CSE462/562: Database Systems (Spring 24)

Lecture 1: Introduction & Course Logistics;

Physical Storage

1/29/2024

University at Buffalo
Department of Computer Science
and Engineering
School of Engineering and Applied Sciences

Knox 109, M 4:00 pm – 6:40 pm. In-person attendance required.

Find more on course website & Piazza:
https://cse.buffalo.edu/~zzhao35/teaching/cse562_spring24/
https://piazza.com/buffalo/spring2024/cse462562

Today’s agenda

- Introduction
  - What is a Database?
  - What is a Database Management System?
  - What is this course about and why should I care?

- Logistics
- Physical Storage

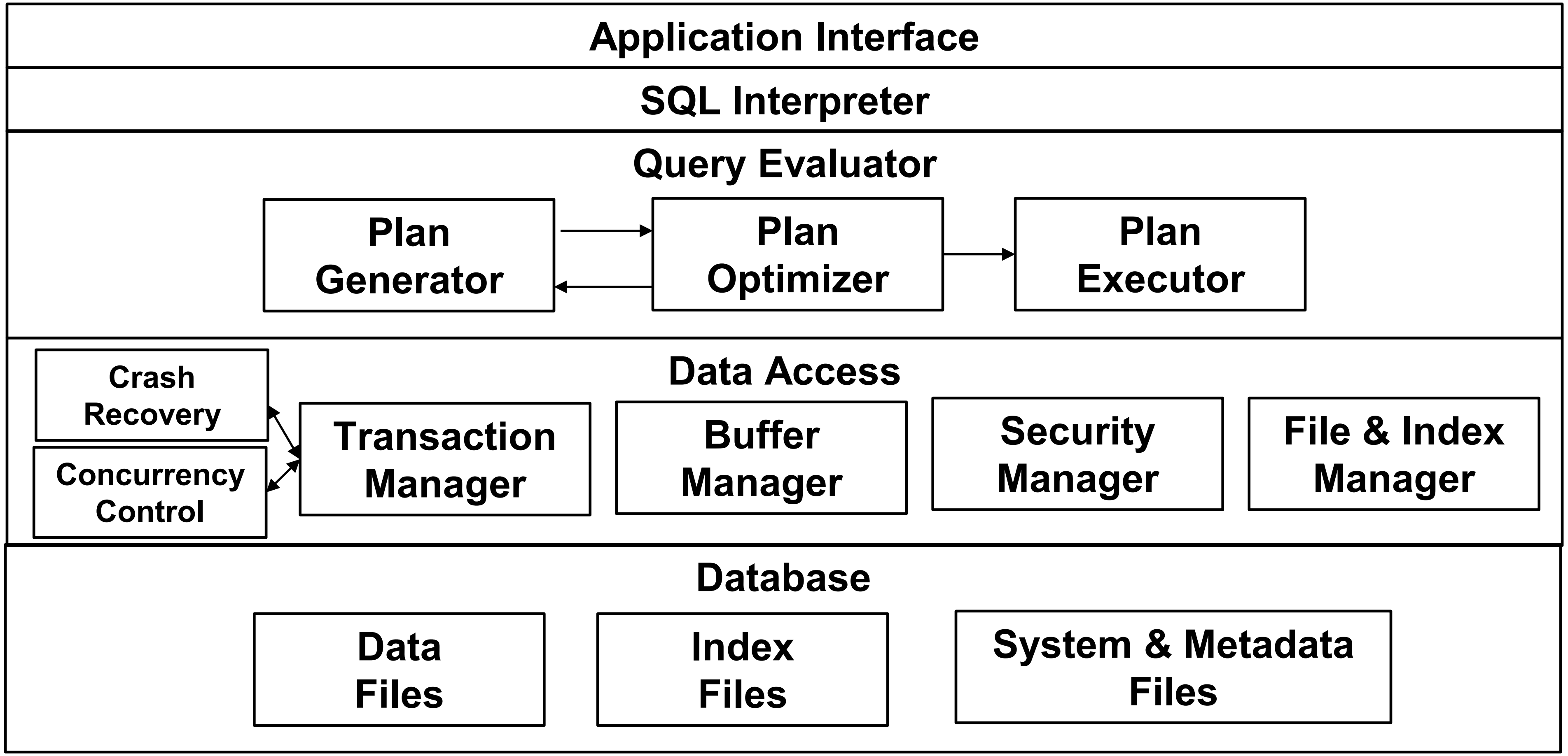
What is a Database?

- Database is
  - a collection of interrelated data
  - often organized in a certain structure for convenient and efficient access

- Databases are found almost everywhere, sometimes unnoticed
  - Business: sales, accounting, human resource, IT support, …
  - Financial industry: banking, credit card, investment platform
  - University: student records, course registration, LMS (e.g., UB Learns), ...
  - Some less obvious examples of databases
    - Software package and configuration DB (e.g., windows registry)
    - Your photo library (e.g., Google Photos)
    - Your personal finance records
    - …

What's a DataBase Management System?

- DataBase Management System (DBMS) is a software system for convenient and efficient data access over databases.

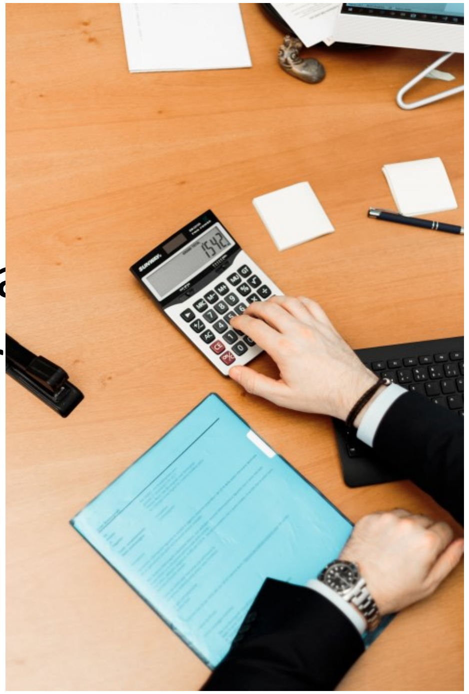


Why using a DataBase Management System?

- Let’s review an example of how to manage a database.

How to manage a database?

- Suppose I’d like to track my daily spending
- What I can do:
  - Step 1: collect all the receipts



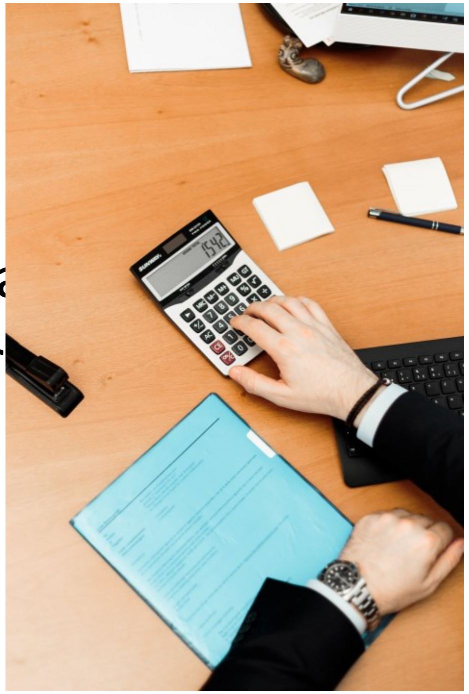
  - Step 2: do some analysis
    - How much did my spend on grocery and fast food in Febura
    - How much could I have saved if I cook by myself in Feburar
    - What about January/last quarter/last year/past five years?

