Carnegie Mellon University

Database Storage
Part I

Lecture #03

Database Systems
15-445/15-645
Fall 2018

AP
Andy Pavlo
Computer Science
Carnegie Mellon Univ.

ADMINISTRIVIA

Homework #1 is due Monday September 10th @ 11:59pm

Project #1 will be released on Wednesday September 12th

UPCOMING DATABASE EVENTS

Kinetica Talk
→ Thursday Sep 6th @ 12pm
→ CIC 4th Floor

SalesForce Talk
→ Friday Sep 7th @ 12pm
→ CIC 4th Floor

Relational AI Talk
→ Wednesday @ Sep 12th @ 4:00pm
→ GHC 8102

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

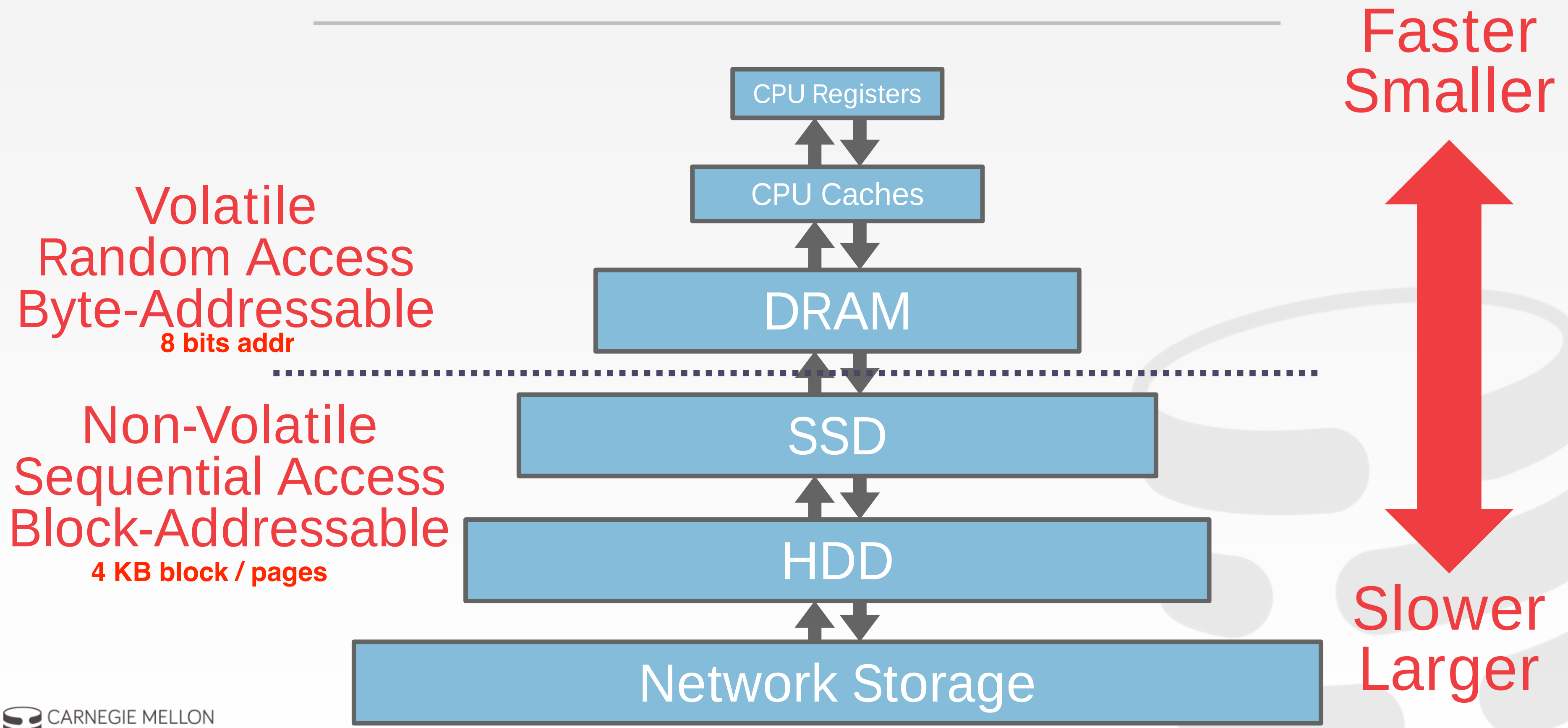
Query Planning
Operator Execution
Access Methods
Buffer Pool Manager
Disk Manager

DISK-ORIENTED ARCHITECTURE

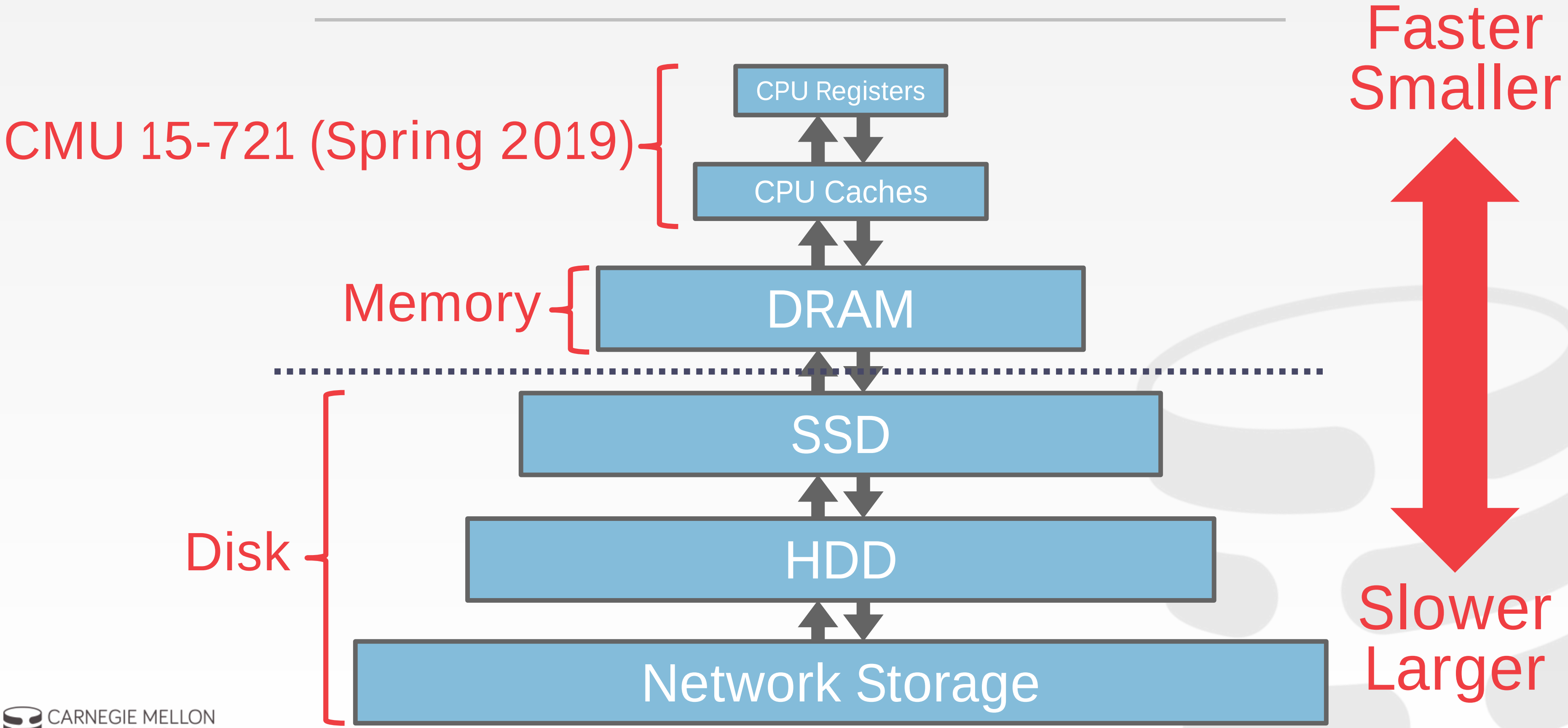
The DBMS assumes that the primary storage location of the database is on non-volatile disk.

