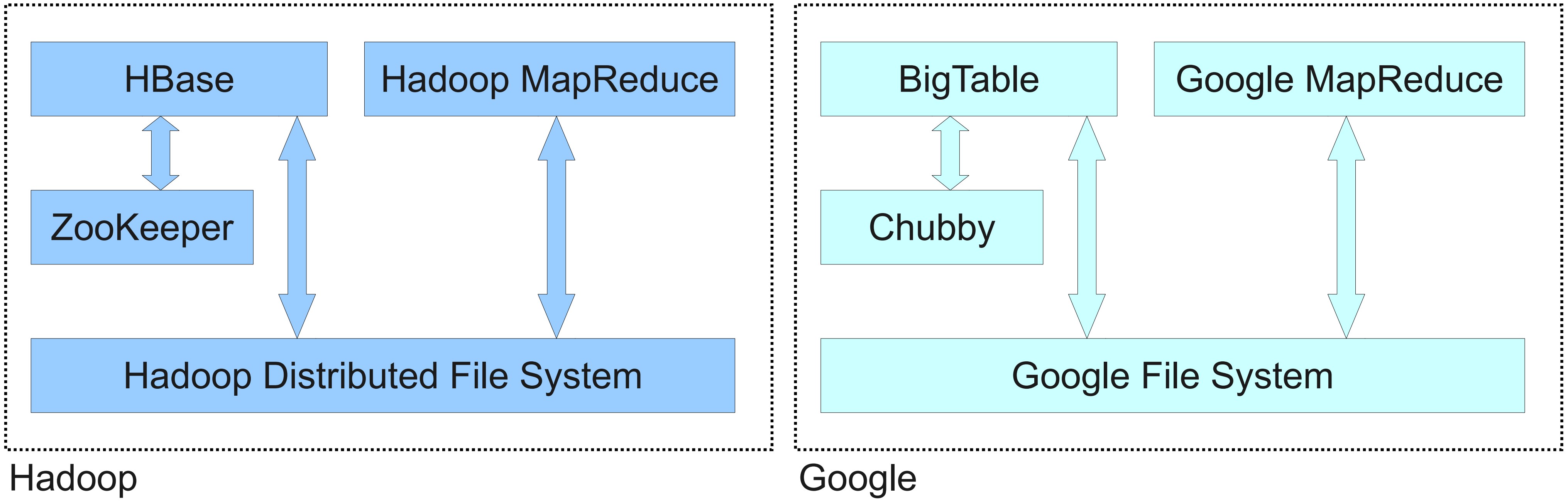
Storage of Structured Data: BigTable and HBase

DISTRIBUTED
SYSTEMS
LABORATORY

HBase and BigTable

- HBase is Hadoop's counterpart of Google's BigTable
- BigTable meets the need for a highly scalable storage system for structured data
  - Provides random and (almost) real-time data access
  - Works on top of Google File System
  - Data is structured into entities (records), aggregated into few huge files and indexed
  - Not a relational database. Offers typical *create*, *read*, *update* and *delete* (CRUD) ops. plus scan of keys
  - Not ACID guarantees
  - Used by many applications in Google

Hadoop's and Google's stacks



HBase/BigTable Tables

- “*A Bigtable is a sparse, distributed, persistent multidimensional sorted map*”

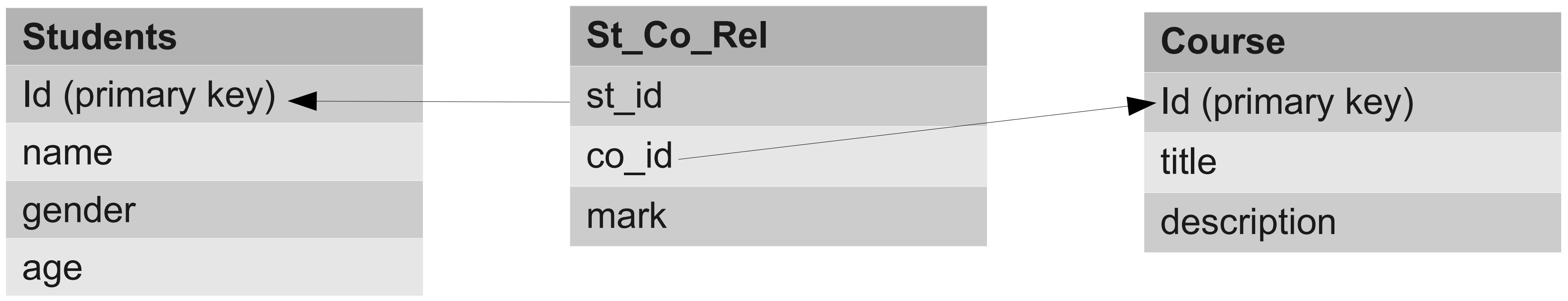
Map	→	Associates keys to values
Sorted	→	Ordered by key (efficient look-ups)
Multidimensional	→	Key is formed by several values
Persistent	→	Once written, it is there until removed
Distributed	→	Stored across different nodes
Sparse	→	Many (most) values are not defined

Hbase/BigTable Datamodel

- *Rows* are composed of *columns*, which are grouped into *column families*
  - Column families group semantically related values
  - Each column family is stored as one file (*HFile*) in HDFS
  - One column family can have *millions* of columns
  - Each column is referenced by *family:qualifier*
- A *row key* is an array of bytes. Keys are ordered lexicographically
- A column value is denoted a *cell*. They have timestamps. Old values are kept

