

# Chubby and Bigtable

# Bigtable

---

- ◆ Distributed storage system for managing structured data such as:
  - URLs: contents, crawl metadata, links, pagerank ...
  - Per-user data: preferences, recent search results ...
  - Geolocations: physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, ...
- ◆ Used for many Google applications
  - Web indexing, personalized Search, Google Earth, Google Analytics, Google Finance, ... and more

# Scalability Requirements

---

- ◆ **Petabytes** of data distributed across thousands of servers Millions of machines
  - Hundreds of millions of users
  - Billions of URLs, many versions/page
  - Thousands of queries/sec
  - 100TB+ of satellite image data
- ◆ Low-level storage optimization helps performance significantly

# Goals

---

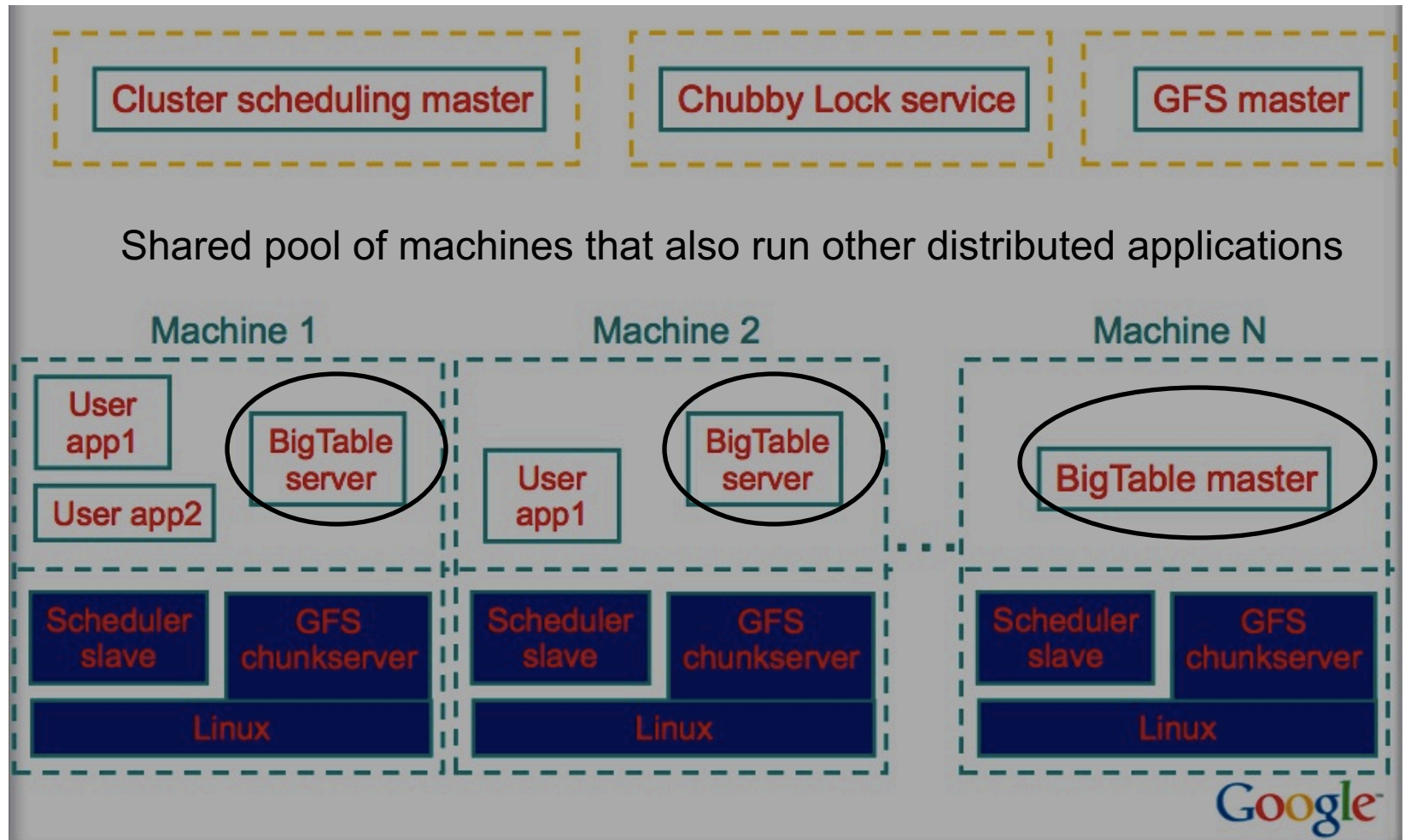
- ◆ Asynchronous processes continuously update different pieces of data
- ◆ Examine data changes over time: e.g. contents of a web page over multiple crawls
- ◆ Requirements
  - Simpler model that supports dynamic control over data and layout format
  - Very high read/write rates (millions ops per second)
  - Efficient scans over all or subsets of data
  - Efficient joins of large one-to-one and one-to-many datasets

