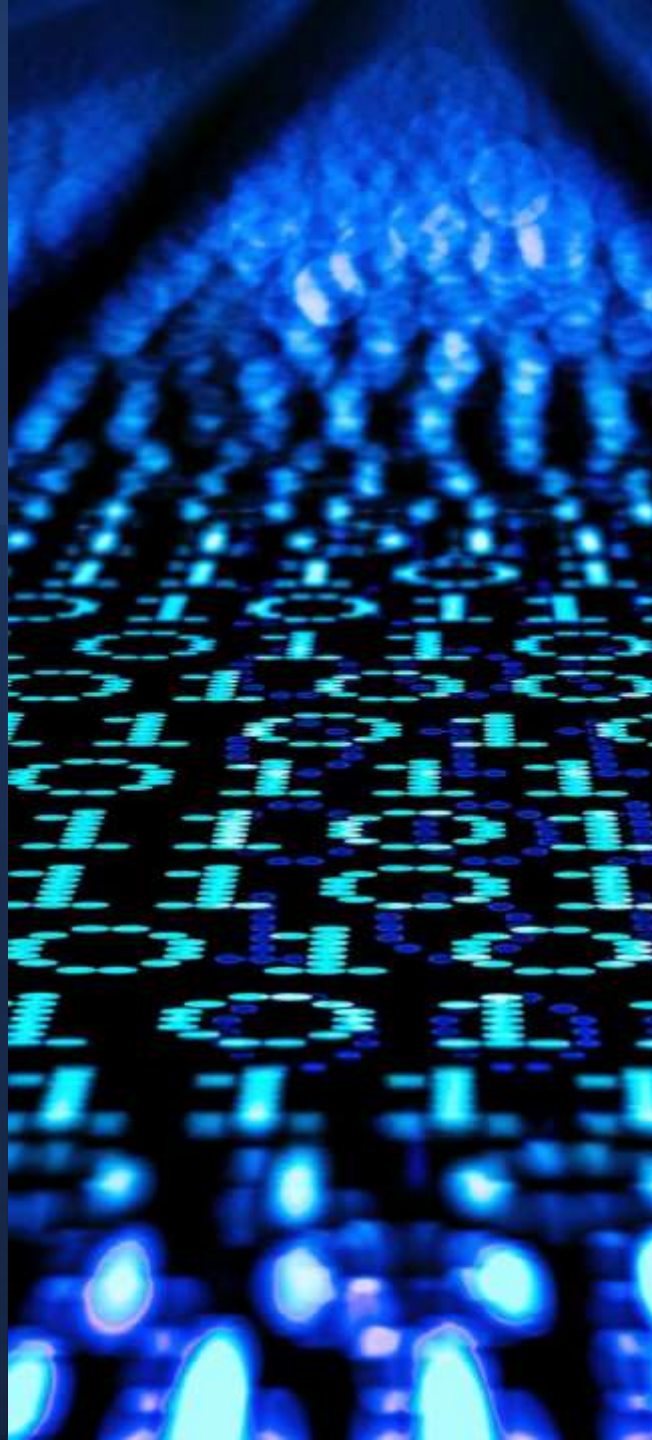
Big Data and NoSQL Systems

Lecture 1



Understanding Big Data

Lecture Outlines

- Why Big Data?
- The Definition of Big Data
- Characteristics/Challenges of Big Data
- Classification of Big Data
- Applications of Big Data
- Enterprise Technologies for Big Data and Business Intelligence

Why Big Data?

The Model of Generating/Consuming Data has Changed

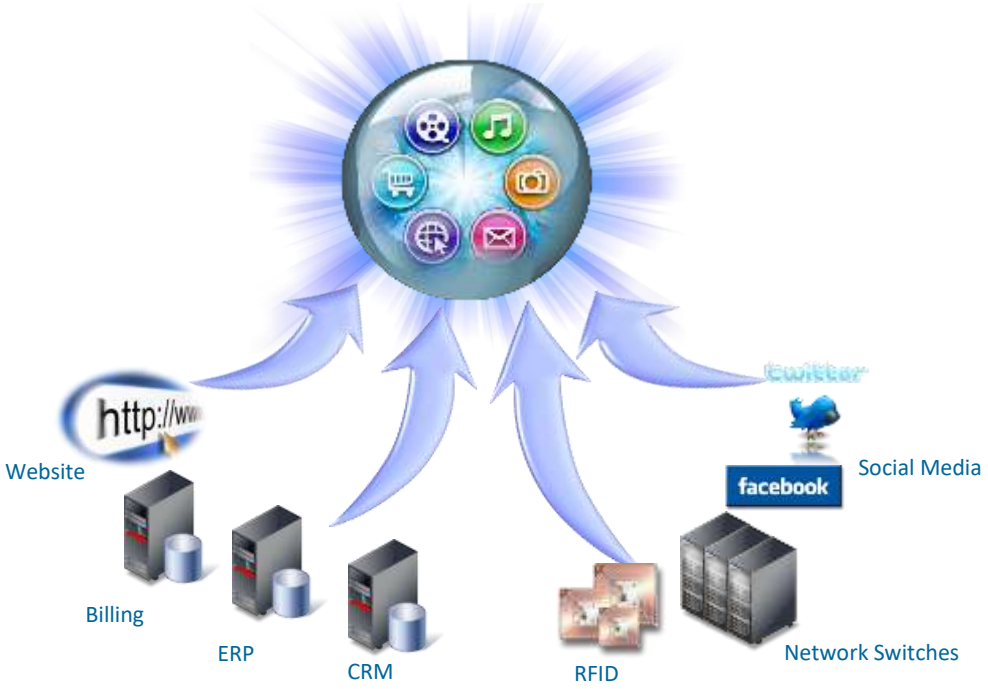
Old Model: Few companies are generating data, all others are consuming data

New Model: all of us are generating data, and all of us are consuming data

Why Big Data?

- 2.5 quintillion (10^{18}) bytes of data are generated every day!

  - Social media sites
  - Sensors
  - Digital photos
  - Business transactions
  - Location-based data



Source: *IBM* *http://www-01.ibm.com/software/data/bigdata/*

Why Big Data?

It is all about deriving *new insight* for the business

- Big data itself isn't new – it is been here for a while and growing exponentially.
- What is new is the technology to process and analyze it.
  - Increase of storage capacities.
  - Increase of processing power.
  - Availability of data.

Available technology can cost effectively manage and analyze all available data in its native form unstructured, structured, streaming

What is Big Data?

Key idea: "Big" is relative!
"*Difficult Data*" is perhaps more apt!

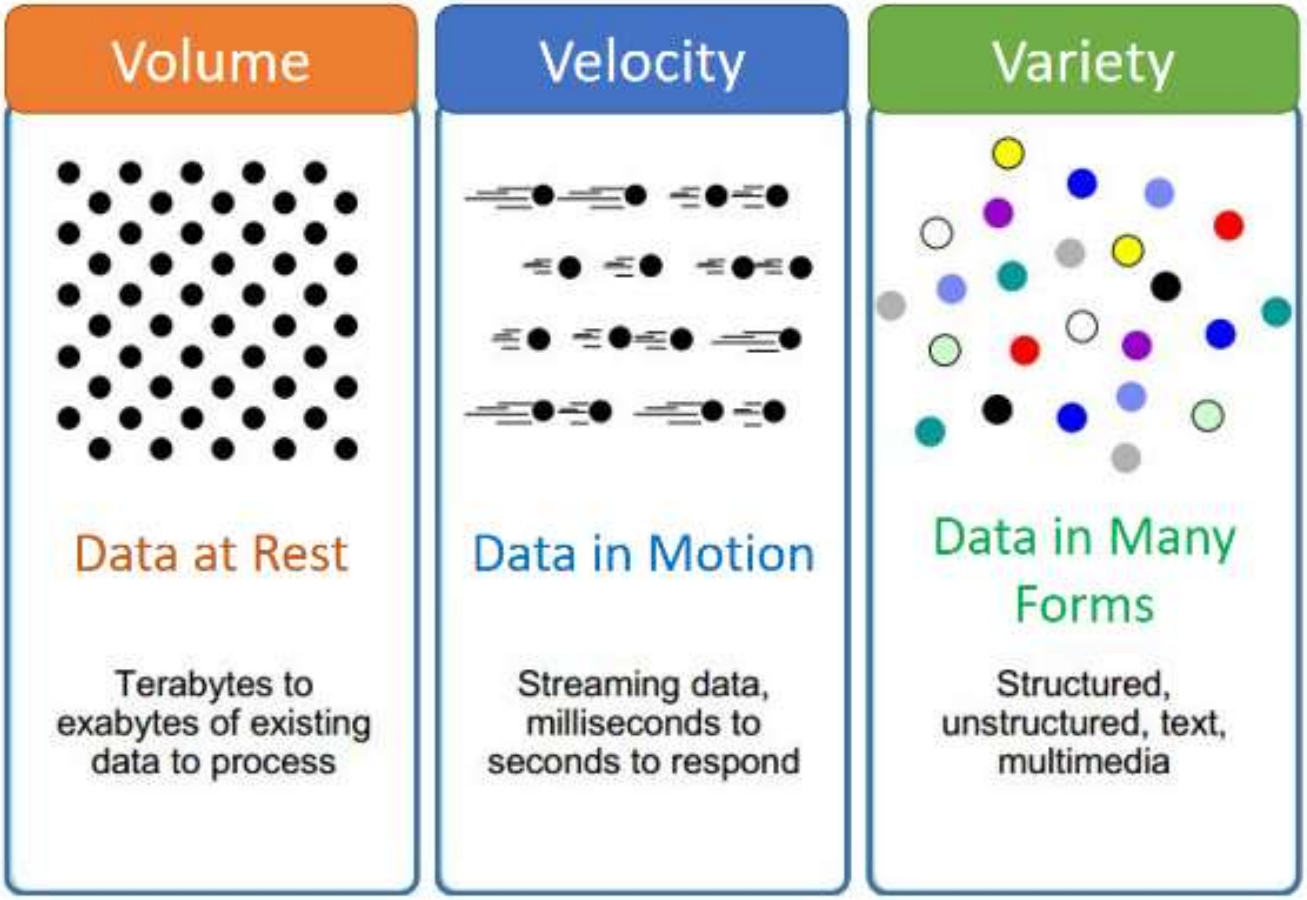
Bill Howe, UW

- **Wikipedia**
  - **Big data** is a term for datasets that are so large or complex that traditional data processing applications are inadequate to deal with them.
  - Challenges include *analysis*, *capture*, *data curation*, *search*, *sharing*, *storage*, *transfer*, *visualization*, *querying*, *updating* and *information privacy*
- **Gartner**
  - Big data is a popular term used to acknowledge the *exponential growth*, *availability* and *use* of information in the data-rich landscape of tomorrow.
- **Academia**
  - Big Data is any data that is expensive to manage and hard to extract value from."