BigTable vs. Relational Datamodels

- Students/courses database



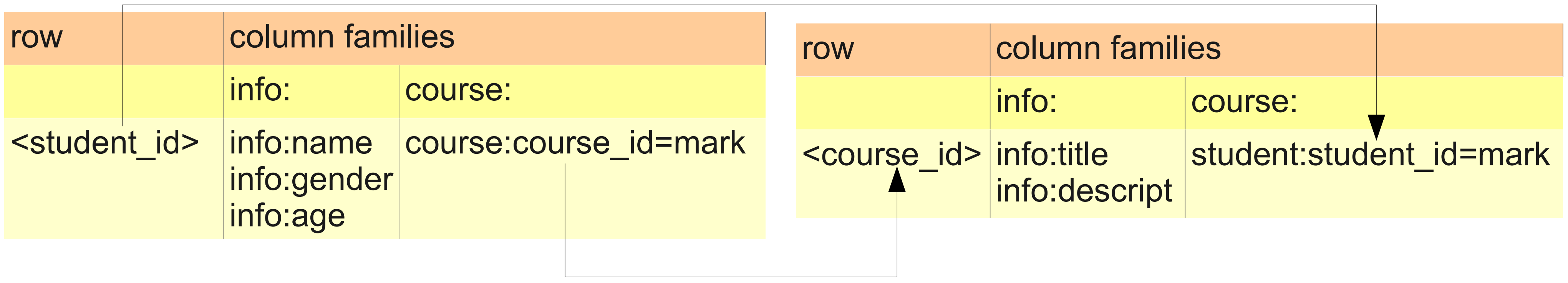
Students
Id (primary key)
name
gender
age

St_Co_Rel
st_id
co_id
mark

Course
Id (primary key)
title
description

Relational Model

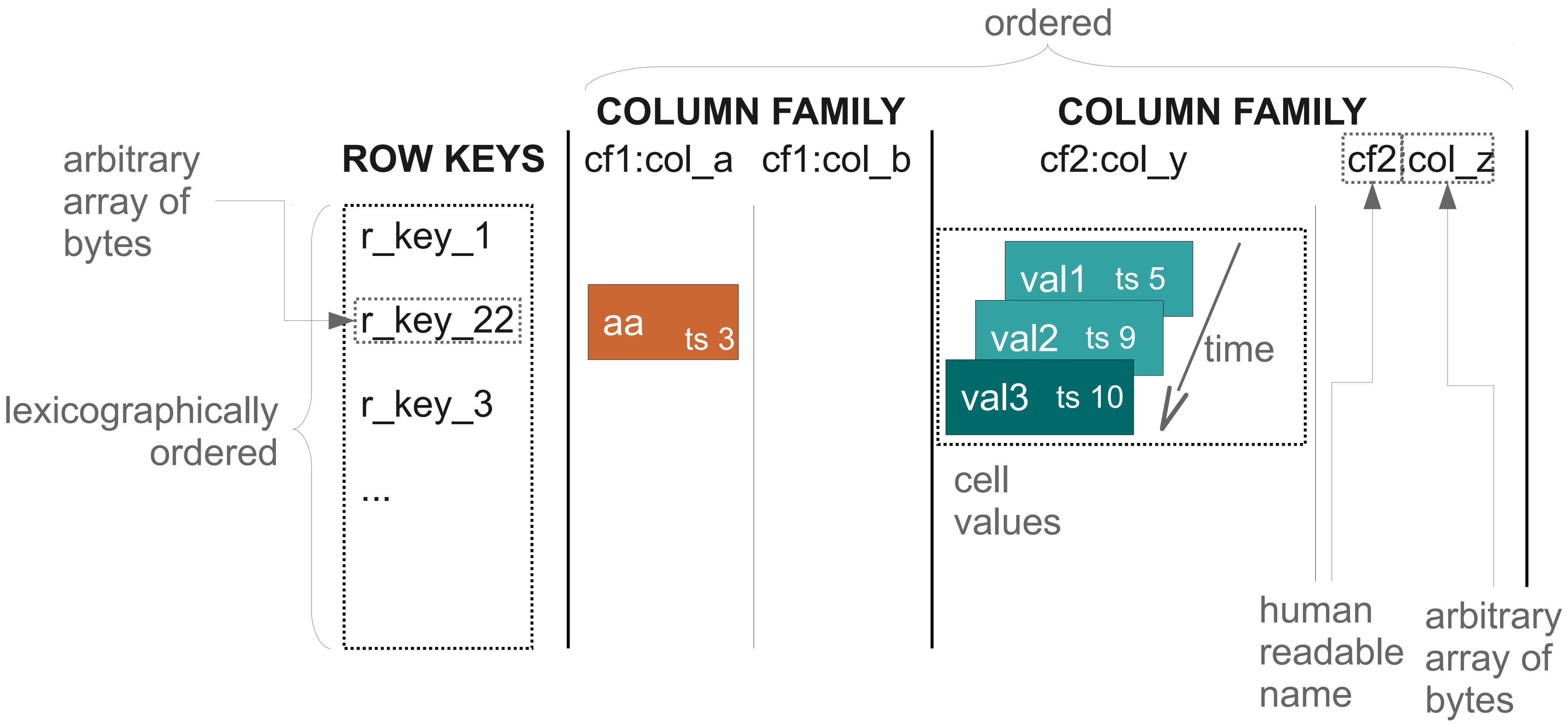
BigTable Model (identifiers handled by app programmers, no referencial integrity)



row	column families	
	info:	course:
<student_id>	info:name info:gender info:age	course:course_id=mark

