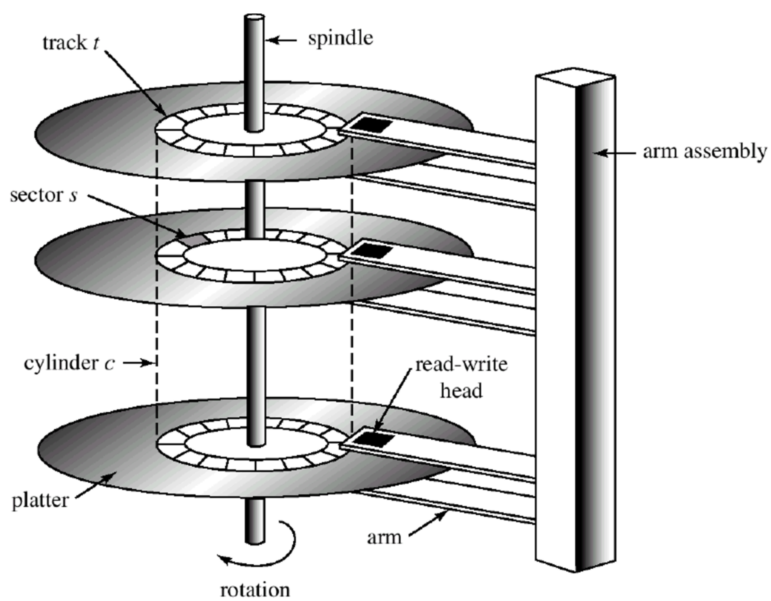
Persistence

- 36 - I/O Devices
- 37 - Hard Disk Drives
- 38 - RAID
- 39 - File and Directories
- 40 - File System Implementation
- 41 - Locality and the Fast File System
- 42 - Crash Consistency
- 43 - Log-structured File Systems
- 44 - Flash-based SSD
- 45 - Data Integrity and Protection

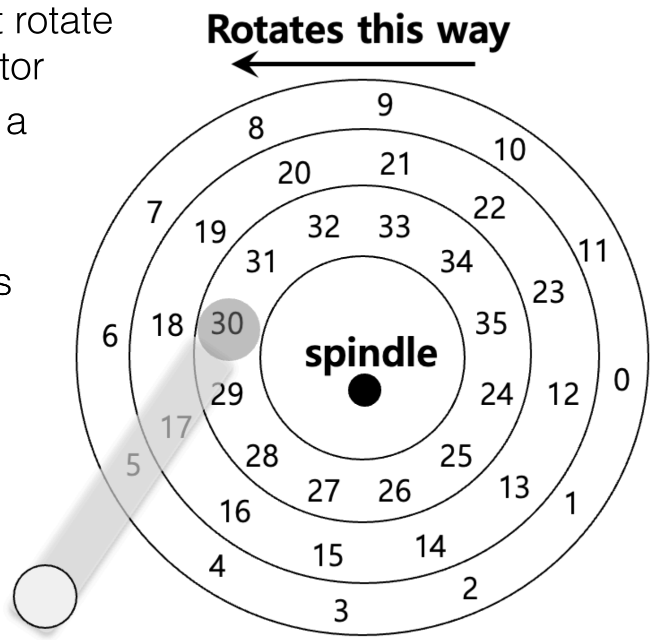
Magnetic Hard Drives

- *platter* has set of concentric tracks
- each track divided into sectors
- sectors read by read-write head



Computing the Cost

- Cost is:
  - + seek time: move to correct track
  - + rotational delay: disk must rotate until we get to correct sector
  - + transfer time: time to read a sector
- Also, disk has:
  - track cache: head always reading, remembering
  - scheduler: more later…



I/O Speeds

- I/O time defined as:
  - $T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$
- Rate of I/O:
  - $R_{I/O} = \frac{Size_{transfer}}{T_{I/O}}$
- Workload types
  - random - need a seek
  - sequential - consecutive blocks should not require seek

Example

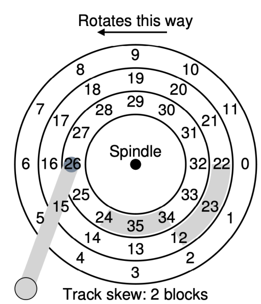
- Examples:
  - WD 6TB Red Plus, 5400 RPM, SATA 6Gb/sec, 128 MB cache (2024)

