

CS 4440 A

# Emerging Database Technologies

---

Lecture 10

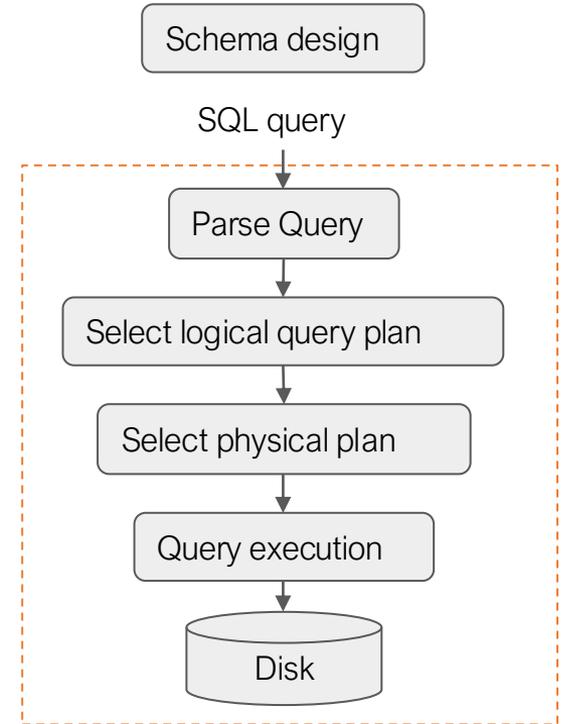
02/18/26

# Announcements

- Assignment 1 grades released
- Office hour change (starting from next week):
  - Instructor: Wednesday 2-3PM

# Next Part: Query Processing and Optimization

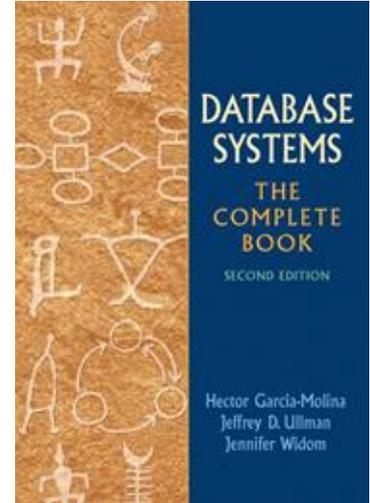
- Hardware and file system structure
- Indexing and hashing
- Query optimization



# Reading Materials

Database Systems: The Complete Book (2nd edition)

- Chapter 13: Secondary Storage Management



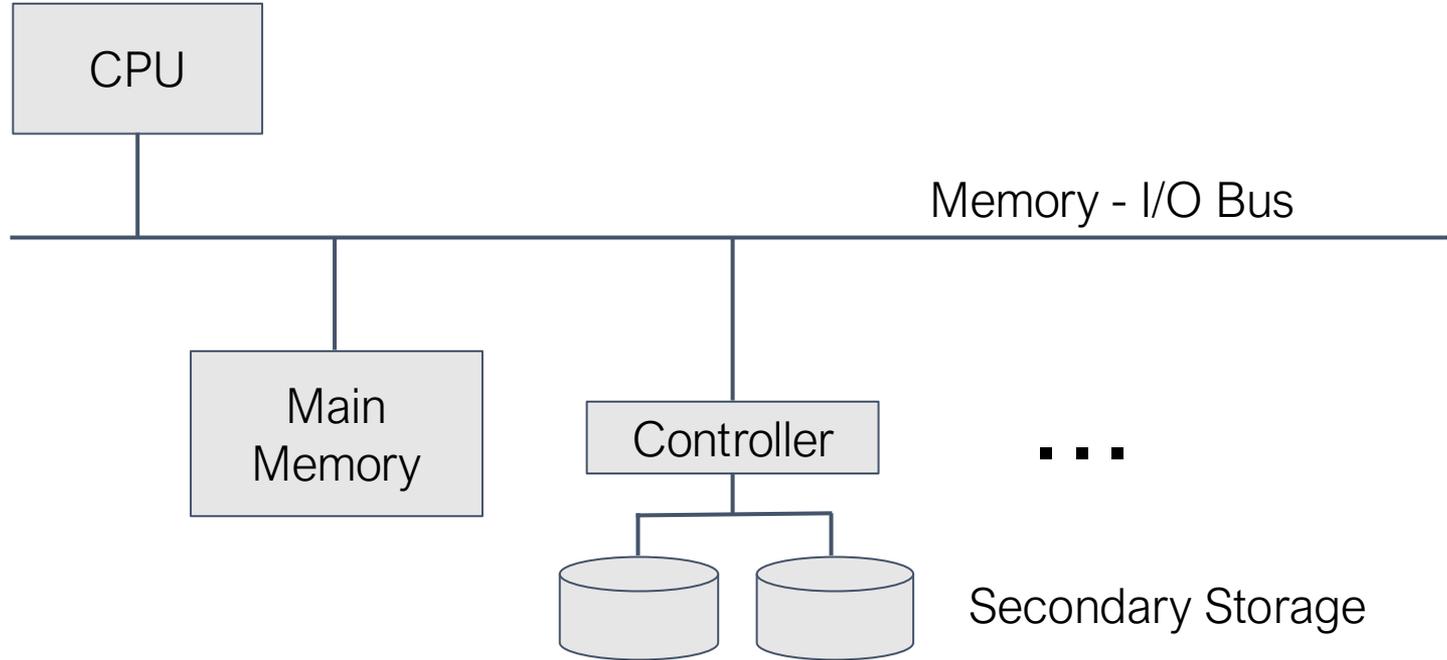
Acknowledgement: The following slides have been adapted from EE477 (Database and Big Data Systems) taught by Steven Whang and CS245 (Principles of Data-Intensive Systems) taught by Matei Zaharia.

