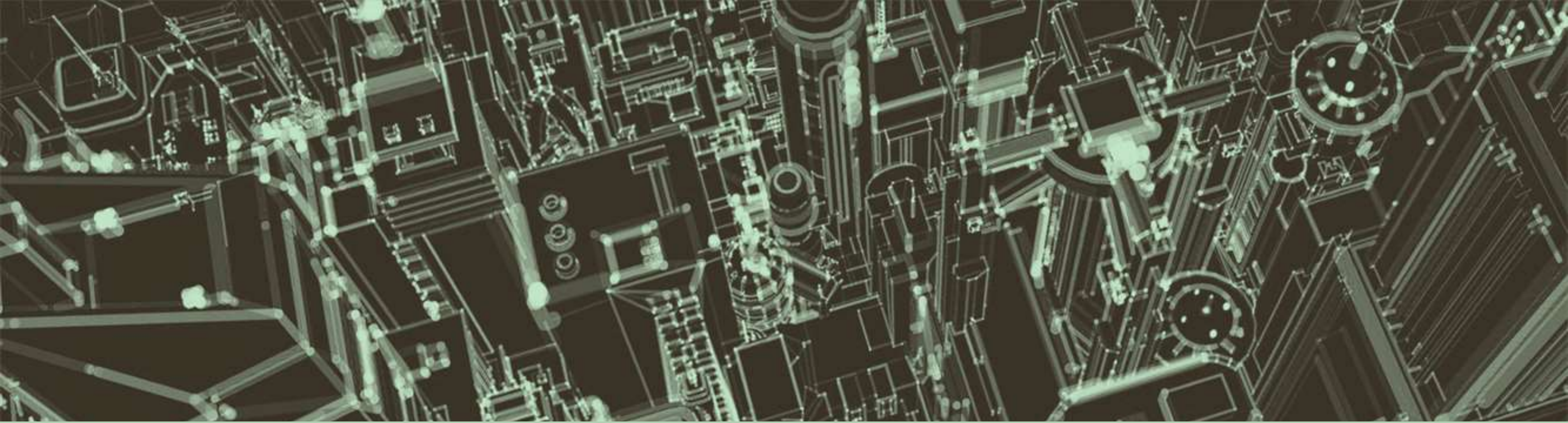


# Google BigTable

DS Seminar

23-05-2016

# Agenda

- A Short history of Google Motivation
- What is GFS ?
- What is Chubby ?
- What is Map Reduce ?
- What is BigTable ?
    - Data Model
    - API
    - Building Blocks
    - Implementation

# Google Motivation

❑ Undoubtedly, Google has **a lot of data**.

❑ Scale of data is **too large**

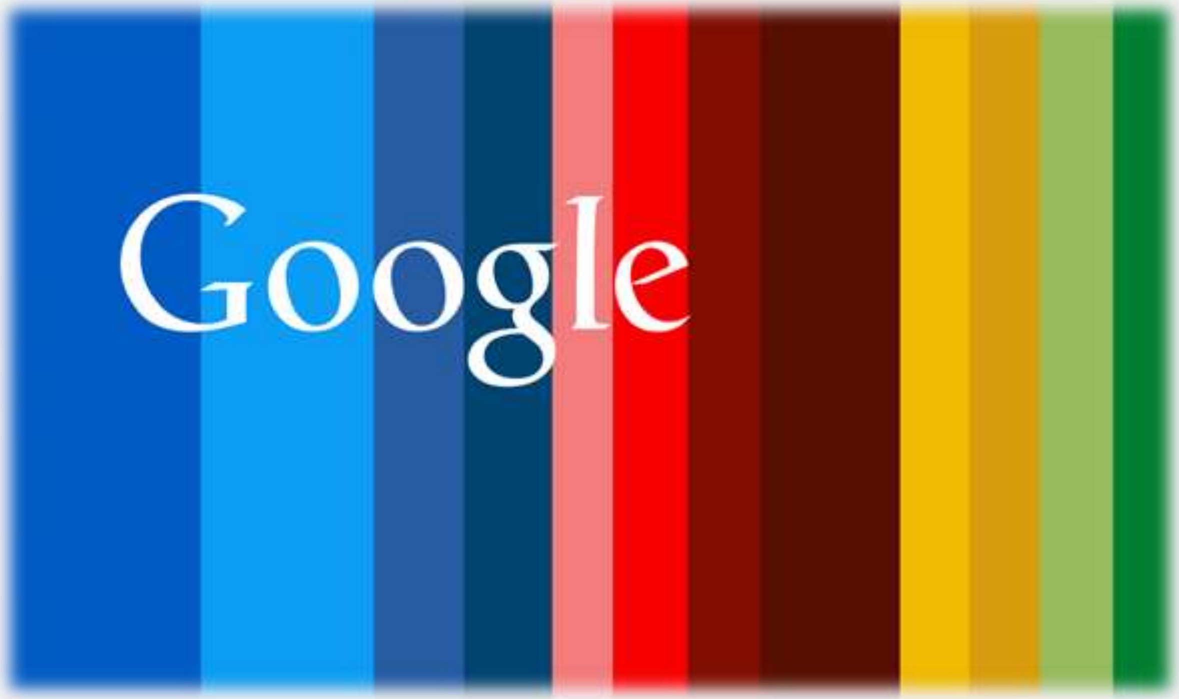
Even for commercial databases.

Even though **Google** is *best known* for it's **reliable** and **fast services**, but what's there working behind the scene?

# Google Motivation

- Undoubtedly, there are number of aspects that matter behind this (like **Hardware**, **Software**, **Operating System**, **Best Staff** in the world etc. )
- But, What I am going to explain here is the Software part.

- GFS
- Chubby
- Map Reduce
- BigTable



# What is GFS ?

- ❑ **GFS** stands for **G**oogle **F**ile **S**ystem.

- ❑ It's a **Proprietary**(means for their personal use, not open source) **distributed file system** developed by Google for their services.

- ❑ It is specially designed to provide **efficient**, **reliable** access to data using **large clusters** of **commodity hardware**, means they are using  low cost hardware, not state-of-the-art computers. Google uses relatively inexpensive computers running Linux Operating System and the GFS works just fine with them !

# What is Chubby ?

❑ Chubby is a **Lock Service**. (It's related to gain access of Shared resources)

❑ It is used to **synchronize accesses** to shared resources.

❑ It is now used as a replacement of Google's Domain Name System.

# What is Map Reduce ?

- ❑ MapReduce is a software framework that process **massive** amounts of unstructured data.

- ❑ It allows developers to write programs that process data in **parallel** across a distributed cluster of processors or stand-alone computers.

- ❑ It is now used by Google mainly for their **Web Indexing** Service, applied since 2004.

- ❑ **Map()** procedure performs all the process related to **Filtering** and **Sorting**.
- ❑ **Reduce()** procedure performs all the **Summary** related operations.

# What is BigTable?

- ❑ BigTable is a **distributed storage system** for **managing structured data** built on

 Google File System, Chubby Lock Service, SSTable (log-structured storage like LevelDB) and a few other Google technologies.

- ❑ **Designed to Scale** to a very large size: petabytes of data across thousands of commodity servers

- ❑ Most important point, It's a **Non-Relational Database**.

- ❑ It uses amazing **Load Balancing Structure** so that it runs on Commodity Hardware.

- ❑ It uses **Snappy** compression utility for compacting the data.

