UNIVERSITY OF WATERLOO

Big Data Infrastructure

CS 489/698 Big Data Infrastructure (Winter 2016)

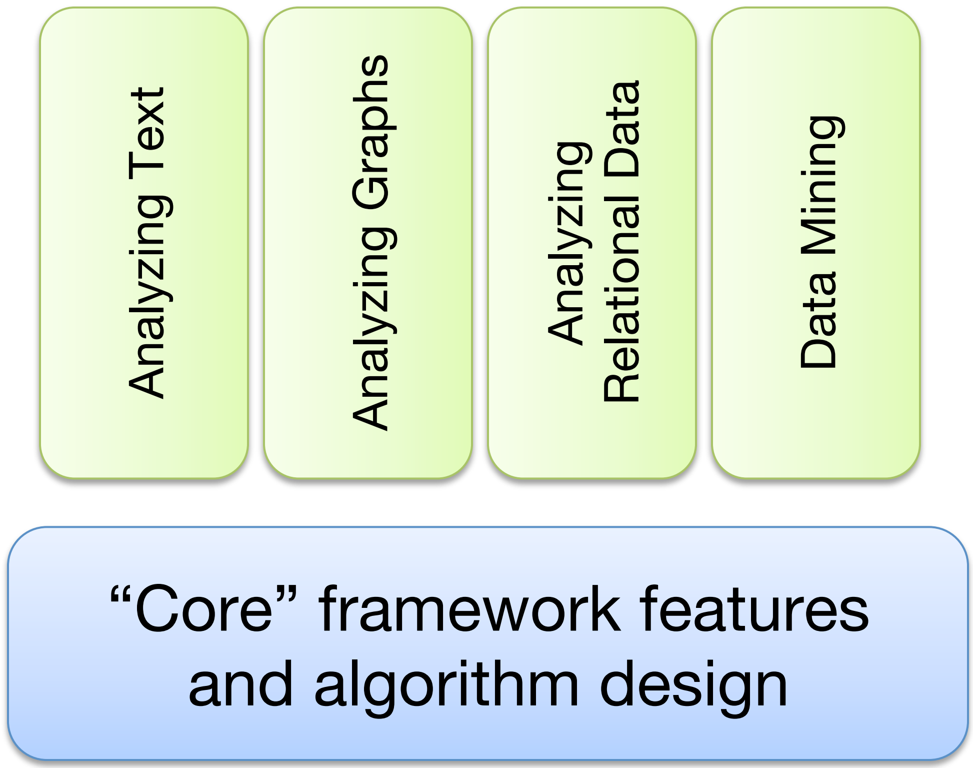
Week 10: Mutable State (1/2)

March 15, 2016

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2016w/

Structure of the Course

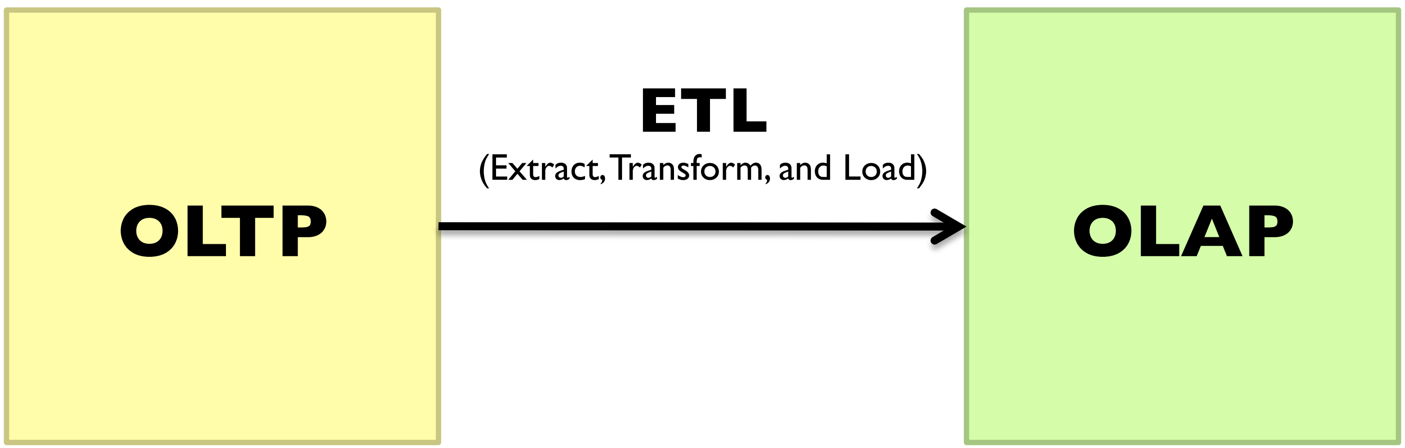


The Fundamental Problem

- We want to keep track of *mutable* state in a *scalable* manner
- Assumptions:
  - State organized in terms of many "records"
  - State unlikely to fit on single machine, must be distributed
- MapReduce won't do!

(note: much of this material belongs in a distributed systems or databases course)

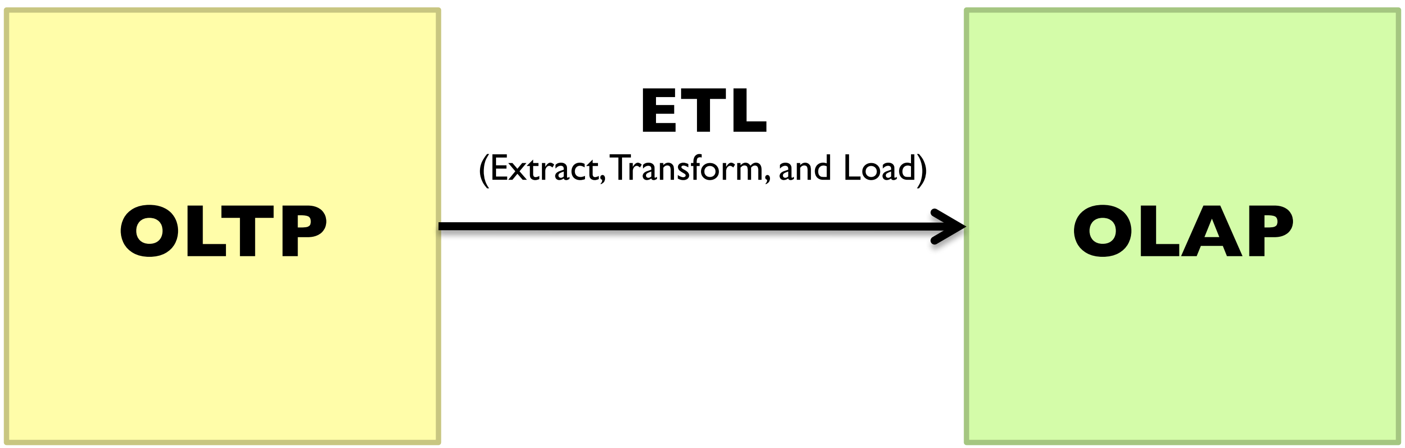
OLTP/OLAP Architecture



Three Core Ideas

- Partitioning (sharding)
  - For scalability
  - For latency
- Replication
  - For robustness (availability)
  - For throughput
- Caching
  - For latency