The DBMS's components manage the movement of data between non-volatile and volatile storage.

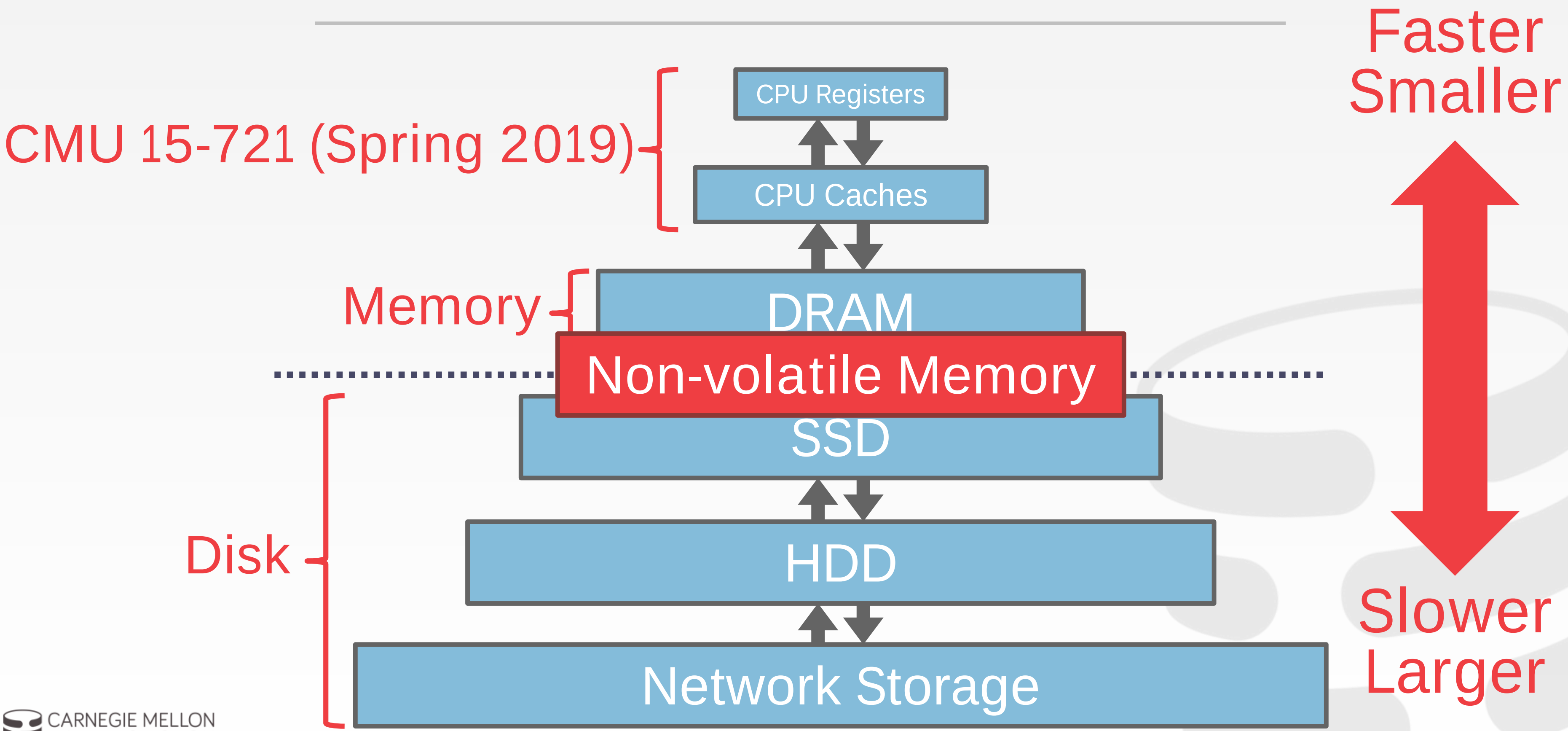
STORAGE HIERARCHY



STORAGE HIERARCHY



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]

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.

We want to have our database system provide the illusion it has more memory than it actually has
e.g. if we have 1GB RAM, and 10 GB database we want to make it appear that the entire database fits in memory

SEQUENTIAL VS. RANDOM ACCESS

Random access on an HDD is much slower than sequential access.

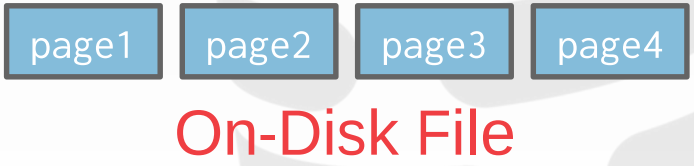
Traditional DBMSs are designed to maximize sequential access.

- → 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.

WHY NOT USE THE OS?

One can use **mmap** to map the contents of a file into a process' address space.

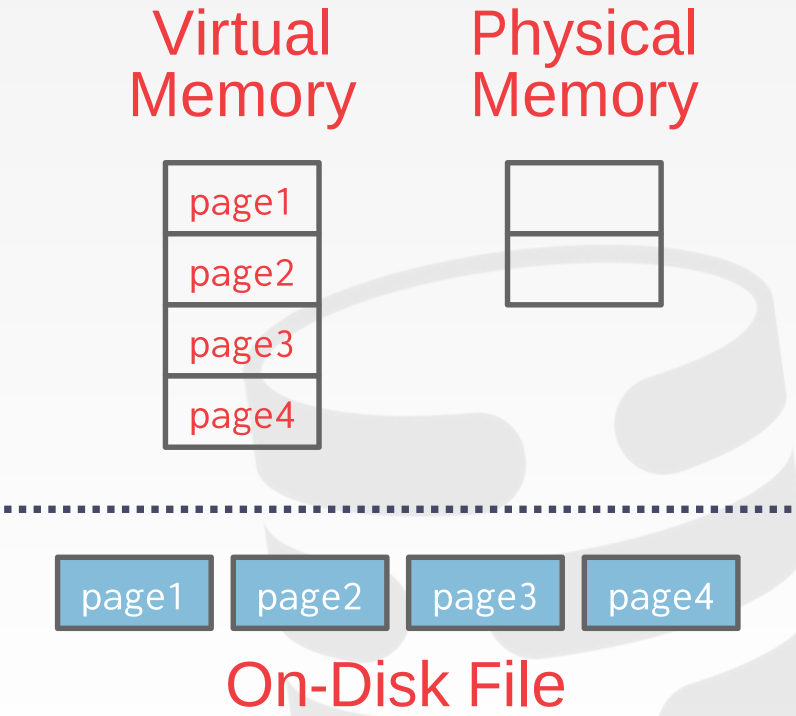
The OS is responsible for moving data for moving the files' pages in and out of memory.



WHY NOT USE THE OS?

One can use **mmap** to map the contents of a file into a process' address space.

