TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

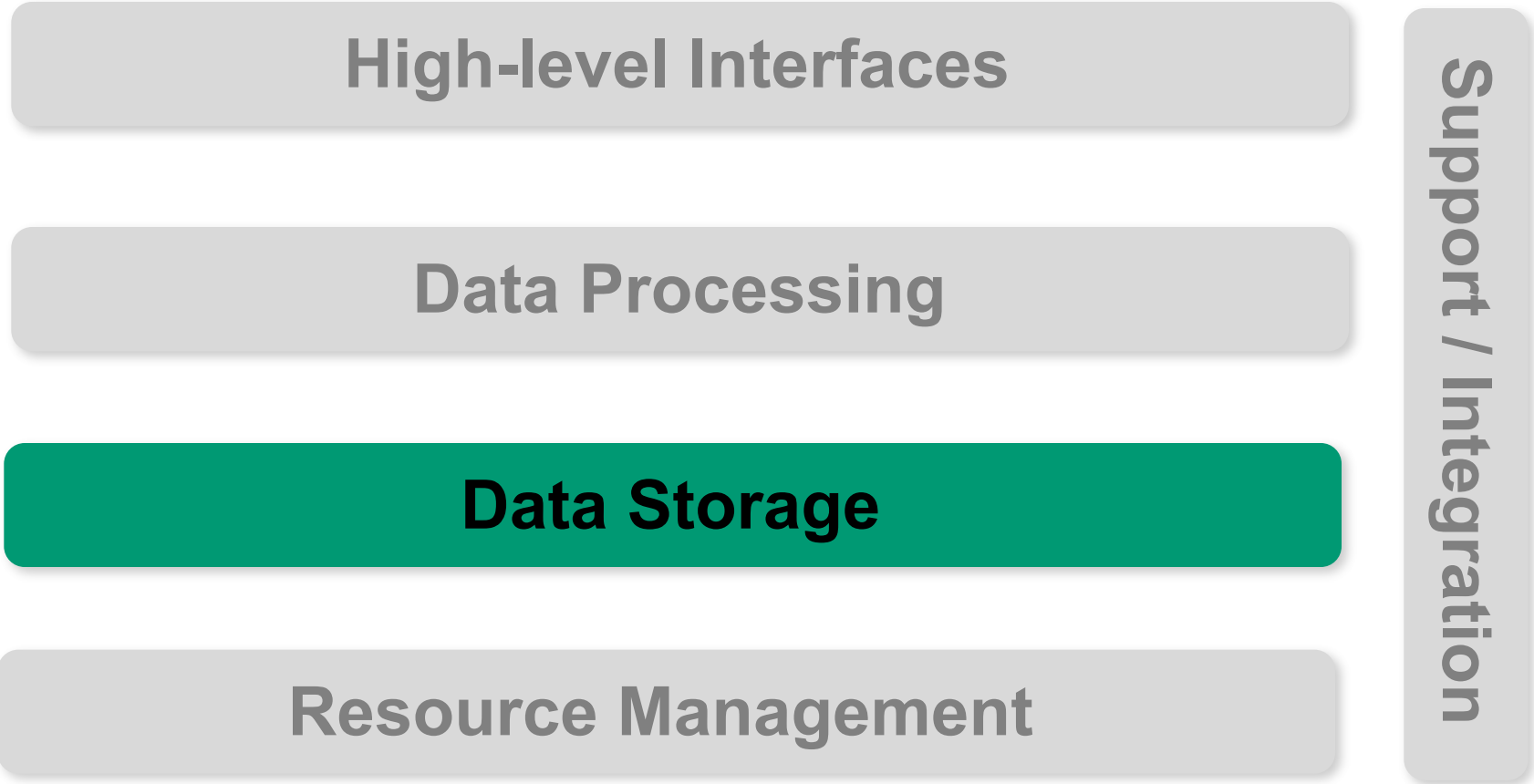
NoSQL: HBase

A.A. 2025/26

Matteo Nardelli

Laurea Magistrale in Ingegneria Informatica - II anno

The reference Big Data stack



Column-family data model

- Strongly aggregate-oriented
  - Lots of aggregates
  - Each aggregate has a key
- Similar to a key/value store, but the value can have multiple attributes (*columns*)
- Data model: a two-level map structure:
  - A set of <row-key, aggregate> pairs
  - Each aggregate is a group of pairs <column-key, value>
  - Column: a set of data values of a particular type
- Structure of the aggregate visible
- Columns can be organized in families
  - Data usually accessed together

HBase

- Apache HBase:
  - open-source implementation providing Bigtable-like capabilities on top of Hadoop and HDFS
  - CP system (in the CAP space)
- Data Model
  - HBase is based on Google's Bigtable model
  - A table store rows, sorted in alphanumerical order
  - A row consists of a set of columns
  - Columns are grouped in column families
  - A table defines a priori its column families (but not the columns within the families)

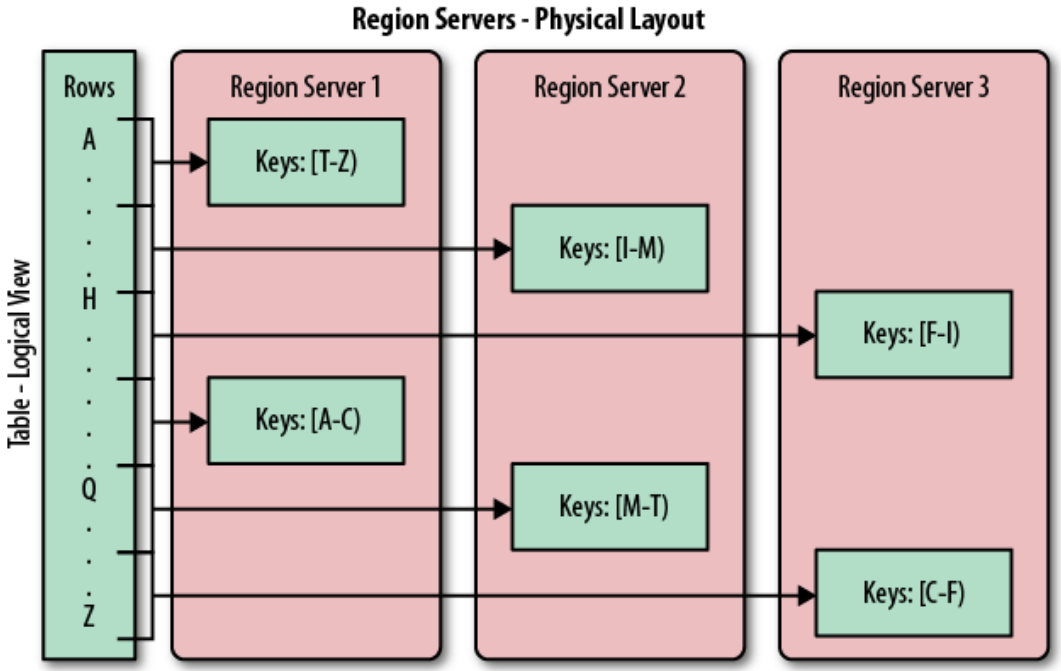
Row key	Column key	Timestamp	Cell value
cutting	info:state	1273516197868	IT
parser	role:Hadoop	1273616297466	g91m

(info and role are column families)

HBase: Auto-sharding

Region:

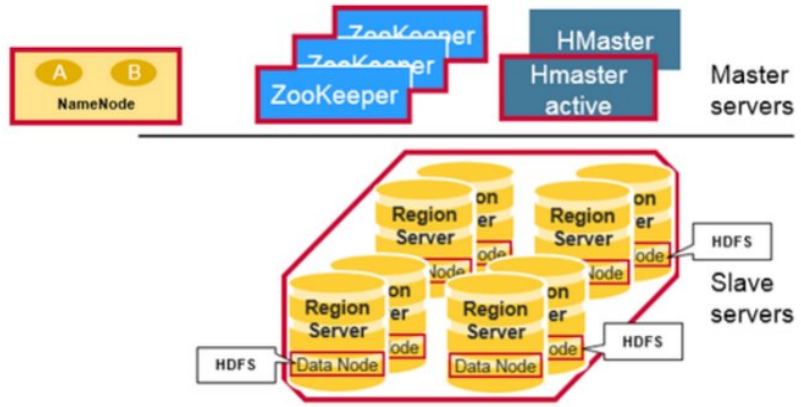
- the basic unit of scalability and load balancing
- similar to the tablet in Bigtable
- a contiguous range of rows stored together
- each region is served by exactly one region server
- they are dynamically split by the system when they become too large



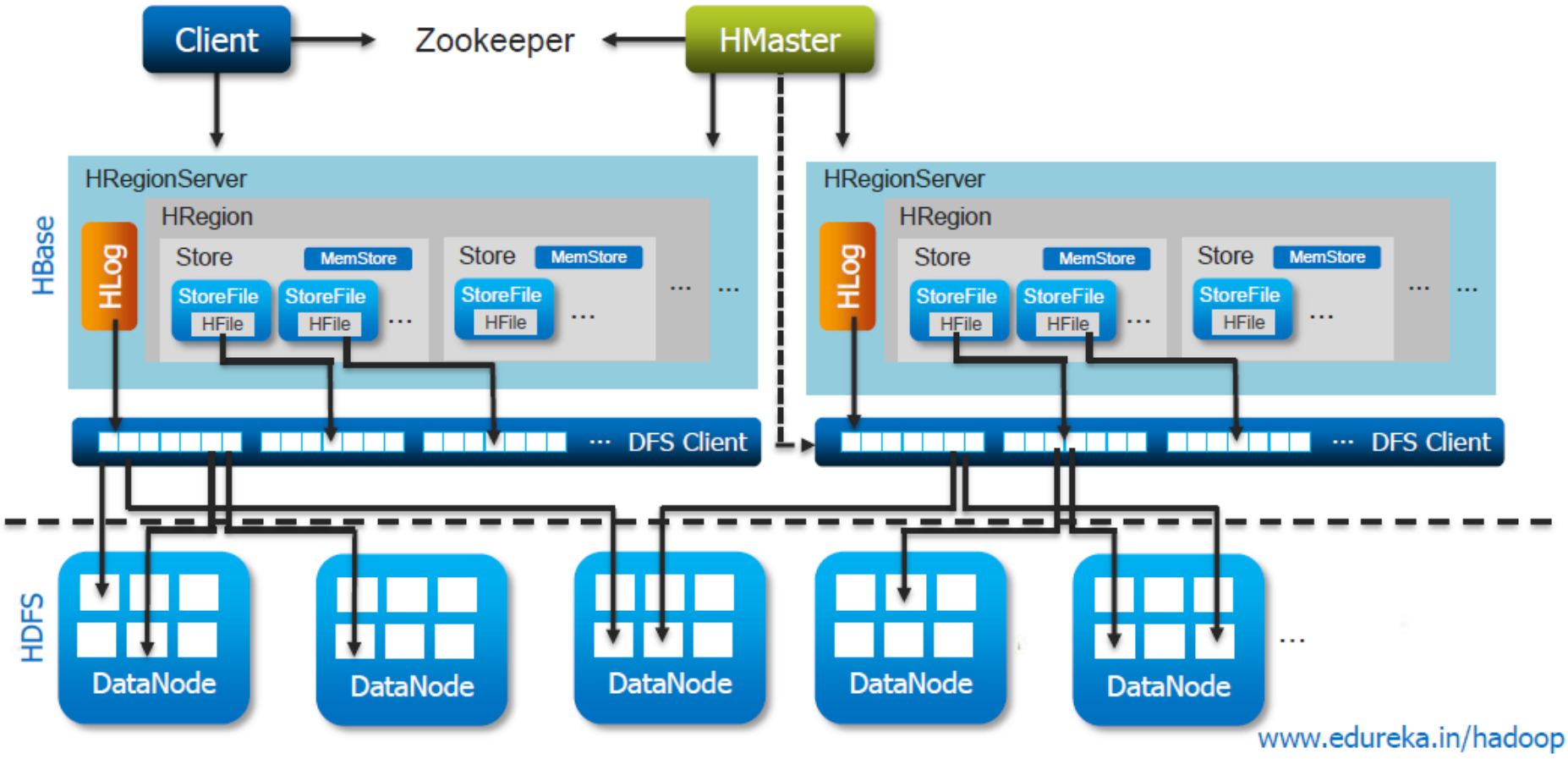
HBase: Architecture

Three major components:

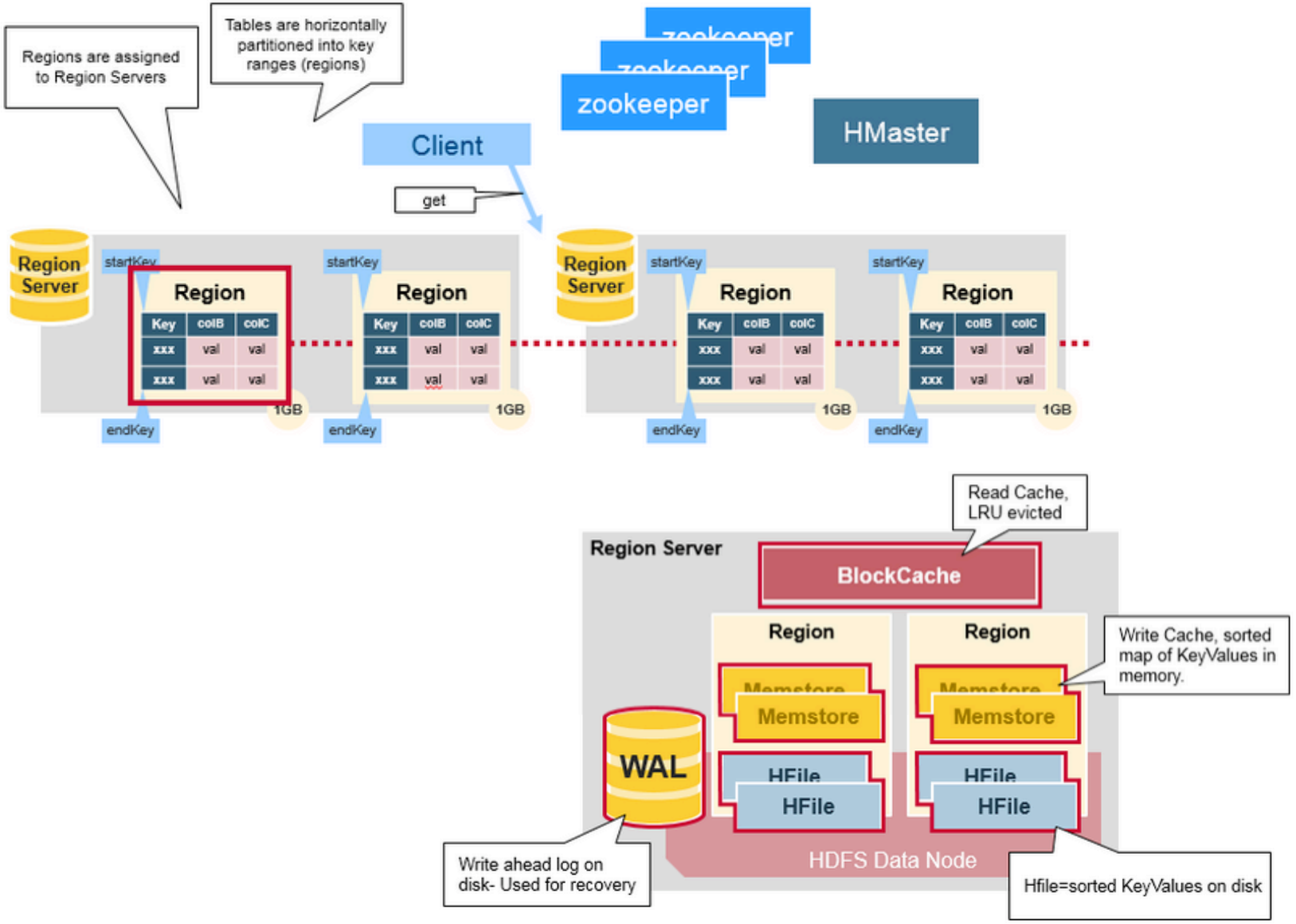
- the client library
- one master server
  - The master is responsible for assigning regions to region servers and uses Apache ZooKeeper to facilitate that task
- many region servers
  - manage the persistence of data
  - region servers can be added or removed while the system is up and running to accommodate changing workloads



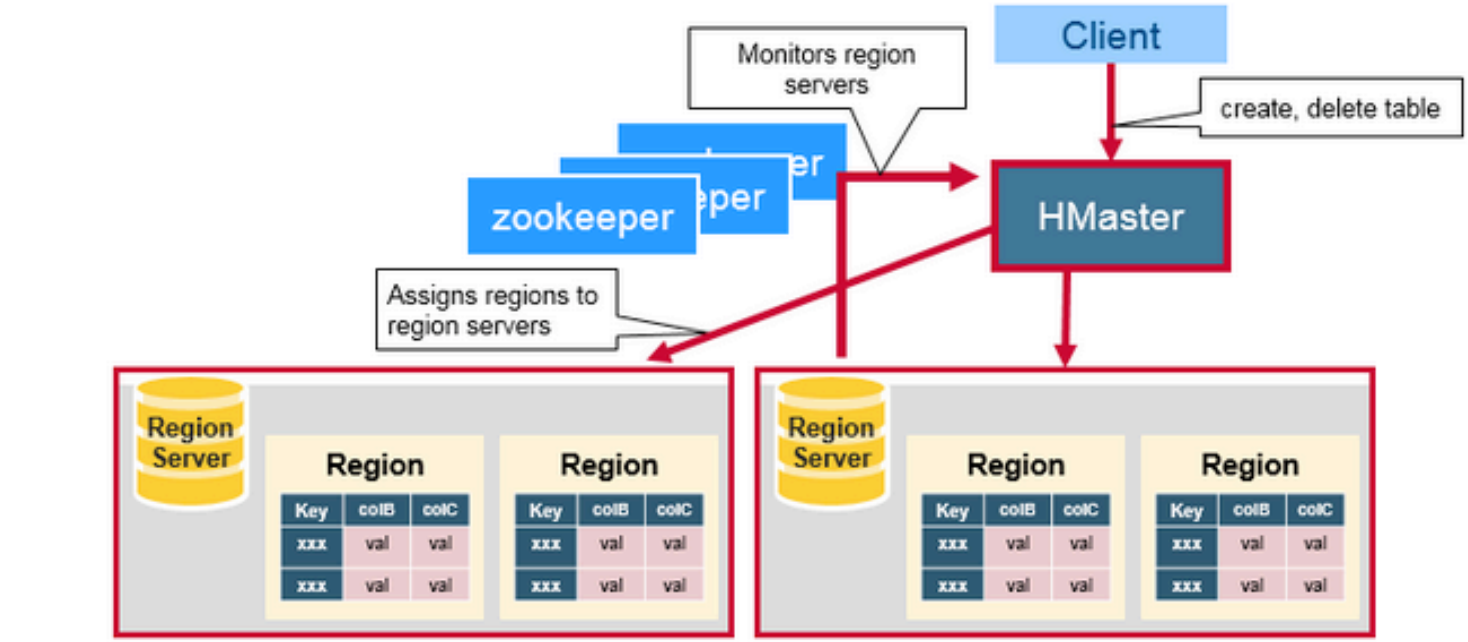
HBase: Architecture



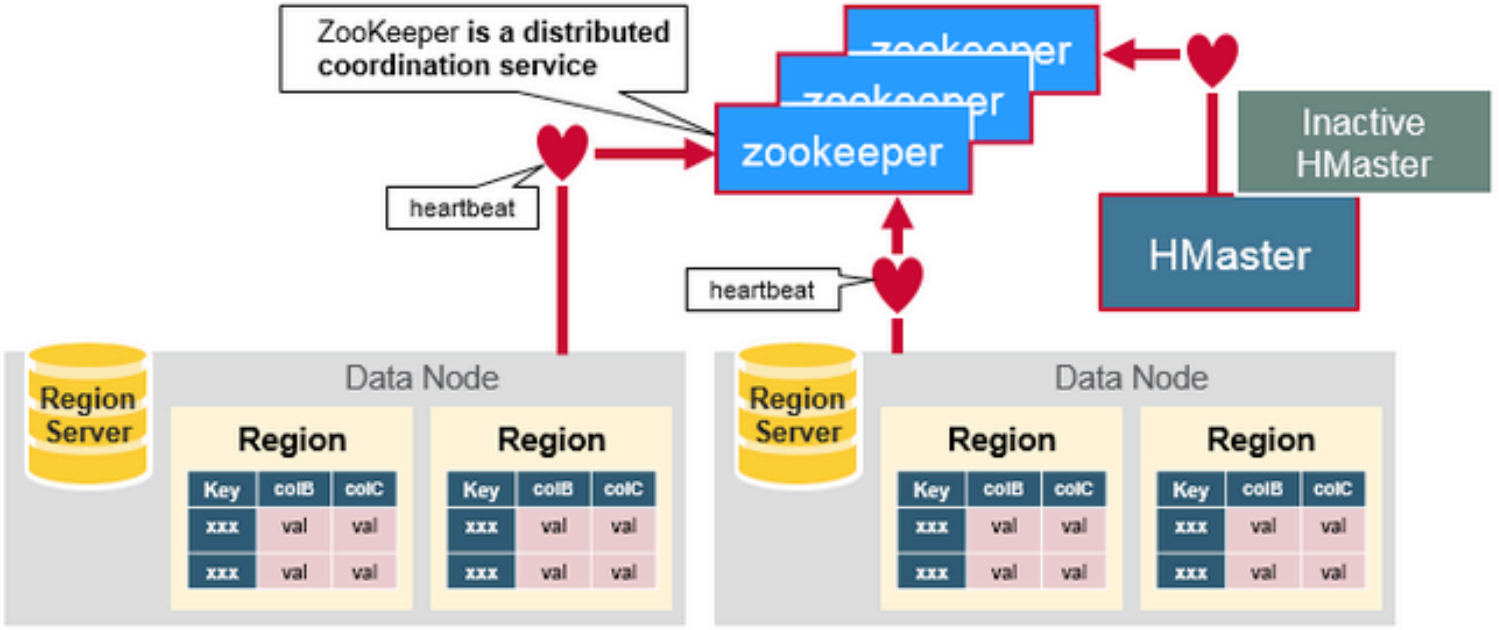
Regions


Regions are assigned to Region Servers
Tables are horizontally partitioned into key ranges (regions)
zookeeper
zookeeper
zookeeper
HMaster
Client
get
Region Server
startKey
Region
Key
colB
colC
xxx
val
endKey
1GB
Region Server
Read Cache, LRU evicted
BlockCache
Region
Memstore
Write Cache, sorted map of KeyValues in memory.
WAL
HFile
HDFS Data Node
Write ahead log on disk- Used for recovery
Hfile=sorted KeyValues on disk

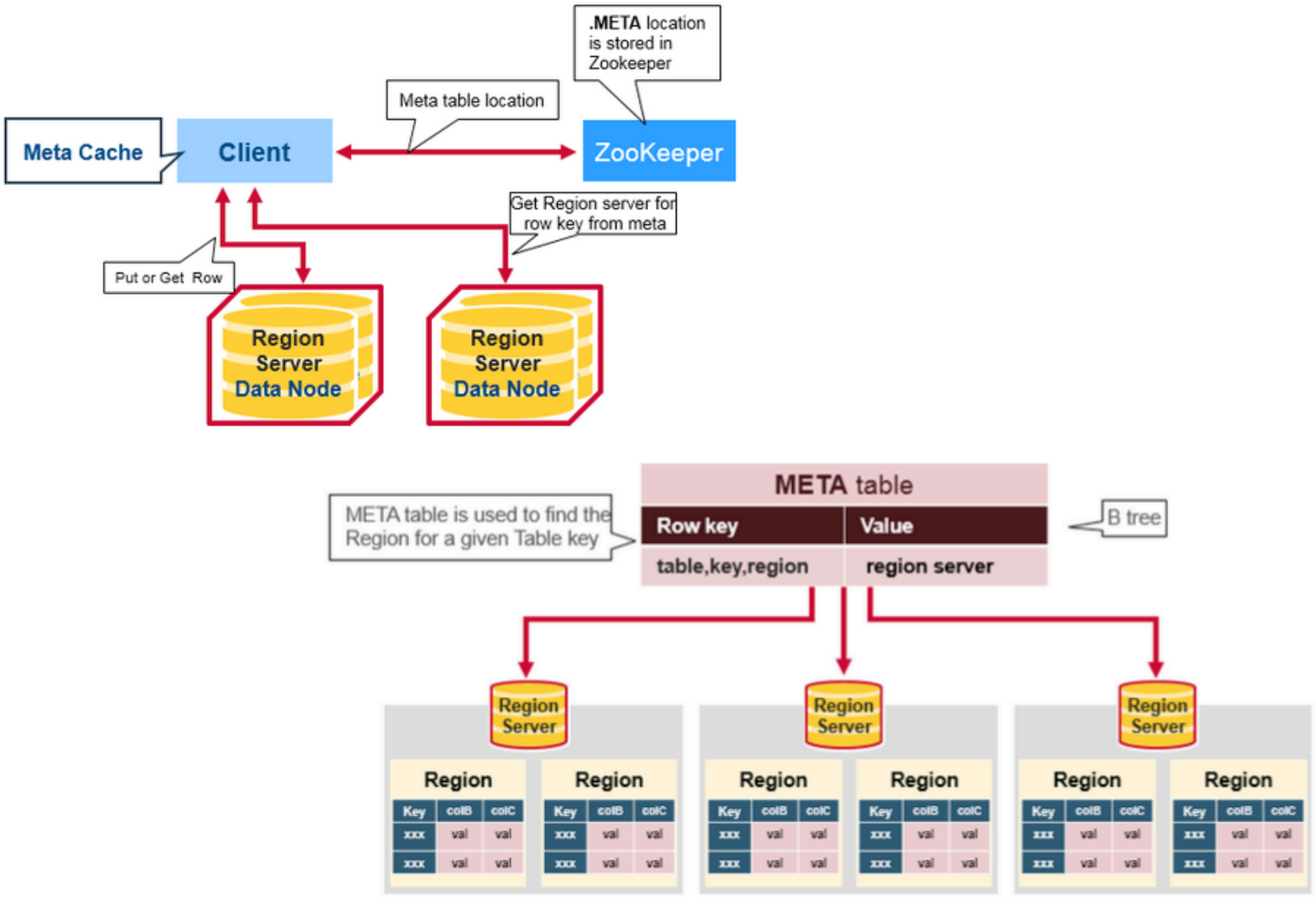
HBase HMaster



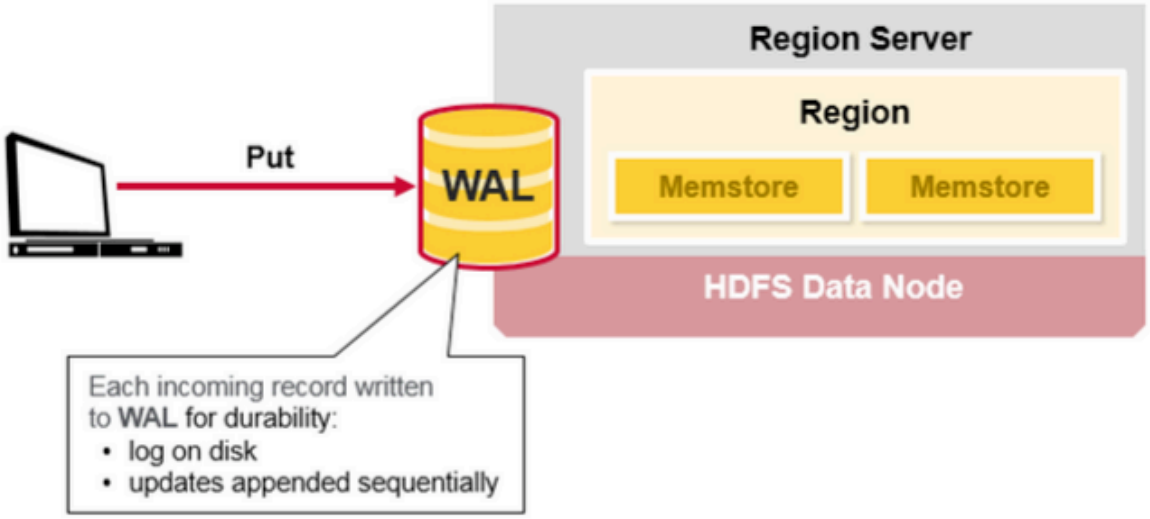
ZooKeeper: the Coordinator

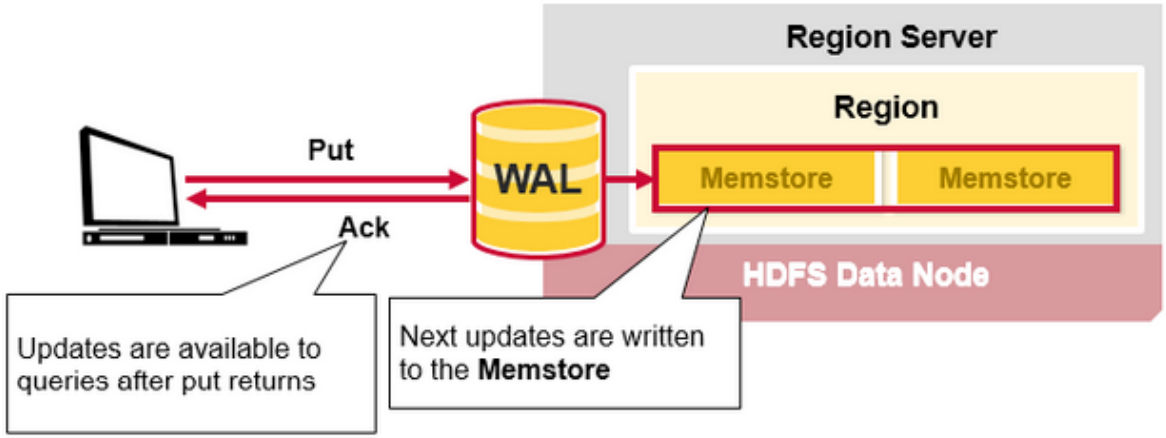


