NoSQL

Structured and Unstructured Data

Taxonomy of NoSQL Implementation

Discussion of basic architecture of Hbase, Cassandra and MongoDb

Structured Data

- Structured data is defined as the data that can fit into the fixed record or file.
- Such data can be stored in relational databases and spreadsheet.
- It is easily searchable by the basic algorithm or basic queries.
- It is written in the format that is easy for machine to understand.
- It is simple to enter, store, query and analyze.
- It must be strictly defined in terms of field name and type.

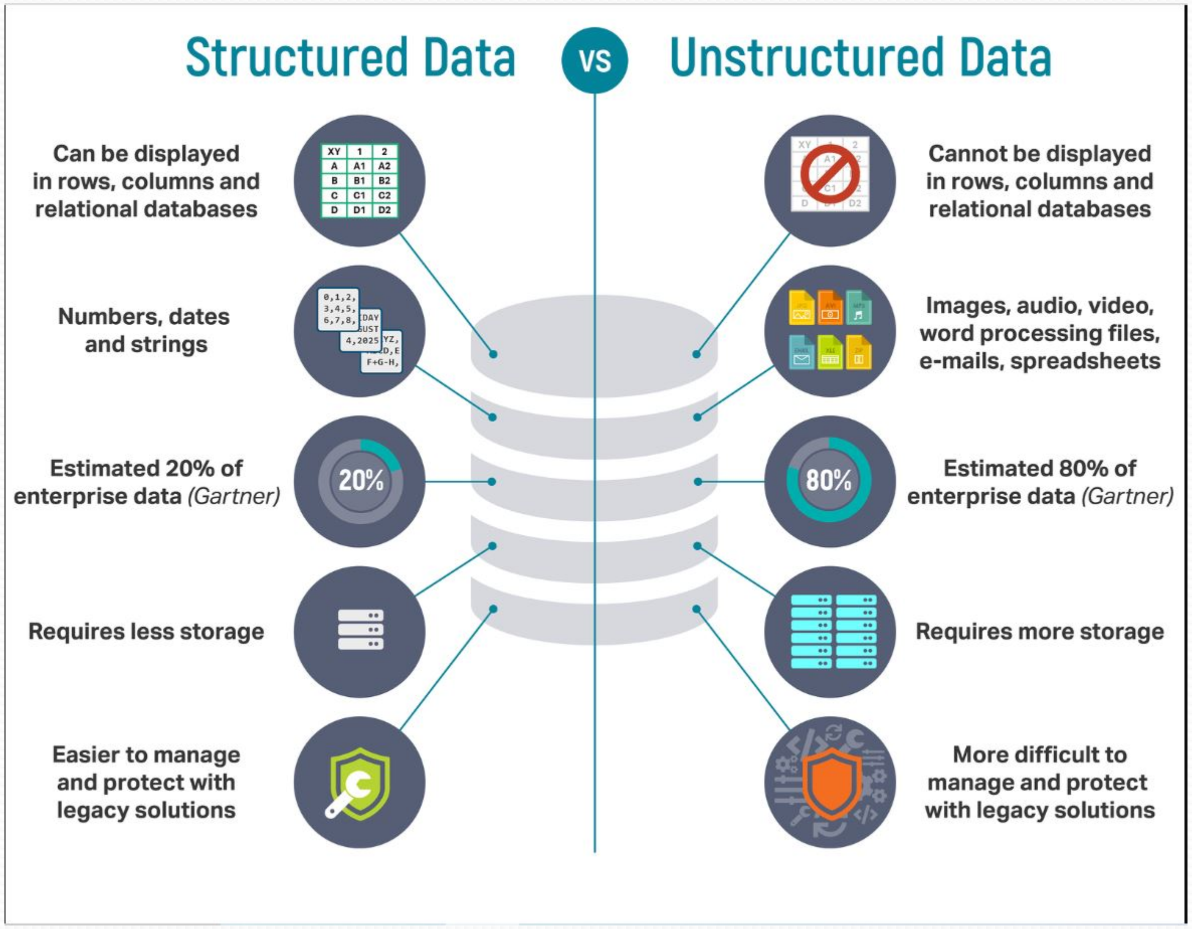
Structured Data

- The example of structured data is:

Employee	**Age**	**Salary**
Ram	25	39000
Shyam	23	42000
Hari	24	490000
Sita	21	450000

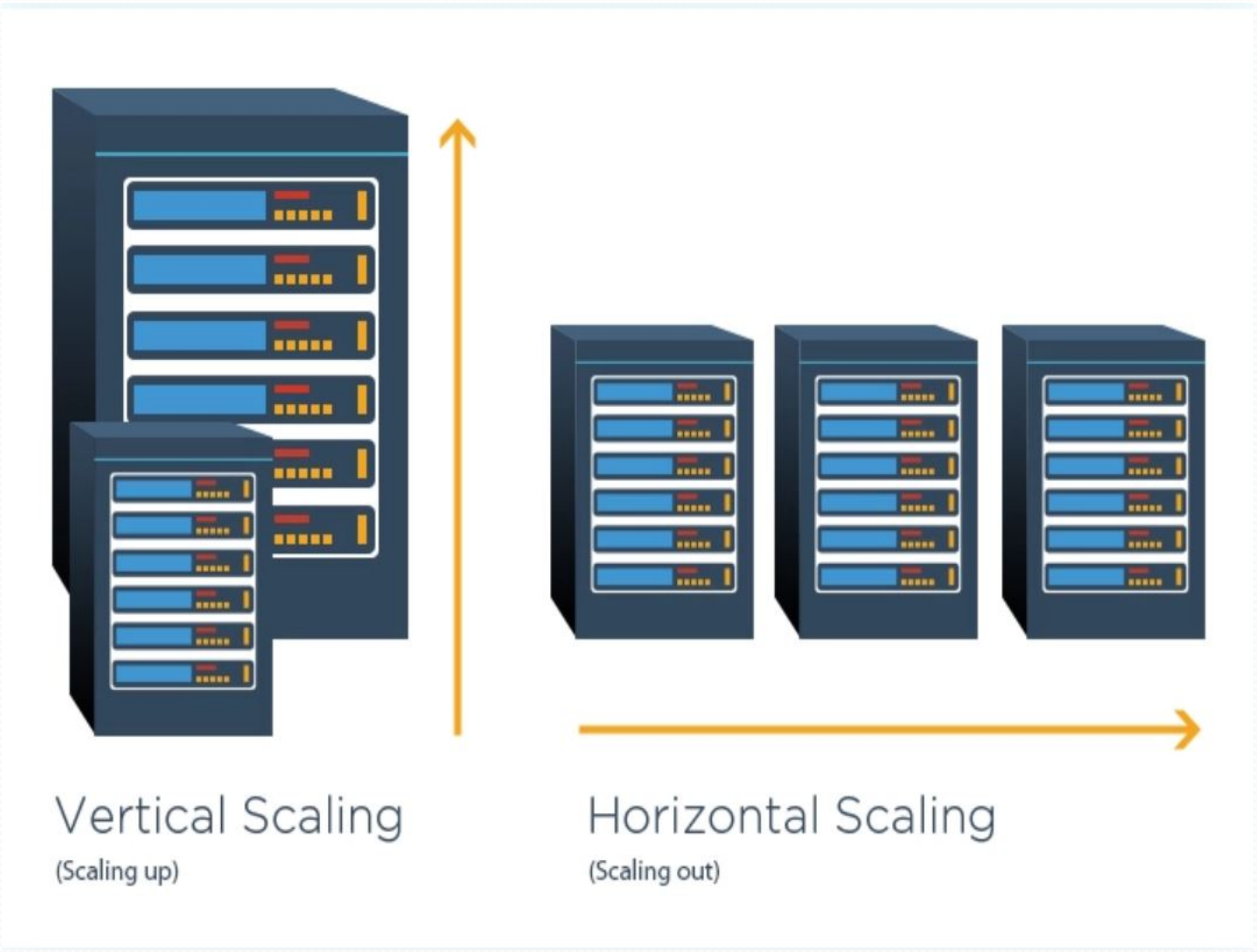
Unstructured Data

- Unstructured data is defined as the data that cannot be classified and cannot fit in a fixed record.
- Such data cannot be stored in relational databases as they do not possess any well-known structure.
- It generally allows keyword based queries or sophisticated conceptual queries.
- t is written in the format that is easy for humans to understand.
- It is being increasingly valuable and available.
- It is difficult to unbox, understand and analyze.
- It refers to the free text data.
- The example of structured data includes personal messaging, business documents, web content and so on.


Structured Data
VS
Unstructured Data
Can be displayed in rows, columns and relational databases
Cannot be displayed in rows, columns and relational databases
Numbers, dates and strings
Images, audio, video, word processing files, e-mails, spreadsheets
Estimated 20% of enterprise data (Gartner)
20%
80%
Estimated 80% of enterprise data (Gartner)
Requires less storage
Requires more storage
Easier to manage and protect with legacy solutions
More difficult to manage and protect with legacy solutions

Distributed system

- A **distributed system** is a network that stores data on more than one node (physical or virtual machines) at the same time.


Vertical Scaling
(Scaling up)
Horizontal Scaling
(Scaling out)

Scaling of Database

- **Vertical Scaling**
  - It is also known as scaling up.
  - It is achieved by upgrading the new hardware requirements to fulfill the processing demands.
  - It is generally configured in the single machine.
  - It may hamper availability of resources at the time of upgrade.

Scaling of Database

- Horizontal Scaling
  - It is also known as scaling out.
  - It is achieved by adding the necessary hardware parallel to the currently available hardware.
  - It is generally configured in multiple machines.
  - It do not hamper availability of resources.
  - It involves database sharing and database replication.
  - Database sharing refers to the stripping of data and storing in multiple machine to allow concurrent access to the database.
  - Database replication refers to the storing of copies of database in multiple machines to ensure availability and prevent from single point failure.

CAP theorem

- CAP stands for Consistency, Availability and Partition Tolerance.
- CAP theorem is used to describe the limitations of distributed databases.
- Consistency refers to the process of maintaining the same state of data in all the replicas at any instance.
- Availability refers to the process of successfully operate at any instance even if some node crashes.
- Partition tolerance refers to the process of operation of the system in presence of network partition.
- CAP theorem states that any distributed database with shared data can have it must two of the three desirable properties (Consistency, Availability and Partition Tolerance).

CAP theorem

- CAP theorem can be summarized as: For any distributed database, one of the following can hold:
  - If a database guarantees availability and partition tolerance, it must forfeit consistency. Egg: Cassandra, CouchDB and so on.
  - If a database guarantees consistency and partition tolerance, it must forfeit availability. Egg: HBase, Mongo DB and so on.
  - If a database guarantees availability and consistency, there is no possibility of network partition. Egg: RDBMS like MySQL, Postgres and so on.

