Carnegie Mellon University

03 Database Storage – Part I

Intro to Database Systems
15-445/15-645
Fall 2021

AC
Andrew Crotty
Computer Science
Carnegie Mellon University

ADMINISTRIVIA

Homework #1 is due September 12th @ 11:59pm

Project #0 is due September 12th @ 11:59pm

Project #1 will be released on September 13th

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).

COURSE OUTLINE

Relational Databases

Storage

Execution

Concurrency Control

Recovery

Distributed Databases

Potpourri

- Query Planning
- Operator Execution
- Access Methods
- Buffer Pool Manager
- Disk Manager

COURSE OUTLINE

Relational Databases

Storage

Execution

Concurrency Control

Recovery

Distributed Databases

Potpourri

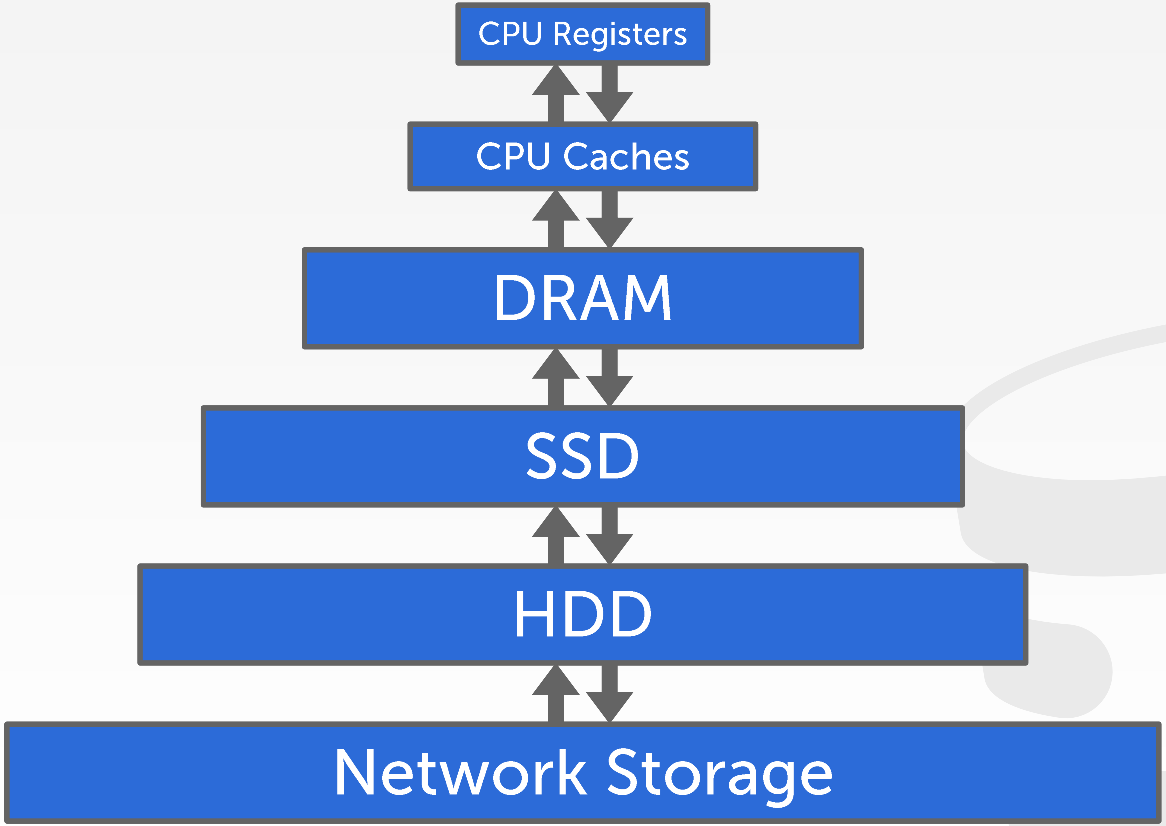
- Query Planning
- Operator Execution
- Access Methods
- Buffer Pool Manager
- Disk Manager

DISK-BASED ARCHITECTURE

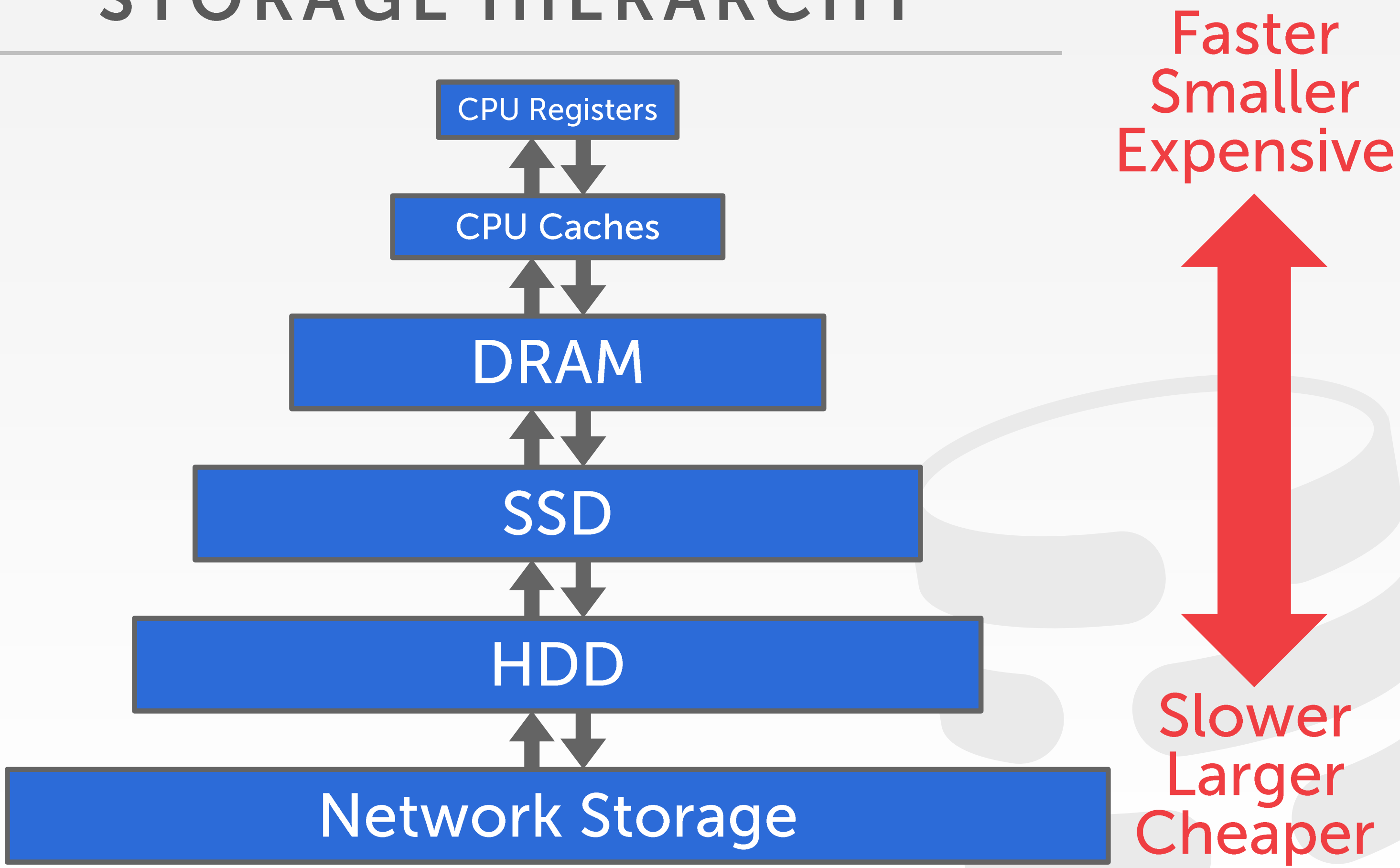
The DBMS assumes that the primary storage location of the database is on non-volatile disk.

