Carnegie Mellon University

03 Database Storage – Part I

Intro to Database Systems
15-445/15-645
Fall 2020

AP
Andy Pavlo
Computer Science
Carnegie Mellon University

ADMINISTRIVIA

Homework #1 is due September 13th @ 11:59pm

Project #0 is due September 13th @ 11:59pm

Project #1 will be released on September 14th

OVERVIEW

We now understand what a database looks like at a logical level and how to write queries to read/write data from it.

We will next learn how to build software that manages a database.

COURSE OUTLINE

Relational Databases

Storage

Execution

Concurrency Control

Recovery

Distributed Databases

Potpourri

Query Planning

Operator Execution

Access Methods
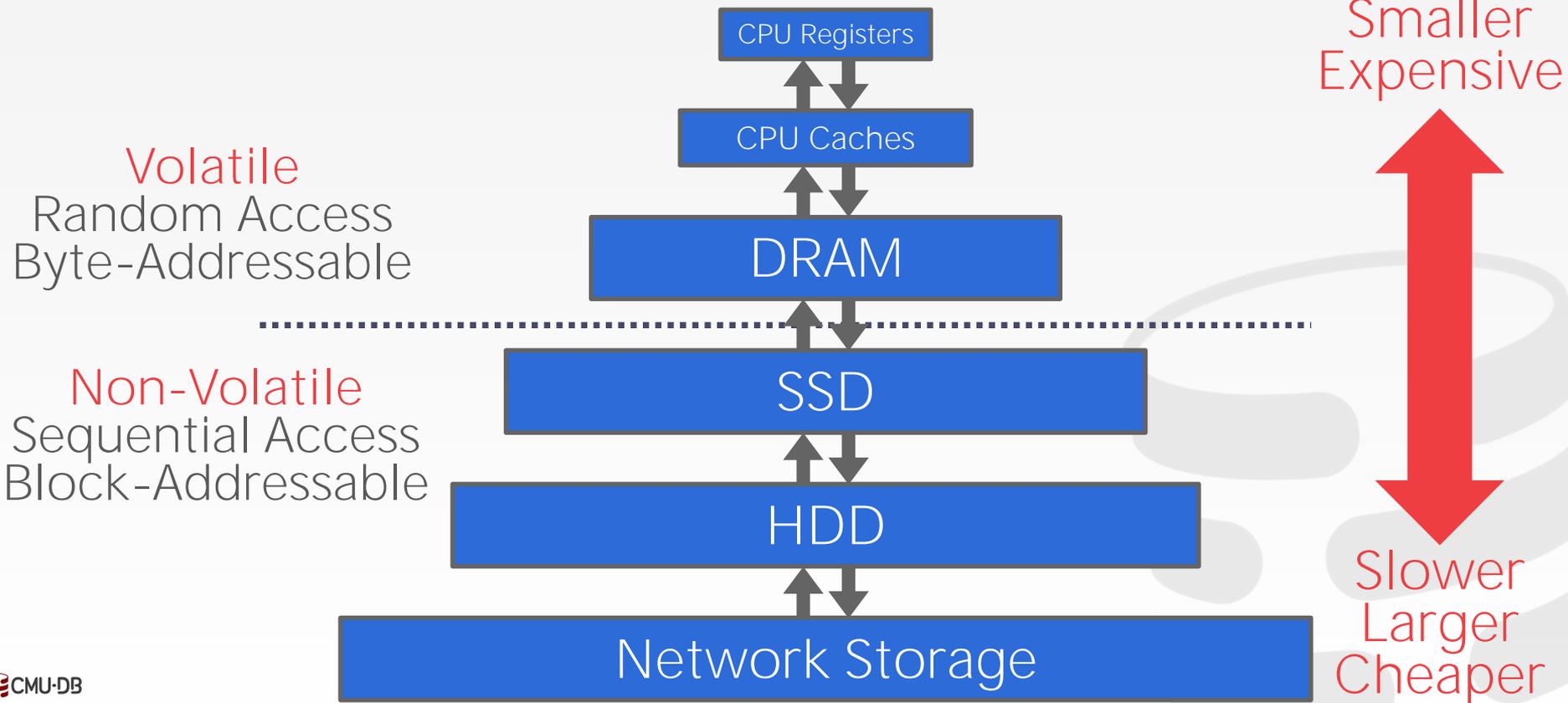
Buffer Pool Manager

Disk Manager

DISK-ORIENTED ARCHITECTURE

The DBMS assumes that the primary storage location of the database is on non-volatile disk.
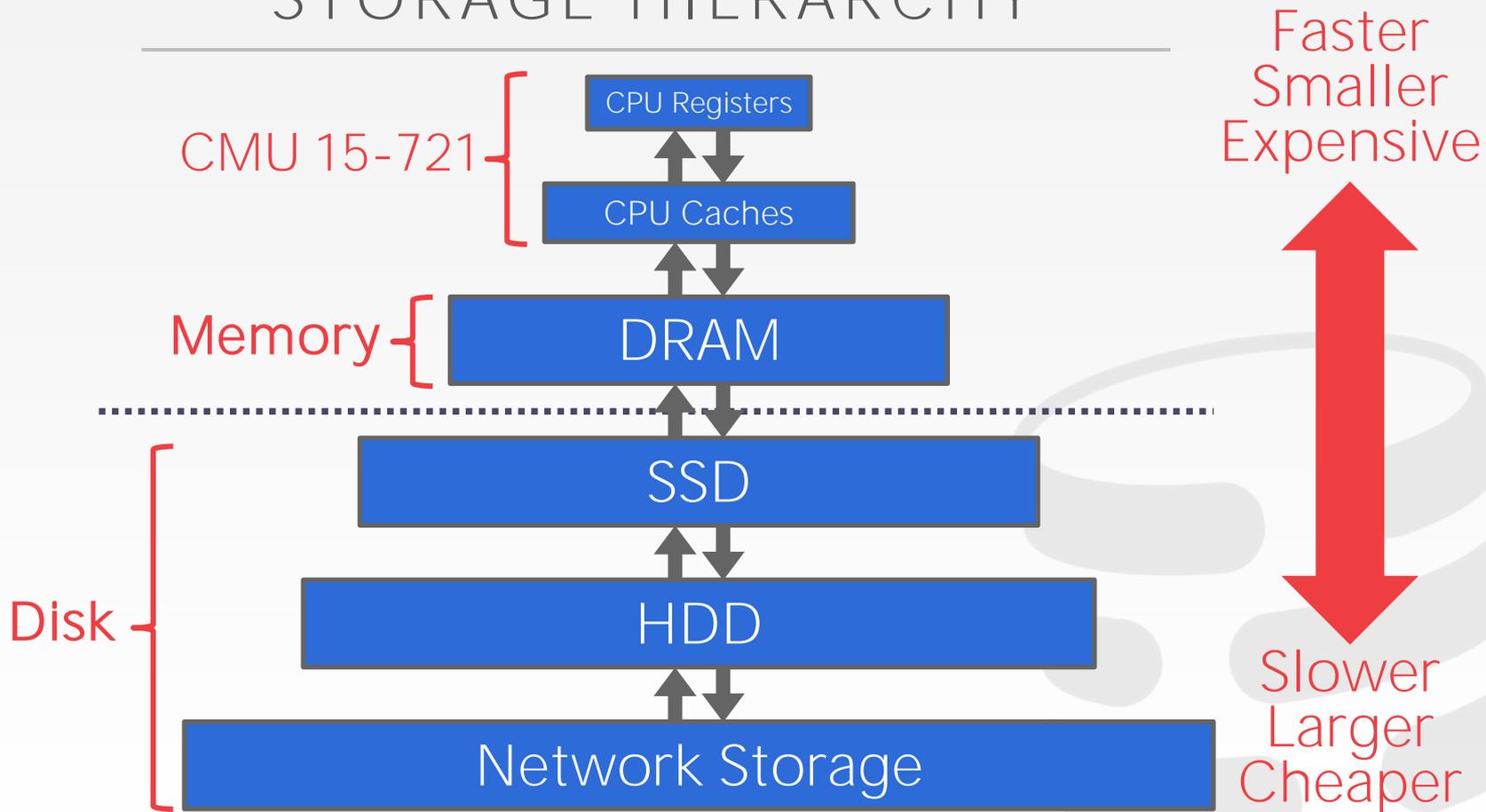
