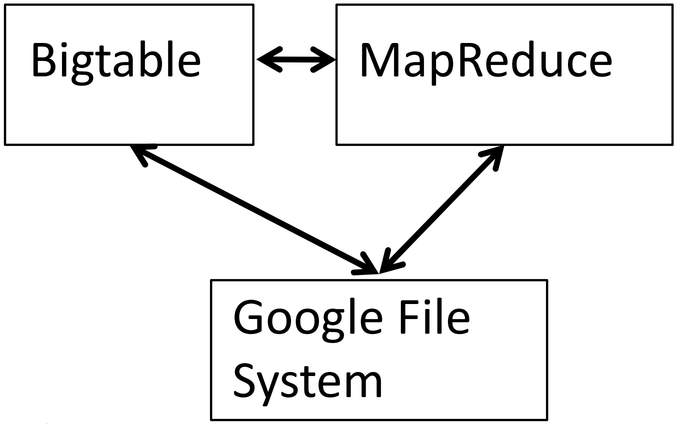
Cloud and Cluster Data Management

BIGTABLE – PART 1

Bigtable – Motivation & Uses

- Store structured data – scale to large size
  - Petabytes on commodity servers
- Goals:
  - Wide applicability
  - Scalablity
  - High performance
  - High availability



- Uses: Web indexing, Google Earth, Google Finance Flexibility:
  - Data type and size: URLs vs. Satellite imagery
  - **Throughput-oriented batch jobs vs. latency-sensitive jobs**
    - bulk processing, real-time data serving
- Updates:
  - Bigtable is now used in more Google products, including Google Cloud AI services and Ads platforms.
  - Modern use cases include time-series data, IoT applications, and real-time analytics.

Bigtable – Key Concepts

- Column family data model
- Dynamic control over data layout and format
- Control whether to serve data out of memory vs. disk
- Data is uninterpreted strings (no types)
- Partitioning: both horizontal and vertical
- Transactions: single-row only
  - But rows are heavyweight

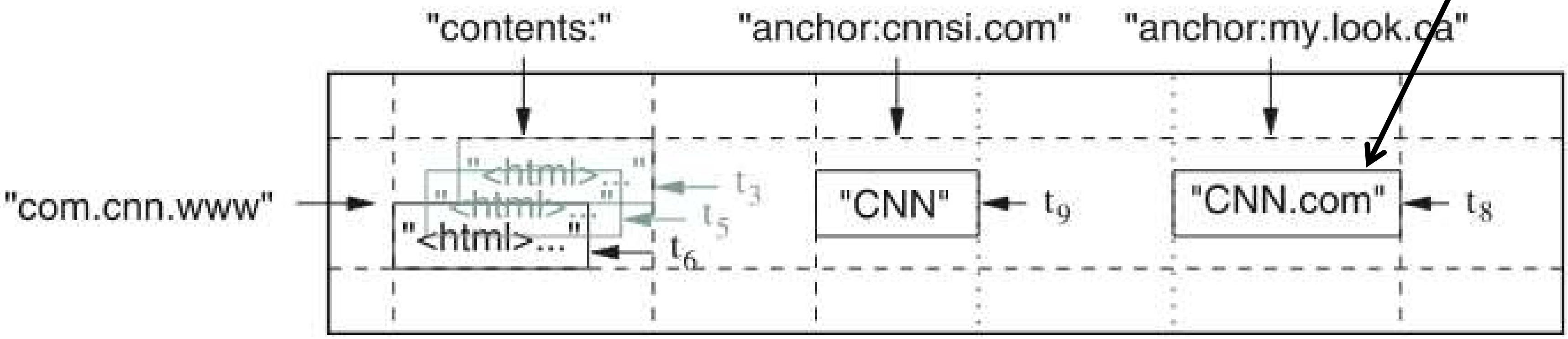
Updates:

- Now integrates more effectively with Google Cloud services, including Dataflow and BigQuery, for seamless analytics.
- New features allow for better schema design and access control policies.

