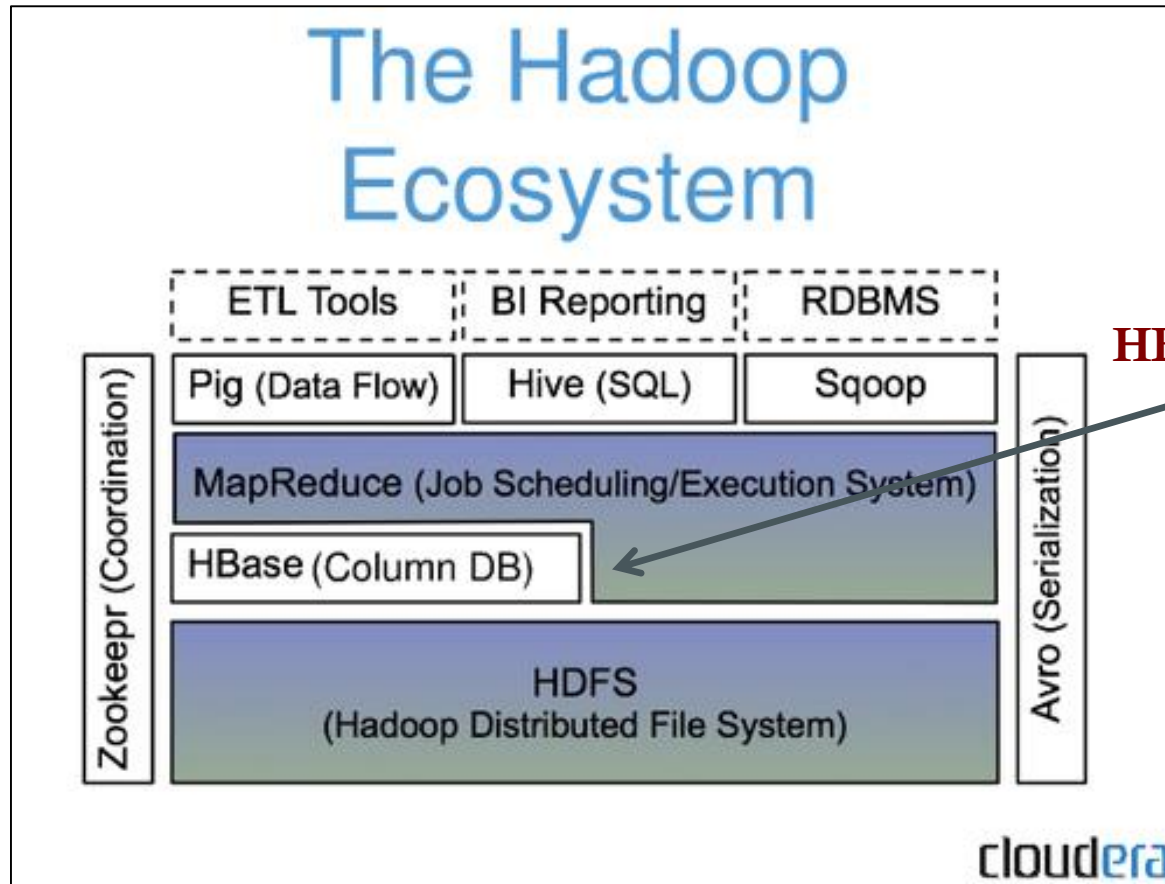# SmartR.in

HBase

By

Dineshkumar S

HBase: Overview

- **HBase is a distributed column-oriented data store built on top of HDFS**

- **HBase is an Apache open source project whose goal is to provide storage for the Hadoop Distributed Computing**

- **Data is logically organized into tables, rows and columns**

HBase: Part of Hadoop's Ecosystem

The Hadoop Ecosystem

Zookeepr (Coordination)	ETL Tools	BI Reporting	RDBMS	Avro (Serialization)
	Pig (Data Flow)	Hive (SQL)	Sqoop	
	MapReduce (Job Scheduling/Execution System)			
	HBase (Column DB)			
	HDFS (Hadoop Distributed File System)			

cloudera

HBase is built on top of HDFS

HBase files are internally stored in HDFS

HBase vs. HDFS

- Both are distributed systems that scale to hundreds or thousands of nodes

- **_HDFS_** is good for batch processing (scans over big files)
  - Not good for record lookup
  - Not good for incremental addition of small batches
  - Not good for updates

HBase vs. HDFS (Cont'd)

- ***HBase*** is designed to efficiently address the above points
  - Fast record lookup
  - Support for record-level insertion
  - Support for updates (not in place)

- HBase updates are done by creating new versions of values

HBase vs. HDFS (Cont'd)

