University of Michigan

14 Database Storage

Database Management Systems
EECS 484
Fall 2024

Lin Ma
Computer Science and Engineering Division

INSTRUCTOR

Short Bio

→ Grew up in a coastal city in China (Qindao)
→ Undergrad from Peking University (Beijing)
→ PhD from Carnegie Mellon University (Pittsburgh)
→ Software Engineer at Databricks

Office Hour

→ https://calendar.app.google/8uy8668X89L2Sd4P7
→ Conceptual topics on course material, general questions
→ IAs/GSIs are better at answering homework- or project-specific questions

OVERVIEW

We now 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 software that manages a database (i.e., a DBMS).

WHY YOU SHOULD CARE

Skills applicable to various software system problems
→ Caching, Efficiency, Concurrency, Crash Recovery, etc.

Large industry
→ In 2023, the global market for data analytics grew by 13% to $150B (Gartner)

COURSE OUTLINE

Storage

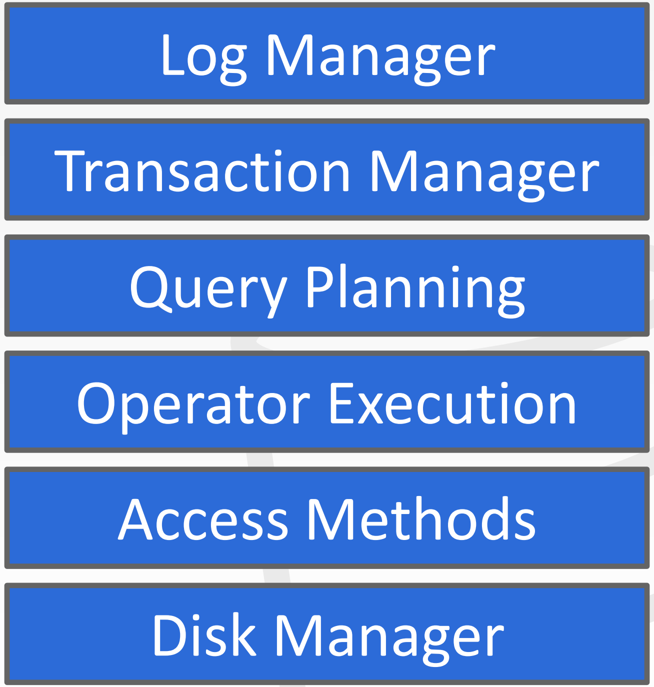
Execution

Planning

Concurrency Control

Recovery

→ Lectures will follow the course schedule at a high-level, but may be faster/slower on specific topics.



COURSE OUTLINE

Storage

Execution

Planning

Concurrency Control

Recovery

→ Lectures will follow the course schedule at a high-level, but may be faster/slower on specific topics.

Log Manager

Transaction Manager

Query Planning

Operator Execution

Access Methods

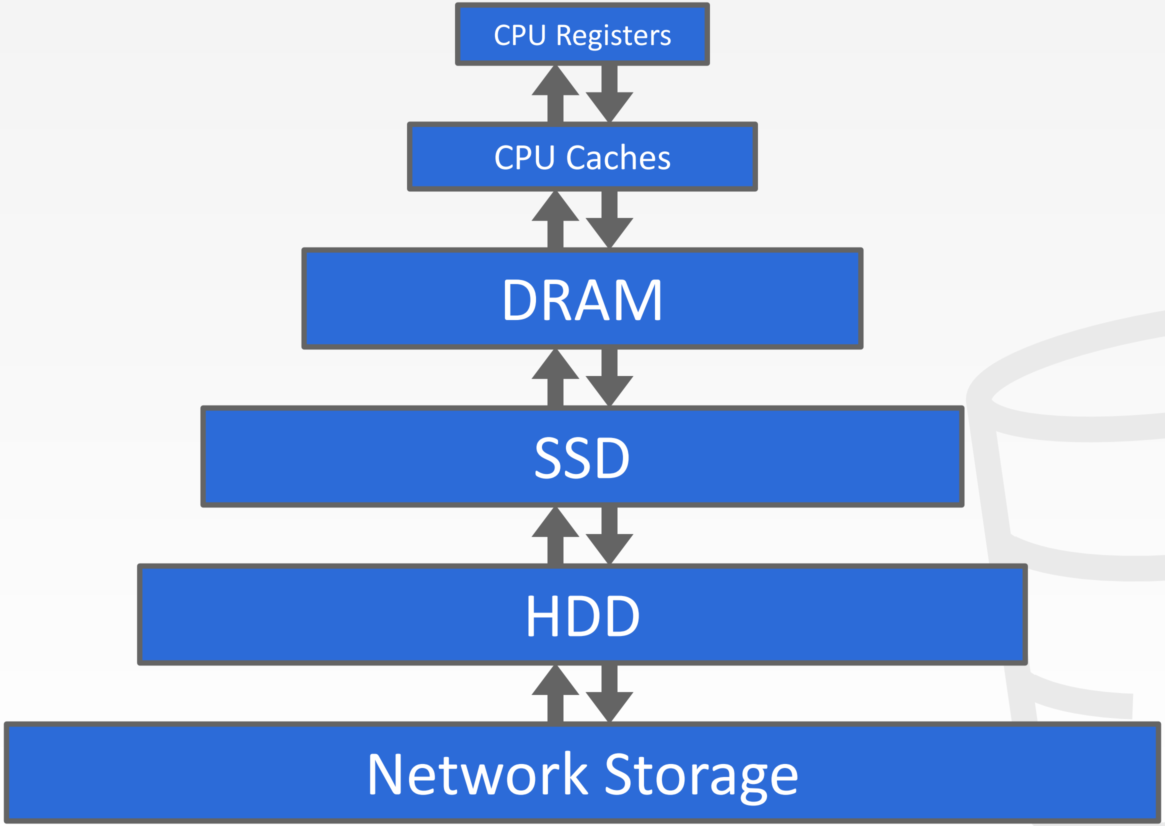
Disk Manager

DISK-BASED ARCHITECTURE

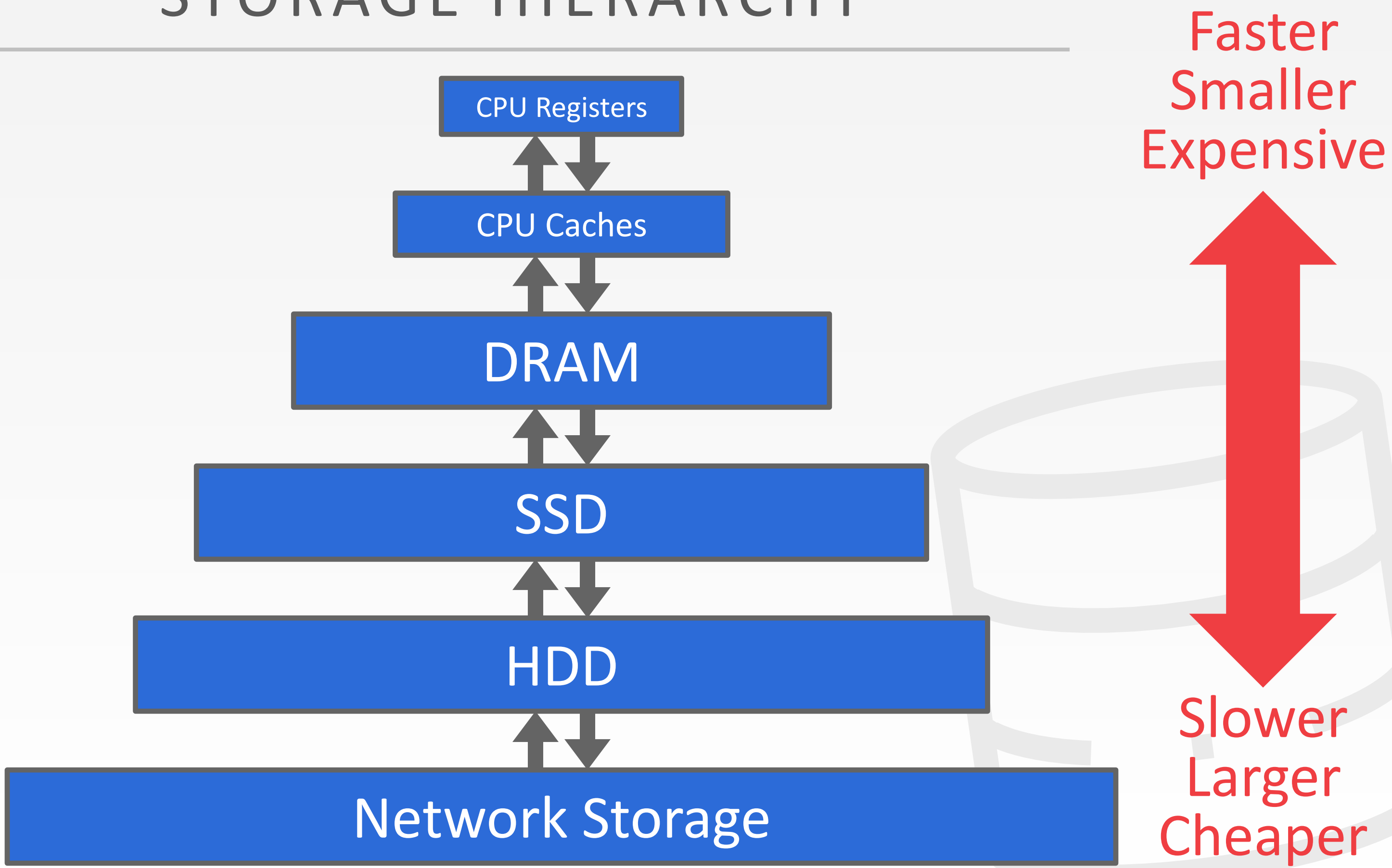
The DBMS assumes that the primary storage location of the database is on non-volatile disk.

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

STORAGE HIERARCHY



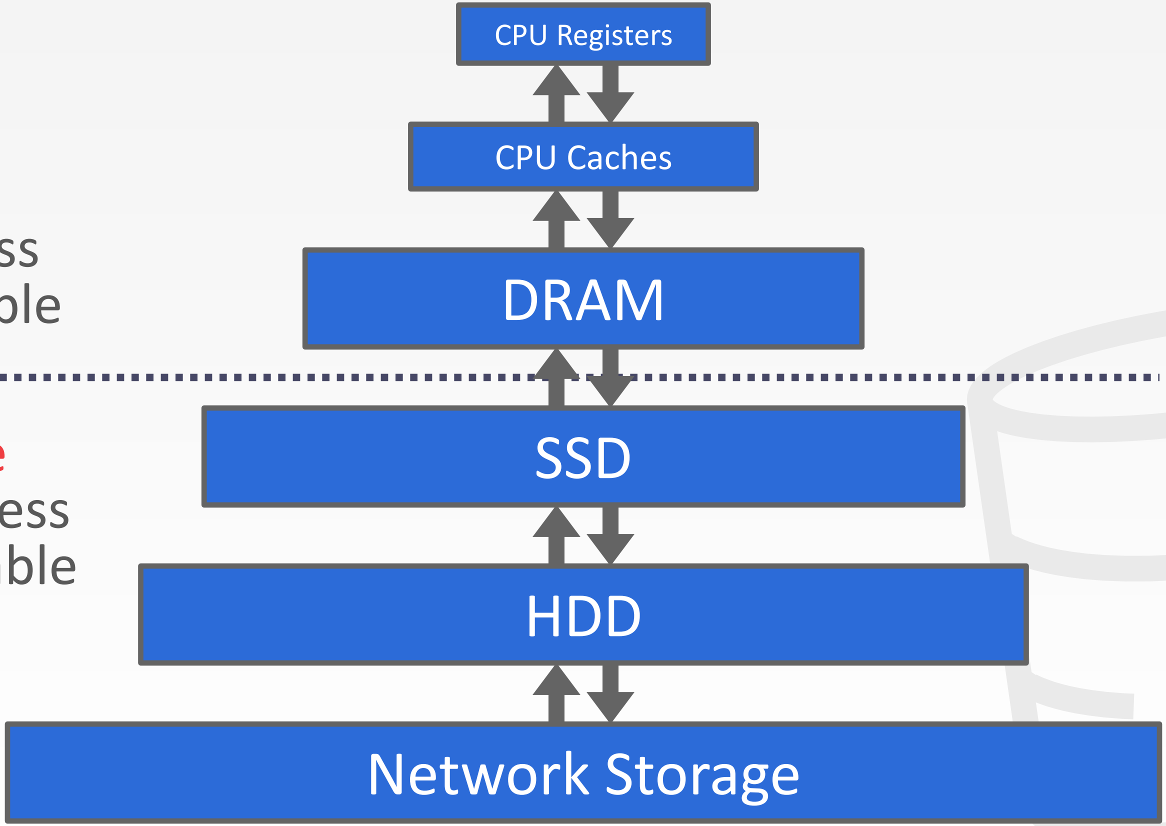
STORAGE HIERARCHY



STORAGE HIERARCHY

Volatile
Random Access
Byte-Addressable

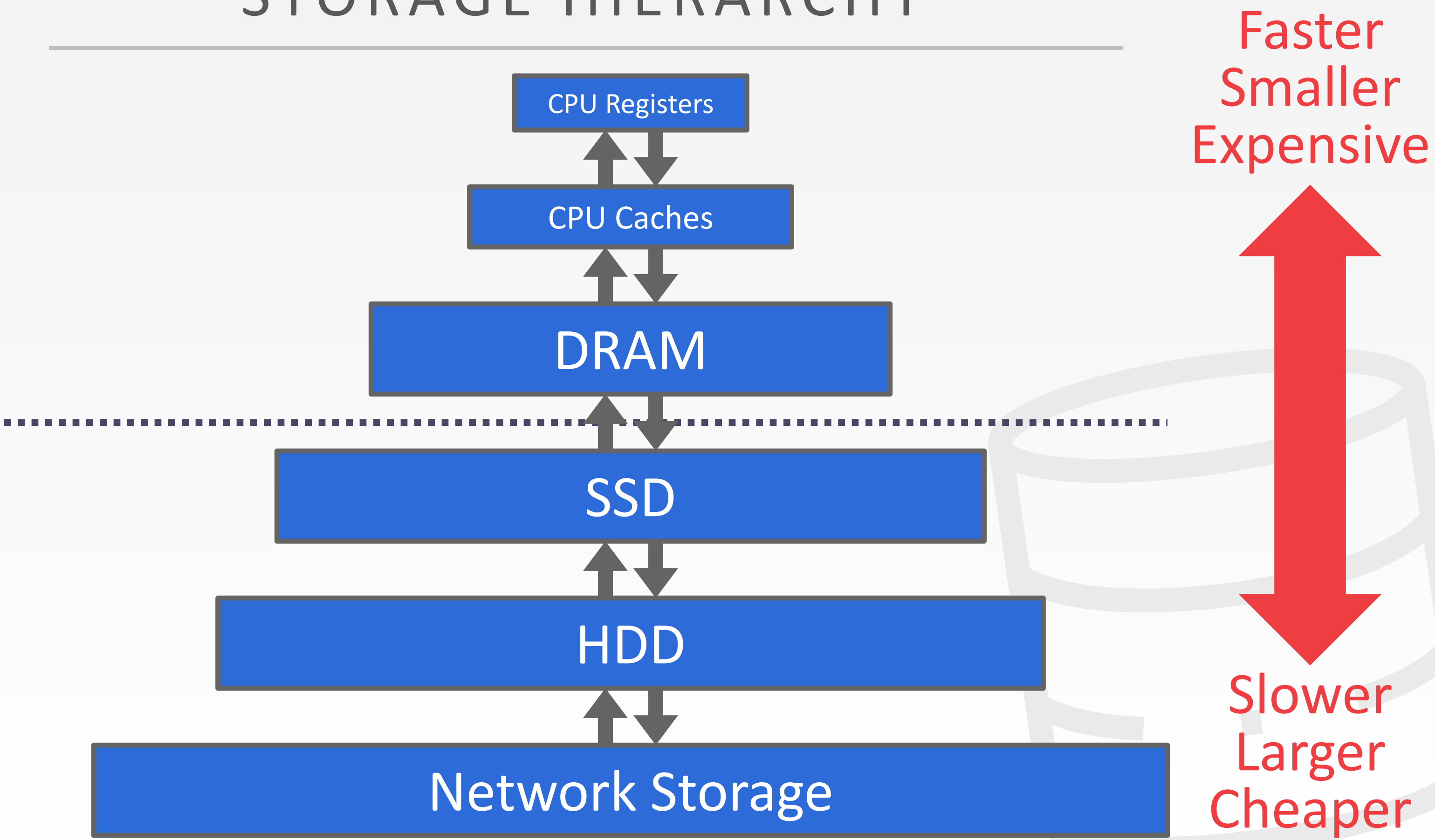
Non-Volatile
Sequential Access
Block-Addressable



