CMSC424: Database Design

Module: Introduction/Overview

Instructor: Amol Deshpande

amol@umd.edu

Welcome to CMSC424: Database Design

- Course description
  - *Students are introduced to database systems and motivates the database approach as a mechanism for modeling the real world. An in-depth coverage of the relational model, logical database design, query languages, and other database concepts including query optimization, concurrency control; transaction management, and log based crash recovery. Distributed and Web database architectures are also discussed.*

- Basics
  - Typical in-person instruction, 2 classes a week etc.
    - Will record lectures but quality may not be as good
    - Plan to substitute a portion of the in-class teaching with videos + in-class activities
  - Will make available pre-recorded videos from Fall 2020
    - Expect about 75% overlap
  - Use the discussion forums, office hours as much as possible – assignments are to be done individually, but you are allowed to discuss among yourselves
  - Fill out the survey covering your background, your expectations, and potential issues (on Gradescope)

No Laptops without permission !

Logistics

- Instructor: Amol Deshpande
  - 5154 IRB, amol@umd.edu
  - Class Webpage: ELMS
- Email to me: write CMSC424 in the title
  - Only if Piazza not suitable for some reason
  - Piazza allows private posts
- TAs: Xinchen Yang, Fnu Ferry, Vanshika Mehta, Ayush Sood

Logistics

- Textbook:
  - Database System Concepts
    - Seventh Edition (6th Edition will be fine too)
    - Abraham Silberschatz, Henry F. Korth, S. Sudarshan
- GitHub:
  - To distribute programming assignments and Slides
  - https://github.com/umddb/cmsc424-spring2024
- Piazza: First resort for any questions
- ELMS: General announcements will be posted there
- GradeScope
  - We will use this for assignments, grading exams, etc.

Grading

- 3 Midterms – 30%
  - 20-25 mins at the beginning of the class
- Final (30%) – minimum of 50% on this + midterms combined to pass
- 6 Assignments (24%) + Setup (1%)
  - SQL, Advanced SQL, SQL + Programming Languages, Normalization/E-R Models (not programming), MongoDB, Spark
  - Late Policy
    - Automatic one-week extension without penalty, but with an explanation of why
    - Beyond that: submissions allowed through the end of the semester with penalties
  - Assignment-0 (Setup) due next Friday
- In-class Activities (15%)
  - Each week – will devote some time in the class to do some worksheets
  - Videos to watch before Tuesday class each week (starting next) – aim to release by Friday night

ChatGPT etc.

- Feel free to use for learning purposes
  - e.g., to debug issues with set up, or to understand some concepts better
  - Beware of the hallucination and other issues
- Will try to provide some starting points and prompts
- Not allowed for directly solving assignments, although debugging would be acceptable

CMSC424: Database Design

Module: Introduction/Overview

Background; What We Will Cover

Instructor: Amol Deshpande
amol@umd.edu

Motivation

- Book Chapters (6th Edition)
  - 1.1, 1.2, 1.3, 1.4, 1.9 (to some extent)

- Key Topics
  - Why managing large volumes of data is difficult
  - Database Systems and Architectures
  - Data Models and Data Abstraction
  - What we will cover in this course

The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume, Velocity, Variety and Veracity**

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

By 2015
4.4 MILLION IT JOBS
will be created globally to support big data, with 1.9 million in the United States

Volume
SCALE OF DATA

40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005

It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day

6 BILLION PEOPLE
have cell phones

WORLD POPULATION: 7 BILLION

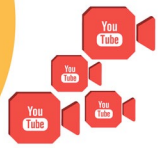
Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored

Variety
DIFFERENT FORMS OF DATA

As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]

By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS

4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month



30 BILLION PIECES OF CONTENT
are shared on Facebook every month

400 MILLION TWEETS
are sent per day by about 200 million monthly active users

Velocity
ANALYSIS OF STREAMING DATA

The New York Stock Exchange captures
1 TB OF TRADE INFORMATION
during each trading session

Modern cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure

By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
– almost 2.5 connections per person on earth

Veracity
UNCERTAINTY OF DATA

1 IN 3 BUSINESS LEADERS
don't trust the information they use to make decisions

in one survey were unsure of how much of their data was inaccurate

Poor data quality costs the US economy around
$3.1 TRILLION A YEAR

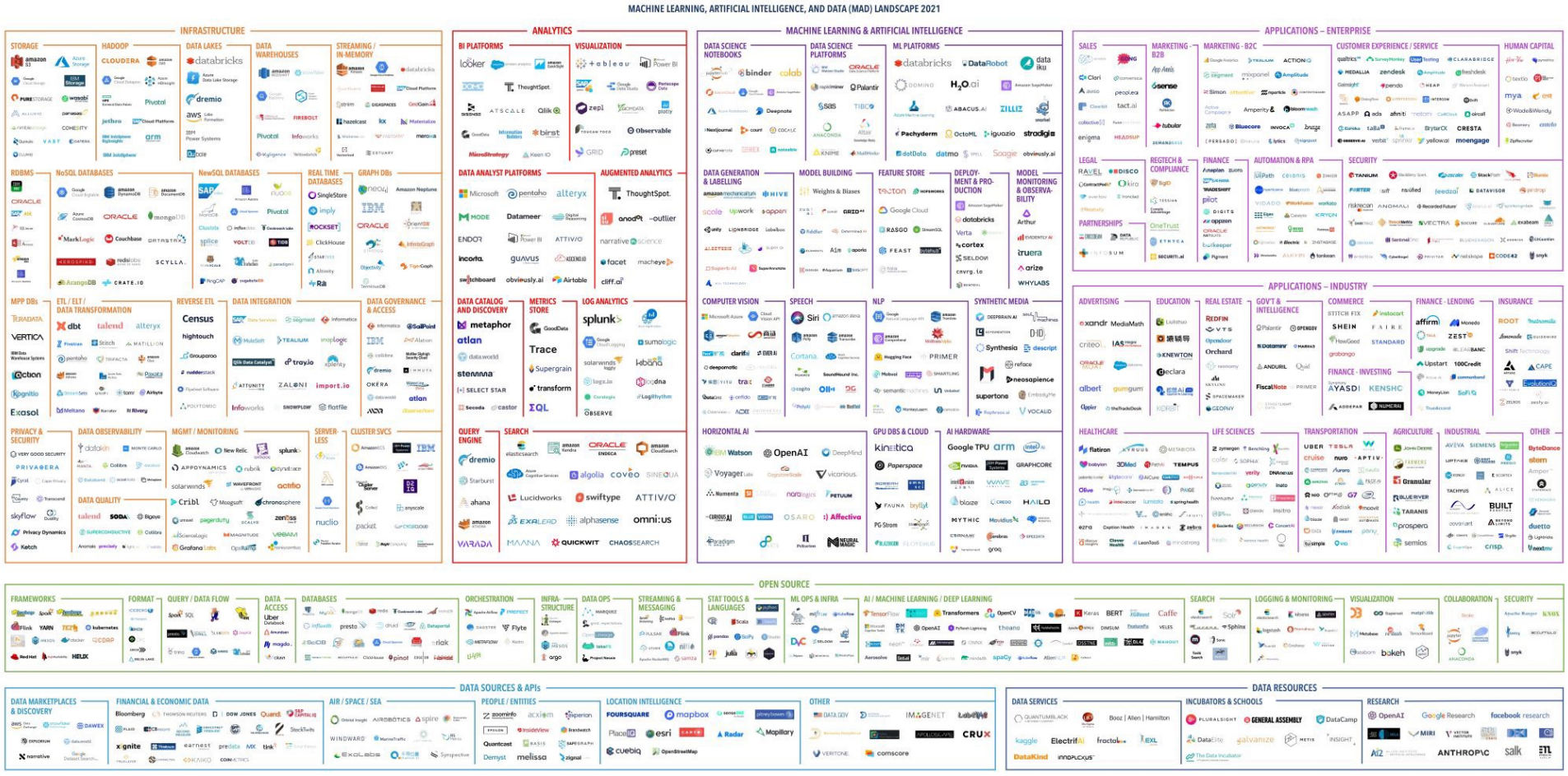
Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

