

# BigTable

## Distributed storage for structured data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber. (2006) Bigtable: A Distributed Storage System for Structured Data. Proceedings of the 7th Symposium on Operating System Design and Implementation (OSDI 2006), pages 205-218.

<http://the-paper-trail.org/blog/bigtable-googles-distributed-data-store/>

<http://read.seas.harvard.edu/cs261/2011/bigtable.html>

<http://www.cs.rutgers.edu/~pxk/417/notes/content/bigtable.html>

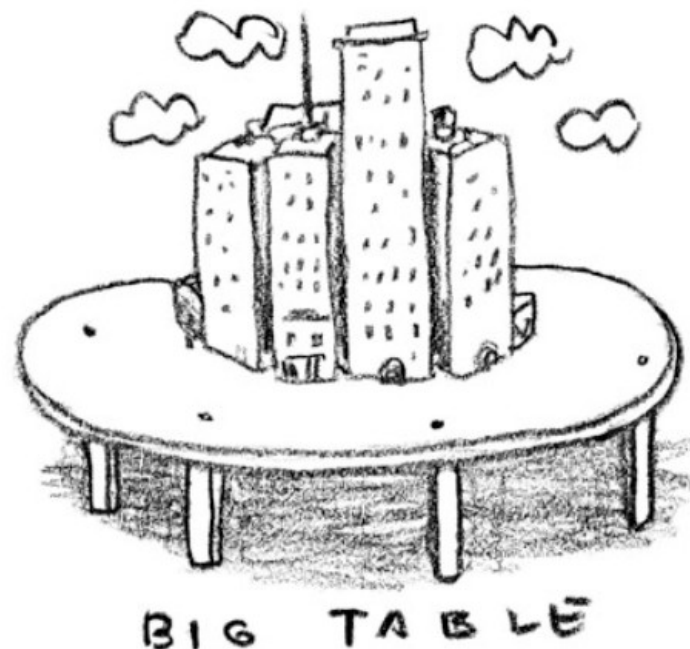
# Overview

## ■ Goals

- scalability
  - petabytes of data
  - thousands of machines
- applicability
  - to Google applications
    - Google Analytics
    - Google Earth
    - ...
  - not a general storage model
- high performance
- high availability


## ■ Structure

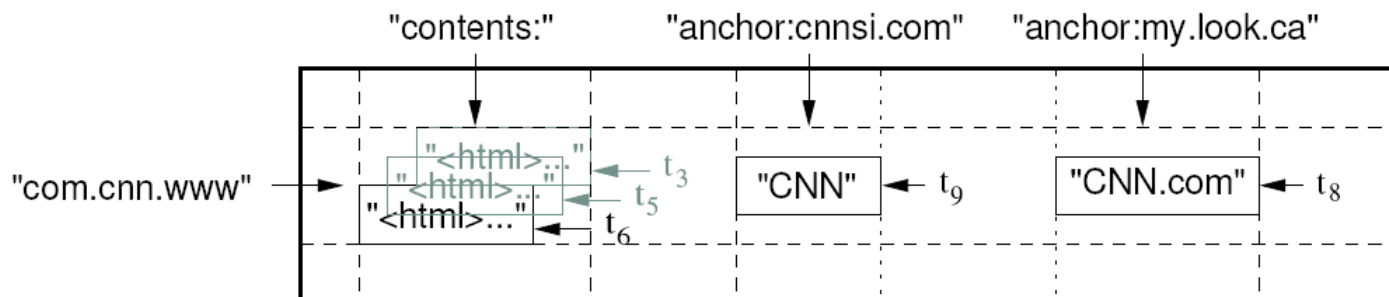
- uses GFS for storage
- uses Chubby for coordination



Note: figure from presentation by Jeff Dean (Google)

# Data Model

(row: string, column: string, timestamp: int64)  string



- A BigTable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, a column key, and a timestamp; each value in the map is an uninterpreted array of bytes
- Row keys
  - up to 64K, 10-100 bytes typical
  - lexicographically ordered
  - reading adjacent row ranges efficient
  - organized into tablets: row ranges
- Column keys
  - grouped into column families - family:qualifier
  - column family is basis for access control


## Column Families

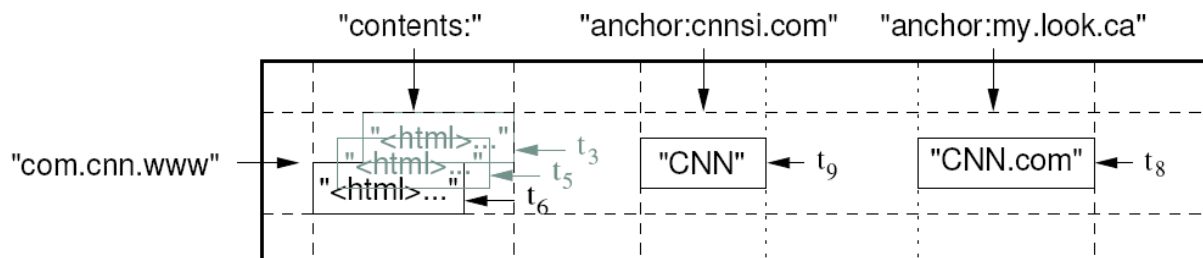
- Column keys are grouped into sets called column families.
  - A column family must be created before data can be stored in a column key.
  - Hundreds of static column families.
  - Syntax is family:key, e.g., Language:English, Language:German, etc.
-

## Timestamps

- 64 bit integers
  - Assigned by:
    - Bigtable: real-time in microseconds,
    - Client application: when unique timestamps are a necessity.
  - Items in a cell are stored in decreasing timestamp order.
  - Application specifies how many versions (n) of data items are maintained in a cell.
    - Bigtable garbage collects obsolete versions.
-

# Data Model

(row: string, column: string, timestamp: int64)  string



## ■ Timestamps

- automatically assigned (real-time) or application defined
- used in garbage collection (last n, n most recent, since time)

## ■ Transactions

- iterator-style interface for read operation
- atomic single-row updates
- no support for multi-row updates
- no general relational model

# Data Model

Row	Timestamp	Column family: animal:		Column family repairs:
		animal:type	animal:size	repairs:cost
enclosure1	t2	zebra		1000 EUR
	t1	lion	big	
enclosure2	...	...	...	...

We may have a huge number (e.g., hundreds of thousands or millions) of columns but the column family for each row will have only a tiny fraction of them populated.

While the number of column families will typically be small in a table (at most hundreds), the number of columns is unlimited.

---

# Data Model

## Column family animal:

(enclosure1, t2, animal:type)	zebra
(enclosure1, t1, animal:size)	big
(enclosure1, t1, animal:type)	lion

## Column family repairs:

(enclosure1, t1, repairs:cost)	1000 EUR
--------------------------------	----------



# Bigtable

- Used in different applications supported by Google.