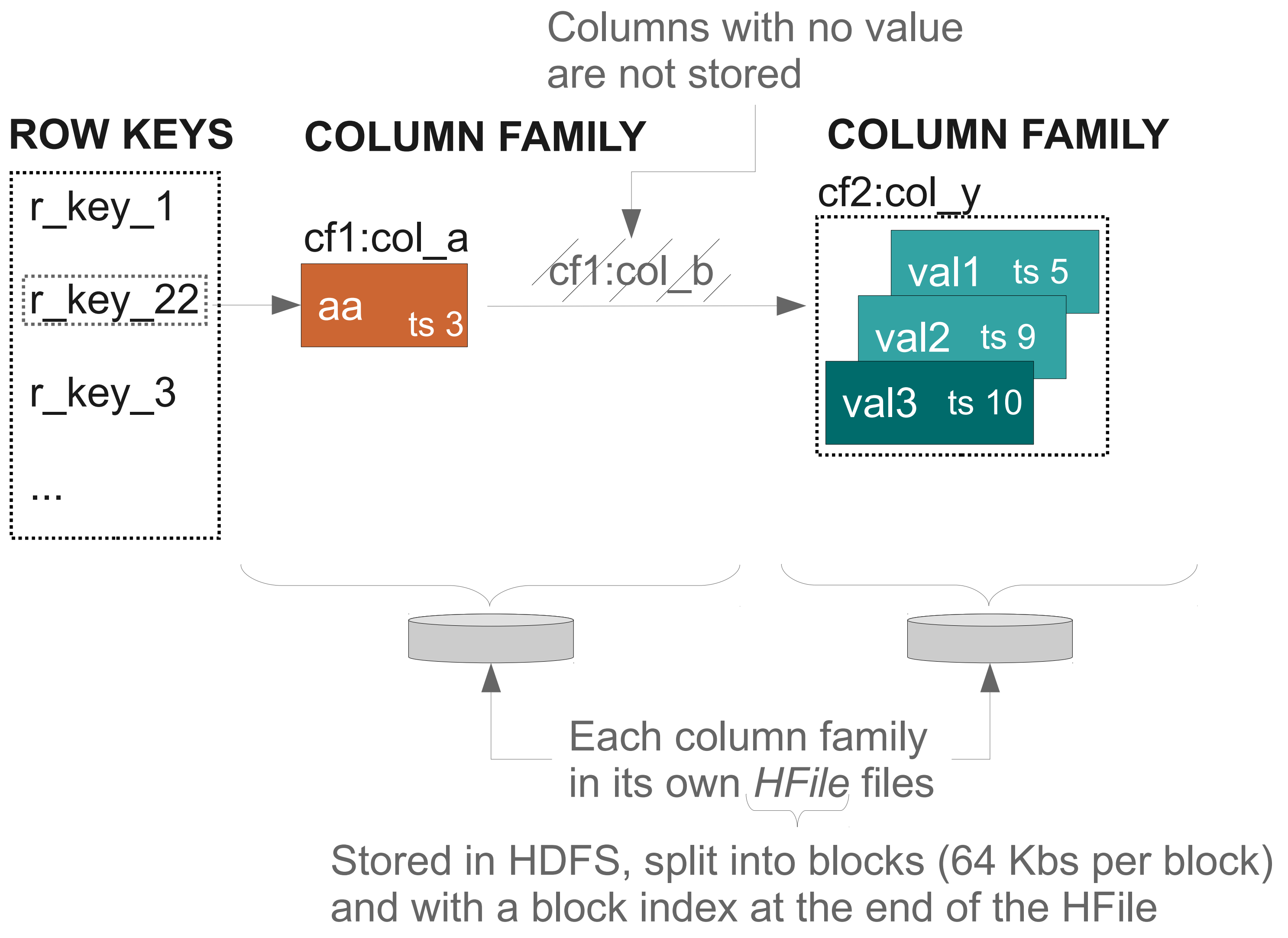
row	column families	
	info:	course:
<course_id>	info:title info:descript	student:student_id=mark

Hbase/BigTable Datamodel View



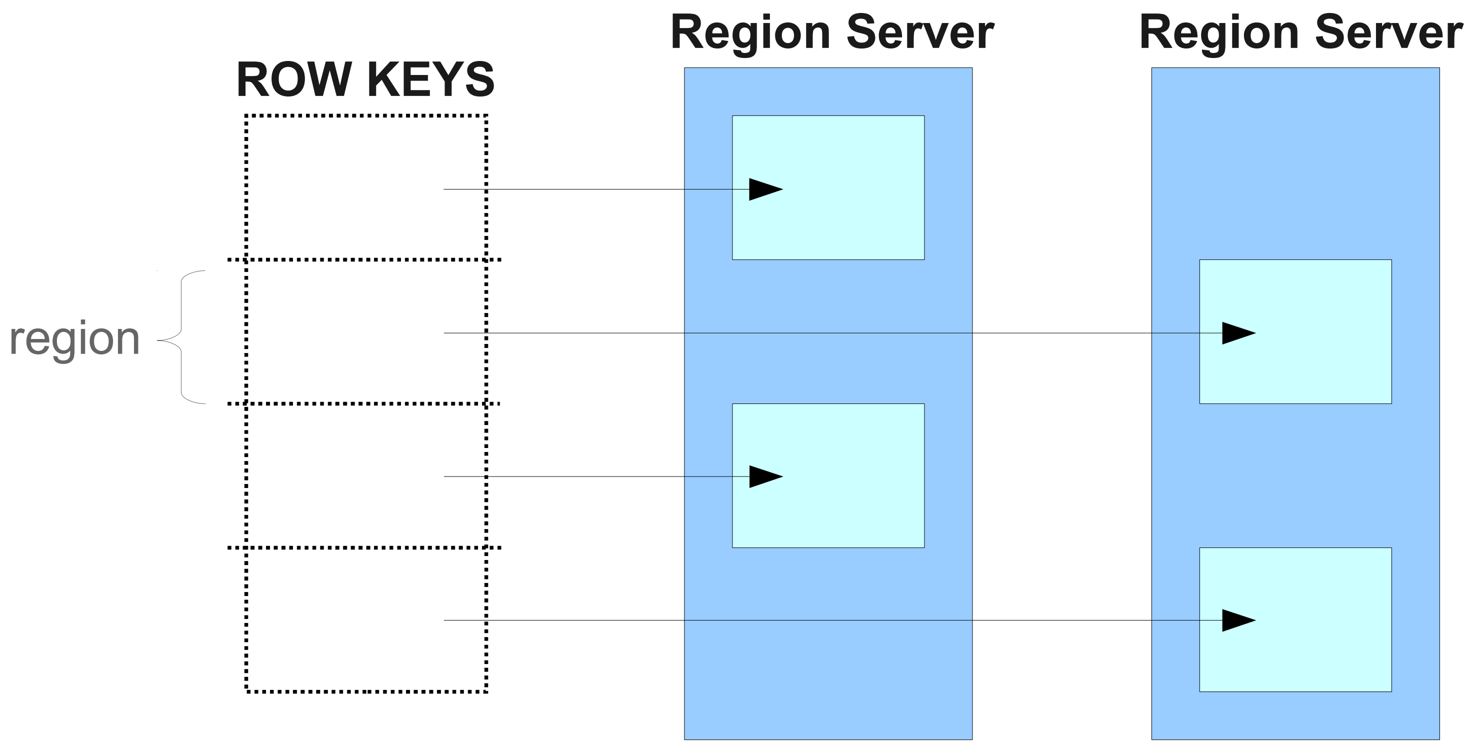
`(Table, RowKey, Family, Column, Timestamp) → Value`

