Data Storage & IO Models

CS 377: Database Systems

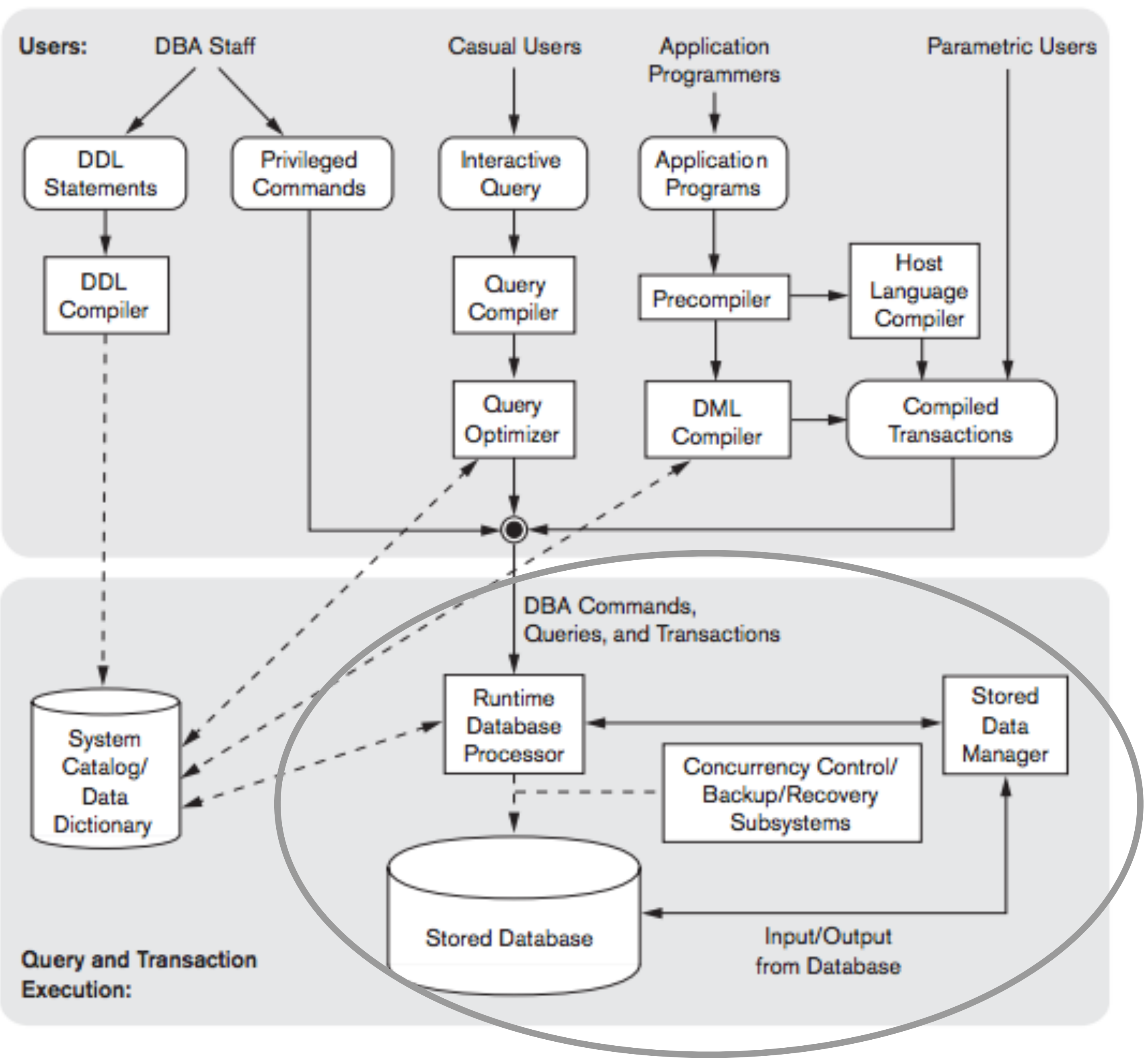
Review: Course Material to Date

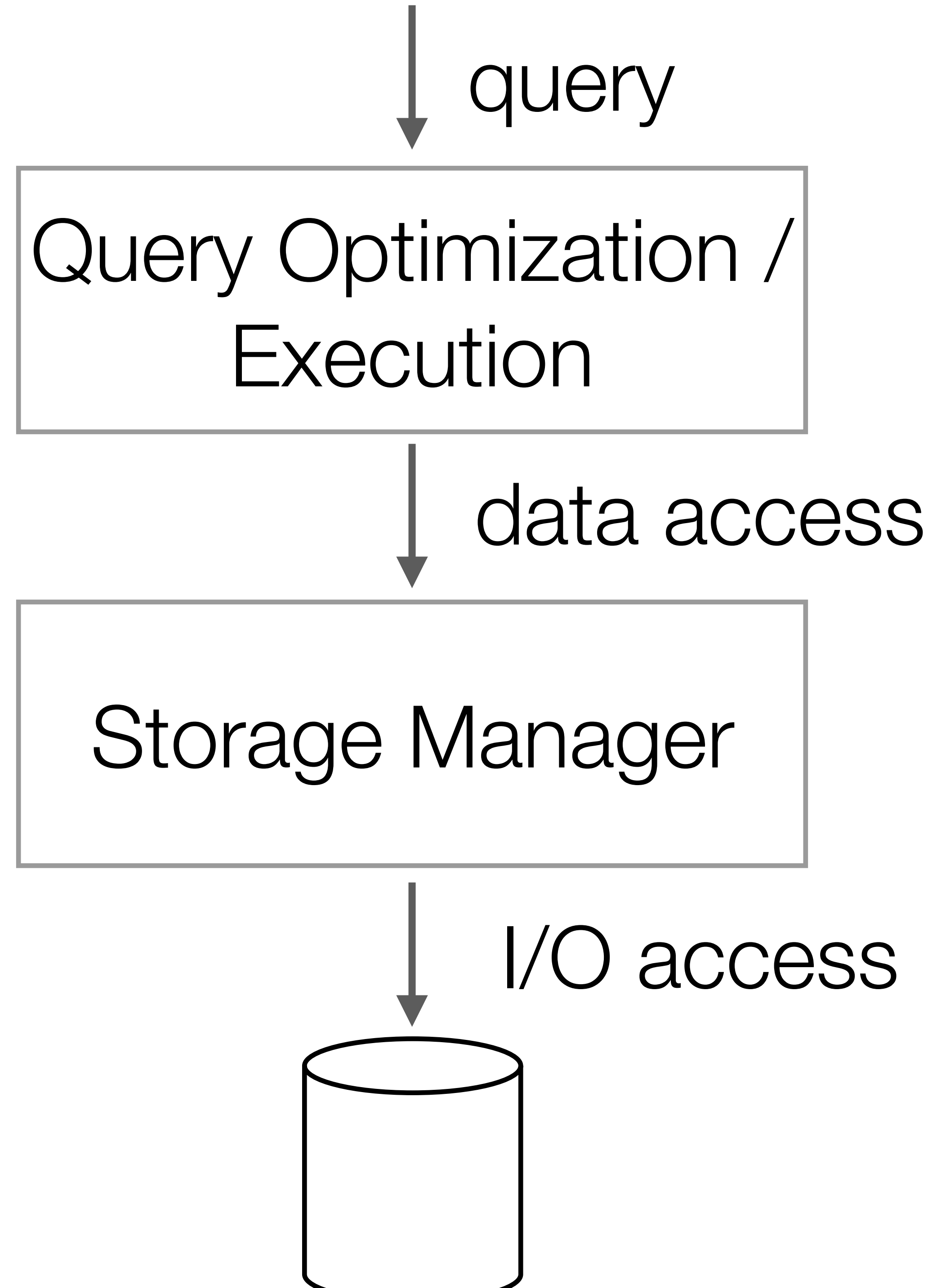
- How to use a databases and how great they are
  - Data modeling with ER
  - Relational model
  - Query languages (relational algebra, relational calculus, SQL)
  - SQL application programming
  - Database design