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

STORAGE HIERARCHY

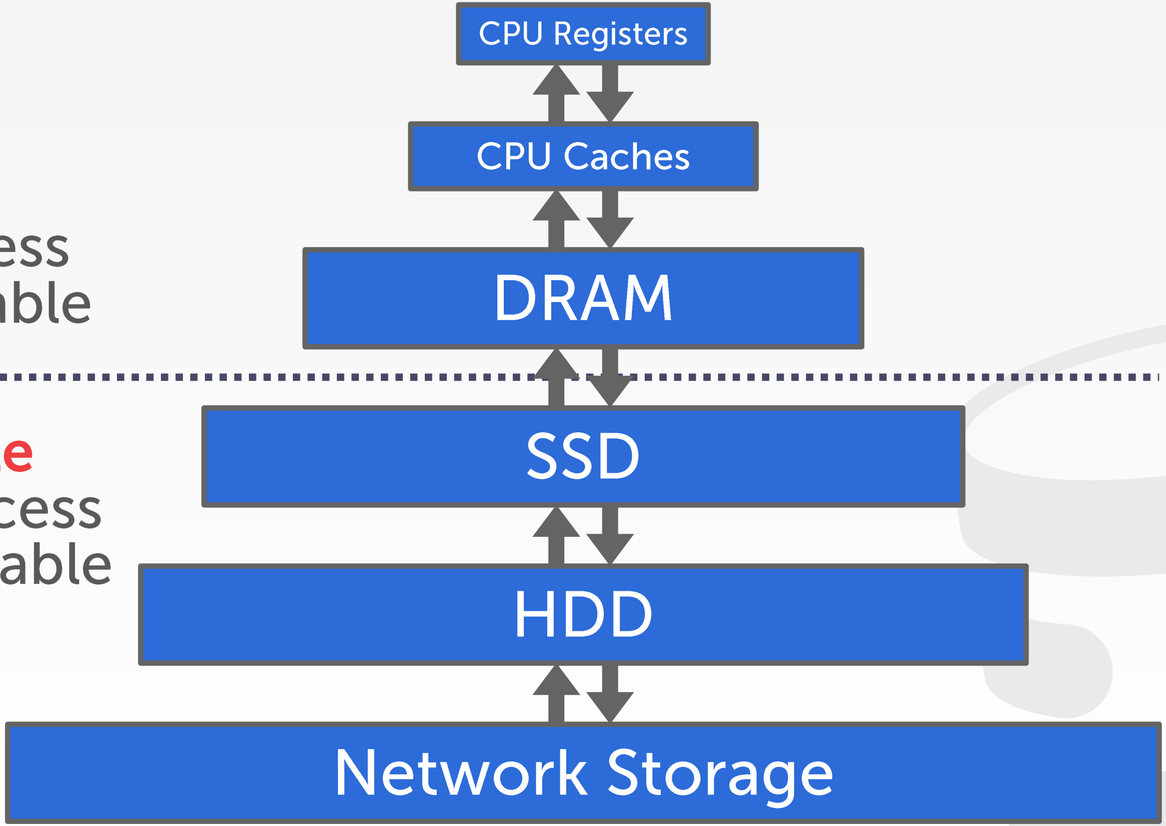


STORAGE HIERARCHY



STORAGE HIERARCHY

Volatile
Random Access
Byte-Addressable

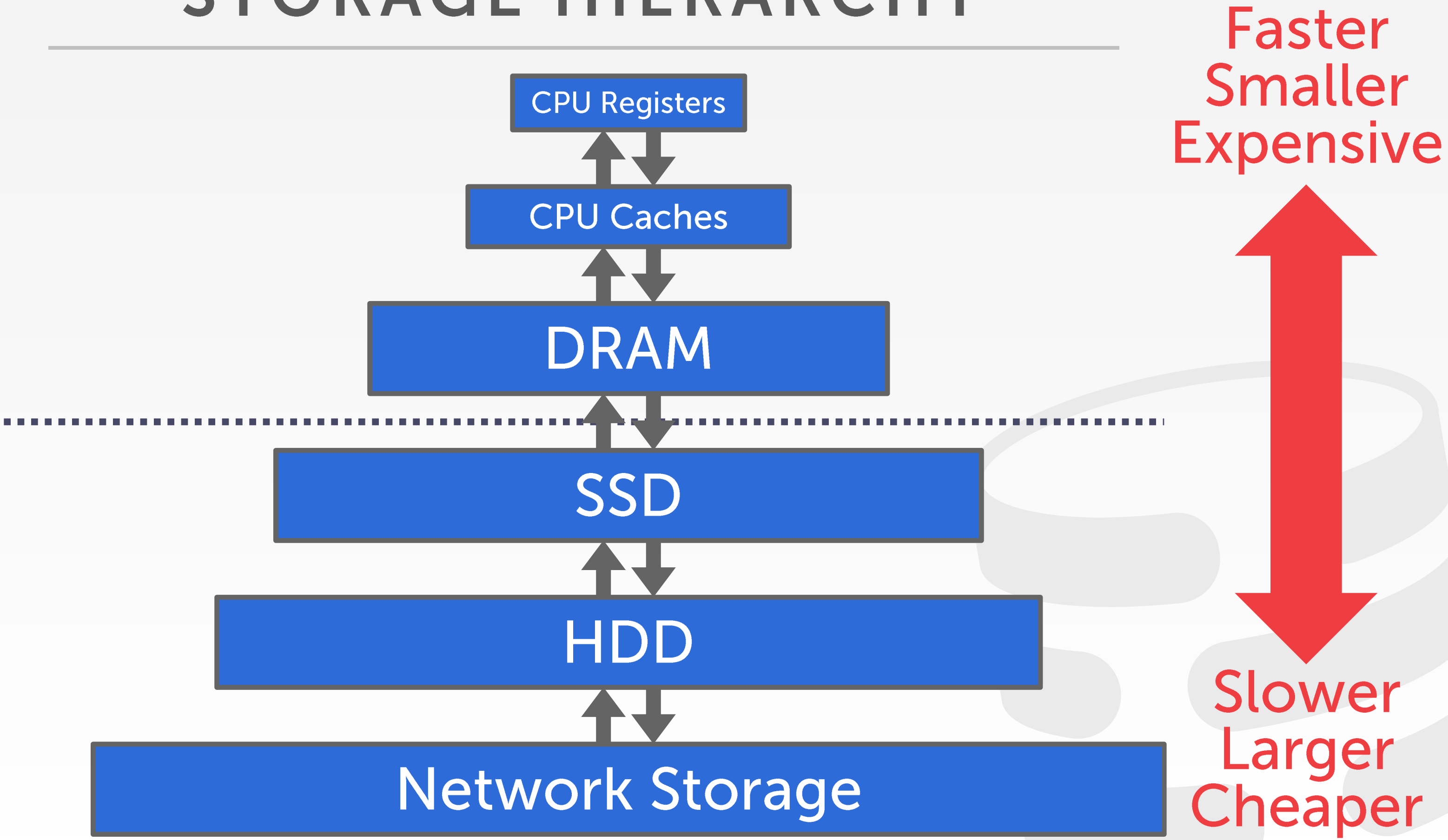


Non-Volatile
Sequential Access
Block-Addressable

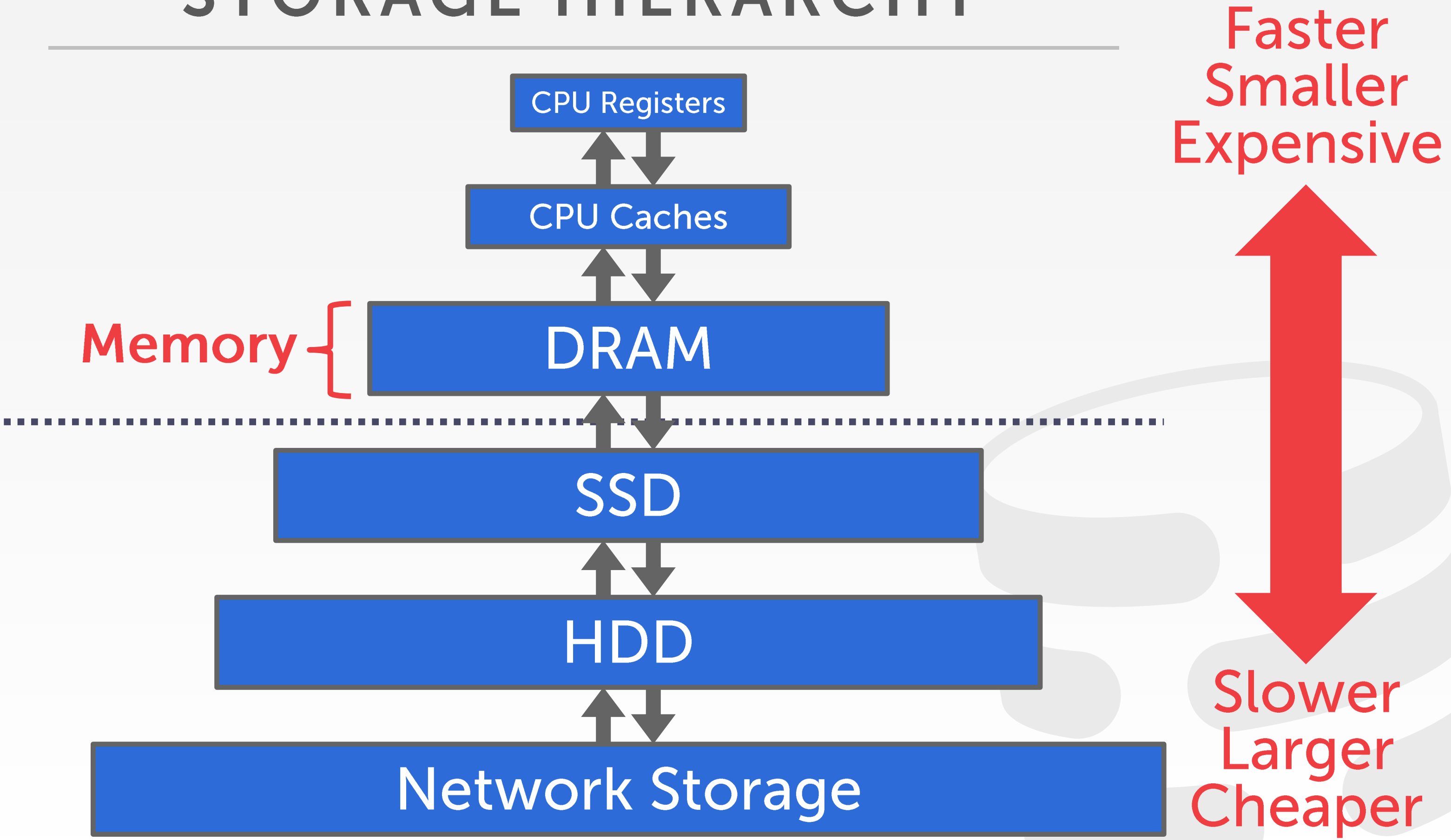
Faster
Smaller
Expensive

Slower
Larger
Cheaper

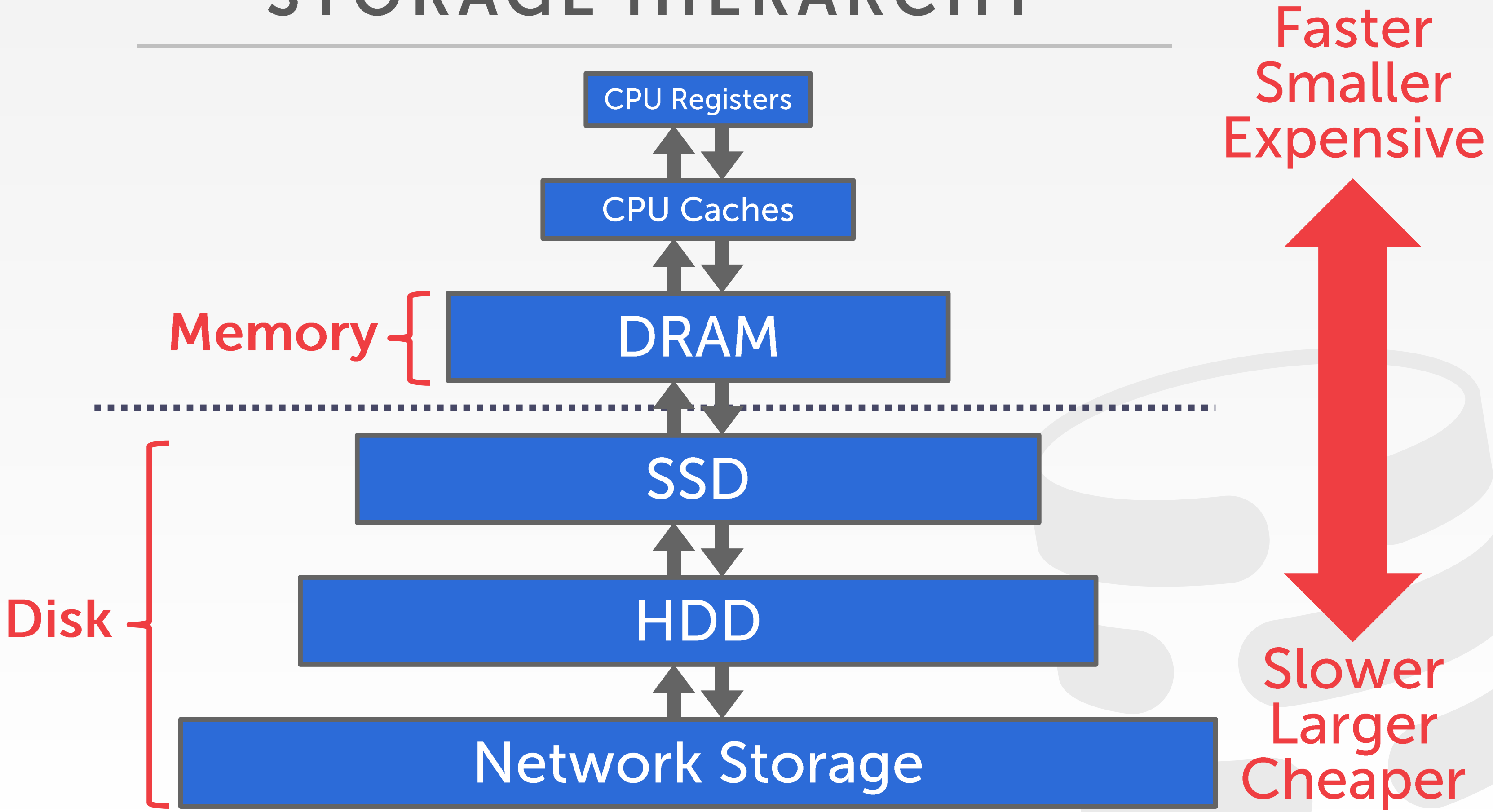
STORAGE HIERARCHY



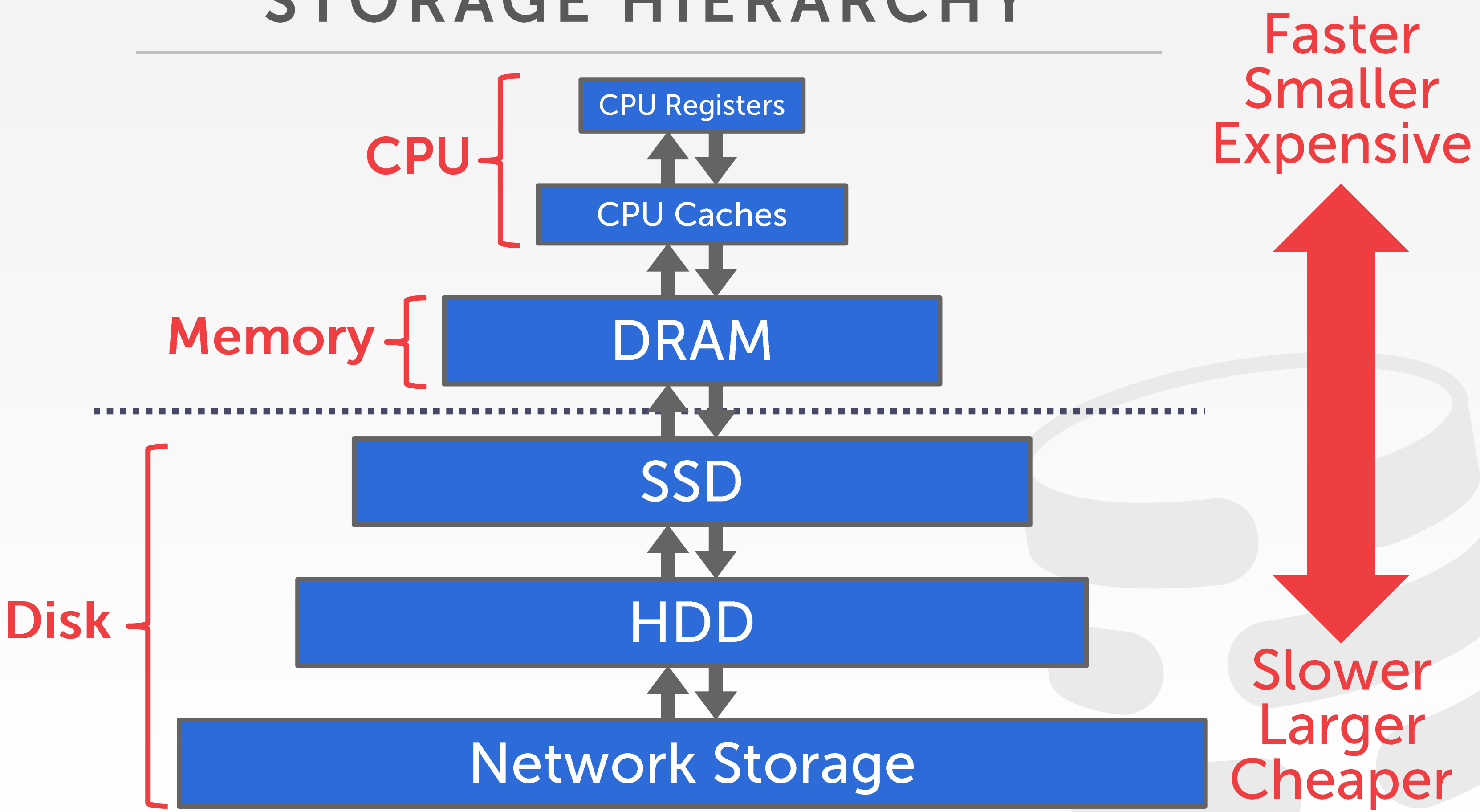
STORAGE HIERARCHY



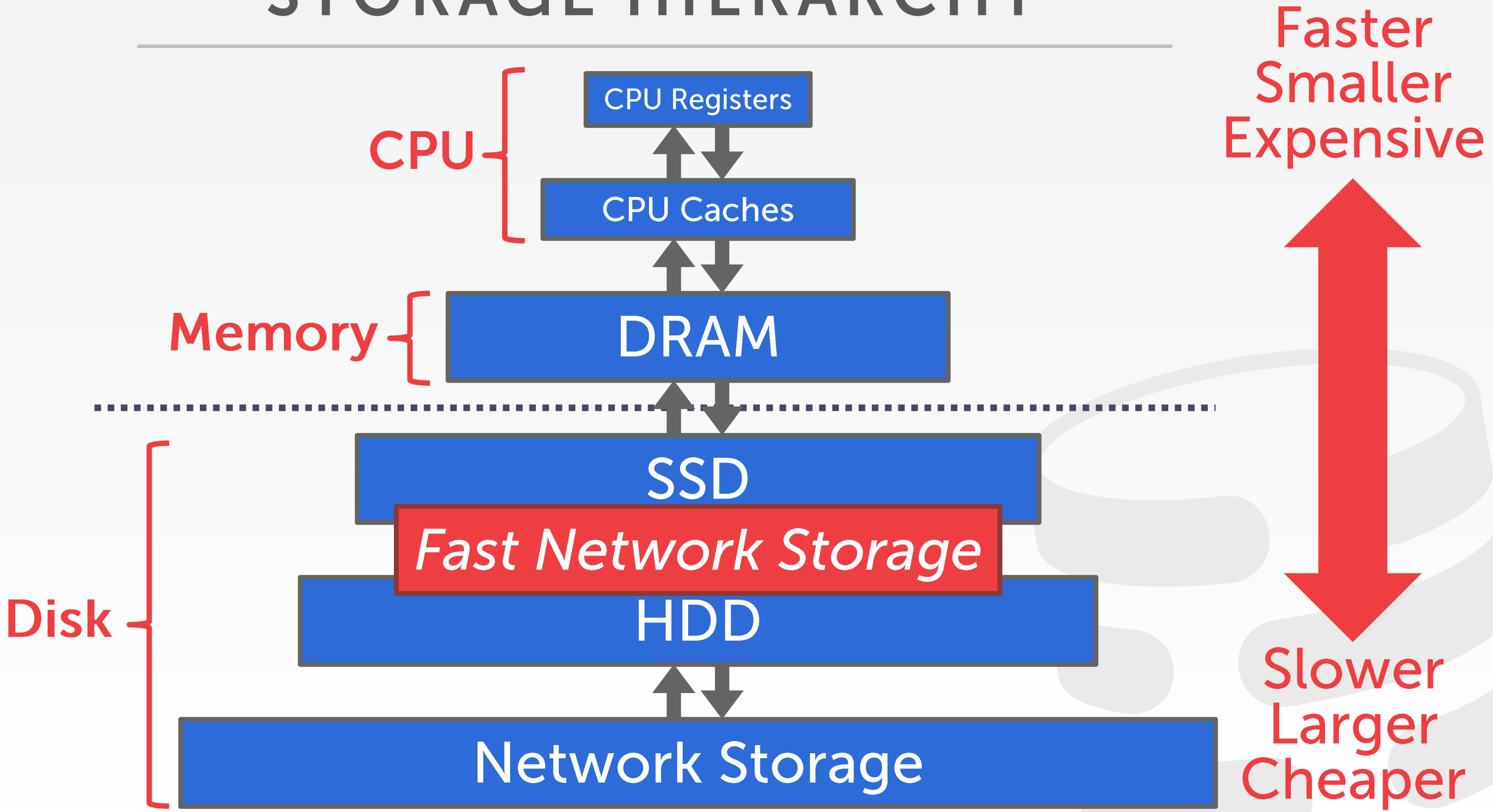
STORAGE HIERARCHY



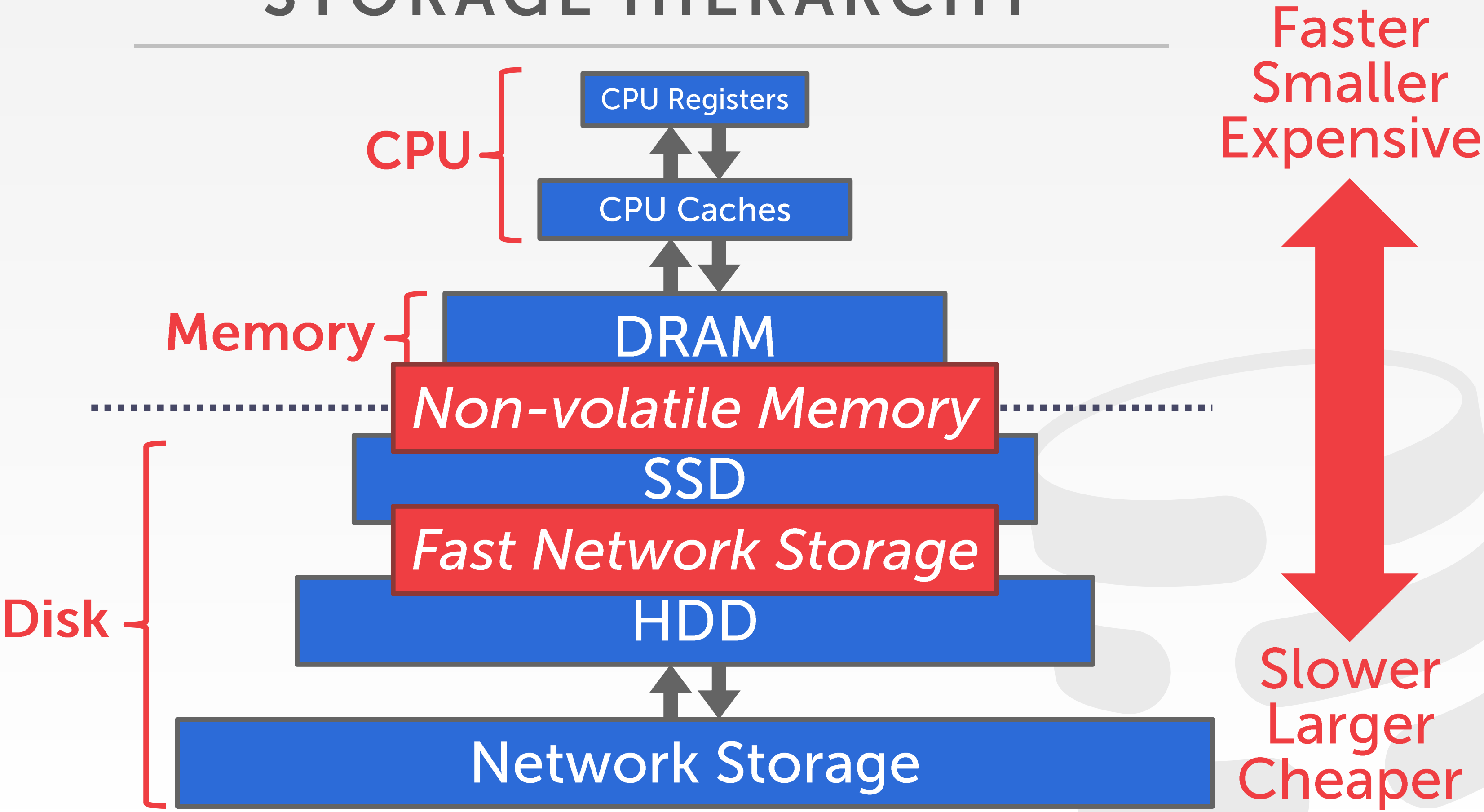
STORAGE HIERARCHY



STORAGE HIERARCHY



STORAGE HIERARCHY



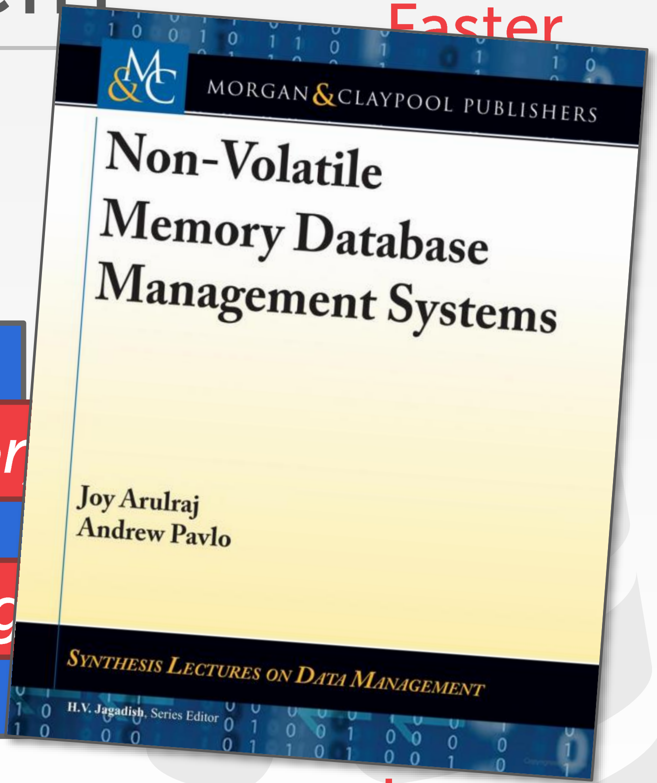
STORAGE HIERARCHY

CPU
- CPU Registers
- CPU Caches

Memory
- DRAM
- *Non-volatile Memory*

