Log Structured Merge Tree (LSM-tree)

Prof. Pramod Bhatotia
https://dse.in.tum.de/bhatotia/

NoSQL systems

APACHE HBASE

RocksDB

Google Cloud Bigtable

LEVELDB

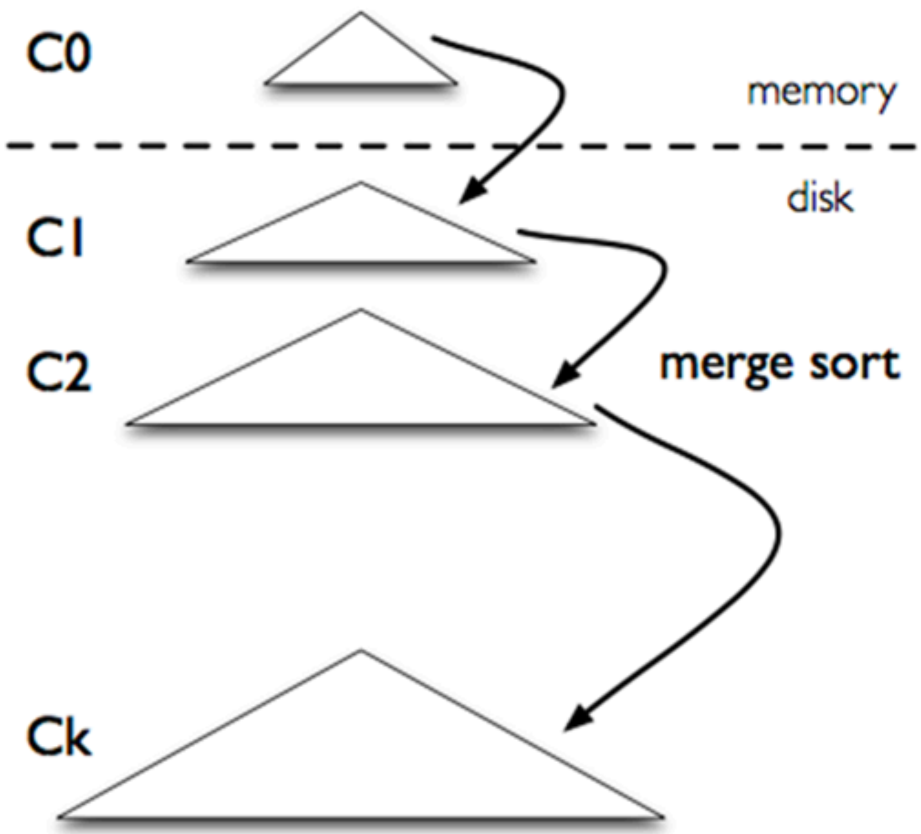
The underlying data structure: LSM-tree data structure

Log-Structured Merge-Tree

- A hierarchical, sequential, disk-oriented data structure
- Offers simple APIs:
  - Put (key, value): write in a key-value pair
  - Get (key): return a value for a key

LSM data structure

- The first level Co is kept in memory
- C1, C2, … Ck are kept on disk
- WAL (write-ahead log) also stored on disk



