12 Mass-Storage Systems

Chapter 11 in the book

- Overview of Mass Storage Structure
- HDD Scheduling
- NVM Scheduling
- Error Detection and Correction
- Storage Device Management
- Swap-Space Management
- Storage Attachment
- RAID Structure

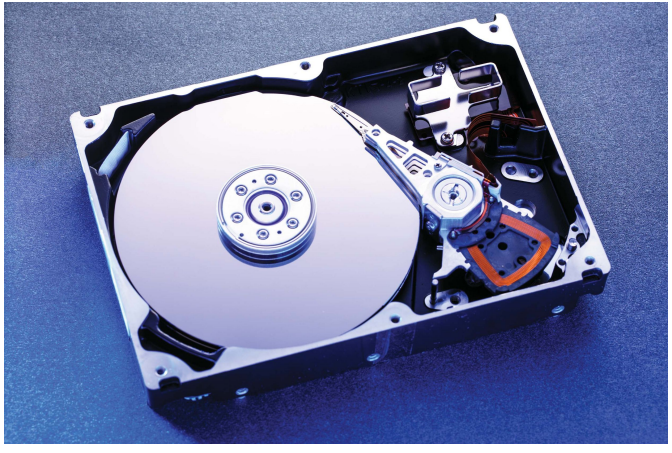
The First Commercial Disk Drive

1956
IBM RAMDAC computer included the IBM Model 350 disk storage system

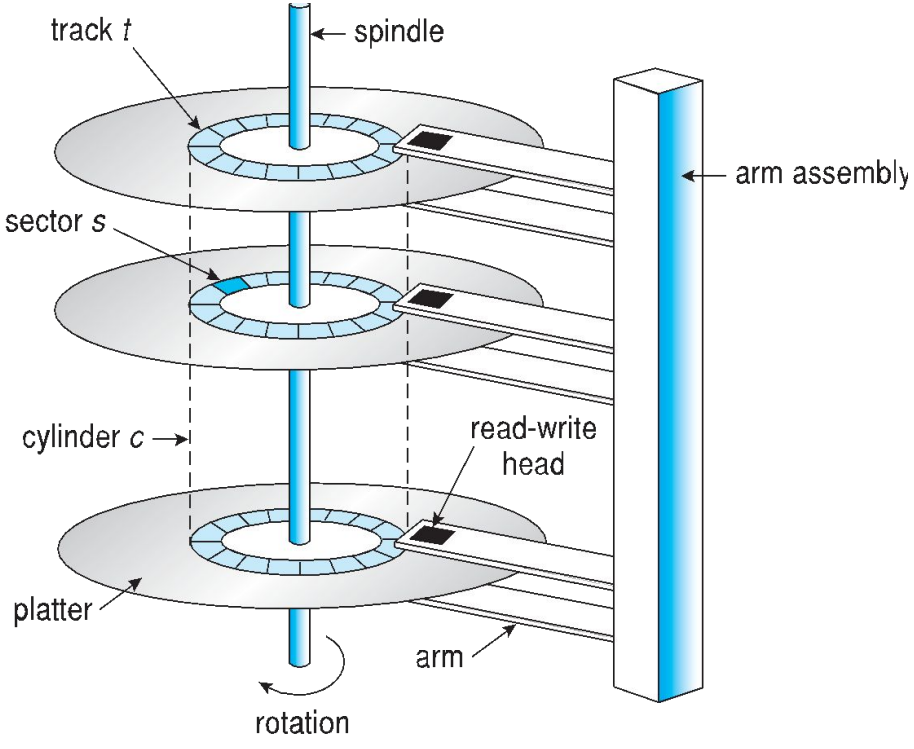
5M (7 bit) characters
50 x 24” platters
Access time = < 1 second

Overview of Mass Storage Structure

- Bulk of secondary storage for modern computers is **hard disk drives** (**HDDs**) and **nonvolatile memory** (**NVM**) devices
- **HDDs** spin platters of magnetically-coated material under moving read-write heads
  - Drives rotate at 60 to 250 times per second
  - **Transfer rate** is rate at which data flow between drive and computer
  - **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
  - **Head crash** results from disk head making contact with the disk surface -- That's bad
- Disks can be removable



Cylinders, tracks, & sectors



Disk positioning system

Move head to specific track and keep it there

- Resist physical shocks, imperfect tracks, etc.

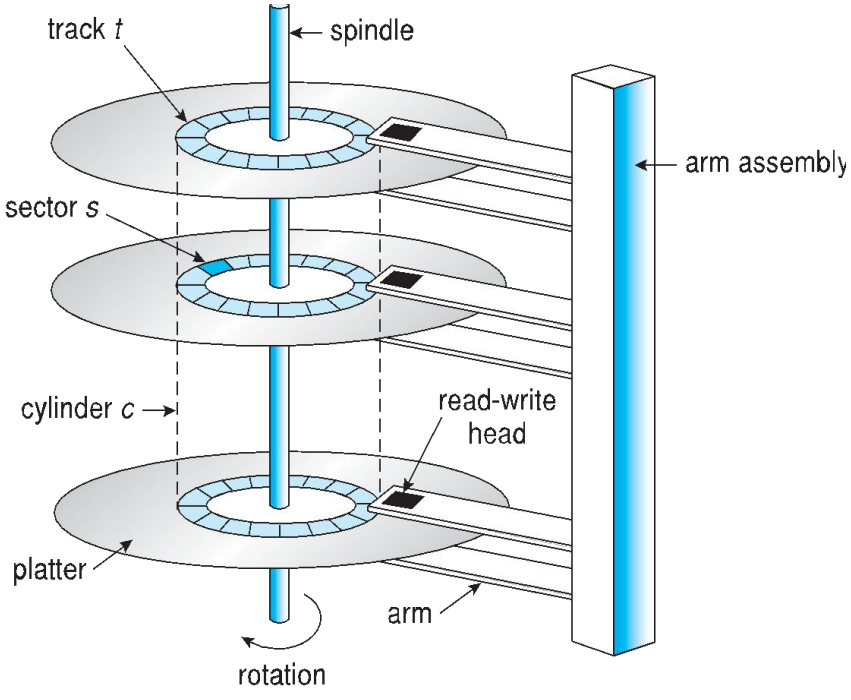
A seek consists of up to four phases:

- - speedup–accelerate arm to max speed or half way point
- - coast–at max speed (for long seeks)
- - slowdown–stops arm near destination
- - settle–adjusts head to actual desired track

Very short seeks dominated by settle time (~1 ms)