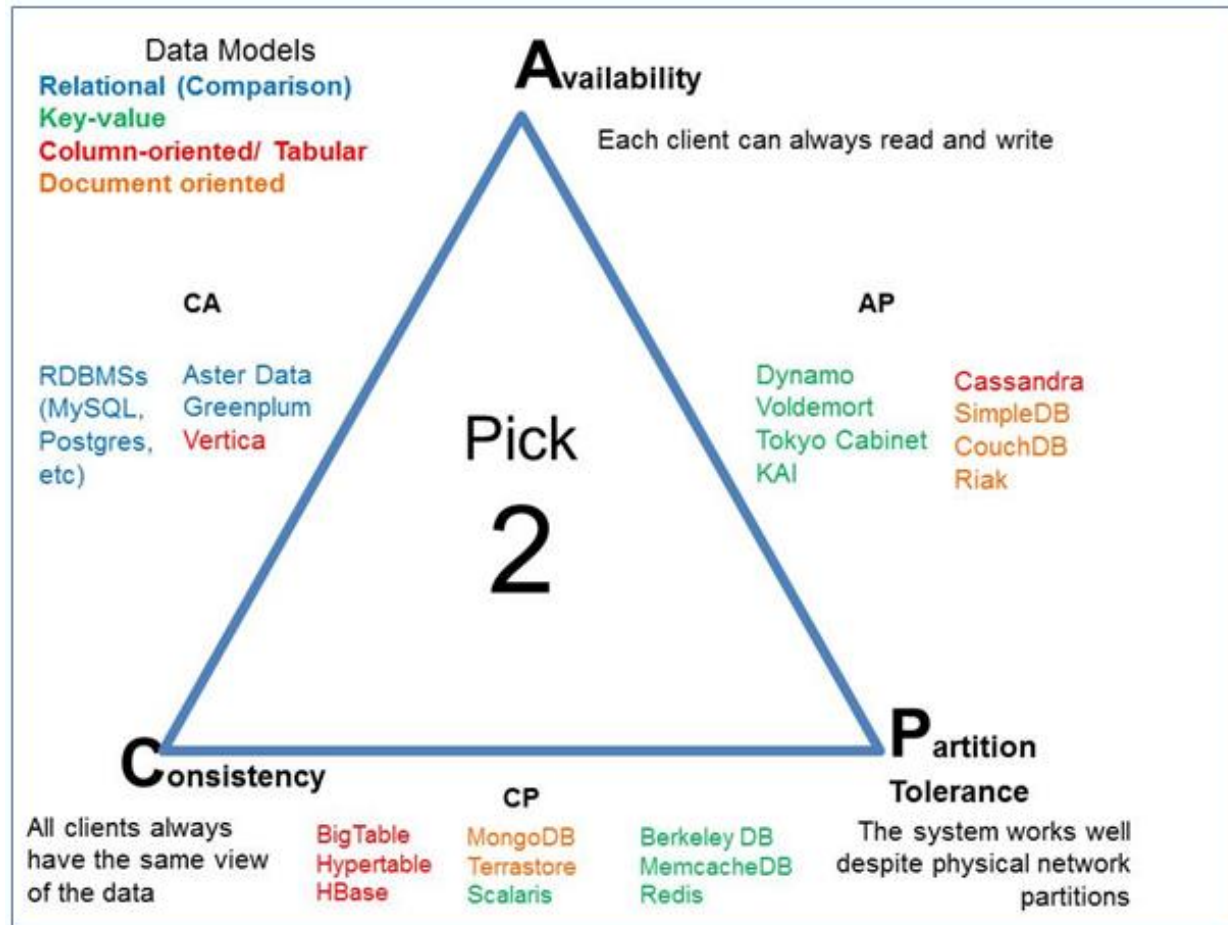|	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