Bigtable – Tables

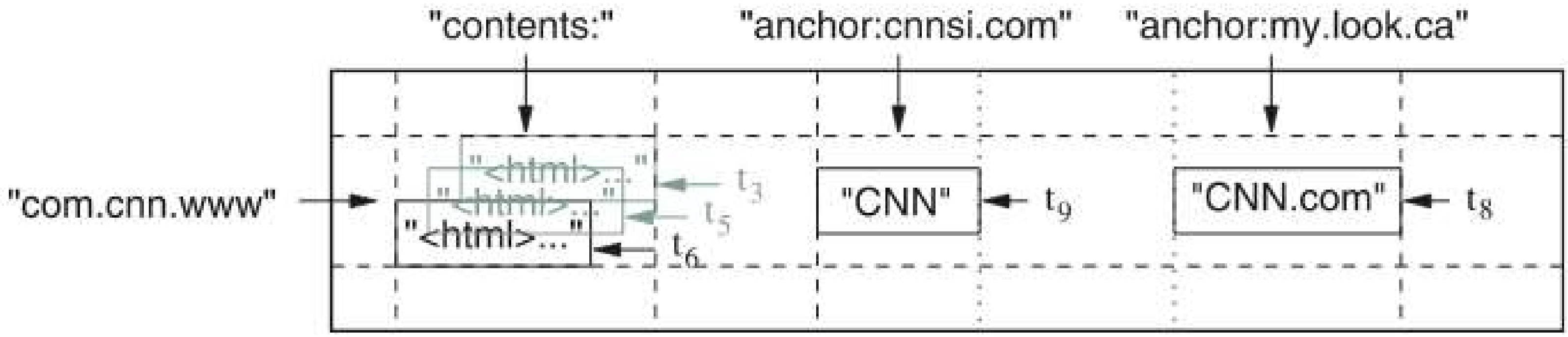
- Bigtable *cluster* – set of processes that runs Bigtable software
  - Each process serves a set of tables
- Tables:
  - **Sparse, Distributed, Persistent, Multi-dimensional Sorted map**
- Three Dimensions:
  - row: string
  - column: string
  - time: int64
- (row:string, column:string, time:int64) -> (uninterpreted) string

