Welcome to

Big Data & Hadoop

Session

Session 6 - NoSQL & HBASE

+91-9538998962

sandeep@knowbigdata.com

WELCOME - KNOWBIGDATA

- Interact - Ask Questions
- Lifetime access of content
- Class Recording
- Cluster Access
- 24x7 support
- Real Life Project
- Quizzes & Certification Test
- 10 x (3hr class)
- Socio-Pro Visibility
- Mock Interviews

ABOUT ME

2014	**KnowBigData**	Founded
2014	**Amazon**	Built High Throughput Systems for Amazon.com site using in-house NoSql.
2012		
2012	**InMobi**	Built Recommender after churning 200 TB
2011		
	tBits Global	Founded tBits Global Built an enterprise grade Document Management System
2006		
	D.E.Shaw	Built the big data systems before the term was coined
2002		
2002	**IIT Roorkee**	Finished B.Tech somehow.

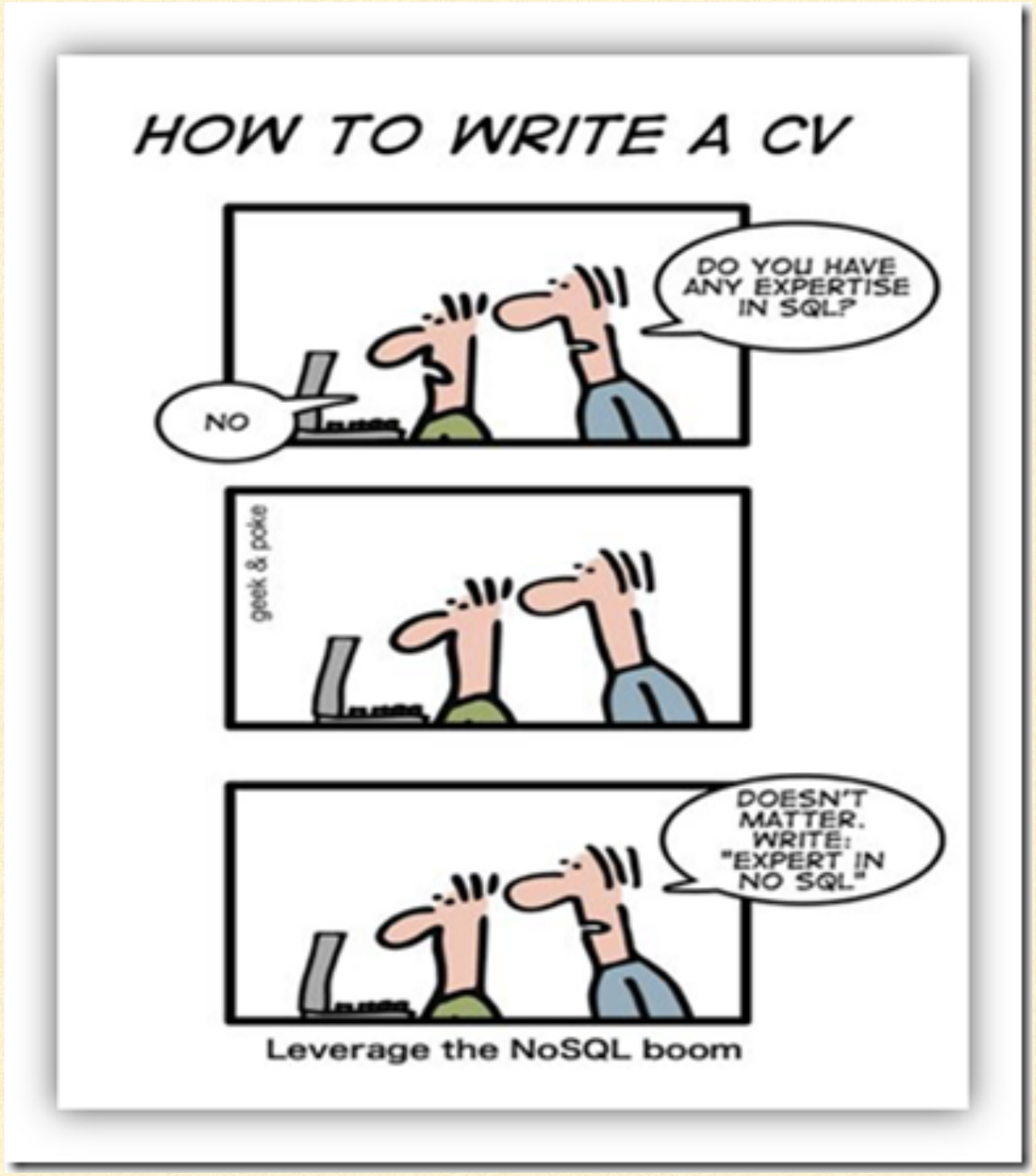
COURSE CONTENT

I	Understanding BigData, Hadoop Architecture
II	Environment Overview, MapReduce Basics
III	Adv MapReduce & Testing
IV	Pig & Pig Latin
V	Analytics using Hive
VI	NoSQL, HBASE
VII	Oozie, Mahout,
VIII	Zookeeper, Apache Storm
IX	Apache Flume, Apache Spark
X	YARN, Big Data Sets & Project Assignment

TODAY'S CLASS

- What & Why NoSQL?
- Type of NoSQL
- NoSQL vs RDBMS
- CAP Theorem
- Column Oriented?
- HBASE
- Architecture
- Region Server
- Bloom Filter
- Quick Demo

NOSQL?


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

WHAT IS A NOSQL DATABASE?

- ID for each row
- Next Generation Databases
- Not Only SQL
- Non-Relational,
- Distributed Architecture
- Open-Source
- Horizontally scalable
- Schema-Free
- Easy Replication
- Simple API
- Manage Huge Data
- Commodity Hardware
- ~150 are in the market

WHY NOSQL?

