Key-Value Repositories

Knowledge Objectives

1. Explain the need for key-value stores
2. Elaborate on the four goals of key-value stores
3. Define what is a schemaless database
4. Describe how key-value stores improve performance by means of parallelism

Understanding Objectives

1. Explain the two main consequences of schemaless databases

Application Objectives

1. Model simple schemaless databases

AN EXAMPLE OF KEY-VALUE ARCHITECTURE

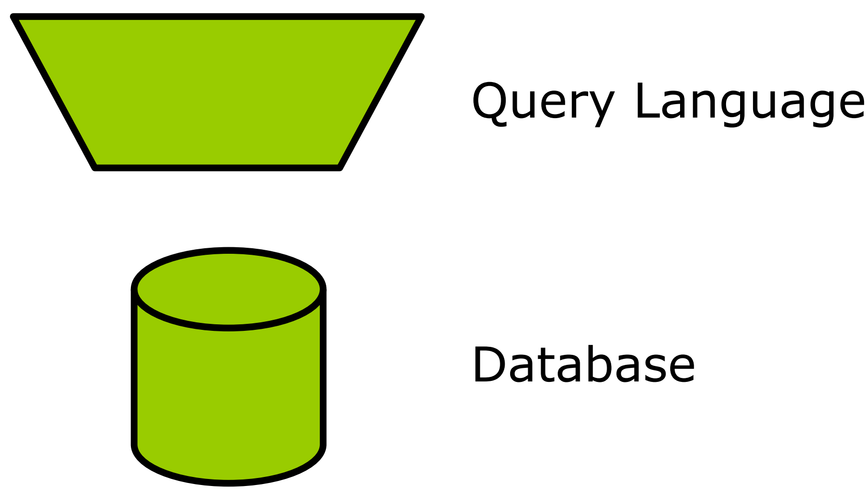
HADOOP, BIGTABLE AND MAPREDUCE

Key-Values: A Piece of History

- Key-values were born as a desperate answer to the RDBMS limitations
- It is widely assumed that Google is the father of Key-value stores
  - Hadoop File System
    - *The Google File System* (2003)
  - MapReduce
    - *Simplified Data Processing on Large Clusters* (2004)
  - HBase
    - *A Distributed Storage System for Structured Data* (2006)

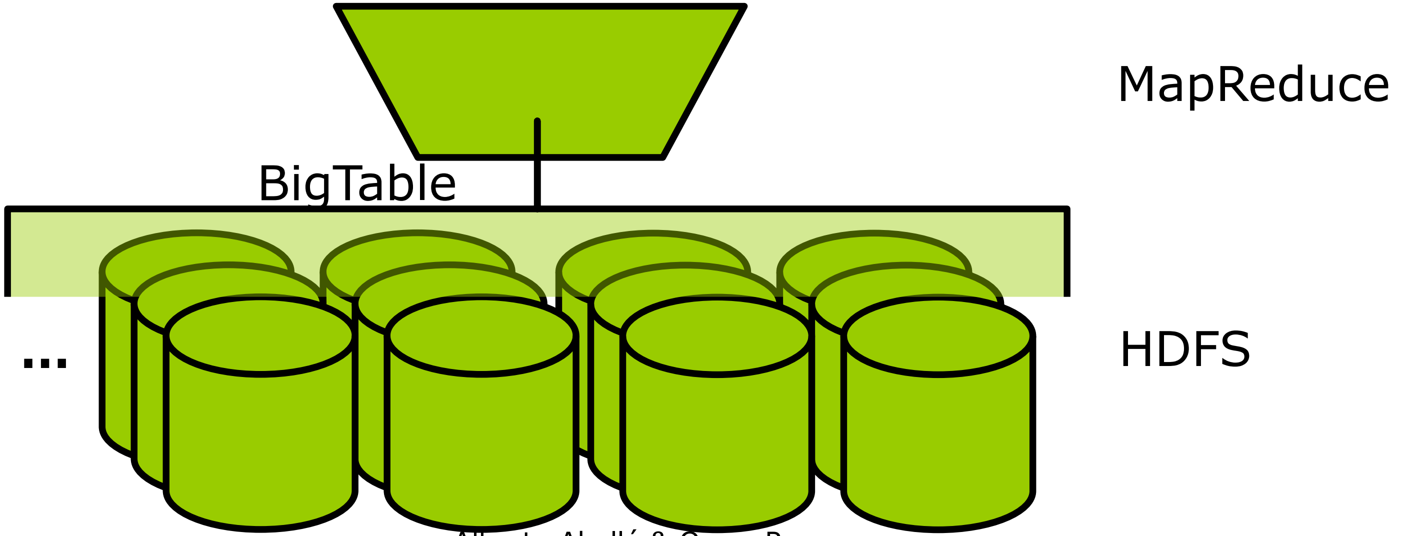
Google Ecosystem

- High-performance is mainly achieved by means of parallelism
  - Divide-and-conquer principle
- MapReduce
  - It is a query language that provides parallelism in a transparent manner



Google Ecosystem

- High-performance is mainly achieved by means of parallelism
  - Divide-and-conquer principle
- MapReduce
  - It is a query language that provides parallelism in a transparent manner