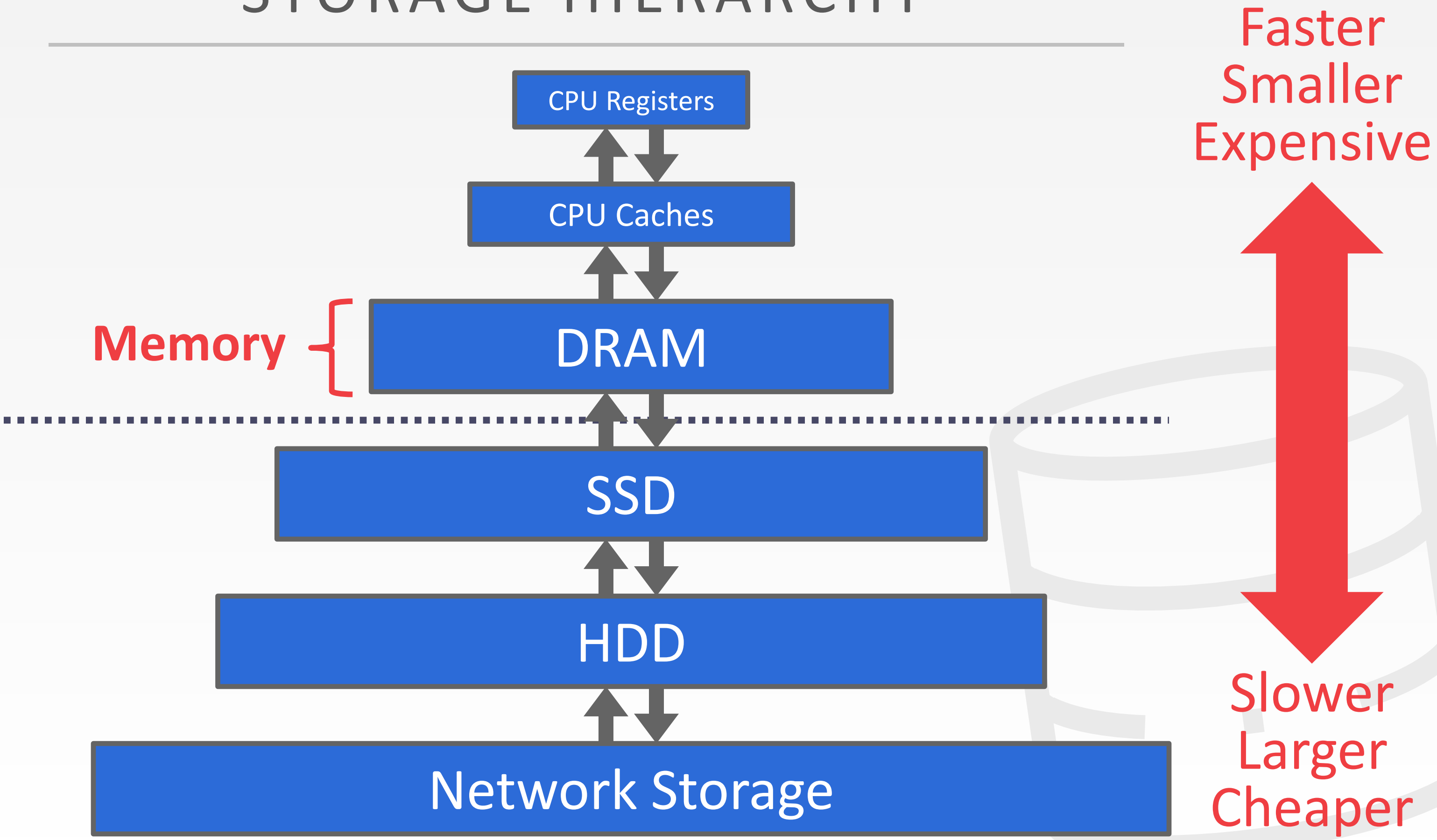
Faster
Smaller
Expensive

Slower
Larger
Cheaper

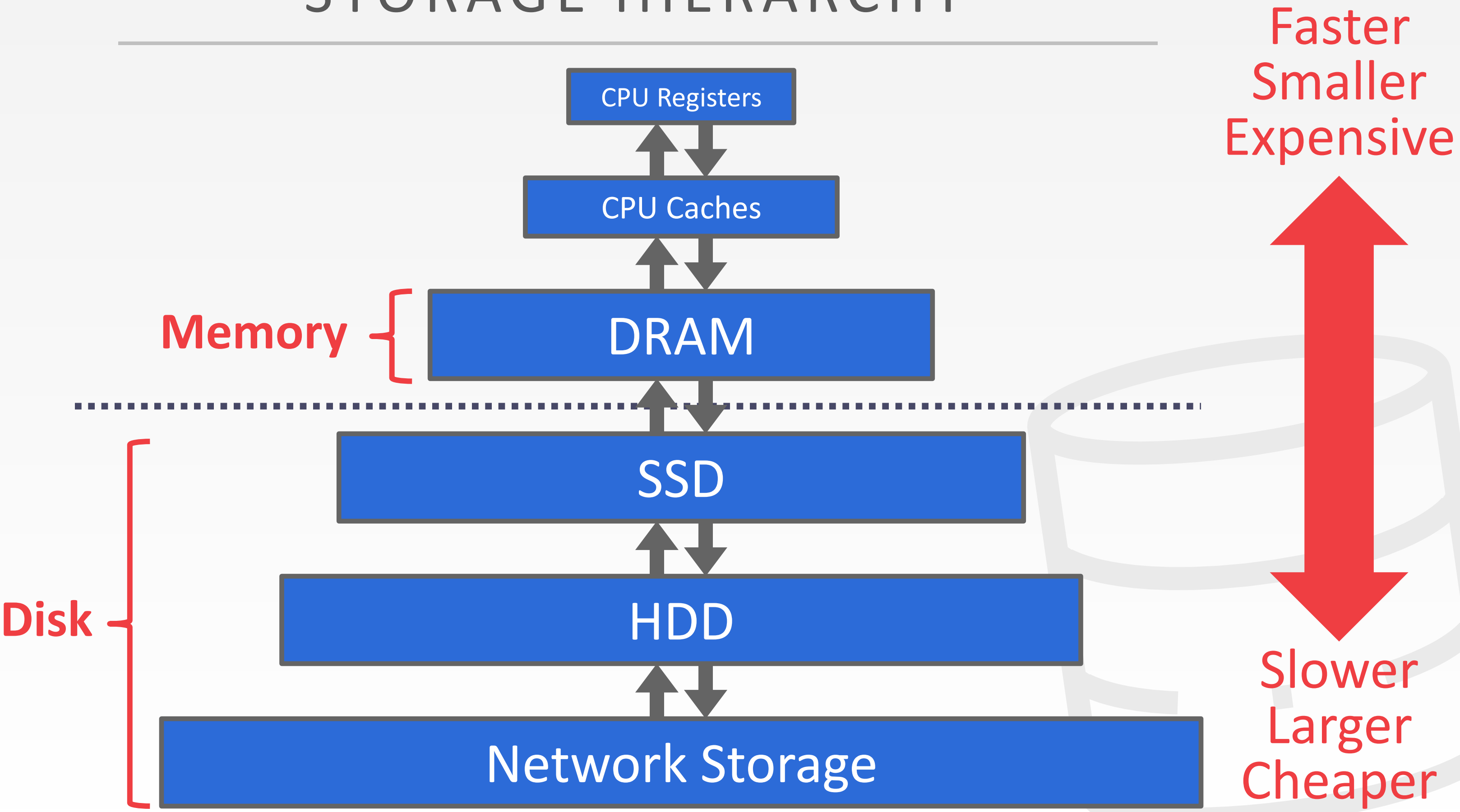
STORAGE HIERARCHY



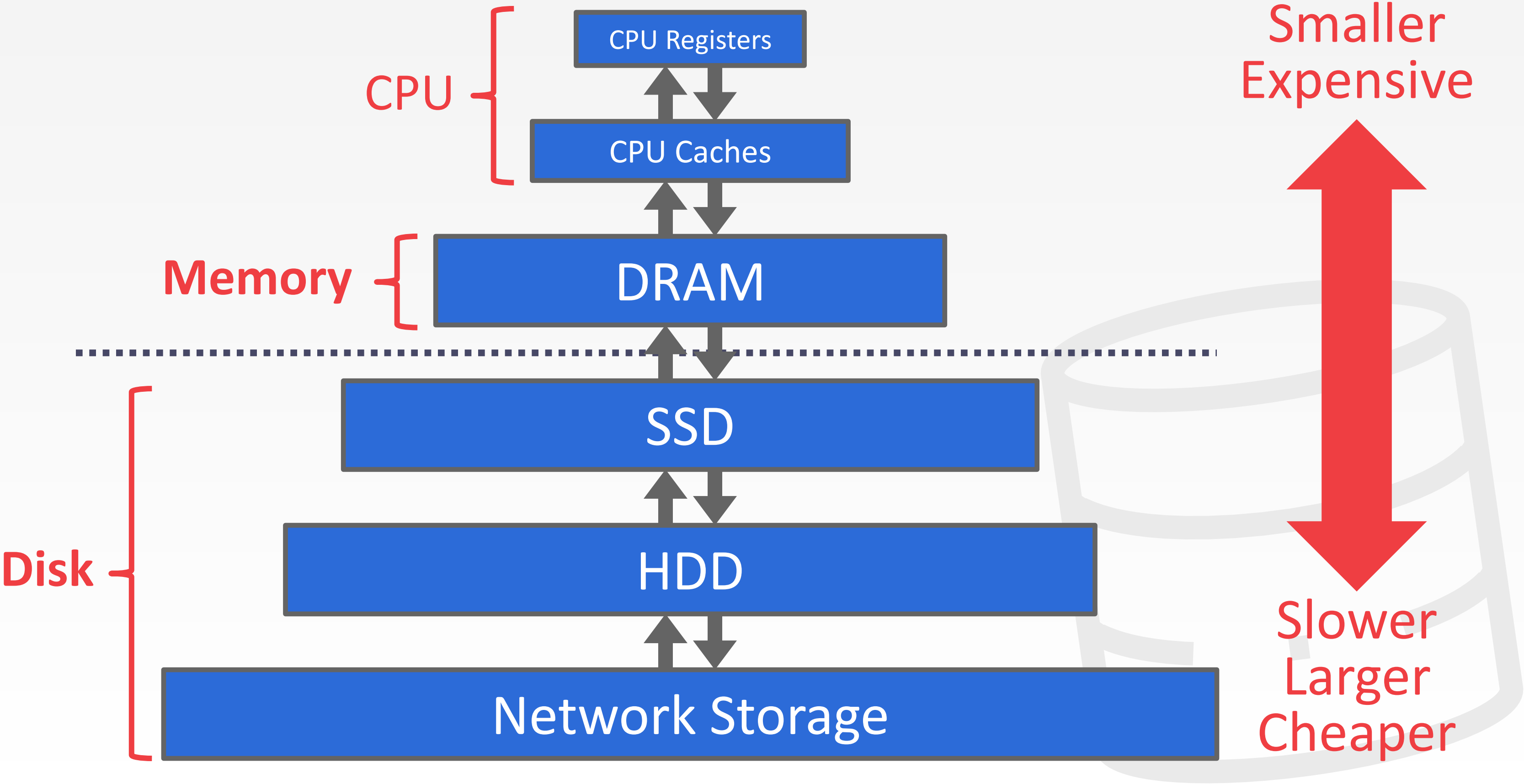
STORAGE HIERARCHY



STORAGE HIERARCHY



STORAGE HIERARCHY



ACCESS TIMES

0.5 ns	L1 Cache Ref
7 ns	L2 Cache Ref
100 ns	DRAM
150,000 ns	SSD
10,000,000 ns	HDD
~30,000,000 ns	Network Storage
1,000,000,000 ns	Tape Archives

[Source]

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

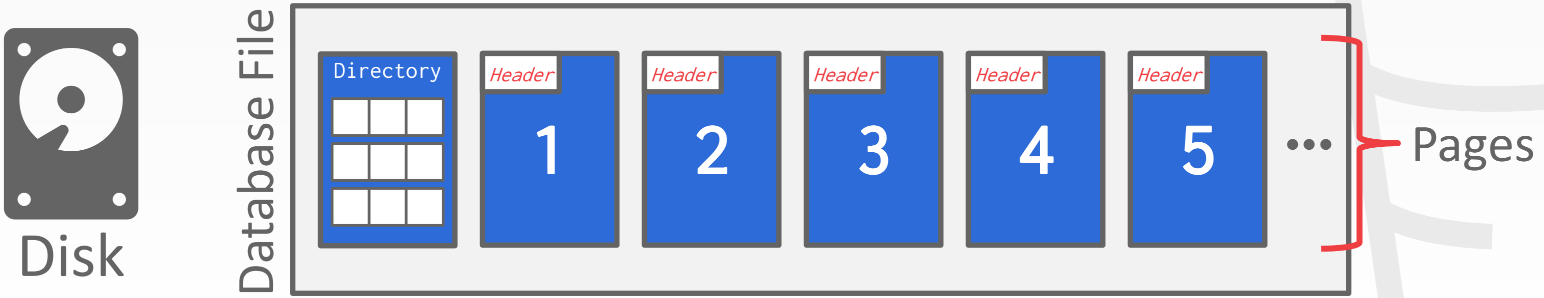
Random access on non-volatile storage is usually much slower than sequential access.

DBMS will want to maximize sequential access.

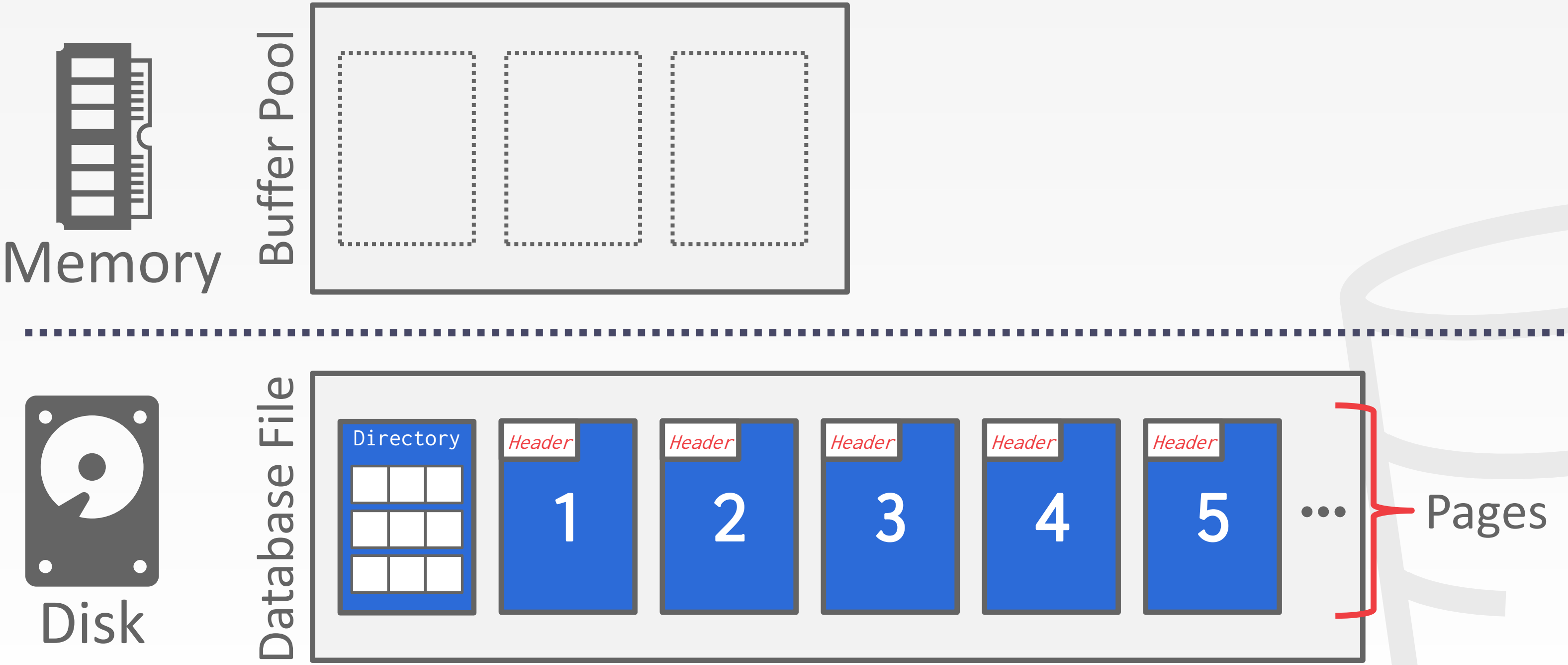
→ Algorithms try to reduce number of writes to random pages so that data is stored in contiguous blocks.

