Carnegie Mellon University

Database Storage

Lecture #06

Database Systems
15-445/15-645
Fall 2017

Andy Pavlo
Computer Science Dept.
Carnegie Mellon Univ.

ADMINISTRIVIA

Homework #2 is due Wednesday September 20th @ 11:59pm

Project #1 is due Wednesday October 4th @ 11:59pm

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

DISK-ORIENTED ARCHITECTURE

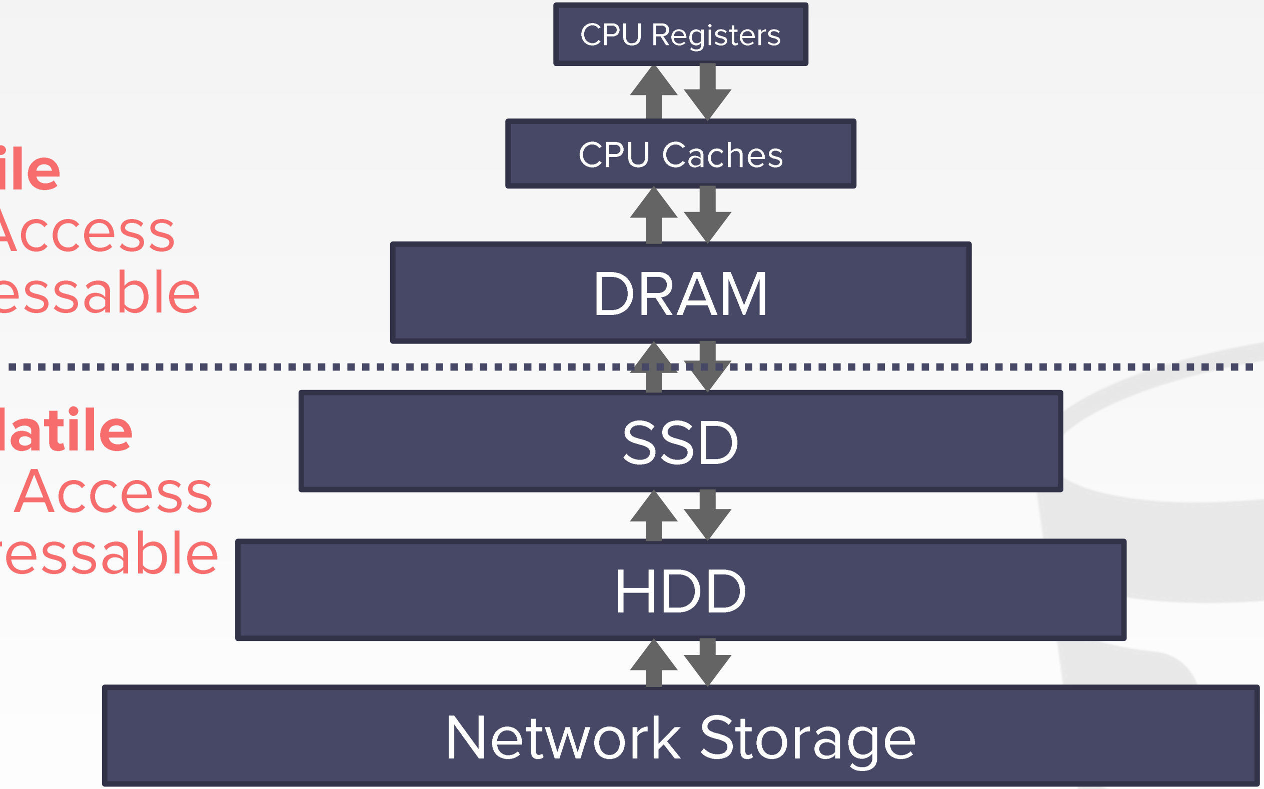
The DBMS assumes that the primary location of the database is on non-volatile disk.

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

STORAGE HIERARCHY

Volatile
Random Access
Byte-Addressable

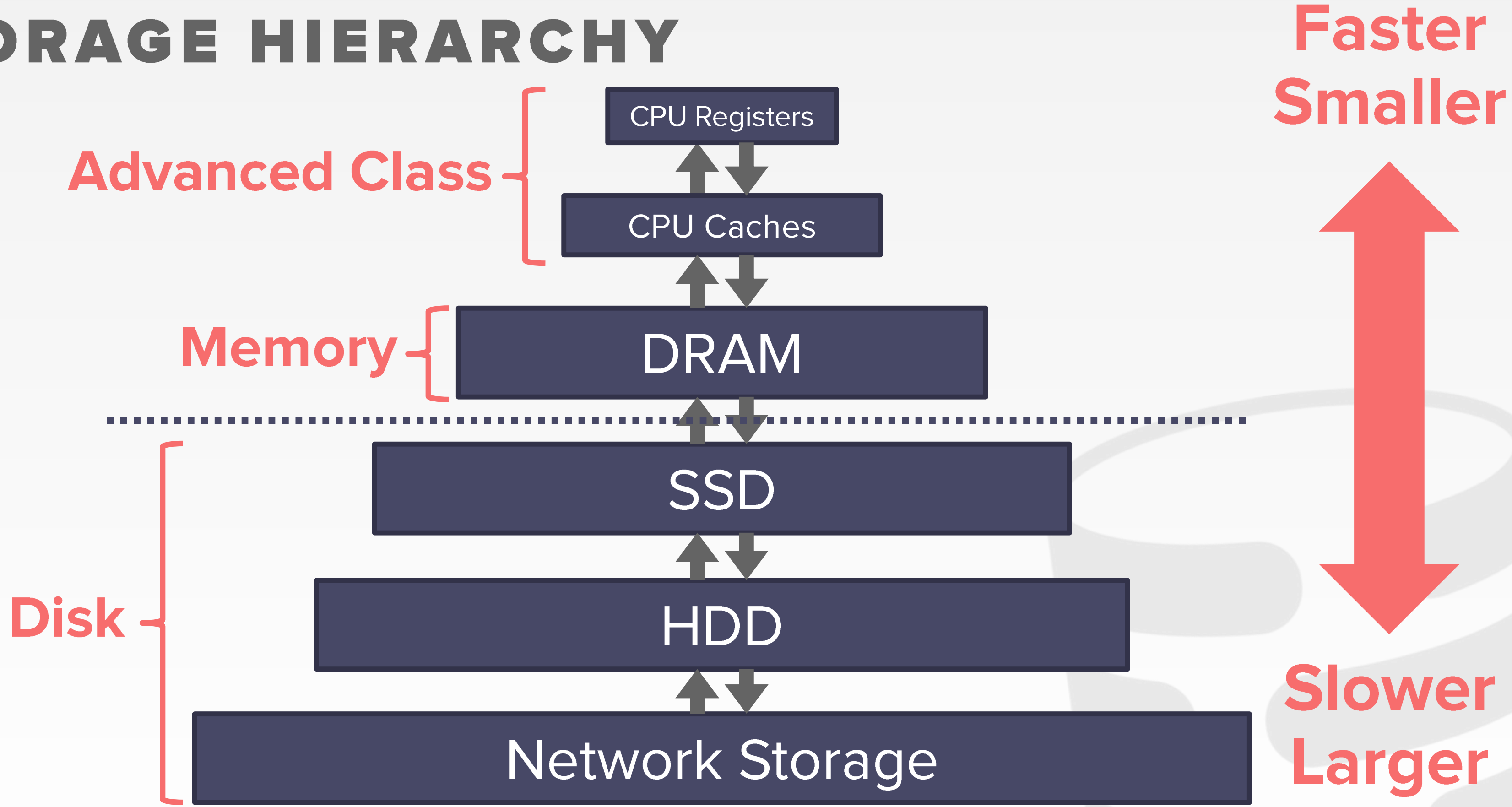
Non-Volatile
Sequential Access
Block-Addressable



