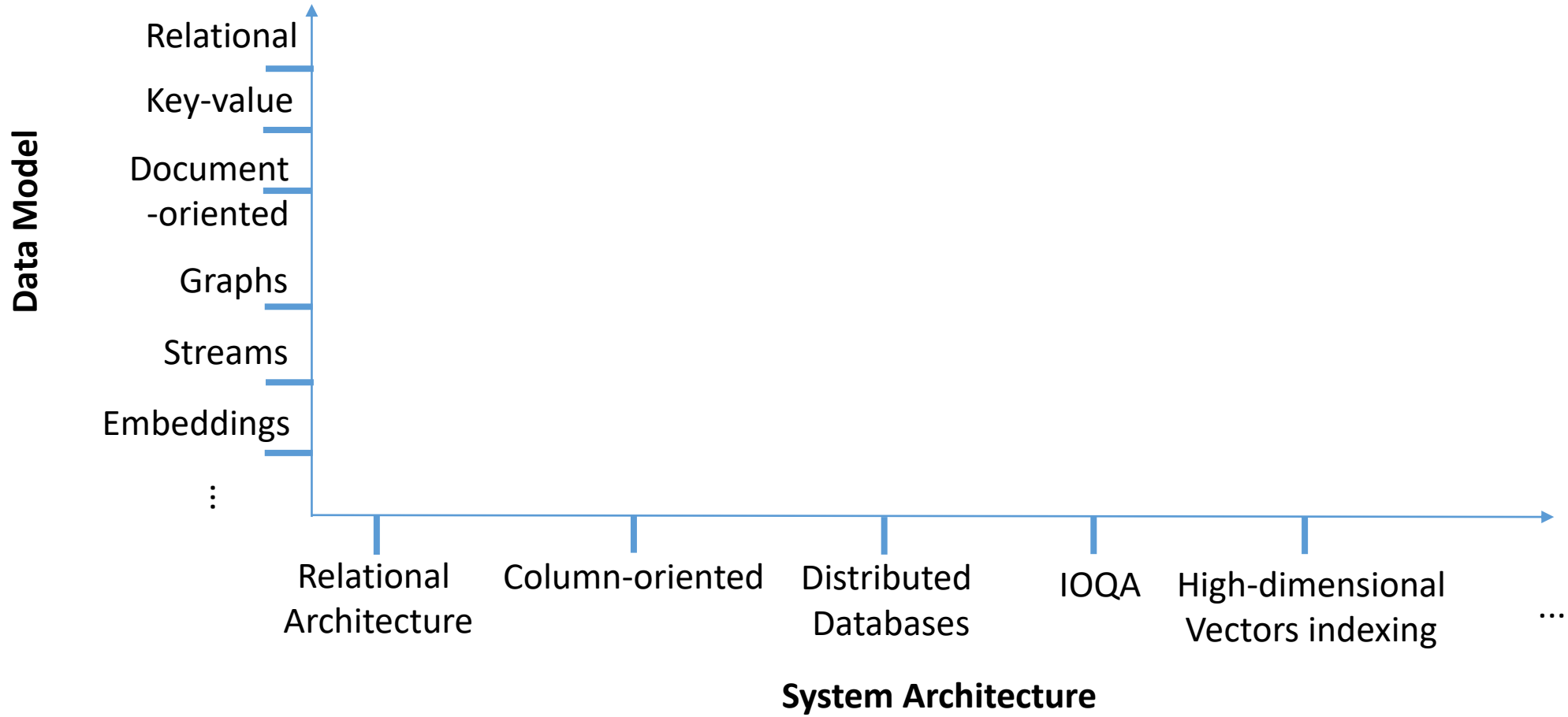
Key-Value Stores

Very Large Databases

The Complexity of Big Data Management

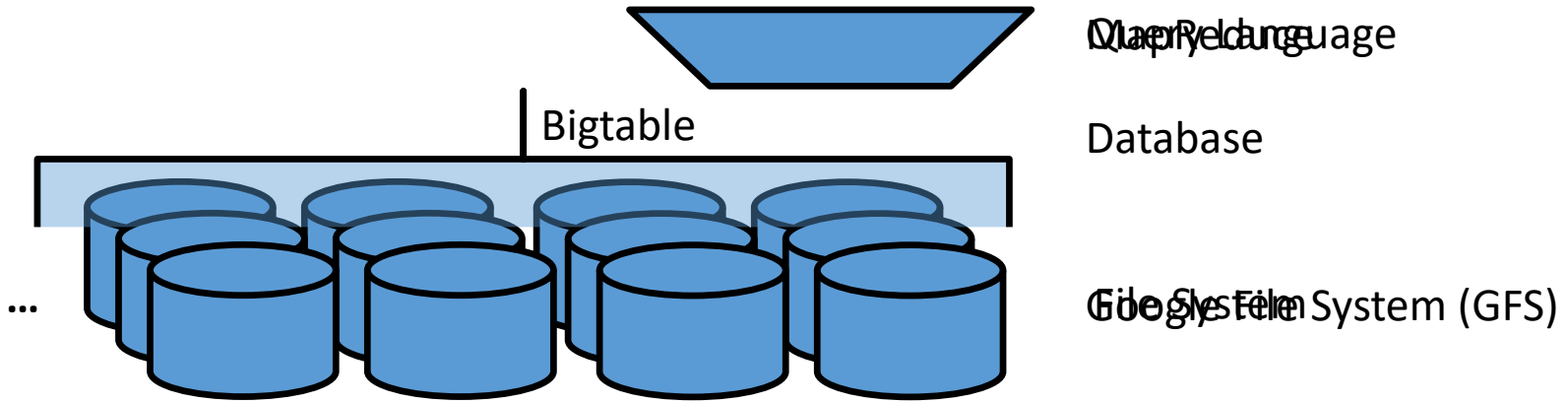


HDFS, HBase and MapReduce

An Example of Key-Value Ecosystem

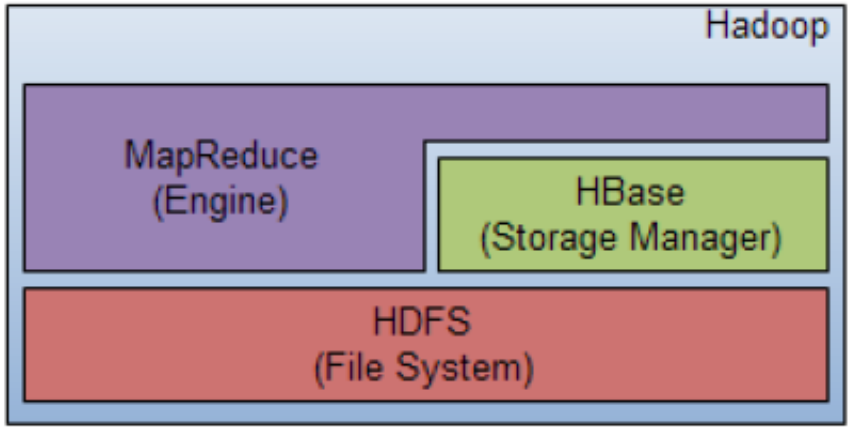
Key-Values: A Piece of History

- 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
- High-performance is mainly achieved by means of parallelism
  - To achieve large-scale parallelism, data is distributed across a cluster
    - Google File System (GFS)
    - GFS + Bigtable
  - MapReduce
    - It is a "query language" that provides parallelism in a transparent manner



