

<http://poloclub.gatech.edu/cse6242>

CSE6242 / CX4242: **Data** & **Visual** Analytics

Scaling Up

HBase

Duen Horng (Polo) Chau

Assistant Professor

Associate Director, MS Analytics

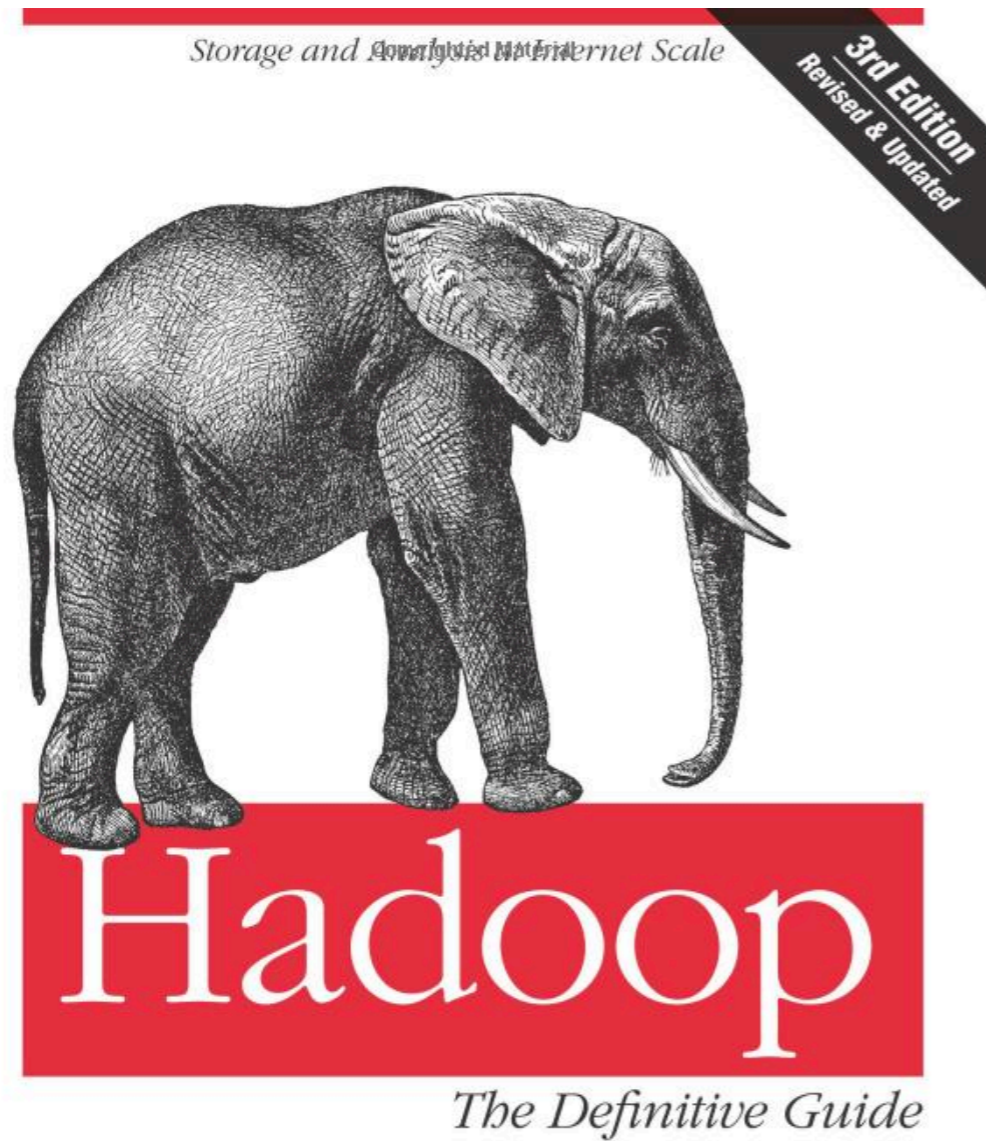
Georgia Tech

Partly based on materials by

Professors Guy Lebanon, Jeffrey Heer, John Stasko, Christos Faloutsos, Parishit Ram (GT PhD alum; SkyTree), Alex Gray

What if you need **real-time**
read/write for large datasets?

Lecture based on these two books.



