12 Mass-Storage Systems

Chapter 11 in the book

What We'll Cover Today:

1. Overview of Mass Storage Structure
2. Hard Disk Drives (HDDs): Mechanics & Performance
3. Disk Scheduling Algorithms
4. Comparative Analysis & Modern Implementations

Learning Objectives:

- Understand physical vs. logical disk organization
- Calculate disk access time from seek time, rotational latency, transfer rate
- Compare and apply disk scheduling algorithms (FCFS, SSTF, SCAN, C-SCAN)
- Recognize modern OS disk scheduling implementations

Historical Perspective

First Commercial Disk Drive:

IBM 350 **(1956)**

- Capacity: 5 MB (yes, megabytes!)
- Size: 50 × 24-inch platters
- Access time: < 1 second (revolutionary for its time)

Historical Perspective

Evolution Timeline:

- 1956: First HDD (5 MB)
- 1980: First GB-capacity drive (1 GB, $40,000)
- 1991: First 1" drive (40 MB)
- 2007: First consumer SSD
- 2024: 20+ TB HDDs, 8+ TB SSDs

Fun Fact:

The first 1GB HDD

- weighed over 500 lbs
- and cost $40,000!

Modern Storage Hierarchy

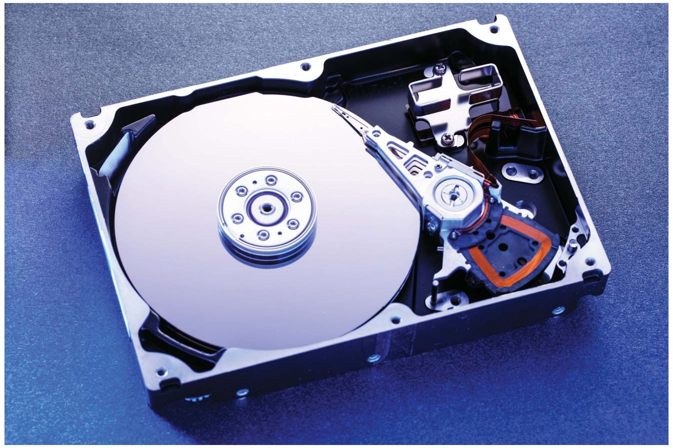
```
CPU Registers (1 ns)

    ↓

CPU Cache (SRAM, ~10 ns)

    ↓

Main Memory (DRAM, ~100 ns)

    ↓

Solid-State Drives (SSD, ~100 μs)

    ↓

Hard Disk Drives (HDD, ~10 ms)

    ↓

Cloud/Network Storage (~100 ms)
```

Key Insight:

Each level is ~10× slower but 10× larger/cheaper

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



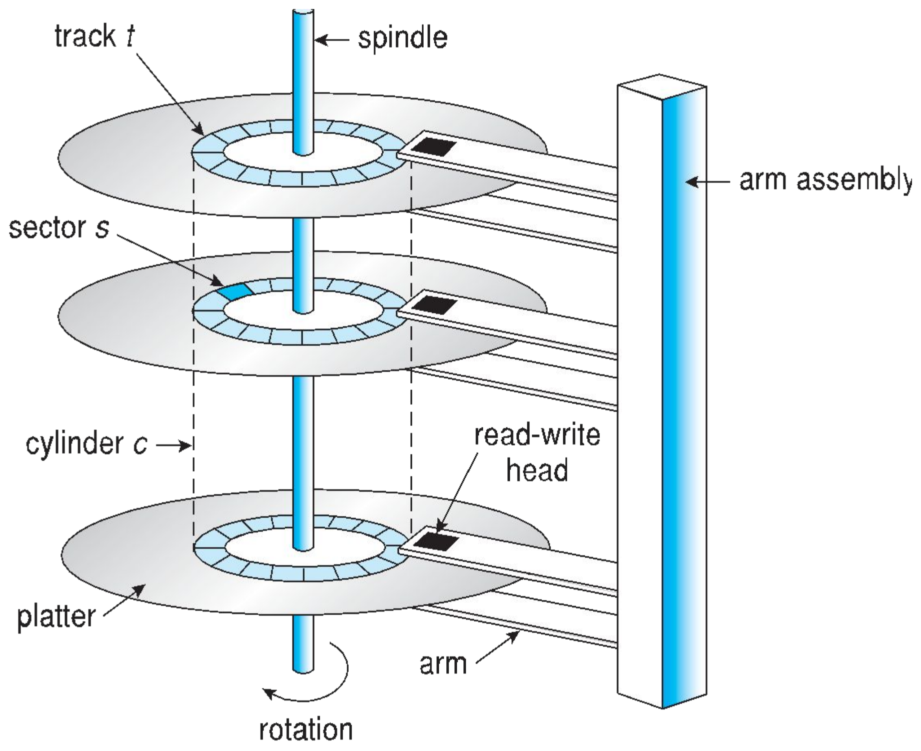
Hard Disk Drive (HDD) Anatomy

Physical Components:

- **Platters:** Circular magnetic-coated disks (aluminum/glass)
- **Spindle:** Rotates platters at constant speed (5400–15000 RPM)
- **Read/Write Heads:** One per platter surface, float on air cushion
- **Actuator Arm:** Moves heads across platters
- **Controller:** Electronics for data management

Logical Organization:

- **Sector:** Smallest addressable unit (traditionally 512B, now 4KB)
- **Track:** Concentric circle of sectors
- **Cylinder:** Same track position across all platters



Disk positioning system

1. Seek Time

- Time to move head to correct track
- Typical: 3–12 ms
- *Average seek time* often measured as 1/3 of full stroke

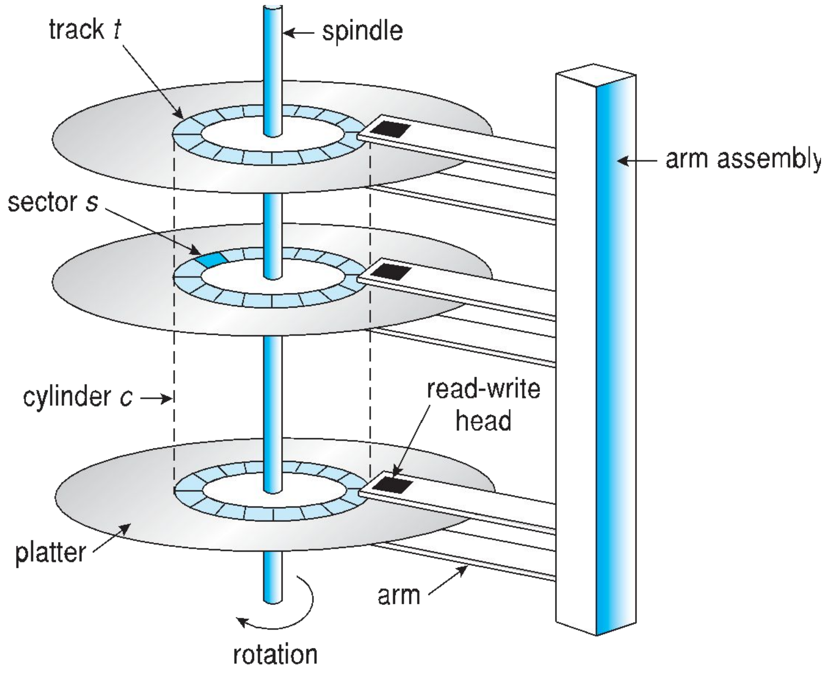
2. Rotational Latency

- Time for correct sector to rotate under head
- Depends on RPM: Average = ½ rotation time
- Formula:

$$\text{Avg Latency} = \frac{60 \text{ seconds}}{2 \times \text{RPM}} \times 1000 \text{ ms}$$

3. Transfer Rate

- Data movement speed between disk and controller
- Theoretical vs. effective (due to overhead)



4. Access Time Formula:

$$T_{\text{access}} = T_{\text{seek}} + T_{\text{rotation}} + T_{\text{transfer}} + T_{\text{controller}}$$

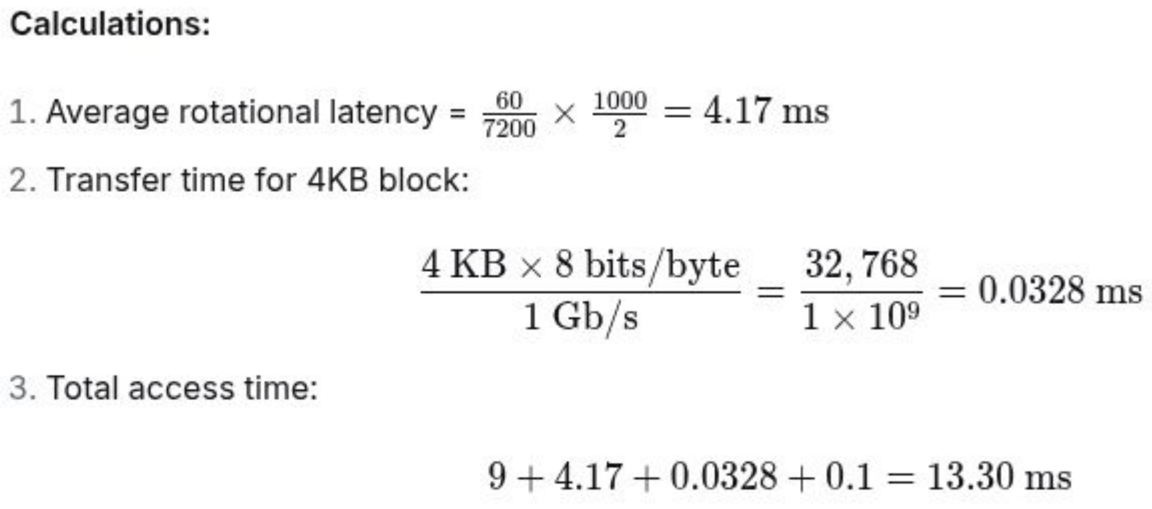
HDD Performance Example

Scenario: 7200 RPM drive,

9 ms average seek,

1 Gb/s transfer rate,

0.1 ms controller overhead



Calculations:

1. Average rotational latency = $\frac{60}{7200} \times \frac{1000}{2} = 4.17 \text{ ms}$
2. Transfer time for 4KB block:

$$\frac{4\text{ KB} \times 8\text{ bits/byte}}{1\text{ Gb/s}} = \frac{32,768}{1 \times 10^9} = 0.0328\text{ ms}$$

3. Total access time:

$$9 + 4.17 + 0.0328 + 0.1 = 13.30\text{ ms}$$

Key Insight: Transfer time is negligible for small blocks; seek + rotation dominate!