# What is BigTable?

□ Distributed

□ Column – Oriented

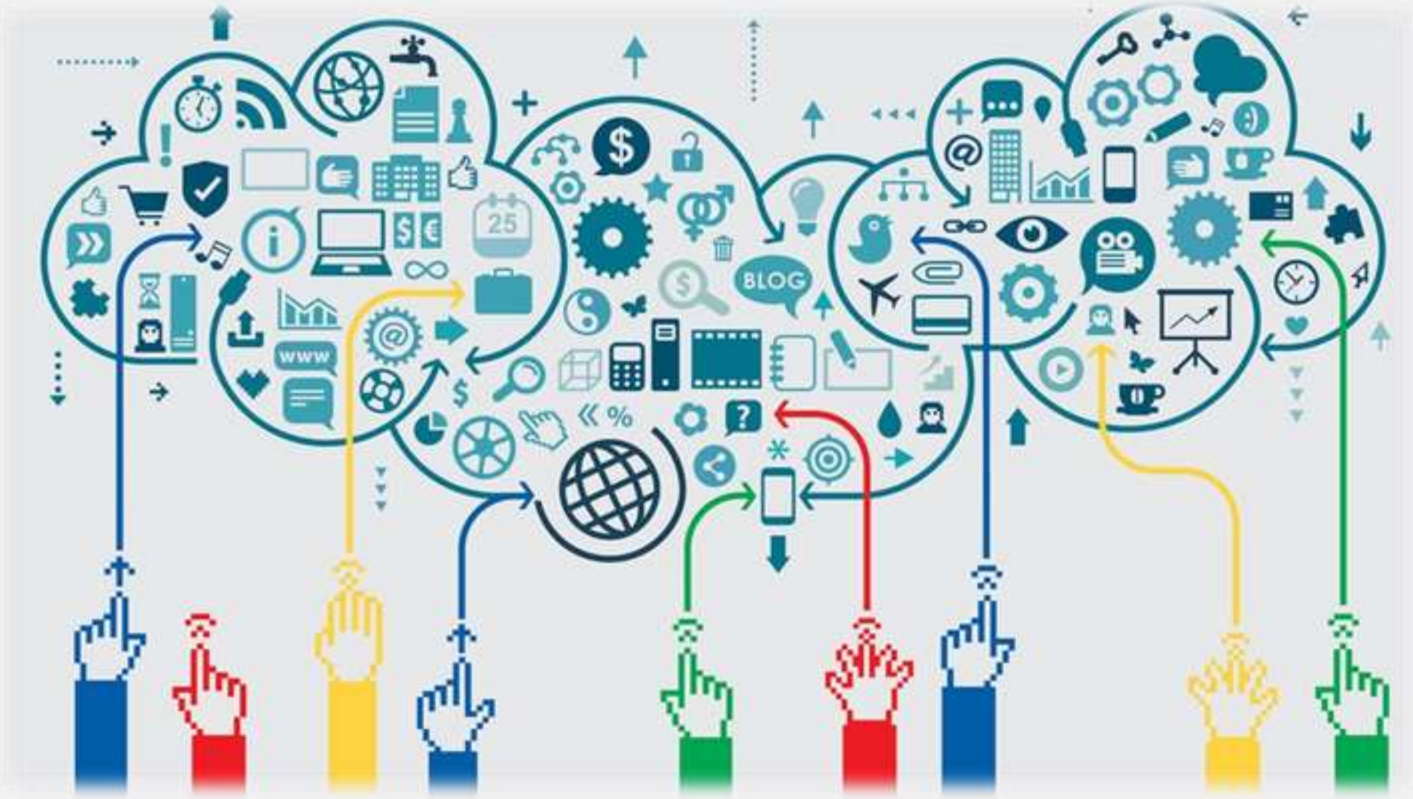
□ Multidimensional

□ High Availability

□ High Performance

□ Store System

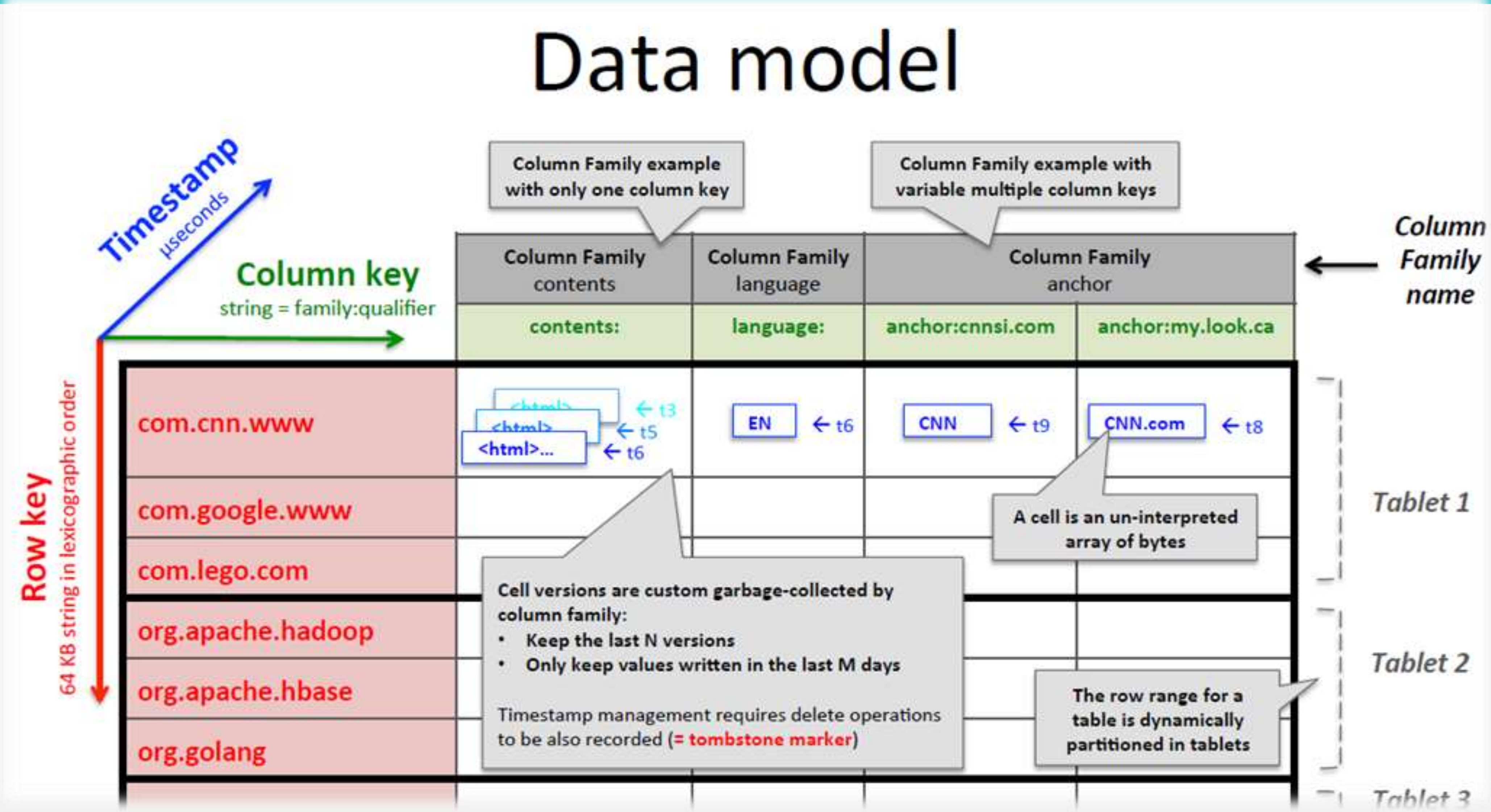
□ Self-managing



# What is BigTable?

- Used by more than 60 Google products
  - Google Analytics
  - Google Finance
  - Personalized Search
  - Google Documents
  - Google Earth
  - Google Fusion Tables
  - …
- Used for variety of demanding workloads
  - Throughput oriented batch processing
  - Latency sensitive data serving

# Data Model



## Data model

Timestamp
µseconds

Column key
string = family:qualifier

Row key
64 KB string in lexicographic order

Column Family example with only one column key

Column Family example with variable multiple column keys

Column Family name

| | Column Family contents | Column Family language | Column Family anchor | |
|---|---|---|---|---|
| | contents: | language: | anchor:cnnsi.com | anchor:my.look.ca |
| com.cnn.www | <html> ← t3<br><html> ← t5<br><html>... ← t6 | EN ← t6 | CNN ← t9 | CNN.com ← t8 |
| com.google.www | | | | |
| com.lego.com | | | | |
| org.apache.hadoop | | | | |
| org.apache.hbase | | | | |
| org.golang | | | | |

A cell is an un-interpreted array of bytes

Cell versions are custom garbage-collected by column family:

- Keep the last N versions
- Only keep values written in the last M days

Timestamp management requires delete operations to be also recorded (= tombstone marker)

The row range for a table is dynamically partitioned in tablets

Tablet 1

Tablet 2

Tablet 3

# Data Model

- ❑ Each table is a **Multi-Dimensional Sparse Map** ( Memory Efficient hash-map implementation).

- ❑ The table consists of

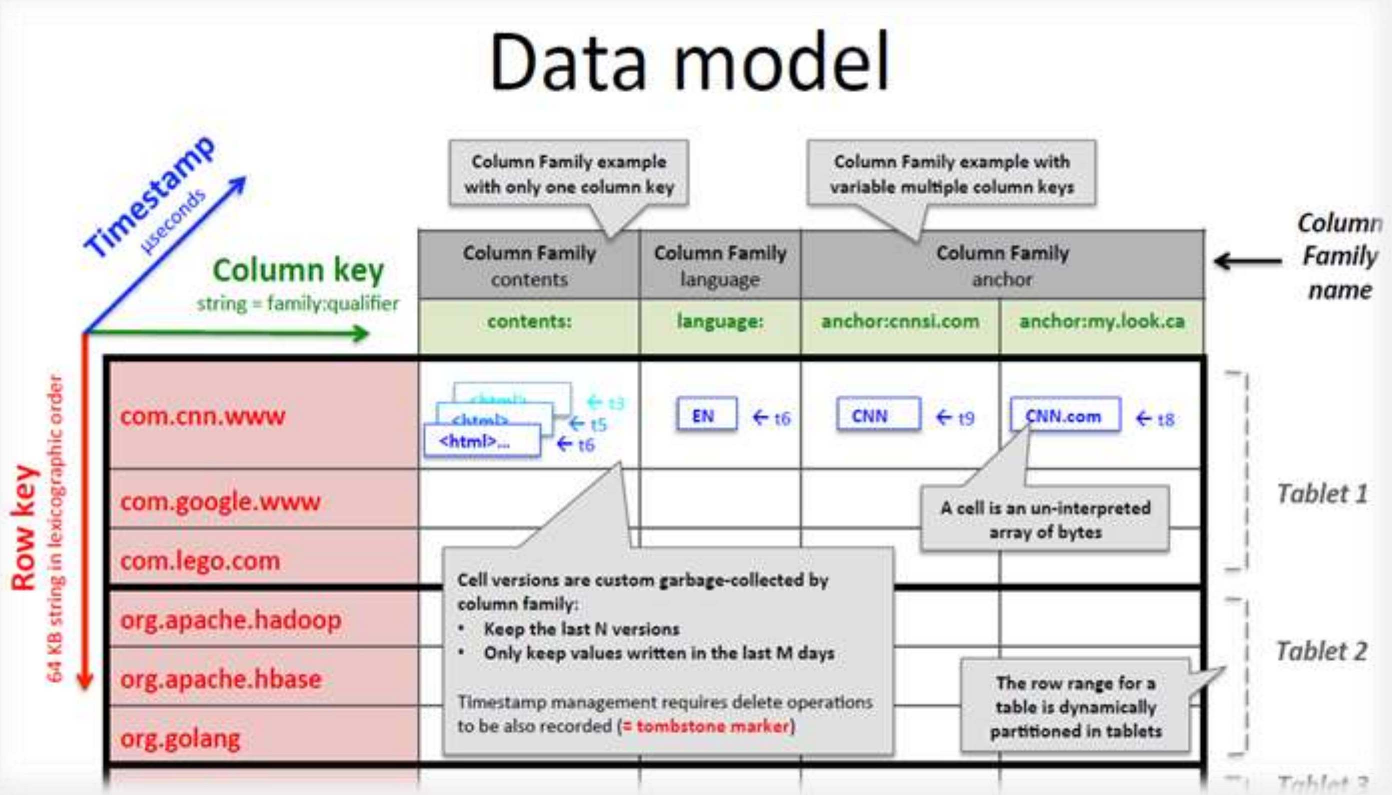
  (1) Rows,

  (2) Columns

  (3) Each cell has a Time Version

  (Time-Stamp).

- ❑ Time Version results in multiple copies of each cell with different times,



## Data model

Column Family example with only one column key

Column Family example with variable multiple column keys

A cell is an un-interpreted array of bytes

Cell versions are custom garbage-collected by column family:

- Keep the last N versions
- Only keep values written in the last M days

Timestamp management requires delete operations to be also recorded (= tombstone marker)

The row range for a table is dynamically partitioned in tablets

# Data Model

❑ Time Version results in multiple copies of each cell with different times, resulting **Unimaginable Redundancy** which is requirement for Google services, so don't ever think it as a drawback of this system.

❑ Google does Web Indexing to get the data of all the websites. They store all the URLs, their titles, time-stamp and many more required fields

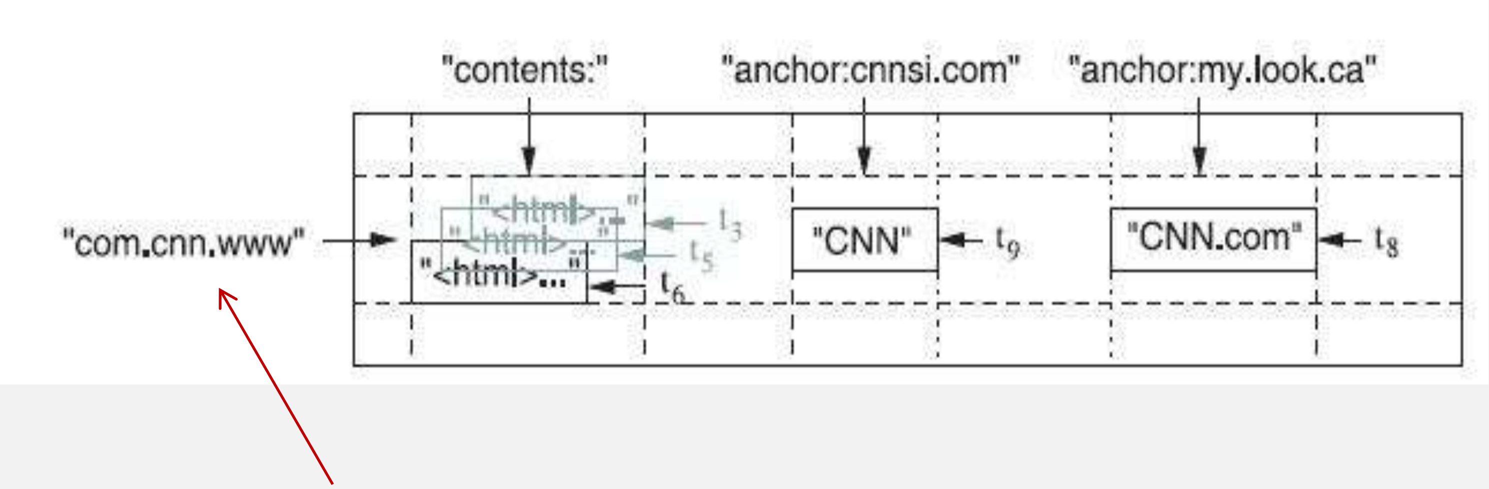
❑ Web Indexing :- indexing the contents of a website

# Data Model-Row

□ The **row keys** in a table are **arbitrary strings**.

□ Data is maintained in **lexicographic order** by row key

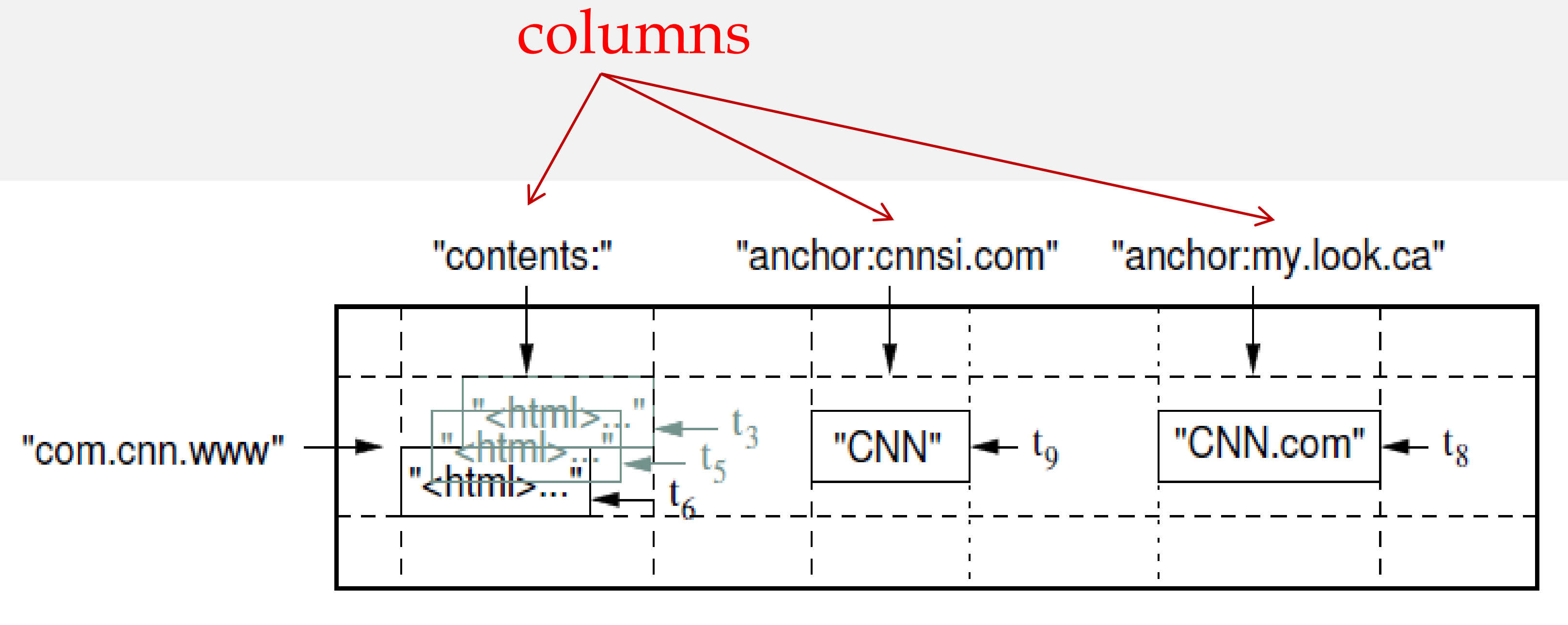
□ Each **row range** is called **a tablet**, which is the unit of distribution and load balancing.



row

# Data Model-Column

- **Column keys** are **grouped into** sets called **column families**.
- Data stored in a column family is usually of the same type
- A column key is named using the syntax: family : qualifier.
- Column family names must be printable , but qualifiers may be arbitrary strings.