Context: Rest of Course

- “Peeking under the hood”
  - Where the data is located and how to keep track of them
  - How to process SQL queries
    - Why do certain queries run faster than others?
    - Why did we learn relational algebra when SQL is easier?
  - How to allow concurrency and transaction semantics

DMBS Architecture

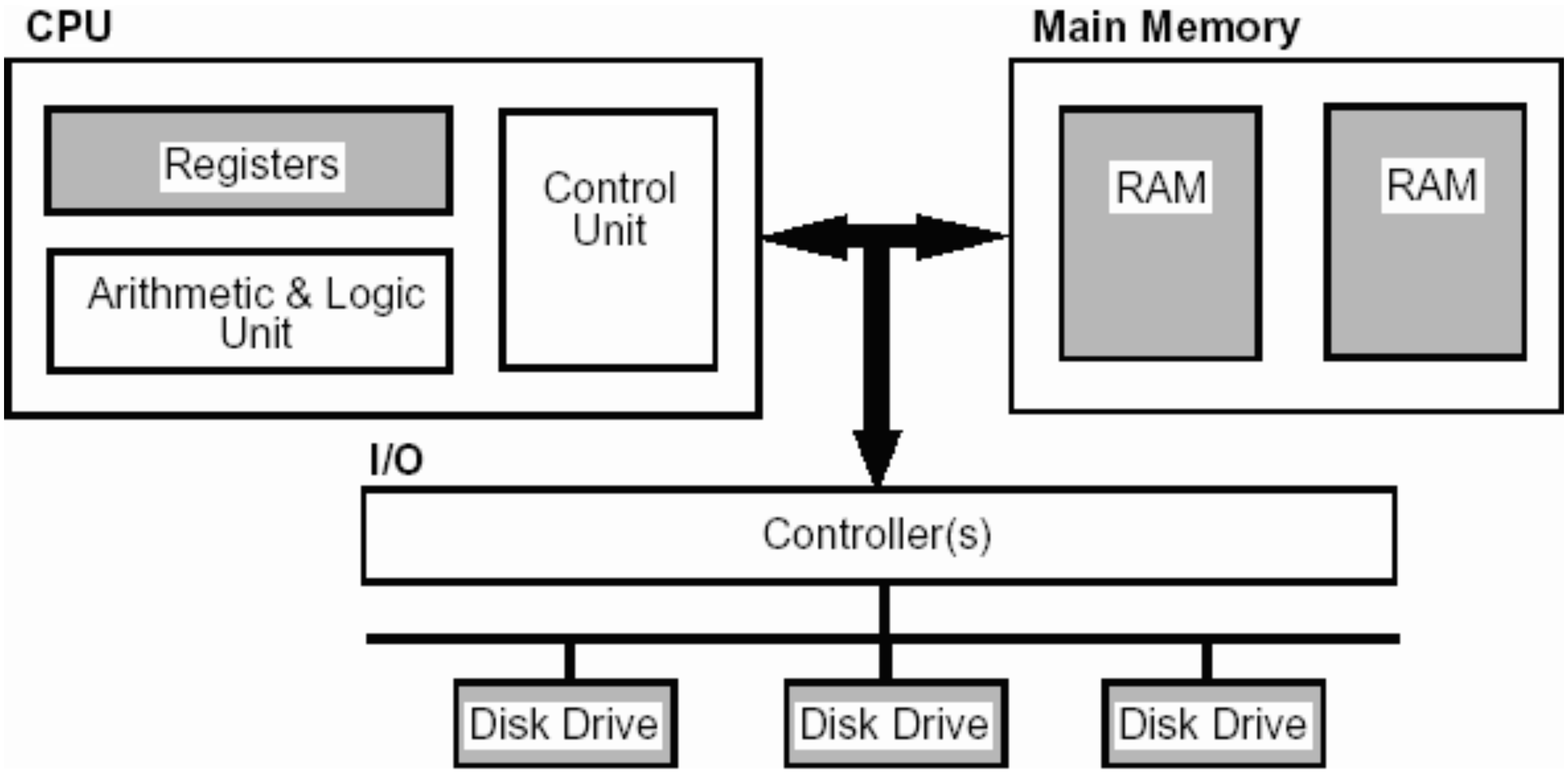




Today's Lecture

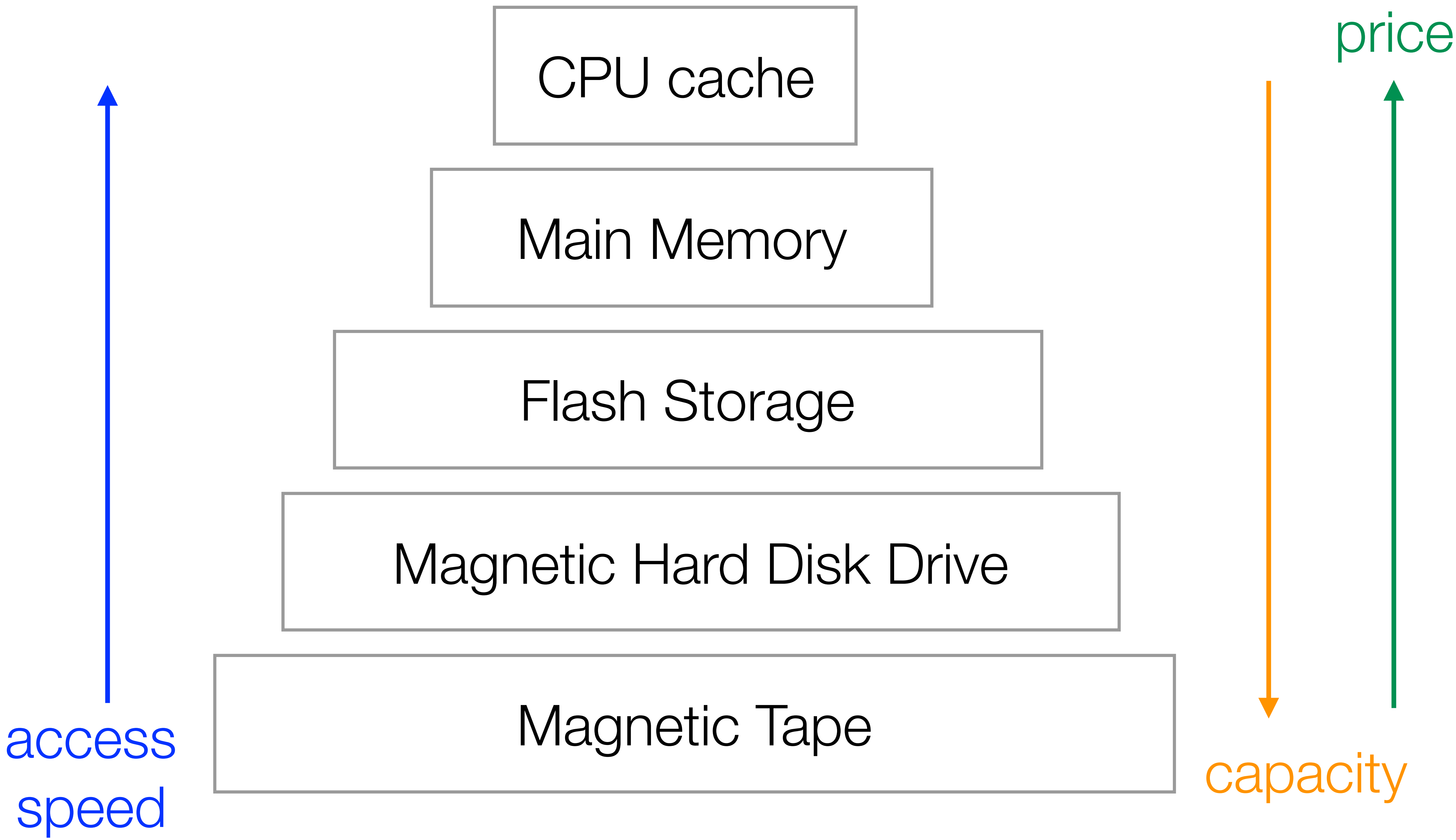
1. Typical storage hierarchy
2. File organization

Computer System Overview



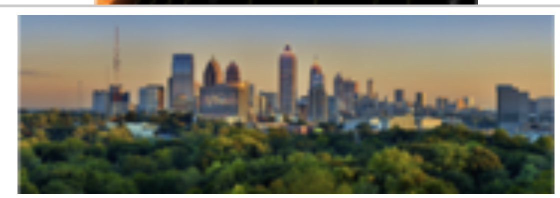
http://www.doc.ic.ac.uk/~eedwards/compsys/memory/memory.gif

Memory Hierarchy



Jim Gray's Storage Latency Analogy