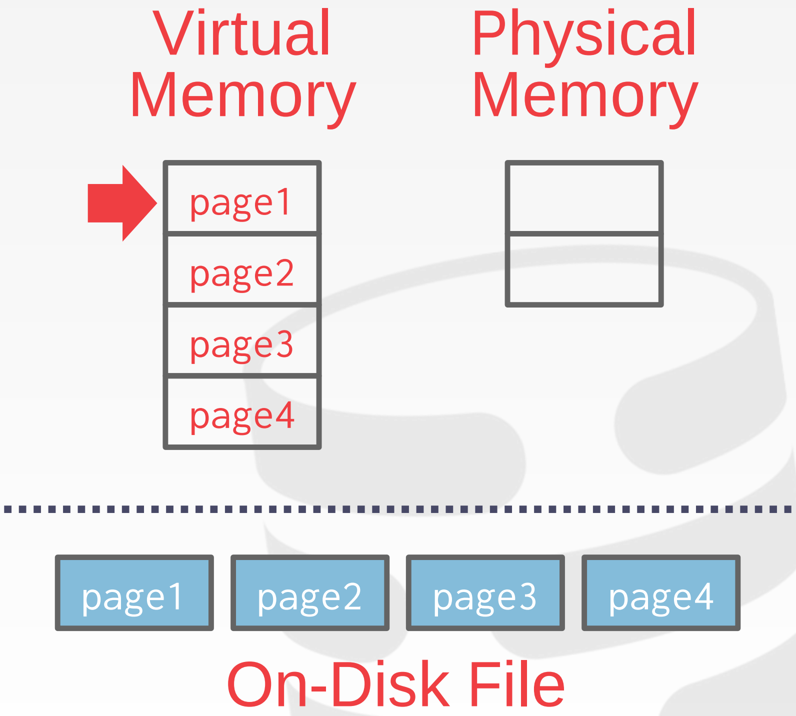
The OS is responsible for moving data for moving the files' pages in and out of memory.



WHY NOT USE THE OS?

One can use **mmap** to map the contents of a file into a process' address space.

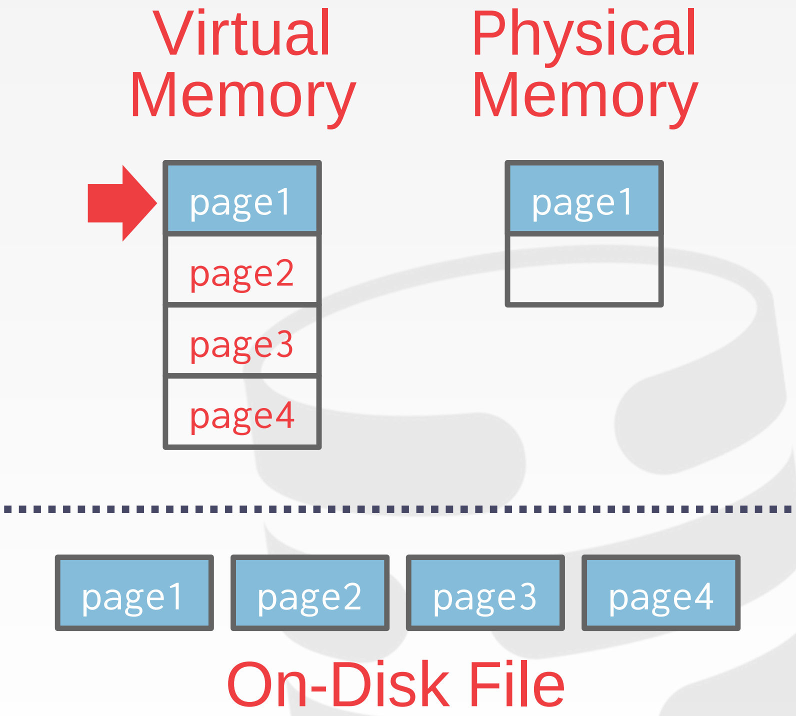
The OS is responsible for moving data for moving the files' pages in and out of memory.



WHY NOT USE THE OS?

One can use **mmap** to map the contents of a file into a process' address space.

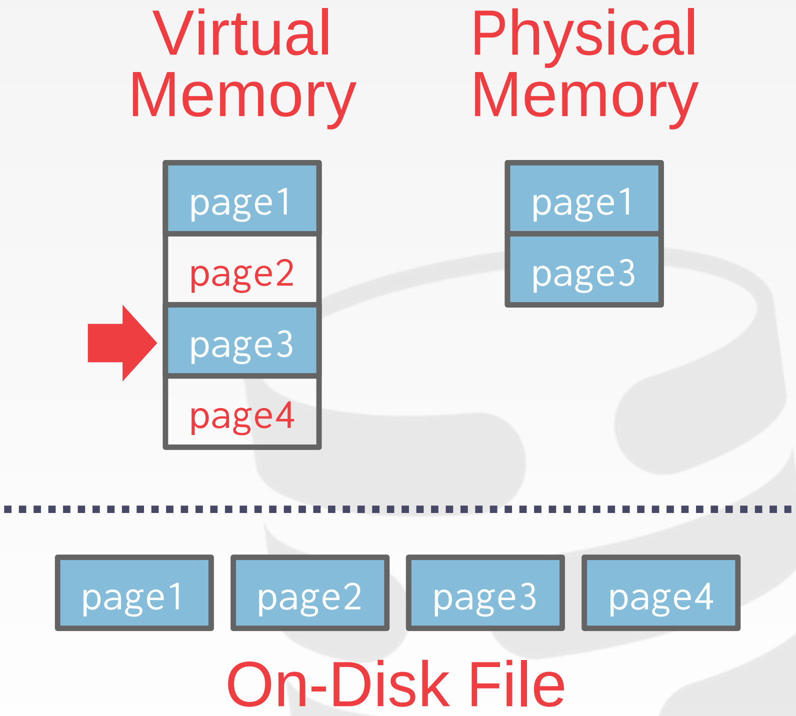
The OS is responsible for moving data for moving the files' pages in and out of memory.



WHY NOT USE THE OS?

One can use **mmap** to map the contents of a file into a process' address space.

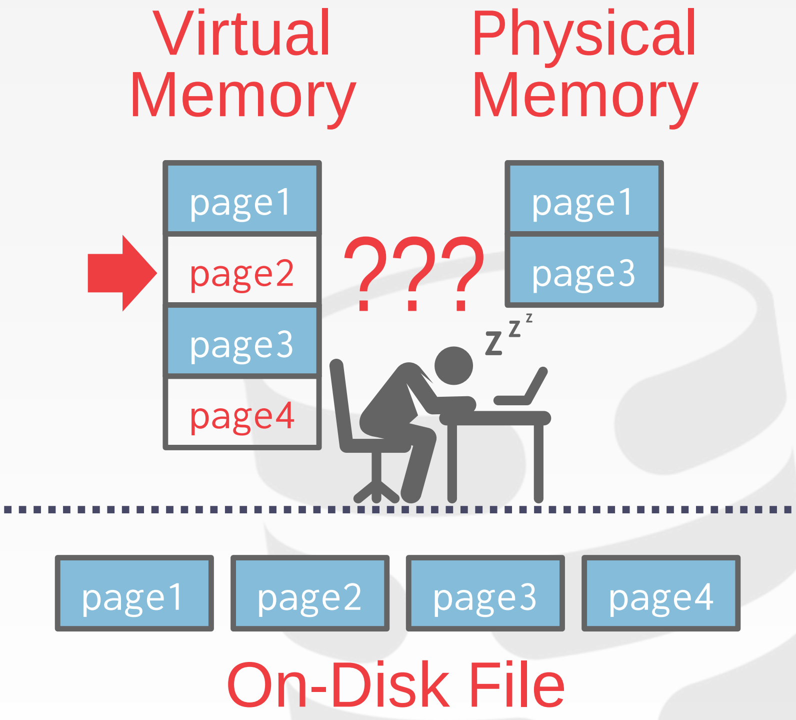
The OS is responsible for moving data for moving the files' pages in and out of memory.



