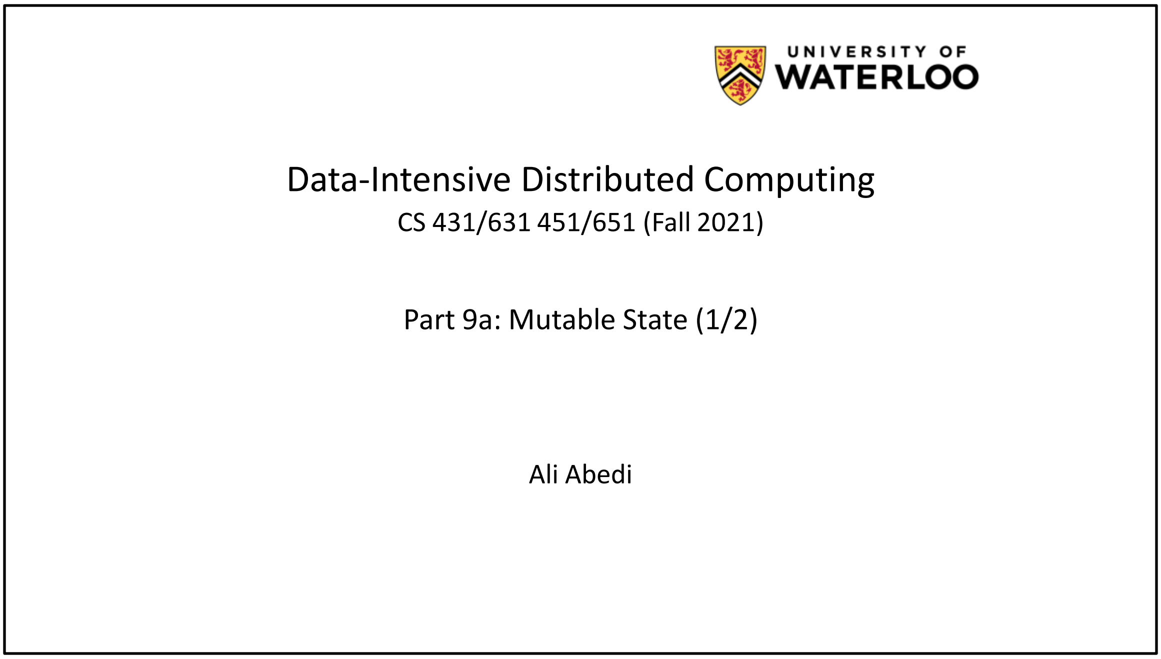


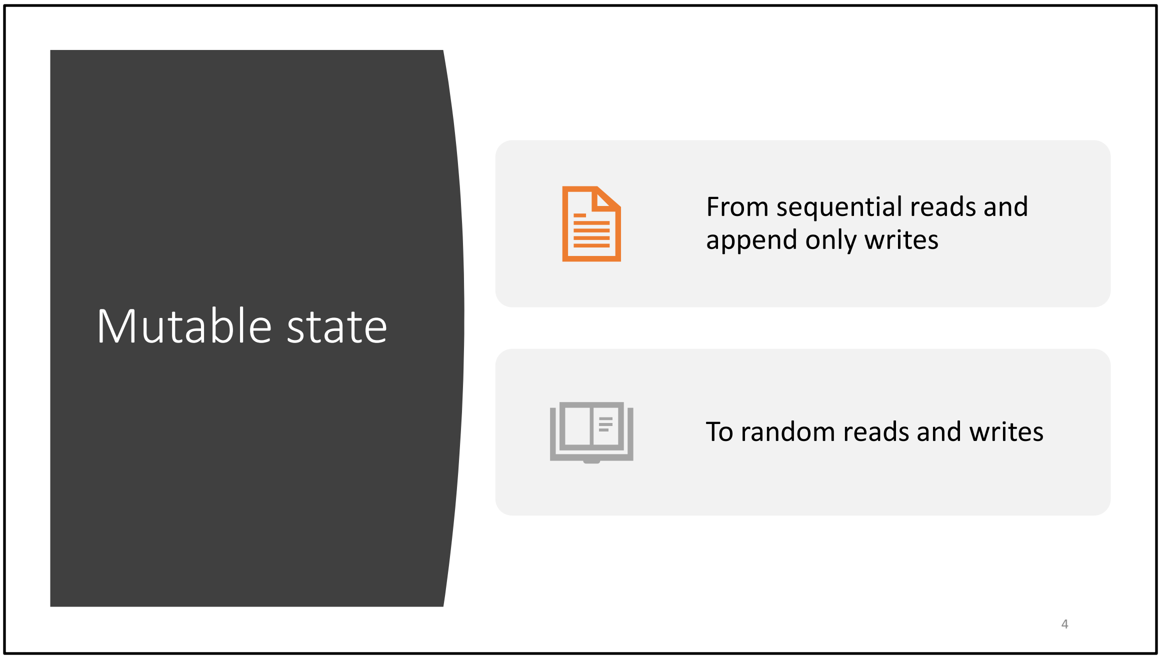
UNIVERSITY OF WATERLOO

Data-Intensive Distributed Computing

CS 431/631 451/651 (Fall 2021)

Part 9a: Mutable State (1/2)

Ali Abedi

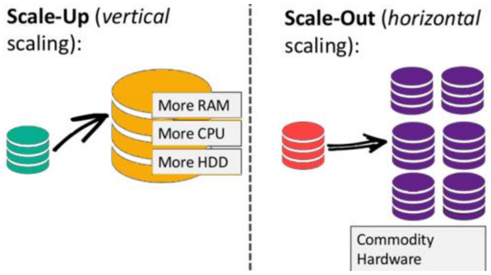


Mutable state

- From sequential reads and append only writes
- To random reads and writes

Why not RDBMS?

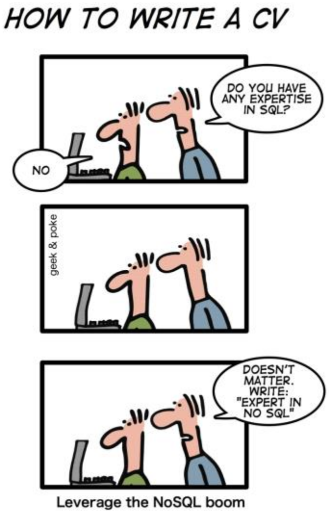
- Does not scale out → expensive



- Does not support semi-structured data

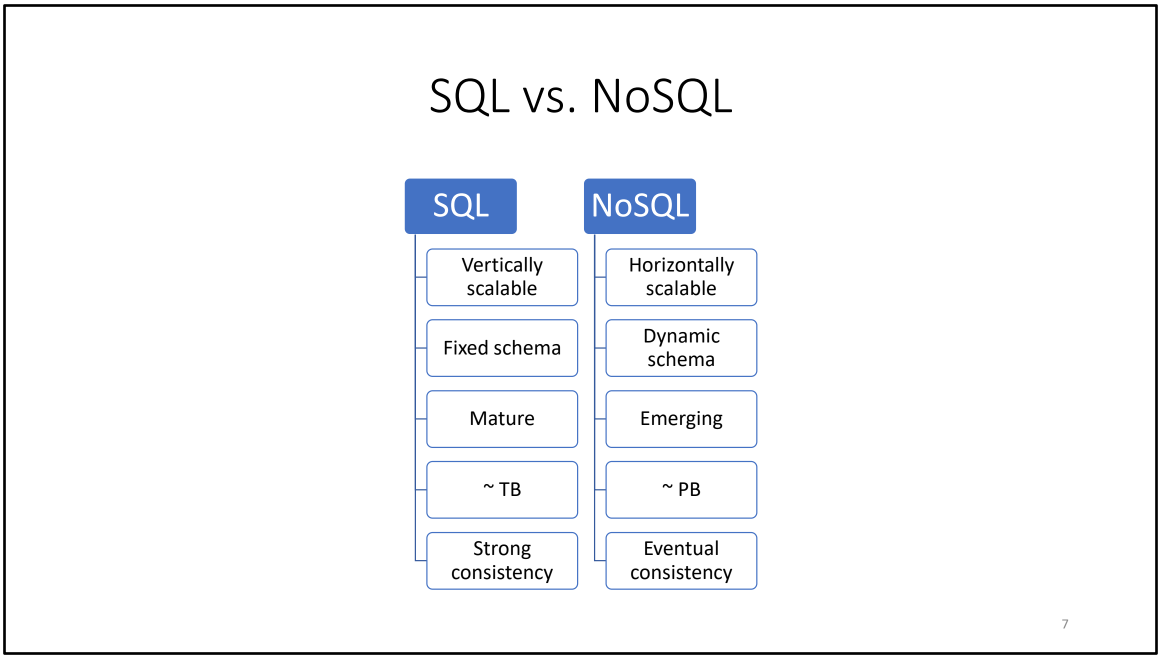
NoSQL
(Not only SQL)