O'REILLY®

Tom White

<http://goo.gl/YNCWN>



O'REILLY®

Lars George

Copyrighted Material

<http://goo.gl/svzTV>



<http://hbase.apache.org>

Built on top of HDFS

Supports **real-time** read/write random access

Scale to very large datasets, many machines

Not relational, does NOT support SQL

(“**NoSQL**” = “not only SQL”) <http://en.wikipedia.org/wiki/NoSQL>

Supports **billions of rows, millions of columns**

(e.g., serving Facebook’s Messaging Platform)

Written in Java; works with other APIs/languages

(REST, Thrift, Scala)

Where does HBase come from?

<http://radar.oreilly.com/2014/04/5-fun-facts-about-hbase-that-you-didnt-know.html>
<http://wiki.apache.org/hadoop/Hbase/PoweredBy>

HBase's "history"

Designed for batch processing



Hadoop & HDFS based on...

- 2003 *Google File System (GFS)* paper

<http://cracking8hacking.com/cracking-hacking/Ebooks/Misc/pdf/The%20Google%20filesystem.pdf>

- 2004 *Google MapReduce* paper

<http://static.googleusercontent.com/media/research.google.com/en/us/archive/mapreduce-osdi04.pdf>

HBase based on ...

- 
- 2006 *Google Bigtable* paper

<http://static.googleusercontent.com/media/research.google.com/en/us/archive/bigtable-osdi06.pdf>

Designed for random access

How does HBase work?

Column-oriented

Column is the most basic unit (instead of row)

- Multiple columns form a row
- A column can have **multiple versions**, each version stored in a **cell**

Rows form a table

- **Row key** locates a row
- Rows **sorted** by row key **lexicographically** (~= alphabetically)

Row key is unique

Think of row key as the “**index**” of an HBase table

- **You look up a row using its row key**

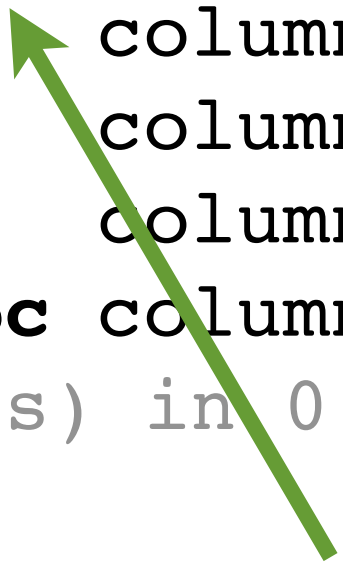
Only one “index” per table (via row key)

HBase does not have **built-in** support for multiple indices; support enabled via **extensions**

Rows sorted lexicographically (=alphabetically)

```
hbase(main):001:0> scan 'table1'
```

```
ROW          COLUMN+CELL
row-1       column=cf1:, timestamp=1297073325971 ...
row-10      column=cf1:, timestamp=1297073337383 ...
row-11      column=cf1:, timestamp=1297073340493 ...
row-2       column=cf1:, timestamp=1297073329851 ...
row-22      column=cf1:, timestamp=1297073344482 ...
row-3       column=cf1:, timestamp=1297073333504 ...
row-abc     column=cf1:, timestamp=1297073349875 ...
7 row(s) in 0.1100 seconds
```

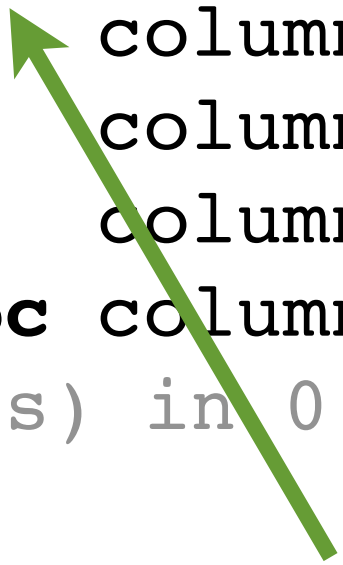


“row-10” comes before “row-2”.
How to fix?