WHY NOT USE THE OS?

One can use **mmap** to map 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.



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

Partial Usage

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.

The database system can always do a better job than the operating system because they know exactly what will happen
It knows what the queries is executing, it knows what the data looks like, it knows how it is going to access that data

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.

The OS doesn't know anything about these files.
- → All of the standard filesystem protections are used.
- → Early systems in the 1980s used custom "filesystems" on raw storage.

These files typically are in a proprietary format that is specific to the database system
So it is not like you can open up a text editor and actually see any data, it's gonna be some binary format that only database system can read and write to

STORAGE MANAGER

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

It organizes the files as a collection of pages.

→ Tracks data read/written to pages.
→ Tracks the available space.

DATABASE PAGES

A page 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.
独立的

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 (1-16KB)

By hardware page, we mean at what level the device can guarantee a "failsafe write".

1KB SQLite

4KB IBM DB2, ORACLE

8KB Microsoft SQL Server, PostgreSQL

16KB MySQL

PAGE STORAGE ARCHITECTURE

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

→ Heap File Organization
→ Sequential / Sorted File Organization
→ Hashing File Organization

At this point in the hierarchy we don't need to know anything about what is inside of the pages.

DATABASE HEAP

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

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

Two ways to represent a heap file:
→ Linked List
→ Page Directory

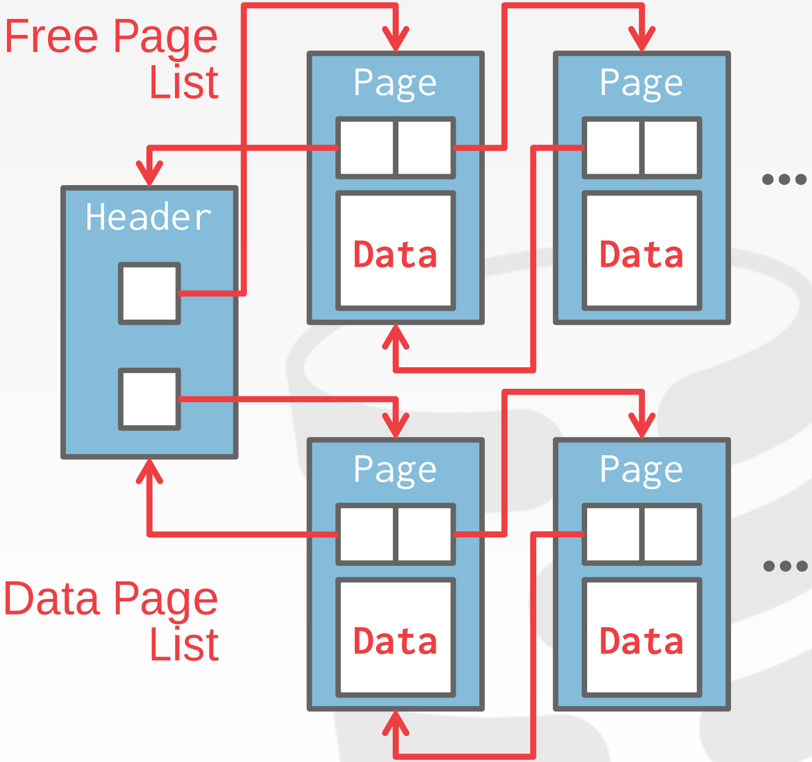
HEAP FILE: LINKED LIST

Maintain a header page at the beginning of the file that stores two pointers:

→ HEAD of the free page list.

→ HEAD of the data page list.

Each page keeps track of the number of free slots in itself.



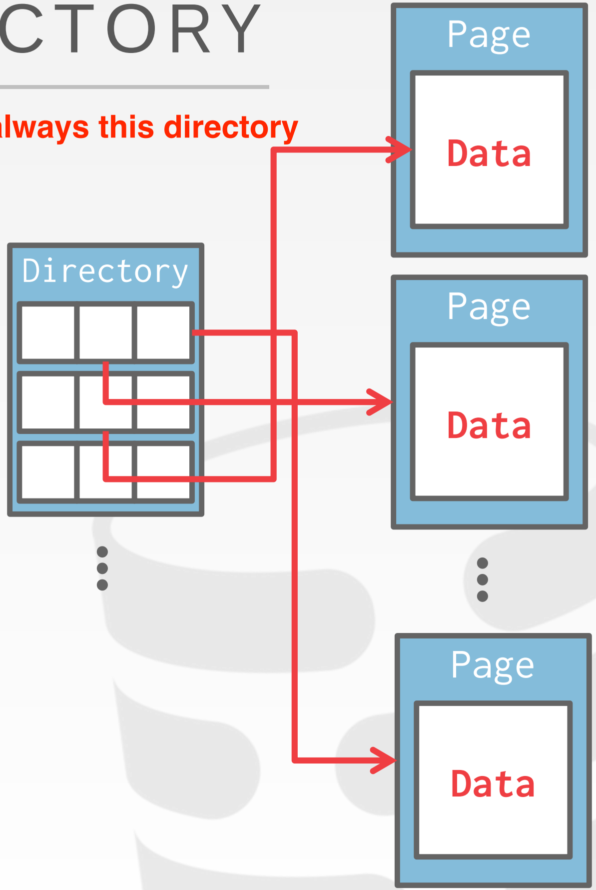
HEAP FILE: PAGE DIRECTORY

in sqlite, the first page is always this 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 has to make sure that the directory pages are in sync with the data pages.



TODAY'S AGENDA

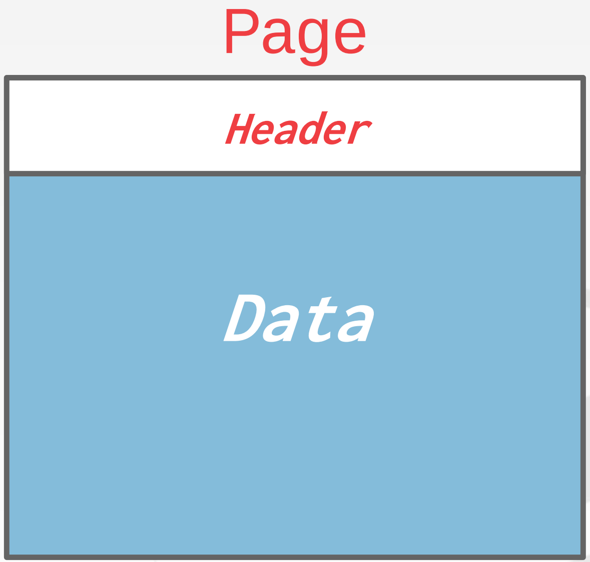
File Storage
Page Layout
Tuple Layout

PAGE HEADER

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

- → Page Size
- → Checksum
- → DBMS Version
- → Transaction Visibility
- → Compression Information

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



PAGE LAYOUT

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

→ We are still assuming that we are only storing tuples.

