AWS Keyspaces

AWS Keyspaces

- Removes the administrative overhead of managing Cassandra
- To understand why, it's helpful to begin with Cassandra architecture and then compare it to Amazon Keyspaces.

What is Cassandra?

Open source NoSQL distributed database

Trusted for scalability and high availability
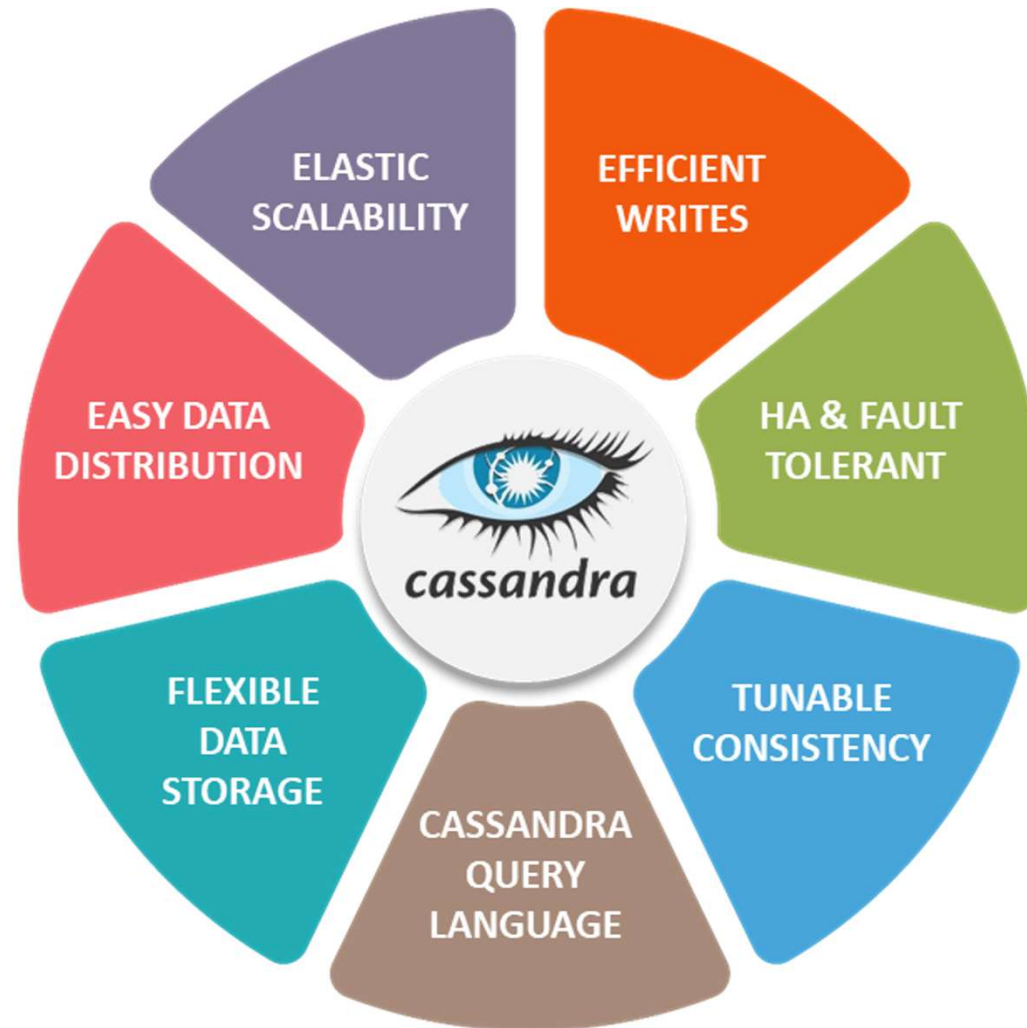
Column-oriented database

Distribution

Created at Facebook

Implements a Dynamo-style replication model with no single point of failure

Being used by some of the biggest companies such as

- Facebook,
- Twitter,
- Cisco,
- Rackspace,
- ebay, Twitter, Netflix, and more.

Features

Use Cases

Messaging

- Can handle a big amount of data
- So it is preferred for the companies that provide Mobile phones and messaging services.
- These companies have a huge amount of data, so Cassandra is best for them.

Handle high speed Applications

- Can handle the high speed data
- So it is a great database for the applications where data is coming at very high speed from different devices or sensors.

Product Catalogs and retail apps

- Cassandra is used by many retailers for durable shopping cart protection and fast product catalog input and output.

Social Media Analytics and recommendation engine

- Cassandra is a great database for many online companies and social media providers for analysis and recommendation to their customers.

When not to use Cassandra

ACID transactions

- If you expect Cassandra to build a system supporting ACID properties unfortunately, it won't work.

Strong consistency

- To achieve high availability, Cassandra sacrifices strong consistency and only grants eventual one.
- Cassandra has measures to remedy it, but it's not enough for high demanding solutions

Lots of updates and deletes

- Cassandra is incredible at writes
- But it's only append-oriented. If you need to update a lot, Cassandra's no good

Lots of scans

- Cassandra reads data pretty well
- But it's good at reading as long as you know the primary key of data you want
- If you don't, Cassandra will have to scan all nodes to find what you need, which will take a while

Real use cases when it's a 'sure'

Sensor data

- Cassandra is designed for intensive write workloads make it exceptionally good for sensor data

Messaging systems

Ecommerce websites

Entertainment websites

Fraud detection for banks

Cassandra benefits

Elastic Scalability

- Cassandra cluster can be scaled up and down as you please without much hassle

Open Source

Peer to Peer Architecture

- Follows a peer-to-peer architecture for execution, thus, resulting in rare chances for failure

Fault Tolerance

- If one server fails, or someone hacks it, the user is able to retrieve the data with utmost ease from another location
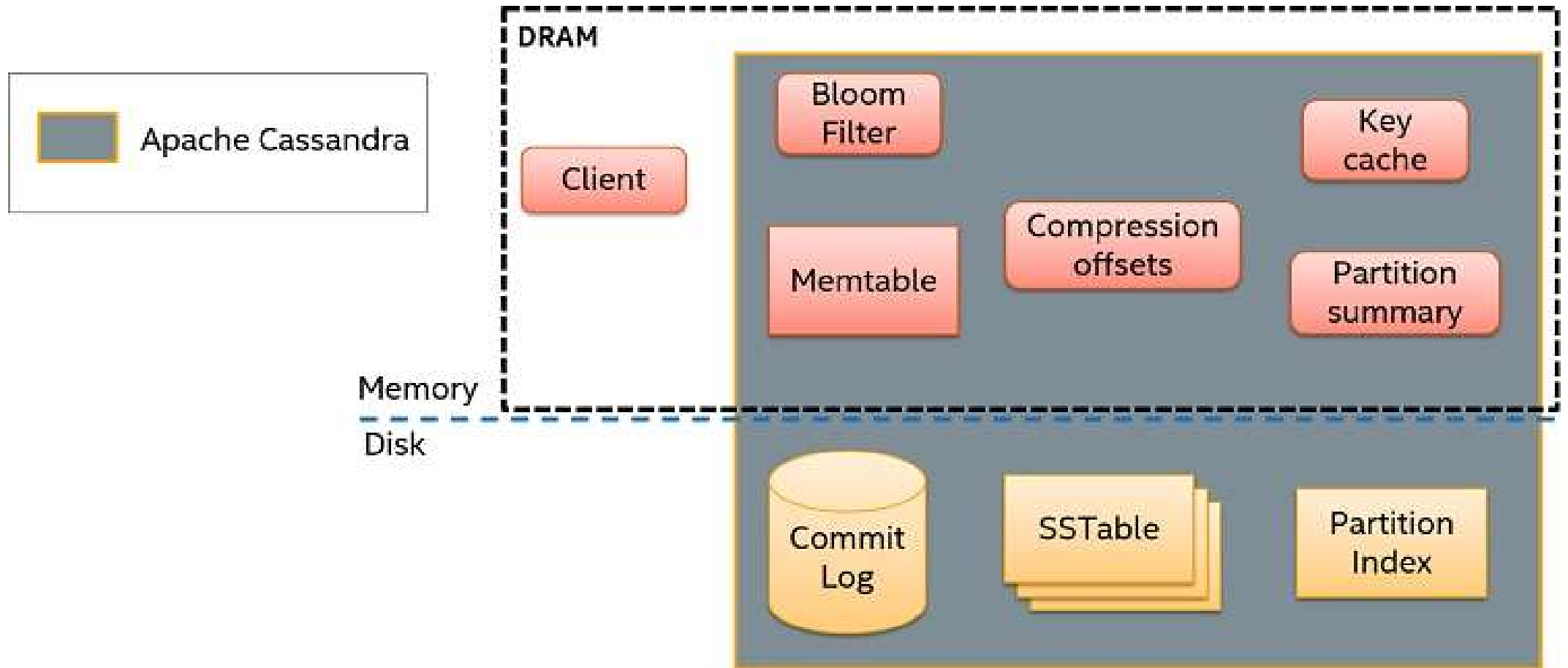
High Availability

- High Availability via data replication at the different-different location and on different data centers which gives the high availability.

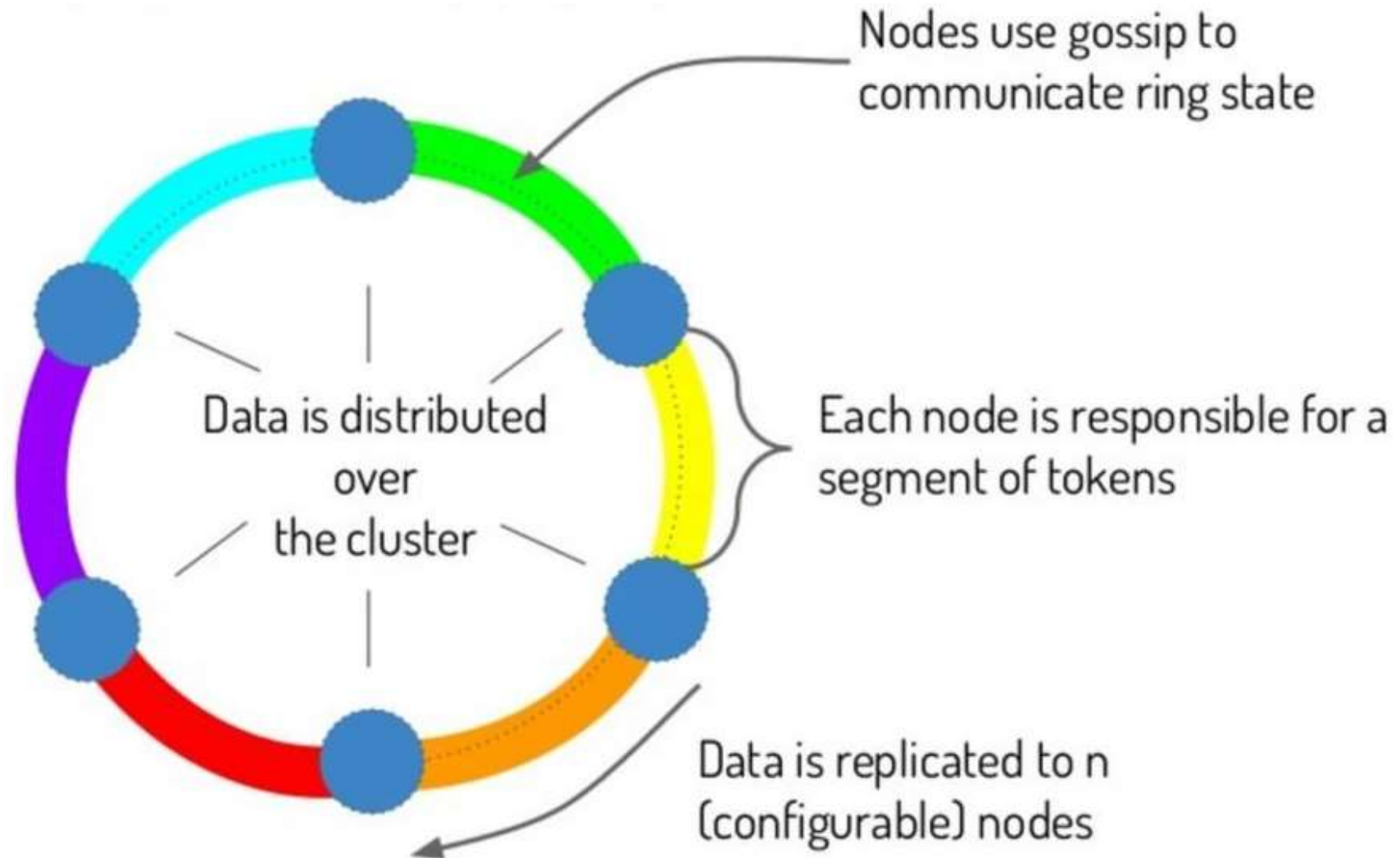## Multi-Data Center and Hybrid Cloud Support

- Designed as a distributed system for the deployment of large numbers of nodes across multiple data centers.

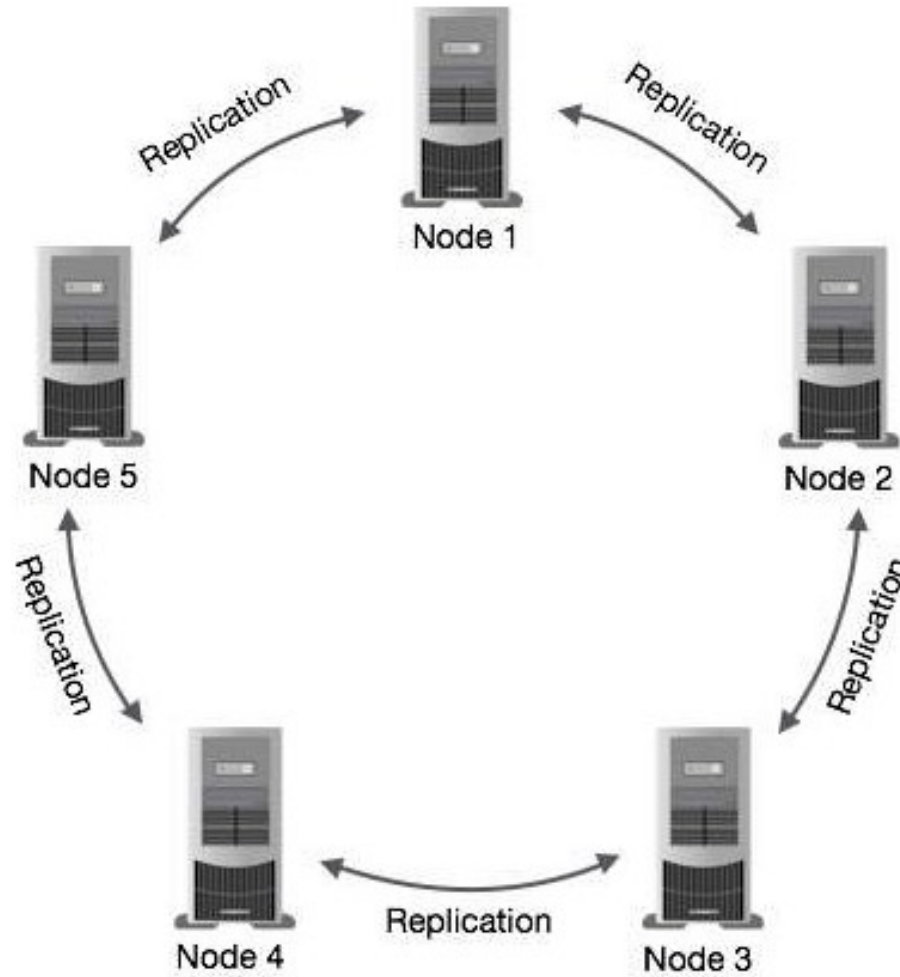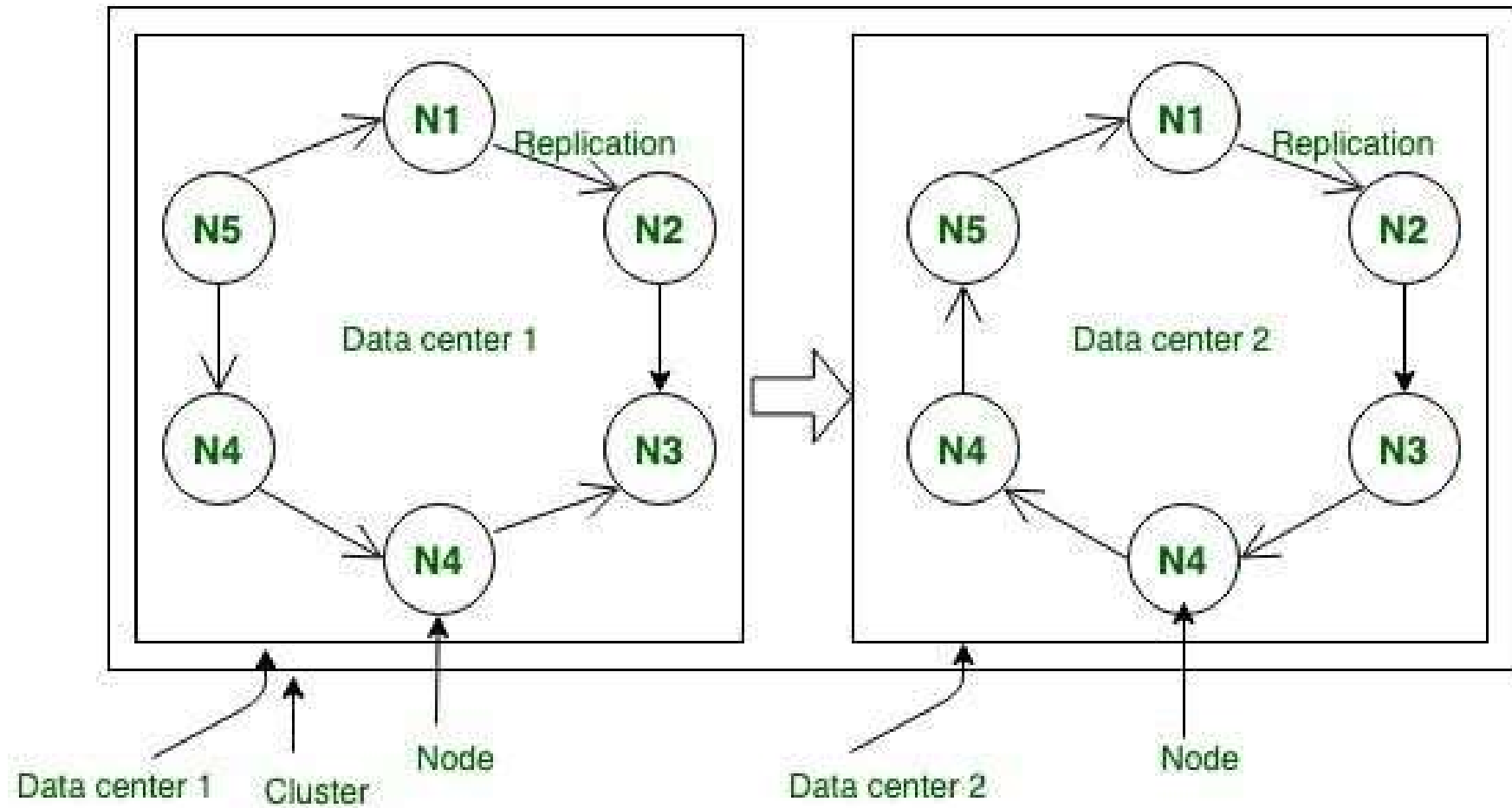# Cassandra Architecture and Terminology