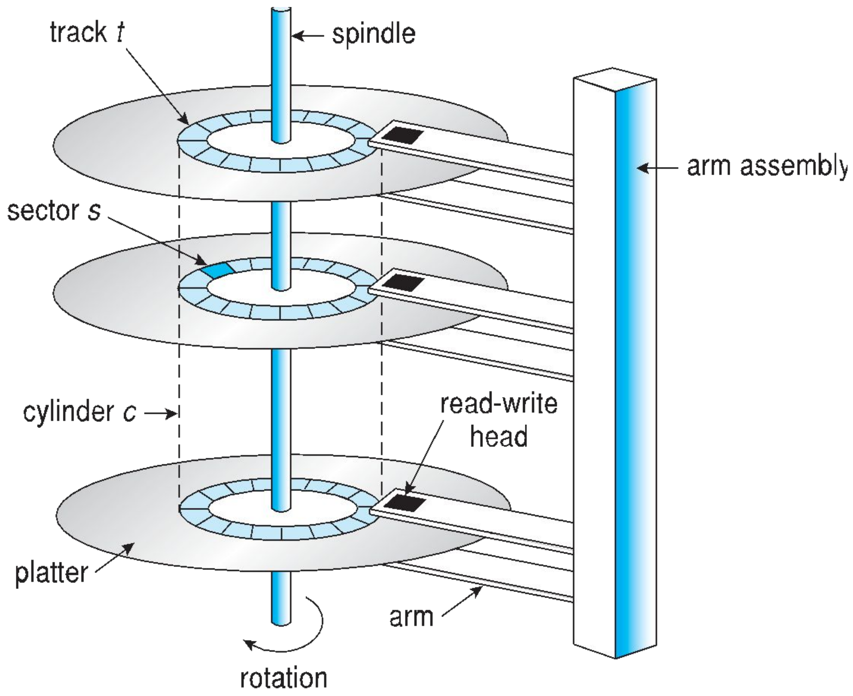
The Seek Process Breakdown

Four Phases of a Seek:

1. **Speedup**: Accelerate arm to maximum velocity (40g acceleration!)
2. **Coast:** Move at constant speed (for long seeks only)
3. **Slowdown:** Decelerate near target track
4. **Settle:** Fine adjustment onto track center (~1 ms)

Interesting Detail:

- Very short seeks (< 10 tracks): dominated by settle time
- Short seeks (200–400 tracks): dominated by acceleration
- Long seeks: include coast phase



Disk Addressing & Zoning

Logical Block Addressing (LBA):

- OS sees linear array of sectors (0, 1, 2, ...)
- Disk controller maps to physical (cylinder, head, sector)

Some Optimizations:

1. **Zoning:** Outer tracks have more sectors (higher linear density)
2. **Track Skewing:** Offset sector 0 between tracks for sequential access
3. **Sector Sparing:** Remap bad sectors to reserved areas

OS Challenge:

Doesn't know physical mapping!

- Consecutive LBAs might be physically distant
- Can't optimize for rotational positioning

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

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

How to schedule concurrent requests?

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

Why Disk Scheduling Matters

Problem: Multiple I/O requests arrive concurrently

Goal: Minimize average response time while maintaining fairness

Key Observations:

1. Sequential I/O >> Random I/O (10–100×faster)
2. Long seeks much slower than short ones
3. Rotational position affects performance significantly

Example: Without scheduling, random 4KB reads might give < 100 IOPS

With good scheduling: potentially 200+ IOPS (2× improvement!)

Disk Scheduling Algorithms

Requests: 98, 183, 37, 122, 14, 124, 65, 67
Starting head position: 53
Disk range: 0–199 cylinders

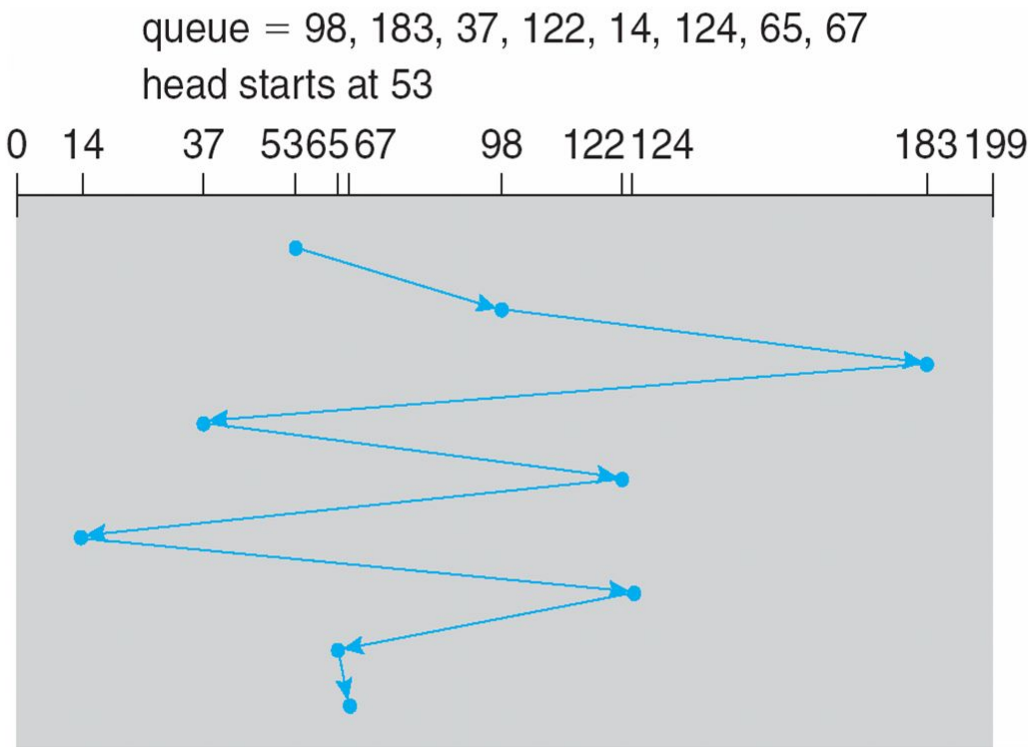
Algorithms

1. **FCFS**: First-Come, First-Served (baseline)
2. **SSTF**: Shortest Seek Time First (greedy)
3. **SCAN**: Elevator algorithm (back-and-forth)
4. **C-SCAN**: Circular SCAN (one-directional)
5. **LOOK/C-LOOK**: Optimized SCAN variants

Scheduling: FCFS

"First Come First Served"

- Process requests in arrival order



Our Example Path: 53 → 98 → 183 → 37 → 122 → 14 → 124 → 65 → 67

Scheduling: FCFS

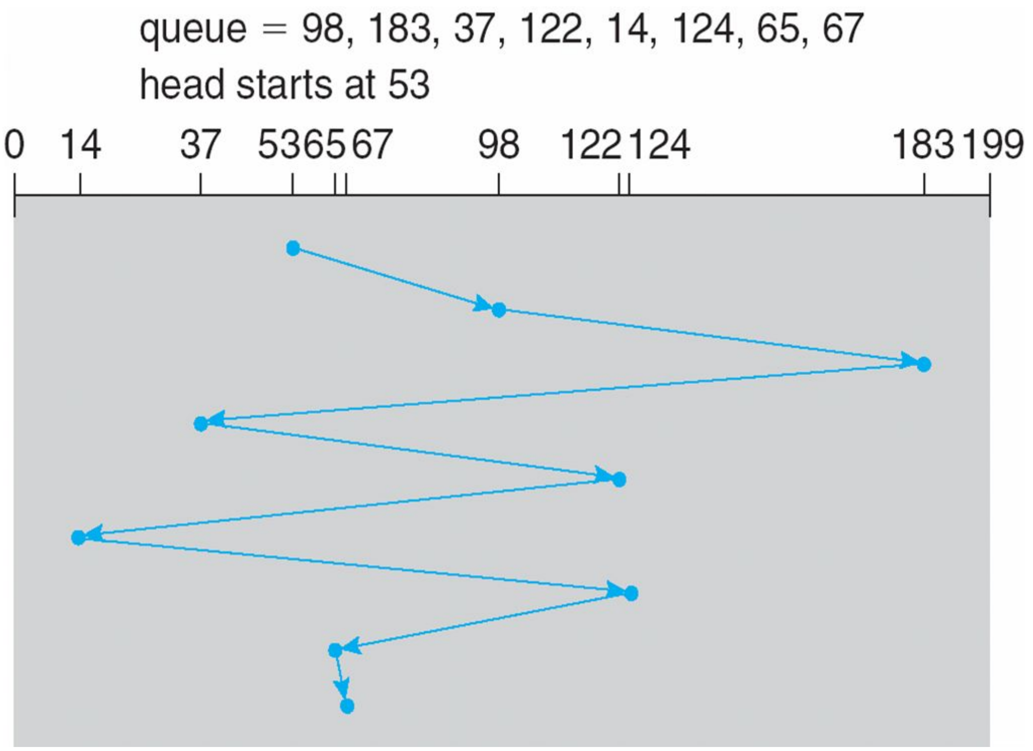
Advantages

- Easy to implement
- Fairness guaranteed (no starvation)

Disadvantages

- Poor performance (no locality exploitation)
- High average seek time

Use Case: Rarely used alone in modern systems



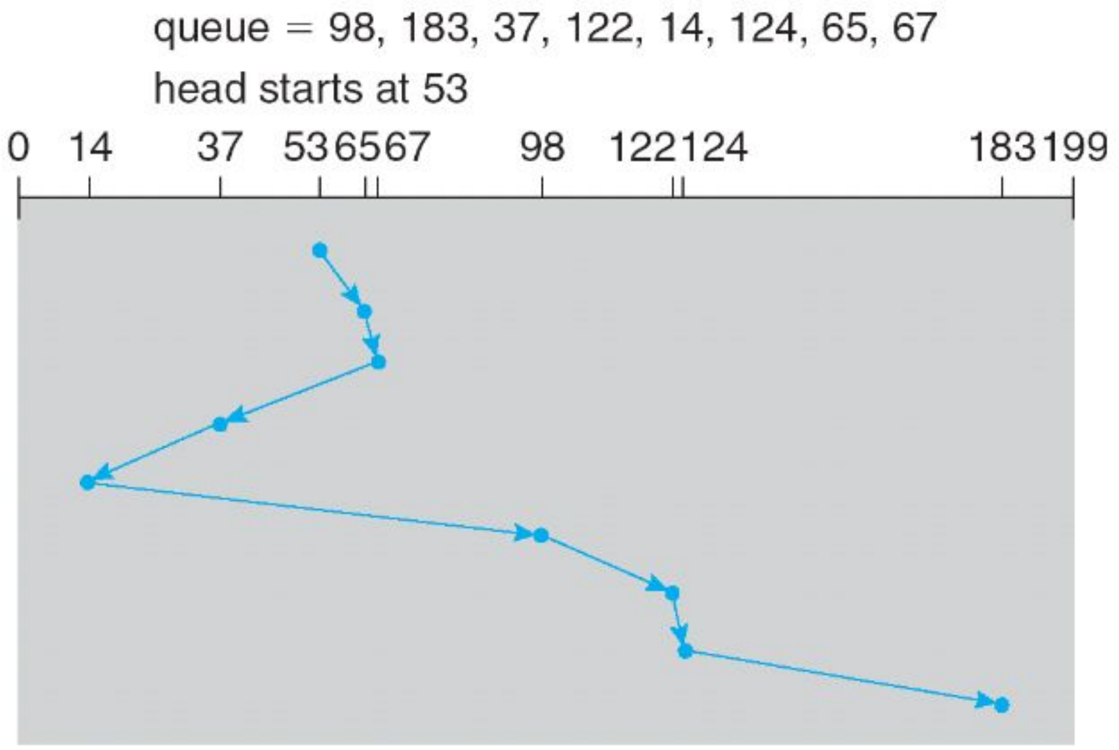
https://www.scs.stanford.edu/24wi-cs212/notes/io_disks.pdf

Shortest positioning time first (SPTF)

Shortest positioning time first (SPTF)

- Always serve closest request

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



Our Example Path: 53 → 65 → 67 → 37 → 14 → 98 → 122 → 124 → 183

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

Shortest positioning time first (SPTF)

Advantages

- Exploits locality of disk requests
  - Minimizes average response time
- Higher throughput

