Welcome to Data C88C!

Lecture 20: SQL
Tuesday, July 29th, 2025
Week 6
Summer 2025
Instructor: Eric Kim (ekim555@berkeley.edu)

Announcements

- Midterm regrades: due this Friday
- August 1st: Change Grade Option deadline

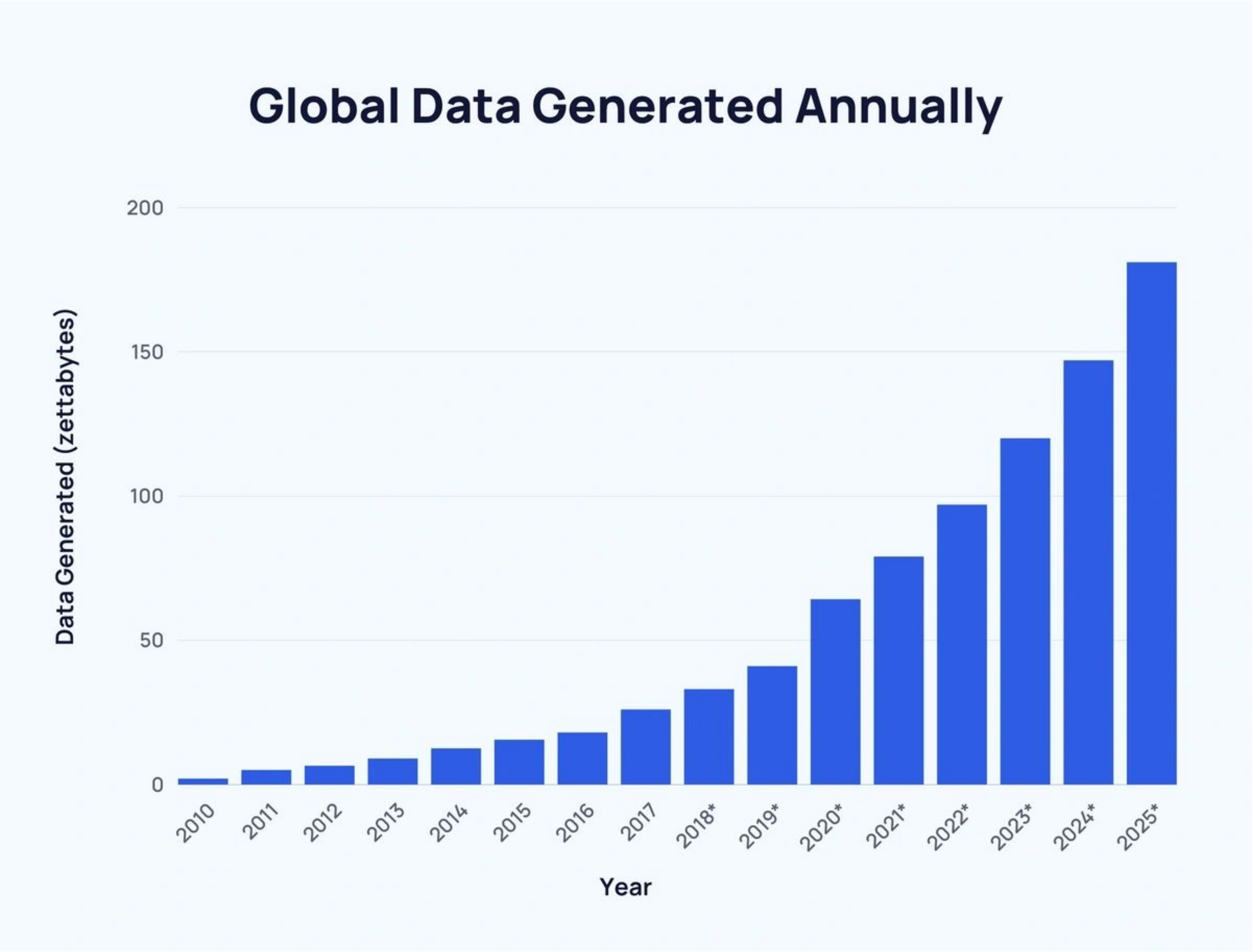
Lecture Overview

- Databases
- SQL

Databases

"Big Data" in today's world (2025)

- "In fact, it is estimated that **90% of the world's data was generated in the last two years alone**. In the space of 13 years, this figure has **increased by an estimated 74x** from just 2 zettabytes in 2010." [1]
- Examples:
  - Media (photos/videos)
    - Streaming (eg Netflix) and social media
  - User engagement data ("big tech")
  - Surveillance data
    - either state-sponsored, or companies tracking your shopping and browsing history