IBM

Data Management Systems

Data management systems at the center of most of the new innovative technologies

An incredible amount of ongoing work in building new types of systems




INFRASTRUCTURE
STORAGE
amazon S3
Azure Storage
Google Cloud Storage
IBM Storage
PURESTORAGE
wasabi
ALLUXIO
panasas
nimblestorage
COHESITY
Qumulo
VAST
DATERA
CLUMIO
HADOOP
CLOUDERA
amazon EMR
Google Cloud Dataproc
Azure HDInsight
HPE Ezmeral Data Fabric
Pivotal
jethro
SAP Cloud Platform
IBM InfoSphere BigInsights
arm
IBM InfoSphere
DATA LAKES
databricks
Azure Data Lake Storage
dremio
aws Lake Formation
IBM Power Systems
Qubole
DATA WAREHOUSES
amazon REDSHIFT
snowflake
Google BigQuery
FIREBOLT
Pivotal
Infoworks
Kyligence
Yellowbrick
STREAMING / IN-MEMORY
amazon Kinesis
databricks
SAP Cloud Platform
striim
GIGASPACES
GridGain
hazelcast
Materialize
meroxa
Vectorized
ESTUARY
RDBMS
ORACLE
SQLServer
amazon RDS
Amazon Aurora
NoSQL DATABASES
Google Cloud Bigtable
amazon DynamoDB
amazon DocumentDB
Azure CosmosDB
ORACLE
mongoDB
MarkLogic
Couchbase
DATASTAX
AEROSPIKE
redislabs
SCYLLA.
ArangoDB
CRATE.IO
NewSQL DATABASES
SAP HANA
Amazon Aurora
NUODB
MariaDB
Pivotal
Clustrix
influxdata
Cockroach Labs
splice
VOLTDB
TiDB
TIMESCALE
Trafodion
paradigm4
PingCAP
yugabyteDB
REAL TIME DATABASES
SingleStore
imply
ROCKSET
ClickHouse
STARTREE
Altinity
Rill
GRAPH DBs
neo4j
Amazon Neptune
IBM
ORACLE
OrientDB
STARDOG
InfiniteGraph
Objectivity
TigerGraph
TerminusDB
MPP DBs
TERADATA
VERTICA
IBM Data Warehouse Systems
Actian
Kognitio
Exasol
ETL / ELT / DATA TRANSFORMATION
dbt
talend
alteryx
Fivetran
Stitch
MATILLION
pentaho
TRIFACTA
amazon Glue
amazon Athena
Azure Data Factory
Paxata
StreamSets
UNIFI
tamr
Airbyte
Meltano
Narrator
Rivery
REVERSE ETL
Census
hightouch
Grouparoo
rudderstack
Flywheel Software
POLYTOMIC
DATA INTEGRATION
SAP Data Services
segment
Informatica
MuleSoft
TEALIUM
snaplogic
Qlik Data Catalyst
tray.io
xplenty
ATTUNITY
ZALONI
import.io
Infoworks
SNOWPLOW
flatfile
DATA GOVERNANCE & ACCESS
Informatica
SailPoint
IBM
Alation
collibra
McAfee Skyhigh Security Cloud
dremio
IMMUTA
OKERA
WaterLine Data
data.world
atlan
ObservePoint
PRIVACY & SECURITY
VERY GOOD SECURITY
PRIVAGERA
Cyral
Cape Privacy
Country
Transcend
skyflow
Duality
Privacy Dynamics
Ketch
DATA OBSERVABILITY
datakin
MONTE CARLO
MANTA
Collibra
datafold
Databand
acceldata
Metaplane
DATA QUALITY
talend
SODA
Bigeye
SUPERCONDUCTIVE
Collibra
Anomalo
precisely
validio
MGMT / MONITORING
amazon Cloudwatch
New Relic
DATADOG
splunk>
APPDYNAMICS
rubrik
dynatrace
solarwinds
WAVEFRONT by vmware
actifio
Cribl
Moogsoft
chronosphere
unravel
pagerduty
SCALYR
zenoss
ScienceLogic
MAGNITUDE
VEEAM
Grafana Labs
OpsRamp
honeycomb.io
SERVER-LESS
Azure
nuclio
CLUSTER SVCS
AmazonECS
IBM Power Systems
IBM
AmazonEKS
Cluster Server
D2IQ
Coiled
anyscale
packet
CYCLECLOUD
Bright Computing

Data Management

A large fraction of the data still in traditional DBMS systems

Still open and active research areas about improving performance, energy efficiency, new functionalities, changing hardware spectrum (SSDs) and so on…

Much of the data not stored in traditional database systems today

For a variety of fairly valid reasons

- Stream processing systems (focusing on *streaming* data)
- Special-purpose data warehousing systems (most start from some RDBMS)
- Batch analysis frameworks (like Hadoop, Pregel, Spark, …)
  - Typically data stored in distributed file systems
- Key-value stores (like HBase, Cassandra, Redis, …)
  - Basically persistent distributed hash tables
- Semi-structured/Document data stores (for XML/JSON query processing)
- Graph databases
- Data lakes (e.g., scientific data, machine learning data)

However, many lessons to be learned from database research

We see much reinvention of the wheel and similar mistakes being made as early on

Data Management Challenges (1)

- How to “represent”/”model” the data so users can easily understand, query, analyze, and update?
  - This is about the “logical” representation rather than “physical” -- we will worry about physical representation later
  - Whatever format we use needs to be easy for humans to work with, over a long period of time as data and usage evolves

As tables (like Excel)?

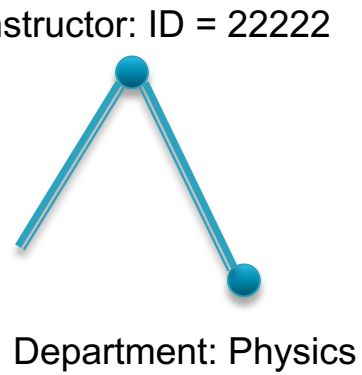
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000

JSON Documents?

```
{“instructors”: [
        {“ID”: “22222”,
         “name”: “Einstein”,
         “dept_name”: “Physics”,
            ….
         },
         ….
```

Graph?