1. Big Data
2. High Availability

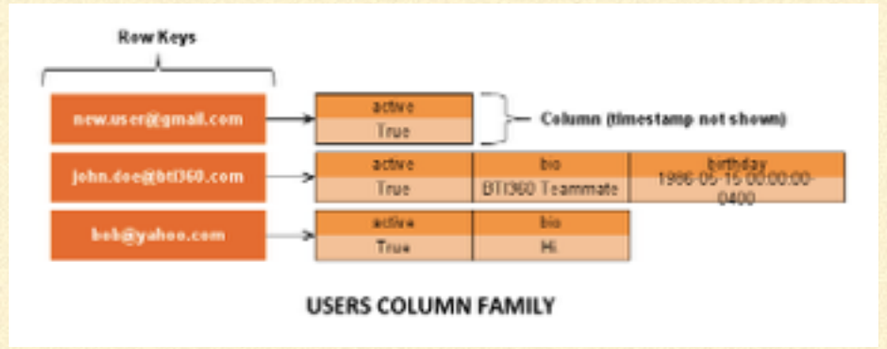
   Cater to many concurrent users
3. Scale-out architecture

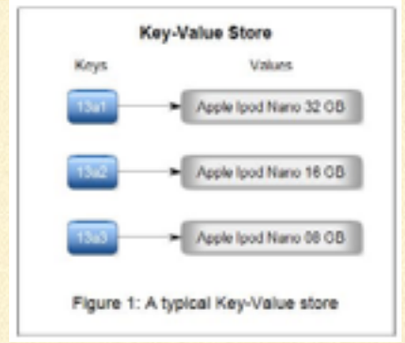
   Commodity hardware

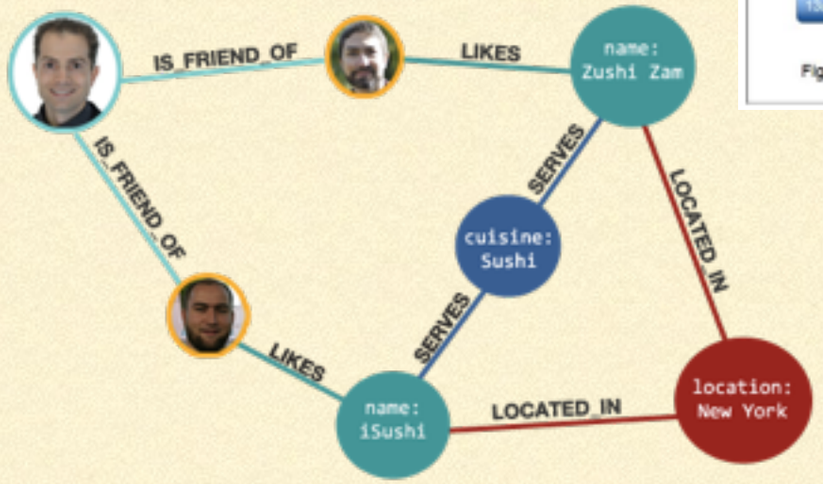
   Instead of expensive, monolithic architecture

TYPES OF NOSQL STORES

- **Column Oriented / wide-column**
  - Accumulo, Cassandra, HBase
- **Document**
  - Clusterpoint, Couchbase, MarkLogic, MongoDB
- **Key-value**
  - Dynamo, MemcacheDB, Project Voldemort, Redis, Riak
- **Graph**
  - Allegro, Neo4J, OrientDB, Virtuoso







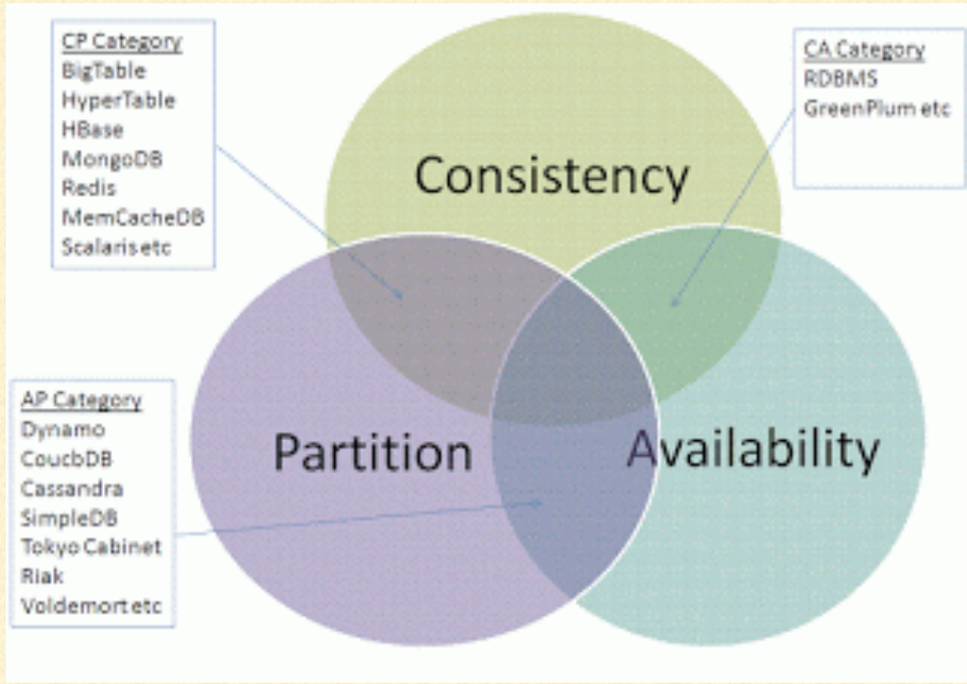
NOSQL VS SQL SUMMARY

	SQL DATABASES	NOSQL DATABASES
Types	One type - minor variations	Many different types
Development History	1970s	2000s
Examples	MySQL, Postgres, Oracle Database	MongoDB, Cassandra, HBase, Neo4j
Data Storage Model	Relational	Varies based on database type.
Schemas	Structure and data types are fixed in advance.	Typically dynamic.
Scaling	Vertically,	Horizontally,
Development Model	Mix	Open-source
Supports Transactions	ACID	In certain circumstances and at certain levels (e.g., document level vs. database level)
Data Manipulation	SQL	Through object-oriented APIs
Consistency	Can be configured for strong consistency	Depends on product.

