Carnegie Mellon University

Database Storage Part II

Lecture #04

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

Important: Go get a flu vaccine.

DISK-ORIENTED ARCHITECTURE

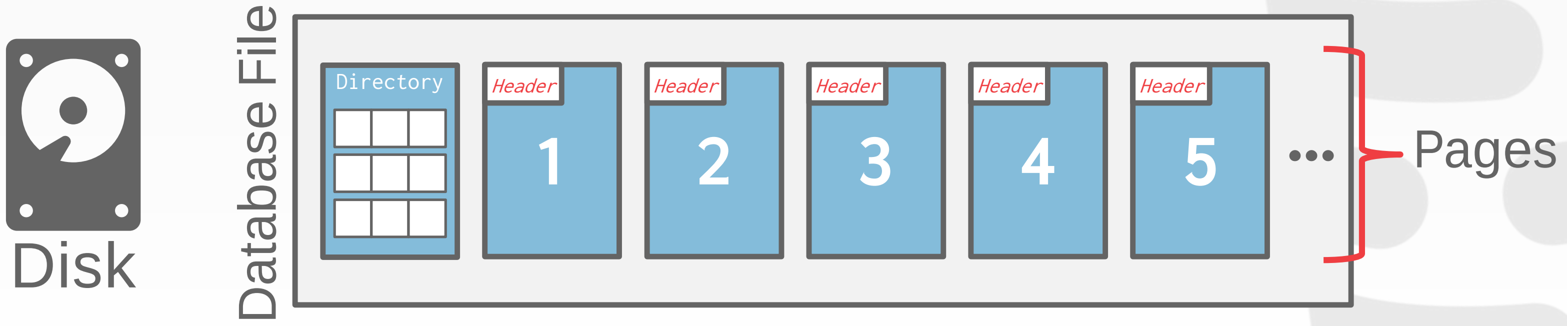
The DBMS assumes that the primary storage location of the database is on non-volatile disk.

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

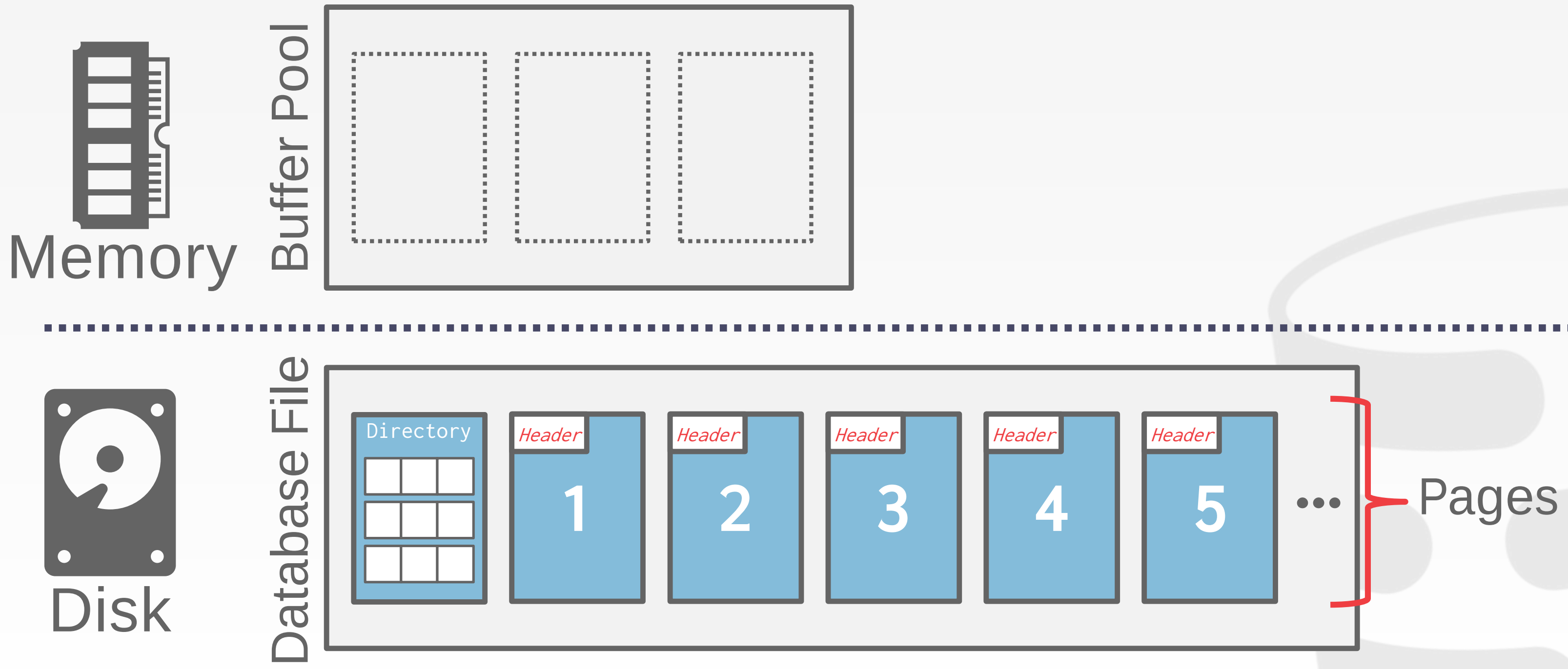
Disk database system where the architecture is predicated on the assumption that the primary search location of the database is on disk

DISK-ORIENTED DBMS

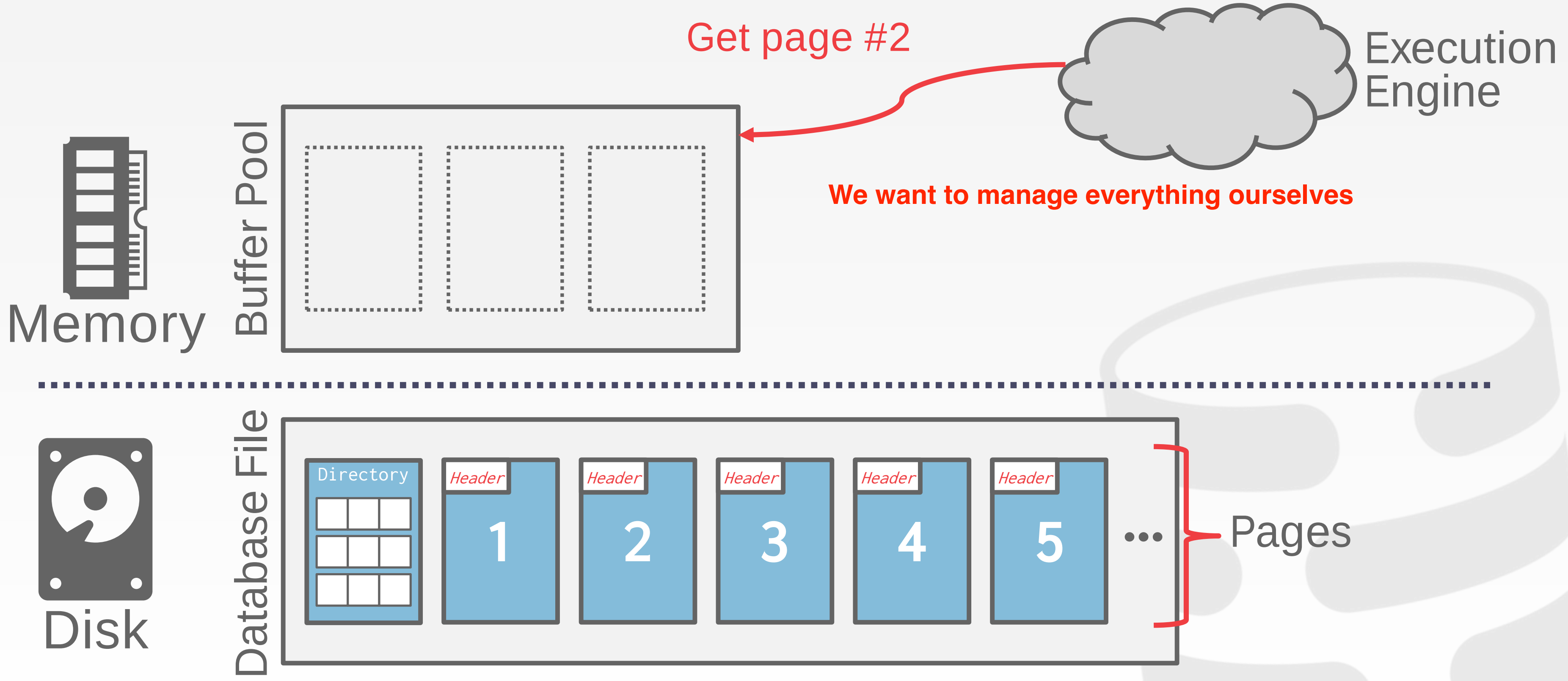
whenever needs to read on one of these tuples, it's going to assume that it is not in memory and therefore it is gonna go out the disk to get it copy it into memory and then it can do whatever it is that it wants