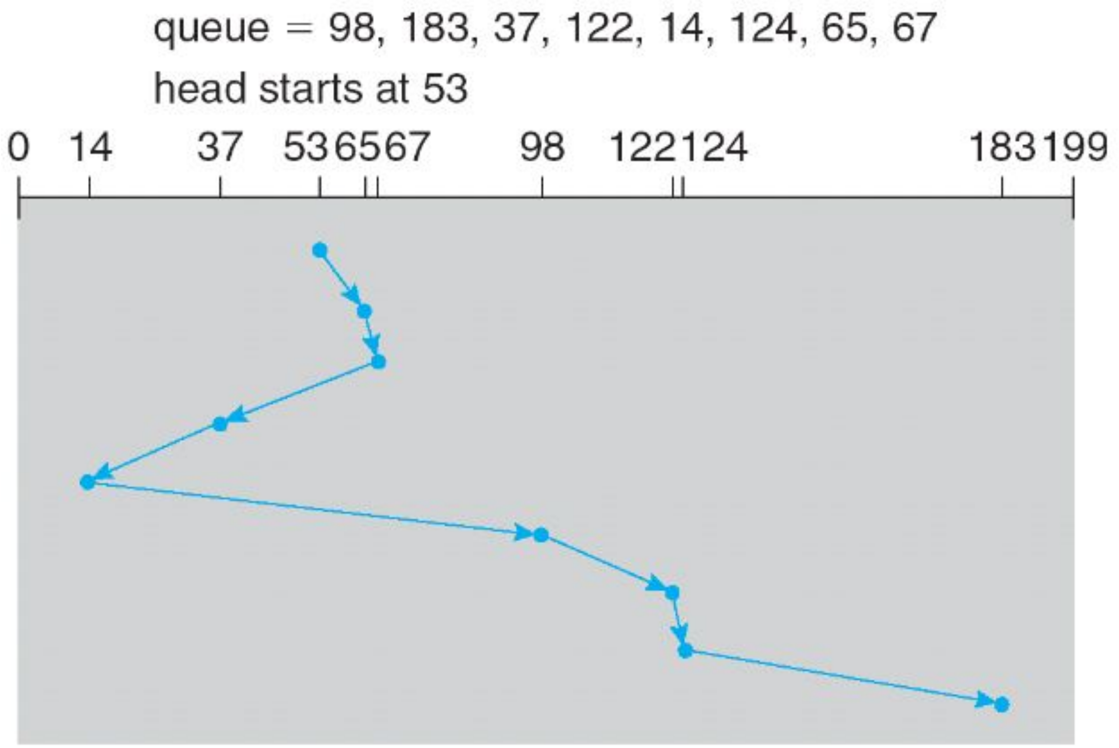
Disadvantages

- **Starvation**
- Don't always know what request will be fastest
  - Not optimal for rotational positioning

Improvement: Aged SPTF

- Give older requests higher priority
- Adjust "effective" seek time with weighting factor:

  Teff = Tpos − W x Twait



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

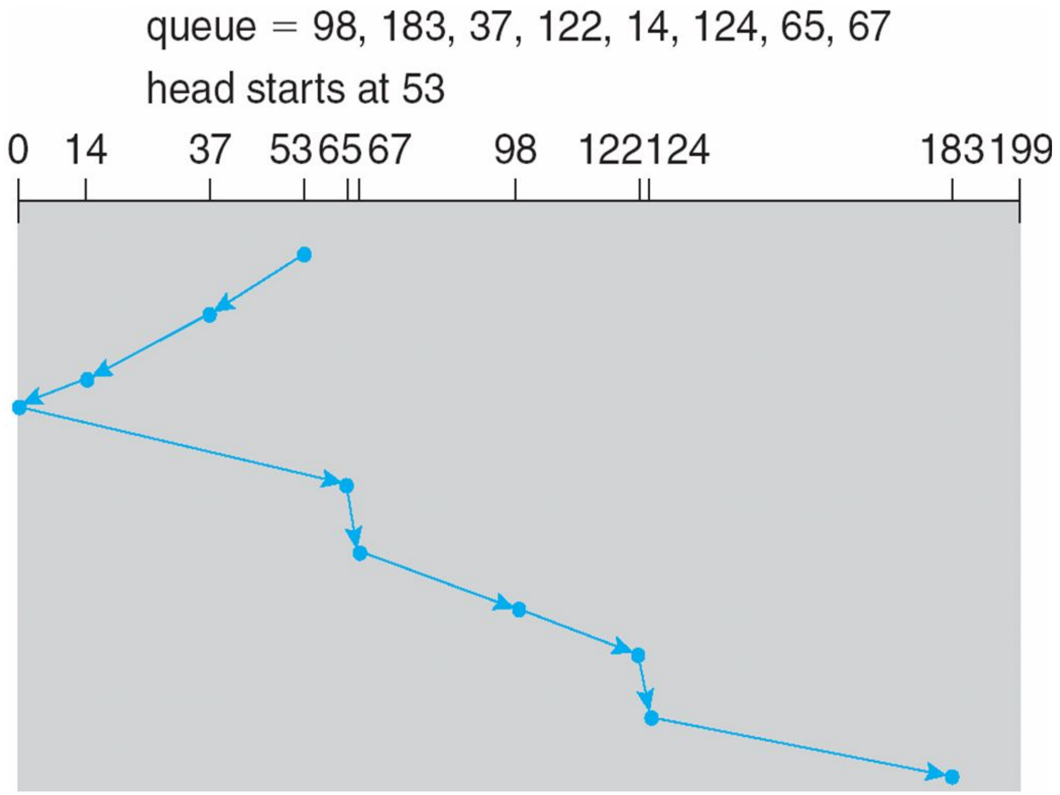
“Elevator” scheduling (SCAN)

Sweep across disk, servicing all requests passed

- Like SPTF, but next seek must be in same direction

Algorithm:

1. Move head in one direction
2. Service all requests along the way
3. Reverse direction when no requests ahead
4. Repeat



Our Example (starting at 53, moving toward 0):
53 → 37 → 14 → 0 → 65 → 67 → 98 → 122 → 124 → 183

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

“Elevator” scheduling (SCAN)

Advantages

- Takes advantage of locality
- Bounded waiting

Disadvantages

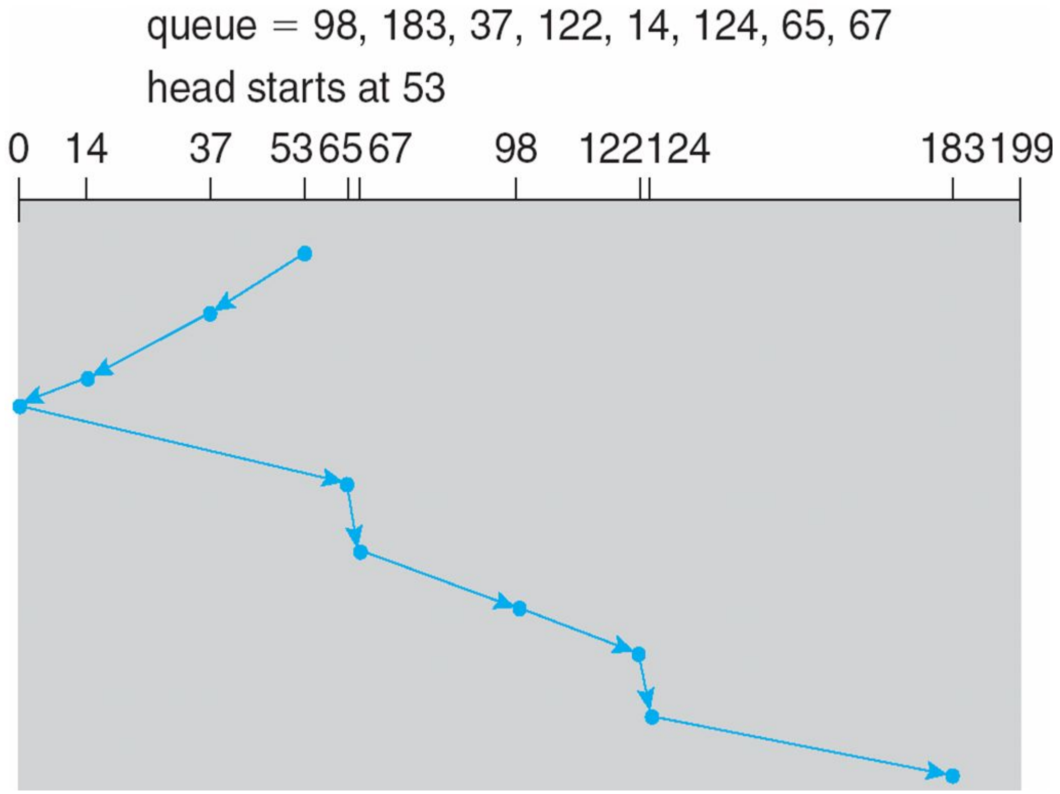
- Cylinders in the middle get better service
- Might miss locality SPTF could exploit

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

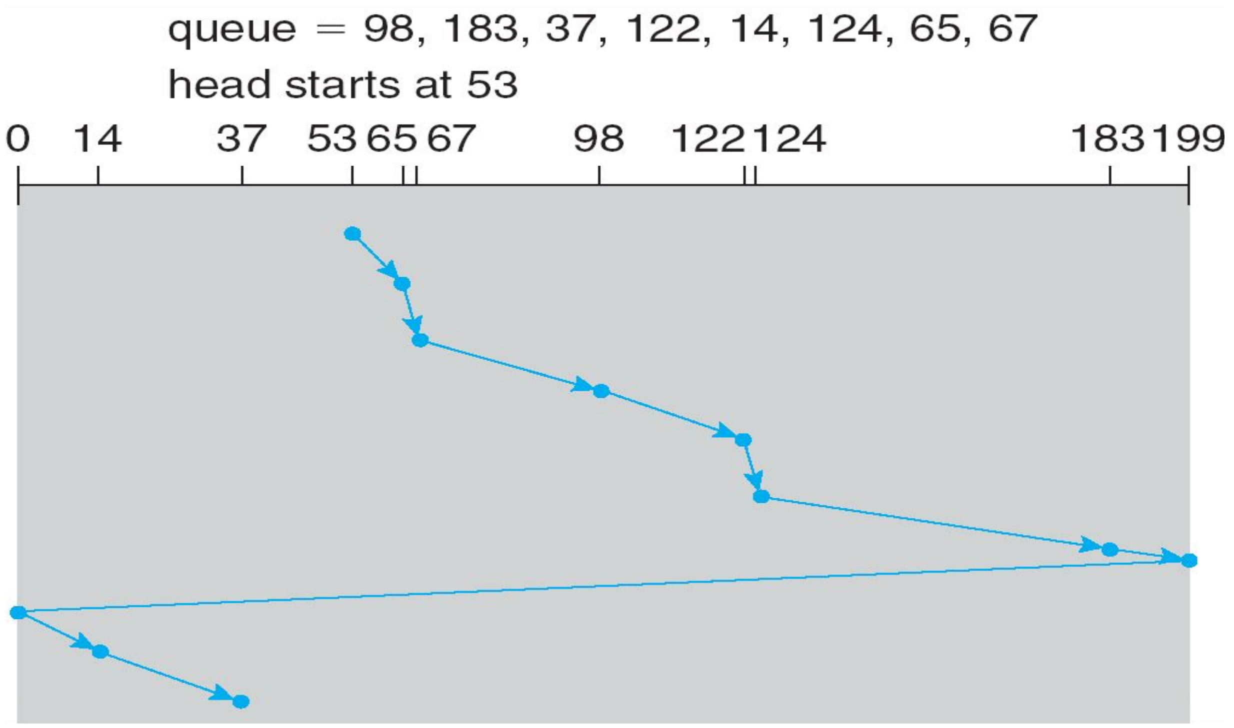
Algorithm:

1. Move head in one direction, servicing requests
2. When reaching end, jump to opposite end without servicing
3. Continue in same direction

Advantages:

- More uniform wait time than SCAN
- Commonly used in Unix/Linux

Variants: LOOK and C-LOOK stop at last request, not physical end



Our Example (starting at 53, moving upward):
53 → 65 → 67 → 98 → 122 → 124 → 183 → 199 → 0 → 14 → 37

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

Algorithm	Avg Seek Time	Starvation?	Throughput	Fairness
FCFS	High	No	Low	Excellent
SSTF	Low	Yes	High	Poor
SCAN	Medium	No	Medium-High	Good
C-SCAN	Medium	No	Medium-High	Very Good

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

Linux Disk Schedulers (as of 2025):

1. mq-deadline: Default for most SSDs
   - Per-request deadlines to prevent starvation
2. bfq (Budget Fair Queueing): Default for rotational disks
   - Ensures fair bandwidth distribution
3. kyber: For low-latency devices (NVMe)
   - Simple, self-tuning

Windows: Uses multi-level queue with C-LOOK variant

Key Insight: Modern OSes use different schedulers for different device types

Exercise

Queue: 45, 21, 67, 90, 12, 150, 33, 78
Start: 50 cylinders
Direction: Initially moving upward

Question: Which algorithm gives shortest total seek distance?
A) FCFS
B) SSTF
C) SCAN
D) C-SCAN

Think-Pair-Share: Discuss with neighbor for 2 minutes!