Meta Table Lookup

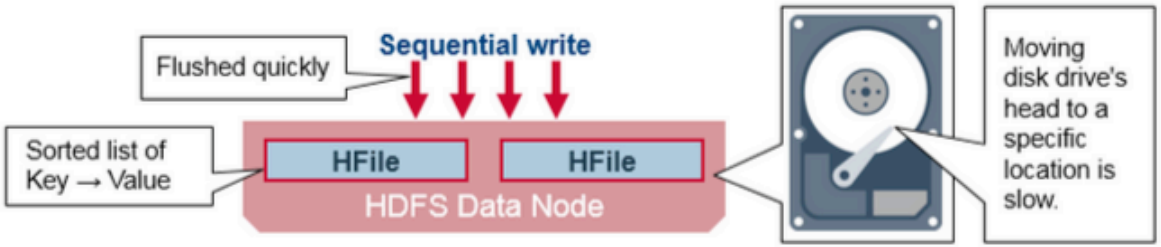


HBase Write Steps



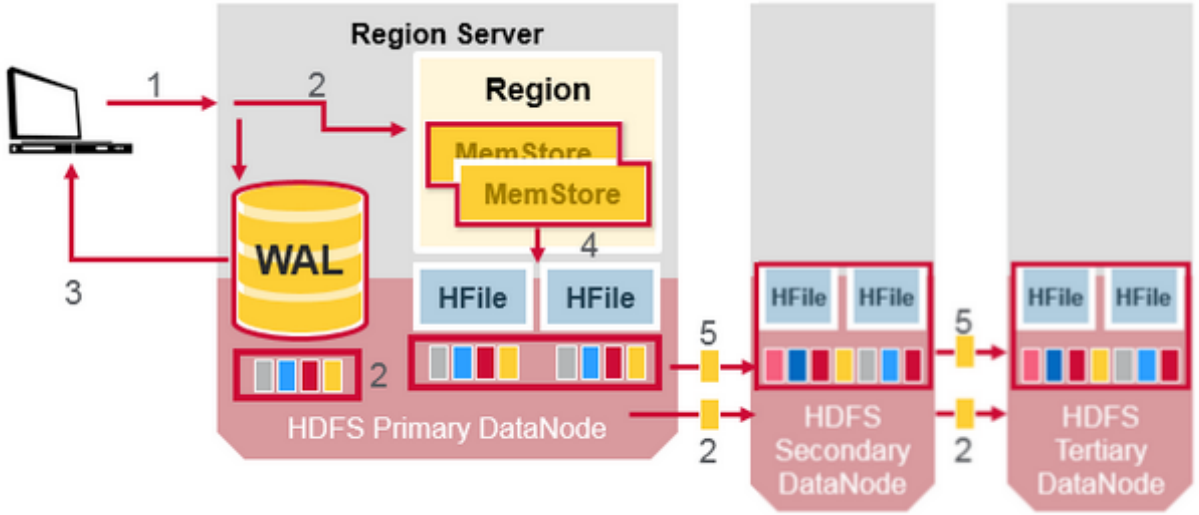


HBase HFiles



Key	CF1:Col	version	value
ra	cf1:ca	v1	1
