CS595—BIG DATA TECHNOLOGIES

Module 12

NoSQL Key/Value + Wide Column Store

Apache HBase (Hadoop)

HBase

- Apache Hadoop has gained popularity for storing, managing and processing high volume and high velocity data
- However, the Hadoop filesystem (HDFS) cannot handle random writes and reads and cannot change a file without rewriting it
- HBase is a NoSQL, column oriented database built on top of Hadoop to overcome the drawbacks of HDFS as it allows fast random writes and reads in an optimized way
- Also, with exponentially growing data, relational databases cannot handle the variety of data to render better performance
- HBase provides scalability and partitioning for efficient storage and retrieval

HBase

- HBase provides real-time read or write access to data in HDFS
- HBase can be referred to as a data store instead of a database…
- As it misses out on some important features of traditional RDBMs…
- Like typed columns, triggers, an advanced (SQL) query language and secondary indexes

HBase Features

- Strongly consistent reads/writes
  - HBase is not an "eventually consistent" data store
  - This makes it very suitable for tasks such as high-speed counter aggregation
- Automatic sharding
  - HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- Automatic failover
- Hadoop/HDFS Integration
  - HBase supports HDFS out of the box as its distributed file system.
- MapReduce
  - HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink

HBase Features

- Java Client API: HBase supports an easy to use Java API for programmatic access.
- Thrift/REST API: HBase also supports Thrift and REST for non-Java front-ends.

When to Consider HBase

- HBase isn’t suitable for every problem.
- First, make sure you have enough data
  - If you have hundreds of millions or billions of rows, then HBase is a good candidate
  - If you only have a few thousand/million rows, then using a traditional RDBMS might be a better choice…
  - Due to the fact that all of your data might wind up on a single node (or two) and the rest of the cluster may be sitting idle
- Make sure you can live without all the extra features that an RDBMS provides
  - Such as typed columns, secondary indexes, transactions, advanced query languages, etc.

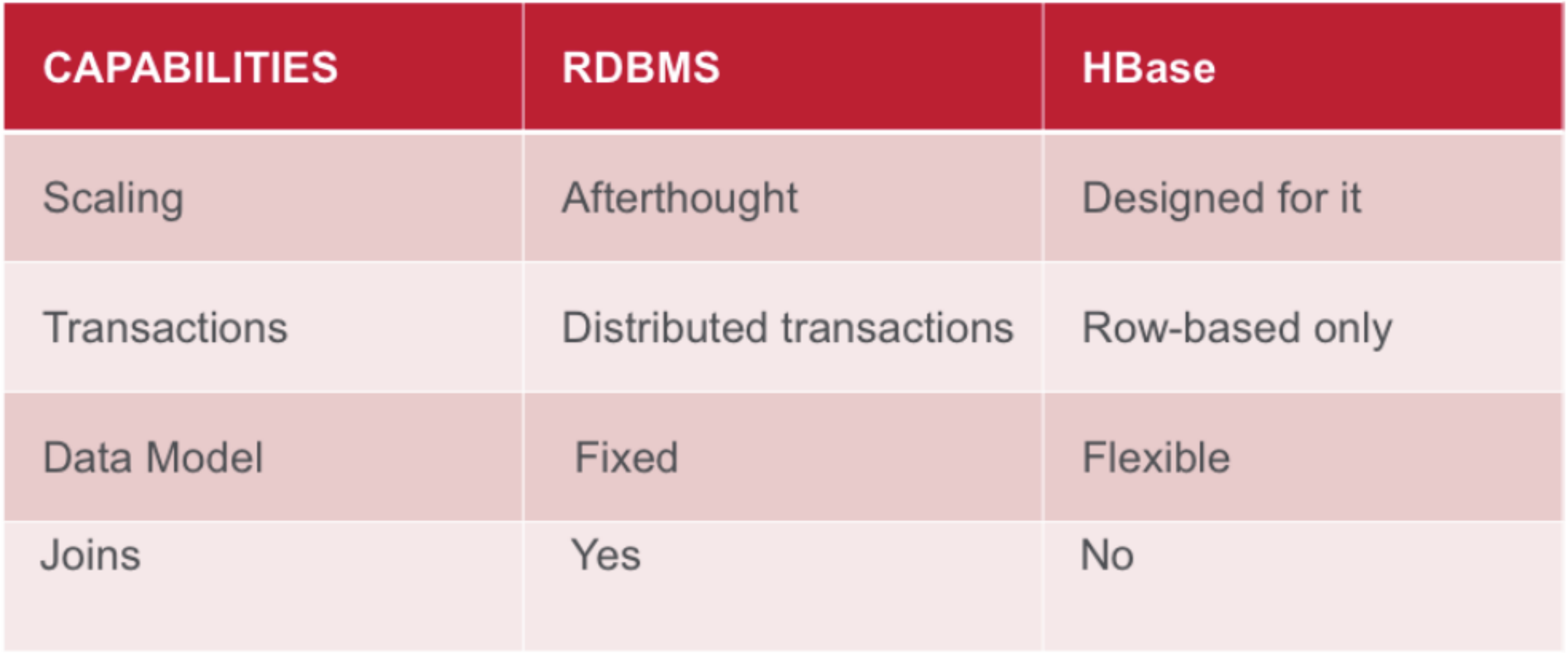
When to Consider HBase

- An application built against an RDBMS cannot be "ported" to HBase by simply changing a JDBC driver, for example
- Consider moving from an RDBMS to using HBase as a complete redesign as opposed to a port

HDFS vs. HBase

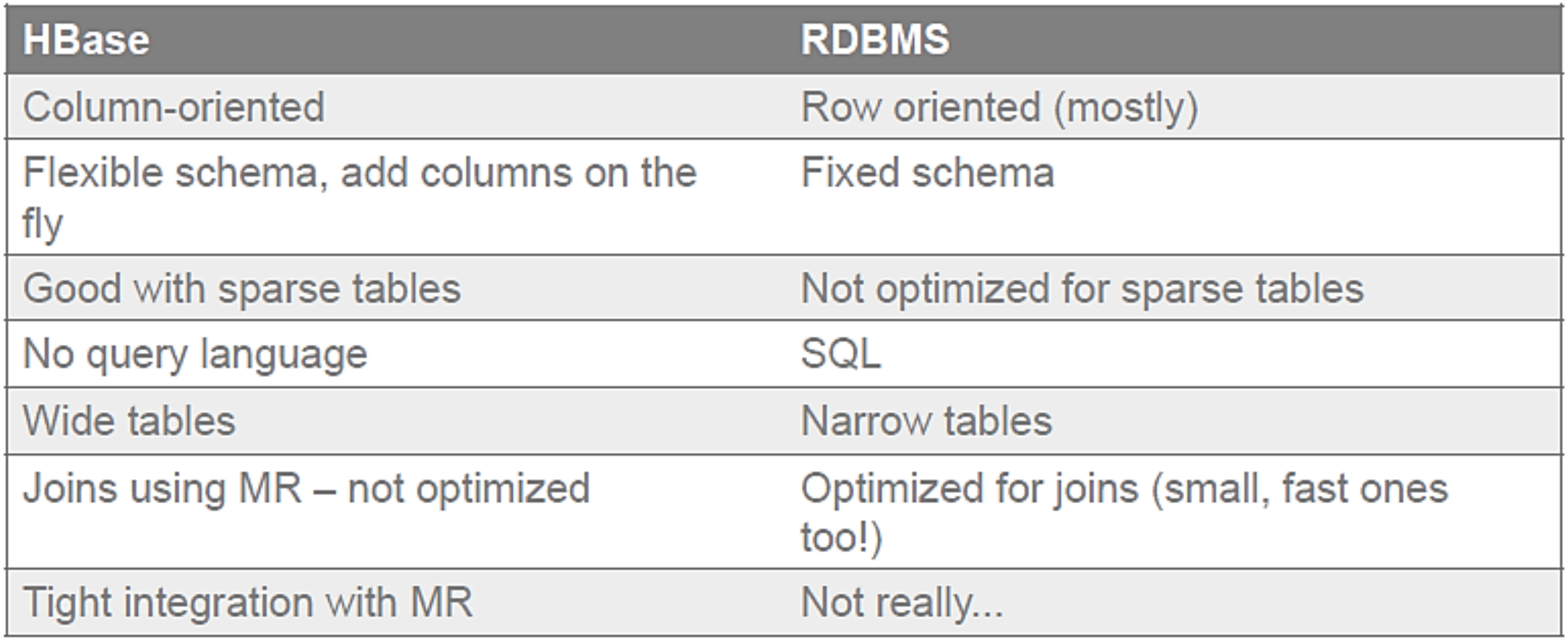
HDFS	HBase
Distributed file system	Built on top of HDFS
No fast data lookups	Fast data lookups via indexed files
Latency : high	Latency : low
Only sequential access	Random access via hash tables

HBase vs. RDBMS Summary



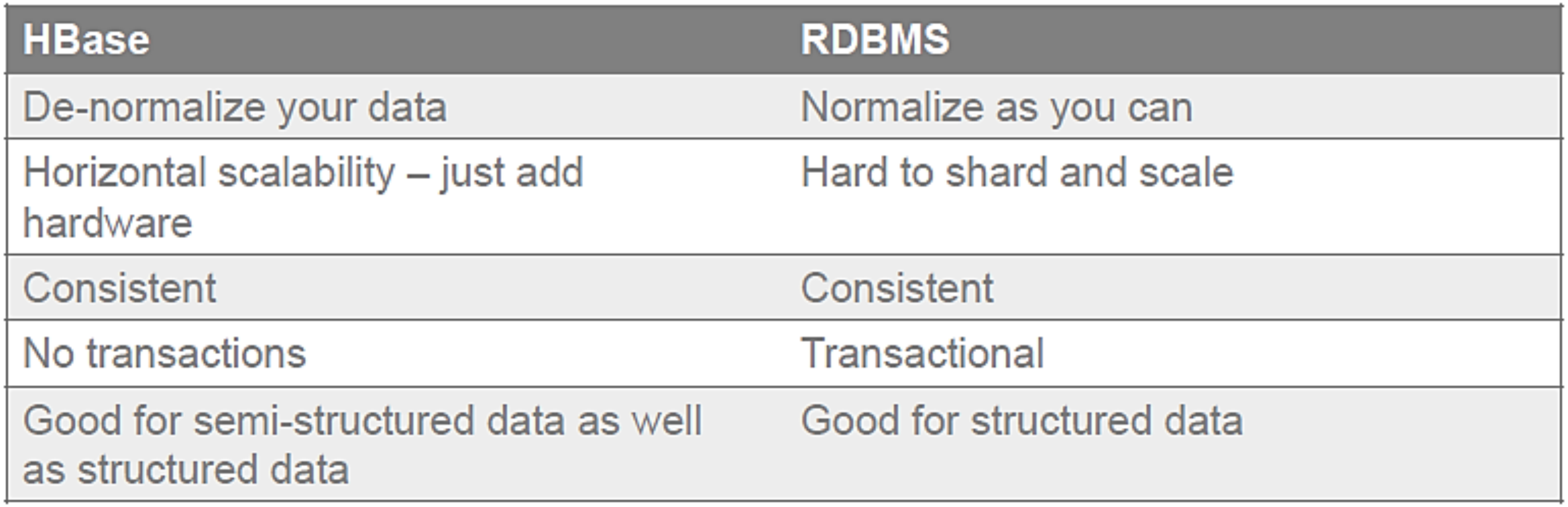
CAPABILITIES	RDBMS	HBase
Scaling	Afterthought	Designed for it
Transactions	Distributed transactions	Row-based only
Data Model	Fixed	Flexible
Joins	Yes	No

