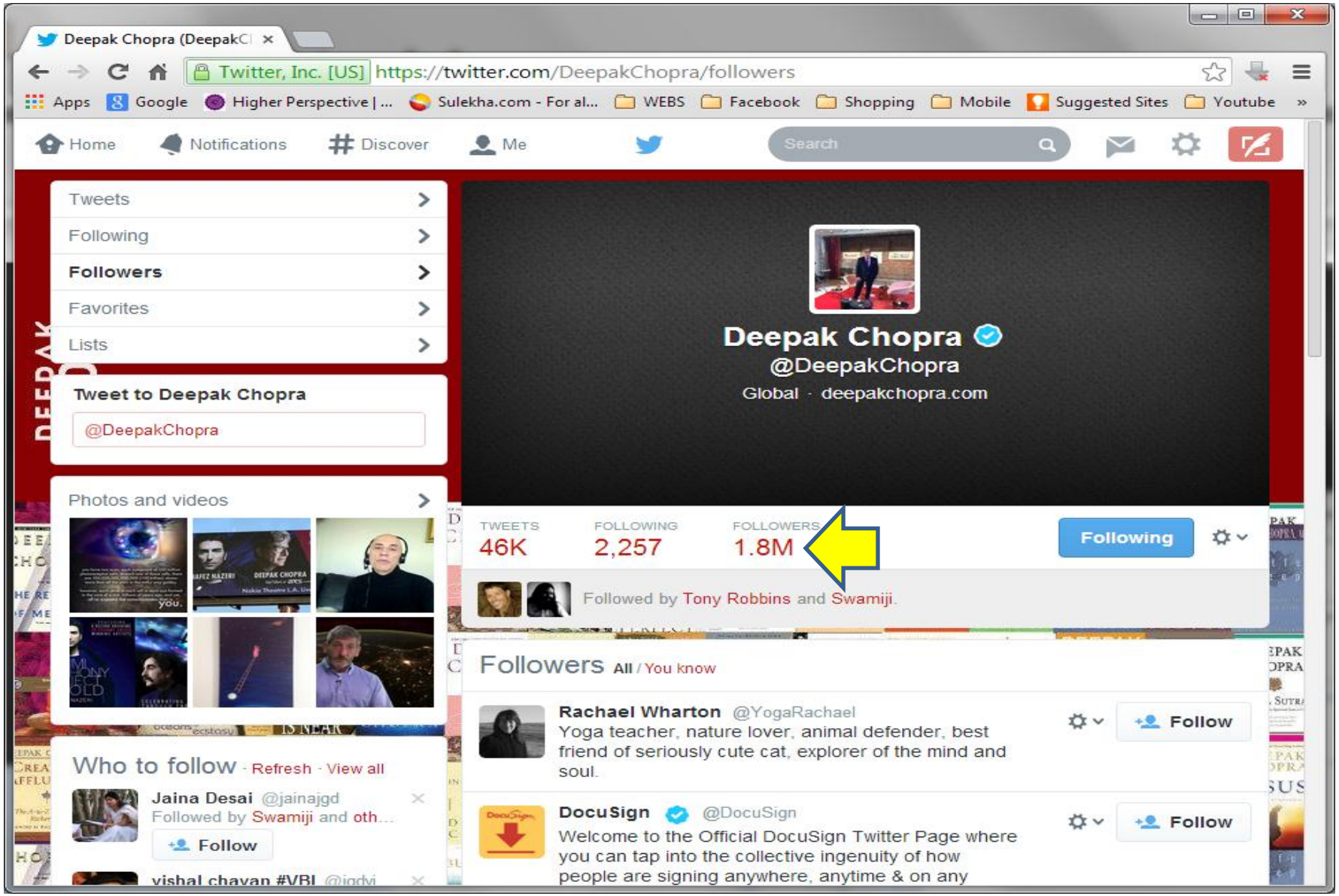
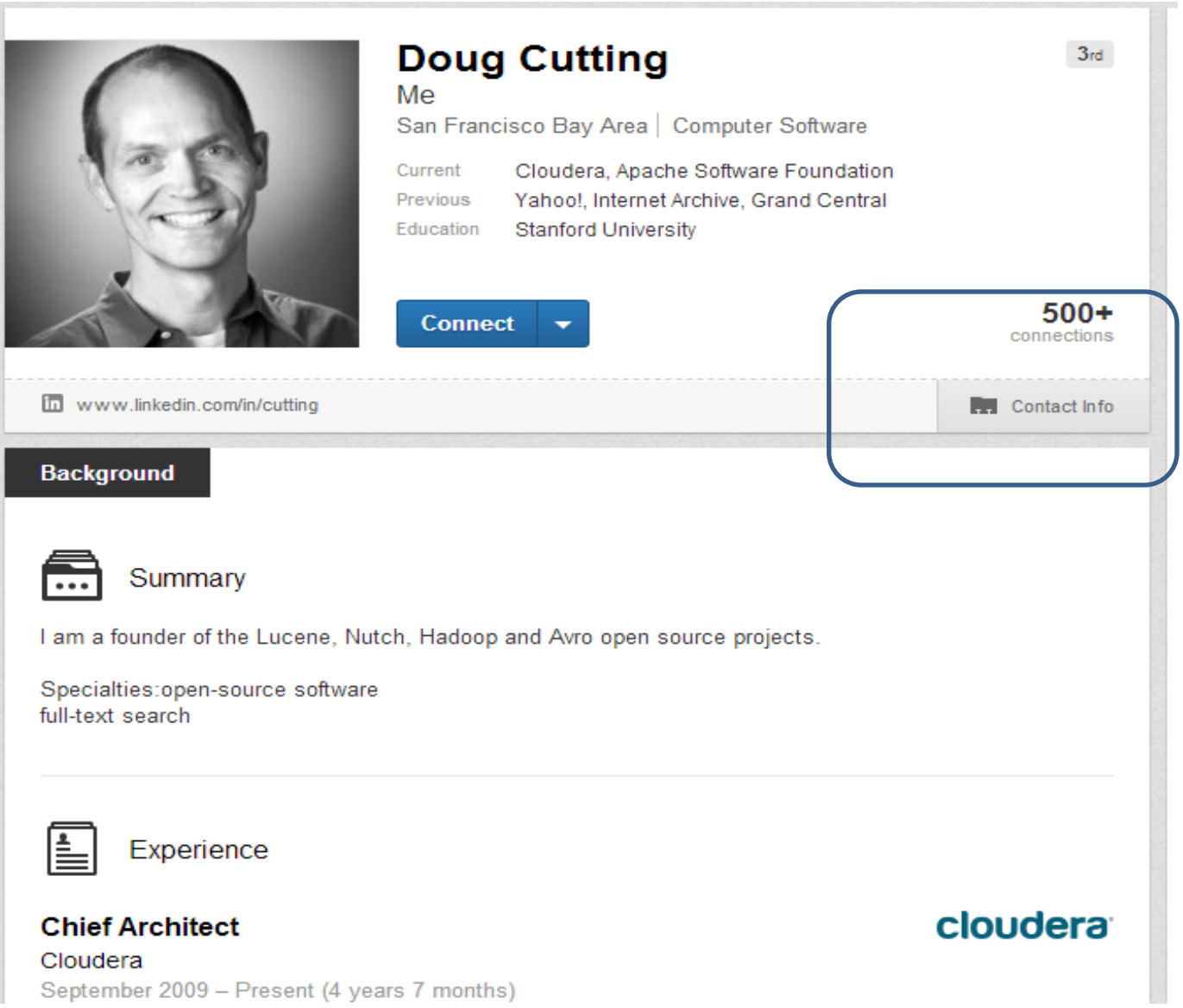
HBASE

Challenges in the Real World



Challenges in the Real World



Doug Cutting 3rd

Me

San Francisco Bay Area | Computer Software

Current Cloudera, Apache Software Foundation
Previous Yahoo!, Internet Archive, Grand Central
Education Stanford University

Connect

500+ connections

www.linkedin.com/in/cutting

Contact Info

Background

Summary

I am a founder of the Lucene, Nutch, Hadoop and Avro open source projects.

