Distributed Storage Systems part 2

Marko Vukolić

Distributed Systems and Cloud Computing

Distributed storage systems

- **Part I**
  - CAP Theorem
  - Amazon Dynamo
- **Part II**
  - Cassandra

Cassandra in a nutshell

- **Distributed key-value store**
  - For storing large amounts of data
  - Linear scalability, high availability, no SPF
- **Tunable consistency**
  - In principle (and a typical deployment): eventually consistent
    - Hence in AP
  - Can also have strong consistency
    - Shifts Cassandra to CP
- **Column-oriented data model**
  - With one key per row

Cassandra in a nutshell

- **Roughly speaking, Cassandra can be seen as a combination of two familiar data stores**
  - HBase (Google BigTable)
  - Amazon Dynamo
- **Hbase data model**
  - One key per row
  - Columns, column families, …
- **Distributed architecture of Amazon Dynamo**
  - Partitioning, placement (consistent hashing)
  - Replication, gossip-based membership, anti-entropy,…
- **There are some differences as well**

Cassandra history

- **Cassandra was a Troyan princess**
  - Daughter of King Priam and Queen Hecuba
- **Origins in Facebook**
  - Initially designed (2007) to fullfill the storage needs of the Facebook's Inbox Search
  - Open sourced (2008)
- **Now used by many companies like Twitter, Netflix, Disney, Cisco, Rackspace, …**
  - Although Facebook opted for HBase for Inbox Search

Apache Cassandra

- **Top-level Apache project**
- **http://cassandra.apache.org/**
  - Latest release 1.2.4

Inbox Search: background

- **MySQL revealed to have at least two issues for Inbox Search**
  - Latency
  - Scalability

- **Cassandra designed to overcome these issues**
  - The maximum of column per row is 2 billion
  - 1-2 orders of magnitude lower latency than MySQL in Facebook's evaluations

We will cover

- **Data partitioning ←**
- **Replication**
- **Data Model**
- **Handling read and write requests**
- **Consistency**

Partitioning

- **Like Amazon Dynamo, partitioning in Cassandra is based on consistent hashing**
- **Two main partitioning strategies**
  - RandomParitioner
  - ByteOrderedParitioner
- **Partitioning strategy cannot be changed on-fly**
  - All data needs to be reshuffled
  - Needs to be chosen carefuly

RandomPartitioner

- **Closely mimics partitioning in Amazon Dynamo**
  - Does not follow virtual nodes though***
  - Q: What are the consequences on load balancing?
- *****Edit: Starting in version 1.2. Cassandra implements virtual nodes just like Amazon Dynamo**

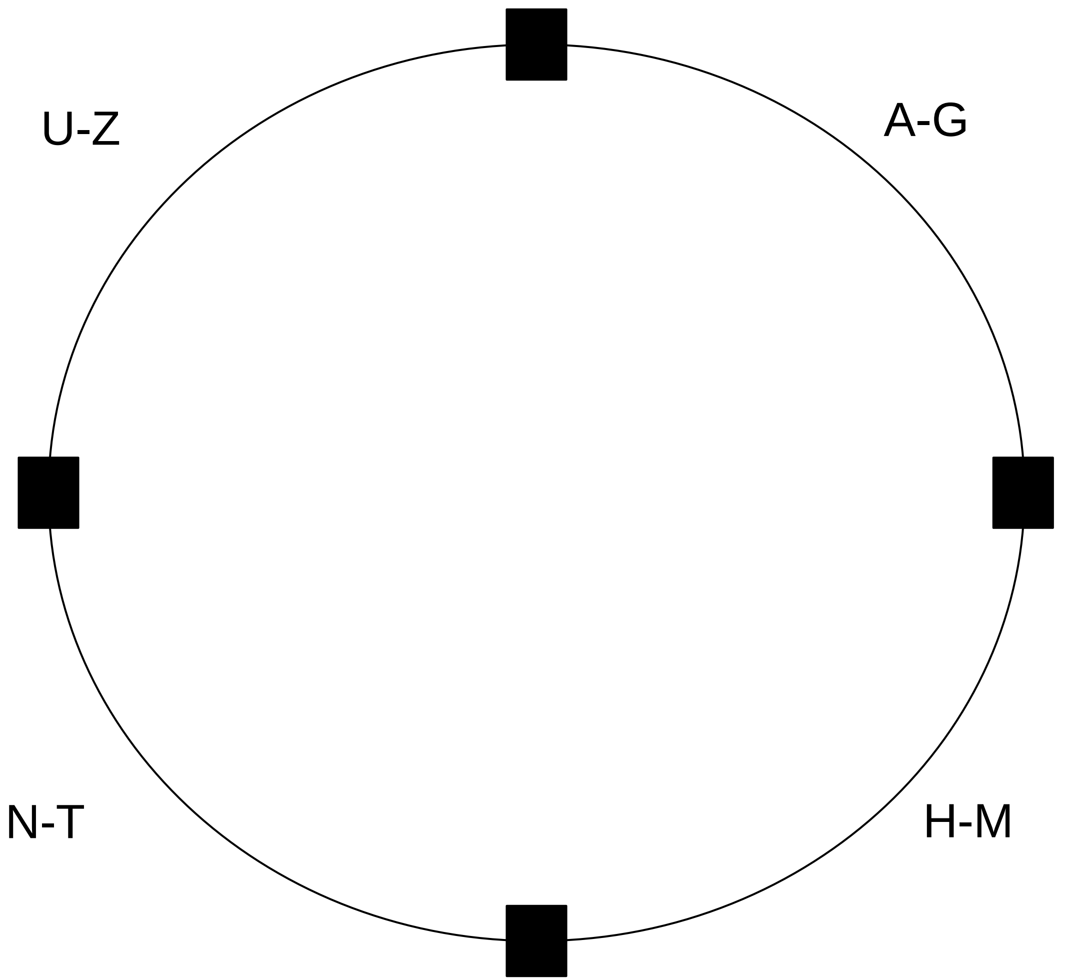
RandomPartitioner (w/o virtual nodes)

- **Uses random assignments of consistent hashing but can analyze load information on the ring**
- **Lightly loaded nodes move on the ring to alleviate heavily loaded**
  - Makes deterministic choices related to load balancing possible
  - Typical deterministic choice
    - Divide the hash-ring evenly wrt. to number of nodes
- **Need to rebalance the cluster when adding removing nodes**

