Key-Value Stores

Very Large Databases

HDFS, HBase and MapReduce

An Example of Key-Value Ecosystem

Key-Values: A Piece of History

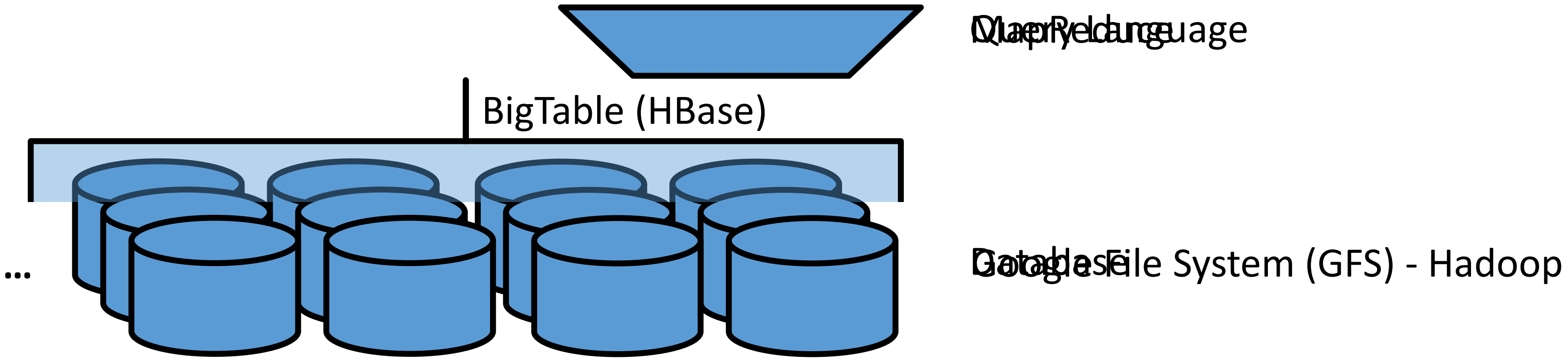
Key-values were born as a desperate answer to the RDBMS limitations

It is widely assumed that Google is the father of Key-value stores

- 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

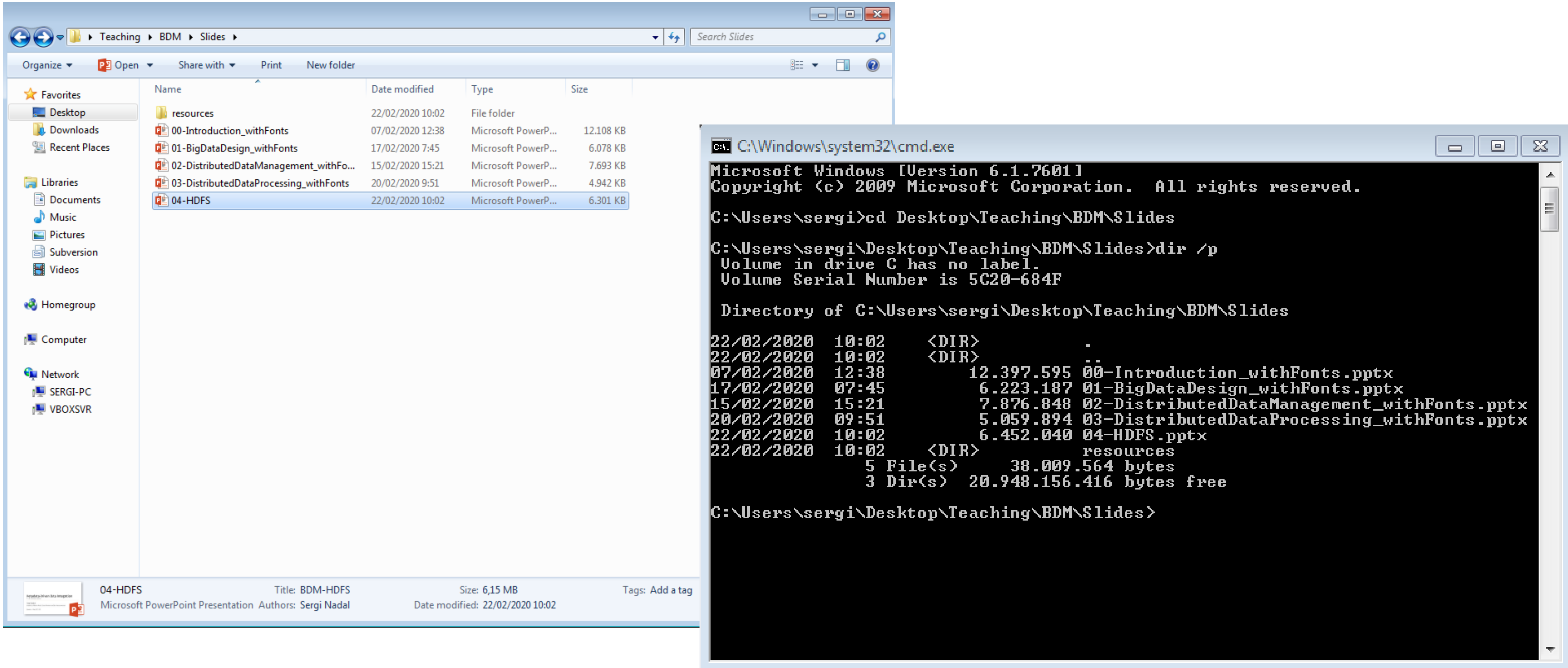
- High-performance is mainly achieved by means of parallelism
- To achieve parallelism it distributes data across the cluster
  - Google File System (GFS)
  - GFS + HBase
- MapReduce
  - It is a query language that provides parallelism in a transparent manner



(Distributed) File Systems

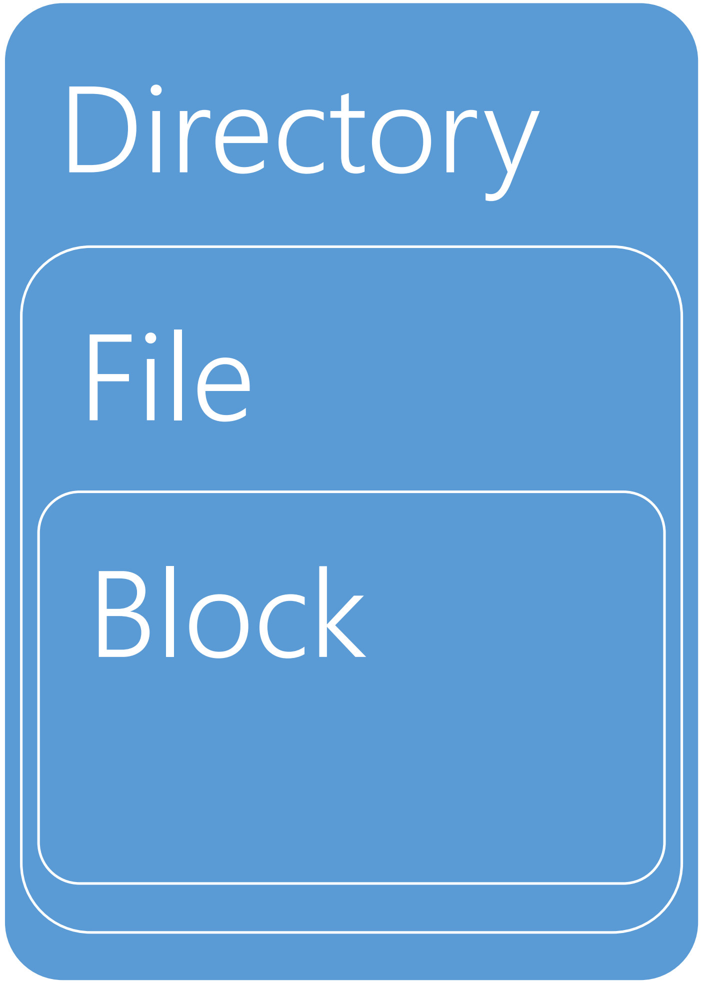
Google File System

What is a file system?



Functionalities provided by a FS

- 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, …



Distributed File Systems

- Same requirements, different setting
  1. Files are huge for traditional standards
  2. Most files are updated by appending data rather than overwriting
     - Write Once and Read Many times (WORM)
  3. Component failures are the norm rather than the exception
- Google File System (GFS)
  - The first large-scale distributed file system
  - Capacity of a GFS cluster

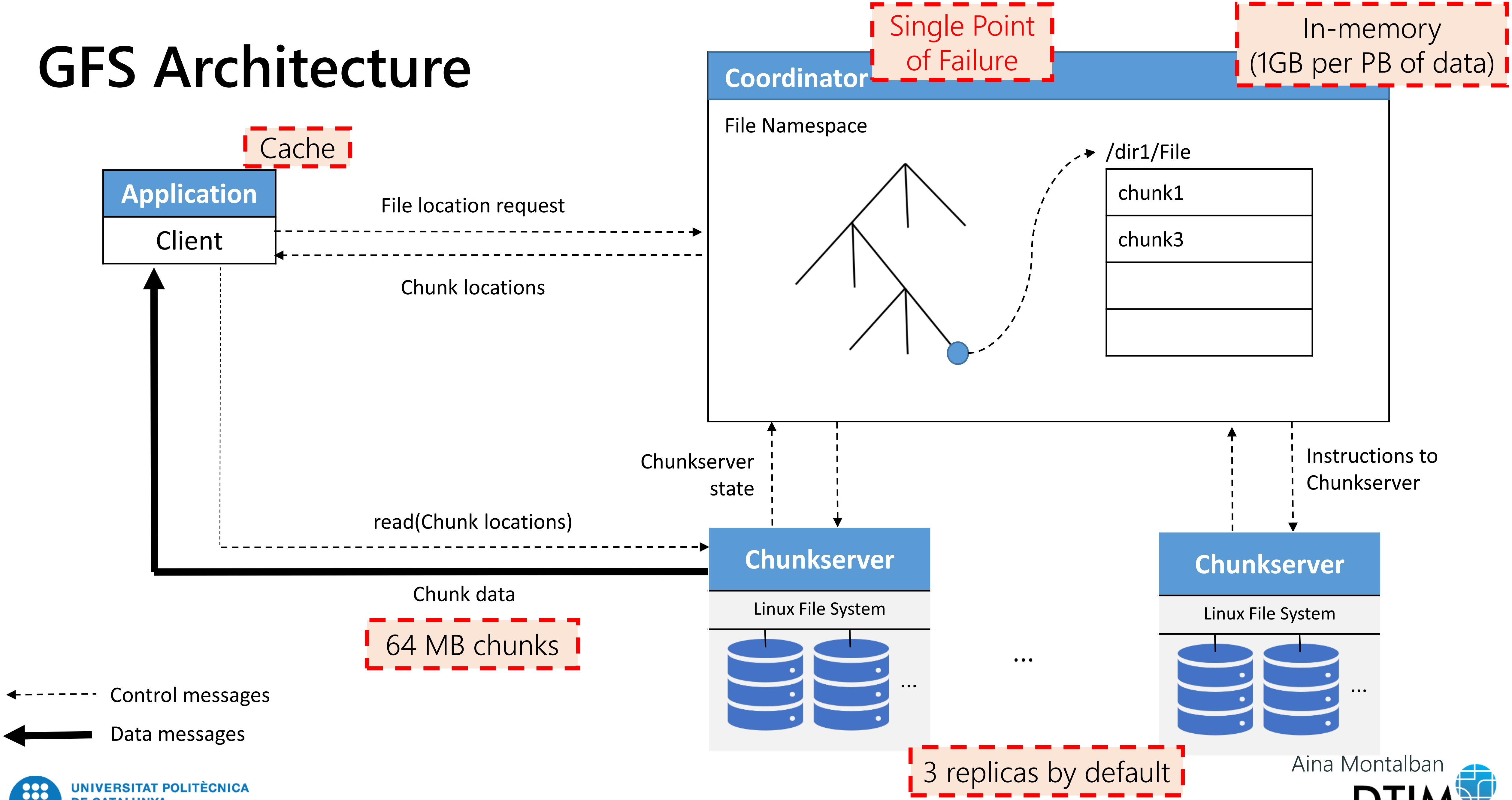
Capacity	Nodes	Clients	Files
10 PB	10.000	100.000	100.000.000