Specialties:open-source software
full-text search

Experience

Chief Architect
Cloudera
September 2009 – Present (4 years 7 months)

cloudera

Challenges in the Real World

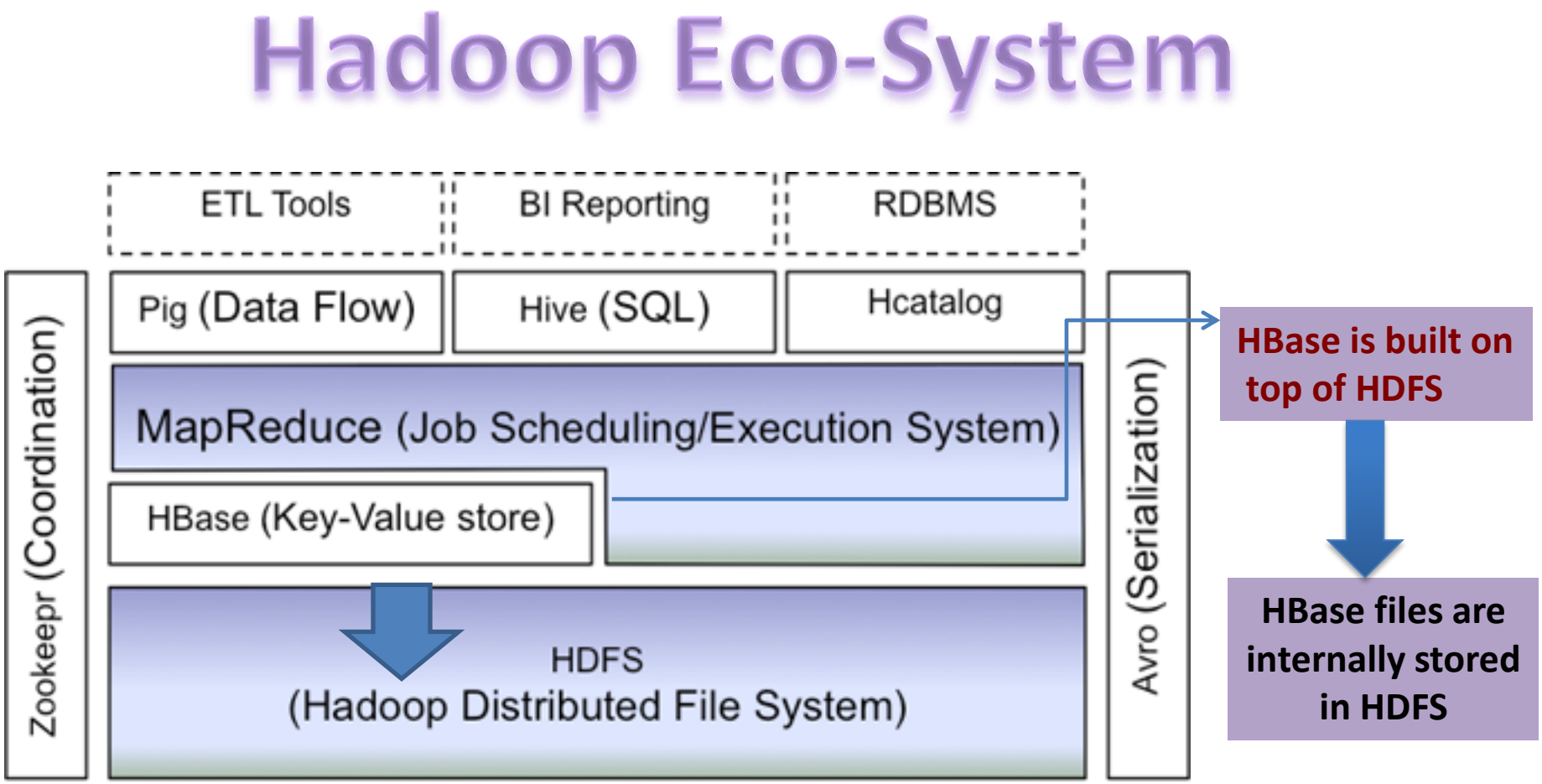
So, what is common ?

- ✓ Fast Random Access
- ✓ Structured Data
- ✓ Huge Data
- ✓ Unstructured Data
- ✓ Variable Schema
- ✓ Need of Compression
- ✓ Need of the Distribution (Sharding)

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 and Hadoop EcoSystem



HBase Vs HDFS

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

- HDFS is for batch processing (scans over big files) and does not support :

  - for record lookup
  - incremental addition of small batches
  - Records level updates

How tradition system will solve this ?

User
User_Id
Name
Sex
Age

Connections
User_id
Connection id
type

User	
User ID	1
Name	Doug Cutting
Sex	M
Age	45

User Connections		
User-id	**Connection-id**	**Type**
1	1-2	Type1
1	1-3	Type1
1	1-4	Type1
1	1-5	Type1
1	1-6	Type1
1	1-7	Type1
.	.	.
.	.	.
1	1-500	Type1

Hbase – Variable Schema

Column Family 1 - User Details				Column Family 2 - Connection-details	
User ID	Name	Sex	Age	Connection-id-01	Type-id-01
1	DC	M	45	1-2	Type1

Column Family 1 - User Details				Column Family 2 - Connection-details			
User ID	Name	Sex	Age	Connection-id-01	Type-id-01	Connection-id-02	Type-id-02
1	DC	M	45	1-2	Type1	1-3	Type1

Column Family 1 - User Details				Column Family 2 - Connection-details							
User ID	Name	Sex	Age	Connection-id-01	Type-id-01	Connection-id-02	Type-id-02	Connection-id-03	Type-id-03		Connection-id-499	Type-id-499
1	DC	M	45	1-2	Type1	1-3	Type1	1-4	Type1	.....	1-499	Type1

Data Model

- Distributed Database
- Sorted Data
- Sparse Data Store
- Automatic sharding
- Multi dimensional
- Map
- Column = Column Family : Column Qualifier

Data Model

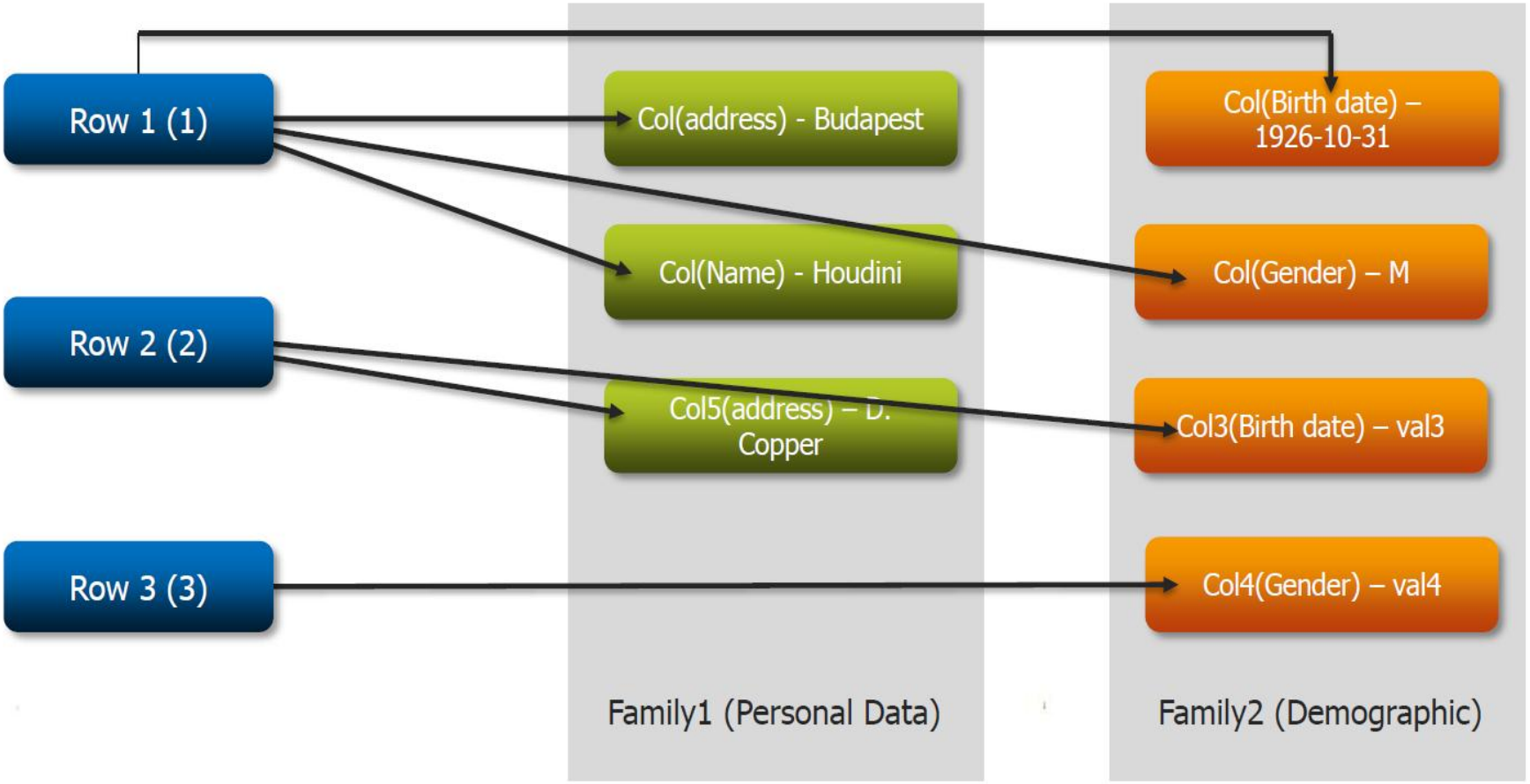
Table Schema only defines it's Column Families