Data Management Challenges (1)

- How to “represent”/”model” the data so users can easily understand, query, analyze, and update?
  - This is about the “logical” representation rather than “physical” -- we will worry about physical representation later
  - Whatever format we use needs to be easy for humans to work with, over a long period of time as data and usage evolves
  - Many questions…
    - Pros and cons of different ways to do this
    - How to discuss whether our chosen data format is good or bad
      - e.g., some ways of storing data may increase conflicts in the data
    - How to deal with changes to the requirements and types of data being stored
      - Backwards compatibility is a huge problem leading to messy representations later on

Data Modeling

Data Management Challenges (2)

- How to query the data? How to process/analyze it ? (from users'/analysts' perspective)
  - "find the stores with the maximum increase in sales in last month"
    - We can't expect the users to write Java programs
  - "how much time from here to Pittsburgh if I start at 2pm ?"
    - Data is there (GPS, live traffic data), but requires predictive capabilities
  - Increasing need to convert "information" to "knowledge": **Data mining/Machine Learning**
    - Predicting which TikToks/tweets to show, …
  - Needs to be easy to use, but powerful enough for what needs to be done
    - Otherwise users will simply take out all the data and write their own programs

Query Languages and Programming Abstractions

Data Management Challenges (3)

- How to query the data? How to process/analyze it ? (from implementation perspective)
  - How to handle the increasing scale of the data
  - How to support more and more complex analytics/machine learning over the data
  - How to process the data in real-time, as it is arriving into the system
  - How to find the best way to execute a query with least resources/time
  - How to utilize large numbers of machines to execute queries more efficiently
  - …

Database Query Engines
Big Data Systems

Data Management Challenges (4)

- Modifying/updating the data given:
  - Many users simultaneously updating the data…
  - While the data is (potentially) replicated across the globe…
  - And any server or the network may go down at any point.

Update 1

```
a = a + 10
b = b - 10
save a & b
```

Update 2

```
inc = a * 0.15
a = a + inc
b = b - inc
save a & b
```

If allowed to run concurrently

```
a = a + 10
inc = a * 0.15
a = a + inc
b = b - 10
save a & b
b = b - inc
save a & b
```

Doesn't happen if forced to run one after the other, but unacceptable performance

Data Management Challenges (4)

- Modifying/updating the data given:
  - Many users simultaneously updating the data…
  - While the data is (potentially) replicated across the globe…
  - And any server or the network may go down at any point.

Update

a = a + 10
b = b - 10
save a
save b

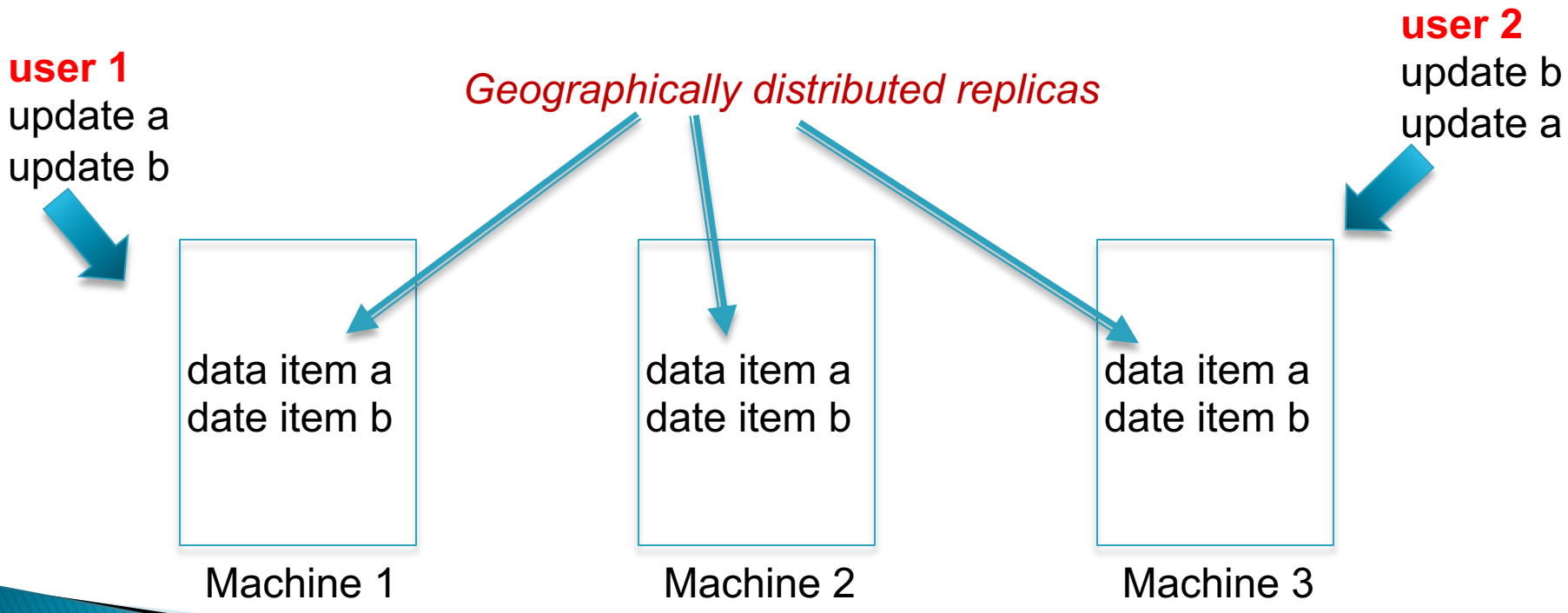
System crashes

Need to figure out how to handle partially finished transactions

More complications if we do update-in-place (i.e., a = a + 10 updates the original value)

Data Management Challenges (4)

- Modifying/updating the data given:
  - Many users simultaneously updating the data…
  - While the data is (potentially) replicated across the globe…
  - And any server or the network may go down at any point.



Data Management Challenges (4)

- Modifying/updating the data given:
  - Many users simultaneously updating the data…
  - While the data is (potentially) replicated across the globe…
  - And any server or the network may go down at any point.
- From Users' perspective:
  - What kinds of guarantees can be provided to the users about the end result, and about the order in which updates are made?
  - What if the users are okay with weaker guarantees?
- From Implementation perspective:
  - How to provide the guarantees efficiently?

Transactions

Data Management Challenges (5-∞)

- Data Integration, Transformation, and Cleaning
  - Large amounts of time spent in just getting data ready for analysis, by some accounts 90-95%
  - Real-world data has a lot of noise, missing values, etc…
- Security
  - Access control, use of encryption, …
- Privacy
  - How to allow users to query data without revealing it
  - How to enable public data releases (e.g., by Census Buereau)
- …

Database System

- A DBMS is a software system designed to store, manage, facilitate access to databases
  - Typically uses a specific *data model*, and
  - Supports some level of *physical and logical data independence*

- Provides:
  - Data Definition Language (DDL)
    - For defining and modifying the schemas
  - Data Manipulation Language (DML)
    - For retrieving, modifying, analyzing the data itself
  - Guarantees about correctness in presence of failures and concurrency, data semantics etc.

