Cassandra: A Decentralized Structured Storage System
A.Lakshman, P.Malik, Facebook

Presented by:

Beliz Kaleli
bkaleli@bu.edu

Vikash Sahu
vksahu@bu.edu

Paritosh Shirodkar
paritosh@bu.edu

Asutosh Patra
asupat12@bu.edu

Agenda

- Background
- Introduction
- Data Model
- Architecture
- Implementation
- Facebook Inbox Search
- Experiment on YCSB
- Conclusion

Background

- Distributed storage system developed by **Facebook**
- Cassandra system was designed to run on cheap commodity hardware and handle high write throughput while not sacrificing read efficiency.
  - Requirements-
    - Performance
    - Reliability
    - Efficiency
    - Support for continuous growth
    - Fail-friendly
  - Designed for Application-
    - Facebook's Inbox Search

COMPANIES USING CASSANDRA

What is Cassandra?

- Apache Cassandra is a highly scalable, high-performance distributed database designed to handle large amounts of data across many commodity servers
- Providing high availability with no single point of failure.
- It is a type of NoSQL database.It provides clients with a simple data model that supports dynamic control over data layout and format.

How is Cassandra different from RDBMS?

- It has a fixed schema.
- In RDBMS, a table is an array of arrays. (ROW x COLUMN)
- Database is the outermost container that contains data corresponding to an application.
- Tables are the entities of a database.
- Row is an individual record in RDBMS.
- Column represents the attributes of a relation.

- Cassandra has a flexible schema.
- In Cassandra, a table is a list of “nested key-value pairs”. (ROW x COLUMN key x COLUMN value)
- Keyspace is the outermost container that contains data corresponding to an application.
- Tables or column families are the entity of a keyspace.
- Row is a unit of replication in Cassandra
- Column is a unit storage in Cassandra

Related Work

- **Google File System**
  - Single master - simple design. Made fault tolerant with Chubby
- **Dynamo**
  - Structured overlay network with 1-hop request routing. Gossip based information sharing
- **Bigtable**
  - GFS based system (Master - Slave). The data model of cassandra is derived from Big Table.

Data Model

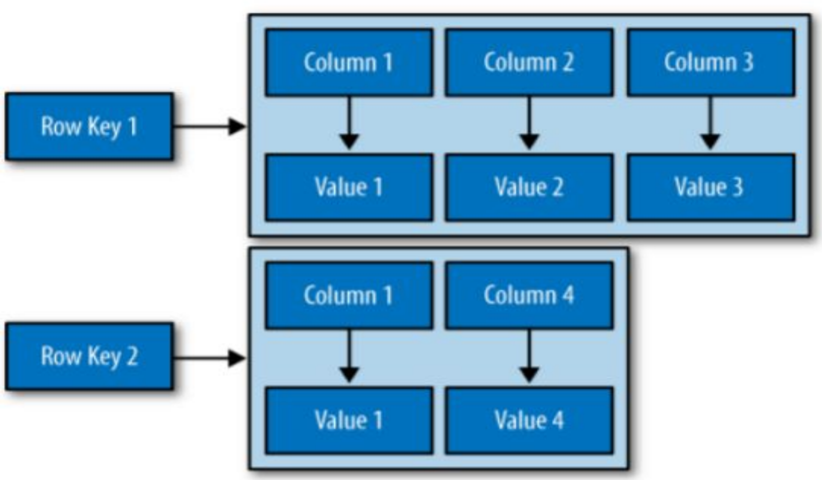
- A row in the map provides access to a set of columns which is represented by a sorted map.
  Map<RowKey, SortedMap<ColumnKey, ColumnValue>>
- Columns are grouped into column families(CF)

  Column Family - group of columns:

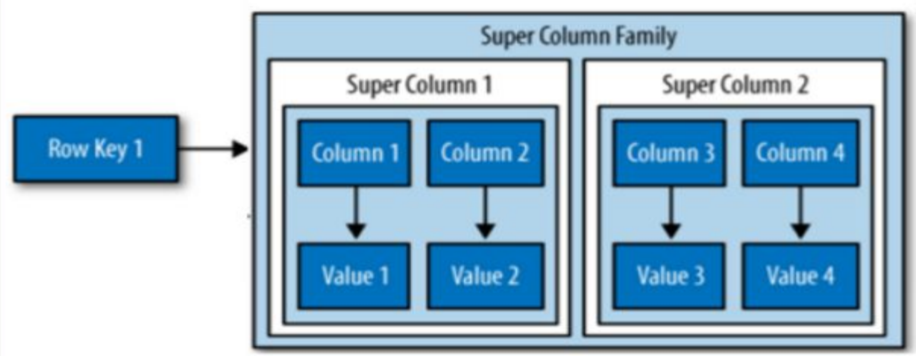
  -Simple(col name, value, timestamp)

  -Super: Family of columns(Query together)

  -Column order: Sorted by timestamp or name



Simple Column Family



Super Column Family

API Functions Provided by Cassandra

The functionalities that Cassandra API provides are :

- Insert(table, key, rowMutation)
- Get(table, key, ColumnName)
- Delete(table, key, ColumnName)

Row Mutation represents changes to one or more tables so that

1) All the tables belong to the same keyspace
2) All the changes have the same partition key. These changes are grouped into Column Family objects

To access a column in a CF;

- if Simple → *column_family : column*
- if Super → *column_family : super_column : column*

Schema Design for Cassandra

- Consider this schema for Music Playlist as an example
- Create table **MusicPlaylist** (SongId int,SongName text,Year int,Singer text,Primarykey((SongId, Year),SongName));
- In above example, table **MusicPlaylist**,
  - Songid and Year are the partition key, and
  - SongName is the clustering column.
  - Data will be clustered on the basis of SongName. In this table, each year, a new partition will be created. All the songs of the year will be on the same node. This primary key will be very useful for the data.
- Query is written in CSL(Cassandra Query Language)
- get (table, key, columnName)
  - [default@keyspace] get User['vksahu']; => (column=656d6169c, value=vksahu@bu.edu,timestamp=135225847342) cqlsh:demo> SELECT * FROM users where lastname= 'Doe';
- delete (table, key, columnName)
  - [default@keyspace] del User['vksahu'];

    row removed.

    cqlsh:demo> DELETE from users WHERE lastname = "Doe";

