Intro to Databases (COMP_SCI 339)

03 Disk Storage

Northwestern University

WINTER 2024

Andrew Crotty

TODAY'S AGENDA

Course Logistics
DBMS Overview
File Storage
Page/Tuple Layout
Storage Models

COURSE LOGISTICS

Lectures: Mon/Wed @ 3:30-4:50pm

Office Hours: See Canvas

Schedule: Canvas
Q&A: Piazza
Projects: Gradescope

GRADING RUBRIC

Projects – 40% (4 x 10% each)

Exams – 60% (3 x 20% each)

Homeworks – ungraded (prep for exams)

LATE POLICY

You will lose 33% of the points on a project for every 24 hours it is late.

You have a total of <u>**4**</u> no-penalty late days that may be used to turn in projects up to 24 hours late.

We will grant additional extensions due to extreme circumstances (e.g., medical emergencies).
→ If something comes up, please contact me as soon as possible.

PLAGIARISM WARNING

The projects must be your own original work.
They are **not** group assignments.

You may **not** copy source code from other students or the web.

Plagiarism is **not** tolerated. You will get lit up.
→ Please ask me if you are unsure.

COURSE OVERVIEW

This course is about the design and implementation of database management systems (DBMSs).

This is **not** a course about how to use a DBMS to build applications or how to administer a DBMS.

DATABASE MANAGEMENT SYSTEM

A database management system (**DBMS**) is software that allows applications to store and analyze information in a database.

A general-purpose DBMS supports the definition, creation, querying, update, and administration of databases in accordance with some data model.

FIRST TWO LECTURES

You should 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 the software that manages a database (i.e., a DBMS).

COURSE OUTLINE

Introduction & SQL
Storage & Indexing
Query Execution
Query Optimization
Concurrency Control
Logging & Recovery

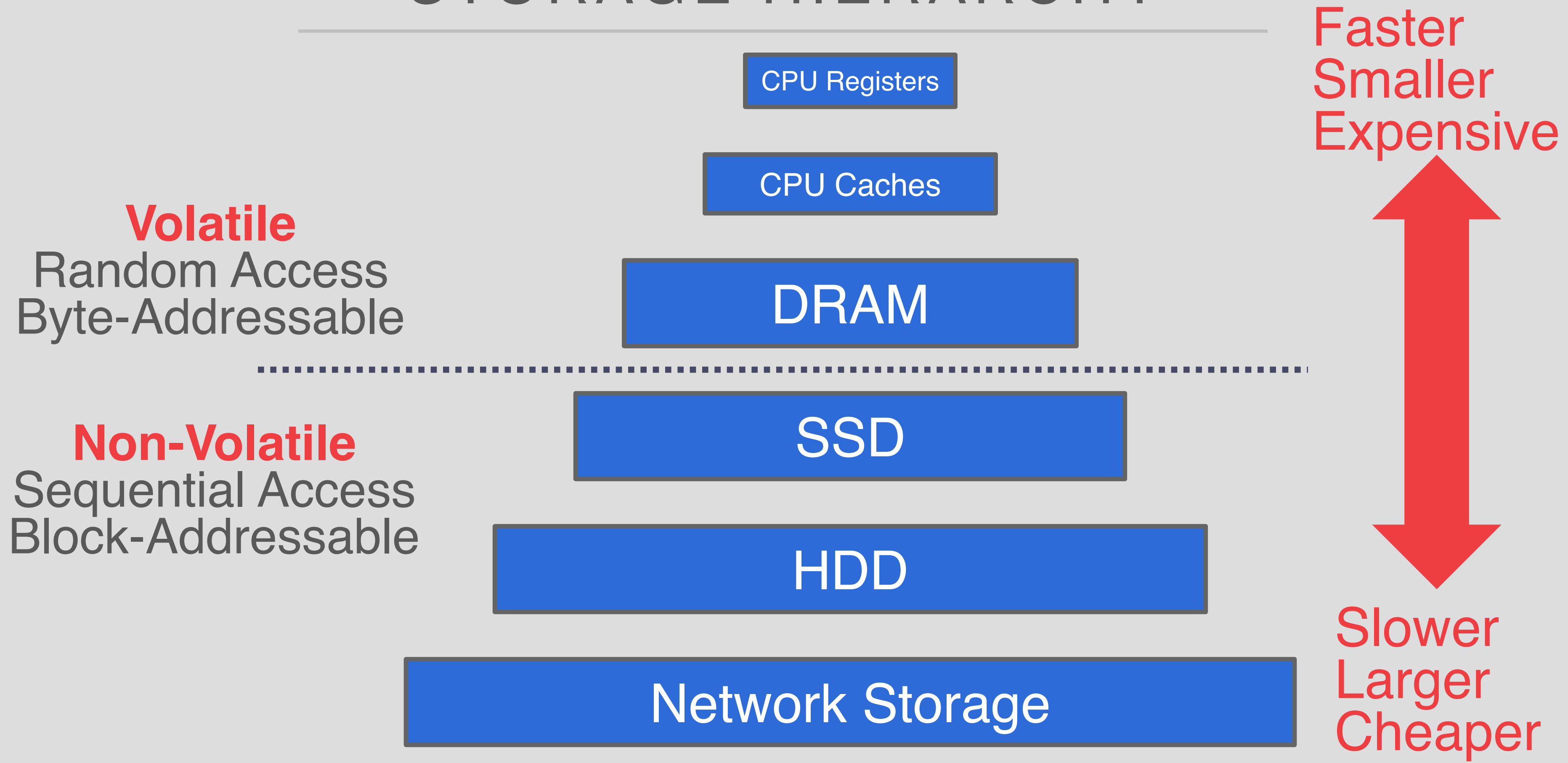
Query Planning
Operator Execution
Access Methods
Buffer Pool Manager
Disk Manager