Example: Relational DBMS and SQL

- **SQL** (sequel): Structured Query Language

- **Data definition (DDL)**
  - **create table** *instructor* (
    ID **char**(5),
    name **varchar**(20),
    dept_name **varchar**(20),
    salary **numeric**(8,2))

- **Data manipulation (DML)**
  - Example: Find the name of the instructor with ID 22222

    select *name*
    from *instructor*
    where *instructor.ID* = '22222'

Example: MongoDB and MongoQL

- **Data definition (DDL)**
  - Tuple/Row ➔ Document; Relation/Table ➔ Collection

```
db.createCollection("people")
```

  - Also, “insert” implicitly creates collections

```
db.people.insertOne(
     { user_id: "abc123",
       age: 55,
       status: "A" } )
```

- **Data manipulation (DML)**
  - Example: Find all users with `status = “A”` and return `user_id` and `status`, but don’t return `_id`

```
db.people.find(
          { status: "A" },
          { user_id: 1, status: 1, _id: 0 } )
```

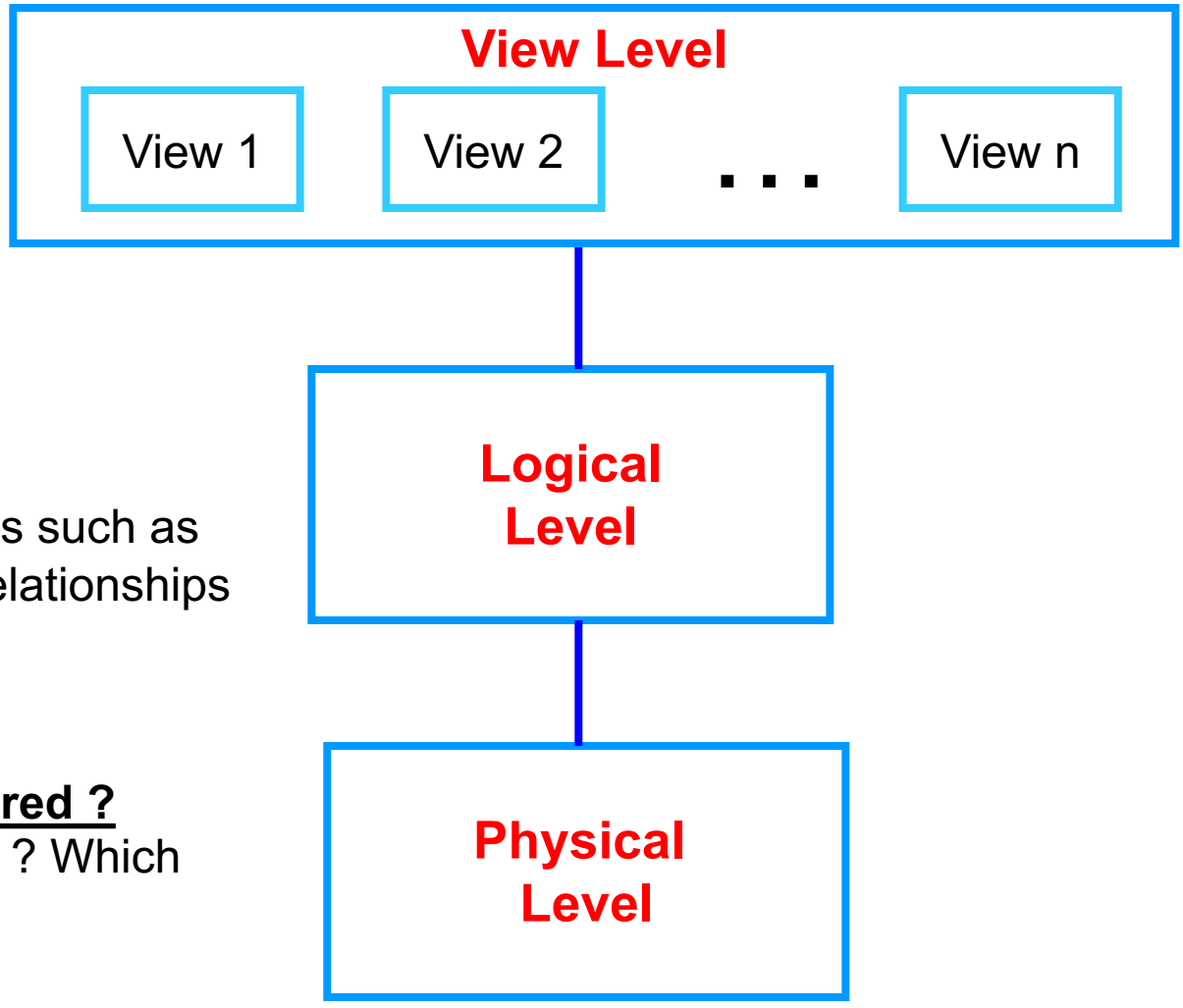
Data Abstraction

- Probably *the* most important purpose of a DBMS
- Goal: Hiding *low-level details* from the users of the system
  - Alternatively: the principle that
    - *applications and users should be insulated from how data is structured and stored*
  - Also called *data independence*

- Through use of *logical abstractions*

Data Abstraction

What data users and application programs see ?