Key-Values: A Piece of History

- Hadoop Distributed File System (HDFS)
  - ❑ Based on *The Google File System* (2003)
- Hadoop MapReduce
  - ❑ Based on *MapReduce: Simplified Data Processing on Large Clusters* (2004)
- HBase
  - ❑ Based on *Bigtable: A Distributed Storage System for Structured Data* (2006)

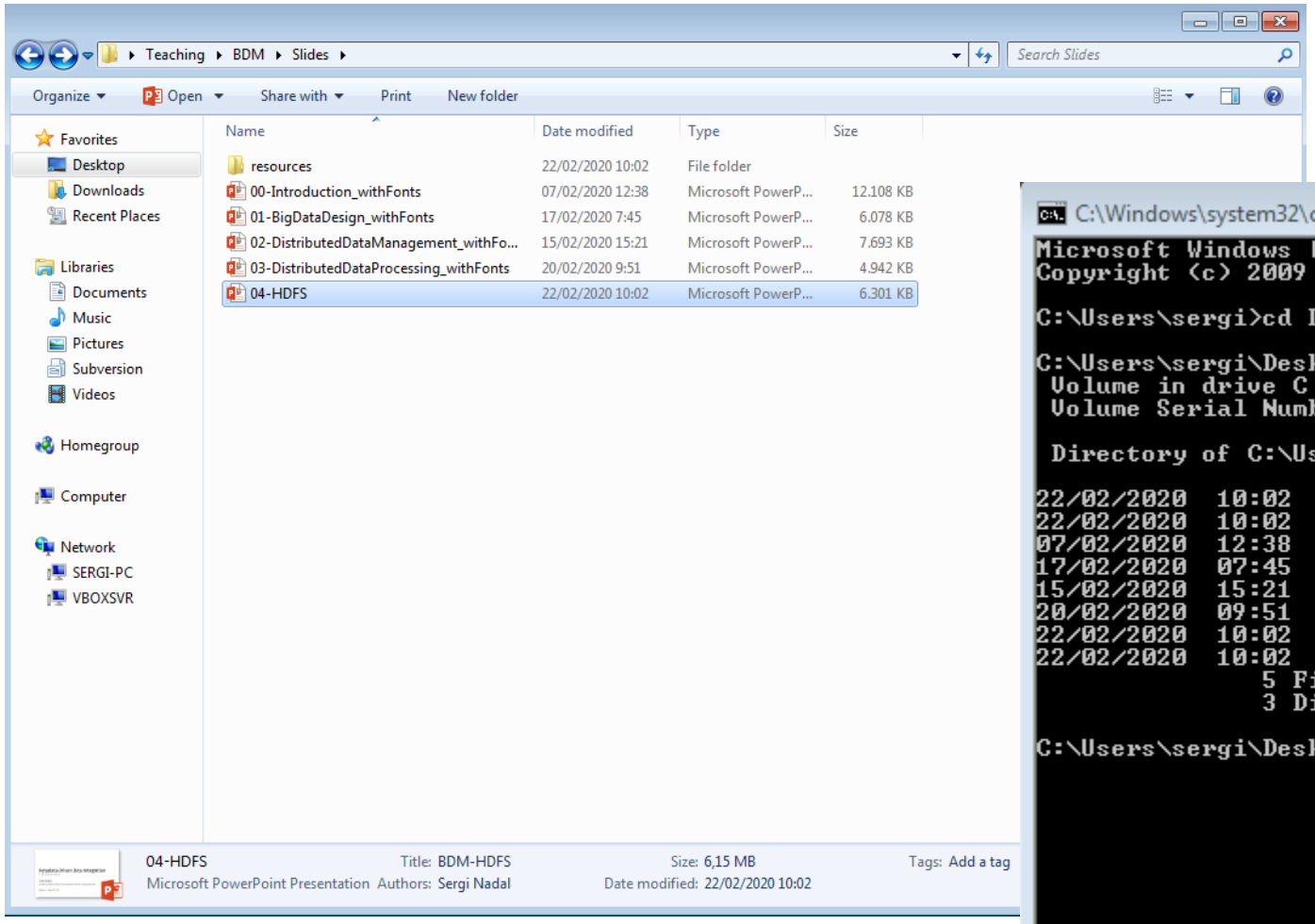


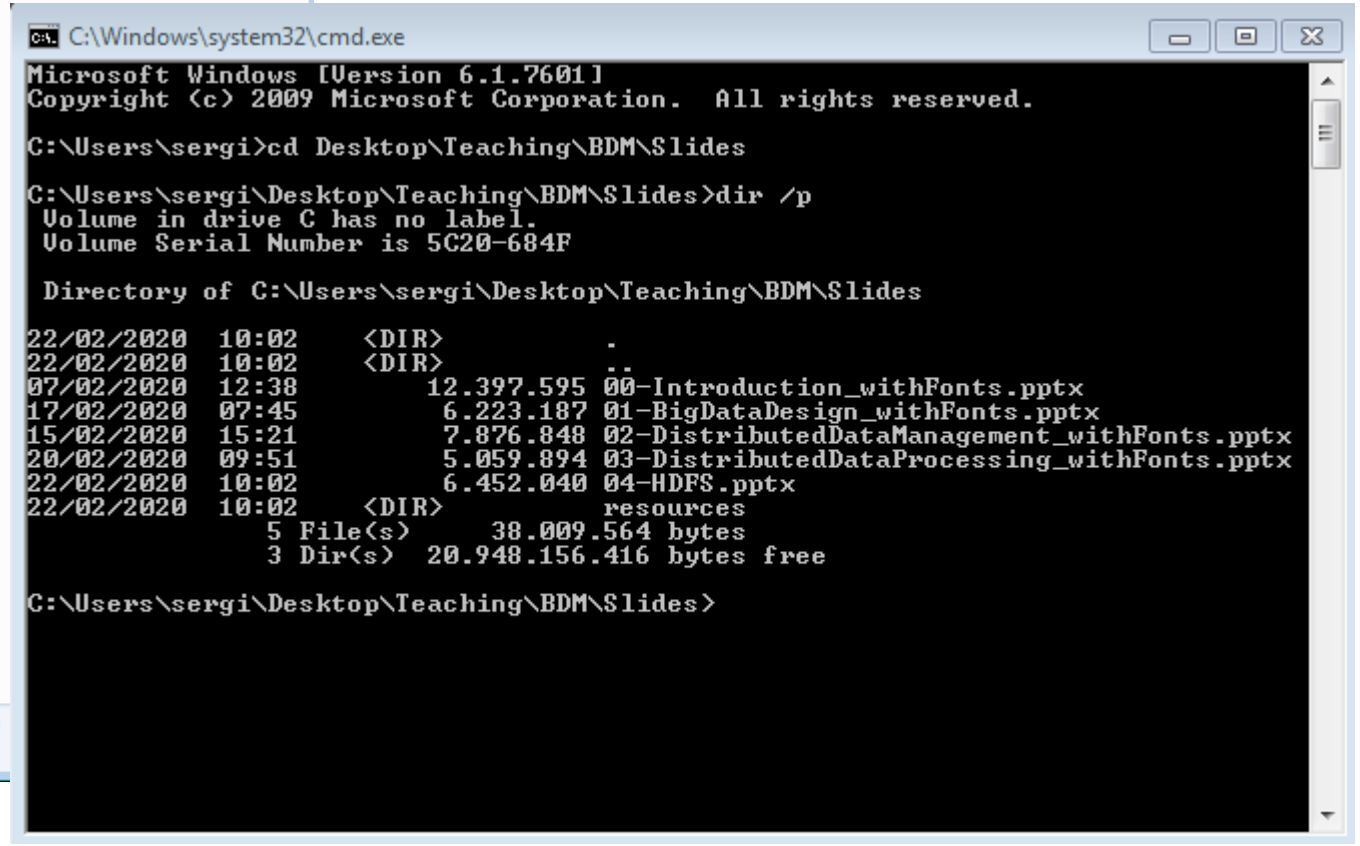
Independent technologies (different processes)