Hbase/BigTable Data Storage

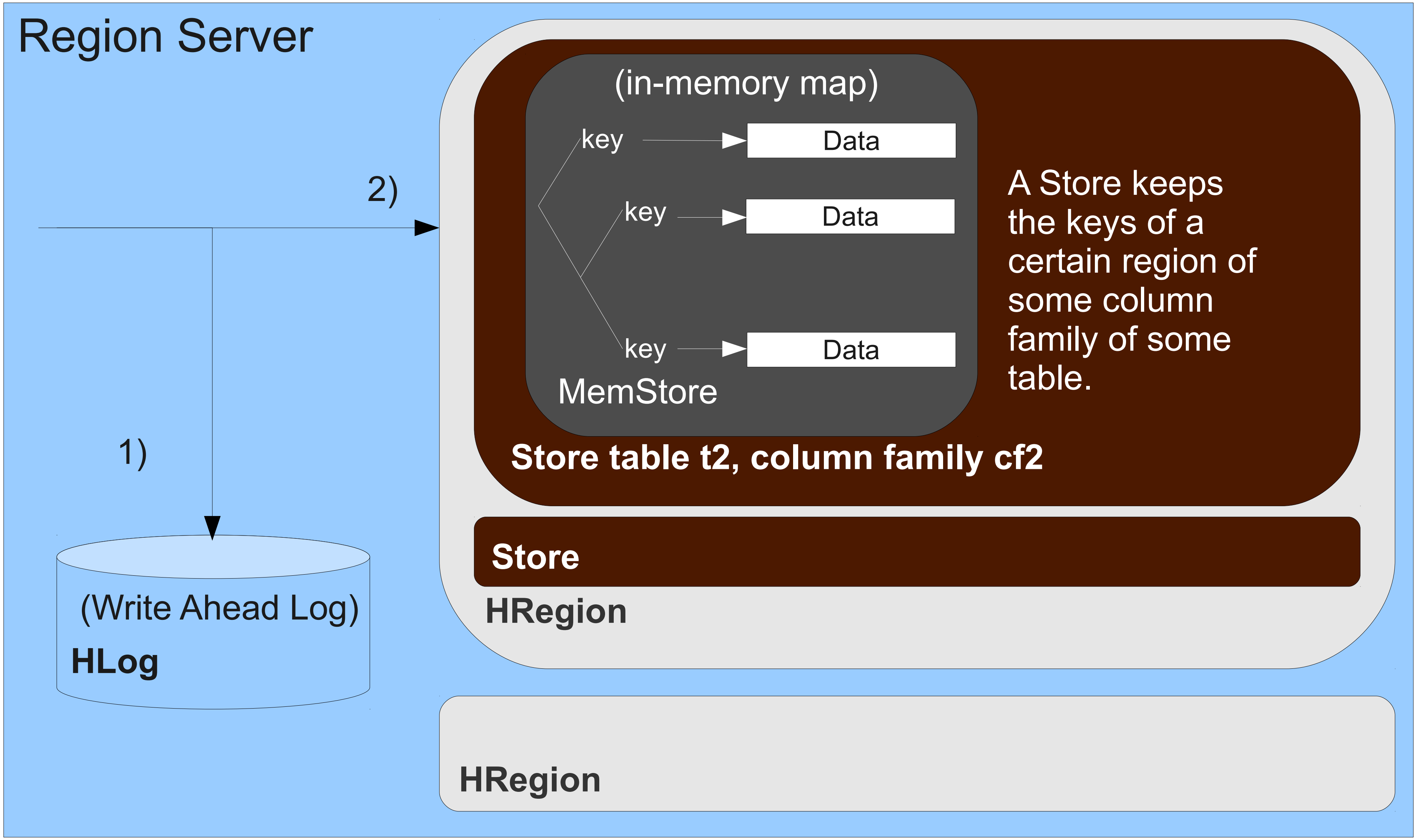


HBase Regions

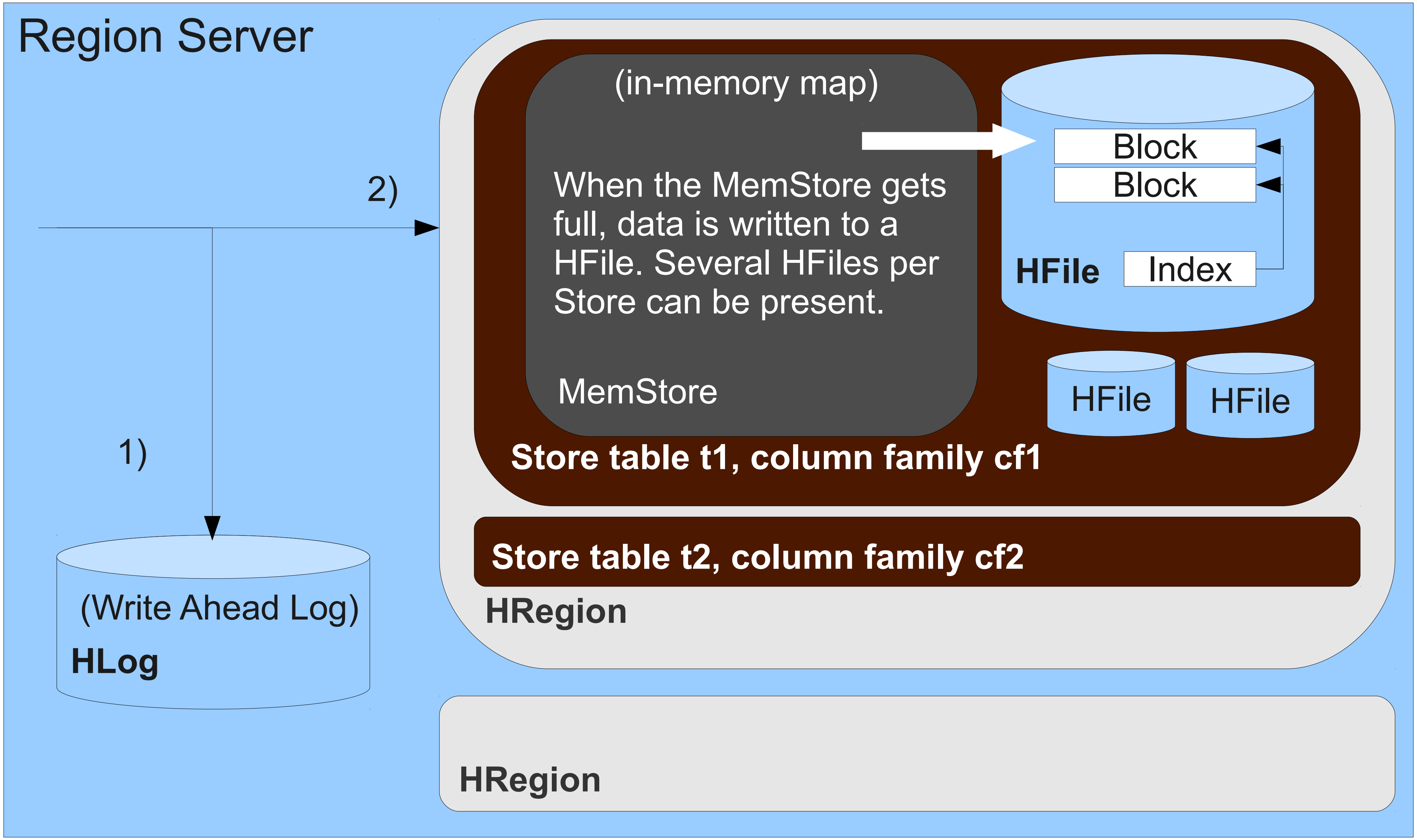
- For scalability, tables are split into *regions* by key
- Each region is assigned to a *region server*