# Agenda

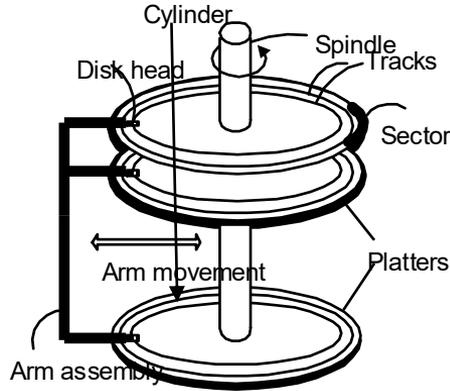
1. Storage hardware
2. Arranging records on disks
3. Collection Storage

# 1. Storage Hardware

# Typical computer system (Von Neumann architecture)



# High-level: Disk vs. Main Memory



## Disk:

- **Fast:** sequential block access
  - Read a blocks (not byte) at a time, so sequential access is cheaper than random
  - Disk read / writes are expensive
- **Durable:** We will assume that once on disk, data is safe!
- **Cheap**

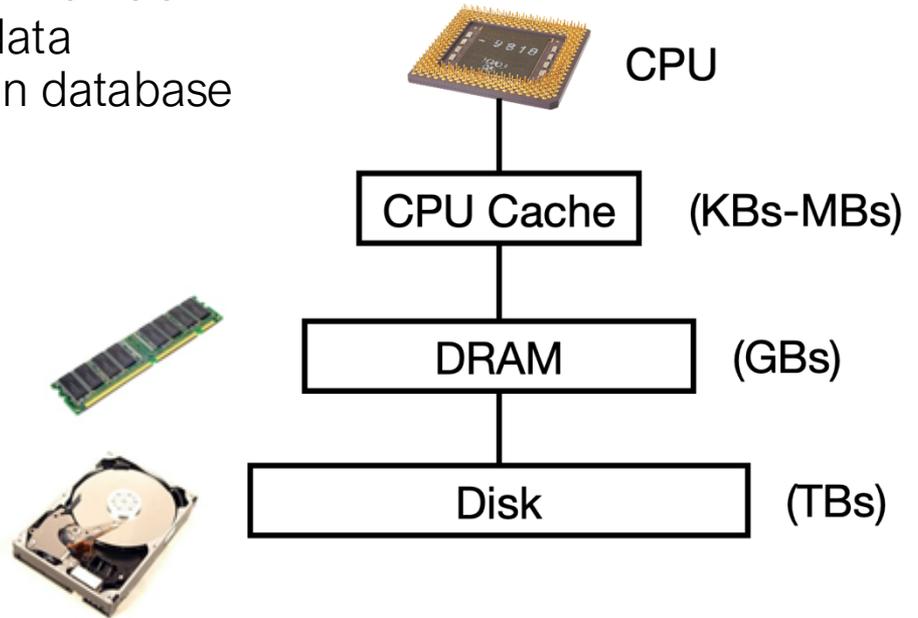
## Random Access Memory (RAM) or Main Memory:

- **Fast:** Random access, byte addressable
  - ~10x faster for sequential access
  - ~100,000x faster for random access!
- **Volatile:** Data can be lost if e.g. crash occurs, power goes out, etc!
- **Expensive:** For \$100, get 16GB of RAM vs. 2TB of disk!

# Storage Hierarchies

Typically **cache** frequently accessed data on faster storage to improve performance

- Main memory stores current data
- Secondary storage stores main database



# Numbers everyone should know

"Numbers Everyone Should Know" from Jeff Dean. [Slides #1](#), [Slides #2](#)

L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	
Mutex lock/unlock	100 ns	
Main memory reference	100 ns	
Compress 1K bytes with Zippy	10,000 ns	0.01 ms
Send 1K bytes over 1 Gbps network	10,000 ns	0.01 ms
Read 1 MB sequentially from memory	250,000 ns	0.25 ms
Round trip within same datacenter	500,000 ns	0.5 ms
Disk seek	10,000,000 ns	10 ms
Read 1 MB sequentially from network	10,000,000 ns	10 ms
Read 1 MB sequentially from disk	30,000,000 ns	30 ms
Send packet CA->Netherlands->CA	150,000,000 ns	150 ms

Where

- 1 ns =  $10^{-9}$  seconds
- 1 ms =  $10^{-3}$  seconds



by Jeff Dean

# Jim Gray's storage latency analogy: how far is the data?



Turing Award,  
1998

$10^9$	Tape			Andromeda	2,000 years
$10^6$	Disk			Pluto	2 years
100	Memory			Columbus, GA	1.5 hours
10	L3 cache			This building	10 min
2	On chip cache			This room	
1	Registers			My head	1 min
(ns)					

# Sizing Storage Tiers

When should we cache data in DRAM vs storing it on disks?

