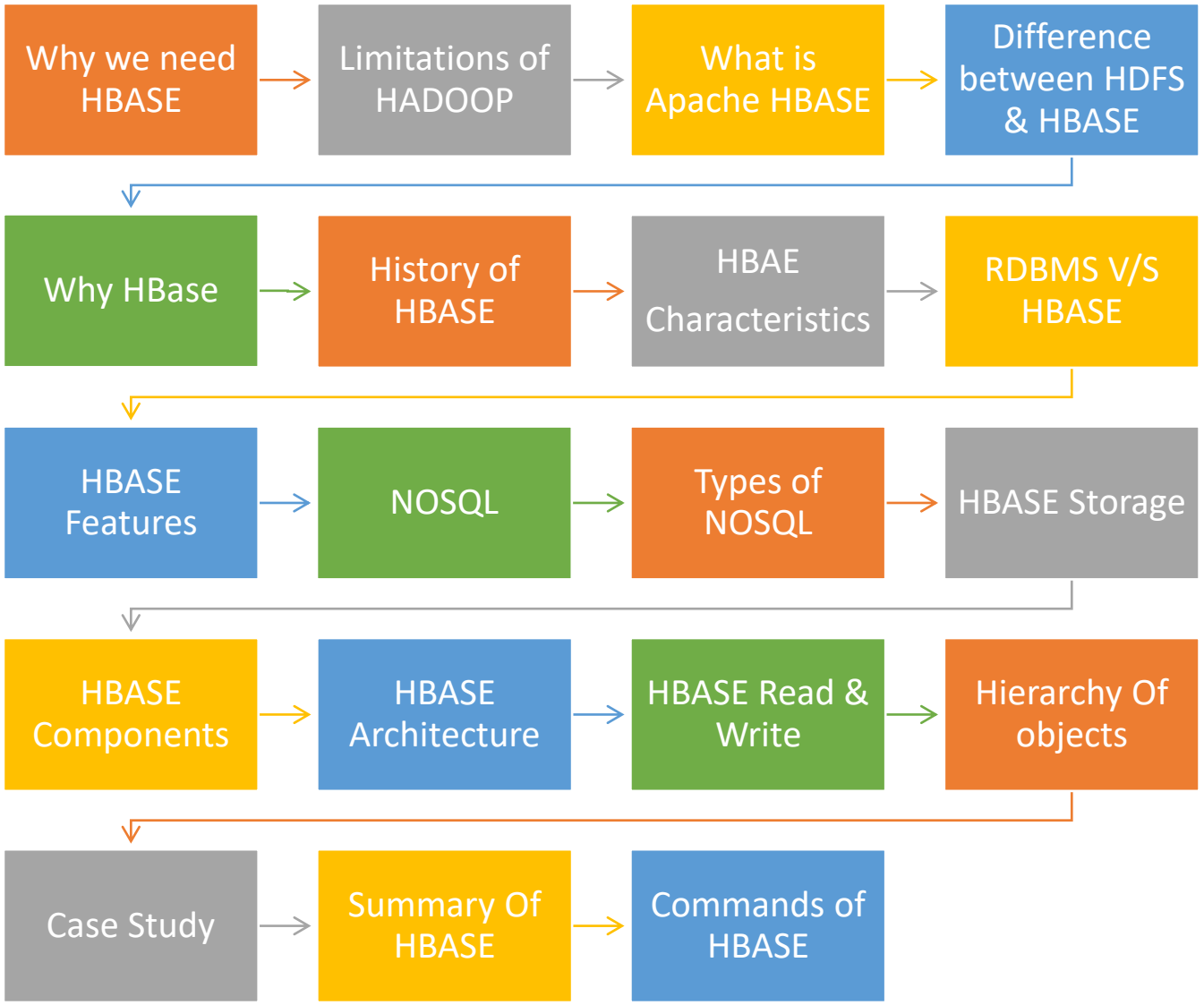
Module 5 : HBASE

AGENDA:

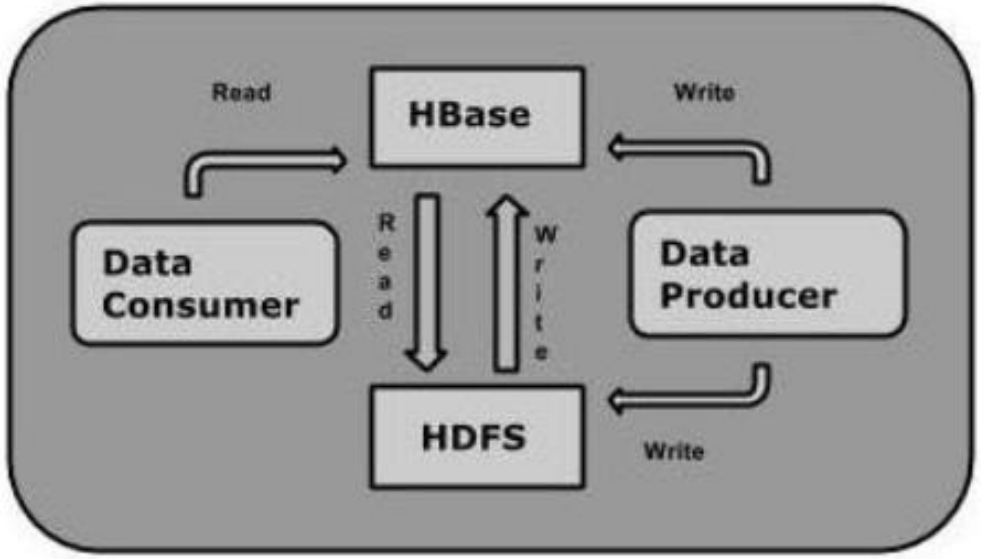

Why we need HBASE
Limitations of HADOOP
What is Apache HBASE
Difference between HDFS & HBASE
Why HBase
History of HBASE
HBAE Characteristics
RDBMS V/S HBASE
HBASE Features
NOSQL
Types of NOSQL
HBASE Storage
HBASE Components
HBASE Architecture
HBASE Read & Write
Hierarchy Of objects
Case Study
Summary Of HBASE
Commands of HBASE

What is 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.
- Provides random real-time read/write access to data in the Hadoop File System.

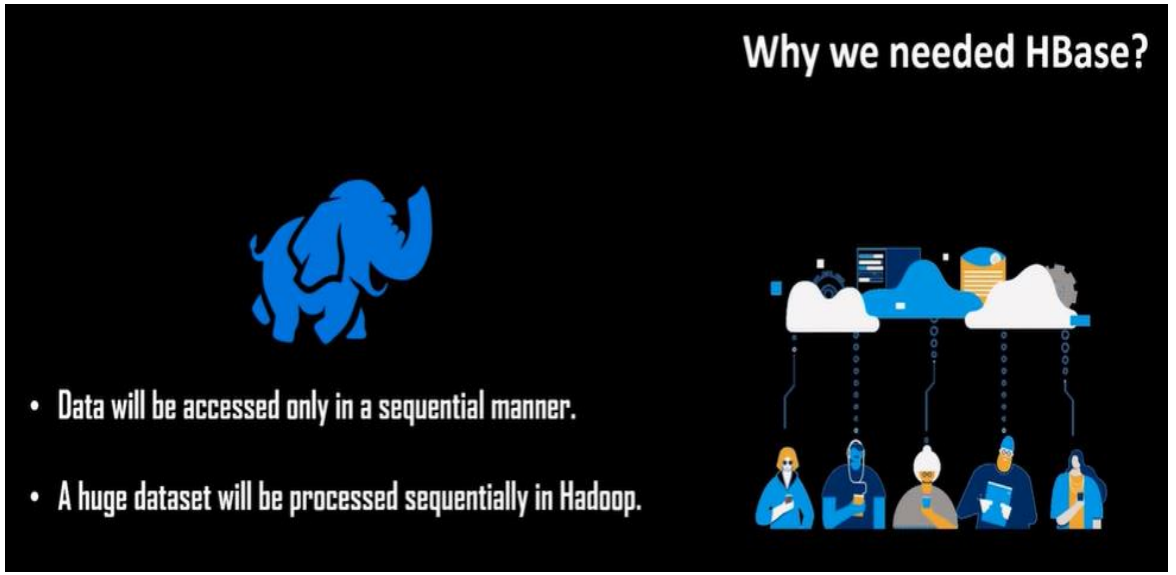
COLUMN FAMILIES

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



Why HBASE?