(row key, column key, timestamp) -> cell



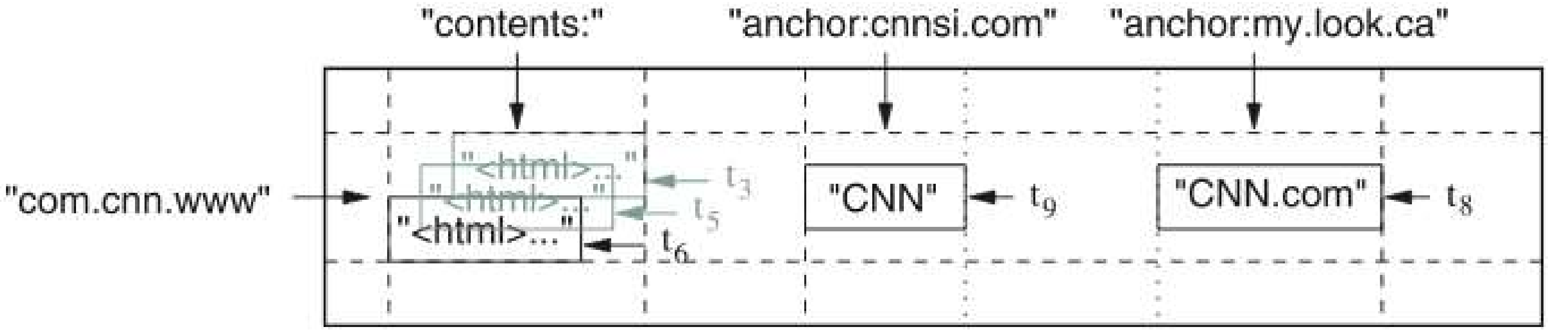
Bigtable – Rows

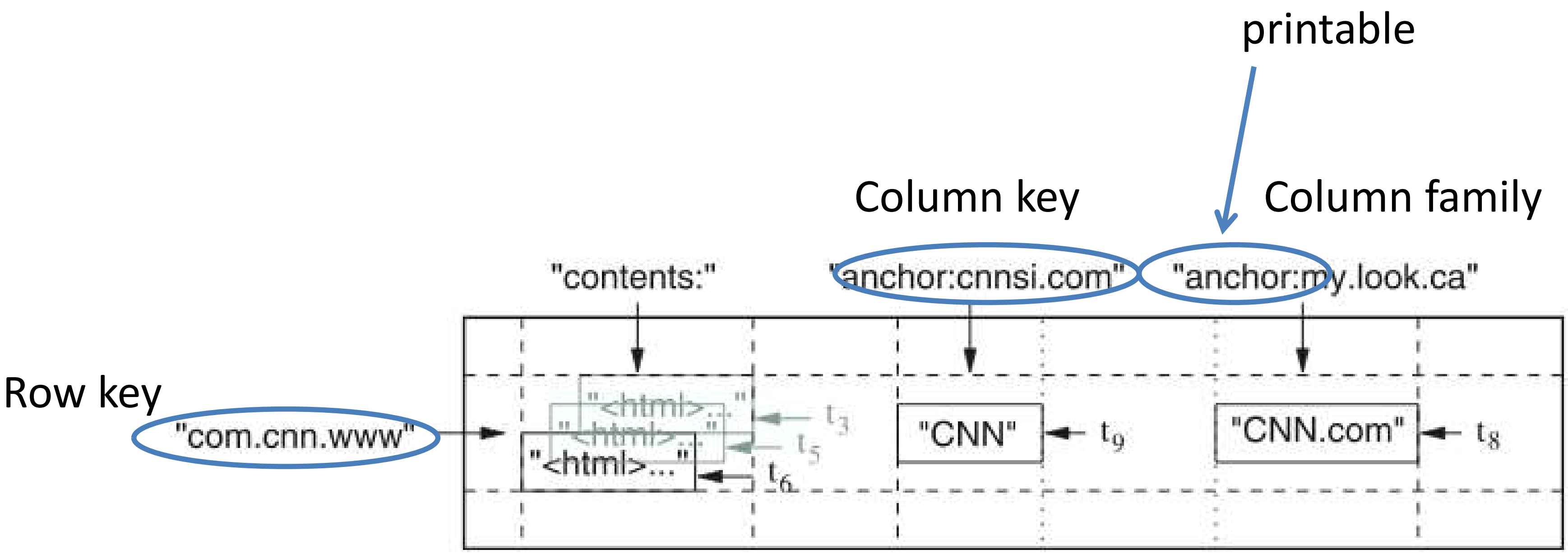
- Rows – kept ordered by row key
  - Choice of row key important
    - Select row key to get good locality of data access (i.e. reverse hostname)
    - Small row ranges → small # machines to access
  - Rows with consecutive keys grouped into *tablets*
  - Partitioning attribute is fixed (in contrast with RDBMS)
  - # rows in a table is unbounded
- **Tablets are unit of distribution and load balancing**
- **Row is unit of transactional consistency**
  - Row read/writes are serializable
  - No transactions across rows



Bigtable – Columns

- Columns grouped into Column Families
  - Data stored in column families is usually of the same type (compressed together)
  - Number of **column families intended to be small (unlimited rows, cols)**
    - Keeps shared meta-data small
  - **Column families must be created explicitly**
  - Column key is family:qualifier
- Example column families:
  - language:<> cell contains: language id
  - anchor: <referring site> cell contains: text associated with link
- **Column family is unit of access control**





printable

Column key

Column family

Row key

Bigtable - Timestamps

- Multiple version of the data in a cell - indexed by timestamp
- Timestamps assigned implicitly by Bigtable or explicitly by clients

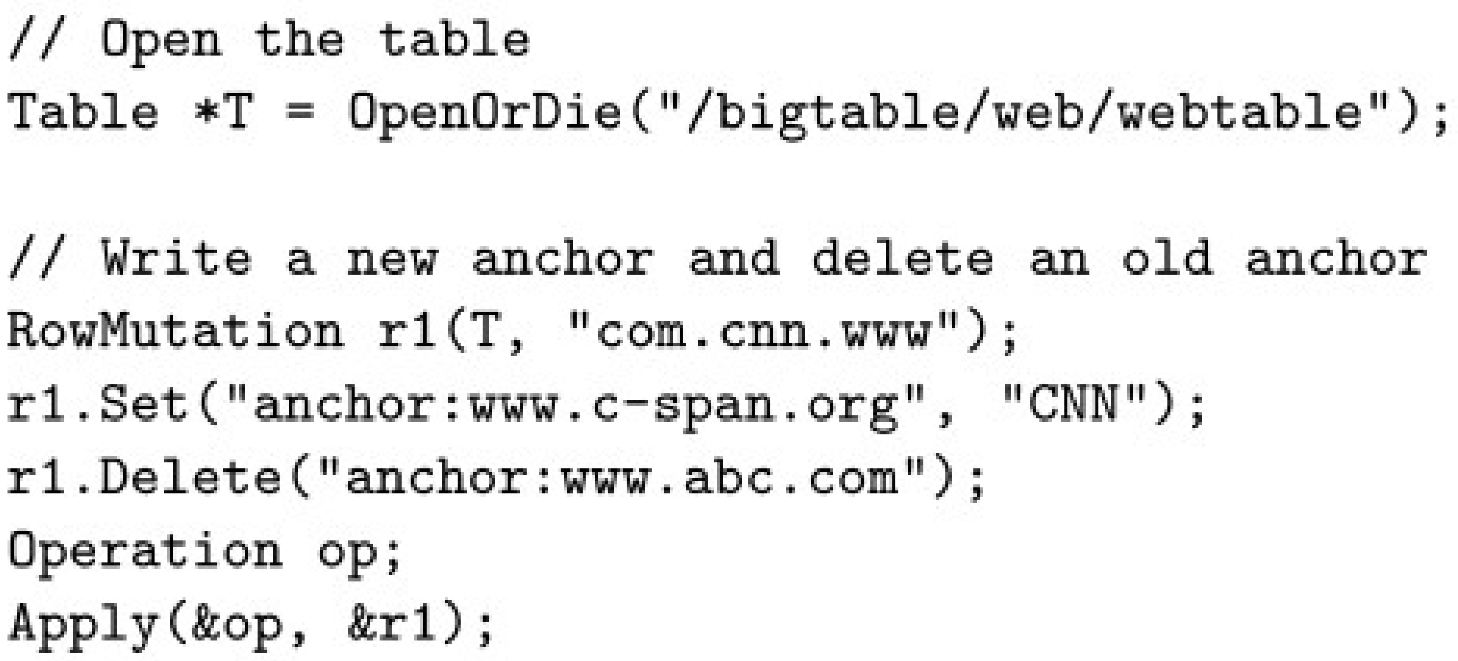
- Bigtable stores in decreasing timestamp order (most recent version read first)