HBase vs. RDBMS Details



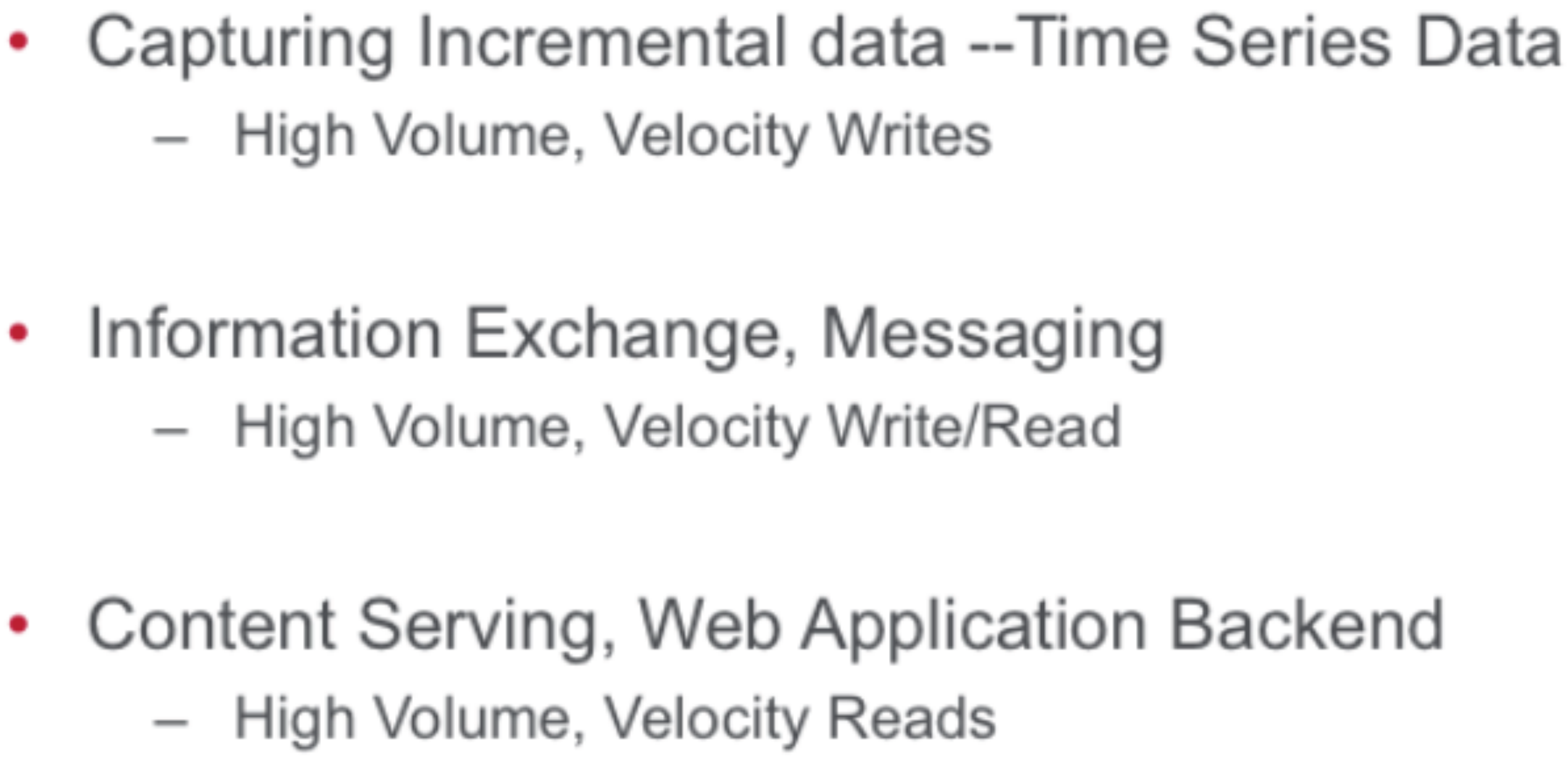
HBase	RDBMS
Column-oriented	Row oriented (mostly)
Flexible schema, add columns on the fly	Fixed schema
Good with sparse tables	Not optimized for sparse tables
No query language	SQL
Wide tables	Narrow tables
Joins using MR – not optimized	Optimized for joins (small, fast ones too!)
Tight integration with MR	Not really...

HBase vs. RDBMS Details



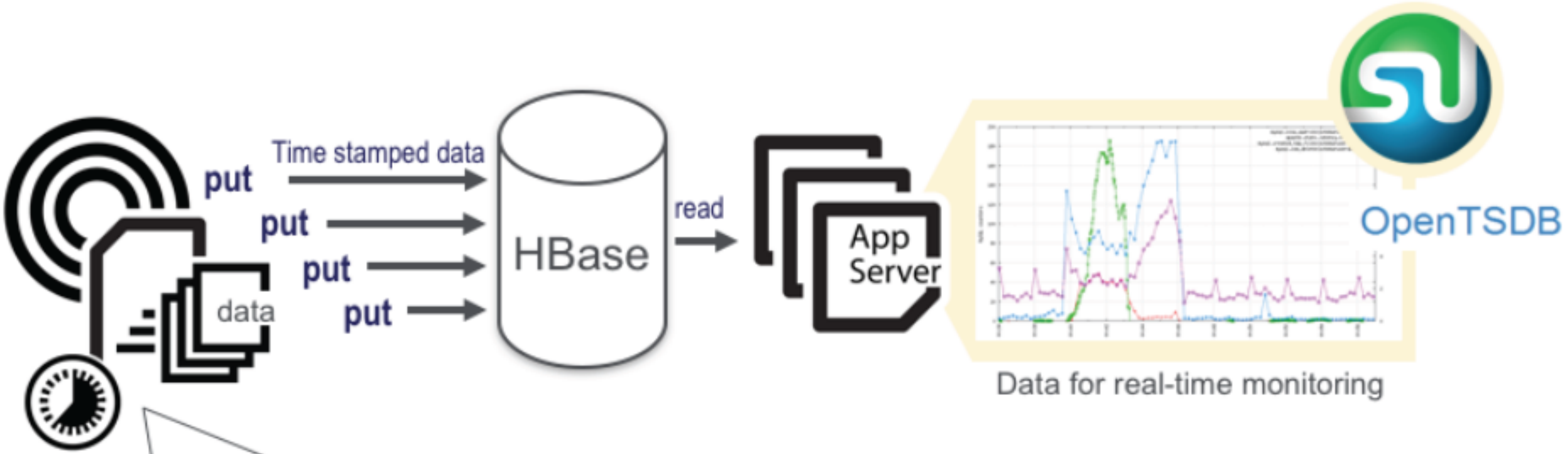
HBase	RDBMS
De-normalize your data	Normalize as you can
Horizontal scalability – just add hardware	Hard to shard and scale
Consistent	Consistent
No transactions	Transactional
Good for semi-structured data as well as structured data	Good for structured data

Main Use Case Categories



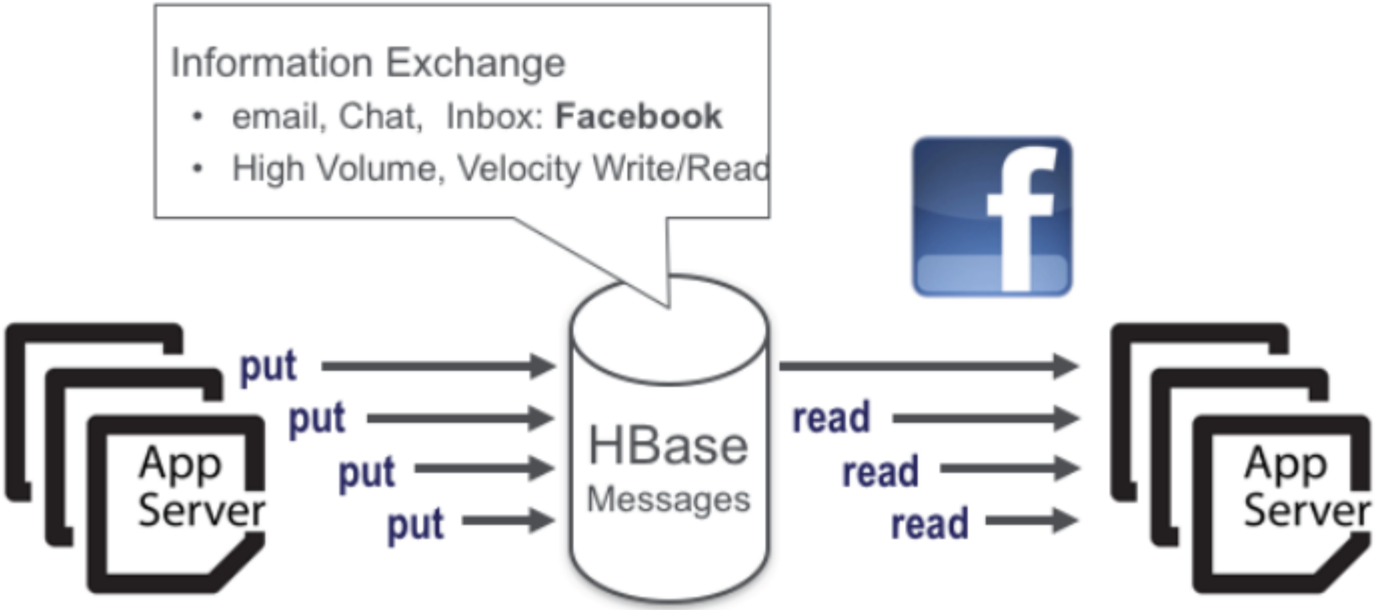
- Capturing Incremental data --Time Series Data
  - High Volume, Velocity Writes
- Information Exchange, Messaging
  - High Volume, Velocity Write/Read
- Content Serving, Web Application Backend
  - High Volume, Velocity Reads

