old-style disk address that consists of a cylinder number, a track number within that cylinder and a sector number within that track.

The number of sectors per track is not a constant on some drives.

- The track which is far from the center of the disk, the length of the track is more and this track can hold more sector than the track that is nearer to the center of disk,
- Tracks in the outermost zone typically hold 40 percent more sectors than do tracks in the innermost zone.
- The drive increases its rotation speed as the head moves from the outer to the inner tracks to keep the same rate of data moving under the head.
- This method is used in CD-ROM and DVD-ROM drives.

DISK SCHEDULING ALGORITHMS

One of the responsibilities of the operating system is to use the hardware efficiently.

The 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.

Different Disk Scheduling algorithms are:

1. FCFS Scheduling
2. SSTF Scheduling
3. SCAN Scheduling
4. C- SCAN Scheduling
5. LOOK Scheduling

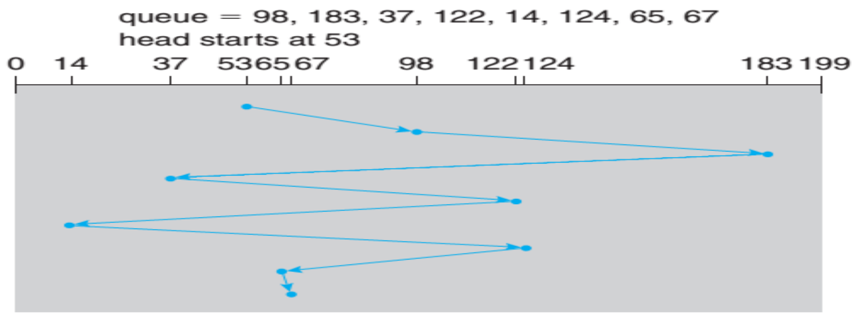
First-Come-First-Serve Algorithm

FCFS does not provide the fastest service.

Consider a disk queue with requests for I/O to blocks on cylinders in the order:

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

The disk head is initially at cylinder **53**. By using FCFS algorithm:



- It will first move from 53 to 98 the head movement of 45 cylinders
- Then 98 to 183 the head movement of 85 cylinders
- Then 183 to 37 the head movement of 146 cylinders
- Then 37 to 122 the head movement of 85 cylinders
- Then 122 to 14 the head movement of 108 cylinders
- Then 14 to 124 the head movement of 110 cylinders
- Then 124 to 65 the head movement of 59 cylinders
- Then 65 to 67 the head movement of 2 cylinders
- The total head movement of 640 cylinders.

FCFS algorithm reduces the system performance.

SSTF Scheduling

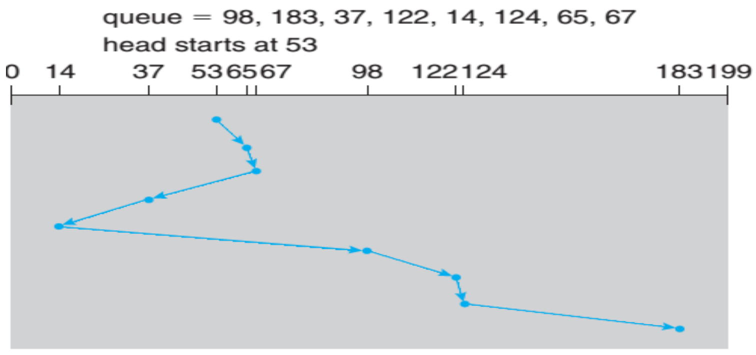
Shortest-Seek-Time-First (SSTF) algorithm service all the requests close to the current head position before moving the head far away to service other requests.

The SSTF algorithm selects the request with the least seek time from the current head position. (i.e.) SSTF chooses the pending request closest to the current head position.

Consider a disk queue with requests for I/O to blocks on cylinders in the order:

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

The disk head is initially at cylinder **53**.



Closest request to the initial head position 53 is at cylinder 65 takes 12 cylinders movements.

- Once we are at cylinder 65, the next closest request is at cylinder 67 (2 moves).
- From 67, the request at cylinder 37 is closer than the one at 98, so 37 is served next.
- Similarly we service the request at cylinder 14, then 98, 122, 124 and finally 183.
- This scheduling method results in a total head movement of only 236 cylinders.

The performance of SSTF is better than FCFS but SSTF causes starvation of some requests.