# Design Overview

---

- ◆ Distributed multi-level map
- ◆ Fault-tolerant, persistent
- ◆ Scalable
  - Thousands of servers
  - Terabytes of in-memory data
  - Petabyte of disk-based data
  - Millions of reads/writes per second, efficient scans
- ◆ Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to load imbalance

# Typical Google Cluster



# 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

# Chubby

---

- ◆ {lock/file/name} service
- ◆ Coarse-grained locks
  - Provides a namespace that consists of directories and small files
  - Each of the directories or files can be used as a lock
- ◆ Each client has a session with Chubby
  - Expires if it is unable to renew its session lease within the lease expiration time.
- ◆ 5-replica Paxos, need a majority vote to be active



# Chubby

---

- ◆ A coarse-grained lock service
  - Provides a means for distributed systems to synchronize access to shared resources
  - Held for hours or days
- ◆ Looks like a file system
- ◆ Known, available, highly reliable location to store small amount of metadata
  - “Root” for bootstrapping distributed data structures

# How is Chubby Used at Google

---

- ◆ GFS: master election
- ◆ BigTable: master election, client discovery, table service locking
- ◆ Partition workloads
- ◆ **Name service** because of its consistent client caching
  - Replacement for DNS inside Google

# External Interface

---

- ◆ Organized as cells
  - 5-replica Paxos, needs majority vote to be active
- ◆ Presents a simple **distributed file system**
- ◆ Clients can open/close/read/write files
  - Reads and writes are whole-file
  - Supports advisory reader/writer locks
  - Clients can register for notification of file update

# Files as Locks

---

Files can have several attributes:

- ◆ The contents of the file is one (primary) attribute
- ◆ Owner of the file
- ◆ Permissions
- ◆ Date modified
- ◆ Whether the file is locked or not

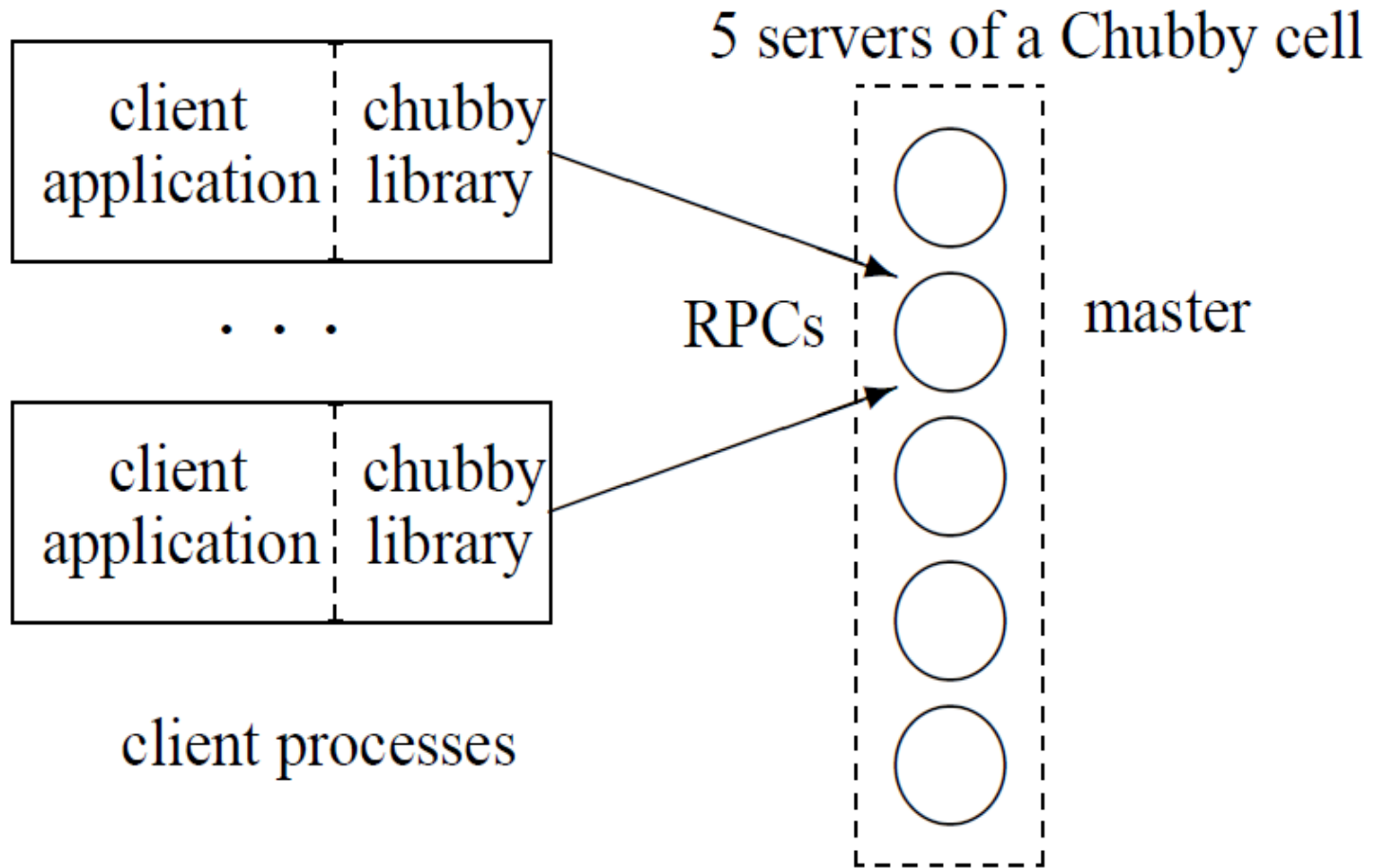
# Using Chubby for Master Election

---

- ◆ All replicas try to acquire a write lock on a designated file
- ◆ The replica who gets the lock is the master
- ◆ Master can then write its address to file; other replicas can read this file to discover the chosen master name

# Chubby Cell

---



# Chubby and Consensus

---

- ◆ Chubby cell is usually 5 replicas
  - 3 replicas must be alive for cell to work
  - Tolerates 2 failures
- ◆ Replicas must agree on their own master and official lock values
  - Replicas promise not to try to elect new master for at least a few seconds (“master lease”)
- ◆ Uses Paxos for consensus
  - Memory for individual “facts” in the network
  - A fact is a binding from a variable to a value

# Client Updates

---

- ◆ All replicas are listed in DNS
- ◆ All client updates go through master
- ◆ Master updates official database; sends copy of update to replicas
  - Majority of replicas must acknowledge receipt of update before master writes its own value
- ◆ Clients find master through DNS
  - Contacting replica causes redirect to master



# APIs

---

## ◆ Open()

- Mode: read/write/change ACL; Events; Lock-delay
- Create new file or directory?

## ◆ Close()

## ◆ GetContentsAndStat(), GetStat(), ReadDir()

## ◆ SetContents(): set all contents; SetACL()

## ◆ Delete()

## ◆ Locks: Acquire(), TryAcquire(), Release()

## ◆ Sequencers: GetSequencer(), SetSequencer(), CheckSequencer()

# Example: Primary Election

---

```
Open("write mode");
If (successful) {
    // primary
    SetContents("identity");
}
Else {
    // replica
    open ("read mode", "file-modification event");
    when notified of file modification:
        primary= GetContentsAndStat();
}
```

# Client Caching

---

- ◆ Clients cache all file content w/ **strict consistency**
  - Lease-based
  - Master invalidates cached copies upon a write request
- ◆ Client must respond to keep-alive message from server at frequent intervals
  - Keep-alive messages include invalidation requests
  - Responding to keep-alive implies acknowledgement of cache invalidation
- ◆ Modification only continues after all caches invalidated or keep-alive times out

# Scalability

---

- ◆ 90K+ clients communicate with a single Chubby master (2 CPUs)
- ◆ System increases lease times from 12 sec up to 60 secs under heavy load
- ◆ Clients cache virtually everything
- ◆ Data is small – all held in RAM (as well as disk)