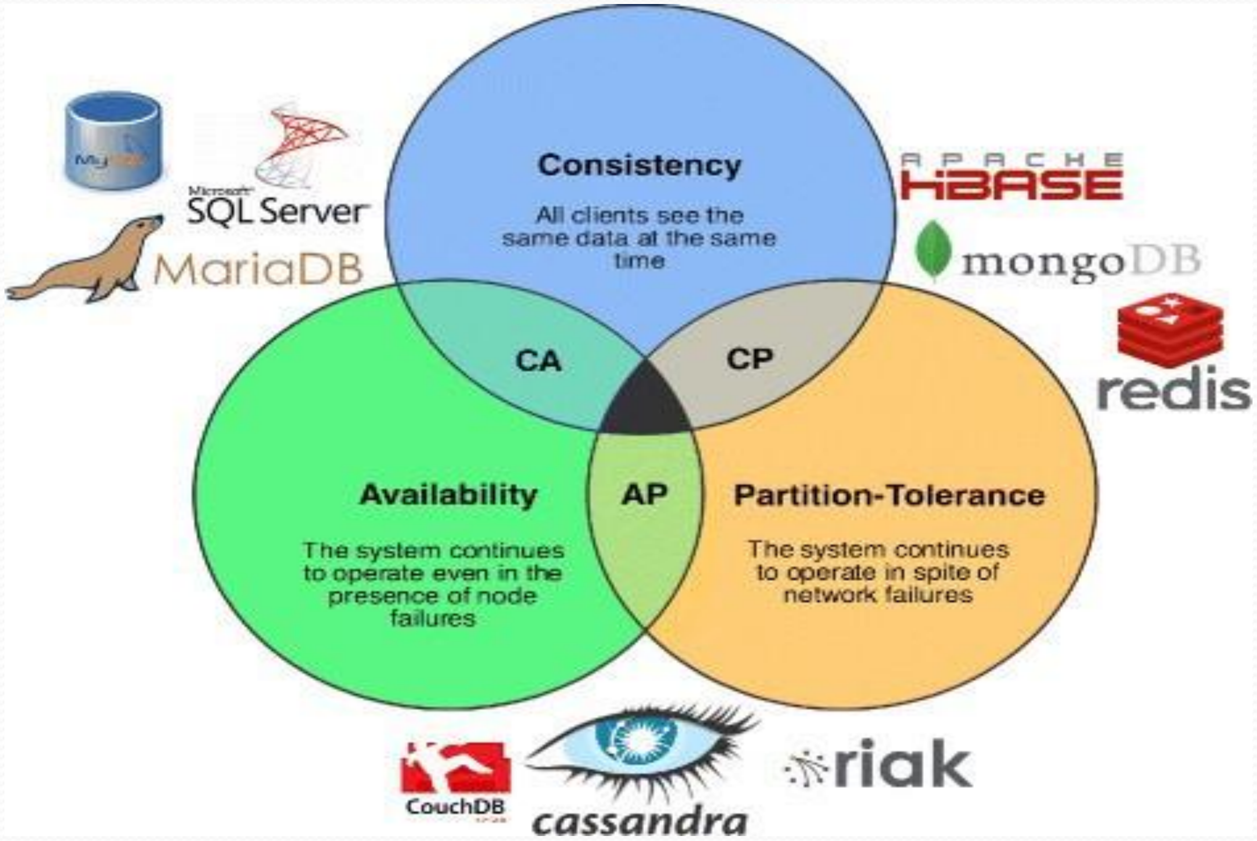
- **Consistency:** means that all clients see the same data at the same time, no matter which node they connect to in a distributed system. To achieve consistency, whenever data is written to one node, it must be instantly forwarded or replicated to all the other nodes in the system before the write is deemed successful.
- **Availability:** means that every non-failing node returns a response for all read and write requests in a reasonable amount of time, even if one or more nodes are down. Another way to state this — all working nodes in the distributed system return a valid response for any request, without failing or exception.

- **Partition Tolerance:** means that the system continues to operate despite arbitrary message loss or failure of part of the system. In other words, even if there is a network outage in the data center and some of the computers are unreachable, still the system continues to perform. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

- The CAP theorem categorizes systems into three categories:
  - **CP (Consistent and Partition Tolerant) database:** A CP database delivers consistency and partition tolerance at the expense of availability. When a partition occurs between any two nodes, the system has to shut down the non-consistent node (i.e., make it unavailable) until the partition is resolved.
  - **Partition** refers to a communication break between nodes within a distributed system. Meaning, if a node cannot receive any messages from another node in the system, there is a partition between the two nodes. Partition could have been because of network failure, server crash, or any other reason.

- **AP (Available and Partition Tolerant) database:** An AP database delivers availability and partition tolerance at the expense of consistency. When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others. When the partition is resolved, the AP databases typically resync the nodes to repair all inconsistencies in the system.
- **CA (Consistent and Available) database:** A CA delivers consistency and availability in the absence of any network partition. Often a single node's DB servers are categorized as CA systems. Single node DB servers do not need to deal with partition tolerance and are thus considered CA systems.

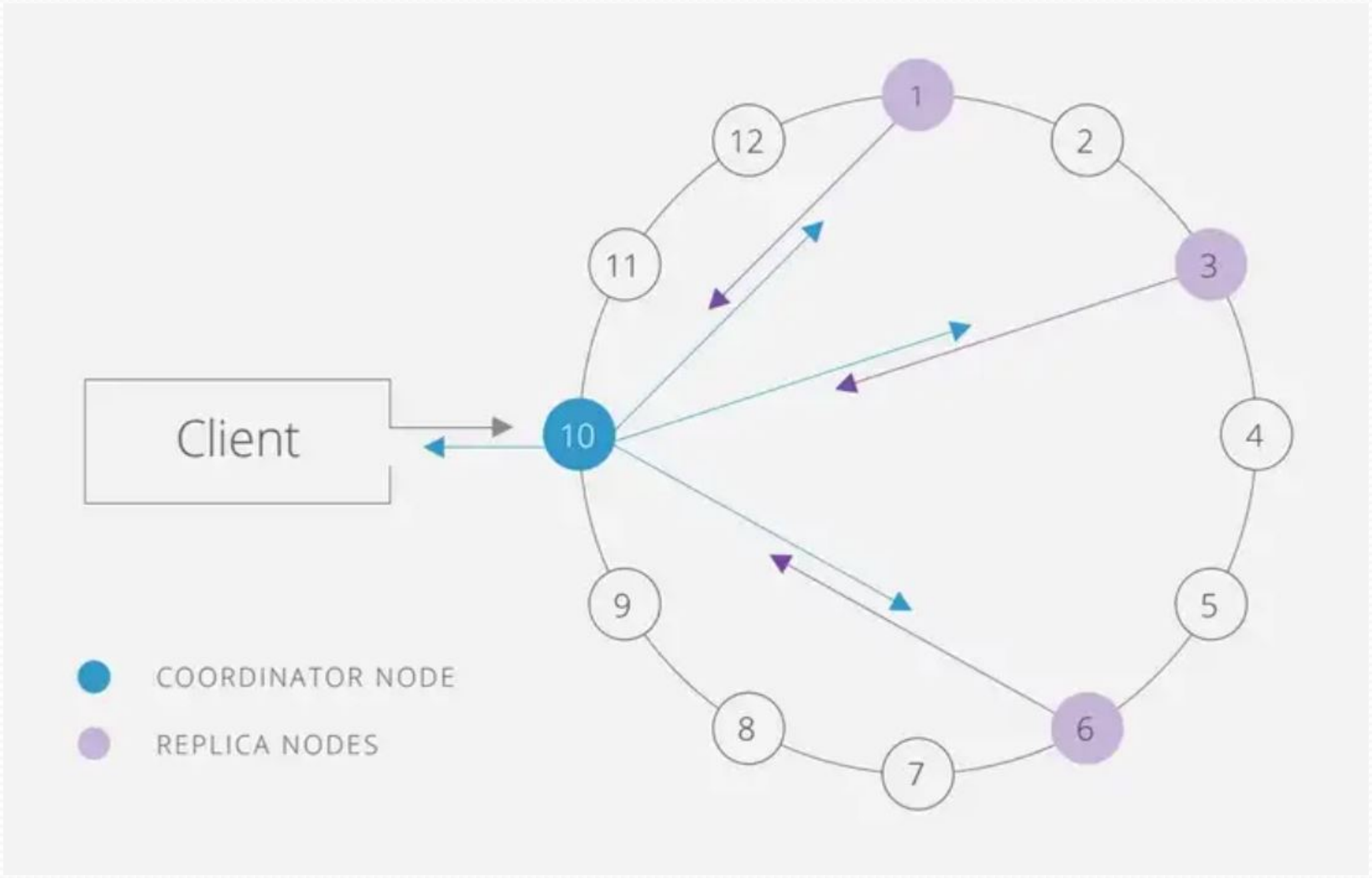
The following diagram shows the classification of different databases based on the CAP theorem.


Consistency
All clients see the same data at the same time
CA
CP
Availability
The system continues to operate even in the presence of node failures
AP
Partition-Tolerance
The system continues to operate in spite of network failures
SQL Server
MariaDB
APACHE HBASE
mongoDB
redis
CouchDB
cassandra
riak

Why absolute consistency is generally sacrificed?

- The large companies generally scales out horizontally, that means the network partition is present that is spread over thousands of nodes.
- Among consistency and availability, such companies have to choose only one as per the CAP theorem.
- Due to large number of network partitioning, there is high chance of failure of some nodes.
- If the data is not become available on time, it means a huge loss to the company.
- So, the company choose availability in terms of financial gain and Big Data trust from the client.
- In this sense, strict consistency is generally sacrificed.
- In other to maintain consistency, the company follow eventual consistency process instead of strict or absolute consistency.

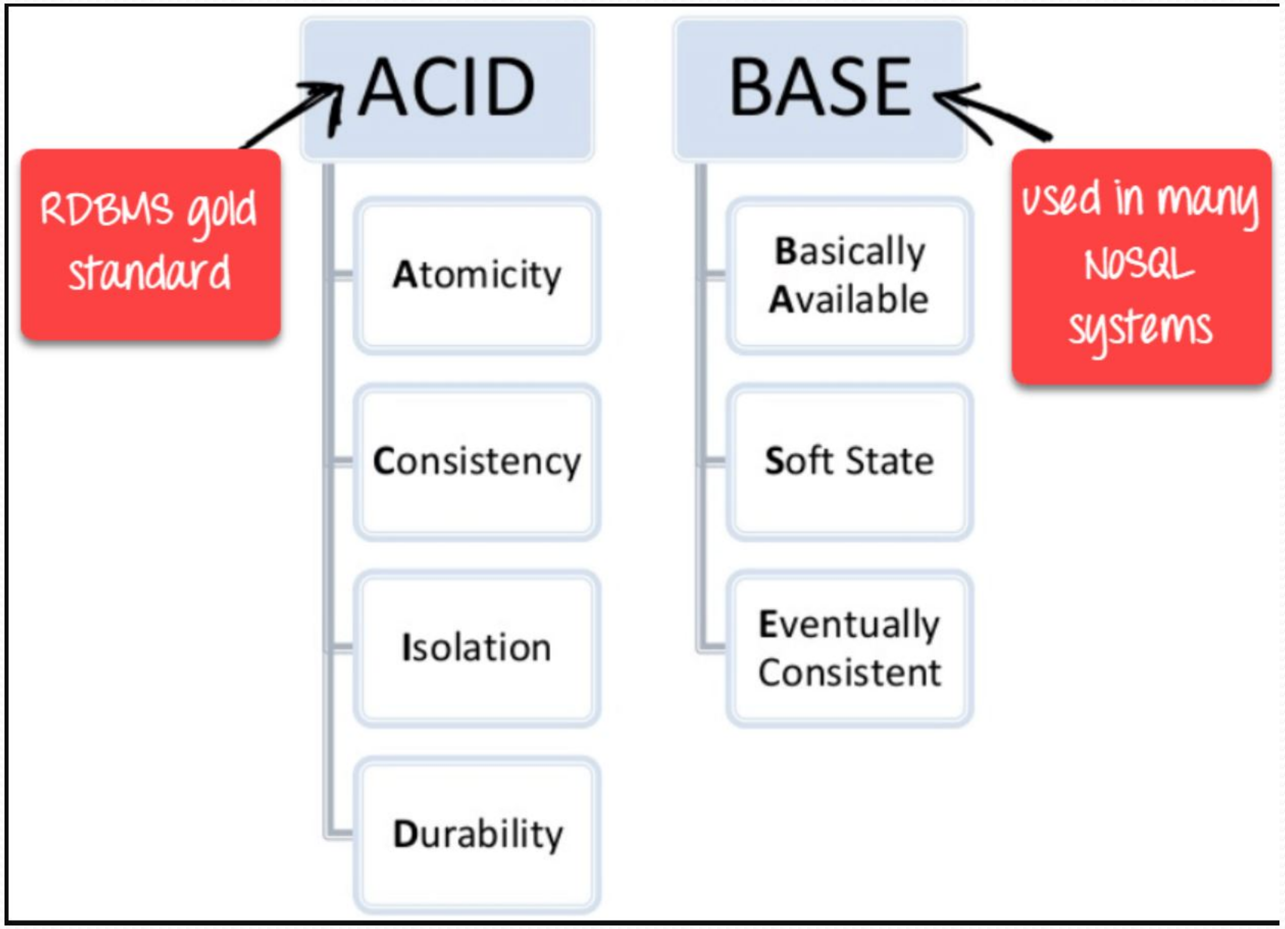
- **Consistency** technically means is that it refers to a situation where all the replica nodes have the exact same data at the exact same point in time.
- Consistency Level (CL): is the number of replica nodes that must acknowledge a read or write request for the whole operation/query to be successful.
  - Write CL controls how many replica nodes must acknowledge that they received and wrote the partition.
  - Read CL controls how many replica nodes must send their most recent copy of partition to the coordinator.