Design goals of GFS

- Efficient management of files
  - Optimized for very large files (GBs to TBs)
- Efficiently append data to the end of files
  - Allow concurrency
- Tolerance to failures
  - Clusters are composed of many inexpensive machines that fail often
    - Failure probability (2-3/1.000 per day)
- Sequential scans optimized
  - Overcome high latency of HDDs (5-15ms) compared to main memory (50-150ns)

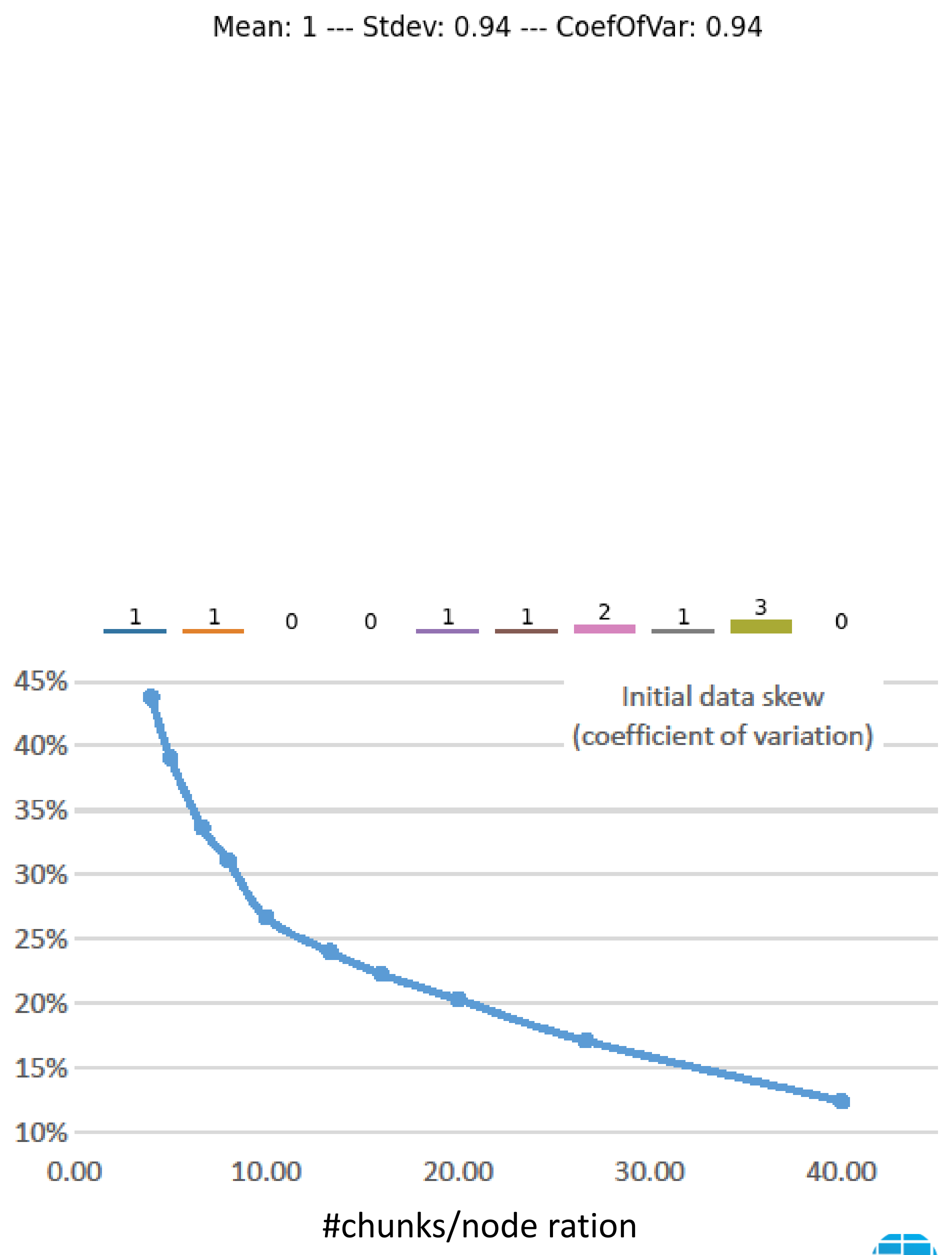
GFS Architecture

GFS Architecture



Other features

- Rebalance
  - Avoids skewness in the distribution of chunks
- Deletion
  - Moves a file to the trash (hidden)
    - Kept for 6h
  - *expunge* to force the trash to be emptied
- Management of stale replicas
  - Coordinator maintains versioning information about all chunks

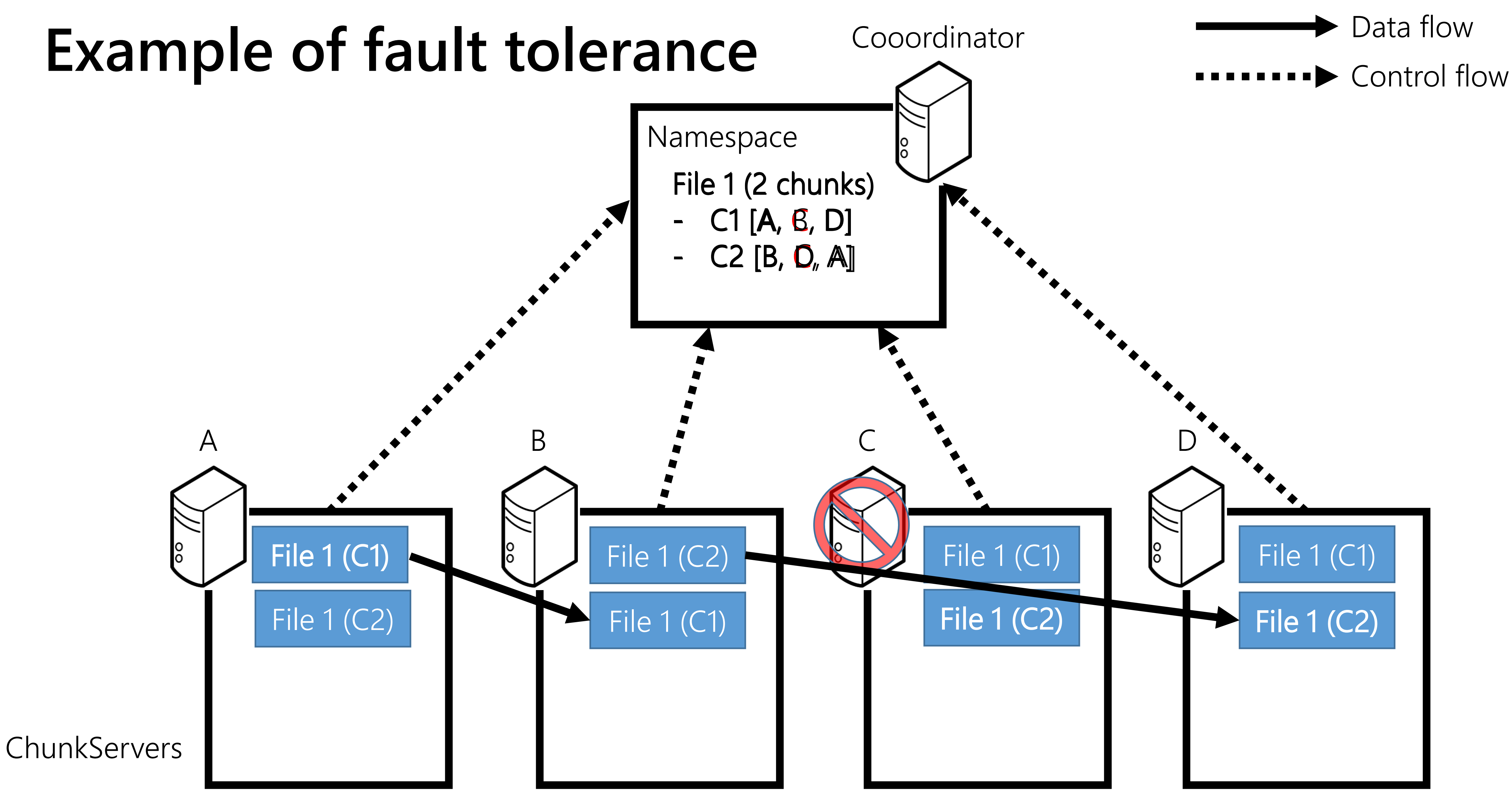


Fault Tolerance

Fault tolerance

- Managed from the coordinator
    - It expects to receive every 3 seconds a *heartbeat* message from chunkservers
- Chunkserver not sending a heartbeat for 60 seconds, a fault is declared
- Corrective actions
    - Update the namespace
    - Copy one of the replicas to a new chunkserver
        - Potentially electing a new primary replica

Example of fault tolerance



Reading Files

Client caching

Cache miss

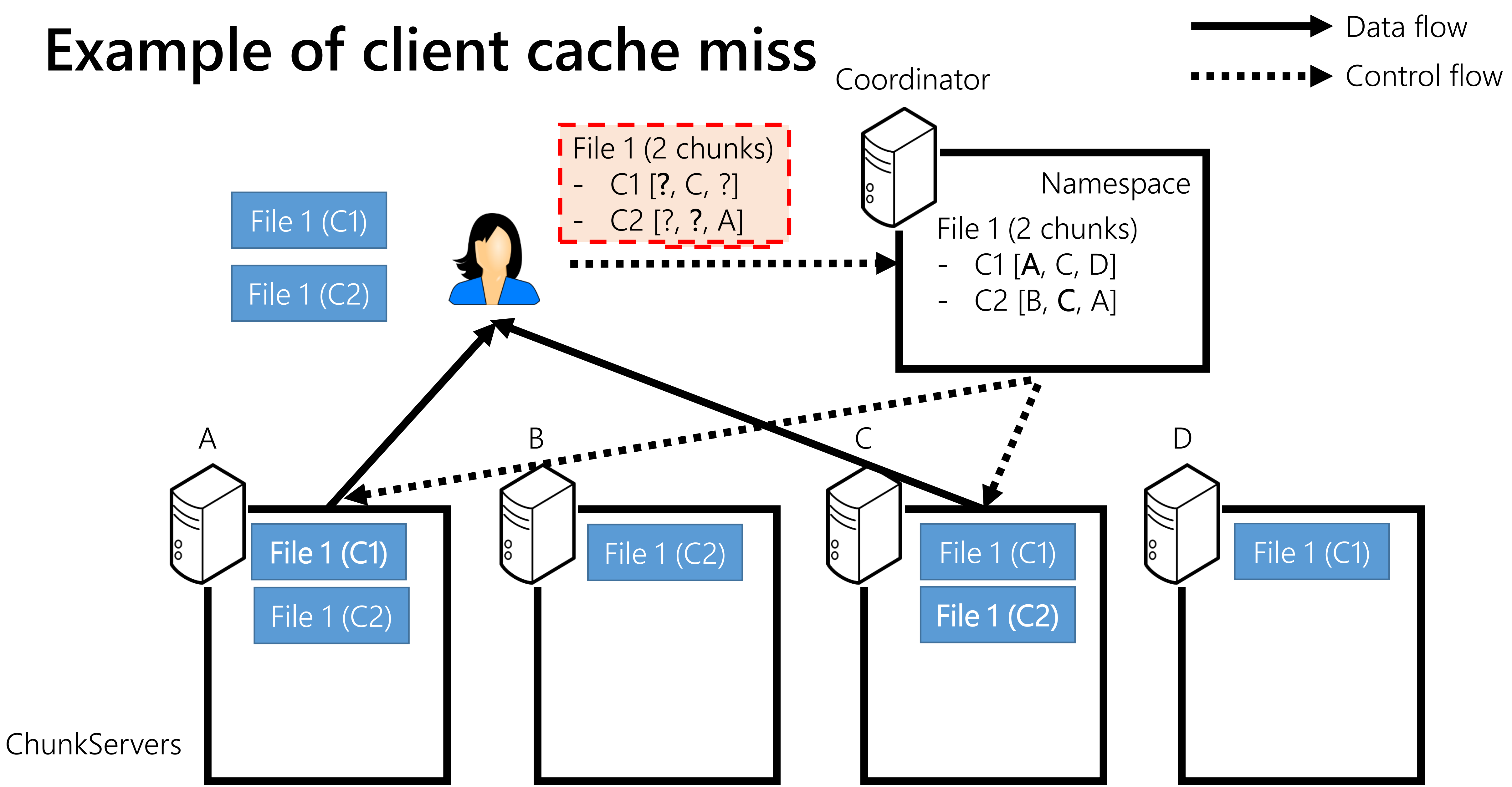
1. The client sends a READ command to the coordinator
2. The coordinator requests chunkservers to send the chunks to the client
   - Ranked according to the closeness in the network
3. The list of locations is cached in the client
   - Not a complete view of all chunks

