CSE462/562: Database Systems (Spring 22)

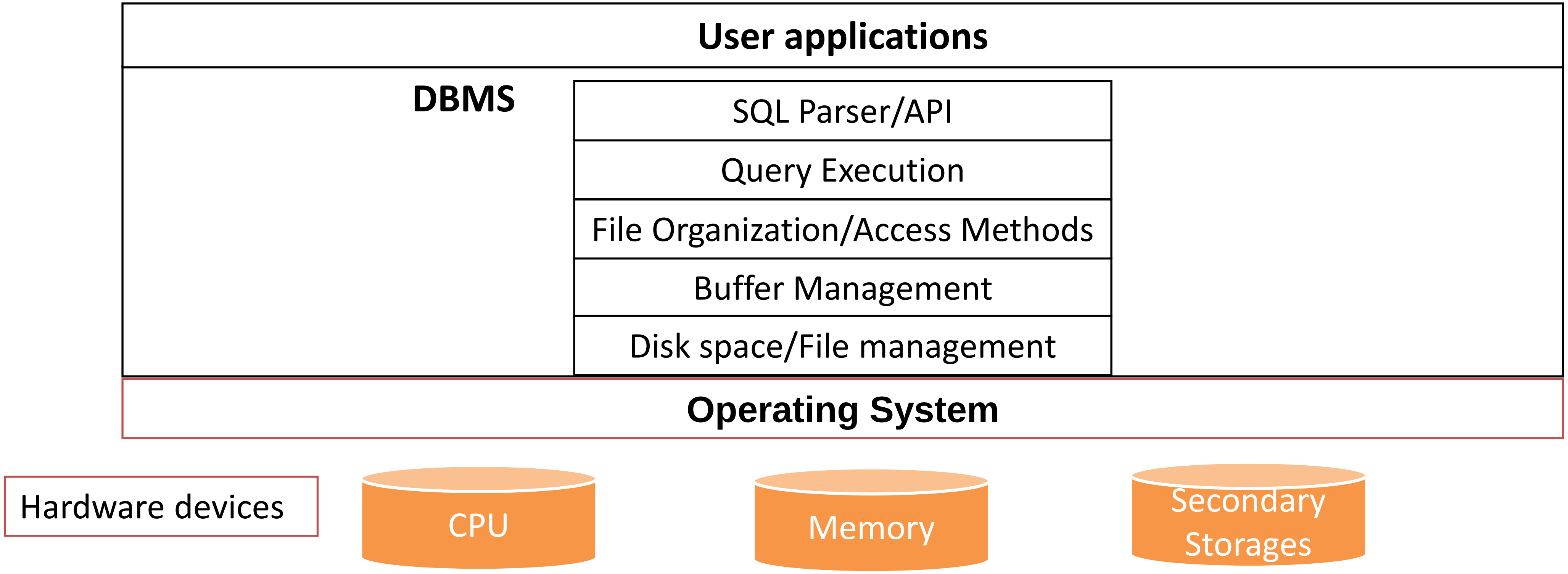
Lecture 5: Physical Storage and Buffer Management