Rows sorted lexicographically (=alphabetically)

```
hbase(main):001:0> scan 'table1'
```

```
ROW          COLUMN+CELL
row-1        column=cf1:, timestamp=1297073325971 ...
row-10       column=cf1:, timestamp=1297073337383 ...
row-11       column=cf1:, timestamp=1297073340493 ...
row-2        column=cf1:, timestamp=1297073329851 ...
row-22       column=cf1:, timestamp=1297073344482 ...
row-3        column=cf1:, timestamp=1297073333504 ...
row-abc      column=cf1:, timestamp=1297073349875 ...
7 row(s) in 0.1100 seconds
```



“row-10” comes before “row-2”.

How to fix?

Pad “row-2” with a “0”.

i.e., “row-02”

Columns grouped into **column families**

- Why?
 - Helps with organization, understanding, optimization, etc.
- **In details...**
 - Columns in the same family stored in same *file* called *HFile*
 - inspired by Google's **SSTable** = large map whose keys are sorted
 - Apply compression on the whole family
 - ...

More on **column family**, **column**

Column family

- An HBase table supports only **few** families (e.g., <10)
 - Due to limitations in implementation
- Family name must be **printable**
- Should be defined when table is created
 - Shouldn't be changed often

Each **column** referenced as “**family:qualifier**”

- Can have **millions** of columns
- Values can be **anything** that's **arbitrarily long**

Cell Value

Timestamped

- Implicitly by system
- Or set explicitly by user

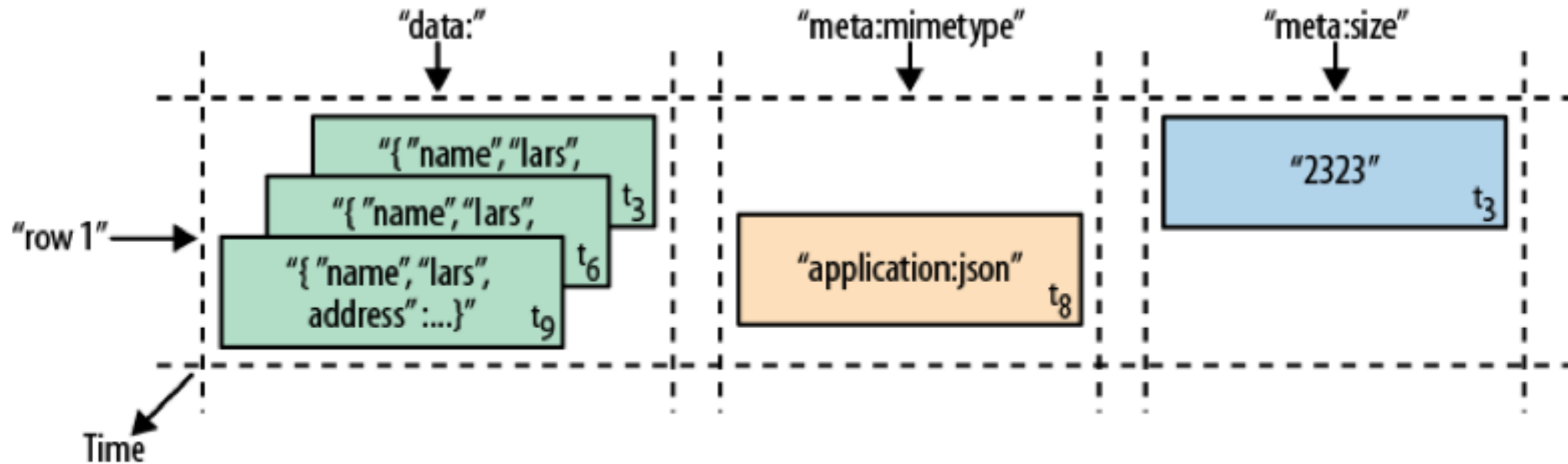
Let you store **multiple versions** of a value

- = values over time

Values stored in **decreasing** time order

- **Most recent value** can be read first

Time-oriented view of a row



Row Key	Time Stamp	Column "meta:"	
		"mimetype"	"size"
"row1"	t_3	<code>{"name": "lars", "address": ...}</code>	<code>"2323"</code>
	t_6	<code>{"name": "lars", "address": ...}</code>	
	t_8		<code>"application/json"</code>
	t_9	<code>{"name": "lars", "address": ...}</code>	

Concise way to describe all these?

