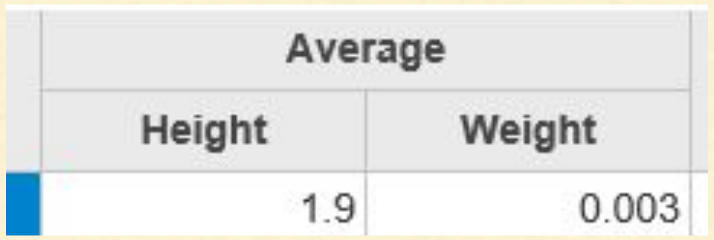
Welcome to HBase

HBase - Quick Definition

Consistent

Partition Tolerant



Average	
Height	Weight
1.9	0.003

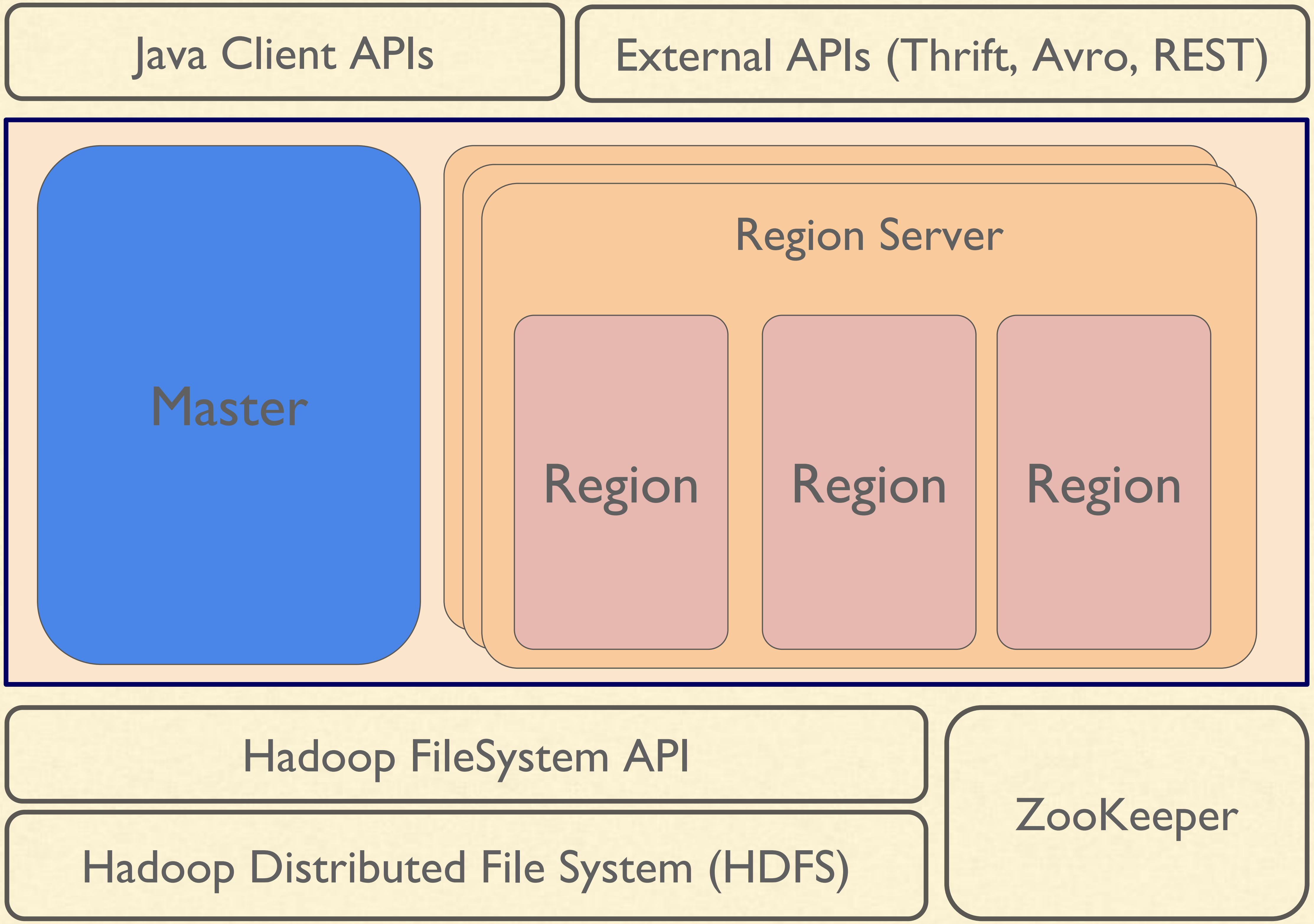
Column Family Oriented

Good for hundreds of millions or billions of rows

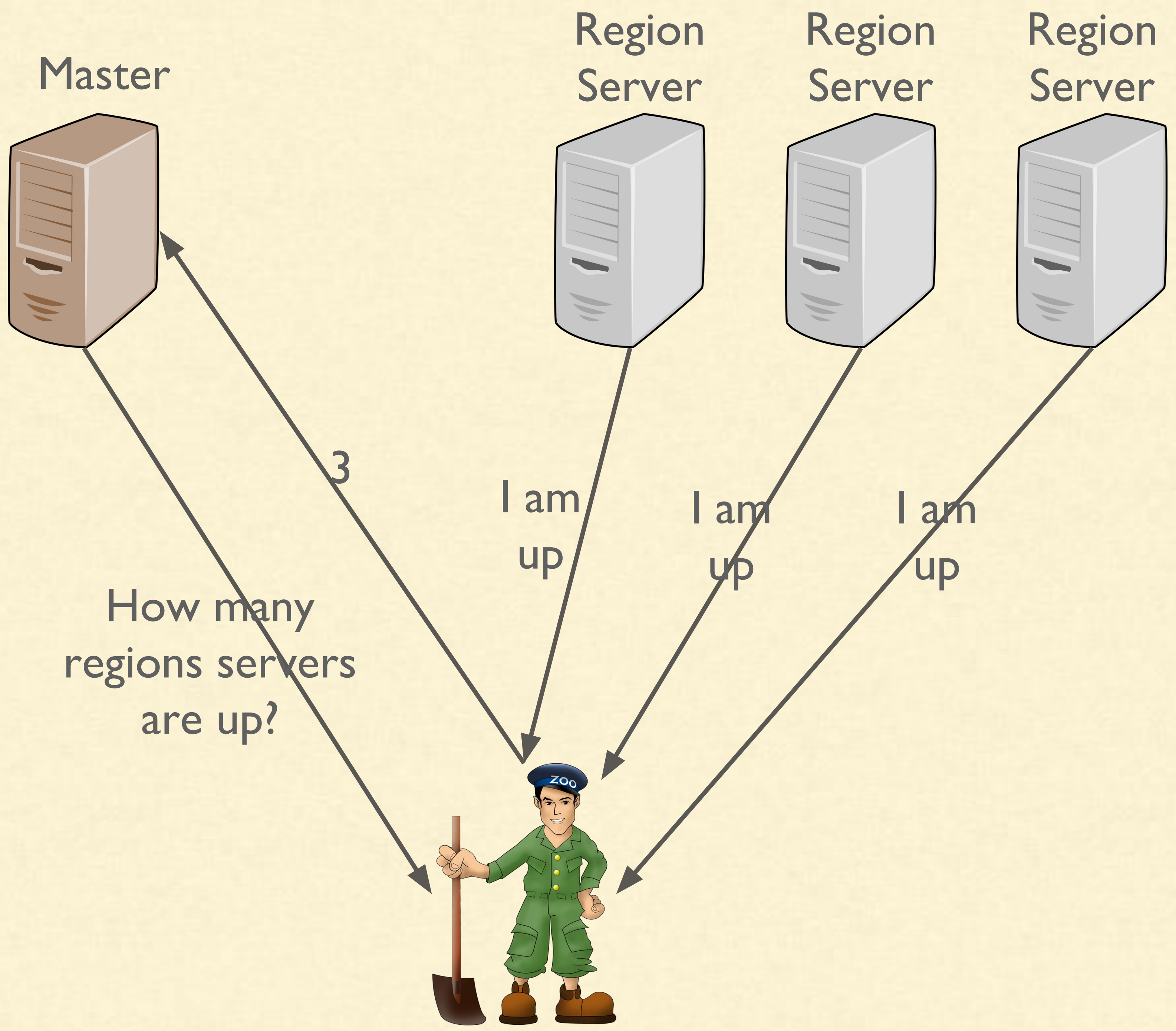
Based on Google's Big Table

Runs on Hadoop / HDFS

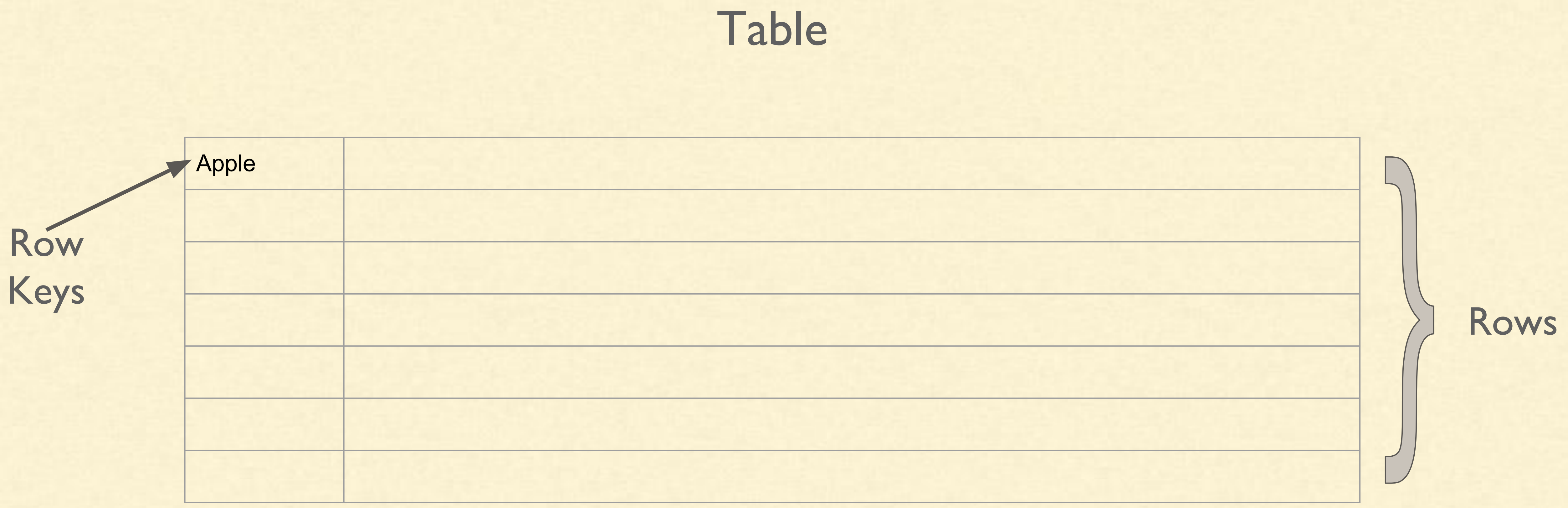