The DBMS's components manage the movement of data between non-volatile and volatile storage.
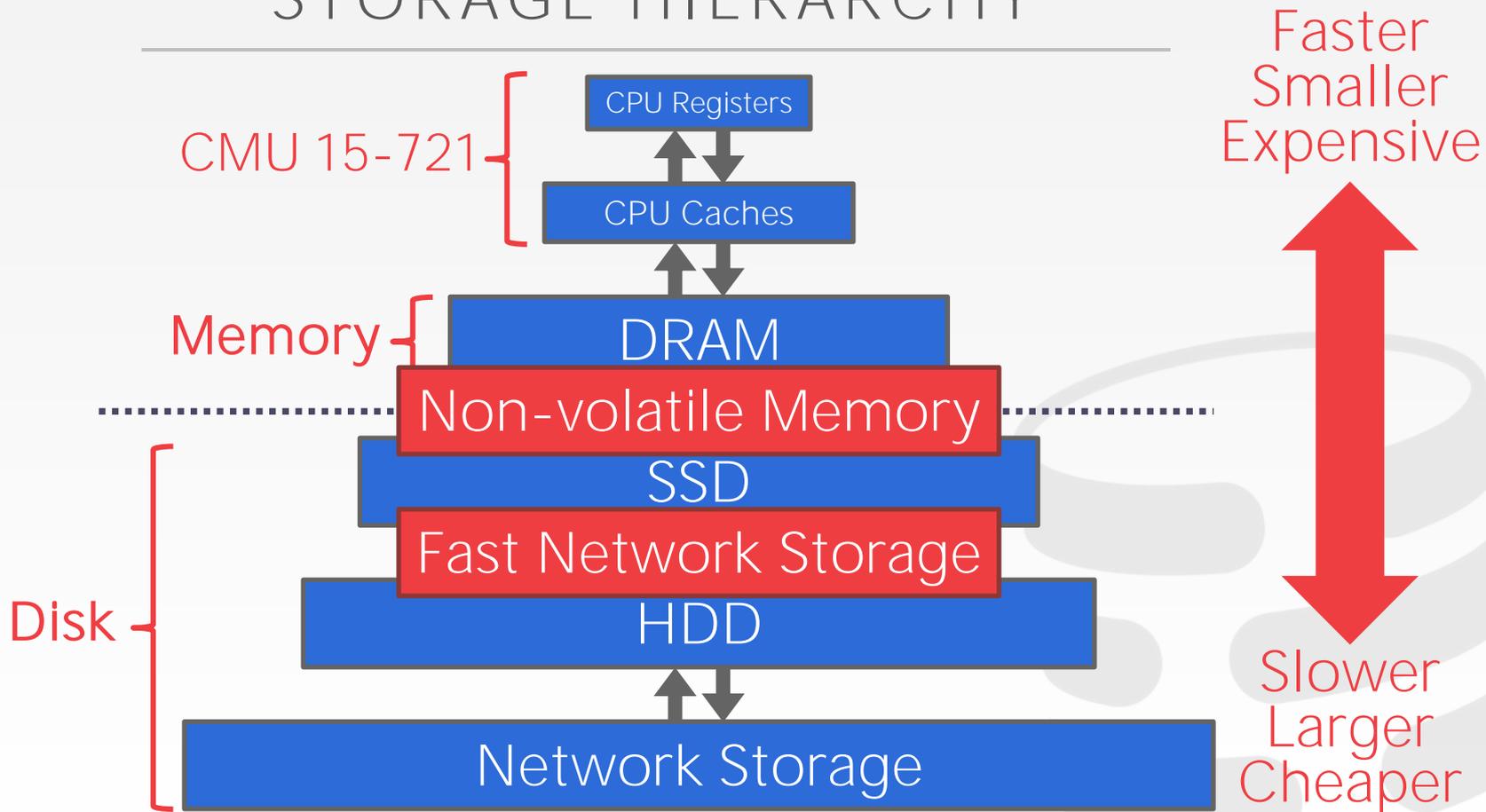
STORAGE HIERARCHY

CPU Registers

CPU Caches

Volatile
Random Access
Byte-Addressable

DRAM

SSD

Non-Volatile
Sequential Access
Block-Addressable

HDD

Network Storage

Faster
Smaller
Expensive

Slower
Larger
Cheaper

STORAGE HIERARCHY

CPU Registers

CMU 15-721

CPU Caches

Memory — DRAM

SSD

Disk — HDD

Network Storage

Faster
Smaller
Expensive

Slower
Larger
Cheaper

STORAGE HIERARCHY

ACCESS TIMES

0.5 ns L1 Cache Ref	← 0.5 sec
7 ns L2 Cache Ref	← 7 sec
100 ns DRAM	← 100 sec
150,000 ns SSD	← 1.7 days
10,000,000 ns HDD	← 16.5 weeks
~30,000,000 ns Network Storage	← 11.4 months
1,000,000,000 ns Tape Archives	← 31.7 years

[Source]

SEQUENTIAL VS. RANDOM ACCESS

Random access on non-volatile storage is usually 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.
→ Allocating multiple pages at the same time is called an extent.