- Consists of any number of columns
- Consists of any number of versions
- Columns exist when inserted
- Columns in a family are sorted & stored together

Data Model

Row key	Personal_data		demographic	
Persons ID	Name	Address	Birth Date	Gender
1	H. Houdini	Budapest	1926-10-31	M
2	D. Copper		1956-09-16	M
3	Merlin		1136-12-03	F
4	.....	.....	.....	M
500,000,000	F. Cadillac	Nevada	1964-01-07	M

Physical Storage

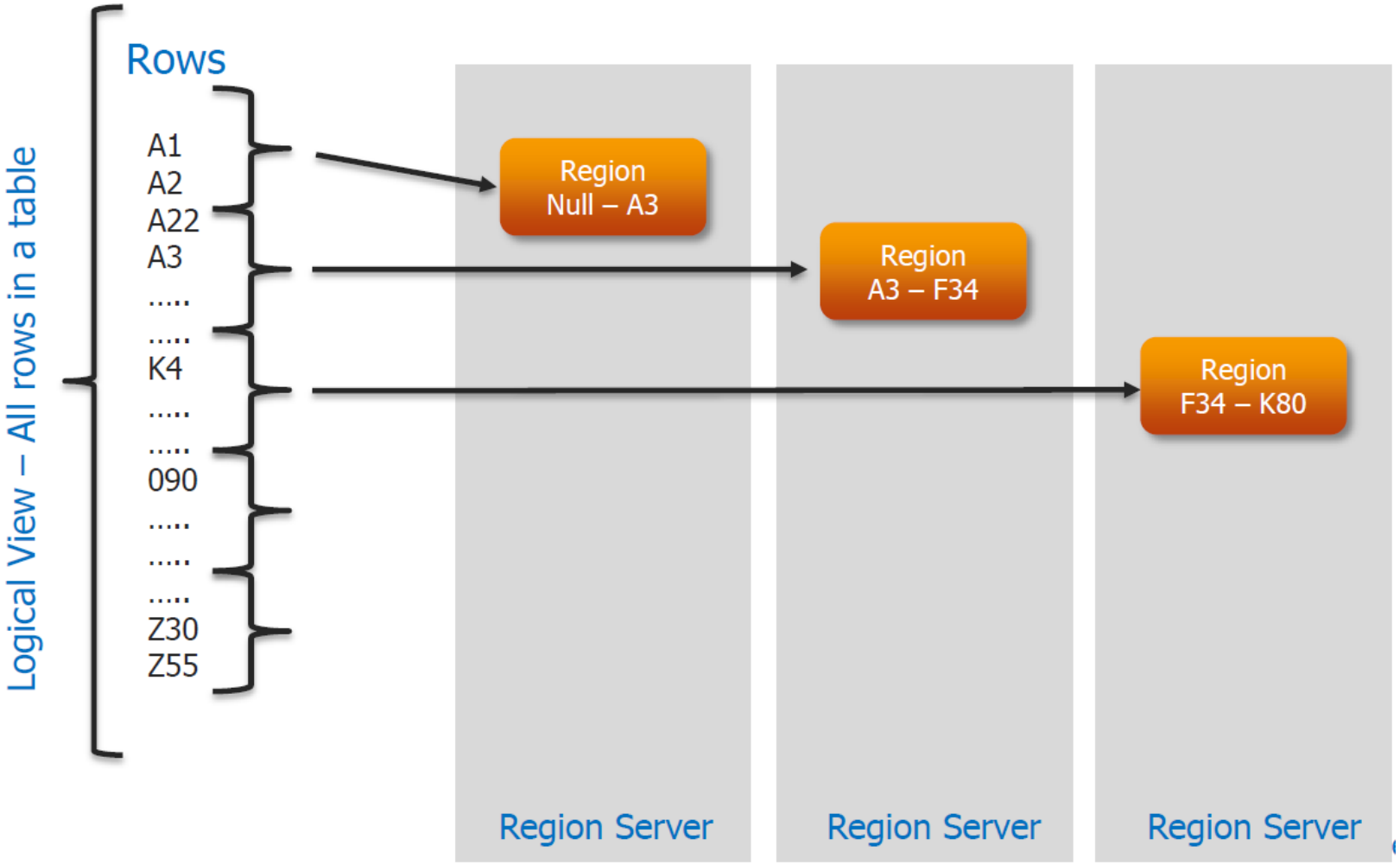


HBase Shell - How does it looks like

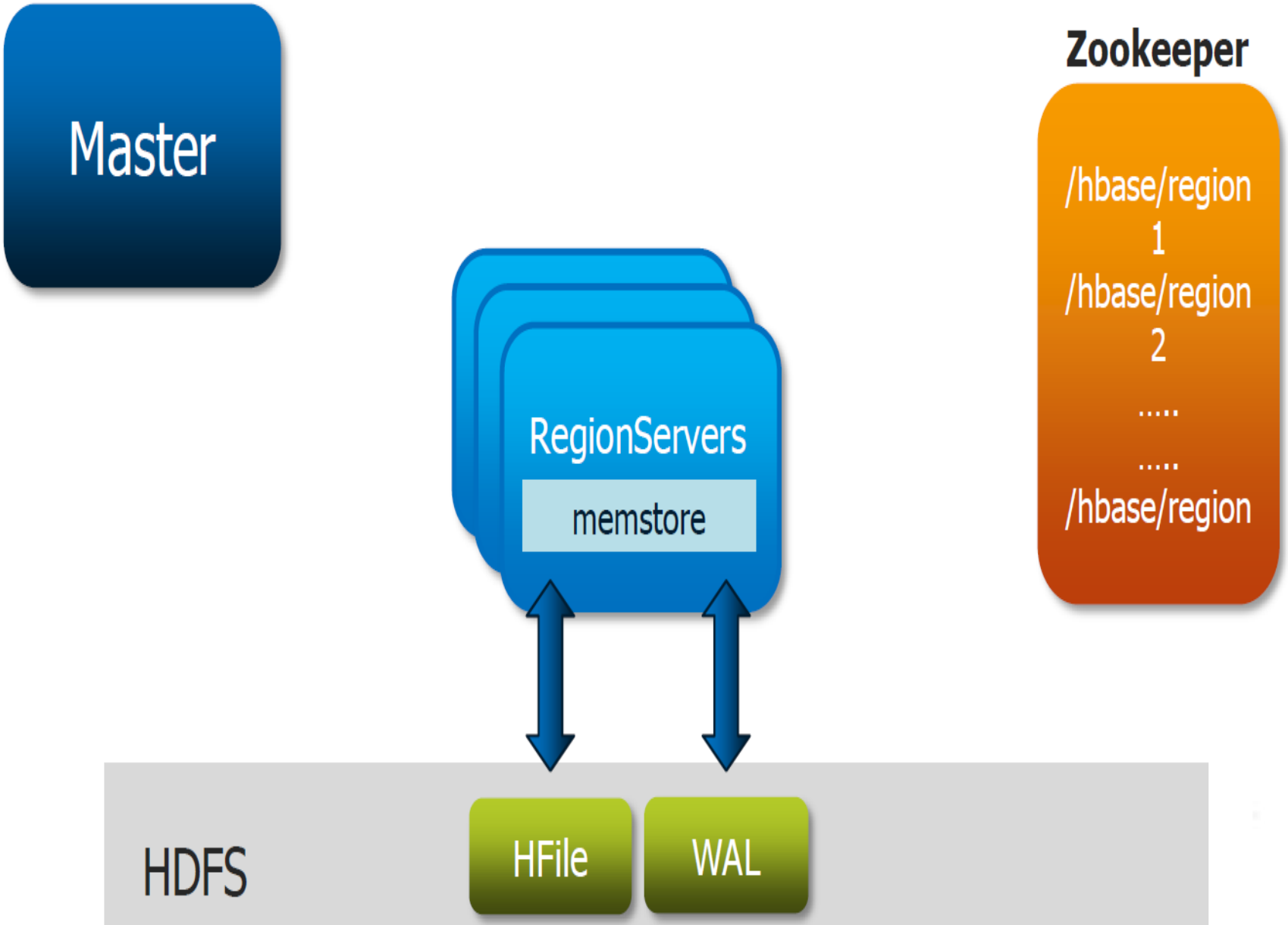
```
hbase(main):002:0> scan 'employee'
ROW                    COLUMN+CELL
 emp1                  column=address:city, timestamp=1396093072595, value=Mumbai
 emp1                  column=address:state, timestamp=1396093081699, value=MAH
 emp1                  column=address:street, timestamp=1396093010180, value=Thakur Village
```