- RDBMS get exponentially slow as the data becomes large
- Expects data to be highly structured, i.e. ability to fit in a well-defined schema
- Any change in schema might require a downtime: Downtime can disrupt **business operations**, **affect productivity**, and potentially lead to data **inconsistency or loss** if not managed properly.
- For sparse datasets (many columns have NULL values), too much of overhead of maintaining NULL values



- It can store huge amounts of data in a tabular format for extremely fast reads and writes.
- HBase is used in a scenario that requires regular, consistent insertion and overwriting of data.
- **HDFS stores, processes, and manages large amounts of data efficiently. Then, why HBASE???**

Why HBase:

HDFS stores, processes, and manages large amounts of data efficiently.
However, it performs only batch processing and the data will be accessed in a sequential manner.

Therefore, a solution is required to access, read, or write data anytime regardless of its sequence in the clusters of data.

Map Reduce (Hadoop)

Bigtable (Hypertable)

anytime

Google File System (Hadoop)

Limitations of HADOOP

Advantages :

Huge data storage

Data accessed – Sequential manner

Disadvantage/Limitations:

To fetch few records it take long time.

It has scan for entire distributed file system – To fetch small record

Hadoop doesn't provide random access to database

Solution

"HBASE"

APACHE HBASE: Definition

HBASE: Opensource non-relation distributed database written in Java. It is developed as part of "Apache Software Foundation's " Apache Hadoop project and runs on "Top of HDFS".

Its a column-oriented database built on top of HDFS.

Open-source project and "Horizontally Scalable"

HBASE – Data Model – Similar – "Google's Big Data Table".

Designed – To provide quick random access to huge amounts of structured data.

It leverages the "fault tolerance" provided by Hadoop file system and it is part of Hadoop ecosystem

It provides "random real –time read and write access to the data in HDFS

HDFS VS HBASE

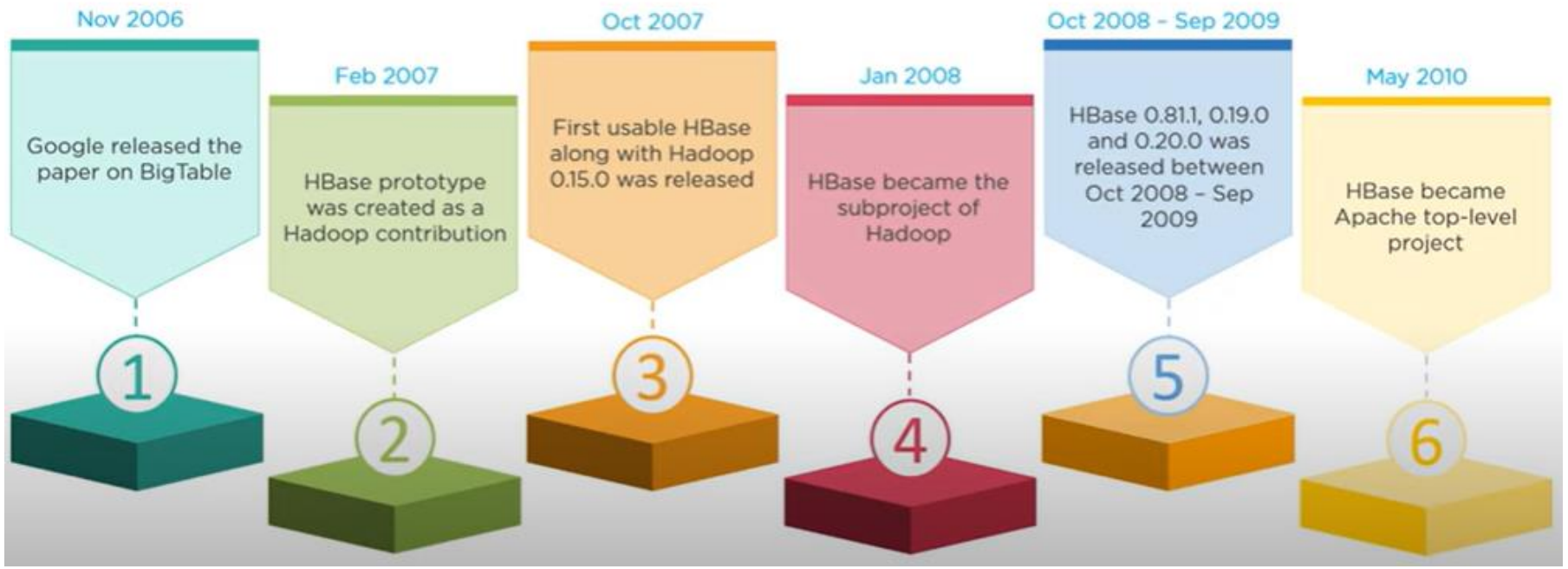
- HDFS is a distributed file system that stores huge files
- HDFS does not Support individual file lookups
- It has High latency
- Only sequential memory access is available

- HBase is built on top of HDFS
- HBase faster and individual file Lookups
- It has Low latency
- It has in-built Hash tables enabling faster lookups.

RDBMS Vs. HBase

HBase	RDBMS
Column-oriented	Row oriented (mostly)
Flexible schema, add columns on the fly	Fixed schema.
Good with sparse tables,	Not optimized for sparse tables.
Joins using MR –not optimized	Optimized for joins.
Tight integration with MR	Not really...
Horizontal scalability –just add hardware	Hard to shard and scale
Good for semi-structured data as well as structured data	Good for structured data

History of HBase



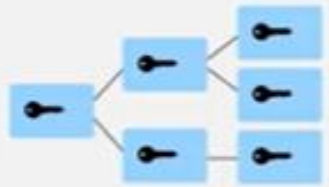
Companies using Hbase:

Characteristics of HBase:

HBase is a type of NoSQL database and is classified as a key-value store.
In HBase:

Value is identified with a key

Key and value are ByteArray



Values are stored in key-orders

Quickly accessed by value keys

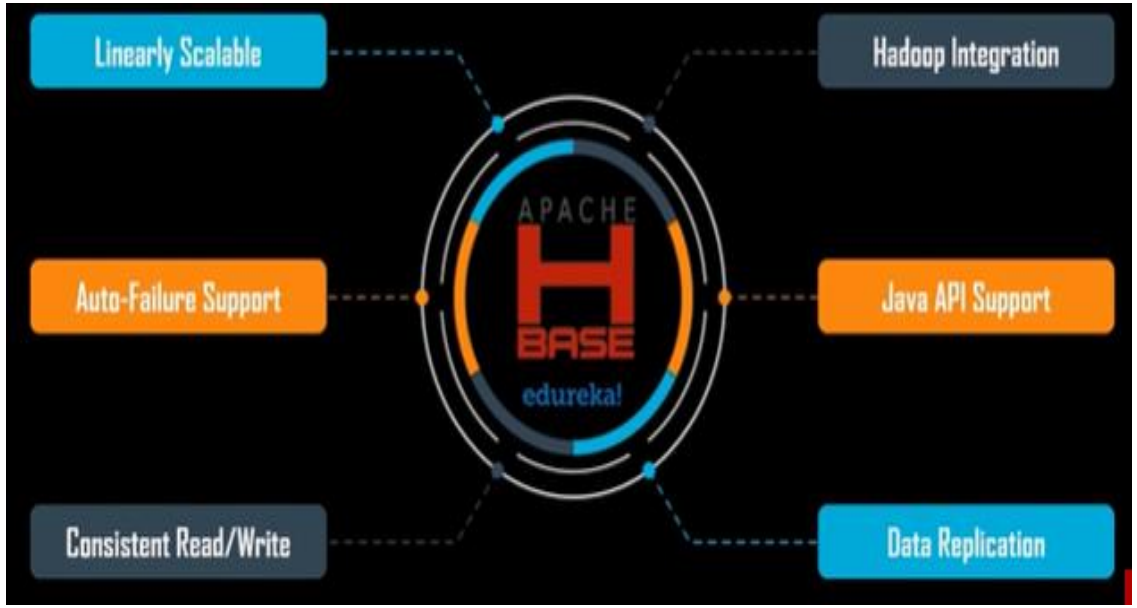
HBase is a database in which tables have no schema. At the time of table creation, column families are defined, not columns.

