CMSC424: Database Design
Storage and Indexes

March 11, 2020

Instructor: Amol Deshpande
amol@cs.umd.edu

Announcements Etc.

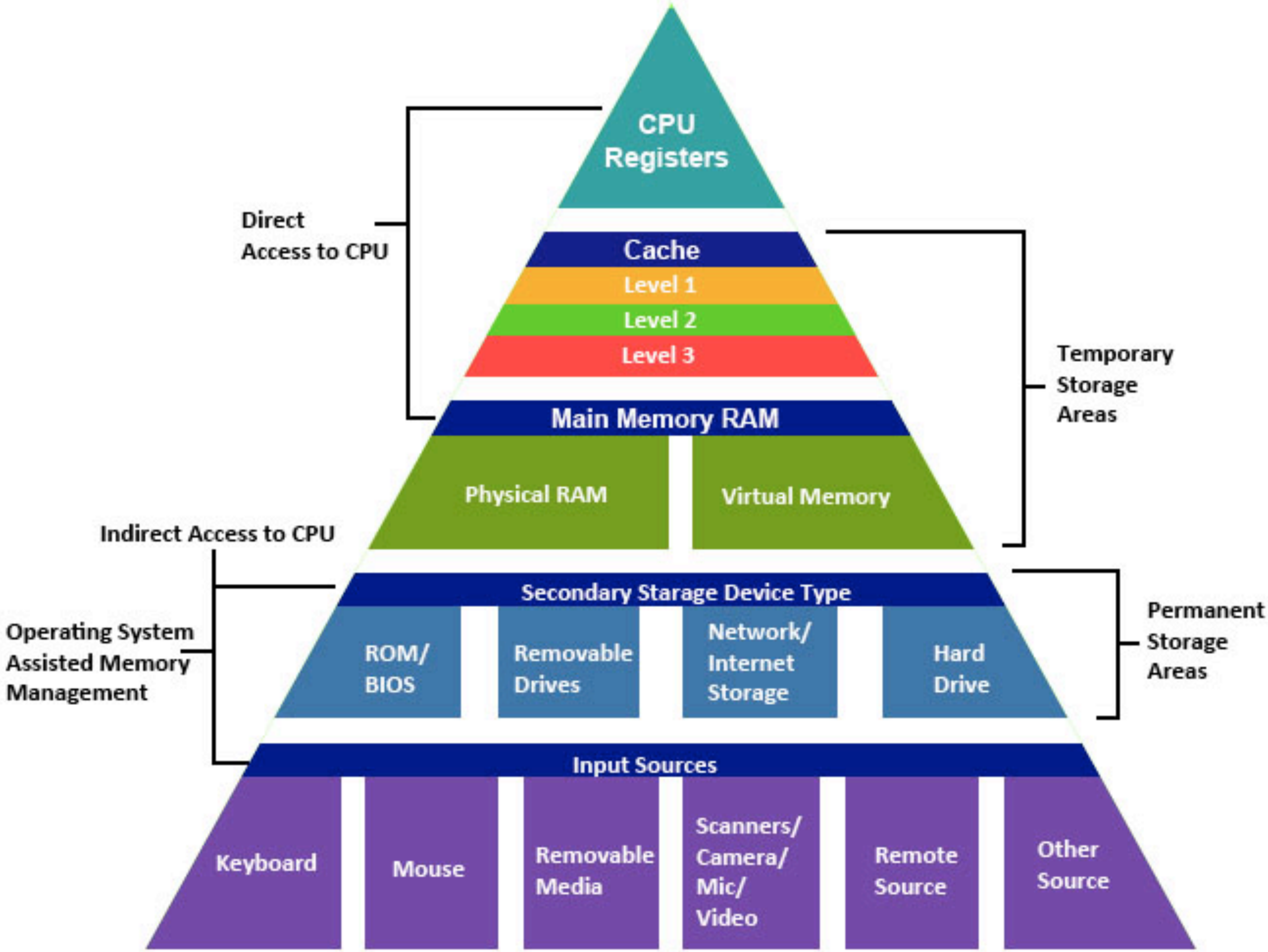
- COVID-19 Stuff
  - Postponed Project 3 due date to March 23
  - No Midterm on April 8 as of now – will look into replacing with timed ELMS quizzes or something analogous
  - Grading of assignments/midterm is also impacted

- Will keep communicating as we get more information and figure out a plan for virtual instruction
  - Keep an eye on CampusWire and ELMS announcements