Michael Franklin
Thomas M. Siebel Center
Professors of Computer Science
Director of the Algorithms, Machines and People Lab
University of Berkeley

Characteristics of Big Data

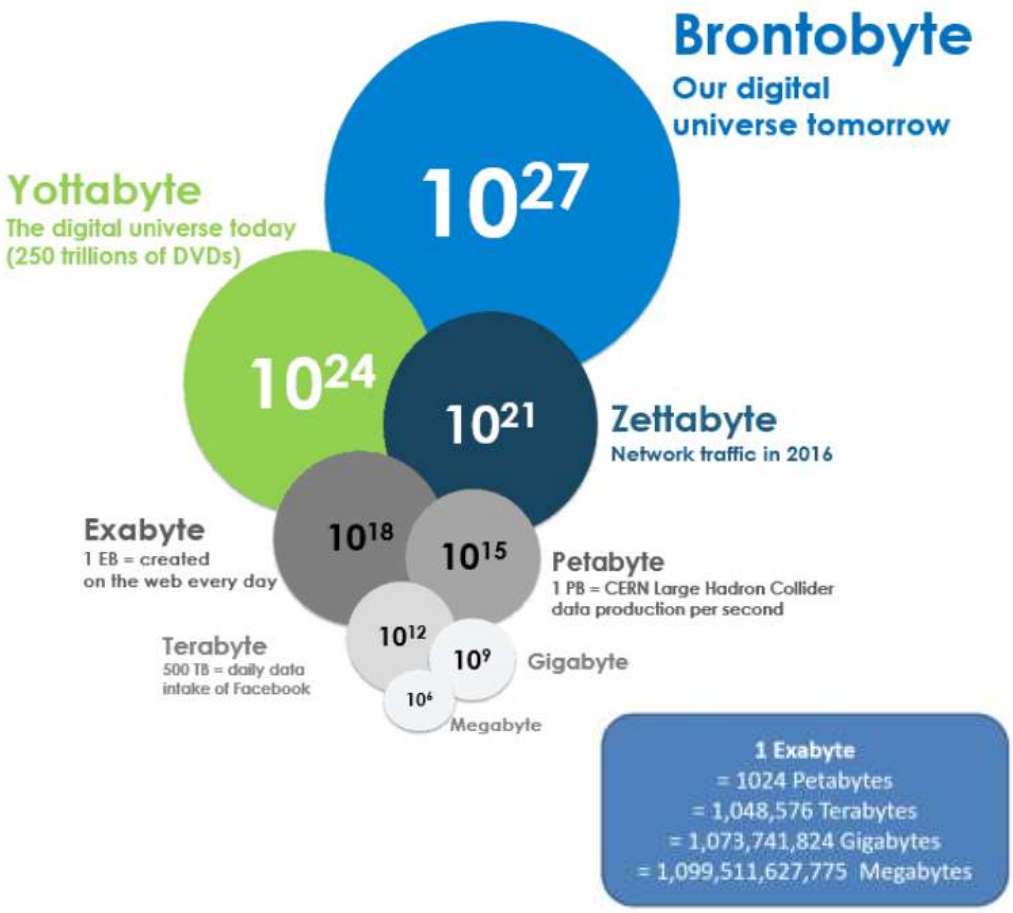
3 Vs



Volume

Super exponential growth in data volume

- Refers to the vast amount of data generated every second.
- It is not only Terabytes, but Zettabytes or Brontobytes.
- This makes most datasets too large to store and analyse using traditional database technology.
- Big data tools use distributed systems to store and analyse data that are dotted around anywhere in the world.



https://www.virtualb.it/en/blog/big-data-or-small-data-the-value-its-not-about-quantity-but-about-quality/

Volume

Super exponential growth in data volume

- High data volumes impose:
  - Distinct data storage and processing demands.
  - Additional data preparation, curation and management processes

Velocity

Data can arrive at fast speeds

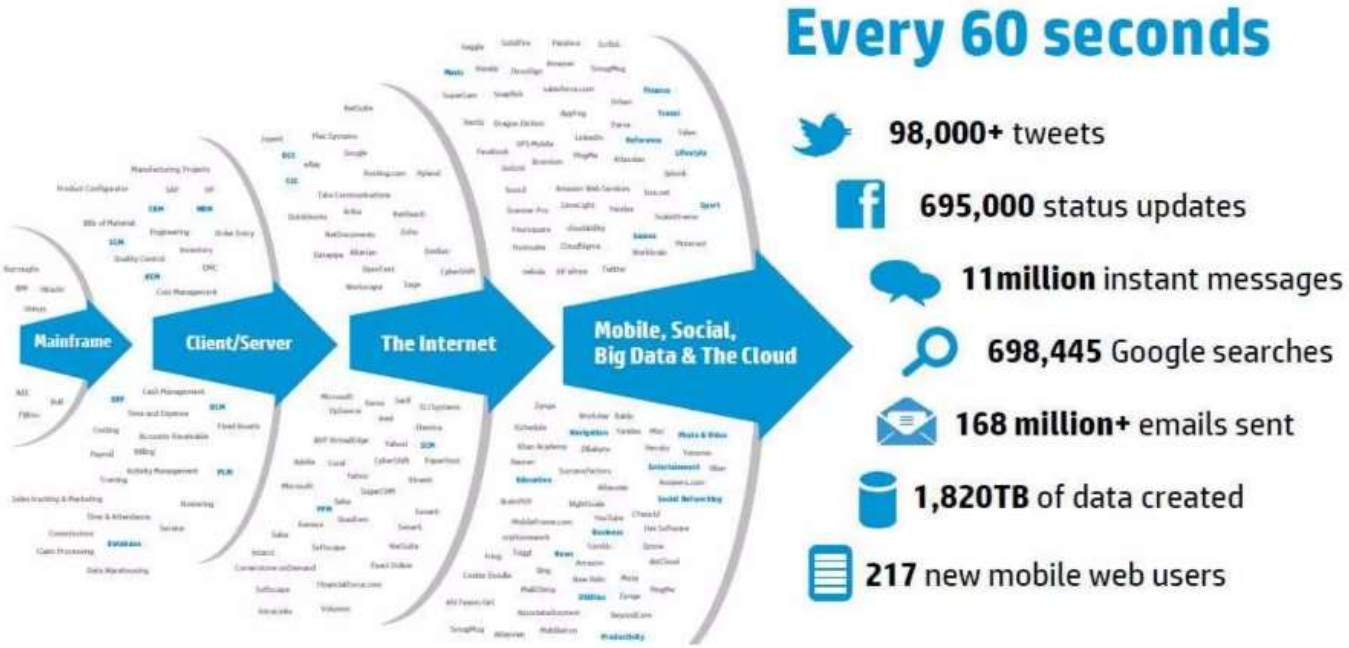
- The most challenging V to conquer, since it has a compounding effect on the other Vs.
- The velocity of data translates into the amount of time it takes for the data to be processed once it arrives .
- Coping with the fast inflow of data requires to design highly elastic and available data processing solutions and corresponding data storage capabilities.

The 3 V's of Big Data: Velocity Remains A Challenge for Many.
Dennis Duckworth | Jan 4, 2023

Velocity

Data can arrive at fast speeds

- It is a challenge to manage, analyze, summarize, visualize, and discover knowledge from the collected data ***in a timely manner and in a scalable fashion***