Time Series Data

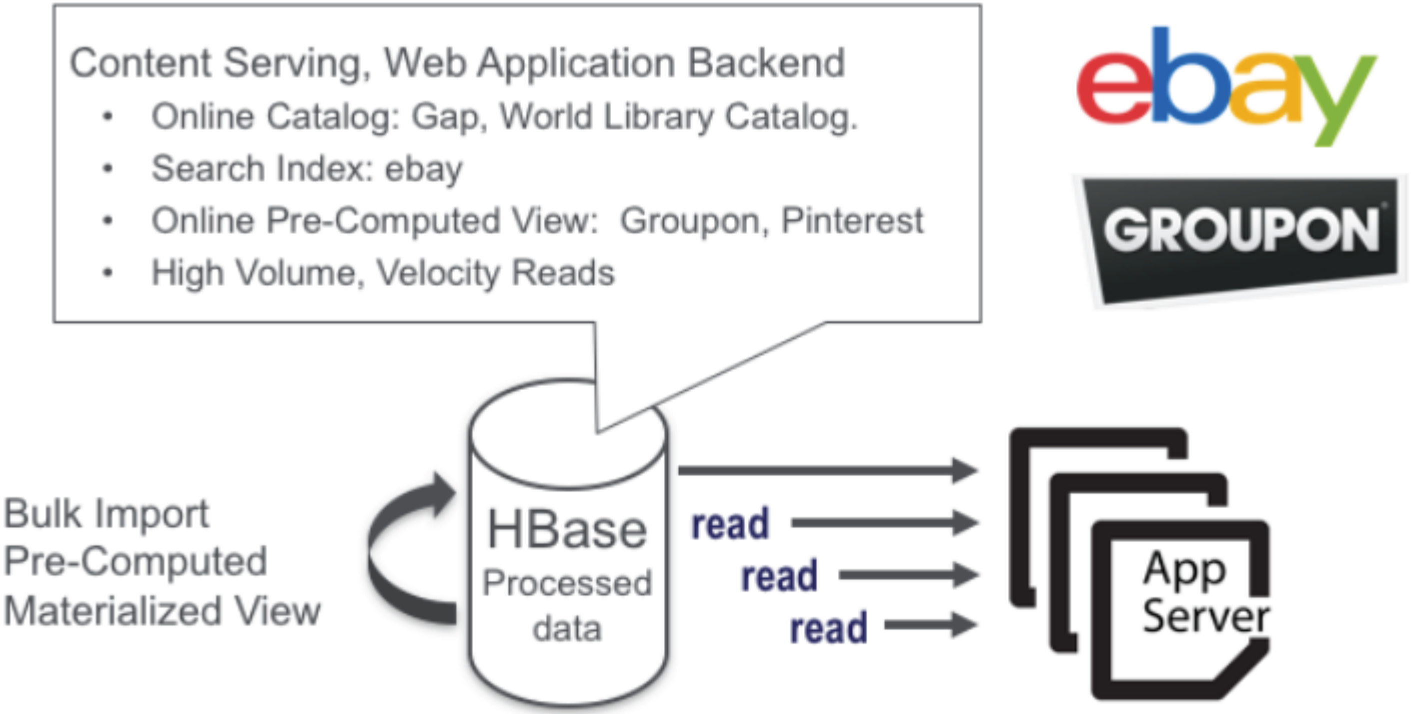


- Sensor, System Metrics, Events, log files
- Stock Ticker, User Activity
- Hi Volume, Velocity Writes

Information Exchange



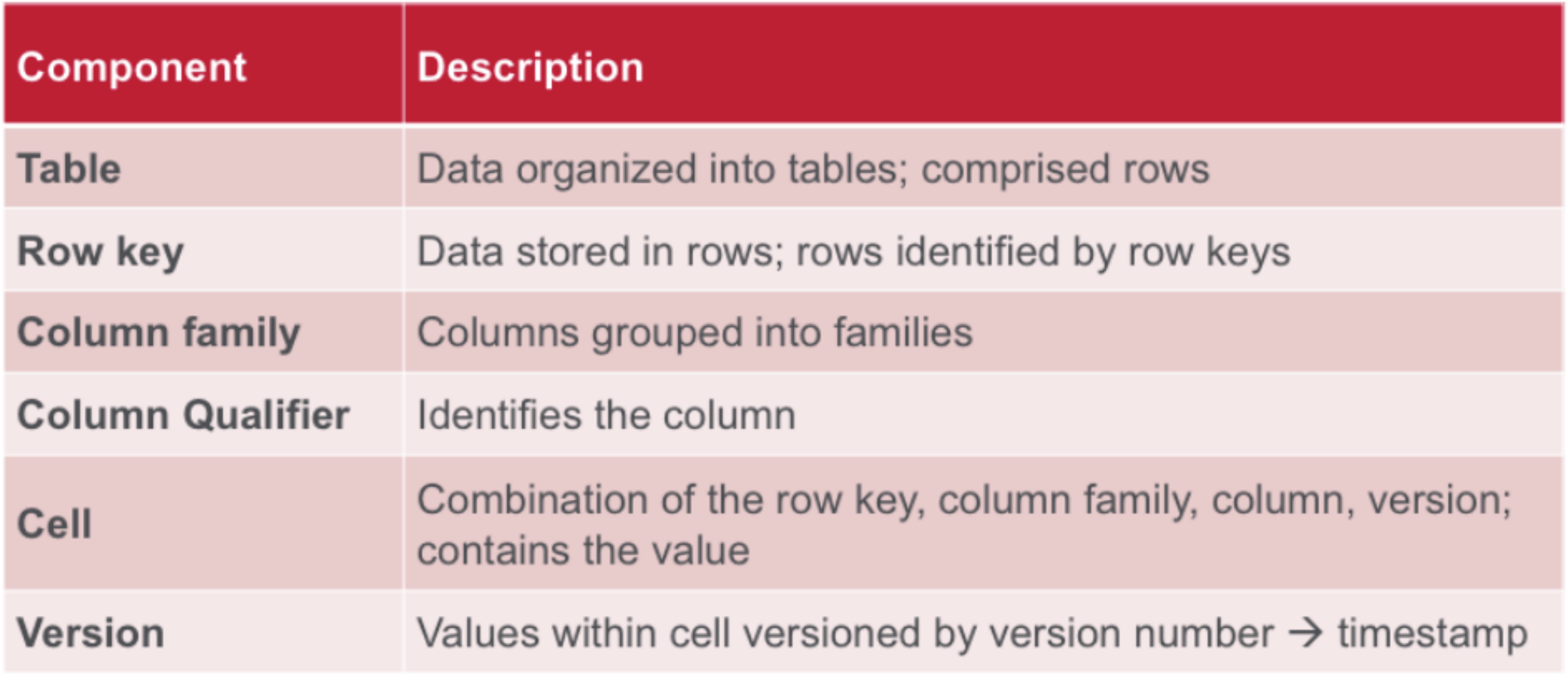
Content Serving


Content Serving, Web Application Backend
• Online Catalog: Gap, World Library Catalog.
• Search Index: ebay
• Online Pre-Computed View: Groupon, Pinterest
• High Volume, Velocity Reads
ebay
GROUPON
Bulk Import
Pre-Computed
Materialized View
HBase
Processed
data
read
read
read
App
Server

HBase Data Model

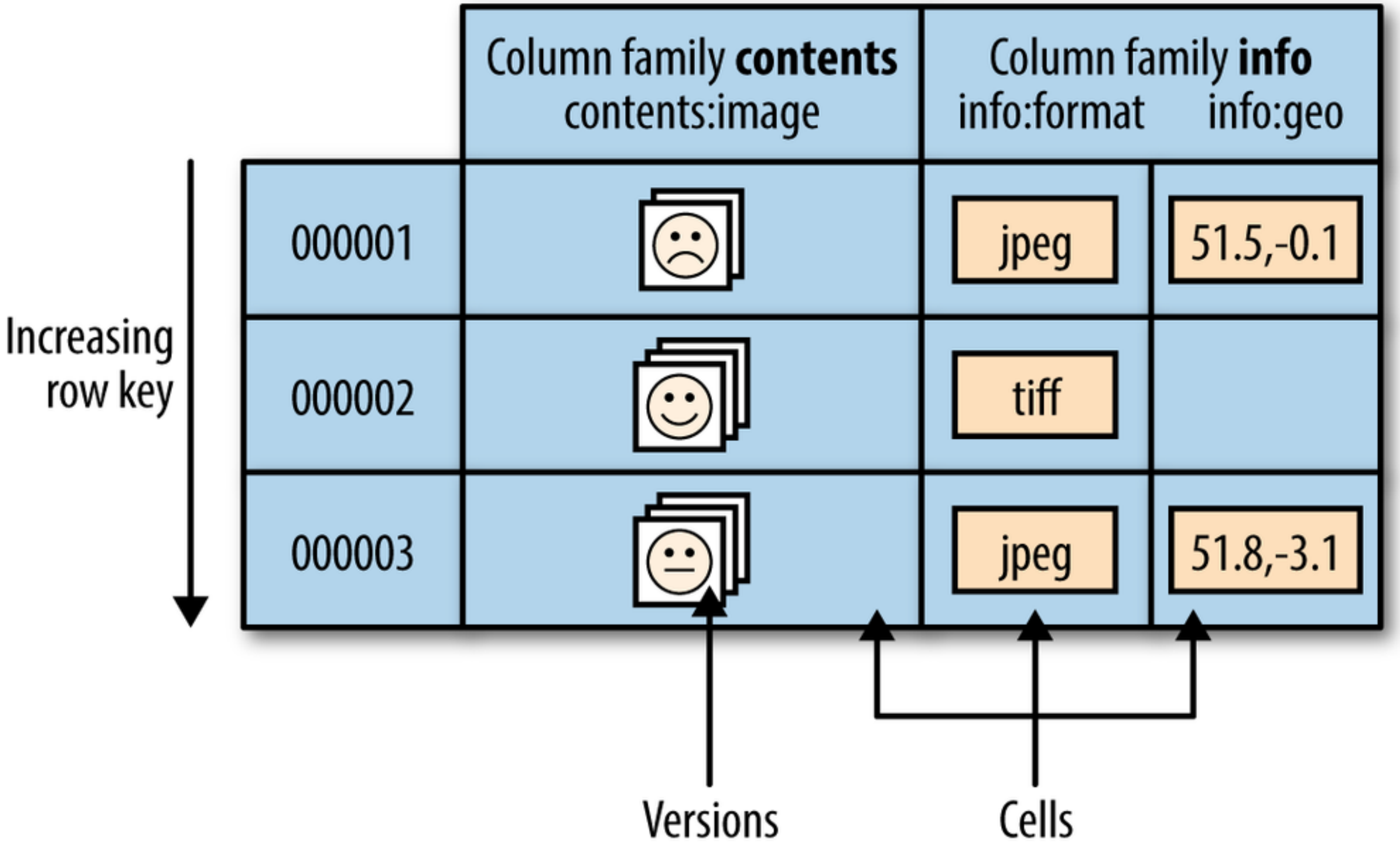
- Applications store data in tables
- Tables are made of rows and columns
- Table cells—the intersection of row and column coordinates—are versioned
- By default, their version is a timestamp auto-assigned by HBase at the time of cell insertion
- A cell’s content is an uninterpreted array of bytes