If application has neither random reads or writes ➜ Stick to HDFS

HBase vs. RDBMS

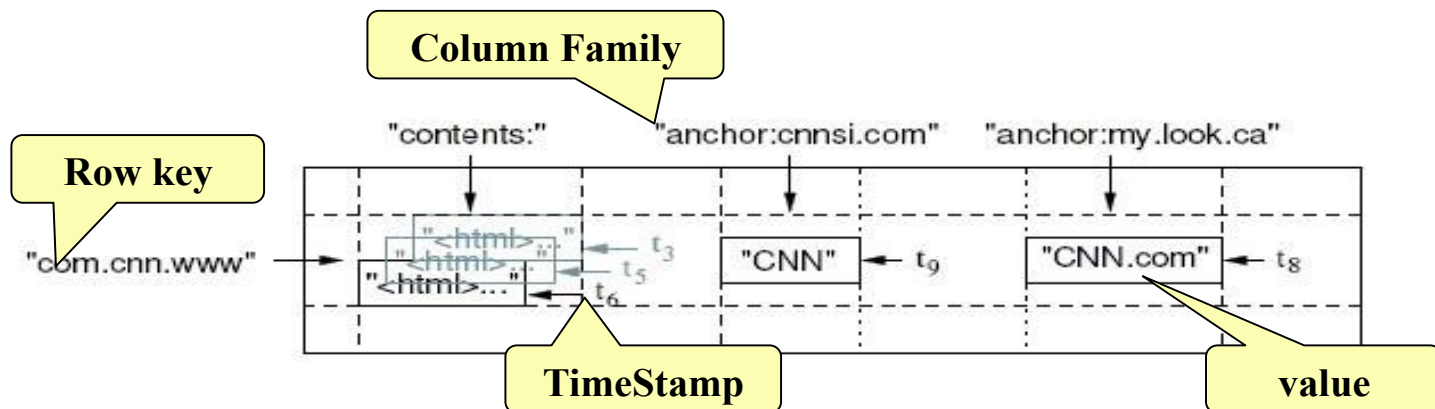|	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorization	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of queries/second

HBase Data Model

CAP Theorem

Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented

Availability

Each client can always read and write

CA

RDBMSs   Aster Data
(MySQL,   Greenplum
Postgres,   Vertica
etc)

AP

Dynamo   Cassandra
Voldemort   SimpleDB
Tokyo Cabinet CouchDB
KAI   Riak

Pick
2

Consistency

All clients always
have the same view
of the data

CP

BigTable   MongoDB   Berkeley DB
Hypertable Terrastore MemcacheDB
HBase   Scalaris   Redis

Partition
Tolerance

The system works well
despite physical network
partitions

HBase Data Model

- **HBase is based on Google's Big table model**
  - **Key-Value pairs**

HBase Logical View

Implicit PRIMARY KEY in RDBMS terms

Data is all `byte[]` in HBase

Different types of data separated into different "column families"

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

Different rows may have different sets of columns(table is *sparse*)

A single cell might have different values at different timestamps

Useful for *-To-Many mappings

HBase: Keys and Column Families

Each record is divided into *Column Families*

Each row has a *Key*

PERSON TABLE

row key	personal_data		demographic		...
PersonID 1 2 3 ... 500,000,000	**Name** H. Houdini D. Copper Merlin ... F. Cadillac	**Address** Budapest, Hungary New Jersey, USA Stonehenge, England ... Nevada, USA	**BirthDate** 1926-10-31 1956-09-16 1136-12-03 ... 1964-01-07	**Gender** M M F ... M	...

Figure 2. Census Data in Column Families

Each column family consists of one or more *Columns*

- **Key**
  - Byte array
  - Serves as the primary key for the table
  - Indexed far fast lookup

- **Column Family**
  - Has a name (string)
  - Contains one or more related columns

- **Column**
  - Belongs to one column family
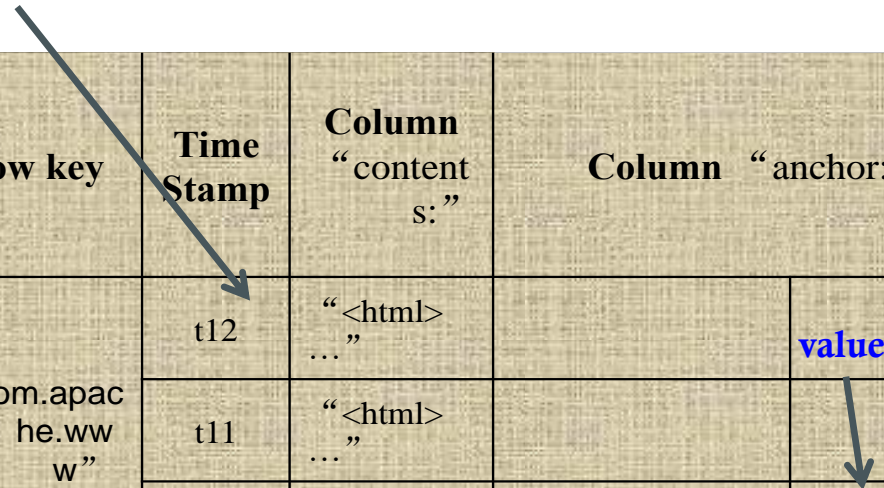 - Included inside the row
    - *familyName:columnName*

Column family named "Contents"

Column family named "anchor"

Column named "apache.com"