How far away is the data?

10^9	Tape	Andromeda		2,000 Years
10^6	Disk	Pluto		2 years
100	Memory	Atlanta		1.5 hours
10	On board chip	This building		10 min
2	On chip cache	This room		
1	register	In my head		1 min

Typical Storage Hierarchy

- Main memory (RAM) for currently used data
- Disk for main database (secondary storage)
- Tapes for archiving older versions of the data (tertiary storage)

Why Not Just Main Memory?

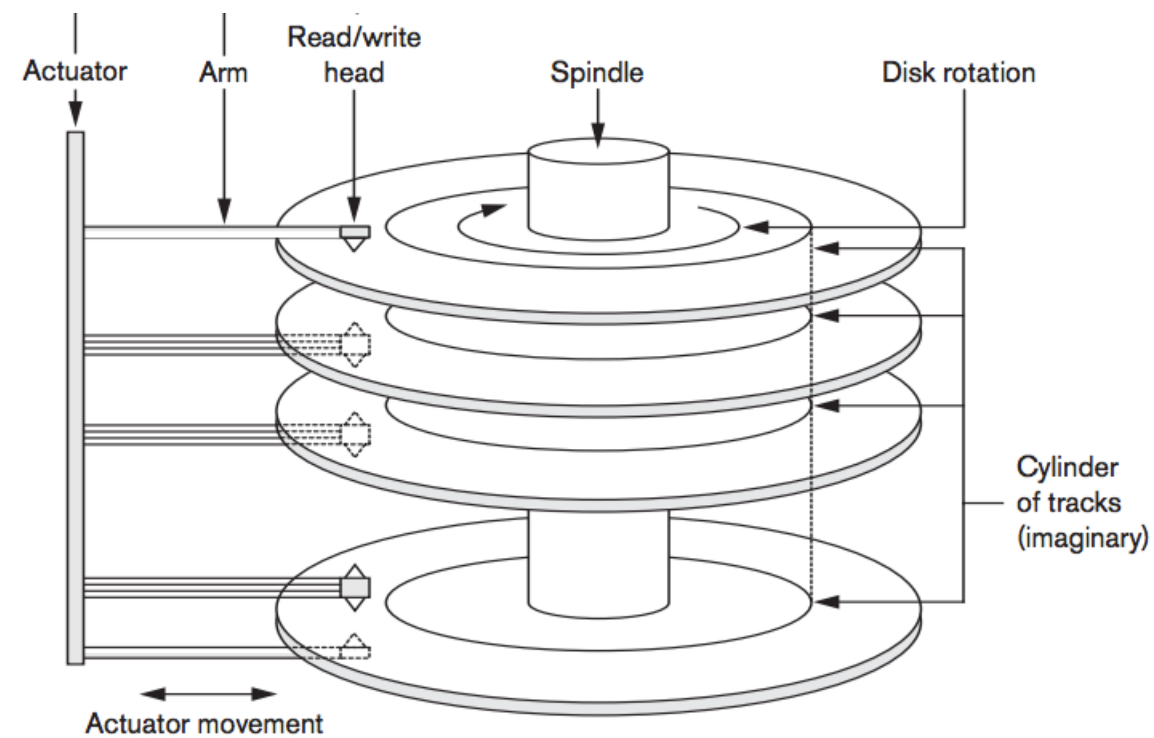
- Main memory is volatile (not persistent)
- Main memory has relatively high cost
  - $100 = 16 GB of RAM or 2 TB of disk (approximately)
  - High-end databases sit in the 10-100 TB range