ByteOrderedPartitioner

- **Departs more significantly from classical consistent hashing**
- **There is still a ring**
  - Keys are ordered lexicographically along the ring by their value
    - In contrast to ordering by hash
- **Pros**
  - ensures that row keys are stored in sorted order
  - allows range scans over rows (as if scanning with a RDBMs cursor)
- **Cons?**

ByteOrderedPartitioner (illustration)



ByteOrderedPartitioner (cons)

- **Bad for load balancing**
  - Hot spots
- **Might improve performance for specific load**
  - But one can have a similar effect to range row scans using column family indexes
- **Typically, RandomPartitioner is strongly preferred**
  - Better load balancing, scalability

Partitioning w. virtual nodes (V1.2)

- **No hash-based tokens**
  - Randomized vnode assignment
- **Easier cluster rebalancing when adding/removing nodes**
- **Rebuilding a failed node is faster (Why?)**
- **Improves the use of heterogeneous machines in a cluster (Why?)**
- **Typical number 256 vnodes**
  - older machine (2x less powerfull) – use 2x less nodes

We will cover

- **Data partitioning**
- **Replication ←**
- **Data Model**
- **Handling read and write requests**
- **Consistency**

Replication

- **In principle, again similar to Dynamo**
  - Walk down the ring and choose N-1 successor nodes as replicas (preference list)
- **2 main replication strategies**
  - SimpleStrategy
  - NetworkTopologyStrategy
- **NetworkTolopogyStrategy**
  - With multiple, geographically distributed datacenters, and/or
  - To leverage information about how nodes are grouped within a single datacenter

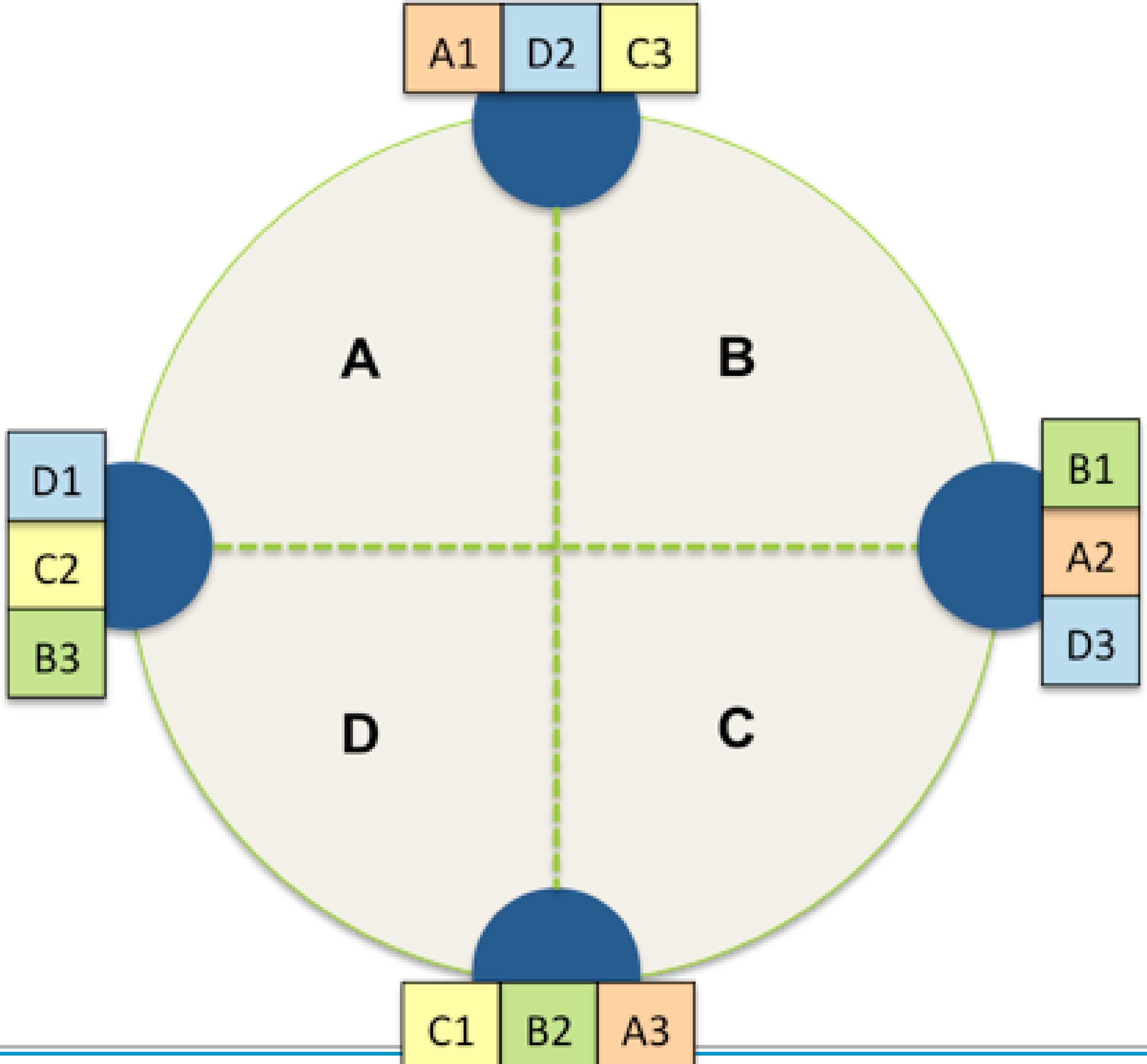
SimpleStrategy (aka Rack Unaware)

- **Node responsible for a key (wrt. Partitioning) is called the main replica (aka coordinator in Dynamo)**

- **Additional N-1 replicas are placed on the successor nodes clockwise in the ring without considering rack or datacenter location**

- **Main replica and N-1 additional ones form a preference list**

SimpleStrategy (aka Rack Unaware)



NetworkTopologyStrategy

- **Evolved from original Facebook's "Rack Aware" and "Datacenter Aware" strategies**
- **Allows better performance when Cassandra admin is given knowledge of the underlying network/datacenter topology**
- **Replication guideliness**
  - Reads should be served locally
  - Consider failure scenarios

NetworkTopologyStrategy

- **Replica placement is determined independently within each datacenter**
- **Within a datacenter:**
- **1) First replica → main replica (coordinator in Dynamo)**
- **2)Additional replicas**
  - walk the ring clockwise until a node in a different rack from the previous replica is found (Why?)
  - If there is no such node, additional replicas will be placed in the same rack