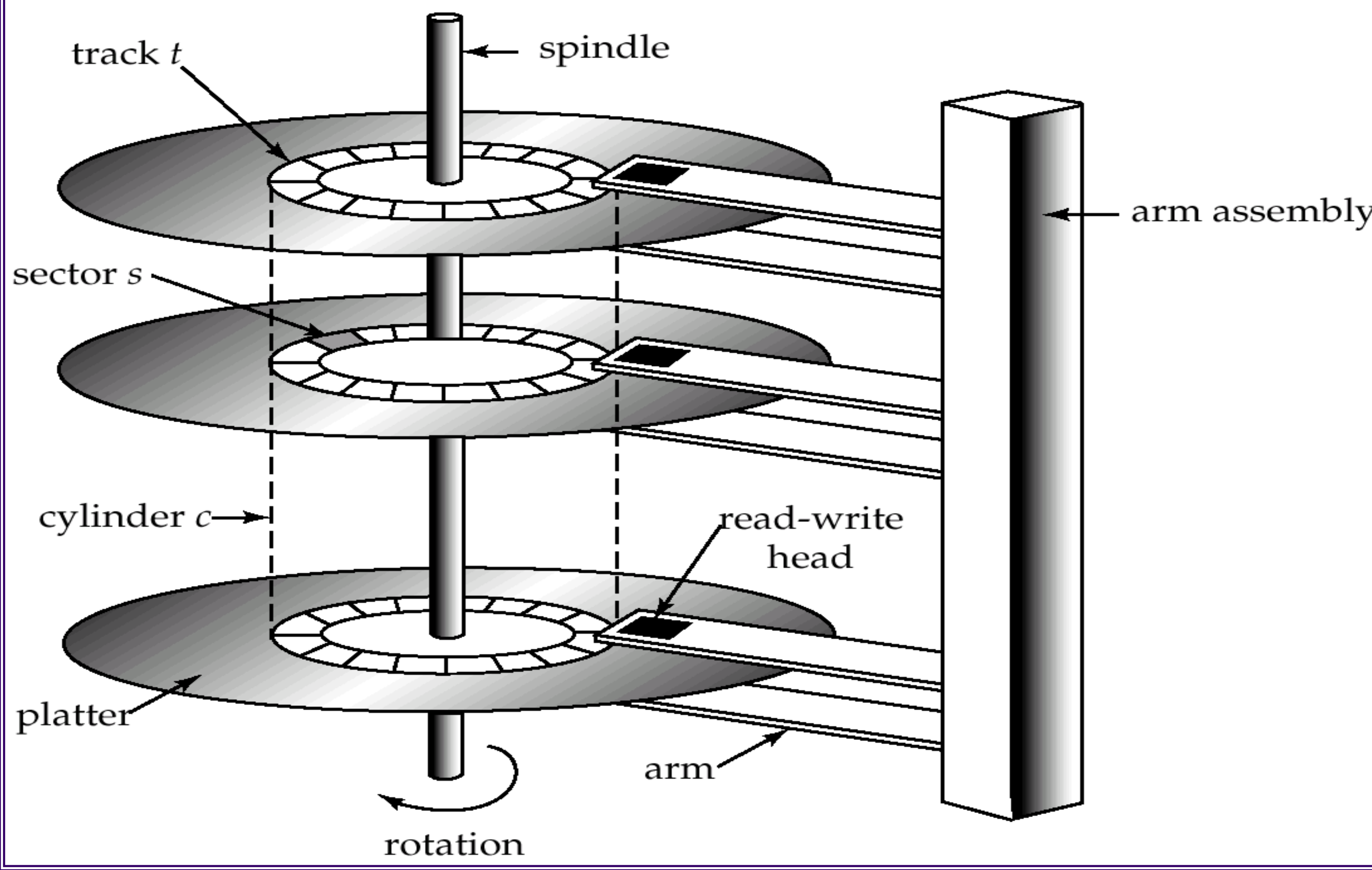
- Will record and upload this week's lectures