A DISTRIBUTED FILE SYSTEM

THE FILE SYSTEM: HADOOP

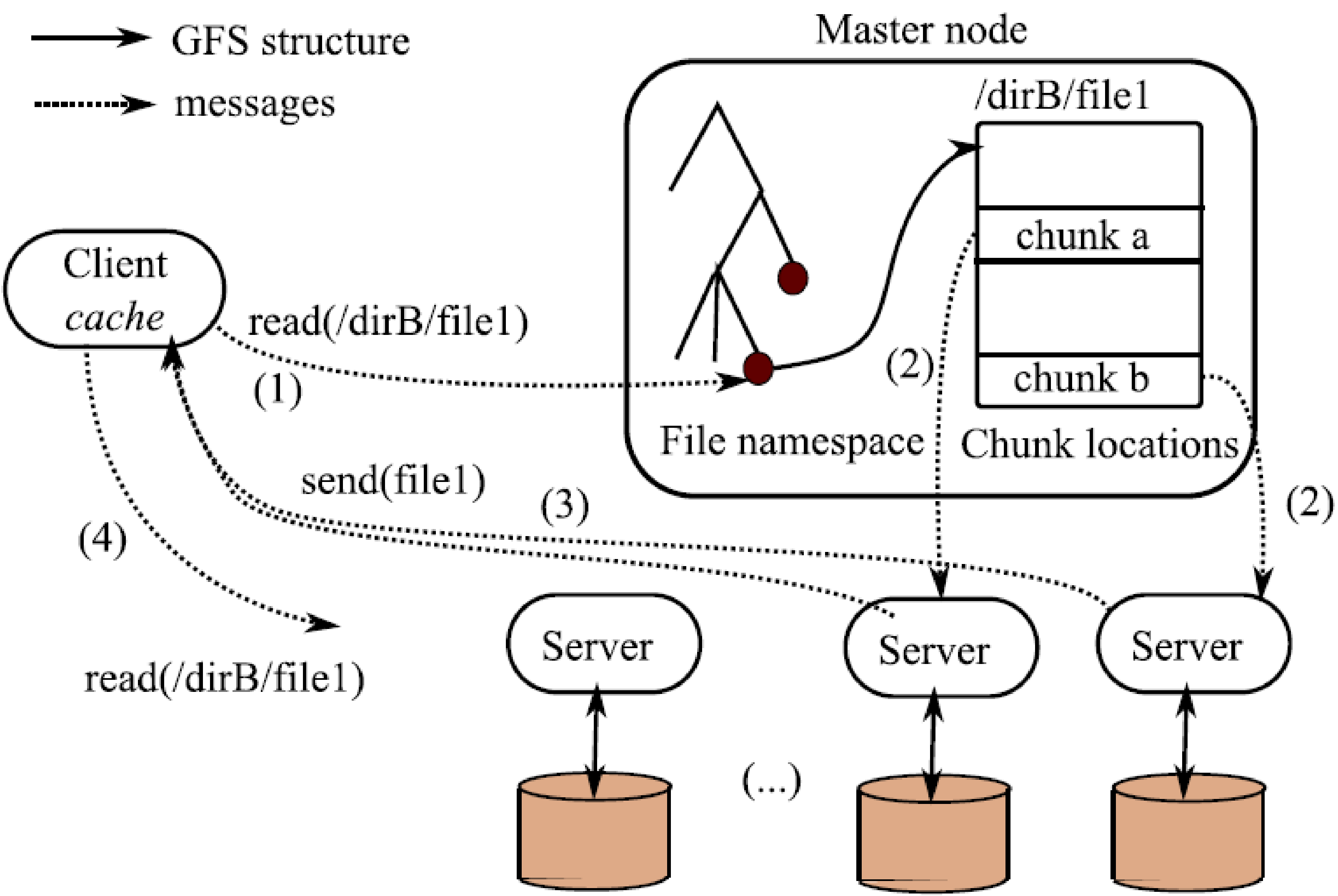
Hadoop File System (HDFS)

- Apache Hadoop (http://hadoop.apache.org)
  - Based on Google File System (GFS)
- Designed to meet the following requirements
  - Handle very large collections of unstructured to semi-structured data
  - Data collections are written once and read many times
  - The infrastructure underlying consists of thousands of connected machines with high failure probability
- Traditional network file systems do partially fulfil these requirements
  - Operating Systems Vs. Database Management System
    - Balancing *query* load (e.g., by means of *fragmentation and replication*) boosts availability and reliability
      - HDFS: Equal-sized file chunks evenly distributed

HDFS in a Nutshell (I)

- A single master (coordinator)
  - Receives client connections
  - Maintains the description of the global file system namespace
  - Keeps track of file chunks (default: 128Mb)
- Many servers
  - Receive file chunks and store them
- A single master design forfeits availability and scalability
  - Availability and reliability: Recovery system
    - Replication (a chunk <u>always</u> in 3 servers, by default)
    - Monitors the system with heartbeat messages to detect failures as soon as possible
    - Specific recovery system to protect the master
  - Scalability: Client cache

HDFS in a Nutshell (II)



HDFS File Formats

- **Parquet**
  - Native columnar storage
  - Rich header metadata (including schema information)
  - Supports advanced block compression techniques
  - Recognised by most popular processing engines, such as Spark
    - It allows SQL-like querying (e.g., Spark SQL)
    - Selections and Projections can be pushed down to disk
    - Statistics used to skip whole fragments
- **Avro**
  - Native row-oriented storage
  - Rich header metadata (including schema information)
  - Supports advanced block compression techniques
  - Recognised by most popular processing engines, such as Spark
    - It allows SQL-like querying (e.g., Spark SQL)
- Text / CSV / JSON formats
  - Do not support block compression
  - Fix-sized splitting with no metadata
  - Not advisable for in-Hadoop processing
  - Typically used to store *raw* data
- ... and many others: Arrow, ORC Files, Sequence Files, etc.

