Apache Hadoop HBASE

Sheetal Sharma

Intern At IBM Innovation Centre

HBase is ..

- A distributed data store that can scale horizontally to 1,000s of commodity servers and petabytes of indexed storage.
- Designed to operate on top of the Hadoop distributed file system (HDFS) or Kosmos File System (KFS, aka Cloudstore) for scalability, fault tolerance, and high availability.

Benefits

- Distributed storage
- Table-like in data structure

  multi-dimensional map
- High scalability
- High availability
- High performance

HBase Is Not …

- Tables have one primary index, the *row key*.
- No join operators.
- Scans and queries can select a subset of available columns, perhaps by using a wildcard.
- There are three types of lookups:
  - Fast lookup using row key and optional timestamp.
  - Full table scan
  - Range scan from region start to end.

HBase Is Not …(2)

- Limited atomicity and transaction support.

  - HBase supports multiple batched mutations of single rows only.

  - Data is unstructured and untyped.

- No accessed or manipulated via SQL.

  - Programmatic access via Java, REST, or Thrift APIs.

  - Scripting via JRuby.

Why HBase ?

- HBase is a Bigtable clone.
- It is open source
- It has a good community and promise for the future
- It is developed on top of and has good integration for the Hadoop platform, if you are using Hadoop already.
- It has a Cascading connector.

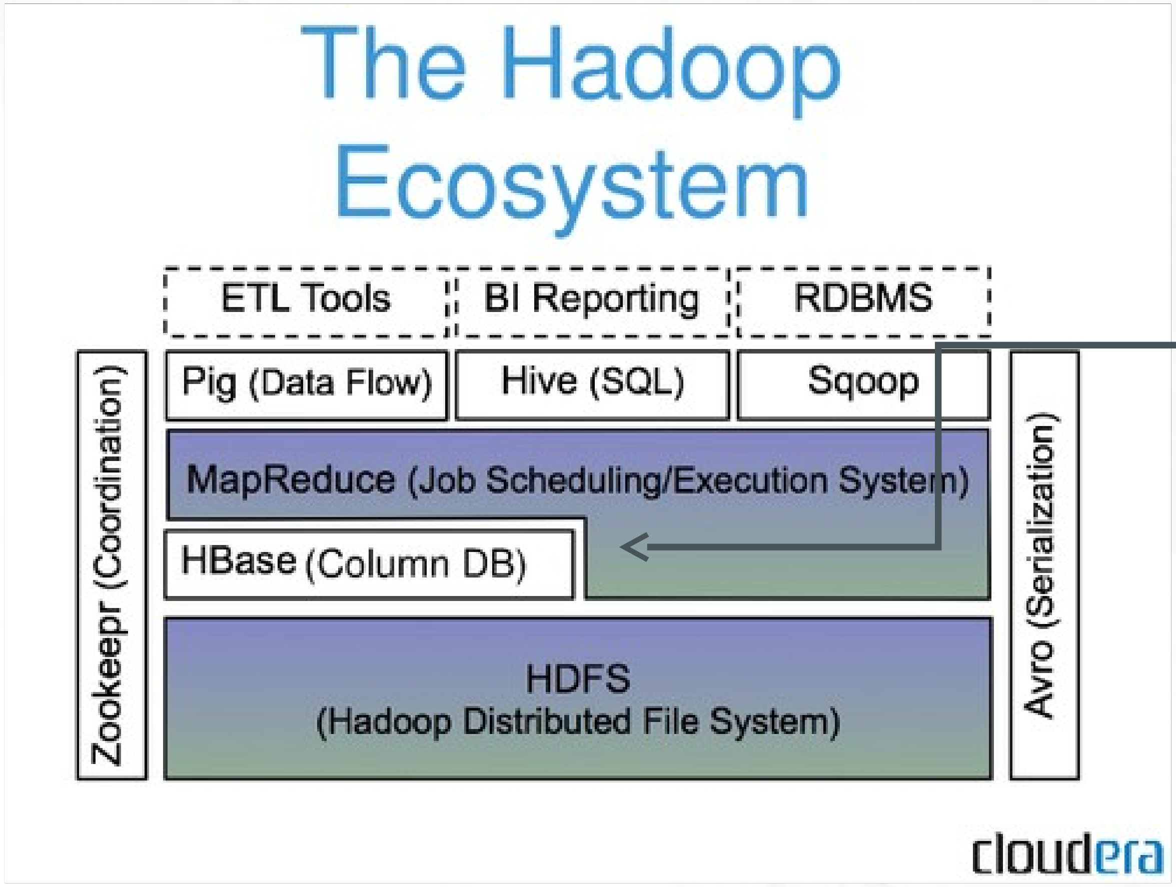
When to use HBase

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

HBase benefits than RDBMS

- *No real indexes*
- *Automatic partitioning*
- *Scale linearly and automatically with new nodes*
- *Commodity hardware*
- *Fault tolerance*
- *Batch processing*

