HBASE

Outline

- Introduction to NoSQL
- CAP Theorem
- Introduction to HBase
- HBase Background
- HBase Features
- HBase Data Model
- HBase Architecture
- HBase Components
- HBase Clients
- HBase Shell
- HBase Java API
- HBase Integration with Hive

Introduction to NoSQL

What is NoSQL?

- It's a whole new way of thinking about a database and encompasses a wide variety of different database technologies.
- A non-relational and largely distributed database system that enables rapid analysis of extremely high-volume, disparate (different) data types.

Why NoSQL?

- **The Benefits of NoSQL :** A very flexible and schema-less data model, horizontal scalability, distributed architectures
- Dynamic Schemas
- Auto-sharding
- Replication
- Integrated Caching

NoSQL Database Types

- **Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key- value pairs, or key-array pairs, or even nested documents.

- **Graph stores** are used to store information about networks, such as social connections. Graph stores include Neo4J and HyperGraphDB.

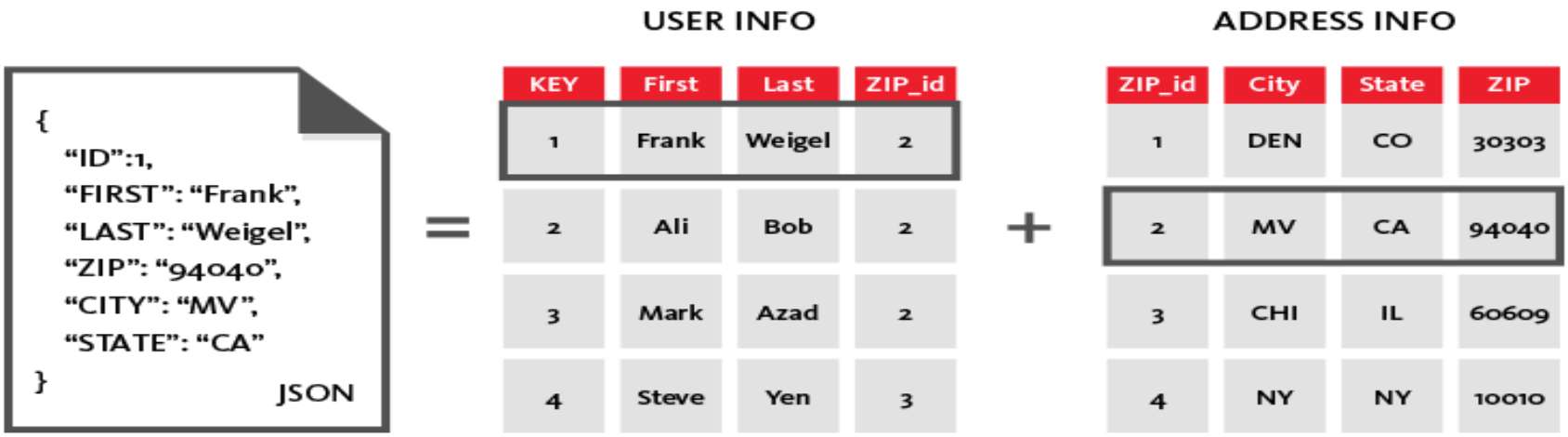
- **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or "key"), together with its value. Examples of key-value stores are Riak and Voldemort. Some key-value stores, such as Redis, allow each value to have a type, such as "integer", which adds functionality.

- **Wide-column stores** such as Cassandra and **HBase** are optimized for queries over large datasets, and store columns of data together, instead of rows

NoSQL vs. SQL Summary

	SQL Database	NoSQL Database
Types	One type (SQL database) with minor variations	Many different types including key-value stores, document databases, wide-column stores, and graph databases
Development History	Developed in 1970s to deal with first wave of data storage applications.	Developed in 2000s to deal with limitations of SQL databases, particularly concerning scale, replication and unstructured data storage.
Examples	MySQL, Postgres, Oracle Database	MongoDB, Cassandra, HBase, Neo4j ,Riak, Voldemort, CouchDB ,DynamoDB
Schemas	Structure and data types are fixed in advance.	Typically dynamic. Records can add new information on the fly, and unlike SQL table
Scaling	Vertically	Horizontally
Data Manipulation	Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE…	Through object-oriented APIs

NoSQL's More Flexible Data Model



- Relational and NoSQL data models are very different. The relational model takes data and separates it into many interrelated tables that contain rows and columns. Tables reference each other through foreign keys that are stored in columns as well.

- NoSQL databases have a very different model. For example, a document-oriented NoSQL database takes the data you want to store and aggregates it into documents using the JSON format.

CAP THEROM

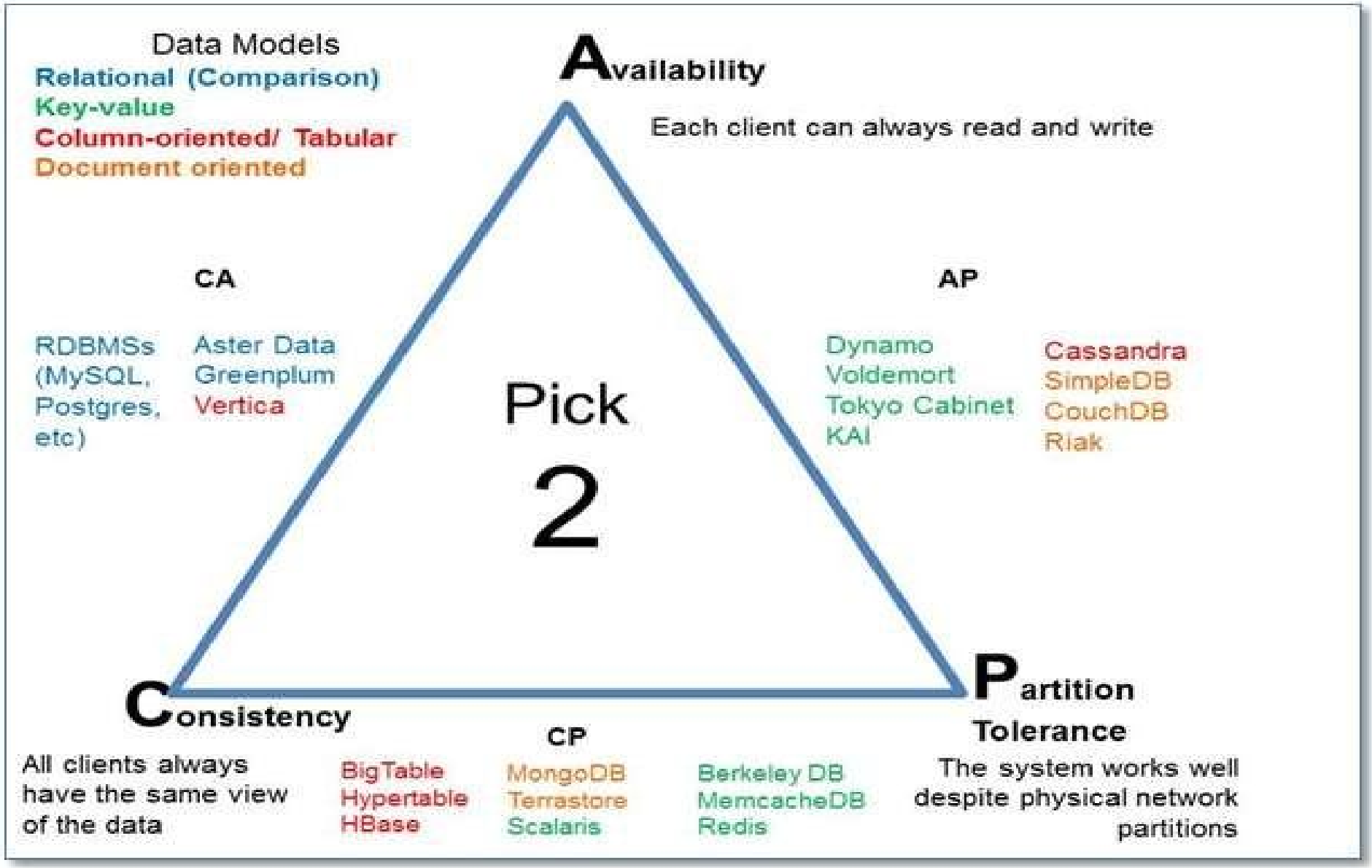
- In theoretical computer science, the CAP theorem, also named *Brewer's theorem.*

- *The CAP theorem was first proposed by Eric Brewer of the University of California, Berkeley, in 2000, and then proven by Seth Gilbert and Nancy Lynch of MIT in 2002.*

- it’s important to keep three core requirements in mind ;
  1. Consistency
  2. Availability
  3. Partition-Tolerance

- **Consistency (C)** requires that all reads initiated after a successful write return the same and latest value at any given logical time.
- **Availability (A)** requires that every node (not in failed state) always execute queries. as Write opertion
- **Partition Tolerance (P)** requires that a system be able to re-route a communication when there are temporary breaks or failures in the network. The goal is to maintain synchronization among the involved nodes.

Brewer's theorem states that it's typically not possible for a distributed system to provide all three requirements Simultaneously because one of them will always be compromised.


Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
Pick 2
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Consistency
All clients always have the same view of the data
CP
BigTable
Hypertable
HBase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance
The system works well despite physical network partitions

HBase

- Apache **HBase** is an open-source, distributed, versioned, fault-tolerant, scalable, column-oriented store modeled after Google's Bigtable, with random real-time read/write access to data.

- HBase is a column-oriented database

- when you need random, real-time read/write access to your Big Data.