HBase Features:

Scalable	Automatic failure support	Consistent read and write	JAVA API for client access	Block cache and bloom filters
				
Data can be scaled across various nodes as it is stored in HDFS	Write Ahead Log across clusters which provides automatic support against failure	HBase provides consistent read and write of data	Provides easy to use JAVA API for clients	Supports block cache and bloom filters for high volume query optimization

HBASE Features

- Linear Scalable : HBASE built on top of HDFS.
  - HDFS – Horizontal scalable (Same feature is adopted by HBASE).
- Auto Failure Support: Support for “Fault Tolerance”
- Consistent Read / Write: Random Access of reading & writing data
- HDFS Integration: Integrates with Hadoop and both as a source and destination
- Java API Support: Easy for Java API for client
- Data Replication: Data replication across all the clusters



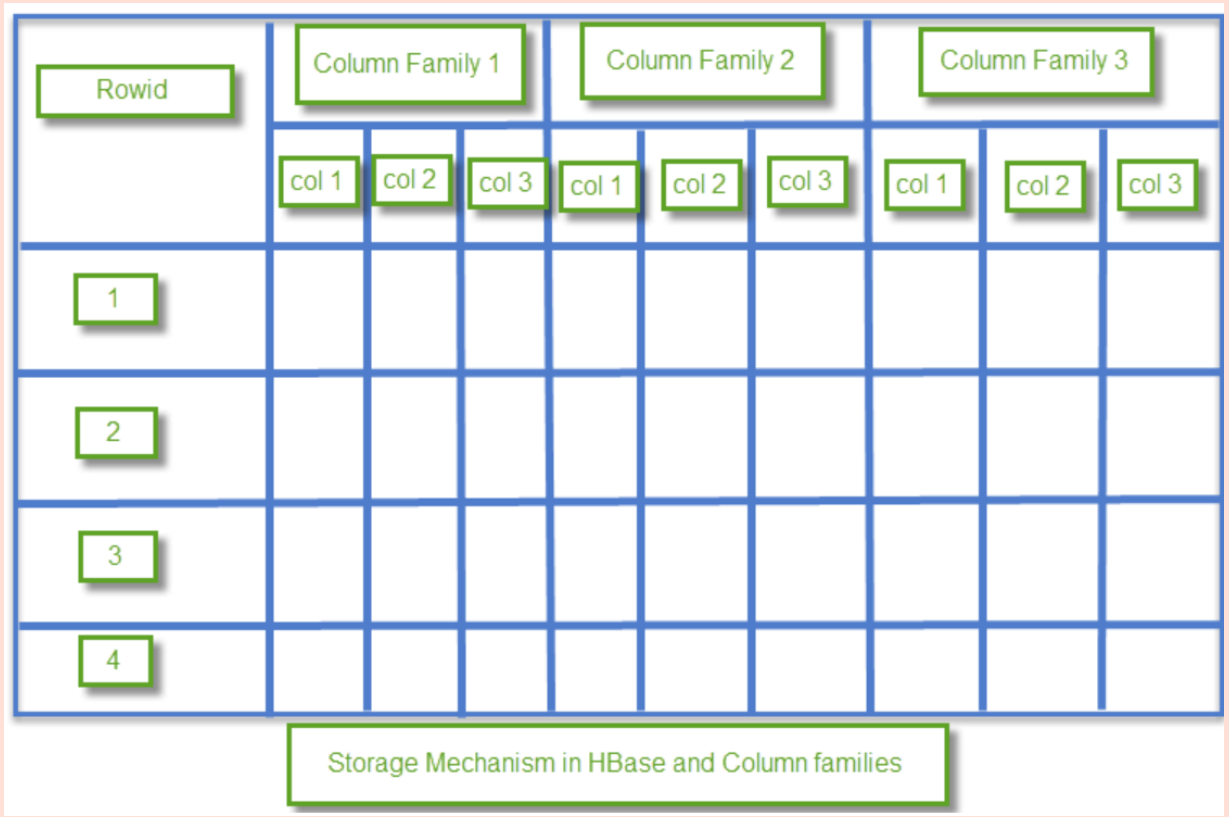
HBASE – Storage Mechanism

Row-ID	Column Family		Column Family	
	Name	Role	Salary	Age
1	Rajesh	Tester	35,000	28

- It is "Column - Oriented" database
- Tables – Sorted by – Row.
- Table Schema – Defines – Only Column families (which are Key –Value Pairs).

- Table – Collection of Rows
- Rows – Collection of Column Families
- Column Families – Collection of Column
- Column – Collection of key value pairs.

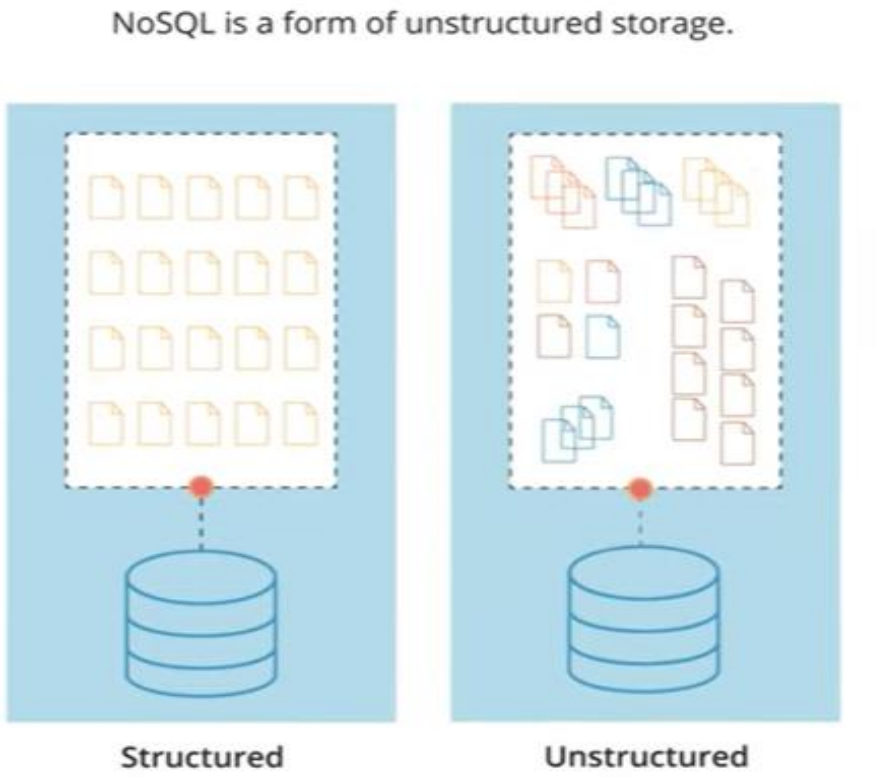
HBASE – Storage Mechanism



- HBase is a column-oriented database and data is stored in tables.
- The tables are sorted by RowId. As shown, HBase has RowId, which is the collection of several column families that are present in the table.
- The column families that are present in the schema are key-value pairs.
- If we observe in detail each column family having multiple numbers of columns.
- The column values stored into disk memory.
- Each cell of the table has its own Metadata like timestamp and other information.

NoSQL:

- NoSQL databases are databases that **store data in a format other than relational tables**.
- **Types of NoSQL** databases include pure **document databases, key-value stores, wide-column databases, and graph databases.**



Types of NoSQL:

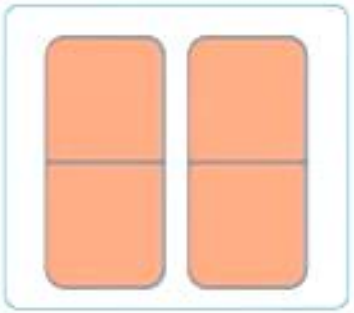
Key Value

Examples:
Oracle NoSQL, Redis server, Scalaris

Document-based

Examples:
MongoDB, CouchDB, OrientDB, RavenDB

Column-based



Examples:
BigTable, Cassandra, HBase, Hypertable

Graph-based

Examples:
Neo4J, InfoGrid, Infinite Graph, FlockDB

Types of NoSQL:

Key value :

- Every data element in the database is stored as a **key value pair** consisting of an attribute name (or "key") and a value. In a sense, a key-value store is **like a relational database** with **only two columns**: the key or attribute name and the value.

- The data can be **retrieved** by using a unique key allotted to each element in the database. The values can be simple data types like strings and numbers or complex objects.

- key-value store is more **flexible** because each **user's data can have different attributes** without requiring changes to the database schema.

Example

Key: "user:1"
Value:{
"name": "Alice",
"email": "alice@email.com",
"age": 30
}

Types of NoSQL

document database:

- It stores data in JSON, BSON, or XML documents (not Word documents or Google Docs, of course). In a document database, documents can be nested. Particular elements can be indexed for faster querying.

- Documents can be **stored and retrieved in a form that is much closer to the data objects** used in applications which means **less translation is required to use these data in the applications**.

- In the Document database, the **particular elements can be accessed by using the index value** that is assigned for faster querying.
- Example

```
{"employees":[
    {"name":"Shyam", "email":"shyamjaiswal@gmail.com"},
    {"name":"Bob", "email":"bob32@gmail.com"},
    {"name":"Jai", "email":"jai87@gmail.com"}
]}
```

Types of NoSQL

column-oriented database

- **non-relational database** that **stores the data in columns** instead of rows. That means when we want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data.

- Columnar databases are designed to read data more efficiently and retrieve the data with greater speed. A columnar database is used to store a large amount of data. Key features of columnar oriented database:
  - Scalability.
  - Compression.
  - Very responsive.

Types of NoSQL

A graph database

- focuses on the relationship between data elements.
- Each element is stored as a node (such as a person in a social media graph). The connections between elements are called links or relationships.
- In a graph database, connections are first-class elements of the database, stored directly.

Key features of graph database:

- In a graph-based database, it is easy to identify the relationship between the data by using the links.
- The Query's output is real-time results.
- The speed depends upon the number of relationships among the database elements.

HBASE :Major Components

- Client library:
  - **Connecting to the HBase Cluster**
  - **Table Operations**
  - **Data Manipulation**
  - **Batch Operations**
  - **Scanning**
  - **Filtering**
- Master Server
- Region Server

1.Master Server:

- Assigns regions to the region servers and takes the help of Apache Zookeeper.
- Handles load balancing of the regions across region servers.
- Maintains the state of the cluster by negotiating the load balancing.

Primary responsibilities of the master server:

1.Cluster Coordination:

1. The master server acts as the central coordinator for the HBase cluster. It monitors the health and status of all the RegionServers in the cluster, ensuring that they are functioning correctly.

2.Assignment of Regions:

1. One of the key responsibilities of the master server is to assign regions of HBase tables to different RegionServers in the cluster. It determines the distribution of regions across RegionServers based on factors such as load balancing, data locality, and cluster capacity.

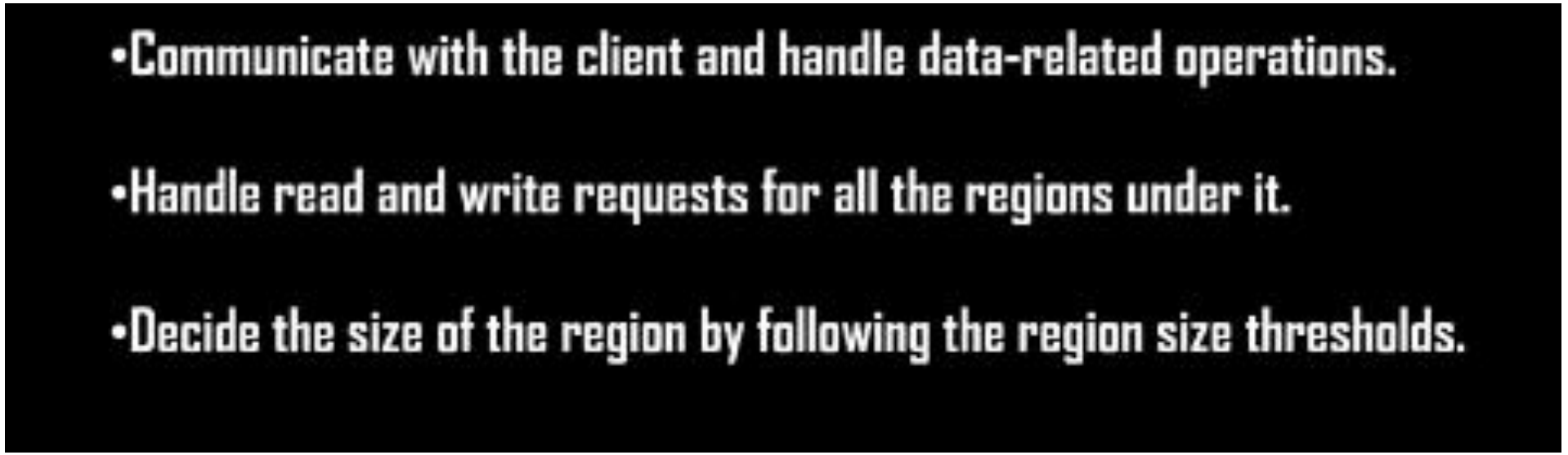
3.Table Management:

1. The master server is responsible for managing HBase tables and their schemas. It handles tasks such as **creating, deleting, enabling, and disabling tables**. When a new table is created, the master server coordinates the initial assignment of regions to RegionServers.

Master Server:

- **Metadata Management**:
  - The master server maintains important **metadata about HBase tables, regions, and RegionServers**. This metadata includes information about table schemas, region locations, and server assignments. Clients use this metadata to locate and access data in the cluster efficiently.
- **Cluster Operations**:
  - The master server provides APIs and interfaces for performing administrative tasks and cluster management operations. This includes tasks such as starting and stopping RegionServers, compacting and splitting regions, and monitoring cluster health and performance.
- **Schema Changes**:
  - If schema modifications are requested (such as adding or removing columns from a table), the master server coordinates these changes across the cluster. It ensures that schema changes are propagated to all relevant RegionServers and that they are applied correctly.
- **Failover Handling**:
  - In the event of a RegionServer failure, the master server detects the failure and coordinates the reassignment of regions hosted by the failed RegionServer to other healthy RegionServers. This ensures continuous availability and fault tolerance of HBase tables.
- **Load Balancing**:
  - The master server is responsible for load balancing across RegionServers to ensure even distribution of data and workload. It monitors the resource usage and data distribution across the cluster and may trigger region reassignments to achieve better load balancing.

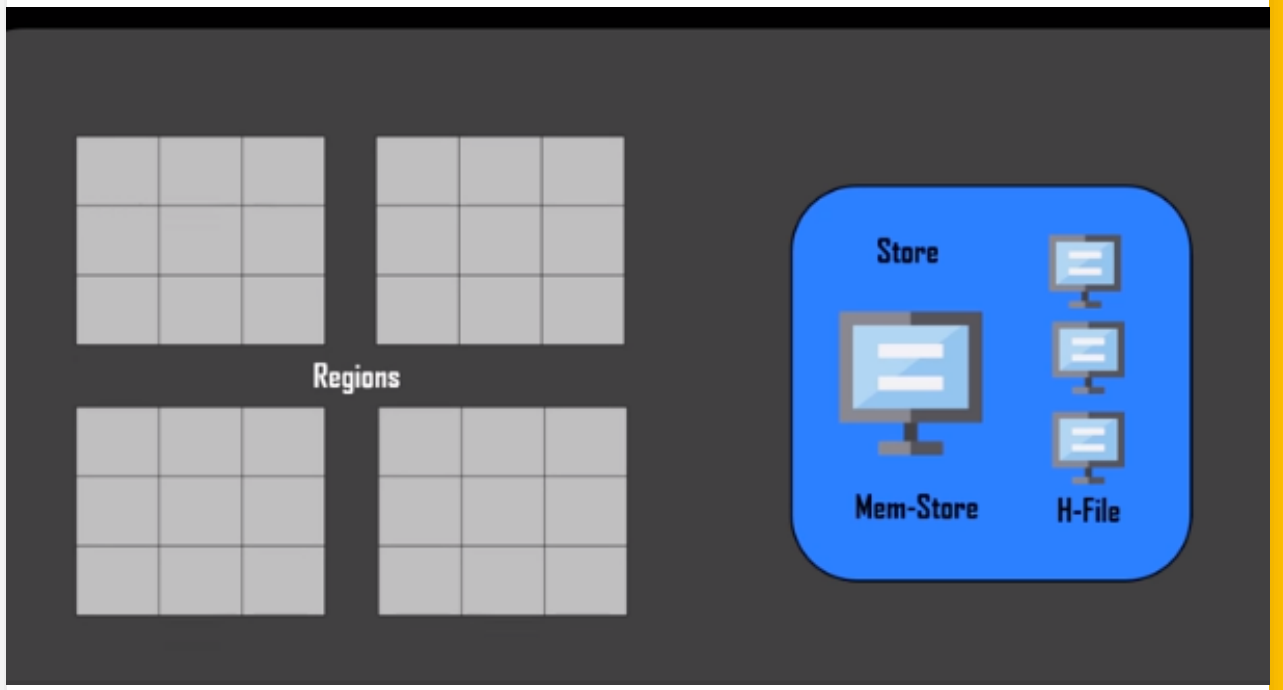
2. Region :