Core Distributed System Techniques Used in Cassandra

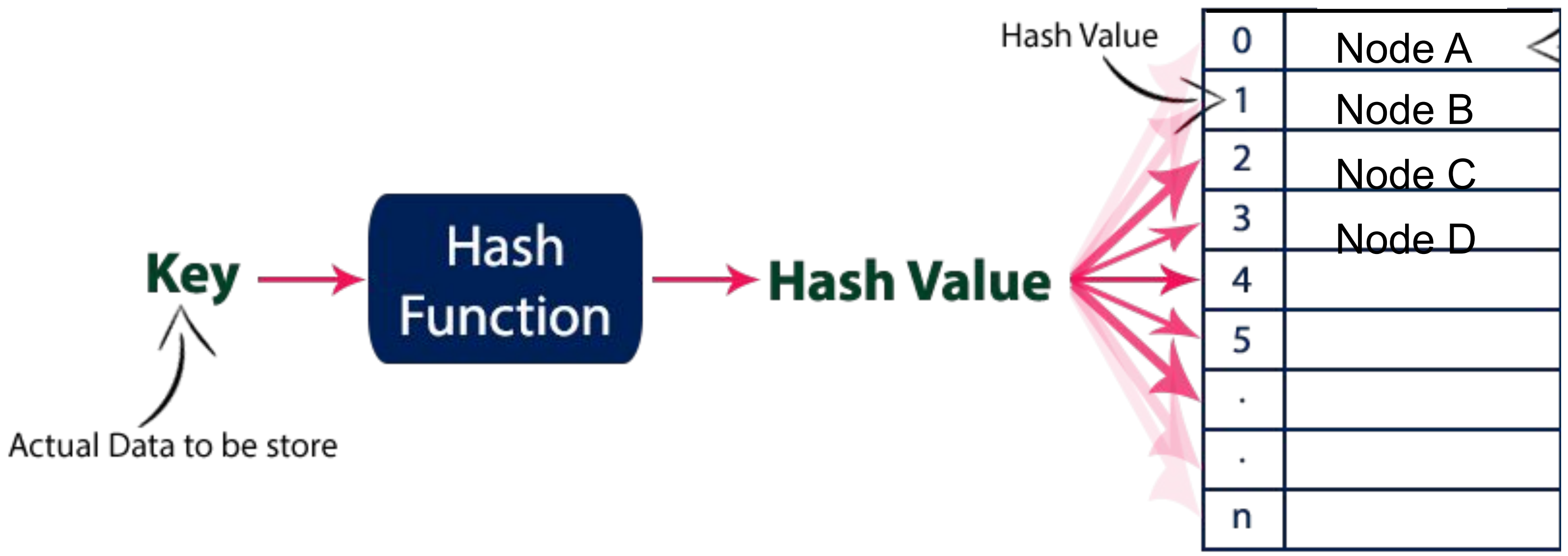
- Partitioning
- Replication
- Membership
- Bootstrapping & Scaling

work in synchrony
to handle read/write requests.

System Architecture - Partitioning

- Map incoming data to nodes
- Partition by using consistent hashing

Static Hashing:



Key	Hash Value	Node
key0	0	A
key1	1	B
key2	2	C
key3	3	D
	4	E

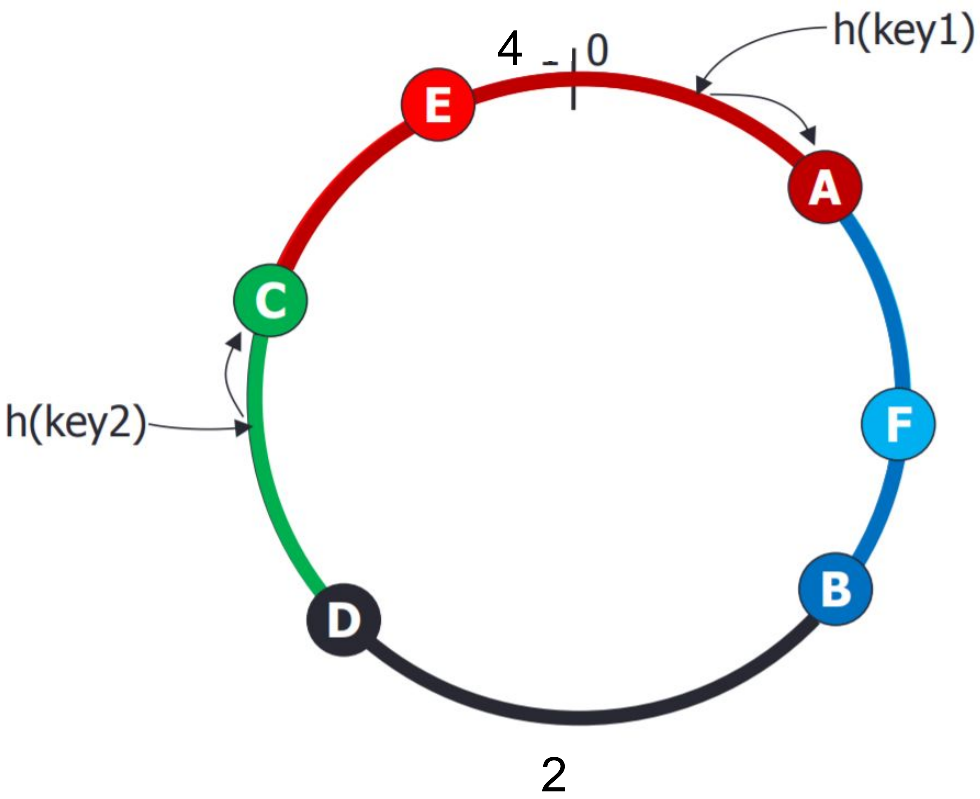
Remove node B

Key	Hash Value	Node
key0	0	A
key1	1	C
key2	2	D
key3	3	E

System Architecture - Partitioning

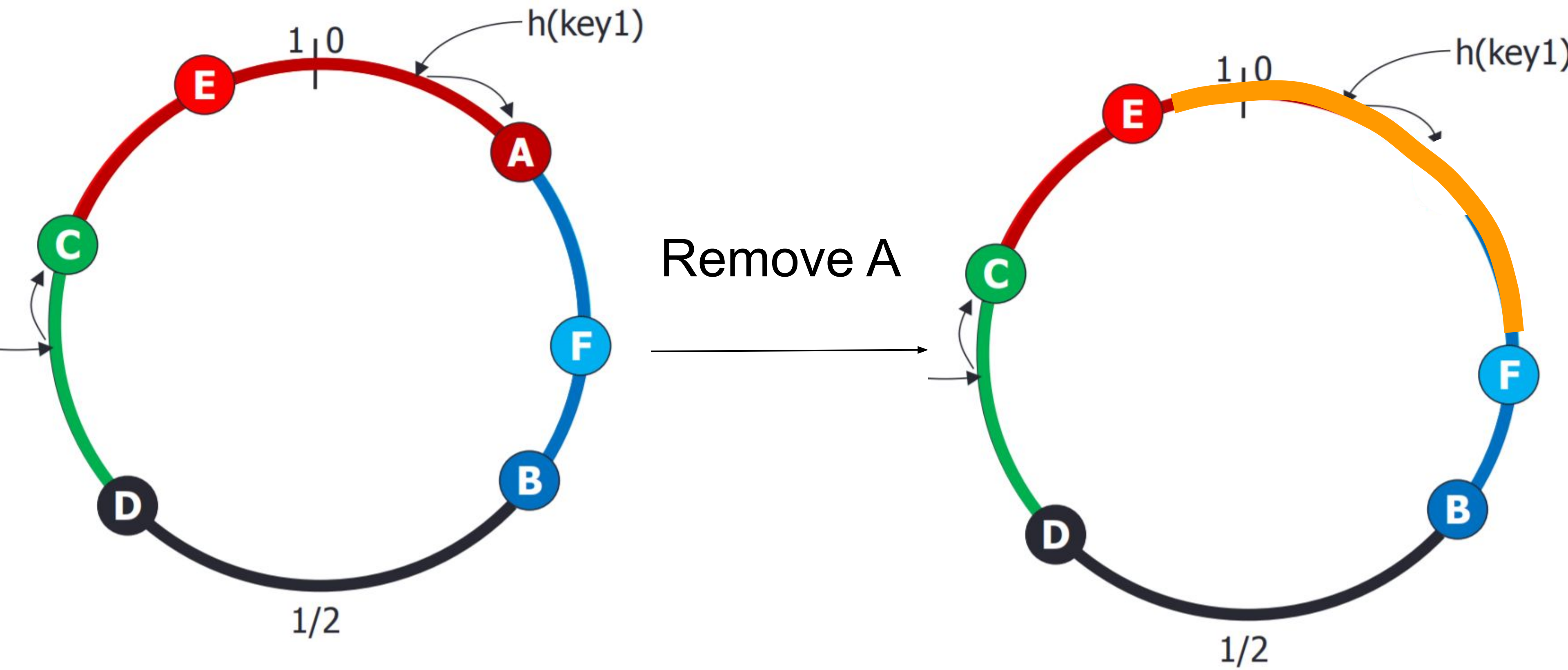
Consistent Hashing:

- Assign a value(output range: 0-4) to each node(A-E)
- Hash data item's key to find a position on ring
- Walk clockwise from that position till first node on the ring
- This node is coordinator of that key



- Application specifies the **key**.

System Architecture - Partitioning



- All of node A’s keys moved to node F

System Architecture - Partitioning (*Cont.*)

Consistent Hashing Challenges:

- Non-uniform load distribution because of the random assignment of the nodes position

- Unaware of node heterogeneity i.e. a node may have more capacity than another node, more keys should be assigned to that node

Solutions
1- Assigning nodes to multiple positions in the ring (Virtual Nodes)   2- Analyze nodes' load information and change the nodes' location   **Cassandra uses the second approach**

System Architecture - Replication

- Availability and durability
- The coordinator is in charge of replication
  - replicates to N-1 nodes in ring

- Client chooses replication policy and N

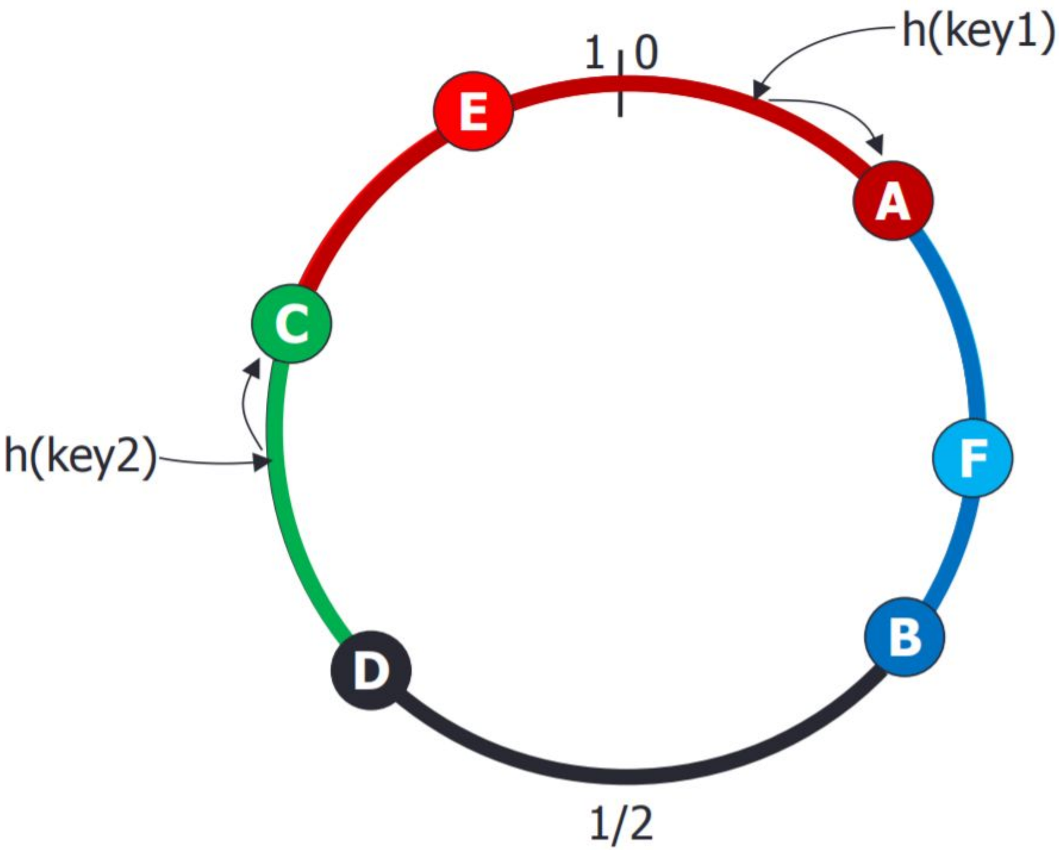
- Policies:
  - Rack Unaware
  - Rack Aware
  - Datacenter Aware

Rack Unaware

Let N = 3, key = key2

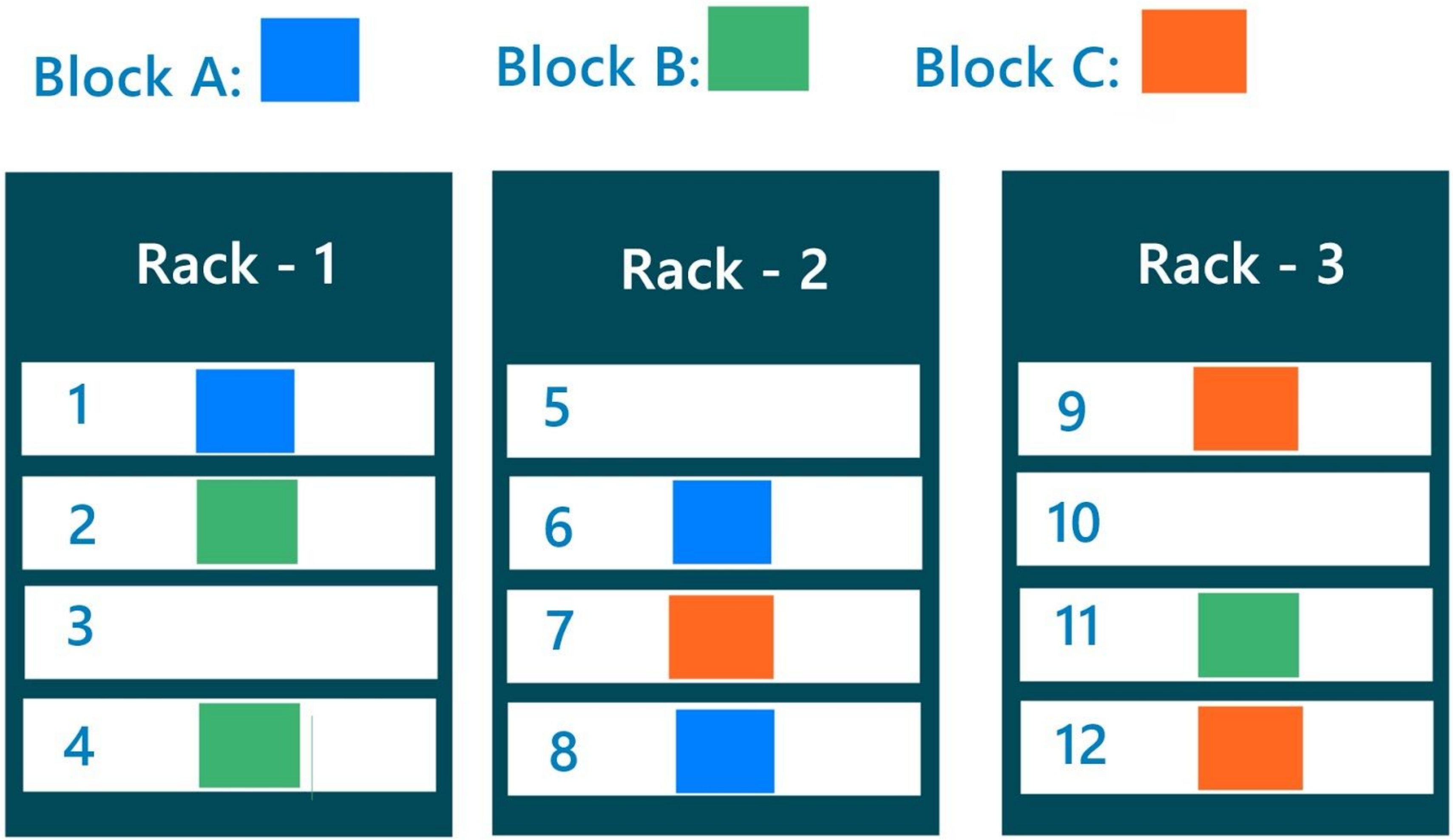
- Gets mapped to node C
- Coordinator(key2) = C
- data(key2) gets