- Garbage collection options:
  - Keep last N versions
  - Keep last D days of data
- Webtable ex: timestamps are times pages were crawled

Bigtable API

- Create and delete tables and column families
- Change cluster, table and column family meta-data
- Single-row transactions
  - Atomic read-modify-write sequences on a single row
- Does not support transactions across row keys

Bigtable API – Row Mutation Example

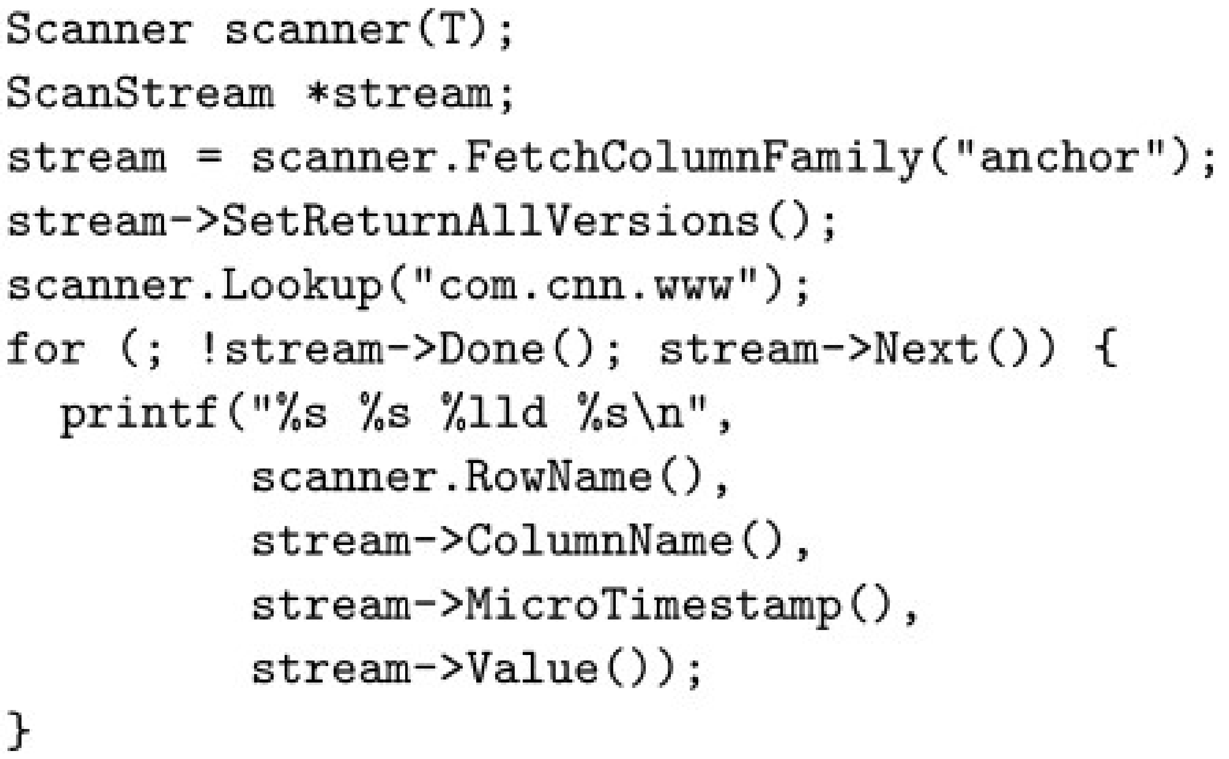


```
// Open the table
Table *T = OpenOrDie("/bigtable/web/webtable");

// Write a new anchor and delete an old anchor
RowMutation r1(T, "com.cnn.www");
r1.Set("anchor:www.c-span.org", "CNN");
r1.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &r1);
```

"Irrelevant details were elided to keep the example short."