2/15/2022

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

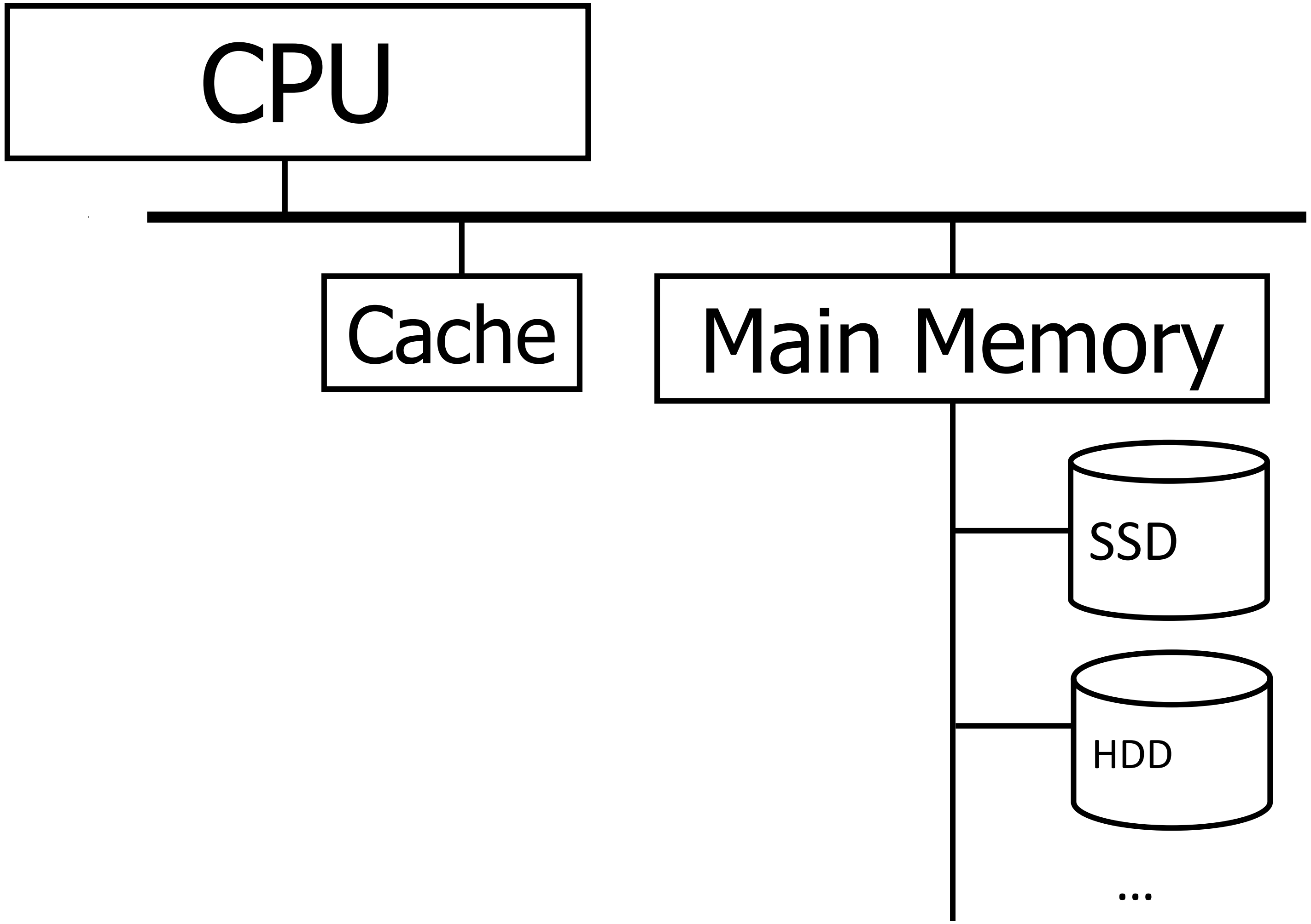
Slide credit to Dong Xie @ PSU

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

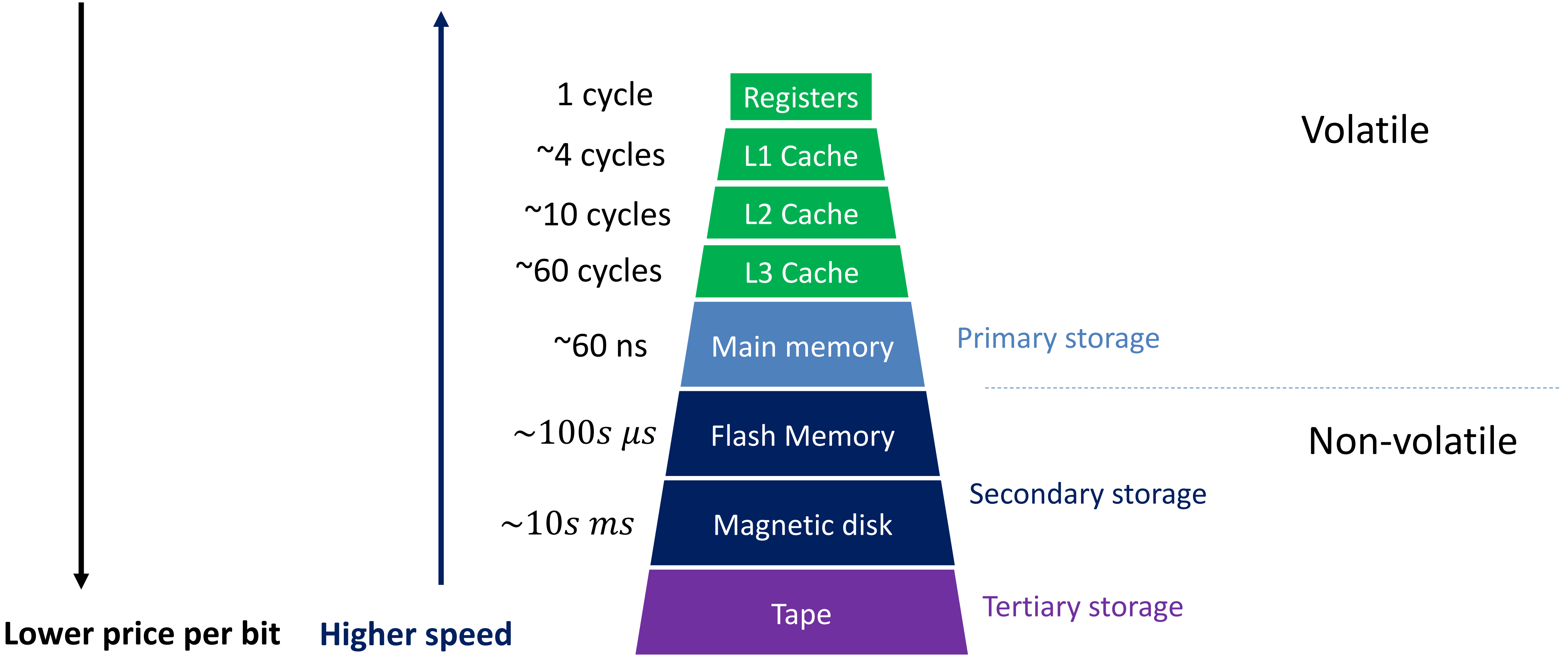
- A simplistic view of a computer



Typical Computer

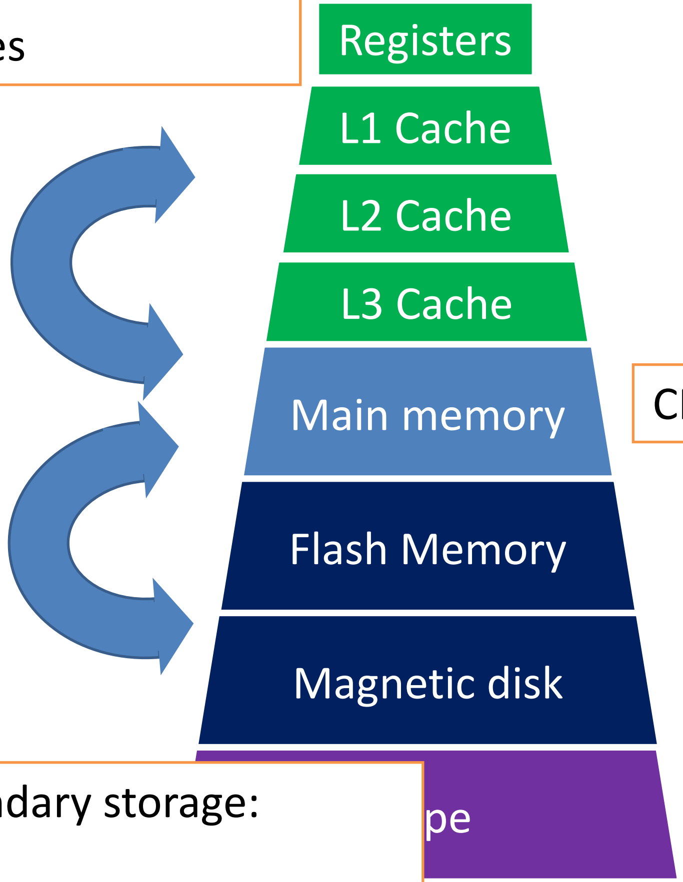
Secondary Storage