AN EXAMPLE OF KEY-VALUE ARCHITECTURE

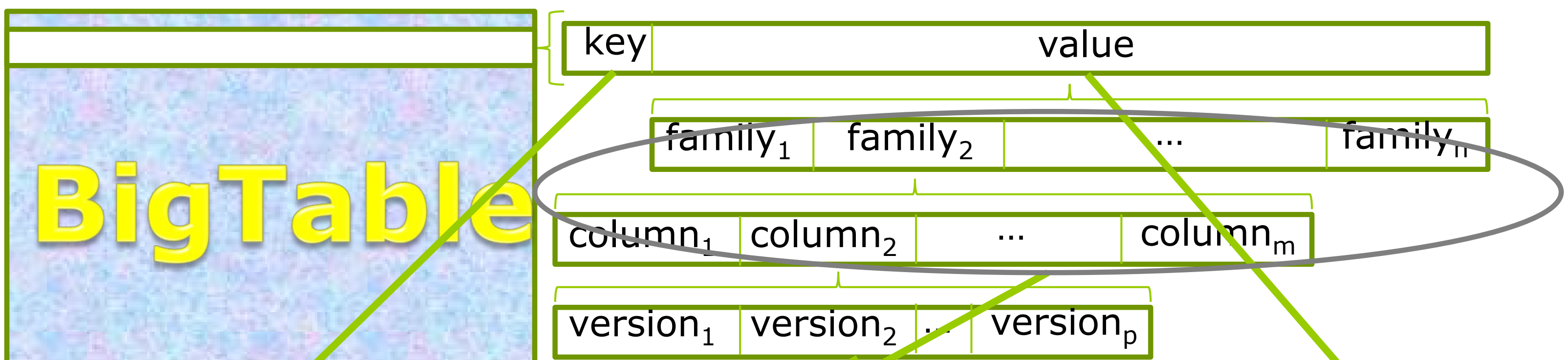
THE DATABASE: HBASE

HBase

- Apache HBase (http://hbase.apache.org)
  - Based on Google's BigTable
- Designed to meet the following requirements
  - Access specific data out of petabytes
  - It must support key search, range search and high throughput for file scans
  - It must support single row transactions
- *Do it yourself* database… own decisions regarding:
  - Data structure
  - Concurrency
  - Recovery
  - CAP trade-off
  - Etc.
- In short, it is a **distributed index cluster** on top of HDFS
  - Distributed B+
  - Tuples are lexicographically sorted according to the key

Schema Elements

- Stores tables (collections) and rows (instances)
  - Data is indexed using row and column names (arbitrary strings)
- Treats data as **uninterpreted** strings (without data types)
- Each cell of a BigTable can contain multiple versions of the same data
  - Stores different versions of the same values in the rows
  - Each version is identified by a timestamp
    - Timestamps can be explicitly or automatically assigned