SYSTEM DESIGN GOALS

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

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.
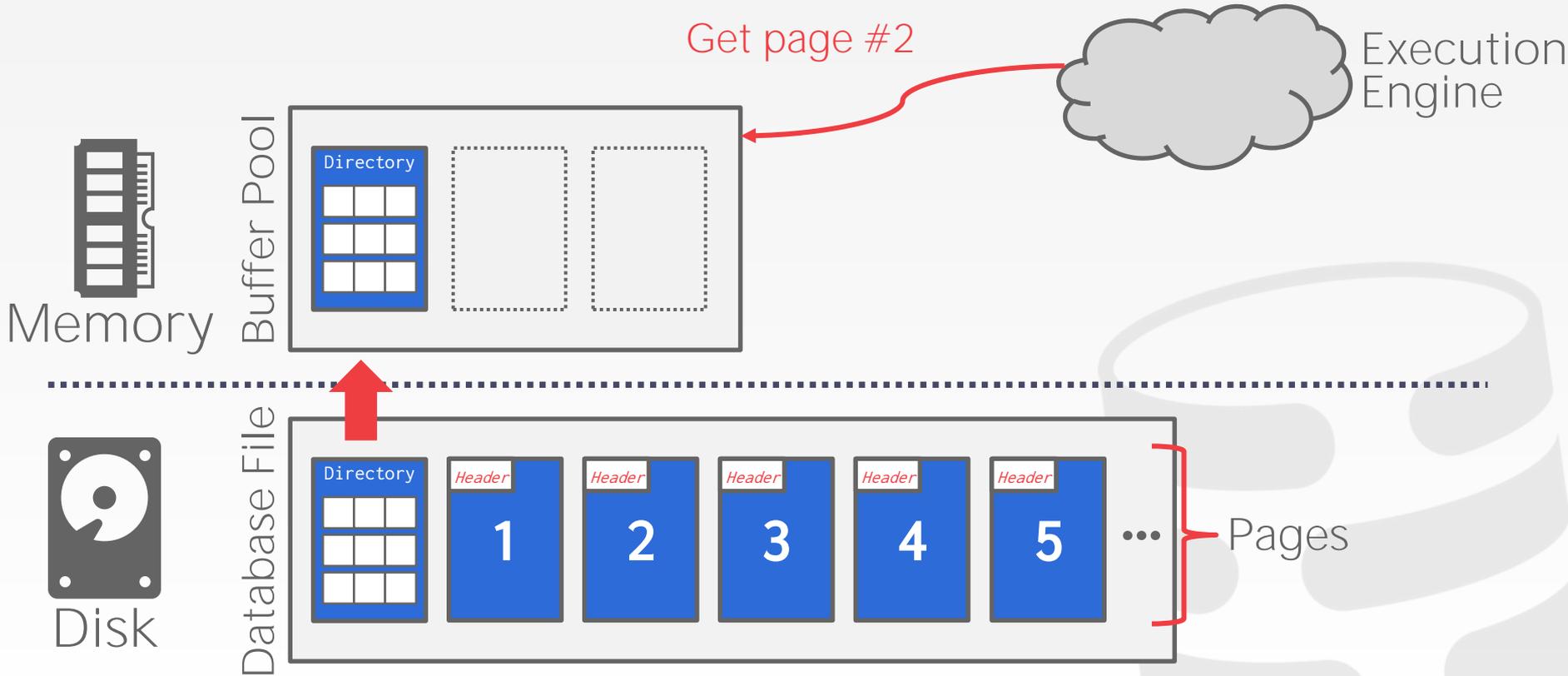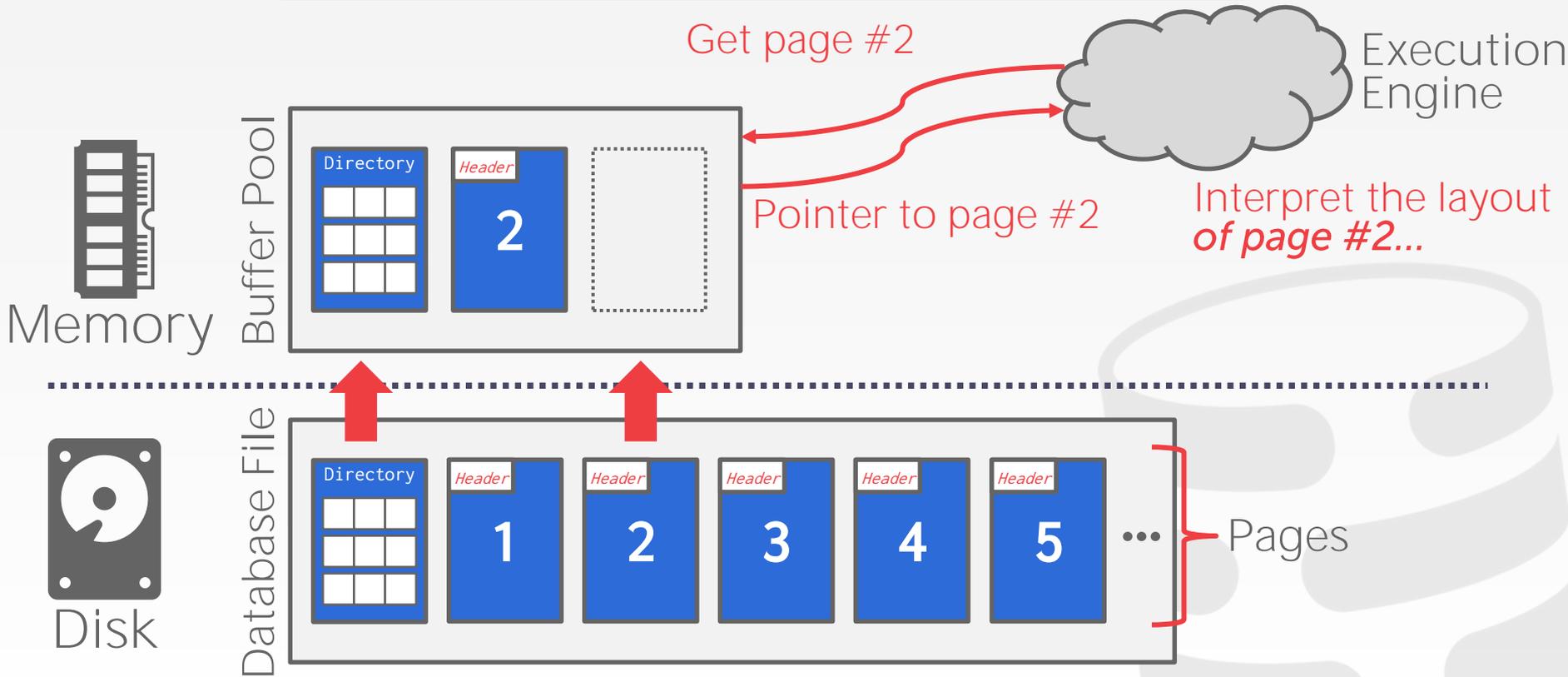
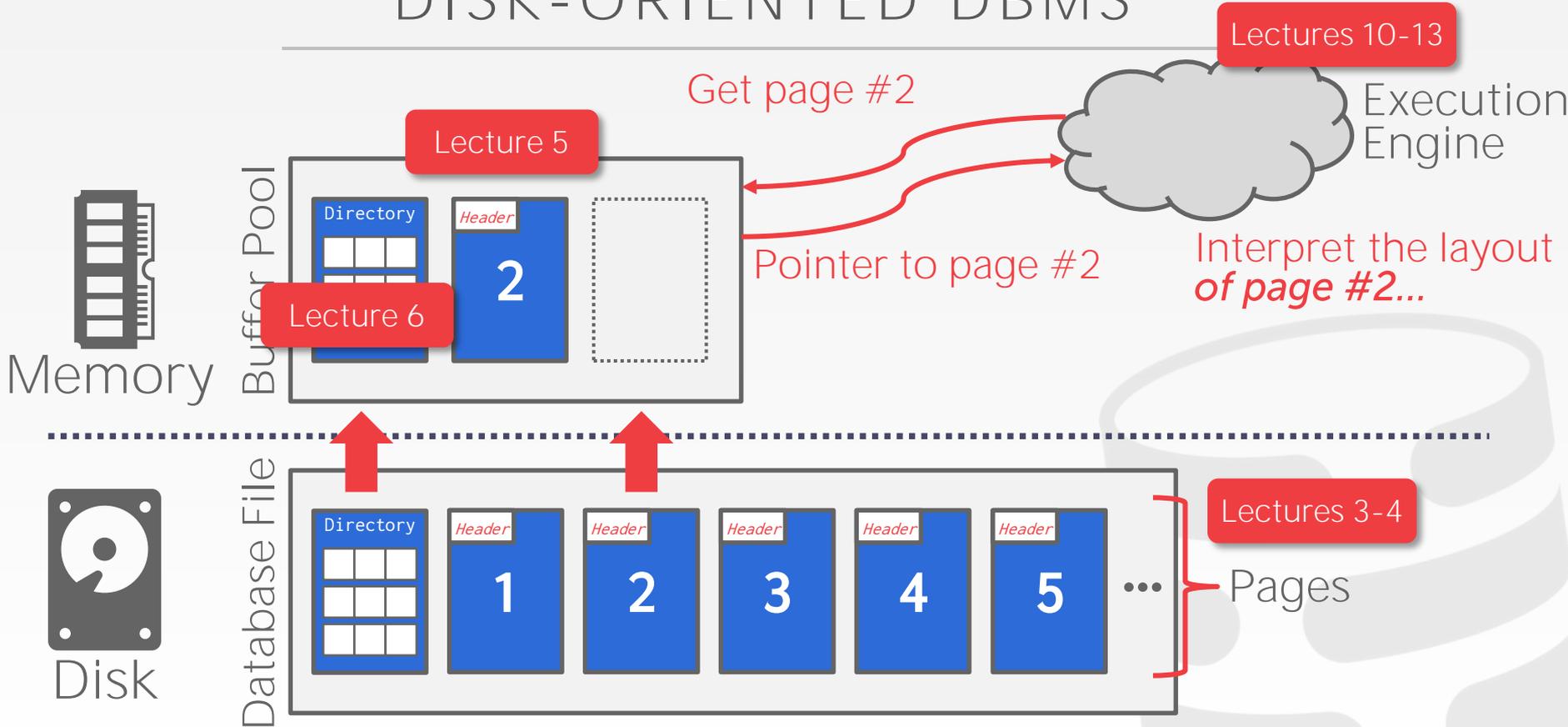# DISK-ORIENTED DBMS