replicated on E and A



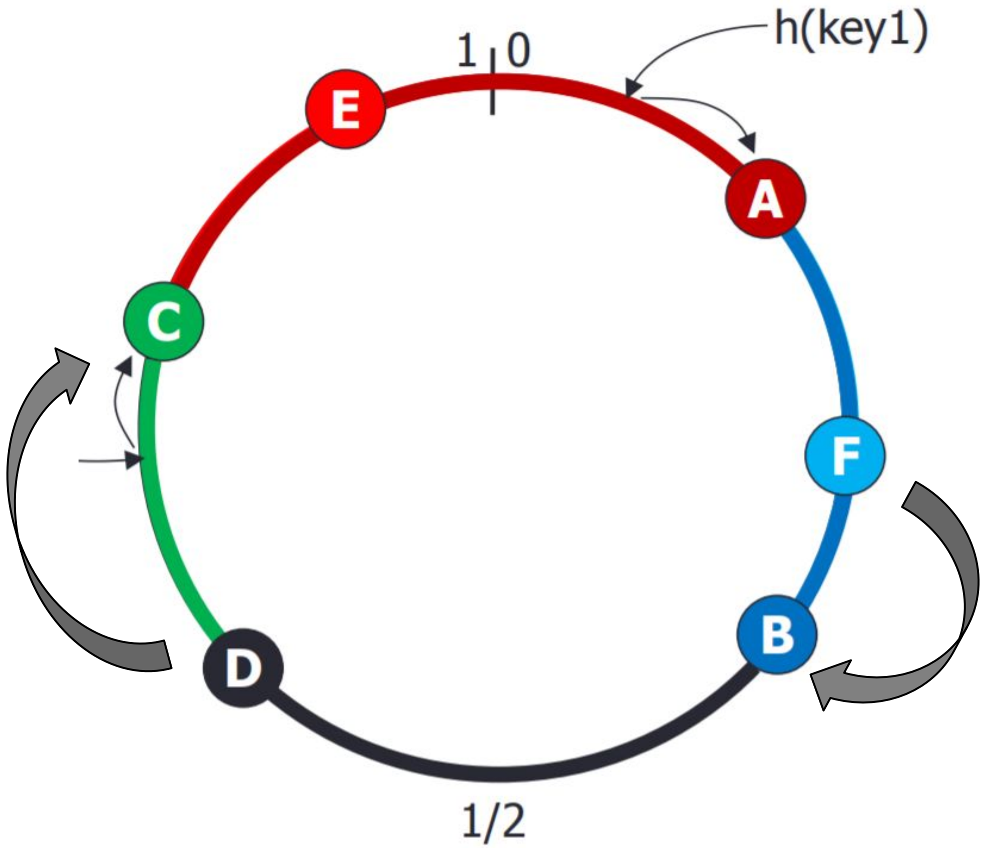
Rack Aware

- Each server group is on physically separate rack
- Protection against rack failures



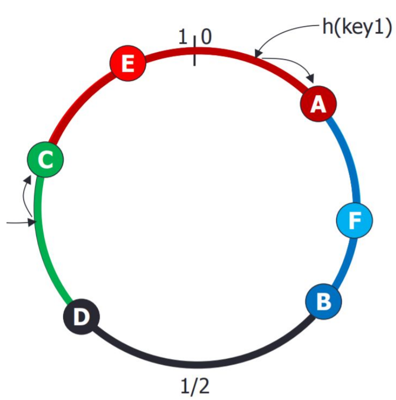
Rack Aware

- Zookeeper elects a leader, let A
- E is newly joined the cluster
- A tells E that it will hold replicas for ranges that B and C are responsible for
- Each nodes responsibility range is cached locally and inside Zookeeper

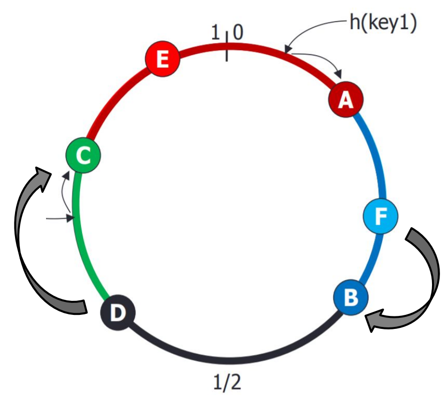


Datacenter Aware

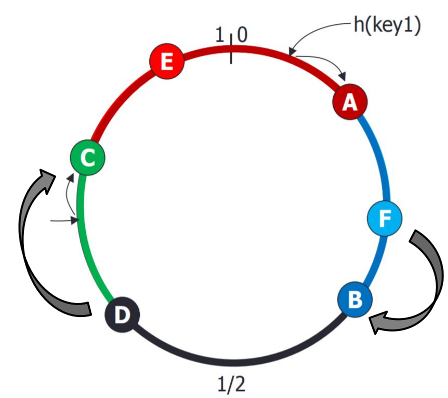
- Protection against datacenter failures.
- Same algorithm as Rack Aware
- Only difference: data is replicated across multiple datacenters
- ex: A_1 tells E_1 that it will hold replicas for ranges that B_2, C_2 and B_3, C_3 are responsible for



datacenter_1



datacenter_2



datacenter_3

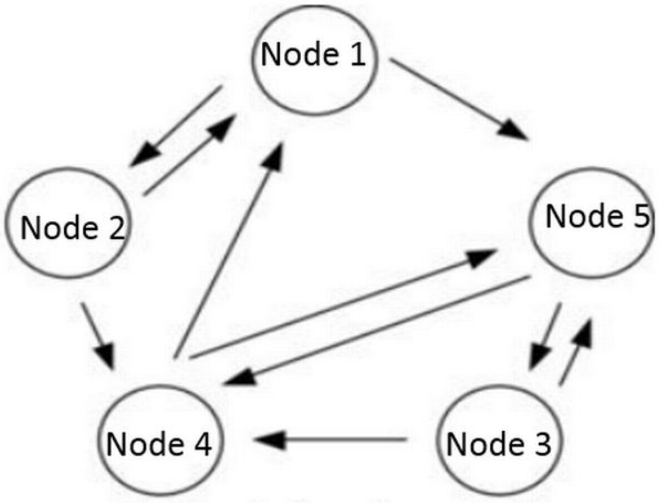
System Architecture - Membership

- Cluster membership is based on Scuttlebutt, very efficient anti-entropy Gossip based mechanism.