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
<i>Crawl</i>	800	11%	1000	16	8	0%	No
<i>Crawl</i>	50	33%	200	2	2	0%	No
<i>Google Analytics</i>	20	29%	10	1	1	0%	Yes
<i>Google Analytics</i>	200	14%	80	1	1	0%	Yes
<i>Google Base</i>	2	31%	10	29	3	15%	Yes
<i>Google Earth</i>	0.5	64%	8	7	2	33%	Yes
<i>Google Earth</i>	70	–	9	8	3	0%	No
<i>Orkut</i>	9	–	0.9	8	5	1%	Yes
<i>Personalized Search</i>	4	47%	6	93	11	5%	Yes

## Application 1: Google Analytics

- Enables webmasters to analyze traffic pattern at their web sites. Statistics such as:
    - Number of unique visitors per day and the page views per URL per day,
    - Percentage of users that made a purchase given that they earlier viewed a specific page.
  - How?
    - A small JavaScript program that the webmaster embeds in their web pages.
    - Every time the page is visited, the program is executed.
    - Program records the following information about each request:
      - User identifier
      - The page being fetched
-

## Application 1: Google Analytics (Cont...)

### ■ Two of the Bigtables

#### □ Raw click table (~ 200 TB)

- A row for each end-user session.
- Row name include website's name and the time at which the session was created.
- Clustering of sessions that visit the same web site. And a sorted chronological order.
- Compression factor of 6-7.

#### □ Summary table (~ 20 TB)

- Stores predefined summaries for each web site.
  - Generated from the raw click table by periodically scheduled MapReduce jobs.
  - Each MapReduce job extracts recent session data from the raw click table.
  - Row name includes website's name and the column family is the aggregate summaries.
- 
- Compression factor is 2-3.

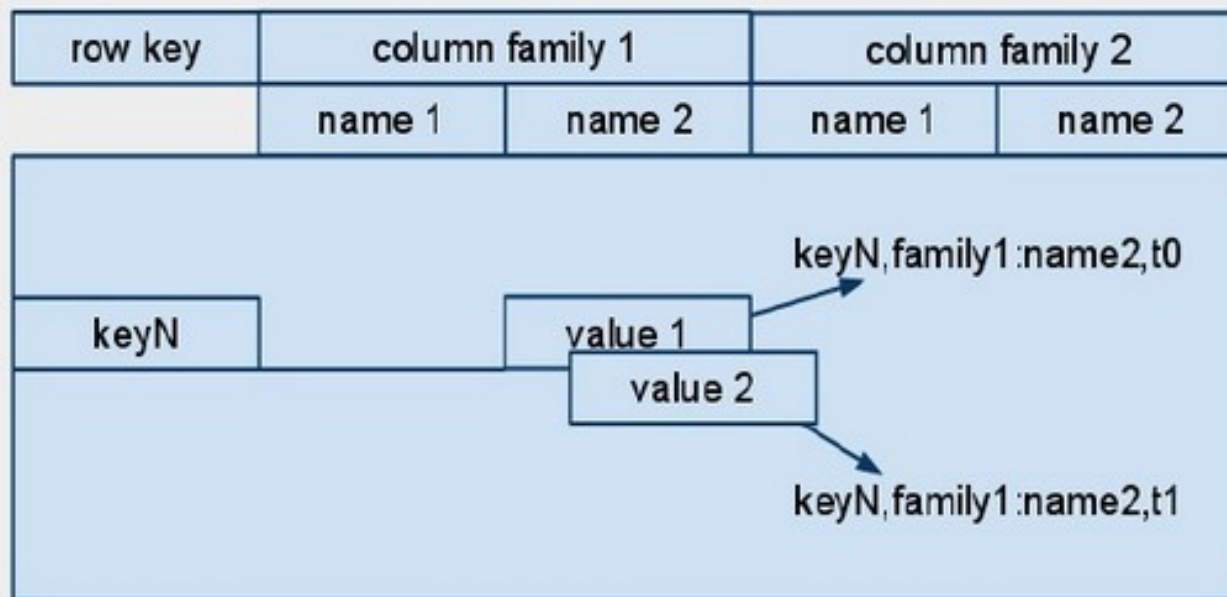
## Application 2: Google Earth & Maps

- **Functionality:** Pan, view, and annotate satellite imagery at different resolution levels.
  - **One Bigtable stores raw imagery (~ 70 TB):**
    - Row name is a geographic segments. Names are chosen to ensure adjacent geographic segments are clustered together.
    - Column family maintains sources of data for each segment.
  - **There are different sets of tables for serving client data, e.g., index table.**
-