Bigtable API – Scanner Example

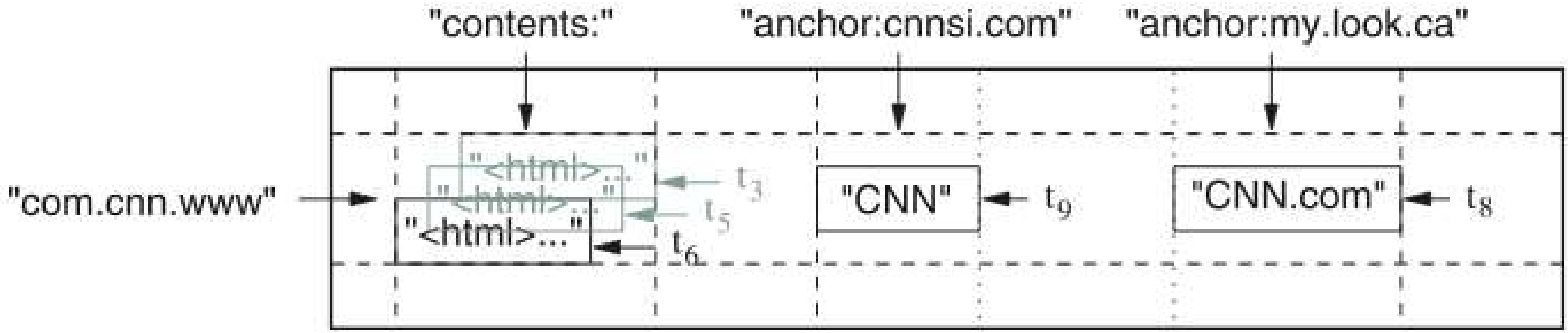


```
Scanner scanner(T);
ScanStream *stream;
stream = scanner.FetchColumnFamily("anchor");
stream->SetReturnAllVersions();
scanner.Lookup("com.cnn.www");
for (; !stream->Done(); stream->Next()) {
  printf("%s %s %lld %s\n",
         scanner.RowName(),
         stream->ColumnName(),
         stream->MicroTimestamp(),
         stream->Value());
}
```

Could restrict the scan:

- produce only anchors whose columns match *anchor:*.cnn.com*,

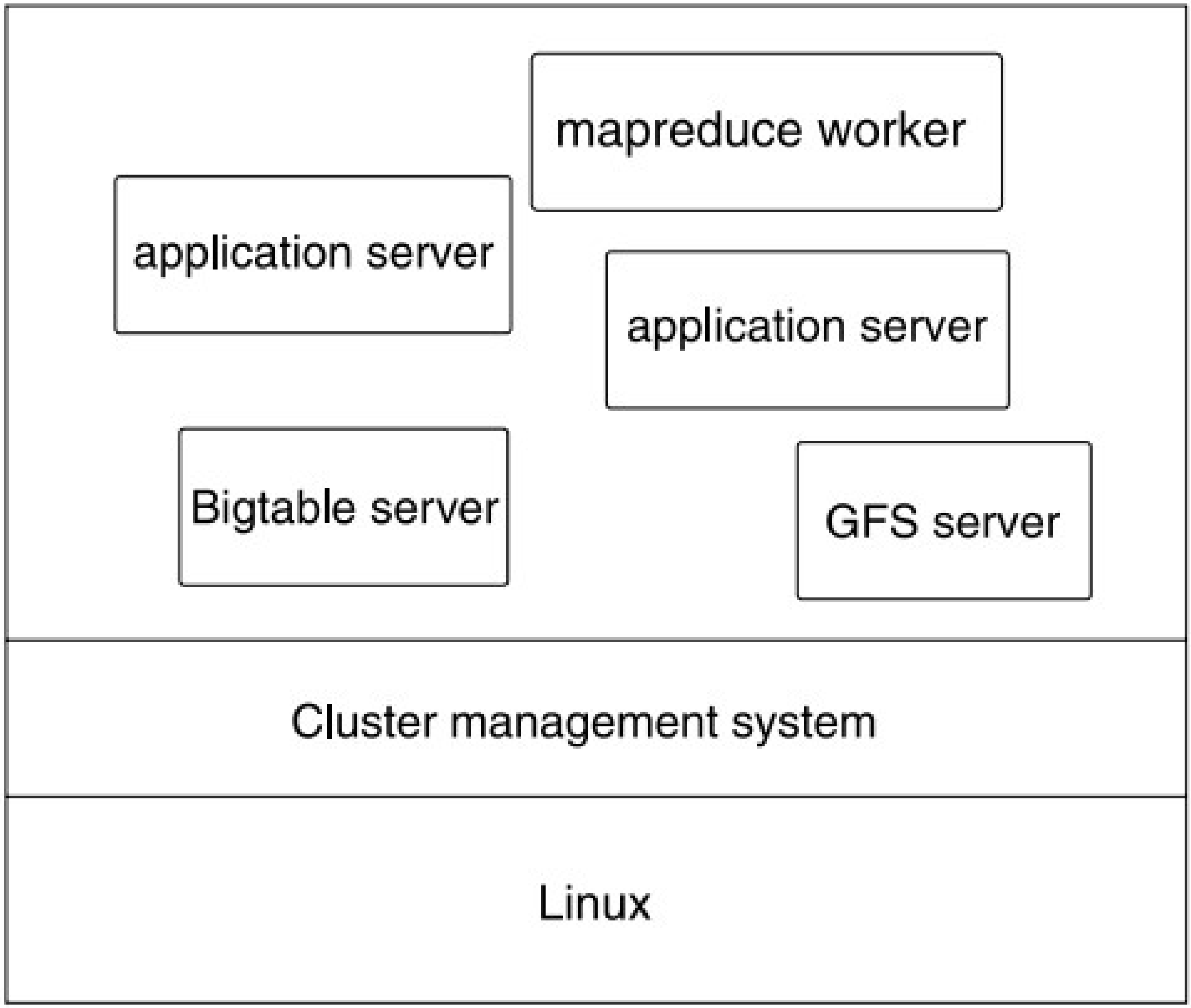
- produce only anchors whose timestamps fall within ten days of the current time