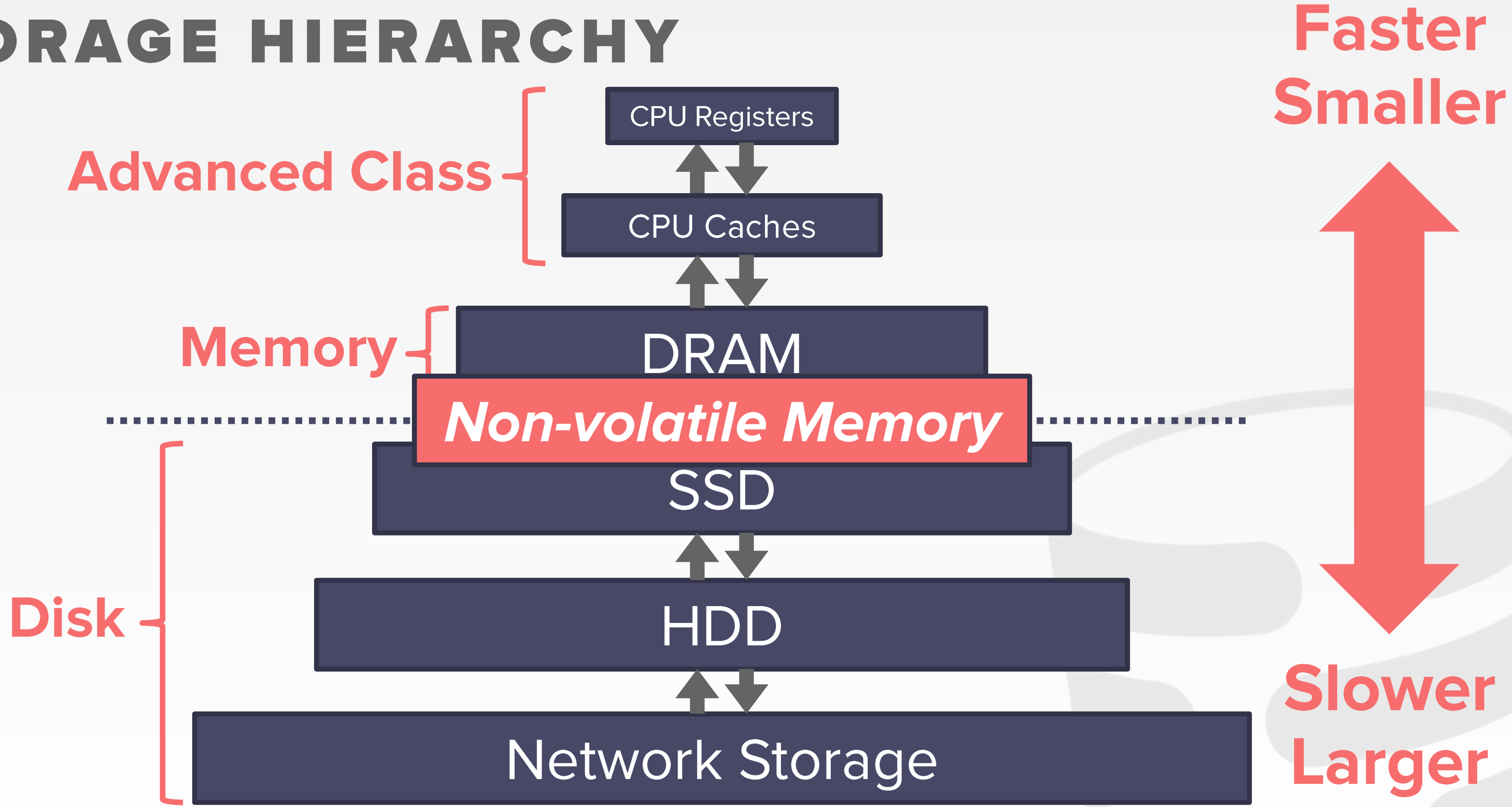
Faster
Smaller

Slower
Larger

STORAGE HIERARCHY



STORAGE HIERARCHY



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.

WHY NOT USE THE OS?

