Outline

- Mechanisms and definitions to work with FDs
  - Armstrong axioms
  - FD closures
  - attribute closures
  - extraneous attributes
  - canonical covers
- Storage….

4. Canonical Cover

- A ***canonical cover*** for ***F*** is a set of dependencies F_c such that
  - *F* logically implies all dependencies in F_c, and
  - F_c logically implies all dependencies in *F*, and
  - No functional dependency in F_c contains an extraneous attribute, and
  - Each left side of functional dependency in F_c is unique
- In some (vague) sense, it is a ***minimal*** version of ***F***
- Create as follows:
- *repeat*
  - use union rule to merge right sides
  - eliminate extraneous attributes
- *until Fc does not change*

4. Canonical Cover

- $A \rightarrow B$, $A \rightarrow C$, $C \rightarrow D$, $AC \rightarrow BD$
- Cover:
  - $A \rightarrow BC$, $C \rightarrow D$, $AC \rightarrow BD$ (union)
    - a extra in $AC \rightarrow BD$?
      - NO: $C^+ = CD$, doesn't include "BD"
    - c extra in $AC \rightarrow BD$?
      - YES: $A^+ = ABCD$, includes "BD"
  - $A \rightarrow BC$, $C \rightarrow D$, $A \rightarrow BD$
  - $A \rightarrow BCD$, $C \rightarrow D$ (union)
    - B extra in $A \rightarrow BCD$?
      - $F' = A \rightarrow CD$, $C \rightarrow D$
      - NO: $A^+ = CD$ in F', not include "B"
    - C extra in $A \rightarrow BCD$?
      - $F' = A \rightarrow BD$, $C \rightarrow D$
      - NO: $A^+ = BD$ in F', not include "C"
    - D extra in $A \rightarrow BCD$?
      - $F' = A \rightarrow BC$, $C \rightarrow D$
      - YES: $A^+ = bcd$ in F', includes "D"
    - $F_c = A \rightarrow BC$, $C \rightarrow D$

repeat
- use union rule to merge right sides
- eliminate extraneous attributes

until F_c does not change

σ is *extraneous* in α iff:
$F \rightarrow F'$, *or*
$(\alpha - \sigma)^+$ includes β under *F*

σ is *extraneous* in β iff:
$F' \rightarrow F$, *or*
α^+ includes σ in *F'*

Recap

- What about 1st and 2nd normal forms ?
- 1NF:
  - Essentially says that no set-valued attributes allowed
  - Formally, a domain is called ***atomic*** if the elements of the domain are considered indivisible
  - A schema is in 1NF if the domains of all attributes are atomic
  - We assumed 1NF throughout the discussion
    - Non 1NF is just not a good idea
- 2NF:
  - Mainly historic interest
  - See Exercise 7.15 in the book

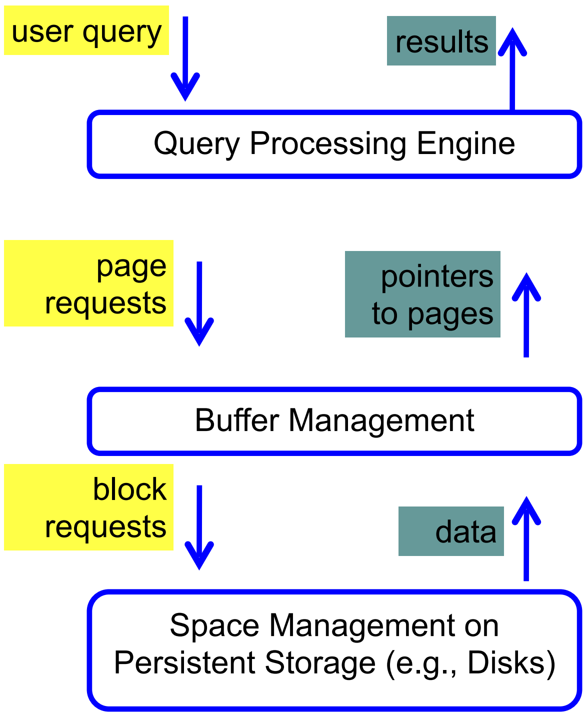
Recap

- We would like our relation schemas to:
  - *Not allow potential redundancy* because of FDs
  - Be ***dependency-preserving:***
    - Make it easy to check for dependencies
    - Since they are a form of integrity constraints
- Functional Dependencies
  - Domain knowledge about the data properties
- Normal forms
  - Defines the rules that schemas must follow
  - Informs deconstruction

