COLUMN ORIENTED DB

RDBMS

- Relational databases generally strive toward *normalization*: making sure every piece of data is stored only once.
- Normalization marks their structural setup.
- If, for instance, you want to store data about a person and their hobbies, you can do so with two tables: one about the person and one about their hobbies.

RDBMS

- Additional table is necessary to link hobbies to persons because of their *many-to-many relationship*: a person can have multiple hobbies and a hobby can have many persons practicing it.

Name	Birthday	PERSONID
Jos The Boss	11-12-1985	1
Fritz von Braun	27-1-1978	2
Freddy Stark		3
Delphine Thewiseone	16-9-1986	4

Person info table: represents person specific information

PERSONID	HOBBYID
1	2
1	2
2	3
2	4
3	5
3	6
3	1

Person-Hobby linking table. This is necessary because of the many to many relationship between hobbies and persons.

HOBBYID	HobbyName	HobbyDescription
1	Archery	Shooting arrows from a bow
2	Conquering the world	looking for trouble with your neighboring countries
3	building things	also known as construction
4	surfing	catching waves on a plank
5	swordplay	fencing with swords
6	lollygagging	hanging around doing nothing

Hobby info table: represents hobby specific information

Relational databases strive towards **normalization** (making sure every piece of data is stored just once). Each table has unique identifiers (primary keys) that are used to model the relation between the entities (tables) hence "relational".

RDBMS

- Every time you look something up in a row-oriented database, every row is scanned, regardless of which columns you require.
- Let's say you only want a list of birthdays in September.
- The database will scan the table from top to bottom and left to right eventually returning the list of birthdays.
- Indexing the data on certain columns can significantly improve lookup speed, but indexing every column brings extra overhead and the database is still scanning all the columns.

ROWID	Name	Birthday	Hobbies
1	Jos The Boss	11-12-1985	archery, conquering the world
2	Fritz von Braun	27-1-1978	building things, surfing
3	Freddy Stark		swordplay, lollygagging, archery
4	Delphine Thewiseone	16-9-1986	

Row-oriented lookup: from top to bottom and for every entry all columns are taken in memory.

Key Lookup in Row Oriented DB

Indexes

Key	RowID
1	0001B008D23A671A
2	0001B008D23A671B
3	0001B008D23A671C
4	0001B008D23A671D
5	0001B008D23A671E

WHERE key=4

Elmer Fudd calls customer service

Phone	RowID
(207) 882-7323	0001B008D23A671D
(209) 375-6572	0001B008D23A671B
(212) 227-1810	0001B008D23A671C
(718) 938-3235	0001B008D23A671A
(978) 744-0991	0001B008D23A671E

WHERE phone='(207) 882-7323'

Indexes on high-cardinality columns make accessing a single row very fast

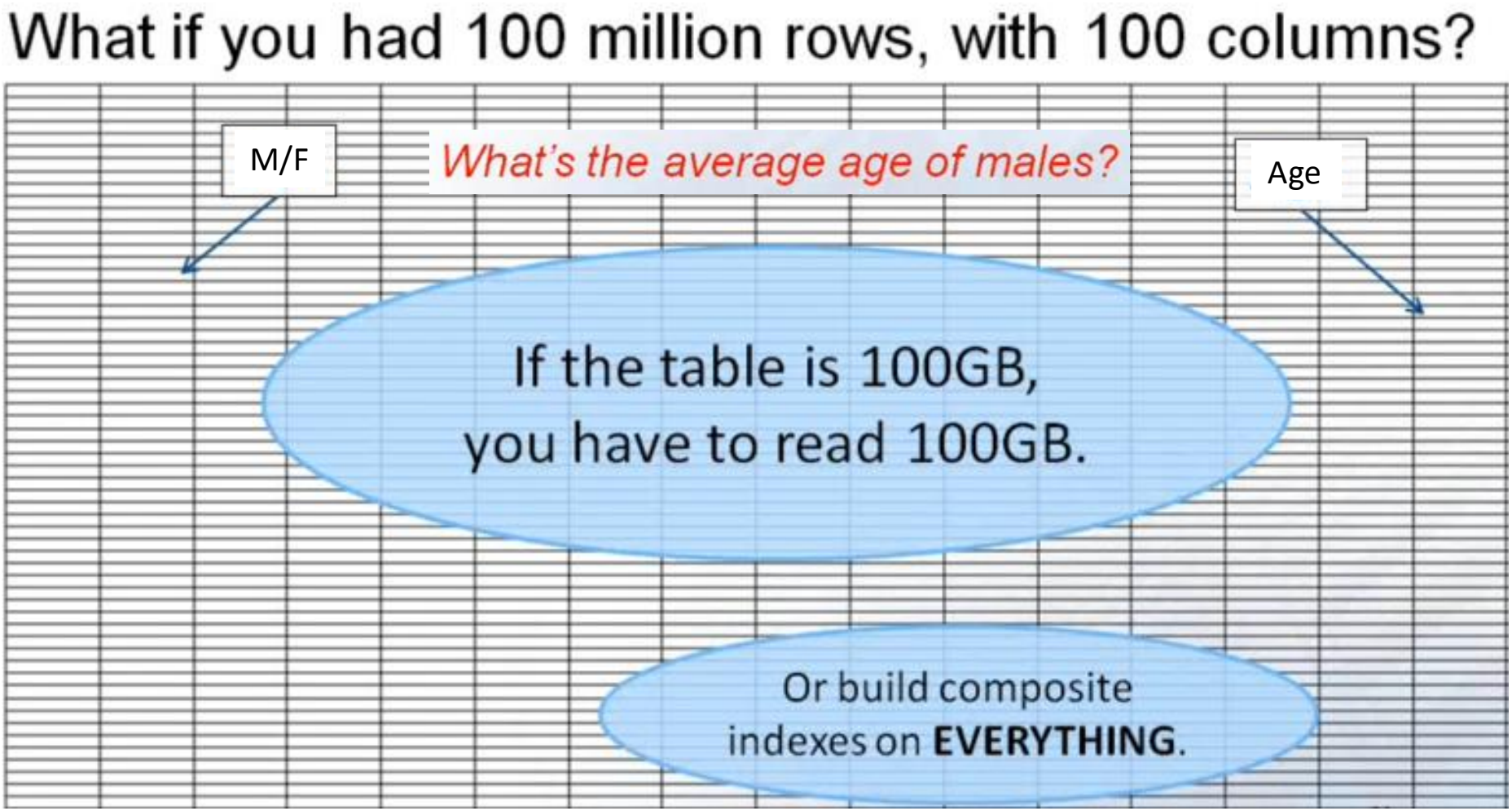
Key	Fname	Lname	State	Zip	Phone	Age	Sex
1	Bugs	Bunny	NY	11217	(718) 938-3235	34	M
2	Yosemite	Sam	CA	95389	(209) 375-6572	52	M
3	Daffy	Duck	NY	10013	(212) 227-1810	35	M
4	Elmer	Fudd	ME	04578	(207) 882-7323	43	M
5	Witch	Hazel	MA	01970	(978) 744-0991	57	F

but don't help on analytical queries scanning many rows

e.g.

What's the average age of males?

Column Oriented DB – IO Reduction



Column Oriented DB

- Column databases store each column separately, allowing for quicker scans when only a small number of columns are involved
- An index maps the row number to the data, whereas a column database maps the data to the row numbers; in that way counting becomes quicker, so it's easy to see how many people like archery, for instance.