How to manage a database?

- Suppose I'd like to track my daily spending
- What I can do:
  - Step 1: collect all the receipts
  - Step 2: write them down on a notebook

Date	Amount	Description
2/1	$20.21	Grocery
2/2	$10.54	Fast food
2/3	$39.22	Cell phone bill
…		
2/27	$33.00	Clothes

  - Step 3: do some analysis
    - How much did my spend on grocery and fast food in Febura
    - How much could I have saved if I cook by myself in Feburar
    - What about January/last quarter/last year/past five years?



How to manage a database?

- Suppose I'd like to track my daily spending
- What I can do:
  - Step 1: collect all the receipts
  - Step 2: ~~write them down on a notebook~~
    store them in a text file

```
Date   Amount   Description
2/1    $20.21    Grocery
2/2    $10.54   Fast food
2/3    $39.22    Cell phone bill
…
2/27   $33.00    Clothes
```

  - Step 3: do some analysis
    - How much did my spend on grocer
    - How much could I have saved if I co
    - What about January/last quarter/la

```
f = open('myspend_feb_22.txt', 'r')
grocery = 0
fast_food = 0
for line in f:
    date, amount, desc = line.split(' ')
    if desc == 'Fast food':
        fast_food += eval(amount)
    elif desc == 'Grocery':
        grocery += eval(amount)
……
```

How to manage a database?

- Suppose I’d like to track my daily spending
- What I can do:
  - Step 1: collect all the receipts
  - Step 2: ~~write them down on a notebook~~
    ~~store them in a text file~~
    use a spreadsheet
  - Step 3: do some analysis
    - How much did my spend on grocery and fast f
    - How much could I have saved if I cook by myse
    - What about January/last quarter/last year/pas

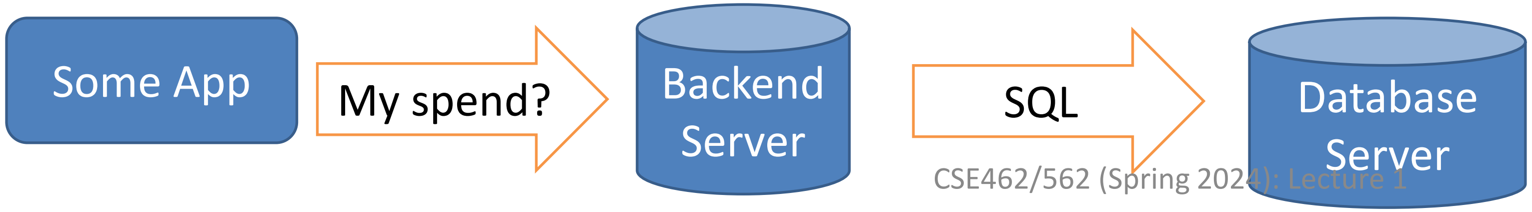
```
Date   Amount   Description
2/1    $20.21    Grocery
2/2    $10.54   Fast food
2/3    $39.22    Cell phone bill
…
2/27   $33.00    Clothes
```

	A	B	C	D	E
1	Date	Amount	Description		
2	1-Feb	20.21	Grocery		
3	2-Feb	10.54	Fast food		
4	3-Feb	39.22	Cell phone		
5					
6					
7		Grocery	=SUMIFS(B2:B4,C2:C4,"Grocery")		

How to manage a database?

- Suppose I'd like to track my daily spending
- What I can do:
  - Step 1: collect all the receipts
  - Step 2: ~~write them down on a notebook~~
    ~~store them in a text file~~
    ~~use a spreadsheet~~
    use/build a personal finance app
  - Step 3: do some analysis
    - How much did my spend on grocery and fast food in Feburary?
    - How much could I have saved if I cook by myself in Feburary?
    - What about January/last quarter/last year/past five years?

```
Date   Amount   Description
2/1    $20.21    Grocery
2/2    $10.54   Fast food
2/3    $39.22    Cell phone bill
…
2/27   $33.00    Clothes
```




```
SELECT category, SUM(amount)
FROM spend
WHERE userid = 123456
GROUP BY category;
```

Why using a DataBase Management System?

- DataBase Management System (DBMS) is a software system for convenient and efficient data access over databases,

  which provides:
  - Data abstraction
    - Flexible data manipulation and query interfaces
    - Scalable data storage
    - Efficient query and transaction processing
  - Integrity checks
  - Concurrency control and atomicity
  - Fault tolerance
  - Security and privacy
  - …

What dose this course cover?

- The design and implementation of DataBase Management System (DBMS)
  - Relational DBMS (RDBMS) as a case study
    - Stores tables that consist of rows and columns
    - Declarative query language (SQL) in the simple yet powerful relational model
  - Focus on principles and techniques generally applicable in Data Management

- Note, this course is not about
  (but we assume you have learned these somewhere else):
  - Database design
  - The relational model and the SQL language (we'll briefly review them)
  - Programming/data structure/algorithm analysis/math…

Why should I care about DBMS internals?

- > 90 billion dollar worth industry
  - Many more are directly or indirectly using DBMS products
- Many vendors and products:
  - Relational: MySQL, Oracle DB, Microsoft SQL Server, IBM Db2, PostgreSQL, SQLite…
  - Graph DB and Graph data processing: Neo4j, Virtuoso, GraphLab, Spark GraphX, …
  - Stream Processing: Apache Flink, Spark Streaming, Apache Storm, …
  - Semi-structured DB: MongoDB, CouchBase, DocumentDB, …
  - Distributed database: Google Spanner, Microsoft CosmosDB, …
  - …
- Used by many other research and application areas:
  - Artificial Intelligence/data mining/search engine/social media/fintech/…

Why should I care about DBMS internals?

- Huge demand in industry for those who can
  - query/manipulate data in database efficiently
  - fine-tune the imperfect DBMS/big data processing systems
  - work seamlessly with the data infrastructure team

- An actively researched area that
  - has strong real-life impacts and connection to the industry
  - has many related open engineering and research positions

- The goal of this course:
  - understanding the common problems and solutions in data management
  - gaining hands-on experience with building a complex software system
  - to be helpful in your future industrial/academic career

Logistics

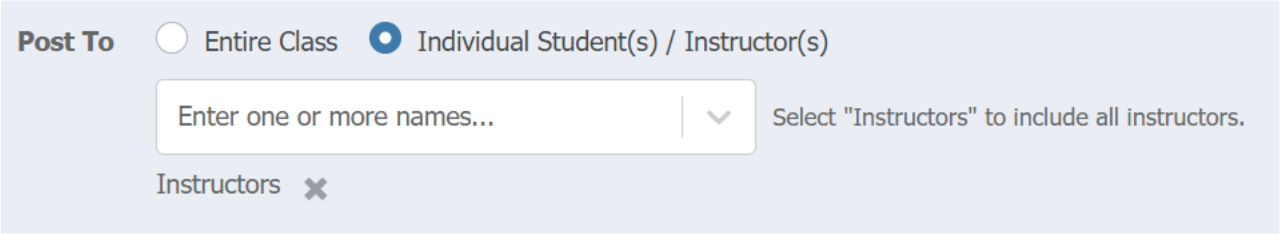
- Knox 109, M 4:00 pm – 6:40 pm.
  - In-person attendance required.
  - Bring some snacks and water if needed ☺
- Instructor: Zhuoyue Zhao
  - Office hours: TBD
- TA/Grader:
  - TBD
- No office hour in week 1
  - Please post on Piazza for help if there's any issue with project 1

- Find more on course website: https://cse.buffalo.edu/~zzhao35/teaching/cse562_spring24/

Logistics

- We mainly use Piazza for communication:
  - https://piazza.com/buffalo/spring2024/cse462562
  - Please post any request/question on Piazza instead of sending emails
    - Piazza reminds me of all unresolved questions but outlook doesn't!