- Regions – Nothing but “Tables”
- Tables – split up across the “region servers”.

3. Region Server

- Store - Contains “memory file” & “H-File” .
- Mem- store – Just like cache memory.
- Anything that is entered in “HBASE” is automatically stored initially.
- Later the data is transferred and saved in H-Files as “blocks”.
- Later the mem-store is erased out .
- Region Management
- Data Storage and Retrieval



Zookeeper

- Zookeeper is an open-source project that provides services like maintaining configuration.
- Zookeeper has ephemeral nodes representing different region servers.
- Clients communicate with region servers via zookeeper.

- Zookeeper: Providing Distributed synchronization, Naming etc
- It has ephemeral node: Master servers use these nodes to **discover available nodes.**
- Based on availability: Nodes are also used to track server failure or network partitions.
- In sudo / stand alone mode: HBASE itself take care of Zookeeper

HBase Architecture:

HBase has two types of Nodes—Master and RegionServer. Following are the characteristics of the two nodes.

Master

- Only one Master node runs at a time. Its high availability is maintained with ZooKeeper.
- It manages cluster operations like assignment, load balancing, and splitting.
- It is not a part of the read/write path.

RegionServer

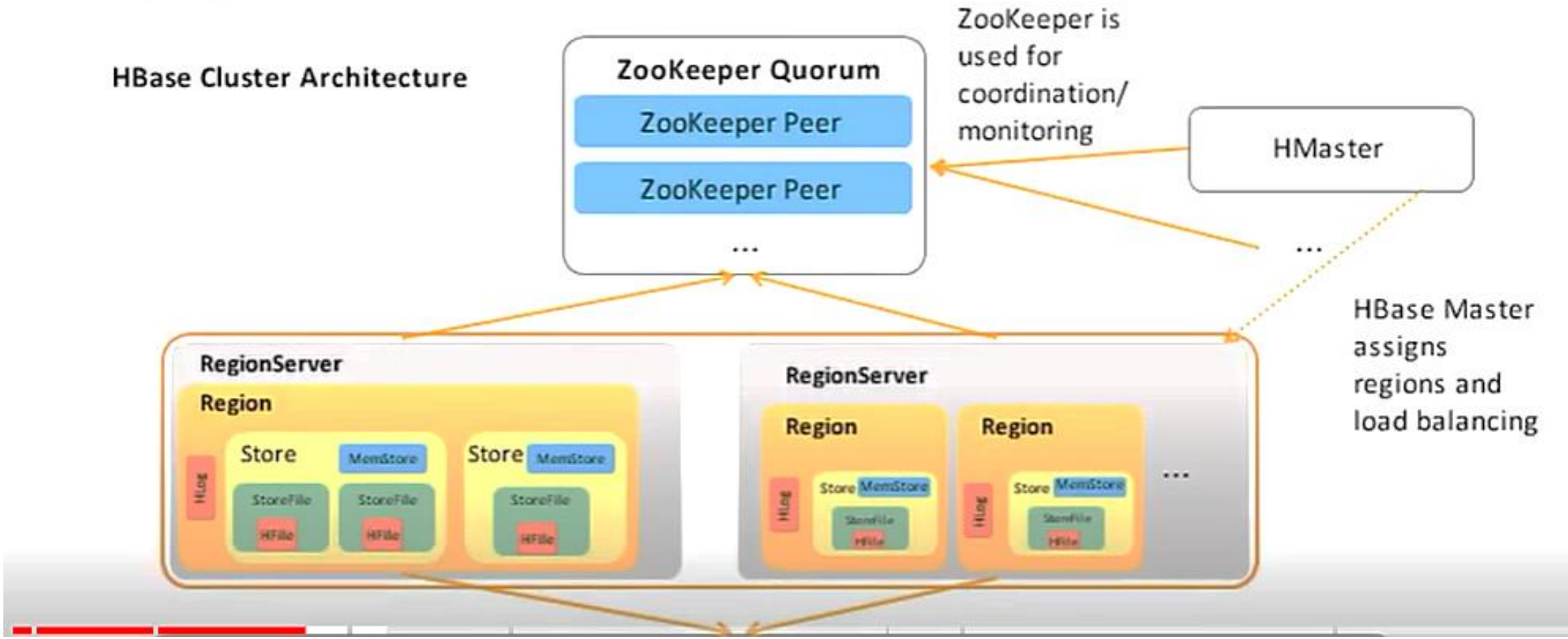
- One or more RegionServers can exist at a time.
- It hosts tables, performs reads, and buffers writes.
- Clients communicate with RegionServers for read/write operation.

HBASE - ARCHIZTECTURE

- Inside HBASE: **Tables are split into “Regions”.**
- Regions – Served by – Server Regions.
- Regions Servers– Vertically Divided by “column families” into stores.
- Stores – are saved as files in HDFS.
- HBASE – 3 components
- Client library
- Master Server
- Region Server

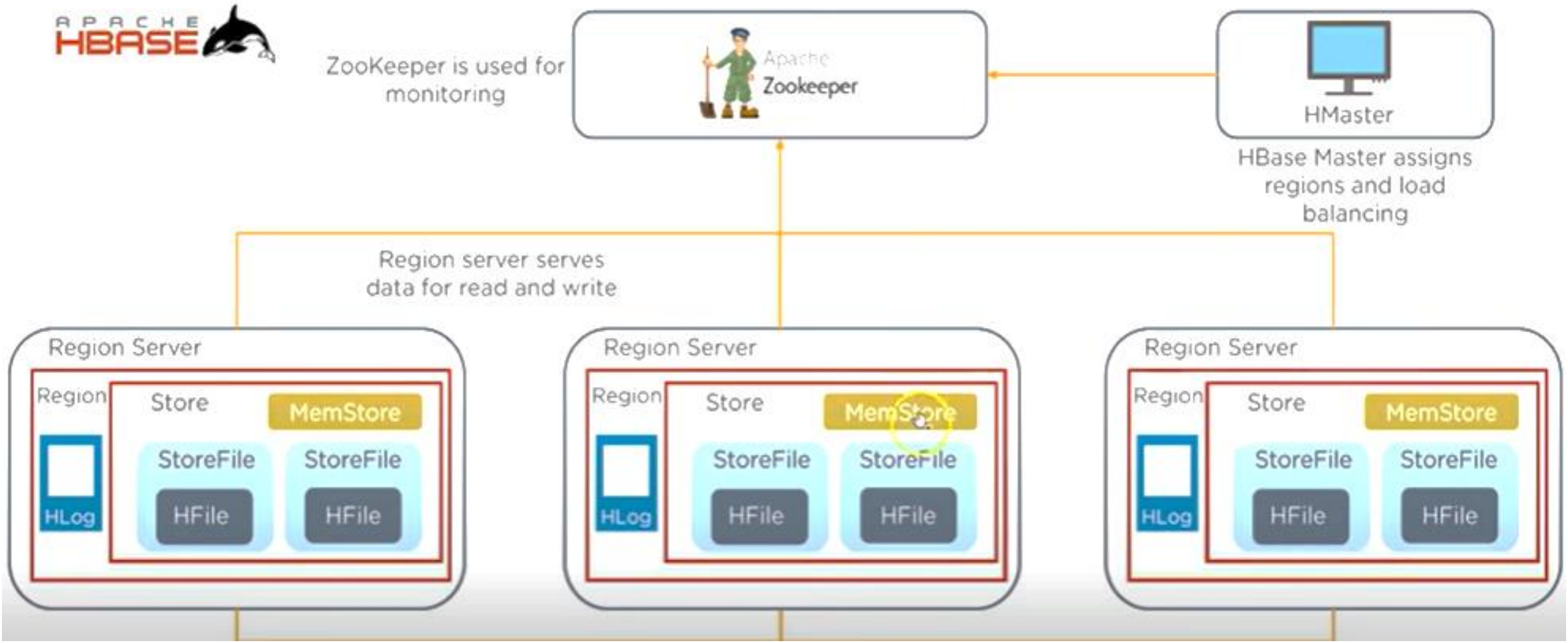
HBase Architecture:

The image represents the components of HBase—HBase Master and RegionServers.



