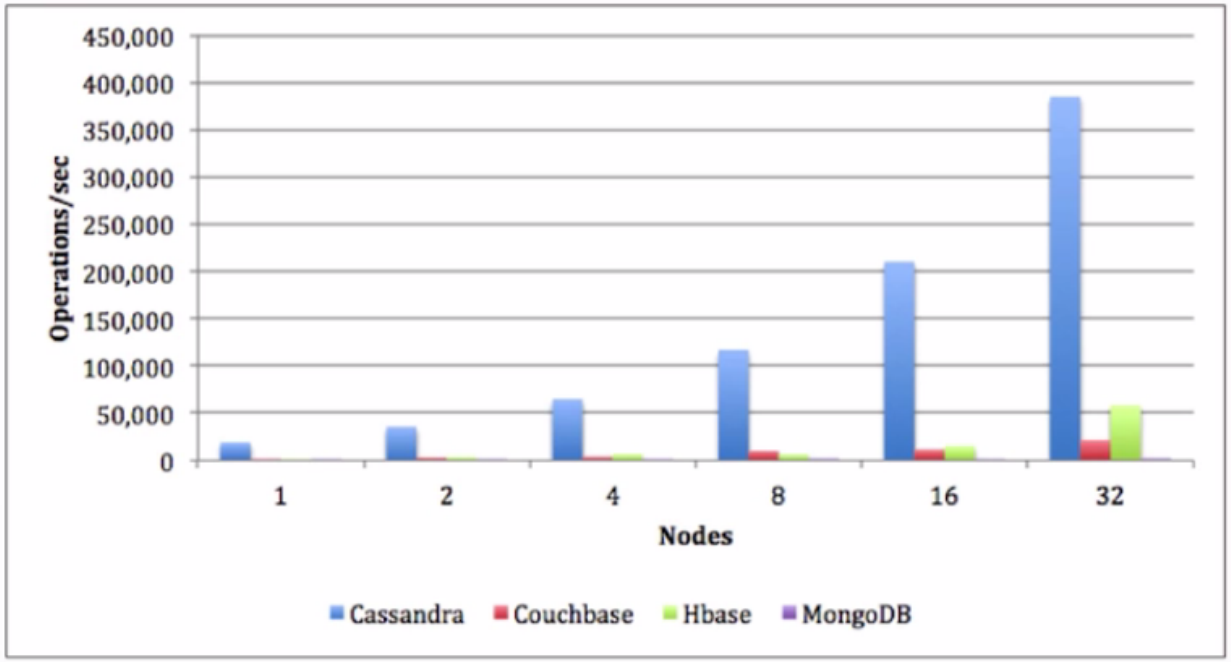
ionidea
CHARGED WITH CREATIVITY

Contents

Introduction to Apache Cassandra

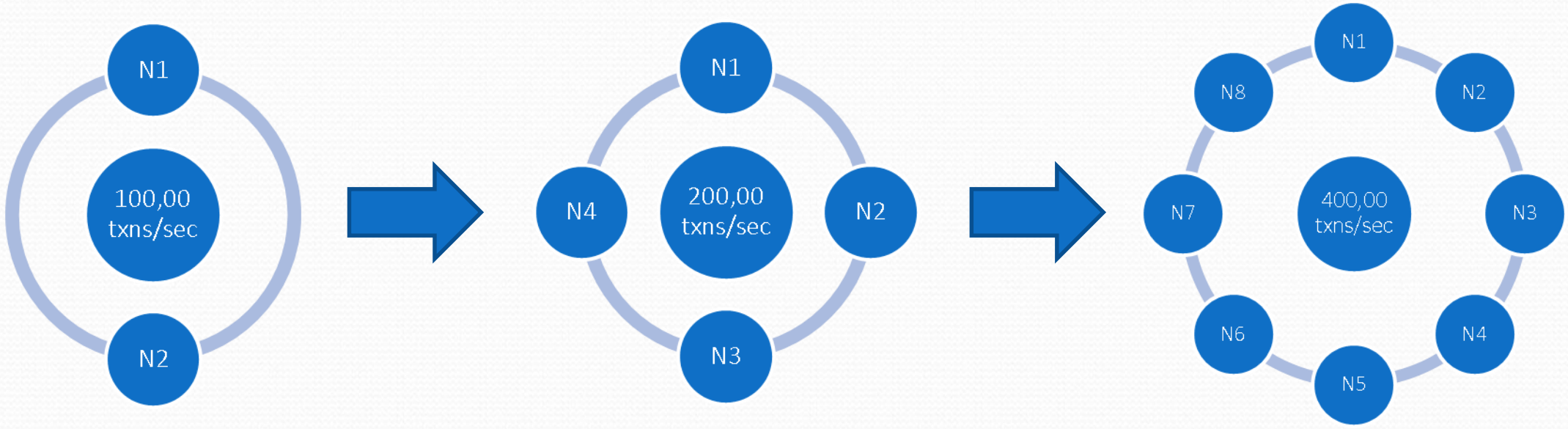
What is Cassandra?

- Massively Linearly scalable NoSQL database
  - Fully distributed, with no single point of failure
  - Free and open source, with deep developer support
  - Highly performant, with near-linear horizontal scaling in proper use cases



What is Cassandra?

- No Single point of failure, due to horizontal scaling
  - *horizontal scaling*: add commodity hardware to a cluster
  - *vertical scaling*: add RAM and CPU's to specialized high performance box



How Cassandra evolved?

- Cassandra can be seen as a combination of two familiar data stores
  - HBase (Google BigTable)
  - Amazon Dynamo
- Column Oriented Design of Hbase
  - One key per row
  - Columns, column families, …
- Distributed architecture of Amazon Dynamo
  - Partitioning, placement (consistent hashing)
  - Replication, gossip-based membership, anti-entropy

How Cassandra evolved?

It combines Amazon Dynamo's fully distributed design with Google Bigtable's column-oriented data model.

Amazon Dynamo	+	Google Big Table
Consistent hashing		Columnar
Gossip protocol		SSTable storage
Hinted handoff		Memtable
Read repair		Compaction
		Append only

History of Cassandra

- Cassandra was created to power the Facebook Inbox Search
- Facebook open-sourced Cassandra in 2008 and became an Apache Incubator project
- In 2010, Cassandra graduated to a top-level Apache project

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

Cassandra

- **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

General Design Features

- Emphasis on performance over analysis
  - Still has support for analysis tools such as Hadoop
- Organization
  - Rows are organized into tables
  - First component of a table's primary key is the partition key
  - Rows are clustered by the remaining columns of the key
  - Columns may be indexed separately from the primary key
  - Tables may be created, dropped, altered at runtime without blocking queries
- Language
  - CQL (Cassandra Query Language) introduced, similar to SQL (flattened learning curve)

Peer to Peer Cluster

- Decentralized design
  - Each node has the same role
- No single point of failure
  - Avoids issues of master-slave DBMS's
- No bottlenecking

Fault Tolerance/Durability

- **Failures happen all the time with multiple nodes**
  - Hardware Failure
  - Bugs
  - Operator error
  - Power Outage, etc.
- **Solution:** Buy cheap, redundant hardware, replicate, maintain consistency

Performance

- Core architectural designs allow Cassandra to outperform its competitors
- Very good read and write throughputs
  - Consistently ranks as fastest amongst comparable NoSql DBMS's with large data sets

"In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes…" - University of Toronto

Reasons for Choosing Cassandra

- Value availability over consistency
- Require high write-throughput
- High scalability required
- No single point of failure

Installation

Apache Cassandra

Download and extract the latest stable version of Apache Cassandra:

http://cassandra.apache.org/download/

Add below environment variables to .bashrc:

```
export CASSANDRA_HOME=/home/hduser/ionidea/apache-cassandra-2.1.9
export PATH=$PATH:$CASSANDRA_HOME/bin
export CLASSPATH=$CLASSPATH:/home/hadoop/Cassandra_jars/*
```

Note: Copy the below Cassandra_jars folder into the sandbox path /home/hduser/ionidea/Cassandra and we need the jars files for Java API

/Module 10 - Cassandra/Installation and Configuration/Cassandra_jars

Create Cassandra lib and log folders:

```
$ sudo mkdir /var/lib/cassandra
$ sudo mkdir /var/log/cassandra

$ sudo chmod 777 /var/lib/cassandra
$ sudo chmod 777 /var/log/cassandra
```

Start Cassandra Service:

```
$ cassandra -f
```

Cassandra OpsCenter package

Modify the aptitude repository source list file (/etc/apt/sources.list.d/datastax.community.list).

$ echo "deb http://debian.datastax.com/community stable main" | sudo tee -a /etc/apt/sources.list.d/datastax.community.list

Add the DataStax repository key to your aptitude trusted keys:

$ curl -L http://debian.datastax.com/debian/repo_key | sudo apt-key add -

Install the OpsCenter package using the APT Package Manager:

$ sudo apt-get update

$ sudo apt-get install opscenter

For most users, the out-of-box configuration should work just fine, but if you need to you can configure OpsCenter differently.

Start OpsCenter:

$ sudo service opscenterd start

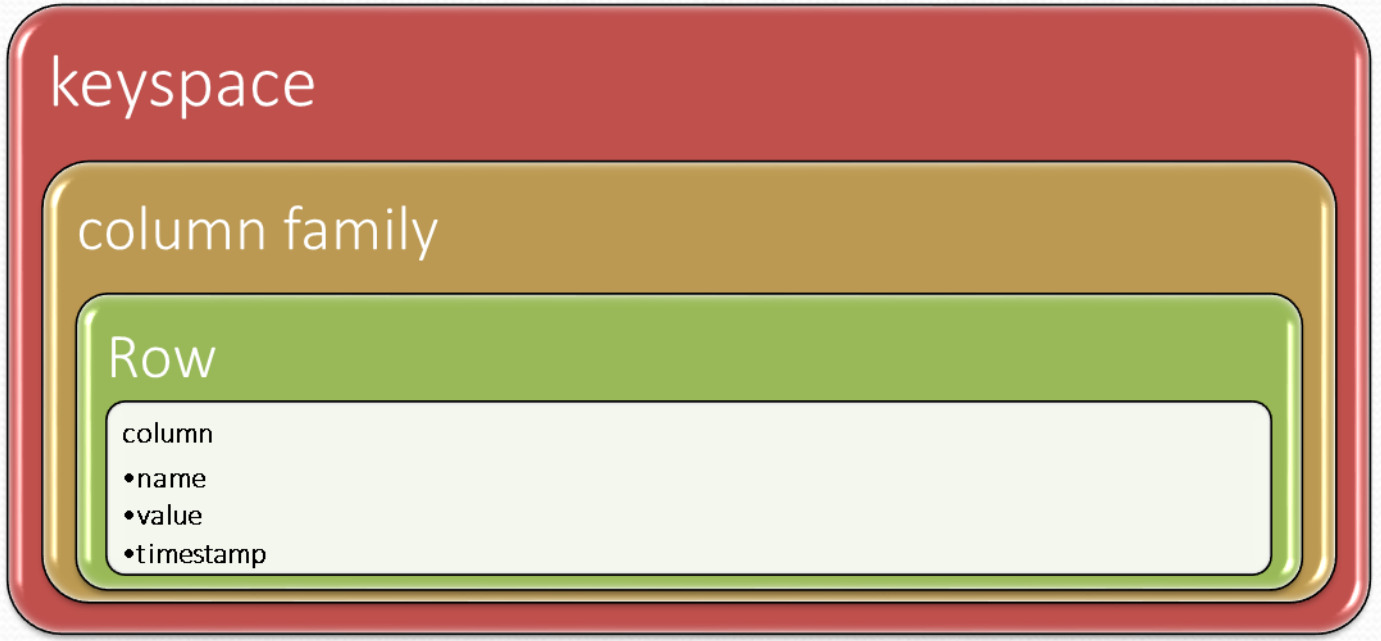
Connect to OpsCenter in a web browser using the following URL:

http://localhost:8888/

Next you can add an existing cluster or provision a new one, please refer to below link for further reading

Data Model

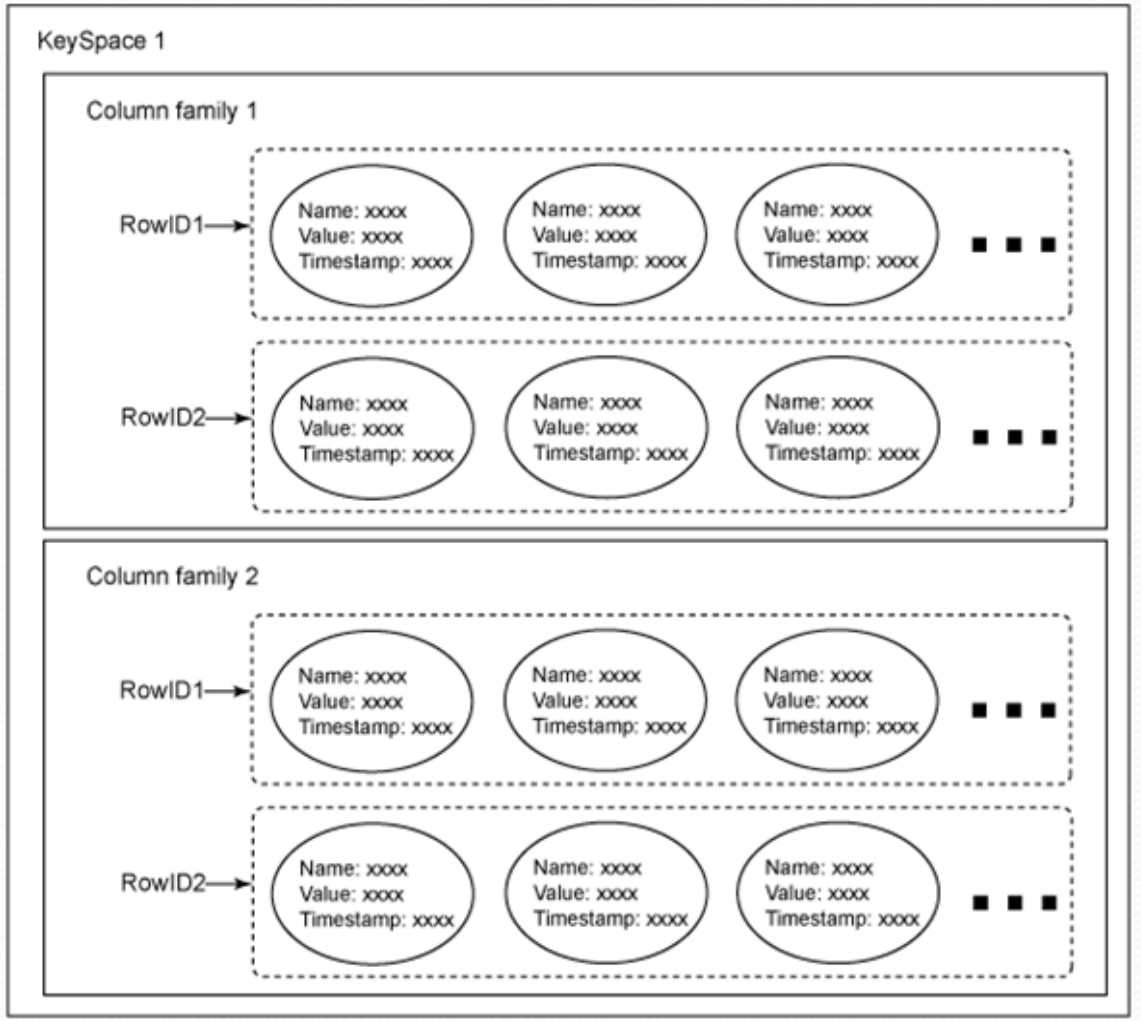
Data Model



Cassandra	RDBMS Equivalent
KEYSPACE	DATABASE/SCHEMA
COLUMN FAMILY	TABLE
ROW	ROW
FLEXIBLE COLUMNS	DEFINED COLUMNS

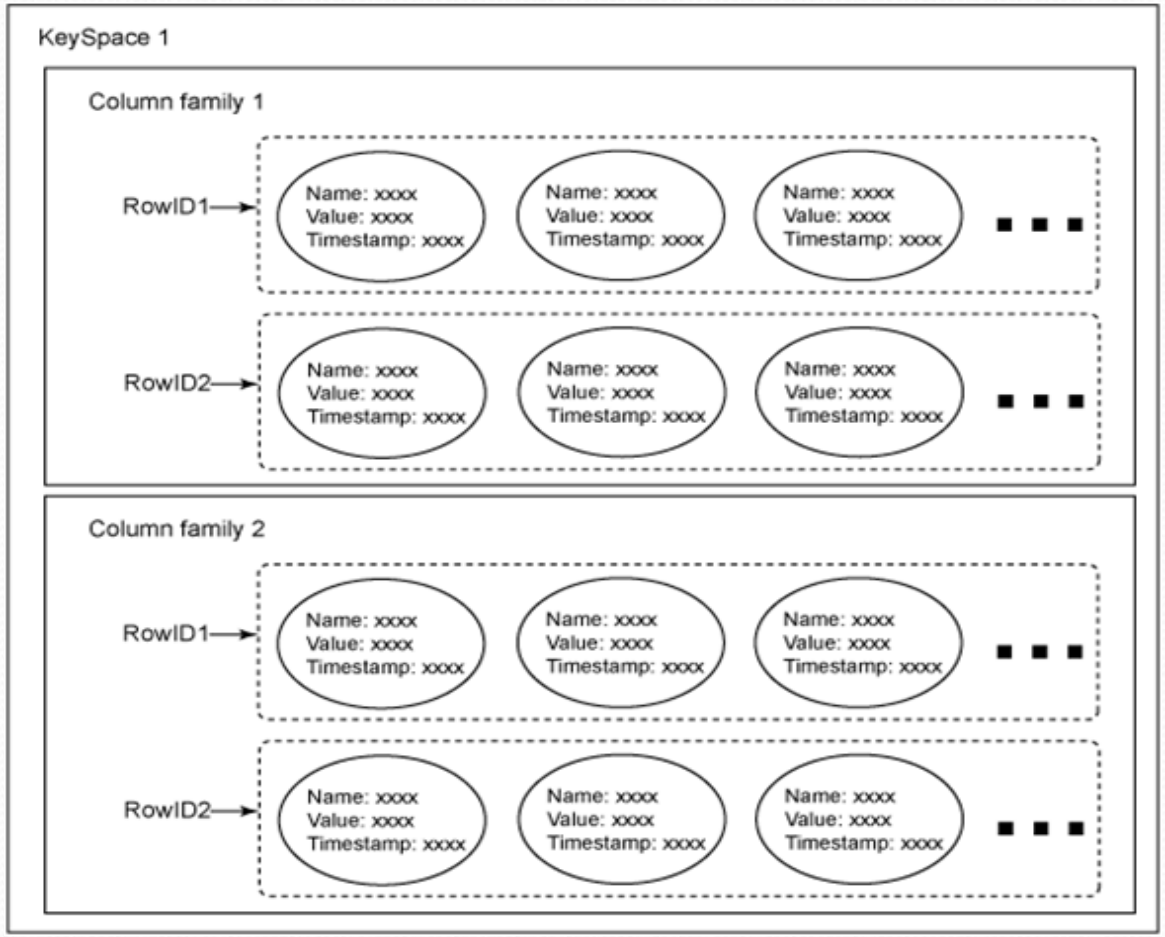
Key-Value Model

- Cassandra is a column oriented NoSQL system
- Column families: sets of key-value pairs
  - column family as a table and key-value pairs as a row (using relational database analogy)
- A row is a collection of columns labeled with a name

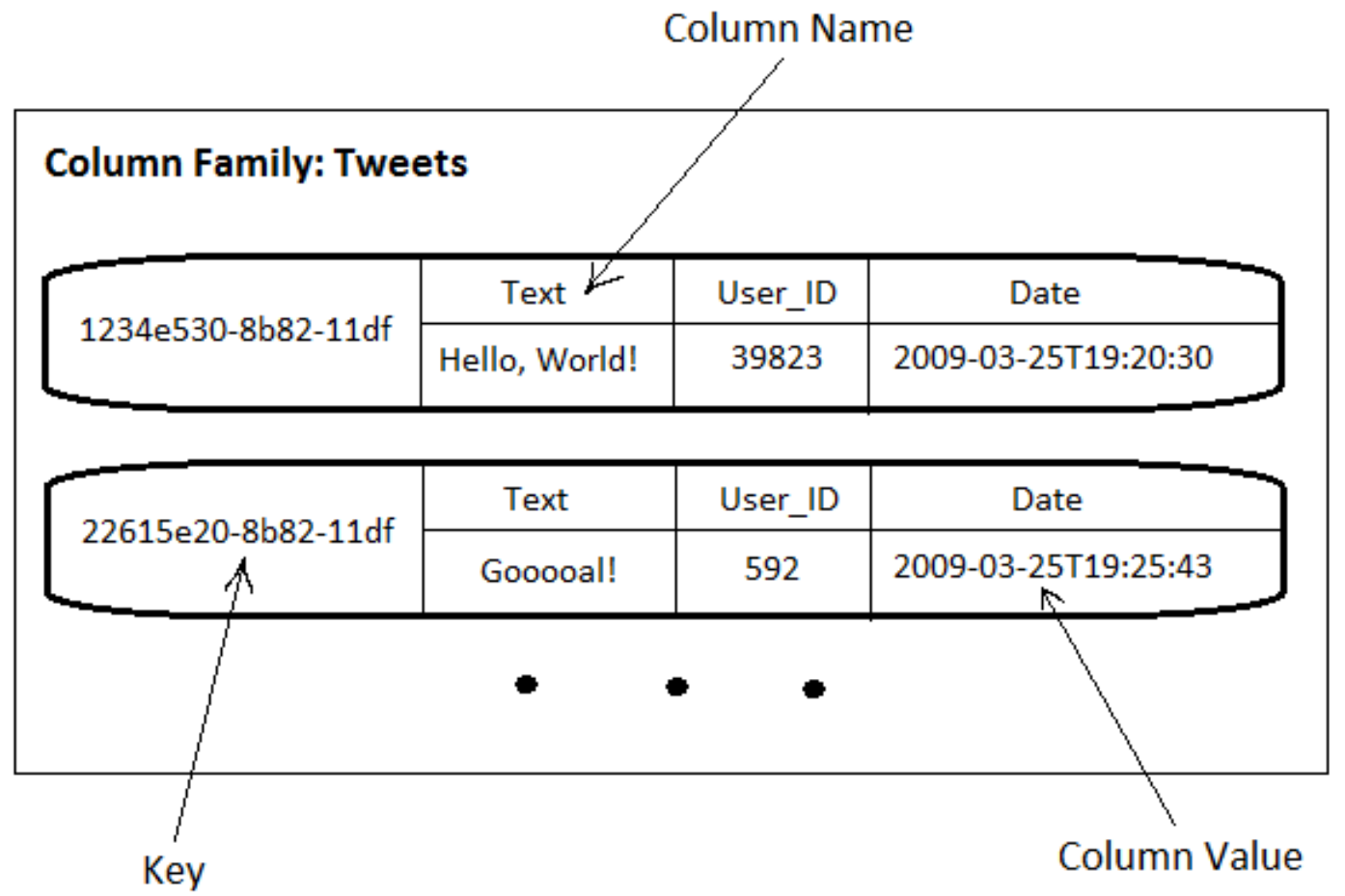


Cassandra Row

- the value of a row is itself a sequence of key-value pairs
- such nested key-value pairs are columns
- key = column name
- a row must contain at least 1 column

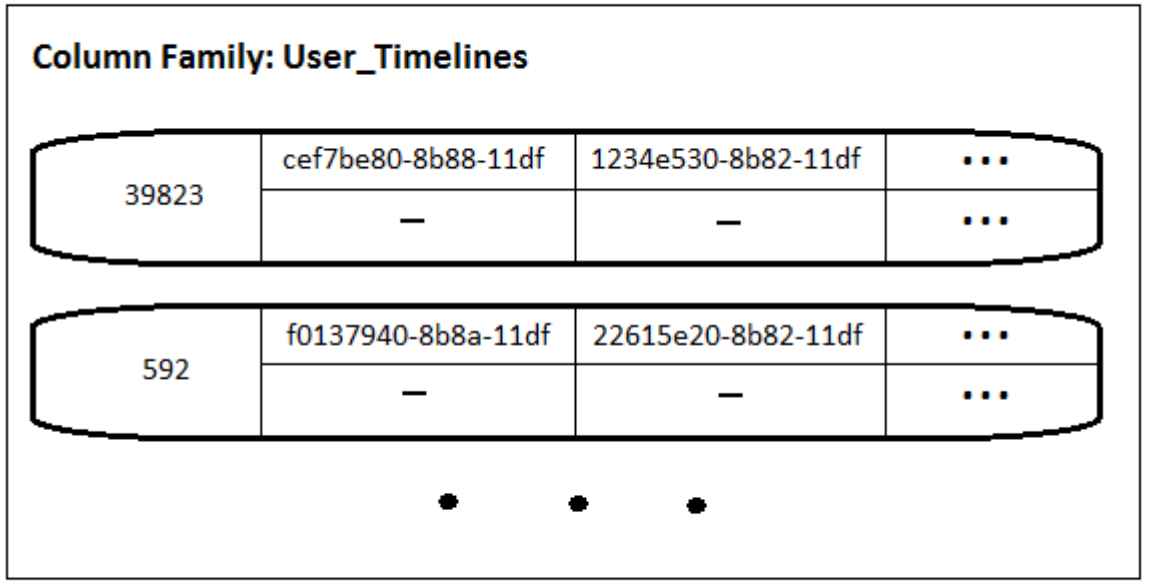


Example of Columns

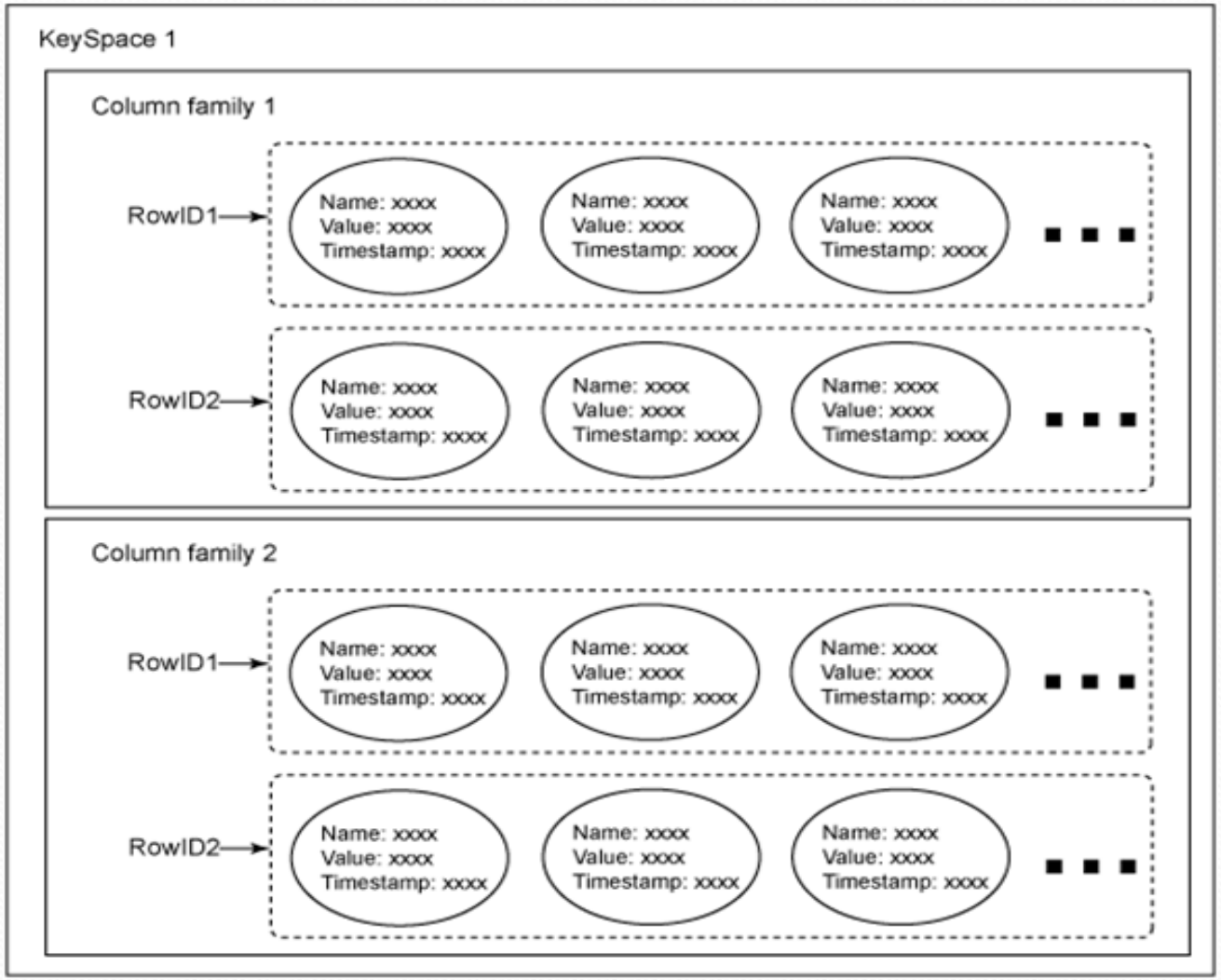


Column names storing values

- key: User ID
- column names store tweet ID values
- values of all column names are set to “-” (empty byte array) as they are not used



KeySpace

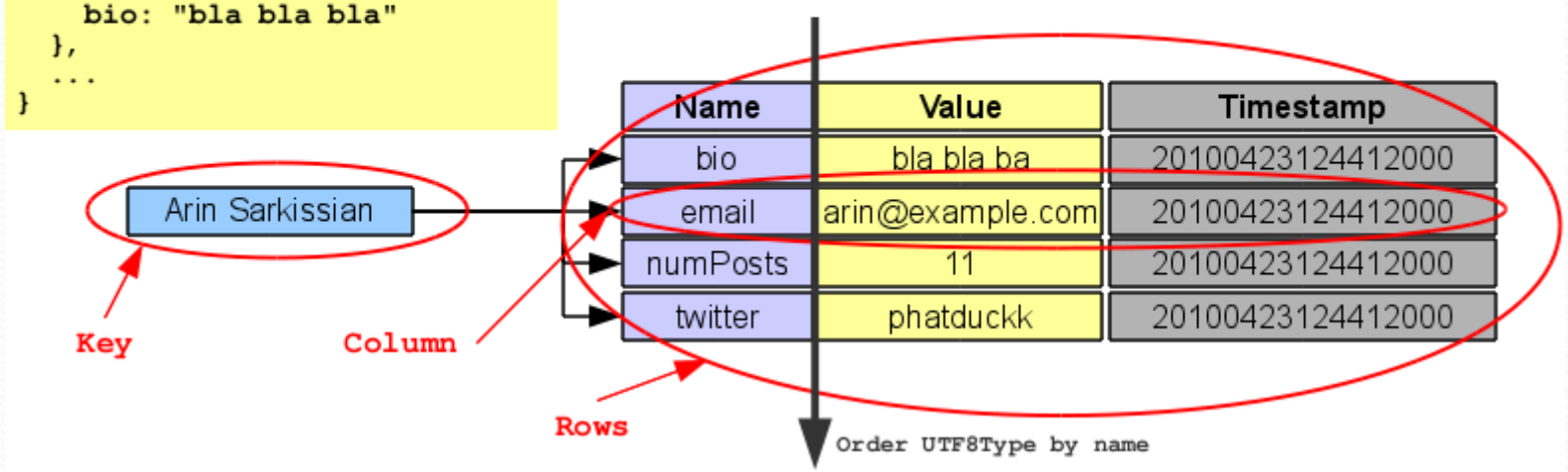


- A Key Space is a group of column families together.
- It is only a logical grouping of column families and provides an isolated scope for names

Column Family

```
Authors : {
  Arin Sarkissian : {
    numPosts: 11,
    twitter: phatduckk,
    email: arin@example.com,
    bio: "bla bla bla"
  },
  ...
}
```

```
<ColumnFamily CompareWith="UTF8Type" Name="Authors"/>
```

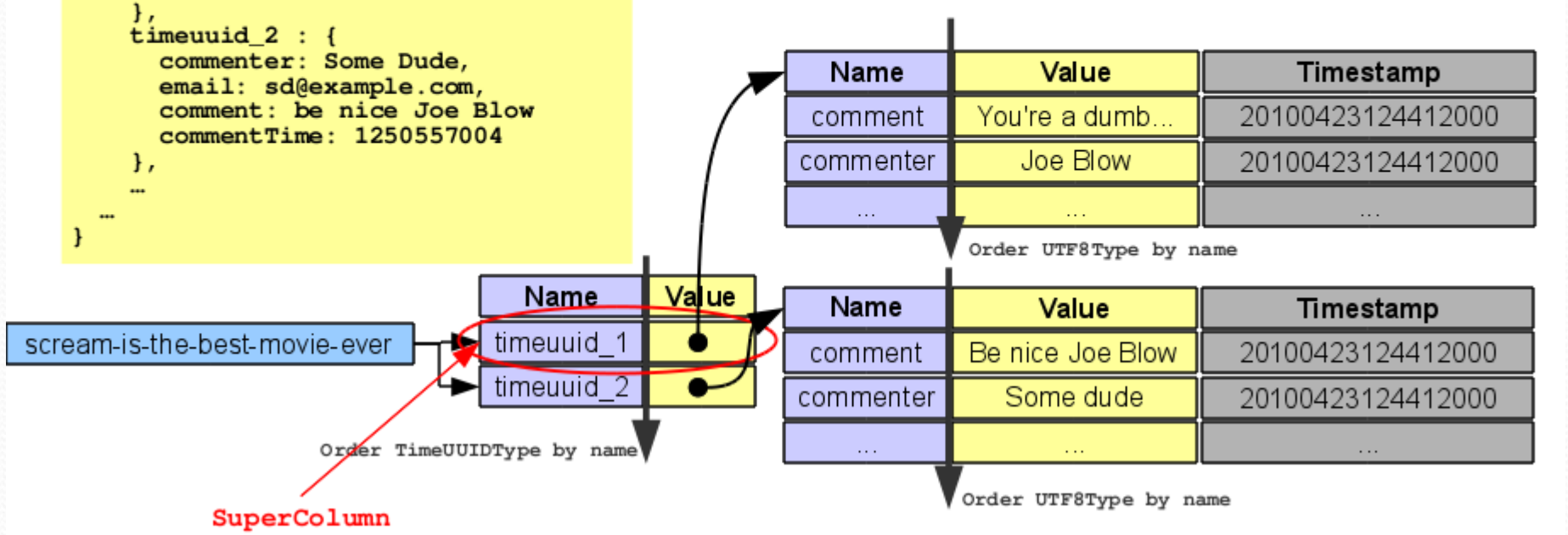


The first column family *Authors* is just a normal (not super) column family. Keys for this family are the author full names and every author/key has some columns to store some info about him (email, twitter nickname or whatever)

Super Column Family

```
Comments : {
  scream-is-the-best-movie-ever : {
    timeuuid_1 : {
      commenter: Joe Blow,
      email: joeb@example.com,
      comment: you're a dumb douche.
      commentTime: 1250438004
    },
    timeuuid_2 : {
      commenter: Some Dude,
      email: sd@example.com,
      comment: be nice Joe Blow
      commentTime: 1250557004
    },
    …
  …
}
```

```
<ColumnFamily CompareWith="TimeUUIDType" ColumnType="Super"
    CompareSubcolumnsWith="UTF8Type" Name="Comments"/>
```



Comments is a super column. The row key for the super column is the blog entry slug, the super columns are the comments for the blog entry. Each super column has the timestamp (the time when the comment was done) as name and the values are the data of each comment as column rows (commenter, commenter email, comment text itself,...)