DISK-ORIENTED DBMS

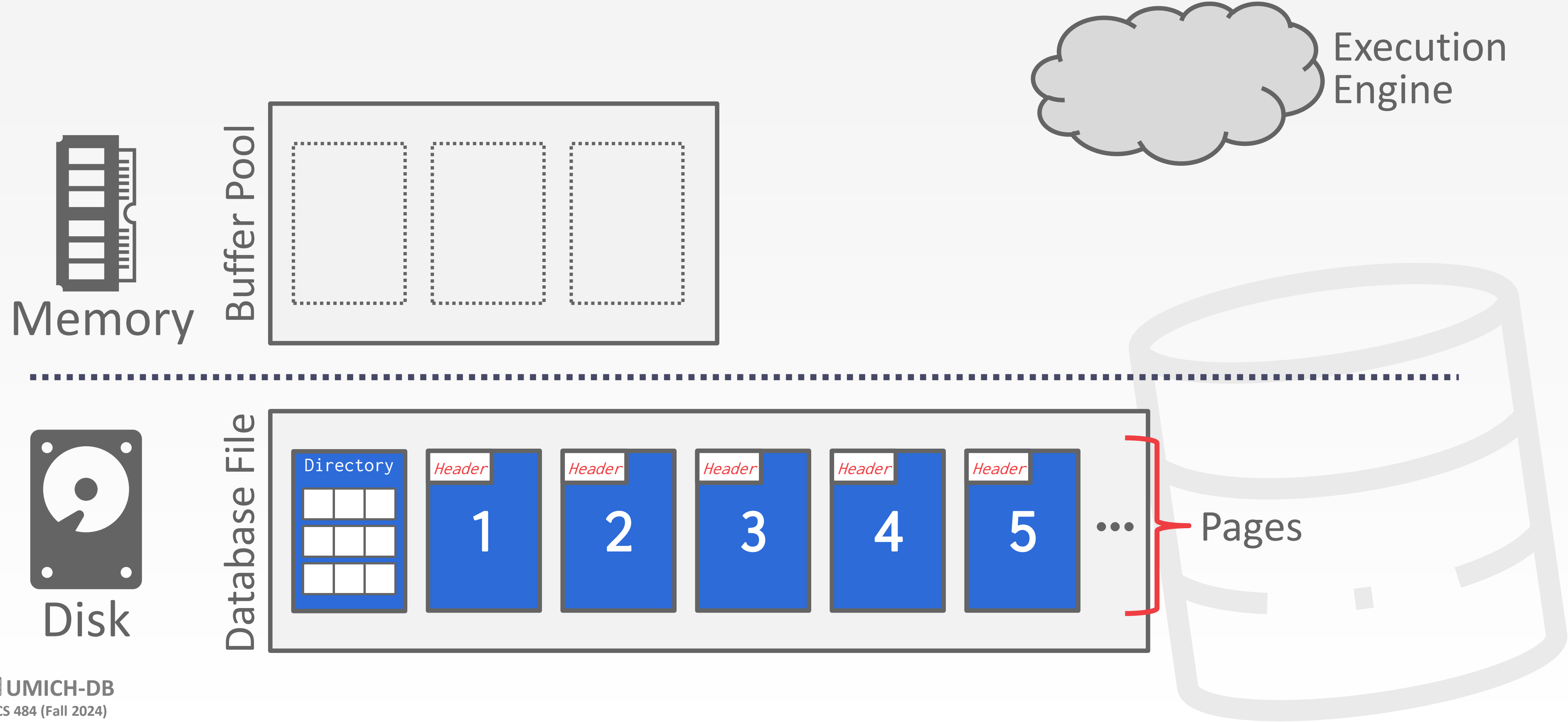
DISK-ORIENTED DBMS



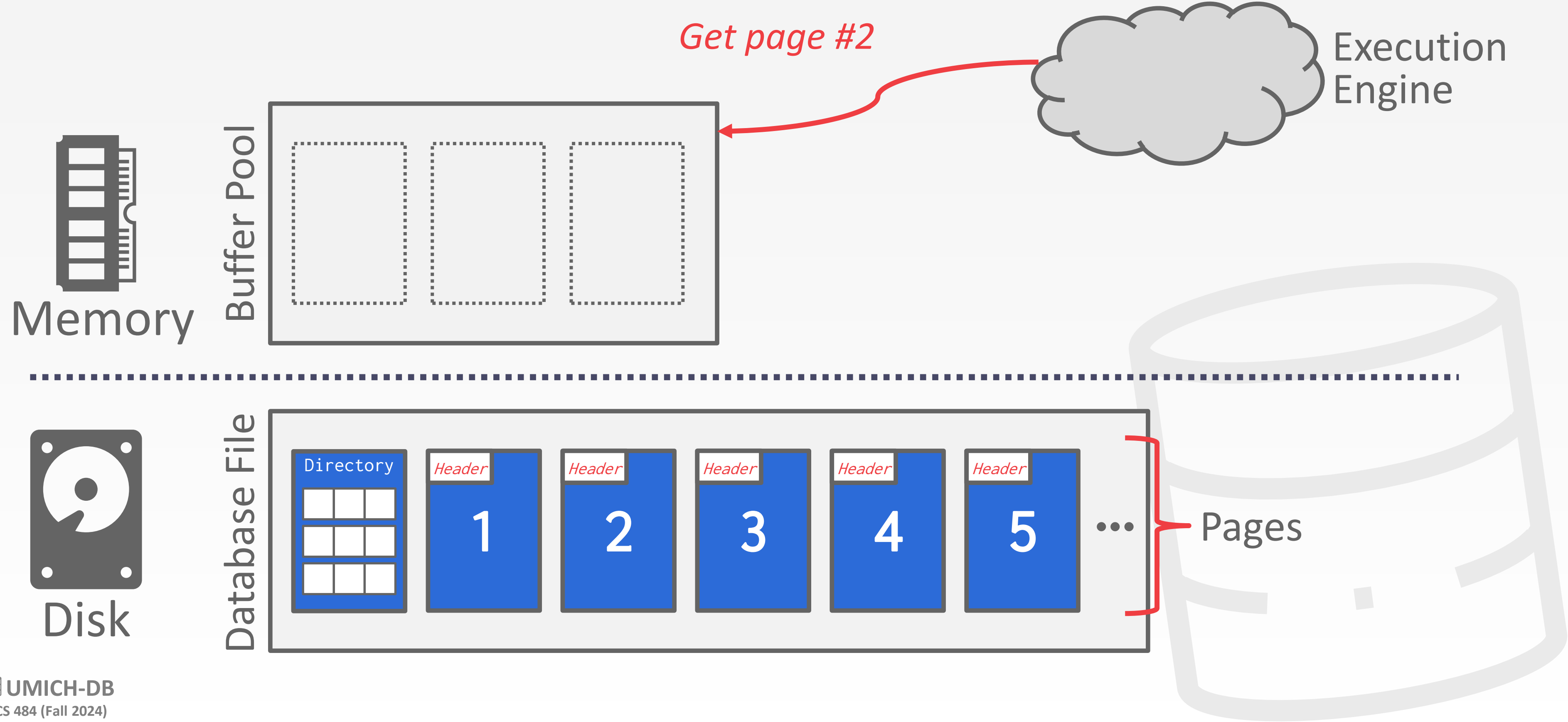
DISK-ORIENTED DBMS



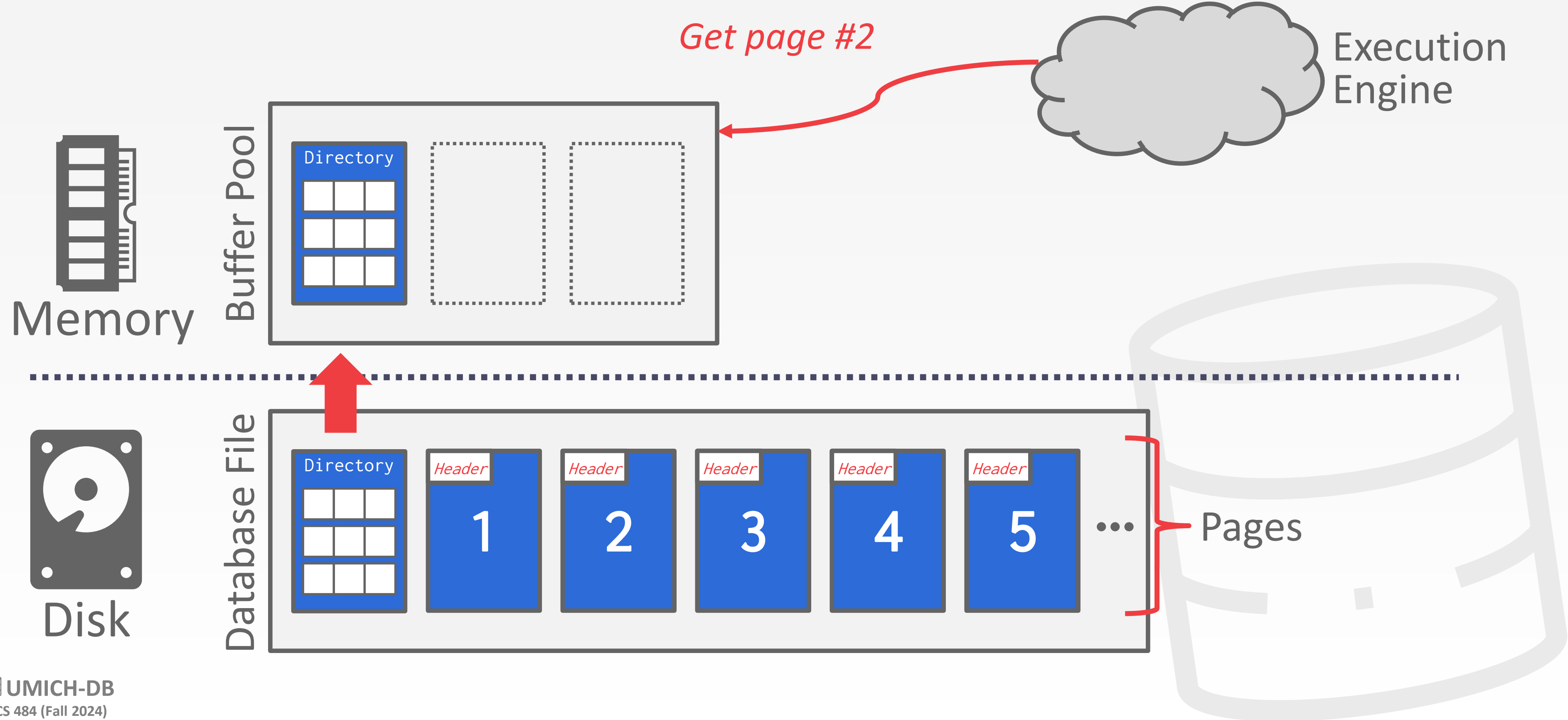
DISK-ORIENTED DBMS



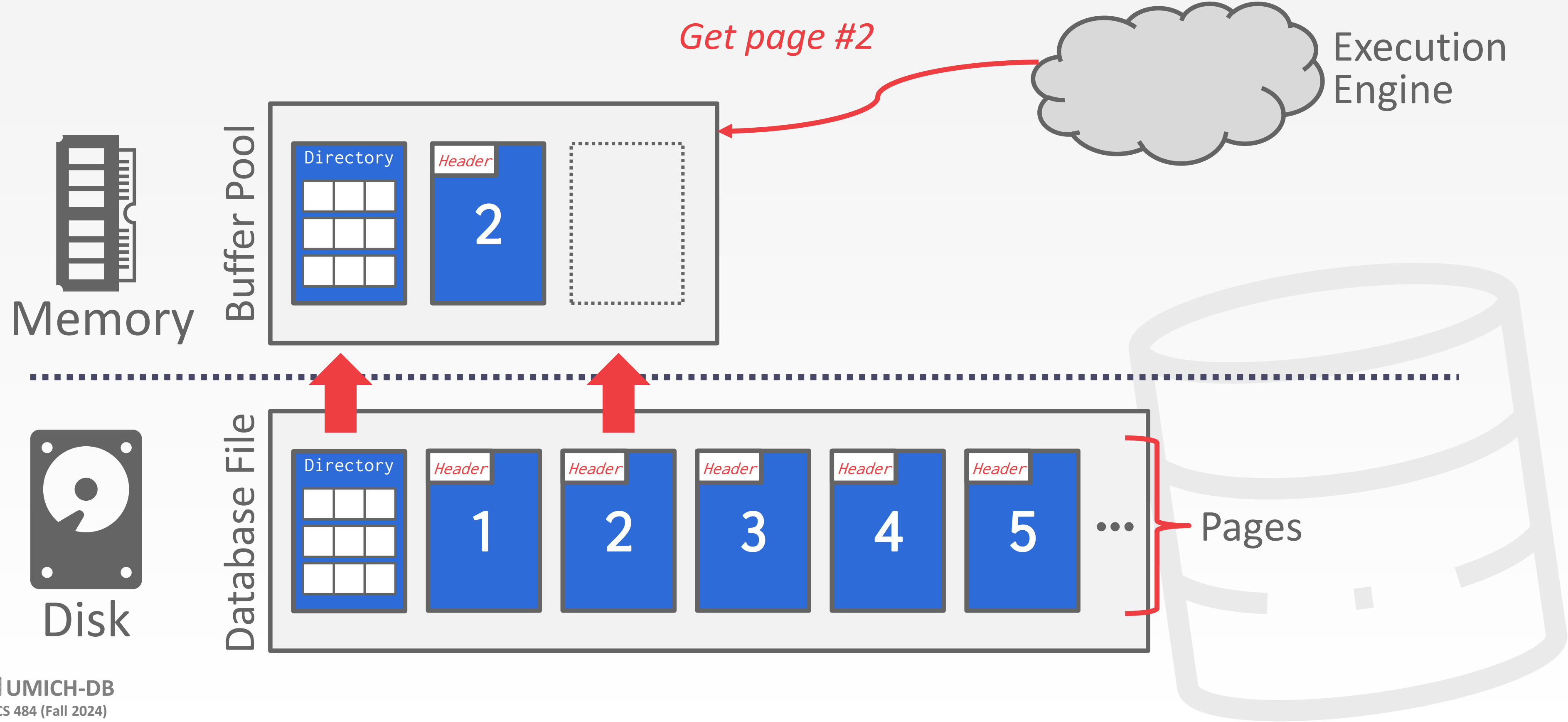
DISK-ORIENTED DBMS



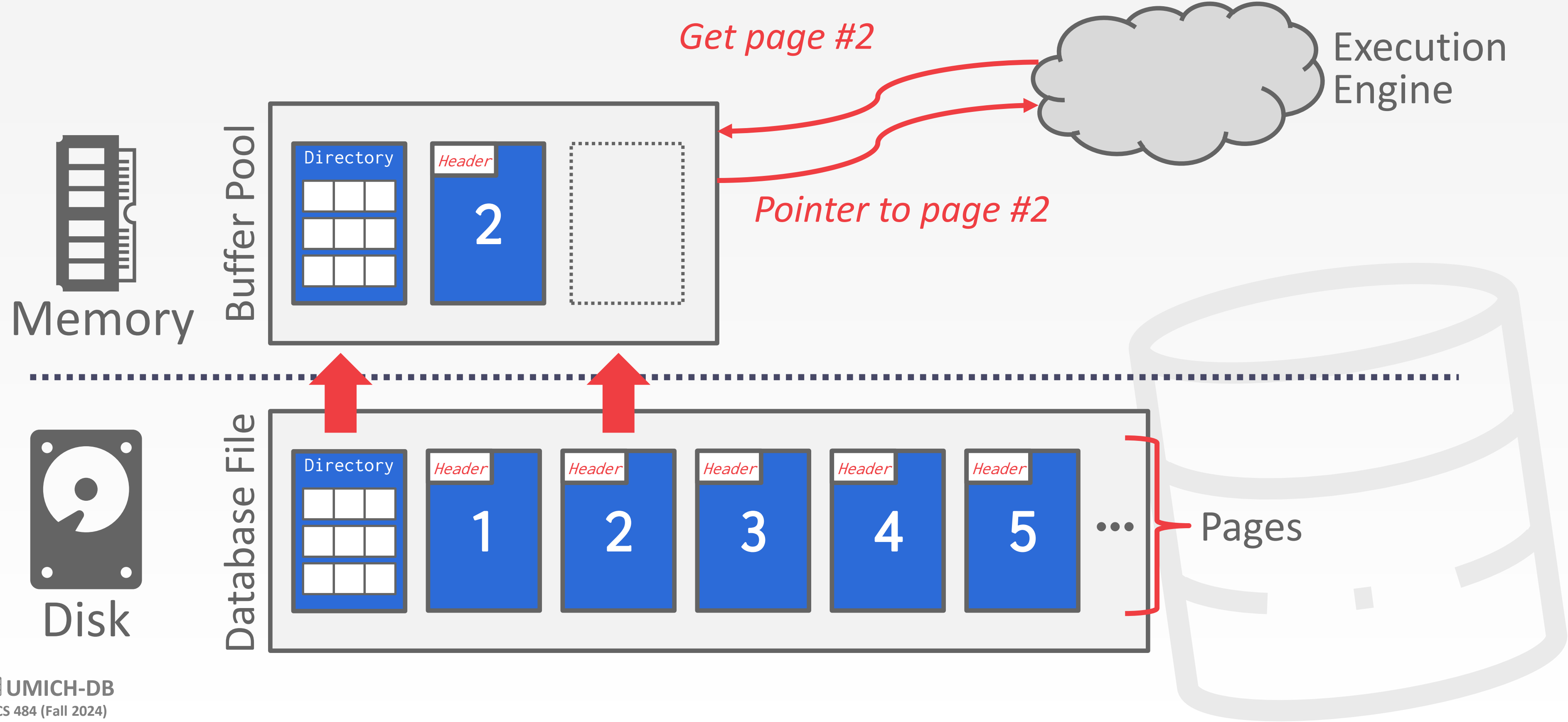
DISK-ORIENTED DBMS



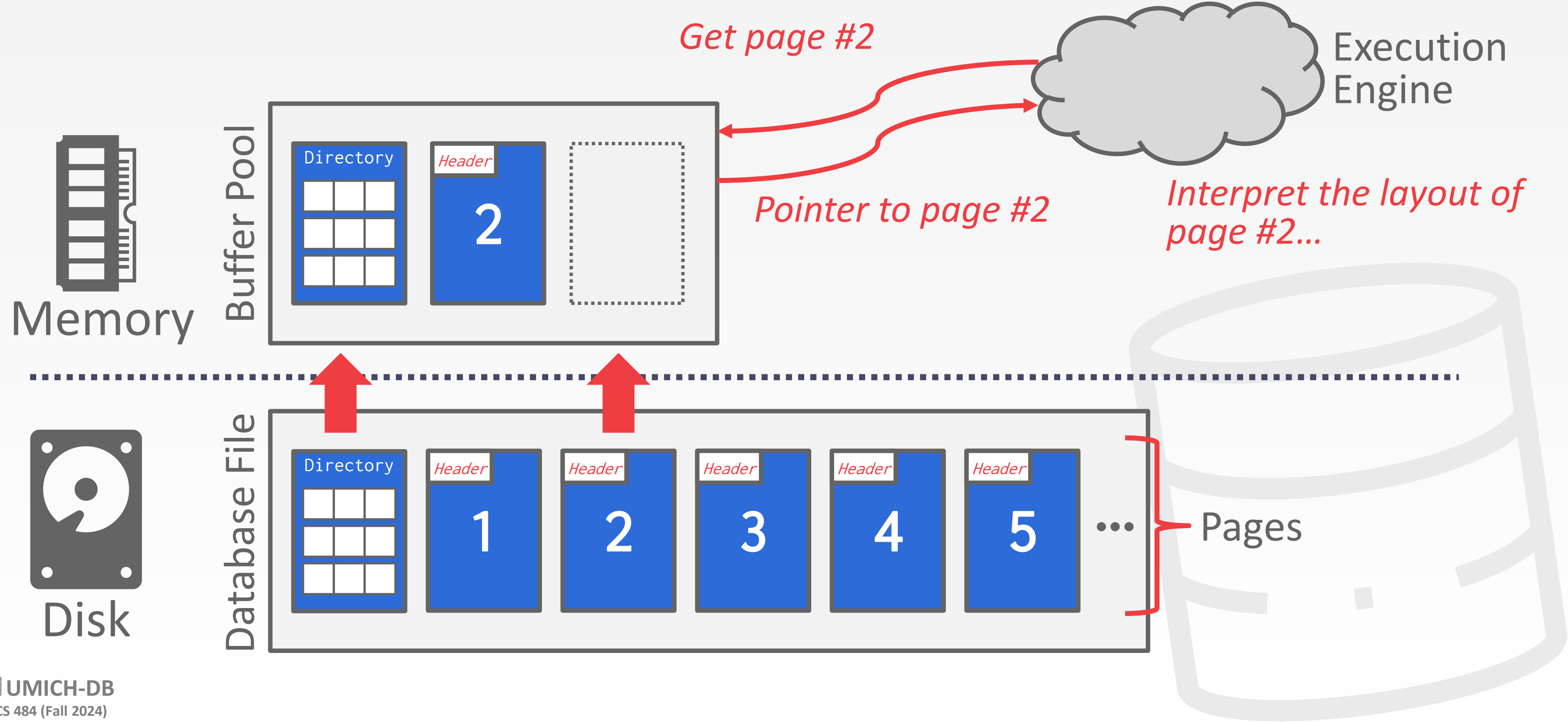
DISK-ORIENTED DBMS



DISK-ORIENTED DBMS



DISK-ORIENTED DBMS



WHY NOT USE THE OS?

