



THE UNIVERSITY  
*of* EDINBURGH

# Advanced Database Systems

Spring 2024

---

Lecture #10:

## HW & Disk Space Management

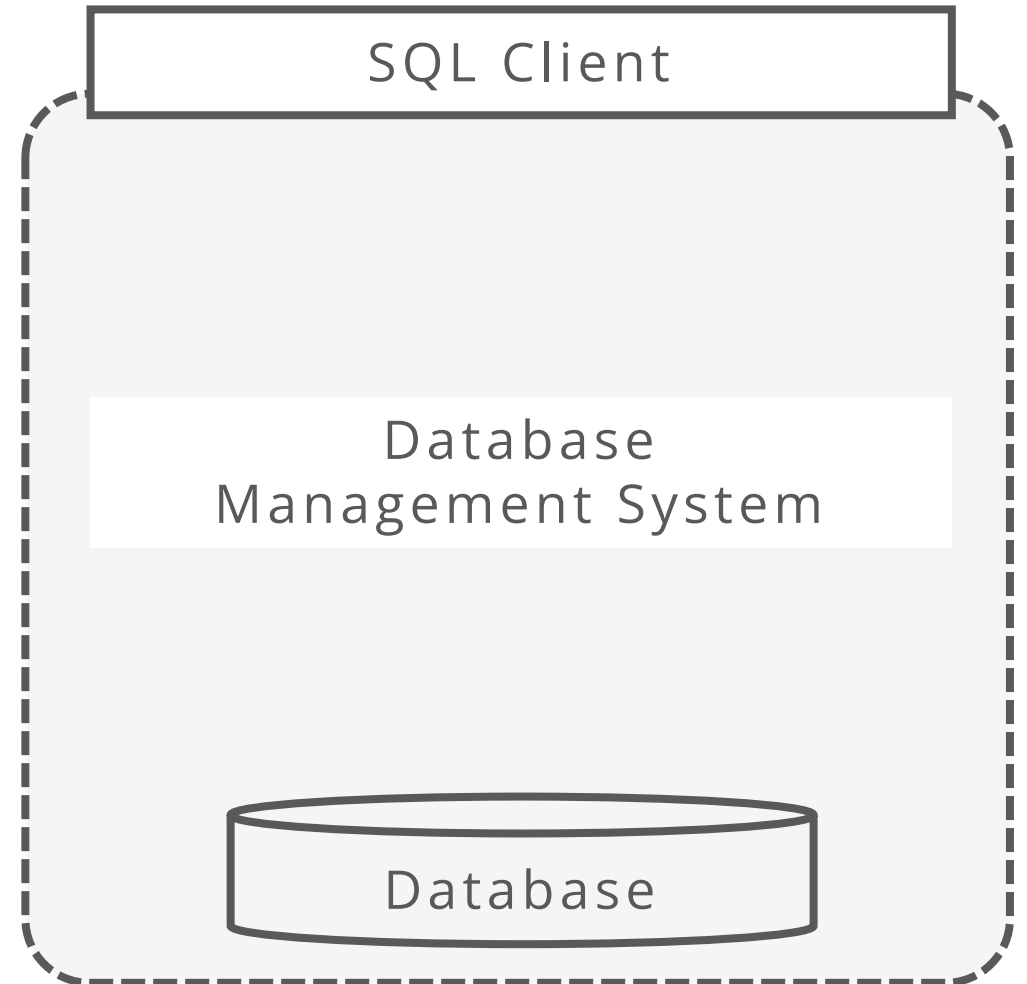
R&G: Chapters 1, 9.1, 9.3

# DBMS: BIG PICTURE

SQL clients interact with a DBMS

You know how to write a SQL query

How is a SQL query executed?