DISK-BASED ARCHITECTURE

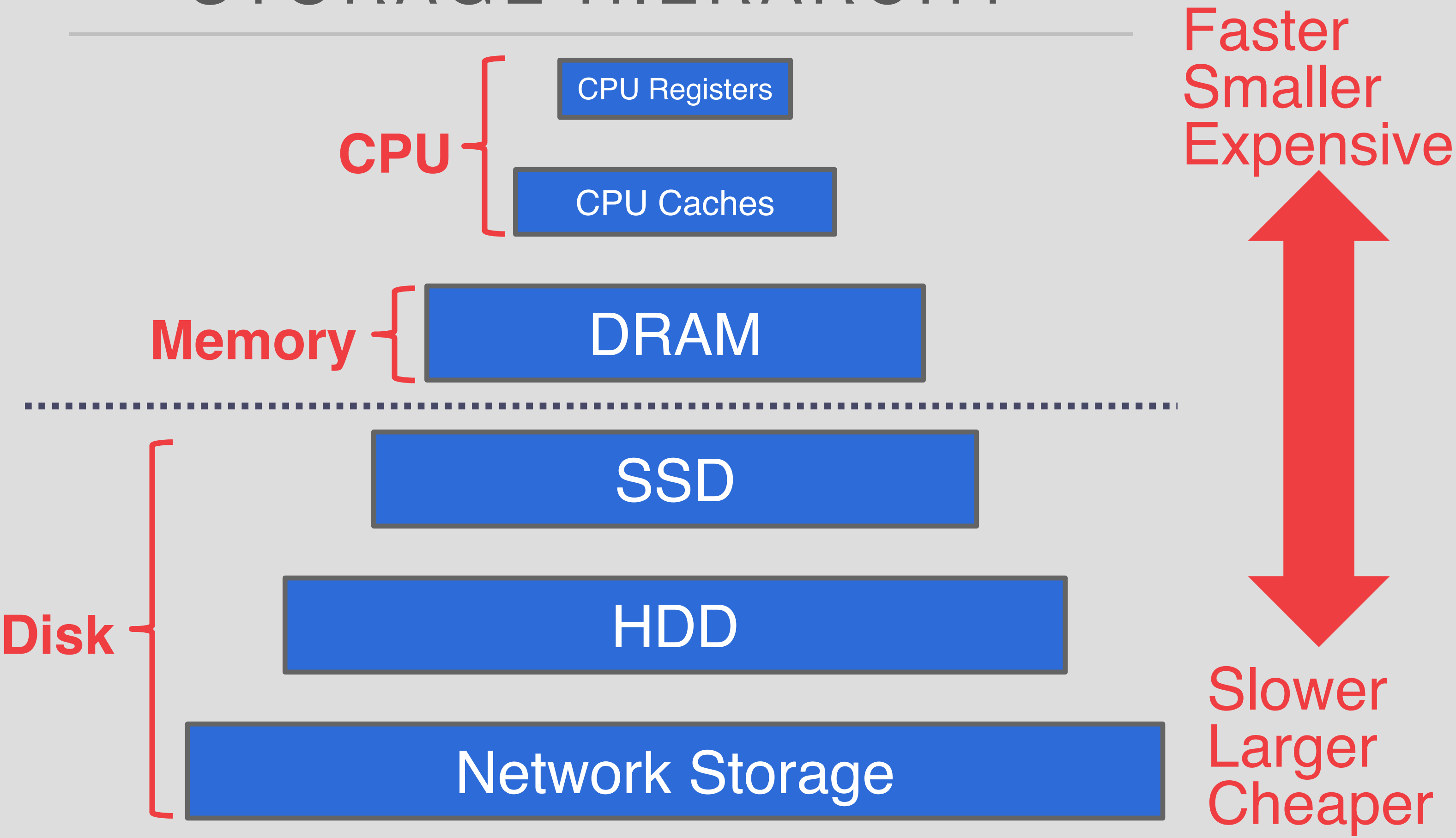
The DBMS assumes that the primary storage location of the database is on non-volatile storage (e.g., HDD, SSD).

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

STORAGE HIERARCHY



STORAGE HIERARCHY



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.

Therefore, the 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 <u>extent</u>.

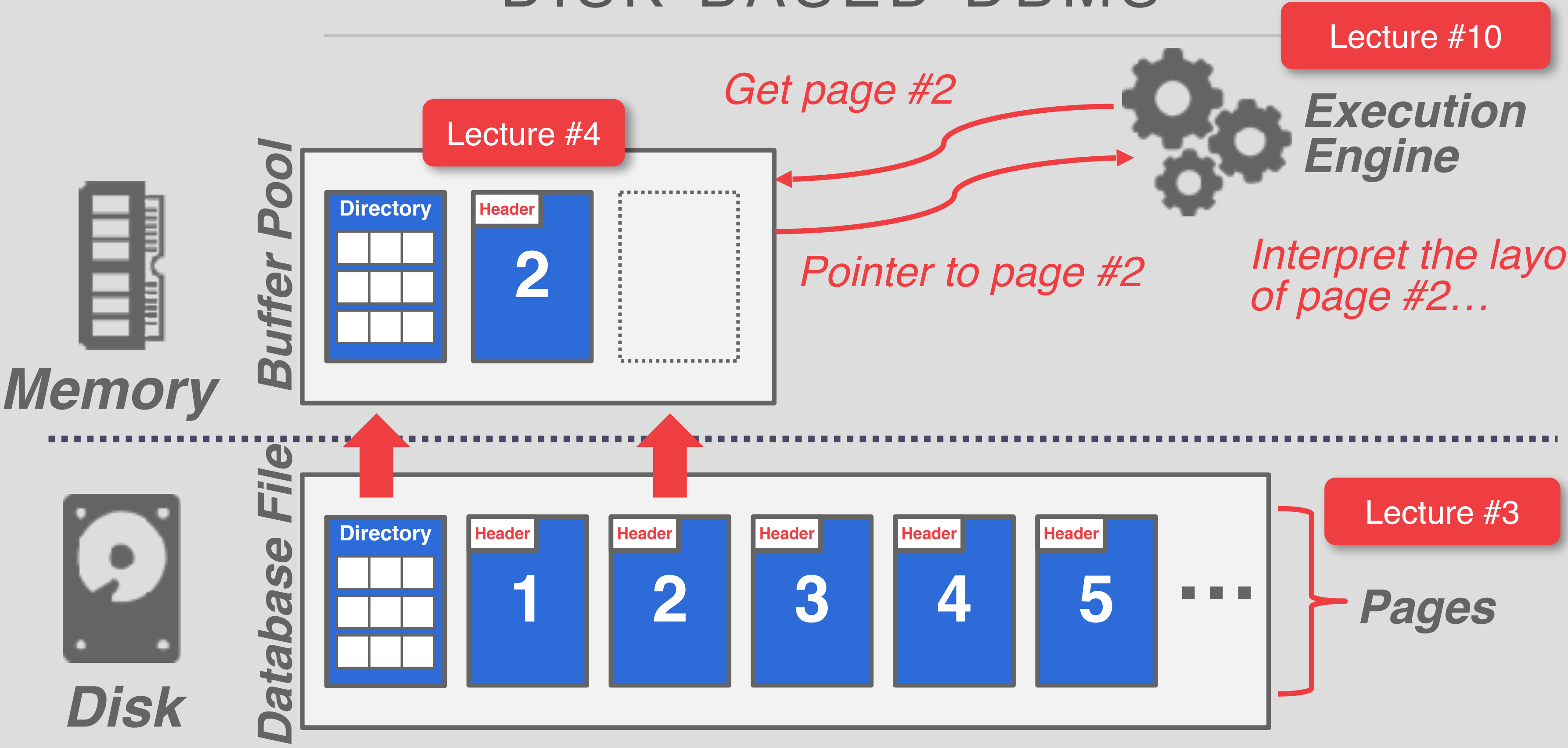
DESIGN GOALS

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.

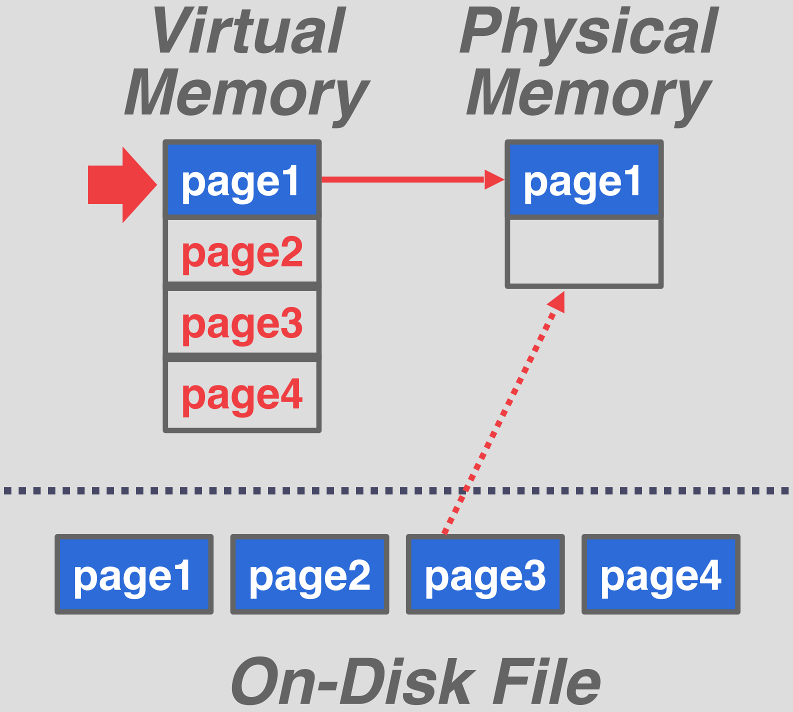
DISK-BASED DBMS



WHY NOT USE THE OS?

