

HBase

Welcome to the world of NoSQL

- Introduction to HBase
- Why to use HBase
- What HBase is not
- Characteristics
- Data Model
- Architecture
- HBase vs HDFS
- HBase vs RDBMS



- HBase is an open source, non-relational, distributed database modeled after Google's BigTable and written in Java.
- Think of it as a sparse, consistent, distributed, multidimensional, sorted map:
 - labeled tables of rows
 - row consist of key-value cells:
 - (row key, column family, column, timestamp) -> value

- Storing large amounts of data (TB/PB)
- Storing unstructured or variable column data
- Big Data with random read and writes
- Horizontally scalable
- Strongly consistent reads and writes
- Simple Java API
- Integration with Map/Reduce framework

What HBase is not

- Not an SQL database
- Not relational
- No joins
- No fancy query language and no sophisticated query engine
- No transactions out-of-the box
- No secondary indices out-of-the box
- Not a drop-in replacement for your RDBMS

- Non-relational (NoSQL)
- Column-Oriented
- Multi-Dimensional
- High Availability
- High Performance
- Open Source

- HBase is a "NoSQL" database.
- "NoSQL" is a general term meaning that the database doesn't support SQL as its primary access language. There are many types of NoSQL databases:
 - **Key-Value Store** – It has a Big Hash Table of keys & values (Example- Riak, Amazon S3 (Dynamo))
 - **Document-based Store** - It stores documents made up of tagged elements. (Example- CouchDB)
 - **Column-based Store** - Each storage block contains data from only one column, (Example- HBase, Cassandra)
 - **Graph-based** - A network database that uses edges and nodes to represent and store data. (Example- Neo4J)

- Every row has a row key (analogous to a primary key)
- Rows are stored by row key for fast lookups
- A table may have 1 or more column families
- Common to have a small number of column families
- A column family can have number of columns
- Each row has a timestamp
- Multiple versions of a row can exist

- Table is a distributed sorted map
 - row key + column family + column + timestamp -> value

Row key	Column key	Timestamp	Cell
Row1	info:aaa	1273516197868	valueA
Row1	info:bbb	1273871824184	valueB
Row1	info:bbb	1273871823022	oldValueB
Row1	info:ccc	1273746289103	valueC
Row2	info:hello	1273878447049	i_am_a_value
Row3	info:	1273616297446	another_value