DISK-ORIENTED DBMS

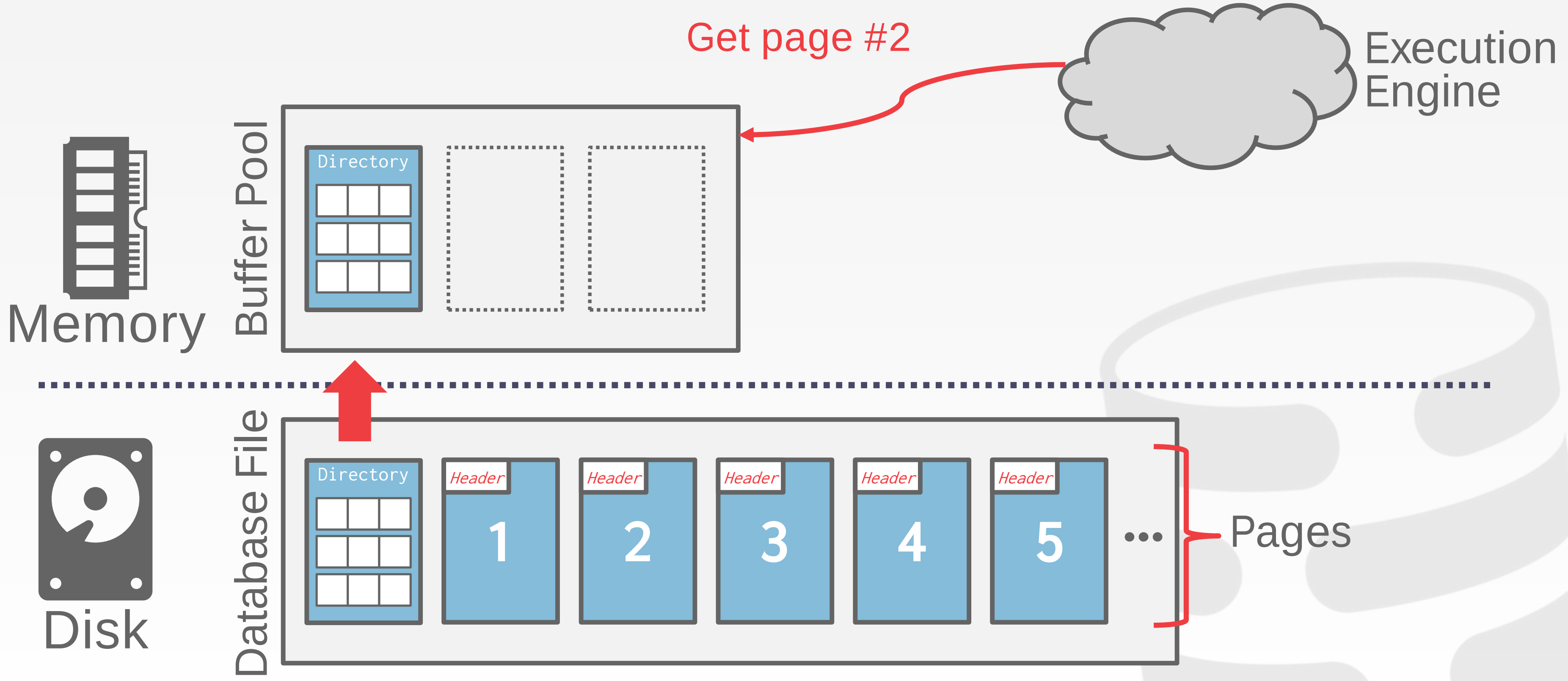


DISK-ORIENTED DBMS

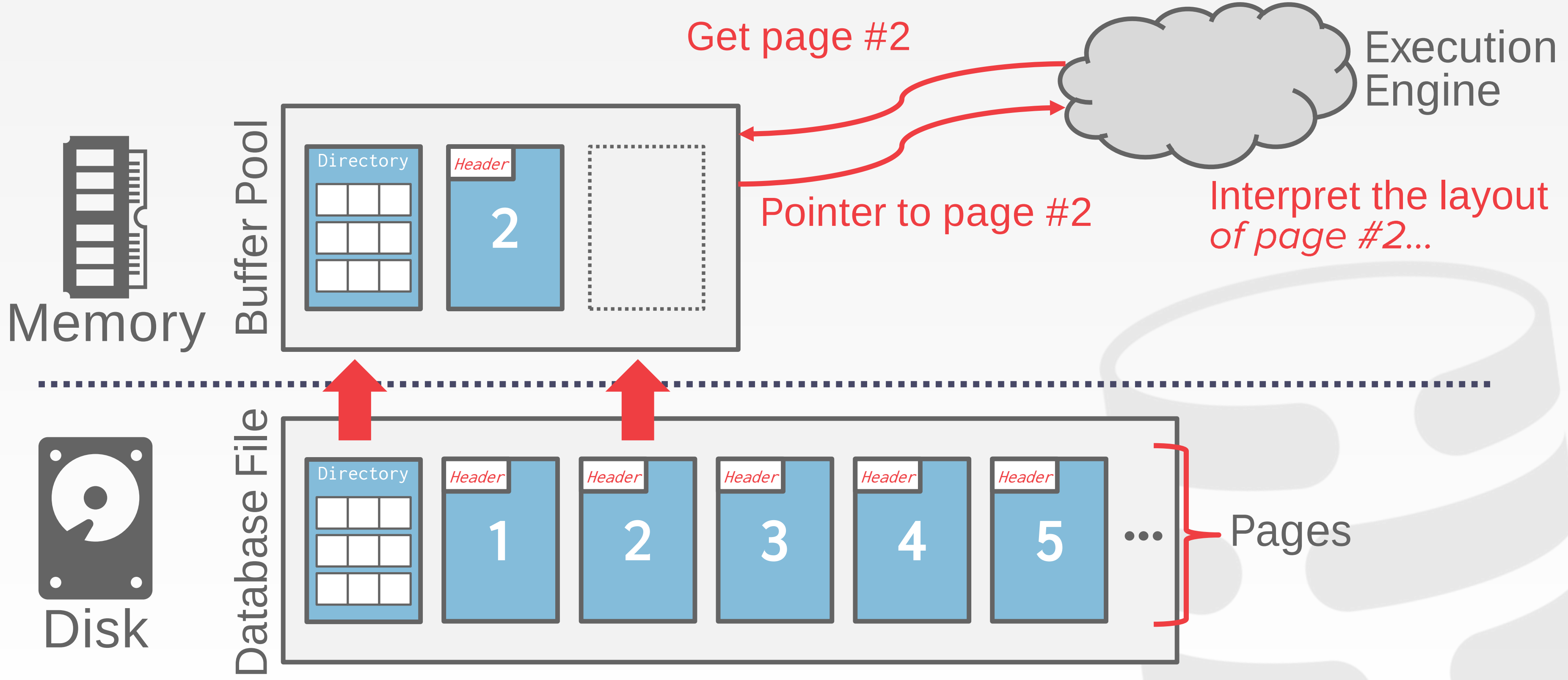


We want to manage everything ourselves

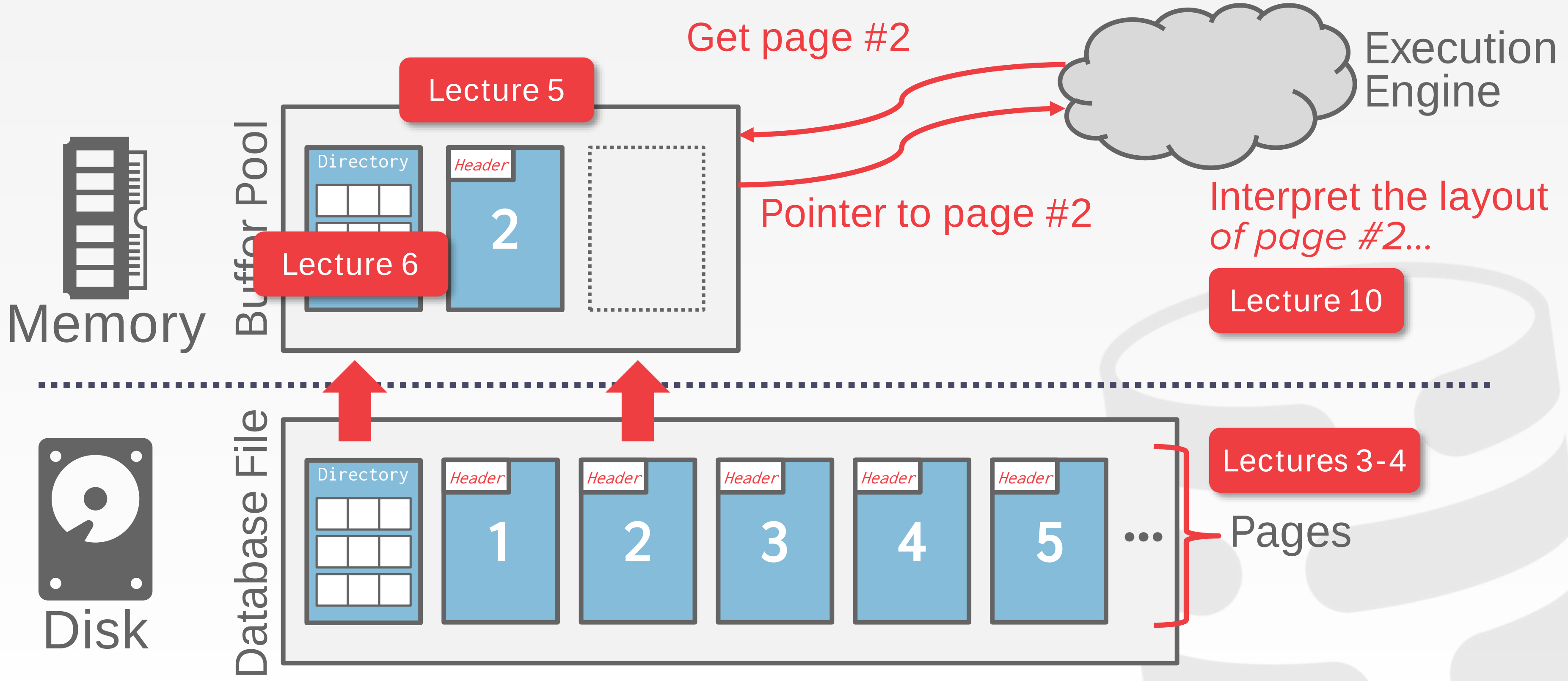
DISK-ORIENTED DBMS



DISK-ORIENTED DBMS



DISK-ORIENTED DBMS



TODAY'S AGENDA

Data Representation
System Catalogs
Storage Models

TUPLE STORAGE

A tuple is essentially a sequence of bytes.

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

The DBMS's catalogs contain the schema information about tables that the system uses to figure out the tuple's layout.

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

