BIGTABLE : A DISTRIBUTED STORAGE SYSTEM FOR STRUCTURED DATA

Written by Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

Presented by Iniya Karunanithi

What is BigTable?

Bigtable is a compressed, highly ditributed, high performance data storage system which is designed to scale to a very large size (petabytes of data)

Why not DBMS?

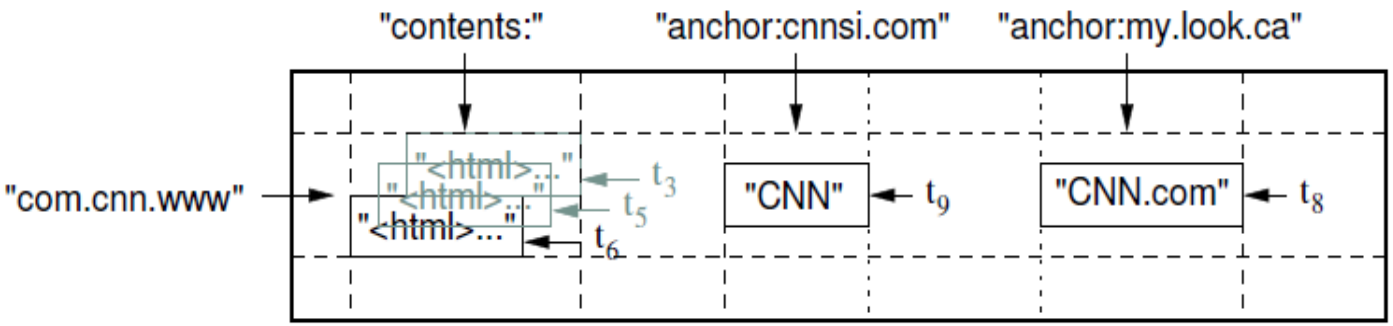
- Scale is too large
- Cost would be very high
- Low-level storage optimizations help performance significantly

Data Model

- A Bigtable is a sparse, distributed, persistent multidimensional sorted map.
- The map is indexed by a row key, column key, and a timestamp.

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

Webtable



Question 1

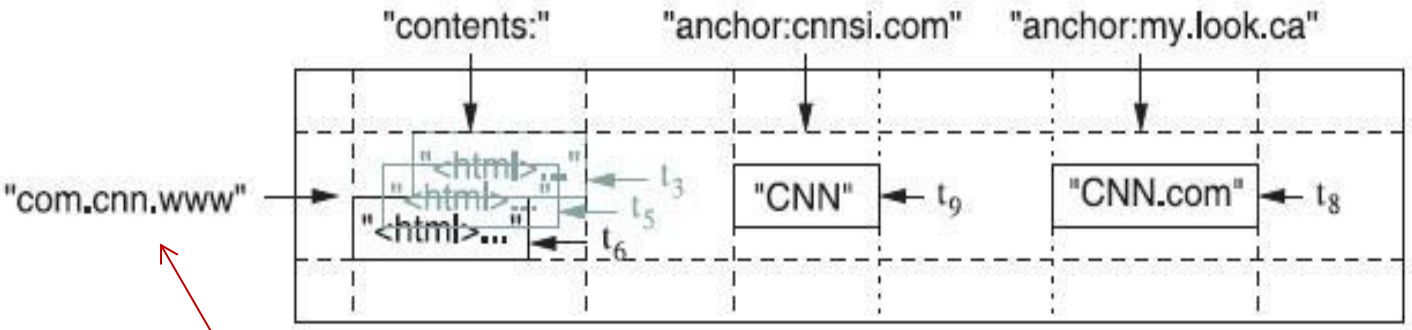
“The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.” While a table is stored in the form of KV(Key-value) items, what is the key?

The key is a combination of the row key, column key, and the timestamp.

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

Data Model - Rows

- The row keys in a table are arbitrary strings.
  - Size is 64KB
- Each read or write of data under a single row key is atomic
- 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.