- Suppose that we have two requests in the queue, for cylinders 14 and 186 and while the request from 14 is being serviced, a new request 30 near 14 arrives.
- This new request 30 will be serviced next, making the request at 186 wait.
- While request 30 is being serviced, another request close to 30 could arrive.
- A continual stream of requests near one another could cause the request for cylinder 186 to wait indefinitely.

SCAN algorithm

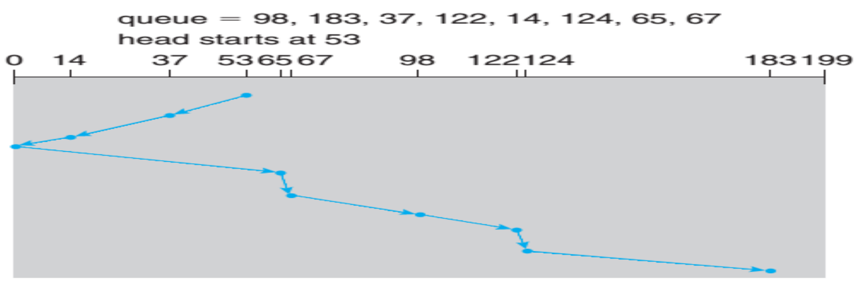
In the **SCAN algorithm**, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed and servicing continues.

The head continuously scans back and forth across the disk. The SCAN algorithm is also called as the **Elevator algorithm**, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

Consider a disk queue with requests for I/O to blocks on cylinders in the order:

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

The disk head is initially at cylinder **53**.



Before applying SCAN algorithm we need to know the the direction of head movement in addition to the head's current position.

- Assuming that the disk arm is moving toward 0 and that the initial head position is again 53 the head will next service 37 and then 14.
- At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65, 67, 98, 122, 124 and 183.
- A request arrives in the queue in front of the head, it will be serviced almost immediately.
- A request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction and comes back.

C-SCAN Scheduling

Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time.

Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.

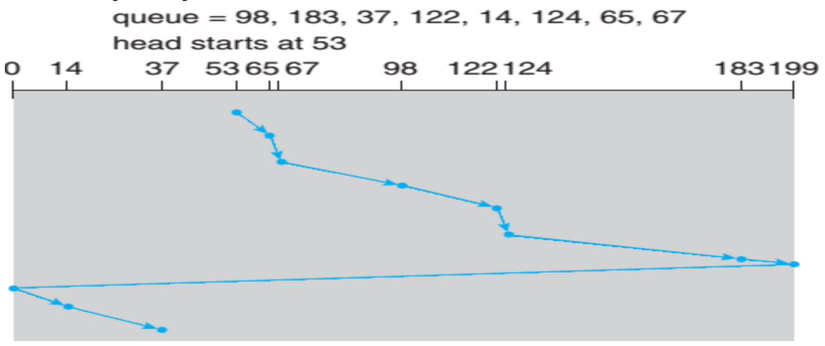
When the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip.

The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

Consider a disk queue with requests for I/O to blocks on cylinders in the order:

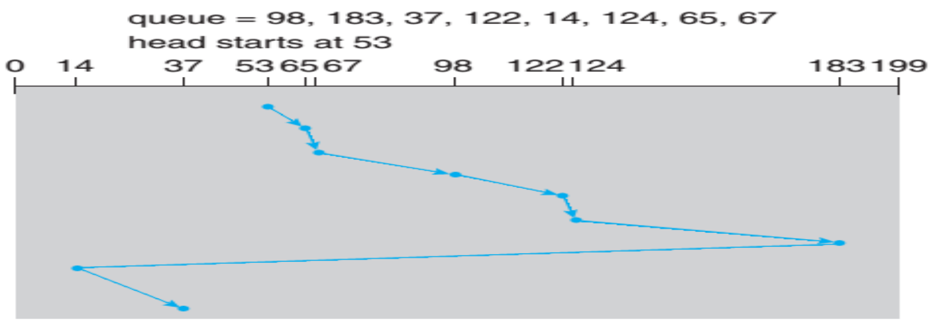
98, 183, 37, 122, 14, 124, 65, 67

The disk head is initially at cylinder **53**.



LOOK Scheduling

- Both SCAN and C-SCAN move the disk arm across the full width of the disk, neither of these algorithms are implemented.
- In LOOK and C-LOOK scheduling, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk. (i.e.) they ***look*** for a request before continuing to move in a given direction.