Can determine based on workload & cost



“The 5 Minute Rule for Trading Memory  
Accesses for Disc Accesses”

Jim Gray & Franco Putzolu

May 1985

# The five minute rule

“Pages referenced every 5 minutes should be memory resident (1987)”

*BreakEvenReferenceInterval (seconds) =*

$$\frac{\text{PagesPerMBofRAM}}{\text{AccessPerSecondPerDisk}} \times \frac{\text{PricePerDiskDrive}}{\text{PricePerMBofRAM}}$$

Technology ratio

Economic ratio

# The five minute rule

“Pages referenced every 5 minutes should be memory resident (1987)”

*BreakEvenReferenceInterval (seconds) =*

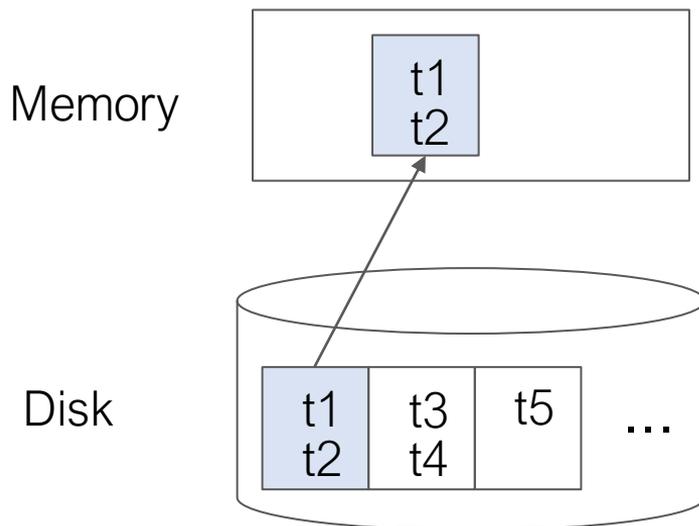
$$\frac{\text{PagesPerMBofRAM}}{\text{AccessPerSecondPerDisk}} \times \frac{\text{PricePerDiskDrive}}{\text{PricePerMBofRAM}}$$

Tier	1987	1997	2007	2017
DRAM–HDD	5m	5m	1.5h	4h
DRAM–SSD	-	-	15m	7m (r) / 24m (w)
SSD–HDD	-	-	2.25h	1d

Source: The Five-minute Rule Thirty Years Later and its Impact on the Storage Hierarchy

# I/O model of computation

- Time to read a block from disk  $\gg$  time to search a record within that block
- Algorithm time  $\approx$  Number of disk I/Os



# Speeding up disk access

The previous analysis was on **random accesses**

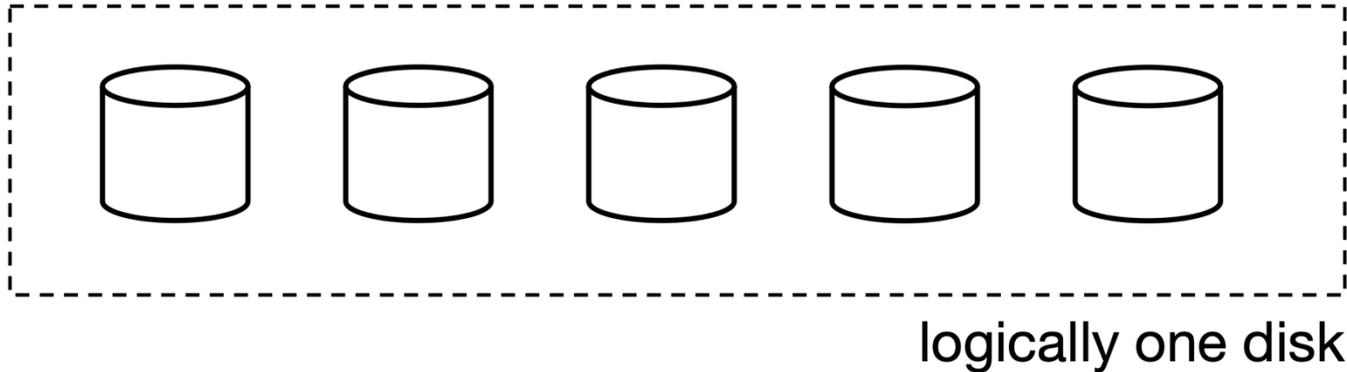
In general, sequential access is much faster than random accesses