HBase - Architecture - Overview



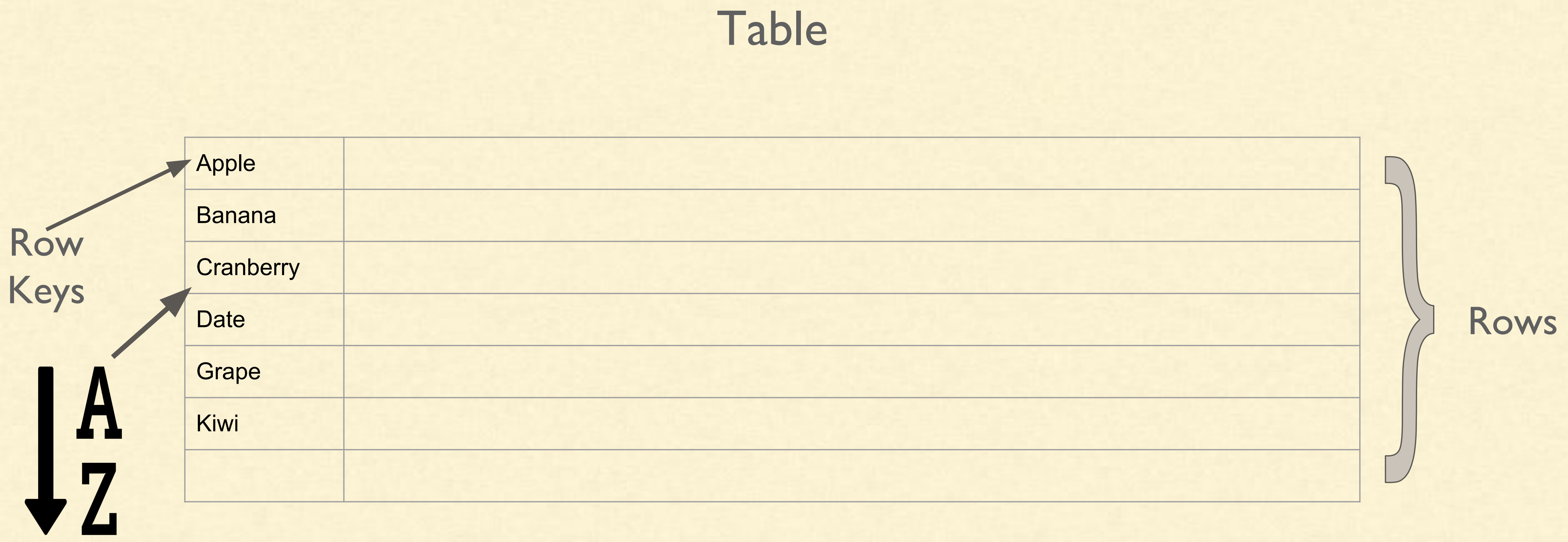
HBase - Architecture



HBase - Architecture

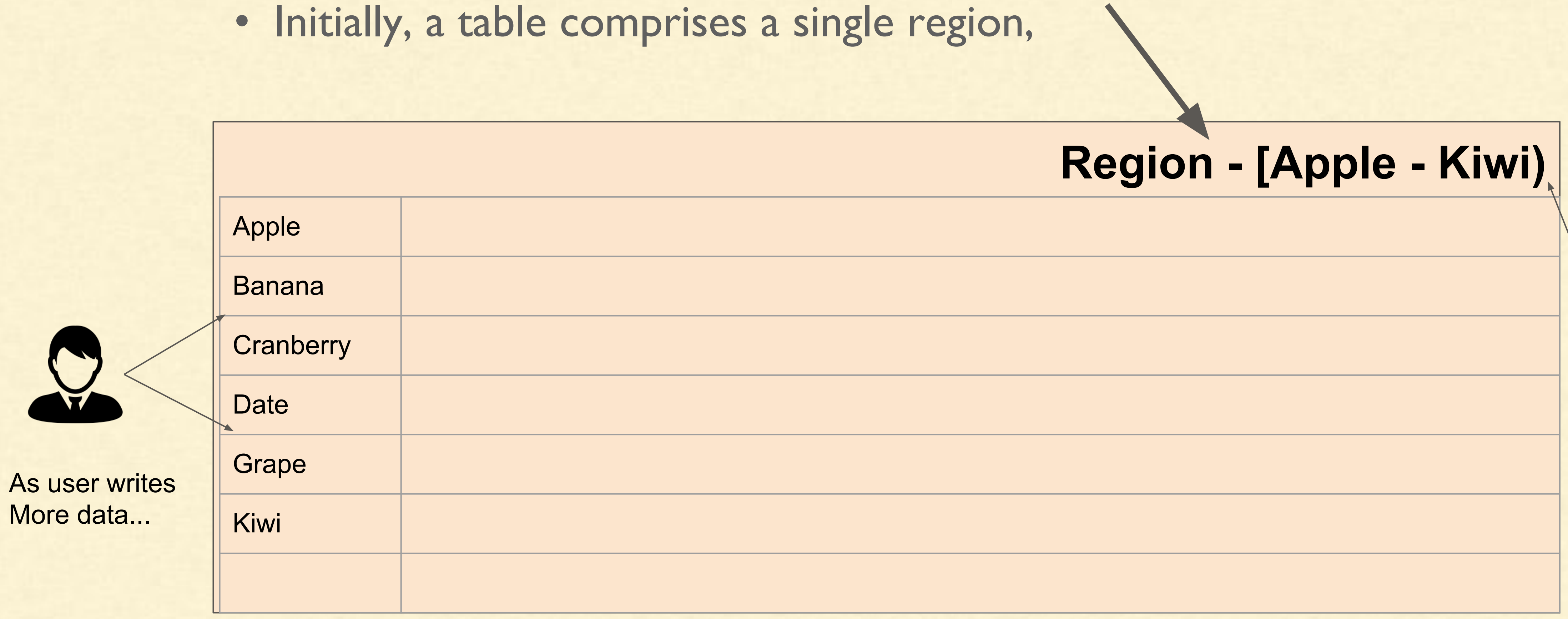


HBase - Architecture



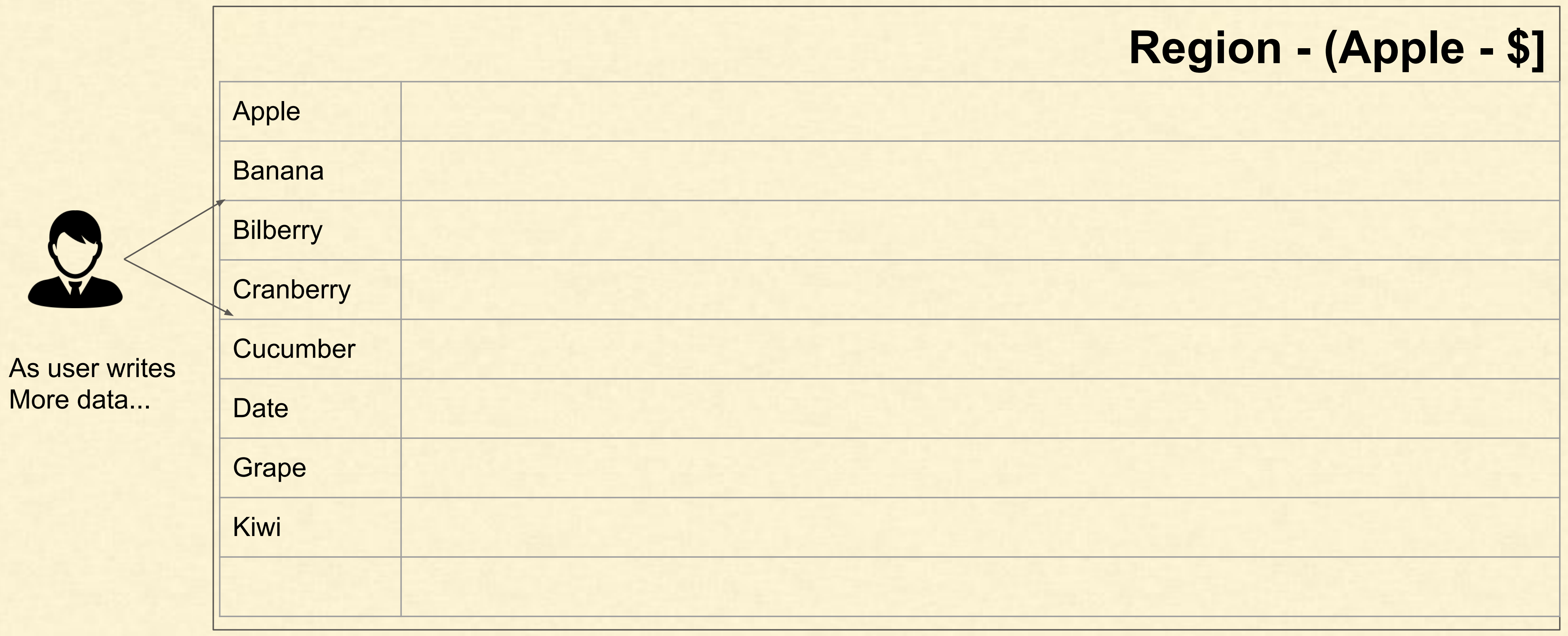
HBase - Architecture

- Tables are automatically partitioned horizontally into regions.
- Each region comprises a subset of a table's rows.
- Initially, a table comprises a single region,

