Bigtable: A distributed Storage System for Structured Data

Fay Chang **Jeffrey Dean** **Sanjay Ghemawat**

Wilson C. Hsieh **Deborah A. Wallach**

Mike Burrows **Tushar Chandra**

Andrew Fikes **Robert E. Gruber**

Google, Inc.

Presented by: Cătălin-Alexandru Avram

February 1st 2010

Overview

- Introduction
- The Data Model
- Building Blocks
- Implementation
- Performance Evaluation
- Conclusion
- Questions & Discussions

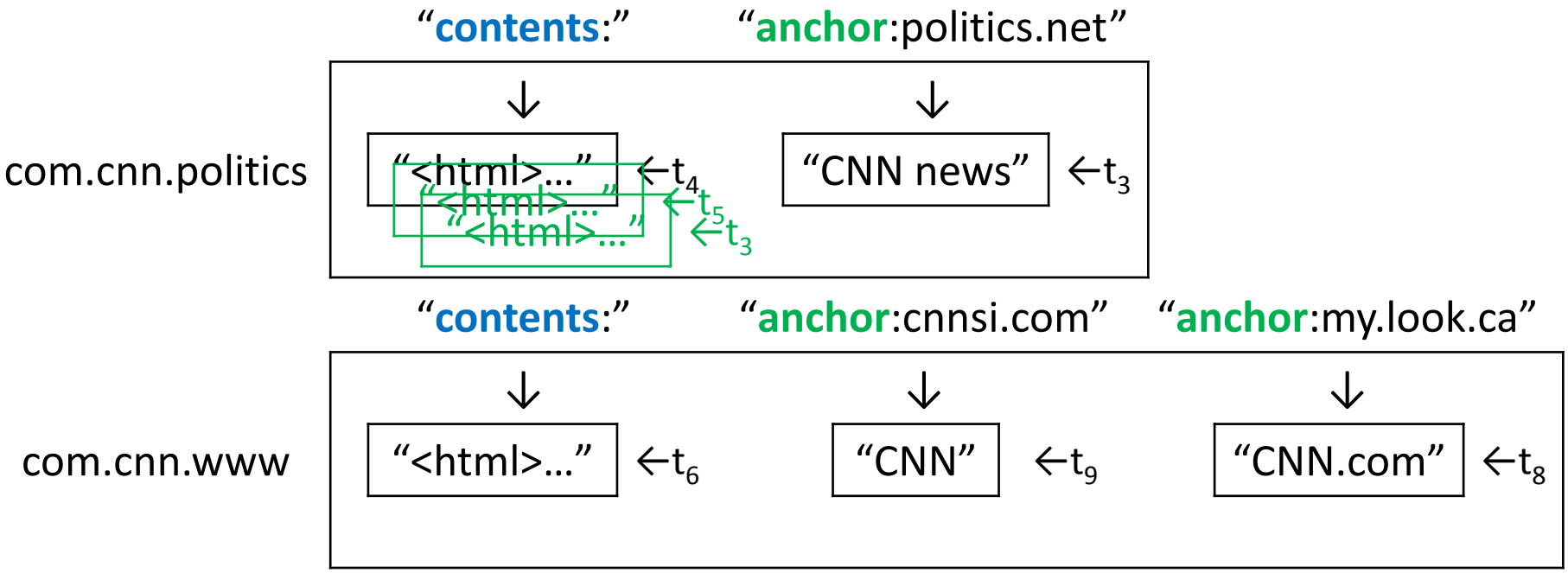
Introduction

- Bigtable is a distributed storage system for managing structured data
- It is extremely scalable (it can work with petabytes worth of data on thousands of machines)
- It is actively used by over 60 Google products with workloads ranging from batch processing to live data serving

