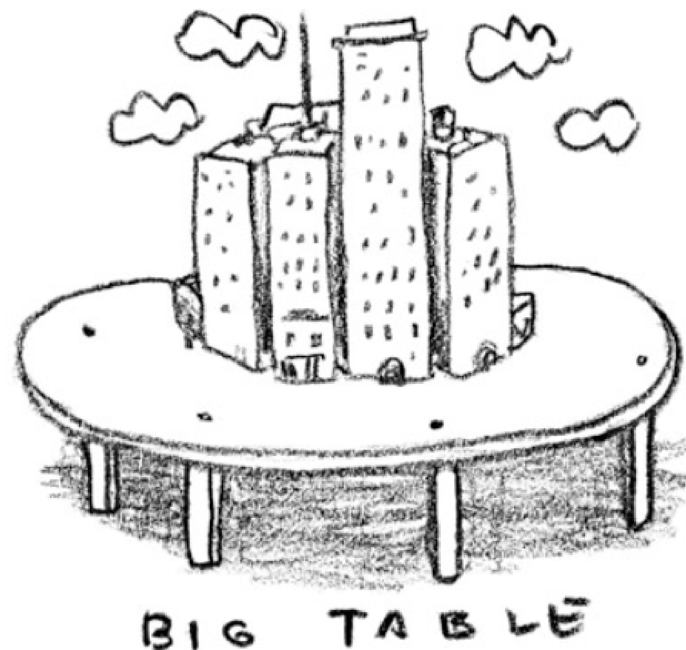
BigTable

Distributed storage for structured data

Overview

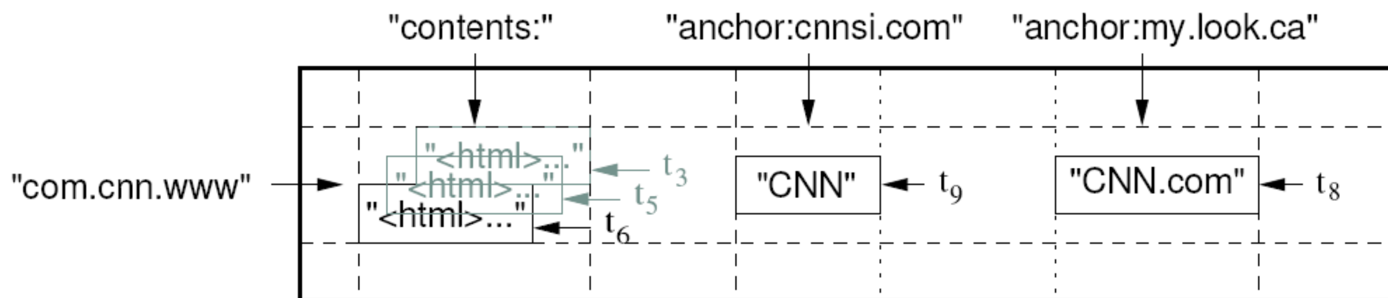
- Goals
  - **scalability**
    - petabytes of data
    - thousands of machines
  - **applicability**
    - to Google applications
      - **Google Analytics**
      - **Google Earth**
      - …
    - not a general storage model
  - **high performance**
  - **high availability**
- Structure
  - **uses GFS for storage**
  - **uses Chubby for coordination**



Note: figure from presentation by Jeff Dean (Google)

Data Model

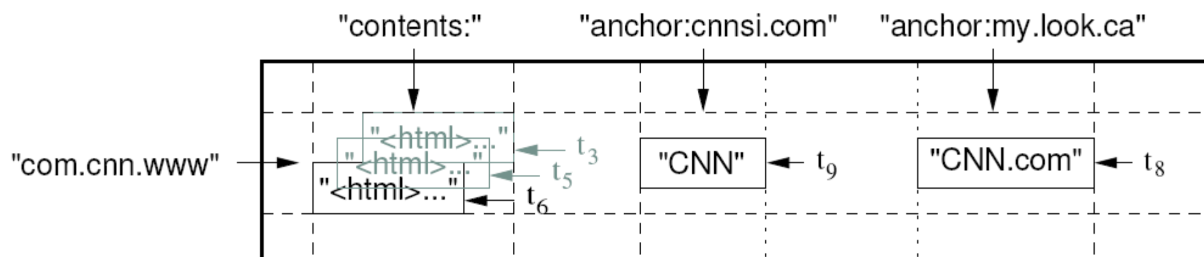
(row: string, column: string, timestamp: int64) → string