Databases

- Data Models
  - Conceptual representation of the data
- Data Retrieval
  - How to ask questions of the database
  - How to answer those questions
- Data Storage
  - How/where to store data, how to access it
- Data Integrity
  - Manage crashes, concurrency
  - Manage semantic inconsistencies

Query Processing/Storage



- Given a input user query, decide how to “execute” it
- Specify sequence of pages to be brought in memory
- Operate upon the tuples to produce results

- Bringing pages from disk to memory
- Managing the limited memory

- Storage hierarchy
- How are relations mapped to files?
- How are tuples mapped to disk blocks?

Outline

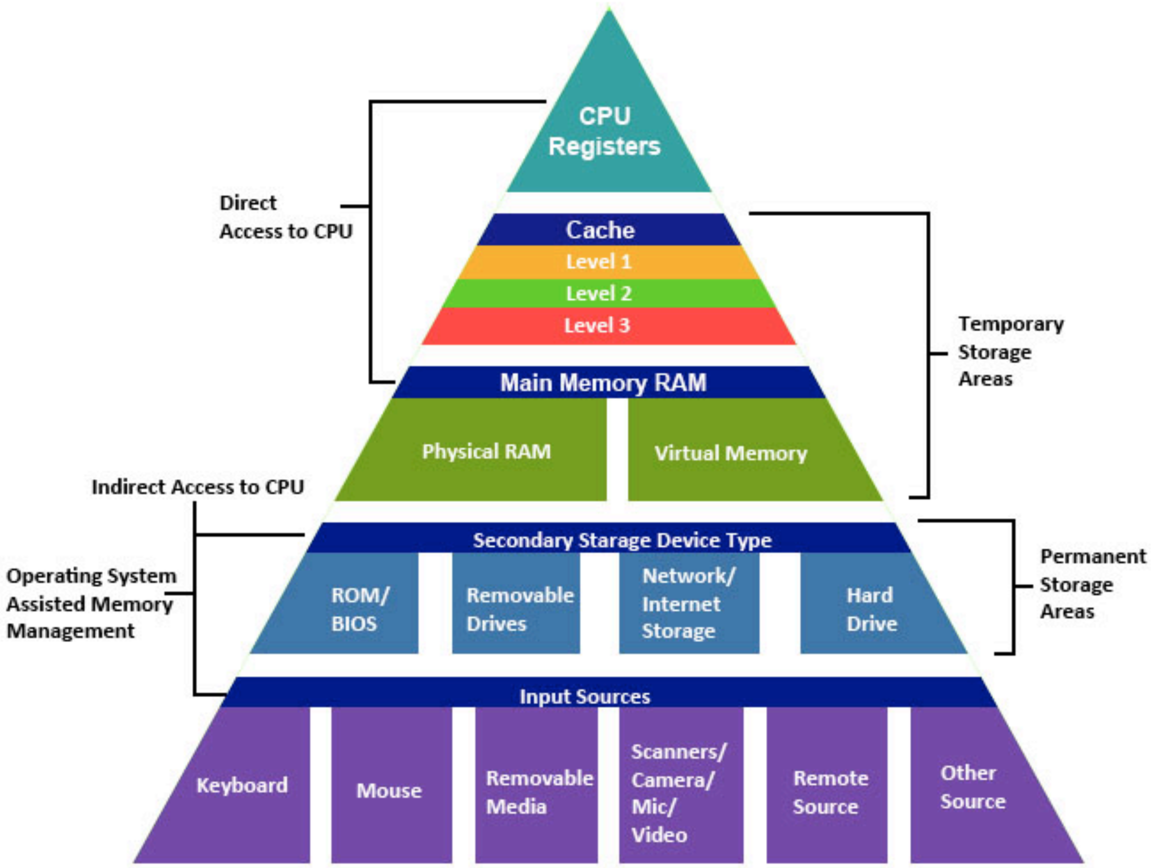
- Storage hierarchy
- Disks
- RAID
- File Organization
- Etc….

Storage Hierarchy

- Tradeoffs between speed and cost of access
- Volatile vs nonvolatile
  - Volatile: Loses contents when power switched off
- Sequential vs random access
  - Sequential: read the data contiguously
    - select * from employee
  - Random: read the data from anywhere at any time
    - select * from employee where name like '__a__b'
- Why care ?
  - Need to know how data is stored in order to optimize, to understand what's going on

How important is this today?