HBase Data Model Components



Component	Description
Table	Data organized into tables; comprised rows
Row key	Data stored in rows; rows identified by row keys
Column family	Columns grouped into families
Column Qualifier	Identifies the column
Cell	Combination of the row key, column family, column, version; contains the value
Version	Values within cell versioned by version number → timestamp

HBase Data Model



	Column family **contents**	Column family **info**	
	contents:image	info:format	info:geo
000001	☹	jpeg	51.5,-0.1
000002	☺	tiff	
000003	😐	jpeg	51.8,-3.1

HBase Data Model

- Table row keys are also byte arrays
  - So theoretically anything can serve as a row key, from strings to binary representations of long or even serialized data structures
- Table rows are sorted by row key, aka the table's primary key. The sort is byte-ordered
- All table accesses are via the primary key, there are no secondary indexes
- Row columns are grouped into *column families*
- All column family members have a common prefix…
- Columns info:format and info:geo are both members of the info column family…
- While contents:image belongs to the contents family

HBase Data Model

- A table's column families must be specified up front as part of the table schema definition
- But new column family members (new columns) can be added on demand
- For example, a new column info:camera can be offered by a client as part of an update, and its value persisted
  - As long as the column family info already exists on the table
- All column family members are stored together on the filesystem

HBase Data Model

- HBase tables are like those in an RDBMS except…
- Cells are versioned
- Rows are sorted
- Columns can be added on the fly by the client as long as the column family they belong to preexists

HBase Shell

```
$ hbase shell
hbase>
```

- After you have started HBase, you can access the database interactively by using the HBase Shell

Using the HBase Shell

- Names identifying tables and columns need to be quoted
- There is no need to quote constants
- Command parameters are separated using commas
- To run a command after typing it in the shell hit enter key
- Double quoting is required when you need to use binary keys or values in the shell
- To separate keys and values you use the => character
- To specify a key you use predefined constants like NAME, VERSIONS and COMPRESSIONS

Using the HBase Shell

- To get help and see all commands, use the help command
- To get help on a specific command, use help "command”

```
hbase> help "create"
```

Create a Table

- In its simplest form the create command is used to create a table by specifying the table name and column family
- To reduce disk space used for storing data it is advisable to use short column family names.
- This is because storage of each value happens in a fully qualified manner
- Frequent change of column names and use of many column families is not good practice
- A compromise design is to have a few column families then you can have many columns in each family.
- The format of naming columns is to specify the column family then the column name (family:qualifier).

Create a Table

- A basic command that creates a table with two column families is shown below

```
CREATE ‘courses’ ‘hadoop’ ‘programming’
```

- To add columns in each column family the command is enhanced as shown below

```
CREATE ‘courses’ ‘hadoop:spark’, ‘programming:java’
```

Create a Table

- HBase allows you to have multiple versions of a row
- This arises because data changes are not applied in place, instead a change results in a new version
- To control how this happens you specify the number of versions or time to live (TTL)
- When any of these settings are exceeded rows are removed when data compaction is done

- CREATE t1, ’f1’, {NAME => ’f2’, VERSIONS => 3}

Put Data into a Table

- If you’re using HBase, then you likely have data sets that are TBs in size
- As a result, you’ll never actually insert data manually
- However, knowing how to insert data manually could prove useful at times
- Put a cell ‘value’ at specified table/row/column:

```
PUT 'cars', 'row1', 'vi:make', 'bmw'
```

Get Data From a Table

- The get command allows you to get one row of data at a time
- You can optionally limit the number of columns returned
- We’ll start by getting all columns in row1

```
GET 'cars', 'row1’
```

- To get one specific column include the COLUMN option.

```
GET 'cars', 'row1', {COLUMN => 'vi:model'}
```

- You can also get two or more columns by passing an array of columns.

```
GET 'cars', 'row1', {COLUMN => ['vi:model', 'vi:year']}
```

Scan Data From a Table

- Selectively query the contents of a table
- Pass table name and optionally a dictionary of scanner specifications
- Scanner specifications may include one or more of:
  - TIMERANGE, FILTER, LIMIT, STARTROW, STOPROW, TIMESTAMP, MAXLENGTH, COLUMNS, CACHE
- If no columns are specified, all columns will be scanned
- To scan all members of a column family, leave the qualifier empty as in ‘col_family:’

Scan Data From a Table

- We’ll start with a basic scan that returns all columns in the cars table.

```
SCAN 'cars’
```

- The next scan we’ll run will limit our results to the make column qualifier.

```
SCAN 'cars', {COLUMNS => ['vi:make']}
```

- If you have a particularly large result set, you can limit the number of rows returned with the LIMIT option

```
SCAN 'cars', {COLUMNS => ['vi:make'], LIMIT => 4}
```

HBase Scalability

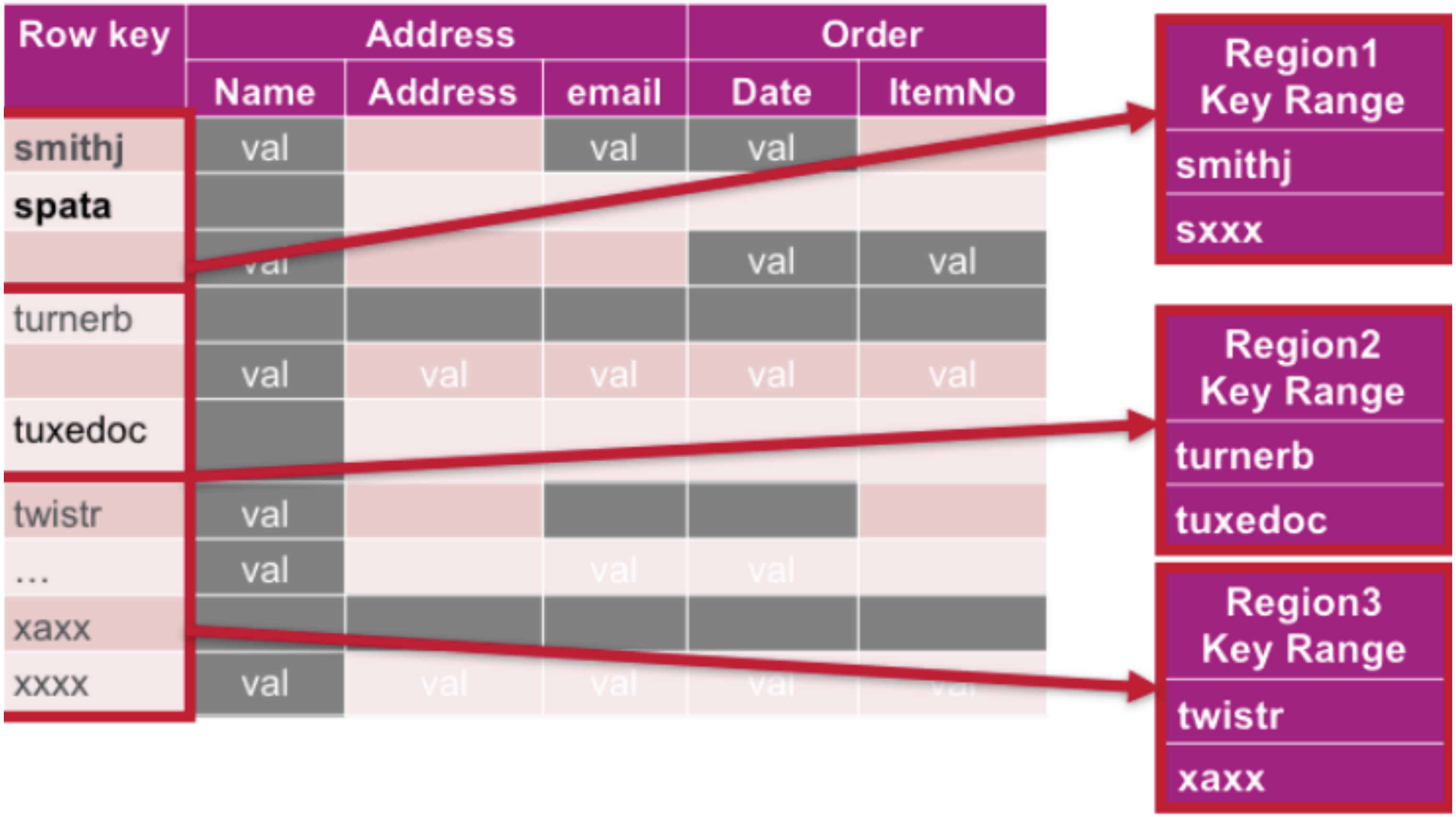
- Tables are automatically partitioned horizontally by HBase into *regions*
- Each region comprises a subset of a table's rows
- A region is denoted by the table it belongs to, its first row (inclusive), and its last row (exclusive)
- Initially, a table comprises a single region, but as the region grows it eventually crosses a configurable size threshold…
- At which point it splits at a row boundary into two new regions of approximately equal size
- Until this first split happens, all loading will be against the single server hosting the original region

HBase Scability

- As the table grows, the number of its regions grows
- Regions are the units that get distributed over an HBase cluster
- In this way, a table that is too big for any one server can be carried by a cluster of servers, with each node hosting a subset of the table’s total regions
- This is also the means by which the loading on a table gets distributed
- The online set of sorted regions comprises the table’s total content.

HBase Scalability

- A continuous, sorted set of rows that are stored together is referred to as a region (subset of table data)