Data Model

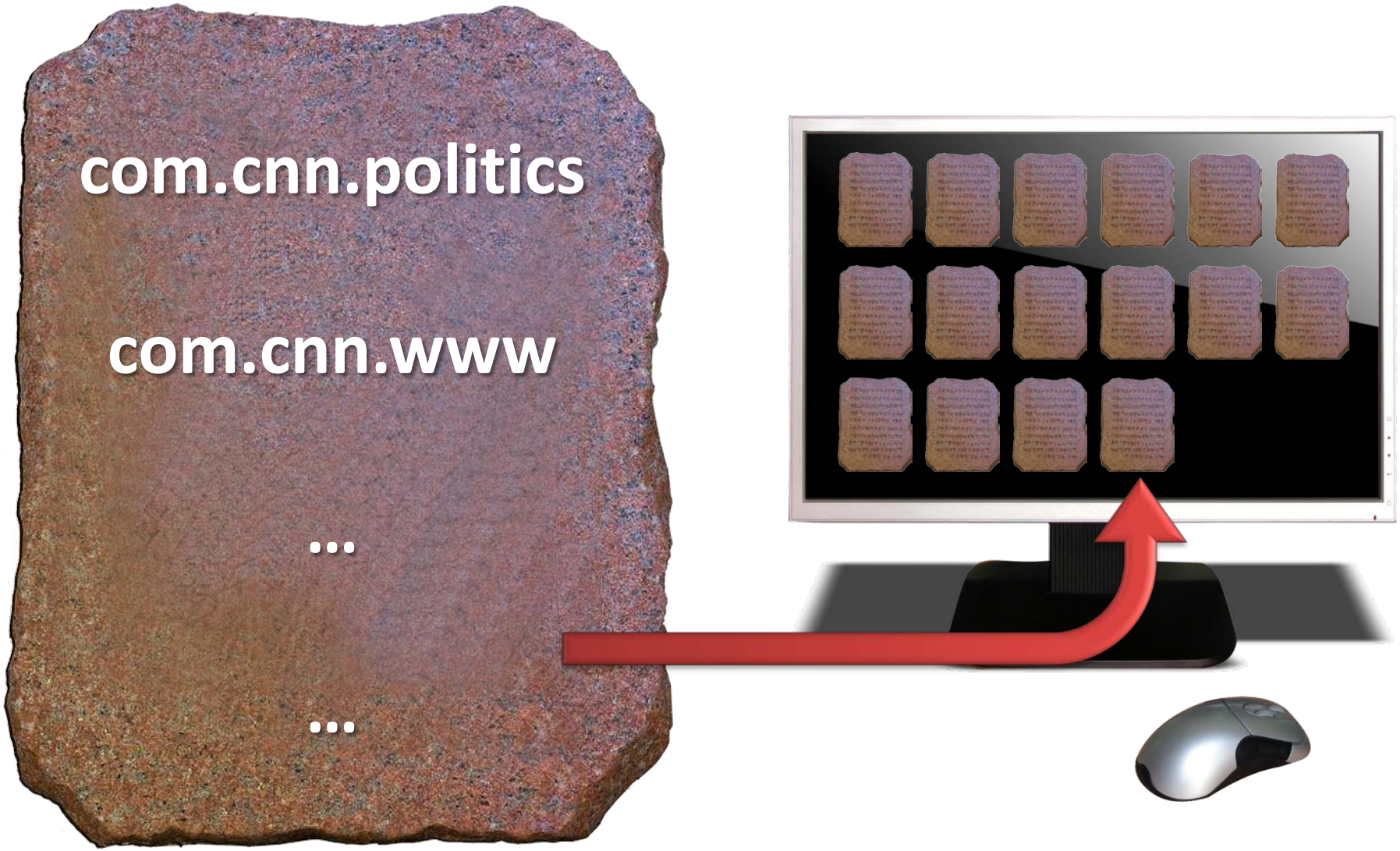
- Bigtable is a sparse, distributed, persistent, multidimensional sorted map
- (row:string, column:string, time:int64) -> string
- Rows are ordered lexicographically and grouped together in tablets
- Columns are grouped in column families
- Each cell may contain multiple versions of the same data – timestamped with either the real time or client generated timestamps

Data Model



- **Atomic row operations**
- **Column family level access control and garbage collection settings**