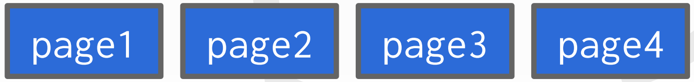
The DBMS can use memory mapping (**mmap**) to store the contents of a file into the address space of a program.

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.

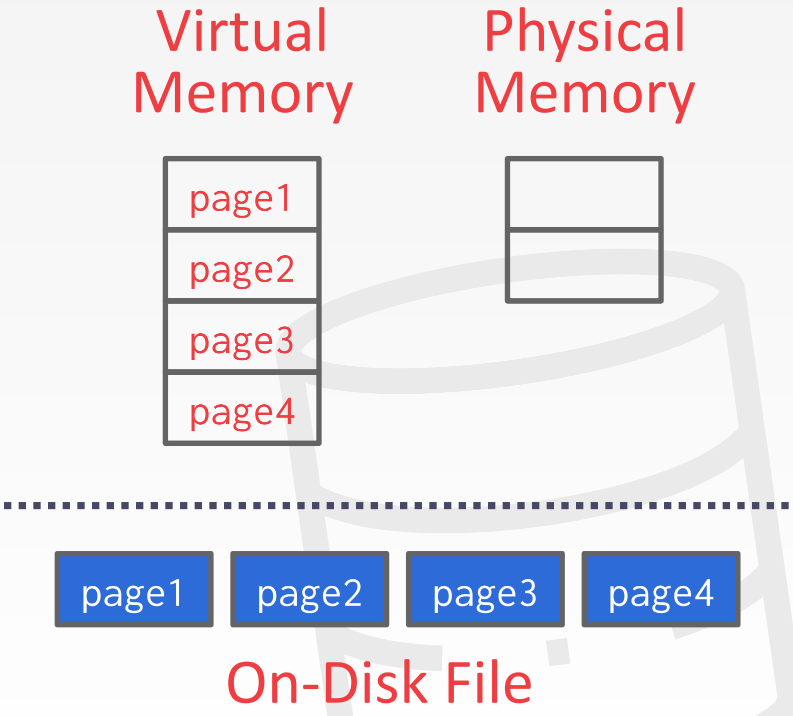


On-Disk File

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.

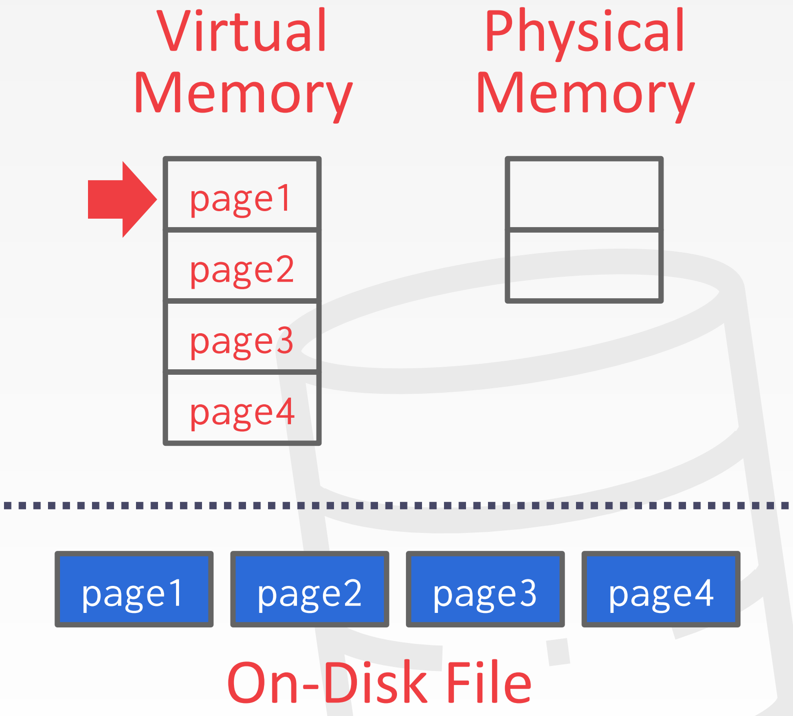
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

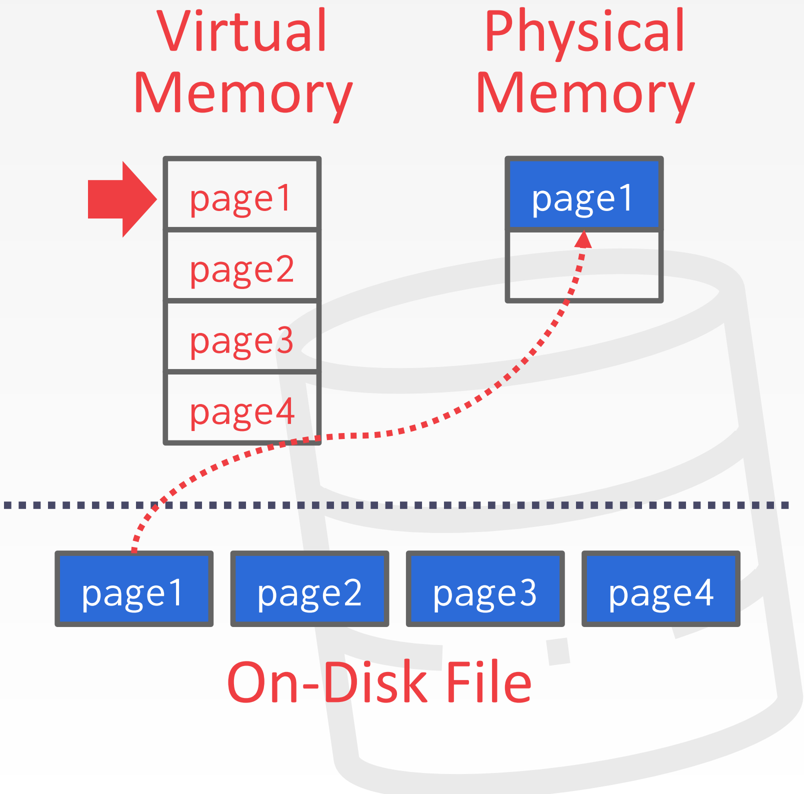
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

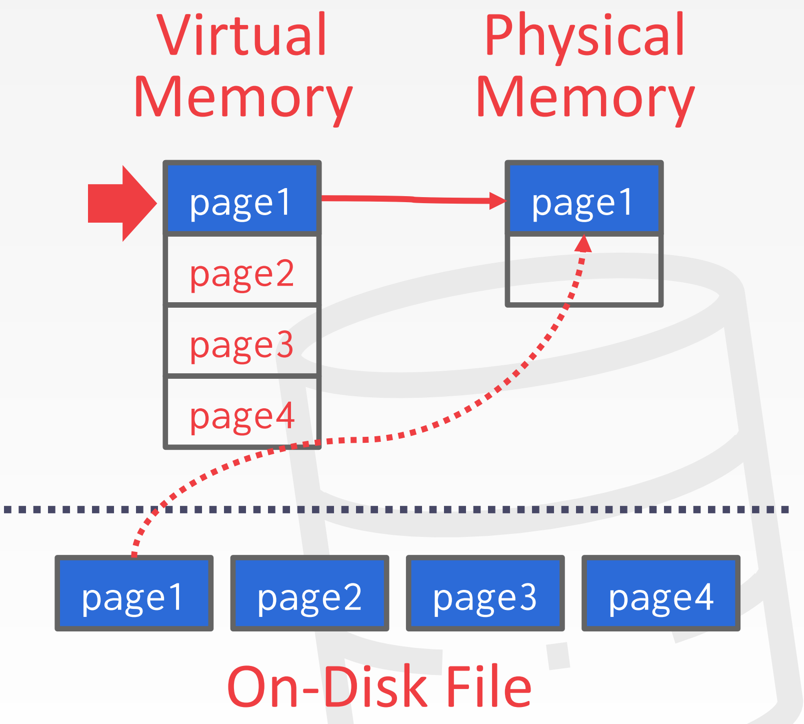
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



WHY NOT USE THE OS?

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

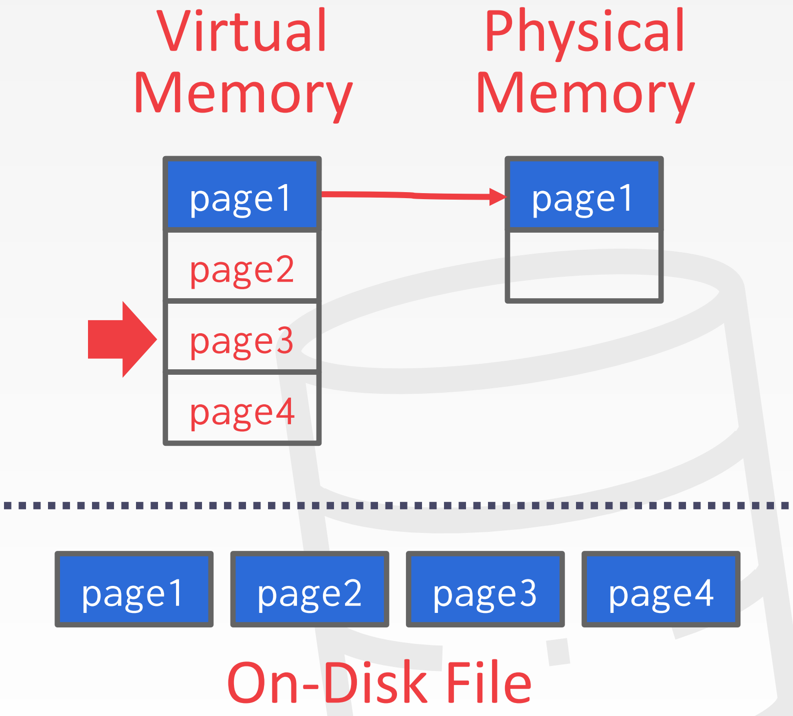
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

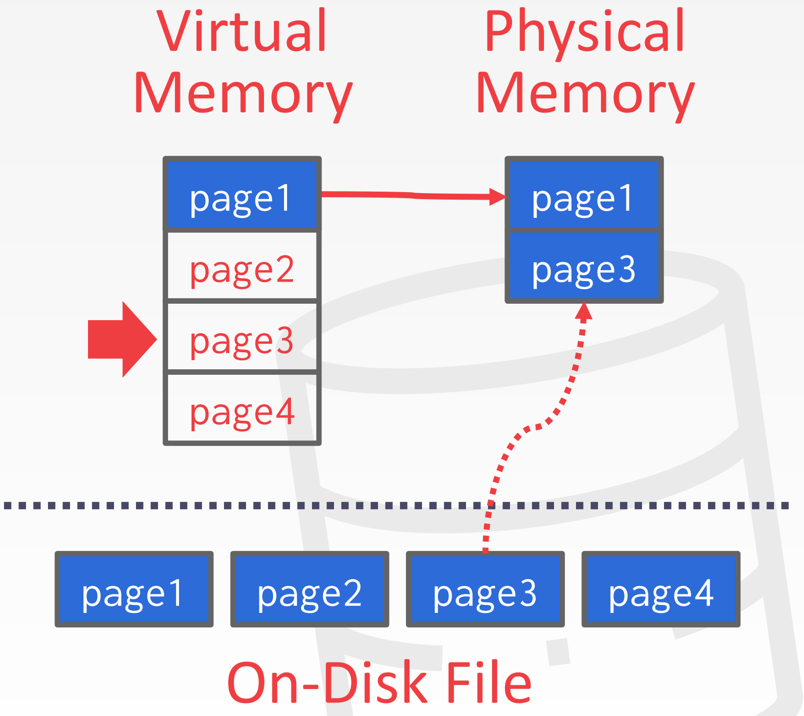
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

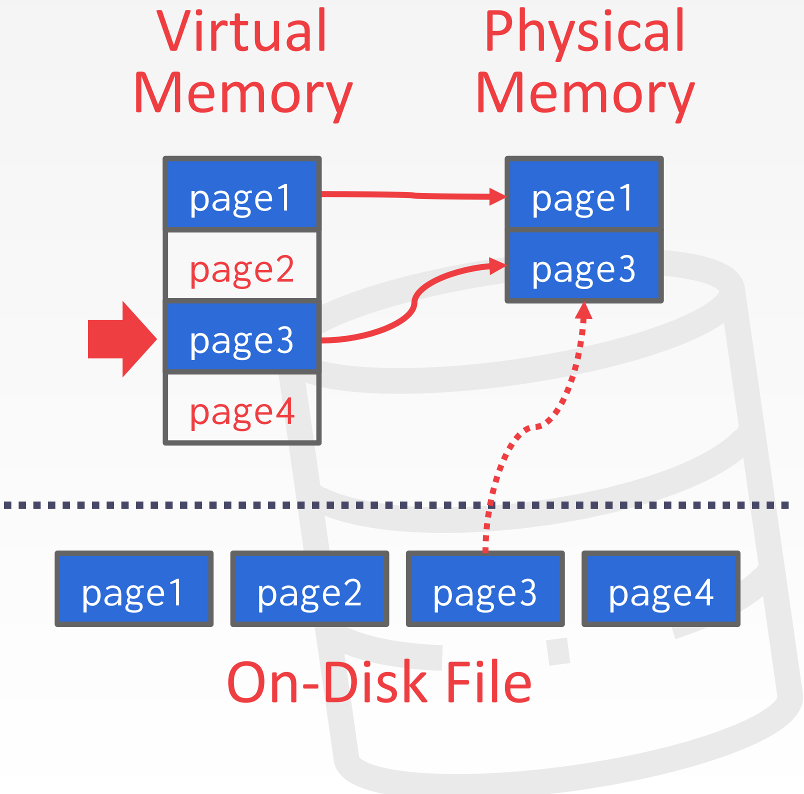
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



WHY NOT USE THE OS?