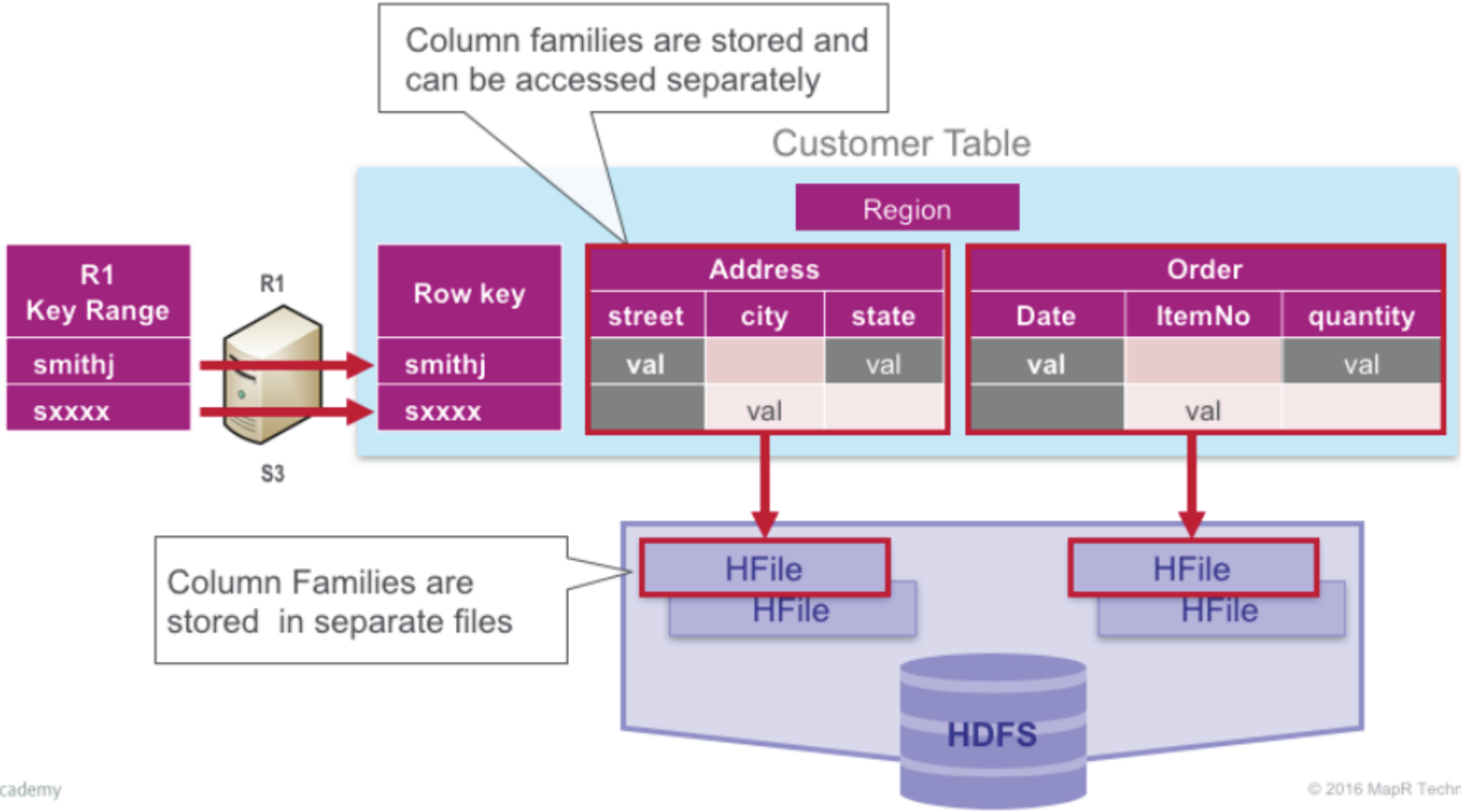
Row key	Address			Order	
	Name	Address	email	Date	ItemNo
smithj	val		val	val	
spata					
	val			val	val
turnerb					
	val	val	val	val	val
tuxedoc					
twistr	val				
…	val		val	val	
xaxx					
xxxx	val	val	val	val	val

Region1 Key Range
smithj
sxxx

Region2 Key Range
turnerb
tuxedoc

Region3 Key Range
twistr
xaxx

HBase Scalability

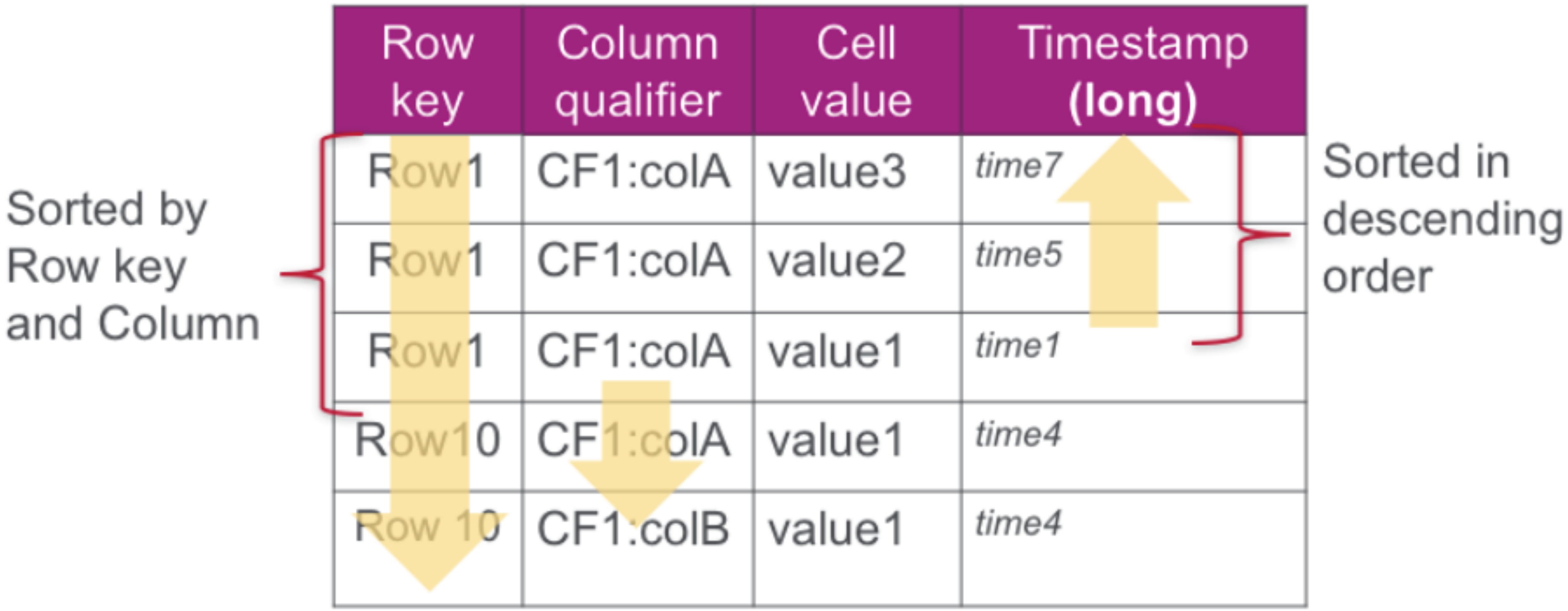


Column families are stored and can be accessed separately

Column Families are stored in separate files

HFile Physical View

Physically data is stored per Column family as a sorted map
Ordered by **row key, column qualifier** in ascending order
Ordered by **timestamp** in descending order



Row key	Column qualifier	Cell value	Timestamp **(long)**
Row1	CF1:colA	value3	*time7*
Row1	CF1:colA	value2	*time5*
Row1	CF1:colA	value1	*time1*
Row10	CF1:colA	value1	*time4*
Row 10	CF1:colB	value1	*time4*