Architecture

Architecture

Nodes use gossip to communicate ring state

Data is distributed over the cluster

Each node is responsible for a segment of tokens
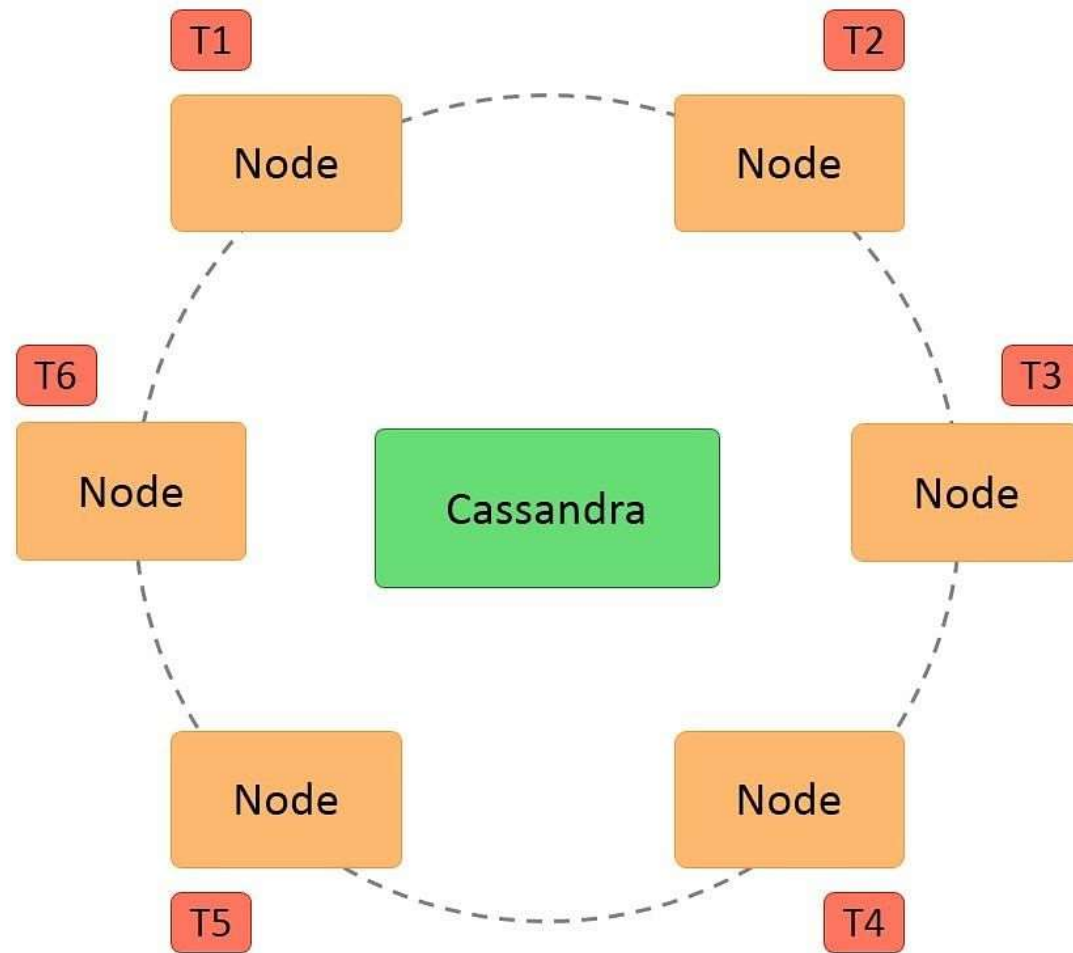
Data is replicated to n (configurable) nodes
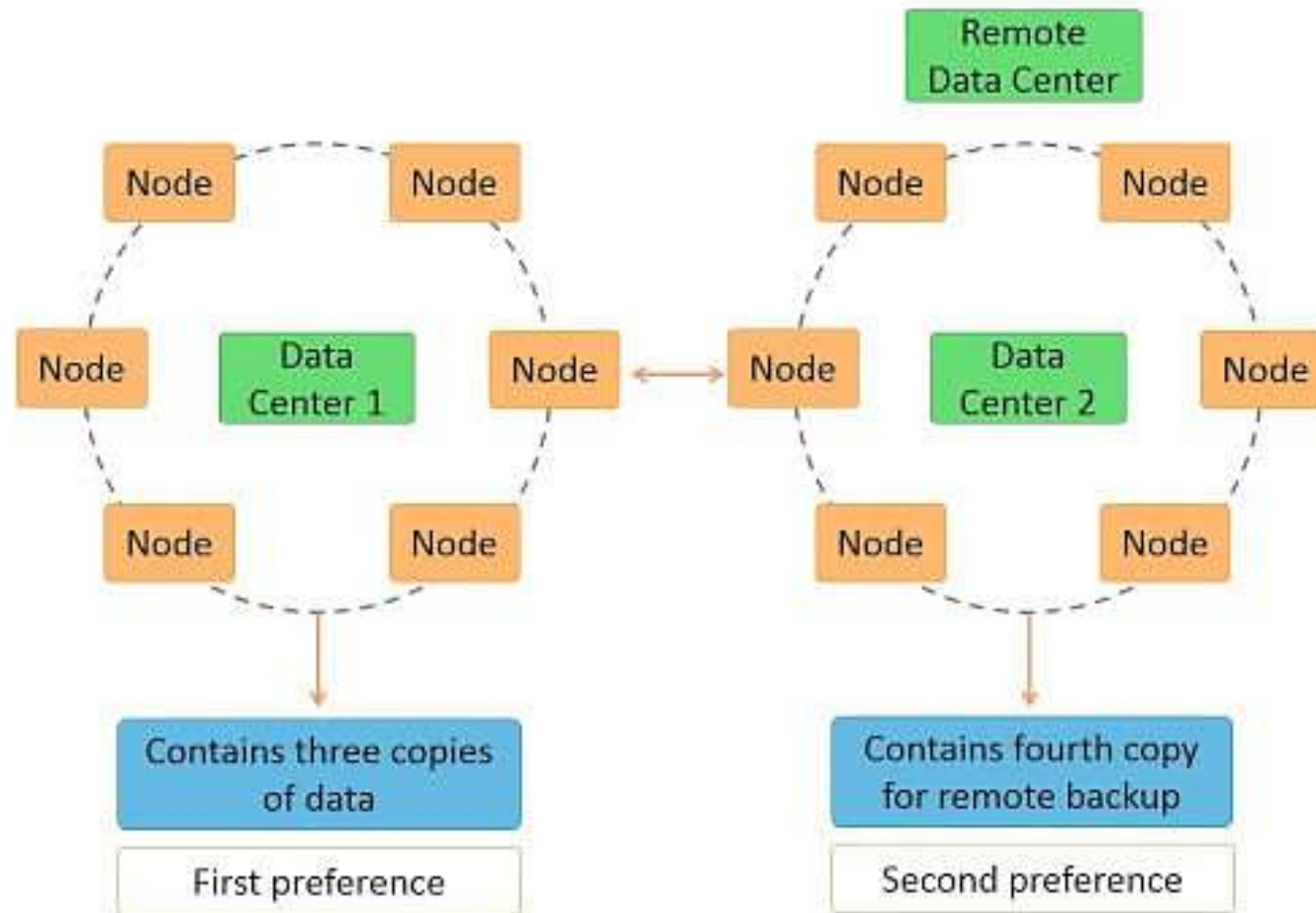
Data Replication

Data Replication

Cassandra Architecture

Cassandra Architecture

Ring Architecture with 3 nodes

20001-30000

1-10000

Node 1

Node 3

Node 2
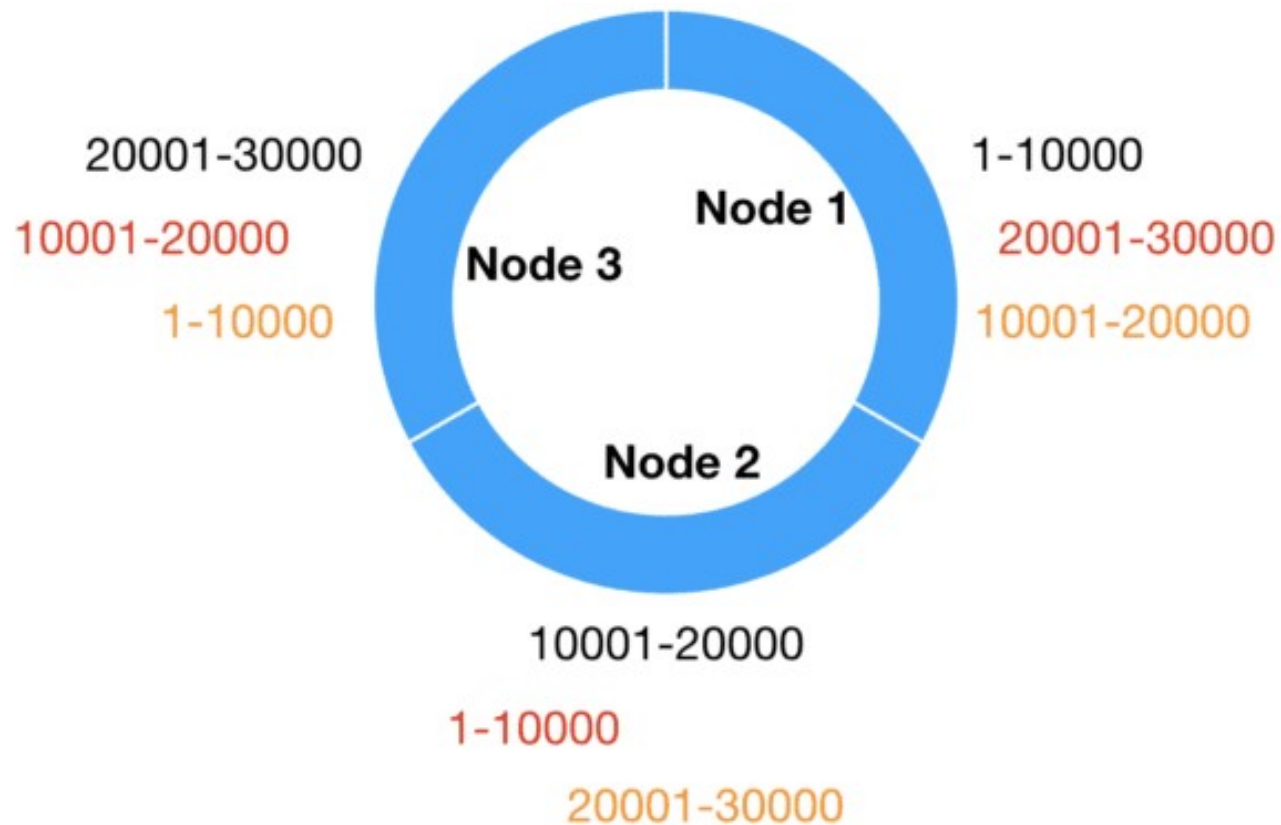
10001-20000
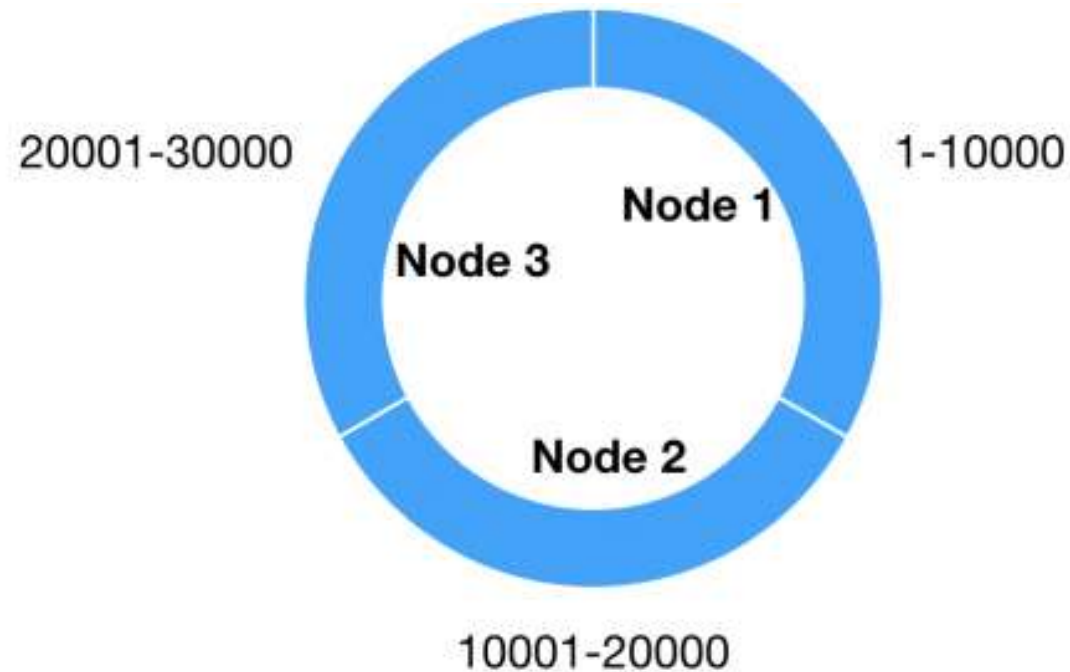
Each node in the cluster is assigned a balanced range of partitions

Ring Architecture with 3 nodes and RF=3

Each node in the cluster is assigned a balanced range of partitions, and serves to store a replication of other token ranges from the other nodes.

Replication Factor & Consistency Level

RF = 3

CL = ONE

Client

Write

Writing is successful

Read

Reading is successful ?

