Chapter 10: Big Data

Database System Concepts, 7th Ed.

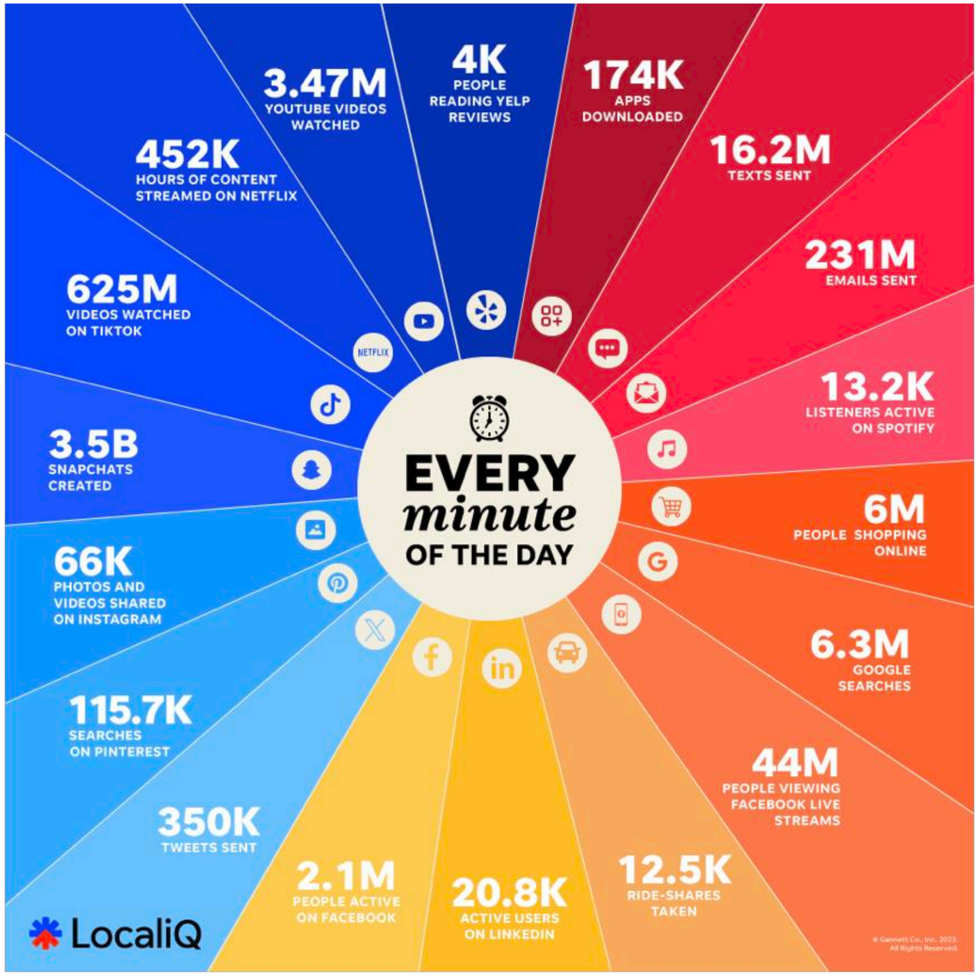
Motivation

- Very large volumes of data being collected
  - Driven by growth of web, social media, and more recently internet-of-things
  - Web logs were an early source of data
    - Analytics on web logs has great value for advertisements, web site structuring, what posts to show to a user, etc
- Big Data: differentiated from data handled by earlier generation databases
  - **Volume**: much larger amounts of data stored
  - **Velocity**: much higher rates of insertions
  - **Variety**: many types of data, beyond relational data

Big Data Producers

- Everything and everyone produces data nowadays
  - Users in the Internet
  - Mobile users (mobile devices, photos/videos, tweets)
  - Science and researchers
  - Industries (plants, cars, connected devices)
  - Computers monitoring processes, and producing data
  - Sensors (e.g. IoT, cameras, remote sensors, satellites)

What happens in an Internet Minute in 2024



https://localiq.com/blog/what-happens-in-an-internet-minute/

Big Data Characteristics

- Volume
- Variety
- Velocity
- Variability
- Veracity

Big data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization.

[Gartner's definition]

"Big Data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or does not fit the structures of your database architectures. To gain value from this data, you must choose an alternative way to process it". (Dumbill, 2013)

Big Data: Volume

- (Gandomi & Haider, 2015):
  - Volume is a characteristic which indicates the magnitude of data
  - Data size is relative and varies according to the periodicity and the type of data. It is impractical to define a specific threshold for Big Data volume, as different types of data require different technologies to deal with it (e.g., tabular data and video data)
- (Krishnan, 2013):
  - Volume characterizes the amount of data that is continuously generated.
- (Zikopoulos & Eaton, 2011):
  - The main cause for the ever increasing volume is the fact that we currently store all our interactions with the majority of services available in our world.

Big Data: Variety

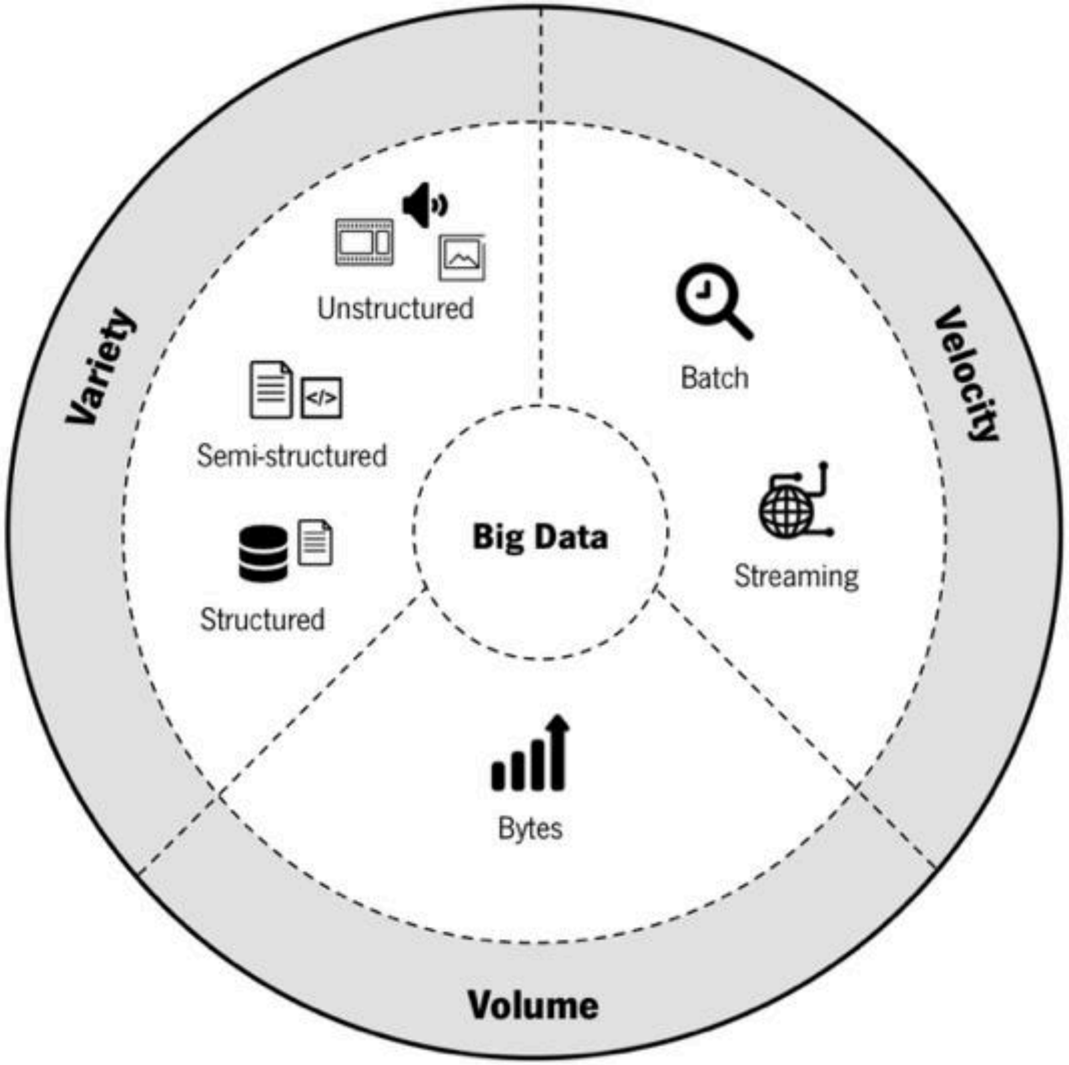
- (Gandomi & Haider, 2015) - Data can be classified as:
  - structured (e.g., transactional data, spreadsheets, relational databases;
  - semi-structured (e.g., web server logs and Extensible Markup Language - XML, JSON);
  - unstructured (e.g., social media posts, audio, video, images);