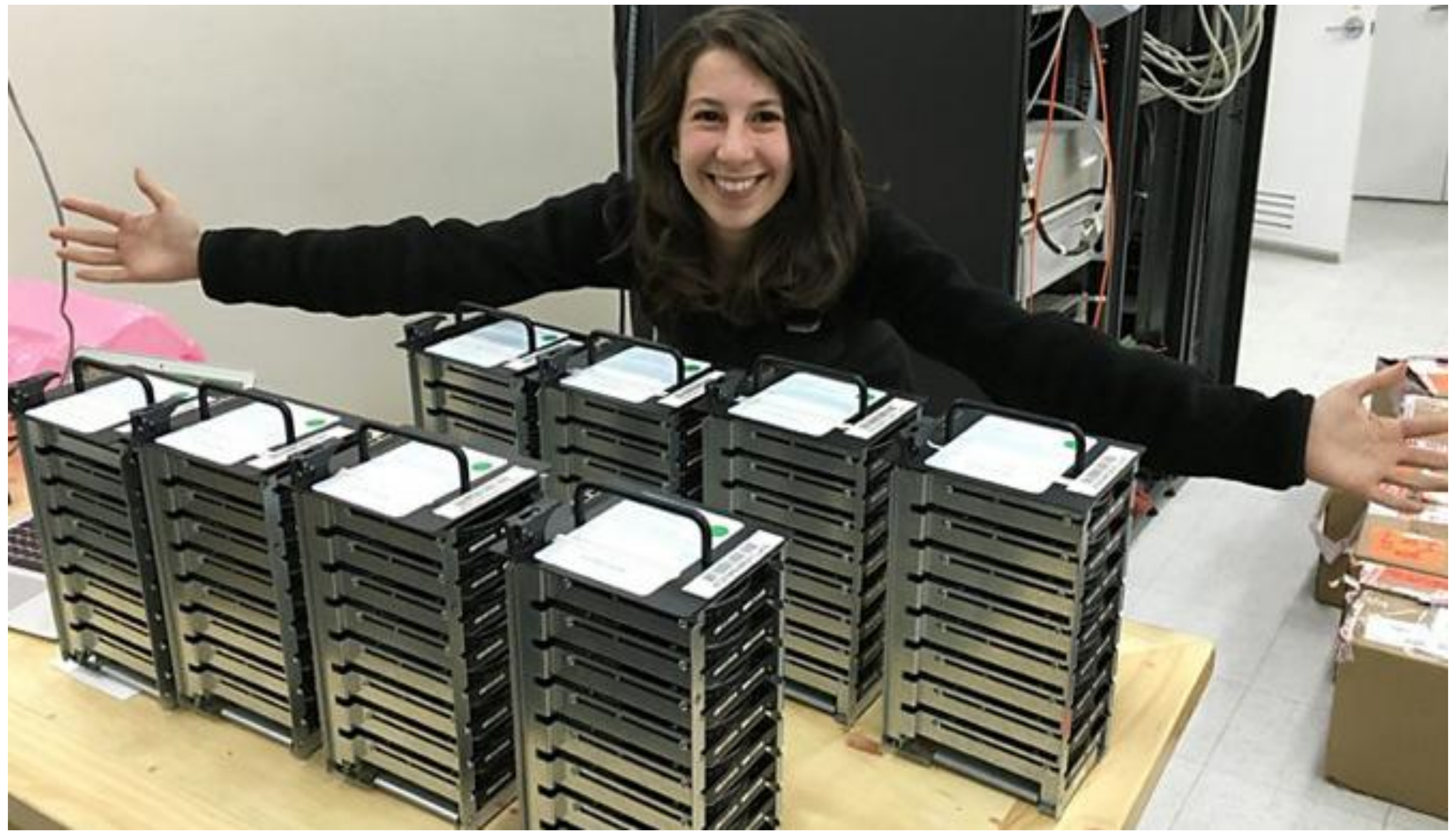
Zettabyte is:

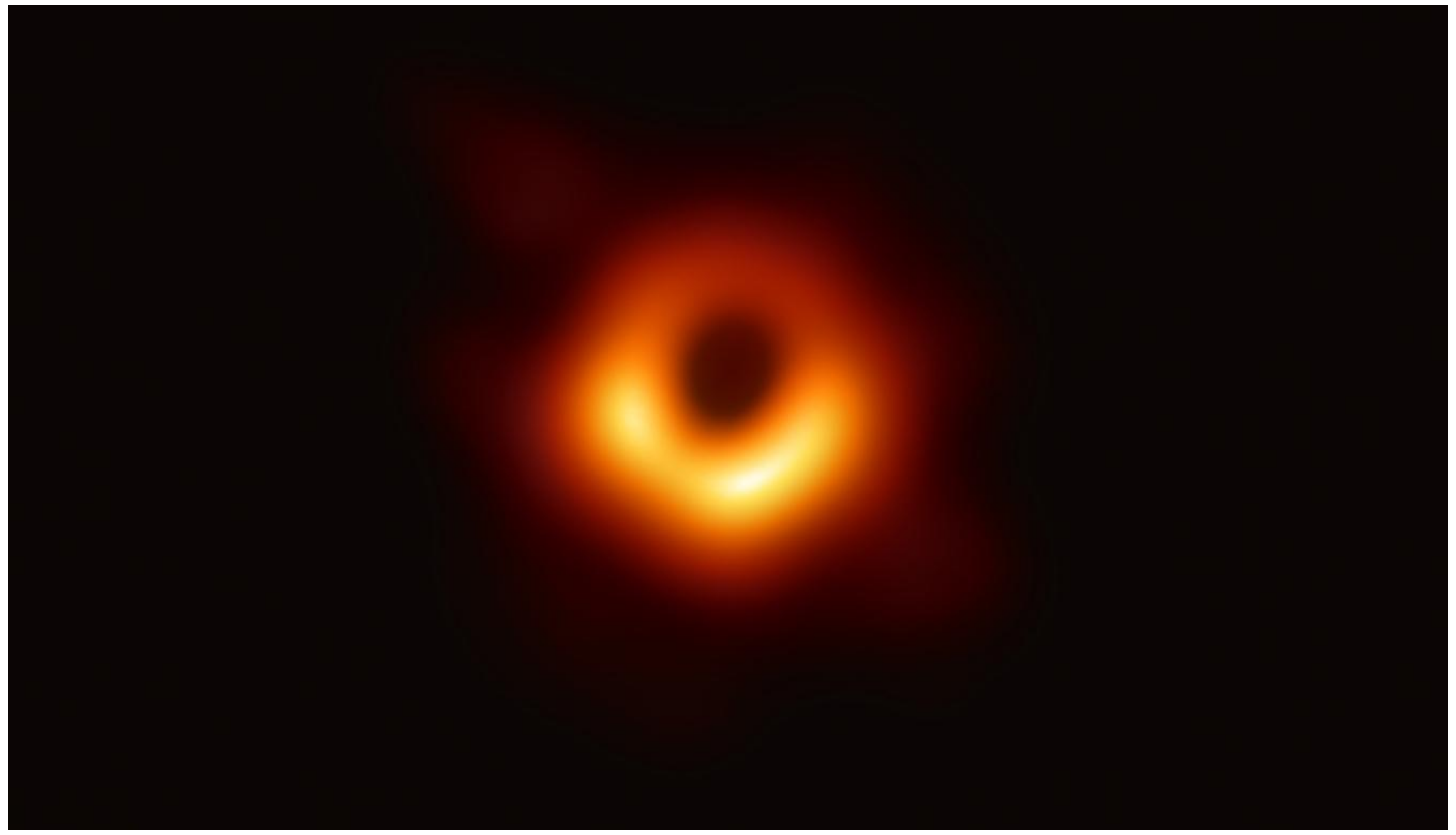
- 10^21 bytes
- 1 billion (10^9) terabytes (!)

[1] Source: https://explodingtopics.com/blog/data-generated-per-day

Scientific Big Data

"The now-famous image of a black hole comes from data collected over a period of seven days. At the end of that observation, the EHT didn't have an image -- it had a mountain of data. Scientists like MIT's Katie Bouman (above) had to develop algorithms to take **5 petabytes** of data and make sense of it."