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/replication (later lectures)

Non-volatile storage

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

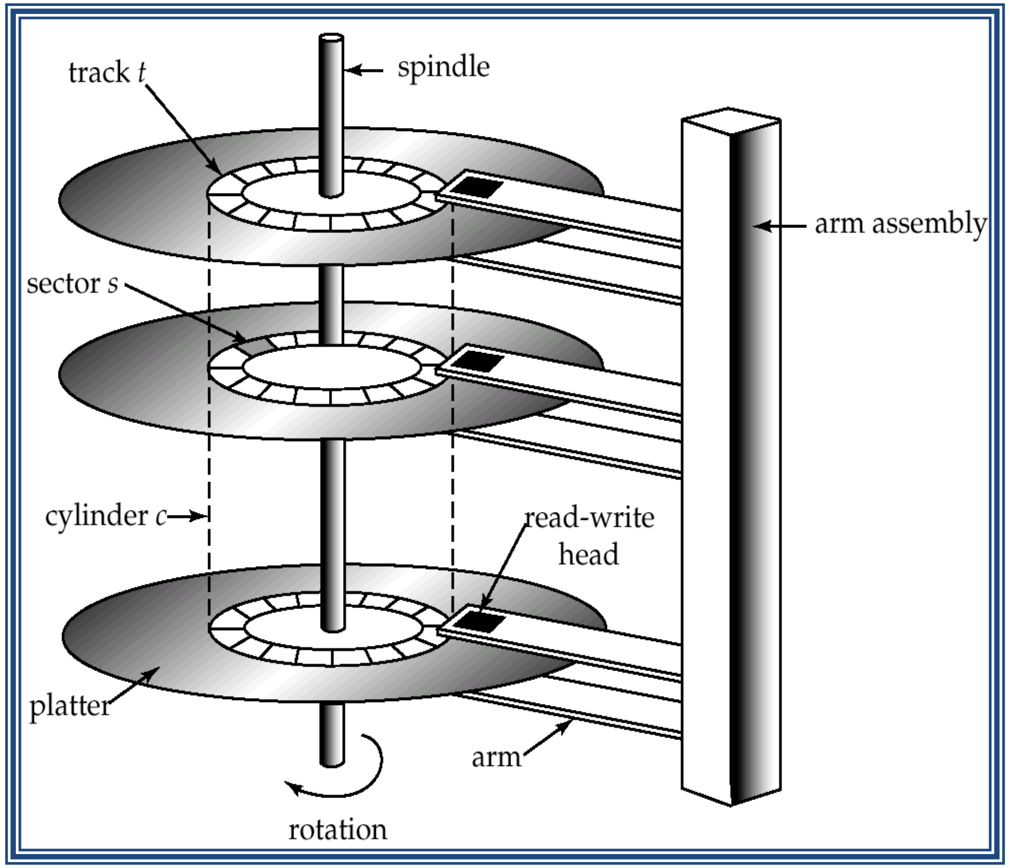
- Advantages
  - Cheaper -- can store much more data than memory with the same cost
  - Non-volatile – data are saved in server shutdown/power failure