- When you have any private question/request for the instructor or TA:
  - please select "Instructors" in Post To



Logistics

- Important Dates:
  - Mid-term exam: 3/27/2024, Knox 104, 7:05 pm – 8:25 pm (80 minutes)
  - Final exam: 5/15/2024, Knox 109, 3:40 pm – 5:20 pm (100 minutes)
- Exam conflict policy:
  - If you have final exam conflicts as defined by the Office of the Registrar
    - please notify the instructor on Piazza by 2/13/2023
    - (we might not have enough seats if you do not notify us by that date)
    - you may still opt for the original final exam at any time with one-week prior notice

Grading

- Grading
  - Mid-term exam: 20%
  - Final exam: 20%
  - Homework Assignments (20%)
  - Projects: 40% + 10% in bonus

- Grading policy:
  - No curving.

[0, 10)	[10, 20)	[20, 30)	[30, 40)	[40, 50)	[50, 60)	[60, 70)	[70, 80)	[80, 90)	[90, +∞)
F	D	C-	C	C+	B-	B	B+	A-	A

Exams and Assignments

- 6 written assignments
  - 5% each, lowest 2 are excluded from your final grade
  - Similar problems that will appear in exams
  - Must be written electronically in LaTeX (encouraged) or word
    - Do not submit scans of handwriting
- Exams
  - Open-book exams
    - Only paper-copy of the course slides, the written assignments and solutions, the optional textbook, and your lecture notes are allowed
    - No electronic devices except a calculator

Course project

- Build a mini RDBMS through 5 projects (C++ 17)
- Teams allowed with up to 2 students
  - teamwork allowed only within teams
  - see academic integrity policy for details
- Using generative AI is disallowed
- Code must be kept in private Github repository, even after this semester

Course project

- Instructions for projects:
  - Project pages contain very detailed instructions.
    - If something requires clarification, it's most likely covered there.
  - Still have questions on project or found bugs?
    - Feel free to post it on Piazza (though we may point you back to the instructions).
    - Your team will get 1 extra credit towards your final grade for every validated bug or question that cannot be answered by the project instruction.

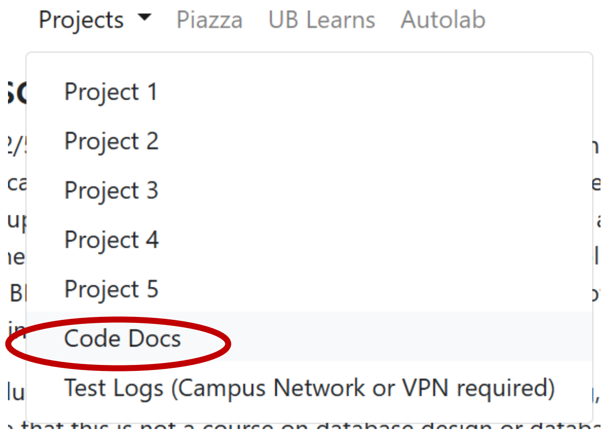
- Where to find project pages: https://cse.buffalo.edu/~zzhao35/teaching/cse562_spring24/

Project 1

- Project 1 is designed as a warm-up project
  - Two labs with two separate submissions required

- Lab 0: project sign-up
  - Please find a teammate, and follow the repository and sign-up instructions
  - Due 2/1, 11:59 PM EST, no late submissions allowed

- Lab 1: build a simple C++ class that encapsulates POSIX I/O interfaces
  - Goal: get familiar with reading documentations
  - Use `man <function_name>` command to find syscall docs
  - Find code docs of Taco-DB from the Project drop-down menu
  - Due 2/4, 11:59 PM EST, see late policy
    - Submission will be open no later than 2/2.



Project/assignment submission & late policy

- All submission are done through Autolab
  - https://autolab.cse.buffalo.edu/courses/cse462-s24
  - If you don't see the course in your Autolab landing page, message us on Piazza
- Late policy:
  - For each submission, 10-minute grace period is allowed.
  - Each student will have 3 grace days throughout the semester.
    - For each project/assignment, you may use up to 1 grace day with no penalty to your grade
  - Examples:
    - You submit project 1 - 3 within a day after the posted deadlines
      - No penalty to the grades.
    - You submit project 1, HW1, project 2, project 3 within a day after the posted deadlines
      - No penalty to the grades of project 1, HW1, project 2.
      - No points will be received for project 3.
    - You submit HW1 after one day after the posted deadline
      - No points will be received for HW1 (but it will be graded to provide you feedbacks)

Academic Integrity Policy

- Academic integrity is critical to the learning process. It is your responsibility to understand and follow all the departmental and university academic integrity policies.
- Zero tolerance towards academic integrity violations, which includes but are not limited to
  - Sharing/copying code in projects or
  - Plagiarizing write-ups
  - Cheating in exam
  - Making project code publicly available or available to any current or future students
  - Submitting code repository that does not belong to you
  - ***(New) Use of generative AI in this class for any coursework***
- Any AI violation will result in an F grade and will be reported to the Office of Academic Integrity
  - unless it's an honest mistake that does not give anyone any undue advantage
    - (e.g., you accidentally set your Github repo to public but changed it back before anyone accesses it)

More on Academic Integrity Policy

- Think of the course projects as take-home exams:
  - you must complete them by yourself (or with your teammate for coding only)
  - please do not discuss any project specifics outside your team

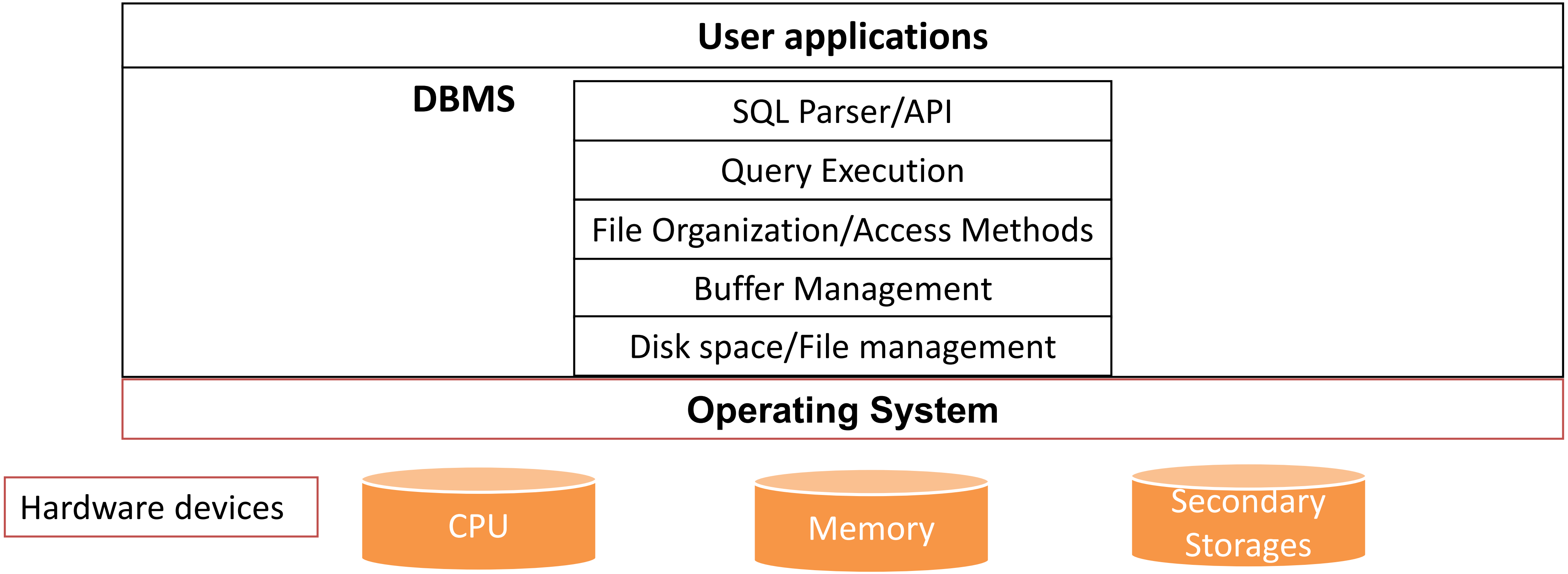
- Examples of AI violation related to course project:
  - Discussion of code with any student who is not your teammate
  - Viewing/committing/submitting code written by anyone who is not your teammate
    - verbatim or with modification
    - ***including those generated or adapted from outputs from generative AI software (e.g., ChatGPT)***
  - Discussion of project write-ups with any student (including your teammate)
  - Viewing/copying/rephrasing answers found online or from a past or current student