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
"com.apache.www"	t12	"<html>…"		
	t11	"<html>…"		
	t10		"anchor:apache.com"	"APACHE"
	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
"com.cnn.www"	t6	"<html>…"		
	t5	"<html>…"		
	t3	"<html>…"		

13

Version number for each row

- **Version Number**
  - Unique within each key
  - By default → System's timestamp
  - Data type is Long

- **Value (Cell)**
  - Byte array

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
"com.apache.www"	t12	"<html>…"		
	t11	"<html>…"		value
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.www"	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html>…"		
	t5	"<html>…"		
	t3	"<html>…"		

14

Notes on Data Model

- HBase schema consists of several ***Tables***

- Each table consists of a set of ***Column Families***
  - Columns are not part of the schema

- HBase has ***Dynamic Columns***
  - Because column names are encoded inside the cells
  - Different cells can have different columns

"Roles" column family has different columns in different cells

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

Notes on Data Model (Cont'd)

- The **_version number_** can be user-supplied
  - Even does not have to be inserted in increasing order
  - Version number are unique within each key

- Table can be very sparse
  - Many cells are empty

- **_Keys_** are indexed as the primary key

Has two columns
[cnnsi.com & my.look.ca]

Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"
"com.cnn.www"	t6	contents:html = "\<html\>..."	
"com.cnn.www"	t5	contents:html = "\<html\>..."	
"com.cnn.www"	t3	contents:html = "\<html\>..."	

HBase Physical Model

HBase Physical Model

- Each column family is stored in a separate file (called ***HTables***)

- Key & Version numbers are replicated with each column family

- Empty cells are not stored

HBase maintains a multi-level index on values:
<key, column family, column name, timestamp>

Table 5.3. ColumnFamily `contents`

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	`contents:html = "<html>..."`
"com.cnn.www"	t5	`contents:html = "<html>..."`
"com.cnn.www"	t3	`contents:html = "<html>..."`

Table 5.2. ColumnFamily `anchor`

Row Key	Time Stamp	Column Family `anchor`
"com.cnn.www"	t9	`anchor:cnnsi.com = "CNN"`
"com.cnn.www"	t8	`anchor:my.look.ca = "CNN.com"`

Example

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' }
	roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' }
	roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

`info` Column Family

Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

`roles` Column Family

Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

Sorted on disk by Row key, Col key, descending timestamp

Milliseconds since unix epoch

cloudera

Column Families

- Different sets of columns may have different properties and access patterns

- Configurable by column family:

  - Compression (none, gzip, LZO)

  - Version retention policies

  - Cache priority

- CFs stored separately on disk: access one without wasting IO on the other.

HBase Regions

- Each HTable (column family) is partitioned horizontally into *regions*

  - Regions are counterpart to HDFS blocks

Table 5.3. ColumnFamily contents

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	contents:html = "<html>..."
"com.cnn.www"	t5	contents:html = "<html>..."
"com.cnn.www"	t3	contents:html = "<html>..."

Each will be one region

HBase Architecture

Three Major Components

- The HBaseMaster
  - One master

- The HRegionServer
  - Many region servers

- The HBase client
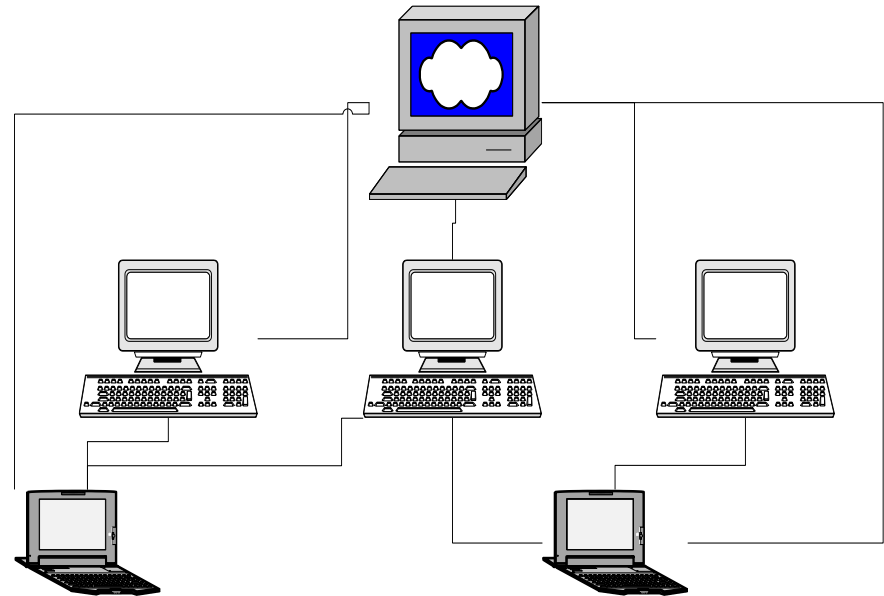
HBase Components

- **Region**
  - A subset of a table's rows, like horizontal range partitioning
  - Automatically done