Short (200-400 cyl.) seeks dominated by speedup

- - Accelerations of 40g



https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

Disk positioning system

Head switches comparable to short seeks

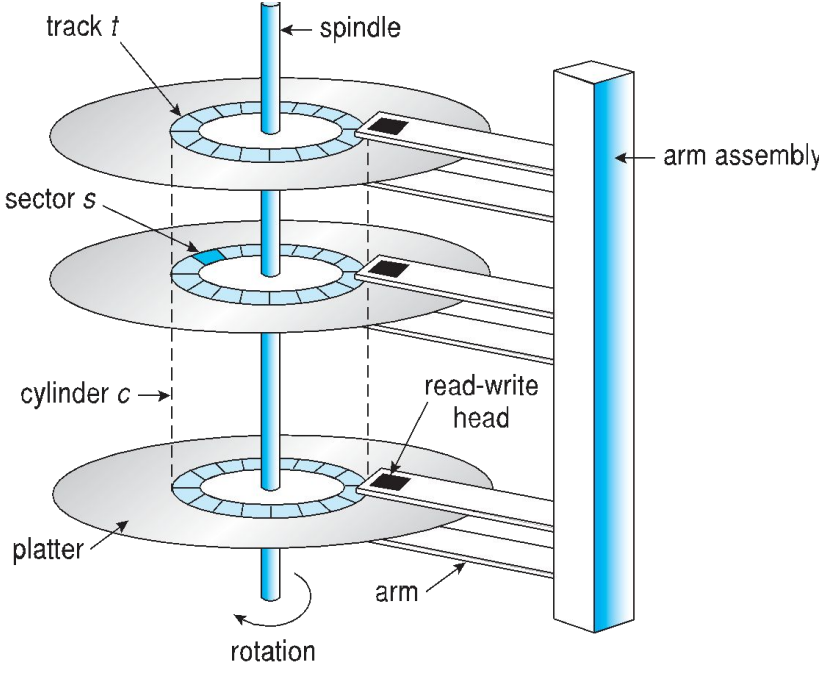
- May also require head adjustment
- Settles take longer for writes than for reads – Why?
  - If read strays from track, catch error with checksum, retry
  - If write strays, you've just clobbered some other track

Disk keeps table of pivot motor power

- Maps seek distance to power and time
- Disk interpolates over entries in table
- Table set by periodic "thermal re-calibration"
- But, e.g., ~500 ms recalibration every ~25 min bad for AV

"Average seek time" quoted can be many things

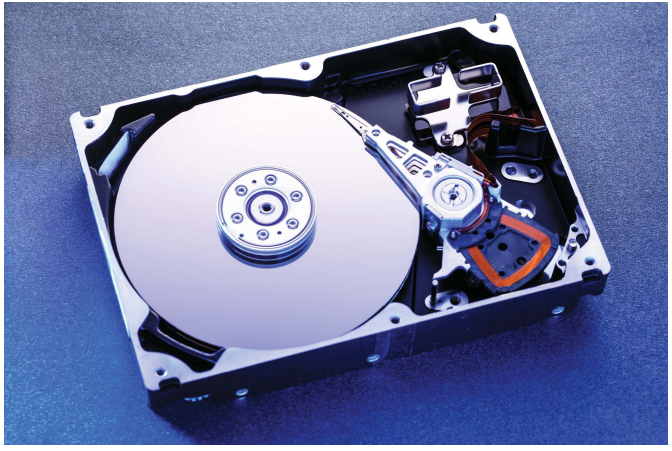
- Time to seek 1/3 disk, 1/3 time to seek whole disk



https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

Hard Disk Drives

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



Sectors

Disk interface presents linear array of sectors

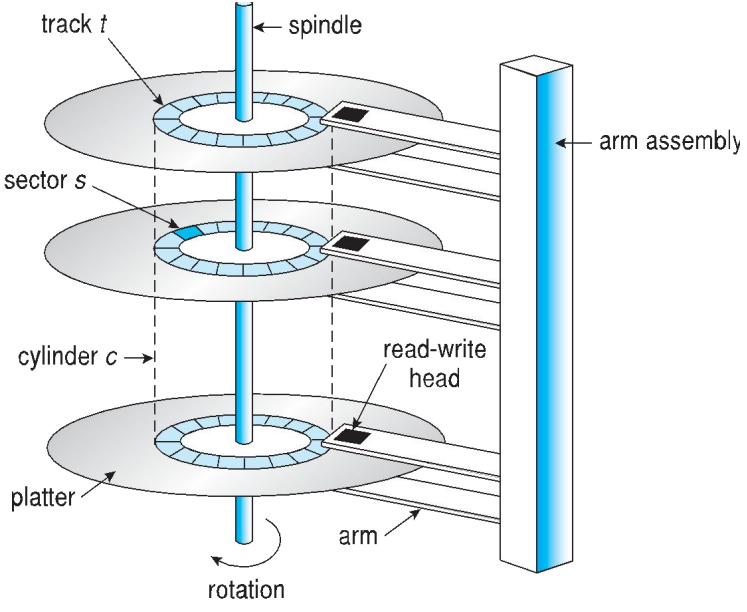
- Historically 512 B, but 4 KiB in “advanced format” disks
- Written atomically (even if there is a power failure)

Disk maps logical sector #s to physical sectors

- Zoning–puts more sectors on longer tracks
- Track skewing–sector 0 pos. varies by track
  - because of sequential access speed
- Sparing–flawed sectors remapped elsewhere

OS doesn’t know logical to physical sector mapping

- Larger logical sector # difference means longer seek time
- Highly non-linear relationship (and depends on zone)
- OS has no info on rotational positions
- Can empirically build table to estimate times



https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

Disk interface

Controls hardware, mediates access

Computer, disk often connected by bus (e.g., ATA, SCSI, SATA)

- Multiple devices may contents for bus

Possible disk/interface features:

- Disconnect from bus during requests
- Command queuing: Give disk multiple requests
  - Disk can schedule them using rotational information
- Disk cache used for read-ahead
  - Otherwise, sequential reads would incur whole revolution
  - Cross track boundaries? Can't stop a head-switch
- Some disks support write caching
  - But data not stable—not suitable for all requests

https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

Hard Disk Performance

- **Access Latency** = **Average access time** = average seek time + average latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead
- For example to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead =
  - 5ms + 4.17ms + 0.1ms + transfer time =
  - Transfer time = 4KB / 1Gb/s * 8Gb / GB * 1GB / 1024^2KB = 32 / (1024^2) = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

Disk performance

Placement & ordering of requests a huge issue

- Sequential I/O much, much faster than random
- Long seeks much slower than short ones
- Power might fail any time, leaving inconsistent state

Must be careful about order for crashes

- More on this in next two lectures

Try to achieve contiguous accesses where possible

- E.g., make big chunks of individual files contiguous

Try to order requests to minimize seek times

- OS can only do this if it has multiple requests to order
- Requires disk I/O concurrency
- High-performance apps try to maximize I/O concurrency