- Trade-offs shifted drastically over last 10-15 years
  - Especially with fast network, SSDs, and high memories
  - However, the volume of data is also growing quite rapidly
- Some observations:
  - Cheaper to access another computer's memory than local disk
  - Cache is playing more and more important role
  - Data often fits in memory of a single machine, or cluster of machines
  - "Disk" considerations less important
    - Still: Disks are where most of the data lives today
  - Similar reasoning/algorithms required though

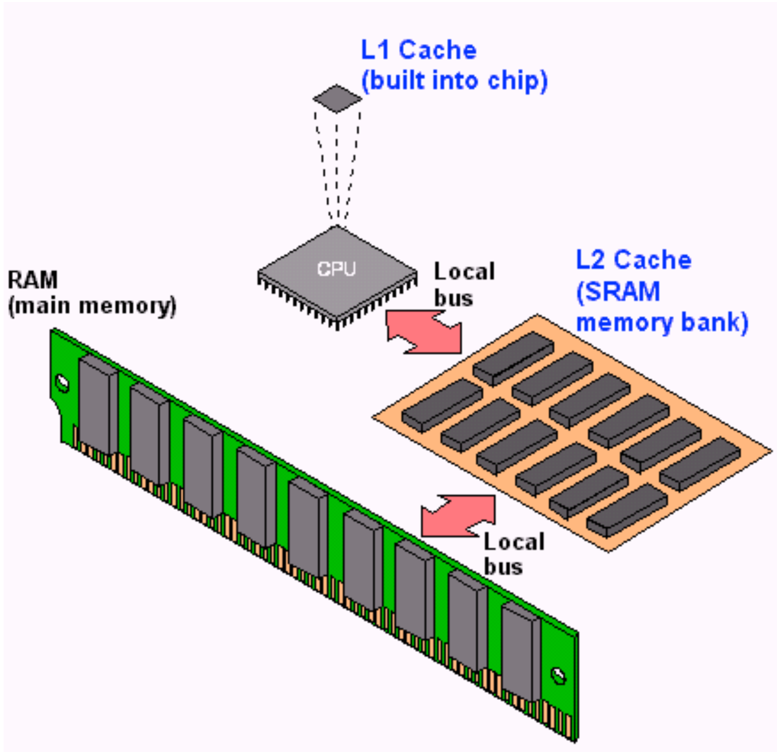


source: http://cse1.net/recaps/4-memory.html

Storage Hierarchy: Cache

- Cache
  - Super fast; volatile; Typically on chip
  - L1 vs L2 vs L3 caches ???
    - L1 about 64KB or so; L2 about 1MB; L3 8MB (on chip) to 256MB (off chip)
    - Huge L3 caches available now-a-days
  - Becoming more and more important to care about this
    - Cache misses are expensive
  - Similar tradeoffs as were seen between main memory and disks
  - Cache-coherency ??