Cache hit

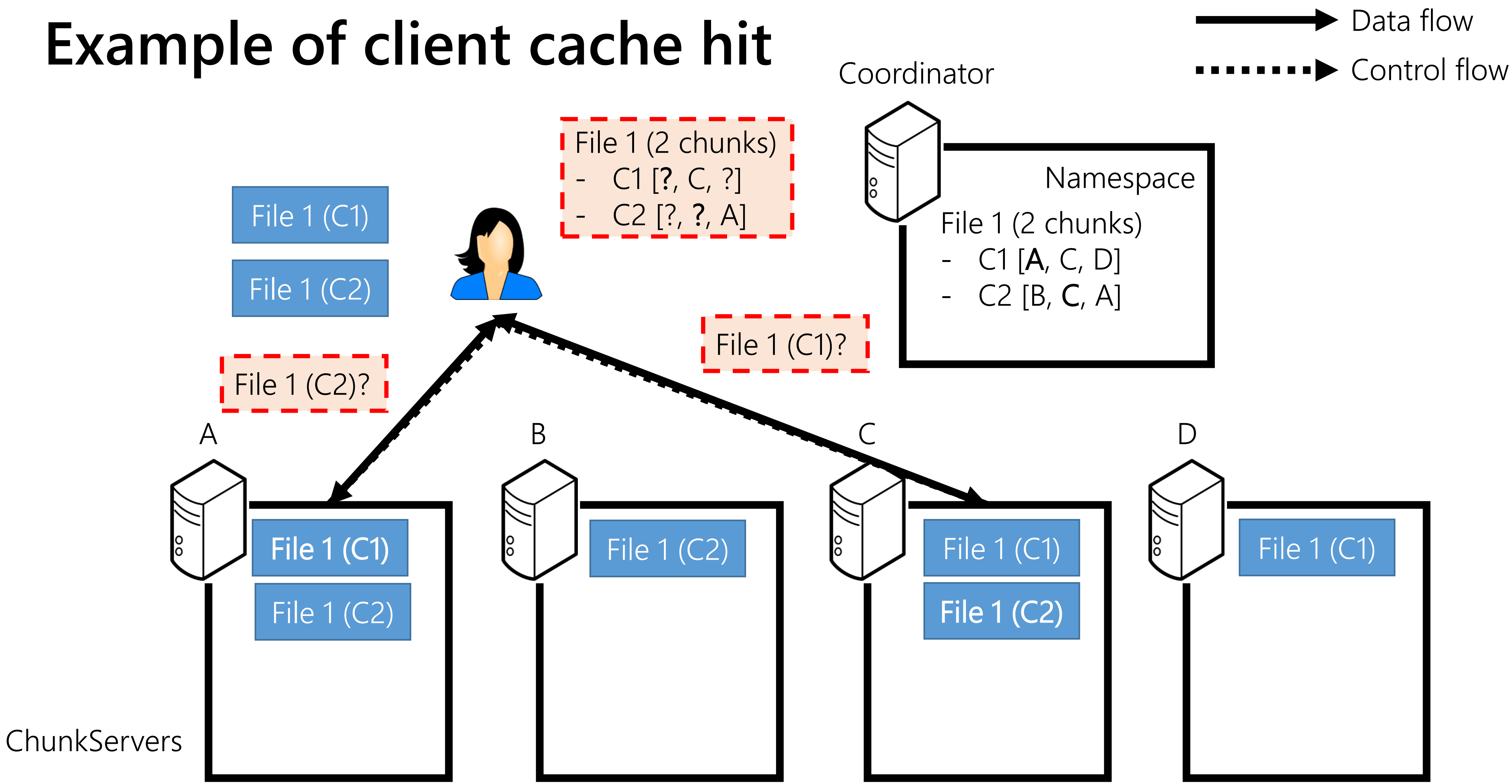
1. The client reads the cache and requests the chunkservers to send the chunks

Avoid coordinator bottleneck
+
One communication step is saved

Example of client cache miss



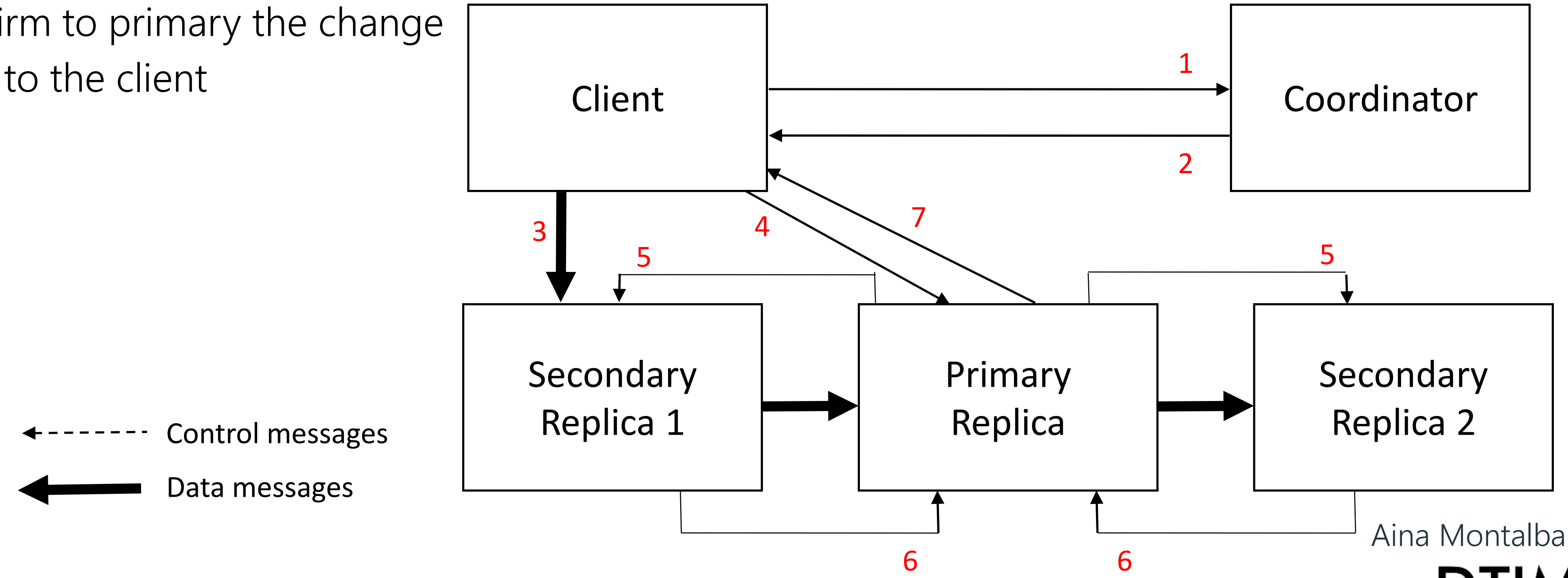
Example of client cache hit



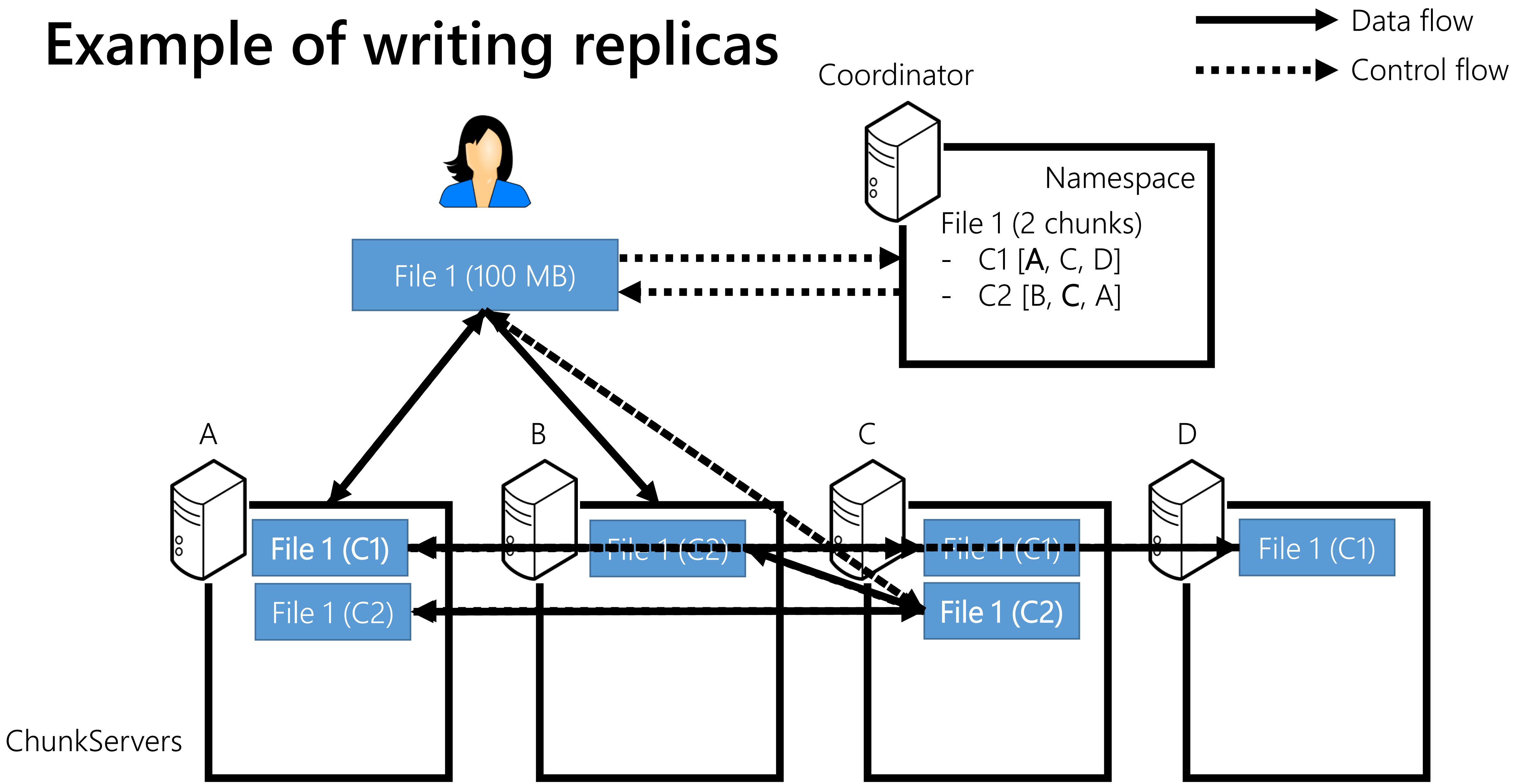
Writing Files

Writing replicas

1. The client requests the list of the replicas of a file
2. Coordinator returns metadata
3. Client sends a chunk to the closest chunkserver in the network
   - This chunk is pipelined to the other chunkservers in the order defined by the master (leases)
4. Client sends WRITE command to primary replica
5. Primary replica sends WRITE command to secondary replicas
6. Secondaries confirm to primary the change
7. Primary confirms to the client