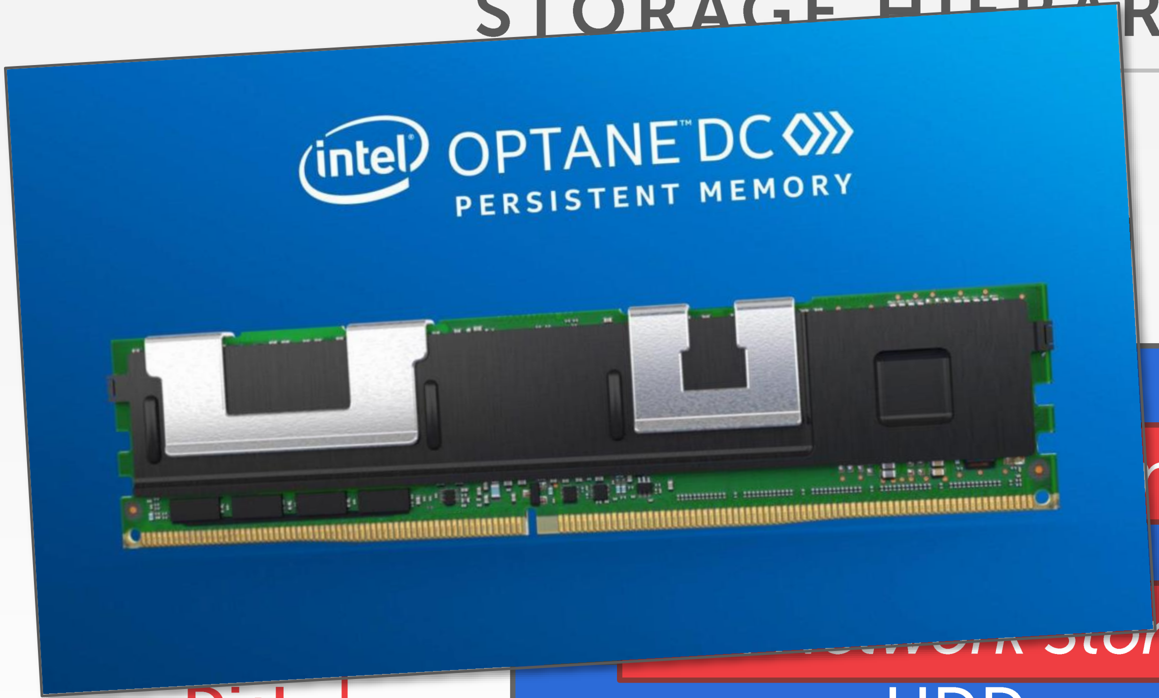
Disk
- SSD
- *Fast Network Storage*
- HDD
- Network Storage

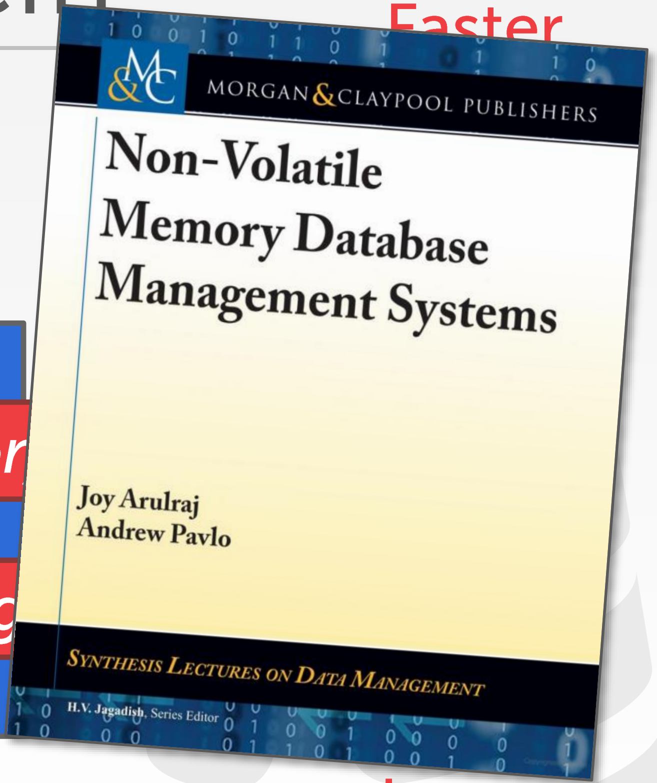
Faster



Larger
Cheaper

STORAGE HIERARCHY





Faster

Disk

HDD

Network Storage

Larger
Cheaper

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.

→ Allocating multiple pages at the same time is called an extent.

SYSTEM DESIGN GOALS

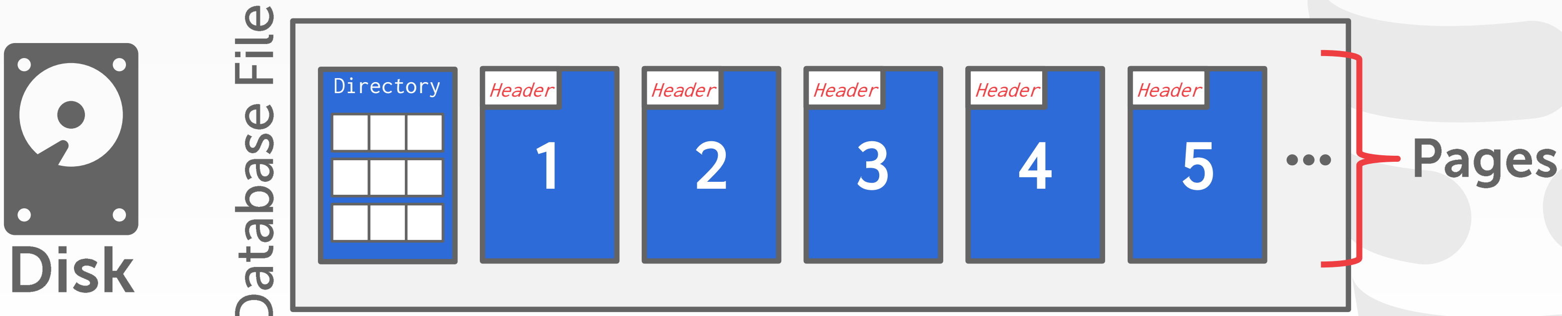
Allow the DBMS to manage databases that exceed the amount of memory available.

Reading/writing to disk is expensive, so it must be managed carefully to avoid large stalls and performance degradation.

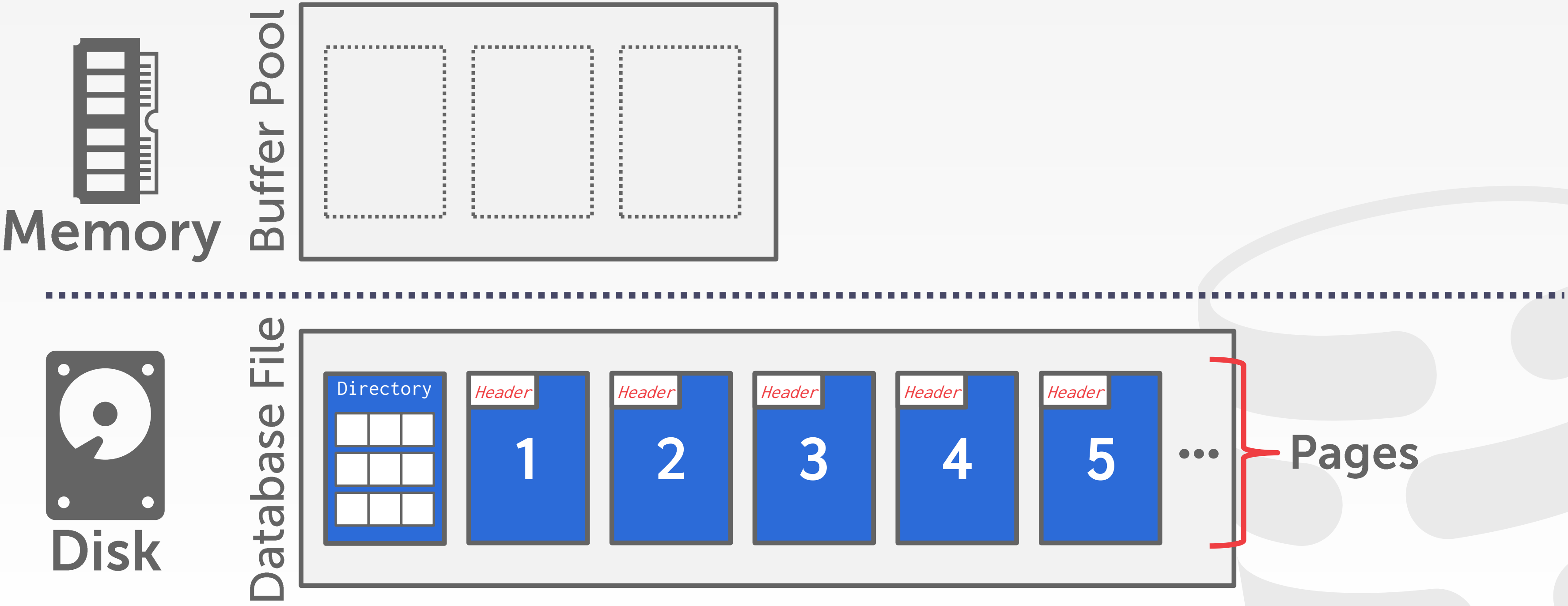
Random access on disk is usually much slower than sequential access, so the DBMS will want to maximize sequential access.