- What is allowed and encouraged (on Piazza/in lecture/offline, publicly or privately)
  - Ask questions about lectures
  - Preparation for mid-term and final exams
  - Seek clarification about projects/homework assignments
  - If you’re unsure, please do ask.

Short break

- Upcoming: physical storage

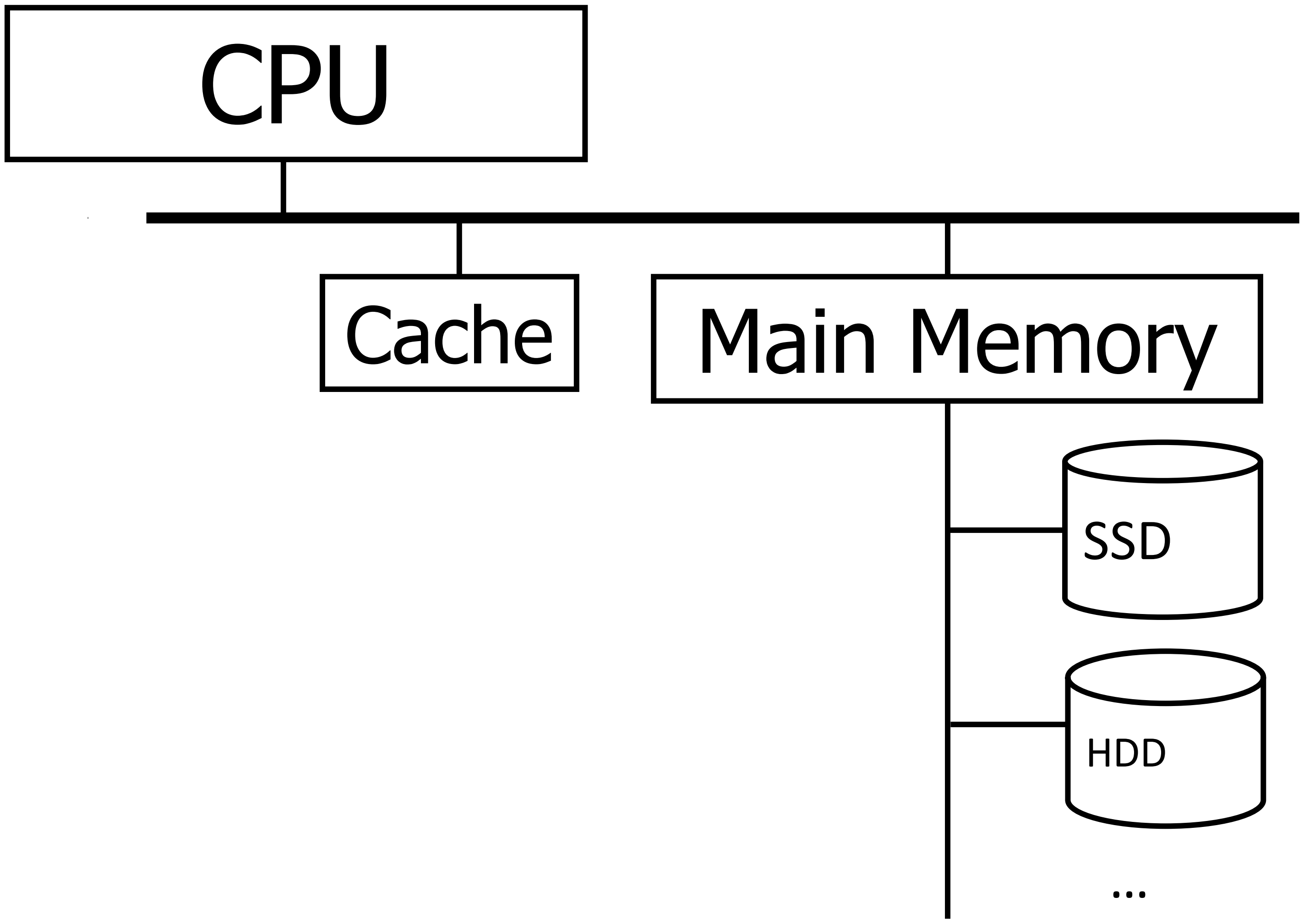
Big Picture



User applications
DBMS
SQL Parser/API
Query Execution
File Organization/Access Methods
Buffer Management
Disk space/File management
Operating System
Hardware devices
CPU
Memory
Secondary Storages