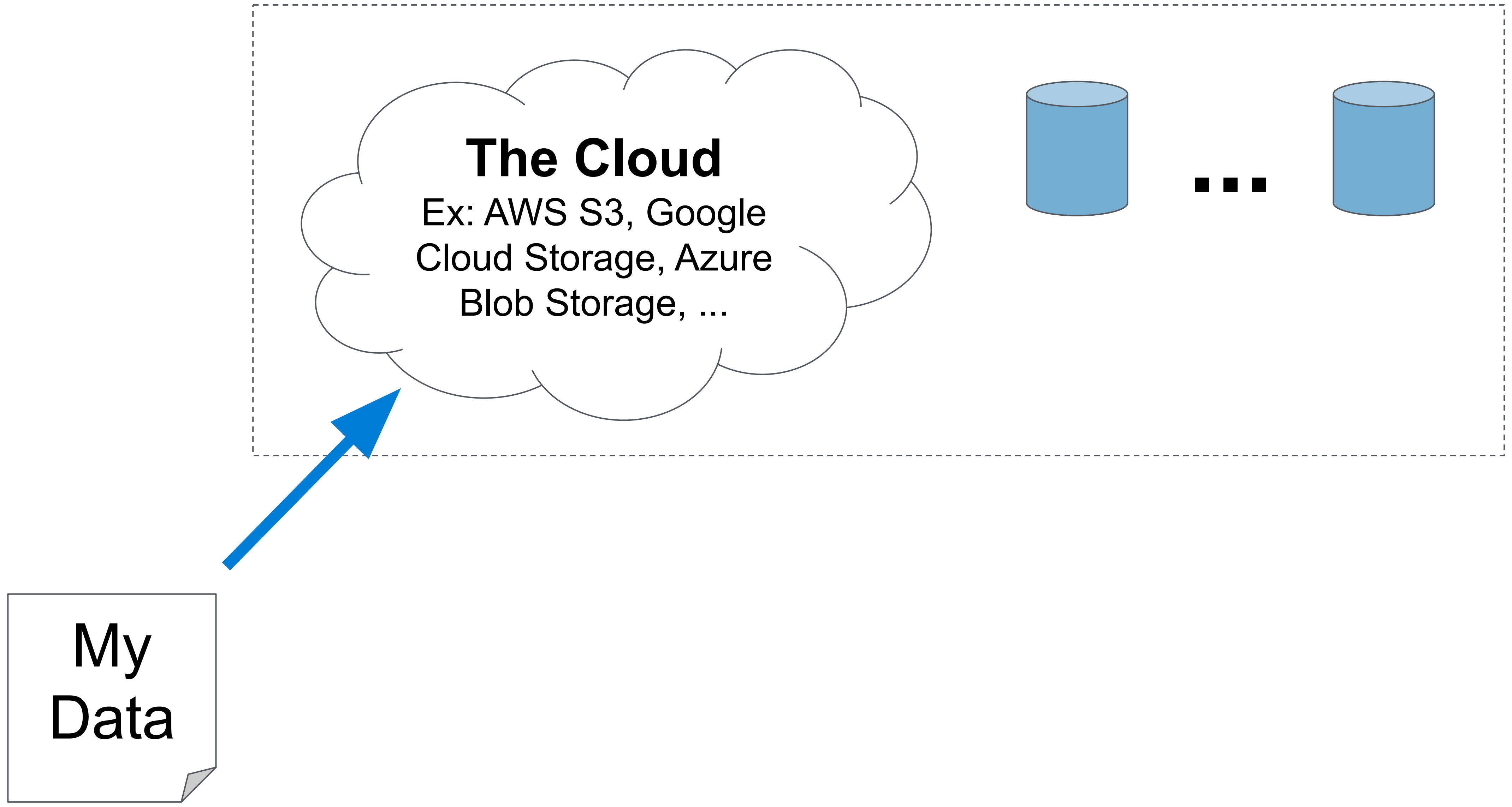


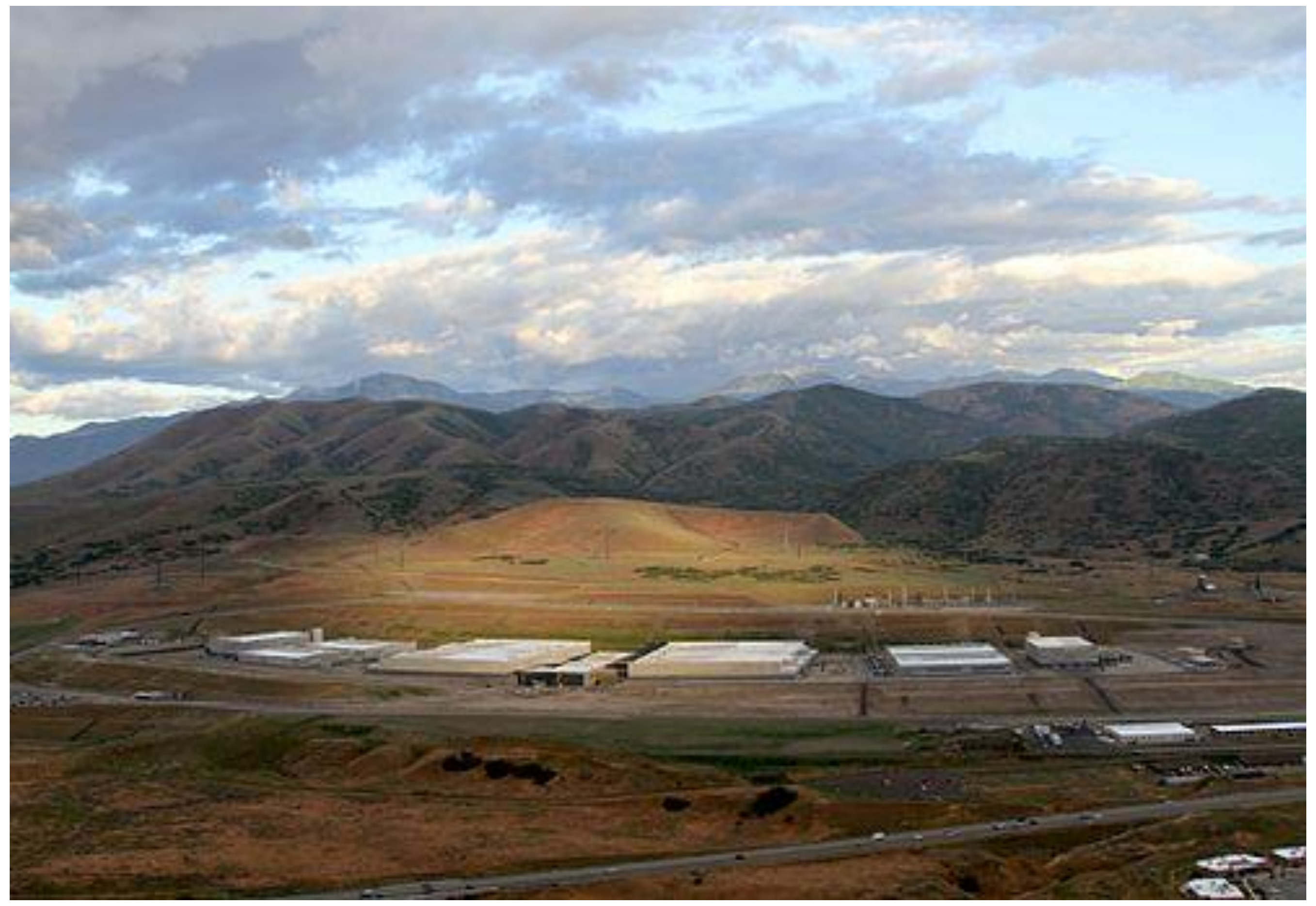
1 petabyte is:
- 10^15 bytes
- 1000 terabytes