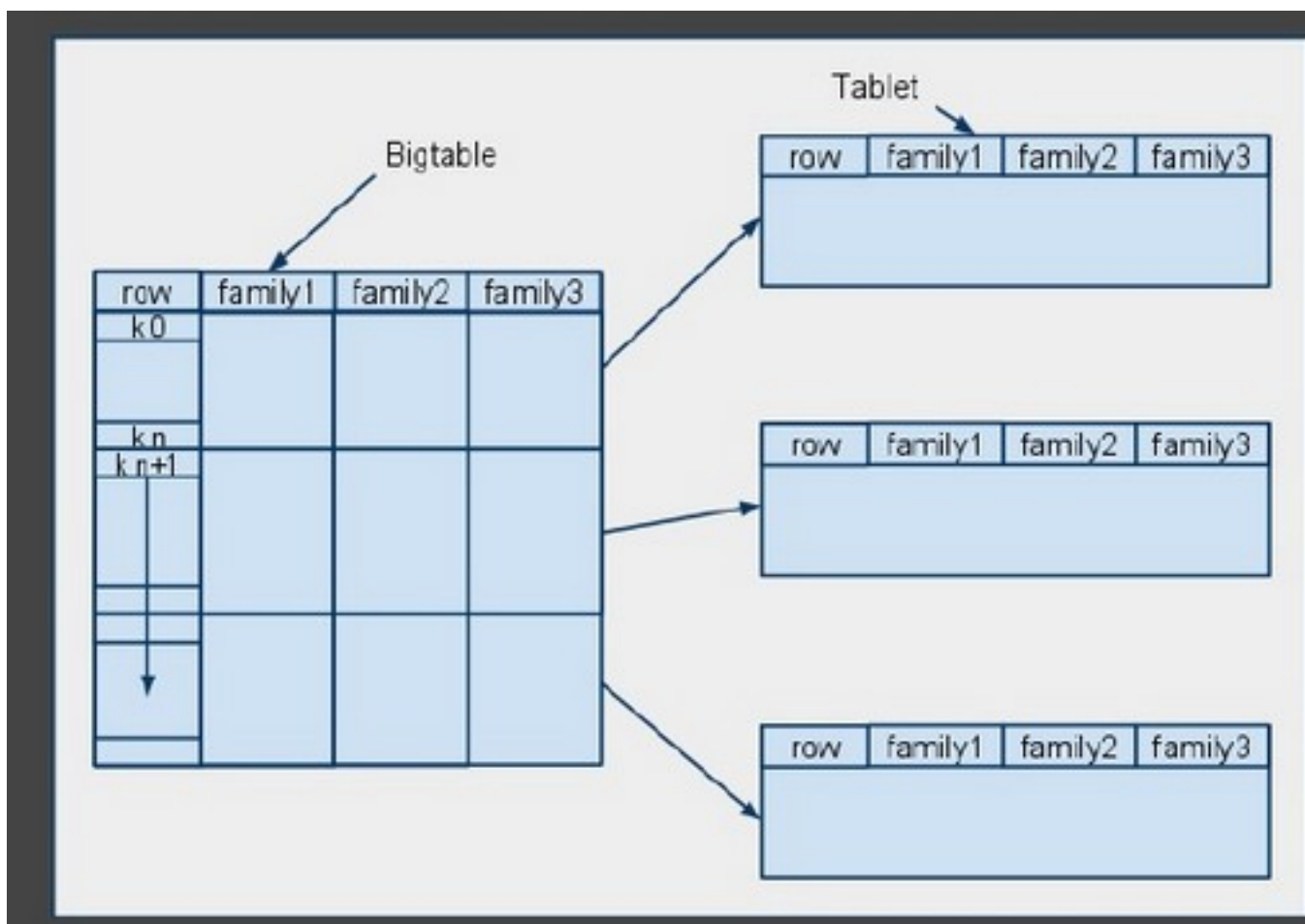
## Application 3: Personalized Search

- Records user queries and clicks across Google properties.
  - Users browse their search histories and request for personalized search results based on their historical usage patterns.
  - One Bigtable:
    - Row name is userid
    - A column family is reserved for each action type, e.g., web queries, clicks.
    - User profiles are generated using MapReduce.
      - These profiles personalize live search results.
    - Replicated geographically to reduce latency and increase availability.
-