DISK-ORIENTED DBMS

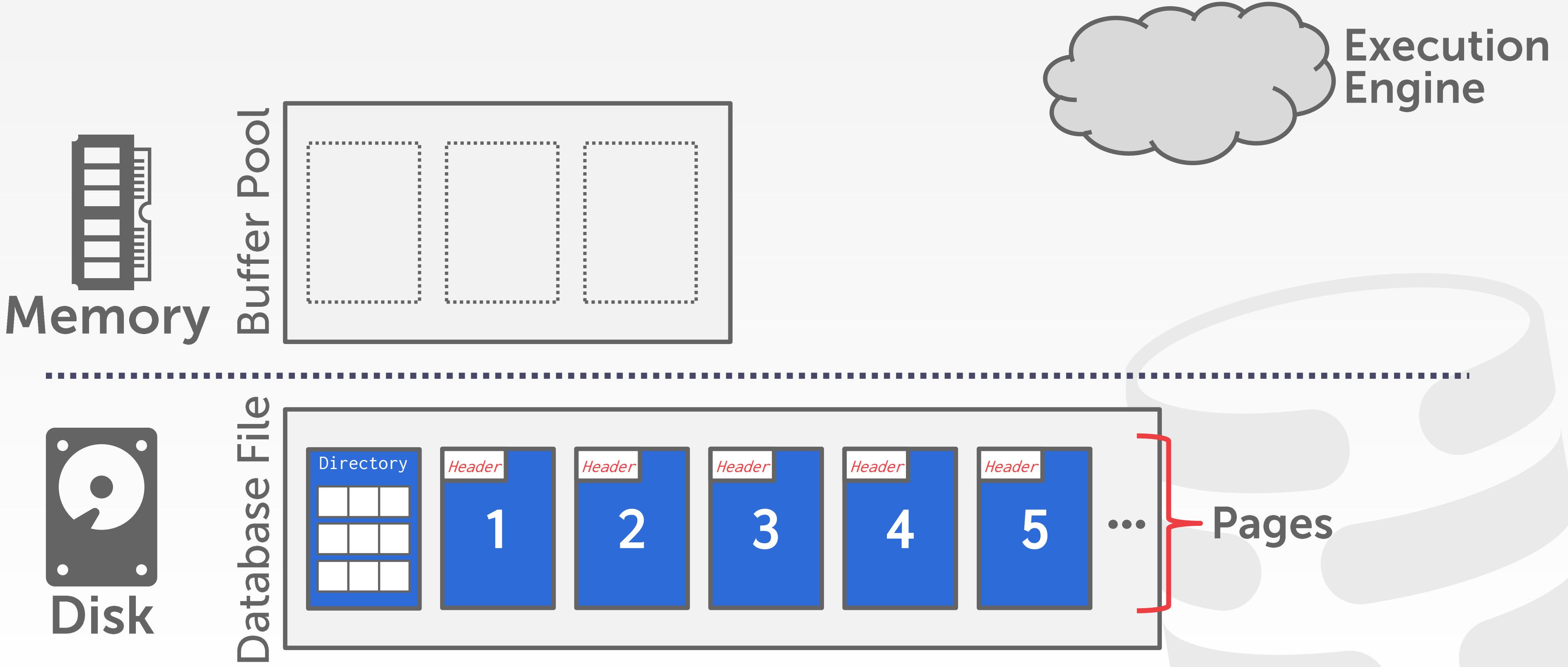
DISK-ORIENTED DBMS



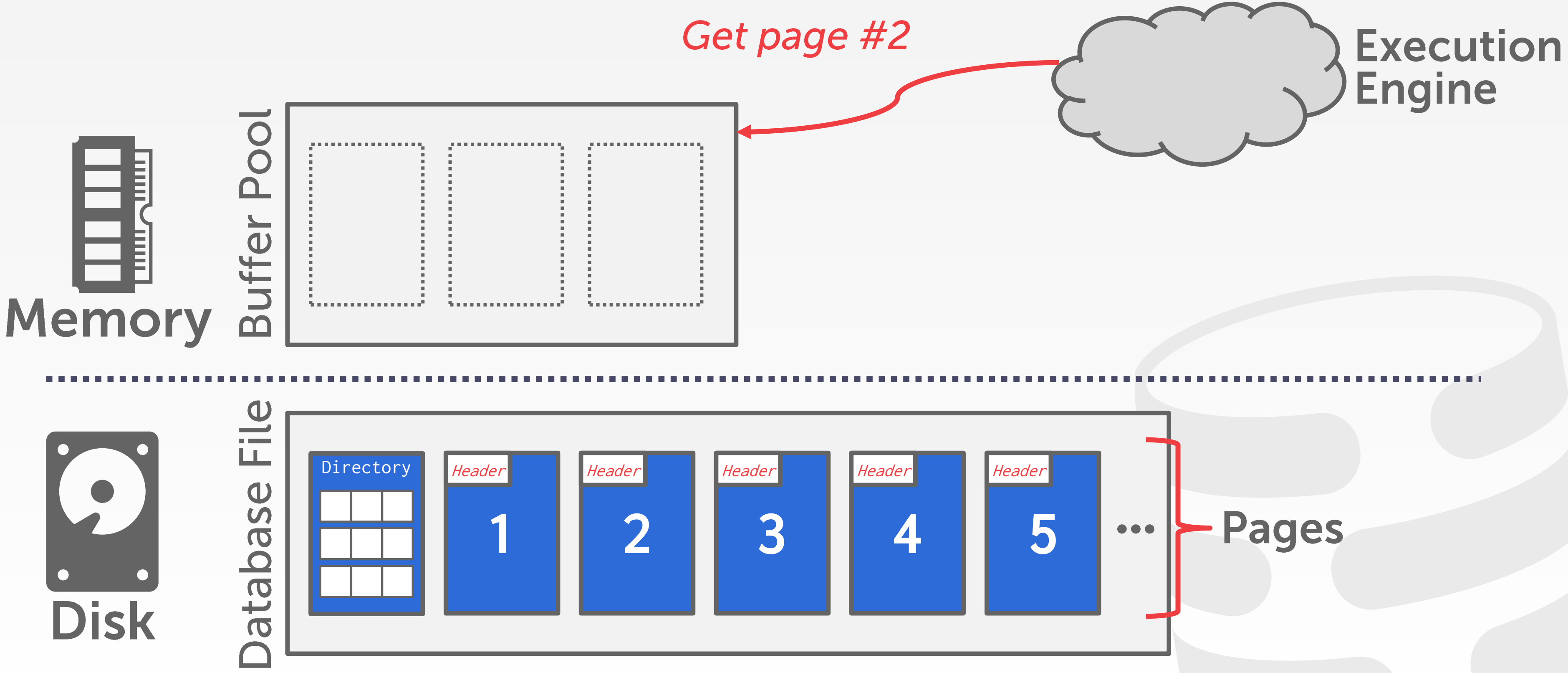
DISK-ORIENTED DBMS



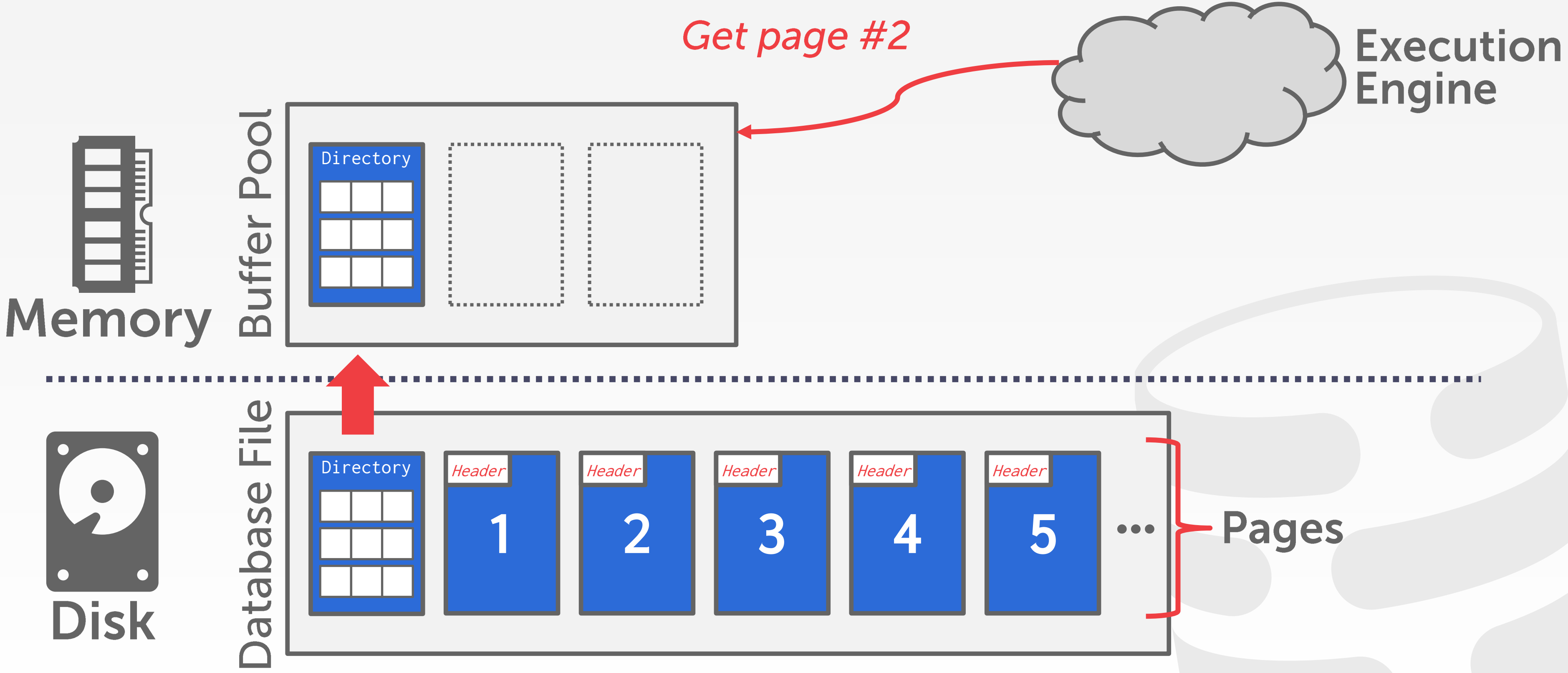
DISK-ORIENTED DBMS



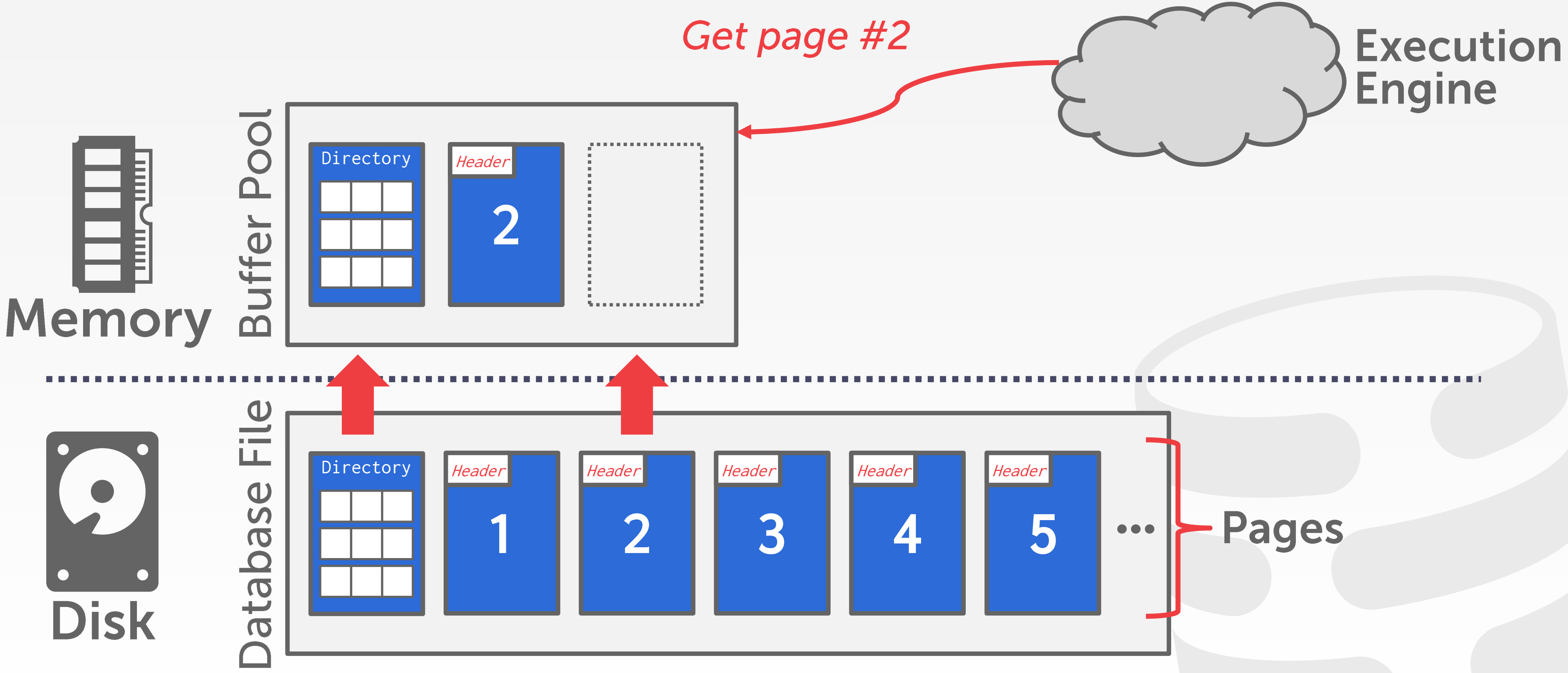
DISK-ORIENTED DBMS



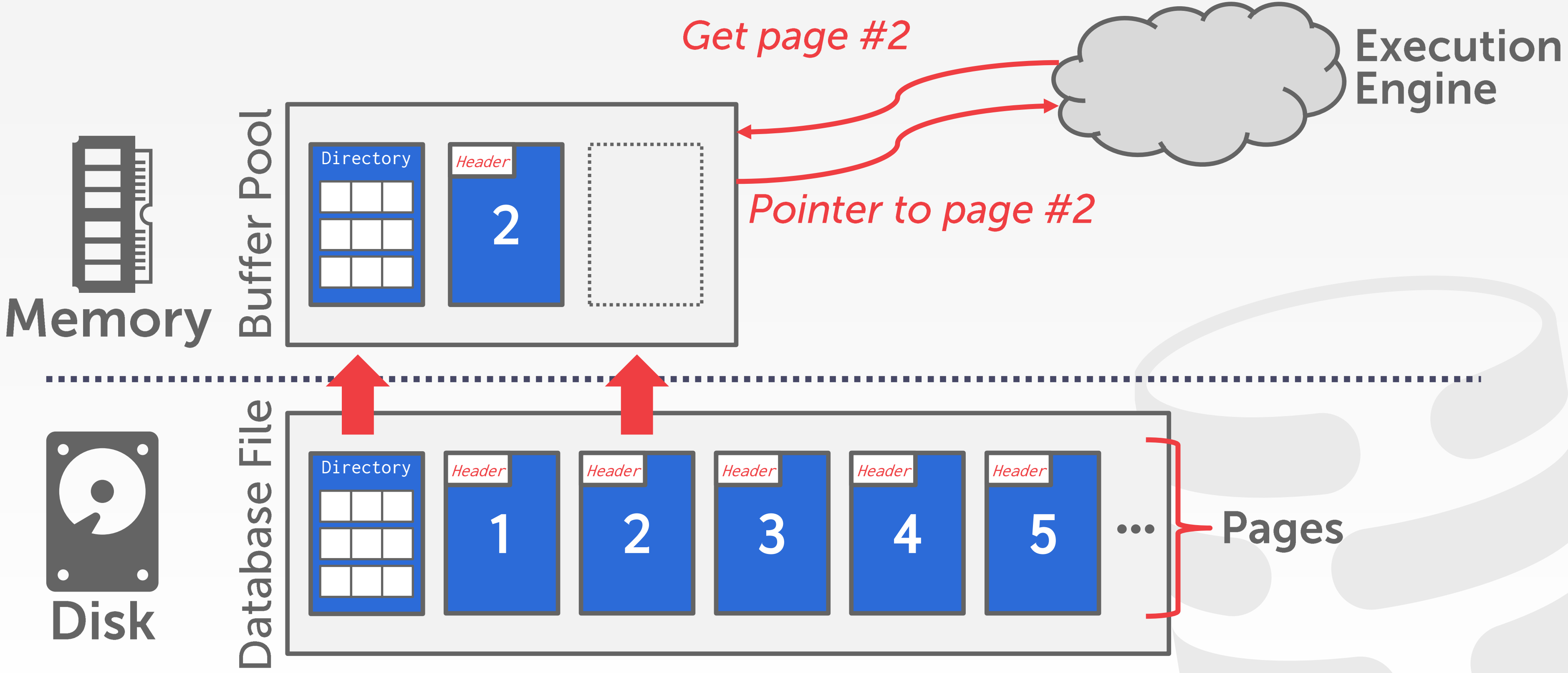
DISK-ORIENTED DBMS



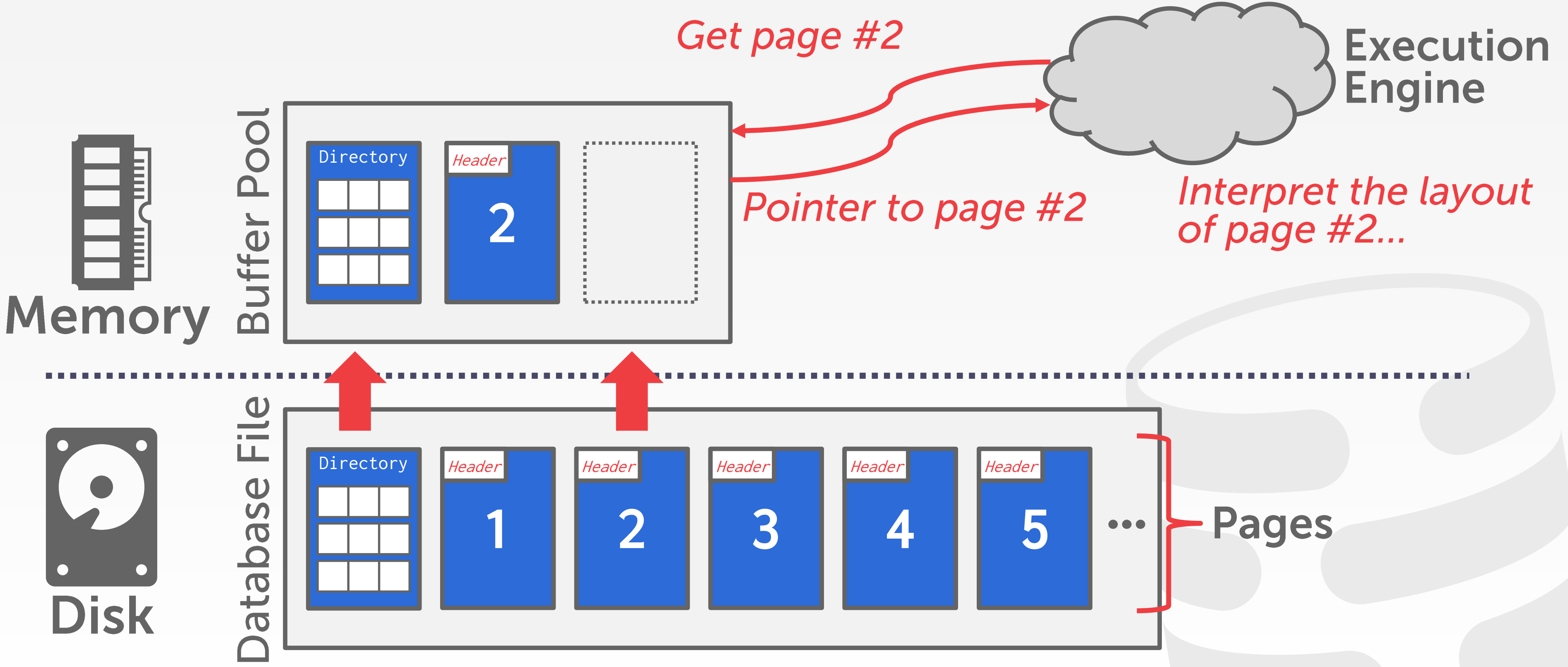
DISK-ORIENTED DBMS



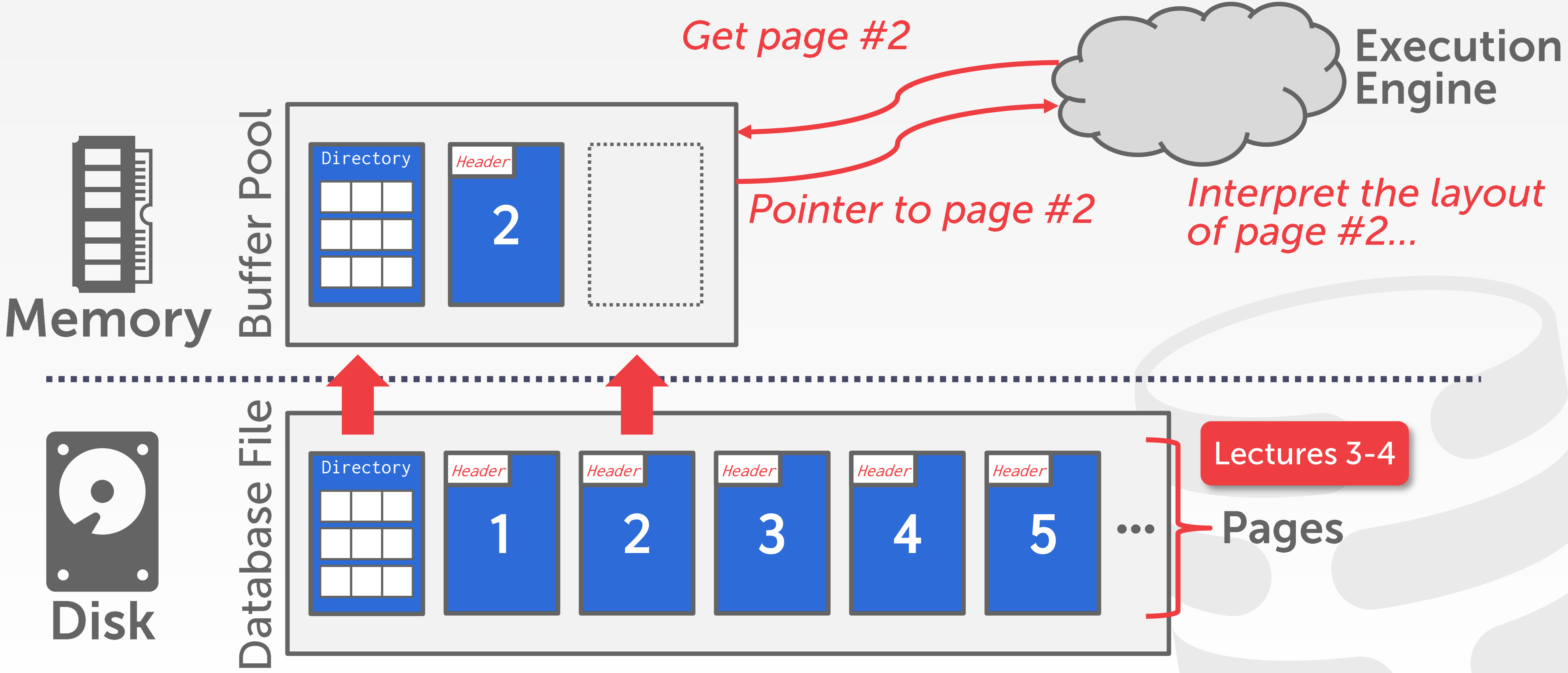
DISK-ORIENTED DBMS



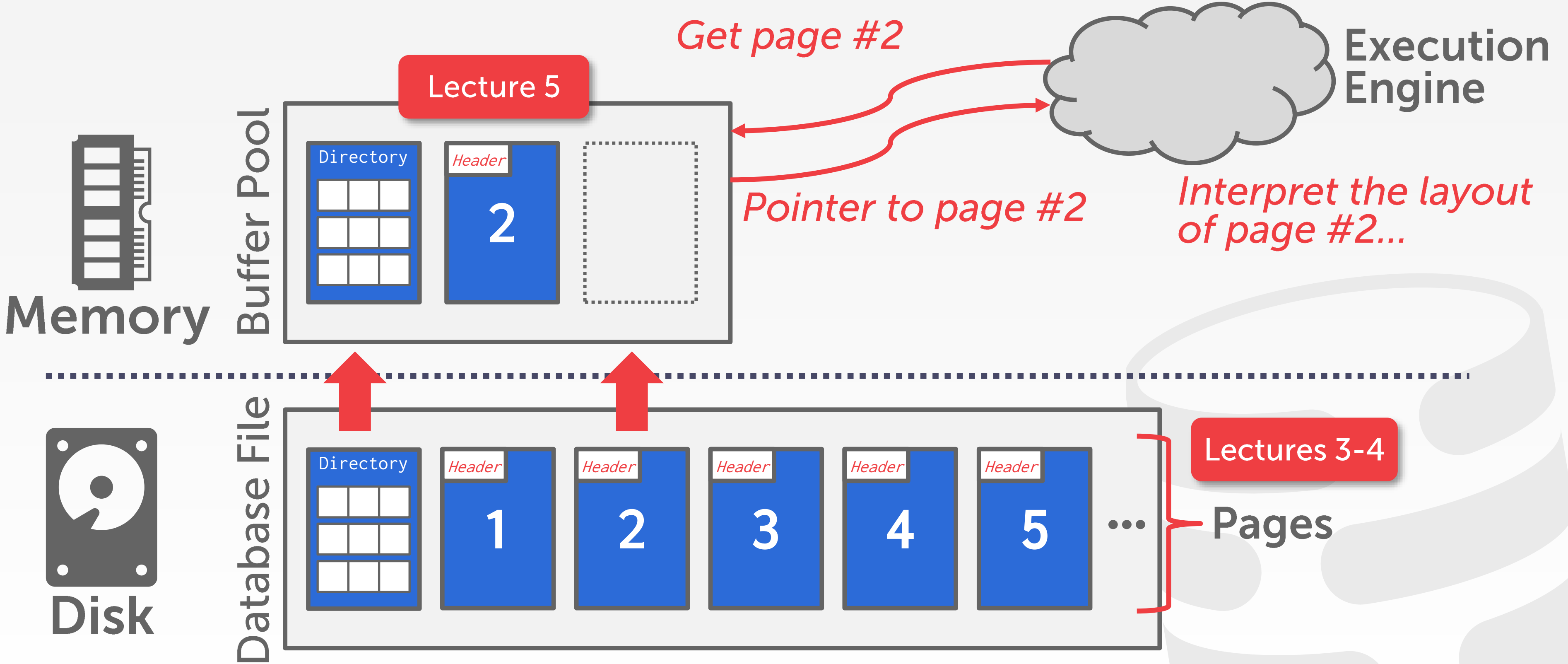
DISK-ORIENTED DBMS



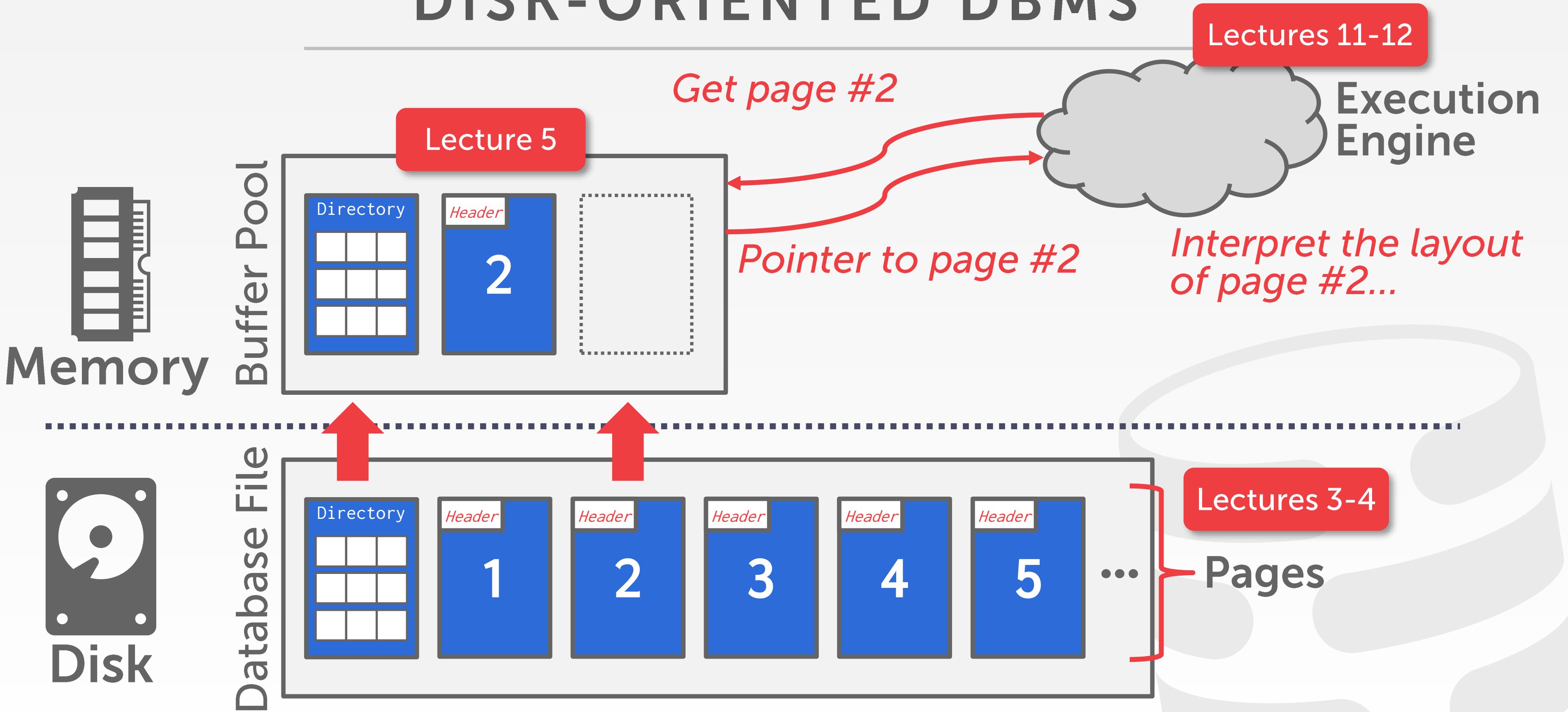
DISK-ORIENTED DBMS



DISK-ORIENTED DBMS



DISK-ORIENTED DBMS



WHY NOT USE THE OS?

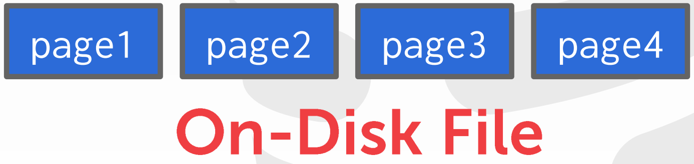