Coordinator Node

7

1

8

2 — New Data with bigger timestamp

3 — Old Data with smaller timestamp

4 — Old Data with smaller timestamp

5

6
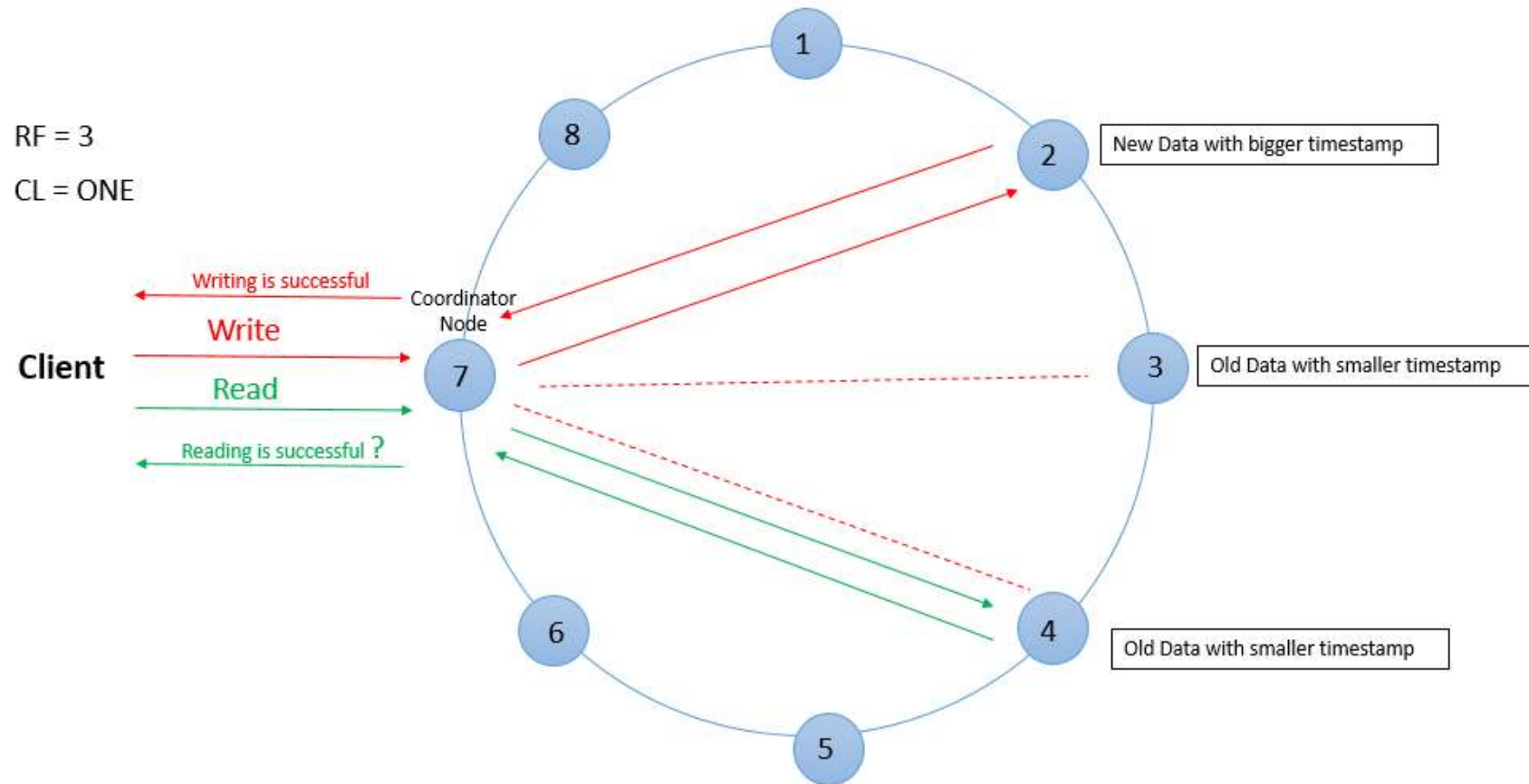
Consistent Hashing & Token Ring

Lets assume partition key is
item_name from previous example

partitions	tokens
Hash('tomato') =	19
Hash('banana') =	21
Hash('cheese') =	33
Hash('eggplant') =	59
Hash('apple') =	87

Token range : 0-25
Node 1

Token Ring
In fact, Cassandra token value is 64 bit integer. For simplicity, it is taken between 0-100.

Node 2 Token range : 25-50

Token range : 75-100 Node 4

Node 3
Token range : 50-75
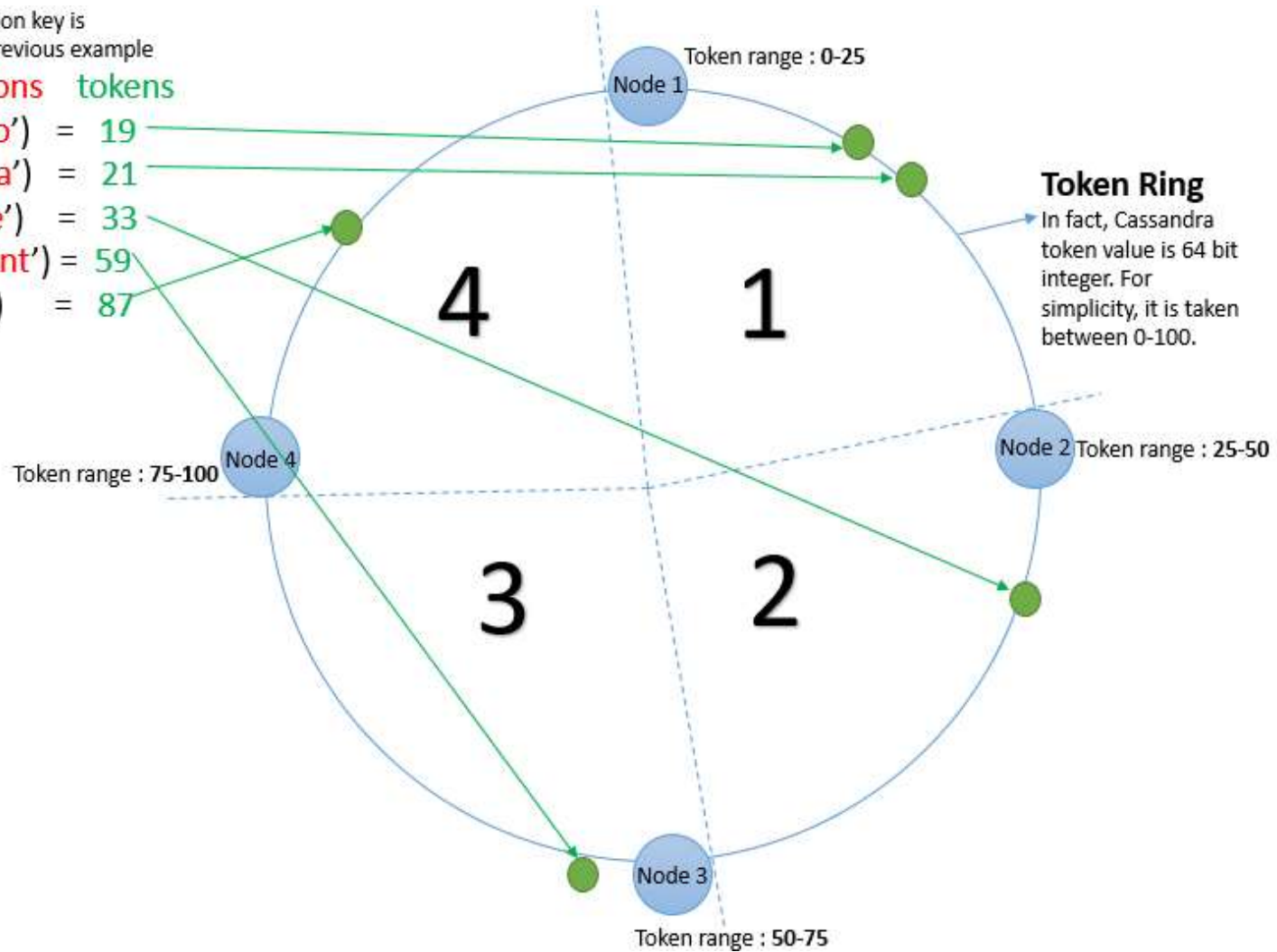
Consistent Hashing & Token Ring
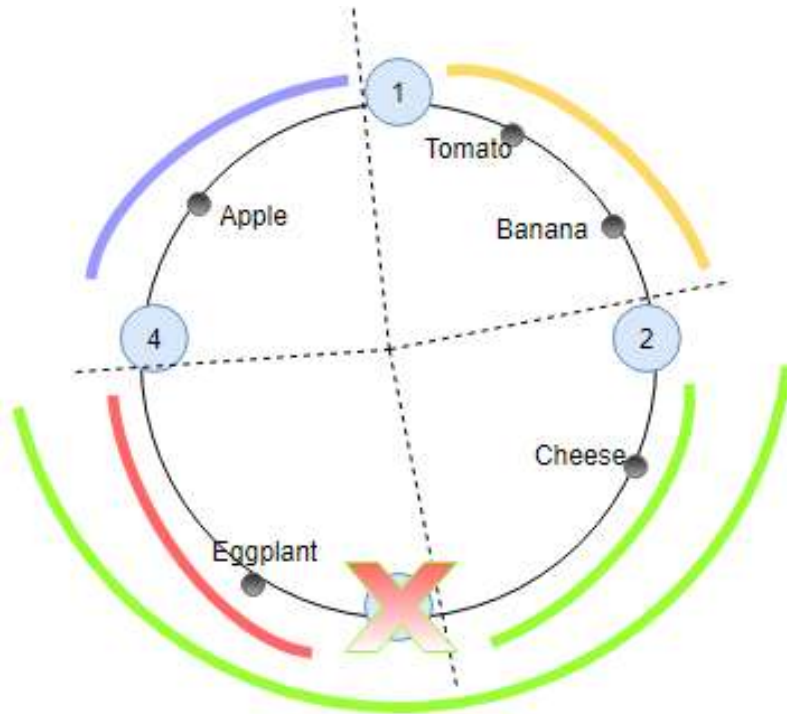
Before removing Node 3:
Node 1 -> Tomato, Banana
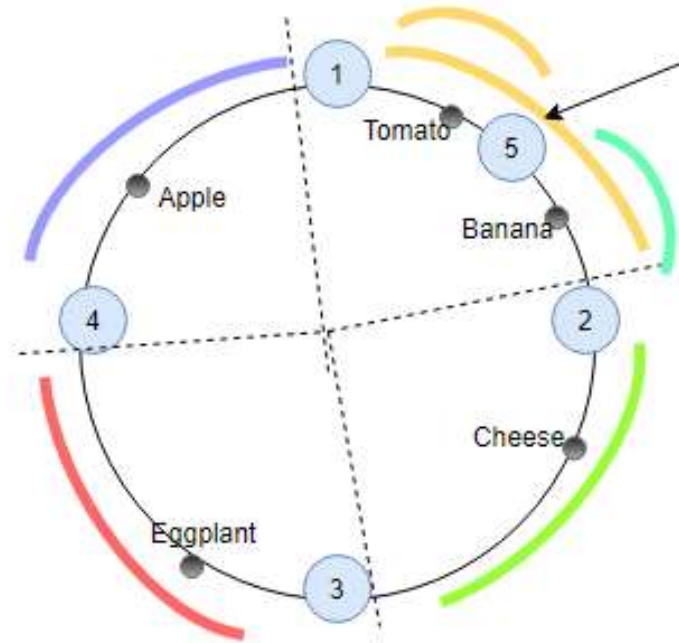Node 2 -> Cheese
Node 3 -> Eggplant
Node 4 -> Apple
After removing Node 3:
Node 1 -> Tomato, Banana
Node 2 -> Cheese, Eggplant
Node 4 -> Apple

Before adding Node 5:
Node 1 -> Tomato, Banana
Node 2 -> Cheese
Node 3 -> Eggplant
Node 4 -> Apple
After adding Node 5:
Node 1 -> Tomato
Node 2 -> Cheese
Node 3 -> Eggplant
Node 4 -> Apple
Node 5 -> Banana

Consistent Hashing & Token Ring

Real nodes

Virtual nodes

Lets assume, we have 3 virtual nodes per real node.
In fact, there are 256 virtual nodes
for a single node in Cassandra by default.
By increasing the number of virtual nodes,
we expect uniformly disribution of data load on nodes.

Virtual nodes actually can be thought as notches on token ring.
All partitions, corresponding to tokens falling into interval until next notch,
will be stored on that node. For instance, on node 1 for this situaiton.

Token Ring

Effects of the Architecture

**Enables transparent distribution of data to nodes**

- Can determine the location of your data in the cluster based on the data
- Any node can accept any request as there are no masters or slaves

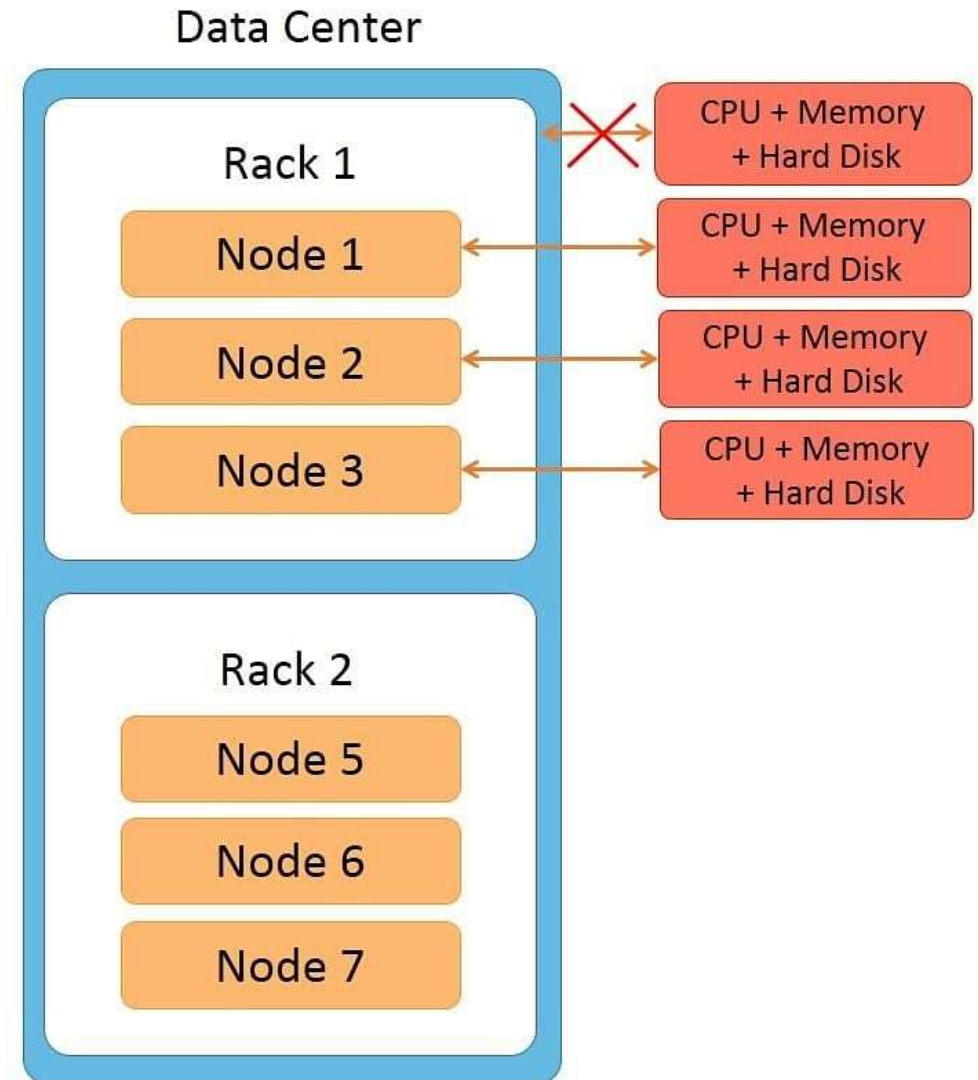**Can specify the number of replicas to achieve the required level of redundancy**

- For example, if the data is very critical, you may want to specify a replication factor of 4 or 5.