HBase: Part of Hadoop's Ecosystem



HBase is built on top of HDFS

HBase files are internally stored in HDFS

HBase vs. HDFS

- 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

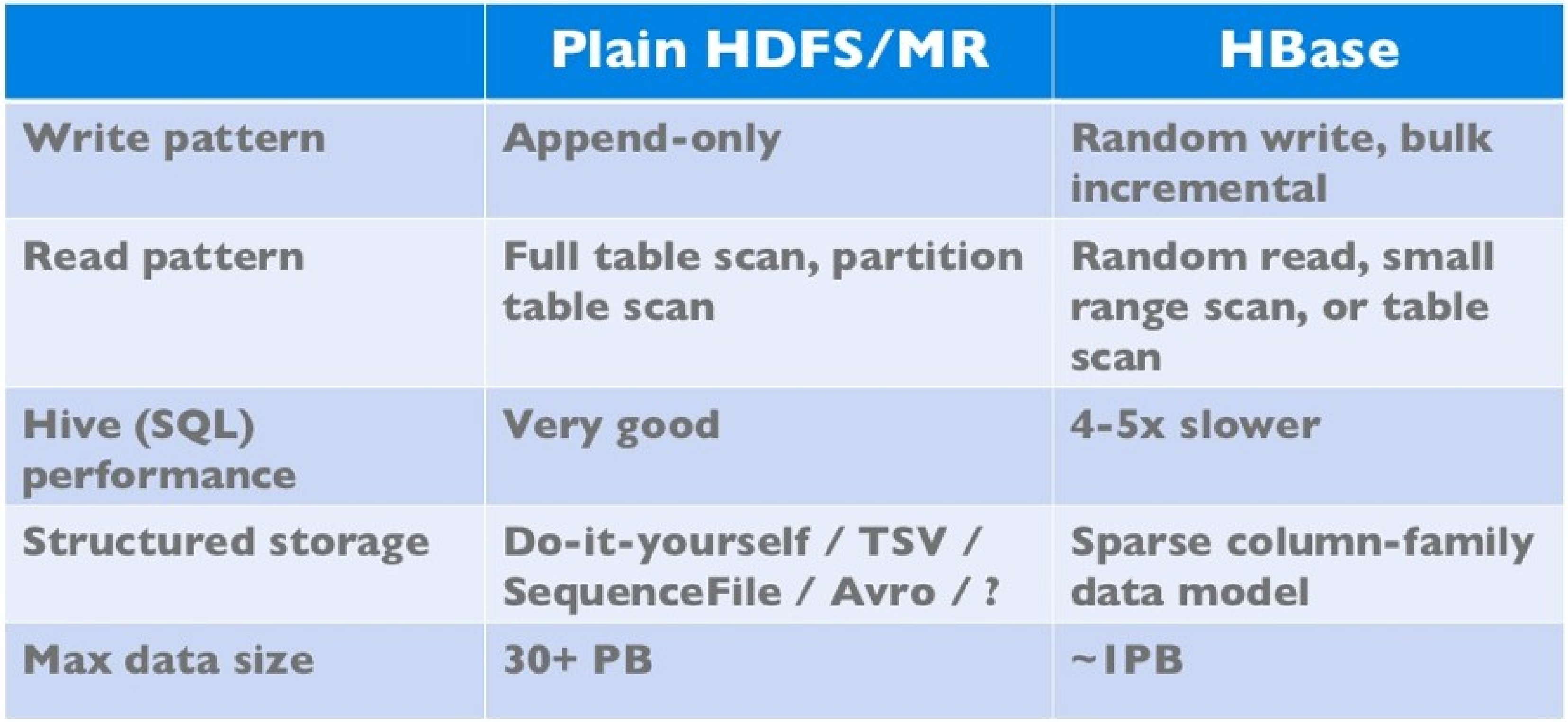
  Not good for updates

HBase vs. HDFS (Cont'd)

- ***HBase*** is designed to efficiently address the above points
  - Fast record lookup
  - Support for record-level insertion
  - Support for updates (not in place)

- HBase updates are done by creating new versions of values

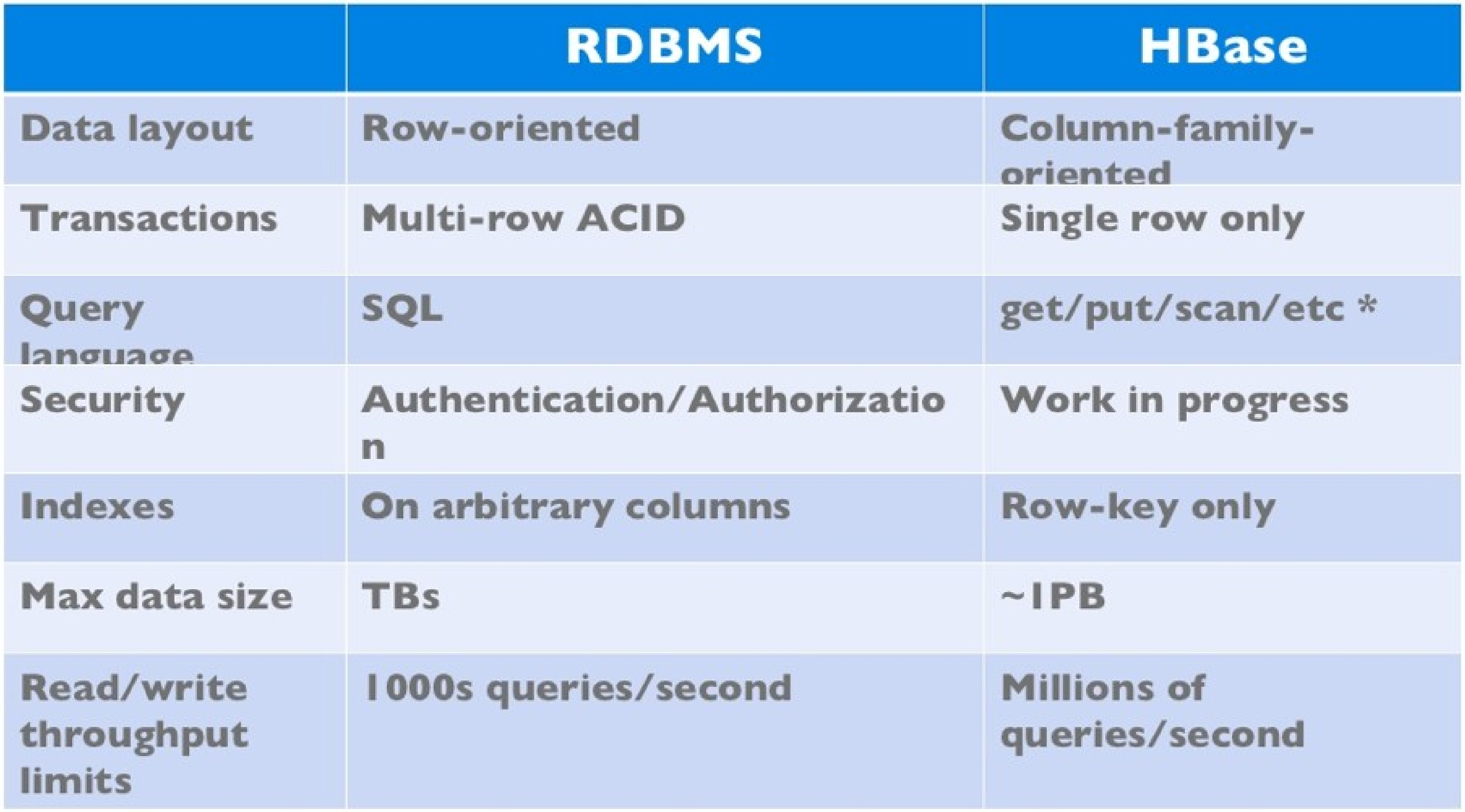
HBase vs. HDFS (Cont'd)