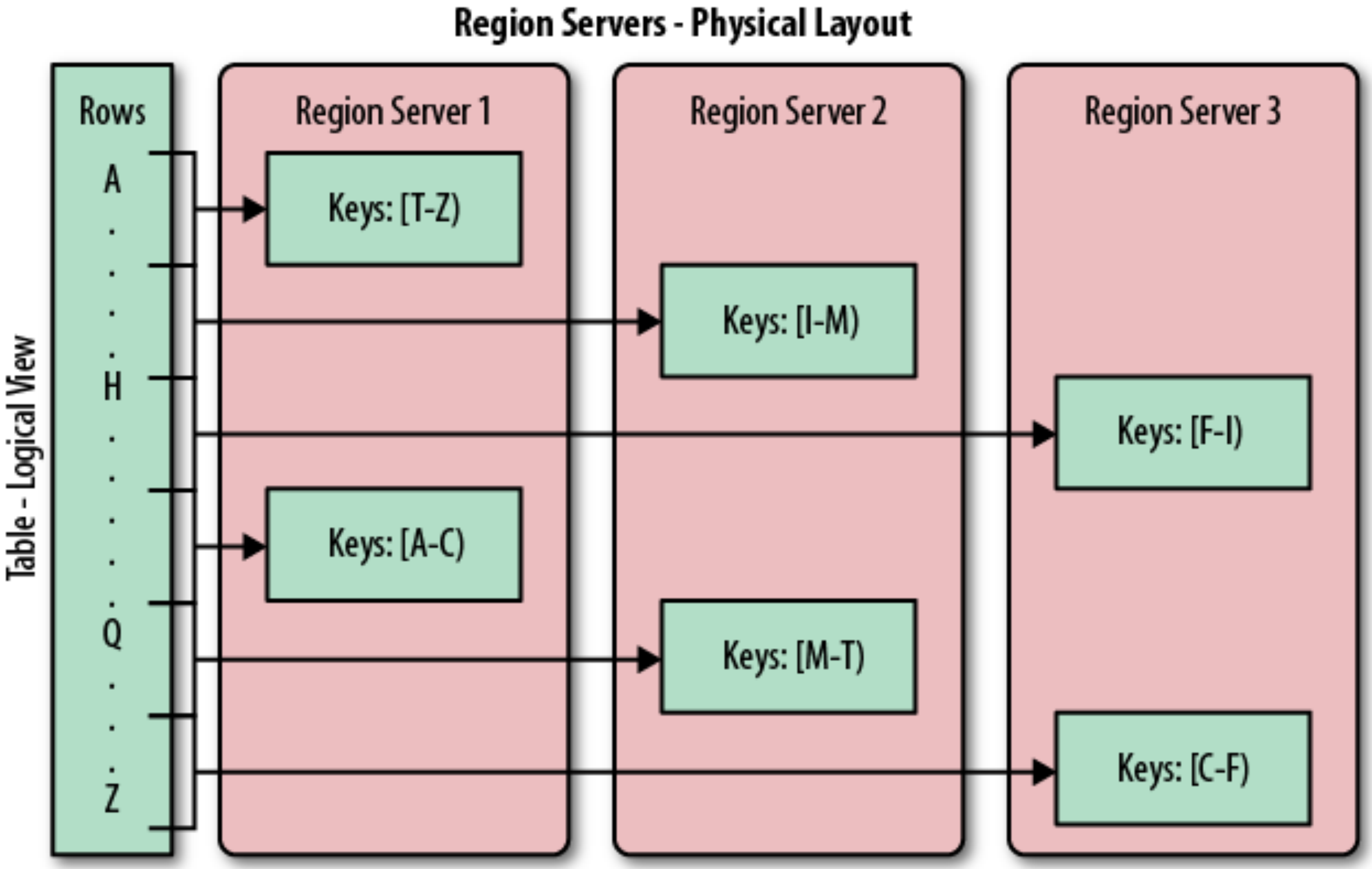
Sorted by Row key and Column

Sorted in descending order

HBase Scalability

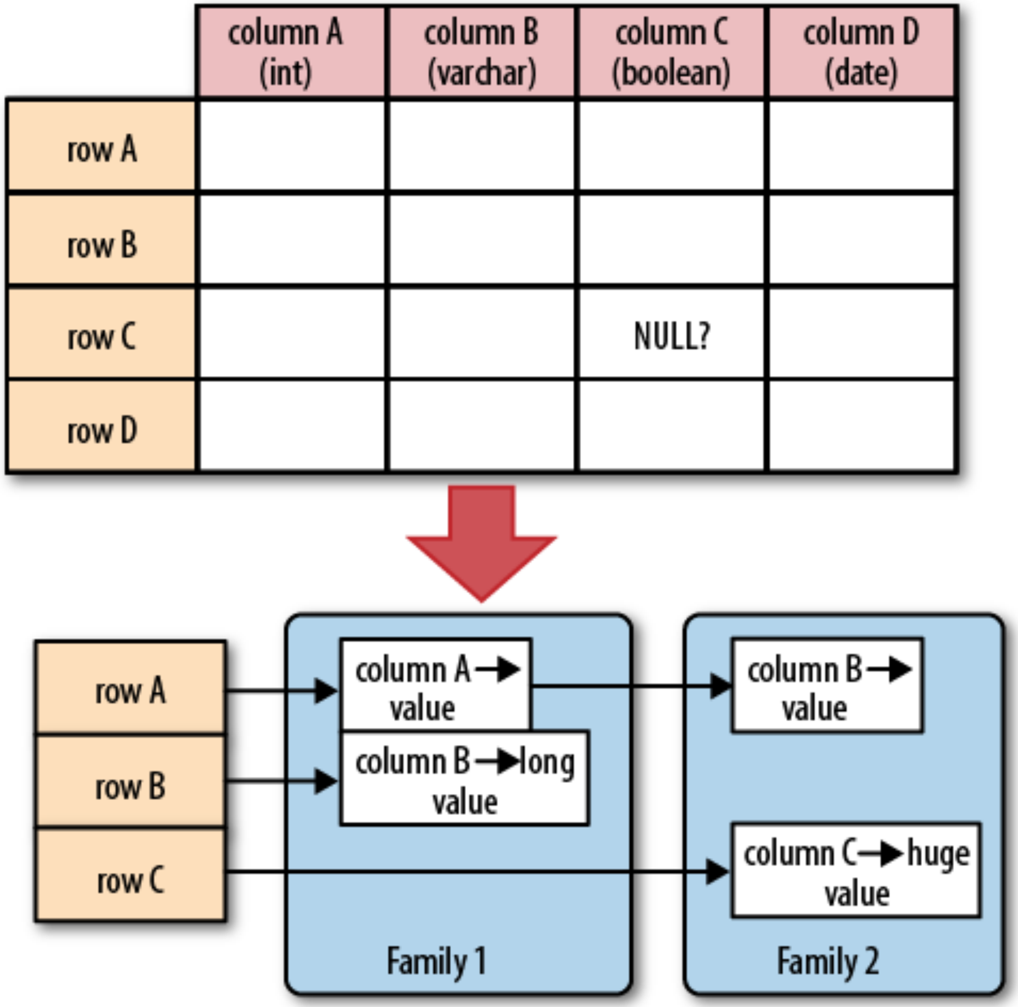
- So this is why HBase is described as a column store
- Each region holds an ordered subset of a table's rows
- Within a region each column family is stored separately (in an HFile)
- So queries on just one column family of a table result in faster lookups and also the need to scan less data
- Queries that span regions can be parallelized to the extent that regions are stored across multiple nodes (region servers)
- Tables can grow to terabytes and more while no region will grow to beyond a configured maximum size before automatic resharding occurs

HBase Scalability



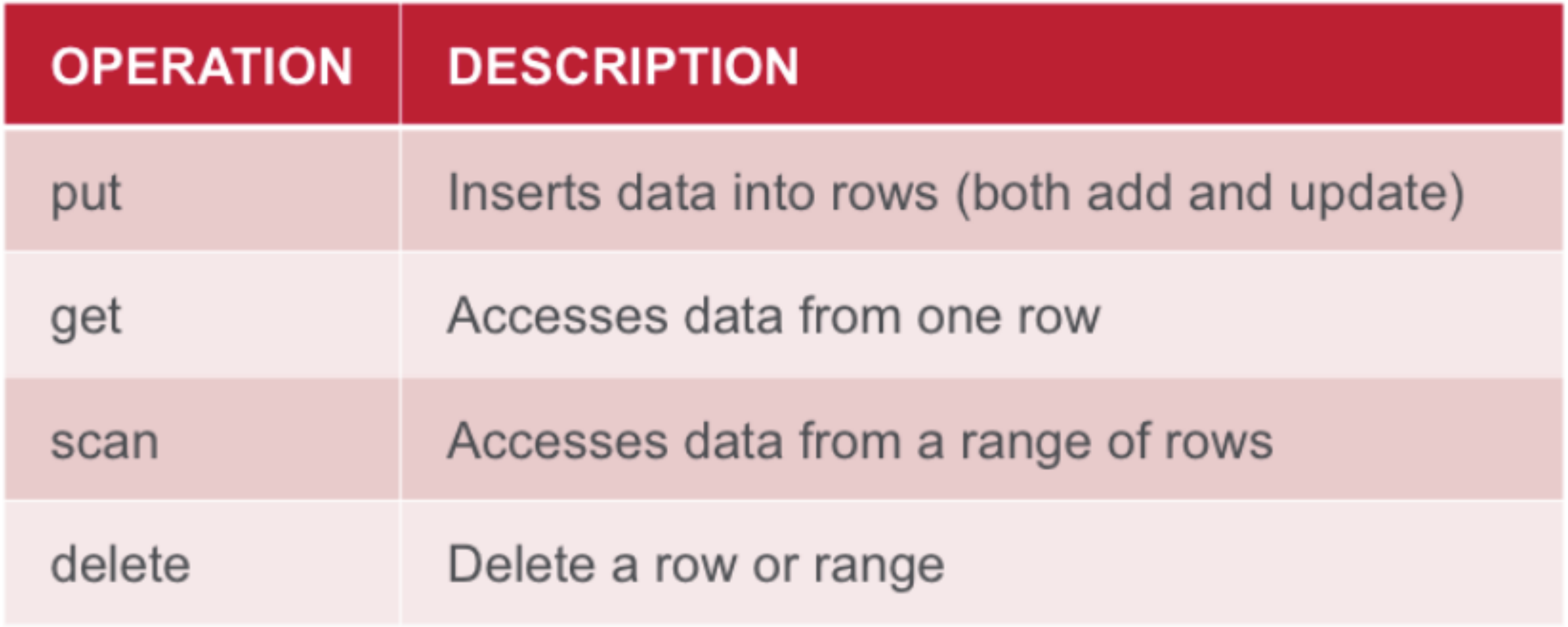
Sparse Table Storage

- For a database with a fixed schema, you have to store NULLs where there is no value
- But for HBase you simply omit the whole column…
- In other words, NULLs do not occupy any storage space



	column A (int)	column B (varchar)	column C (boolean)	column D (date)
row A				
row B				
row C			NULL?	
row D				

Basic Data Access Operations



OPERATION	DESCRIPTION
put	Inserts data into rows (both add and update)
get	Accesses data from one row
scan	Accesses data from a range of rows
delete	Delete a row or range

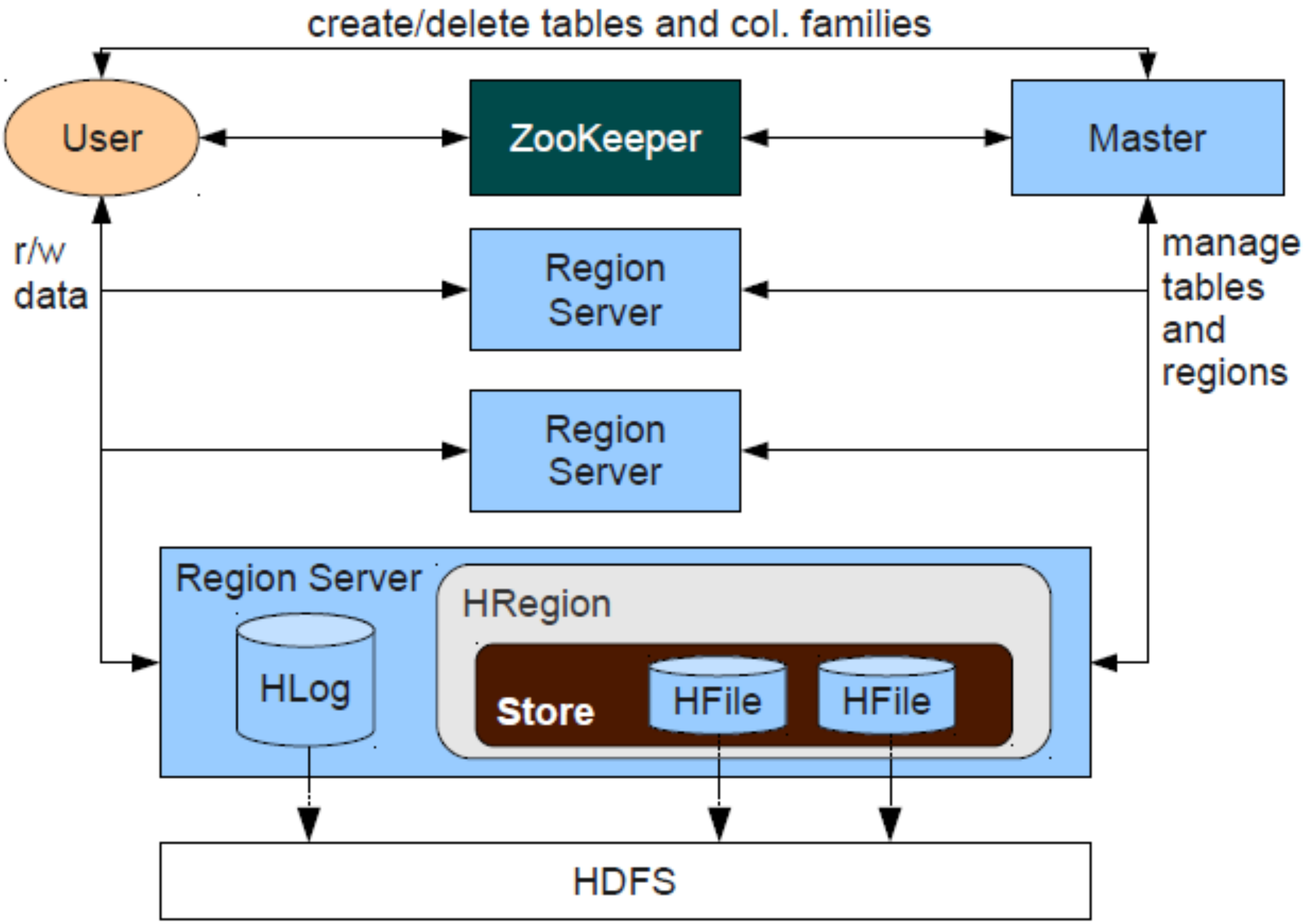
ACID Properties

- HBase is not ACID-compliant, but does guarantee certain specific properties…
- All changes are atomic within a row
- Any put will either wholly succeed or wholly fail
- But API calls that change several rows will not be atomic across the multiple rows
- All rows returned via any access API will consist of a complete (consistent) row that existed at some point in the table's history

