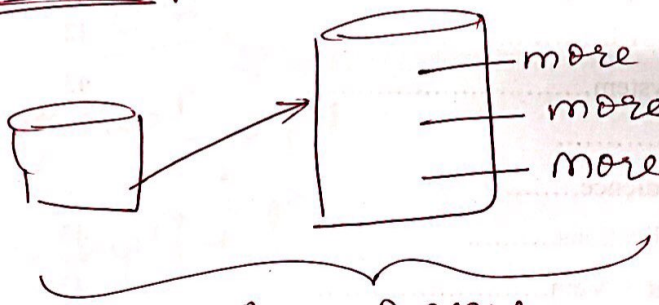
* NoSQL Database

→ Non-relational DMS., that does not require a fixed schema.
→ avoids joins & easy to scale.
→ NoSQL is used for Big Data & real time web Applications
→ "Not only SQL" or "Not SQL"

* Why NoSQL ?

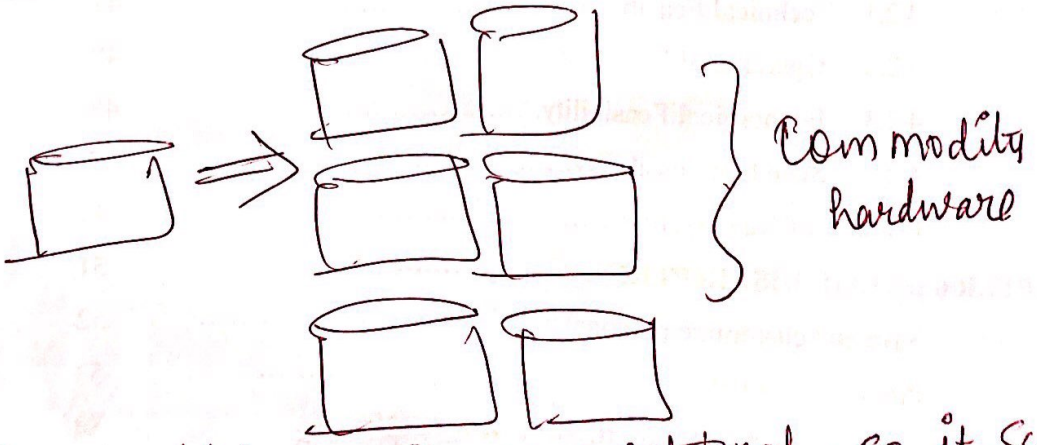
→ The system response time becomes slow when you use RDBMS for massive volumes of data

→ Scale-up - With RDBMS (vertical scaling)



→ expensive process.

→ Scale-out - NoSQL - (horizontal scaling)



→ NoSQL database is non-relational, so it scales out better than relational Databases.

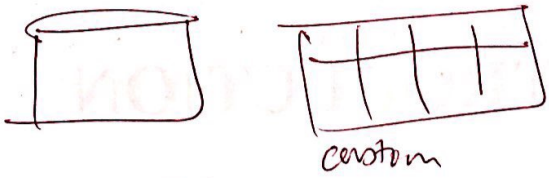
* Features of NoSQL:

① Non-relational.

② Schema-free – or relaxed schema
Implicit schema

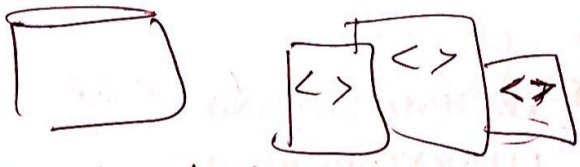
RDBMS:

select Name, age from customer
(explicit schema)



MoSQL DB:

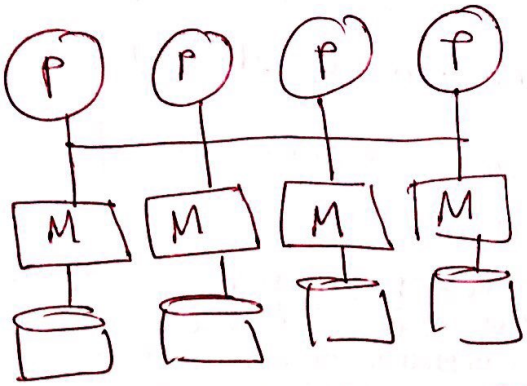
Item [price]
Item [DBcount]



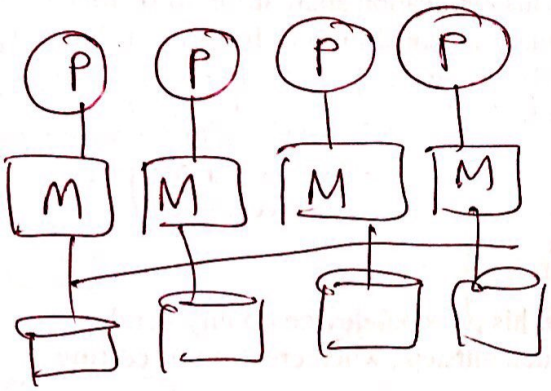
③ Simple API

④ Distributed
- auto scaling & fail-over capabilities
- often ACID concept can be sacrificed for scalability & throughput.
- Mostly no syncronous replication beth distributed nodes.
- only provide eventual consistency.
- shared nothing Architecture.
enables less coordination & higher distribution.