If we do it by ourselves instead of hardware, we may have better accuracy, but we're gonna pay a performance penalty

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**

The hardware can’t actually accurately represent decimals or floating-point numbers

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
```

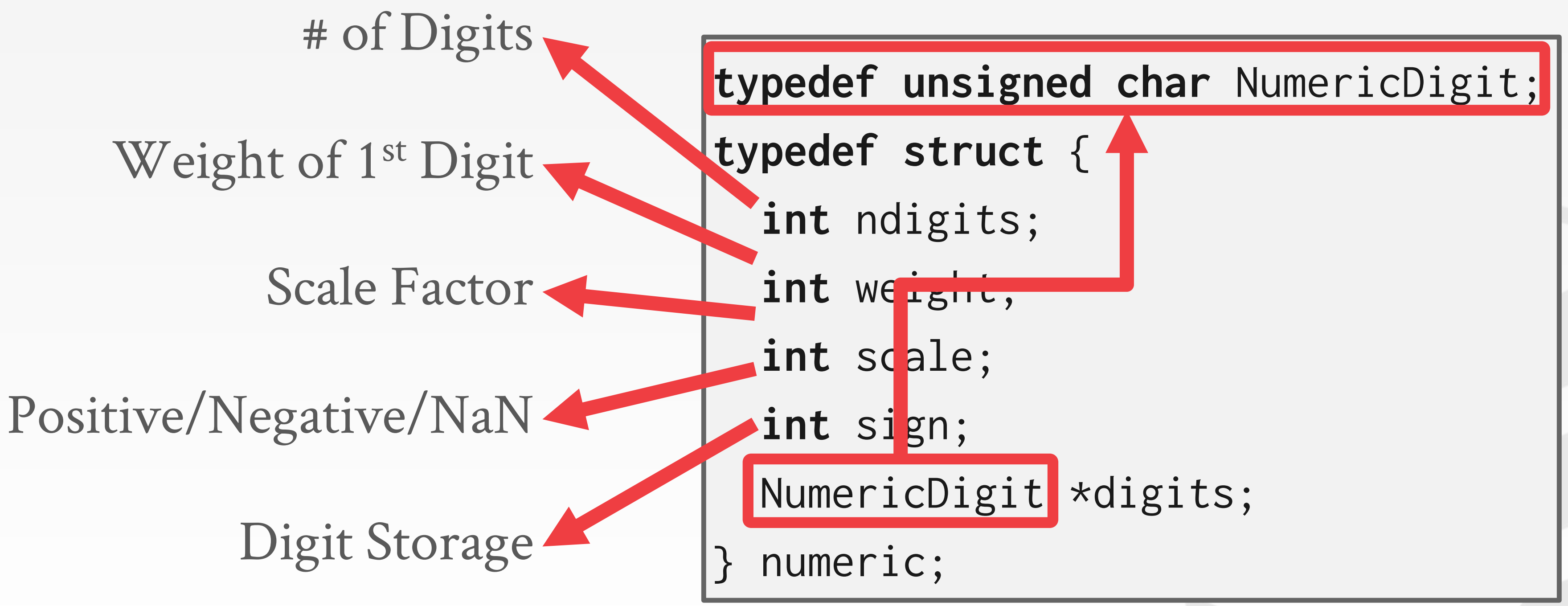
FIXED PRECISION NUMBERS

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

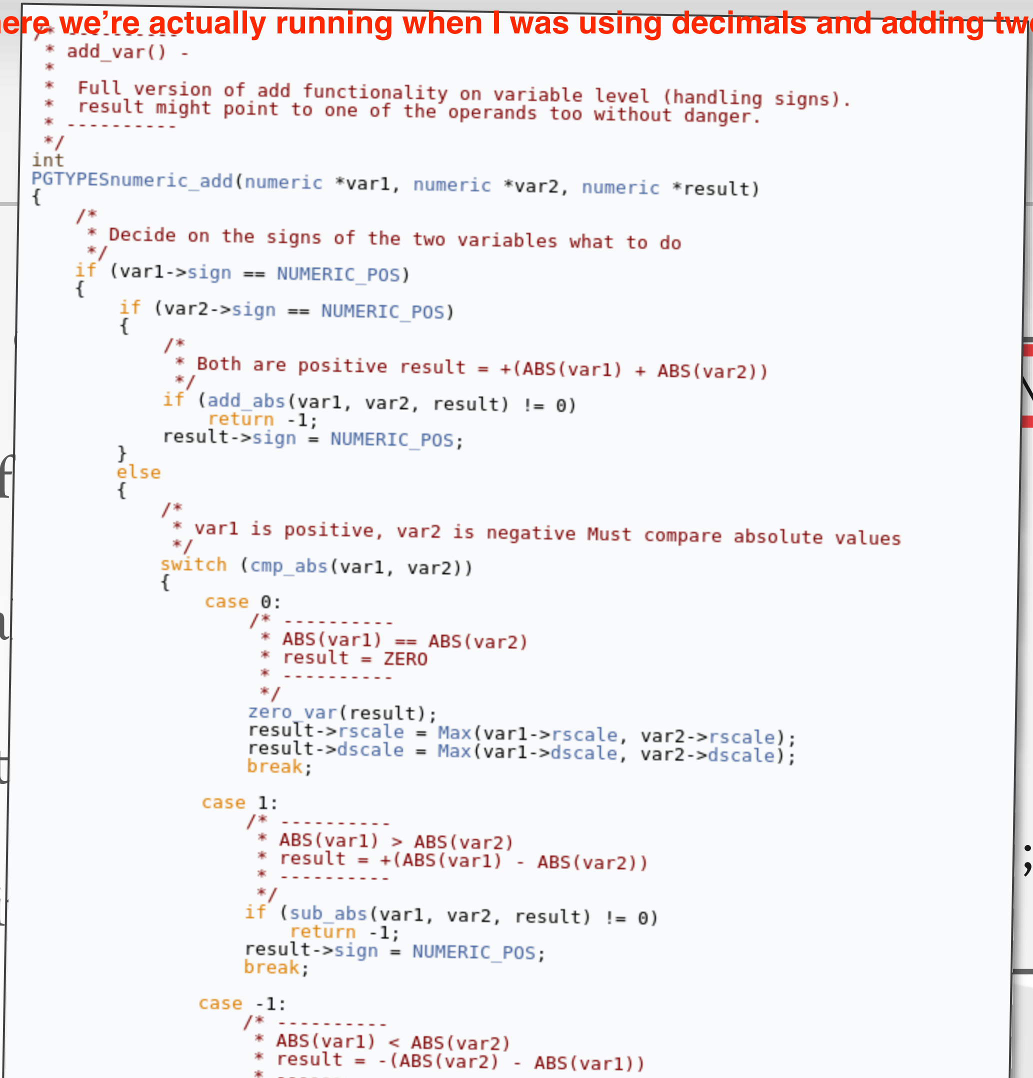
Demo: Postgres

POSTGRES: NUMERIC



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

This is the code where we're actually running when I was using decimals and adding two numbers together