- When project's goal is the hosting of very large tables
  -- billions of rows X millions of columns

Features of HBase

- Scalable
- Automatic failover
- Consistent reads and writes
- Sharding of tables
- Failover support
- Classes for backing hadoop mapreduce jobs
- Java API for client access
- Thrift gateway and a REST-ful Web
- Shell support

Background

- 2006.NOV
  - Google releases paper on BigTable
- 2007.FEB
  - Initial HBase prototype created as Hadoop contrib.
- 2007.OCT
  - First useable HBase
- 2008.JAN
  - Hadoop become Apache top-level project and HBase becomes subproject
- 2008.MAR
  - HBase Initial Public Release
- 2020.MAY

  Hbase Latest Stable Release 2.2.5

Difference Between HDFS and Hbase

- HDFS is a distributed file system that is well suited for the storage of large files. It's documentation states that it is not, however, a general purpose file system, and does not provide fast individual record lookups in files. HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables.

What it is not

- No sql
- No relation
- No joins
- Not a replacement of RDBMS

What Is The Difference Between HBase and RDMS?

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
No query language	SQL
Wide tables	Narrow tables
Joins using MR - not optimized	Optimized for joins (small, fast ones too!)
Tight integration with MR	Not really...

Row-Oriented vs. Column-Oriented data stores.

Row-oriented data stores –

- Data is stored and retrieved one row at a time and hence could read unnecessary data if only some of the data in a row is required.
- Easy to read and write records
- Well suited for OLTP systems
- Not efficient in performing operations applicable to the entire dataset and hence aggregation is an expensive operation
- Typical compression mechanisms provide less effective results than those on column-oriented data stores

Column-oriented data stores –

- Data is stored and retrieved in columns and hence can read only relevant data if only some data is required
- Read and Write are typically slower operations
- Well suited for OLAP systems
- Can efficiently perform operations applicable to the entire dataset and hence enables aggregation over many rows and columns
- Permits high compression rates due to few distinct values in columns

HBase Data Model

Tables – The HBase Tables are more like logical collection of rows stored in separate partitions called Regions. As shown above, every Region is then served by exactly one Region Server. The figure above shows a representation of a Table.

Rows – A row is one instance of data in a table and is identified by a *rowkey*. Rowkeys are unique in a Table and are always treated as a byte[].

Column Families – Data in a row are grouped together as Column Families. Each Column Family has one more Columns and these Columns in a family are stored together in a low level storage file known as HFile. Column Families form the basic unit of physical storage to which certain HBase features like compression are applied. Hence it's important that proper care be taken when designing Column Families in table. The table above shows Customer and Sales Column Families. The Customer Column Family is made up 2 columns – Name and City, whereas the Sales Column Families is made up to 2 columns – Product and Amount.

Columns – A Column Family is made of one or more columns. A Column is identified by a Column Qualifier that consists of the Column Family name concatenated with the Column name using a colon – example: columnfamily:columnname. There can be multiple Columns within a Column Family and Rows within a table can have varied number of Columns.

Cell – A Cell stores data and is essentially a unique combination of *rowkey*, Column Family and the Column (Column Qualifier). The data stored in a Cell is called its value and the data type is always treated as byte[].

Version – The data stored in a cell is versioned and versions of data are identified by the timestamp. The number of versions of data retained in a column family is configurable and this value by default is 3.

HBase Data Model

Row Key	Customer		Sales	
Customer Id	Name	City	Product	Amount
101	John White	Los Angeles, CA	Chairs	$400.00
102	Jane Brown	Atlanta, GA	Lamps	$200.00
103	Bill Green	Pittsburgh, PA	Desk	$500.00
104	Jack Black	St. Louis, MO	Bed	$1600.00

Column Families

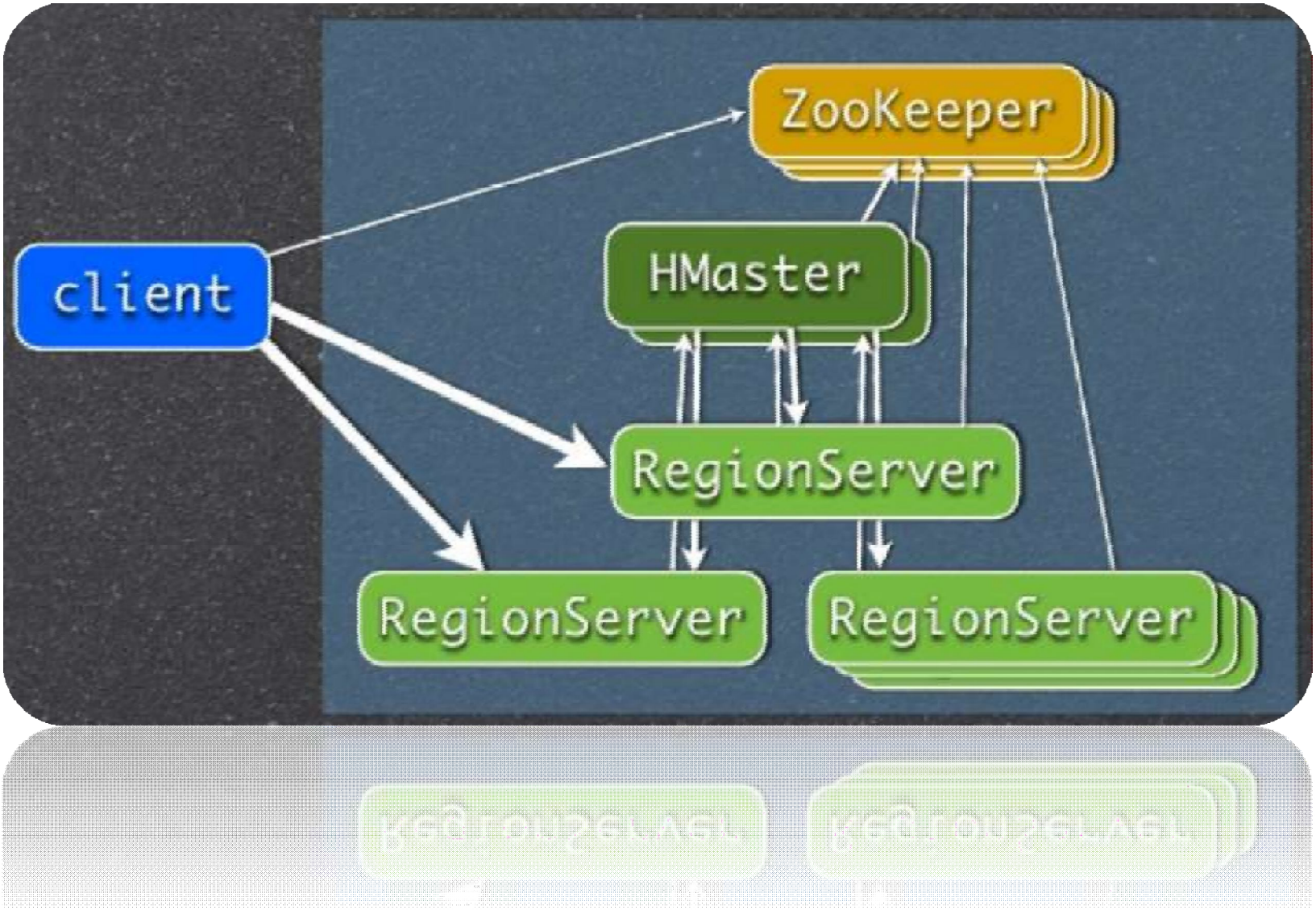
HBase Architectural Components

- HBase is composed of three types of servers in a master slave type of architecture.
- Region assignment, DDL (create, delete tables) operations are handled by the HBase Master process

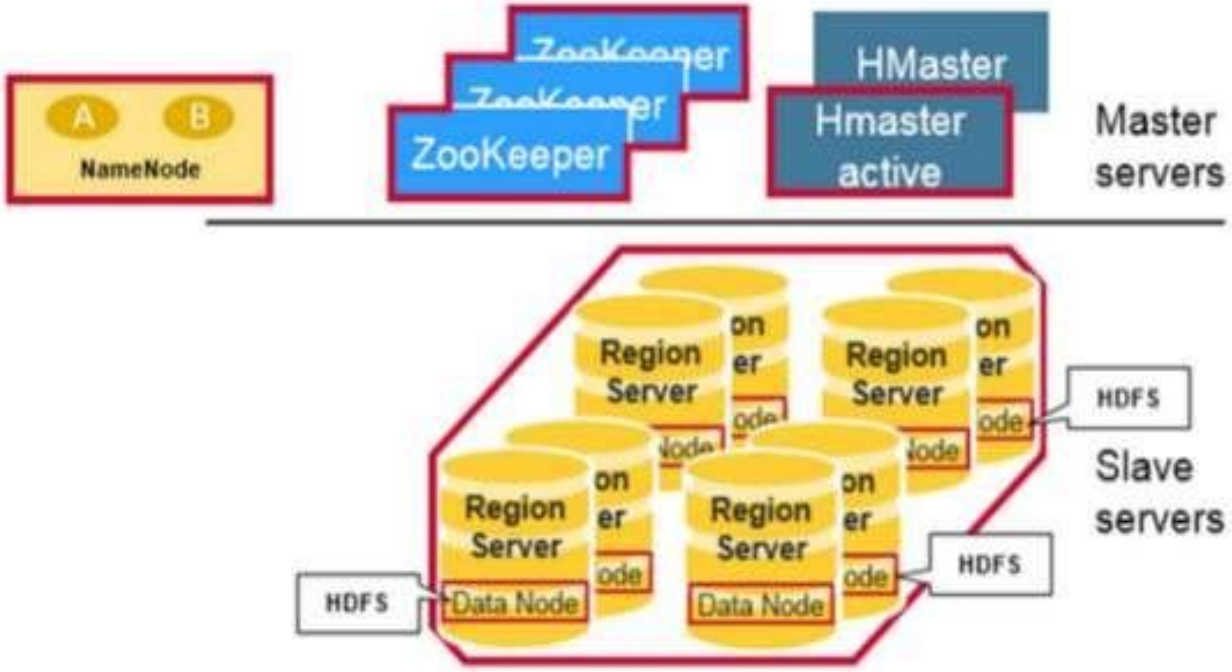
- Region servers serve data for reads and writes. When accessing data, clients communicate with HBase RegionServers directly..