	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

If application has neither random reads or writes → Stick to HDFS

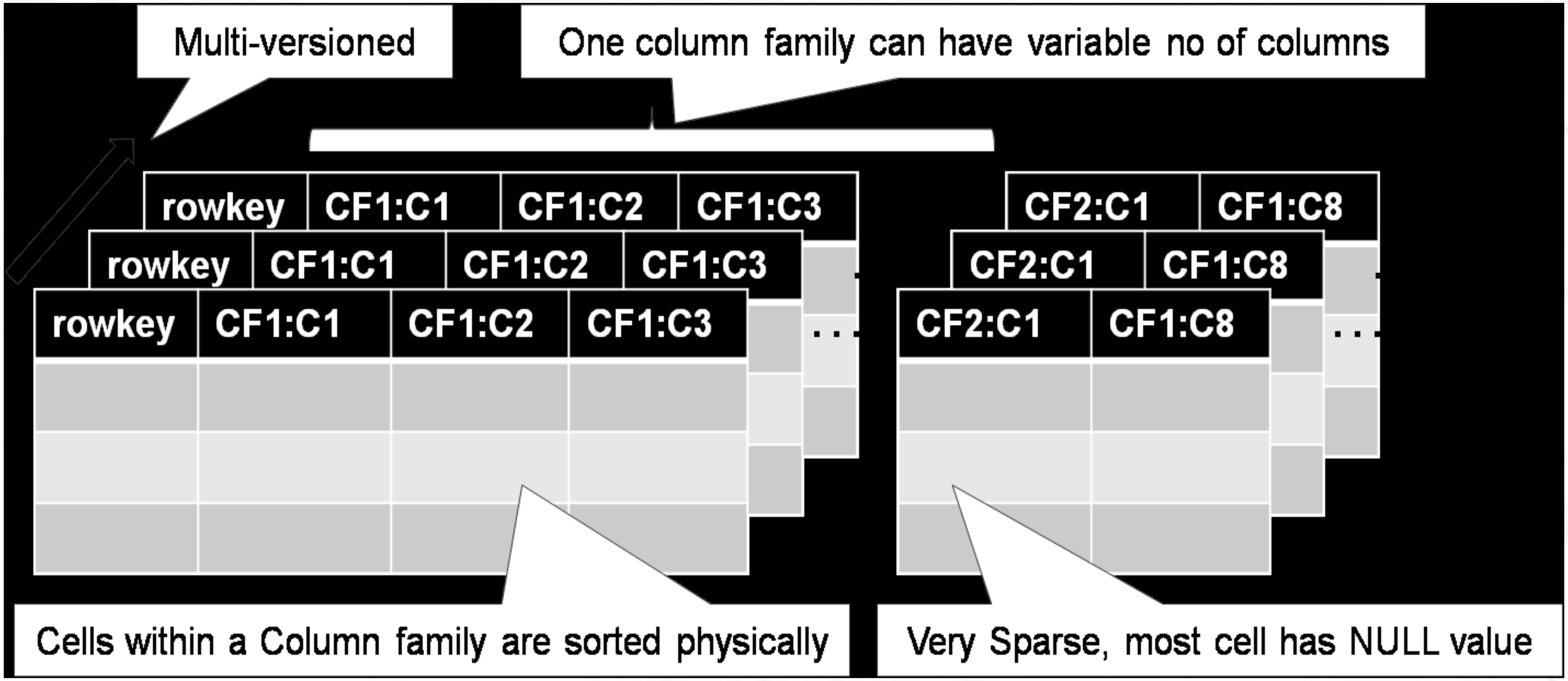
HBase vs. RDBMS



	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorizatio n	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of queries/second

HBase Data Model

- Data is divided into various *tables*
- Table is composed of *columns,* columns are grouped into *column-families*


Multi-versioned
One column family can have variable no of columns
rowkey
CF1:C1
CF1:C2
CF1:C3
CF2:C1
CF1:C8
. . .
Cells within a Column family are sorted physically
Very Sparse, most cell has NULL value

HBase Storage Model

- Partitioning

  - A table is horizontally partitioned into *regions,* each region is composed of sequential range of keys

  - Each region is managed by a *RegionServer,* a single RegionServer may hold multiple regions

- Persistence and data availability

  - HBase stores its data in HDFS, it doesn't replicate RegionServers and relies on HDFS replication for data availability.

  - Region data is cached in-memory

    * Updates and reads are served from in-memory cache (MemStore)

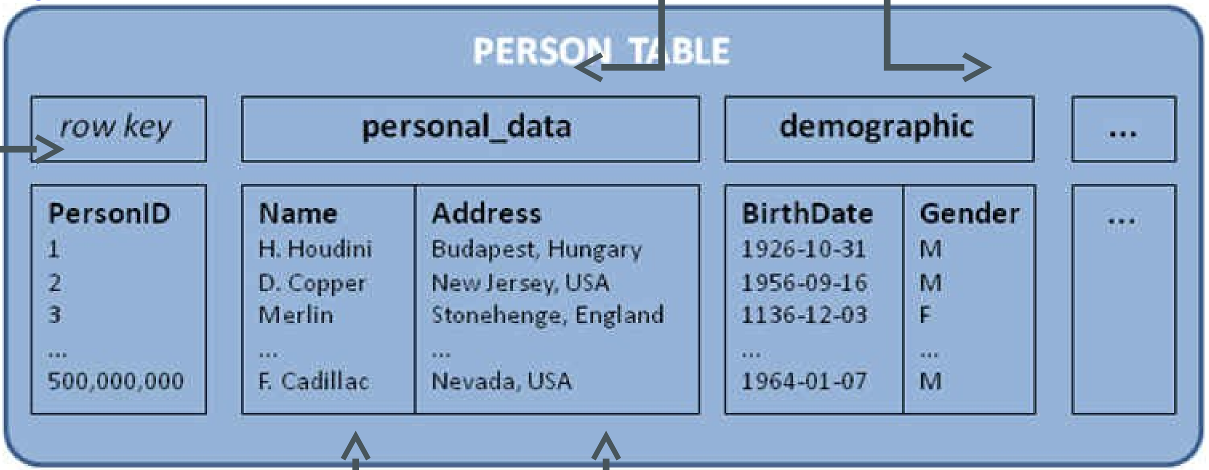
    * MemStore is flushed periodically to HDFS