CAP THEOREM

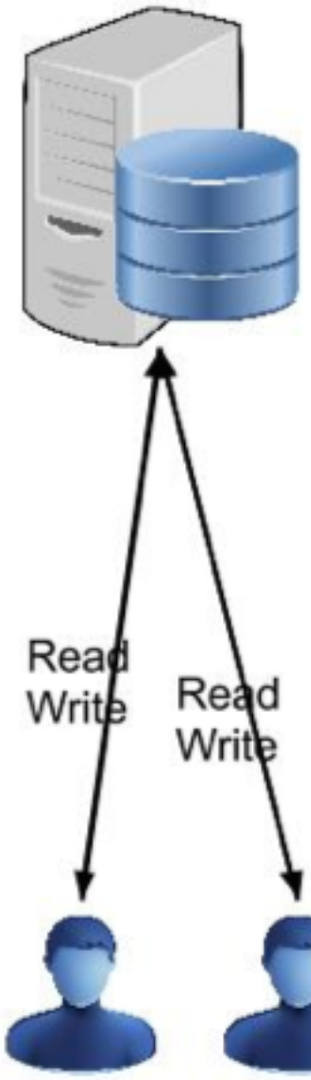
IMPOSSIBLE TO GET ALL THREE OF

1. Consistency
    1. All nodes see the same data at the same time
2. Availability
    1. A guarantee that every request receives a response about whether it was successful or failed
3. Partition tolerance
    1. The system continues to operate despite arbitrary message loss or failure of part of the system

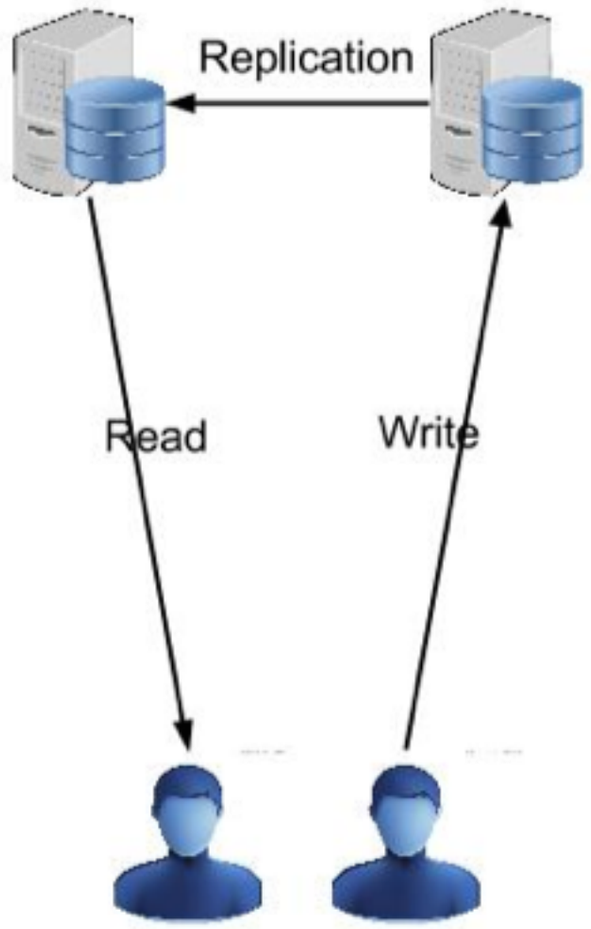


Cap Theorem

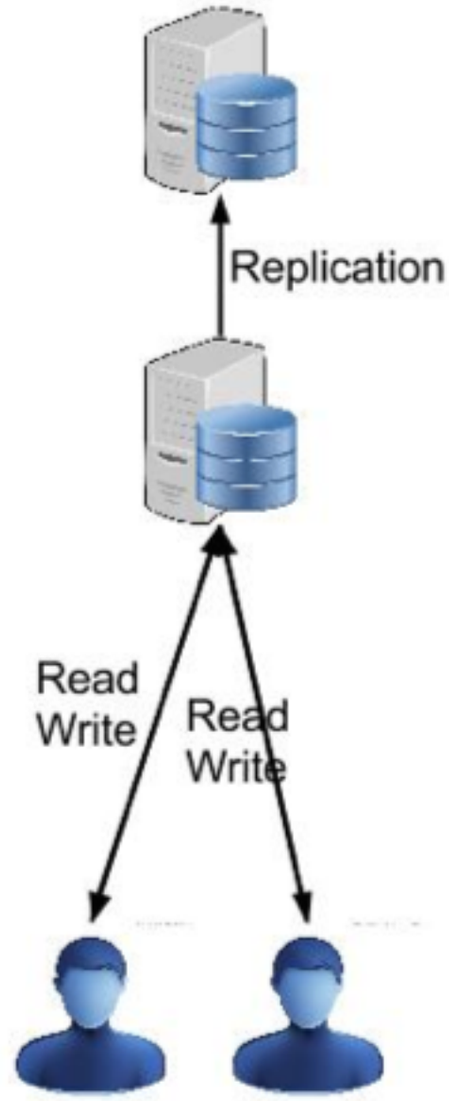
Consistent
~~**A**vailable~~
~~**P**artition Tolerant~~



~~**C**onsistent~~
Available
Partition Tolerant



Consistent
~~**A**vailable~~
Partition Tolerant



Sandeep Giri

RDBMS - STORY

1.Initial public launch

Move from local workstation to shared, remotely hosted MySQL instance with a well-defined schema.

2. Service becomes more popular; too many reads hitting the database

Add memcached to cache common queries. Reads are now no longer strictly ACID; cached data must expire.

3. Service continues to grow in popularity; too many writes hitting the database

Scale MySQL vertically by buying a beefed-up server with 16 cores, 128 GB of RAM, and banks of 15 k RPM hard drives. Costly.

RDBMS - STORY

New features increases query complexity; now we have too many joins

De-normalize your data to reduce joins.
(That's not what they taught me in DBA school!)

Rising popularity swamps the server;

Things are too slow. Stop doing any server-side computations.

Some queries are still too slow

Periodically prematerialize the most complex queries, and try to stop joining in most cases.

Reads are OK, but writes are getting slower and slower

Drop secondary indexes and triggers (no indexes?).

COLUMN ORIENTED DATABASE

Data in columns stored nearby
as opposed to the rows being nearby

EMPID	NAME
10	Joe
12	Mary
11	Cathy

001:10,Joe;002:12,Mary;003:11,Cathy;

Row Oriented

10:001,12:002,11:003;Joe:001,Mary:002,Cathy:003;

Column Oriented

COLUMN FAMILY ORIENTED DATABASE

Data in columns stored nearby as opposed to the rows being nearby

EMPID	NAME	AGE
10	Joe	23
12	Mary	33
11	Cathy	45

001:10,joe;002:12,Mary;003:11,Cathy;
001:23,002:33,003:45

Column Family cf1:empid, name, cf2:age

HBASE

- CP - Column Family Oriented Database
- Based on Google's Big Table
- Good for hundreds of millions or billions of rows
- Strongly consistent reads/writes not eventually
- Automatic sharding - region servers
- Automatic RegionServer failover
- Hadoop/HDFS Integration
- Massively parallelized processing (MapReduce)
- API - Thrift & Java
- Coprocessors
- Bloom Filters

CHARACTERISTICS

No real indexes

Rows are stored sequentially, as are the columns in each row.
Therefore, no issues with index bloat
insert performance is independent of table size.