rb	cf1:cb	v2	4
rb	cf1:cb	v1	3
rc	cf1:ca	v1	5

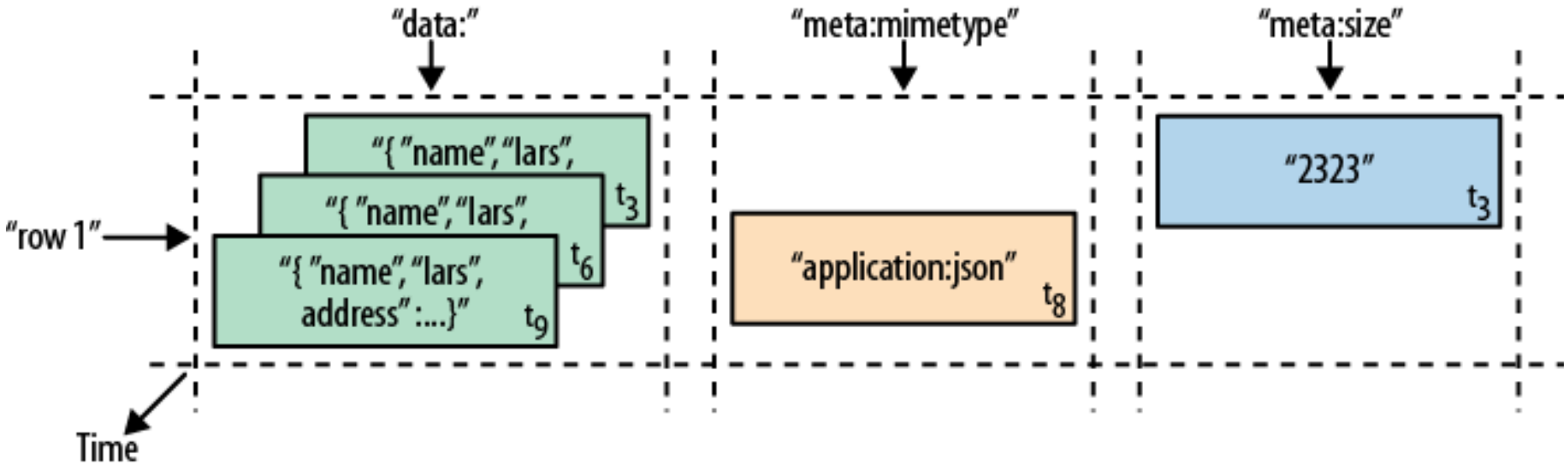
Key	CF2:Col	version	value
ra	cf2:ca	v1	2
rc	cf2:ca	v2	7
rc	cf2:ca	v1	6
rc	cf2:cd	v1	8