- 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

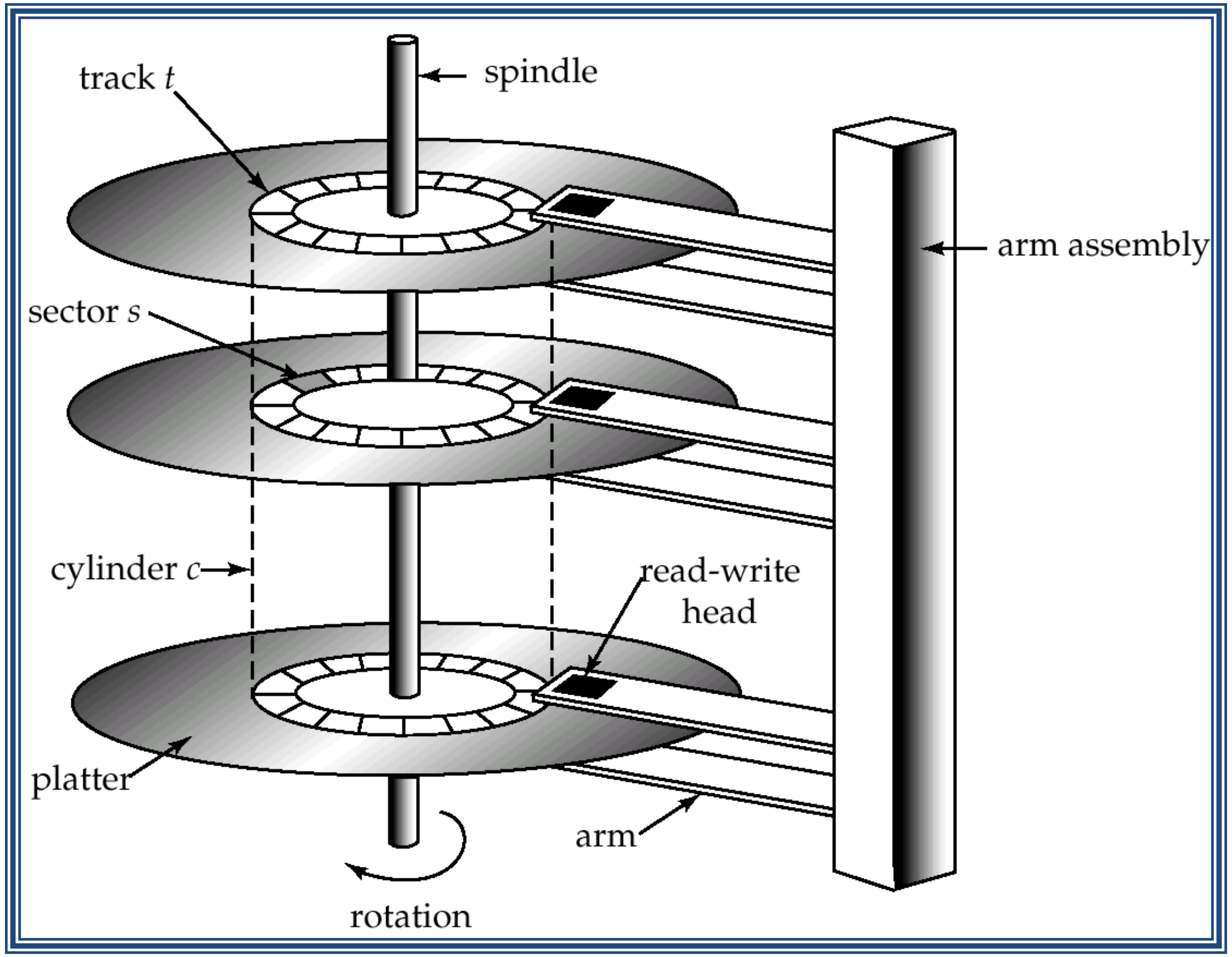
Magnetic disk organization

- 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 (usually 512 bytes)
    - 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