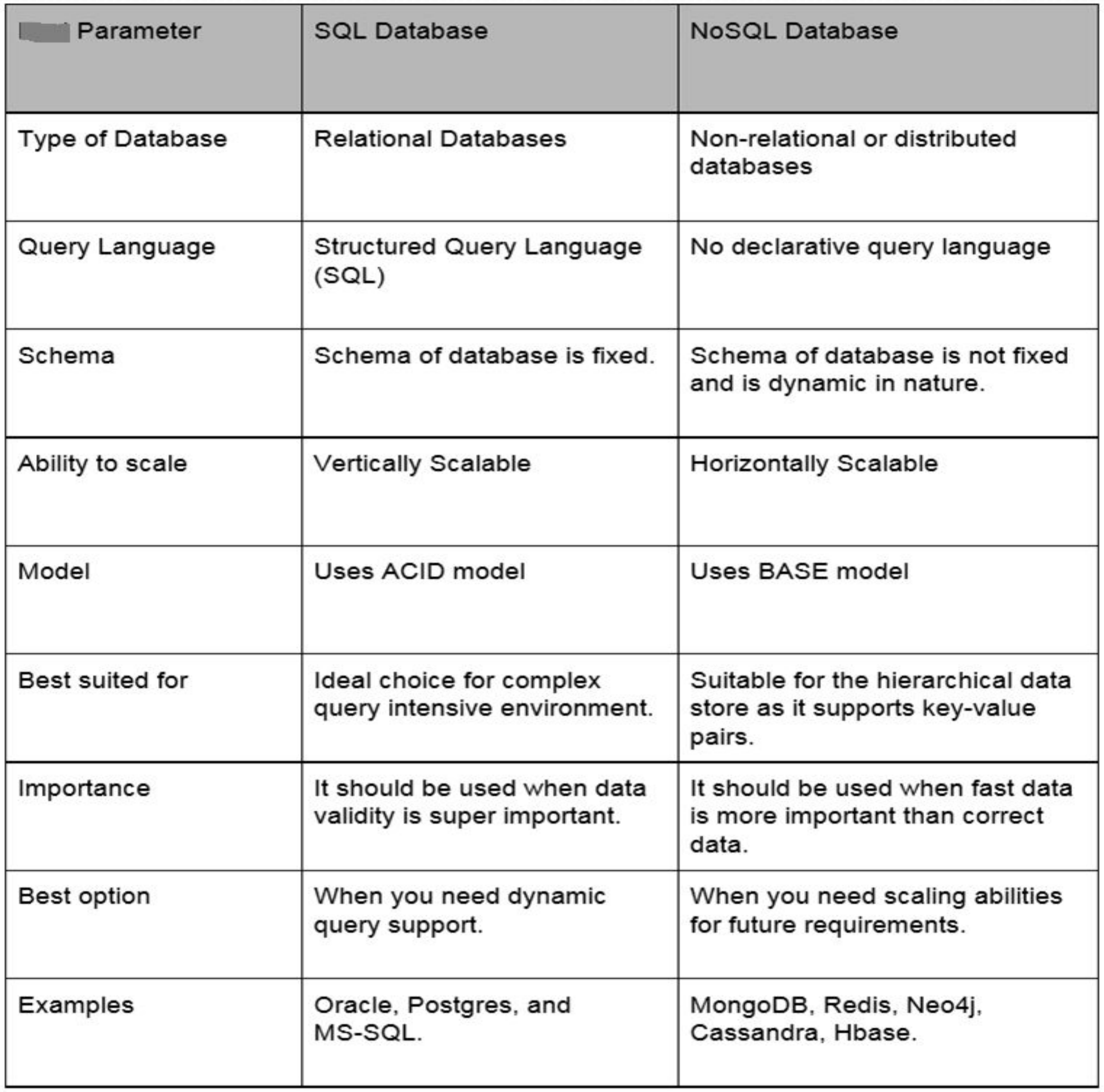

Client
1
2
3
4
5
6
7
8
9
10
11
12
COORDINATOR NODE
REPLICA NODES

- **Immediate consistency:** is having the identical data on all replica nodes at any given point in time.
- **Eventual consistency:** by controlling our read and write consistencies, we can allow our data to be different on our replica nodes, but our queries will still return the most correct version of the partition data.
- **Tunable Consistency** means that you can set the CL for each read and write request. So, you can contorl how consistent your data is. You can allow some queries to be immediately consistent and other queries to be eventually consistent.
- **Quorum consistency** is consistency for high mechanism and to ensure that how many nodes will respond when we will define the read and write consistency. In Quorum consistency a majority of (n/2 +1) nodes of the replicas must respond.

BASE: **B**asically **A**vailable, **S**oft state, **E**ventual consistency

- Basically, available means DB is available all the time as per CAP theorem
- Soft state means the state of the system could change over time
- Eventual consistency means that the system will become consistent over time


ACID
BASE
RDBMS gold standard
used in many NoSQL systems
Atomicity
Consistency
Isolation
Durability
Basically Available
Soft State
Eventually Consistent



[redacted] Parameter	SQL Database	NoSQL Database
Type of Database	Relational Databases	Non-relational or distributed databases
Query Language	Structured Query Language (SQL)	No declarative query language
Schema	Schema of database is fixed.	Schema of database is not fixed and is dynamic in nature.
Ability to scale	Vertically Scalable	Horizontally Scalable
Model	Uses ACID model	Uses BASE model
Best suited for	Ideal choice for complex query intensive environment.	Suitable for the hierarchical data store as it supports key-value pairs.
Importance	It should be used when data validity is super important.	It should be used when fast data is more important than correct data.
Best option	When you need dynamic query support.	When you need scaling abilities for future requirements.
Examples	Oracle, Postgres, and MS-SQL.	MongoDB, Redis, Neo4j, Cassandra, Hbase.

NoSQL

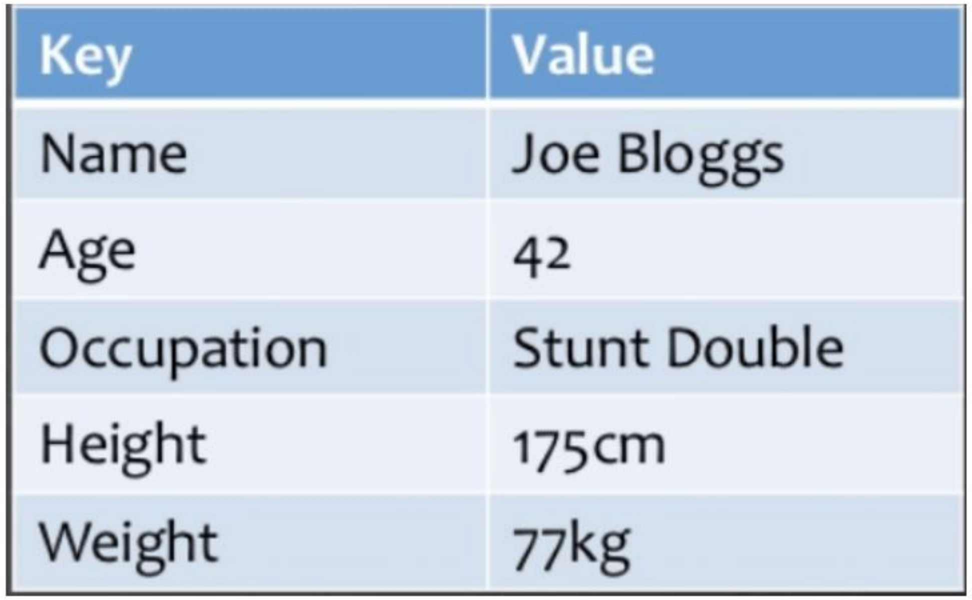
- **NoSQL** Database is a non-relational Data Management System, that does not require a fixed schema. It avoids joins, and is easy to scale.
- The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs.
- NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.
- **NoSQL database** stands for "Not Only SQL" or "Not SQL." Though a better term would be "NoREL", NoSQL
- Carl Strozz introduced the NoSQL concept in 1998.

NoSQL Taxonomy

- **NoSQL Databases** are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented.
- Every category has its unique attributes and limitations.
- None of the above-specified database is better to solve all the problems.
- Users should select the database based on their product needs.
- Types of NoSQL Databases:
  - Key-value Pair Based
  - Column-oriented Graph
  - Graphs based
  - Document-oriented

Key Value Pair Based

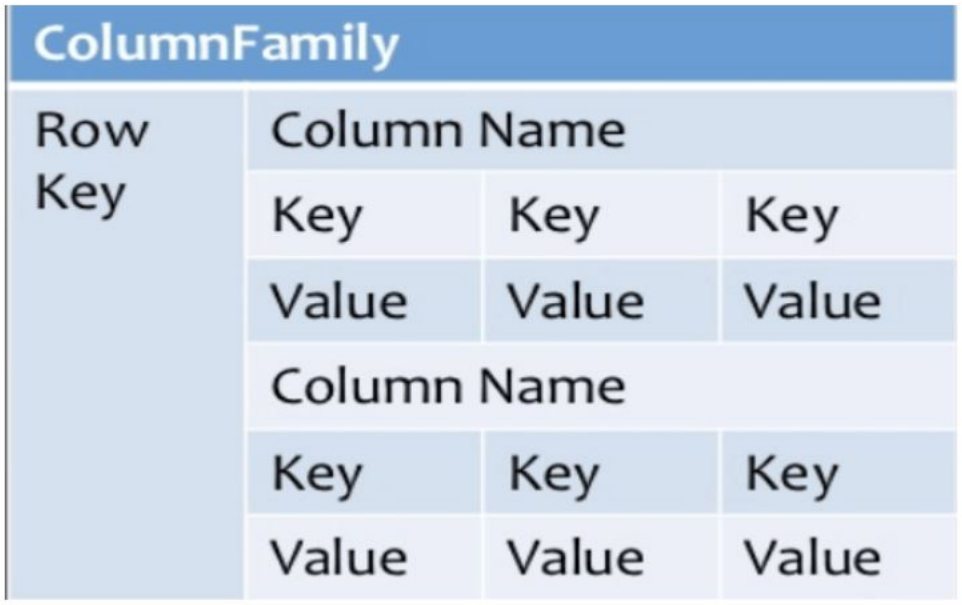
- Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.
- Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.
- Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. They are all based on Amazon's Dynamo paper.



Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

Column-based

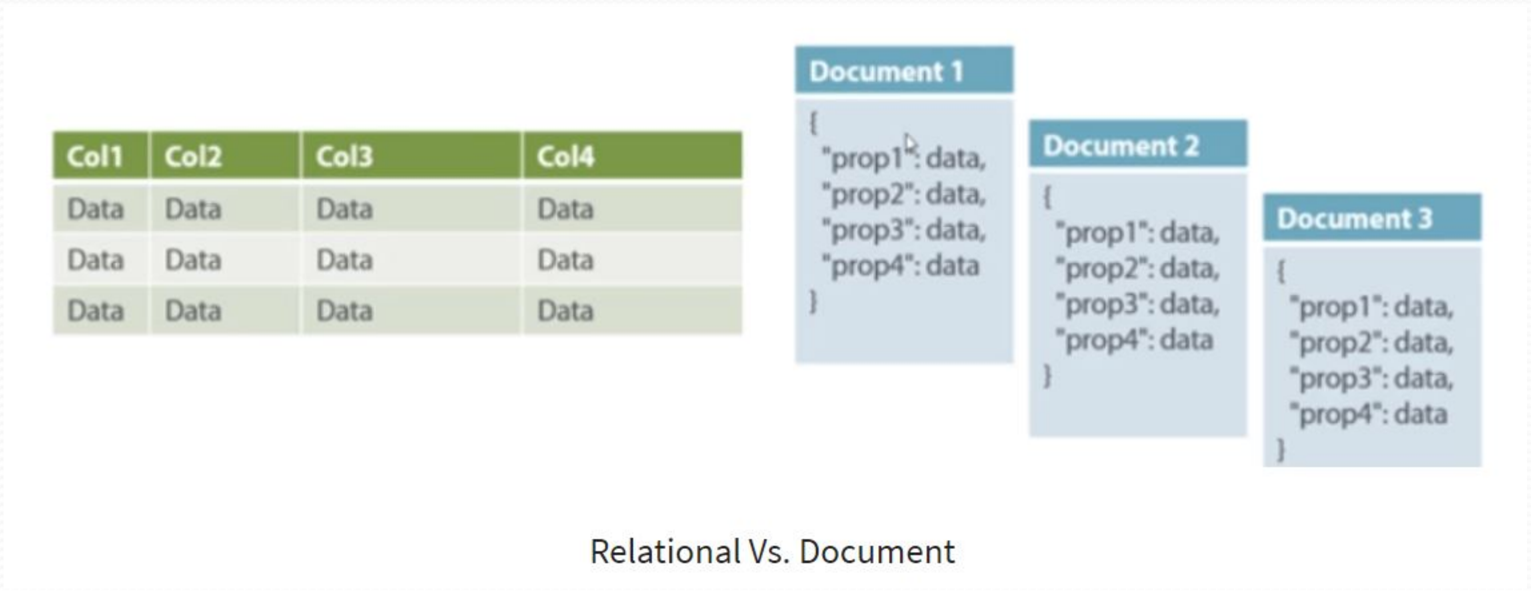
- Column-oriented databases work on columns and are based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously.
- Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs,
- HBase, Cassandra, HBase, Hypertable are NoSQL query examples of column based database.



ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

Document-Oriented

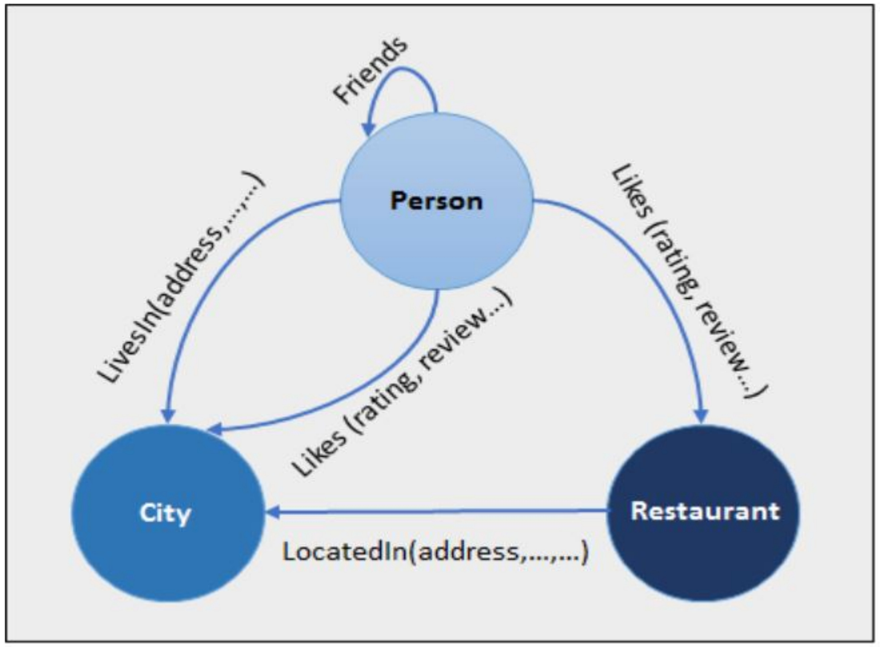
- Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.
- Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB, are popular Document originated DBMS systems



Relational Vs. Document

Graph-Based

- A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.
- Graph base database mostly used for social networks, logistics, spatial data.
- Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases.



Friends
Person
LivesIn(address,....,...)
Likes (rating, review...)
Likes (rating, review...)
City
Restaurant
LocatedIn(address,...,...)

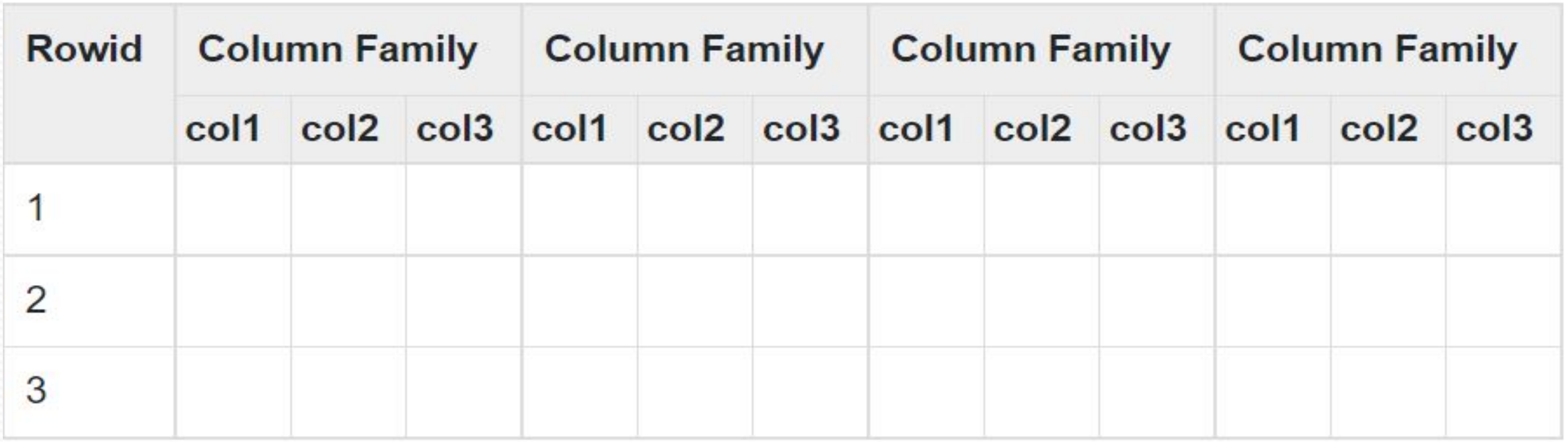
HBase

- HBase is a distributed **column-oriented** database built on top of the Hadoop file system. It is an open-source project and is **horizontally scalable**.
- Apache HBase is a column-oriented key/value data store built to run on top of the Hadoop Distributed File System (HDFS).
- Hadoop is a framework for handling large datasets in a distributed computing environment.

Storage Mechanism in HBase

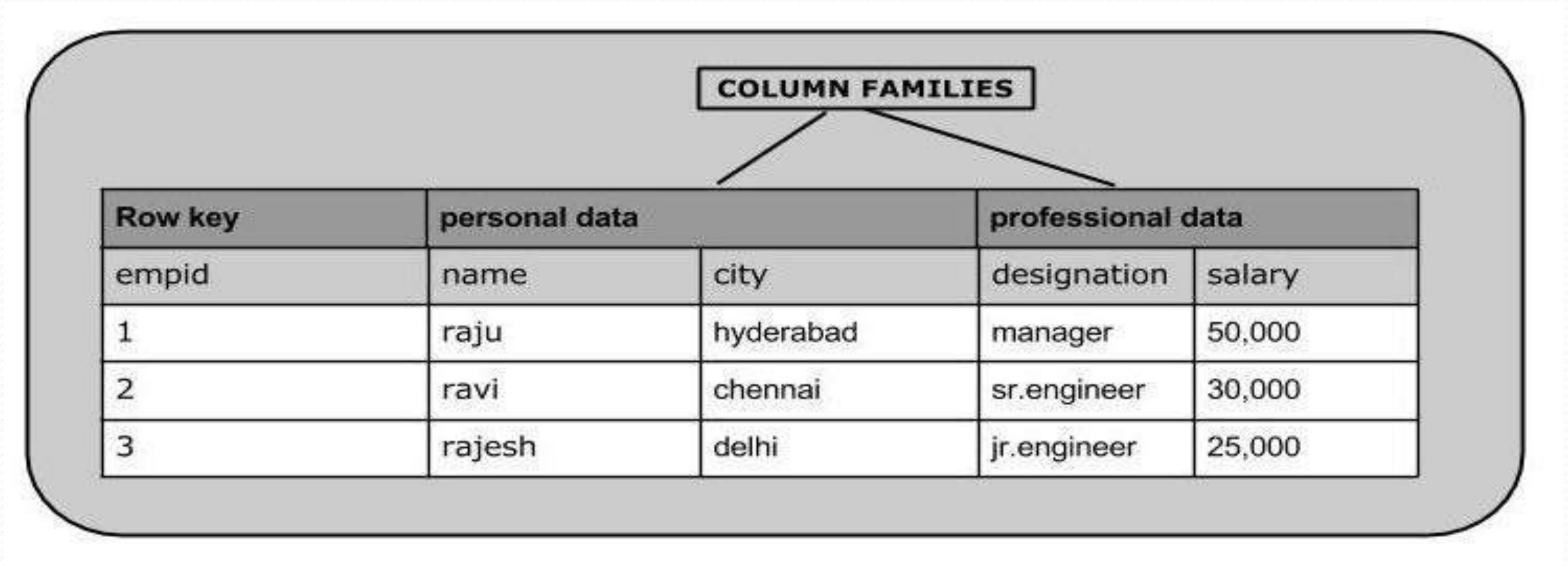
- HBase is a **column-oriented database** and the tables in it are sorted by row.
- The table schema defines only column families, which are the key value pairs.
- A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp.
- In short, in an HBase:
  - Table is a collection of rows.
  - Row is a collection of column families.
  - Column family is a collection of columns.
  - Column is a collection of key value pairs.

Given below is an example schema of table in HBase.



Rowid	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	col3
1												
2												
3												