HBase: Versioning

- Cells may exist in multiple versions, and different columns have been written at different times.

By default, the API provides a coherent view of all columns wherein it automatically picks the most current value of each cell.



HBase: Strengths

- The column-oriented architecture allows for huge, wide, sparse tables as storing NULLs is free.
- Highly scalable due to the flexible schema and row-level atomicity
- Since a row is served by exactly one server, HBase is strongly consistent, and using its multi-versioning can help you to avoid edit conflicts
- The storage format is ideal for reading adjacent key/value pairs
- Table scans run in linear time and row key lookups or mutations are performed in logarithmic order
- Bigtable has been in use for a variety of different use cases from batch-oriented processing to real-time data-serving

Hands-on HBase
(Docker image)

HBase with Dockers

- We use a lightweight container with a standalone HBase

```
$ docker pull harisekhon/hbase:2.1
```

- We can now create an instance of HBase; since we are interesting to use it from our local machine, we need to forward several HBase ports and update the hosts file;

```
$ docker run -ti --name=hbase-docker -h hbase-docker -p 2181:2181 -p
8080:8080 -p 8085:8085 -p 9090:9090 -p 9095:9095 -p 16000:16000 -p
16010:16010 -p 16201:16201 -p 16301:16301 harisekhon/hbase:1.4
```

```
# append the following line to /etc/hosts
127.0.0.1          hbase-docker
```

HBase Client

- We interact with HBase through its Java APIs
- Using Maven, include the hbase-client dependency:

```
<dependency>
        <groupId>org.apache.hbase</groupId>
        <artifactId>hbase-client</artifactId>
        <version>2.5.8</version>
</dependency>
```

HBase Client

```
public Connection getConnection() throws ... {

    Configuration conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", ZOOKEEPER_HOST);
    conf.set("hbase.zookeeper.property.clientPort",

ZOOKEEPER_PORT);
    conf.set("hbase.master", HBASE_MASTER);

    /* Check configuration */
    HBaseAdmin.checkHBaseAvailable(conf);

    Connection connection =
                connectionFactory.createConnection(conf);
    return connection;

}
```

This is only an excerpt, check the HBaseClient.java file

HBase Client: Create Table

```
public void createTable(String table,
                                String... columnFamilies) {

        Admin admin = ...
        HTableDescriptor tableDescriptor = ... table ...

        for (String columnFamily : columnFamilies) {
                tableDescriptor.addFamily(columnFamily);
        }

        admin.createTable(tableDescriptor);

}
```

This is only an excerpt, check the HBaseClient.java file

HBase Client: Drop Table

```
public void dropTable(String table) {

        Admin admin = ...
        TableName tableName = ... table ...

    // To delete a table or change its settings,
        // you need to first disable the table
    admin.disableTable(tableName);

        admin.deleteTable(tableName);

}
```

This is only an excerpt, check the HBaseClient.java file

HBase Client: Put Data

```
public void put(String table, String rowKey,
                        String columnFamily,
                        String column, String value) {

    Table hTable =
                    getConnection().getTable( ... table ... );

    Put p = new Put(b(rowKey));
    p.addColumn(b(columnFamily), b(column), b(value));

    // Saving the put Instance to the HTable
    hTable.put(p);

    hTable.close();
}
```

This is only an excerpt, check the HBaseClient.java file

HBase Client: Get Data

```
public String get(String table, String rowKey,
                  String columnFamily,
                  String column) {

    Table hTable =
                getConnection().getTable( ... table ... );

    Get g = new Get(b(rowKey));
    g.addColumn(b(columnFamily), b(column));

    Result result = hTable.get(g);

    return Bytes.toString(result.getValue());

}
```

This is only an excerpt, check the HBaseClient.java file

HBase Client: Delete Data

```
public void delete(String table, String rowKey) {

    Table hTable =
                getConnection().getTable( ... table ... );

    Delete delete = new Delete(b(rowKey));

    // deleting the data
    hTable.delete(delete);

    // closing the HTable object
    hTable.close();

}
```

This is only an excerpt, check the HBaseClient.java file