OLTP/OLAP Architecture



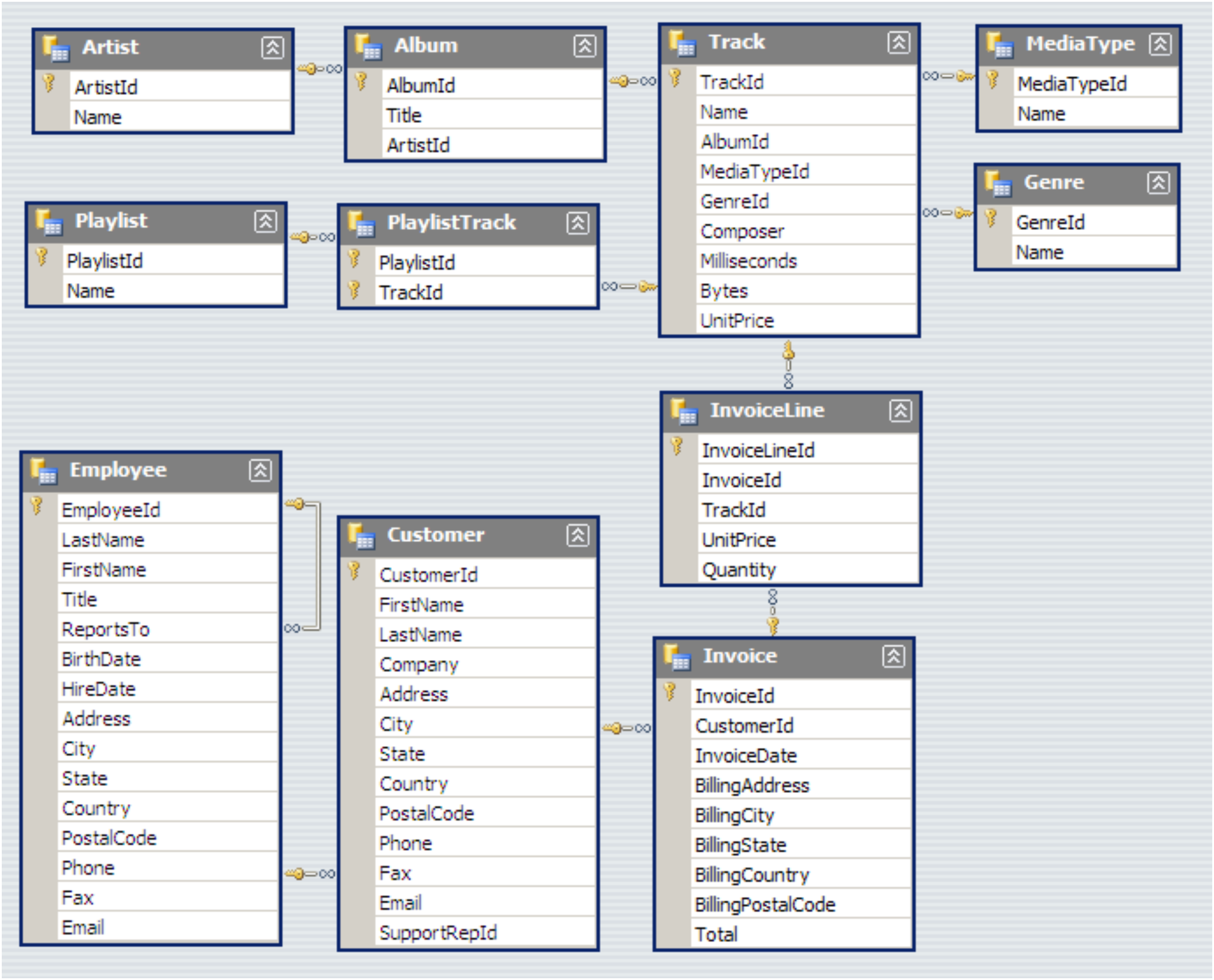
What do RDBMSes provide?

- Relational model with schemas
- Powerful, flexible query language
- Transactional semantics: ACID
- Rich ecosystem, lots of tool support

RDBMSes: Pain Points

Source: www.flickr.com/photos/spencerdahl/6075142688/

#1: Must design up front, painful to evolve



Note: Flexible design doesn't mean *no* design!

Remember the camelSnake!

This should really be a list…

```
{
  "token": 945842,
  "feature_enabled": "super_special",
  "userid": 229922,
  "page": "null",
  "info": { "email": "my@place.com" }
}
```

Is this really an integer?

Is this really null?

What keys? What values?

JSON to the Rescue!

Flexible design doesn’t mean no design!

#2: Pay for ACID!

Source: Wikipedia (Tortoise)

#3: Cost!

Source: www.flickr.com/photos/gnusinn/3080378658/

What do RDBMSes provide?

- Relational model with schemas
- Powerful, flexible query language
- Transactional semantics: ACID
- Rich ecosystem, lots of tool support

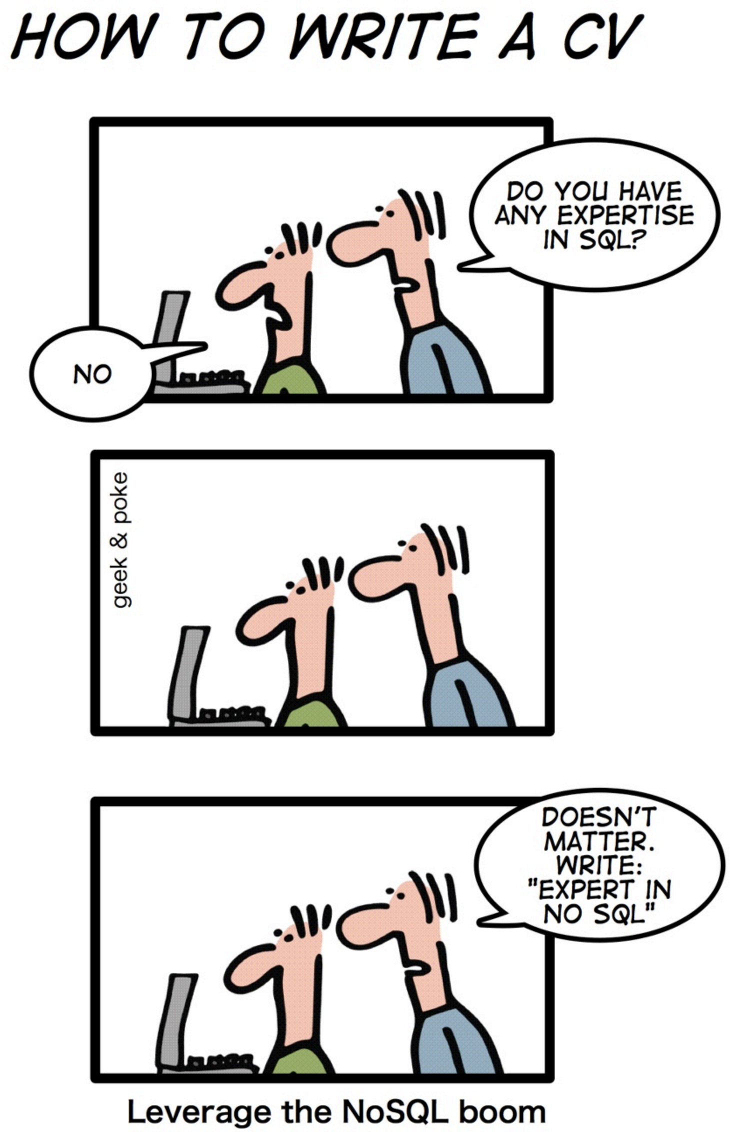
What if we want *a la carte*?

Source: www.flickr.com/photos/vidiot/18556565/

Features *a la carte*?

- What if I'm willing to give up consistency for scalability?
- What if I'm willing to give up the relational model for something more flexible?
- What if I just want a cheaper solution?

Enter… NoSQL!



Source: geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

NoSQL (Not only SQL)

1. Horizontally scale "simple operations"
2. Replicate/distribute data over many servers
3. Simple call interface
4. Weaker concurrency model than ACID
5. Efficient use of distributed indexes and RAM
6. Flexible schemas

But, don't blindly follow the hype…
Often, (sharded) MySQL is what you really need!

Source: Cattell (2010). Scalable SQL and NoSQL Data Stores. *SIGMOD Record*.

(Major) Types of NoSQL databases

- Key-value stores
- Column-oriented databases
- Document stores
- Graph databases

Key-Value Stores

Source: Wikipedia (Keychain)

Key-Value Stores: Data Model

- Stores associations between keys and values
- Keys are usually primitives
  - For example, ints, strings, raw bytes, etc.
- Values can be primitive or complex: usually opaque to store
  - Primitives: ints, strings, etc.
  - Complex: JSON, HTML fragments, etc.

Key-Value Stores: Operations

- Very simple API:
  - Get – fetch value associated with key
  - Put – set value associated with key
- Optional operations:
  - Multi-get
  - Multi-put
  - Range queries
- Consistency model:
  - Atomic puts (usually)
  - Cross-key operations: who knows?

Key-Value Stores: Implementation

- Non-persistent:
  - Just a big in-memory hash table
- Persistent
  - Wrapper around a traditional RDBMS

What if data doesn't fit on a single machine?

Simple Solution: Partition!