Example of writing replicas



The Database: HBase

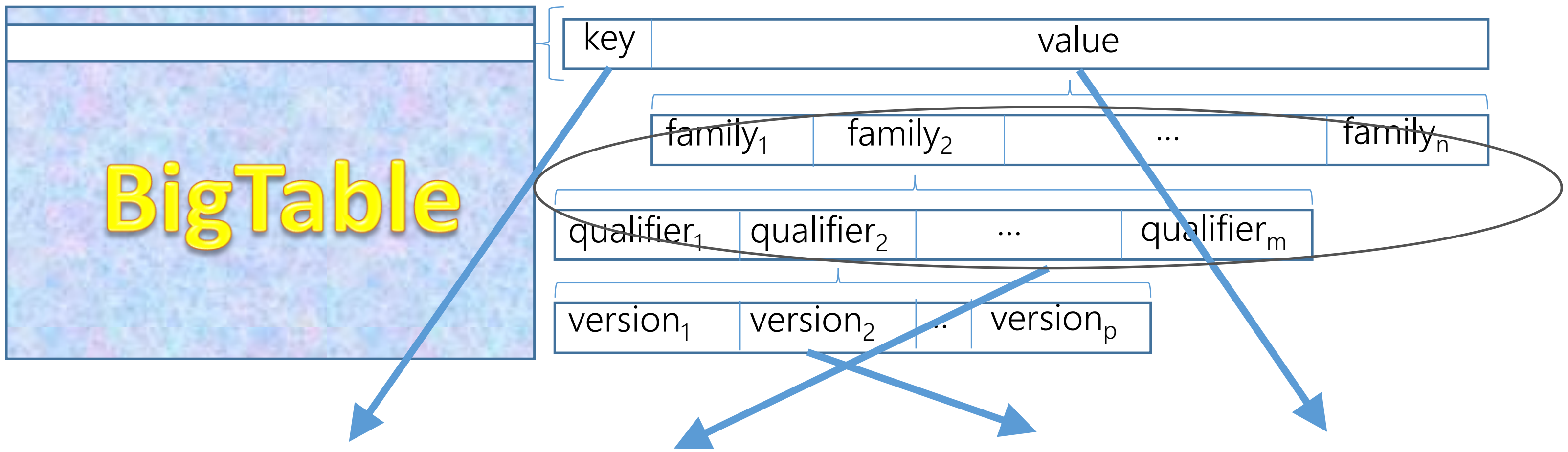
Richer Data Structure, Recovery, Concurrency and Indexing Capabilities

HBase

- Apache project
  - Based on Google's Bigtable
- Designed to meet the following requirements
  - Access specific data out of petabytes of data
  - It must support
    - Key search
    - Range search
    - High throughput file scans
  - It must support single row transactions

BigTable 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

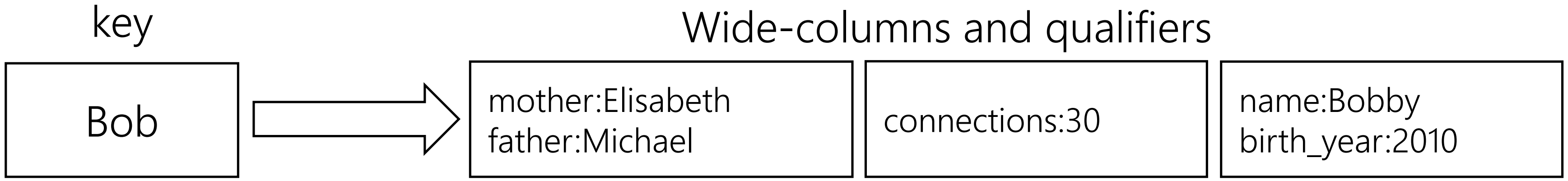


(row:string, column:string[, time:int64])→string

Key-Value

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

- Bigtable (Wide-column key-value stores)
  - Entries in form of key-values
    - But now values are split in wide-columns
  - Typically query on key
    - May have some support for values
  - Schemaless within a wide-column



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*

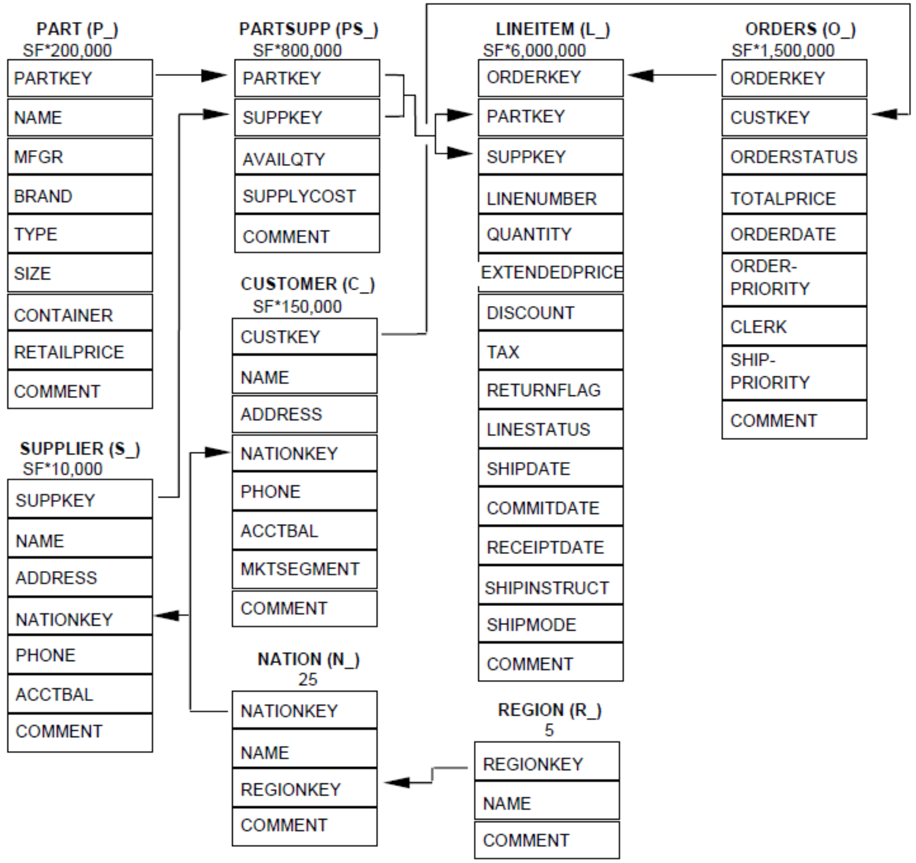
SELECT l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue, o_orderdate, o_shippriority

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

HBase Architecture

HBase shell

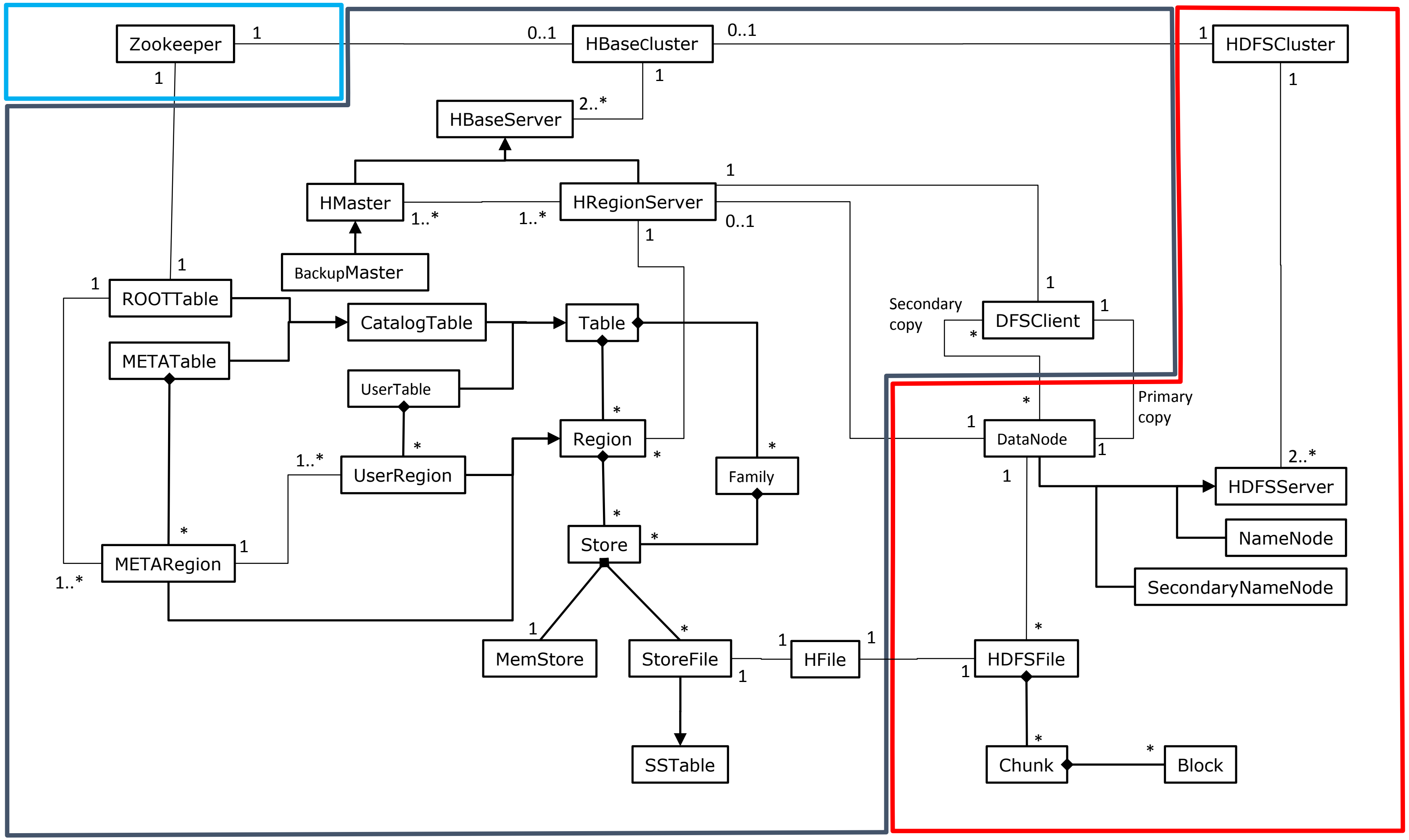
- ALTER <tablename>, <columnfamilyparam>
- COUNT <tablename>
- CREATE TABLE <tablename>
- DESCRIBE <tablename>
- DELETE <tablename>, <rowkey>[, <columns>]
- DISABLE <tablename>
- DROP < tablename>
- ENABLE <tablename>
- EXIT
- EXISTS <tablename>
- GET <tablename>, <rowkey>[, <columns>]
- LIST
- PUT 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

https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands

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
- HRegionServer - Region Servers
  - Serves HBase client requests
    - Manage stores containing all column families of the region
  - Logs changes
  - Guarantees “atomic” updates to one column family
- HFiles - Consist of large (e.g., 128MB) chunks
- 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)
      - 3 copies of one chunk for availability (default)
    - HLog: region’s log file (allows flush/fsync for small append-style writes)

Functional components (II)