Sorted by Row key and Column

Column family

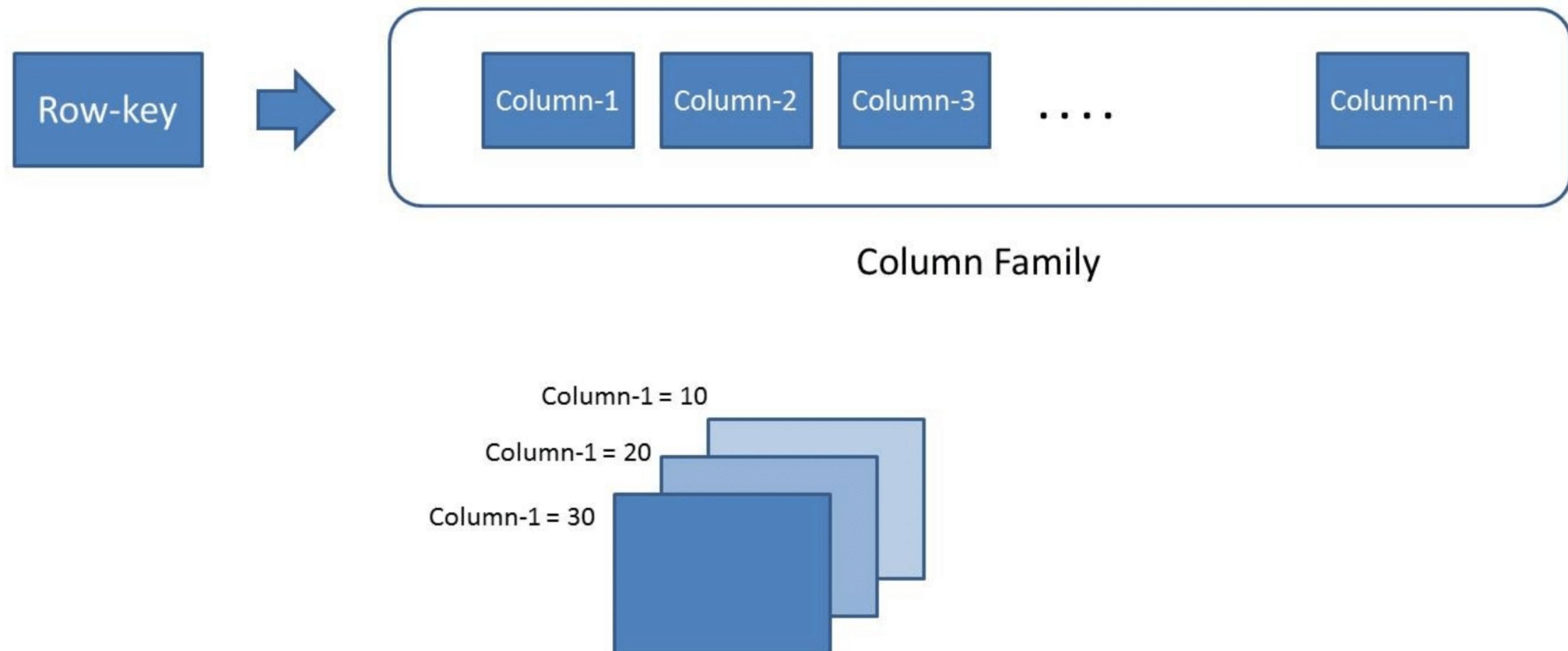
Column key can be empty

Timestamp is a long value

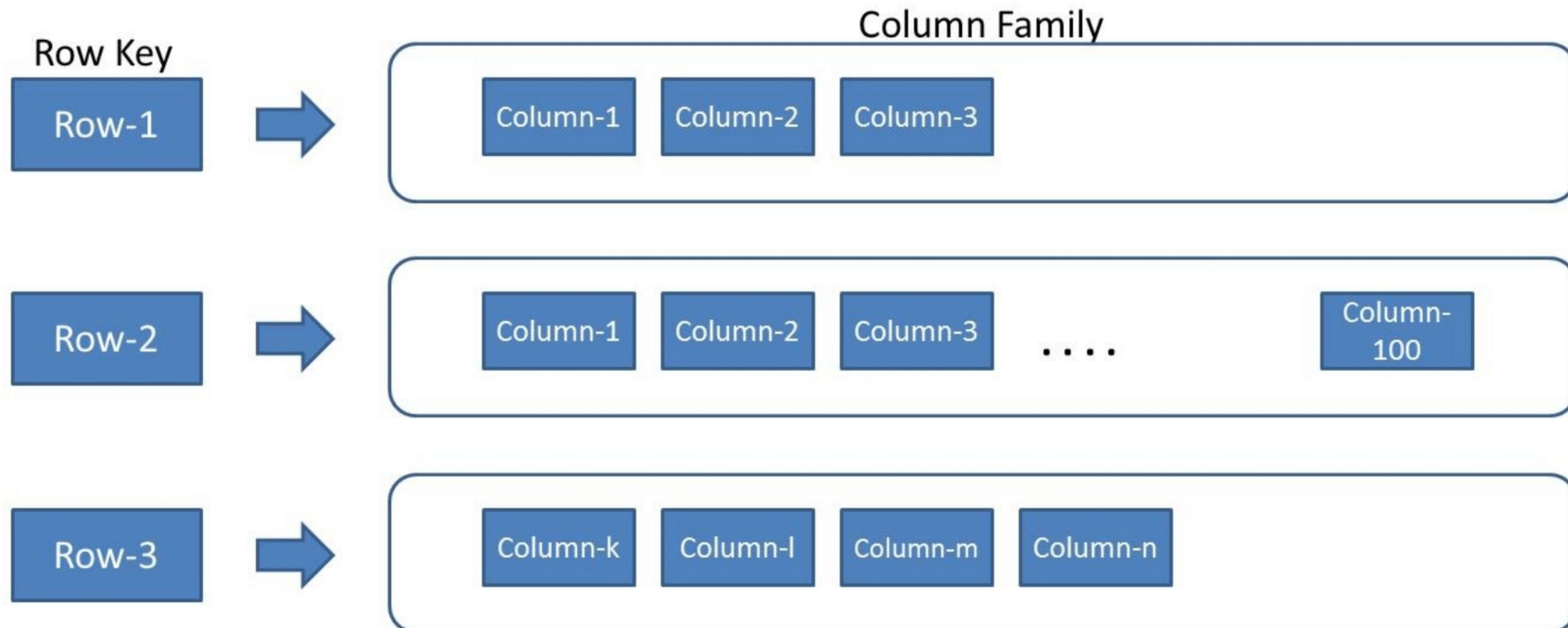
2 Versions of this row

- Stores multiple versions of columns
- With each version a timestamp is stored

Key	Column	Value	Timestamp	
rowA	Fam:foo	New value	1275340679713	Sorted in descending order
rowA	Fam:foo	Old value	1275091706190	
rowB	Fam:foo	Some value	1274999316683	