NetworkTopologyStrategy

Racks in a Datacenter

NetworkTopologyStrategy

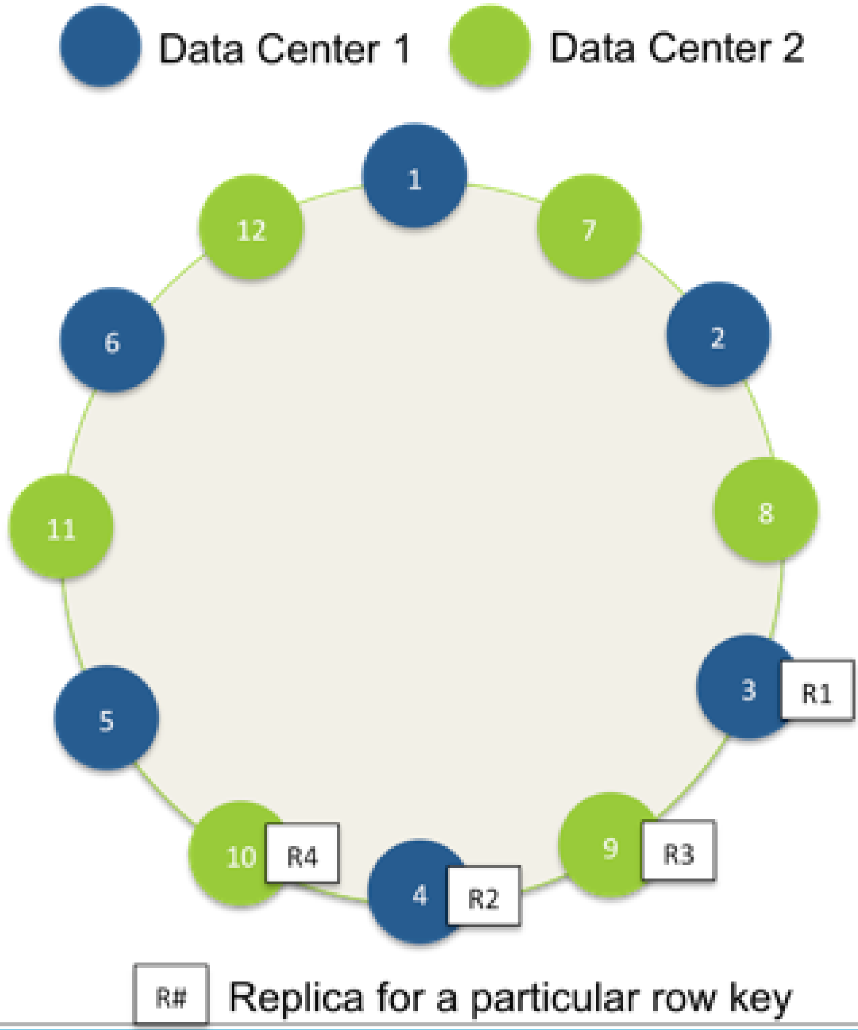
- **With multiple datacenters**

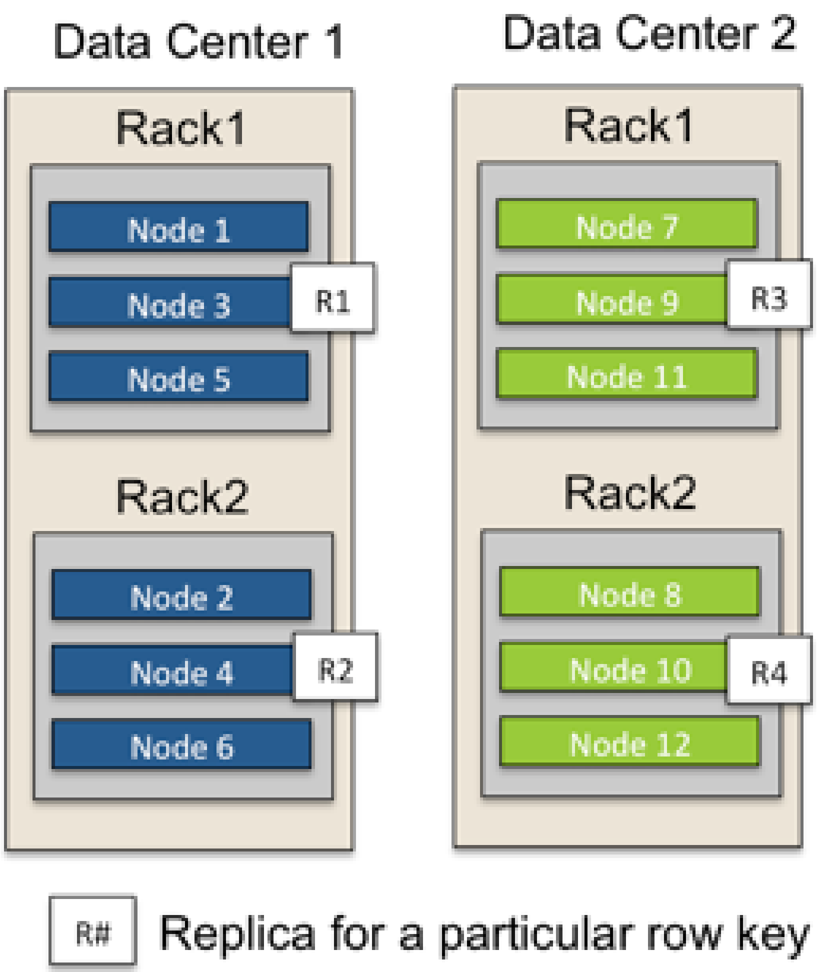
- **Repeat the procedure for each datacenter**
  - Instead of a coordinator the first replica in the “other” datacenter is the closest successor of the main replica (again, walking down the ring)

- **Can choose**
  - Number of replicas (total)
  - Number of replicas per datacenter (can be assymetric)

NetworkTopologyStrategy (example)

N=4, 2 replicas per datacenter (2 datacenters)





EURECOM

Alternative replication schemes

- **3 replicas per datacenter**

- **Assymetrical replication groupings, e.g.,**
  - 3 replicas per datacenter for real-time apps
  - 1 replica per datacenter for running analytics