Programming:

Implement a disk scheduling simulator in Python!

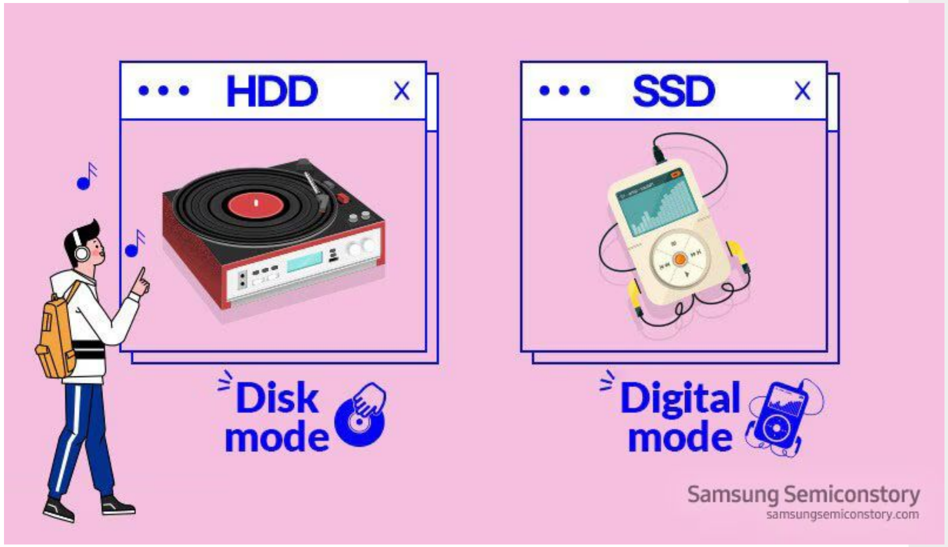
Compare performance on random input strings

Summary

1. HDD Performance = seek time + rotational latency + transfer time
2. Sequential I/O is dramatically faster than random I/O
3. Disk scheduling trades off throughput vs. fairness
4. Modern systems use adaptive schedulers based on device type
5. Understanding these concepts helps optimize database, file system, and application performance

Next: Non-Volatile Memory (SSDs), Flash Translation Layer, and RAID!

Non-Volatile Memory (NVM) 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

SSD Performance Characteristics

Operation	HDD	SSD (SATA)	SSD (NVMe)
Read 4KB random	10-15 ms	100 µs	50 µs
Write 4KB random	10-15 ms	150 µs	70 µs
Sequential Read	200 MB/s	550 MB/s	3500 MB/s
Sequential Write	200 MB/s	500 MB/s	3000 MB/s

IOPS Comparison:

- HDD: 100-200 IOPS
- SATA SSD: 50,000-100,000 IOPS
- NVMe SSD: 500,000-1,000,000+ IOPS

Key Insight: SSDs excel at random I/O, HDDs better at sequential throughput per $

NVM Device Landscape

Type	Use Case	Characteristics
NAND Flash	Main storage (SSDs, USB drives)	High density, block erase, needs FTL
3D XPoint (Optane)	Cache/tiered storage	Byte-addressable, high endurance
NOR Flash	Firmware/BIOS storage	Random read, execute-in-place
MRAM/FRAM	Specialized applications	Unlimited endurance, fast

Focus: NAND Flash (99% of consumer SSDs)

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

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

The Shift to Solid-State Storage

From Spinning to Static:

- 2007: Apple introduces first SSD in MacBook Air
- 2024: SSDs dominate laptops, increasingly in servers
- Key Trend: Price per GB of SSD dropped 100× in 10 years!

Why the Shift?

- Performance: 100× faster random access than HDD
- Durability: No moving parts → resistant to shock/vibration
- Power: Lower energy consumption (1-3W vs. 5-10W)
- Form Factor: Smaller (M.2, U.2)

But... SSDs have unique challenges!

NAND Flash Memory Cells

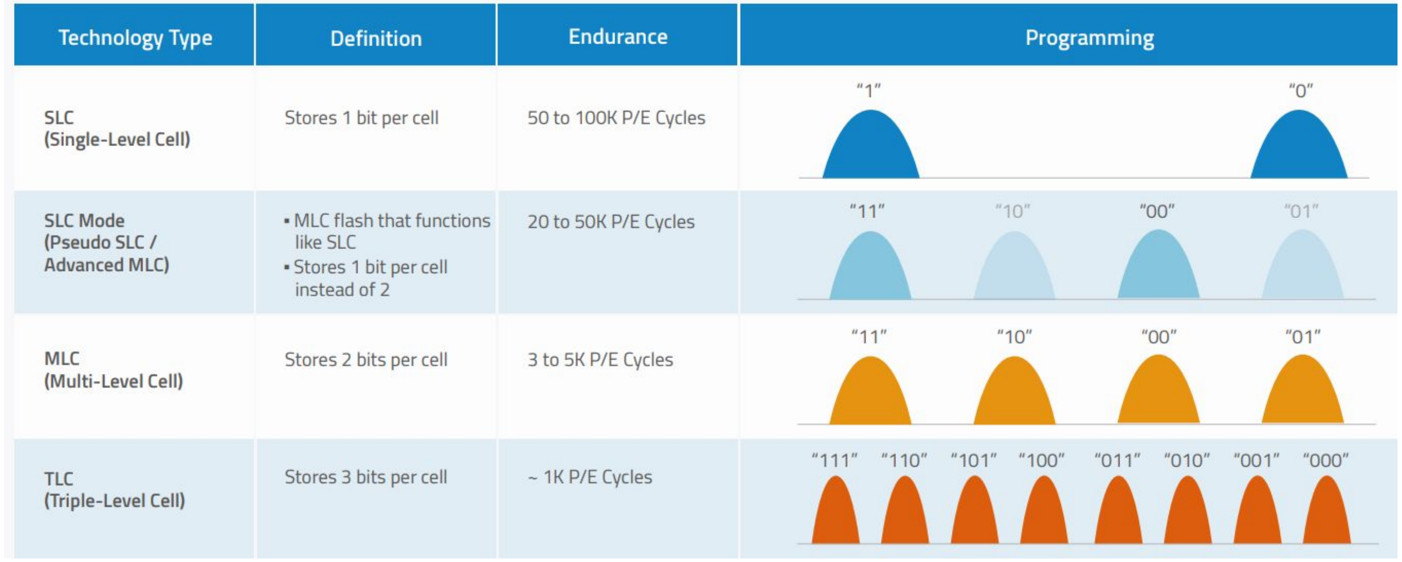
How It Stores Data:

- Floating-gate transistor traps electrons
- Charge level = bit value
- Key Limitation: Cells wear out with erase cycles

Cell Types (Trade-off: Cost vs. Performance vs. Endurance):

1. SLC (Single-Level Cell): 1 bit/cell, 100K+ cycles, expensive
2. MLC (Multi-Level Cell): 2 bits/cell, 10K cycles, mainstream
3. TLC (Triple-Level Cell): 3 bits/cell, 3K cycles, consumer
4. QLC (Quad-Level Cell): 4 bits/cell, 1K cycles, capacity-focused

Trend: More bits/cell → cheaper but slower, less durable



Technology Type	Definition	Endurance	Programming
SLC (Single-Level Cell)	Stores 1 bit per cell	50 to 100K P/E Cycles	"1" "0"
SLC Mode (Pseudo SLC / Advanced MLC)	▪ MLC flash that functions like SLC ▪ Stores 1 bit per cell instead of 2	20 to 50K P/E Cycles	"11" "10" "00" "01"
MLC (Multi-Level Cell)	Stores 2 bits per cell	3 to 5K P/E Cycles	"11" "10" "00" "01"
TLC (Triple-Level Cell)	Stores 3 bits per cell	~ 1K P/E Cycles	"111" "110" "101" "100" "011" "010" "001" "000"

https://www.simms.co.uk/Uploads/Resources/50/f4366381-b992-425a-bec4-cca409b51a6c.pdf

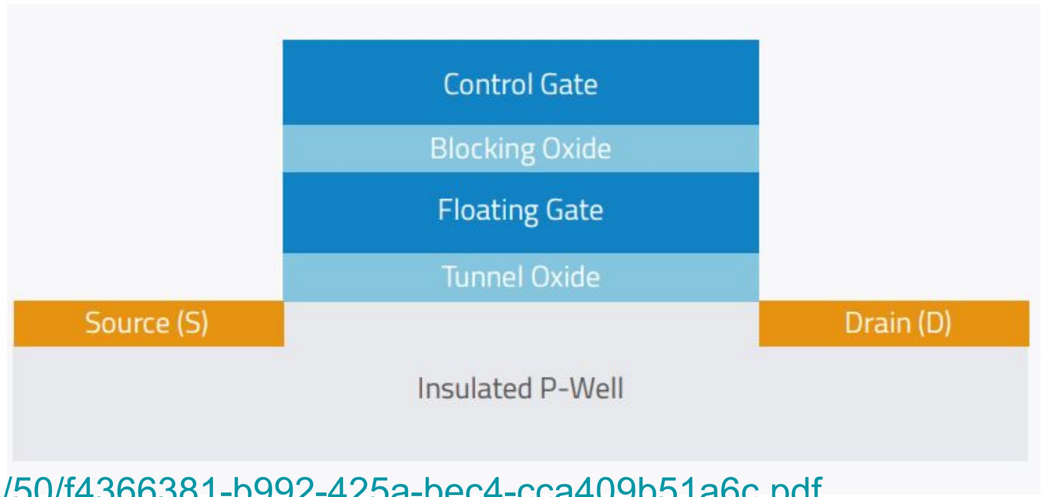
NAND Flash Cell Structure

The basic building block of flash memory is the NAND flash cell.

Each cell contains a floating gate transistor, where the data are stored, through program/erase operations.

Oxide layers insulate the Floating Gate (storage layer), preventing the electrons to leak out of it.

Any electron stored in the FG is trapped, which means the cell is programmed (charged) and has a binary value of 0. When the FG holds no charge, it has a binary value of 1.