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

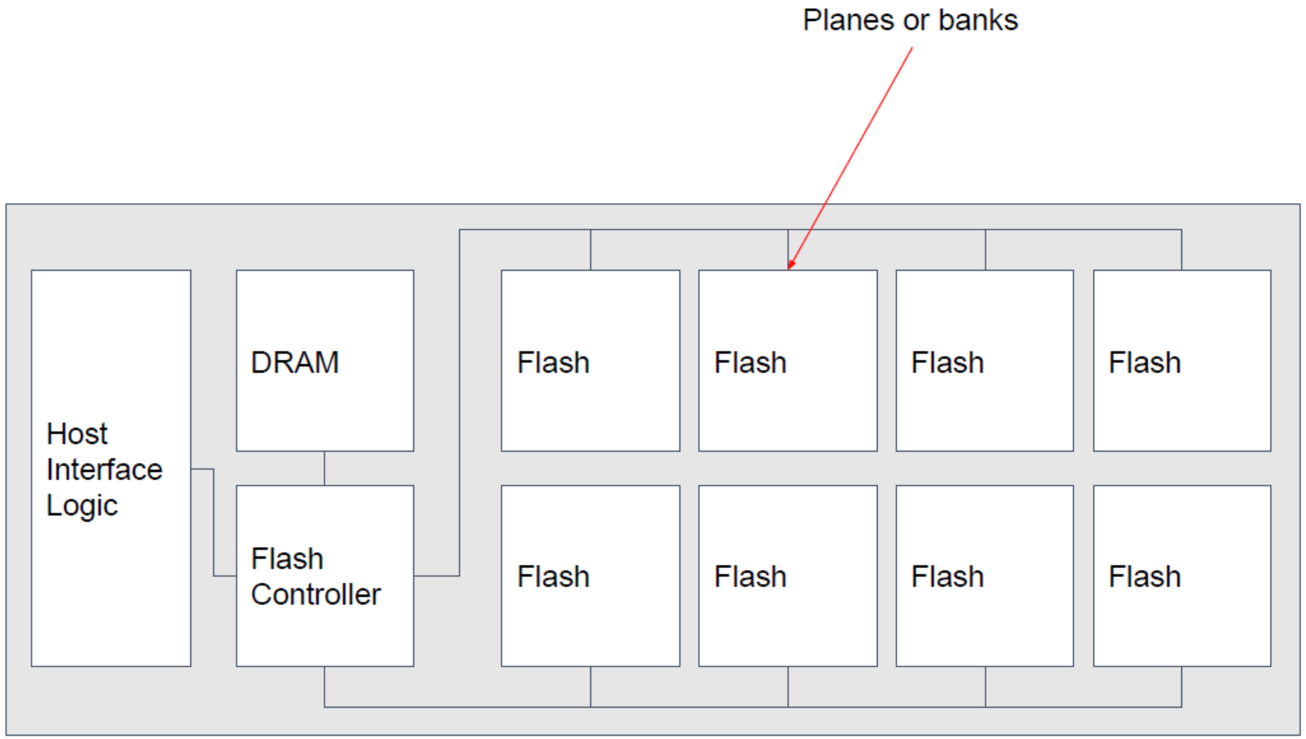


Impact of I/O pattern on magnetic disk

- 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 storage media for solid state drives
  - SSD that uses (e.g., Intel 3D XPoint and etc.)
- 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

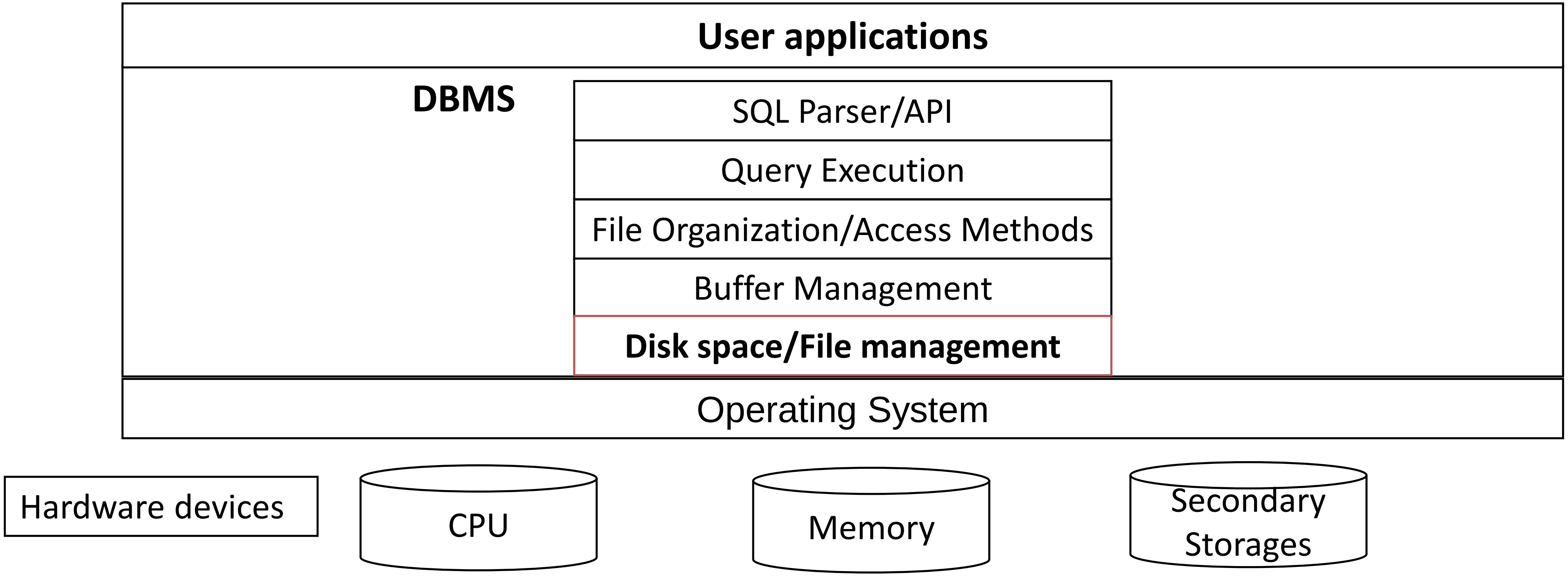
  - Three ops: read/write/erase
    - Read/write works on pages (usually 4KB)
      - Write can only change some bits from 1 to 0 (not the other way around!)
      - Muse erase before write a page.
    - Erase works on blocks (e.g., 256 KB)
      - Resets all bits in a block to 1
      - Flash translation 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.

