

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.
- HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data.
- HBase has C + P properties
  - Consistency and Partition aware

# HBase vs HDFS

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

# HBase vs RDBMS

- HBase is schema-less, doesn't have ACID properties  
RDBMS has schema and follows ACID properties
- HBase is non-transactional  
RDBMS is transactional
- HBase is good for semi-structured and structured data  
RDBMS is for structured data

# Storage Mechanism

- Column-oriented
- Table -> collection of rows.
- Tables are sorted by rowid
- Table schema only defines column families. Each family identified by prefix i.e. UTD:columnName.
- Columns can be added to column families dynamically.
- Each cell has a timestamp.

# HBase structure




[illegible]




# HBase structure

## COLUMN FAMILIES

Row key	personal data		professional data	
empid	name	city	designation	salary
1				
2				
3				

Increasing row key

	Column family <b>contents</b> contents:image	Column family <b>info</b> info:format    info:geo	
000001		jpeg	51.5,-0.1
000002		tiff	
000003		jpeg	51.8,-3.1

Versions
 Cells



# HBase Data Model Components

Component	Description
<b>Table</b>	Data organized into tables; comprised rows
<b>Row key</b>	Data stored in rows; Rows identified by Rowkeys
<b>Column family</b>	Columns grouped into families
<b>Column Qualifier</b>	Identifies the column
<b>Cell</b>	Combination of the row key, column family, column, timestamp; contains the value
<b>Version</b>	Values within cell versioned by version number → timestamp

# HBase Data Model – Column Family

Column Family can contain an arbitrary number of columns

Column Family 1

Column Family 2

Row Key	Address			Order			
	street	city	state	Date	ItemNo	Quantity	Price
smithj	Central Dr	Nashville	TN	2/2/15	10213A	1	109.99
spata	High Ave	Columbus	OH	1/31/14	23401V	24	21.21
turnerb	Cedar St	Seattle	WA	7/8/14	10938A	15	1.54

# HBase Data Model – Cells

- Data is stored in **Key Value** format
- Value for each **cell** is specified by complete coordinates:  
(Row key, ColumnFamily, Column Qualifier, timestamp ) => Value
  - RowKey:CF:Col:Version:Value
  - smithj:data:city:1391813876369:nashville

Cell Coordinates= Key				Value
Row key	Column Family	Column Qualifier	Timestamp	Value
Smithj	Address	city	1391813876369	Nashville

## Question:

The cell in an HBase table is called KeyValue in HBase terms.  
The Key consists of:

- a) Row key, column family name, cell
- b) Column family name, column name, cell, version
- c) Row key, column family name, column name, version
- d) Row key, column name, version, value