Disk

Database File

DISK-ORIENTED DBMS

Get page #2

Execution Engine

Buffer Pool

Directory

Memory

Database File

Disk

Directory

Header	Header	Header	Header	Header
1	2	3	4	5

•••

Pages

DISK-ORIENTED DBMS

Get page #2

Execution Engine

Interpret the layout *of page #2...*

Pointer to page #2

Memory

Buffer Pool

Directory

Header

2

Disk

Database File

Directory

Header **1**
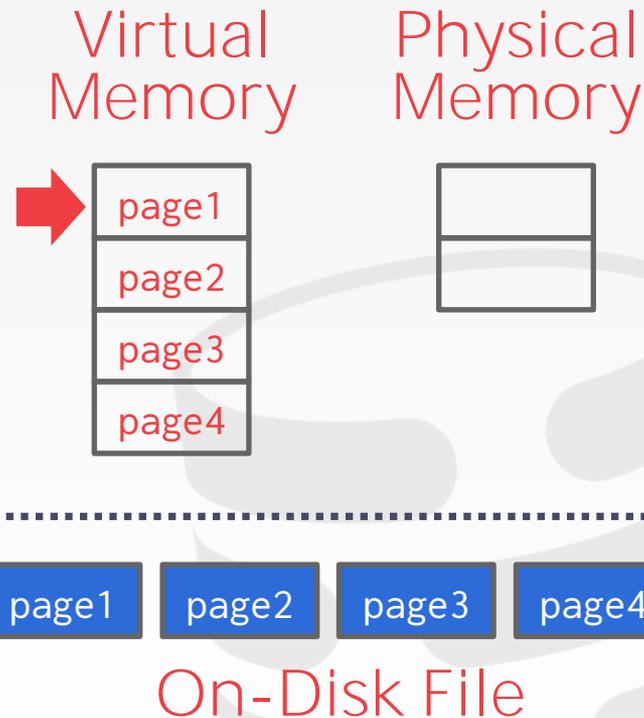
Header **2**

Header **3**

Header **4**

Header **5**

...

Pages

WHY NOT USE THE OS?

One can use memory mapping (**mmap**) to store the contents of a file into a process' address space.

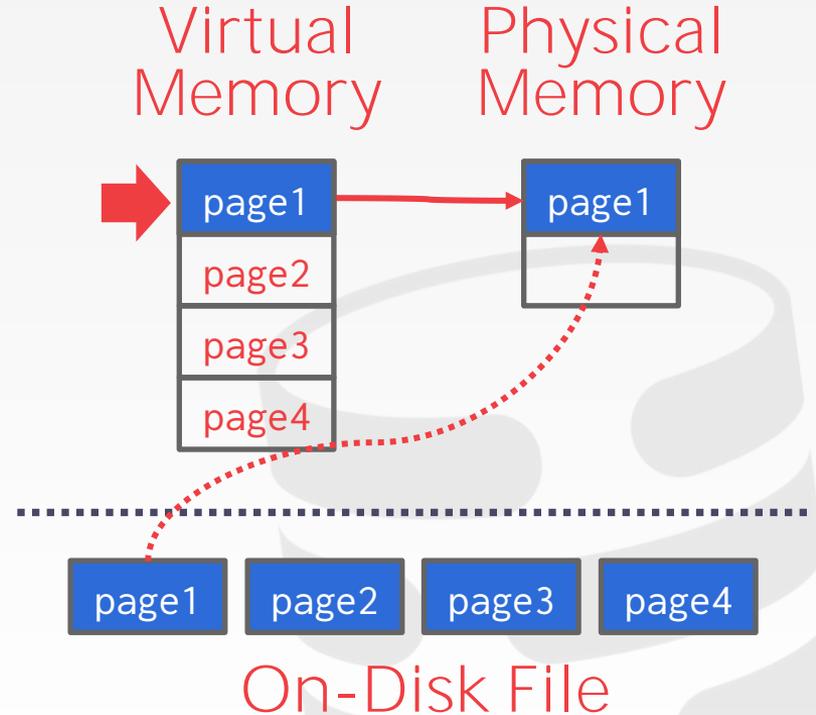The OS is responsible for moving data for moving the files' pages in and out of memory.

Virtual Memory

Physical Memory

| page1 |
| page2 |
| page3 |
| page4 |

On-Disk File

| page1 | page2 | page3 | page4 |

WHY NOT USE THE OS?

One can use memory mapping (**mmap**) to store the contents of a file into a process' address space.

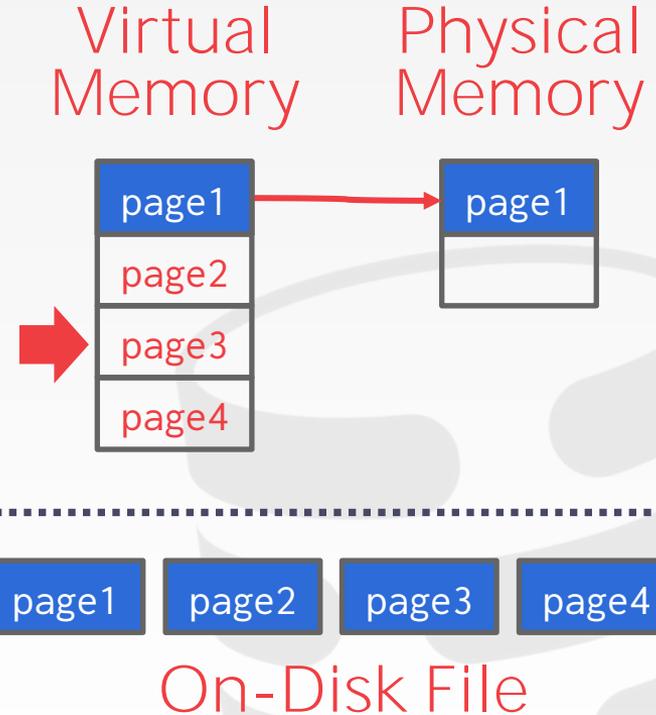The OS is responsible for moving data for moving the files' pages in and out of memory.

Virtual Memory

Physical Memory

page1 → page1

page2

page3

page4

page1 page2 page3 page4

On-Disk File

WHY NOT USE THE OS?

One can use memory mapping (**mmap**) to store the contents of a file into a process' address space.

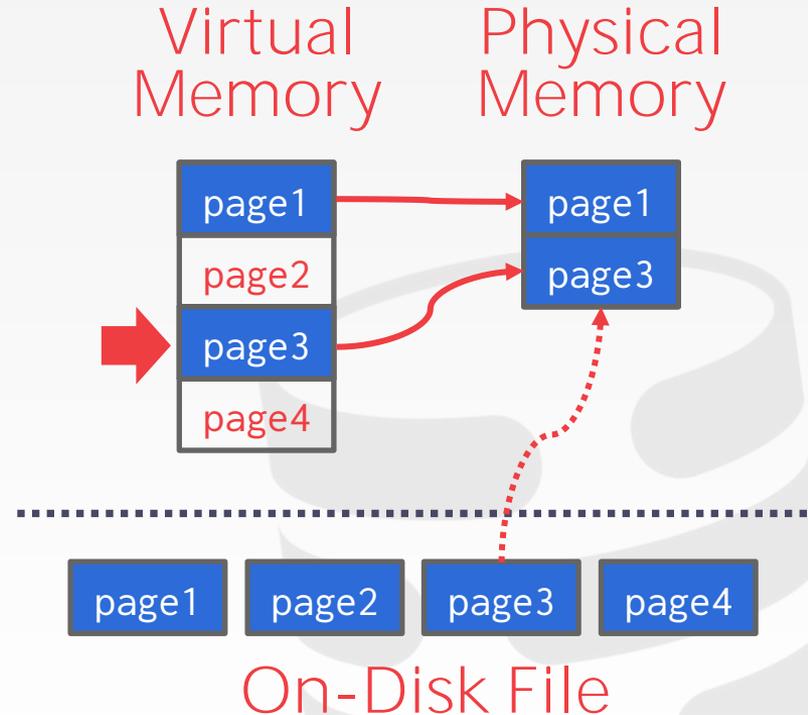The OS is responsible for moving data for moving the files' pages in and out of memory.