Bigtable API – More details

- Interface for batching writes across row keys at the client
- Execution of client-supplied scripts in server address space (Sawzall)
  - Filtering, summarization, but no writes into Bigtable
- Wrappers written so Bigtable can be an input source and output source for MapReduce jobs

Bigtable – Building Blocks



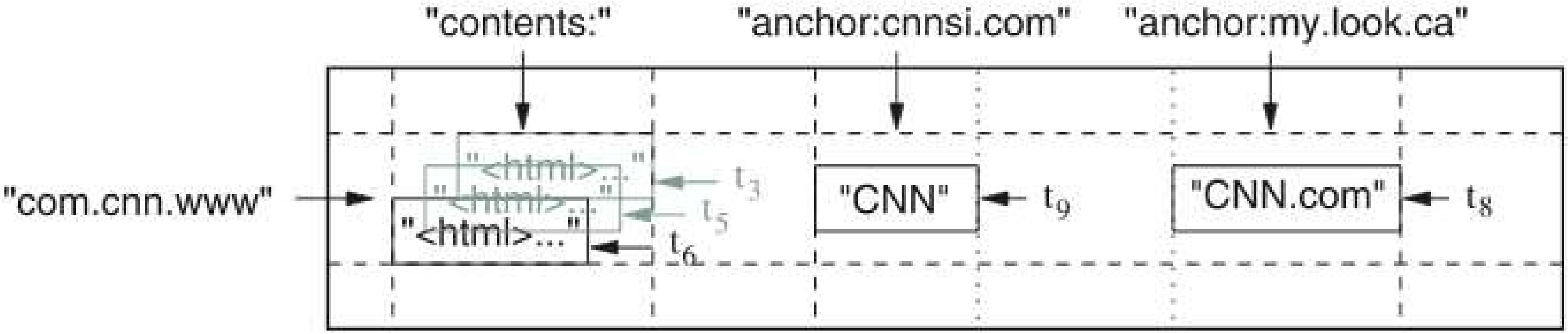
A typical set of processes that run on a Google machine. A machine typically runs many jobs from many different users.