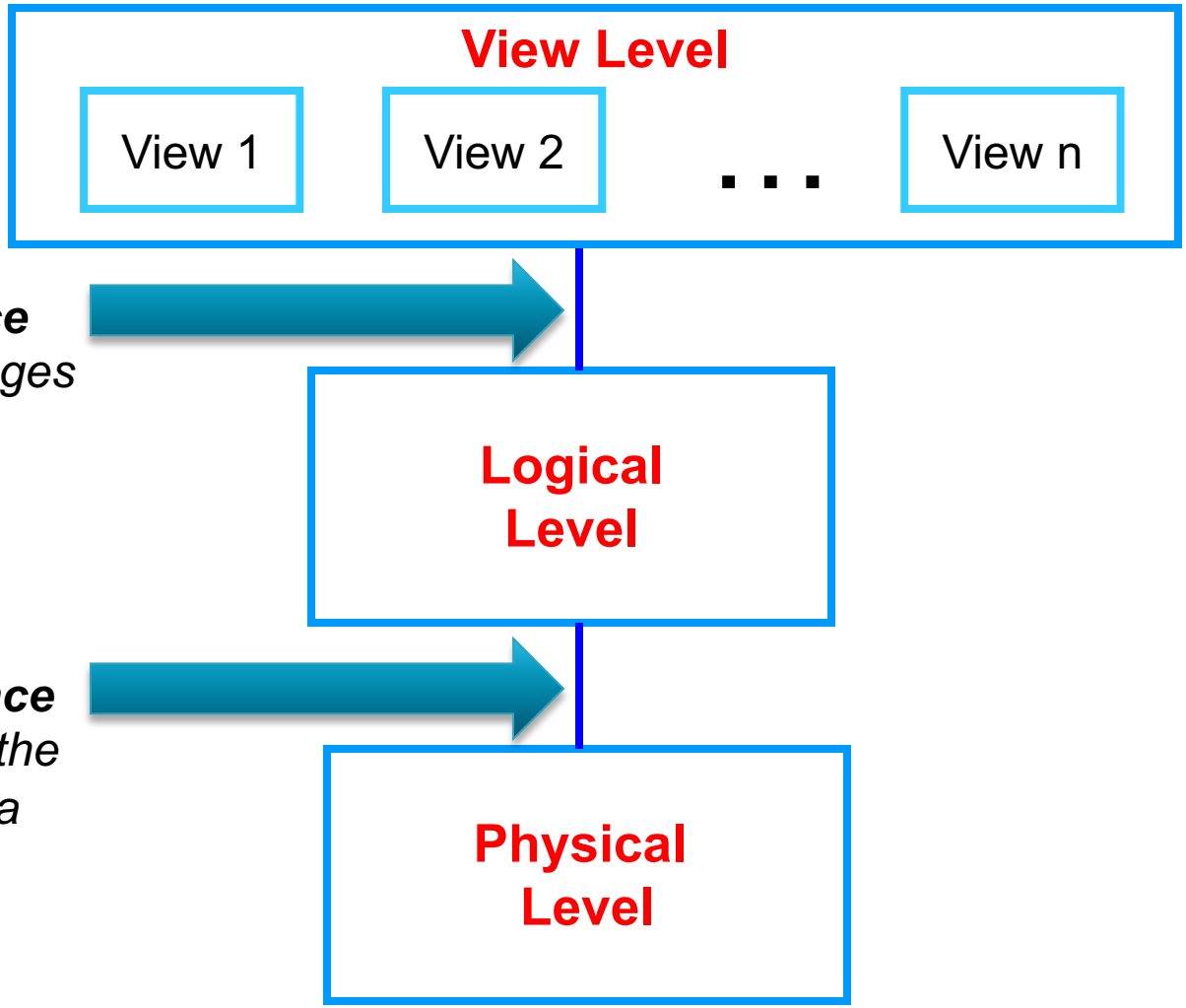
What data is stored ?

describe data properties such as data semantics, data relationships

How data is actually stored ?

e.g. are we using disks ? Which file system ?

Data Abstraction

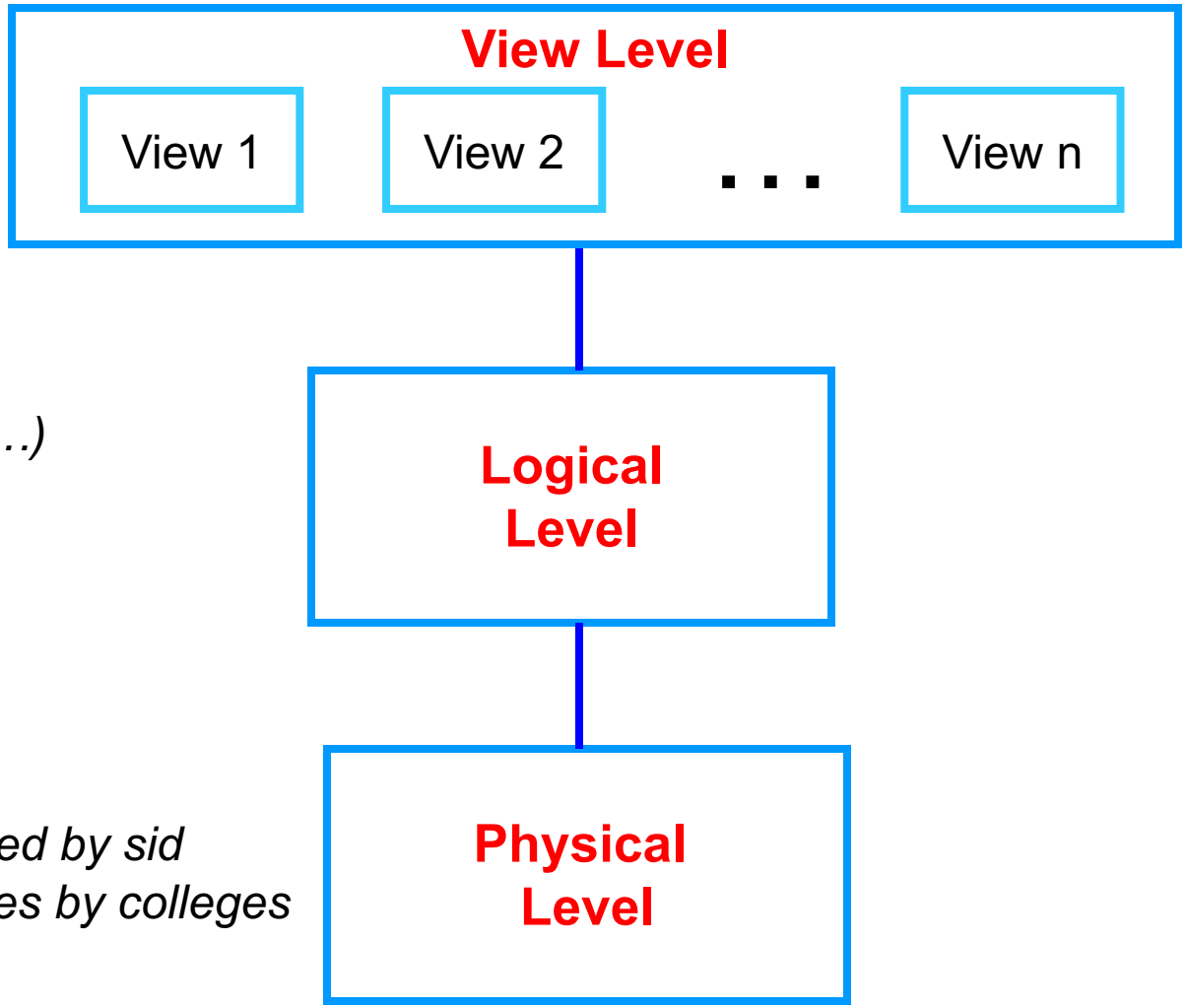


Logical Data Independence
Protection from logical changes to the schema

Physical Data Independence
Protection from changes to the physical structure of the data

Data Abstractions: Example

A View Schema
course_info(#registered,…)