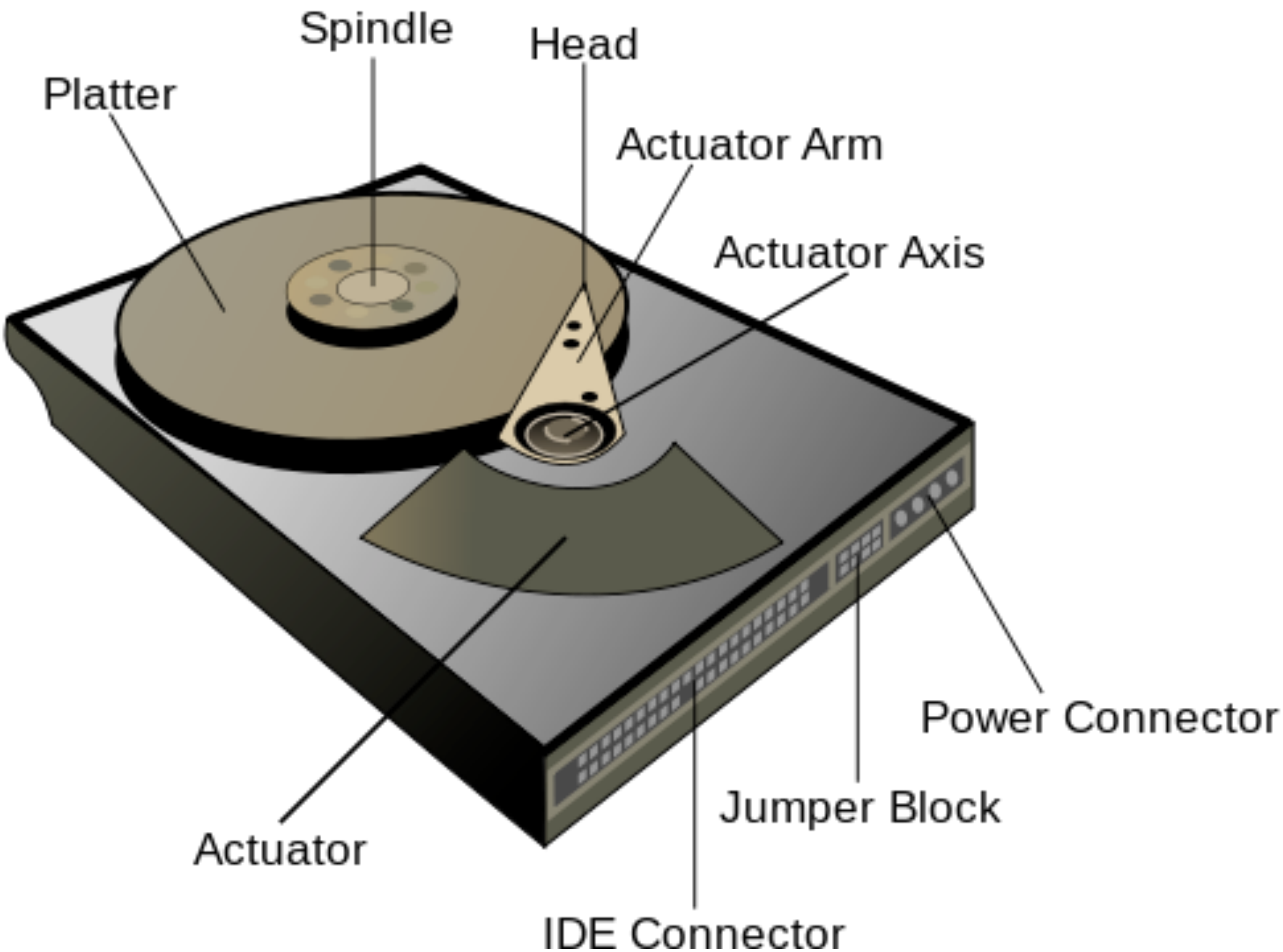
Plan for Today

- Review:
  - Storage Hierarchy
  - Disks
- Solid State Drives
- RAID: Redundant Array of Independent Disks
  - Although it says “disks”, the ideas are more general
- Buffer Manager
- File Organization on Storage