Below shows column families in a column-oriented database

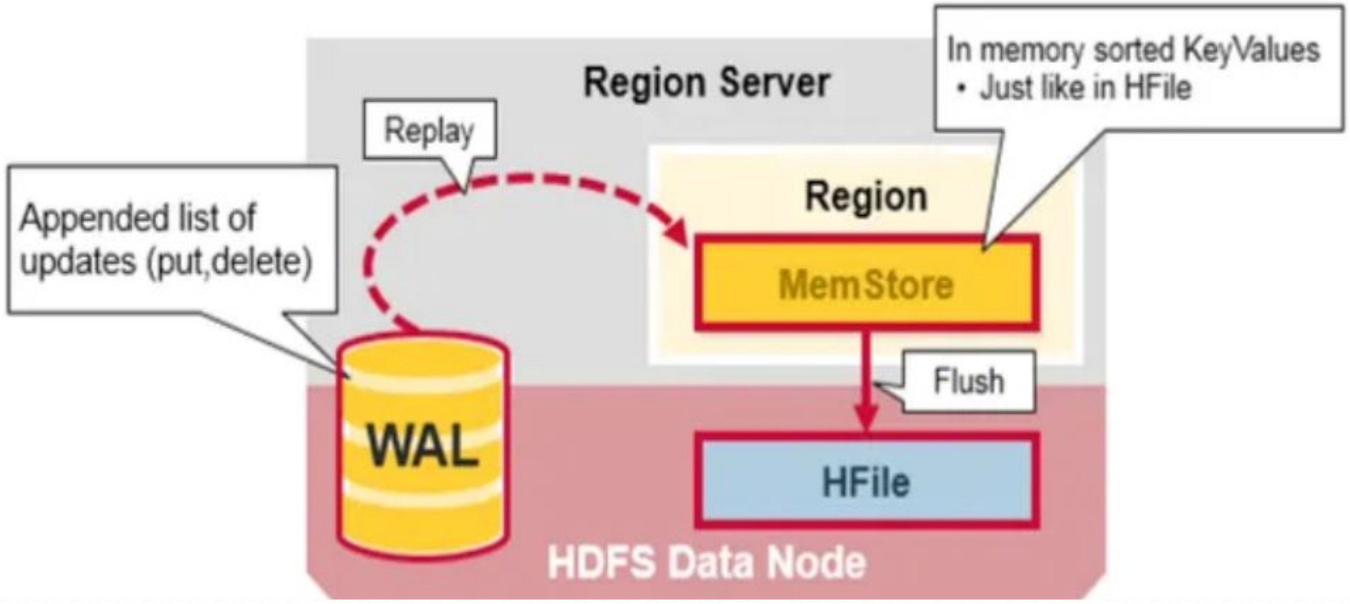


Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

Features of Hbase

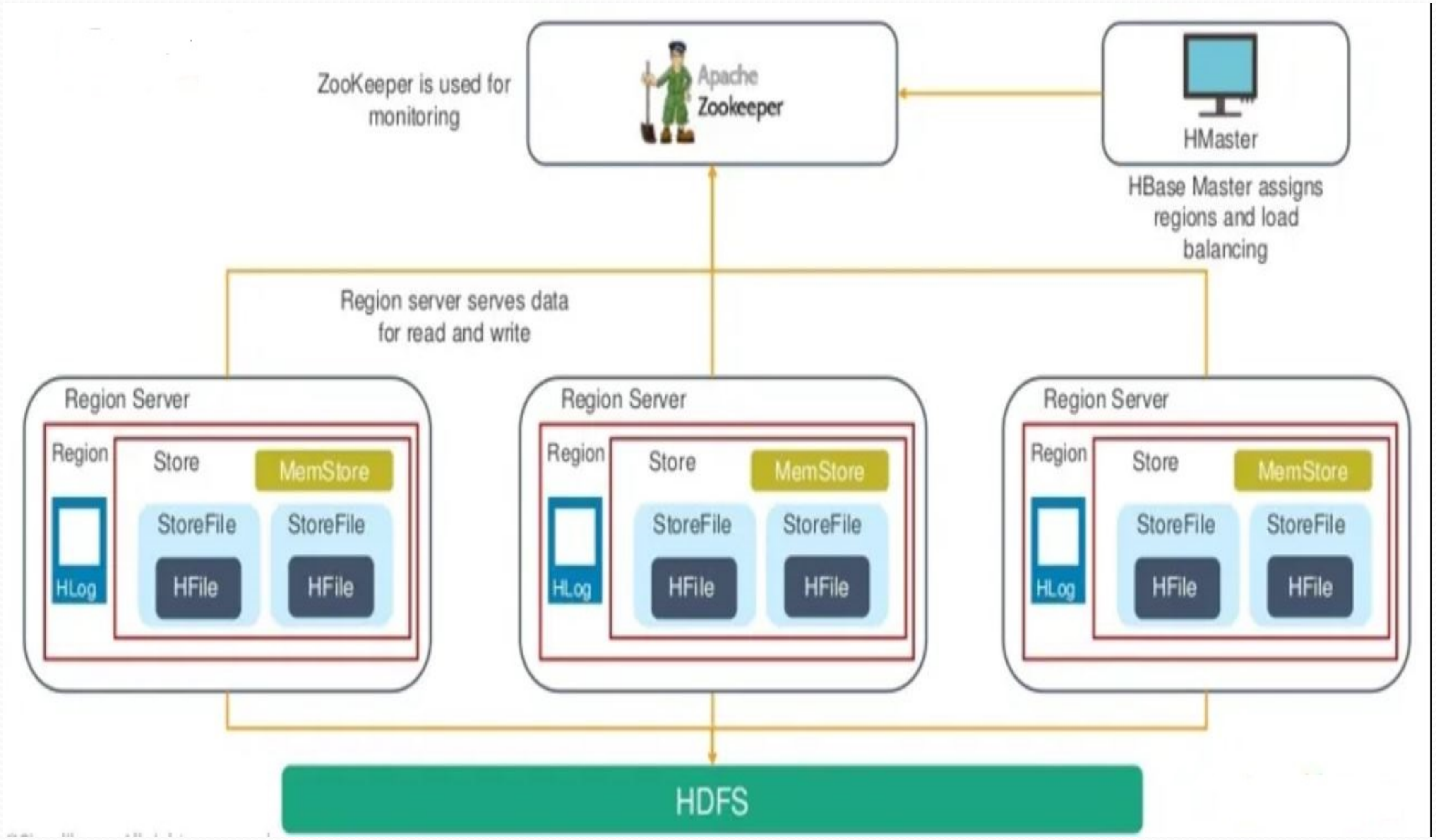
- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent read and writes.
- It integrates with Hadoop, both as a source and a destination.
- It has easy java API for client.
- It provides data replication across clusters.

Automatic Recovery from Failure using write ahead log(WAL)



- **HFile** : stores the rows of data as sorted keyvalue on disk
- **MemStore** : is a write cache that stores new data that has not yet been written to disk, there is one memstore per column family
- A HBase Store hosts a MemStore and 0 or more StoreFiles (HFiles). A Store corresponds to a column family for a table for a given region.
- **The Write Ahead Log** (WAL) records all changes to data in HBase, to file-based storage. if a RegionServer crashes or becomes unavailable before the MemStore is flushed, the WAL ensures that the changes to the data can be replayed.

Hbase Architecture


ZooKeeper is used for monitoring
Apache Zookeeper
HMaster
HBase Master assigns regions and load balancing
Region server serves data for read and write
Region Server
Region
Store
MemStore
StoreFile
HFile
HLog
HDFS

- HBase has three crucial components:
  - Zookeeper used for monitoring.
  - HMaster Server assigns regions and load-balancing.
  - Region Server serves data for write and read. it refers to different computers in the Hadoop cluster. Each Region Server have a region, HLog, a store ,memstore.

HMaster

- **HMaster** in HBase is the implementation of a Master server in HBase architecture.
- It acts as a monitoring agent to monitor all Region Server instances present in the cluster and acts as an interface for all the metadata changes.
- In a distributed cluster environment, Master runs on NameNode. Master runs several background threads.

HMaster

- The following are important roles performed by HMaster in HBase.
  - Plays a vital role in terms of performance and maintaining nodes in the cluster.
  - HMaster provides admin performance and distributes services to different region servers.
  - HMaster assigns regions to region servers.
  - HMaster has the features like controlling load balancing and failover to handle the load over nodes present in the cluster.
  - When a client wants to change any schema and to change any Metadata operations, HMaster takes responsibility for these operations.

HMaster

- Some of the methods exposed by HMaster Interface are primarily Metadata oriented methods.
  - Table (createTable, removeTable, enable, disable)
  - ColumnFamily (add Column, modify Column)
  - Region (move, assign)
- The client communicates in a bi-directional way with both HMaster and ZooKeeper. For read and write operations, it directly contacts with HRegion servers. HMaster assigns regions to region servers and in turn, check the health status of region servers.
- In entire architecture, we have multiple region servers. Hlog present in region servers which are going to store all the log files.

HBase Region Servers

- When HBase Region Server receives writes and read requests from the client, it assigns the request to a specific region, where the actual column family resides.
- However, the client can directly contact with HRegion servers, there is no need of HMaster mandatory permission to the client regarding communication with HRegion servers.
- The client requires HMaster help when operations related to metadata and schema changes are required.
- HRegionServer is the Region Server implementation. It is responsible for serving and managing regions or data that is present in a distributed cluster. The region servers run on Data Nodes present in the Hadoop cluster.

HBase Region Servers

- HMaster can get into contact with multiple HRegion servers and performs the following functions.
  - Hosting and managing regions
  - Splitting regions automatically
  - Handling read and writes requests
  - Communicating with the client directly

HBase Regions

- HRegions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families.
- It contains multiple stores, one for each column family.
- It consists of mainly two components, which are Memstore and Hfile.

ZooKeeper

- HBase Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization. Distributed synchronization is to access the distributed applications running across the cluster with the responsibility of providing coordination services between nodes. If the client wants to communicate with regions, the server's client has to approach ZooKeeper first.
- It is an open source project, and it provides so many important services.

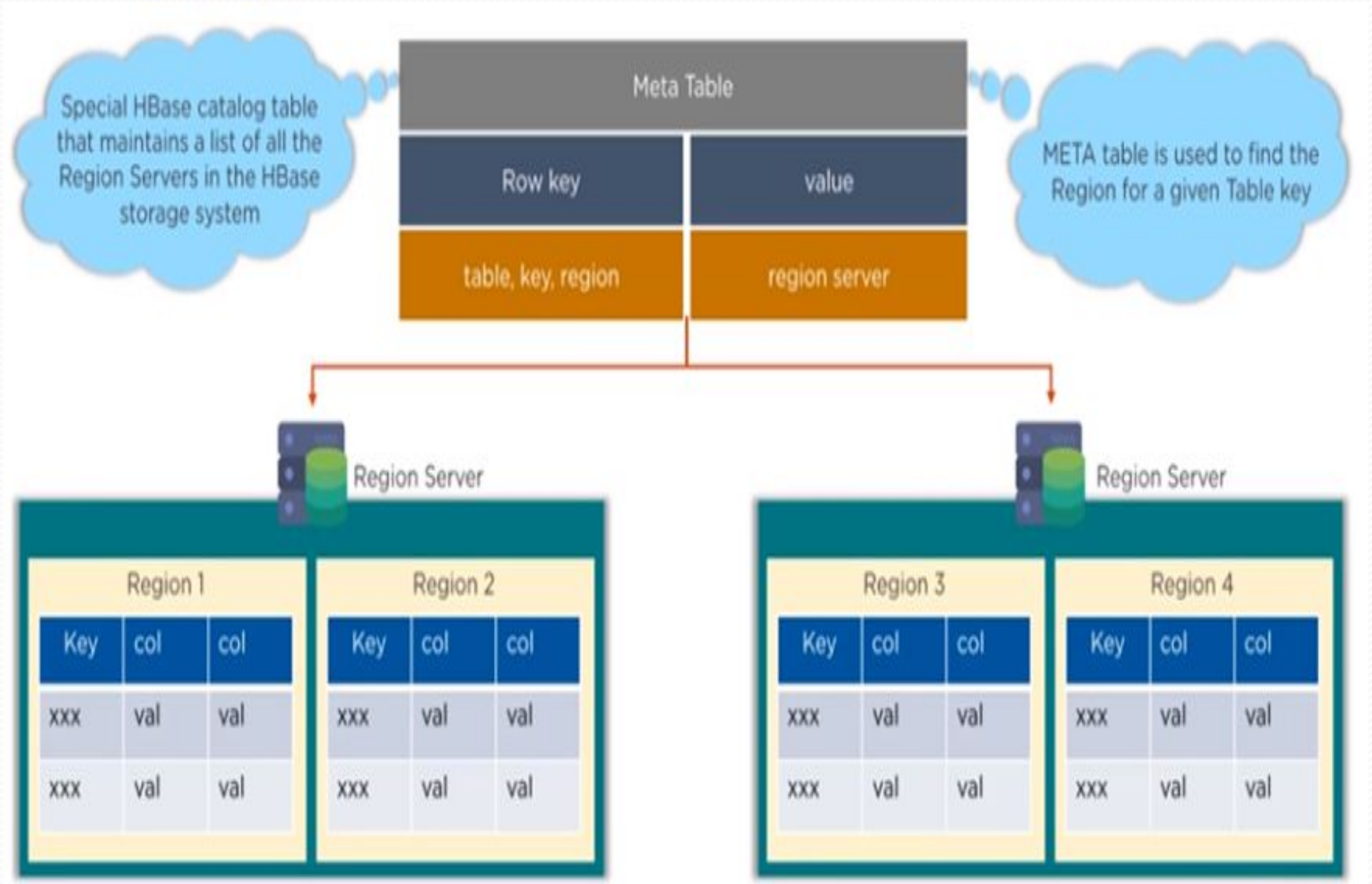
ZooKeeper

- Services provided by ZooKeeper
  - Maintains Configuration information
  - Provides distributed synchronization
  - Client Communication establishment with region servers
  - Provides ephemeral nodes for which represent different region servers
  - Master servers usability of ephemeral nodes for discovering available servers in the cluster
  - To track server failure and network partitions

ZooKeeper

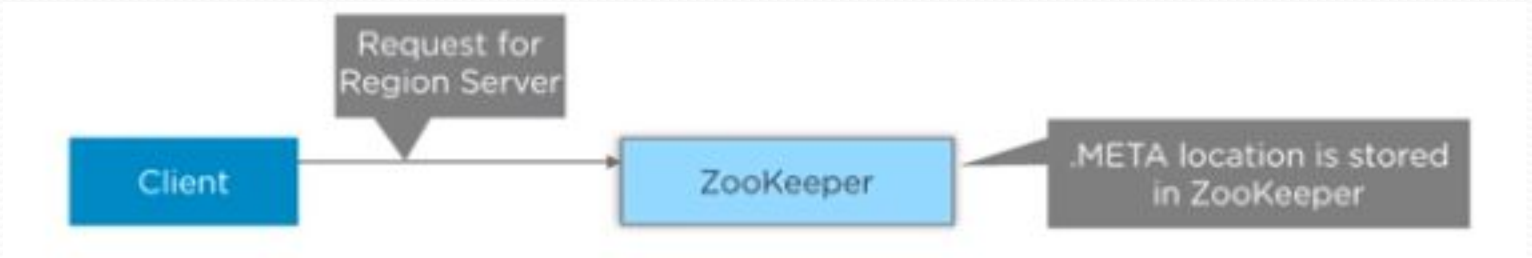
- Master and HBase slave nodes (region servers) registered themselves with ZooKeeper. The client needs access to ZK(zookeeper) quorum configuration to connect with master and region servers.
- During a failure of nodes that present in HBase cluster, ZKquoram will trigger error messages, and it starts to repair the failed nodes.