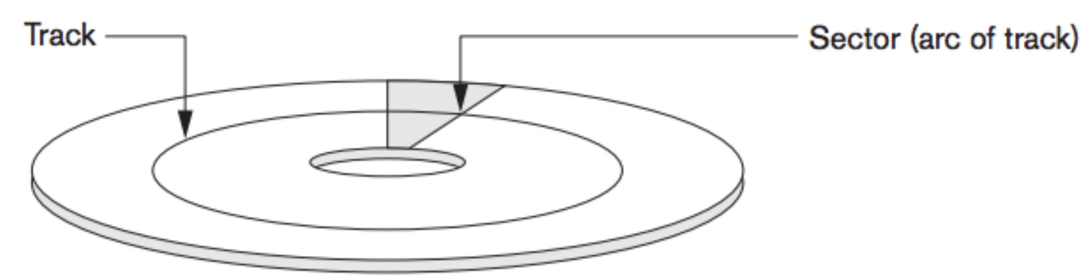
Disks & Files

- How does a DBMS store and access data?
    - Disk & main memory
- Why is this important?
    - READ/WRITE: transfer data from disk <—> main memory
    - Both are high-cost operations —> major implications on database design

Components of Hard Disk

- Data is encoded in concentric circles of sectors called tracks
- Disk head: mechanism to read / write data
- Boom (disk arm) moves to position disk head on the desired track
- Exactly one head reads/writes at any time





Hard Disks

- Data is stored and retrieved in units called disk blocks or pages
  - Typical numbers these days are 64 KB per block
- Retrieval time depends upon the location of the disk
  - Placement of blocks on disk has major impact on DBMS performance

Understanding Block Access Time

- Time to access (read/write) a page
  - Seek time: move arms to position disk head on track
  - Rotational delay: wait for page to rotate under head
  - Transfer time: move data to/from disk surface

Block Access: Dominant Factors

- Seek time and rotational delay are dominant factors
  - Seek time: ~ 0 to 10 ms
  - Rotational delay: ~ 0 to 10 ms
  - Transfer rate: ~100 MB / s

Disk Access Situations

- Random access: collection of short processes that execute in parallel, share the same disk, and cannot be predicted in advance
  - Very expensive I/O
- Sequential access: blocks are accessed in a sequence that can be predicted (e.g., accessing all the records in a single relation)
  - Much less expensive I/O

Example: Disk Specifications

	Seagate HDD
Capacity	3TB
RPM	7,200
Average Seek Time	9 ms
Max Transfer Rate	210 MB/s
# Platters	3

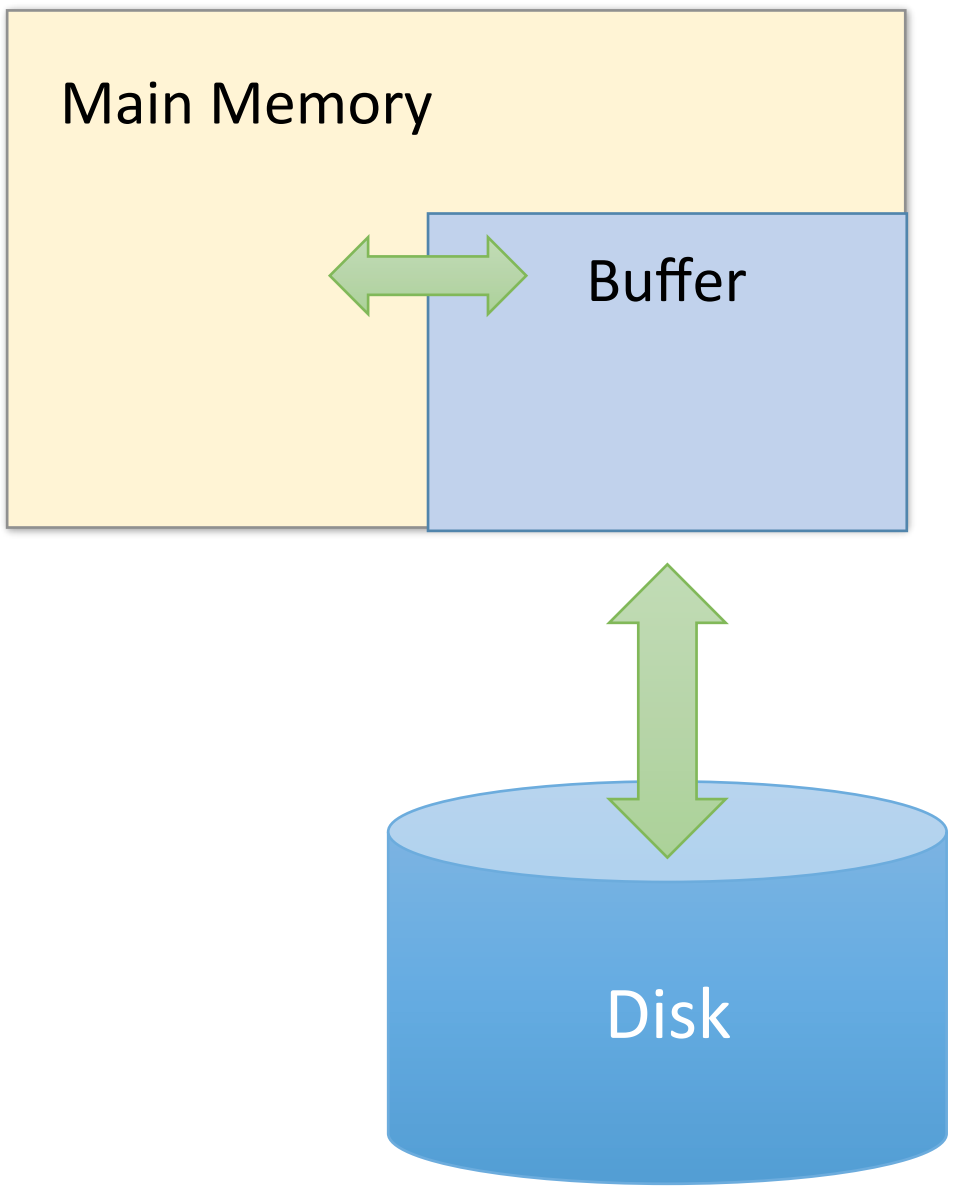
- What are I/O rates for block size of 4 KB?
  - Random workload: ~0.3 MB/s
  - Sequential workload: ~210 MB/s

Speeding up Disk Access

- Blocks in a file should be arranged sequentially on disk to minimize seek and rotational delay
- ‘Next’ block concept
  - Blocks on same track
  - Blocks on same cylinder
  - Blocks on adjacent cylinder
- For sequential scan, pre-fetch several blocks at a time!