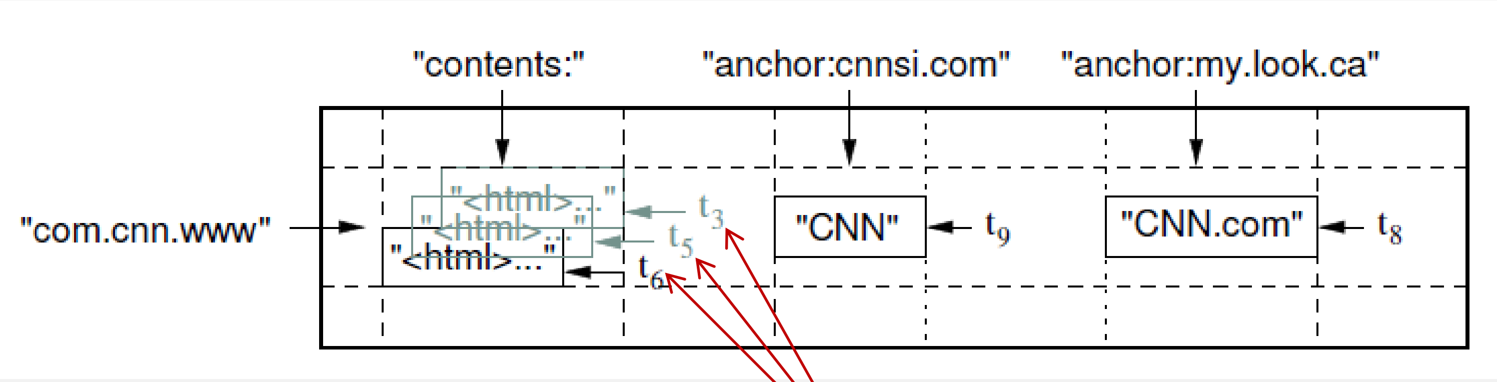


# Data Model-Timestamp

- Each cell in a Bigtable  can contain multiple versions of the same data

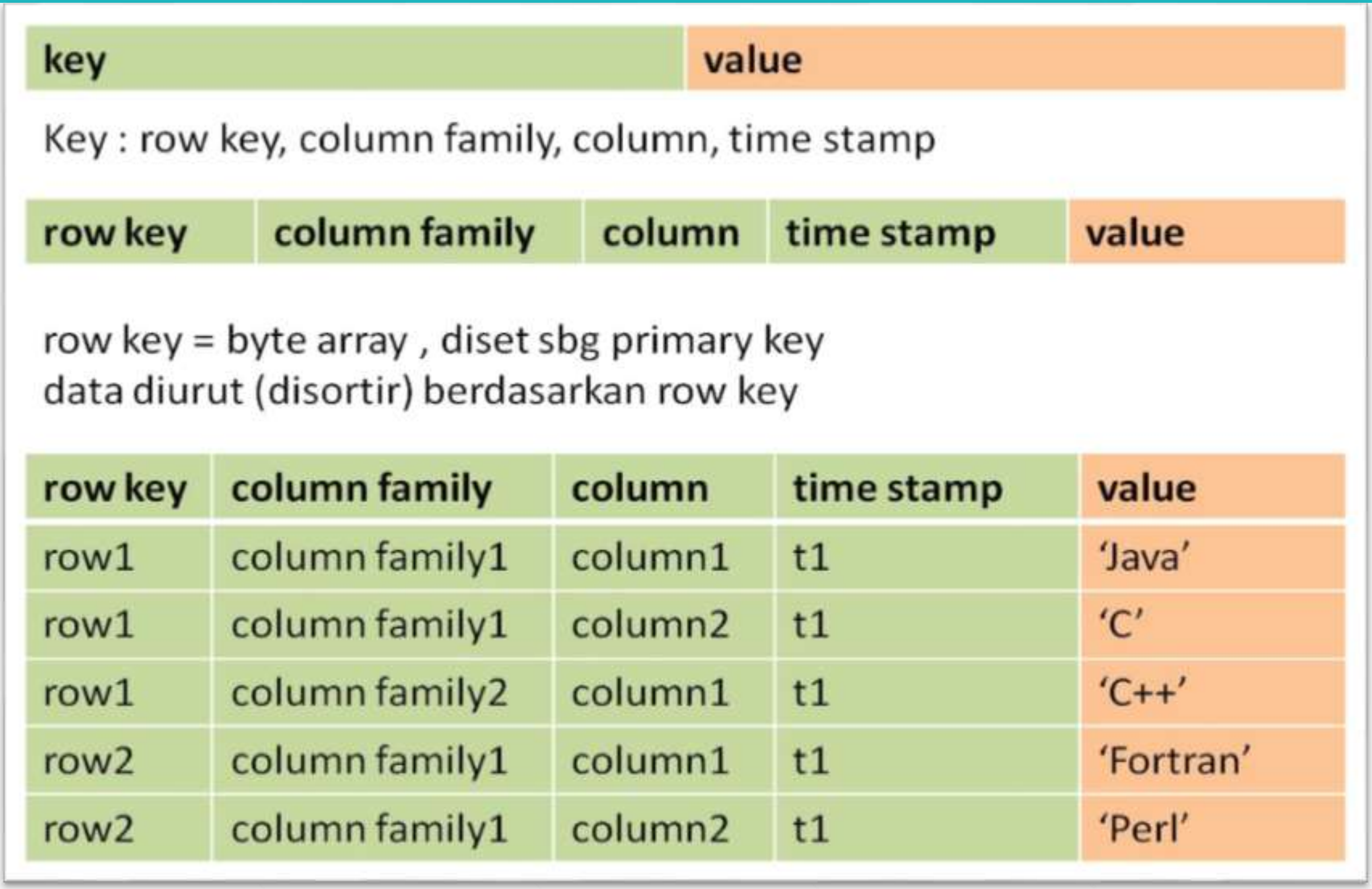
- Versions are indexed by **64-bit integer** timestamps

- Timestamps can be assigned:
  - automatically by Bigtable , or
  - explicitly by client applications



Timestamp

# Data Model



| key | value |
|---|---|

Key : row key, column family, column, time stamp

| row key | column family | column | time stamp | value |
|---|---|---|---|---|

row key = byte array , diset sbg primary key
data diurut (disortir) berdasarkan row key

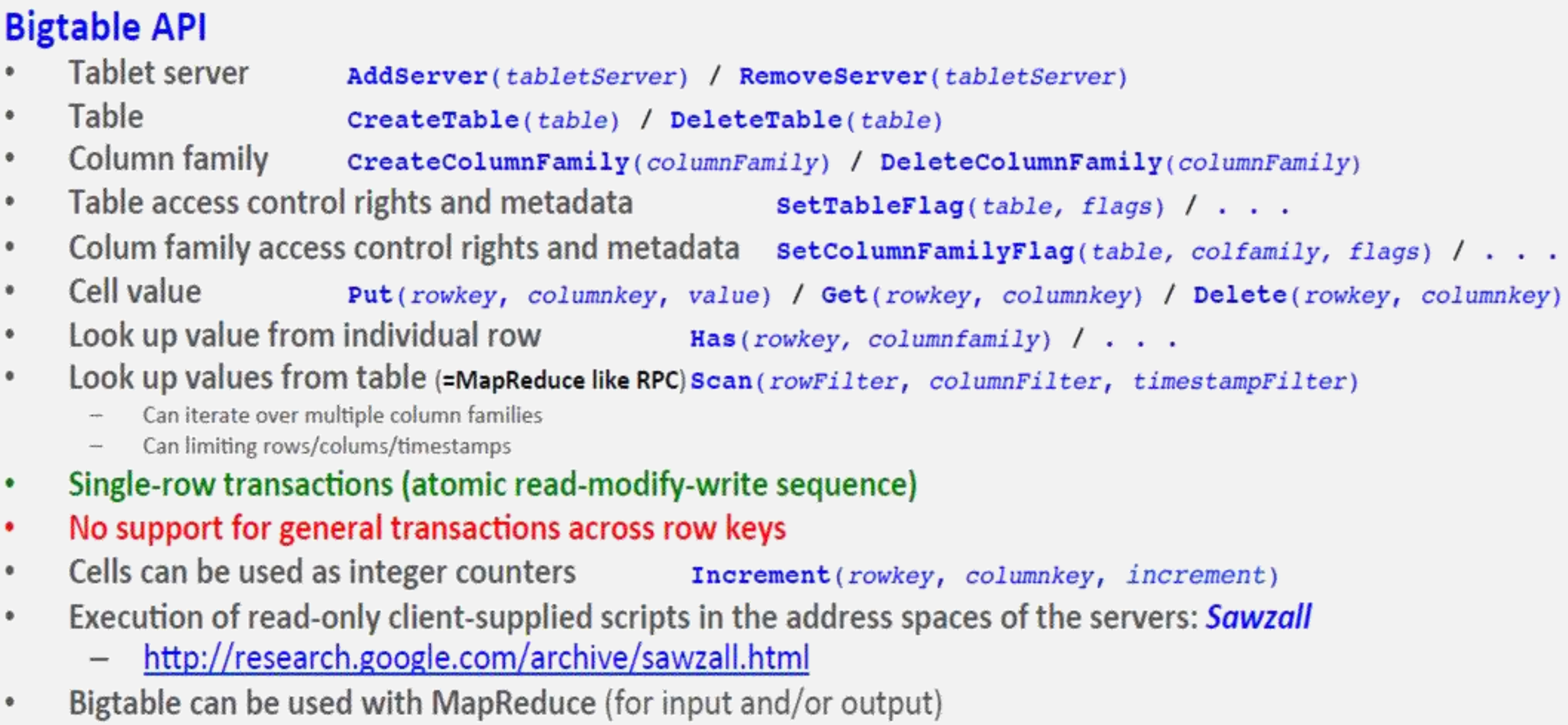
| row key | column family | column | time stamp | value |
|---|---|---|---|---|
| row1 | column family1 | column1 | t1 | 'Java' |
| row1 | column family1 | column2 | t1 | 'C' |
| row1 | column family2 | column1 | t1 | 'C++' |
| row2 | column family1 | column1 | t1 | 'Fortran' |
| row2 | column family1 | column2 | t1 | 'Perl' |

# API

❑The Bigtable API provides functions :

- Creating and deleting tables and column families.
- Changing cluster , table and column family metadata.
- Support for single row transactions
- Allows cells to be used as integer counters
- Client supplied scripts can be executed in the  address space of servers
- Input and output of Map/Reduce jobs