Big Data: Velocity

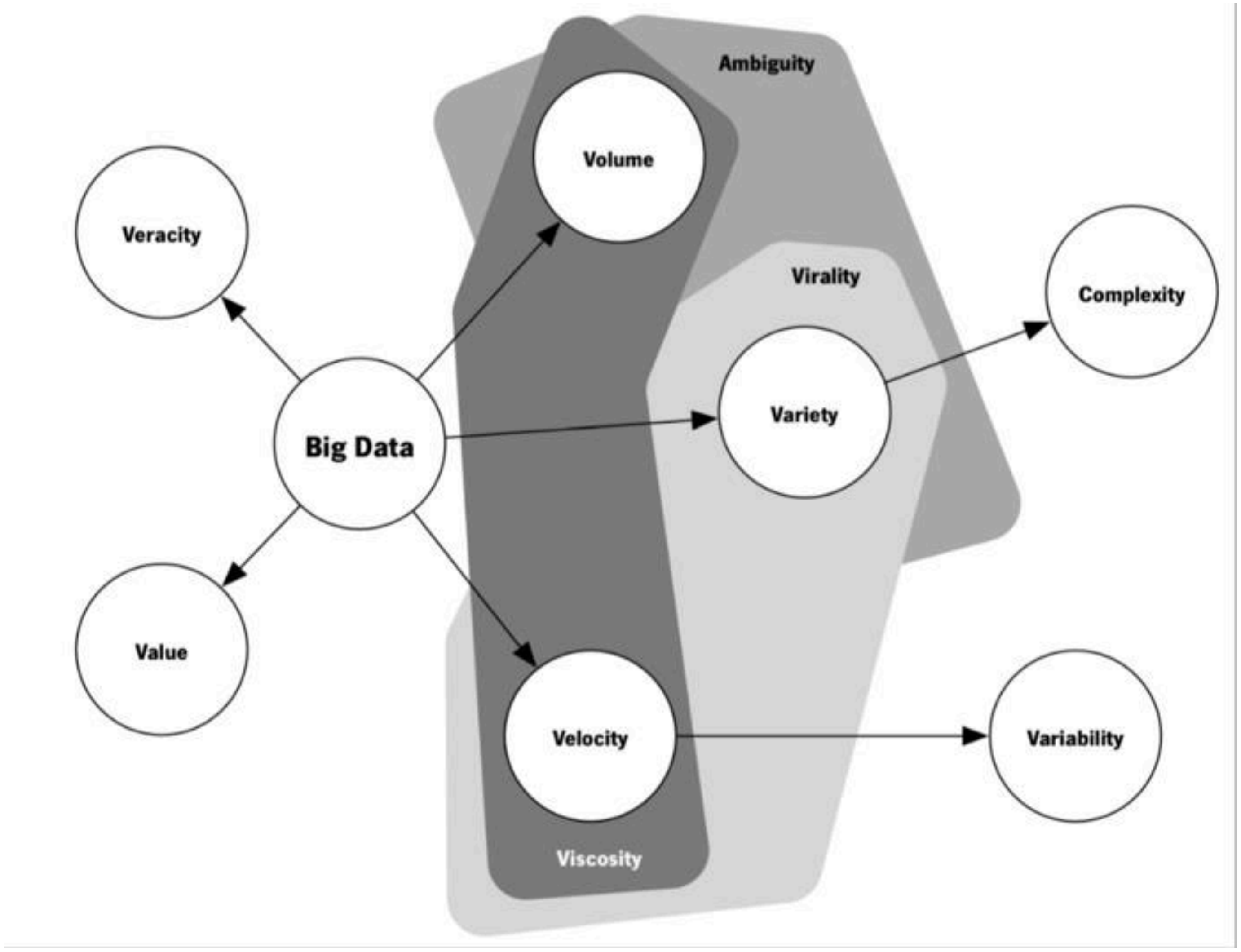
- (Gandomi & Haider, 2015) - Velocity, referring either to
  - the rate at which data is generated or
  - to the speed of analysis and decision support.

- Data can be generated at different rates, ranging from batch to real-time (streaming)

Big Data Characteristics

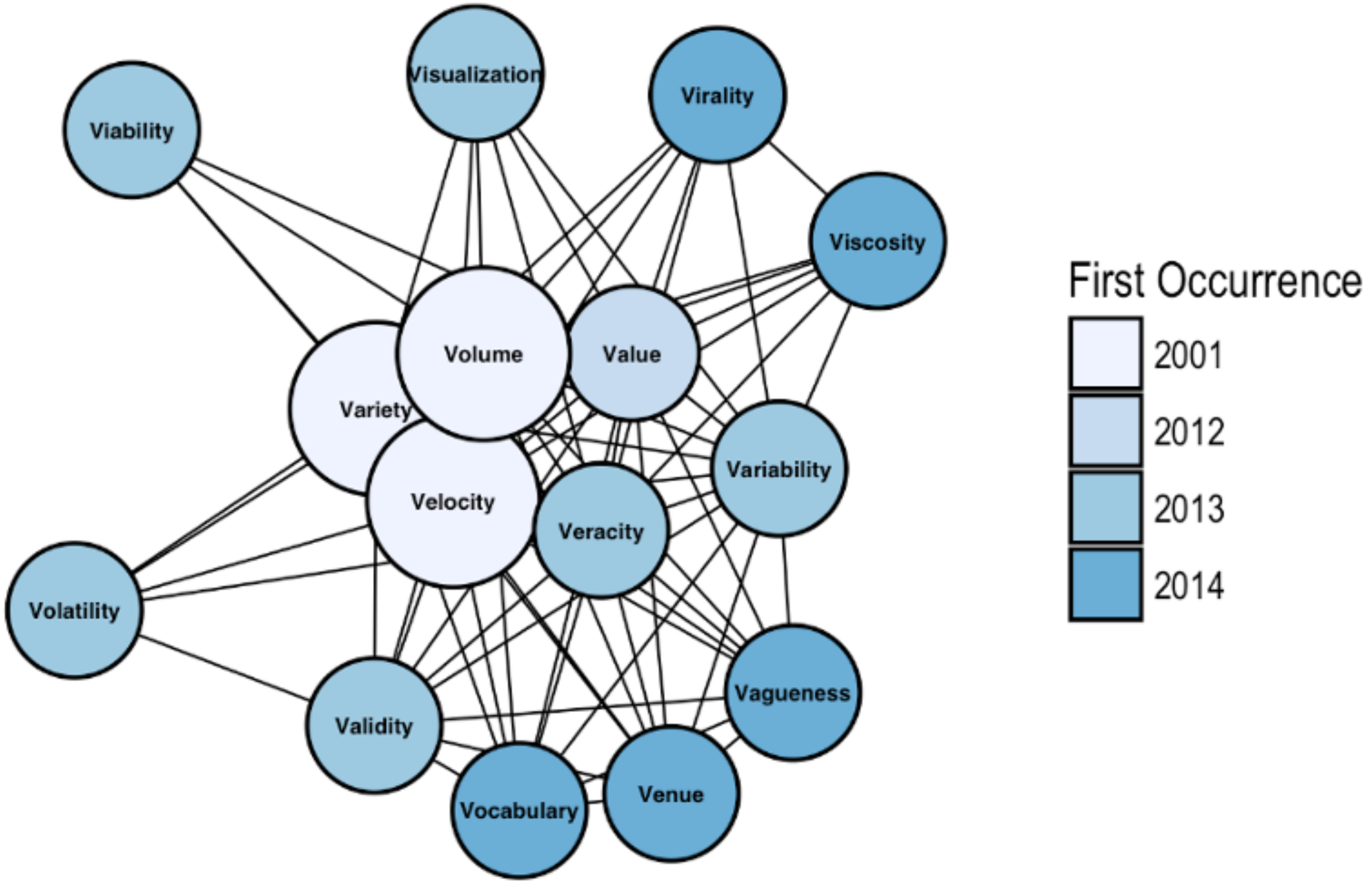


Big Data Characteristics



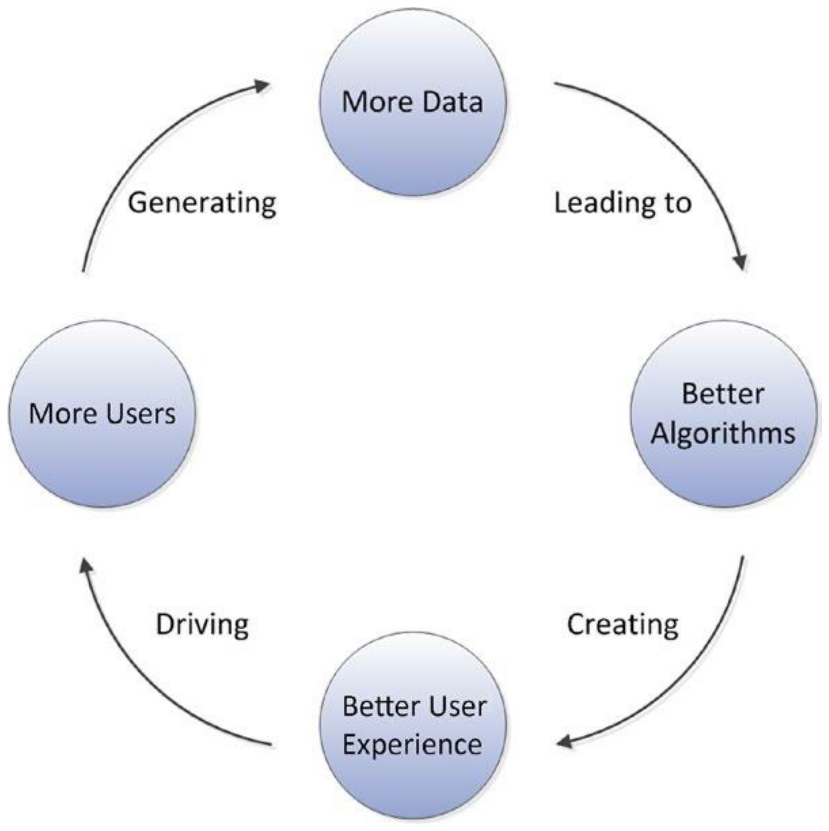
NOVA SCHOOL OF SCIENCE & TECHNOLOGY

The never ending story



The 42 V's of Big Data and Data Science (Tom Shafer, April 1st 2017)