HBase data model (= Bigtable's model)

- Sparse, distributed, persistent, **multidimensional map**
- Indexed by **row** key + **column** key + **timestamp**

(Table, RowKey, Family, Column, Timestamp) → Value

An exercise

How would you use HBase to create a *wehtable* store **snapshots** of every **webpage** on the planet, **over time**?

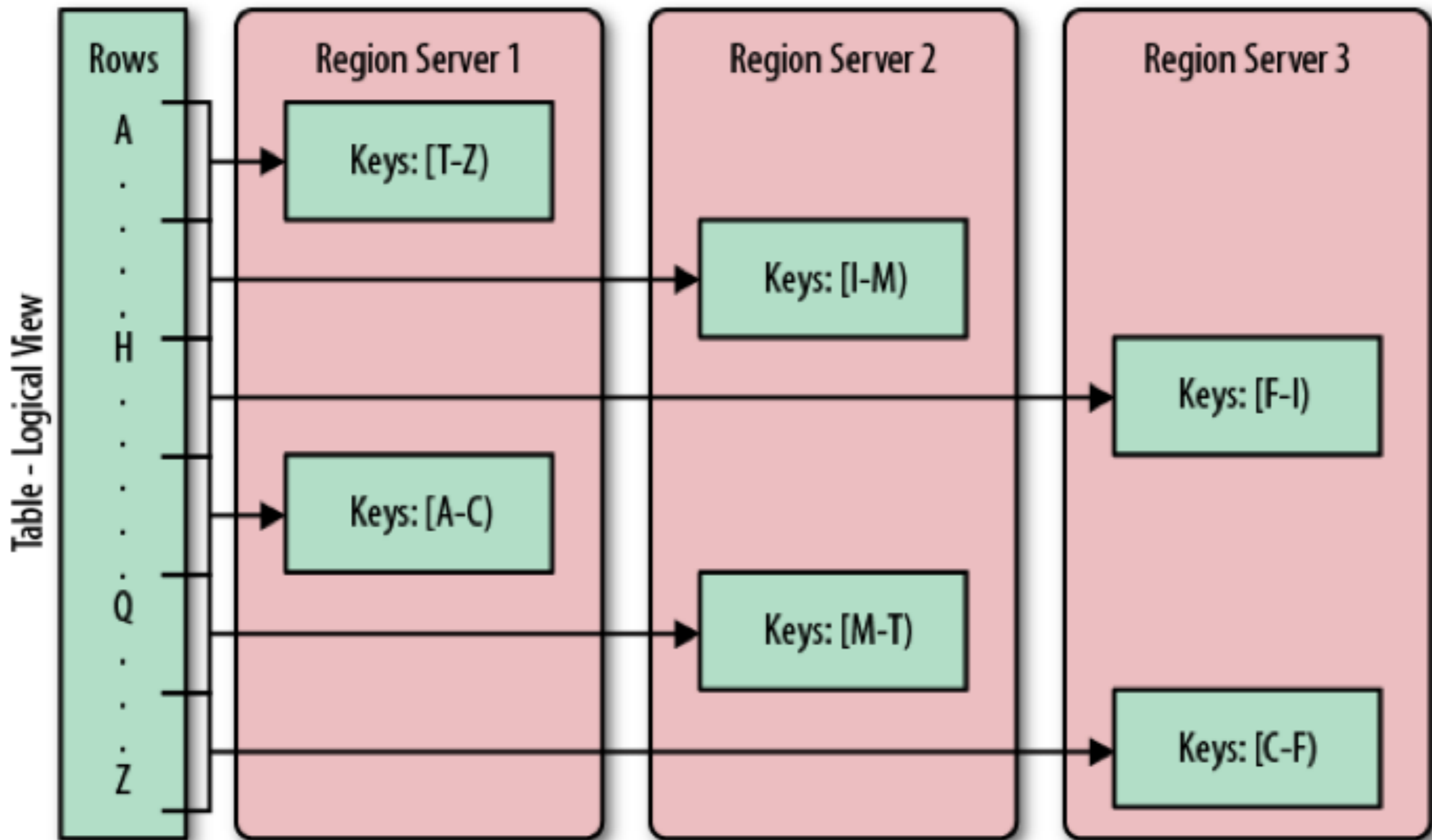
Details: How does HBase **scale up storage & balance load?**

Automatically divide contiguous ranges of rows into regions

Start with one region, split into two when getting too large

Details: How does HBase scale up storage & balance load?