https://www.researchgate.net/figure/Examples-of-big-data-velocity_fig3_313400371

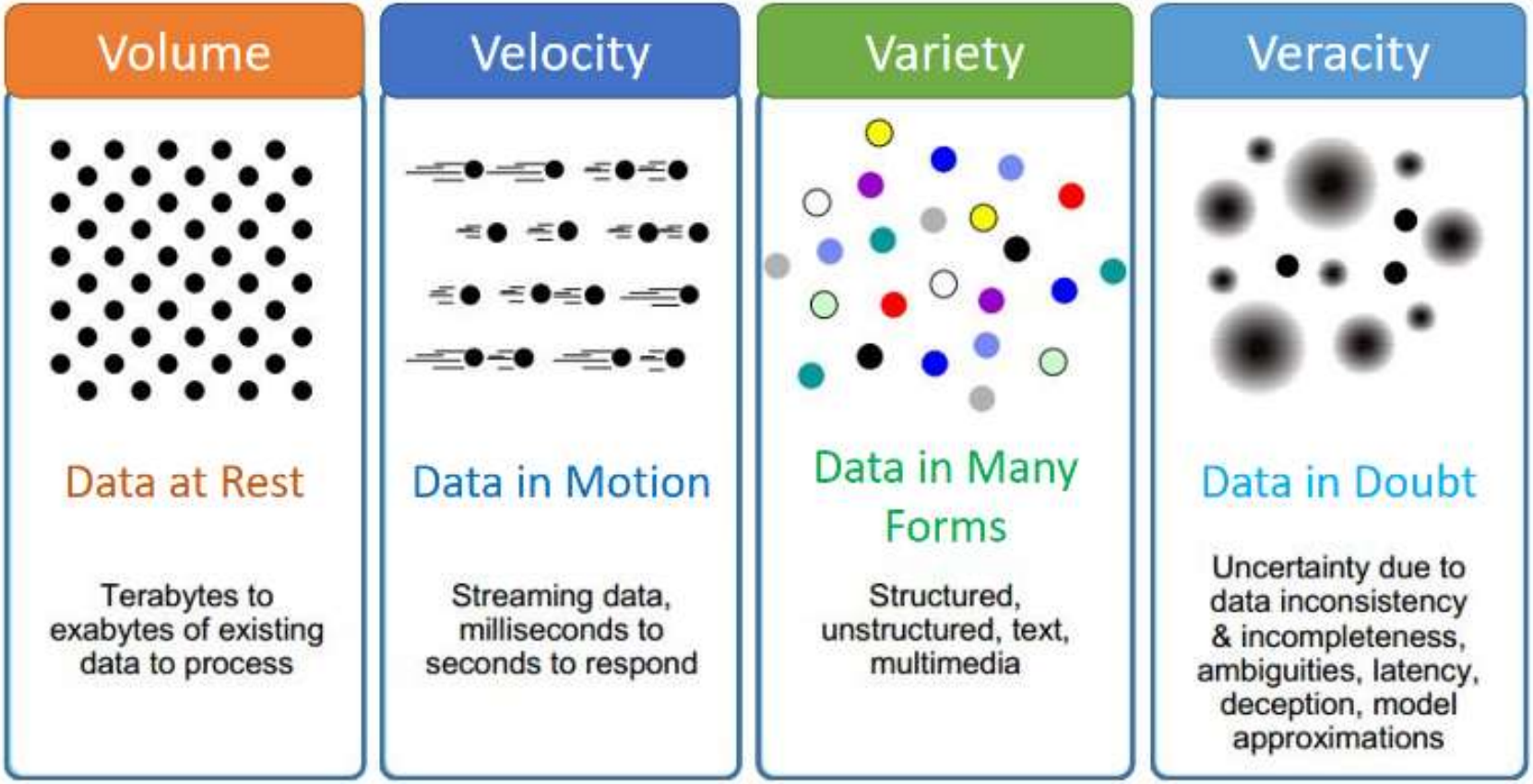
Variety

Multiple formats and types of data

- Refers to the different types of data we need to use. In fact, 80% of the world's data is unstructured.
- Data variety brings challenges in terms of data integration, transformation, processing, and storage.

- Structured data
  - Financial transactions, students' records, etc.
- Documents
  - Unstructured text data (Web).
  - Semi-structured data (XML, RDF triples, etc.)
- Graphs
  - Social networks, Semantic Web (RDF graphs), road networks, etc.

- Data streams
  - Sensor data, RFID data, network data, trajectory data, etc.
- Time series data
  - Stock exchange data, video/audio data, trajectory, EEG data, etc.
- Multimedia data
  - Audio, video, image, etc.

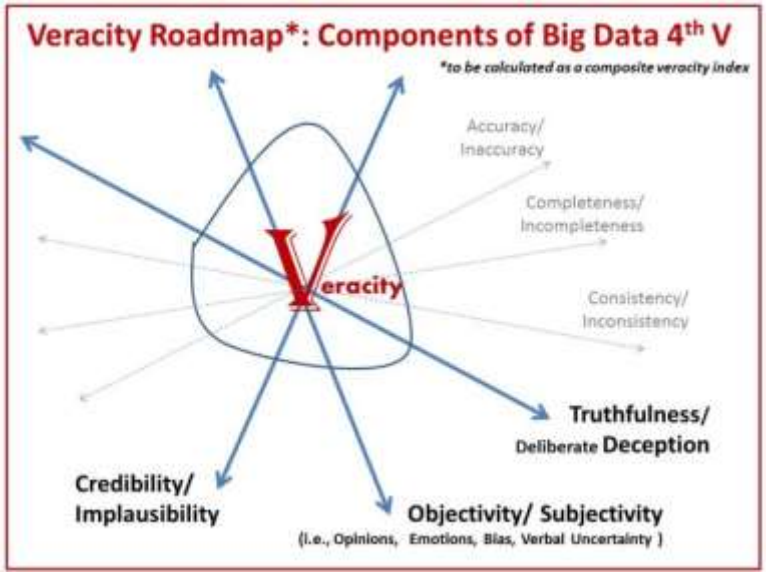


They could also be 4 V's

Veracity

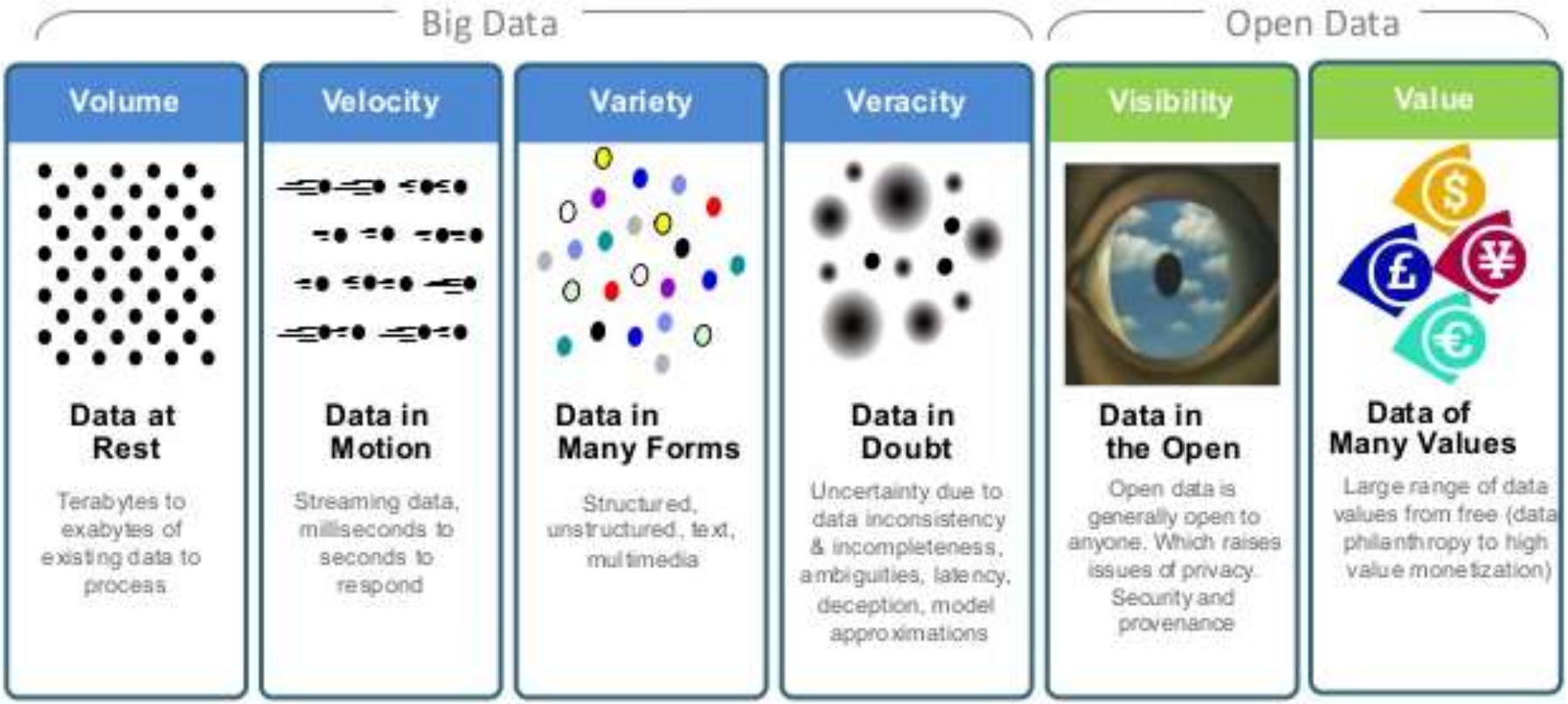
How accurate or truthful a data set may be

- Veracity is the degree to which data is accurate, precise, and trustworthy because of the biasedness, noise, abnormality in data.
- It also refers to incomplete data or the presence of errors, outliers, and missing values.
- To convert this type of data into a consistent, consolidated, and united source of information creates a big challenge for the enterprise



https://www.researchgate.net/figure/Conceptualization-of-the-Components-of-Big-Data-Veracity_fig3_260178341

Veracity: The Most Important "V" of Big Data
Aug 29, 2019

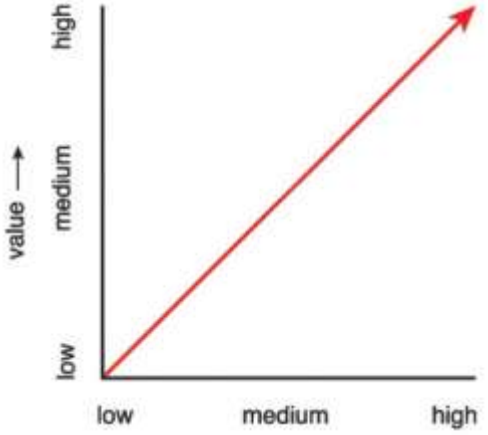


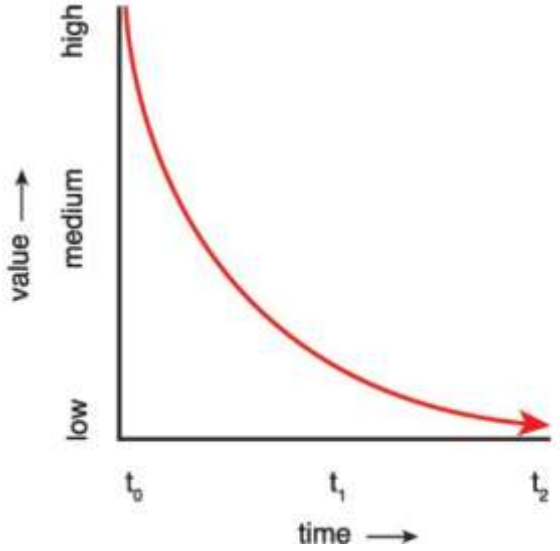
OR 5 V's

Value

The usefulness of data

- The value characteristic is intuitively related to the veracity characteristic in that the higher the data fidelity, the more value it holds for the business.
- Value is also dependent on how long data processing takes because analytics results have a shelf-life;
  - i.e., a 20 minute delayed stock quote has little to no value for making a trade compared to a quote that is 20 milliseconds old.
- Value and time are inversely related. The longer it takes for data to be turned into meaningful information, the less value it has for a business.