How does Cassandra model data?

- Cassandra Query Language (CQL)
  - Provides a familiar, row-column. SQL-like approach
    - CREATE.ALTER. DROP
    - SELECT. INSERT. UPDATE. DELETE
  - Replaced the complex, storage-oriented Thrift API used in prior versions
  - Provides clear schema definitions in a flexible (NoSQL) schema context

```
CREATE TABLE Performer (
name VARCHAR ,
type VARCHAR,
country VARCHAR,
style VARCHAR,
born INT,
died INT,
PRIMARY KEY (name)
);
```

Cassandra Data Types

Data Type	Constants	Description
ascii	strings	Represents ASCII character string
bigint	bigint	Represents 64-bit signed long
blob	blobs	Represents arbitrary bytes
Boolean	booleans	Represents true or false
counter	integers	Represents counter column
decimal	integers, floats	Represents variable-precision decimal
double	integers	Represents 64-bit IEEE-754 floating point
float	integers, floats	Represents 32-bit IEEE-754 floating point
int	integers	Represents 32-bit signed int
text	strings	Represents UTF8 encoded string
timestamp	integers, strings	Represents a timestamp
varchar	strings	Represents uTF8 encoded string
varint	integers	Represents arbitrary-precision integer

Cassandra Collection Types

Collection	Description
list	A list is a collection of one or more ordered elements.
map	A map is a collection of key-value pairs.
set	A set is a collection of one or more elements.

Cassandra Query Language - CQL

creating a *keyspace* - namespace of tables

```
CREATE KEYSPACE demo
WITH replication = {‘class’: ’SimpleStrategy’,
‘replication_factor’: 3};
```

to use namespace:

```
USE demo;
```

Cassandra Query Language - CQL

Creating tables:

```
CREATE TABLE users(
  email varchar,
  bio varchar,
  birthday timestamp,
  active boolean,
  PRIMARY KEY (email));
```

```
CREATE TABLE tweets(
  email varchar,
time_posted timestamp,
    tweet varchar,
  PRIMARY KEY (email, time_posted));
```

Cassandra Query Language - CQL

Inserting data

```
INSERT INTO users (email, bio, birthday, active)
VALUES ('john.doe@bti360.com', 'BT360 Teammate',
    516513600000, true);
```

** timestamp fields are specified in *milliseconds since epoch*

Cassandra Query Language - CQL

Querying tables

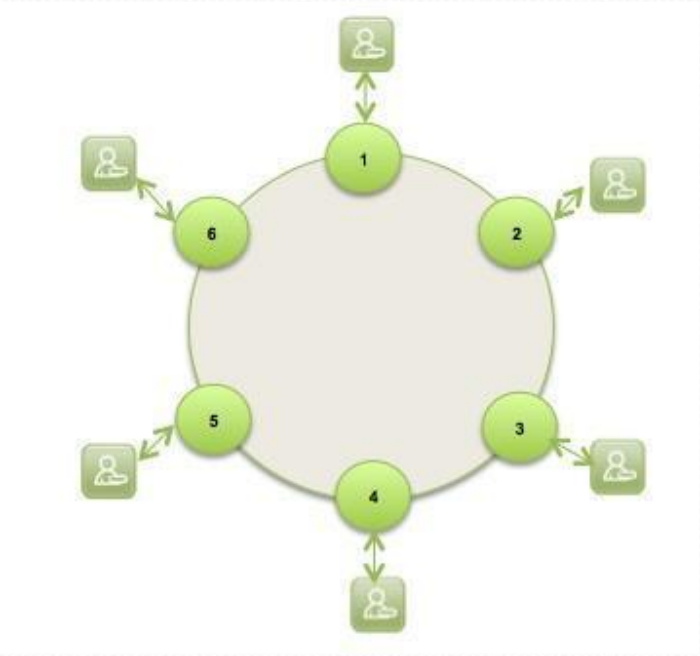
SELECT expression reads one or more records from Cassandra column family and returns a result-set of rows

```
SELECT * FROM users;

SELECT email FROM users WHERE active = true;
```

Architecture

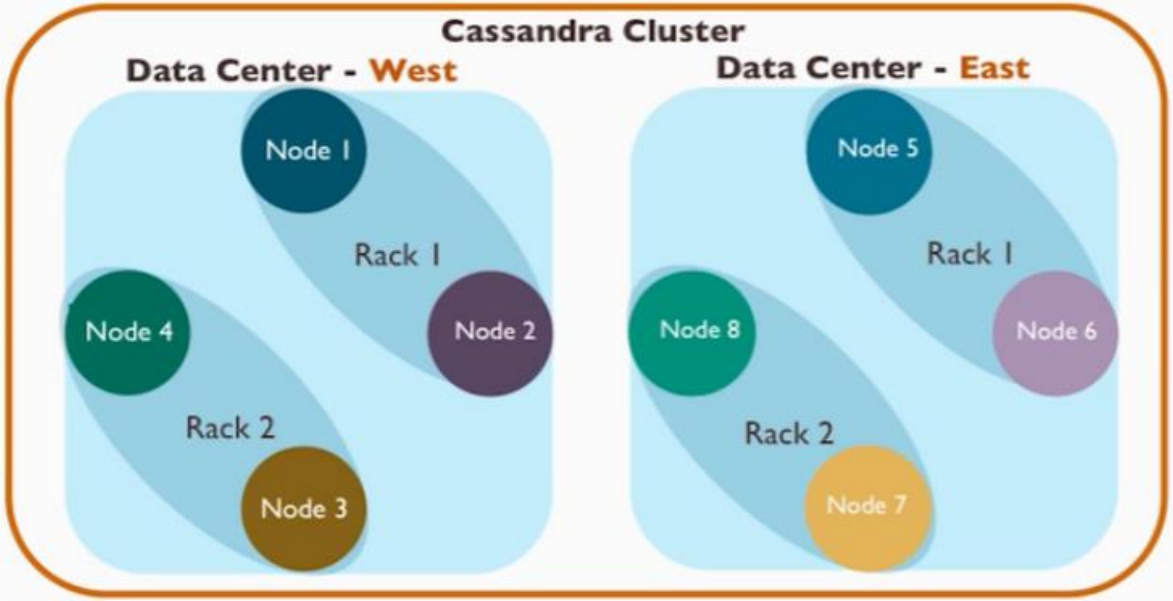
Cassandra Architecture - Overview

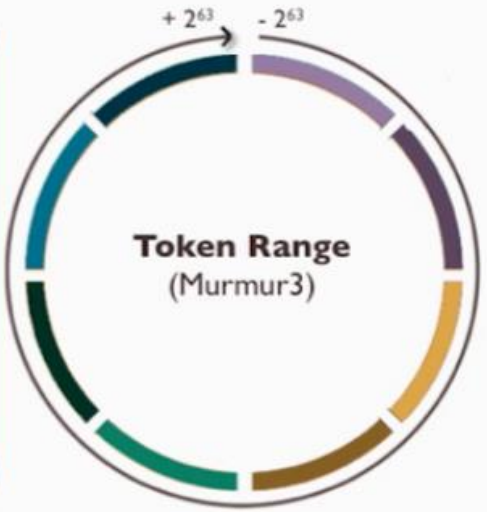
- Cassandra was designed with the understanding that system/ hardware failures can and do occur
- Peer-to-peer, distributed system
- All nodes are the same, Data partitioned among all nodes in the cluster
- Custom data replication to ensure fault tolerance
- Read/Write-anywhere design
- Google BigTable - data model
  - Column Families
  - Memtables
  - SSTables
- Amazon Dynamo - distributed systems technologies
  - Consistent hashing
  - Partitioning
  - Replication
  - One-hop routing



What is a cluster?

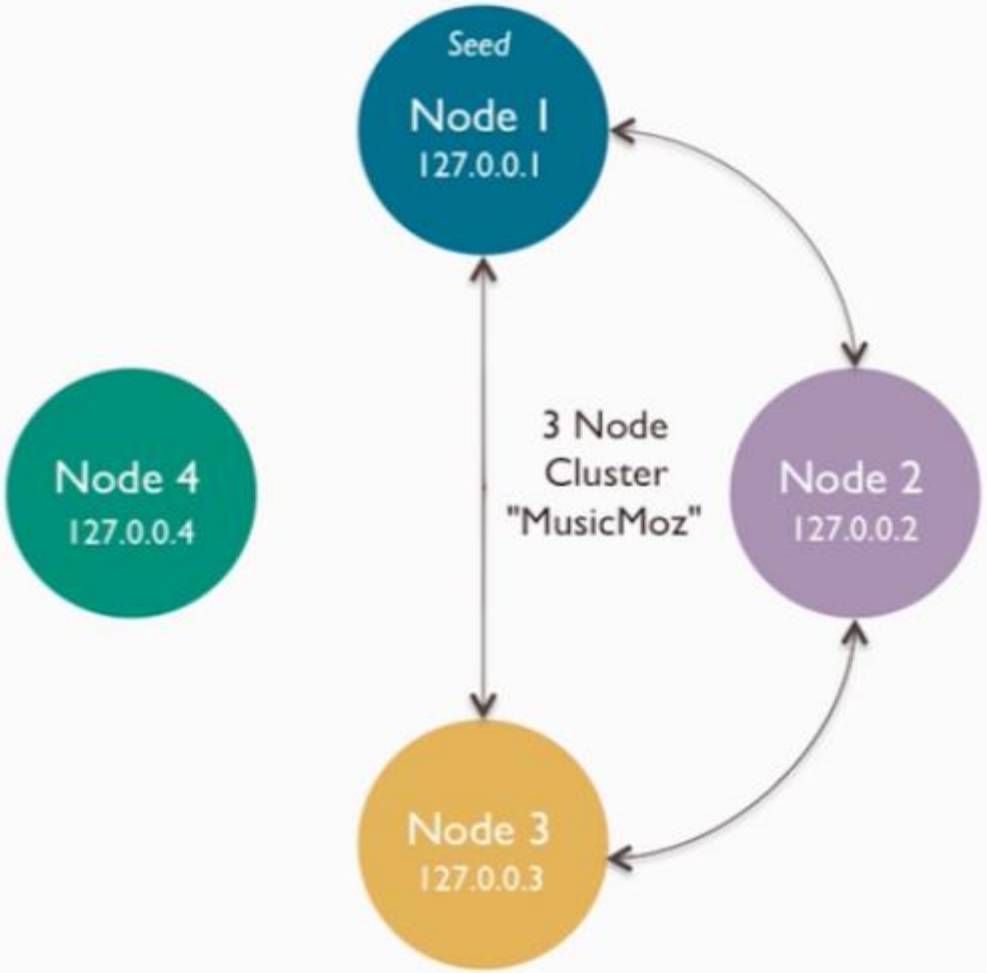
- A peer to peer set of nodes
  - Node – one Cassandra instance
  - Rack – a logical set of nodes
  - Data Center – a logical set of racks
  - Cluster – the full set of nodes which map to a single complete token ring





What is a cluster?

- Nodes join a cluster based on the configuration of their configuration file – conf/cassandra.yaml
- Key settings include
  - cluster_name – shared name to logically distinguish a set of nodes
  - seeds – IP addresses of initial nodes for a new node to contact and discover the cluster topology
  - listen_address – IP address through which this particular node communicates



Token Calculation

- Initial Tokens decide who is "responsible for" data.

The formula to calculate the ideal Initial Tokens is:

Initial_Token= Zero_Indexed_Node_Number * ((2^127) / Number_Of_Nodes)

For a five node cluster. the initial token for the 3rd node would be:

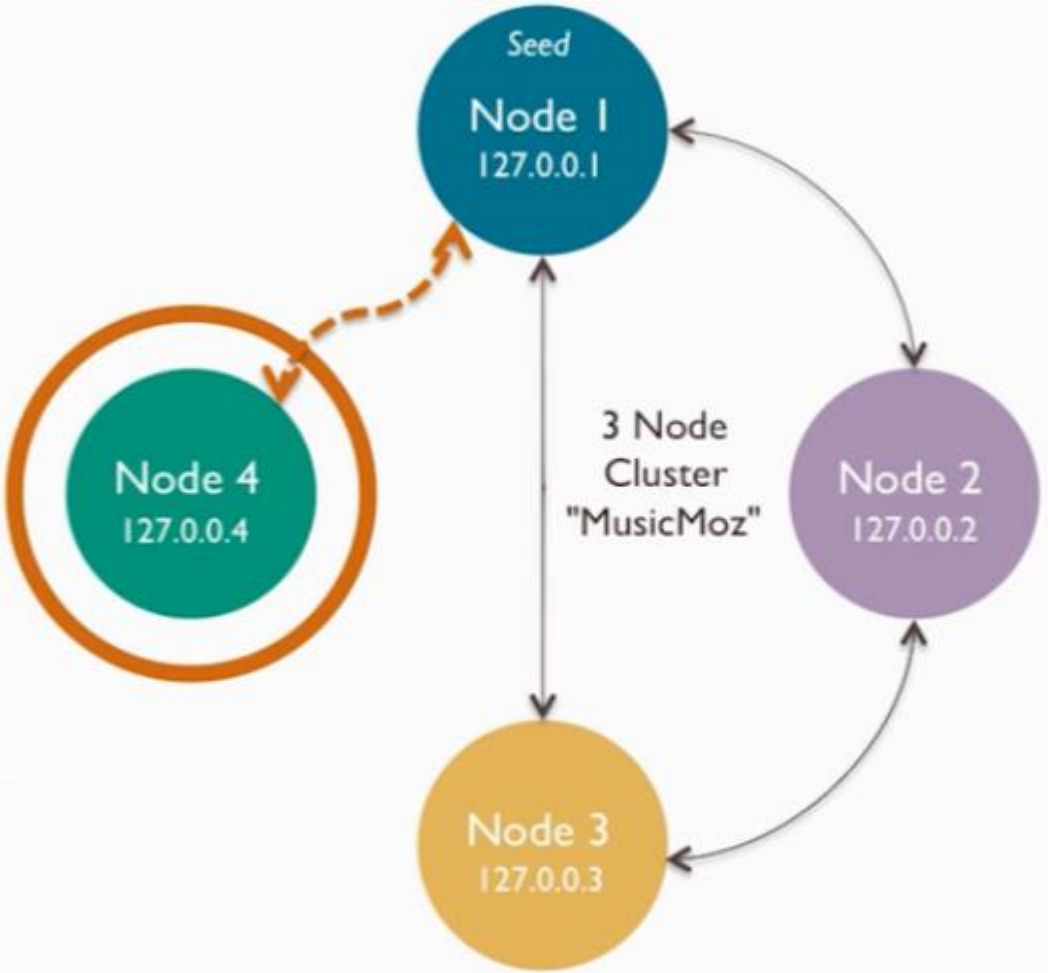
initial token=2 * ((2^127) / 5)

initial token=184467440737095162

Reference: http://www.geroba.com/cassandra/cassandra-token-calculator/

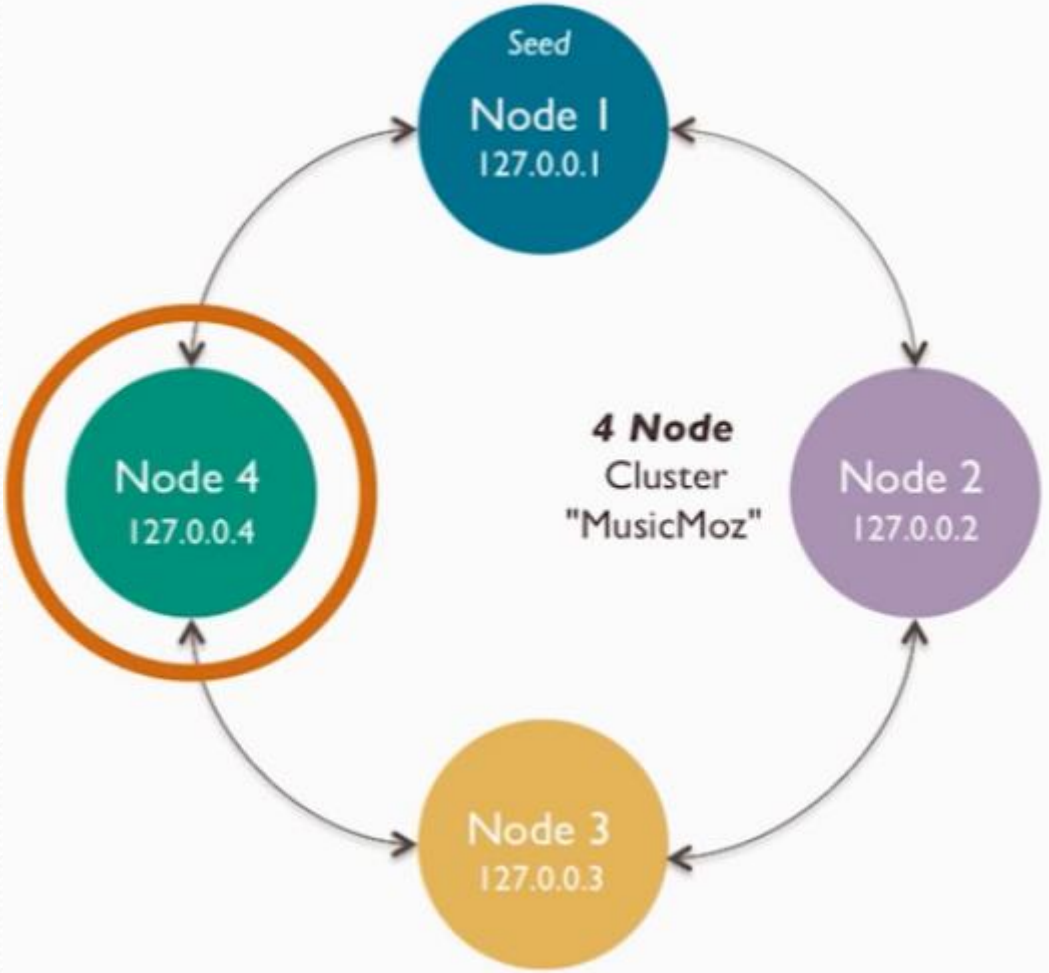
What is a cluster?

- Nodes join a cluster based on the configuration of their configuration file – conf/cassandra.yaml
- Key settings include
  - cluster_name – shared name to logically distinguish a set of nodes
  - seeds – IP addresses of initial nodes for a new node to contact and discover the cluster topology
  - listen_address – IP address through which this particular node communicates



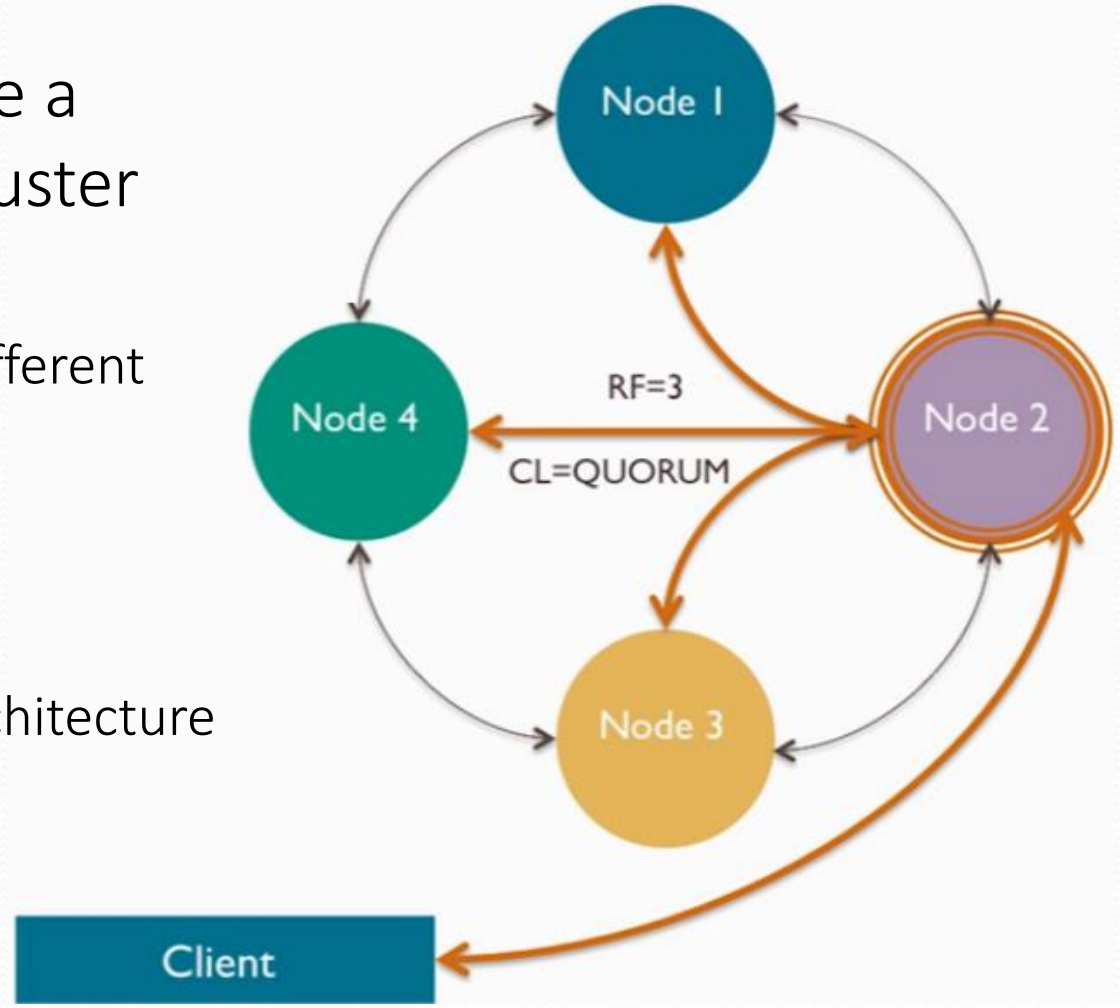
What is a cluster?

- Nodes join a cluster based on the configuration of their configuration file – conf/cassandra.yaml
- Key settings include
  - cluster_name – shared name to logically distinguish a set of nodes
  - seeds – IP addresses of initial nodes for a new node to contact and discover the cluster topology
  - listen_address – IP address through which this particular node communicates



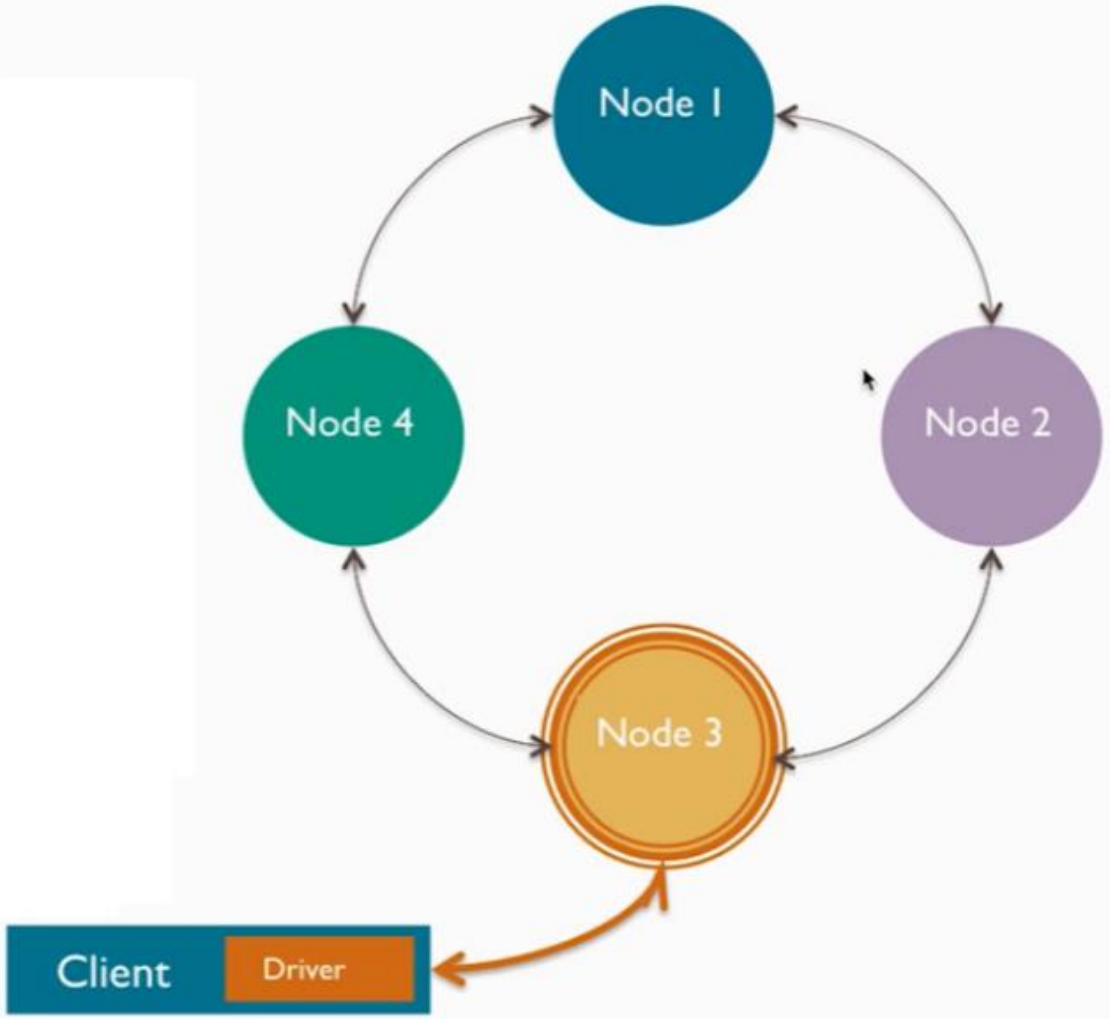
What is a coordinator?

- The node chosen by the client to receive a particular read or write request to its cluster
  - A node can coordinate any request
  - Each client request may be coordinated by a different node
- No single point of failure
  - This principle is fundamental to Cassandra's architecture



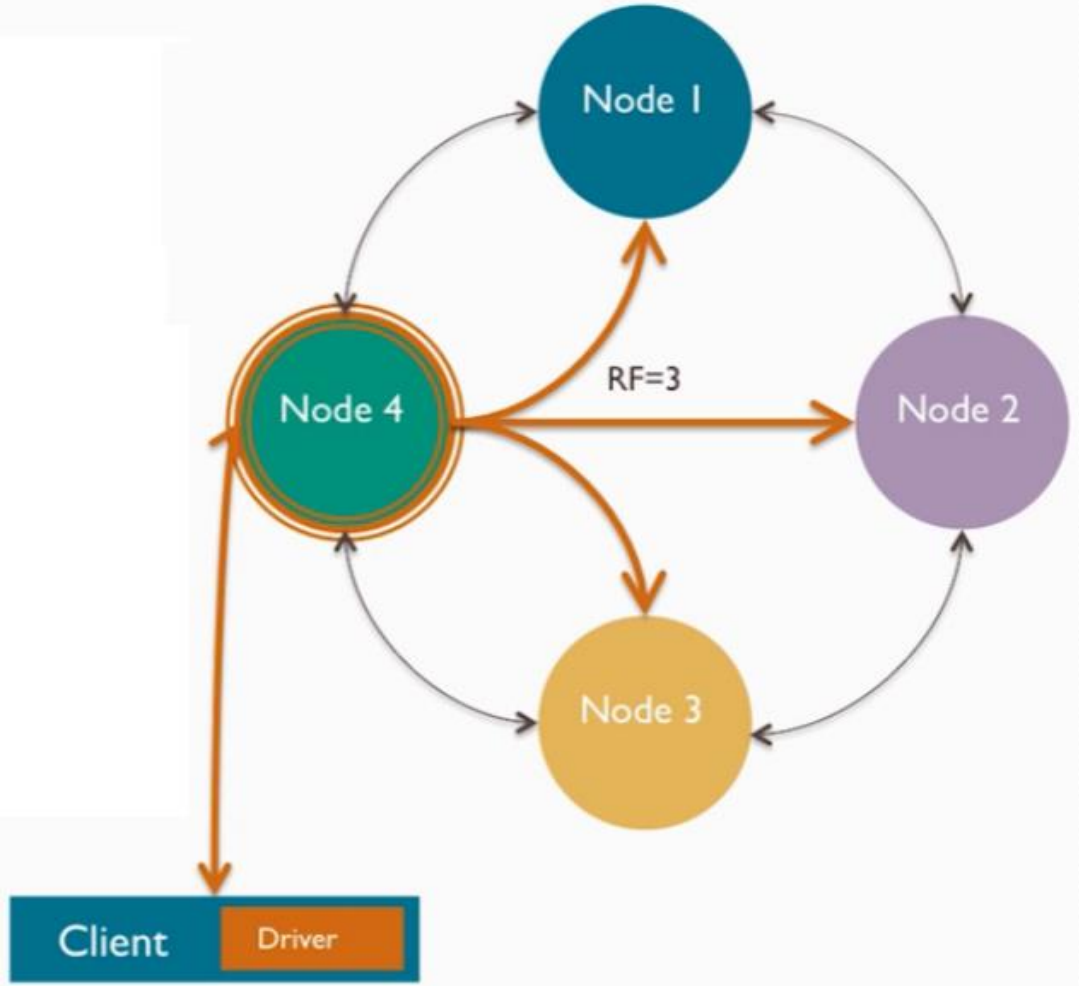
How are client requests coordinated?