Buffer

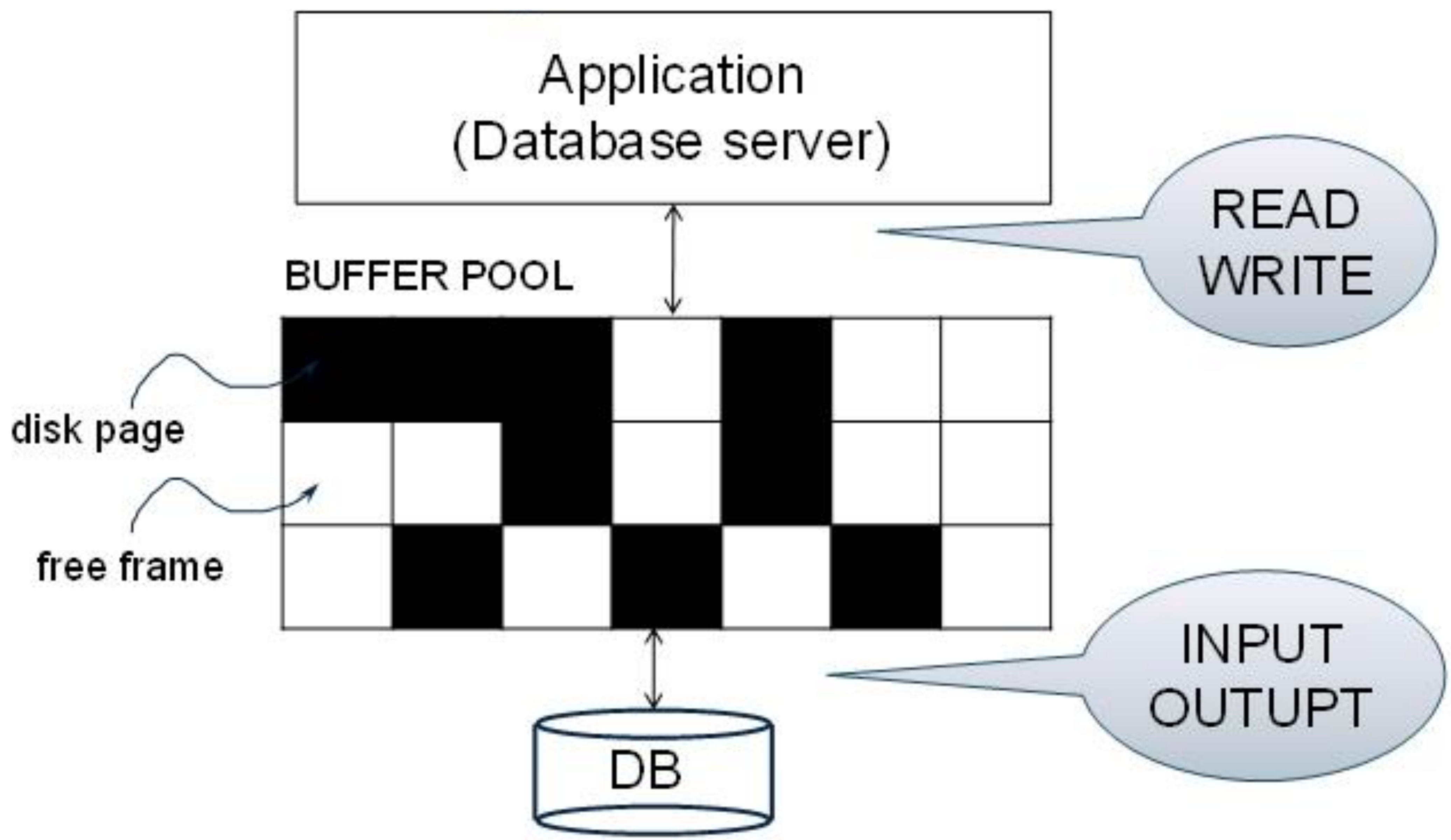
- Buffer is region of physical memory used to store temporary data
    - For purpose of this class, think of it as an intermediary between RAM and disk
- Key idea: Reading / writing to disk is slow - so prefetch / cache data



Buffer Manager

- Data should be in RAM for DBMS to operate on it efficiently
- All pages may not fit into main memory
- Buffer manager is responsible for bringing blocks from disk to main memory as needed
    - Allocate space in the buffer if not exist (replace some other block to make space for new block)
    - Reads the block from disk to buffer

Buffer Manager (Pictorially)



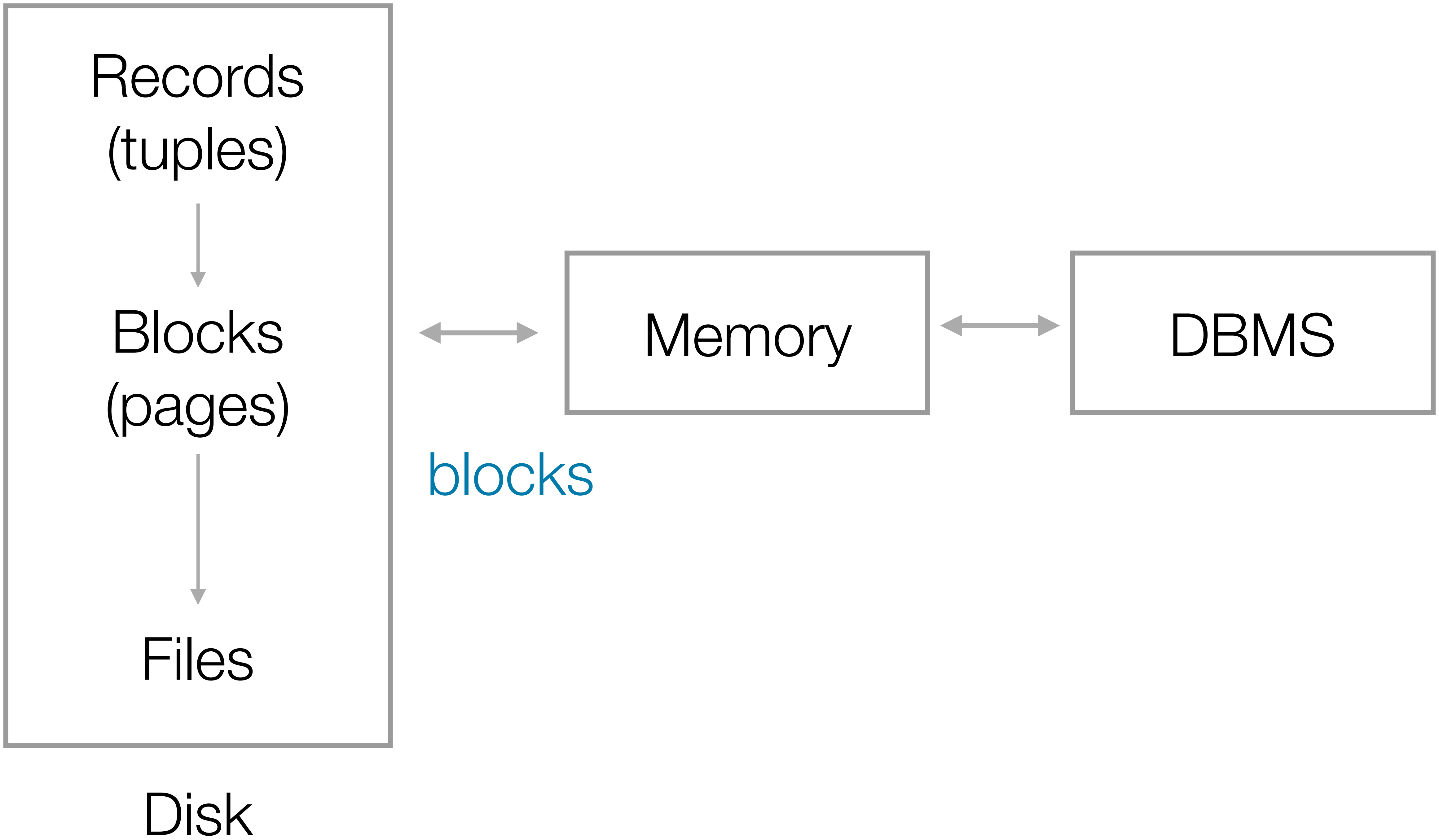
http://courses.cs.washington.edu/courses/csep544/14wi/video/archive/html5/csep544_14wi_6/slide487.jpg