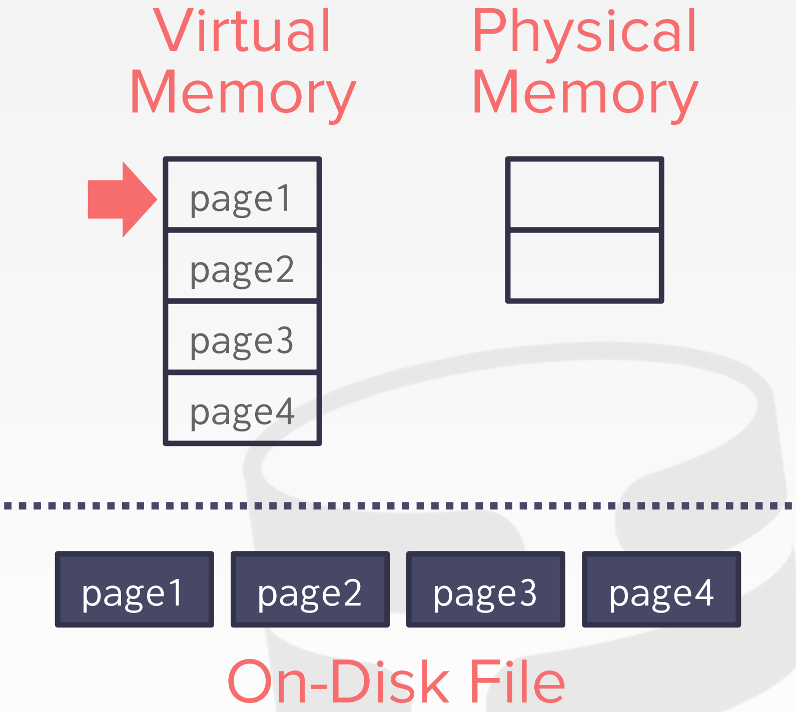
One can use `mmap` to map the contents of a file into a process' address space.

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.

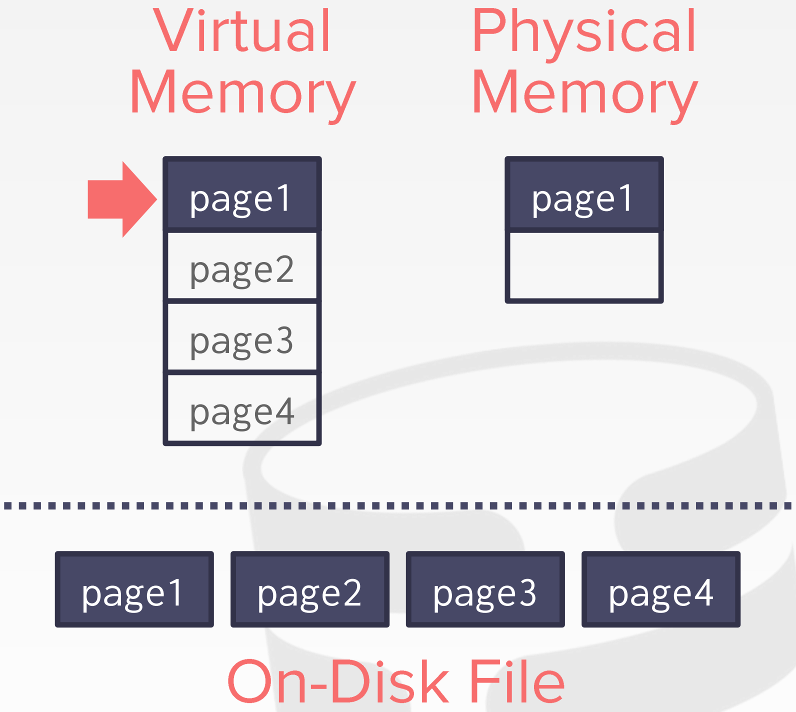
The OS is responsible for moving data for moving the files' pages in and out of memory.



WHY NOT USE THE OS?

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

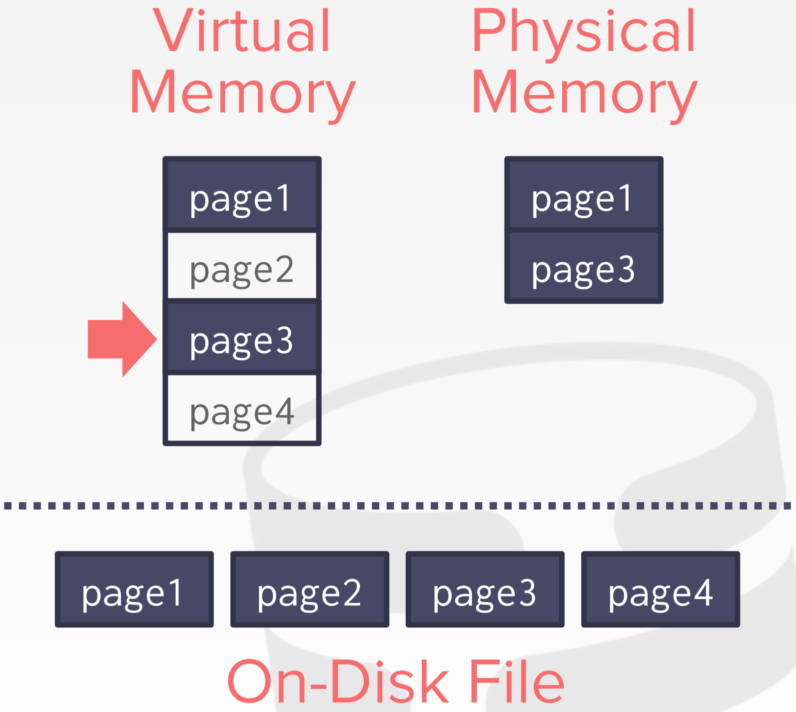
The OS is responsible for moving data for moving the files' pages in and out of memory.



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.