## Question:

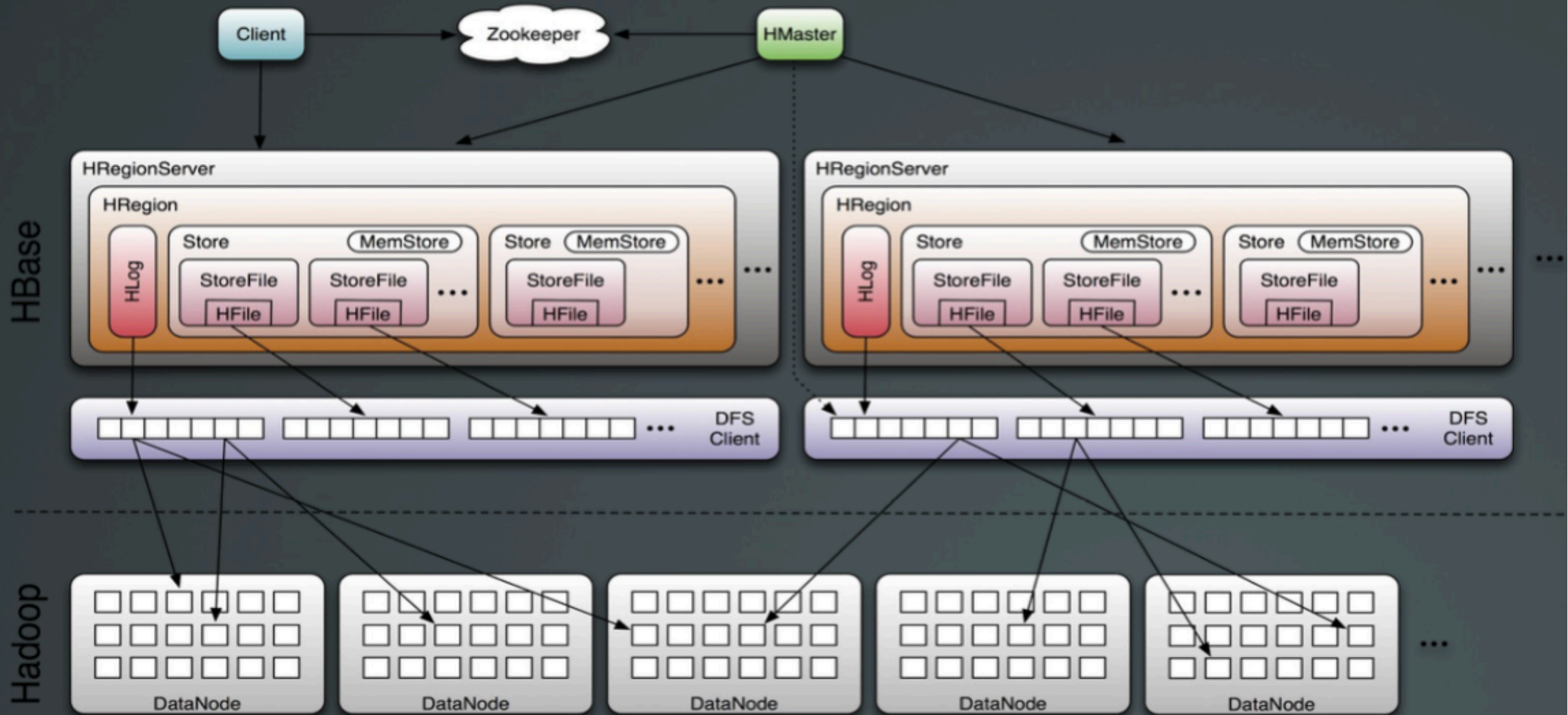
The cell in an HBase table is called KeyValue in HBase terms.  
The Key consists of:

- a) Row key, column family name, cell
- b) Column family name, column name, cell, version
- c) Row key, column family name, column name, version ✓
- d) Row key, column name, version, value

# HBase Architecture

- HBase tables are divided into regions.
- Regions represent a subset of the table's rows.
- Each region is served by a RegionServer.
- RegionServer serves data for reads and writes.
- The RegionServers are coordinated by a master.
- Master also assigns regions, detects failures of RegionServers