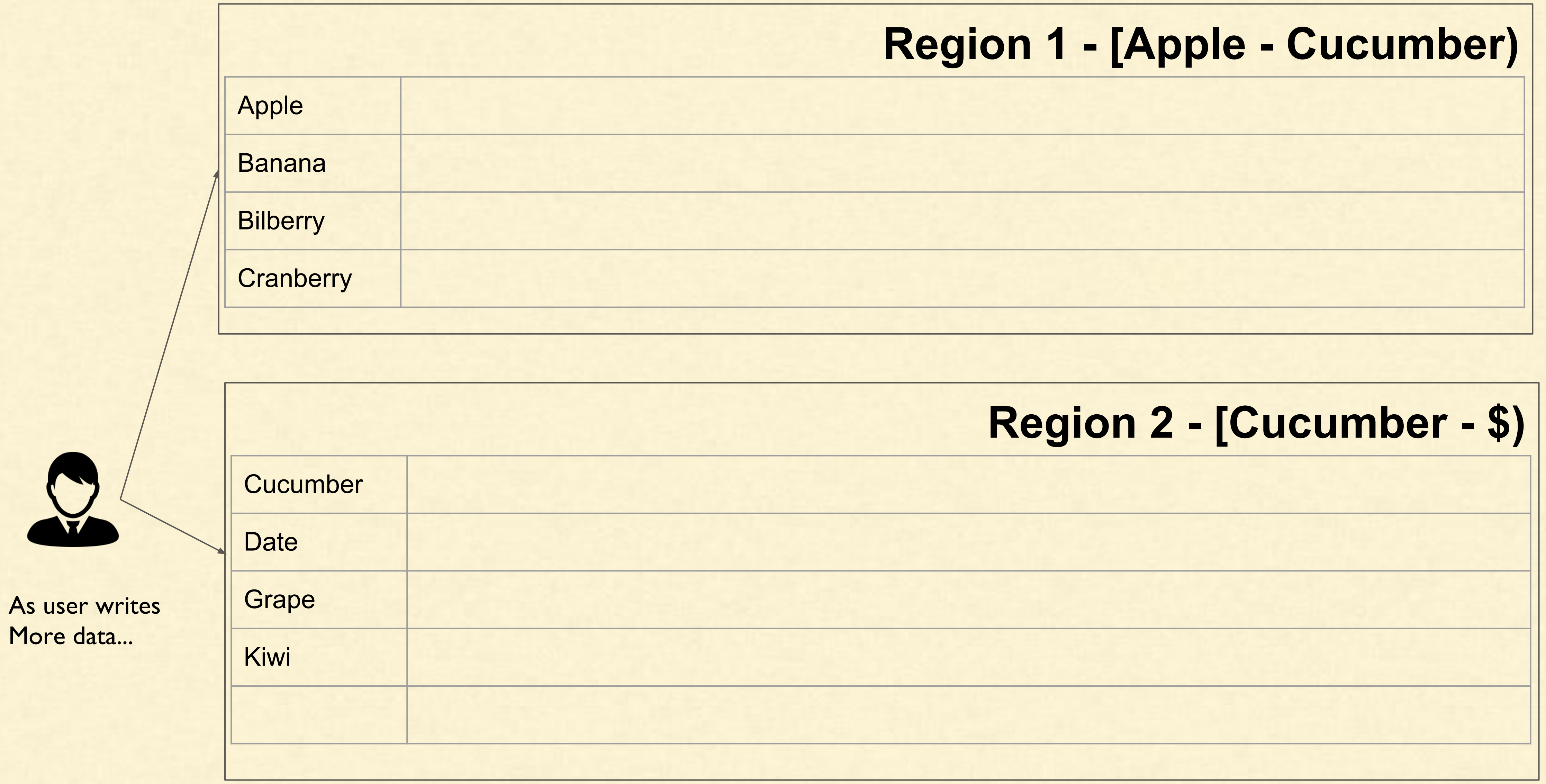


- A region is denoted by
  - Table it belongs to,
  - Its first row, inclusive
  - Last row, exclusive

HBase - Architecture

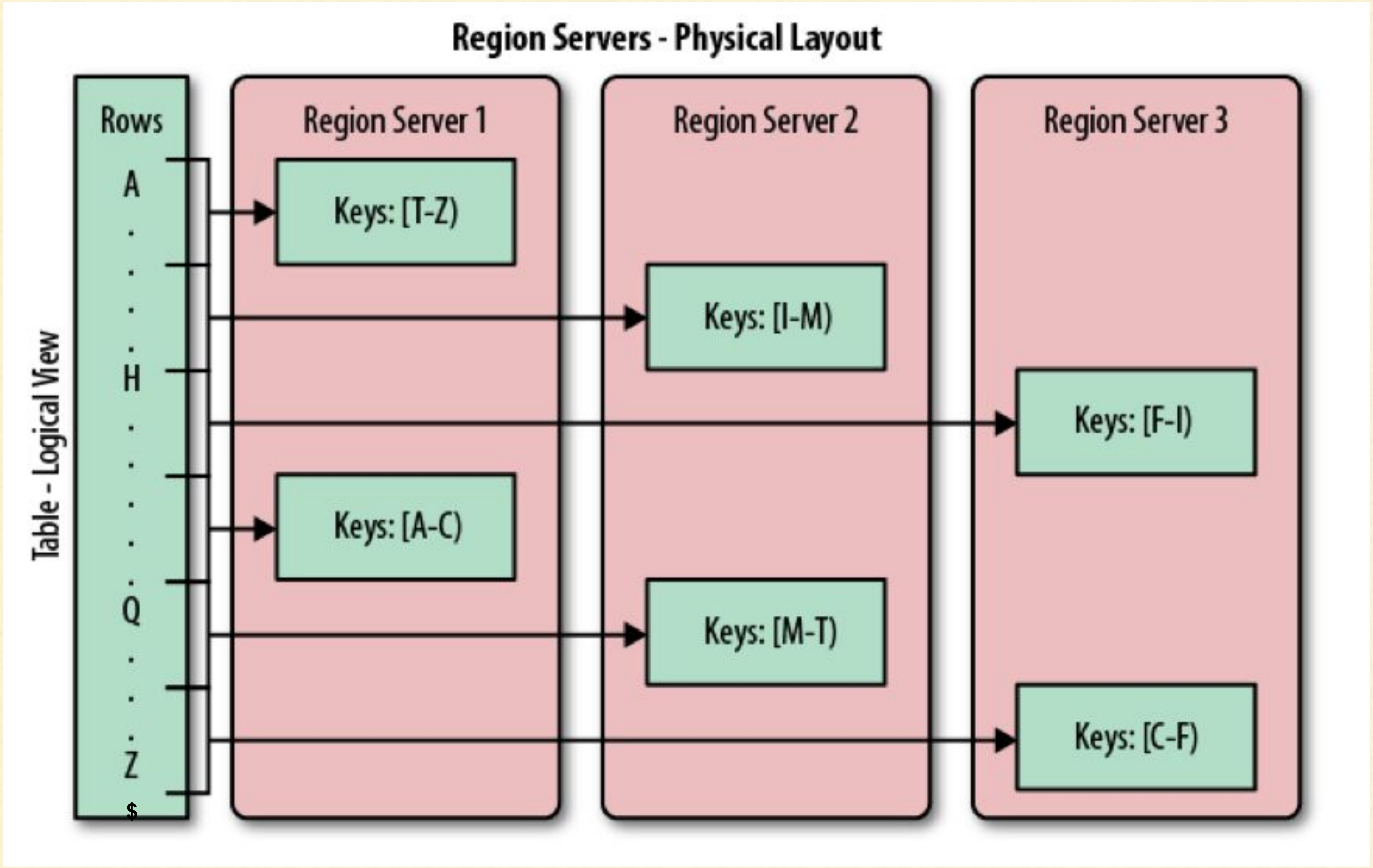


HBase - Architecture



- As size of the region grows beyond threshold, it splits into 2 halves
- As the table grows, the number of its regions grows.

HBase - Region Servers



HBase - Data Model

Data Modeling?

Process of structuring your data using the constructs provided by a datastore to solve your problem.

Requirements

Technology
- Database features
- Limitations

HBase - Data Model

***Based on Bigtable**:*

- *Map*
- *Sorted*
- *Multidimensional*
- *Persistent*
- *Distributed*
- *Sparse*

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

HBase - Data Model

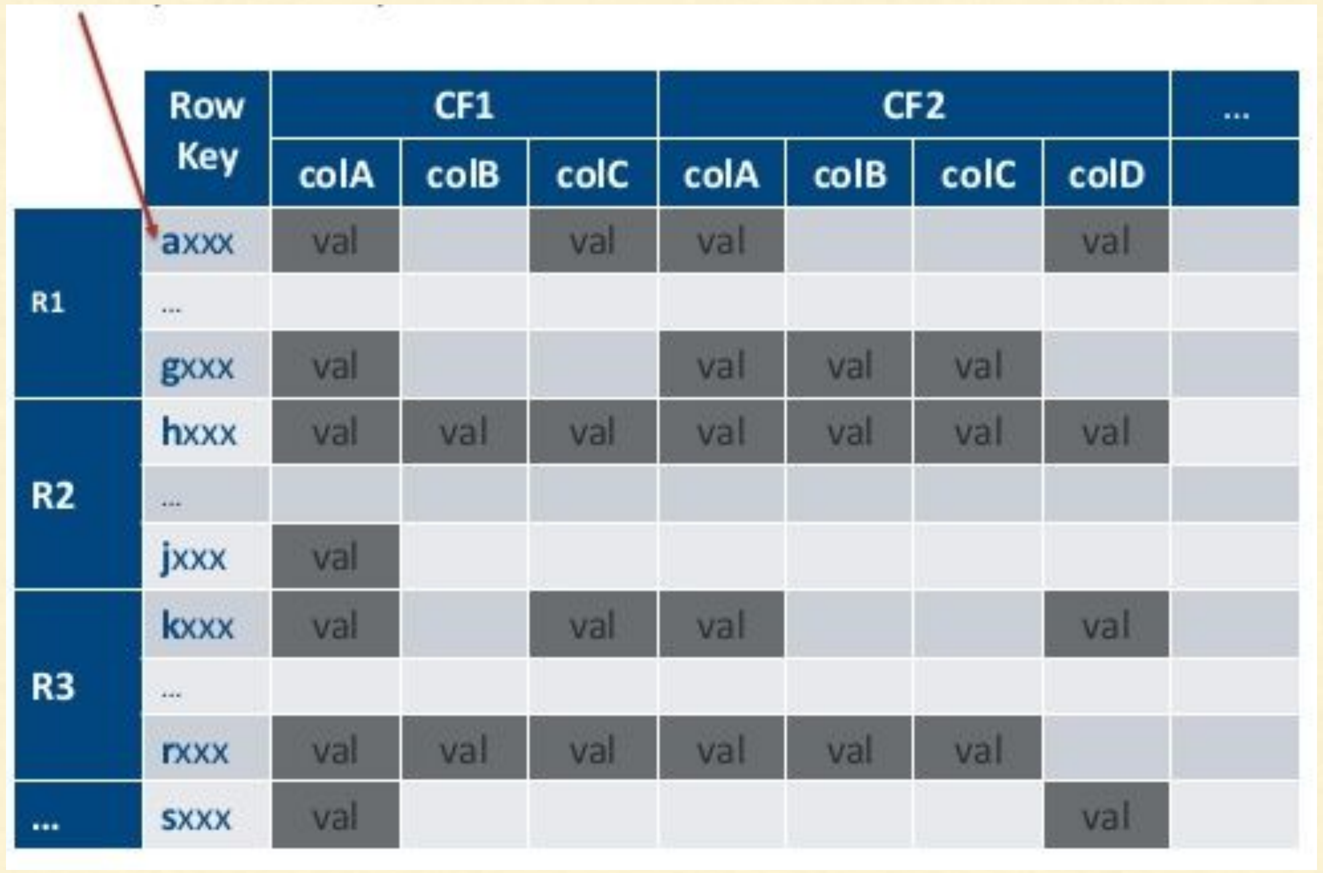
Table:

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

HBase - Data Model

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



	Row Key	CF1			CF2				...
		colA	colB	colC	colA	colB	colC	colD	
R1	axxx	val		val	val			val	
	...								
	gxxx	val			val	val	val		
R2	hxxx	val	val	val	val	val	val	val	
	...								
	jxxx	val							
R3	kxxx	val		val	val			val	
	...								
	rxxx	val	val	val	val	val	val		
...	sxxx	val						val	