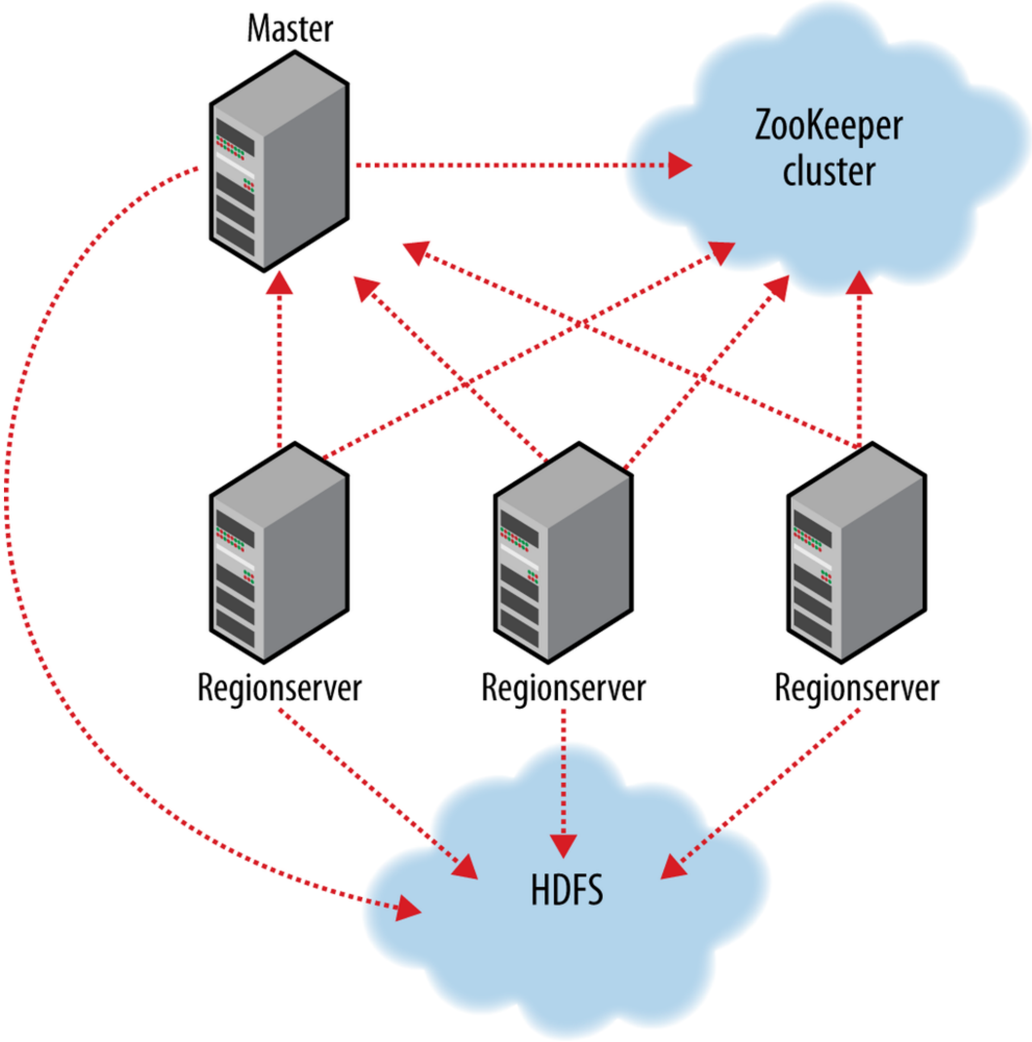
HBase Architecture

- HBase made up of an HBase *master* node orchestrating a cluster of one or more *regionserver* workers
- The HBase master is responsible for assigning regions to registered regionservers, and for recovering regionserver failures
- The regionservers carry zero or more regions and handle client read/write requests
- They also manage region splits, informing the HBase master about the new regions