1. Horizontal scaling
2. Replicate/distribute data over many servers
3. Simple call interface
4. Weaker concurrency model than ACID
5. Flexible schemas


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

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

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

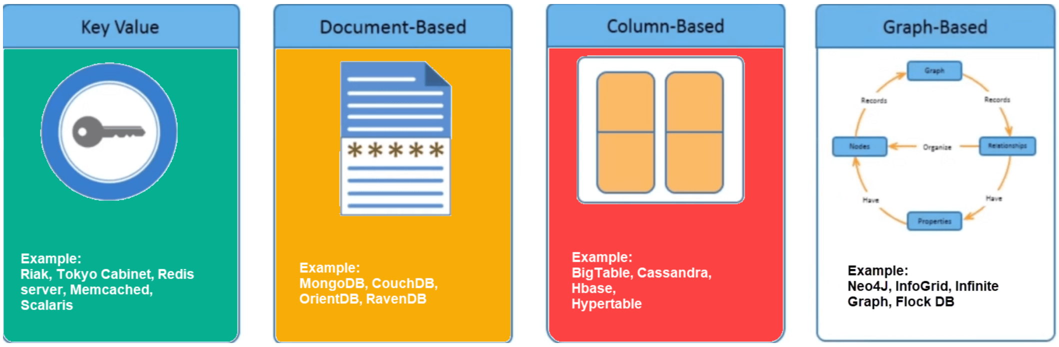


SQL vs. NoSQL

SQL	NoSQL
Vertically scalable	Horizontally scalable
Fixed schema	Dynamic schema
Mature	Emerging
~ TB	~ PB
Strong consistency	Eventual consistency

7

(Major) Types of NoSQL databases



Key Value

Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris

Document-Based

Example:
MongoDB, CouchDB, OrientDB, RavenDB

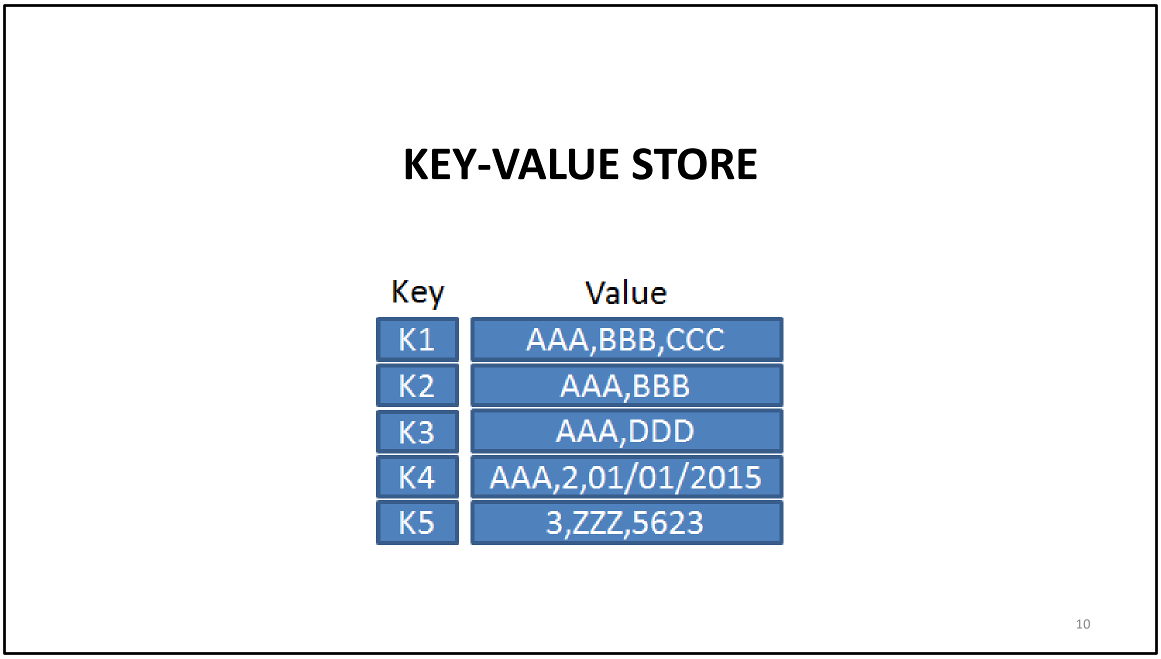
Column-Based

Example:
BigTable, Cassandra, Hbase, Hypertable

Graph-Based

Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

Source: guru99.com/nosql-tutorial.html



KEY-VALUE STORE

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Three Core Ideas

Keeping track of the partitions?

Partitioning (sharding)

To increase scalability and to decrease latency

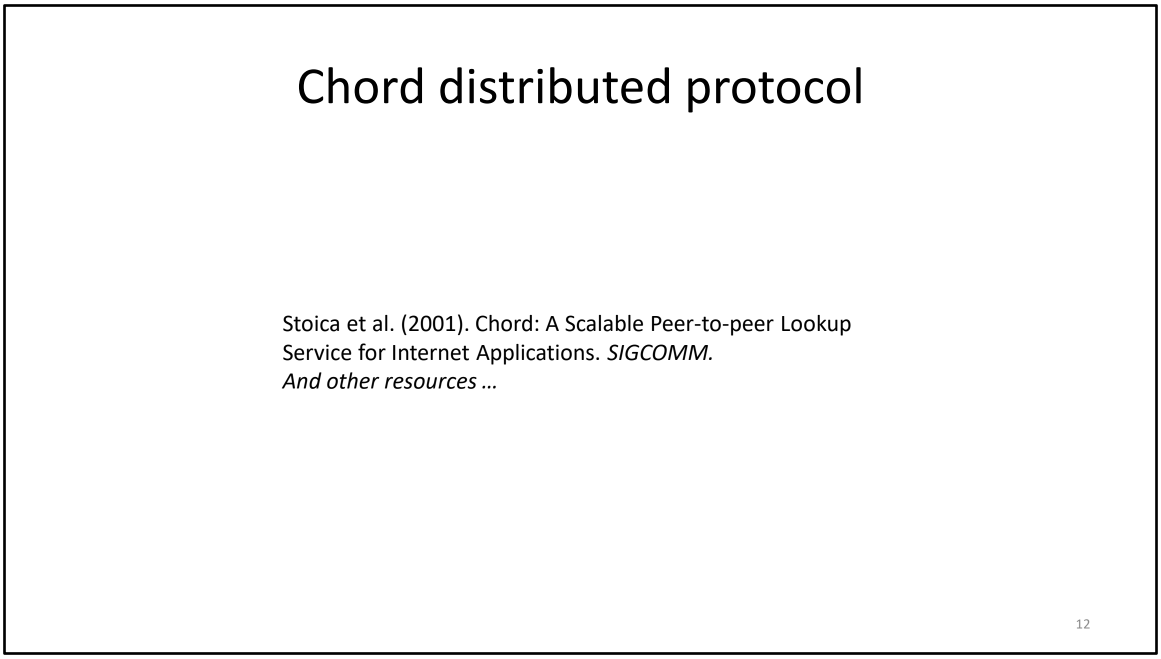
Consistency?