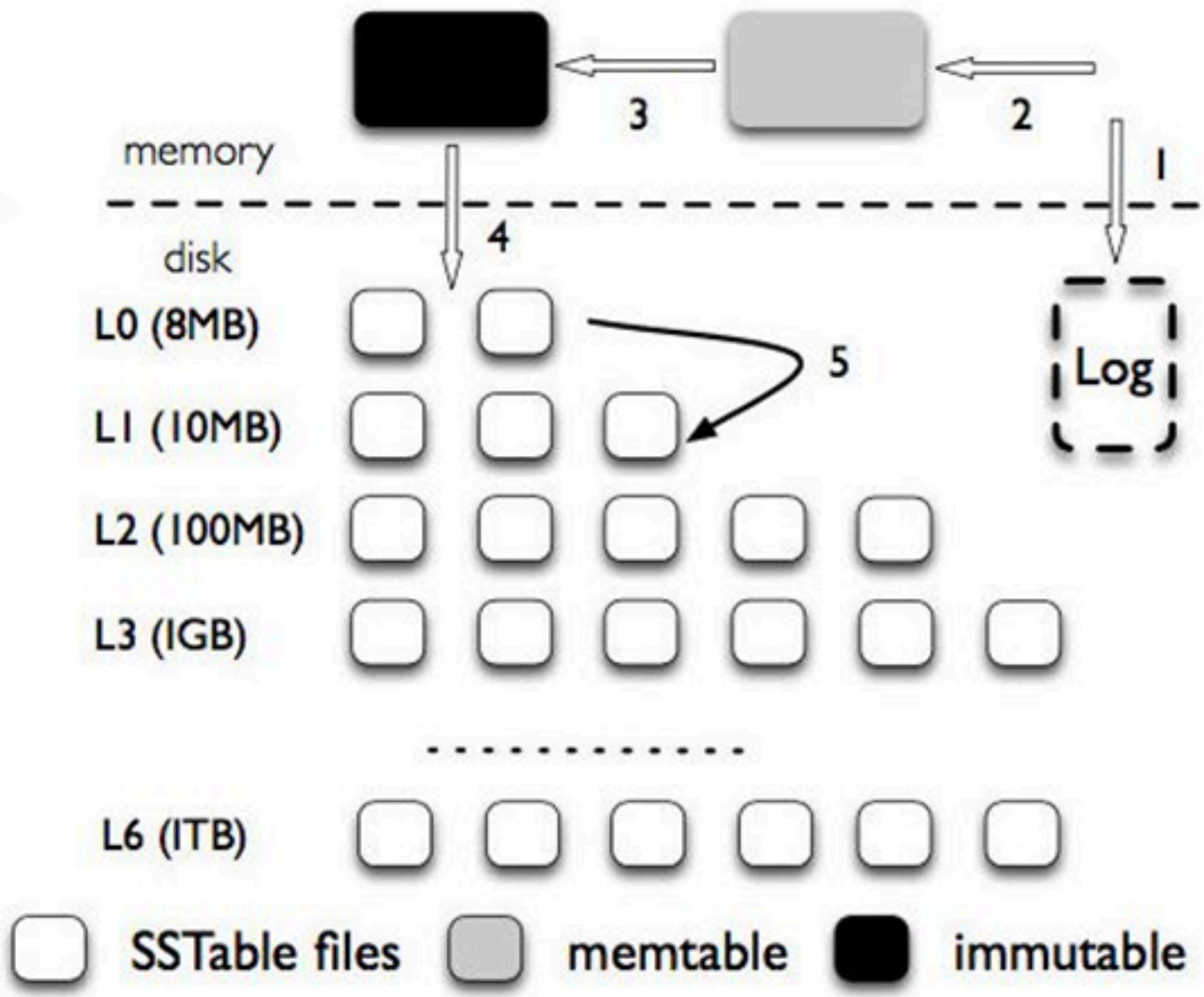
LSM data structures

- MemTable
  - In-memory data structure (non-volatile)
  - Skip list (balanced tree for log (n) lookup operating)

- SSTable
  - Disk based structure (Persistent)
  - Ordered immutable key value store (binary search)

- WAL
  - Append only interface for fault tolerance
  - Two types of logs: UNDO and REDO

LSM layered architecture



Write operation: Put (K, V)

1. The write operation is appended in a WAL file in case of system crash
2. Data is written to the Memtable
3. Upon reaching a configured threshold, the Memtable is marked as immutable Memtable, and a new Memtable will be generated to buffer incoming data
4. Compaction:
    1. The immutable Memtable is flushed to SSTable on disk
    2. Each time when the size of SSTable files reaches its threshold, one or several files will be merged with files from the next layer with overlapping keys

Compaction phase

- Garbage collection
- Compaction is performed asynchronously in the background without blocking the writes

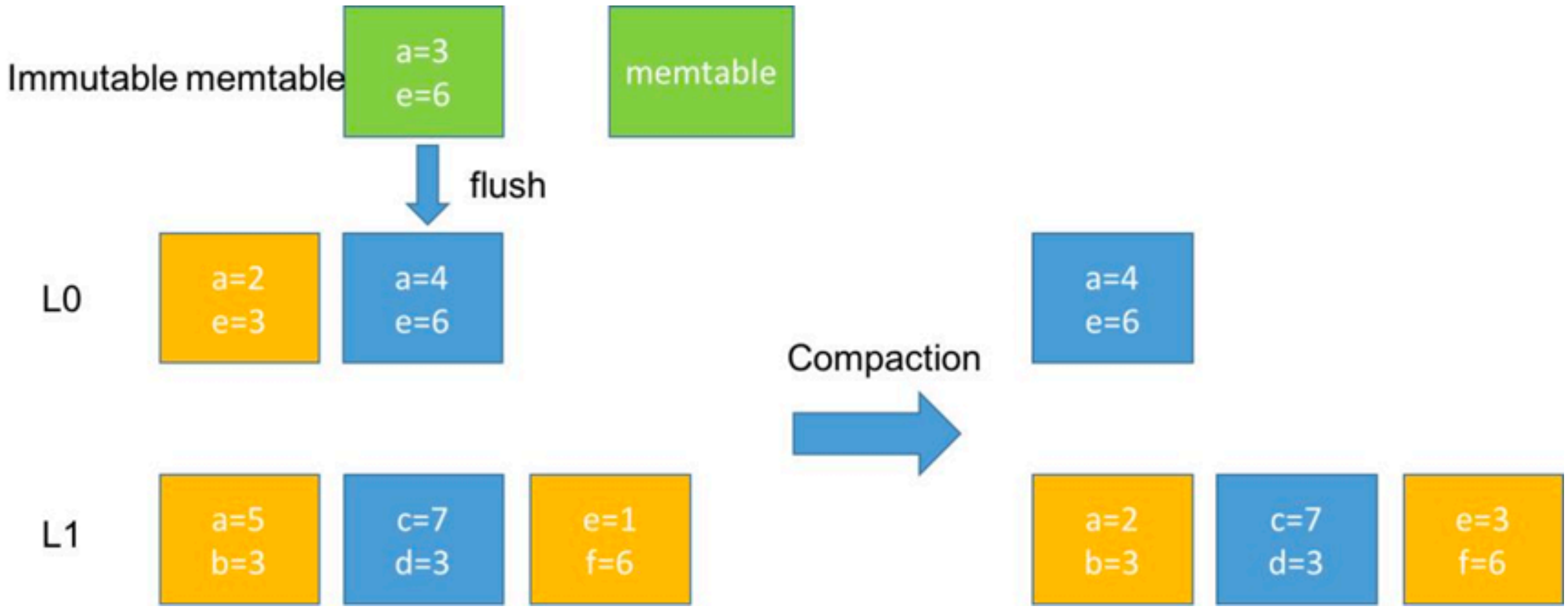
Read operation: Get(k)

1. The key is first looked up in the Memtable and immutable Memtable, and the value is returned if the key was found
2. Otherwise, it will be looked up through next layers L1, L2 … L6 in turn, till the key was found or a null value is returned.