- 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**

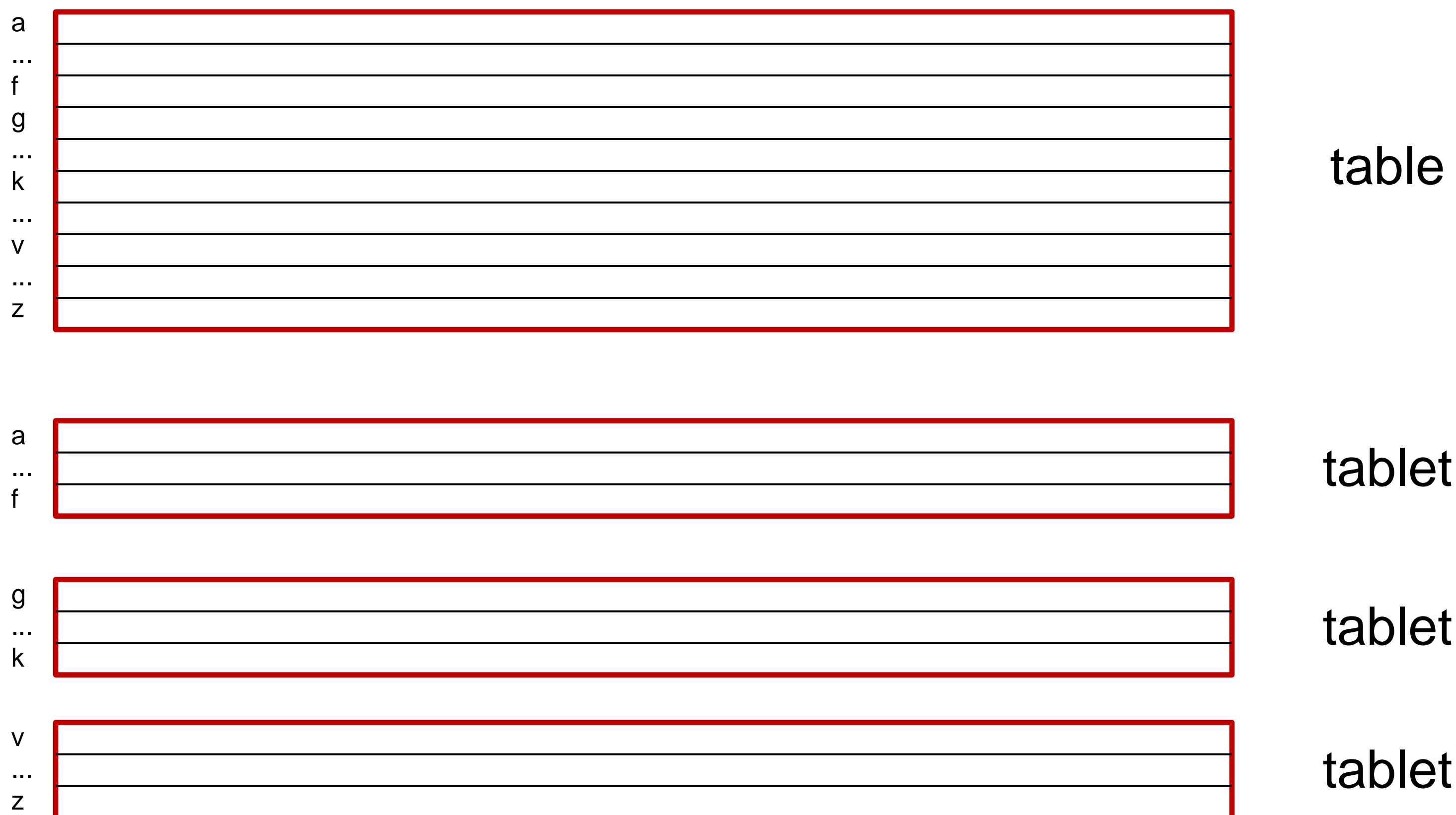
Data Model

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



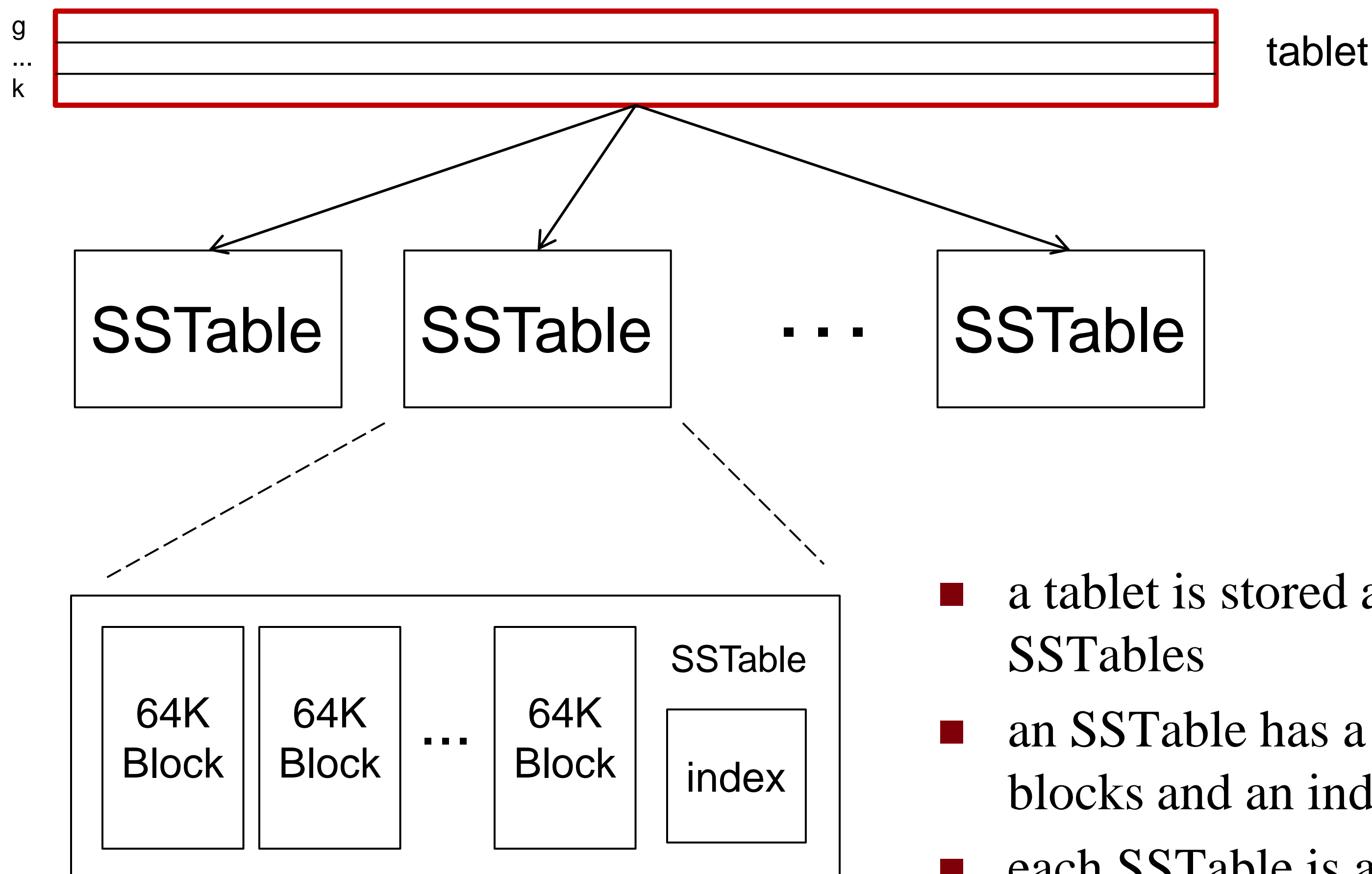
- Timestamps
  - **automatically assigned (real-time) or application defined**
  - **used in garbage collection (last n, n most recent, since time)**
- Transactions
  - **iterator-style interface for read operation**
  - **atomic single-row updates**
  - **no support for multi-row updates**
  - **no general relational model**