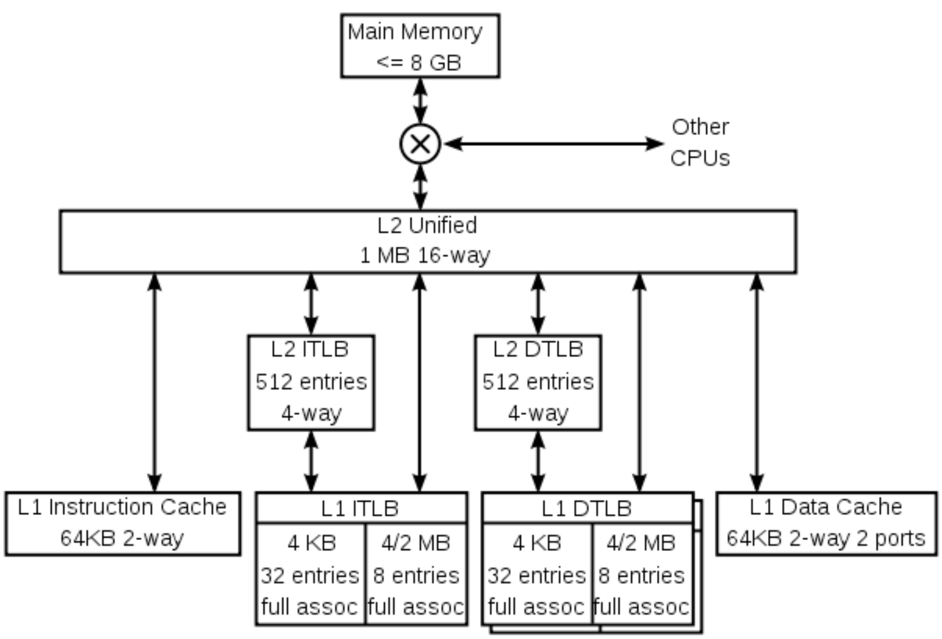
Storage Hierarchy: Cache



source: http://cse1.net/recaps/4-memory.html

Storage Hierarchy: Cache

K8 core in the AMD Athlon 64 CPU



Storage Hierarchy

- Main memory
  - 10s or 100s of ns; volatile
  - Pretty cheap and dropping: 1GByte << $100
  - Main memory databases feasible now-a-days
- Flash memory (EEPROM)
  - Limited number of write/erase cycles
  - Non-volatile, slower than main memory (especially writes)
  - Examples ?
- *Question*
  - *How does what we discuss next change if we use flash memory only ?*
  - *Key issue: Random access as cheap as sequential access*

Storage Hierarchy

- Magnetic Disk (HDD or just "Hard Drive")
  - Non-volatile
  - Sequential access much much faster than random access
  - Discuss in more detail later
- Optical Storage - CDs/DVDs; Jukeboxes
  - Used more as backups… Why ?
  - Very slow to write (if possible at all)
- Tape storage
  - Backups; super-cheap; painful to access
  - IBM just released a secure tape drive storage solution

Storage…

- Primary
  - e.g. Main memory, cache; typically volatile, fast
- Secondary
  - e.g. Disks; Solid State Drives (SSD); non-volatile
- Tertiary
  - e.g. Tapes; Non-volatile, super cheap, slow

Storage Hierarchy

Storage type	Access time	Relative access time
L1 cache	0.5 ns	Blink of an eye
L2 cache	7 ns	4 seconds
1MB from RAM	0.25 ms	5 days
1MB from SSD	1 ms	23 days
HDD seek	10 ms	231 days
1MB from HDD	20 ms	1.25 years

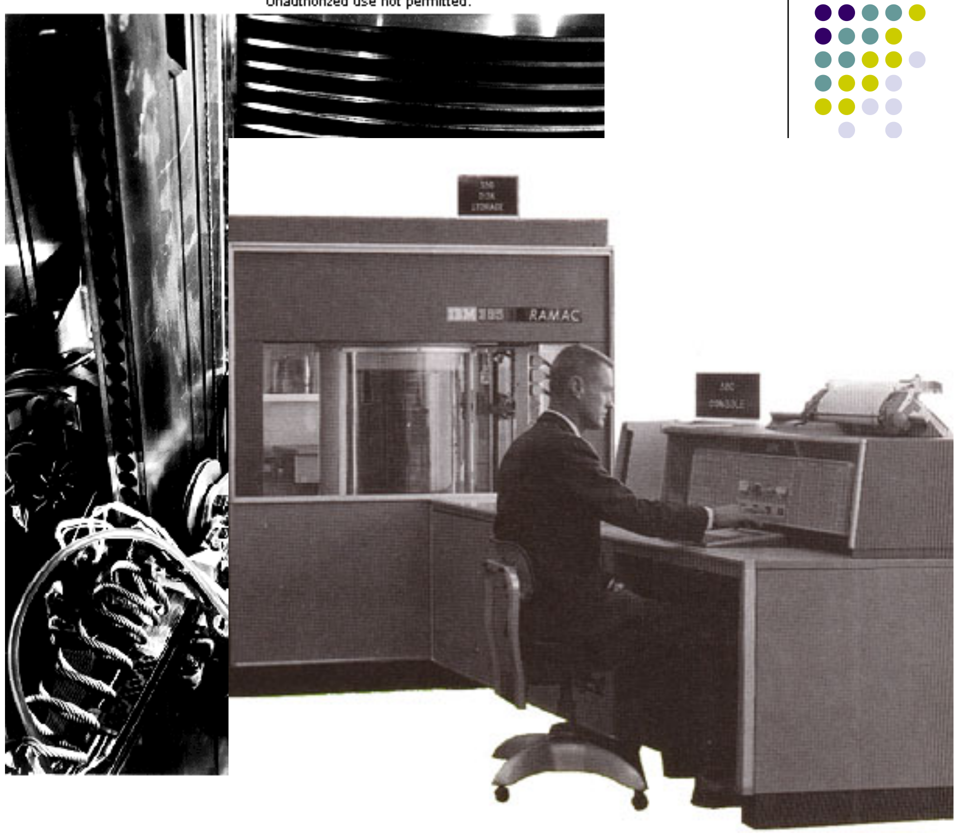
source: http://cse1.net/recaps/4-memory.html