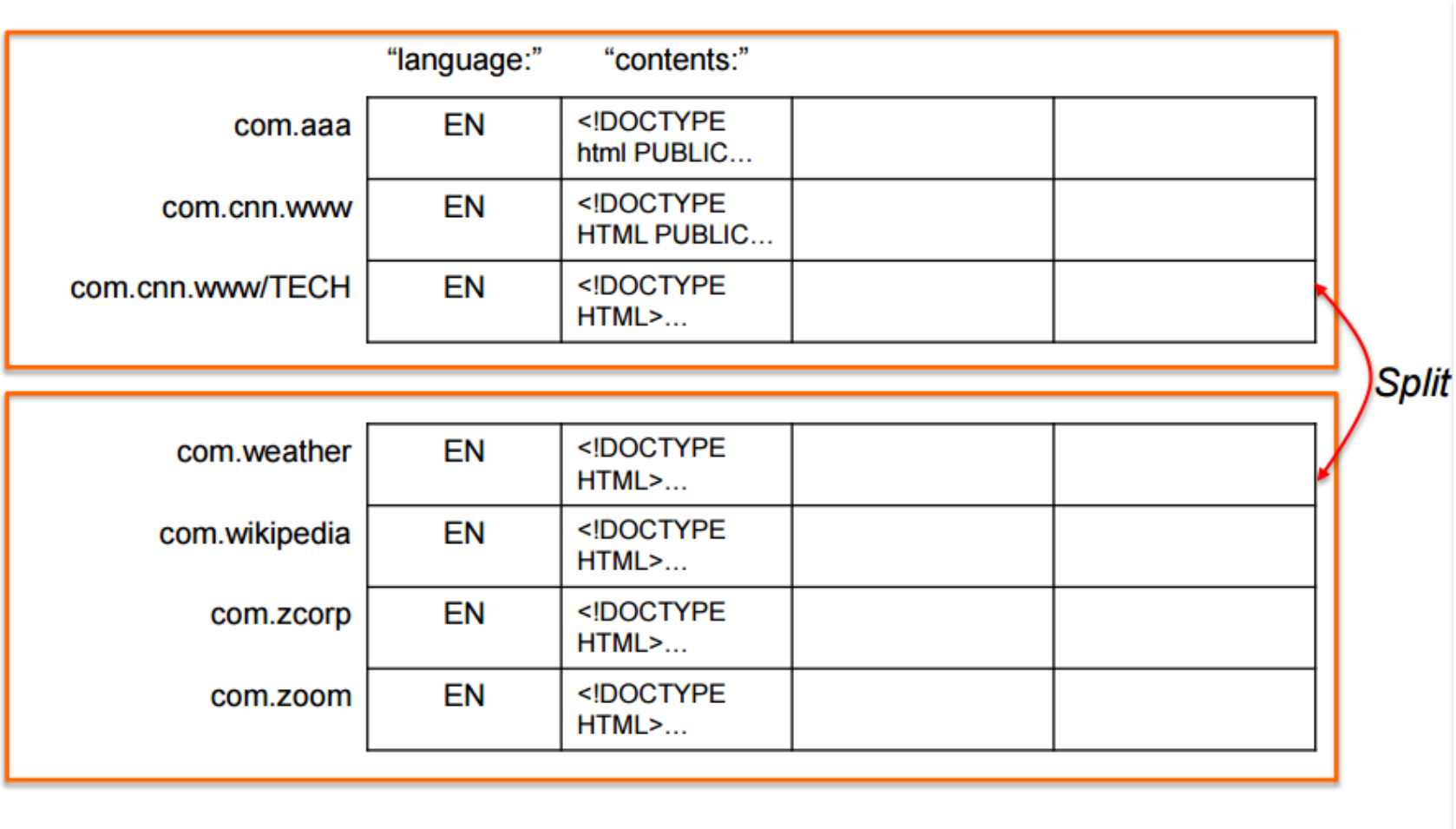


Row key

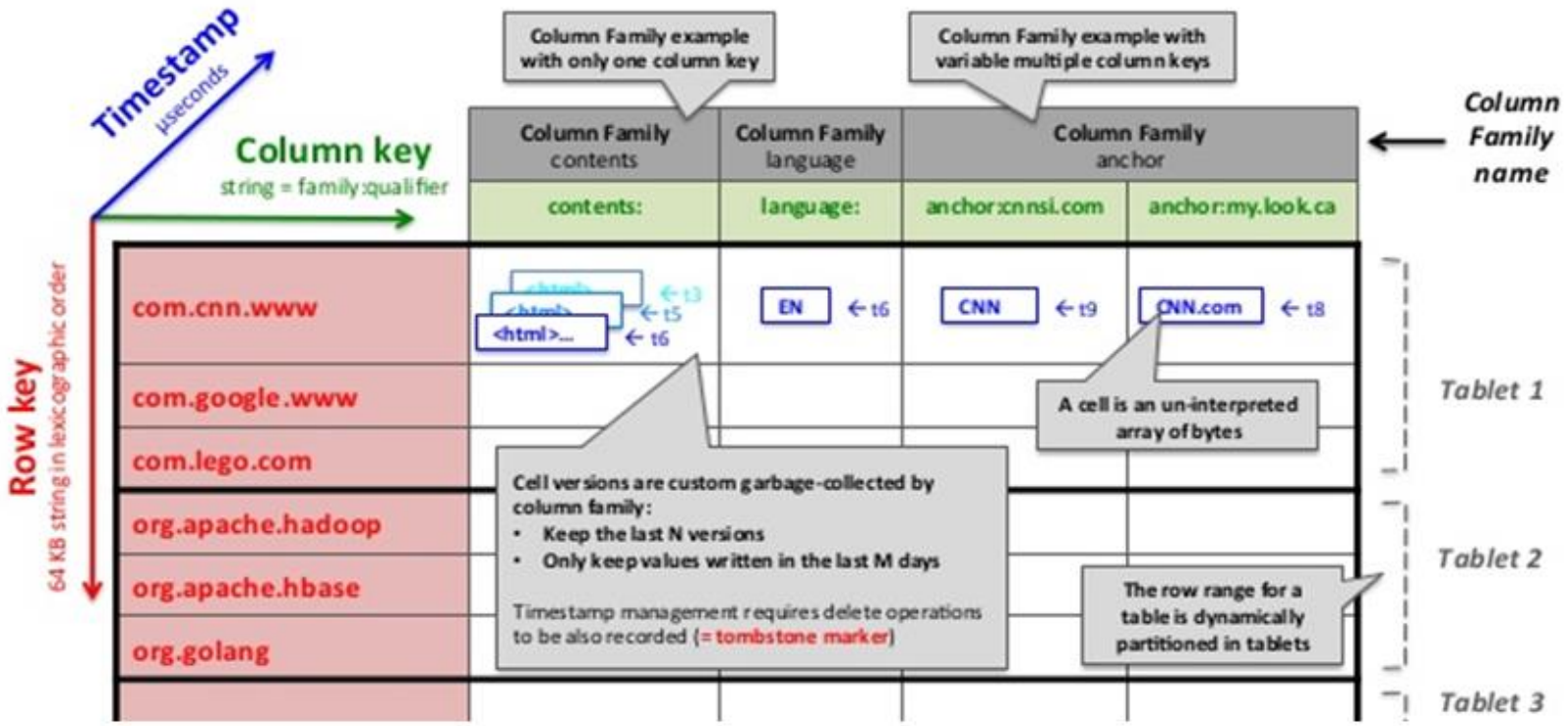
- A table starts as one tablet
- As it grows, it it split into multiple tablets
  - Approximate size: 100-200 MB per tablet by default