- **META table** which holds the location of the regions in the cluster

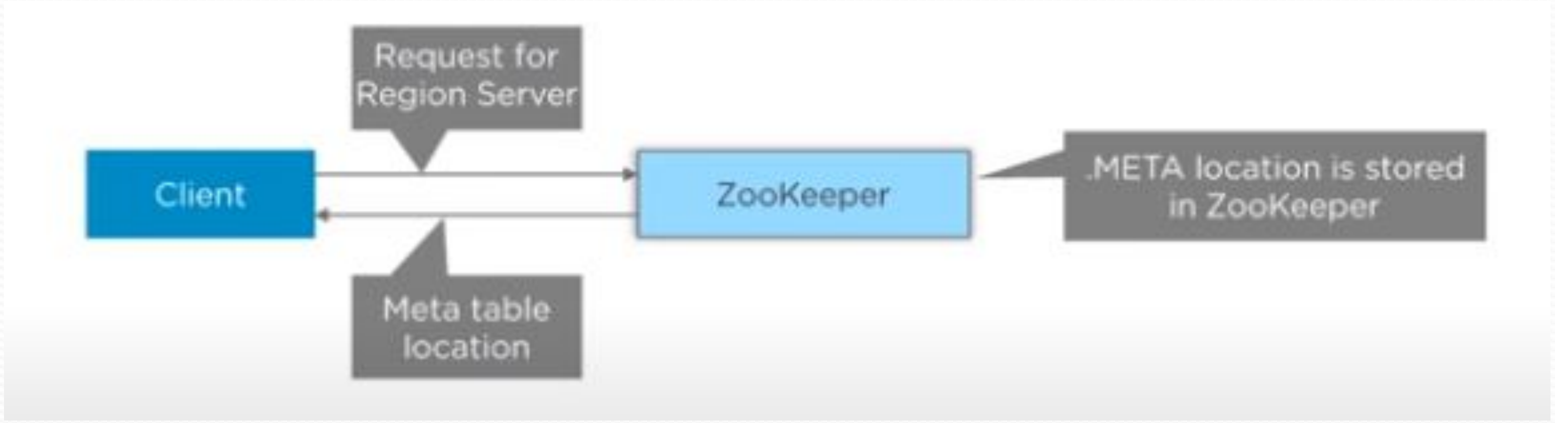

Special HBase catalog table that maintains a list of all the Region Servers in the HBase storage system
Meta Table
Row key
value
table, key, region
region server
META table is used to find the Region for a given Table key
Region Server
Region Server
Region 1
Region 2
Region 3
Region 4
Key
col
col
xxx
val
val
xxx
val
val

HBase : read mechanism

- The client sends a request to get the region server that hosts the META table from ZooKeeper.

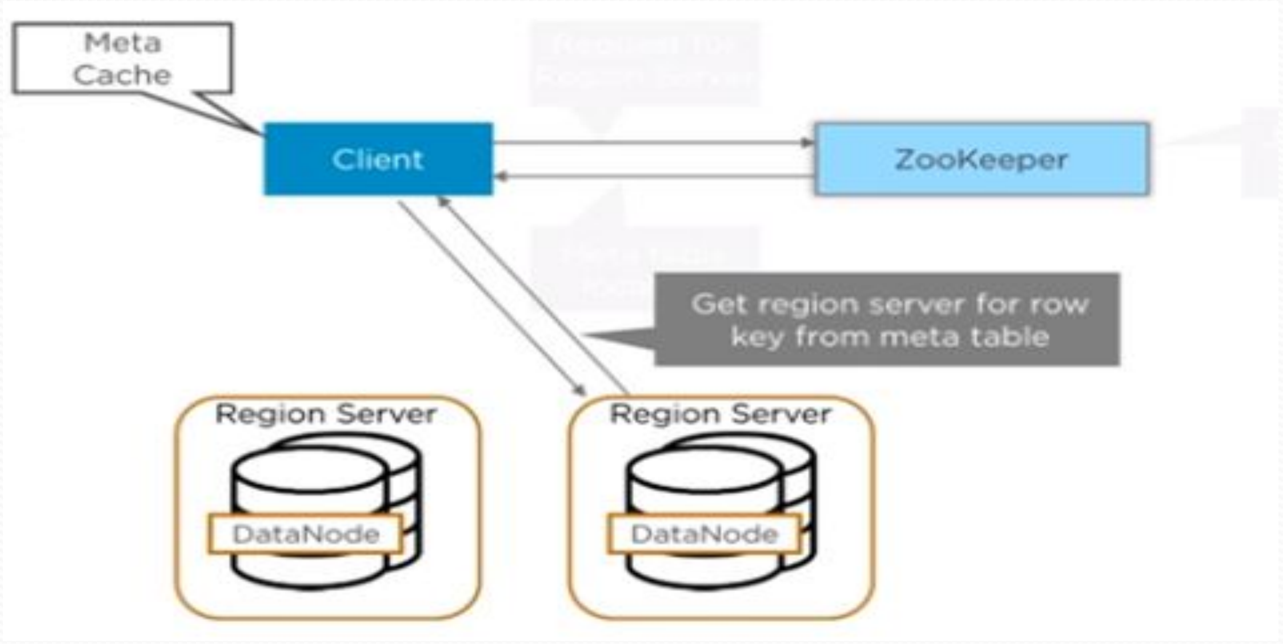

Request for Region Server
Client
ZooKeeper
.META location is stored in ZooKeeper

- The Zookeeper replies by sending the META table location.


Request for Region Server
Client
ZooKeeper
.META location is stored in ZooKeeper
Meta table location

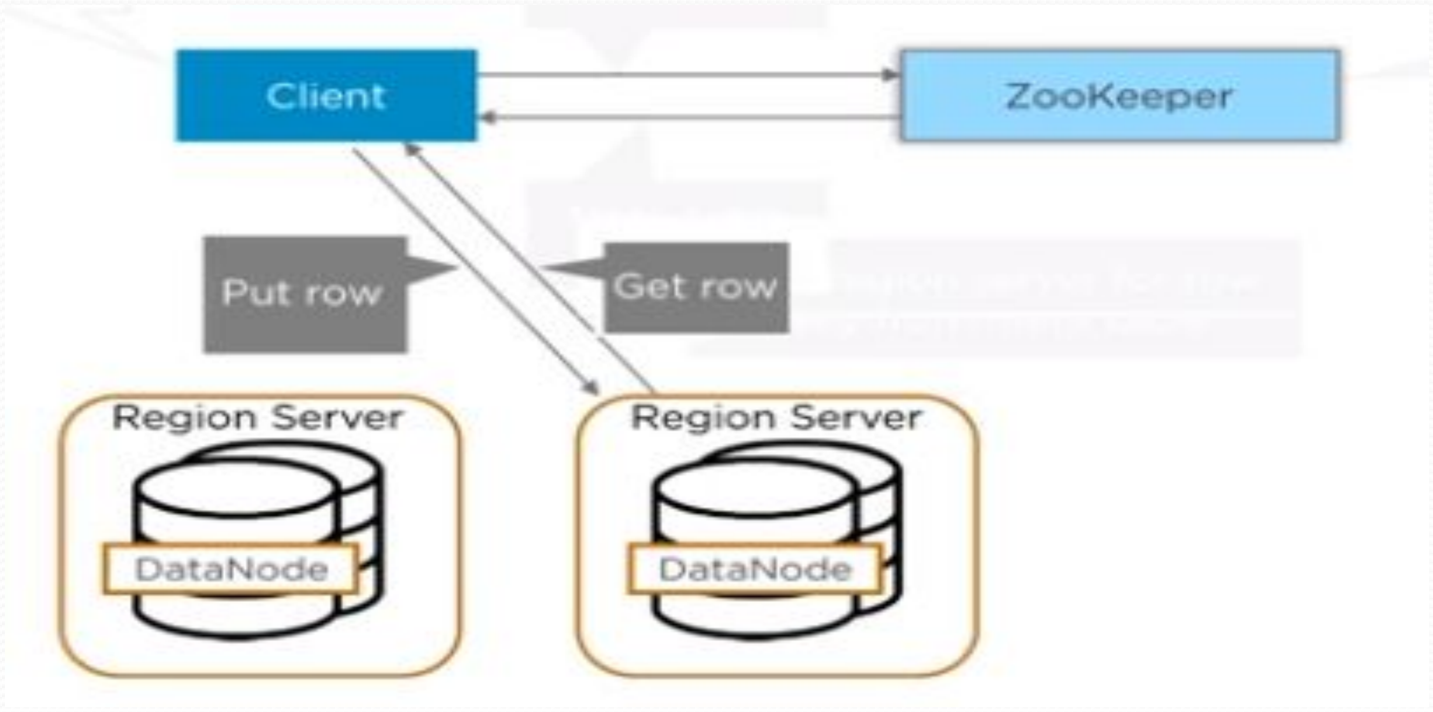
HBase : read mechanism

- The client will query the META server to get the region server corresponding to the row key it wants to access
- The client caches this information along side with the META table location

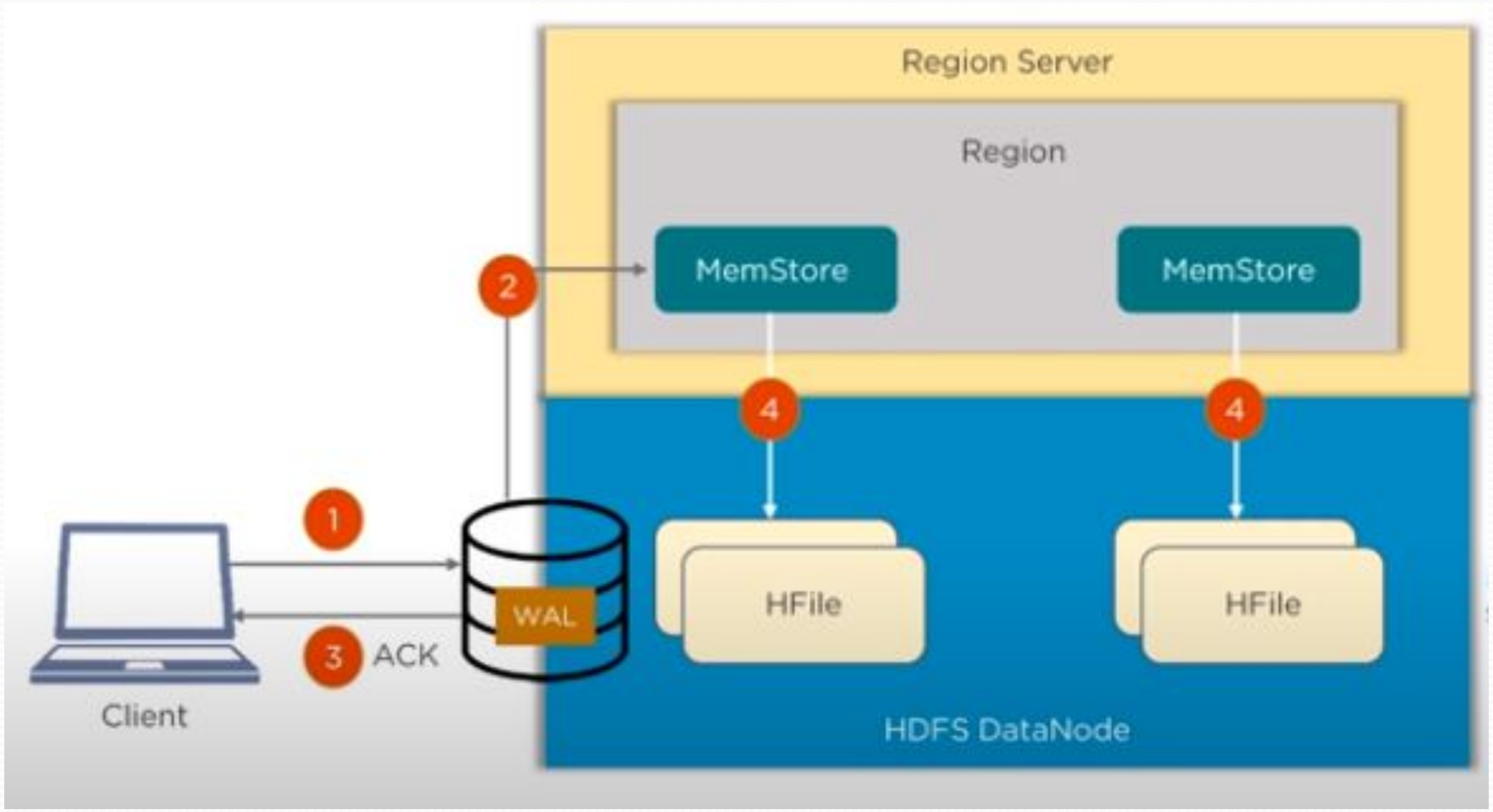


HBase : read mechanism

- Finally the region Server answer with the row key, so now it could get row or rows.