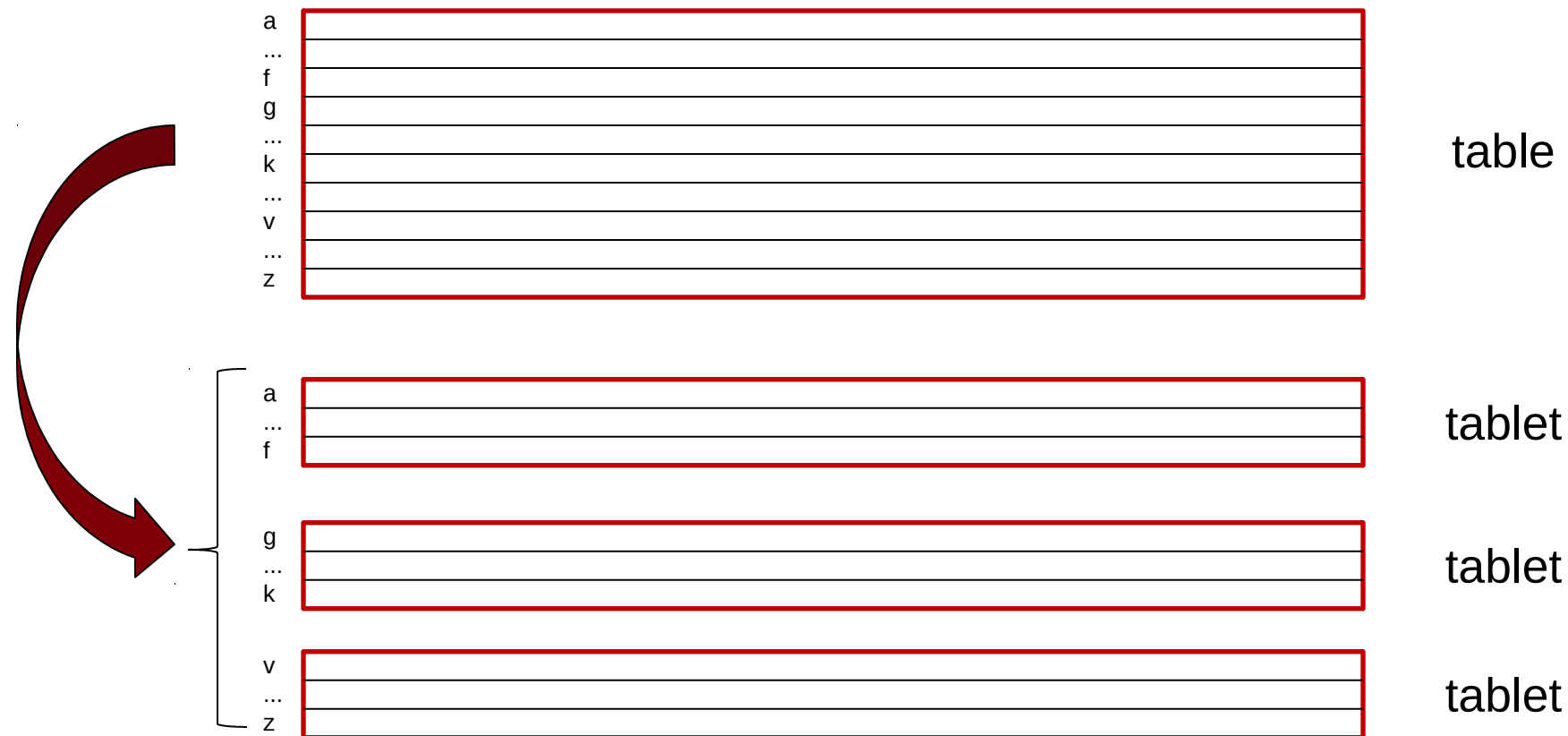
# Data Model



# Data Model



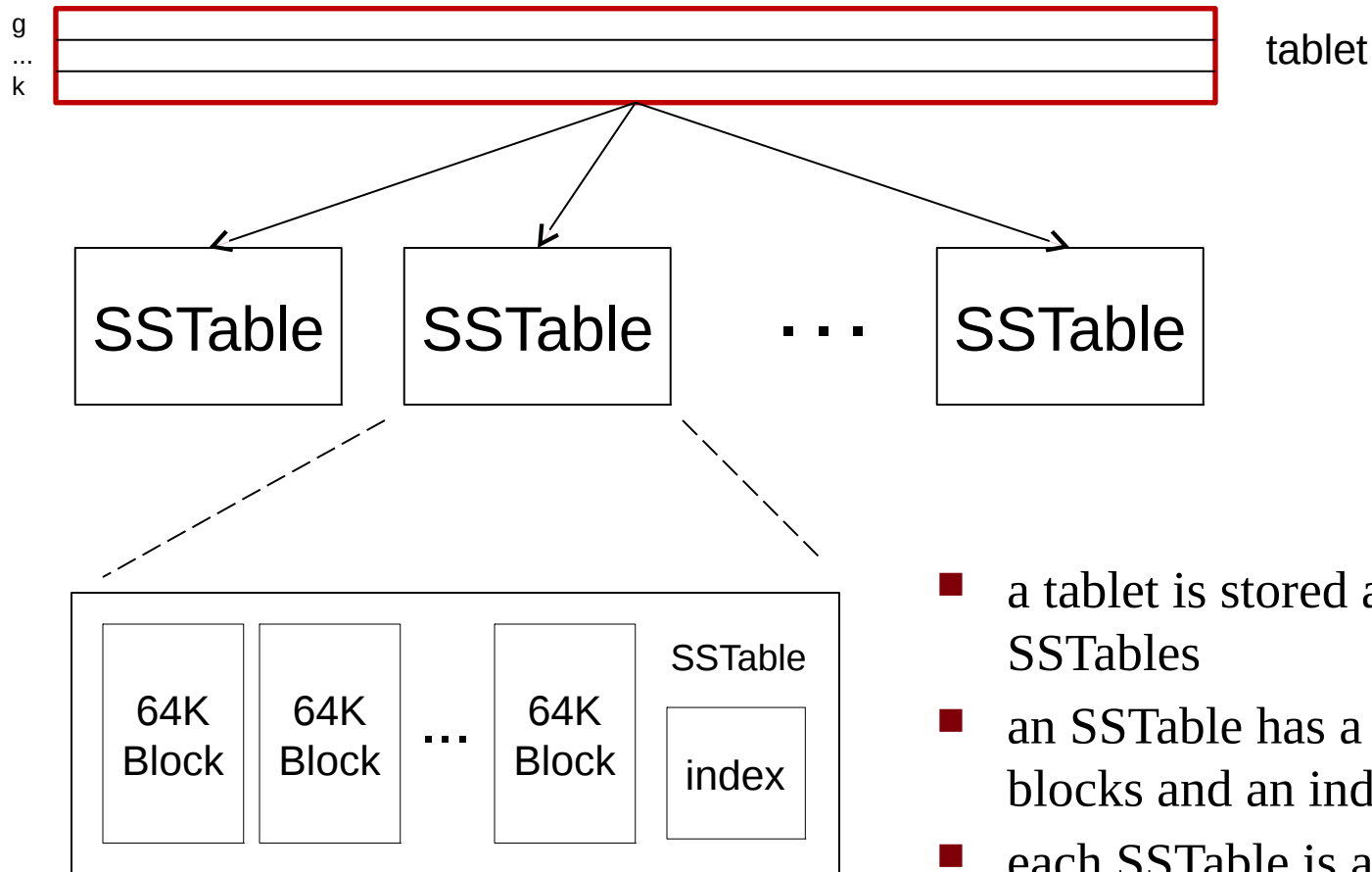
# Table implementation



- a table is divided into a set of tablets, each storing a set of consecutive rows
- tablets typically 100-200MB