# API

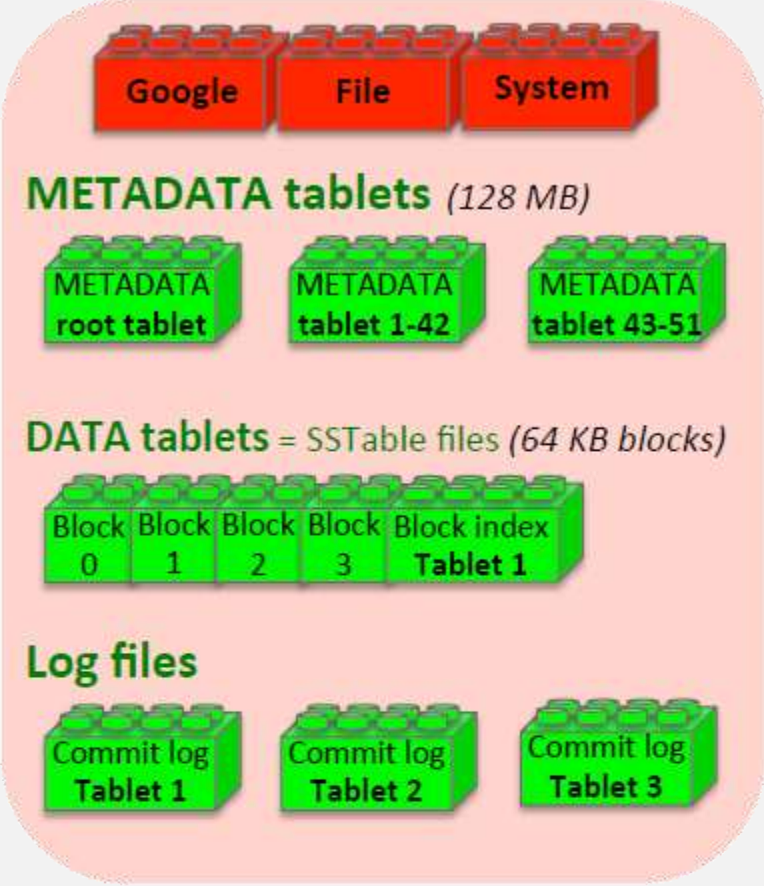


## Bigtable API

- Tablet server `AddServer(tabletServer) / RemoveServer(tabletServer)`
- Table `CreateTable(table) / DeleteTable(table)`
- Column family `CreateColumnFamily(columnFamily) / DeleteColumnFamily(columnFamily)`
- Table access control rights and metadata `SetTableFlag(table, flags) / . . .`
- Colum family access control rights and metadata `SetColumnFamilyFlag(table, colfamily, flags) / . . .`
- Cell value `Put(rowkey, columnkey, value) / Get(rowkey, columnkey) / Delete(rowkey, columnkey)`
- Look up value from individual row `Has(rowkey, columnfamily) / . . .`
- Look up values from table (=MapReduce like RPC) `Scan(rowFilter, columnFilter, timestampFilter)`
  - Can iterate over multiple column families
  - Can limiting rows/colums/timestamps
- Single-row transactions (atomic read-modify-write sequence)
- No support for general transactions across row keys
- Cells can be used as integer counters `Increment(rowkey, columnkey, increment)`
- Execution of read-only client-supplied scripts in the address spaces of the servers: *Sawzall*
  - http://research.google.com/archive/sawzall.html
- Bigtable can be used with MapReduce (for input and/or output)

# Building Blocks

- BigTable is composed of several other innovative, distribution oriented components.
  - Google File System (GFS)
  - SSTable
  - Chubby



# Building Blocks

- ❑ Google File System (GFS) :

  Used to store log and data files

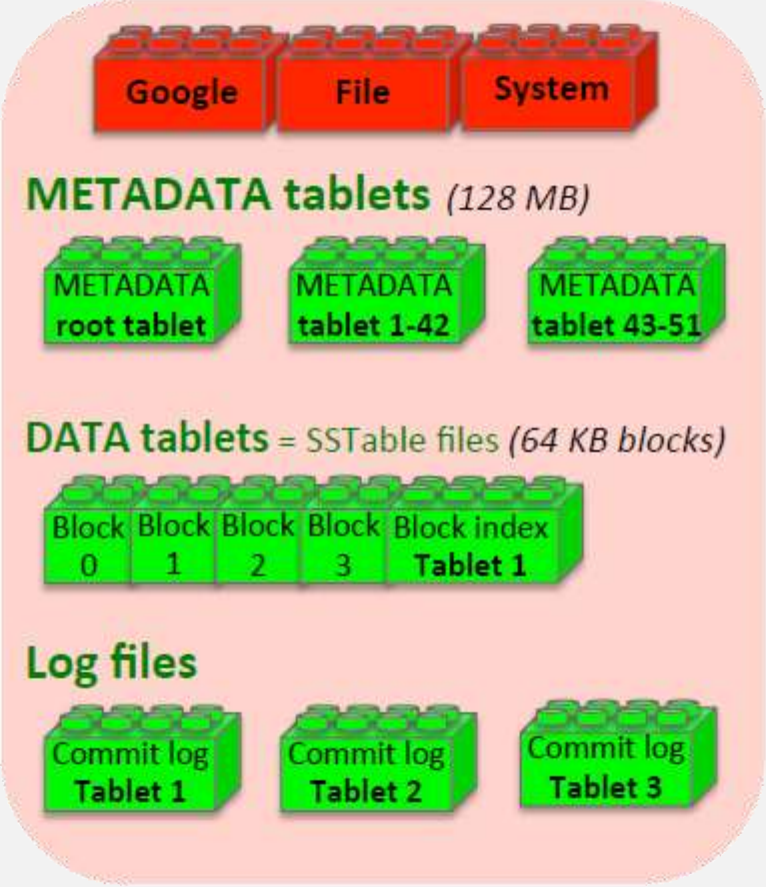
- ❑ SSTable (Sorted String Table) :

  Used to store table data in GFS

used to **store** and **retrieve** the pairs <Key, Value>

Used as **pointers** to pairs <Key, Value>

Stored in GFS



# Building Blocks

❑ Google Chubby :

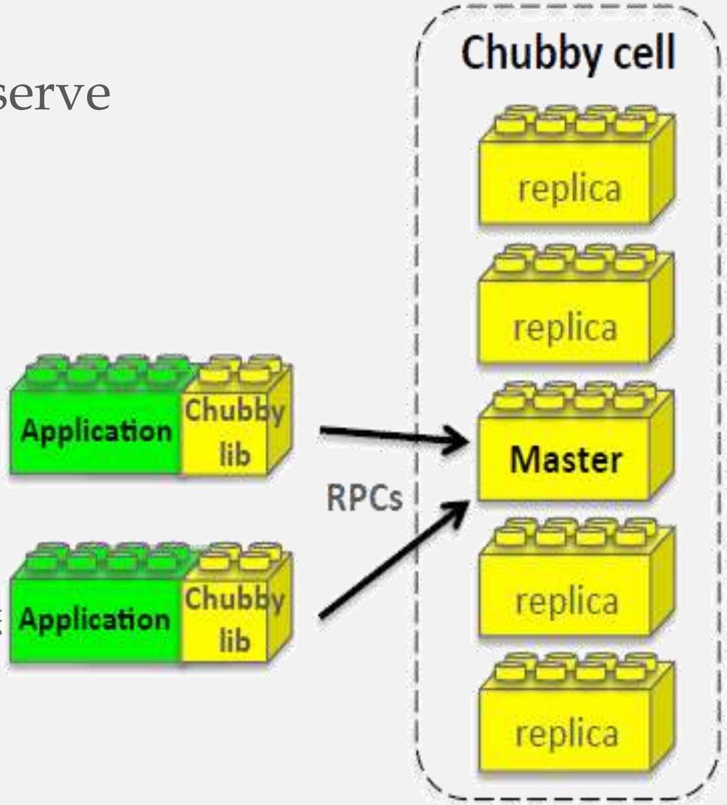
- Chubby is High available and persistent distributes service
- Chubby service consists of **5 active replicas** with one master to serve requests

➢ Each directory or file can be used as a lock

➢ Reads and writes to a file are atomic

➢ Chubby client library provides consistent caching of Chubby file

➢ Each Chubby client maintains a session with a Chubby service



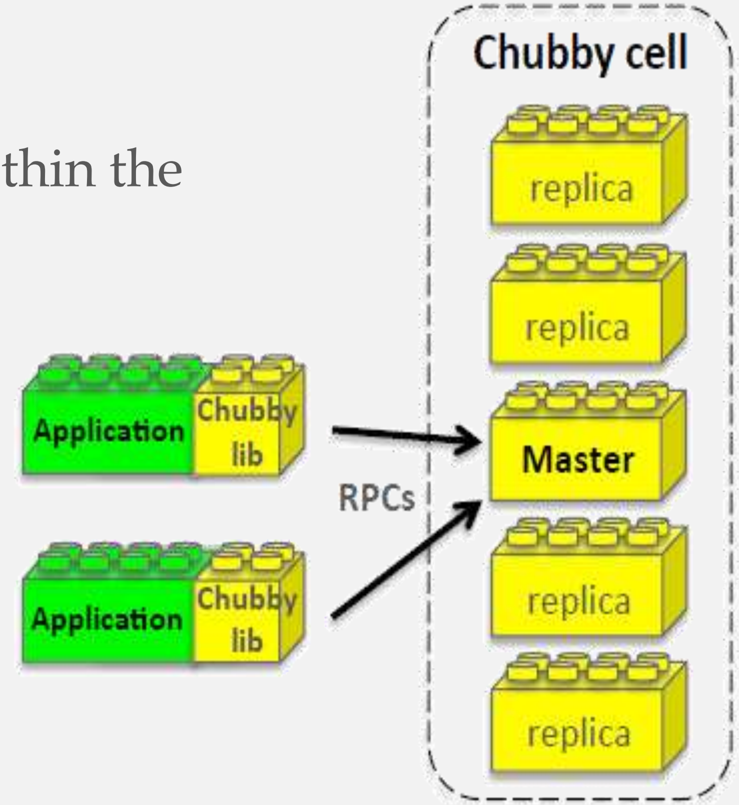
# Building Blocks

- Google Chubby :

- Client's session expires if is unable to renew its session lease within the lease expiration time

- When a client's session expires, it loses any locks and open handles

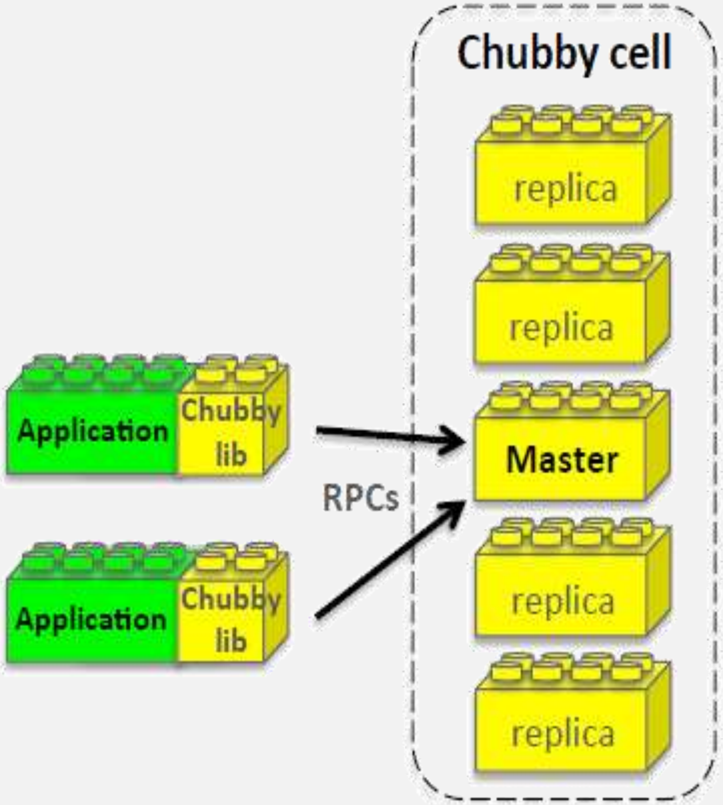
- Chubby clients can also register callbacks on Chubby files and directories for notification of changes or session expiration



# Building Blocks

- BigTable uses Chubby for a variety of tasks

- To ensure there is at most one active master at any time
- To store the bootstrap location of BigTable data (Root tablet)
- To discover tablet servers and finalize tablet server deaths
- To  store BigTable schema information (column family information for each table)
- To store access control lists (ACL)
- Chubby unavailable = BigTable unavailable



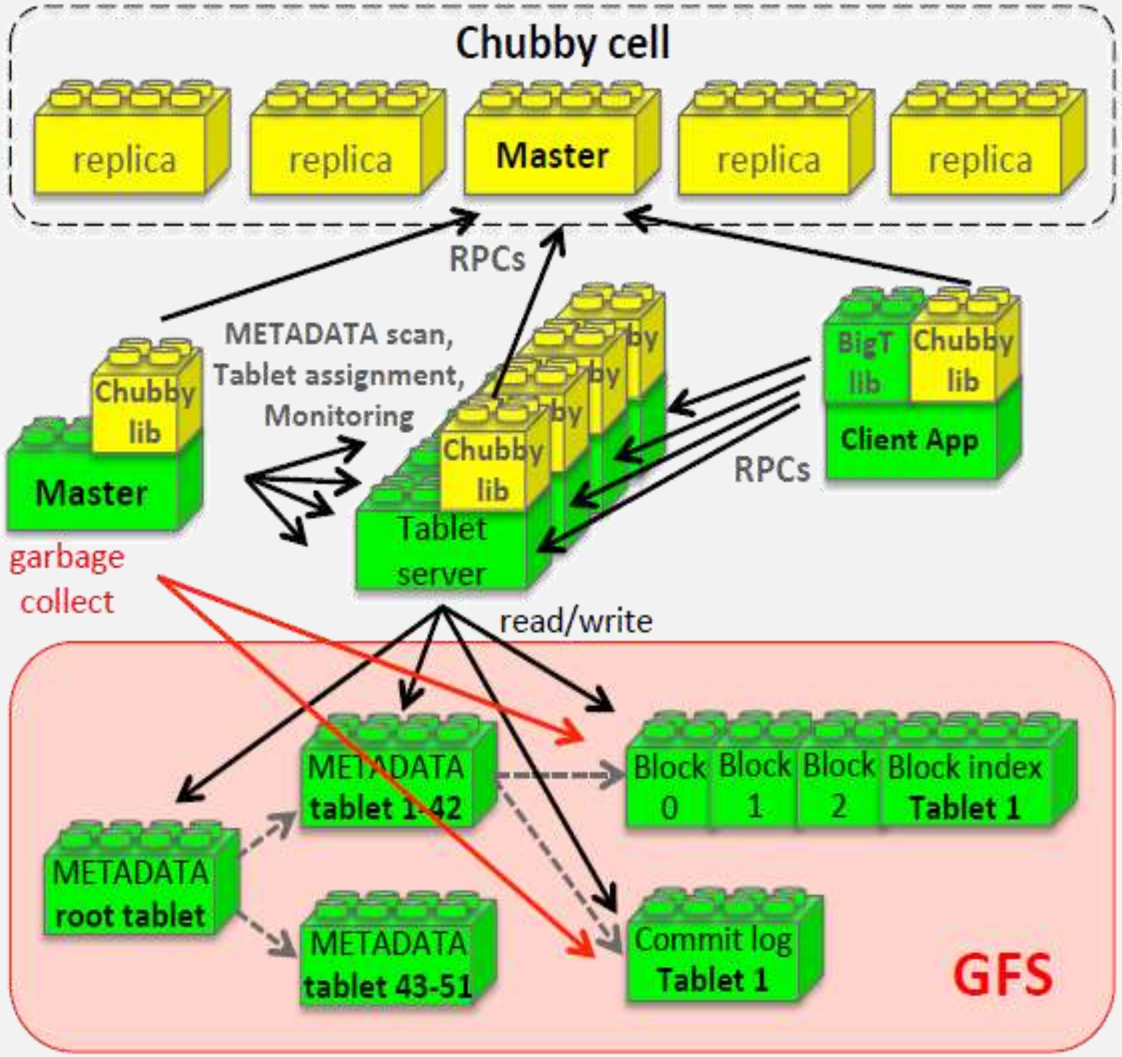
# Implementation

- The implementation has three major components
  - **One Master server**
  - **Many tablet servers**
  - **A library that is linked into every client**

- BigTable runs over Google File System

- BigTable is store in a structure called SSTable.

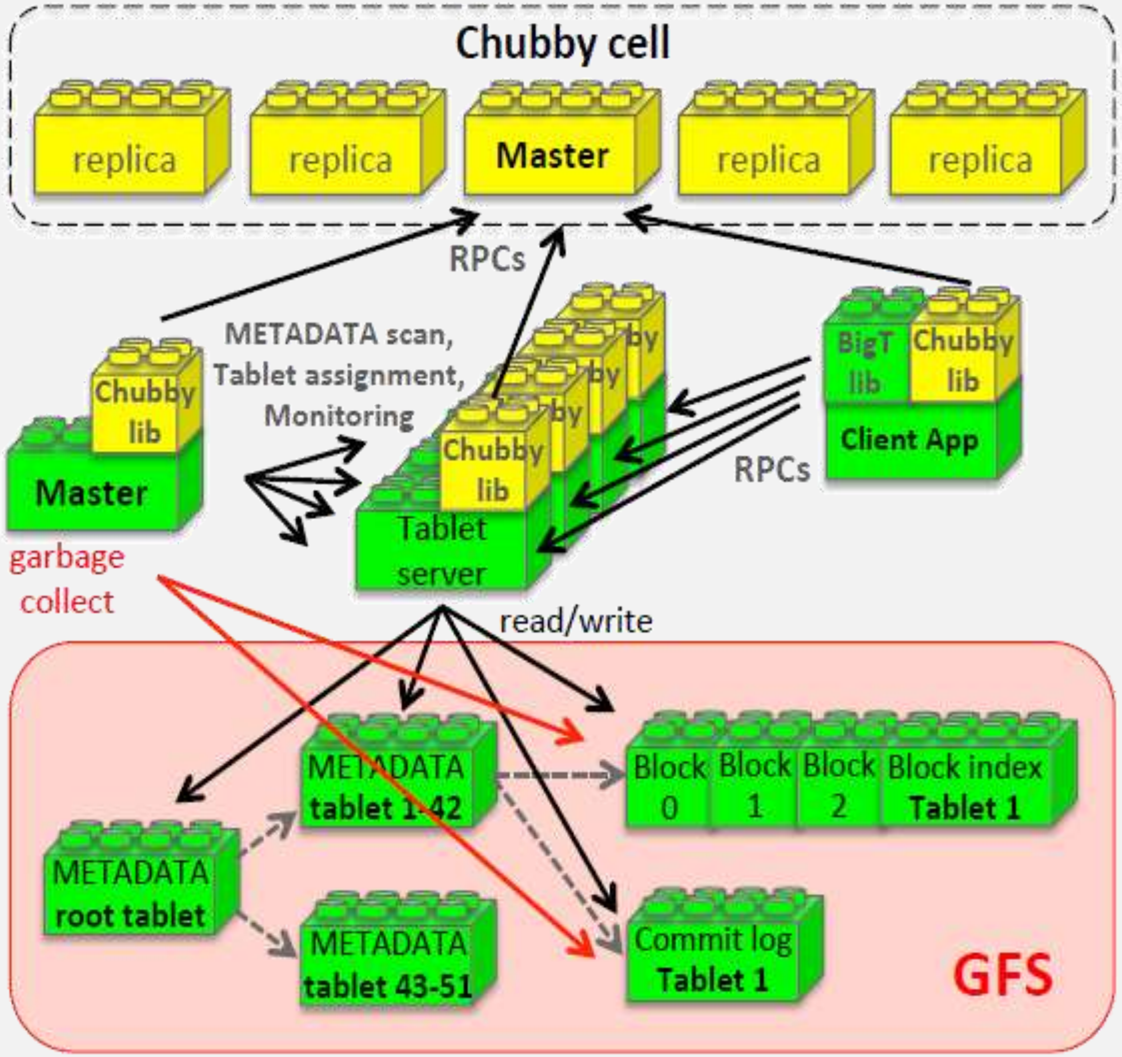
Each SSTable is divided into 64KB blocks. A SSTable can be loaded to Memory



# Implementation

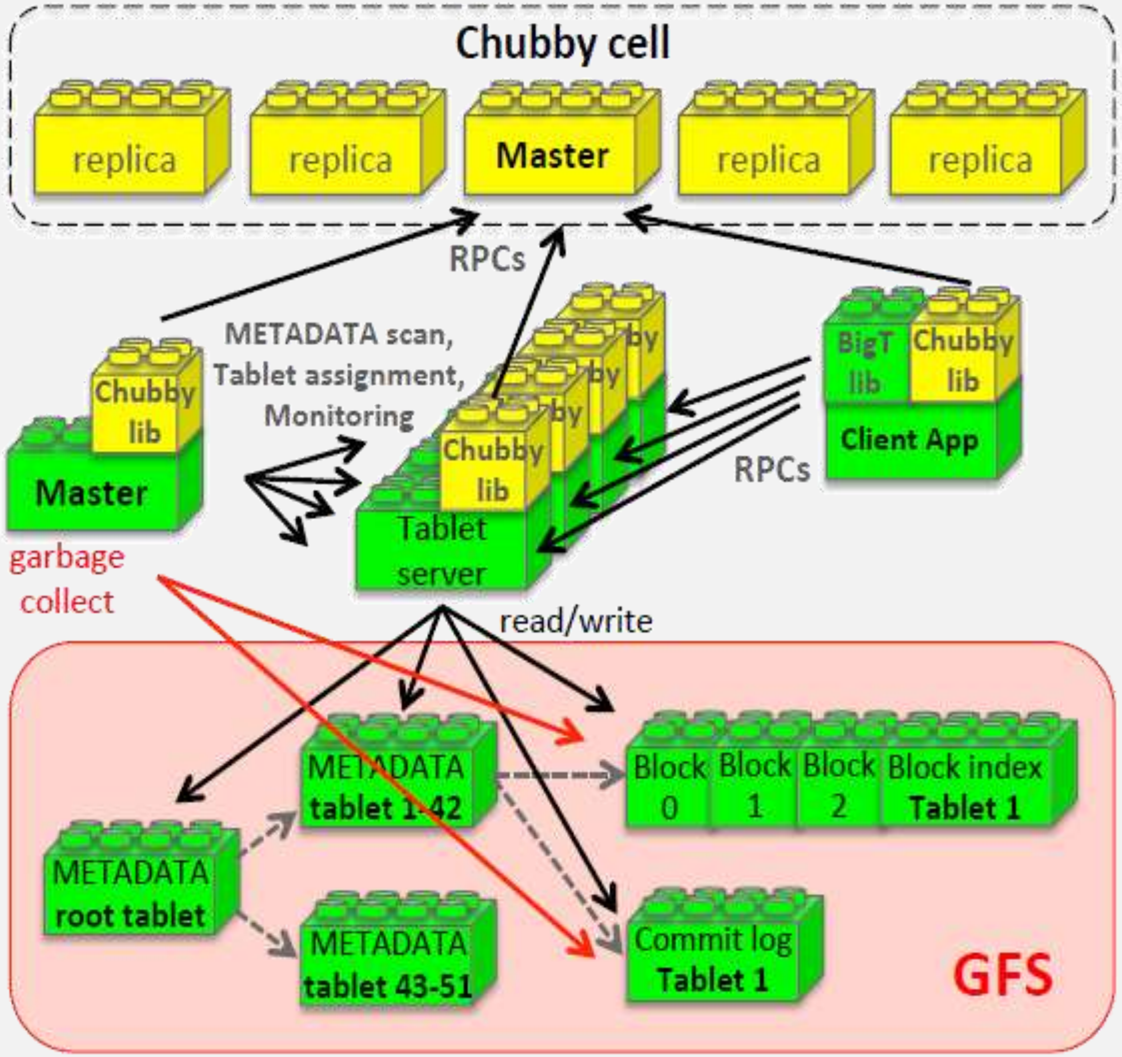
- **One Master server**

- ✓ Assigning tablets to tablet servers
- ✓ Detecting the addition and expiration of tablet servers
- ✓ Balancing tablet server load
- ✓ Garbage collecting of files in GFS
- ✓ Handling schema changes (table creation, column family creation/deletion



# Implementation

- **Many tablet servers**
- ✓ Manages a set of tablets
- ✓ Handles read and write request to the tablets
- ✓ Splits tablets that have grown too large (100--200 MB)

