Logistics

Bootcamp 2 will cover C++ needed for Project 1

We need Bootcamp 1 Feedback!

Feedback Form

COMP 421: Files & Databases

Lecture 3: Storage Manager

UNC
DEPARTMENT OF
COMPUTER SCIENCE

Logistics

Project 0 Due: TONIGHT (11:59) 9/3

Project 1 Releases 9/8

Sneak Preview

- P1: Buffer Pool Manager
- P2: B+ Tree
- P3: Query Executor / Optimizer

Logistics

Bootcamp 2 RSVP (week of 9/15)

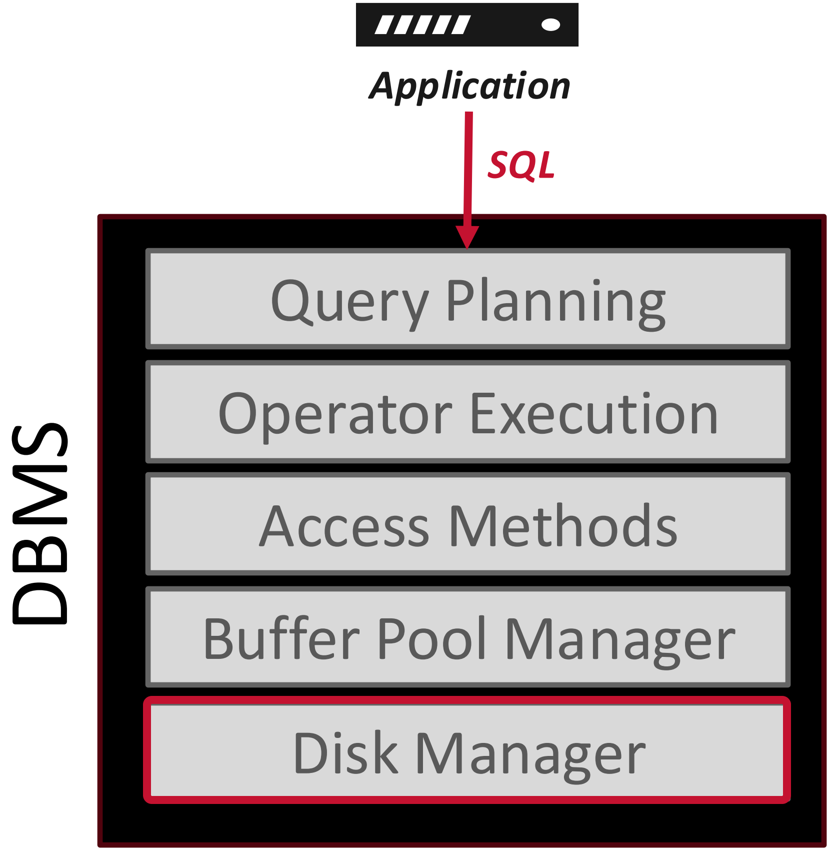
Last Class

We now understand what a database looks like at a logical level and how to write queries to read/write data (e.g., using SQL).

We will next learn how to build software that manages a database (i.e., a DBMS).

Course Outline

- Relational Databases
- ➡ Storage
- Query Execution
- Concurrency Control
- Database Recovery
- Distributed Databases
- Potpourri