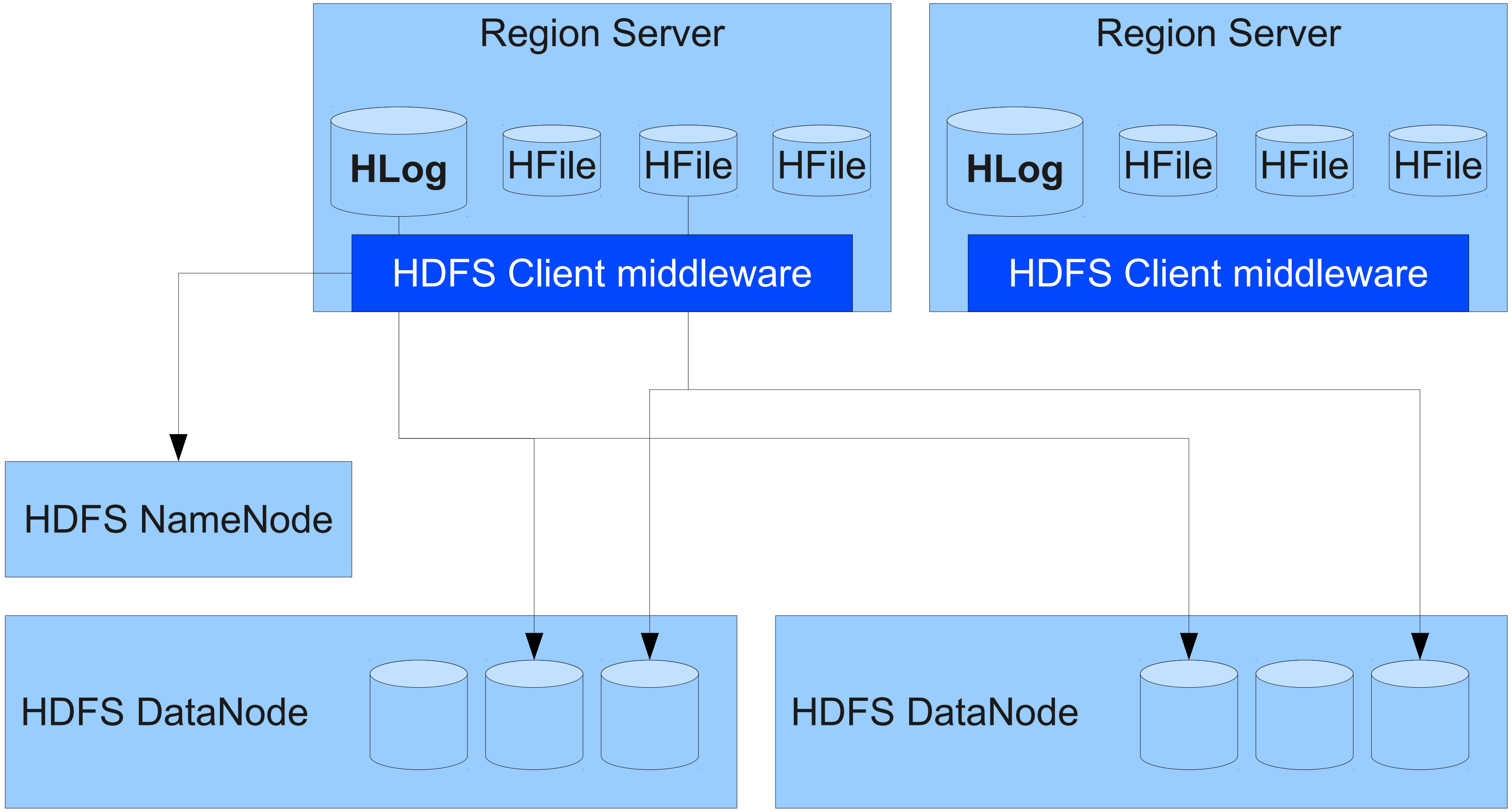
Region Server Internal Architecture


Region Server
2)
1)
(in-memory map)
key
Data
key
Data
key
Data
MemStore
A Store keeps the keys of a certain region of some column family of some table.
Store table t2, column family cf2
Store
HRegion
(Write Ahead Log)
HLog
HRegion

Region Server Internal Architecture


Region Server
2)
1)
(Write Ahead Log)
HLog
(in-memory map)
When the MemStore gets full, data is written to a HFile. Several HFiles per Store can be present.
MemStore
Block
Block
HFile
Index
HFile
HFile
Store table t1, column family cf1
Store table t2, column family cf2
HRegion
HRegion

HBase over HDFS



Reading Data

- When reading data by row key the RegionServer attending the corresponding region is queried