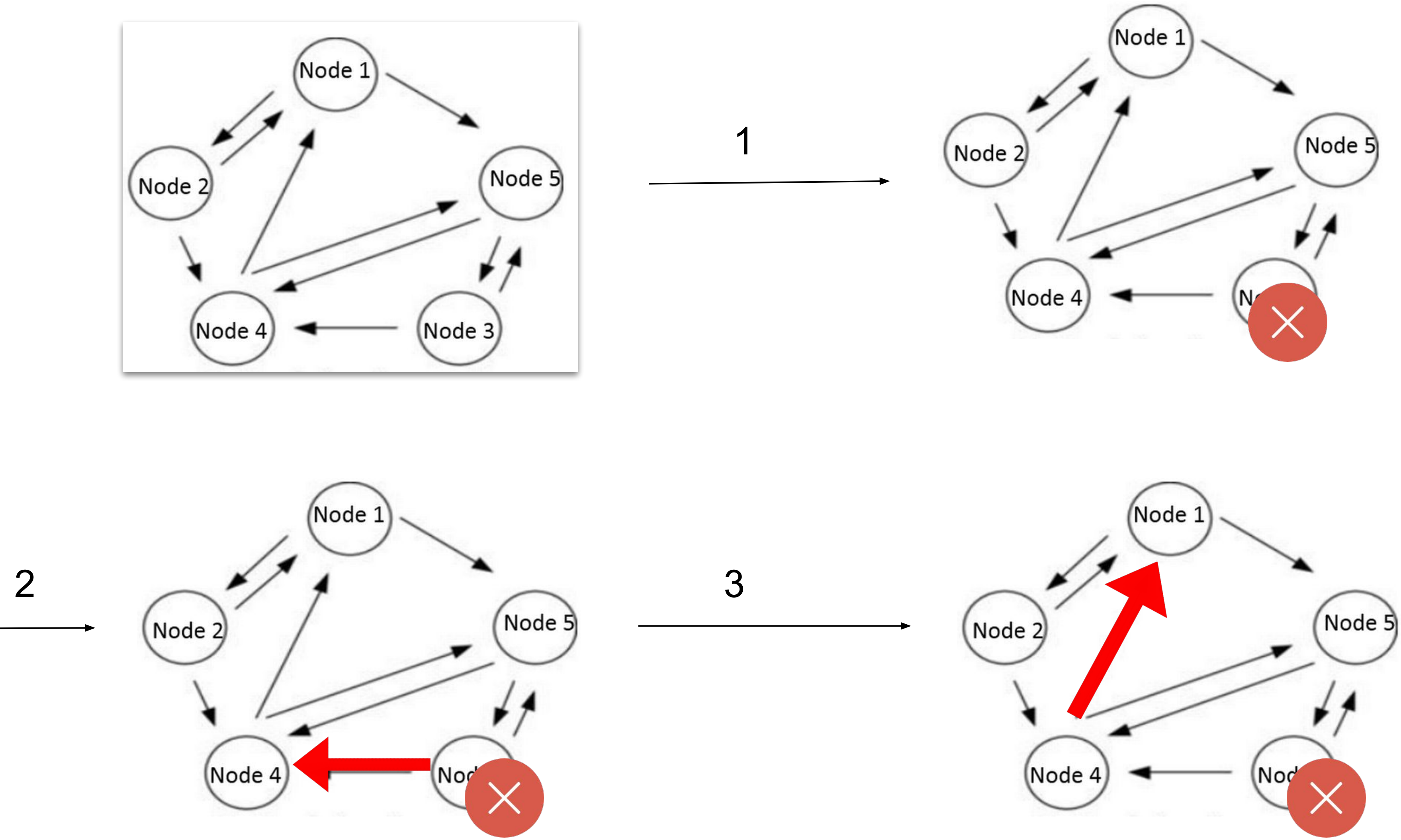
- **Gossip protocol:** Each node gossip with a random small subset of the nodes, updating its knowledge on the state and position in the ring of other nodes.

- **Anti-entropy:** Repairs information by comparing and reconciling differences

- Nodes do not exchange information with every other node in the cluster in order to reduce network load.
- Over a period of time, information about every node propagates throughout the cluster.