- assume 100 sectors/track*, sector 4KB, seek time 3 msec:
  - 5400 RPM $\Rightarrow \frac{1}{5400/60} = 11.1\text{msec/rot}$ ⇒ avg rot latency = 5.50 msec
  - $t_{transfer} = 11.1\text{msec}/100$ = 0.11 msec
  - seek time = 3.00 msec
  - total: = 8.61 msec
  - *Implies*: $1000/8.61 = 116\text{sectors/sec} = 116 \times 4096$ = **475 MB/sec**

- But…they claim much higher average throughput
  - constantly reading/caching everything under head
  - locality, locality, locality.
  - *sequential I/O* is a Good Thing

* modern disks have more sectors on outer tracks

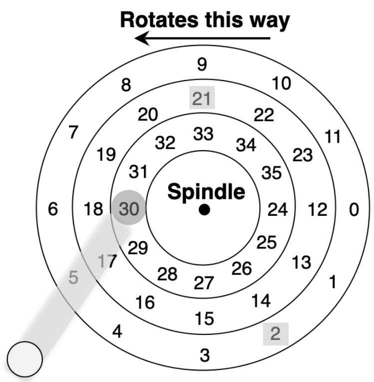
Optimizations



- track cache:
  - read head always reading
- track *skew*:
  - sectors laid out so if cross track boundaries, no extra rot delay
- When to ack back to OS/program:
  - write-back
    - ack when data in memory *dangerous! but fast!*
  - write-through
    - ack when data on disk *safe*

Disk Scheduling

- Shortest-seek-time First (SSTF)
  - order the request queue by track
  - pick requests on the nearest queue



SSTF: Scheduling Request 21 and 2

Issue the request to 21 → issue the request to 2

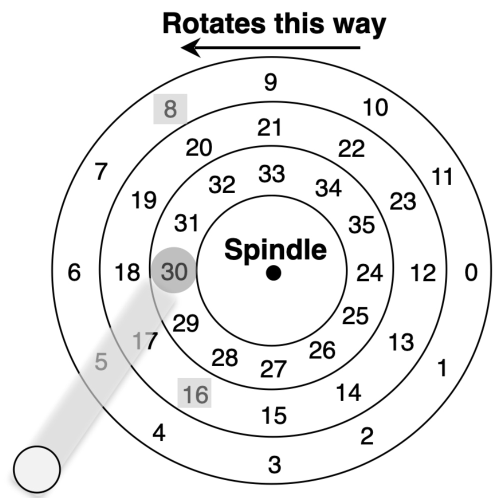
- Downsides
  - OS doesn't know drive geometry
  - *starvation…*

Elevator

- Move across the disk servicing requests in order of tracks
  - SCAN: back and forth across tracks
    - outer-to-inner, then inner-to-outer
    - If request arrives for track on current sweep, it is queued until next sweep
  - F-SCAN
    - Freeze queue while doing a sweep
    - Avoids starvation of distant requests
  - C-SCAN (circular scan)
    - Sweep from outer-to-inner, reset, then outer-to-inner, etc.

How to Account for Positioning?

- If seeks much slower than rot. lat.:
  - optimize for shorter seeks
  - request **16 is next**
  - SSTF is fine
- If seeks much faster than rot. lat.:
  - optimize for smaller rotation lat.
  - 8 **is next**
- SPTF:
  - Shortest positioning time first
  - OS does not have information
- On-disk scheduler
  - efficient SPTF
  - I/O merging



SSTF: Sometimes Not Good Enough

Sequential vs Random Example

- sequential (S) vs random (R). Assume:
  - **Sequential** : transfer 10 MB on average as continuous data.
  - **Random** : transfer 10 KB on average.
  - Average seek time: 7 ms
  - Average rotational delay: 3 ms
  - Transfer rate of disk: 50 MB/s
- Results:
  - S = $\frac{Amount\ of\ Data}{Time\ to\ access} = \frac{10\ MB}{210\ ms}$ = 47.62 MB /s
  - R = $\frac{Amount\ of\ Data}{Time\ to\ access} = \frac{10\ KB}{10.195\ ms}$ = 0.981 MB /s