    * Write Ahead Log (stored in HDFS) is used for durability of updates

HBase: Keys and Column Families

Each record is divided into *Column Families*

Each row has a *Key*



PERSON TABLE

row key	personal_data		demographic		...
PersonID	**Name**	**Address**	**BirthDate**	**Gender**	...
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	D. Copper	New Jersey, USA	1956-09-16	M	
3	Merlin	Stonehenge, England	1136-12-03	F	
...	...	...	...	...	
500,000,000	F. Cadillac	Nevada, USA	1964-01-07	M	

Figure 2 - Census Data in Column Families

Each column family consists of one or more *Columns*

- **Key**
  - Byte array
  - Serves as the primary key for the table
  - Indexed far fast lookup
- **Column Family**
  - Has a name (string)
  - Contains one or more related columns
- **Column**
  - Belongs to one column family
  - Included inside the row
    - ***familyName:columnName***

Column family named “Contents”

Column family named “anchor”

Column named “apache.com”

Row key	Time Stamp	Column “contents:”	Column “anchor:”	
“com.apache.www”	t12	“<html>…”		
	t11	“<html>…”		
	t10		“anchor:apache.com”	“APACHE”
“com.cnn.www”	t15		“anchor:cnnsi.com”	“CNN”
	t13		“anchor:my.look.ca”	“CNN.com”
	t6	“<html>…”		
	t5	“<html>…”		
	t3	“<html>…”		

Version number for each row

- **Version Number**
  - Unique within each key
  - By default→ System’s timestamp
  - Data type is Long
- **Value (Cell)**
  - Byte array

value

Row key	Time Stamp	Column “contents:”	Column “anchor:”	
“com.apache.www”	t12	“<html>…”		
	t11	“<html>…”		
	t10		“anchor:apache.com”	“APACHE”
“com.cnn.www”	t15		“anchor:cnnsi.com”	“CNN”
	t13		“anchor:my.look.ca”	“CNN.com”
	t6	“<html>…”		
	t5	“<html>…”		
	t3	“<html>…”		

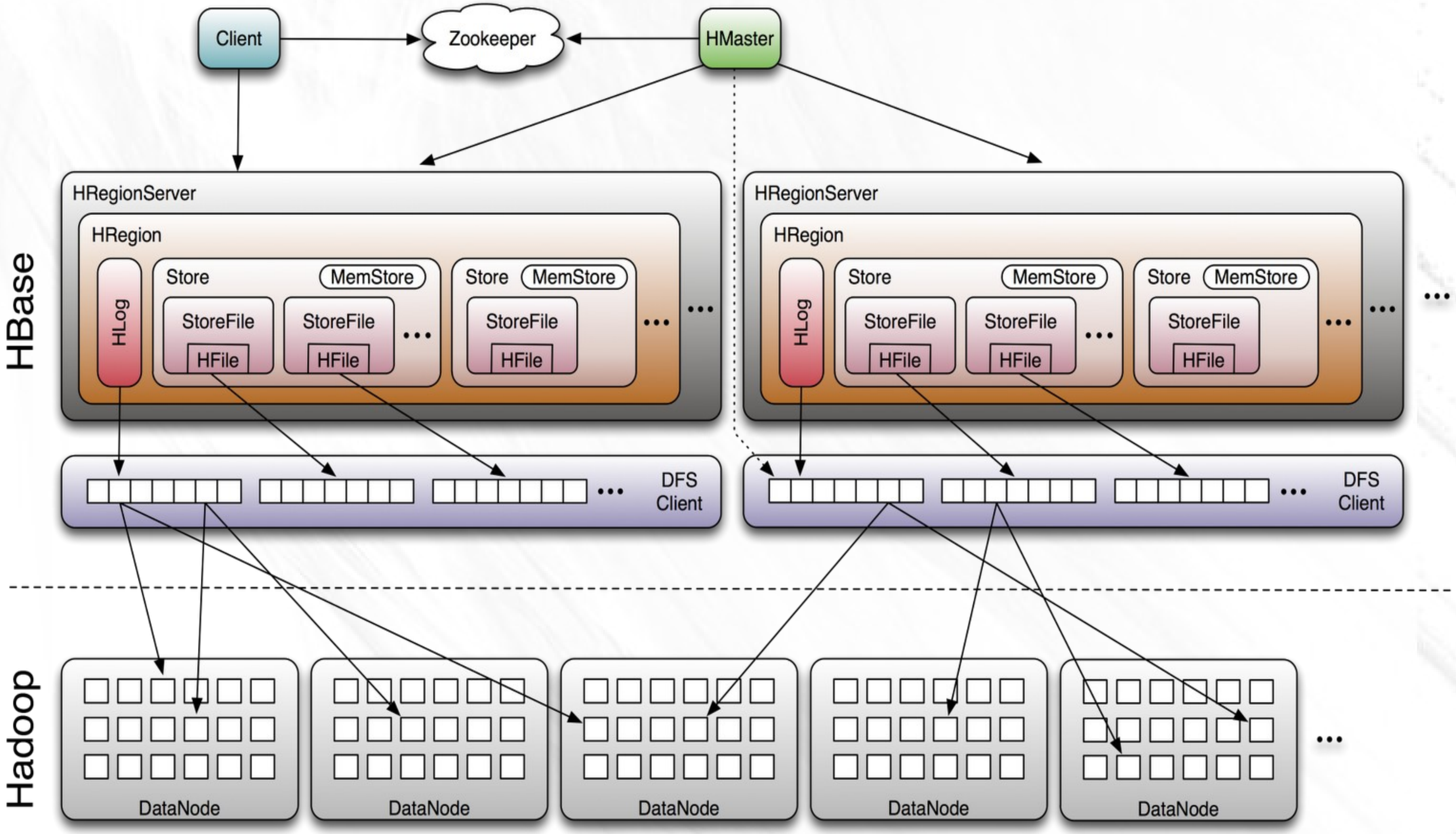
HBase Architecture
Three Major Components