Replication

To increase robustness (availability) and to increase throughput

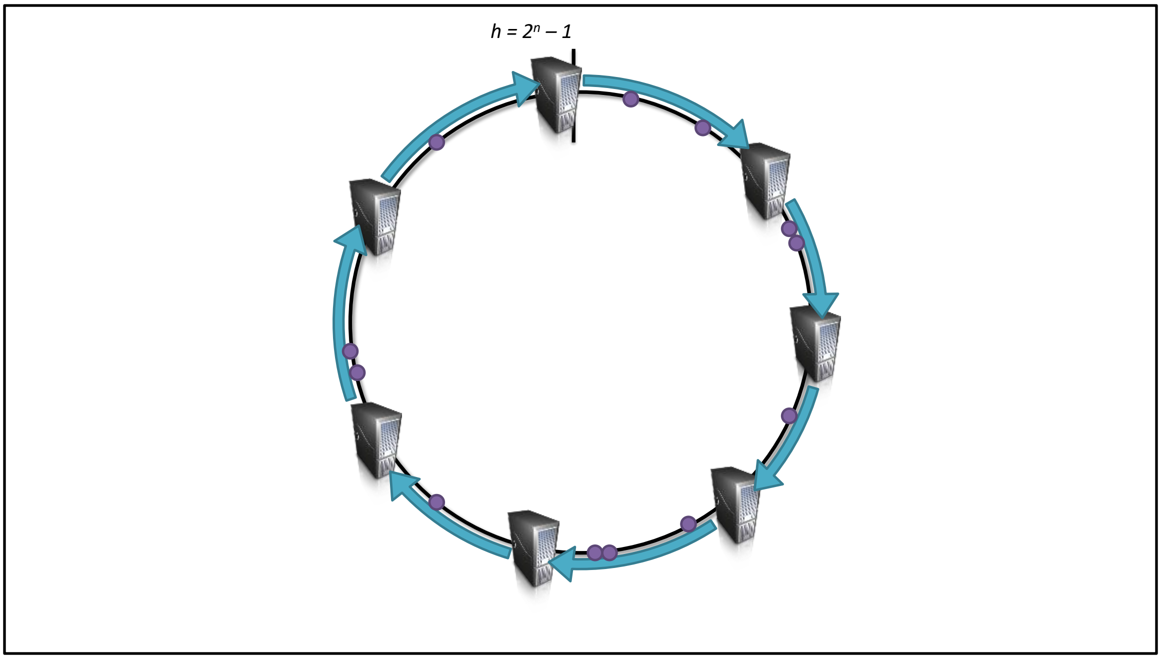
Caching

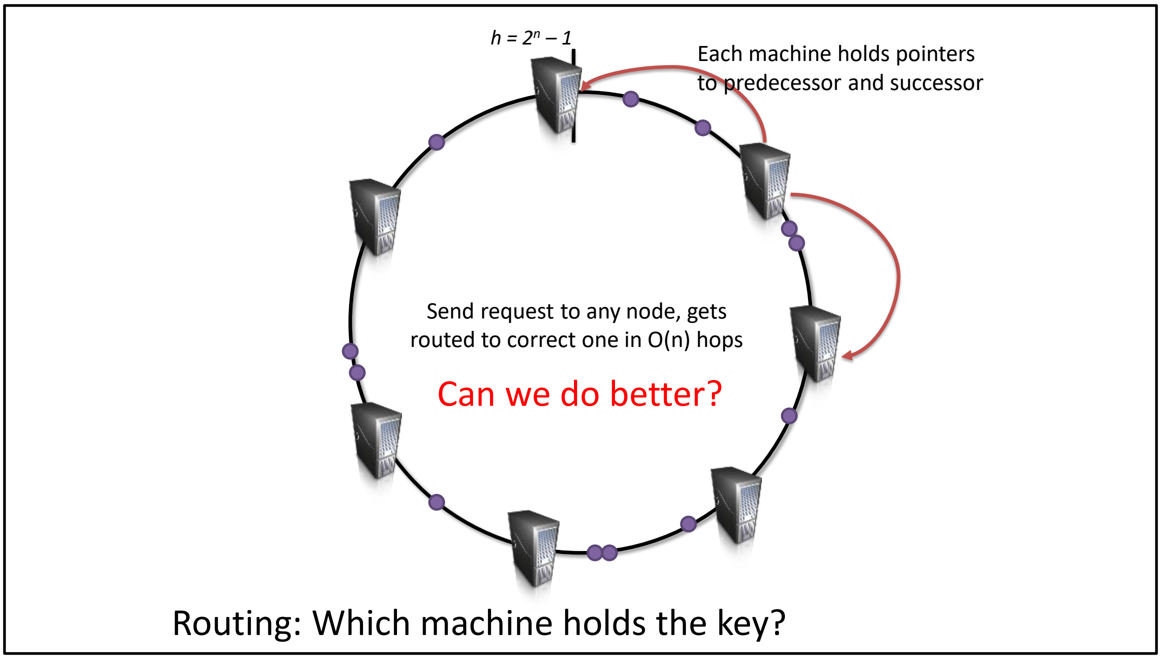
To reduce latency



Chord distributed protocol

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


$h = 2^n - 1$

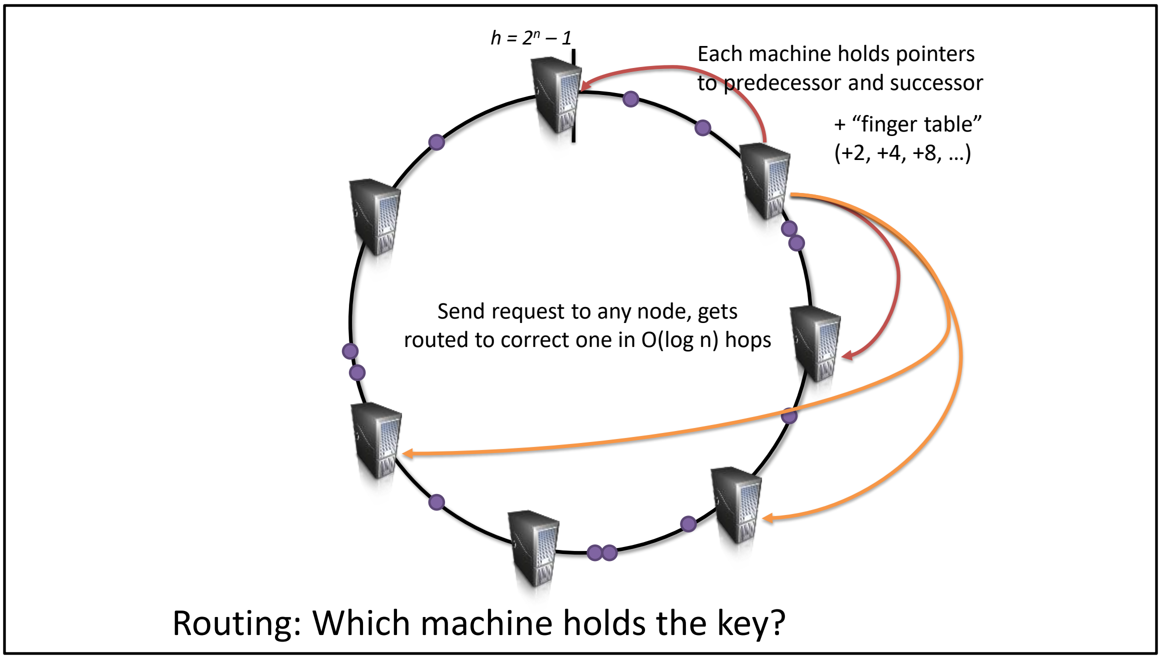


Each machine holds pointers to predecessor and successor

Send request to any node, gets routed to correct one in O(n) hops

Can we do better?

Routing: Which machine holds the key?



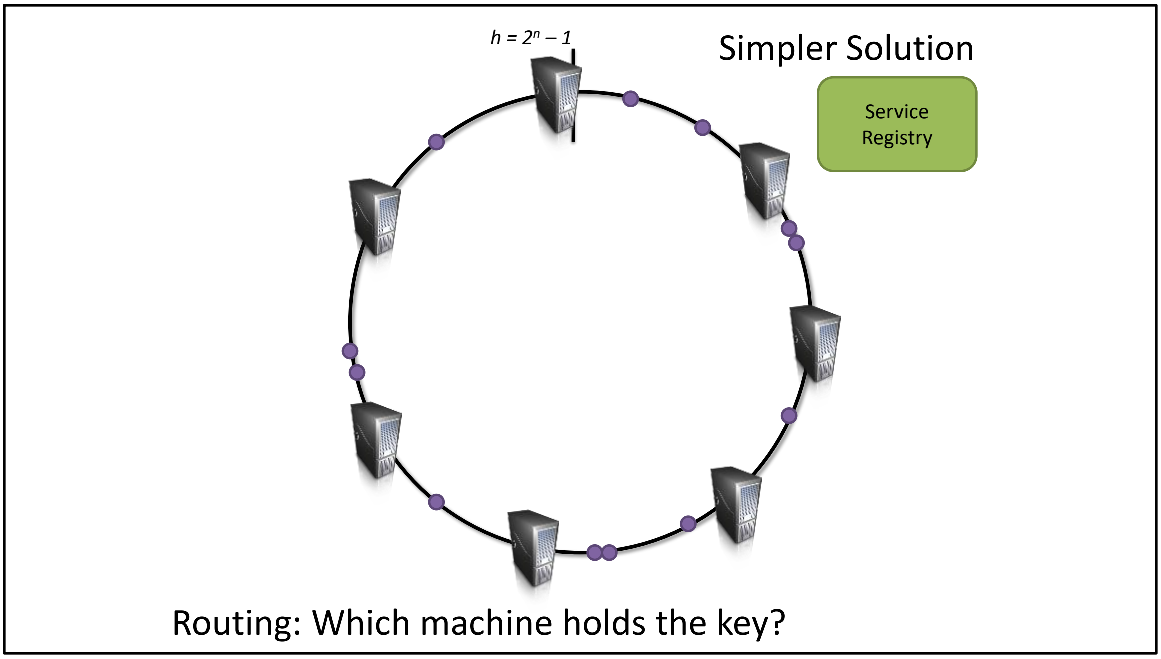
$h = 2^n - 1$

Each machine holds pointers to predecessor and successor

+ “finger table” (+2, +4, +8, …)

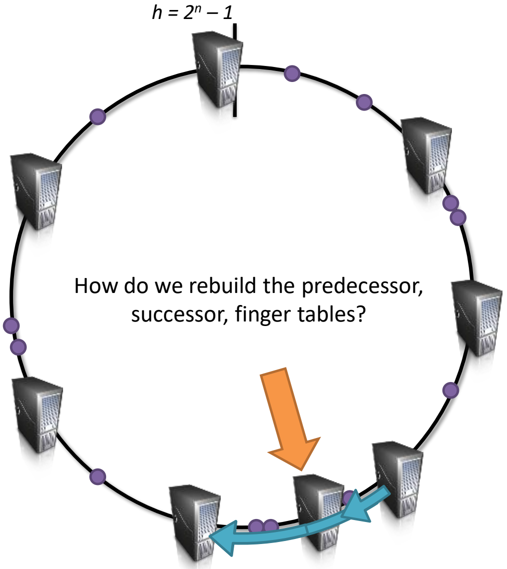
Send request to any node, gets routed to correct one in O(log n) hops

Routing: Which machine holds the key?