	"language:"	"contents:"		
com.aaa	EN	<!DOCTYPE html PUBLIC…		
com.cnn.www	EN	<!DOCTYPE HTML PUBLIC…		
com.cnn.www/TECH	EN	<!DOCTYPE HTML>…		
com.weather	EN	<!DOCTYPE HTML>…		

tablet


"language:"
"contents:"
com.aaa
EN
<!DOCTYPE html PUBLIC…
com.cnn.www
EN
<!DOCTYPE HTML PUBLIC…
com.cnn.www/TECH
EN
<!DOCTYPE HTML>…
com.weather
EN
<!DOCTYPE HTML>…
com.wikipedia
EN
<!DOCTYPE HTML>…
com.zcorp
EN
<!DOCTYPE HTML>…
com.zoom
EN
<!DOCTYPE HTML>…
Split

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
com.google.www
com.lego.com
org.apache.hadoop
org.apache.hbase
org.golang
<html>...
t3
t5
t6
EN
t6
CNN
t9
CNN.com
t8
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

Question 2

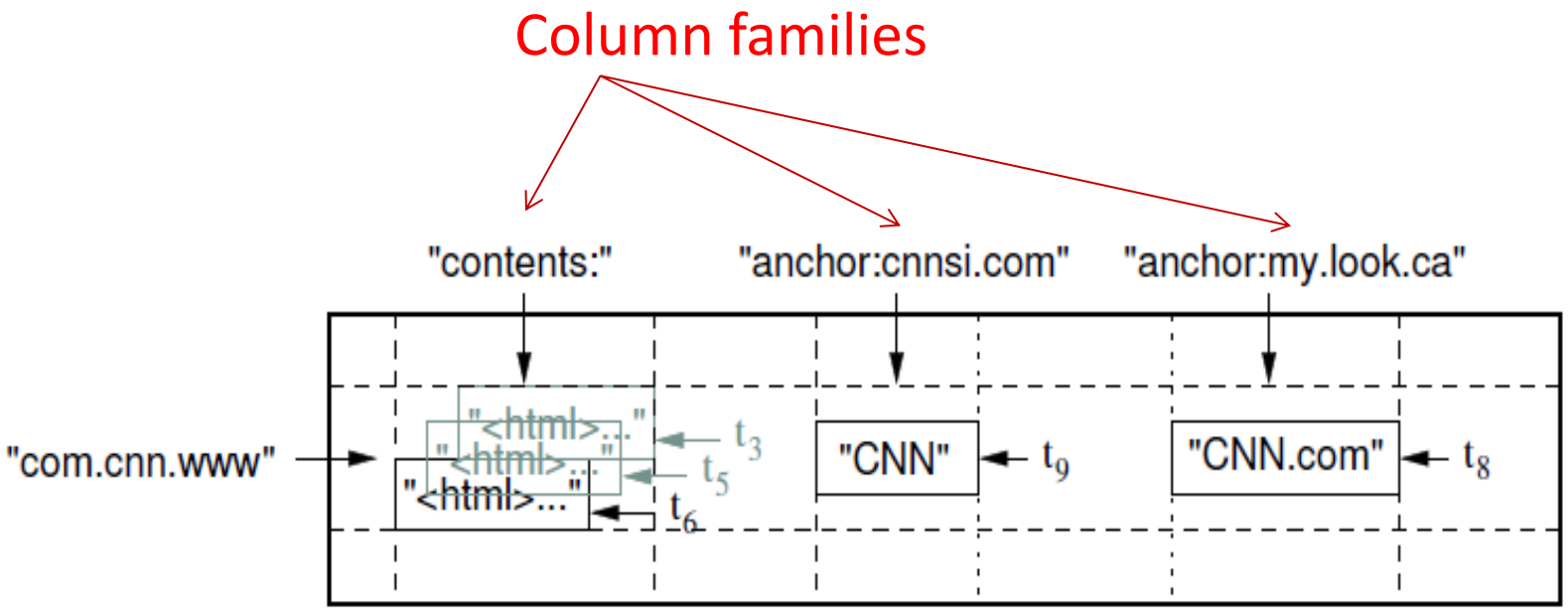
“Clients can exploit this property by selecting their row keys so that they get good locality for their data accesses.” How would clients select keys to get good locality? What possible advantages could a client obtain by having the locality?