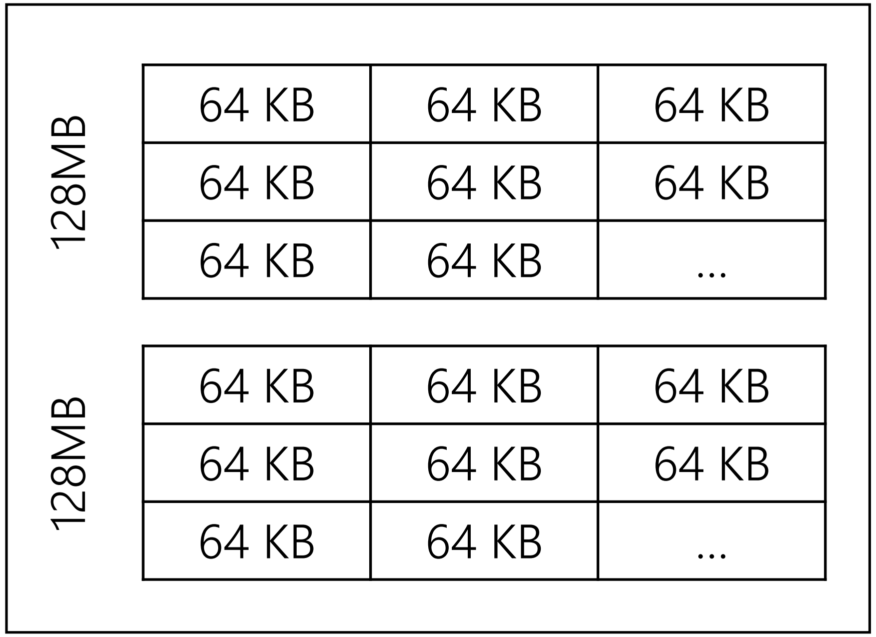


Association
Aggregation
Specialization

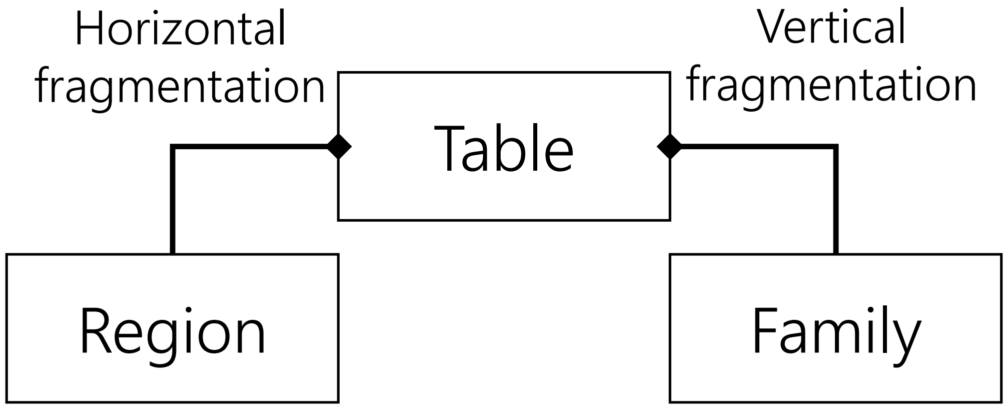
StoreFiles

- When the MemStore is full (128MB), data are flushed to HDFS
- A StoreFile is generated
  - Format HFile
- An HFile stores data into HDFS chunks
  - Chunks are structured into Hbase blocks
    - Size 64 KB

Storefile
(HFile format)



Logical structure



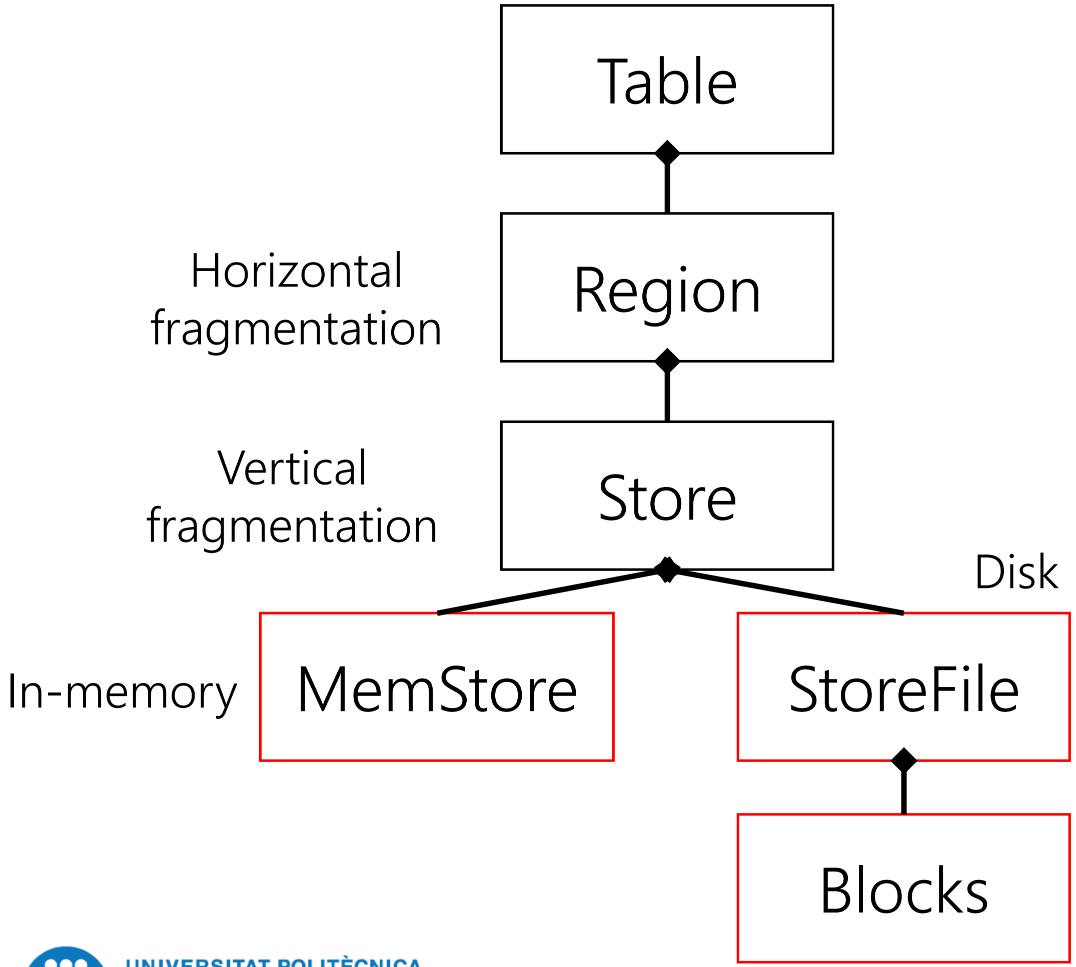
Region 1

	family A		family B
	qualifier a	qualifier b	qualifier c
Key 1	value	value	value
Key 2	value	value	value

Region 2

	family A		family B
	qualifier a	qualifier b	qualifier c
Key 3	value	value	value
Key 4	value	value	value

Physical structure



Region 1

	Store 1.1		Store 1.2
	family A		family B
	qualifier a	qualifier b	qualifier c
Key 1	value	value	value
Key 2	value	value	value

Region 2

	Store 2.1		Store 2.2
	family A		family B
	qualifier a	qualifier b	qualifier c
Key 3	value	value	value
Key 4	value	value	value

HBase Processes

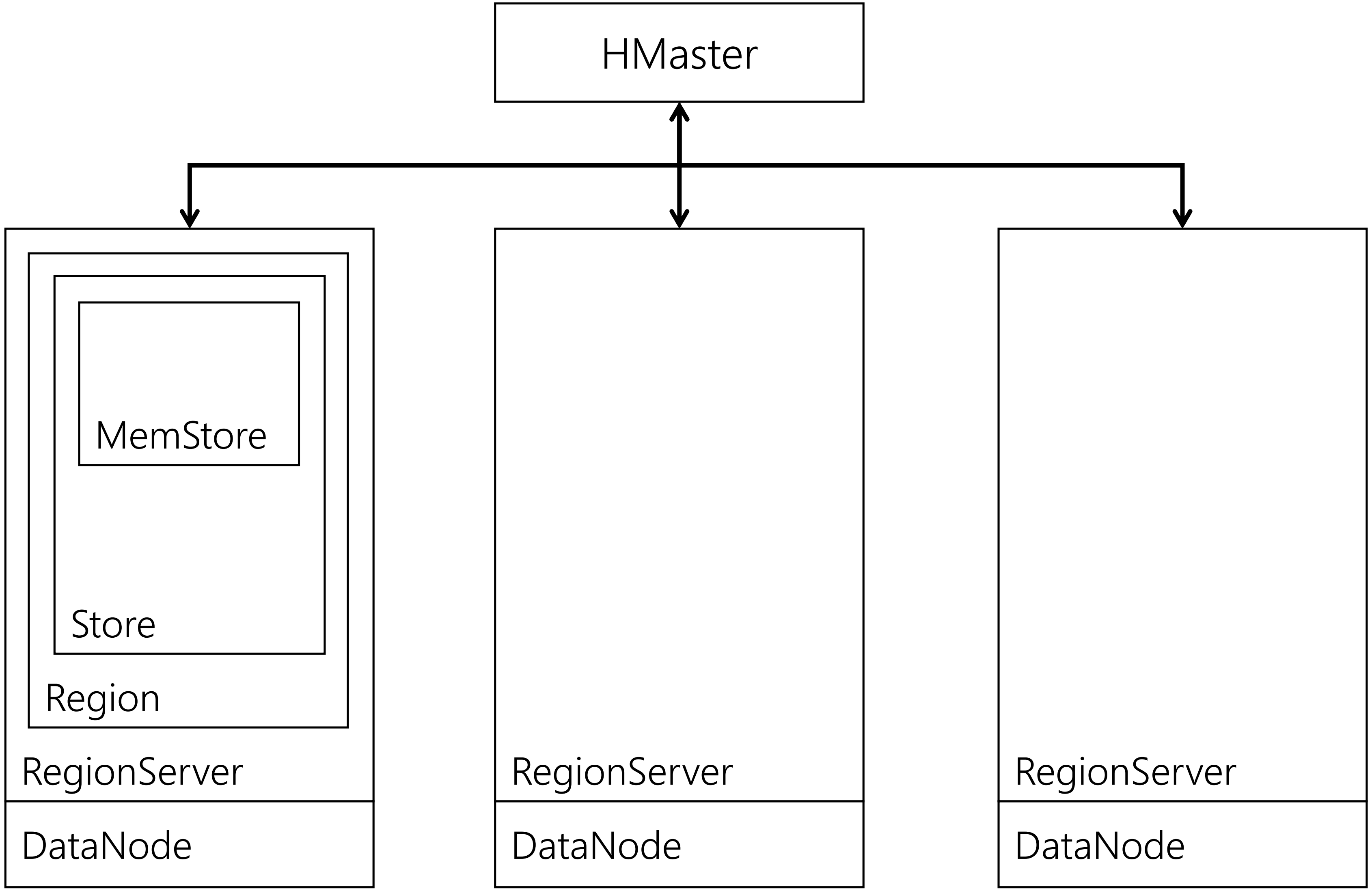
HBase processes

a) Flush
   - On memory structure reaching threshold
   - Takes memory content and store it into an SSTable
   - Generates different disk versions of the same record