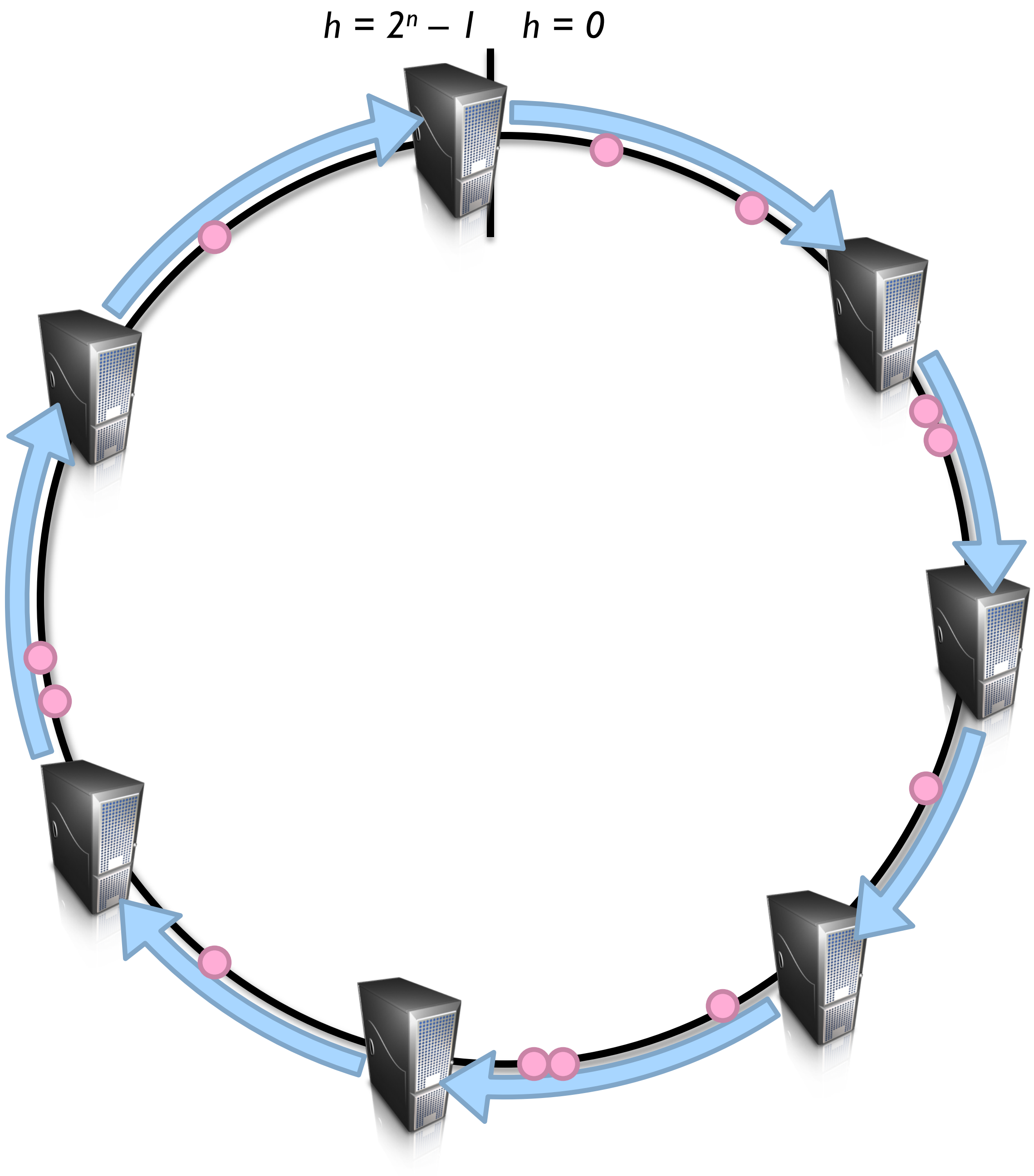
- Partition the key space across multiple machines
  - Let's say, hash partitioning
  - For *n* machines, store key *k* at machine *h(k)* mod *n*
- Okay… But:
  1. How do we know which physical machine to contact?
  2. How do we add a new machine to the cluster?
  3. What happens if a machine fails?

See the problems here?

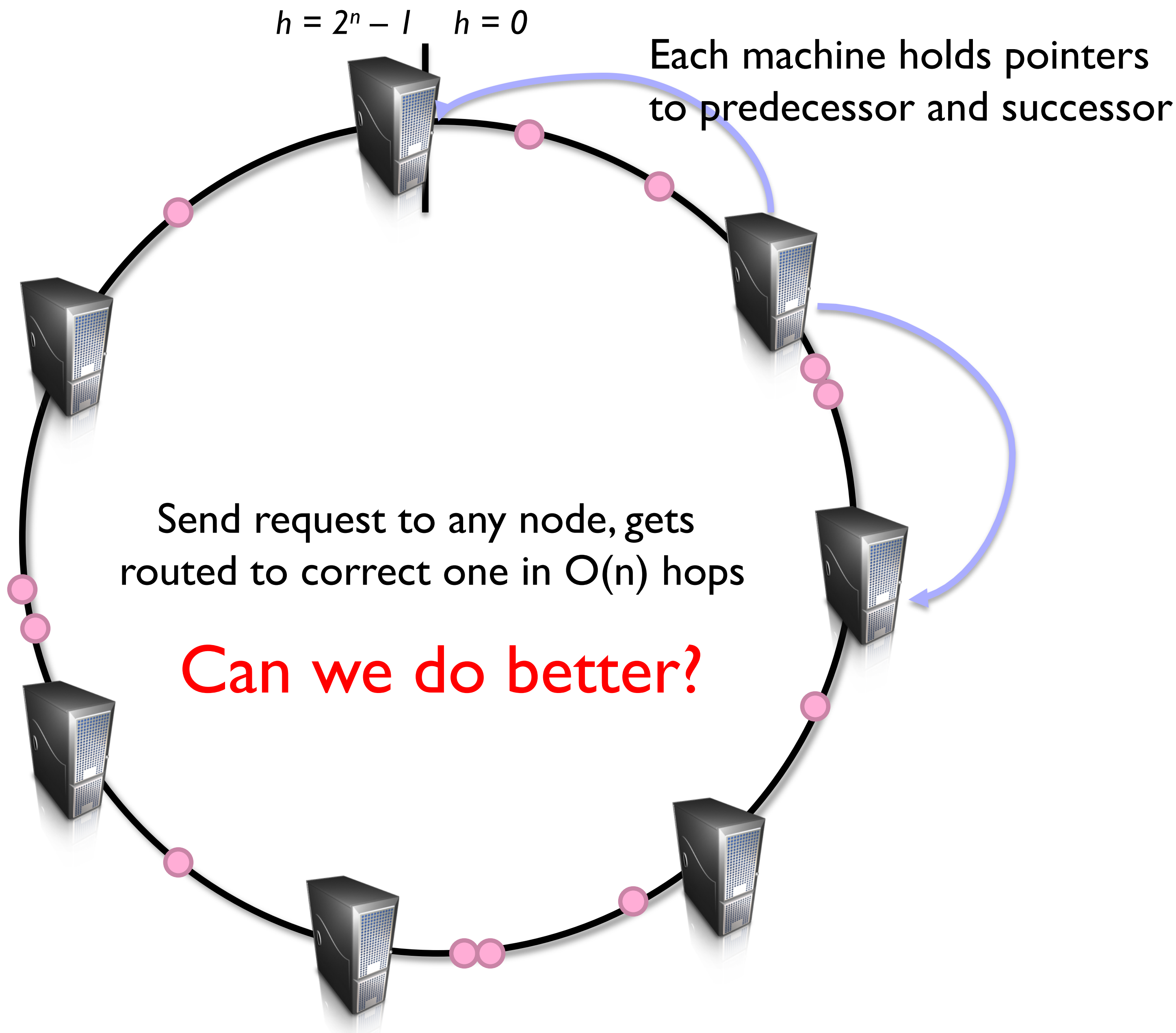
Clever Solution

- Hash the keys
- Hash the machines also!