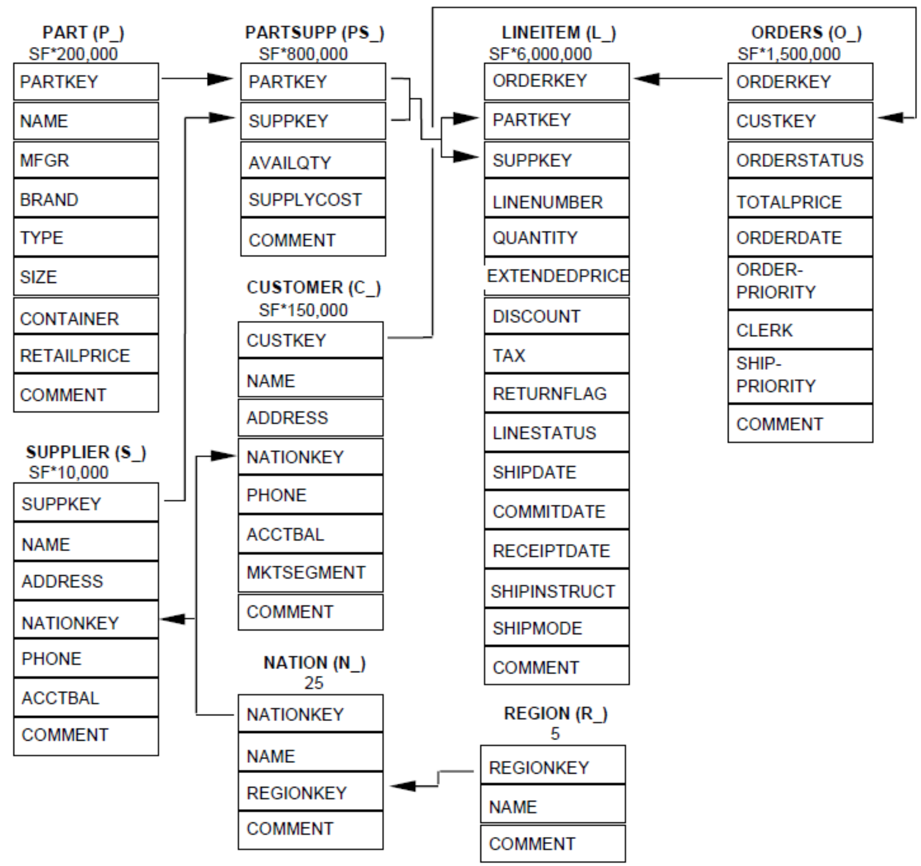


(row:string, column:string[, time:int64])→string

Activity: Key-Value Design

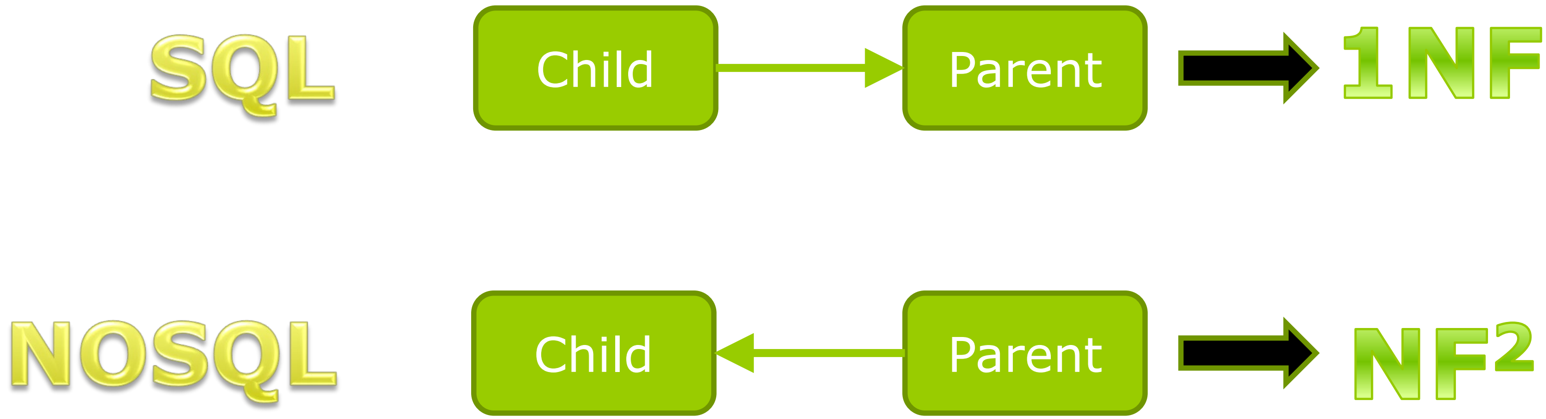
- Objective: Learn the basic design principles of key-value stores
- Tasks:
  1. (20') By pairs, solve the following exercise
     - Model in HBase the lineitem and order tables
     - Model the whole schema
  2. (5') Discussion

```
Q1
SELECT l_orderkey, sum(l_extendedprice*(1-
l_discount)) as revenue, o_orderdate,
o_shippriority

FROM customer, orders, lineitem

WHERE c_mktsegment = '[SEGMENT]' AND c_custkey =
o_custkey AND l_orderkey = o_orderkey AND
o_orderdate < '[DATE]' AND l_shipdate > '[DATE]'

GROUP BY l_orderkey, o_orderdate, o_shippriority

ORDER BY revenue desc, o_orderdate;
```