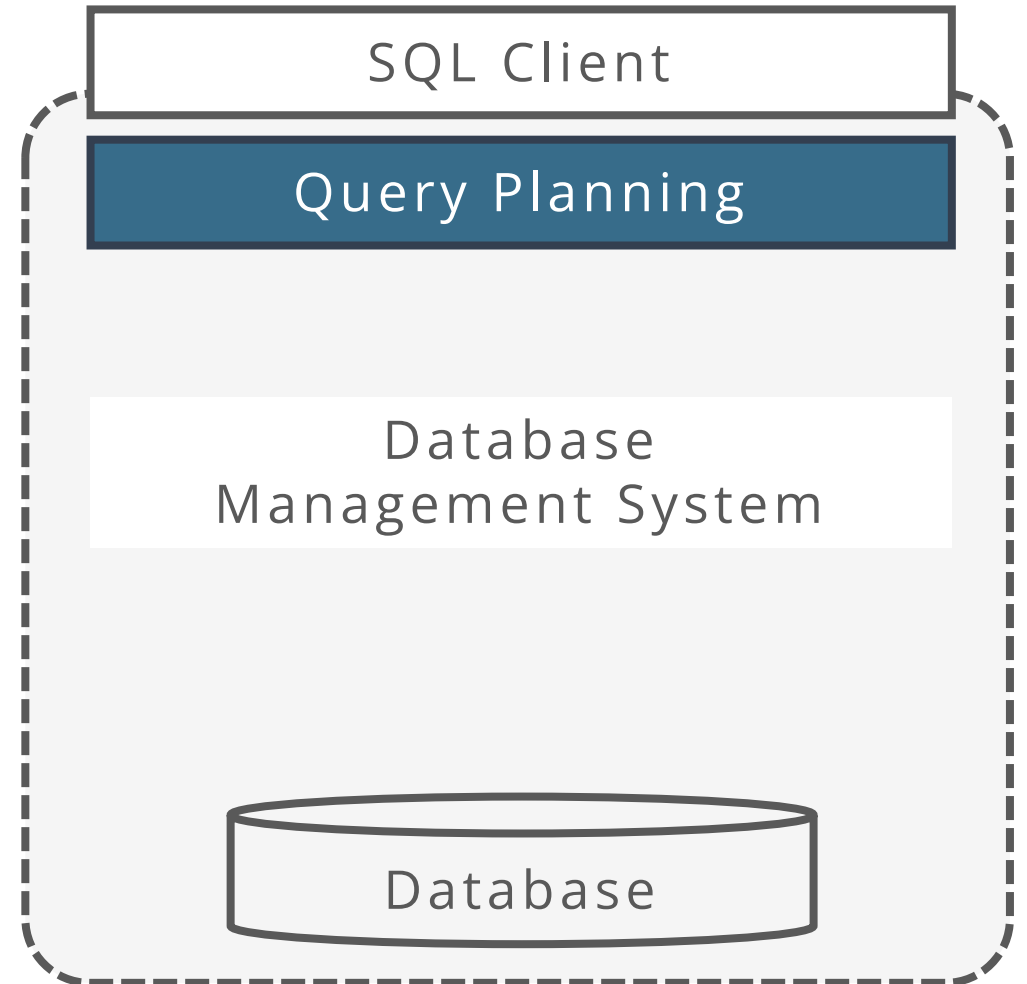


# DBMS: QUERY PLANNING

Parse, check, and verify the SQL query

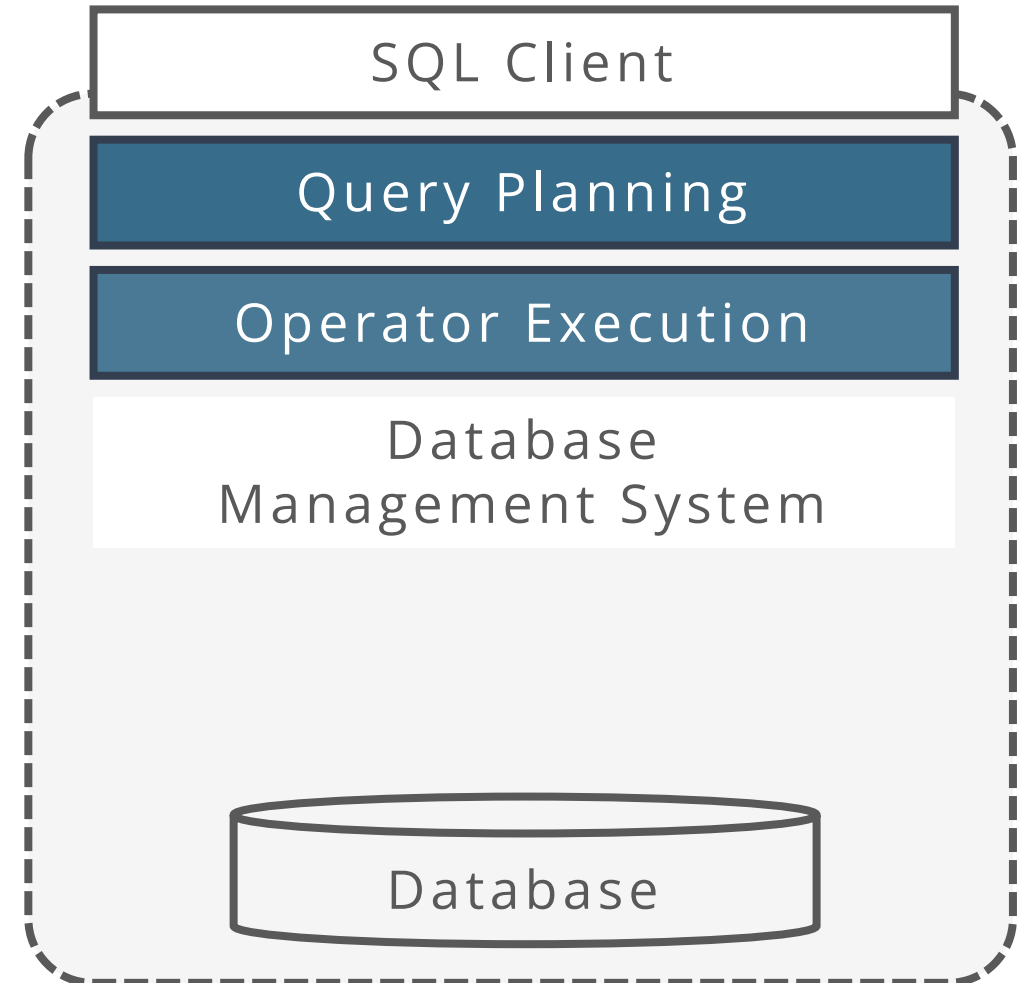
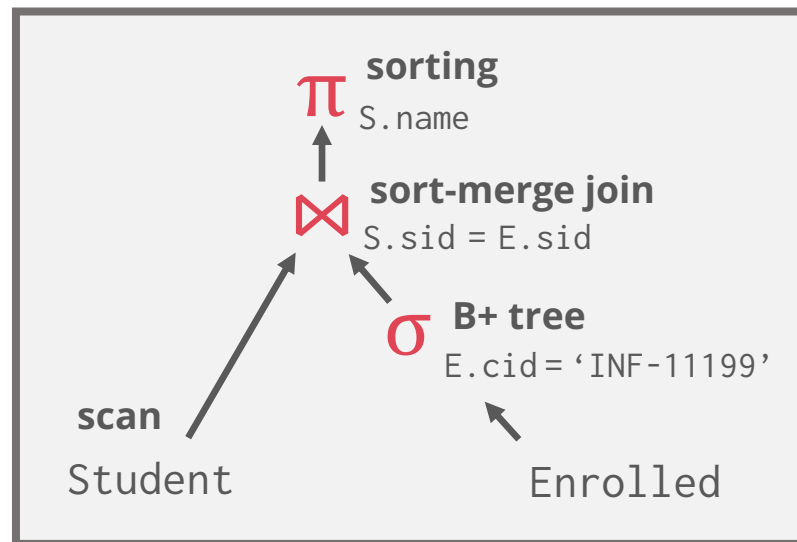
```
SELECT S.name
FROM Student S JOIN Enrolled E
ON S.sid = E.sid
WHERE E.cid = 'INF-11199'
```

Translate into an efficient relational query plan that can be executed



# DBMS: OPERATOR EXECUTION

Execute a dataflow by operating on *records and files*



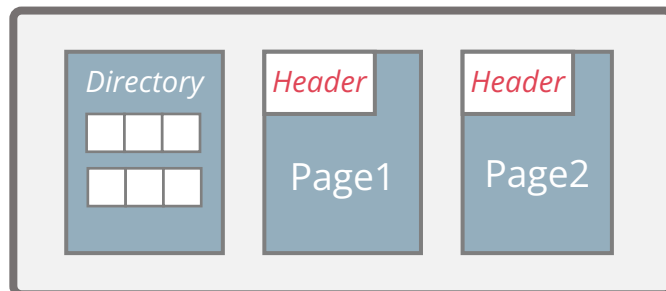
# DBMS: FILES & INDEX MANAGEMENT

Organise tables and records as groups of pages in a logical file

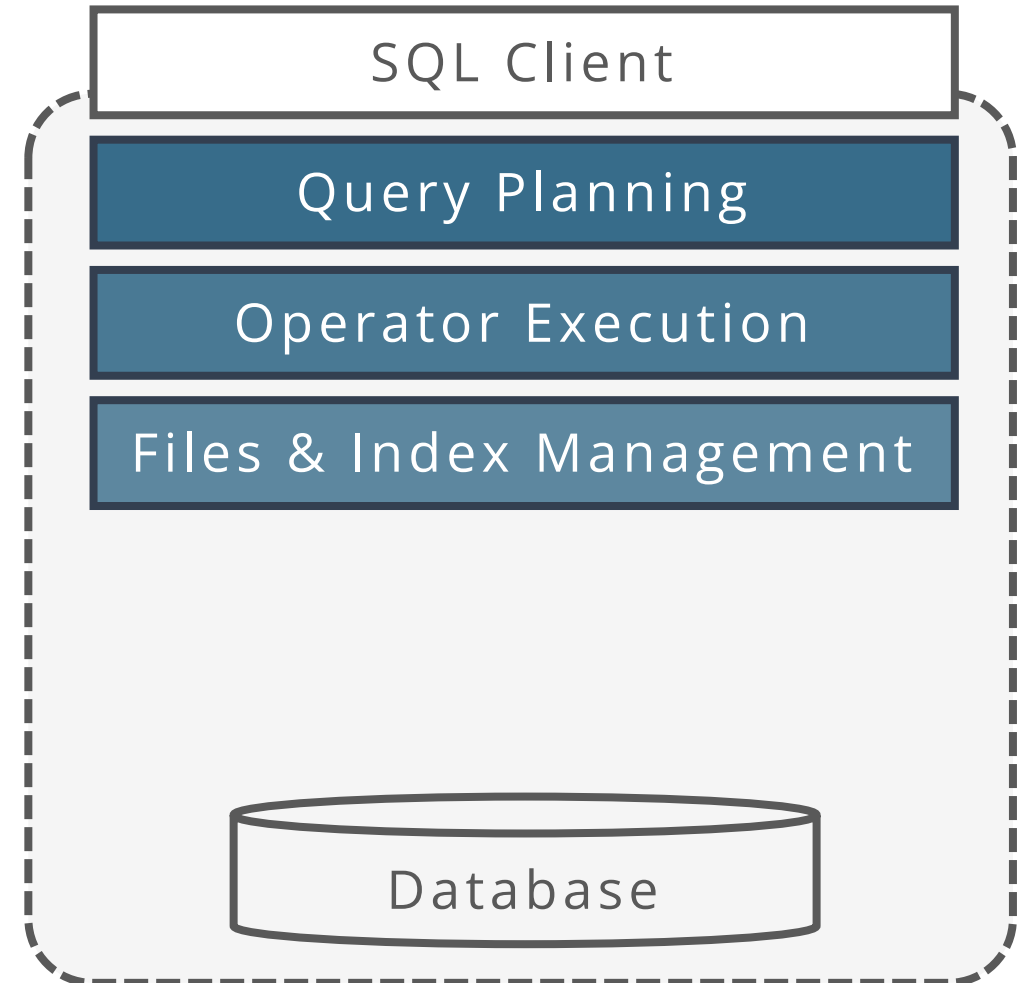
sid	name	dept	age
12344	Jones	CS	18
12355	Smith	Physics	23
12366	Gold	CS	21