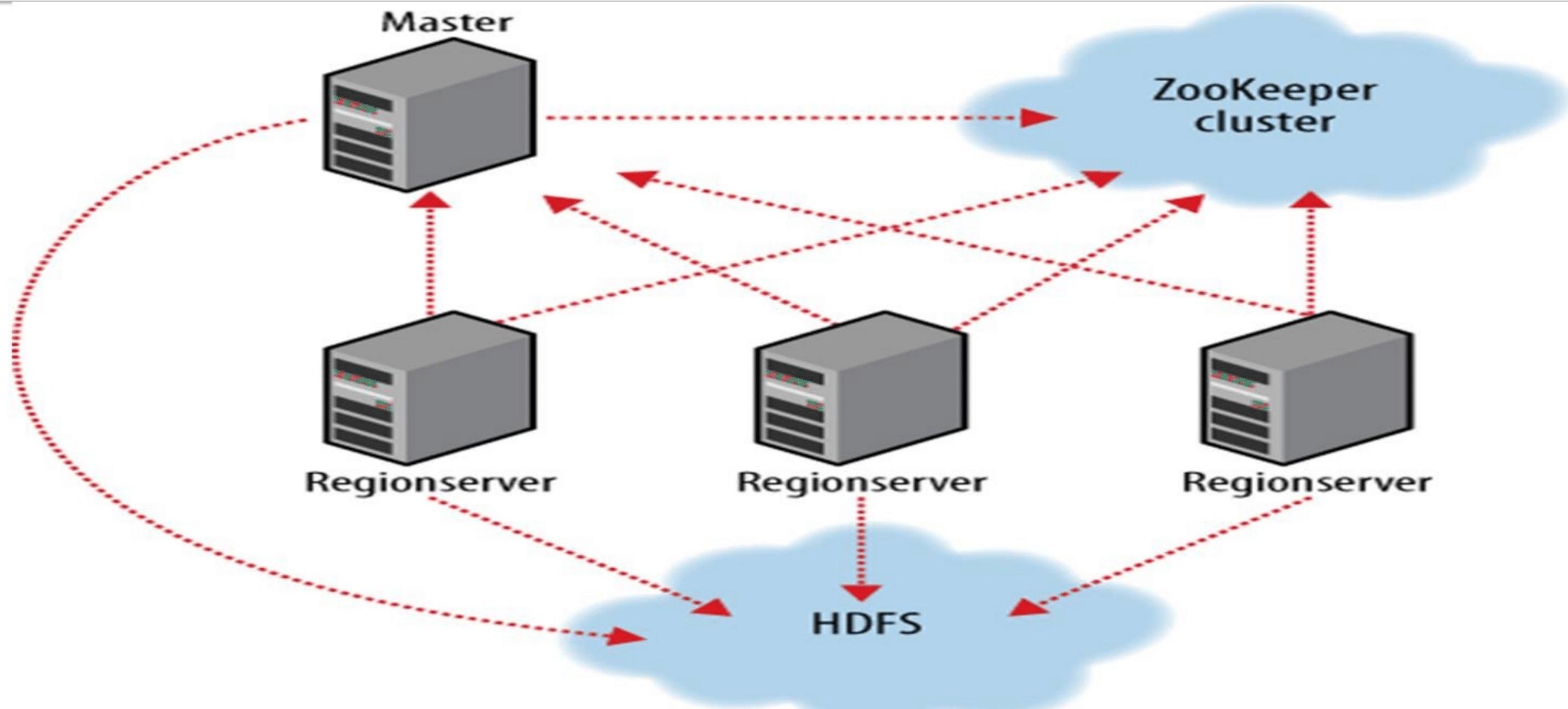


Create Table specify: Table-Name, Column-Family-Name



- HBase modeled with an HBase *master* node orchestrating a cluster of one or more *regionserver* slaves.
- The HBase master is responsible for bootstrapping a virgin install, for assigning regions to registered region servers and for recovering region server failures.

HBase Architecture



- Column – oriented database (column families)
- Table consists of Rows, each which has a primary key(row key)
- Each Row may have any number of columns
- Table schema only defines Column familes (column family can have any number of columns)
- Each cell value has a timestamp

- HBase shell
 - Scan, List, Create, Get, Put, Delete
- Java API
 - Get, Put, Delete, Scan,...
- Non-Java Clients
 - Thrift
 - REST
- HBase MapReduce API
 - `hbase.mapreduce.TableMapper;`
 - `hbase.mapreduce.TableReducer;`
- High Level Interface
 - Pig, Hive

- 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
- 0 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

- 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
 - Not good for Random read write

- HBase is designed to efficiently address the above points
 - Fast record lookup
 - Support for record-level insertion
 - Support for updates
- HBase updates are done by creating new versions of values

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	No Integration
Horizontal scalability –just add hardware	Hard to scale
Consistent	Consistent
No transactions	Transactional
Good for structured, semi-structured & un-structured data	Good for structured data
Millions of queries /second	100s queries/second

- Facebook Analytics
 - Real-time counters of URLs shared, preferred links
- Twitter
 - 25 TB of message every month
- Mozilla
 - Store crashes report, 2.5 million per day.

Thank You

DataFlair Web Services Pvt Ltd

+91-8451097879 / +91-7718877477

info@data-flair.com

<http://data-flair.com>

<https://www.facebook.com/DataFlairWS>