The virtuous cycle of big data



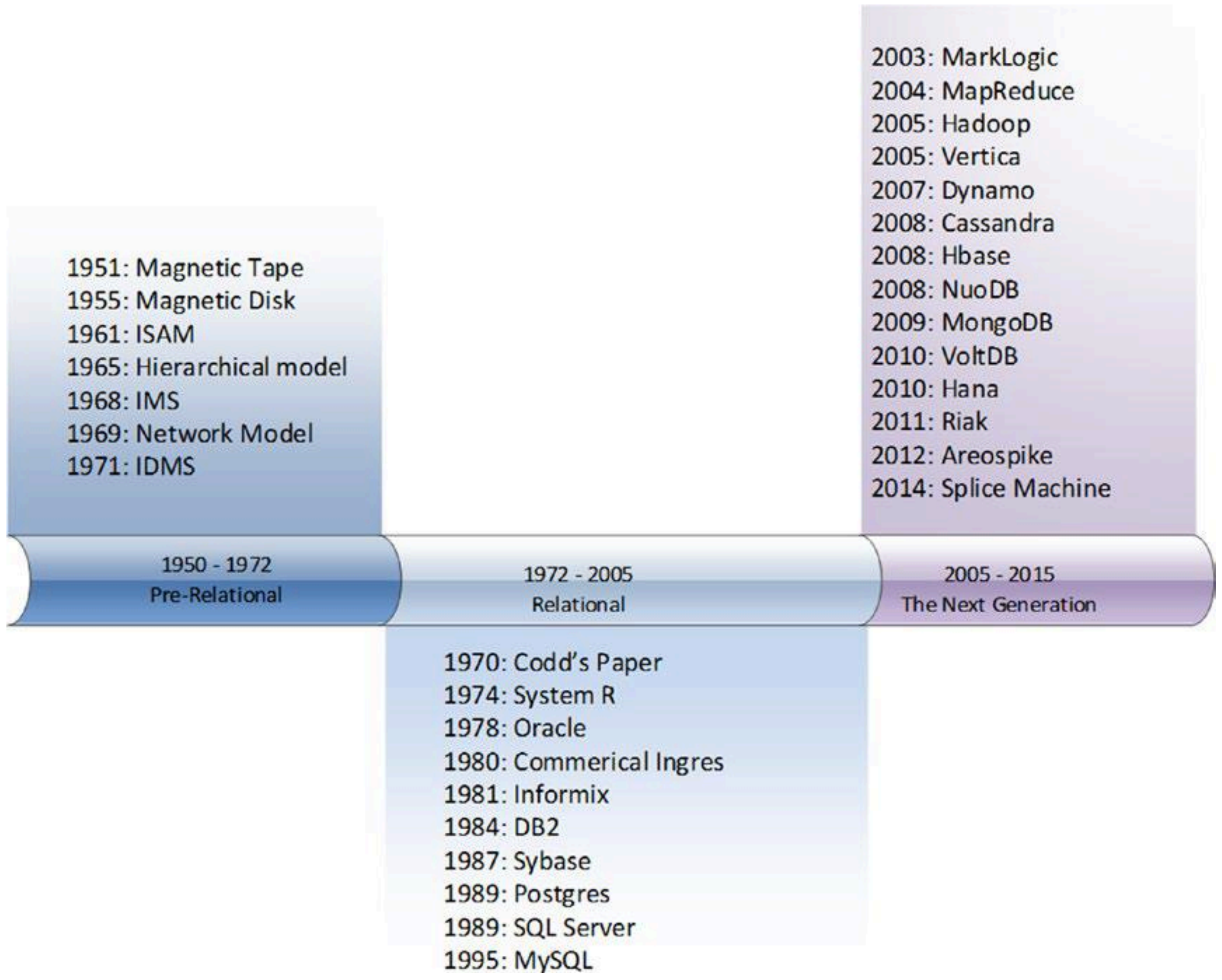
Big Data Impacts

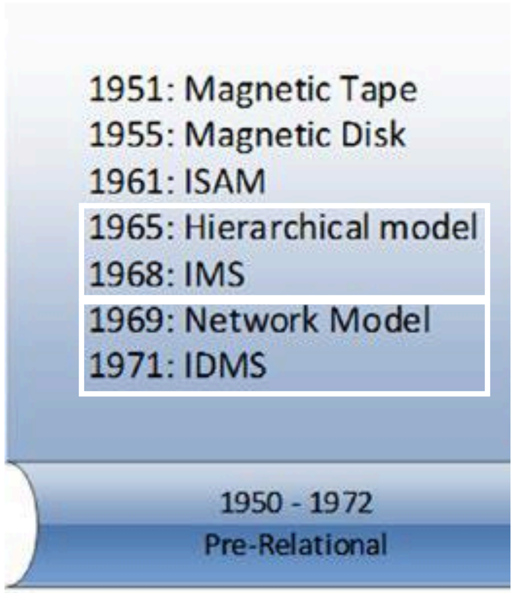
- Digital footprint (produced anyways for free)
- n = N (no sampling, but potential bias)
- Data-fusion (unstructured and incomplete)
- Real-time (dynamic)
- Machine Learning (no need for theory)

[What is Big Data]

DATABASE SYSTEMS EVOLUTION

Three database revolutions





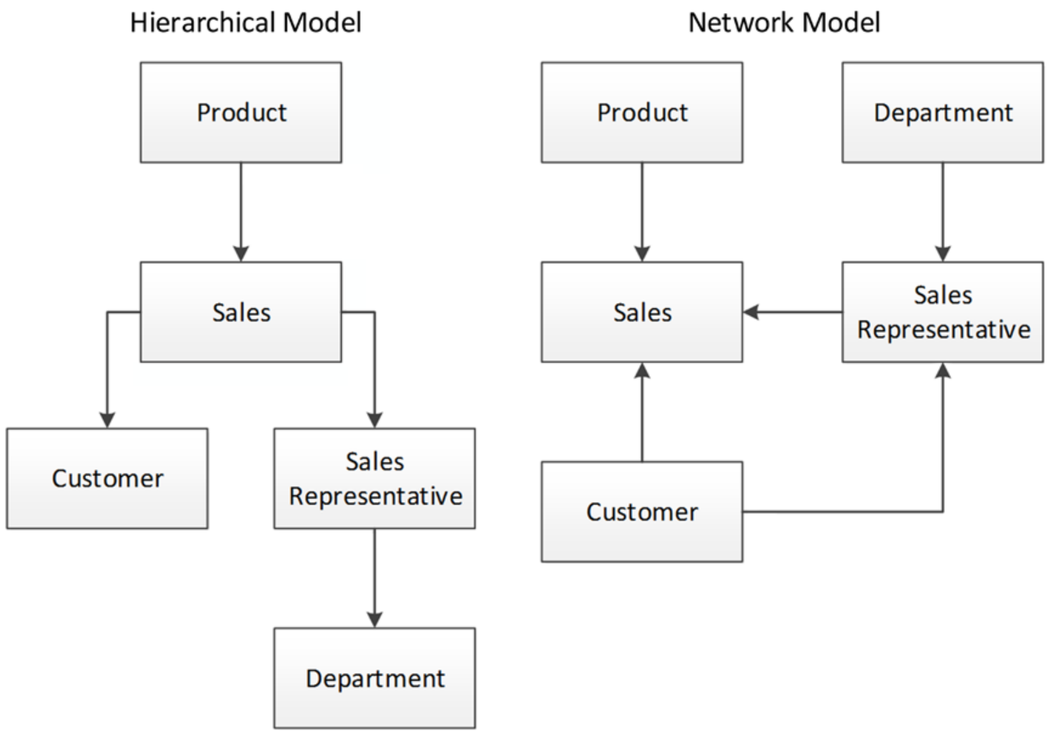
1951: Magnetic Tape
1955: Magnetic Disk
1961: ISAM
1965: Hierarchical model
1968: IMS
1969: Network Model
1971: IDMS

1950 - 1972
Pre-Relational

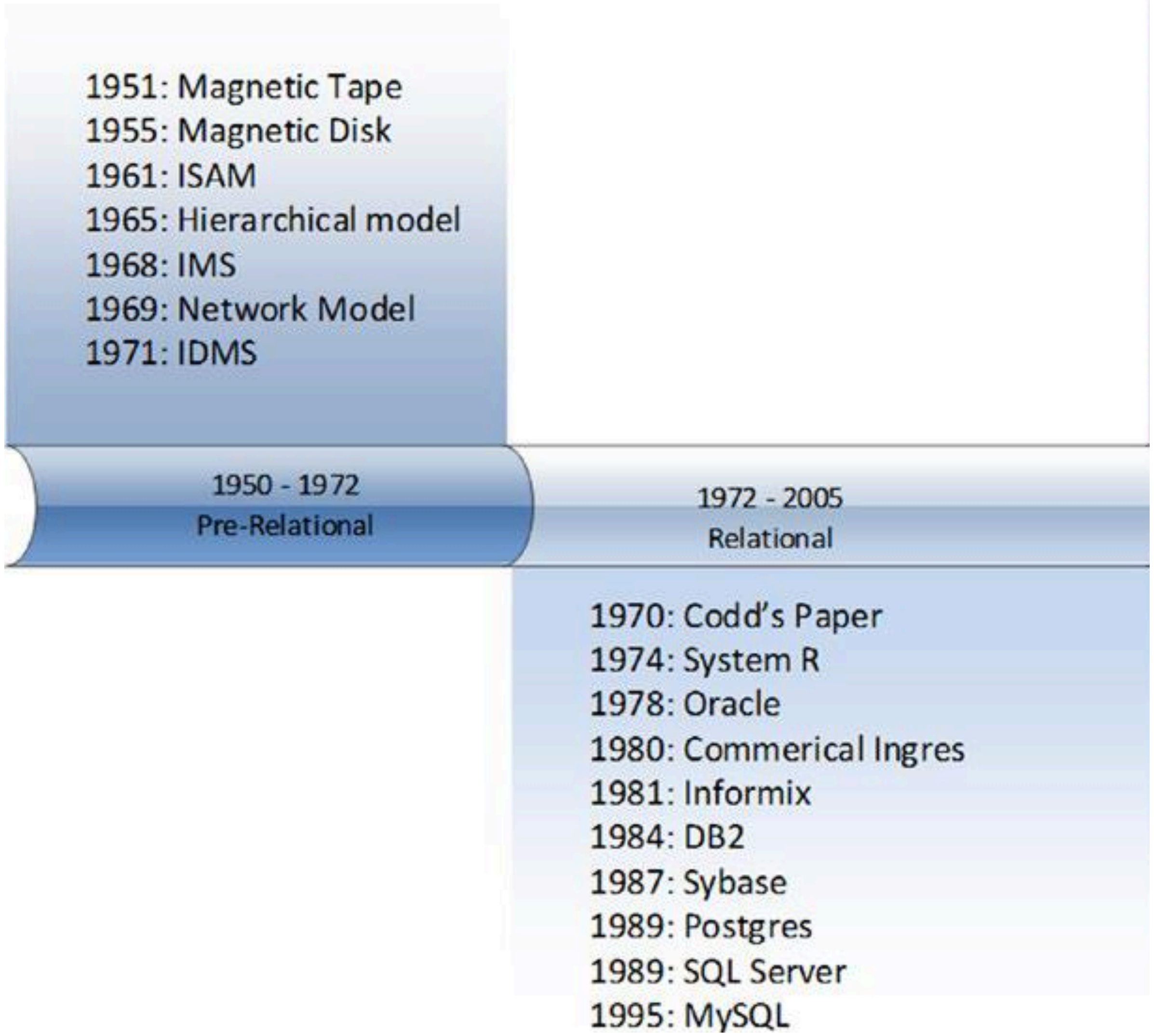
Index Sequential Access Method

First database revolution

IMS (IBM)