The DBMS can use memory mapping (**mmap**) to store the contents of a file into the address space of a program.

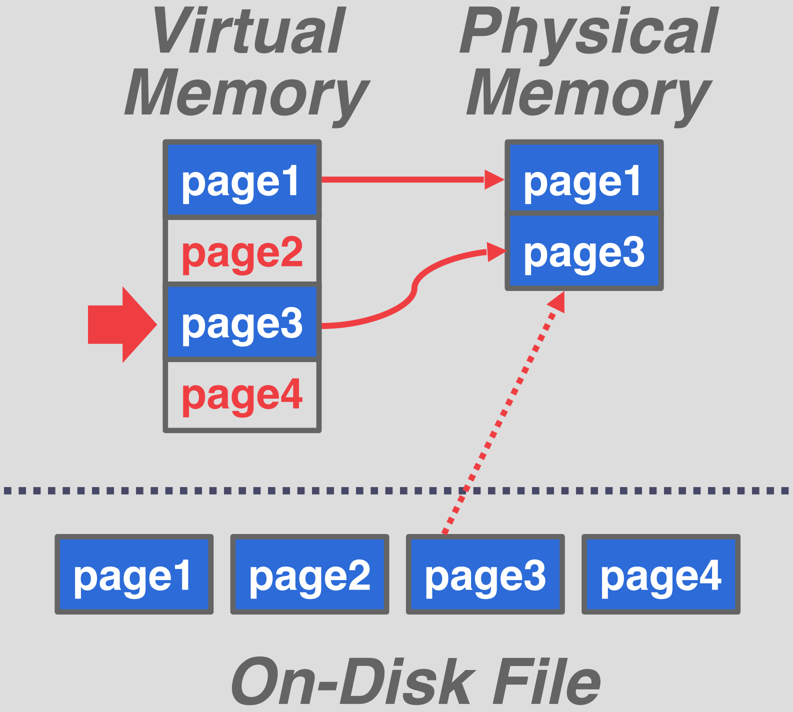
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



WHY NOT USE THE OS?

The DBMS can use memory mapping (**mmap**) to store the contents of a file into the address space of a program.

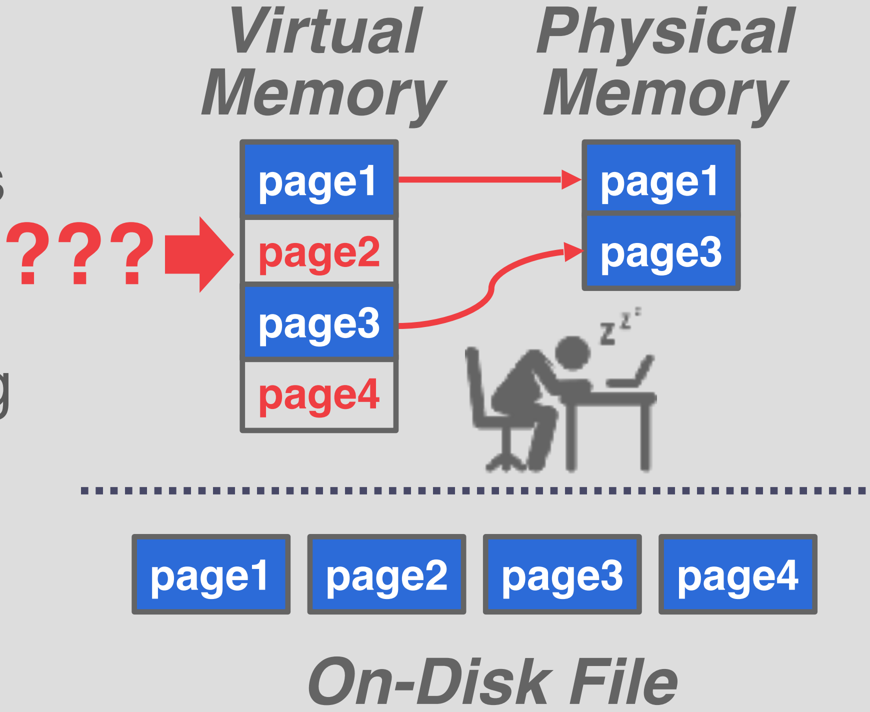
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



WHY NOT USE THE OS?

The DBMS can use memory mapping (**mmap**) to store the contents of a file into the address space of a program.

The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



WHY NOT USE THE OS?

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

This works well enough for read-only access.

It is complicated when there are multiple writers…

MEMORY MAPPED I/O PROBLEMS

Problem #1: Transaction Safety
→ OS can flush dirty pages at any time.

Problem #2: I/O Stalls
→ DBMS doesn't know which pages are in memory. The OS will stall a thread on page fault.

Problem #3: Error Handling
→ Difficult to validate pages. Any access can cause a **SIGBUS** that the DBMS must handle.

Problem #4: Performance Issues
→ OS data structure contention. TLB shootdowns.

WHY NOT USE THE OS?

DBMS (almost) always wants to control things itself and can do a better job than OS.

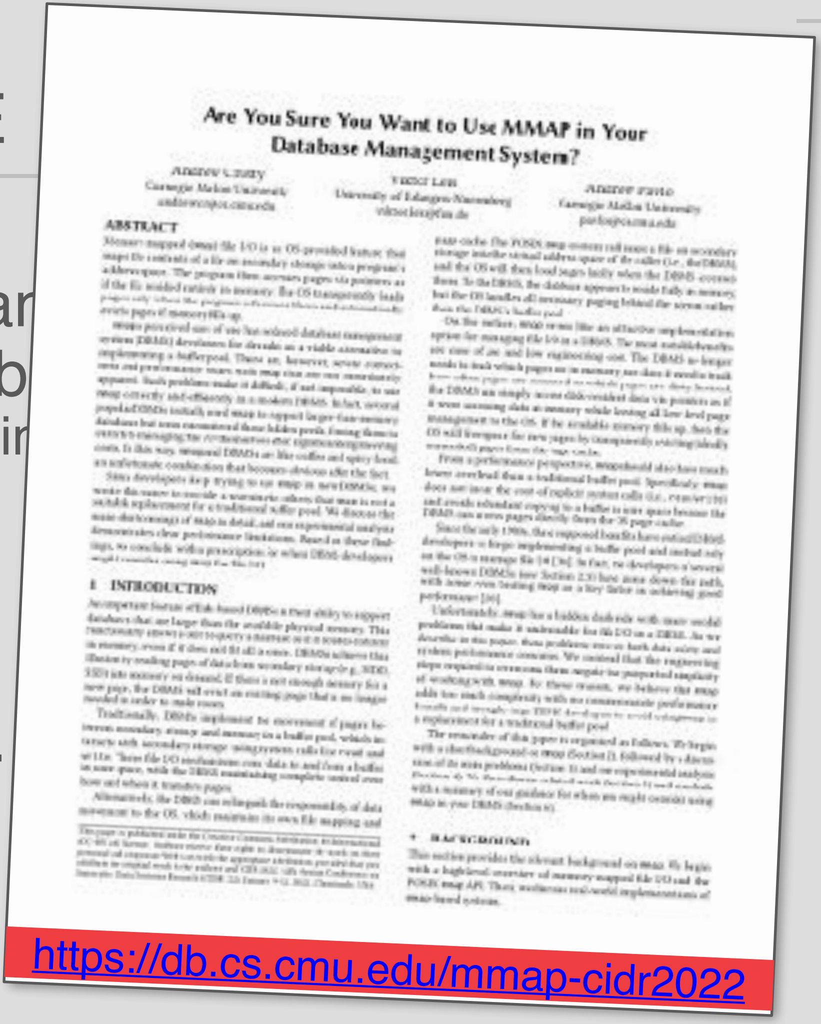
→ Flushing dirty pages to disk in the correct order.
→ Specialized prefetching.
→ Buffer replacement policy.
→ Thread/process scheduling.