- **RegionServer (many slaves)**
  - Manages data regions
  - Serves data for reads and writes (*using a log*)

- **Master**
  - Responsible for coordinating the slaves
  - Assigns regions, detects failures
  - Admin functions

Big Picture
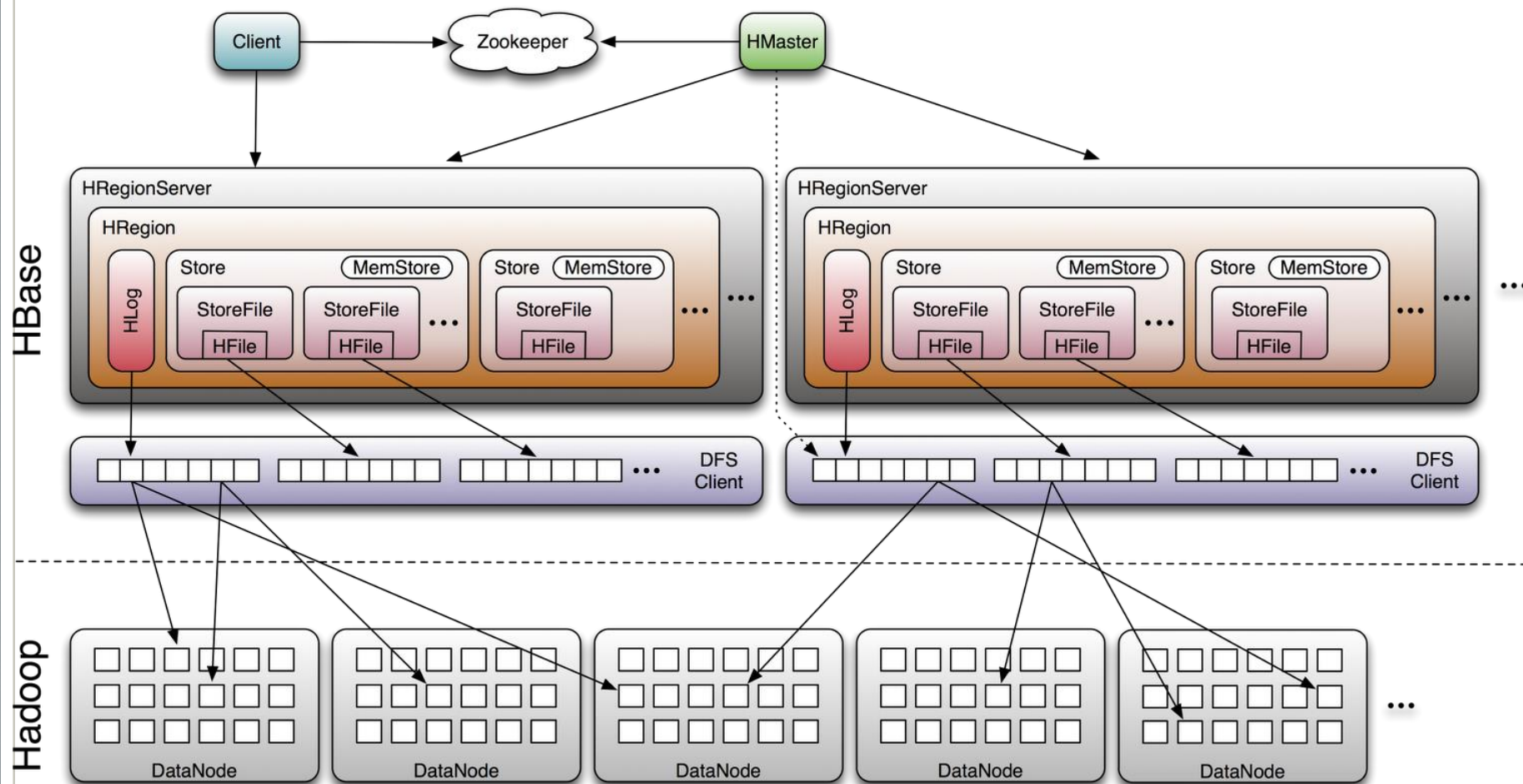
Hbase Assist by ZooKeeper

- HBase depends on ZooKeeper

- By default HBase manages the ZooKeeper instance
  - E.g., starts and stops ZooKeeper