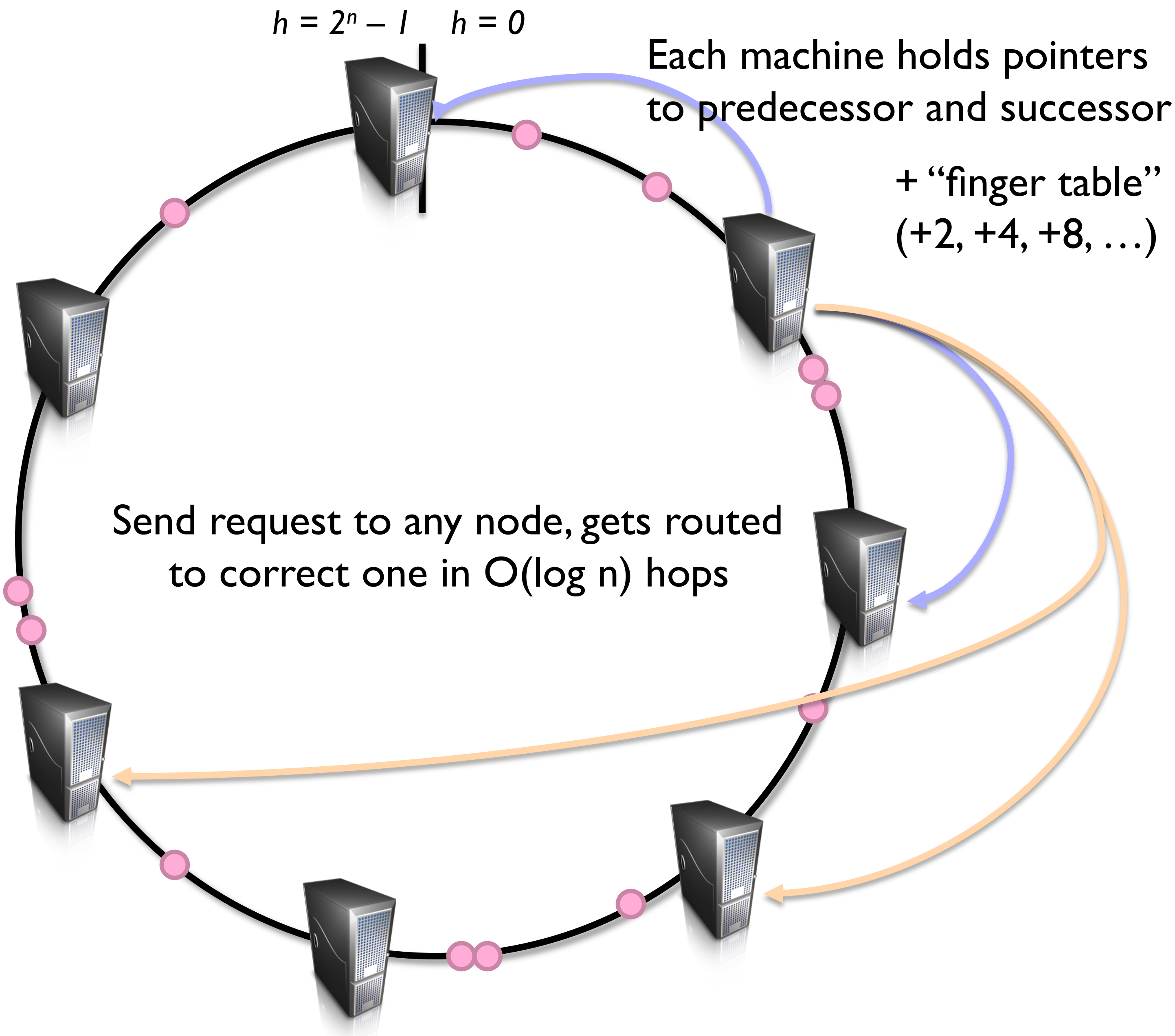
Distributed hash tables!
(following combines ideas from several sources…)



$h = 2^n - 1$
$h = 0$



Routing: Which machine holds the key?



Routing: Which machine holds the key?

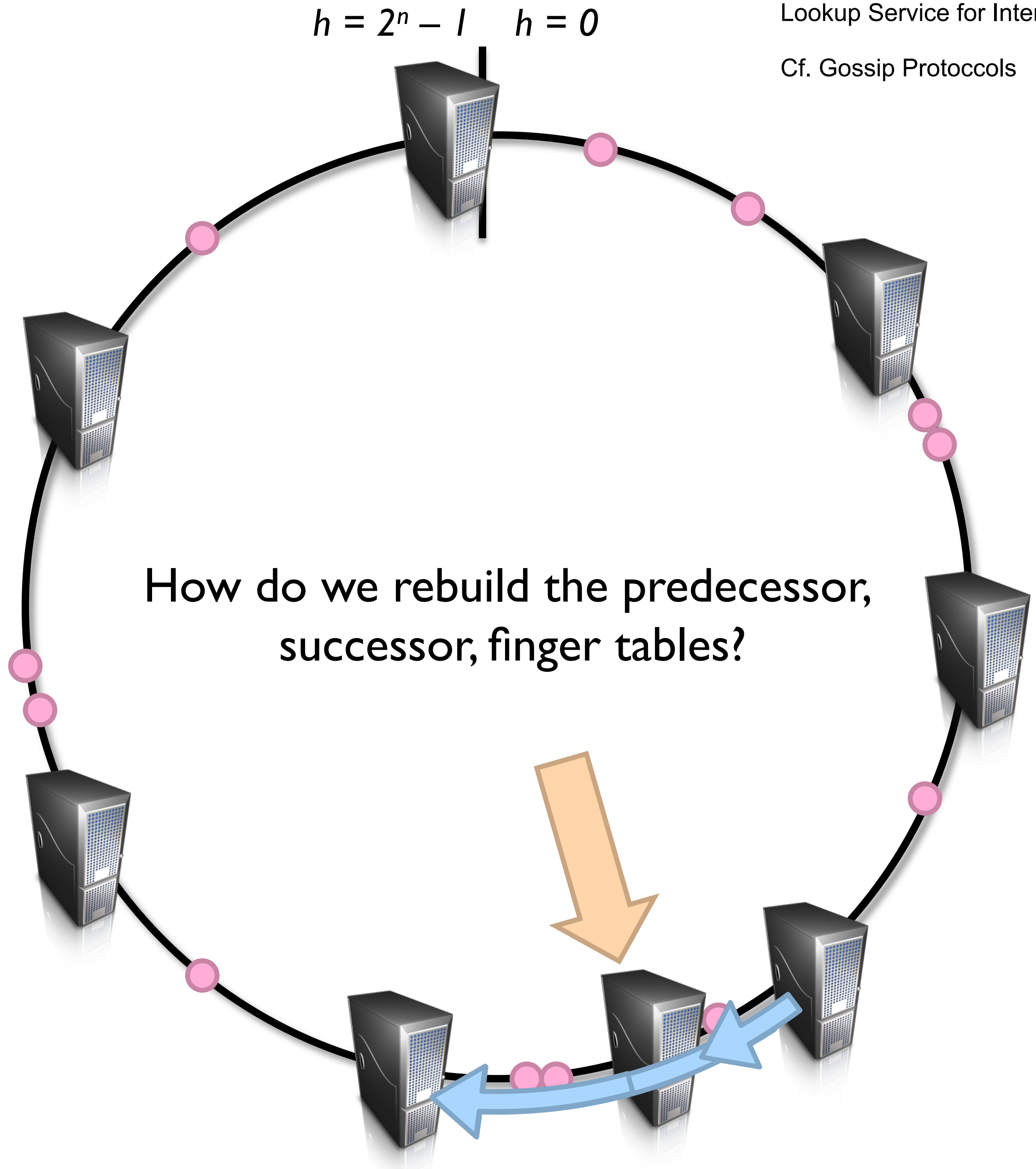
Simpler Solution

Service Registry

Routing: Which machine holds the key?

Stoica et al. (2001). Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *SIGCOMM.*