Automatic partitioning

As your tables grow, they will automatically be split into regions
and distributed across all available nodes.

Scale linearly and automatically with new nodes

Add a node, point it to the existing cluster
run the region server.
Regions will automatically rebalance,
and load will spread evenly.

CHARACTERISTICS

Commodity hardware

Clusters are built on $1,000–$5,000 nodes not $50,000 nodes.
RDBMSs are I/O hungry, requiring more costly hardware.

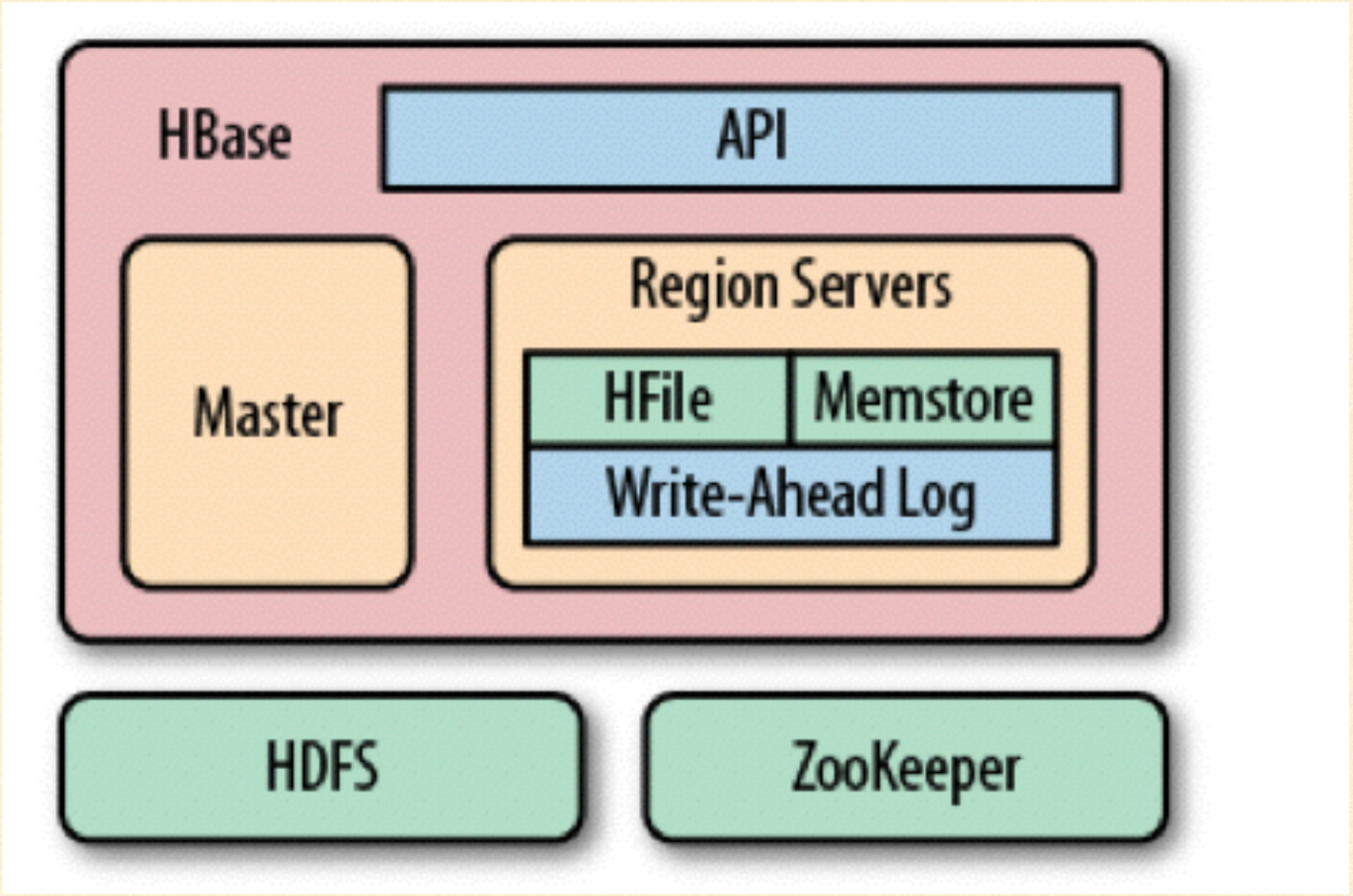
Fault tolerance

Lots of nodes means each is relatively insignificant.
No need to worry about individual node downtime.

Batch processing

MapReduce integration allows fully parallel, distributed jobs against your data with locality awareness.

HBASE ARCHITECTURE



DATA MODEL

Bigtable: a sparse, distributed, persistent multidimensional sorted map.

Table:

- HBase organizes data into tables.
- Table names are Strings that are safe for file system path.

Row:

- Within a table, data is stored according to its row.
- Rows are identified uniquely by their row key.
- Row keys do not have a data type & are treated as byte array

DATA MODEL

Column Family:

- Data within a row is grouped by column family.
- Impacts the physical arrangement of data.
- They must be defined up front and are not easily modified.
- Every row in a table has the same column families
- A row need not store data in all its families.
- Are Strings that are safe for use in a file system path

DATA MODEL

Column Qualifier or Column:

- Data within a column family is addressed by column qualifier
- Column qualifiers need not be specified in advance.
- Column qualifiers need not be consistent between rows.
- Like row keys, column qualifiers don't have data type
- Are always treated as a byte[ ].

Cell:

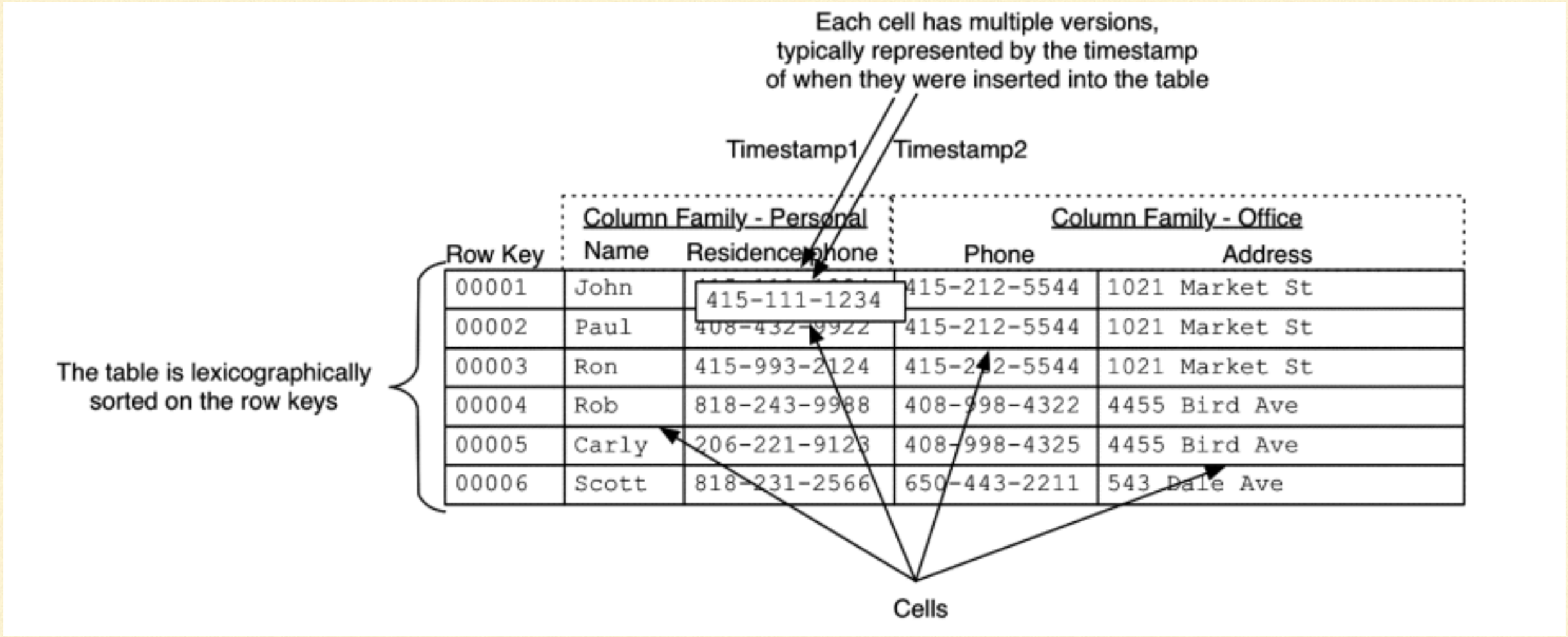
- (tablename, row key, column family, column qualifier, version)
  - identifies a cell.
- Values are of type byte[ ].

DATA MODEL

Timestamp:

- Values within a cell are versioned.
- Version number, which by default is the write timestamp
- If not specified in write, the current time is used.
- If not specified in read, the latest value is returned.
- The number of versions retained by HBase is configured for each column family.
- The default number of cell versions is three

DATA MODEL - LOGICAL VIEW



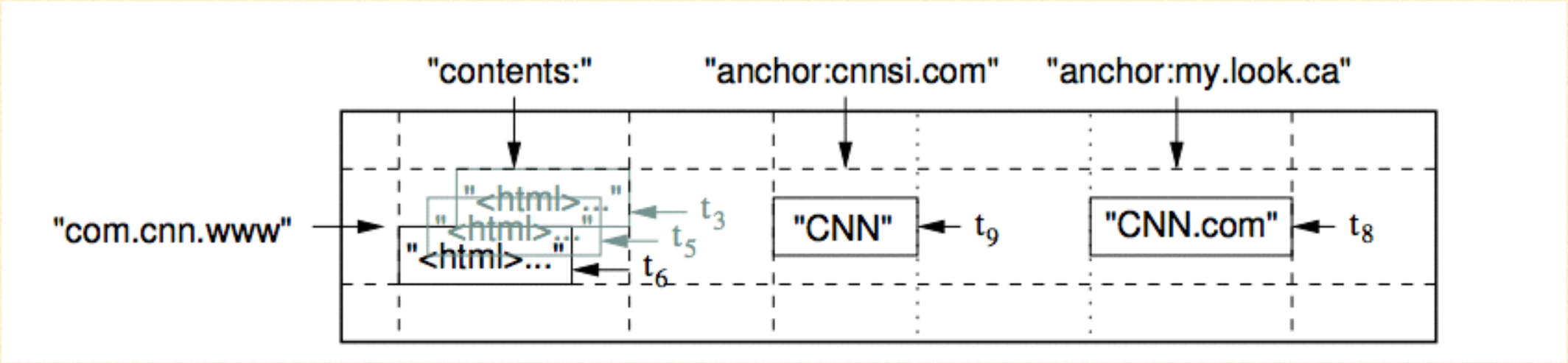
DATA MODEL: Physical View

Bigtable: a sparse, distributed, persistent multidimensional sorted map.

00001
- Office
  - Address
    - Timestamp1
      - **1021 Market St.**
  - Phone
    - Timestamp1
      - **415-212-5544**

Personal
- Name
  - Timestamp1
    - **John**
- Residence Phone
  - Timestamp1
    - **411-111-1111**
  - Timestamp2
    - **415-111-1234**

DATA MODEL - ANOTHER EXAMPLE - WEBTABLE



A region of an example table that stores Web pages.

- row key is a reversed URL.
- contents column family contains the page contents
- anchor column family contains the text of any anchors that reference the page.
- CNN's home page is referenced by both the cnnsi.com & MY-look.ca
- Each anchor cell has one version; the contents column has three versions, at timestamps t3, t5, and t6.
- If one more website *xyz.co* hrefs to www.cnn.com as "News", then it will have one more column *anchor:xyz.co* with value is "News"

DATA MODEL - PHYSICAL VIEW - WEBTABLE

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"

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>..."

DATA MODEL - DESIGN GUIDELINES

- Indexing is based on the Row Key.
- Tables are sorted based on the row key.
- Each region of the table has sorted row key space - [start key, end key|.
- Everything in HBase tables is stored as a byte[ ].
- Atomicity is guaranteed only at a row level. No multi-row transactions.
- Column families have to be defined up front at table creation time.
- Column qualifiers are dynamic and can be defined at write time. They are stored as byte[ ] so you can even put data in them.
- All data model operations return data in sorted order:
    - *row key, ColumnFamily, column qualifier, timestamp desc*

REGIONS

- Tables are automatically partitioned horizontally into regions.
- Each region comprises a subset of a table's rows.
- A region is denoted by
  - Table it belongs to,
  - Its first row, inclusive
  - Last row, exclusive
- Initially, a table comprises a single region,
- As size of the region grows beyond threshold, split into 2 halves
- Until first split, loading is against a single machine
- As the table grows, the number of its regions grows.

PHYSICAL STORE

```
/ apps/ hbase/ data/ data/ default
    /<Table>              (Tables in the cluster)
        /<Region>           (Regions for the table)
            /<ColumnFamily>      (ColumnFamilies for the Region for the table)
                /<StoreFile>       (StoreFiles for the ColumnFamily for the Regions for the table)
```

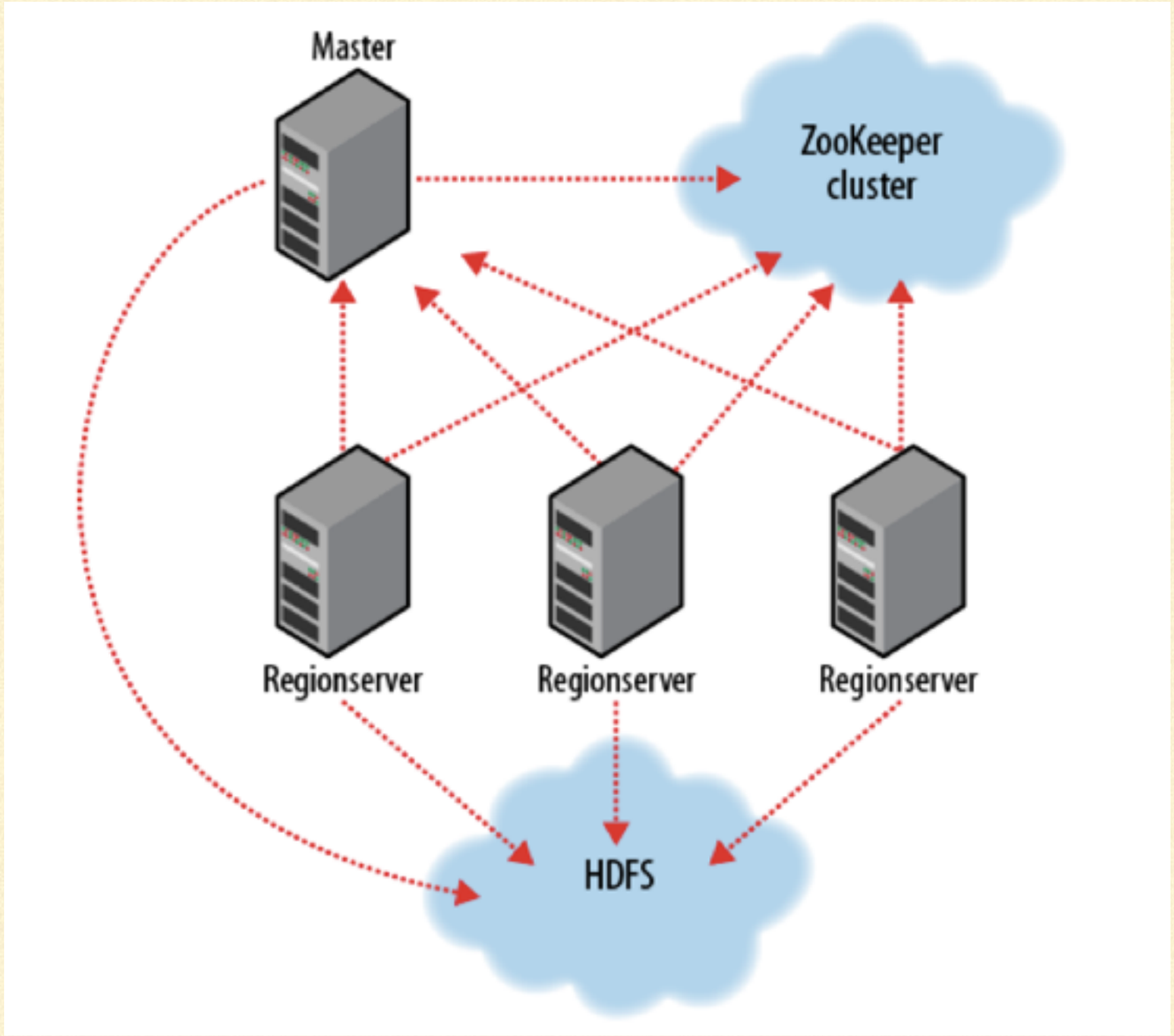
LOCKING

Row updates are atomic, no matter how many row columns constitute the row-level transaction. This keeps the locking model simple.