- Mainframe computers
- Enforce Schema and access path
- Inflexible
- Record at a time processing
- Reporting written in procedural languages (COBOL)

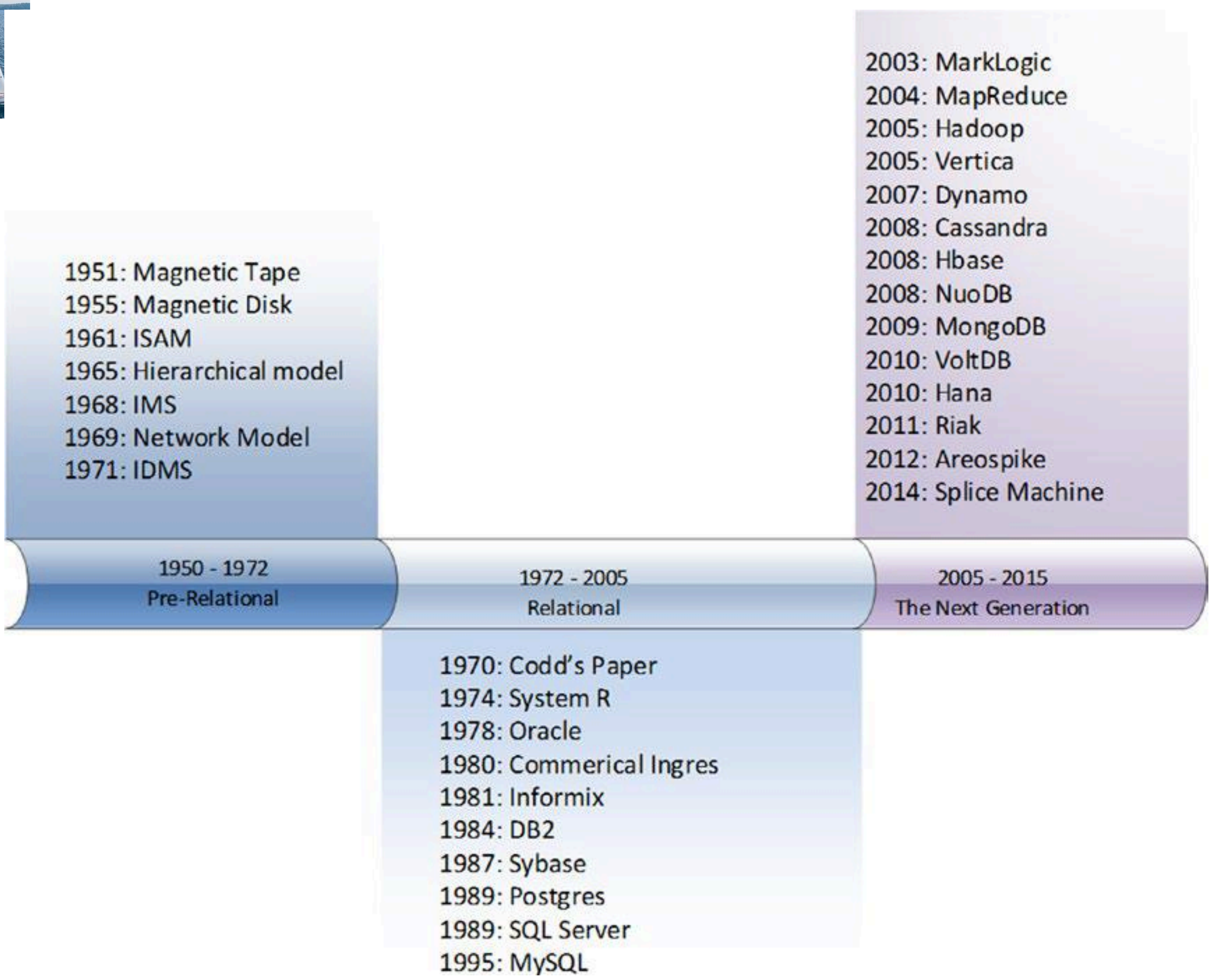


1951: Magnetic Tape
1955: Magnetic Disk
1961: ISAM
1965: Hierarchical model
1968: IMS
1969: Network Model
1971: IDMS

1950 - 1972
Pre-Relational

1972 - 2005
Relational

1970: Codd's Paper
1974: System R
1978: Oracle
1980: Commerical Ingres
1981: Informix
1984: DB2
1987: Sybase
1989: Postgres
1989: SQL Server
1995: MySQL



1950 - 1972
Pre-Relational

- 1951: Magnetic Tape
- 1955: Magnetic Disk
- 1961: ISAM
- 1965: Hierarchical model
- 1968: IMS
- 1969: Network Model
- 1971: IDMS

1972 - 2005
Relational

- 1970: Codd's Paper
- 1974: System R
- 1978: Oracle
- 1980: Commerical Ingres
- 1981: Informix
- 1984: DB2
- 1987: Sybase
- 1989: Postgres
- 1989: SQL Server
- 1995: MySQL

2005 - 2015
The Next Generation

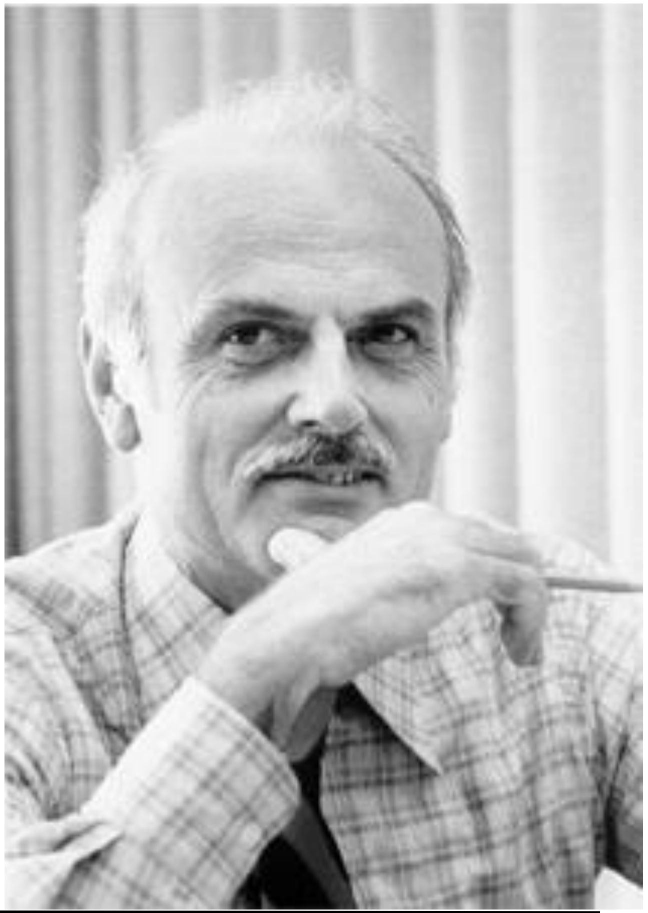
- 2003: MarkLogic
- 2004: MapReduce
- 2005: Hadoop
- 2005: Vertica
- 2007: Dynamo
- 2008: Cassandra
- 2008: Hbase
- 2008: NuoDB
- 2009: MongoDB
- 2010: VoltDB
- 2010: Hana
- 2011: Riak
- 2012: Areospike
- 2014: Splice Machine

Second database revolution

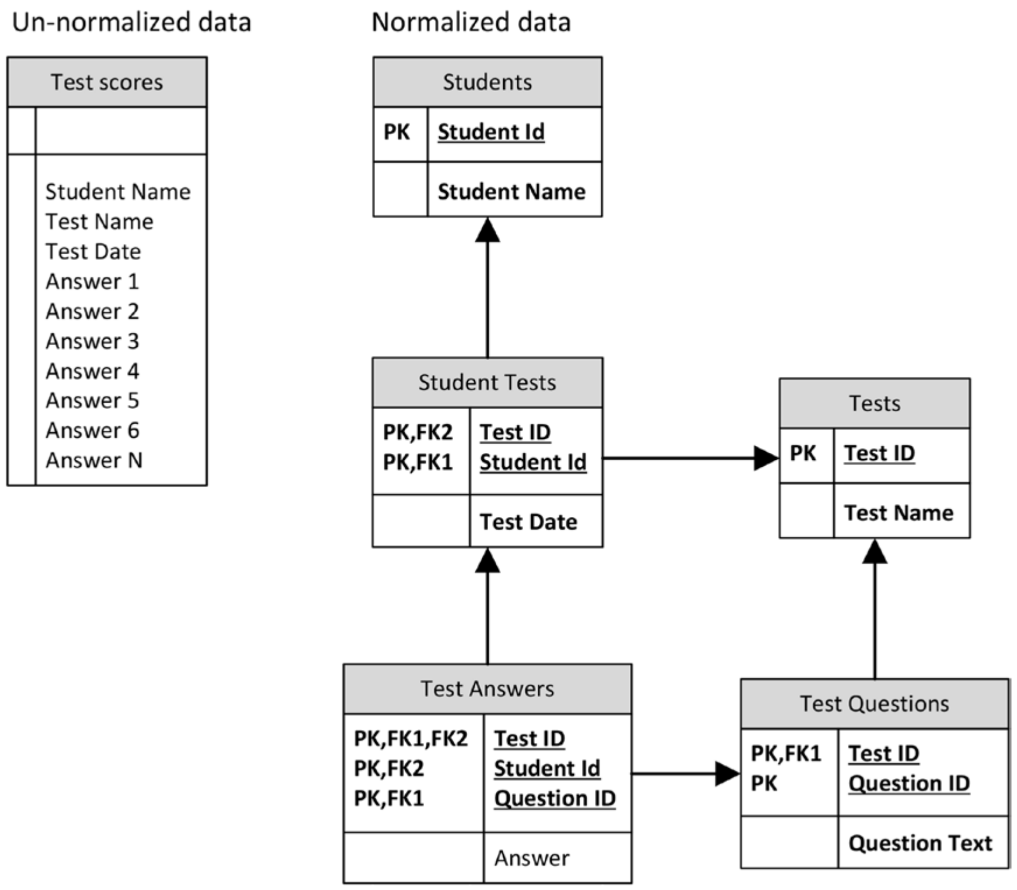
Edgar F. Codd realized at the end of 1960s that existing database systems:

- Were **hard** to use
- Lacked a theoretical foundation
- Mixed logical and physical implementations

Codd, E. F. (1970). "A relational model of data for large shared data banks" (PDF). Communications of the ACM. 13 (6): 377–387. doi:10.1145/362384.362685.