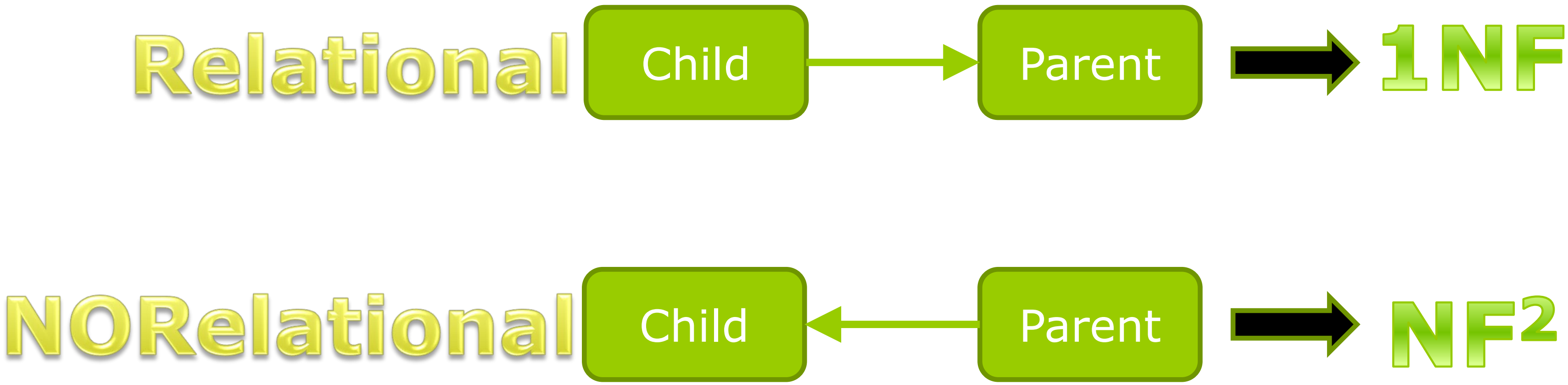


TPC-H Benchmark

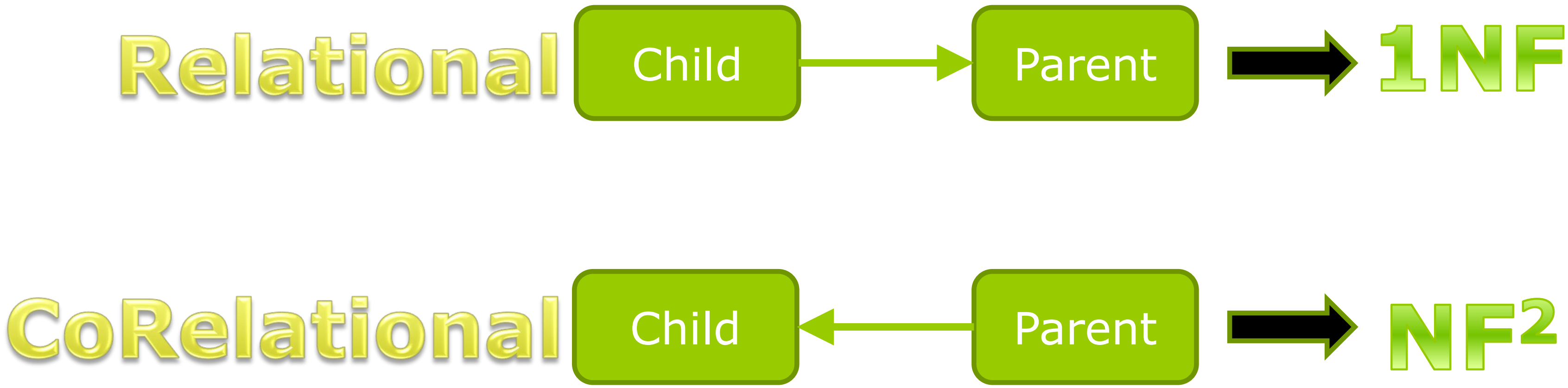
Just Another Point of View



Just Another Point of View



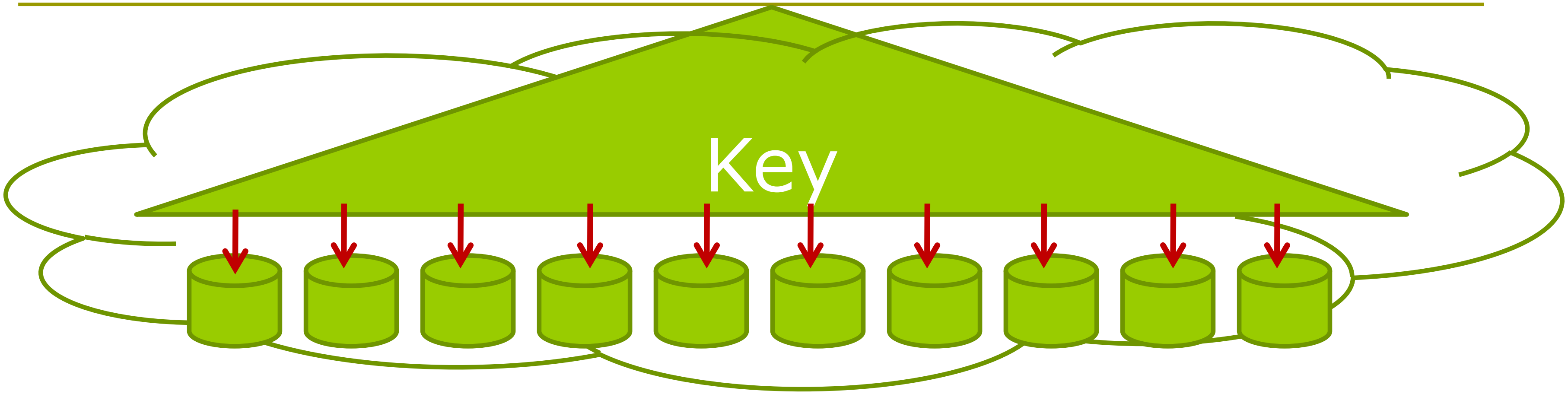
Just Another Point of View



HBase Shell

- ALTER <tablename>, <columnfamilyparam>
- COUNT <tablename>
- CREATE TABLE <tablename>
- DESCRIBE <tablename>
- DELETE <tablename>, <rowkey>[, <columns>]
- DISABLE <tablename>
- DROP < tablename>
- ENABLE <tablename>
- EXIT
- EXISTS <tablename>
- GET <tablename>, <rowkey>[, <columns>]
- LIST
- PUT <tablename>, <rowkey>, <columnid>, <value>[, <timestamp>]
- SCAN <tablename>[, <columns>]
- STATUS [{<u>summary</u>|simple|detailed}]
- SHUTDOWN