Disk

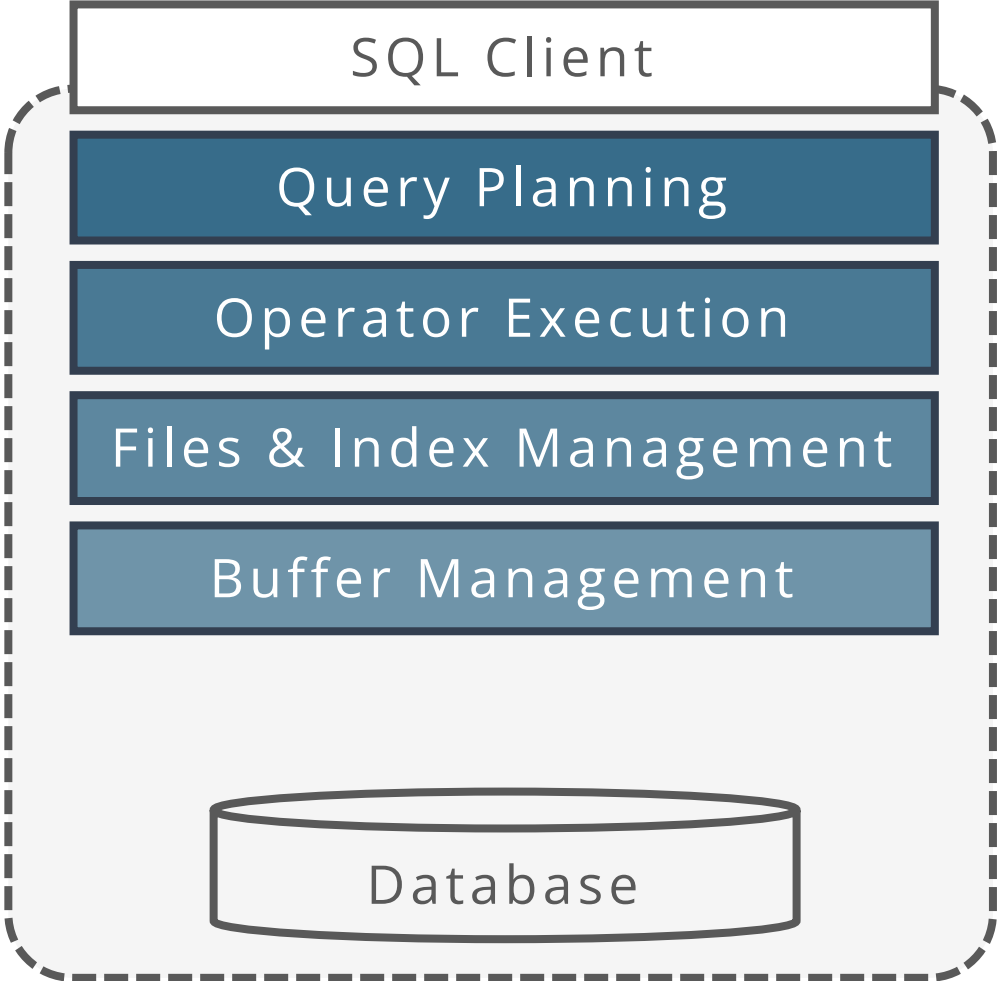
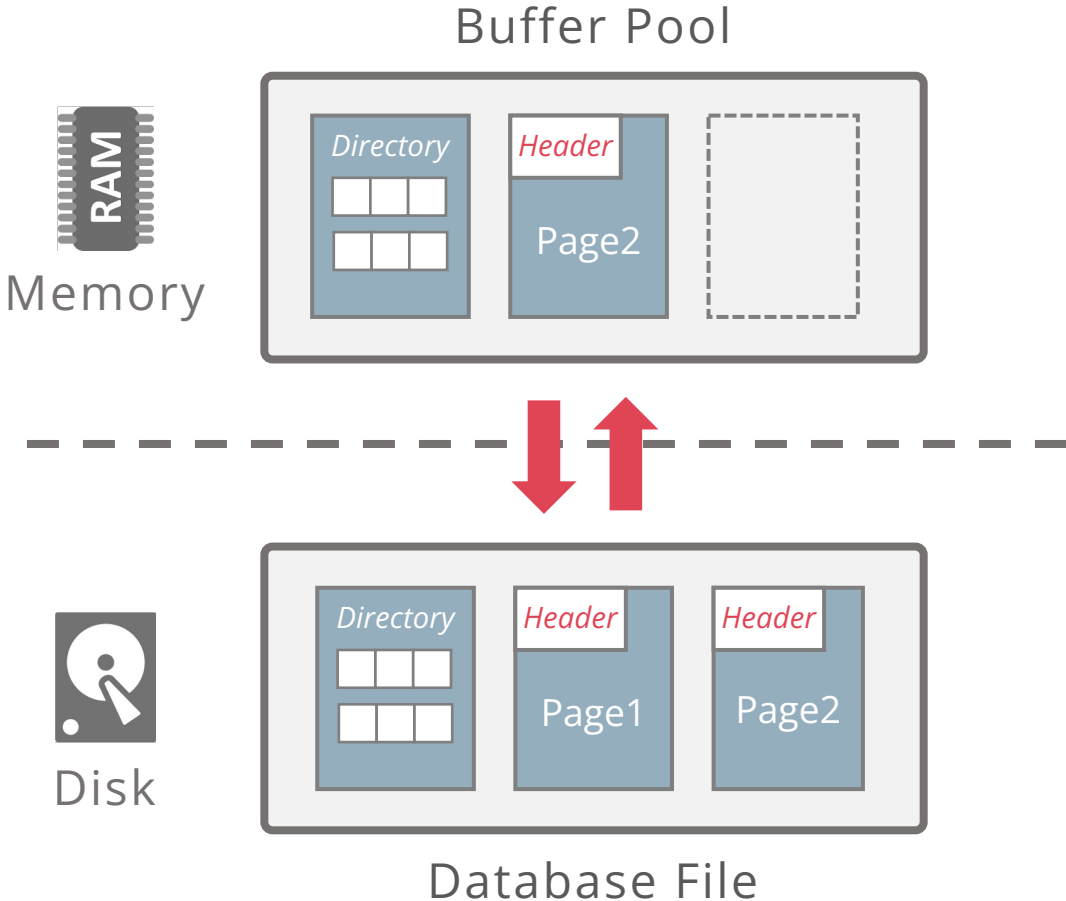