Impact on partitioning

- **With partitioning and placement as described so far**
  - could end up with nodes in a given data center that own a disproportionate number of row keys
  - Partitioning is balanced across the entire system, but not necessarily within a datacenter
- **Remedy**
  - Each data center should be partitioned as if it were its own distinct ring

NetworkTopologyStrategy

- **Network information provided by Snitches**
  - a configurable component of a Cassandra cluster used to define how the nodes are grouped together within the overall network topology (e.g., racks, datacenters)
  - SimpleSnitch, RackInferringSnitch, PropertyFileSnitch, GossipingPropertyFileSnitch, EC2Snitch, EC2MultiRegionSnitch, Dynamic Snitching, …

- **In production, may also leverage Zookeeper coordination service**
  - Can also ensure no node is responsible for replicating more than N ranges

Snitches

- **Give Cassandra information about network topology for efficient routing**
- **Allow Cassandra to distribute replicas by grouping machines into datacenters and racks**
- **SimpleSnitch**
  - default
  - Does not recognize datacenter/rack information
  - Used for single-datacenter deployments or single-zone in public clouds

Snitches (cont’d)

- **RackInferringSnitch (RIS)**
  - Determines the location of nodes by datacenter and rack from the IP address (2nd and 3rd octet respectively)
  - 4th octet – node octet
  - 100.101.102.103
- **PropertyFileSnitch (PFS)**
  - Like RIS, except that it uses user-defined description of the network details located in the cassandra-topology.properties f
  - Can be used when IPs are not uniform (see RIS)

Snitches (cont'd)

- **GossipingPropertyFileSnitch**
  - uses gossip for propagating PFS information to other nodes.
- **EC2Snitch (EC2S)**
  - for simple cluster deployments on Amazon EC2 where all nodes in the cluster are within a single region.
  - With RIS in mind
    - an EC2 region is treated as the data center and the availability zones are treated as racks within the data center.
    - Example, if a node is in us-east-1a, us-east is the data center name and 1a is the rack location.

Snitches (cont'd)

- **EC2MultiRegionSnitch**
  - for deployments on Amazon EC2 where the cluster spans multiple regions
  - Like with EC2S, regions are treated as datacenters and availability zones are treated as racks within a data center.
  - uses public IPs as broadcast_address to allow cross-region connectivity.
- **Dynamic Snitching**
  - By default, all snitches also use a dynamic snitch layer that monitors read latency and, when possible, routes requests away from poorly-performing nodes.

We will cover

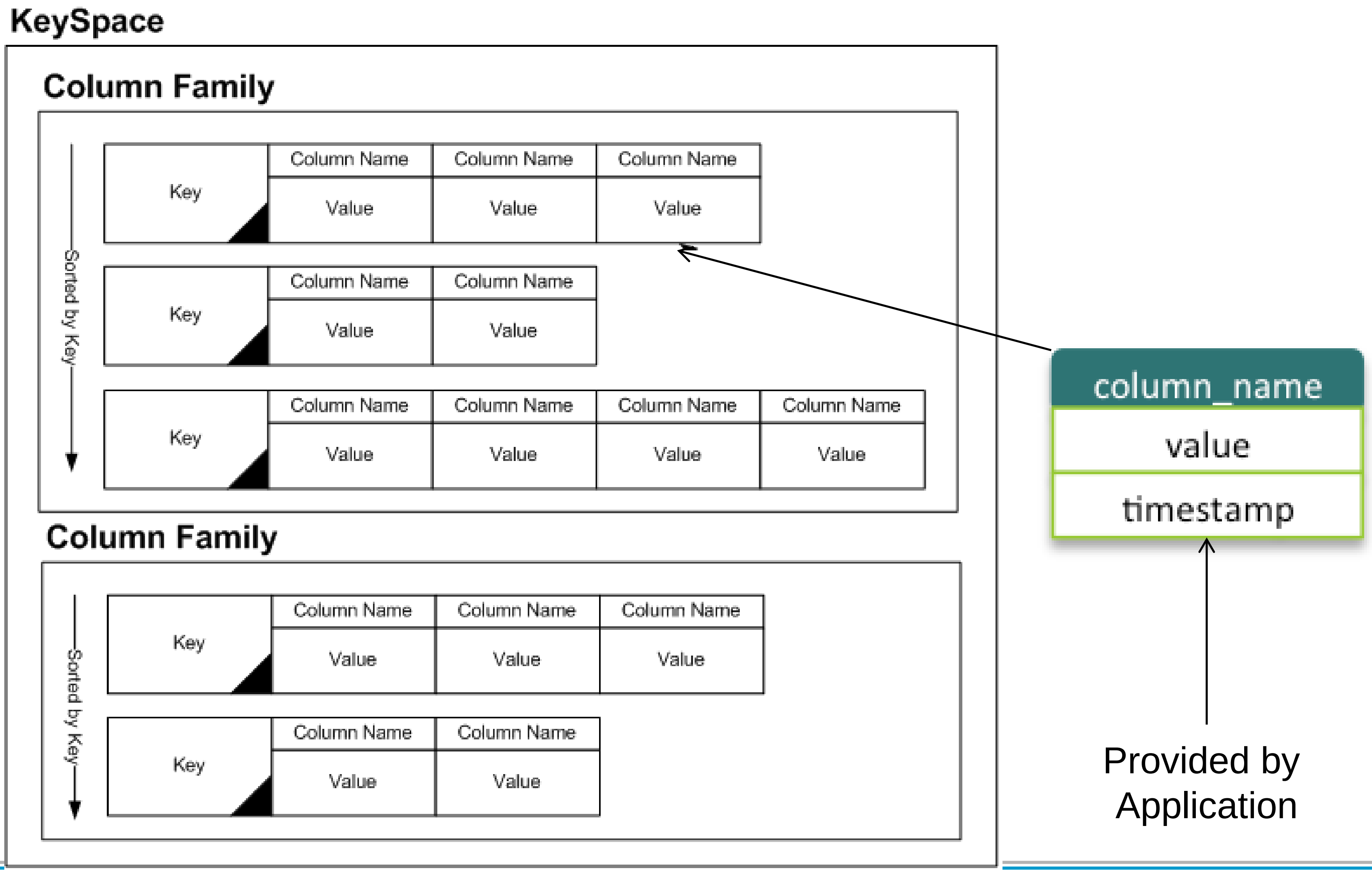
- Data partitioning
- Replication
- Data Model ←
- Handling read and write requests
- Consistency