Today's Agenda

Background

File Storage

Page Layout

Tuple Layout

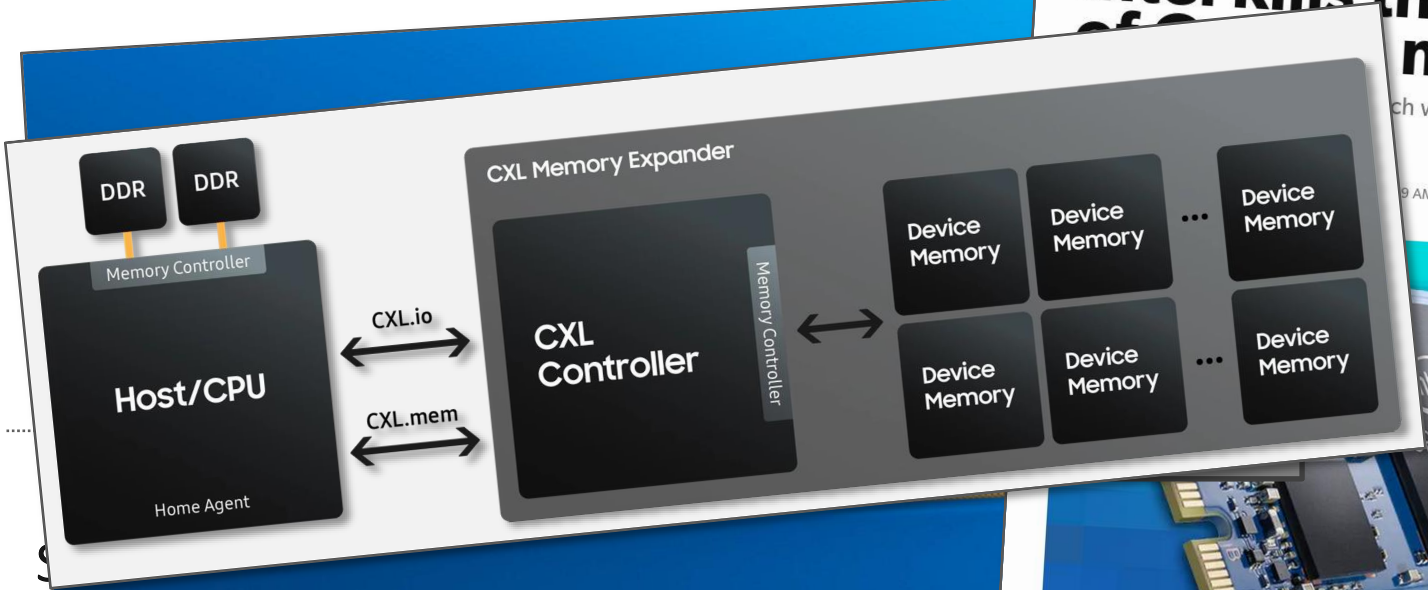
Disk-based Architecture

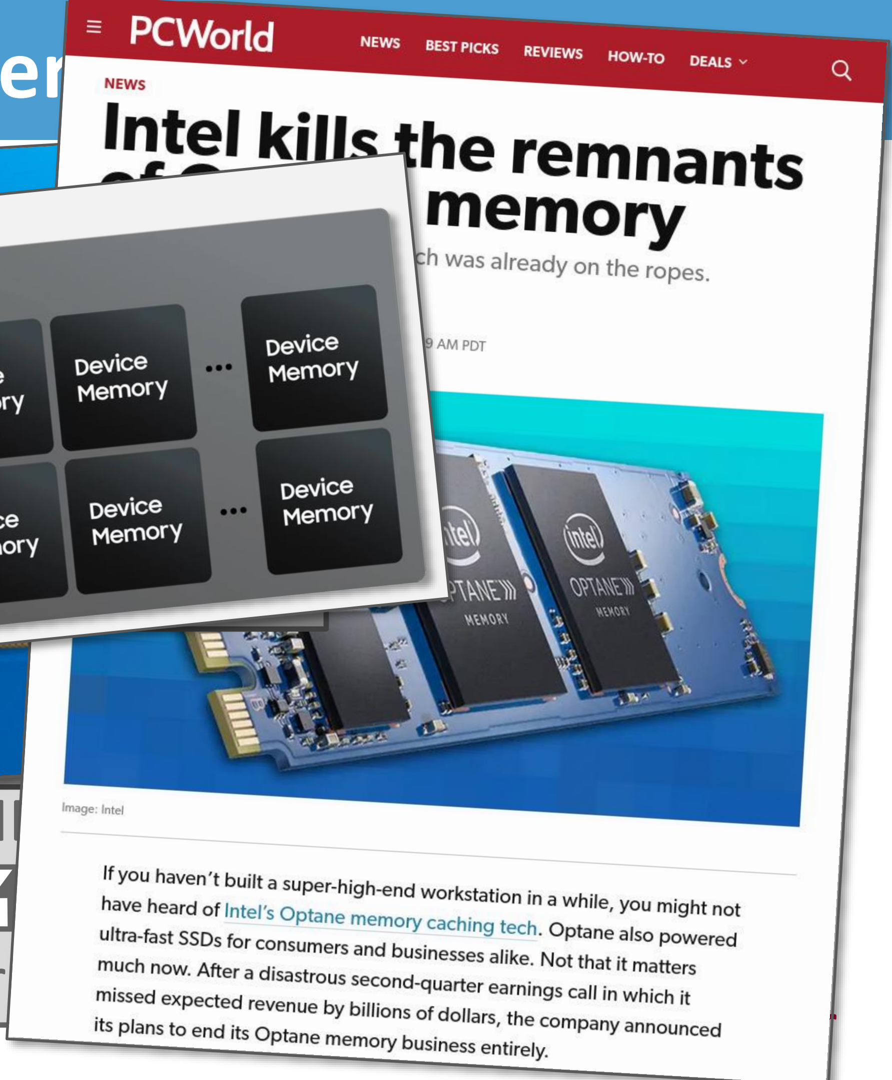
The DBMS assumes that the primary storage location of the database is on non-volatile disk.

For computation, data needs to be brought into volatile memory.

The DBMS's components manage the movement of data between storage and memory.

Storage Hier





NEWS

Intel kills the remnants of O... memory

...ch was already on the ropes.

...9 AM PDT

Image: Intel

If you haven't built a super-high-end workstation in a while, you might not have heard of Intel's Optane memory caching tech. Optane also powered ultra-fast SSDs for consumers and businesses alike. Not that it matters much now. After a disastrous second-quarter earnings call in which it missed expected revenue by billions of dollars, the company announced its plans to end its Optane memory business entirely.