Outline

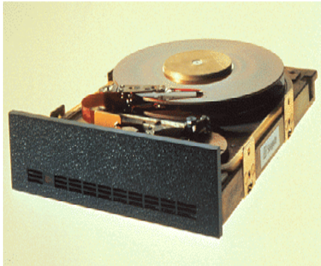
- Storage hierarchy
- Disks
- RAID
- File Organization
- Etc….

1956
IBM RAMAC
24” platters
100,000 characters each
5 million characters

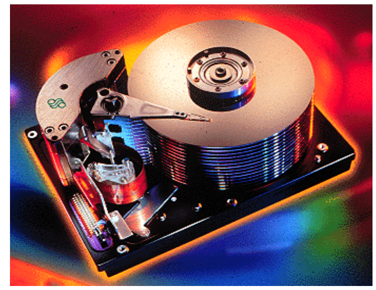
From Computer Desktop Encyclopedia
Reproduced with permission.



1979
SEAGATE
5MB



1998
SEAGATE
47GB



2006
Western Digital
500GB
Weight (max. g): 600g

2000-ish:

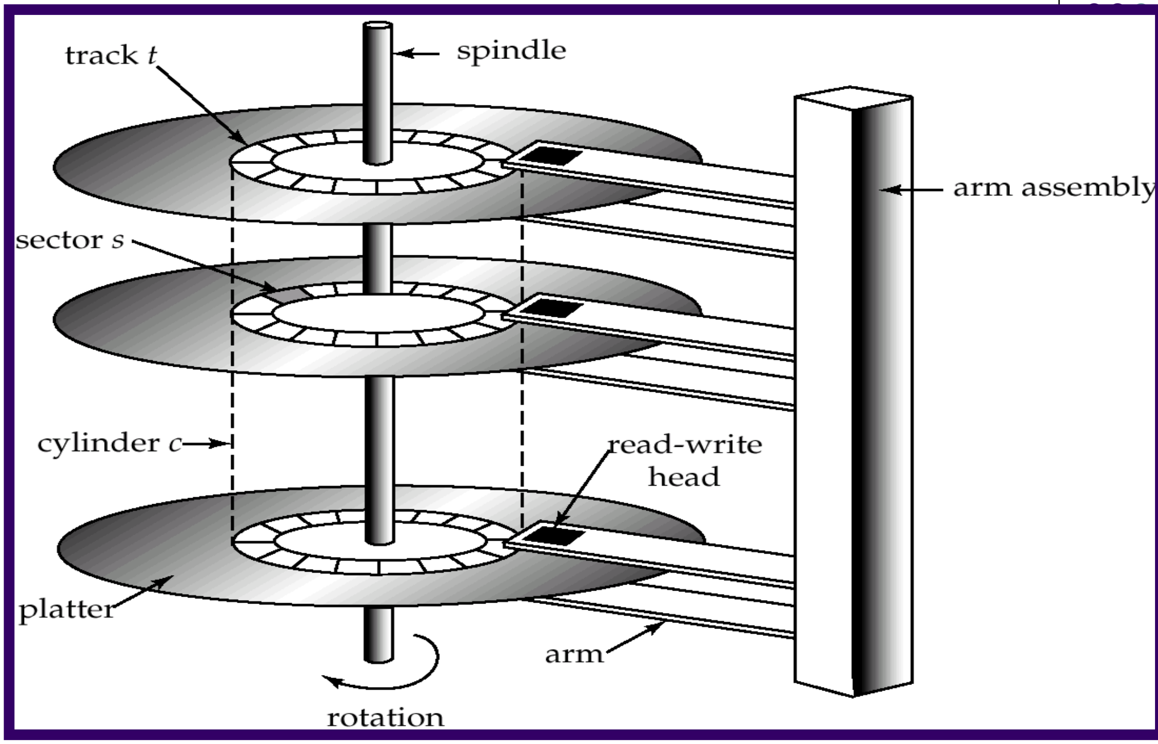
Single hard drive:
  Seagate Barracuda 7200.10 SATA
  750 GB
  7200 rpm
  weight: 720g
  Uses “perpendicular recording”