Data Model

- Of an HBase
- Grouping by column families
- Not required to have all columns

- Review the data model of HBase

Data Model



KeySpace
Column Family
Sorted by Key
Key
Column Name
Value
Column Family
column_name
value
timestamp
Provided by Application

Data Model: Special Columns

- **Counter, Expiring and Super columns**
- **Counter columns**
  - Used to store a number that incrementally counts the occurrences of a particular event or process (e.g., no. of page hits)
  - No application timestamp needed
  - Current release of Cassandra relies on node generated timestamps to deduce precedence relations (must use NTP)

Data Model: Special Columns

- **Expiring columns**
  - Have a TTL (in secs), tombstone after expiration
- **Super columns**
  - Column family can contain either regular columns or *super columns*,
    - another level of nesting to the regular column family structure
  - Used to group multiple columns based on a common lookup value
    - e.g., home address super column, grouping “street”, “city”, “ZIP” columns
  - No timestamp (columns in a Super column may have timestamps)

We will cover

- Data partitioning
- Replication
- Data Model
- Handling read and write requests ←
- Consistency

Handling client's requests

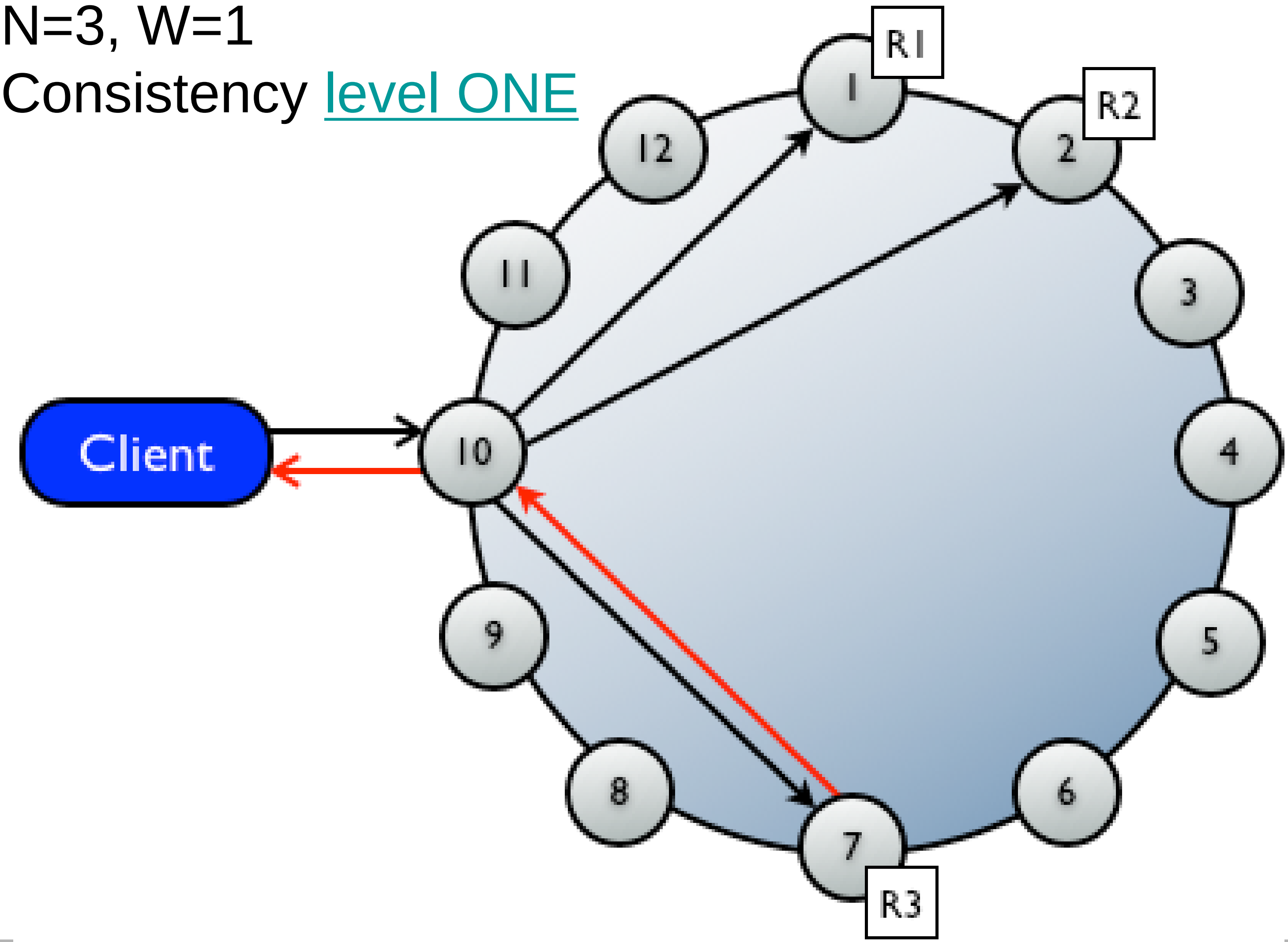
- **Similar to Dynamo**
- **A read/write request for a key gets routed to any node in the Cassandra cluster**
  - The node serves as a *proxy*
  - Does not have to route to the main replica
  - Proxy (called coordinator in Cassandra parlance) handles the interaction between a client and Cassandra
- **The proxy first determines the replicas for this particular**
  - Depending on partitioning and placement strategies
  - Zookeeper may reveal very useful