Row Key	Column Family	Column qualifier	Values
Value- emp1 • Unique for each row • Identified each row	Value - address • For Faster access - less no of Column family are preferred. • CF are fixed	Value – city, state, street • Col qualifiers are NOT fixed. Can be added on the fly.	Value – Mumbai, MAH, Thakur Village • Various version of values are maintained

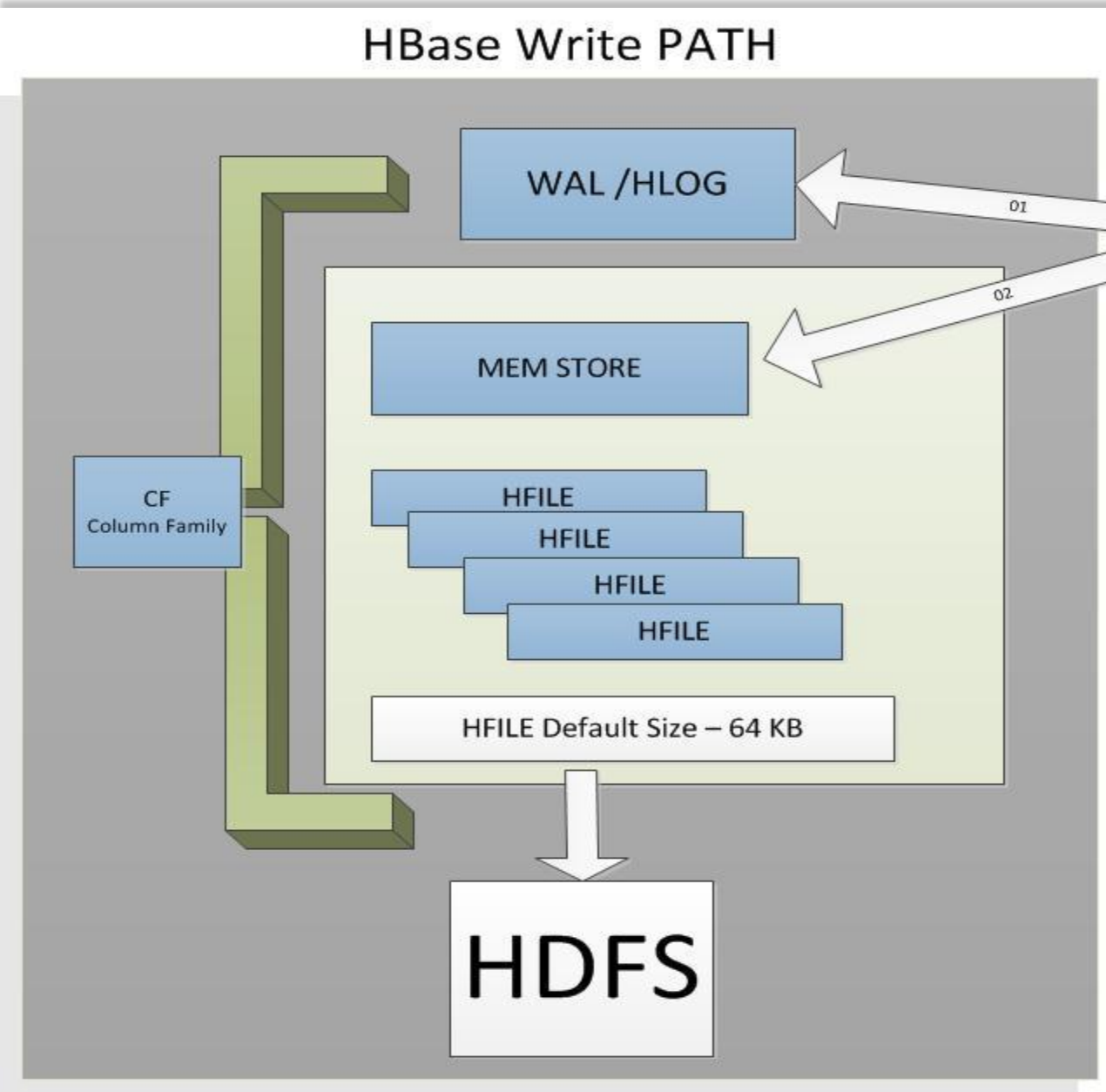
Data Distribution



Data components



Data Model

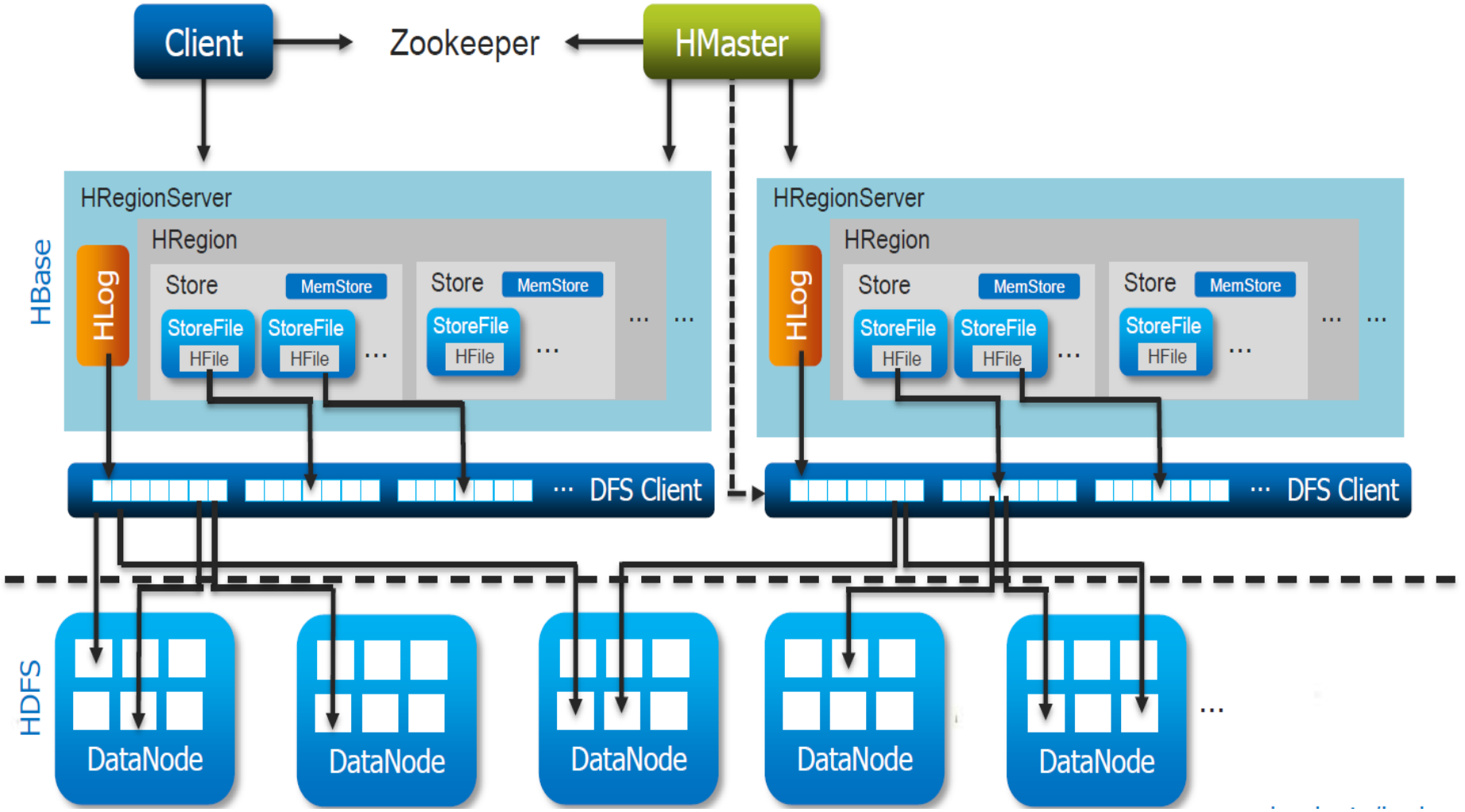


Mem-Store :