- The Cassandra driver chooses the node to which each read or write request is sent
  - Client library providing APIs to manage client read/write requests
  - Default policy is TokenAware/DCAWARE
  - DataStax maintains open source drivers for Java, C#, Python, Node.js, Ruby, C/C++
  - Cassandra Community maintains drivers for PHP, Perl, Go, Clojure, Haskell, R, Scala



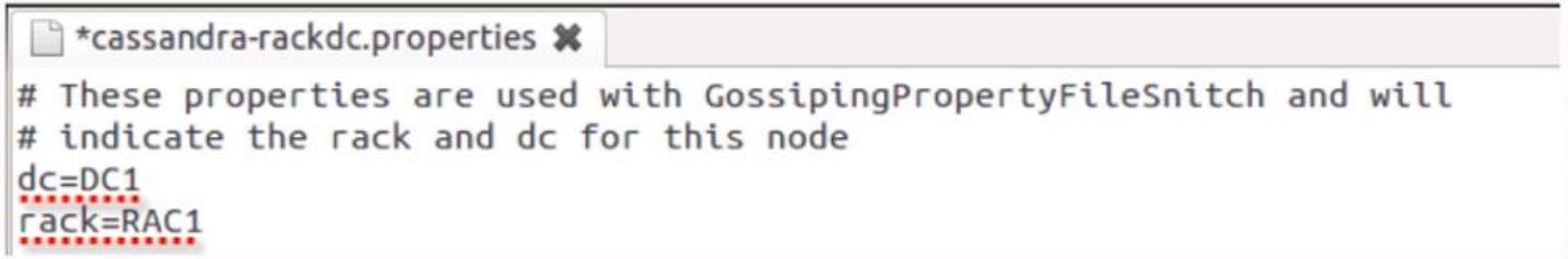
How are client requests coordinated?

- The coordinator manages the Replication Factor (RF)
  - **Replication factor (RF)** - onto how many nodes should a write be copied?
  - Possible values range from I to the total of planned nodes for the cluster
  - RF is set for an entire keyspace, or for each data center, if multiple
- Every write to every node is individually time-stamped



How are nodes organized as racks and data centers?

- A cluster of nodes can be logically grouped as racks and data centers
  - **Node** - the virtual or physical host of a single Cassandra instance
  - **Rack** - a logical grouping of physically related nodes
  - **Data Center** - a logical grouping of a set of racks
- Enables geographically aware read and write request routing
  - Cluster topology is communicated by Gossip protocol
- Each node belongs to one rack in one data center
  - A default Cassandra node belongs to rack1 in datacenter1
- The identity of each node's rack and data center may be configured in a property file



How does the keyspace impact replication?

- Replication factor is configured when a keyspace is created
  - SimpleStrategy (learning use only) - one factor for entire cluster
    - assigned as "replication_factor”

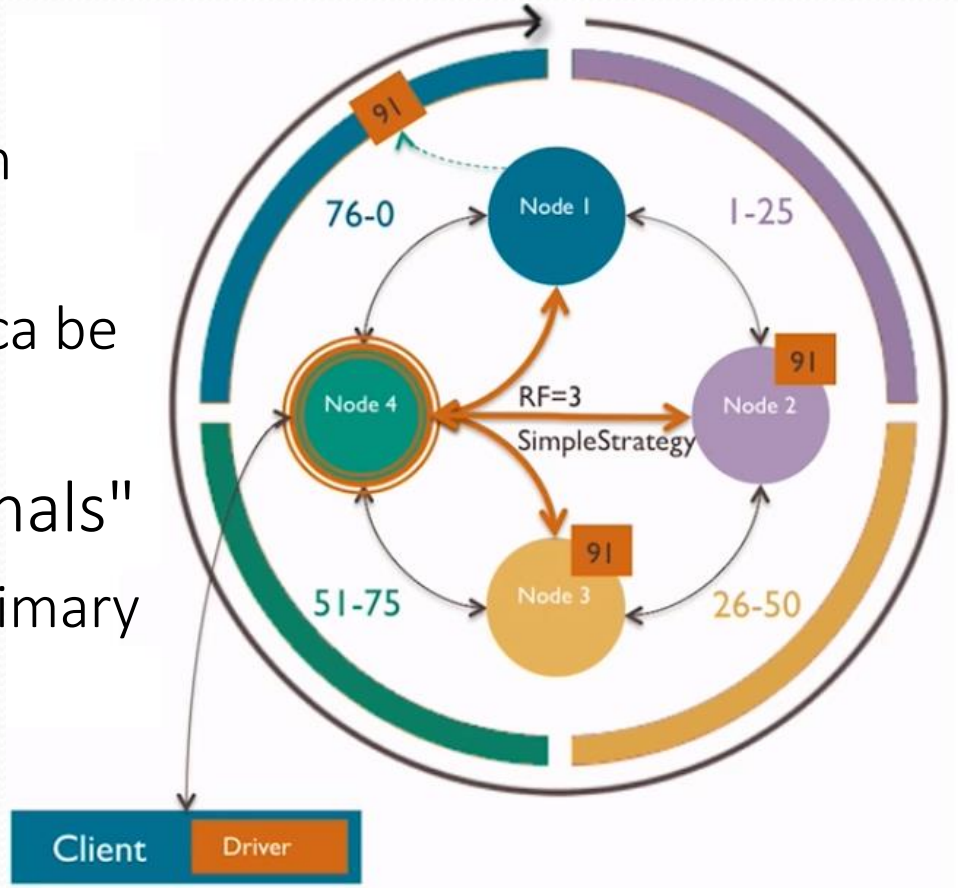
```
CREATE KEYSPACE simple-demo
WITH REPLICATION =
{'class':'SimpleStrategy',
'replication_factor':2}
```

- NetworkTopologyStrategy - separate factor for each data center in cluster
  - assigned by data center id (as also used in cassandra-rackdc.properties)

```
CREATE KEYSPACE simple-demo
WITH REPLICATION =
{'class':’NetworkTopologyStrategy',
'dc-east’:2, 'dt-west’:3}
```

How does a coordinator forward write requests?

- The target table's keyspace determines
  - **Replication factor** - how many replicas to make of each partition
  - **Replication strategy** - on which node should each replica be placed
- All partitions are "replicas", there are no "originals"
  - **First replica** - placed on the node owning its token's primary range
  - **(Subsequent) replicas** (if RF > 1) - placed in "secondary range" of other nodes, per the *replication strategy*

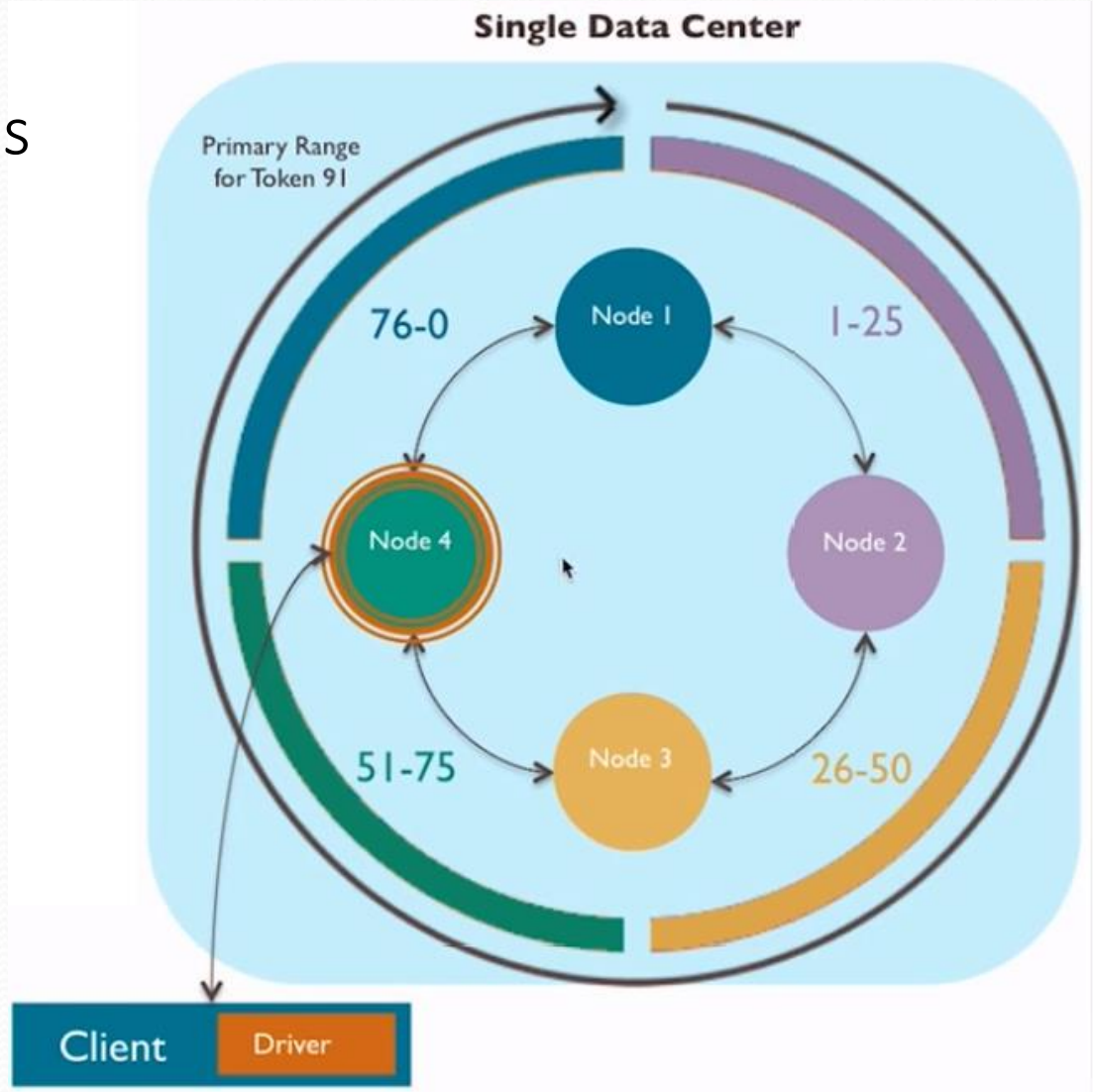


How is data replicated among nodes?

- **SimpleStrategy** - create replicas on nodes subsequent to the primary range node

  CREATE KEYSPACE demo WITH REPLICATION =
  {'class':'**SimpleStrategy**',
  'replication_factor':**3**}

- replication factor of 3 is a recommended minimum

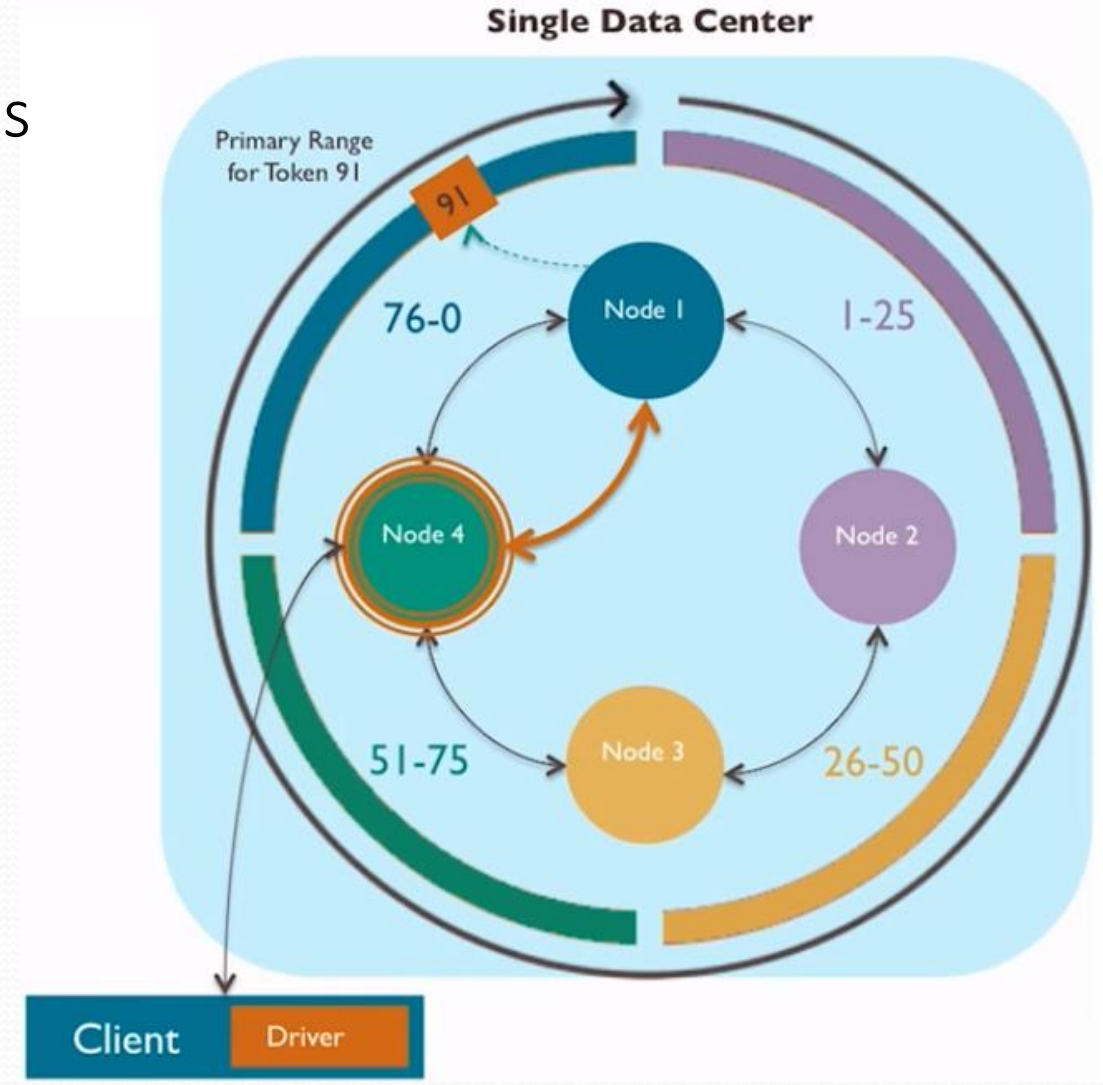


How is data replicated among nodes?

- **SimpleStrategy** - create replicas on nodes subsequent to the primary range node

  CREATE KEYSPACE demo WITH REPLICATION =
  {'class':'**SimpleStrategy**',
  'replication_factor':**3**}

- replication factor of 3 is a recommended minimum

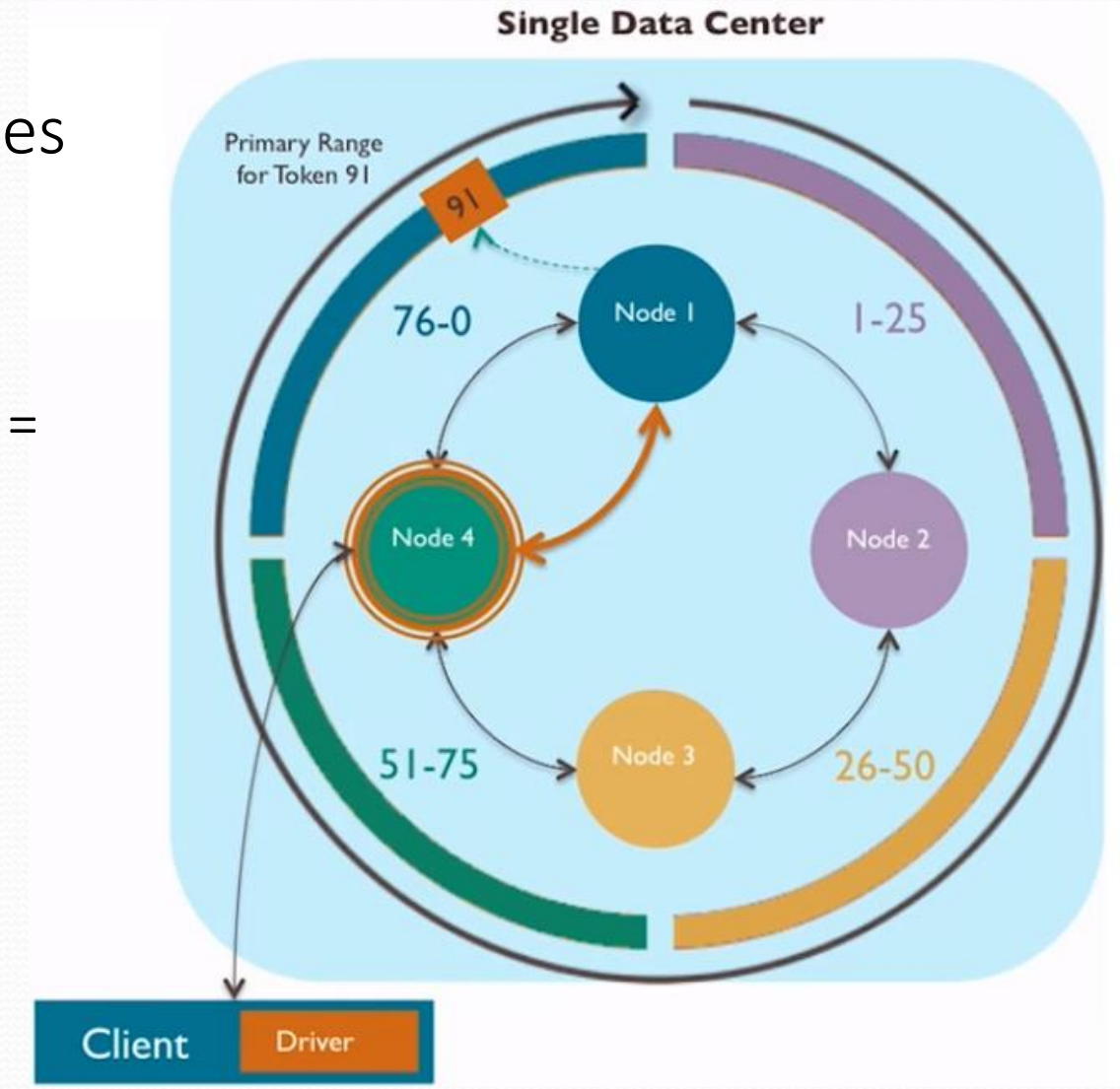


How is data replicated among nodes?

- **SimpleStrategy** - create replicas on nodes subsequent to the primary range node

  CREATE KEYSPACE demo WITH REPLICATION =
  {'class':'**SimpleStrategy**',
  'replication_factor':**3**}

- replication factor of 3 is a recommended minimum

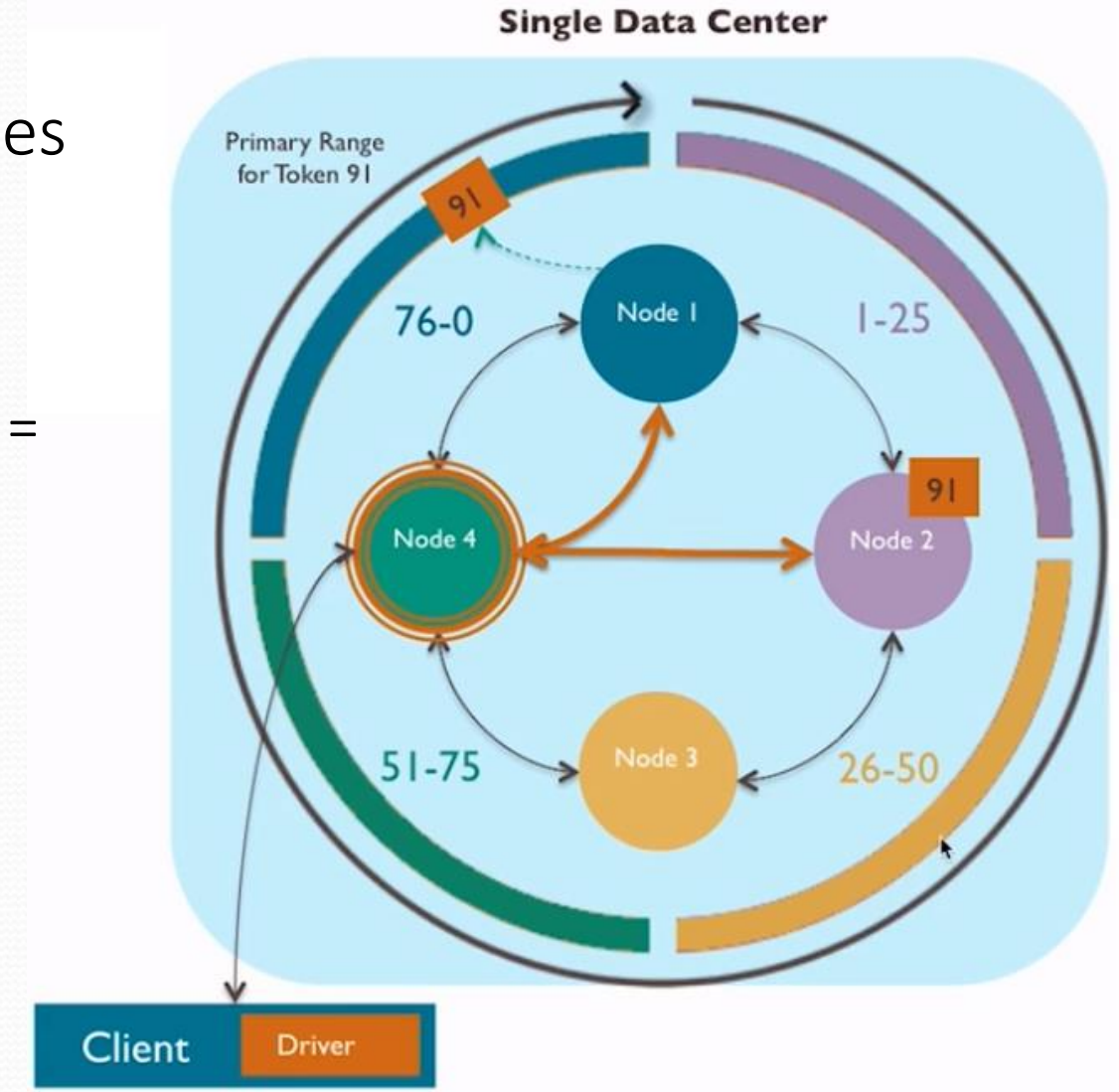


How is data replicated among nodes?

- **SimpleStrategy** - create replicas on nodes subsequent to the primary range node

  CREATE KEYSPACE demo WITH REPLICATION =
  {'class':'**SimpleStrategy**',
  'replication_factor':**3**}

- replication factor of 3 is a recommended minimum

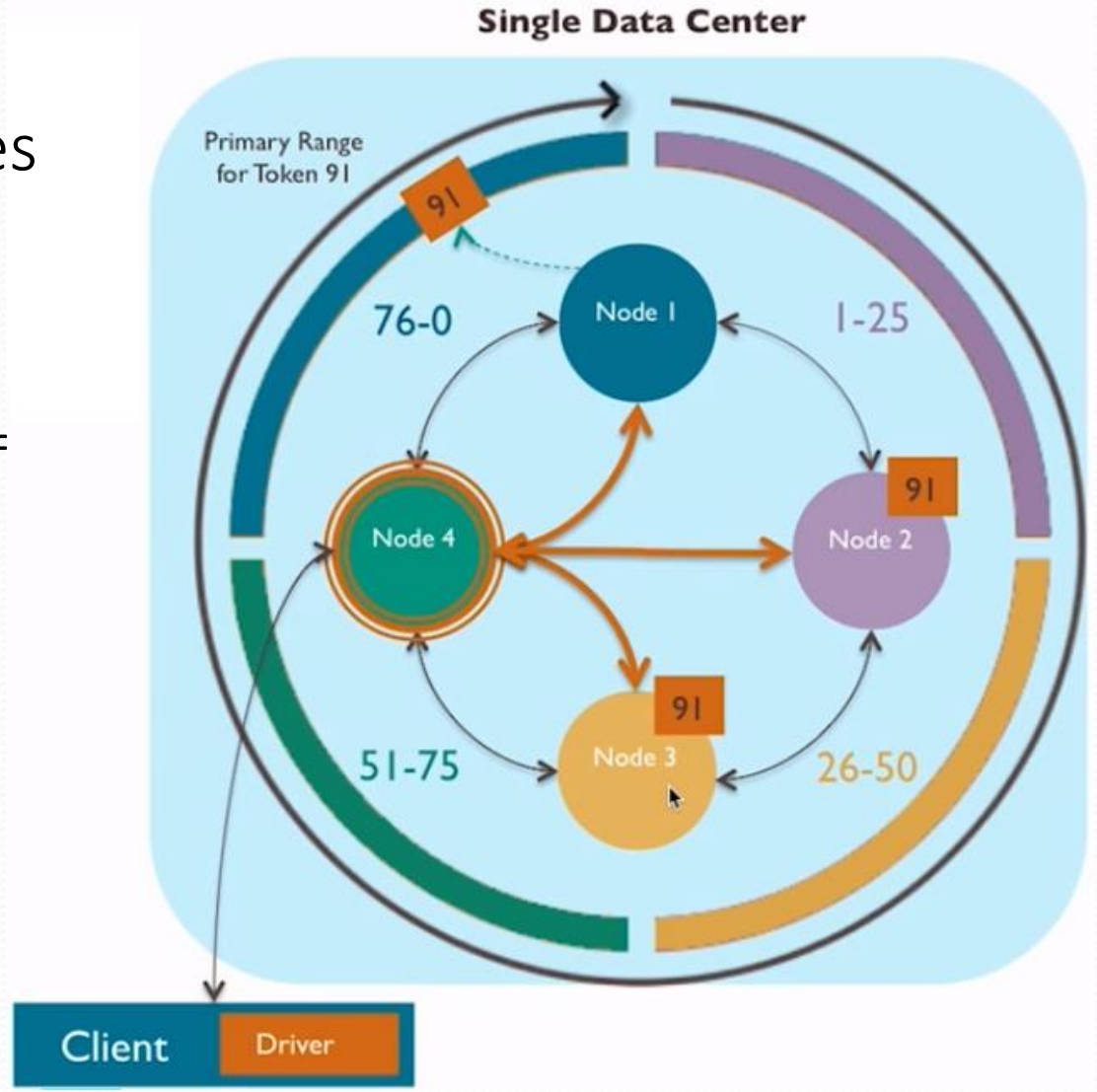


How is data replicated among nodes?

- **SimpleStrategy** - create replicas on nodes subsequent to the primary range node

```
CREATE KEYSPACE demo WITH REPLICATION =
{'class':'SimpleStrategy',
'replication_factor':3}
```

- replication factor of 3 is a recommended minimum



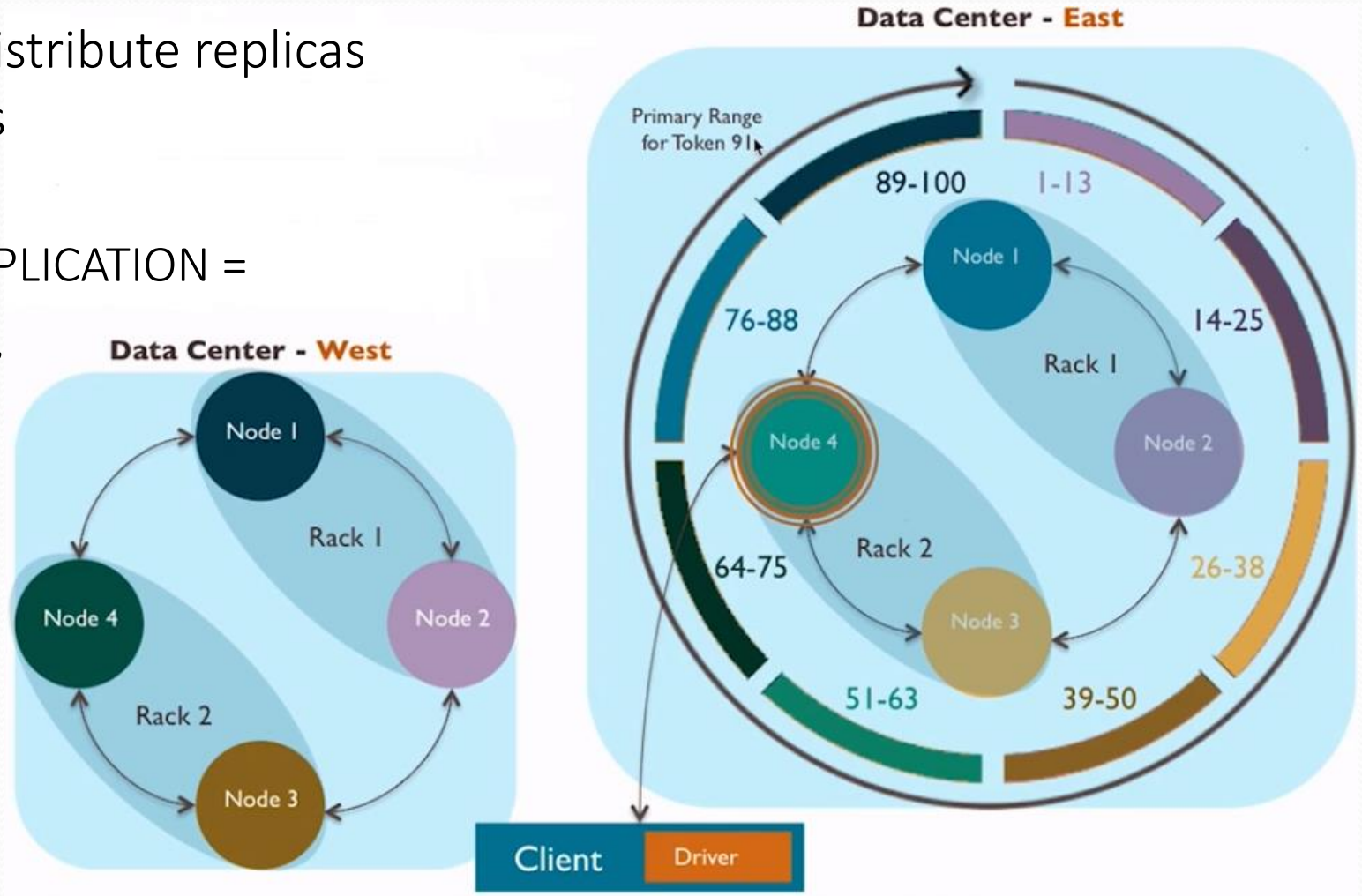
How is data replicated between data centers?

- **NetworkTopologyStrategy** - distribute replicas across racks and data centers

CREATE KEYSPACE demo WITH REPLICATION =

{‘class':'**NetworkTopologyStrategy**‘,

'dc-east':**2**, 'dc-west':**3**}



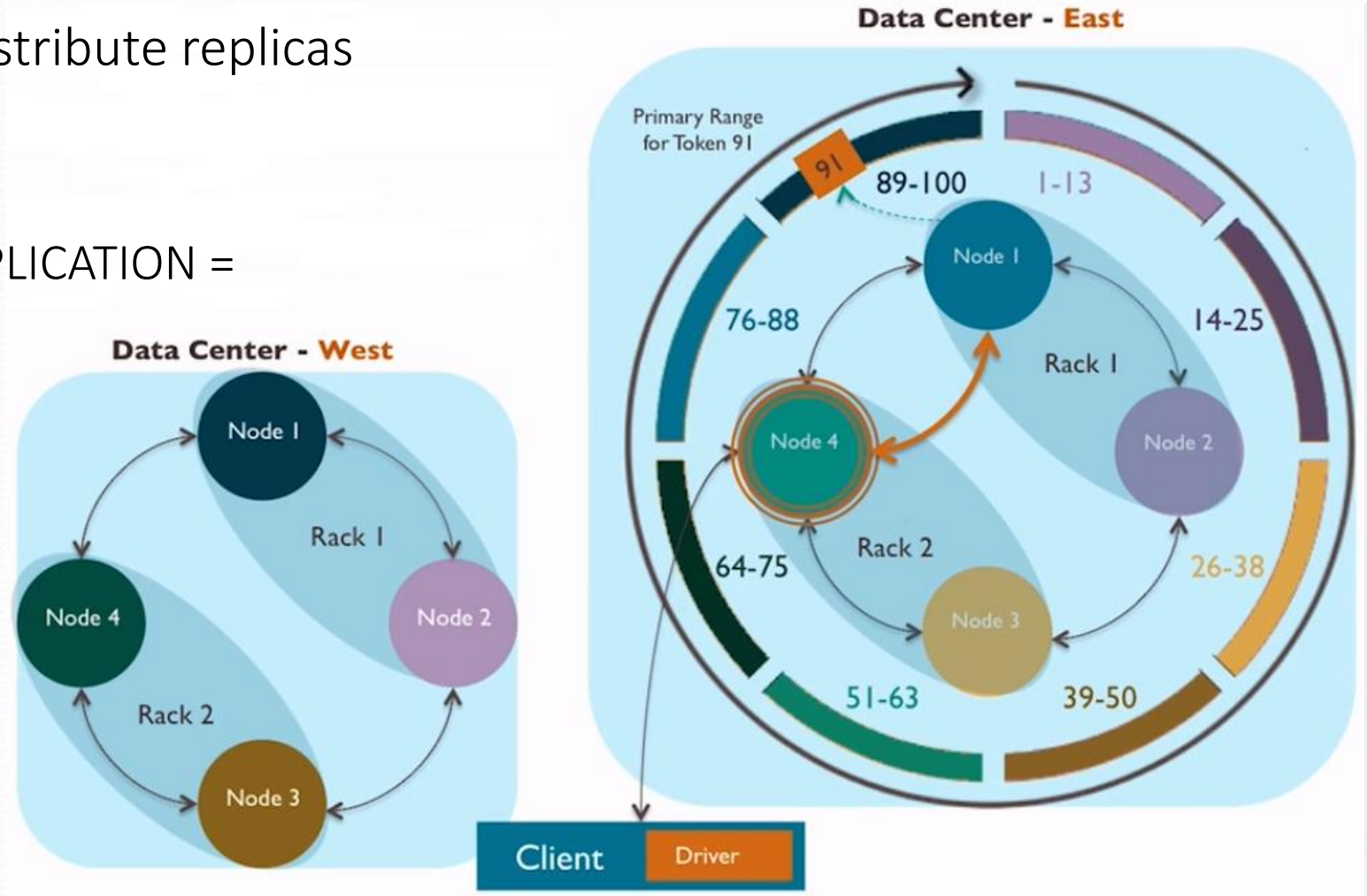
How is data replicated between data centers?