Value

The usefulness of data

- Apart from veracity and time, value is also impacted by the following lifecycle-related concerns:
  - How well has the data been stored?
  - Were valuable attributes of the data removed during data cleansing?
  - Are the right types of questions being asked during data analysis?
  - Are the results of the analysis being accurately communicated to the appropriate decision-makers?

3 Vs	Volume	Vast amount of data that has to be captured, stored, processed and displayed
	Velocity	Rate at which the data is being generated, or analyzed
	Variety	Differences in data structure (format) or differences in data sources themselves (text, images, voice, geospacial data)
5 Vs	Veracity	Truthfulness (uncertainty) of data, authenticity, provenance, accountability
	Validity	Suitability of the selected dataset for a given application, accuracy and correctness of the data for its intended use
7 Vs	Volatility	Temporal validity and fluency of the data, data currency and availability, and ensures rapid retrieval of information as required
	Value	Usefulness and relevance of the extracted data in making decisions and capacity in turning information into action
10 Vs	Visualization	Data representation and understandability of methods (data clustering or using tree maps, sunbursts, parallel coordinates, circular network diagrams, or cone trees)
	Vulnerability	Security and privacy concerns associated with data processing
	Variability	the changing meaning of data, inconsistencies in the data, biases, ambiguities, and noise in data

And 10 V's

A General View

- Databases are designed to do two things: *store data* and *retrieve data*.
- To meet these objectives, the database management systems must do three things:
  - Store Data Persistently.
  - Maintain Data Consistency.
  - Ensure Data Availability.

You can have consistent data with high availability, but transactions will require longer times to execute than if you did not have those requirements.

A General View

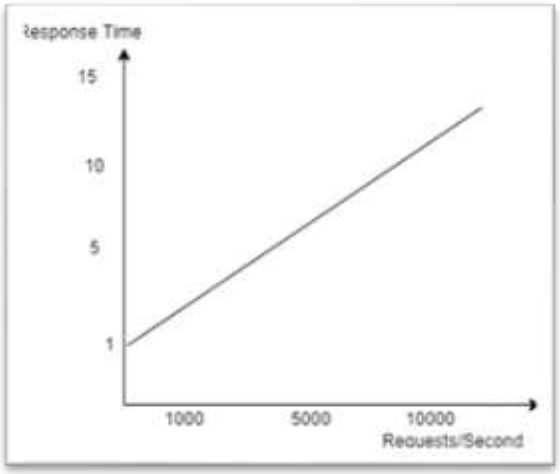
- Big data supports storing and querying *huge amounts of variant data* with *multiple copies*.
  - E.g., Facebook and google stores and process Exabyte and Zettabyte of data.
- So, we need *innovative storage strategies* and technologies that are Scalable, Flexible, Available, and Cost-effective.
- As data volumes increase, it becomes more difficult and expensive to *scale up*. A more appealing option is to *scale out*.
- However, *scaling out (over a cluster) introduces complexity*—so it's not something to do unless the benefits are compelling.

Scalability

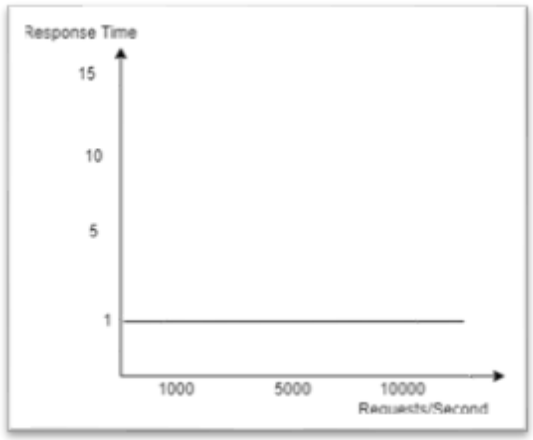
Handling the workload as per the magnitude of the work

- Scalability is the measure of a system's ability to increase or decrease in *performance* and *cost* in response to changes in system processing demands.
- It is a property of a system to handle a growing amount of work by *adding resources to the system*
  - How well a hardware system performs when the number of users is increased
  - How well a database withstands growing numbers of queries.

An overview of scalability principles.



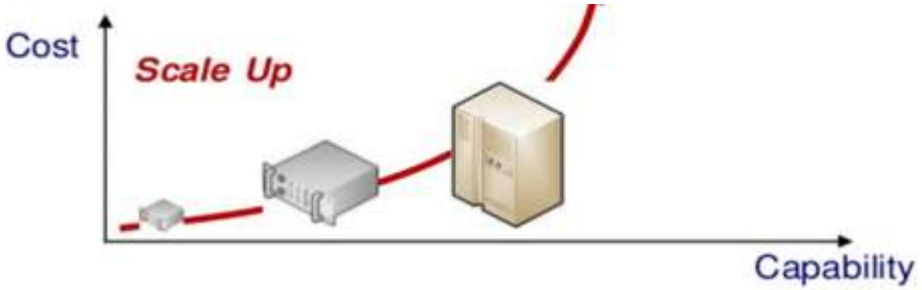
System that do not scale



A scalable System

Scale Up vs Scale Out

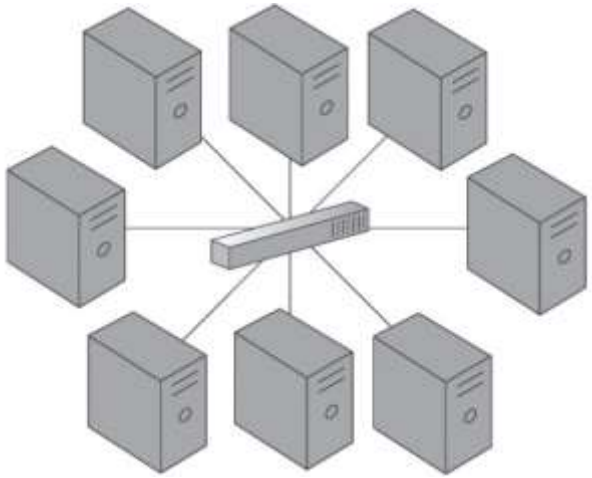
- **Scaling up**
  - Making a component bigger or faster so that it can handle more load.
  - ➢ Upgrade storage or processors.



- **Scaling out**
  - Adding more components in parallel to spread out a load.
  - ➢ Independent CPU, independent memory, etc.

Clusters

- A cluster is a tightly coupled *collection of servers*, or nodes *connected* via a network to work as a *single unit*.
- Each node in the cluster has its own *dedicated resources*, such as memory, a processor, and a hard drive.
- A cluster can execute a task by splitting it into small pieces and distributing their execution onto different computers that belong to the cluster



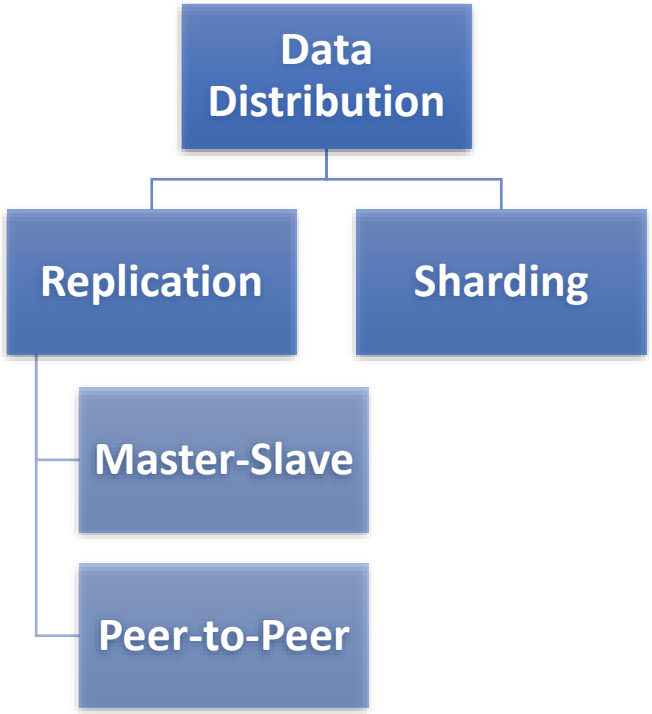
Erl T, Khattak W, Buhler P. *Big data fundamentals: concepts, drivers & techniques*. Prentice Hall Press; 2016 Jan

Distributed File Systems

- A distributed file system is a file system that can store large files spread across the nodes of a cluster.
- To the user, files appear to be local; however, physically the files are distributed throughout the cluster.
- This local view is presented via the distributed file system, and it enables the files to be accessed from multiple locations.
  - Examples include the Google File System (GFS) and Hadoop Distributed File System (HDFS).