Write requests

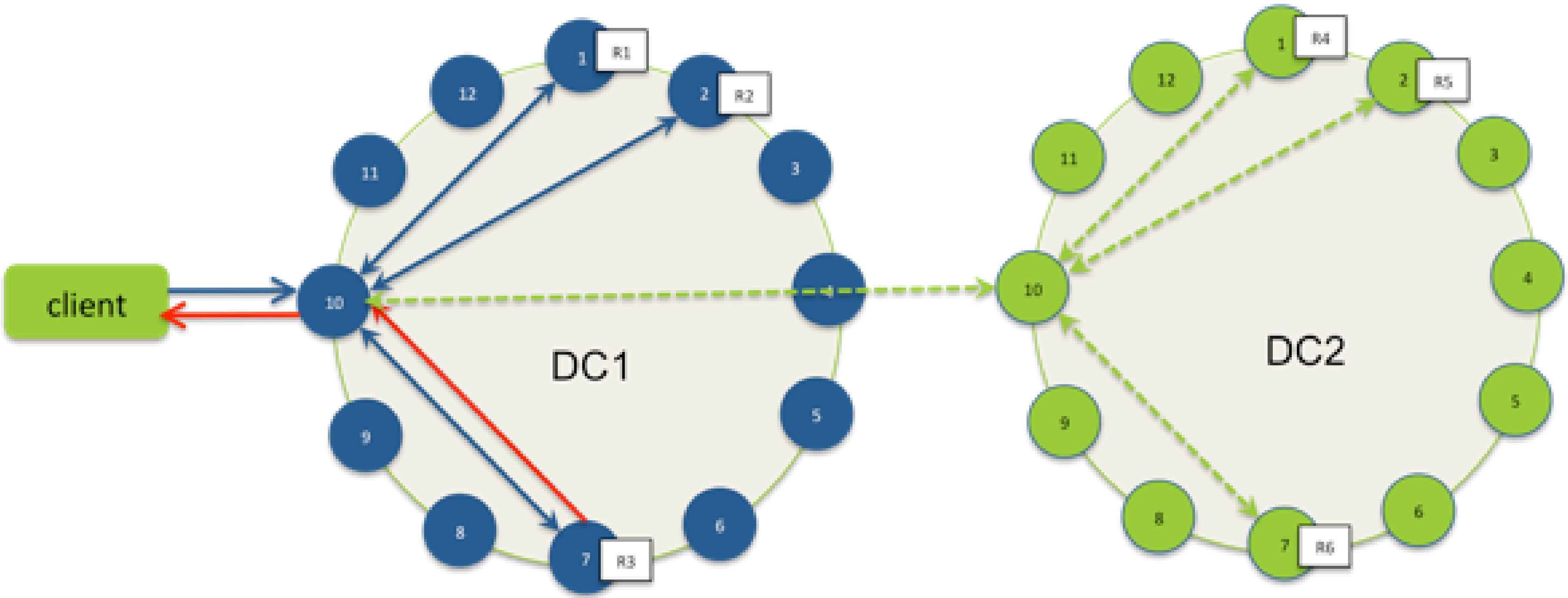
- **The proxy sends the write to *all* N replicas**
  - Regardless of the consistency level (discussed a bit later)

Write requests (single datacenter)

N=3, W=1
Consistency level ONE



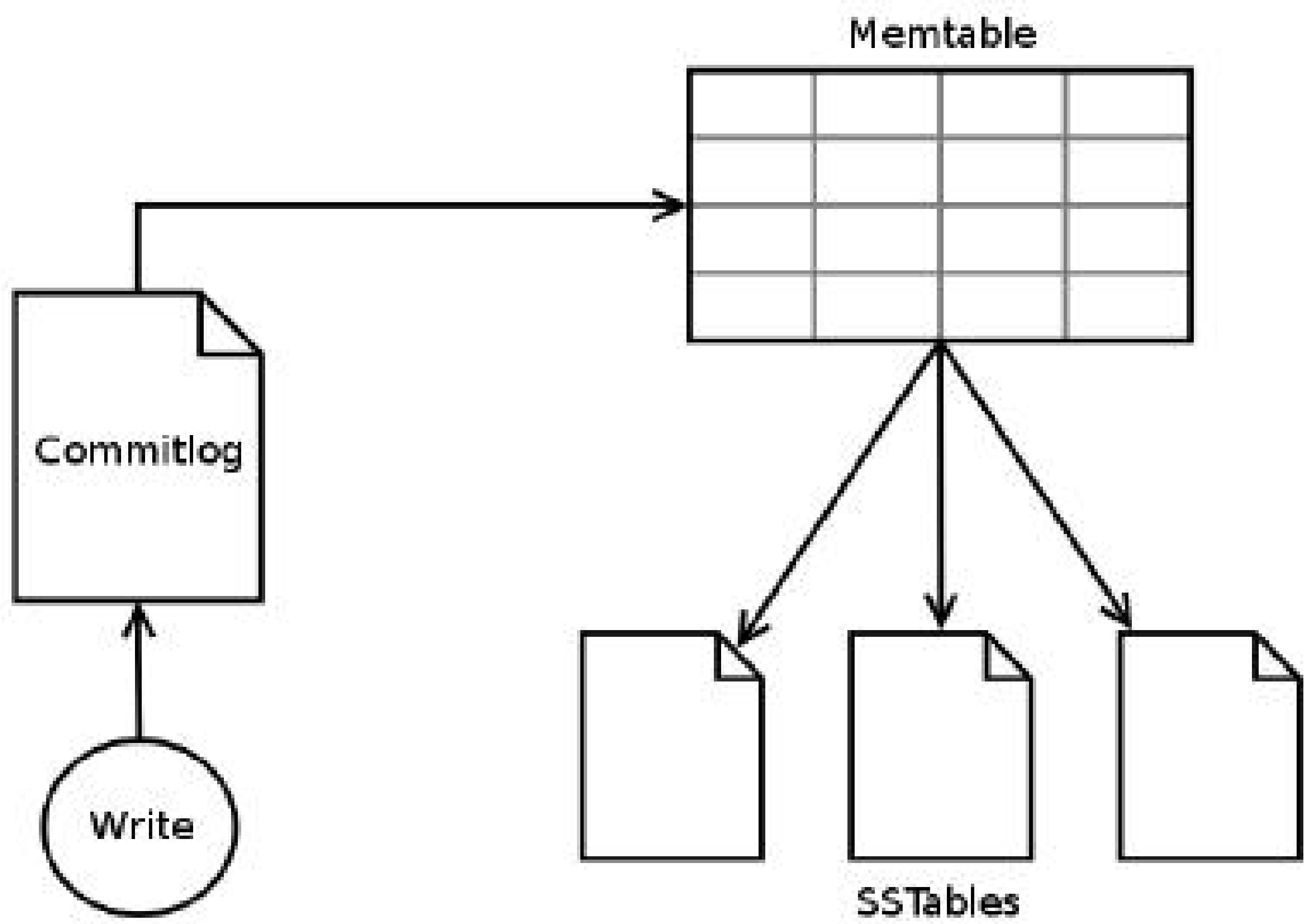
Write requests across multiple datacenters