Update & delete operations

- since LSM-tree has an append-only principle, the data could not be updated or deleted in-place from the log
- To update or delete is similar to write the date by appending to the end of the log
- Updates are performed by appending a new KV pair
- Data will be deleted by appending a key value entry with a “tombstone” written as the value. T
- The older values in this case will also be removed in the compaction phase — same as in the write operation

Example


Immutable memtable
a=3
e=6
memtable
flush
L0
a=2
e=3
a=4
e=6
L1
a=5
b=3
c=7
d=3
e=1
f=6
Compaction
a=4
e=6
a=2
b=3
c=7
d=3
e=3
f=6

References

- Log structured merge trees
  - Blog entry: https://medium.com/@qiaojialinwolf/lsm-tree-the-underlying-design-of-nosql-database-cf30218e82f3
  - Original paper: https://www.cs.umb.edu/~poneil/lsmtree.pdf
  - LevelDB: https://github.com/google/leveldb
  - RocksDB: http://rocksdb.org/

Distributed Data Management with Google's BigTable

Prof. Pramod Bhatotia
https://dse.in.tum.de/bhatotia/

Limitations of GFS/HDFS

- Designed for unstructured data
  - Sequential reads (less random reads)
  - High throughput, not so latency sensitive
  - Unstructured data

(Semi-) Structured data

- Web data
  - Contents, crawl metadata, links, pagerank, …
- Per-user data
  - User preference settings, recent queries/search results, …
- Map data
  - Physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …

Why not to use DB systems?

Not scalable enough at Google's scale!

BigTable/HBase

A distributed storage system for structured data

BigTable features

- Structure data model
- Scalable
  - Thousands of servers
  - Terabytes of in-memory data
  - Petabyte of disk-based data
  - Millions of reads/writes per second, efficient scans
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to load imbalance
- Extremely popular at Google (as of 2008)
  - Web indexing, personalized search, Google Earth, Google Analytics, Google Finance, …

Design goal #1: Scalable

- Billions of Web pages, many versions/page (~20K/version)
- Hundreds of millions of users, thousands of q/sec
- 100TB+ of satellite image data

Design goal #2: Efficient

- Very high read/write rates (millions of ops per second)
- Efficient retrieval of small subsets of the data
- Efficient scans over entire or subsets of the data

Design goal #3: Structured data

- Targets structured database
  - BigTable doesn't support full relational model
  - A simple data model with dynamic control over data layout and format
- Data model
  - Row and columns
- Usage:
  - Data storage for batch processing (MapReduce) to low-latency query processing

What is BigTable/Hbase?

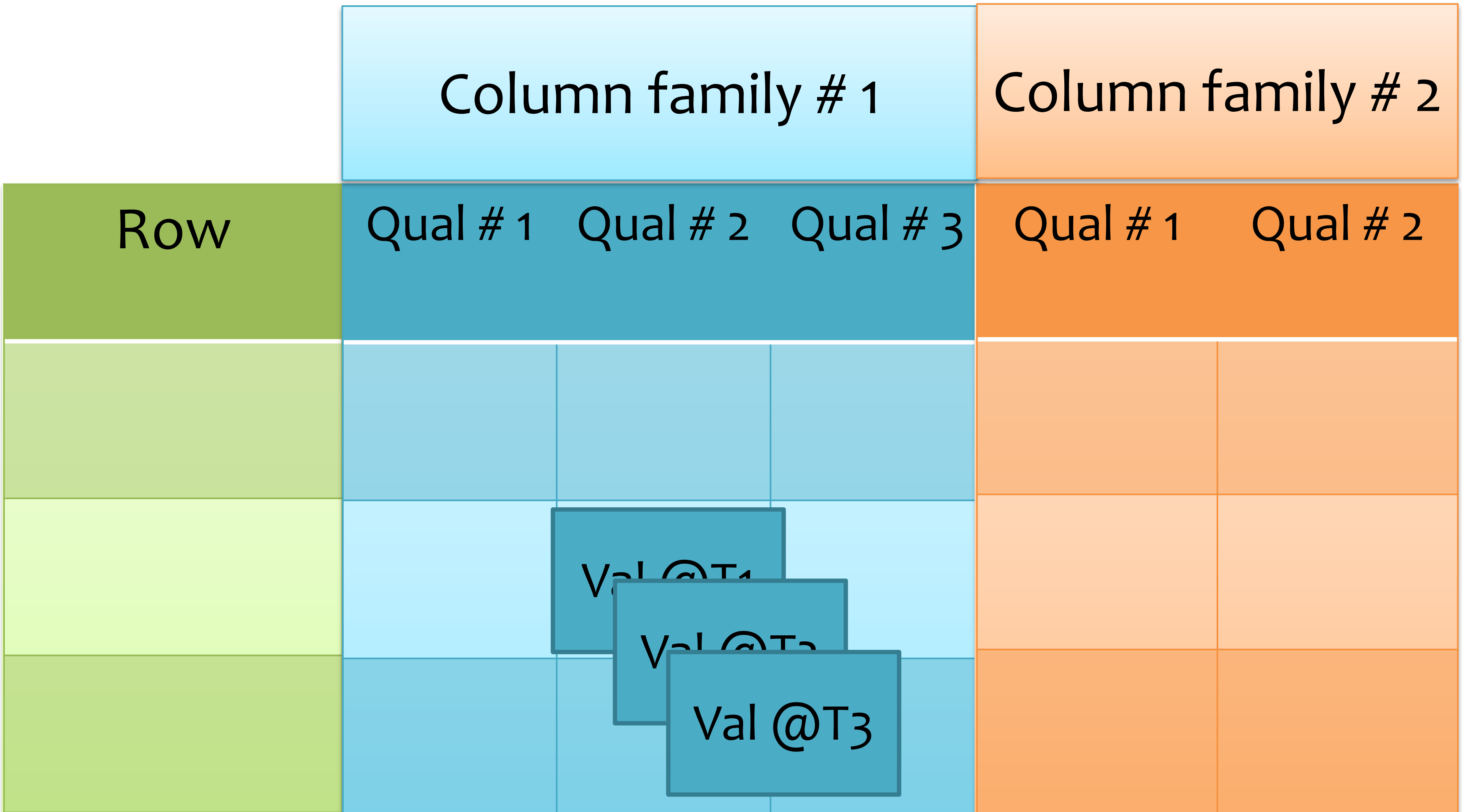
A BigTable is
sparse, distributed, persistent multi-dimensional sorted map