Data Model



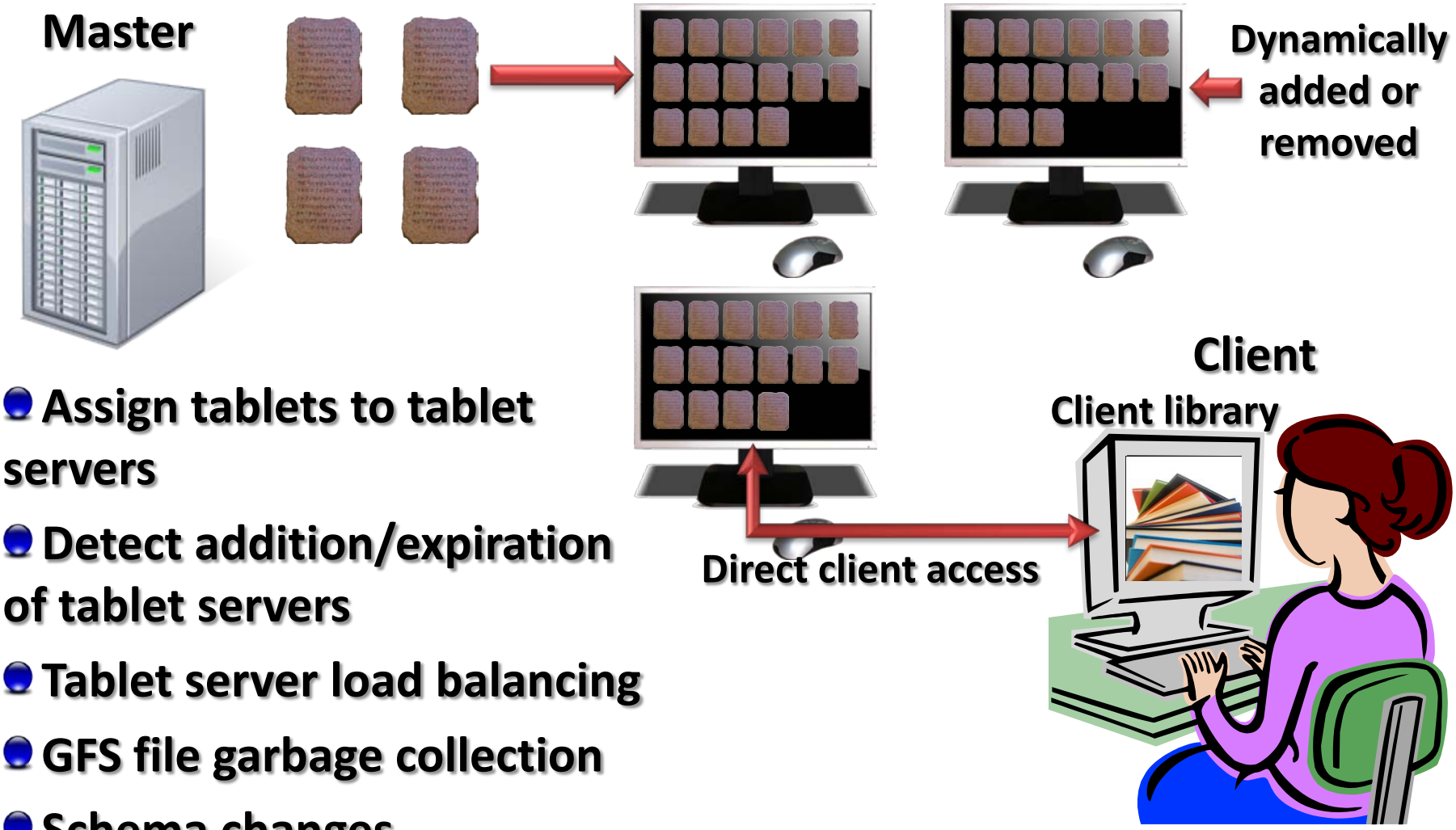
API

- An API is provided to handle relevant Bigtable functions
- Regular expressions for lookups
- Single-row transactions are supported
- Execution of client Sawzall scripts in the server's address space
- Wrappers are provided to allow Bigtable to act as an input source or output target to MapReduce