Cassandra Write Process
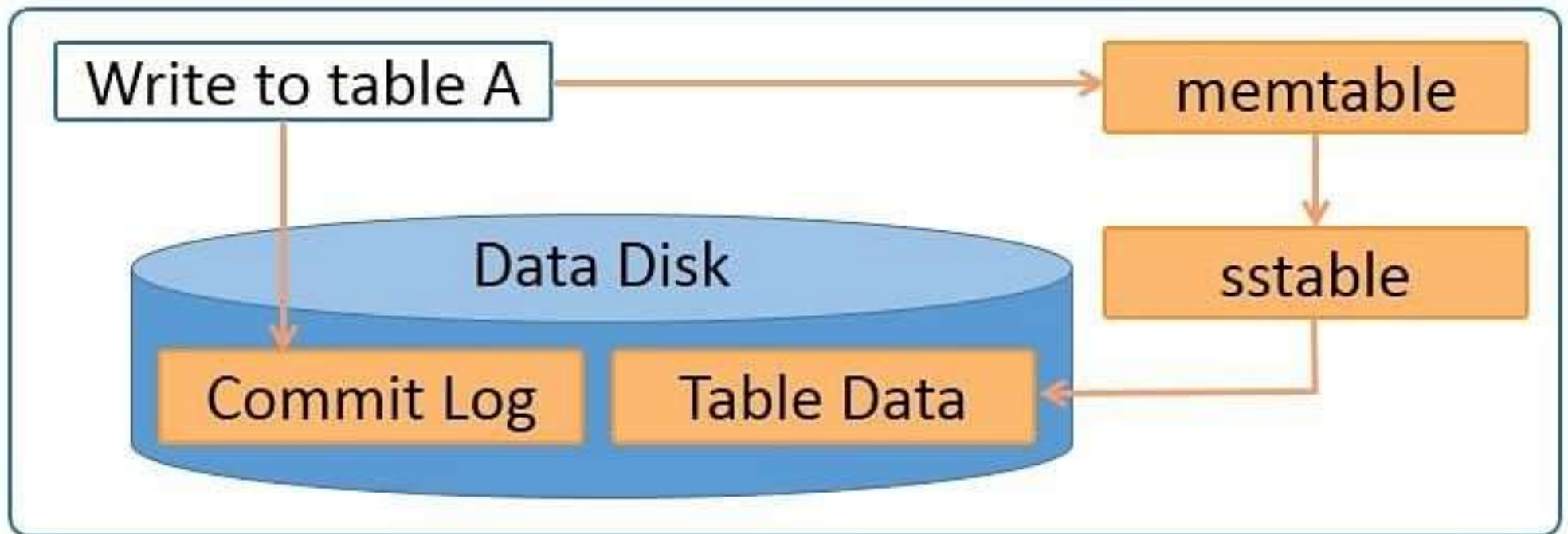
Rack

- A rack is a group of machines housed in the same physical box
- Each machine in the rack has its own CPU, memory, and hard disk

Data Center

Rack 1
- Node 1
- Node 2
- Node 3

CPU + Memory + Hard Disk
CPU + Memory + Hard Disk
CPU + Memory + Hard Disk
CPU + Memory + Hard Disk

Rack 2
- Node 5
- Node 6
- Node 7

Cassandra Read Process
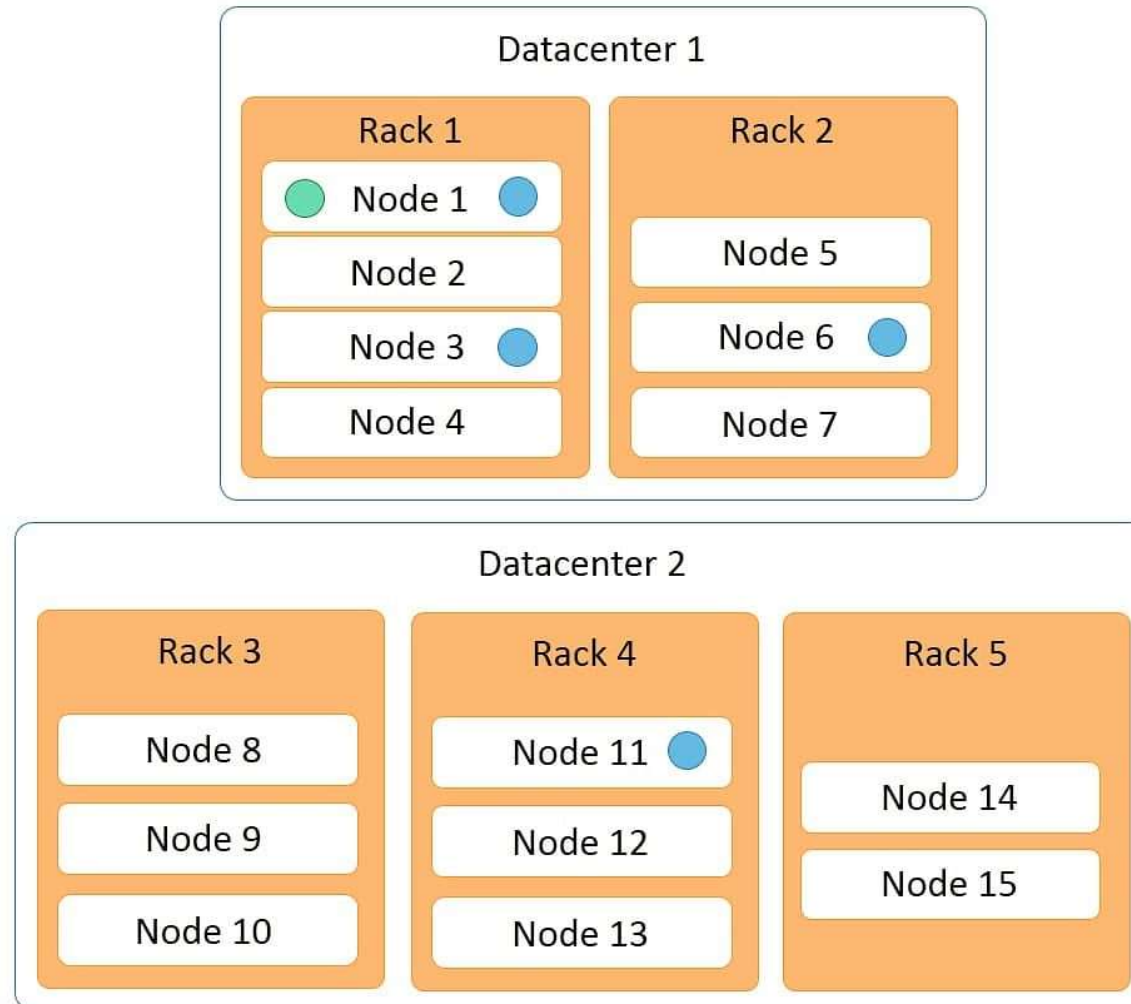
Example of Cassandra Read Process

Coordinator node
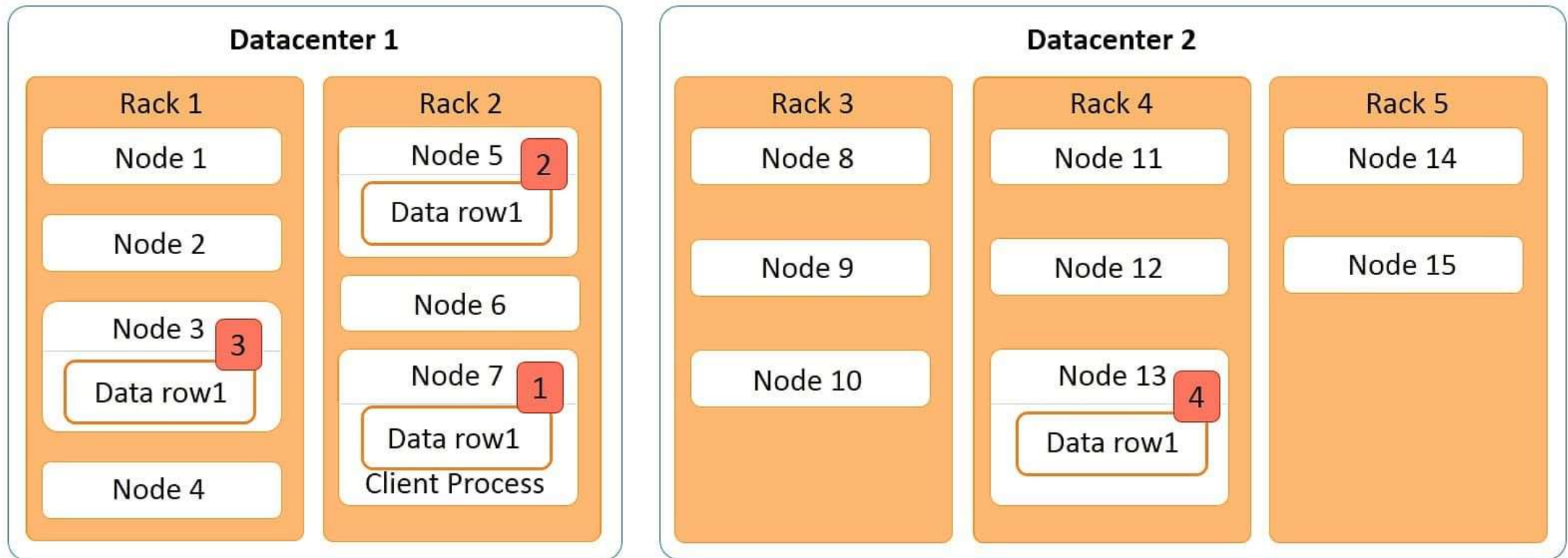
Legend:
- Coordinator node (yellow)
- Chosen node (green)
- Write response (red line)
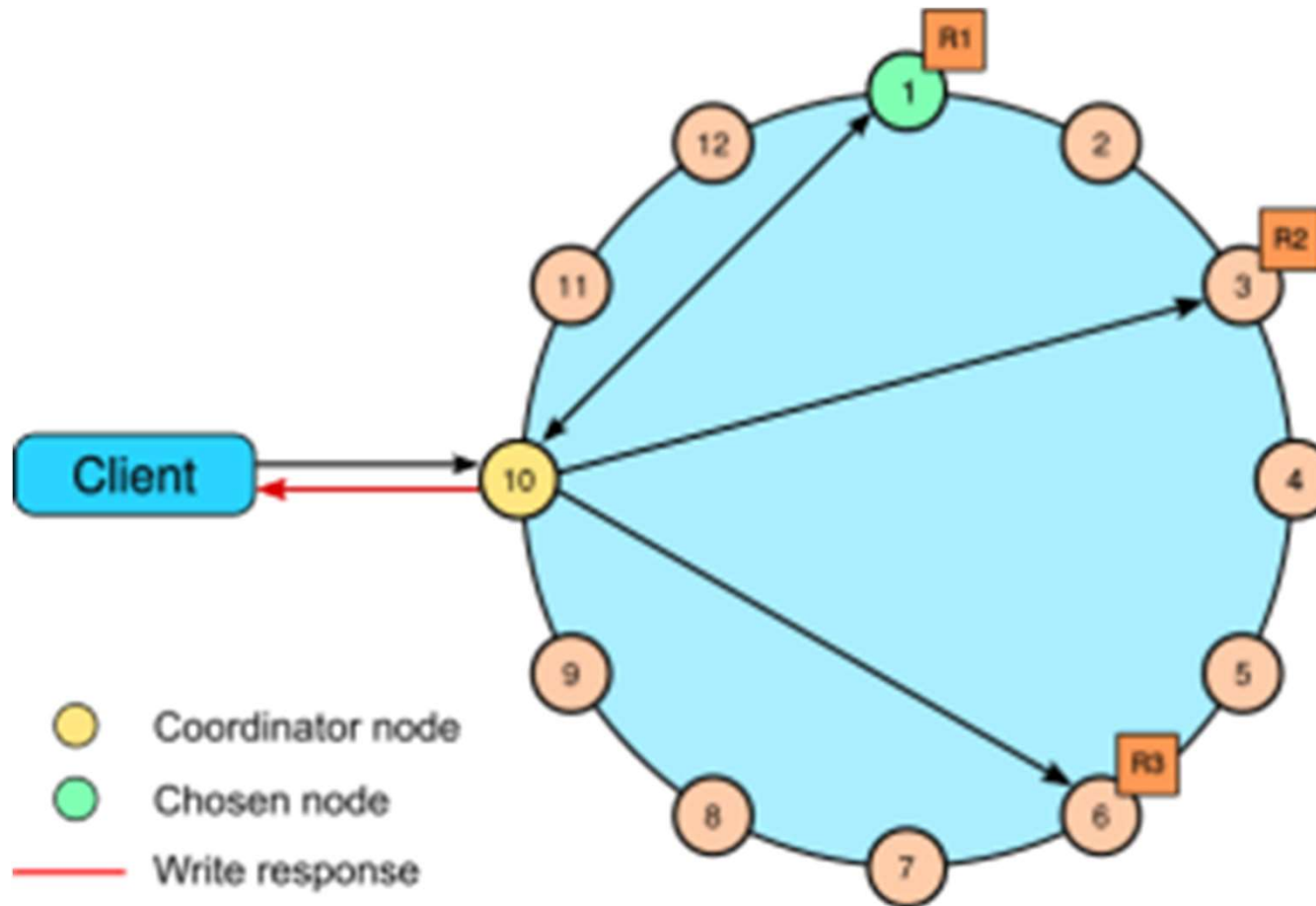
Data Partitions
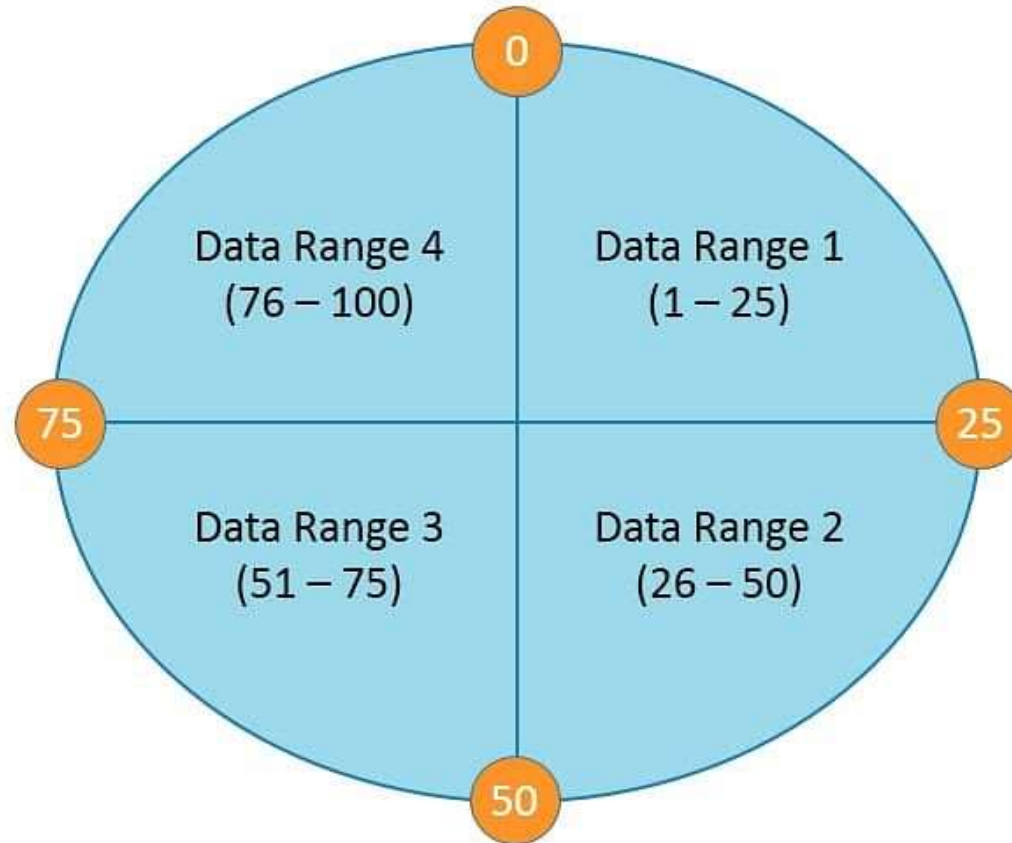
Replication in Cassandra
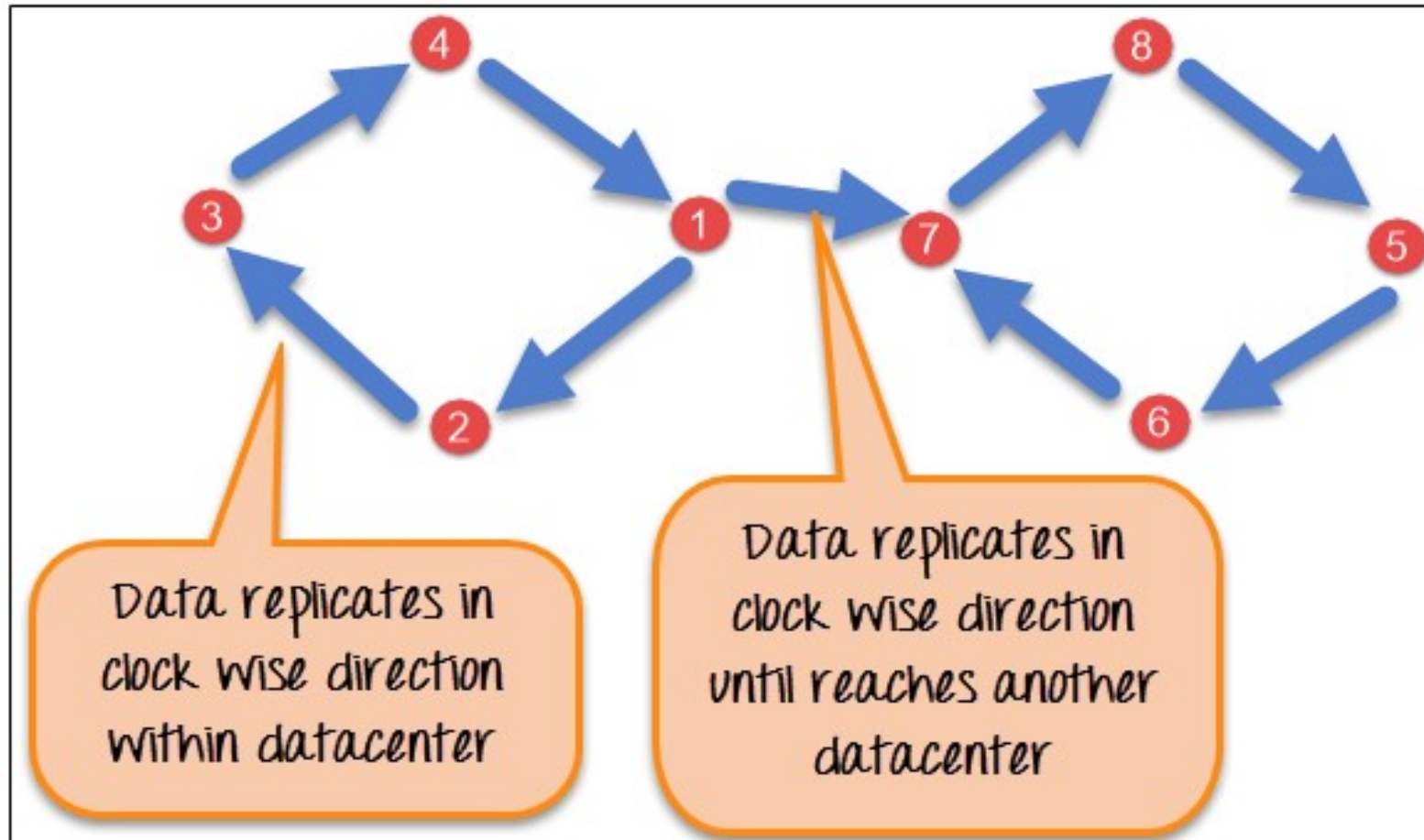
Network Topology