After data is written to the WAL the RegionServer saves KeyValues in memory store

- Flush to disk based on size, see hbase.hregion.memstore.flush.size
- Default size is 64MB
- Uses snapshot mechanism to write flush to disk while still serving from it and accepting new data at the same time
- Snapshots are released when flush has succeeded

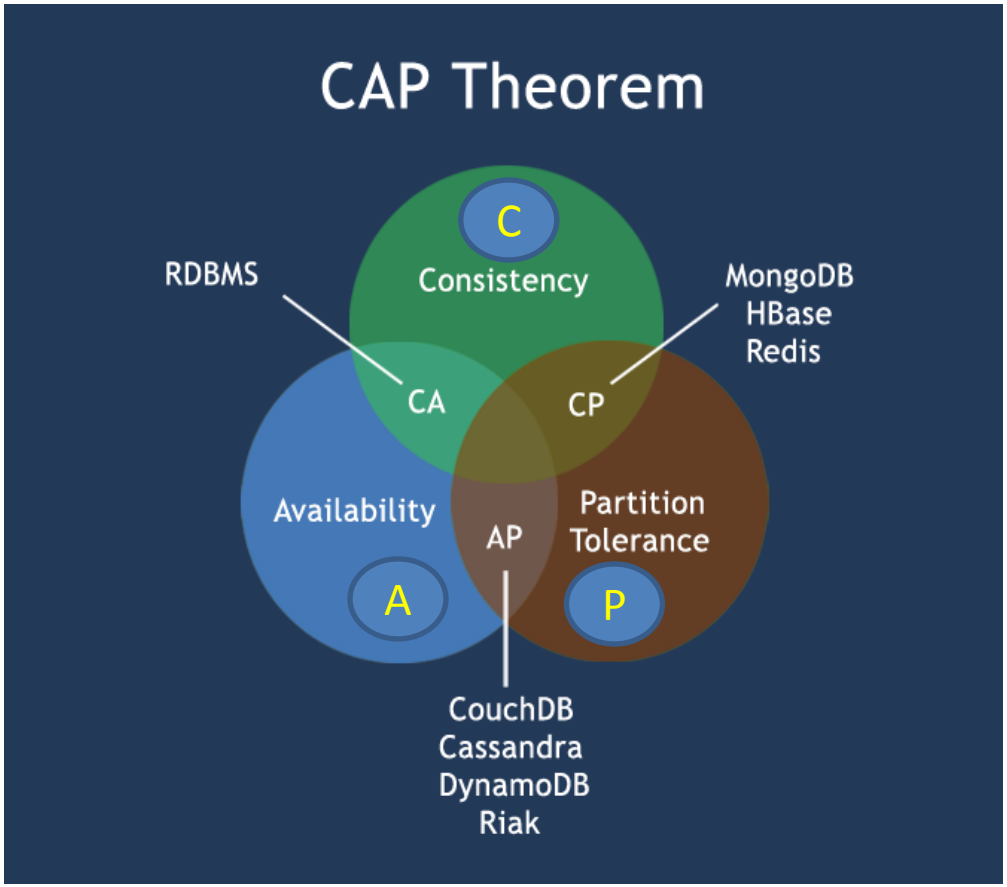
Hbase Storage Architecture



Compactions

- ✓ HBase writes out immutable files as data is added
  - ✓ Each store consists of rowkey-ordered files
  - ✓ Immutable – More files accumulate over time.
- ✓ Compaction rewrites several files into one.
  - ✓ Lesser files – faster reads.
- ✓ Major compaction rewrites all files in a store into one.
  - ✓ Can drop deleted records and old versions.
- ✓ In a minor compaction, files to compact are selected based on a heuristic.

CAP Theorem



Choose Any Two : CA, CP or AP

C **C**onsistent - **C**ommits are atomic across the entire DFS

A **A**vailable - Remain **A**ccessible and operated all the time

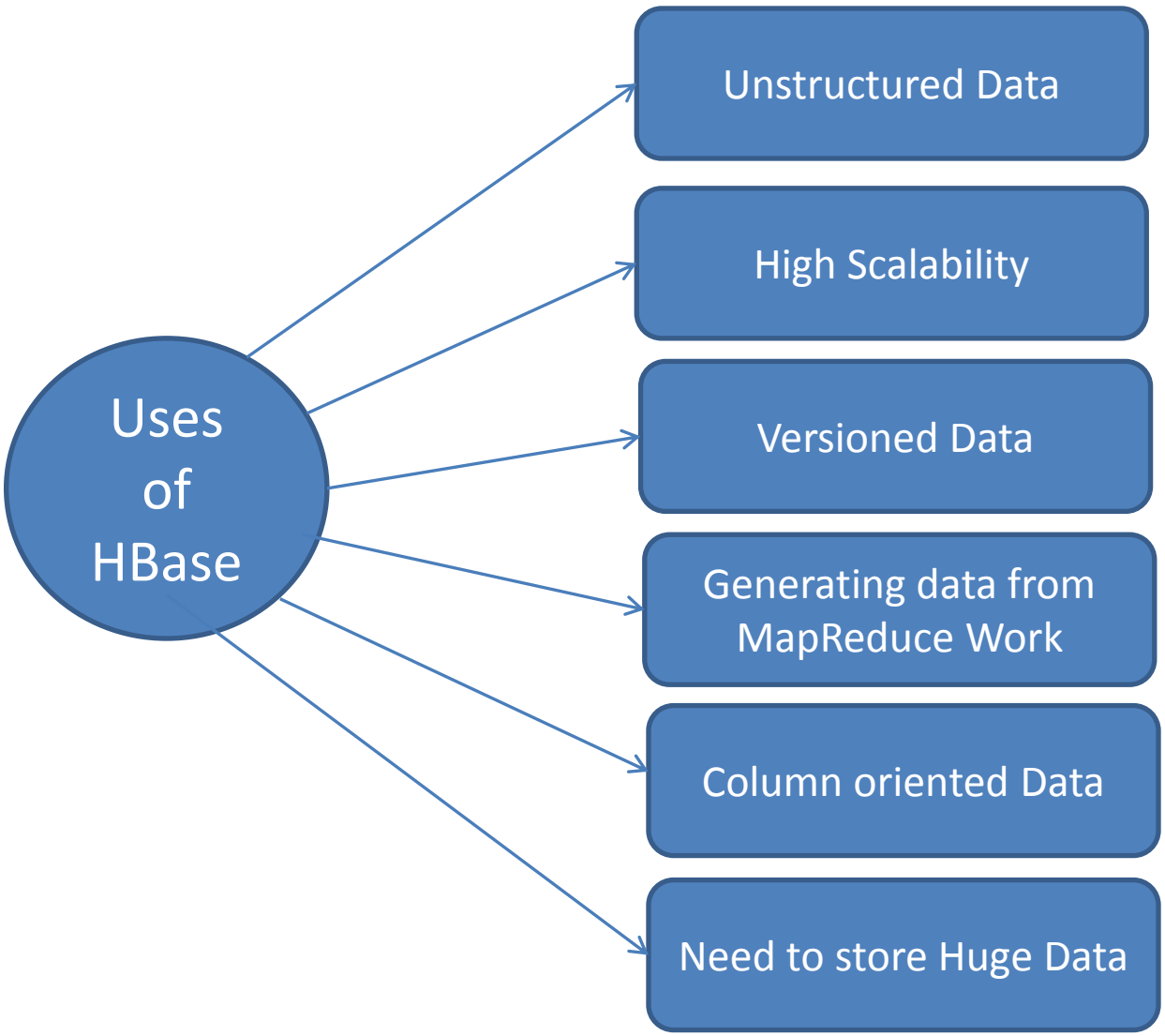
P **P**artition Tolerance : Only total network failure can cause system to response incorrectly