Database File



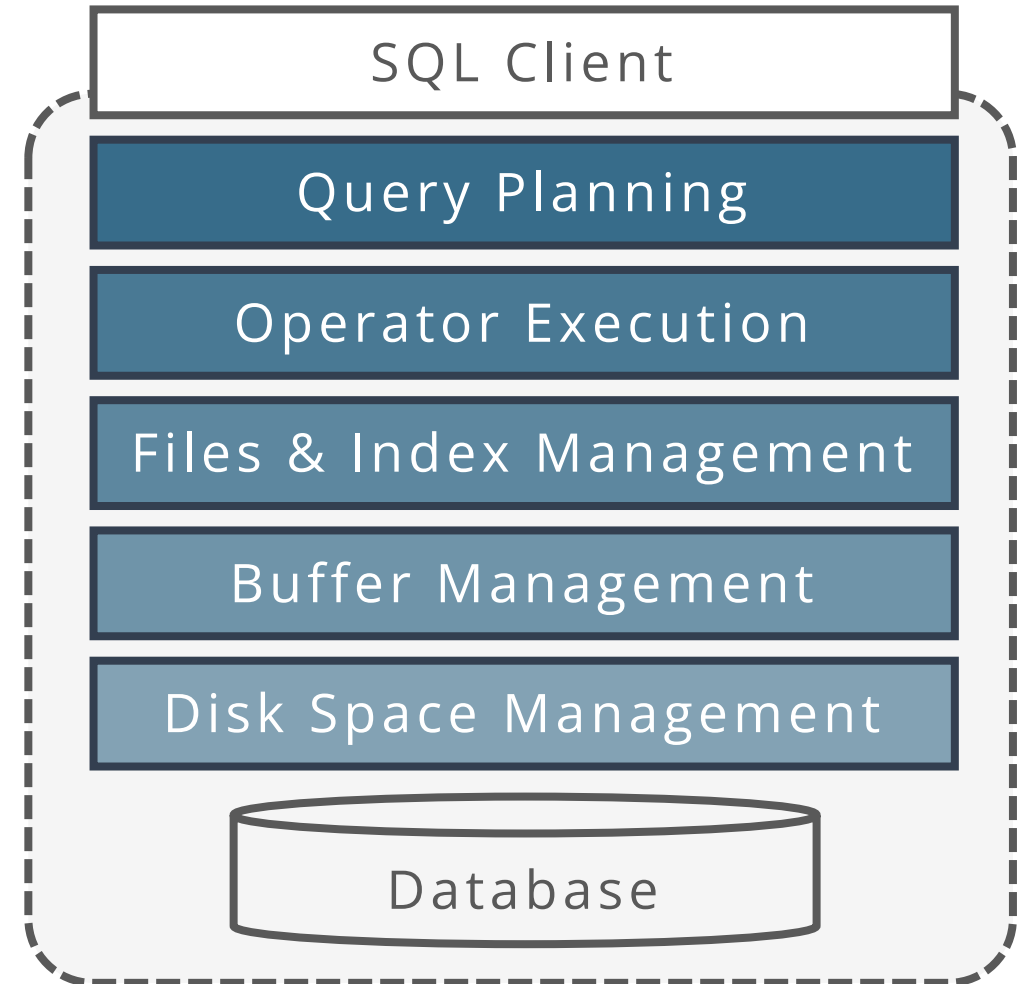
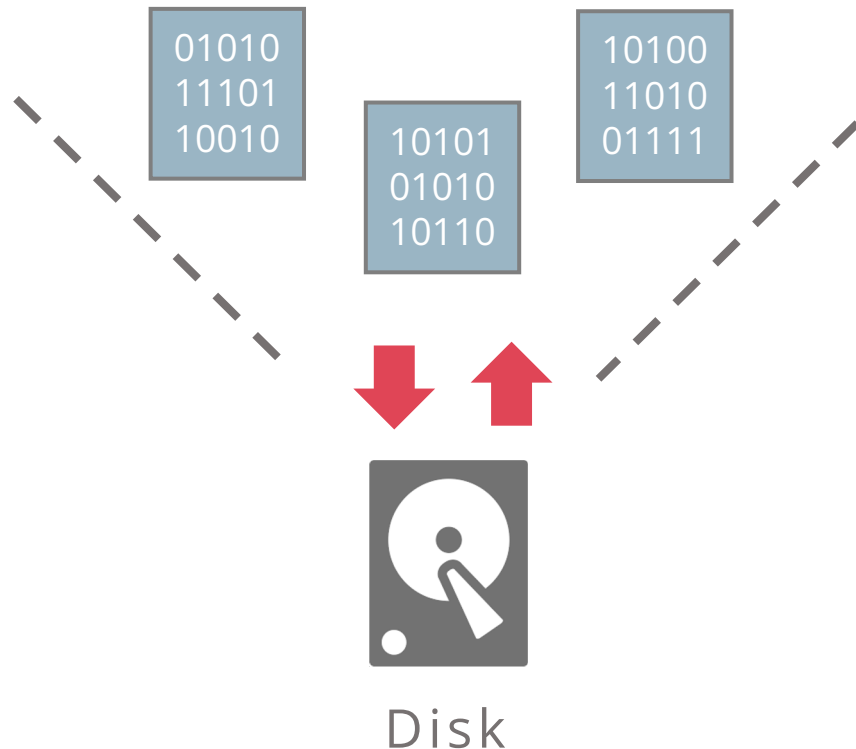
# DBMS: BUFFER MANAGEMENT