Data normalization



Relational theory defines

- Tuples
- Relations
- Constraints
- Algebra (operations)

Data should be normalized

Commercial database systems

- IBM started the development of a prototypical relational database system in 1974
- Relational databases gained importance in the 1980s with the advent of minicomputers
  - First release of Oracle in 1979
  - Mid 1980s the benefits were clear, namely SQL for reporting and analytics
  - Gained the "Database Wars" by the end of the 1980s
  - In the 1990s client-server computing was introduced and fitted well with RDBMs

Commercial database systems

- Three major characteristics of RDBMs:
  - Codd's relational model
  - SQL language
  - ACID transaction model
    - **Atomic**: The transaction is indivisible—either all the statements in the transaction are applied to the database or none are.
    - **Consistent**: The database remains in a consistent state before and after transaction execution.
    - **Isolated**: multiple transactions can be executed by one or more users simultaneously, one transaction should not see the effects of other in-progress transactions.
    - **Durable**: changes are expected to persist

OLTP systems

Operational Systems (most - OLTP)

- OLTP – On Line Transaction Processing
  - Systems that support the running activities of the organization
  - Examples:
    - Point of sale in stores;
    - ATM and Bank operations
    - e-commerce (amazon, iTunes, etc)
- Some characteristics:
  - Thousand of operations per second
  - Repeated operations dealing with small amounts of data (insert, update, remove)
  - Real Time

DW and OLAP systems

- OLAP – On Line Analytical Processing
  - Systems that provide the users the necessary capabilities to analyze many and different aspects of organization activities and its performance.
  - Examples
    - How well certain product is selling in different regions? How well is the evolution in the market from its introduction?
    - Which are the top ten selling product in each region? and globally?

Multidimensional Design

Characteristics of OLAP systems

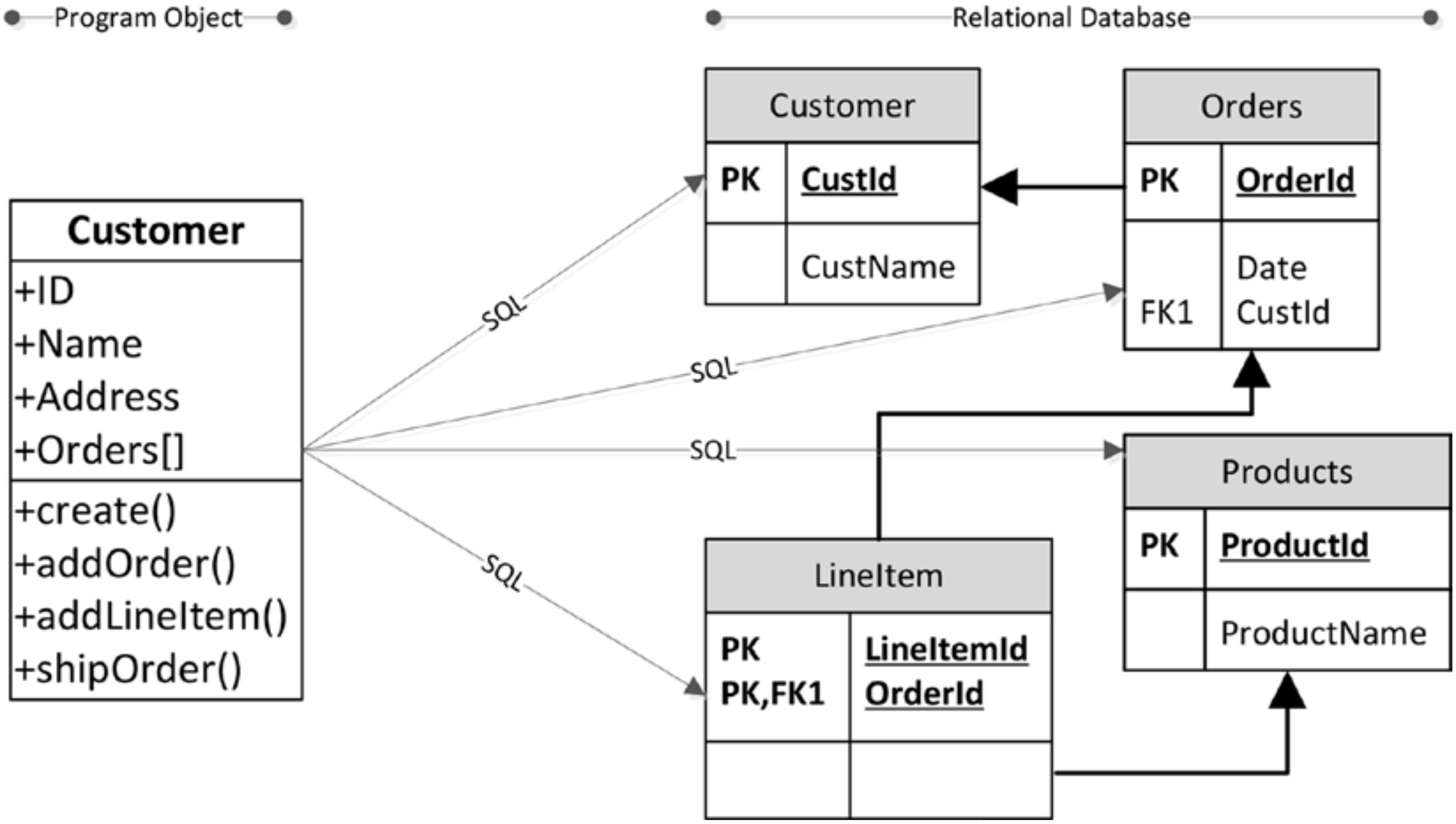
- Small number of queries (per day), when compared with OLTP systems
- Large amount of data processed in each query, in order to obtain a small output.
- It is hard to predict the queries and in general they are much more diverse, when compared with OLTP systems
- Reading and processing data but no writing.

Multidimensional Design

Object oriented programming

- OO programming resulted in productivity gains in the IT sector
- OO was a serious challenge to RDBMS due to representation mismatch of the models
- OO is akin to network models
- Could lead to performance problems…

Multiple SQL queries required for OO



NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Object oriented DBMS

- OODBMs did not succeed in the market
- Big RDBMs vendors introduced these features in their products (that were rarely used)
- This was a programmers' issue and introduced problems for the business data consumers
- Compromise attained with Object Relational Mapping frameworks
- RDBMs mastered the scene till mid 2000s

- Complex Data Types: Spatial, XML, JSON

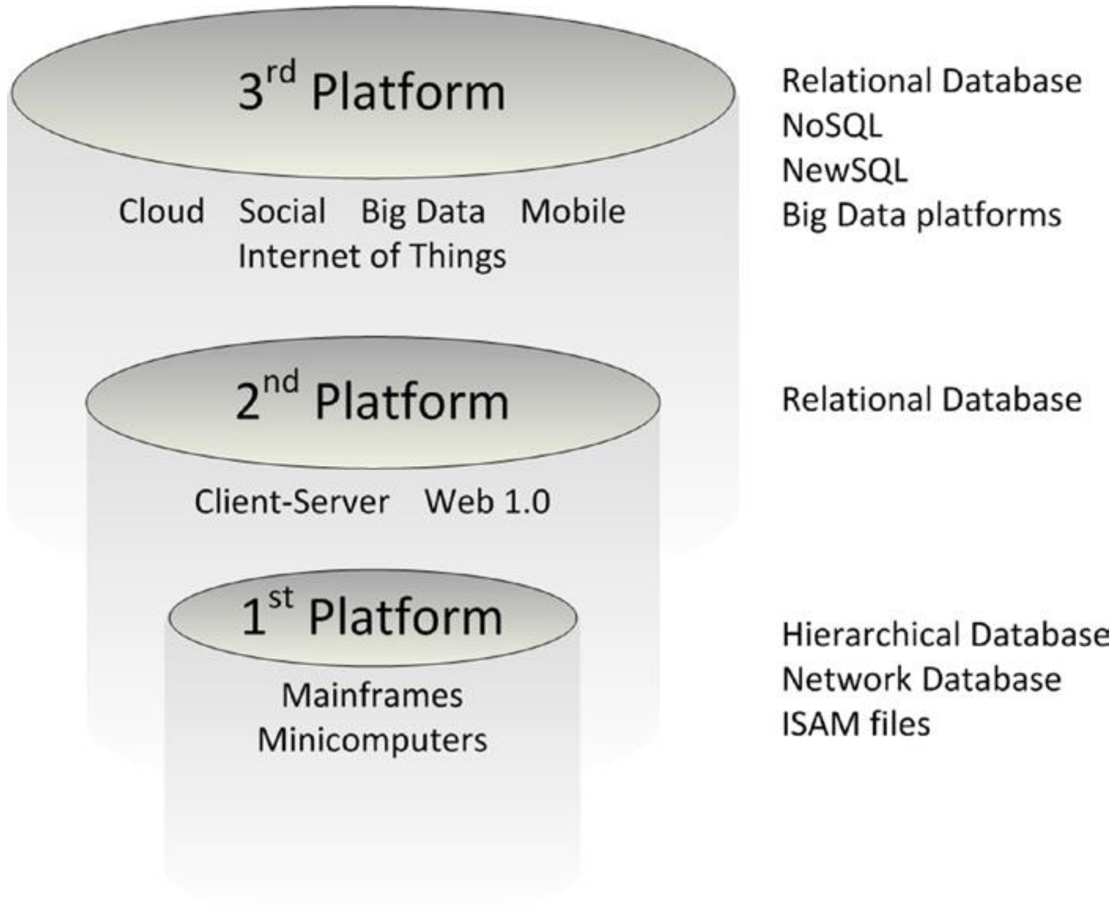
Third Database revolution

- Massive web-scale applications required new advancements in database technology
  - To represent and process search engine indexes: Hadoop, Elasticsearch
  - To handle massive online e-commerce (e.g. Amazon): DynamoDB
  - Cloud computing: AWS and DynamoDB
  - XML and JSON storage: document databases (e.g. CouchDB, MongoDB)
- NoSQL databases:
  - Reject constraints of the relational model in particular strict consistency and schemas
- NewSQL databases:
  - Enhanced or modified the fundamental principles: relational model or ACID transactions
- BigData systems: Hadoop ecosystem, including Spark