Key:(row, column, time) → value

Data Model

- A BigTable of data with rows and columns
- Rows are uniquely identified by a key
- Columns are organized column families
    - Each family has a set of related columns identified by the column qualifier
- Values in each cell are versioned based on timestamp

Example


Column family # 1
Column family # 2
Row
Qual # 1
Qual # 2
Qual # 3
Qual # 1
Qual # 2
Val @T3

Row

- Row
  - Row name/key is an arbitrary string
  - Sorted lexicographically
  - Access to data in a row is atomic
  - Does not support relation model (transactions)
  - **Tablet:** a range of rows
    - A unit of distribution and load balancing

Column

- Columns are organized hierarchical in families
  - Each column family has a set of columns
- Column_family: column_qualifier
- Column family
  - Unit of access control
  - Stored/compressed together

Time stamp

- Versioning: used to store different versions of data in a cell
  - 64-bits integer (UNIX timestamp)
  - New writes default to current time, but timestamps for writes can also be set explicitly by clients
- Lookup options
  - "Return most recent K values"
  - "Return all values in timestamp range (or all values)"
- Garbage collection:
  - "Only retain most recent K values in a cell"
  - "Keep values until they are older than K seconds"

Programming APIs

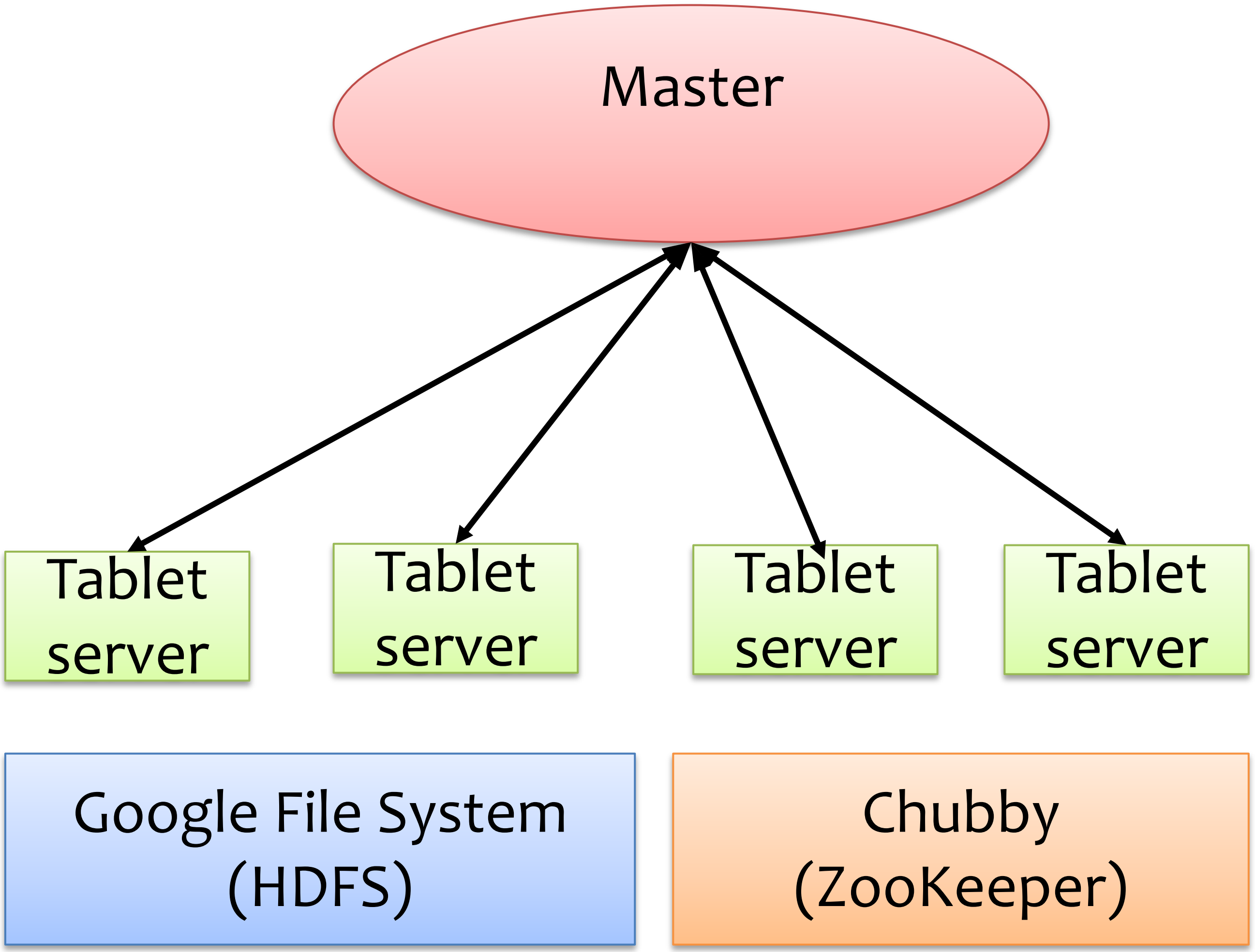
- Create/delete/manage tables
  - Also, new column familes
- CRUD:
  - Create a row via PUT
  - Read a row via GET
  - Write/Update a row (atomically) via PUT
  - Updates: Can be conditional
  - Delete a row via DELETE
  - Transaction: single row read-modify-write
- Scan: Iterator
  - Sequential read over ranges of rows (sorted)