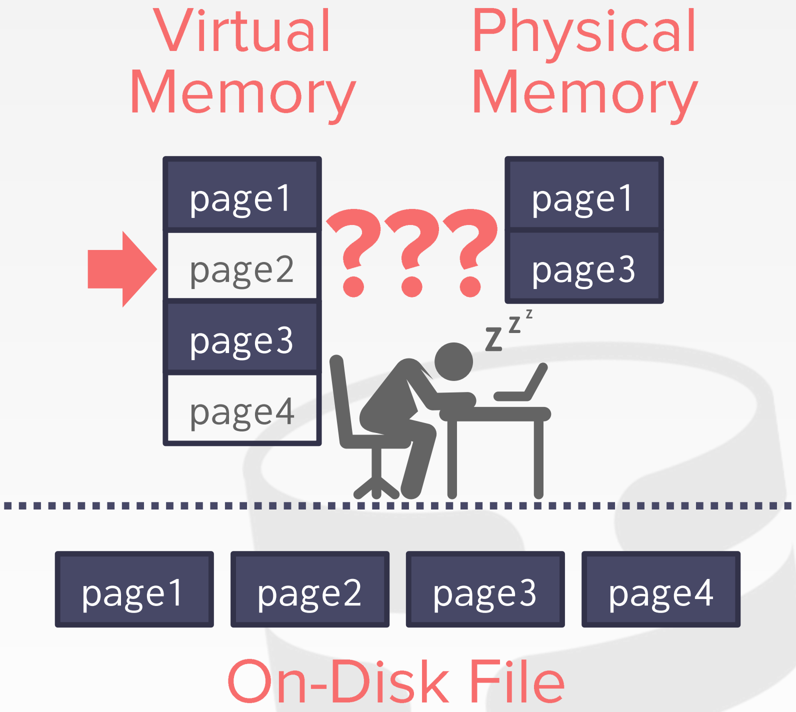


CARNEGIE MELLON DATABASE GROUP

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.



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]

WHY NOT USE THE OS?

What if we allow multiple threads to access the `mmap` files?

This makes things complicated...

WHY NOT USE THE OS?

DBMS (almost) always wants to control things itself.

→ Specialized prefetching
→ Buffer replacement policy
→ Thread/process scheduling
→ Flushing data to disk

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.

SEQUENTIAL VS. RANDOM ACCESS

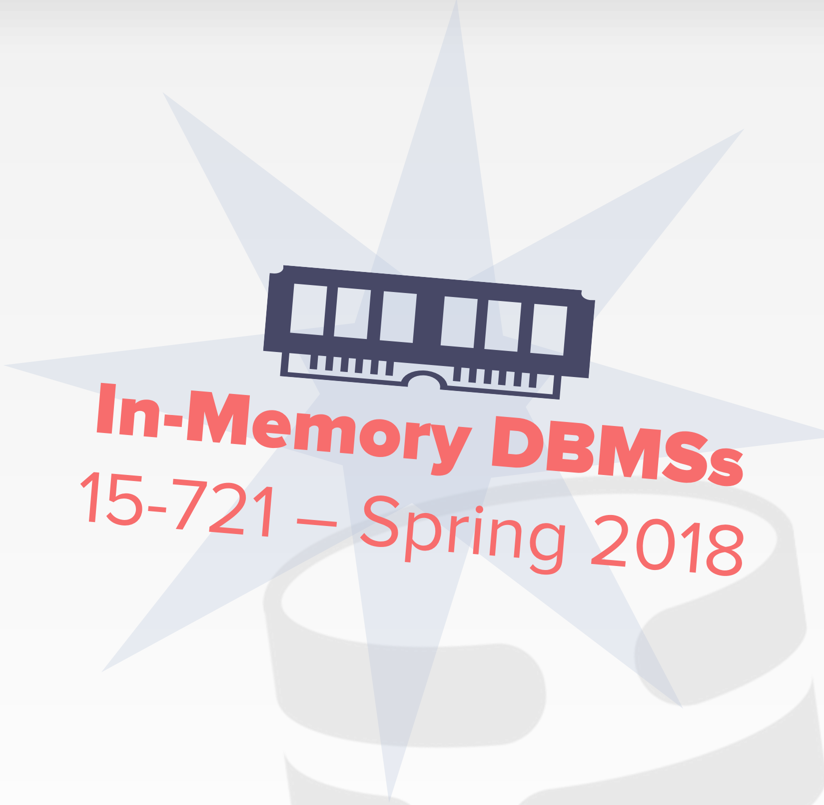
Random access on an HDD is slower than sequential access.

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 <u>extent</u>.

Not always necessary now…



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.

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.

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 (4KB)
→ Database Page (1-16KB)

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

DATABASE PAGES

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

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)

PAGE STORAGE ARCHITECTURE

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

→ Heap File Organization
→ Sequential / Sorted File Organization
→ Hashing File Organization
→ Log-Structured File Organization

DATABASE HEAP

A heap file is an unordered collection of pages where tuples that are stored in random order.