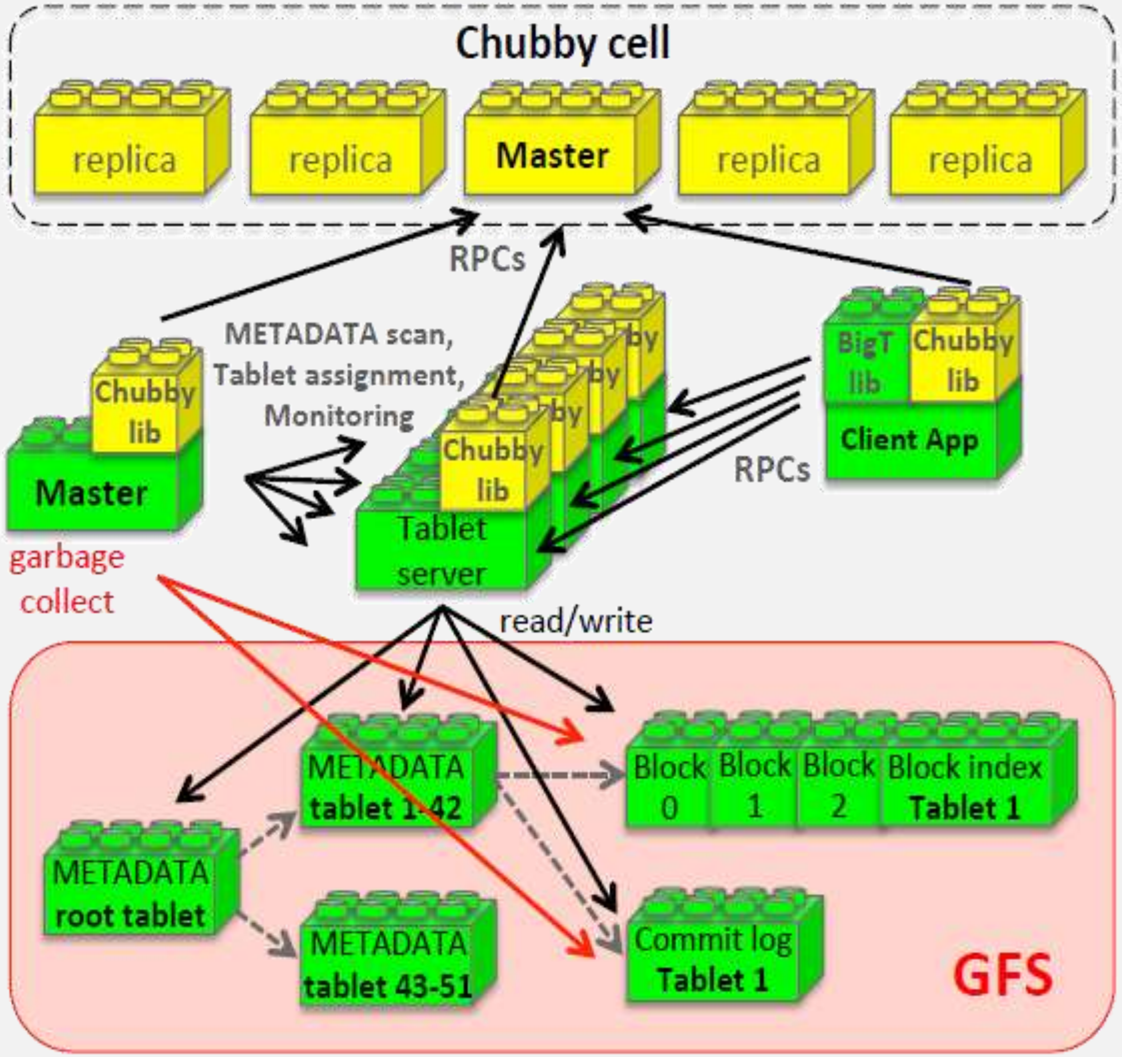


# Implementation

- A library that is linked into every client

- ✓ Do not rely on the master for tablet location information

- ✓ Communicates directly with tablet servers for reads and writes

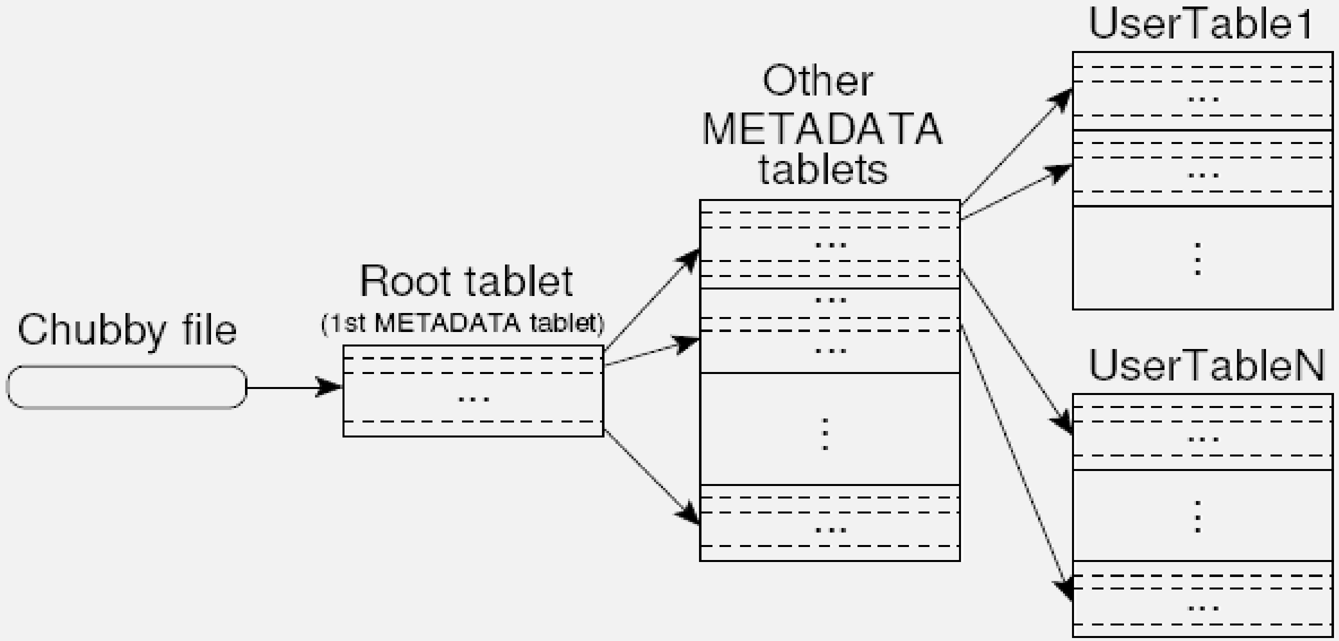


# Tablet Location

- **Chubby File**: Provides an **namespace** to **access** the **root table**. This is the first entry point to locate a user table.

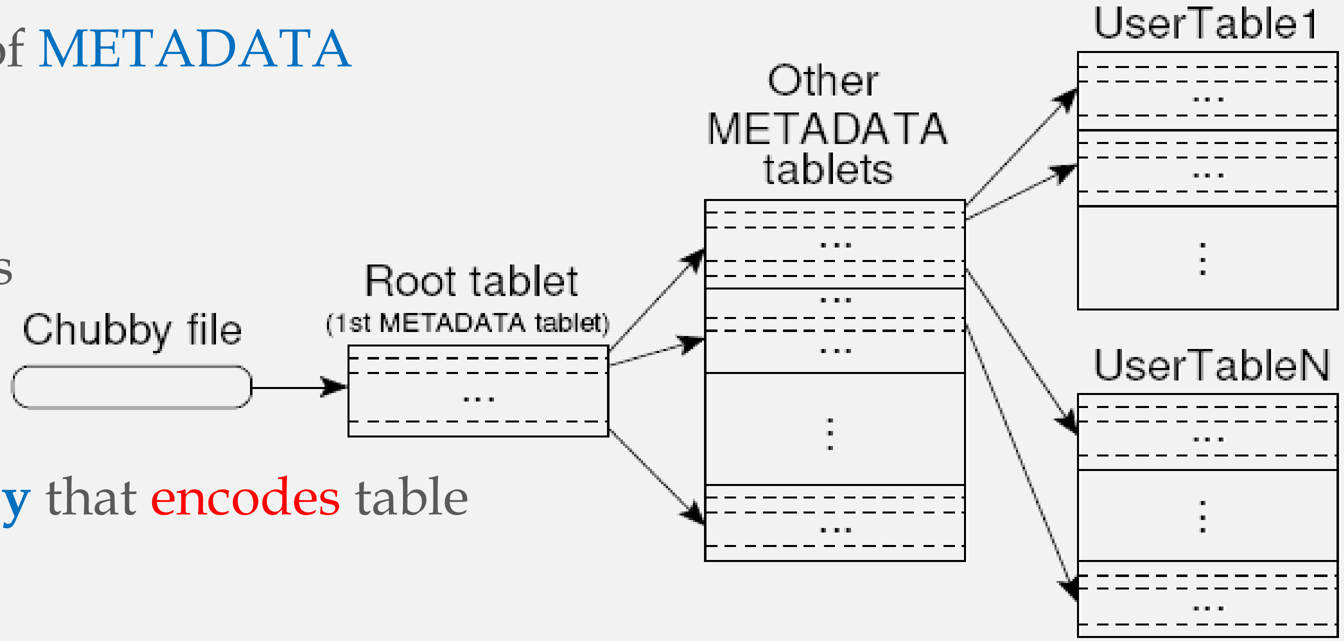
  The service is distributed. The cubby service is used for:

  - ✓ Bootstrap the location of BigTable
  - ✓ Discover server tablets
  - ✓ Finalize tablets servers deaths



# Tablet Location

- **Three level hierarchy**
- Level **1**: **Chubby file** containing location of the root tablet
- Level **2**: **Root tablet** contains the location of METADATA tablets
- Level **3**: Each **METADATA tablet** contains the location of user tablets
- Location of tablet is stored under a **row key** that encodes table identifier and its end row



# Tablet LookUp

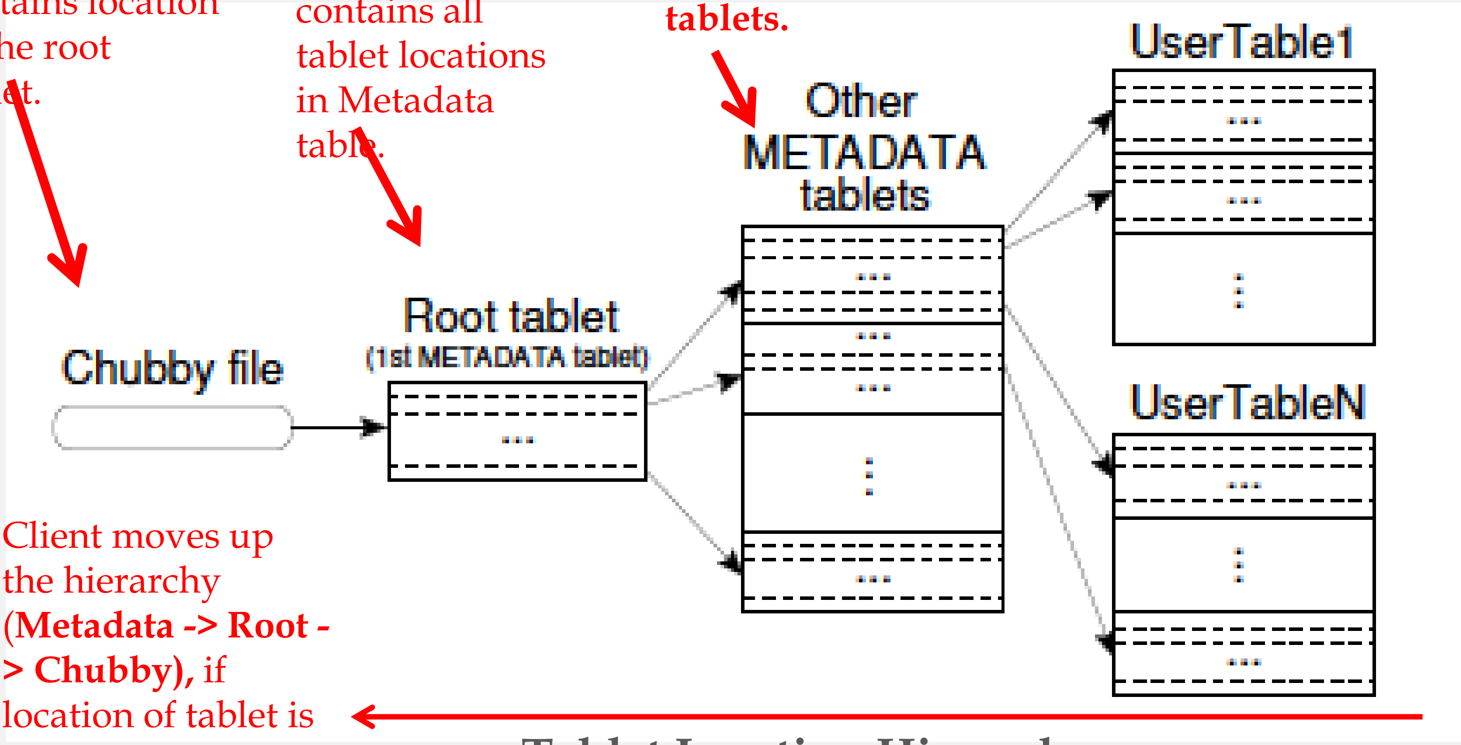
- Lookup is a **three-level system.**

- **Benefit :- NO Big Bottleneck** in the system and it also make heavy use of **Pre-Fetching** and **Caching**

Chubby file contains location of the root tablet.