There are several techniques for decreasing average disk access time

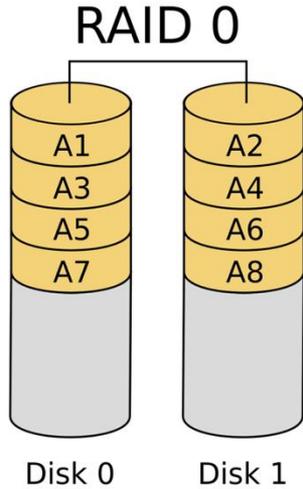
- RAID
- Prefetching

# RAID: Combining storage devices

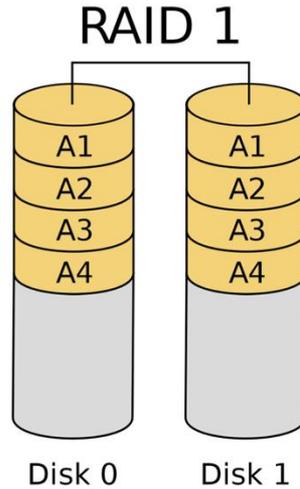
- **RAID:** redundant array of inexpensive disks
- Many flavors of “RAID”: striping, mirroring, etc to increase performance and reliability



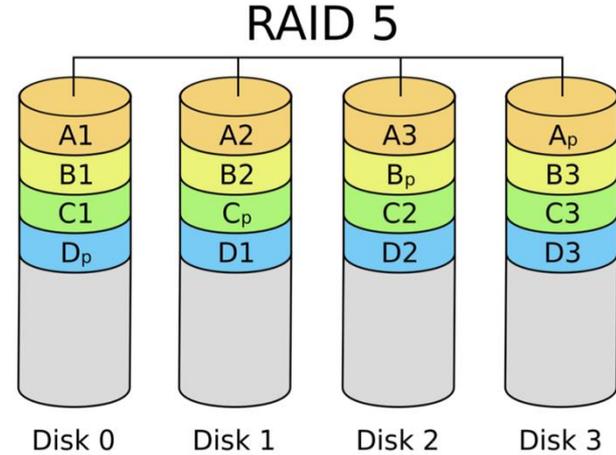
# Common RAID Levels



Striping across 2 disks: adds performance but not reliability



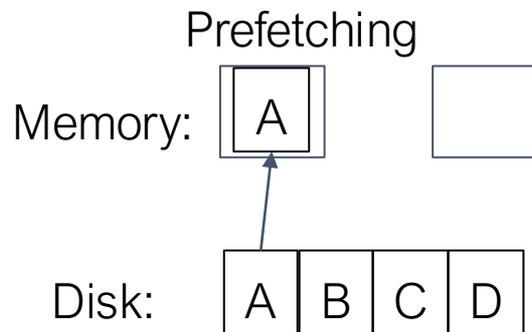
Mirroring across 2 disks: adds reliability but not performance (except for reads)



Striping + 1 parity disk: adds performance and reliability at lower storage cost

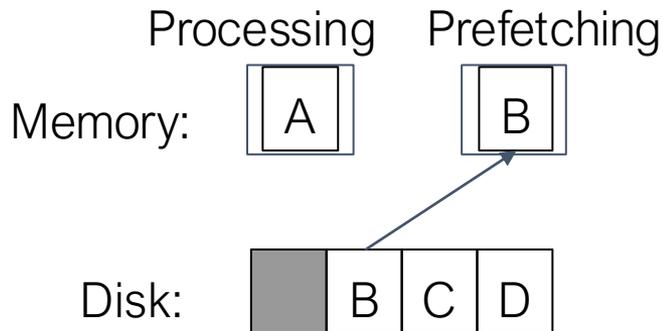
# Prefetching/Double buffering

- Predict block request order and load into memory before needed
- Reduces average block access time



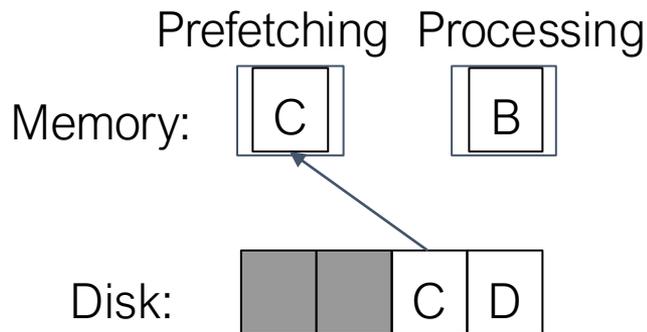
# Prefetching/Double buffering

- Predict block request order and load into memory before needed
- Reduces average block access time



# Prefetching/Double buffering

- Predict block request order and load into memory before needed
- Reduces average block access time



# In-class Exercise

Suppose

- $P$  = processing time / block
- $R$  = I/O time / block
- $N$  = number of blocks

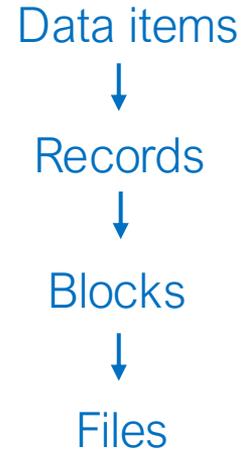
If  $P \geq R$ , what is the processing time of