- Zookeeper, which is part of HDFS, maintains a live cluster state.
- The Hadoop DataNode stores the data that the Region erver is managing.
- All HBase data is stored in HDFS files.

Components


ZooKeeper
client
HMaster
RegionServer
RegionServer
RegionServer

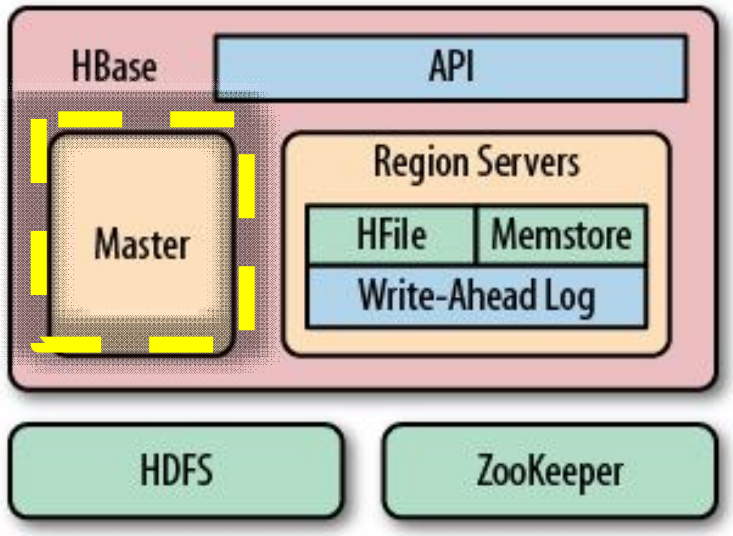
Other way around



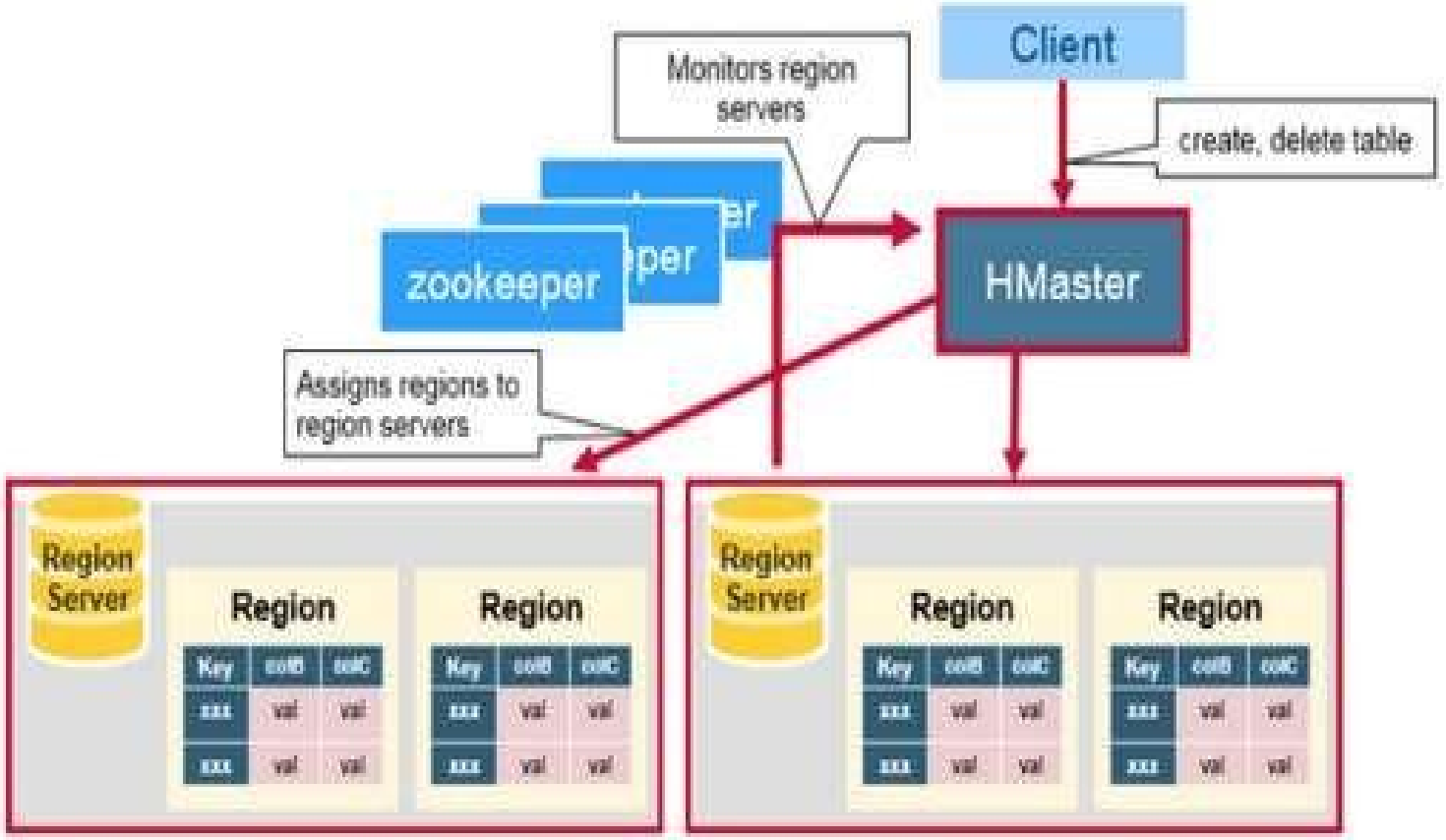
HMaster

- Master server is responsible for monitoring all RegionServer instances in the cluster, and is the interface for all metadata changes, it runs on the server which hosts namenode.

- Master controls critical functions such as RegionServer failover and completing region splits. So while the cluster can still run for a time without the Master, the Master should be restarted as soon as possible.