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

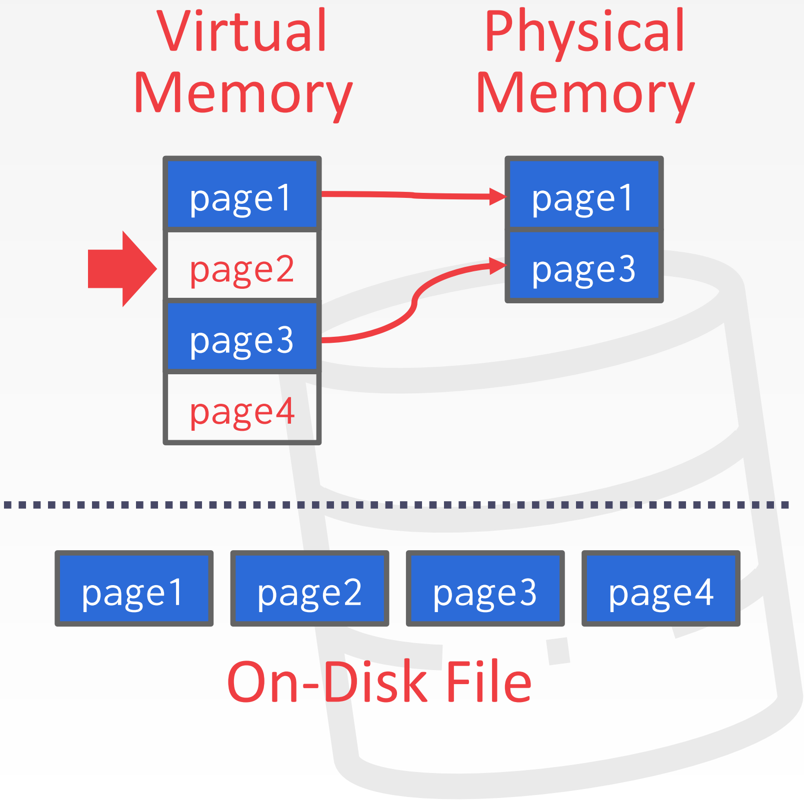
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

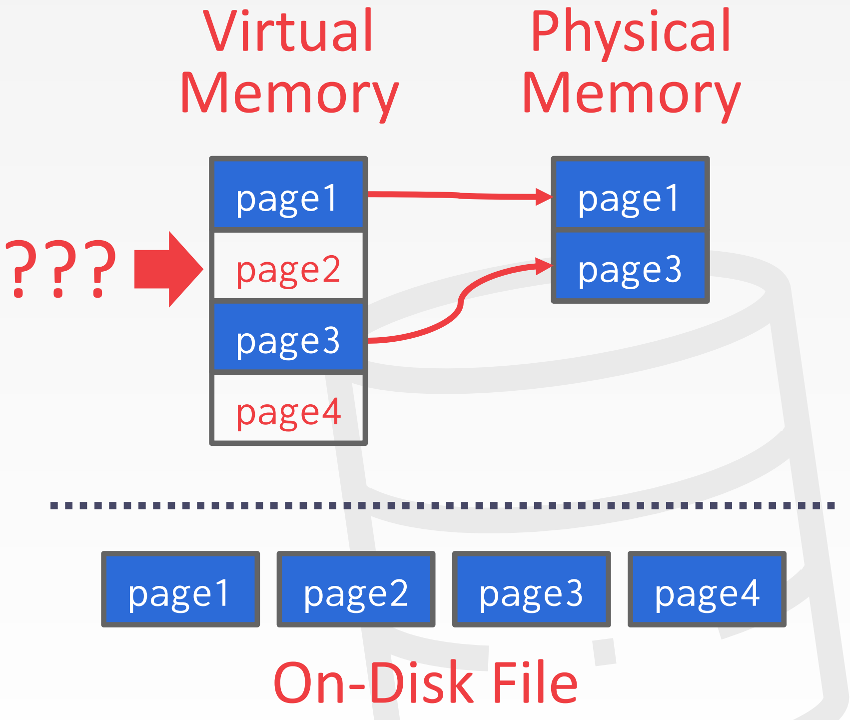
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

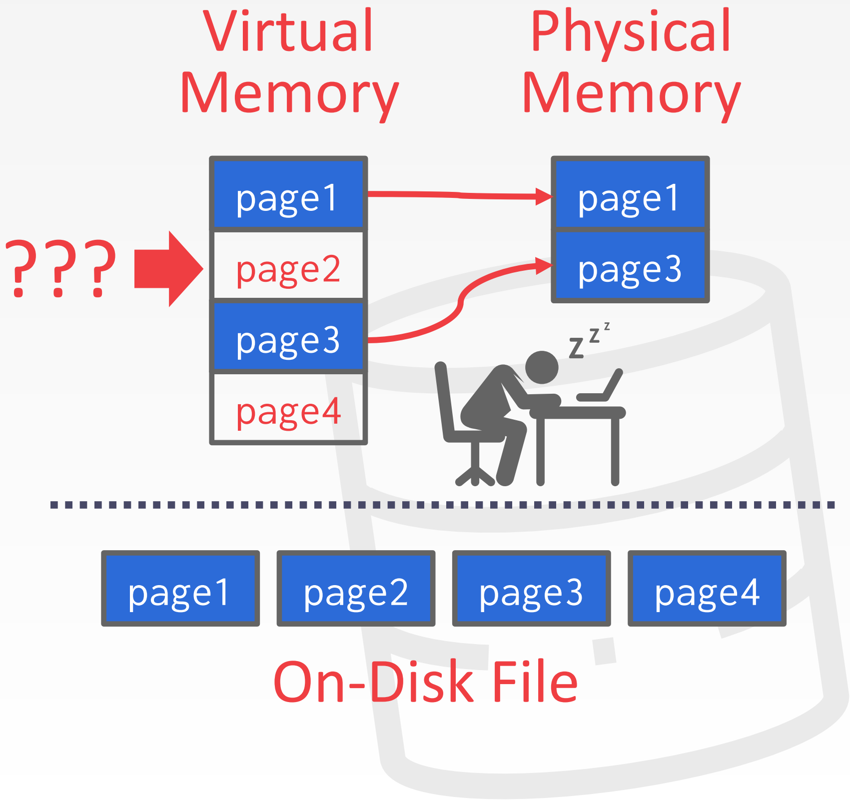
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



WHY NOT USE THE OS?

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 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.

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?

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 than the OS.

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

TODAY'S AGENDA

File Storage

Page Layout

Storage Models

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.

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: SQLite, IBM DB2, ORACLE

8KB: Microsoft SQL Server, PostgreSQL

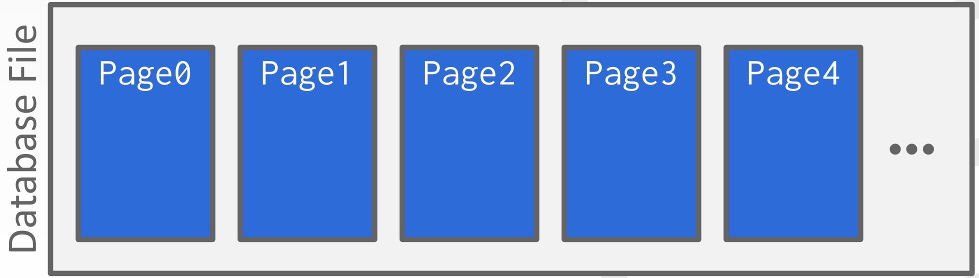
16KB: MySQL

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.

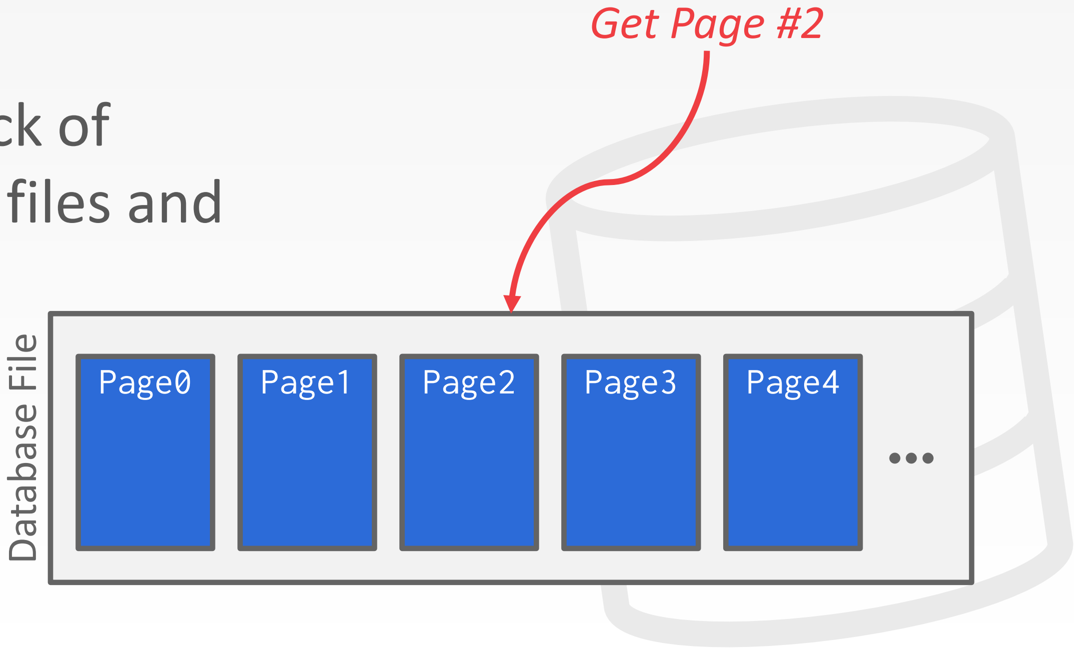
→ Page Directory



DATABASE HEAP

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

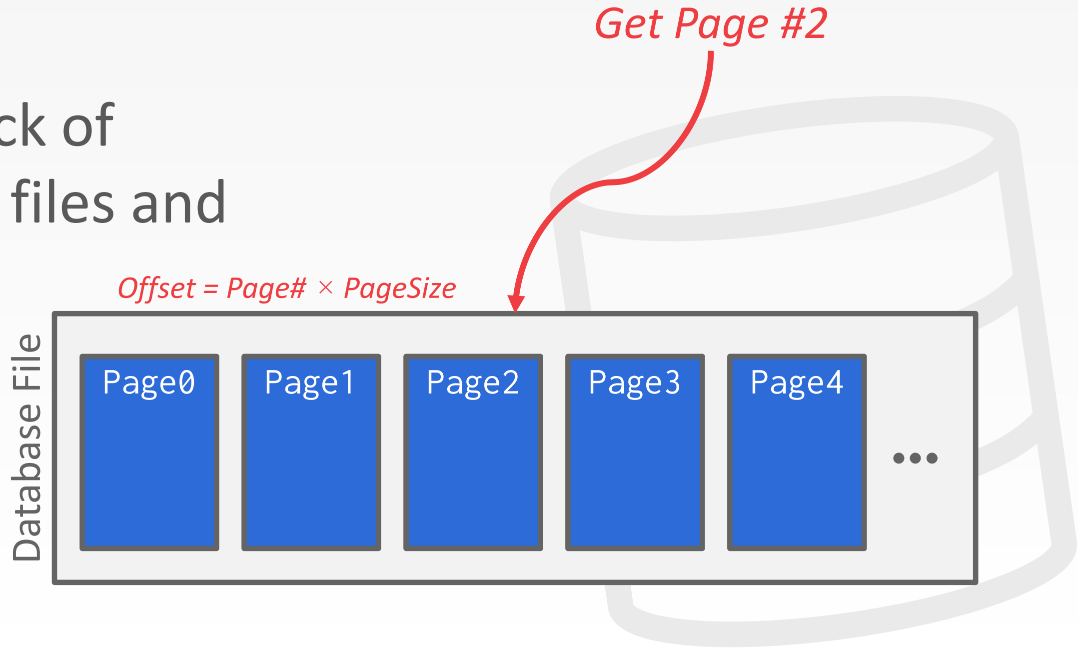
Need meta-data to keep track of what pages exist in multiple files and which ones have free space.
→ Page Directory



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.
→ Page Directory

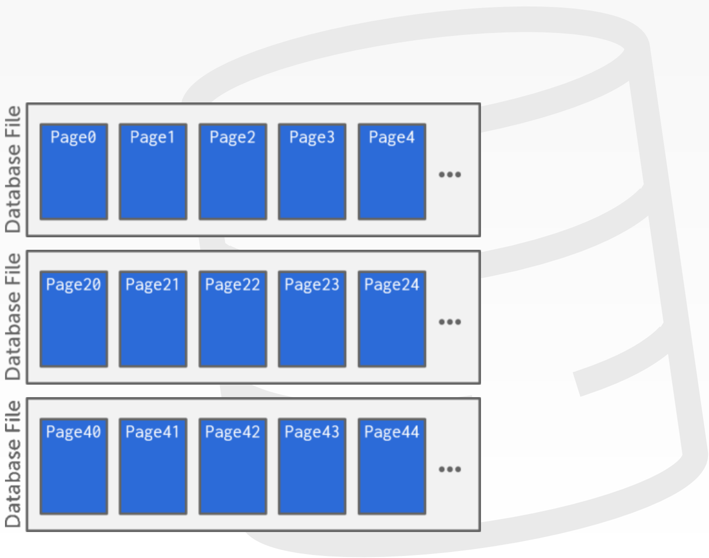


DATABASE HEAP

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.
→ Page Directory

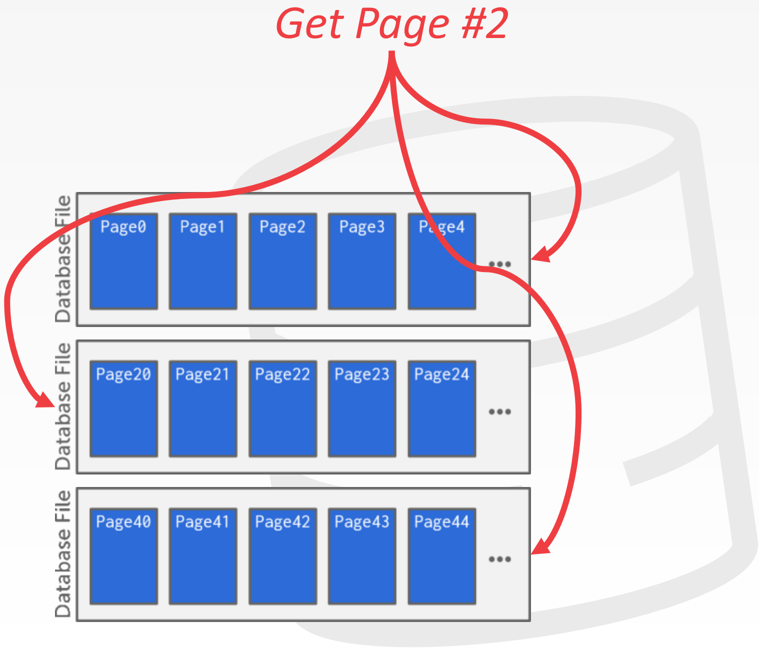
Get Page #2



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.
→ Page Directory



TODAY'S AGENDA

File Storage

Page Layout

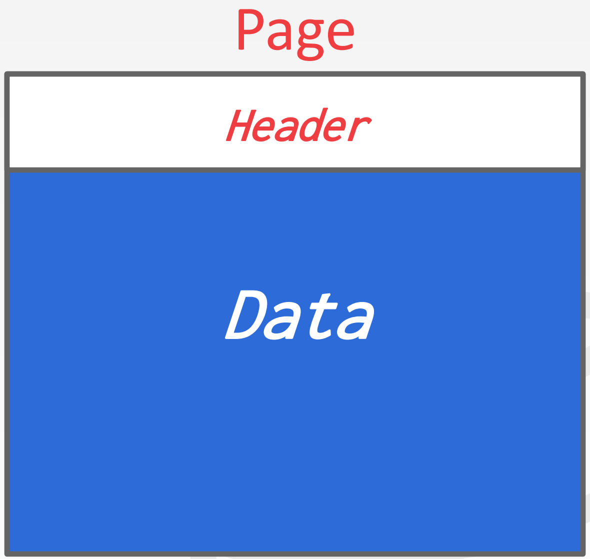
Storage Models

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 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

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.

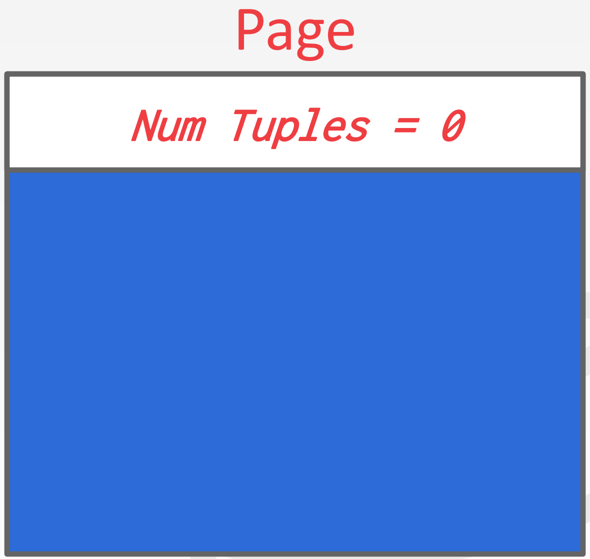
Two approaches:

→ Tuple-oriented

→ Log-structured

TUPLE STORAGE

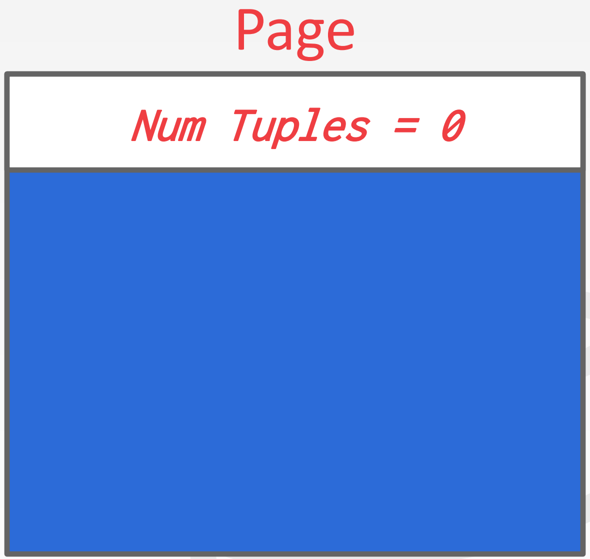
How to store tuples in a page?



TUPLE STORAGE

How to store tuples in a page?

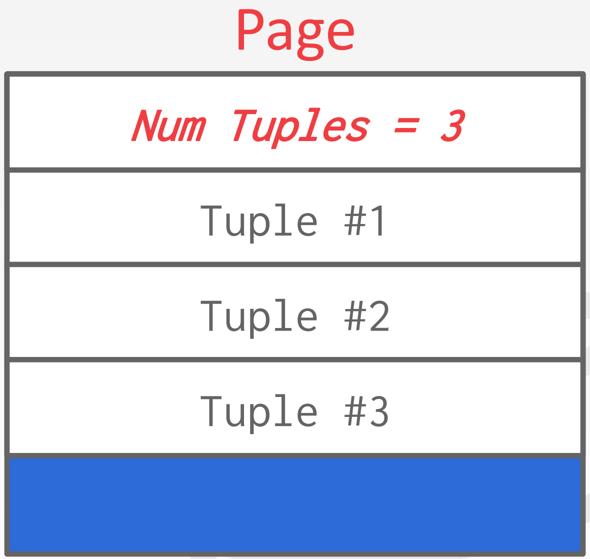
Strawman Idea: Keep track of the number of tuples in a page and then just append a new tuple to the end.