- Single buffering
- Double buffering

## 2. Arranging Records on Disks

# File system structure

- Next let's look at how disks are used to store databases
- A tuple is represented by a record, which consists of consecutive bytes in a disk block



# Physical Representation of Data Items

Example data items that we want to store:

- Date
- Salary
- Name
- Picture

Data items

Records

Blocks

Files

What we have available: bytes



← 8 →  
bits

# Fixed length items

Integer: fixed # of bytes (e.g., 2 bytes)

e.g., 35 is 00000000 00100011

Floating-point: n-bit mantissa, m-bit exponent

Character: encode as integer (e.g. ASCII)

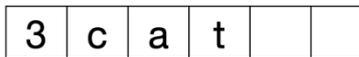
# Variable length items

String of characters:

- Null-terminated



- Length + data



- Fixed-length

Bag of bits:



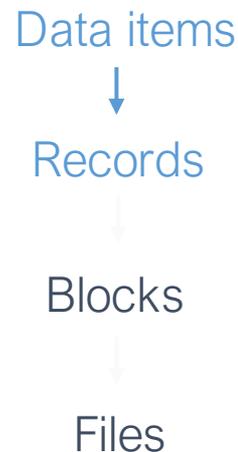
# Storing Records

Record (tuple): consecutive bytes in disk blocks

- e.g. employee record:
  - name field
  - salary field
  - date-of-hire field

Design choices:

- Fixed vs variable **length**
- Fixed vs variable **format**



# Fixed-format records

A **schema** for all records in table specifies:

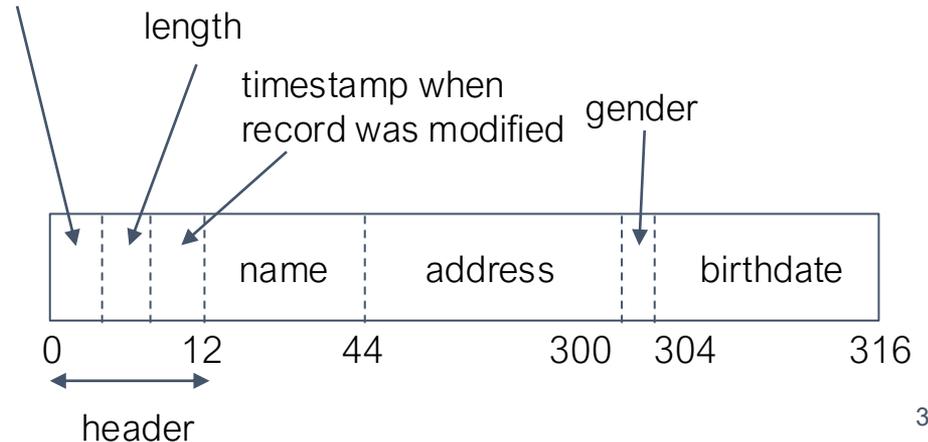
- # of fields
- type of each field
- order in record
- meaning of each field

# Fixed-length records

- header + fixed-length region of record's information
- It is common for field addresses to be multiples of 4 or 8 to align data for efficient reading/writing of main memory (a CPU accesses memory one word at a time)

```
CREATE TABLE MovieStar (  
  name          CHAR(30),  
  address       CHAR(255),  
  gender        CHAR(1),  
  birthdate    DATE  
);
```

pointer to schema for finding  
fields of the record



# Variable-length records

Some records may not have a fixed schema with a list of fixed-length fields

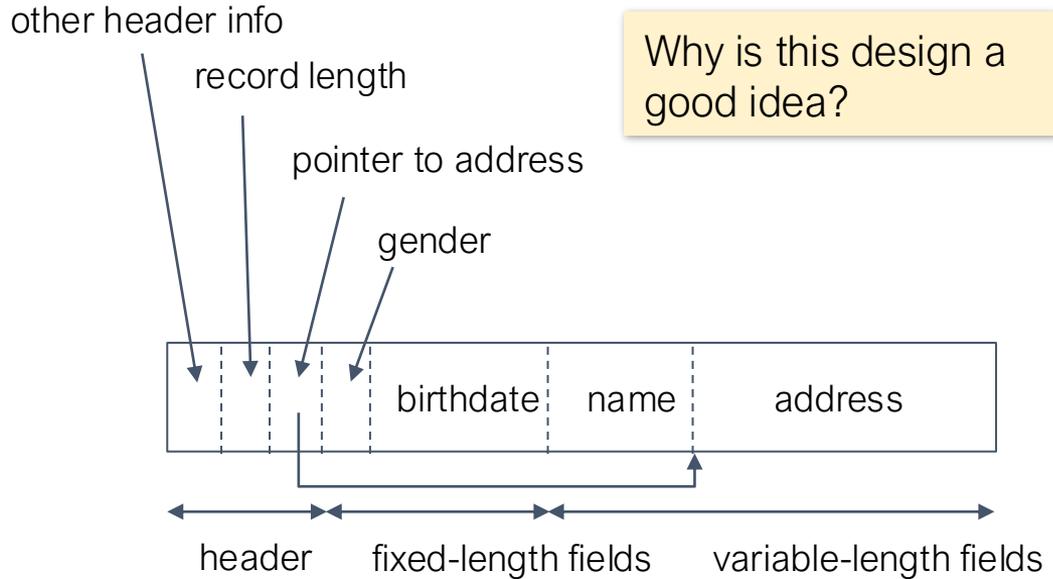
- e.g., VARCHAR
- other data models (e.g., semi-structured)