Transfer data between disk and memory



# DBMS: DISK SPACE MANAGEMENT

Translate page requests into reading & writing physical bytes on devices



# ARCHITECTURE OF A DBMS

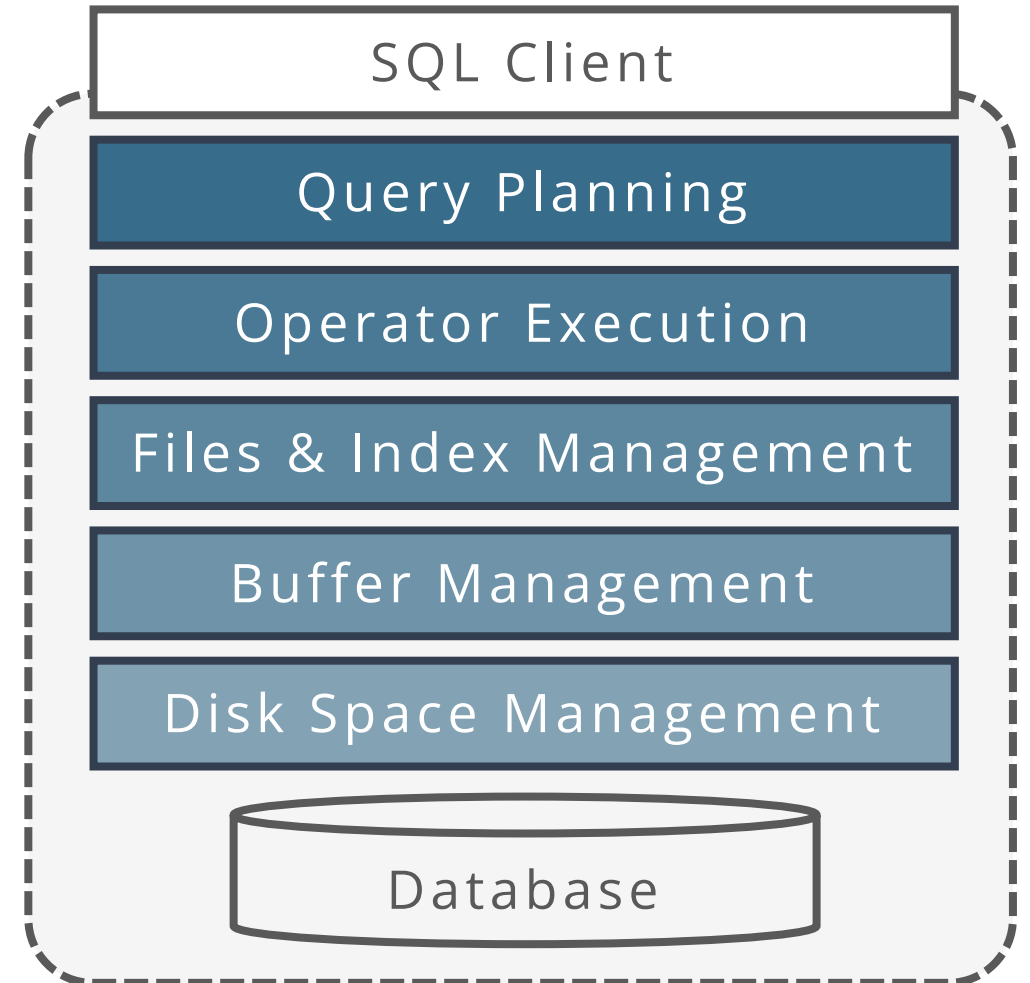
Organised in layers

Each layer abstracts the layer below

Manage complexity

Performance assumptions

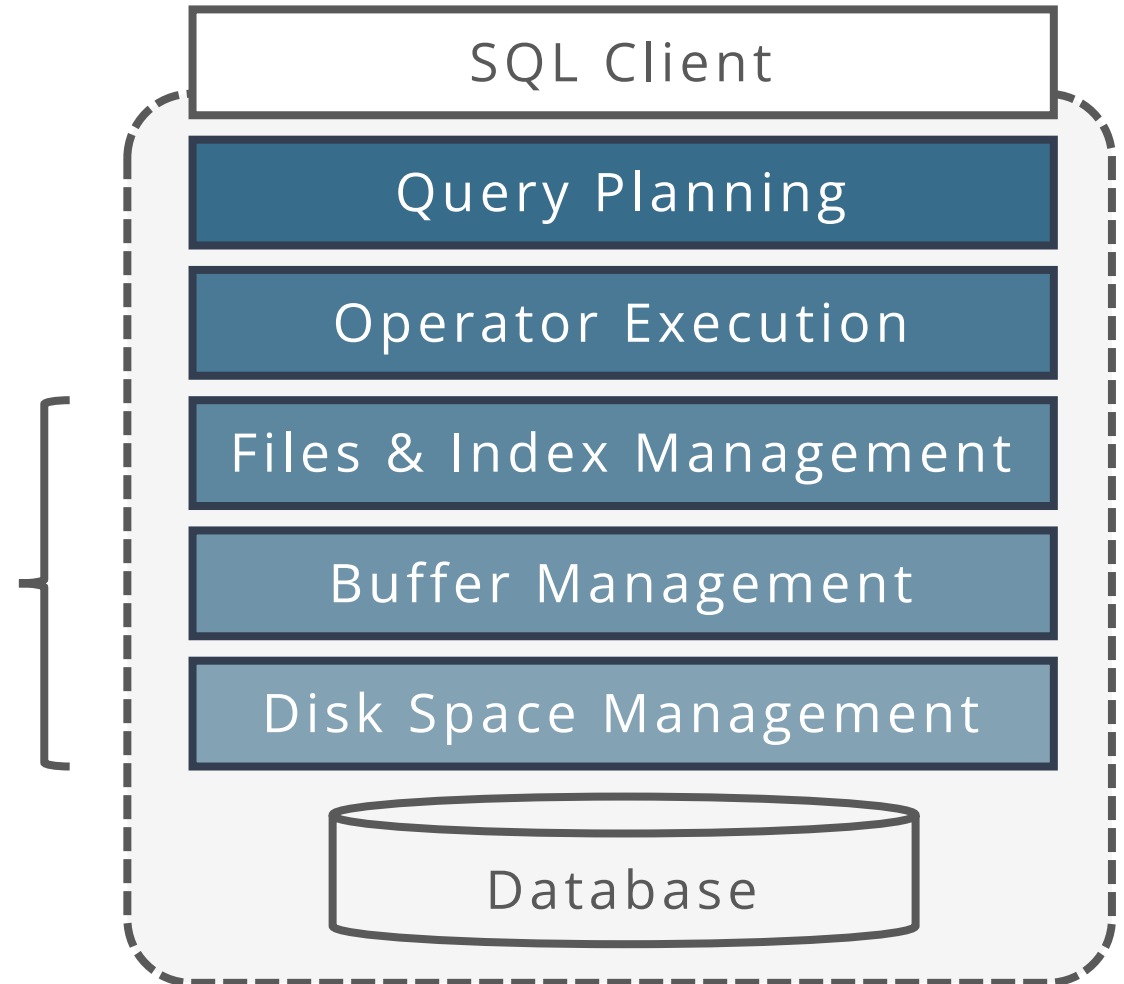
Example of good systems design





# DBMS: CONCURRENCY & RECOVERY

Two cross-cutting modules related to storage and memory management



# OUTLINE

Storage Media

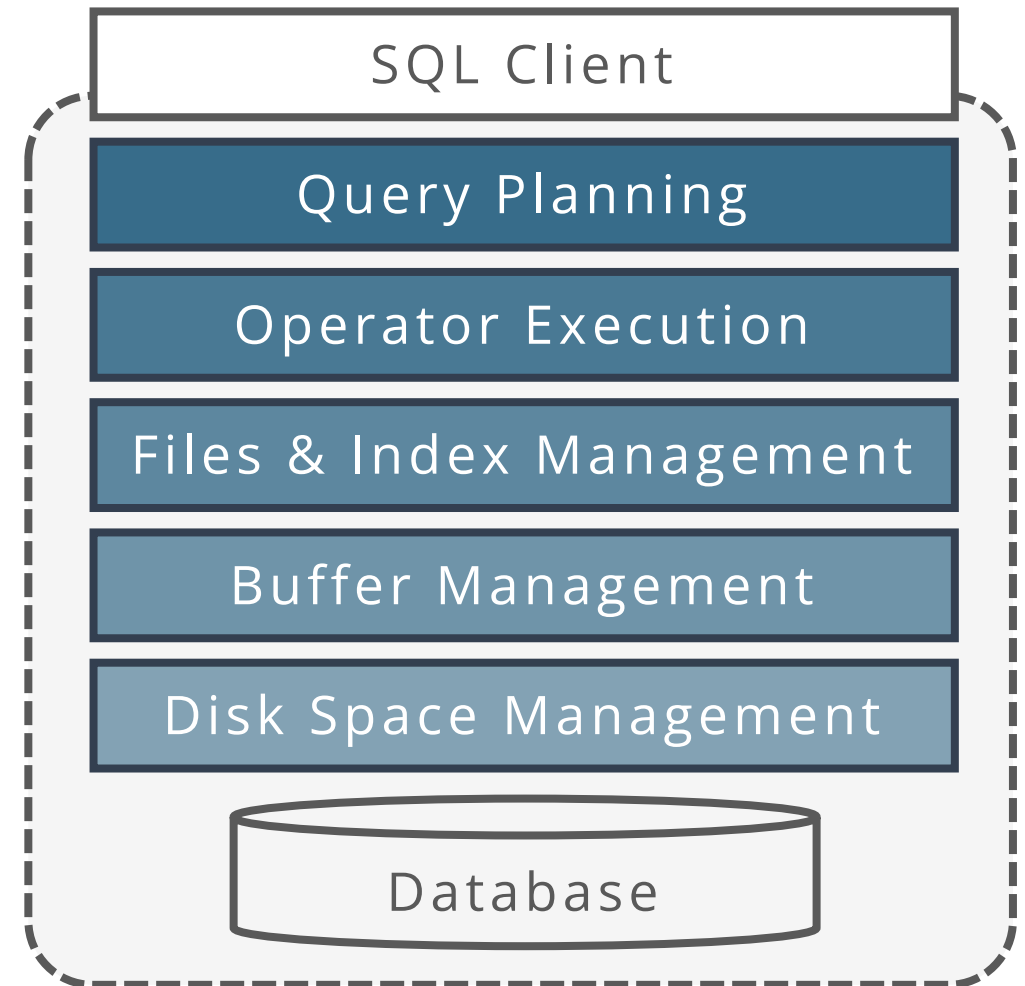
Disk Space Management

Buffer Management

File Layout

Page Layout

Record Layout



# DISK-ORIENTED ARCHITECTURE

Most database systems are designed for non-volatile **disk storage**\*

The primary location of the database is on disks (HDD and/or SSD)

Data processing happens in volatile **main memory**

The DBMS responsible for moving data between disk and main memory

## Major implications

Data stored on disk is not byte addressable. Instead, an API:

READ: transfer “page” of data from disk to RAM

WRITE: transfer “page” of data from RAM to disk

Disk reads & writes are **very, very slow!** ⇒ Must plan carefully!