- Since Bigtable maintains the row keys in lexicographic (alphabetic) order, clients can select row keys that are alphabetically close to each other(reversing the hostname of URL) to get good locality.
- Advantage:

  When reading data, reading a short range of rows will be more efficient and require less machines to communicate to get the values

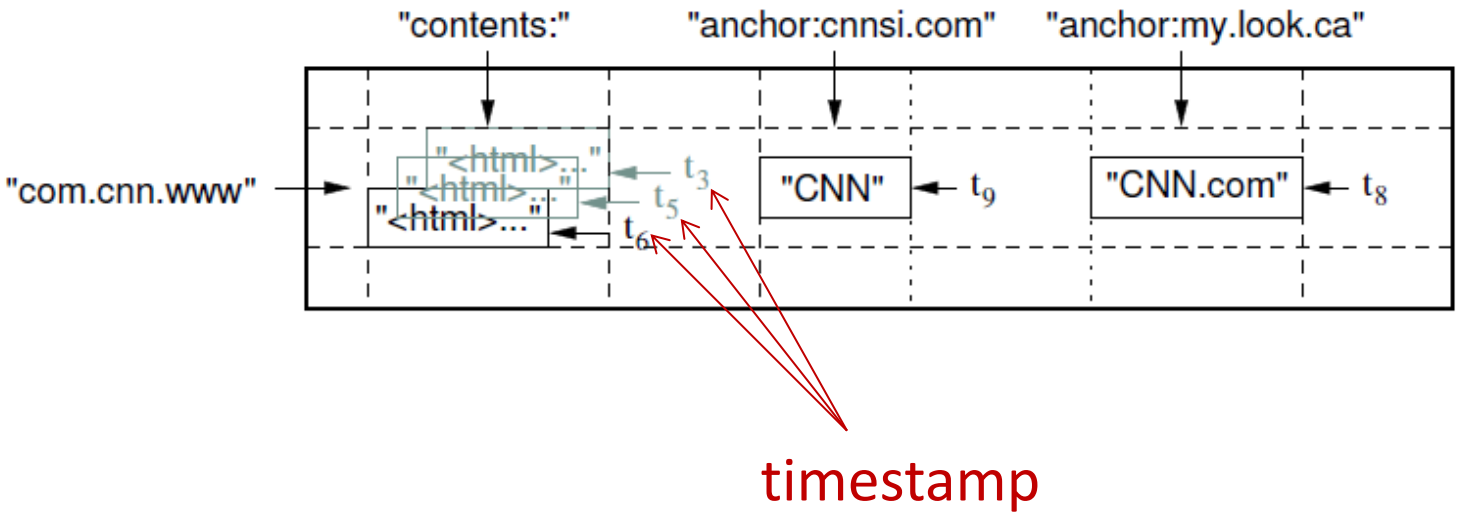
Data Model – Column Families

- Columns have two-level name structure:
  - *family:optional_qualifier*
- Column family
  - Unit of access control
  - Has associated type of information