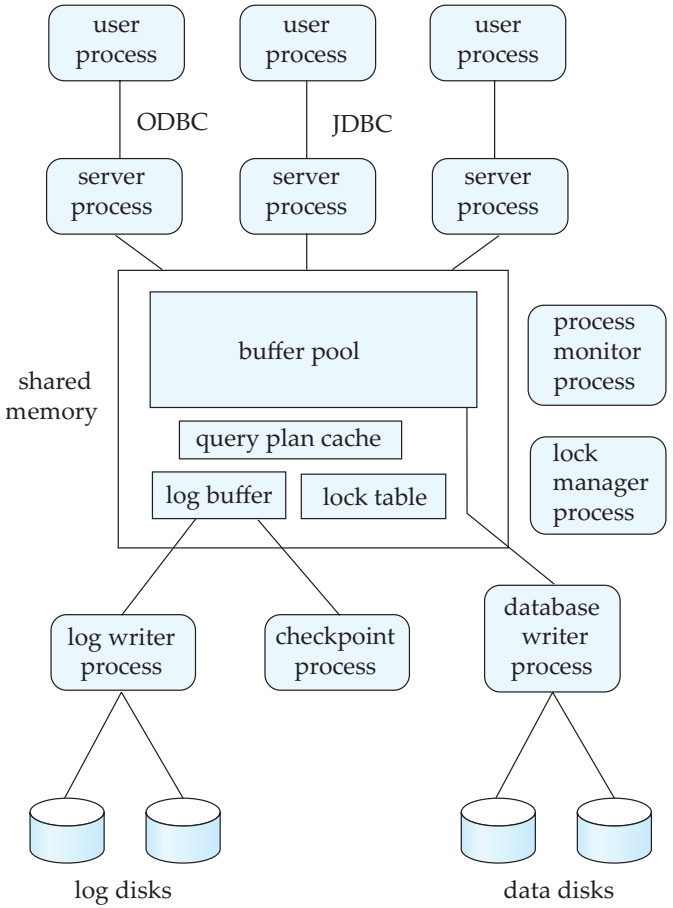


Logical Schema
students(sid, name, major, …)
courses(cid, name, …)
enrolled(sid, cid, …)

Physical Schema
all students in one file ordered by sid
courses split into multiple files by colleges

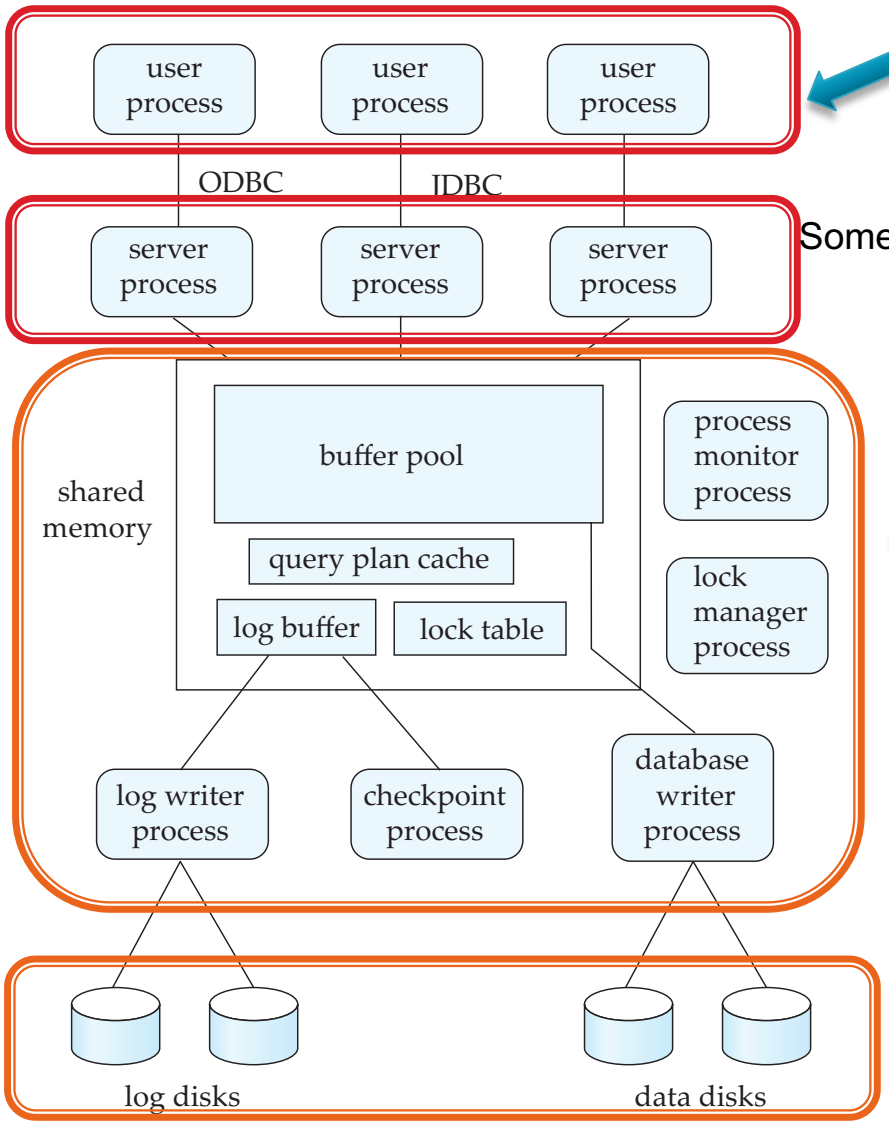
Database Architecture: Pre-2000's

- All data was typically in hard disks or arrays of hard disks
- RAM (Memory) was never enough
  - So always had to worry about what was in memory vs not
- Almost no real "distributed" execution
  - Different from "parallel", i.e., on co-located clusters of computers
- Relatively well-understood use cases
  - Report generation
  - Interactive data analysis and exploration
  - Supporting transactions



From Chapter 20

Traditional RDBMS Architecture



Clients may be anywhere – e.g., ATMs, desktops, laptops, web apps etc.

Talk to the database using standard protocols like JDBC/ODBC, SOAP, or REST (today), or proprietary protocols

Some sort of load balancer or intake mechanism

Typical components in a database system: some for queries, some for transactions

Maybe on a single physical computer or a cluster connected by a fast network

Data Storage Systems:
(1) Punch cards (long time ago)
(2) Hard disks (still prevalent)
(3) SSDs