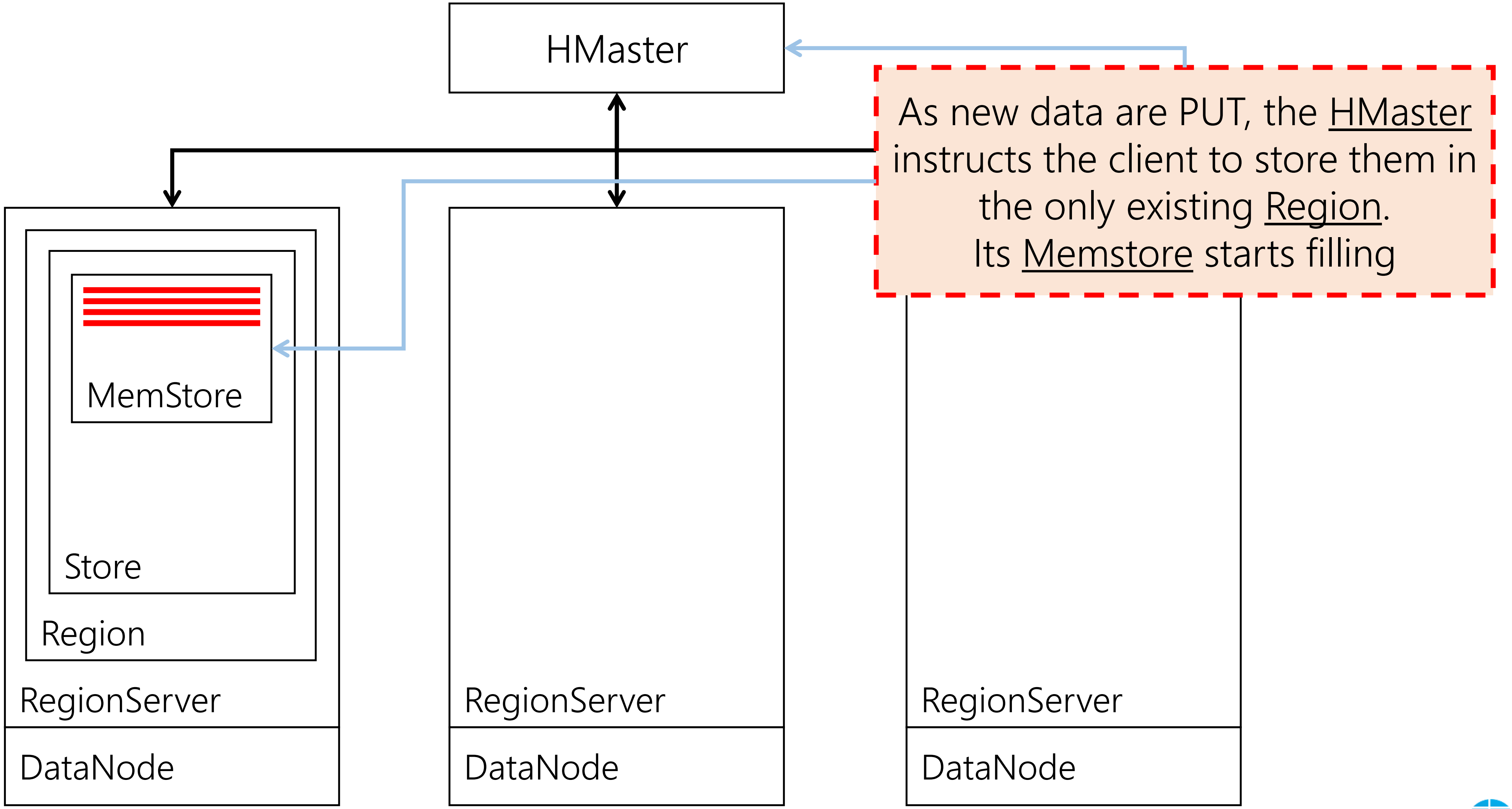
b) Minor compactation
   - Runs regularly in the background
   - Merges a given number of equal size SSTables into one
   - Does not remove all record versions (only some)

c) Major compactation
   - Triggered manually
   - Merges all SSTables
   - Leaves one single SSTable
   - All versions of a record are merged into one

Example of Flush

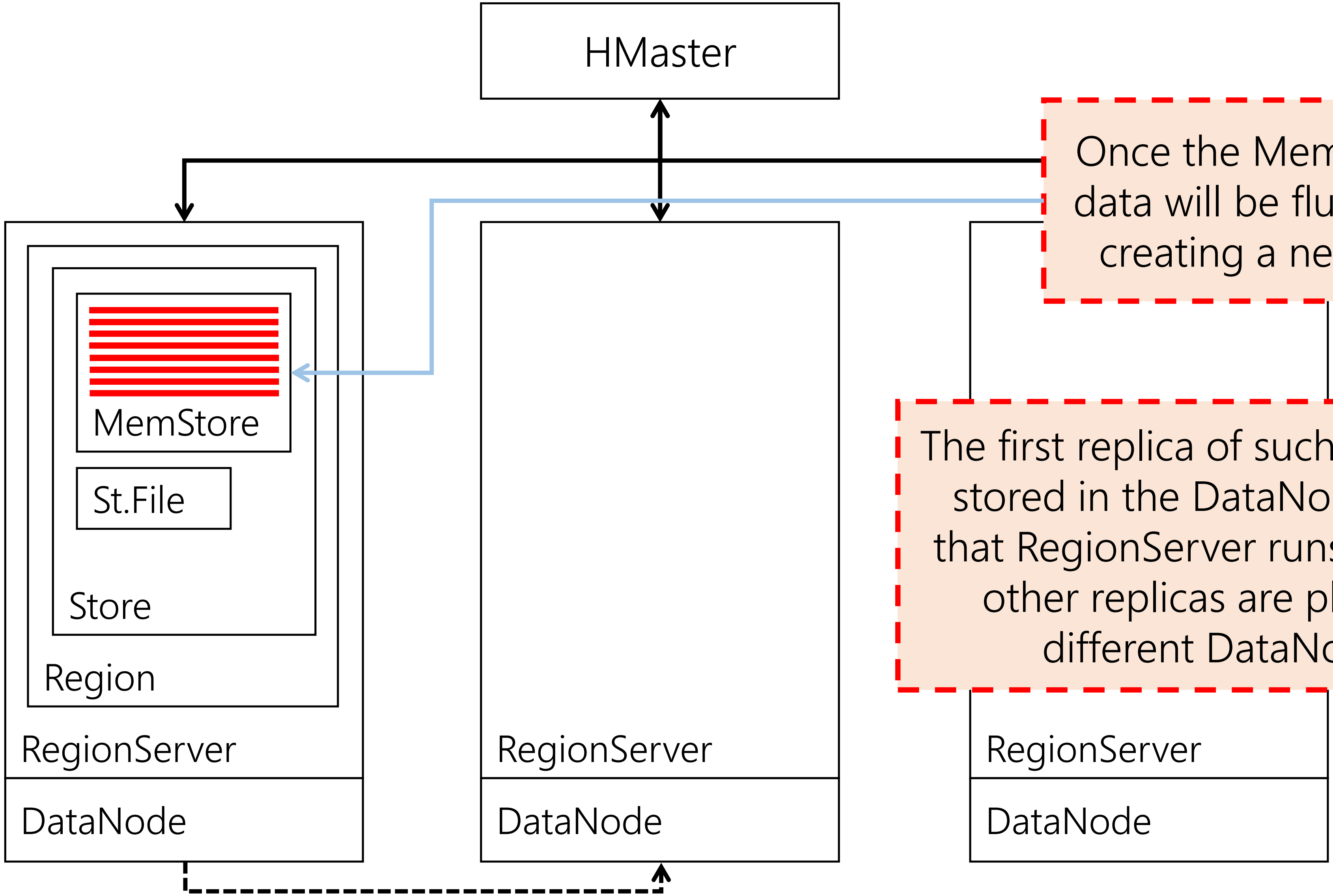


Example of Flush



As new data are PUT, the HMaster instructs the client to store them in the only existing Region.
Its Memstore starts filling