HDFS

HBase Architecture:



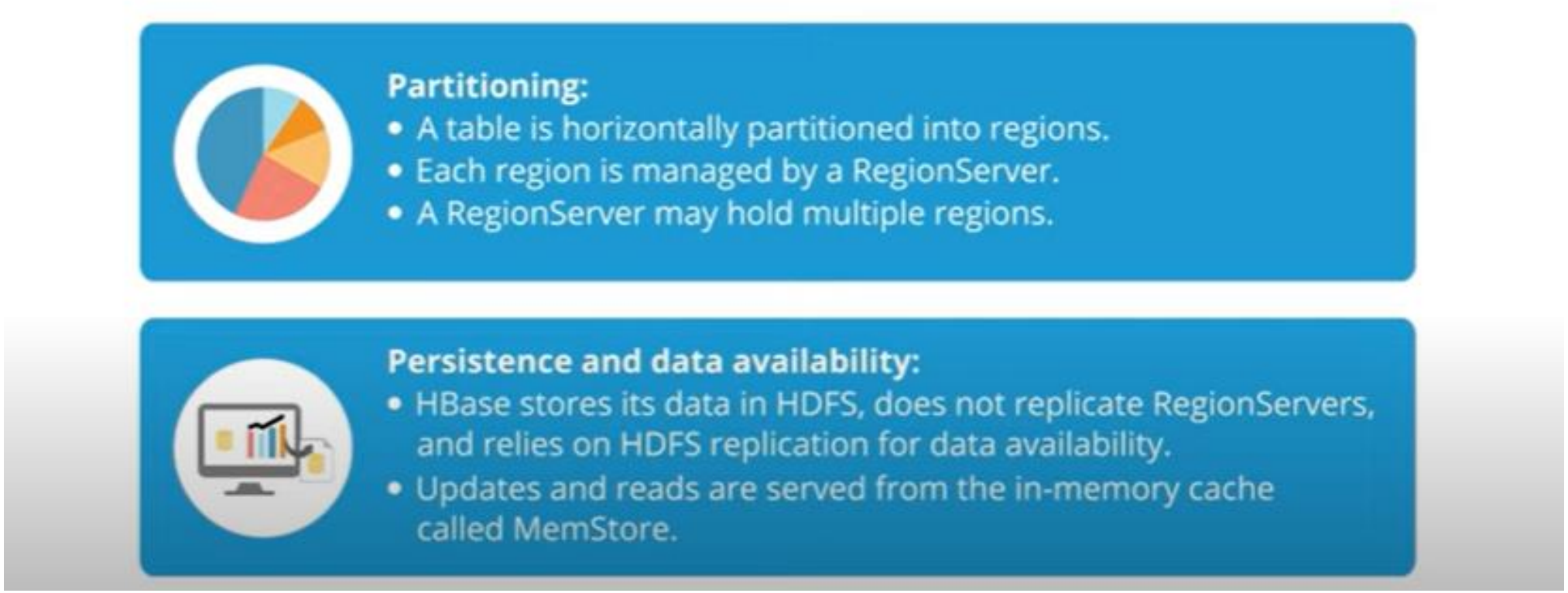
HDFS

Column – Oriented v/s Row Oriented

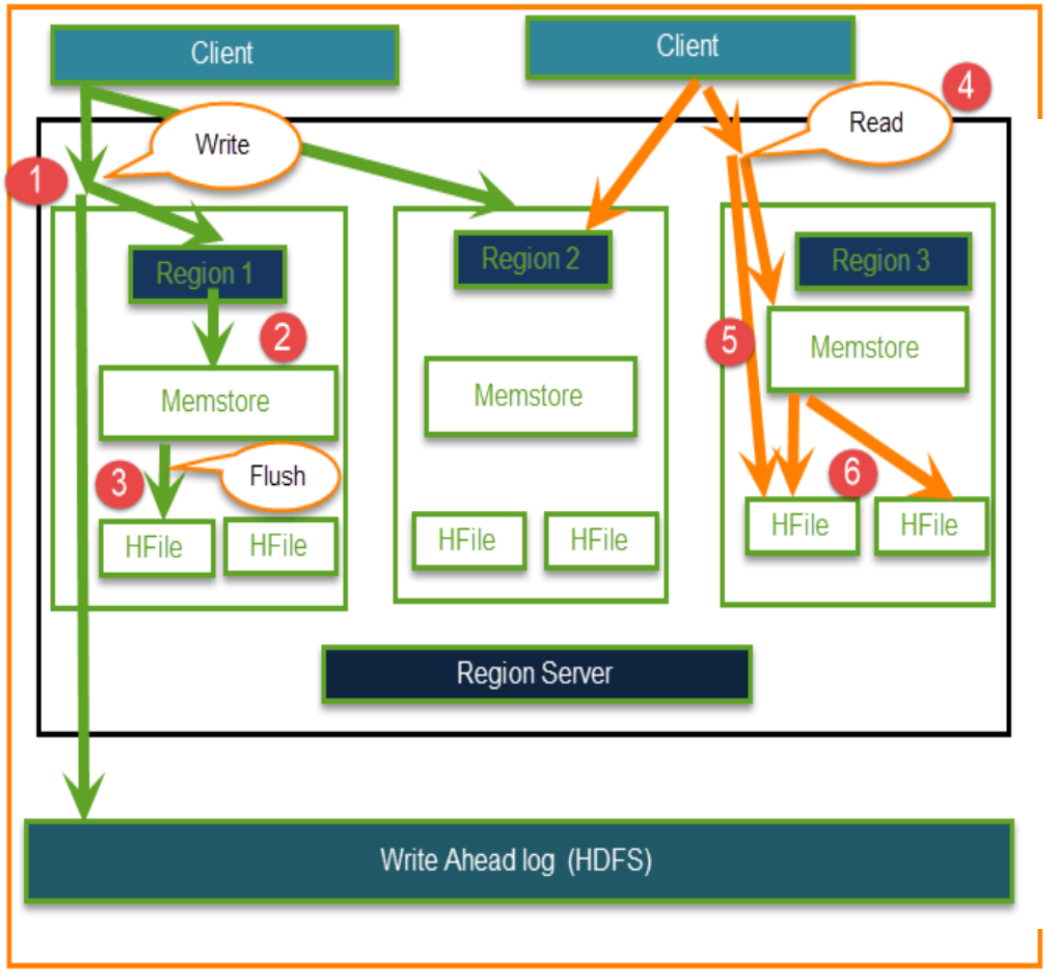
Column-oriented Database	Row oriented Database
• When the situation comes to process and analytics we use this approach. Such as **Online Analytical Processing** and it's applications.	• **Online Transactional process** such as banking and finance domains use this approach.
• The amount of data that can able to store in this model is very huge like in terms of petabytes	• It is designed for a small number of rows and columns.

Storage Model of HBase:

The two major components of the storage model are as follows:



HDFS – Read & Write Data



Step 1) Client wants to write data and in turn first communicates with Regions server and then regions

Step 2) Regions contacting memstore for storing associated with the column family

Step 3) First data stores into Memstore, where the data is sorted and after that, it flushes into HFile. The main reason for using Memstore is to store data in a Distributed file system based on Row Key. Memstore will be placed in Region server main memory while HFiles are written into HDFS.