- The HBaseMaster
  - One master
- The HRegionServer
  - Many region servers
- The HBase client

HBase Components

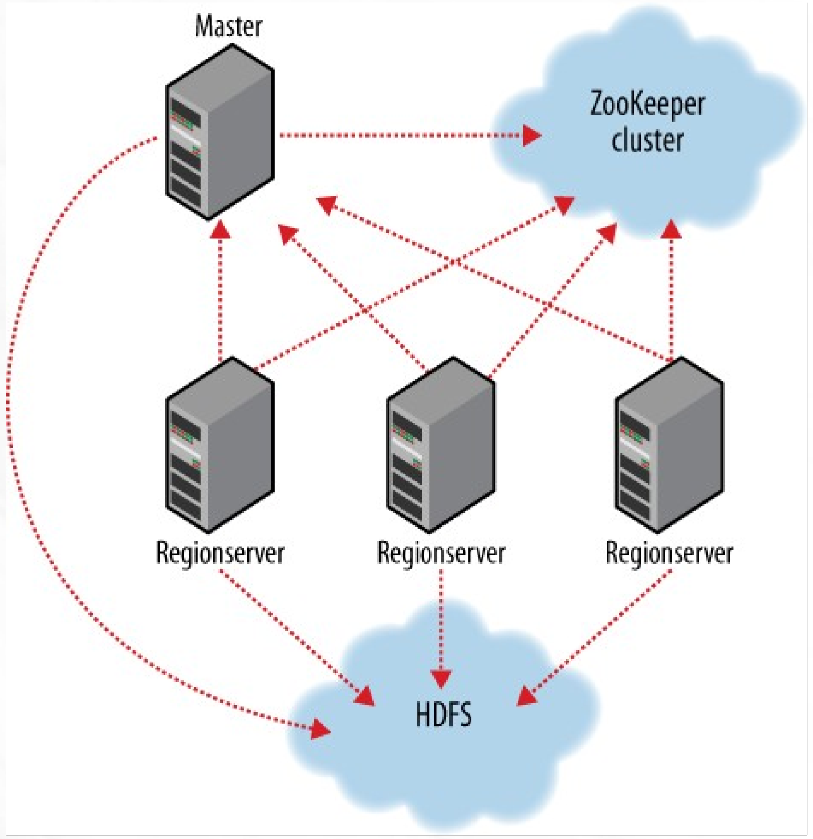
- **Region**
  - A subset of a table's rows, like horizontal range partitioning
  - Automatically done
- **RegionServer (many slaves)**
  - Manages data regions
  - Serves data for reads and writes (***using a log***)
- **Master**
  - Responsible for coordinating the slaves
  - Assigns regions, detects failures
  - Admin functions

Big Picture


Client
Zookeeper
HMaster
HBase
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
...
DFS Client
Hadoop
DataNode

ZooKeeper

- HBase depends on ZooKeeper
- By default HBase manages the ZooKeeper instance
  - E.g., starts and stops ZooKeeper
- HMaster and HRegionServers register themselves with ZooKeeper

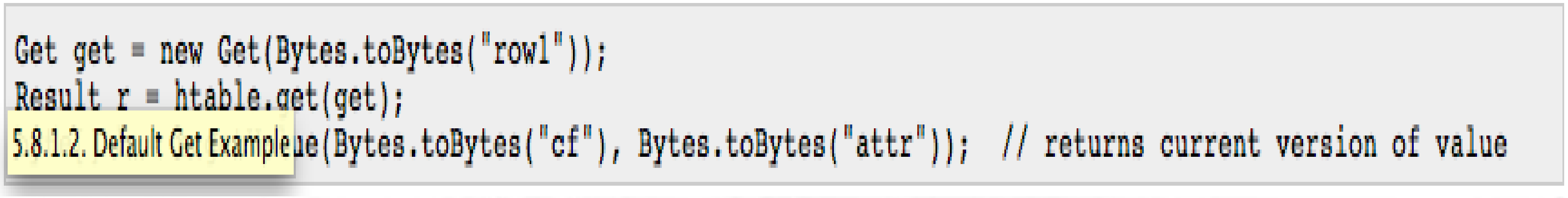


Creating a Table

```
HBaseAdmin admin= new HBaseAdmin(config);

HColumnDescriptor []column;

column= new HColumnDescriptor[2];

column[0]=new HColumnDescriptor("columnFamily1:");

column[1]=new HColumnDescriptor("columnFamily2:");

HTableDescriptor desc= new
HTableDescriptor(Bytes.toBytes("MyTable"));

desc.addFamily(column[0]);

desc.addFamily(column[1]);

admin.createTable(desc);
```

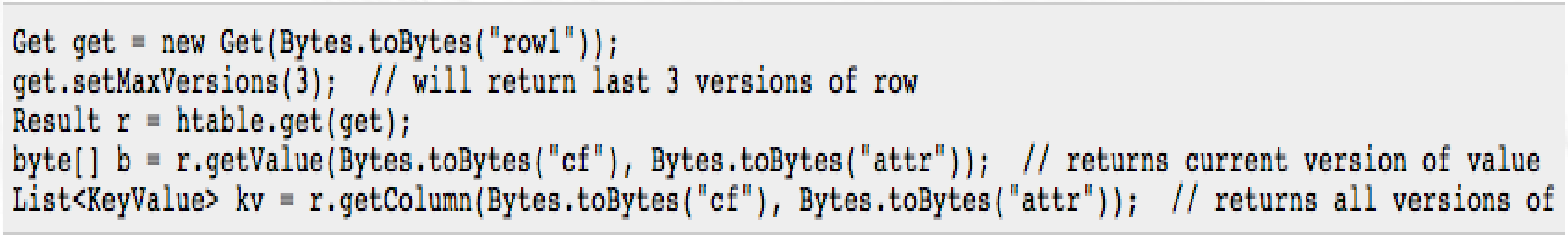
Operations On Regions: **Get()**

- Given a key → return corresponding record
- For each value return the highest version



```
Get get = new Get(Bytes.toBytes("row1"));
Result r = htable.get(get);
5.8.1.2. Default Get Exampleue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
```