# Records with variable-length fields

- Put all fixed-length fields ahead of the variable-length fields
- Header stores pointers to be beginning of all variable-length fields other than the first

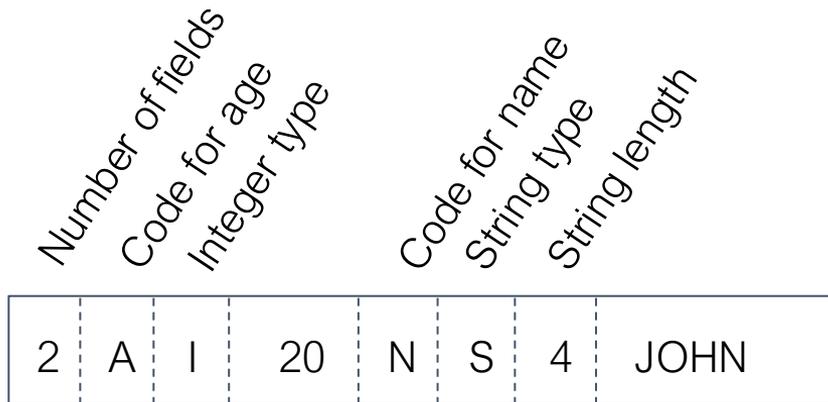
Why is this design a good idea?

```
CREATE TABLE MovieStar (  
  name          VARCHAR(30),  
  address       VARCHAR(100),  
  gender        CHAR(1),  
  birthdate     DATE  
);
```



# Variable-format records

- Records may not have a fixed schema (e.g., JSON)
- Use tagged fields to make record “self-describing”



# Discussion

What are pros and cons of variable format vs fixed format?

Evolving formats

“Sparse” records

Repeating fields

But many waste space...

# 3. Collection Storage

# Collection Storage Questions

How do we place data items and records for efficient access?

- Locality
- Searchability

How do we physical encode records in blocks and files?



# Place Data for Efficient Access

Locality: which items are accessed together

- When you read one field of a record, you're likely to read other fields of the same record
- When you read one field of record 1, you're likely to read the same field of record 2

Searchability: quickly find relevant records

- E.g. sorting the file lets you do binary search

# Locality Example: Row Stores vs Column Stores

## Row Store

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

Fields stored contiguously  
in one file

## Column Store

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

Each column in a different file

# Locality Example: Row Stores vs Column Stores

## Row Store

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

Fields stored contiguously  
in one file

## Column Store

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

Each column in a different file

Accessing all fields of one record: 1 random I/O for row, 3 for column

# Locality Example: Row Stores vs Column Stores

## Row Store

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

Fields stored contiguously  
in one file

## Column Store

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

Each column in a different file

Accessing one field of all records: 3x less I/O for column store

# Can We Have Hybrids Between Row & Column?

Yes! For example, colocated column groups:

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

File 1

File 2: age & state

Which works better for transactional workload, row vs column store?

Helpful if age & state are frequently co-accessed

# Improving Searchability: Ordering

**Ordering** the data by a field will give:

- Smaller I/Os if queries tend to read data with nearby values of the field (e.g. time ranges)
- Option to accelerate search via an ordered index (e.g., B+-tree), binary search, etc

What's the downside of having an ordering?

# Improving Searchability: Partitions

Place data into buckets based on a field  
“coarse-grained” searchability

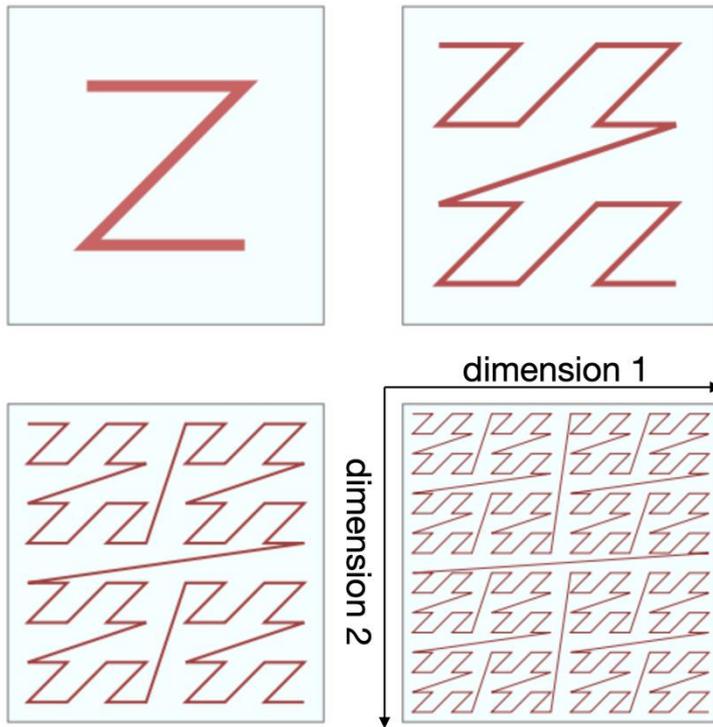
E.g. Hive table storage over a filesystem:

```
/my_table/date=20190101/file1.parquet  
                               /file2.parquet  
  /date=20190102/file1.parquet  
                               /file2.parquet  
  /date=20190103/file1.parquet  
                               ...
```