# Bigtable Data Model

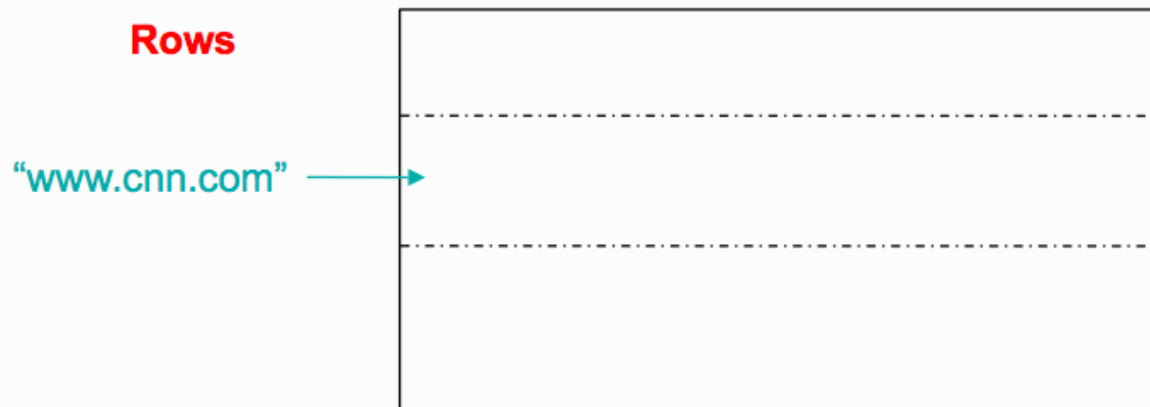
---

- ◆ A sparse, distributed, persistent multi-dimensional sorted map
- ◆ (row, column, timestamp) -> cell contents
  - Rows, column are arbitrary strings

# Rows

---

- ◆ Arbitrary string
- ◆ Access to data in a row is atomic
  - Row creation is implicit upon storing data
  - Ordered lexicographically



# Rows

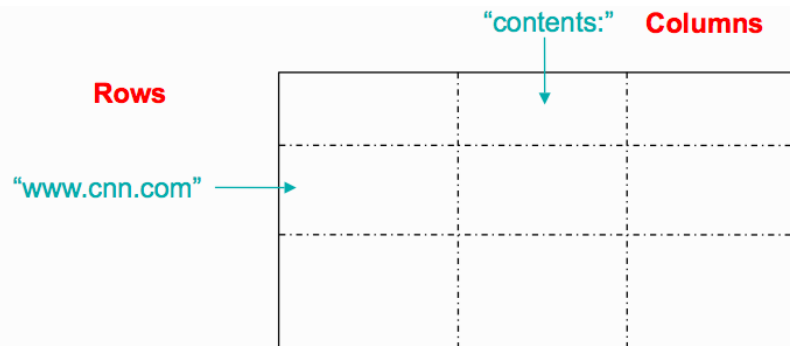
---

- ◆ Rows close together lexicographically usually on one or a small number of machines
- ◆ Reads of short row ranges are efficient and typically require communication with a small number of machines
- ◆ Can exploit lexicographic order by selecting row keys so they get good locality for data access
  - Example: `edu.cornell.tech`, `edu.cornell.cs` ... instead of `tech.cornell.edu`, `cs.cornell.edu`

# Columns

---

- ◆ Two-level name structure: **family: qualifier**
- ◆ Family is the unit of access control
  - Has associated type information
- ◆ Qualifier gives unbounded columns
  - Additional levels of indexing, if desired





# Example

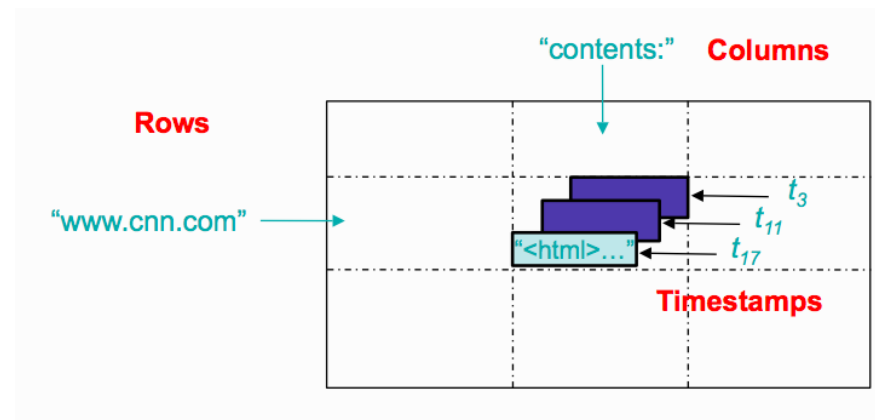
<https://cloud.google.com/bigtable/docs/overview>

	"follows" column family			
	Follows			
Row Key	gWashington	jadams	tjefferson	wmckinley
gWashington		1		
jadams	1		1	
tjefferson	1	1		1
wmckinley			1	