Hbase write Mechanism



Hbase write Mechanism

1. write the data to the write-ahead-log (WAL), HBase always has WAL to look into, if any error occurs while writing data.
2. Once the data is written to the WAL, it is then copied to the MemStore
3. Once the data is placed in the MemStore, the client then receives the acknowledgement (ACK)
4. When the MemStore reaches the threshold, it dumps or commit the data into HFile

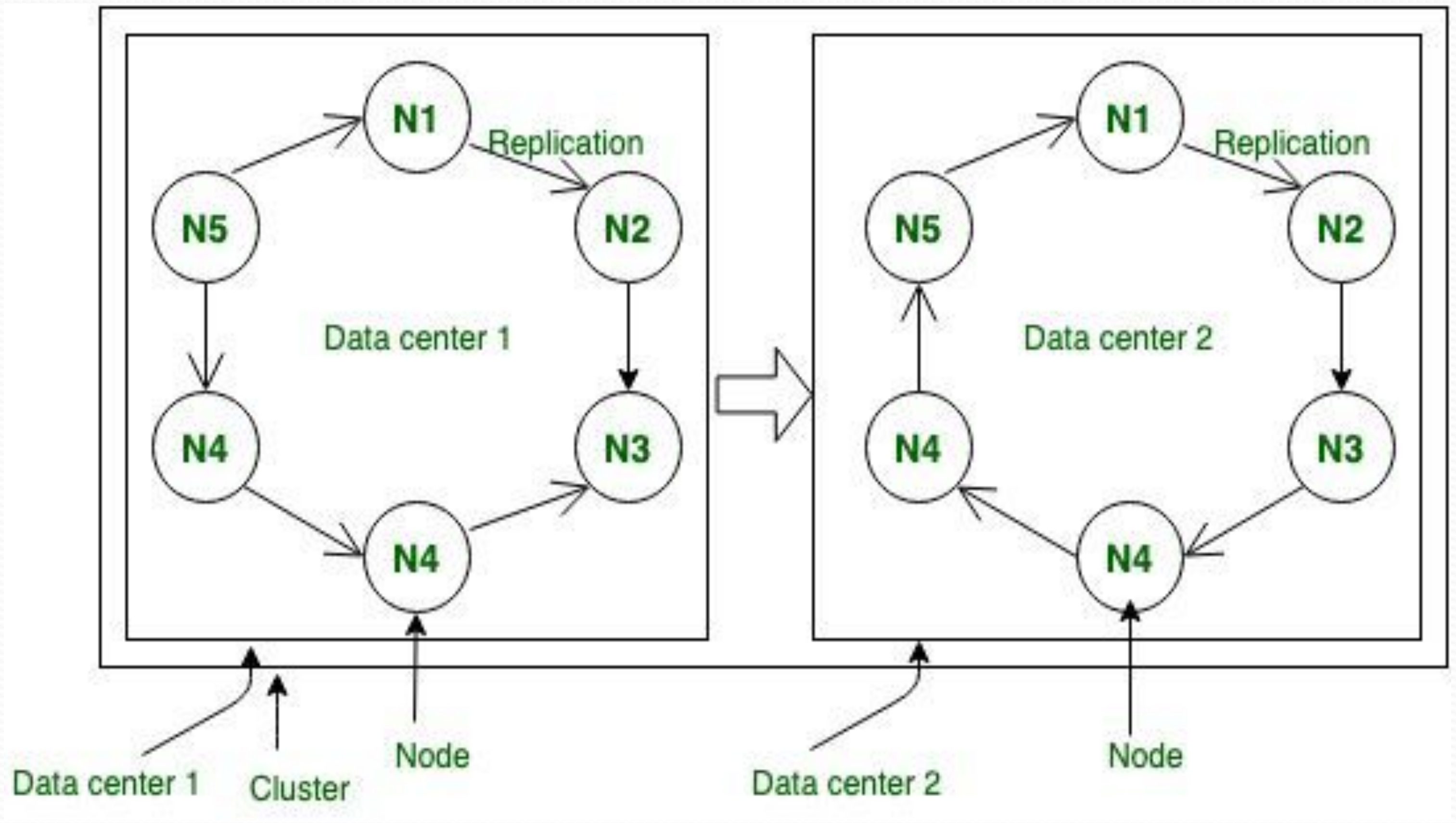
Cassandra

- Apache Cassandra is an open-source, distributed, NoSQL database. It presents a partitioned wide column storage model with eventually consistent semantics.
- The reasons for choosing Cassandra are as follows:
  - Value availability over consistency
  - Require high write throughput
  - High scalability required
  - No single point of failure

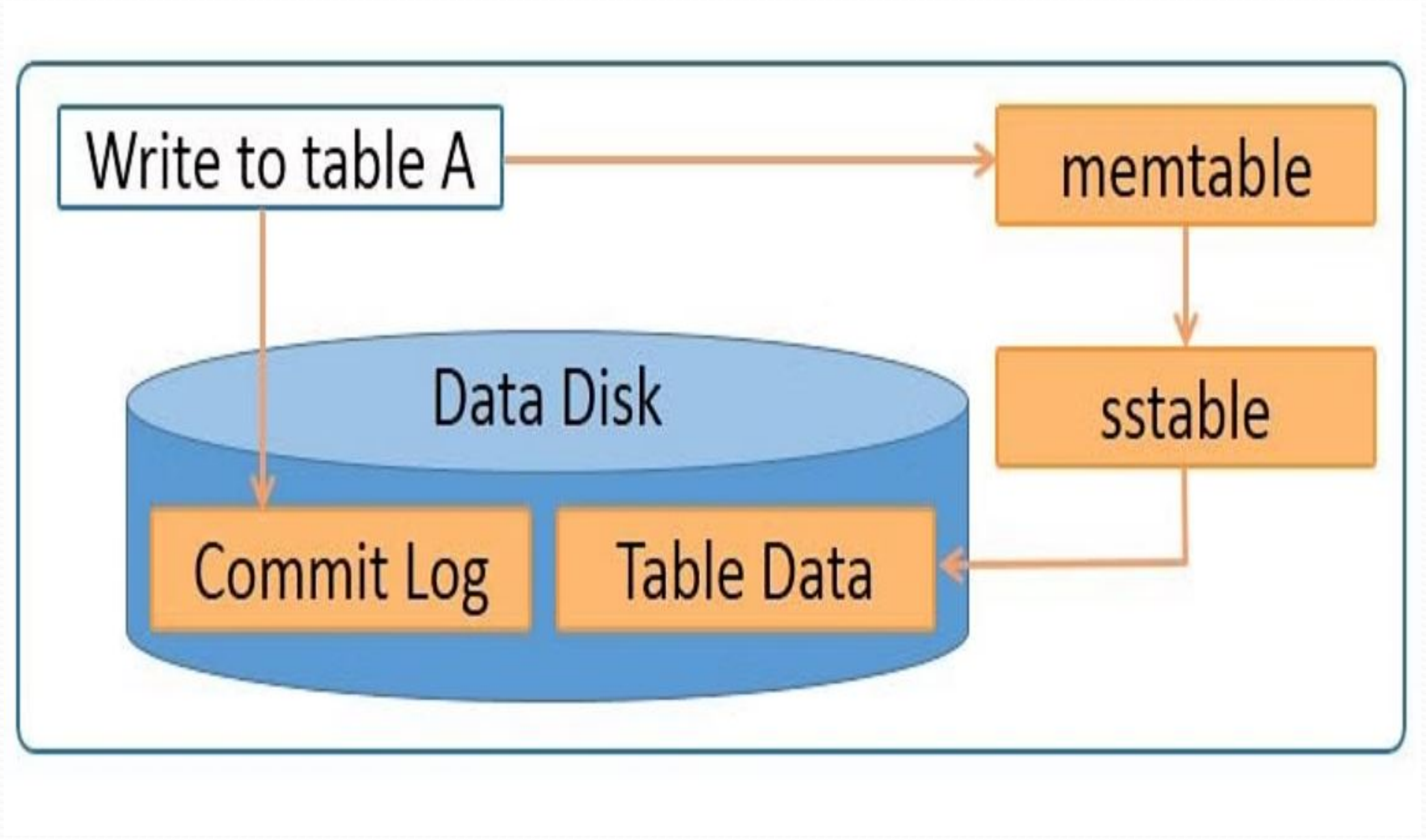
- **Features of Cassandra Architecture**
  - No masters and slaves (Peer to peer).
  - Ring type architecture
  - Automatic data distribution across all nodes
  - Replication of data across nodes
  - Data kept in memory and written to disk in a lazy fashion
  - Hash values of the keys are used to distribute data among nodes

- **The key components of Cassandra are as follows:**
  - **Node** - place where data is stored
  - **Data center** - collection of related nodes
  - **Cluster** - collection of one or more data centers
  - **Commit logs** - write operation is written for crash recovery
  - **Mem table** - after commit log, data is written to mem table
  - **Sstable (Sorted Strings Table)** - disk file to which data is flushed from mem table when its contents reach threshold value
  - **Bloom filter** - algorithm for testing whether the element is a member of a set

Cassandra : Ring type architecture



Cassandra : Write process



Cassandra Data Model

- Cassandra is a NoSQL database, which is a key-value store. Some of the features of Cassandra data model are as follows:
- Data in Cassandra is stored as a set of rows that are organized into tables.
- Tables are also called column families.
- Each Row is identified by a primary key value.
- Data is partitioned by the primary key.
- We can get the entire data or some data based on the primary key.

Tunable Consistency – Write CL (Consistency Level)

- Write consistency means having consistent data (immediate or eventual) after your write query to your Cassandra cluster.
- Write CL controls how many replica nodes must acknowledge that they received and wrote the partition.
- You can tune the write consistency for performance (by setting the write CL as ONE) or immediate consistency for critical piece of data (by setting the write CL as ALL) Following is how it works:
- A client sends a write request to the coordinator.
- The coordinator forwards the write request (INSERT, UPDATE or DELETE) to all replica nodes whatever write CL you have set.
- The coordinator waits for n number of replica nodes to respond. n is set by the write CL.
- The coordinator sends the response back to the client.

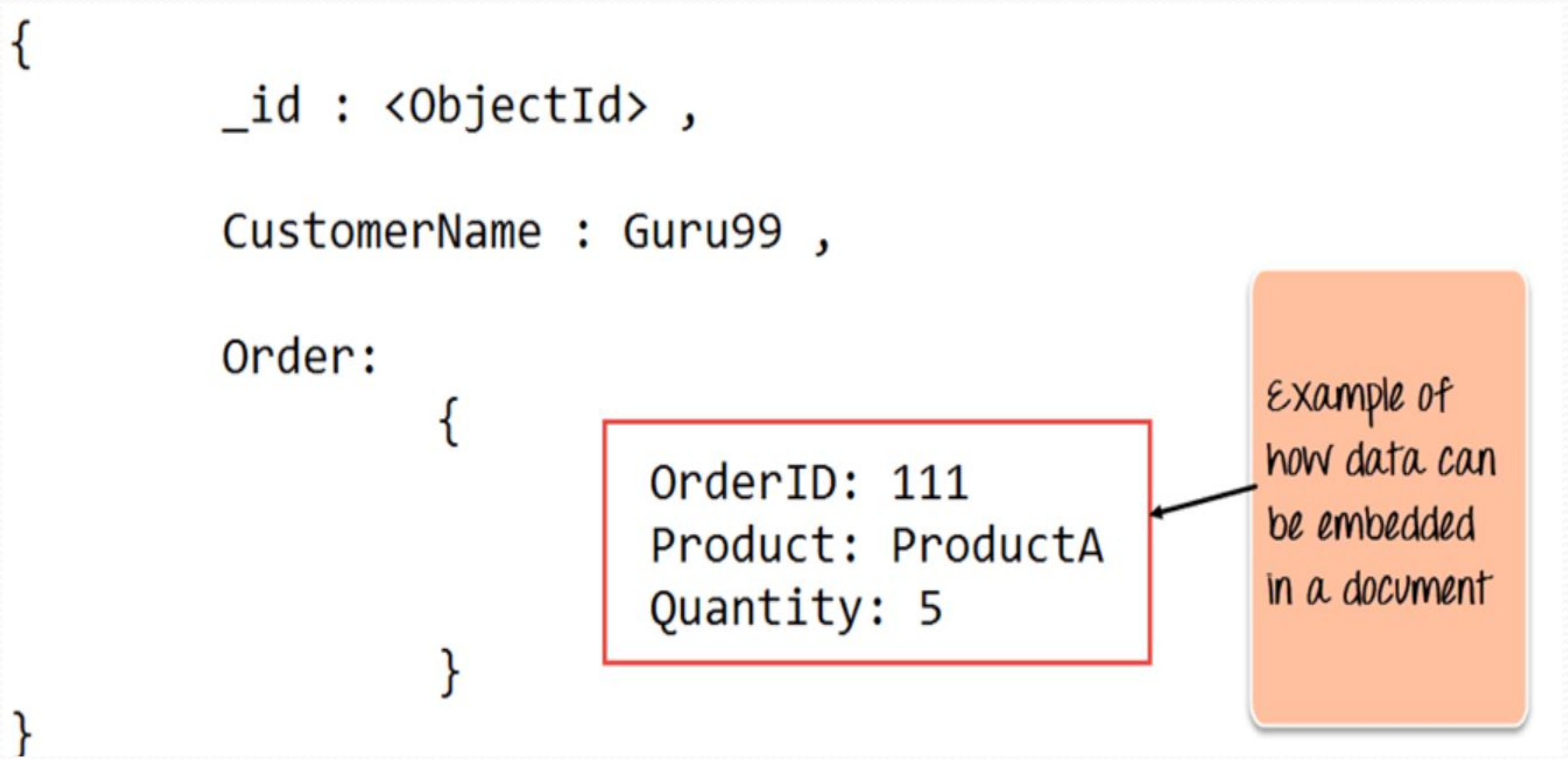
Tunable Consistency – Read CL (Consistency Level)