Typical (& oversimplified) computer architecture

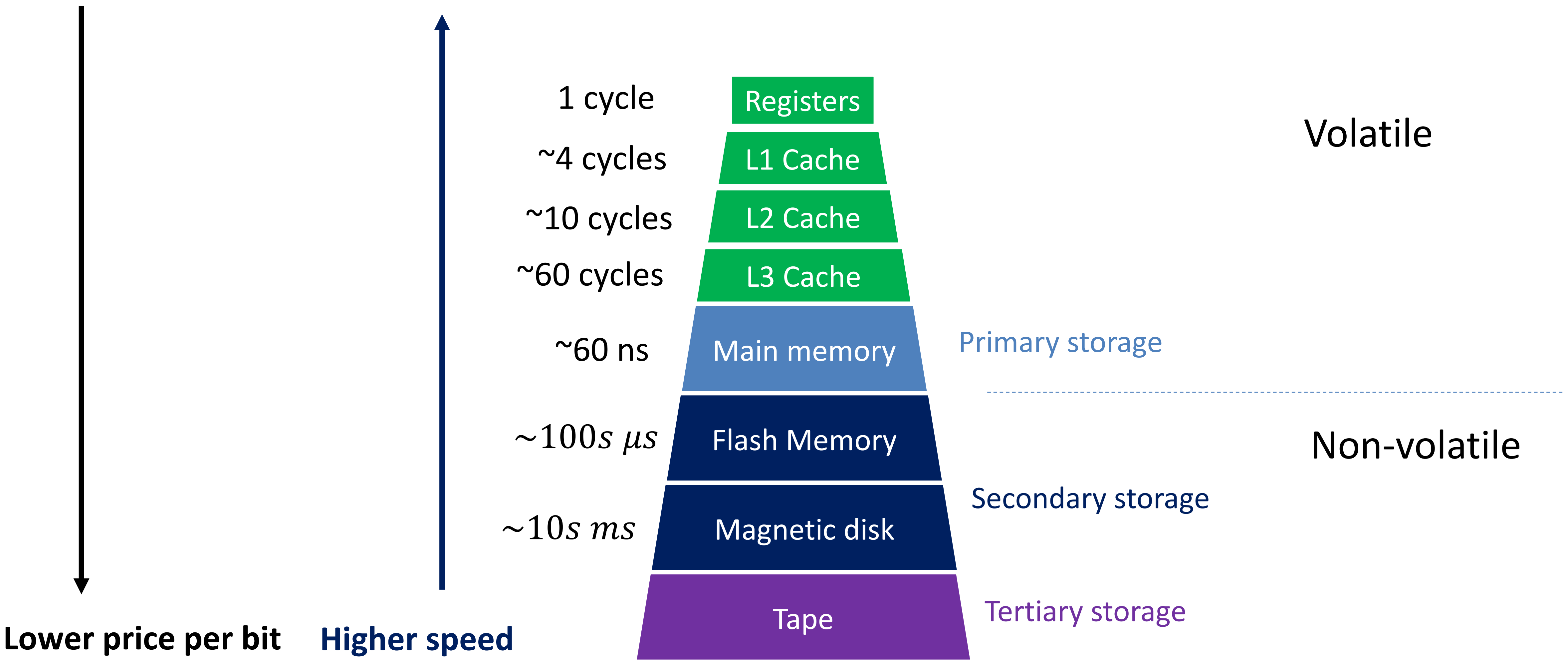
- A simplistic view of a computer



Typical Computer

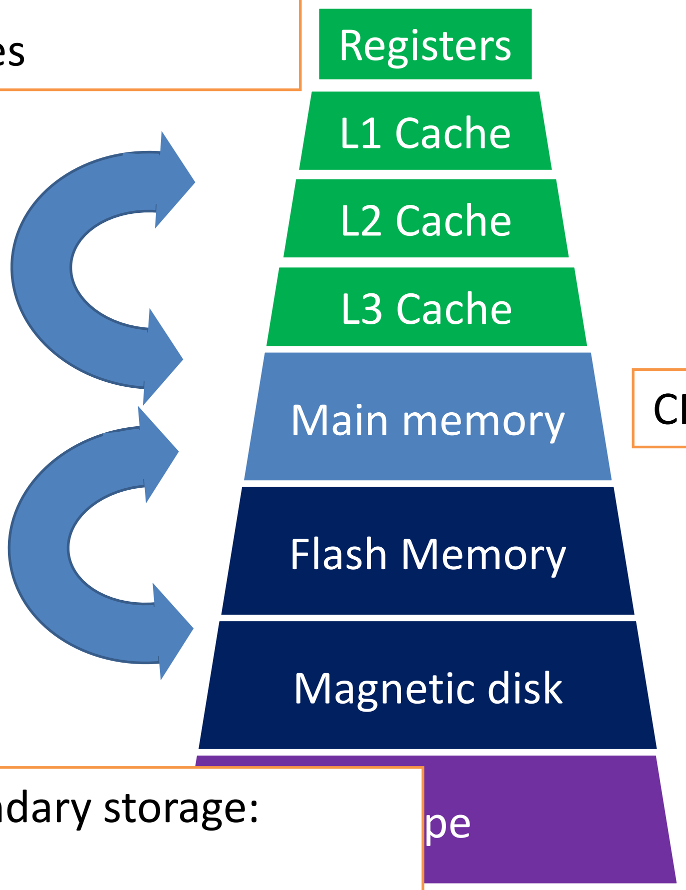
Secondary Storage

Storage Hierarchy



Data Transfers

Between cache and main memory:
hardware/OS controlled
usually in small units of cache lines



Volatile

CPU operates on main memory (byte addressable)

Non-volatile

Between main memory and secondary storage:
DBMS controlled (read/write)
usually with large block I/O

Volatile storage

- Register
  - Very fast but very limited amount
  - CPU directly operates on registers
- Cache
  - Faster than main memory but takes multiple cycles to access
  - Stores cache lines that are likely to be read/write again
  - Usually managed by CPU
- Main memory
  - Still quite fast albeit it takes hundreds of cycles
  - CPU instructions can read/write byte addressable data into/from registers

Why not store everything in memory?

- Too expensive
  - Data growth is much faster than what you can afford
- Volatile
  - Power loss -> data loss
- Typical storage hierarchy in (traditional) DBMS
  - Main memory as buffer/working space
  - Disk as the main database storage
  - Tape for archiving old data

  - Main memory DB actually uses memory for main database storage
    - Persistency of data? Logging & checkpointing (later lectures)

Non-volatile storage

- Common non-volatile (secondary) storage
  - Flash memory (e.g., SSD)
  - Magnetic disk

- Advantages
  - Cheaper -- can store much more data than memory with the same cost
  - Non-volatile – data are saved in server shutdown/power failure

- Disadvantages
  - Block device: read/write in the units of sectors (usually 512B/4096B)
  - Higher latency: usually >= 1 – 2 orders of magnitude slower than main memory

- Tertiary storage: tape (sequential I/O only)
  - Very slow but inexpensive; good for archiving data

Closer look at non-volatile storage

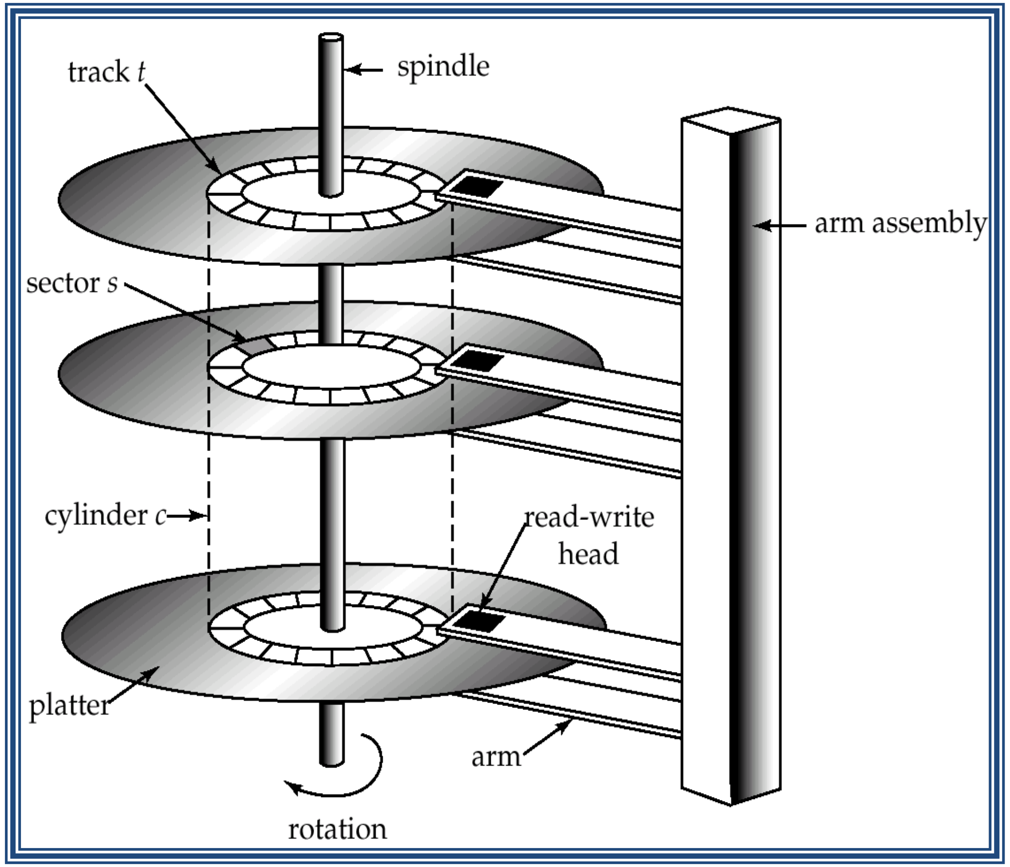
- We need to know the performance characteristics of non-volatile storage
  - to optimize database storage design