S

B

Disk

HI

Networ

Access Times

Latency Numbers Every Programmer Should Know

1 ns	L1 Cache Ref	⇐	**1 sec**
4 ns	L2 Cache Ref	⇐	**4 sec**
100 ns	DRAM	⇐	**100 sec**
16,000 ns	SSD	⇐	**4.4 hours**
2,000,000 ns	HDD	⇐	**3.3 weeks**
~50,000,000 ns	Network Storage	⇐	**1.5 years**
1,000,000,000 ns	Tape Archives	⇐	**31.7 years**

Source: Colin Scott

Sequential Vs. Random Access

Random access on non-volatile storage is almost always much slower than sequential access.

DBMS will want to maximize sequential access.

→ Algorithms try to reduce number of writes to random pages so that data is stored in contiguous blocks.

→ Remember: random vs. sequential ≈100x difference!

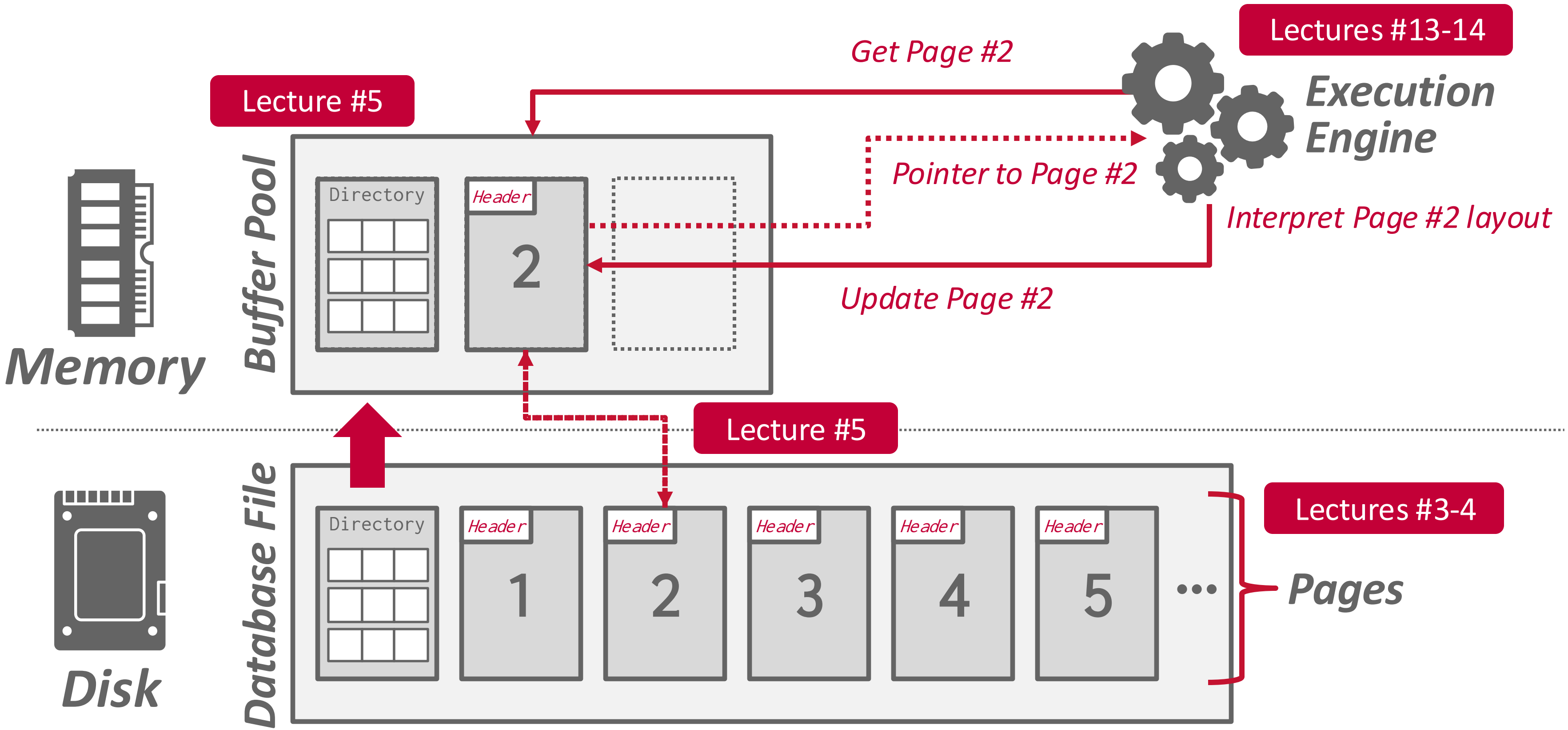
System Design Goals

Reading/writing to disk is expensive, so it must be managed carefully to avoid large stalls and performance degradation.

Random access on disk is usually much slower than sequential access, so the DBMS will want to maximize sequential access.

Allow the DBMS to manage databases that exceed the amount of memory available.