# Hbase Architecture



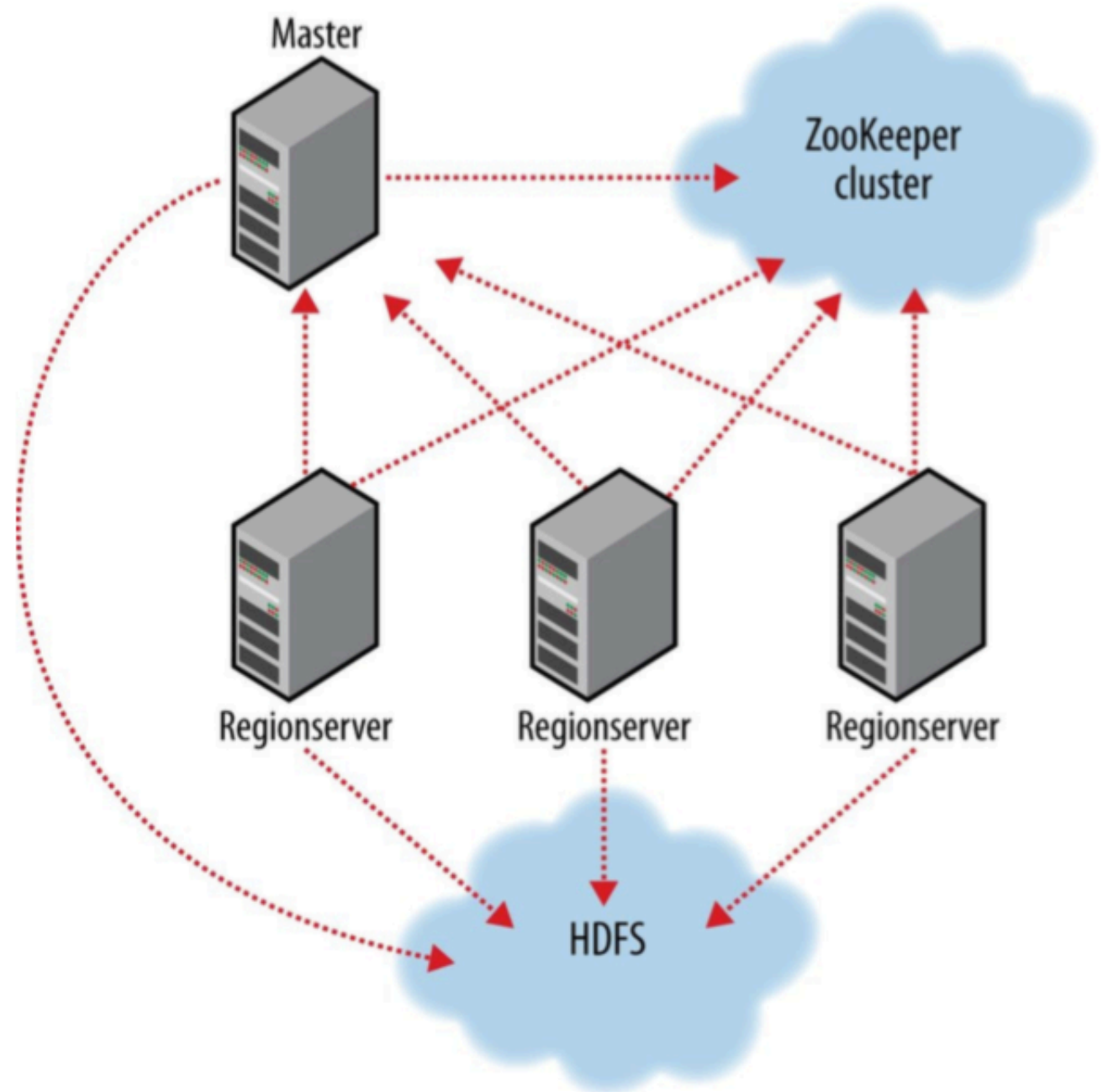
# HBase Architecture

- Initially, a table comprises a single region,
- As the region grows it eventually crosses a configurable size threshold, at which point it splits at a row boundary into two new regions of approximately equal size.
- Regions are the units that get distributed over an HBase cluster
- Internally, HBase keeps a special catalog table named hbase:meta, within which it maintains the current list, state, and locations of all user-space regions afloat on the cluster.