DBMS vs. OS File System

Why not let OS handle disk management and buffer management?

- DBMS better at predicting reference patterns
- Buffer management is necessary to implement concurrency control and recovery
- More control of the overlap of I/O with computation
- Leverage multiple disks more effectively

Data Store Overview



Records

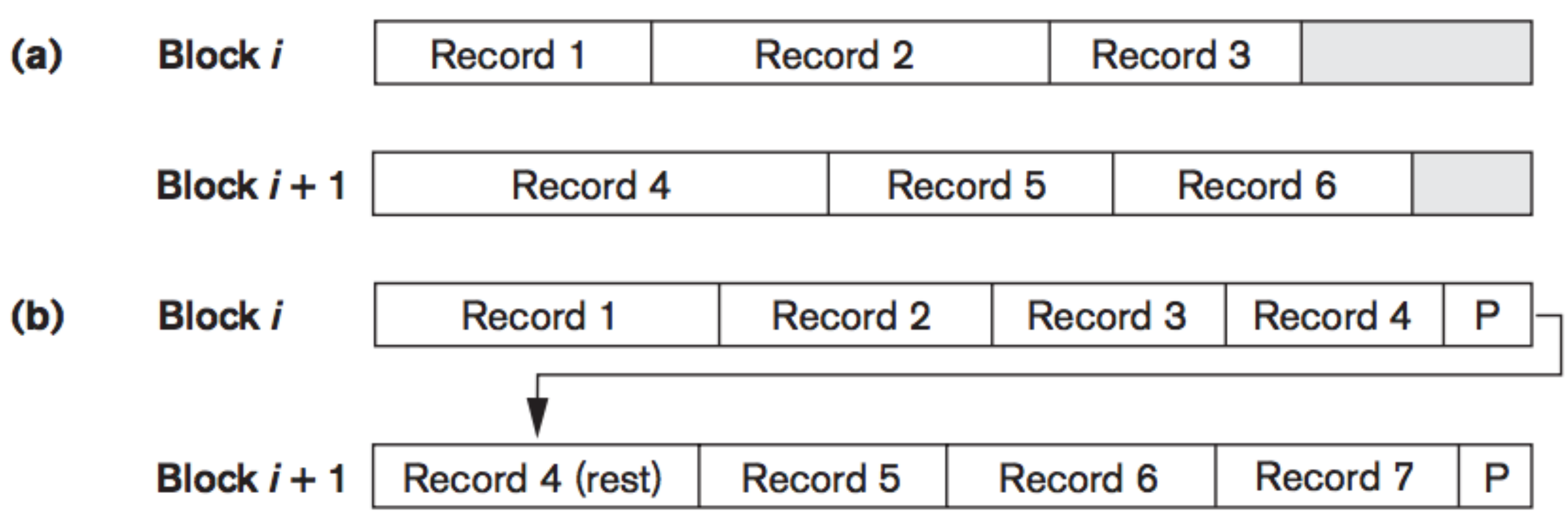
- Records contain fields which have values of a particular type (e.g., amount, date, time, age)
- Fields themselves may be fixed length or variable length

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Blocks

Blocks contain records

- Unspanned: records must be within one block, simple but can lead to unused space
- Spanned: record size can be larger than block size, pointer to rest of record



Files

- Disk space is organized into files
- Files consist of blocks (pages)
- Blocks consist of records (tuples)
- Organization of records in files
    - Heap
    - Ordered (sequential)

Unordered (Heap) Files

- Contains records in no particular order
- New records are inserted at the end of the file
- Insert: very efficient, last disk block of file is copied into buffer, add new record, and rewrite back onto disk
- Linear search: O(b)
- Reading the records in order of a particular filed requires sorting the file records

Ordered (Sequential) File

- File whose records are sorted by some attribute (usually its primary key)
- Search: binary search in $O(\log_2(b))$
- Insert: more expensive to keep records in ordered file
- Reading the records in order of the ordering field is quite efficient

	Name	Ssn	Birth_date	Job	Salary	Sex
Block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
Block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
Block 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
Block 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					
Block 5	Anderson, Zach					
	Angeli, Joe					
	⋮					
	Archer, Sue					
Block 6	Arnold, Mack					
	Arnold, Steven					
	⋮					
	Atkins, Timothy					
⋮						
Block *n*−1	Wong, James					
	Wood, Donald					
	⋮					
	Woods, Manny					
Block *n*	Wright, Pam					
	Wyatt, Charles					
	⋮					
	Zimmer, Byron					

Average Access Times

Table 17.2 Average Access Times for a File of b Blocks under Basic File Organizations

Type of Organization	Access/Search Method	Average Blocks to Access a Specific Record
Heap (unordered)	Sequential scan (linear search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary search	$\log_2 b$

External Sort-Merge

Importance of Sort Algorithms

- Data requested from DB in sorted order is extremely common
  - Example: Find students by last name or first name
- Why not just use quicksort in main memory?
  - What if we need to sort 1 TB of data with 16 GB of RAM?