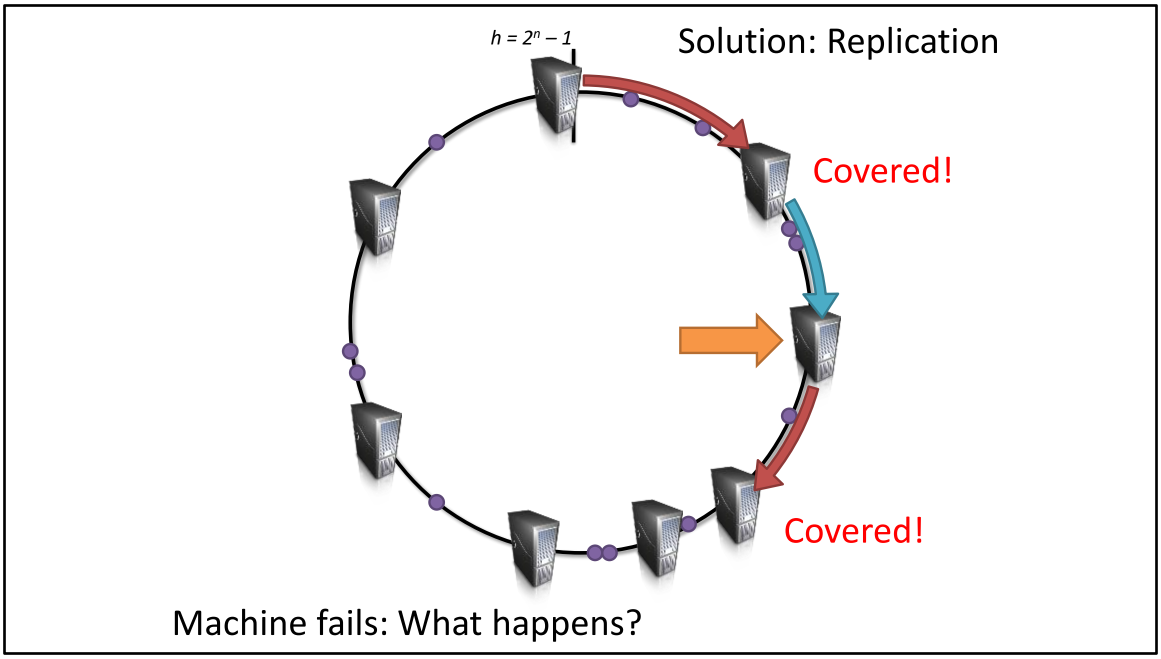
Simpler Solution

Routing: Which machine holds the key?



How do we rebuild the predecessor, successor, finger tables?

New machine joins: What happens?

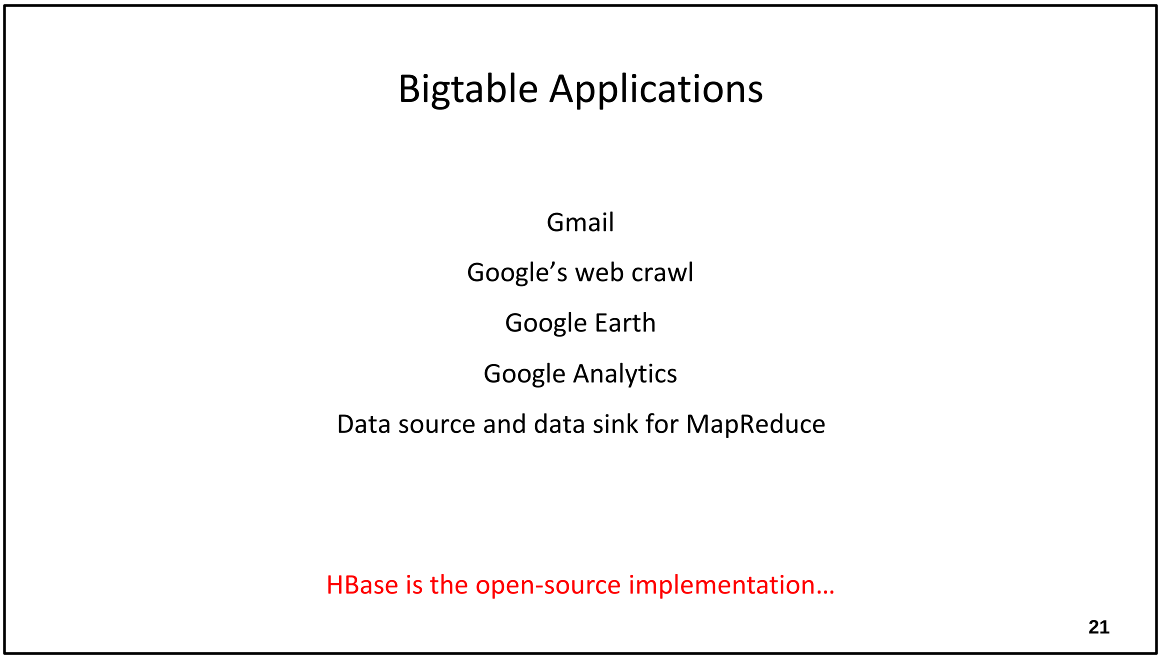


Solution: Replication

Machine fails: What happens?

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

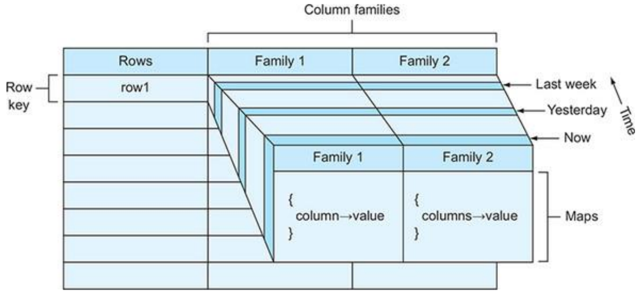
A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map

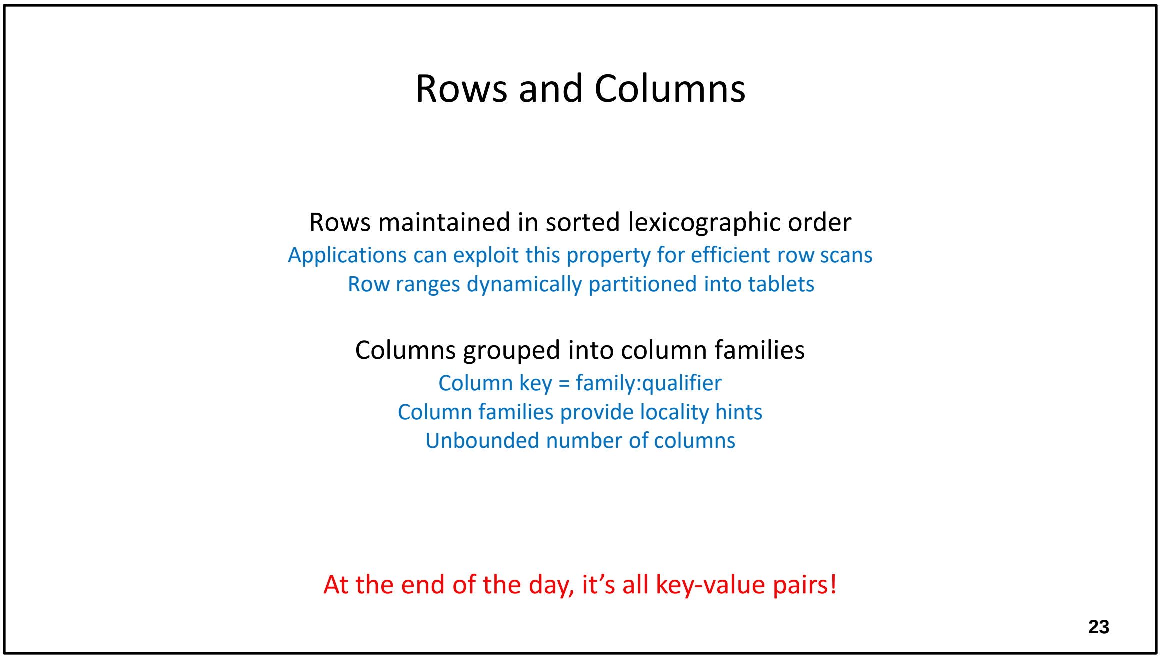
Map indexed by a row key, column key, and a timestamp