Disk-oriented Dbms

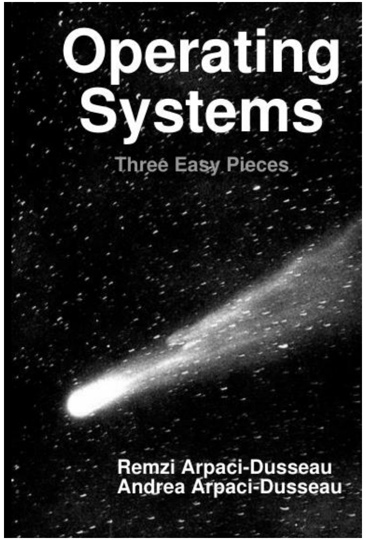


The Dark Side

Goal: Allow the DBMS to manage databases that exceed the amount of memory available.

Idea: Let database "pretend" all data fits in memory, shuffle data in the background

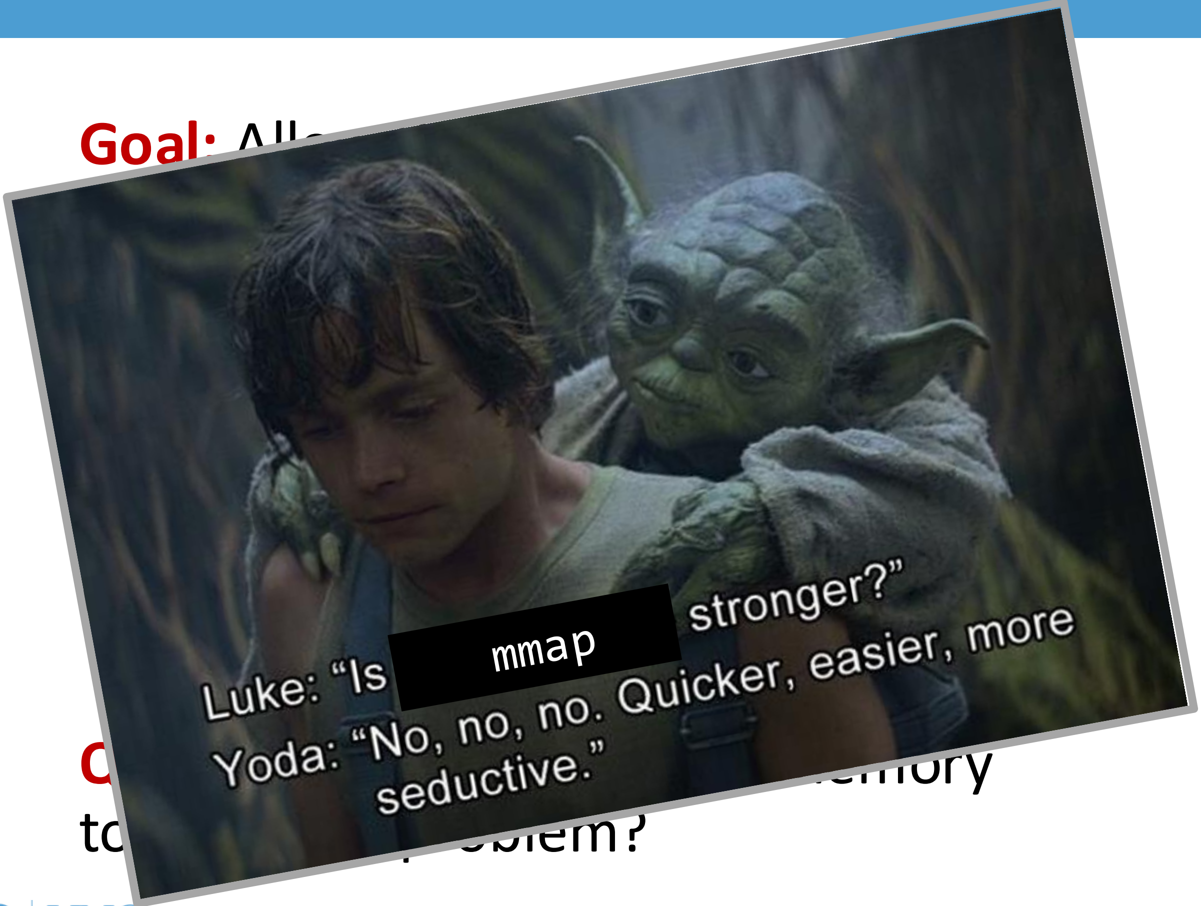
Q: ~~Can~~ Should we leverage virtual memory to solve our problem?



A: Technically, yes, using `mmap`

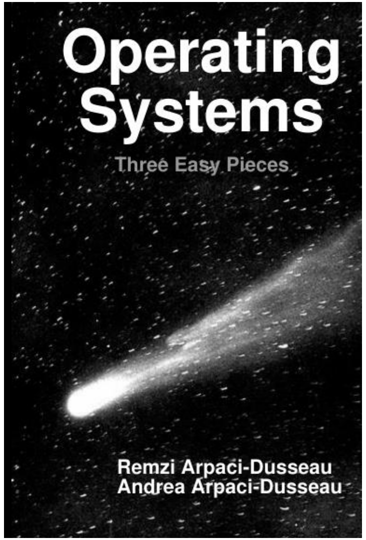
The Dark Side

Goal: All…



Q… …emory

to… …problem?