- Read CL controls how many replica nodes must send their most recent copy of partition to the coordinator.
- Read consistency refers to having same data on all replica nodes for any read request. Following is how it works:
  - A client sends a read request to the coordinator.
  - The coordinator forwards the read (SELECT) request to n number of replica nodes. n is set by the read CL.
  - The coordinator waits for n number of replica nodes to respond.
  - The coordinator then merges (finds out most recent copy of written data) the n number of responses to a single response and sends response to the client.

MongoDB

- **MongoDB** is a document-oriented NoSQL database used for high volume data storage.
- Instead of using tables and rows as in the relational databases, MongoDB makes use of collections and documents.
- Documents consist of key-value pairs which are the basic unit of data in MongoDB.
- Collections contain sets of documents and function which is the equivalent of relational database tables.

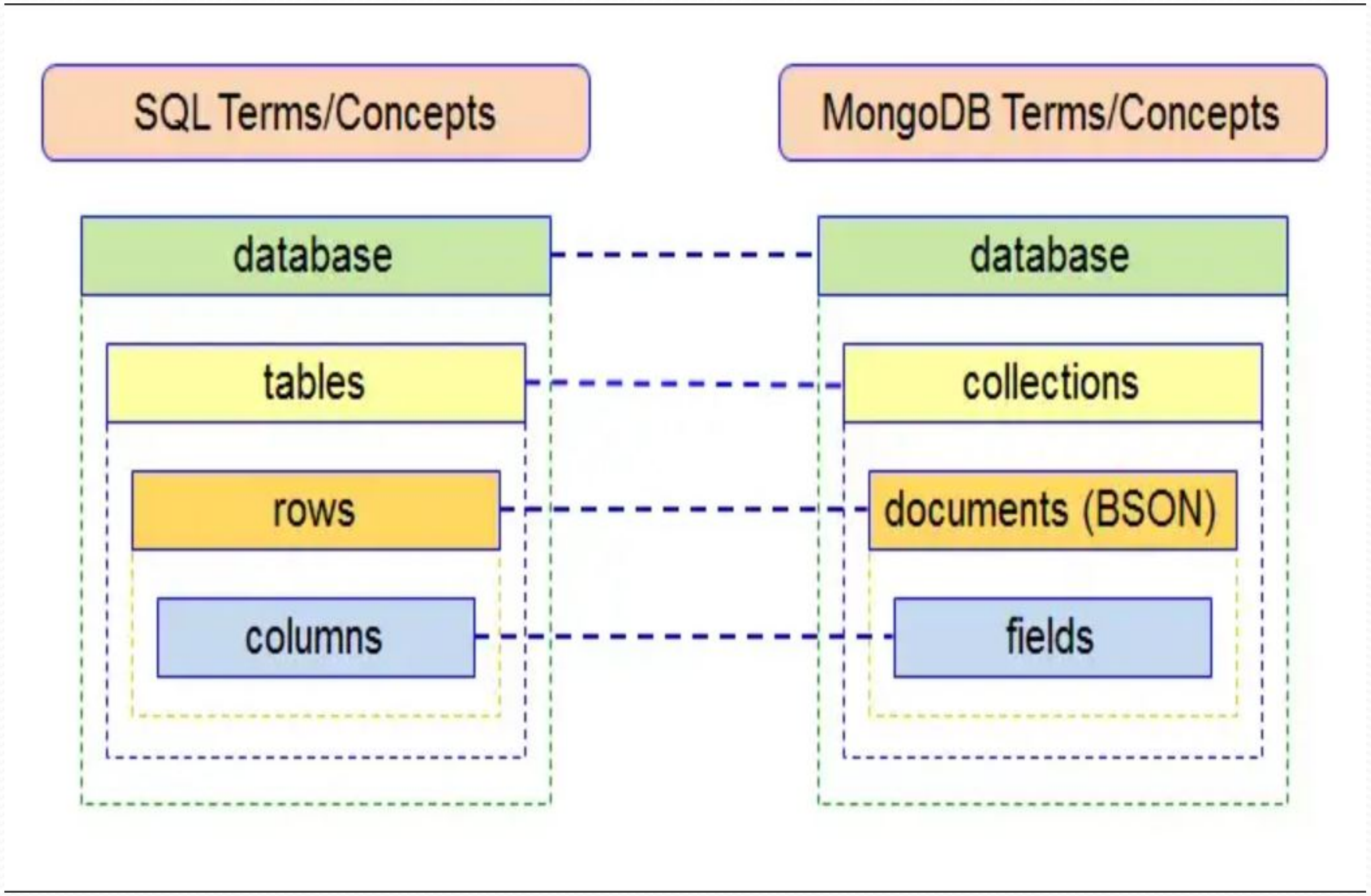
MongoDB Example



```
{
        _id : <ObjectId> ,

        CustomerName : Guru99 ,

        Order:
                {
                        OrderID: 111
                        Product: ProductA
                        Quantity: 5
                }
}
```

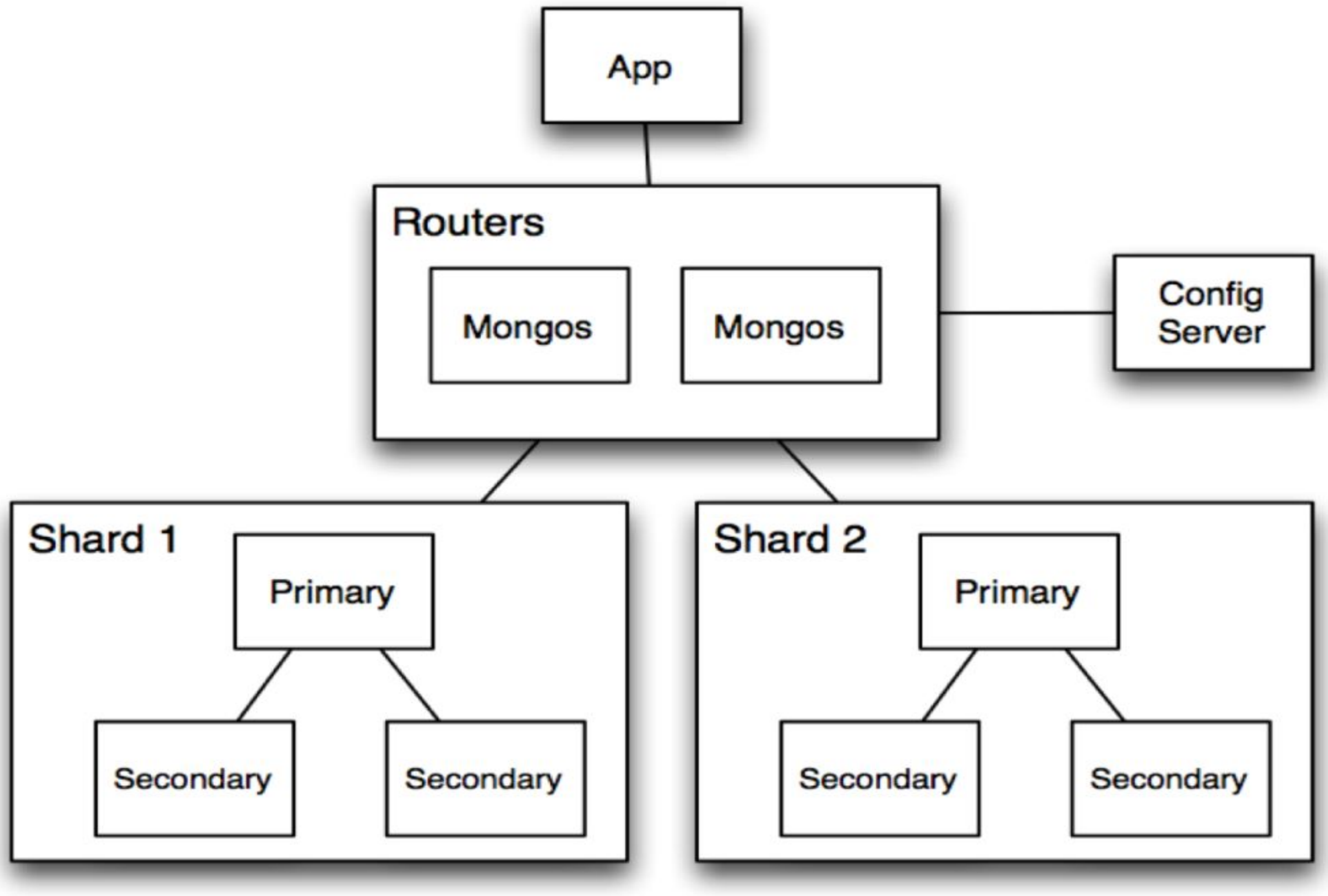

SQL Terms/Concepts
MongoDB Terms/Concepts
database
database
tables
collections
rows
documents (BSON)
columns
fields

- **Database:** Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.
- **Collection :** Collection is a group of documents and is similar to an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields.
- **Document :** A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Sharding

- Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.
- Database systems with large data sets or high throughput applications can challenge the capacity of a single server. For example, high query rates can exhaust the CPU capacity of the server. Working set sizes larger than the system's RAM stress the I/O capacity of disk drives.
- MongoDB supports horizontal scaling through sharding.

Sharding cluster



- **Shard**: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set to provide redundancy and high availability. Together, the cluster's shards hold the entire data set for the cluster.
- **Mongos:** The mongos acts as a query router, providing an interface between client applications and the sharded cluster.
- **Config Servers:** Config servers store metadata and configuration settings for the cluster. They are also deployed as a replica set.

References

- https://medium.com/@yuneeh/unstructured-data-vs-structured-data-explained-with-real-life-examples-a62dbadbb49d
- https://medium.com/@varun.sja/structured-data-vs-unstructured-data-vs-semi-structured-data-what-is-the-difference-f0e88eaba560
- https://medium.com/analytics-vidhya/vertical-vs-horizontal-scaling-b2754d68d77f
- https://www.spiceworks.com/tech/cloud/articles/horizontal-vs-vertical-cloud-scaling/
- https://medium.com/hands-on-apache-hbase/an-introduction-to-apache-hbase-2cdd1d9ff13

References

- https://medium.com/hands-on-apache-hbase/an-introduction-to-apache-hbase-2cdd1d9ff13#:~:text=HBase%20is%20a%20distributed%20column,Distributed%20File%20System%20(HDFS)
- https://www.guru99.com/hbase-architecture-data-flow-usecases.html
- https://informationit27.medium.com/hbase-architecture-1d508455fe65
- https://medium.com/@wangwei09310931/apache-hbase-a-brief-introduction-7e2e3a1bbc91
- https://medium.com/@aymannaitcherif/beginners-guide-to-learn-cassandra-part-1-cassandra-overview-bf1634e4ce30
- https://medium.com/nerd-for-tech/all-basics-of-mongodb-in-10-minutes-baddaf6b6625