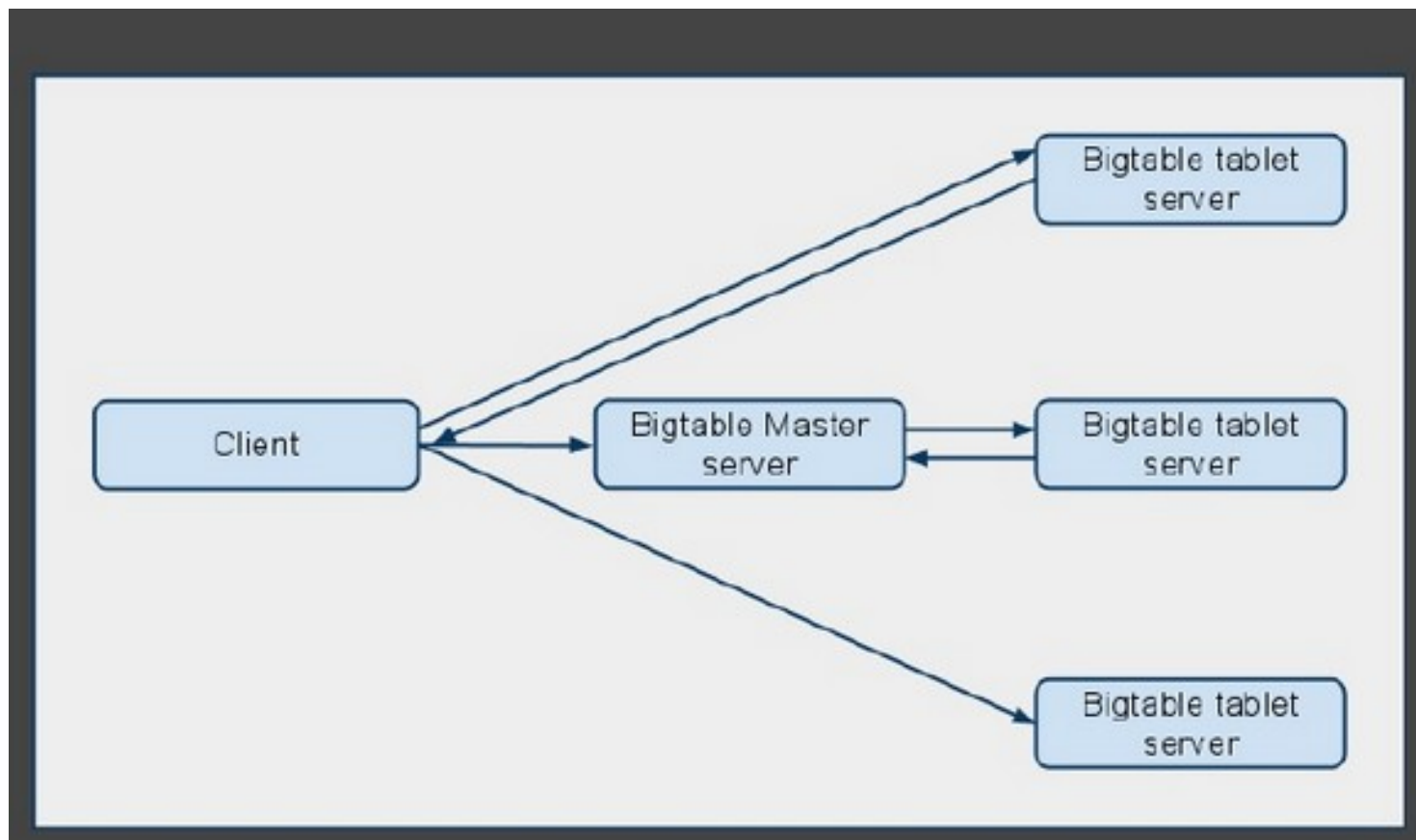


# Table implementation

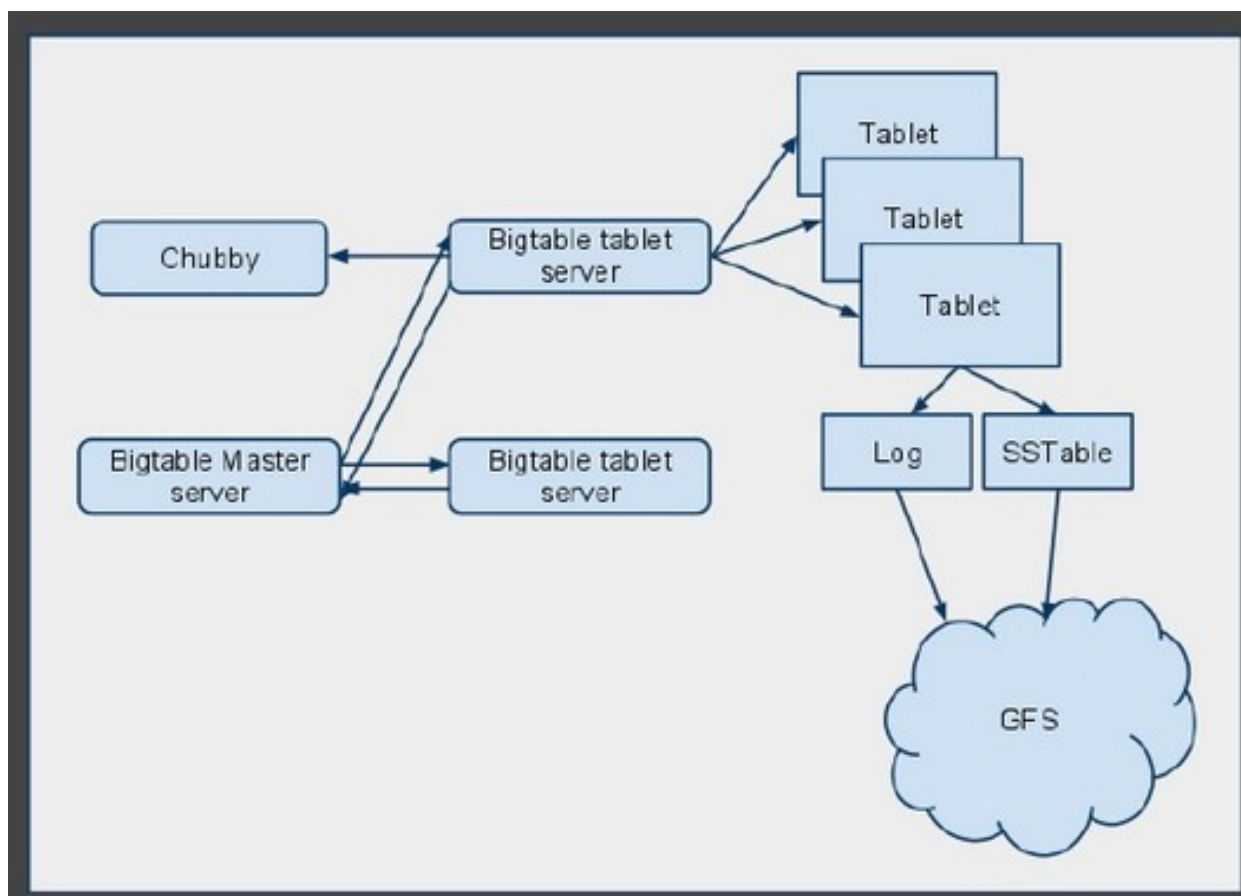


- a tablet is stored as a set of SSTables
- an SSTable has a set of 64K blocks and an index
- each SSTable is a GFS file

# Table Implementation



# Implementation



# APIs

- Metadata operations
  - Create/delete tables, column families, change metadata
- Writes
  - Set(): write cells in a row
  - DeleteCells(): delete cells in a row
  - DeleteRow(): delete all cells in a row
- Reads
  - Scanner: read arbitrary cells in a bigtable
    - Each row read is atomic
    - Can restrict returned rows to a particular range
    - Can ask for just data from 1 row, all rows, etc.
    - Can ask for all columns, just certain column families, or specific columns