A: Technically, yes, using mmap

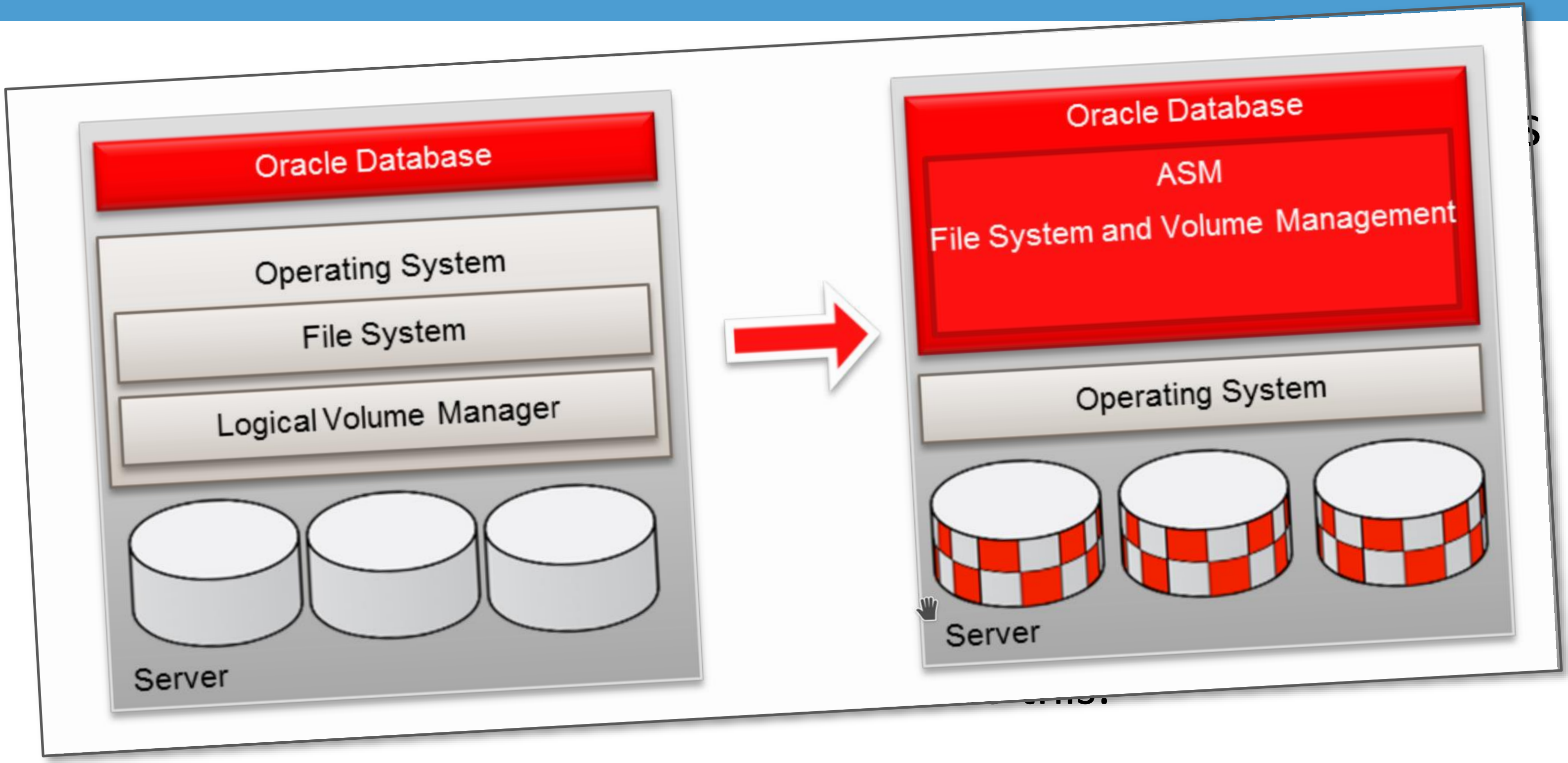
Database Storage

Problem #1: How the DBMS represents the database in files on disk.

← **Today**

Problem #2: How the DBMS manages its memory and moves data back-and-forth from disk.

File Storage



Storage Manager

The storage manager is responsible for maintaining a database's files.

→ Some do their own scheduling for reads and writes to improve spatial and temporal locality of pages.

It organizes the files as a collection of pages.

→ Tracks data read/written to pages.

→ Tracks the available space.

A DBMS typically does not maintain multiple copies of a page on disk.

→ Assume this happens above/below storage manager.

Database Pages

A <u>page</u> is a fixed-size block of data.

→ It can contain tuples, meta-data, indexes, log records…

→ Most systems do not mix page types.

→ Some systems require a page to be self-contained.

Each page is given a unique identifier (**<u>page ID</u>**).

→ A page ID could be unique per DBMS instance, per database, or per table.

→ The DBMS uses an indirection layer to map page IDs to physical locations.