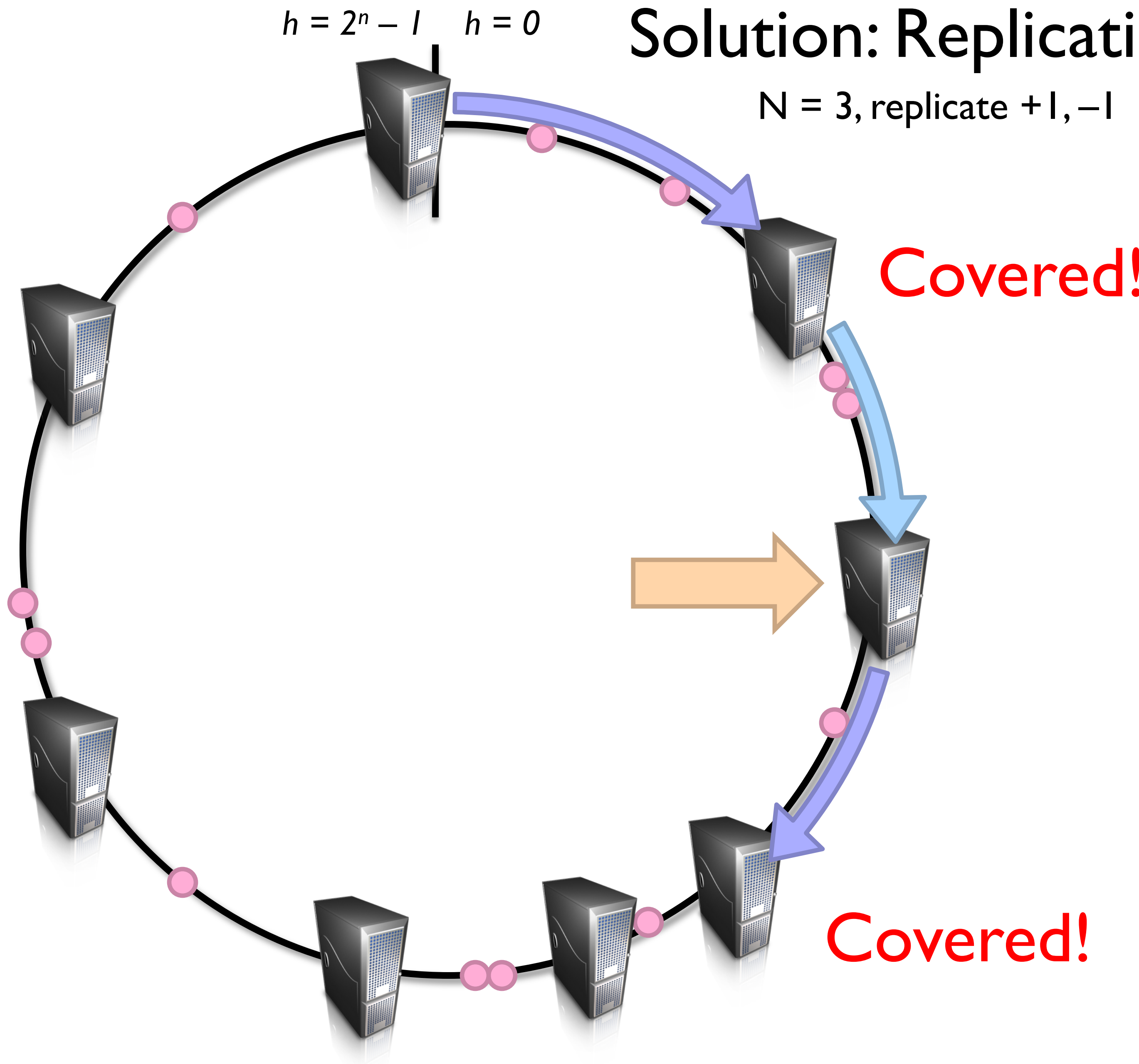
Cf. Gossip Protoccols



New machine joins: What happens?

Solution: Replication

N = 3, replicate +1, –1



Machine fails: What happens?

Another Refinement: Virtual Nodes

- Don’t directly hash servers
- Create a large number of virtual nodes, map to physical servers
  - Better load redistribution in event of machine failure
  - When new server joins, evenly shed load from other servers

Bigtable

Source: Wikipedia (Table)

Bigtable Applications

- Gmail
- Google's web crawl
- Google Earth
- Google Analytics
- Data source and data sink for MapReduce

HBase is the open-source implementation…

Data Model

- A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map
- Map indexed by a row key, column key, and a timestamp
  - (row:string, column:string, time:int64) → uninterpreted byte array
- Supports lookups, inserts, deletes
  - Single row transactions only

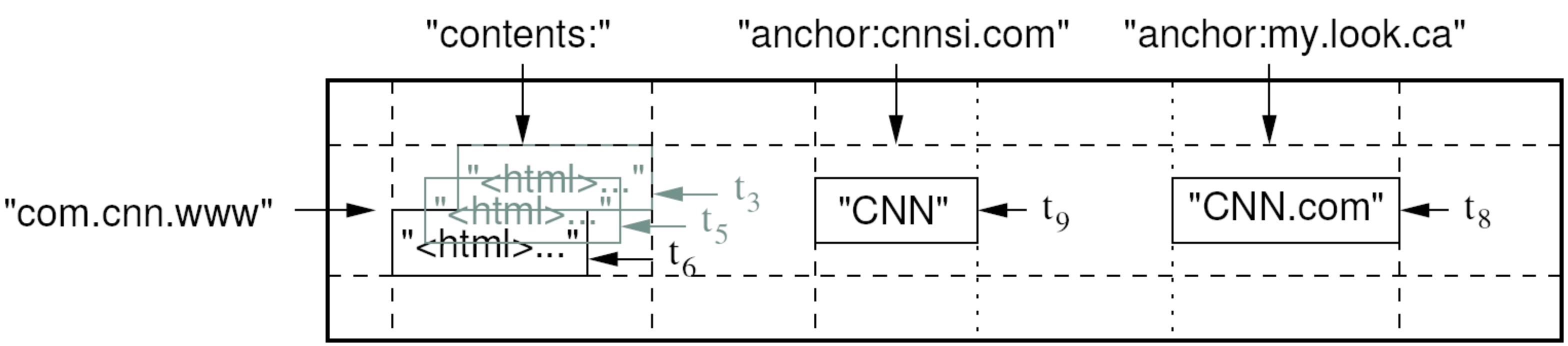


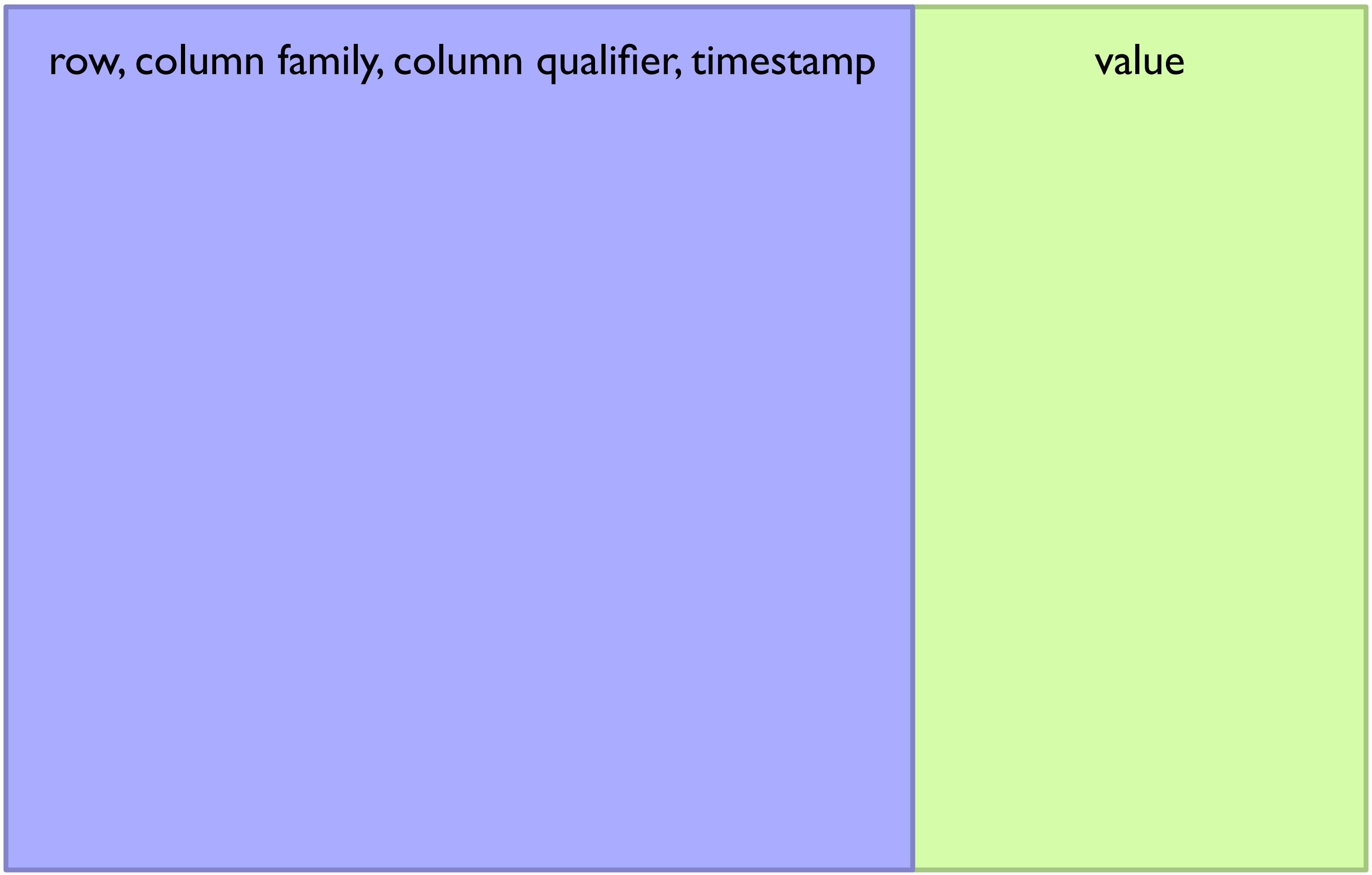
Image Source: Chang et al., OSDI 2006

Rows and Columns

- Rows maintained in sorted lexicographic order
  - Applications can exploit this property for efficient row scans
  - Row ranges dynamically partitioned into tablets
- Columns grouped into column families
  - Column key = *family:qualifier*
  - Column families provide locality hints
  - Unbounded number of columns

At the end of the day, it's all key-value pairs!