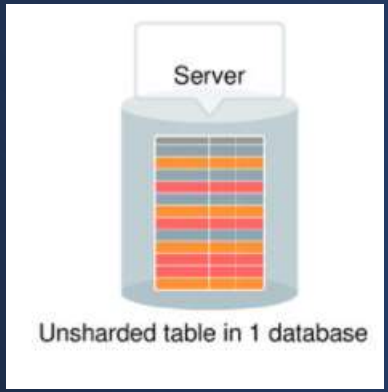
Data Distribution

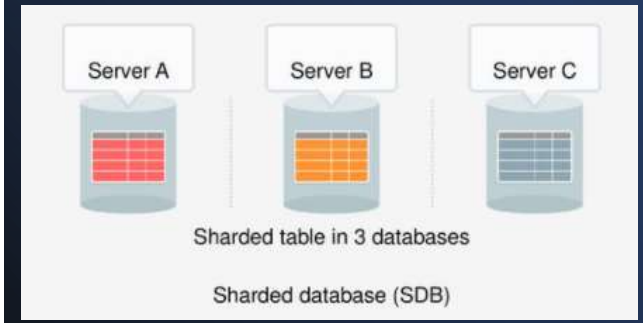
- **Data Distribution Models**
  - Single server (is an option for some applications).
  - Multiple servers.
- **Orthogonal aspects of data distribution**
  - Sharding: *different data* on *different nodes*.
  - Replication: the *same data* copied over *multiple nodes*.



Sharding

- Sharding is the process of *horizontally partitioning* a large dataset into a collection of smaller, *more manageable* datasets called *shards*.
- The shards are *distributed across multiple nodes*, where a node is a server or a machine.
- Each shard *shares the same schema*, and all shards collectively represent the complete dataset.
- It allows the *distribution of processing* loads across multiple nodes to achieve horizontal scalability.





Sharding

- Since each node is responsible for only a part of the whole dataset, *read/write times* are greatly *improved*.
- It provides *partial tolerance* toward failures, in case of a node failure, only data stored on that node is affected.
- With regards to data partitioning, *query patterns* need to be considered so that shards themselves do not become *performance bottlenecks*.
  - For example, queries requiring data from multiple shards will impose performance penalties.
- *Data Locality* keeps commonly accessed data co-located on a single shard and helps counter such performance issues.

Sharding - Improving Performance

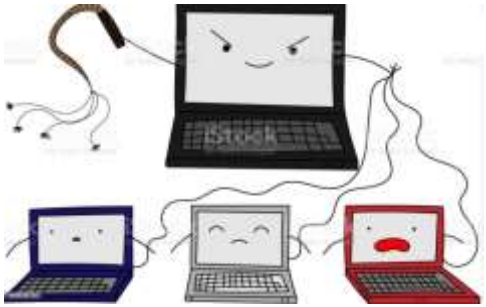
- **Main rules of sharding:**

1. Place the data close to where it's accessed – *Data Locality*
   - Orders form Aswan: data in the Upper Egypt data centre.
2. Try to keep the *load even*
   - All nodes should get equal amounts of the load.
3. Put together data that may be *read in sequence*
   - Same orders, same node.

Replication

- Replication stores *multiple copies* of a dataset, known as replicas, *on multiple nodes*.
- It provides *scalability* and *availability* since the same data is replicated on various nodes.
- *Fault tolerance* is also achieved since data redundancy ensures that data is not lost when an individual node fails.
- There are two different methods that are used to implement replication:
  - Master-Slave
  - Peer-to-peer

Master-Slave



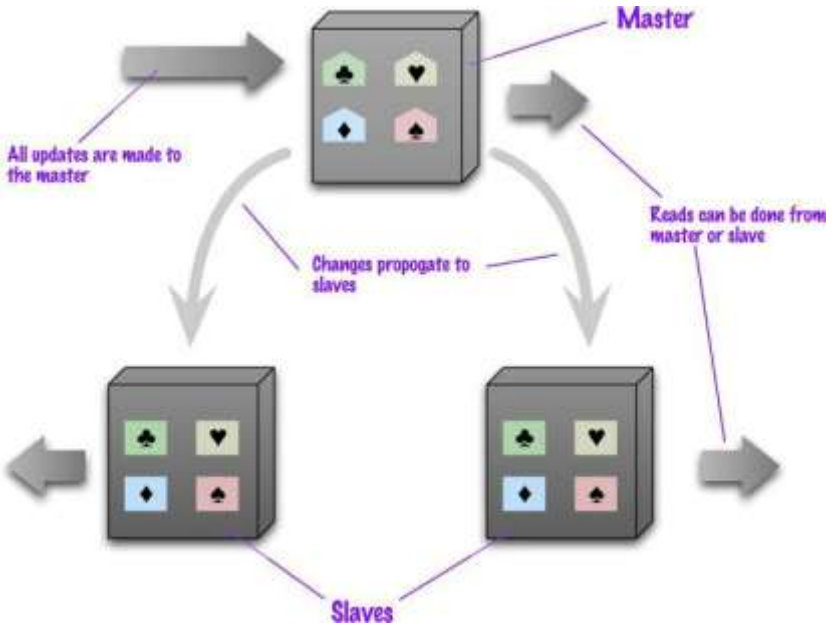
https://www.istockphoto.com/vector/master-slave-topology-gm899039750-248082021

Peer-to-Peer

https://www.dreamstime.com/stock-image-peer-to-peer-network-image26762941

Master-Slave

- In a master-slave configuration, *all data* is written to a *master node*. Once saved, the data is *replicated* over to multiple *slave nodes*.
- All external *write requests*, including insert, update and delete, occur on the *master node*.
- Whereas *read requests* can be fulfilled by any *slave node*.
- Ideal for *read intensive loads* rather than write intensive loads
  - Write performance will suffer as the amount of writes increases. WHY ???
  - If the master node fails, reads are still possible via any of the slave nodes.

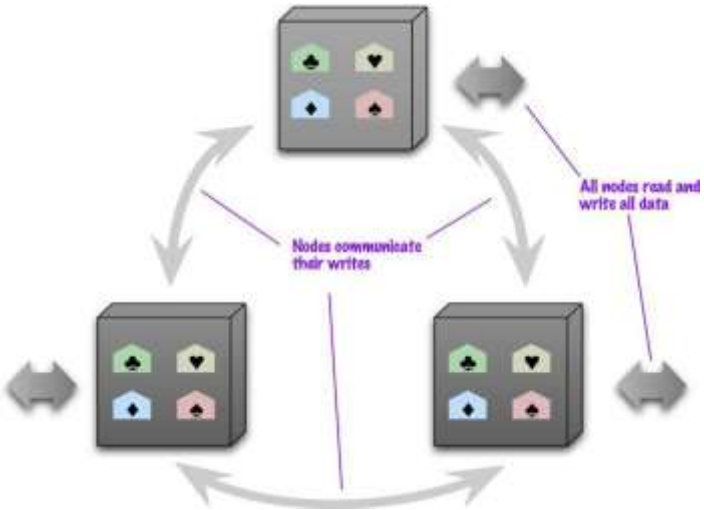


Master-Slave

- A *slave node* can be configured as a *backup* node for the master node.
- If the *master node fails*, writes are not supported until a master node is re-established.
  - The master node is either restore to life, or a new master node is chosen from the slave nodes.
- One concern with master-slave replication is *read inconsistency*, which can be an issue if a slave node is read prior to an update to the master being copied to it.
  - A voting system can be implemented where a read is declared consistent if the majority of the slaves contain the same version of the record.
  - Implementation of such a voting system requires a reliable and fast communication mechanism between the slaves.

Peer-to-Peer Replication

- All the replicas have equal weight, there is no a master-slave relationship between the nodes.
- The loss of any of them doesn't prevent access to the data store.
- Each node, known as a peer, is equally capable of handling reads and writes.
- Each write is copied to all peers - prone to *write inconsistencies*.

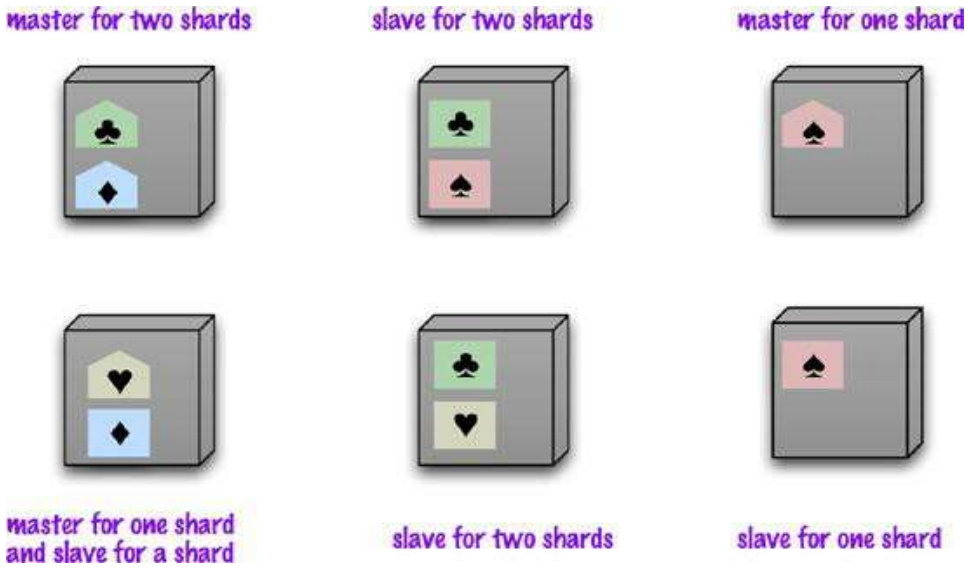


Peer-to-Peer Replication

- Peer-to-peer replication is prone to *write inconsistencies* that occur as a result of a simultaneous update of the same data across multiple peers.
- This can be addressed by implementing either one of the following concurrency strategy:
- Pessimistic concurrency - proactive strategy that prevents inconsistency.
  - It uses *locking* to ensure that only one update to a record can occur at a time *on the cost of availability* since the database record being updated remains unavailable until all locks are released.
- Optimistic concurrency - reactive strategy that does not use locking.
  - Instead, it *allows inconsistency to occur* with knowledge that eventually consistency will be achieved after all updates have propagated.
- To ensure read consistency, a *voting system* can be implemented