DISK MANAGEMENT

The operating system is responsible for several other aspects of disk management such as:

1. Disk Formatting
2. Boot Block
3. Bad Blocks

Disk Formatting

A new magnetic disk is an empty disk. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called **Low-Level Formatting** or **Physical Formatting**.

- Low-level formatting fills the disk with a special data structure for each sector.
- The data structure for a sector consists of a header, a data area and a trailer. The data are is of 512 bytes.
- The header and trailer contain information used by the disk controller, such as a sector number and an **error-correcting code (ECC)**.
- When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area.
- When the sector is read, the ECC is recalculated and compared with the stored value.
- If the stored and calculated numbers are different, this mismatch indicates that the data area of the sector has become corrupted and that the disk sector may be bad.
- The ECC is an error-correcting code because it contains the information of how many bits of data have been corrupted and identifies the corrupted bits and corrects the bits.

Before operating system can use a disk to hold files, the operating system still needs to record its own data structures on the disk. This will be done in two steps:

1. A disk is to be **partition** into one or more groups of cylinders. The operating system can treat each partition as a separate disk. **Example:** One partition can hold a copy of the operating system's executable code, while another partition holds user files.
2. **Logical Formatting:** It means creation of a file system. the operating system stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space and an initial empty directory.

Raw Disk is a partition that does not contain any file system. An I/O operation done on this raw disk is termed as Raw I/O.

Note: To increase efficiency, most file systems group blocks together into larger chunks, frequently called **clusters**. Disk I/O is done via blocks, but file system I/O is done via clusters.

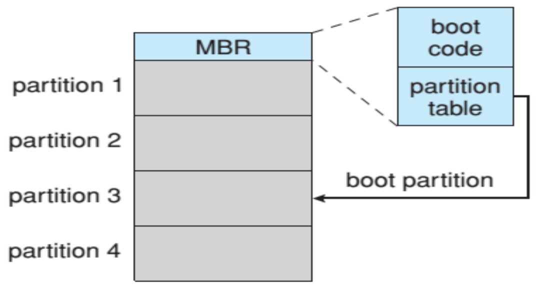
Boot Block

- When a computer is powered up or rebooted an initial program called bootstrap program will run. The bootstrap program initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory and then starts the operating system.
- To start the operating system, the bootstrap program finds the operating-system kernel on disk, loads that kernel into memory and jumps to an initial address to begin the operating-system execution.
- The bootstrap is stored in **read-only memory (ROM)**, because ROM needs no initialization and it is at a fixed location that the processor can start executing when powered up or reset. Since ROM is read only, it cannot be infected by a computer virus.

Problem: Changing this bootstrap code requires changing the ROM hardware chips.
Solution: Most systems store a tiny bootstrap loader program in the boot ROM whose only job is to bring in a full bootstrap program from disk.

- The full bootstrap program can be changed easily: a new version is simply written onto the disk.
- The full bootstrap program is stored in the "boot blocks" at a fixed location on the disk. A disk that has a boot partition is called a **boot disk** or **system disk**.
- The code in the boot ROM instructs the disk controller to read the boot blocks into memory (no device drivers are loaded at this point) and then starts executing that code.

Booting in Windows Operating system
Windows allows a hard disk to be divided into partitions and one partition identified as the **boot partition.**



- **The boot partition** contains the operating system and device drivers.
- The Windows system places its boot code in the first sector on the hard disk called the **Master Boot Record** (**MBR)**.
- Booting begins by running code that is resident in the system's ROM memory. This code directs the system to read the boot code from the MBR.
- The MBR also contains a table listing the partitions for the hard disk and a flag indicating which partition the system is to be booted from.
- Once the system identifies the boot partition, it reads the boot sector from that partition and continues with the remainder of the boot process, which includes loading the various subsystems and system services.

Bad Blocks

The disks are prone to failure. If the disk is completely failed then the disk is to be replaced and contents are restored from backup media to the new disk.
If the failure is partial (i.e.) one or more sectors become defective, These sectors are called Bad Blocks.
Depending on the disk and controller in use, these blocks are handled in a variety of ways:

- While the disk is being formatted, the disk can be scanned to find the bad blocks.
- Any bad blocks that are discovered are flagged as unusable so that the file system does not allocate them.
- If blocks go bad during normal operation, a special program must be run manually to search for the bad blocks and to lock them away.
- Data that resided on the bad blocks usually are lost.