```
/* ----------
 * 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))
```

#

Weight of

Sca

Positive/Negat

Digi

```
NumericDigit;
```

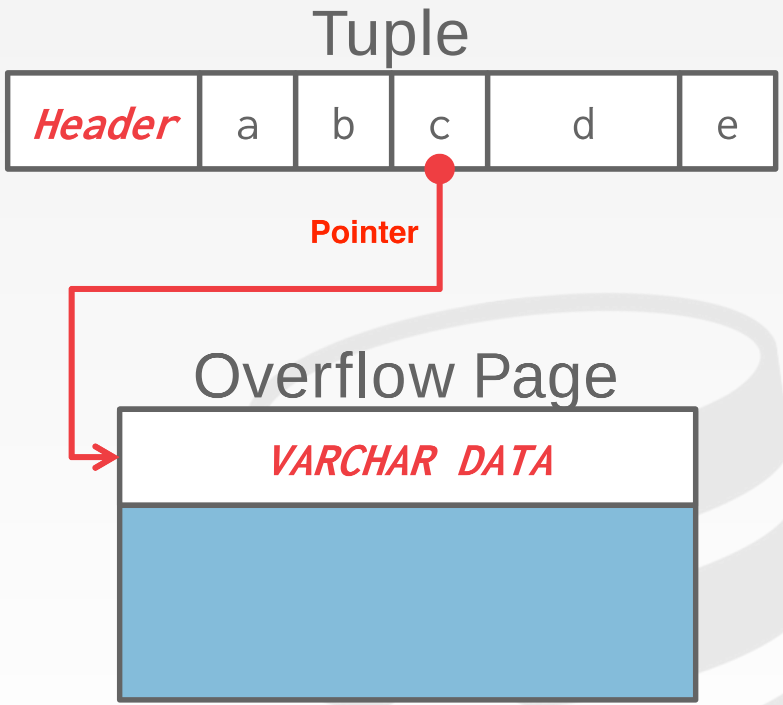
;

CARNEGIE MELLON DATABASE GROUP

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)



EXTERNAL VALUE STORAGE

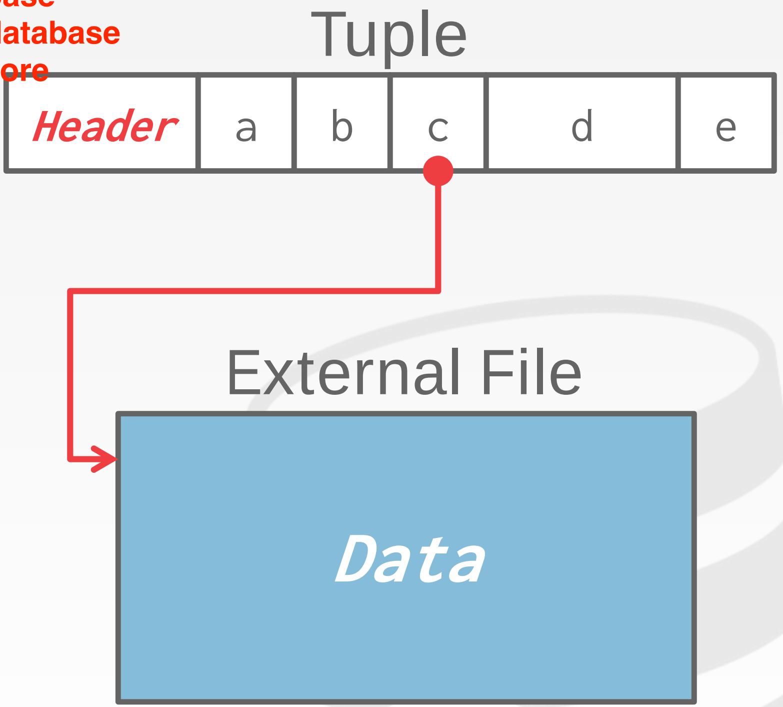
Say I have a photo application, and my database is storing photo information
Instead of storing the really large photo files inside of my database
I can just have them strewn across a directory and then inside my database
system just has a pointer to where that file is that it wants to store

Some systems allow you to store a really large value in an external file. Treated as a **BLOB** type.

→ Oracle: **BFILE** data type
→ Microsoft: **FILESTREAM** data type

The DBMS **cannot** manipulate the contents of an external file.

→ No durability protections.
→ No transaction protections.



SYSTEM CATALOGS

A DBMS stores meta-data about databases in its internal catalogs.
→ Tables, columns, indexes, views
→ Users, permissions
→ Internal statistics

Almost every DBMS stores their a database's catalog in itself.
→ Wrap object abstraction around tuples.
→ Specialized code for "bootstrapping" catalog tables.

SYSTEM CATALOGS

You can query the DBMS's internal **INFORMATION_SCHEMA** catalog to get info about the database.

→ ANSI standard set of read-only views that provide info about all of the tables, views, columns, and procedures in a database

DBMSs also have non-standard shortcuts to retrieve this information.

ACCESSING TABLE SCHEMA

List all of the tables in the current database:

```
SELECT *                                        SQL-92
  FROM INFORMATION_SCHEMA.TABLES
 WHERE table_catalog = '<db name>';
```

```
\d;                                   Postgres
```

```
SHOW TABLES;                          MySQL
```

```
.tables;                              SQLite
```

ACCESSING TABLE SCHEMA

List all of the columns in the student table:

SQL-92

```
SELECT *
  FROM INFORMATION_SCHEMA.TABLES
 WHERE table_name = 'student'
```

Postgres

```
\d student;
```

MySQL

```
DESCRIBE student;
```

SQLite

```
.schema student;
```

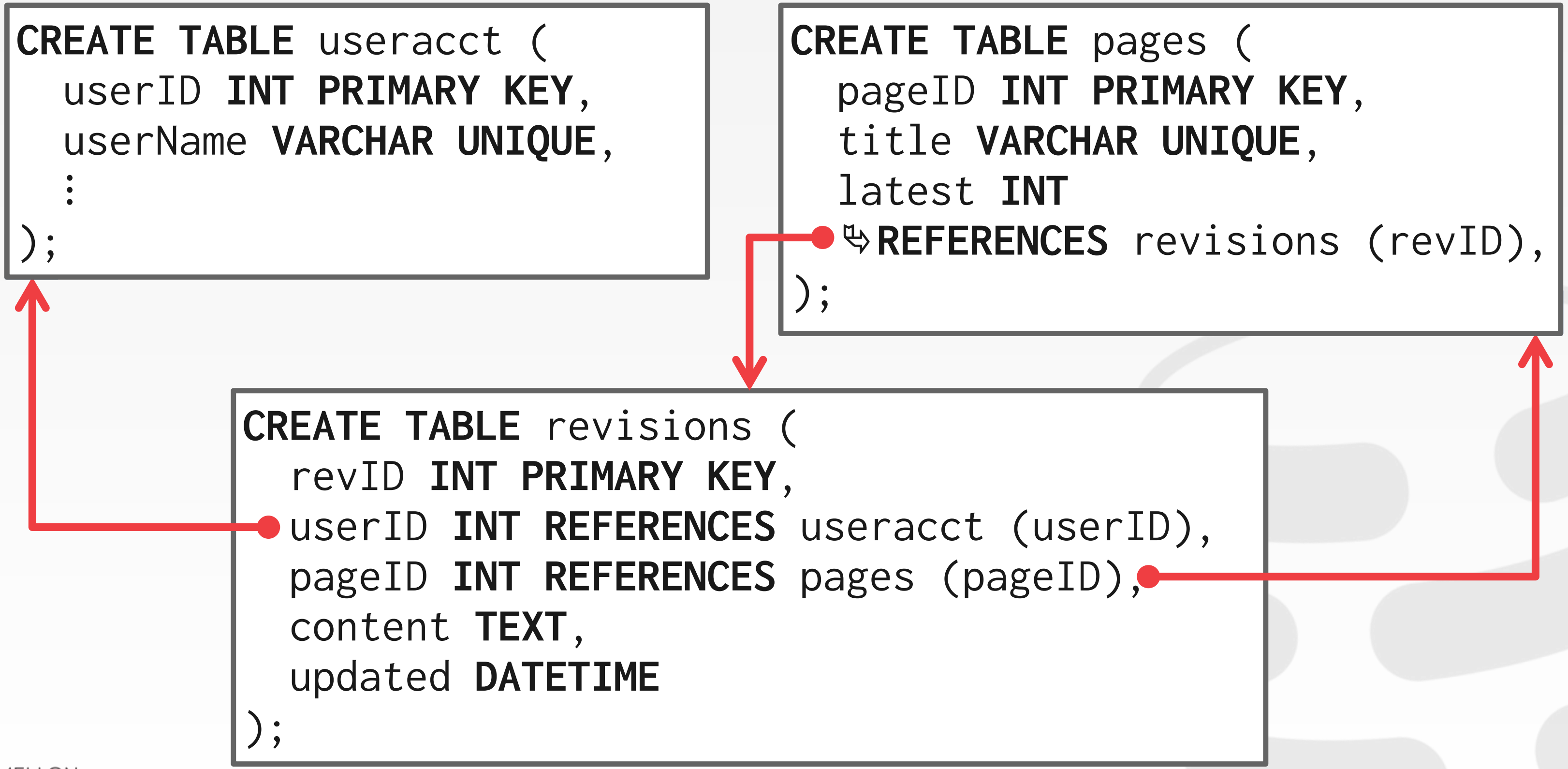
OBSERVATION

Relation model is a high-level logical concept

The relational model does **not** specify that we have to store all of a tuple's attributes together in a single page.

This may **not** actually be the best layout for some workloads…

WIKIPEDIA EXAMPLE