Next: How to schedule concurrent requests

https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

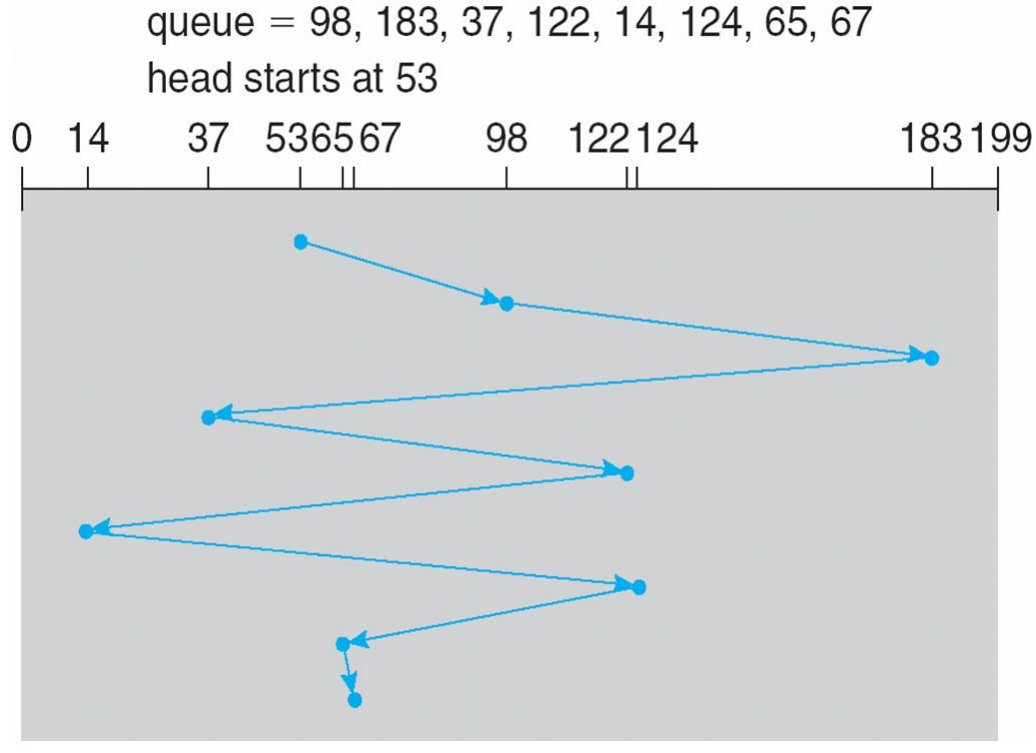
Disk Scheduling

- 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

Scheduling: FCFS

“First Come First Served”

- Process disk requests in the order they are received



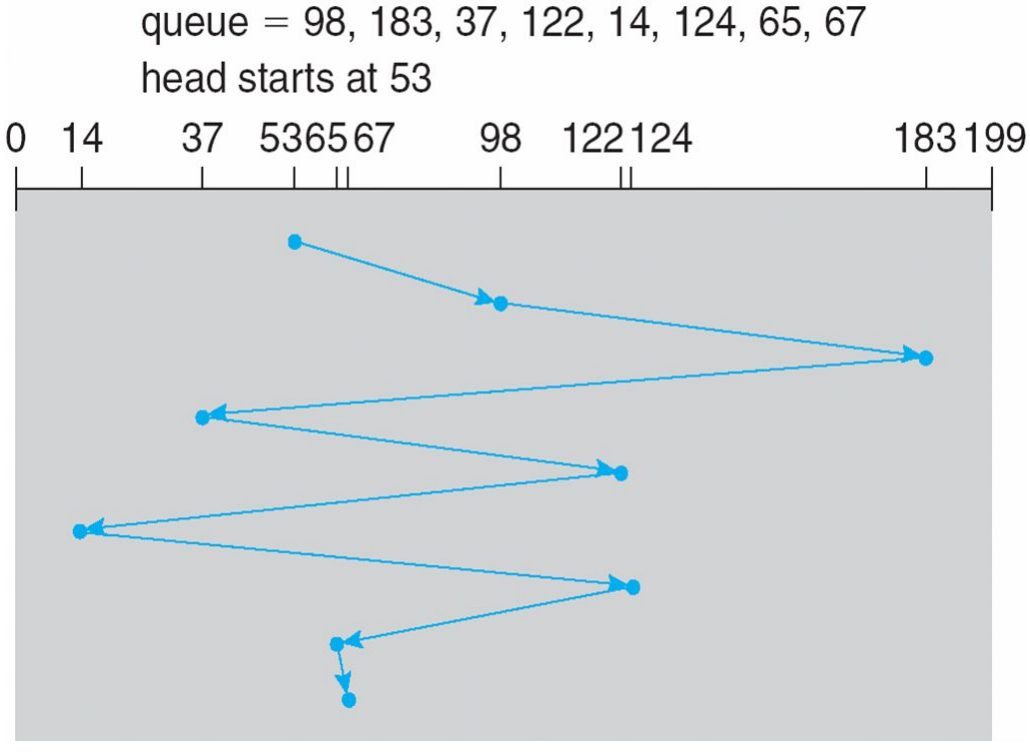
Scheduling: FCFS

Advantages

- Easy to implement
- Good fairness

Disadvantages

- Cannot exploit request locality
- Increases average latency, decreasing throughput



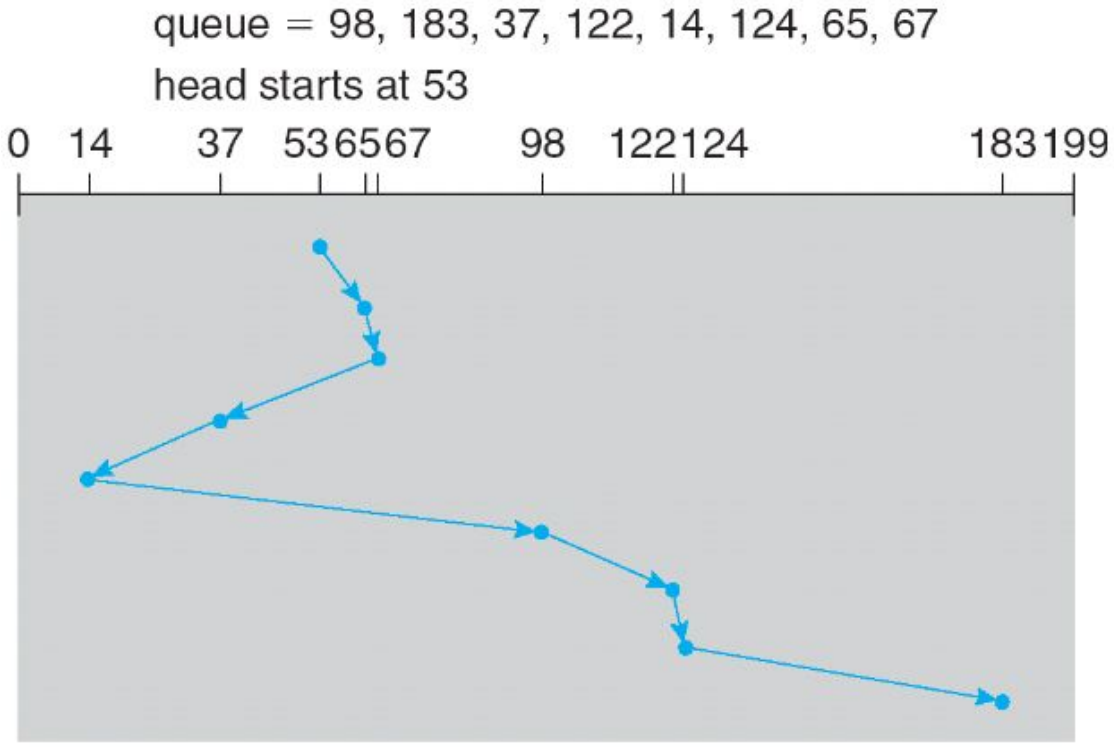
https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