The File System: HDFS

A Distributed File System

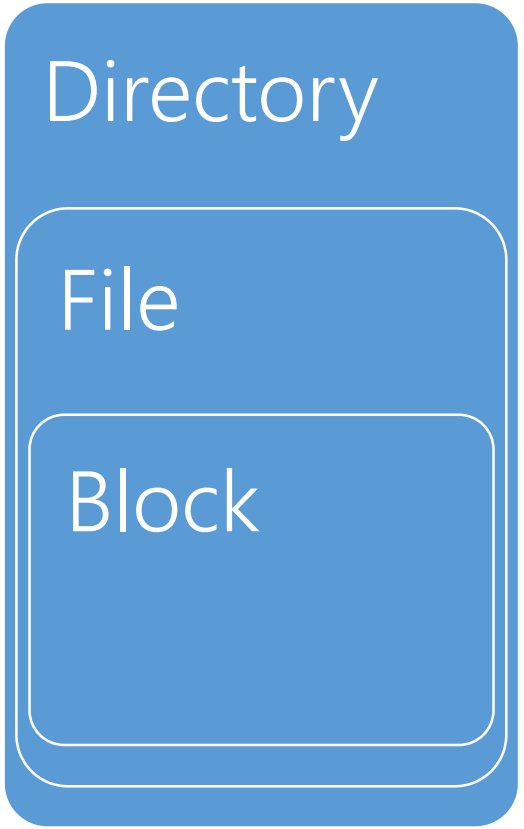
What is a file system?





Functionalities provided by a File System

- Creates a hierarchical structure of data
  - Splits and stores data into files and blocks
- Provides interfaces to read/write/delete
- Maintains directories/files metadata
  - Size, date of creation, permissions, …



Hadoop File System (HDFS)

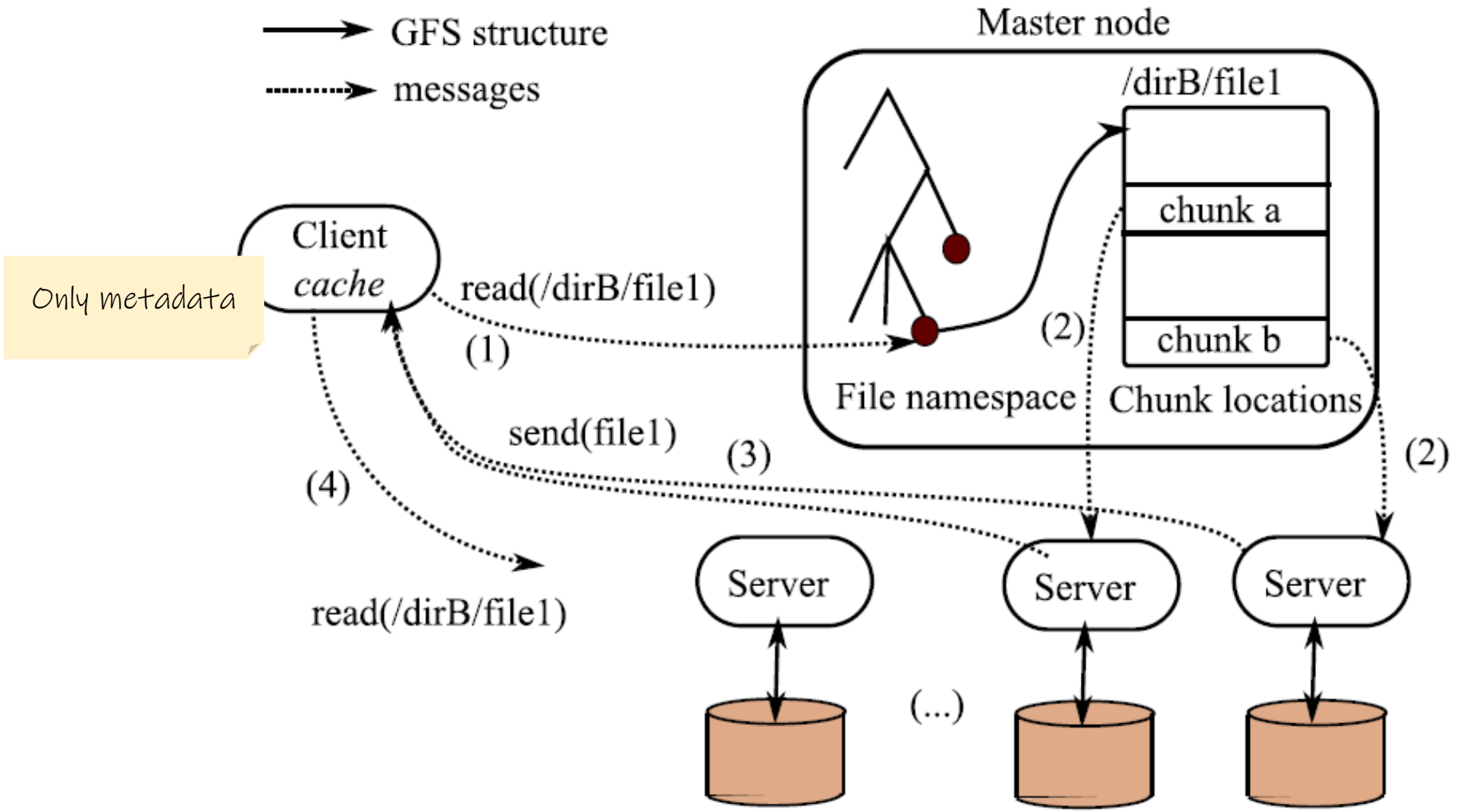
- Apache Hadoop (http://hadoop.apache.org)
  - Based on Google File System (GFS)
- Designed to meet the following requirements
  - Handle many and very large files
  - Data collections are written once and read many times (sequential access pattern)
  - Infrastructure consisting of thousands of inexpensive commodity machines with high failure probability
- Traditional network file systems do partially fulfil these requirements
  - File System 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