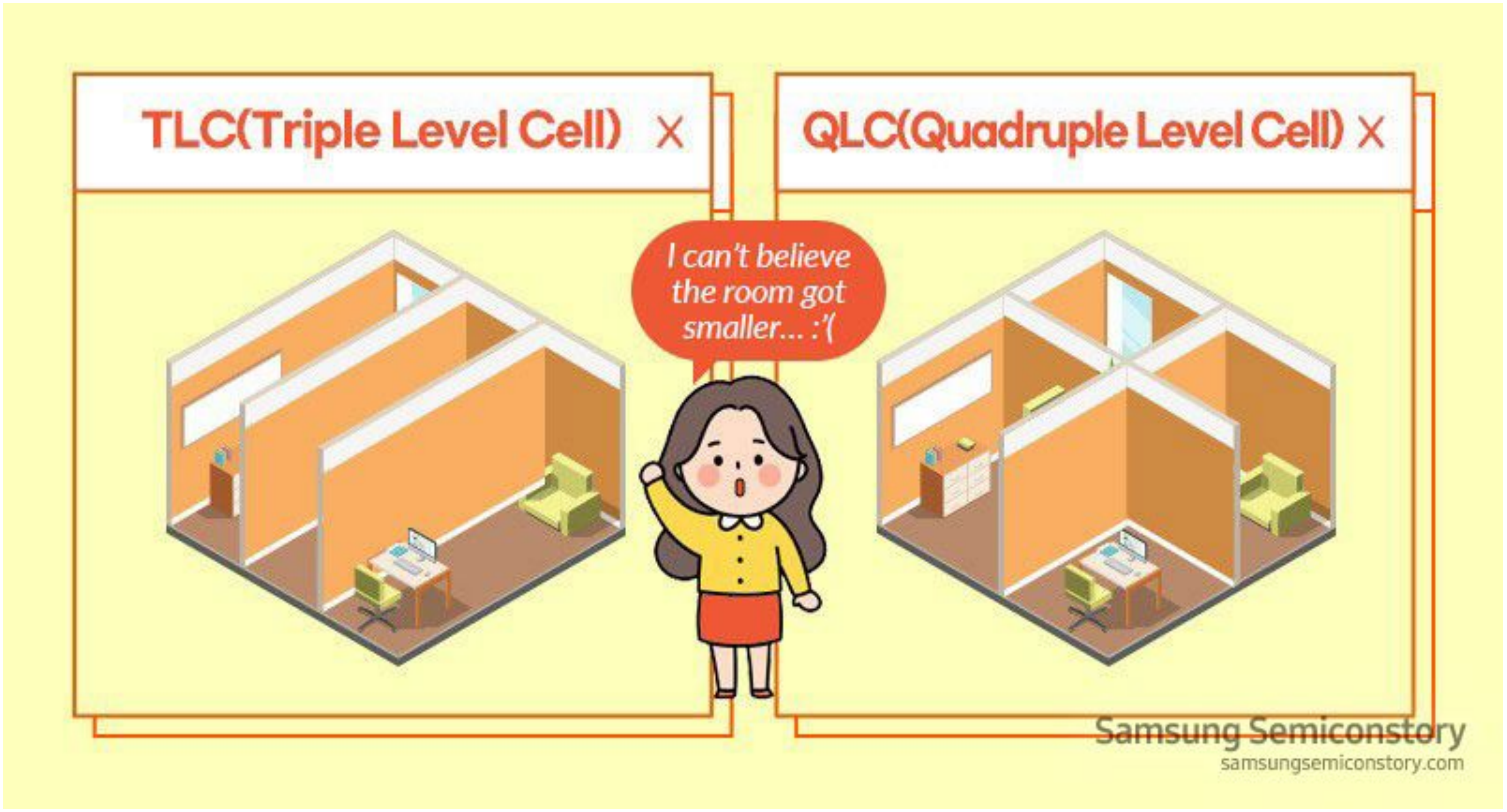


https://www.simms.co.uk/Uploads/Resources/50/f4366381-b992-425a-bec4-cca409b51a6c.pdf

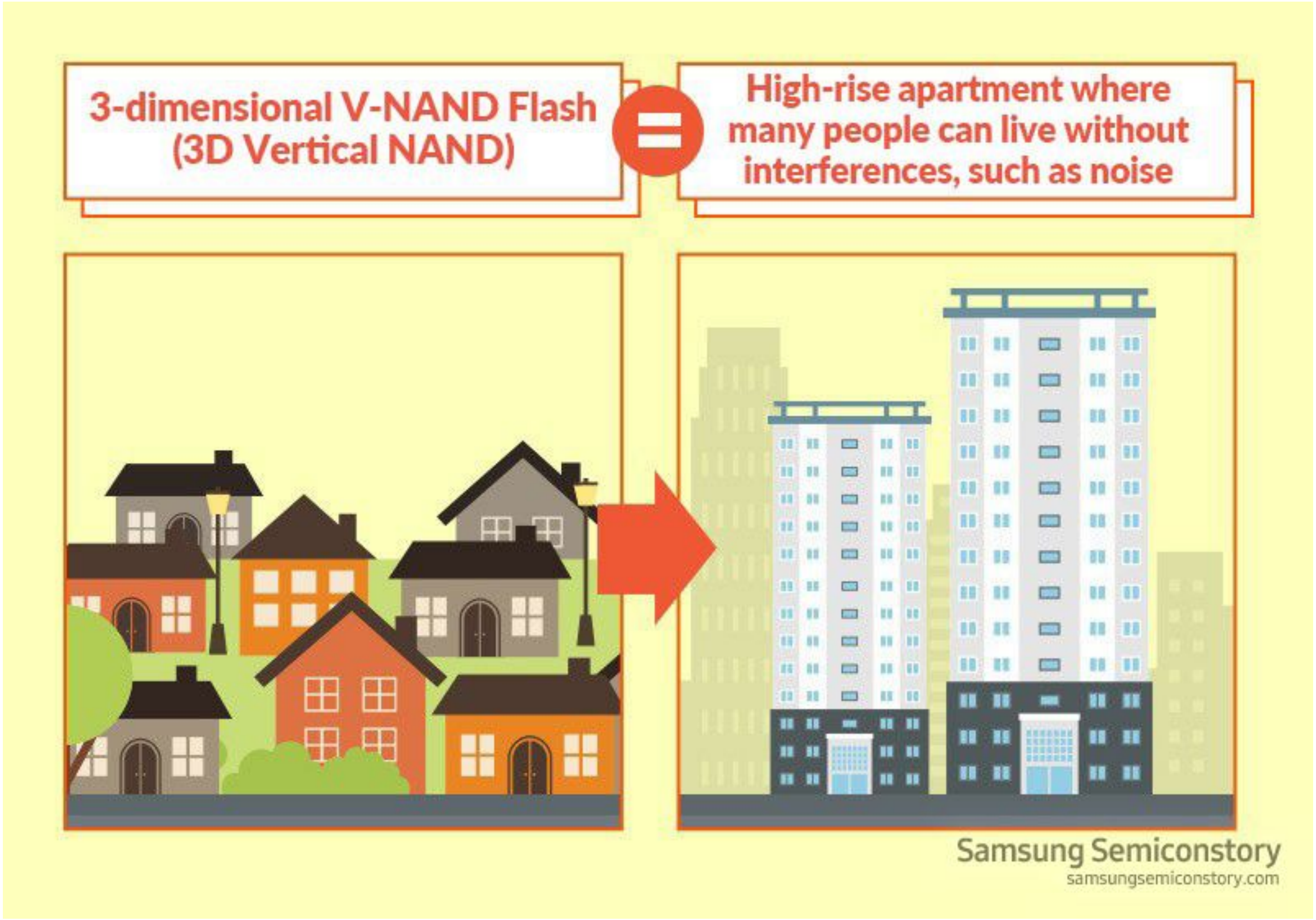
SLC(Single Level Cell)
A studio used by one person
A spacious room that I get to use alone ♡
MLC(Multi Level Cell)
Two people using a studio by dividing it with dividers
Samsung Semiconstory
samsungsemiconstory.com

SLC (Single Level Cell), a single data is stored in a single cell in the form of a ‘1’ or a ‘0’. This data uses the whole room for itself, and therefore the rent (or price) is high. That said, in SLC NAND flash the ‘head count’ process takes the least amount of time.

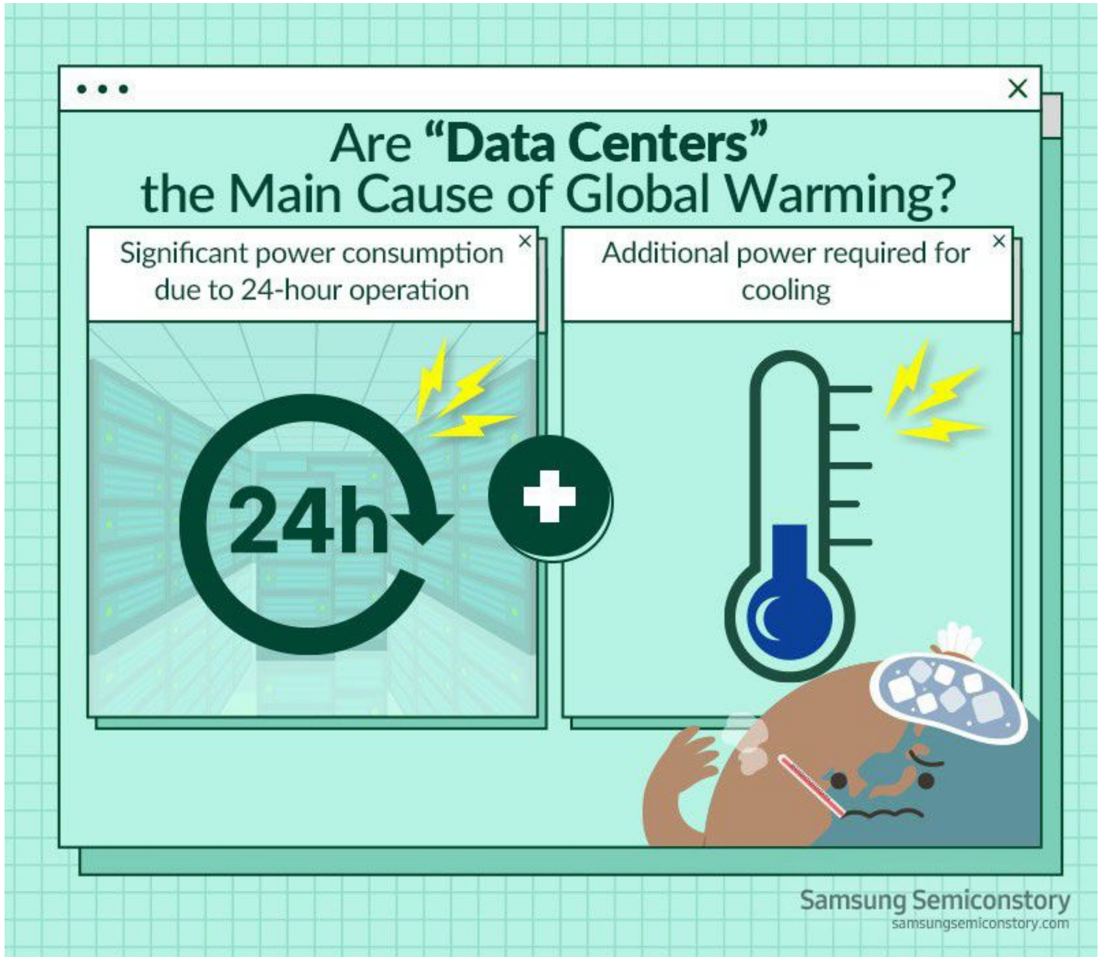
Two pieces of data are stored in a single cell, in the form of ‘00, 01, 10, 11’ and so on. The rooms here are smaller, making MLC cheaper than SLC. But data processing speed is slow compared to SLC.