SSTable

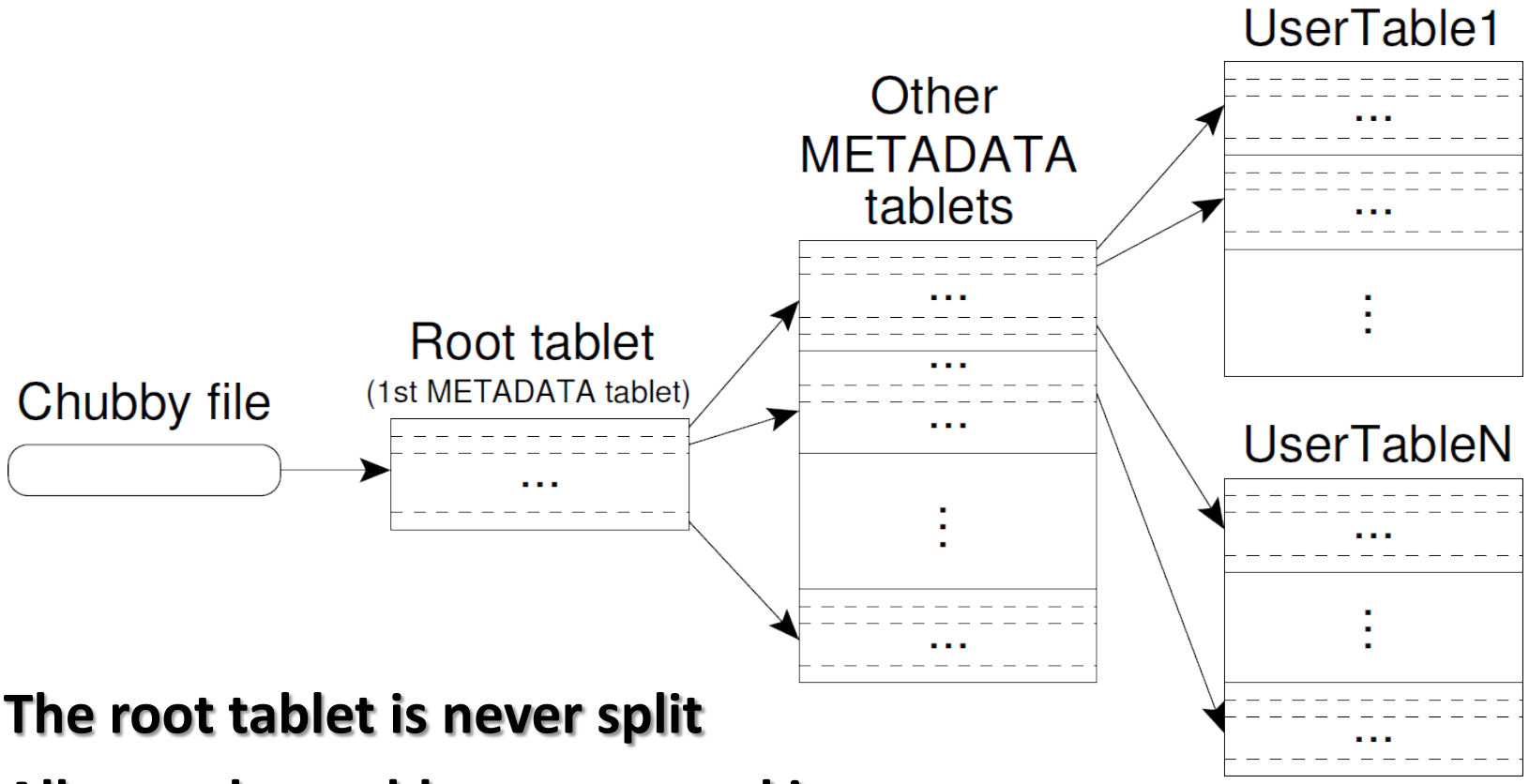
- An SSTable provides a persistent ordered immutable map from keys to values (strings)
- Split into blocks (typically 64KB in size)
- A block index is stored at the end of the SSTable
- The block index is loaded into memory when the SSTable is open
- Optionally the entire SSTable may be loaded in memory

Implementation



- Assign tablets to tablet servers
- Detect addition/expiration of tablet servers
- Tablet server load balancing
- GFS file garbage collection
- Schema changes

Tablet Location



- The root tablet is never split
- All metadata tablets are stored in memory
- 128 MB / tablet is sufficient to address 2^{34} tablets
- The client library caches tablet location

Tablet Assignement

- Tablet servers each acquire a lock on a Chubby file, allowing the master to keep track of them
- If the file no longer exists the server kills itself
- The master assigns tablets to tablet servers (the list of all tablets is kept in the METADATA table)
- The master handles tablet creation, deletion and merging
- The tablet servers handle tablet splitting