Database Pages

There are three different notions of "pages" in a DBMS:

→ Hardware Page (usually 4KB)
→ OS Page (usually 4KB, x64 2MB/1GB)
→ Database Page (512B-32KB)

A hardware page is the largest block of data that the storage device can guarantee failsafe writes.

DBMSs that specialize in read-only workloads have larger page sizes.

Default DB Page Sizes

4KB: SQLite, ORACLE, IBM DB2, RocksDB, WIREDTIGER

8KB: Microsoft SQL Server, PostgreSQL

16KB: MySQL

Page Storage Architecture

Different DBMSs manage pages in files on disk in different ways.

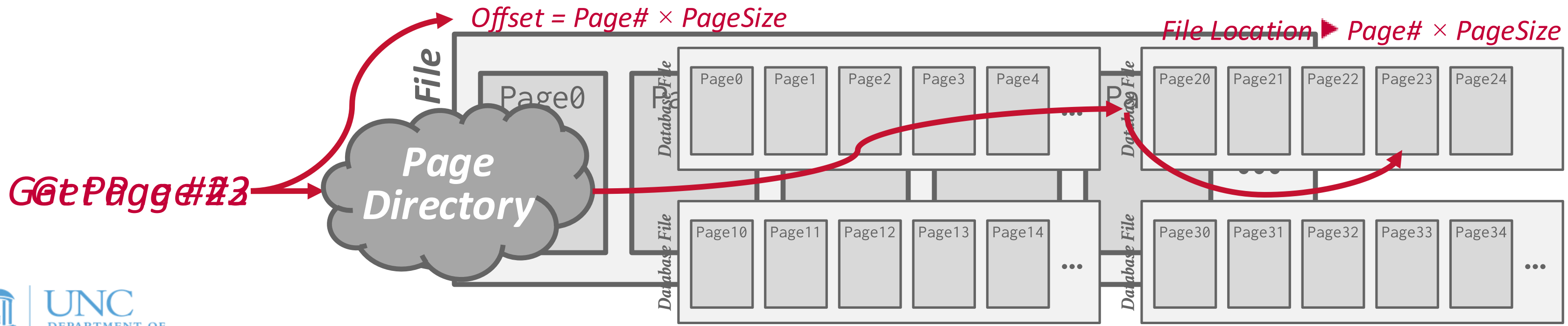
→ Heap File Organization
→ Tree File Organization
→ Sequential / Sorted File Organization (ISAM)
→ Hashing File Organization

At this point in the hierarchy, we do <u>not</u> need to know anything about what is inside of the pages.

Heap File

A heap file is an unordered collection of pages with tuples that are stored in random order.
→ Create / Get / Write / Delete Page
→ Must also support iterating over all pages.

Need additional meta-data to track location of files and free space availability.



Heap File: Page Directory

The DBMS maintains special pages that tracks the location of data pages in the database files.

→ One entry per database object.

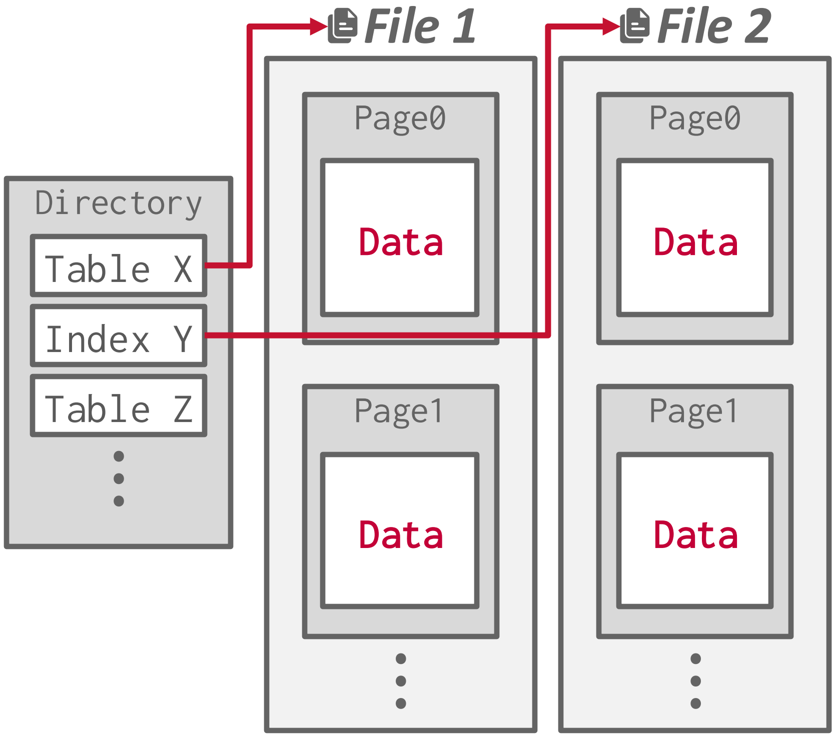
→ Must make sure that the directory pages are in sync with the data pages.

DBMS also keeps meta-data about pages' contents:

→ Amount of free space per page.