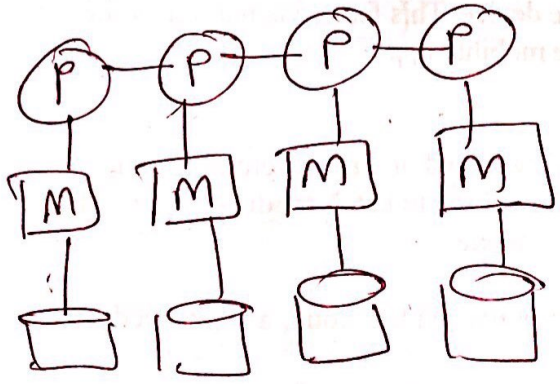
Oracle 11g - shared memory



Oracle RAC - shared ~~RAC~~ Disk



Nosql - shared nothing



* Types of NOSQL Database:

1. Key-value
2. column-Oriented
3. Graph based
4. Document orient

1. Key-value:

→ Data is stored in key-value pair.
→ It is designed in a such a way to handle lots of data & heavy load.
→ Store data as hash table where each key is unique, and the value can be JSON, BLOB, string etc.
→ Website → Domain name

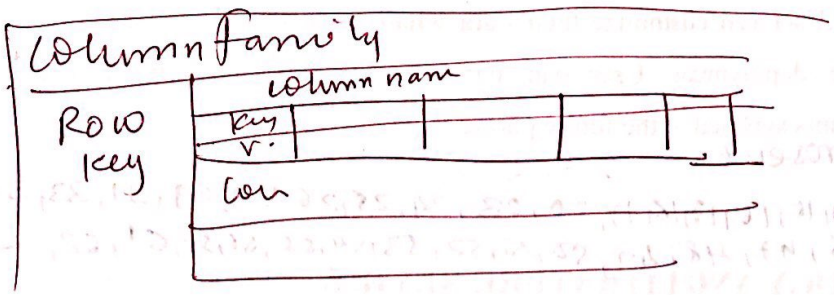
Key	value
Name	
Age	
comp-	

Key → value
Cust1237 → xyz
Cust 1237 → { name : "xyz",
street : " ",
city : " ",
state : " " }

→ used as collection, dictionaries, arrays etc.
→ Best for → Shopping Cart
→ ex: Redis
Dynamo
Riak.
→

(2) Column-Based

→ Based on Big Table paper by google.
→ every column is treated separately
→ Values of single column databases are stored contiguously.




```
[
  {
    "message":
    "timestamp":
    "SenderId":
    "seen":
  },
  {
  },
]
```

Row Based

```
{
"messages":
["Hi --", "--",
 "      "],
"timestamp"
```

→ ~~wildly~~ deliver high performance on aggregation queries like sum, count etc. as data is readily available in column
→ ~~Ap~~ Best for - BI, crm, etc.
→ ex. HBase, Cassandra,

③ Document Oriented:

→ Stores & retrieve data as a key-value pair but value part is stored as document.

→ document is stored in JSON or XML format

JSON object → document

→ Best for - cms system, blogging platform, e-commerce.

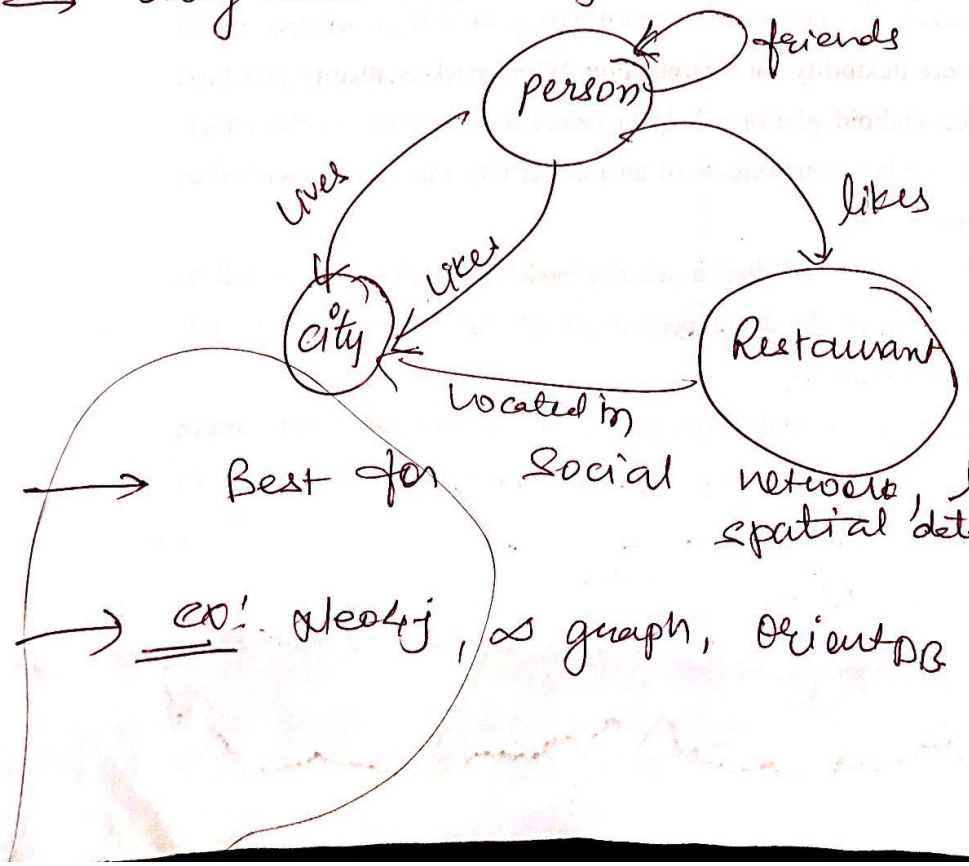
→ ex. couch DB, MongoDB, Riak, lotus notes, Amazon SimpliDB