The DBMS can use memory mapping (`mmap`) to store the contents of a file into the address space of a program.

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.

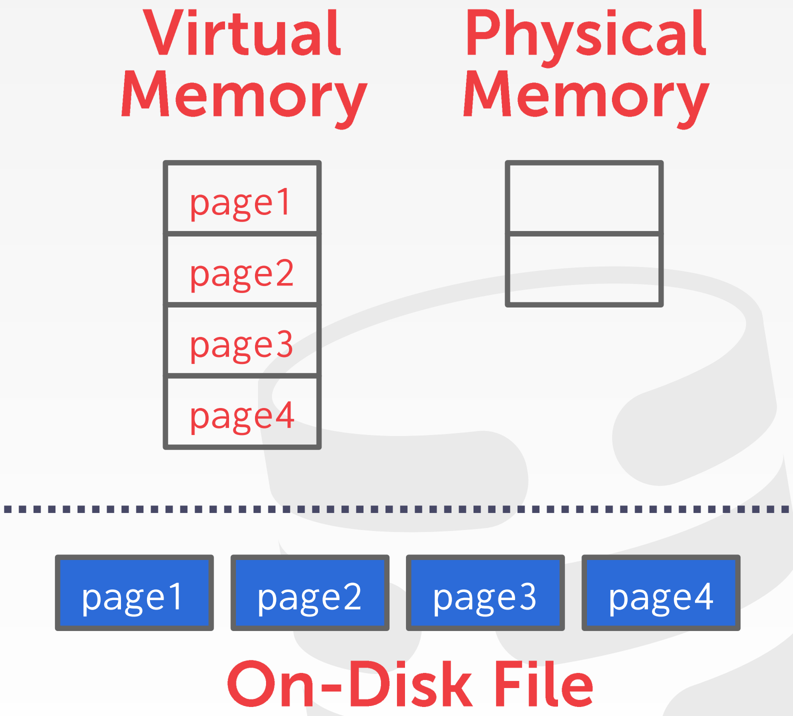
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

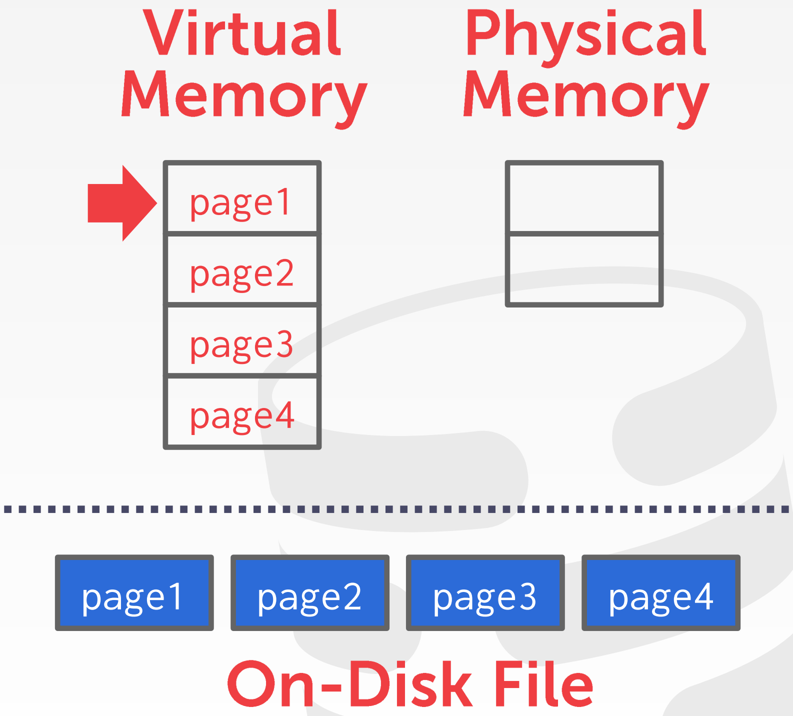
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

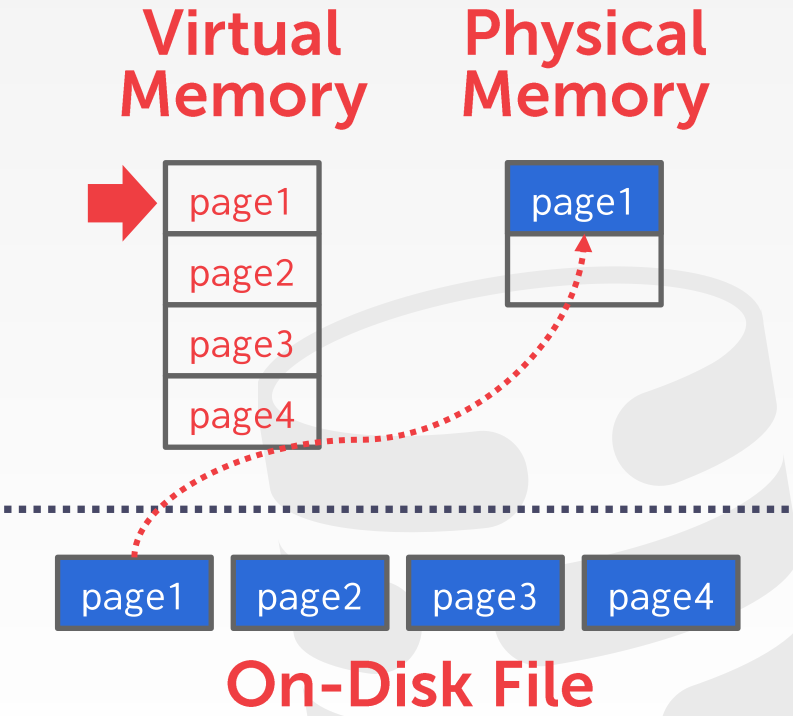
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

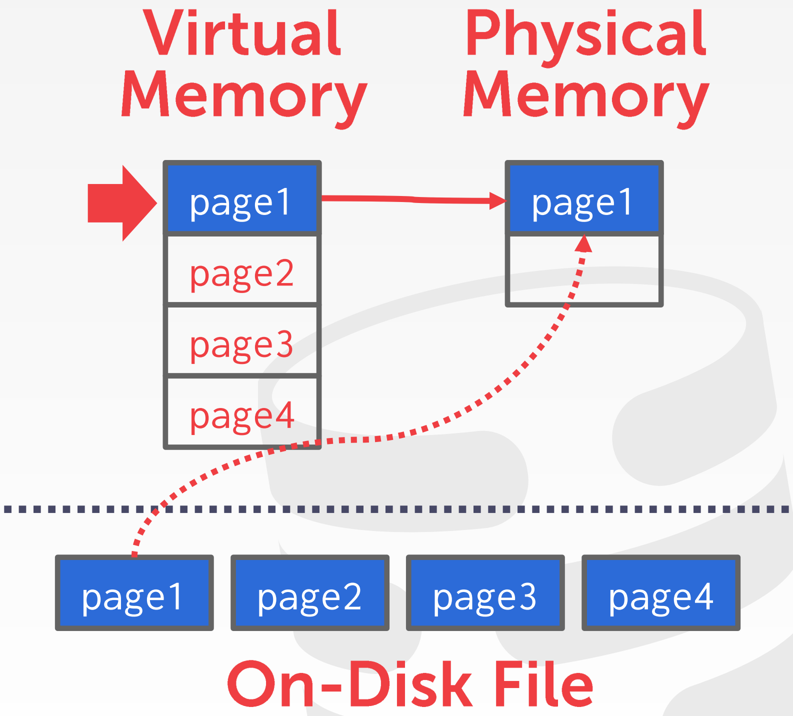
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

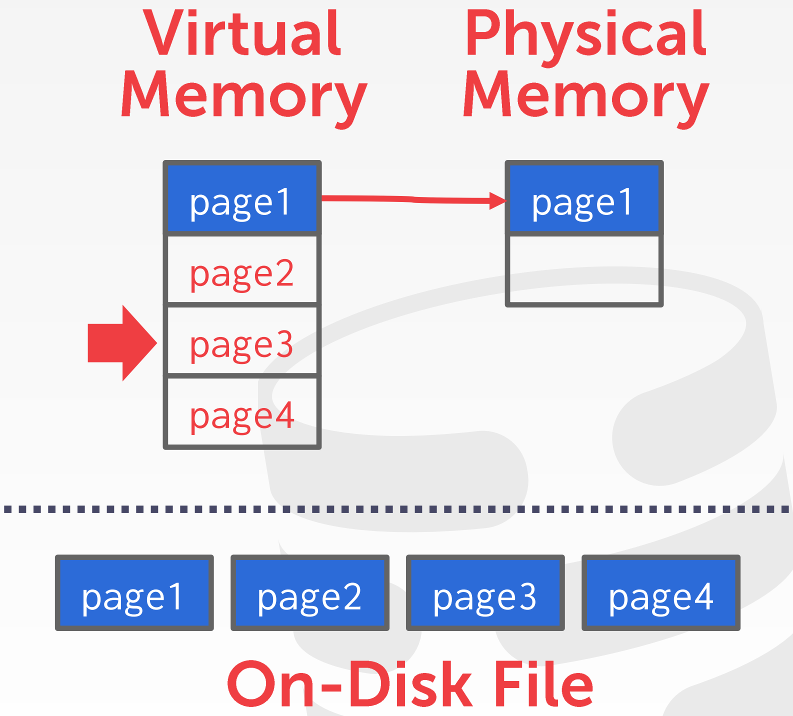
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

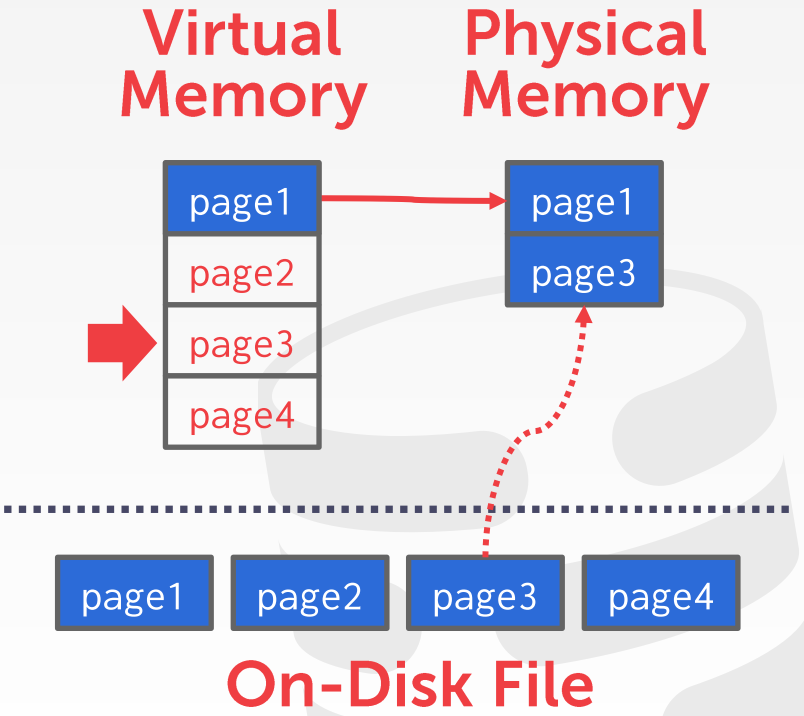
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