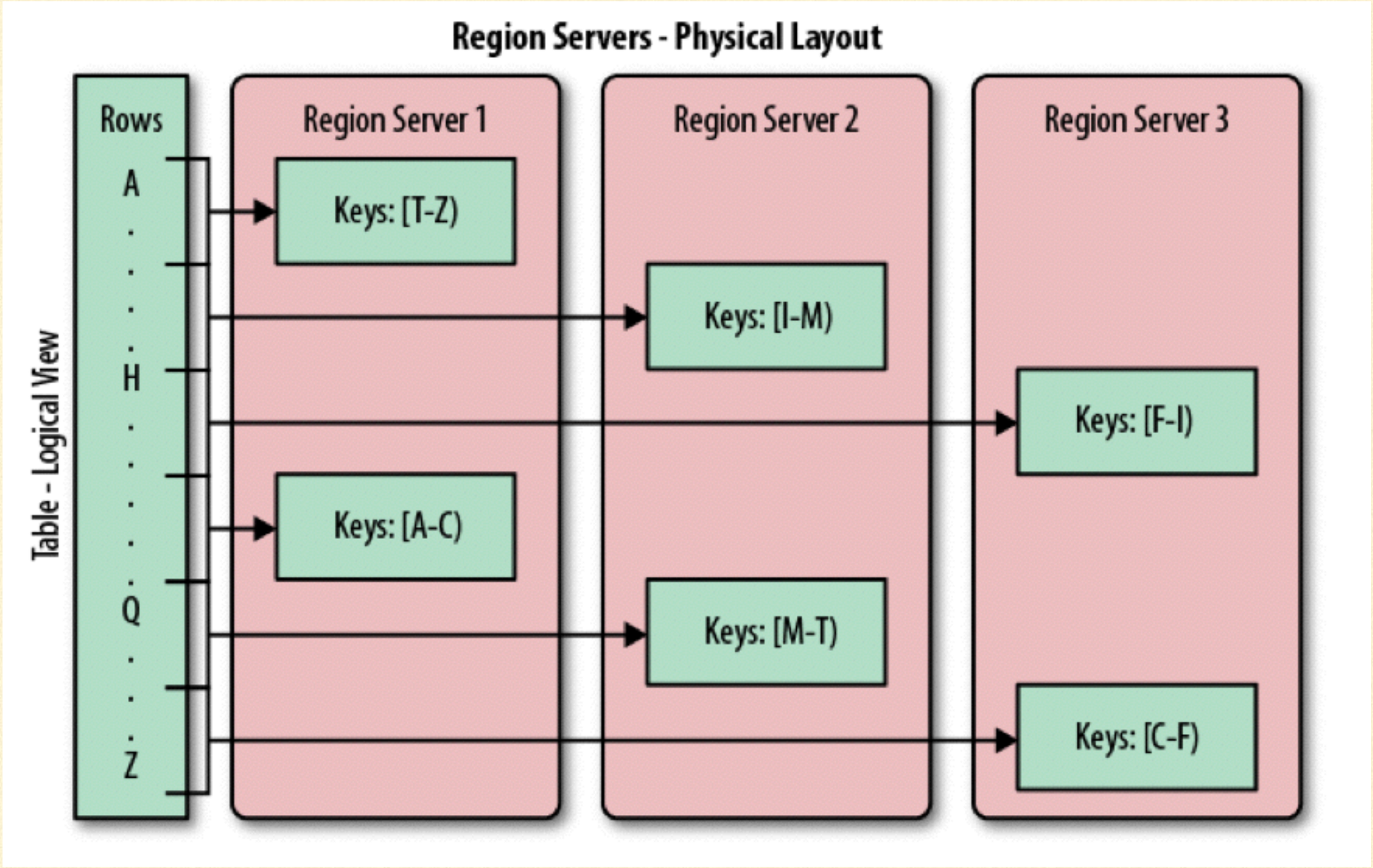
SCANNERS

HBase scanners are like cursors in a traditional database.
They have to be closed after use.
Scanners return rows in order.
Users obtain scanner by ResultScanner rs = HTable.getScanner(scan)
Where Scan parameter:
  row start of the scan
  row stop to scan
  which columns to return
  (optionally), a filter to run on the server side
Each invocation of ***next***(nRows) is a trip back to regionserver

ARCHITECTURE



REGION SERVERS



CLIENTS - JAVA

```
Configuration config = HBaseConfiguration.create();
// Create table
HBaseAdmin admin = new HBaseAdmin(config);
HTableDescriptor htd = new HTableDescriptor("test");
HColumnDescriptor hcd = new HColumnDescriptor("data"); htd.addFamily(hcd);
admin.createTable(htd);
byte [] tablename = htd.getName();
HTableDescriptor [] tables = admin.listTables();
if (tables.length != 1 && Bytes.equals(tablename, tables[0].getName())) {
throw new IOException("Failed create of table"); }
// Run some operations -- a put, a get, and a scan -- against the table. HTable table = new
HTable(config, tablename);
byte [] row1 = Bytes.toBytes("row1");
Put p1 = new Put(row1);
byte [] databytes = Bytes.toBytes("data");
p1.add(databytes, Bytes.toBytes("1"), Bytes.toBytes("value1")); table.put(p1);
Get g = new Get(row1);
…
```

CLIENTS - MAPREDUCE - MAPPER

```
public static class MyMapper extends TableMapper<Text, IntWritable>  {
    public static final byte[] CF = "cf".getBytes();
    public static final byte[] ATTR1 = "attr1".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws
IOException, InterruptedException {
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

CLIENTS - MAPREDUCE - REDUCER

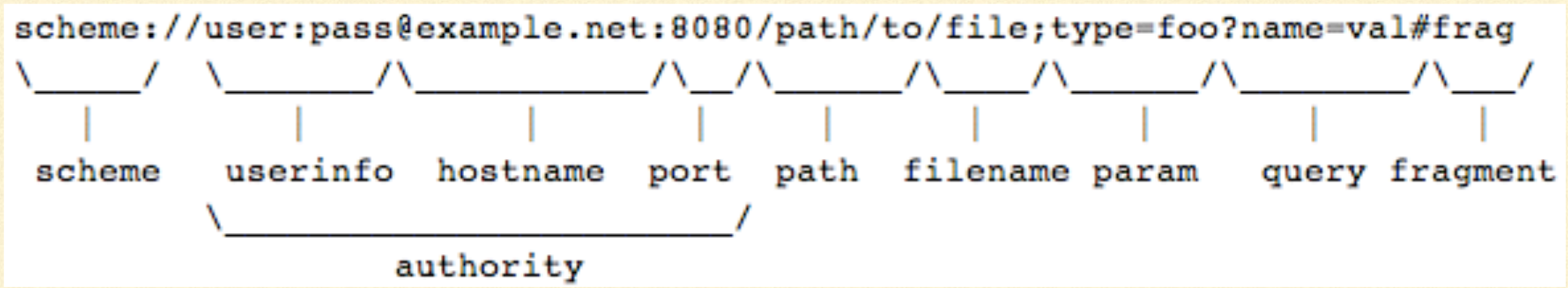
```
public static class MyTableReducer extends TableReducer<Text, IntWritable,
ImmutableBytesWritable>  {
    public static final byte[] CF = "cf".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.add(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