    - But they have much better random I/O performance than magnetic disk
      - 10k - 1M IOPS
    - and higher bandwidth as well
      - up to 7GB on PCIe 4.0, ~500MB 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

Disk Space Management

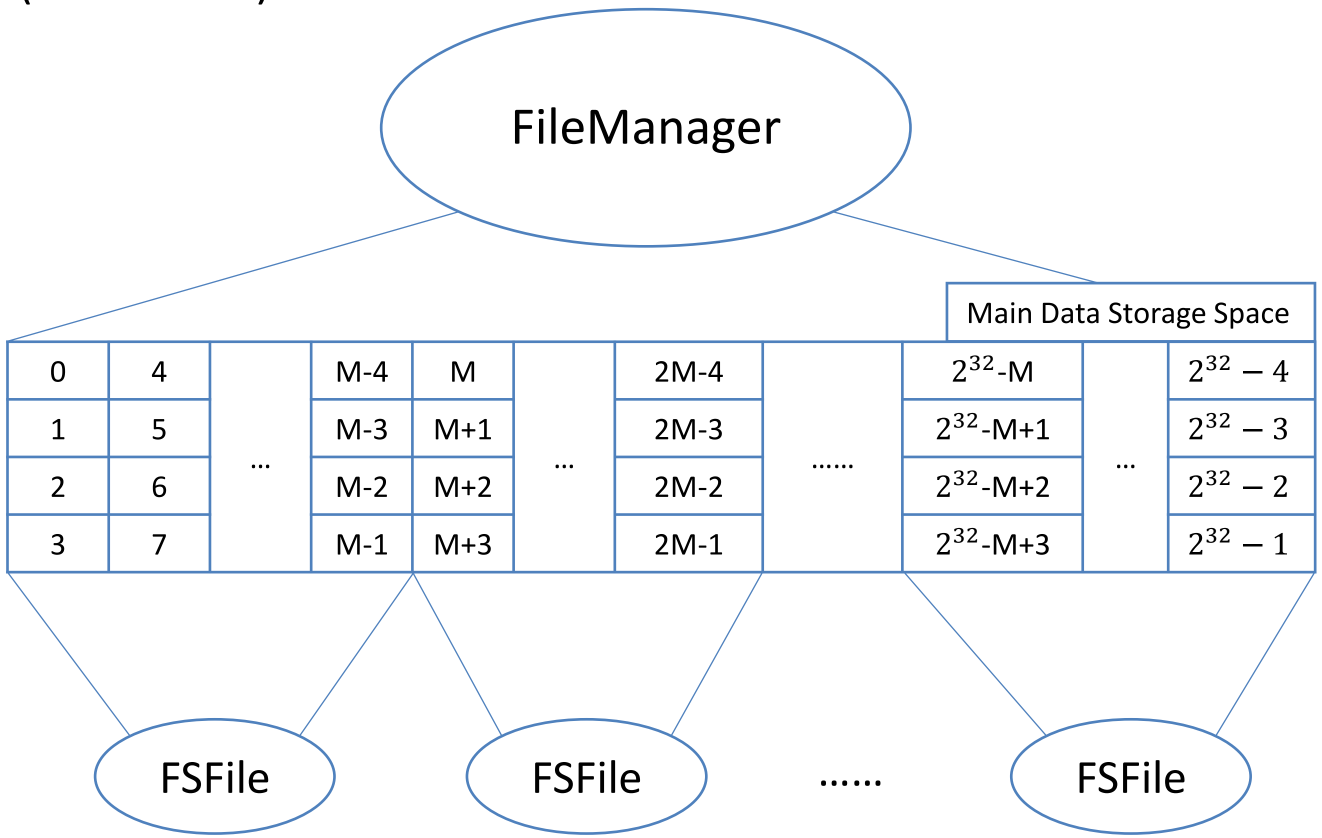
- Lowest layer of DBMS software manages space on disk
  - Disk space is usually organized in *pages*
    - which may not necessarily directly be mapped to disk sectors/file system pages!
    - common choices are 4KB, 8KB, 16KB, etc.
  - Using the OS file system or not? Some do and some don't!
  - Even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

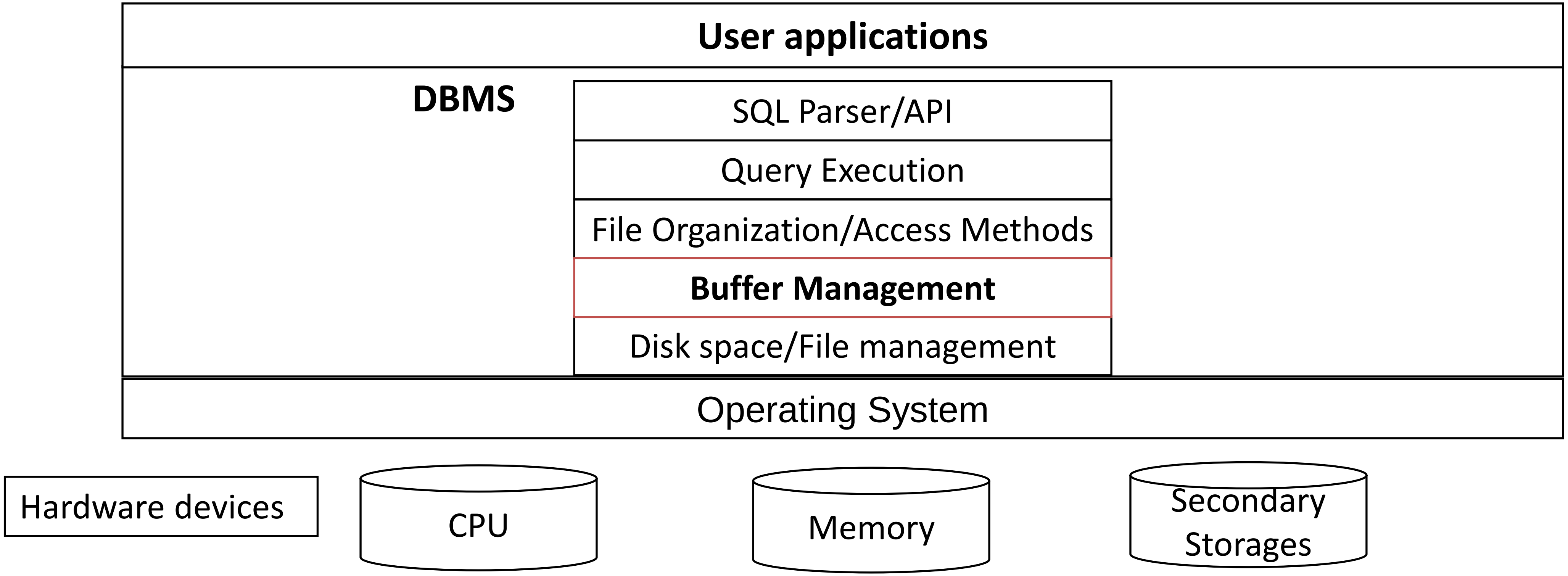
- Best if a request for a sequence of pages is satisfied by pages stored sequentially on disk!
  - Responsibility of disk space manager.
  - Higher levels don't know how this is done, or how free space is managed.
  - Though they may assume sequential access for files!
    - Hence, disk space manager 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;
  - FileManager manages many (virtual) files -- (not FSFile)
    - Each is a double-linked list of pages, allocated in groups of 64 consecutive pages
    - Each file maintains its own free list
  - Concurrency? Recovery? (to be done)