Table implementation



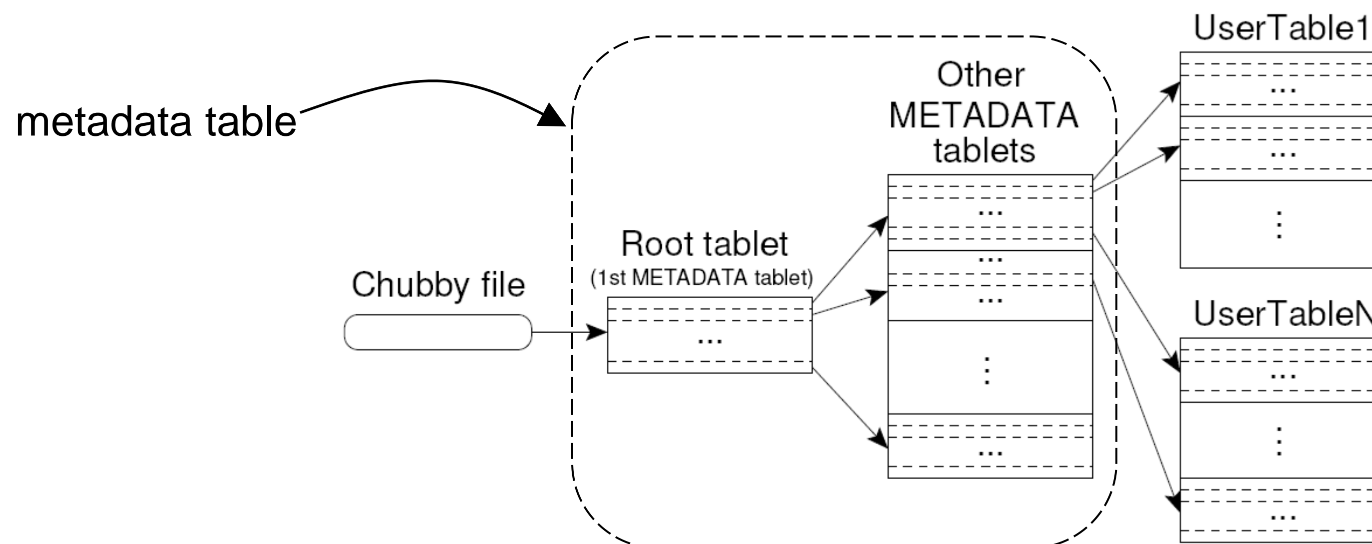
- a table is divided into a set of tablets, each storing a set of consecutive rows
- tablets typically 100-200MB