\* Volatile storage only maintains its data while the device is powered

# WHY NOT STORE ALL IN MAIN MEMORY?

## Costs too much

Cost of 1TB storage (2020): 50\$ for HDD, 200\$ for SSD, 6000\$ for RAM

High-end databases today in the petabyte range!

Roughly 60% of the cost of a production system is in the disks

## Main memory is volatile

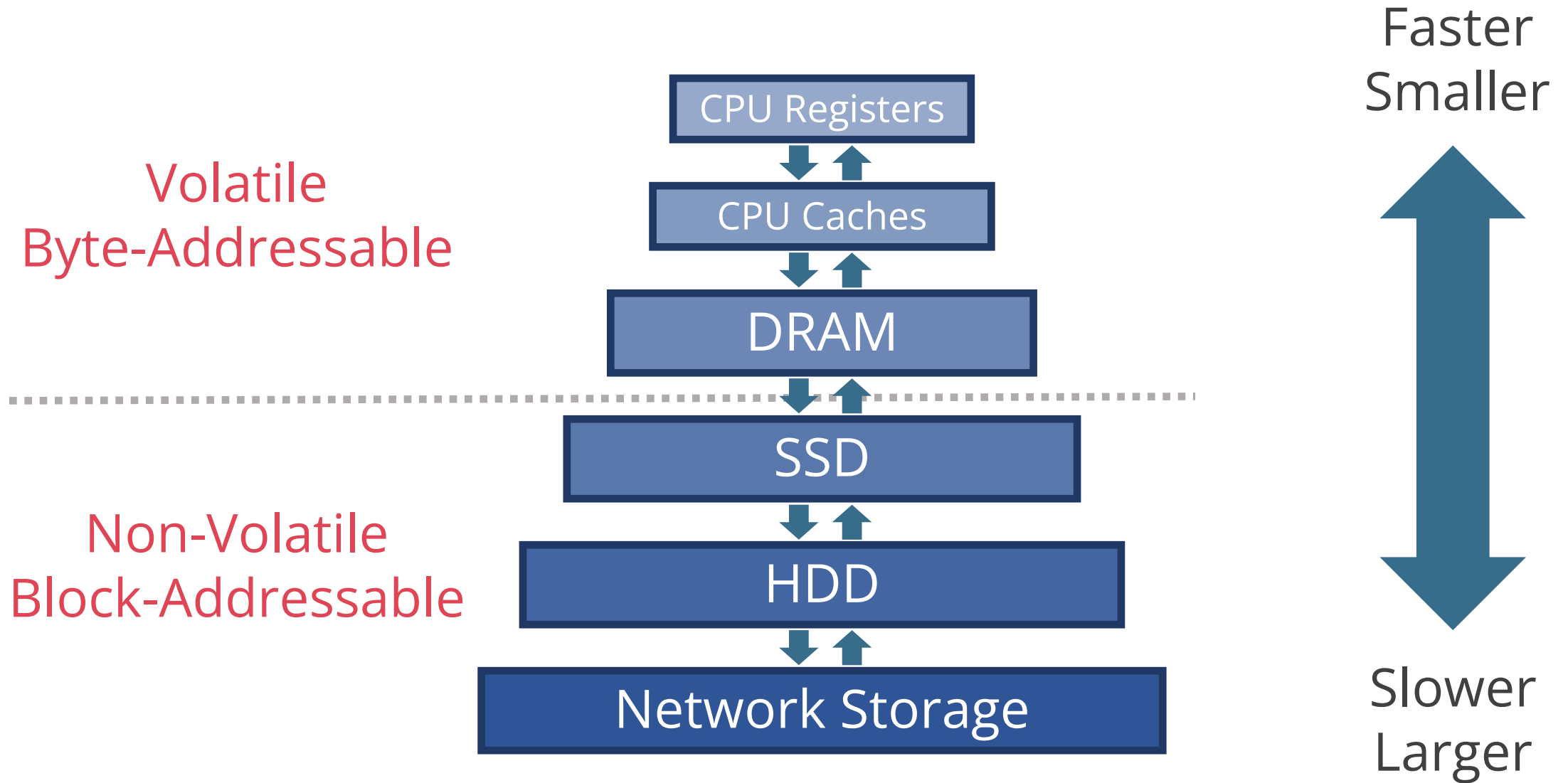
Obviously important if DB stops/crashes. We want data to be saved!

Some specialised systems do store entire databases in main memory

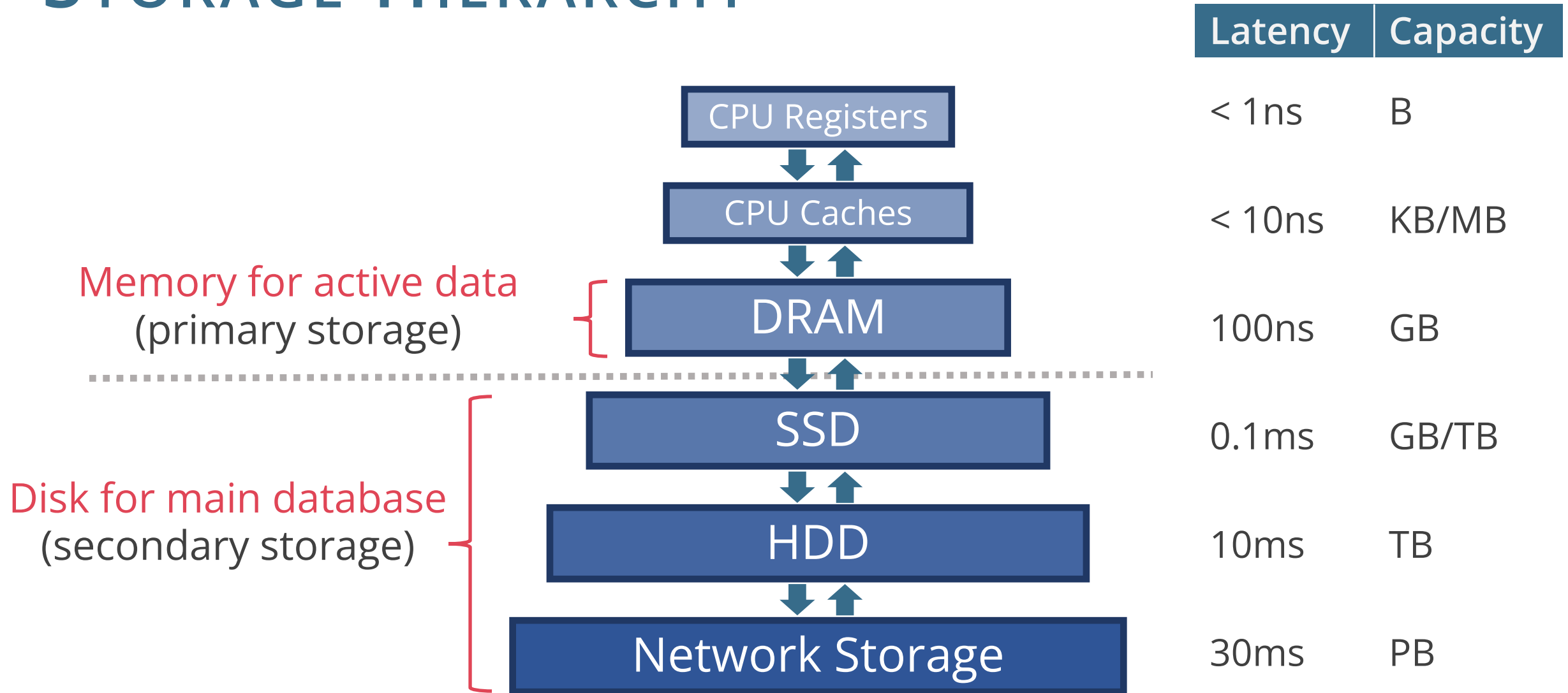
Faster than disk-oriented but with much higher cost/GB