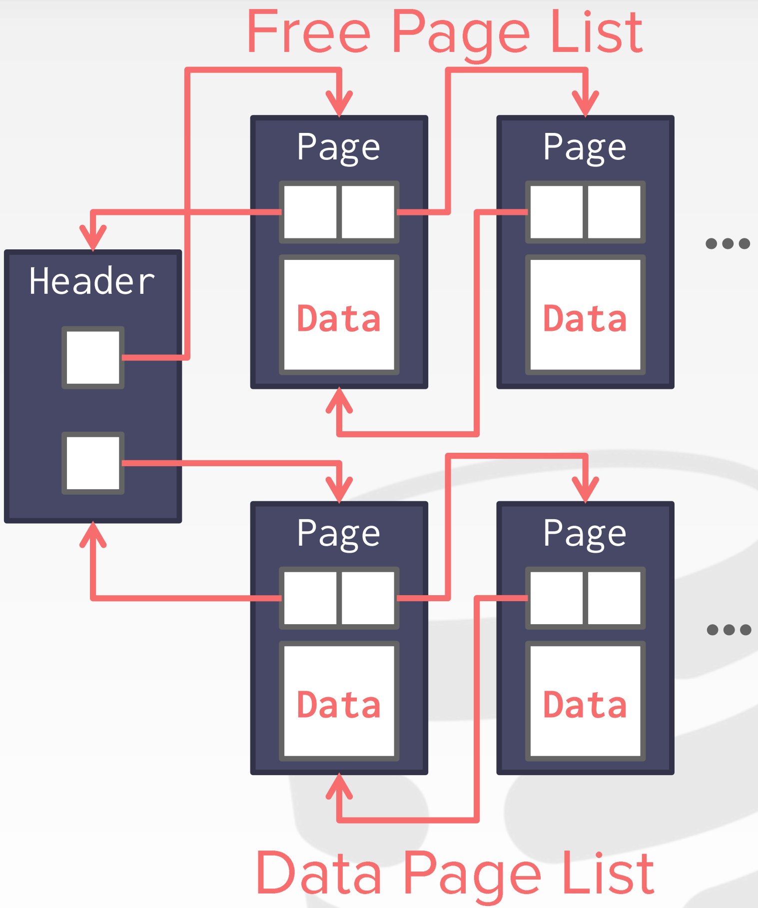
The DBMS needs a way to find a page on disk for a given page id.

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.

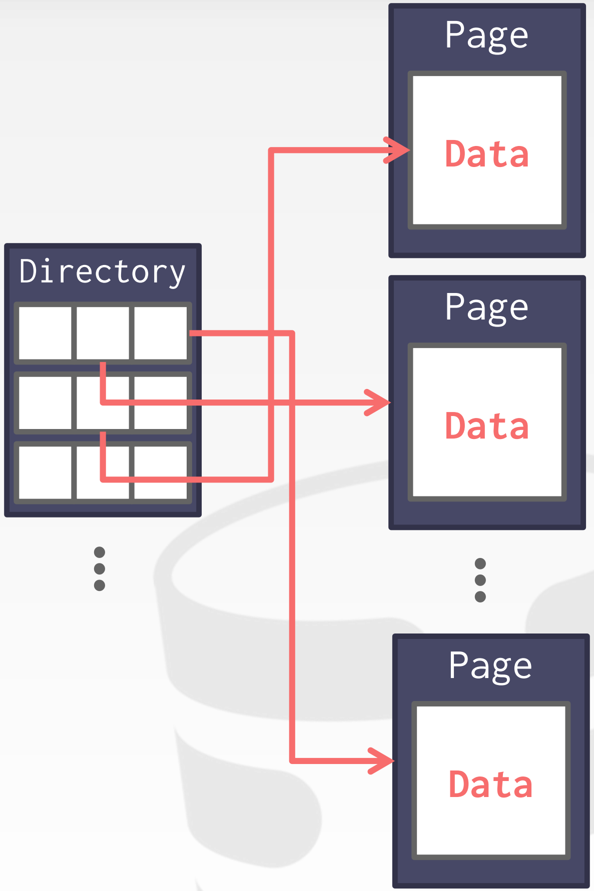


HEAP FILE: PAGE DIRECTORY

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.

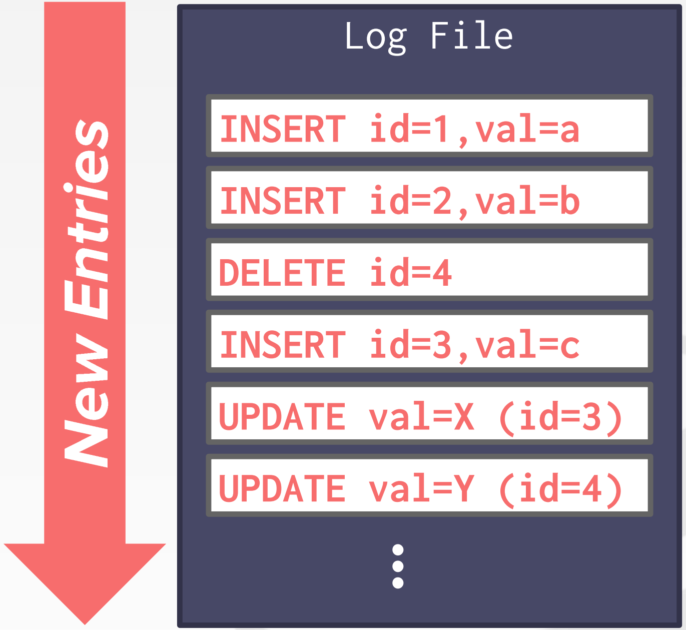


CARNEGIE MELLON DATABASE GROUP

LOG-STRUCTURED FILE ORGANIZATION

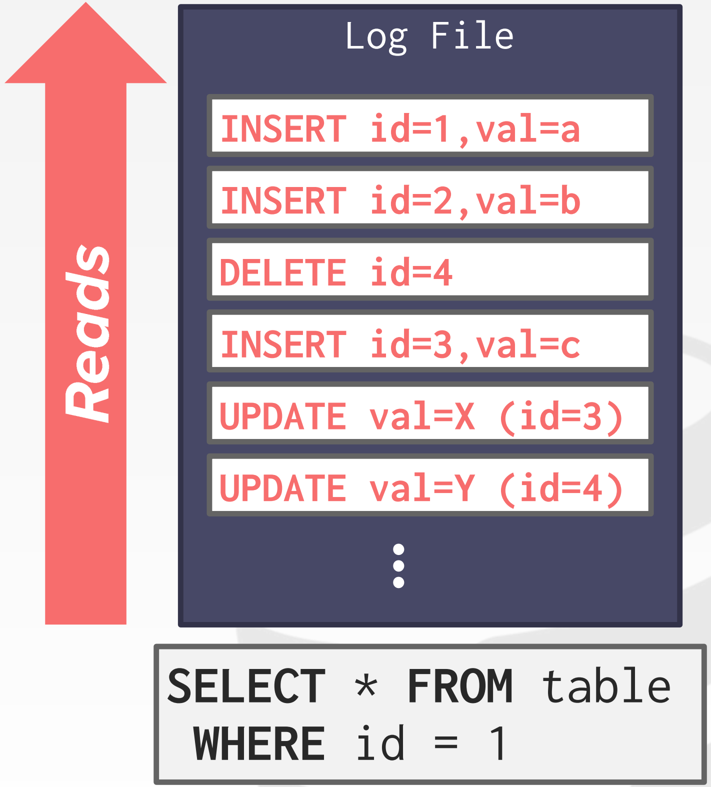
Instead of storing tuples in pages, the DBMS only stores log records.