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

Locating a tablet

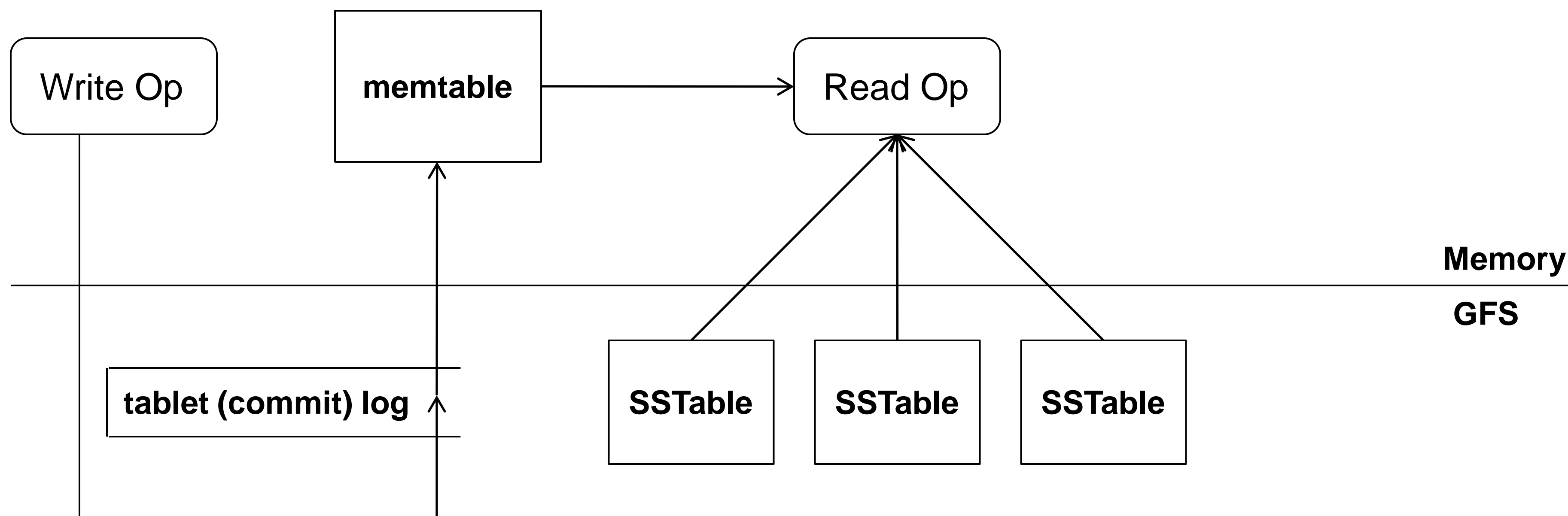


- metadata table stores location information for user table
- metadata table index by row key: (table id, end row)
- root tablet of metadata table stores location of other metadata tablets
- location of root tablet stored as a Chubby file
- metadata consists of
  - **list of SSTables**
  - **redo points in commit logs**

Master/Servers

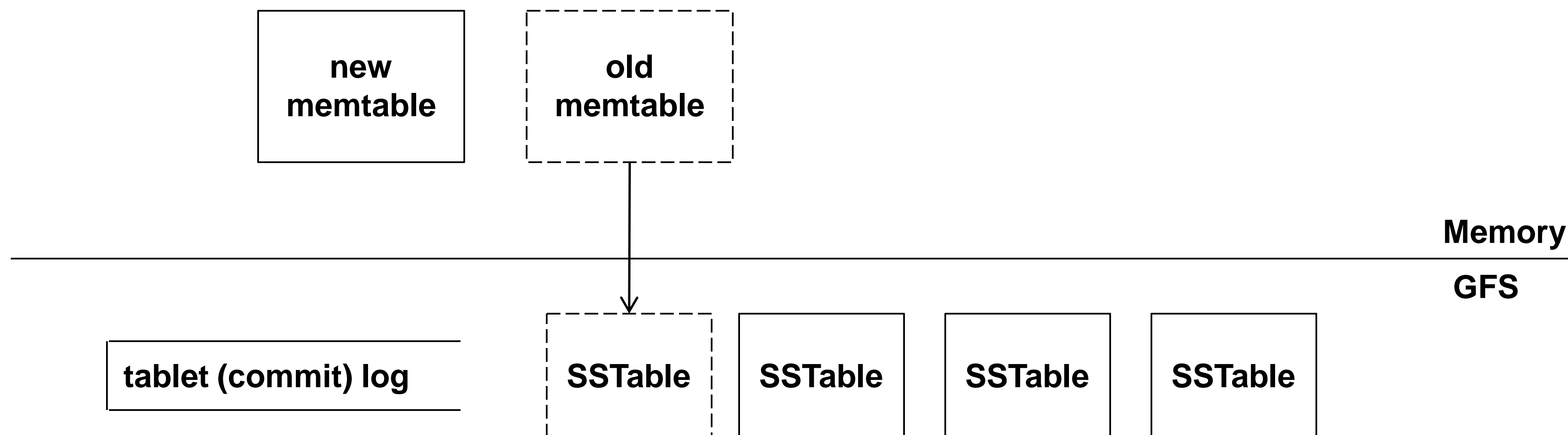
- Multiple tablet servers
  - Performs read/write operations on set of tables assigned by the master
  - Each creates, acquires lock on uniquely named file in a specific (Chubby) directory
  - Server is alive as long as it holds lock
  - Server aborts if file ceases to exist
- Single master
  - Assigns tablets to servers
  - Maintains awareness (liveness) of servers
    - List of servers in specific (servers) directory
    - Periodically queries liveness of table server
      - If unable to verify liveness of server, master attempts to acquire lock on server's file
      - If successful, delete server's file