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



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

BUILDING BLOCKS

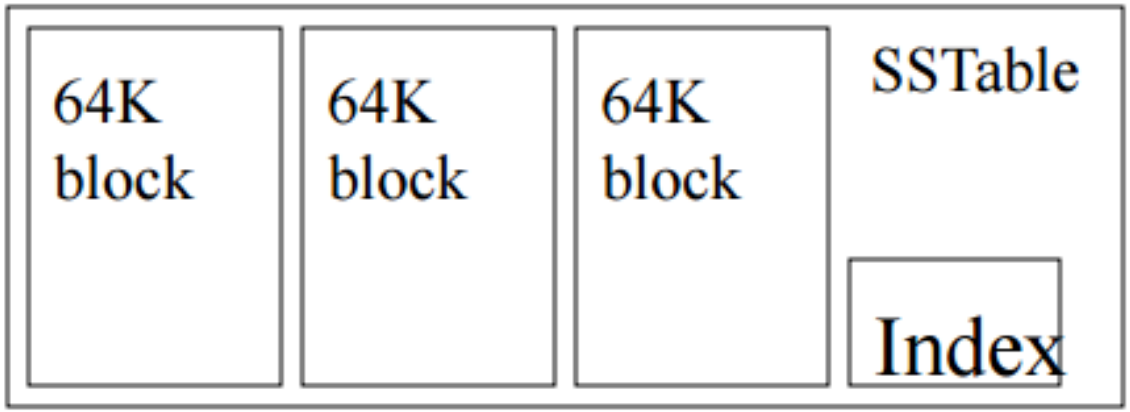
- Google File System (GFS)
- The Google SSTable (Sorted String Table) file format

Question 3

“Bigtable uses the distributed Google File System (GFS) to store log and data files.” To ensure high data reliability, does BigTable need to maintain multiple replicas for each of its data items?

Since Bigtable uses GFS, it can rely on GFS to ensure high data reliability as GFS replicates the data on to three different chunk servers for safety.

SSTable



Question 4