Data analytics integration

- BigTable supports query processing
  - BigQuery: https://cloud.google.com/bigquery

- MapReduce/distributed batch processing

Architecture



BigTable components

- A client library
- One master server
- Many tablet servers
- Zookeeper/chubby
  - Leader election (Master)
  - Reliable storage: tablet info, schema, ACLs, etc.

Tablets

- A Bigtable table is partitioned into many tablets based on row keys
  - Tablets (100-200MB each) are stored in a particular structure in GFS
  - Each tablet is served by one tablet server

Master

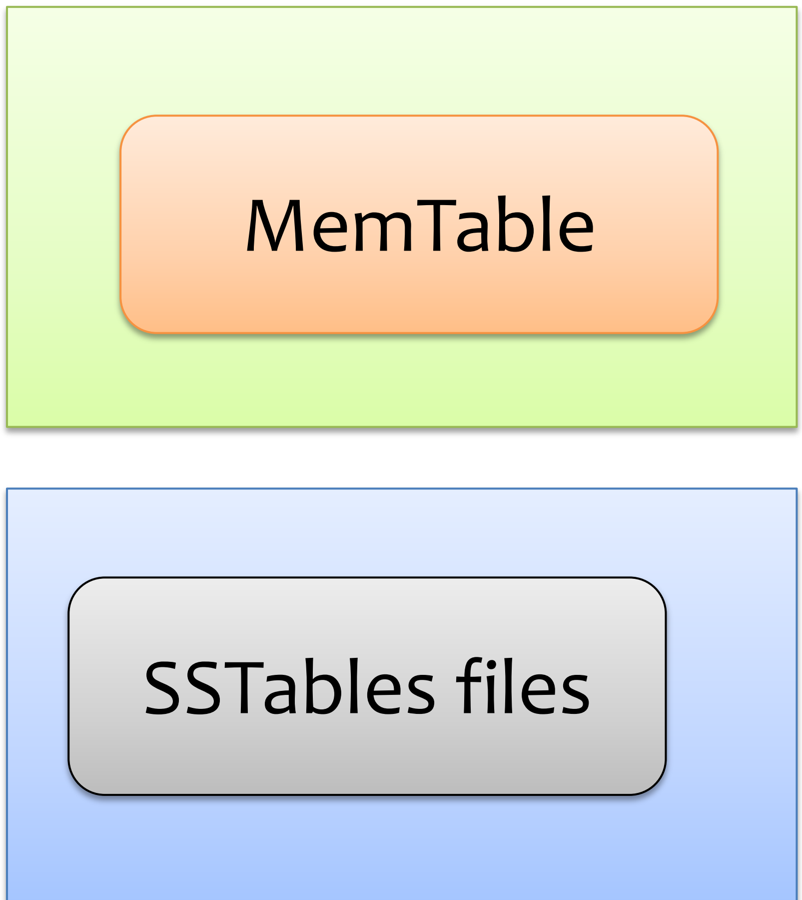
- Assigns/load-balances tablets to tablet servers
- Detects up/down tablet servers
- Garbage collects deleted tablets
- Coordinates metadata/schema updates
- Does NOT provide tablet location

Tablet servers

- Tablet servers handle R/W requests to their tablets
- Split tablets that have grown too large
- Tablet servers are also stateless – their state is in GFS
- Tablets can be dynamically added (or removed)