- Multiple versions

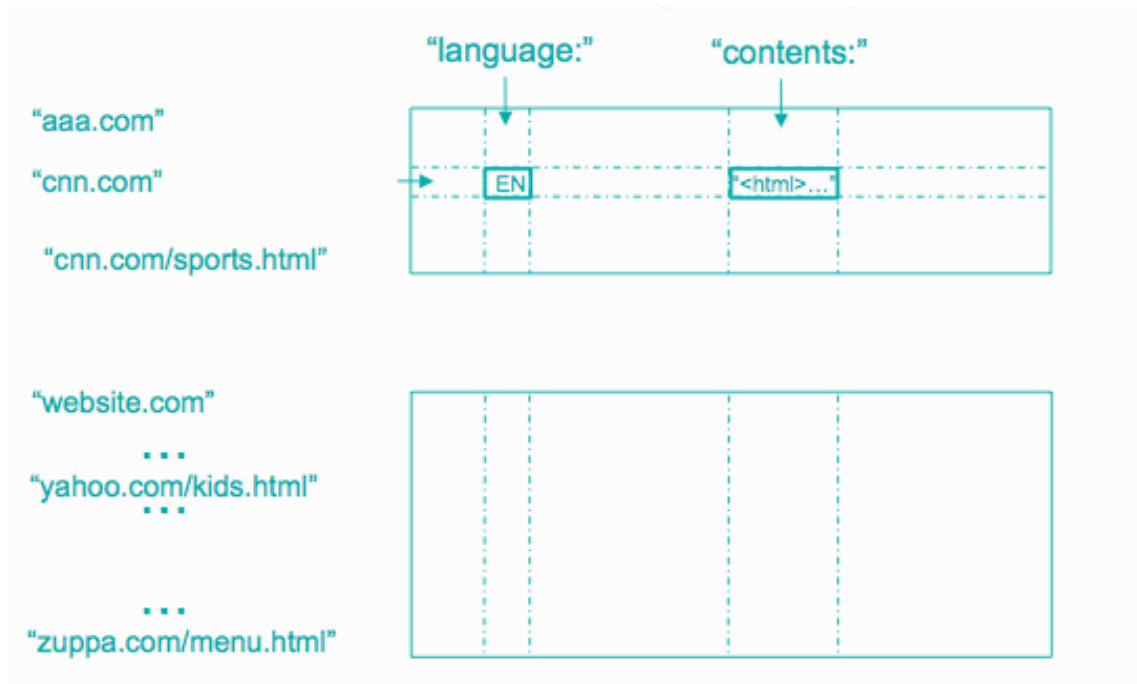
# Timestamps

- ◆ Store different versions of data in a cell
  - New writes default to current time, but timestamps for writes can also be set explicitly by clients
- ◆ Lookup options
  - Return most recent K values
  - Return all values
- ◆ Column families can be marked w/ attributes
  - Retain most recent K values in a cell
  - Keep values until they are older than K seconds



# Tablet

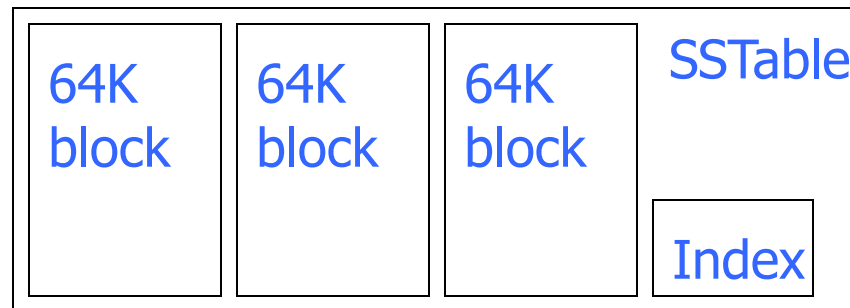
- ◆ The row range for a table is dynamically partitioned, each range is called a **tablet**
- ◆ Tablet is the unit for distribution and load balancing