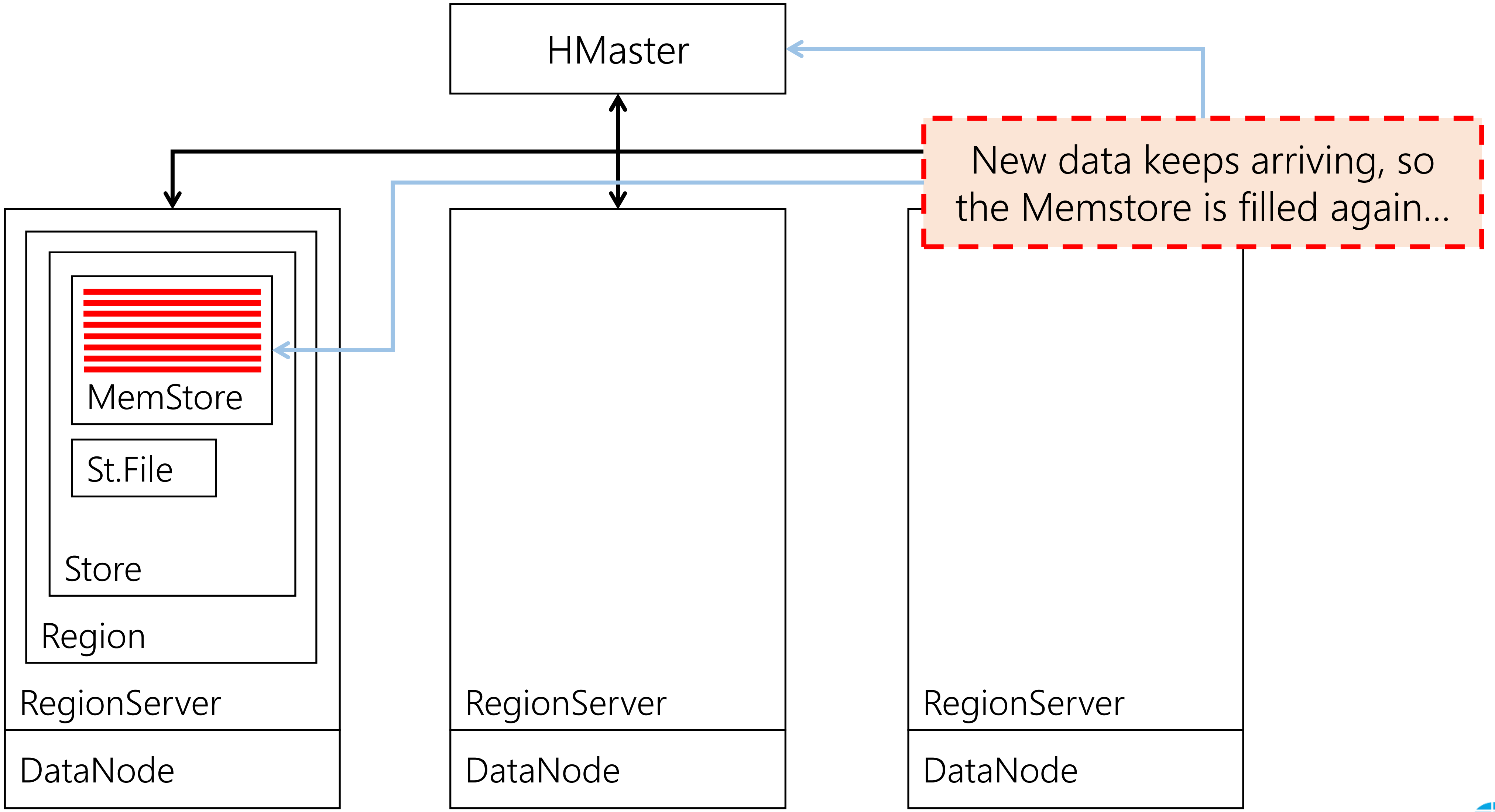
Example of Flush



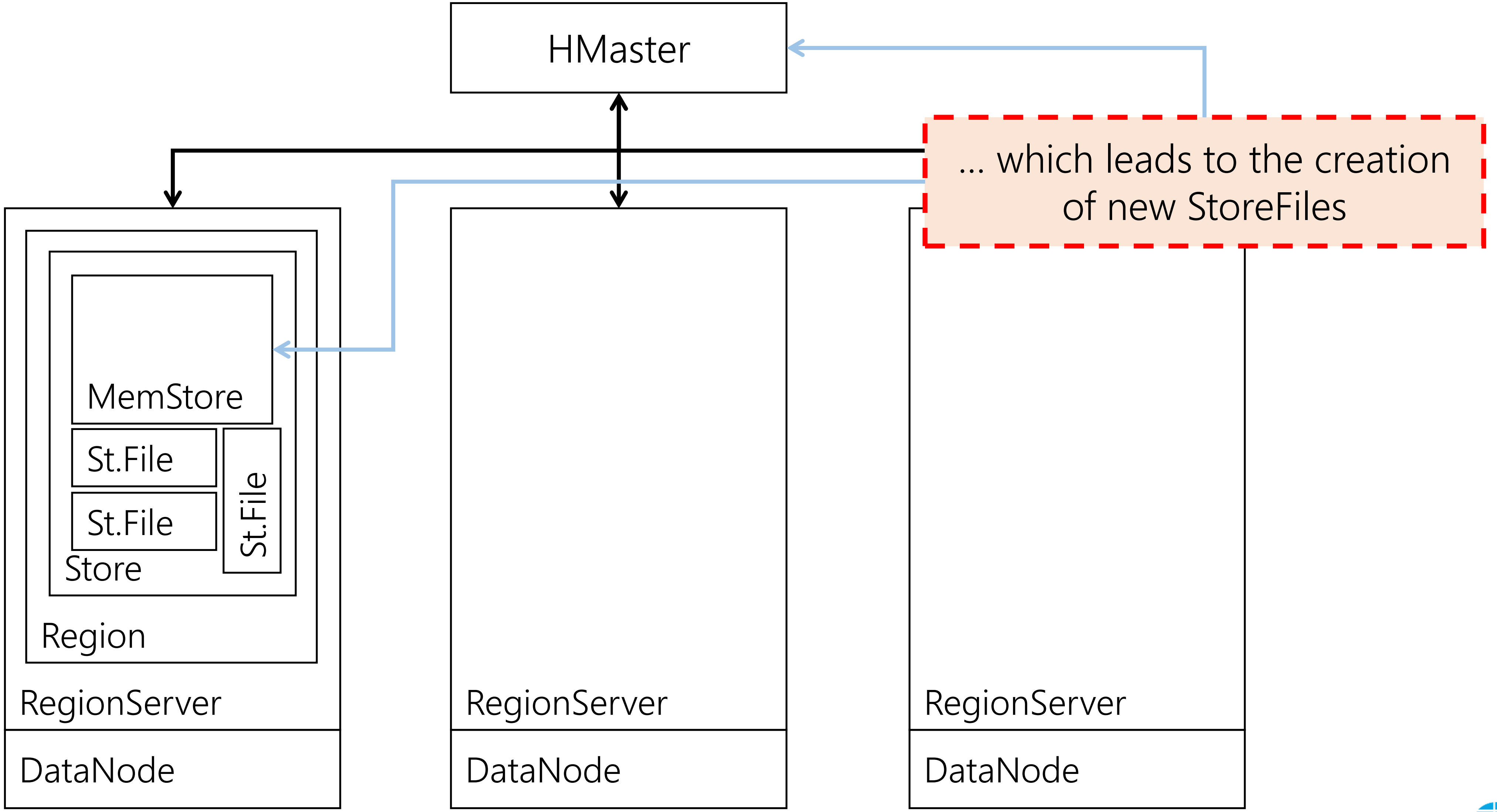
Once the MemStore is full, data will be flushed to disk creating a new StoreFile

The first replica of such Storefile is stored in the DataNode where that RegionServer runs on, while other replicas are placed in different DataNodes

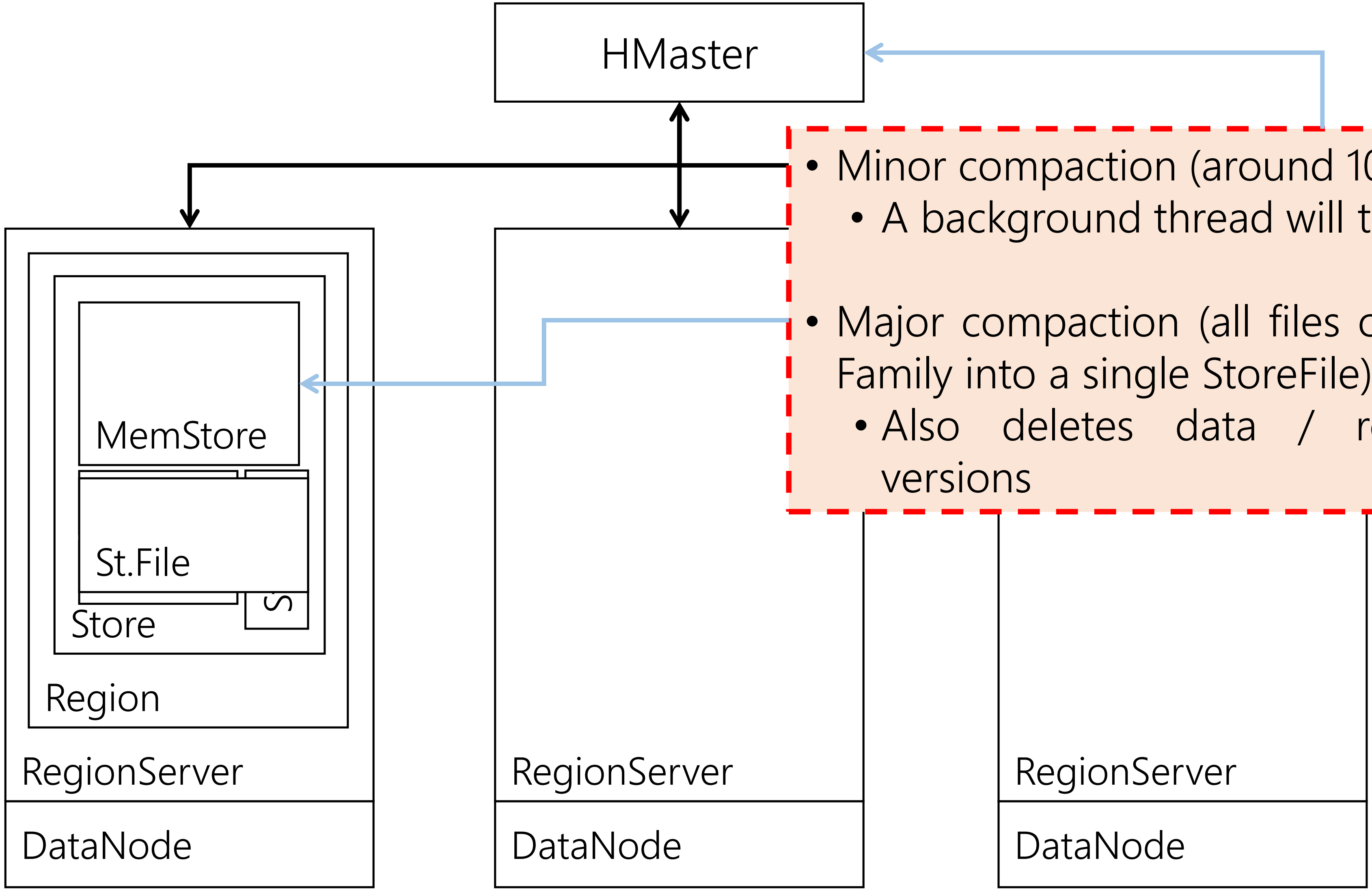
Example of Compactation



Example of Compactation

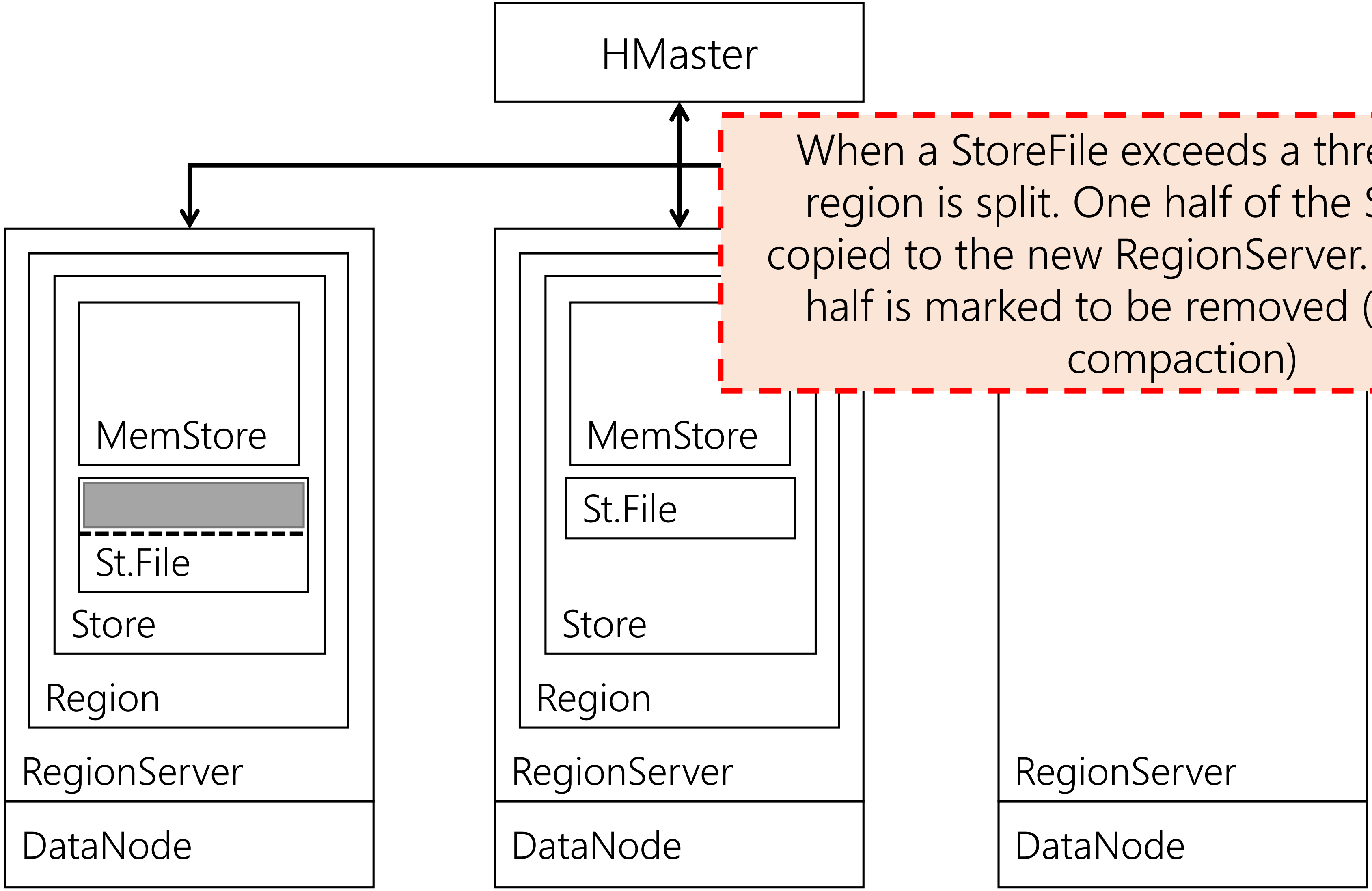


Example of Compactation



- Minor compaction (around 10 files)
  - A background thread will trigger it
- Major compaction (all files of a Column-Family into a single StoreFile)
  - Also deletes data / removes old versions

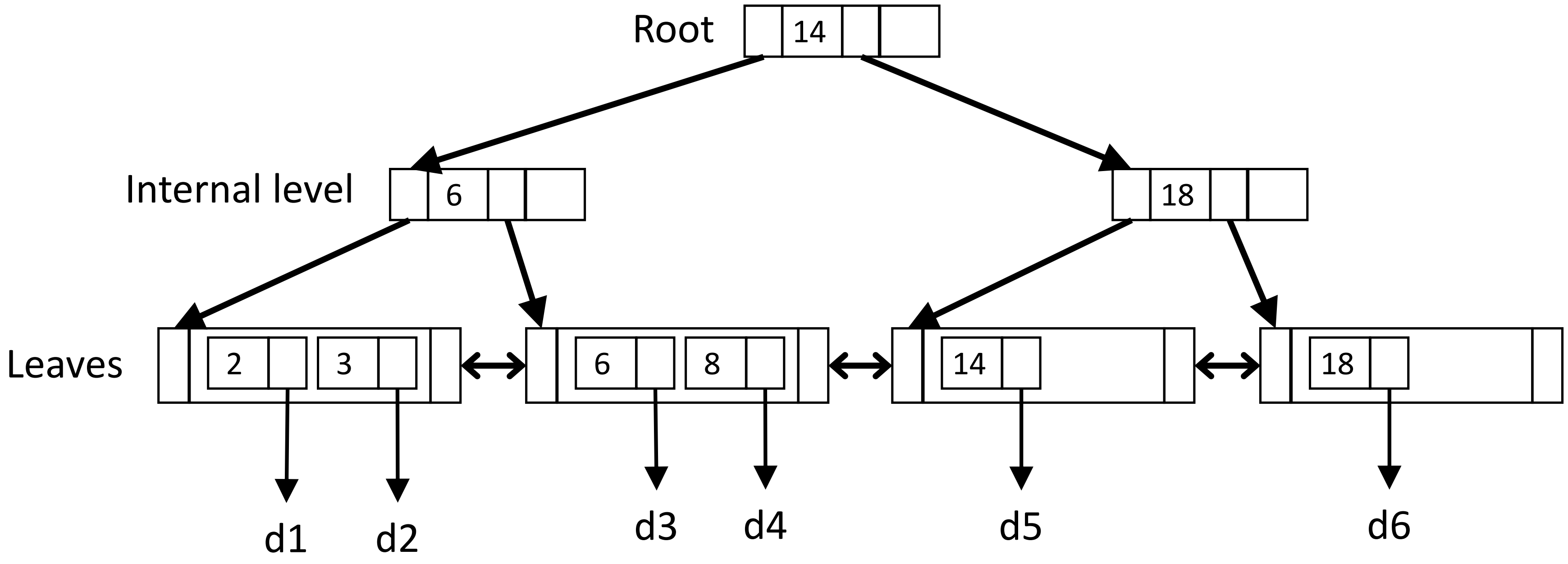
Example of Split



When a StoreFile exceeds a threshold, the region is split. One half of the StoreFile is copied to the new RegionServer. The original half is marked to be removed (in a major compaction)