Need "redundancy" and "fault-tolerance"
Data once stored should always be there

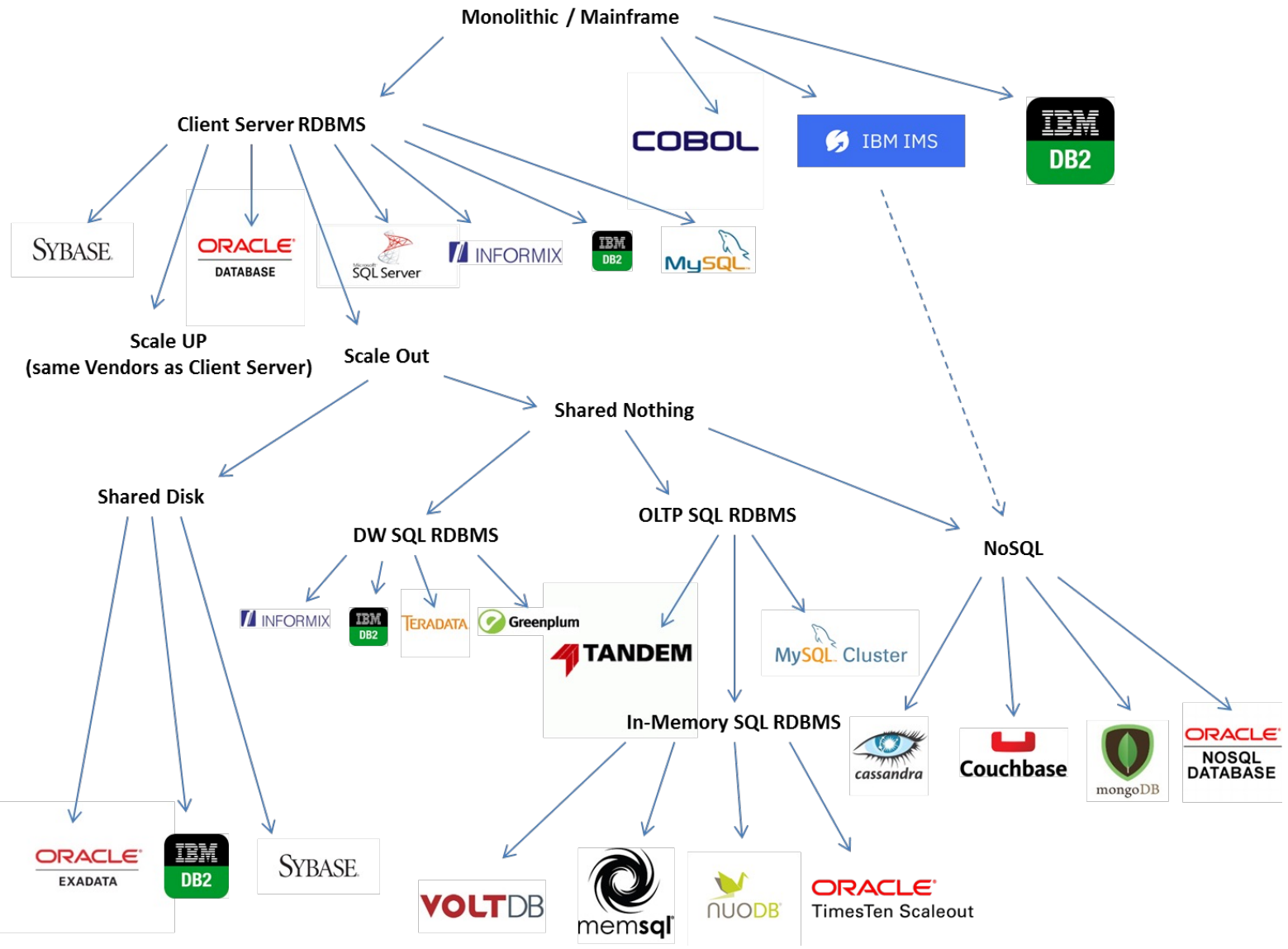
RAID = Redundant Array of Independent Disks

Database Architecture : Today

- Much more diversity in the architectures that we see
  - More modern hardware architectures
    - Massively parallel computers
    - SSDs
    - Massive amounts of RAM – often don't need to worry about data fitting in memory
    - Much faster networks, even over a wide area
    - Virtualization and Containerization
    - Cloud Computing
  - As a result: Data and execution typically distributed all over the place

- Much more diversity in data processing applications
  - Much more non-relational data (images, text, video)
  - Data Analytics/Machine learning more common use-cases

- Much more diversity in "data models"
  - Document data models (JSON, XML), Key-value data model, Graph data model, RDF



From: https://blogs.oracle.com/timesten/the-evolution-of-db-architectures
(Oracle-focused)

Data Warehouses

For: Large-scale data processing (TBs to PBs)
Parallel architectures (lots of co-located computers)
SQL and Reporting
No transactions

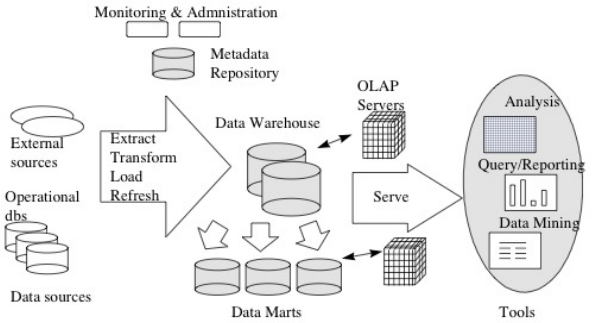
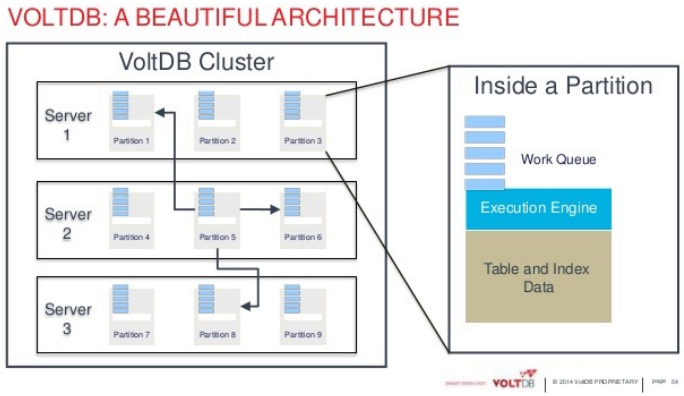


Figure 1. Data Warehousing Architecture

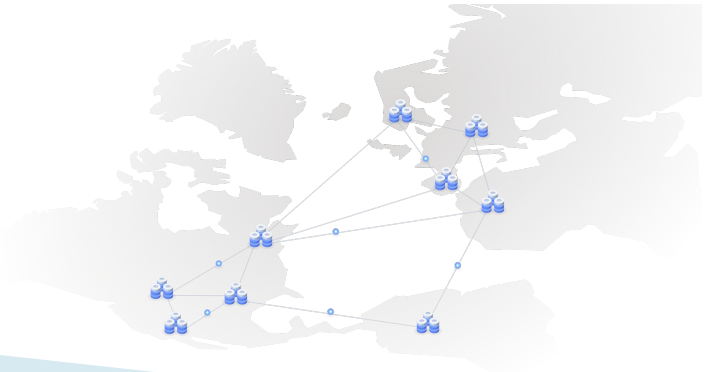
In-memory OLTP (on-line transaction processing)

For: Extremely fast transactions
Many-core or parallel architectures
Very limited SQL – mostly focused on “writes”
Typically assume data fits in memory across servers



Highly available, distributed OLTP

For: Distributed scenarios where clients are all over the world
Focus on “consistency” – how to make sure all users see the same data
Limited SQL – mostly focused on “writes”
Considerations of memory vs disk less important