- According to the theorem, a distributed system cannot satisfy all three of these guarantees at the same time

Hbase v/s RDBMS

HBASE	RDBMS
Column Oriented	Row Oriented (Mostly)
Flexible schema, columns can be added in run time	Fixed schema
Good with sparse tables	Not optimized for sparse tables
Joins using MR- not optimized..Design in such a way as to avoid joins….millions of columns ?	Optimized for Joins
Tight integration with MR	Not really
Scalability - Scale Out Storage (Horizontally)	Scale Up - Difficult to shard and scale
Good for semi-structured data as well as structured data	Good for semi-structured data

When to use HBase



When **NOT** to use HBase

- When you have only a few thousands/millions of rows
- Lacks RDBMS Commands
- When you have hardware less than 5 data nodes when replication factor is 03

More Hbase Implementation

A number of applications including people search rely on HBase internally for data generation.

Uses HBase to power their Messages infrastructure http://sites.computer.org\debull\A12june\facebook.pdf

We use HBase as a real time data storage and analytics platform.

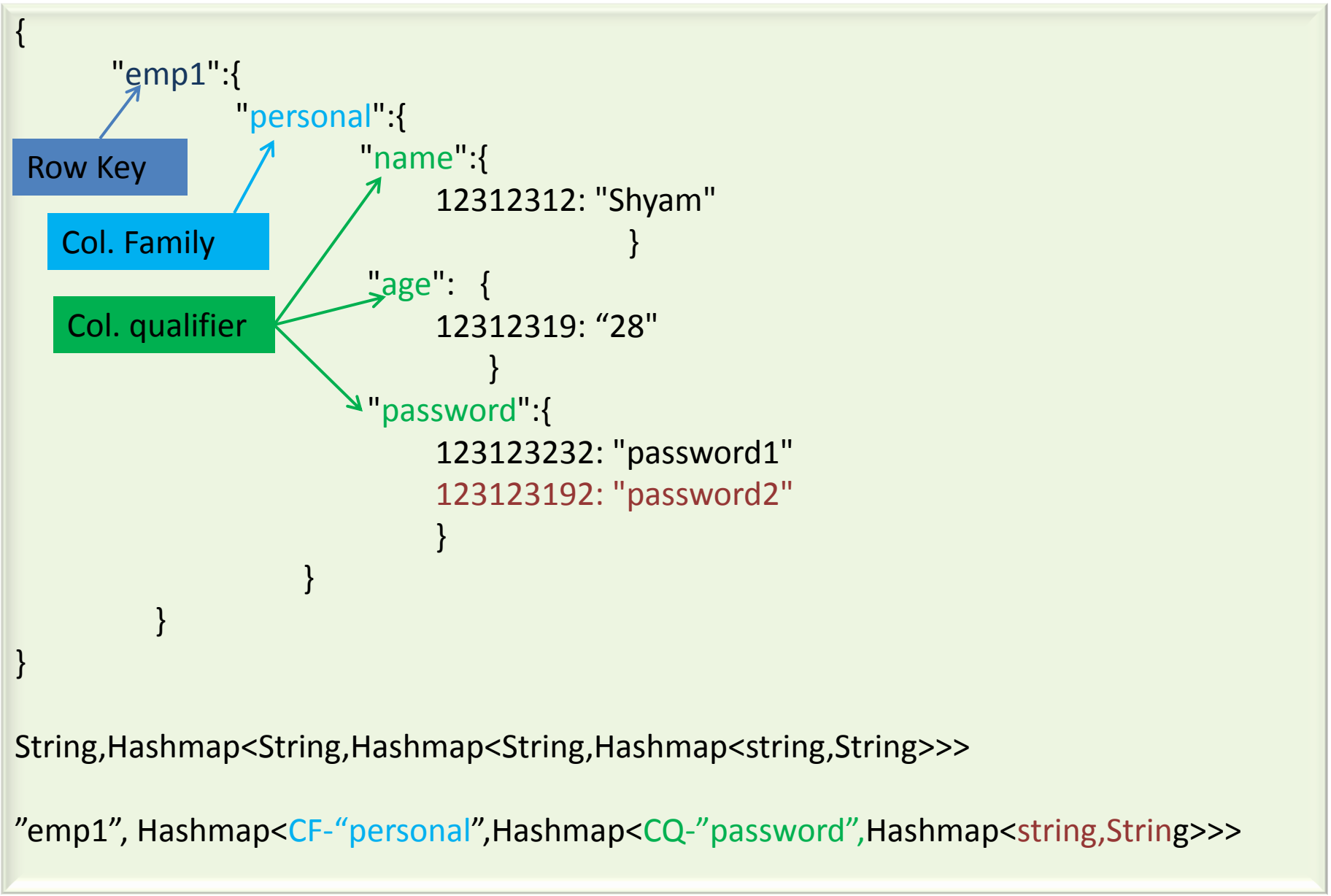
Uses HBase to store document fingerprint for detecting near-duplications. We have a cluster of few nodes that runs HDFS, mapreduce, and HBase.

Uses an HBase cluster containing over a billion anonymized clinical records.

Uses HBase as a foundation for cloud scale storage for a variety of applications.