HBase Distributed Catalog

Metadata hierarchical structure

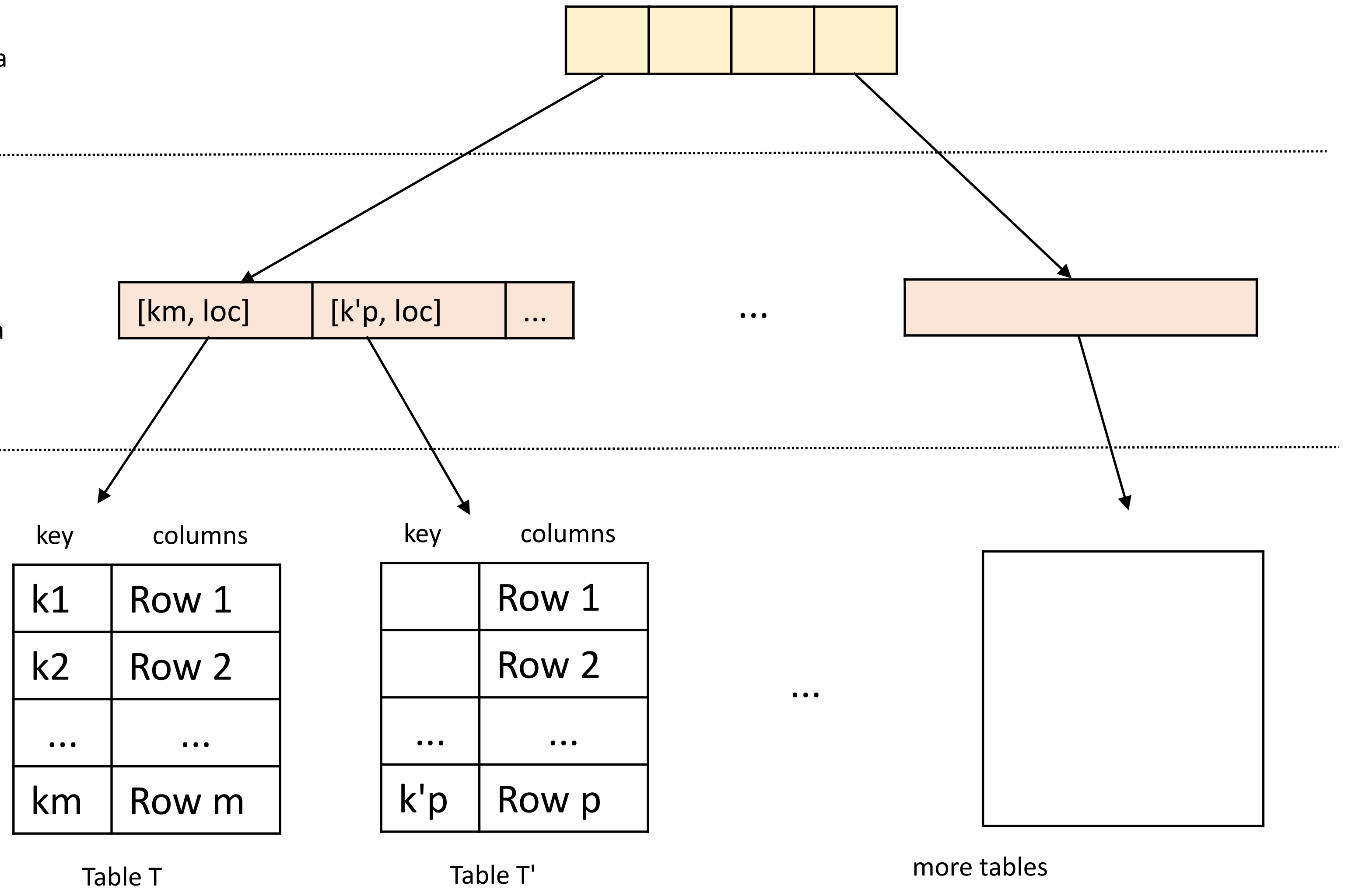


Metadata hierarchical structure

Root table
Store locations of metadata tables

Metadata tables
Store locations of user data tables

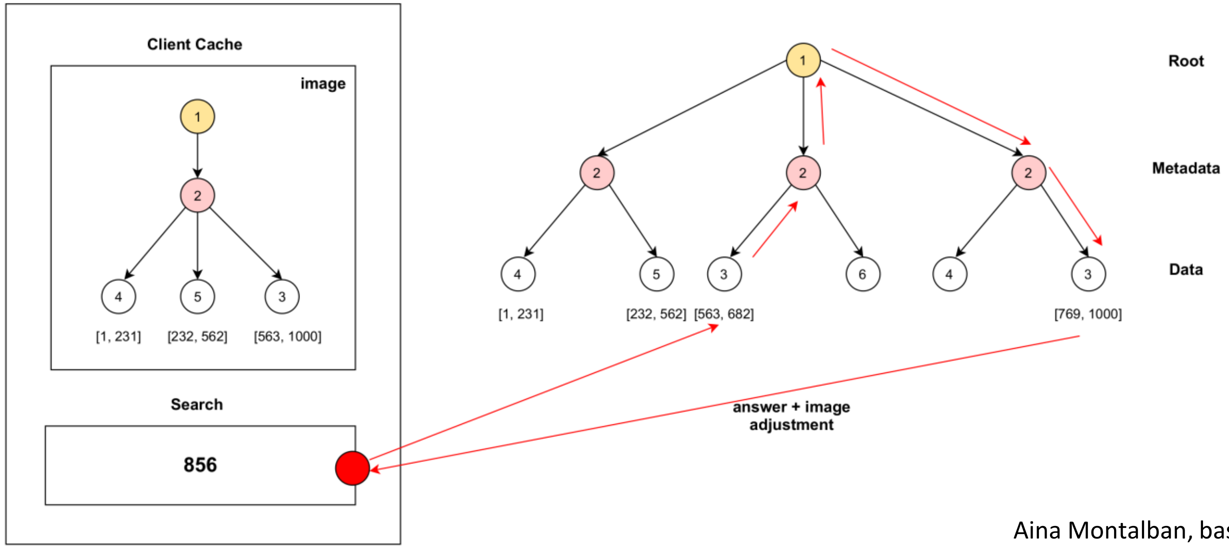
User data tables
Store data



Aina Montalban, based on S. Abiteboul et al.

Metadata synchronization in HBase

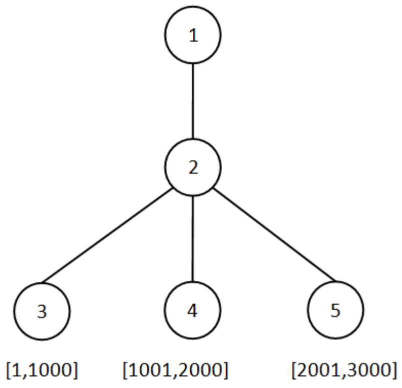
- Split and merge invalidate the cached metadata
  a) Gossiping
  b) Lazy updates
    - Discrepancies may cause out-of-range errors, which trigger a stabilization protocol (i.e., **mistake compensation**)
      - Apply forwarding path
        - If an out-of-range error is triggered, it is forwarded upward
        - In the worst case (i.e., reaching the root), 6 network round trips



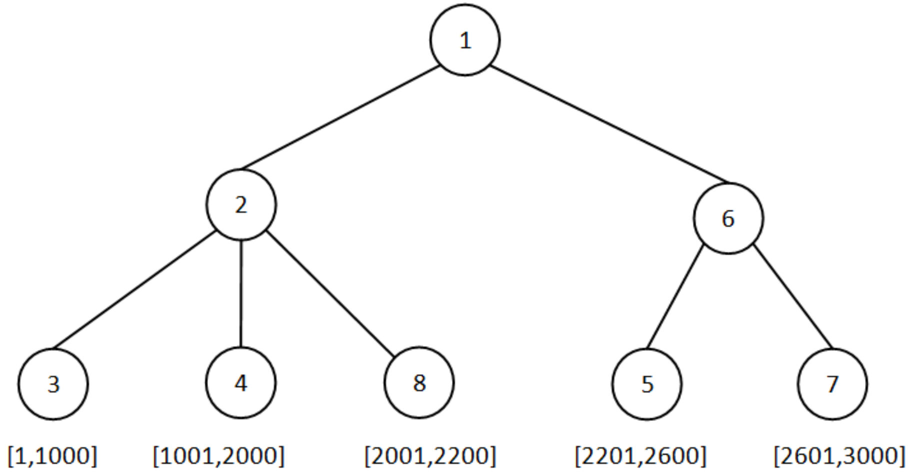
Aina Montalban, based on S. Abiteboul et al.

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 Design Goals

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

HBase:
Concurrency: column level, multiversion timestamping
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: CP]

References (I)

- S. Ghemawat et al. *The Google File System*. OSDI'03
- K. V. Shvachko. *HDFS scalability: the limits to growth*. 2010
- S. Abiteboul et al. *Web data management*. Cambridge University Press, 2011
- A. Jindal et al. *Trojan data layouts: right shoes for a running elephant*. SOCC, 2011
- F. Färber et al. *SAP HANA database: data management for modern business applications*. SIGMOD, 2011
- V. Raman et al. *DB2 with BLU Acceleration: So Much More than Just a Column Store*. VLDB, 2013
- D. Abadi, et al. *Column-stores vs. row-stores: how different are they really?* SIGMOD Conference, 2008
- M. Stonebraker et al. *C-Store: A Column-oriented DBMS*. VLDB, 2005
- G. Copeland and S. Khoshafian. *A Decomposition Storage Model*. SIGMOD Conference, 1985
- F. Munir. *Storage Format Selection and Optimization of Materialized Intermediate Results in Data-Intensive Flows*. PhD Thesis (UPC), 2019
- A. Hogan. *Procesado de Datos Masivos* (U. Chile)

References (II)

- P. O'Neil et al. *The log-structured merge-tree (LSM-tree).* Acta Informatica, 33(4). Springer, 1996
- F. Chang et al. *Bigtable: A Distributed Storage System for Structured Data*. OSDI'06
- S. Abiteboul et al. Web Data Management. Cambridge University Press, 2011. http://webdam.inria.fr/Jorge
- N. Neeraj. *Mastering Apache Cassandra*. Packt, 2015
- O. Romero et al. *Tuning small analytics on Big Data: Data partitioning and secondary indexes in the Hadoop ecosystem*. Information Systems, 54. Elsevier, 2016
- A. Petrov. *Algorithms Behind Modern Storage Systems*. Communications of the ACM 61(8), 2018