Physical Implementation

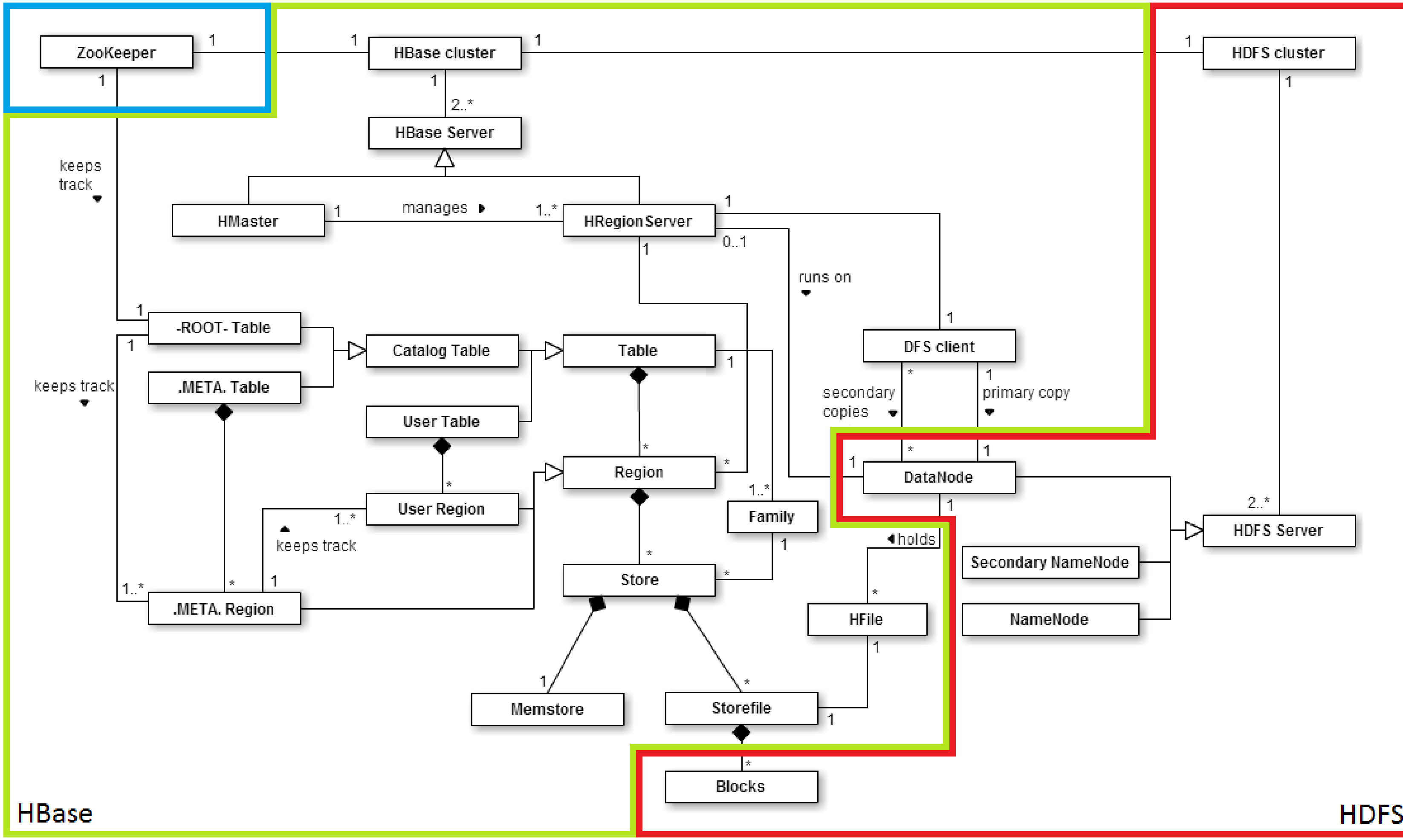


- Each table is horizontally fragmented into *Regions (tablets in BigTable)*
  - Dynamic fragmentation
    - By default into few hundreds of Mbs
  - Distributed on a cluster of machines or cloud
  - Uses the distributed B+ to decide in what region a tuple is stored
- At each Region rows are stored column-wise according to families (hybrid fragmentation)
  - In disk, as many files as families were defined
  - Static fragmentation (determined by the famílies; also determine data locality)
  - Block compression can be enabled (i.e., column families are compressed together)
- Massive usage of in-memory storage
  - Files in disk have an in-memory counterpart
  - Changes happen in main memory and are not flushed to disk until a compactation happens
    - A compactation merges the in-memory and disk files
- Metadata table (~ catalog)
  - It is the physical implementation of the catalog
  - Tuples are lexicographically sorted according to the key
    - Each row (entry) consists of <key, loc>
      - Key: it is the last key value in *that* Region
      - Loc: it is the physical address of a Region
  - This is a **distributed index cluster** (LSM-tree) on top of HDFS

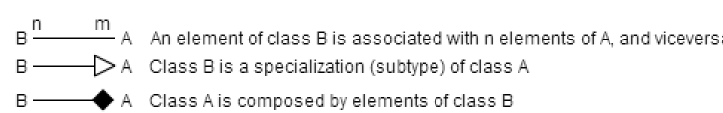
Functional Components (I)

- Zookeeper
  - Quorum of servers that stores HBase system config info
- HMaster
  - Coordinates splitting of regions/rows across nodes
  - Controls distribution of HFile chunks
- Region Servers (HRegionServer)
  - Services HBase client requests
    - Manages stores containing all column families of the region
  - Logs changes
  - Guarantees "atomic" updates to one column family
  - Holds (caches) chunks of HFile into Memstores, waiting to be written
- HDFS
  - Stores all data including columns and logs
    - NameNode holds all metadata including namespace
    - DataNodes store chunks of a file
  - HBase uses two HDFS file types
    - HFile: regular data files (holds column data)
    - HLog: region's log file (allows flush/fsync for small append-style writes)
- HFiles
  - Consist of large (e.g., 64MB) chunks
    - 3 copies of one chunk for availability (default)
- Clients
  - Read and write chunks
    - Locality & load determine which copy to access

Functional Components (II)