(row:string, column:string, time:int64) → uninterpreted byte array

Supports lookups, inserts, deletes

Single row transactions only





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

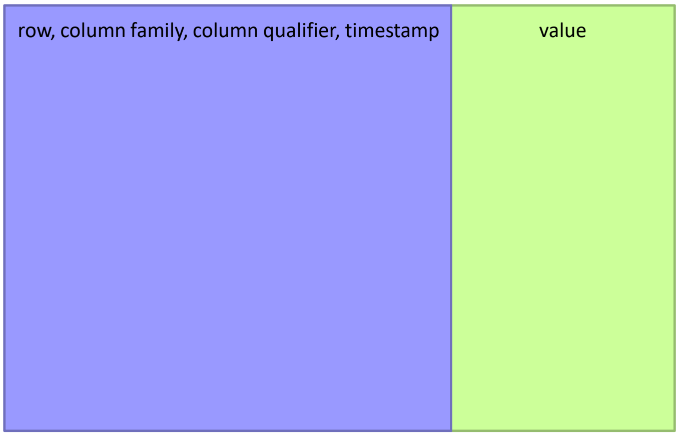
Column key = family:qualifier

Column families provide locality hints

Unbounded number of columns

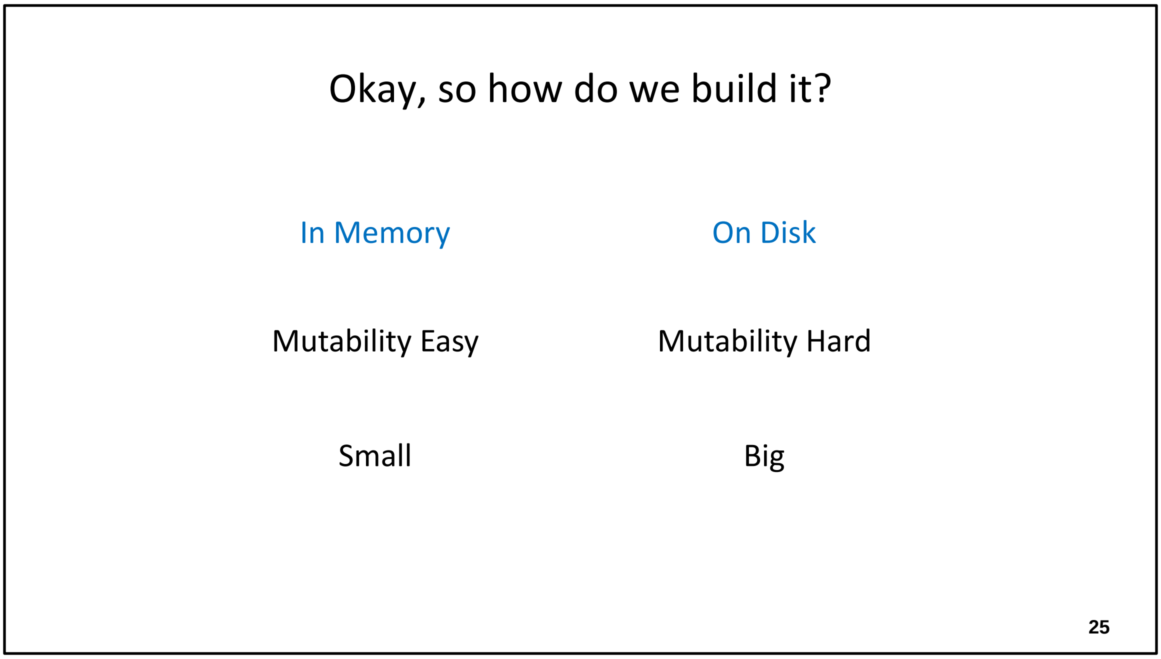
At the end of the day, it’s all key-value pairs!

Key-Values



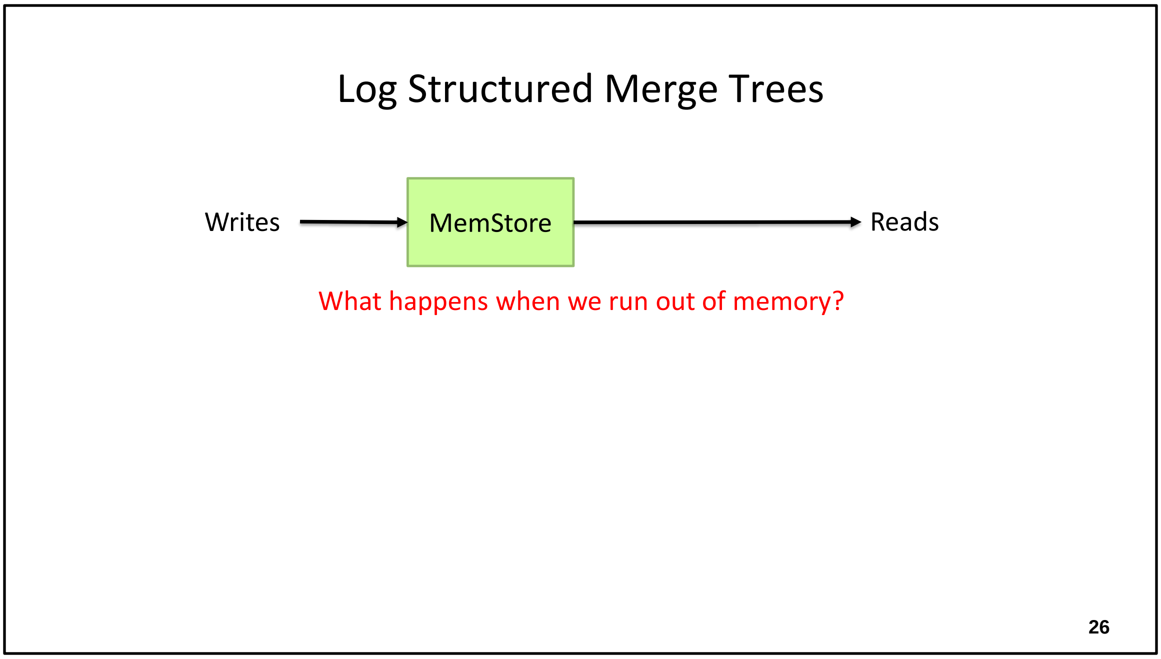
row, column family, column qualifier, timestamp

value



Okay, so how do we build it?