Network Topology Strategy

Request

Rack 1

Replica

Client

Coordinator

Replica

Rack 2

Replica
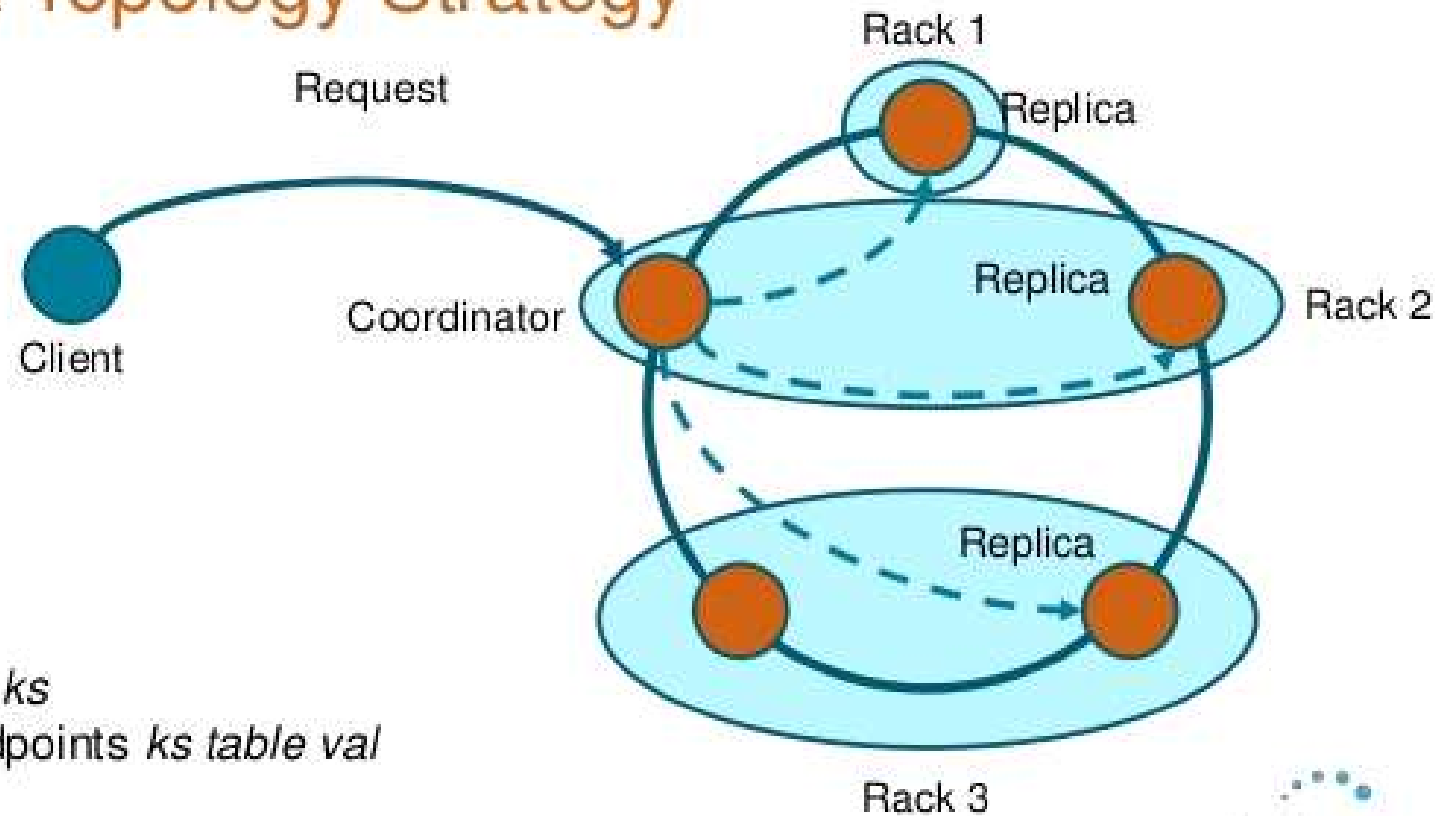
Tools:
nodetool status *ks*
nodetool getendpoints *ks table val*

Replica

Rack 3

Gossip Protocol

Seed Nodes

Virtual Nodes

Node 1

Node 4

Node 2

Node 3

Each node with 4 vnodes

Total of 16 partitions of data
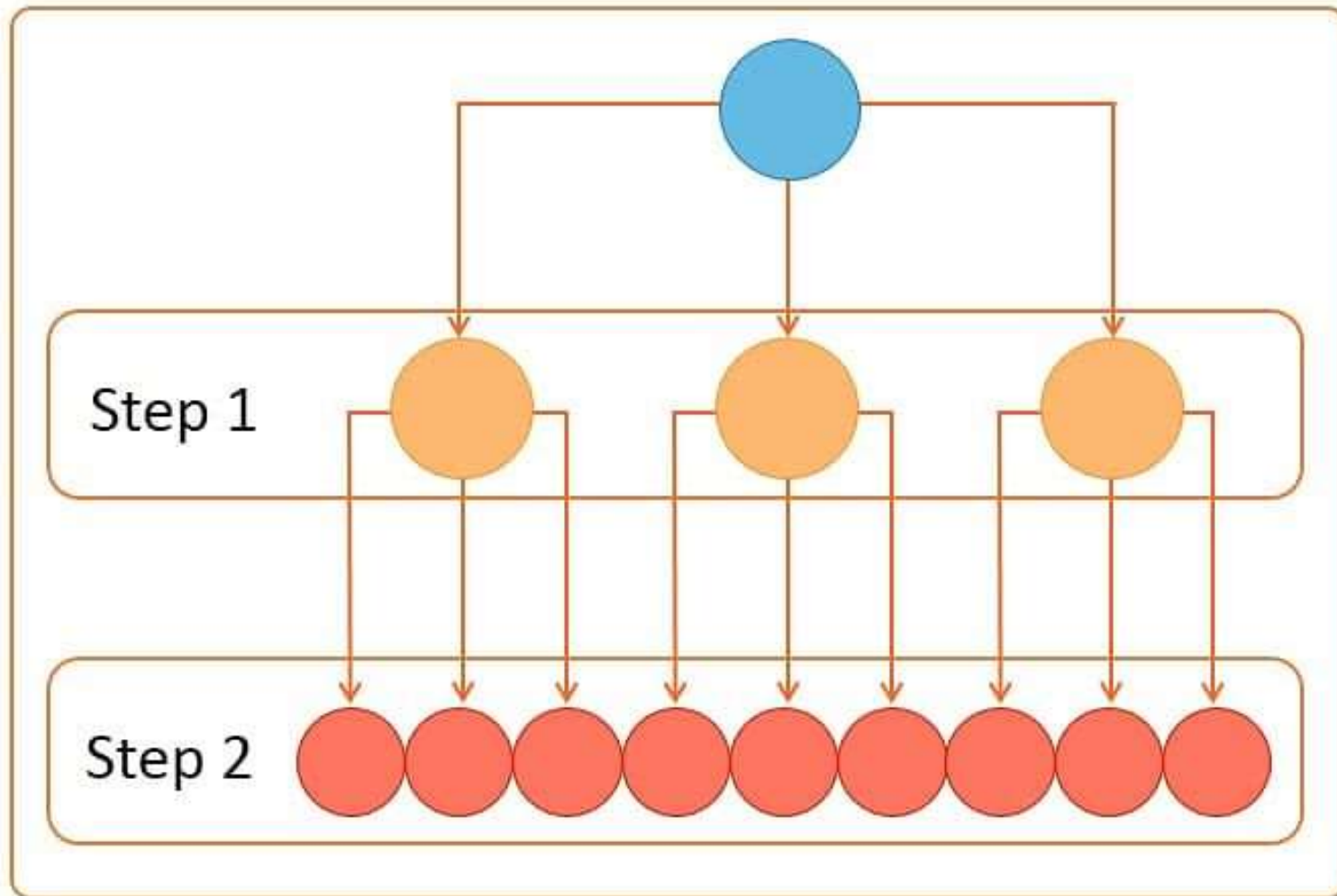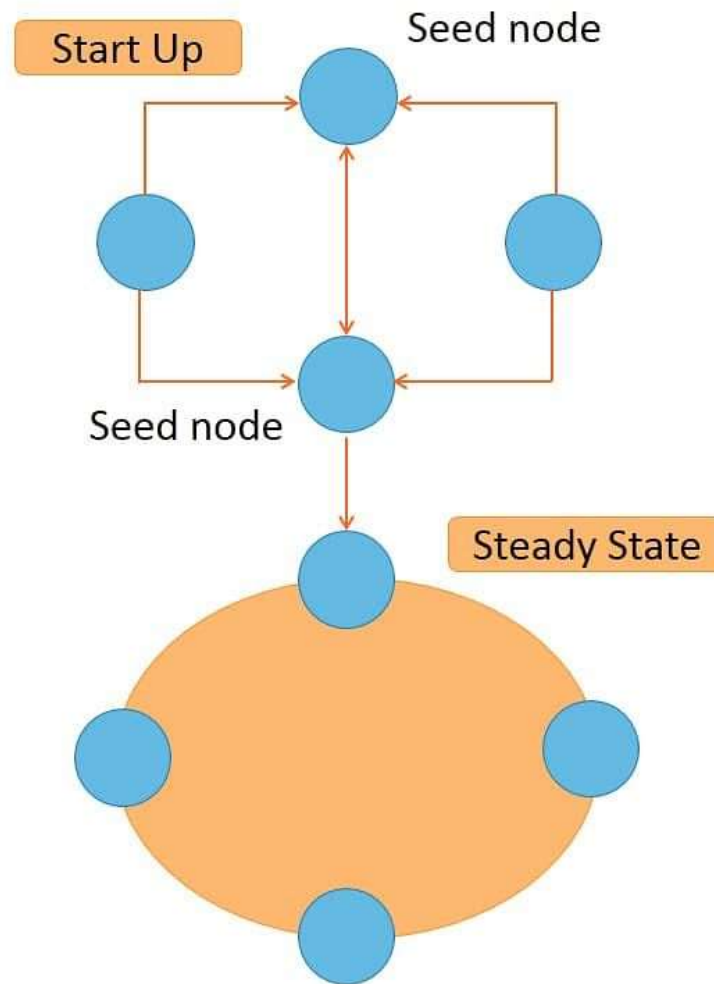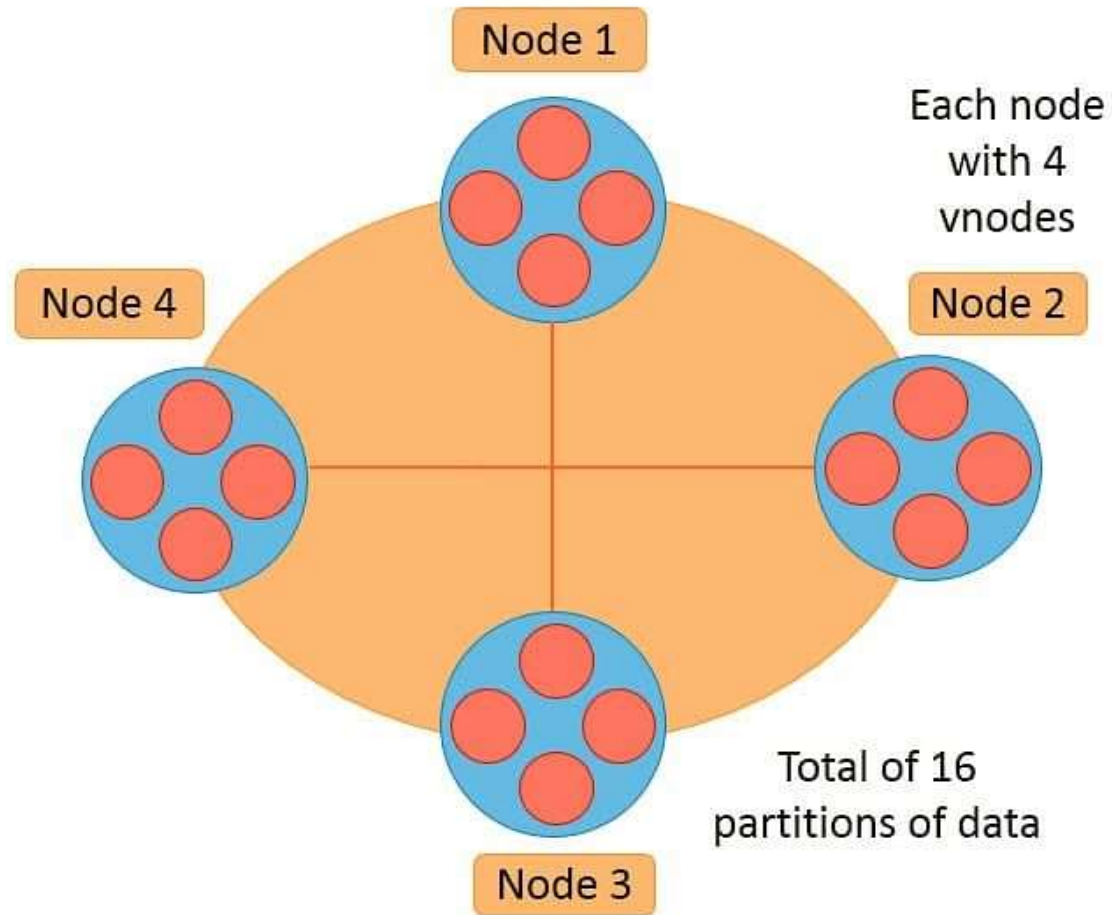
Failure Scenarios: Node Failure
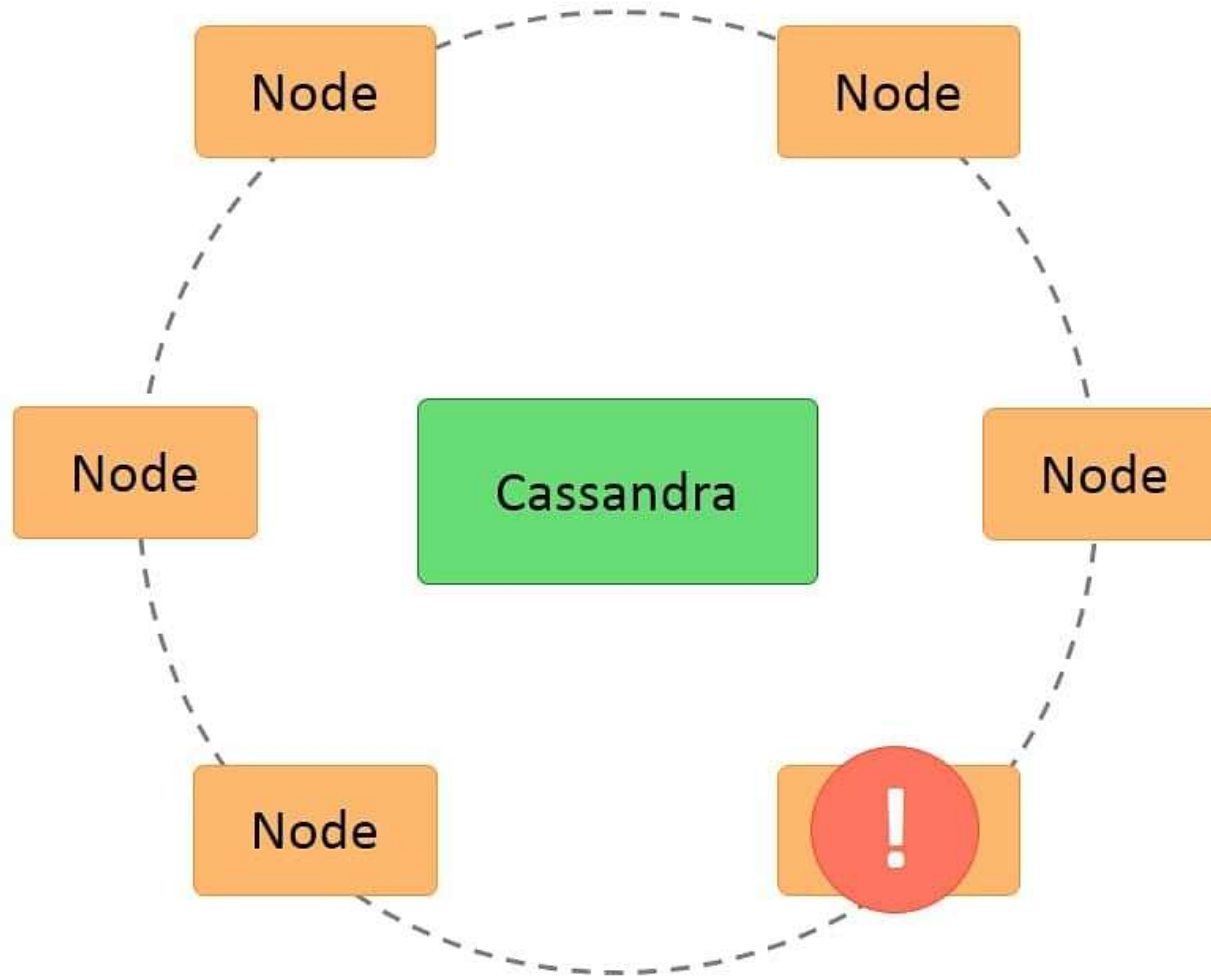
Failure Scenarios: Disk Failure