- **NetworkTopologyStrategy** - distribute replicas across racks and data centers

CREATE KEYSPACE demo WITH REPLICATION =

{‘class':'**NetworkTopologyStrategy**‘,

'dc-east':**2**, 'dc-west':**3**}



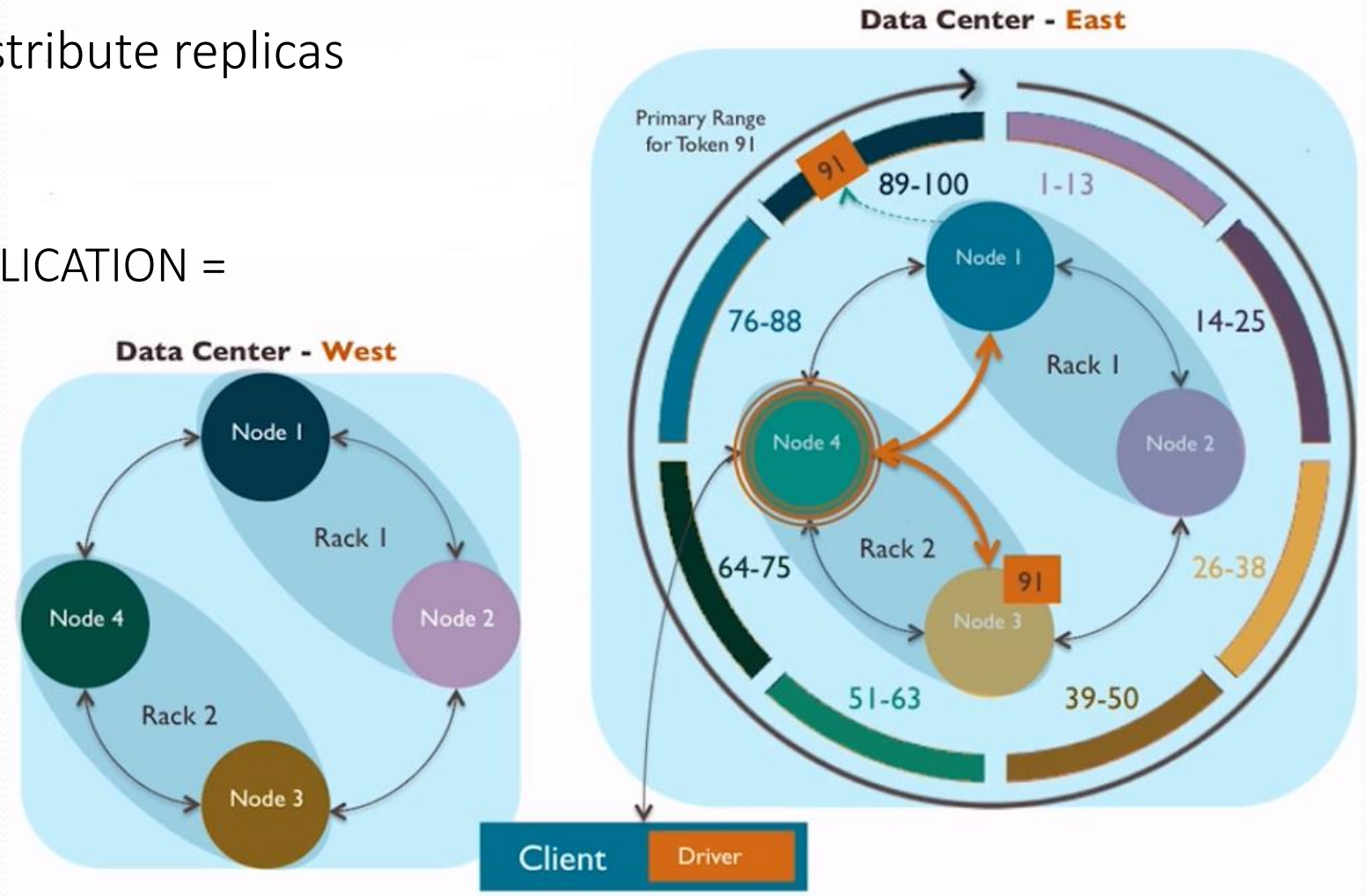
How is data replicated between data centers?

- **NetworkTopologyStrategy** - distribute replicas across racks and data centers

CREATE KEYSPACE demo WITH REPLICATION =

{'class':'**NetworkTopologyStrategy**',

'dc-east':**2**, 'dc-west':**3**}



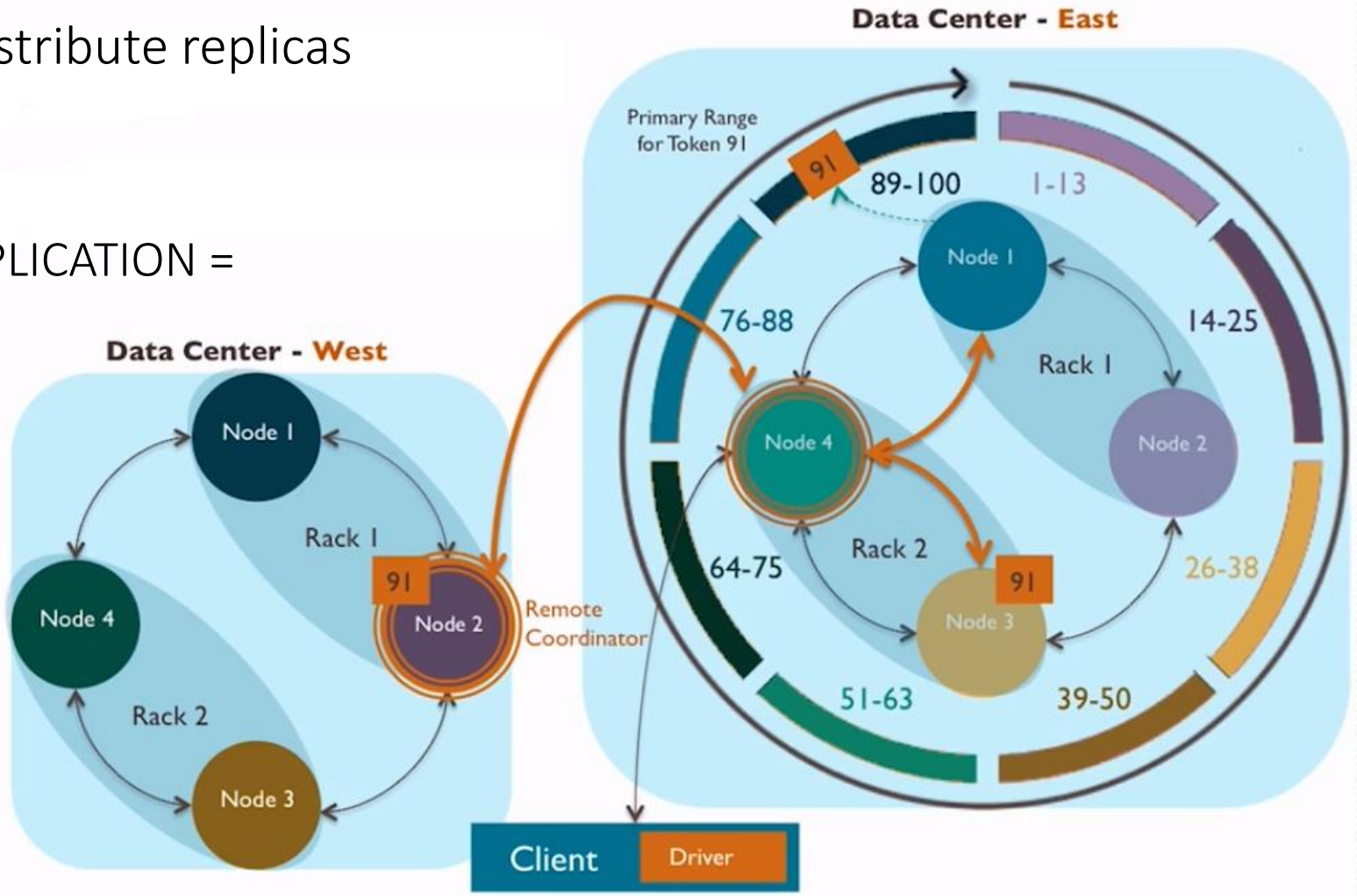
How is data replicated between data centers?

- **NetworkTopologyStrategy** - distribute replicas across racks and data centers

CREATE KEYSPACE demo WITH REPLICATION =

{'class':'**NetworkTopologyStrategy**',

'dc-east':**2**, 'dc-west':**3**}



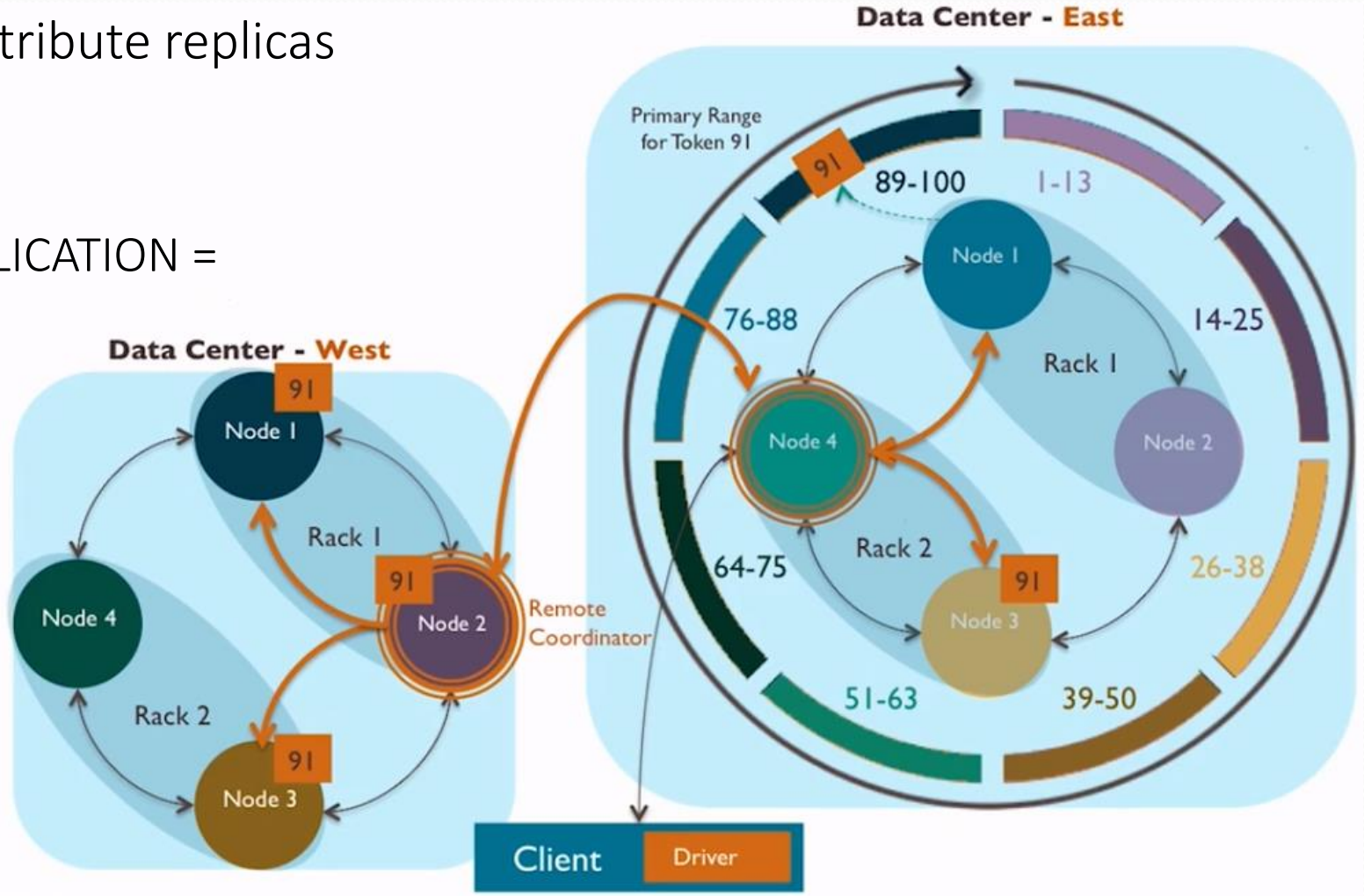
How is data replicated between data centers?

- **NetworkTopologyStrategy** - distribute replicas across racks and data centers

CREATE KEYSPACE demo WITH REPLICATION =

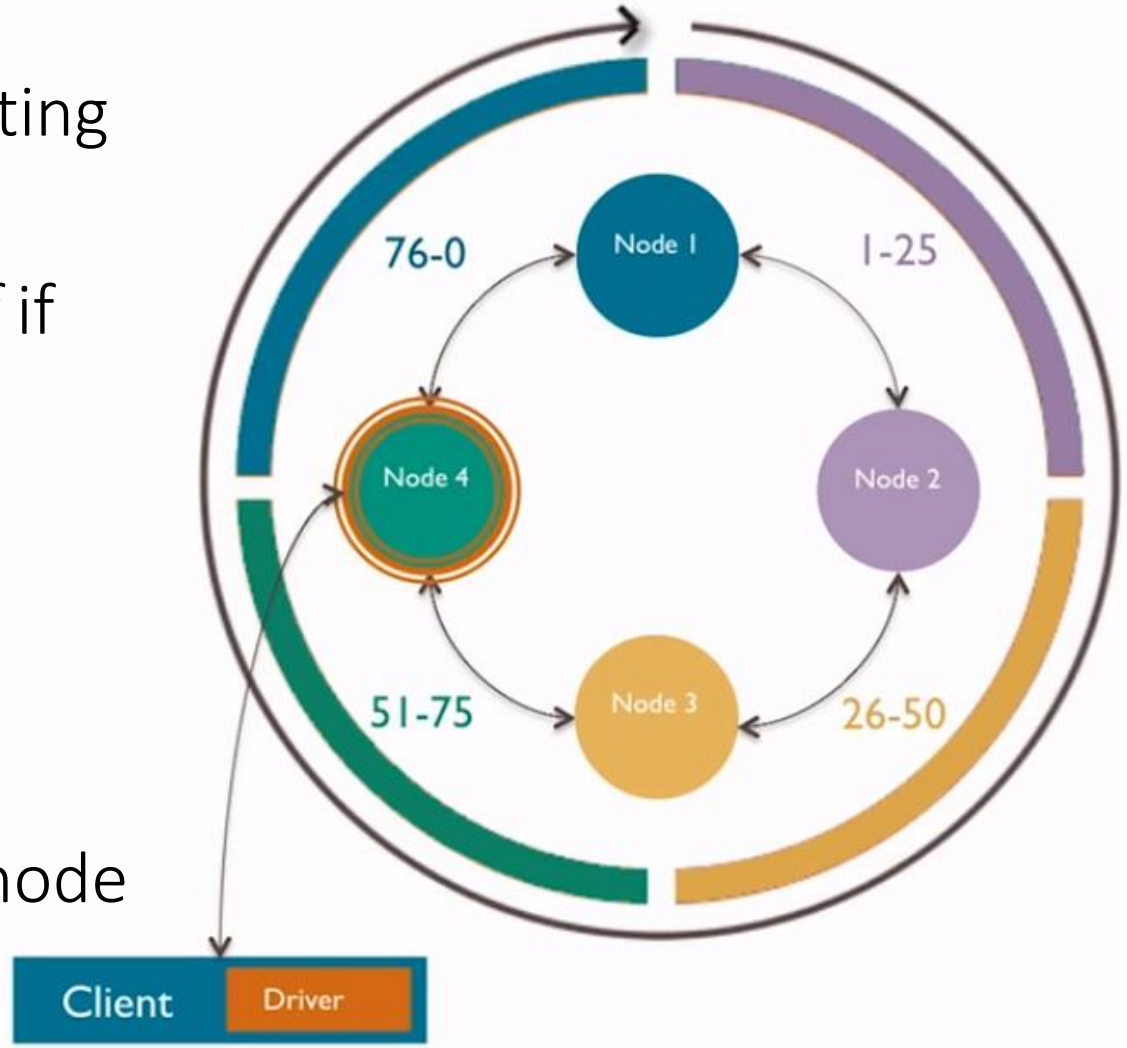
{‘class':'**NetworkTopologyStrategy**‘,

'dc-east':**2**, 'dc-west':**3**}



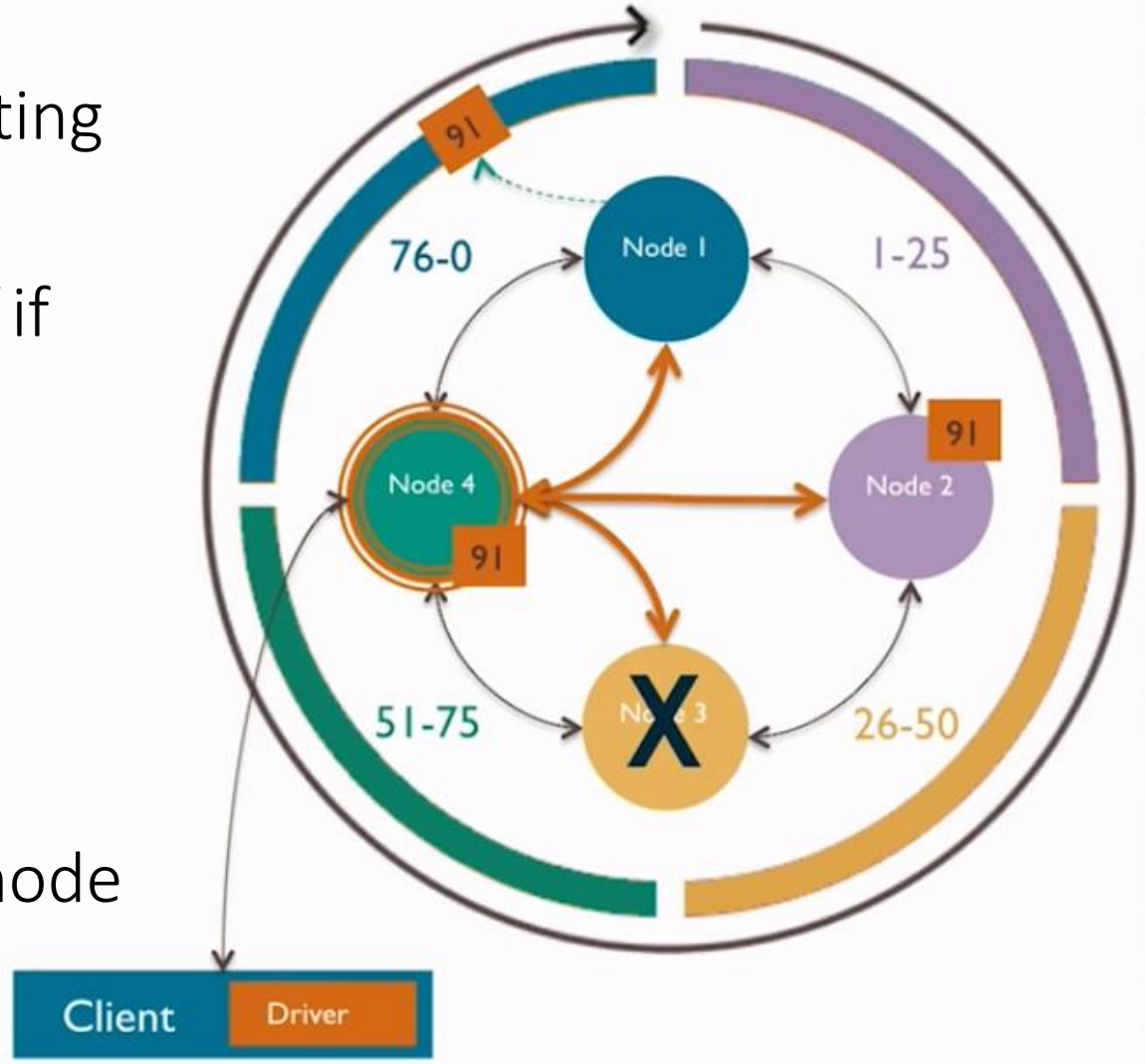
What is a hinted handoff?

- A recovery mechanism for writes targeting offline nodes
- Coordinator can store a *hinted handoff* if target node for a write
  - is known to be down, or
  - fails to acknowledge
- Coordinator stores the hint in its *system.hints* table
- The write is replayed when the target node comes online



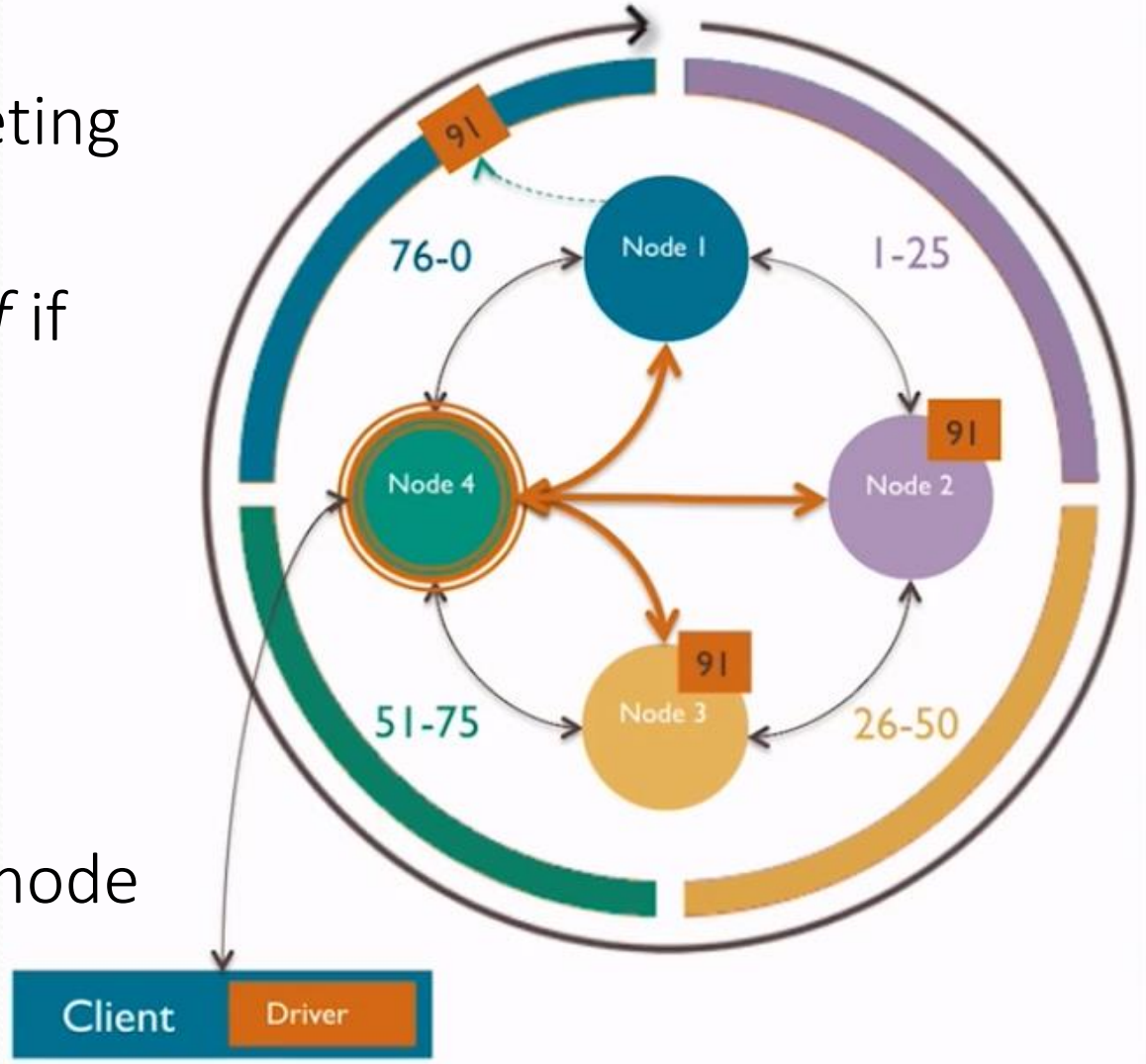
What is a hinted handoff?

- A recovery mechanism for writes targeting offline nodes
- Coordinator can store a *hinted handoff* if target node for a write
  - is known to be down, or
  - fails to acknowledge
- Coordinator stores the hint in its *system.hints* table
- The write is replayed when the target node comes online



What is a hinted handoff?

- A recovery mechanism for writes targeting offline nodes
- Coordinator can store a *hinted handoff* if target node for a write
  - is known to be down, or
  - fails to acknowledge
- Coordinator stores the hint in its *system.hints* table
- The write is replayed when the target node comes online



What is a hinted handoff?

- The **hinted handoff** is comprised of
  - the target node location which is down
  - the partition which requires a replay
  - the data to be written
- Configurations in the cassandra.yaml file include
  - **hinted_handoff_enabled** (default: true) — HH enabled per DC or disabled
  - **max_hint_window_in_ms** (default**: 3 hours**) — after this consecutive outage period hints are no longer generated until target node comes back online
    - nodes offline longer are made consistent using repair or other operations

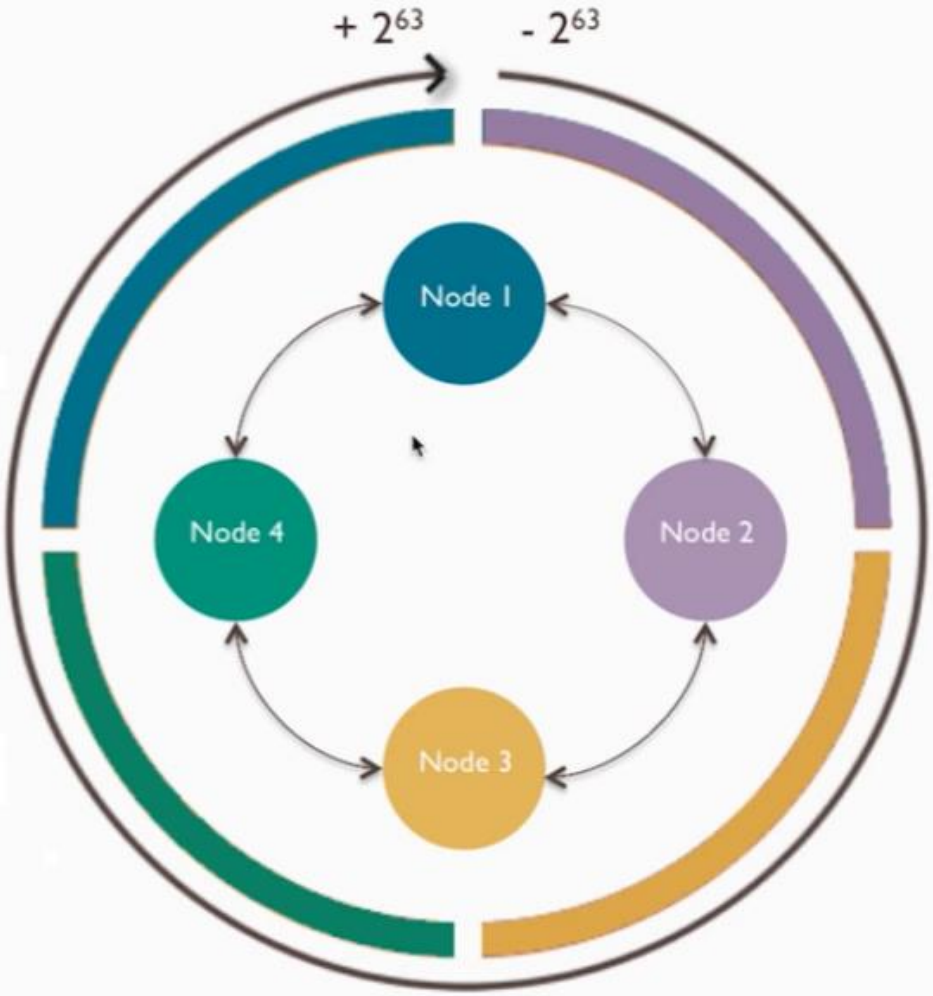
cassandra.yaml

```
# See http://wiki.apache.org/cassandra/HintedHandoff
# May either be "true" or "false" to enable globally, or contain a list
# of data centers to enable per-datacenter.
# hinted_handoff_enabled: DC1,DC2
hinted_handoff_enabled: true
# this defines the maximum amount of time a dead host will have hints
# generated.  After it has been dead this long, new hints for it will not be
# created until it has been seen alive and gone down again.
max_hint_window_in_ms: 10800000 # 3 hours
```

Consistent Hashing

- Data is stored on nodes in partitions, each identified by a unique token
  - **Partition** - a storage location on a node (analogous to a "table row")
  - **Token** - integer value generated by a hashing algorithm. identifying a partition's location within a cluster
- The 2^{64} value token range for a cluster is used as a single ring
  - So, any partition in a cluster is locatable from one consistent set of hash values, regardless of its node
- Specific token range varies by choice of Partitioner

http://www.wolframalpha.com/input/?i=2^128

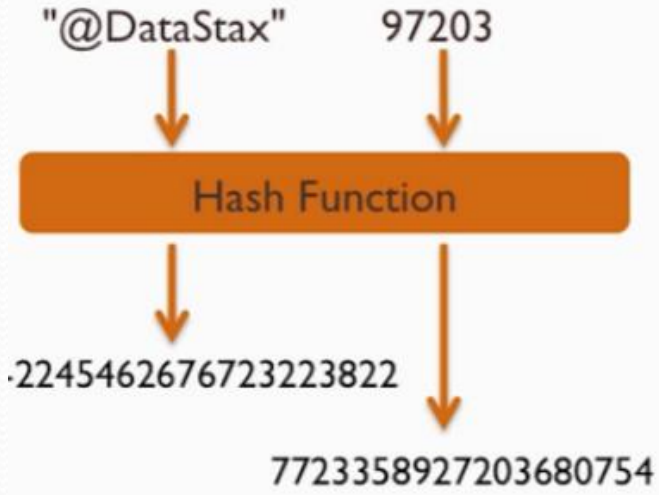


Partitioning

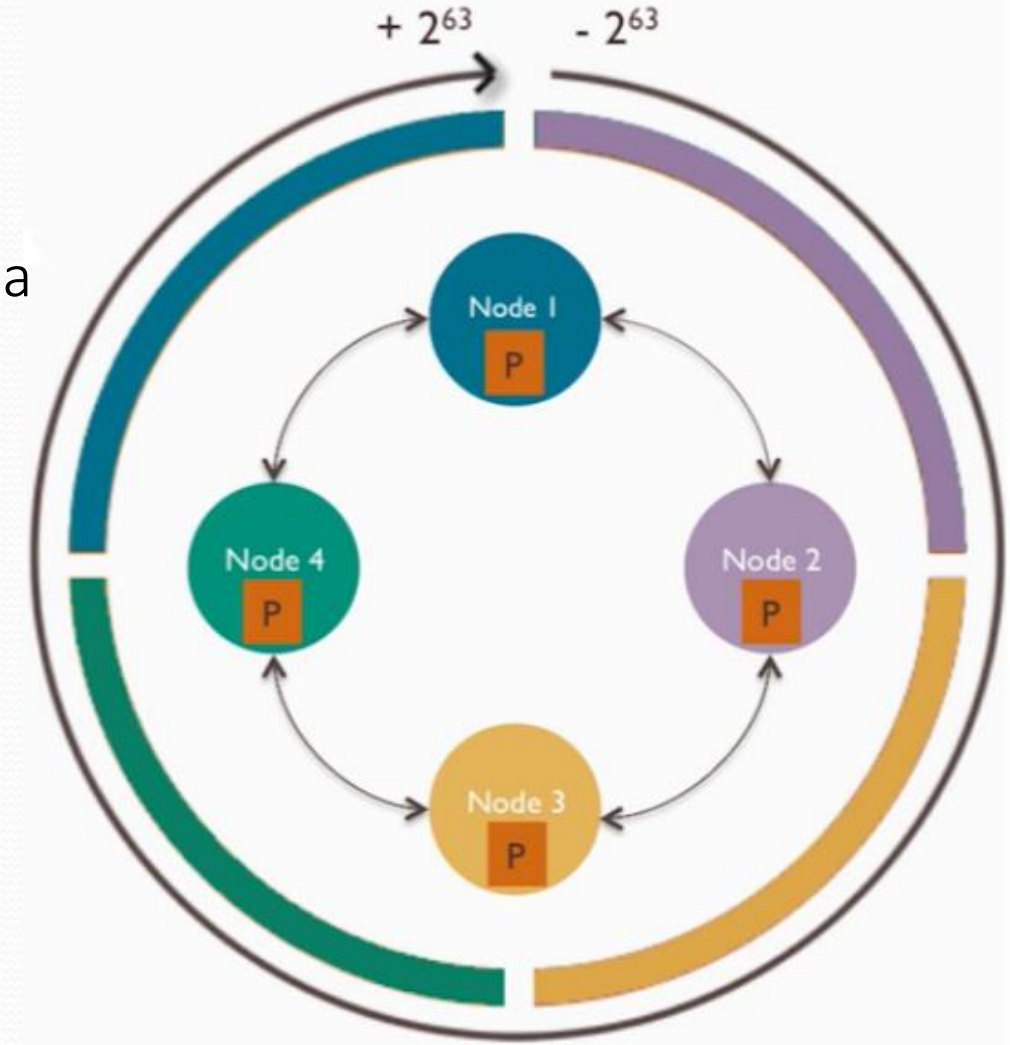
- Nodes are *logically* structured in Ring Topology.
- Hashed value of key associated with data partition is used to assign it to a node in the ring.
- Hashing rounds off after certain value to support ring structure.
- Lightly loaded nodes moves position to alleviate highly loaded nodes.