The OS is **<u>not</u>** your friend.

WHY NOT USE

DBMS (almost) always war
things itself and can do a b
→ Flushing dirty pages to disk ir
→ Specialized prefetching.
→ Buffer replacement policy.
→ Thread/process scheduling.

The OS is **not** your friend.


Are You Sure You Want to Use MMAP in Your Database Management System?

ABSTRACT

1 INTRODUCTION

2 BACKGROUND

https://db.cs.cmu.edu/mmap-cidr2022

DATABASE STORAGE

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

Problem #2: How the DBMS manages memory and transfers data to/from disk.

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 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.

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 (usually 4KB)
→ Database Page (512B-16KB)

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

4KB
SQLite
IBM DB2
ORACLE

8KB
Microsoft SQL Server
PostgreSQL

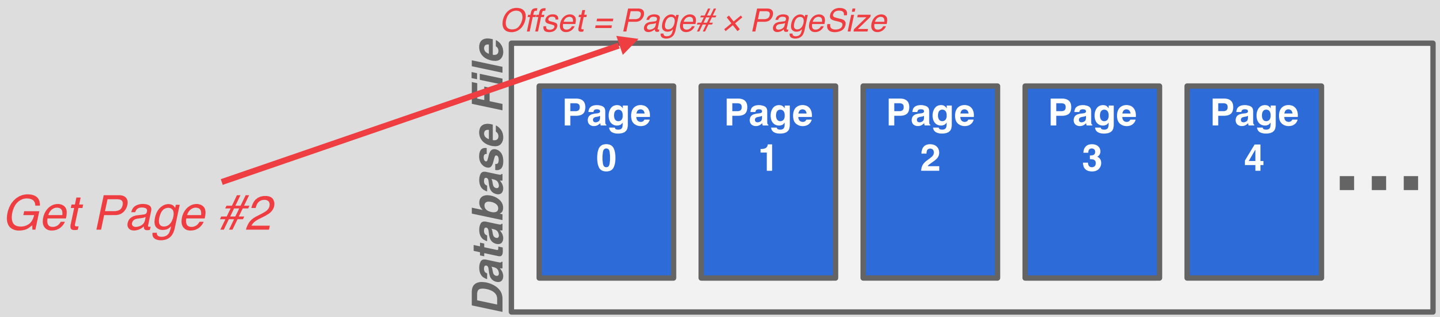
16KB
MySQL

PAGE STORAGE ARCHITECTURE

Different DBMSs manage pages in files on disk in different ways.
→ Heap File Organization
→ Tree File Organization
→ Sequential / Sorted File Organization (ISAM)
→ Hashing File Organization

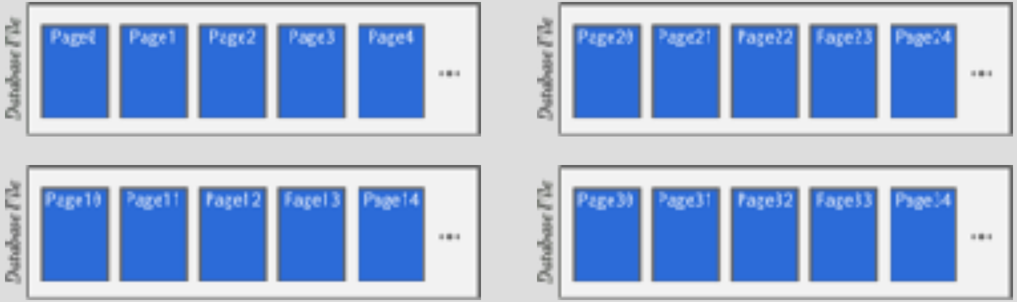
At this point in the hierarchy we don't need to know anything about what is inside of the pages.

HEAP FILE


Offset = Page# × PageSize
Get Page #2
Database File
Page 0
Page 1
Page 2
Page 3
Page 4
...

HEAP FILE

Get Page #2



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.

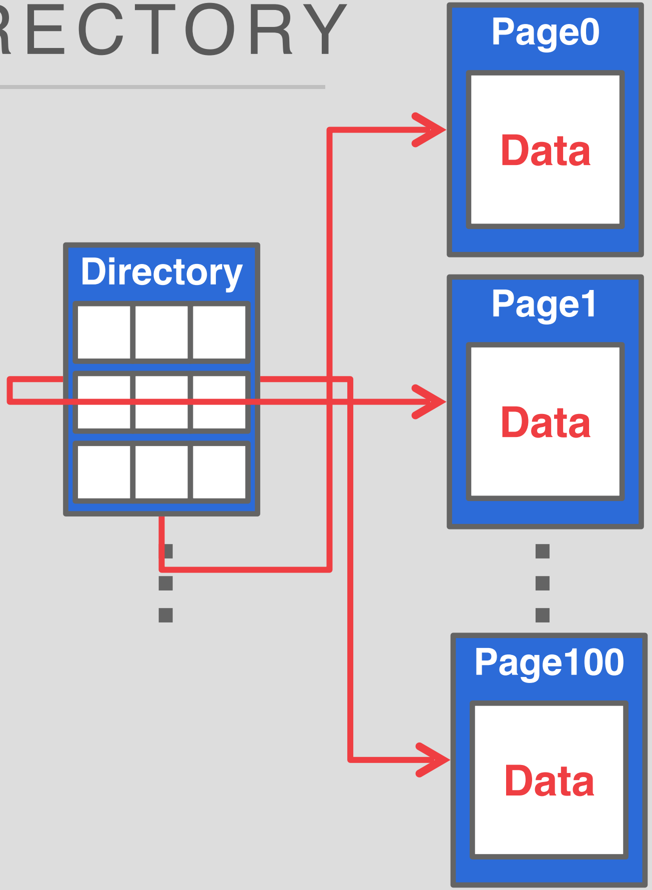
It is easy to find pages if there is only a single file.

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

HEAP FILE: PAGE DIRECTORY

The DBMS maintains special pages that tracks the location of data pages in the database files.
→ Must make sure that the directory pages are in sync with the data pages.

The directory also records meta-data about available space:
→ The number of free slots per page.
→ List of free / empty pages.