Bad Block Recovery

- The controller maintains a list of bad blocks on the disk.
- This Bad Block list is initialized during the low-level formatting at the factory and this list is updated over the life of the disk.
- Low-level formatting also sets aside spare sectors not visible to the operating system.
- The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as **Sector Sparing** or **Forwarding**.
- Most disks are formatted to provide a few spare sectors in each cylinder and a spare cylinder.
- When a bad block is remapped, the controller uses a spare sector from the same cylinder.

SWAP-SPACE MANAGEMENT

Swap-Space Management is another low-level task of the operating system.

- Virtual memory uses disk space as an extension of main memory by using swap space.
- Since disk swap space access is much slower than main memory access the system performance decreases.
- The main goal for the design and implementation of swap space is to provide the best throughput for the virtual memory system.

Swap-Space Use

- Swap space is used in various ways by different operating systems, depending on the memory-management algorithms in use.
- Systems that implement swapping may use swap space to hold an entire process image, including the code and data segments.
- Systems that support Paging may simply store pages that have been pushed out of main memory.
- The amount of swap space needed on a system can therefore vary from a few megabytes of disk space to gigabytes, depending on the amount of physical memory, the amount of virtual memory it is using for back-up and the way in which the virtual memory is used.

Example: Older Linux systems has suggested setting swap space to double the amount of physical memory but the modern Linux systems use considerably less swap space.

Note: Some operating systems including Linux allow the use of multiple swap spaces, including both files and dedicated swap partitions.

Swap-Space Location

A swap space can reside in one of two places:

1. It can be carved out of the normal file system.
2. It can be in a separate disk partition.

- Swap-space can be created in a separate **Raw Partition**.
- No file system or directory structure is placed in this space instead a separate swap-space storage manager is used to allocate and deallocate the blocks from the raw partition.
- Swap space manager uses algorithms optimized for speed rather than for storage efficiency, because swap space is accessed much more frequently than file systems when it is used.
- The raw-partition approach creates a fixed amount of swap space during disk partitioning.
- Adding more swap space requires either repartitioning the disk or adding another swap space elsewhere.

REDUNDANT ARRAYS OF INDEPENDENT DISKS (RAID) STRUCTURE

- Disk drives have continued to get smaller and cheaper, so it is now economically feasible to attach many disks to a computer system.
- Having a large number of disks in a system and if they are operated in parallel we can improve the rate at which data can be read or written.
- This setup offers the potential for improving the reliability of data storage, because redundant information can be stored on multiple disks. Thus, failure of one disk does not lead to loss of data.
- This disk-organization techniques is called as **Redundant Arrays Of Independent Disks (RAID)**.
- RAIDs are used for Higher reliability and Higher data-transfer rates.

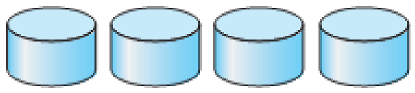
RAID Levels

RAIDs can be implemented in different levels:

1. RAID 0: Non-Redundant Striping
2. RAID 1: Mirrored Disks.
3. RAID 2: Memory-Style Error-Correcting Codes.
4. RAID 3: Bit-Interleaved Parity.
5. RAID 4: Block-Interleaved Parity.
6. RAID 5: Block-Interleaved Distributed Parity
7. RAID 6: P+Q redundancy
8. RAID 0 + 1 and 1 + 0

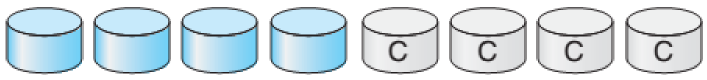
RAID 0: Non-Redundant Striping

RAID level 0 refers to disk arrays with striping at the level of blocks but without any redundancy. (i.e.) some part of the data is stored in one disk other part of the data is stored in other disks without duplicating the data.


(a) RAID 0: non-redundant striping.

RAID level 1: Disk Mirroring

The entire data in the disk is copied in to other disks. (i.e.) the data that is stored in one disk the same data will be copied in other disk. If one disk fails we can recover the data from its copied disk called as Backup disk.



(b) RAID 1: mirrored disks.

Note: Mirroring provides high reliability, but it is expensive. Striping provides high data-transfer rates, but it does not improve reliability.