```
CREATE TABLE useracct (
  userID INT PRIMARY KEY,
  userName VARCHAR UNIQUE,
  ⋮
);
```

```
CREATE TABLE pages (
  pageID INT PRIMARY KEY,
  title VARCHAR UNIQUE,
  latest INT
  REFERENCES revisions (revID),
);
```

```
CREATE TABLE revisions (
  revID INT PRIMARY KEY,
  userID INT REFERENCES useracct (userID),
  pageID INT REFERENCES pages (pageID),
  content TEXT,
  updated DATETIME
);
```

OLTP

On-line Transaction Processing:

→ Simple queries that read/update a small amount of data that is related to a single entity in the database.

This is usually the kind of application that people build first.

```
SELECT P.*, R.*
  FROM pages AS P
 INNER JOIN revisions AS R
    ON P.latest = R.revID
 WHERE P.pageID = ?
```

```
UPDATE useracct
   SET lastLogin = NOW(),
       hostname = ?
 WHERE userID = ?
```

```
INSERT INTO revisions
VALUES (?,?…,?)
```

OLAP

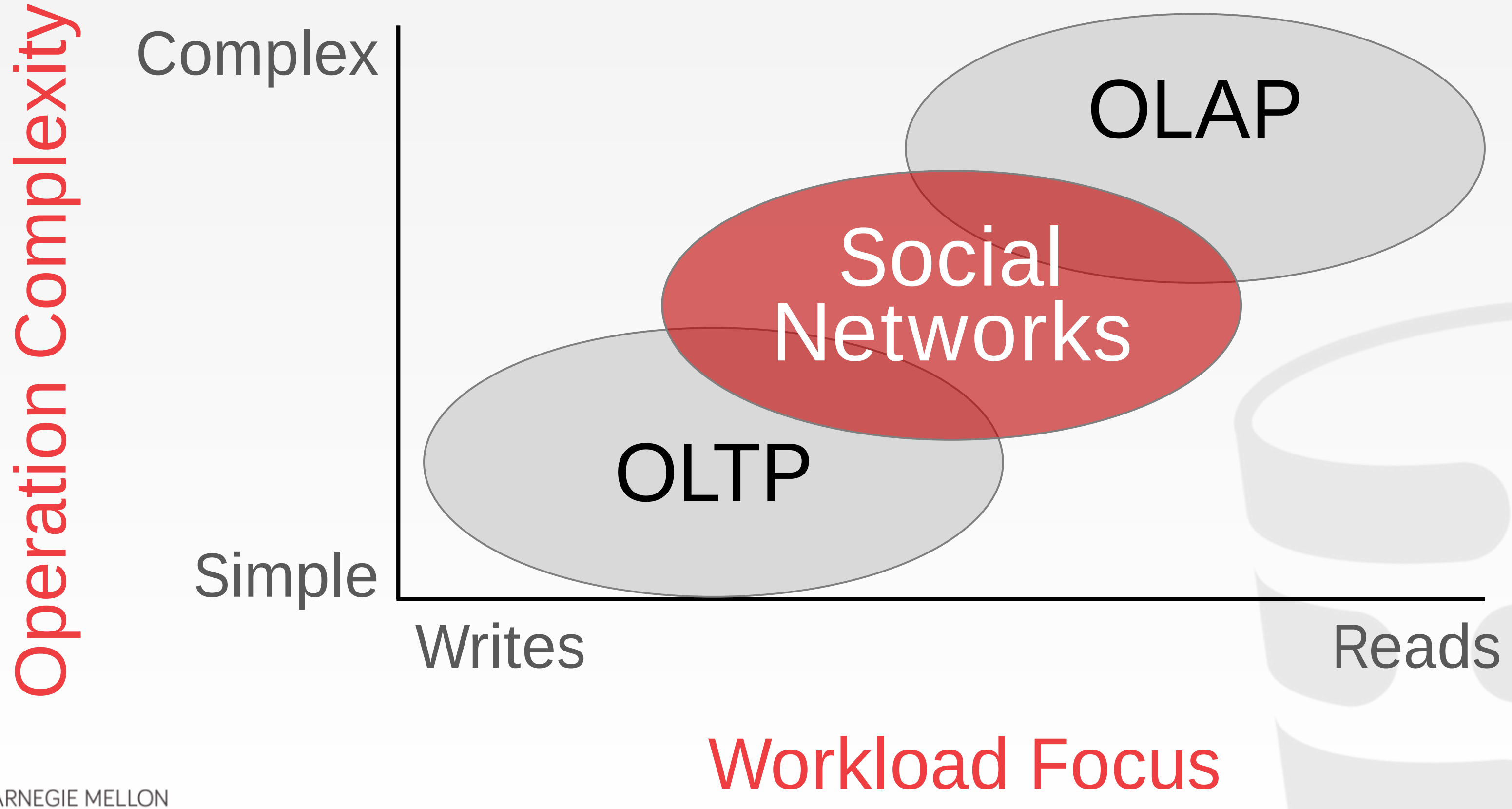
On-line Analytical Processing:

→ Complex queries that read large portions of the database spanning multiple entities.

You execute these workloads on the data you have collected from your OLTP application(s).

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM
            U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY
  EXTRACT(month FROM U.lastLogin)
```

WORKLOAD CHARACTERIZATION



[SOURCE]

DATA STORAGE MODELS

The DBMS can store tuples in different ways that are better for either OLTP or OLAP workloads.

We have been assuming the **n-ary storage model** (aka "row storage") so far this semester.

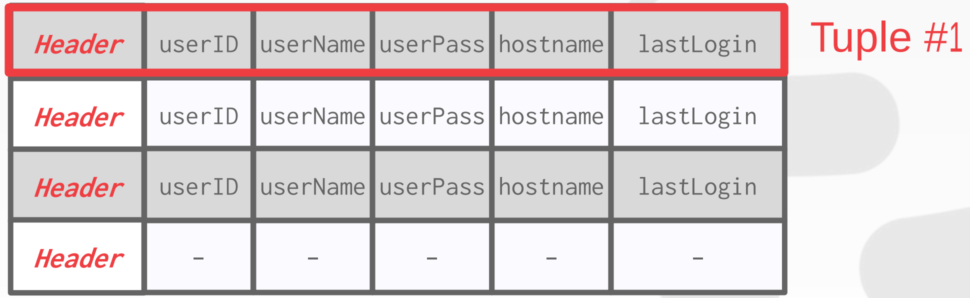
N-ARY STORAGE MODEL (NSM)

The DBMS stores all attributes for a single tuple contiguously in a page.

Ideal for OLTP workloads where queries tend to operate only on an individual entity and insert-heavy workloads.

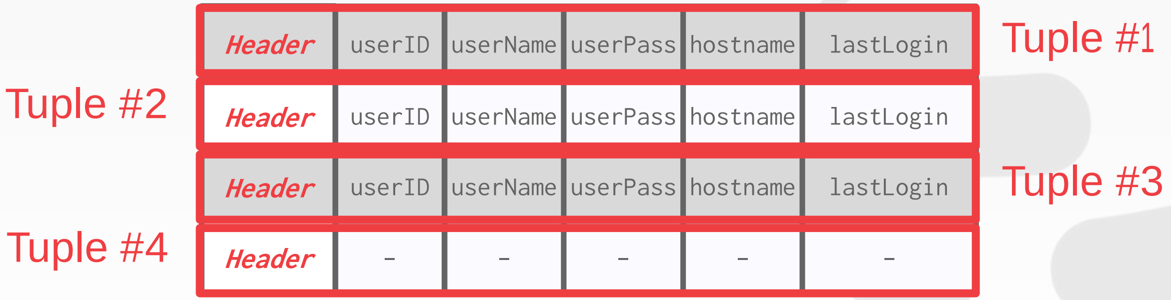
N-ARY STORAGE MODEL (NSM)

The DBMS stores all attributes for a single tuple contiguously in a page.