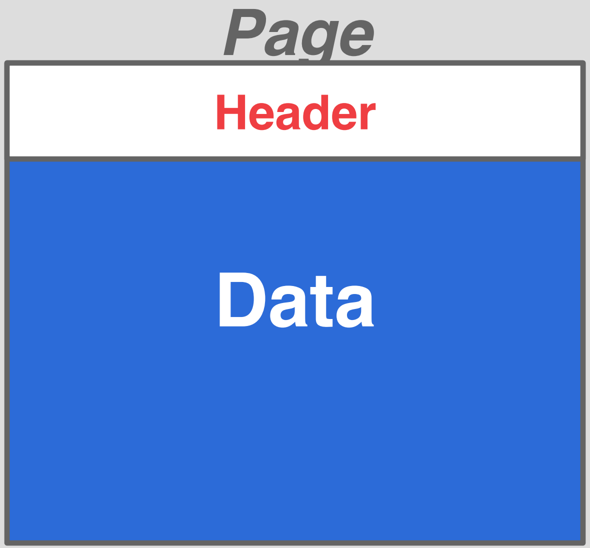


PAGE HEADER

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 STORAGE ARCHITECTURE

Insert a new tuple:

→ Check page directory to find a page with a free slot.
→ Retrieve the page from disk (if not in memory).
→ Check slot array to find empty space in page that will fit.

Update an existing tuple using its record id:

→ Check page directory to find location of page.
→ Retrieve the page from disk (if not in memory).
→ Find offset in page using slot array.
→ Overwrite existing data (if new data fits).

PAGE LAYOUT

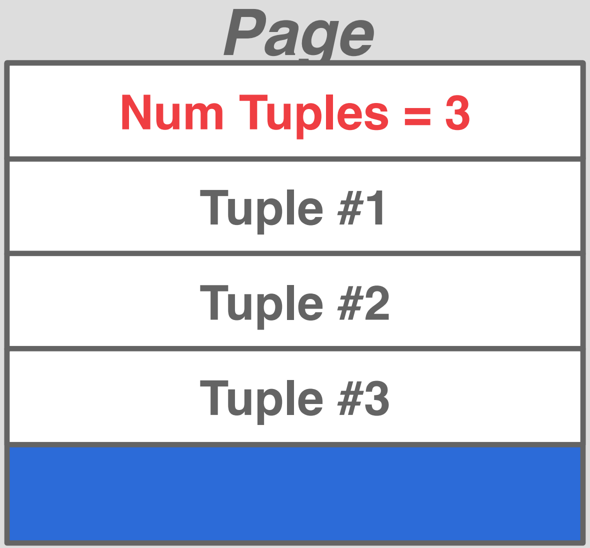
For any page storage architecture, we now need to decide how to organize the data inside of the page.
→ We are still assuming that we are only storing tuples.

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.

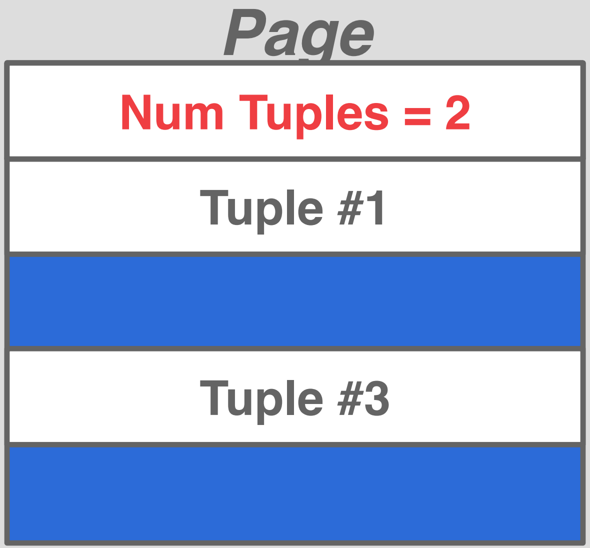


TUPLE STORAGE

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?

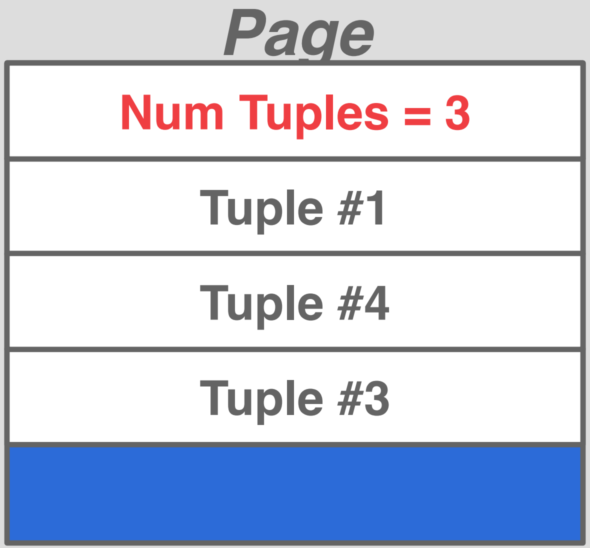


TUPLE STORAGE

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?



RECORD ID

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.

PostgreSQL
CTID (6-bytes)

SQLite
ROWID (8-bytes)

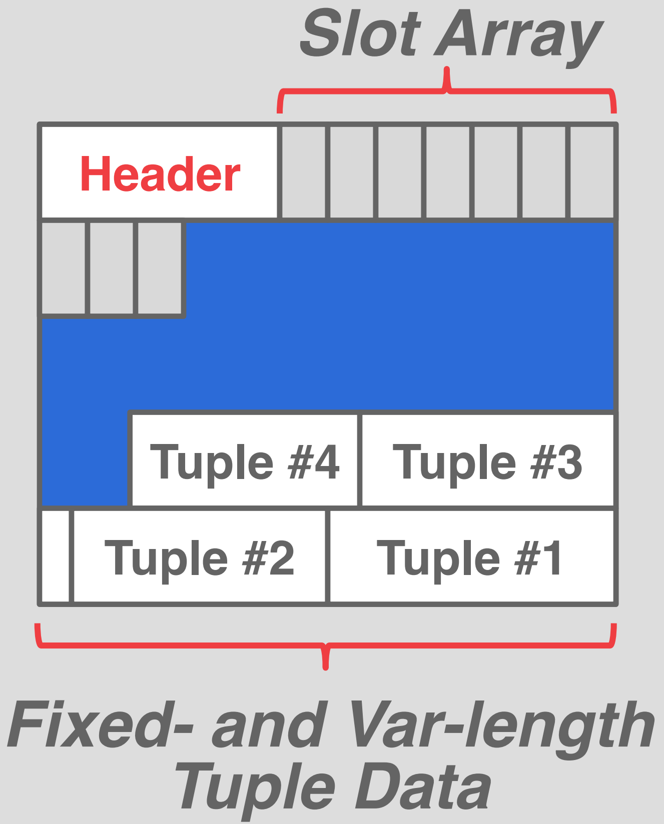
ORACLE
ROWID (10-bytes)

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.