When a disk becomes corrupt, Cassandra detects the problem and takes corrective action

The effects of Disk Failure are as follows:

- The data on the disk becomes inaccessible
- Reading data from the node is not possible
- This issue will be treated as node failure for that portion of data
- Memtable and sstable will not be affected as they are in-memory tables
- Commitlog has replicas and they will be used for recovery

Failure Scenarios: Rack Failure
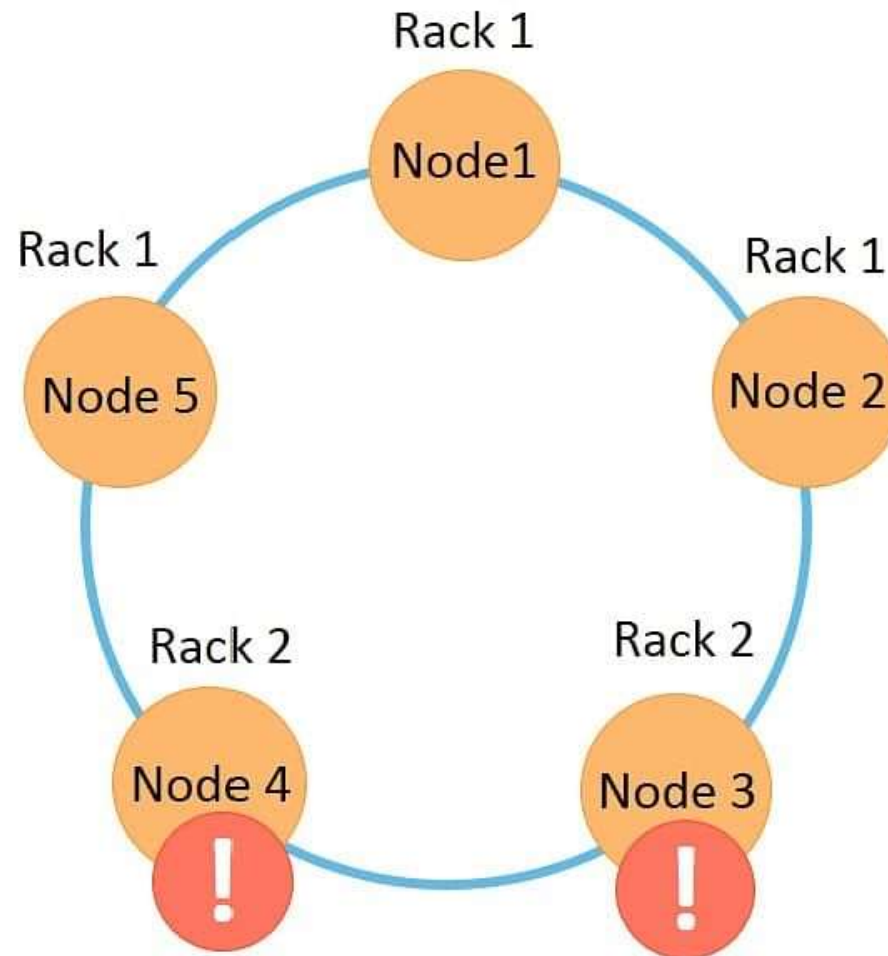
Failure Scenarios: Data Centre Failure

Data centre failure occurs when a data center is shut down for maintenance or when it fails due to natural calamities. When that happens:

All data in the data centre will become inaccessible.

All reads have to be routed to other data centres.

The replica copies in other data centres will be used.

Though the system will be operational, clients may notice slowdown due to network latency. This is because multiple data centres are normally located at physically different locations and connected by a wide area network.

Read and write paths
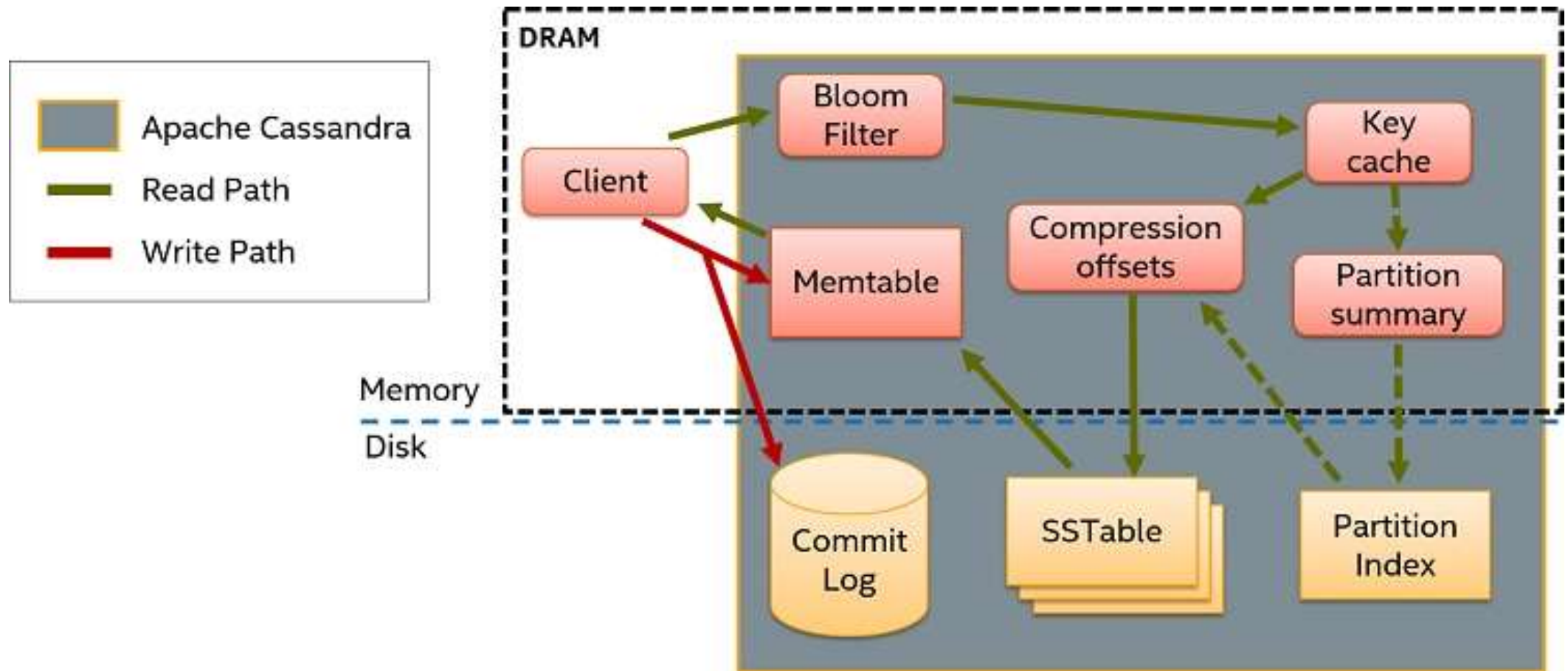
Read Path
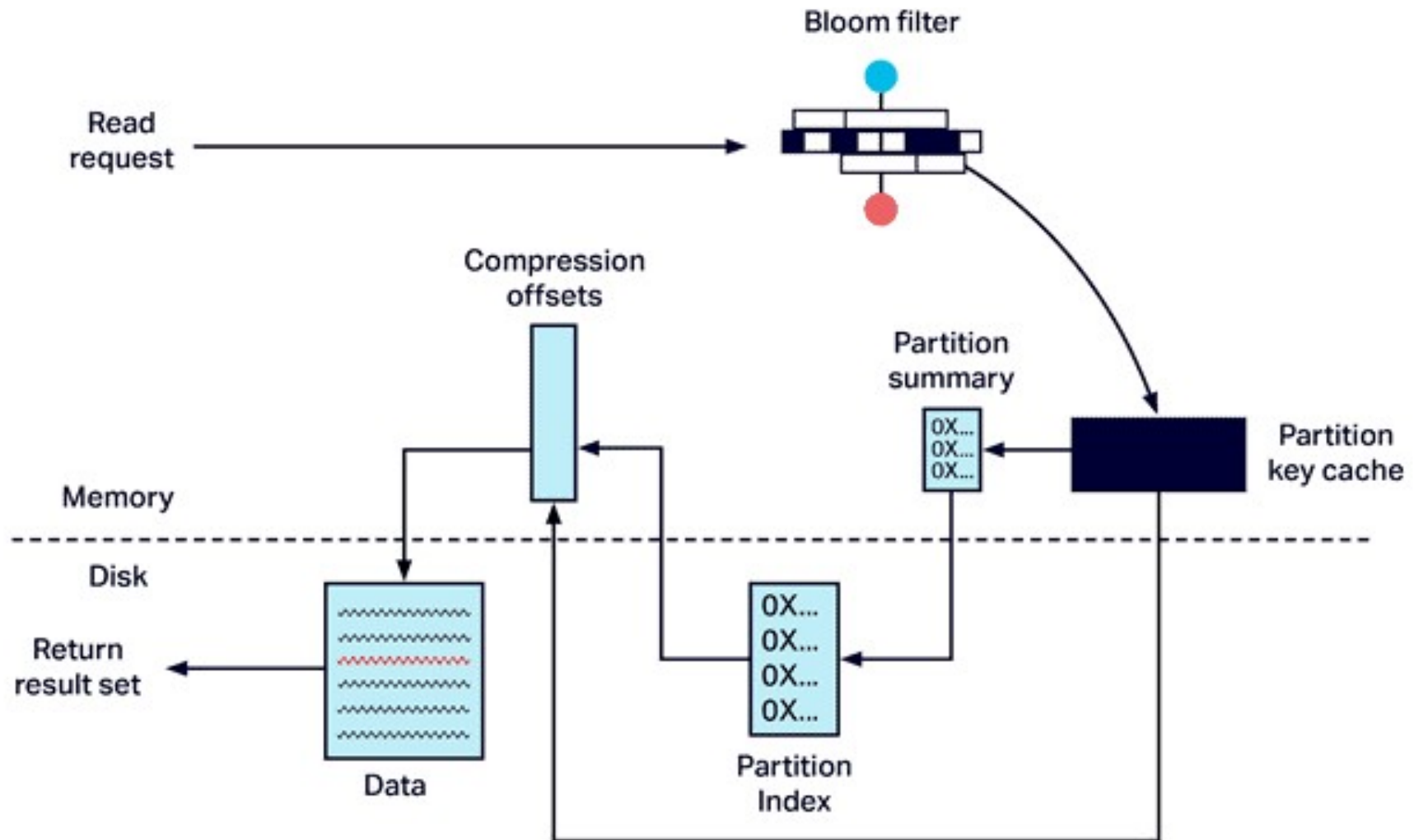
Write Path
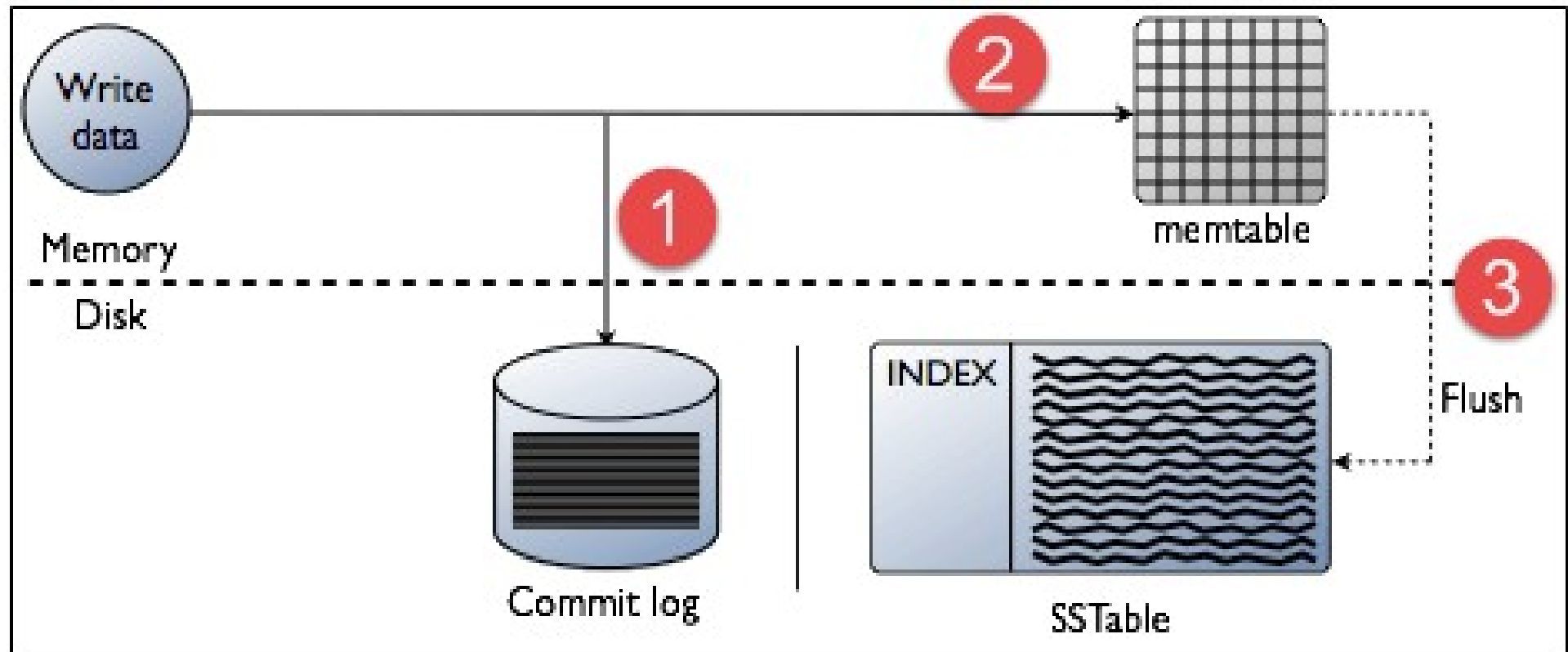
Apache Cassandra vs. Amazon Keyspaces

Traditional Apache Cassandra

Traditional Apache Cassandra is deployed in a cluster made up of one or more nodes

You are responsible for managing each node and adding and removing nodes as your cluster scales

A client program accesses Cassandra by connecting to one of the nodes and issuing Cassandra Query Language (CQL) statements

CQL is similar to SQL, the popular language used in relational databases

Even though Cassandra is not a relational database, CQL provides a familiar interface for querying and manipulating data in Cassandra.