Modeling the data

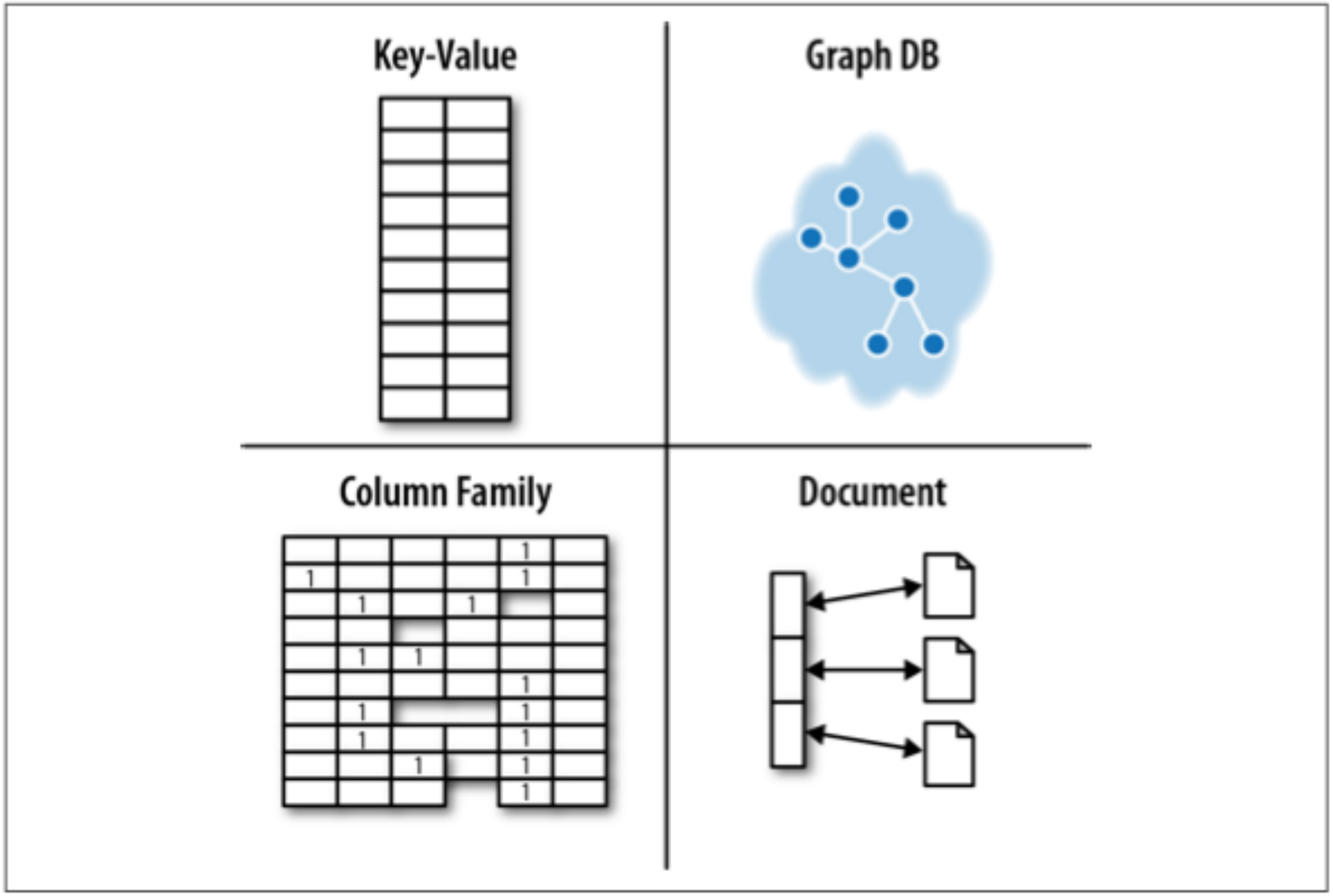
- Relational databases most of the times are not appropriate for supporting big data applications because:
  - Huge amounts of data (volume)
  - High frequency generation (velocity)
  - The structure constantly evolving and changing, storing everything (variety)
  - Mash-up of data from different sources
  - Need to explore data links that often require joins
  - High-availability requirements of Web 2.0
  - Difficult to scale horizontally
- Motivated the NoSQL movement:
  - "non SQL", "non relational" or "not only SQL"