Combining Sharding and Master-Slave Replication

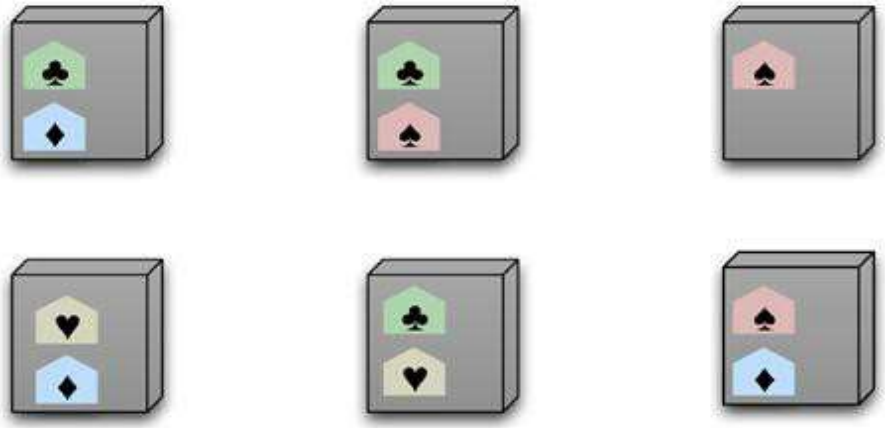
- We have multiple masters, but *each data item has a single master*.
- Two schemes:
  - A node can be a master for some data and slaves for others.
  - Nodes are dedicated for master or slave duties.



- *Write consistency* is maintained *by the master-shard*. However, if the master-shard becomes non-operational, fault tolerance with regards to write operations is impacted.
- *Replicas of shards* are kept on multiple slave nodes to provide *scalability* and *fault tolerance* for read operations.

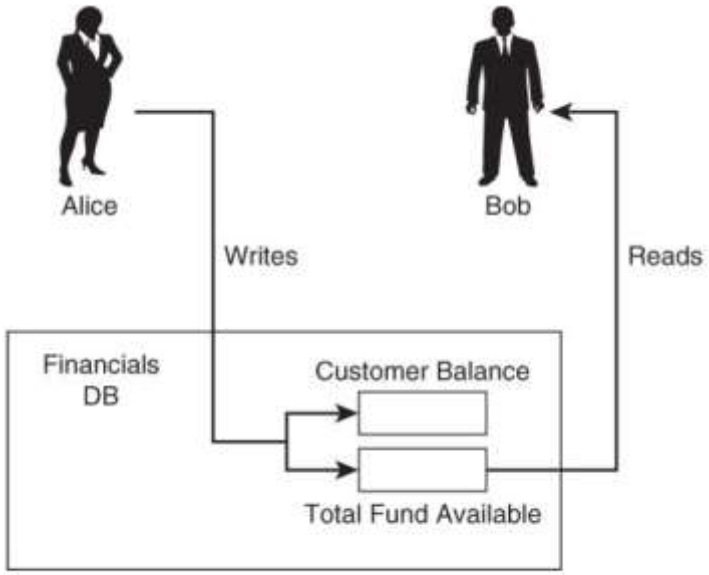
Combining Sharding and Peer-to-Peer Replication

- Each shard is replicated to multiple peers.
- Each peer is only responsible for a subset of the overall dataset.
- Collectively, this helps achieve increased scalability and fault tolerance.
  - As there is no master involved, there is *no single point of failure* and fault-tolerance for both read and write operations is supported.



Maintain Data Consistency

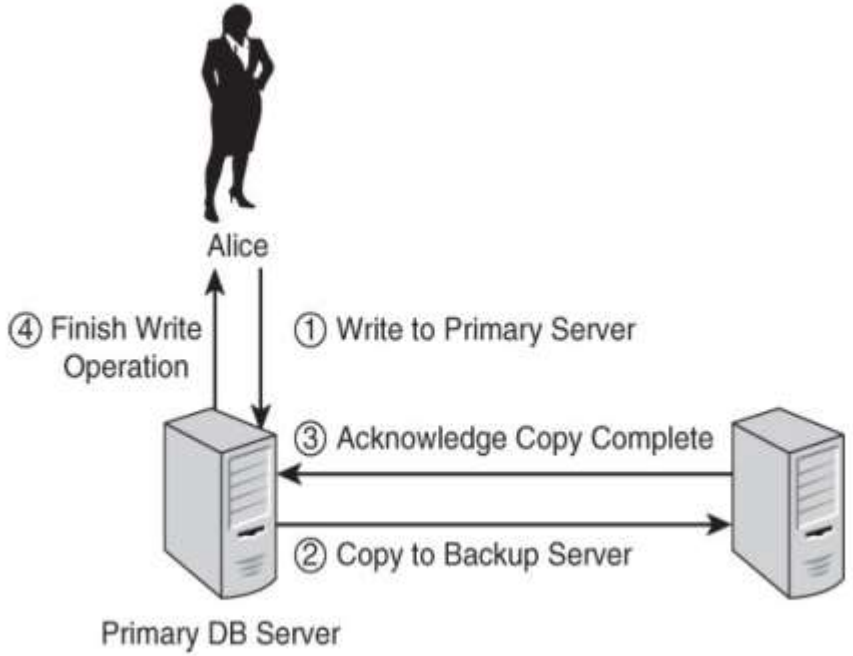
- Consistency means that only valid data, according to all defined rules, will be written to the persistent storage.
- In the context of distributed systems, consistency also refers to maintaining a single and logically coherent view of data in.
- Relational database systems are designed to support consistency by the concept of atomic transaction.



Data should reflect a consistent state.

Ensure Data Availability

- Availability is the degree to which data can be instantly accessed even if a failure occurs.
- One way to avoid unavailability is to have two database servers:
  - Primary Server
  - Backup Server.
- To keep them consistent, a *Two-Phase Commit Protocol* can be used.



Two-phase commits are complete only when operations on both databases are complete.

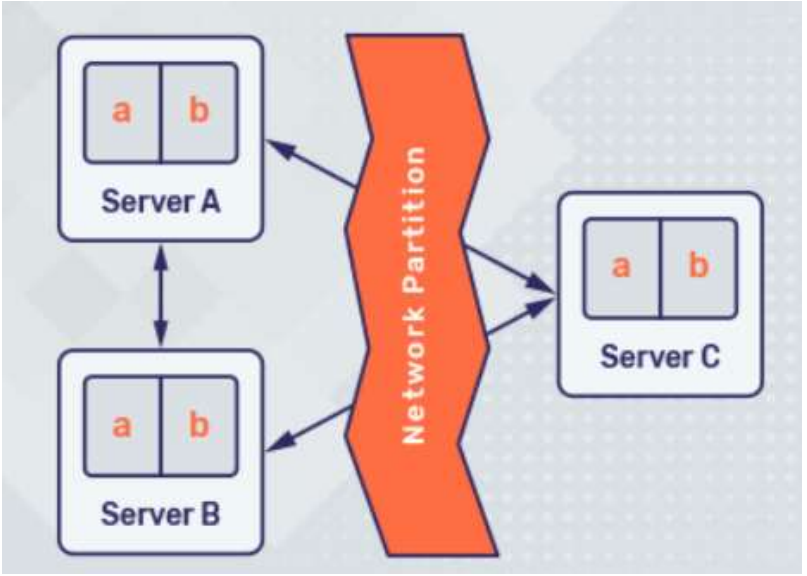
Consistency and Availability

- There is a fundamental *trade-off* between Consistency and availability
- There are two broad options:
  - Ensuring consistency among all replicas on the cost availability.
  - Accepting and coping with inconsistent writes to ensure availability.
- These points are at the ends of a spectrum where we trade off consistency for availability.
- Different domains have *different tolerances for inconsistency*, and we need to take this tolerance into account as we make our decisions.

Network Partition

- Network partitioning is a network failure that causes the nodes to split into multiple groups such that a node in a group cannot communicate with nodes in other groups.

- In a partition scenario, all sides of the original cluster operate independently assuming nodes in other sides are failed.

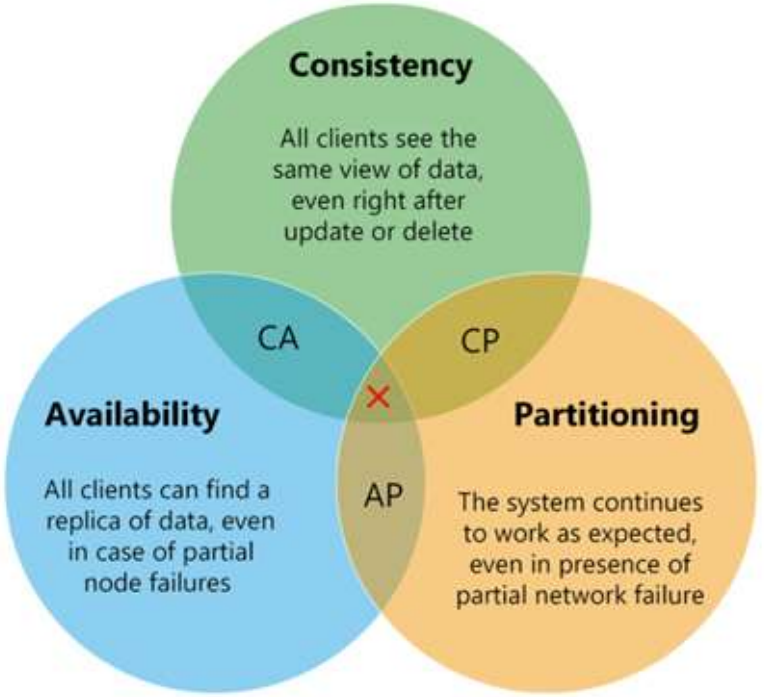


https://www.yugabyte.com/blog/achieving-fast-failovers-after-network-partitions-in-a-distributed-sql-database/