Importance of Sort Algorithms

- Sorting is used for eliminating duplicate copies in a collection of records
- Sorting is needed for the ordered sequential file during bulk load
- Sort-merge join algorithm involves sorting (more on this several lectures later)

External Merge Algorithm

- How can we efficiently merge two sorted files when both are much larger than our main memory?

- Key idea: To find an element that is no larger than all elements in two lists, one only needs to compare minimum elements from each list

  - If: $A_1 \leq A_2 \leq \cdots \leq A_N$ Then: $\min(A_1, B_1) \leq A_i, \forall i$

    $B_1 \leq B_2 \leq \cdots \leq B_M$ $\min(A_1, B_1) \leq B_j, \forall j$

External Sort-Merge Algorithm

- Problem: Sort r records, stored in b file blocks with a total memory space of M blocks
- Create sorted runs with i = 0
  - Read M blocks of relation into memory
  - Sort the in-memory blocks
  - Write sorted data to run Ri, increment i

External Sort-Merge Algorithm

- Merge the sorted runs: merge subfiles until 1 remains
  - Select the first record in sort order from each of the buffers
  - Write the record to the output
  - Delete the record from the buffer page, and read the next block if empty
- Total cost: $b_r(2\lceil \log_{M-1}(b_r/M) \rceil + 1)$

Example: External Merge Sort

Sort fragments of file in memory using internal sort — where each run size is the size of the block