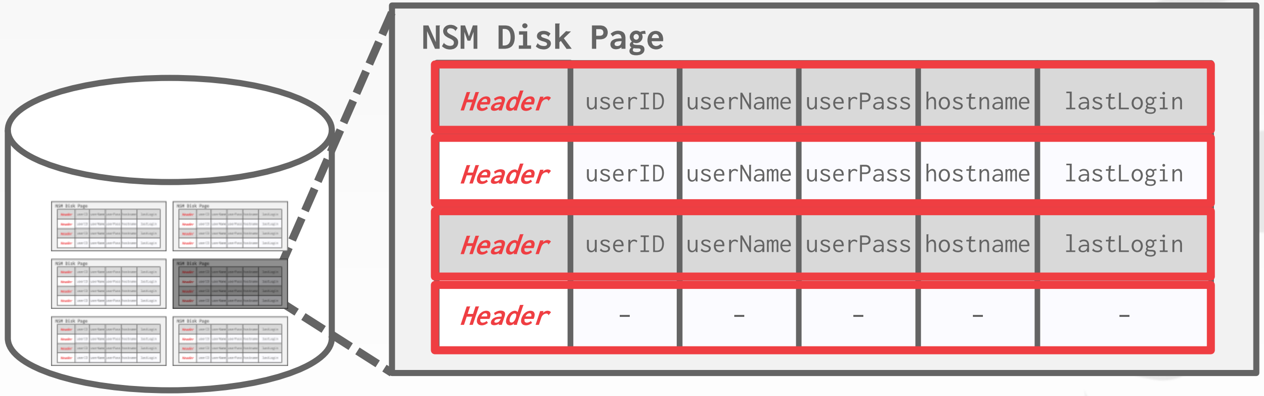
N-ARY STORAGE MODEL (NSM)

The DBMS stores all attributes for a single tuple contiguously in a page.


Tuple #1
Header userID userName userPass hostname lastLogin
Tuple #2
Header userID userName userPass hostname lastLogin
Tuple #3
Header userID userName userPass hostname lastLogin
Tuple #4
Header - - - - -

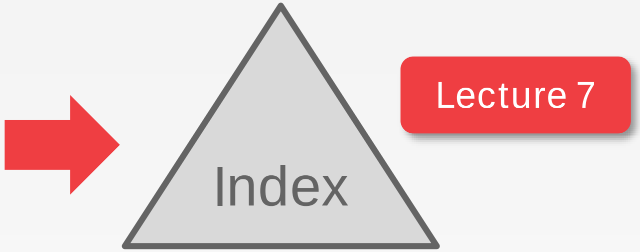
N-ARY STORAGE MODEL (NSM)

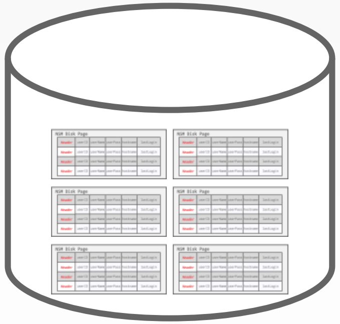
The DBMS stores all attributes for a single tuple contiguously in a page.



N-ARY STORAGE MODEL (NSM)

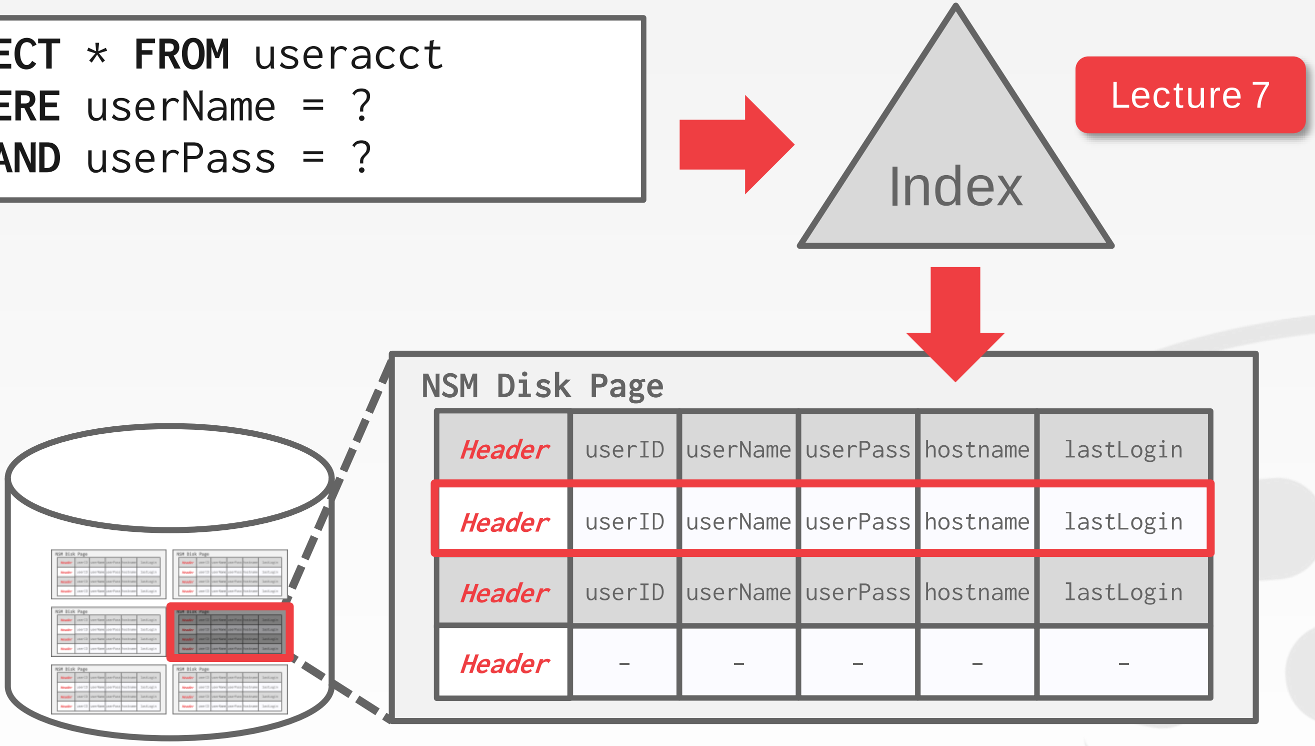
```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```





N-ARY STORAGE MODEL (NSM)