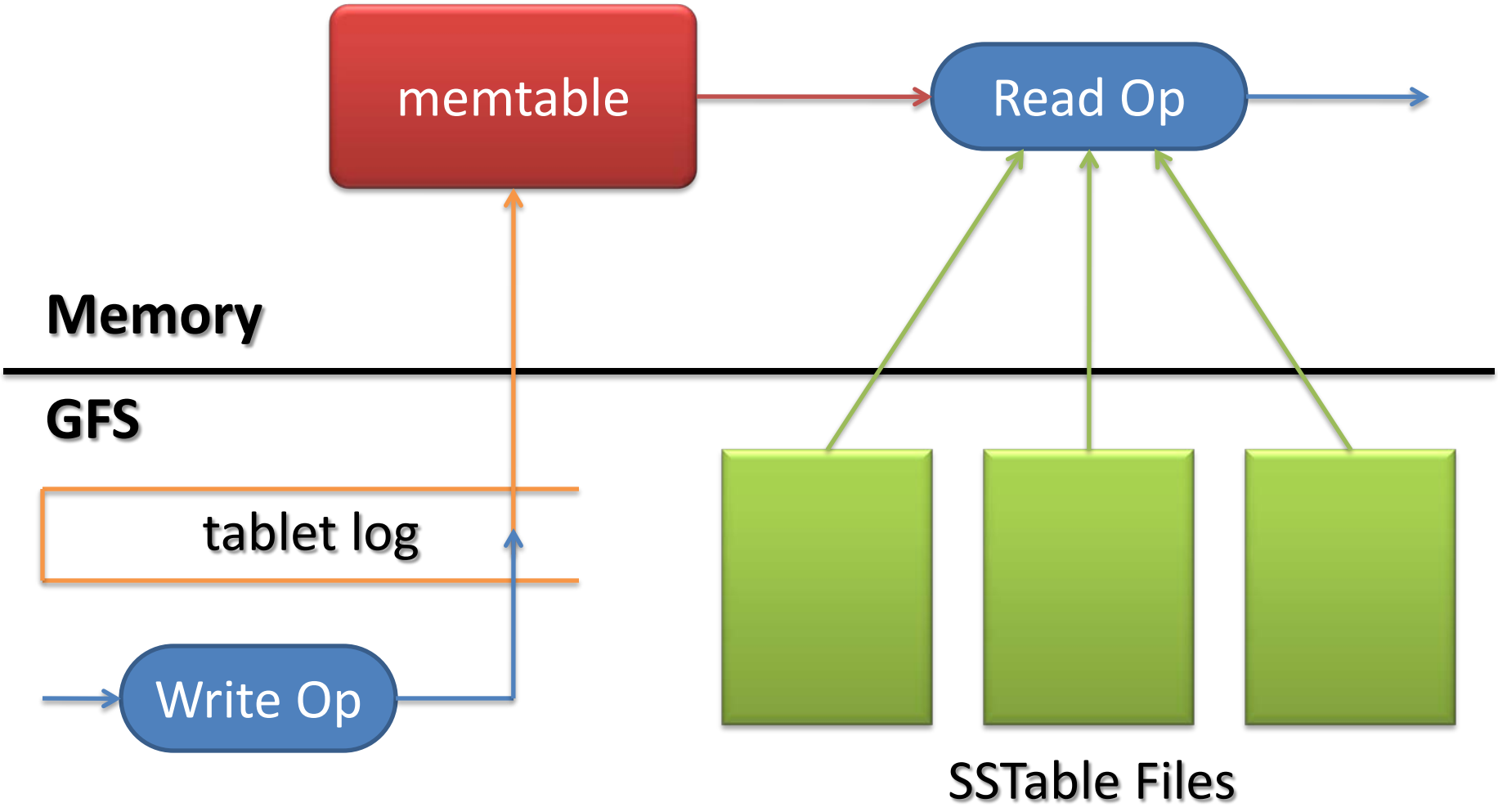
Tablet Splitting

- When tablets become too large (typically 100-200 MB, tablet servers will split the tablet into 2 parts
- The process involves creating a new tablet and committing the operation by adding the new information in the METADATA table
- The master is notified after the commit
- If the notification is lost, the master will be informed when it asks a tablet server to load the initial tablet -> the tablet server will only see part of the tablet it was asked to load when querying the METADATA table

Tablet Storing

- A tablet comprises of a list of immutable SSTables stored under GFS
- Recently committed operations are stored in memory in so called memtables
- Commit logs are kept to ensure recovery from failure
- “redo points” stored in the METADATA table are just pointers to entries in these commit logs
- Memtables are compacted into SSTables once they reach a certain size (minor compaction)
- Multiple SSTables are compacted together to speed up read operations (major/merging compaction)