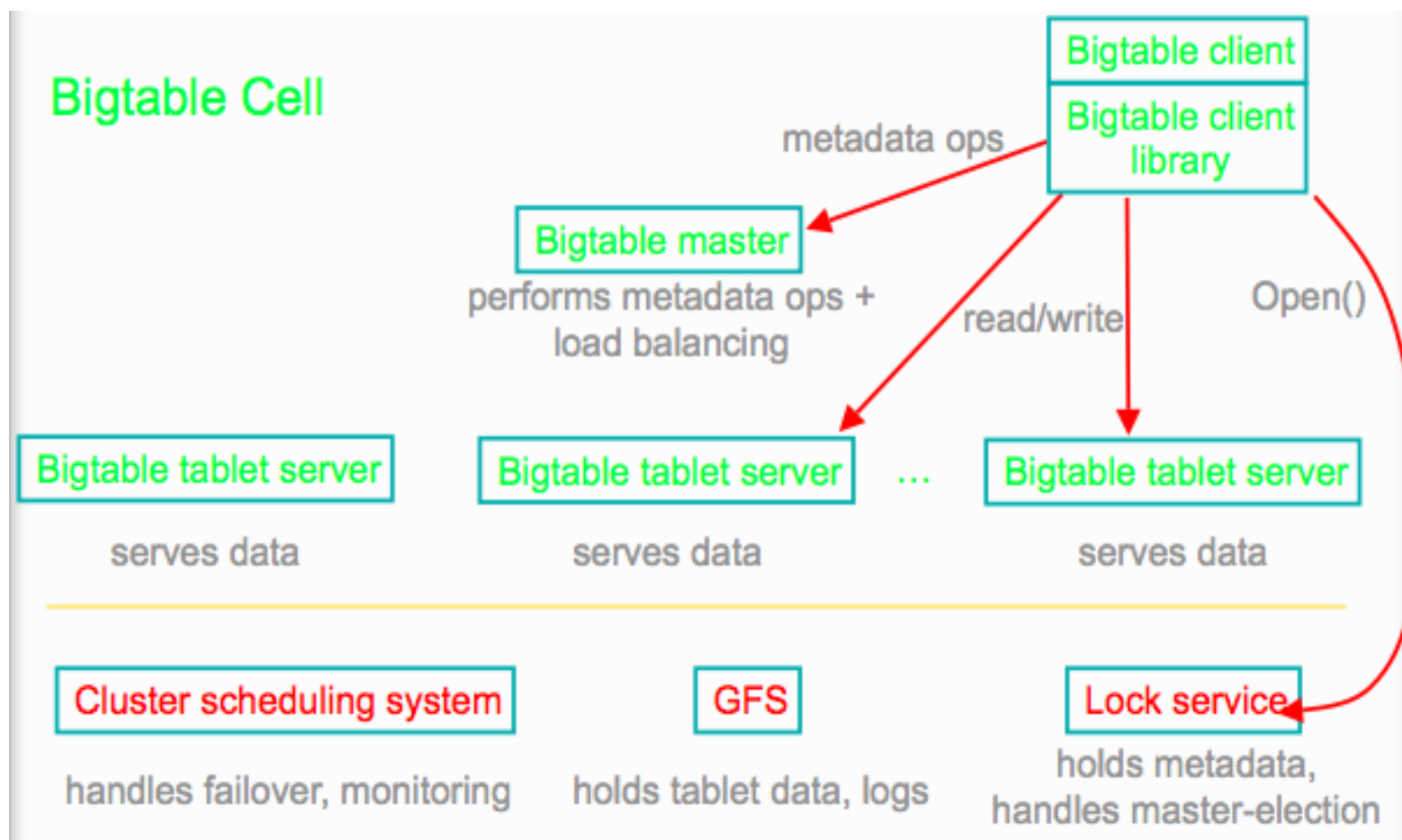
## Building Blocks

- Google File System (GFS)
  - stores persistent data (SSTable file format)
- Scheduler
  - schedules jobs onto machines
- Chubby
  - Lock service: distributed lock manager
  - master election, location bootstrapping
- MapReduce (optional)
  - Data processing
  - Read/write Bigtable data

# Implementation

- Single-master distributed system
- Three major components
  - Library that linked into every client
  - One master server
    - Assigning tablets to tablet servers
    - Detecting addition and expiration of tablet servers
    - Balancing tablet-server load
    - Garbage collection
    - Metadata Operations
  - Many tablet servers
    - Tablet servers handle read and write requests to its table
    - Splits tablets that have grown too large

# Implementation



## Chubby

- A persistent and distributed lock service.
  - Consists of 5 active replicas, one replica is the master and serves requests.
  - Service is functional when majority of the replicas are running and in communication with one another – when there is a quorum.
  - Implements a nameservice that consists of directories and files.
-



## Software Infrastructure

1. A Bigtable library linked to every client.
  2. Many tablet servers.
    - Tablet servers are added and removed dynamically.
    - Ten to a thousand tablets assigned to a tablet server.
    - Each tablet is typically 100-200 MB in size.
  3. One master server responsible for:
    - Assigning tablets to tablet servers,
    - Detecting the addition and deletion of tablet servers,
    - Balancing tablet-server load,
    - Garbage collection of files in GFS.
- □ Client communicates directly with tablet server for reads/writes.
-

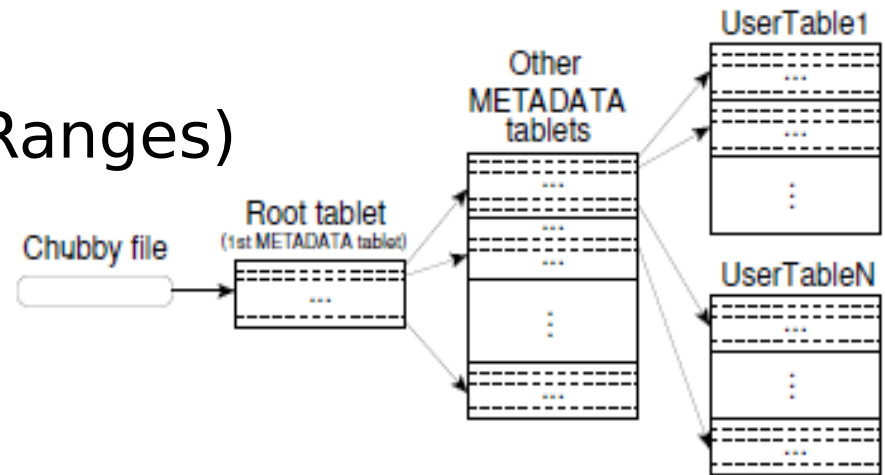
## BigTable & GFS

- BigTable is built on GFS, which it uses as a backing store both log and data files.
- GFS provides reliable storage for *SSTables*, which are used to persist table data.
- SSTables store data as a simple key->value map which can be looked up with a single disk access by searching an index (which is stored at the end of the SSTable) and then doing a read from the indexed location.
- Alternatively, SSTables can be completely ~~memory-mapped which avoids the disk read.~~

## Finding Tablets

- The location of a root tablet, which contains metadata for a BigTable instance, is located in Chubby and can be read from the filesystem there.
- The root tablet contains the locations of other metadata tablets, which themselves point to the location of data tablets.
- Together the root and metadata tablets form the METADATA table, which is itself a BigTable table.
- Each row in the METADATA table maps the pair (table name, last row) to a location for a tablet whose last row is as given.

## Location of Tablets (Ranges)



- 3 Level-B+ tree
- 1<sup>st</sup> Level: A file stored in chubby contains location of the root tablet, i.e., a directory of ranges (tablets) and associated meta-data.
  - The root tablet never splits.
- 2<sup>nd</sup> Level: Each meta-data tablet contains the location of a set of user tablets.
- 3<sup>rd</sup> Level: A set of SSTable identifiers for each tablet.
- Since this three-level lookup requires a lot of network round trips, clients cache the location of tablets.
- If a cached location is no longer valid, the client can go back to the network. To save yet further on round-trips, locations are speculatively pre-fetched by piggy-backing them on real location queries.

## Placement of Tablets

- A tablet is assigned to one tablet server at a time.
- Master maintains:
  - The set of live tablet servers,
  - Current assignment of tablets to tablet servers (including the unassigned ones)
- Chubby maintains tablet servers:
  - A tablet server creates and acquires an eXclusive lock on a uniquely named file in a specific chubby directory (named *server directory*),
  - Master monitors *server directory* to discover tablet server,
  - A tablet server stops processing requests if it loses its X lock (network partitioning).
    - Tablet server will try to obtain an X lock on its uniquely named file as long as it exists.
    - If the uniquely named file of a tablet server no longer exists then the tablet server kills itself. Goes back to a free pool to be assigned tablets by the master.

## Placement of Tablets

- Master detects when a tablet server is in the free pool.
  - How? Master periodically probes each tablet server for the status of its lock.

lock. If a tablet server reports that it has lost its lock, or if the master was unable to reach a server during its last several attempts, the master attempts to acquire an exclusive lock on the server's file. If the master is able to acquire the lock, then Chubby is live and the tablet server is either dead or having trouble reaching Chubby, so the master ensures that the tablet server can never serve again by deleting its server file. Once a server's file has been deleted, the master can move all the tablets that were previously assigned to that server into the set of unassigned tablets. To ensure that a Bigtable cluster is not vulnera-