Virtual Memory

Physical Memory

| page1 | → | page1 |

page2

page3

page4

On-Disk File

| page1 | page2 | page3 | page4 |

WHY NOT USE THE OS?

One can use memory mapping (**mmap**) to store the contents of a file into a process' address space.

The OS is responsible for moving data for moving the files' pages in and out of memory.

Virtual Memory   Physical Memory

WHY NOT USE THE OS?

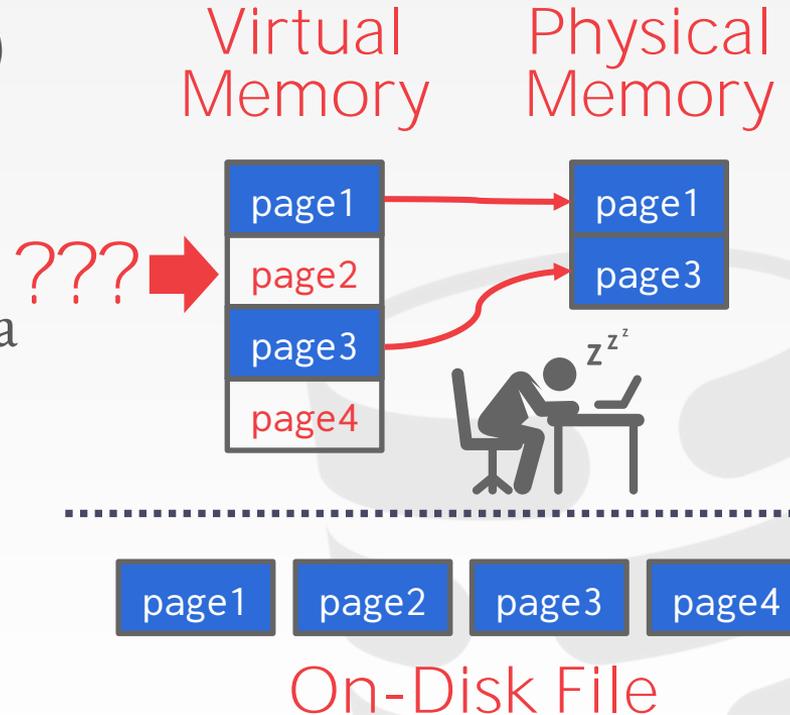One can use memory mapping (**mmap**) to store the contents of a file into a process' address space.

The OS is responsible for moving data for moving the files' pages in and out of memory.

Virtual Memory

Physical Memory

On-Disk File

WHY NOT USE THE OS?

What if we allow multiple threads to access the `mmap` files to hide page fault stalls?

This works good enough for read-only access.
It is complicated when there are multiple writers…

WHY NOT USE THE OS?

There are some solutions to this problem:
→ **madvise**: Tell the OS how you expect to read certain pages.
→ **mlock**: Tell the OS that memory ranges cannot be paged out.
→ **msync**: Tell the OS to flush memory ranges out to disk.

Full Usage

monetdb
LMDB
RAVENDB
levelDB
elasticsearch

Partial Usage

mongoDB
MEMSQL
SQLite
influxdb

WHY NOT USE THE OS?

DBMS (almost) always wants to control things itself and can do a better job at it.
→ Flushing dirty pages to disk in the correct order.
→ Specialized prefetching.
→ Buffer replacement policy.
→ Thread/process scheduling.

The OS is **not** your friend.

DATABASE STORAGE

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

← Today

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

TODAY'S AGENDA

File Storage

Page Layout

Tuple Layout

FILE STORAGE

The DBMS stores a database as one or more files on disk typically in a proprietary format.
→ The OS doesn't know anything about the contents of these files.

Early systems in the 1980s used custom filesystems on raw storage.
→ Some "enterprise" DBMSs still support this.
→ Most newer DBMSs do not do this.

STORAGE MANAGER

The <u>storage manager</u> 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 <u>pages</u>.
→ Tracks data read/written to pages.
→ Tracks the available space.

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.
→ 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)
→ Database Page (512B-16KB)

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

DATABASE PAGES

There are three different notions of "pages" in a DBMS:
→ Hardware Page (usually 4KB)
→ OS Page (usually 4KB)
→ Database Page (512B-16KB)

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

4KB

8KB

16KB

DATABASE HEAP

A <u>heap file</u> is an unordered collection of pages where tuples that are stored in random order.
→ Create / Get / Write / Delete Page
→ Must also support iterating over all pages.
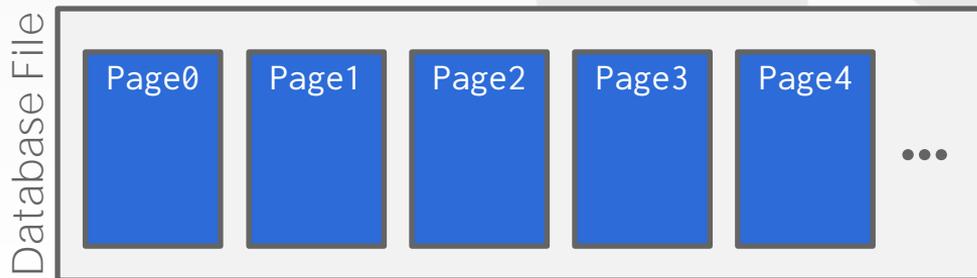
Two ways to represent a heap file:
→ Linked List
→ Page Directory

DATABASE HEAP

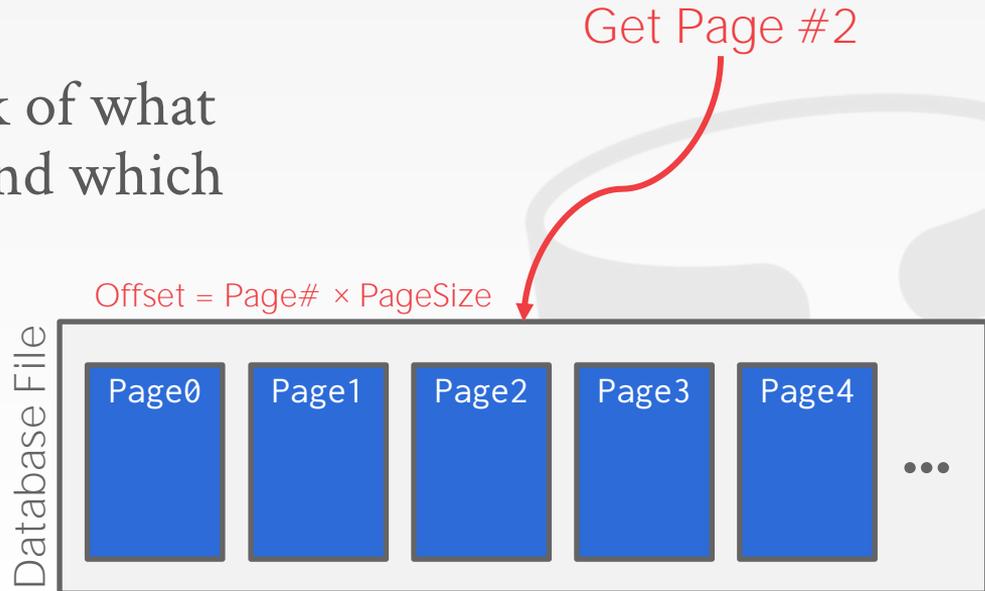It is easy to find pages if there is only a single heap file.