- All HRegions which store a column family whose data is requested by the query must be checked
  - For that, the in-memory map and the HBase files must all be read (merged read)
  - HBase files can use bloom filters to speed-up readings
- Unless otherwise specified, only the last version of each value is returned

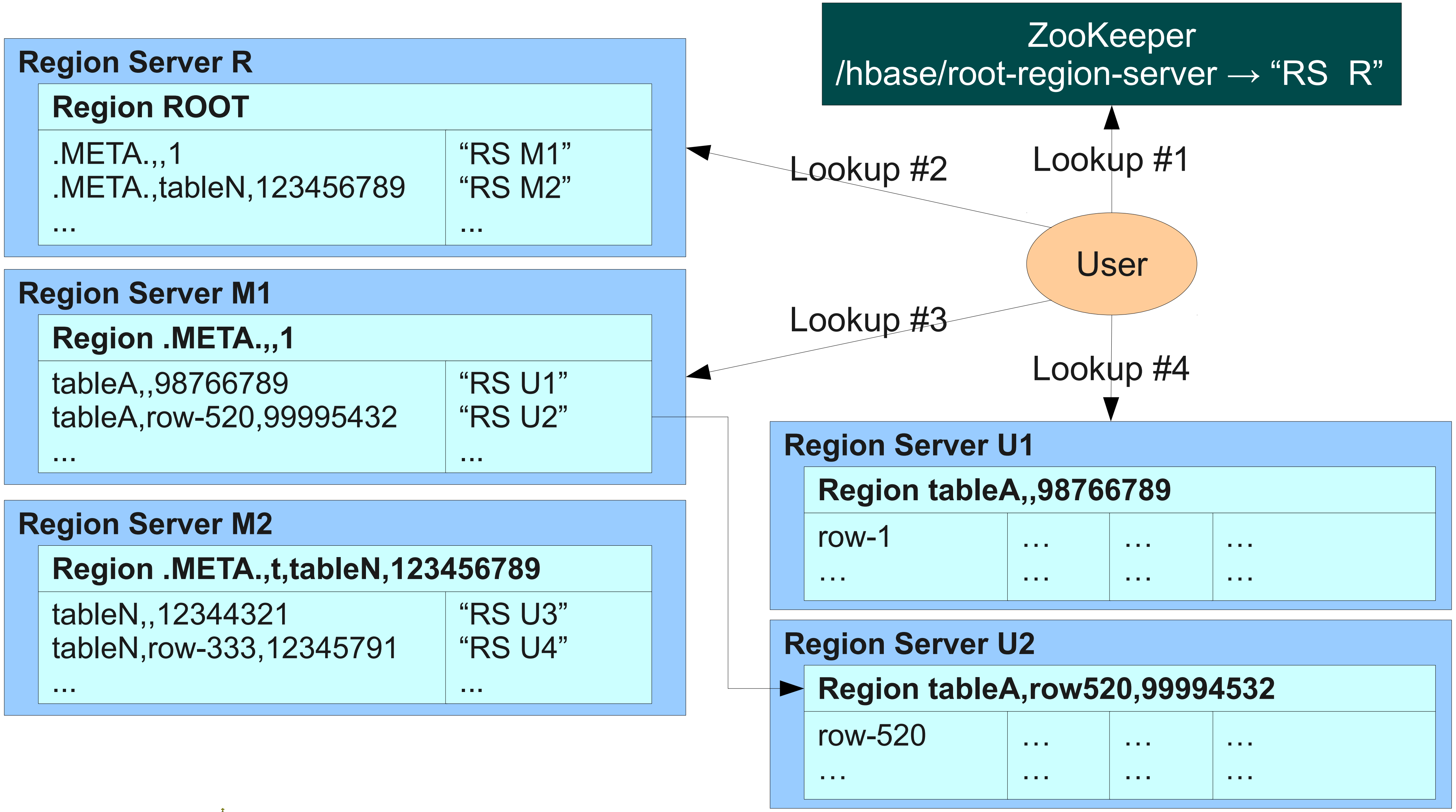
Writing, Updating, and Deleting Data

- Rows are written on the in-memory map of the corresponding RegionServer
- Update operations just write a new version of data
- Row deletion depends on where the row is located
  - Rows in the in-memory map are just deleted
  - HBase files are immutable! *Deletion markers* are used
- To prevent HBase files from using too much space they are merged
  - Rows marked as deleted and old versions of data are removed