HBase - 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

HBase - Data Model

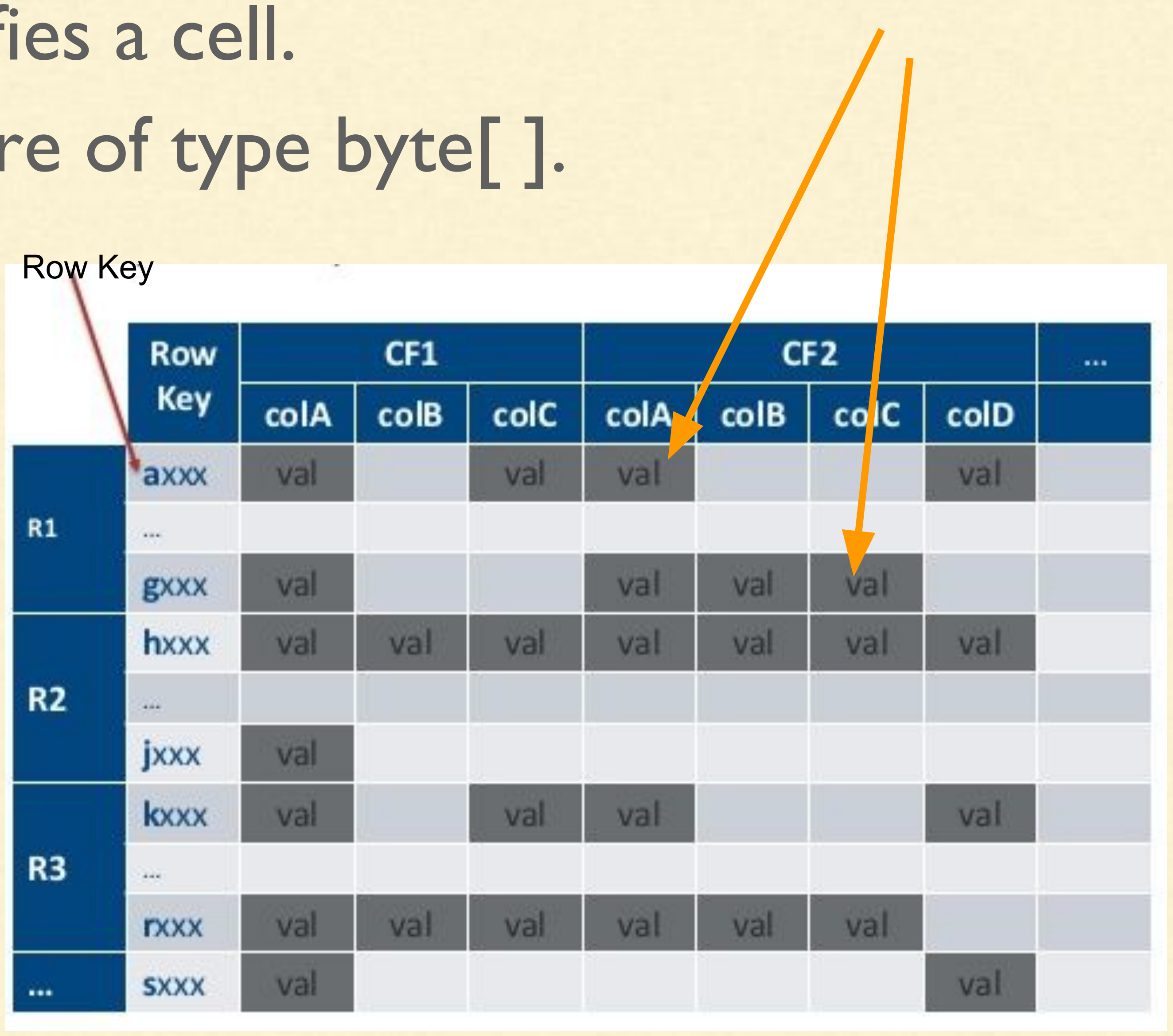
Column Qualifier or Column:

- 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[ ].

HBase - Data Model

Cell:

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



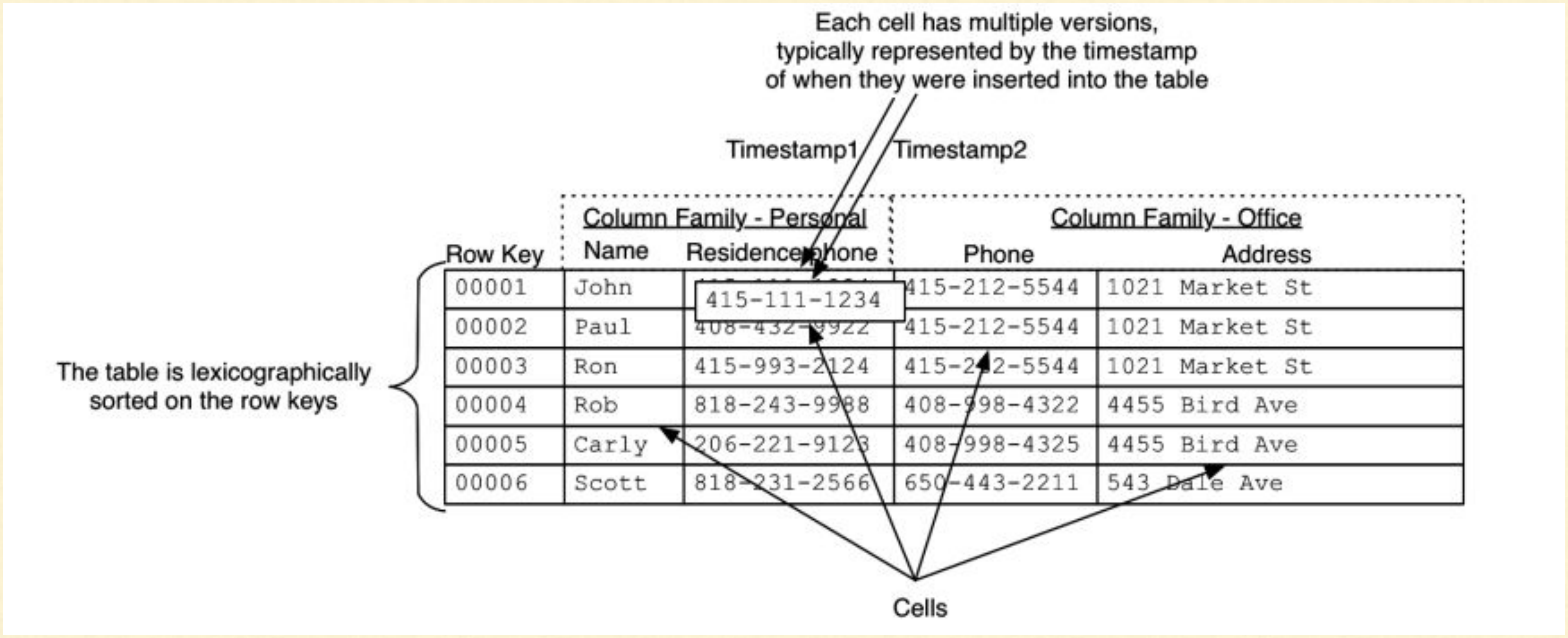
	Row Key	CF1			CF2				...
		colA	colB	colC	colA	colB	colC	colD	
R1	axxx	val		val	val			val	
	...								
	gxxx	val			val	val	val		
R2	hxxx	val	val	val	val	val	val	val	
	...								
	jxxx	val							
R3	kxxx	val		val	val			val	
	...								
	rxxx	val	val	val	val	val	val		
...	sxxx	val						val	

HBase - Data Model

Version / 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

HBase - Data Model - Logical View



Each cell has multiple versions, typically represented by the timestamp of when they were inserted into the table

The table is lexicographically sorted on the row keys

	Column Family - Personal		Column Family - Office	
Row Key	Name	Residence phone	Phone	Address
00001	John	415-111-1234	415-212-5544	1021 Market St
00002	Paul	408-432-9922	415-212-5544	1021 Market St
00003	Ron	415-993-2124	415-212-5544	1021 Market St
00004	Rob	818-243-9988	408-998-4322	4455 Bird Ave
00005	Carly	206-221-9123	408-998-4325	4455 Bird Ave
00006	Scott	818-231-2566	650-443-2211	543 Dale Ave

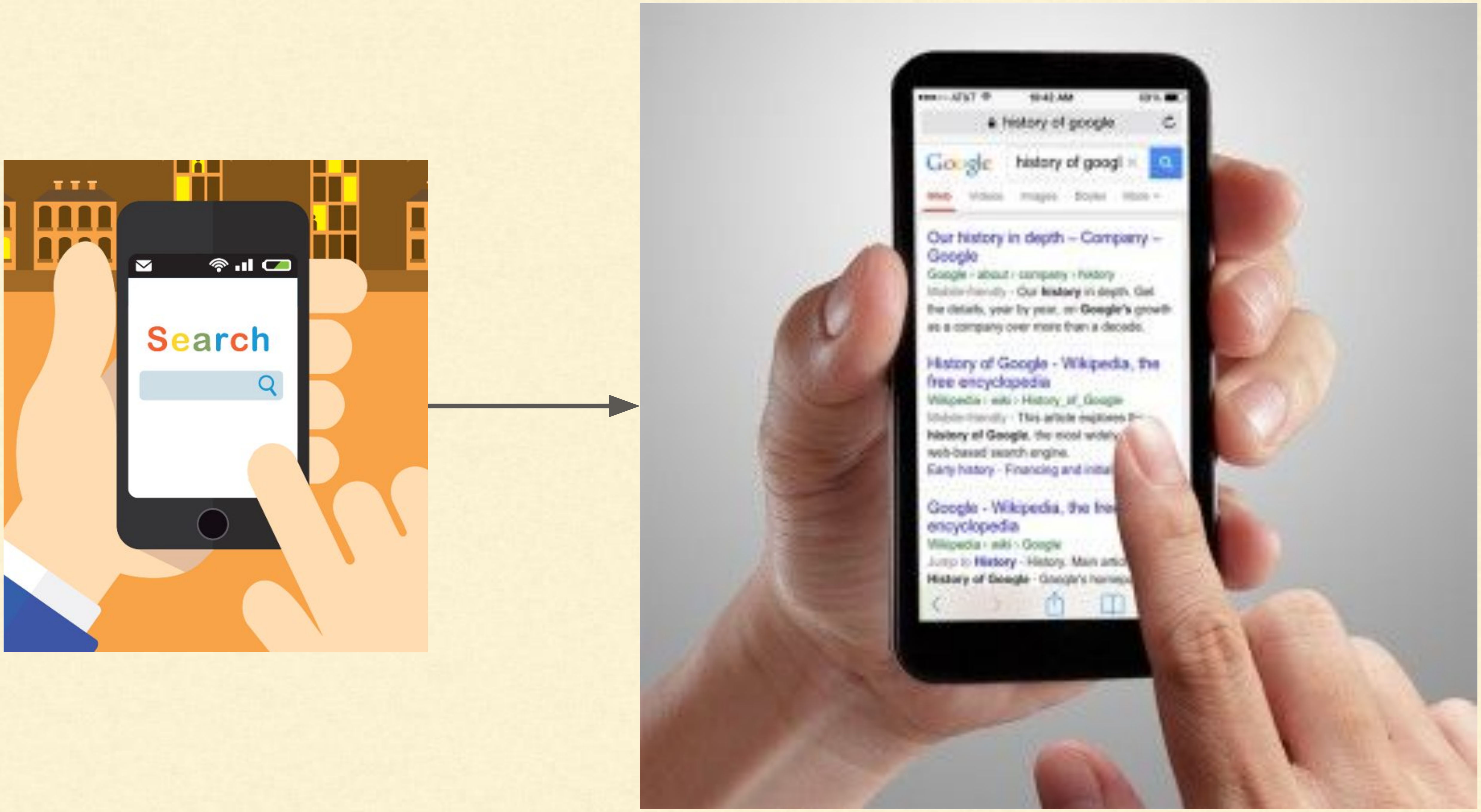
HBase - Data Model - Physical View

```
00001
    Office
        Address
            Timestamp1
                1021 Market
            Timestamp 2
                1021 Market St.
        Phone
            Timestamp1
                415-212-5544
    Personal
        Name
            Timestamp1
                John
        Phone
            Timestamp1
                411-111-1111
            Timestamp2
                415-111-1234
```

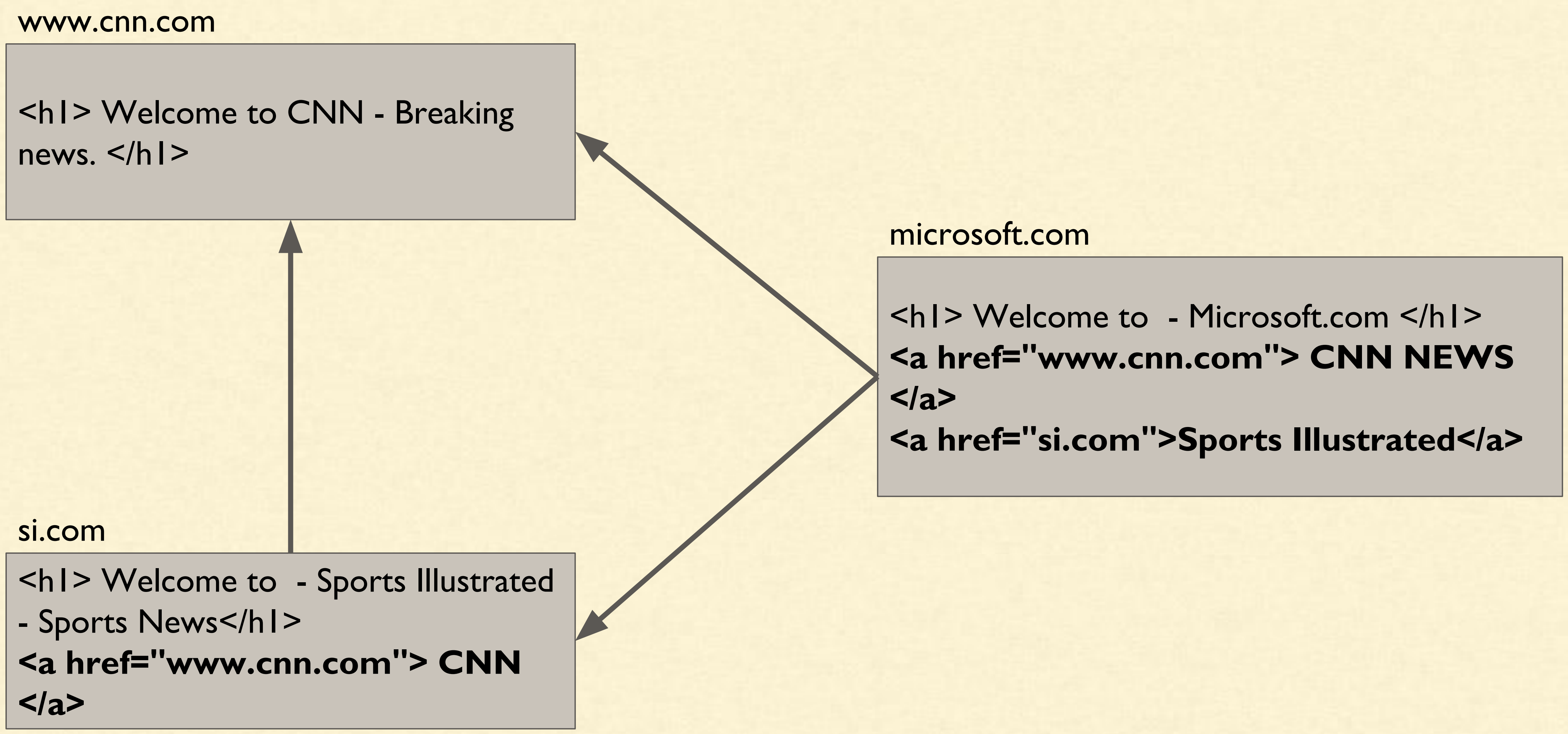
Data Model - Design Guidelines

- Table is indexed based on the Row Key
- A row key can contain any data
- Must to provide the row key.
- We can keep adding columns.
- A column has to be added to a column family
- A column's name which can contain any data
- Atomicity of transaction is guaranteed only at a row level.
- Column families have to be defined up front at table creation time.
- All data model operations return data in sorted order:
    - row key, ColumnFamily, column qualifier, timestamp desc.

Data Model - Example - Webtable - Search Engine


Search
history of google
Google
Our history in depth – Company – Google
History of Google - Wikipedia, the free encyclopedia
Google - Wikipedia, the free encyclopedia

Data Model - Example - Webtable


www.cnn.com
<h1> Welcome to CNN - Breaking news. </h1>
microsoft.com
<h1> Welcome to - Microsoft.com </h1>
<a href="www.cnn.com"> CNN NEWS </a>
<a href="si.com">Sports Illustrated</a>
si.com
<h1> Welcome to - Sports Illustrated - Sports News</h1>
<a href="www.cnn.com"> CNN </a>

Data Model - Example - Webtable

	contents:	anchor:
www.cnn.com	<h1> Welcome to CNN - Breaking news. </h1>	

Data Model - Example - Webtable

	contents:	anchor:
www.cnn.com	<h1> Welcome to CNN - Breaking news. </h1>	
si.com	<h1> Welcome to - Sports Illustrated - Sports News</h1> **<a href="www.cnn.com"> CNN </a>**	

Data Model - Example - Webtable

	contents:	anchor:si.com
www.cnn.com	<h1> Welcome to CNN - Breaking news. </h1>	CNN
si.com	<h1> Welcome to - Sports Illustrated - Sports News</h1> **<a href="www.cnn.com"> CNN </a>**	

Data Model - Example - Webtable

	contents:	anchor:si.com
www.cnn.com	<h1> Welcome to CNN - Breaking news. </h1>	CNN
si.com	<h1> Welcome to - Sports Illustrated - Sports News</h1> **<a href="www.cnn.com"> CNN </a>**	
microsoft.com	<h1> Welcome to - Microsoft.com </h1> **<a href="www.cnn.com"> CNN NEWS </a>** **<a href="si.com">Sports Illustrated</a>**	

Data Model - Example - Webtable

	contents:	anchor:si.com	anchor:microsoft.com
www.cnn.com	<h1> Welcome to CNN - Breaking news. </h1>	CNN	CNN News
si.com	<h1> Welcome to - Sports Illustrated - Sports News</h1> **<a href="www.cnn.com"> CNN </a>**		Sports Illustrated
microsoft.com	<h1> Welcome to - Microsoft.com </h1> **<a href="www.cnn.com"> CNN NEWS </a> <a href="si.com">Sports Illustrated</a>**		

Data Model - Example - Webtable

hdinsights.microsoft.com
learn.cloudxlab.com
mail.cloudxlab.com
outlook.microsoft.com
www.cloudxlab.com
www.microsoft.com

$\rightarrow$

com.cloudxlab.learn
com.cloudxlab.mail
com.cloudxlab.www
com.microsoft.hdinsights
com.microsoft.outlook
com.microsoft.www

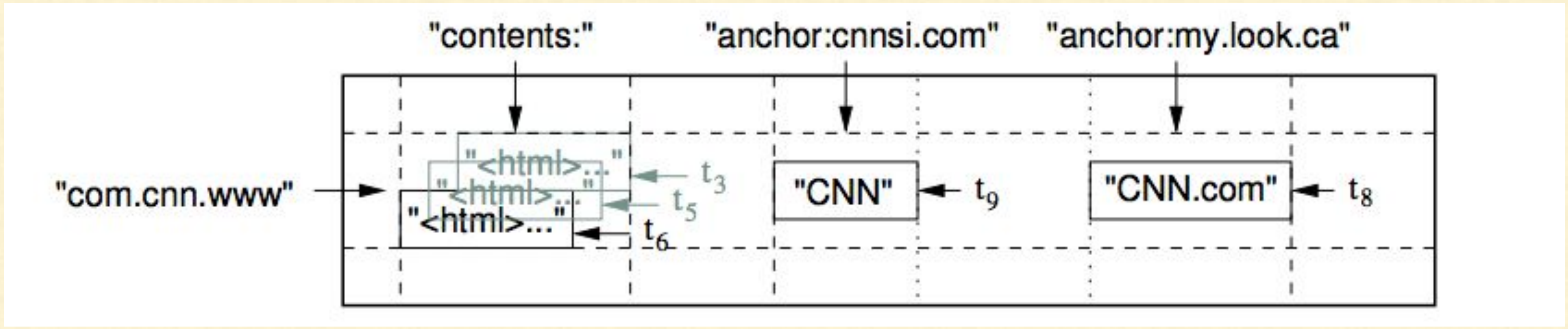
Since the table is sorted by rowkey, reversing the url has brought the rowkeys for same domain together.

Data Model - Example - Webtable

Reversed URL

	contents:	anchor:si.com	anchor:microsoft.com
com.cnn.www	<h1> Welcome to CNN - Breaking news. </h1>	CNN	CNN News
com.microsoft	<h1> Welcome to - Microsoft.com </h1> **<a href="www.cnn.com"> CNN NEWS </a> <a href="si.com">Sports Illustrated</a>**		
com.si	<h1> Welcome to - Sports Illustrated - Sports News</h1> **<a href="www.cnn.com"> CNN </a>**		Sports Illustrated

Data Model - Example - Webtable



Each cell's value could keep past few versions

Hands On

HBase - Shell

Quick Hello!

- hbase shell
- status
- create 'mytable_26nov2017', 'mycf'
- list
- list '.*mytable.*'
- put 'mytable_26nov2017', 'row2','mycf:col1',1234
- put 'mytable_26nov2017', 'row1','mycf:col2',1234
- scan 'mytable_26nov2017'
- get 'mytable_26nov2017', 'row1', 'mycf:col2'
- describe 'mytable_26nov2017'
- disable 'mytable_26nov2017'
- drop 'mytable_26nov2017'
- See the status at http://b.cloudxlab.com:16010