```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```

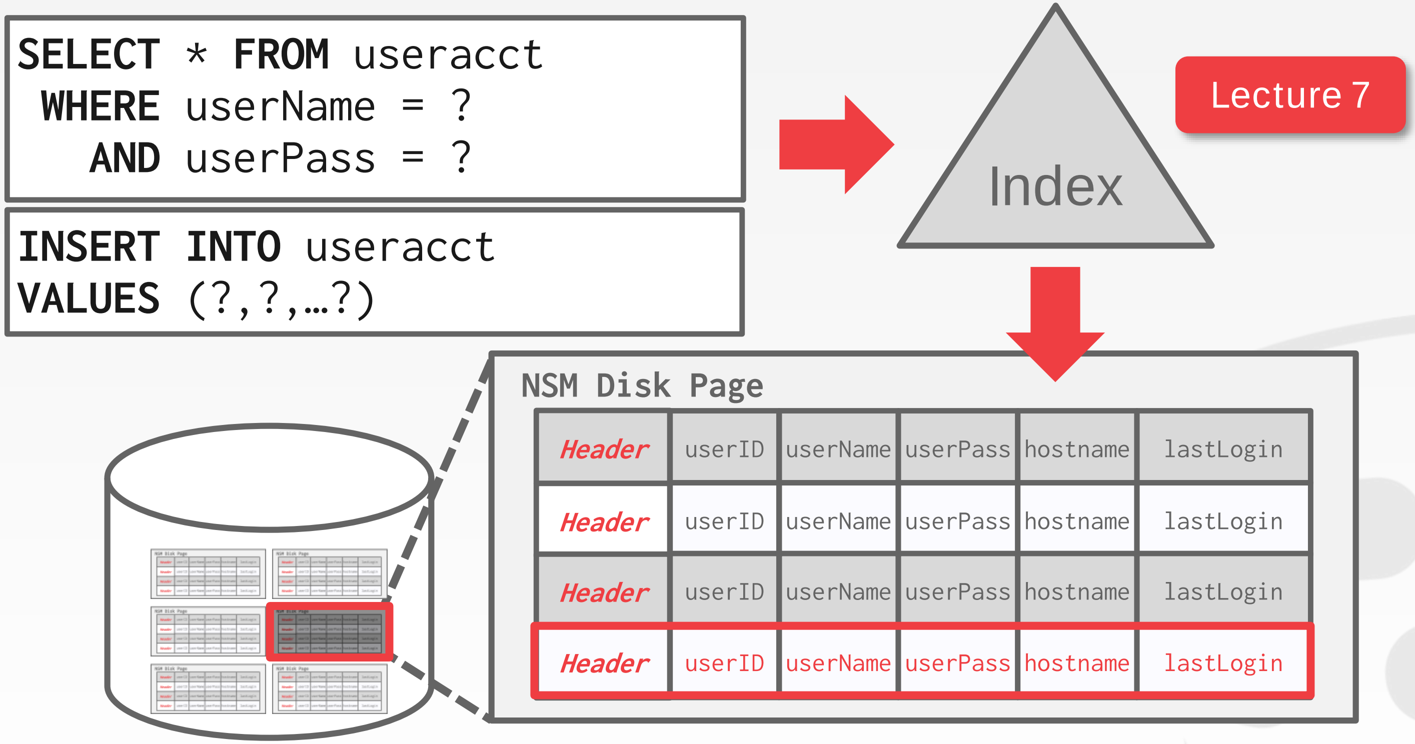


Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	-	-	-	-	-

N-ARY STORAGE MODEL (NSM)

```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```

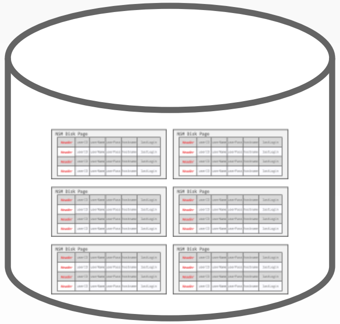
```
INSERT INTO useracct
VALUES (?,?,…?)
```



Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

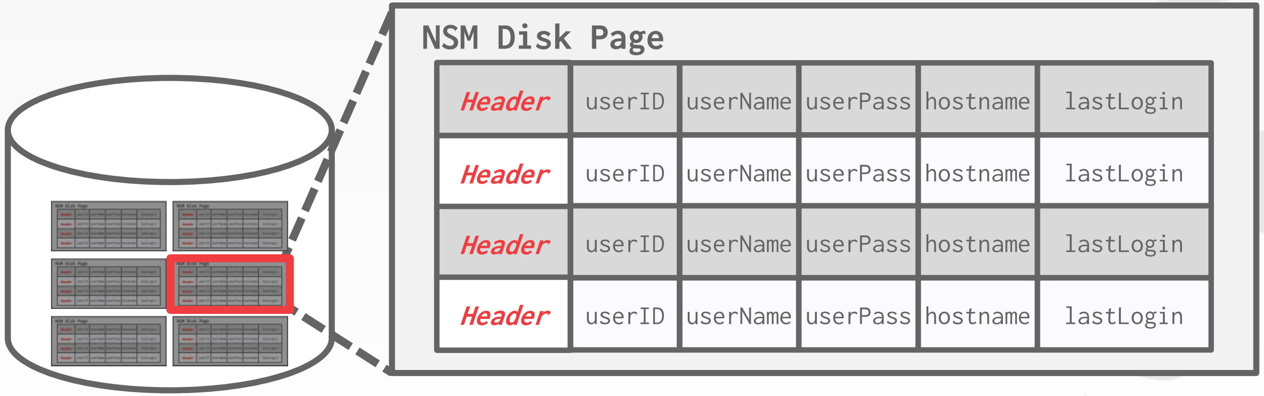
N-ARY STORAGE MODEL (NSM)

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



N-ARY STORAGE MODEL (NSM)

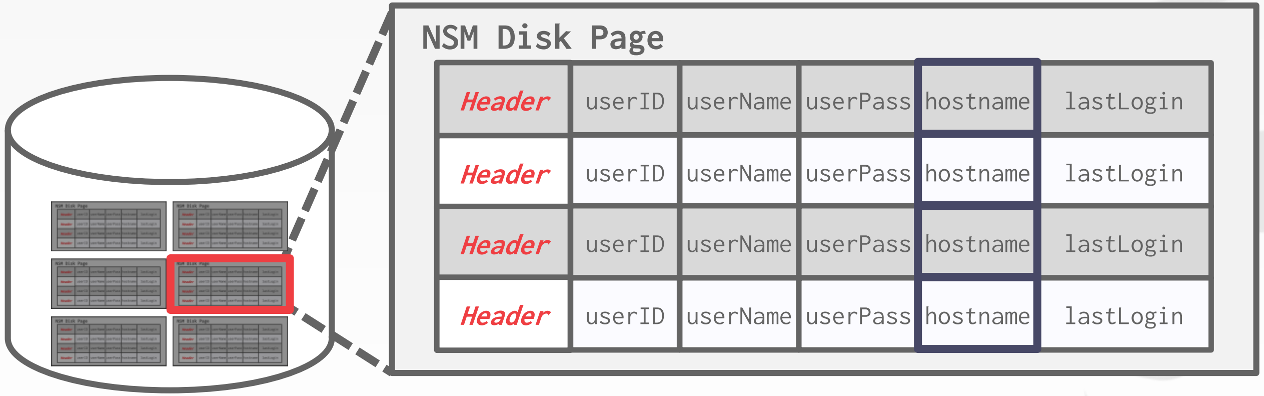
```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

N-ARY STORAGE MODEL (NSM)