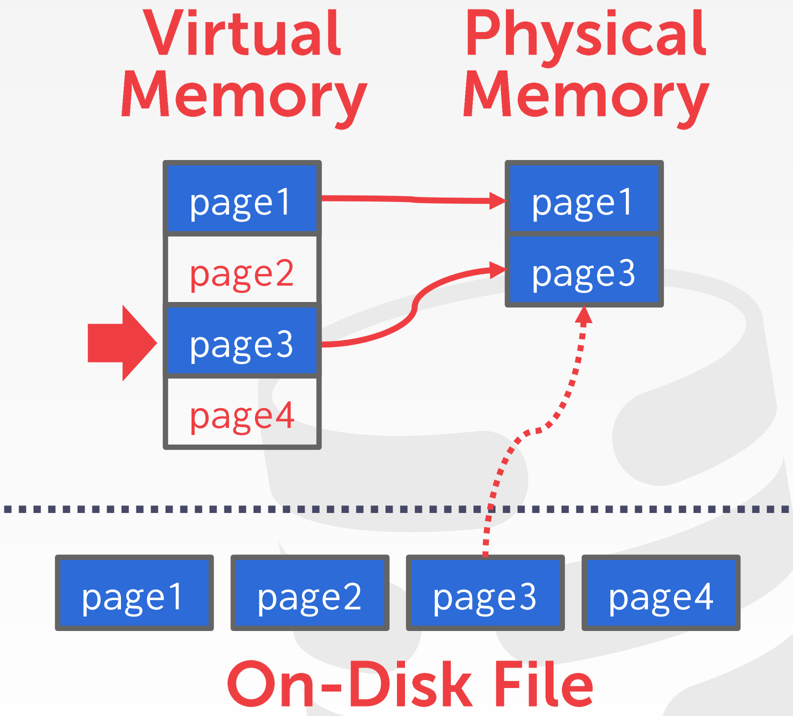
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

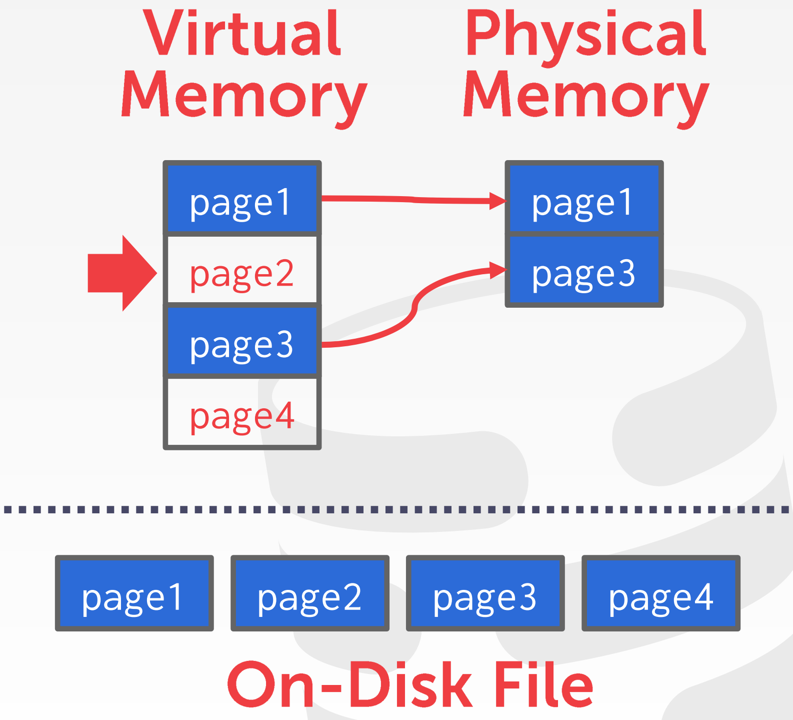
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

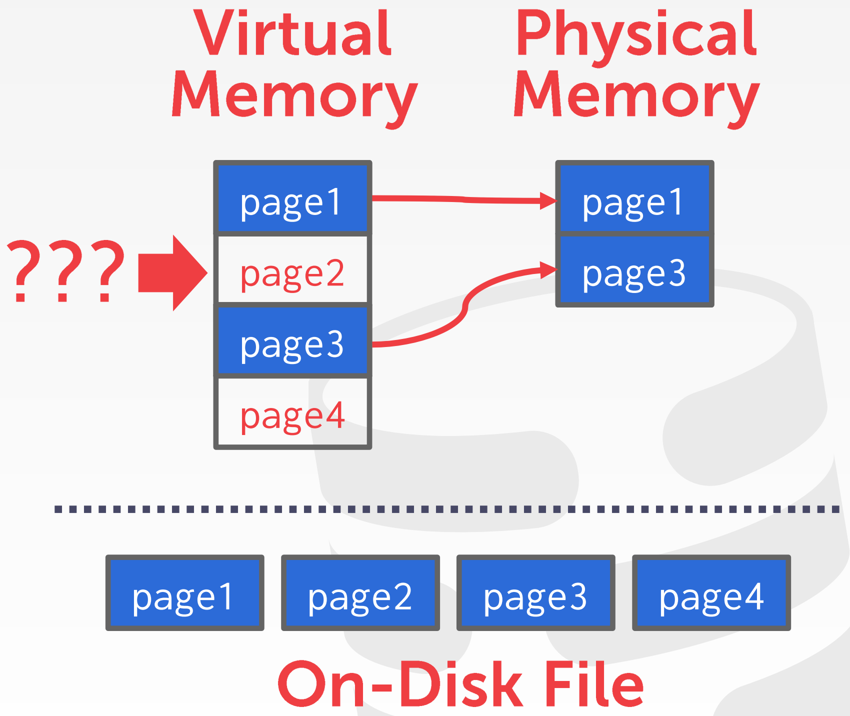
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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.

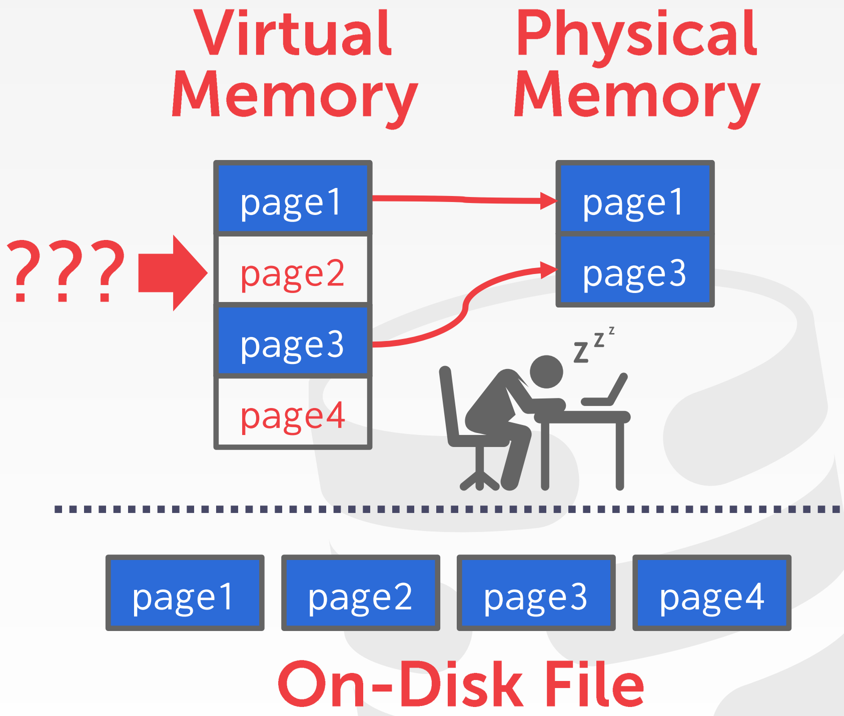
The OS is responsible for moving the pages of the file in and out of memory, so the DBMS doesn't need to worry about it.



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

monetdb LMDB RAVENDB levelDB elasticsearch

Partial Usage

mongoDB MEMSQL SQLite influxdb

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.

The OS is **not** your friend.

DATABASE STORAGE

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

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

DATABASE STORAGE

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

← Today

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

DATABASE STORAGE

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

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

TODAY'S AGENDA

File Storage

Page Layout

Tuple Layout

FILE STORAGE

The DBMS stores a database as one or more files on disk 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

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.

Two ways to represent a heap file:

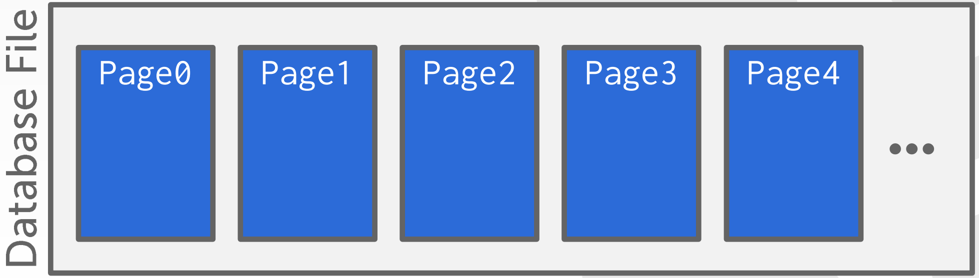
→ Linked List

→ Page Directory

DATABASE HEAP

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

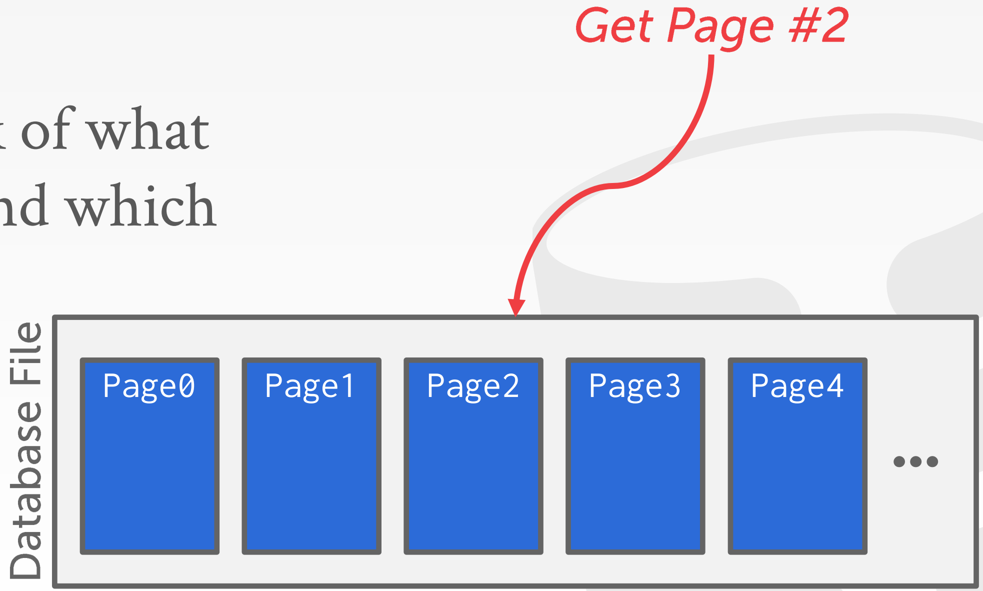
Need meta-data to keep track of what pages exist in multiple files and which ones have free space.



DATABASE HEAP

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

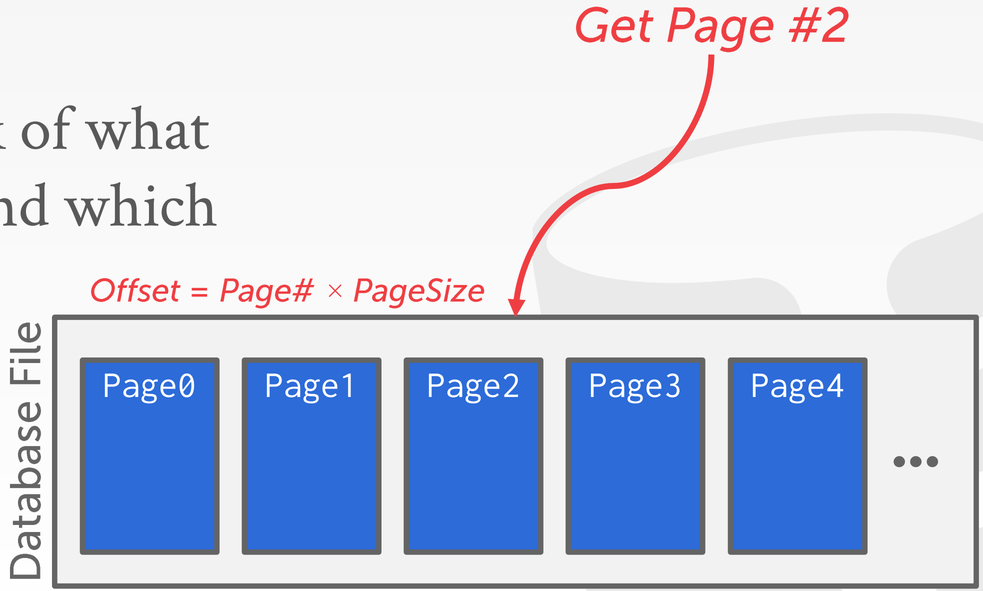
Need meta-data to keep track of what pages exist in multiple files and which ones have free space.



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.



DATABASE HEAP

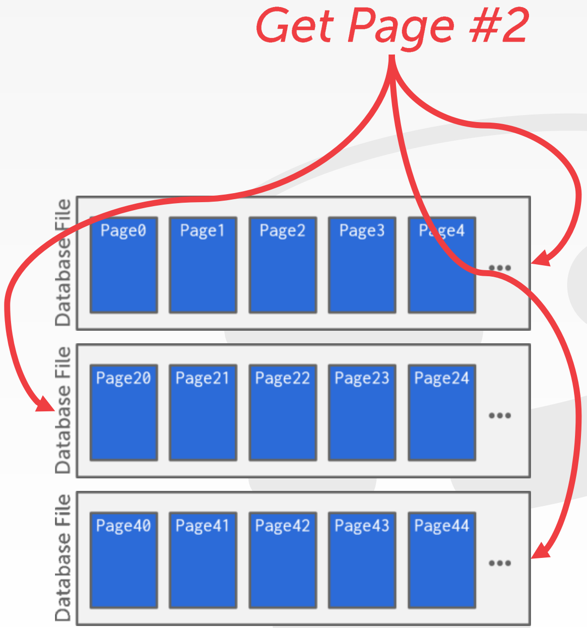
It is easy to find pages if there is only a single heap file.

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

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.

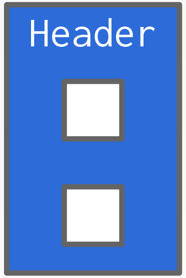


HEAP FILE: LINKED LIST

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 how many free slots they currently have.

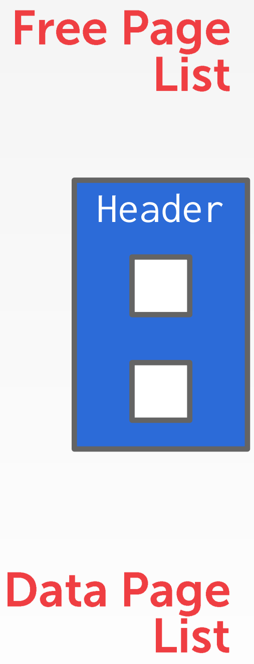
HEAP FILE: LINKED LIST

Maintain a <u>header page</u> at the beginning of the file that stores two pointers:

→ HEAD of the <u>free page list</u>.

→ HEAD of the <u>data page list</u>.

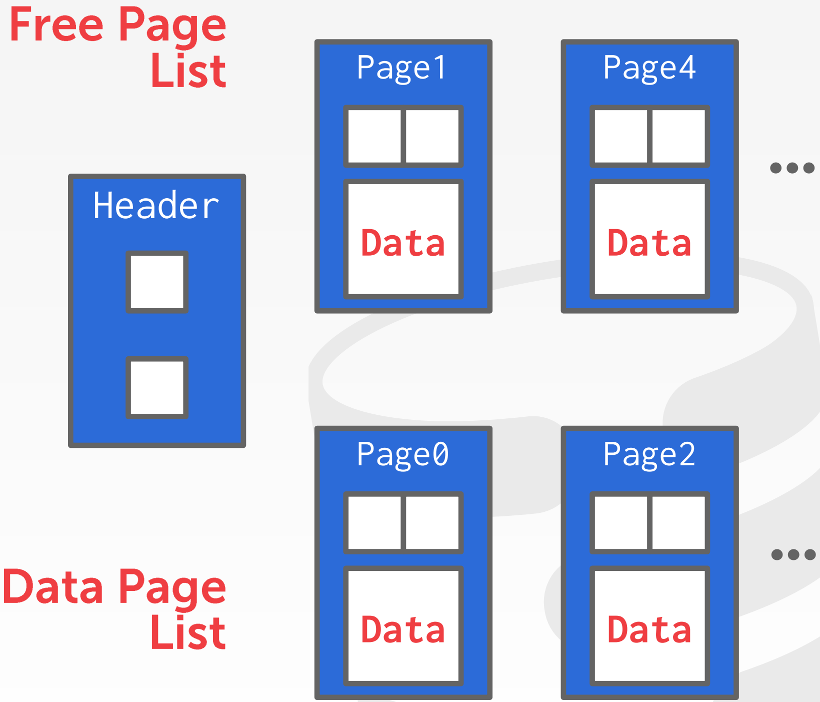
Each page keeps track of how many free slots they currently have.



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.