What is the Partitioner?

- A system on each node which hashes tokens from designated values in rows being added
  - **Hash function** - converts a variable length value to a corresponding fixed length value

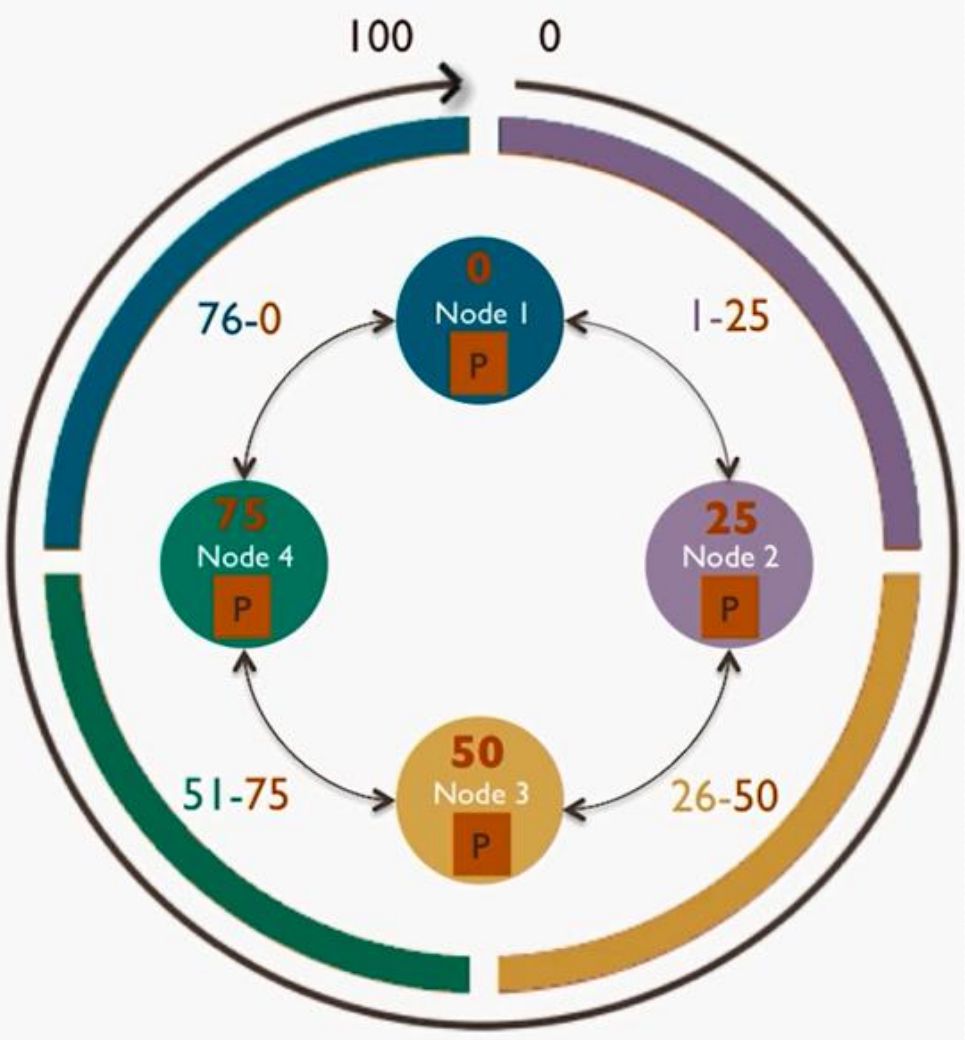


  - Various partitioners available



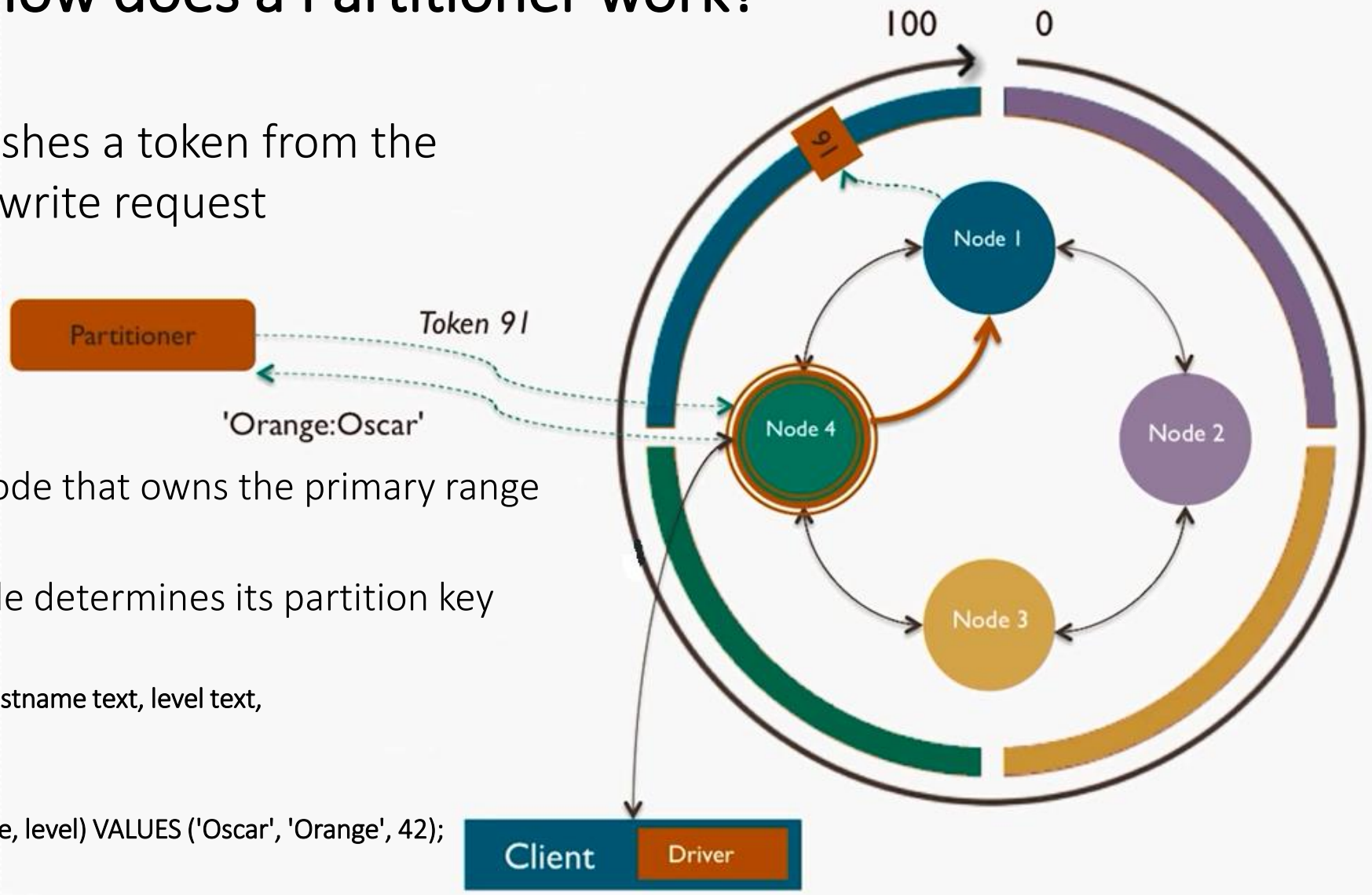
What is the Partitioner?

- Imagine a 0 to 100 token range (instead of -2^63 to +2^63)
  - Each node is assigned a token, just like each of its partitions
  - Node tokens are the highest value in the segment owned by that node
- This segment is the primary token range of replicas owned by this node
  - Nodes also store replicas keyed to tokens outside this range ("secondary range")



How does a Partitioner work?

- A node's partitioner hashes a token from the partition key value of a write request



- First replica written to node that owns the primary range for this token
- The primary key of a table determines its partition key value

```
CREATE TABLE Users ( firstname text, lastname text, level text,
PRIMARY KEY ((lastname, firstname)) );

INSERT INTO Users (firstname, lastname, level) VALUES ('Oscar', 'Orange', 42);
```

What Partitioners does Cassandra offer?

- Cassandra offers three partitioners
  - **Murmur3Partitioner** (default) — uniform distribution based on Murmur3 hash
  - **RandomPartitioner** uniform distribution based on MD5 hash
  - **ByteOrderedPartitioner** (legacy only) — lexical distribution based on key bytes
- Murmur3Partitioner is the default and best practice
- The partitioner is configured in the cassandra.yaml file
  - Must be the same across all nodes in the cluster

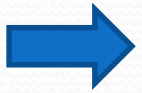
cassandra.yaml

```
# The partitioner is responsible for distributing rows (by key) across
# nodes in the cluster.  Any IPartitioner may be used, including your
# own as long as it is on the classpath.  Out of the box, Cassandra
# provides org.apache.cassandra.dht.{Murmur3Partitioner, RandomPartitioner
# ByteOrderedPartitioner, OrderPreservingPartitioner (deprecated)}.
#
# See http://wiki.apache.org/cassandra/Operations for more on
# partitioners and token selection.
partitioner: org.apache.cassandra.dht.Murmur3Partitioner
```

Consistent Hashing

- Consistent hashing allows distribution of data across a cluster to minimize reorganization when nodes are added or removed. Consistent hashing partitions data based on the partition key. (For an explanation of partition keys and primary keys
- For example, if you have the following data: Cassandra assigns a hash value to each partition key:

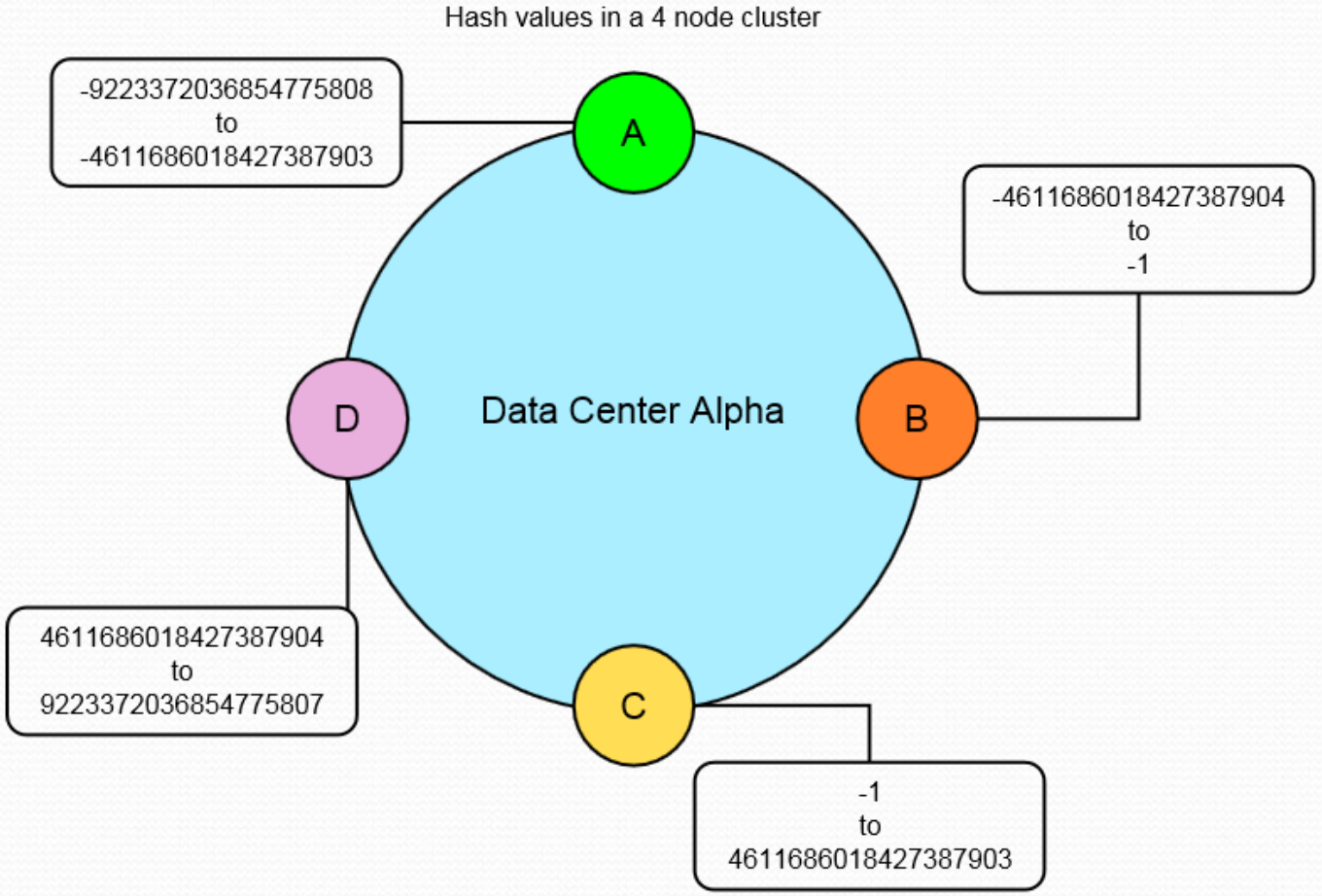
name	age	car	gender
jim	36	camaro	M
carol	37	bmw	F
johnny	12		M
suzy	10		F



Partition key	Murmur3 hash value
jim	-2245462676723223822
carol	7723358927203680754
johnny	-6723372854036780875
suzy	1168604627387940318

Consistent Hashing

- Each node in the cluster is responsible for a range of data based on the hash value:



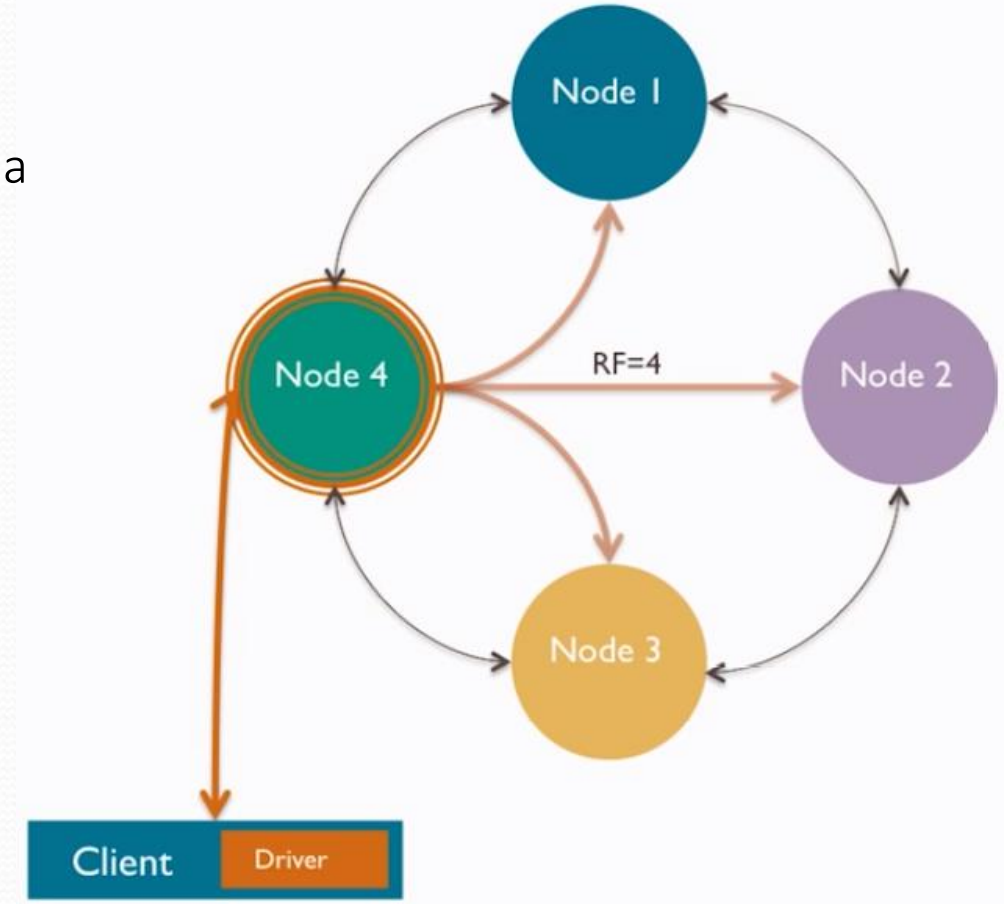
Consistent Hashing

- Cassandra places the data on each node according to the value of the partition key and the range that the node is responsible for. For example, in a four node cluster, the data in this example is distributed as follows:

Node	Start range	End range	Partition key	Hash value
A	-9223372036854775808	-4611686018427387903	johnny	-6723372854036780875
B	-4611686018427387904	-1	jim	-2245462676723223822
C	0	4611686018427387903	suzy	1168604627387940318
D	4611686018427387904	9223372036854775807	carol	7723358927203680754

What is consistency?

- The partition key determines which nodes are sent any given request
  - **Consistency Level** - sets how many of the nodes to be sent a given request must acknowledge that request, for a response to be returned to the client
- The meaning varies by type
  - **Write request** - how many nodes must acknowledge they received and wrote the write request?
  - **Read request** - how many nodes must acknowledge by sending their most recent copy of the data?



Per-Query Consistency Levels

- Latency increases the more nodes you have to involve.

ANY: For writes only. Writes to any available node and expects Cassandra to sort it out. Fire and forget.

ONE: Reads or writes to the closest replica.

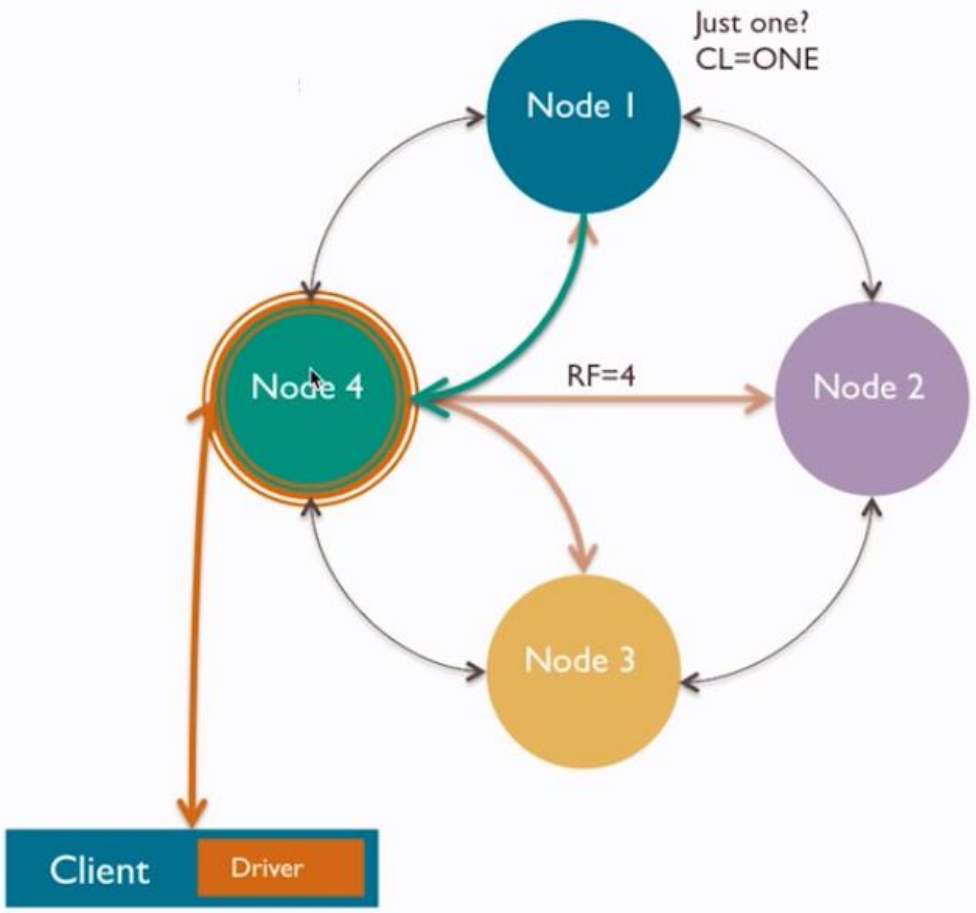
QUORUM: Writes to half+1 of the appropriate replicas before the operation is successful. A read is successful when half+1 replicas agree on a value to return.

LOCAL_QUORUM: Same as above, but only to the local datacenter in a multi-datacenter topology.

ALL: For writes, all replicas need to ack the write. For reads, returns the record with the newest timestamp once all replicas reply. In both cases, if were missing even one replica, the operation fails

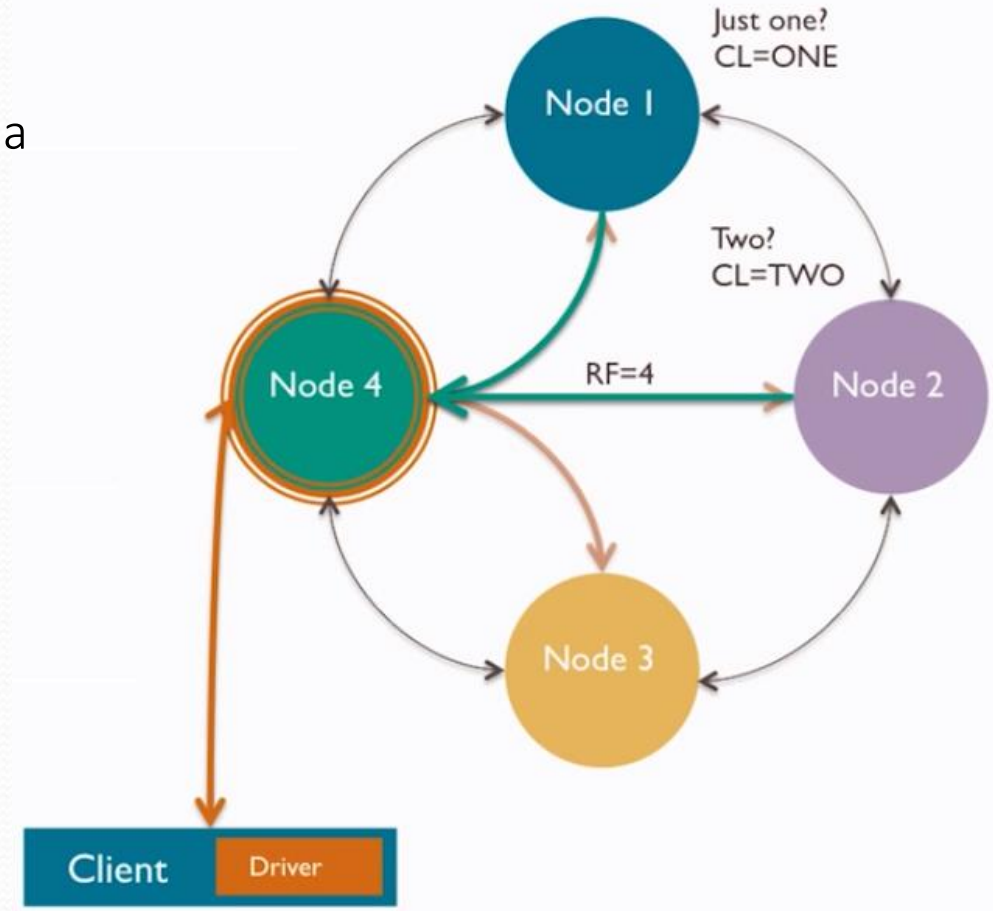
What is consistency?

- The partition key determines which nodes are sent any given request
  - **Consistency Level** - sets how many of the nodes to be sent a given request must acknowledge that request, for a response to be returned to the client
- The meaning varies by type
  - **Write request** - how many nodes must acknowledge they received and wrote the write request?
  - **Read request** - how many nodes must acknowledge by sending their most recent copy of the data?



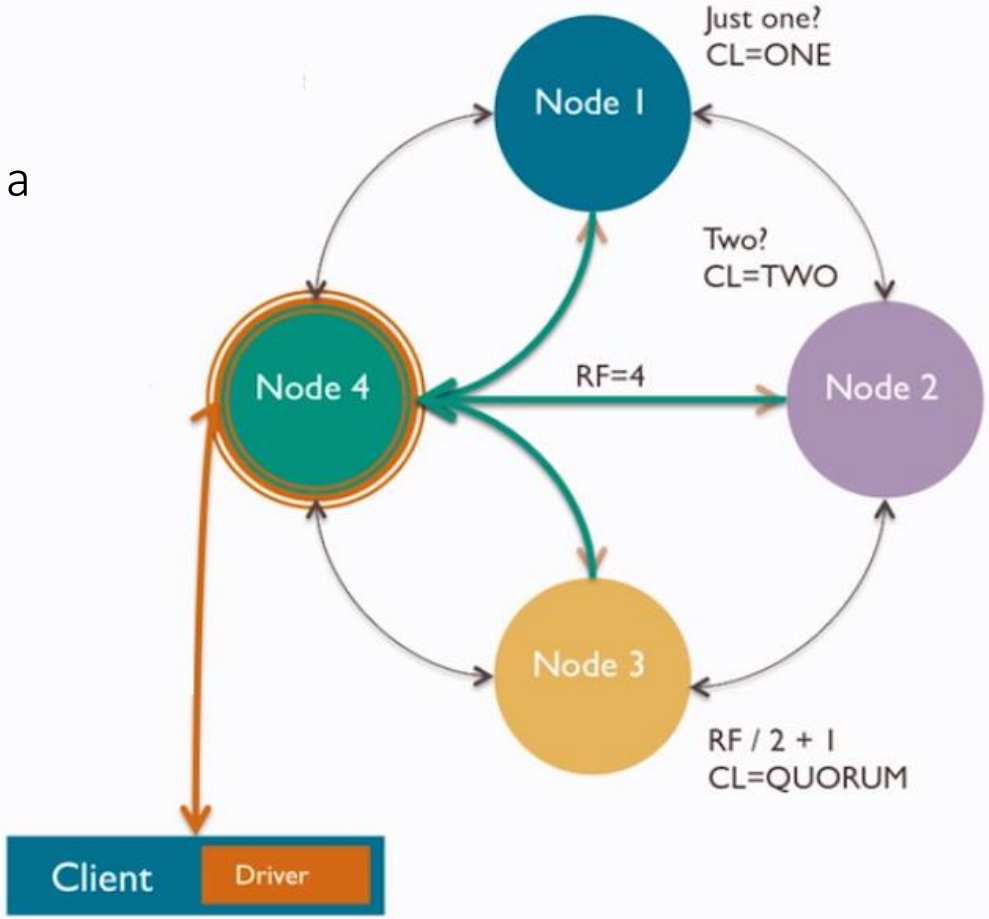
What is consistency?

- The partition key determines which nodes are sent any given request
  - **Consistency Level** - sets how many of the nodes to be sent a given request must acknowledge that request, for a response to be returned to the client
- The meaning varies by type
  - **Write request** - how many nodes must acknowledge they received and wrote the write request?
  - **Read request** - how many nodes must acknowledge by sending their most recent copy of the data?



What is consistency?

- The partition key determines which nodes are sent any given request
  - **Consistency Level** - sets how many of the nodes to be sent a given request must acknowledge that request, for a response to be returned to the client
- The meaning varies by type
  - **Write request** - how many nodes must acknowledge they received and wrote the write request?
  - **Read request** - how many nodes must acknowledge by sending their most recent copy of the data?

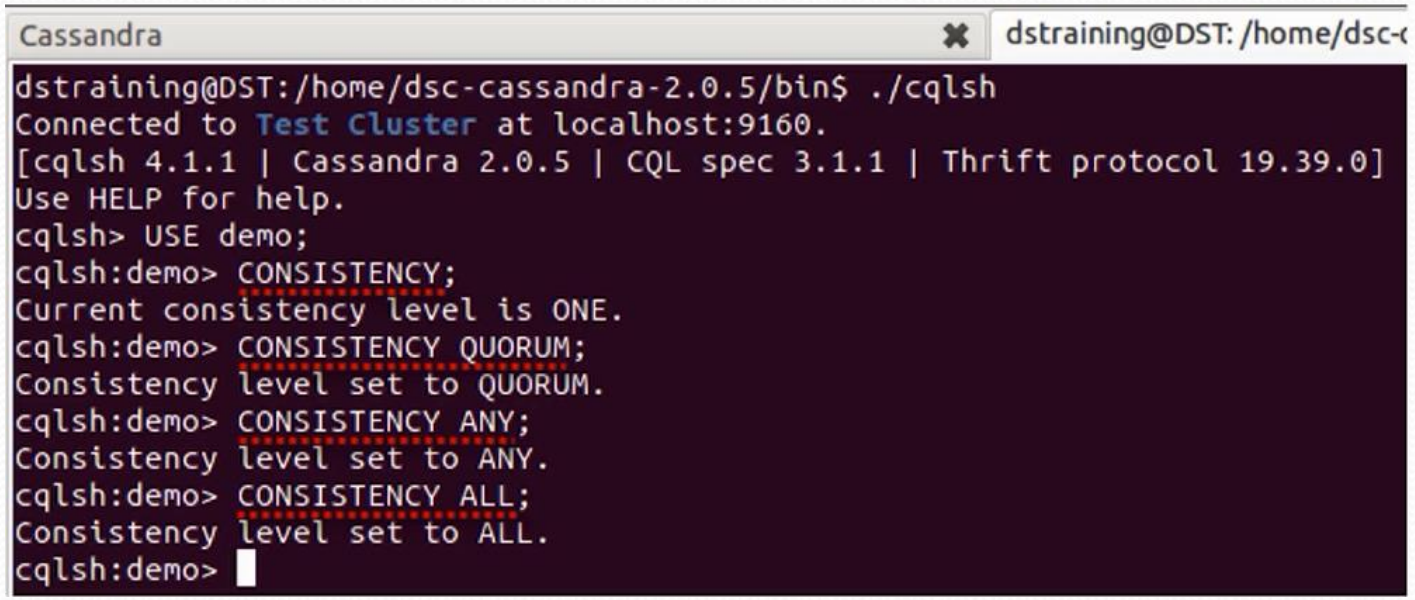


What consistency levels are available?