Need meta-data to keep track of what pages exist in multiple files and which ones have free space.

Database File: Page0 Page1 Page2 Page3 Page4 ...

DATABASE HEAP

It is easy to find pages if there is only a single heap file.

Need meta-data to keep track of what pages exist in multiple files and which ones have free space.

Get Page #2

Offset = Page# × PageSize

Database File

| Page0 | Page1 | Page2 | Page3 | Page4 | ••• |

DATABASE HEAP

It is easy to find pages if there is only a single heap file.

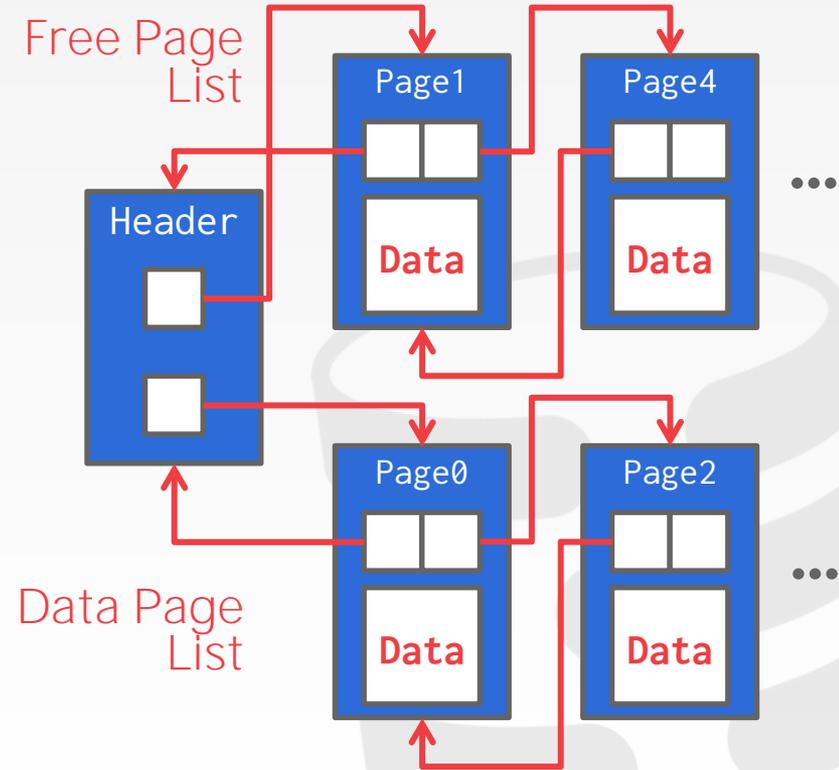Need meta-data to keep track of what pages exist in multiple files and which ones have free space.

Get Page #2
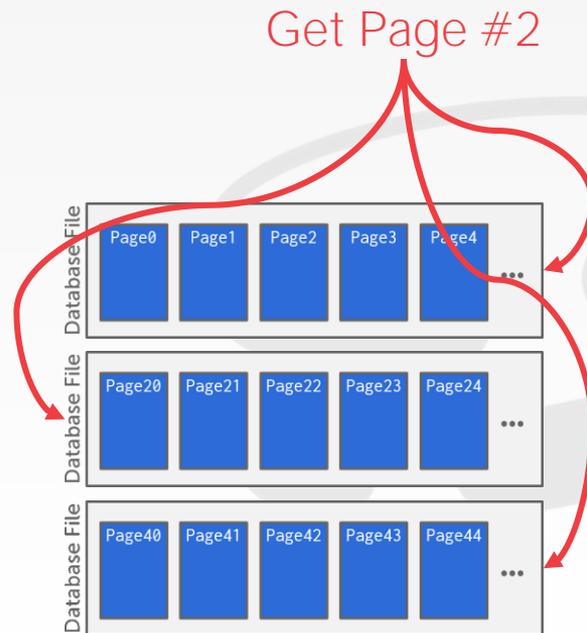
HEAP FILE: LINKED LIST

Maintain a <u>header page</u> at the beginning of the file that stores two pointers:
→ HEAD of the <u>free page list</u>.
→ HEAD of the <u>data page list</u>.

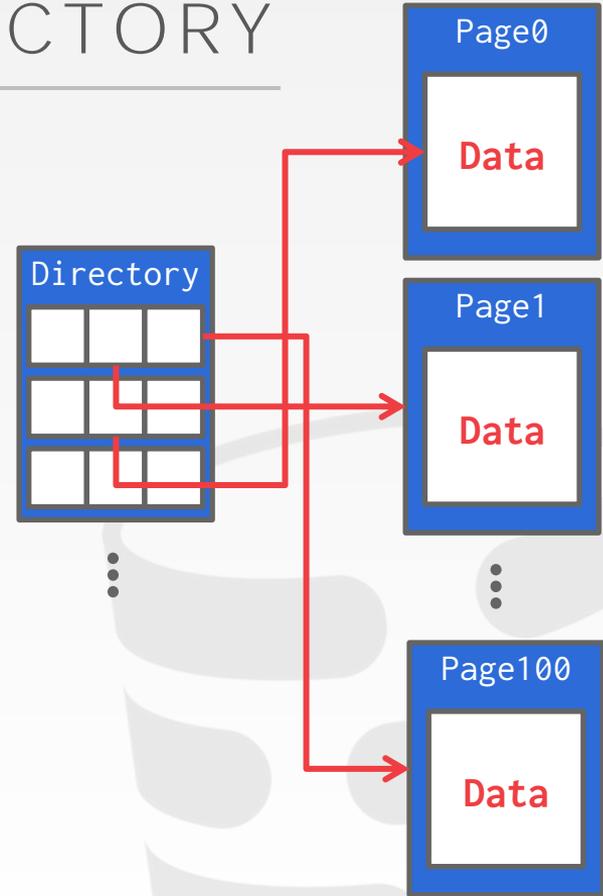Each page keeps track of how many free slots they currently have.

HEAP FILE: PAGE DIRECTORY

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

The directory also records the number of free slots per page.

The DBMS must make sure that the directory pages are in sync with the data pages.

TODAY'S AGENDA

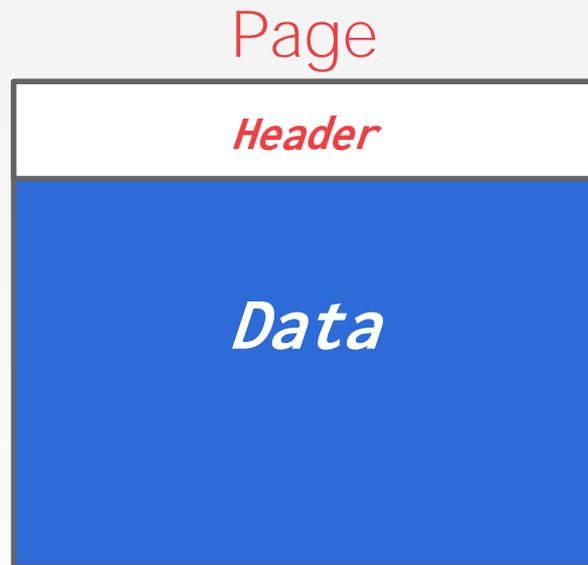File Storage

Page Layout

Tuple Layout

PAGE HEADER

Every page contains a <u>header</u> of meta-data about the page's contents.
→ Page Size
→ Checksum
→ DBMS Version
→ Transaction Visibility
→ Compression Information

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

Page

Header
Data

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.
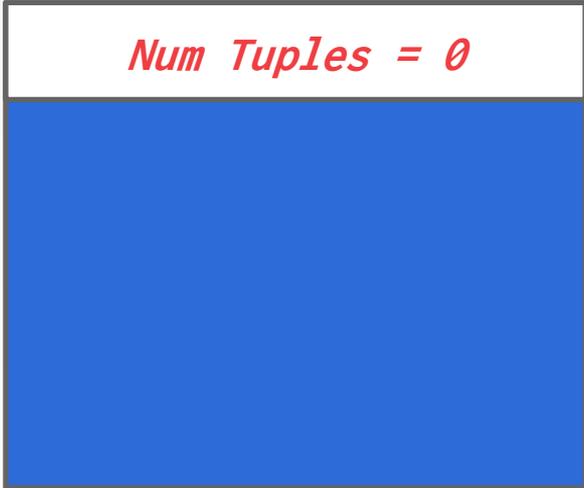
Two approaches:
→ Tuple-oriented
→ Log-structured

TUPLE 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.