“The Google SSTable file format is used internally to store Bigtable data. An SSTable provides a persistent, ordered immutable map from keys to values, where both keys and values are arbitrary byte strings.” What does it mean by “immutable”? Why is this feature required?

- Immutable meaning that once SSTable is created, it cannot be modified

- Immutability is required because the cost of trying to modify SSTables as write requests come in is very high. Instead, it is faster to let the SSTables be immutable and store the changes in the memtable elsewhere.

Question 5

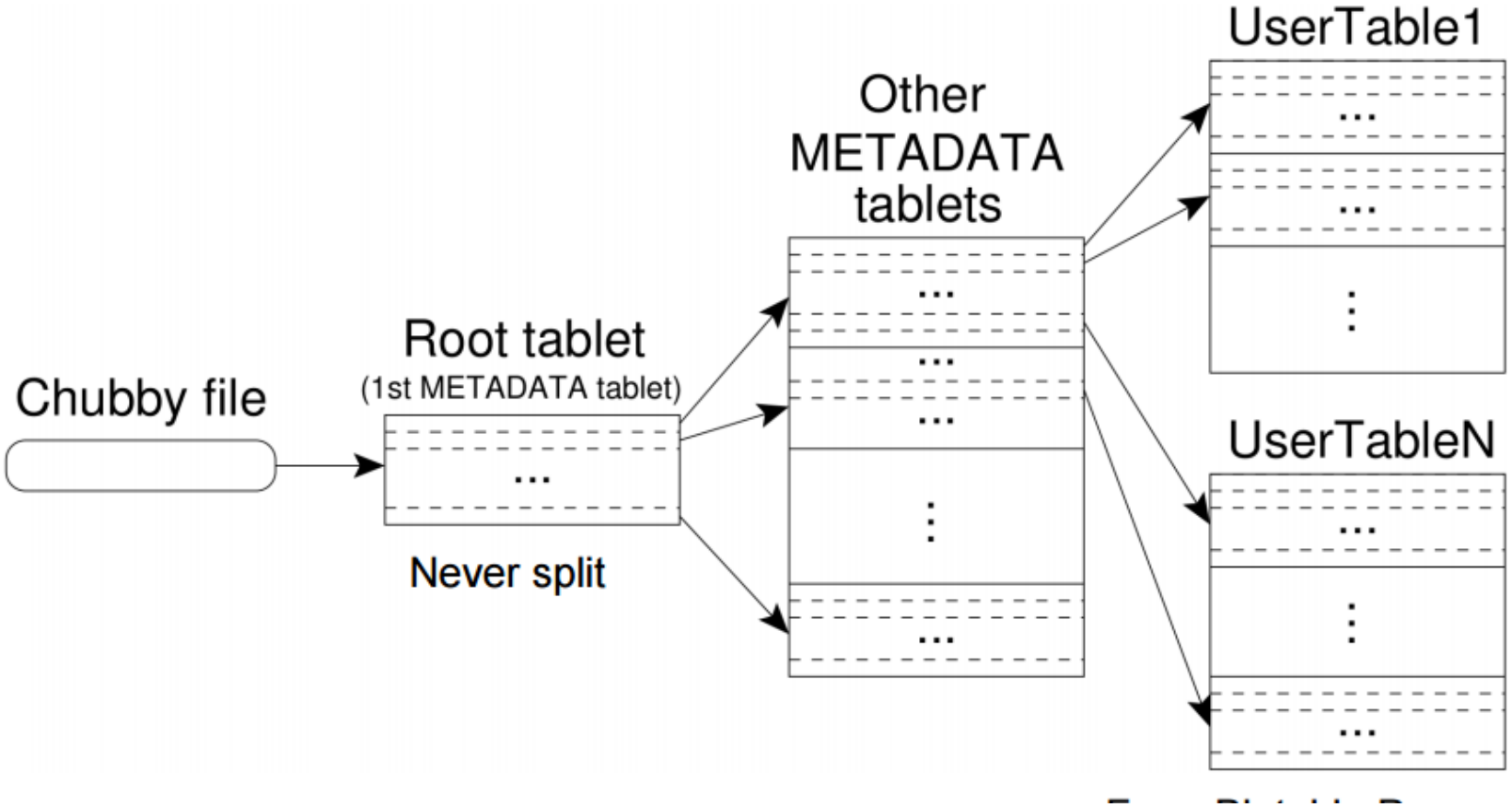
“A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: … ” Describe how a KV item is retrieved from an SSTable and why only one disk access is required for a lookup? [Hint: assume each block in an SSTable is 4KB, the disk access unit.]

