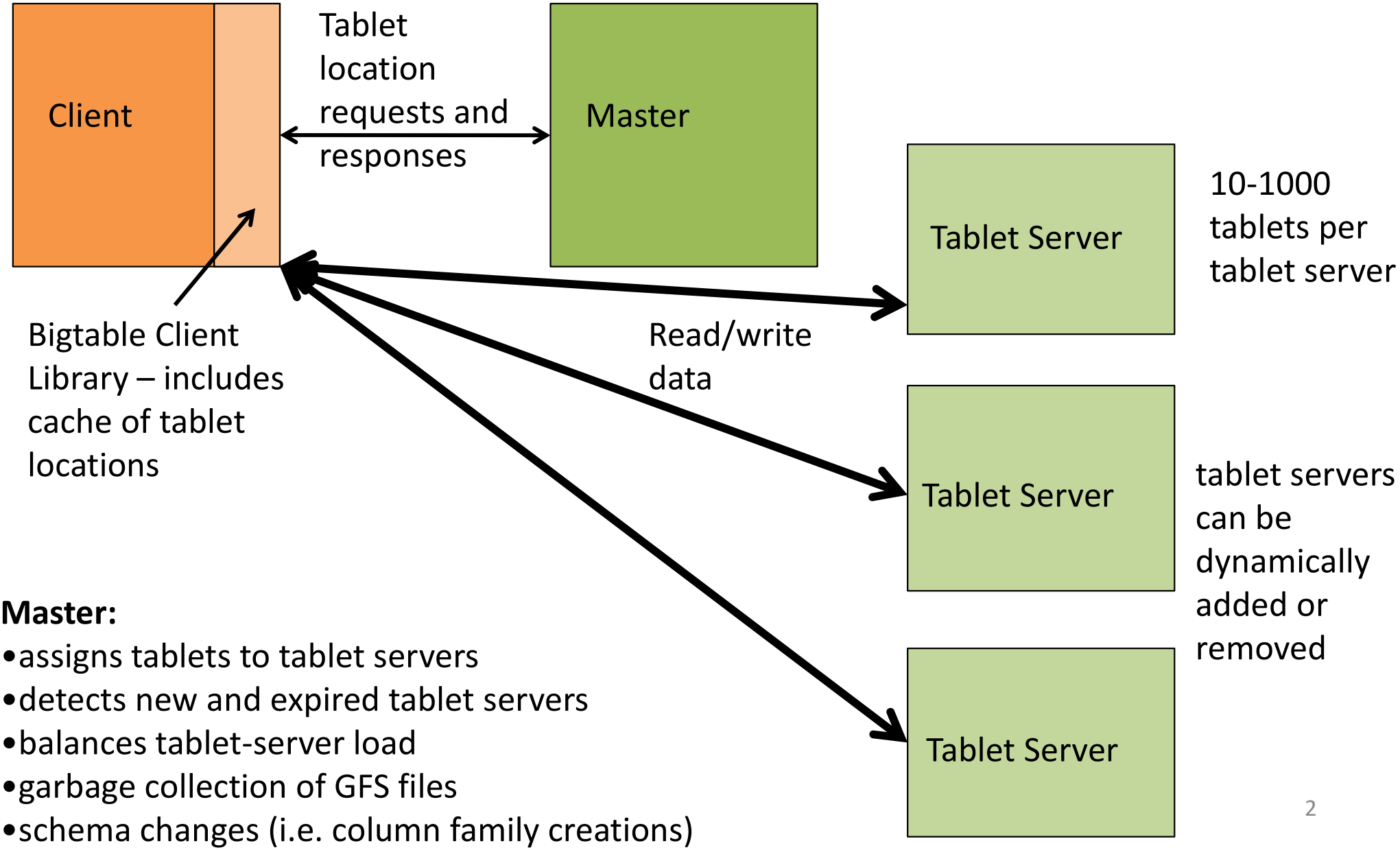
Cloud and Cluster Data Management

BIGTABLE – PART 2

Bigtable – Implementation



Master:

- assigns tablets to tablet servers
- detects new and expired tablet servers
- balances tablet-server load
- garbage collection of GFS files
- schema changes (i.e. column family creations)

Bigtable – Tablets

- Table consists of a set of tablets
- Each tablet contains all of the data associated with a row range (range partitioning on row key)
- Table automatically split into multiple tablets – each tablet about 1GB in size (though can be bigger for large rows)