The CAP Theorem

Brewer's Theorem

- A triple constraint related to distributed database systems
  - **Consistency** - read from any node results in the same data across multiple nodes
  - **Availability** - a guarantee that every request receives a response about whether it was successful or failed)
  - **Partition tolerance** - the system continues to operate despite arbitrary message loss or failure of part of the system



"Given the properties of Consistency, Availability, and Partition tolerance, you can only get two"

The CAP Theorem

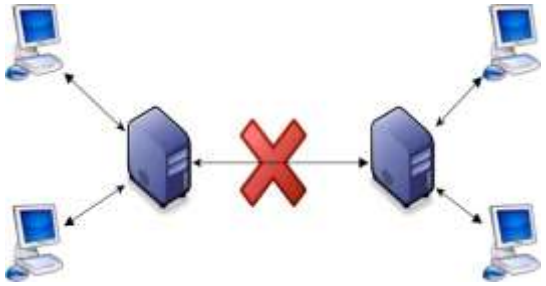
- It is impossible for a distributed system to simultaneously provide all three properties together.
  - If consistency (C) and availability (A) are required, available nodes need to communicate to ensure consistency (C). Therefore, partition tolerance (P) is not possible.
  - If consistency (C) and partition tolerance (P) are required, nodes cannot remain available (A) as the nodes will become unavailable while achieving a state of consistency (C).
  - If availability (A) and partition tolerance (P) are required, then consistency (C) is not possible because of the data communication requirement between the nodes. So, the database can remain available (A) but with inconsistent results.

The CAP Theorem

- In a distributed database, scalability and fault tolerance can be improved through additional nodes, although this challenges consistency (C).
- The addition of nodes can also cause availability (A) to suffer due to the latency caused by increased communication between nodes.
- In distributed database systems, partition tolerance (P) must always be supported; therefore, CAP is generally a choice between choosing either ***C+P*** or ***A+P***.
- The requirements of the system will dictate which is chosen.

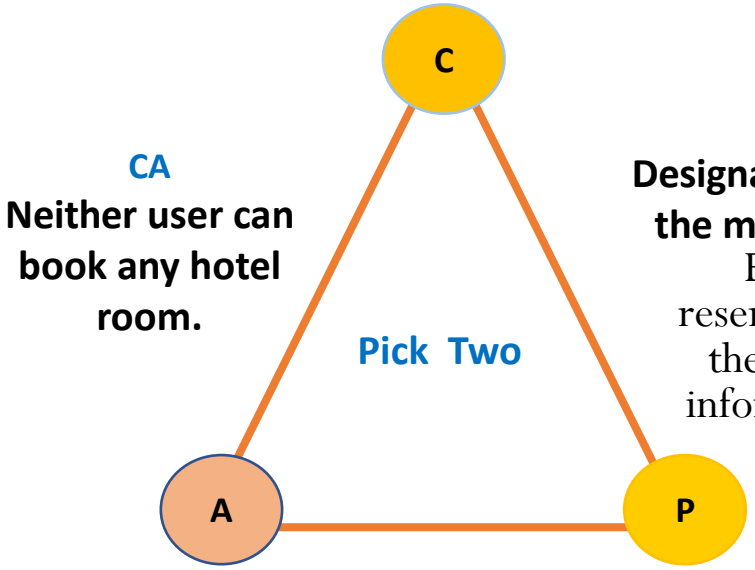
An Example

- Alice is trying to book a room of the ABC Hotel on a node located in London of a booking system.
- Bob is trying to do the same on a node located in Mumbai
- The booking system uses a peer-to-peer distribution
- There is only one room available
- The network link breaks



Possible Solutions

These situations are closely tied to the domain. Financial exchanges, Blogs, Shopping charts.



CA

Neither user can book any hotel room.

CP

Designate Mumbai node as the master for ABC hotel

Bob can make the reservation, Alice can see the inconsistent room information, and cannot book the room.

AP

Both nodes accept the hotel reservation

Overbooking

[CAP Theorem and Distributed Database Management Systems]
Syed Sadat Nazrul | Apr 24, 2018

ACID -- Requirement for SQL DBs

- ACID is a database design principle related to transaction management. It is an acronym that stands for:
  - **Atomicity**. All of the operations in the transaction will complete, or none will.
  - **Consistency**. Transactions never observe or result in inconsistent data.
  - **Isolation**. The transaction will behave as if it is the only operation being performed upon the database (i.e. uncommitted transactions are isolated)
  - **Durability**. Upon completion of the transaction, the operation will not be reversed (i.e. committed transactions are permanent)

Forms of Consistency

Strong (or immediate) consistency	• ACID transaction
Logical consistency	• No read-write conflicts (atomic transactions)
Sequential consistency	• Updates are serialized
Session (or read-your-writes) consistency	• Within a user's session
Eventual consistency	• You may have replication inconsistencies but eventually all nodes will be updated to the same value

The BASE Properties

It favours availability over consistency by relaxing the strong consistency constraints mandated by the ACID properties

- BASE is a database design principle based on the CAP theorem and leveraged by database systems that use distributed technology.
  - ***BA**sically Available* means that there can be a partial failure in some parts of the distributed system and the rest of the system continues to function.
  - ***S**oft State* means data will expire if it is not refreshed - data may eventually be overwritten with more recent data
  - ***E**ventually Consistent* means that there may be times when the database is in an inconsistent state.
    - Casual consistency
    - Read-your-writes consistency
    - Session consistency
    - Monotonic read consistency
    - Monotonic write consistency

RDBs and Scalability

- The ACID properties are the cornerstone of SQL databases, ensuring reliable processing of transactions.
- The ACID properties seem *indispensable*, and yet they are *incompatible* with availability and performance *in very large systems*.
- Maintaining these properties across a distributed system is a challenge
  - Relational databases are designed to scale up on expensive single machines.

Why Is It Hard to Horizontally Scale SQL Databases?
Arslan Ahmad | November 6th, 2023

NoSQL Datastores

- The emergence of NoSQL datastores can primarily be attributed to the volume, velocity and variety characteristics of Big Data datasets.
- NoSQL databases (aka "not only SQL") are *non-tabular databases* and store data differently than relational tables.
- They have better *horizontal scaling* capability, *fault-tolerant* , and *improved performance* for big data at the cost of having *less rigorous consistency models*.
- These systems are optimized for fast retrieval and appending operations on records where *real-time performance* is *more important than consistency*.
- NoSQL databases come in a *variety of types* based on their data model.

NoSQL vs SQL- 4 Reasons Why NoSQL is better for Big Data applications
Feb 2023

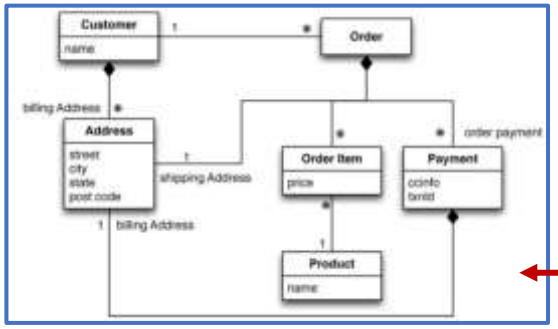
NoSQL Datastores Characteristics

- This list should only be considered a general guide, as not all NoSQL storage devices exhibit all of these features:
  - *Schema-less* data model – Data can exist in its raw form.
  - *Scale out* rather than scale up – More nodes can be added to efficiently meet the needs for varying workloads.
  - *Highly available* – This is built on cluster-based technologies that provide fault tolerance out of the box.
  - *Lower operational costs* – Many NoSQL databases are built on Open-Source platforms with no licensing costs. They can often be deployed on commodity hardware.
  - *BASE not ACID* – Maintain high availability in the event of network/node failure, while not requiring the database to be in a consistent state whenever an update occurs. The database can be in a soft/inconsistent state until it eventually attains consistency.

NoSQL Datastores Characteristics

- *Auto sharding and replication* – To support horizontal scaling and provide high availability, a NoSQL storage device automatically employs sharding and replication techniques where the dataset is partitioned horizontally and then copied to multiple nodes.
- *Distributed query support* – NoSQL storage devices maintain consistent query behaviour across multiple shards.
- *Polyglot persistence* – An approach of persisting data using different types of storage technologies within the same solution architecture.
- *Aggregate-oriented* – NoSQL storage devices store de-normalized aggregated data thereby eliminating the need for joins
  - One exception, however, is that graph database storage devices are not aggregate-focused.

Aggregate Data Models



The relational model divides the information into tables of tuples. This simple structure for data is one of the key aspects of its success and dominance

Aggregate oriented models take a different approach.
They tend to operate on data in units that have a more complex structure.

```
// in customers
{
"id":1,
"name":"Martin",
"billingAddress":[{"city":"Chicago"}]
}

// in orders
{
"id":99,
"customerId":1,
"orderItems":[
  {
  "productId":27,
  "price": 32.45,
  "productName": "NoSQL Distilled"
    }
  ],
"shippingAddress": {"city":"Chicago"}
"orderPayment":[
  {
    "ccinfo":"1000-1000-1000-1000",
    "txnId":"abelif879rft",
    "billingAddress": {"city": "Chicago"}
  }
  ],
}
```

- An aggregate is a collection of data that we manipulate and manage as a unit.
  - complex record with simple fields, arrays, records nested inside
- *Aggregate-oriented databases* work best when most data interaction is done with the same aggregate (intra)
- *Aggregate-ignorant databases* are better when interactions use data organized in many different formations (inter)

Schemeless Databases

Freedom and flexibility

double-edged sword

Schemaless allows for *more flexibility* than schema-based databased.