Referred - http://wiki.apache.org/hadoop/Hbase/PoweredBy

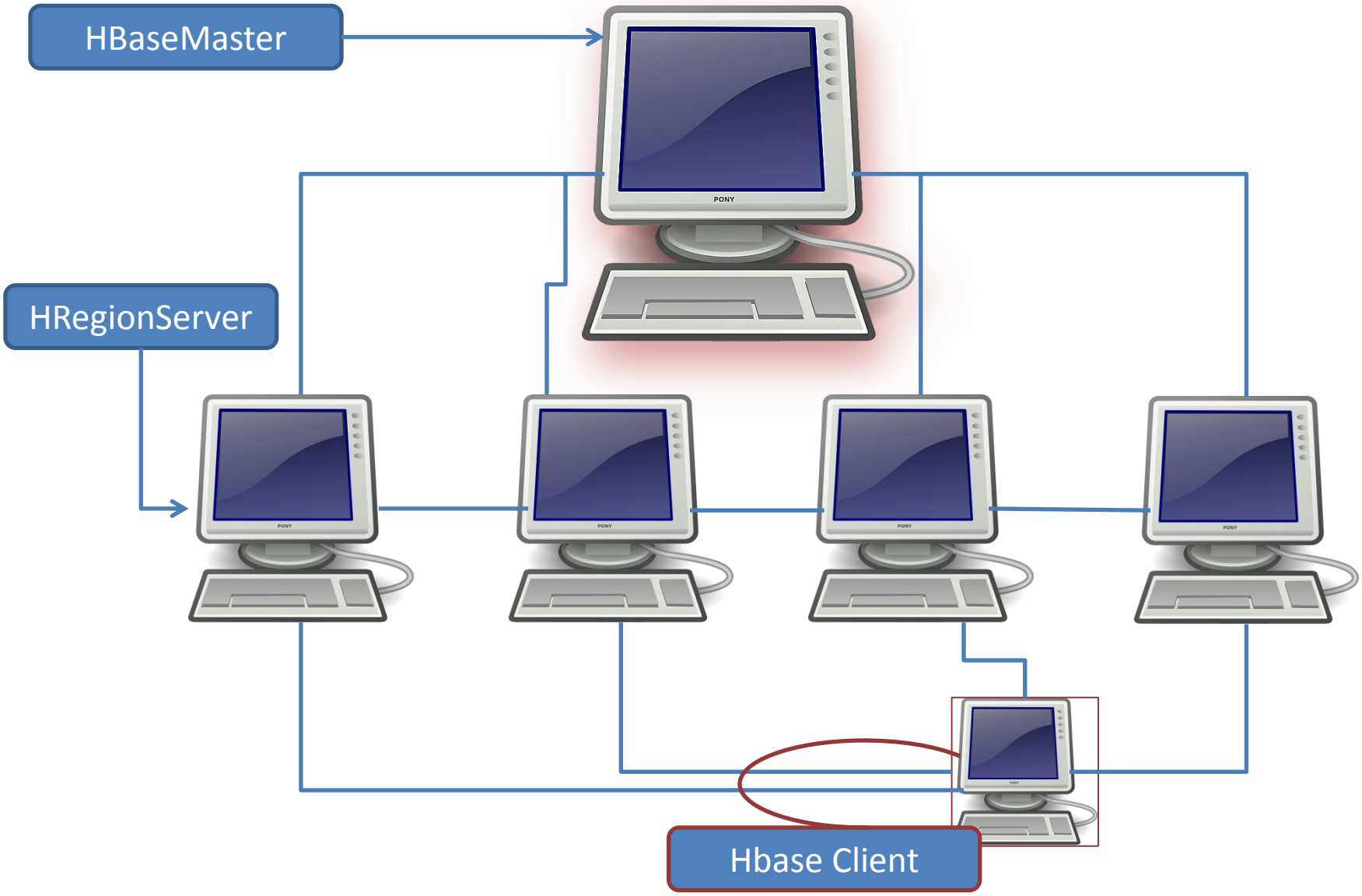
HBASE : Multi-Dimension



String,Hashmap<String,Hashmap<String,Hashmap<string,String>>>

”emp1”, Hashmap<CF-“personal”,Hashmap<CQ-”password”,Hashmap<string,String>>>

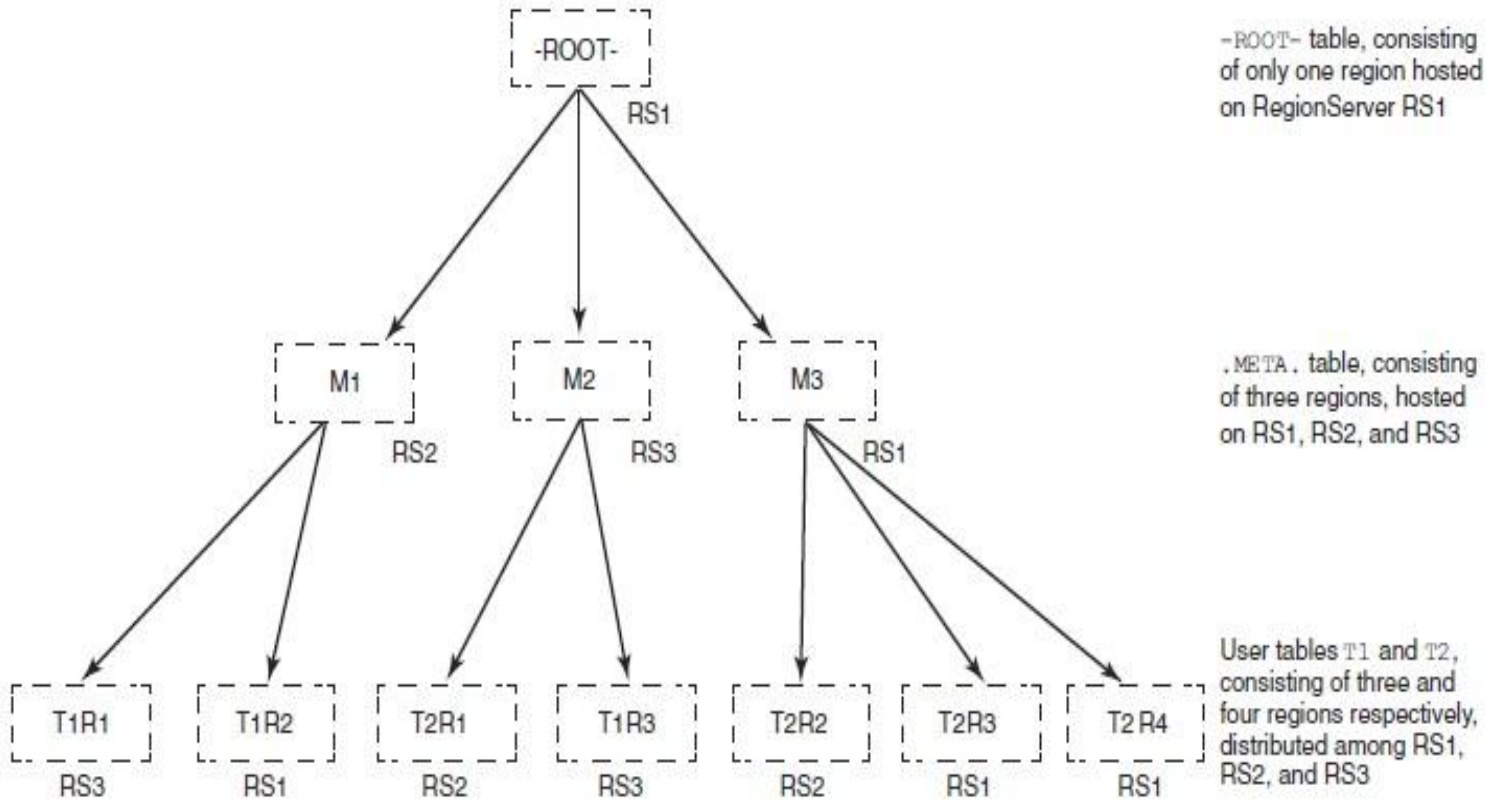
HBase - Major Components



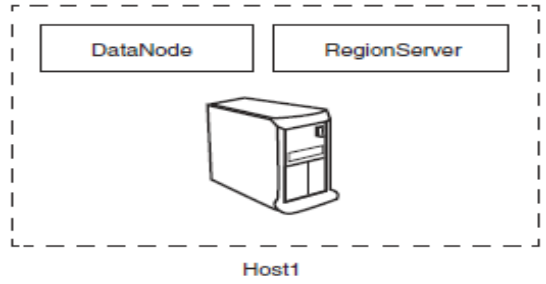
When **NOT** to use HBase

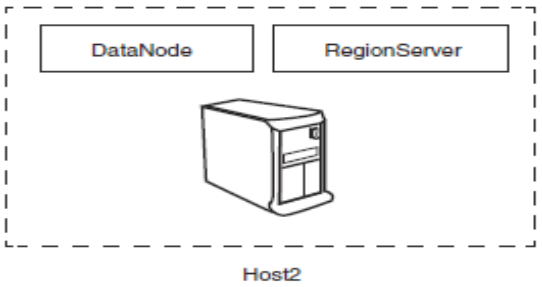
- When you have only a few thousands/millions of rows
- Lacks RDBMS Commands
- When you have hardware less than 5 data nodes when replication factor is 03

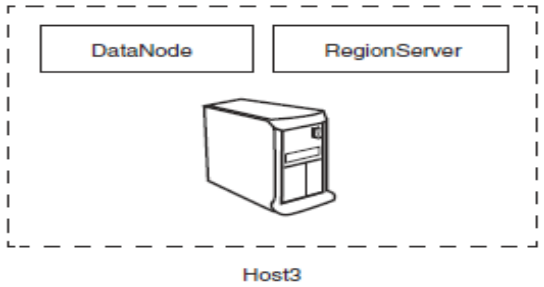
How do I find ROOT . META and specific Region – for a particular record



How do I find ROOT . META and specific Region – for a particular record







T1-R1

00001	John	415-111-1234
00002	Paul	408-432-9922
00003	Ron	415-993-2124
00004	Bob	818-243-9988

RegionServer 1 (RS1), hosting regions R1 of the user table `T1` and M2 of `.META.`

.META. - M2

T1:00009-T1:00012	R3	RS2

T1-R2

00005	Carly	206-221-9123
00006	Scott	818-231-2566
00007	Simon	425-112-9877
00008	Lucas	415-992-4432

T1-R3

00009	Steve	530-288-9832
00010	Kelly	916-992-1234
00011	Betty	650-241-1192
00012	Anne	206-294-1298

RegionServer 2 (RS2), hosting regions R2, R3 of the user table and M1 of `.META.`