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.

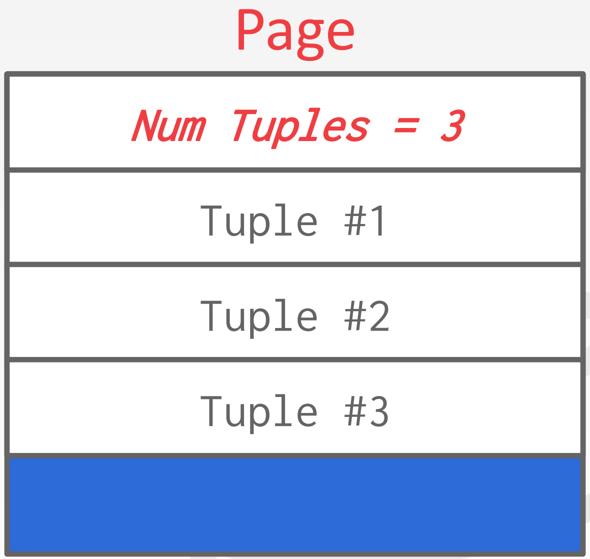


TUPLE STORAGE

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?

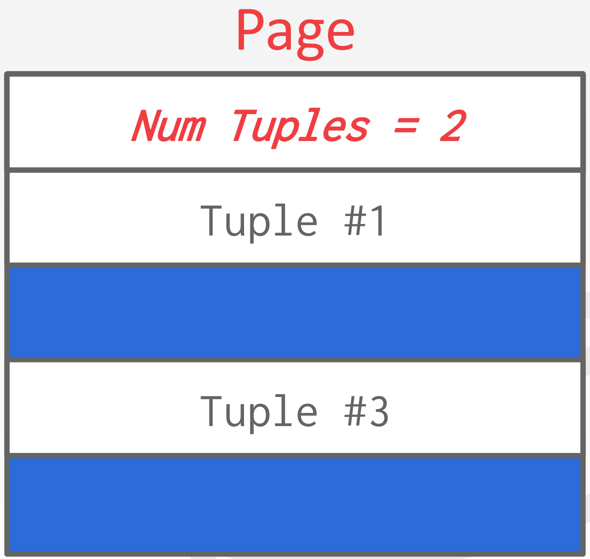


TUPLE STORAGE

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?

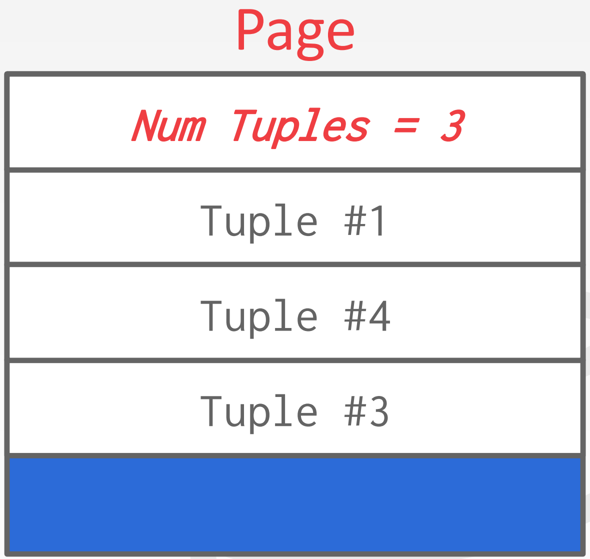


TUPLE STORAGE

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?

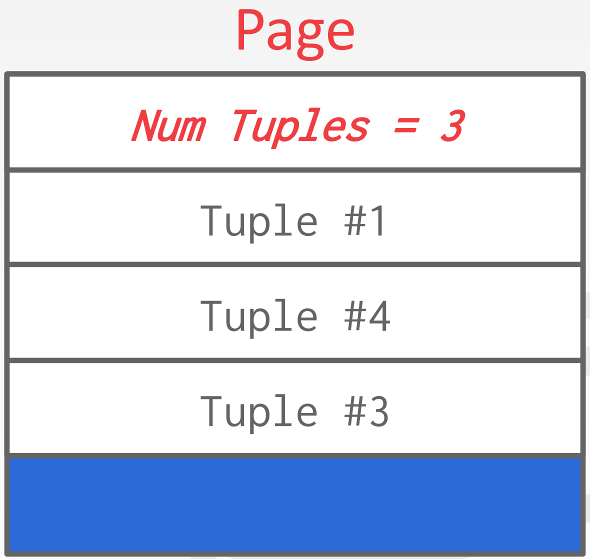


TUPLE STORAGE

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

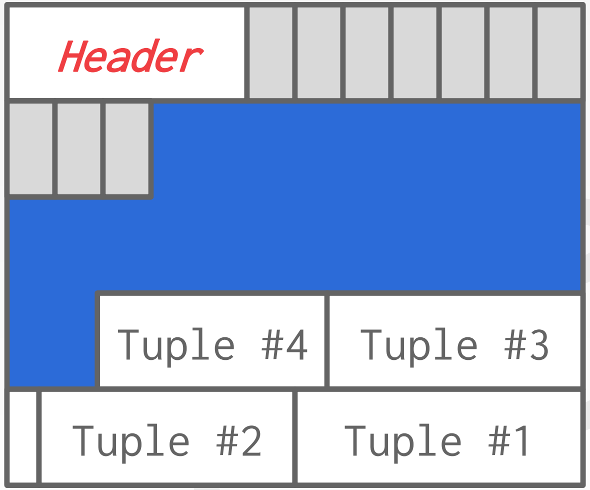
The most common layout scheme is called slotted pages.

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.

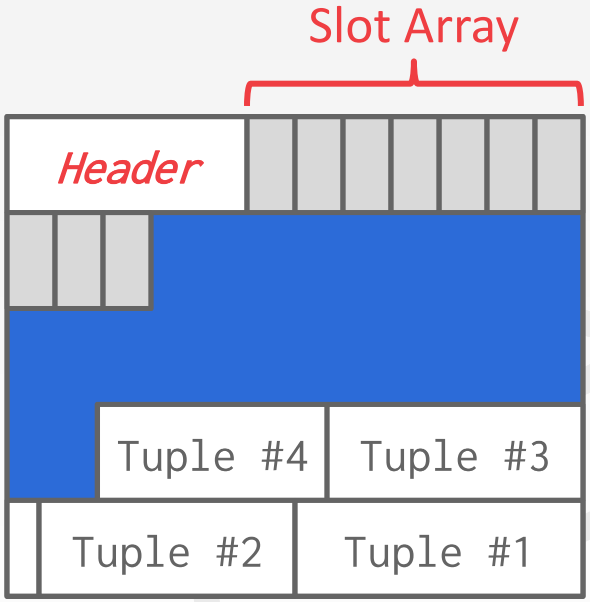


SLOTTED PAGES

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.

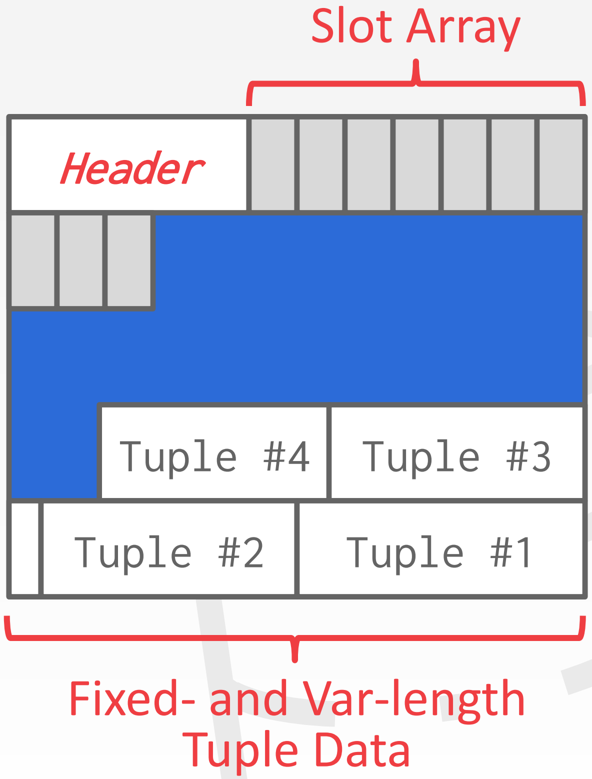


SLOTTED PAGES

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.



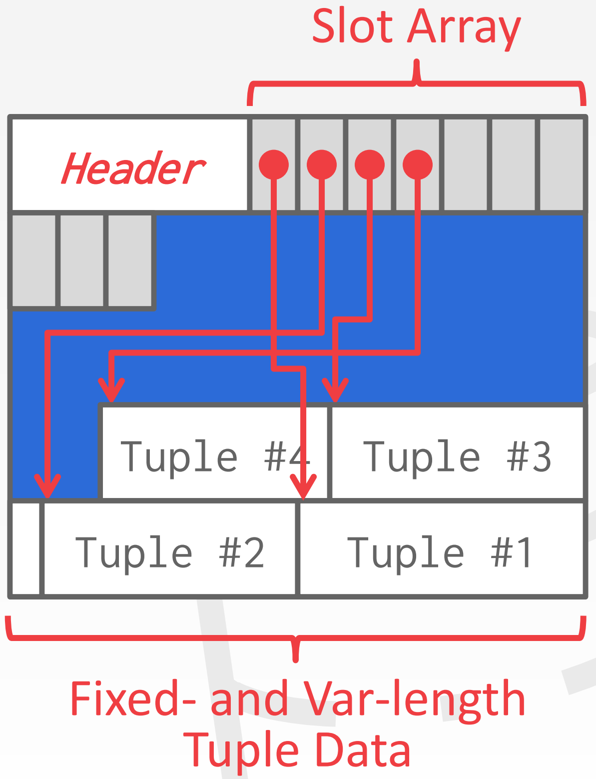
SLOTTED PAGES

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.

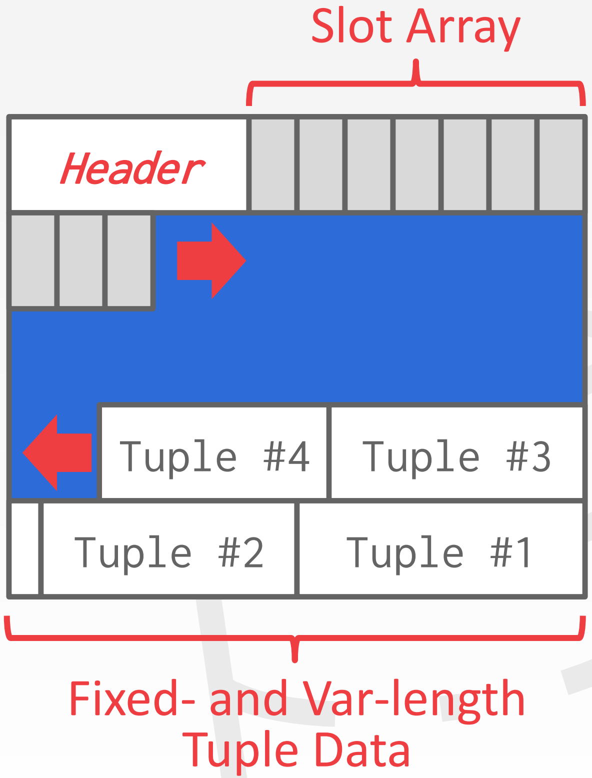


SLOTTED PAGES

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.

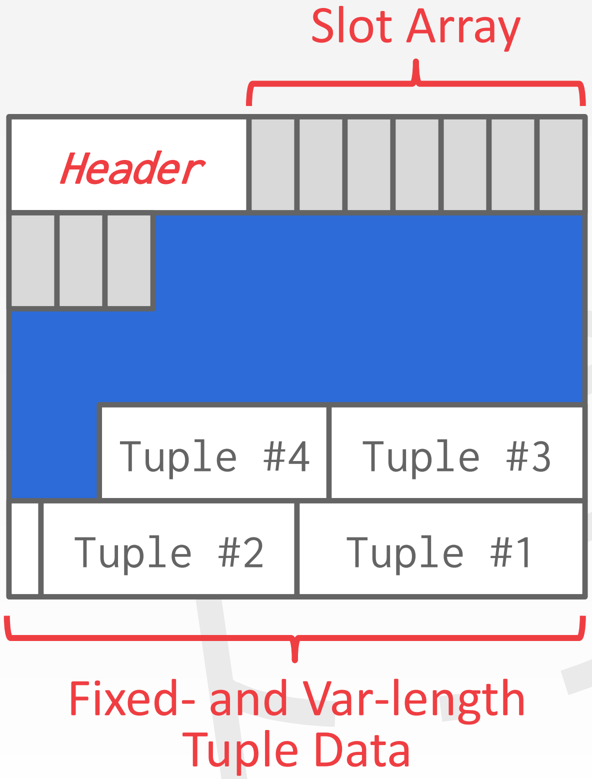


SLOTTED PAGES

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

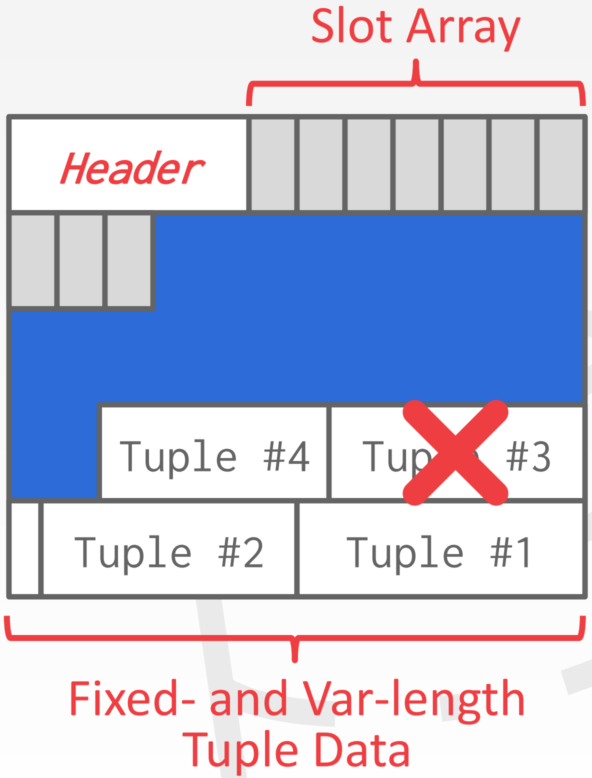
The most common layout scheme is called <u>slotted pages</u>.

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.

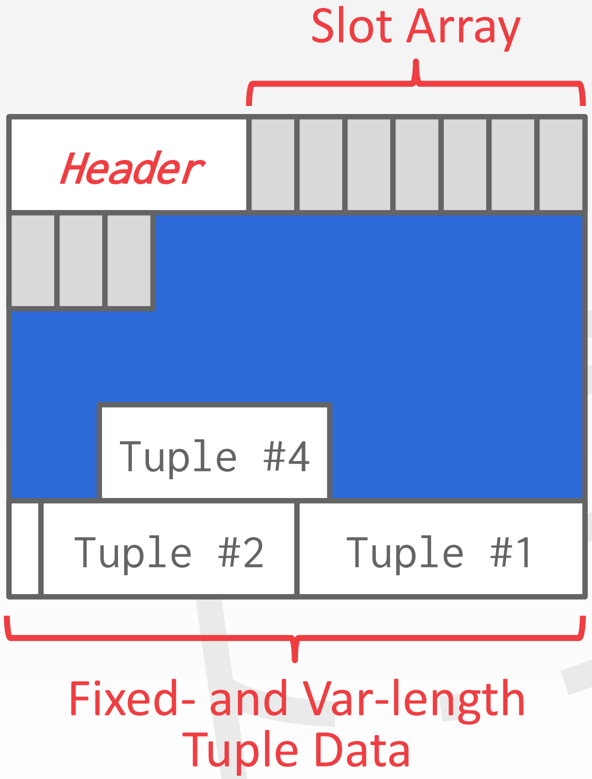


SLOTTED PAGES

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.

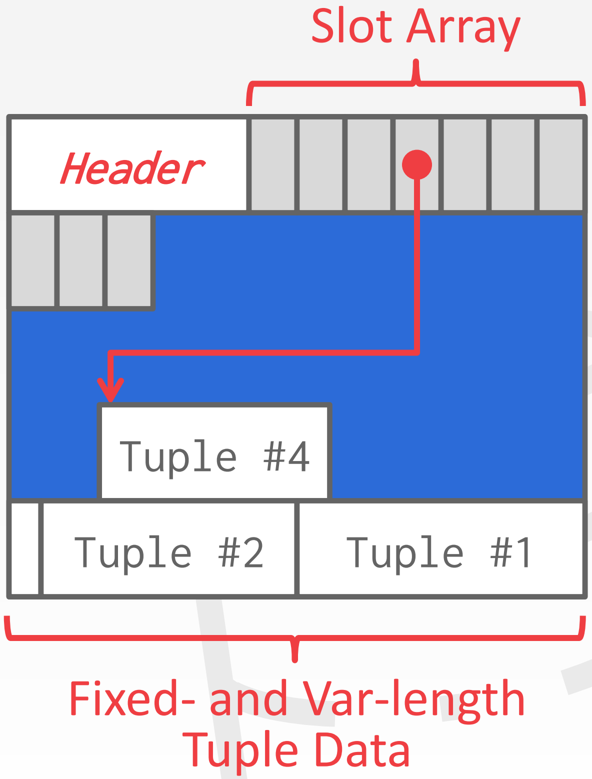


SLOTTED PAGES

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.