Page

Num Tuples = 0

TUPLE 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.

Page

Num Tuples = 3
Tuple #1
Tuple #2
Tuple #3

TUPLE 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?

Page

| Num Tuples = 2 |
| Tuple #1 |
| |
| Tuple #3 |
| |

TUPLE STORAGE

Page

How to store tuples in a page?

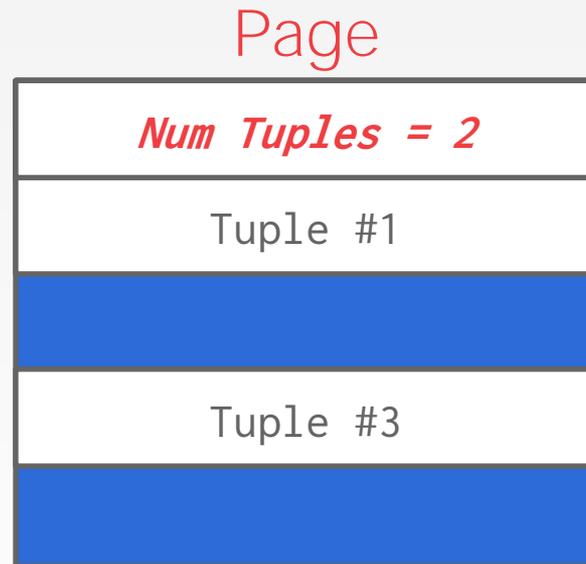**Strawman Idea:** Keep track of the number of tuples in a page and then just append a new tuple to the end.
→ What happens if we delete a tuple?

Num Tuples = 3
Tuple #1
Tuple #4
Tuple #3

TUPLE STORAGE

Page

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?

Num Tuples = 3

Tuple #1

Tuple #4

Tuple #3

SLOTTED PAGES

The most common layout scheme is called <u>slotted pages</u>.

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.

Slot Array

| Header | | | | | | | |

Tuple #4 | Tuple #3

Tuple #2 | Tuple #1

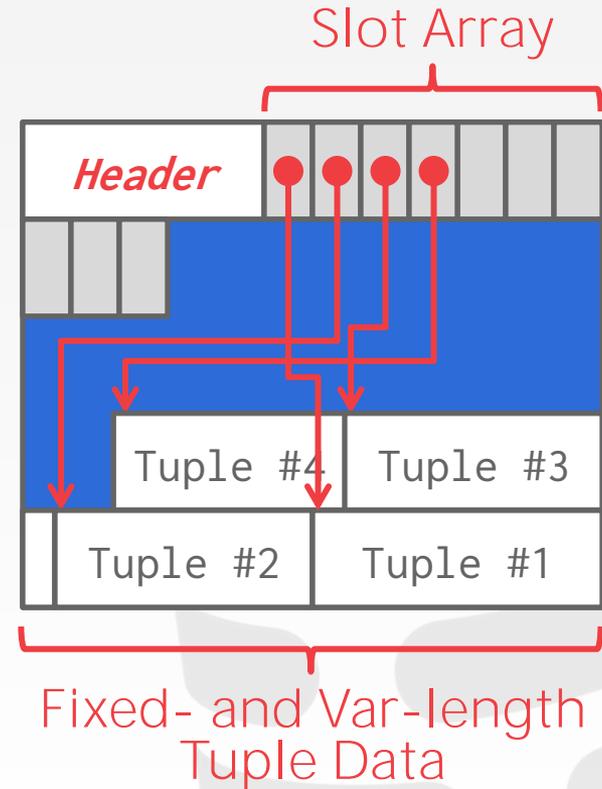Fixed- and Var-length Tuple Data

SLOTTED PAGES

The most common layout scheme is called <u>slotted pages</u>.

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.

Slot Array

Header

Tuple #4   Tuple #3

Tuple #2   Tuple #1
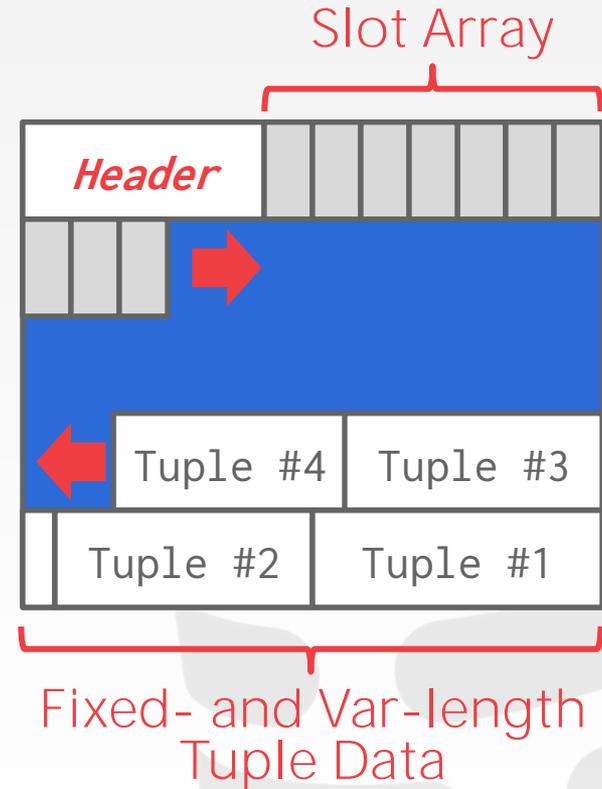
Fixed- and Var-length Tuple Data

SLOTTED PAGES

The most common layout scheme is called <u>slotted pages</u>.

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.

Slot Array

Header

Tuple #4 | Tuple #3

Tuple #2 | Tuple #1
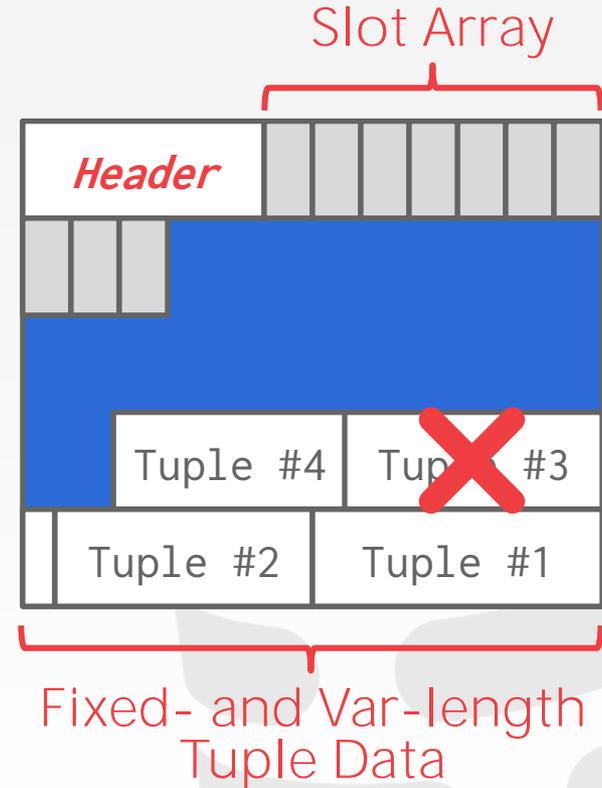
Fixed- and Var-length Tuple Data

SLOTTED PAGES

The most common layout scheme is called <u>slotted pages</u>.