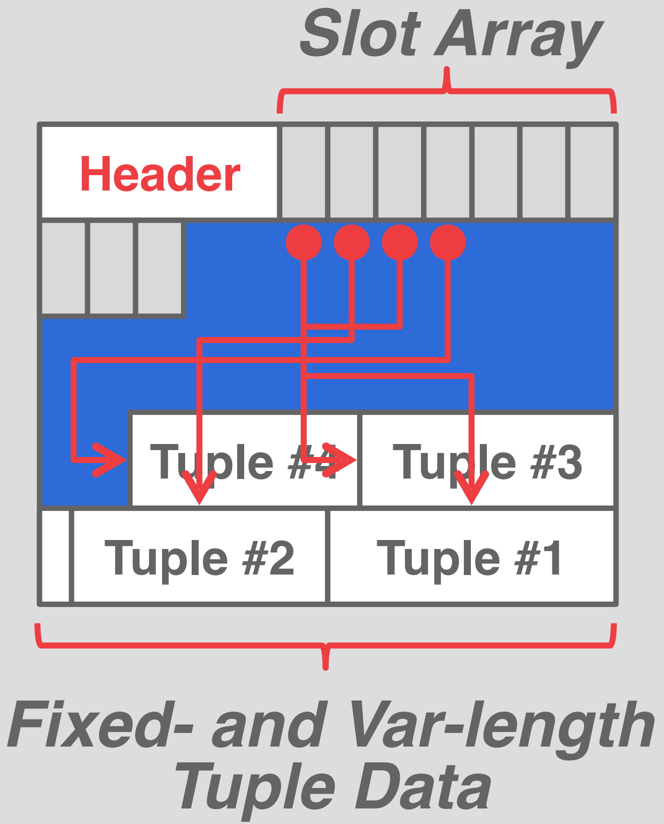


SLOTTED PAGES

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.

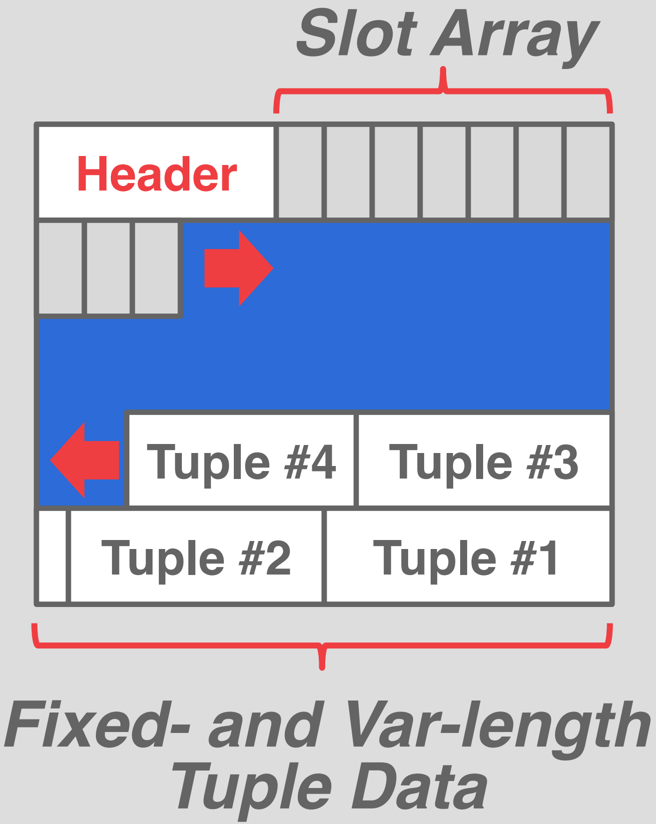


SLOTTED PAGES

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.

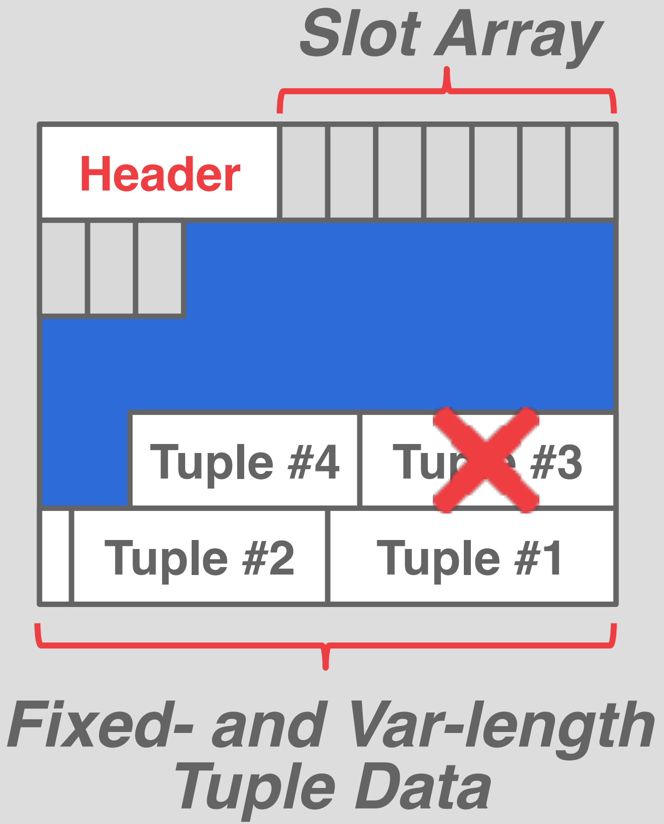


SLOTTED PAGES

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.

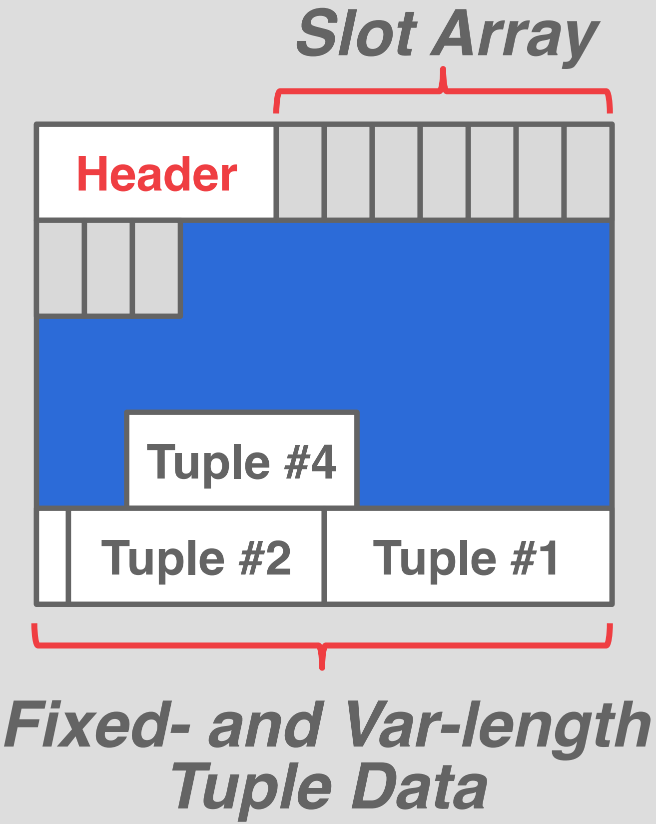


SLOTTED PAGES

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.