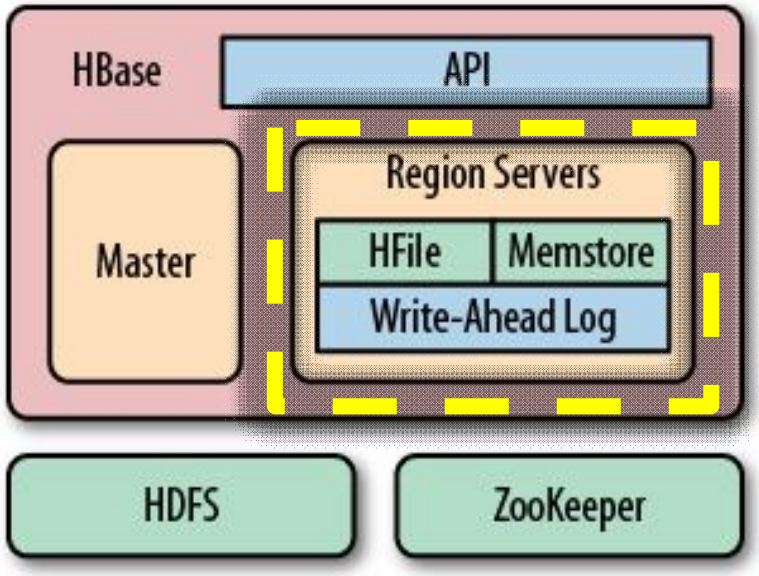


HMaster

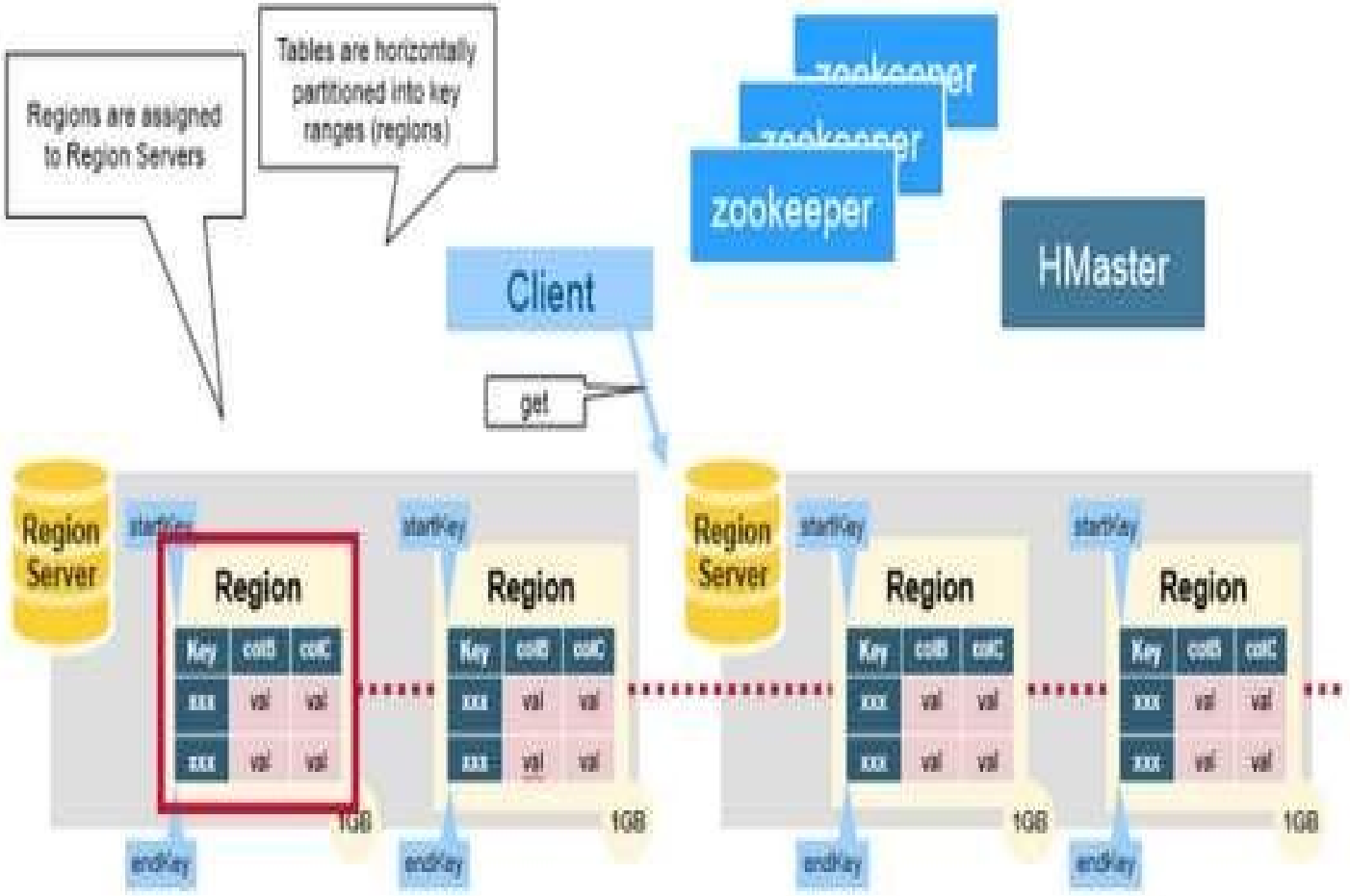

Client
Monitors region servers
create, delete table
zookeeper
HMaster
Assigns regions to region servers
Region Server
Region
Region
Region Server
Region
Region

Region Servers

- It is responsible for serving and managing regions, its like a data node for HBase.
- These can be thought of Datanode for Hadoop cluster. It serve the client request for the data.It handle the actual data storage and request.
- It consists of Regions or in better words tables.
- Regionservers are usually configured to run on servers of HDFS Datanode. Running Regionservers on the Datanode server has the advantage of data locality too

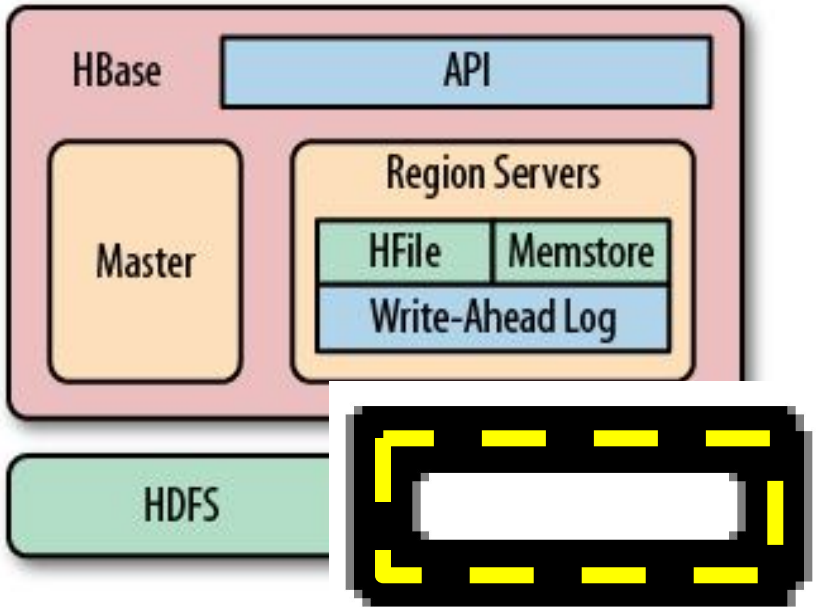


Region Servers

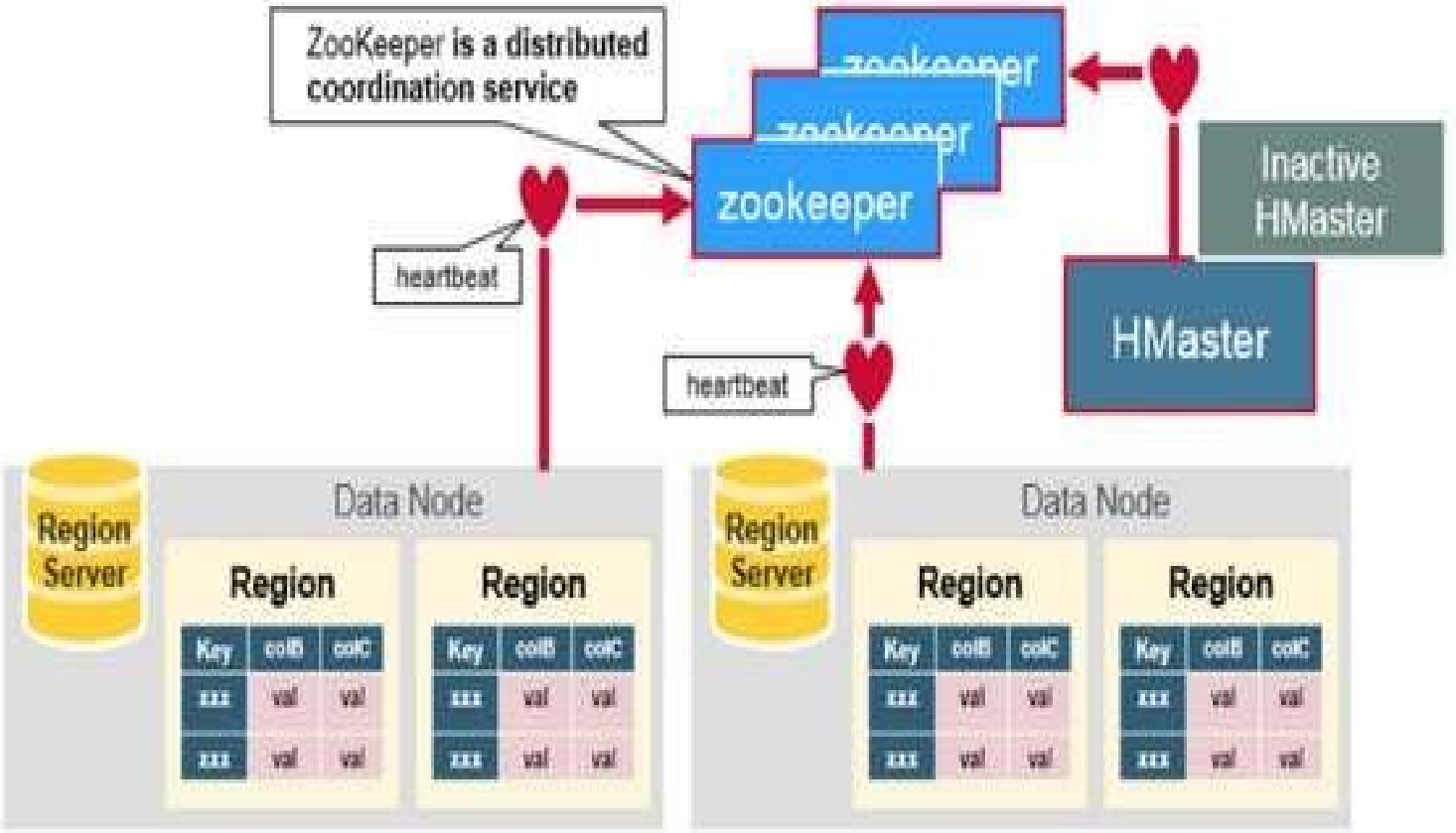


Zookeeper

- Zookeeper is an open source software providing a highly reliable, distributed coordination service
- Entry point for an HBase system
- It includes tracking of region servers, where the root region is hosted

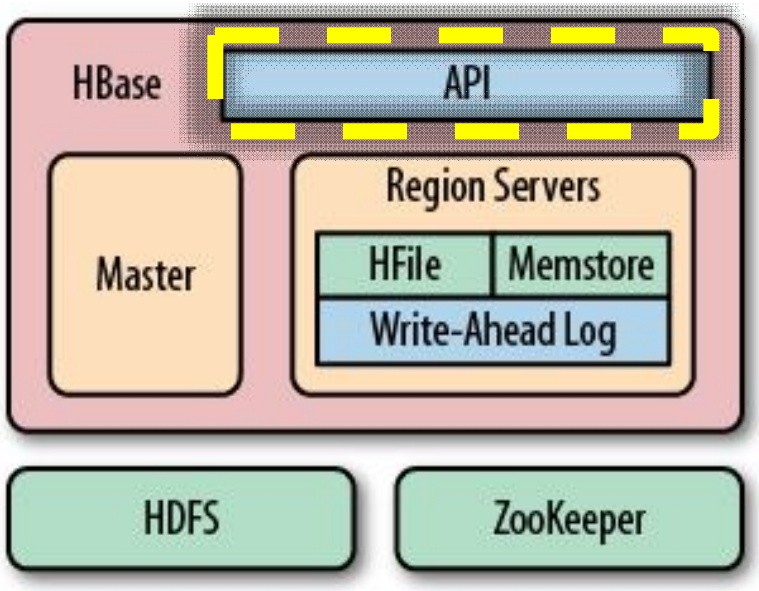


Zookeeper


ZooKeeper is a distributed coordination service
zookeeper
zookeeper
zookeeper
heartbeat
heartbeat
Inactive HMaster
HMaster
Region Server
Data Node
Region
Key
colB
colC
Region
Region Server
Data Node
Region
Region