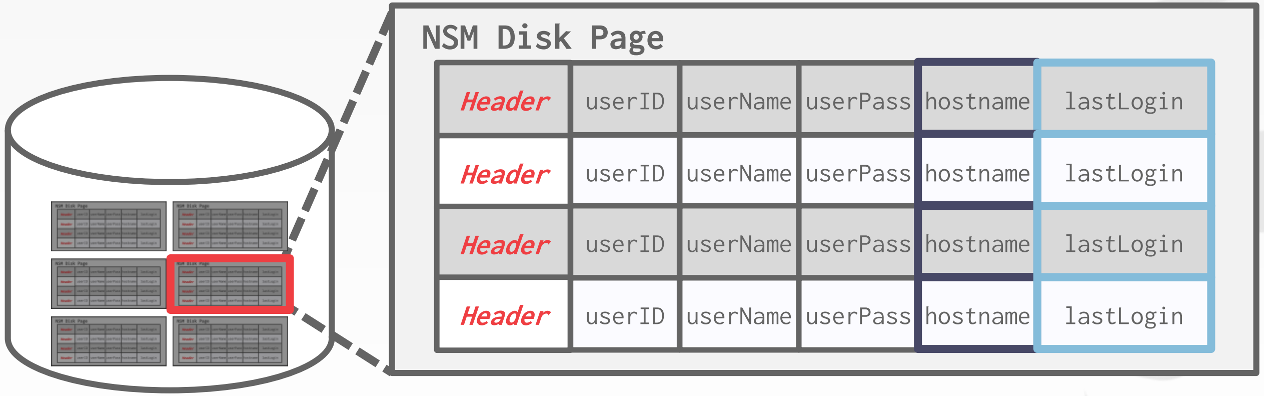
```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

N-ARY STORAGE MODEL (NSM)

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```

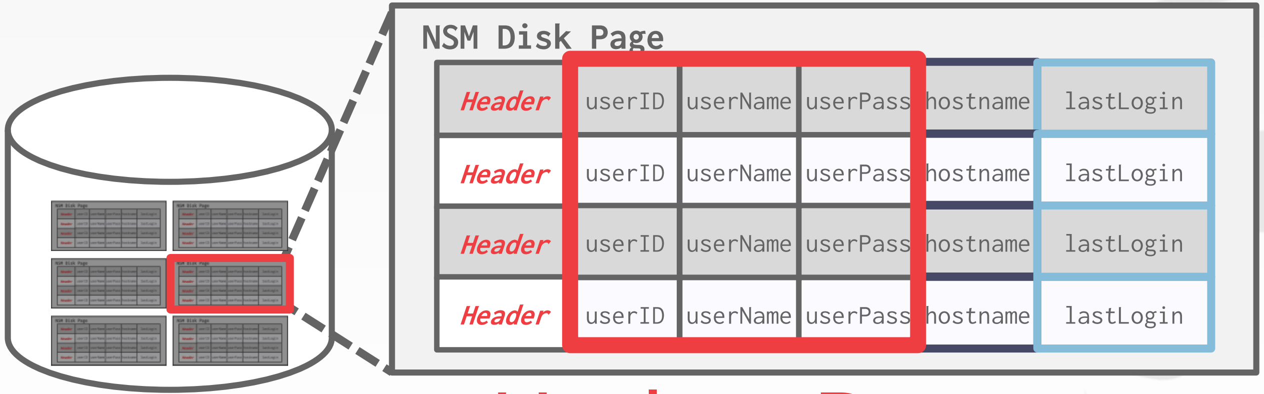


Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

CARNEGIE MELLON DATABASE GROUP

N-ARY STORAGE MODEL (NSM)

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



Useless Data

N-ARY STORAGE MODEL

Advantages

→ Fast inserts, updates, and deletes.
→ Good for queries that need the entire tuple.

Disadvantages

→ Not good for scanning large portions of the table and/or a subset of the attributes.

DECOMPOSITION STORAGE MODEL (DSM)

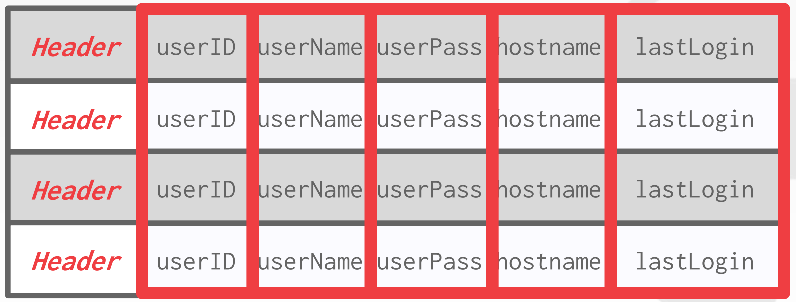
The DBMS stores the values of a single attribute for all tuples contiguously in a page.
→ Also known as a "column store".

Ideal for OLAP workloads where read-only queries perform large scans over a subset of the table's attributes.

DECOMPOSITION STORAGE MODEL (DSM)

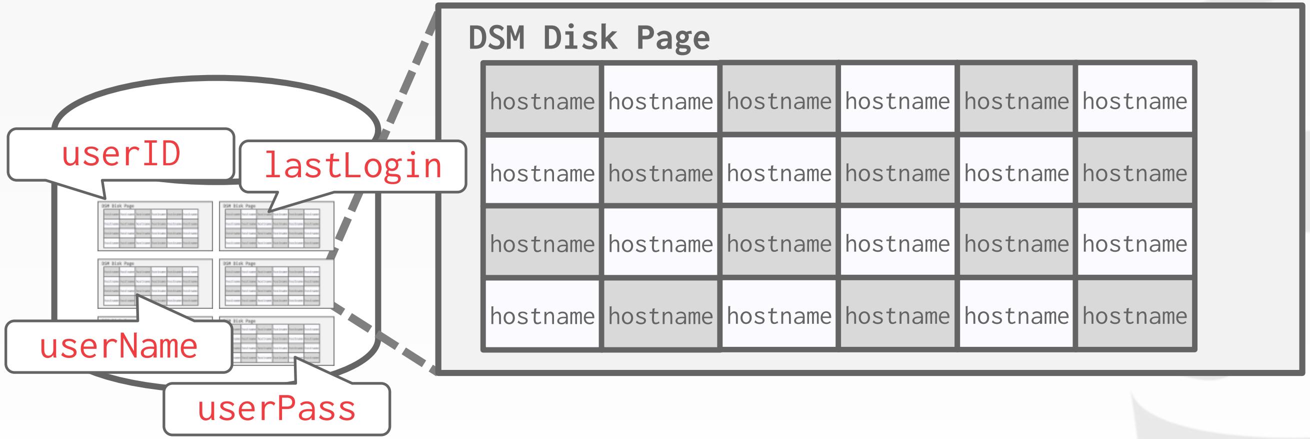
The DBMS stores the values of a single attribute for all tuples contiguously in a page.

→ Also known as a "column store".



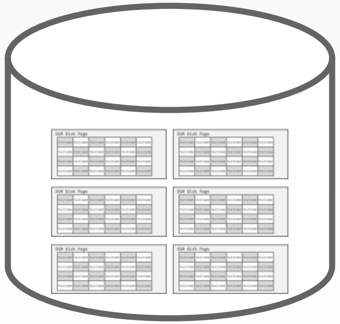
DECOMPOSITION STORAGE MODEL (DSM)

The DBMS stores the values of a single attribute for all tuples contiguously in a page.
→ Also known as a "column store".



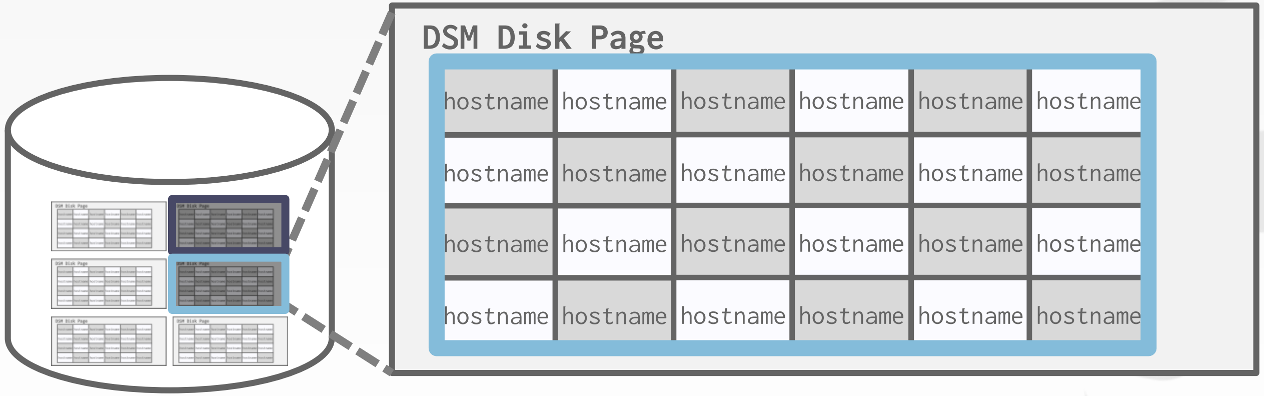
DECOMPOSITION STORAGE MODEL (DSM)

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



DECOMPOSITION STORAGE MODEL (DSM)

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```

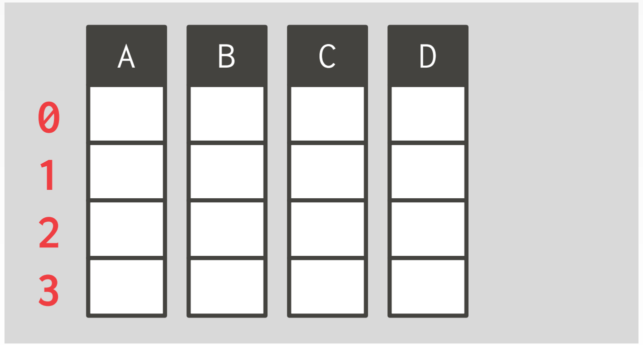


TUPLE IDENTIFICATION

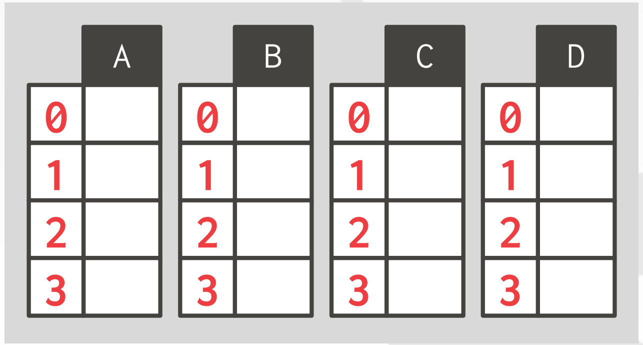
Choice #1: Fixed-length Offsets
→ Each value is the same length for an attribute.

Choice #2: Embedded Tuple Ids
→ Each value is stored with its tuple id in a column.

Offsets



Embedded Ids



DECOMPOSITION STORAGE MODEL (DSM)

Advantages

→ Reduces the amount wasted I/O because the DBMS only reads the data that it needs.

→ Better query processing and data compression (more on this later).

Disadvantages

→ Slow for point queries, inserts, updates, and deletes because of tuple splitting/stitching.

DSM SYSTEM HISTORY

1970s: Cantor DBMS
1980s: DSM Proposal
1990s: SybaseIQ (in-memory only)
2000s: Vertica, VectorWise, MonetDB
2010s: Everyone

CONCLUSION

The storage manager is not entirely independent from the rest of the DBMS.

It is important to choose the right storage model for the target workload:
→ OLTP = Row Store
→ OLAP = Column Store

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**