TLC(Triple Level Cell) X
QLC(Quadruple Level Cell) X
I can't believe the room got smaller... :'(
Samsung Semiconstory
samsungsemiconstory.com

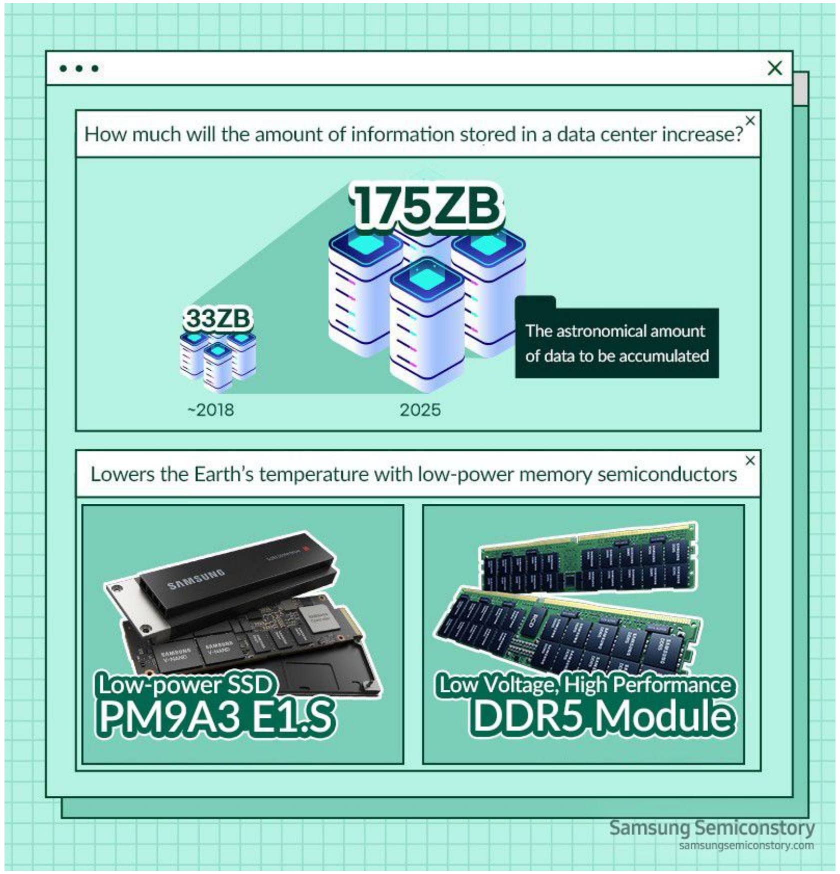
Three pieces of data are stored in a single cell in TLC (Triple Level Cell), while four pieces of data are stored in a QLC (Quadruple Level Cell). TLC and QLC are capable of storing large data volumes, but with more data per room, their data processing speeds are relatively slow compared to SLC and MLC.

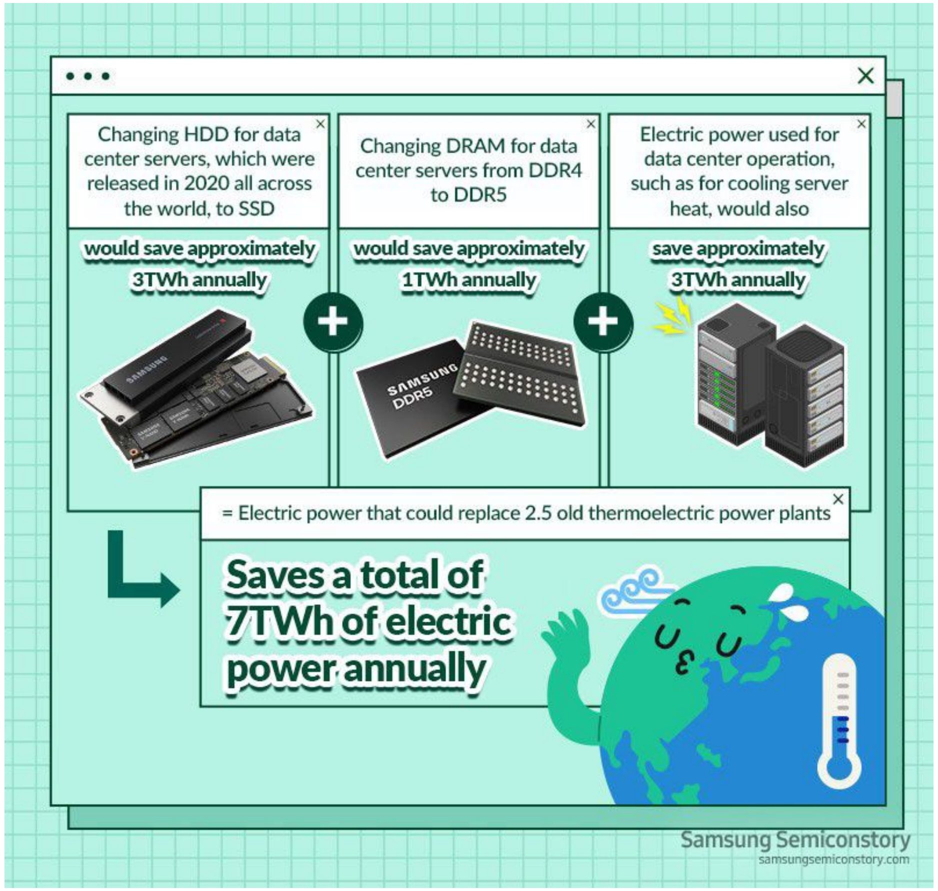

3-dimensional V-NAND Flash (3D Vertical NAND)
High-rise apartment where many people can live without interferences, such as noise
Samsung Semiconstory
samsungsemiconstory.com

Low-power Semiconductor


Are "Data Centers" the Main Cause of Global Warming?
Significant power consumption due to 24-hour operation
Additional power required for cooling
24h
Samsung Semiconstory
samsungsemiconstory.com

Low-power Semiconductor


How much will the amount of information stored in a data center increase?
175ZB
33ZB
~2018
2025
The astronomical amount of data to be accumulated
Lowers the Earth's temperature with low-power memory semiconductors
Low-power SSD PM9A3 E1.S
Low Voltage, High Performance DDR5 Module
Samsung Semiconstory
samsungsemiconstory.com


Changing HDD for data center servers, which were released in 2020 all across the world, to SSD
would save approximately 3TWh annually
Changing DRAM for data center servers from DDR4 to DDR5
would save approximately 1TWh annually
Electric power used for data center operation, such as for cooling server heat, would also
save approximately 3TWh annually
= Electric power that could replace 2.5 old thermoelectric power plants
Saves a total of 7TWh of electric power annually
Samsung Semiconstory
samsungsemiconstory.com

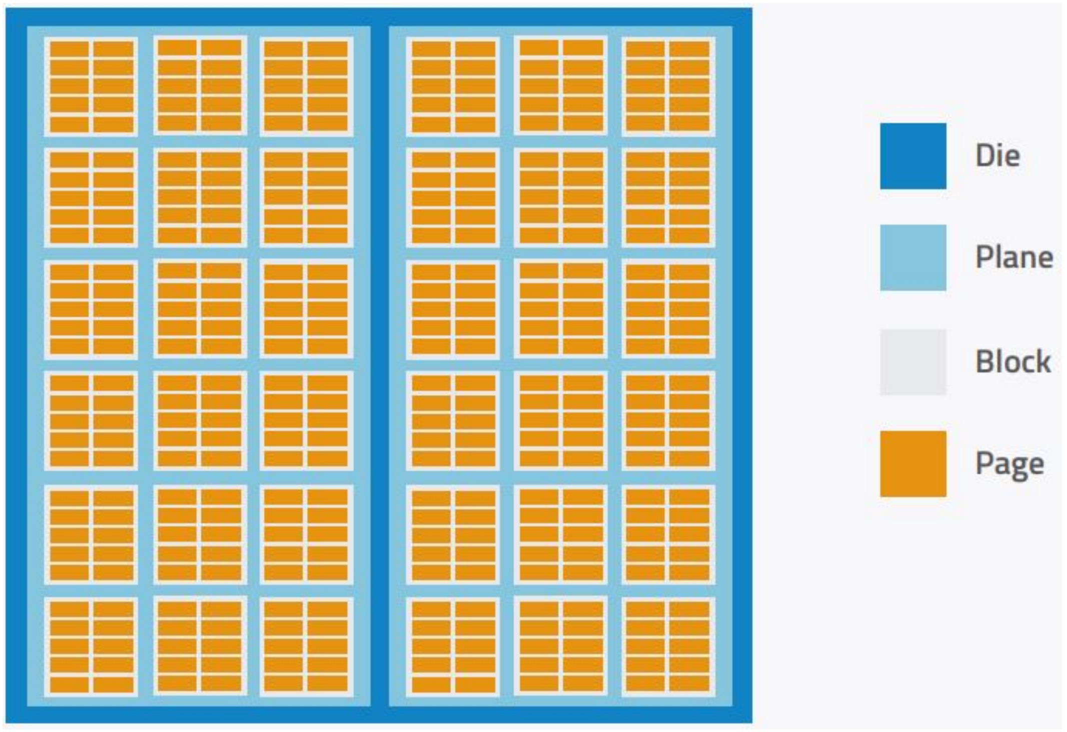
NAND Flash Layout

A physical NAND Page is the group of NAND flash cells belonging to the same block,which share, horizontally, the same Control Gate (called Word Line).

A NAND Block is composed of several pages.

Several NAND blocks form a **Plane.**

Finally, planes form a **Die.** Each memory chip contains one or more dies.



https://www.simms.co.uk/Uploads/Resources/50/f4366381-b992-425a-bec4-cca409b51a6c.pdf

NAND Flash Architecture

Physical Hierarchy:

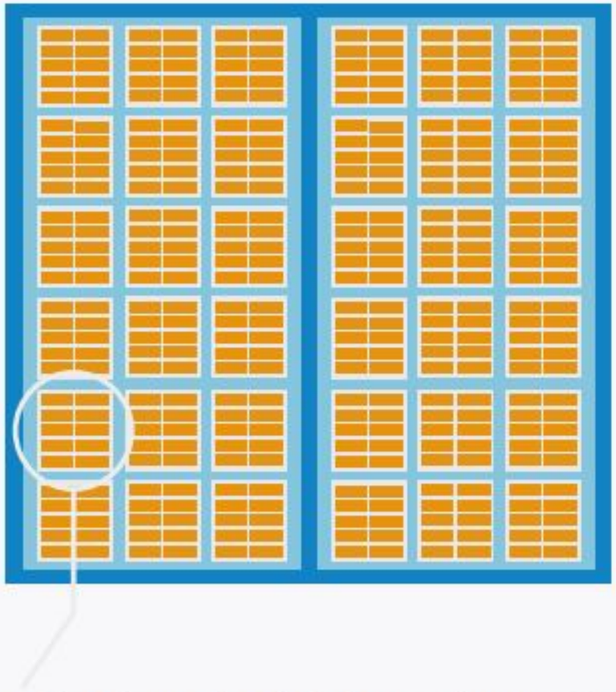
1. Die: Single silicon chip
2. Plane (2-4 per die): Parallel operations
3. Block (128-256 pages): **Smallest erasable unit**
4. Page (4-16KB): **Smallest read/write unit**

Critical Asymmetry:

- Read: Page level (~25 μs)
- Write: Page level (200-800 μs)
- **Erase: Block level (2-4 ms)** ← **The bottleneck!**

Analogy: Like a notebook where you can:

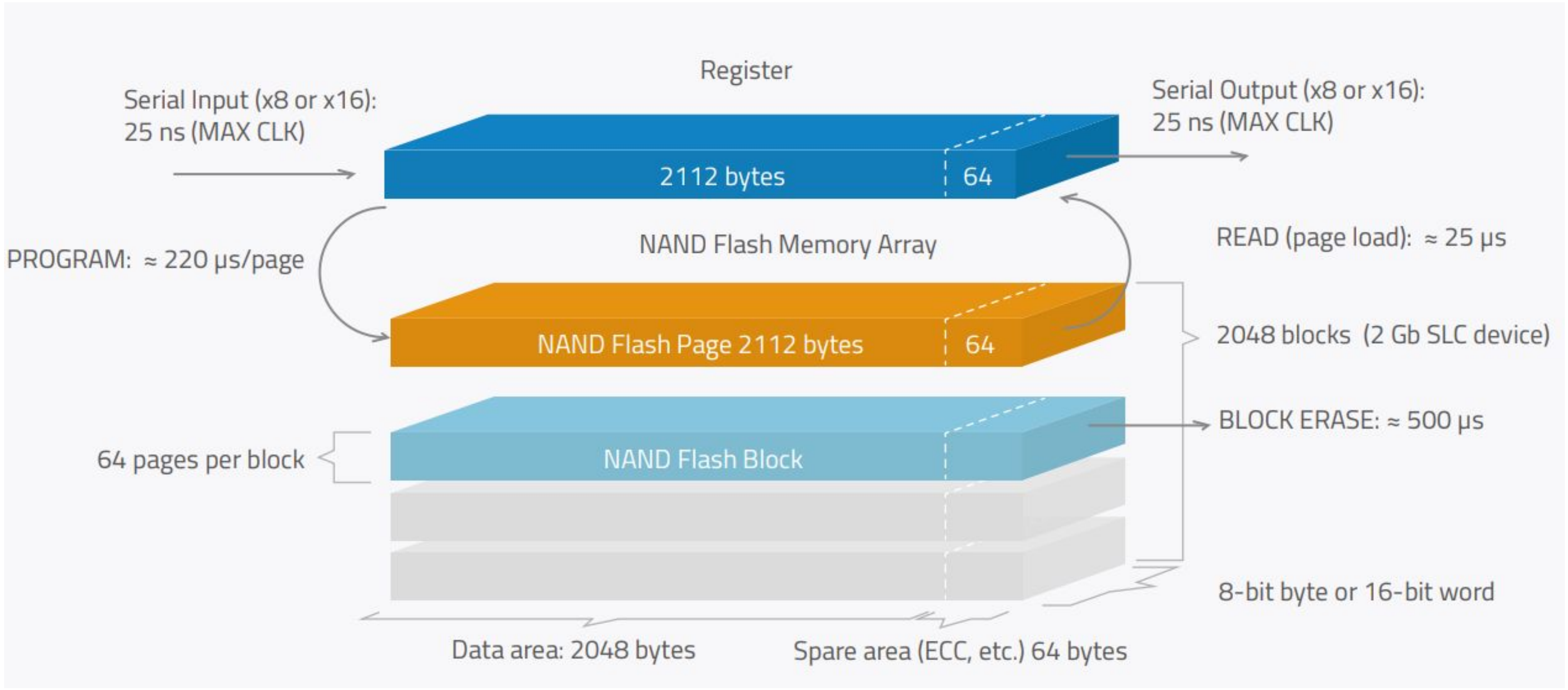
- Read a page quickly
- Write on blank pages
- But must erase entire chapters at once!



- An erase operation clears the data from all pages in the block.
- If some pages contain active data, these are first copied to a spare block. The old block is then erased and ready to be written with new data.

https://www.simms.co.uk/Uploads/Resources/50/f4366381-b992-425a-bec4-cca409b51a6c.pdf

an example of a 2 Gb flash device organized as 2048 blocks


Register
Serial Input (x8 or x16): 25 ns (MAX CLK)
Serial Output (x8 or x16): 25 ns (MAX CLK)
2112 bytes
64
PROGRAM: ≈ 220 µs/page
NAND Flash Memory Array
READ (page load): ≈ 25 µs
NAND Flash Page 2112 bytes
64
2048 blocks (2 Gb SLC device)
BLOCK ERASE: ≈ 500 µs
64 pages per block
NAND Flash Block
8-bit byte or 16-bit word
Data area: 2048 bytes
Spare area (ECC, etc.) 64 bytes

https://www.simms.co.uk/Uploads/Resources/50/f4366381-b992-425a-bec4-cca409b51a6c.pdf

The Overwrite Problem

Why You Can't "Just Write":

1. Pages start erased (all 1s)
2. Can write 0s (add electrons)
3. Can't write 1s back without erasing entire block
4. Erase is slow and wears out cells

Consequence: No in-place updates!

- Must write to new location
- Mark old data as invalid
- Eventually garbage collect

Example: Updating one 4KB sector requires:

1. Read entire block (128 pages = 512KB)
2. Modify in memory
3. Write to new block
4. Erase old block (later)

Flash Translation Layer (FTL)

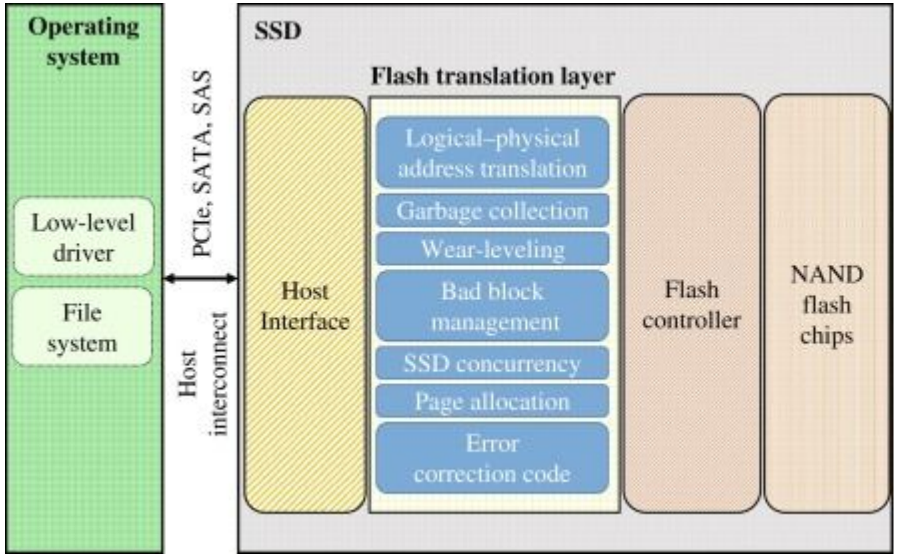
The SSD's "Secret Sauce"

FTL Responsibilities:

1. Logical-to-Physical Mapping: Like virtual memory for flash
2. Wear Leveling: Spread writes evenly across cells
3. Garbage Collection: Reclaim invalid pages
4. Bad Block Management: Replace failing cells

Mapping Granularity:

- Page-level: More flexible, more RAM needed
- Block-level: Less RAM, less flexible
- Hybrid: Common in modern SSDs



https://www.sciencedirect.com/topics/computer-science/flash-translation-layer#:~:text=A%20Flash%20Translation%20Layer%20(FTL,correction%2C%20and%20bad%20block%20management.

FTL Implementation Example

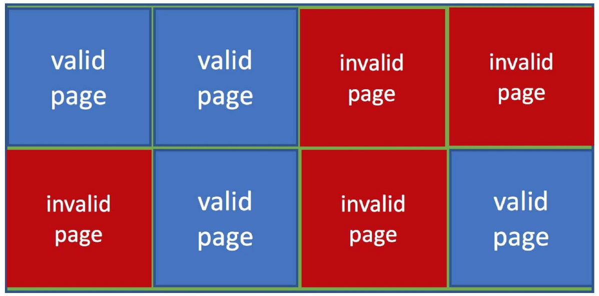
In-Memory Mapping Table:

```
Logical Page 0 → Physical Page 127

Logical Page 1 → Physical Page 42

Logical Page 2 → Physical Page 89

  ...
```

On Write (LPN 0 = new data):

1. Find free physical page (e.g., PPN 201)
2. Write data to PPN 201
3. Update mapping: LPN 0 → PPN 201
4. Mark PPN 127 as invalid

Power Loss Protection: Mapping table must be persistent!



Write Amplification Problem

Definition: Physical writes > Logical writes

Example Scenario:

- Block is 90% valid data, 10% free
- OS writes 4KB (one page)
- SSD must:
  1. Read 90% of block (115 pages)
  2. Write 115 old pages + 1 new page
  3. Erase old block
- Write Amplification Factor (WAF) = 116/1 = 116!

Real-World WAF:

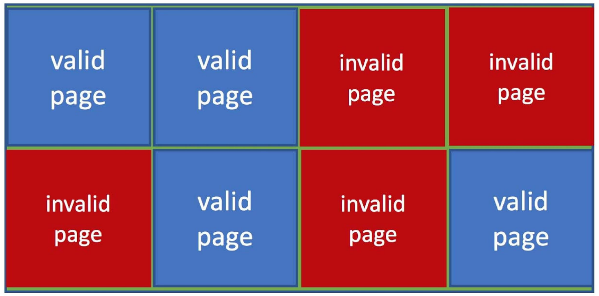
- Ideal: 1.0 (impossible with NAND)
- Good: 1.1-1.5 (sequential workloads)
- Bad: 10-50 (random writes on full drive)

Impact: Reduces performance, wears out drive faster

Garbage Collection & Over-Provisioning

Garbage Collection (GC) Process:

1. Find block with many invalid pages
2. Copy valid pages to new block
3. Erase old block
4. Add to free block pool



Over-Provisioning (OP):

- Extra physical capacity not visible to OS
- Example: 1TB drive has 1.2TB of physical flash
- OP provides "working space" for GC

Standard OP Levels:

- Consumer: 7-10% (hidden)
- Enterprise: 20-28% (configurable)
- More OP → better performance, longer life

Wear Leveling

The Problem: Some logical blocks written frequently

- Example: File system metadata, swap file
- Would wear out specific physical blocks quickly

Solution: Distribute writes evenly

Types:

1. Dynamic WL: New writes go to least-worn blocks
2. Static WL: Occasionally move cold data to worn blocks
3. Global vs. Local: Whole drive vs. regions

Endurance Metric:

- TBW (Terabytes Written): Total data guaranteed
- DWPD (Drive Writes Per Day): Daily write capacity over warranty
  - Example: 1TB SSD, 5-year warranty, 365 TBW = 0.2 DWPD

NVMe Revolution

From SATA to NVMe:

- SATA: HDD-era interface (6 Gb/s = 600 MB/s bottleneck)
- NVMe: Designed for flash, PCIe interface
  - Parallel queues (64K vs. SATA's 1 queue)
  - Lower latency (eliminates AHCI overhead)
  - More efficient interrupt handling

PCIe Generations:

- PCIe 3.0: 1 GB/s per lane (x4 = 4 GB/s)
- PCIe 4.0: 2 GB/s per lane (x4 = 8 GB/s)
- PCIe 5.0: 4 GB/s per lane (x4 = 16 GB/s) ← Current high-end

Real-World Impact: NVMe is why modern PCs feel "instant"

SSD-Optimized File Systems

Traditional FS Issues:

- Journaling causes extra writes
- In-place updates trigger write amplification
- Not aware of erase block boundaries

Modern Solutions:

1. f2fs (Flash-Friendly File System): Linux, designed for flash
2. APFS (Apple File System): Copy-on-write, space sharing
3. ReFS (Microsoft): Optimized for storage spaces
4. ZFS/Btrfs: Copy-on-write, compression, checksums

Key Techniques:

- Copy-on-write (no in-place updates)
- TRIM command (inform SSD of deleted data)
- Aligned writes (to erase block boundaries)

SSD Management in OS

TRIM Command:

- OS tells SSD which blocks are no longer needed
- Allows proactive garbage collection
- Critical for maintaining performance

Discard Mount Option (Linux):

```
mount -o discard /dev/nvme0n1p1 /mnt/ssd
```

Windows: TRIM automatic (weekly optimization)

Over-Provisioning Tools:

- Some SSDs allow adjusting OP via vendor tools
- More OP → better performance consistency

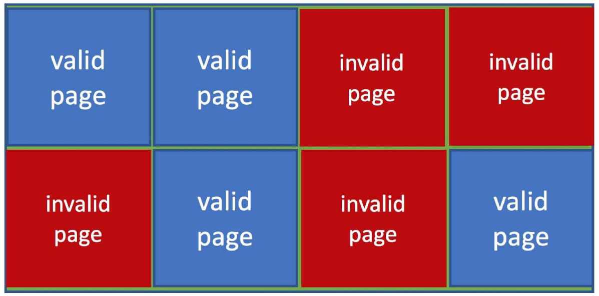
NVM Summary & Takeaways

Key Points:

1. NAND Flash = read/write at page level, erase at block level
2. FTL hides flash limitations from OS (mapping, wear leveling, GC)
3. Write Amplification = physical writes ÷ logical writes
4. SSDs excel at random I/O, HDDs better $/GB for sequential
5. NVMe is the modern interface (PCIe-based, parallel queues)

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

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

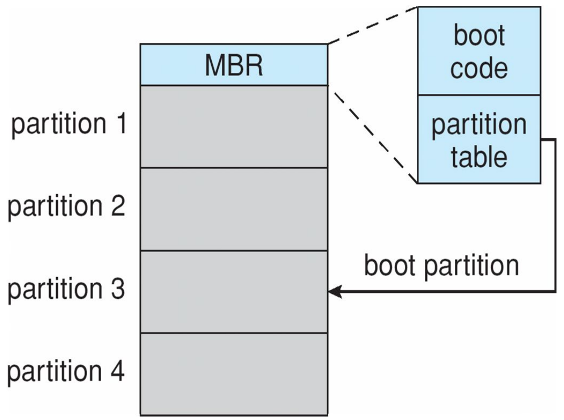
- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
    - Each sector can hold header information, plus data, plus error correction code (**ECC**)
    - Usually 512 bytes of data but can be selectable
- To use a disk to hold 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.)