Name	Description	Usage
ANY (writes only)	Write to any node, and store *hinted handoff* if all nodes are down.	Highest availability and lowest consistency (writes)
ALL	Check all nodes. Fail if any is down.	Highest consistency and lowest availability
ONE (TWO, THREE)	Check closest node to coordinator.	Highest availability and lowest consistency (reads)
QUORUM	Check quorum of available nodes.	Balanced consistency and availability
LOCAL_ONE	Check closest node to coordinator, in the local data center only.	Highest availability, lowest consistency, and no cross-data-center traffic
LOCAL_QUORUM	Check quorum of available nodes, in the local data center only.	Balanced consistency and availability, with no cross-data-center traffic
EACH_QUORUM	Only valid for writes. Check quorum of available nodes, in each data center of the cluster.	Balanced consistency and availability, with cross-data-center consistency
SERIAL	Conditional write to quorum of nodes. Read current state with no change.	Used to support linearizable consistency for lightweight transactions
LOCAL_SERIAL	Conditional write to quorum of nodes in local data center.	Used to support linearizable consistency for lightweight transactions

How do you set consistency per request?

- The default consistency level for all requests is ONE
  - In cqlsh, the *CONSISTENCY* command modifies this value for all subsequent requests during the same cqlsh session
  - In client drivers, a *ConsistencyLevel* constant is passed as part of each request



What is immediate vs. eventual consistency?

- For any given read, how likely is it the data may be stale?
- Immediate Consistency — reads always return the most recent data
  - Consistency Level ALL guarantees immediate consistency, because all replica nodes are checked and compared before a result is returned
  - Highest latency because all replicas are checked and compared
- Eventual Consistency — reads may return stale data
  - Consistency Level ONE carries the highest risk of stale data, because only one replica node is checked before a result is returned
  - Lowest latency because the response from one replica is immediately returned

How do you choose a consistency level?

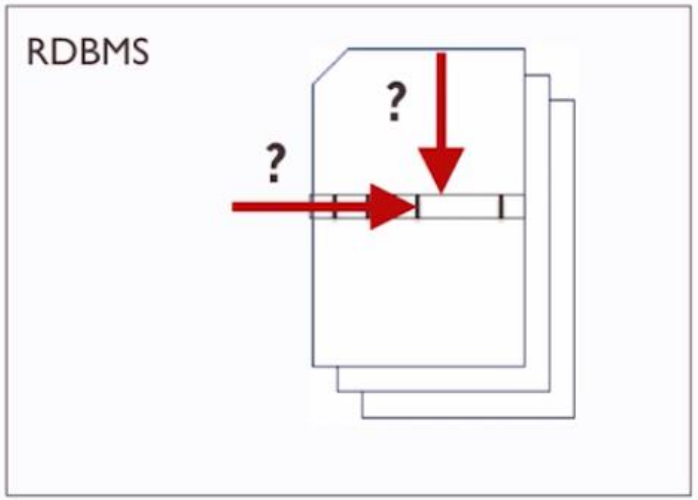
- In any given scenario, is the value of immediate consistency worth the latency cost?
  - Netflix uses CL ONE and measures its "eventual" consistency in milliseconds
  - Consistency Level ONE is your friend ...

Consistency Level ONE	Consistency Level QUORUM	Consistency Level ALL
Lowest latency	Higher latency (than ONE)	Highest latency
Highest throughput	Lower throughput	Lowest throughput
Highest availability	Higher availability (than ALL)	Lowest availability
Stale read possible (if read CL + write CL < RF)	No stale reads (if read and write at quorum)	No stale reads (if either read or write at ALL)

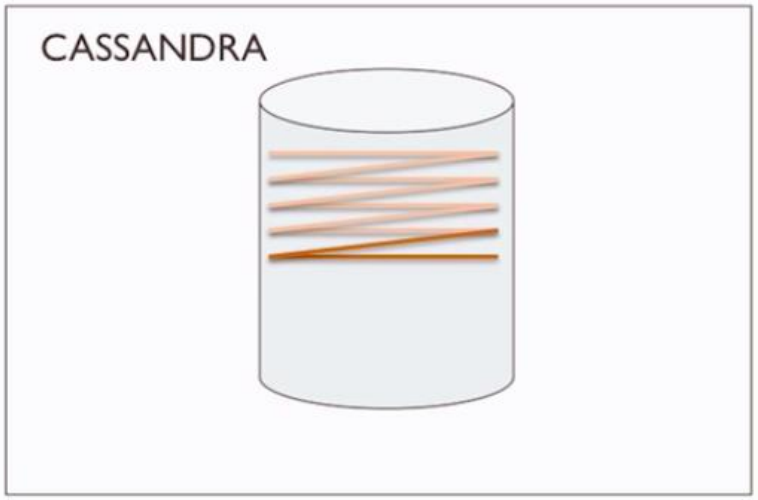
Write Path

How does Cassandra write so fast?

- Cassandra is a log-structured storage engine
  - Data is sequentially appended, not placed in pre-set locations



Seeks and writes values to various pre-set locations



Continuously appends to a log

Key components of the write path

- Each node implements four key components to handle its writes
  - Memtables - in-memory tables corresponding to CQL tables, with indexes
  - CommitLog — append-only log, replayed to restore downed node's Memtables
  - SSTables — Memtable snapshots periodically flushed to disk, clearing heap
  - Compaction — periodic process to merge and streamline SSTables

- When any node receive any write request

I. The record appends to the CommitLog, and

2. The record appends to the Memtable for this record's target CQL table

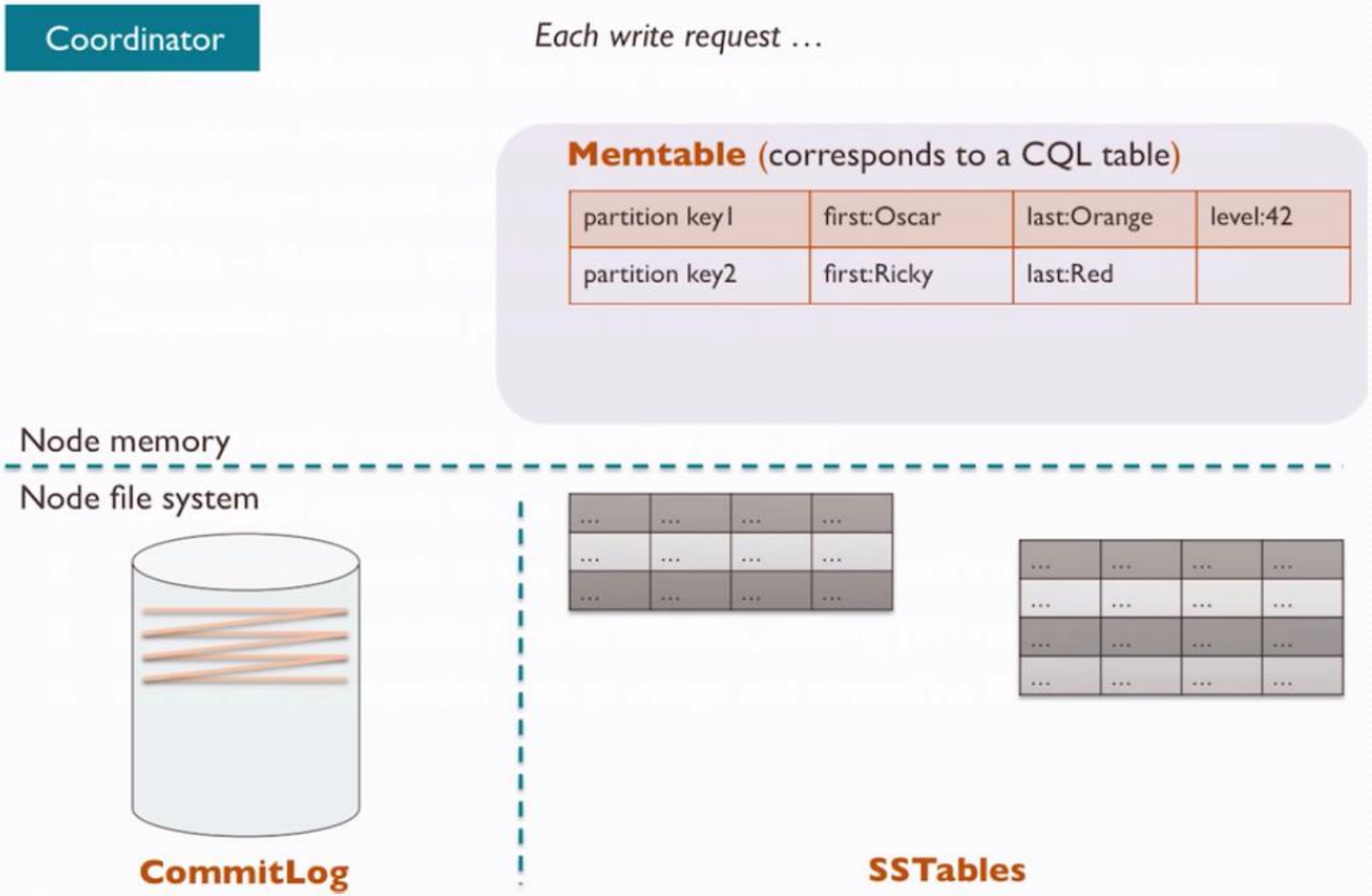
3. Periodically. Memtables flush to SSTables, clearing jVM heap and CommitLog

4. Periodically, Compaction runs to merge and streamline SSTables

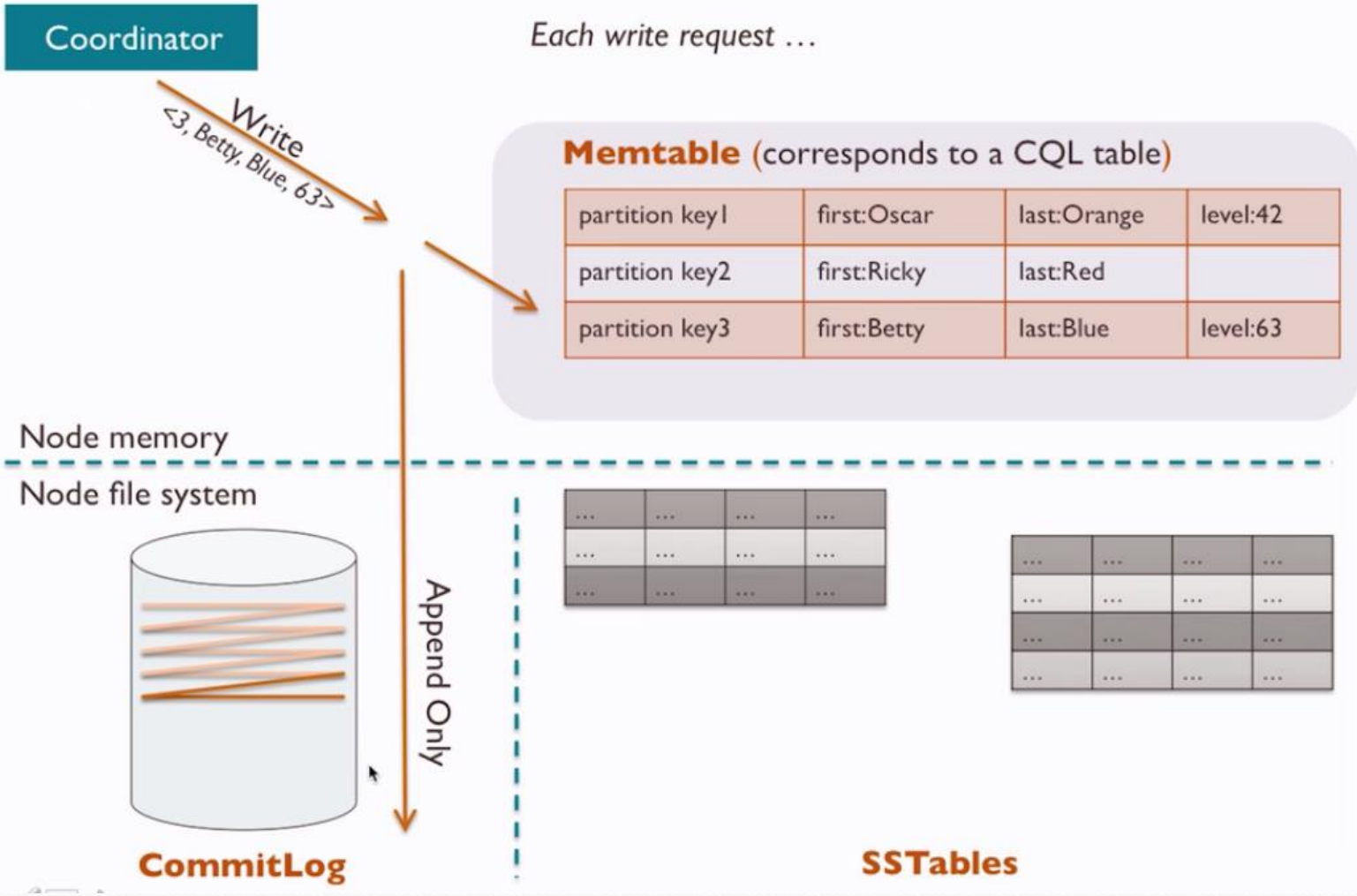
Cassandra Write Path

- Cassandra identifies which node owns the token you're trying to write based on your partitioning, replication and placement strategies.
- Data Written to ***CommitLog***
  - Sequential writes to disk, kind of like a MySQL binlog.
  - Mostly written to, is only read from upon a restart.
- Data Written to ***Memtable***.
  - Acts as a Write-back cache of data.
- ***Memtable*** hits a threshold (configurable) it is flushed to disk as an ***SSTable***. An ***SSTable*** (Sorted String Table) is an immutable file on disk. More on compaction later.

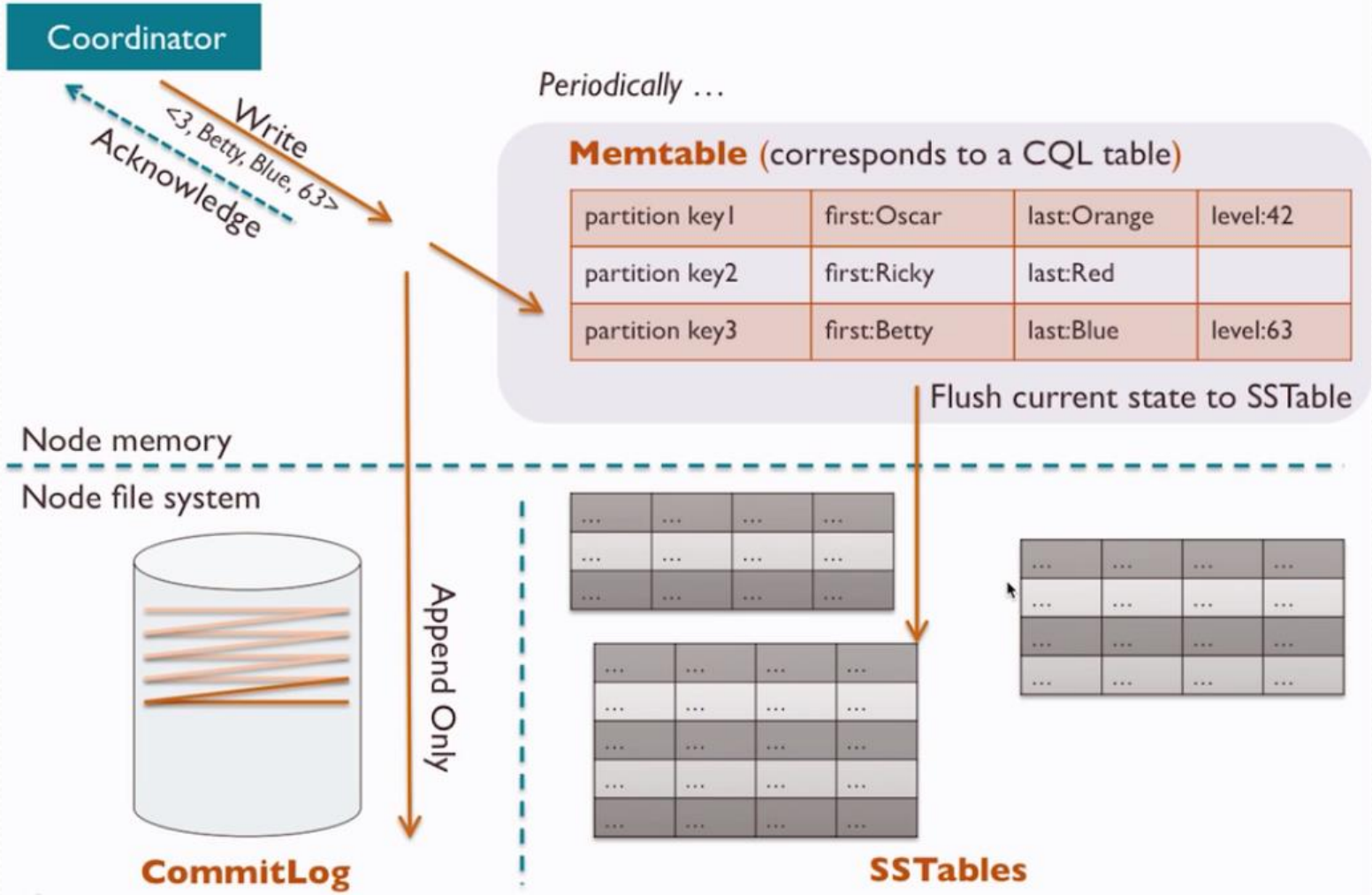
How does the write path flow on a node?



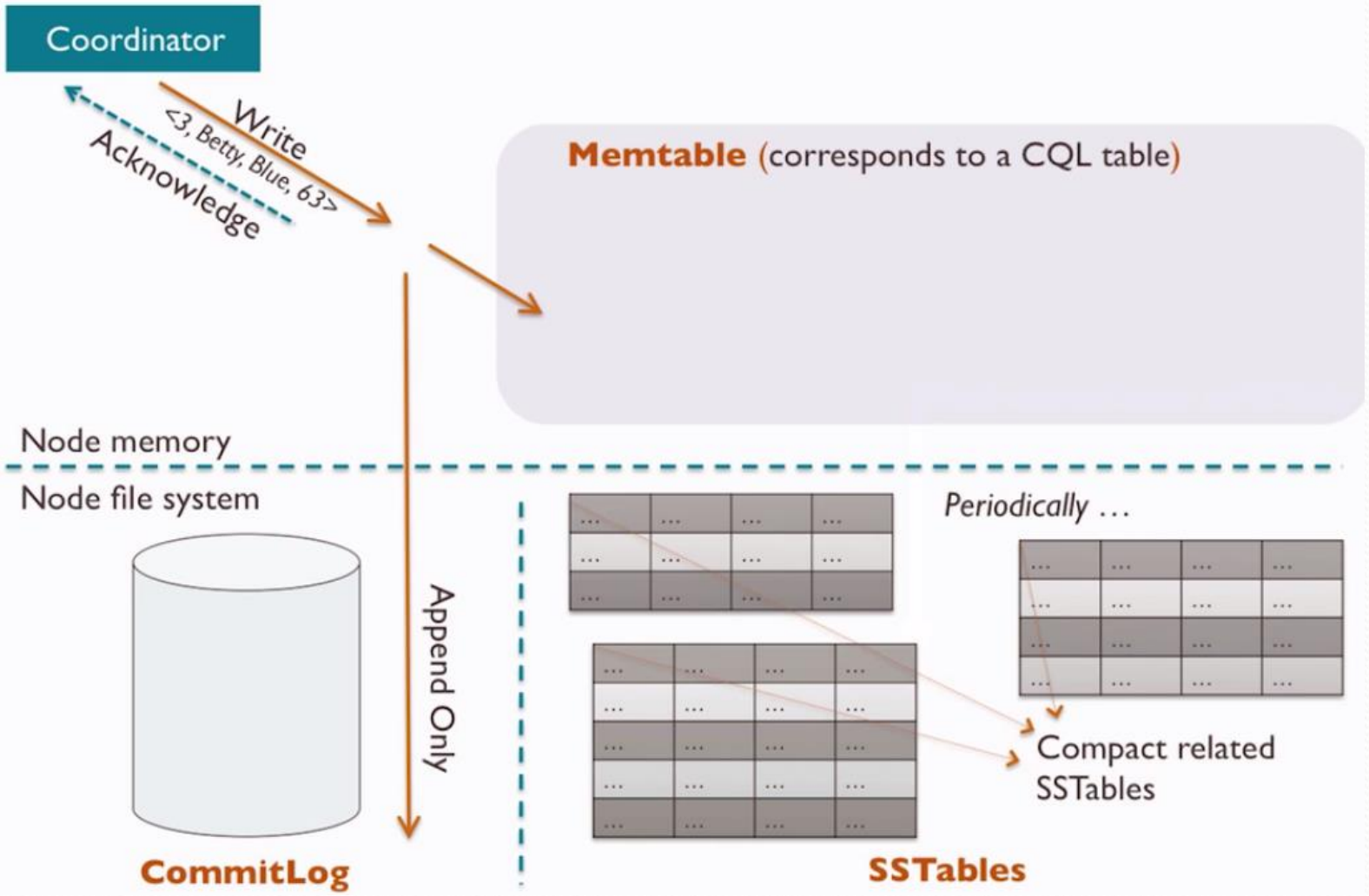
How does the write path flow on a node?



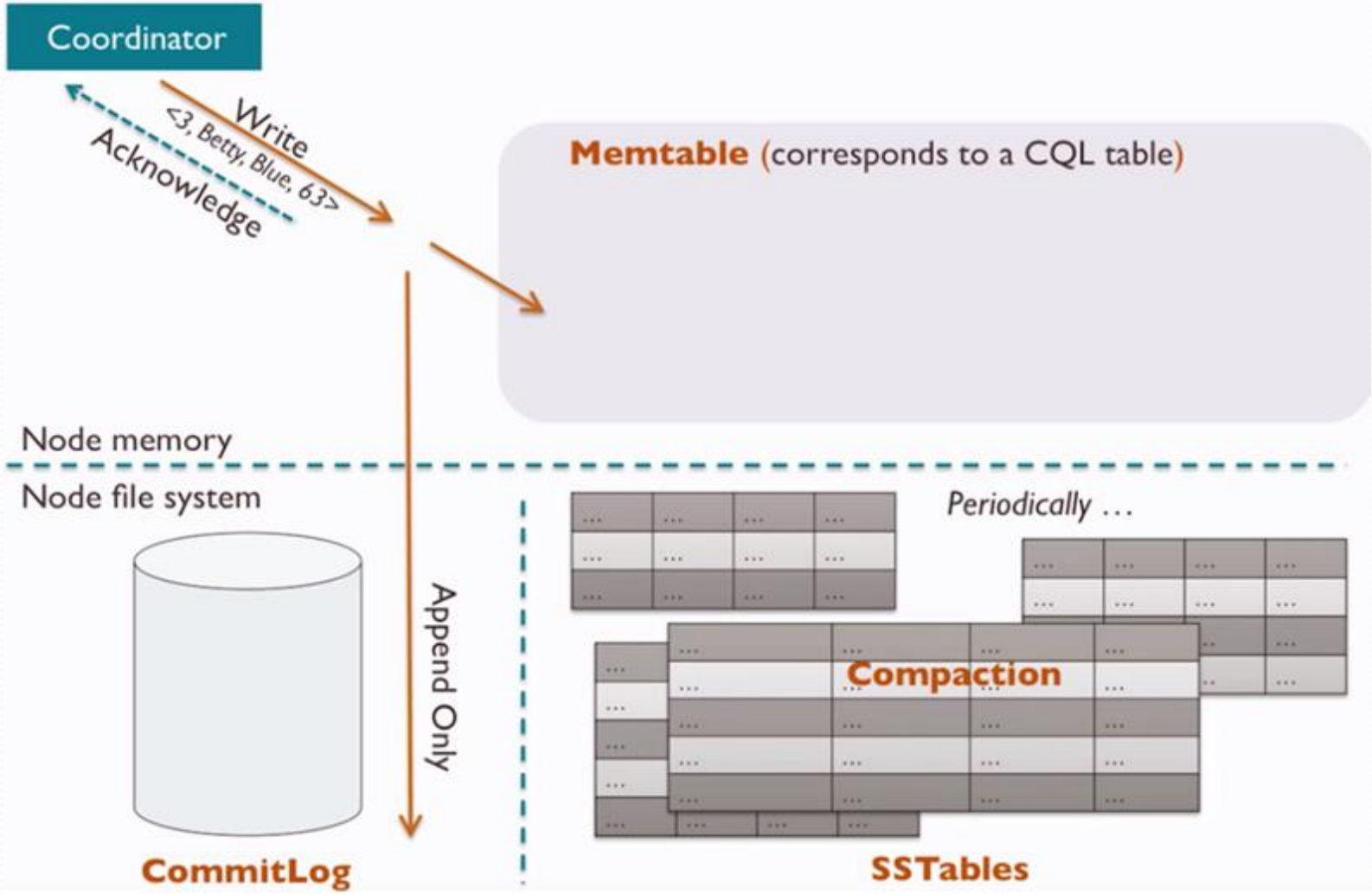
How does the write path flow on a node?



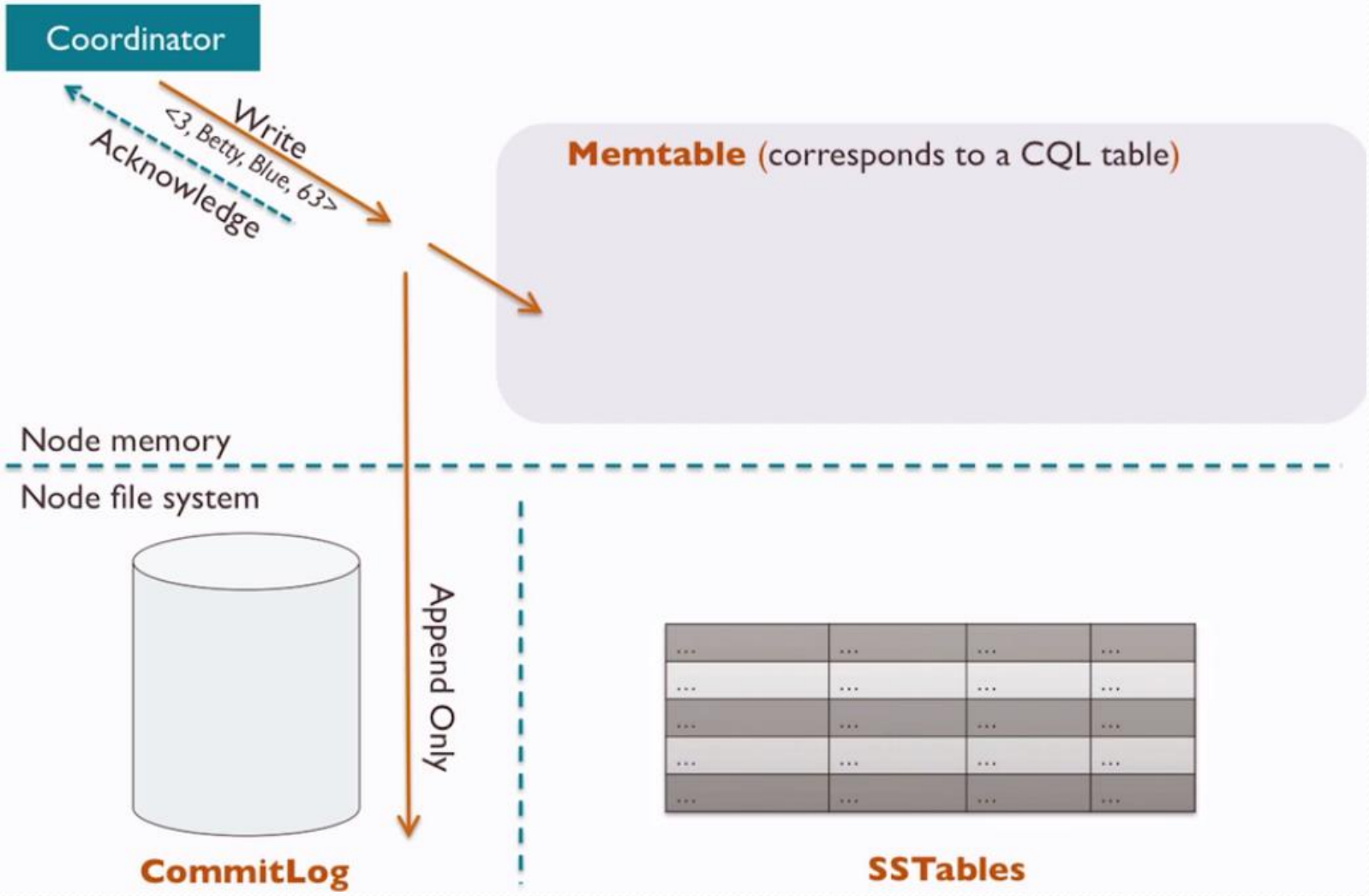
How does the write path flow on a node?



How does the write path flow on a node?



How does the write path flow on a node?



What is the CommitLog and how is it configured?

- An append-only log used to automatically rebuild Memtables on restart of a downed node, configured in conf/cassandra.yaml
- Memtables flush to disk when CommitLog size reaches total allowed space
  - **commitlog_total_space_in_mb** - size at which oldest Memtable log segment will be flushed to disk (default: l024 for 64bit JVMs)
  - **commitlog_segment_size_in_mb** — max size of individual log segments (default: 32)
- Entries are marked as flushed, as corresponding Memtable entries flush to disk as an SSTable
  - Flushed CommitLog segments are periodically recycled
- Best practice is to locate CommitLog on its own disk to minimize write head movement, or on SSD
  - **commitlog_directory** - default is /var/lib/cassandra/commitlog (package install) or install_location/data/commitlog (binary tarball)

What are Memtables and how are they flushed to disk?

- Memtables are in-memory representations of a CQL table
  - Each node has a Memtable for each CQL table in the keyspace
  - Each Memtable accrues writes and provides reads for data not yet flushed
  - Updates to Memtables mutate the in-memory partition
- When a Memtable flushes to disk
  1. Current Memtable data is written to a new immutable SSTable on disk
  2. JVM heap space is reclaimed from the flushed data
  3. Corresponding CommitLog entries are marked as flushed

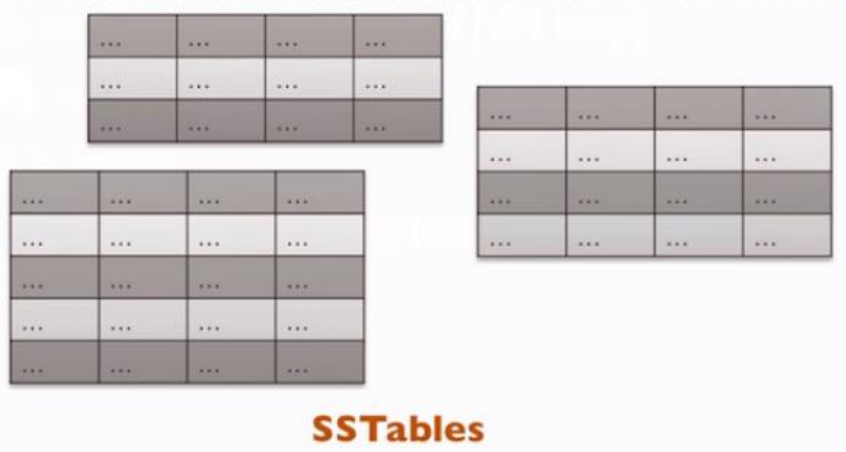
What are Memtables and how are they flushed to disk?