Tablet operations



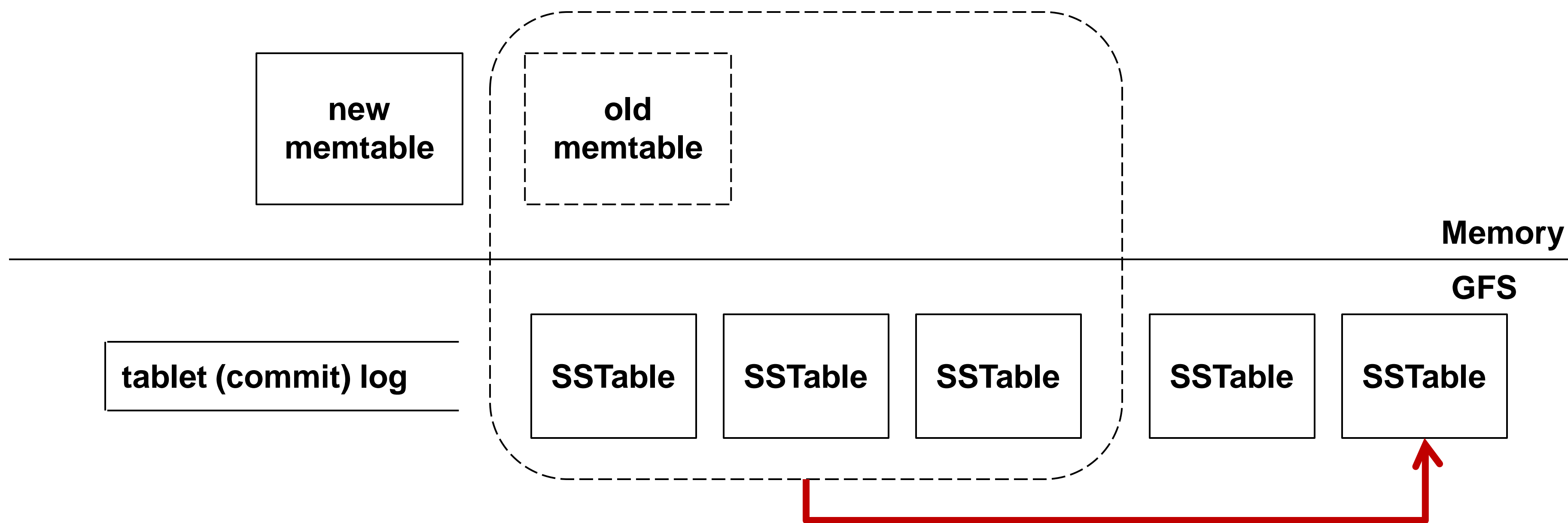
- Updates are written in a memory table after being recorded in a log
- Reads combine information in the memtable with that in the SSTables