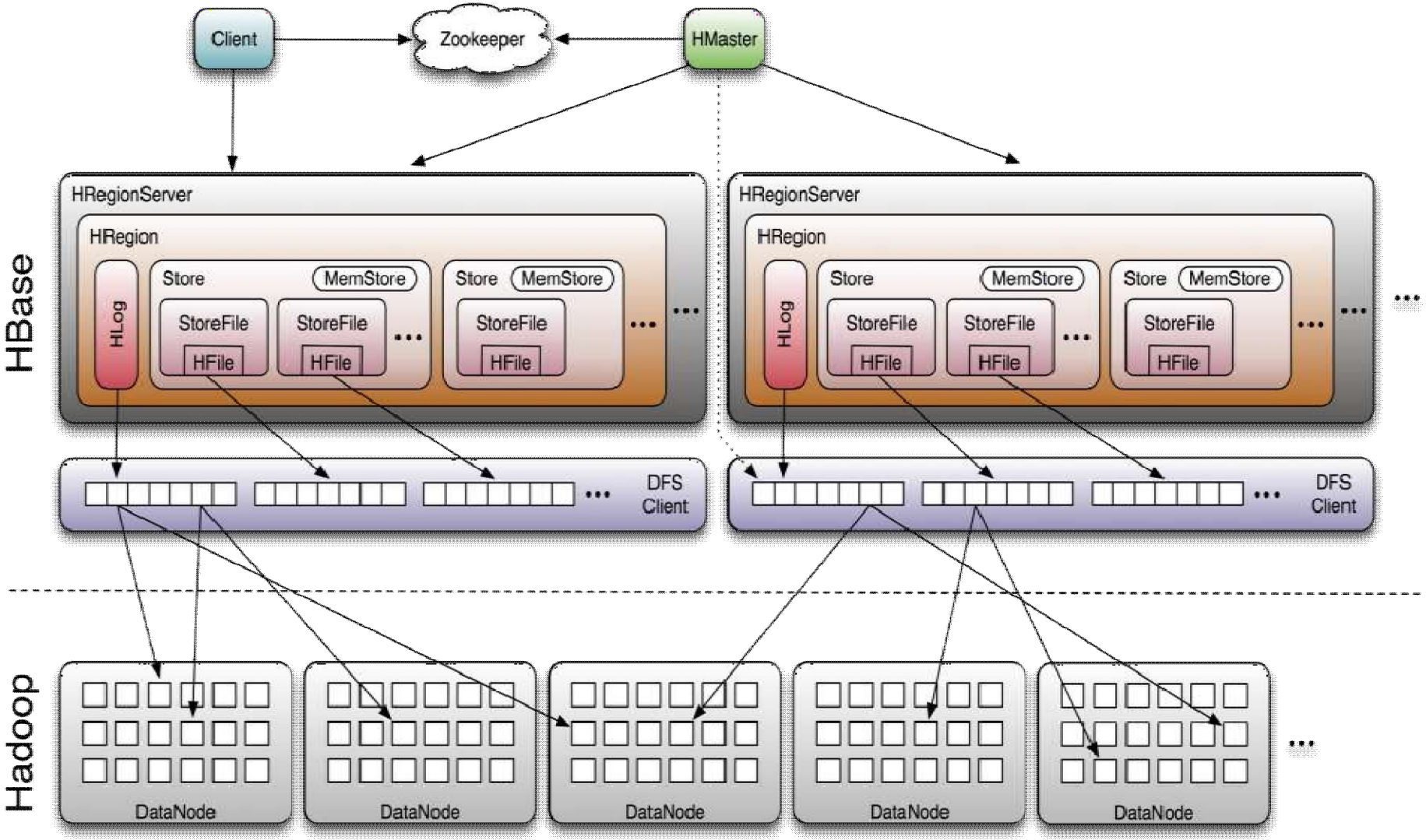
API

- Interface to HBase
- Using these we can we can access HBase and perform read/write and other operation on HBase.
- Java API, REST and Thrift

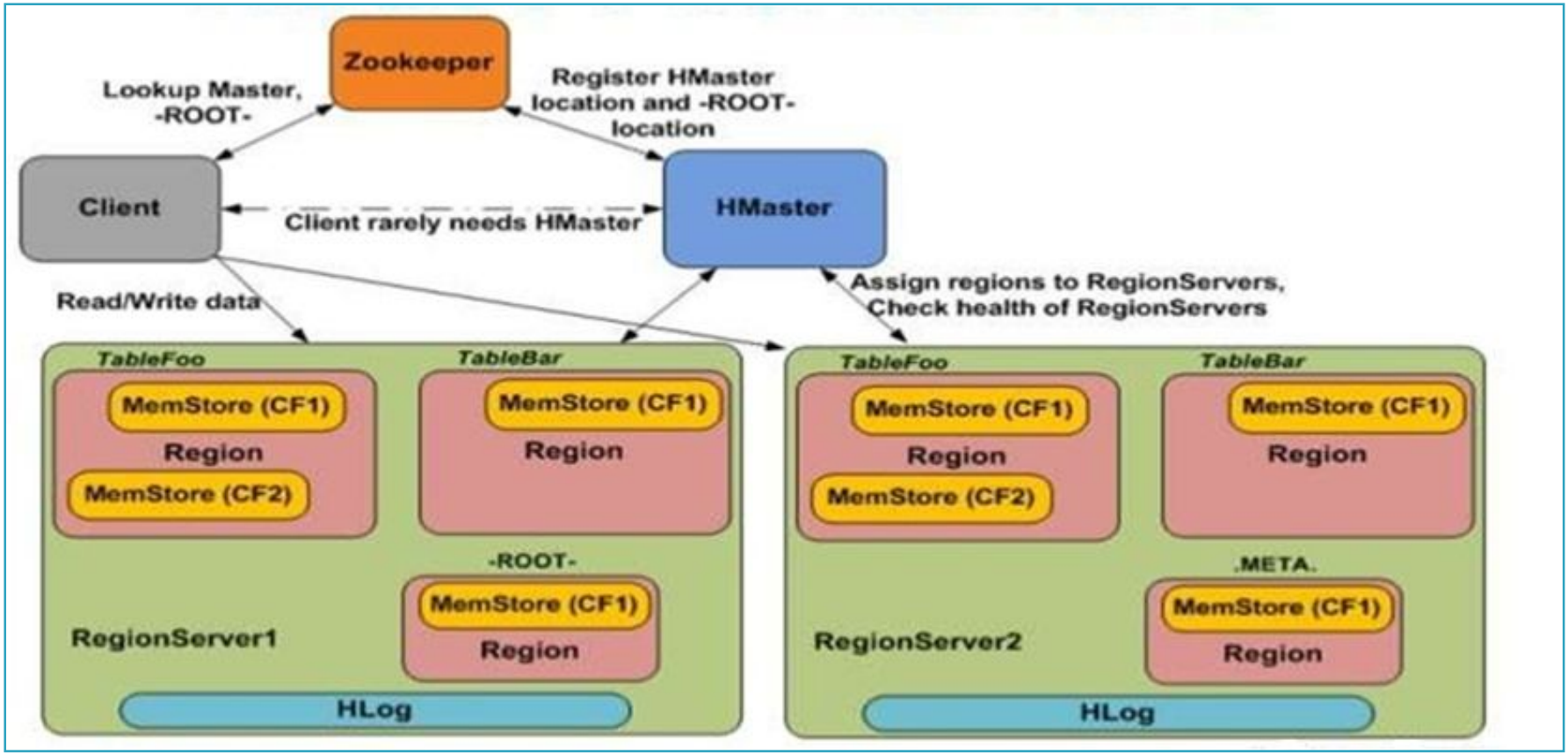


- Thrift API framework, for scalable cross- language services development, combines a software stack with a code generation engine to build services that work efficiently and seamlessly between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages.