Shortest positioning time first (SPTF)

Shortest positioning time first (SPTF)

- Always pick request with shortest seek time

Also called **Shortest Seek Time First (SSTF)**



https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

Shortest positioning time first (SPTF)

Advantages

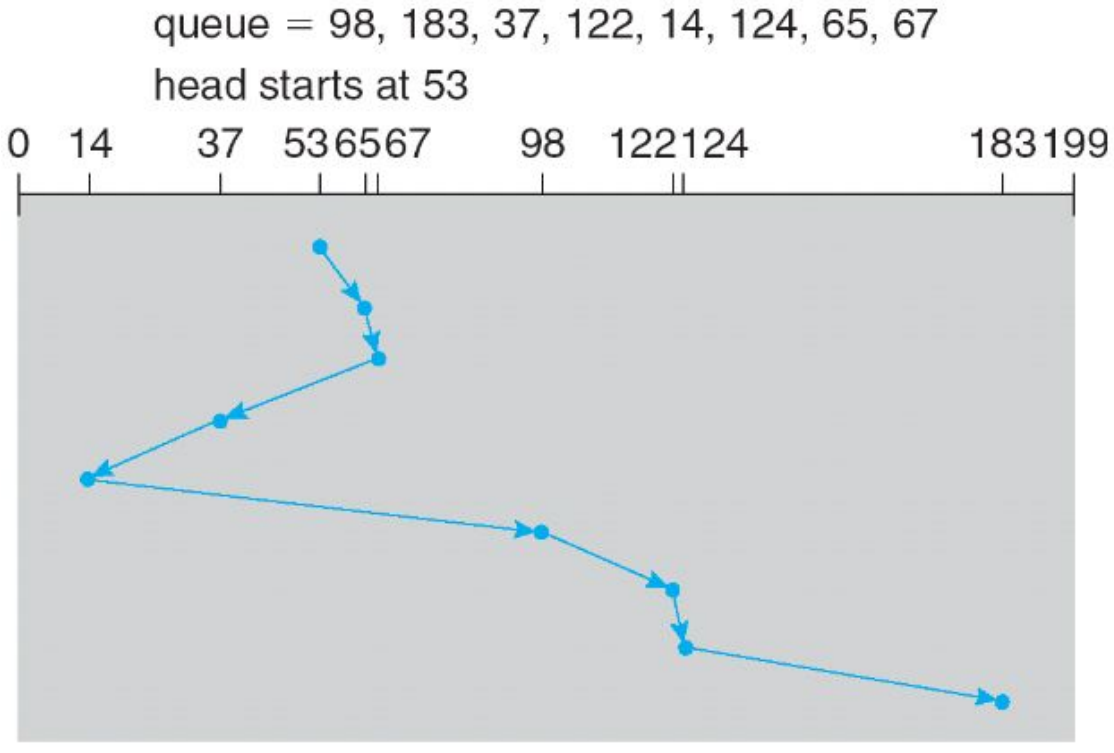
- - Exploits locality of disk requests
- - Higher throughput

Disadvantages

- **Starvation**
- **Don’t always know what request will be fastest**

Improvement: Aged SPTF

- **Give older requests higher priority**
- **Adjust “effective” seek time with weighting factor:**
- **Teff = Tpos − W · Twait**

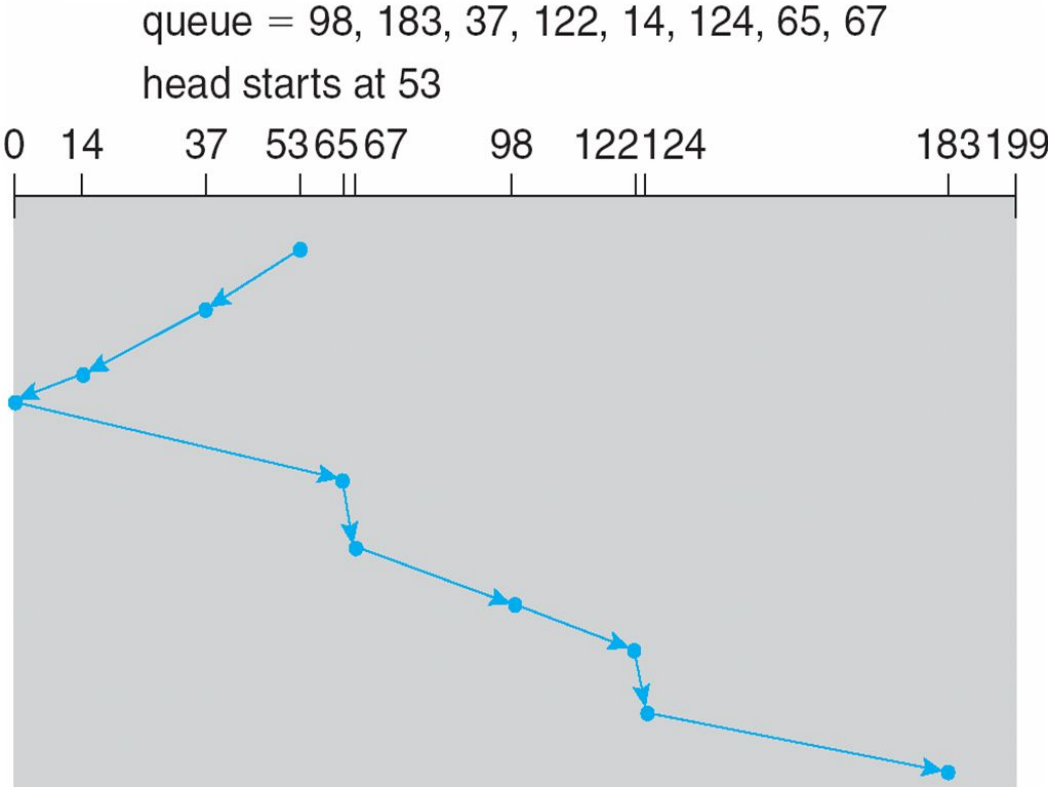


https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

“Elevator” scheduling (SCAN)

Sweep across disk, servicing all requests passed

- Like SPTF, but next seek must be in same direction
- Switch directions only if no further requests



https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

“Elevator” scheduling (SCAN)