Gossip Example

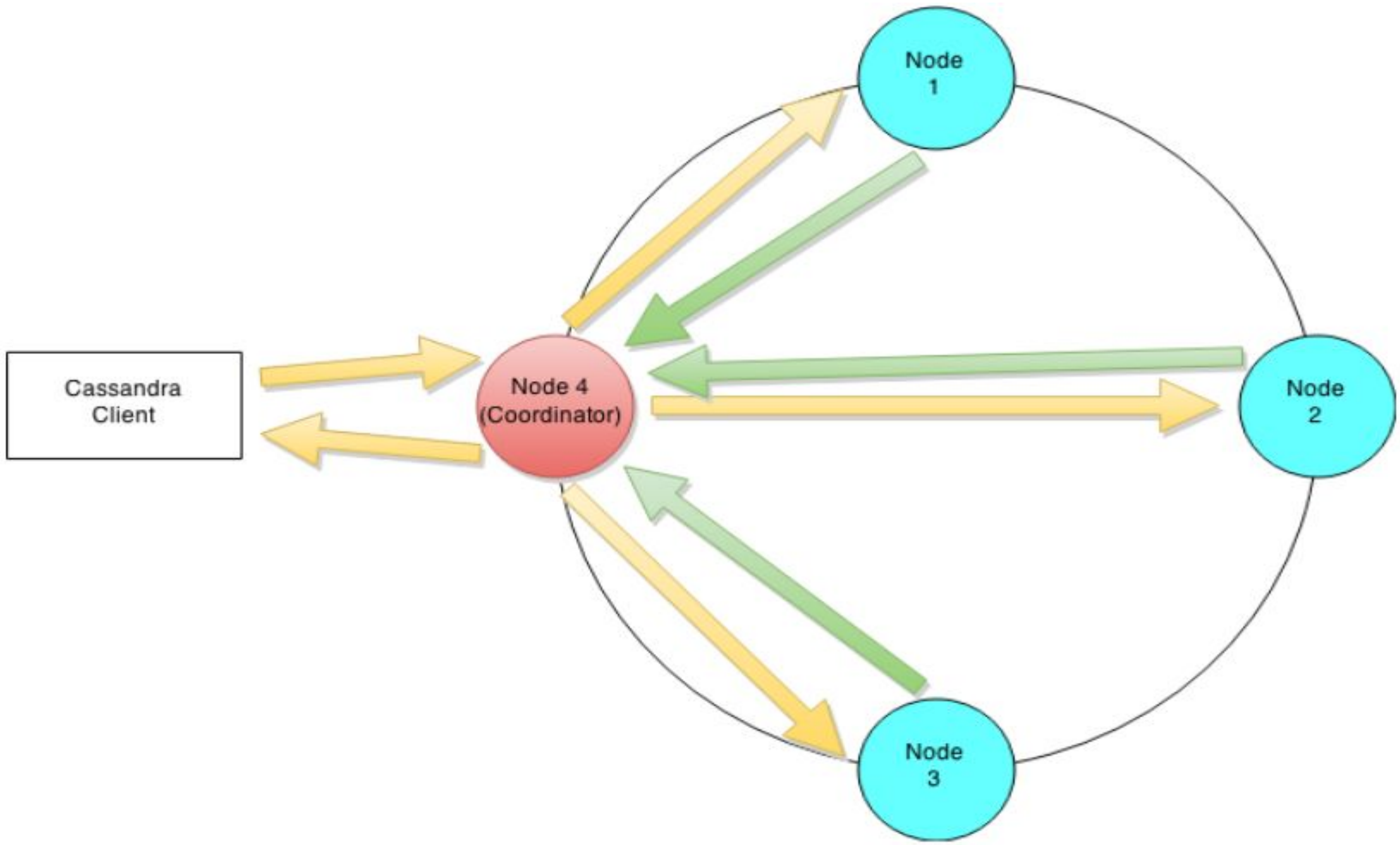


System Architecture - Bootstrapping & Scaling

- When a node starts for the first time(in cluster startup), it chooses a random position in ring
- When a new node is added into the system, it gets assigned a position in ring such that it can alleviate a heavily loaded node.
  - Data is copied from heavily loaded node to new node.
- For fault tolerance, the mapping is persisted in disk and Zookeeper
- Node's position in the ring is then gossiped around the cluster

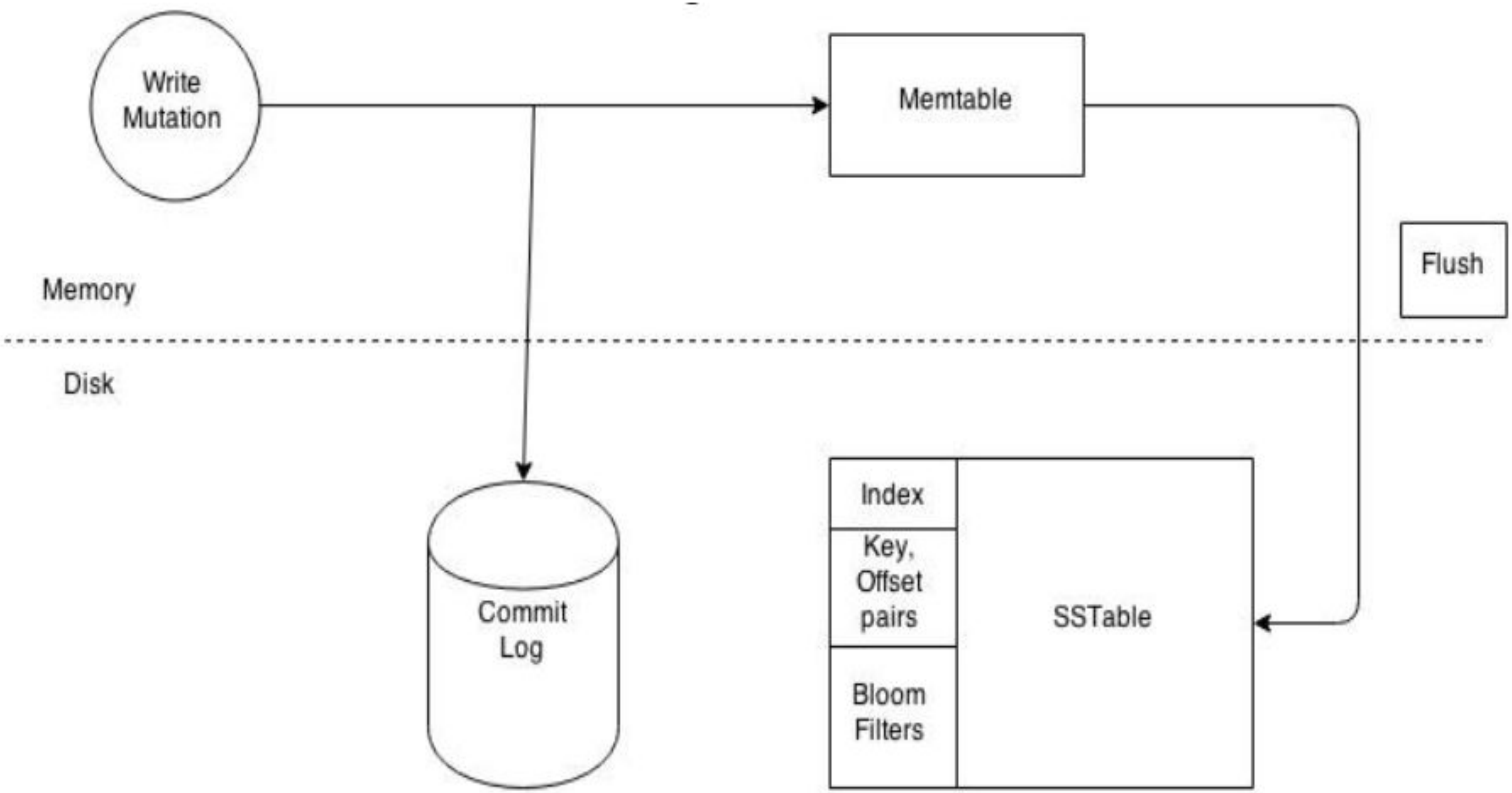
→ All nodes know about all nodes, any node can route a request to correct node!

Client interaction with Cassandra cluster



- The node that a client connects to is designated as the coordinator.
- The coordinators is responsible for satisfying the clients request.
- The consistency level determines the number of nodes that the coordinator needs to hear from in order to notify the client of a successful mutation.
- All inter-node requests are sent through a messaging service and in an asynchronous manner.
- Based on the partition key and the replication strategy used the coordinator forwards the mutation to all applicable nodes.

Write Operations at the Node level



Write Back Cache

- A write back cache is where the write operation is only directed to the cache and completion is immediately confirmed.

- This is different from Write-through cache where the write operation is directed at the cache but is only confirmed once the data is written to both the cache and the underlying storage structure.

Memtable

- A memtable is a write back cache residing in memory which has not been flushed to disk yet.

Sorted String Tables (SSTables)

- A Sorted String Table (SSTable) ordered is an immutable key value map.
- It is basically an efficient way of storing large sorted data segments in a file.

- Each node processes the request individually.
- Every node first writes the mutation to the commit log and then writes the mutation to the memtable.
- Writing to the commit log ensures durability of the write as the memtable is an in-memory structure and is only written to disk when the memtable is flushed to disk.
- A memtable is flushed to disk when:
  -- It reaches its maximum allocated size in memory.

  -- The number of minutes a memtable can stay in memory elapses.