HBASE ARCHITECTURE



HBASE ARCHITECTURE



HBASE ARCHITECTURE

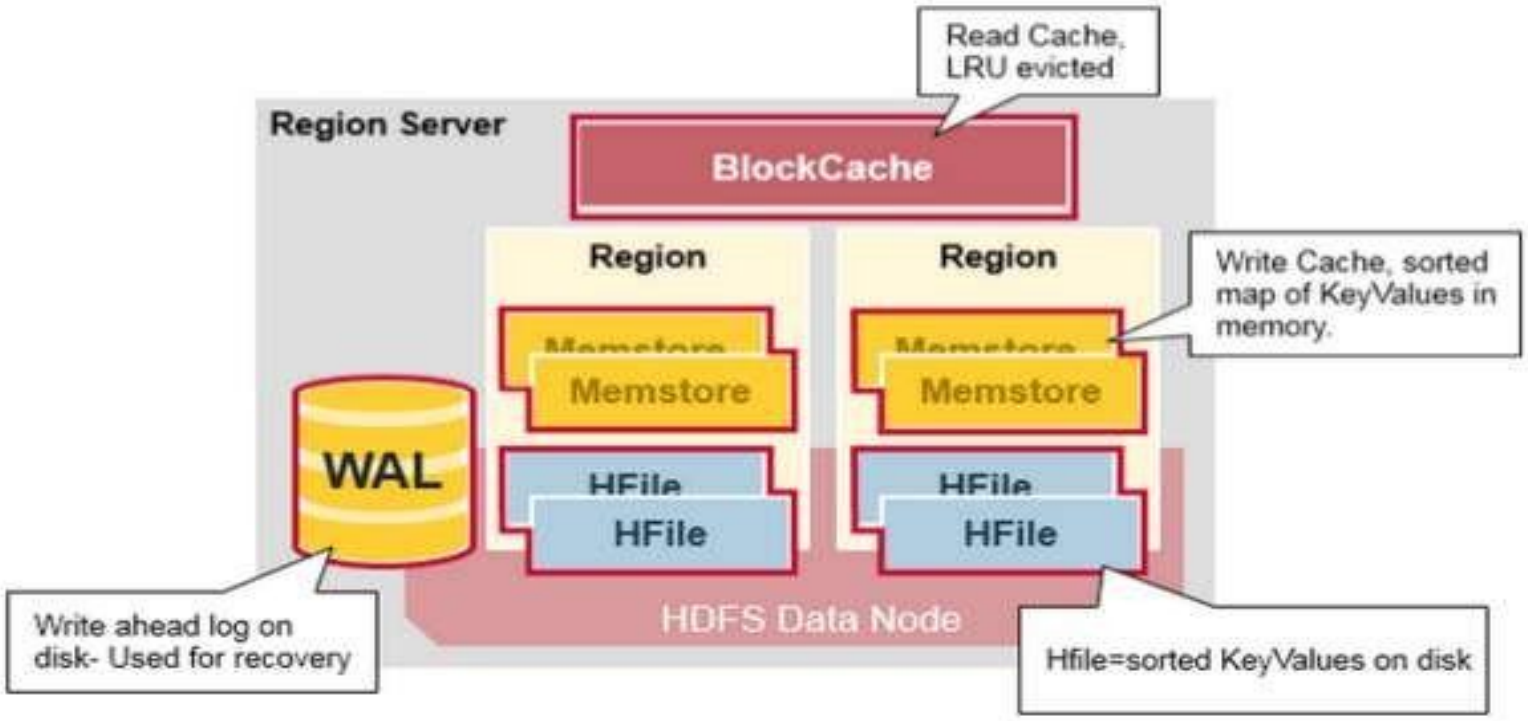
- WAL: Write Ahead Log is a file on the distributed file system. The WAL is used to store new data that hasn't yet been persisted to permanent storage; it is used for recovery in the case of failure.

- BlockCache: is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full.

- MemStore: is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk.
  There is one MemStore per column family per region.

- Hfiles store the rows as sorted KeyValues on disk.

HBASE ARCHITECTURE



HBase First Read or Write

- There is a special HBase Catalog table called the META table, which holds the location of the regions in the cluster. ZooKeeper stores the location of the META table.

- ZooKeeper stores the location of the META table.

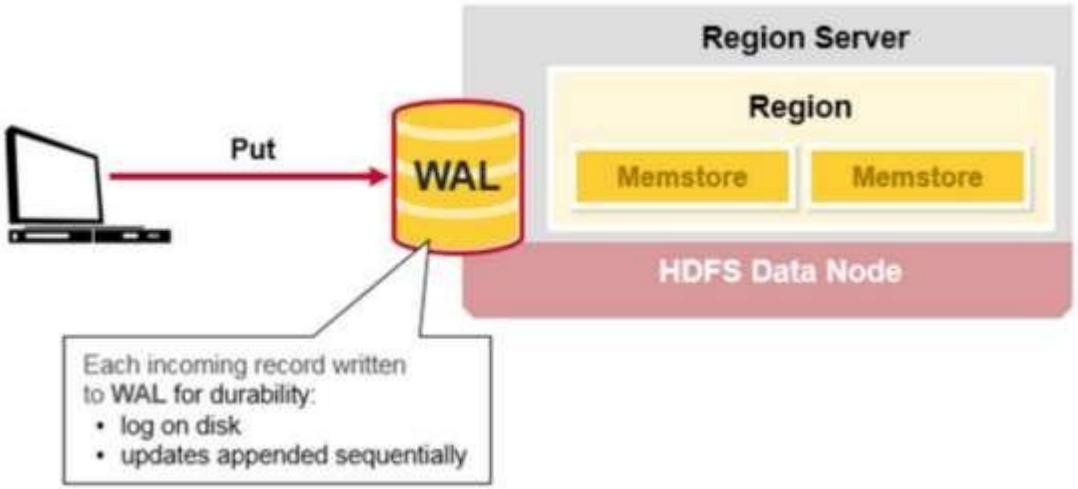
- This is what three steps happens the first time a client reads or writes to HBase:

1. The client gets the Region server that hosts the META table from ZooKeeper.
2. 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 with the META table location.
3. It will get the Row from the corresponding Region Server.

HBase Write Steps (1)

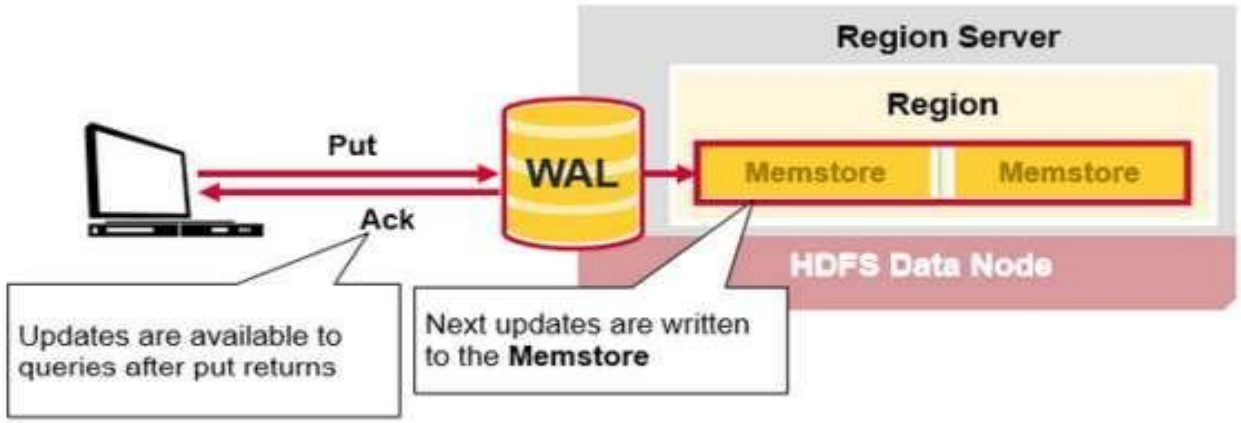
- When the client issues a Put request, the first step is to write the data to the write-ahead log, the WAL:

  - Edits are appended to the end of the WAL file that is stored on disk
  - The WAL is used to recover not-yet-persisted data in case a server crashes.



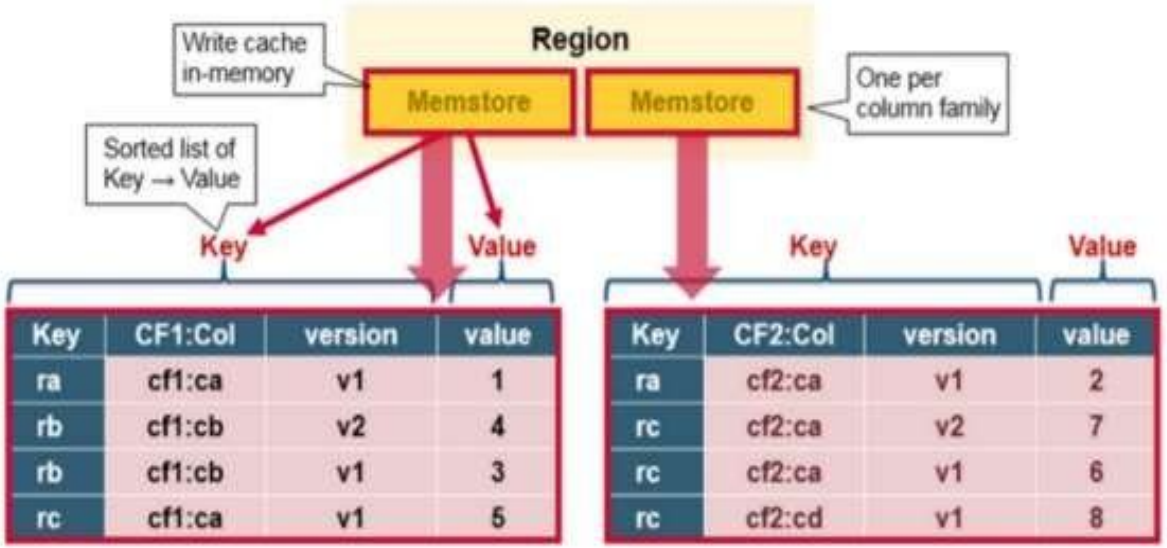
HBase Write Steps (2)

Once the data is written to the WAL, it is placed in the MemStore. Then, the put request acknowledgement returns to the client.