Step 4) Client wants to read data from Regions

Step 5) In turn Client can have direct access to Mem store, and it can request for data.

Step 6) Client approaches HFiles to get the data. The data are fetched and retrieved by the Client.

HBase Architecture: HBase Write Mechanism

Step 1: Whenever the client has a write request, the client writes the data to the WAL (Write Ahead Log).

- The edits are then appended at the end of the WAL file.
- This WAL file is maintained in every Region Server and Region Server uses it to recover data which is not committed to the disk.

Step 2: Once data is written to the WAL, then it is copied to the MemStore.

Step 3: Once the data is placed in MemStore, then the client receives the acknowledgment.

Step 4: When the MemStore reaches the threshold, it dumps or commits the data into a HFile.

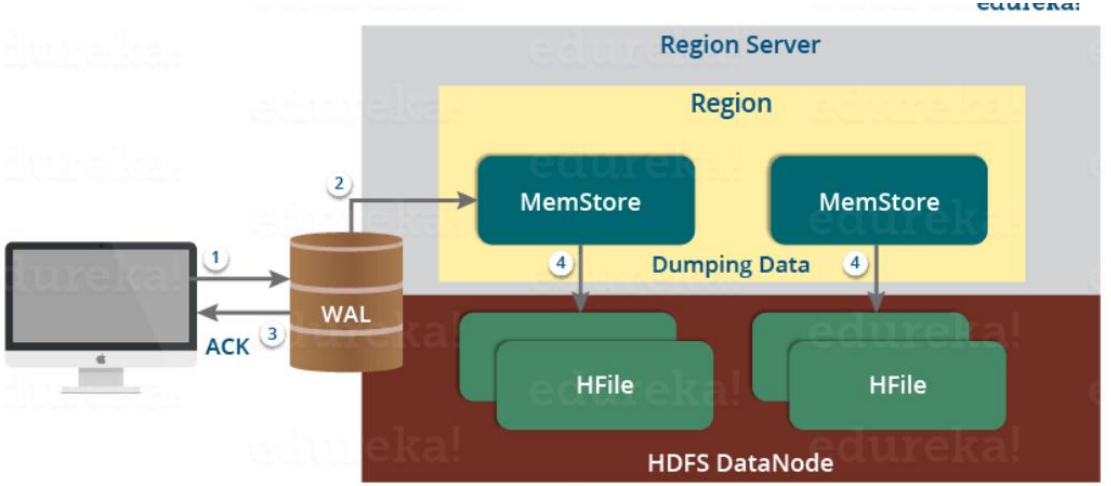


Figure: Write Mechanism in HBase

HBase Write Mechanism- MemStore

•The MemStore always updates the data stored in it, in a lexicographical order (sequentially in a dictionary manner) as sorted KeyValues. There is one MemStore for each column family, and thus the **updates are stored in a sorted manner for each column family.**

•When the MemStore reaches the threshold, it dumps all the data into a new HFile in a sorted manner. This HFile is stored in HDFS. HBase contains multiple HFiles for each Column Family.

•Over time, the number of **HFile grows as MemStore dumps the data**.

•**MemStore also saves the last written sequence number**, so Master Server and MemStore both knows, that **what is committed so far and where to start from**. When region starts up, the last sequence number is read, and from that number, new edits start.

HBase Architecture: HBase Write Mechanism- Hfile

•The writes are placed sequentially on the disk. Therefore, the movement of the disk's read-write head is very less. This makes write and search mechanism very fast.

•The HFile indexes are loaded in memory whenever an HFile is opened. This helps in finding a record in a single seek.

•The **trailer is a pointer which points to the HFile's meta block** . It is written at the end of the committed file. It contains information about timestamp and bloom filters.

•Bloom Filter helps in searching key value pairs, it skips the file which does not contain the required rowkey. Timestamp also helps in searching a **version of the file**, it helps in skipping the data.

Hierarchy of Objects:

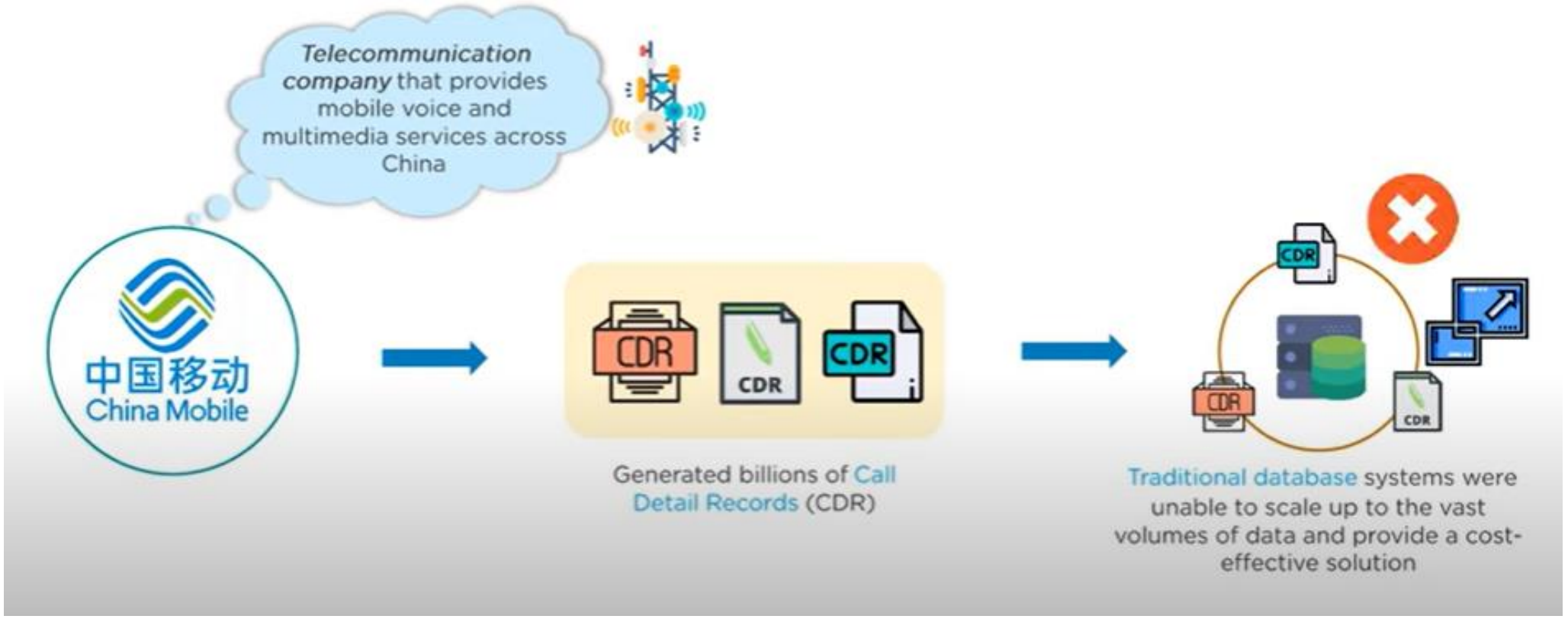
Table	HBase table present in the HBase cluster
Region	HRegions for the presented tables
Store	It stores per ColumnFamily for each region for the table
Memstore	• Memstore for each store for each region for the table • It sorts data before flushing into HFiles • Write and read performance will increase because of sorting
StoreFile	StoreFiles for each store for each region for the table
Block	Blocks present inside StoreFiles

- Memstore holds in-memory modifications to the store.
- The hierarchy of objects in HBase Regions is as shown from top to bottom in below table.

Case Study : Telecom

Problem Statement	Solution
Telecom Industry faces following Technical challenges • Storing billions of CDR (Call detailed recording) log records generated by telecom domain • Providing real-time access to CDR logs and billing information of customers • Provide cost-effective solution comparing to traditional database systems	HBase is used to store billions of rows of detailed call records. If 20TB of data is added per month to the existing RDBMS database, performance will deteriorate. To handle a large amount of data in this use case, HBase is the best solution. HBase performs fast querying and displays records.