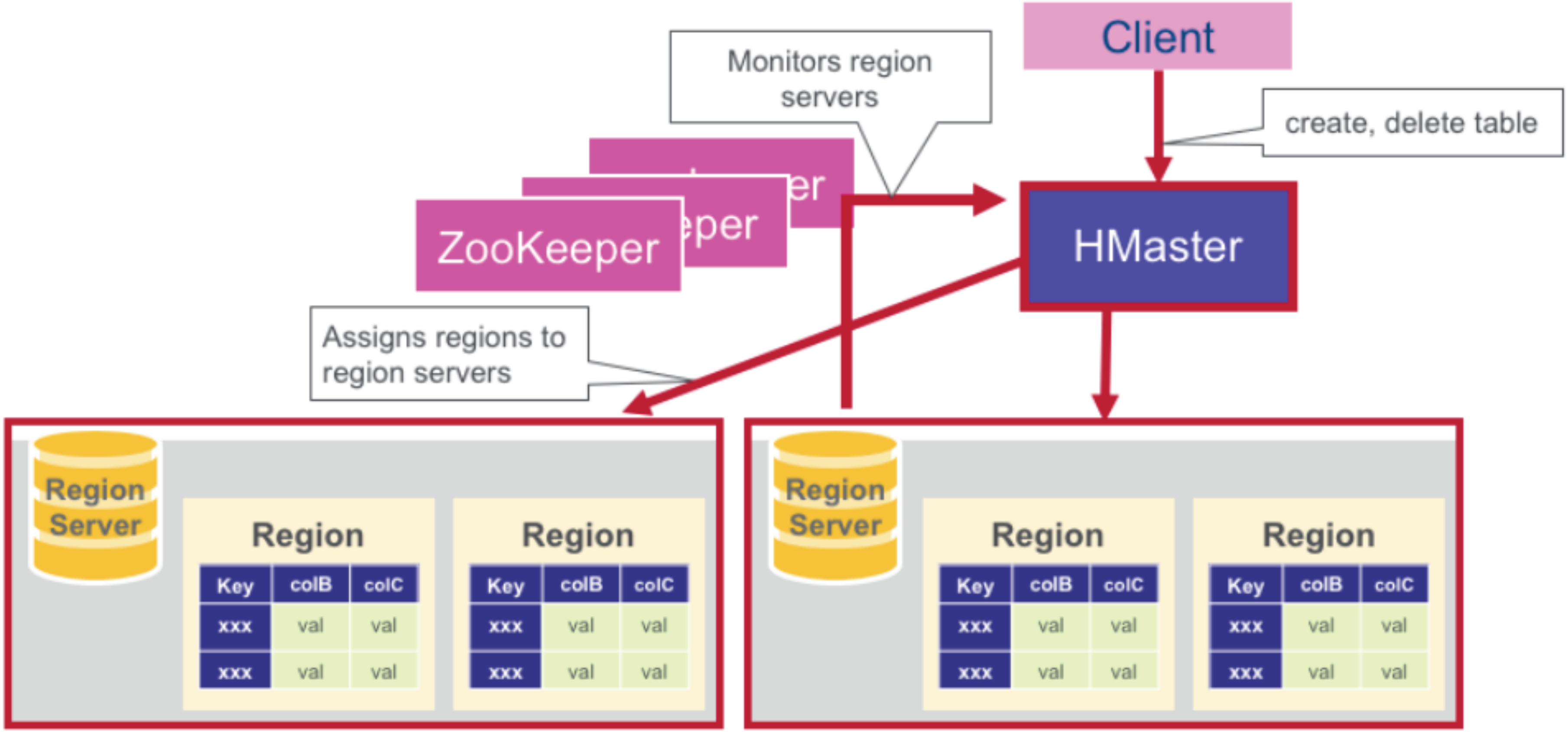
HBase Logical Architecture



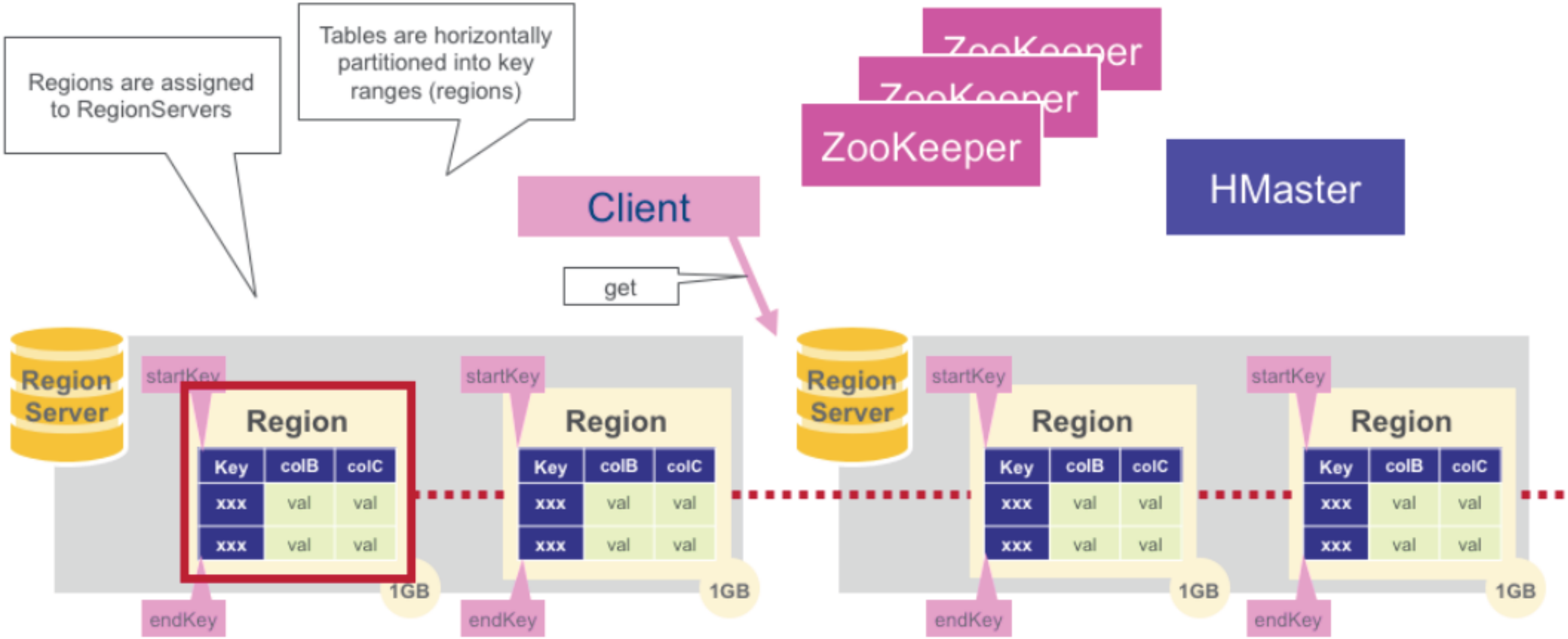
HBase Physical Architecture



HBase HMaster



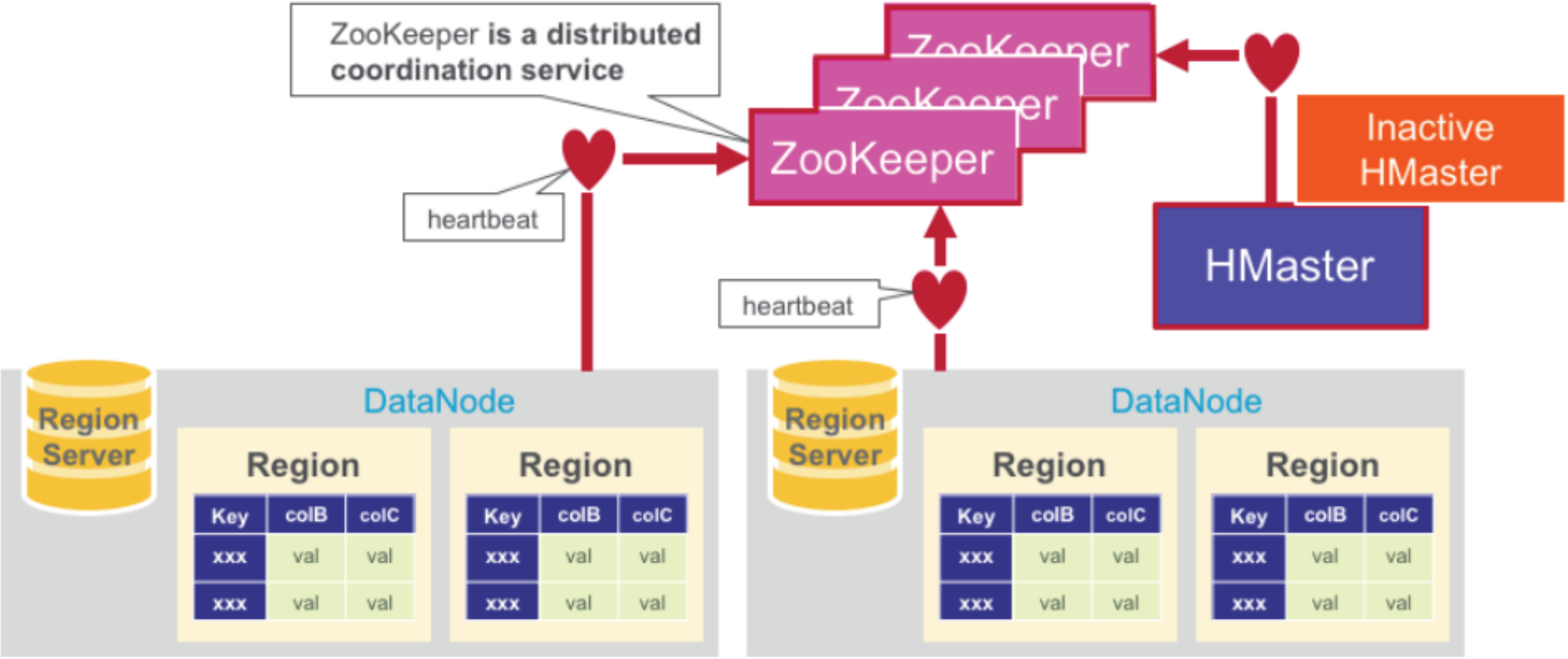
Regions



Regions are assigned to RegionServers

Tables are horizontally partitioned into key ranges (regions)

Zookeeper (Coordinator)



ZooKeeper is a distributed coordination service

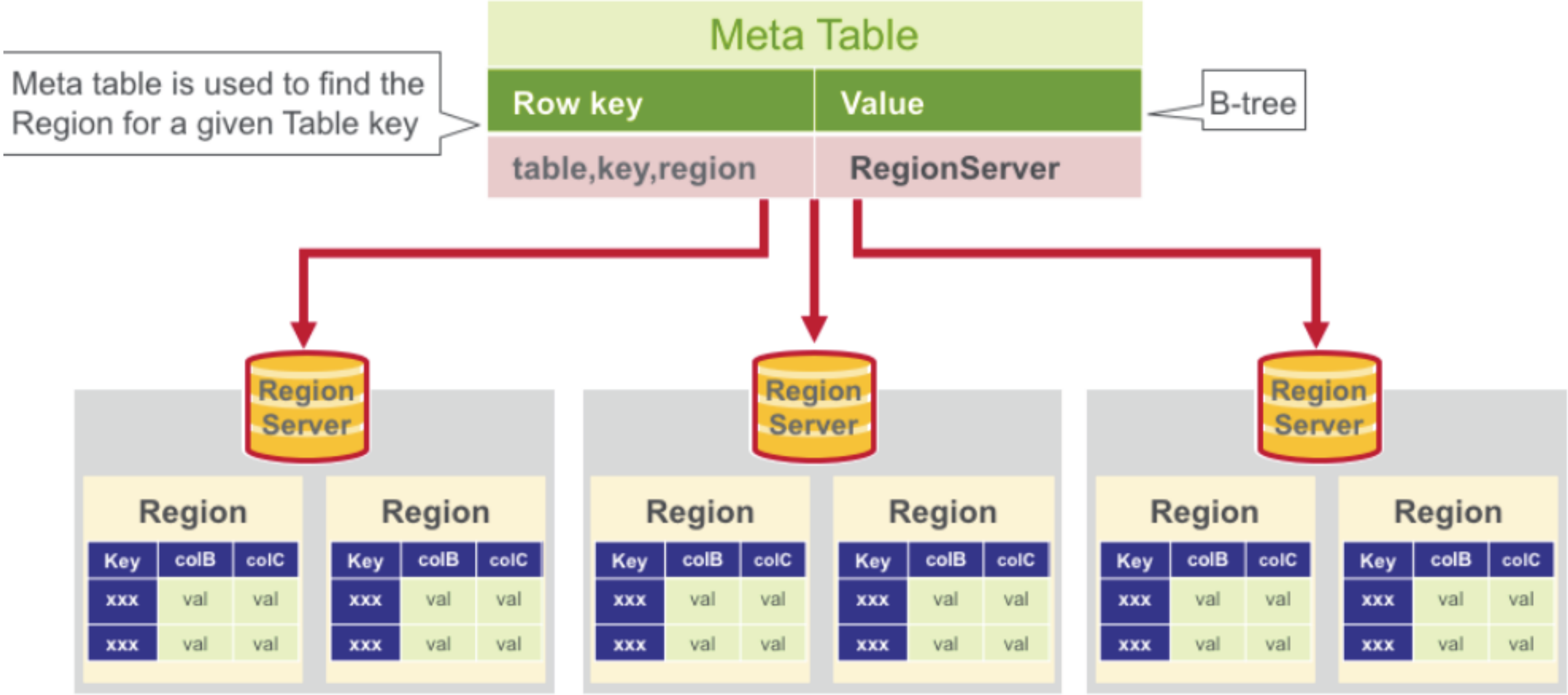
Key	colB	colC
xxx	val	val
xxx	val	val

Key	colB	colC
xxx	val	val
xxx	val	val

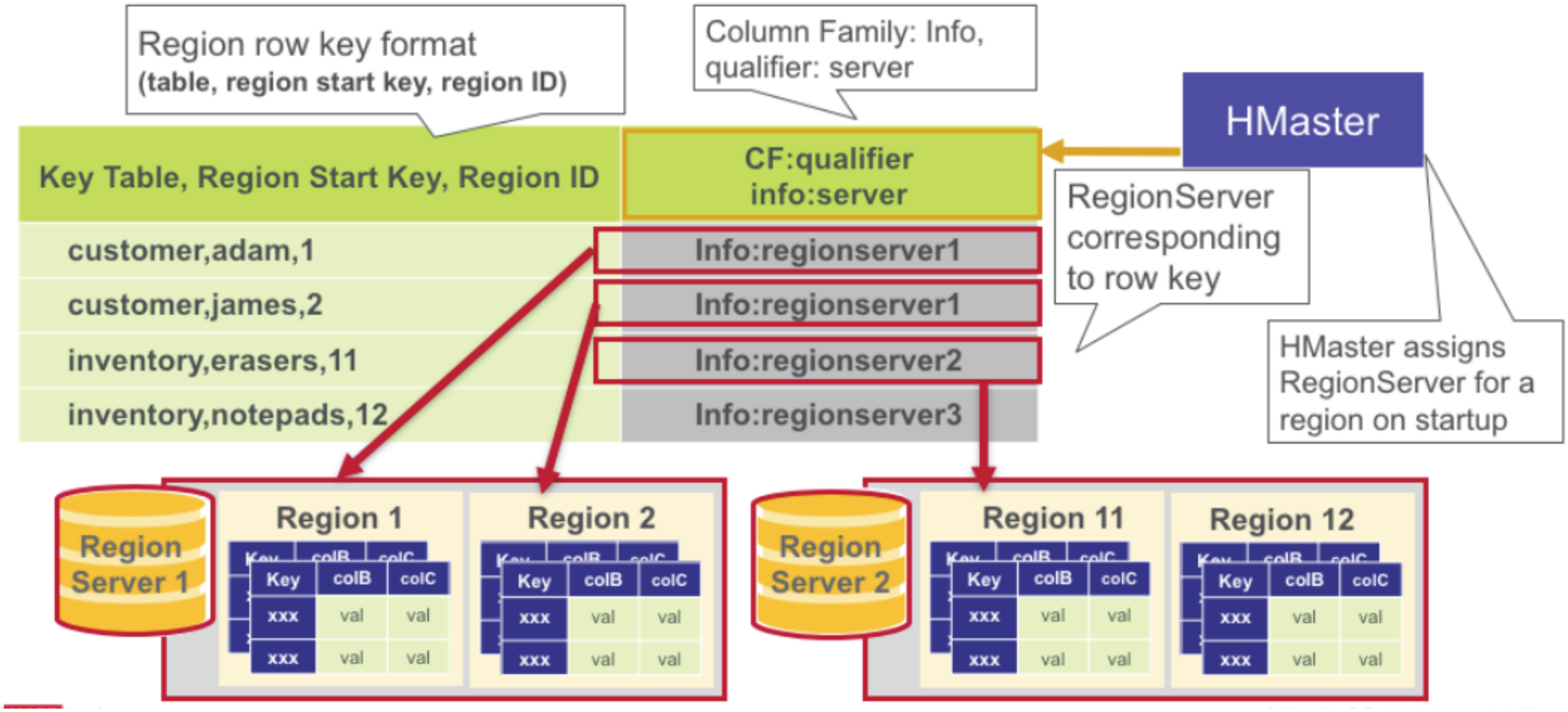
Key	colB	colC
xxx	val	val
xxx	val	val

Key	colB	colC
xxx	val	val
xxx	val	val

HBase Meta Table



HBase Meta Table



Region row key format
(table, region start key, region ID)

Column Family: Info,
qualifier: server

Key Table, Region Start Key, Region ID	CF:qualifier info:server
customer,adam,1	Info:regionserver1
customer,james,2	Info:regionserver1
inventory,erasers,11	Info:regionserver2
inventory,notepads,12	Info:regionserver3

RegionServer corresponding to row key

HMaster assigns RegionServer for a region on startup