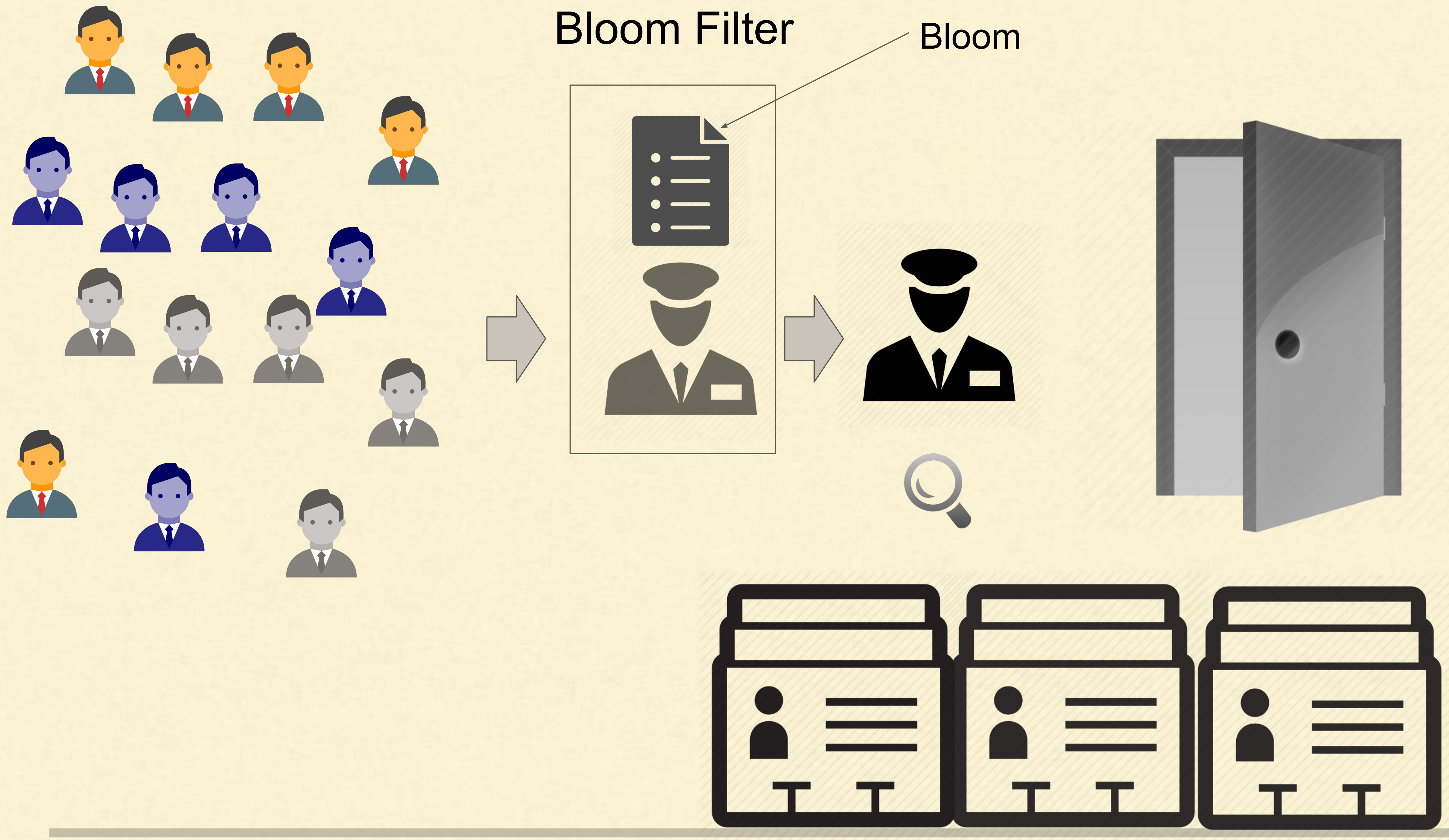
HBase - Physical Store

```
/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)
```

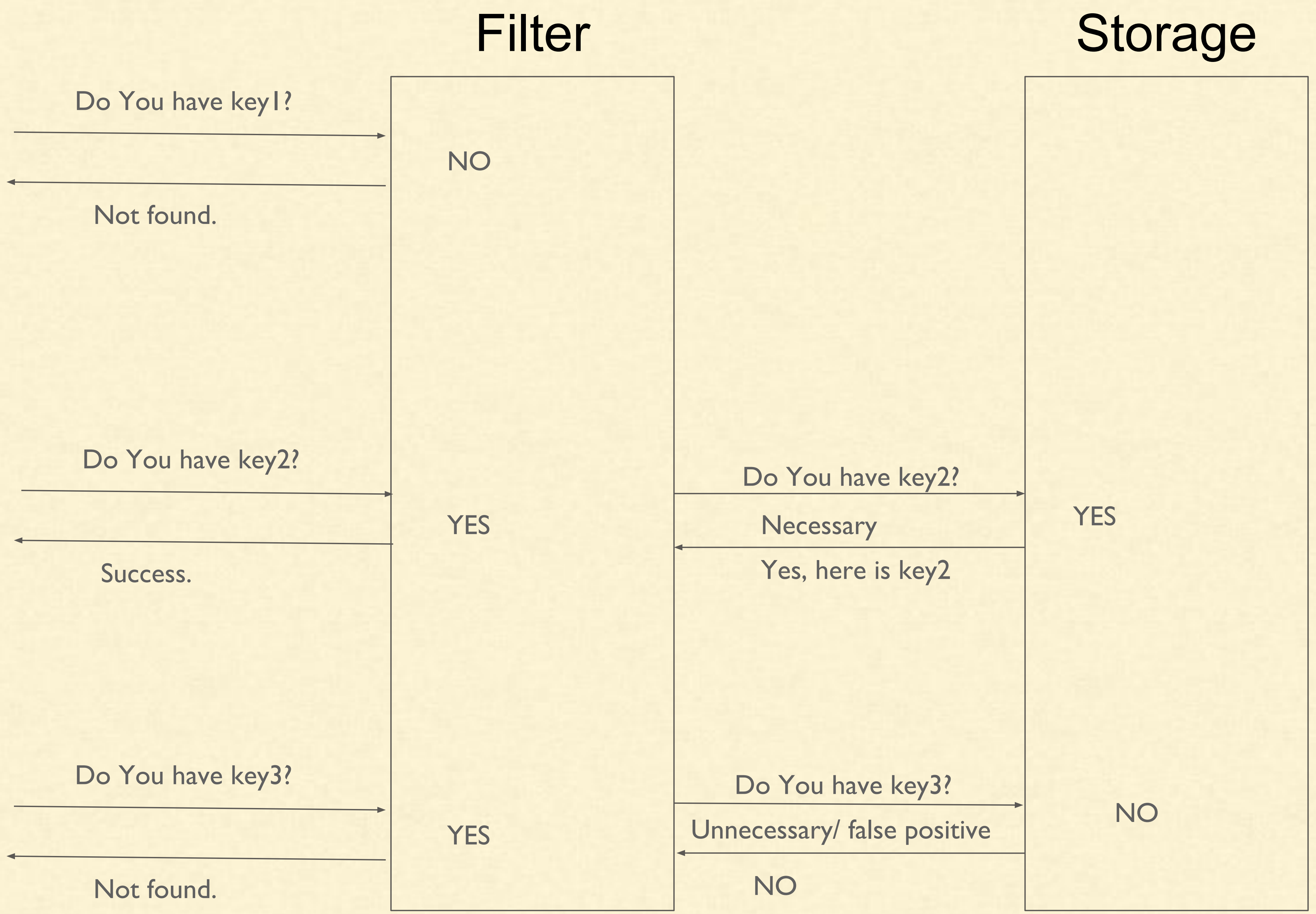
HBase - Bloom Filter

HBase - Bloom Filter

HBase - Bloom Filter



HBase - Bloom Filter



HBase - Bloom

Hashing Function

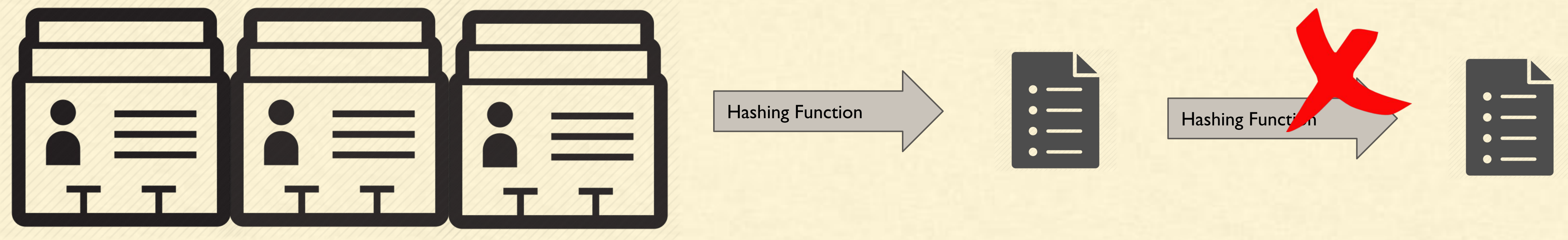
A hashing function converts huge text into fixed size numbers.

A hashing function could be as simple as:

- Sum of the ascii value of all characters % 100
- This value will always be less than 100 no matter how big is the text.
- the bloom will at most contain 100 entries

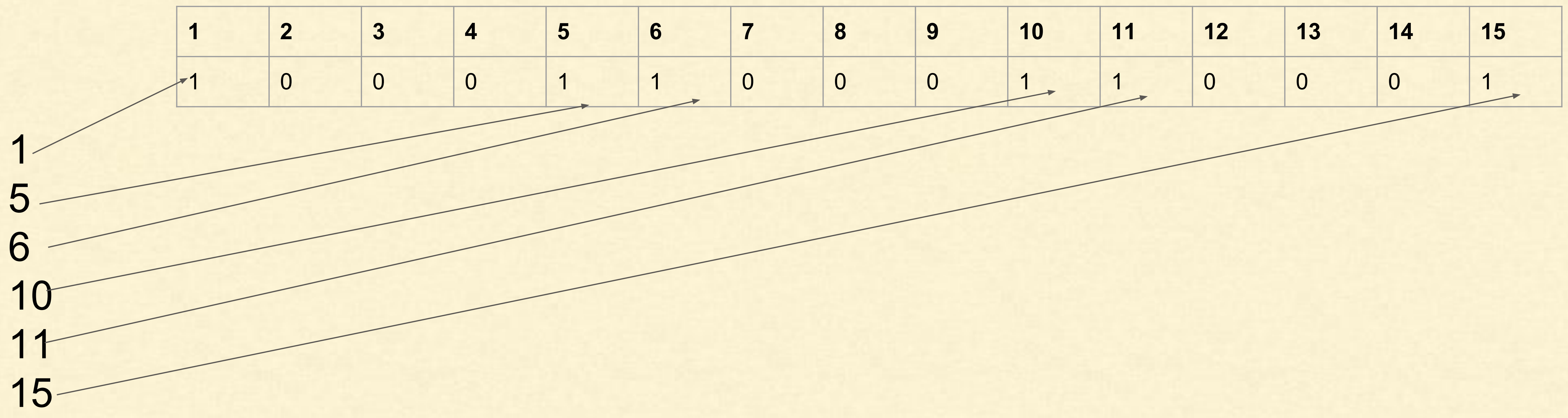
HBase - Bloom

Does it make sense to have more than one bloom filter or chaining bloom filters?



No.
If a bloom filter is slow, instead of chaining another bloom filter, we make existing bloom filter fast by either making bloom smaller, more spread out or increasing system's memory.

HBase - Bloom - Data Structure



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	1	1	0	0	0	1	1	0	0	0	1

A bloom is generally kept in the memory in the form of bit vector or bit array.
For each value that exist in the list we set the bit to true at that location.