Keyspaces

Don't need to provision, patch, or manage servers, so you can focus on building better applications

Amazon Keyspaces offers two throughput capacity modes for reads and writes

- On-demand
- Provisioned

On-demand mode

- Pay for only the reads and writes that your application actually performs

Provisioned capacity mode

- Helps optimize the price of throughput if you have predictable application traffic and can forecast your table's capacity requirements in advance
- Specify the number of reads and writes per second that you expect your application to perform
- You can increase and decrease the provisioned capacity for your table automatically by enabling automatic scaling

You can change the capacity mode of your table once per day

Stores three copies of your data in multiple Availability Zones for durability and high availability

Keyspaces
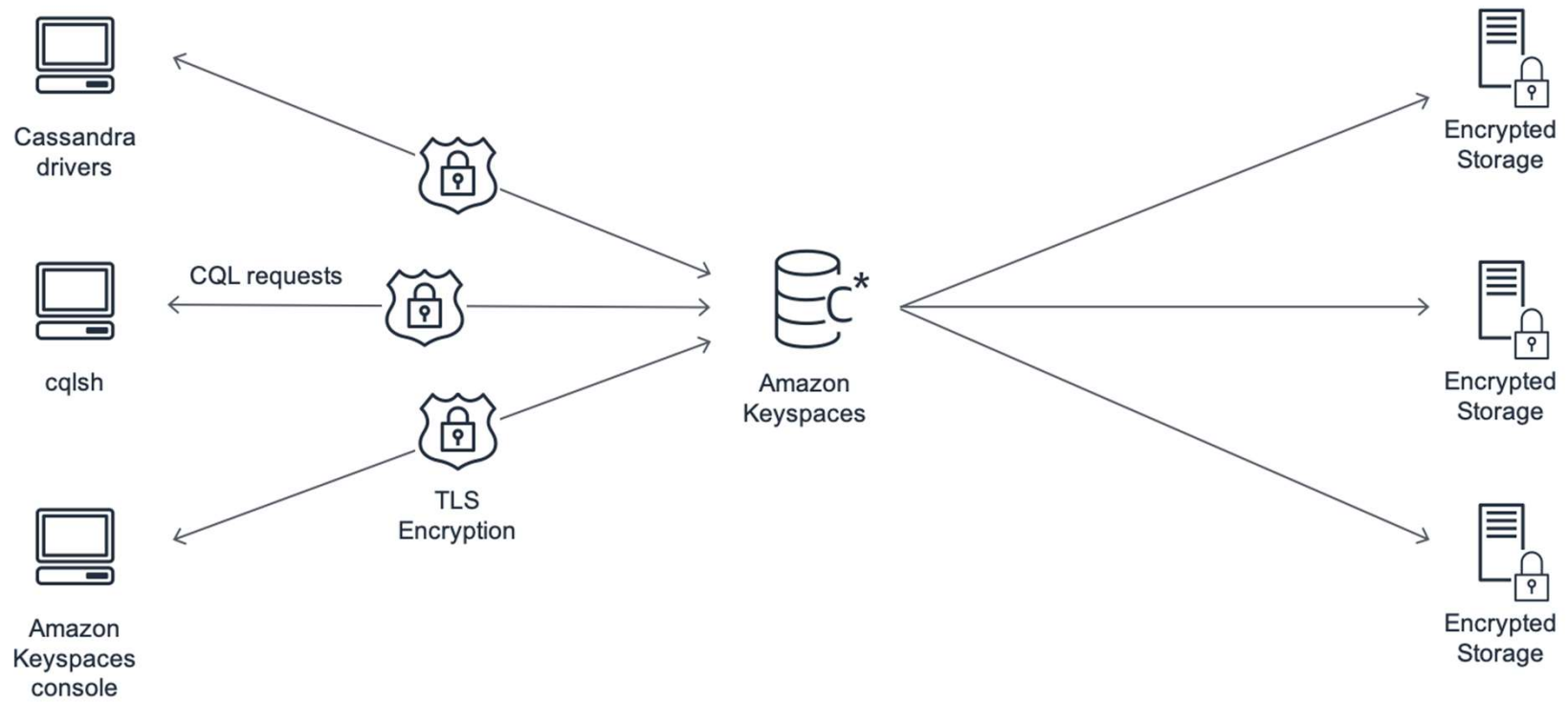
Cassandra data model

Cassandra data model

The backends of Cassandra and relational databases are very different

Some of the features of Cassandra data model are as follows:

- Data is stored as a set of rows that are organized into tables.
- Tables are also called column families.
- Each Row is identified by a primary key value.
- Data is partitioned by the primary key.
- You can get the entire data or some data based on the primary key.

Cassandra Data Model Components
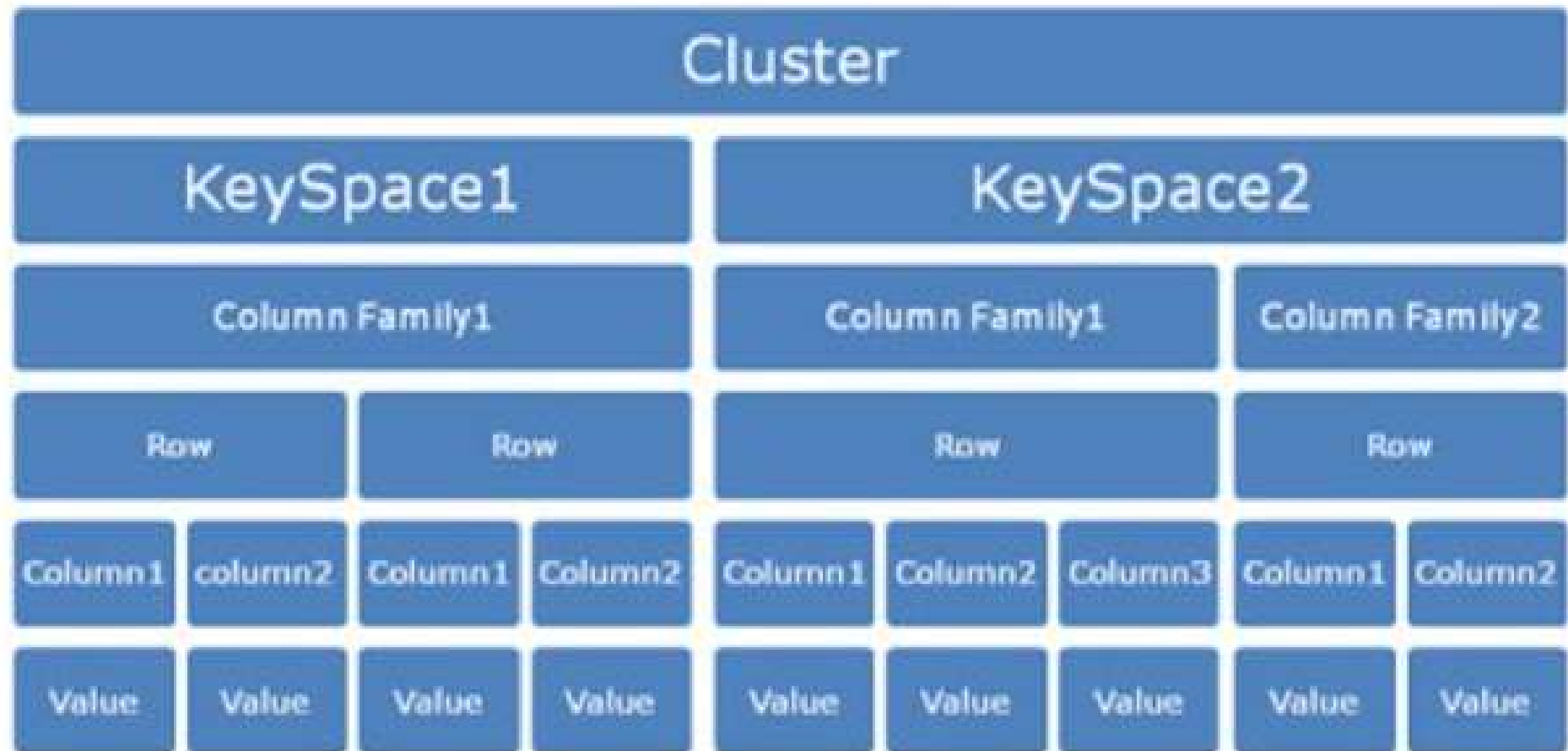
Keyspaces

```
create keyspace TestDB
    with replication = {
        'class' : 'SimpleStrategy',
        'replication_factor' : 3
    };
Use TestDB;
```

Replication factor for this keyspace

Command to start using the keyspace

Tables

```
Create table Department (
        deptID    int primary key,
        numEmployees counter) ;
Update Department set numEmployees = numEmployees + 1
            where deptID = 1000;
```

Columns

Column represents a single piece of data

Some of its features are:

- Columns consist of various types, such as integer, big integer, text, float, double, and Boolean.

- Cassandra also provides collection types such as set, list, and map.

UUID and TimeUUID

Universal Unique Identity or UUID

- This is similar to the sequence numbers in relational databases. It is a 128-bit integer.
- The given example for this column is represented as hexadecimal digits with hyphens in between:
- Sample UUID is – 01234567-0123-0123-0123-0123456789ab.

TimeUUID

- It contains a timestamp and guarantees no duplication. TimeUUID uses time in 100 nanosecond intervals. You can use the function now() to get the TimeUUID.
- Sample TimeUUID is – D2177dd0-eaa2-11de-a572-001b779c76e3.

Counter

```
Create table Department (
    deptID   int primary key,
    numEmployees counter) ;
Update Department set numEmployees = numEmployees + 1
    where deptID = 1000;
```

Compound Keys

Primary key ((key1), key2, key3, key4)

- Here the data is partitioned by key1 and clustered by key2, key3, key4

Primary key ((key1, key2), key3, key4)

- Here the data is partitioned by key1, key2 and clustered by key3, key4

Collection Columns

Collection columns are used to represent a group of data in a single column.

Cassandra provides three types of collection columns:

Set

- { 'XYZ', 'ABC','PQR' }

List

- ['2011', '2012', '2013']

Map

- { 'key1': 'value1', 'key2': 'value2'}

DDL Statements

CREATE TABLE

- create table stocks(ticker text, tradeDate timestamp, value double,
- primary key (ticker, tradeDate)) clustered by tradeDate desc;

ALTER TABLE

- alter table stocks add status text;

DROP TABLE

- drop table stocks;

DML Statements

Insert into employee (empid, empFirstName, empLastName) values (100 , 'Jack', 'Frank') if not exists;

Update employee set empFirstName = 'John' where empid = 100;

Update department set numEmployees = numEmployees + 1 where departmentId = 1000;

DML Statements

Copy employee (empid, empFirstName, empLastName) FROM employeeData.csv ;

Copy employee FROM employeeData.bar WITH HEADER = TRUE AND DELIMITER = '|' ;

DML Statements

Select * from employee where empid = 1000; // Allowed as empid is the primary key

Select empid, empFirstName from employee where empFirstName = 'Jack'; // allowed only if there is index on empFirstName

Valid and Invalid SELECT Statements

Select * from testTab where key1 = 100; //allowed

Select * from testTab where key1 = 100 and key2 = 200; // allowed

Select * from testTab where key1 = 100 and key3 = 50; // not allowed

- Cannot specify the value for key3 without specifying the value for the previous key, key2.

Select * from testTab where key1 = 100 and key2 = 20 and key4 = 3; // not allowed

- This is also not allowed as the value for key4 is specified without specifying a value for key3.

DML Statements - DELETE

If the columns are specified in the DELETE statement, only the data from the columns will be removed

Delete employeeFirstName from employee where employeeId = 2000;

- The second statement removes the value in the employeeFirstname column, where the employee ID equals 2000.

Delete from employee where employeeId = 2000

(IF salary > 100000);

- Deletes the row with employee ID whose value is 2000 along with the IF condition, where the salary is more than 100,000.

Accessing Amazon Keyspaces from an application

Implements the Apache Cassandra Query Language (CQL) API

Can use CQL and Cassandra drivers that you already use

Python Program

Consider the following Python program, which connects to a Cassandra cluster and queries a table.

- from cassandra.cluster import Cluster
- #TLS/SSL configuration goes here
- ksp = 'MyKeyspace'
- tbl = 'WeatherData'
- cluster = Cluster(['NNN.NNN.NNN.NNN'], port=NNNN)
- session = cluster.connect(ksp)
- session.execute('USE ' + ksp)
- rows = session.execute('SELECT * FROM ' + tbl)
- for row in rows:
- print(row)

Amazon Keyspaces and Cassandra

Nine-node, Apache Cassandra 3.11.2 cluster

Supports drivers and clients that are compatible with Apache Cassandra 3.11.2.

Supports the 3.x Cassandra Query Language (CQL) API

Backward-compatible with version 2.x

Can run Cassandra workloads on AWS using the same Cassandra application code

Supports all commonly used Cassandra data-plane operations, such as

- Creating keyspaces and tables,
- Reading data, and
- Writing data

Is serverless, so you don't have to provision, patch, or manage servers.

You also don't have to install, maintain, or operate software

Cassandra control plane API operations to manage cluster and node settings are not required to use Amazon Keyspaces

Settings such as replication factor and consistency level are configured automatically

Accessing Amazon Keyspaces

Accessing Amazon Keyspaces using the console

Can use the console to do the following in Amazon Keyspaces:

- Create, delete, describe, and list keyspaces and tables.
- Insert, update, and delete data.
- Run queries using the CQL editor.

Connecting programmatically to Amazon Keyspaces

Amazon Keyspaces supports drivers and clients that are compatible with Apache Cassandra 3.11.2

Steps:

- Create Credentials
- Service endpoints
- Using cqlsh
- Using the AWS CLI
- Using the API
- Working with AWS SDKs
- Using a Cassandra client driver

Generate service-specific credentials

Open AWS Identity and Access Management console at

Choose Users, and then choose the user that you created earlier

Choose Security Credentials

Under Credentials for Amazon Keyspaces, choose Generate credentials to generate the service-specific credentials

Service endpoints

Programmatic Access	Port	Authentication mechanism
CQLSH	9142	TLS
Cassandra Driver	9142	TLS
AWS CLI	443	HTTPS
AWS SDK	443	HTTPS

Using cqlsh to connect to Amazon Keyspaces

Can use the cqlsh-expansion

- A toolkit that contains common Apache Cassandra tooling like cqlsh and helpers
- Preconfigured for Amazon Keyspaces

Install

- pip3 install --user cqlsh-expansion
- cqlsh-expansion.init
- export AWS_DEFAULT_REGION=us-east-1
- export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
- export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
- cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 –ssl

Using the AWS CLI

aws keyspaces create-keyspace --keyspace-name 'catalog'

aws keyspaces get-keyspace --keyspace-name 'catalog'

aws keyspaces create-table --keyspace-name 'catalog' --table-name 'book_awards' --schema-definition 'file://schema_definition.json'

Using Amazon Keyspaces with an AWS SDK

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples🔗
AWS SDK for Go	AWS SDK for Go code examples🔗
AWS SDK for Java	AWS SDK for Java code examples🔗
AWS SDK for JavaScript	AWS SDK for JavaScript code examples🔗
AWS SDK for Kotlin	AWS SDK for Kotlin code examples🔗
AWS SDK for .NET	AWS SDK for .NET code examples🔗
AWS SDK for PHP	AWS SDK for PHP code examples🔗
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples🔗
AWS SDK for Ruby	AWS SDK for Ruby code examples🔗
AWS SDK for Rust	AWS SDK for Rust code examples🔗
AWS SDK for Swift	AWS SDK for Swift code examples🔗

Read/write capacity modes

Two read/write capacity modes

On-demand (default)

Provisioned

Read request units and write request units

One RRU represents

- One LOCAL_QUORUM read request, or
- Two LOCAL_ONE read requests, for a row up to 4 KB in size

One WRU represents

- One write for a row up to 1 KB in size
- All writes are using LOCAL_QUORUM consistency

Peak traffic and scaling properties – On Demand

On-demand capacity mode automatically adapt to your application's traffic volume

On-demand capacity mode instantly accommodates up to double the previous peak traffic on a table

For example, your application's traffic pattern might vary between 5,000 and 10,000 LOCAL_QUORUM reads per second, where 10,000 reads per second is the previous traffic peak.

With this pattern, on-demand capacity mode instantly accommodates sustained traffic of up to 20,000 reads per second

If your application sustains traffic of 20,000 reads per second, that peak becomes your new previous peak, enabling subsequent traffic to reach up to 40,000 reads per second.

Initial throughput for on-demand capacity mode

Newly created table with on-demand capacity mode

- Serve up to 4,000 WRUs and 12,000 RRUs.

Existing table switched to on-demand capacity mode

- The previous peak is half the previous WCUs and RCUs provisioned for the table or the settings for a newly created table with on-demand capacity mode, whichever is higher.

Provisioned throughput capacity mode

Specify the number of reads and writes per second that are required for your application.