Write requests (local processing)

- When a replica receives a write request it processes the request much like Hbase does
- 1) Write to the commit log
- 2) Write to in memory data structure (memtable)
- 3) At this point write is (locally) deemed successful
- 4) Writes are batched in memtable and periodically flushed to disk to a persistent table structure called an *SSTable* (sorted string table)

Write requests (local processing)



Write requests (local processing)

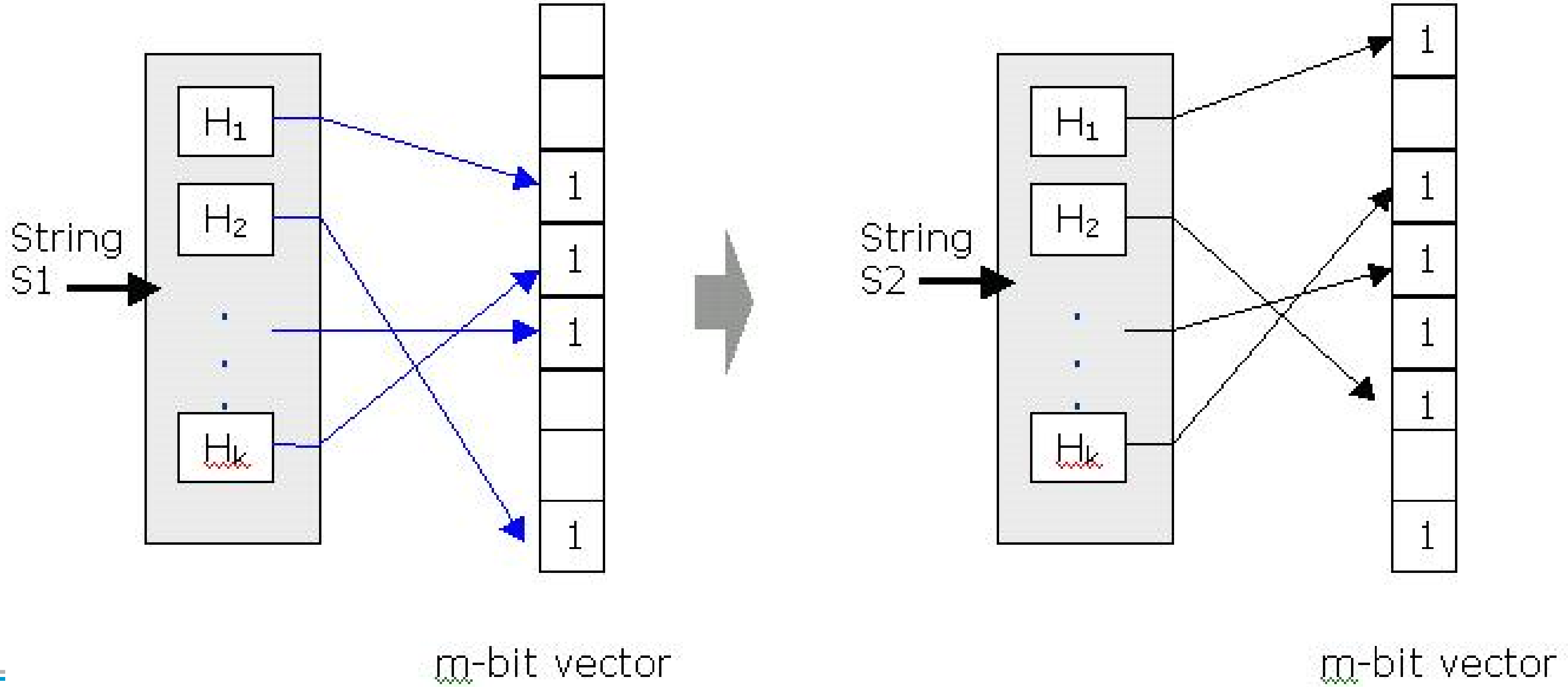
- **Memtables**
  - organized in sorted order by row key
  - flushed to SSTables sequentially (no random seeking as in relational databases)
- **SSTables**
  - immutable (no rewrite after they have been flushed)
  - Implies that a row is typically stored in many SSTables
  - At read time, a row must be combined from all SSTables on disk (as well as unflushed memtables) to produce the requested data
  - To optimize this combining process, Cassandra uses an in-memory structure called a *bloom filter*

Bloom filters

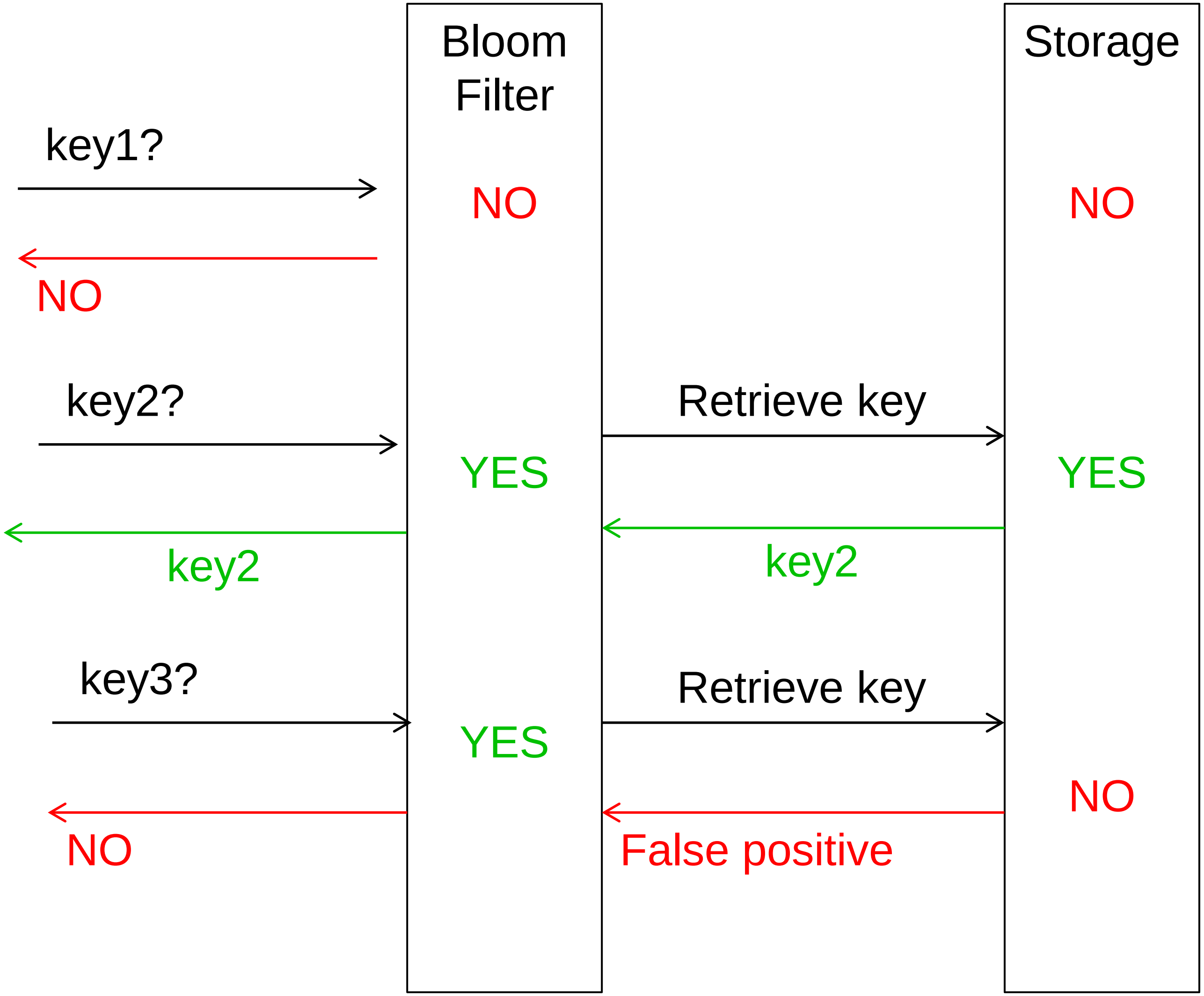
- **One for each SSTable**
  - Used in combining from row data from multiple SSTables, memtable
  - Used to check if a requested row key exists in the SSTable before doing any disk seeks
- **Bloom filters used to test whether element is in a set or not**
  - False negatives not possible
  - False positives are possible (consequences?)

Bloom filters

- **k hash functions hashing into the same m-bit space**
- **Query: if any of the hashes is 0, the element is certainly not in the set**



Bloom filters



Read requests

- **The number of replicas contacted in read depends on the chosen consistency level. E.g.,**
  - Proxy routes the requests to the closest replica or
  - Proxy routes the requests to all replicas and waits for a quorum of responses,
  - …
- **Like in Dynamo**
  - Proxy will initiate read repair (aka writeback) if it detects inconsistent replicas
  - This is done in background, after the read has been returned to the client

Read requests (local processing)

- **Upon a node receives the read request**
  - row must be combined from all SSTables on that node that contain columns from the row in question
  - as well as from any unflushed memtables
- **This produces the requested data**
- **Key techniques for better performance**
  - row-level column index
  - Bloom filters (as described earlier)

Read performance

- **As described so far, Cassandra may have higher read latency than RDBMSs**
  - Not because of SSTables inherently
  - But because of combining from multiple SSTables
    - An intution of a typical average: 2-4 SSTables to be combined

- **Solution**
  - Read cache (in memory)
  - Have to be careful with consistency implications, invalidation, etc.
  - Not going into details here

We will cover

- Data partitioning
- Replication
- Data Model
- Handling read and write requests
- Consistency ←

Tunable consistency

- **Consistency in Cassandra is tunable**
  - Hence is the availability (per CAP)
- **N replicas in the preference list**
- **Write requests: all N replicas are contacted**
  - Write ends when W respond
- **Read requests: R replicas are contacted**
  - This is done optimistically, may need to contact all N
- **Choices of W and R define consistency level**
  - Dynamo: W+R>N (notice the extended preference lists in Dynamo, sloppy quorums)
  - Cassandra: W+R>N not mandatory

Consistency levels

- **ONE**
  - W=1
    - one replica must write to commit log and memtable
  - R=1
    - Returns a response from the closest replica (as determined by the snitch).
    - By default, a read repair runs in the background to make the other replicas consistent.
  - Regardless of N!

Consistency levels

- **QUORUM**
  - W=floor(N/2+1) (a majority)
    - A write must be written to the commit log and memory table on a quorum of W replicas.
  - R=floor(N/2+1) (a majority)
    - Read returns the record with the most recent timestamp once a quorum of R replicas has responded.
    - Notice that the timestamp is application timestamp
- **LOCAL_QUORUM**
  - Restricts QUORUM approach to the proxy’s datacenter
- **EACH_QUORUM**
  - QUORUM invariants must be satisfied for each datacenter individually

Consistency levels

- **ALL**
  - W=N
    - Must complete the write at all nodes in the cluster
  - R=N
    - Read returns the record with the most recent timestamp once all replicas respond
- **ANY**
  - Additional consistency for writes
  - Allows writes to complete even if all N replicas in the preference list are down
    - e.g., a replica responsible for hinted handoff might handle the write
    - Such a write will be unreadable until repair of a replica in a preference list

Tunability

- **Can choose Read consistency and Write consistency**
  - independently from each other
  - on fly!

- **SELECT * FROM users WHERE dept=‘06' USING CONSISTENCY QUORUM;**

- **It is the responsibility of application to mind the consistency consequences**

We will cover

- **Data partitioning**
- **Replication**
- **Data Model**
- **Handling read and write requests**
- **Consistency**
- **Many more aspects**
  - Hinted handoff, background gossiping, anti-entropy,…
    - ☞Along the lines of Amazon Dynamo
  - Compaction, deletion,…
    - ☞Along the lines of HBase

Further reading (recommended)

Avinash Lakshman, Prashant Malik: Cassandra: a decentralized structured storage system. Operating Systems Review 44(2): 35-40 (2010)

Apache Cassandra 1.2 Documentation. Datastax.

http://www.datastax.com/docs/1.2/index

Further Reading (optional)

- **Eben Hewitt: Cassandra: The definitive Guide. O’Reilly. (2010)**
  - Useful reading about Apache Cassandra, get obsolete quickly as the code base progresses
  - Pdf link on http://bit.ly/JHwwR6
- **Edward Capriolo: Cassandra High Performance Cookbook. Packt Publishing. (2011)**
  - Apache Cassandra 0.8