- Can control the number of versions you want



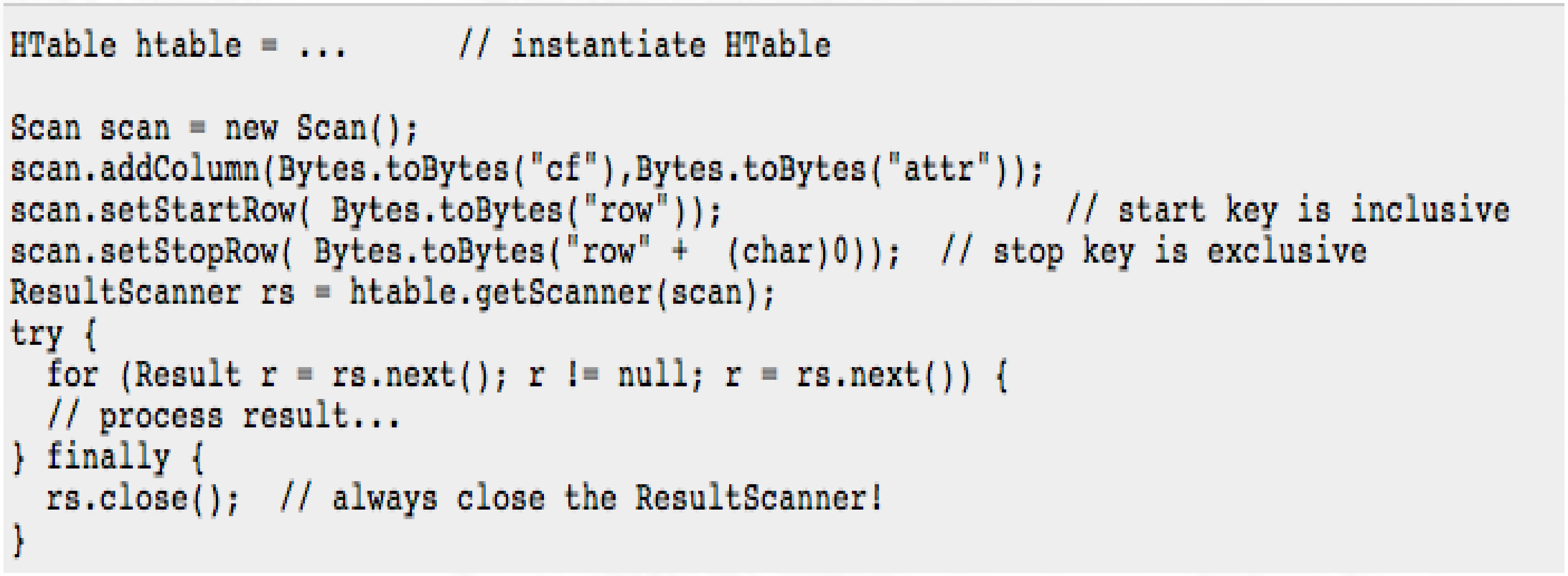
```
Get get = new Get(Bytes.toBytes("row1"));
get.setMaxVersions(3);  // will return last 3 versions of row
Result r = htable.get(get);
byte[] b = r.getValue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
List<KeyValue> kv = r.getColumn(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns all versions of
```

Get()

Select value from table where key=‘com.apache.www’ AND label=‘anchor:apache.com’

Row key	**Time Stamp**	**Column** “anchor:”	
	t12		
“com.apache.www”	t11		
	t10	“anchor:apache.com”	**“APACHE”**
	t9	“anchor:cnnsi.com”	“CNN”
	t8	“anchor:my.look.ca”	“CNN.com”
“com.cnn.www”	t6		
	t5		
	t3		

Operations On Regions: **Scan()**



```
HTable htable = ...      // instantiate HTable

Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("cf"),Bytes.toBytes("attr"));
scan.setStartRow( Bytes.toBytes("row"));                    // start key is inclusive
scan.setStopRow( Bytes.toBytes("row" +  (char)0));  // stop key is exclusive
ResultScanner rs = htable.getScanner(scan);
try {
  for (Result r = rs.next(); r != null; r = rs.next()) {
  // process result...
} finally {
  rs.close();  // always close the ResultScanner!
}
```

Scan()

Select value from table
where anchor='cnnsi.com'

Row key	**Time Stamp**	**Column** "anchor:"	
	t12		
"com.apache.www"	t11		
	t10	"anchor:apache.com"	"APACHE"
	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"
"com.cnn.www"	t6		
	t5		
	t3		

Operations On Regions: **Put()**

- Insert a new record (with a new key), Or
- Insert a record for an existing key

Implicit version number (timestamp)

```
Put put = new Put(Bytes.toBytes(row));
put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), Bytes.toBytes( data));
htable.put(put);
```