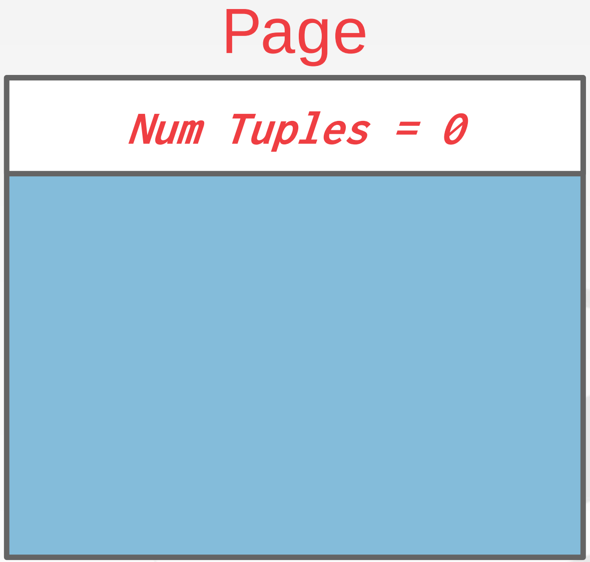
Two approaches:

→ Tuple-oriented
→ Log-structured

TUPLE STORAGE

How to store tuples in a page?

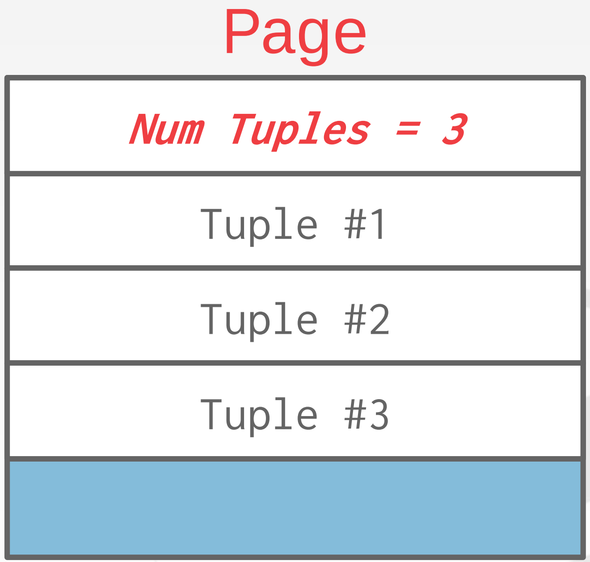
Strawman Idea: Keep track of the number of tuples in a page and then just append a new tuple to the end.



TUPLE STORAGE

How to store tuples in a page?

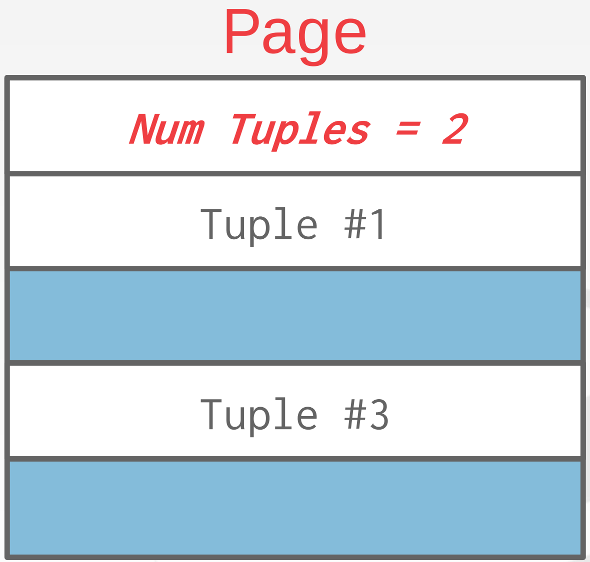
Strawman Idea: Keep track of the number of tuples in a page and then just append a new tuple to the end.



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?

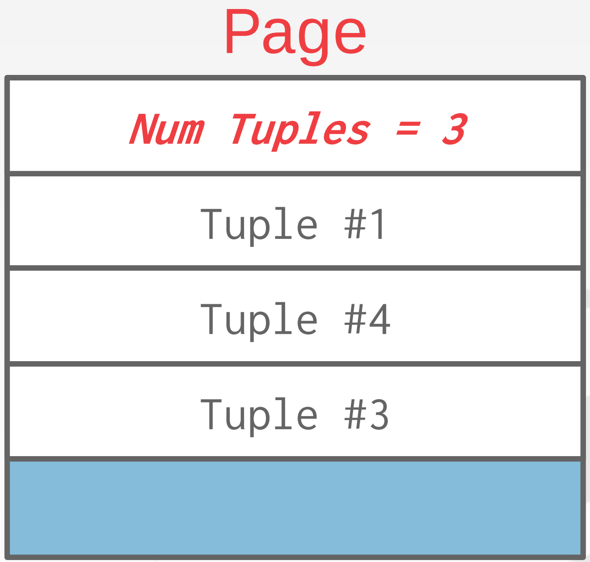


TUPLE STORAGE

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?

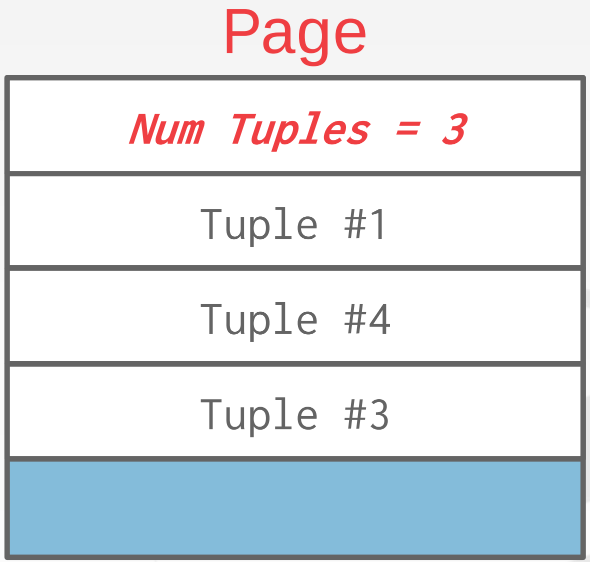


TUPLE STORAGE

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

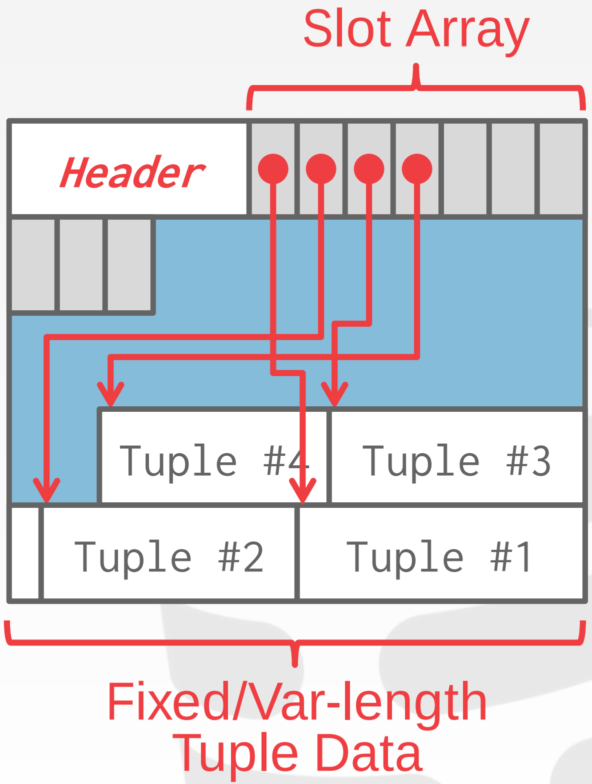


SLOTTED PAGES

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.