RAID level 2. Memory-Style Error-Correctingcode (ECC) Organization

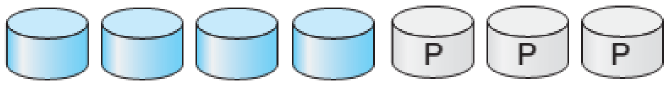
- It uses parity bits to detect errors. Each byte in a memory system may have a parity bit associated with it that records whether the number of bits in the byte set to 1 is even (parity = 0) or odd (parity = 1).
- If one of the bits in the byte is damaged (i.e.) either a 1 becomes a 0 or a 0 becomes a 1, the parity of the byte changes and thus does not match the stored parity.
- Similarly, if the stored parity bit is damaged, it does not match the computed parity.
- Thus, all single-bit errors are detected by the memory system.

ECC can be used directly in disk arrays via striping of bytes across disks.

Example: 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.

If one of the disks fails, the remaining bits of the byte and the associated error-correction bits can be read from other disks and used to reconstruct the damaged data.

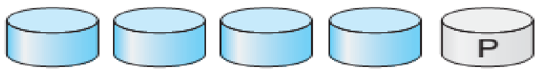
RAID level 2 requires only three disks whereas RAID1 requires four disks.



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

RAID level 3: Bit-Interleaved Parity

- Here, the disk controllers can detect whether a sector has been read correctly, so a single parity bit can be used for error correction as well as for detection.
- If one of the sectors is damaged, we know exactly which sector it is and we can figure out whether any bit in the sector is a 1 or a 0 by computing the parity of the corresponding bits from sectors in the other disks.
- If the parity of the remaining bits is equal to the stored parity, the missing bit is 0; otherwise, it is 1.
- RAID level 3 is less expensive than RAID2, it requires only one extra disk.

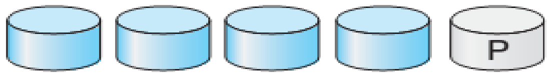


(d) RAID 3: bit-interleaved parity.

RAID level 4: Block-Interleaved Parity Organization

It Uses block-level striping and keeps a parity block on a separate disk for corresponding blocks from *N* other disks.

If one of the disks fails, the parity block can be used with the corresponding blocks from the other disks to restore the blocks of the failed disk.



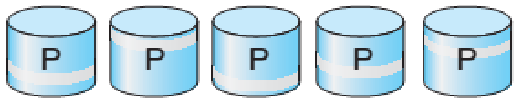
(e) RAID 4: block-interleaved parity.

RAID level 5: Block-Interleaved Distributed Parity

It differs from level 4 in that it spreads data and parity among all *N*+ 1 disks, rather than storing data in *N* disks and parity in one disk. For each block, one of the disks stores the parity and the others store data.

Ex: With an array of five disks, the parity for the *n*th block is stored in disk **(*n* mod 5)+1**.

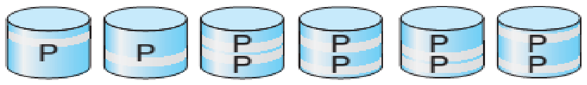
- The *n*th blocks of the other four disks store actual data for that block.
- A parity block cannot store parity for blocks in the same disk, because a disk failure would result in loss of data as well as of parity and the loss would not be recoverable.
- By spreading the parity across all the disks in the set, RAID 5 avoids potential overuse of a single parity disk, which can occur with RAID 4.
- RAID 5 is the most common parity RAID system.



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

RAID level 6: P + Q redundancy scheme

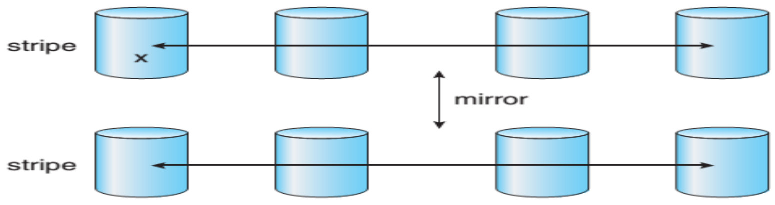
- It is like RAID level 5 but stores extra redundant information to guard against multiple disk failures.
- Instead of parity, error-correcting codes such as the **Reed–Solomon codes** are used.
- 2 bits of redundant data are stored for every 4 bits of data compared with 1 parity bit in level 5 and the system can tolerate two disk failures.



(g) RAID 6: P + Q redundancy.