# Storage: SSTable

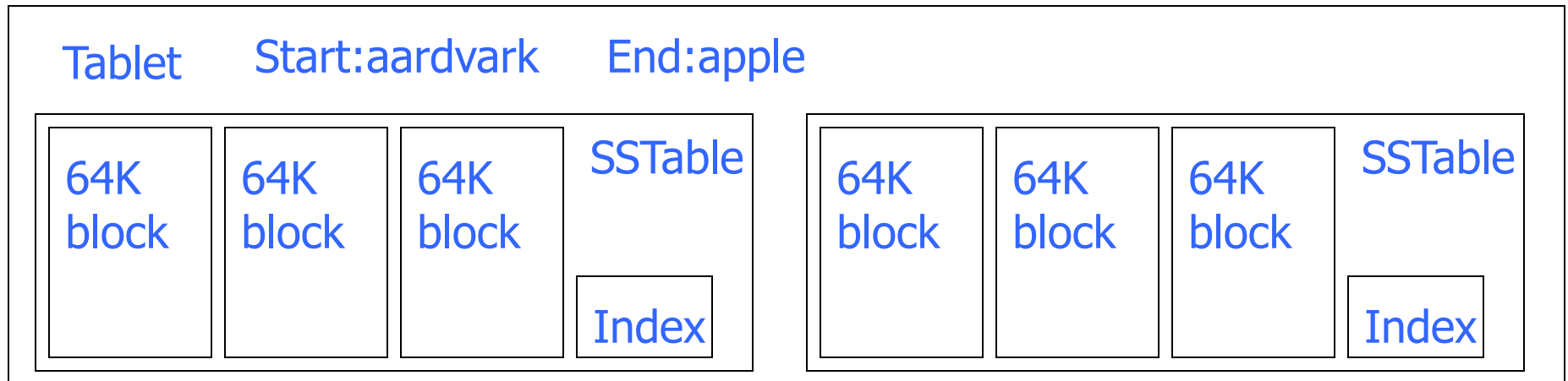
---

- ◆ Immutable, sorted file of key-value pairs
- ◆ Can be completely mapped into memory (option)
- ◆ Chunks of data plus an index
  - Index is of block ranges, not values
  - Index is loaded into memory when SSTable is open



# Tablet vs. SSTable

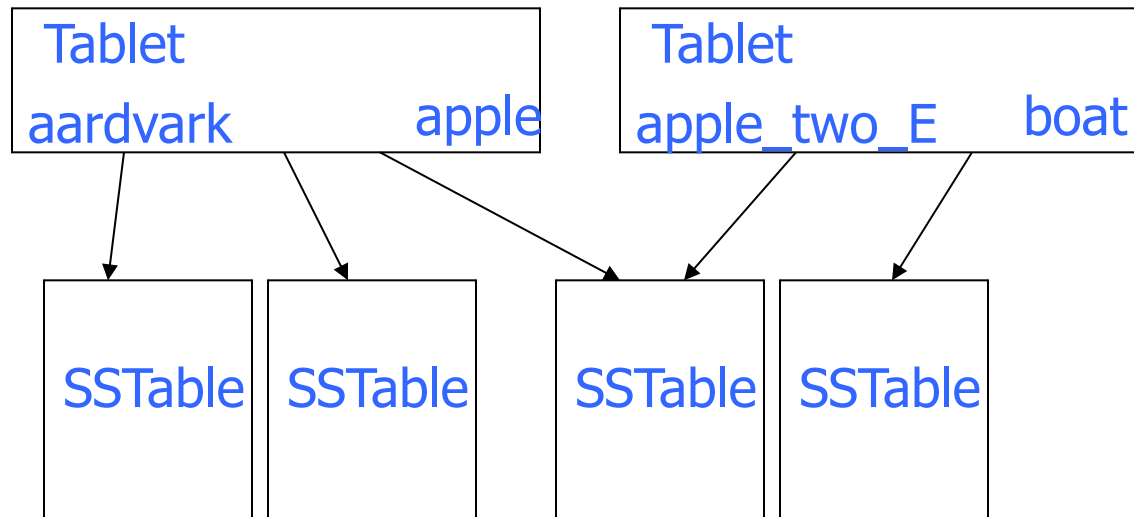
# Tablet is built out of multiple SSTables



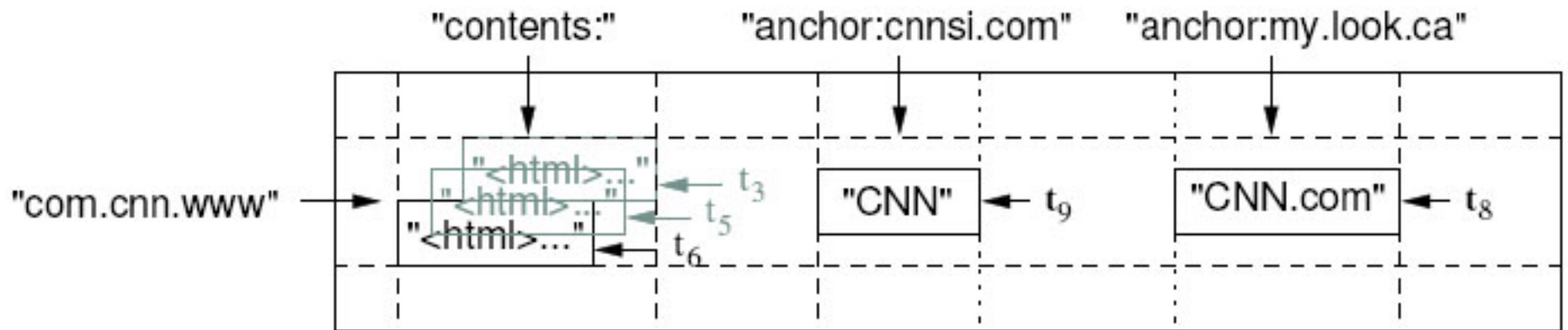
# Table vs. Tablet vs. SSTable

---

- ◆ Multiple tablets make up the table
- ◆ SSTables can be shared
- ◆ Tablets do not overlap, SSTables can overlap