Index is only loaded into the memory, not the table as a whole. When we need to lookup, binary search is performed in the in-memory index and if it is there, then appropriate block is read from the disk and this involves single disk seek.

Implementation

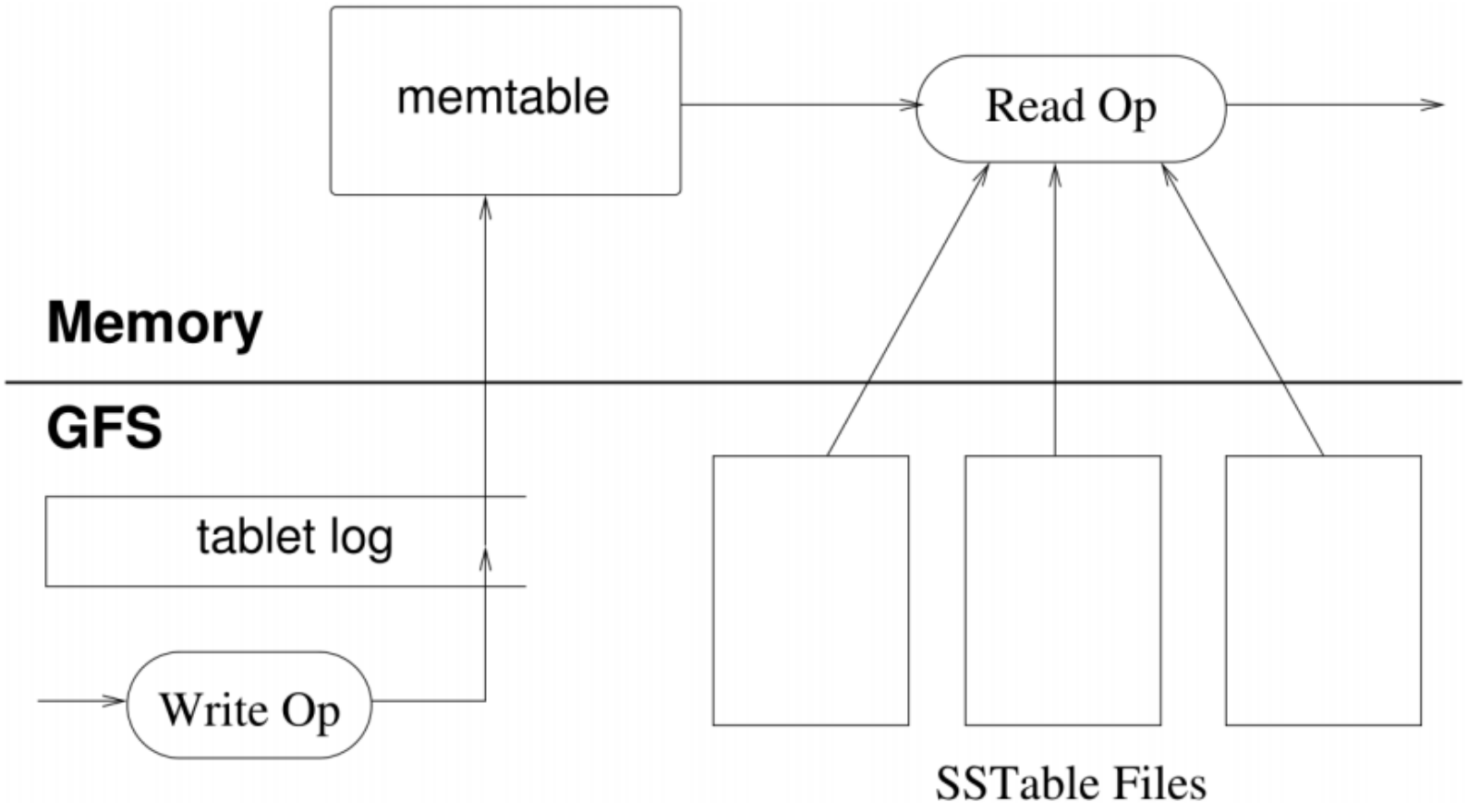
- Three major components:
  - Library linked into every client
  - Single master server
  - Many tablet servers
- Clients communicate with tablet servers directly

Tablet Location



- Each METADATA row is 1KB of memory
- The limit for METADATA table is 128MB
- Can address up to 2^{34} tablets
- Client library caches tablet location
- Clients pre-fetch the tablet location

Tablet Serving



Question 6

"Of these updates, the recently committed ones are stored in memory in a sorted buffer called a memtable; the older updates are stored in a sequence of SSTables.". Why do older updates exist and possibly exist in a sequence of SSTables?

- Since SSTables are immutable, it is not possible to add or remove immediately. Instead, older updates exist in SSTable temporarily and newer ones are in memtable. But later at some point during compaction, addition or deletion will be updated in SSTable.

Question 7

"A merging compaction that rewrites all SSTables into exactly one SSTable is called a major compaction." What is minor compaction, and what is major compaction? Why is major compaction needed? How is a KV item deleted?

- Minor compaction is converting the Memtable to SSTable
- Major compaction is combining a number of SSTables into possibly smaller number of SSTables
- Major compaction is needed:
  - so that the level of SSTables can be reduced to a smaller amount which enables faster read process
  - No deletion records, only live data (ensures deleted data disappears from the system in a timely fashion).

- To delete a KV item
  - Delete operation sent to Bigtable
  - Using the key, KV item will be marked as deleted in the in-memory. During next read, although it is still in the in-memory, it won’t be returned.
  - SSTable produced by minor compaction will contain special deletion entry that suppresses the deleted data in older SSTables that are still live.
  - During major compaction to combine SSTables, data to be deleted will be excluded.

References

- Bigtable: A Distributed Storage System for Structured Data http://research.google.com/archive/bigtable-osdi06.pdf

- ECE 7650 Lecture 8 http://www.ece.eng.wayne.edu/~sjiang/ECE7650-winter-16/lecture-8.pdf