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.



LOG-STRUCTURED FILE ORGANIZATION

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

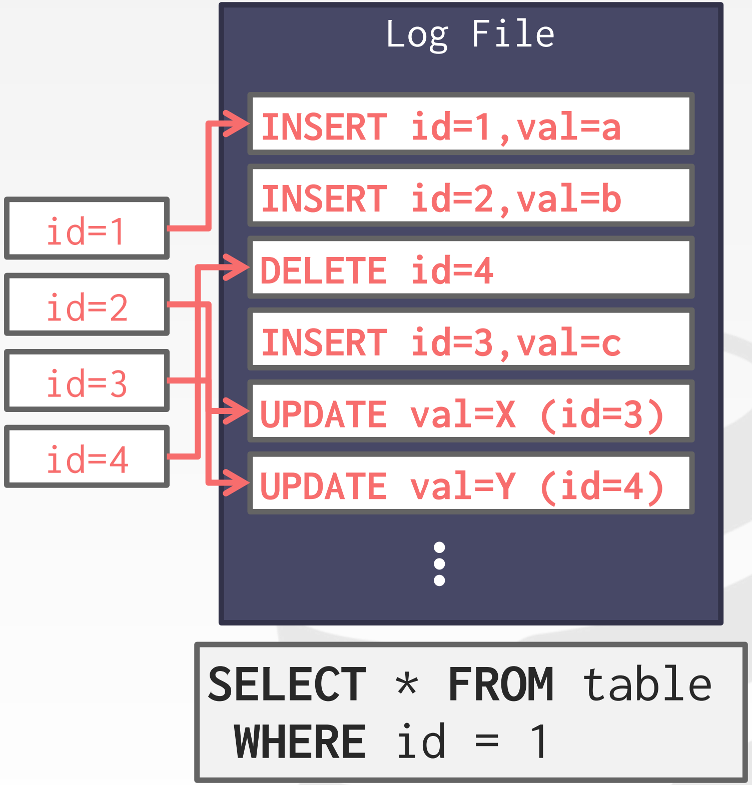


CARNEGIE MELLON DATABASE GROUP

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.

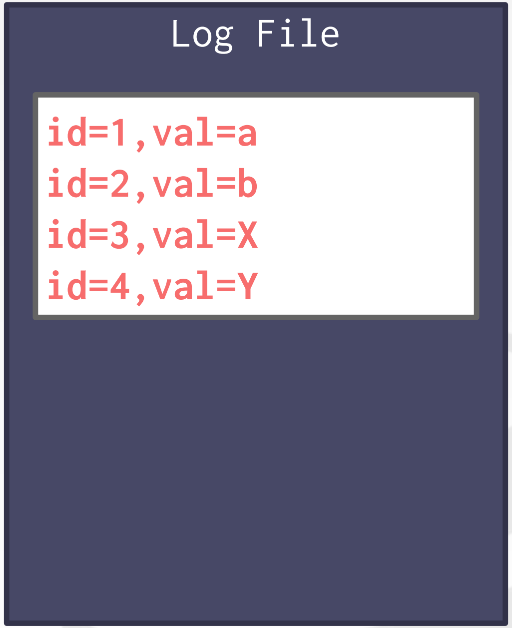


LOG-STRUCTURED FILE ORGANIZATION

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.



PAGE LAYOUT

We now need to understand how tuples on pages.

This discussion does not apply to the log-structured organization.

PAGE HEADER

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

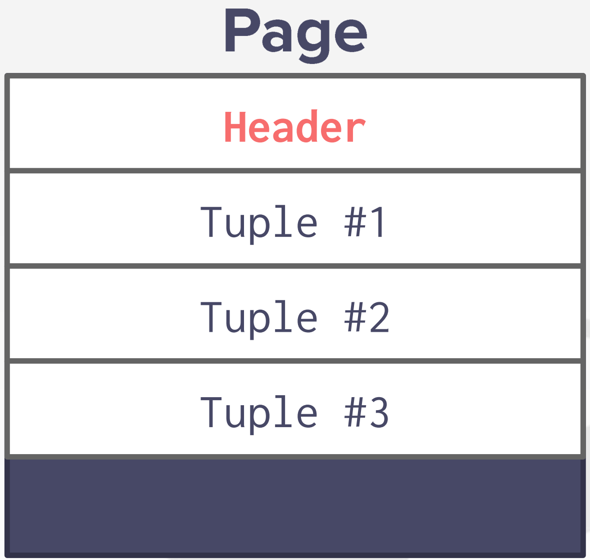
→ Page Size
→ Checksum
→ DBMS Version
→ Transaction Visibility

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

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.



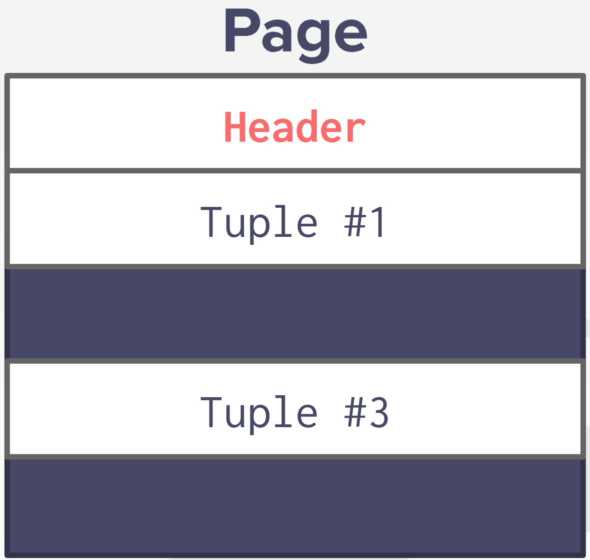
CARNEGIE MELLON
DATABASE GROUP

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.