→ List of free / empty pages.

→ Page type (data vs. meta-data).



Today's Agenda

File Storage

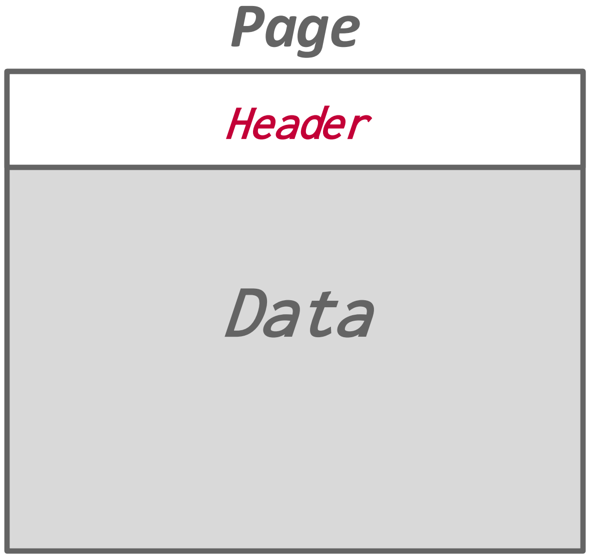
Page Layout

Tuple Layout

Page Header

Every page contains a header of meta-data about the page's contents.

→ Page Size
→ Checksum
→ DBMS Version
→ Transaction Visibility
→ Compression / Encoding Meta-data
→ Schema Information
→ Data Summary / Sketches



Some systems require pages to be self-contained (e.g., Oracle).

Page Layout

For any page storage architecture, we now need to decide how to organize the data inside of the page.

→ We are still assuming that we are only storing tuples in a row-oriented storage model.

Lecture #6

Approach #1: Tuple-oriented Storage ← **Today**

Approach #2: Log-structured Storage

Approach #3: Index-organized Storage

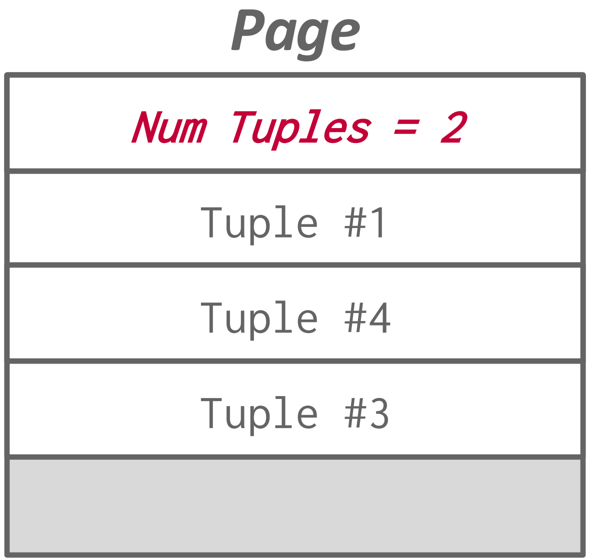
Lecture #4

Tuple-oriented Storage

How to store tuples in a page?

Strawman Idea: Keep track of the number of tuples in a page and then just append a new tuple to the end.

→ What happens if we delete a tuple?
→ What happens if we have a variable-length attribute?

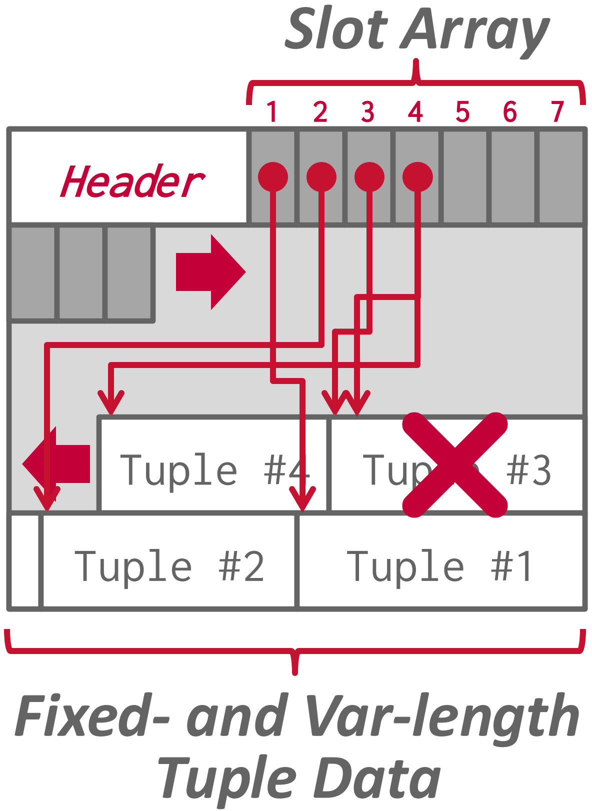


Slotted Pages

The most common layout scheme is called slotted pages.

The slot array maps "slots" to the tuples' starting position offsets.

The header keeps track of:
→ The # of used slots
→ The offset of the starting location of the last slot used.