Microdrives

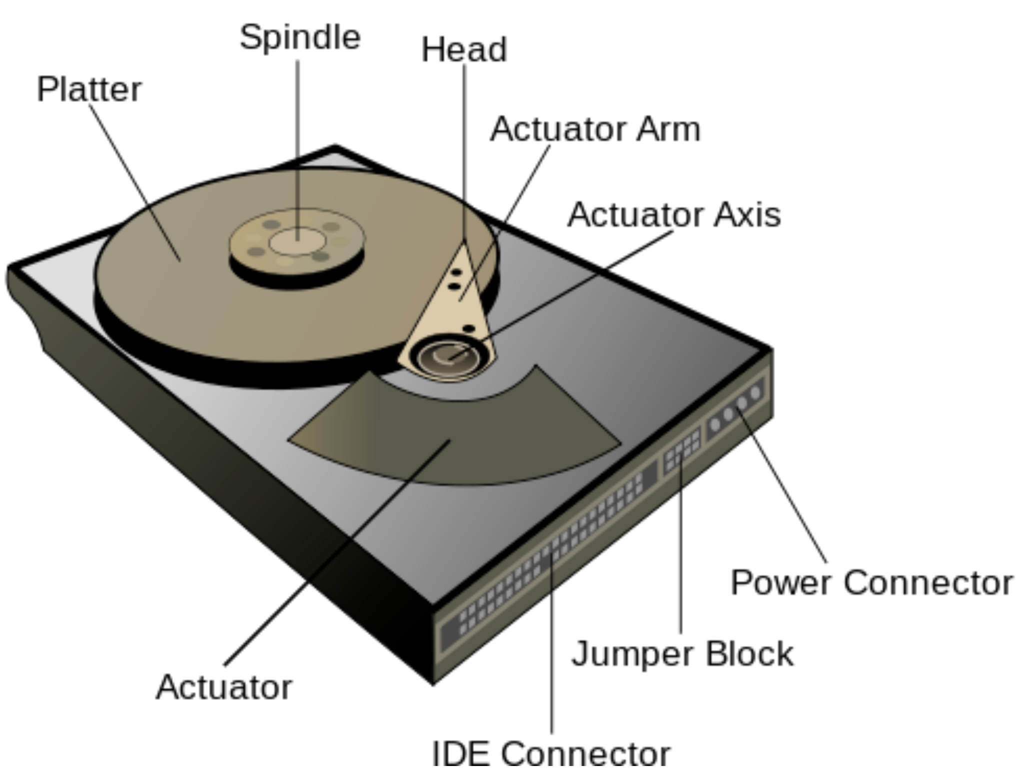
Now:
- 4 TB HDD $99
- 4 TB SSD $289

Toshiba 80GB

IBM 1 GB



track t
spindle
arm assembly
sector s
cylinder c
read-write head
platter
arm
rotation



Spindle
Head
Platter
Actuator Arm
Actuator Axis
Power Connector
Jumper Block
Actuator
IDE Connector

"Typical" Values

Diameter: 1 inch → 15 inches

Cylinders: 100 → 2000

Surfaces: 1 or 2

(Tracks/cyl) 2 (floppies) → 30

Sector Size: 512B → 50K

Capacity → 360 KB to 2TB (as of Feb 2010)

Rotations per minute (rpm) → 5400 to 15000

Accessing Data

- Accessing a sector
  - Time to seek to the track (seek time)
    - average 4 to 10ms
  - \+ Waiting for the sector to get under the head (rotational latency)
    - average 4 to 11ms
  - \+ Time to transfer the data (transfer time)
    - very low
  - About 10ms per access
    - So if randomly accessed blocks, can only do 100 block transfers
    - 100 x 512bytes = 50 KB/s
- Data transfer rates
  - Rate at which data can be transferred (w/o any seeks)
  - 30-50MB/s to up to 200MB/s (Compare to above)
    - Seeks are bad !