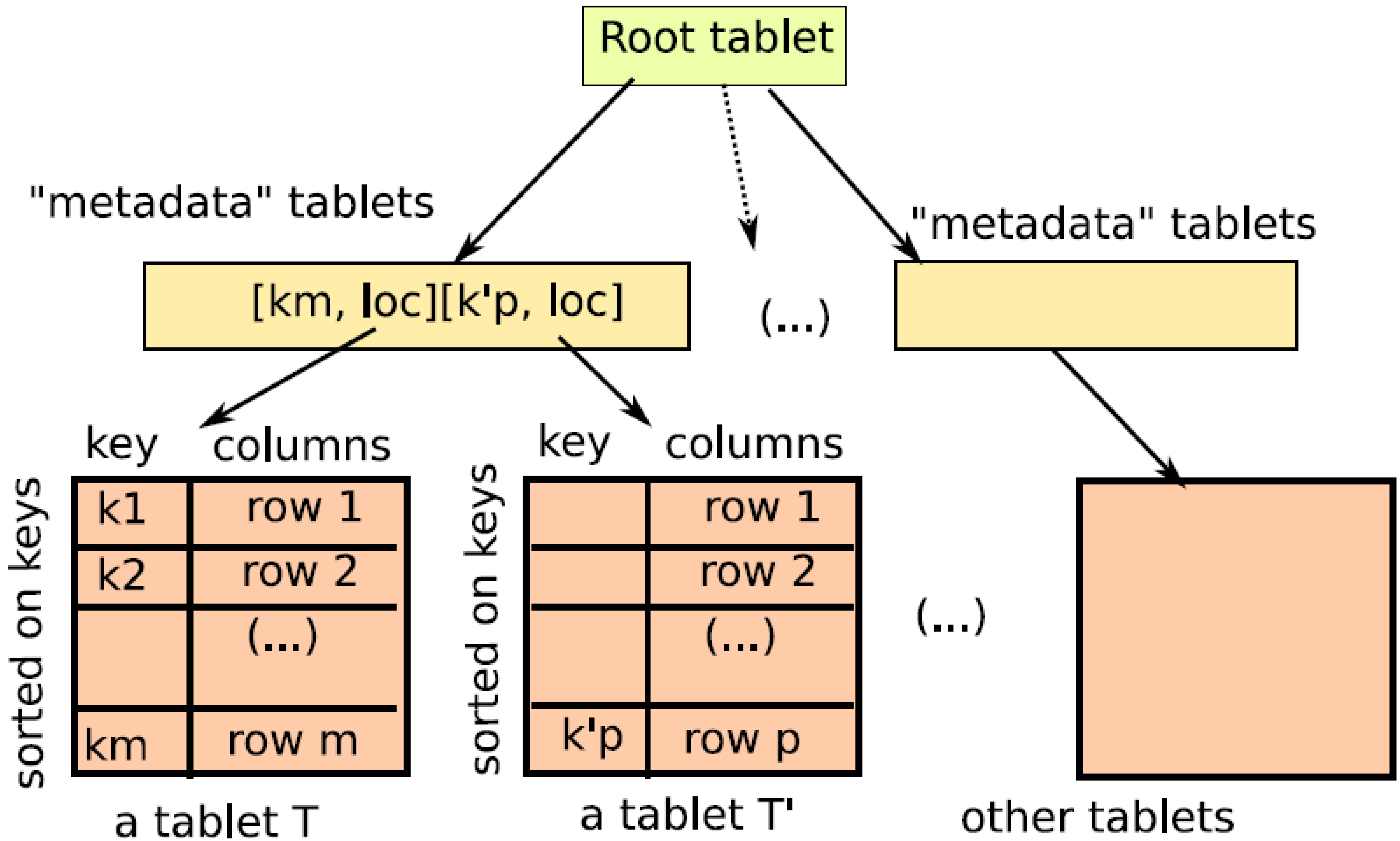


- A primary copy must be stored in the same DataNode the HRegionServer runs on.
- Secondary copies can be stored in any DataNode different from the DataNode the HRegionServer runs on.
- All the stores of a given family correspond to the same table as this family.



By Víctor Herrero

HBase: A Distributed Index Cluster



Serge Abiteboul et al.

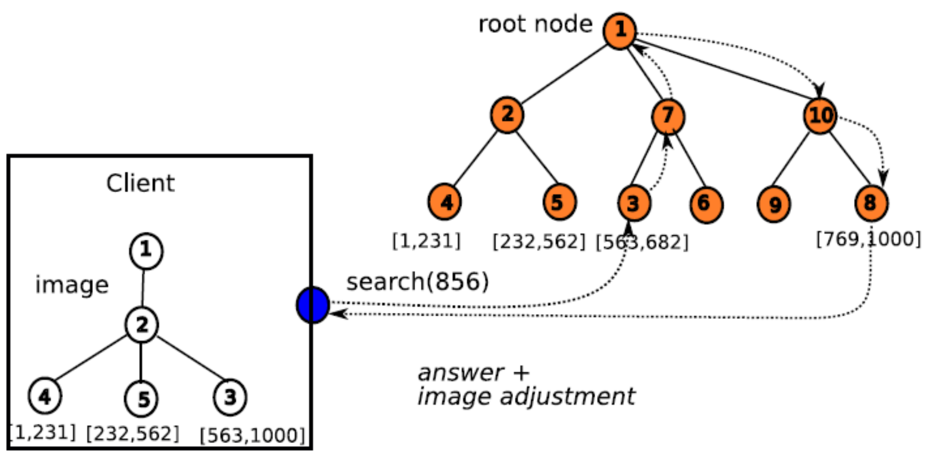
HBase Design Decisions (I)

- One master server
  - Maintenance of the table schemas
    - Root region
  - Monitoring of services (heartbeating)
  - Assignment of regions to servers
- Many region servers
  - Each handling around 100-1000 regions (i.e., horizontal fragments)
  - Apply concurrency and recovery techniques
  - Managing split of regions
    - A region server decides to split (e.g., >128MB)
    - Half of its regions are sent to another server
  - Managing merge of regions
- Client nodes

HBase Design Decisions (II)

- Split and merge affects the distributed tree, which must be updated
  - *Gossiping*
  - Lazy updates: discrepancies may cause out-of-range errors, which triggers a stabilization (**mistake compensation**) protocol
- Mistake compensation
  - The client keeps in cache the tree sent by the master and uses it to access data
  - If an out-of-range error is triggered, it is forwarded to the parent
    - In the worst case, 6 network round trips

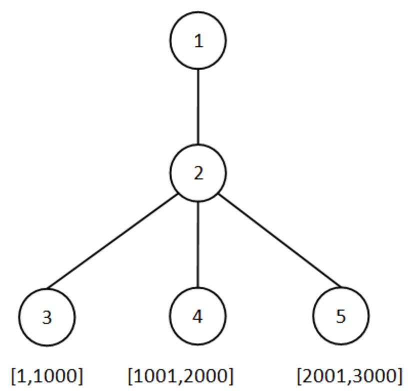
HBase Design Decisions (II)