Chunks are "equivalent" to blocks in a centralized system

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 always 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)



Only metadata

HDFS File Formats

HDFS provides different file formats to store data:

Text / CSV / JSON formats

- Do not support block compression
- Fix-sized fragments with no metadata
- Not advisable for advanced processing
- Typically used to store *raw* data

Avro

- Native row-oriented storage
- Rich header metadata (including schema information)
- Supports block compression techniques
- Recognised by most popular processing engines, such as Spark
  - It allows SQL-like querying (e.g., Spark SQL)

Parquet

- Native column-oriented storage
- Rich header metadata (including schema information and statistics)
- 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
  - Statistics used to skip whole fragments

... and many others: Arrow, ORC Files, Sequence Files, etc.

The Database: HBase

Richer Data Structure, Recovery, Concurrency and Indexing Capabilities

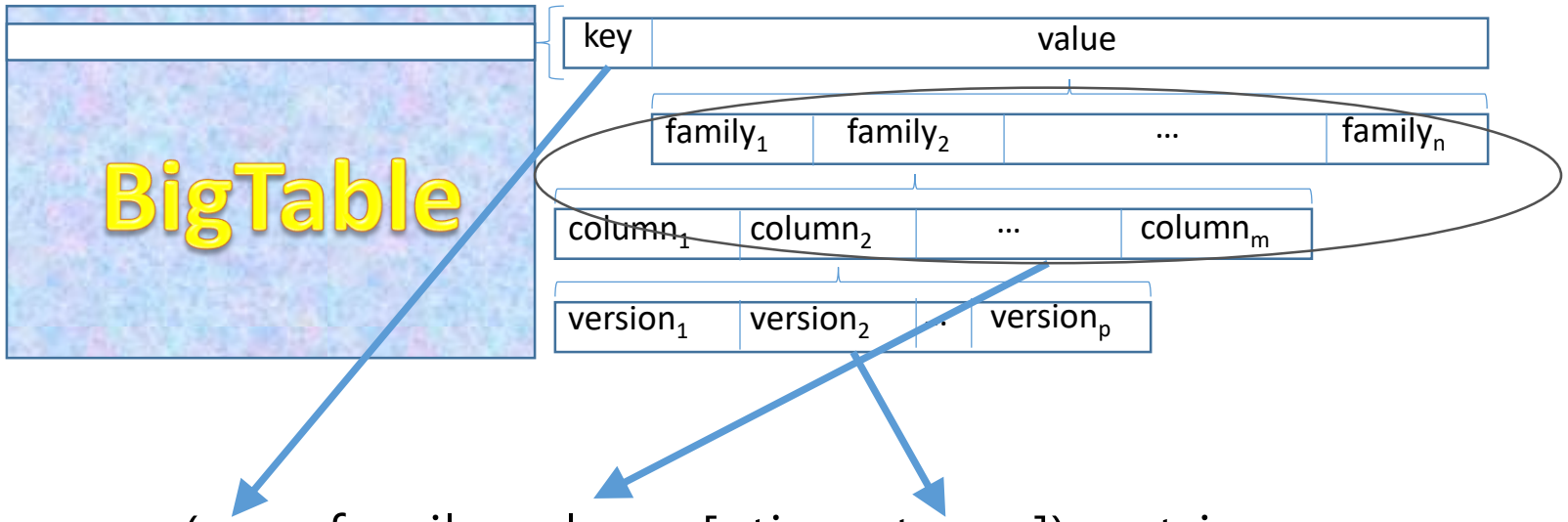
HBase

- 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 scans
- With regard to HDFS, it provides:
  - Richer key-value structure (hybrid fragmentation strategy: horizontal + vertical)
  - Indexing capabilities (to enable efficient key and range searches)
  - Concurrency
  - Recovery
- From an architecture point of view, it is a **distributed index cluster** on top of HDFS
  - Distributed clustered B+
  - Tuples are lexicographically sorted according to the key

Provides random access capabilities on top of HDFS

Schema Elements

- Stores tables (collections) and rows (instances)
  - A cell is identified by row key, column family, and column (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

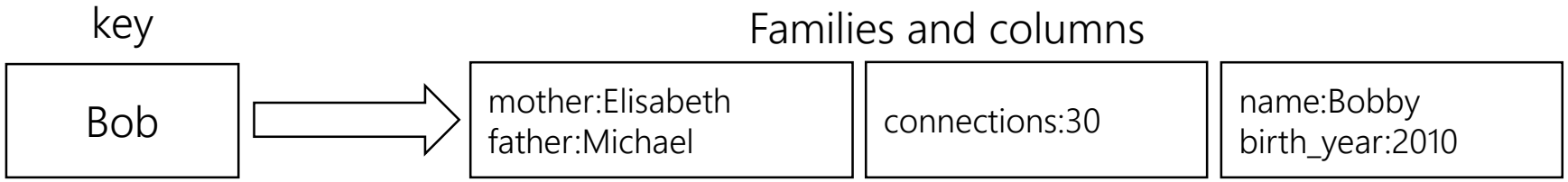


Key-Value

- Key-value stores
  - Entries in form of key-values
    - One key maps only to one value
  - Query on key only
  - Schemaless

- HBase (Wide-column key-value store)
  - Entries in form of key-values
    - But now values are split in wide-columns (column families)
  - Typically query on key, but some support for values (columns)
  - Schemaless within a family