- A Memtable flushes the oldest CommitLog segments to a new corresponding SSTable on disk when
  - **memtable_total_space_in_mb** is reached (default: 25% of JVM heap)
  - **commitlog_total_space_in_mb** is reached
  - **nodetool flush** command is issued
- The *nodetool flush* command force-flushes designated Memtables

./nodetool flush [keyspace] [table(s)]

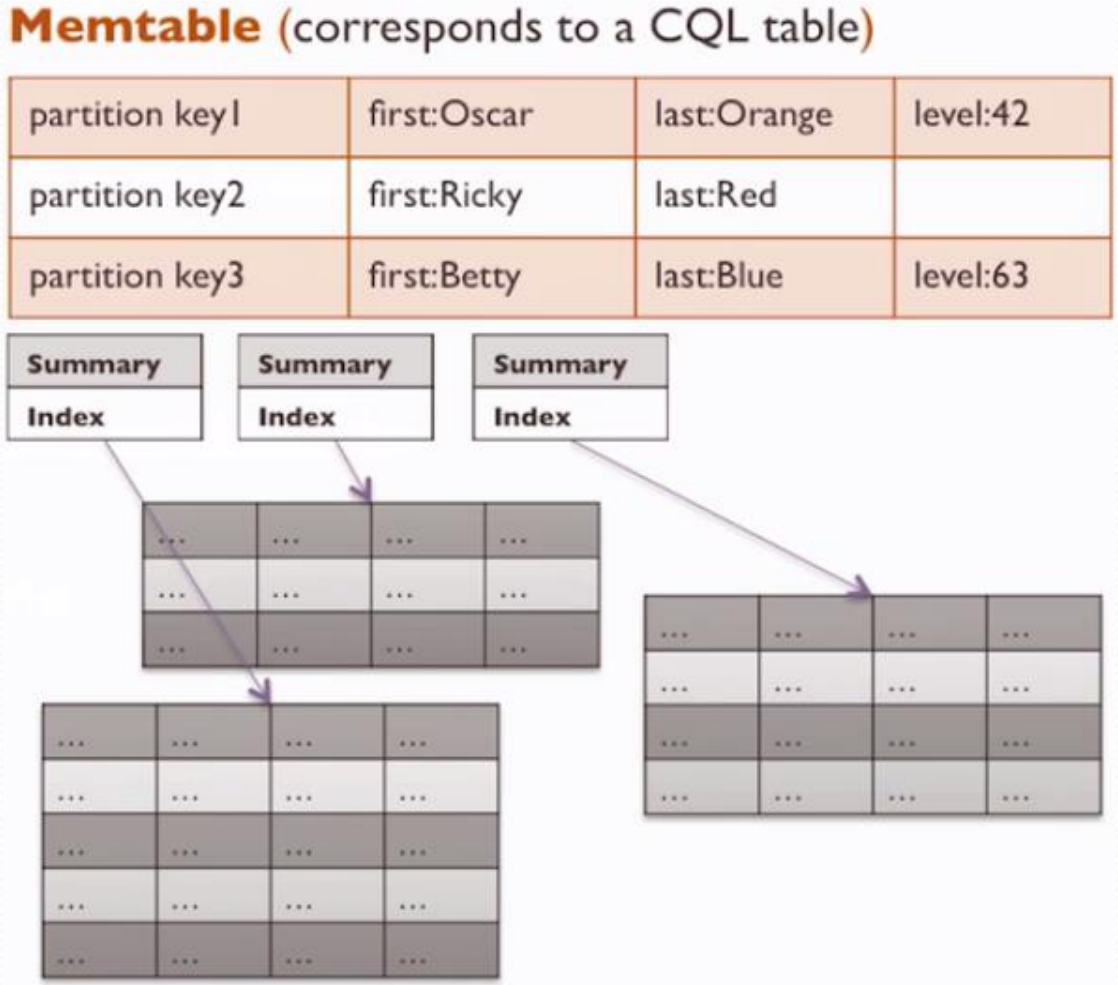
What is an SSTable and what are its characteristics?

- An SSTable ("sorted string table") is
  - an immutable file of sorted partitions
  - written to disk through fast, sequential i/o
  - contains the state of a Memtable when flushed
- The current data state of a CQL table is comprised of
  - its corresponding Memtable plus
  - all current SSTables flushed from that Memtable
- SSTables are periodically compacted from many to one



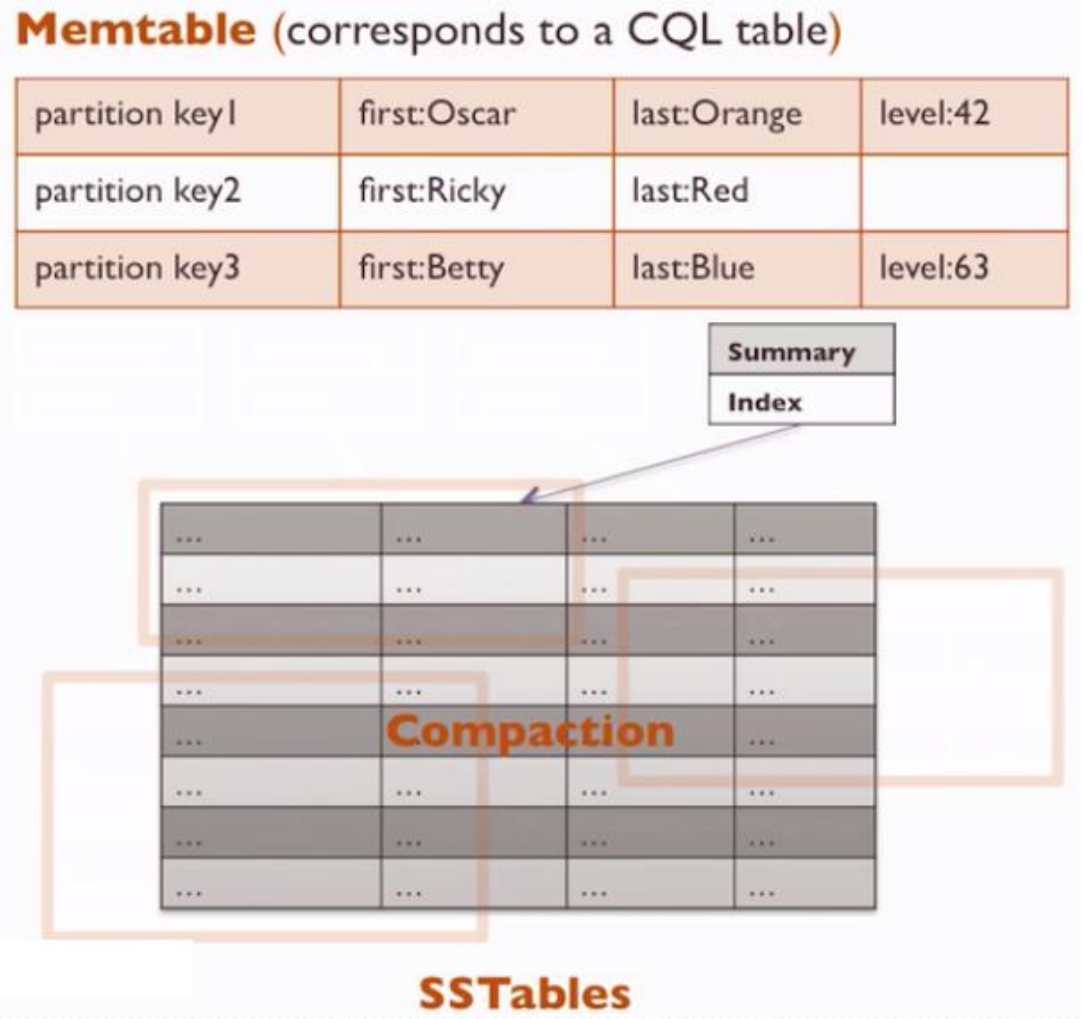
What is an SSTable and what are its characteristics?

- For each SSTable, two structures are created
  - **Partition index** - list of its primary keys and row start positions
  - **Partition summary** — in-memory sample of its partition index (default: I partition key of 128)



What is compaction?

- Updates do mutate Memtable partitions, but its SSTables are immutable
  - no SSTable seeks/overwrites
  - SSTables just accrue new timestamped updates
- So, SSTables must be periodically compacted
  - related SSTables are merged
  - most recent version of each column is compiled to one partition in one new SSTable
  - partitions marked for deletion are evicted
  - old SSTables are deleted



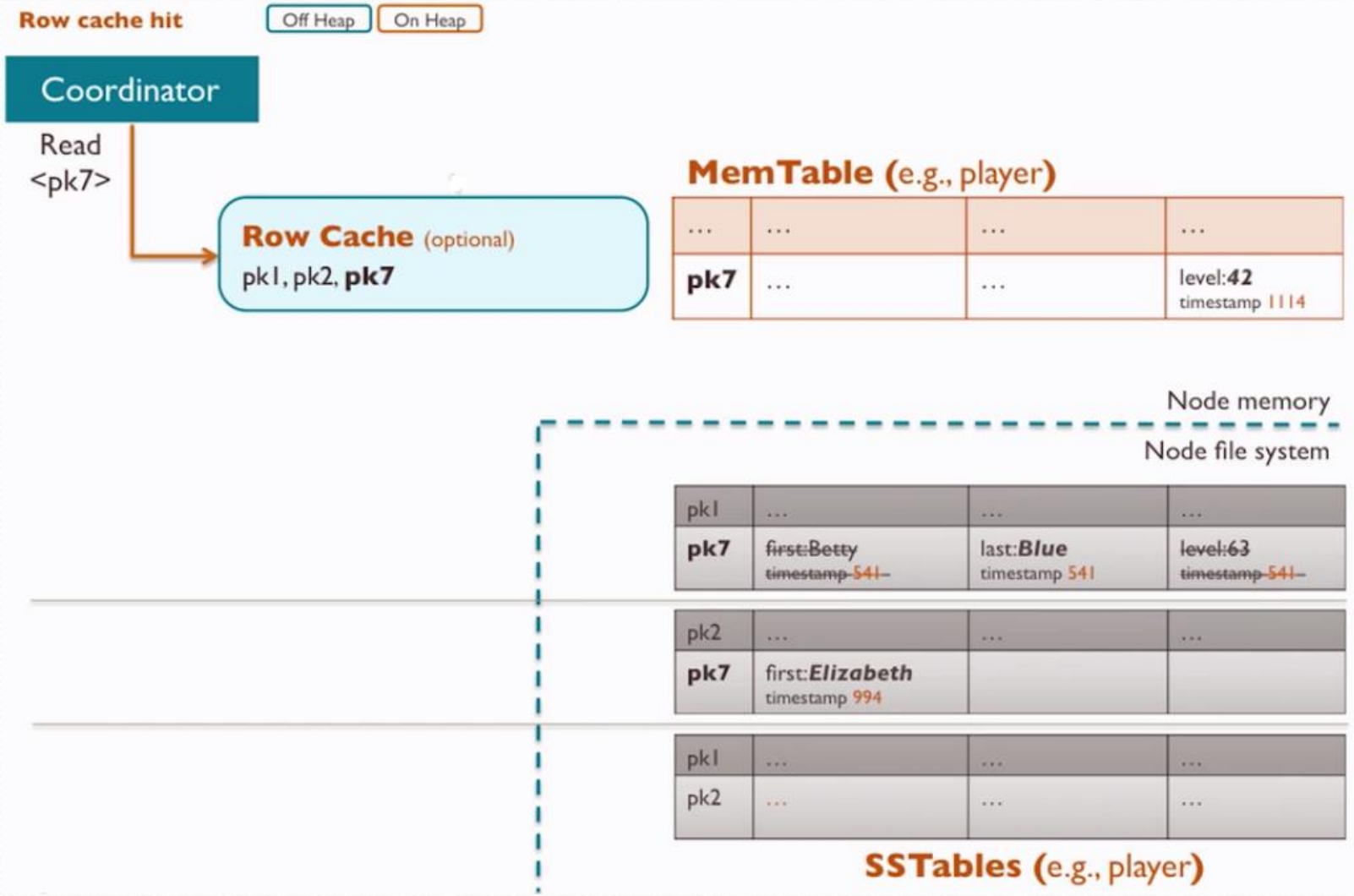
Cassandra Read Path

- Cassandra identifies which node owns the token you're trying to read based on your partitioning, replication and placement strategies.
- First checks the Bloom filter, which can save us some time.
  - A space-efficient structure that tests if a key is on the node.
  - False positives are possible.
  - False negatives are impossible.
- Then checks the index.
  - Tells us which *SStable* file the data is in.
  - And how far into the *SStable* file to look so we don't need to scan the whole thing

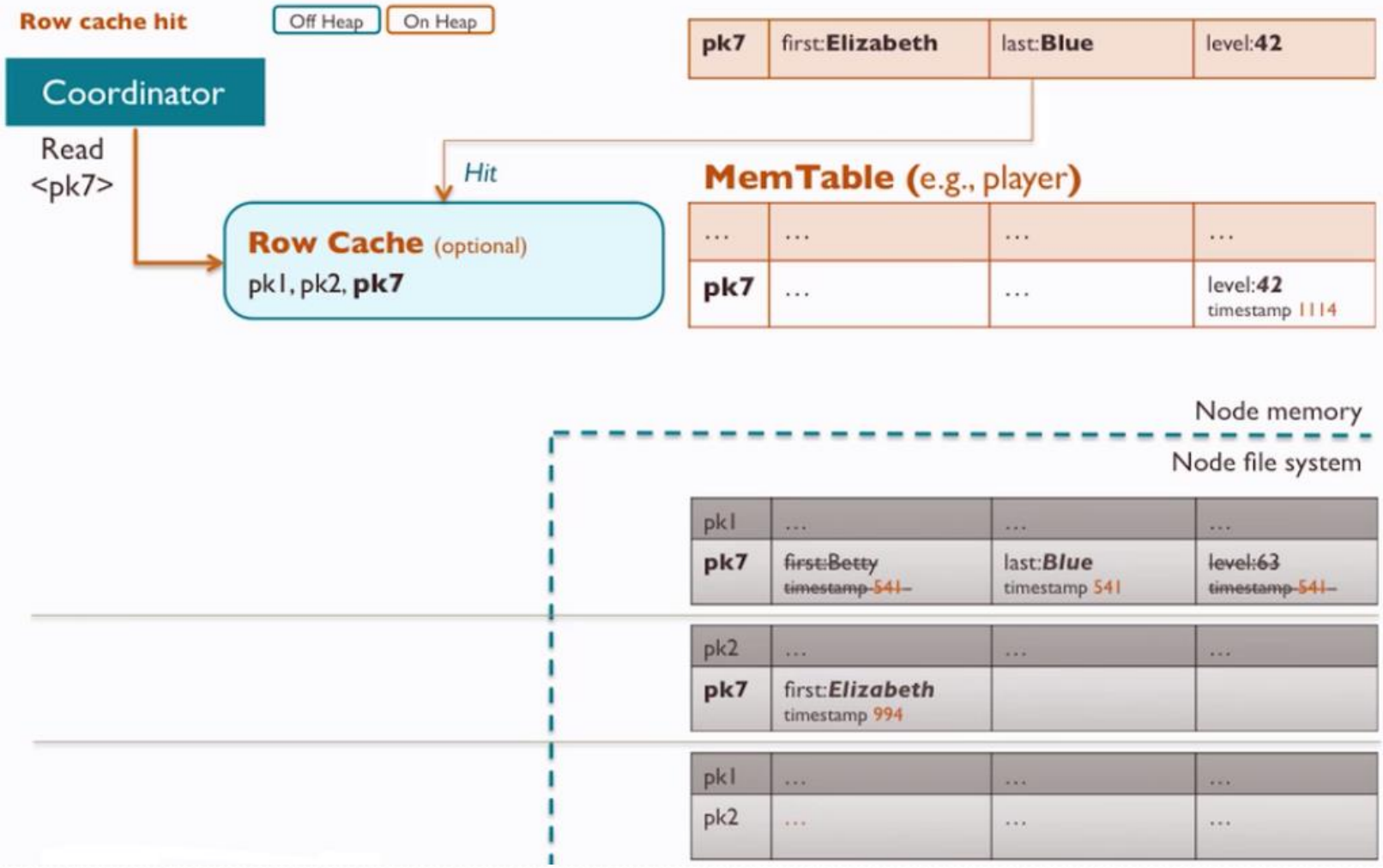
Key components of the read path

- Each node implements in-memory structures for each CQL table
  - **MemTable** - in-memory table serves data as part of the merge process
  - **Row Cache** - in-memory cache stores recently read rows (optional)
  - **Bloom Filters** - reports if a partition key M be in its corresponding SSTable
  - **Key Caches** - maps recently read partition keys to specific SSTable offsets
  - **Partition Summaries** - Sampling from partition index
- Each node implements these on disk for each CQL table
  - Partition Indexes - Sorted partition keys mapped to their SSTable offsets
  - SSTables - static files periodically flushed from a MemTable
- Merge - unless served from the row cache, a read uses a partition key to locate, merge, and return values from a MemTable and any related SSTable storing values for that key

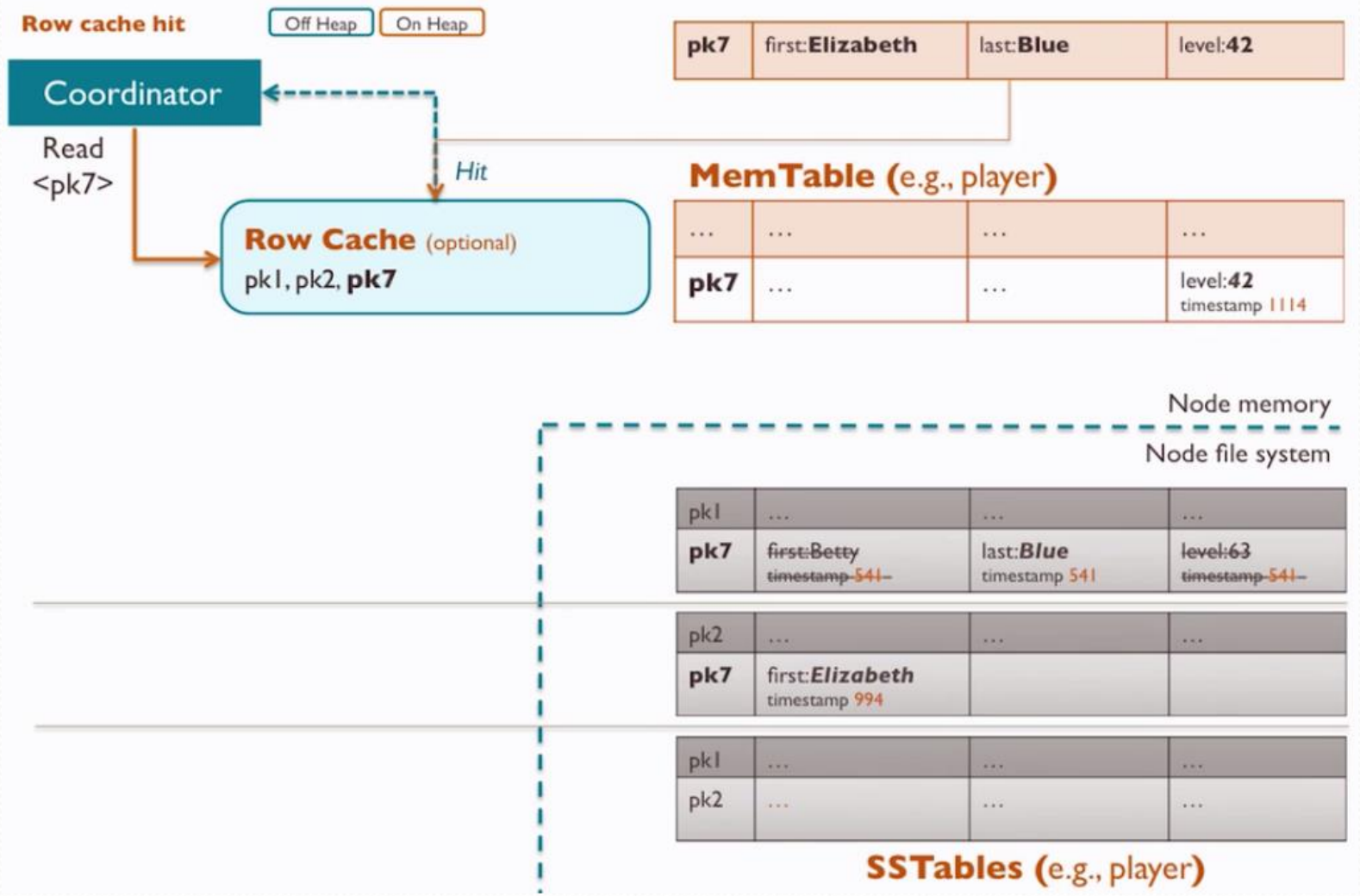
How does the read path flow on a node?



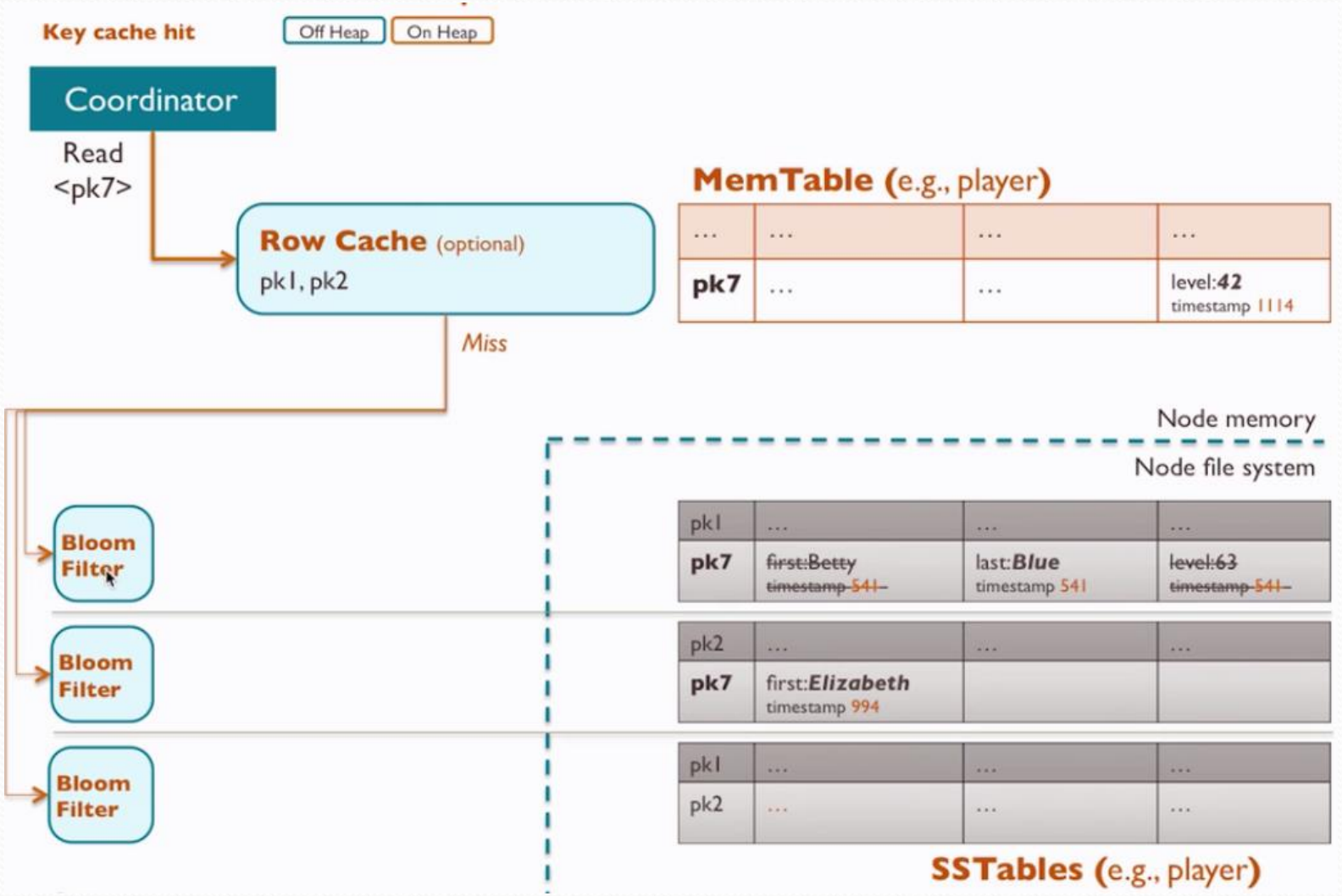
How does the read path flow on a node?



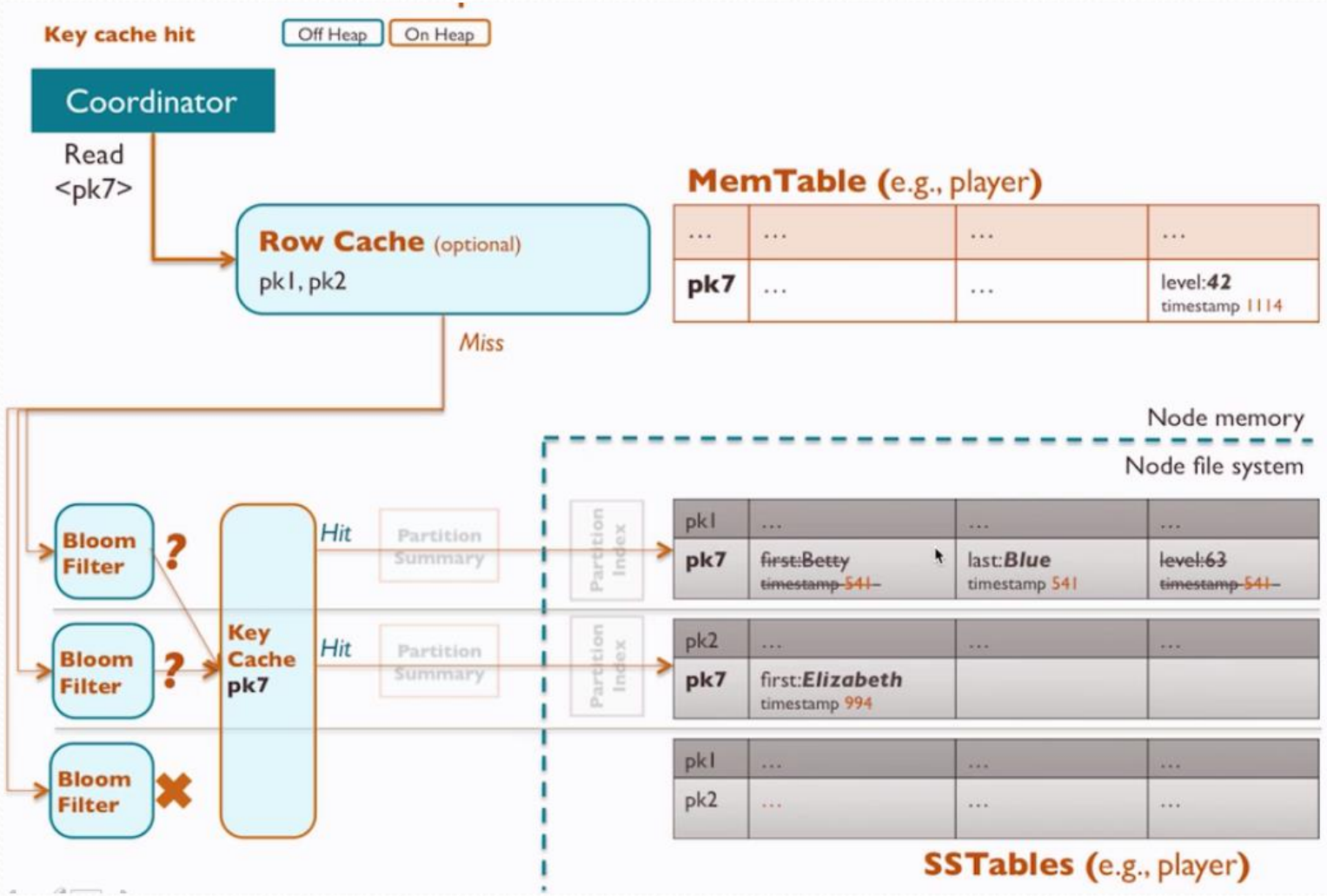
How does the read path flow on a node?



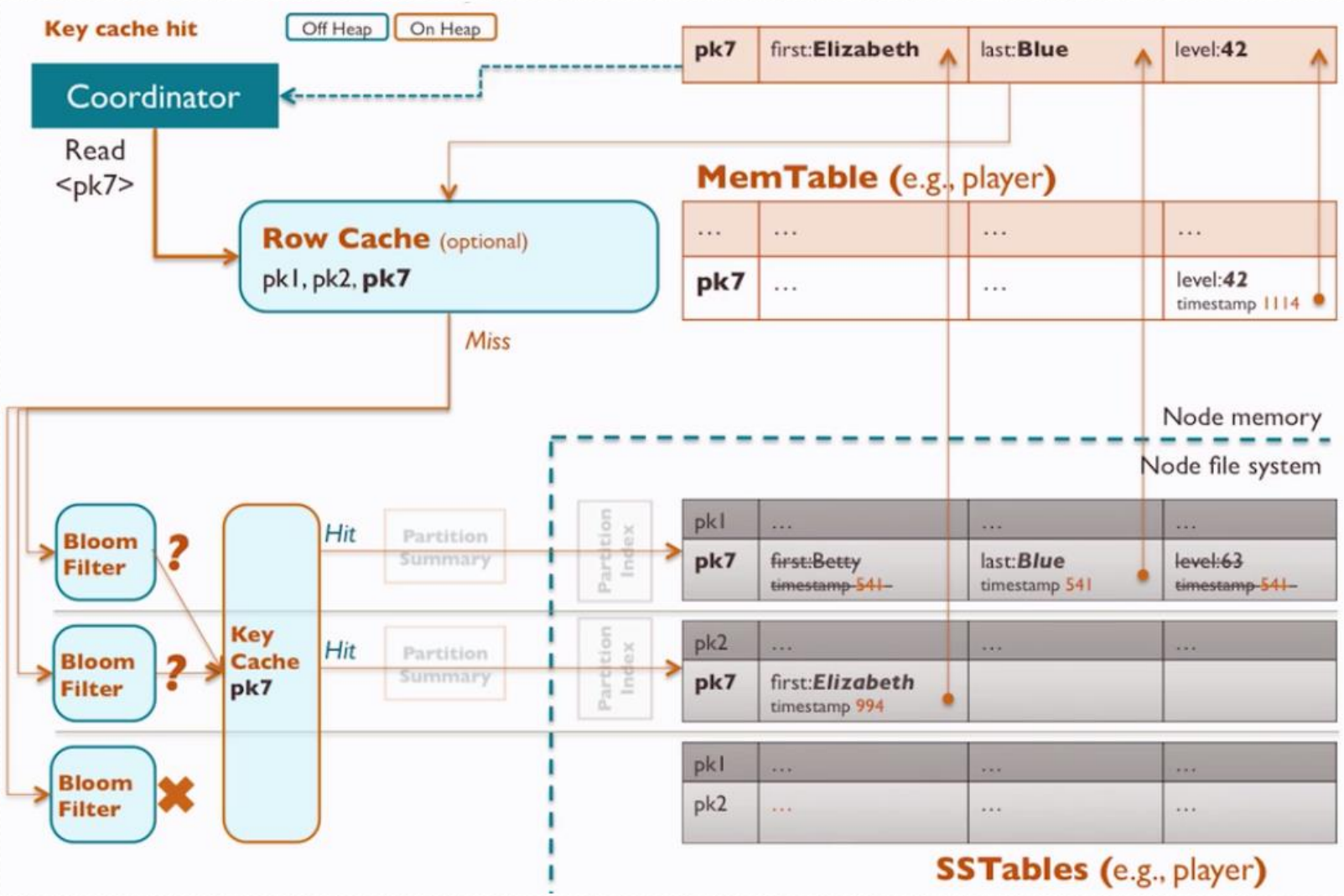
How does the read path flow on a node?