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


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

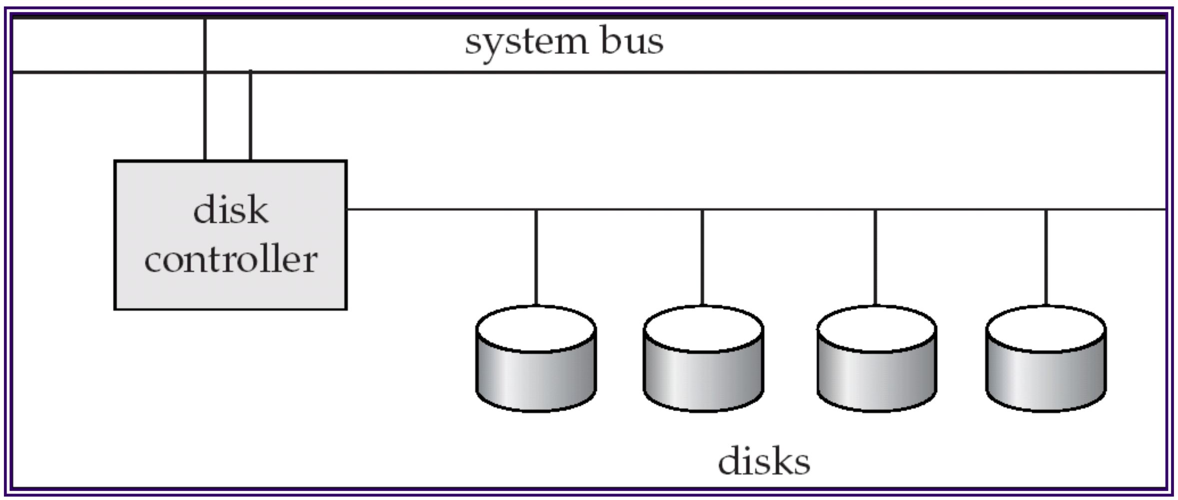
Diameter:	1 inch → 15 inches
Cylinders:	100 → 2000
Surfaces:	1 or 2
(Tracks/cyl)	2 (floppies) → 30
Sector Size:	512B → 50K
Capacity →	360 KB to 2TB (as of Feb 2010)
Rotations per minute (rpm) →	5400 to 15000

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 !

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
  - 512 bytes to several Kbytes
  - 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 more expensive right now

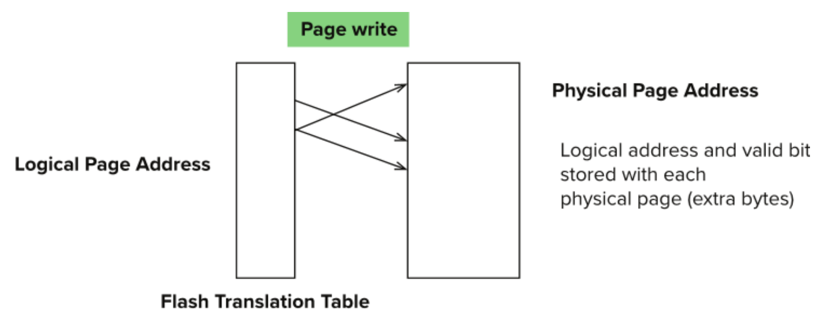
- Has led to perhaps the most radical hardware configuration change in a while

Flash Storage

- NOR flash vs NAND flash
- NAND flash
  - used widely for storage, cheaper than NOR flash
  - requires page-at-a-time read (page: 512 bytes to 4 KB)
    - 20 to 100 microseconds for a page read
    - Not much difference between sequential and random read
  - Page can only be written once
    - Must be erased to allow rewrite
- **Solid state disks**
  - Use standard block-oriented disk interfaces, but store data on multiple flash storage devices internally
  - Transfer rate of up to 500 MB/sec using SATA, and up to 3 GB/sec using NVMe PCIe

Flash Storage (Cont.)

- Erase happens in units of **erase block**
  - Takes 2 to 5 millisecs
  - Erase block typically 256 KB to 1 MB (128 to 256 pages)
- **Remapping** of logical page addresses to physical page addresses avoids waiting for erase
- **Flash translation table** tracks mapping
  - also stored in a label field of flash page
  - remapping carried out by **flash translation layer**



- After 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
  - **wear leveling**

SSD Performance Metrics

- Random reads/writes per second
  - Typical 4 KB reads: 10,000 reads per second (10,000 IOPS)
  - Typical 4KB writes: 40,000 IOPS
  - SSDs support parallel reads
    - Typical 4KB reads:
      - 100,000 IOPS with 32 requests in parallel (QD-32) on SATA
      - 350,000 IOPS with QD-32 on NVMe PCIe
    - Typical 4KB writes:
      - 100,000 IOPS with QD-32, even higher on some models
- Data transfer rate for sequential reads/writes
  - 400 MB/sec for SATA3, 2 to 3 GB/sec using NVMe PCIe