HBase ROOT and .META Tables

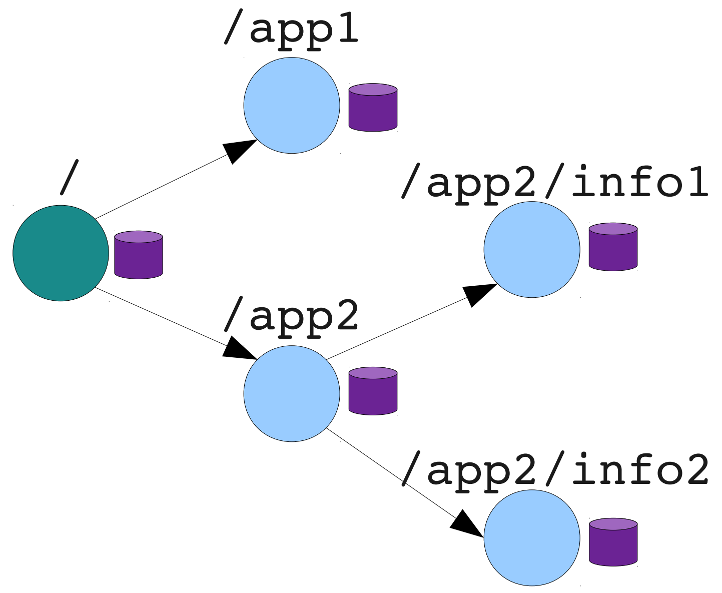
- Used to organize data in HBase
- Special system tables
- Clients use them to locate which RegionServer serves a certain key of a certain table
- They are partitioned into regions and served by RegionServers as normal tables

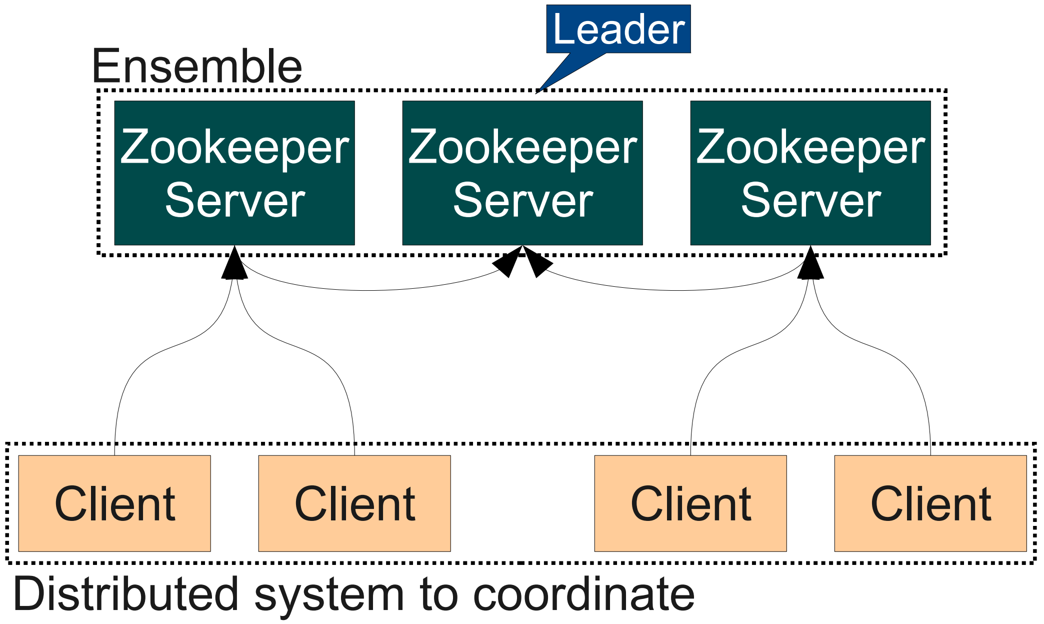
Example of Lookup for data in HBase



Zookeeper

- Zookeeper is used to coordinate members of distributed systems
- Implements a hierarchical namespace where clients write/read to share state information. Each element in the path can store data *and* other elements
- A typical Zookeeper deployment has several servers, grouped in an *ensemble*. The middleware takes care of consistency, load balancing, crash recovery...