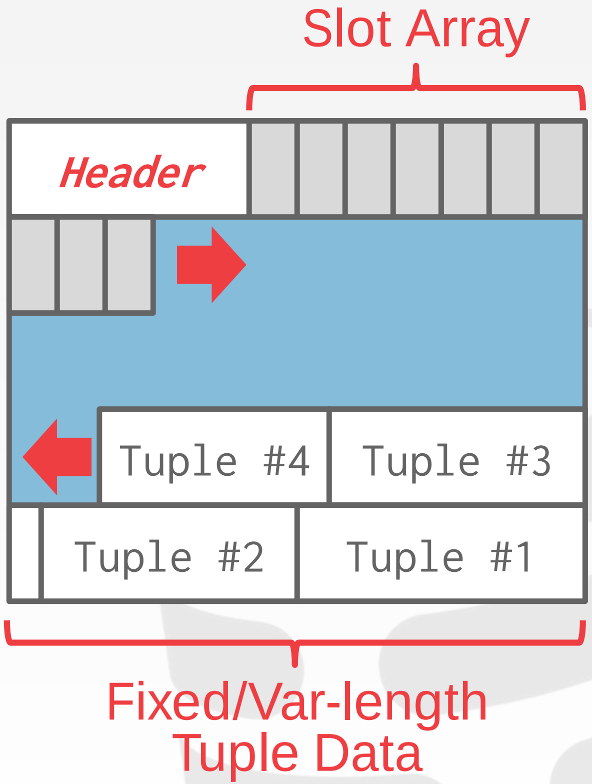


SLOTTED PAGES

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.



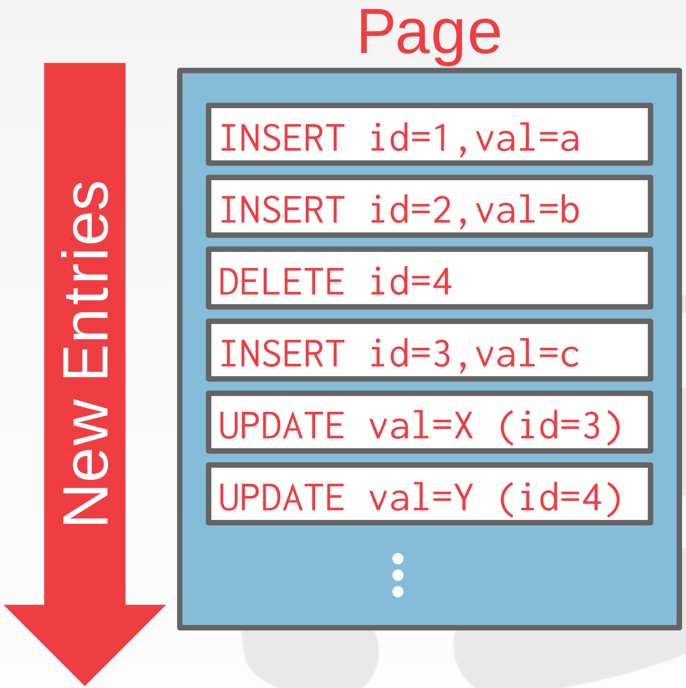
LOG-STRUCTURED FILE ORGANIZATION

Instead of storing tuples in pages, the DBMS only stores log records.

The system appends log records to the file of how the database was modified:

→ Inserts store the entire tuple.
→ Deletes mark the tuple as deleted.
→ Updates contain the delta of just the attributes that were modified.

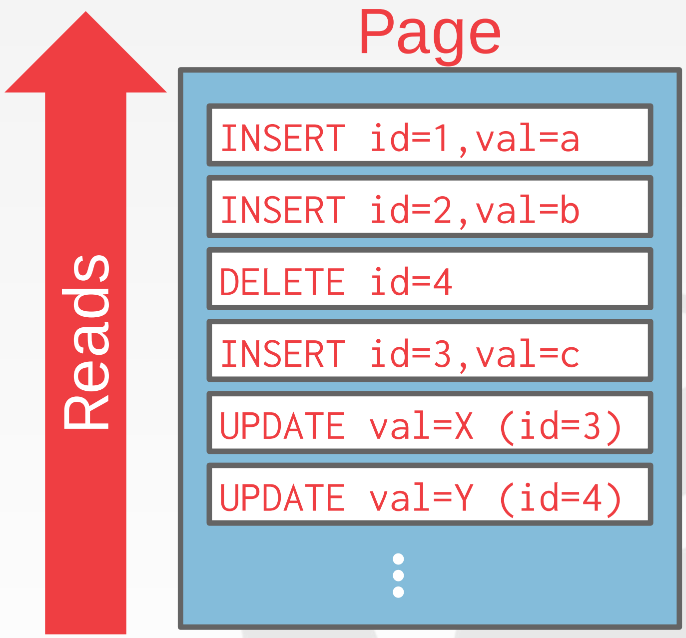
faster write
slower read



LOG-STRUCTURED FILE ORGANIZATION

To read a record, the DBMS scans the log backwards and "recreates" the tuple to find what it needs.

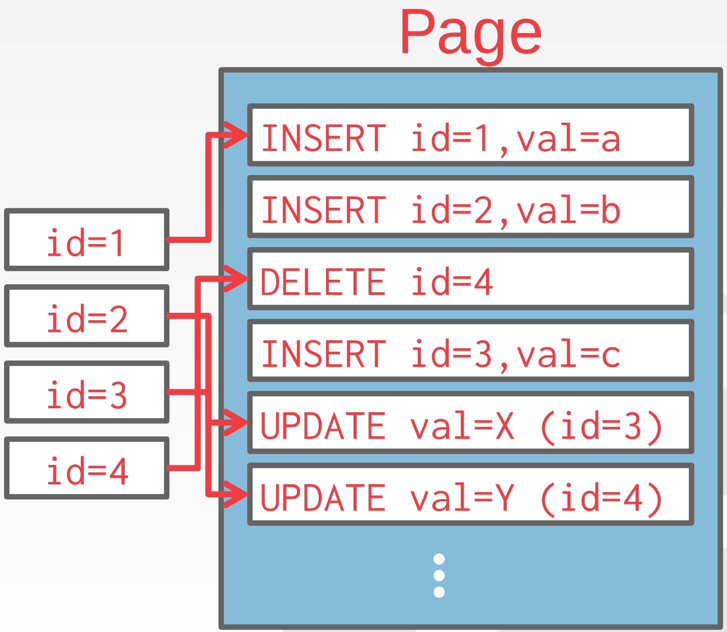
Build indexes to allow it to jump to locations in the log.



LOG-STRUCTURED FILE ORGANIZATION

To read a record, the DBMS scans the log backwards and "recreates" the tuple to find what it needs.

Build indexes to allow it to jump to locations in the log.

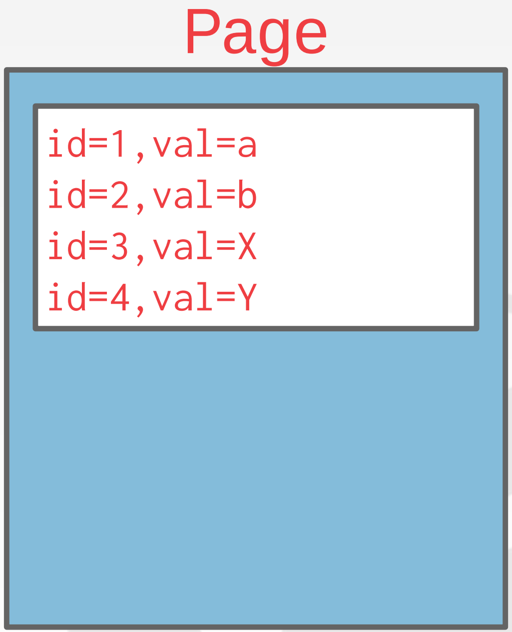


LOG-STRUCTURED FILE ORGANIZATION

To read a record, the DBMS scans the log backwards and "recreates" the tuple to find what it needs.