- HMaster and HRegionServers register themselves with ZooKeeper
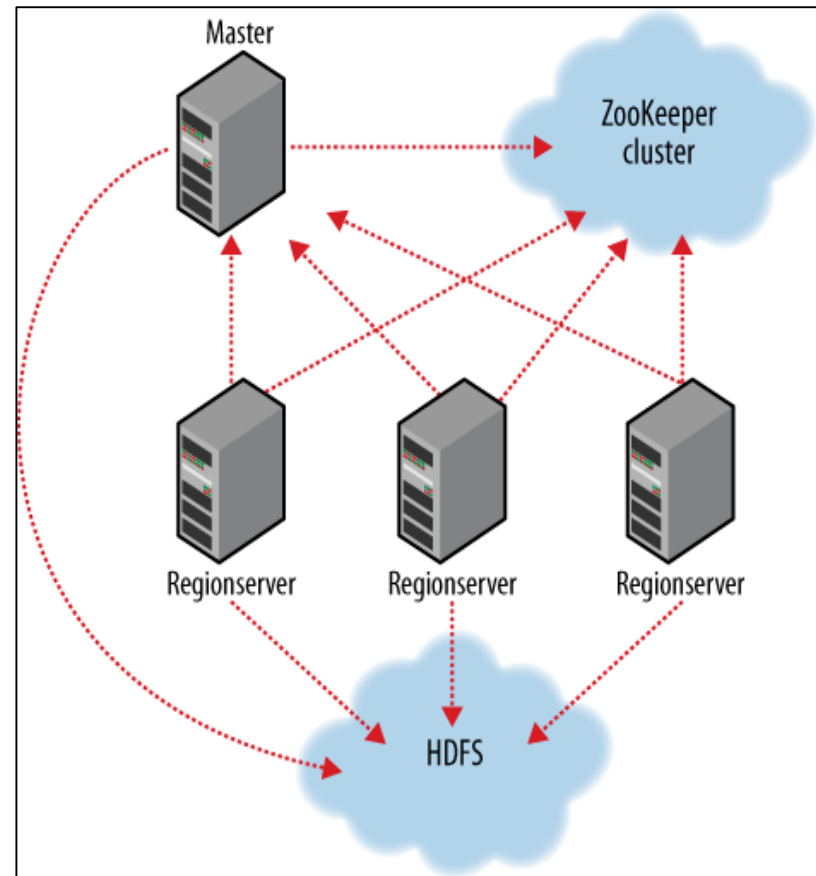
Creating a Table

```
HBaseAdmin admin= new HBaseAdmin(config);

HColumnDescriptor []column;

column= new HColumnDescriptor[2];

column[0]=new HColumnDescriptor("columnFamily1:");

column[1]=new HColumnDescriptor("columnFamily2:");

HTableDescriptor desc= new HTableDescriptor(Bytes.toBytes("MyTable"));

desc.addFamily(column[0]);

desc.addFamily(column[1]);

admin.createTable(desc);
```

Operations On Regions: **Get()**

- Given a key → return corresponding record

- For each value return the highest version

```
Get get = new Get(Bytes.toBytes("row1"));
Result r = htable.get(get);
5.8.1.2. Default Get Example.ue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
```

- Can control the number of versions you want

```
Get get = new Get(Bytes.toBytes("row1"));
get.setMaxVersions(3);  // will return last 3 versions of row
Result r = htable.get(get);
byte[] b = r.getValue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
List<KeyValue> kv = r.getColumn(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns all versions of
```

Operations On Regions: **Scan()**

```
HTable htable = ...        // instantiate HTable

Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("cf"),Bytes.toBytes("attr"));
scan.setStartRow( Bytes.toBytes("row"));                    // start key is inclusive
scan.setStopRow( Bytes.toBytes("row" +  (char)0));  // stop key is exclusive
ResultScanner rs = htable.getScanner(scan);
try {
  for (Result r = rs.next(); r != null; r = rs.next()) {
  // process result...
} finally {
  rs.close();  // always close the ResultScanner!
}
```

Get()

Select value from table where key='com.apache.www' AND label='anchor:apache.com'

Row key	Time Stamp	Column "anchor:"	
"com.apache.www"	t12		
	t11		
	t10	"anchor:apache.com"	**"APACHE"**
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"
	t6		
	t5		
	t3		

Scan()

Select value from table
where anchor='cnnsi.com'

Row key	Time Stamp	Column "anchor:"	
"com.apache.www"	t12		
	t11		
	t10	"anchor:apache.com"	"APACHE"
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"
	t6		
	t5		
	t3		