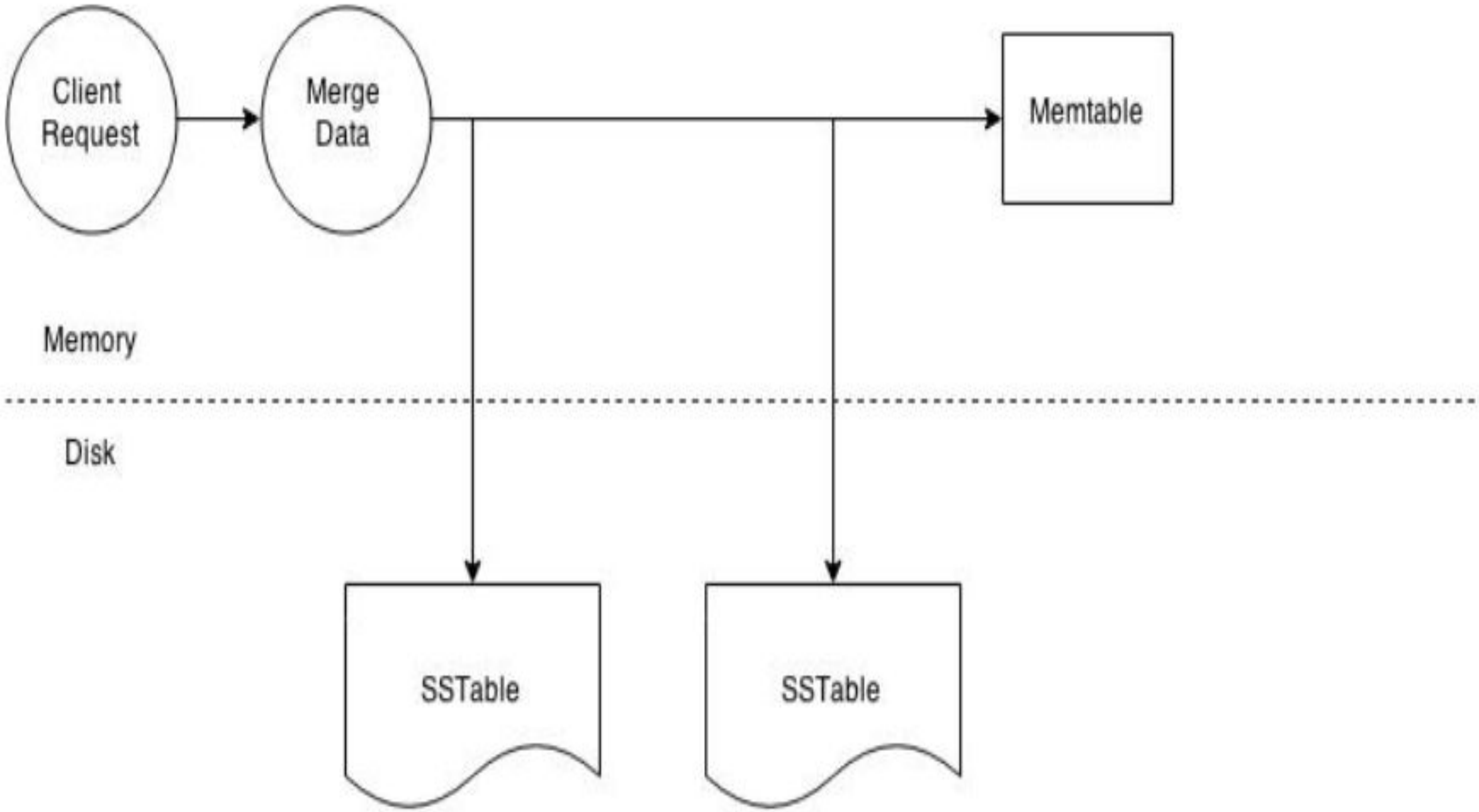
  -- Manually flushed by a user

- A memtable is flushed to an immutable structure called and SSTable (Sorted String Table).
- The commit log is used for playback purposes in case data from the memtable is lost due to node failure. For example the machine has a power outage before the memtable could get flushed.
- Every SSTable creates three files on disk which include a bloom filter, a key index and a data file.
- Over a period of time a number of SSTables are created.
- This results in the need to read multiple SSTables to satisfy a read request.
- Compaction is the process of combining SSTables so that related data can be found in a single SSTable.
- This helps with making reads much faster.

Cassandra Read Path

- At the cluster level a read operation is similar to a write operation.
- A row key must be supplied for every read operation.
- The coordinator uses the row key to determine the first replica.