https://www.extremetech.com/extreme/289423-it-took-half-a-ton-of-hard-drives-to-store-eht-black-hole-image-data

Dealing with Big Data

- How to **store** the data?
  - Either gigantic in-house data warehouses (ex: "Utah Data Center", USA NSA data warehouse, cost **$2B** to build), or on the cloud (ex: Amazon S3)
- How to **process** the data?
  - Big-data pipelines are now a ubiquitous technology.
  - One popular programming language to efficiently process structured tabular data is: **SQL**





Utah Data Center estimated capacity: 3 - 12 exabytes
1 exabyte: 1 million terabytes (!)

https://nsa.gov1.info/utah-data-center/
https://en.wikipedia.org/wiki/Utah_Data_Center

Structured vs unstructured data

- **Structured data**: data that follows a strict format ("schema")
  - Most common type: tabular data ("tables")
- **Unstructured data**: arbitrary data with no explicit structure
  - Examples: raw text files, images/videos
- Tip: in practice, enforcing structure on your data is helpful: it not only lets us use tools like SQL to run queries against your data, but it can also improve performance (and simplify) data analysis

user_id	**timestamp**	**user_action**	**item**
erickim	2025-07-28 01:15:00	USER_WEBSITE_CLICK	https://c88c.org/su25/
erickim	2025-07-28 01:18:00	USER_VIDEO_WATCH	https://www.youtube.com/watch?v=dQw4w9WgXcQ&list=RDdQw4w9WgXcQ&start_radio=1
some_student	2025-07-29 15:12:00	USER_WEBSITE_CLICK	https://c88c.org/su25/

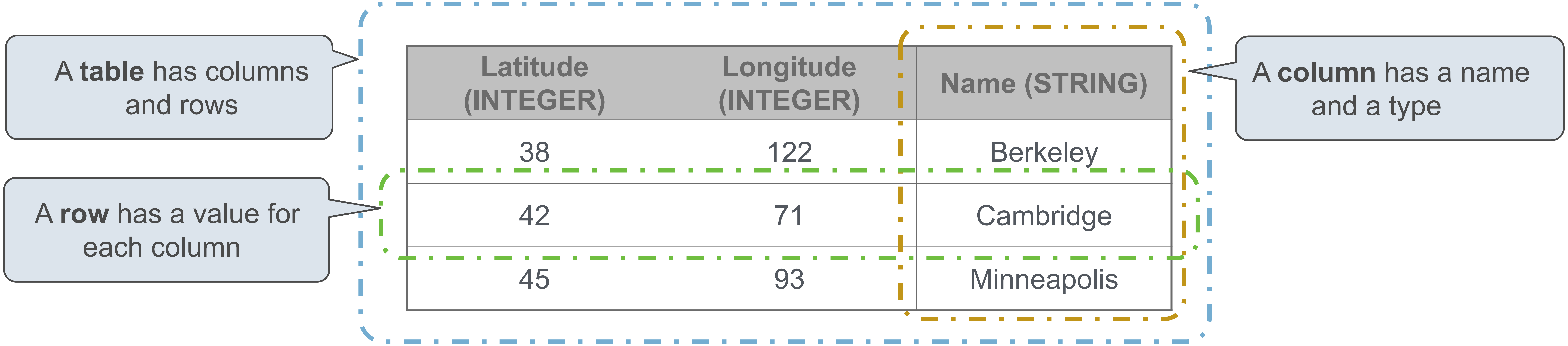
YouTube

NETFLIX

Database Management Systems

Database management systems (DBMS) are important, heavily used, and interesting!

A table consists of a **schema**, and a **collection of records**, which are rows that have a value for each column



A **table** has columns and rows

A **row** has a value for each column

A **column** has a name and a type

Latitude (INTEGER)	Longitude (INTEGER)	Name (STRING)
38	122	Berkeley
42	71	Cambridge
45	93	Minneapolis

The Structured Query Language (SQL) is perhaps the most widely used programming language

SQL is a *declarative* programming language

Structured Query Language (SQL)

What is SQL?