Root tablet contains all tablet locations in Metadata table.

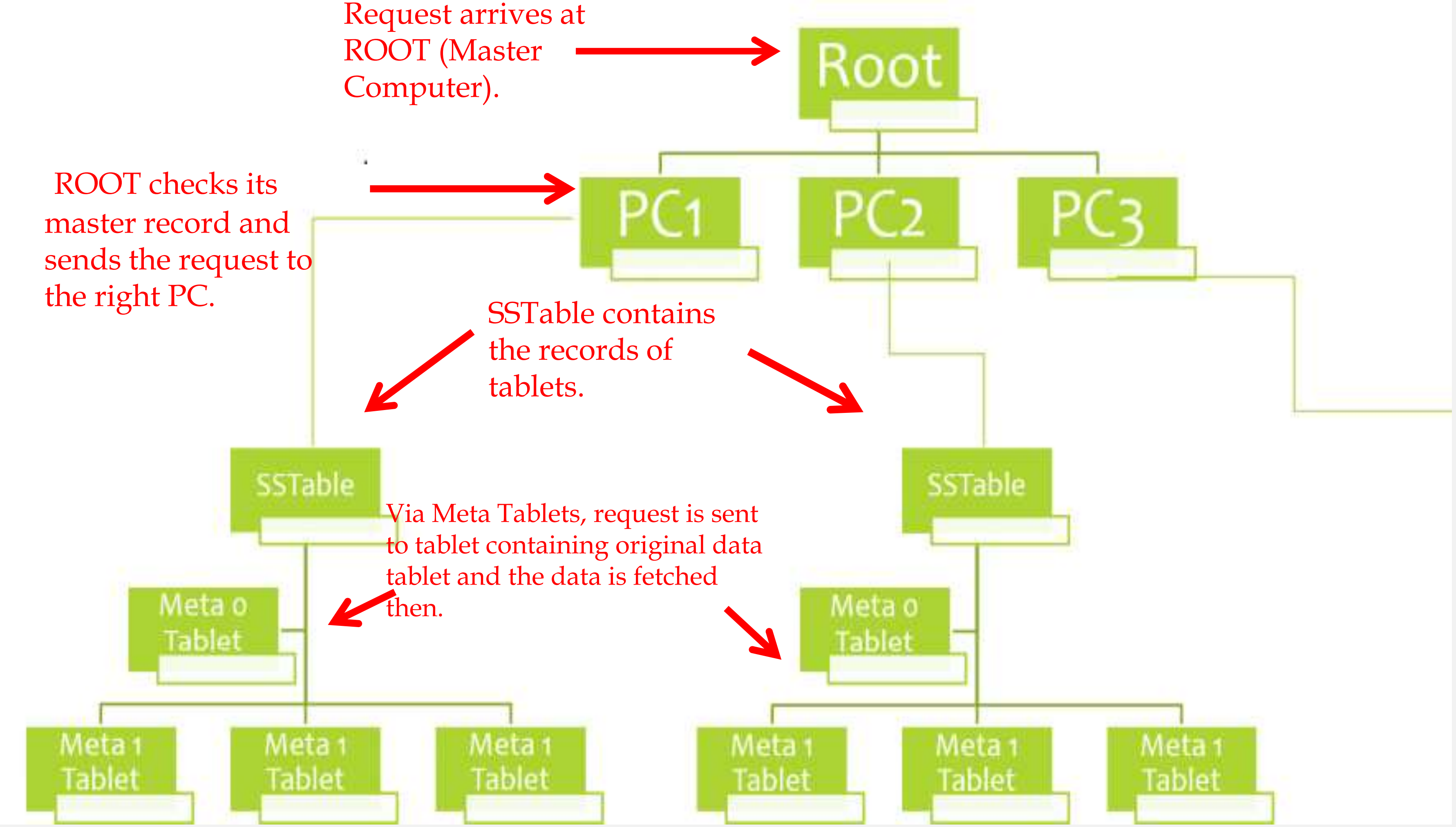
**Metadata table stores locations of actual tablets.**



Client moves up the hierarchy (**Metadata -> Root -> Chubby**), if location of tablet is **unknown or incorrect.**

Tablet Location Hierarchy

# Actual Hierarchical Load Balancing Structure



Request arrives at ROOT (Master Computer).

ROOT checks its master record and sends the request to the right PC.

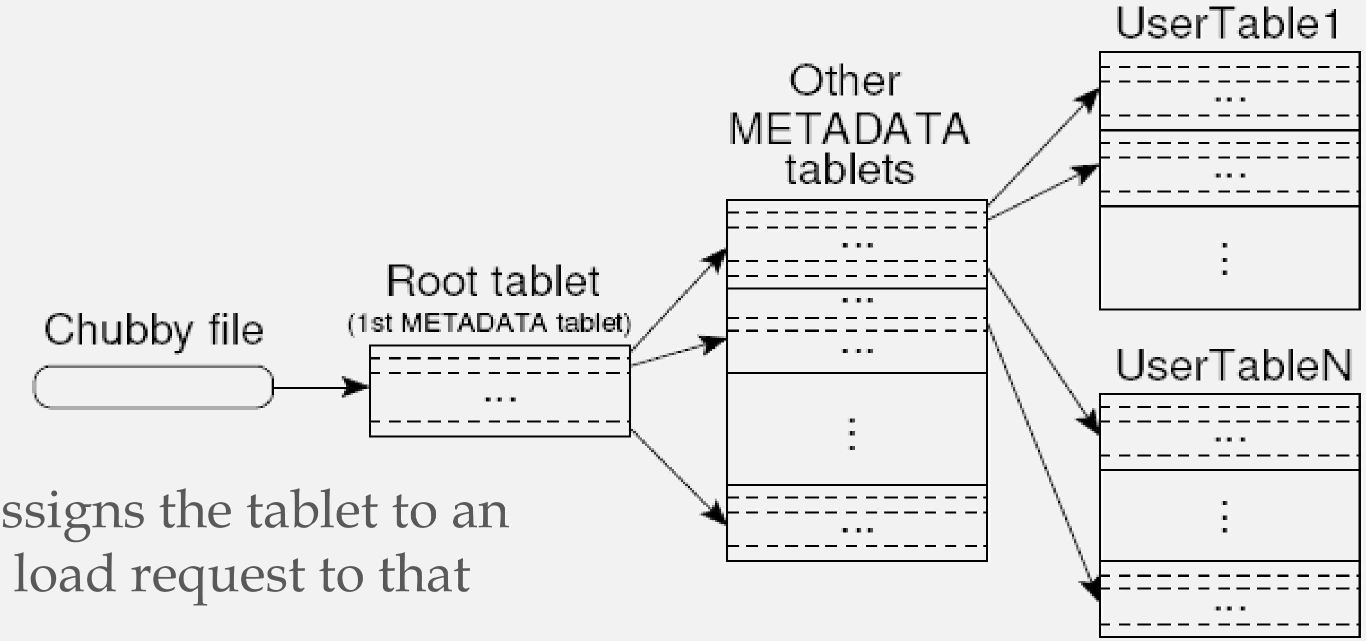
SSTable contains the records of tablets.

Via Meta Tablets, request is sent to tablet containing original data tablet and the data is fetched then.

**This is how, it works**

# Tablet Assignment

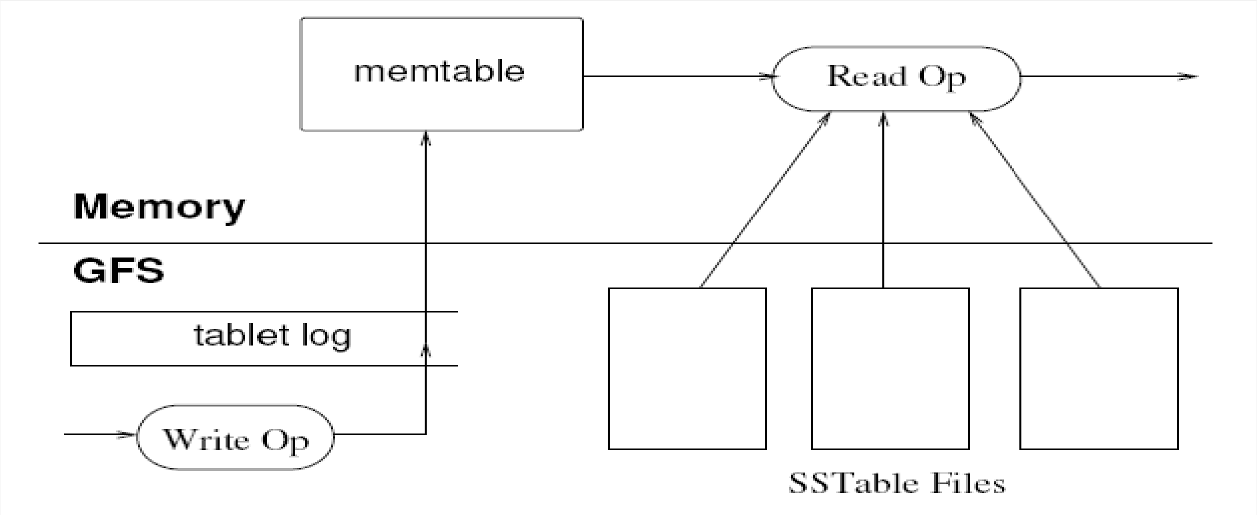
- Each  tablet is assigned to **one** tablet server at a **time**
- **Master**  keeps tracks of
  - – the set of live tablet servers (tracking via Chubby)
  - – the current assignment of tablet to tablet servers
  - – the current unassigned tablets



- When a tablet is unassigned, the **master** assigns the tablet to an available tablet server by sending a tablet load request to that tablet server

# Tablet Serving

- Updates committed to a **commit log**
- Recently committed updates are stored in memory –**MEMtable**
- Older updates are stored in a sequence of **SSTables**.



# Tablet Serving

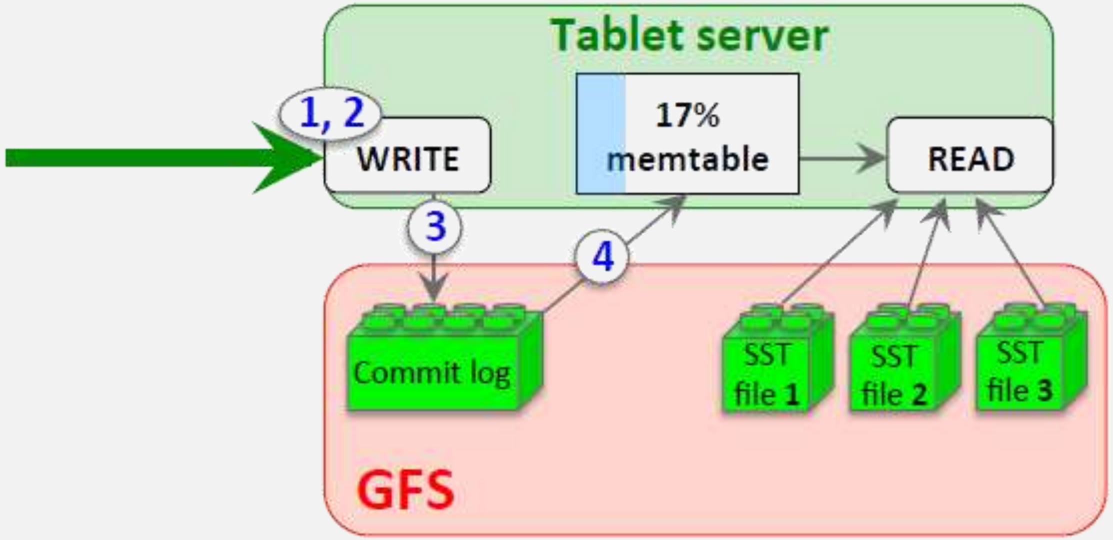
❑ **Write operation:**

1. Server checks that the request is **well-formed**

2. Server checks that the sender is **authorized** to write (list of permitted writers in a Chubby file)

3. A **valid mutation** is written to the commit log that stores redo records (group commit to improve throughput)

4. After the mutation has been committed, its **contents** are inserted into the **MEMtable** (= in memory sorted buffer)