Columns are defined at insertion time



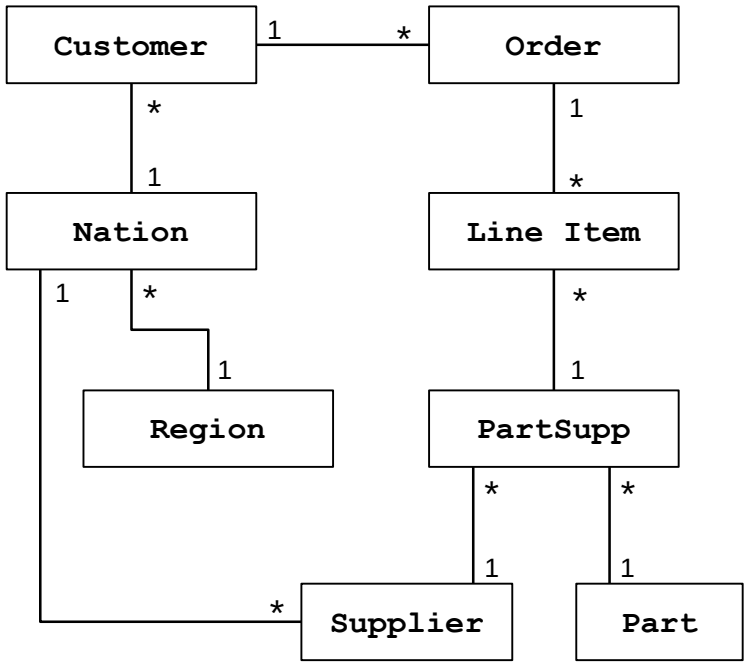
HBase Shell

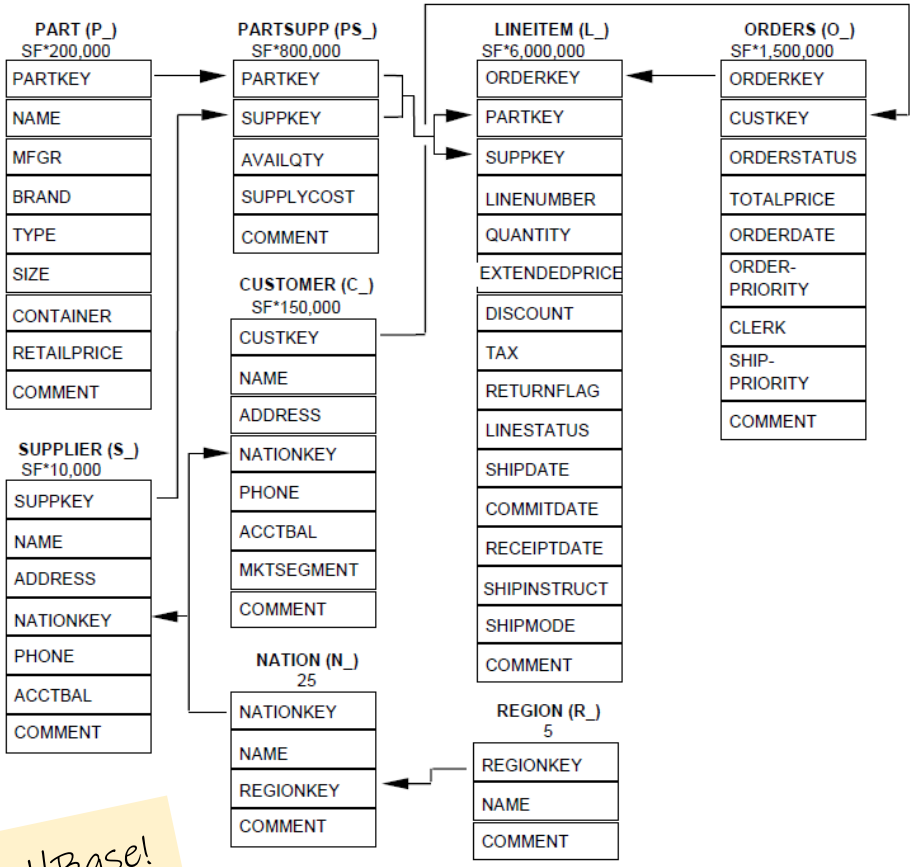
- ALTER <tablename>, <columnfamilyparam>
- COUNT <tablename>
- CREATE TABLE <tablename>
- DESCRIBE <tablename>
- DELETE <tablename>, <rowkey>[, <columns>]
- DELETEALL <tablename>, <rowkey>
- DROP < tablename>
- EXIT
- EXISTS <tablename>
- GET <tablename>, <rowkey>[, <columns>]
- LIST
- PUT <tablename>,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

SCAN is parallel!

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 Orders tables*
  2. *(5') Discussion*





TPC-H Benchmark

No JOINs in HBase!

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 Orders tables*
  2. *(5') Discussion*

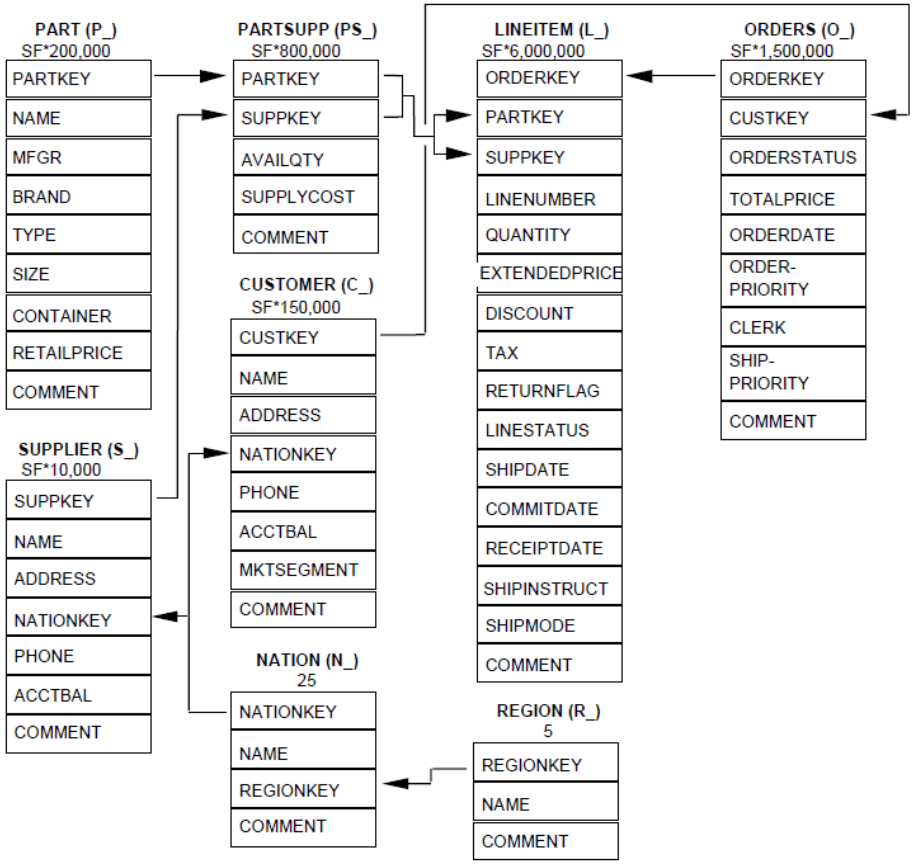
SELECT l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue, o_orderdate, o_shippriority