Suitable for small databases

# STORAGE HIERARCHY



# STORAGE HIERARCHY



# ANATOMY OF A DISK

**Platters rotate** (say 15000 rpm)

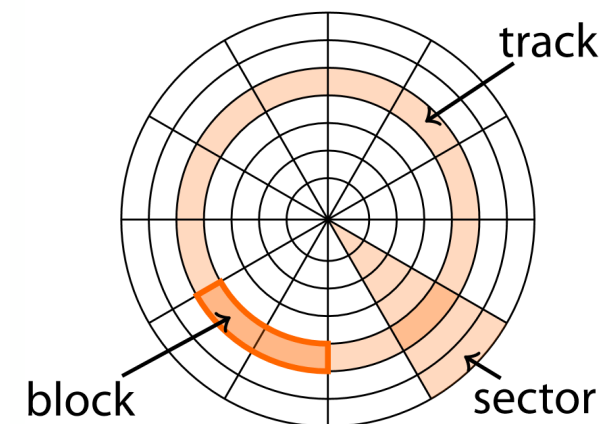
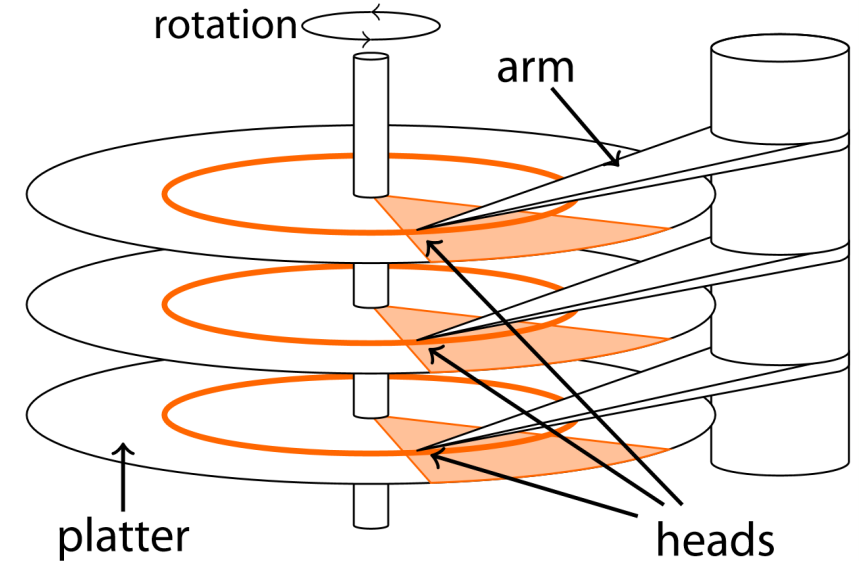
**Disk arm moves in or out to position disk heads on a desired track**

Tracks under heads make a “cylinder”

Only one head reads/writes at any one time

**Block size is a multiple of (fixed) sector size**

Sector = minimum storage unit (512B or 4KB)



[Video on how disk drives work](#)

# ACCESSING A DISK PAGE

Data is stored and retrieved in units called **disk blocks**

Block size is determined by the filesystem (usually 4KB, sometimes up to 64KB)

Unlike RAM, time to retrieve a block depends on its location

Time to access (read/write) a disk block:

**Seek time:** moving disk arm to position disk heads on track

**Rotational delay:** waiting for target block to rotate under a head

**Transfer time:** actually moving data to/from disk surface





## Seagate Cheetah 15K.7

4 disks, 8 heads, avg. 512 KB/track, 600GB capacity

rotational speed: 15 000 rpm

average seek time: 3.4 ms

transfer rate  $\approx$  163 MB/s

### Access time to read one block of size 8KB

Average seek time		3.40 ms
Average rotational delay	$1/2 \cdot 1/15000 \text{ min}$	2.00 ms
Transfer time	8KB / 163 MB/s	0.05 ms
<b>Total access time</b>		<b>5.45 ms</b>

**Seek time and rotational delay dominate!**

# SEQUENTIAL VS. RANDOM ACCESS

What about accessing 1000 blocks of size 8 KB

Random:  $1000 \cdot 5.45 \text{ ms} = 5.45 \text{ s}$

Sequential:  $3.4 \text{ ms} + 2 \text{ ms} + 1000 \cdot 0.05 \text{ ms} \approx 55 \text{ ms}$

tracks store only 512KB  $\Rightarrow$  some additional ( $< 5 \text{ ms}$ ) track-to-track seek time

**Sequential I/O orders of magnitude faster than random I/O**



**avoid random I/O at all cost**

# ARRANGING BLOCKS ON DISK

'Next' block concept:

sequential blocks on same track, followed by  
blocks on same cylinder, followed by  
blocks on adjacent cylinder

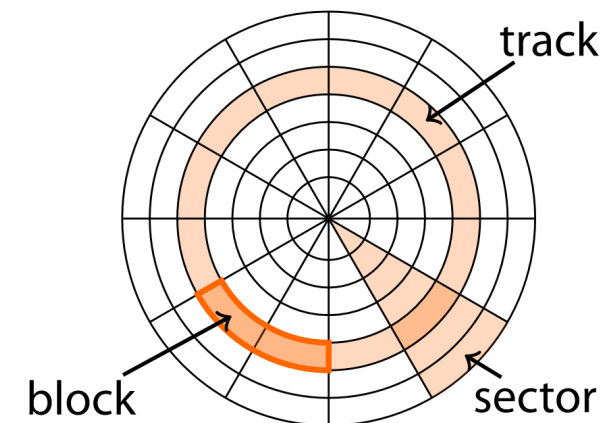
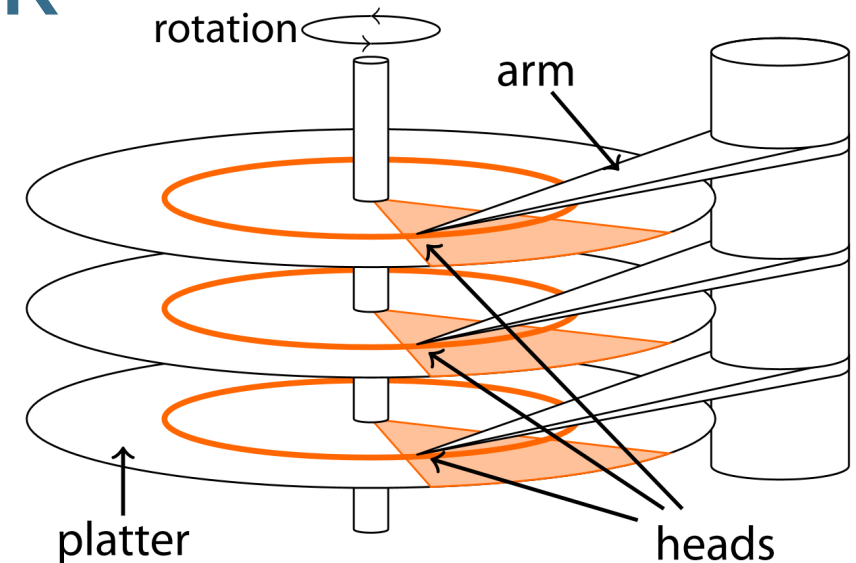
Arrange file pages sequentially by 'next' on disk

Minimize seek and rotational delay

For a **sequential scan**, pre-fetch several blocks at a time!