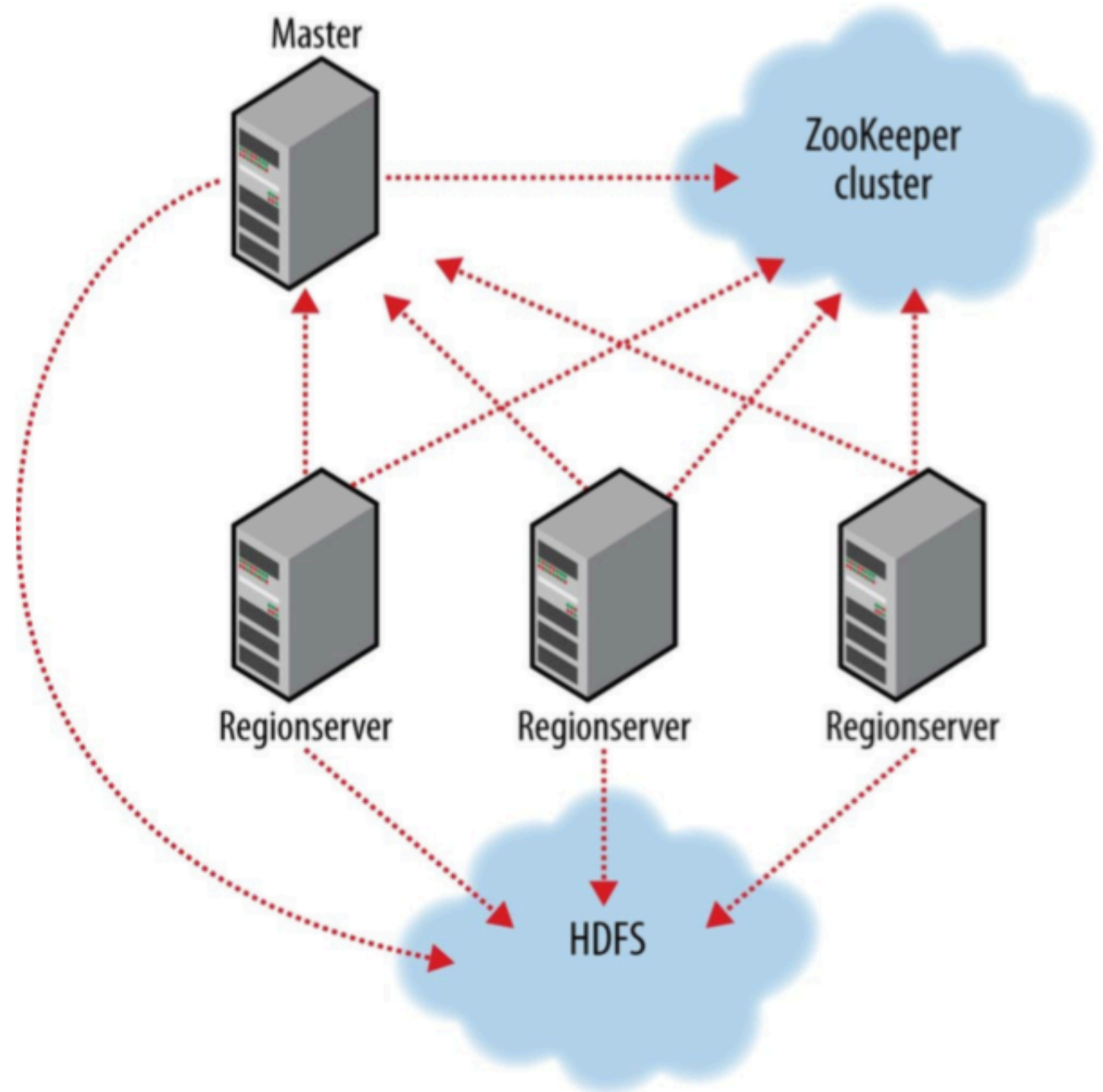
# HBase Architecture

- HBase made up of an HBase master node orchestrating a cluster of one or more regionserver workers
- **Master's roles:**
  - bootstrapping install
  - assigning regions to regionserver
  - recovering from regionserver failures



# HBase Architecture

- HBase made up of an HBase master node orchestrating a cluster of one or more regionserver workers
- **RegionServer's roles:**
  - manages region splits - informing master about changes
- HBase depends on ZooKeeper, and by default it manages a ZooKeeper instance as the authority on cluster state



# Logical vs Physical

Logical Model

RowKey	Address		Order	
	Street	City	Date	ItemNo
smithj	Central Dr	Nashville	2/2/15	10213A
spata	High Ave	Columbus	1/31/14	23401V
turnerb	Cedar St.	Seattle	7/8/14	10938A

Key			Value
Row Key	CF:Col	version	value
smithj	Address:street	1	Central Dr

Physical Storage

Key			Value
Row Key	CF:Col	version	value
smithj	OrderDate	1	2/2/15

Physical Storage

# Table Physical View

Physically data is stored per Column family as a sorted map

- Ordered by **row key, column qualifier** in **ascending** order
- Ordered by **timestamp** in **descending** order