Tthree waves of DB Techs

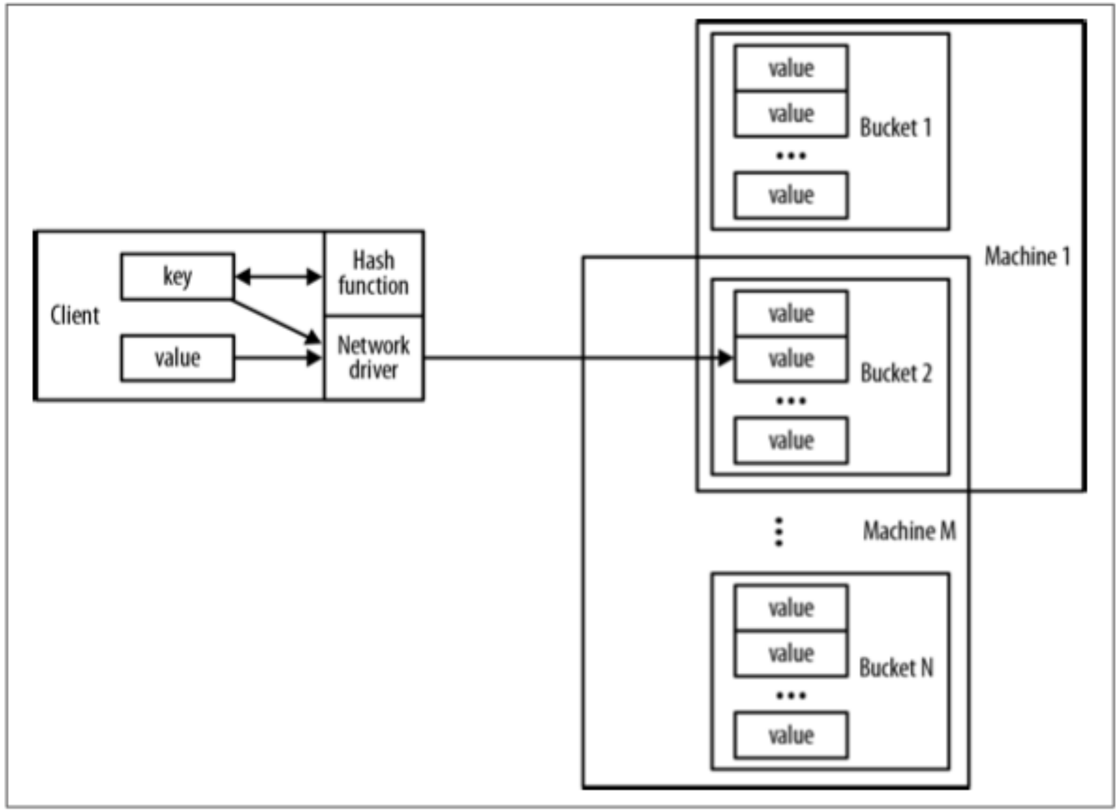


NoSQL quadrants

https://en.wikipedia.org/wiki/NoSQL



Key-value stores



Amazon DynamoDB

redis

Columnar Data Store

https://en.wikipedia.org/wiki/List_of_column-oriented_DBMSes

Columnar storage for Hadoop

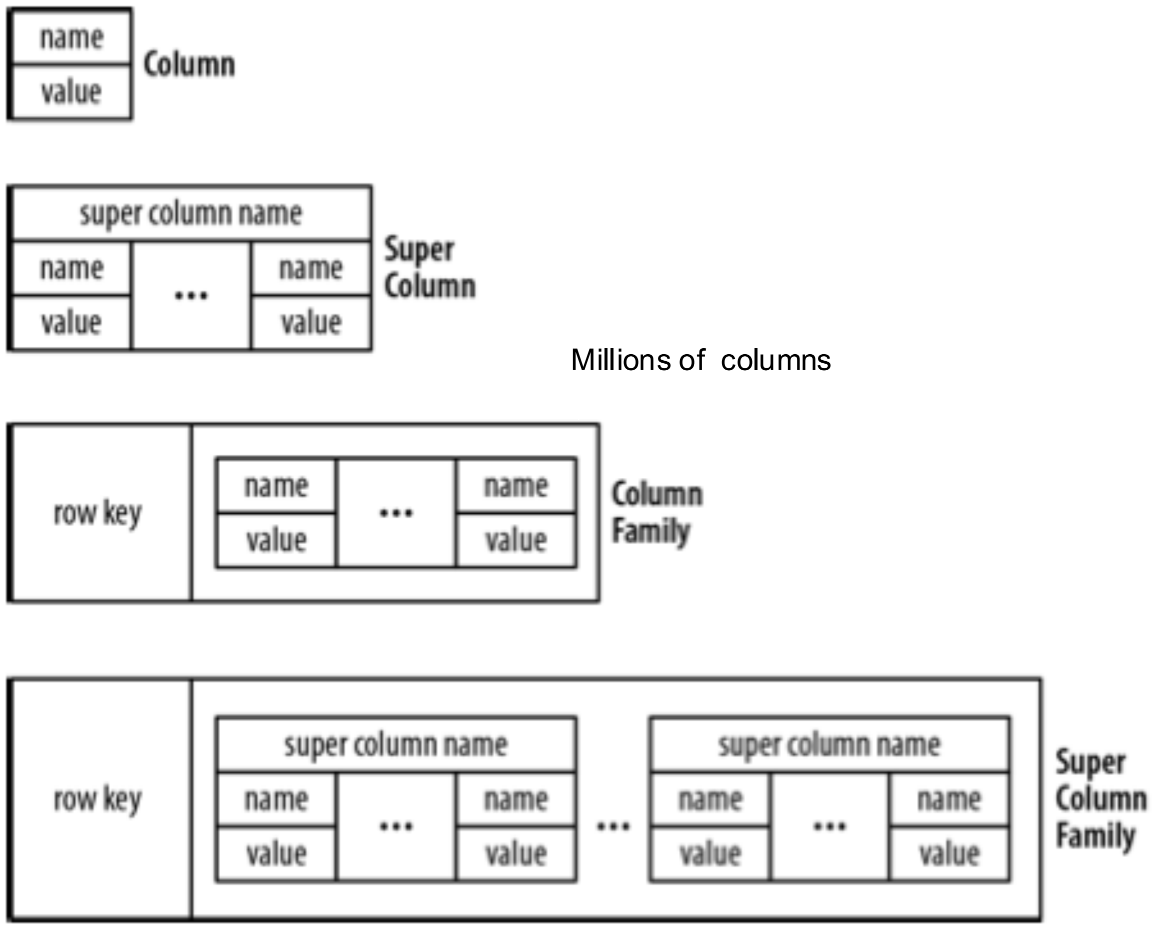
https://www.citusdata.com/getting-started/

Distributed Columnar Storage engine optimized for OLAP workloads

- Read Efficiency
- Data Compression for analytical or data warehousing

https://kudu.apache.org/

Wide-column stores are not column stores

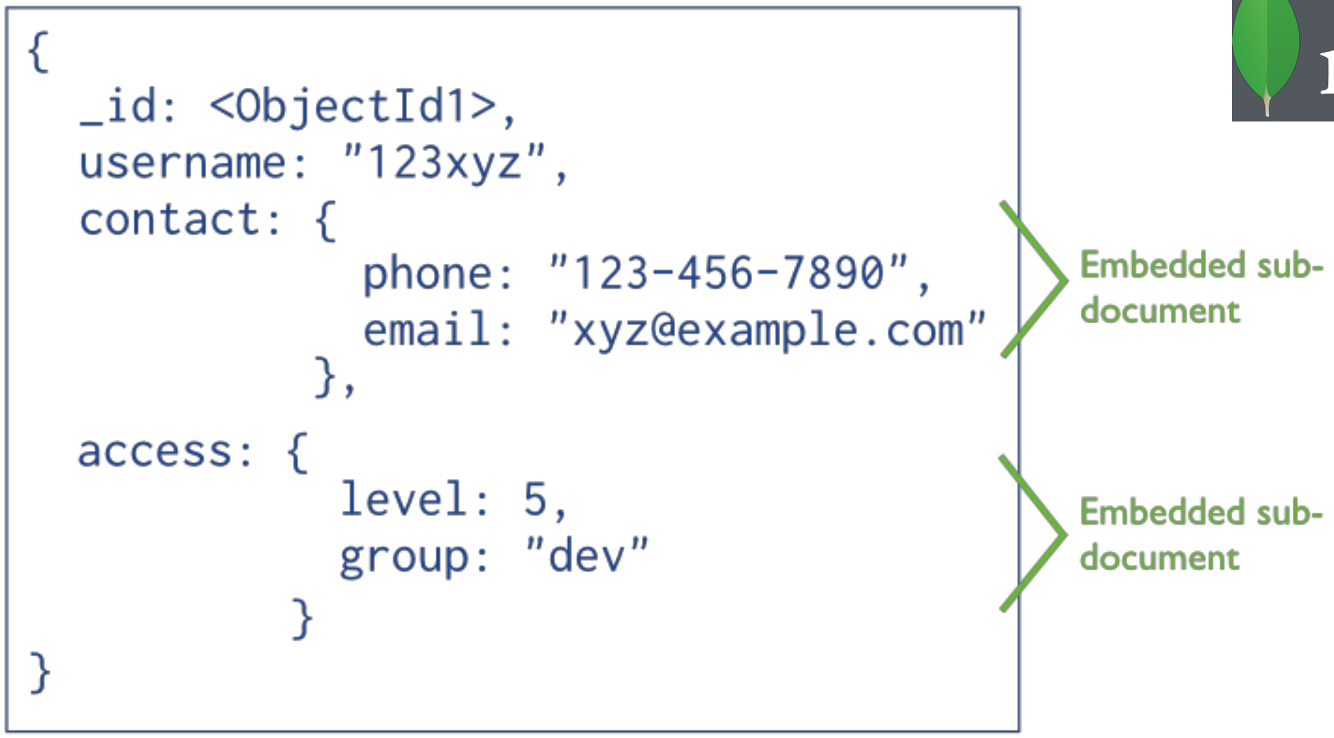


Millions of columns

APACHE CASSANDRA

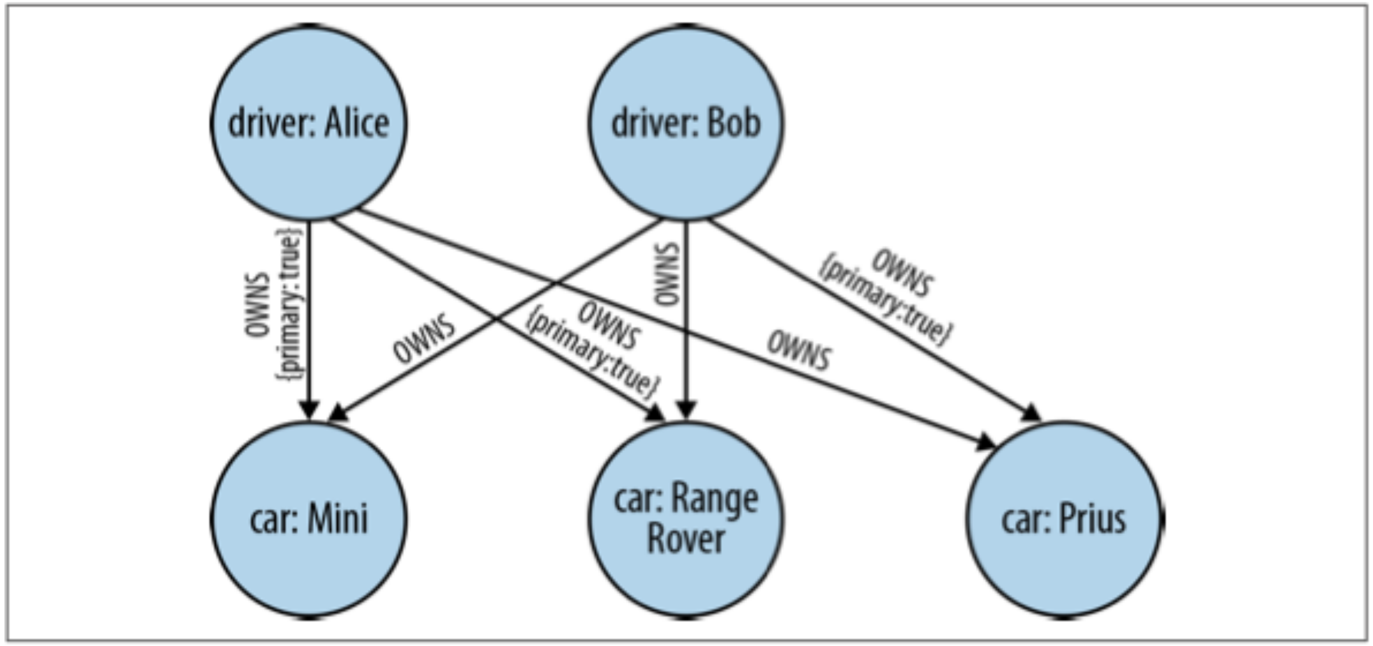
https://cwiki.apache.org/confluence/display/CASSANDRA2/DataModel

Document Stores



```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
          }
}
```

Graph databases



Main lesson

There are no silver-bullet solutions, in most of the cases you will end with a RDBMs but occasionally not.

One Size doesn't fit all!!!