Magnetic disk (HDD)

Solid State Drive (SSD)

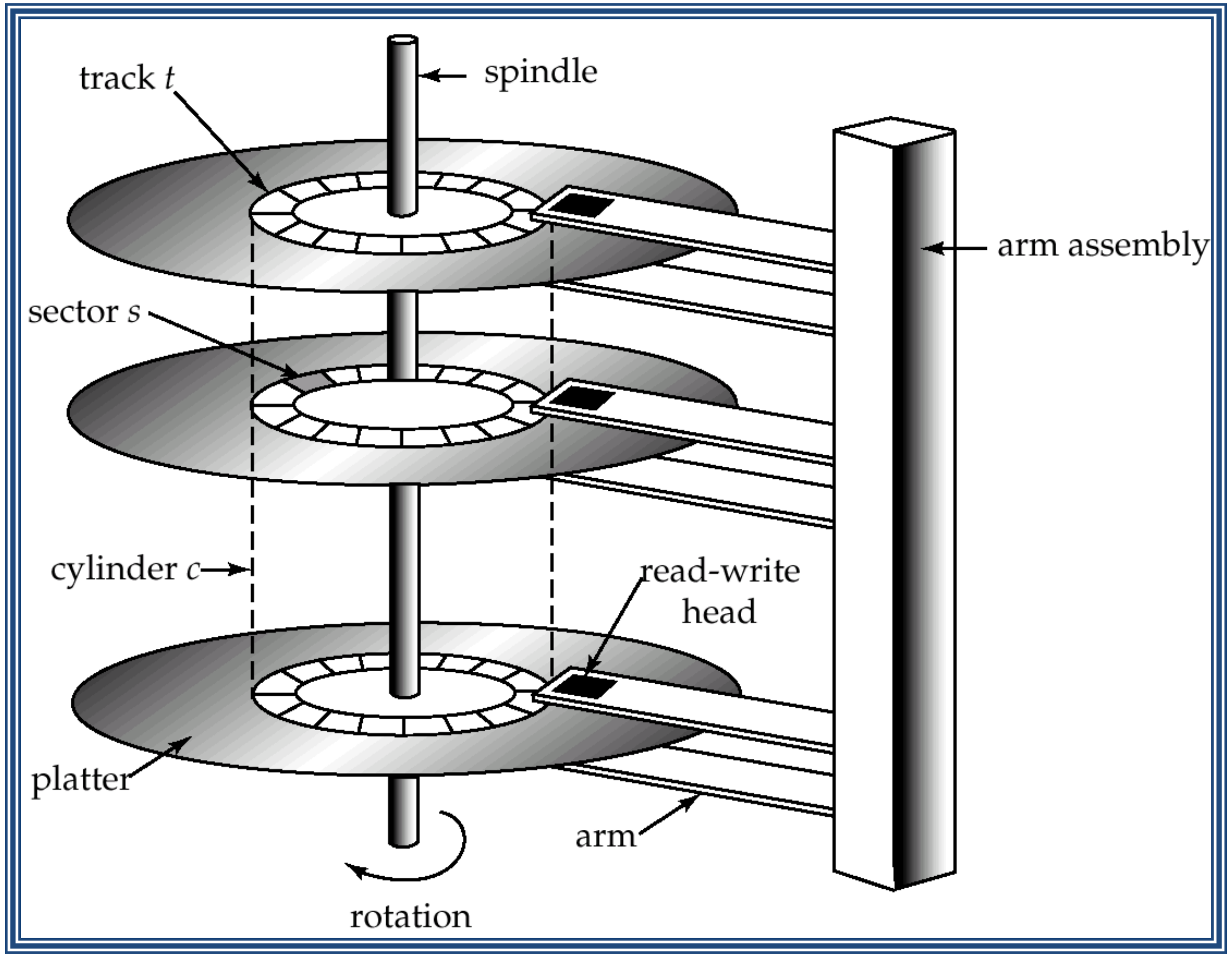
Magnetic disk organization

- Multiple platters
  - Each platter has *two* surfaces for data storage
  - Platters spin at the *same* rate (e.g., 7200 rpm)
  - A ring on a surface is called a track
    - A track is divided into many sectors of fixed size (512 B)
    - A sector is the *smallest* unit of I/O
- A single arm assembly with multiple disk heads
  - Can only move inward/outward *together*
  - The vertical stack of tracks is called a cylinder
    - Disk heads can be over the tracks of the *same* cylinder at the *same* time
  - Usually one read/writes at the same time
- Address of a sector: cylinder - head - sector
  - (0, 0, 0) : first sector; (0, 0, 1): second sector, …
    (0, 1, 0) : the S^{th} sector, (1, 0, 0) the $(SH)^{th}$
    where S is the max # of sectors/track and H is the # of heads
  - Reality: today's disks use logical block addressing (linear block #)
    - Translated to the actual geometry by disk controller
    - Nevertheless, this is still a good model for understanding HDD performance.



Magnetic disk I/O latency

- File systems perform I/O in units of multiple sector (page)
  - 4KB~16KB are most common
- Break-down of I/O latency of a page
  - Seek time: moving arms to the cylinder
    - 2 ~ 20 ms per seek
    - 4 ~ 10 ms on average
  - Rotation delay:
    wait for the sector to be under a head
    - Depending on rotation speed (5400 rpm - 15000 rpm)
    - E.g, 7200 rpm = 120 rotations/second
      => 1/120 = 8.33 ms / rotation
      on average it needs a half rotation
      => 8.33 / 2 = 4.17 ms on average
  - Transfer time: time for reading/writing data
    - Data transfer rate: 50 - 200 MB/s
    - ⇔ 0.02 ~ 0.08 ms for 4KB pages
- Average access time
  - 4KB page, 7200 rpm: roughly 8 ~ 15 ms