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]'

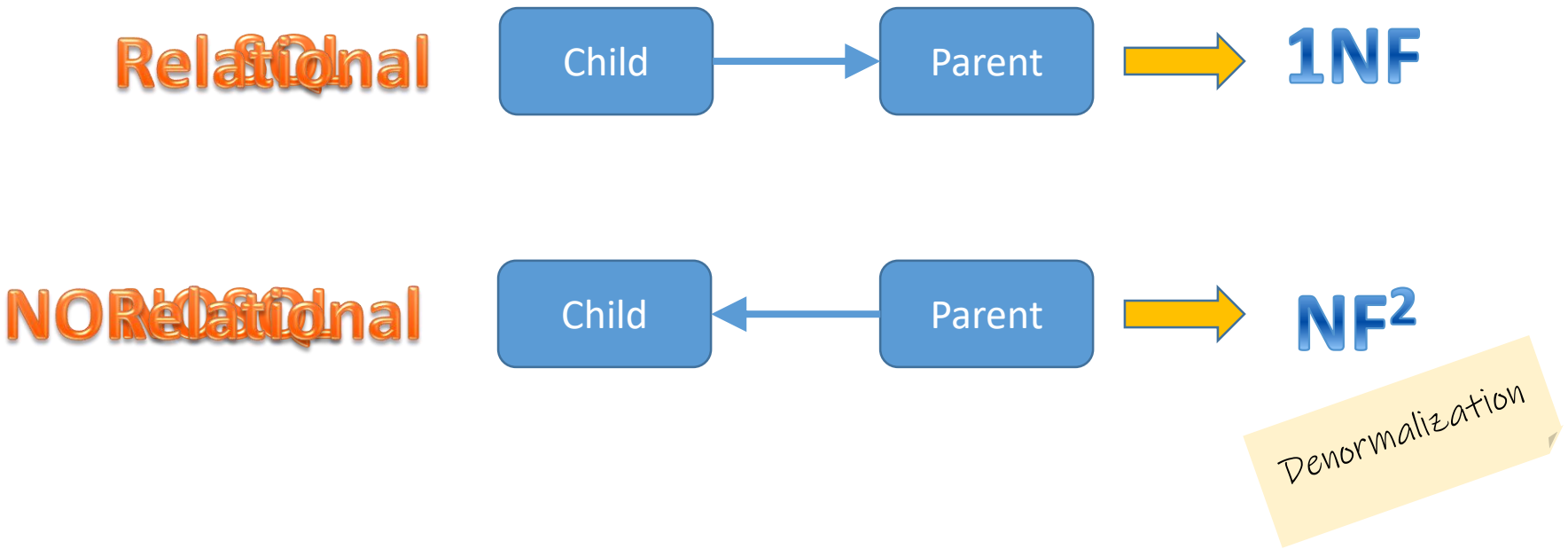
GROUP BY l_orderkey, o_orderdate, o_shippriority

ORDER BY revenue desc, o_orderdate;