Region Servers - Physical Layout



How to use HBase

Interactive shell

- Will show you an example, locally (on your computer, without using HDFS)

Programmatically

- e.g., via Java, Python, etc.

Example, using interactive shell

```
$ cd /usr/local/hbase-0.91.0-SNAPSHOT
$ bin/start-hbase.sh
starting master, logging to \
/usr/local/hbase-0.91.0-SNAPSHOT/bin/../logs/hbase-<username>-master-localhost.out
$ bin/hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.91.0-SNAPSHOT, r1130916, Sat Jul 23 12:44:34 CEST 2011

hbase(main):001:0> status
1 servers, 0 dead, 2.0000 average load
```

Start HBase

Start Interactive Shell

Check HBase is running

Example: Create table, add values

```
hbase(main):002:0> create 'testtable', 'colfam1'  
0 row(s) in 0.2930 seconds
```

```
hbase(main):003:0> list 'testtable'  
TABLE  
testtable  
1 row(s) in 0.0520 seconds
```

```
hbase(main):004:0> put 'testtable', 'myrow-1', 'colfam1:q1', 'value-1'  
0 row(s) in 0.1020 seconds
```

```
hbase(main):005:0> put 'testtable', 'myrow-2', 'colfam1:q2', 'value-2'  
0 row(s) in 0.0410 seconds
```

```
hbase(main):006:0> put 'testtable', 'myrow-2', 'colfam1:q3', 'value-3'  
0 row(s) in 0.0380 seconds
```

Example: Scan (show all cell values)

```
hbase(main):007:0> scan 'testtable'
```

```
ROW          COLUMN+CELL
myrow-1      column=colfam1:q1, timestamp=1297345476469, value=value-1
myrow-2      column=colfam1:q2, timestamp=1297345495663, value=value-2
myrow-2      column=colfam1:q3, timestamp=1297345508999, value=value-3
```

```
2 row(s) in 0.1100 seconds
```

Example: Get (look up a row)

```
hbase(main):008:0> get 'testtable', 'myrow-1'
```

```
COLUMN
```

```
CELL
```

```
colfam1:q1
```

```
timestamp=1297345476469, value=value-1
```

```
1 row(s) in 0.0480 seconds
```

Can also look up a particular cell value, with a certain timestamp, etc.

Example: Delete a value

```
hbase(main):009:0> delete 'testtable', 'myrow-2', 'colfam1:q2'  
0 row(s) in 0.0390 seconds
```

```
hbase(main):010:0> scan 'testtable'
```

ROW	COLUMN+CELL
myrow-1	column=colfam1:q1, timestamp=1297345476469, value=value-1
myrow-2	column=colfam1:q3, timestamp=1297345508999, value=value-3

```
2 row(s) in 0.0620 seconds
```

Example: Disable & drop table

```
hbase(main):011:0> disable 'testtable'  
0 row(s) in 2.1250 seconds
```

```
hbase(main):012:0> drop 'testtable'  
0 row(s) in 1.2780 seconds
```


RDBMS vs HBase

RDBMS (=Relational Database Management System)

- MySQL, Oracle, SQLite, Teradata, etc.
- Really great for many applications
 - Ensure strong data consistency, integrity
 - Supports transactions (ACID guarantees)
 - ...

RDBMS vs HBase

How are they different? When to use what?

RDBMS vs HBase

How are they different?

- Hbase when you don't know the structure/schema
- HBase supports sparse data (many columns, most values are not there)
- Use RDBMS if you only work with a small number of columns
- Relational databases good for getting “whole” rows
- HBase: Multiple versions of data
- RDBMS support multiple indices, minimize duplications
- Generally a lot cheaper to deploy HBase, for same size of data (petabytes)

More topics to learn about

Other ways to get, put, delete... (e.g., **programmatically** via Java)

- Doing them in **batch**

Maintaining your cluster

- **Configurations, specs** for “master” and “slaves”?
- Administrating cluster
- Monitoring cluster’s health

Key design (<http://hbase.apache.org/book/rowkey.design.html>)

- **bad keys** can decrease performance

Integrating with MapReduce

Cassandra, MongoDB, etc.

<http://db-engines.com/en/system/Cassandra%3BHBase%3BMongoDB>

<http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis>