Computer Science 425
Distributed Systems
CS 425 / ECE 428

Indranil Gupta (Indy)

Oct 31, 2013

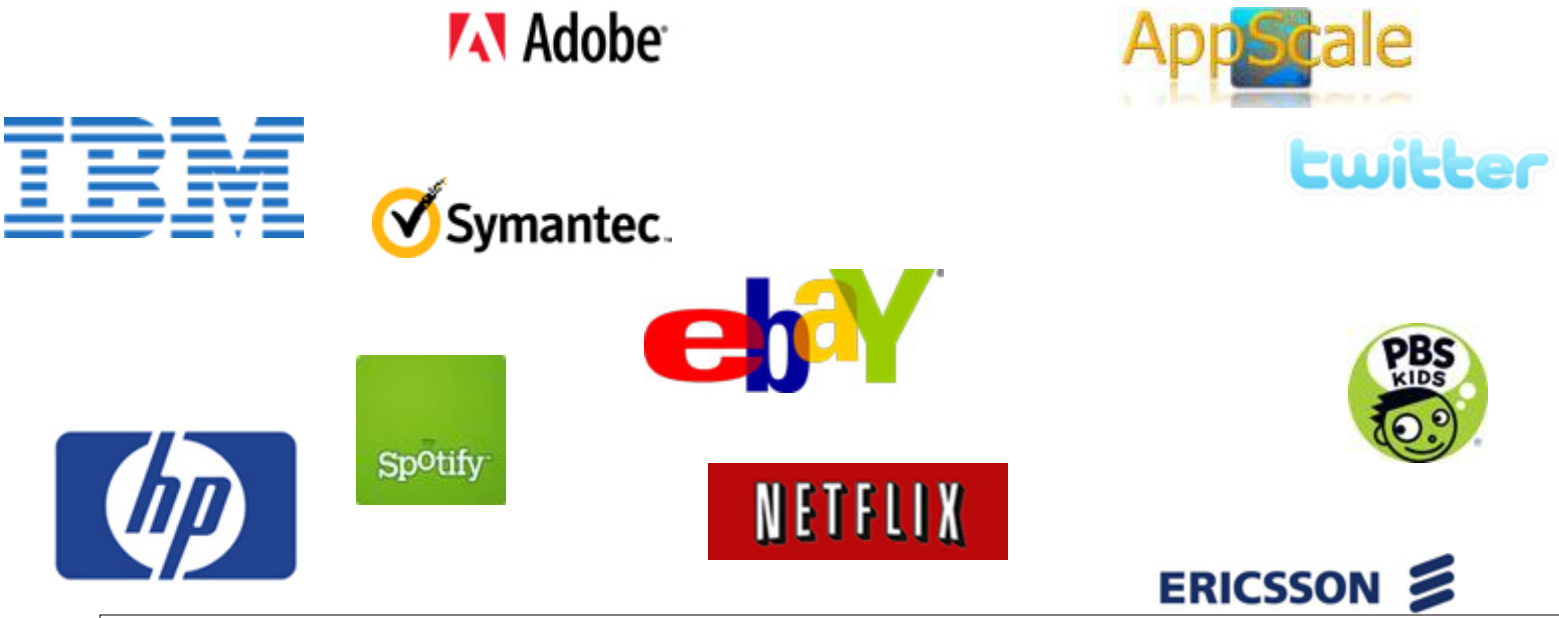
Key-value/NoSQL Stores

Lecture 20

- Cassandra 2.0 Documentation at datastax.com
- Cassandra 2.0 Paper
- Cassandra NoSQL presentation
- Cassandra 1.0 documentation at datastax.com
- Cassandra Apache project wiki
- Hbase

Cassandra

- Originally designed at Facebook
- Open-sourced
- Some of its myriad users:



- With this many users, one would think
  - Its design is very complex
  - We in our class won't know anything about its internals
  - Let's find out!

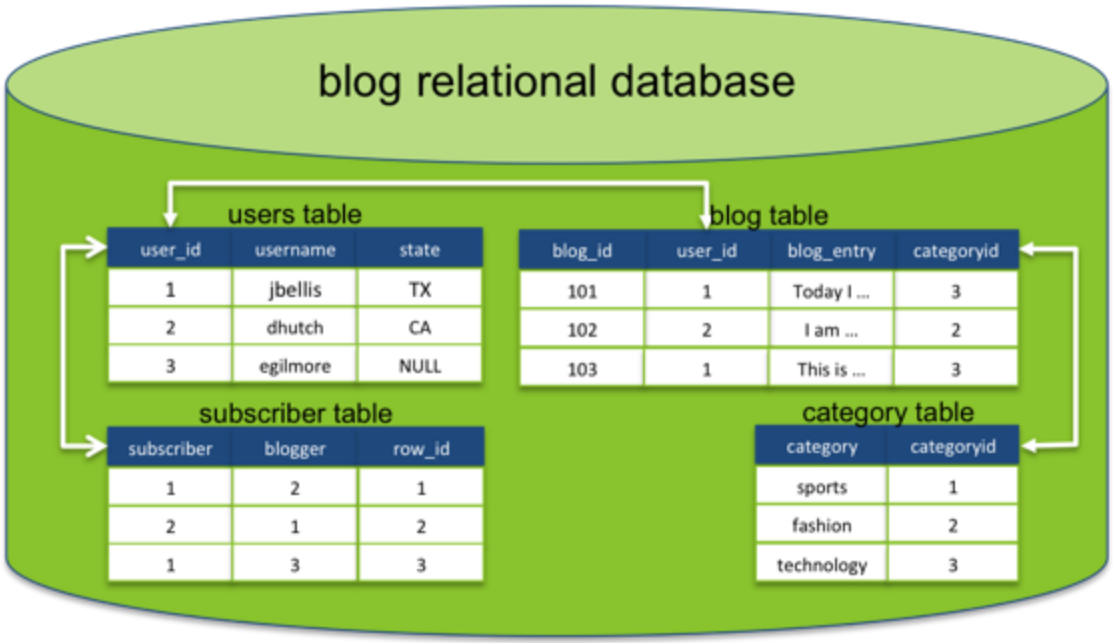
Why Key-value Store?

- **(Business) Key -> Value**
- **(twitter.com) tweet id -> information about tweet**
- **(kayak.com) Flight number -> information about flight, e.g., availability**
- **(yourbank.com) Account number -> information about it**
- **(amazon.com) item number -> information about it**

- **Search is usually built on top of a key-value store**

Isn't that just a database?

- **Yes**
- **Relational Databases (RDBMSs) have been around for ages**
- **MySQL is the most popular among them**
- **Data stored in tables**
- **Schema-based, i.e., structured tables**
- **Queried using SQL**



SQL queries: SELECT user_id from users WHERE username = "jbellis"

Example's Source

RDBMS Mismatch with today's workloads

Today:

- **Data: Large and unstructured**
- **Lots of random reads and writes**
- **Foreign keys rarely needed**
- **Need**
  - **Speed**
  - **No Single point of failure**
  - **Low TCO and admin**
  - **Incremental Scalability**
  - **Scale out, not up**

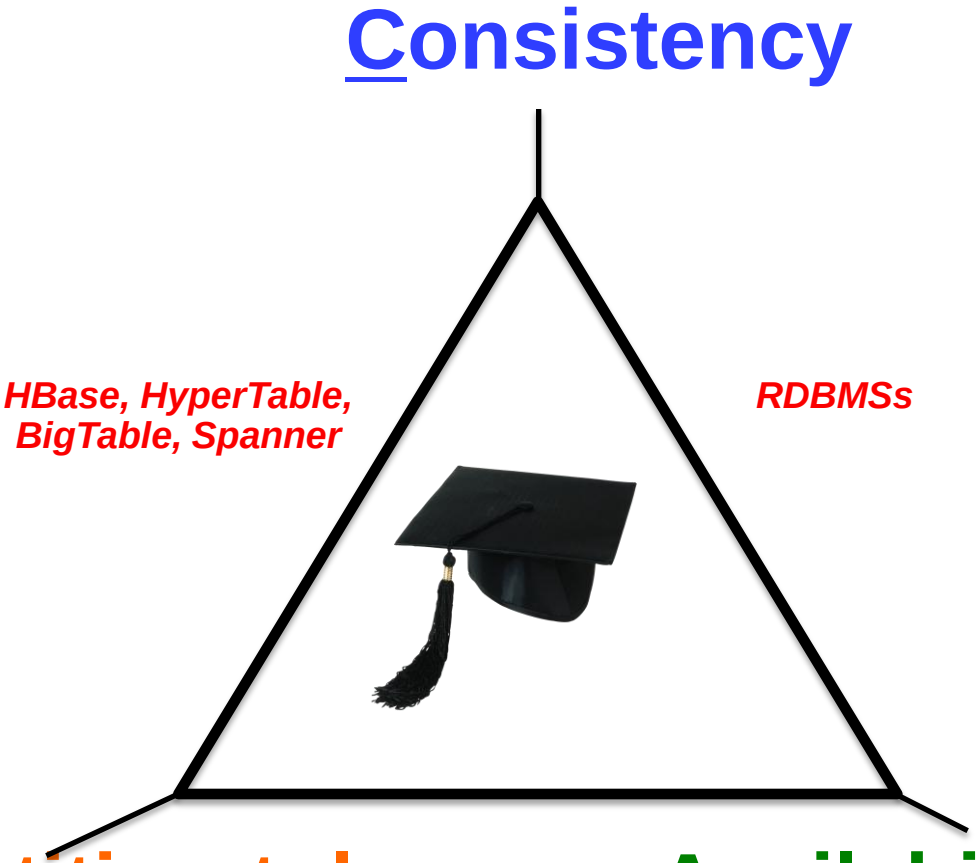
CAP Theorem

- **Proposed by Eric Brewer (Berkeley)**
- **Subsequently proved by Gilbert and Lynch**
- **In a distributed system you can satisfy at most 2 out of the 3 guarantees**
  1. **Consistency: all nodes have same data at any time**
  2. **Availability: the system allows operations all the time**
  3. **Partition-tolerance: the system continues to work in spite of network partitions**

- **Cassandra**
  - **Eventual (weak) consistency, Availability, Partition-tolerance**
- **Traditional RDBMSs**
  - **Strong consistency over availability under a partition**

CAP Tradeoff

- Starting point for NoSQL Revolution
- Conjectured by Eric Brewer in 2000, proved by Gilbert and Lynch in 2002
- A distributed storage system can achieve **at most two of C, A, and P**.
- When partition-tolerance is important, you have to choose between consistency and availability

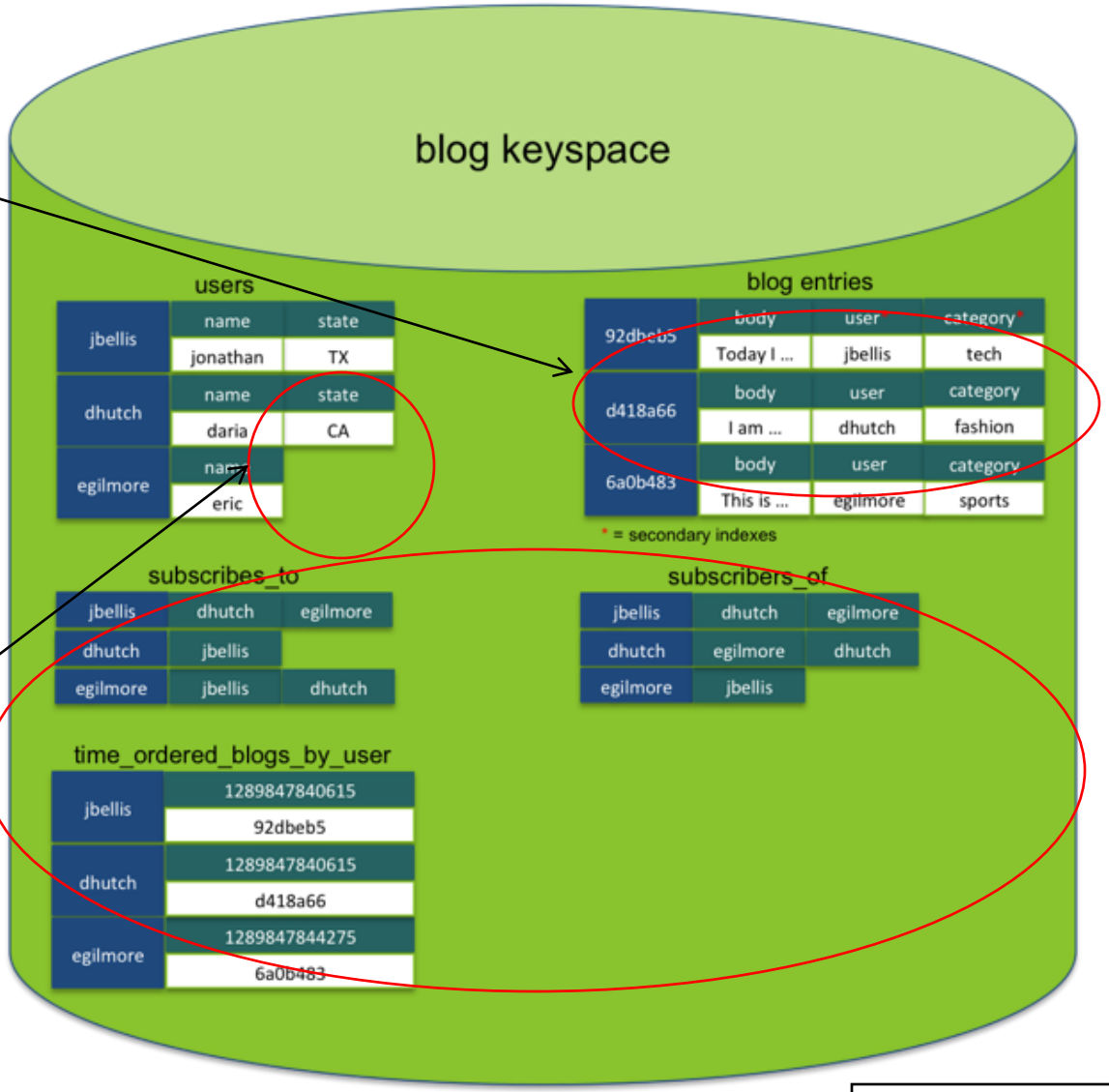


Eventual Consistency

- ***If all writers stop (to a key), then all its values (replicas) will converge eventually.***
- **If writes continue, then system always tries to keep converging.**
  - **Moving "wave" of updated values always lagging behind the latest values sent by clients.**
- **May return stale values to clients (e.g., if many back to back writes).**
- **But works well if writers are rare and system converges quickly.**

Cassandra Data Model

- **Column Families:**
  - Like SQL tables
  - but may be unstructured (client-specified)
  - Can have index tables
- **Hence "column-oriented databases"/ "NoSQL"**
  - "Not Only SQL"
  - No schemas
  - Some columns missing
  - Columns stored together (rather than rows)
  - Supports get(key) and put(key, value)
  - Often write-heavy workloads

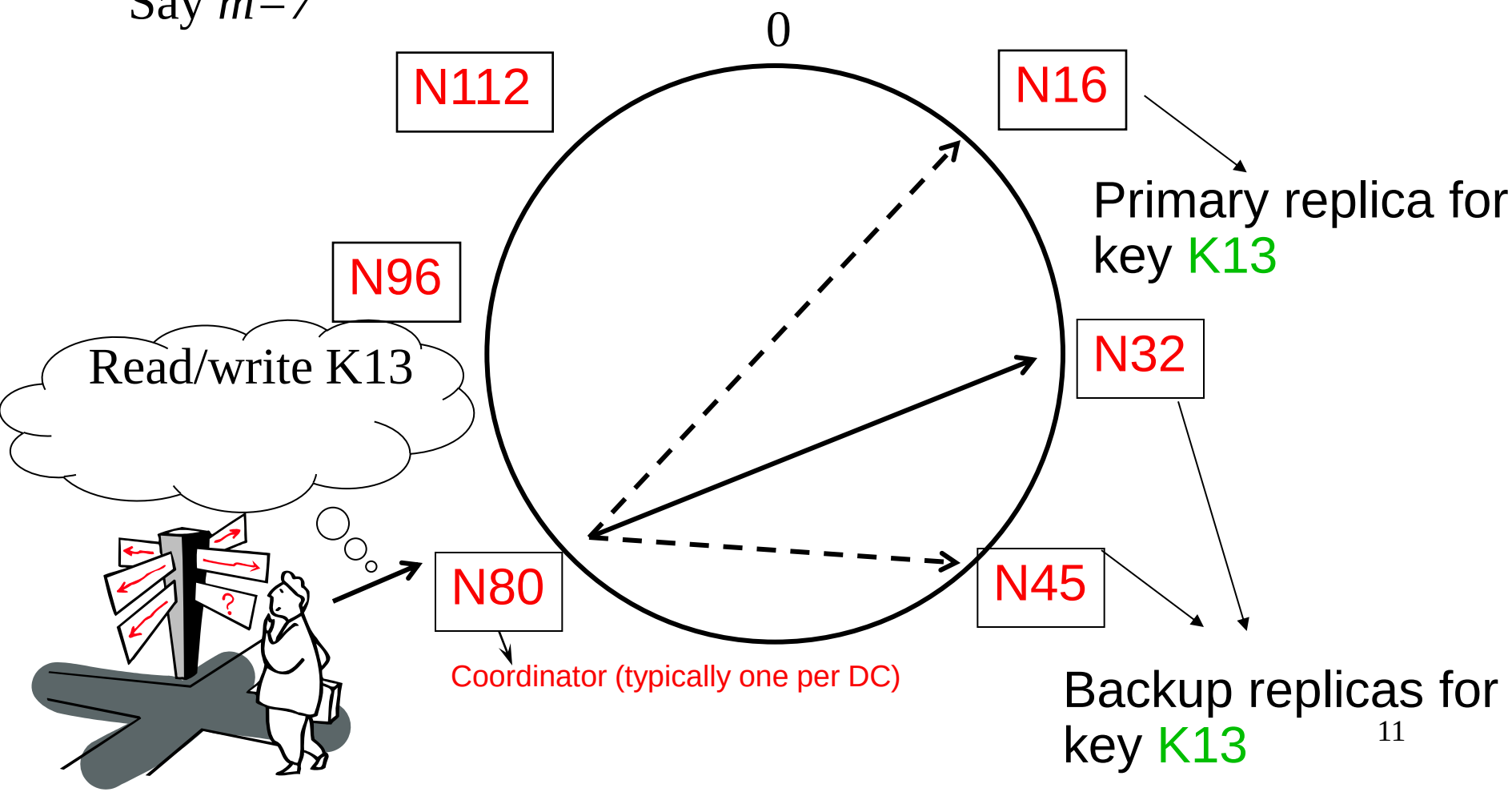


Let's go Inside: Key -> Server Mapping

- **How do you decide which server(s) a key-value resides on?**

(Remember this?)

Say $m=7$



Cassandra uses a Ring-based DHT but without routing

Dynamic repartitioning: use virtual nodes, or move servers along ring

Writes

- Need to be lock-free and fast (no reads or disk seeks)
- Client sends write to one front-end node in Cassandra cluster
  - Front-end = Coordinator, assigned per key
- Which (via Partitioning function) sends it to all replica nodes responsible for key
  - Always writable: Hinted Handoff
    - » If any replica is down, the coordinator writes to all other replicas, and keeps the write locally until down replica comes back up.
    - » When all replicas are down, the Coordinator (front end) buffers writes (for up to a few hours).
  - Provides Atomicity for a given key (i.e., within ColumnFamily)
- One ring per datacenter
  - Per-DC coordinator elected to coordinate with other DCs
  - Election done via Zookeeper, which runs a Paxos variant

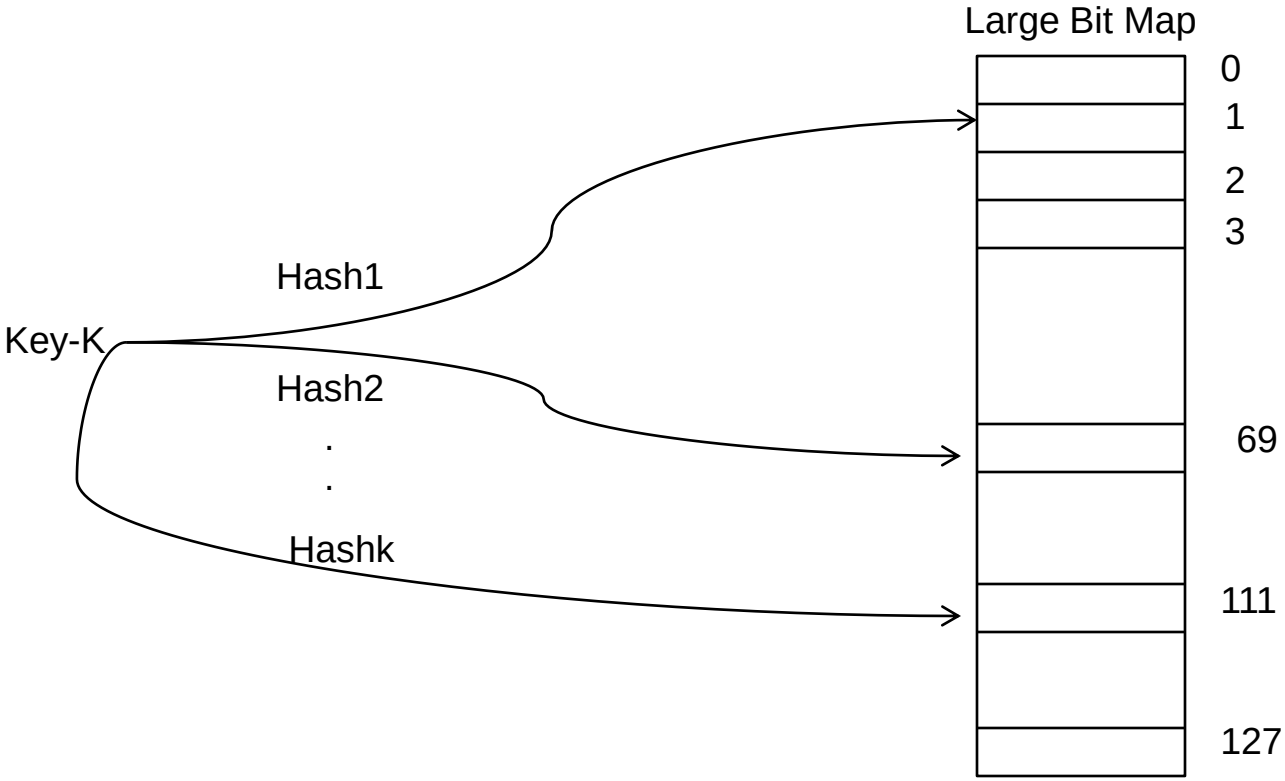
Writes at a replica node

On receiving a write

- **1. log it in disk commit log**
- **2. Make changes to appropriate memtables**
  - **In-memory representation of multiple key-value pairs**
- **Later, when memtable is full or old, flush to disk**
  - **Data File: An SSTable (Sorted String Table) – list of key value pairs, sorted by key**
  - **Index file: An SSTable – (key, position in data sstable) pairs**
    - » **And a Bloom filter (for efficient search)**
- **Compaction: Data udpates accumulate over time and sstables and logs need to be compacted**
  - **Merge SSTables, e.g., by merging key updates, etc.**
- **Reads need to touch log and multiple SSTables**
  - **May be slower than writes**

Bloom Filter

- **Compact way of representing a set of items**
- **Checking for existence in set is cheap**
- **Some probability of false positives: an item not in set may check true as being in set**
- **Never false negatives**



On insert, set all hashed bits.

On check-if-present, return true if all hashed bits set.

- False positives

False positive rate low

- k=4 hash functions
- 100 items
- 3200 bits
- FP rate = 0.02%

Deletes and Reads

- **Delete: don't delete item right away**
  - **add a tombstone to the log**
  - **Compaction will eventually remove tombstone and delete item**
- **Read: Similar to writes, except**
  - **Coordinator can contacts a number of replicas (e.g., in same rack) specified by consistency level**
    - » **Forwards read to replicas that have responded quickest in past**
    - » **Returns latest timestamp value**
  - **Coordinator also fetches value from multiple replicas**
    - » **check consistency in the background, initiating a read-repair if any two values are different**
    - » **Brings all replicas up to date**
  - **Need to check SSTables => reads slower than writes (but still fast)**

Cassandra uses Quorums

(Remember this?)

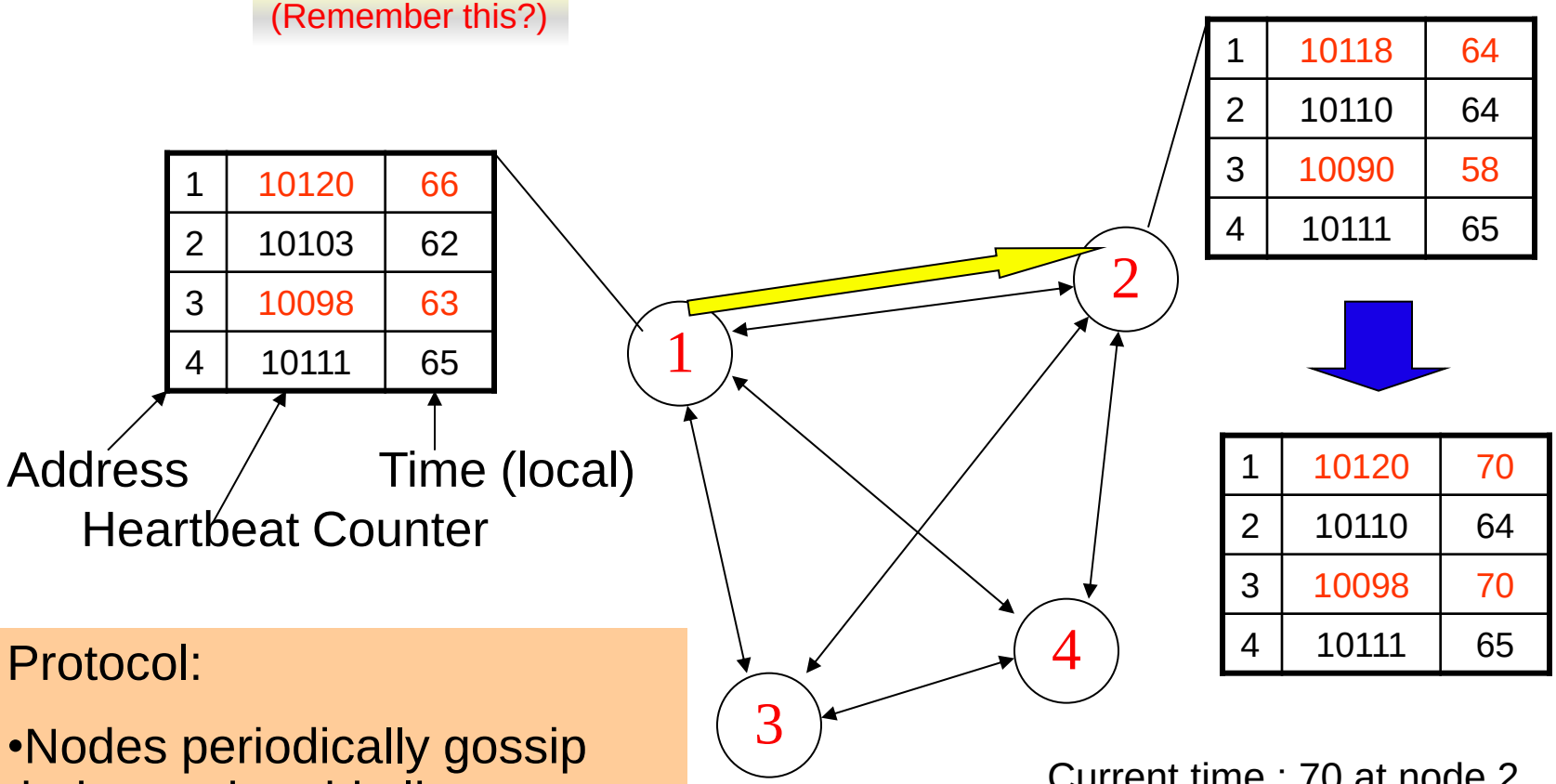
- **Reads**
  - **Wait for R replicas (R specified by clients)**
  - **In background check for consistency of remaining N-R replicas, and initiate read repair if needed (N = total number of replicas for this key)**
- **Writes come in two flavors**
  - **Block until quorum is reached**
  - **Async: Write to any node**
- **Quorum Q = N/2 + 1**
- **R = read replica count, W = write replica count**
- **If W+R > N and W > N/2, you have consistency**
- **Allowed (W=1, R=N) or (W=N, R=1) or (W=Q, R=Q)**

Cassandra uses Quorums

- **In reality, a client can choose one of these levels for a read/write operation:**
  - **ANY: any node (may not be replica)**
  - **ONE: at least one replica**
  - **Similarly, TWO, THREE**
  - **QUORUM: quorum across all replicas in all datacenters**
  - **LOCAL_QUORUM: in coordinator's DC**
  - **EACH_QUORUM: quorum in every DC**
  - **ALL: all replicas all DCs**
- **For Write, you also have**
  - **SERIAL: claims to implement linearizability for transactions**

Cluster Membership

(Remember this?)



Protocol:

- Nodes periodically gossip their membership list
- On receipt, the local membership list is updated

Cassandra uses gossip-based cluster membership

Fig and animation by: Dongyun Jin and Thuy Ngyuen

Cluster Membership, contd.

(Remember this?)

- Accrual detector: Tune failure declaration to past behavior of node
- FD outputs a value (PHI) representing suspicion
- Apps set an appropriate threshold to declare a failure
- PHI = 5 => 10-15 sec detection time
- PHI calculation for a member
  - Inter-arrival times for gossip messages
  - PHI(t) = - log(CDF or Probability(t_now – t_last))/log 10
  - PHI basically determines the detection timeout, but is sensitive to actual inter-arrival time variations for gossiped heartbeats

Cassandra uses gossip-based cluster membership

Fig and animation by: Dongyun Jin and Thuy Ngyuen

Data Placement Strategies

- **Replication Strategy: two options:**
  1. ***SimpleStrategy***
  2. ***NetworkTopologyStrategy***

1. SimpleStrategy: uses a "Partitioner"

1. ***RandomPartitioner*: Chord-like**
2. ***ByteOrderedPartitioner*: Assigns ranges of keys to servers.**
   - » **Easier for range searches (e.g., Get me all twitter users starting with [a-b])**

2. NetworkTopologyStrategy: for multi-DC deployments

- **Two replicas per DC: allows a consistency level of ONE**
- **Three replicas per DC: allows a consistency level of LOCAL_QUORUM**
- **Per DC**
  - » **First replica placed according to Partitioner**
  - » **Then go clockwise around ring until you hit different rack**

Snitches

- **Maps: IPs to racks and DCs. Configured in cassandra.yaml config file**
- **Some options:**
  - **SimpleSnitch: Unaware of Topology (Rack unaware)**
  - **RackInferring: Assumes topology of network by octet of server's IP address**
    - **101.201.301.401 = x.<DC octet>.<rack octet>.<node octet>**
  - **PropertyFileSnitch: uses a config file**
  - **EC2Snitch: uses EC2.**
    - **EC2 Region = DC**
    - **Availability zone = rack**
- **Other snitch options available**

Vs. SQL

- **MySQL is the most popular (and has been for a while)**
- **On > 50 GB data**
- **MySQL**
  - **Writes 300 ms avg**
  - **Reads 350 ms avg**
- **Cassandra 1.0**
  - **Writes 0.12 ms avg**
  - **Reads 15 ms avg**

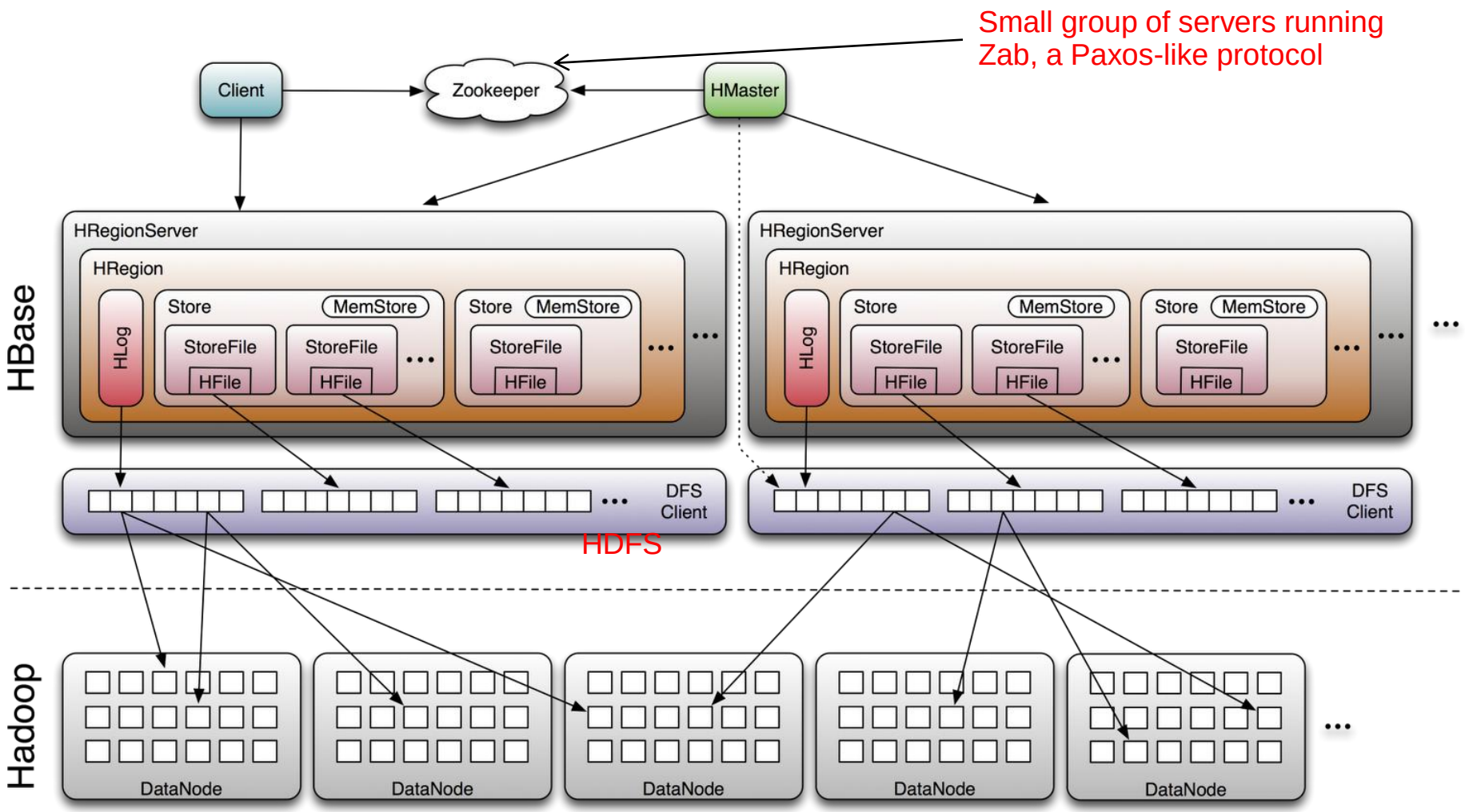
Cassandra Summary

- While RDBMS provide ACID (Atomicity Consistency Isolation Durability)
- Cassandra provides BASE
  - Basically Available Soft-state Eventual Consistency
  - Prefers Availability over consistency
- Other NoSQL products
  - MongoDB, Riak (look them up!)
- Next: HBase
  - Prefers (strong) Consistency over Availability

HBase

- Google's BigTable was first "blob-based" storage system
- Yahoo! Open-sourced it -> HBase
- Major Apache project today
- Facebook uses HBase internally
- API
  - Get/Put(row)
  - Scan(row range, filter) – range queries
  - MultiPut

HBase Architecture

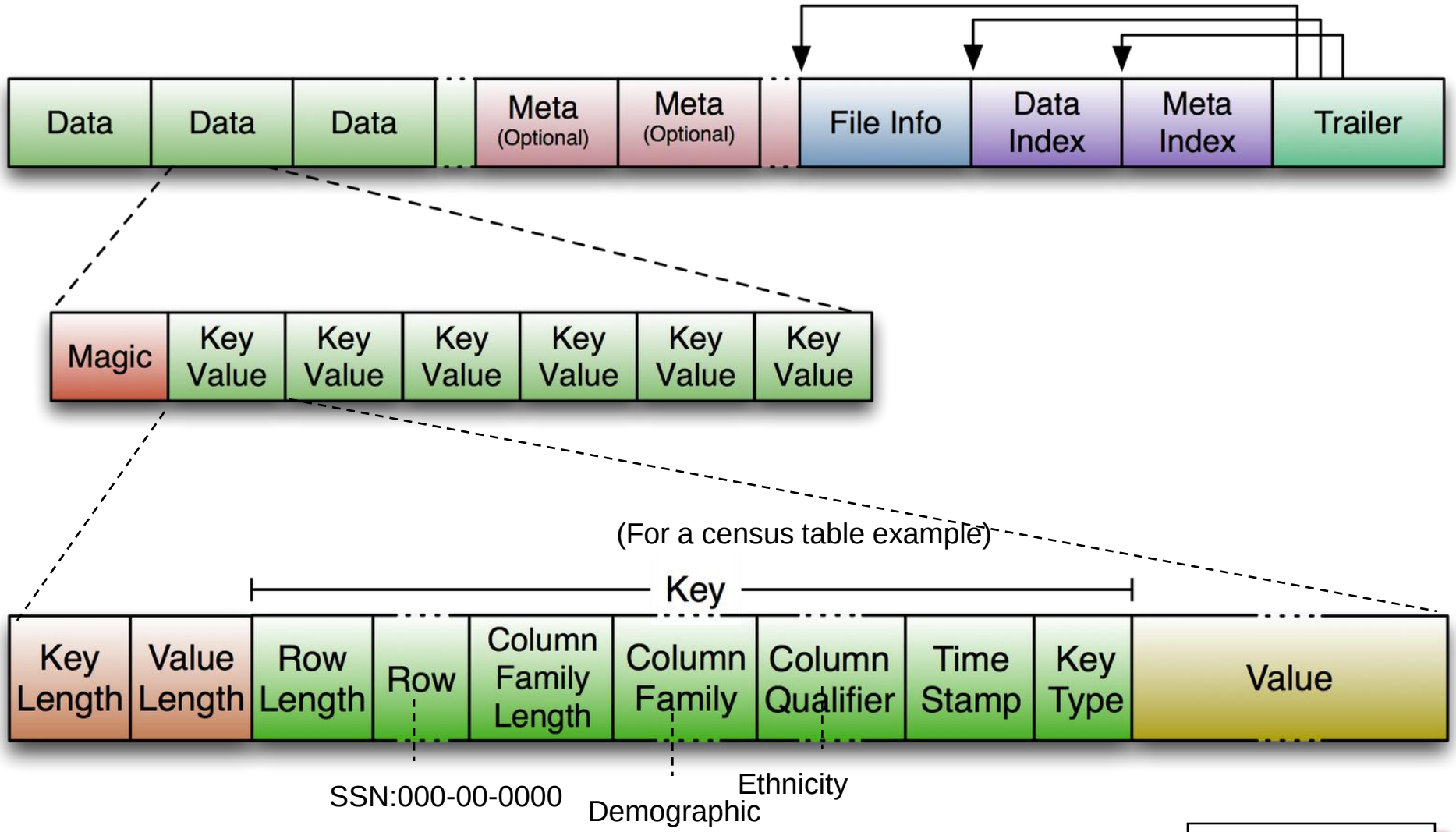


Source: http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html

HBase Storage hierarchy

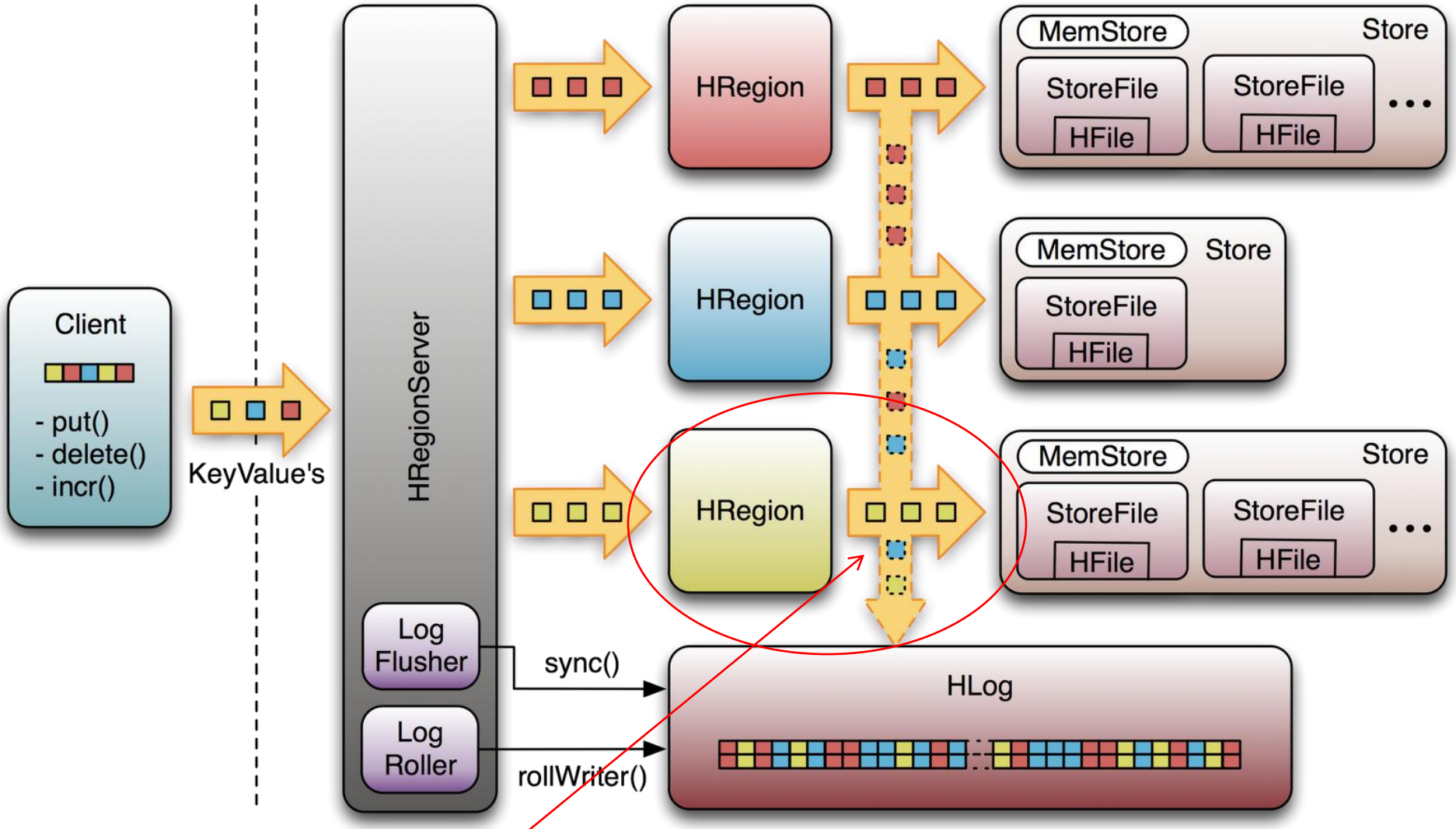
- **HBase Table**
  - Split it into multiple regions: replicated across servers
    - » One Store per ColumnFamily (subset of columns with similar query patterns) per region
      - Memstore for each Store: in-memory updates to Store; flushed to disk when full
        - StoreFiles for each store for each region: where the data lives
          - Blocks
- **HFile**
  - SSTable from Google's BigTable

HFile



Source: http://blog.cloudera.com/blog/2012/06/hbase-io-hfile-input-output/

Strong Consistency: HBase Write-Ahead Log



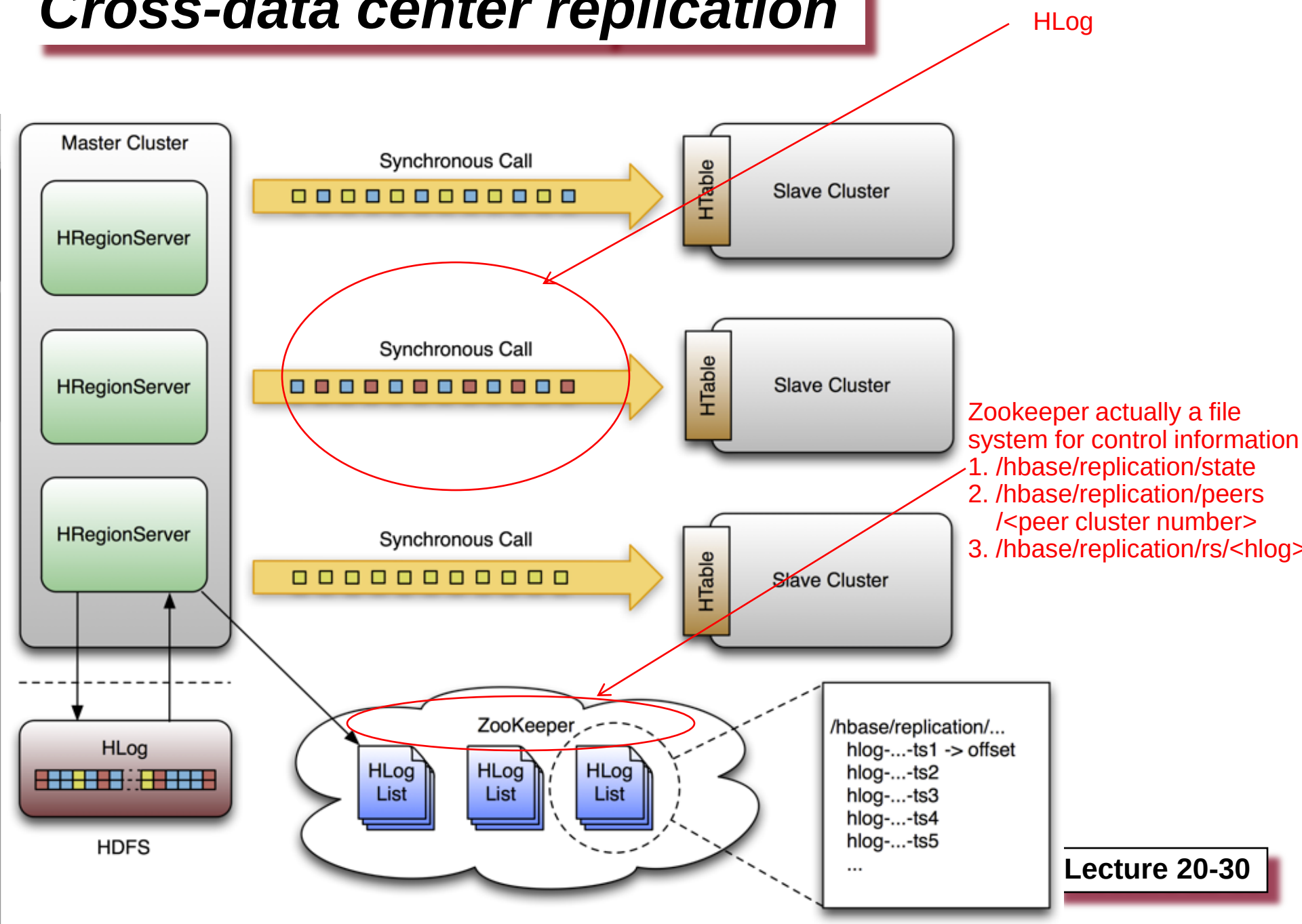
Write to HLog before writing to MemStore
Can recover from failure

Source: http://www.larsgeorge.com/2010/01/hbase-architecture-101-write-ahead-log.html

Log Replay

- **After recovery from failure, or upon bootup (HRegionServer/HMaster)**
  - **Replay any stale logs (use timestamps to find out where the database is w.r.t. the logs)**
  - **Replay: add edits to the MemStore**
- **Why one HLog per HRegionServer rather than per region?**
  - **Avoids many concurrent writes, which on the local file system may involve many disk seeks**

Cross-data center replication



Summary

- **Key-value stores and NoSQL faster but provide weaker guarantees**
- **Next week: Stream processing in clouds**
- **MP3: By now, you must have a basic working system (may not yet satisfy all the requirements)**
- **HW3: due Nov 14**