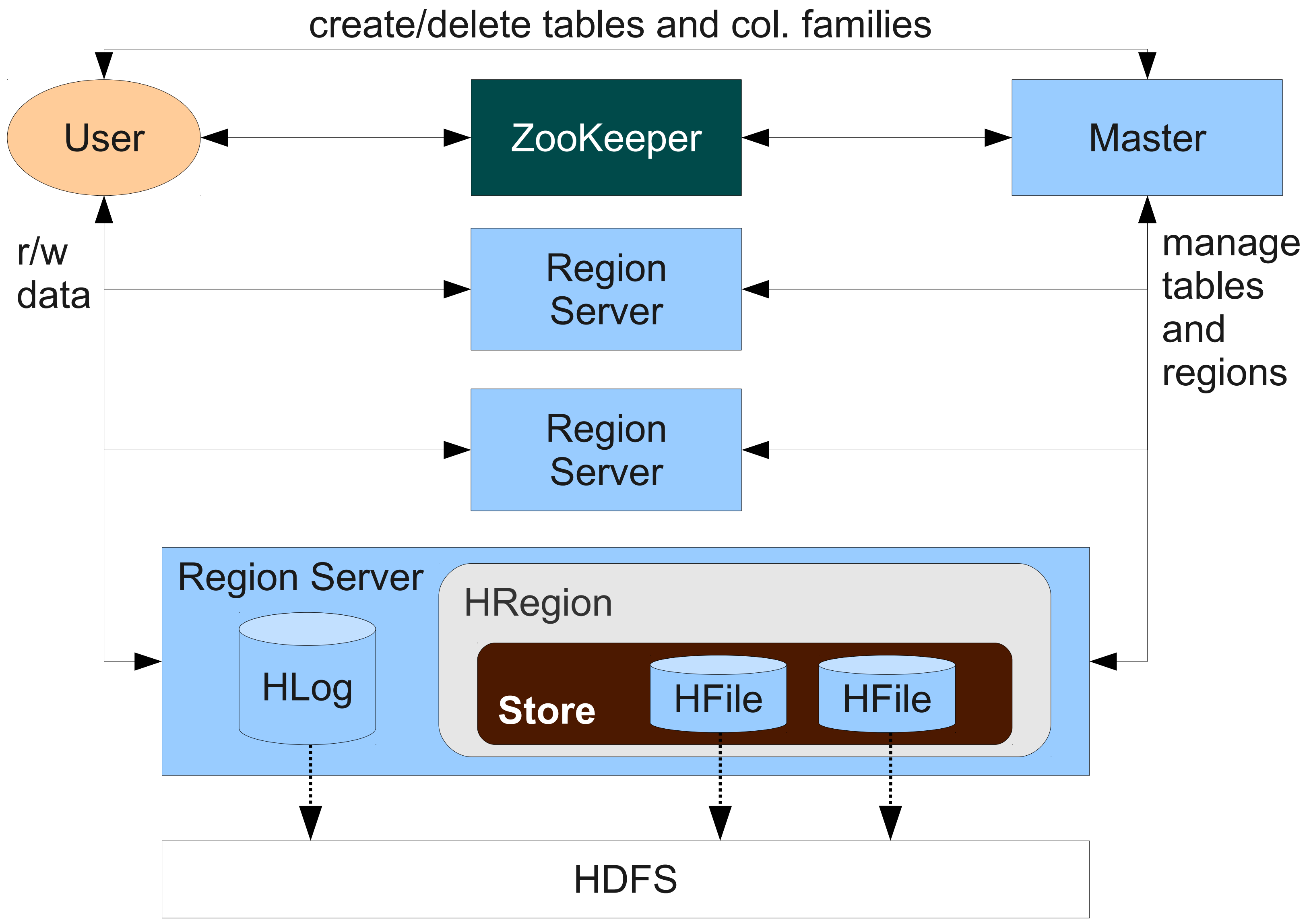




HMaster

- Splits the key space of all tables and assigns the resulting regions to the present RegionServers
- Balances load by re-assigning regions
- Handles metadata changes requests from clients
- It is **NOT** involved in read/write operations
- It uses Zookeeper to keep track of RegionServers and to emit information for clients (like which RS holds the ROOT table)

HBase Architecture



HBase Client API

- `HBaseAdmin` is used to:
  - Create/delete tables: `createTable(); deleteTable()`

```
HBaseAdmin hbAdm = new
                   HbaseAdmin(HBaseConfiguration.create());
hbAdm.createTable(new HTableDescriptor("TestTable"));
```

  - Add/delete column families to tables: `addColumn(); deleteColumn()`

```
hbAdm.addColumn("TestTable",
                new HcolumnDescriptor("TestColFamily"));
```

HBase Client API (2)

- `HTable` class is the basic data access entity:
  - Read data with `get()`; `getScanner(Scan)`

```
Get get = new Get(Bytes.toBytes(“testRow”));
Result result = testTable.get(get);
for(byte[] family: result.keySet())
  for(byte qual: result.get(family).keySet())
    for(Long ts: result.get(family).get(qual).keySet())
      String val = Bytes.toString(
                   Result.get(family).get(qual).get(ts));
```

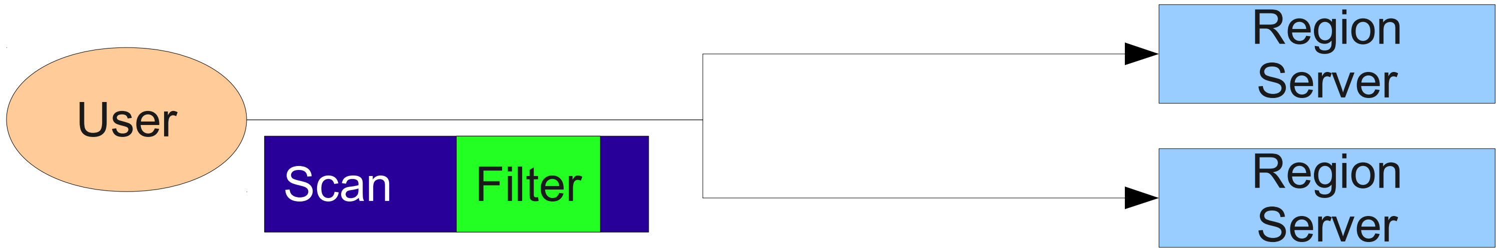
  - Write data with `put()`; `checkAndPut()`

```
Put put = new Put(Bytes.toBytes(“testRow”));
put.add(Bytes.toBytes(“testFam”),
        Bytes.toBytes(“testQual”), Bytes.toBytes(“value”));
testTable.put(put);
```

  - Delete data with `delete()`; `checkAndDelete()`

HBase Advanced Features

- *Filters*: When scanning tables, `Filter` instances refine the results returned to the client



- *Counters*: support for read-and-update atomic operations
- *Coprocessors*: to extend HBase functionality with users' custom code that it is run by the framework. Example: secondary indexes, access control...

Bibliography

- (Paper) "*Bigtable: A Distributed Storage System for Structured Data*". Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. ACM Transactions on Computer Systems, Volume 26 Issue 2, 2008.

- (Book) "HBase: The Definitive Guide" (2nd edition). Lars George. O'Reilly Media, Inc., 2011

(Example with HBase)