④ Graph Based

→ A graph type database stores entities as well the relations amongst those entities.

→ entity → node
relation → edge

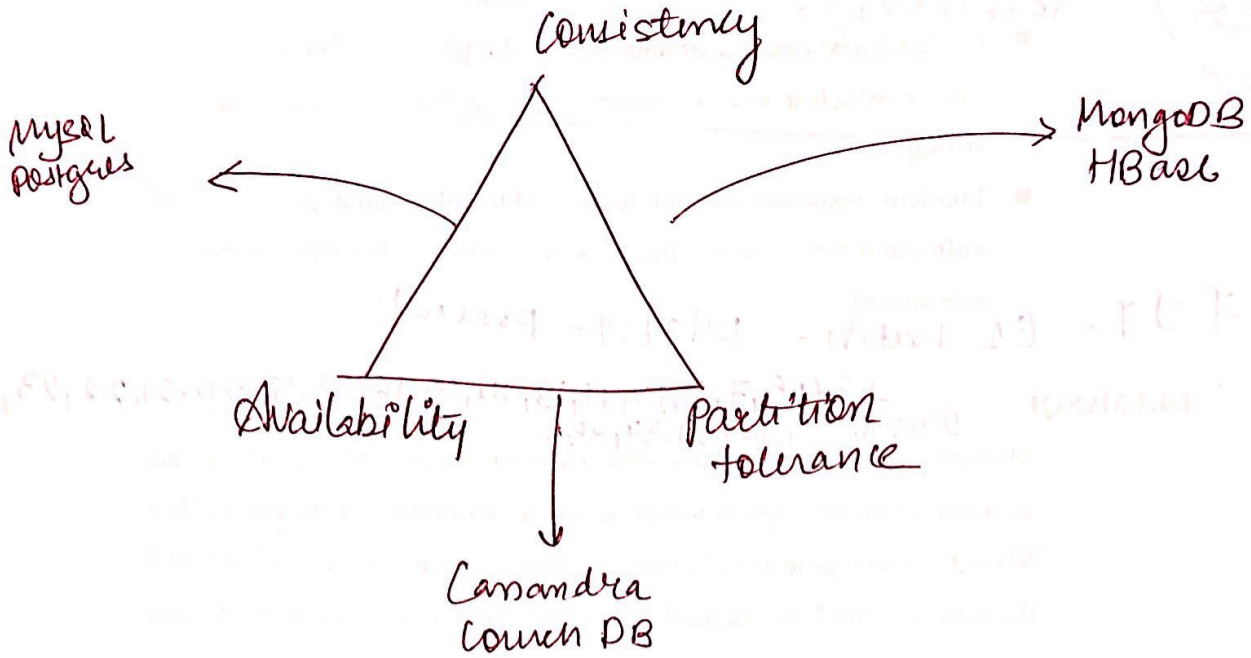
→ every node and edge has a unique identifier.



→ Best for social networks, logistics, spatial data.

→ ex: Neo4j, ∞ graph, OrientDB etc.

* CAP Theorem:



Consistency: data is the same across the cluster, so you can read or write from/to any node and get the same data

Availability: Ability to access the cluster even if a node in the cluster goes down

Partition Tolerance: cluster continues to function even if there is a "partition" between (2) nodes

communication break

i.e both nodes are up, but can't communicate.

→ In order to get both Availability & partition tolerance, you have to give up consistency.

➢ Supports stunnel 4 connections for access to MySQL

→ Consider if you have (2) nodes.

→ There is break betn n/w communication between x & y.

→ So they can't sync updates.

→ At this point. you can either:

(1) Allow the nodes to get out of sync. (giving up consistency)

(2) Consider the cluster to be down (giving up availability).