Impact of I/O pattern on magnetic disk

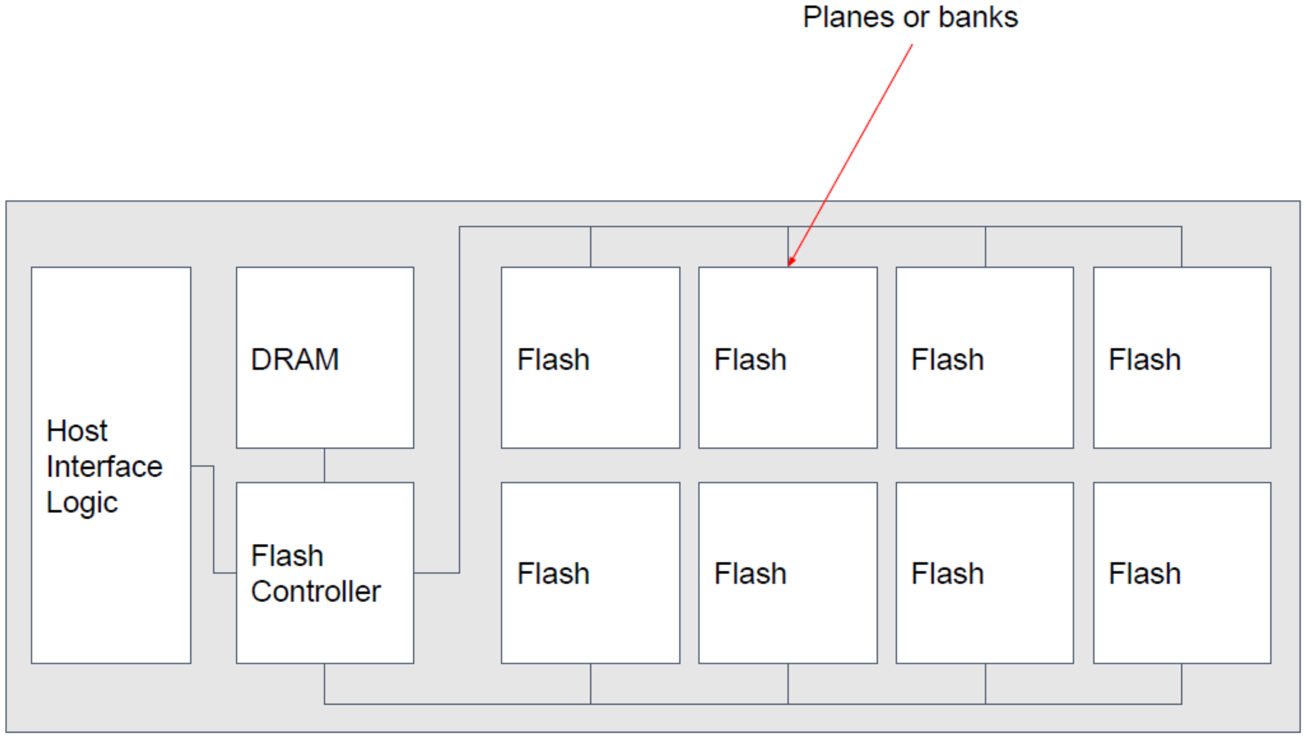
- I/O pattern has a huge impact on I/O performance
  - E.g., 4KB page size
    - Sequential read/write: usually 100 ~ 200+ MB/s
    - Random read/write: 50 ~ 200 IOPS ⇔ 200 KB ~ 800 KB /s

    - > 2 orders of magnitude difference in terms of data transfer rate

  - Rule of thumb:
    - Random I/O: very slow; avoid reading a lot of data from random location
    - Sequential I/O: better for accessing a lot of data

Flash memory / solid state drive

- NAND Flash is the most common storage media for solid state drives
- No mechanical parts (magnetic disk can have head crash => data corruption/loss)
  - More reliable; less likely to fail due to physical shocks
- Faster than magnetic disk



Flash memory / solid state drive

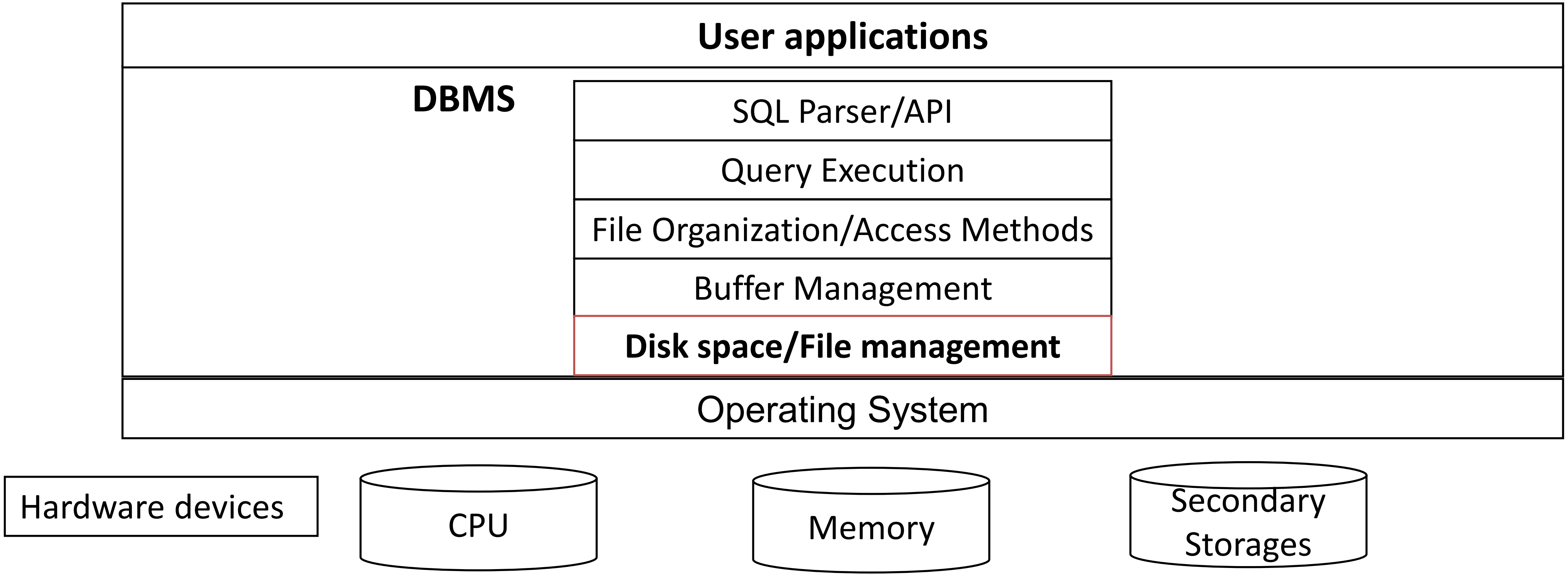
- NAND SSD has asymmetric read/write performance
  - 4KB page, typical SSD internal performance numbers
    - Read latency: 20 to 100 μs ; throughput: > 500 MB/s
    - Write latency: 200 μs; throughput: > 500 MB/s
    - Erase latency: ~2 ms

  - Three ops: read/write/erase
    - Read/write works on pages (usually 4KB)
      - Write can only change some bits from 1 to 0 (not the other way around!)
      - Muse erase before write a page.
    - Erase works on blocks (e.g., 256 KB)
      - Resets all bits in a block to 1
      - Flash translation layer: indirection of page numbers to physical pages
        - Solves two problems: slow erase and flash wear
  - Actual performance also often bound by peripheral bus's bandwidth and IOPS

Flash memory / solid state drive

- NAND SSD has asymmetric read/write performance
  - The performance from DB stand of view?
    - No single answer depending on how you use it
      - I/O queue depth, I/O api, access pattern, page size, peripheral bus type and etc.
    - In a typical case:
      - Sequential I/O is still preferred, although random I/O isn't as bad as in HDD
      - SSDs have much better random I/O performance than magnetic disk
        - 10k - 1M IOPS
      - and higher bandwidth as well
        - up to 7GB/s on PCIe 4.0, ~500MB/s on SATA

Big Picture


User applications
DBMS
SQL Parser/API
Query Execution
File Organization/Access Methods
Buffer Management
Disk space/File management
Operating System
Hardware devices
CPU
Memory
Secondary Storages

File System Interface

- POSIX I/O interface
  - A standard synchronous I/O interface
  - Agnostic to the underlying storage device/file system

A *file descriptor* is a reference to an *open file description*, an entry in the system-wide table of open files that records file offsets and file status flags.

open(2): open and possibly create a file -> *file descriptor* (int)

```
int fd = open("/data/a.dat", O_RDONLY | O_CREAT, 0644);
```

- `fd`: Case 1: fd >= 0 on success. Case 2: fd == -1 if an error occurred -- check `errno` for reasons; also see `strerror(3)`
- `"/data/a.dat"`: opens the file at path `/data/a.dat`
- `O_RDONLY | O_CREAT`:
  1. read-only access
  2. create the file if it does not exist
- `0644`: The permission bits if the file is created. 0644 = rw allowed for user (file owner); read only for group & others.