Build indexes to allow it to jump to locations in the log.

Periodically compact the log.



LOG-STRUCTURED COMPACTION

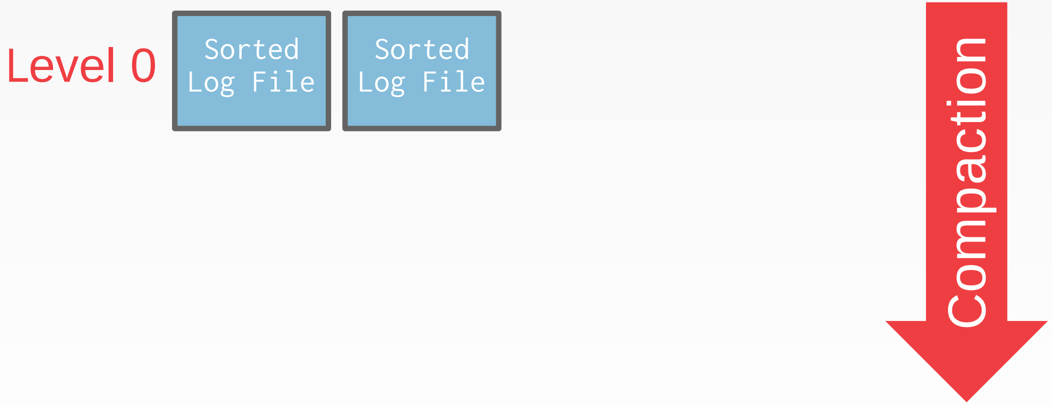
Compaction coalesces larger log files into smaller files by removing unnecessary records.

Level Compaction

LOG-STRUCTURED COMPACTION

Compaction coalesces larger log files into smaller files by removing unnecessary records.

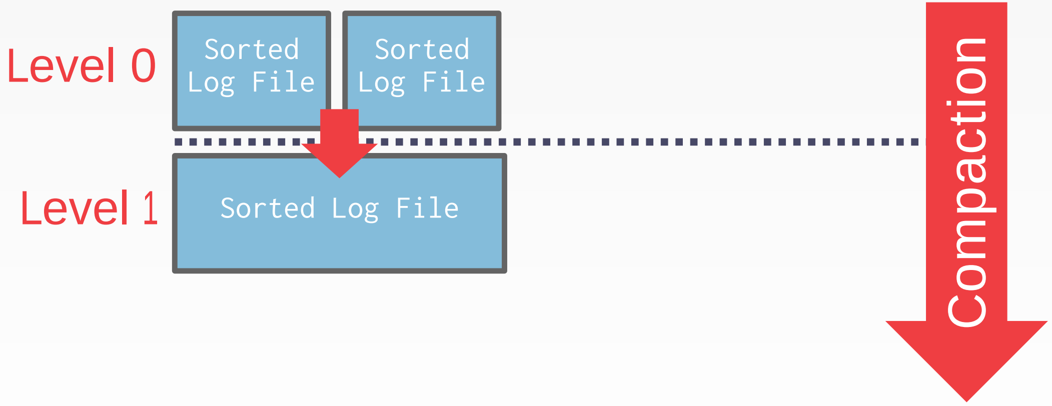
Level Compaction



LOG-STRUCTURED COMPACTION

Compaction coalesces larger log files into smaller files by removing unnecessary records.

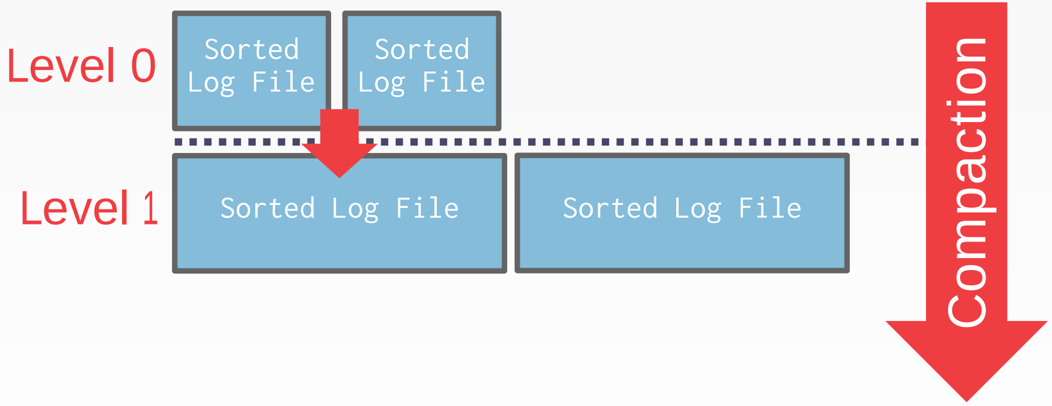
Level Compaction



LOG-STRUCTURED COMPACTION

Compaction coalesces larger log files into smaller files by removing unnecessary records.