Break…

- Why do we care about these things?
- Specific hardware architecture being used has a significant impact on performance
- Need to reason about the details in order to "tune" the system
  - Has gotten a little too complex
  - Often hard to abstract

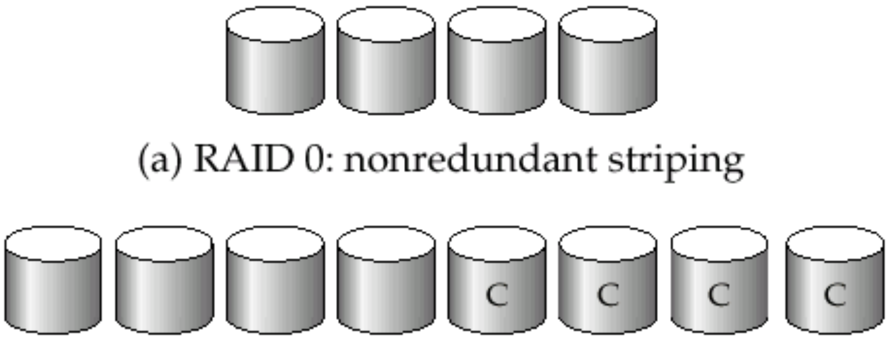
Plan for Today

- Review:
  - Storage Hierarchy
  - Disks
- Solid State Drives
- RAID: Redundant Array of Independent Disks
  - Although it says "disks", the ideas are more general
- Buffer Manager
- File Organization on Storage

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

RAID Levels



(a) RAID 0: nonredundant striping

(b) RAID 1: mirrored disks

(a) No redundancy.

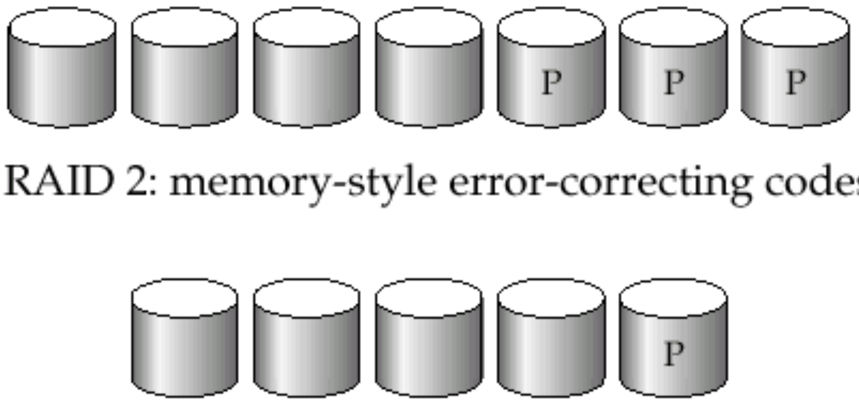
(b) Make a copy of the disks.

If one disk goes down, we have a copy.

Reads: Can go to either disk, so higher data rate possible.

Writes: Need to write to both disks.

RAID Levels



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

(d) RAID 3: bit-interleaved parity

(c) Memory-style Error Correcting

Keep extra bits around so we can reconstruct.

Superceeded by below.

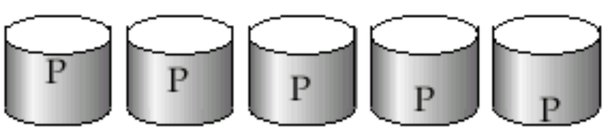
(d) One disk contains "parity" for the main data disks.

Can handle a single disk failure.

Little overhead (only 25% in the above case).

RAID Level 5

- Distributed parity "blocks" instead of bits
- Subsumes Level 4
- Normal operation:
  - "Read" directly from the disk. Uses 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'