Tablet Location

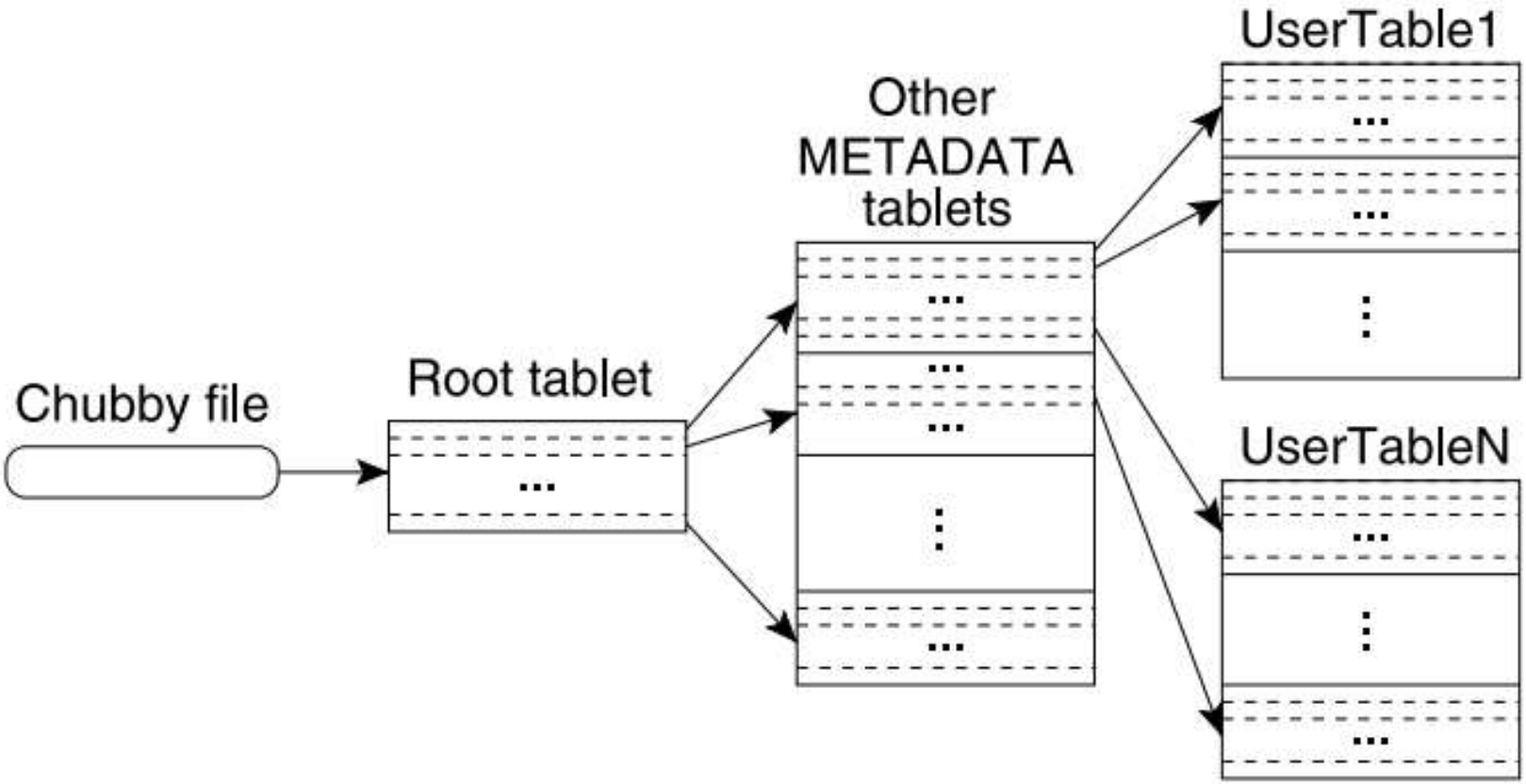


Fig. 5. Tablet location hierarchy.

What is Chubby?