HBase - Bloom Filter

- Bloom Filters can be enabled per-ColumnFamily

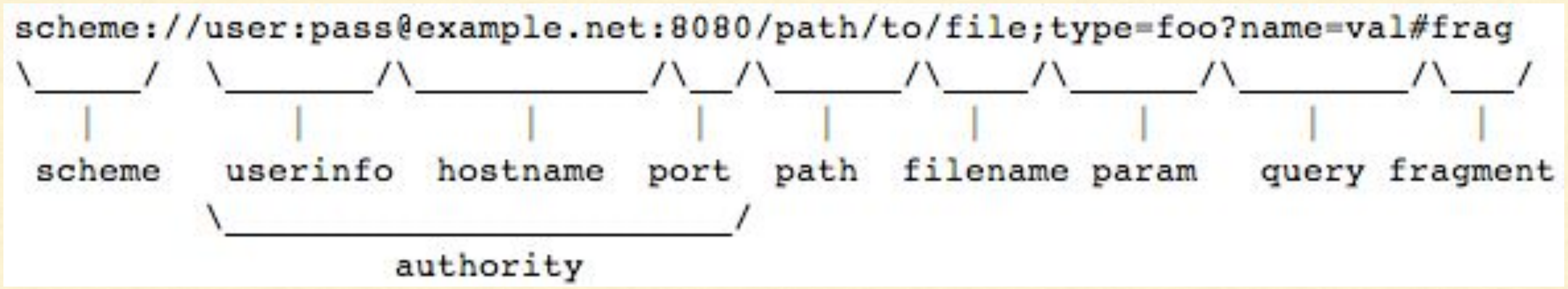
- Use HColumnDescriptor.setBloomFilterType(NONE | ROW | ROWCOL)
    - If NONE, no bloom filter
    - If ROW, the hash of the row key will be added to the bloom on each insert.
    - If ROWCOL, the hash of the row key + column family + column qualifier will be added to the bloom on each key insert.

See More details:

1. https://www.quora.com/How-are-bloom-filters-used-in-HBase
2. http://hbase.apache.org/0.94/book/perf.reading.html#blooms

HBase - REST

RESTful Web services are one way of providing interoperability between computer systems



```
scheme://user:pass@example.net:8080/path/to/file;type=foo?name=val#frag
\_____/  \_______/\___________/\__/\______/\____/\______/\________/\___/
   |         |          |        |      |      |       |       |       |
 scheme  userinfo  hostname   port   path  filename param   query fragment
         \________________________/
                  authority
```

HBase - Clients - REST

Start: *hbase rest start -p 4040*

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

http://e.cloudxlab.com:4040/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

HBase - Clients - JDBC

Hive:

1. Data relatively Static
2. We do have data updation but it is primarily used for querying.
3. Mostly in data ware housing
4. Follows rdbms sort of structure: You don't add columns

HBase:

1. When there are lot of writes.
2. It does not provide any querying capability. you can use map-reduce, hive or sparksql to query.
3. Mostly used in the web application, stream processing
4. columns can be added on the fly just rows.

HBase - Clients - JDBC

http://phoenix.apache.org/

APACHE Phoenix "We put the SQL back in NoSQL"

Thank you!

HBase - 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.

HBase - 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 - 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

HBase - Clients - Java

```
Configuration config = HBaseConfiguration.create();
// Create table
HBaseAdmin admin = new HBaseAdmin(config);
HTableDescriptor htd = new HTableDescriptor("test");
HColumnDescriptor hcd = new HColumnDescriptor("cf1"); 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("cf1");
p1.add(databytes, Bytes.toBytes("col1"), Bytes.toBytes("value1")); table.put(p1);
Get g = new Get(row1);
…
```

HBase - Clients - MapReduce - Mapper

```
//NOTE: Creates one mapper per region
//See more at http://hbase.apache.org/0.94/book/mapreduce.html
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);
    }
}
```

HBase - 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);
    }
}
```

HBase - 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!");
}
```

HBase - Coprocessor

1. Create a class & export to the jar

```
package org.apache.hadoop.hbase.coprocessor;
import java.util.List;
import org.apache.hadoop.hbase.KeyValue;
import org.apache.hadoop.hbase.client.Get;
// Sample access-control coprocessor. It utilizes RegionObserver
// and intercept preXXX() method to check user privilege for the given table
// and column family.
public class AccessControlCoprocessor extends BaseRegionObserverCoprocessor {
  // @Override
  public Get preGet(CoprocessorEnvironment e, Get get)
      throws CoprocessorException {
    // check permissions..
    if (access_not_allowed)  {
      throw new AccessDeniedException("User is not allowed to access.");
    }
    return get;
  }
  // override prePut(), preDelete(), etc.
}
```

HBase - Coprocessor

2. Copy to the region server
3. Add it in the hbase config (/etc/hbase/conf/hbase-site.xml):

```
<property>
        <name>hbase.coprocessor.region.classes</name>
    <value>
      mypgk.AccessControlCoprocessor,
      mypgk.ColumnAggregationProtocol
    </value>
    <description></description>
</property>
```

HBase - Coprocessor

OR

3. Load from table attribute

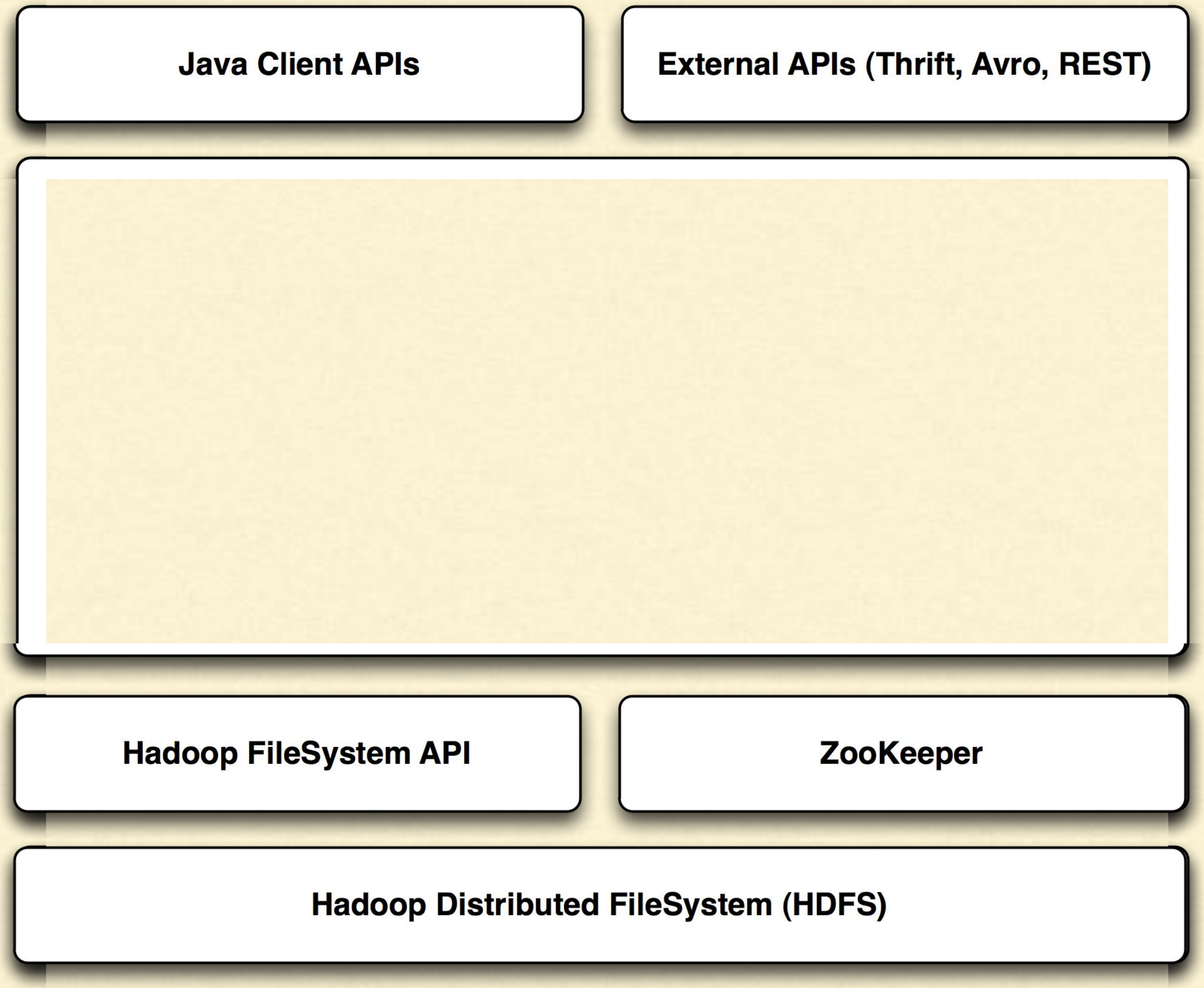
```
Path path = new Path(fs.getUri() + Path.SEPARATOR +
"TestClassloading.jar");

// create a table that references the jar
HTableDescriptor htd = new
HTableDescriptor(TableName.valueOf(getClass().getTableName()));
htd.addFamily(new HColumnDescriptor("test"));
htd.setValue("Coprocessor$1",
path.toString() + ":" + classFullName + ":" + Coprocessor.Priority.USER);
HBaseAdmin admin = new HBaseAdmin(this.conf);
admin.createTable(htd);
```

More details at :
https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/coprocessor/package-summary.html

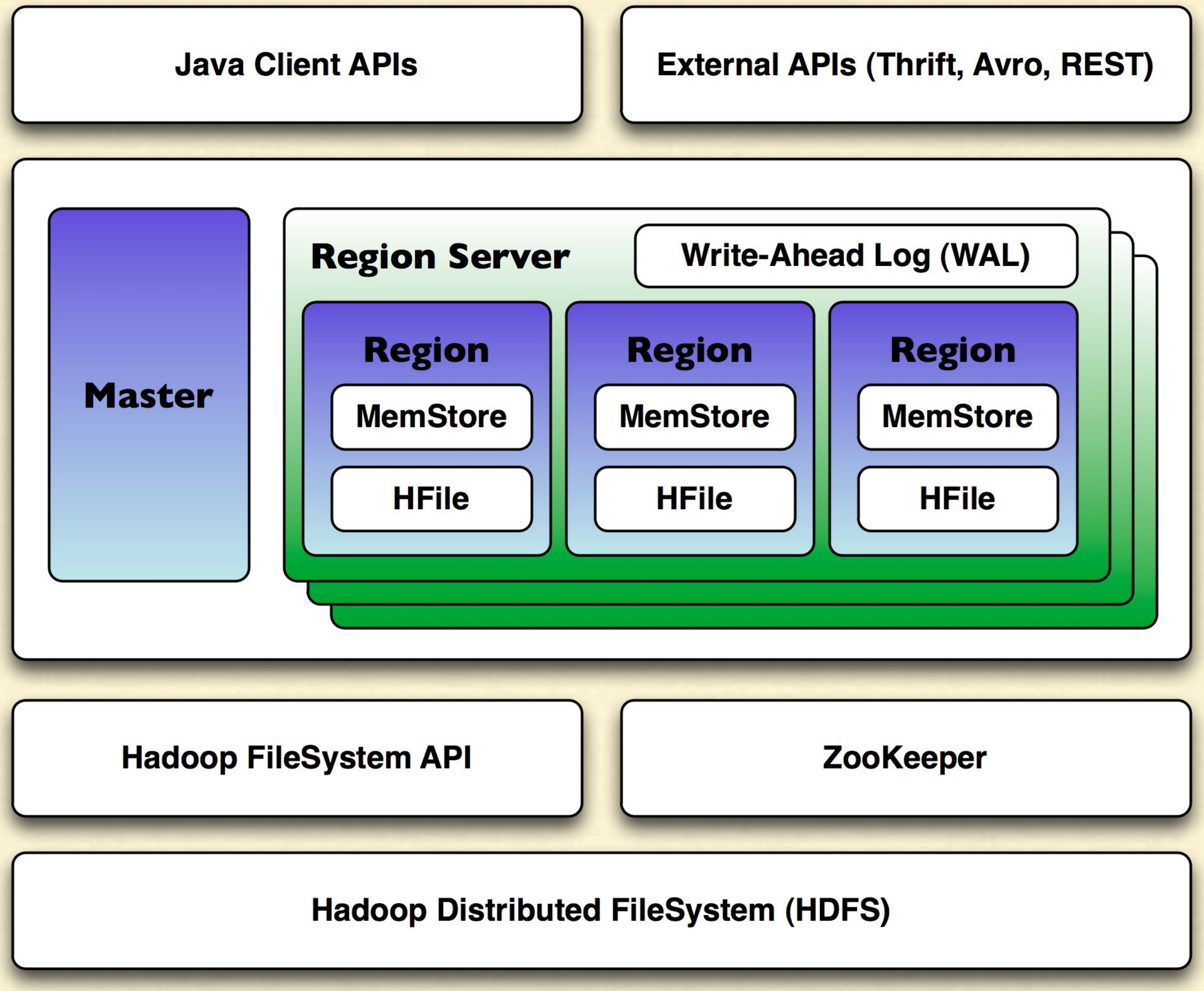
HBase - Quick Definition

- Consistent & Partition Tolerant
- Column Family Oriented NoSQL Database
- Based on Google’s Big Table
- Good for hundreds of millions or billions of rows
- Run on Hadoop/HDFS