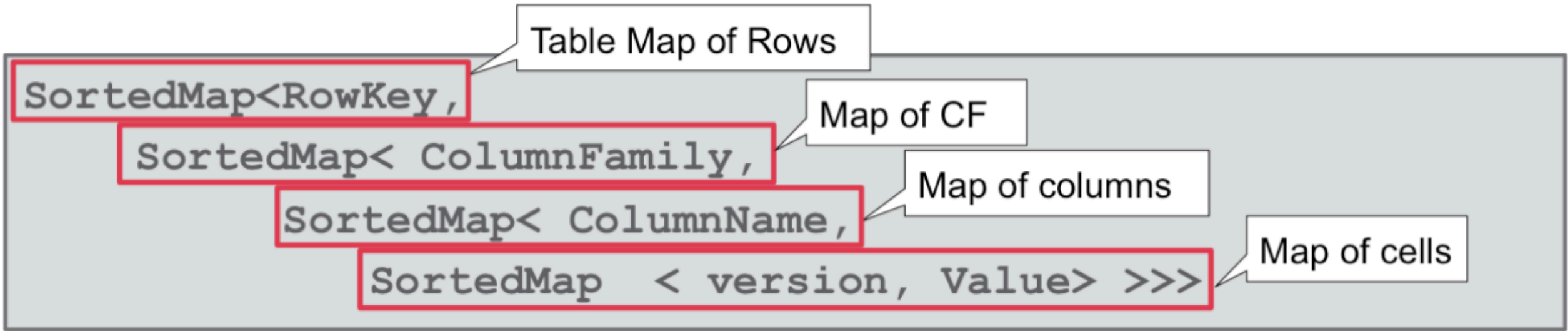
Sorted by Row key and Column

Row key	Column qualifier	Cell value	Timestamp (long)
Row1	CF1:colA	value3	time7
Row1	CF1:colA	value2	time5
Row1	CF1:colA	value1	time1
Row10	CF1:colA	value1	time4
Row 10	CF1:colB	value1	time4

Sorted in descending order

# HBase stored as sorted map of maps

In Java this is the table:



Key	CF:Col	version	value
smithj	Address:street	v2	Main St
smithj	Address:street	v1	Central Dr
spata	Address:street	v1	High Ave
turnerb	Address:street	v1	Cedar St

Key	CF:Col	version	value
smithj	Order:Date	v1	2/2/15
spata	Order:Date	v1	1/31/14
turnerb	Order:Date	v1	7/8/14

# Zookeeper

- To manage master election and server availability we use Zookeeper
- Sets up a cluster, provides distributed coordination primitives
- An excellent tool for building cluster management systems

# HBase Interface

- Java
- Thrift(Ruby,Php,Python,Perl,C++,...)
- Hbase Shell

# Basic Data Access Operations

## Basic data access operations (CRUD)

OPERATION	DESCRIPTION
put	Inserts data into rows (both add and update)
get	Accesses data from one row
scan	Accesses data from a range of rows
delete	Delete a row or range



# HBase Shell

```
hbase> create '/user/user01/Customer', {NAME => 'Address'} , {NAME => 'Order' }  
hbase> put '/user/user01/Customer', 'smithj', 'Address:street', 'Central Dr'  
hbase> put '/user/user01/Customer', 'smithj', 'Order:Date', '2/2/15'  
hbase> put '/user/user01/Customer', 'spata', 'Address:city', 'Columbus'  
hbase> put '/user/user01/Customer', 'spata', 'Order:Date', '1/31/14'
```

Row Key	Address			Order	
	street	city	state	Date	ItemNo
smithj	Central Dr	Nashville	TN	2/2/15	10213A
spata	High Ave	Columbus	OH	1/31/14	23401V
turnerb	Cedar St	Seattle	WA	7/8/14	10938A

# HBase Java

```
public class ExampleClient {  
  
    public static void main(String[] args) throws IOException {  
        Configuration config = HBaseConfiguration.create();  
        // Create table  
        HBaseAdmin admin = new HBaseAdmin(config);  
        try {  
            TableName tableName = TableName.valueOf("test");  
            HTableDescriptor htd = new HTableDescriptor(tableName);  
            HColumnDescriptor hcd = new HColumnDescriptor("data");  
            htd.addFamily(hcd);  
            admin.createTable(htd);  
            HTableDescriptor[] tables = admin.listTables();  
            if (tables.length != 1 &&  
                Bytes.equals(tableName.getName(), tables[0].getTableName().getName())) {  
                throw new IOException("Failed create of table");  
            }  
        }  
    }  
}
```

# Hbase shell

- `hbase(main):003:0> create 'test', 'cf'`
- 0 row(s) in 1.2200 seconds
- `hbase(main):004:0> put 'test', 'row1', 'cf:a', 'value1'`
- 0 row(s) in 0.0560 seconds
- `hbase(main):005:0> put 'test', 'row2', 'cf:b', 'value2'`
- 0 row(s) in 0.0370 seconds
- `hbase(main):006:0> put 'test', 'row3', 'cf:c', 'value3'`
- 0 row(s) in 0.0450 seconds

# Facebook application

- Realtime counters of URLs shared, links "liked", impressions generated
- 20 billion events/day (200K events/sec)
- ~30 sec latency from click to count
- Heavy use of incrementColumnValue API
- Tried MySQL, Cassandra, settled on Hbase

# When to use HBase

- You need random write, random read or both (but not neither)
- You need to do many thousands of operations per sec on multiple TB of data
- Your access patterns are simple