Minor compaction



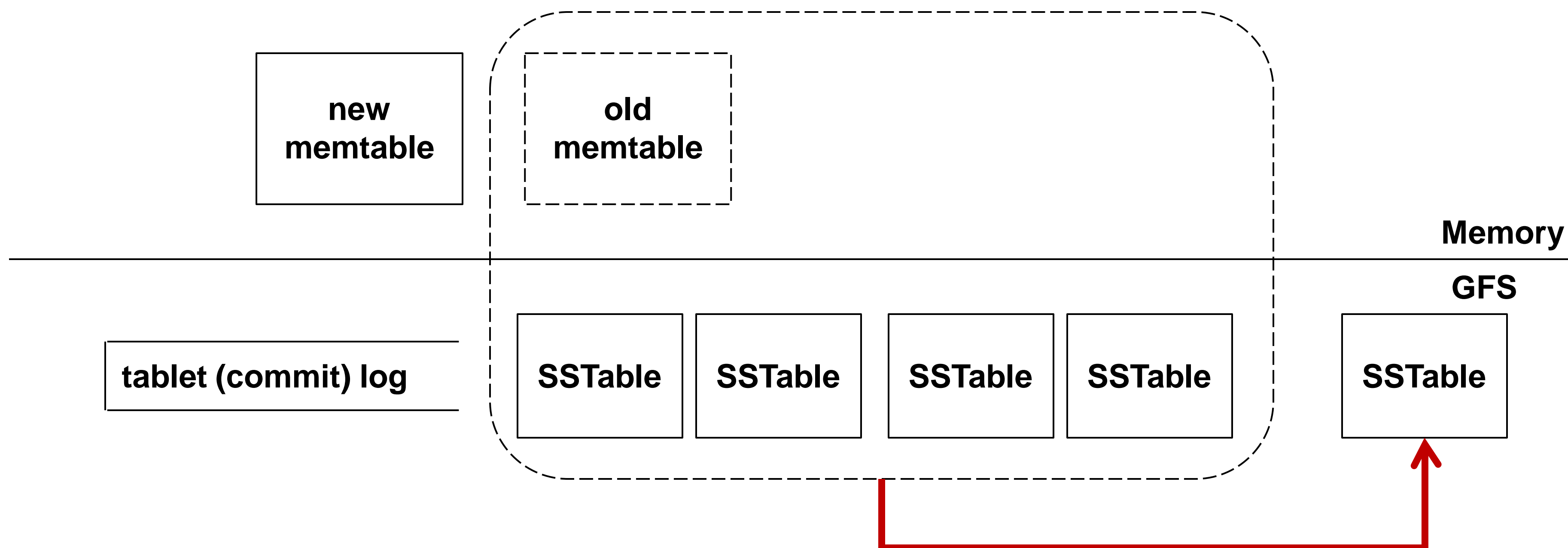
- Triggered when memtable reaches a threshold
- Reduces memory footprint
- Reduces data read from commit log on recovery from failure
- Read/write operations continue during compaction