Tablet server

Memory

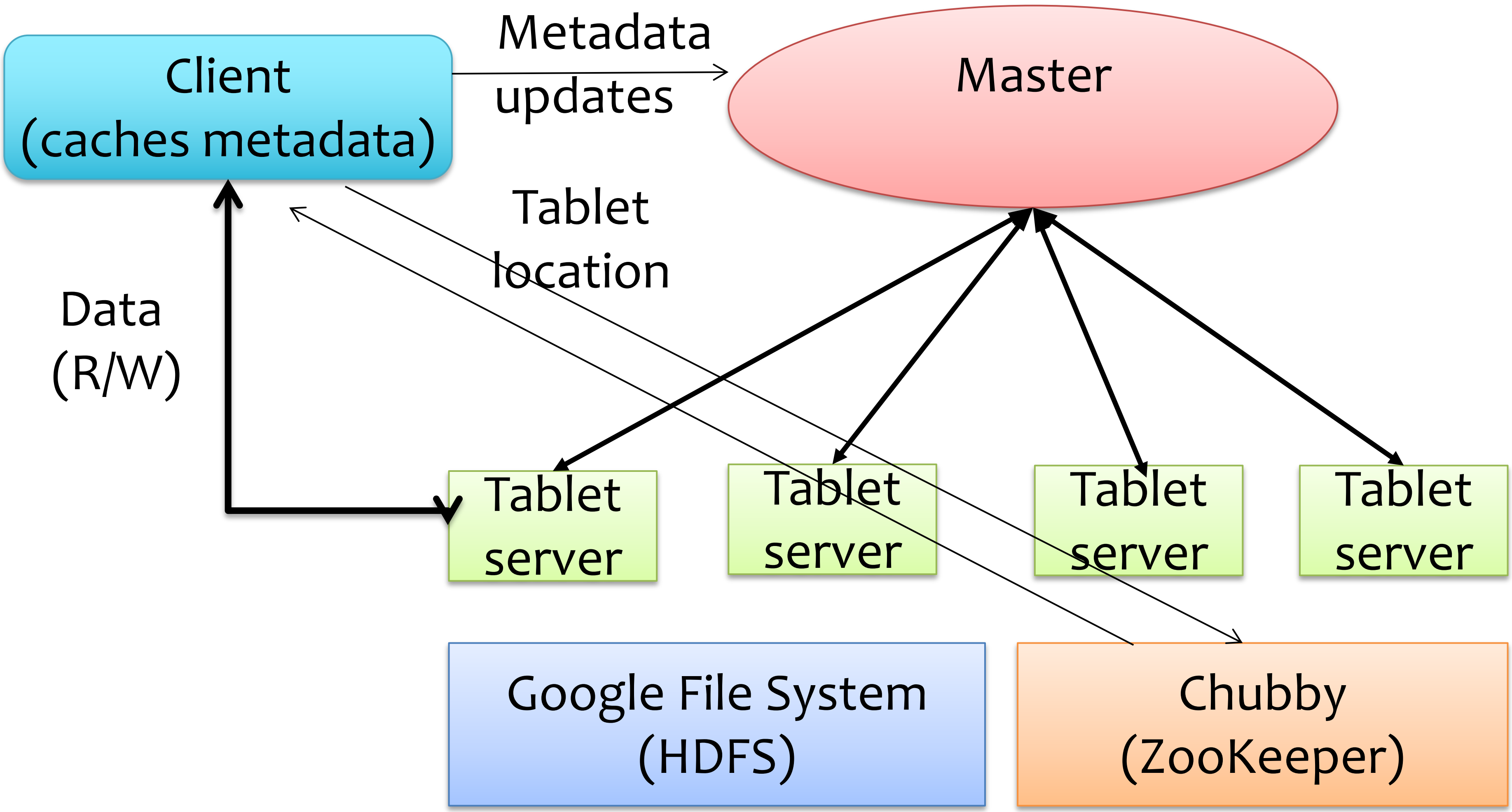


In-memory
Key-value store
(Skip list)