CLIENTS - MAPREDUCE - STITCH

```
Configuration config = HBaseConfiguration.create();
Job job = new Job(config,"ExampleSummary");
job.setJarByClass(MySummaryJob.class);     // class that contains mapper and reducer

Scan scan = new Scan();
scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
scan.setCacheBlocks(false);  // don't set to true for MR jobs
// set other scan attrs

TableMapReduceUtil.initTableMapperJob(
        sourceTable,        // input table
        scan,               // Scan instance to control CF and attribute selection
        MyMapper.class,     // mapper class
        Text.class,         // mapper output key
        IntWritable.class,  // mapper output value
        job);
TableMapReduceUtil.initTableReducerJob(
        targetTable,        // output table
        MyTableReducer.class,    // reducer class
        job);
job.setNumReduceTasks(1);   // at least one, adjust as required

boolean b = job.waitForCompletion(true);
if (!b) {
        throw new IOException("error with job!");
}
```

CLIENTS - REST

Start: *hbase-daemon.sh start rest*; **Stop**: *hbase-daemon.sh stop rest*



GET:
path := '/' <table>
'/' <row>
('/' (<column> (':' <qualifier>)?
(',' <column> (':' <qualifier>)?)+)?
('/' (<start-timestamp> ',')? <end-timestamp>)?)?
query := ('?' 'v' '=' <num-versions>)?

http://localhost:8080/mytable/row1/cf1:colName

PUT:
path := '/' <table> '/' <row> '/' <column> (':' <qualifier>)?
('/' <timestamp>)?
http://localhost:8080/mytable/row1/cf1:colName?value=123

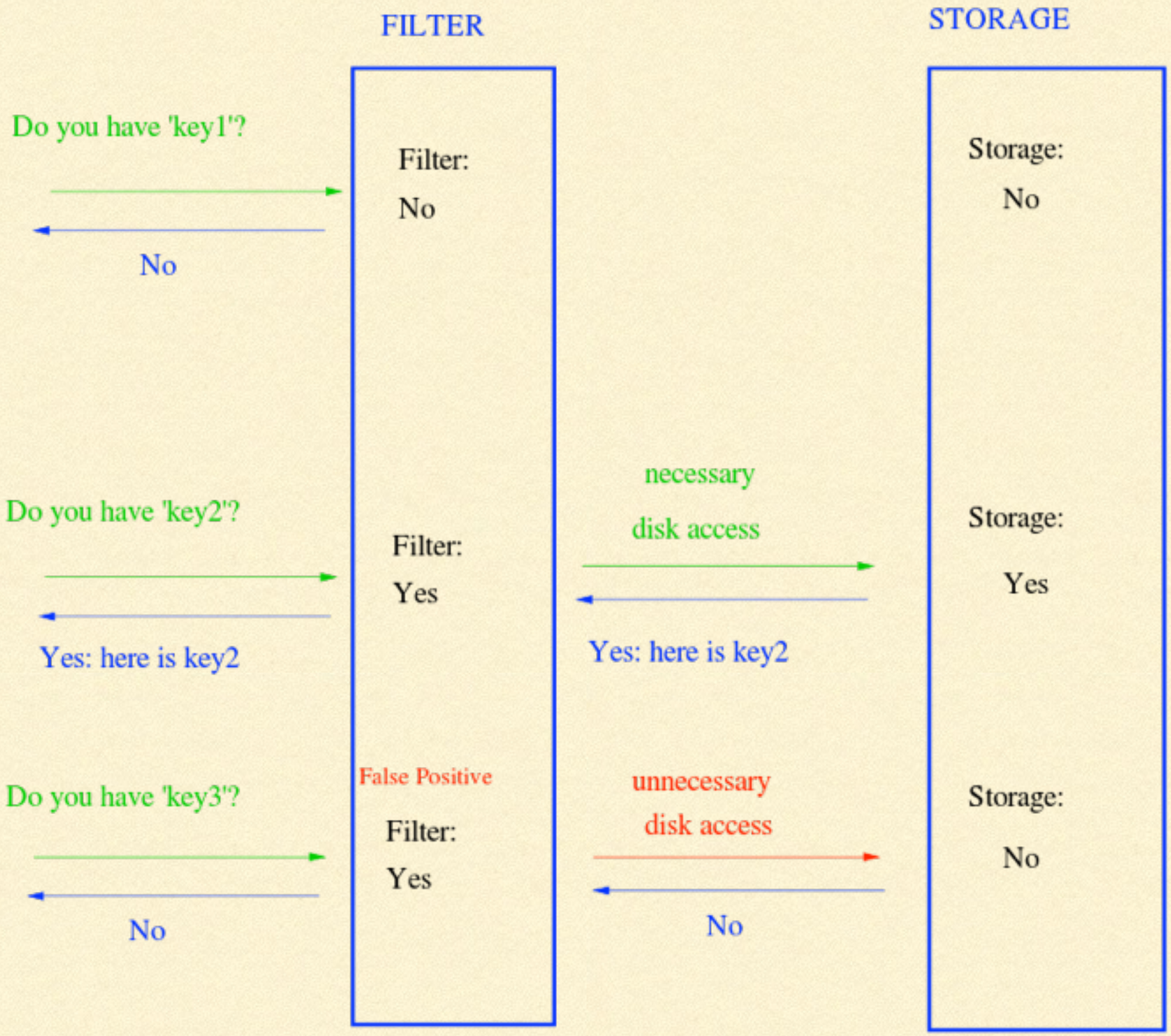
See More at : http://wiki.apache.org/hadoop/Hbase/Stargate

CLIENTS - JDBC

http://phoenix.apache.org/

APACHE Phoenix "We put the SQL back in NoSQL"

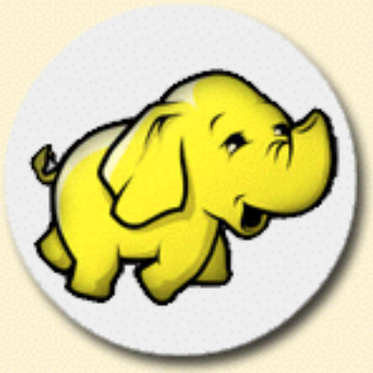
BLOOM FILTER



HBASE

Quick Hello!

- hbase shell
- status
- create 'mytable', 'mycf'
- list
- list '.*table'
- put 'mytable', 'row1','mycf:col1',1234
- scan 'mytable'
- get 'mytable', 'row1', 'mycf:col1'
- describe 'mytable'
- disable 'mytable'
- drop 'mytable'
- See the status at http://hadoop1.knowbigdata.com:60010



Big Data & Hadoop

Thank you.

+91-9538998962

sandeep@knowbigdata.com