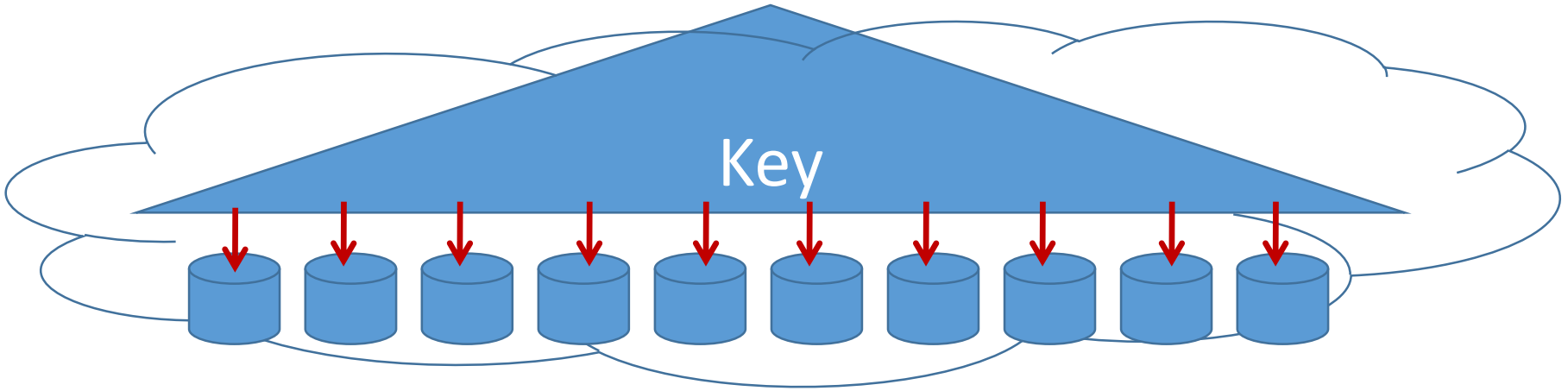


TPC-H Benchmark

Just Another Point of View



Physical Implementation

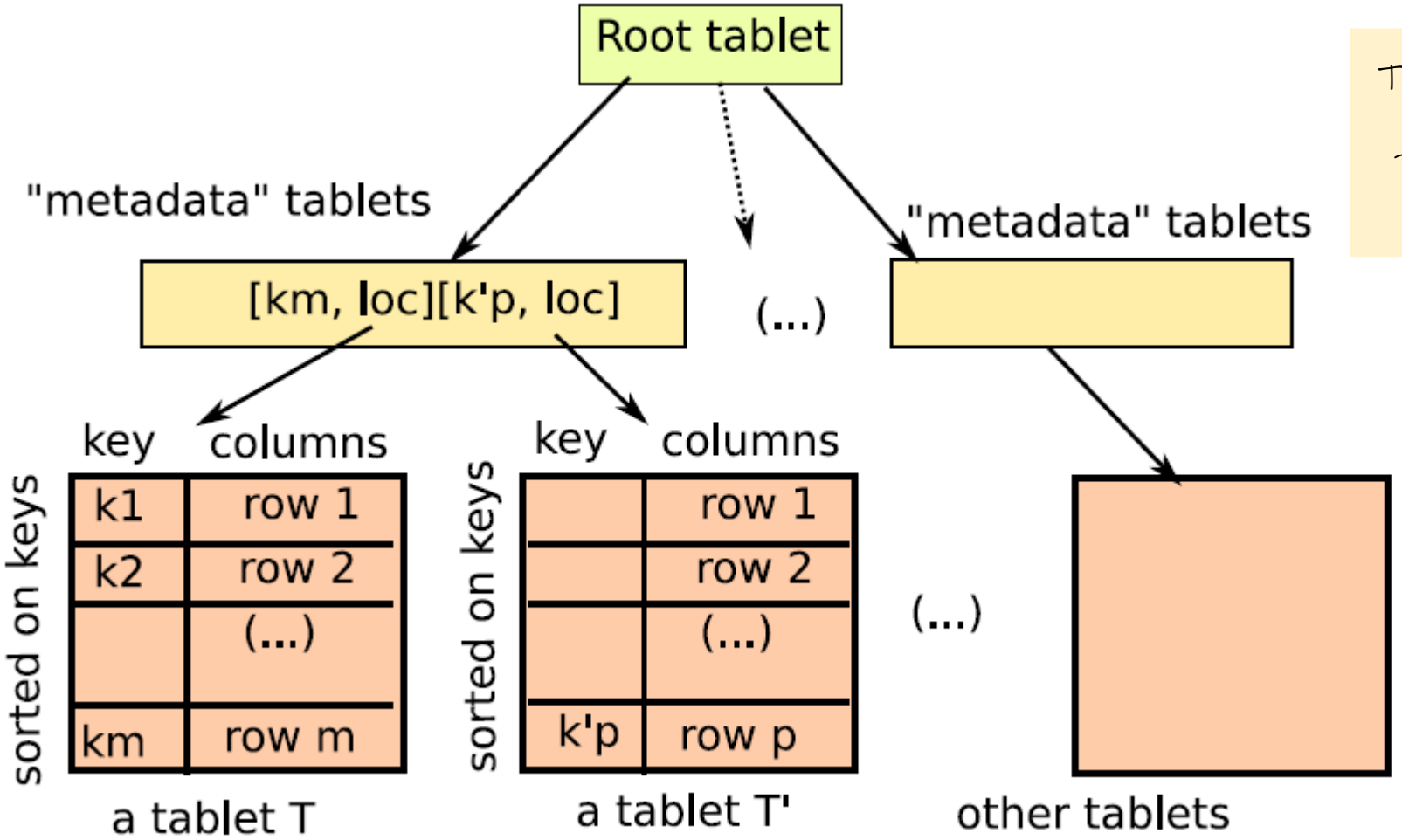


- It is a layer on top of HDFS (HDFS stores data)
- Each table is horizontally fragmented into *Regions (tablets in BigTable)*
  - Dynamic fragmentation
    - By default into few hundreds of MBs (system parameter)
  - Distributed on a cluster of machines or cloud
- At each Region rows are stored column-wise according to families (hybrid fragmentation)
  - In disk, as many files as families
  - Static fragmentation (determined by
  - Block compression can be enabled

Implications on modeling!

- 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 (stores the global 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
  - LSM-tree index: combination of in-memory and disk-based structures

HBase: A Distributed Index Cluster



Tablet (BigTable)
=
Region (HBase)

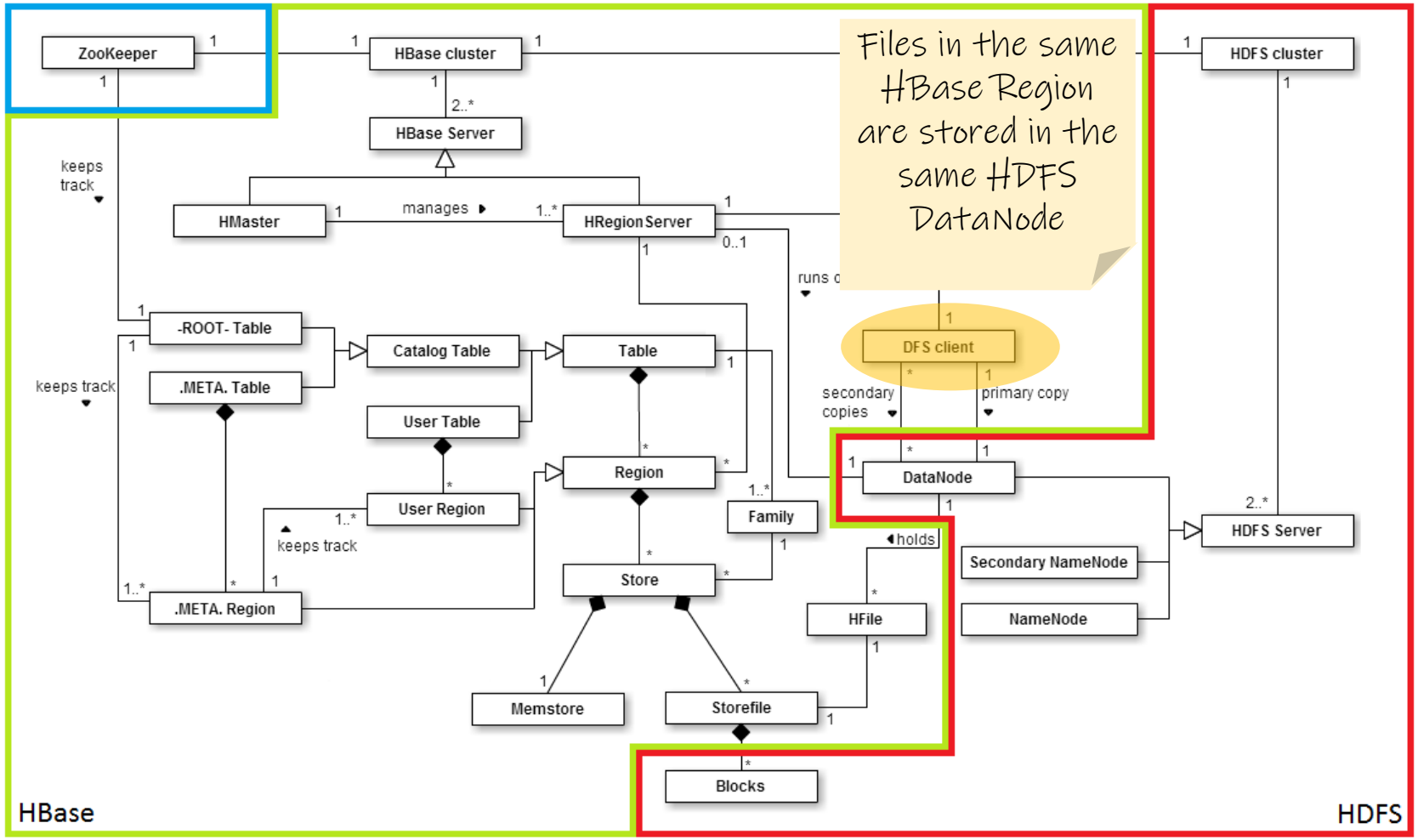
Serge Abiteboul et al.

Functional Components (I)

- Zookeeper
  - Open-source server for highly reliable distributed coordination of cloud applications
  - Stores the HBase cluster config info
- HMaster
  - Coordinates splitting of regions/rows across nodes
  - Controls distribution of HFile chunks
- Region Servers (HRegionServer)
  - Store regions (i.e., horizontal fragments)
    - Within the store region, the horizontal fragment is further fragmented vertically
  - Logs changes
  - Guarantees "atomic" updates
  - Holds (caches) chunks of HFile into Memstores, waiting to be written
- HDFS
  - Stores all data including columns and logs
    - DataNodes store chunks of a file
    - NameNode holds all metadata including namespace
  - 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)
- Clients
  - Read and write chunks
    - Locality & load determine which copy to access

Functional Components (II)

Zookeeper for configuration and global coordination



Files in the same HBase Region are stored in the same HDFS DataNode

- 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.

B n—m A An element of class B is associated with n elements of A, and viceversa

B —▷ A Class B is a specialization (subtype) of class A

B —◆ A Class A is composed by elements of class B

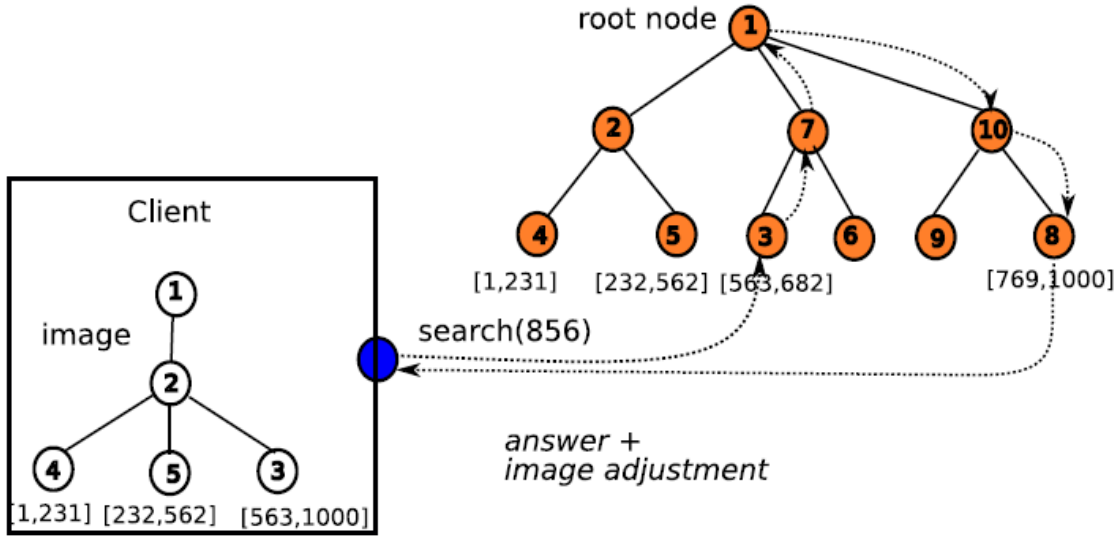
HDFS for storage and replica management

HBase Design Decisions (I)

- One master server
  - Responsible to maintain the table schemas and their fragments (i.e., catalog)
  - Assignment of regions to servers
  - Monitors the cluster (*heartbeating*)
- Many region servers
  - Each handling around 100-1000 regions (i.e., horizontal fragments)
  - Further apply vertical fragmentation (families) to regions
  - 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

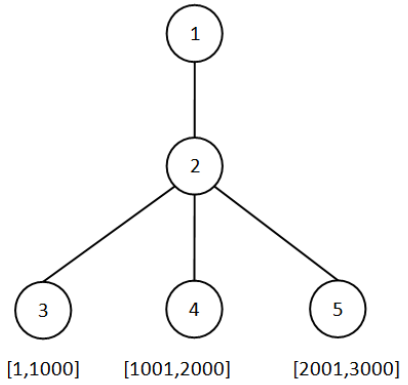
HBase Design Decisions (II)

- 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

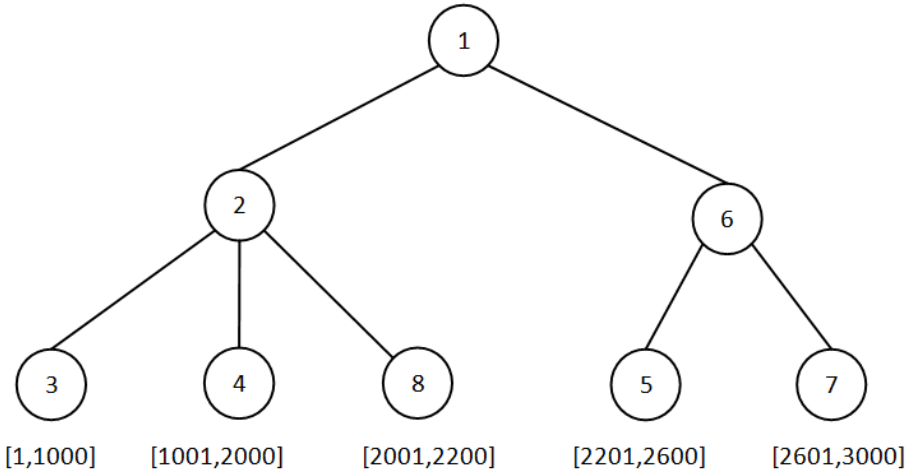


Activity: Mistake Compensation

- *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 Get(2602)*
     b) *Number of round trips if Get(2602) again*
  2. *(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 (i.e., heavy in-memory processing)
    - Merge-sort for merging it with *old* data
    - LSM-tree: a distributed B+ able to deal with in-memory and disk data
  - Richer key-value data structure: Table / family / row
    - Hybrid fragmentation (first horizontal, then vertical)
    - Thus, it cannot fully benefit from column-oriented processing
  - Concurrency: Timestamping MVCC (Multiversion Concurrency Control))
    - Guarantees ACID semantics per-row (even across families)
  - Recovery: Logging is used at the Region Servers
    - WAL (write-ahead protocol)
    - REDO login (No Force / No Steal)

HBase Architecture

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

Horizontal fragmentation (fixed-size chunks)
Secondary vertical fragmentation (not allowing HBase to benefit from column-oriented processing)
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 the catalog copies

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

Not covered by HBase but done by MapReduce / Spark
HBase API only covers single-key or range key searches

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

Concurrency: per-row guarantees only
Recovery: checkpointing logging per Region Server

  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]

Summary

- The Hadoop Ecosystem: Storage Layer
  - HDFS, HBase
- The File System (HDFS)
  - Architecture: functional components
  - File formats
- The Database (HBase)
  - Schema elements
  - Data model:
    - Data structure: Key-value
    - Operations: GET, SCAN
    - Constraints: NONE!
  - Architecture: functional components
  - Design decisions

Bibliography

F. Chang et all. *Bigtable: A Distributed Storage System for Structured Data*. OSDI'06

Sanjay Ghemawat et al. *The Google File System*. SOSP'03

D. Jiang et al. *The performance of MapReduce: An In-depth Study*. VLDB'10