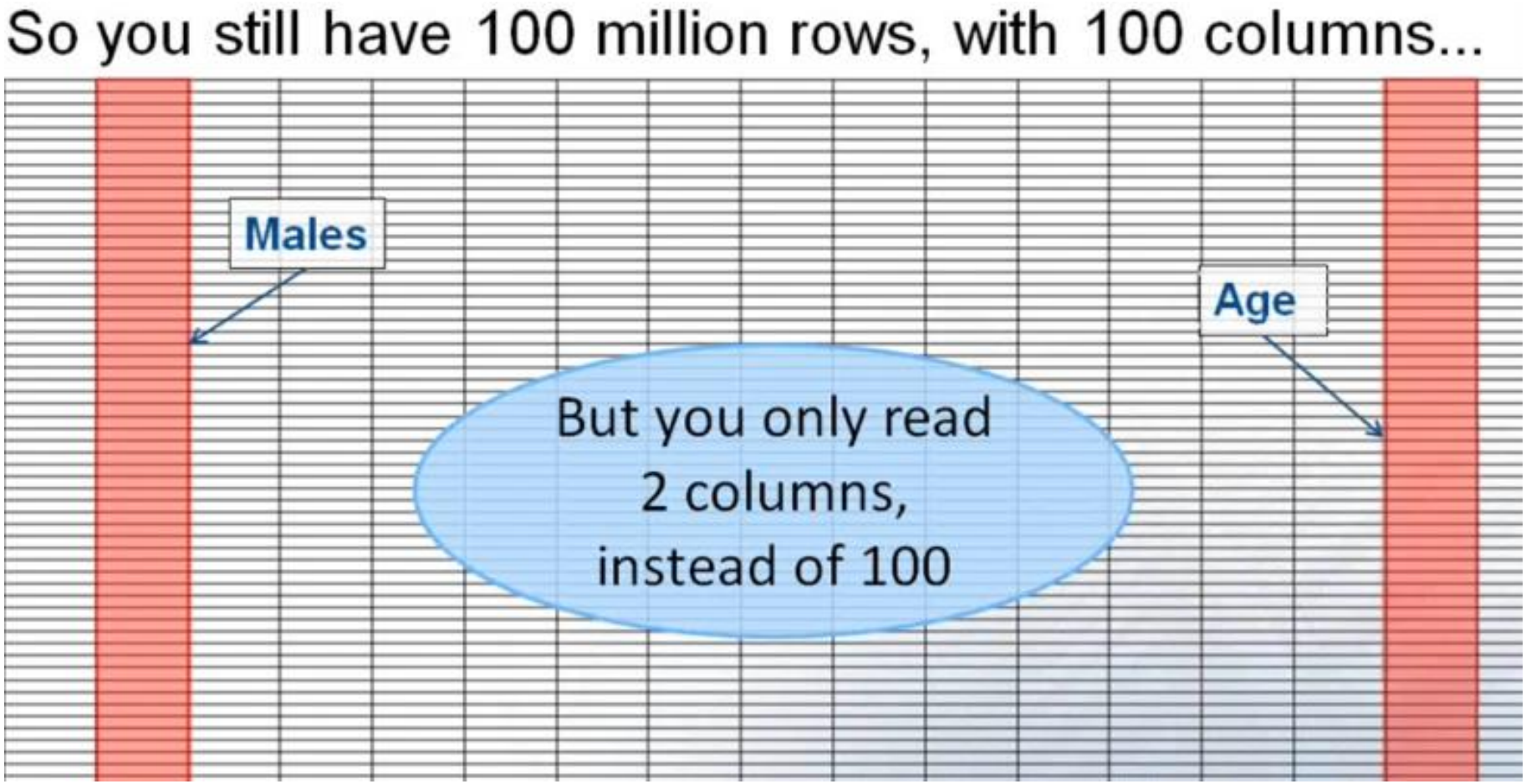
Name	ROWID
Jos The Boss	1
Fritz Schneider	2
Freddy Stark	3
Delphine Thewiseone	4

Birthday	ROWID
11-12-1985	1
27-1-1978	2
16-9-1986	4

Hobbies	ROWID
archery	1, 3
conquering the world	1
building things	2
surfing	2
swordplay	3
lollygagging	3

A column-oriented database stores each column separately

Column Oriented DB – IO Reduction



Column Oriented DB

1	Bugs	Bunny	Brooklyn	NY	11217	(718) 938-3235
2	Yosemite	Sam	Wawona	CA	95389	(209) 375-6572
3	Daffy	Duck	New York	NY	10013	(212) 227-1810
4	Elmer	Fudd	Wiscasset	ME	04578	(207) 882-7323
⋮	⋮	⋮	⋮	⋮	⋮	⋮
8m	Snoopy	Brown	Springfield	MA	01105	(413) 781-6500

- Each Column may be stored in a separate file, Only read the files you need
- Each column for a given row is at the same offset
- Improved compression, because column data may be of same type
- With the mapping known, partition elimination at query time

Column Oriented DB

- When should you use a row-oriented database and when should you use a column-oriented database?
- In a column-oriented database it's easy to add another column because none of the existing columns are affected by it.
- But adding an entire record requires adapting all tables.
- This makes the row-oriented database preferable over the column-oriented database for online transaction processing (OLTP) because this implies adding or changing records constantly.

Column Oriented DB

Row Oriented : Rebuilding of full table may be needed

Key	Fname	Lname	State	Zip	Phone	Age	Golf
1	Bugs	Bunny	NY	11217	(718) 938-3235	34	Y
2	Yosemite	Sam	CA	95389	(209) 375-6572	52	N
3	Daffy	Duck	NY	10013	(212) 227-1810	35	Y
4	Elmer	Fudd	ME	04578	(207) 882-7323	43	Y
5	Witch	Hazel	MA	01970	(978) 744-0991	57	N

Column Oriented : Just Create Another File

Key
1
2
3
4
5

Fname
Bugs
Yosemite
Daffy
Elmer
Witch

Lname
Bunny
Sam
Duck
Fudd
Hazel

State
NY
CA
NY
ME
MA

Zip
11217
95389
10013
04578
01970

Phone
(718) 938-3235
(209) 375-6572
(212) 227-1810
(207) 882-7323
(978) 744-0991

Age
34
52
35
43
57

Golf
Y
N
Y
Y
N

Column Oriented DB

Row Oriented: Insert new rows addition easy

Key	Fname	Lname	State	Zip	Phone	Age
1	Bugs	Bunny	NY	11217	(718) 938-3235	34
2	Yosemite	Sam	CA	95389	(209) 375-6572	52
3	Daffy	Duck	NY	10013	(212) 227-1810	35
4	Elmer	Fudd	ME	04578	(207) 882-7323	43
5	Witch	Hazel	MA	01970	(978) 744-0991	57
6	Marvin	Martian	CA	91602	(818) 761-9964	26

Column Oriented: New value goes to each file

Key
1
2
3
4
5
6

Fname
Bugs
Yosemite
Daffy
Elmer
Witch
Marvin

Lname
Bunny
Sam
Duck
Fudd
Hazel
Martian

State
NY
CA
NY
ME
MA
CA

Zip
11217
95389
10013
04578
01970
91602

Phone
(718) 938-3235
(209) 375-6572
(212) 227-1810
(207) 882-7323
(978) 744-0991
(818) 761-9964

Age
34
52
35
43
57
26

Column Oriented DB

Row Oriented: Updates easy

Key	Fname	Lname	State	Zip	Phone	Age
1	Bugs	Bunny	NY	11217	(718) 852-2352	34
2	Yosemite	Sam	CA	95389	(209) 375-6572	52
3	Daffy	Duck	NY	10013	(212) 227-1810	35
4	Elmer	Fudd	ME	04578	(207) 882-7323	43
5	Witch	Hazel	MA	01970	(978) 744-0991	57

Column Oriented: Updates easy

Key
1
2
3
4
5

Fname
Bugs
Yosemite
Daffy
Elmer
Witch

Lname
Bunny
Sam
Duck
Fudd
Hazel

State
NY
CA
NY
ME
MA

Zip
11217
95389
10013
04578
01970

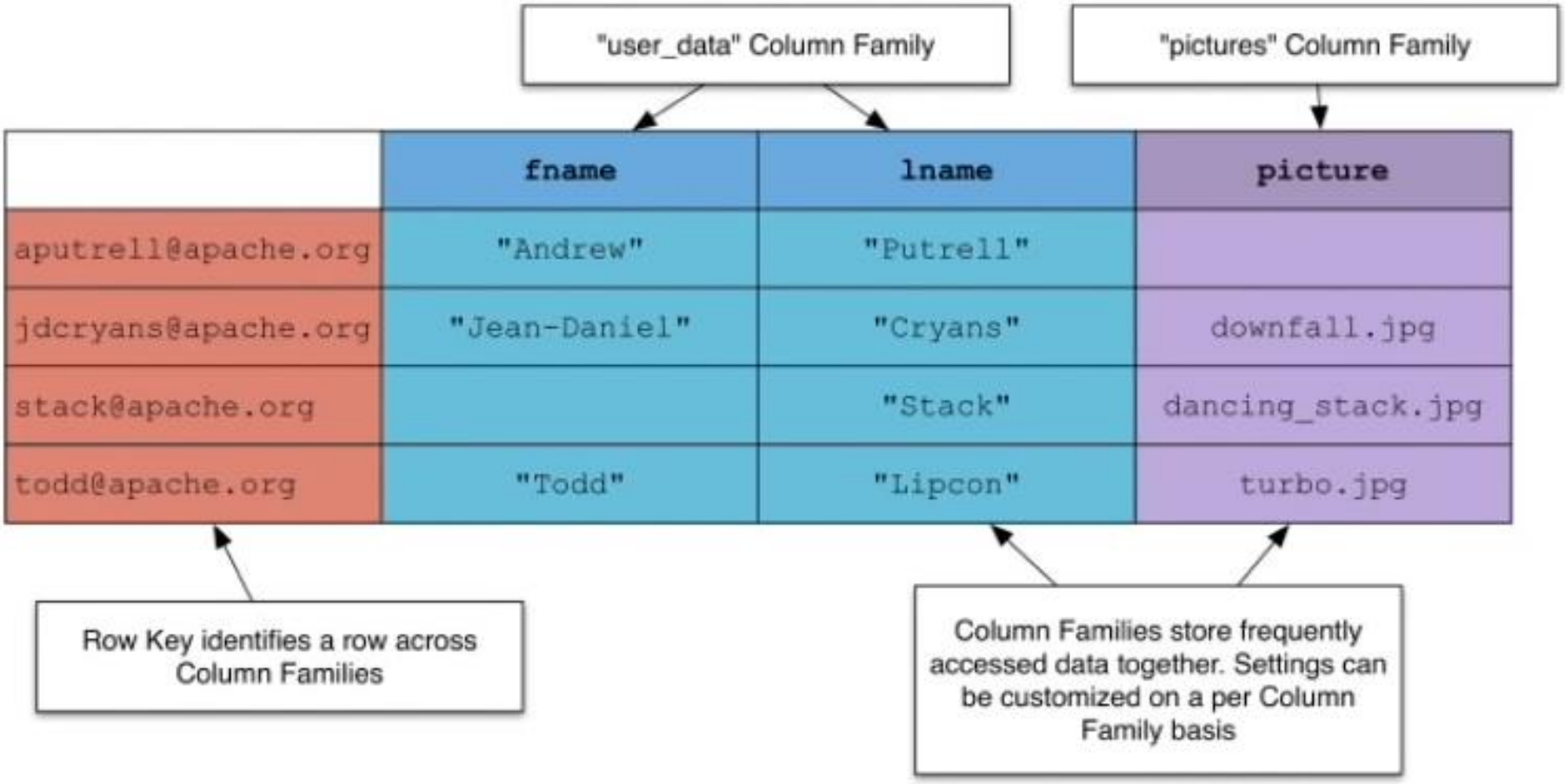
Phone
(718) 852-2352
(209) 375-6572
(212) 227-1810
(207) 882-7323
(978) 744-0991

Age
34
52
35
43
57

Column Oriented DB

- The column-oriented database shines when performing analytics and reporting: summating values and counting entries (OLAP).
- A row-oriented database is often the operational database of choice for actual transactions (such as sales, OLTP).
- Overnight batch jobs bring the column-oriented database up to date, supporting lightning-speed lookups and aggregations
- Examples of column-family stores are Apache HBase, Facebook's Cassandra, Hypertable, and the grandfather of wide-column stores, Google BigTable.

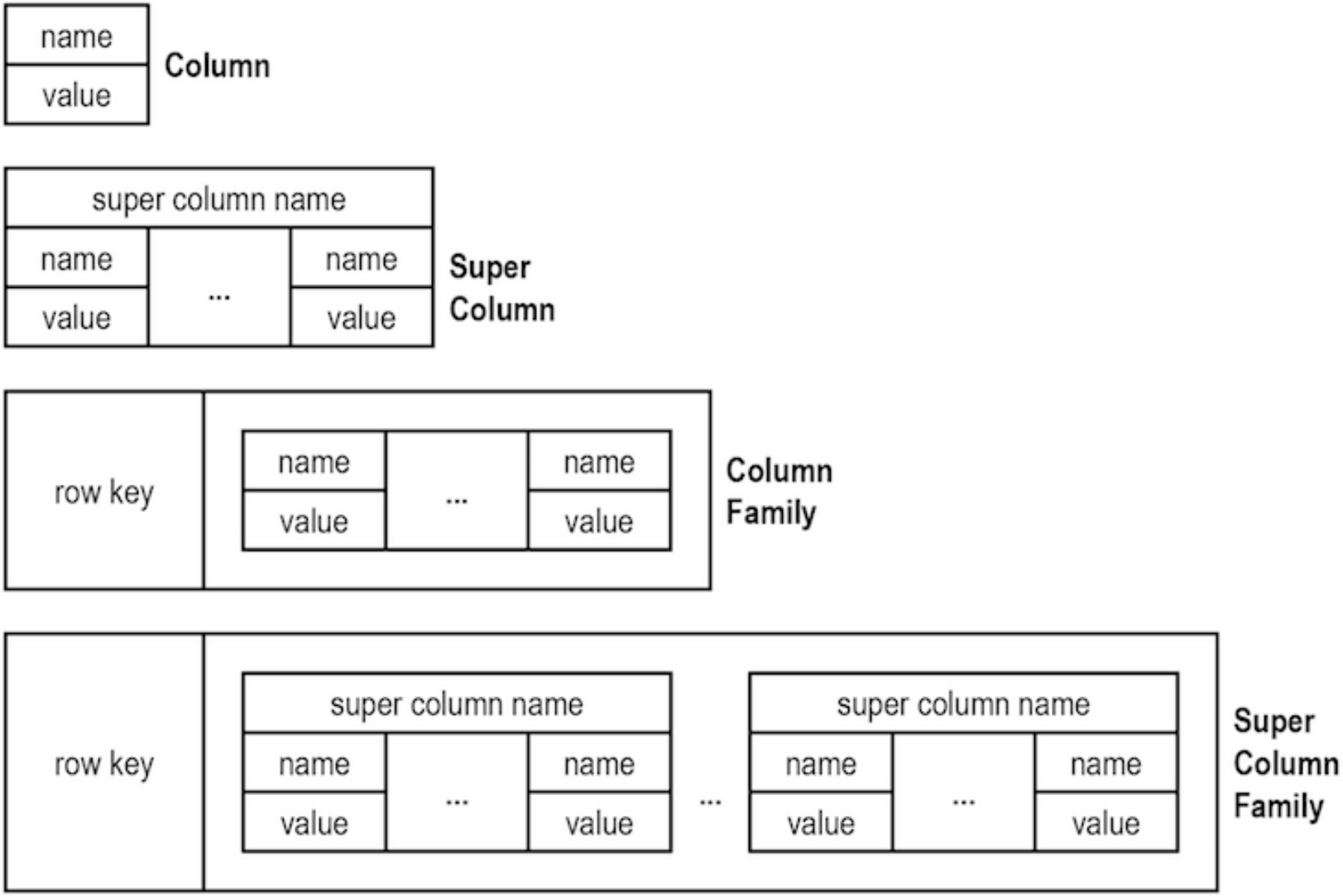
Column Oriented DB



Column Oriented DB

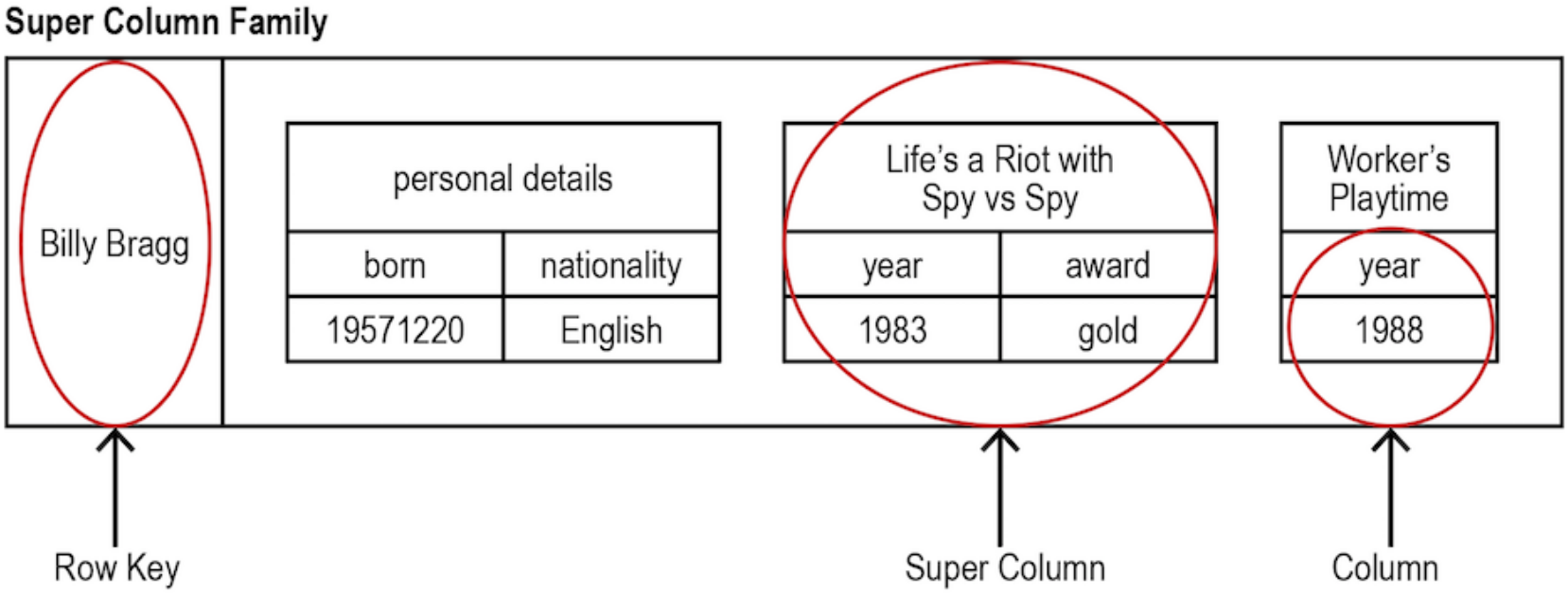
- Wide column stores are based on a sparsely populated table whose rows can contain arbitrary columns and where keys provide for natural indexing.
- The simplest unit of storage is the *column* itself consisting of a name-value pair.
- Any number of columns can then be combined into a *super column*, which gives a name to a particular set of columns.
- Columns are stored in rows, and when a row contains columns only, it is known as a *column family*, but when a row contains super columns, it is known as a *super column family*.

Column Oriented DB



Column Oriented DB

- Super column family mapping out a recording artist and his albums.
- Each row in the table represents everything about an artist.
- These column families are containers for related pieces of data, such as the artist's name and discography.
- Within the column families we find actual key-value data, such as album release dates and the artist's date of birth.



Column-Based or Wide Column NOSQL Systems

- BigTable: Google's distributed storage system for big data
  - Used in Gmail
  - Uses Google File System for data storage and distribution
- Apache HBase a similar, open source system
  - Uses Hadoop Distributed File System (HDFS) for data storage
  - Can also use Amazon's Simple Storage System (S3)

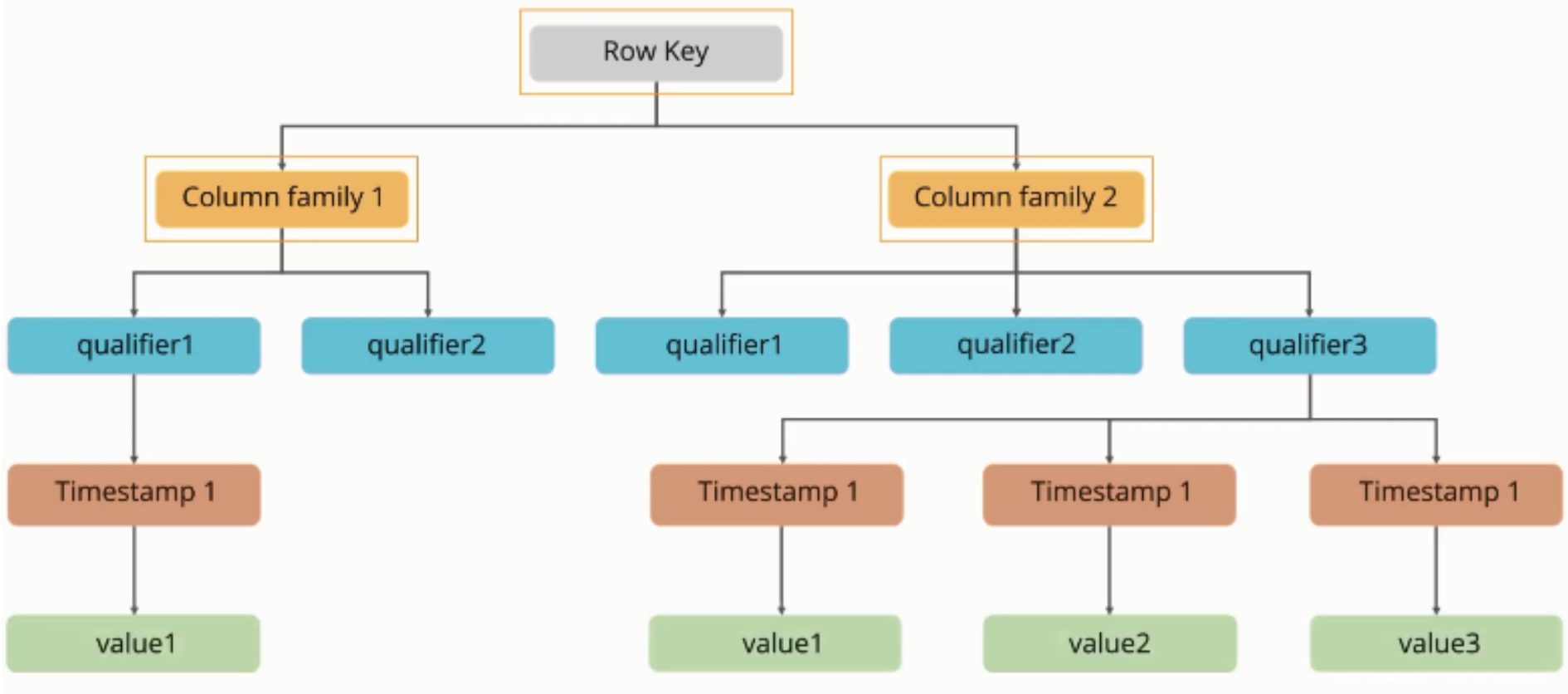
HBase Data Model and Versioning

- Data organization concepts
  - Namespaces
  - Tables
  - Column families
  - Column qualifiers
  - Columns
  - Rows
  - Data cells
- Data is self-describing

HBase Data Model and Versioning (cont'd.)

- Namespace is collection of tables
- Table associated with one or more column families
- Each row in a table has a unique row key
- Column qualifiers can be dynamically specified as new table rows are created and inserted
- Different rows can have different self-describing column qualifiers
- Cell holds a basic data item
- HBase stores multiple versions of data items
  - Timestamp associated with each version

Column Oriented DB



(a) creating a table:

```
create 'EMPLOYEE', 'Name', 'Address', 'Details'
```

(b) inserting some row data in the EMPLOYEE table:

```
put 'EMPLOYEE', 'row1', 'Name:Fname', 'John'
put 'EMPLOYEE', 'row1', 'Name:Lname', 'Smith'
put 'EMPLOYEE', 'row1', 'Name:Nickname', 'Johnny'
put 'EMPLOYEE', 'row1', 'Details:Job', 'Engineer'
put 'EMPLOYEE', 'row1', 'Details:Review', 'Good'
put 'EMPLOYEE', 'row2', 'Name:Fname', 'Alicia'
put 'EMPLOYEE', 'row2', 'Name:Lname', 'Zelaya'
put 'EMPLOYEE', 'row2', 'Name:MName', 'Jennifer'
put 'EMPLOYEE', 'row2', 'Details:Job', 'DBA'
put 'EMPLOYEE', 'row2', 'Details:Supervisor', 'James Borg'
put 'EMPLOYEE', 'row3', 'Name:Fname', 'James'
put 'EMPLOYEE', 'row3', 'Name:Minit', 'E'
put 'EMPLOYEE', 'row3', 'Name:Lname', 'Borg'
put 'EMPLOYEE', 'row3', 'Name:Suffix', 'Jr.'
put 'EMPLOYEE', 'row3', 'Details:Job', 'CEO'
put 'EMPLOYEE', 'row3', 'Details:Salary', '1,000,000'
```

(c) Some Hbase basic CRUD operations:

Creating a table: create <tablename>, <column family>, <column family>, …
Inserting Data: put <tablename>, <rowid>, <column family>:<column qualifier>, <value>
Reading Data (all data in a table): scan <tablename>
Retrieve Data (one item): get <tablename>,<rowid>

```
HBase> create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'}  HBase> #
The above in shorthand would be the following:  HBase> create 't1', 'f1', 'f2', 'f3'
```

```
HBase> describe 't1'
```

```
HBase> disable 't1'
```

```
HBase> drop 't1'
```

```
HBase> list
```

Figure: Examples in HBase (a) Creating a table called EMPLOYEE with three column families: Name, Address, and Details (b) Inserting some in the EMPLOYEE table; different rows can have different self-describing column qualifiers (Fname, Lname, Nickname, Mname, Minit, Suffix, … for column family Name; Job, Review, Supervisor, Salary for column family Details). (c) Some CRUD operations of Hbase

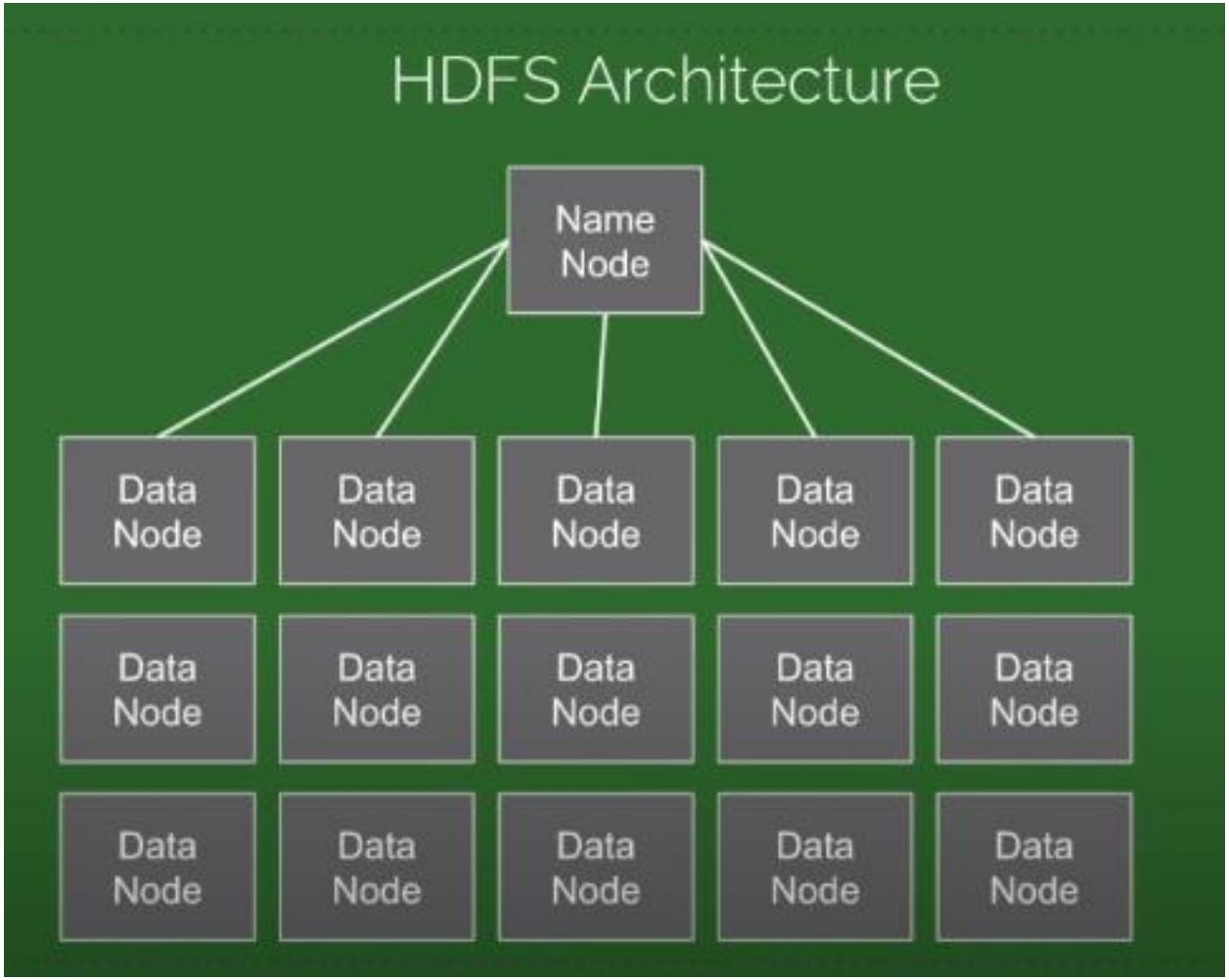
HBase CRUD Operations

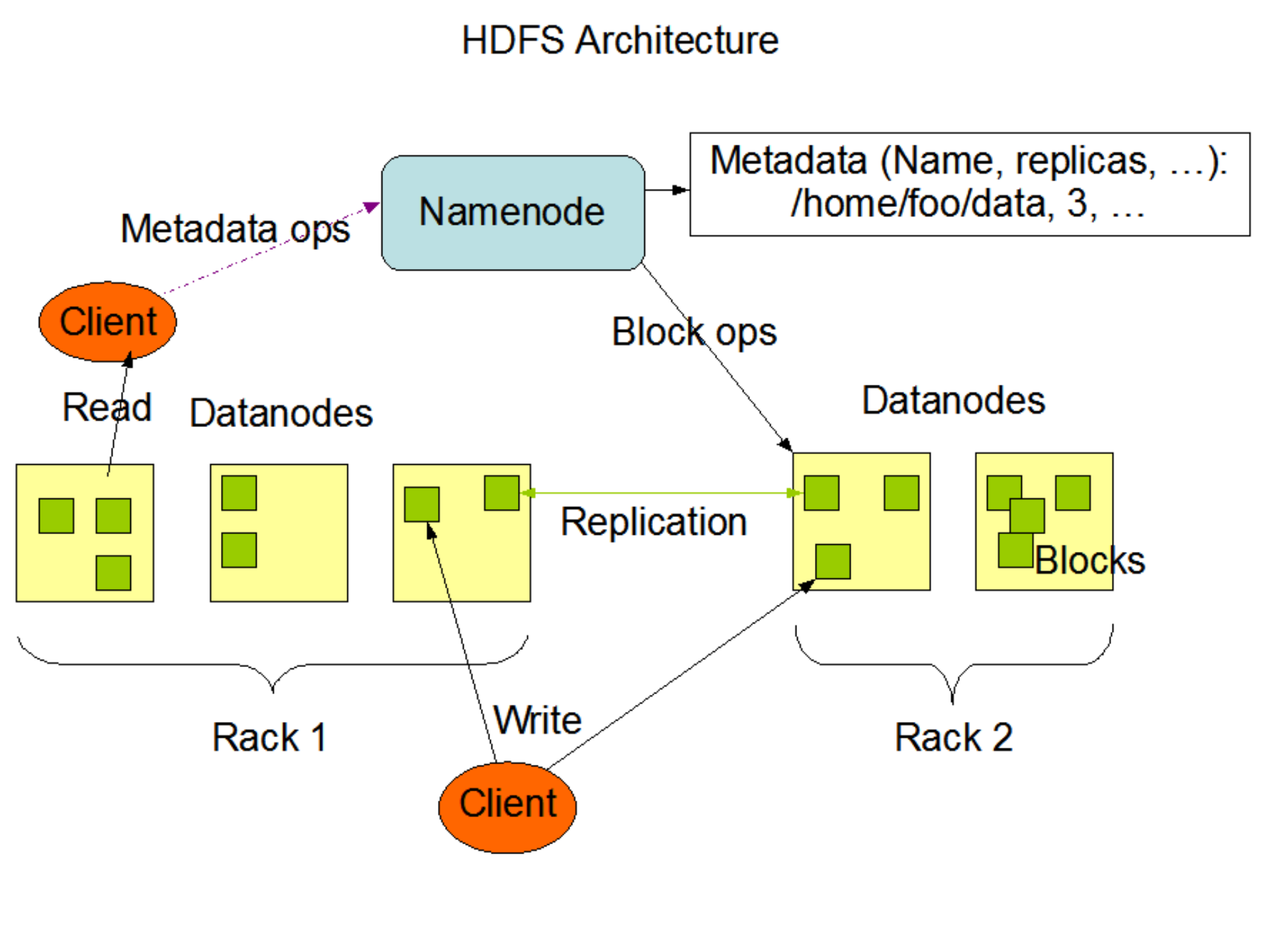
- Provides only low-level CRUD (create, read, update, delete) operations
- Application programs implement more complex operations
- **Create:** Creates a new table and specifies one or more column families associated with the table
- **Put:** Inserts new data or new versions of existing data items
- **Get:** Retrieves data associated with a single row
- **Scan:** Retrieves all the rows