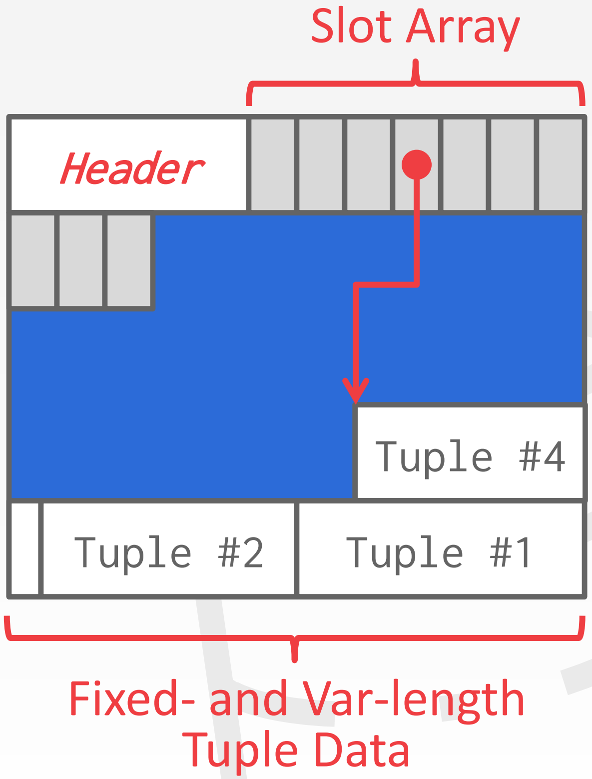


SLOTTED PAGES

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.



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.

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 (6-bytes)

SQLite

ROWID (8-bytes)

ORACLE

ROWID (10-bytes)

TODAY'S AGENDA

File Storage

Page Layout

Storage Models

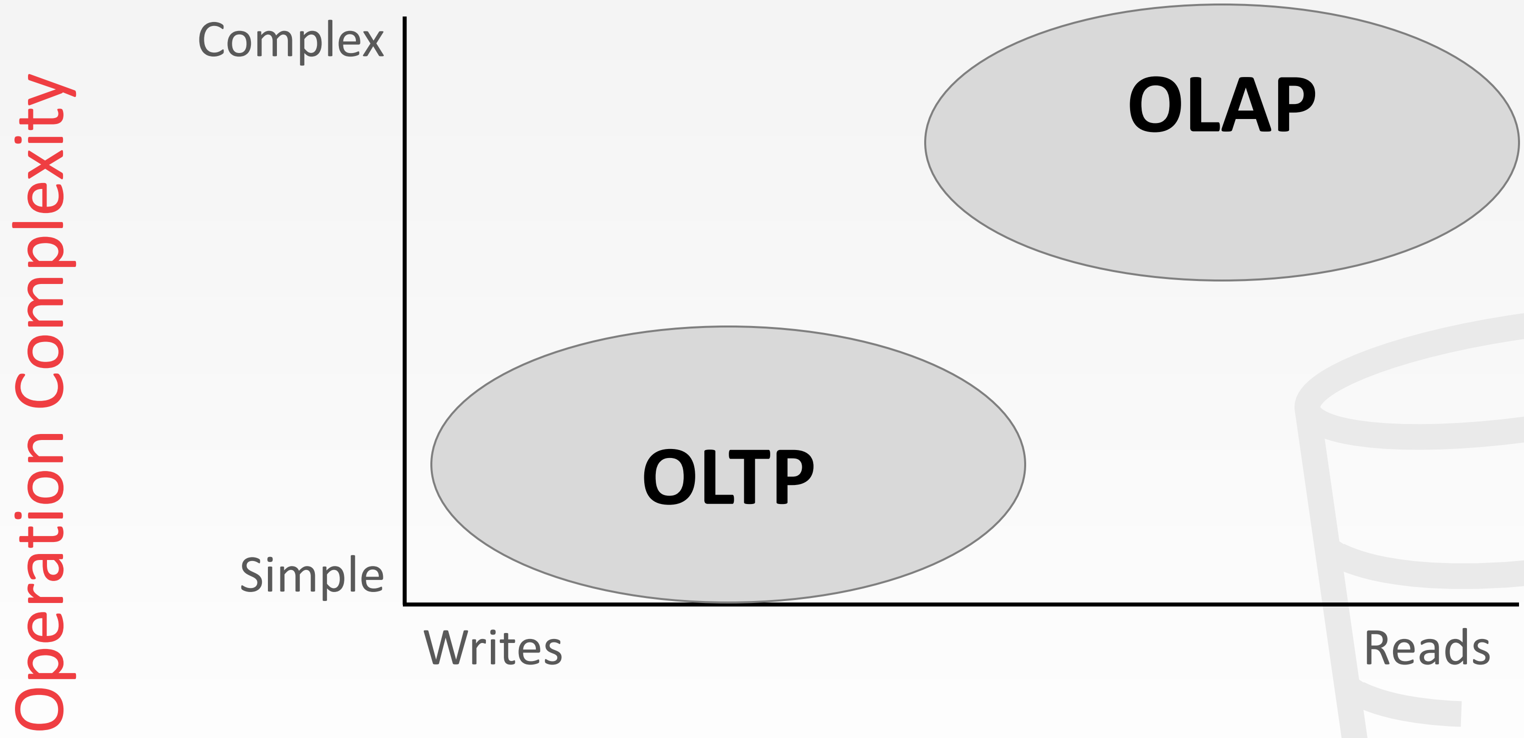
DATABASE WORKLOADS

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 instance

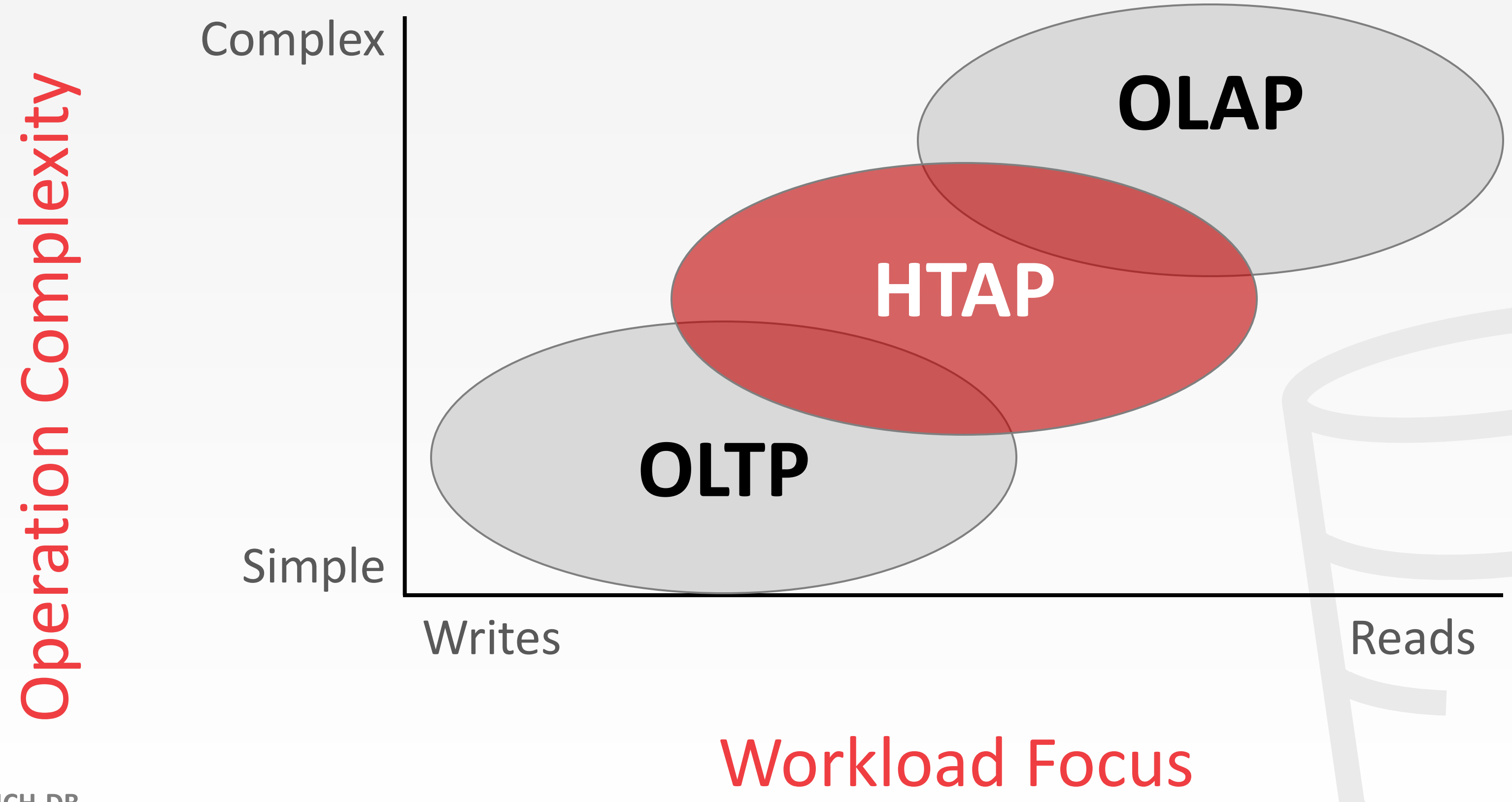
DATABASE WORKLOADS



Workload Focus

[SOURCE]

DATABASE WORKLOADS



[SOURCE]

OBSERVATION

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
);
```

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
);
```

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.

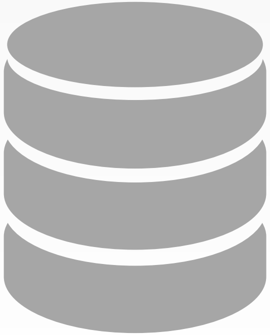
N-ARY STORAGE MODEL (NSM)

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)

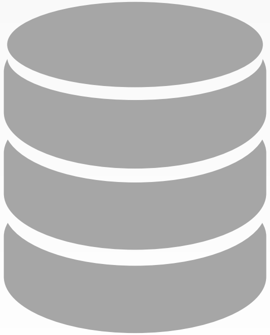
The DBMS stores all attributes for a single tuple contiguously in a page.



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

N-ARY STORAGE MODEL (NSM)

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

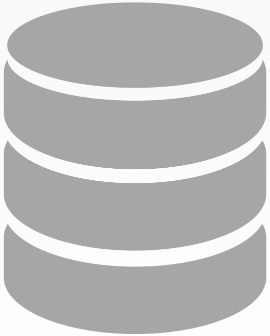


Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	-	-	-	-	-

←Tuple #1

N-ARY STORAGE MODEL (NSM)

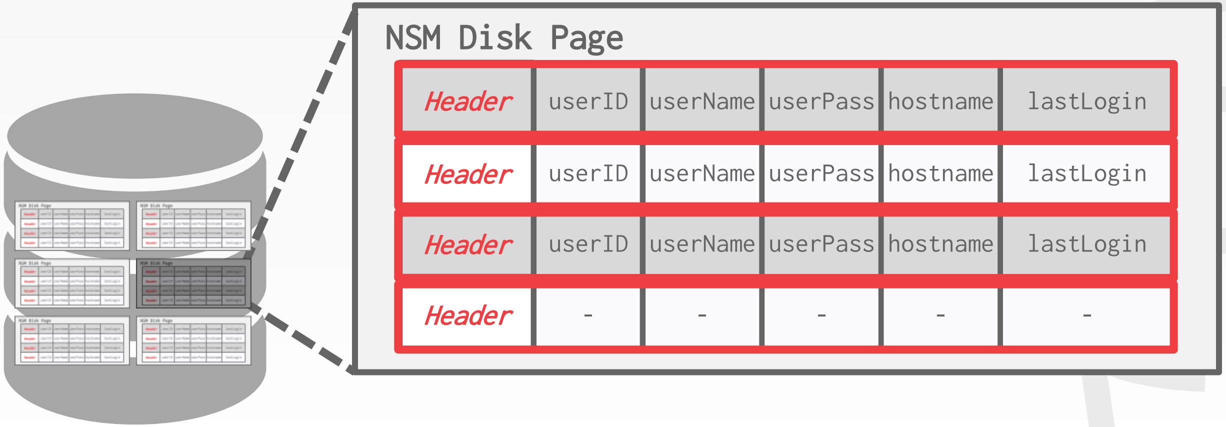
The DBMS stores all attributes for a single tuple contiguously in a page.



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.

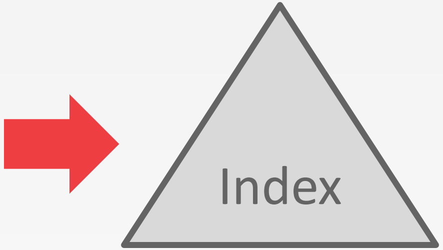


N-ARY STORAGE MODEL (NSM)

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

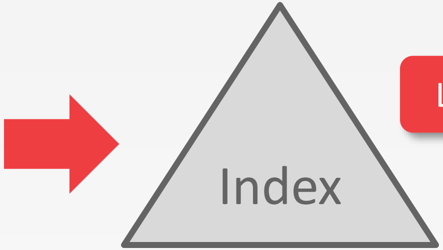
N-ARY STORAGE MODEL (NSM)

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



N-ARY STORAGE MODEL (NSM)

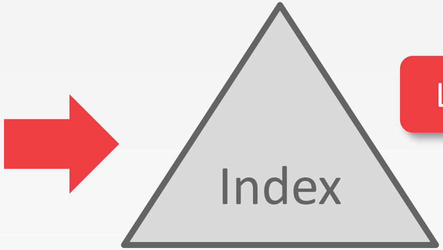
```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```



Lecture 16

N-ARY STORAGE MODEL (NSM)

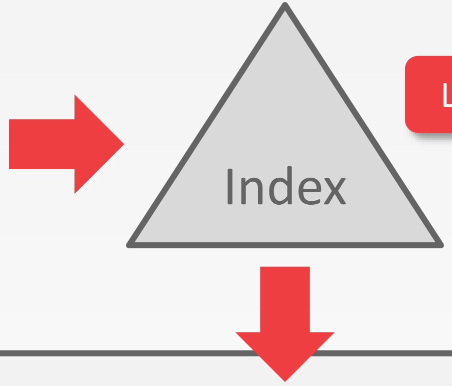
```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```



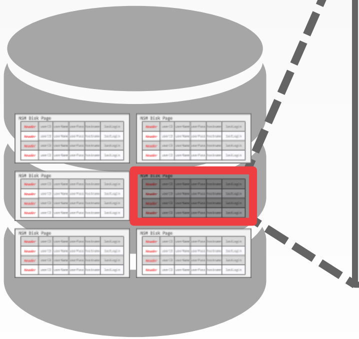
Lecture 16

N-ARY STORAGE MODEL (NSM)

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



Lecture 16



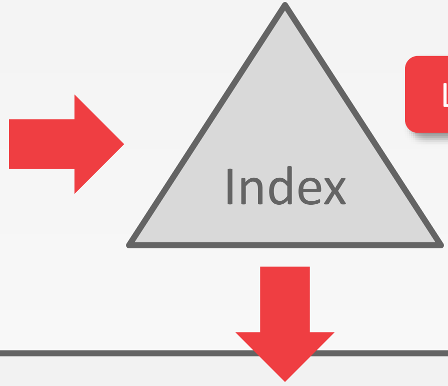
NSM Disk Page

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

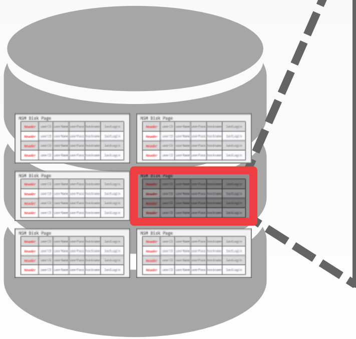
N-ARY STORAGE MODEL (NSM)

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

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



Lecture 16



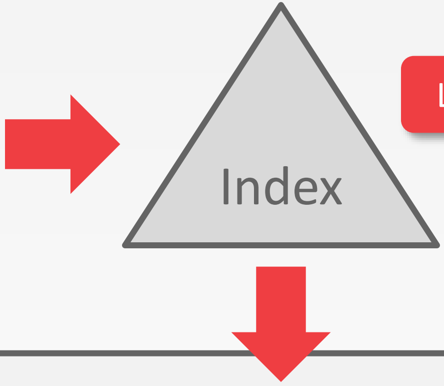
NSM Disk Page

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

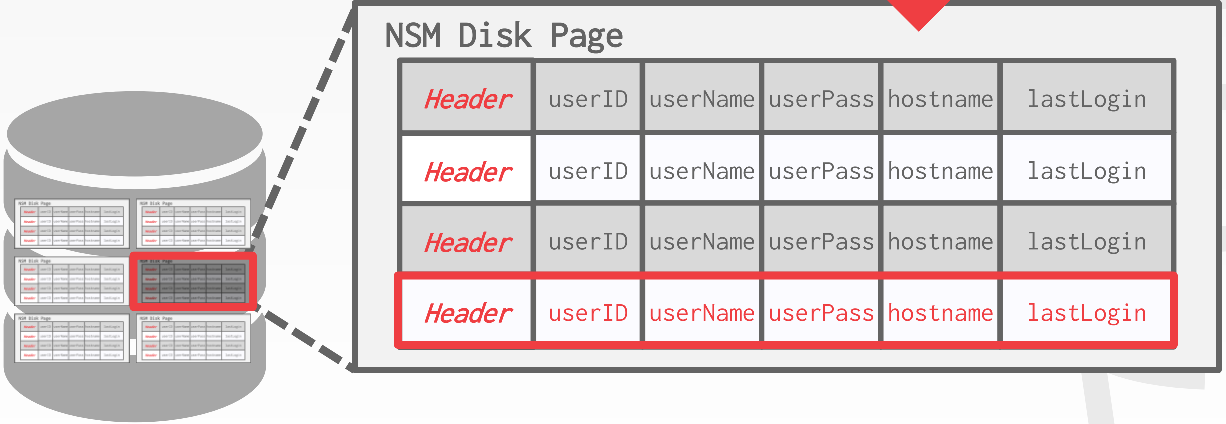
N-ARY STORAGE MODEL (NSM)