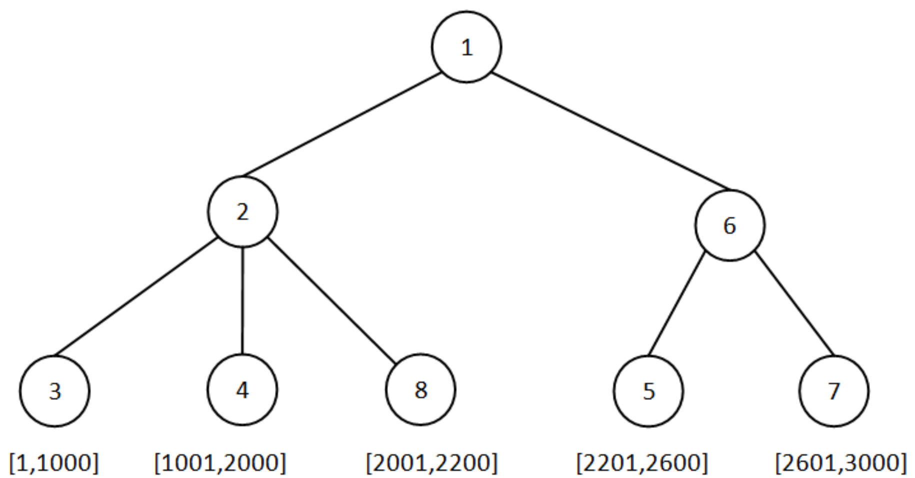
- Mistake compensation
  - The client keeps in cache the tree sent by the master and uses it to access data
  - If an out-of-range error is triggered, it is forwarded to the parent
    - In the worst case, 6 network round trips

Activity: Mistake Compensation

- *Objective: Understand how the global catalog is handled in HBase*
- *Tasks:*
  1. *(10') By pairs, solve the following exercise*
     a) *Number of round trips if search(2602)*
     b) What is the expected number of round trips (i.e., in the average) for the next operation in the client
  2. *(-) Hand in the solution*
  3. *(5') Discussion*



Cache (@client)



Global catalog (@master)

HBase in a Nutshell

- HBase was thought to provide a specific answer for a specific problem
  - Architecture: Distributed index cluster
    - In-memory sorted map for new rows
    - Merge-sort for merging it with *old* data
  - Data structure: Table / row / families
    - Hybrid fragmentation (first horizontal, then vertical)
    - Thus, it cannot fully benefit from columnar processing
  - Concurrency: Timestamping MVCC (Multiversion)
    - Supports single-row transactions per file
  - Logging (RegionServers)
    - WAL (write-ahead protocol)
    - REDO login (No Force / No Steal)
  - CAP Theorem: CP

HBase Architecture

- Refreshing the NOSQL Challenges
  I. Distributed DB design
     - Data fragments
     - Data replication
     - Node distribution

Horizontal fragmentation (fixed-size chunks)
Replication performed by HDFS: Eager/ primary copy
Load balancing. Tuneable, by default depends on #RegionServers and size of the family file

  II. Distributed DB catalog
     - Fragmentation trade-off: Where to place the DB catalog
       - Global or local for each node
       - Centralized in a single node or distributed
       - Single-copy vs. Multi-copy

Global catalog: distributed tree
Clustered data
Centralized and multi-copy catalog (if several masters)
Eager replication / secondary copy between them

  III. Distributed query processing
     - Data distribution / replication
     - Communication overhead

MapReduce / Spark:
Data locality & parallelism
Mostly query shipping but also data shipping
Fault-tolerant

  IV. Distributed transaction management
     - How to enforce the ACID properties CP
       - Replication trade-off: Queries vs. Data consistency between replicas (updates)
       - Distributed recovery system
       - Distributed concurrency control system

No transactions

HBase:
Concurrency: row level, multiversion timestamping
Recovery: checkpointing logging

  V. Security issues
     - Network security Nothing!

HBase Architecture

- NOSQL Goals
  - Schemaless: No explicit schema [column-family key-value]
  - Reliability / availability: Keep delivering service even if its software or hardware components fail [recovery] / [distribution]
  - Scalability: Continuously evolve to support a growing amount of tasks [distribution]
  - Efficiency: How well the system performs, usually measured in terms of response *time* (latency) and *throughput* (bandwith)

[distribution: CP]

Summary

- Goals of key-value stores
  - Schemaless
    - Consequences
  - Availability
    - Relationship to the CAP theorem
  - Scalability
    - By using the USL
  - Performance
    - Parallelize

Bibliography

- J. Dittrich et al. *Hadoop++: Making a yellow elefant run like a cheetah (without it even noticing)*. VLDB'10
- D. Jiang et al. *The performance of MapReduce: An In-depth Study*. VLDB'10
- Eric Brewer. *The CAP Theorem 12 years later*. http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
- E. Brewer. *Towards Robust Distributed Systems*. Ann. ACM Symp. Principles of Distributed Computing (PODC 00). ACM, 2000
- F. Chang et all. *Bigtable: A Distributed Storage System for Structured Data*. OSDI'06
- Sanjay Ghemawat et al. *The Google File System*. SOSP'03
- Jeffrey Dean et al. *MapReduce: Simplified Data Processing on Large Clusters*. OSDI'04
- L. Liu and M.T. Özsu (Eds.). *Encyclopedia of Database Systems*. Springer, 2009
- N. J. Gunther. *A Simple Capacity Model of Massively Parallel Transaction Systems*. CMG National Conference, 1993