→ All the combinations. available are:

CA: data is consistent between all node as long as all nodes are online.

CP: data is consistent between all node, & maintain partition tolerance by becoming unavailable when node goes down

AP: nodes remain online even if they can't communicate.

✱ Strict Consistency:

ACID compliance:

(1) Bank balance is $50
(2) deposit $100
(3) queried from any ATM anywhere, is $150
(4) withdraw $40
(5) queried from any ATM anywhere, is $110

✱ Eventual consistency: wether report

→ To have copies of data on multiple machines to get high availability & scalability.
→ Thus, changes made to any data item on one machine has to be propogated to other replicas.
→ Data replication may not be instantaneous as some copies will be updated immediately while others in due course of time.
→ These copies may be mutually, but in due course of time, they become consistent.

✱ BASE → Basically Available, Soft State, Eventual consistency

→ DB is available all the time as per CAP theorem

Soft state → even without an input, the system may change
eventual consi → The system will become consistent over time.

* HBASE

→ designed to provide quick random access to huge amounts of structured data.

→ column-oriented database & the tables in it are sorted by row.

→ The table schema defines only column families, which are key-value pairs.

→ A table have multiple column families & each column family can have any no. of columns

→ Subsequent column values are stored contiguously on the disk.

→ In HBase:

- Table - collection of rows.
- Row - collection of column families
- Column family - " " columns.
- columns - " " key value pairs.

Row id	column family				Column family			Column family		
	col1	col2	col3	col4	col1	col2	col3	col1	col2	col3

→ Rows are sorted alphabetically by the row key. as they are stored.

→ for this reason, the design of the row key is very important.

→ The reason is to store data in a such a way that related rows are near each other.

→ A common row key pattern is website domain.

→ store them in reverse.

e.g { org.apache.www,
org.apache.mail,
org.apache.jira }

This way all of the apache domains are near each other in the table, rather than being spread out.

→ A table in HBase would look like.

	column family - Personal		column family - office	
Rowkey	Name	Residance phone	phone	Address.
00001	John	415-111-1234	415-212-[illegible]	1021 market-[illegible]
00002	Paul			

```
{
  "00001":
    { "personal":
      { "Name":
          { "Timestamp1": "John"
          }
        "Residence phone":
          { "TS1": "—",
            "TS2": "—"
          }
      }
    "office":
    }
}
```

* Row-Oriented Vs Column Oriented

RDBMS - Row-oriented
HBase - Column-Oriented.

→ Row-oriented databases store table records in a sequence of rows.

→ column-oriented databases store table records in a sequence of columns.
i.e the entries in a column are stored in contigious locations on disks.

e.x

Cust Id	Name	Addr	Product Id
1	John	US	231
2	Paul	UK	520

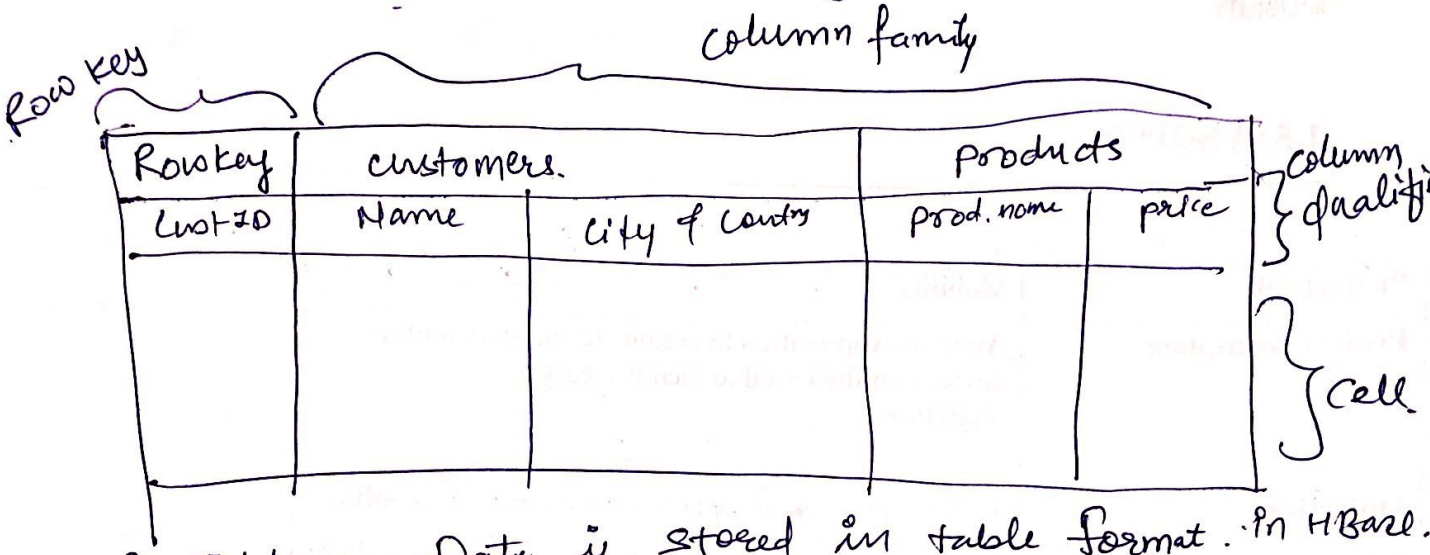
→ Row oriented → 1, John, US, 231
2, Paul, UK, 520.