- A highly-available and persistent distributed lock service
- Provides a namespace that consists of directories and small files
- Each directory or file can be used as a lock
  - Reads and writes to a file are atomic
- Chubby client provides consistent caching of Chubby files
  - All Tablet Servers and Applications see consistent metadata and mapping information
- Chubby client has a session lease. If the lease expires, the client loses its locks.
  - Used to detect failed tablet servers

Tablet Location

- Location tree is like B+ tree
- Chubby stores the location of the root tablet
  - Root tablet – stores locations of tablets of a special METADATA table
- METADATA tablets contains locations of user tablets
- Root is never split
  - Hierarchy is always 3 levels
  - Up to 2^{61} bytes with 128MB Metadata Tablets (2^{34} tablets)
- METADATA table/tablets store location and a rowkey = (tableid + end row)
- Each METADATA row ~1k

Client – Tablet Location

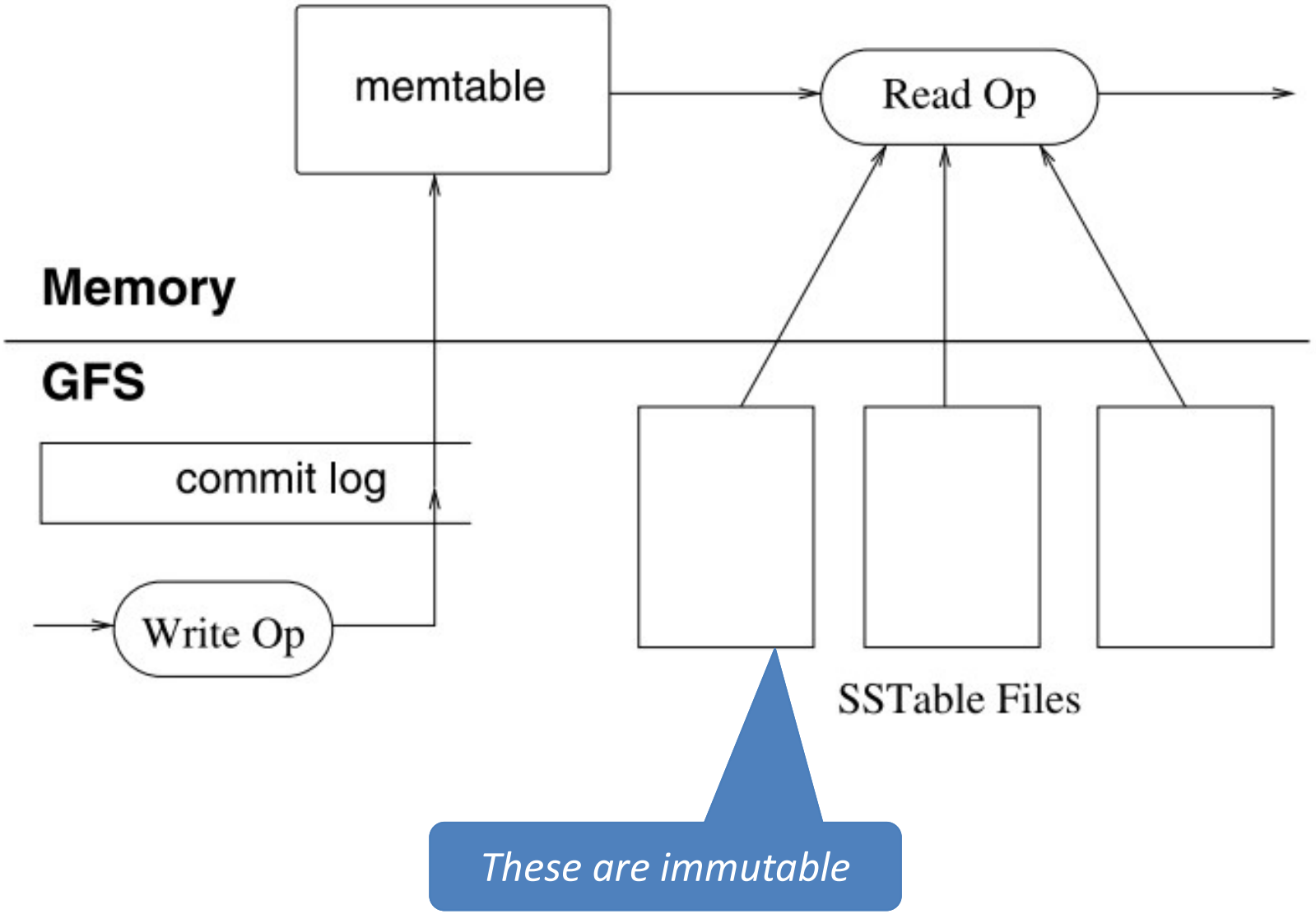
- Client traverses hierarchy to locate tablets, caches locations
- Empty / stale cache
  - Moves up in hierarchy if information is incorrect or not known (like B+ tree)
  - Stale cache entries discovered on misses
- Client also uses prefetching to limit round trips to master