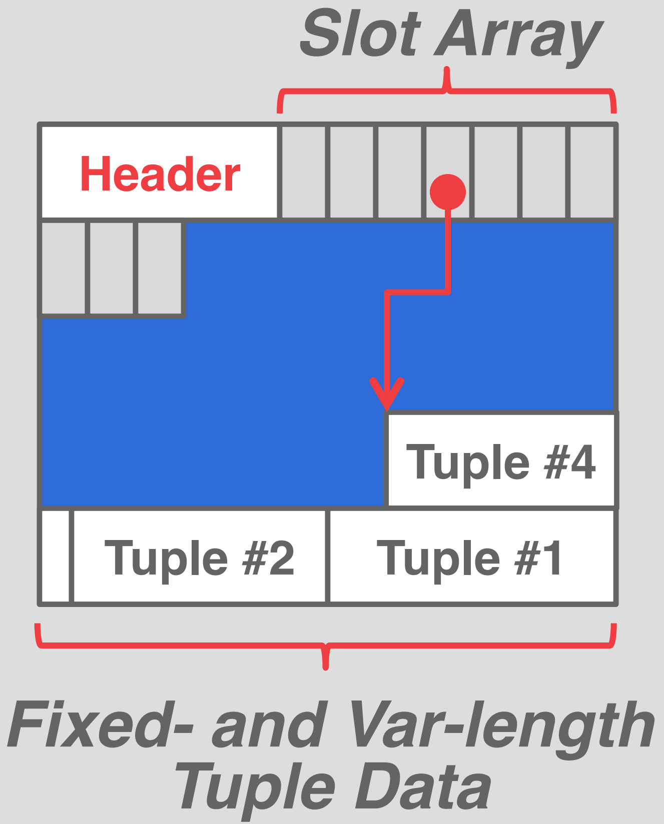


SLOTTED PAGES

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.



TUPLE LAYOUT

A tuple is essentially a sequence of bytes.

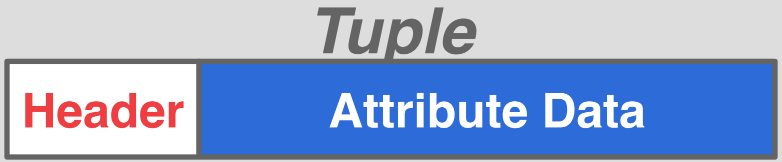
It's the job of the DBMS to interpret those bytes into attribute types and values.

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.



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.
→ Need to worry about collations / sorting.

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

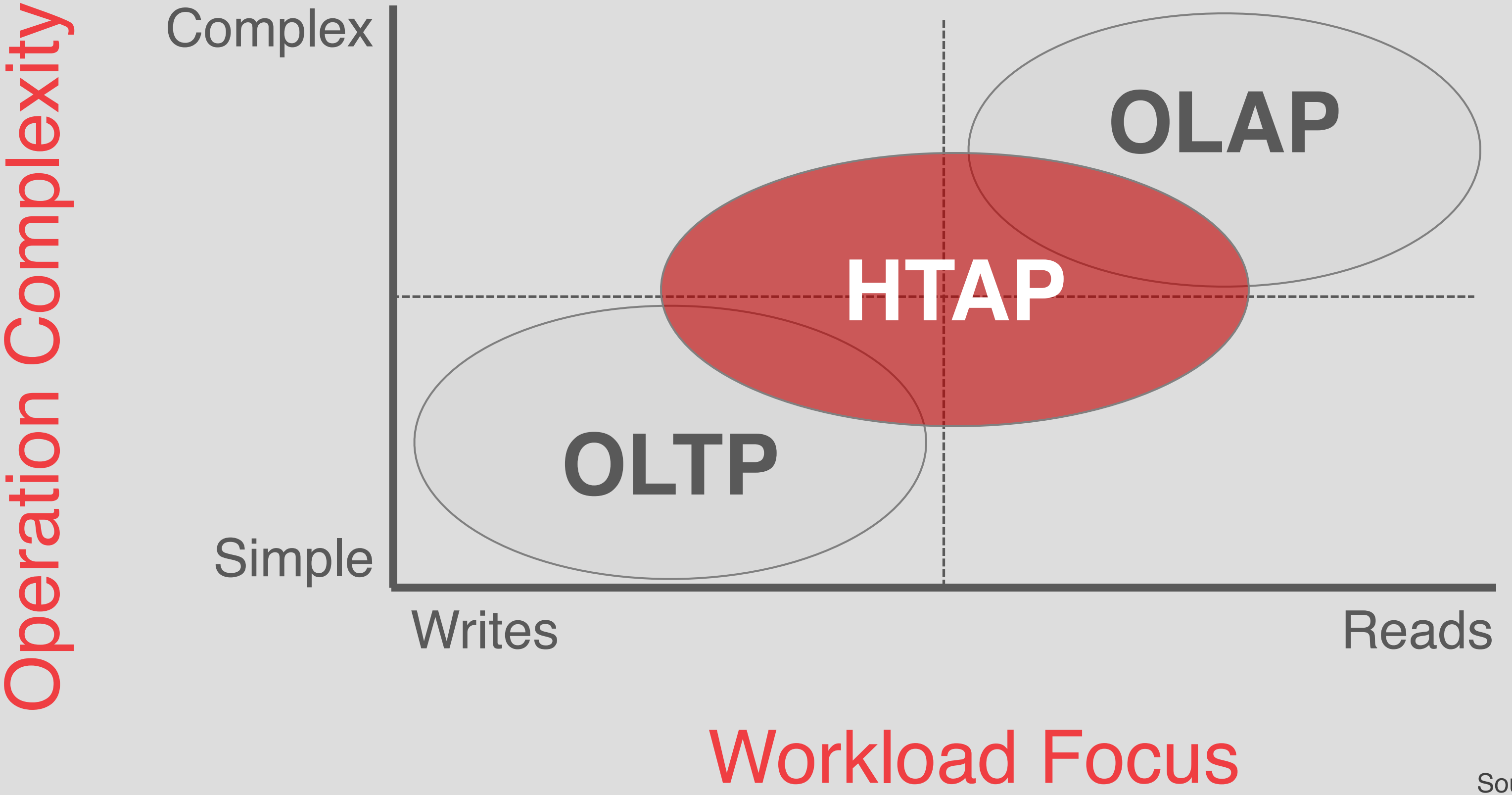
DATABASE WORKLOADS

On-Line Transaction Processing (OLTP)
→ Fast operations that only read/update a small amount of data each time.

On-Line Analytical Processing (OLAP)
→ Complex queries that read a lot of data to compute aggregates.

Hybrid Transaction + Analytical Processing
→ OLTP + OLAP together on the same database.

DATABASE WORKLOADS



Source: [Mike Stonebraker]

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.

```
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)
```

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

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.

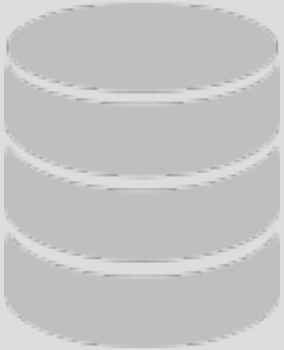
N-ARY STORAGE MODEL (NSM)

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)

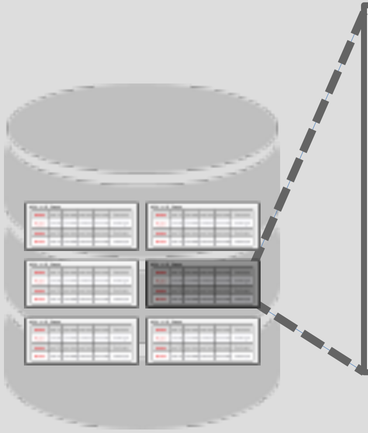
The DBMS stores all attributes for a single tuple contiguously in a page.