HBase MemStore

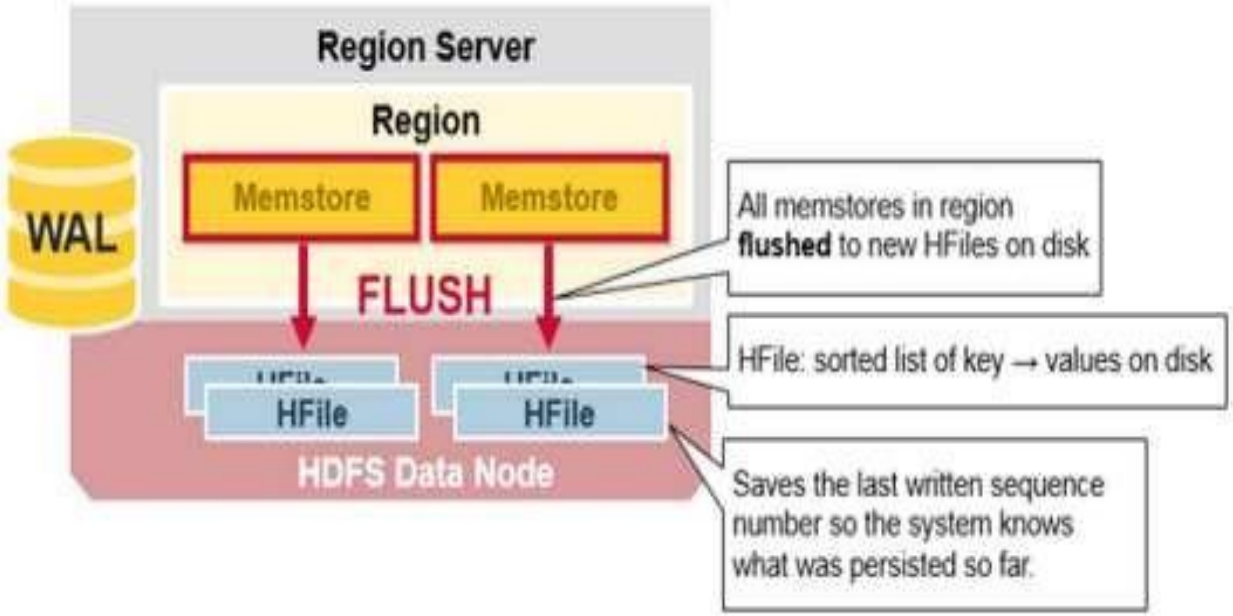
The MemStore stores updates in memory as sorted KeyValues, the same as it would be stored in an HFile. There is one MemStore per column family. The updates are sorted per column family.



Key	CF1:Col	version	value
ra	cf1:ca	v1	1
rb	cf1:cb	v2	4
rb	cf1:cb	v1	3
rc	cf1:ca	v1	5

Key	CF2:Col	version	value
ra	cf2:ca	v1	2
rc	cf2:ca	v2	7
rc	cf2:ca	v1	6
rc	cf2:cd	v1	8

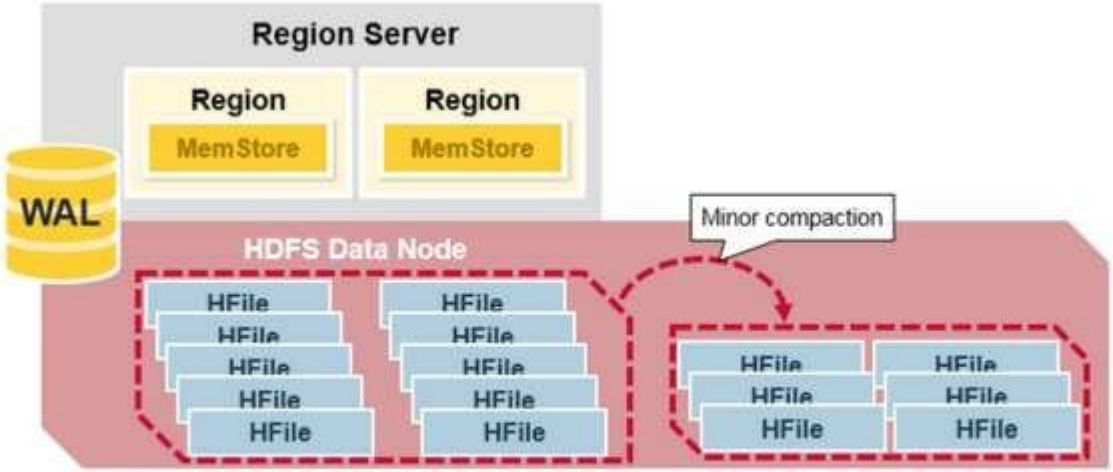
HBase HFile


Region Server
Region
Memstore
Memstore
WAL
FLUSH
HFile
HFile
HDFS Data Node
All memstores in region flushed to new HFiles on disk
HFile: sorted list of key → values on disk
Saves the last written sequence number so the system knows what was persisted so far.

HBase Compaction

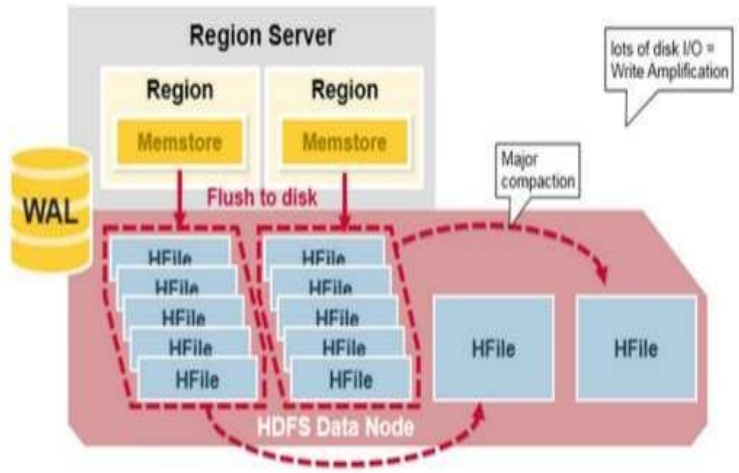
HBase Minor Compaction

HBase will automatically pick some smaller HFiles and rewrite them into fewer bigger Hfiles. This process is called minor compaction. Minor compaction reduces the number of storage files by rewriting smaller files into fewer but larger ones, performing a merge sort.



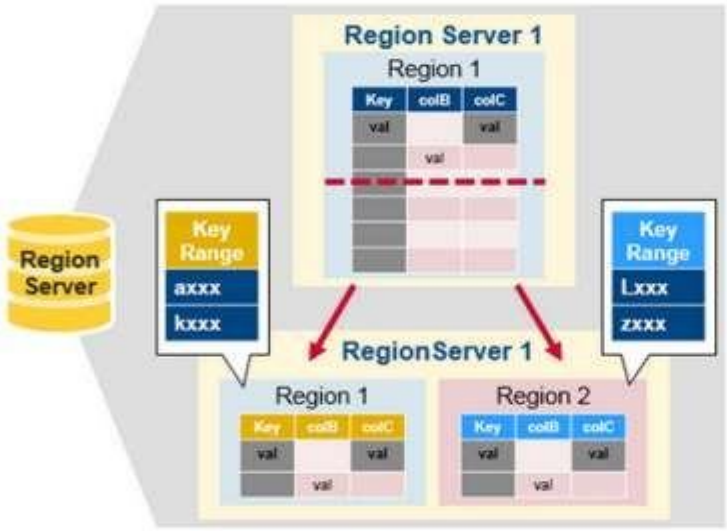
HBase Major Compaction

- Major compaction merges and rewrites all the HFiles in a region to one HFile per column family, and in the process, drops deleted or expired cells.
- since major compaction rewrites all of the files, lots of disk I/O and network traffic might occur during the process. This is called write amplification.
- Major compactions can be scheduled to run automatically. Due to write amplification, major compactions are usually scheduled for weekends or evenings.



Region Split

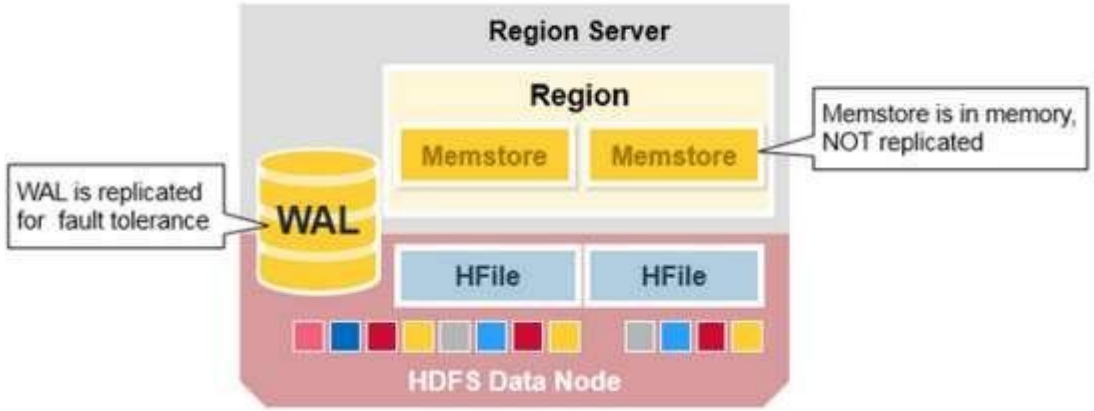
Initially there is one region per table. When a region grows too large, it splits into two child regions. Both child regions, representing one-half of the original region, are opened in parallel on the same Region server, and then the split is reported to the HMaster. For load balancing reasons, the HMaster may schedule for new regions to be moved off to other servers.



when region size > hbase.hregion.max. filesize → split

HDFS Data Replication

- The WAL file and the Hfiles are persisted on disk and replicated,



Start Hbase Server and Shell

- To start Hbase server, go up to HBase folder and run following command in terminal

  $ ./bin/start-hbase.sh

Note: HMaster, Zookeeper, Regionserver is started when we use above command

- Stop your HBase instance by running the stop script
  - $bin/stop-hbase.sh
  - stopping HBase.....

Web Interface

Master Web Interface

The Master starts a web-interface on port 60010 by default.

EX: http://localhost:60010/

Regionserver Web Interface

Regionserver starts a web-interface on port 60030 by default.

EX: http://localhost:60030/

HBASE Shell

- What is HBaseshell?

Hbase shell is a command line interface for reading and writing data to HBase database.

- To start Shell : Run following to start HBase shell

$ ./bin/hbase shell

This will connect hbase shell to hbase server.