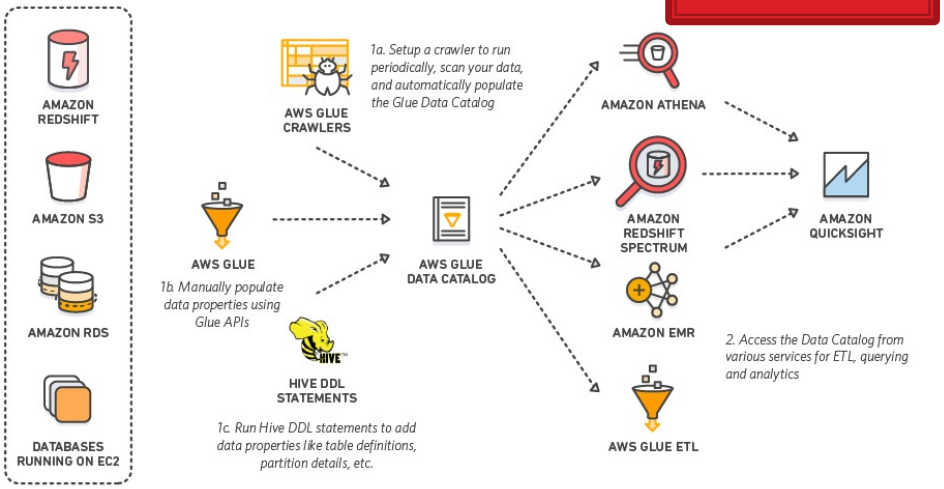


Extract-Transform-Load Systems, or Map-Reduce, or Big Data Frameworks

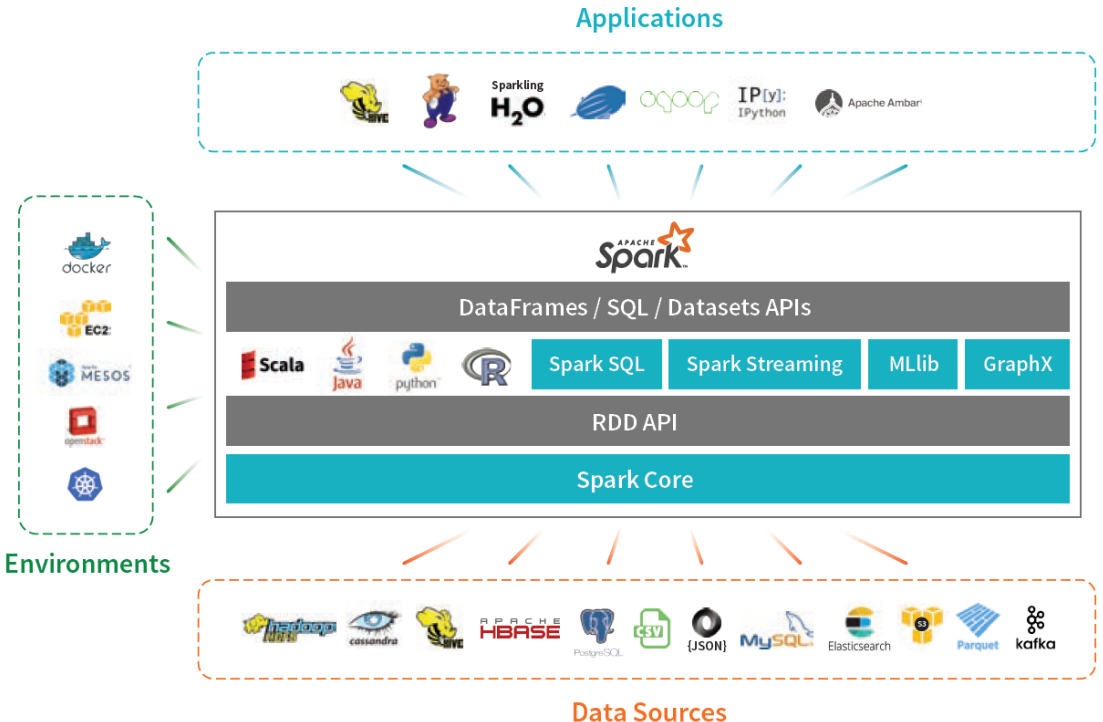
For: Large-scale, “ad hoc” data analysis

Mix of parallel and distributed architectures
Data usually coming from many different sources
Mix of SQL, Machine Learning, and ad hoc tasks (e.g., do image analysis, followed by SQL)

AWS Glue



Apache Spark



Okay…

- Key takeaway: Modern data architectures are all over the place
  - We haven't talked about NoSQL (MongoDB, etc.), Machine Learning, "Streaming"…
- Fundamentals haven't changed that much though
  - We are still either:
    - Going from some "input datasets" to an "output dataset" (queries/analytics)
    - Modifying data (transactions)
  - SQL is still very common, albeit often disguised
    - Spark RDD operations map nicely to SQL joins and aggregates (unified now)
    - MongoDB lookups, filters, and aggregates map to joins, selects, and aggregates in SQL
- But "performance trade-offs" are all over the place now
  - 30 years ago, we worried a lot about hard disks and things fitting in memory
  - Today, focus more on networks
- Focus has shifted to other aspects of data processing pipelines
  - Analytics/Machine learning, data cleaning, statistics

What we will cover…

- Data Modeling
  - Data represented in tabular forms (Relational)
  - JSON/Document Data Model (MongoDB)
  - Entity-relationship Model for Schema Design
  - Normalization theory to formally define a ”good” schema
  - We will discuss a more primitive data model as well (Apache Spark)
- Query Languages and Programming Abstractions
  - SQL, MongoDB Query Language, Apache Spark MapReduce-like Framework, Some others
- Query Engines and Big Data Frameworks
  - Considerations in efficient implementations
- Transactions
  - How to reason about consistency and durability guarantees (ACID)
  - Basics of how to achieve those guarantees in centralized and distributed settings

What we will cover…

- This course is NOT about learning how to write SQL queries
  - Although we will practice that a fair bit…
- It is about learning the foundations of how data management systems are built and why they are built that way, so you:
  - Can think about the design decisions more systematically (e.g., through use of normalization theory and E-R modeling)
  - Understand the role of data modeling and how different models impact design decisions
  - Develop a deeper understanding of the differences and similarities between different types of database systems (e.g., PostgreSQL vs MongoDB)
  - Are equipped (to some extent) to design next-generation data systems
  - Are able to reason about performance of the queries/tasks

Structure of the Course

- Introduction
  - Motivation, data abstraction, common data systems architectures today
- Relational Model + SQL (Three Programming assignments)
  - How to model and query data using SQL
  - How to update data using transactions and considerations therein
- Schema Design: Entity-relationship Models and Normalization (One assignment)
  - How to create a database schema, and how to ensure it is “good”
- NoSQL (somewhat of a misnomer) (Two Programming assignments)
  - Document, key-value, and graph data models
  - MongoDB and its Query Language
  - Map-reduce Model and Apache Spark
- Implementation Issues
  - Different types of storage, and how to ensure reliability in presence of failures
  - Indexes for faster retrieval of data
  - How an SQL query is processed and optimized
  - How to do concurrent updates correctly
  - How to ensure consistency in presence of failures