→ column oriented → 1, 2, John, Paul, US, UK, 231, 520.

→ when amount of data is very huge, like in terms of petabytes or exabytes, we use column-oriented approach.

→ eg, the data of a single column is stored together and can be accessed faster

→ HBase tables has following components.



① Tables: Data is stored in table format. In HBase. here, tables are in column-oriented format

② Row key: used to search records which makes searches fast.

③ column families: various columns are combined in a column family.
- These column families are stored together which makes the searching process faster cz data belonging to same column family can be accessed together in a single seek.

④ column qualifiers: column's name

⑤ cell: data is stored in cells.
- The data is dumped into cells which are specifically identified by rowkey & column qualifier

⑥ Timestamp: Combination of date & time. whenever data is stored, it is stored with its timestamp.
- This makes it easy to search for a particular version of data.

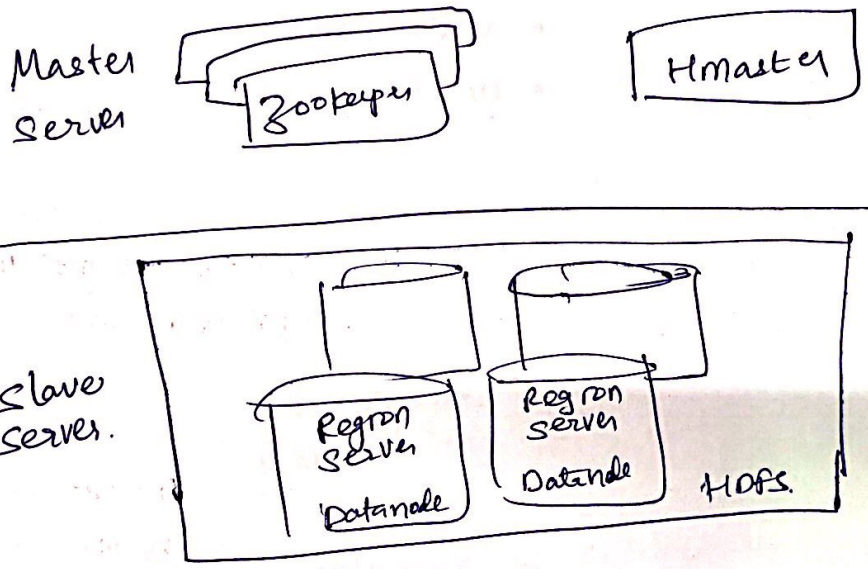
* HBASE ARCHITECTURE :

⇒ Components of HBase Architecture:

(3) main components.

1. HMaster server
2. HBase Region server
3. Regions.

and 4th component is Zookeeper.



→ All these servers - (HMaster, Region server, Zookeeper) are placed to coordinate & manage Regions & perform various operations inside Regions.

* Re

✱ Regions:

→ A region contains all the rows between the start key and the end key assigned to that region.

→ HBase tables divided into such a way that all columns of a column family is stored one region.

→ Each region contains the rows in a sorted order.

→ Many regions are assigned to a Region Server, which is responsible for handling, managing, executing reads & writes operations on that set of regions.

→ Region has a default size of 256 MB which can be configured according to the need.

→ A group of regions is served to the clients by a Region Server.

→ A region server can serve approximately 1000 regions to the client.

✱ HMaster:

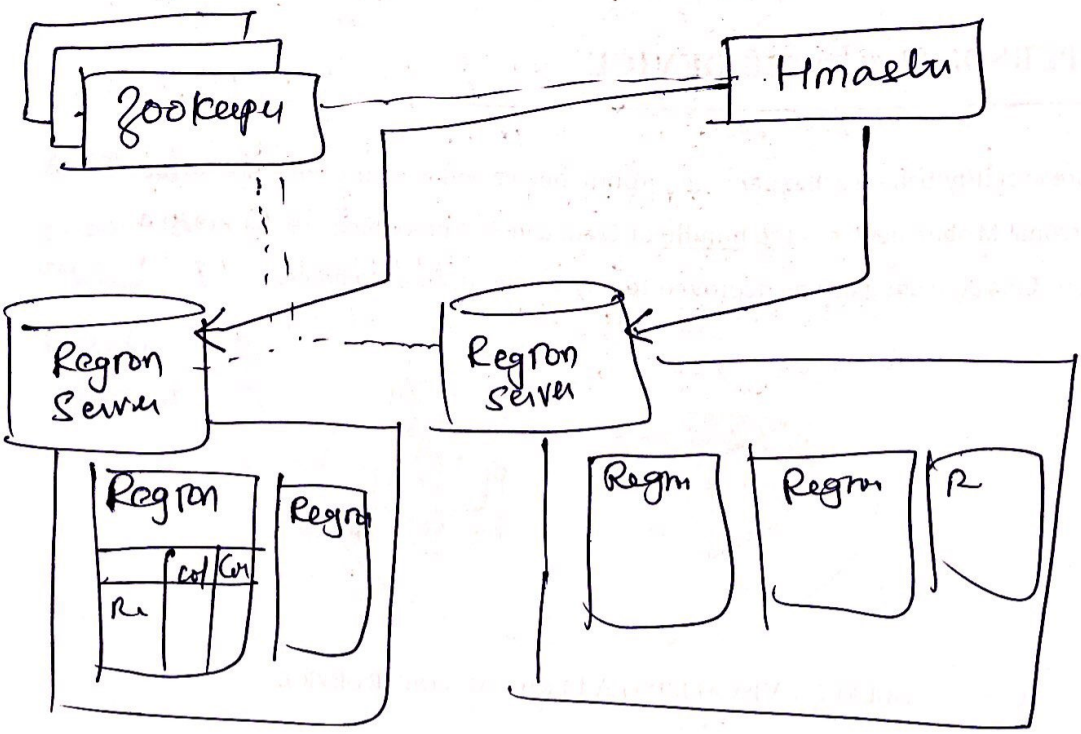
→ HMaster handles a collection of Region Server which resides on a Datanode.

→ HBase HMaster performs DDL operations & assigns regions to the Region Servers.

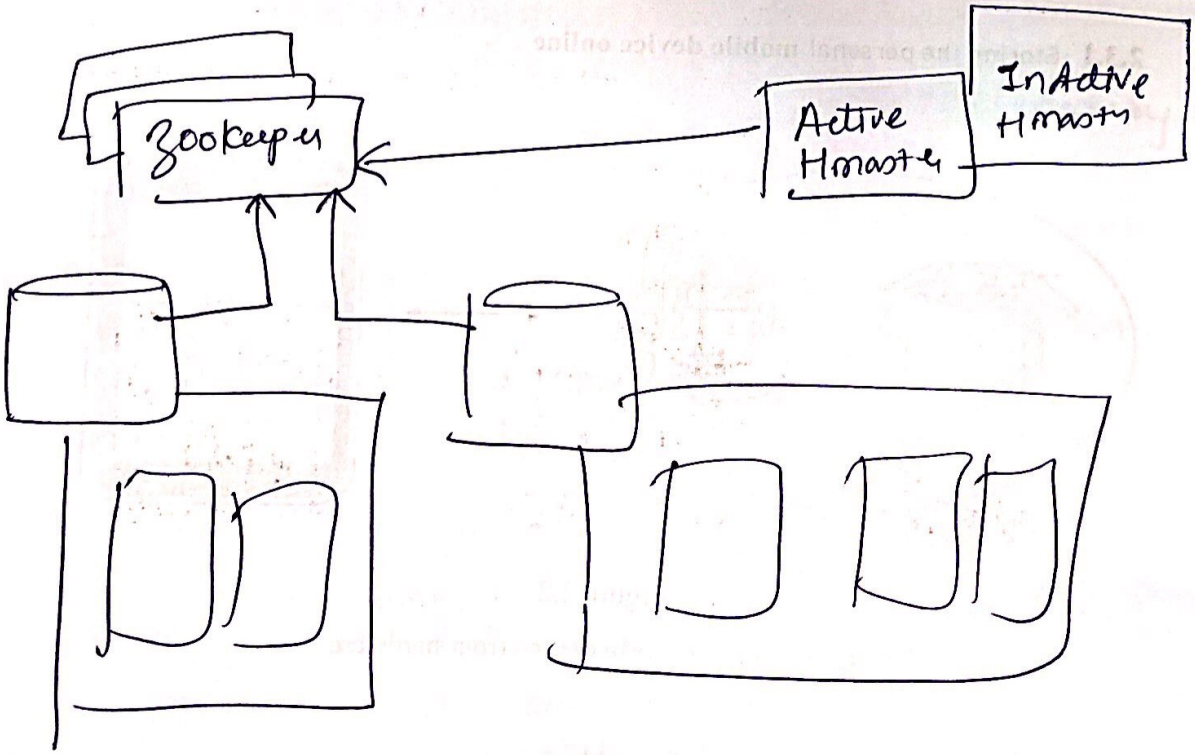
→ It coordinates and manages the Region Server.

→ It monitors all the Region Servers instances in the cluster (with the help of Zookeeper) & performs recovery activities whenever any region server is down.

→ It provides an Interface for creating, deleting & updating tables.



★ Zookeeper - The coordinates



→ Zookeeper acts like a coordinator inside HBase distributed environment.

→ It helps in maintaining server state inside the cluster by communicating through sessions.

→ Every Region server along with HMaster server sends continious heartbeat at regular interval to Zookeeper and it checks which server is alive and available.