SQL is a language that lets us run **queries** against **tables**.

Example: suppose we had the `employees` table:

Here is a query that computes the average salary by role:

employees

name (STRING)	title (STRING)	salary (INTEGER)
Alyssa P. Hacker	Software Engineer	120000
Oliver Warbucks	Boss	1000000
Louis Reasoner	Software Engineer Intern	30000
John Doe	Software Engineer	90000
Jane Doe	Software Engineer	90000

```
sqlite> select title, avg(salary) from employees group by title;
Boss|1000000.0
Software Engineer|100000.0
Software Engineer Intern|30000.0
```

(Demo: 20.sql:Demo00)

How to run SQL in C88C?

- Two main ways of running SQL interpreters
- **Labs/HWs**: `python3 sqlite_shell.py`
- **Additional practice**: https://code.cs61a.org/ "Start SQL Interpreter"
  - Tip: this has a **"query explainer"**, and has some default tables

Naming Tables

A **select** statement creates a new table and displays it.

A **create table** statement names the result of a **select** statement.

```
create table [name] as [select statement];
```

```
create table parents as
select "d" as parent, "h" as child union
select "a"          , "b"          union
select "a"          , "c"          union
select "f"          , "a"          union
select "f"          , "d"          union
select "f"          , "g"          union
select "e"          , "f";
```

Note: here, SQL is inferring the column datatypes (eg STRING)

Parents:

parent	child
a	b
a	c
d	h
f	a
f	d
f	g
e	f

Select Statements Project Existing Tables

A **select** statement can specify an input table using a **from** clause

A subset of the rows of the input table can be selected using a **where** clause

An ordering over the remaining rows can be declared using an **order by** clause

Column descriptions determine how each input row is projected to a result row

```
select [expression] as [name], [expression] as [name], ... ;

select [columns] from [table] where [condition] order by
[order];
select child from parents where parent = "a";

select parent from parents where parent > child;
```

child
b
c

parent
f
f

Parents:

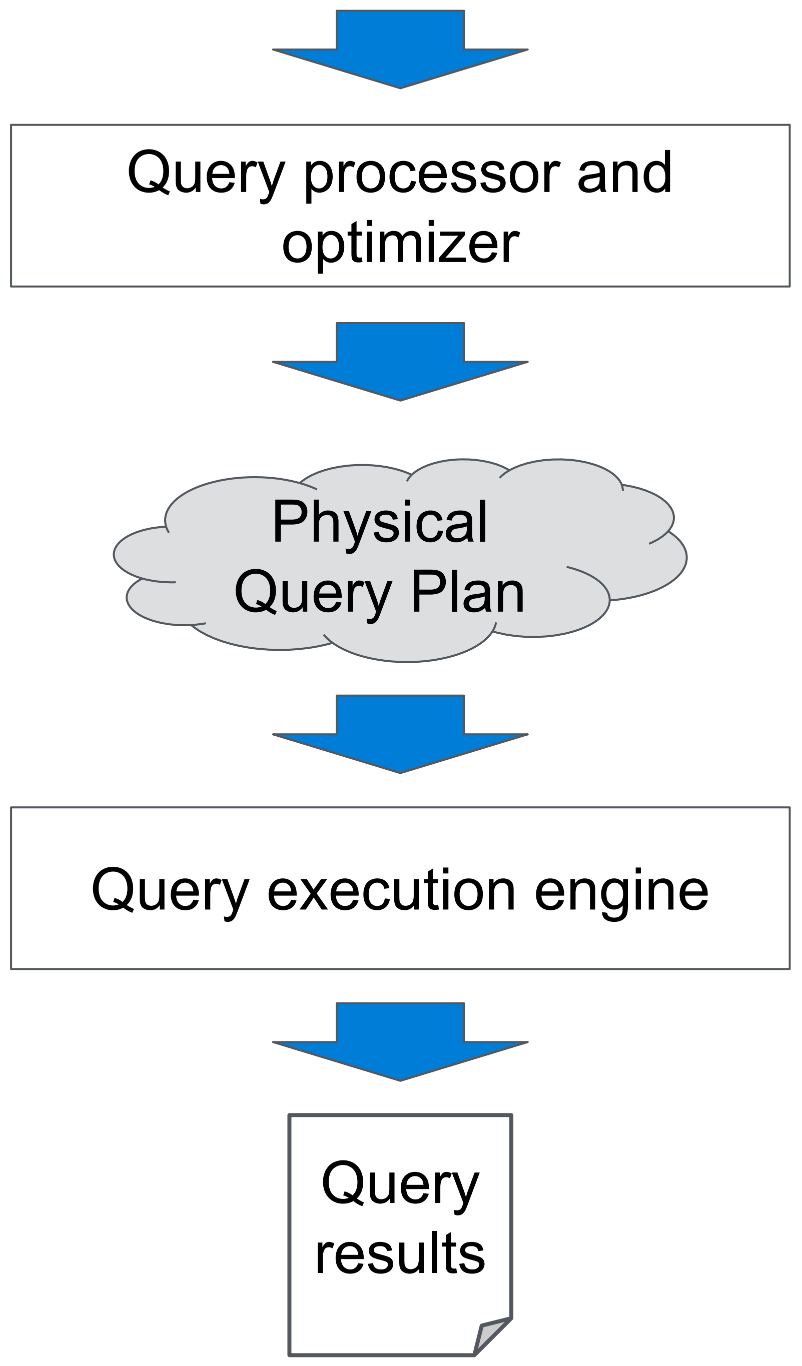
parent	child
a	b
a	c
d	h
f	a
f	d
f	g
e	f