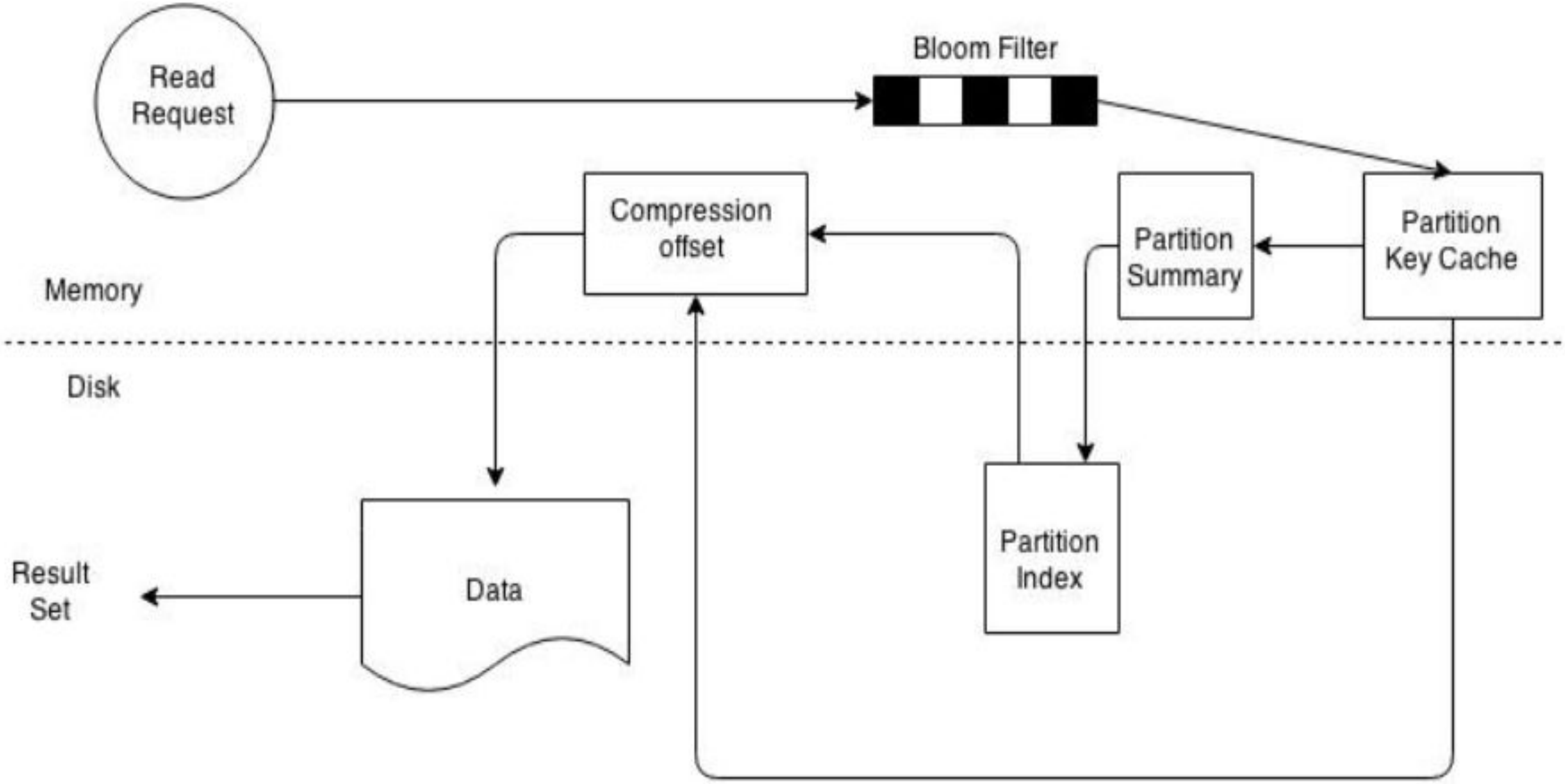
Node level read operation



- The replication strategy in conjunction with the replication factor is used to determine all other applicable replicas.
- As with the write path the consistency level determines the number of replicas that must respond before successfully returning data.
- If the contacted replicas has a different version of the data the coordinator returns the latest version to the client and issues a read repair command to the node/nodes with the older version of the data.
- The read repair operation pushes the newer version of the data to nodes with the older version.

- Every Column Family stores data in a number of SSTables.
- Thus Data for a particular row can be located in a number of SSTables and the memtable.
- Thus for every read request Cassandra needs to read data from all applicable SSTables (all SSTables for a column family) and scan the memtable for applicable data fragments.
- This data is then merged and returned to the coordinator.

SSTable Read Path



Bloom Filters

- A bloom filter is an extremely fast way to test the existence of a data structure in a set.
- A bloom filter can tell if an item might exist in a set or definitely does not exist in the set.
- False positives are possible but false negatives are not.
- Bloom filters are a good way of avoiding expensive I/O operation.

- Every SSTable has an associated bloom filter which enables it to quickly ascertain if data for the requested row key exists on the corresponding SSTable.
- This reduces IO when performing an row key lookup.
- A bloom filter is always held in memory since the whole purpose is to save disk IO.
- Cassandra also keeps a copy of the bloom filter on disk which enables it to recreate the bloom filter in memory quickly.
- If the bloom filter returns a negative response no data is returned from the particular SSTable.
- This is a common case as the compaction operation tries to group all row key related data into as few SSTables as possible.

- If the bloom filter provides a positive response the partition key cache is scanned to ascertain the compression offset for the requested row key.
- It then proceeds to fetch the compressed data on disk and returns the result set.
- If the partition cache does not contain a corresponding entry the partition key summary is scanned.
- The partition summary is a subset to the partition index and helps determine the approximate location of the index entry in the partition index.
- The partition index is then scanned to locate the compression offset which is then used to find the appropriate data on disk.

Practical Experiences

- Use of MapReduce jobs to index the inbox data
  - 7 TB data
  - Cassandra instance bottlenecked by the network bandwidth
- Atomic operations per key
- Failure detection is difficult
  - Initial: 2 minutes for a 100-node setup. Later, 15 seconds using acural detector

Facebook Inbox Search

- Per user index of all the messages
- Two search features: Two column families.
    - **Interactions**
        - Individual message identifiers are the columns.
    - **Term Search**
        - Individual message identifiers that contain the word become columns.

- 50 TB, on 150 Node cluster (East/West coast data centers).

Latency Stat	Search Interactions	Term Search
Min	7.69 ms	7.78 ms
Median	15.69 ms	18.27 ms
Max	26.13 ms	44.41 ms

Comparison using YCSB

- YCSB is **Yahoo Cloud Server Benchmarking framework**

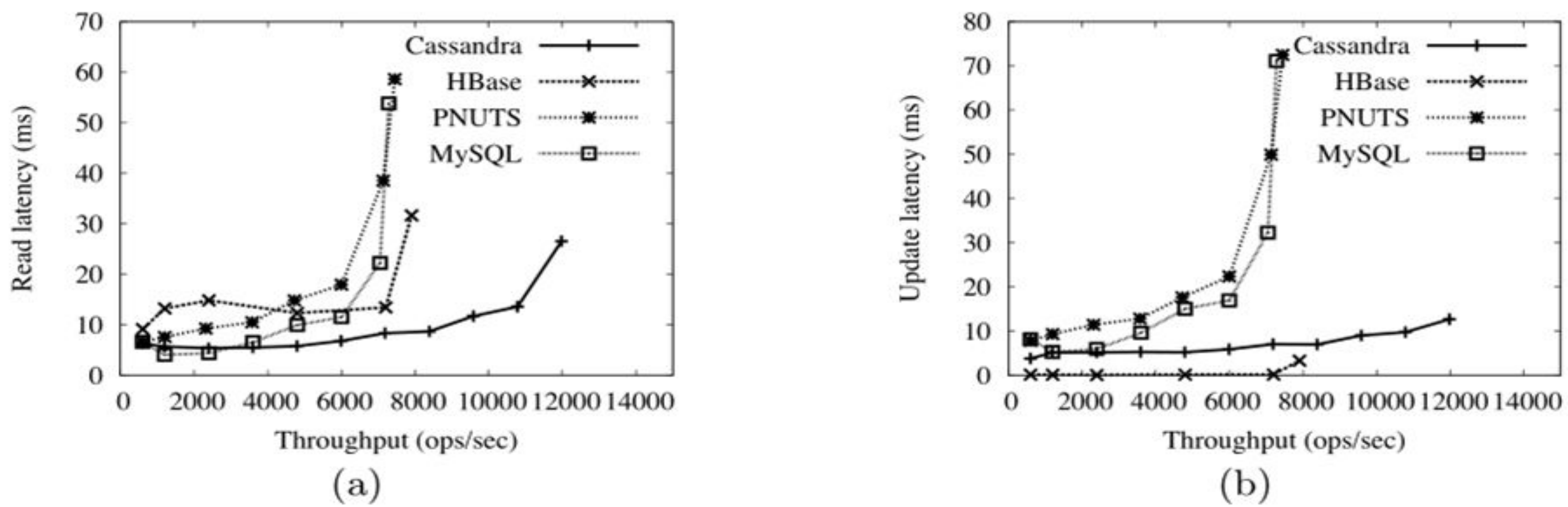
Experiment-

- 'Benchmarking Cloud Serving Systems with YCSB' by Brain F Cooper et al.
- Comparison between **Cassandra, HBase, PNUTS, and MySQL**
- Replication is disabled for this experiment
- 6 Server machines
- 20 GB data / server
- Two operations
  - **Read** - Retrieve an entire record
  - **Write** - Modify 1 of the 10 fields

Comparison using YCSB (*Cont.*)

Observation-

- Cassandra achieved the best **throughput** and the lowest **latency** for reads
- HBase does not sync to disk, but relies on in-memory replication across multiple servers for durability; this increases write throughput



Conclusion

- Cassandra is a **noSQL** database
- It is a great system if used properly
  - Application demanding high update throughput with low latency
- It provides-
  - Scalability
  - High Performance
  - Wide Applicability
- Takeaway
  - Designed to handle high write throughput without sacrificing read efficiency
  - Use of consistent Hashing
  - Persist data in the disk which is used for recovery when any node fails