→ Here, Inactive server act as a backup for active server.

→ Active master sends the heartbeat to the zookeeper while Inactive Hmaster listens for the notification send by active HMaster.

→ If the active HMaster fails to send a heartbeat, the session is delited and the Inactive HMaster becomes active.

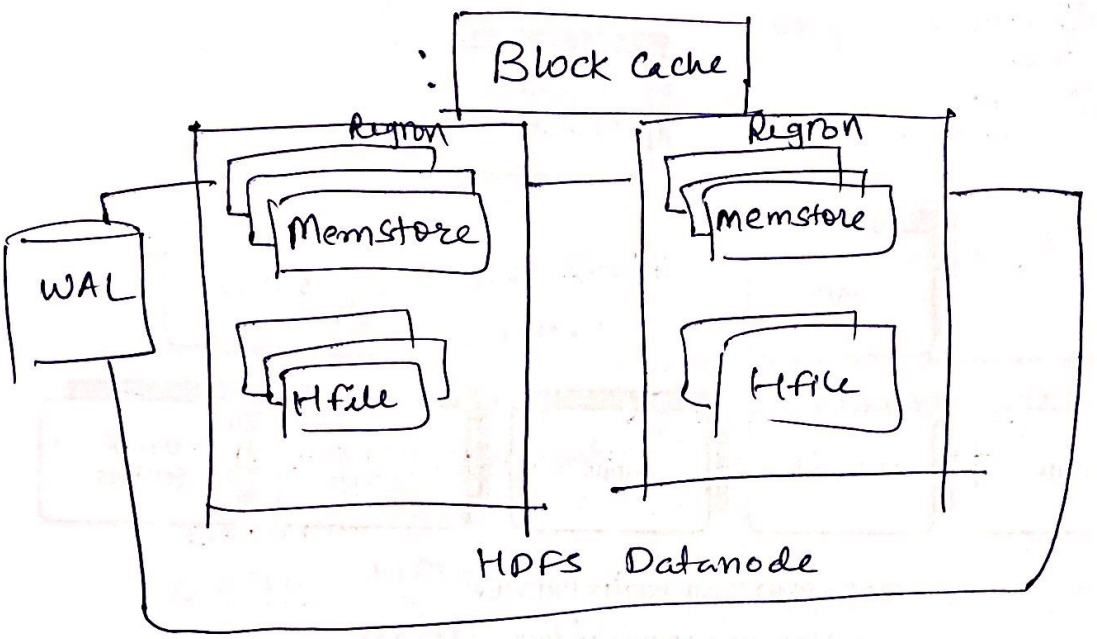
→ While if Region server fails to send a heartbeat, the session is expired & all listeners are notified about it.

→ Then HMaster performs suitable recovery actions.

→ Zookeeper also maintains the .META server's path, which helps any client in searching for any region.

* Region Server:

→ Region Server maintains various regions running on top of HDFS.



① ~~WAL~~:

→ Components of Region server:

① WAL: Write Ahead log.
- It is a file attached to every region server inside the distributed environment.
- The WAL stores the new data that hasn't been persisted or committed to permanent storage.
- It is used in case of failure to recover datasets.

② Block Cache: It stores the frequently read data in the memory.
- If the data in Block Cache is least recently used, then that data is removed from Block Cache.

③ Memstore: It is write cache.
- It stores all the incoming data before committing it to the disk or permanent memory
- Memstore for each column family in a region.
- The data is sorted in lexographical order before committing it to the disk.

④ Hfile: It stores the actual cells on the disk.

* Search mechanism

* → Zookeeper stores the META table location.