In Memory	On Disk
Mutability Easy	Mutability Hard
Small	Big

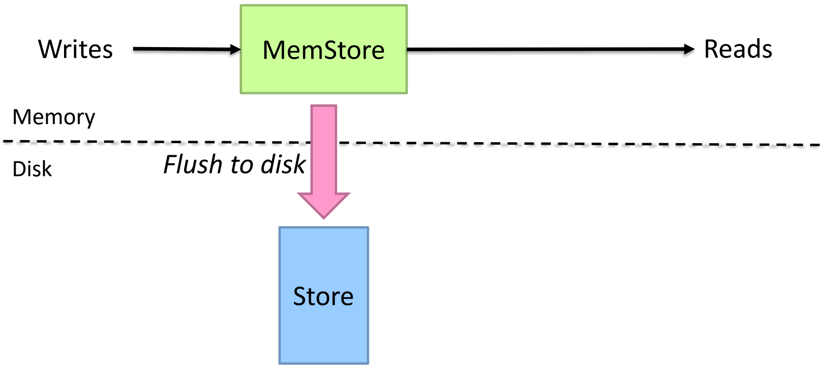


Log Structured Merge Trees

Writes → MemStore → Reads

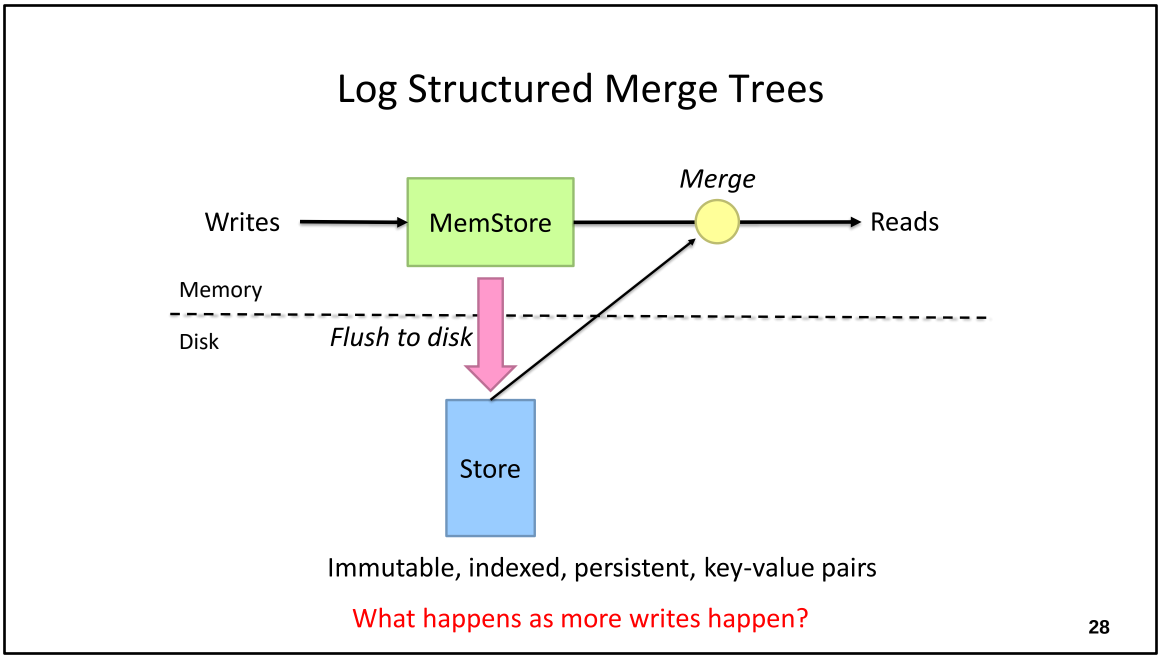
What happens when we run out of memory?

Log Structured Merge Trees



Immutable, indexed, persistent, key-value pairs

What happens to the read path?



Log Structured Merge Trees

Writes

MemStore

Merge

Reads

Memory

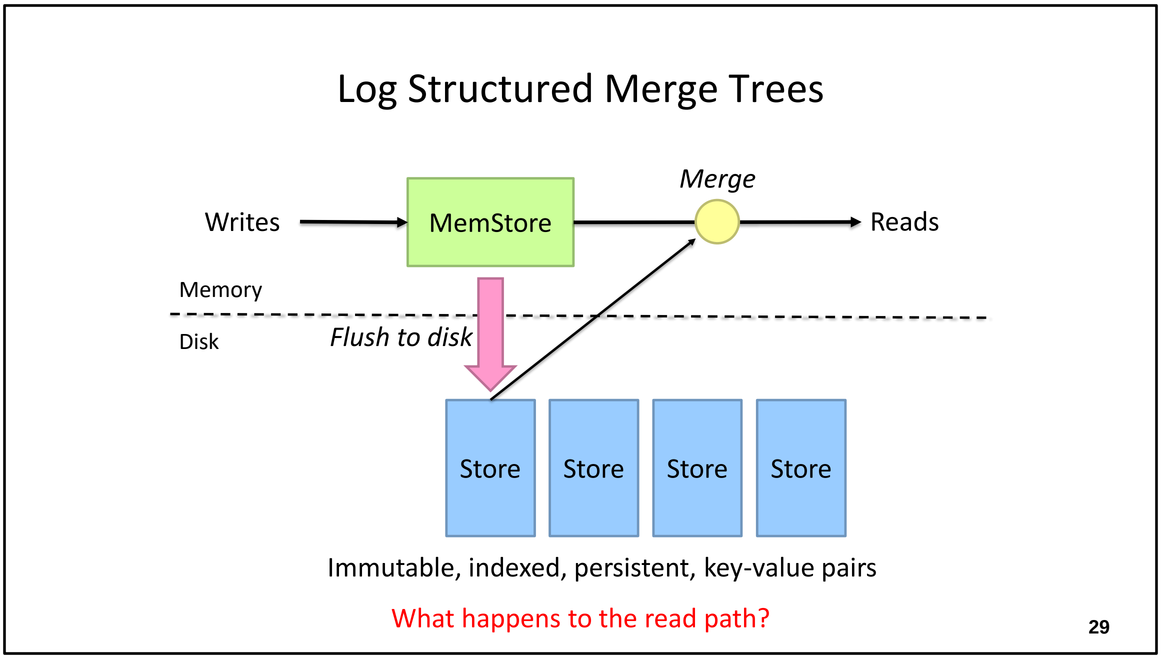
Disk

Flush to disk

Store

Immutable, indexed, persistent, key-value pairs

What happens as more writes happen?

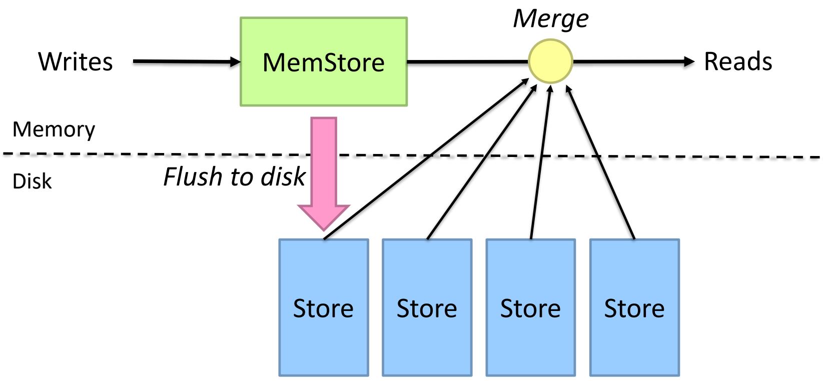


Log Structured Merge Trees

Immutable, indexed, persistent, key-value pairs

What happens to the read path?

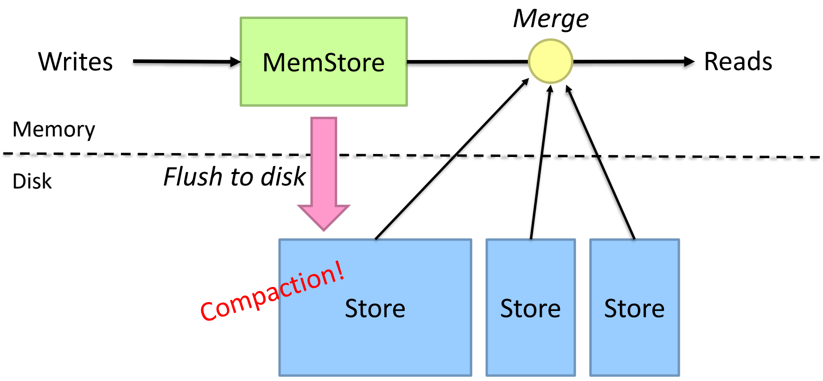
Log Structured Merge Trees



Immutable, indexed, persistent, key-value pairs

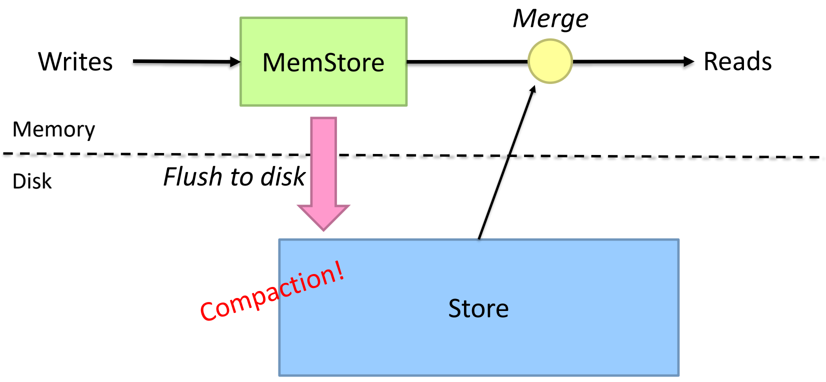
What's the next issue?