Advantages

- Takes advantage of locality
- Bounded waiting

Disadvantages

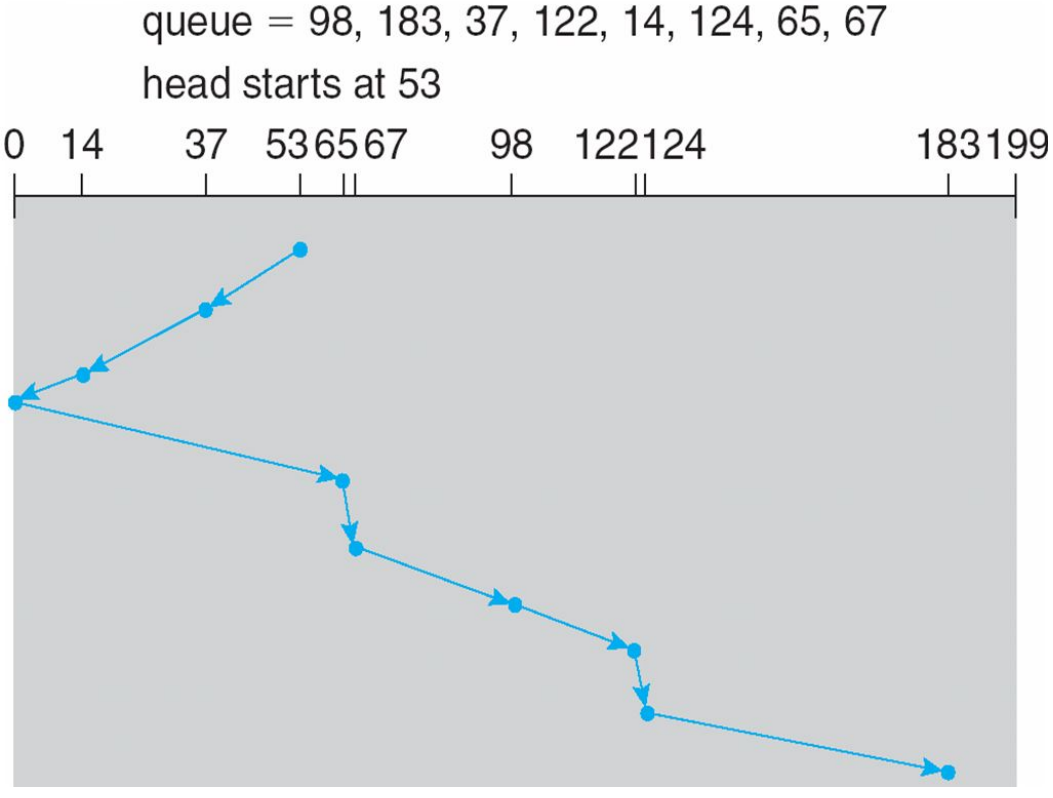
- Cylinders in the middle get better service
- Might miss locality SPTF could exploit

CSCAN: Only sweep in one direction

- Very commonly used algorithm in Unix

Also called LOOK/CLOOK in textbook

- **- (Textbook uses [C]SCAN to mean scan entire disk uselessly)**

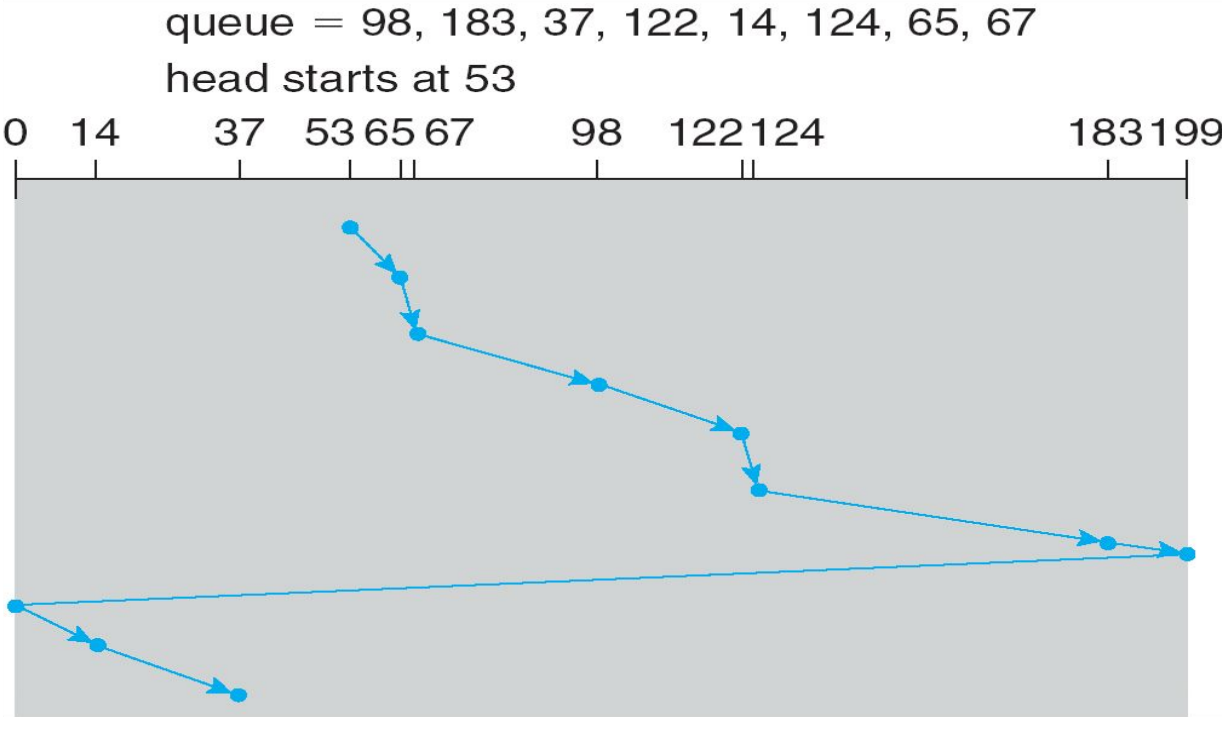


https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

C-SCAN

CSCAN: Only sweep in one direction

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



VSCAN(r)

Continuum between SPTF and SCAN

- Like SPTF, but slightly changes “effective” positioning time
- **If request in same direction** as previous seek:
  - **Teff = Tpos**
- **Otherwise**:
  - **Teff = Tpos + r · Tmax**

when r = 0, get SPTF, when r = 1, get SCAN

E.g., r = 0.2 works well

• **Advantages and disadvantages**

- Those of SPTF and SCAN, depending on how r is set

• See [Worthington] for good description and evaluation of

various disk scheduling algorithms

https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

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, but still possible
- To avoid starvation Linux implements **deadline** scheduler
- In RHEL 7 also **NOOP** and **completely fair queueing** scheduler (**CFQ**) also available, defaults vary by storage device