Shell Commands

Commands	description
create	create a table in database
put	add a record in a table
get	retrieve a record from a table
Scan	retrieve a set of records from a table
delete, deleteall	delete a entire row or a column, or a cell from a table
alter	Alter a table (add or delete column family)
describe	Describe the named table
list	List all tables in database
disable/enable	Disable/enable the named table
drop	Drop the named table

Creating a Table using HBase Shell

- You can create a table using the **create** command, here you must specify the table name and the Column Family name. The **syntax** to create a table in HBase shell is shown below.

- create ‘<table name>’,’<column family>’

- **create 'emp', 'personal data', ’professional data’**

  Given below is a sample schema of a table named emp. It has two column families: “personal data” and “professional data”.

hbase(main):001:0 > list

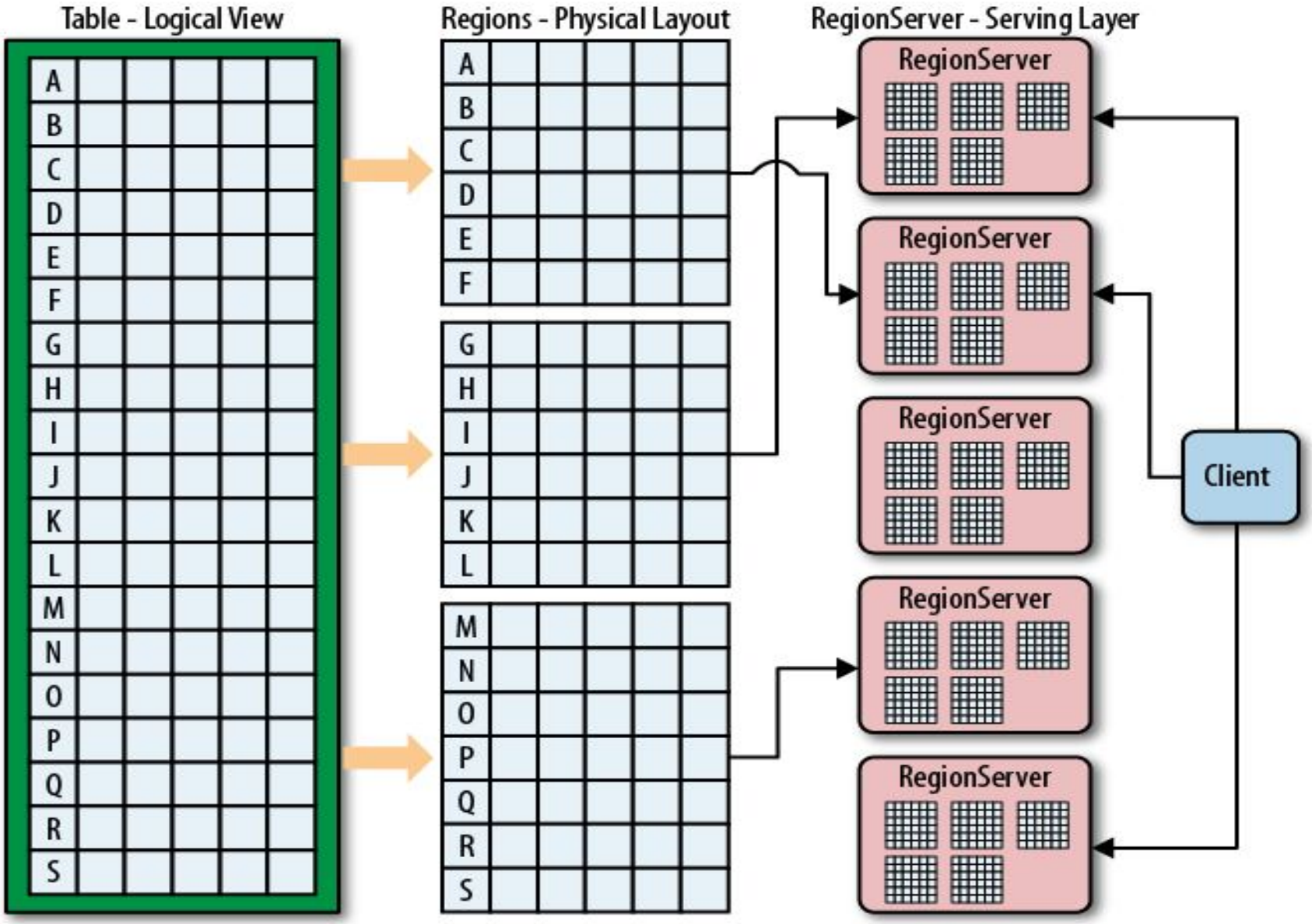
Inserting the First Row

- **put** command, you can insert rows into a table.;

- put ’<table name>’,’Key’,’<colfamily:colname>’,’<value>’

- put 'emp','1','personal data:name','raju‘
- put 'emp','1','personal data:city',‘Pune‘
- put 'emp','1','professional data:designation','manager‘
- put 'emp','1','professional data:salary','50000' 0

Logical and physical layout of rows within regions



Hbase Compression

- ***Compression***

HBase has pluggable compression algorithm support that allows you to choose the best compression or none for the data stored in a particular column family. The possible

Supported compression algorithms

Value	Description
NONE	Disables compression (default)
GZ	Uses the Java-supplied or native GZip compression
LZO	Enables LZO compression; must be installed separately
SNAPPY	Enables Snappy compression; binaries must be installed separately

Updating Data using HBase Shell

- You can update an existing cell value using the **put** command. To do so, just follow the same syntax and mention your new value as shown b

- put ‘table name’,’row ’,'Column family:column name',’new value’ below.

- put 'emp','row1','personal:city','Delhi‘

Reading Data

- get ’<table name>’,’row1’
- get 'emp', '1‘

Reading a Specific Column

- hbase> get 'table name', ‘rowid’, {COLUMN ⇒ ‘column family:column name ’}

- get 'emp', 'row1', {COLUMN ⇒ 'personal:name'}

- Deleting a Specific Cell in a Table
- delete ‘<table name>’, ‘<row>’, ‘<column name >’, ‘<time stamp>’

Connect from JAVA

```
public class CreateTable {

   public static void main(String[] args) throws IOException {

      // Instantiating configuration class
      Configuration con = HBaseConfiguration.create();

      // Instantiating HbaseAdmin class
      HBaseAdmin admin = new HBaseAdmin(con);

      // Instantiating table descriptor class
      HTableDescriptor tableDescriptor = new
      TableDescriptor(TableName.valueOf("emp"));

      // Adding column families to table descriptor
      tableDescriptor.addFamily(new HColumnDescriptor("personal"));
      tableDescriptor.addFamily(new HColumnDescriptor("professional"));

      // Execute the table through admin
      admin.createTable(tableDescriptor);
      System.out.println(" Table created ");
   }
}
```

```
public class InsertData{

   public static void main(String[] args) throws IOException {

      // Instantiating Configuration class
      Configuration config = HBaseConfiguration.create();

      // Instantiating HTable class
      HTable hTable = new HTable(config, "emp");

      // Instantiating Put class
      // accepts a row name.
      Put p = new Put(Bytes.toBytes("row1"));

      // adding values using add() method
      // accepts column family name, qualifier/row name ,value
      p.add(Bytes.toBytes("personal"),
      Bytes.toBytes("name"),Bytes.toBytes("raju"));

      p.add(Bytes.toBytes("personal"),
      Bytes.toBytes("city"),Bytes.toBytes("hyderabad"));

      p.add(Bytes.toBytes("professional"),Bytes.toBytes("designation"),
      Bytes.toBytes("manager"));

      p.add(Bytes.toBytes("professional"),Bytes.toBytes("salary"),
      Bytes.toBytes("50000"));

      // Saving the put Instance to the HTable.
      hTable.put(p);
      System.out.println("data inserted");

      // closing HTable
      hTable.close();
   }
}
```

HIVE Hbase Integration

Create HBase table

```
create 'hivehbase', 'ratings'
put 'hivehbase', 'row1', 'ratings:userid', 'user1'
put 'hivehbase', 'row1', 'ratings:bookid', 'book1'
put 'hivehbase', 'row1', 'ratings:rating', '1'

put 'hivehbase', 'row2', 'ratings:userid', 'user2'
put 'hivehbase', 'row2', 'ratings:bookid', 'book1'
put 'hivehbase', 'row2', 'ratings:rating', '3'

put 'hivehbase', 'row3', 'ratings:userid', 'user2'
put 'hivehbase', 'row3', 'ratings:bookid', 'book2'
put 'hivehbase', 'row3', 'ratings:rating', '3'

put 'hivehbase', 'row4', 'ratings:userid', 'user2'
put 'hivehbase', 'row4', 'ratings:bookid', 'book4'
put 'hivehbase', 'row4', 'ratings:rating', '1'
```

Check table data by using: `scan 'tablename'`.

Provide necessary jars to Hive

Create a folder named *auxlib* in Hive root directory and put the following jars in it. The jars can be found in HBase lib directory. Hive-HBase handler is present in Hive lib directory.

- Guava
- Hive-Hbase handler
- HBase
- Zookeeper

```
CREATE EXTERNAL TABLE hbasehive_table
(key string, userid string,bookid string,rating int)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES
("hbase.columns.mapping" = ":key,ratings:userid,ratings:bookid,ratings:rating")
TBLPROPERTIES ("hbase.table.name" = "hivehbase");
```

Note : the first column must be the key column which would also be same as the HBase's key column. Also if you have some numeric data columns you can use the `ratings:rating#b` in the HBase column mappings.

Querying HBase via Hive

```
hive> select * from hbasehive_table;
OK
row1    user1   book1   1
row2    user2   book1   3
row3    user2   book2   3
row4    user2   book4   1
Time taken: 0.254 seconds, Fetched: 4 row(s)
```

Thank You!