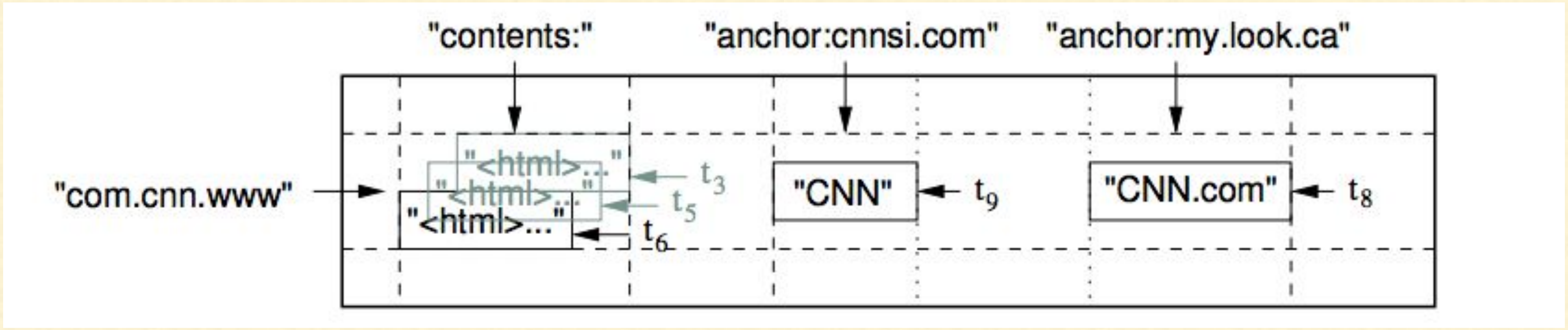
HBase - Architecture


Java Client APIs
External APIs (Thrift, Avro, REST)
Hadoop FileSystem API
ZooKeeper
Hadoop Distributed FileSystem (HDFS)

HBase - Architecture



Data Model - Example - Webtable



An example table that stores Web pages

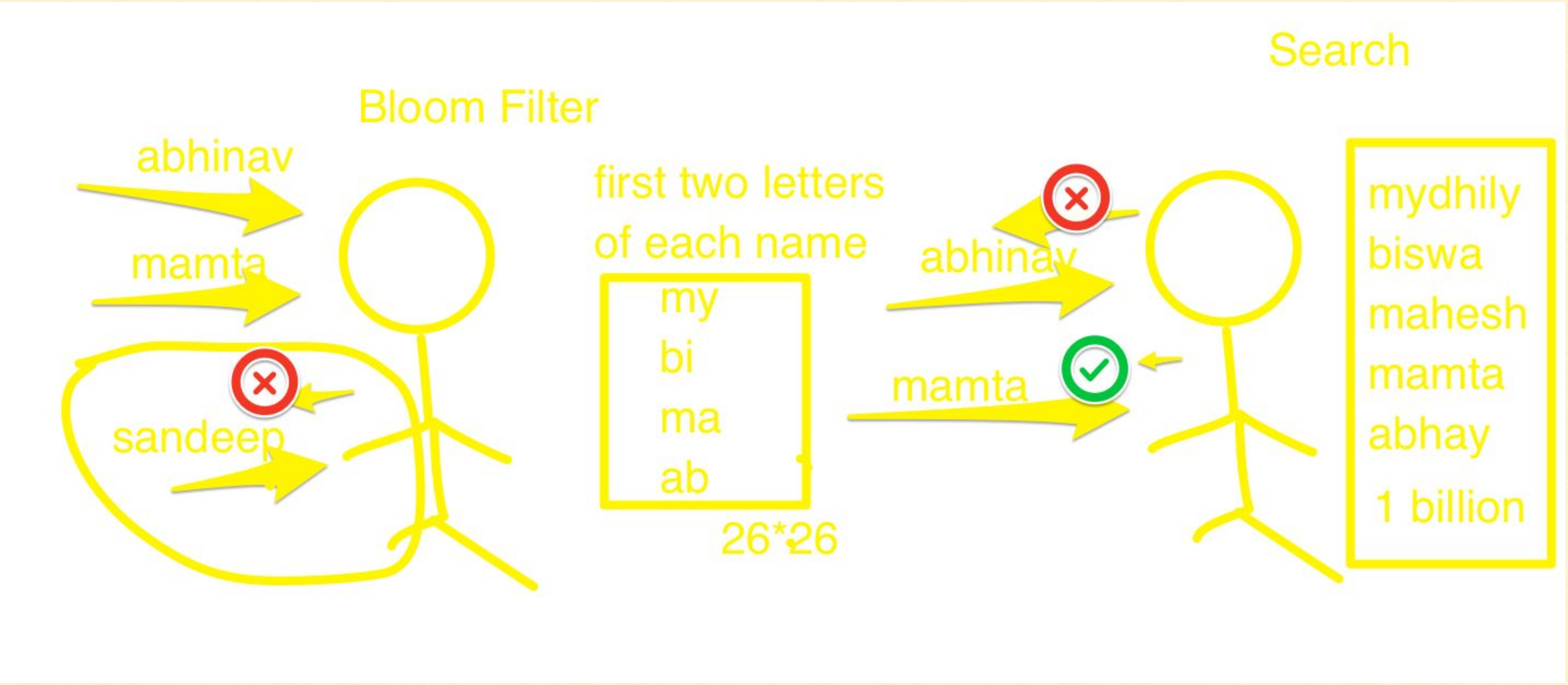
- row key is a reversed URL.
- contents column family contains the page contents. column name is blank
- anchor column family contains the text of any anchors that reference the page.
- CNN's home page is referenced by both the cnnsi.com & MY-look.ca
- Each anchor cell has one version; the contents column 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"

HBase - Shell

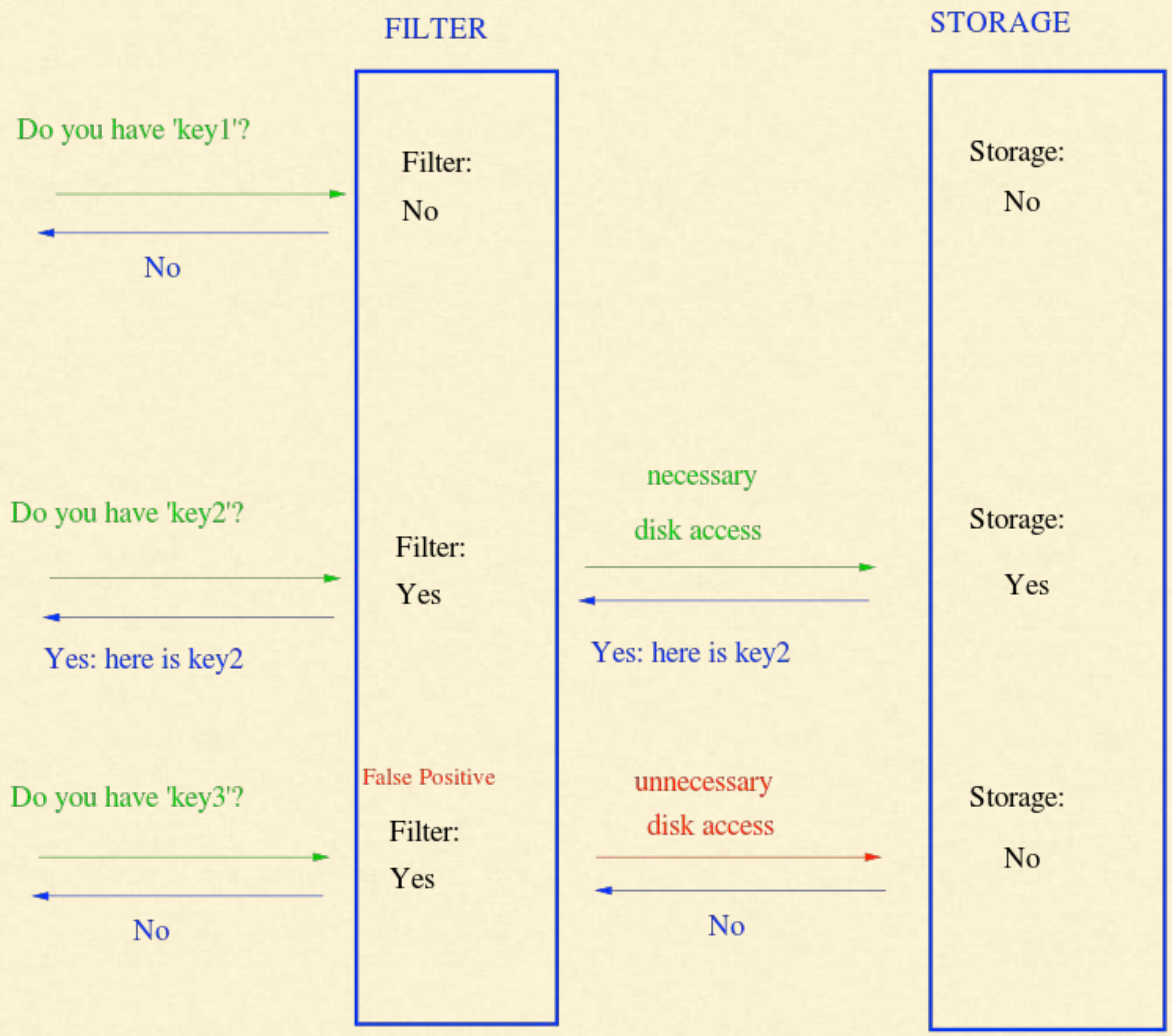
Quick Hello!

- hbase shell
- status
- create 'mytable4june', 'mycf'
- list
- list '.*table'
- put 'mytable4june', 'row2','mycf:col1',1234
- put 'mytable4june', 'row1','mycf:col2',1234
- scan 'mytable4june'
- get 'mytable4june', 'row1', 'mycf:col1'
- describe 'mytable4june'
- disable 'mytable4june'
- drop 'mytable4june'
- See the status at http://b.cloudxlab.com:16010

HBase - Bloom Filter



HBase - Bloom Filter



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*