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

Refinements

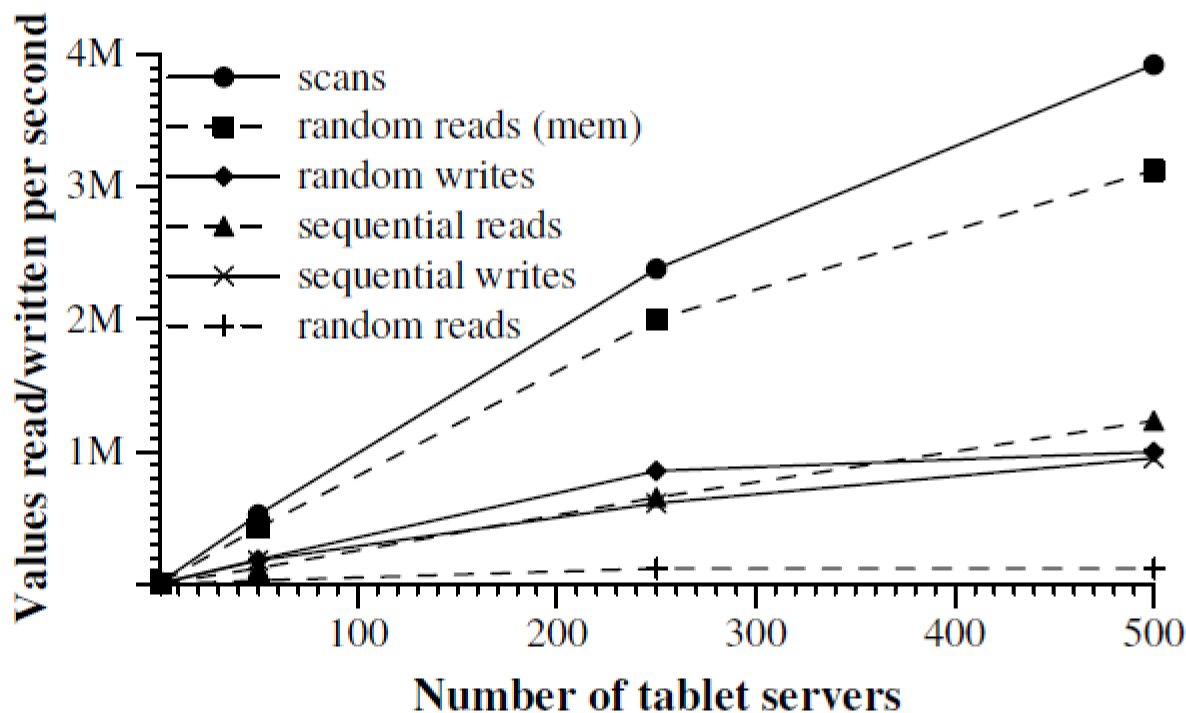
- Locality groups
  - **Client defines group as one or more column families**
  - **Separate SSTable created for group**
  - **Anticipates locality of reading with a group and less across groups**
- Compression
  - **Optionally applied to locality group**
  - **Fast: 100-200MB/s (encode), 400-1000MB/s (decode)**
  - **Effective: 10-1 reduction in space**
- Caching
  - **Scan Cache:**
    - key-value pairs held by tablet server
    - Improves re-reading of data
  - **Block Cache:**
    - SSTable blocks read from GFS
    - Improves reading of "nearby" data
- Bloom filters
  - **Determines if an SSTable might contain relevant data**

Performance

Experiment	# of Tablet Servers			
	1	50	250	500
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

- Random reads slow because tablet server channel to GFS saturated
- Random reads (mem) is fast because only memtable involved
- Random & sequential writes > sequential reads because only log and memtable involved
- Sequential read > random read because of block caching
- Scans even faster because tablet server can return more data per RPC

Performance



- Scalability of operations markedly different
- Random reads (mem) had increase of ~300x for an increase of 500x in tablet servers
- Random reads has poor scalability

Lessons Learned

- Large, distributed systems are subject to many types of failures
  - Expected: network partition, fail-stop
  - Also: memory/network corruption, large clock skew, hung machines, extended and asymmetric network partitions, bugs in other systems (e.g., Chubby), overflow of GFS quotas, planned/unplanned hardware maintenance
- System monitoring important
  - Allowed a number of problems to be detected and fixed

Lessons Learned

- Delay adding features unless there is a good sense of their being needed
  - **No general transaction support, not needed**
  - **Additional capability provided by specialized rather than general purpose mechanisms**
- Simple designs valuable
  - **Abandoned complex protocol in favor of simpler protocol depending on widely-used features**