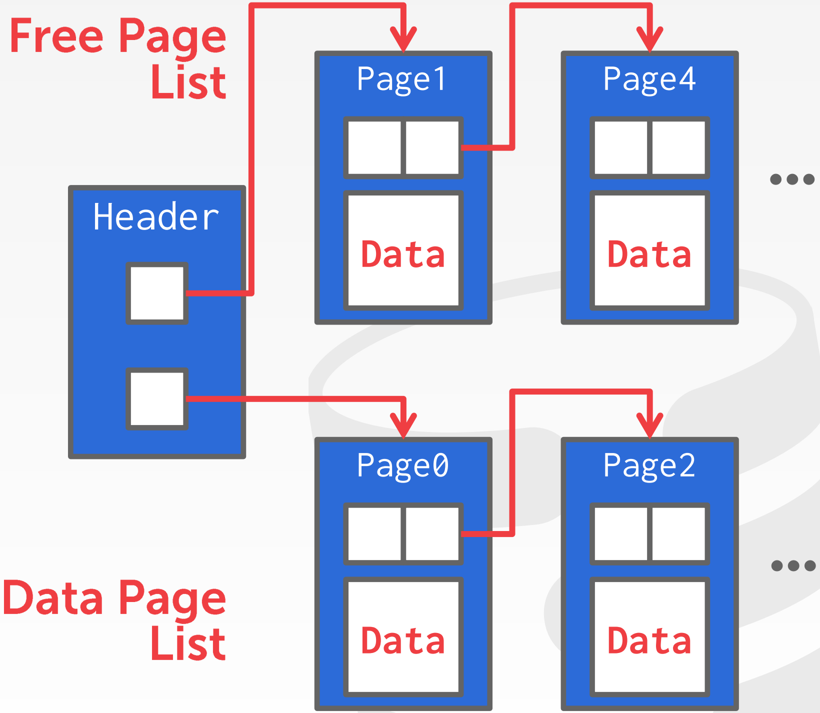
Each page keeps track of how many free slots they currently have.



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 how many free slots they currently have.

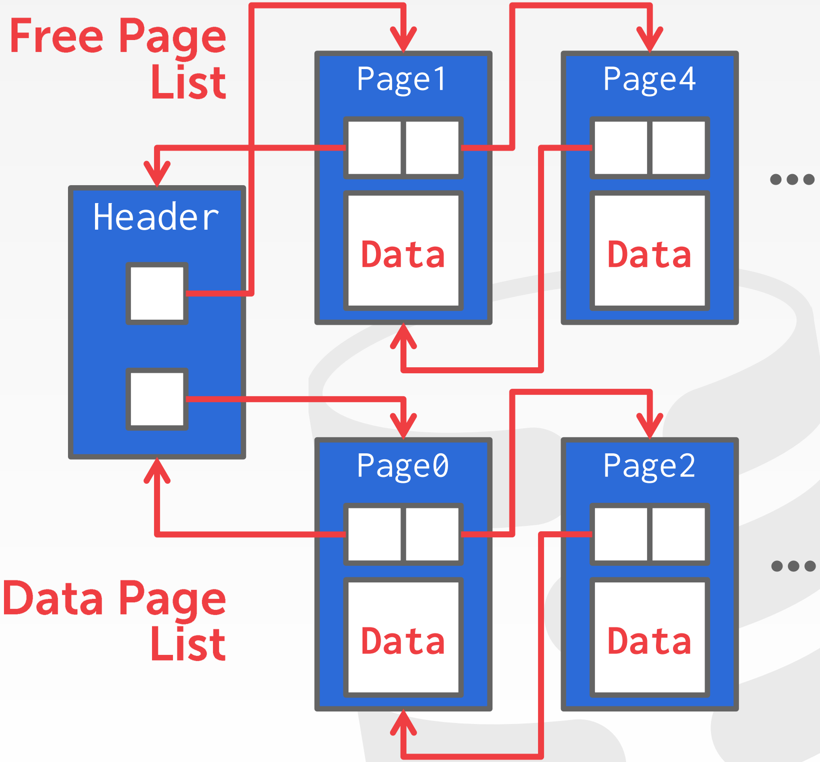


HEAP FILE: LINKED LIST

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 how many free slots they currently have.

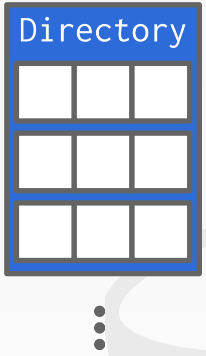


HEAP FILE: PAGE DIRECTORY

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.

Must make sure that the directory pages are in sync with the data pages.

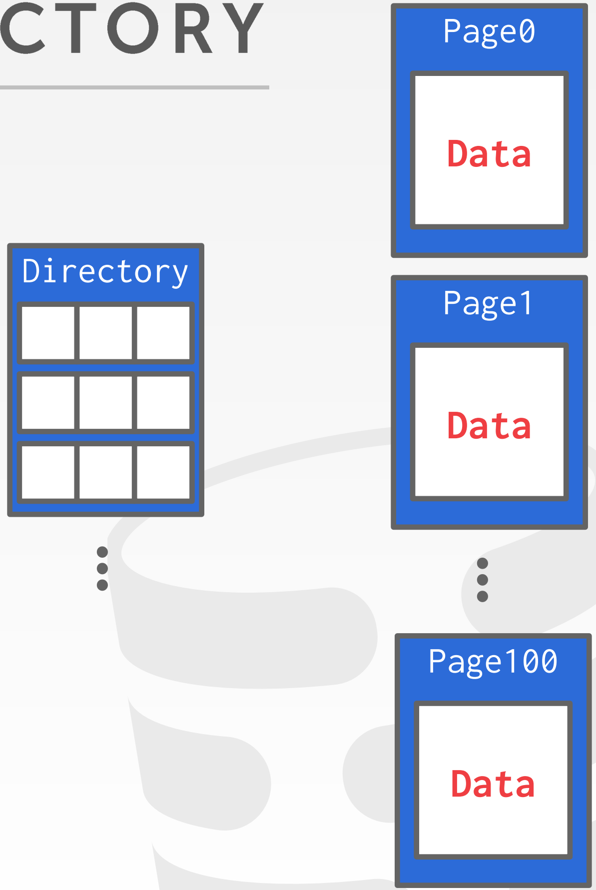


HEAP FILE: PAGE DIRECTORY

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.

Must make sure that the directory pages are in sync with the data pages.

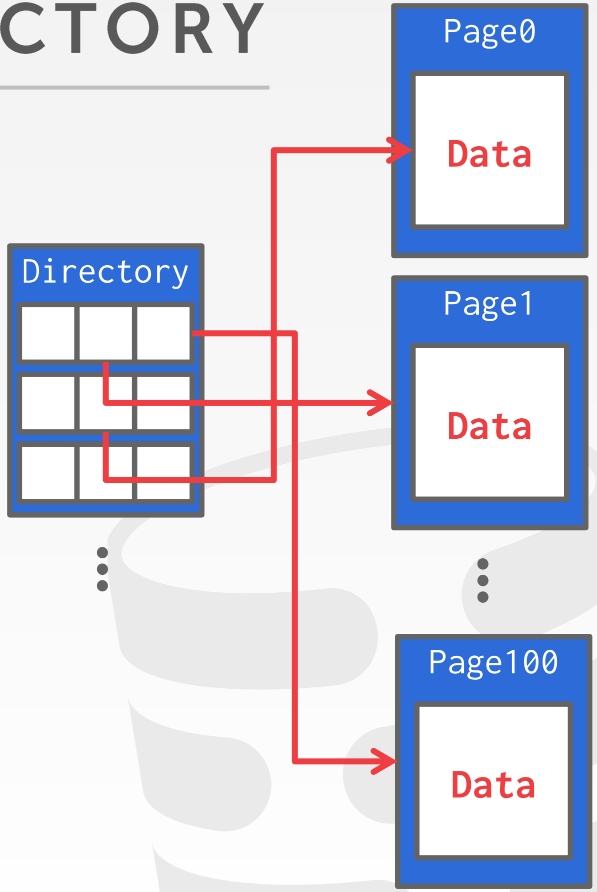


HEAP FILE: PAGE DIRECTORY

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.

Must make sure that the directory pages are in sync with the data pages.



TODAY'S AGENDA

File Storage

Page Layout

Tuple Layout

PAGE HEADER

Every page contains a <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.

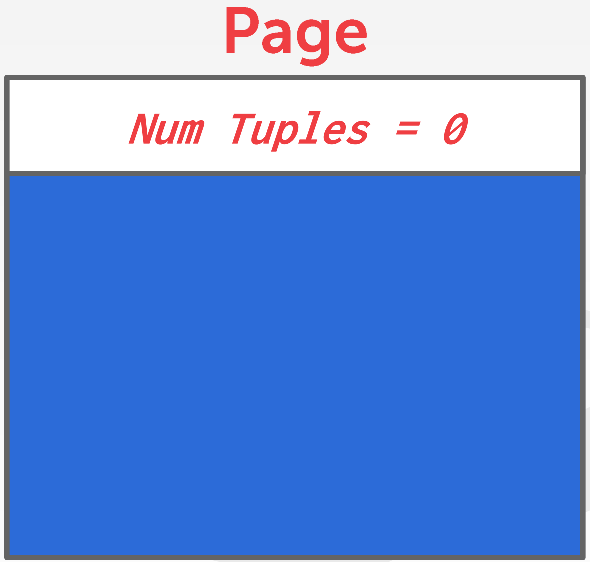
Two approaches:

→ Tuple-oriented

→ Log-structured

TUPLE STORAGE

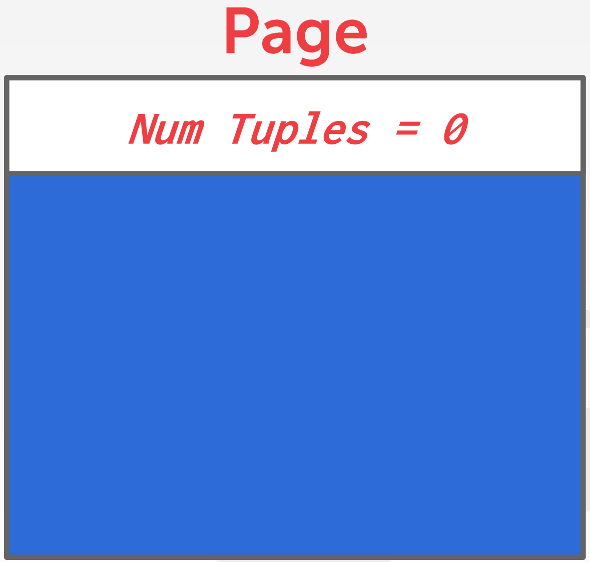
How to store tuples in a page?



TUPLE STORAGE

How to store tuples in a page?

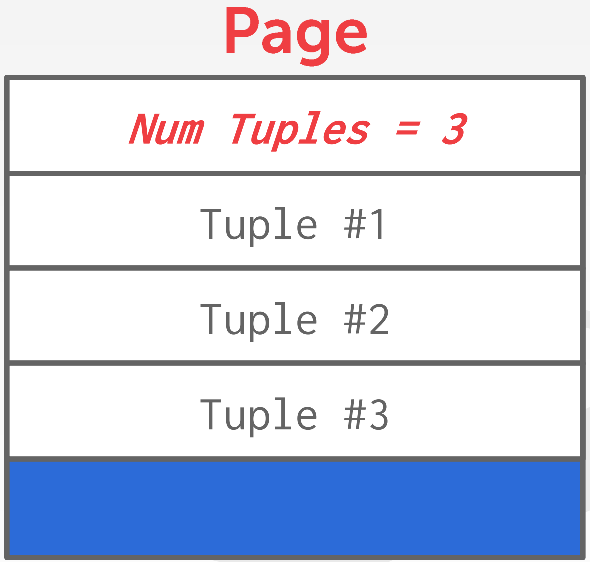
Strawman Idea: Keep track of the number of tuples in a page and then just append a new tuple to the end.



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.

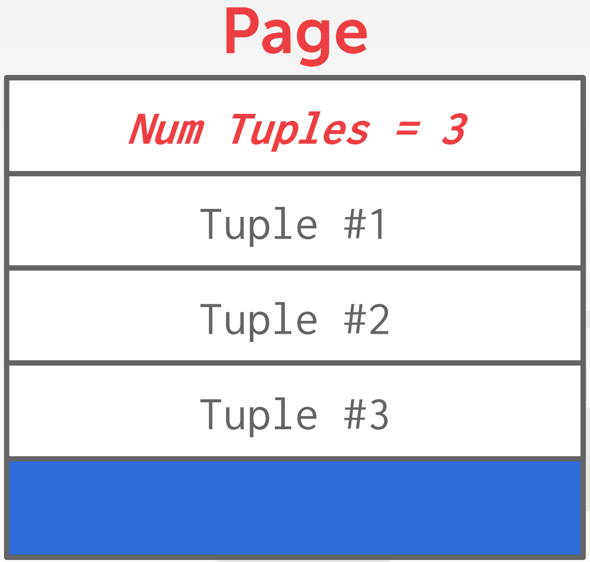


TUPLE STORAGE

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?

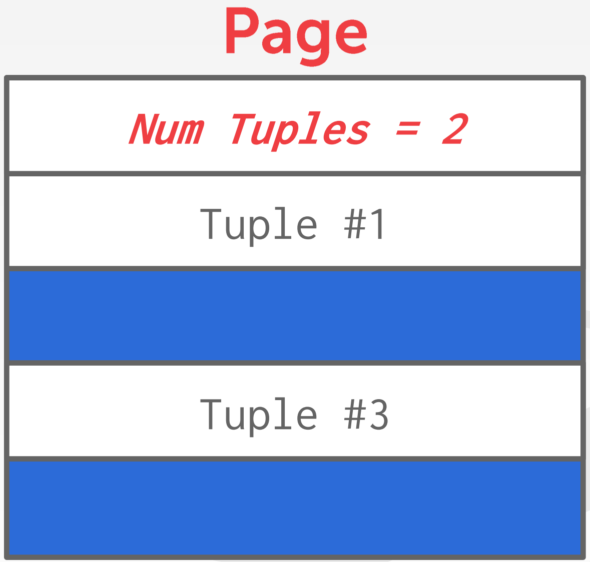


TUPLE STORAGE

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?

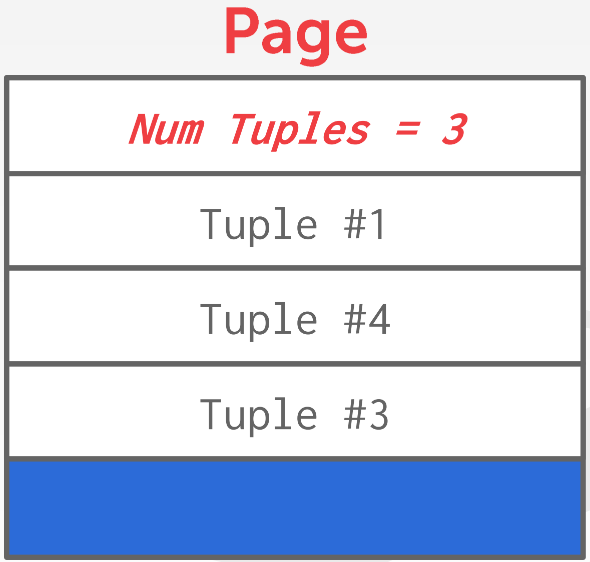


TUPLE STORAGE

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?



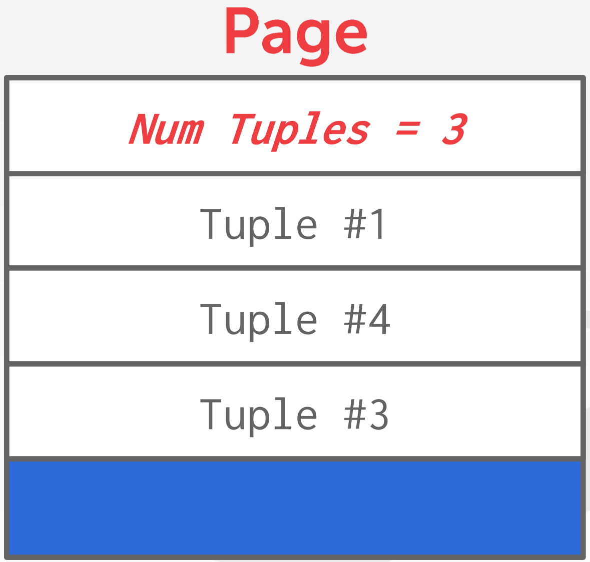
TUPLE STORAGE

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?

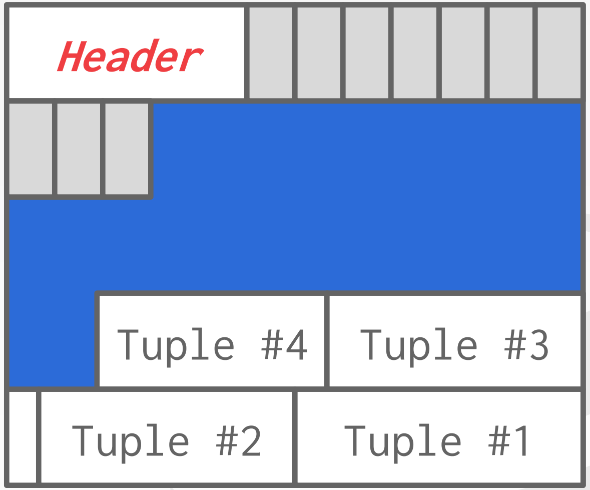


SLOTTED PAGES

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.

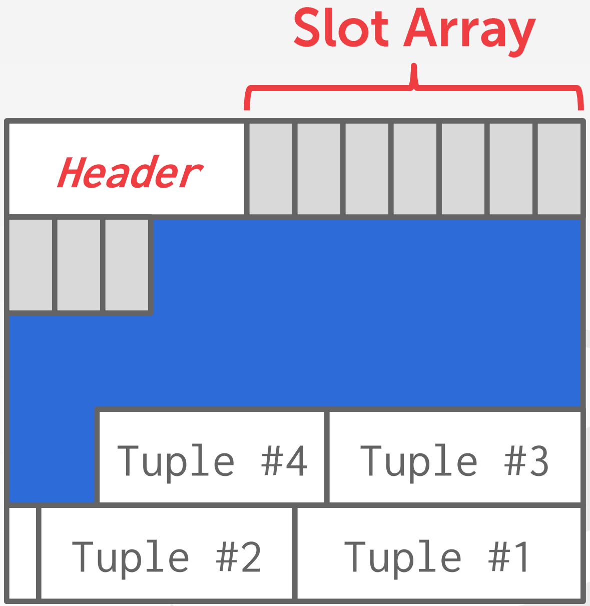


SLOTTED PAGES

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.