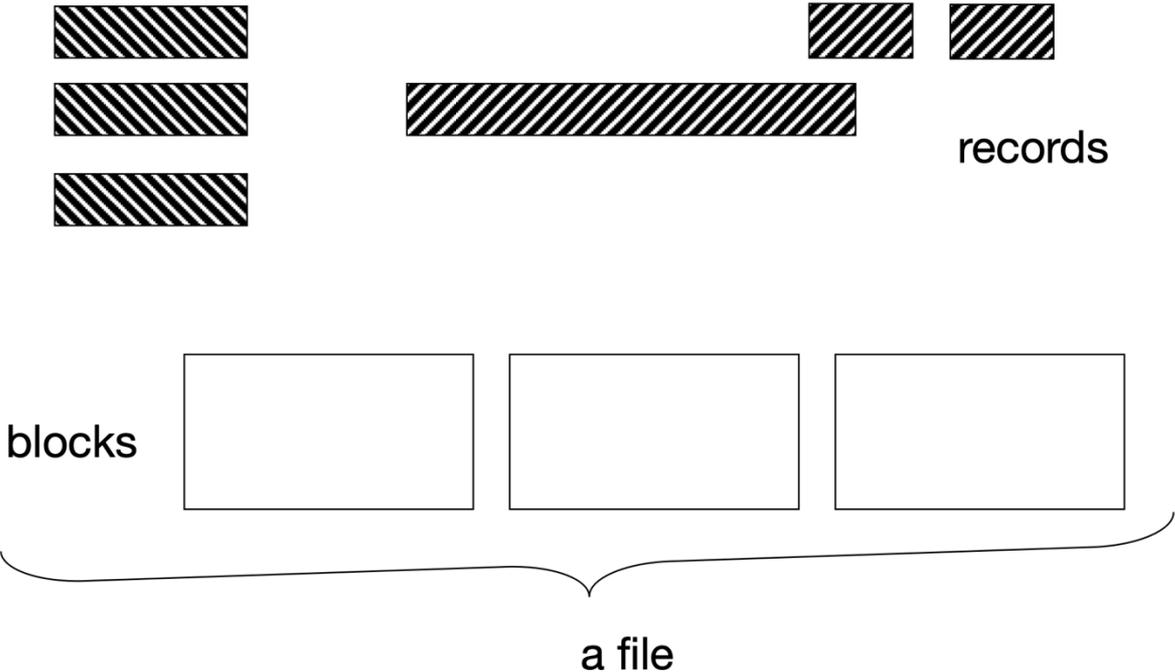
# Can We Have Searchability on Multiple Fields at Once?

Yes! Many possible ways:

- 1) Multiple partition or sort keys (e.g., partition by date, then sort by userID)
- 2) Interleaved orderings such as Z-ordering



# How Do We Encode Records into Blocks & Files?



# Storing records into blocks

Records are stored in blocks, which are moved into main memory.

Several design choices:

- (1) how to separate records
- (2) spanned vs. unspanned
- (3) indirection

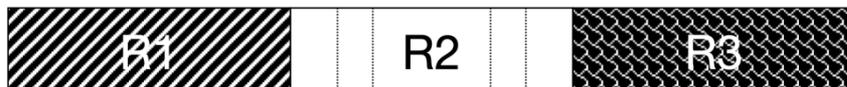
# (1) Separating Records

(a) no need to separate - fixed size records.

(b) special marker

(c) give record lengths (or offsets)

- within each record
- in block header



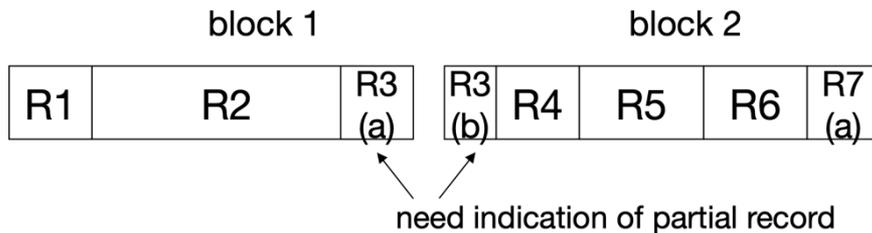
Block

## (2) Spanned vs Unspanned

Unspanned: records must be within one block



Spanned:



## (3) Indirection

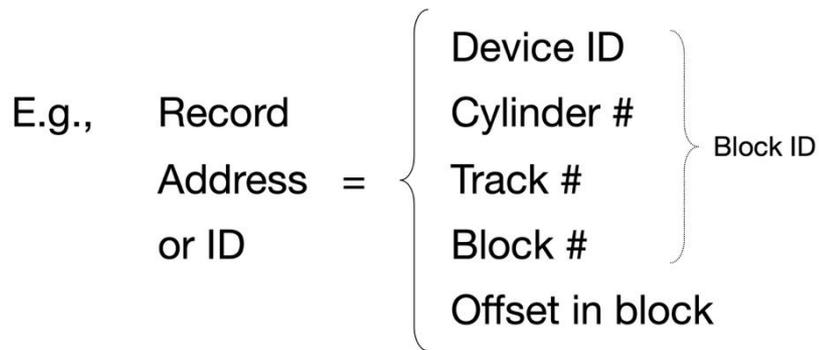
How does one refer to other records?



Many options: physical vs indirect

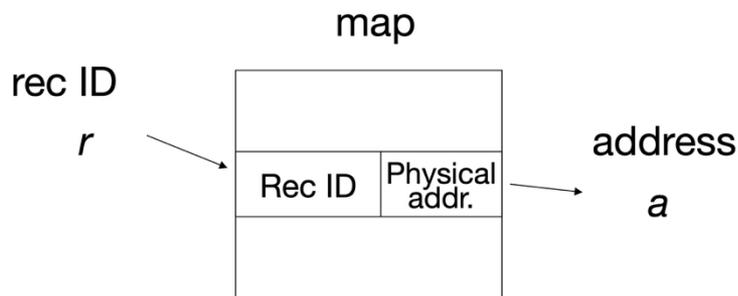
# (3) Indirection

## Purely Physical



## Fully Indirect

E.g., Record ID is arbitrary bit string



### Tradeoff:

Flexibility to move records <> cost of indirection

# Inserting Records

**Easy case:** records not ordered

- Insert record at end of file or in a free space
- Harder if records are variable-length

**Hard case:** records are ordered

- If free space close by, not too bad...
- Otherwise, use an overflow area and reorganize the file periodically

# Deleting Records

Immediately reclaim space

OR

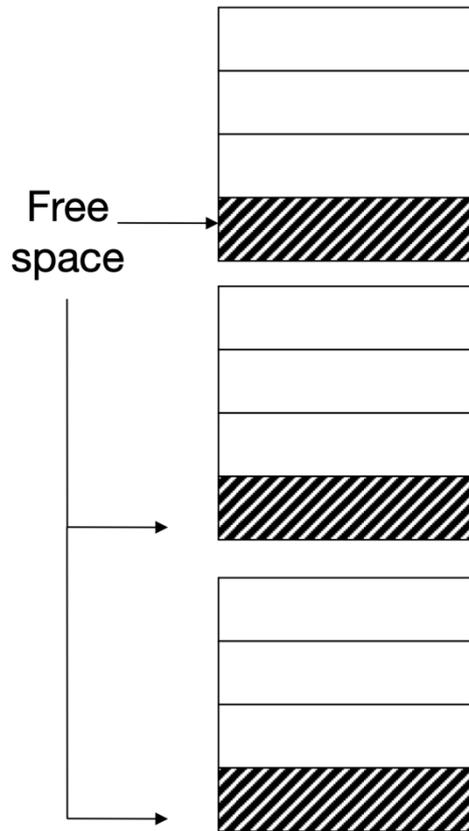
Mark deleted

- And keep track of freed spaces for later use

# Interesting Problems

How much free space to leave in each block, track, cylinder, etc?

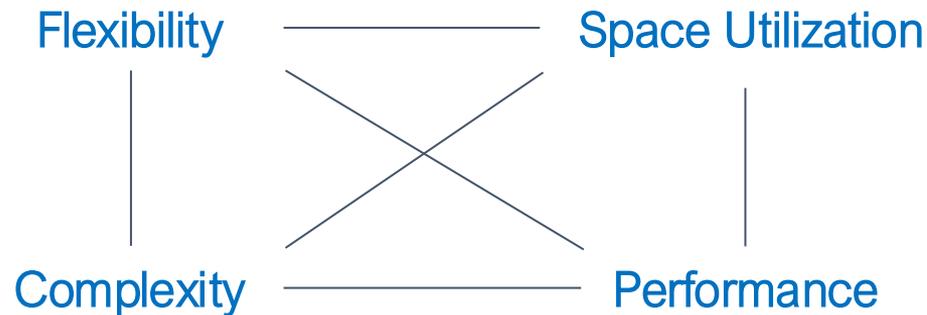
How often to reorganize file + merge overflow?



# Summary

Many ways to store data on disk!

Key tradeoffs:



# To Evaluate a Strategy, Compare:

Space used for expected data

Expected time to

- fetch record given key
- read whole file
- insert record
- delete record
- update record
- reorganize file
- ...