SQL: declarative vs imperative

SQL is a **declarative** language, which means we (the programmer) describe to the computer what we want, and the computer figures out how to generate the desired output.

In contrast, Python is an **imperative** language: one where we (the programmer) instructs the computer the exact steps to generate the desired output.

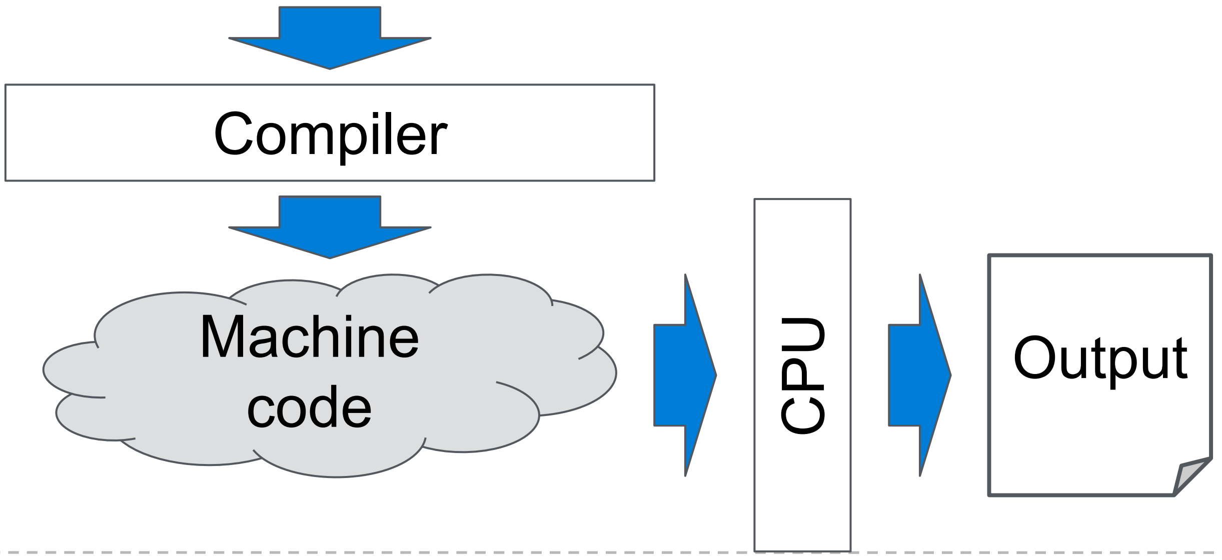
```
select role, avg(salary) from employees group by role;
```



cs186: Databases

vs

```python
def compute_avg_salary_by_title(table_employees):
    # organize salaries by title
    title_to_salaries = {}
    for title, salary in table_employees.get(("title", "salary")):
        if title not in title_to_salaries:
            title_to_salaries[title] = []
        title_to_salaries[title].append(salary)
    # compute average of each salary bucket
    out = []
    for title, salaries in title_to_salaries.items():
        out.append((title, sum(salaries) / len(salaries)))
    return out
```



cs164: Programming languages and compilers

cs61C: Great Ideas in Computer Architecture (Machine Structures)

Declarative Programming

In **declarative programming**:

- A "program" is a description of the desired result
- The interpreter figures out how to generate the result

SQL Server Query Plan:

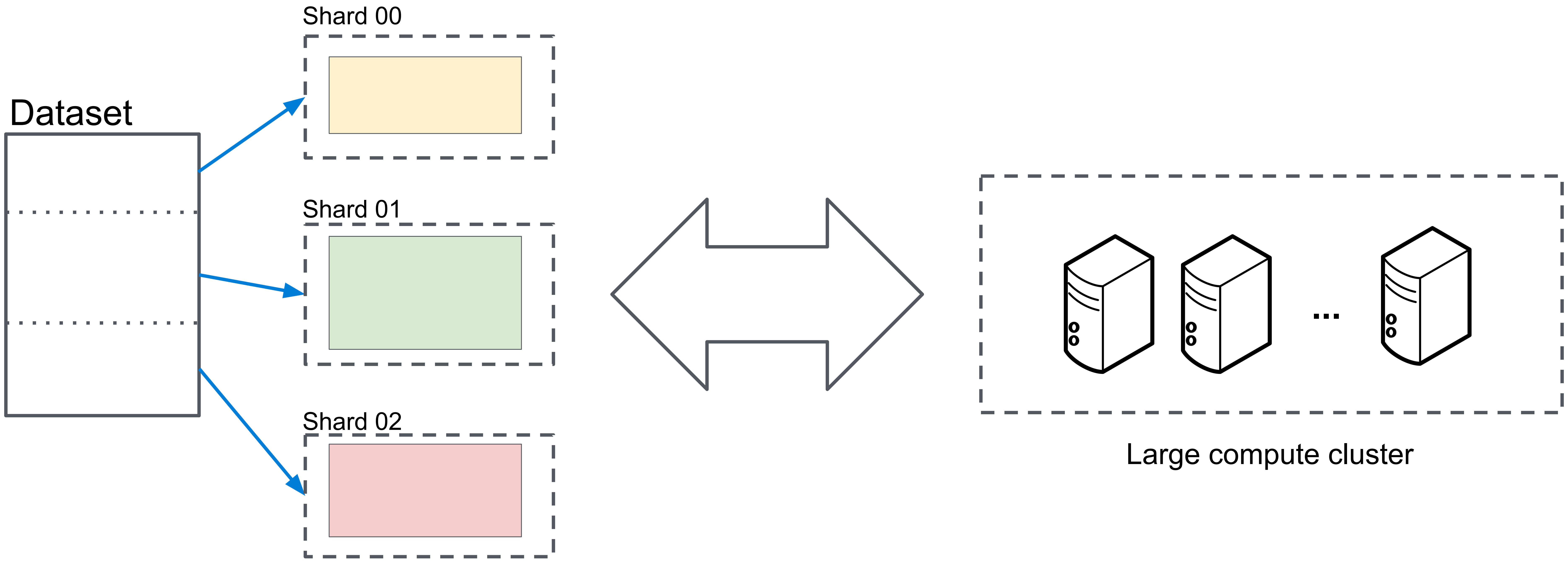
Results
Messages
Execution plan
SELECT
Cost: 0 %
Hash Match
(Right Outer Join)
Cost: 18 %
0.062s
290 of
505 (57%)
Compute Scalar
Cost: 0 %
Index Scan (NonClustered)
[EmailAddress].[IX_EmailAddress_Ema...
Cost: 3 %
0.003s
19972 of
19972 (100%)
Hash Match
(Right Outer Join)
Cost: 0 %
0.046s
290 of
505 (57%)
Compute Scalar
Cost: 0 %
Clustered Index Scan (Clustered)
[PhoneNumberType].[PK_PhoneNumberTy...
Cost: 0 %
0.000s
3 of
3 (100%)
Compute Scalar
Cost: 0 %
Hash Match
(Right Outer Join)
Cost: 13 %
0.046s
290 of
291 (99%)
Compute Scalar
Cost: 0 %
Index Scan (NonClustered)
[PersonPhone].[IX_PersonPhone_Phone...
Cost: 2 %
0.003s
19972 of
19972 (100%)
Hash Match
(Inner Join)
Cost: 1 %
0.030s
290 of
291 (99%)
Compute Scalar
Cost: 0 %
Index Scan (NonClustered)
[CountryRegion].[AK_CountryRegion_N...
Cost: 0 %
0.000s
238 of
238 (100%)

```
Query 1: Query cost (relative to the batch): 100%
SELECT e.[BusinessEntityID], p.[Title], p.[FirstName], p.[MiddleName], p.[LastName], p.[Suffix], e.[JobTitle], pp.[PhoneNumber], pnt.[Name] AS [PhoneNumberType], ea.[E
```

https://learn.microsoft.com/en-us/sql/relational-databases/performance/display-an-actual-execution-plan?view=sql-server-ver16

(Aside) SQL + Big Data

- Big data: key technical challenges are:
  - (1) What if the data is **too large** to fit on a single machine?
  - (2) How can we **speed up** the performance of queries?
- To solve (1): distribute the data across multiple machines (**"data sharding"**)
- To solve (2): throw more machines at the problem to process queries faster (**"horizontal scaling*"**)



* Aka "throw more money at the problem"

(Aside) SQL + Big Data

- Good news: there exist popular, open-source libraries that lets you efficiently run SQL queries on sharded data via distributed computing clusters.
- **SparkSQL**: compiles SQL queries into Spark code, and the Spark engine handles the complexity of orchestrating the distributed computation across data shards and compute cluster machines ("executors").
  - **Spark**: a distributed computing framework that is popular in industry
  - **PySpark**: a Python library on top of Spark (!)
- Common setup:
  - **Data storage**: Either in the cloud (ex: AWS S3), or in an onsite data warehouse
  - **Compute cluster**: Either the cloud (ex: AWS EC2), or an onsite compute cluster
  - **Compute engine/orchestrator**: SparkSQL (or other distributed computing frameworks like MapReduce)