Reading large consecutive blocks

"Amortises" seek time and rotational delay



# SOLID STATE DRIVES

Alternative to conventional hard disks

Data accessed in pages, internally pages are organised into blocks

Fine-grain reads (4-8KB pages), coarse-grain writes (1-2MB blocks)

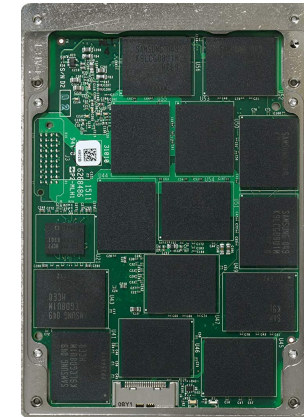
Issues in current generation (NAND)

**Write amplification:** Writing data in small pages causes erasing big blocks

**Limited endurance:** Only 2K-3K erasures before cell failure

**Wear levelling:** SSD controller needs to keep moving hot write units around

**Price:** SSD is 2-5x more expensive than HDD



# SOLID STATE DRIVES

Read is fast and predictable

Single read access time: 30  $\mu$ s

4KB random reads: ~500 MB/sec

Sequential reads: ~525 MB/sec

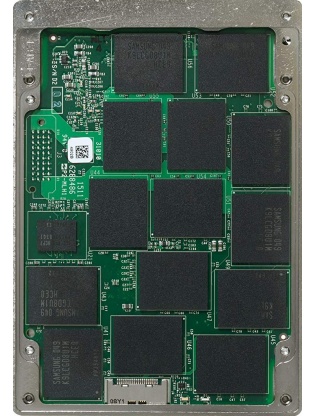
But write is not! Slower for random

Single write access time: 30  $\mu$ s

4KB random writes: ~120 MB/sec

Sequential writes: ~480 MB/sec

Random access **still slower** than sequential access



# SSD vs. HDD

SSD can achieve 1-10x the bandwidth (bytes/sec) of ideal HDD

Note: Ideal HDD spec numbers are hard to achieve

Expect 10-100x bandwidth for non-sequential reads

Locality matters for both

Reading/writing to “far away” blocks on HDD requires slow seek/rotation delay

Writing 2 “far away” blocks on SSD can require writing multiple much larger units

High-end flash drives are getting much better at this

And don't forget

SSD is 2-5x more expensive than HDD

# BOTTOM LINE

Very large DBs: relatively traditional

- Disk still offers the best cost/GB by a lot

- SSDs improve performance and performance variance

Smaller DB story is changing quickly

- SSDs win at the low end (modest DB sizes)

- Many interesting databases fit in RAM

Lots of change brewing on the HW storage tech side

- Non-volatile memory likely to affect the design of future systems

We will focus on traditional RAM and disk

# DATABASE STORAGE

Most DBMSs store data as one or more **files** on disk

Files consist of **pages** (loaded in memory), pages contain **records**

Data on disk is read & written in large chunks of sequential bytes

**Block** = Unit of transfer for disk read/write

**Page** = A common synonym for “block”

In some textbooks, “page” = a block-sized chunk of RAM

We will treat “block” and “page” as synonyms

**I/O operation** = read/write disk operation

**Sequential pages:** reading “next” page is fastest



# SYSTEM DESIGN GOALS

**Goal:** allow the DBMS to manage databases  $>$  available main memory

Disk reads/writes are expensive  $\Rightarrow$  must be managed carefully

Minimise disk I/O, maximise usage of data per I/O

## Spatial control

Where to write pages on disk

**Goal:** keep pages often used together as physically close as possible on disk

## Temporal control

When to read pages into memory and when to write them to disk

**Goal:** minimise the number of CPU stalls from having to read data from disk

# DISK SPACE MANAGEMENT

Lowest layer of DBMS, manages space on disk

Map pages to locations on disk

Load pages from disk to memory

Save pages back to disk

Introduces the concept of a **page**

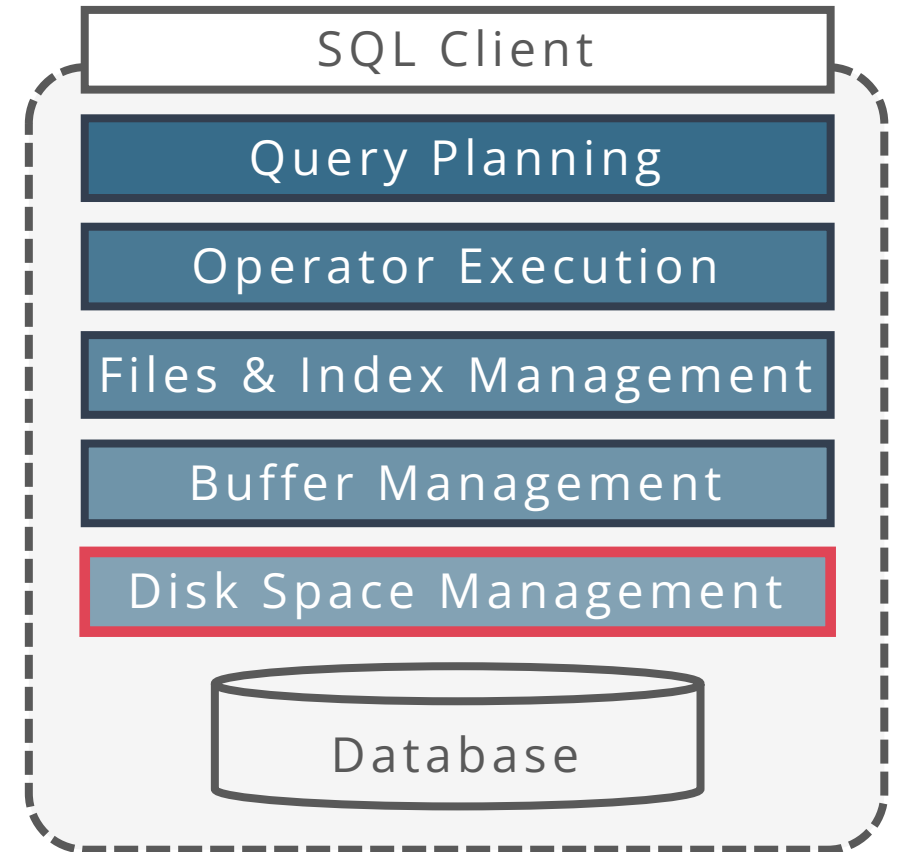
Typical page size: 4 – 64KB (a multiple of 4KB)

Each page has a unique identifier: **page ID**

Higher levels call upon this layer to:

Allocate/de-allocate a page

Read/write a page



# DISK SPACE MANAGEMENT: PAGE REQUESTS

Disk space manager can get requests for a **sequence of pages**

E.g., when higher levels execute a scan operator on a relation

Such requests are best satisfied by pages stored sequentially on disk

Physical details hidden from higher levels of system

Higher levels may “safely” assume **Next Page** is fast, so they will simply expect sequential runs of pages to be quick to scan

Disk space manager aims to intelligently lay out data on disk

to meet the performance expectation of higher levels as best as possible

# DISK SPACE MANAGEMENT: IMPLEMENTATION

## Using local filesystem (FS)

Allocate one large “contiguous” file on an empty disk

Rely on OS and FS that sequential pages in this file are physically contiguous on disk

A logical database “file” may span multiple FS files on multiple disks/machines

Disk space manager maintains a **mapping** from page IDs to physical locations

physical location = filename + offset within that file

The OS and other apps know nothing about the contents of these files

Only the DBMS knows how to decipher their contents

Early DBMSs in the 1980s used custom ‘filesystems’ on raw storage

# SUMMARY

## Magnetic disk and flash storage

Random access vs. sequential access (10x)

Physical data placement is important

## Disk space management

Exposes data as a collection of pages

Pages: block-level organisation of bytes on disk

API to read/write pages to disk

Provides “next” locality

Abstracts device and file system details