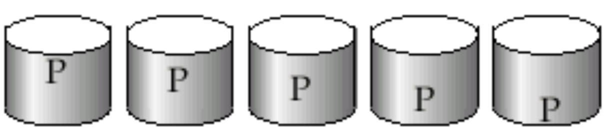


(f) RAID 5: block-interleaved distributed parity

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

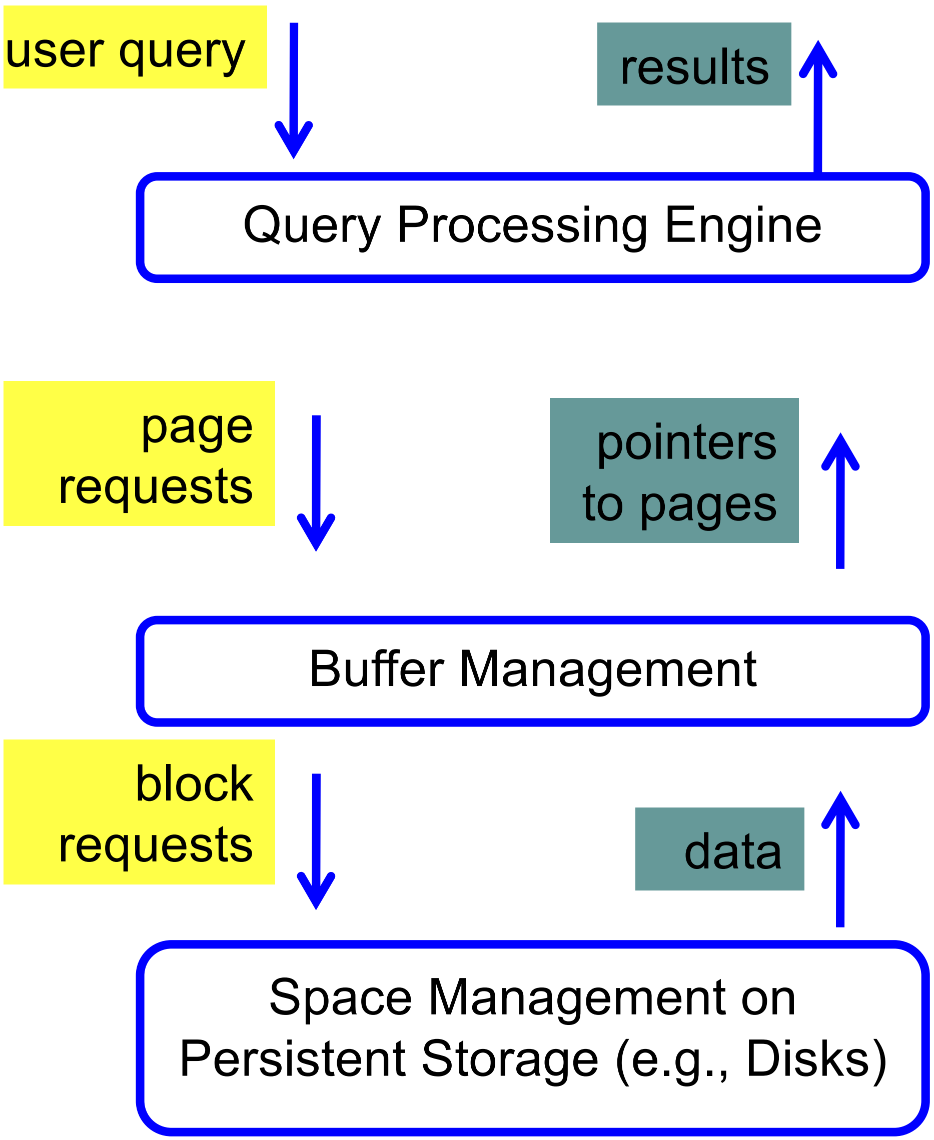
Choosing a RAID level

- 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
  - Level 1 60% more disks
  - Level 5 is preferred for applications with low update rate, and large amounts of data

Plan for Today

- Review:
  - Storage Hierarchy
  - Disks
- Solid State Drives
- RAID: Redundant Array of Independent Disks
  - Although it says “disks”, the ideas are more general
- Buffer Manager
- File Organization on Storage