Operations On Regions: **Put()**

- Insert a new record (with a new key), Or

- Insert a record for an existing key

Implicit version number (timestamp)

```
Put put = new Put(Bytes.toBytes(row));
put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), Bytes.toBytes( data));
htable.put(put);
```

Explicit version number

```
Put put = new Put( Bytes.toBytes(row));
long explicitTimeInMs = 555;  // just an example
put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), explicitTimeInMs, Bytes.toBytes(data));
htable.put(put);
```

Operations On Regions: **Delete()**

- Marking table cells as deleted

- **Multiple levels**
  - Can mark an entire column family as deleted
  - Can make all column families of a given row as deleted

- **All operations are logged by the RegionServers**
- **The log is flushed periodically**

HBase: **Joins**

- HBase does not support joins

- Can be done in the application layer
  - Using scan() and get() operations

Altering a Table

```
Configuration config = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
String table = "myTable";

admin.disableTable(table);

HColumnDescriptor cf1 = ...;
admin.addColumn(table, cf1);        // adding new ColumnFamily
HColumnDescriptor cf2 = ...;
admin.modifyColumn(table, cf2);     // modifying existing ColumnFamily

admin.enableTable(table);
```
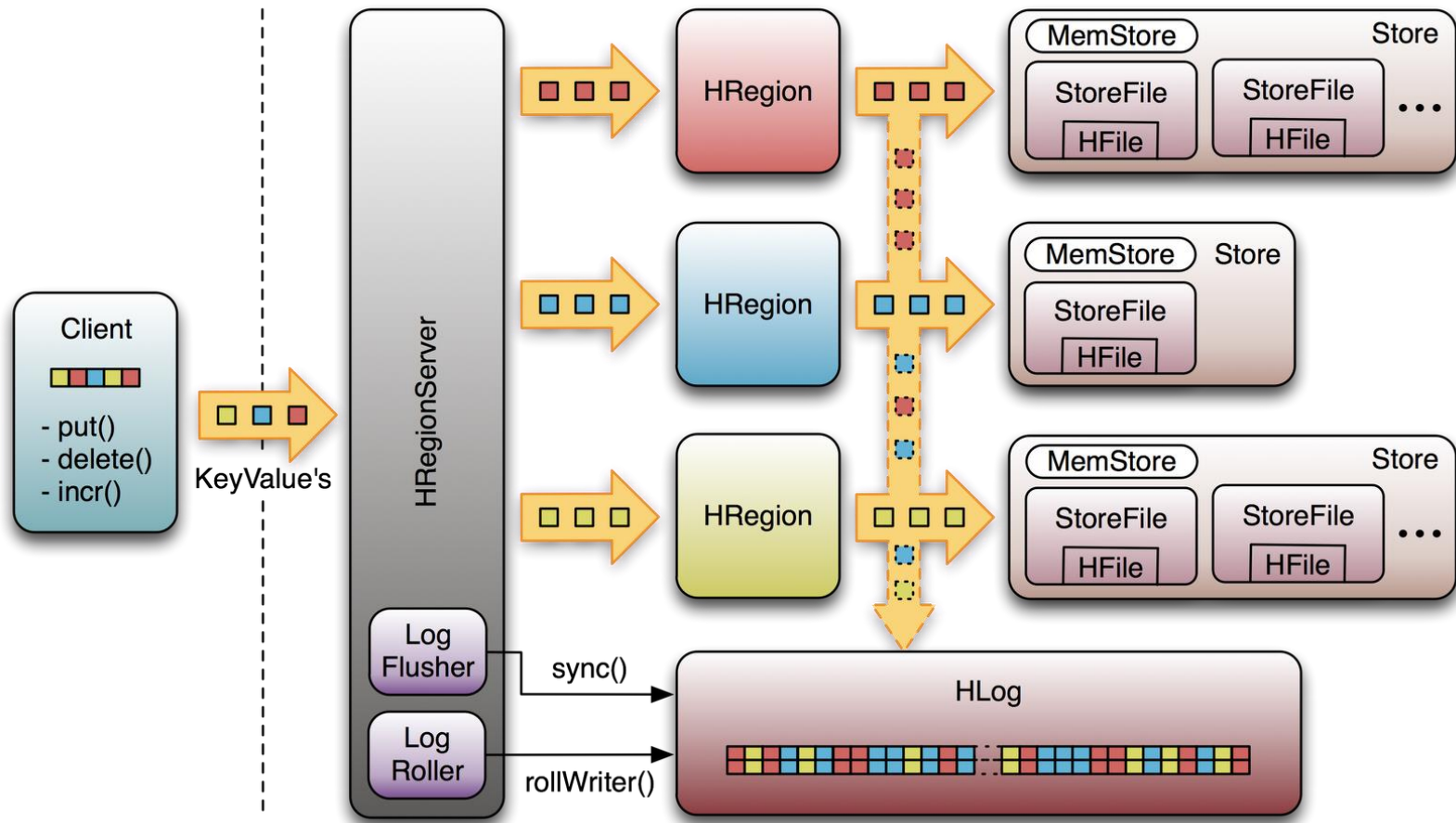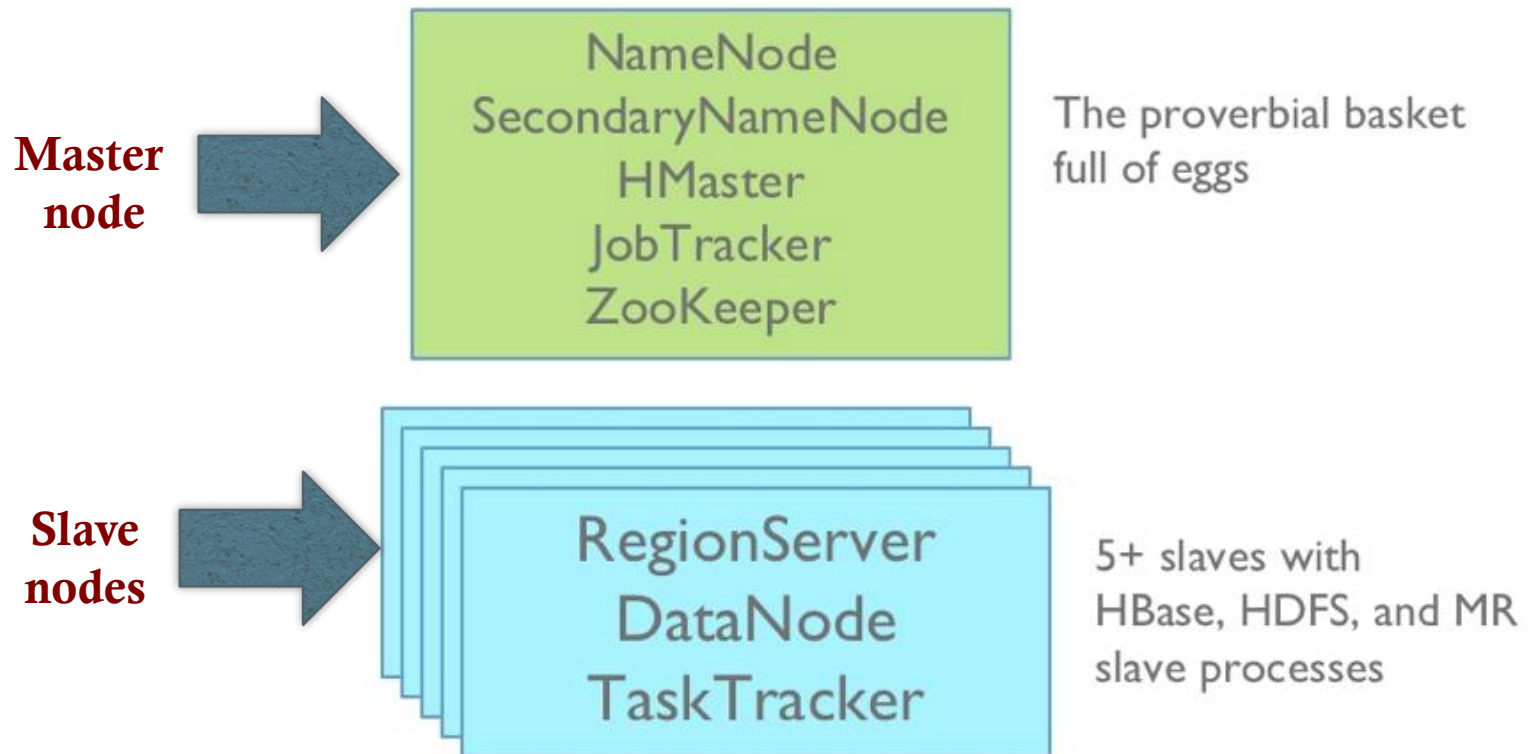
Disable the table before changing the schema

6.1. Schema Creation

Logging Operations

HBase Deployment

Master node → NameNode / SecondaryNameNode / HMaster / JobTracker / ZooKeeper — The proverbial basket full of eggs

Slave nodes → RegionServer / DataNode / TaskTracker — 5+ slaves with HBase, HDFS, and MR slave processes

When to use HBase

- You need random write, random read, or both (*but not neither*)

- You need to do many thousands of operations per second on multiple TB of data

- Your access patterns are well-known and simple