Nonvolatile Memory Devices

- If disk-drive like, then called **solid-state disks** (**SSDs**)
- Other forms include **USB drives** (thumb drive, flash drive), DRAM disk replacements, surface-mounted on motherboards, and main storage in devices like smartphones
- Can be more reliable than HDDs
- More expensive per MB
- Maybe have shorter life span – need careful management
- Less capacity
- But much faster
- Busses can be too slow -> connect directly to PCI for example
- No moving parts, so no seek time or rotational latency

Nonvolatile Memory Devices

- Have characteristics that present challenges
- Read and written in “page” increments (think sector) but can’t overwrite in place
  - Must first be erased, and erases happen in larger ”block” increments
  - Can only be erased a limited number of times before worn out – ~ 100,000
  - Life span measured in **drive writes per day** (**DWPD**)
    - A 1TB NAND drive with rating of 5DWPD is expected to have 5TB per day written within warrantee period without failing

Flash memory

Today, people increasingly using flash memory

Completely solid state (no moving parts)

- Remembers data by storing charge
- Lower power consumption and heat
- No mechanical seek times to worry about

Limited # overwrites possible

- Blocks wear out after 10,000 (MLC) – 100,000 (SLC) erases
- Requires flash translation layer (FTL) to provide wear leveling,
  - so repeated writes to logical block don't wear out physical block
- FTL can seriously impact performance
- In particular, random writes very expensive [Birrell]

Limited durability

- Charge wears out over time
- Turn off device for a year, you can potentially lose data

https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

Types of flash memory

Types of flash memory

NAND flash (most prevalent for storage)

- Higher density (most used for storage)
- Faster erase and write
- More errors internally, so need error correction

NOR flash

- Faster reads in smaller data units
- Can execute code straight out of NOR flash
- Significantly slower erases

Single-level cell (SLC) vs. Multi-level cell (MLC)

- MLC encodes multiple (two) bits in voltage level
- MLC slower to write than SLC
- MLC has lower durability (bits decay faster)

Nowadays, most flash drives are TLC (or even QLC)

https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

NAND flash overview

Flash device has 2112-byte pages

- 2048 bytes of data + 64 bytes metadata & ECC

Blocks contain 64 (SLC) or 128 (MLC) pages

Blocks segregated into 2–4 planes

- All planes contend for same package pins
- But can access their blocks in parallel to overlap latencies

Can read one page at a time

- - Takes 25 µsec + time to get data off chip

Must erase whole block before programing

- Erase sets all bits to 1—very expensive (2 msec)
- Programming pre-erased block requires moving data to internal buffer, then 200 (SLC)–800 (MLC) µsec

https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

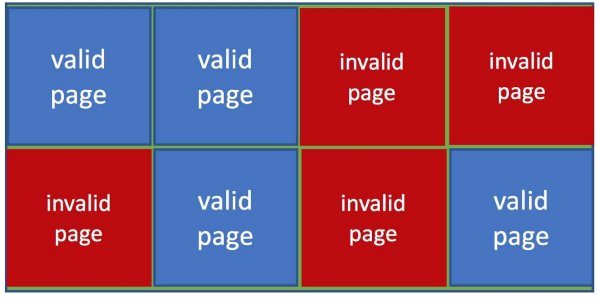
Flash characteristics

Parameter		SLC	MLC
	Density Per Die (GB)	4	8
	Page Size (Bytes)	2048+32	2048+64
	Block Size (Pages)	64	128
	Read Latency (μs)	25	25
	Write Latency (μs)	200	800
	Erase Latency (μs)	2000	2000
40MHz, 16-bit bus	Read b/w (MB/s)	75.8	75.8
	Program b/w (MB/s)	20.1	5.0
133MHz	Read b/w (MB/s)	126.4	126.4
	Program b/w (MB/s)	20.1	5.0

http://cseweb.ucsd.edu/~swanson/papers/Asplos2009Gordon.pdf
https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

NAND Flash Controller Algorithms

- With no overwrite, pages end up with mix of valid and invalid data
- To track which logical blocks are valid, controller maintains **flash translation layer** (**FTL**) table
- Also implements **garbage collection** to free invalid page space
- Allocates **overprovisioning** to provide working space for GC
- Each cell has lifespan, so **wear leveling** needed to write equally to all cells



NAND block with valid and invalid pages

FTL straw man: in-memory map

Keep in-memory map of logical → physical page #

- On write, pick unused page, mark previous physical page free
- Repeated writes of a logical page will hit different physical pages

Store map in device memory, but must rebuild on power-up

idea: Put header on each page, scan all headers on power-up:

⟨logical page #, **A**llocated bit, **W**ritten bit, **O**bsolete bit⟩

- - A-W-O = 1-1-1: free page
- - A-W-O = 0-1-1: about to write page
- - A-W-O = 0-0-1: successfully written page
- - A-W-O = 0-0-0: obsolete page (can erase block without copying)

Why the 0-1-1 state?

- Why the 0-1-1 state? After power failure partly written (not free)

What's wrong still?

- FTL requires alot of RAM on device

https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

More realistic FTL

Store the FTL map in the flash device itself

- Add one header bit to distinguish map page from data page
- Logical read may miss map cache, require 2 flash reads
- Keep smaller “map-map” in memory, cache some map pages

Must garbage-collect blocks with obsolete pages

- Copy live pages to a new block, erase old block
- Always need free blocks, can’t use 100% physical storage

Problem: write amplification

- Small random writes punch holes in many blocks
- If small writes require garbage-collecting a 90%-full blocks
  - ...means you are writing 10× more physical than logical data!

Must also periodically re-write even blocks w/o holes

- **Wear leveling** ensures active blocks don’t wear out first

https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

NVM Scheduling

- No disk heads or rotational latency but still room for optimization
- In RHEL 7 **NOOP** (no scheduling) is used but adjacent LBA requests are combined
  - NVM best at random I/O, HDD at sequential
  - Throughput can be similar
  - **Input/Output operations per second** (**IOPS**) much higher with NVM (hundreds of thousands vs hundreds)
  - But **write amplification** (one write, causing garbage collection and many read/writes) can decrease the performance advantage

the rest is skipped in the lecture.

Error Detection and Correction

- Fundamental aspect of many parts of computing (memory, networking, storage)
- **Error detection** determines if there a problem has occurred (for example a bit flipping)
  - If detected, can halt the operation
  - Detection frequently done via parity bit
- Parity one form of **checksum** – uses modular arithmetic to compute, store, compare values of fixed-length words
  - Another error-detection method common in networking is **cyclic redundancy check** (**CRC**) which uses hash function to detect multiple-bit errors
- **Error-correction code** (**ECC**) not only detects, but can correct some errors
  - Soft errors correctable, hard errors detected but not corrected

Storage Device Management

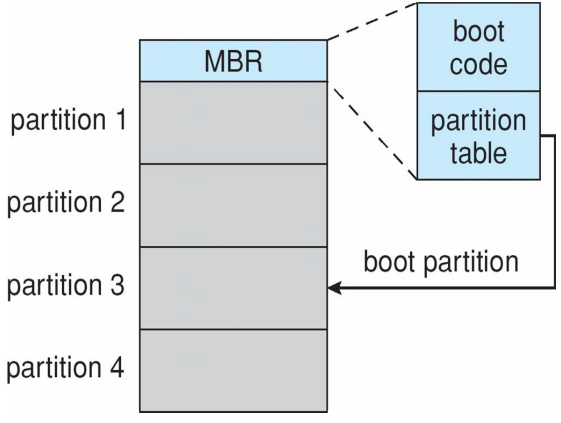
- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, plus data, plus error correction code (**ECC**)
  - Usually 512 bytes of data but can be selectable
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk
  - **Partition** the disk into one or more groups of cylinders, each treated as a logical disk
  - **Logical formatting** or “making a file system”
  - To increase efficiency most file systems group blocks into **clusters**
    - Disk I/O done in blocks
    - File I/O done in clusters

Storage Device Management (cont.)