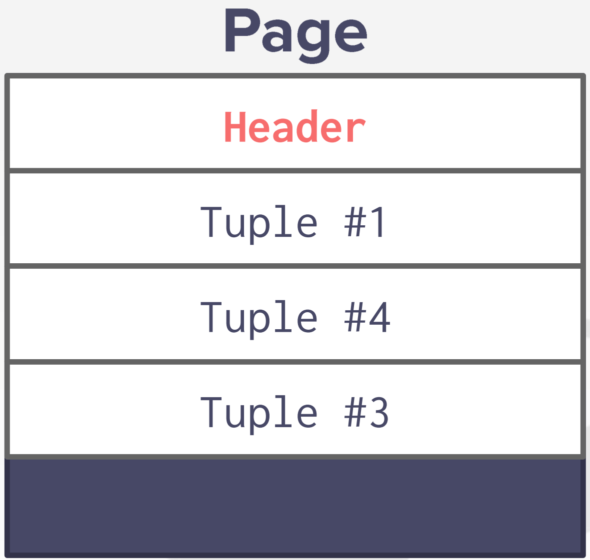
→ What happens if we delete a tuple?



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?

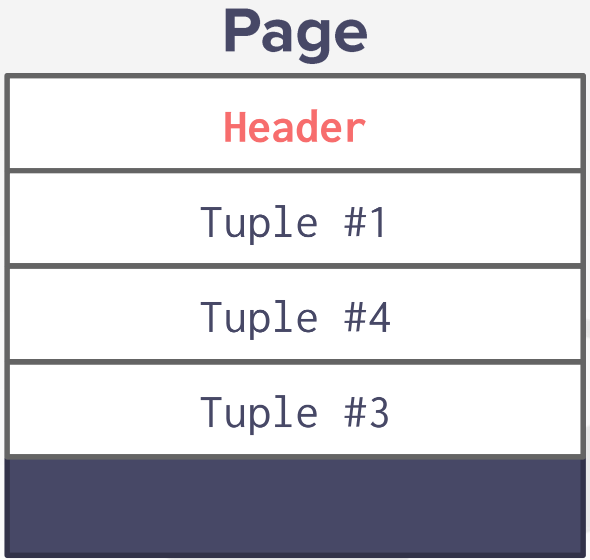


TUPLE STORAGE

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?



CARNEGIE MELLON DATABASE GROUP

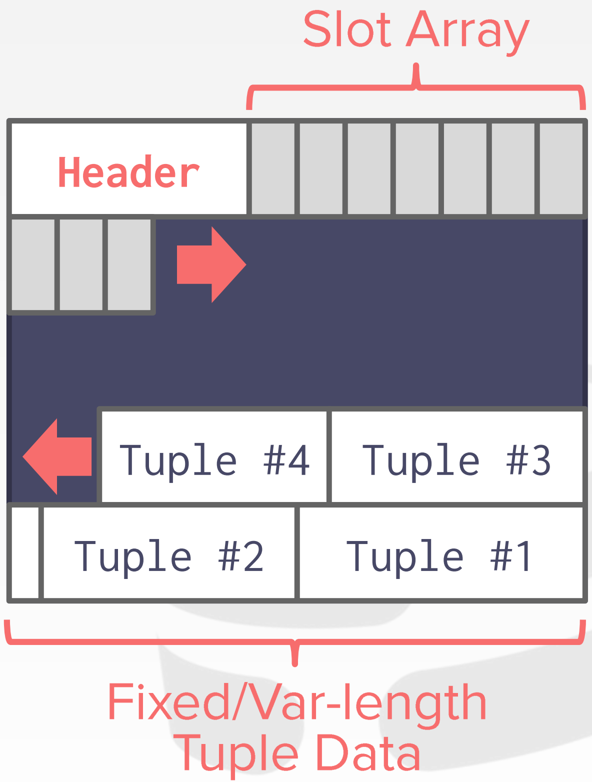
SLOTTED PAGES

The most common layout scheme is called slotted pages.

The page maps "slots" to offsets.

The header keeps track of:

→ The # of used slots
→ The offset of the starting location of the last slot used.

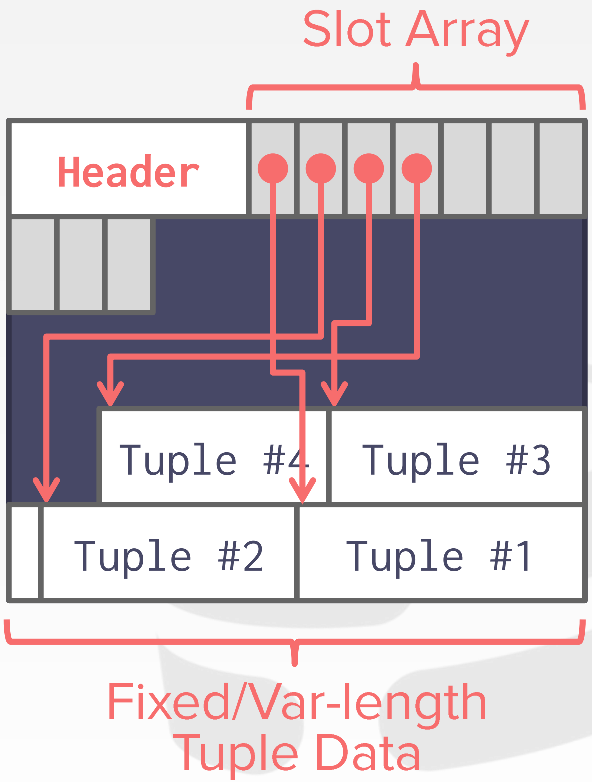


SLOTTED PAGES

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

The page maps "slots" to offsets.