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

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



Lecture 16

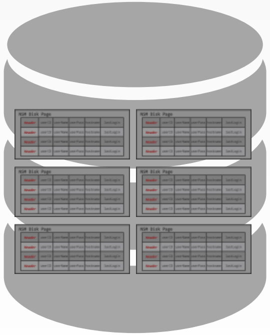


N-ARY STORAGE MODEL (NSM)

```
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)
```

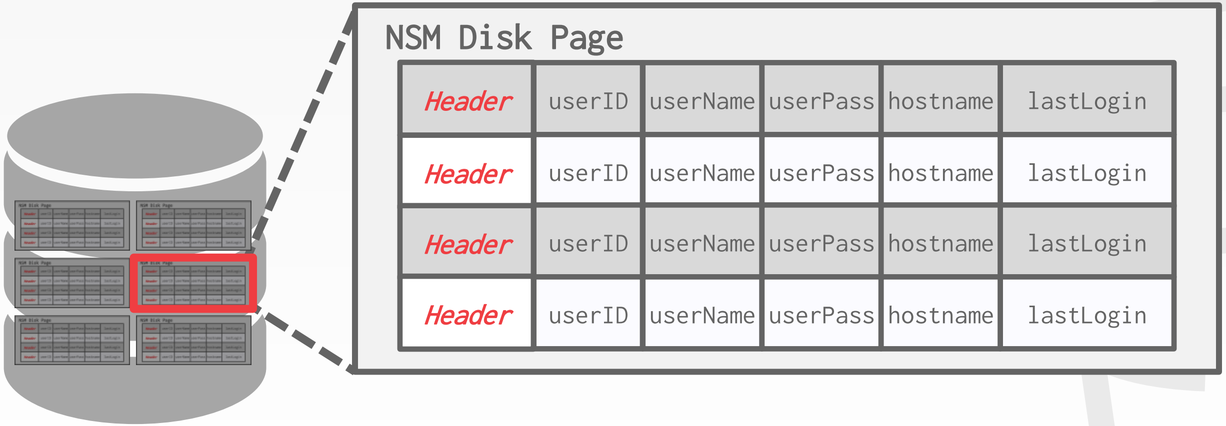
N-ARY STORAGE MODEL (NSM)

```
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)
```



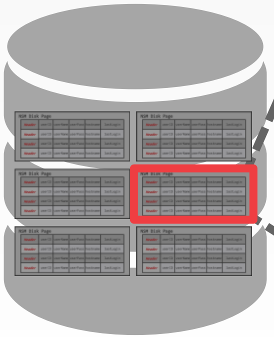
N-ARY STORAGE MODEL (NSM)

```
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)
```

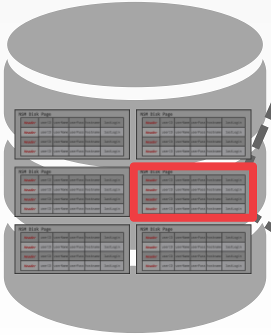


NSM Disk Page

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)
```

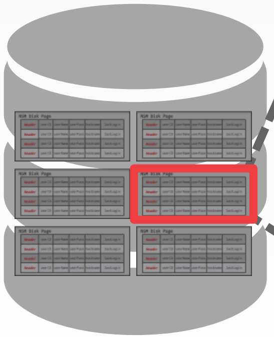


NSM Disk Page

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

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)
```

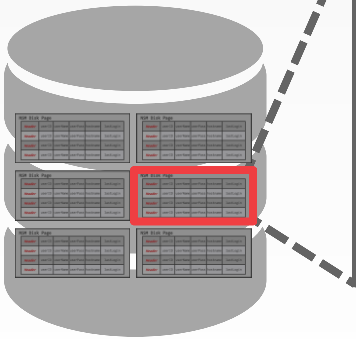


NSM Disk Page

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)
```

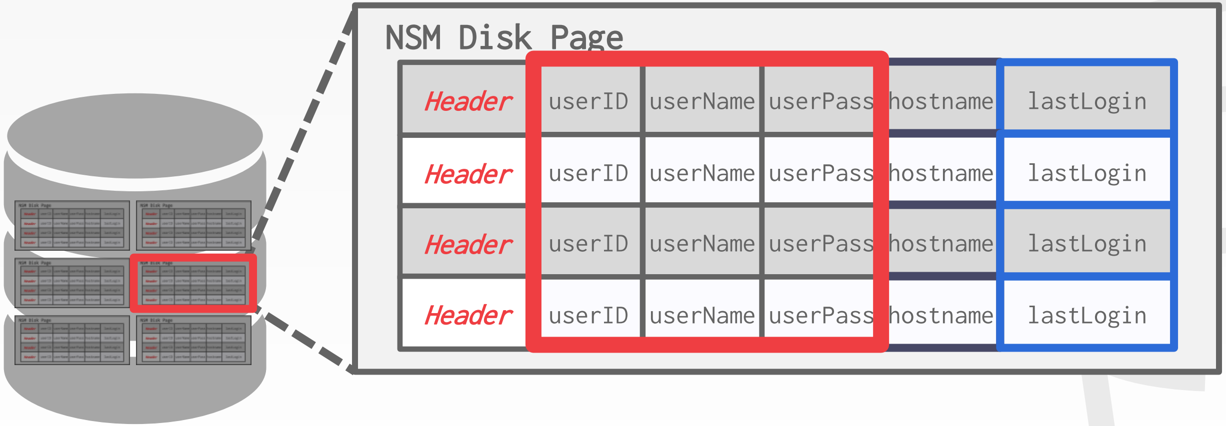


NSM Disk Page

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

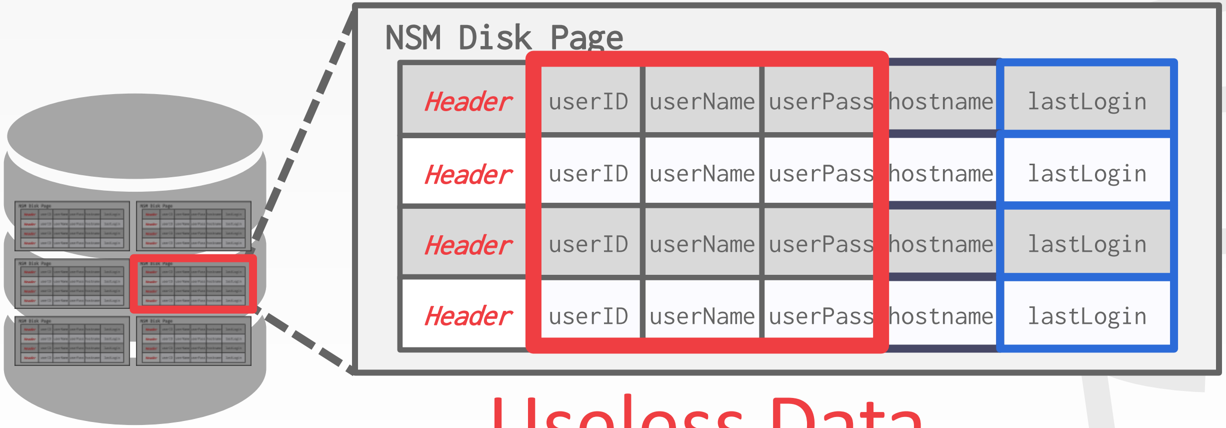
N-ARY STORAGE MODEL (NSM)

```
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)
```



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.

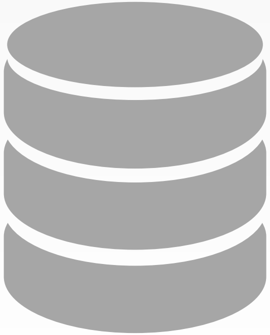
→ Also known as a "column store"

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".

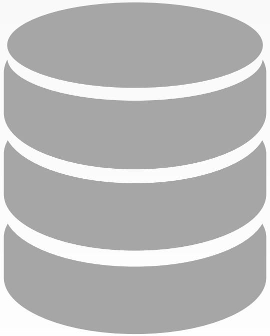


Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

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".

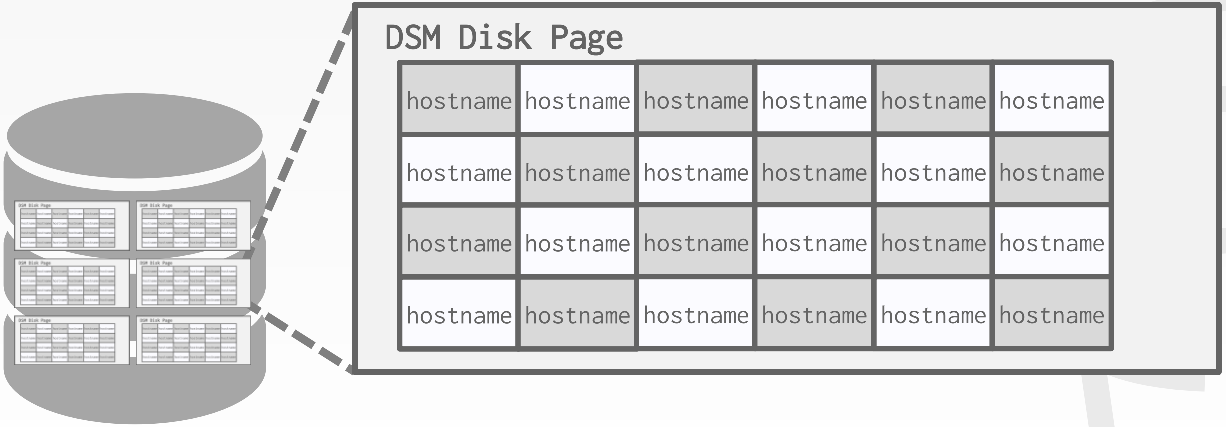


Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

DECOMPOSITION STORAGE MODEL (DSM)

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

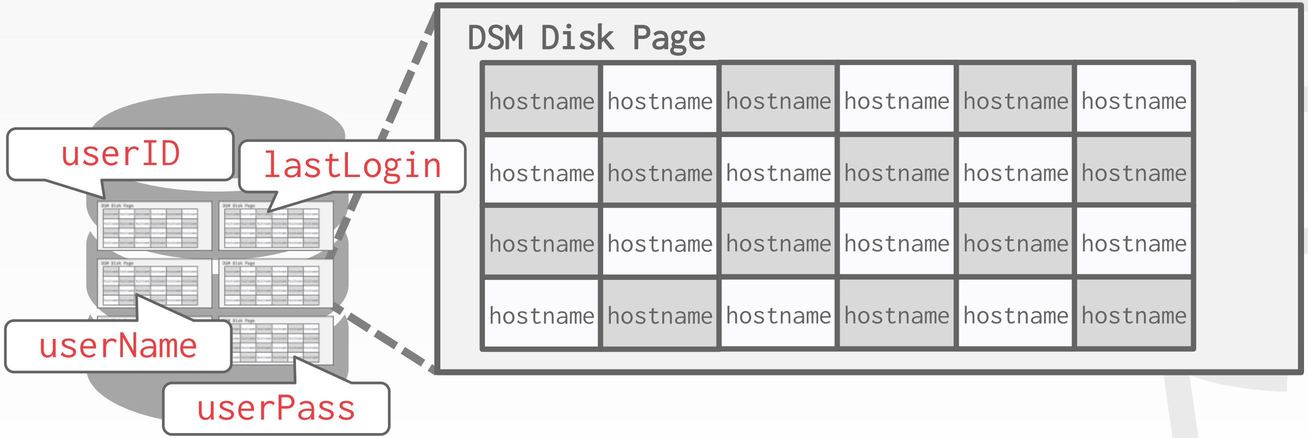
→ Also known as a "column store".



DECOMPOSITION STORAGE MODEL (DSM)

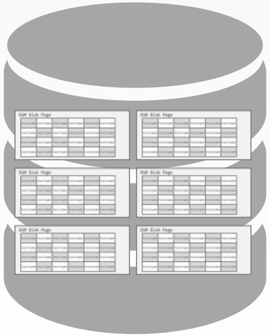
The DBMS stores the values of a single attribute for all tuples contiguously in a page.

→ Also known as a "column store".



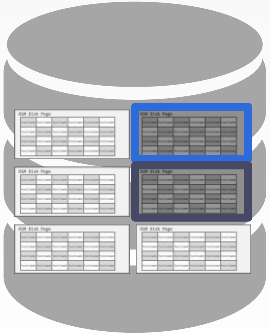
DECOMPOSITION STORAGE MODEL (DSM)

```
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)
```



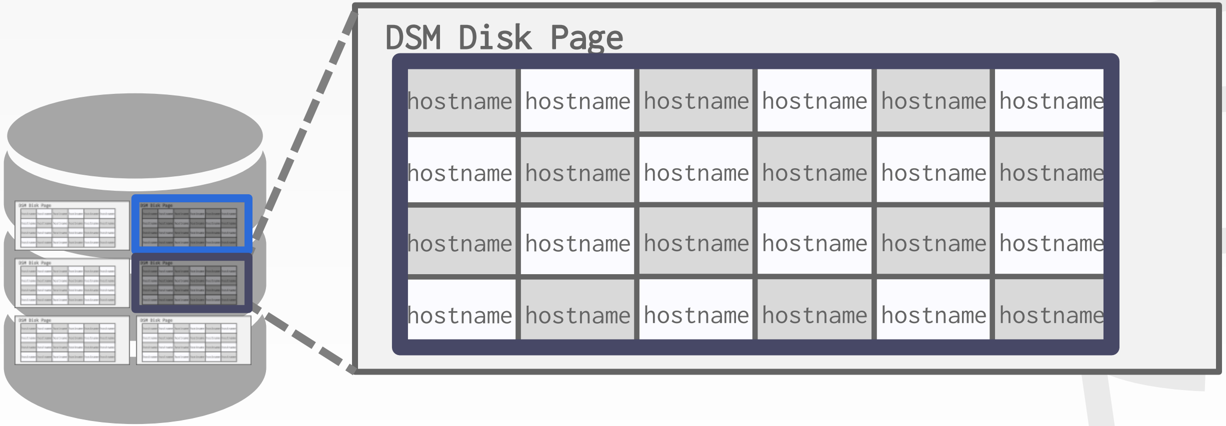
DECOMPOSITION STORAGE MODEL (DSM)

```
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)
```



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)
```



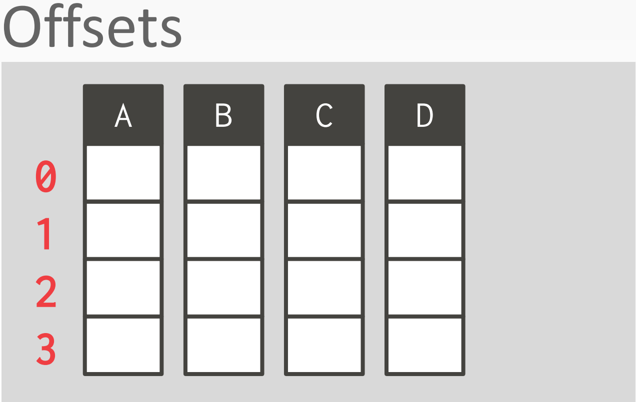
TUPLE IDENTIFICATION

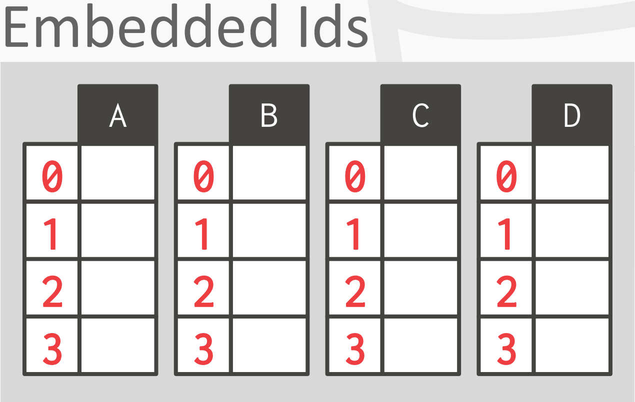
Choice #1: Fixed-length Offsets

→ 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.





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.

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

DSM SYSTEM HISTORY

1970s: Cantor DBMS

1980s: DSM Proposal

1990s: SybaseIQ (in-memory only)

2000s: Vertica, VectorWise, MonetDB

2010s: Everyone

DSM SYSTEM HISTORY

1970s: Cantor DBMS

1980s: DSM Proposal

1990s: SybaseIQ (in-memory only)

2000s: Vertica, VectorWise, MonetDB

2010s: Everyone

DSM SYSTEM HISTORY

1970s: Cantor DBMS

1980s: DSM Proposal

1990s: SybaseIQ (in-memory only)

2000s: Vertica, VectorWise, MonetDB

2010s: Everyone

CONCLUSION

Database is organized in pages.

Different ways to track pages.

Different ways to store pages.

It is important to choose the right storage model for the target workload:

→ OLTP = Row Store

→ OLAP = Column Store

NEXT CLASS

Hash Tables