Tablet Serving



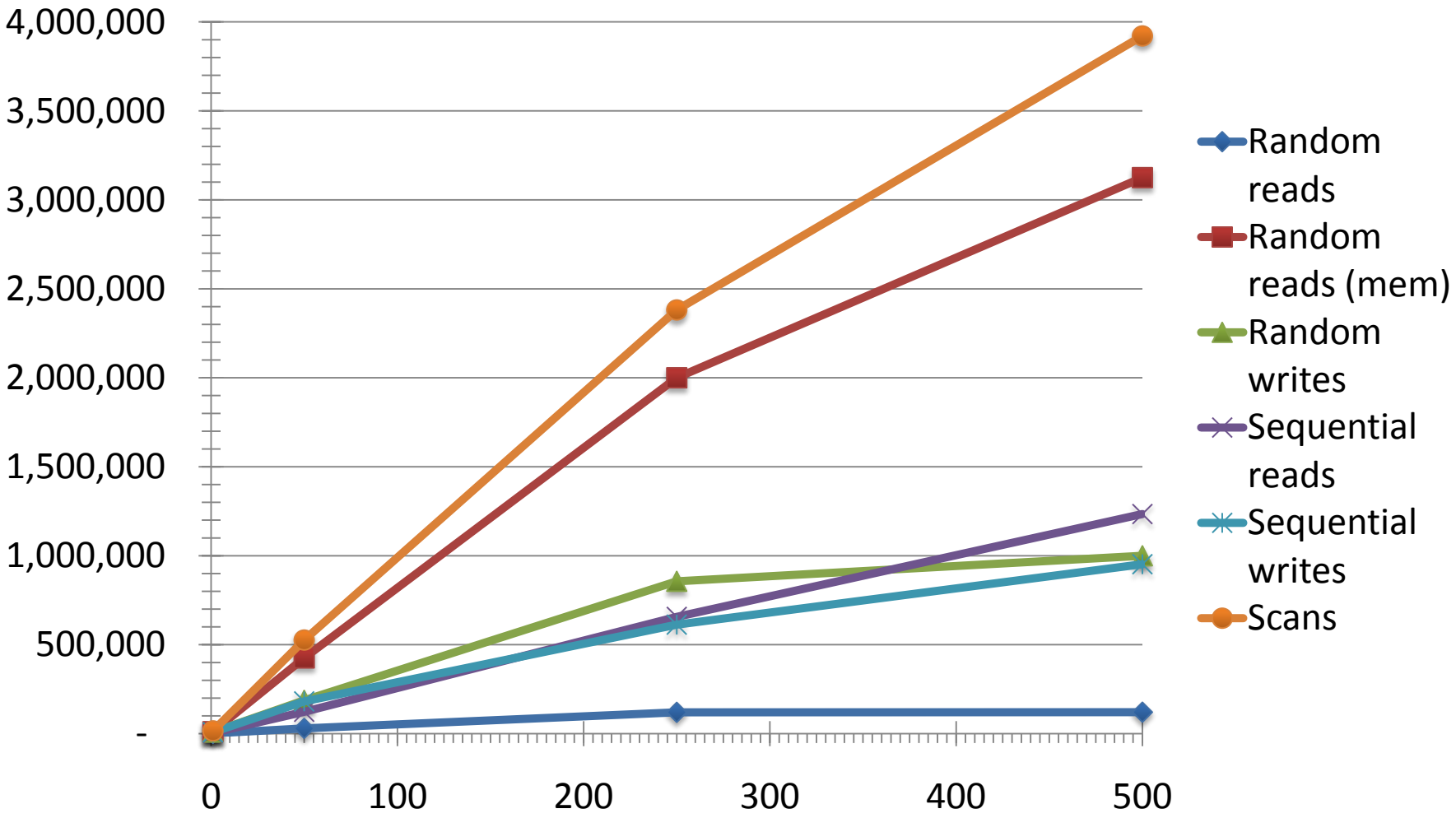
Refinements

- Locality groups (for column families)
- Compression (block level)
- Caching (Scan Cache and Block Cache)
- Bloom filters
- Tablet server level commit-log
- Minor compactions before tablet movement
- Exploiting immutability

Performance Evaluation

Experiment	Number of Tablet Servers			
	1	50	250	500
Random reads	1,212	593	479	241
Random reads (mem)	10,811	8,511	8,000	6,250
Random writes	8,850	3,745	3,425	2,000
Sequential reads	4,425	2,463	2,625	2,469
Sequential writes	8,547	3,623	2,451	1,905
Scans	15,385	10,526	9,524	7,843

Performance Evaluation



Conclusion

- Bigtable provides an unconventional alternative to distributed databases
- It offers great scalability and performance
- Users have increased flexibility, but intelligent schema designs are required in order to maintain performance at high levels
- It plays a pivotal role in Google's infrastructure, being used by over 60 deployed products

Discussion Points

- How big of a problem is the lack of general transaction support ?
- Is the performance of reading operations too low (especially if the data accessed is relatively new) ?
- The system performs well when faced with Google's application needs; will it fare as well in other types of applications ?
- Comparison with standard distributed database systems.