Bigtable – Building Blocks

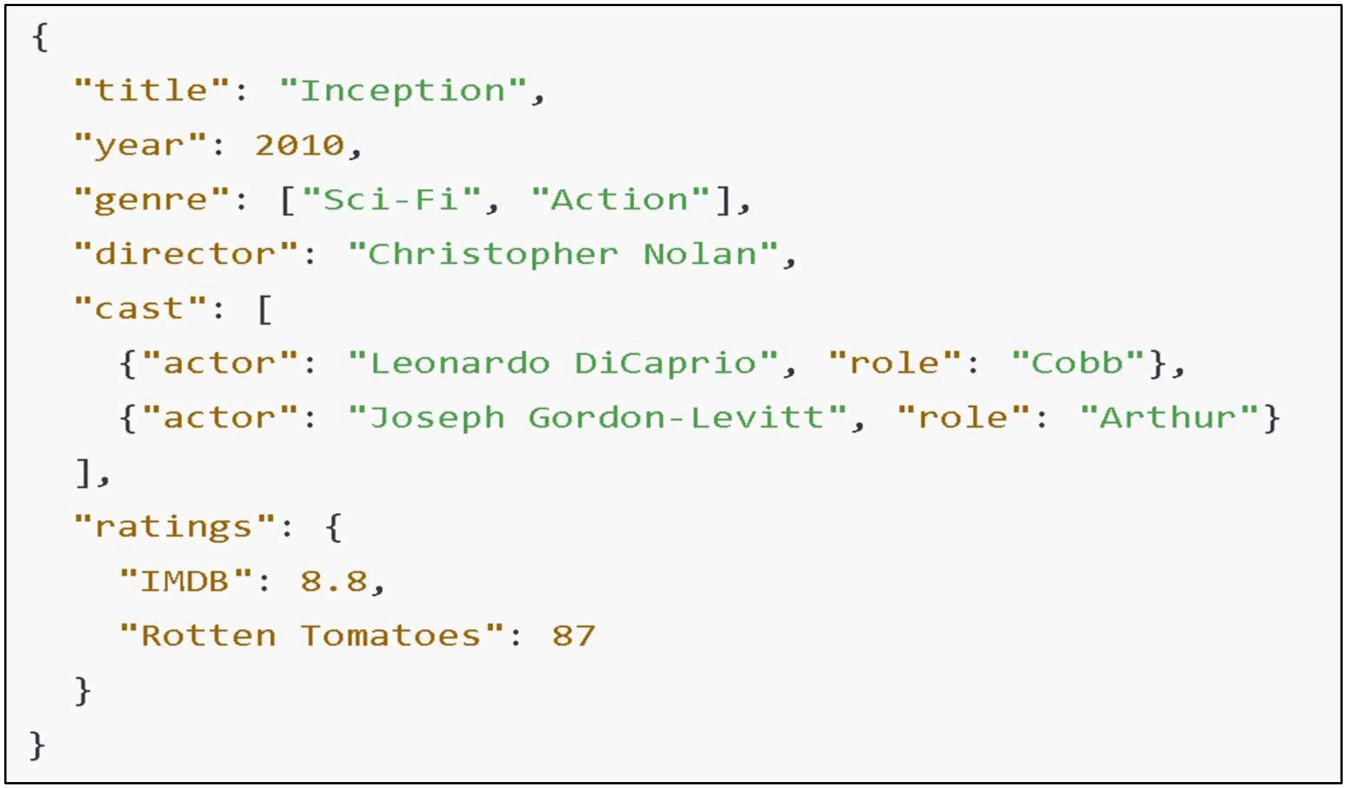
- Shared pool of machines – many applications
- Uses Google cluster management system for scheduling jobs, managing resources, monitoring machine status, dealing with machine failures
- GFS used to store log files and data files
  - Files are replicated with GFS
- SSTable immutable-file format (*Sorted Strings Table*)
  - **Persistent, ordered immutable map** from keys->values
  - Keys and values are arbitrary byte strings
- Operations: lookup key and iterate over key/values in a range
- SSTable contains blocks (64KB in size), block index at end of file
  - Block index always cached in memory – lookup is one disk seek
- SSTable can be memory mapped

Summary: Bigtable – Rows

- Rows – kept ordered by row key
  - Choice of row key important
    - Select row key to get good locality of data access (i.e. reverse hostname)
    - Small row ranges => small # machines
  - Rows with consecutive keys grouped into tablets
  - Partitioning is fixed (in contrast with RDBMS)
  - # rows in a table is unbounded
- **Tablets are unit of distribution and load balancing**
- **Row is unit of transactional consistency**
  - Row read/writes are serializable
  - No transactions across rows



Participation Question 1



```
{
  "title": "Inception",
  "year": 2010,
  "genre": ["Sci-Fi", "Action"],
  "director": "Christopher Nolan",
  "cast": [
    {"actor": "Leonardo DiCaprio", "role": "Cobb"},
    {"actor": "Joseph Gordon-Levitt", "role": "Arthur"}
  ],
  "ratings": {
    "IMDB": 8.8,
    "Rotten Tomatoes": 87
  }
}
```

1. Identify column families for storing this dataset in Bigtable.
2. How would you structure the column keys within those families?