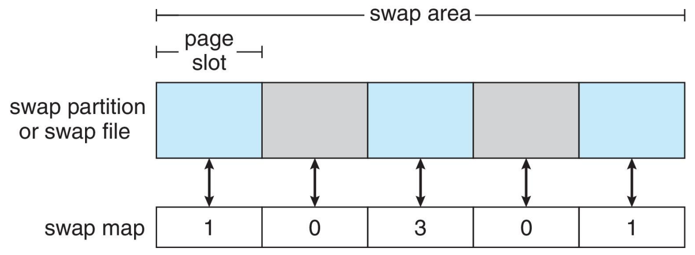
- Raw disk access for apps that want to do their own block management, keep OS out of the way (databases for example)
- Boot block initializes system
    - The bootstrap is stored in ROM, firmware
    - **Bootstrap loader** program stored in boot blocks of boot partition
- Methods such as **sector sparing**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

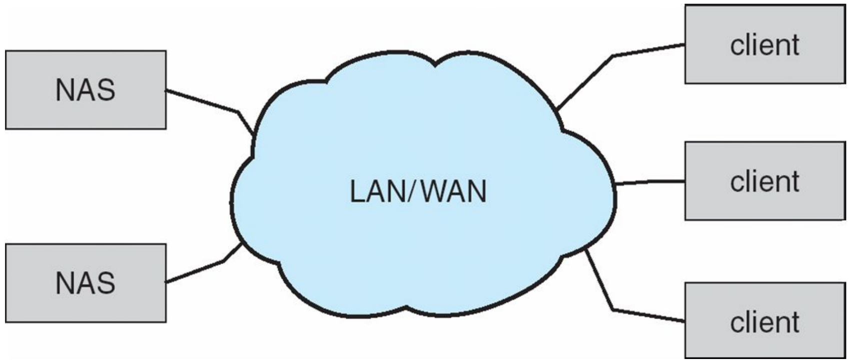


Storage Attachment

- 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
- **iSCSI** protocol uses IP network to carry the SCSI protocol
  - Remotely attaching to devices (blocks)



Cloud Storage

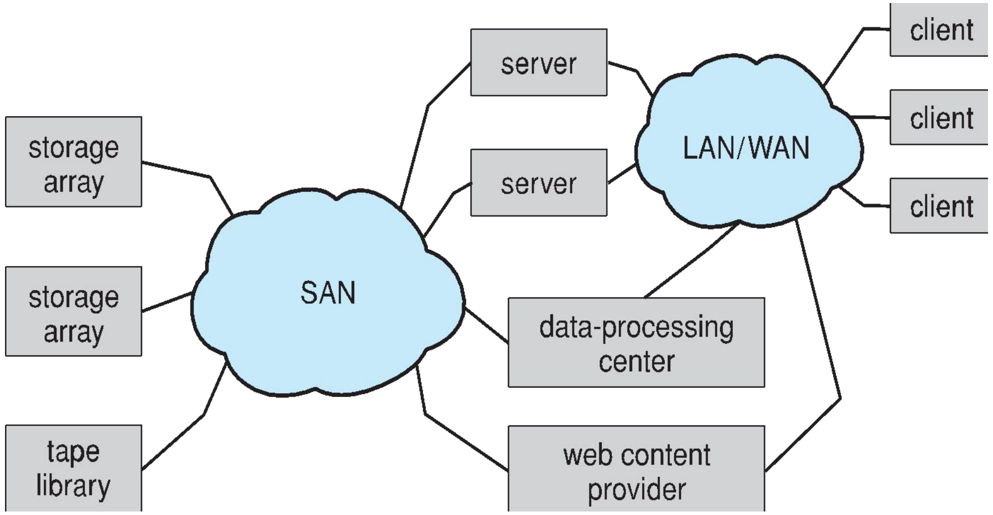
- Similar to NAS, provides access to storage across a network
  - Unlike NAS, accessed over the Internet or a WAN to remote data center
- NAS presented as just another file system, while cloud storage is API based, with programs using the APIs to provide access
  - Examples include Dropbox, Amazon S3, 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

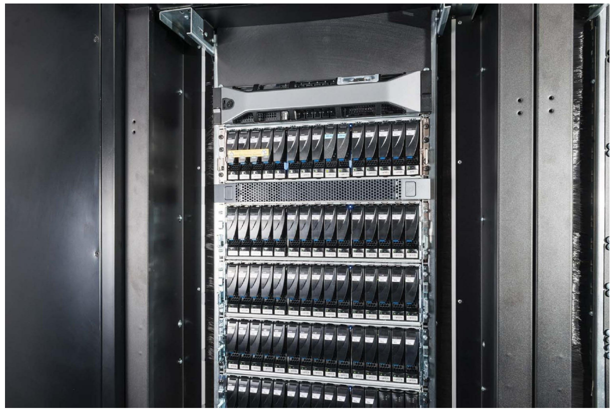
Storage Area Network

- 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**

Key Metrics

- MTTF (Mean Time To Failure): For single disk
- MTTR (Mean Time To Repair): Time to replace failed disk
- MTTDL (Mean Time To Data Loss): For RAID array

- Frequently combined with **NVRAM** to improve write performance
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively

RAID (Cont.)

Core Idea: Multiple physical disks → one logical unit

Goals:

- Reliability (tolerance to disk failures)
- Performance (parallel I/O)
- Capacity (large logical volumes)

RAID is arranged into six different levels

- Disk **striping** uses a group of disks as one storage unit
- **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 is arranged into six different levels

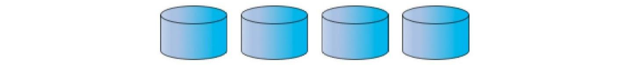
RAID 0 - Striping

RAID 1 - Mirroring

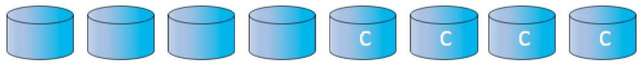
RAID 4 - Block-Level Parity

RAID 5 - Distributed Parity

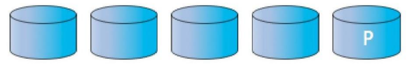
RAID 6 - Dual Parity



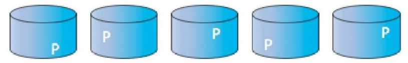
(a) RAID 0: non-redundant striping.



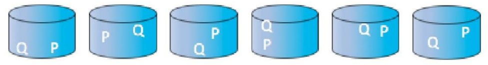
(b) RAID 1: mirrored disks.



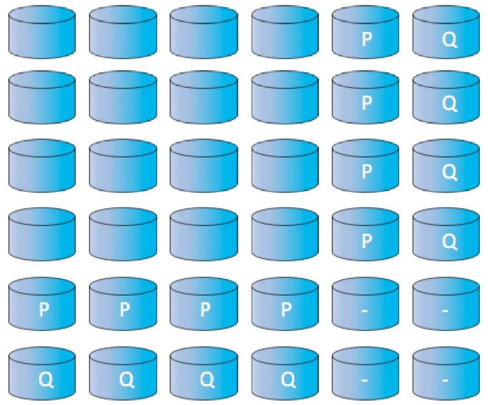
(c) RAID 4: block-interleaved parity.



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



(e) RAID 6: P + Q redundancy.



(f) Multidimensional RAID 6.

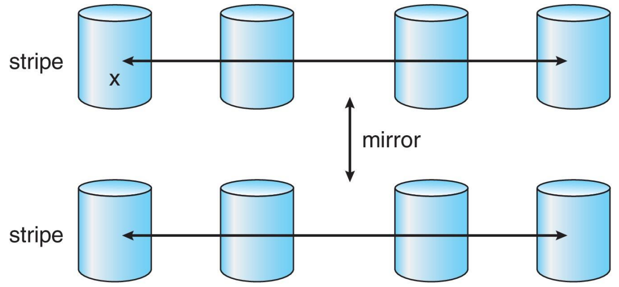
RAID 01 (0 + 1) and 10 (1 + 0)

RAID 10 (Striped Mirrors):

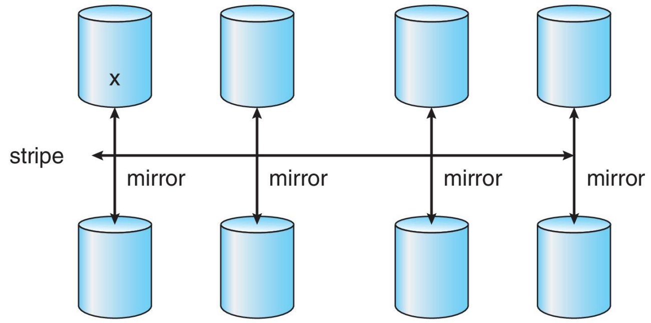
1. Create mirrors (RAID 1 pairs)
2. Stripe across mirrors

RAID 01 (Mirrored Stripes):

1. Create stripe (RAID 0)
2. Mirror the stripe



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



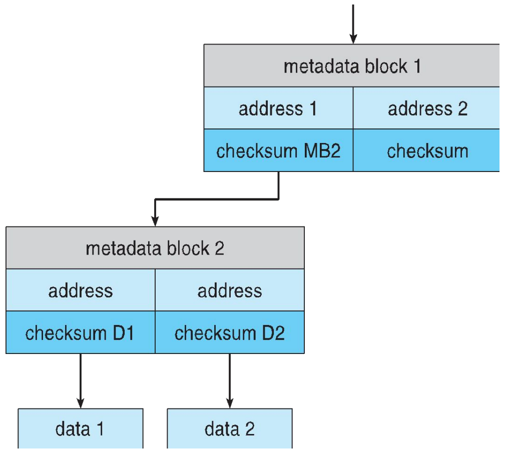
b) RAID 1 + 0 with a single disk failure.

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

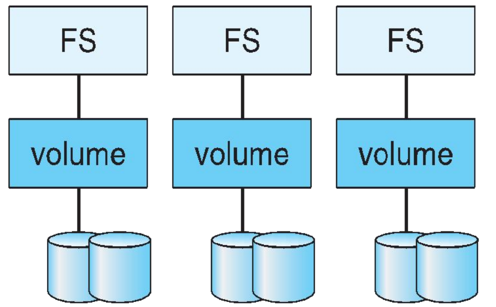
Extensions

- 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

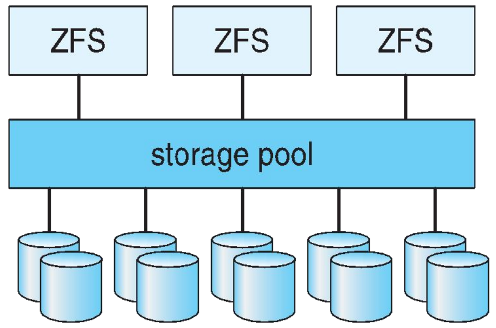


ZFS checksums all metadata and data

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