.META. - M1

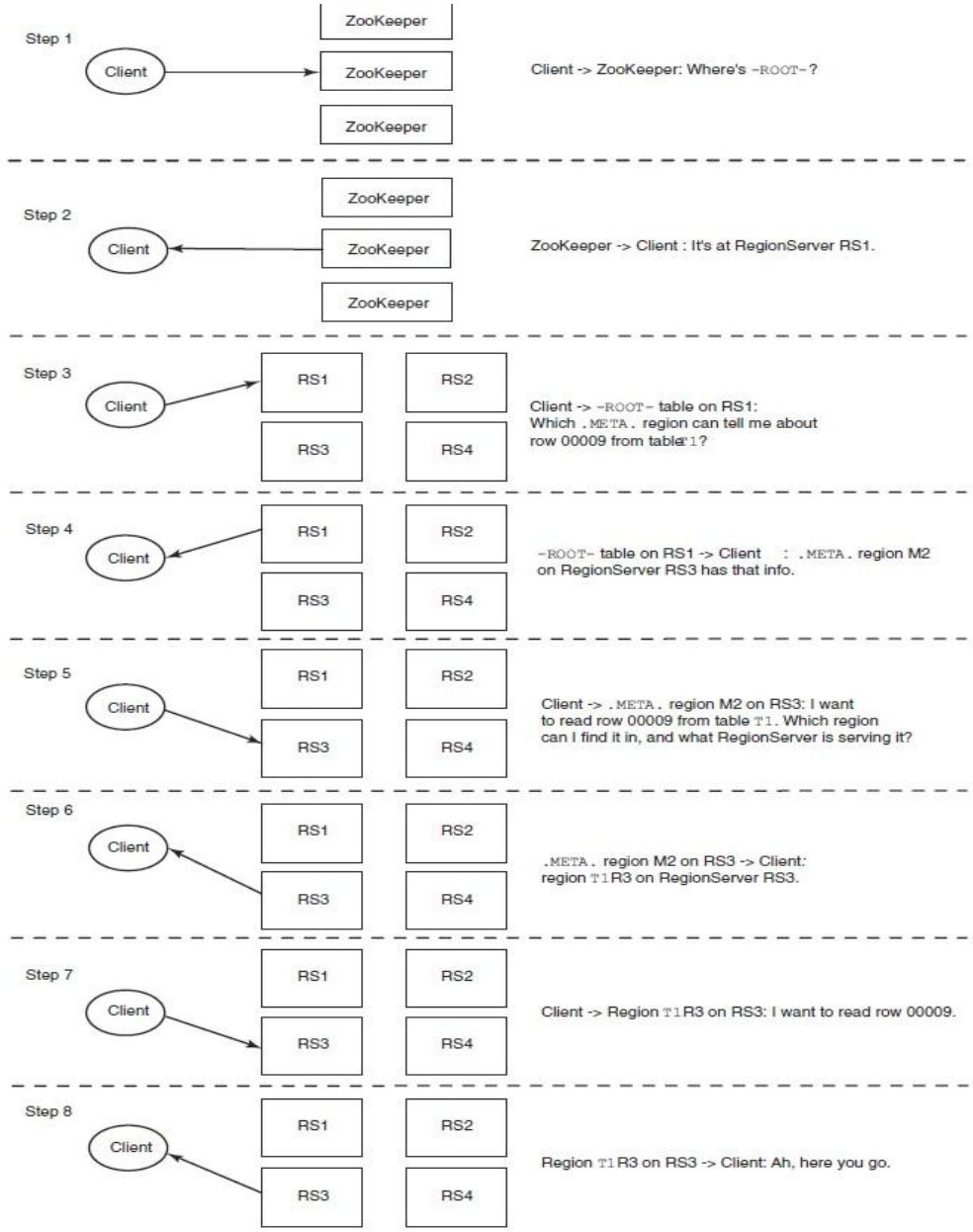
T1:00001-T1:00004	R1	RS1
T1:00005-T1:00008	R2	RS2

-ROOT-

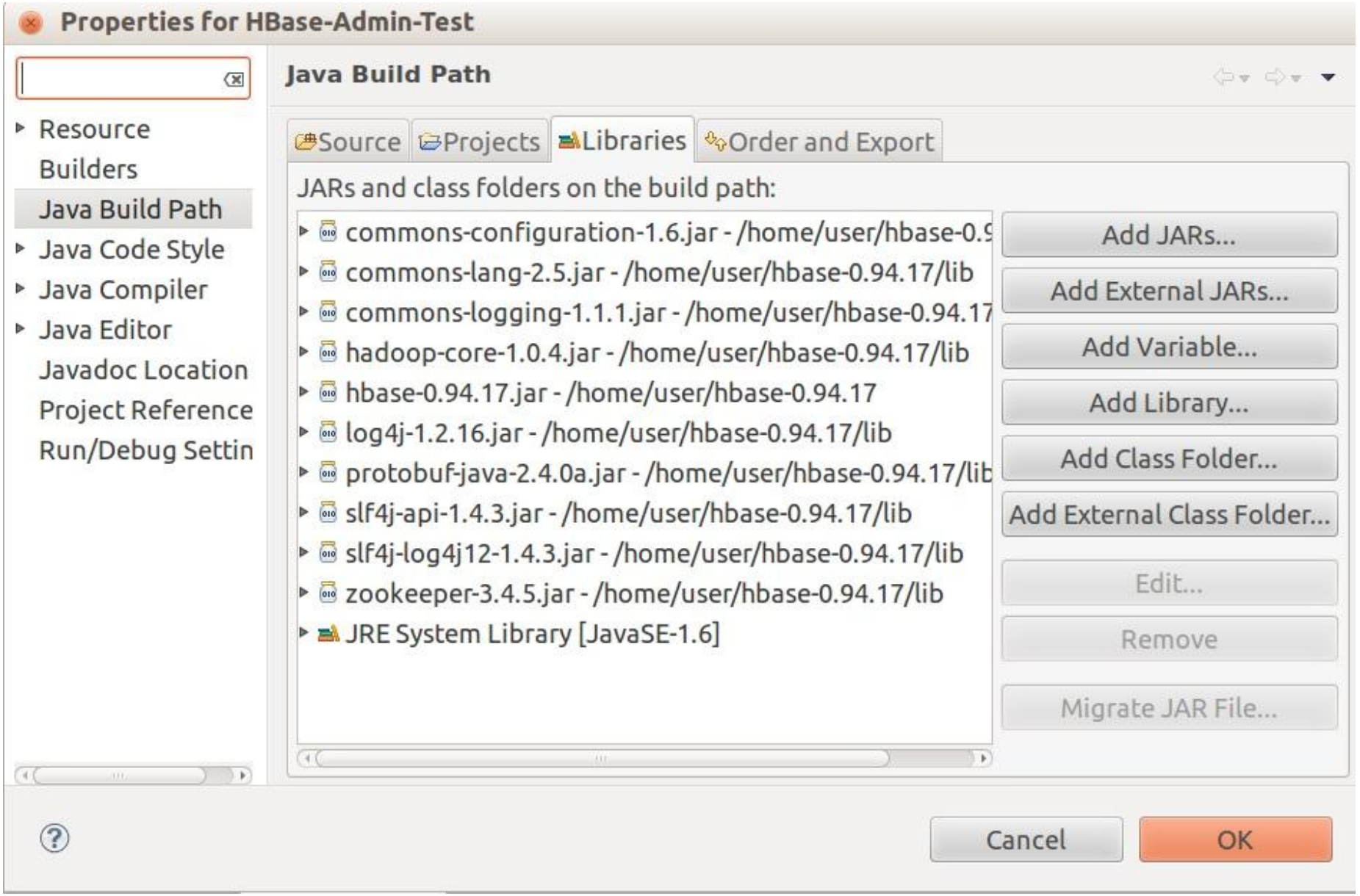
M:T100001-M:T100009	M1	RS2
M:T100010-M:T100012	M2	RS1

RegionServer 3 (RS3), hosting just `-ROOT-`

How do I find ROOT . META and specific Region – for a particular record



List of classes to be included in eclipse before running HBASE program



Bulk Load in Hbase using TSV Format

Before bulk upload, create ‘emp’ table in hbase with column family as ‘cf’

Please ensure to move file ‘first.tsv ‘ to move HDFS before following command

```
Hbase$ bin/hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
-Dimporttsv.columns=HBASE_ROW_KEY, cf:fn, cf:ln emp /first.tsv
```