Big Data Storage Systems

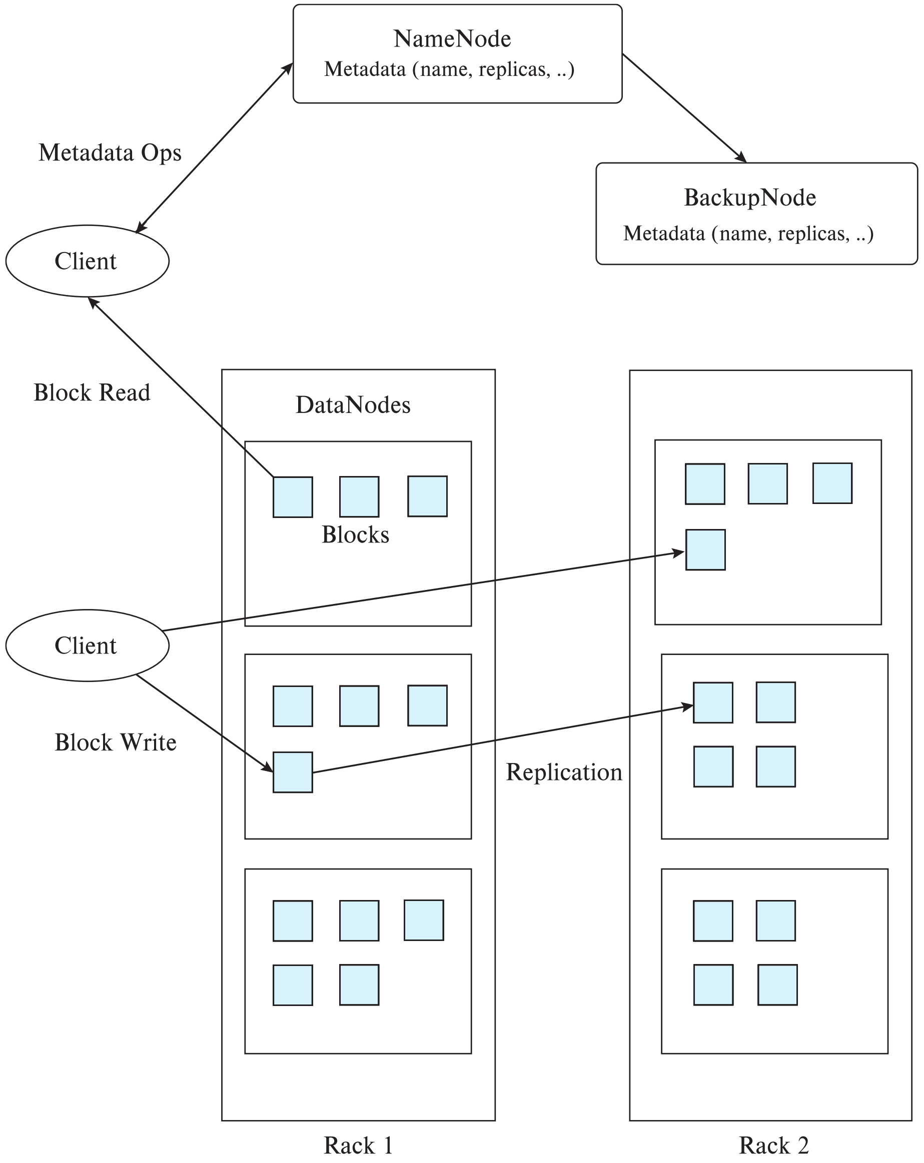
- Distributed file systems
- Sharding across multiple databases
- Key-value storage systems
- Parallel and distributed databases

Distributed File Systems

- A distributed file system stores data across a large collection of machines, but provides single file-system view
- Highly scalable distributed file system for large data-intensive applications.
  - E.g., 10K nodes, 100 million files, 10 PB
- Provides redundant storage of massive amounts of data on cheap and unreliable computers
  - Files are replicated to handle hardware failure
  - Detect failures and recovers from them
- Examples:
  - Google File System (GFS)
  - Hadoop File System (HDFS)

Hadoop File System Architecture

- Single Namespace for entire cluster
- Files are broken up into blocks
  - Typically 64 MB block size
  - Each block replicated on multiple DataNodes
- Client
  - Finds location of blocks from NameNode
  - Accesses data directly from DataNode



Hadoop Distributed File System (HDFS)

- **NameNode**
  - Maps a filename to list of Block IDs
  - Maps each Block ID to DataNodes containing a replica of the block
- **DataNode**: Maps a Block ID to a physical location on disk
- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Distributed file systems good for millions of large files
  - But have very high overheads and poor performance with billions of smaller tuples

Sharding

- **Sharding**: partition data across multiple databases
- Partitioning usually done on some ***partitioning attributes*** (also known as ***partitioning keys*** or ***shard keys*** e.g. user ID
  - E.g., records with key values from 1 to 100,000 on database 1, records with key values from 100,001 to 200,000 on database 2, etc.
- Application must track which records are on which database and send queries/updates to that database
- Positives: scales well, easy to implement
- Drawbacks:
  - Not transparent: application has to deal with routing of queries, queries that span multiple databases
  - When a database is overloaded, moving part of its load out is not easy
  - Chance of failure more with more databases
    - need to keep replicas to ensure availability, which is more work for application

Key Value Storage Systems

- Key-value storage systems store large numbers (billions or even more) of small (KB-MB) sized records
- Records are **partitioned** across multiple machines and
- Queries are routed by the system to appropriate machine
- Records are also **replicated** across multiple machines, to ensure availability even if a machine fails
  - Key-value stores ensure that updates are applied to all replicas, to ensure that their values are **consistent**

Key Value Storage Systems

- Key-value stores may store
  - **uninterpreted bytes**, with an associated key
    - E.g., Amazon S3, Amazon Dynamo
  - **Wide-table** (can have arbitrarily many attribute names) with associated key
    - Google BigTable, Apache Cassandra, Apache Hbase, Amazon DynamoDB
    - Allows some operations (e.g., filtering) to execute on storage node
  - JSON
    - MongoDB, CouchDB (document model)
- **Document stores** store semi-structured data, typically JSON
- Some key-value stores support multiple versions of data, with timestamps/version numbers

Data Representation

- An example of a JSON object is:

```
{
    "ID": "22222",
    "name": {
            "firstname: "Albert",
            "lastname: "Einstein"
    },
    "deptname": "Physics",
    "children": [
            { "firstname": "Hans", "lastname": "Einstein" },
            { "firstname": "Eduard", "lastname": "Einstein" }
    ]
}
```

Key Value Storage Systems

- Key-value stores support
  - ***put***(key, value): used to store values with an associated key,
  - ***get***(key): which retrieves the stored value associated with the specified key
  - ***delete***(key) -- Remove the key and its associated value
- Some systems also support ***range queries*** on key values
- Document stores also support queries on non-key attributes
  - See book for MongoDB queries
- Key value stores are not full database systems
  - Have no/limited support for transactional updates
  - Applications must manage query processing on their own
- Not supporting above features makes it easier to build scalable data storage systems
  - Also called **NoSQL** systems

Parallel and Distributed Databases

- Parallel databases run multiple machines (cluster)
  - Developed in 1980s, well before Big Data
- Parallel databases were designed for smaller scale (10s to 100s of machines)
  - Did not provide easy scalability
- **Replication** used to ensure data availability despite machine failure
  - But typically restart query in event of failure
    - Restarts may be frequent at very large scale
    - Map-reduce systems (coming up next) can continue query execution, working around failures

Replication and Consistency

- **Availability** (system can run even if parts have failed) is essential for parallel/distributed databases
  - Via replication, so even if a node has failed, another copy is available
- **Consistency** is important for replicated data
  - All live replicas have same value, and each read sees latest version
  - Often implemented using majority protocols
    - E.g., have 3 replicas, reads/writes must access 2 replicas
      - Details in chapter 23
- **Network partitions** (network can break into two or more parts, each with active systems that can't talk to other parts)
- In presence of partitions, cannot guarantee both availability and consistency
  - Brewer's CAP "Theorem"

Replication and Consistency

- Very large systems will partition at some point
  - Choose one of consistency or availability
- Traditional database choose consistency
- Most Web applications choose availability
  - Except for specific parts such as order processing
- More details later, in Chapter 23

The MapReduce Paradigm

- Platform for reliable, scalable parallel computing
- Abstracts issues of distributed and parallel environment from programmer
  - Programmer provides core logic (via map() and reduce() functions)
  - System takes care of parallelization of computation, coordination, etc.
- Paradigm dates back many decades
  - But very large scale implementations running on clusters with 10^3 to 10^4 machines are more recent
  - Google Map Reduce, Hadoop, ..
- Data storage/access typically done using distributed file systems or key-value stores
- Discussed in other courses

GRAPH DATABASES

Graph Data Model

- Graphs are a very general data model
- ER model of an enterprise can be viewed as a graph
  - Every entity is a node
  - Every binary relationship is an edge
  - Ternary and higher degree relationships can be modelled as binary relationships

Graph Data Model (Cont.)

- Graphs can be modelled as relations
  - *node*(*ID, label, node_data*)
  - *edge*(*fromID, toID, label, edge_data*)
- Above representation too simplistic
- Graph databases like Neo4J can provide a **graph view of relational schema**
  - Relations can be identified as representing either nodes or edges
- Query languages for graph databases make it
  - easy to express queries requiring edge traversal
  - allow efficient algorithms to be used for evaluation

Graph Data Model (Cont.)

- Suppose
  - Relations *instructor* and *student* are nodes, and
  - Relation *advisor* represents edges between instructors and student
- Query in Neo4J:
  match (*i*:*instructor*)<-[:*advisor*]-(*s*:*student*)
  where *i*.*dept name*= 'Comp. Sci.'
  return *i*.ID **as** ID, *i*.*name* **as** *name*, **collect**(*s*.*name*) **as** *advisees*
- **match** clause matches nodes and edges in graphs
- Recursive traversal of edges is also possible
  - Suppose *prereq*(*course_id, prereq_id*) is modeled as an edge
  - Transitive closure can be done as follows:

    match (*c1*:*course*)-[:*prereq* *1..]->(*c2*:*course*)
    return *c*1.*course id*, *c2*.*course id*

Parallel Graph Processing

- Very large graphs (billions of nodes, trillions of edges)
  - Web graph: web pages are nodes, hyper links are edges
  - Social network graph: people are nodes, friend/follow links are edges
- Two popular approaches for parallel processing on such graphs
  - Map-reduce and algebraic frameworks
  - **Bulk synchronous processing (BSP)** framework
- Multiple iterations are required for any computations on graphs
  - Map-reduce/algebraic frameworks often have high overheads per iteration
  - BSP frameworks have much lower per-iteration overheads
- Google’s Pregel system popularized the BSP framework
- Apache Giraph is an open-source version of Pregel
- Apache Spark’s GraphX component provides a Pregel-like API

Bulk Synchronous Processing

- Each vertex (node) of a graph has data (state) associated with it
  - Vertices are partitioned across multiple machines, and state of node kept in-memory
- Analogous to map() and reduce() functions, programmers provide methods to be executed for each node
  - Node method can send messages to or receive messages from neighboring nodes
- Computation consists of multiple iterations, or supersteps
- In each **superstep**
  - Nodes process received messages
  - Update their state, and
  - Send further messages or vote to halt
  - Computation ends when all nodes vote to halt, and there are no pending messages;

End of Chapter 10