## Master

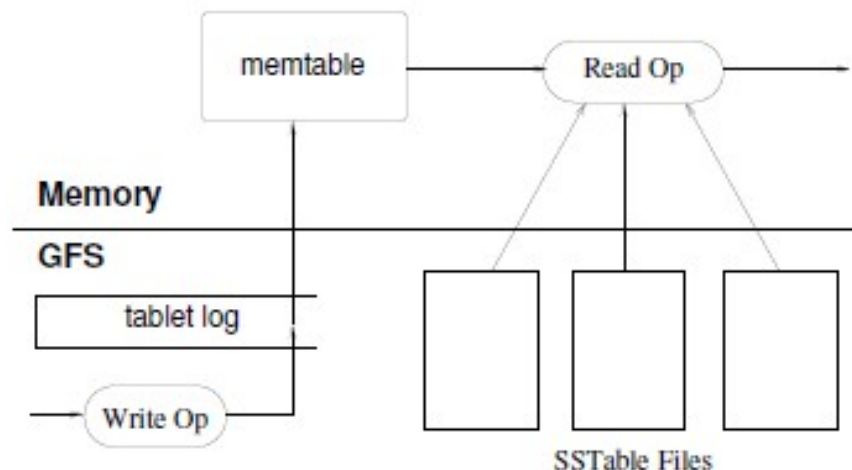
- Should the Master die, a new Master is initiated. The master executes the following steps:

the following steps at startup. (1) The master grabs a unique *master* lock in Chubby, which prevents concurrent master instantiations. (2) The master scans the servers directory in Chubby to find the live servers. (3) The master communicates with every live tablet server to discover what tablets are already assigned to each server. (4) The master scans the METADATA table to learn the set of tablets. Whenever this scan encounters a tablet that is not already assigned, the master adds the tablet to the set of unassigned tablets, which makes the tablet eligible for tablet assignment.

---

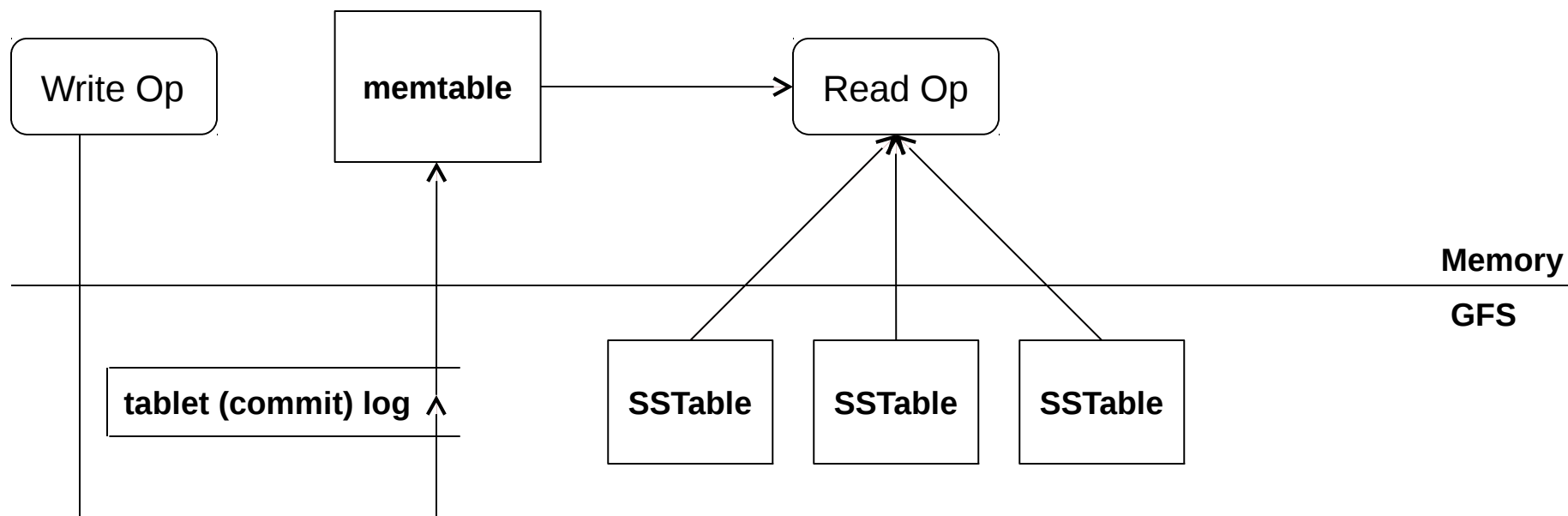
# Client Write & Read Operations

- Write operation arrives at a tablet server:
  - Server ensures the client has sufficient privileges for the write operation (Chubby),
  - A log record is generated to the commit log file,
  - Once the write commits, its contents are inserted into the memtable.
- Read operation arrives at a tablet server:
  - Server ensures client has sufficient privileges for the read operation (Chubby),
  - Read is performed on a merged view of (a) the SSTables that constitute the tablet, and (b) the memtable.





# Tablet operations



- Updates are written in a memory table after being recorded in a log
- Reads combine information in the memtable with that in the SSTables

# Refinement – Locality groups & Compression

## ■ Locality Groups

- Can group multiple column families into a *locality group*
  - Separate SSTable is created for each locality group in each tablet.
- Segregating columns families that are not typically accessed together enables more efficient reads.
  - In WebTable, page metadata can be in one group and contents of the page in another group.

## Write Operations

- As writes execute, size of memtable increases.
  - Once memtable reaches a threshold:
    - Memtable is frozen,
    - A new memtable is created,
    - Frozen memtable is converted to an SSTable and written to GFS.
  - This minimizes memory usage of tablet server, and reduces recovery time in the presence of crashes (checkpoints).
  - (in the background) reads a few SSTables and memtable to produce one SSTable. (Input SSTables and memtable are discarded.)
  - rewrites all SSTables into exactly one SSTable (containing no deletion entries).
-

# Compactions

## ■ Minor compaction

- Converts the memtable into an SSTable
- Reduces memory usage and log traffic on restart

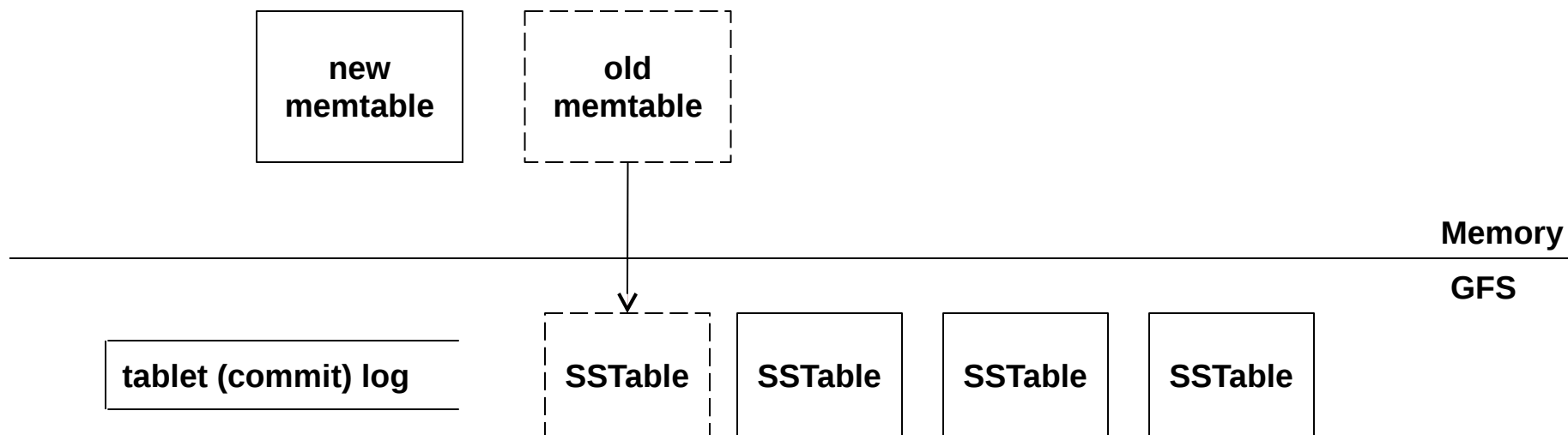
## ■ Merging compaction

- Reads the contents of a few SSTables and the memtable, and writes out a new SSTable
- Reduces number of SSTables