Explicit version number

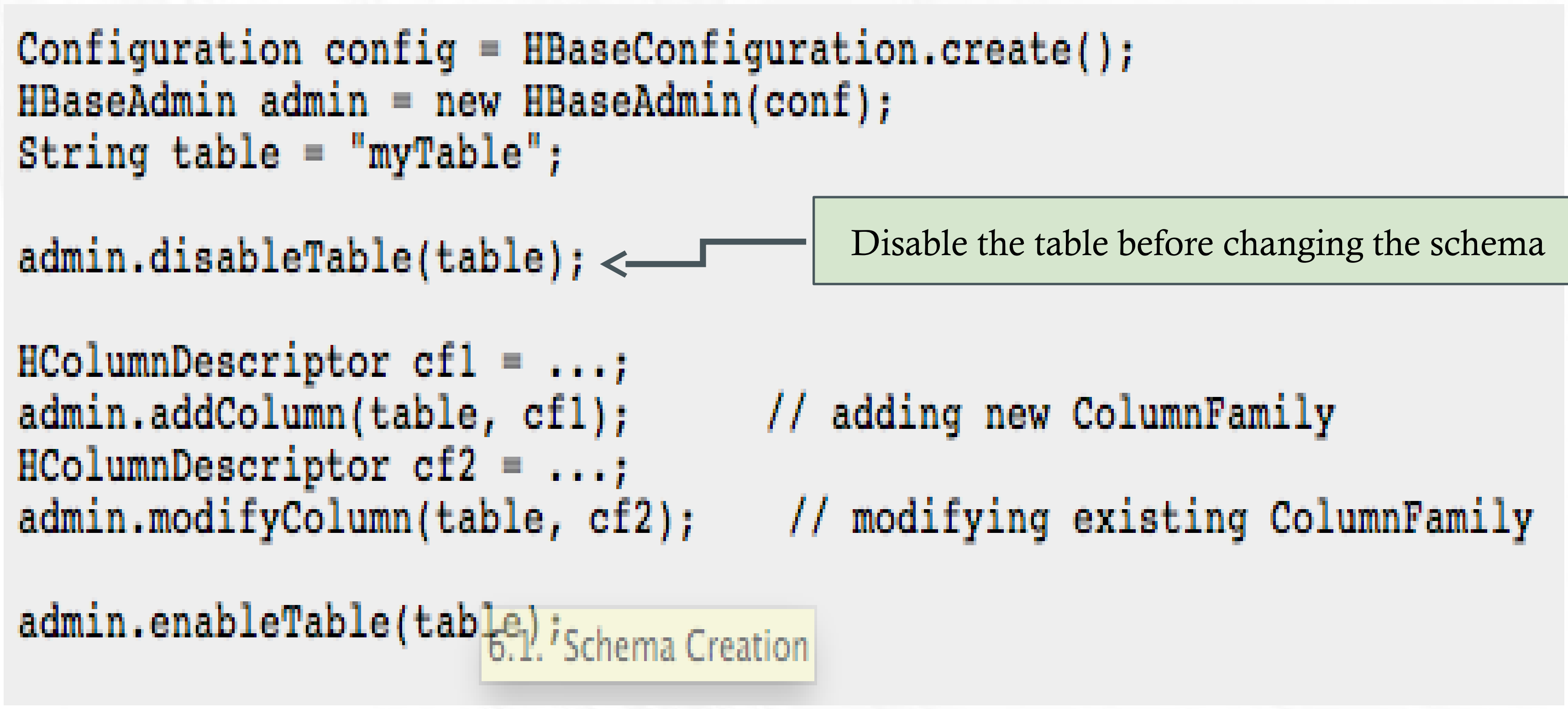
```
Put put = new Put( Bytes.toBytes(row));
long explicitTimeInMs = 555;  // just an example
put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), explicitTimeInMs, Bytes.toBytes(data));
htable.put(put);
```

Operations On Regions: **Delete()**

- Marking table cells as deleted
- **Multiple levels**
  - Can mark an entire column family as deleted
  - Can make all column families of a given row as deleted

- **All operations are logged by the RegionServers**
- **The log is flushed periodically**