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

How does database access data pages?

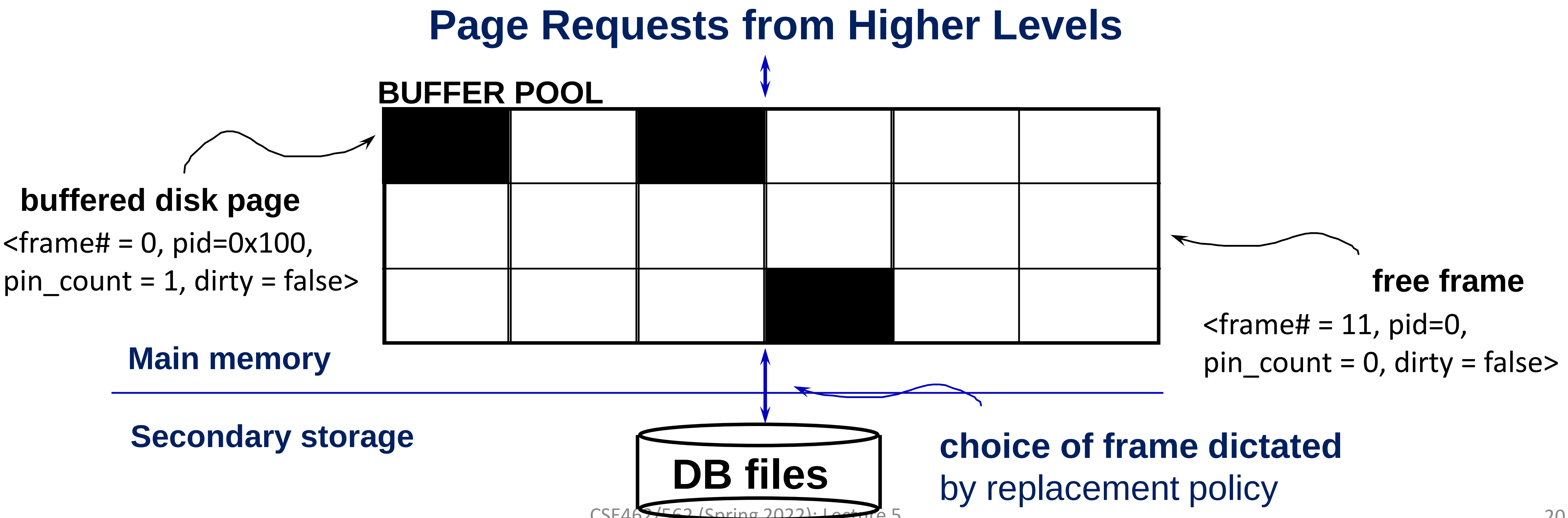
- Data pages usually need to be in main memory for DBMS to operate on it

- Suppose we want to read/write a 32-bit integer on a data page
  - Option 1: read/write the entire page before reading/writing the integer <- very slow
  - Option 2: read all data pages into memory at the beginning <- very expensive
    - May not fit in memory
    - What to do on modify?
      - Immediately write back? Or Flush when program shutsdown?
      - Data persistence?

- Solution?