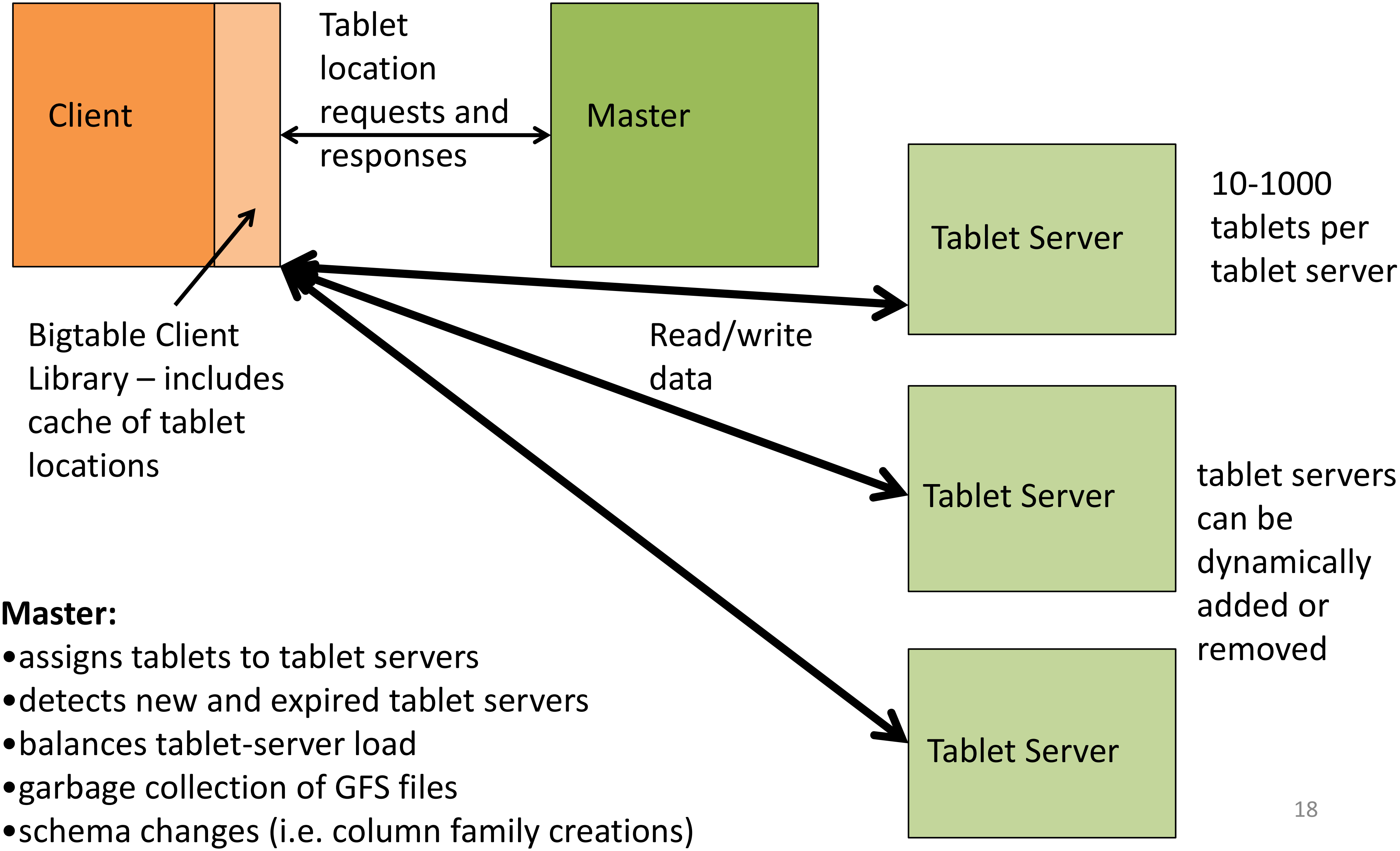
Link to activity:

https://docs.google.com/presentation/d/15b-8y4ZZ0rHcvvXHwNd7BIpN1THVPRa4IIIVIXeH4lU/edit?usp=sharing

Bigtable – Implementation

- Three components
  - Library linked into clients
  - One master server
  - Tablet servers (dynamically removed or added)
- Master Duties:
  - Assigns tablets to Tablet Servers
  - Detects addition & expiration of tablet servers
  - Load Balancing
  - Garbage collection of GFS files
  - Schema changes (table & column family additions and deletions)
- Client
  - **Communicates with Tablet Servers for data**
  - Clients cache Tablet Server location information
  - **Most client calls don’t communicate with the master**

Bigtable – Implementation



Client

Tablet location requests and responses

Master

Tablet Server

10-1000 tablets per tablet server

Bigtable Client Library – includes cache of tablet locations

Read/write data

Tablet Server

tablet servers can be dynamically added or removed

Tablet Server

Master:

- assigns tablets to tablet servers
- detects new and expired tablet servers
- balances tablet-server load
- garbage collection of GFS files
- schema changes (i.e. column family creations)