Key-Values



Okay, so how do we build it?

In Memory	On Disk
Mutability Easy	Mutability Hard
Small	Big

HBase
Bigtable Building Blocks

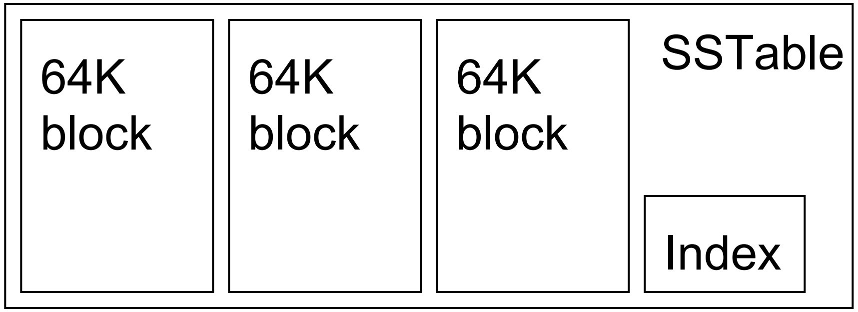
- GFS HDFS
- Chubby Zookeeper
- SSTable HFile

SSTable HFile

- Basic building block of Bigtable
- Persistent, ordered immutable map from keys to values
  - Stored in GFS

We get replication for free!

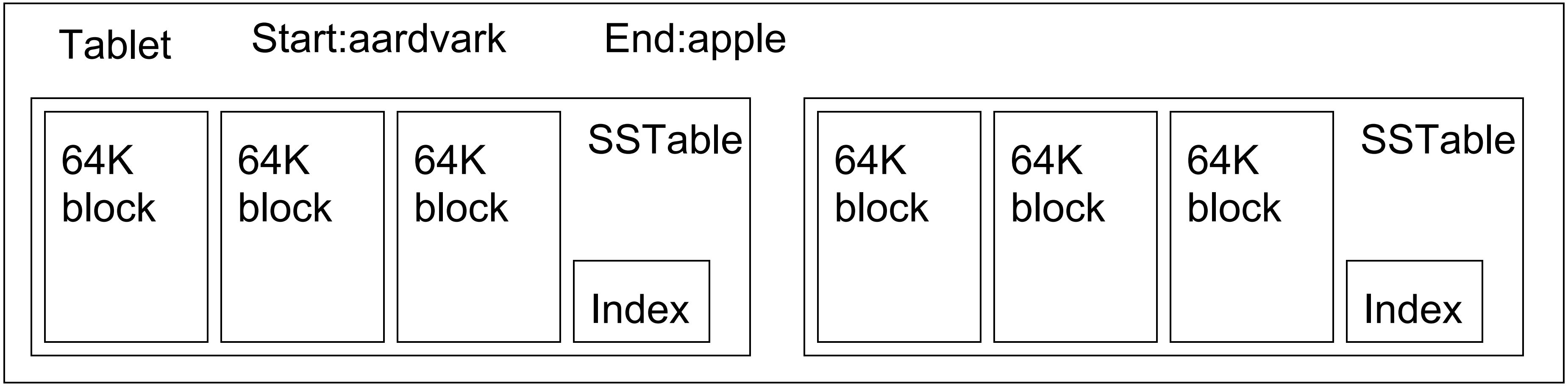
- Sequence of blocks on disk plus an index for block lookup
  - Can be completely mapped into memory
- Supported operations:
  - Look up value associated with key
  - Iterate key/value pairs within a key range



Source: Graphic from slides by Erik Paulson

Tablet Region

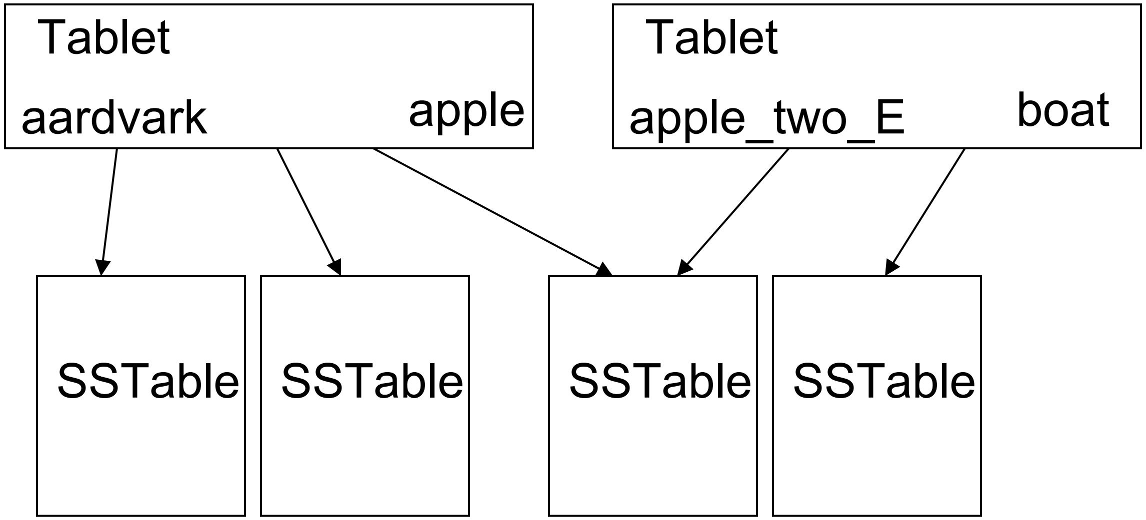
- Dynamically partitioned range of rows
- Built from multiple SSTables



Source: Graphic from slides by Erik Paulson

Table

- Multiple tablets make up the table
- SSTables can be shared



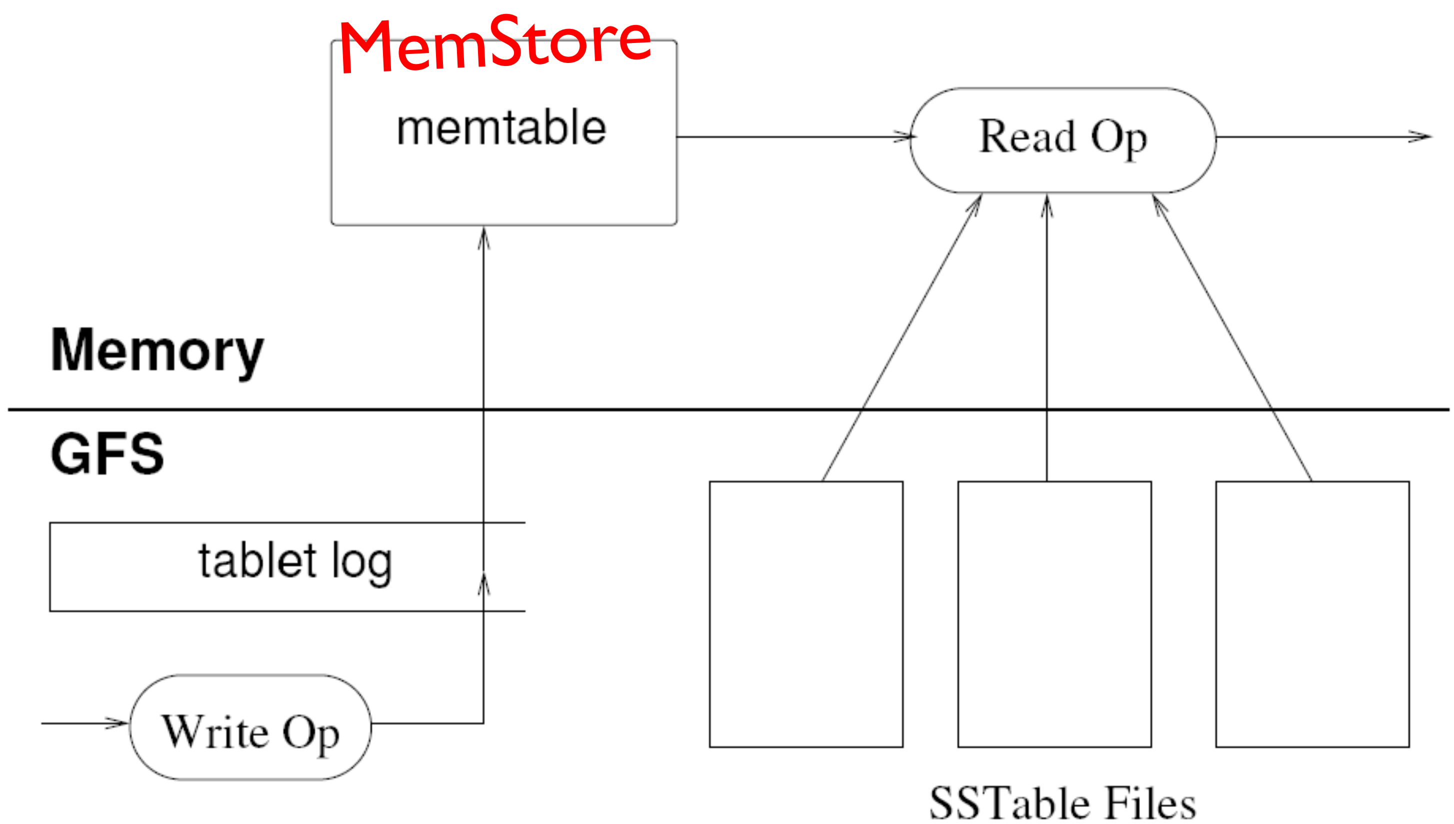
Source: Graphic from slides by Erik Paulson

How do I get mutability?

Easy, keep everything in memory!

What happens when I run out of memory?

Tablet Serving



"Log Structured Merge Trees"

Image Source: Chang et al., OSDI 2006

Architecture

- Client library
- Single master server HMaster
- Tablet servers RegionServers

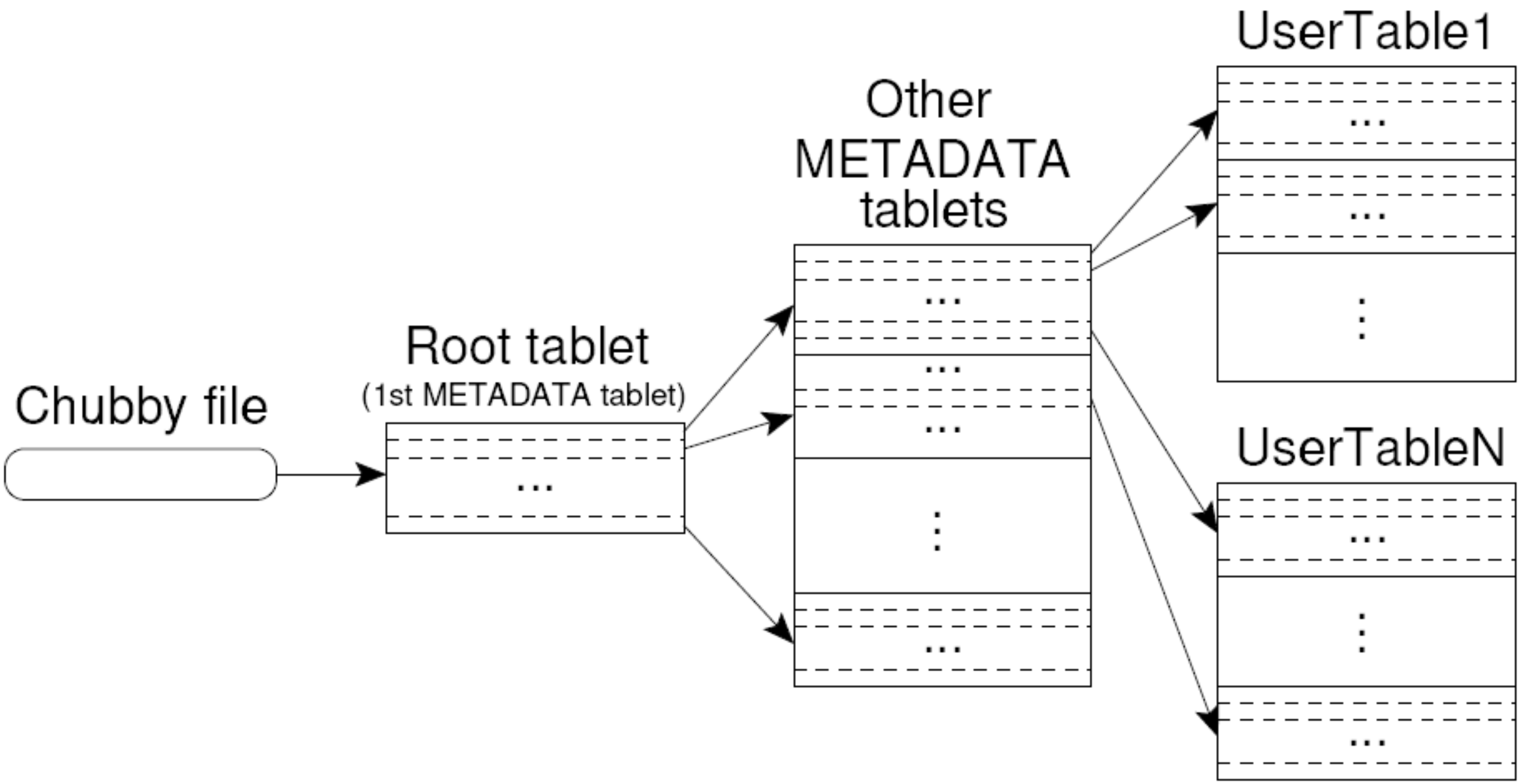
Bigtable Master

- Assigns tablets to tablet servers
- Detects addition and expiration of tablet servers
- Balances tablet server load
- Handles garbage collection
- Handles schema changes

Bigtable Tablet Servers

- Each tablet server manages a set of tablets
  - Typically between ten to a thousand tablets
  - Each 100-200 MB by default
- Handles read and write requests to the tablets
- Splits tablets that have grown too large

Tablet Location



Upon discovery, clients cache tablet locations

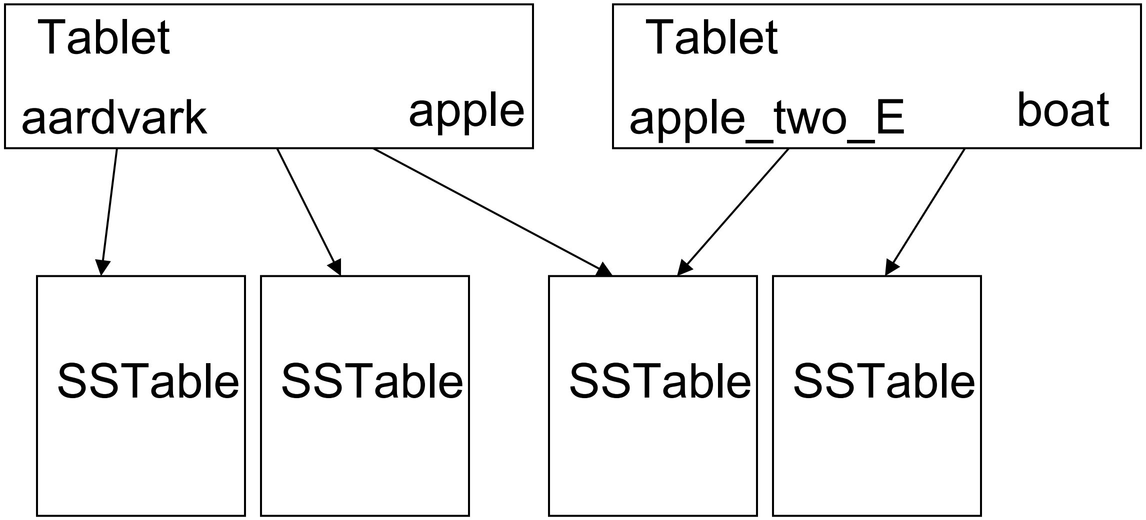
Image Source: Chang et al., OSDI 2006

Tablet Assignment

- Master keeps track of:
  - Set of live tablet servers
  - Assignment of tablets to tablet servers
  - Unassigned tablets
- Each tablet is assigned to one tablet server at a time
  - Tablet server maintains an exclusive lock on a file in Chubby
  - Master monitors tablet servers and handles assignment
- Changes to tablet structure
  - Table creation/deletion (master initiated)
  - Tablet merging (master initiated)
  - Tablet splitting (tablet server initiated)