Tablet Assignment

- Tablet assigned to at most one tablet server

  Reliability from logging and GFS replication
- Master keeps track of live tablet servers and assignment of tablets to tablet servers
- Chubby used to keep track of tablet servers (TS)
  - TS starts – creates and acquires exclusive lock on file in specific Chubby directory (servers directory)
  - Master monitors this directory
  - Tablet stops serving if it loses its exclusive lock (i.e. network partition causes loss of Chubby session)
- Master periodically pings tablet servers to make sure they still have their locks

Tablet Representation



Tablet Serving – General Features

- Persistent state of tablet stored in GFS
- Updates written to a commit log that stores redo records
- Recently-committed updates stored in memtable
- Copy on write for updates
- Redo after crash:
  - Read metadata from METADATA table
    - SSTables & set of redo points
  - Read indices of SSTables into memory & reconstruct memtable by redoing updates since last redo point (checkpoint)
- Tablet servers handle tablet splits, other changes handled by master (table created, tablets merged)

Tablet Serving – Writes & Reads

- Write operation
  - Check for well-formed
  - Check for authorization (Chubby file, perms are at column-family level)
  - Valid mutation written to commit log
  - After commit, contents inserted into memtable
- Reads
  - Check for well-formed
  - Check for authorization
  - Read operation executed on a merged view of SSTables and the memtable (both sorted)

Compactions

- Memtable increases in size with write operations
  - At threshold – minor compaction
    - memtable frozen
    - new memtable created
    - frozen memtable converted to an SSTable and written to GFS
  - Goals:
    - Shrinks memory usage of memtable
    - Reduces amount of data that has to be read from the commit log if server dies

Participation Question 1

Bigtable's tablet location hierarchy is designed to minimize lookup overhead by using a three-level structure (Chubby → Root Tablet → METADATA Tablets → User Tablets). Imagine that a new Bigtable deployment experiences frequent tablet migrations due to fluctuating workloads. How would this affect client performance, and what strategies could you use to optimize tablet lookups in such a scenario?

1. **Discuss within your group**:
    1. What happens when **tablet locations change frequently**?
    2. How does **Chubby, caching, and prefetching** help minimize lookup overhead?
    3. What are the possible **downsides** of excessive tablet movement?
    4. What **optimizations** can be introduced to reduce client lookup delays?

Compactions

- Merging compaction
  - Reads SSTables and memtable and outputs a new SSTable
  - Run in background
  - Discard memtable and SSTable when done
- Major compaction – leaves only one SSTable per tablet
  - Non-major compactions can leave deletion entries and deleted data
  - Major compaction leaves no deleted data
  - Major compactions done periodically on all tables

Bigtable Schemas – Chubby

- Schemas stored in Chubby
- Chubby client sends update to Chubby master, ACL is checked
- Master installs new schema by (atomically) writing new schema file
- Tablet servers get schema by reading appropriate file from Chubby

  File is up-to-date due to consistent caching
- Comment: note effect of having only column families at schema level

Bigtable – Refinements

- Goal of refinements: performance, availability, reliability
- Locality groups
  - Column families assigned to client-defined locality group
  - SSTable generated for each locality group in each tablet (vertical partitioning)
  - Segregate column families that are not typically accessed together
- Ex: locality groups for Webtable
  - page meta-data (language, checksums)
  - page contents
- Locality groups can be declared to be in-memory
  - Good for small pieces of data that are accessed frequently
- Optional compression of SSTables
  - Have seen up to 90% reduction

Bigtable Caching / Commit Log

- Scan Cache
  - High-level cache that caches key-value pairs
  - Useful for applications that read the same data repeatedly
- Block Cache
  - Lower-level cache, caches SSTable blocks
  - Useful for applications that read data that is close to the data they recently read (sequential read, random reads)
- Commit Log
  - If commit logs were separate files, lots of disk seeks for writes
  - One commit log per tablet server – good performance during normal operation, but needs to be split up on recovery

Real Applications

- Google Analytics
  - Raw click table (200TB) – row for each end-user session – name is web site name and session creation time
    - Sessions for the same web site are contiguous and sorted chronologically
  - Summary table (20TB) – predefined summaries for each web site
    - Generated from Raw Click table by Map Reduce jobs
- Google Earth
  - Preprocessing pipeline uses table to store raw imagery (70TB) – served from disk
  - Preprocessing – Map Reduce over Bigtable to transform data
  - Serving system – one table to index data stored in GFS (500GB) – in-memory column families used

Reference

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. **Bigtable: A Distributed Storage System for Structured Data**. ACM Trans. Comput. Syst. 26, 2, Article 4 (June 2008)