Seagate Barracuda: 1TB

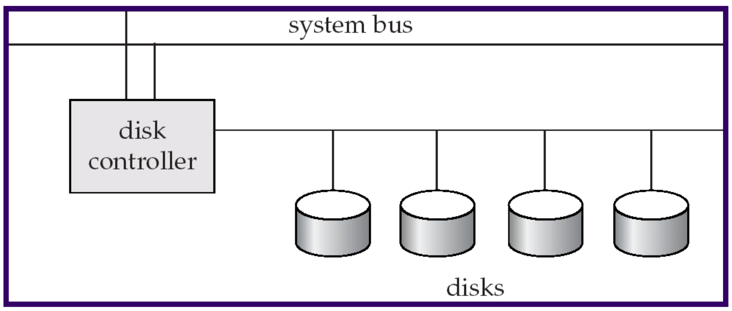
- Heads 8, Disks 4
- Bytes per sector: 512 bytes
- Default cylinders: 16,383
- Defaults sectors per track: 63
- Defaults read/write heads: 16
- Spindle speed: 7200 rpm
- Internal data transfer rate: 1287 Mbits/sec max
- Average latency: 4.16msec
- Track-to-track seek time: 1msec-1.2msec
- Average seek: 8.5-9.5msec
- We also care a lot about power now-a-days
  - Why ?

Reliability

- Mean time to/between failure (MTTF/MTBF):
  - 57 to 136 years
- Consider:
  - 1000 new disks
  - 1,200,000 hours (136 years) of MTTF each
  - On average, one will fail in1200 hours = 50 days !

Disk Controller

- Interface between the disk and the CPU
- Accepts the commands
- checksums to verify correctness
- Remaps bad sectors



Optimizing block accesses

- Typically sectors too small
- Block: A contiguous sequence of sectors
  - 4k to 16k
  - All data transfers done in units of blocks
- Scheduling of block access requests ?
  - Considerations: performance and fairness
  - Elevator algorithm

Solid State Drives

- Essentially flash that emulates hard disk interfaces
- No seeks → Much better random reads performance
- Writes are slower, the number of writes at the same location limited
  - Must write an entire block at a time
- About a factor of ~~10~~ …3 more expensive right now
- Leading to radical hardware configuration change

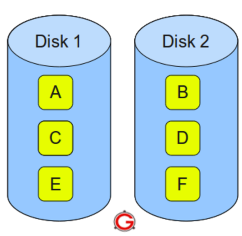
Outline

- Storage hierarchy
- Disks
- RAID
- File Organization
- Etc….

RAID

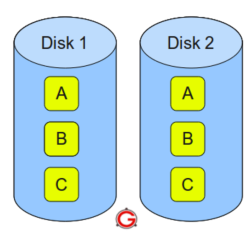
- Redundant array of independent disks
- Goal:
  - Disks are very cheap
  - Failures are very costly
  - Use “extra” disks to ensure reliability
    - If one disk goes down, the data still survives
  - Also allows faster access to data
- Many raid “levels”
  - Different reliability and performance properties

Redundant Array Independent Disks



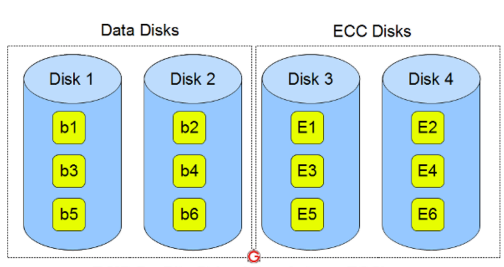
RAID 0 – Blocks Striped. No Mirror. No Parity.

Fast!



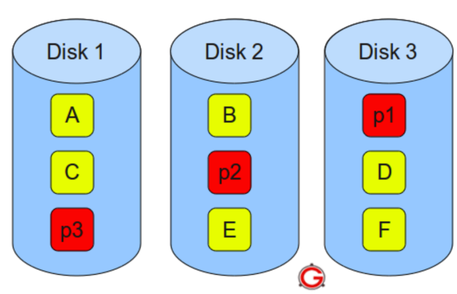
RAID 1 – Blocks Mirrored. No Stripe. No parity.

Redundant!



RAID 2 – Bits Striped. (and stores ECC)

Weird!

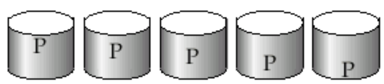


RAID 5 – Blocks Striped. Distributed Parity.

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'

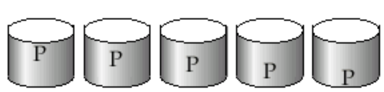


(f) RAID 5: block-interleaved distributed parity

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 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
- SSD?
  - performance already good, just care about fault tolerance