① client retrieves the location of the META table from Zookeeper

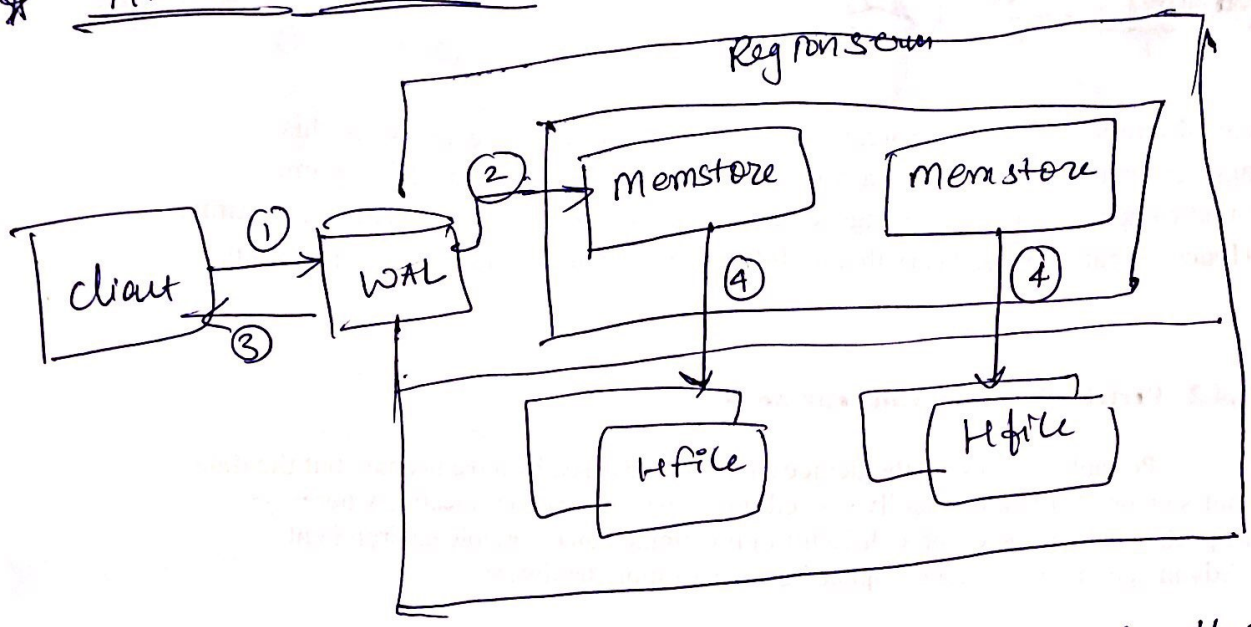
② client then requests for the location of the Region server of corresponding row key from the META table to access it.

③ Then it will get the row location by requesting from the corresponding Region server.

→ For future references, the client uses its Cache to retrieve the location of META table & previously read row key's Region server.

→ client will not refer to the META table, until & unless there is a miss.

* HBase Write Mechanism:



(1) Whenever client has a write request, the client writes the data to the WAL
- The edits are then appended at the end of the WAL file.
- This WAL file is maintained in every Region server uses it to recover data which is not committed to the disk.

(2) Once the data is written to the WAL, then it copied to the memstore

(3) Once the data is placed in memstore, then client receives the acknowledgement.

(4) When memstore reaches the threshold, it dumps or commits the data into a Hfile.

* HBase Read Mechanism:

→ first client retrieves the location of the Region server from .META Server if the client does not have it in its cache memory.

→ (1) for reading the data, the scanner first looks for the row cell in Block cache.
- Here all the recently read key value pairs are stored.

(2) If scanner fails to find the required result, it moves to the memstore.
- then it searches for the most recently written files, which has not been dumped yet in Hfile.

(3) At last, it will use block cache to load the data from Hfile.

* HBase: compaction

→ HBase combines Hfiles to reduce the storage and reduce the no. of disk seek needed for a read.

→ This process is called compaction.

→ compaction chooses some Hfiles from a region and combines them.

→ (2) types

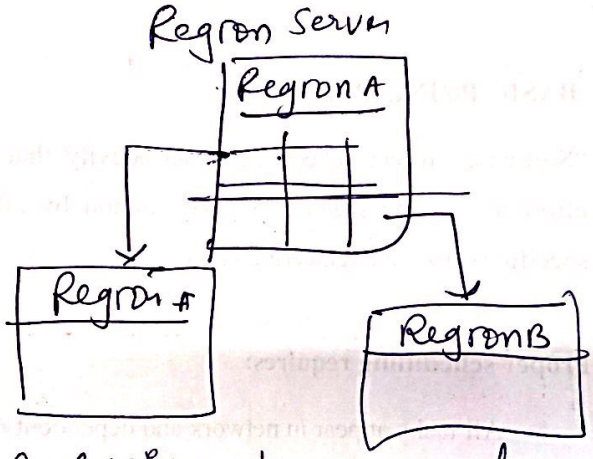
↳ (1) Minor compaction
- HBase automatically picks smaller Hfiles and recommits them to bigger Hfiles
- It performs merge sort for committing smaller Hfiles to bigger Hfiles.
- helps in storage space optimization.

↳ (2) Major compaction:

- HBase merges and recommits the smaller Hfiles of a region to a new Hfile.
- Same column families are placed together in the new Hfile.
- It drops deleted & expired cell in this process.
- It increases read performance.

→ During compaction process, input-output disk & n/w traffic might get congested.
→ known as write amplification.

* HBase: Region split



→ whenever a region becomes large, it is divided into (2) child regions.
→ The split is reported to the HMaster.
→ This is handled by the same Region server until the HMaster allocates them to a new Region server for load balancing.

* HBase Crash and Recovery:

→ whenever a Region Server fails, Zookeeper notifies to the HMaster about the failure.

→ Then HMaster distributes and allocates the regions of crashed RegionServer to many active region servs.

→ To recover the data of the memstore of the failed Region server, the HMaster distributes the WAL to all the Region servers.

→ Each region server re-executes the WAL to build the Memstore for that failed region's column family.

→ The data is written in chronological order (in a timely order) in WAL. Therefore, Re-executing that WAL means making all the change that were made and stored in the memstore file.

→ So, after all the Region servers executes the WAL, the memstore data for all column family is recovered.