# Example: WebTable



- ◆ Want to keep copy of a large collection of web pages and related information
- ◆ Use URLs as row keys
- ◆ Various aspects of web page as column names
- ◆ Store contents of web pages in the "contents:" column under the timestamps when they were fetched

# Implementation

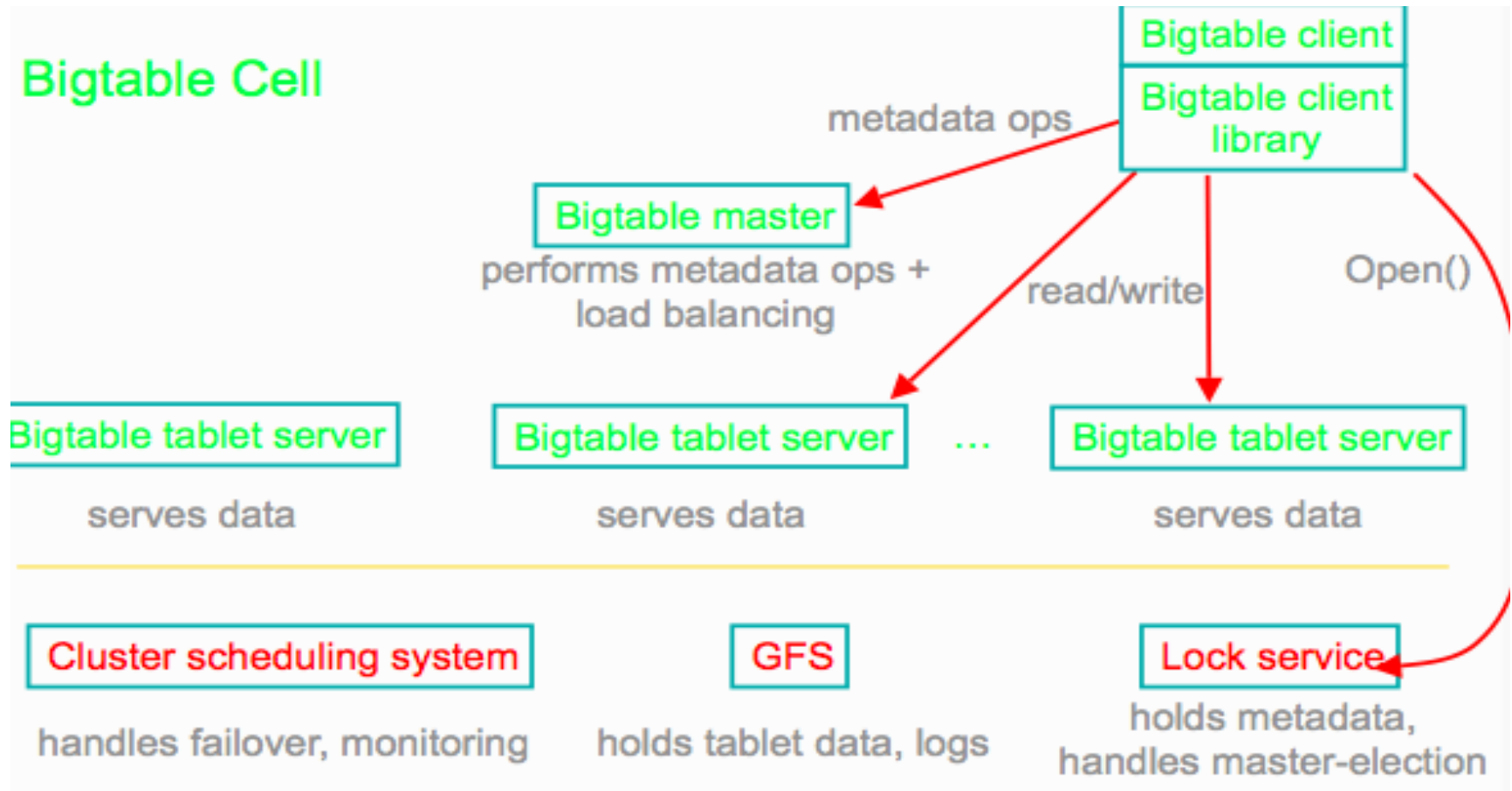
---

- ◆ Library linked into every client
- ◆ One master server responsible for:
  - Assigning tablets to tablet servers, balancing load
  - Detecting addition and expiration of tablet servers
  - Garbage collection
  - Handling schema changes, eg, table and column family creation
- ◆ Many tablet servers, each of them:
  - Handles read and write requests to its table
  - Splits tablets that have grown too large
- ◆ Clients communicate directly with tablet servers