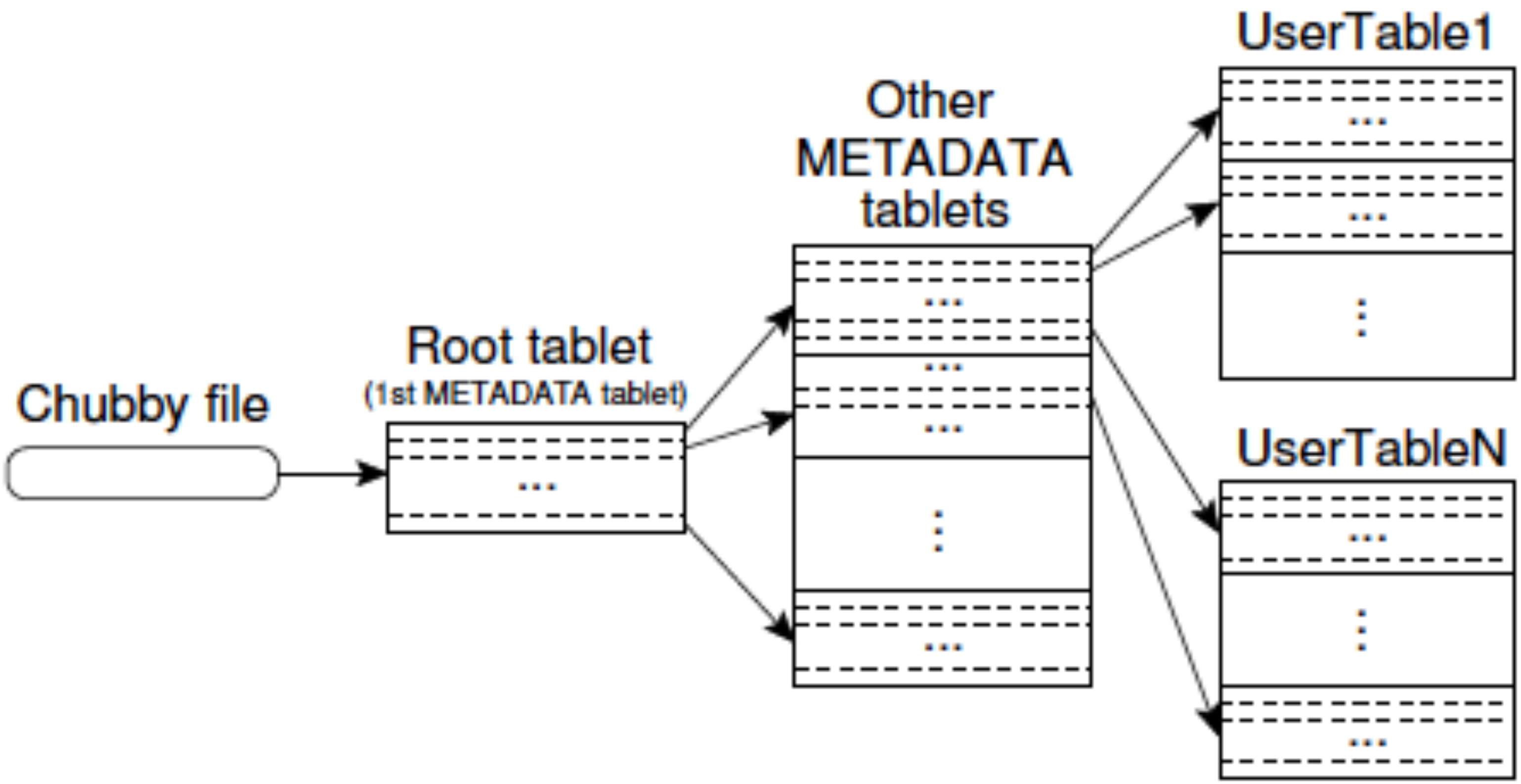
GFS

Persistent,
ordered,
immutable map
from keys to values

BigTable Read/write operations

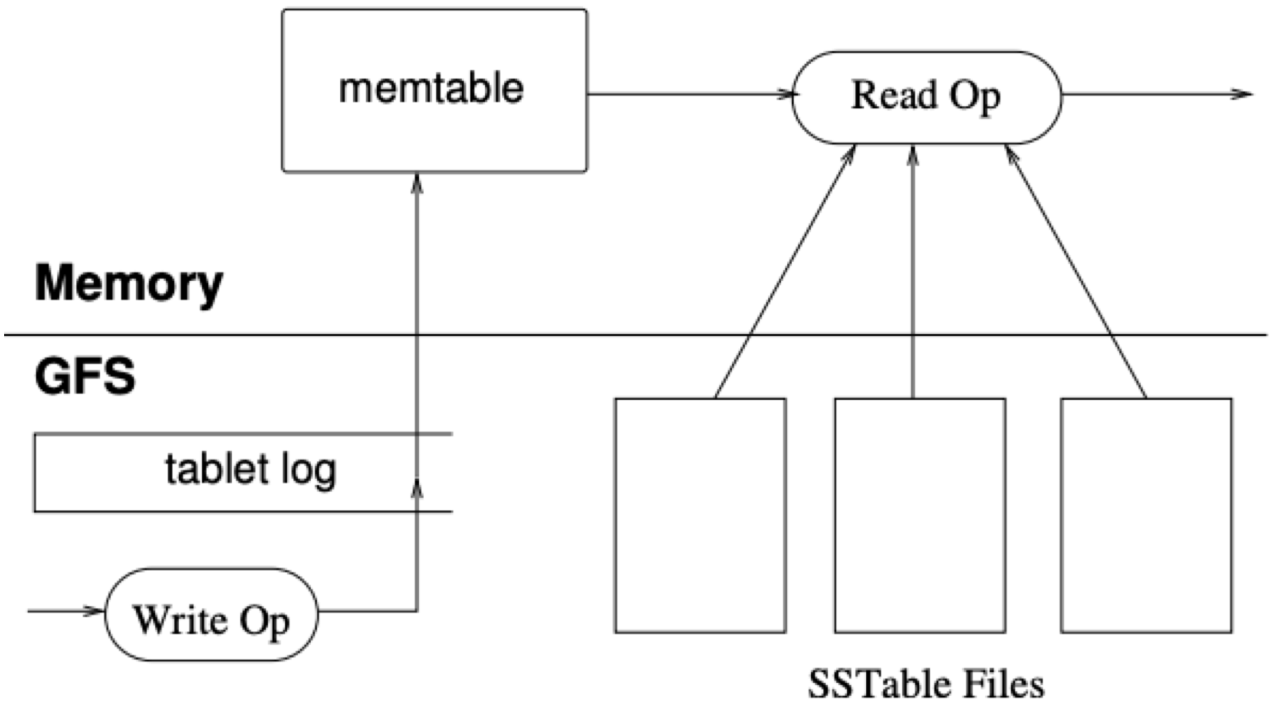


Tablet location



- A three-level hierarchy (B+ tree) to store tablet location
- Stores a mapping from a row key to tablet server