Log Structured Merge Trees



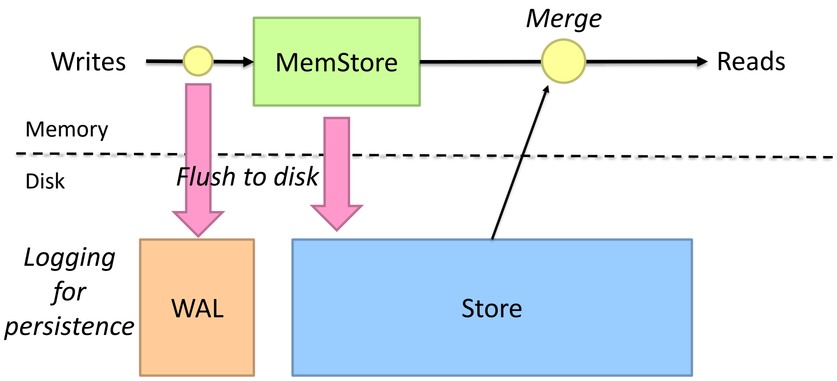
Immutable, indexed, persistent, key-value pairs

Log Structured Merge Trees



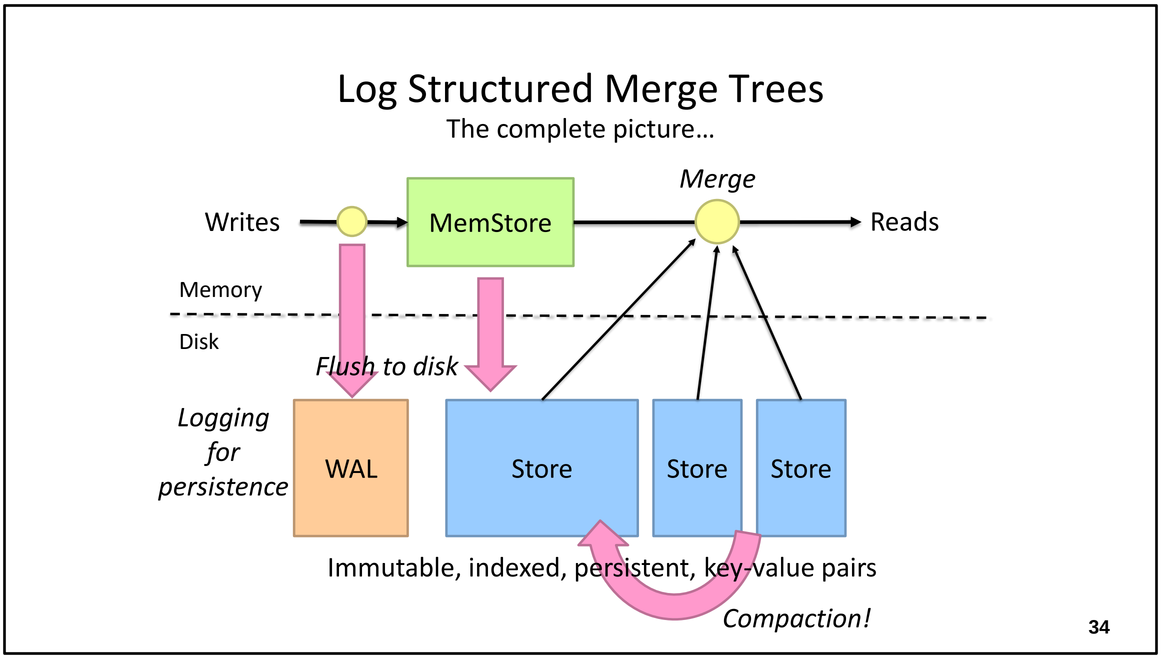
Immutable, indexed, persistent, key-value pairs

Log Structured Merge Trees



Immutable, indexed, persistent, key-value pairs

One final component…



Log Structured Merge Trees

The complete picture…

Writes

MemStore

Merge

Reads

Memory

Disk

Flush to disk

Logging for persistence

WAL

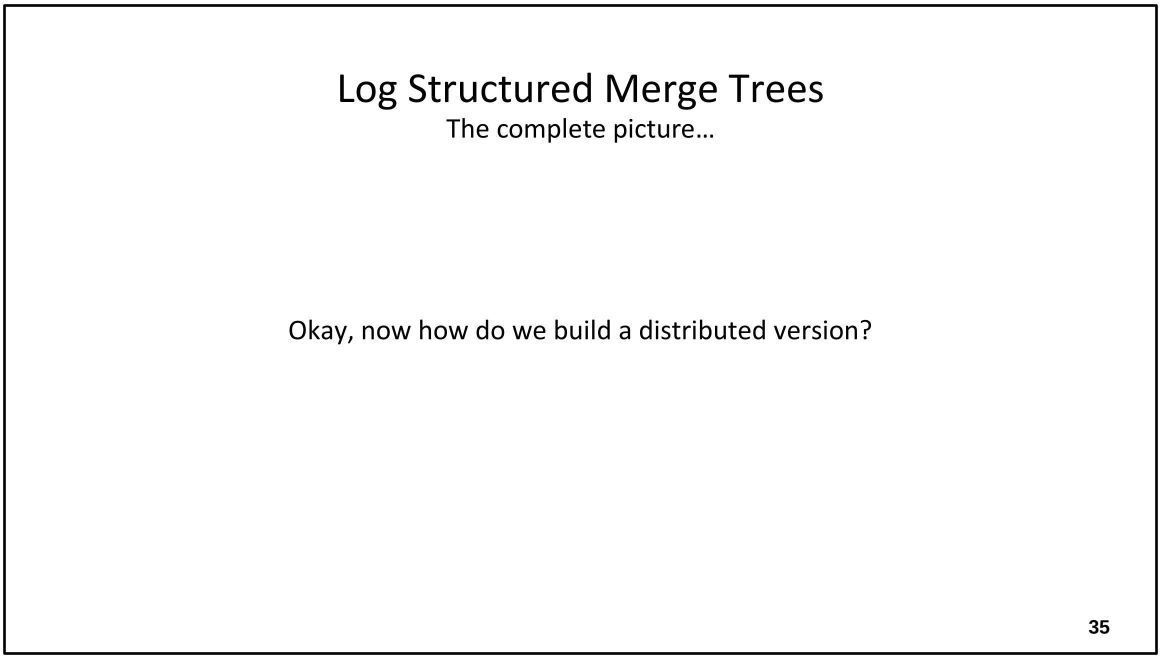
Store

Store

Store

Immutable, indexed, persistent, key-value pairs

Compaction!



Log Structured Merge Trees

The complete picture…

Okay, now how do we build a distributed version?

HBase Bigtable building blocks

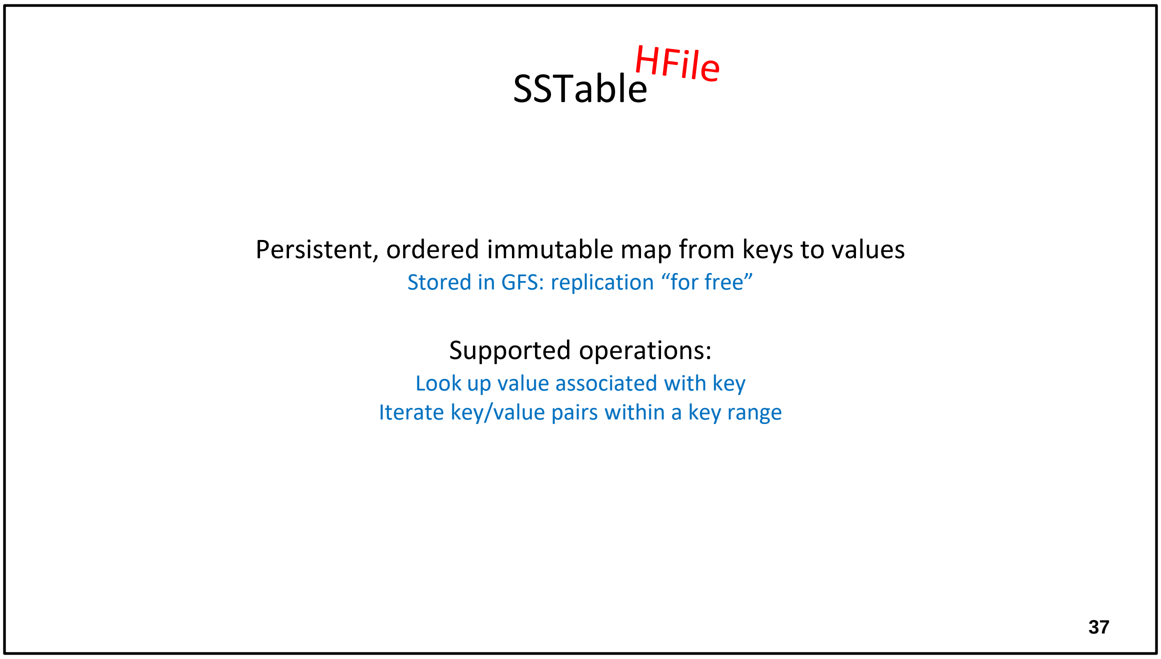
HDFS
GFS

HFile SSTable

Tablet Region

Regions Server
Tablet Server

Chubby
Zookeeper



SSTable HFile

Persistent, ordered immutable map from keys to values

Stored in GFS: replication “for free”

Supported operations:

Look up value associated with key

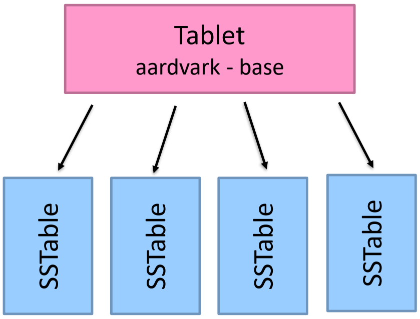
Iterate key/value pairs within a key range