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

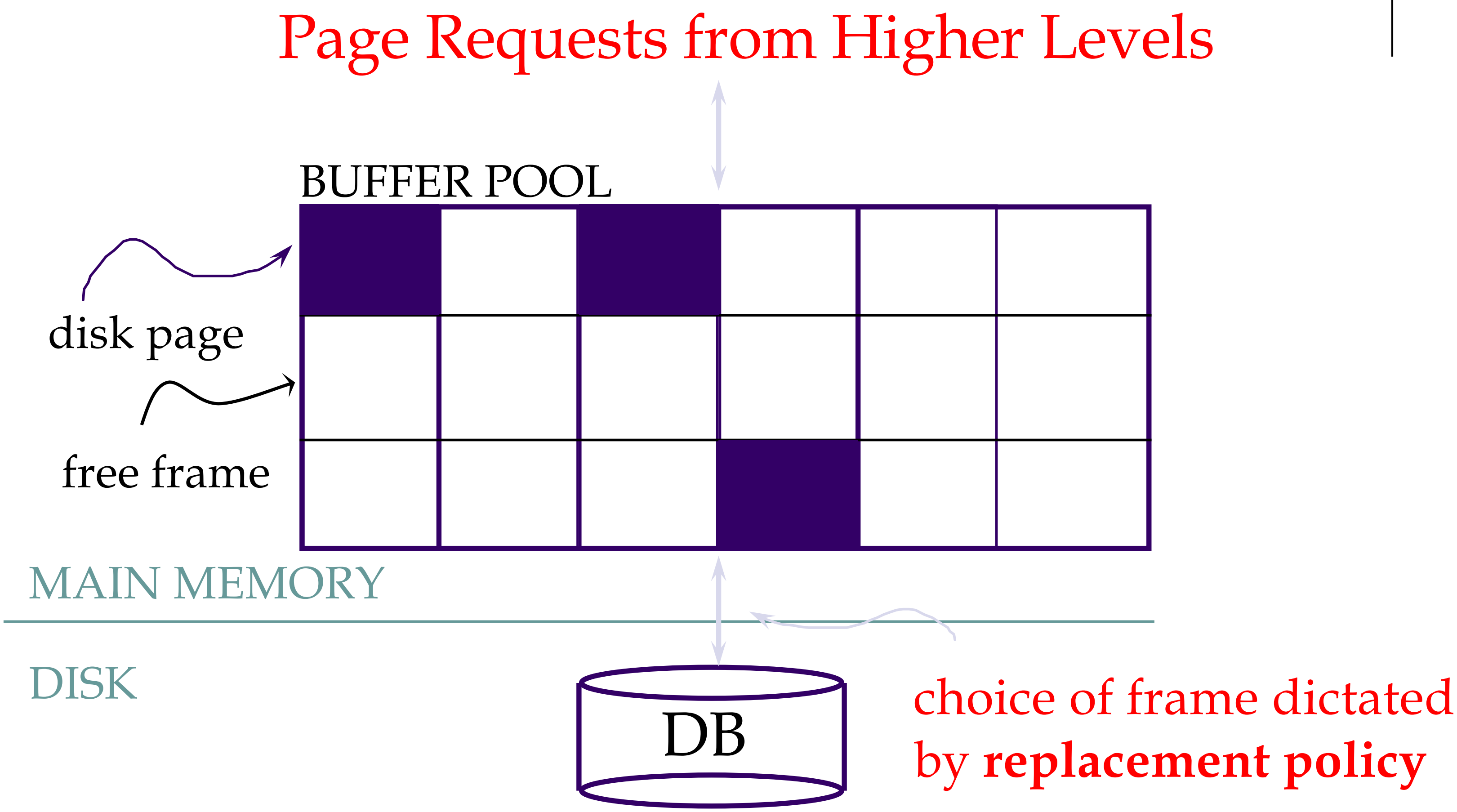
- Bringing pages from disk to memory
- Managing the limited memory

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

Buffer Manager

- When the QP wants a block, it asks the “buffer manager”
  - The block must be in memory to operate upon
- Buffer manager:
  - If block already in memory: return a pointer to it
  - If not:
    - Evict a current page
      - Either write it to temporary storage,
      - or write it back to its original location,
      - or just throw it away (if it was read from disk, and not modified)
    - and make a request to the storage subsystem to fetch it

Buffer Manager



Buffer Manager

- Similar to *virtual memory manager*
- Buffer replacement policies
  - What page to evict ?
  - LRU: Least Recently Used
    - Throw out the page that was not used in a long time
  - MRU: Most Recently Used
    - The opposite
    - Why ?
  - Clock ?
    - An efficient implementation of LRU

Buffer Manager

- *Pinning* a block
  - Not allowed to write back to the disk
- *Force-output (force-write)*
  - Force the contents of a block to be written to disk
- *Order the writes*
  - This block must be written to disk before this block
- Critical for fault tolerant guarantees
  - Otherwise the database has no control over whats on disk and whats not on disk

Plan for Today

- Review:
  - Storage Hierarchy
  - Disks
- Solid State Drives
- RAID: Redundant Array of Independent Disks
  - Although it says “disks”, the ideas are more general
- Buffer Manager
- File Organization on Storage