- **Root partition** contains the OS, other partitions can hold other Oses, other file systems, or be raw
  - **Mounted** at boot time
  - Other partitions can mount automatically or manually
- At mount time, file system consistency checked
  - Is all metadata correct?
    - If not, fix it, try again
    - If yes, add to mount table, allow access
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
  - Or a boot management program for multi-os booting

Device Storage Management (Cont.)

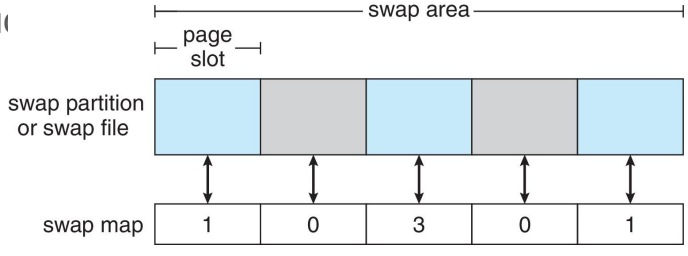
- Raw disk access for apps that want to do their own block management, keep OS out of the way (databases for example)
- Boot block initializes system
  - The bootstrap is stored in ROM, firmware
  - **Bootstrap loader** program stored in boot blocks of boot partition
- Methods such as **sector sparing** used to handle bad blocks



Booting from secondary storage in Windows

Swap-Space Management

- Used for moving entire processes (swapping), or pages (paging), from DRAM to secondary storage when DRAM not large enough for all processes
- Operating system provides **swap space management**
  - Secondary storage slower than DRAM, so important to optimize performance
  - Usually multiple swap spaces possible – decreasing I/O load on any given device
  - Best to have dedicated devices
  - Can be in raw partition or a file within a file system (for convenience of adding)
  - Data stru

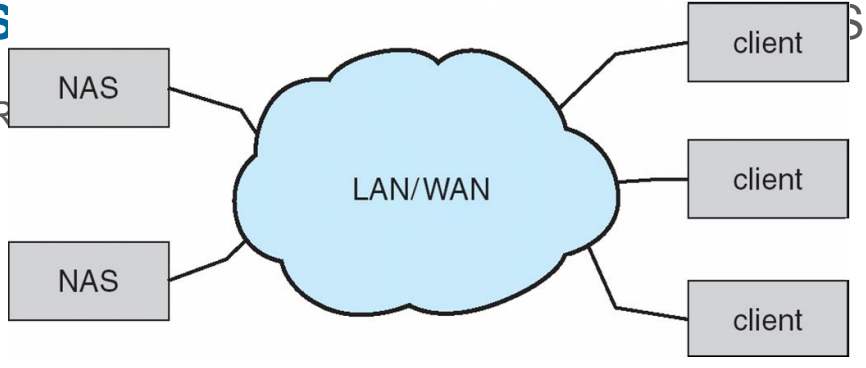


Storage Attachment

- Computers access storage in three ways
  - host-attached
  - network-attached
  - cloud
- Host attached access through local I/O ports, using one of several technologies
  - To attach many devices, use storage busses such as USB, firewire, thunderbolt
  - High-end systems use **fibre channel** (**FC**)
    - High-speed serial architecture using fibre or copper cables
    - Multiple hosts and storage devices can connect to the FC fabric

Network-Attached Storage

- Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)
  - Remotely attaching to file systems
- NFS and CIFS are common protocols
- Implemented via remote procedure calls (RPCs) between host and storage over typically TCP or UDP on IP network
- **iSCS** SCSI protocol
  - R



Cloud Storage

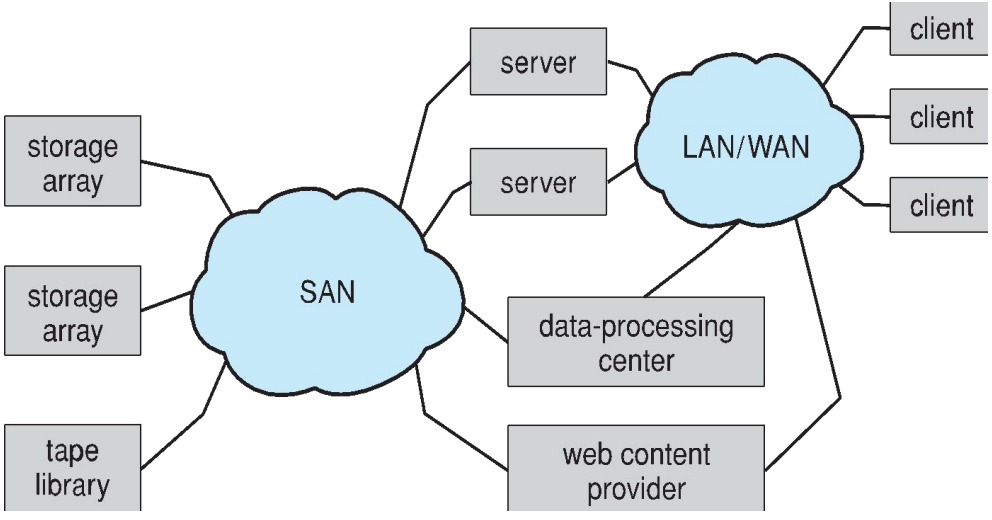
- Similar to NAS, provides access to storage across a network
  - Unlike NAS, accessed over the Internet or a WAN to remote data center
- NAS presented as just another file system, while cloud storage is API based, with programs using the APIs to provide access
  - Examples include Dropbox, Amazon S3, Microsoft OneDrive, Apple iCloud
  - Use APIs because of latency and failure scenarios (NAS protocols wouldn’t work well)

Storage Array

- Can just attach disks, or arrays of disks
- Avoids the NAS drawback of using network bandwidth
- Storage Array has controller(s), provides features to attached host(s)
  - Ports to connect hosts to array
  - Memory, controlling software (sometimes NVRAM, etc)
  - A few to thousands of disks
  - RAID, hot spares, hot swap (discussed later)
  - Shared storage -> more efficiency
  - Features found in some file systems
    - Snaphots, clones, thin provisioning, replication, deduplication, etc

Storage Area Network

- Common in large storage environments
- Multiple hosts attached to multiple storage arrays – flexible



Storage Area Network (Cont.)