Region

Tablet

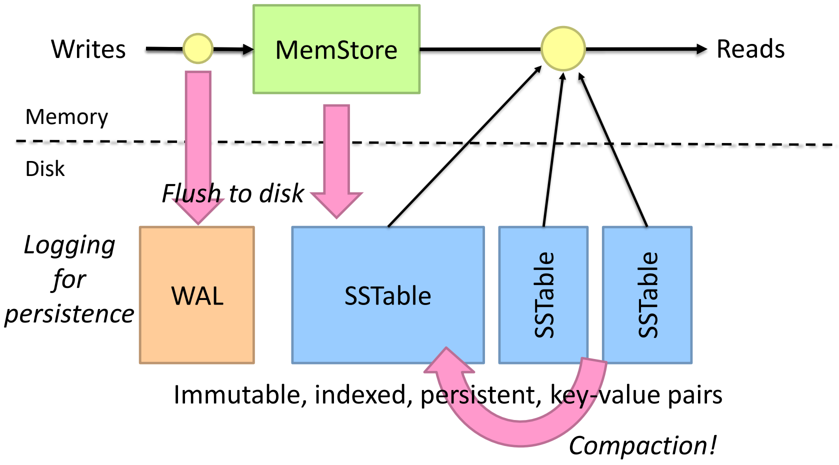
Dynamically partitioned range of rows

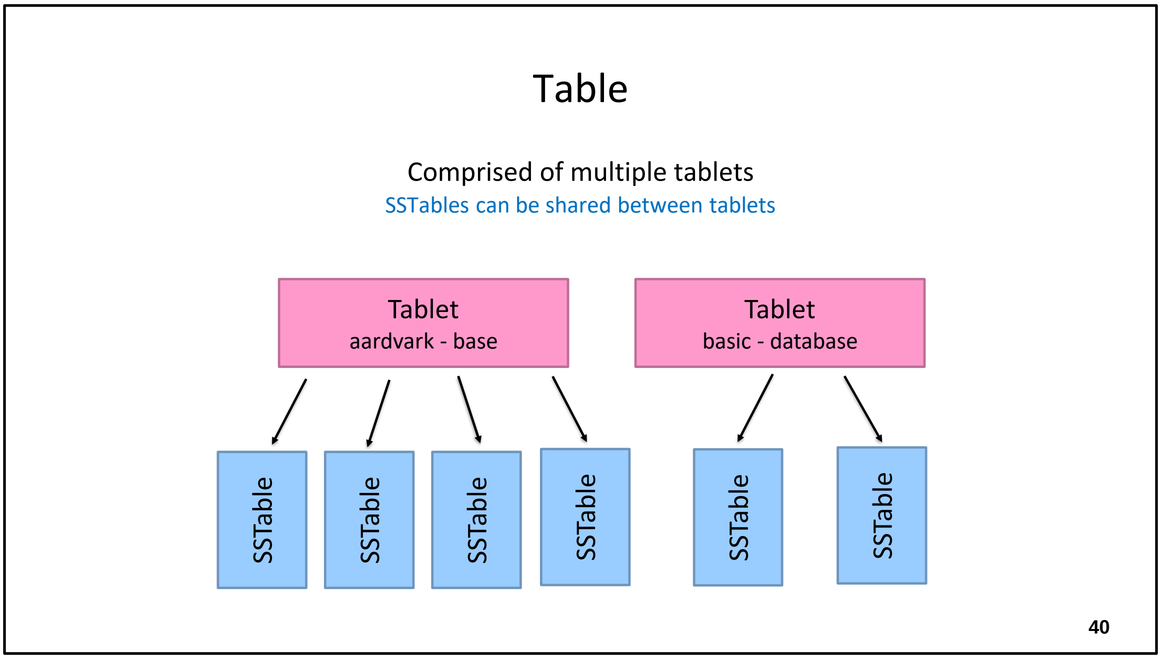
Comprised of multiple SSTables



Region Server

Tablet Server





Table

Comprised of multiple tablets

SSTables can be shared between tablets

Tablet
aardvark - base

Tablet
basic - database

SSTable

SSTable

SSTable

SSTable

SSTable

SSTable

Region Region Server

Tablet to Tablet Server Assignment

Each tablet is assigned to one tablet server at a time

Exclusively handles read and write requests to that tablet

What happens when a tablet grow too big?

What happens when a tablet server fails?

We need a lock service!

HBase Bigtable building blocks

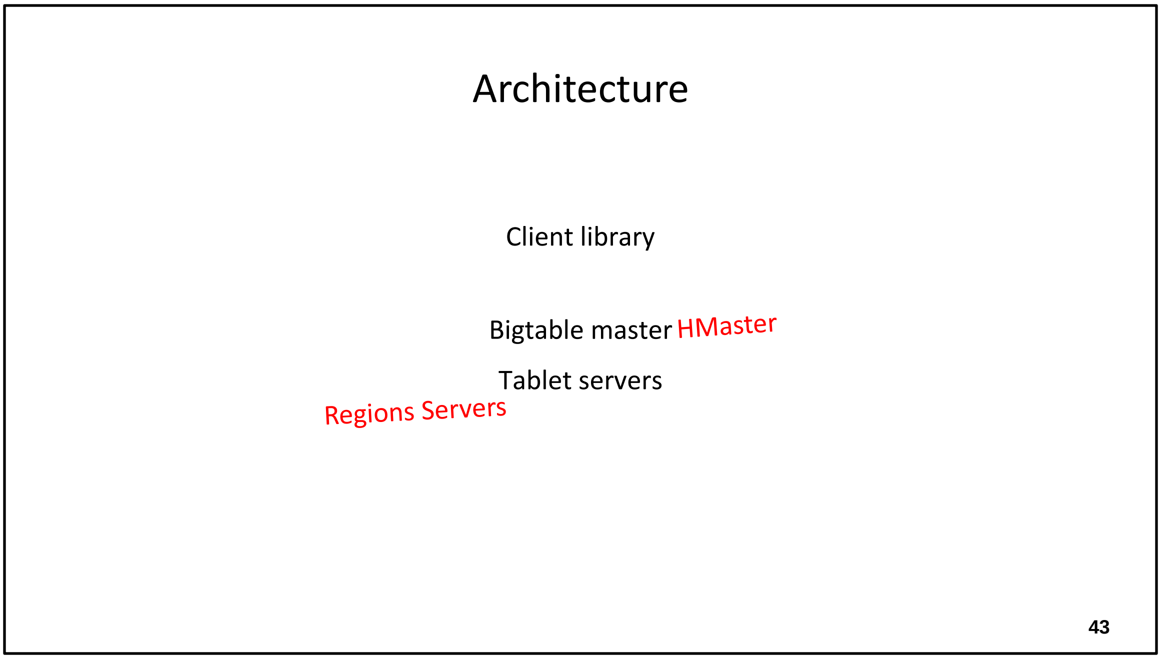
HDFS
GFS

HFile SSTable

Tablet Region

Regions Server
Tablet Server

Chubby
Zookeeper



Architecture

Client library

Bigtable master HMaster

Tablet servers

Regions Servers

Bigtable Master

Roles and responsibilities:

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

Tablet structure changes:

Table creation/deletion (master initiated)
Tablet merging (master initiated)
Tablet splitting (tablet server initiated)

HBase

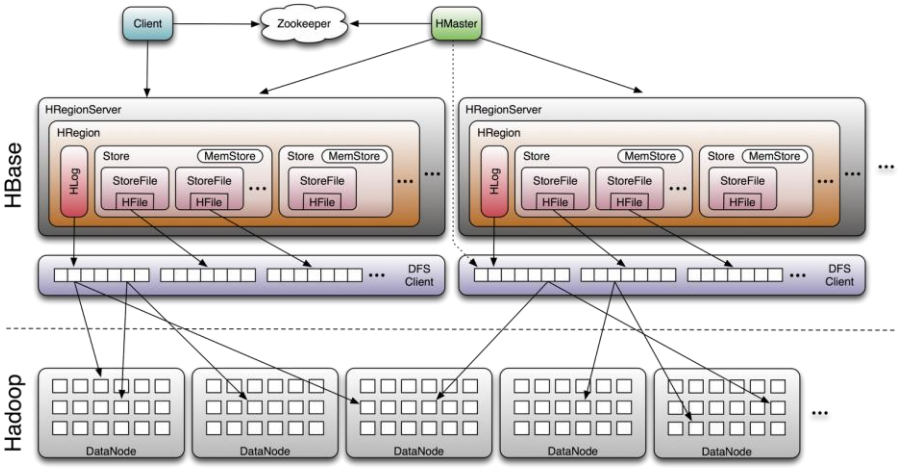


Image Source: http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html