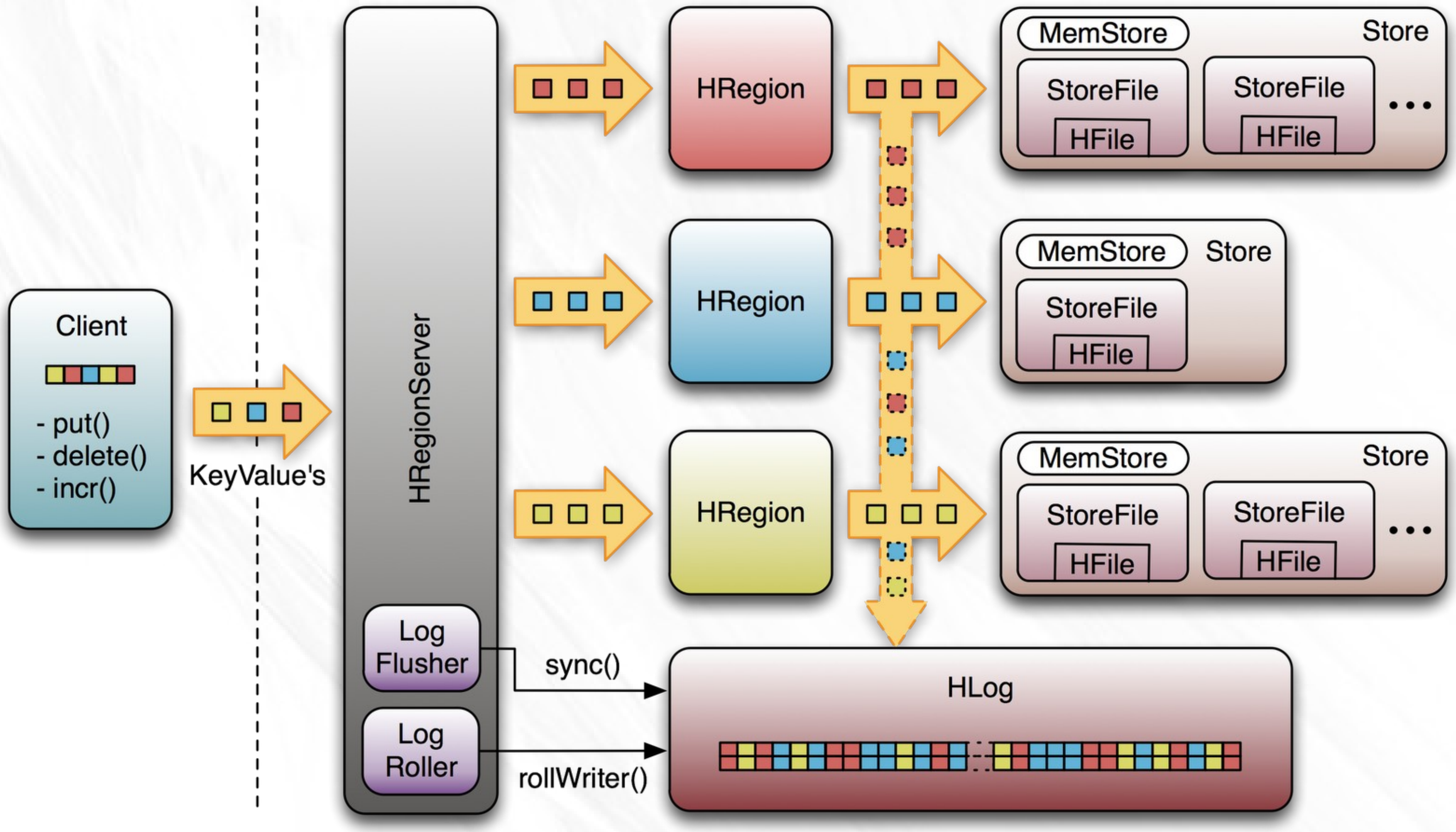
Altering a Table



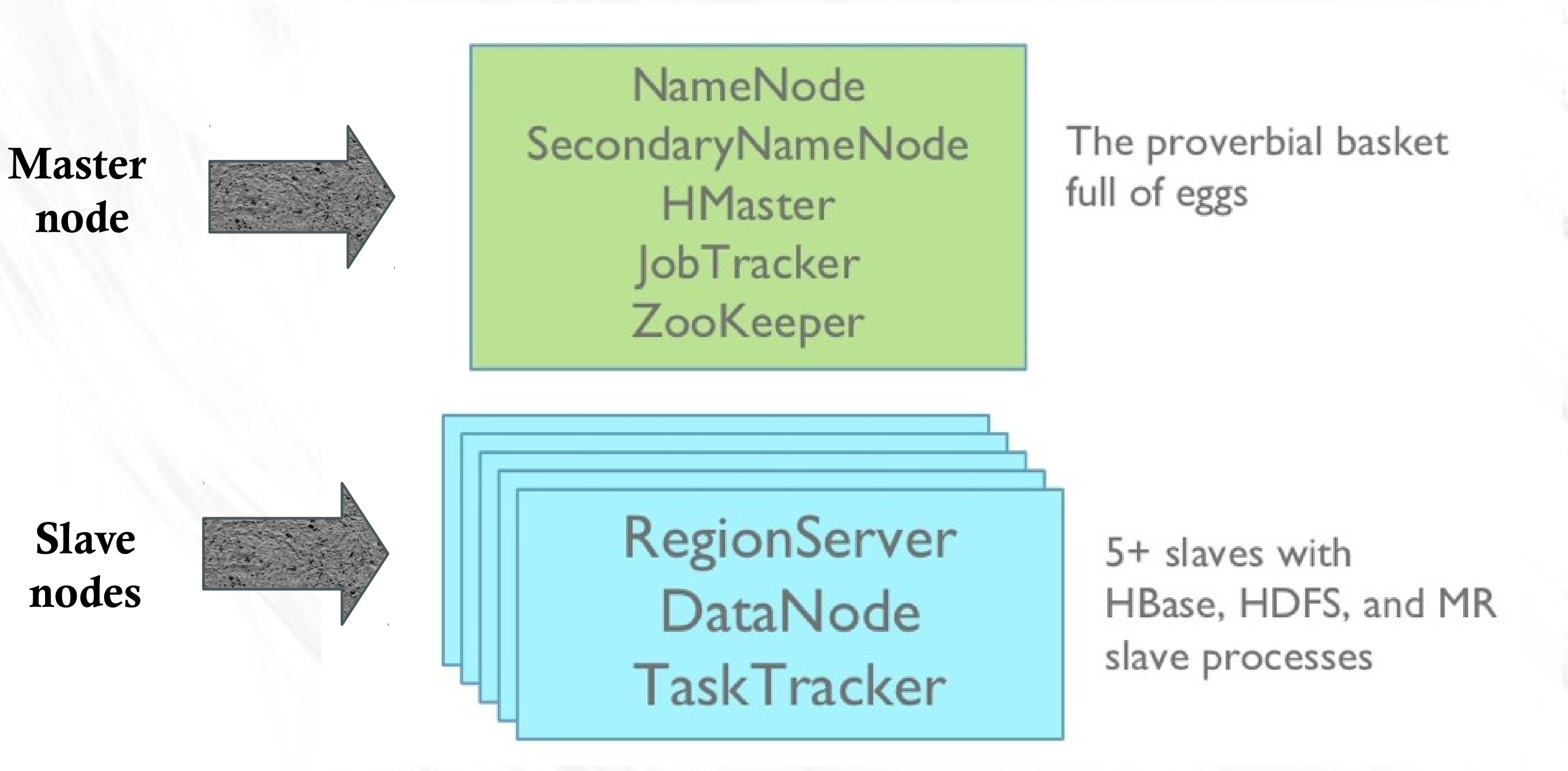
```
Configuration config = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
String table = "myTable";

admin.disableTable(table);

HColumnDescriptor cf1 = ...;
admin.addColumn(table, cf1);      // adding new ColumnFamily
HColumnDescriptor cf2 = ...;
admin.modifyColumn(table, cf2);    // modifying existing ColumnFamily

admin.enableTable(table);
```

Disable the table before changing the schema

Logging Operations



HBase Deployment



References

- Introduction to Hbase

*trac.nchc.org.tw/cloud/raw-attachment/wiki/.../**hbase**_intro.ppt*

- *web.cs.wpi.edu/~cs525/s13-MYE/lectures/5/HBase.pptx*
- *www-users.cselabs.umn.edu/classes/Spring.../Hadoop-HBase-Tutorial.ppt*
- *www.cs.kent.edu/~jin/Cloud12Spring/HbaseHivePig.pptx*

THANK
YOU