- SAN is one or more storage arrays
  - Connected to one or more Fibre Channel switches or **InfiniBand** (**IB**) network
- Hosts also attach to the switches
- Storage made available via **LUN Masking** from specific arrays to specific servers
- Easy to add or remove storage, add new host and allocate it storage
- Why have separate storage networks and communications networks?
  - Consider iSCSI, FCOE

A Storage Array

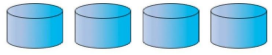
RAID Structure

- **RAID** – **redundant array of inexpensive disks**
  - multiple disk drives provides reliability via **redundancy**
- Increases the **mean time to failure**
- **Mean time to repair –** exposure time when another failure could cause data loss
- **Mean time to data loss** based on above factors
- If mirrored disks fail independently, consider disk with 1300,000 **mean time to failure** and 10 hour mean time to repair
  - Mean time to data loss is $100,000^2 / (2 * 10) = 500 * 10^6$ hours, or 57,000 years!
- Frequently combined with **NVRAM** to improve write performance
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively

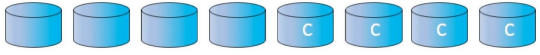
RAID (Cont.)

- 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 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

RAID Levels



(a) RAID 0: non-redundant striping.

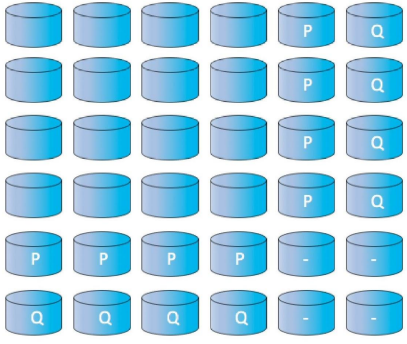


(b) RAID 1: mirrored disks.

(c) RAID 4: block-interleaved parity.

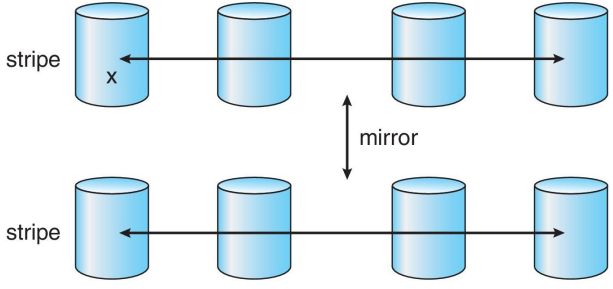
(d) RAID 5: block-interleaved distributed parity.

(e) RAID 6: P + Q redundancy.

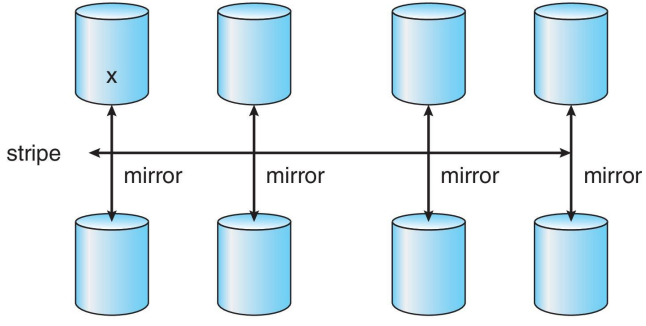


(f) Multidimensional RAID 6.

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



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



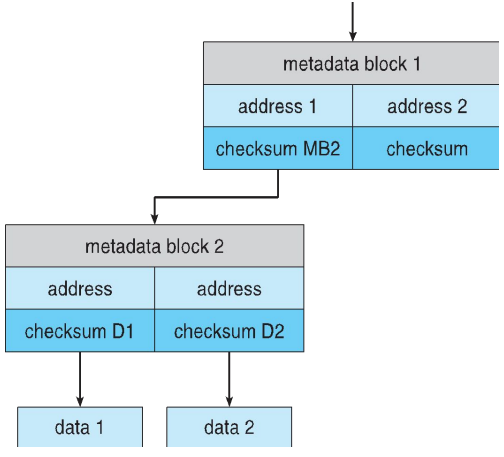
b) RAID 1 + 0 with a single disk failure.

Other Features

- Regardless of where RAID implemented, other useful features can be added
- **Snapshot** is a view of file system before a set of changes take place (i.e. at a point in time)
  - More in Ch 12
- Replication is automatic duplication of writes between separate sites
  - For redundancy and disaster recovery
  - Can be synchronous or asynchronous
- Hot spare disk is unused, automatically used by RAID production if a disk fails to replace the failed disk and rebuild the RAID set if possible
  - Decreases mean time to repair

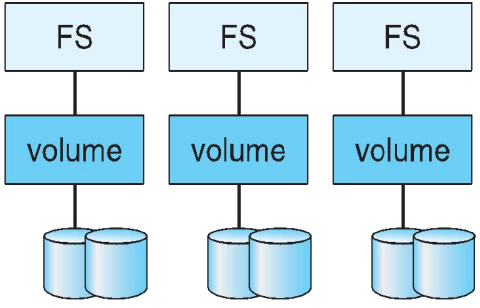
Extensions

- RAID alone does not prevent or detect data corruption or other errors, just disk failures
- Solaris ZFS adds **checksums** of all data and metadata
- Checksums kept with pointer to object, to detect if object is the right one and whether it changed
- Can detect and correct data and metadata corruption
- ZFS also removes volumes, partitions
  - Disks allocated in **pools**
  - Filesystems with a pool share that pool, use and release space like `malloc()` and `free()` memory allocate / release calls

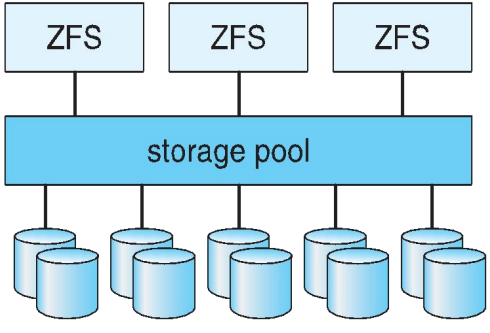


ZFS checksums all metadata and data

Traditional and Pooled Storage



(a) Traditional volumes and file systems



(b) ZFS and pooled storage.

Object Storage

- General-purpose computing, file systems not sufficient for very large scale
- Another approach – start with a storage pool and place objects in it
  - Object just a container of **data**
  - No way to navigate the pool to find objects (no directory structures, few services
  - Computer-oriented, not user-oriented
- Typical sequence
  - Create an object within the pool, receive an object ID
  - Access object via that ID
  - Delete object via that ID

Object Storage (Cont.)

- Object storage management software like **Hadoop file system** (**HDFS**) and **Ceph** determine where to store objects, manages protection
  - Typically by storing N copies, across N systems, in the object storage cluster
  - **Horizontally scalable**
  - **Content addressable**, **unstructured**

End of Chapter 11