Provisioned throughput capacity mode is a good option if any of the following is true:

- Have predictable application traffic
- Run applications whose traffic is consistent or ramps up gradually
- Can forecast capacity requirements to optimize price

Read capacity units and write capacity units

Specify throughput capacity in terms of read capacity units (RCUs) and write capacity units (WCUs)

One RCU represents one LOCAL_QUORUM read per second, or two LOCAL_ONE reads per second, for a row up to 4 KB in size.

One WCU represents one write per second for a row up to 1 KB in size.

All writes are using LOCAL_QUORUM consistency

Provisioned throughput is the maximum amount of throughput capacity an application can consume from a table

If your application exceeds your provisioned throughput capacity, you might observe insufficient capacity errors.

To change the throughput capacity settings for tables, you can use the AWS Management Console or the ALTER TABLE statement using CQL

Managing and viewing capacity modes

```
SELECT * from system_schema_mcs.tables where
keyspace_name = 'mykeyspace' and table_name = 'mytable';
```

Managing Amazon Keyspaces automatic scaling policies

Hands-on

Using Burst Capacity Effectively

Whenever you're not fully using a partition's throughput, Amazon Keyspaces reserves a portion of that unused capacity for later bursts of throughput to handle usage spikes.

Retains up to 5 minutes

Supported consistency levels

Which Apache Cassandra consistency levels are supported for read and write operations

Write consistency levels

Amazon Keyspaces replicates all write operations three times across multiple Availability Zones for durability and high availability

Writes are durably stored before they are acknowledged using the LOCAL_QUORUM consistency level.

Read consistency levels

Amazon Keyspaces supports three read consistency levels:

- ONE,
- LOCAL_ONE, and
- LOCAL_QUORUM

During a LOCAL_QUORUM read, Amazon Keyspaces returns a most recent response

Unsupported consistency levels

Apache Cassandra	Amazon Keyspaces
EACH_QUORUM	Not supported
QUORUM	Not supported
ALL	Not supported
TWO	Not supported
THREE	Not supported
ANY	Not supported
SERIAL	Not supported
LOCAL_SERIAL	Not supported

User Administration

Accessing Amazon Keyspaces

You can access Amazon Keyspaces

- Using the console
- Programmatically by running a cqlsh client, or
- By using an Apache 2.0 licensed Cassandra driver

Connecting programmatically to Amazon Keyspaces

Generate service-specific credentials

- Open the AWS Identity and Access Management
- In the navigation pane, choose Users, and then choose the user that you created earlier
- Choose Security Credentials
- Under Credentials for Amazon Keyspaces, choose Generate credentials to generate the service-specific credentials

Create and configure AWS credentials

Sign in to the AWS Management Console

Choose Users and then choose Add users.

Type the user name for the new user

Select Access key - Programmatic access to create an access key for the new user.

Ports and Protocols

Programmatic Access	Port	Authentication mechanism
CQLSH	9142	TLS
Cassandra Driver	9142	TLS
AWS CLI	443	HTTPS
AWS SDK	443	HTTPS

Global Service endpoints

Region Name	Endpoint	Protocol
US East (Ohio)	cassandra.us-east-2.amazonaws.com	HTTPS and TLS
US East (N. Virginia)	cassandra.us-east-1.amazonaws.com cassandra-fips.us-east-1.amazonaws.com	HTTPS and TLS TLS
US West (N. California)	cassandra.us-west-1.amazonaws.com	HTTPS and TLS
US West (Oregon)	cassandra.us-west-2.amazonaws.com cassandra-fips.us-west-2.amazonaws.com	HTTPS and TLS TLS

Using cqlsh to connect

- export AWS_DEFAULT_REGION=us-east-1

- export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE

- export
  AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

- cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 –ssl

- Or

- cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 -u myUserName -p myPassword --ssl

Point-in-time recovery

Enabling point-in-time recovery

Open the Amazon Keyspaces console

In the navigation pane, choose Tables and select the table you want to edit.

On the Backups tab, choose Edit.

In the Edit point-in-time recovery settings section, select Enable Point-in-time recovery.

Enabling PITR using CQL

- ALTER TABLE mykeyspace.mytable
- WITH custom_properties = {'point_in_time_recovery': {'status': 'enabled'}}

- CREATE TABLE "my_keyspace1"."my_table1"(
-     "id" int,
-     "name" ascii,
-     "date" timestamp,
-     PRIMARY KEY("id"))
- WITH CUSTOM_PROPERTIES = {
-     'capacity_mode':{'throughput_mode':'PAY_PER_REQUEST'},
-     'point_in_time_recovery':{'status':'enabled'}
- }

Time window for PITR continuous backups

The maximum backup window is 35 days which can't be modified

Can restore to any point in time between EarliestRestorableDateTime and CurrentTime.

Can also restore deleted tables

Restoring an Amazon Keyspaces table to a point in time

Open the Amazon Keyspaces console

In the navigation pane on the left side of the console, choose Tables.

In the list of tables, choose the mytable table.

On the Backups tab of the mytable table, in the Point-in-time recovery section, choose Restore.

For the new table name, enter mytable_restored.

Restoring a table to a point in time with CQL

- RESTORE TABLE mykeyspace.mytable_restored
- FROM TABLE mykeyspace.mytable
- WITH restore_timestamp = '2020-06-30T19:19:21.175Z';

Restoring a deleted table with CQL

- RESTORE TABLE mykeyspace.mytable_restored
- FROM TABLE mykeyspace.mytable;

Multi-Region Replication

Multi-Region Replication

- To replicate data with automated, fully managed, active-active replication across the AWS Regions of your choice.

Benefits

Global reads and writes with single-digit millisecond latency.

- Can use Amazon Keyspaces multi-Region tables for global applications that need a fast response time anywhere in the world.

Improved business continuity and protection from single-Region degradation.

- Can recover from degradation in a single AWS Region by redirecting your application to a different Region

High-speed replication across Regions.

Consistency and conflict resolution.

- Cell-level timestamps and a last writer wins reconciliation between concurrent updates.

Creating a multi-Region keyspace

Open the Amazon Keyspaces console

In the navigation pane, choose Keyspaces, and then choose Create keyspace.

For Keyspace name, enter the name for the keyspace.

In the Multi-Region replication section, you can add up to five additional Regions that are available in the list.

To finish, choose Create keyspace.

Best practices

Approaching NoSQL design

The first step in designing your Amazon Keyspaces application is to identify the specific query patterns that the system must satisfy.

Understand three fundamental properties of your application's access patterns

- Data size
  - Knowing how much data will be stored and requested at one time helps to determine the most effective way to partition the data.
- Data shape
  - NoSQL database organizes data so that its shape in the database corresponds with what will be queried
- Data velocity (speed)
  - Amazon Keyspaces scales by
    - Increasing the number of physical partitions to process queries, and
    - By efficiently distributing data across those partitions
  - Knowing in advance what the peak query loads will be might help determine how to partition data to best use I/O capacity.

General principles that govern performance

Keep related data together

- locality of reference

Use sort order

Distribute queries

- Design data keys to distribute traffic evenly across partitions as much as possible

Data modeling

The primary key table can consist of one or multiple partition key columns

These columns determine which partitions the data is stored in, and

One or more optional clustering column, which define how data is clustered and sorted within a partition.

How you chose your partition key can have a significant impact upon the performance of your queries

Primary Key = Partition Key + [Clustering Columns]

Data modeling

Sensor #	Date	Timestamp	Metric1	Metric2	Metric3
1	2015-01-01	20150101-000000	5.01	5.67	0.678
1	2015-01-01	20150101-000010	5.01	5.67	0.678
1	2015-01-01	20150101-000020	5.05	5.8	0.678
1	2015-01-02	20150102-000000	5.01	5.67	0.678
1	2015-01-02	20150102-000010	5.01	5.67	0.678
1	2015-01-02	20150102-000020	5.05	5.8	0.678
2	2015-01-02	20150102-000000	6.01	7.67	0.978
2	2015-01-02	20150102-000010	6.01	7.67	0.698
2	2015-01-02	20150102-000020	6.05	8.8	0.679

Node 1

Node 2

Partition Key

Clustering Key

Primary Key

PRIMARY KEY ((Sensor,Date),Timestamp)

Write sharding

```
CREATE TABLE IF NOT EXISTS tracker.blogs (

publish_date date,

shard int,

title text,

description int,

PRIMARY KEY ((publish_date, shard)));
```

Optimizing costs

Optimizing costs

- The Cost Explorer tool found within the AWS Management Console allows you to see costs broken down by type, such as read, write, storage and backup charges
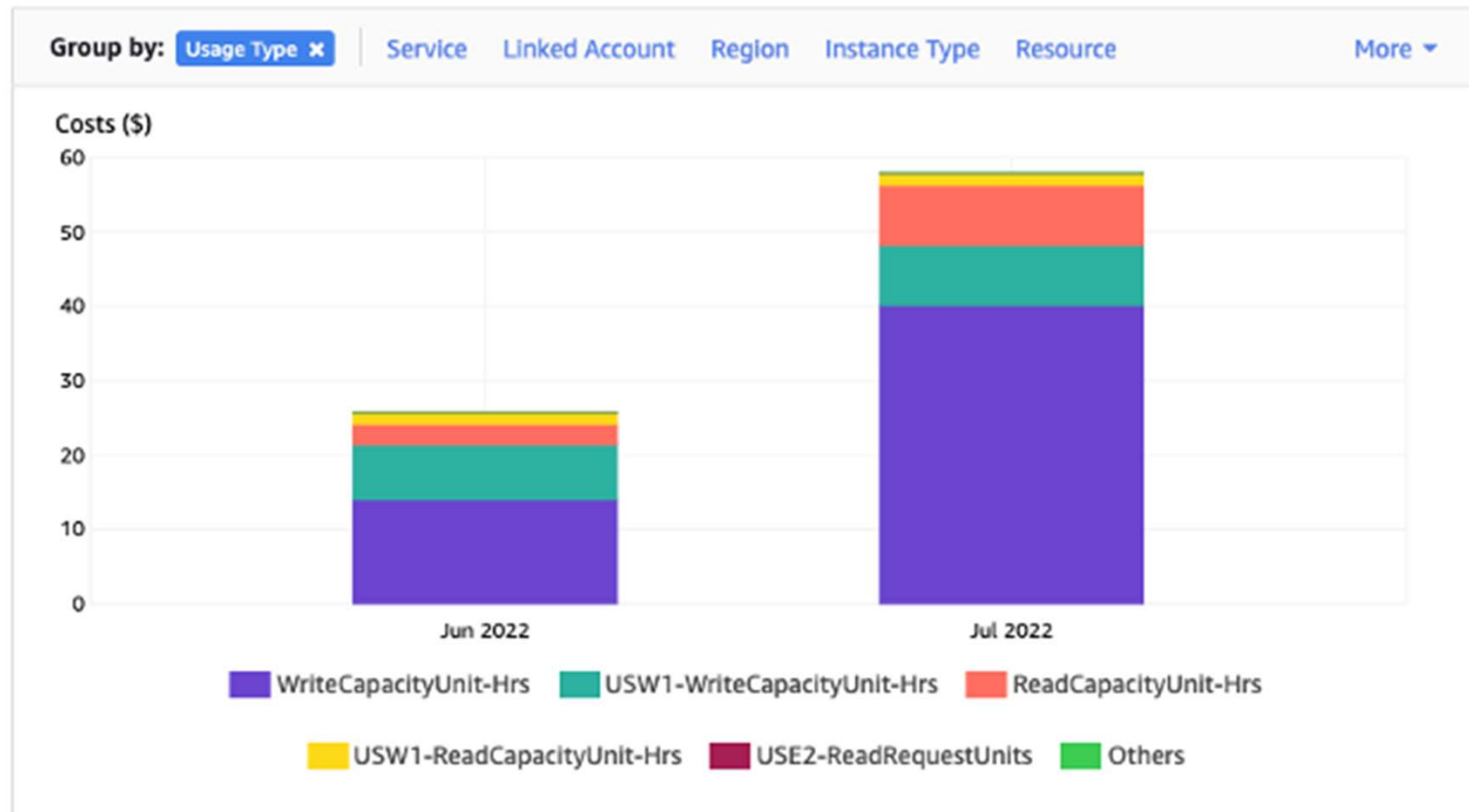
How to use and apply table tags in Cost Explorer

Set a tag with the key of table_name and the value of MyTable.

Activate the tag within Cost Explorer and then filter on the tag value to gain more visibility into each table's costs.

Evaluate your table's capacity mode

Should use the right capacity mode out of these 2

- On-demand capacity mode
- Provisioned capacity mode

Evaluate table's Auto Scaling settings

Defining the correct value for the target utilization, initial step, and final values is an activity that requires involvement from operations team.

When you set a high utilization target (a target around 90%) it means your traffic needs to be higher than 90% for a period of time before the Application Auto Scaling is activated.

You should only use a high utilization target when application is very constant and doesn't receive spikes in traffic.

When you set a very low utilization (a target less than 50%), it usually translates into unused capacity and wasted resources.

How to address workloads with seasonal variance

Users start to connect to the application before 9 AM and the capacity units increases considerably (the maximum value you've seen is 1500 WCU)

On average, your application usage varies between 800 to 1200 during working hours

If this scenario applies to your application, consider using scheduled application auto scaling

Identify your unused resources

Should ensure no resources are unused and contributing to unnecessary Amazon Keyspaces costs.

To identify unused tables you can take a look at the following CloudWatch metrics:

- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits

Cleaning up unused PITR backups

- PITR backups have costs associated with them.

Evaluate your table usage patterns

Perform fewer strongly-consistent read operations

- Strongly consistent reads are charged at 1 RCU for up to 4 KB of data, essentially doubling your read costs

Enable Time to Live (TTL)

Evaluate your provisioned capacity

- Should modify your operational procedures appropriately
    - Especially when your Amazon Keyspaces table is configured in provisioned mode and you have the risk to over-provision or under-provision your tables.
- To evaluate the table capacity, monitor the following CloudWatch metrics

Read Capacity Units	Write Capacity Units
ConsumedReadCapacityUnits	ConsumedWriteCapacityUnits
ProvisionedReadCapacityUnits	ProvisionedWriteCapacityUnits
ReadThrottleEvents	WriteThrottleEvents

Security best practices

Use encryption at rest

Use IAM roles to authenticate access to Amazon Keyspaces

Use IAM policies for Amazon Keyspaces base authorization

Consider client-side encryption

- Encrypting your sensitive data in transit

Use AWS CloudTrail to monitor

Use CloudTrail to monitor Amazon Keyspaces DDL operations

Tag your Amazon Keyspaces resources for identification and automation

Security aspects

Data protection

The AWS shared responsibility model applies to data protection in Amazon Keyspaces

AWS is responsible for protecting the global infrastructure

We are responsible for maintaining control over content that is hosted on this infrastructure

Protect AWS account credentials and set up individual users

Use multi-factor authentication (MFA) with each account.

Use SSL/TLS to communicate with AWS resources

Set up logging with AWS CloudTrail

Use advanced managed security services such as Amazon Macie

Encryption at rest

When creating a new table, you can choose one of the following AWS KMS keys (KMS keys):

- AWS owned key
- Customer managed key (Chargeable)

Can switch at any given time

Create an encrypted table

Open the Amazon Keyspaces console

In the navigation pane, choose Tables, and then choose Create table.

On the Create table page in the Table details section, select a keyspace and provide a name for the new table.

In the Schema section, create the schema for your table.

In the Table settings section, choose Customize settings.

Continue to Encryption settings.

In this step, you select the encryption settings for the table.

AWS Identity and Access Management

Control access in AWS by creating policies and attaching them to AWS identities

Example: Accessing Amazon Keyspaces tables

```
{
  "Version":"2012-10-17",
  "Statement":[
  {
  "Effect":"Allow",
  "Action":[
  "cassandra:Select"
  ],
  "Resource":[
  "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
  ]
  }
  ]
}
```

Logging and monitoring

With CloudWatch

With CloudTrail

- Captures Data Definition Language (DDL) API calls for Amazon Keyspaces as events

Keyspaces information in CloudTrail

CloudTrail eventName	Statement	CQL action	AWS SDK action
CreateTable	DDL	`CreateTable`	`CreateTable`
CreateKeyspace	DDL	`CreateKeyspace`	`CreateKeyspace`
DropKeyspace	DDL	`DropKeyspace`	`DeleteKeyspace`
DropTable	DDL	`DropTable`	`DeleteTable`
AlterTable	DDL	`AlterTable`	`UpdateTable`, `TagResource`, `UntagResource`
no CloudTrail events	DML	`Select`	`GetKeyspace`, `GetTable`, `ListKeyspaces`, `ListTables` `ListTagsForResource`

Resilience and disaster recovery in Amazon Keyspaces

With Availability Zones automatically fail over between Availability Zones without interruption.

Multi-Region Replication

- Can replicate your Amazon Keyspaces tables across up to six different AWS Regions

Point-in-time recovery (PITR)

Thanks