HBase Storage and Distributed System Concepts

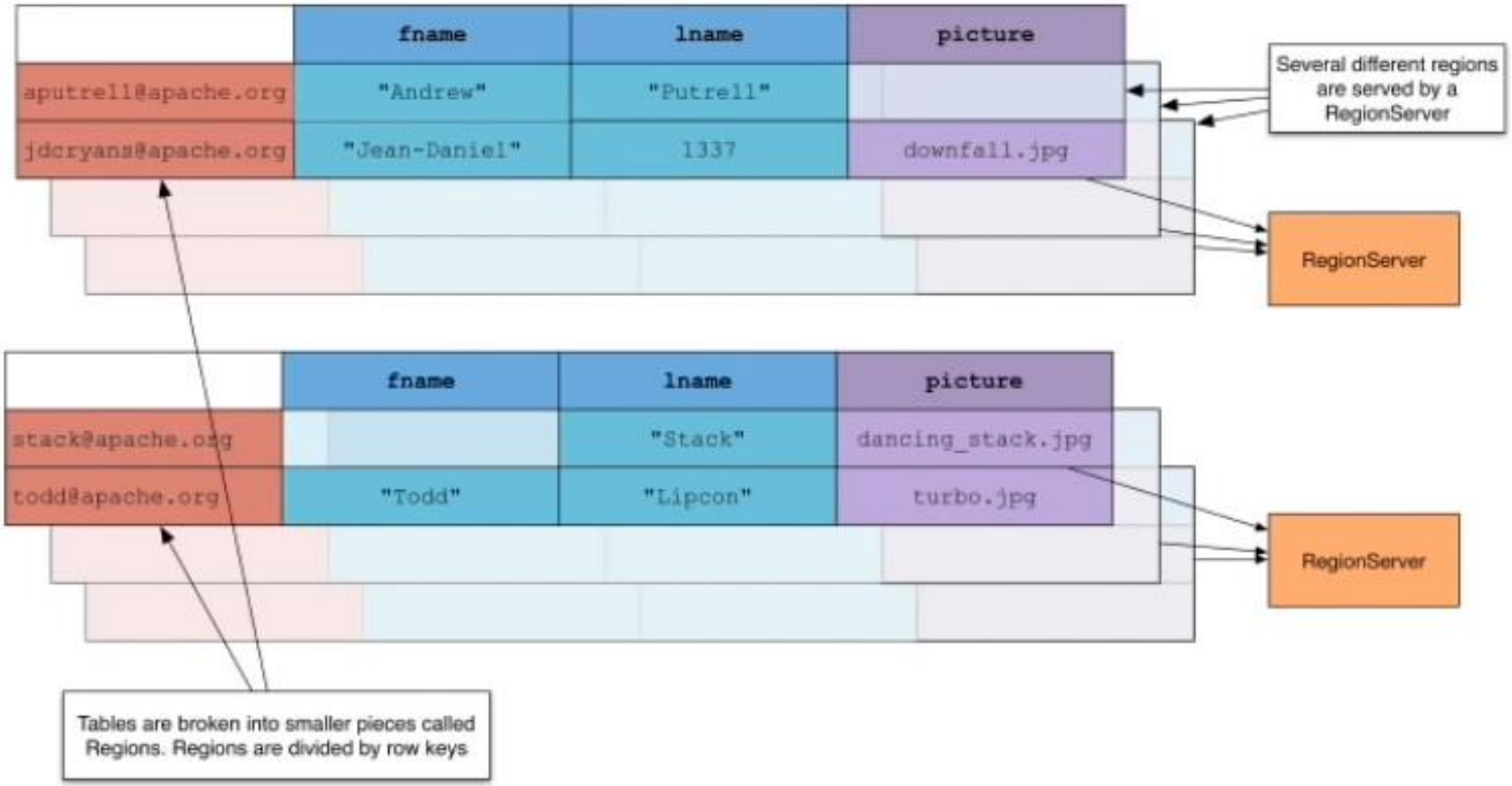
- Each HBase table divided into several regions
  - Each region holds a range of the row keys in the table
  - Row keys must be lexicographically ordered
  - Each region has several stores
    - Column families are assigned to stores
- Regions assigned to region servers for storage
  - Master server responsible for monitoring the region servers
- Hbase uses Apache Zookeeper and HDFS

Hbase Storage and Distributed System Concepts

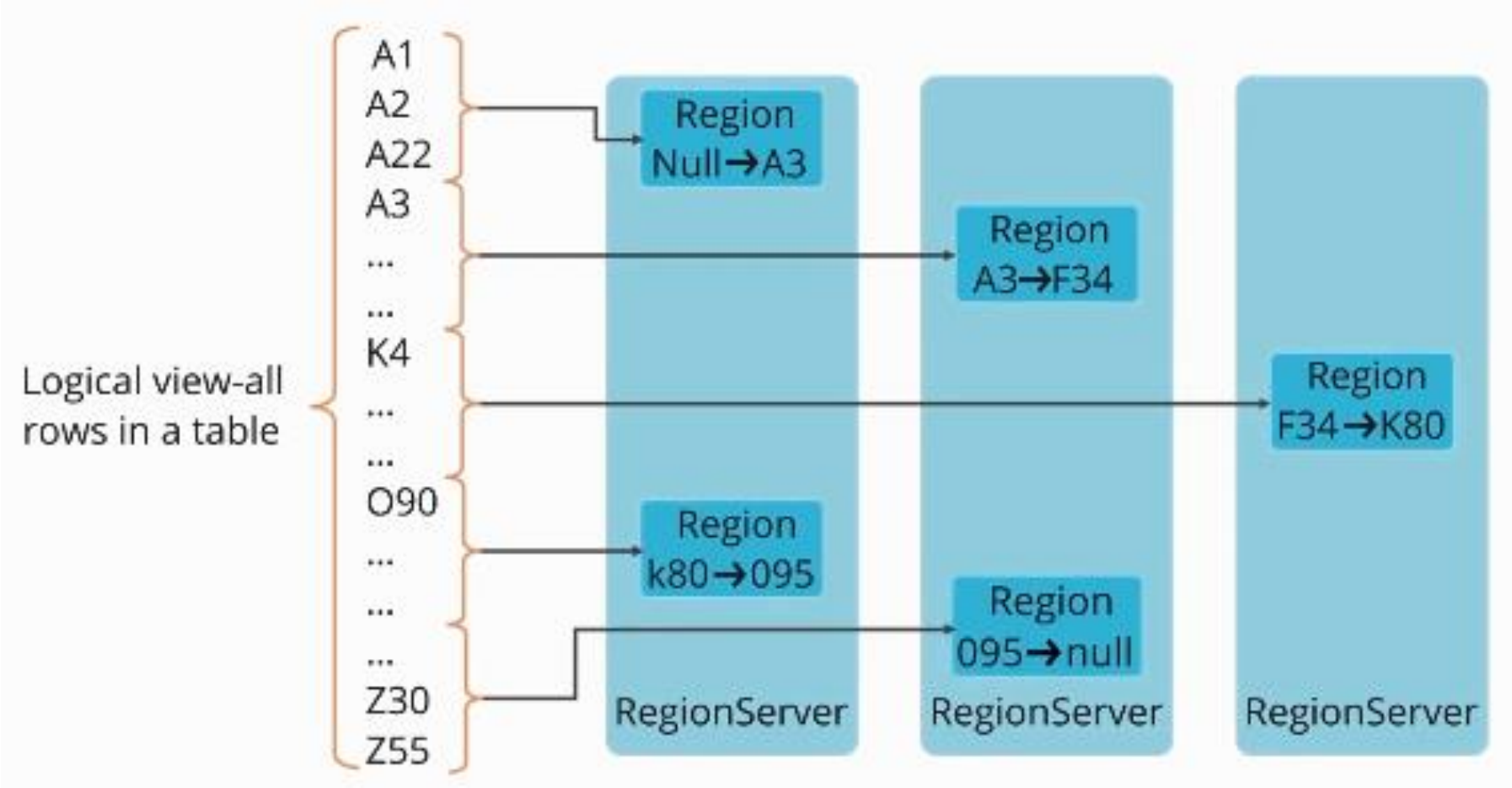



HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Client
Rack 2

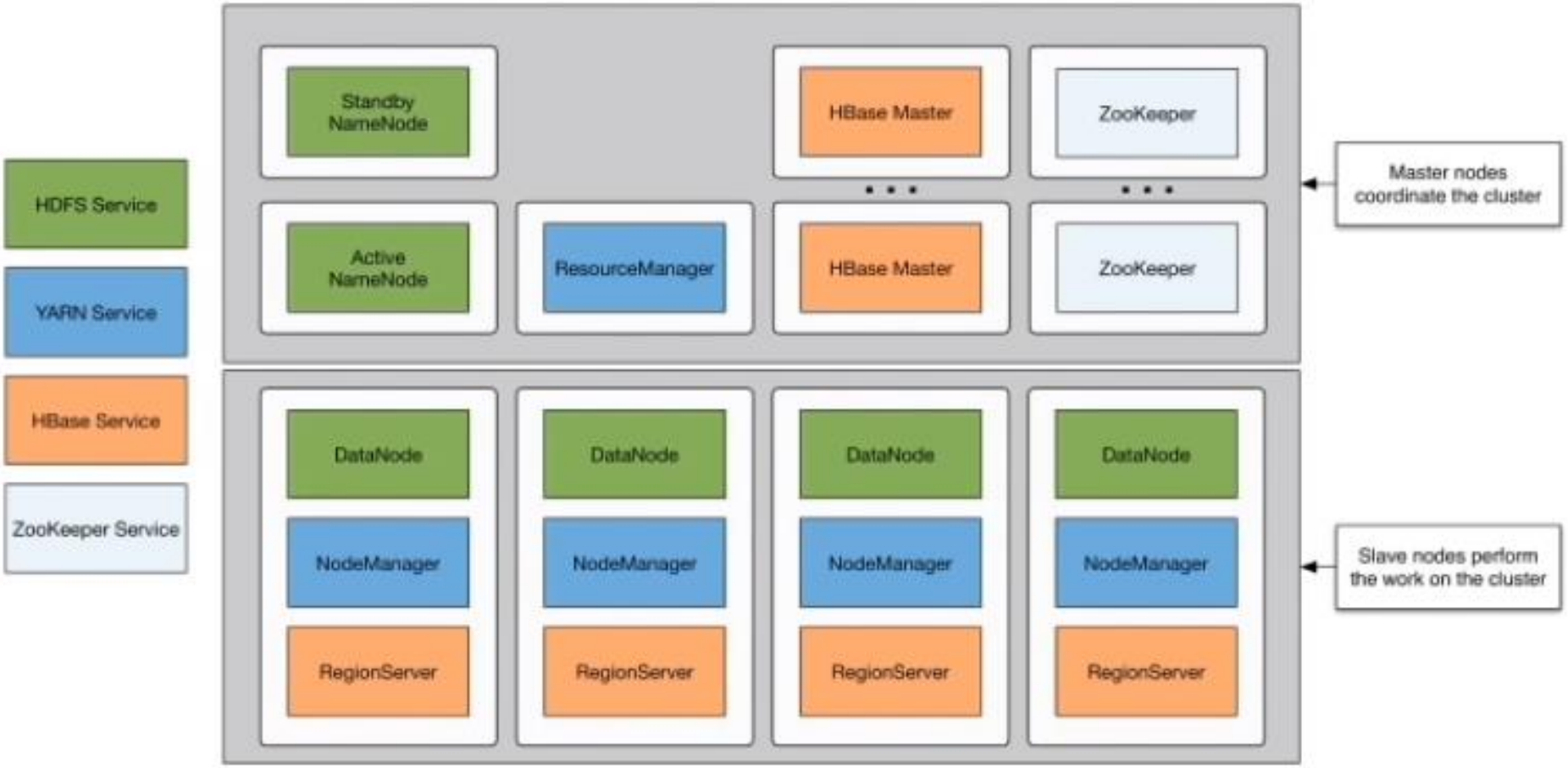
Hbase Storage and Distributed System Concepts – Horizontal Partition



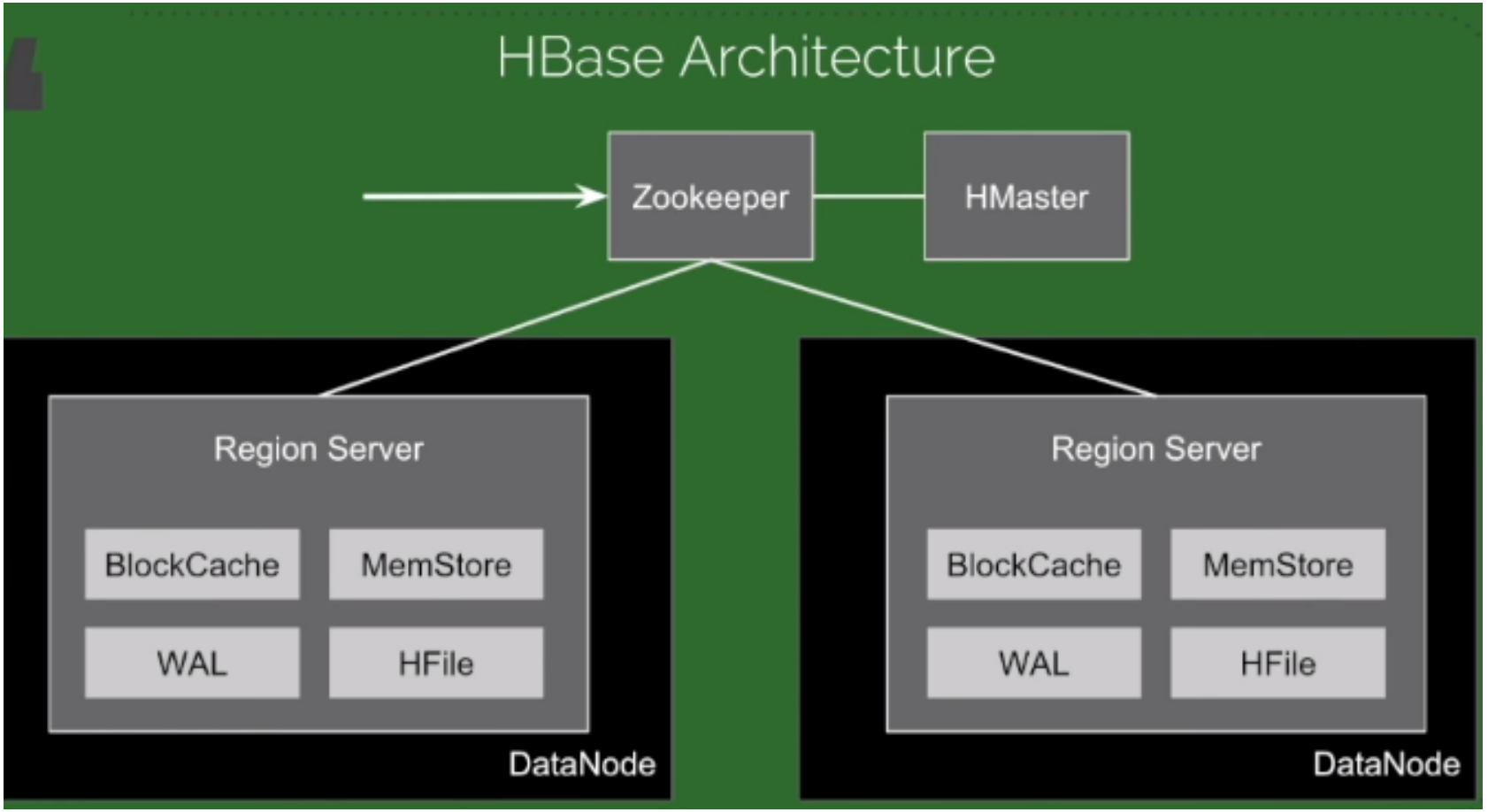
Hbase Storage and Distributed System Concepts