Record Ids

The DBMS assigns each logical tuple a unique record identifier that represents its physical location in the database.

→ File Id, Page Id, Slot #
→ Most DBMSs do not store ids in tuple.
→ SQLite uses ROWID as the true primary key and stores them as a hidden attribute.

Applications should never rely on these IDs to mean anything.

PostgreSQL
CTID (6-bytes)

SQLite
ROWID (8-bytes)

Microsoft SQL Server
%%physloc%% (8-bytes)

ORACLE
ROWID (10-bytes)

Today's Agenda

File Storage

Page Layout

Tuple Layout

Tuple Layout

A tuple is essentially a sequence of bytes.
→ These bytes do not have to be contiguous.

It is the job of the DBMS to interpret those bytes into attribute types and values.

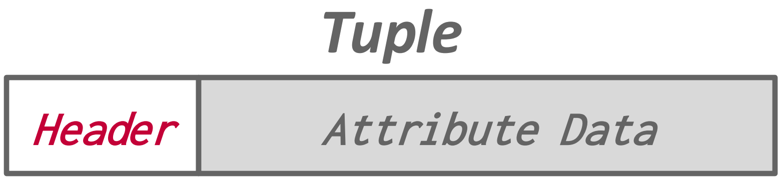
Tuple Header

Each tuple is prefixed with a header that contains meta-data about it.

→ Visibility info (concurrency control)

→ Bit Map for `NULL` values.

We do not need to store meta-data about the schema.

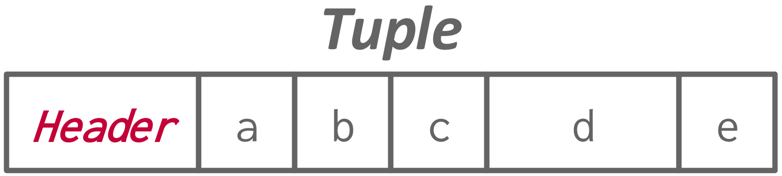


Tuple Data

Attributes are typically stored in the order that you specify them when you create the table.

This is done for software engineering reasons (i.e., simplicity).

However, it might be more efficient to lay them out differently.




```
CREATE TABLE foo (
  a INT PRIMARY KEY,
  b INT NOT NULL,
  c INT,
  d DOUBLE,
  e FLOAT
);
```

Denormalized Tuple Data

DBMS can physically ***denormalize*** (e.g., "pre-join") related tuples and store them together in the same page.

→ Potentially reduces the amount of I/O for common workload patterns.

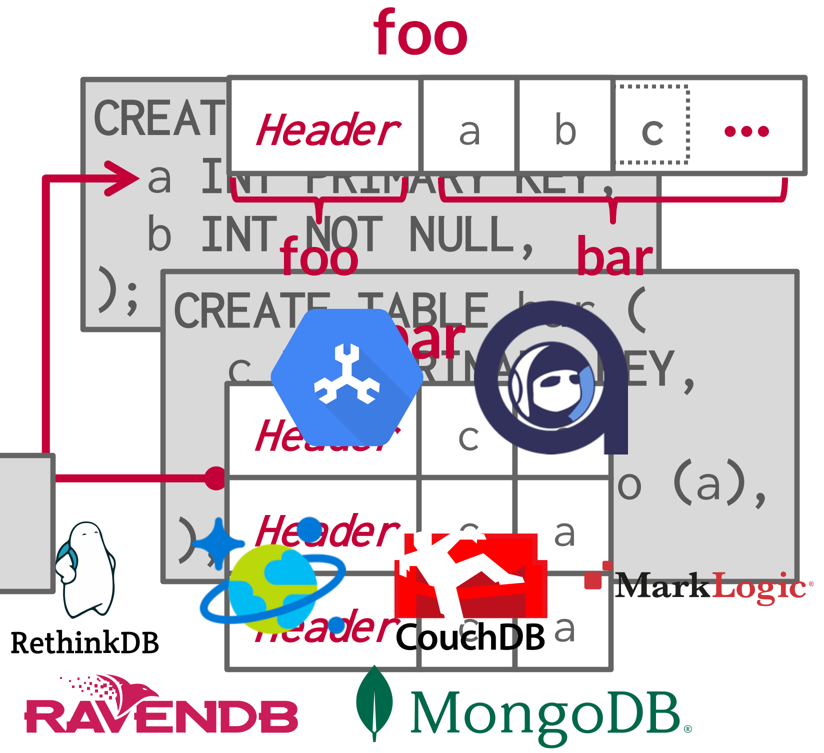
→ Can make updates more expensive.

```
SELECT * FROM foo JOIN bar
    ON foo.a = bar.a;
```

Not a new

→ IBM System R did this in the 1970s.

→ Several NoSQL DBMSs do this without calling it physical denormalization.



Conclusion

Database is organized in pages.

Different ways to track pages.

Different ways to store pages.

Different ways to store tuples.

Next Class

Log-Structured Storage

Index-Organized Storage

Value Representation

Catalogs