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.

Slot Array

Header

Tuple #4

Tuple #2   Tuple #1

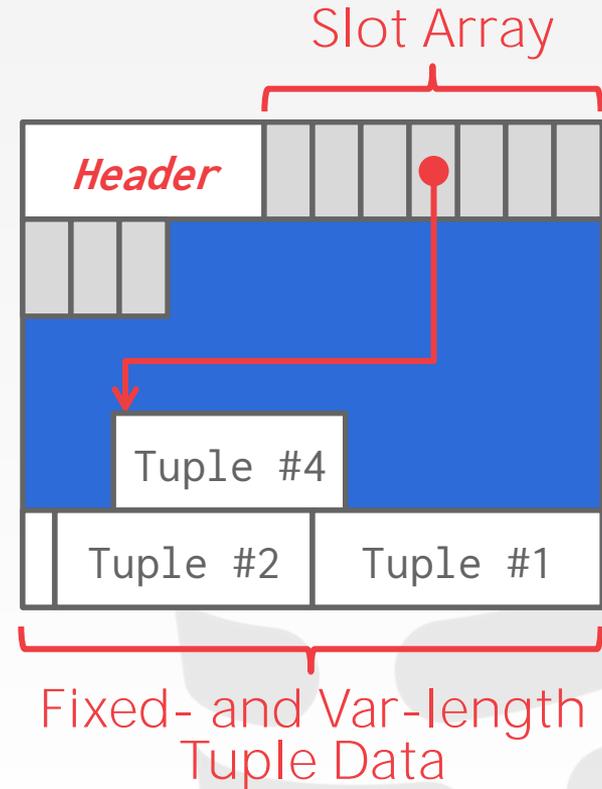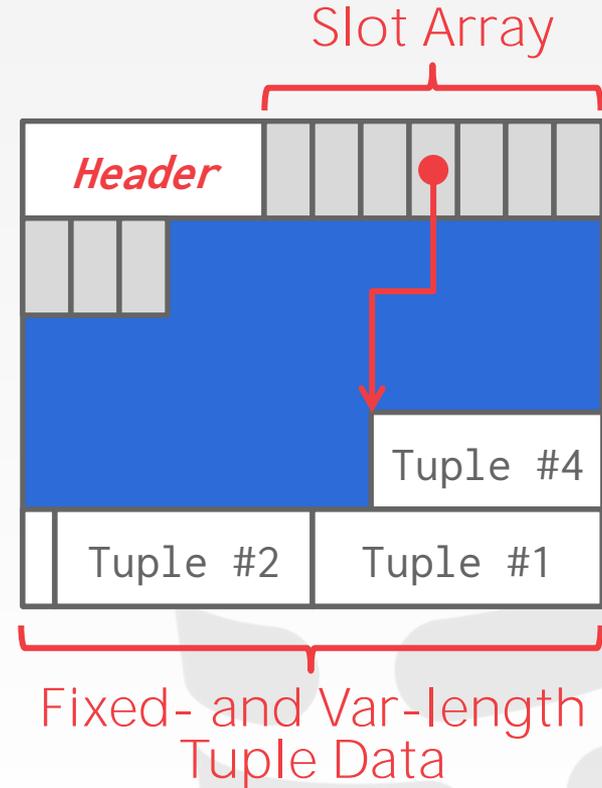Fixed- and Var-length Tuple Data

SLOTTED PAGES

The most common layout scheme is called <u>slotted pages</u>.

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.

Slot Array

| Header | | | | | | |

Tuple #4

Tuple #2 | Tuple #1

Fixed- and Var-length Tuple Data

RECORD IDS

The DBMS needs a way to keep track of individual tuples.

Each tuple is assigned a unique <u>record identifier</u>.
→ Most common: **page_id** + **offset/slot**
→ Can also contain file location info.

An application <u>cannot</u> rely on these ids to mean anything.

PostgreSQL
CTID (6-bytes)

SQLite
ROWID (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.

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

TUPLE HEADER

Tuple

Header	Attribute Data

Each tuple is prefixed with a <u>header</u> that contains meta-data about it.

→ Visibility info (concurrency control)

→ Bit Map for **NULL** values.

We do <u>not</u> need to store meta-data about the schema.

TUPLE DATA

Tuple

Header	a	b	c	d	e

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

This is done for software engineering reasons.

We re-order attributes automatically in CMU's new DBMS…

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

DENORMALIZED TUPLE DATA

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.

```
CREATE TABLE foo (
  a INT PRIMARY KEY,
  b INT NOT NULL,
);
```

```
CREATE TABLE bar (
  c INT PRIMARY KEY,
  a INT
    ⮳REFERENCES foo (a),
);
```
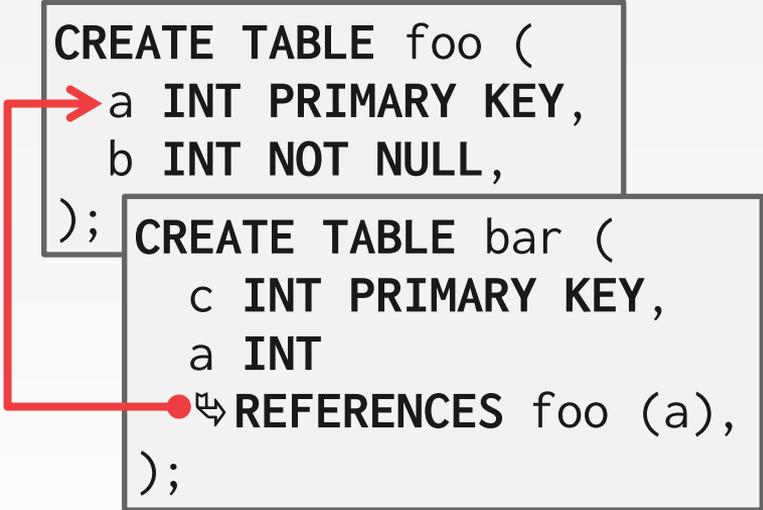
DENORMALIZED TUPLE DATA

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.

foo

Header	a	b

bar

| Header	c	a
Header	c	a
Header	c	a

DENORMALIZED TUPLE DATA

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.

Not a new idea.
→ IBM System R did this in the 1970s.
→ Several NoSQL DBMSs do this without calling it physical denormalization.

foo

| Header | a | b | c | c | c | ••• |

foo   bar

RethinkDB

CouchDB

MarkLogic™

RAVENDB

mongoDB
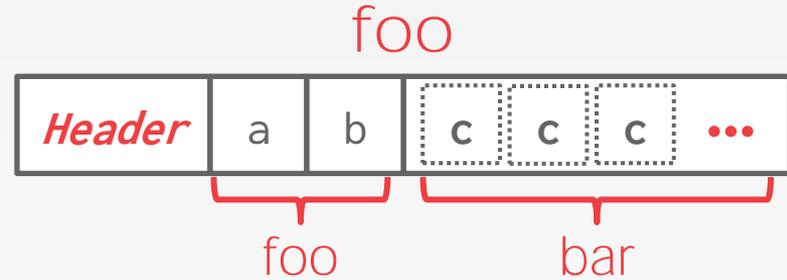
CONCLUSION

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

Value Representation

Storage Models