File System Interface

- POSIX I/O interface
  - A standard synchronous I/O interface
  - Agnostic to the underlying storage device/file system

A *file descriptor* is a reference to an *open file description*, an entry in the system-wide table of open files that records file offsets and file status flags.

open(2): open and possibly create a file -> *file descriptor* (int)

```
int fd = open("/data/a.dat", O_RDONLY | O_CREAT, 0644);
```

pread(2), pwrite(2): read from or write to a file descriptor at a given offset

```
char buf[4096];
ssize_t sz = pread(fd, buf, 4096, 1048576);
if (sz == 4096) /* success */; else /* error */;
```

reading 4096 bytes at file offset 1048576 = 4096 * 256 (i.e., reading page 255 from a file assuming 4KB pages)

File System Interface

- POSIX I/O interface
  - A standard synchronous I/O interface
  - Agnostic to the underlying storage device/file system

A *file descriptor* is a reference to an *open file description*, an entry in the system-wide table of open files that records file offsets and file status flags.

open(2): open and possibly create a file -> *file descriptor* (int)

```
int fd = open("/data/a.dat", O_RDONLY | O_CREAT, 0644);
```

pread(2), pwrite(2): read from or write to a file descriptor at a given offset

posix_fallocate(3), fallocate(2)

fsync(2), fdatasync(2),

close(2)

Check man pages for more details.

Disk Space Management

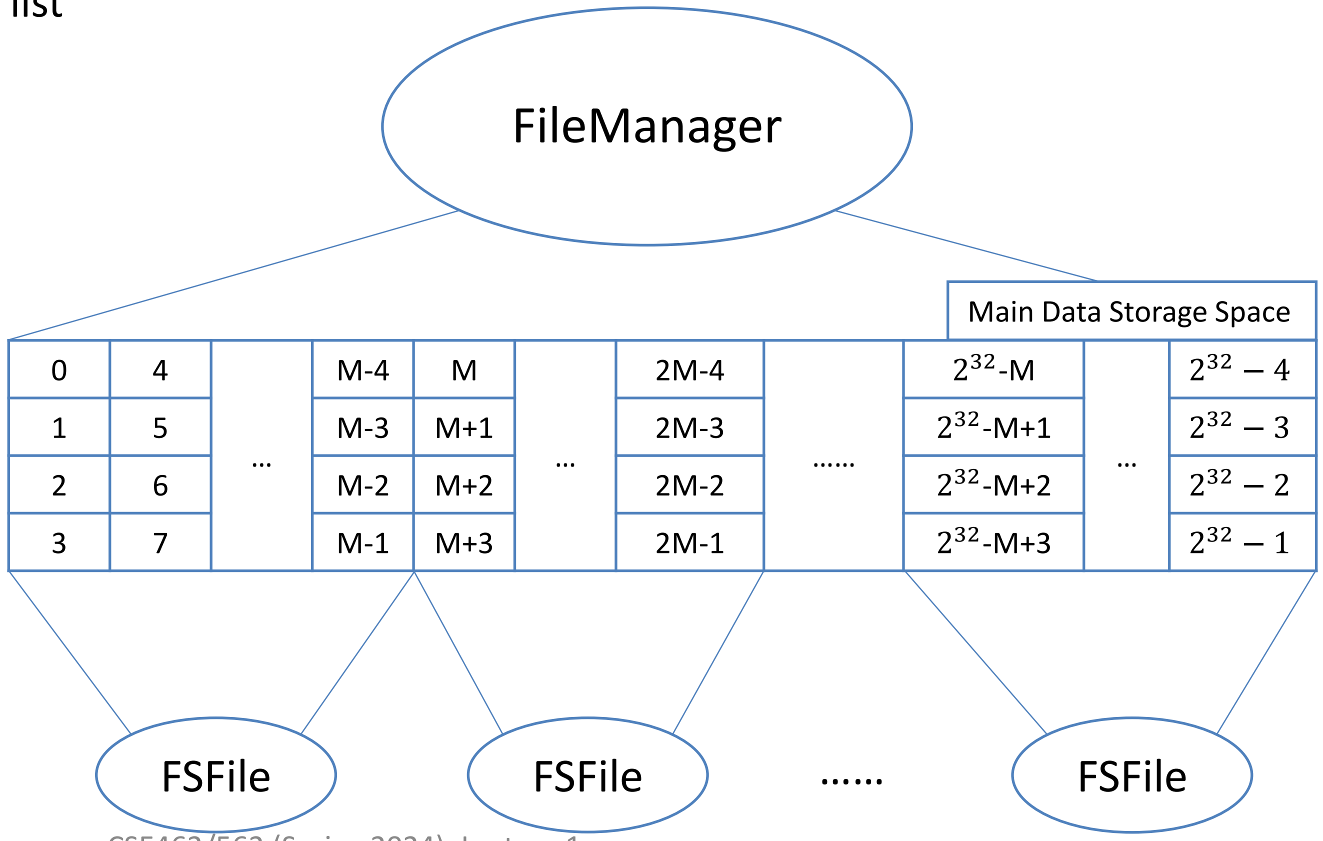
- Lowest layer of DBMS software manages space on disk
  - Disk space is usually organized in *pages*
    - which may not necessarily directly be mapped to disk sectors/file system pages!
    - common choices are 4KB, 8KB, 16KB, etc.
  - Using the OS file system or not? Some do and some don't!
  - Even with file system
    - How to organize pages (in one file/multiple files)?
    - How to deal with concurrency/recovery?
    - …

- Higher levels call upon this layer to:
  - allocate/de-allocate a page
  - read/write a page

- Best if a request for a sequence of pages is satisfied by pages stored sequentially on disk!
  - Responsibility of disk space manager.
  - Higher levels don't know how this is done, or how free space is managed.
  - Though they may assume sequential access for files!
    - Hence, disk space manager should do a decent job.

Disk Space Management in course project Taco-DB

- A flat main data storage page from page 0 to page $2^{32} - 1$
  - Stored as 64GB files on the local file system;
  - One instance of FSFile manage a real file in the file system (e.g., allocate/read/write a page).
    - This is your task in Project 1 – lab 1.
  - FileManager manages many virtual files *(more on this next week)*
    - Each is a double-linked list of pages, allocated in groups of 64 consecutive pages
    - Each file maintains its own free list



Summary

- This lecture
  - Introduction & logistics
  - Storage hierarchy and storage devices
  - Disk space management

- Next lecture
  - Buffer management
  - File organization in DBMS
  - Data storage layout

- Project 1 released
  - Due 2/1 23:59 PM EST (lab 0), 2/4 23:59 PM EST (lab 1)