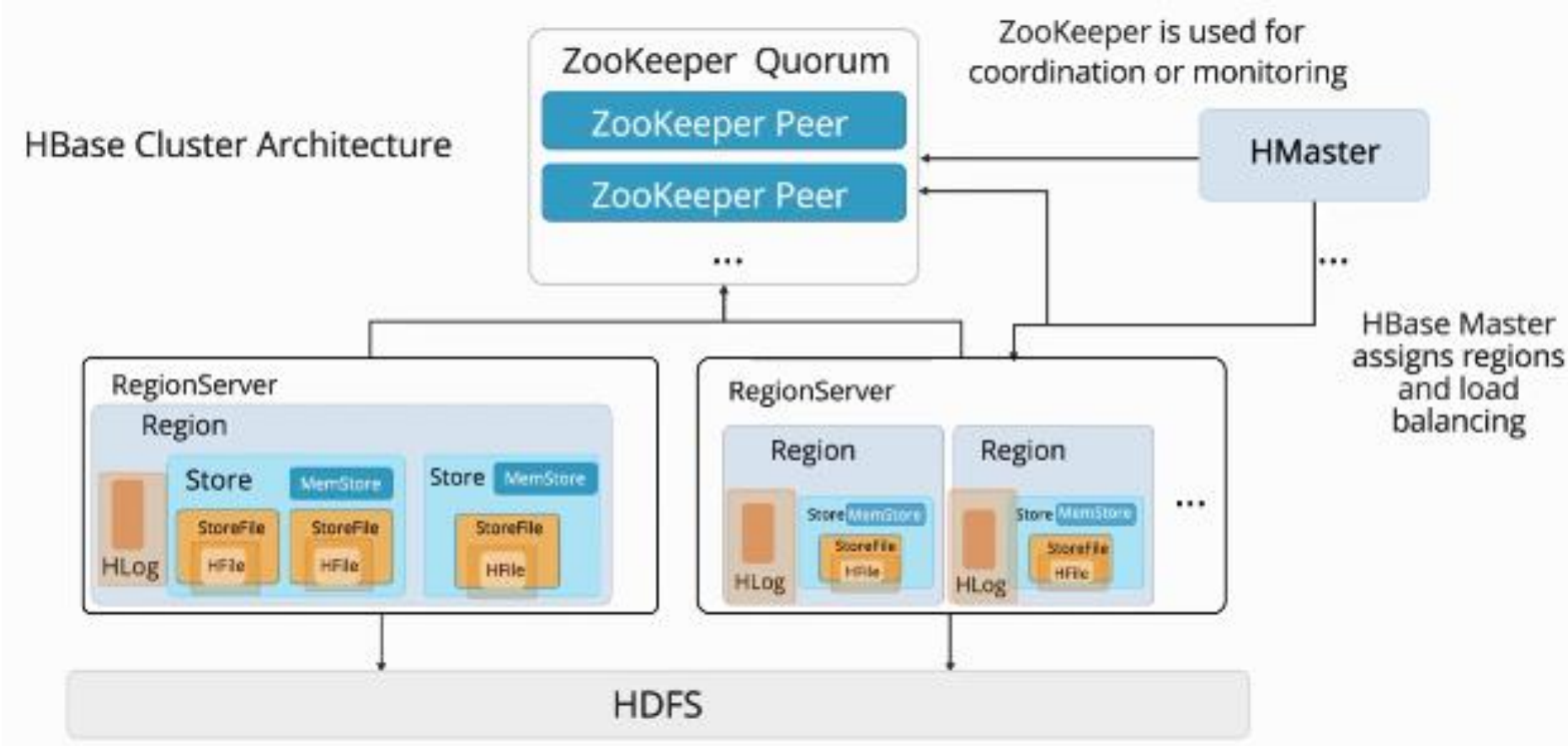
Hbase Storage and Distributed System Concepts



Hbase Storage and Distributed System Concepts



Hbase Storage and Distributed System Concepts



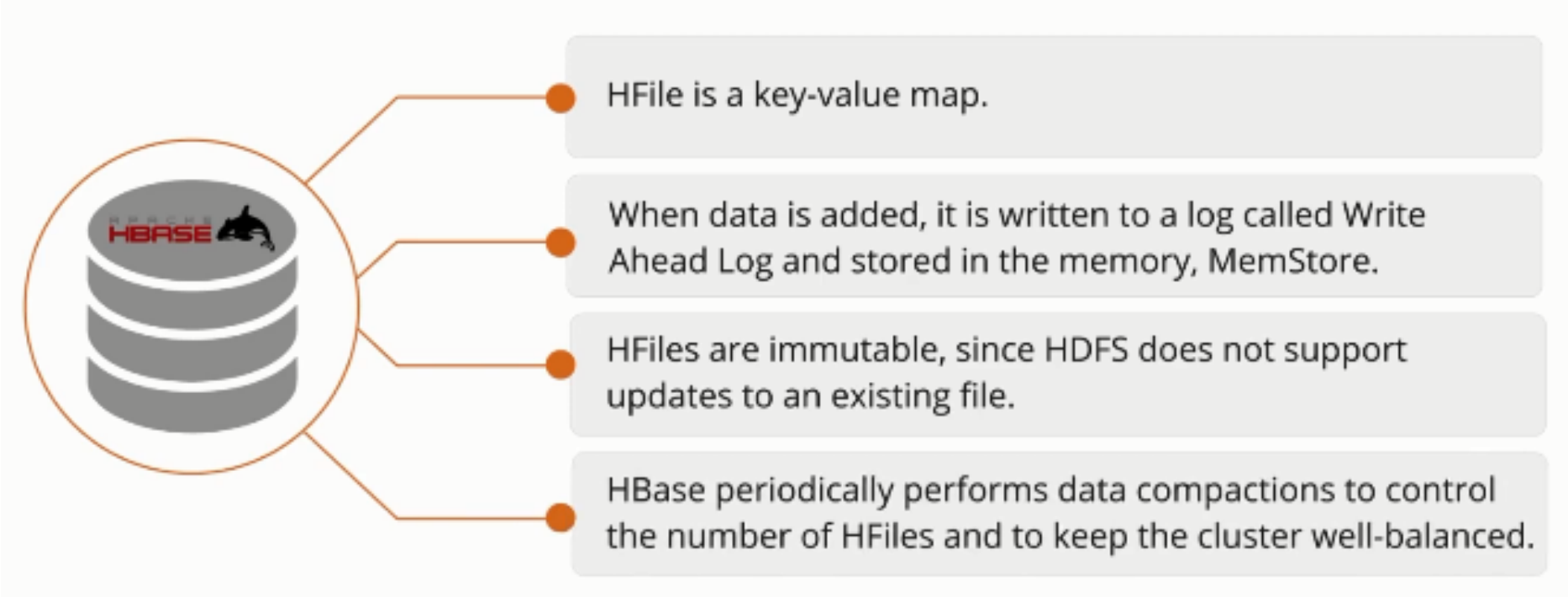
Hbase Storage and Distributed System Concepts

- Table partitioned in to regions
- Rowkey, Columnfamily-columns and timestamp with version will identify a specific row value
- HMaster as Master node, can be multiple but at a time one will be active
- HMaster: Manages the cluster, administers the table creation jobs, handles failovers, also takes up the call for schema or meta data changes in region servers
- Client will ask HMaster where is region for the given Row key then after client will directly talk with that region server
- Zookeeper acts as service registry does bookkeeping job

Hbase Storage and Distributed System Concepts

- Region servers are data nodes
- Region servers may be holding multiple regions
- Region Servers have Blockcache, Memstore, WAL, Hfile
- WAL is the write ahead log before new data becomes persistent
- Block cache (read) will hold frequently accessed data in the cluster
- Memstore acts as write cache, retains data to write until its written to disk, periodically flushing of data to HDFS
- Yarn service for processing and aggregation job

Hbase Storage and Distributed System Concepts



- Data to and from in terms of Bytes as bytearrays
- HFile is actual file stores row sorted key value on disk
- HBase does not replicate region servers instead relies on HDFS replication for data availability

Hbase Storage and Distributed System Concepts