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

N-ARY STORAGE MODEL (NSM)

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



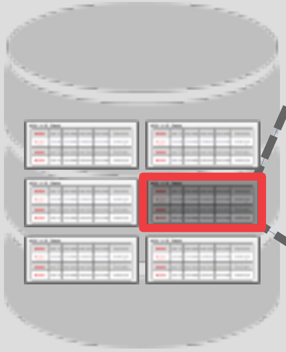
NSM Disk Page

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 = ?
```

Lectures #5-6



NSM Disk Page

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

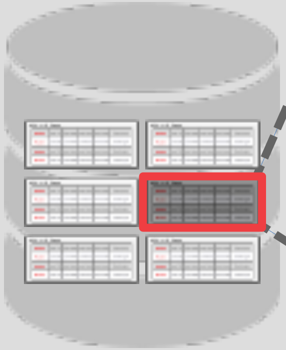
N-ARY STORAGE MODEL (NSM)

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

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

Index

Lectures #5-6

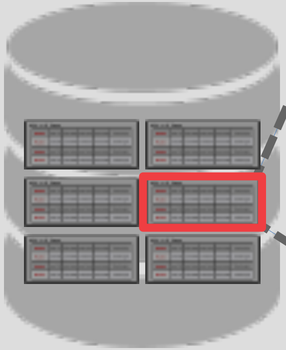


NSM Disk Page

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)

```sql
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)
```

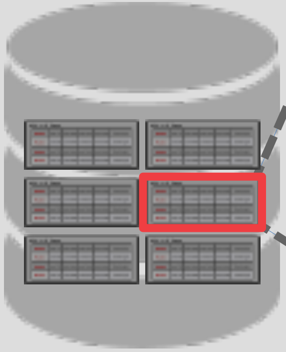


NSM Disk Page

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)
```



NSM Disk Page

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

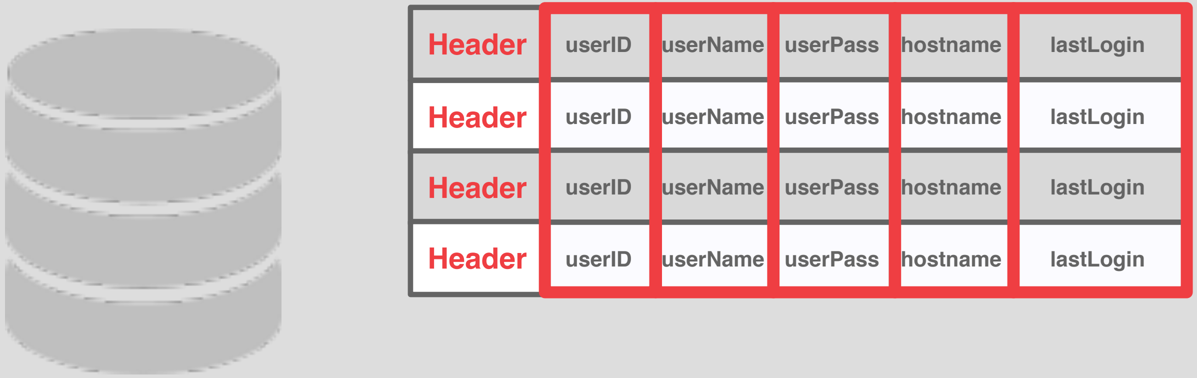
The DBMS stores the values of a single attribute for all tuples contiguously in a page.
→ Also known as a "column store"

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

DECOMPOSITION STORAGE MODEL (DSM)

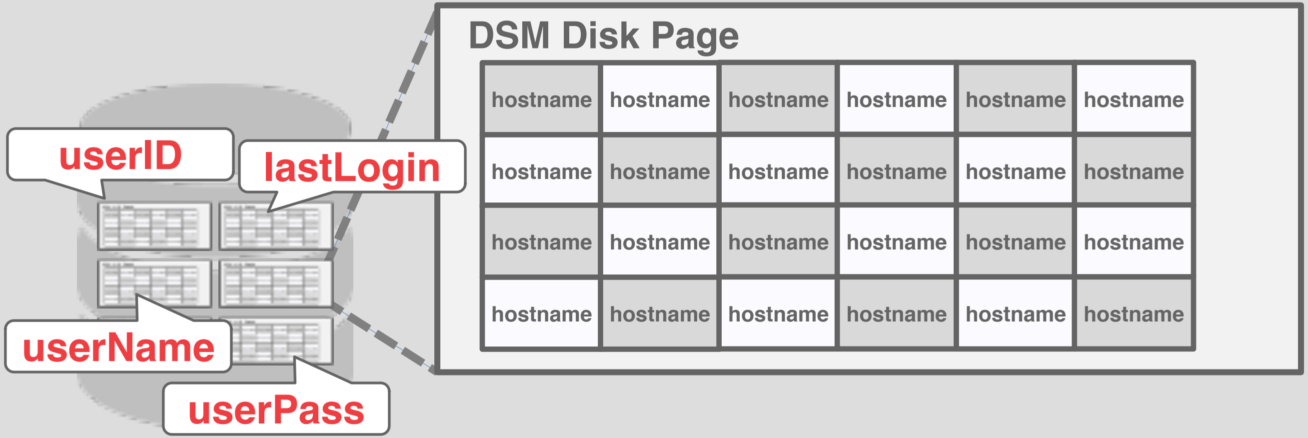
The DBMS stores values of a single attribute across multiple tuples contiguously in a page.

→ Also known as a "column store".



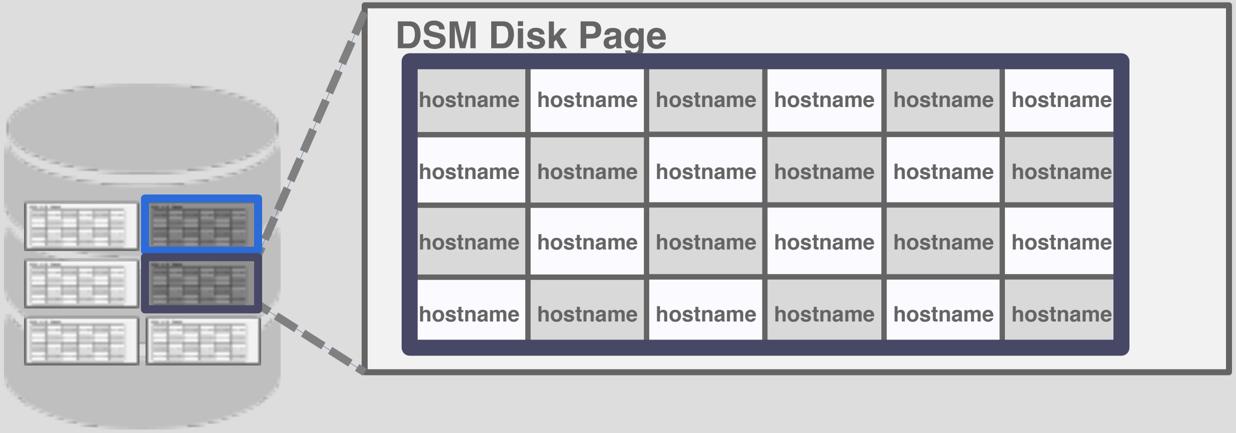
DECOMPOSITION STORAGE MODEL (DSM)

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



DECOMPOSITION STORAGE MODEL (DSM)

```sql
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)

Advantages

→ Reduces the amount of 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

OBSERVATION

I/O is the main bottleneck if the DBMS fetches data from disk during query execution.

The DBMS can **<u>compress</u>** pages to increase the utility of the data moved per I/O operation.

Key trade-off is speed vs. compression ratio

→ Compressing the database reduces DRAM requirements.

→ It <u>may</u> decrease CPU costs during query execution.

CONCLUSION

A database is stored as a series of pages.

There are many different ways to organize pages and store tuples within those pages.

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 memory and transfers data to/from disk.

← Next