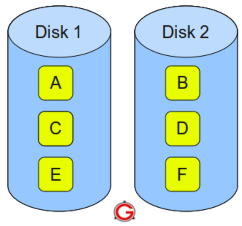
Persistence

- 36 - I/O Devices
- 37 - Hard Disk Drives
- 38 - RAID
- 39 - File and Directories
- 40 - File System Implementation
- 41 - Locality and the Fast File System
- 42 - Crash Consistency
- 43 - Log-structured File Systems
- 44 - Flash-based SSD
- 45 - Data Integrity and Protection

RAID

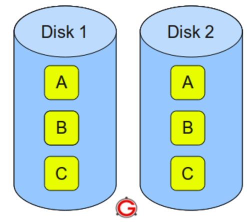
- Redundant Array of Independent Disks
- Goal: make disks faster and more more reliable
    - Disks are very cheap
    - Failures are very costly
    - Use “extra” disks to ensure reliability
        - If one disk goes down, the data still survives
    - Also allows faster access to data
- Many raid “levels”
    - Different reliability and performance properties

RAID



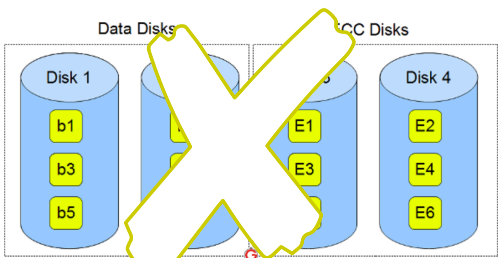
RAID 0 – Blocks Striped. No Mirror. No Parity.

Fast!



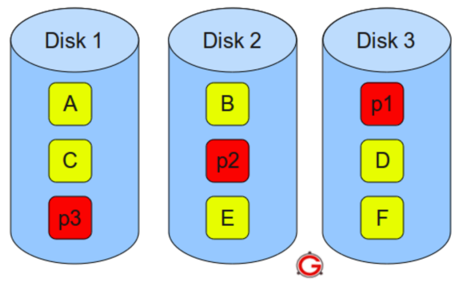
RAID 1 – Blocks Mirrored. No Stripe. No parity.

Reliable!



RAID 2 - its Striped. (and stores ECC)

Weird!

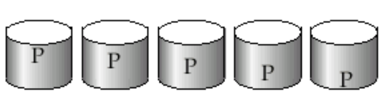


RAID 5 – Blocks Striped. Distributed Parity.

pics from thegeekstuff.com

RAID Level 5

- Distributed parity "blocks" instead of bits
- Normal operation:
  - "Read" directly from single disk.
    - Load distributed across all 5 disks
  - "Write": Need to read and update the parity block
    - To update 9 to 9'
      - read 9 and P2
      - compute P2' = P2 *xor* 9 *xor* 9'
      - write 9' and P2'

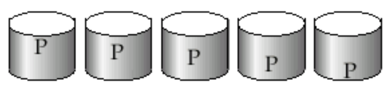


(f) RAID 5: block-interleaved distributed parity

P0	0	1	2	3
4	P1	5	6	7
8	9	P2'	10	11
12	13	14	P3	15
16	17	18	19	P4

RAID Level 5

- Failure operation (disk 3 has failed)
  - "Read block 0": Read it directly from disk 2
  - "Read block 1" (which is on disk 3)
    - Read P0, 0, 2, 3 and compute 1 = P0 *xor* 0 *xor* 2 *xor* 3
  - "Write":
    - To update 9 to 9'
      - read 9 and P2
        - Oh… P2 is on disk 3
        - So no need to read or update it
      - Write 9'



(f) RAID 5: block-interleaved distributed parity

P0	0		2	3
4	P1		6	7
8	9		10	11
12	13		P3	15
16	17		19	P4

Choosing a RAID level

- RAID 0 striping fastest, but no fault tolerance
- Main choice between RAID 1 and RAID 5
- Level 1 better write performance than level 5
  - Level 5: 2 block reads and 2 block writes to write a single block
  - Level 1: only requires 2 block writes
  - Level 1 preferred for high update environments such as log disks
- Level 5 lower storage cost
  - Usable storage for Level 1 only 50% of raw disk capacity
  - Level 5 is preferred for applications with low update rate, and large amounts of data