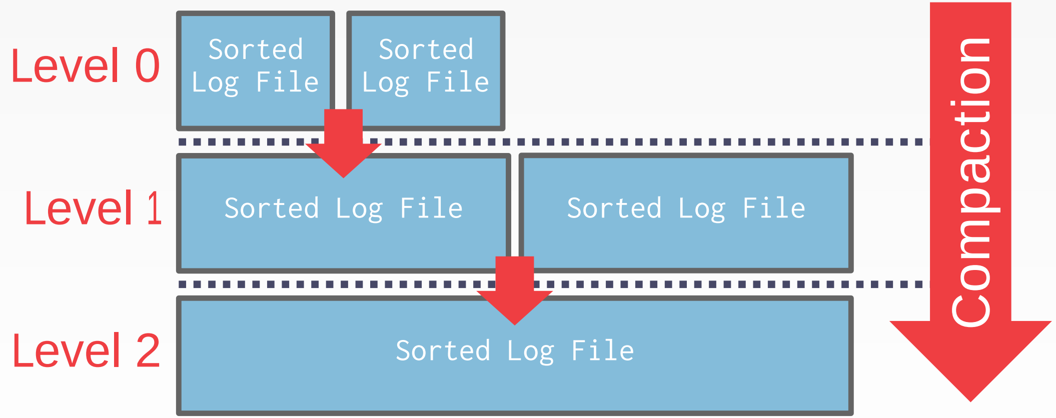
Level Compaction



LOG-STRUCTURED COMPACTION

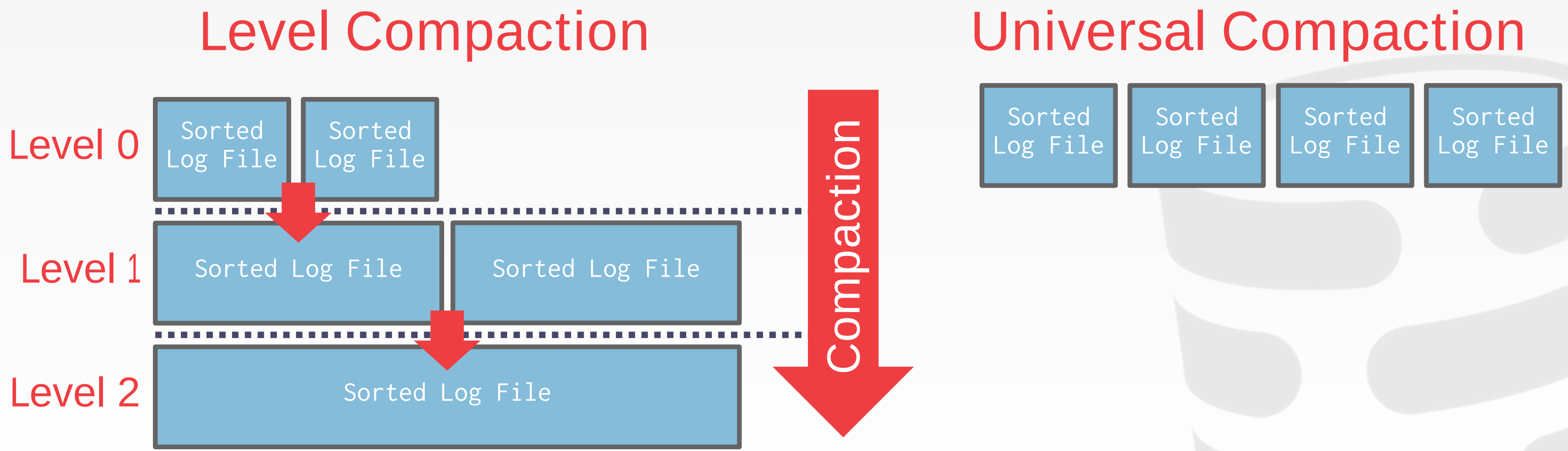
Compaction coalesces larger log files into smaller files by removing unnecessary records.

Level Compaction



LOG-STRUCTURED COMPACTION

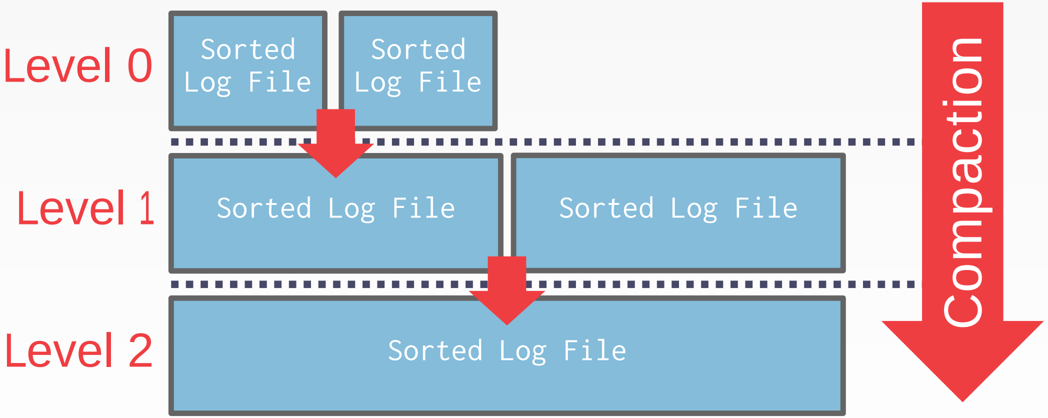
Compaction coalesces larger log files into smaller files by removing unnecessary records.



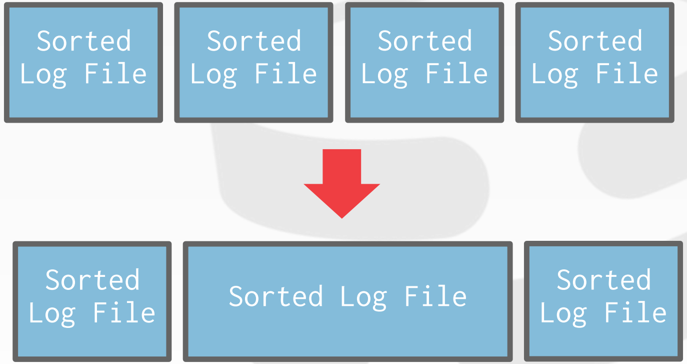
LOG-STRUCTURED COMPACTION

Compaction coalesces larger log files into smaller files by removing unnecessary records.

Level Compaction



Universal Compaction



TODAY'S AGENDA

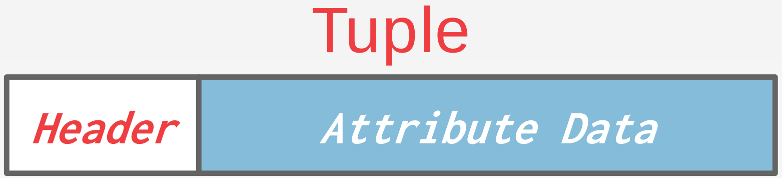
File Storage
Page Layout
Tuple Layout

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



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.

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

Tuple

Header	a	b	c	d	e

```
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,
);
```

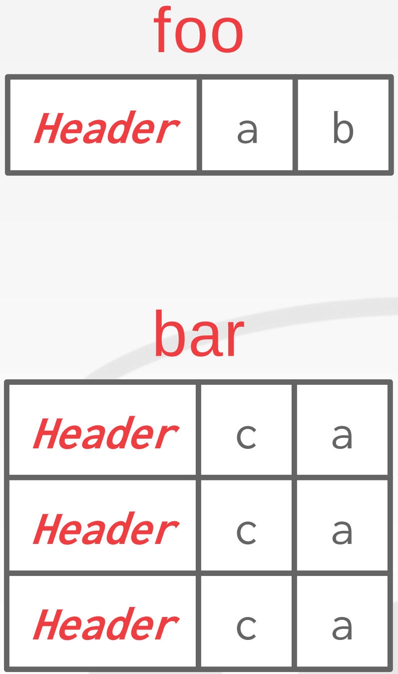
```
CREATE TABLE bar (
  c INT PRIMARY KEY,
  a INT
  ⮱REFERENCES foo (a),
);
```

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.

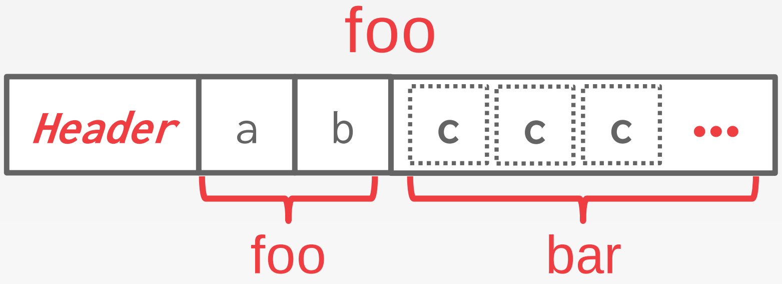


DENORMALIZED TUPLE DATA

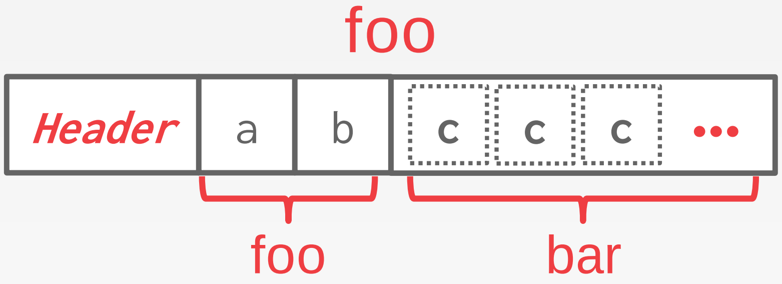
Can physically ***denormalize*** (e.g., "pre join") related tuples and store them together in the same page.

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

→ Can make updates more expensive.



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.

mongoDB

RECORD IDS

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

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

An application cannot rely on these ids to mean anything.

PostgreSQL
CTID (4-bytes)

SQLite
ROWID (8-bytes)

ORACLE
ROWID (10-bytes)

CONCLUSION

Database is organized in pages.

Different ways to track pages.

Different ways to store pages.

Different ways to store tuples.

NEXT CLASS

Value Representation

Storage Models