## ■ Major compaction

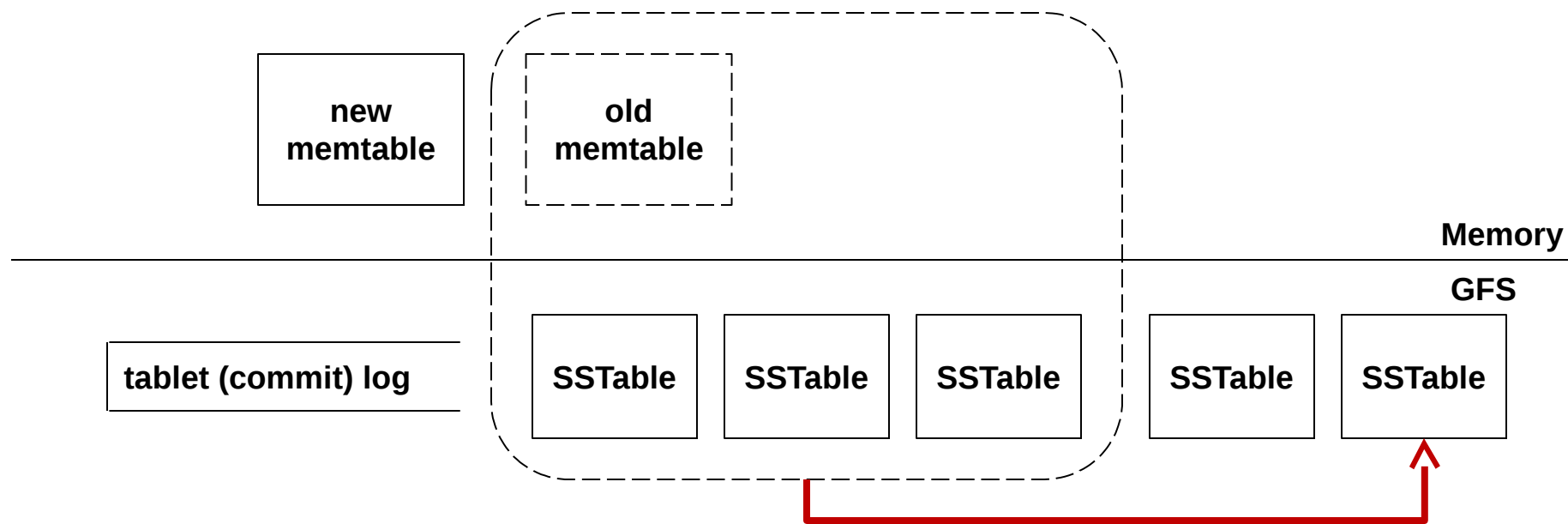
- Merging compaction that results in only one SSTable
  - No deletion records, only live data
-

# Minor compaction



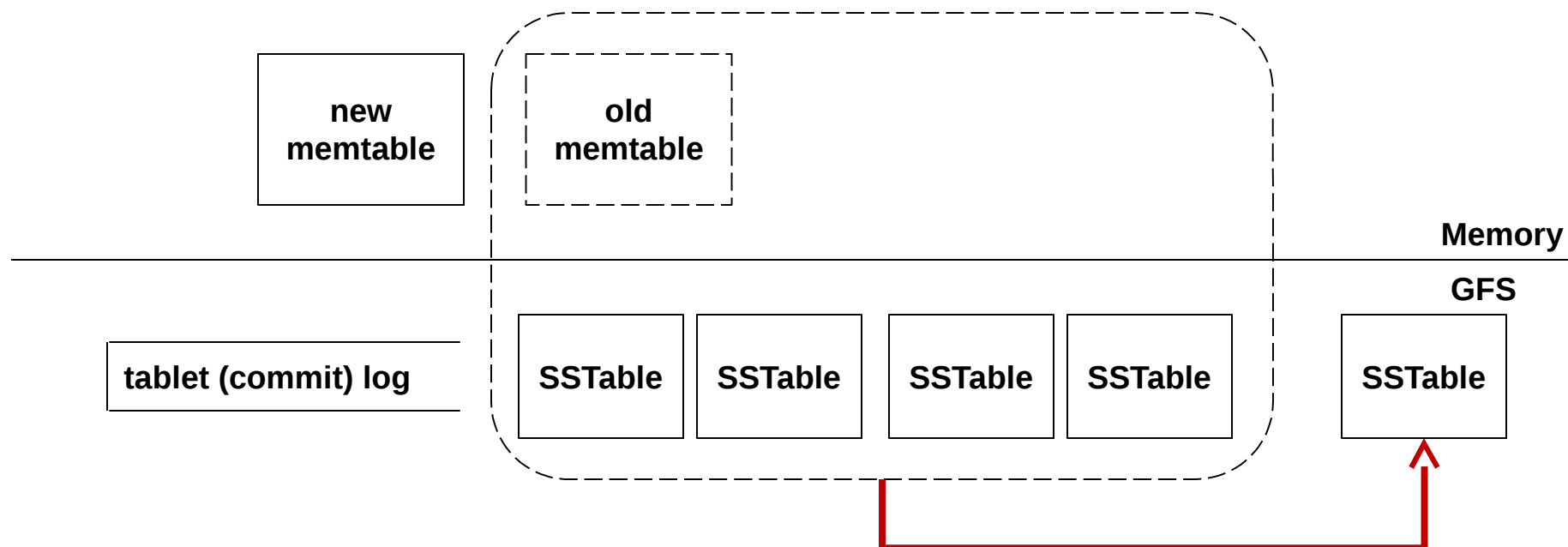
- Triggered when memtable reaches a threshold
- Reduces memory footprint
- Reduces data read from commit log on recovery from failure
- Read/write operations continue during compaction

# Merging compaction



- Compacts existing memtable and some number of SSTables into a single new SSTable
- Used to control number of SSTables that must be scanned to perform operations
- Old memtable and SSTables are discarded at end of compaction

# Major compaction



- Compacts existing memtable and **all** SSTables into a single SSTable