Buffer management in DBMS

- Buffer manager manages a fixed-size pool of in-memory page frames which
  - are of the same size as the data pages
  - buffer data pages being read/written or to be read/written
- Meta information table contains an entry for each buffer frame:
  - <frame#, page_id, pin_count, dirty>



How to handle a page request

- Meta information table contains an entry for each buffer frame:
  - <frame#, page_id, pin_count, dirty>

- If the request page is not found in the buffer pool
  - Choose an *unpinned* frame for eviction
  - If the chosen frame is *dirty*, write it back to disk
  - Read the requested page into the chosen frame
- Then,
  - Add 1 to the pin count of the frame that has the requested page
  - Return the address to the buffer frame

- If the caller modifies the page -> must set the dirty bit
- When the caller no longer needs the page
  - Subtract 1 from the pin count of the frame that has the page

Buffer eviction policy

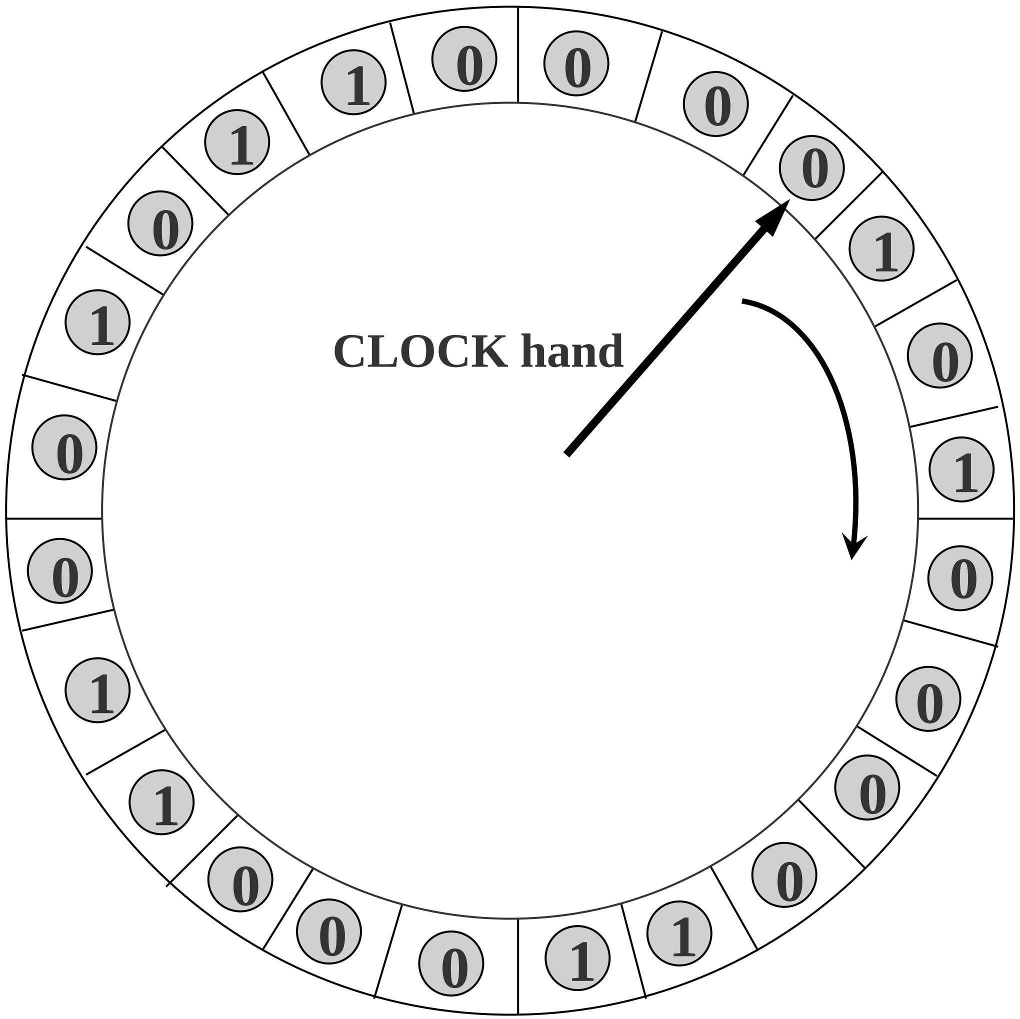
- Page eviction policy (aka replacement policy)
  - An algorithm for choosing unpinned frames when there's no free frame
  - Many choices:
    - Least recently used (LRU)
    - Most recently used (MRU)
    - Clock
    - Random
    - …
  - It can have huge impacts on the # of I/Os, depending on the access pattern

Least recently used policy

- Least Recently Used (LRU)
  - for each page in buffer pool, the order of the pages were last unpinned
  - replace the frame which has the oldest (earliest) time
  - very common policy: intuitive and simple
    - Works well for repeated accesses to popular pages -> typical transactional workload
- Problems?
  - Sequential flooding:
    - LRU + repeated sequential scans.
    - # buffer frames < # pages in file means each page request causes an I/O.
  - Idea: MRU better in this scenario?
- DB may know the access pattern before hand so that it can adapt its replacement policies
  - Switching MRU? Small ring buffer?
- How to implement?

Clock policy

- Approximate LRU
- Each buffer frame has a clock bit
  - Set upon page pinned
- When we need an eviction, move the clock hand
  - If bit is set, clear it
  - If bit is clear, evict it
  - i.e., second chance
  - Can use third/fourth chance, with a small capped count
- Why this might be faster than LRU?



DBMS vs. OS File System

OS does disk space & buffer management as well: why not let OS manage these tasks?

- Some limitations, e.g., files can't span disks.
- Buffer management in DBMS requires ability to:
  - pin a page in buffer pool, force a page to disk & order writes (important for implementing CC, concurrency control, & recovery)
  - adjust *replacement policy,* and pre-fetch pages based on access patterns in typical DB operations.

Summary

- This lecture
  - Storage hierarchy and storage devices
  - Disk space management
  - Buffer management

- Next lecture
  - File organization in DBMS