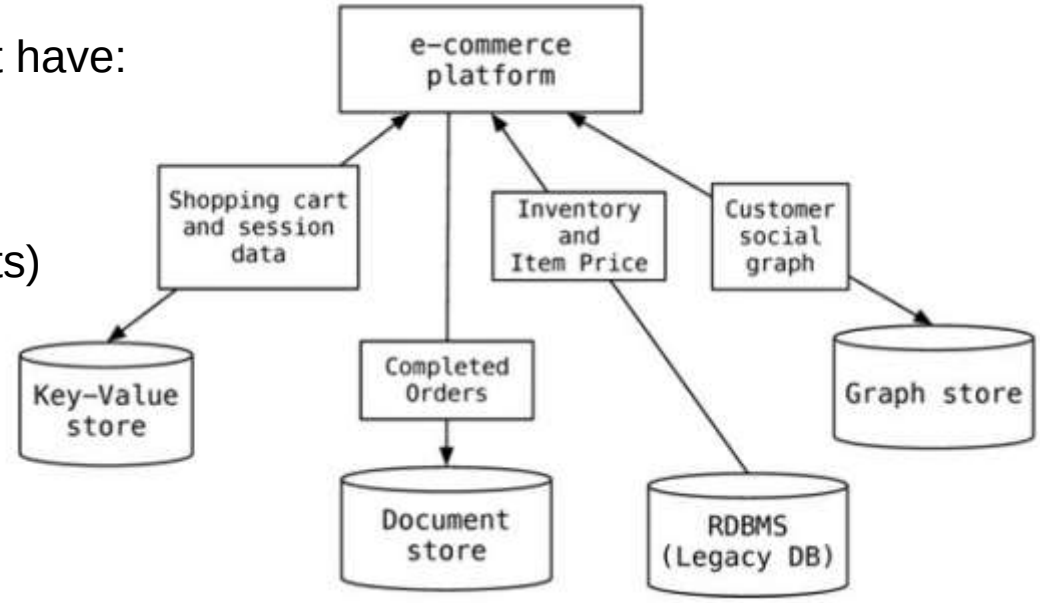
- However, there is less opportunity to automatically enforce data integrity rules.
- There should be an *implicit schema* expected by users of the data.

  A set of assumptions about the structure of the data in the code that manipulates it.

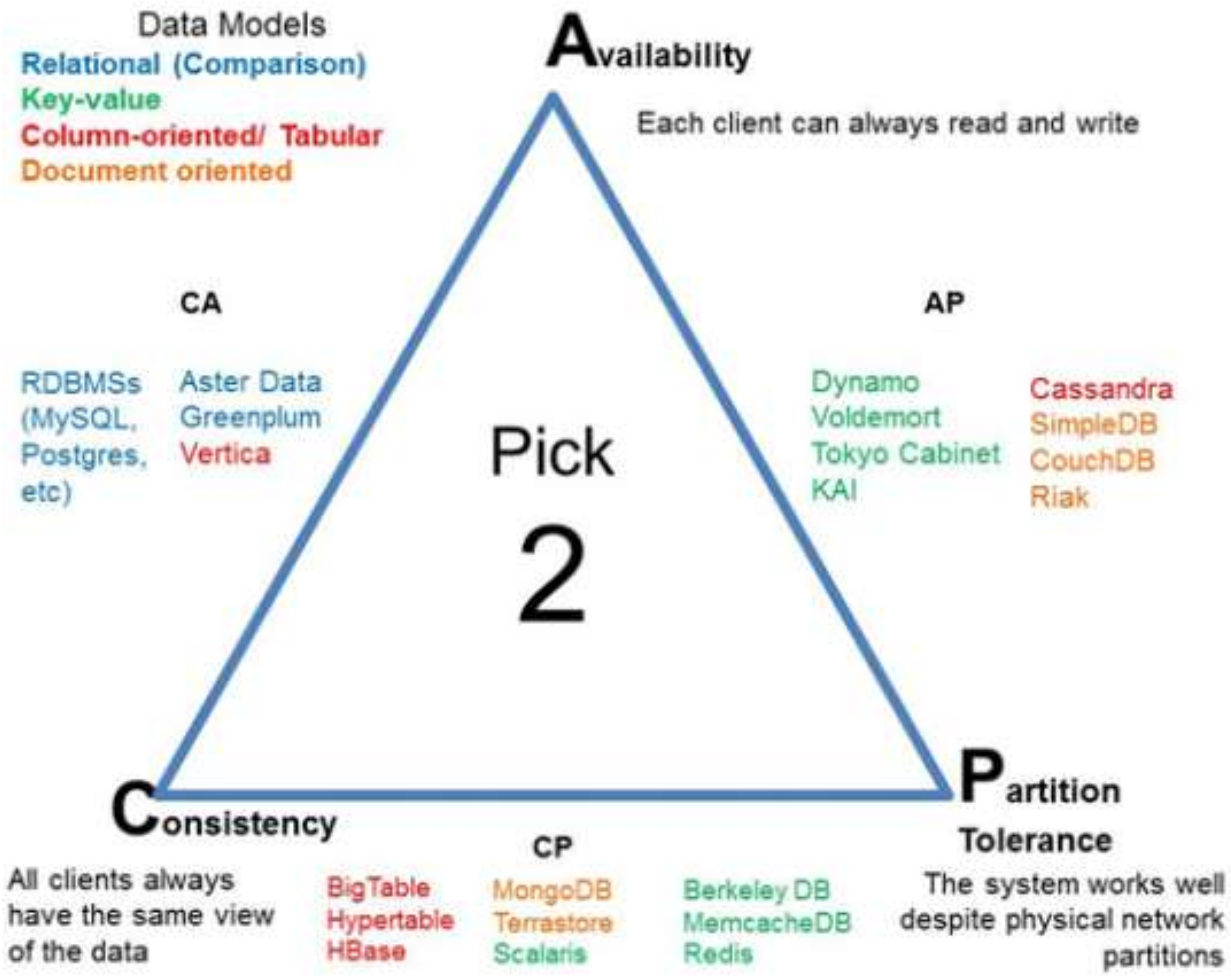
- Schemaless database shifts the schema into the application code that accesses it.
- This becomes problematic if multiple applications, developed by different people, access the same database.
- *In order to understand the structure of the data, you have to understand the application code.*
- Having a schemaless affects the efficiency of storing and retrieving the data.

Polyglot Persistence

- Using *multiple specialized persistent stores* rather than one single general-purpose database.
  - *"Monoglot"* was (and still is) fine for simple application (one type of workload)
- But… applications become complex.
- A simple E-commerce platform must have:
  - Session data (Add to Basket)
  - Search Engine (Search for products)
  - Recommendation engine.
  - Payment platform.
  - Geo Location service

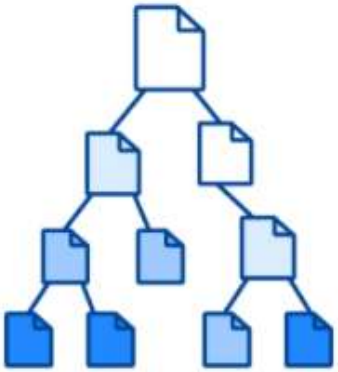


NoSQL Systems and CAP

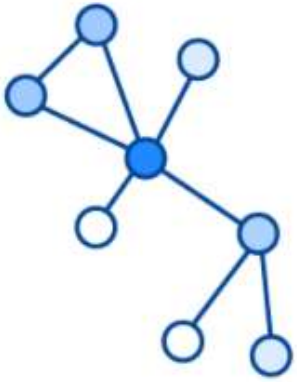


Four Types of NoSQL Databases

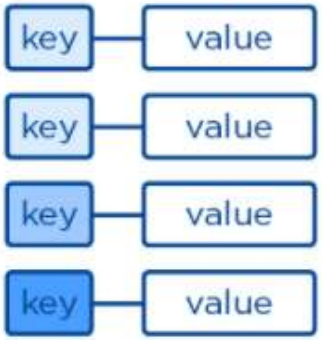
Document



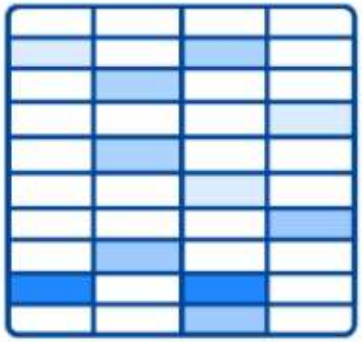
Graph



Key-Value



Wide-column



Where NoSQL Is Used?

- Google (BigTable, LevelDB)
- LinkedIn (Voldemort)
- Facebook (Cassandra)
- Twitter (Hadoop/Hbase, FlockDB, Cassandra)
- Netflix (SimpleDB, Hadoop/HBase, Cassandra)
- CERN (CouchDB)

Who Uses NoSQL

- Over the last few years, NoSQL database technology has experienced explosive growth and accelerating use by large enterprises. For example:
  - Tesco uses NoSQL to support its catalogue, pricing, inventory, and coupon applications.
  - McGraw-Hill uses NoSQL to power its online learning platform
  - Sky uses NoSQL to manage user profiles for 20 million subscribers

The Top 10 Enterprise NoSQL Use Cases
2015

NewSQL Databases

- NewSQL storage devices combine the ACID properties with the scalability and fault tolerance offered by NoSQL storage devices.
- They generally *support SQL compliant syntax* for data definition and data manipulation operations, and they often use *a logical relational data model* for data storage.
- NewSQL databases can be used *for developing OLTP* systems with very high volumes of transactions as they leverage in-memory storage.
  - E.g. example a banking system. They can also be used for realtime analytics,
- Compared to a NoSQL storage device, a NewSQL storage device provides an easier transition from a traditional RDBMS to a highly scalable database.
- Examples of NewSQL databases include VoltDB, NuoDB and InnoDB.

VOLTDB

nuoDB®