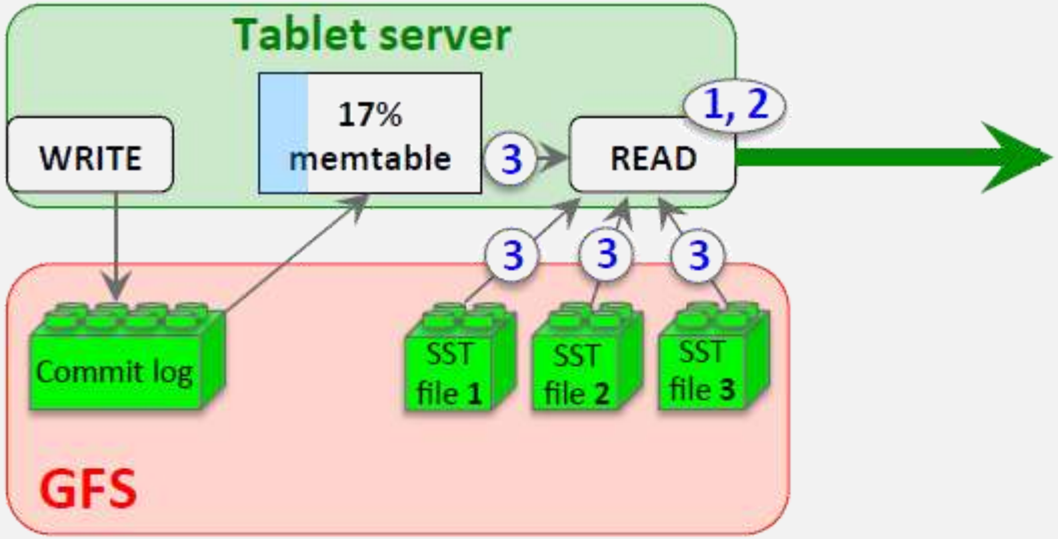
# Tablet Serving

❑ **Read operation:**

1. Server checks that the request is **well-formed**

2. Server checks that the sender is **authorized** to read (list of permitted writers in a Chubby file)

3. **Valid read operation** is executed on a merged view of the sequence of SSTables and the MEMtable

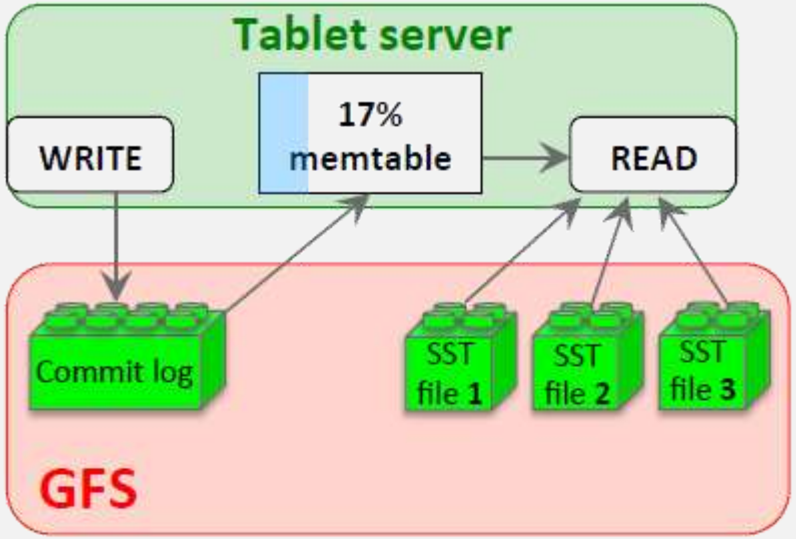


# Tablet Serving

- ❑ **Tablet Recovery**

1. Tablet server reads its metadata from the METADATA table (lists of SSTables that comprise a tablet and a set of a redo points, which are pointers into any commit logs that may contain data for the tablet)

2. The tablet server reads the indices of the SSTables into memory and reconstructs the MEMtable by applying all of the updates that have a commted since the redo points



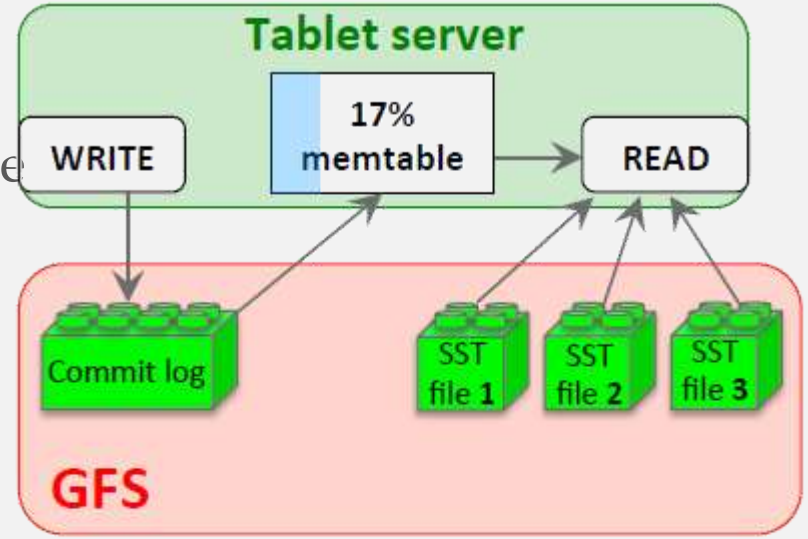
# Compaction

- ❑ In order to control size of MEMtable, tablet log, and SSTable files, "compaction" is used.

1. **Minor Compaction** - Move data from **MEMtable** to **SSTable**.

2. **Merging Compaction** - Merge multiple **SSTables** and **MEMtable** to a single **SSTable**.

3. **Major Compaction** - that re-writes all **SSTables** into exactly one **SSTable**

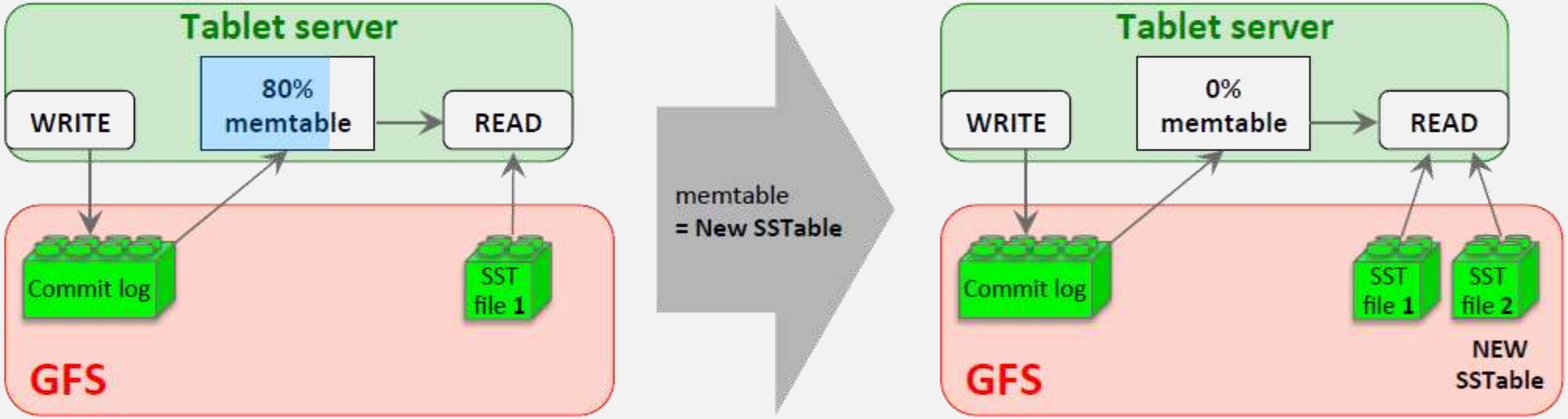


# Compaction

1. **Minor Compaction**

– When MEMtable size reaches a threshold, MEMtable is frozen, a new MEMtable is created, and the frozen MEMtable is converted to a new SSTable and written to GFS

– Two goals: shrinks the memory usage of the tablet server, reduces the amount of data that has to be read from the commit log during a recovery

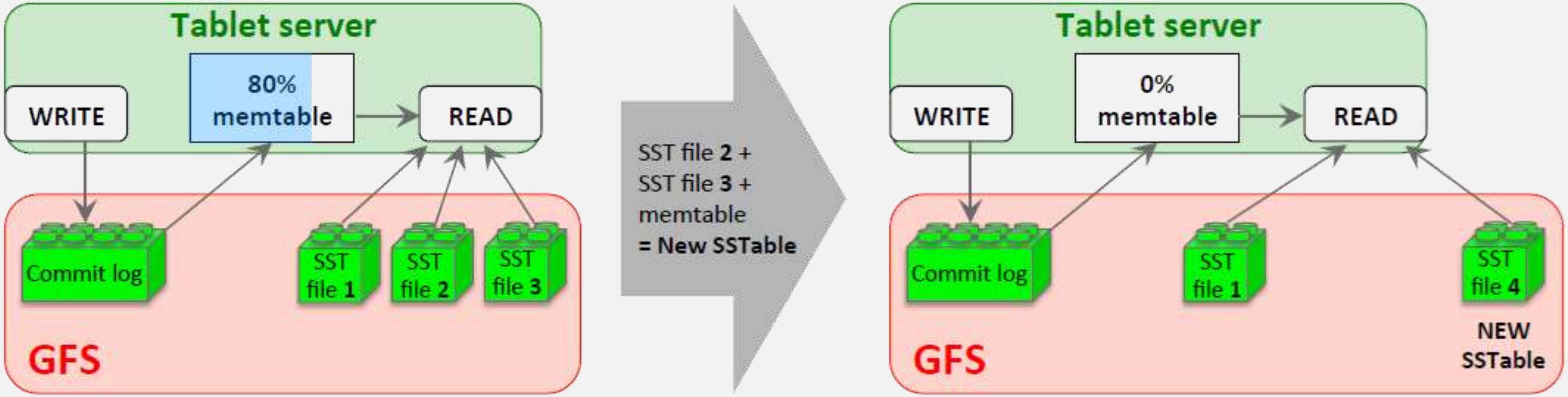


# Compaction

## 2. Merging Compaction

– **Problem**: every minor compaction creates a **new SSTable** (→arbitrary number of SSTables !)

– **Solution**: periodic merging of a few **SSTables** and the **MEMtable**

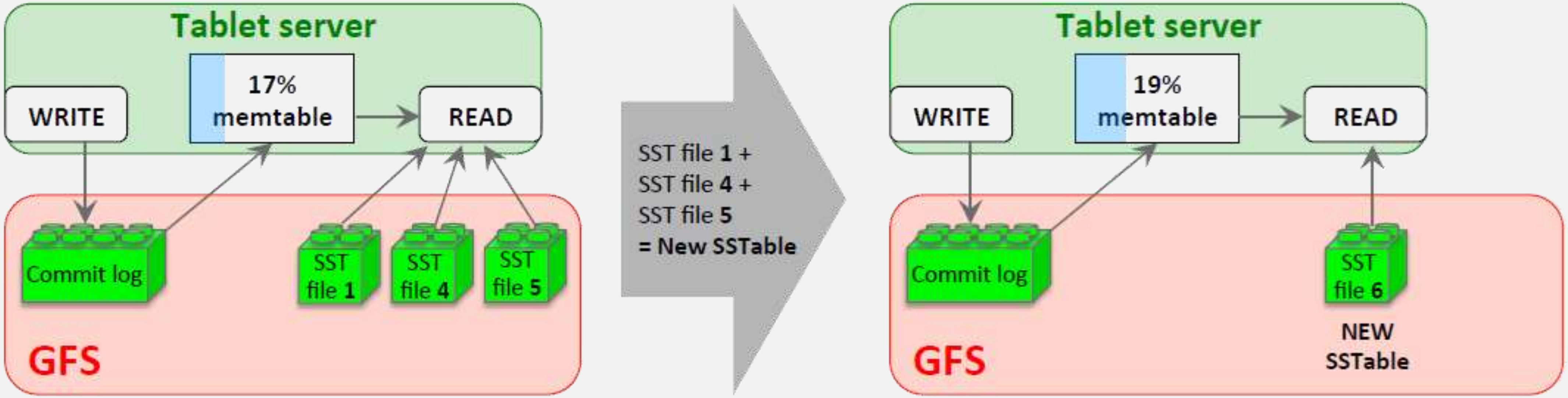


# Compaction

## 3. Major Compaction

– It is a merging compaction that rewrites all SSTables into exactly **one SSTable that contains no deletion information or deleted data**

– BigTable cycles through all of it tablets and regularly applies major compaction to them (=reclaim resources used by deleted data in a timely fashion)



# Conclusion

- BigTable has achieved its goals of high performance, data availability and scalability.
- It has been successfully deployed in real apps (Personalized Search, Orkut, Google Maps, …)

- Significant advantages of building own storage system like flexibility in designing data model, control over implementation and other infrastructure on which Bigtable relies on.

# Thanks For Listening