Tablet serving for reads and writes

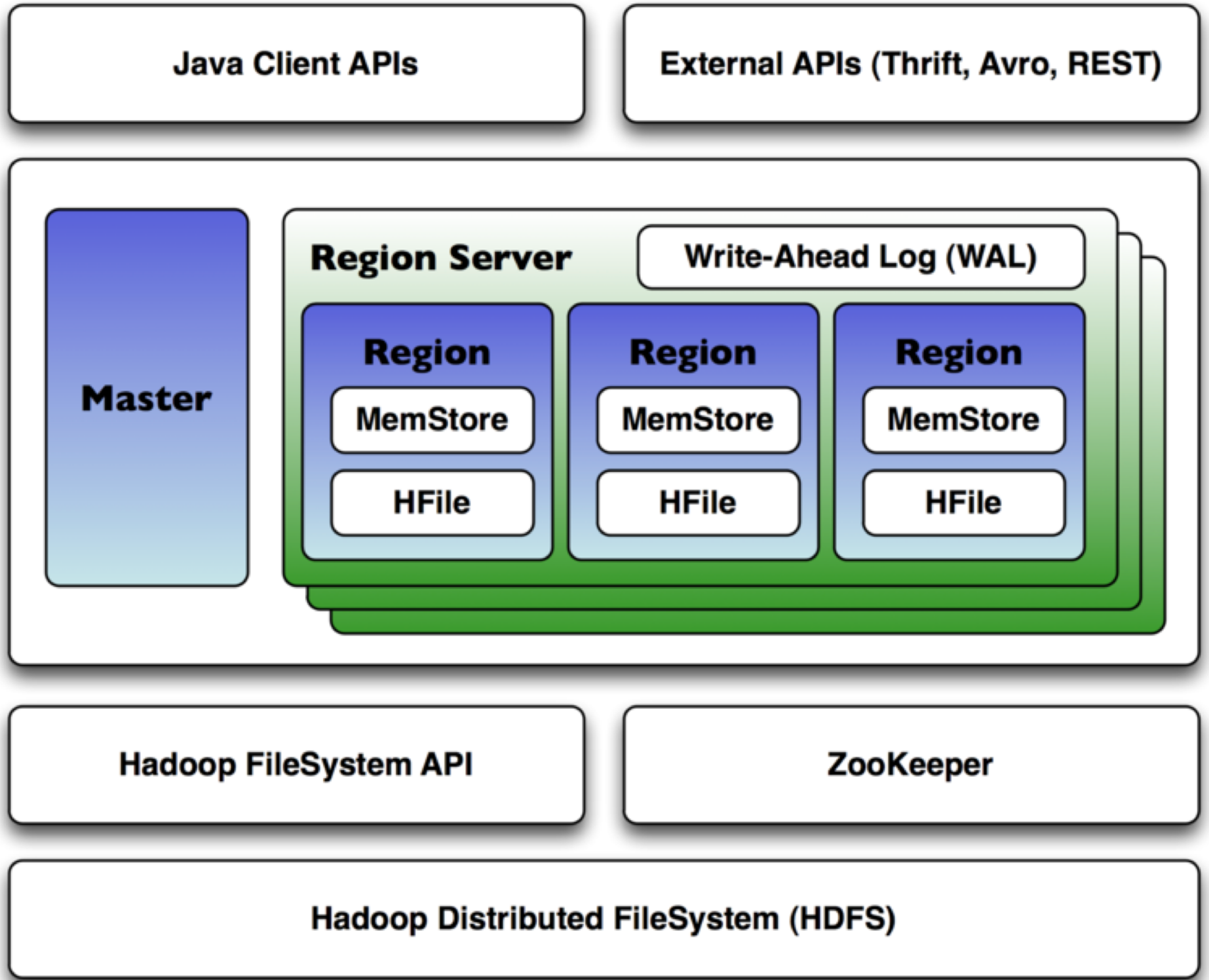


Read and write operations are very similar to LSM-tree data structure

How does Tablet recovery work?

- A tablet server reads its metadata from the METADATA table (root node in Chubby)
- Reconstruct the state in MemTable from
  - A set of SSTable files
  - Applying pending updates from the REDO log (WAL)

HBase architecture



Reference

- Compulsory reading: BigTable [OSDI'06]
  - Original paper for BigTable
  - https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf
- Recommended reading: Spanner [OSDI'12]
  - Distributed databases with serializable transactions
  - https://www.usenix.org/system/files/conference/osdi12/osdi12-final-16.pdf

Summary

- Data-management with BigTable
  - Distributed storage architecture
  - BigTable: A database system for structured data

- Resources:
  - HBase: http://hbase.apache.org/
  - BigTable: https://cloud.google.com/bigtable