For this example, use block size = 3 tuples

g	24
a	19
d	31
c	33
b	14
e	16
r	16
d	21
m	3
p	2
d	7
a	14

initial relation

run 1

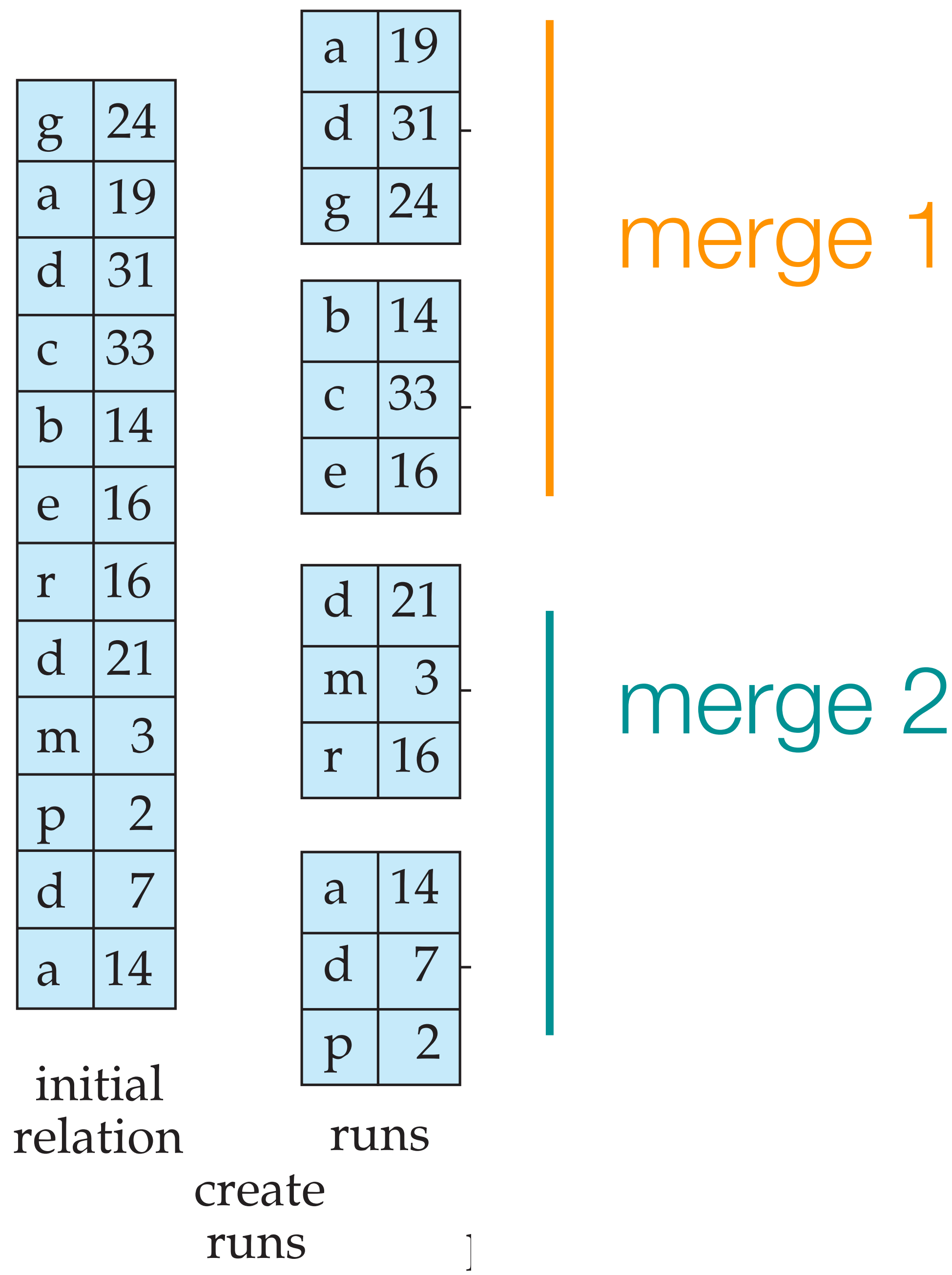
run 2

run 3

run 4

Figure 12.4 from Database System Concepts book

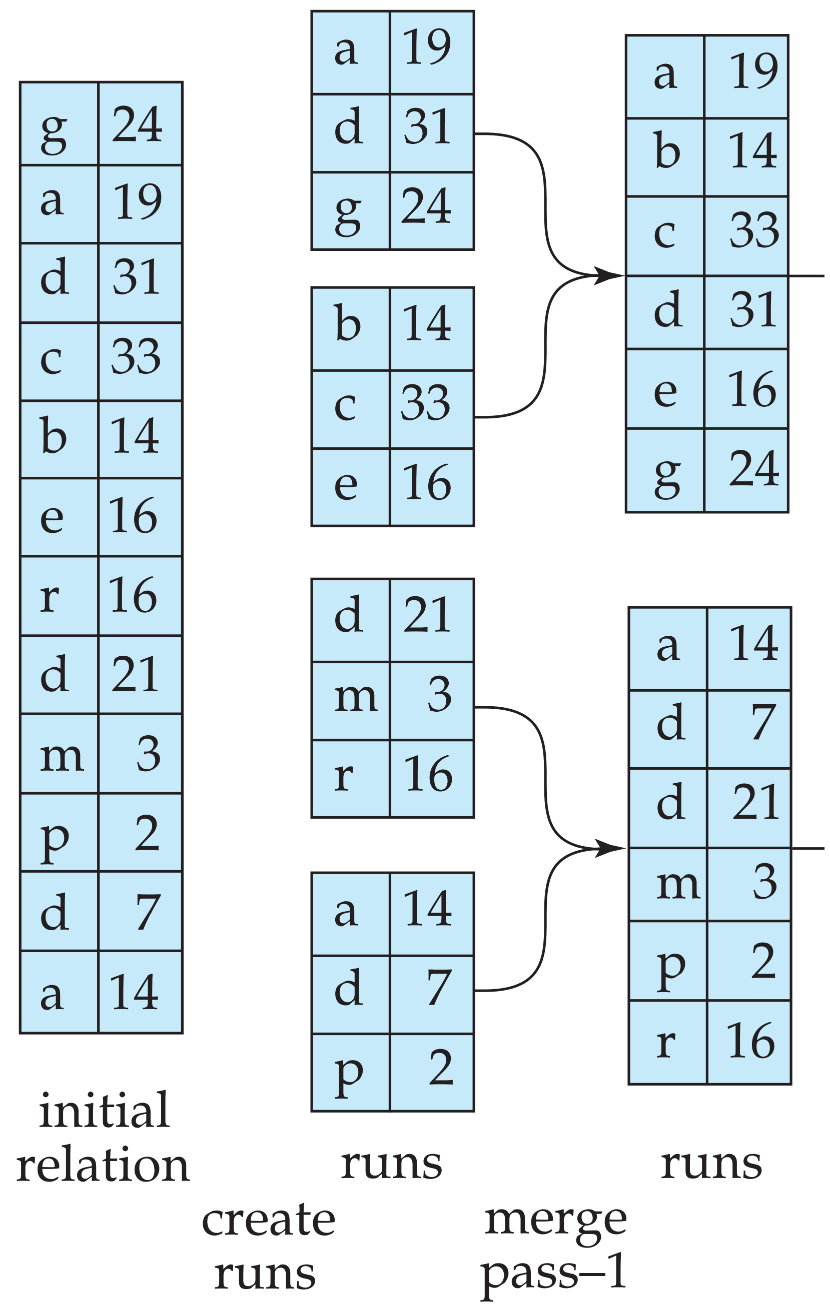
Example: External Merge Sort (2)



Once each run is sorted, we will merge two runs together at a time

Figure 12.4 from Database System Concepts book

Example: External Merge Sort (3)



Another layer of sorted runs, so again merge 2 runs at a time…

Figure 12.4 from Database System Concepts book

Example: External Merge Sort (4)

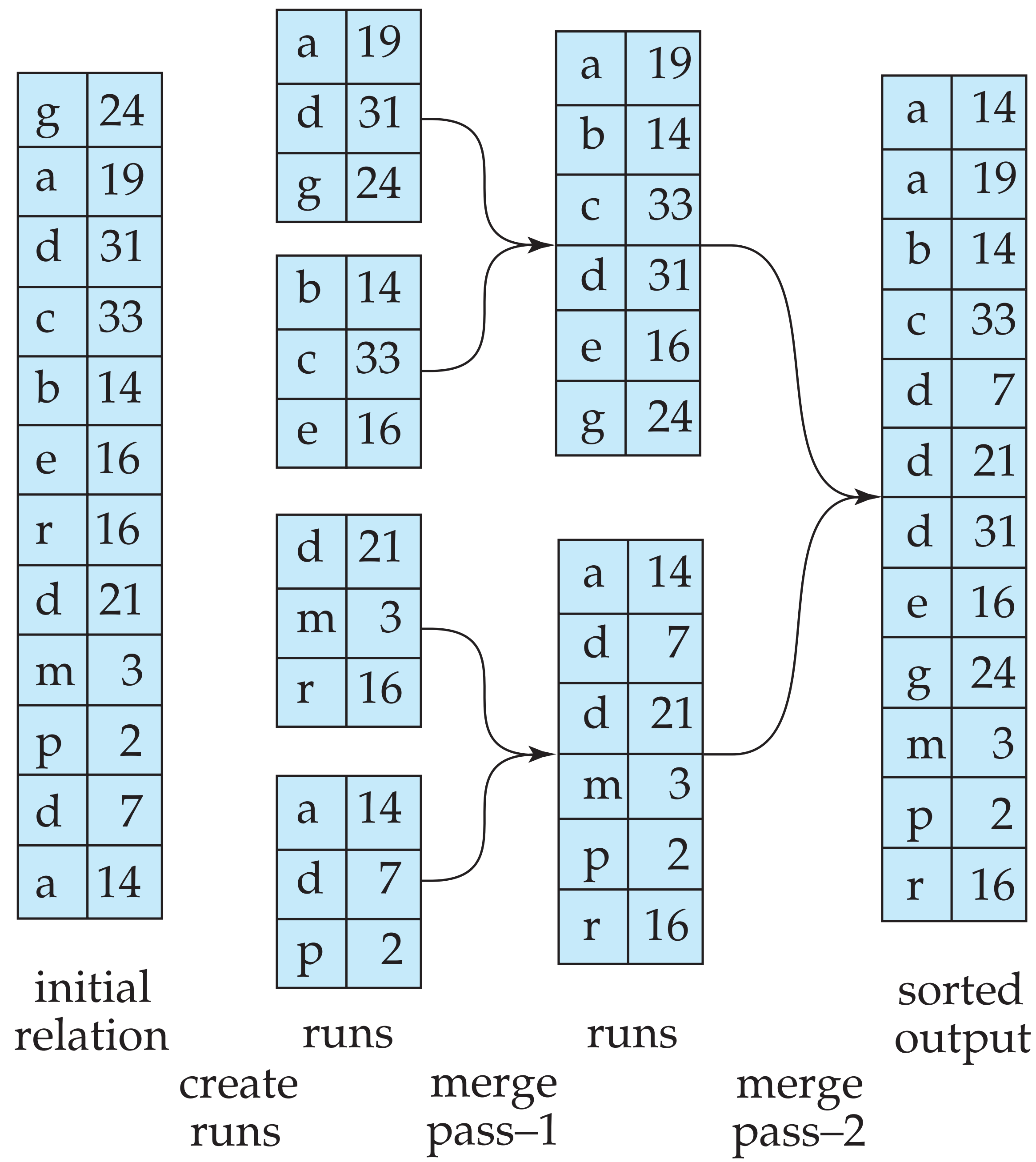


Figure 12.4 from Database System Concepts book

Data Storage: Recap

- How DBMS stores data
  - Disk, main memory
  - Files, blocks, records
  - Organization of records in files
- External Merge Sort