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



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 (Concurrency Control)
→ Bit Map for NULL values.

Note that 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.

Tuple

Header	a	b	c	d	e

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

DATA REPRESENTATION

INTEGER/BIGINT/SMALLINT/TINYINT
→ C/C++ Representation

FLOAT/REAL vs. **NUMERIC/DECIMAL**
→ IEEE-754 Standard / Fixed-point Decimals

VARCHAR/VARBINARY/TEXT/BLOB
→ Header with length, followed by data bytes.

TIME/DATE/TIMESTAMP
→ 32/64-bit integer of (micro)seconds since Unix epoch

VARIABLE PRECISION NUMBERS

Inexact, variable-precision numeric type that uses the "native" C/C++ types.

Store directly as specified by **IEEE-754**.

Typically faster than arbitrary precision numbers.

→ Example: FLOAT, REAL/DOUBLE

VARIABLE PRECISION NUMBERS

Rounding Example

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    float x = 0.1;
    float y = 0.2;
    printf("x+y = %.20f\n", x+y);
    printf("0.3 = %.20f\n", 0.3);
}
```

Output

```
x+y = 0.30000001192092895508
0.3 = 0.29999999999999998890
```

FIXED PRECISION NUMBERS

Numeric data types with arbitrary precision and scale. Used when round errors are unacceptable.

→ Example: NUMERIC, DECIMAL

Typically stored in a exact, variable-length binary representation with additional meta-data.

→ Like a VARCHAR but not stored as a string

POSTGRES: NUMERIC

of Digits

Weight of 1st Digit

Scale Factor

Positive/Negative/NaN

Digit Storage

```
typedef unsigned char NumericDigit;
typedef struct {
  int ndigits;
  int weight;
  int scale;
  int sign;
  NumericDigit *digits;
} numeric;
```

POSTGRE

o

Weight of

Scale

Positive/Negati

Digit

NumericDigit;

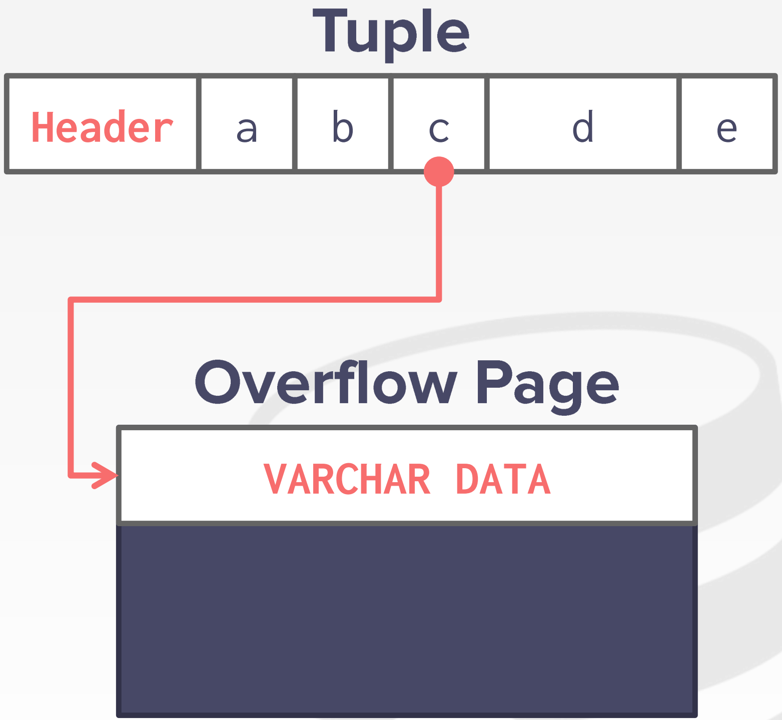
;

```
/* ----------
 * add_var() -
 *
 *  Full version of add functionality on variable level (handling signs).
 *  result might point to one of the operands too without danger.
 * ----------
 */
int
PGTYPESnumeric_add(numeric *var1, numeric *var2, numeric *result)
{
    /*
     * Decide on the signs of the two variables what to do
     */
    if (var1->sign == NUMERIC_POS)
    {
        if (var2->sign == NUMERIC_POS)
        {
            /*
             * Both are positive result = +(ABS(var1) + ABS(var2))
             */
            if (add_abs(var1, var2, result) != 0)
                return -1;
            result->sign = NUMERIC_POS;
        }
        else
        {
            /*
             * var1 is positive, var2 is negative Must compare absolute values
             */
            switch (cmp_abs(var1, var2))
            {
                case 0:
                    /* ----------
                     * ABS(var1) == ABS(var2)
                     * result = ZERO
                     * ----------
                     */
                    zero_var(result);
                    result->rscale = Max(var1->rscale, var2->rscale);
                    result->dscale = Max(var1->dscale, var2->dscale);
                    break;

                case 1:
                    /* ----------
                     * ABS(var1) > ABS(var2)
                     * result = +(ABS(var1) - ABS(var2))
                     * ----------
                     */
                    if (sub_abs(var1, var2, result) != 0)
                        return -1;
                    result->sign = NUMERIC_POS;
                    break;

                case -1:
                    /* ----------
                     * ABS(var1) < ABS(var2)
                     * result = -(ABS(var2) - ABS(var1))
```

LARGE VALUES

Most DBMSs don't allow a tuple to exceed the size of a single page.

To store values that are larger than a page, the DBMS uses separate overflow storage pages.
→ Postgres: TOAST (>2KB)
→ MySQL: Overflow (>½ size of page)



CONCLUSION

Database is organized in pages.

Different ways to track pages.

Different ways to store pages.

Different ways to store tuples.

Log-structured organization is a different beast.

DATABASE STORAGE

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

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

← **Next Class**