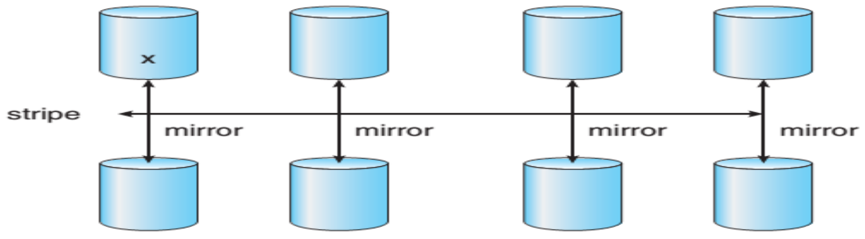
RAID levels 0 + 1 and 1 + 0

- RAID level 0 + 1 refers to a combination of RAID levels 0 and 1.
- RAID 0 provides the performance, while RAID 1 provides the reliability.
- In RAID 0 + 1, a set of disks are striped and then the stripe is mirrored to another, equivalent stripe.



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

- RAID level 1 + 0, in which disks are mirrored in pairs and then the resulting mirrored pairs are striped.
- If a single disk fails in RAID 0 + 1, an entire stripe is inaccessible, leaving only the other stripe.
- With a failure in RAID 1 + 0, a single disk is unavailable, but the disk that mirrors it is still available, as are all the rest of the disk.



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

DISK ATTACHMENT

Computers access disk storage in two ways.

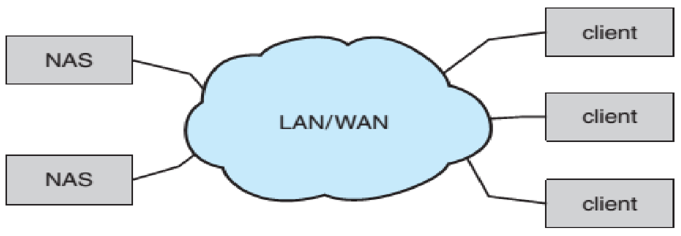
1. Host-Attached Storage (HAS)
2. Network-Attached Storage (NAS)

Host-Attached Storage

- Host-attached storage is storage accessed through local I/O ports.
- The typical desktop PC uses an I/O bus architecture called IDE or ATA or SATA.
- This architecture supports a maximum of two drives per I/O bus.
- Hard disk drives, RAID arrays and CD, DVD and tape drives are storage devices that are suitable for use as Host-Attached Storage.
- The I/O commands that initiate data transfers to a host-attached storage device are reads and writes of logical data blocks directed to specifically identified storage units.

Network-Attached Storage

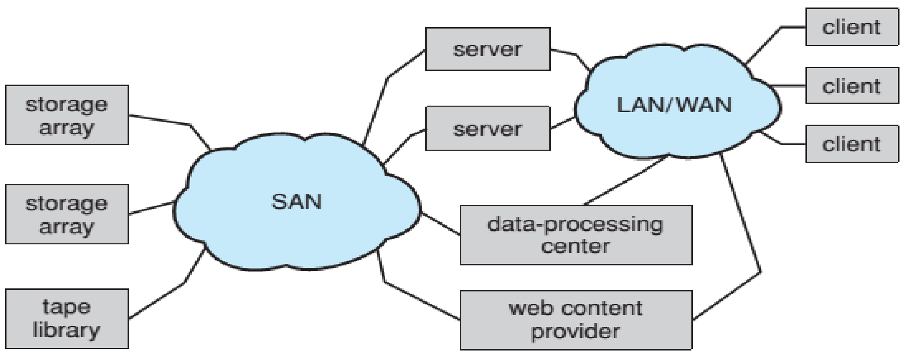
- A network-attached storage (NAS) device is a special-purpose storage system that is accessed remotely over a data network.



- Clients access network-attached storage via a remote-procedure-call interface such as NFS for UNIX systems or CIFS for Windows machines.
- The remote procedure calls (RPCs) are carried via TCP or UDP over an IP network usually the same local-area network (LAN) that carries all data traffic to the clients.
- Network-attached storage provides a convenient way for all the computers on a LAN to share a pool of storage with the same ease of naming and access enjoyed with local host-attached storage.

Storage-Area Network

A storage-area network (SAN) is a private network connecting servers and storage units.



- Multiple hosts and multiple storage arrays can attach to the same SAN and storage can be dynamically allocated to hosts.
- A SAN switch allows or prohibits access between the hosts and the storage.
- Example: If a host is running low on disk space, the SAN can be configured to allocate more storage to that host.
- SANs make it possible for clusters of servers to share the same storage and for storage arrays to include multiple direct host connections.