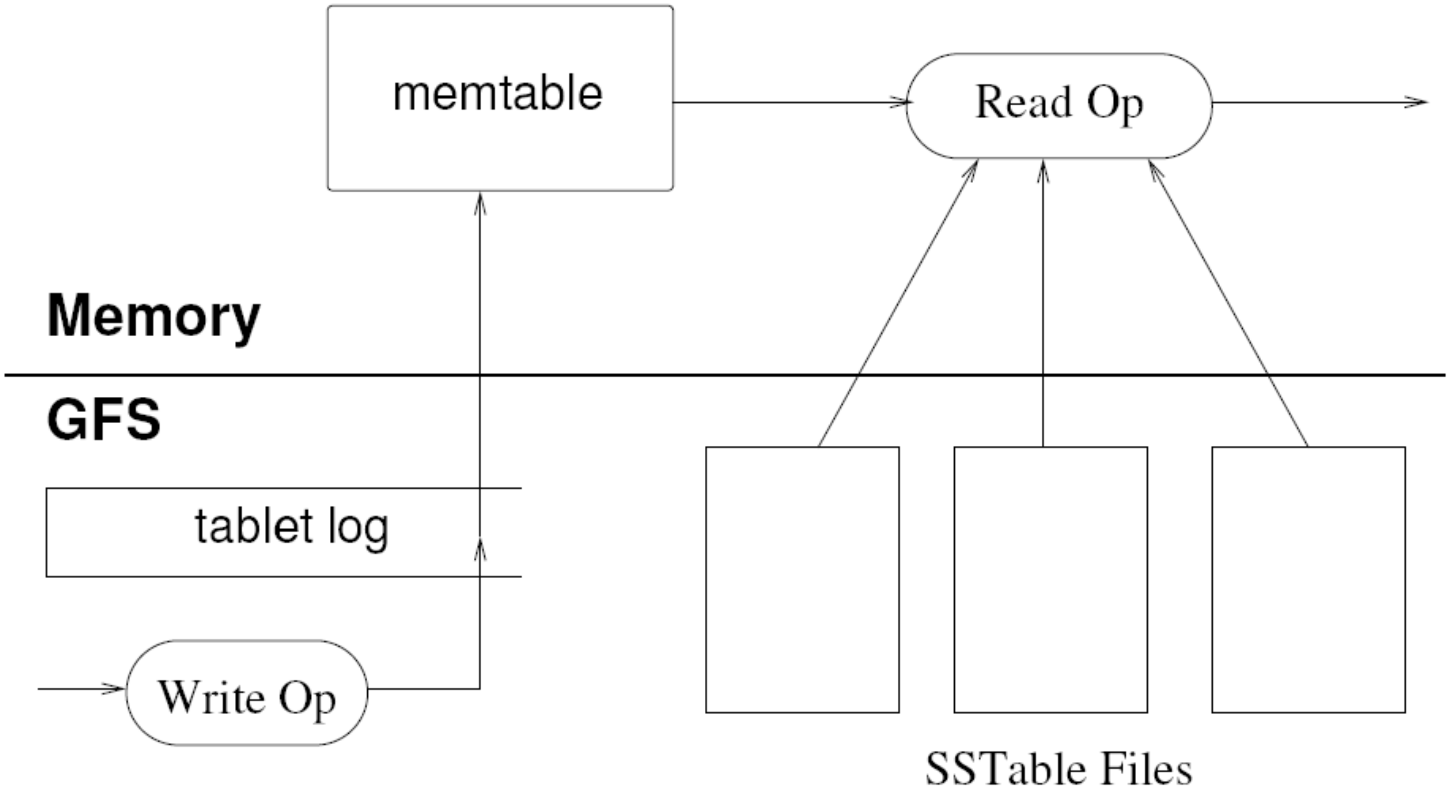
Table

- Multiple tablets make up the table
- SSTables can be shared



Source: Graphic from slides by Erik Paulson

Tablet Serving



"Log Structured Merge Trees"

Image Source: Chang et al., OSDI 2006

Compactions

- Minor compaction
  - Converts the memtable into an SSTable
  - Reduces memory usage and log traffic on restart
- Merging compaction
  - Reads the contents of a few SSTables and the memtable, and writes out a new SSTable
  - Reduces number of SSTables
- Major compaction
  - Merging compaction that results in only one SSTable
  - No deletion records, only live data

HBase

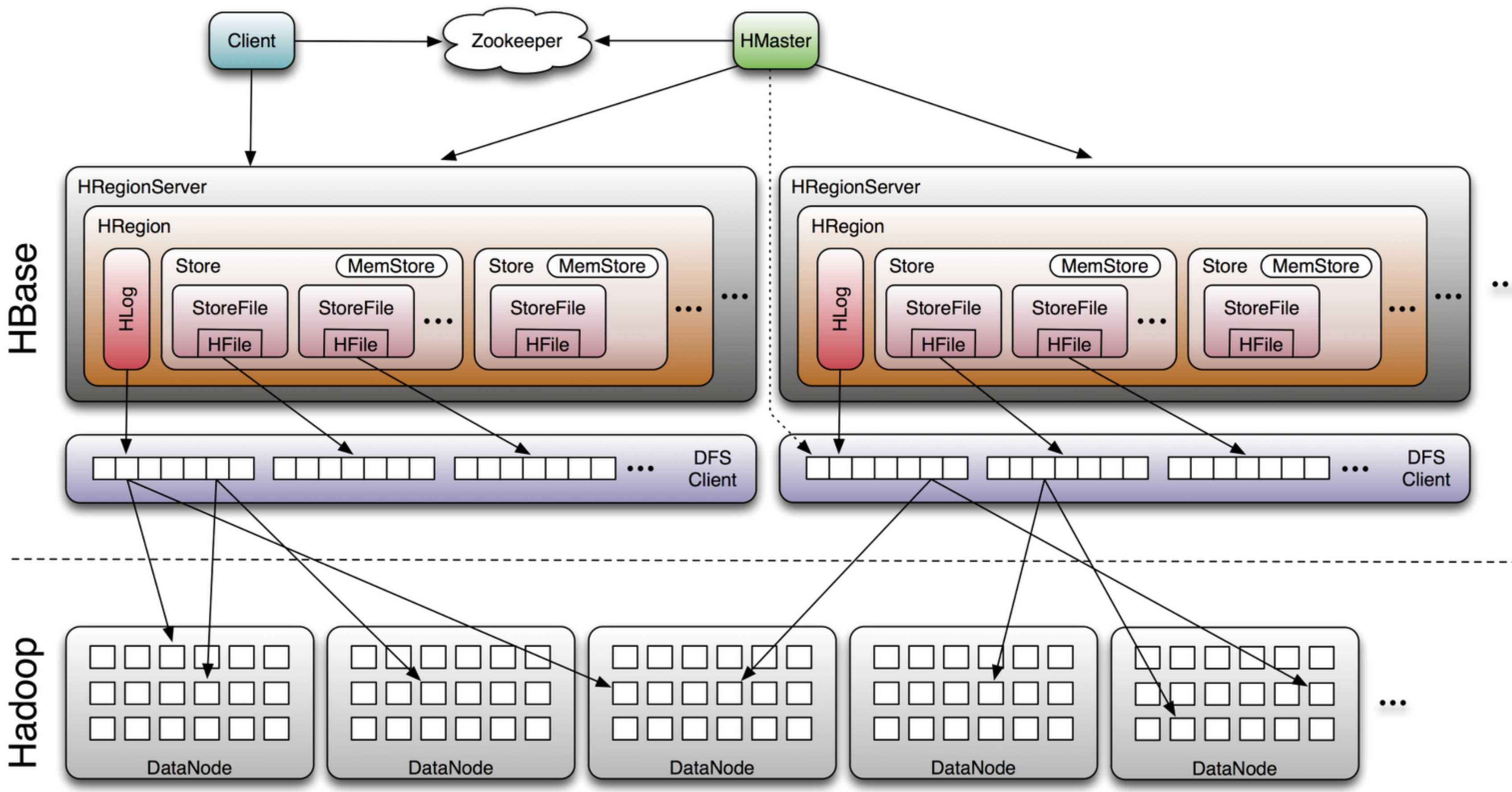


Image Source: http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html

Source: Wikipedia (Cake)


Questions?
Source: Wikipedia (Japanese rock garden)