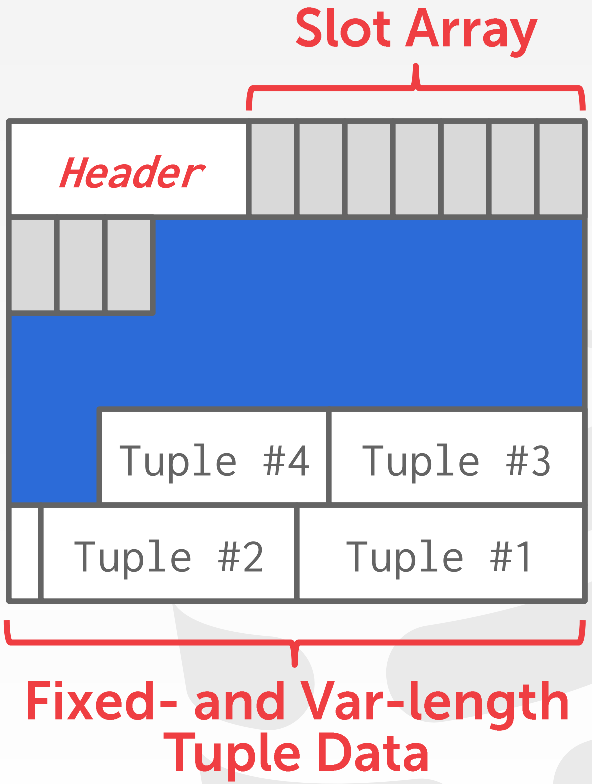


SLOTTED PAGES

The most common layout scheme is called <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.

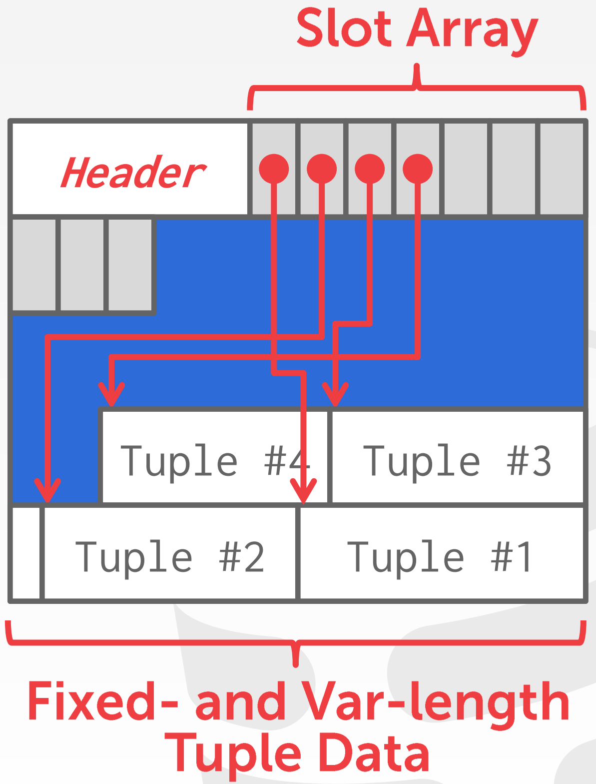


SLOTTED PAGES

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

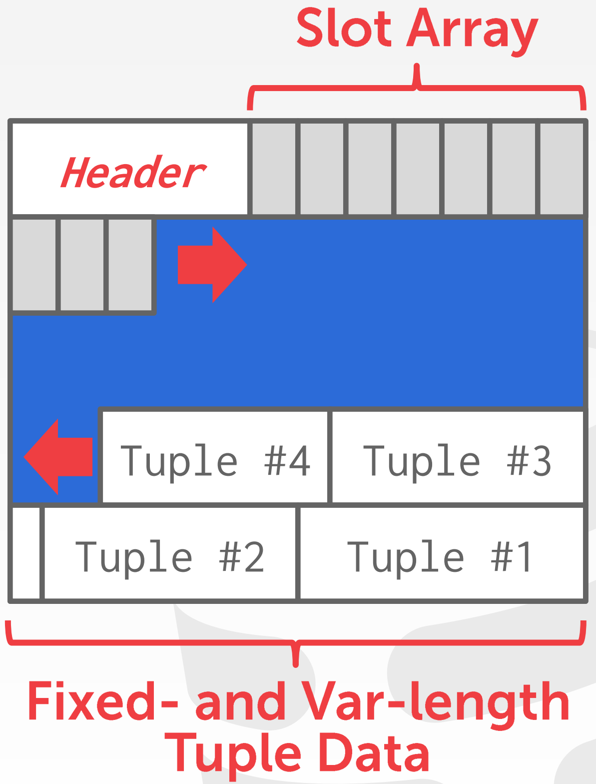
The most common layout scheme is called slotted pages.

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

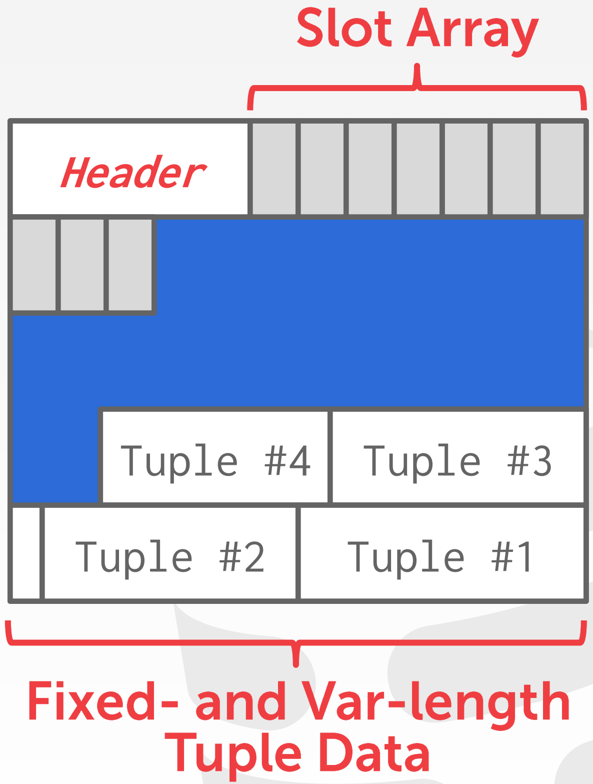
The most common layout scheme is called slotted pages.

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.

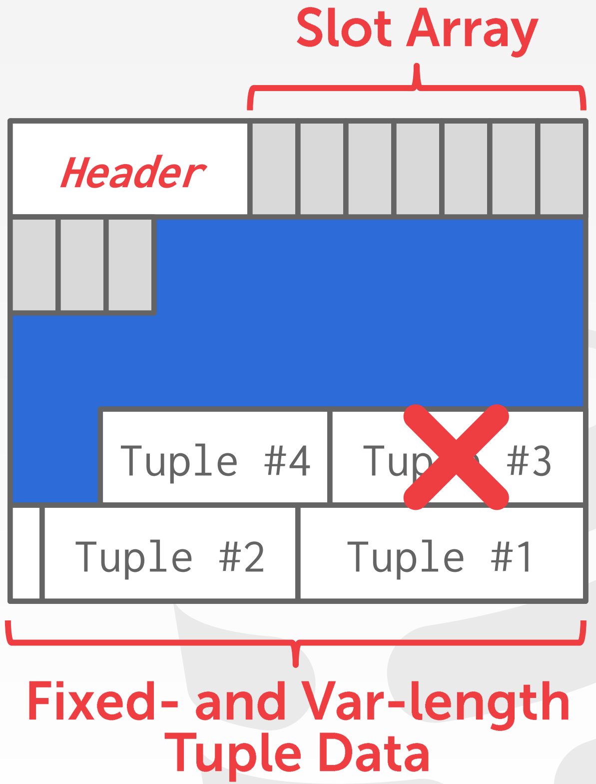


SLOTTED PAGES

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.

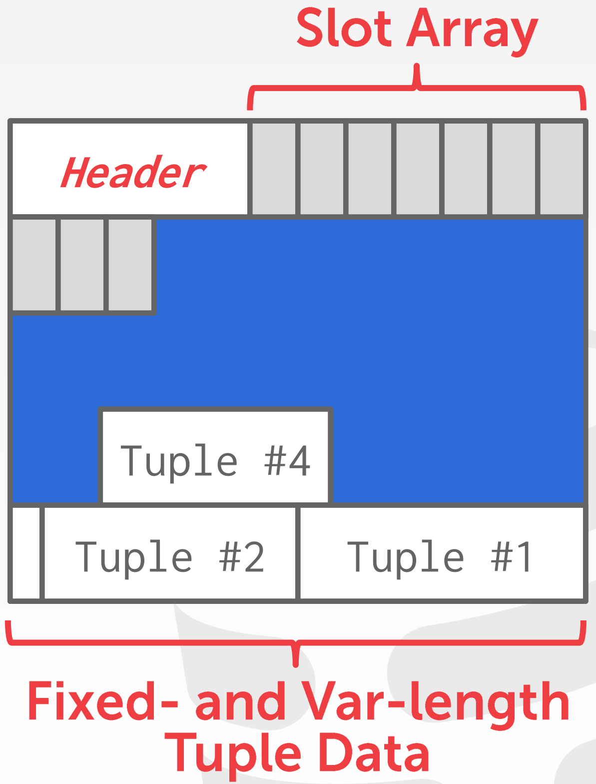


SLOTTED PAGES

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.

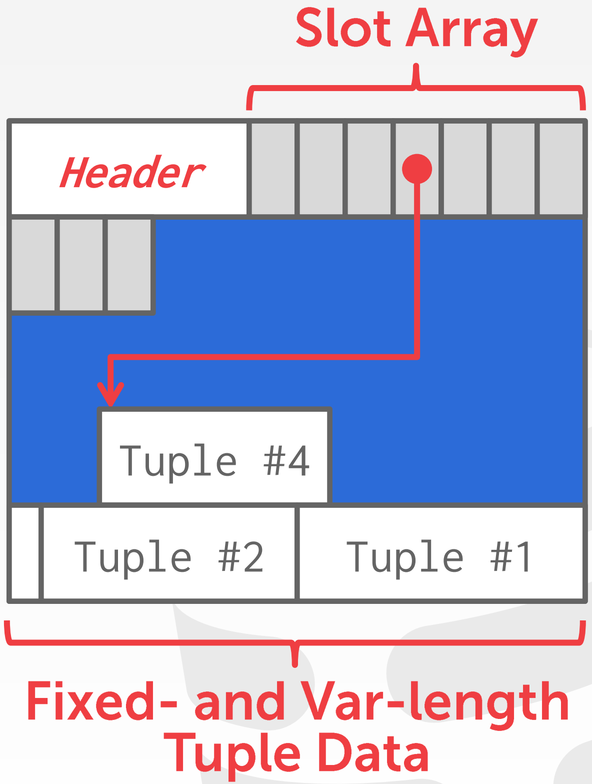


SLOTTED PAGES

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.



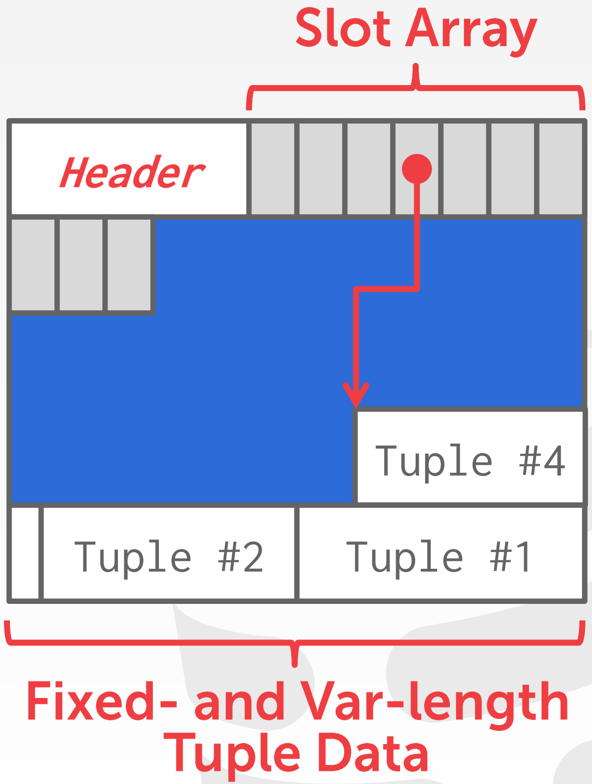
SLOTTED PAGES

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

Tuple Layout

TUPLE LAYOUT

A tuple is essentially a sequence of bytes.

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

TUPLE HEADER

Each tuple is prefixed with a header that contains meta-data about it.
→ Visibility info (concurrency control)
→ Bit Map for **NULL** values.

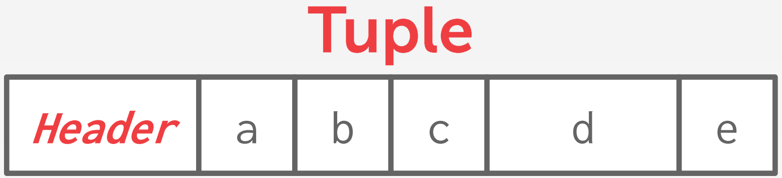
We do not need to store meta-data about the schema.

TUPLE DATA

Attributes are typically stored in the order that you specify them when you create the table.

This is done for software engineering reasons (i.e., simplicity).

However, it might be more efficient to lay them out differently.




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

DENORMALIZED TUPLE DATA

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

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

→ Can make updates more expensive.

```
CREATE TABLE foo (
  a INT PRIMARY KEY,
  b INT NOT NULL,
);
```

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

DENORMALIZED TUPLE DATA

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

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

→ Can make updates more expensive.

```
CREATE TABLE foo (
  a INT PRIMARY KEY,
  b INT NOT NULL,
);
```

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

DENORMALIZED TUPLE DATA

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

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

→ Can make updates more expensive.

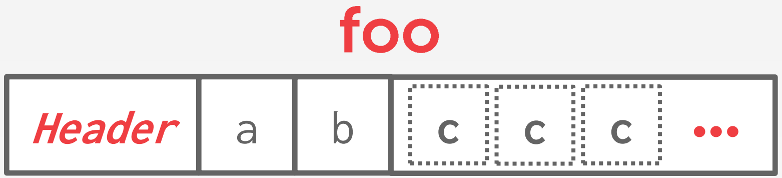
foo

Header	a	b

bar

Header	c	a
Header	c	a
Header	c	a

DENORMALIZED TUPLE DATA



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

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

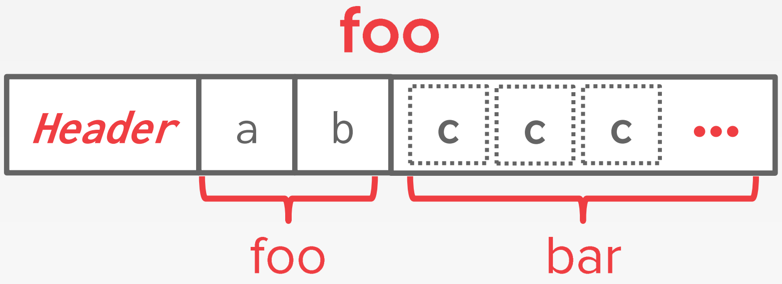
→ Can make updates more expensive.

DENORMALIZED TUPLE DATA

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

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

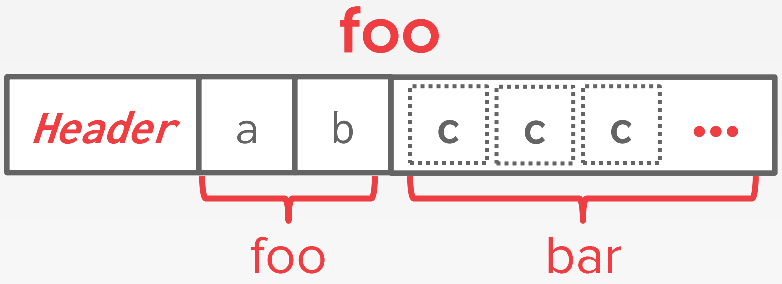
→ Can make updates more expensive.



DENORMALIZED TUPLE DATA

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

→ Potentially reduces the amount of I/O for common workload patterns.
→ Can make updates more expensive.



Not a new idea.

→ IBM System R did this in the 1970s.
→ Several NoSQL DBMSs do this without calling it physical denormalization.

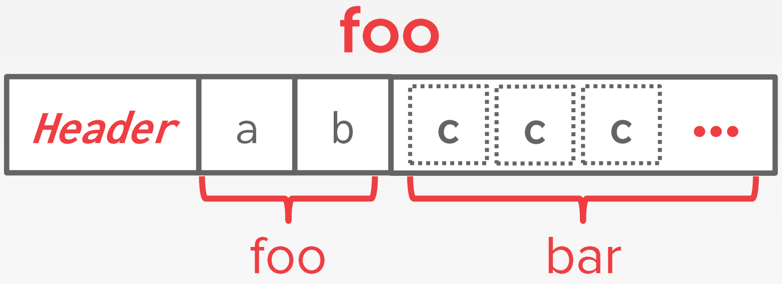
DENORMALIZED TUPLE DATA

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

→ Potentially reduces the amount of I/O for common workload patterns.
→ Can make updates more expensive.

Not a new idea.

→ IBM System R did this in the 1970s.
→ Several NoSQL DBMSs do this without calling it physical denormalization.



CONCLUSION

Database is organized in pages.

Different ways to track pages.

Different ways to store pages.

Different ways to store tuples.

NEXT CLASS

Log-Structured Storage

Value Representation

Storage Models