HBase Use Case:



Case Study : Banking

> The **Banking industry** generates millions of records on a daily basis. In addition to this, the banking industry also needs an analytics solution that can detect Fraud in money transactions
>
> To store, process and update vast volumes of data and performing analytics, an ideal solution is – HBase integrated with several Hadoop ecosystem components.

When to Use Hbase:

- Whenever there is a need to write heavy applications.
- Performing online log analytics and to generate compliance reports.

HBase Commands

•**Create:** Creates a new table identified by 'table1' and Column Family identified by 'colf'.

•**Put:** Inserts a new record into the table with row identified by 'row..'

•**Scan:** returns the data stored in table\

•**Get:** Returns the records matching the row identifier provided in the table

•**Help:** Get a list of commands

```
create 'table1', 'colf'

list 'table1'

put 'table1', 'row1', 'colf:a', 'value1'

put 'table1', 'row1', 'colf:b', 'value2'

put 'table1', 'row2', 'colf:a', 'value3'

scan 'table1'

get 'table1', 'row1'
```

Summary – HBASE:

- HBase architecture components: HMaster, HRegion Server, HRegions, ZooKeeper, HDFS
- HMaster in HBase is the implementation of a Master server in HBase architecture.
- 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
- HRegions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families.
- HBase Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization.
- HDFS provides a high degree of fault–tolerance and runs on cheap commodity hardware.
- HBase Data Model is a set of components that consists of Tables, Rows, Column families, Cells, Columns, and Versions.
- Column and Row-oriented storages differ in their storage mechanism.

Commands: HBASE

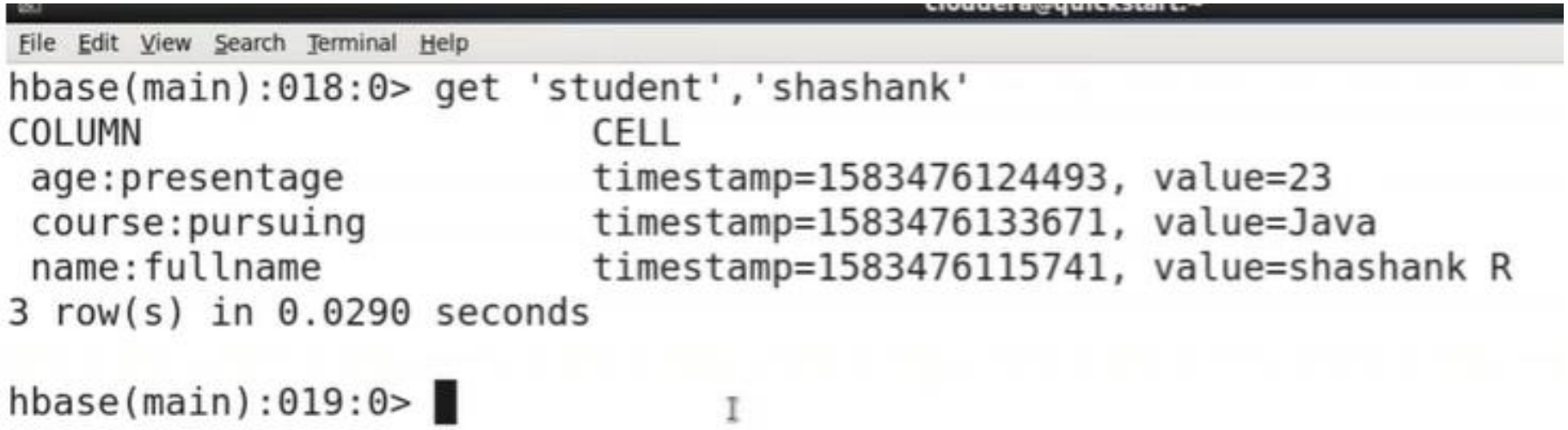
```
File  Edit  View  Search  Terminal  Help
hbase(main):005:0> create 'employee', 'Designation','Salary','Department'
0 row(s) in 1.4460 seconds

=> Hbase::Table - employee
hbase(main):006:0> list
TABLE
employee
1 row(s) in 0.0300 seconds

=> ["employee"]
hbase(main):007:0> disable 'employee'
0 row(s) in 2.4640 seconds

hbase(main):008:0> scan 'employee'
```

- disable ‘table’
- drop ‘table’

Adding Data – Table

```
File  Edit  View  Search  Terminal  Help
hbase(main):011:0> create 'student', 'name', 'age', 'course'
0 row(s) in 1.2410 seconds

=> Hbase::Table - student
hbase(main):012:0> put 'student','sharath', 'name:fullname', 'sharath kumar'
0 row(s) in 0.2560 seconds

hbase(main):013:0> put 'student','sharath', 'age:presentage', '24'
0 row(s) in 0.0130 seconds

hbase(main):014:0> put 'student','sharath', 'course:pursuing', 'Hadoop'
0 row(s) in 0.0170 seconds

hbase(main):015:0> put 'student','shashank', 'name:fullname', 'shashank R'
0 row(s) in 0.0140 seconds

hbase(main):016:0> put 'student','shashank', 'age:presentage', '23'
0 row(s) in 0.0150 seconds

hbase(main):017:0> put 'student','shashank', 'course:pursuing', 'Java'
```

Entire Record / Specific Record




```
hbase(main):020:0> get 'student','sharath','course'
COLUMN                          CELL
 course:pursuing                timestamp=1583476107762, value=Hadoop
1 row(s) in 0.0540 seconds

hbase(main):021:0> get 'student','sharath','name'
COLUMN                          CELL
 name:fullname                  timestamp=1583476045340, value=sharath kumar
1 row(s) in 0.0210 seconds
```

File Edit View Search Terminal Help

```
hbase(main):022:0> scan 'student'
ROW                           COLUMN+CELL
 sharath                      column=age:presentage, timestamp=1583476082211, value=24
 sharath                      column=course:pursuing, timestamp=1583476107762, value=Hadoop
 sharath                      column=name:fullname, timestamp=1583476045340, value=sharath kumar
 shashank                     column=age:presentage, timestamp=1583476124493, value=23
 shashank                     column=course:pursuing, timestamp=1583476133671, value=Java
 shashank                     column=name:fullname, timestamp=1583476115741, value=shashank R
2 row(s) in 0.0840 seconds
```

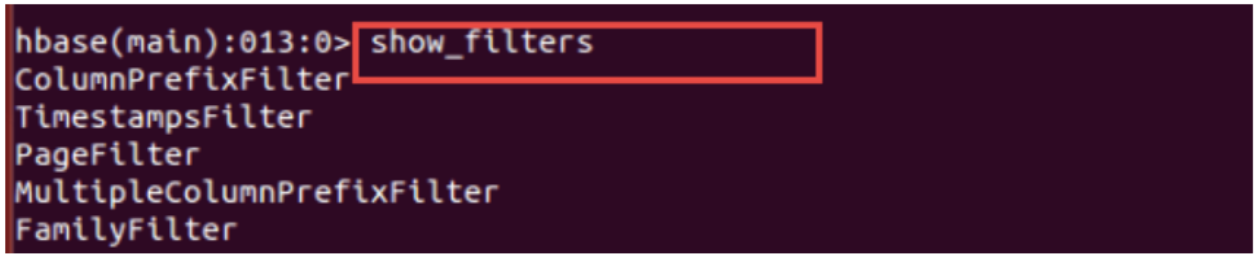
```
hbase(main):016:0> scan 'guru99'
ROW                                                   COLUMN+CELL
 r1                                                   column=c1:, timestamp=30, value=value
 r2                                                   column=c1:, timestamp=15, value=value
 r3                                                   column=c1:, timestamp=15, value=value
3 row(s) in 0.0340 seconds
```

scanning 'guru99' table records

```
hbase(main):023:0> count 'student'
2 row(s) in 0.0320 seconds

=> 2
hbase(main):024:0> alter 'student',NAME=>'age',VERSIONS=>5
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 3.1190 seconds
```

show_filters

```
Syntax: show_filters
```



This command displays all the filters present in HBase like ColumnPrefix Filter, TimestampsFilter, PageFilter, FamilyFilter, etc.

- Filtering – Reading data from HBASE using get / scan operations.
- Return subset of results to clients.
- Inorder to use these filters – import "java" classes to Hbase shell


THANK
YOU.