# Deployment

## Bigtable Cell



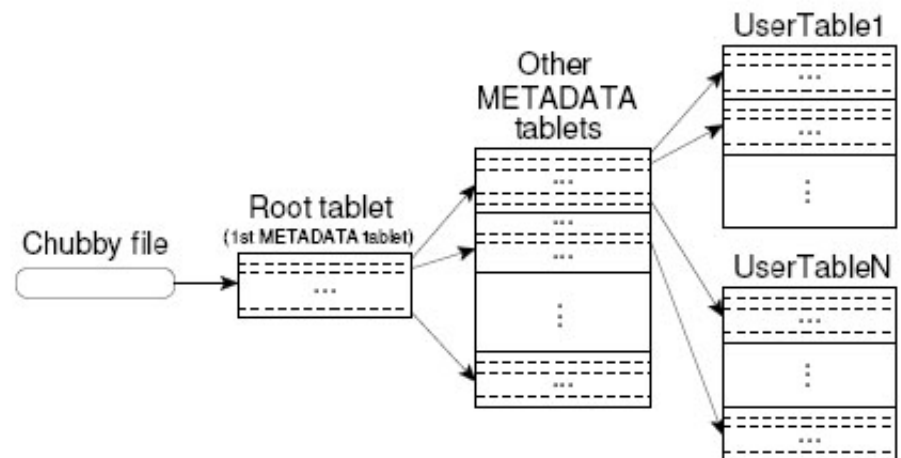
# More about Tablets

---

- ◆ Each tablet server responsible for 10 – 1000 tablets (usually about 100)
- ◆ Fast recovery
  - 100 machines each pick up 1 tablet for failed server
- ◆ Fine-grained load balancing
  - Migrate tablets away from overloaded machine
  - Master makes load-balancing decisions

# Tablet Location

- ◆ How do clients find the right machine for a row?
  - Find tablet whose row range covers the target row
  - Tablets move around from server to server
- ◆ METADATA
  - Key: table id + end row, Data: location
- ◆ Aggressive caching and prefetching at client side



# Tablet Assignment

---

- ◆ Each tablet assigned to one tablet server at a time
- ◆ Master server
  - Keeps track of the set of live tablet servers and current assignments of tablets to servers
  - Keeps track of unassigned tablets
- ◆ When a tablet is unassigned, master assigns the tablet to a tablet server with sufficient room
- ◆ Master uses Chubby to monitor health of tablet servers and restart/replace failed servers

# Tablet Assignment with Chubby

---

- ◆ Tablet server registers itself with Chubby by getting a lock in a specific directory of Chubby
- ◆ “Lease” on lock must be renewed periodically
- ◆ Master monitors this directory to find which servers exist/are alive
  - If server not contactable or has lost lock, master grabs lock and reassigns tablets
- ◆ Prefer to start tablet server on same machine that the data is already at
  - Data replicated by GFS

# Bigtable API

---

## ◆ Metadata operations

- Create/delete tables, column families, change metadata

## ◆ Writes (atomic)

- Set(): write cells in a row
- DeleteCells(), DeleteRow()

## ◆ Read

- 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 families, or specific columns

# Refinements: 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.

# Refinements: Compression

---

- ◆ Many opportunities for compression
  - Similar values in the same row/column at different timestamps
  - Similar values in different columns
  - Similar values across adjacent rows
- ◆ Two-pass custom compression scheme
  - First pass: compress long common strings across a large window
  - Second pass: look for repetitions in small window
- ◆ Speed emphasized, but good (10x) space reduction



# Refinements: Bloom Filters

---

- ◆ Read operation has to read from disk when desired SSTable is not in memory
- ◆ Reduce number of accesses by specifying a Bloom filter
  - Allows to ask if a SSTable might contain data for a specified row/column pair
  - Small amount of memory for Bloom filters drastically reduces the number of disk seeks for read operations
  - Results in most lookups for non-existent rows or columns not needing to touch disk

# Real Applications

---

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

HBase is the open-source version

# Limitations

---

- ◆ No transactions
- ◆ Does not support full relational data model
- ◆ Achieved throughput is limited by GFS
- ◆ Difficult to use for applications that (1) have complex, evolving schemas; (2) want strong consistency in the presence of wide-area replication