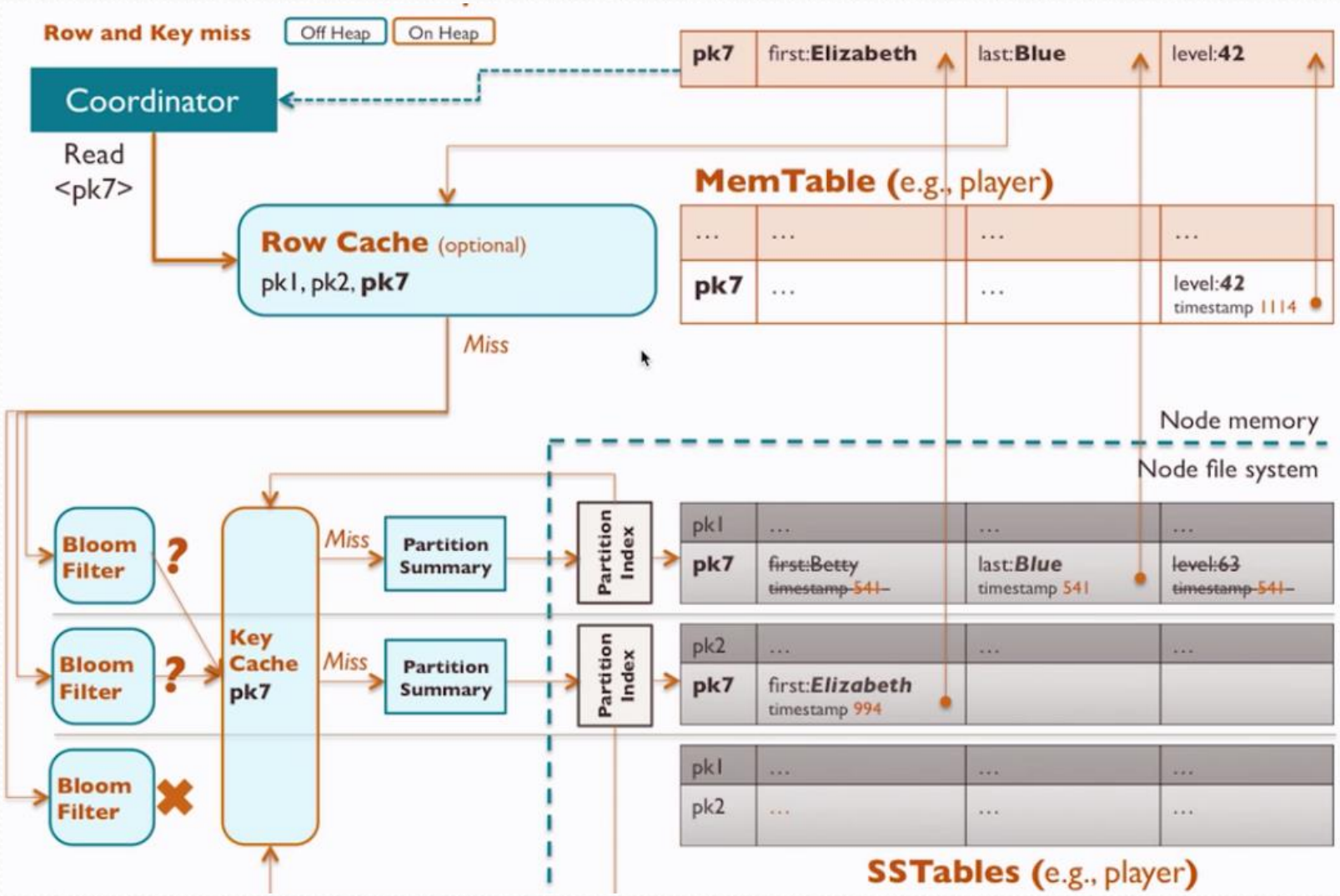
How does the read path flow on a node?



How does the read path flow on a node?

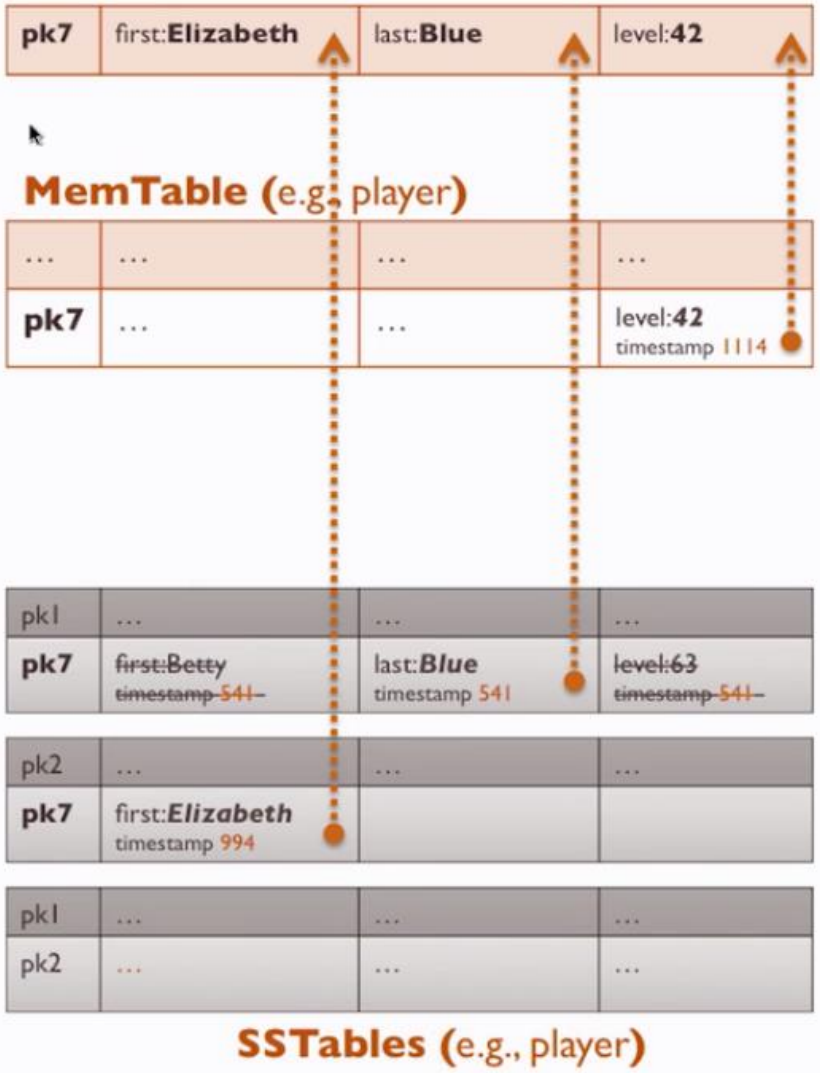


How does the read path flow on a node?



How is a MemTable and its SSTables used during a read?

- Both a MemTable and its recent SSTables are checked when reading for a partition key
  - the most current column values are combined to form the result



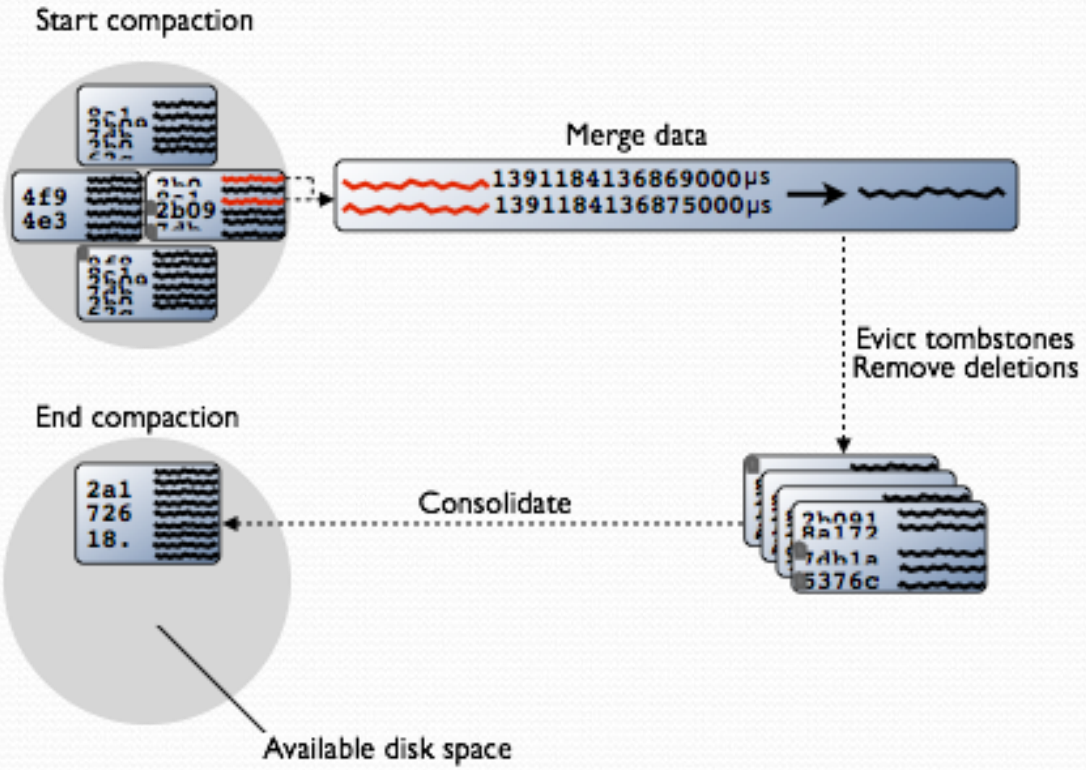
Delete Operations

- Tombstones
  - On delete request, records are marked for deletion.
  - Similar to “Recycle Bin.”
  - Data is actually deleted on major compaction or configurable timer

Compaction

- Compaction runs periodically to merge multiple SSTables
  - Reclaims space
  - Creates new index
  - Merges keys
  - Combines columns
  - Discards tombstones
  - Improves performance by minimizing disk seeks
- Two types
  - Major
  - Read-only

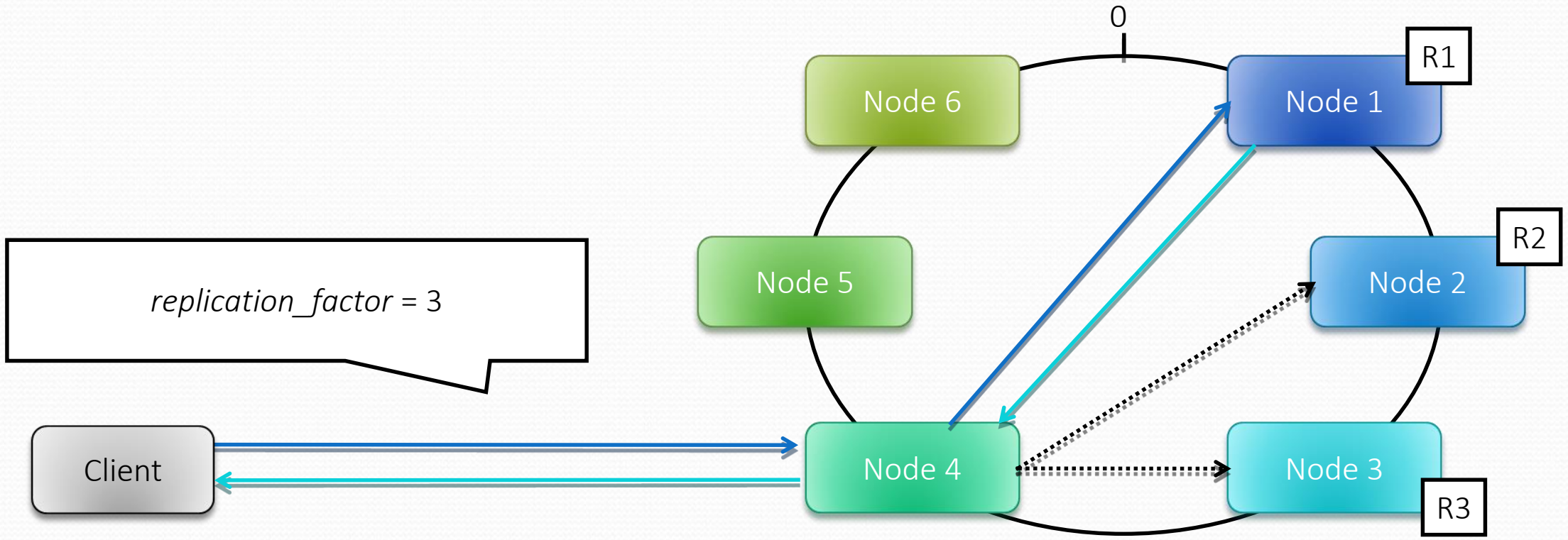
Compaction


Start compaction
4f9
4e3
2b09
Merge data
1391184136869000µs
1391184136875000µs
Evict tombstones
Remove deletions
End compaction
2a1
726
18.
Consolidate
Available disk space

Read Operations

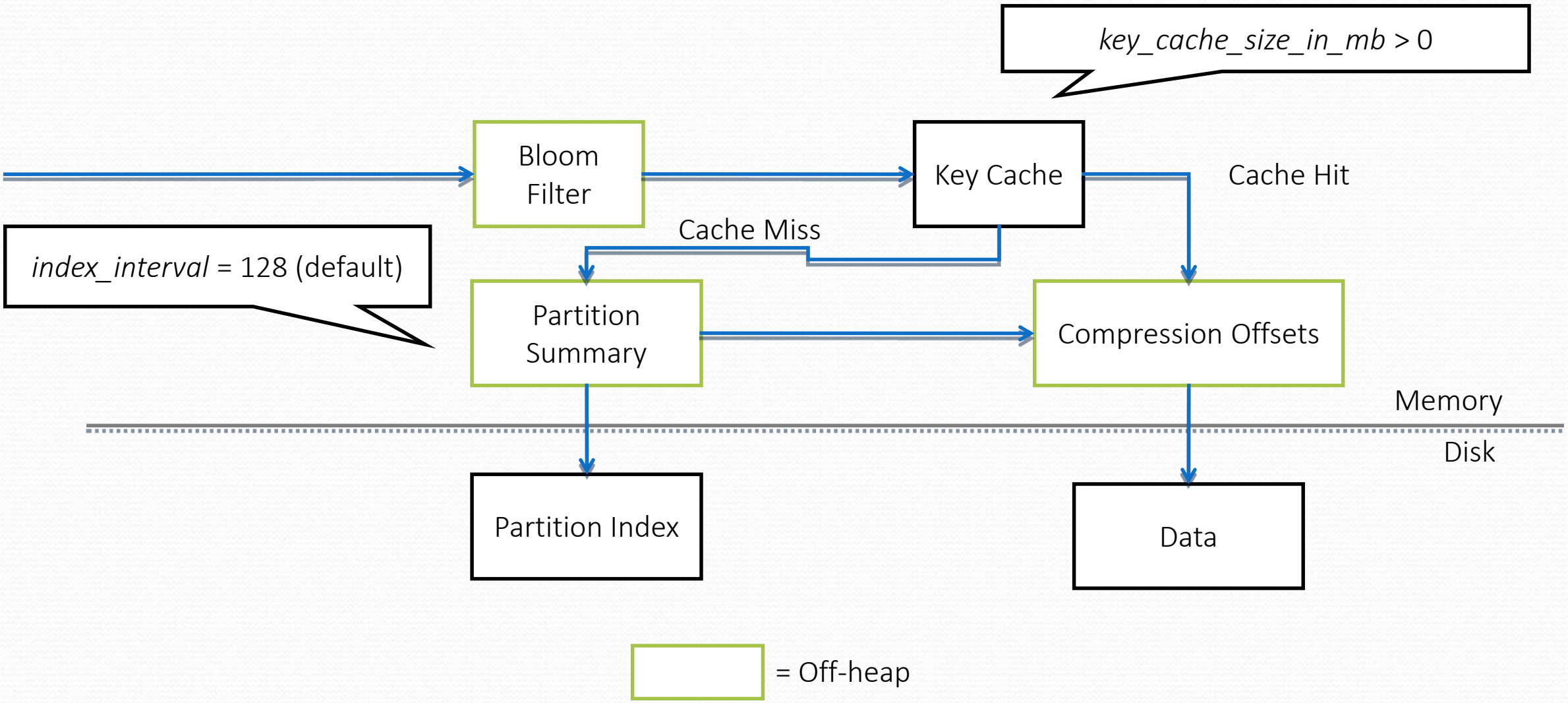
- **Read Repair**
  - On read, nodes are queried until the number of nodes which respond with the most recent value meet a specified consistency level from ONE to ALL.
  - If the consistency level is not met, nodes are updated with the most recent value which is then returned.
  - If the consistency level is met, the value is returned and any nodes that reported old values are then updated.

Read Repair



SELECT * FROM table USING CONSISTENCY ONE

Read


key_cache_size_in_mb > 0
Bloom Filter
Key Cache
Cache Hit
Cache Miss
index_interval = 128 (default)
Partition Summary
Compression Offsets
Memory
Disk
Partition Index
Data
= Off-heap

Conclusion

Cassandra Advantages

- Perfect for time-series data
- High performance
- Decentralization
- Nearly Linear Scalability
- Replication support
- No single points of failure
- MapReduce support

Cassandra Weaknesses

- No referential integrity
  - No concept of JOIN
- Querying options for retrieving data are limited
- Sorting data is a design decision
  - No GROUP BY
- No support for atomic operations
  - If operation fails, changes can still occur
- First think about queries, then about data model

Cassandra use cases

- Cassandra is particularly useful for
  - Playlists and collections (such as Spotify)
  - Personalization and recommendation engines (such as Ebay)
  - Messaging (such as Instagram)
  - Fraud detection (such as Barracuda)
  - Sensor data (such as Zonar)
- Many, many functional and industry use cases available
  - http://planetcassandra.org/functional-use-cases/

Data directories

Where are the data directories located?

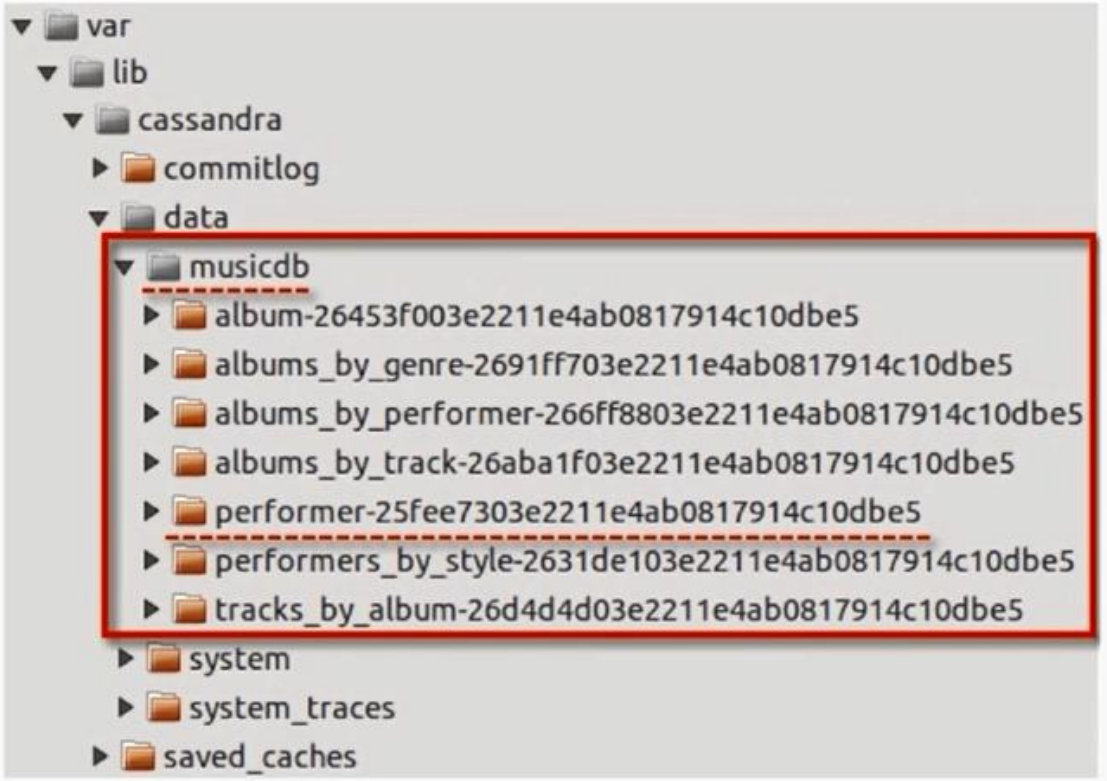
- The SSTable and CommitLog directory locations are configured in conf/cassandra.yaml
  - **data_file_directories** — if multiple locations, distribution is balanced
  - **commitlog_directory** — best practice to place on separate disk
  - By default, the files are all placed in /var/lib/cassandra or in install_location/data

```
# Directories where Cassandra should store data on disk.  Cassandra
# will spread data evenly across them, subject to the granularity of
# the configured compaction strategy.
data_file_directories:
    - /var/lib/cassandra/data

# commit log
commitlog_directory: /var/lib/cassandra/commitlog
```

How are data directories created for a keyspace?

- Data directories are created by keyspace and table name / id
  .../data/keyspace/tablename-tableid

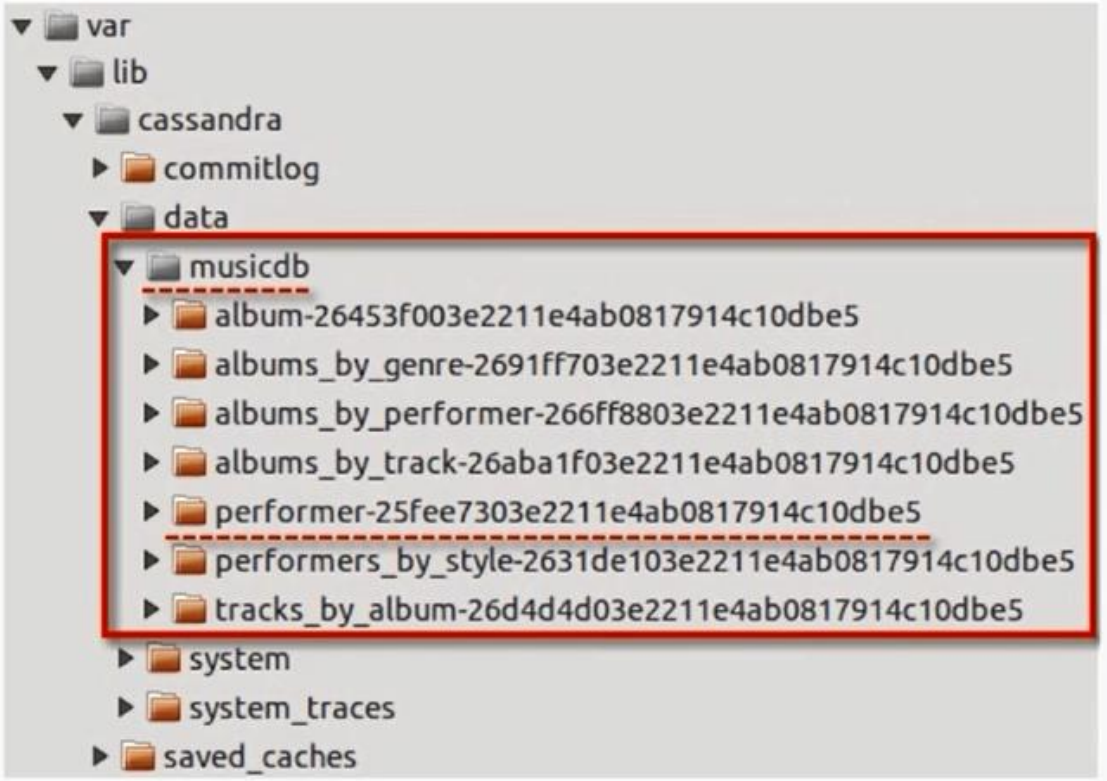
```
CREATE KEYSPACE musicdb
WITH replication = { 'class' : 'simplestrategy',
 'replication_factor' : 1 };

CREATE TABLE performer (
name VARCHAR,
type VARCHAR,
country VARCHAR,
style VARCHAR,
founded INT,
born INT,
died INT,
 PRIMARY KEY (name) ) ;
```



How are data directories created for a keyspace?

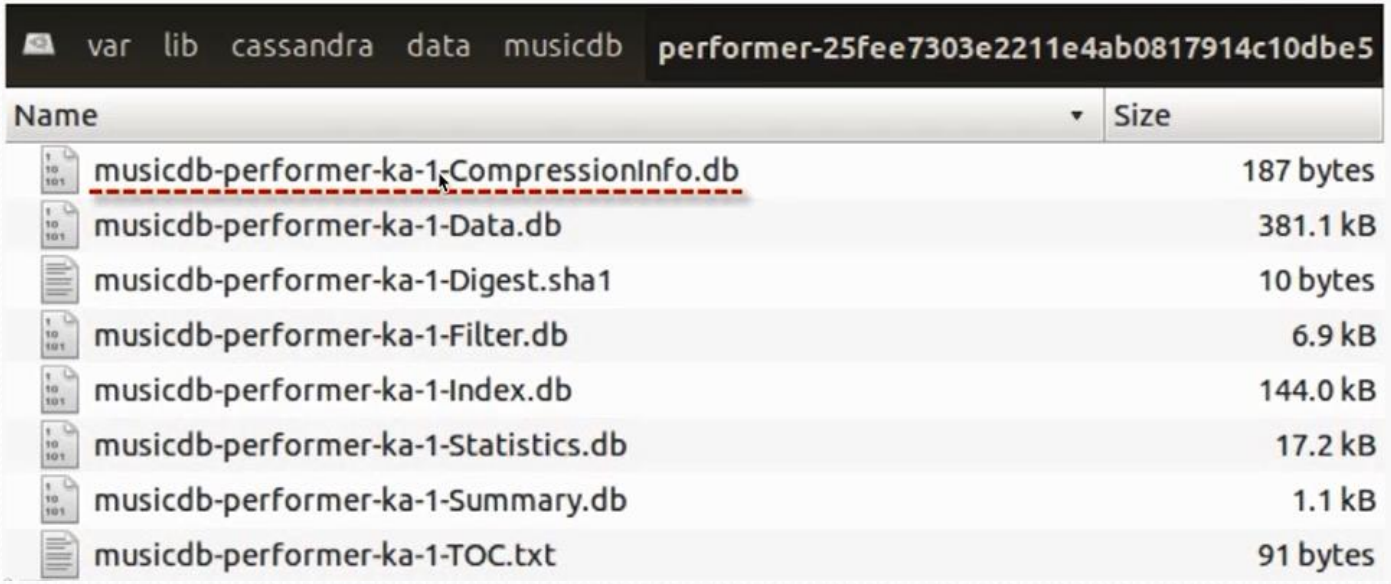
- Data directories are created by keyspace and table name / id
  .../data/keyspace/tablename-tableid

```
CREATE KEYSPACE musicdb
WITH replication = { 'class' : 'simplestrategy',
 'replication_factor' : 1 };

CREATE TABLE performer (
name VARCHAR,
type VARCHAR,
country VARCHAR,
style VARCHAR,
founded INT,
born INT,
died INT,
 PRIMARY KEY (name) ) ;
```



What files result from Memtable flush or compaction?

- Data files are created by keyspace name, table name, plus
  - Version — SSTable format version (e.g., ‘ka' is Cassandra 2.1)
  - Generation — incremented each time SSTables flush from a Memtable
  - Component — describes the type of file content
- <keyspace>-<table>-<version>-<generation>-<component>



What files result from Memtable flush or compaction?

- -CompressionInfo.db — metadata for Data file compression
- -Data.db — base SSTable data including
  - row key, data size, columns index, row level tombstone info, column count, and column list in sorted order by name
- -Filter.db — SSTable partition keys Bloom filter, to optimize reads
- -Index.db — index for this SSTable, used to optimize reads
  - sorted row keys mapped to offsets in Data file; newer versions also include column index, tombstone, and bloom filter info
- -statistics.db — statistics for this SSTable
  - row size and column count estimate, generation numbers of files from which this SSTable was compacted, more
- -summary.db — sampling from Index file, used to optimize reads
  - sample size determined by index_interval (default: 1 of each 128)
- -TOC.txt - component list for this SSTable

What is sstable2json?

- tools/bin/sstable2json is a utility which exports an SSTable in JSON format, for testing and debugging
  - -k display only the partitions for the specified set of keys (limit: 500)
  - -x exclude a specified set of keys (limit: 500)
  - -e enumerate keys only
- ./sstable2json [full_path_to_ssTable_Data_file] I more

```
student@cascor:~/cassandra/bin$ sstable2json ~/cassandra/data/data/musicdb/
performer-40f07fd06a3d11e49732d16f287d9d64/musicdb-performer-ka-1-Data.db
[
{"key": "Sheryl Crow",
 "cells": [["","",1415777208220432],
           ["born","1962",1415777208220432],
           ["country","United States",1415777208220432],
           ["died",1415777208,1415777208220432,"d"],
           ["founded",1415777208,1415777208220432,"d"],
           ["style","Rock",1415777208220432],
           ["type","artist",1415777208220432]]},
{"key": "Black Bottle Scotch Whisky Pipe Band",
 "cells": [["","",1415777208220560],
           ["born",1415777208,1415777208220560,"d"],
           ["country","Scotland",1415777208220560],
           ["died",1415777208,1415777208220560,"d"],
```

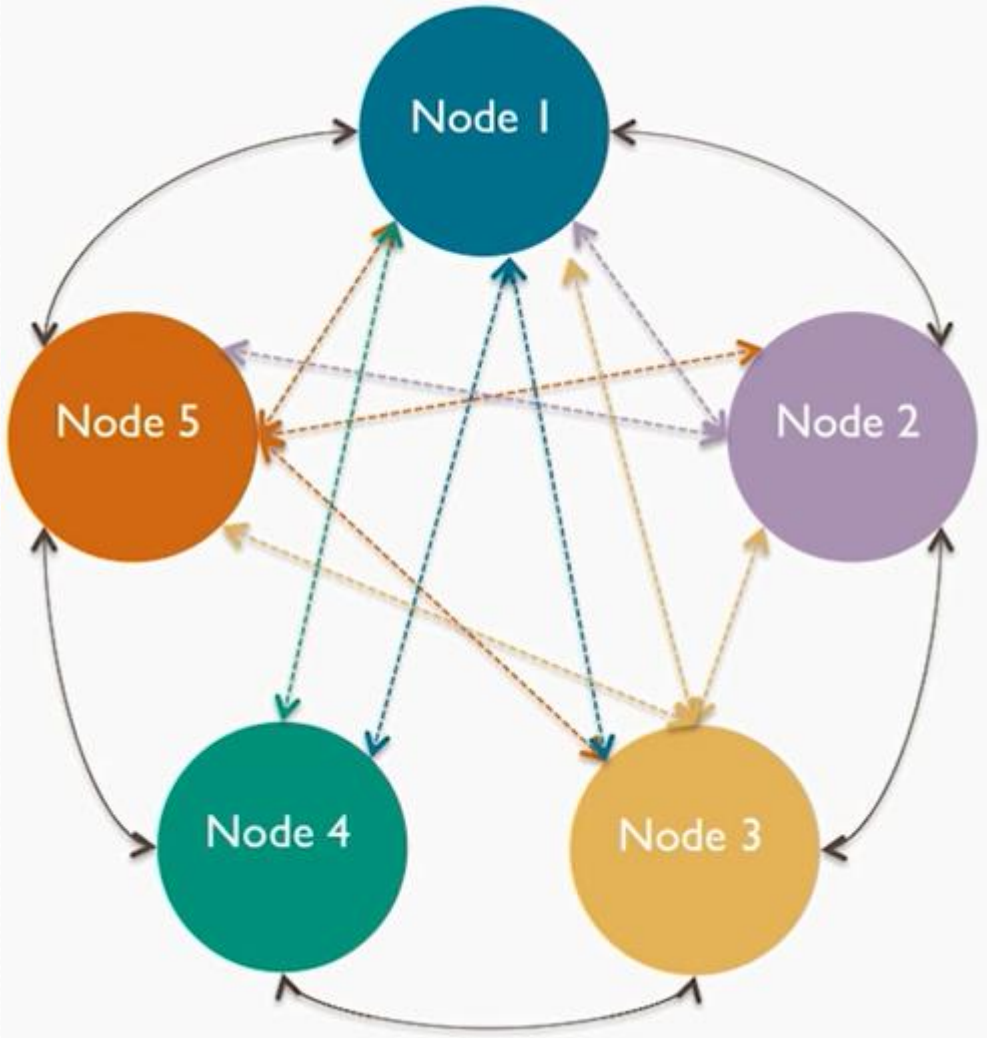
Cassandra Utilities

- nodetool
  - For operating and monitoring Cassandra nodetool is the utility
- cqlsh
  - Cassandra query language shell (cqlsh) that allows users to communicate with Cassandra
- ccm
  - Cassandra cluster manager is tool that creates multi-node Cassandra clusters on the local machine

http://www.datastax.com/dev/blog/ccm-a-development-tool-for-creating-local-cassandra-clusters

What is the Gossip protocol?

- Once per second, each node contacts I to 3 others, requesting and sharing updates about
  - Known node states ("heartbeats")
  - Known node locations
  - Requests and acknowledgments are time stamped, so information is continually updated and discarded



What is the Gossip protocol?

- As a node joins a cluster, it gossips with the seed nodes set in its cassandra.yaml to learn its cluster's topology
  - Assign the same seed nodes to each node in each data center
  - If more than one data center, include a seed node from each

cassandra.yaml

```
# any class that implements the SeedProvider interface and has a
# constructor that takes a Map<String, String> of parameters will do.
seed_provider:
    # Addresses of hosts that are deemed contact points.
    # Cassandra nodes use this list of hosts to find each other and learn
    # the topology of the ring.  You must change this if you are running
    # multiple nodes!
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
          # seeds is actually a comma-delimited list of addresses.
          # Ex: "<ip1>,<ip2>,<ip3>"
          - seeds: "127.0.0.1"
```

Some Cassandra Data Model Objectives

- SQL-like interface (CQL)
- Tabular presentation of results
- Easy, deterministic distribution and replication of data across machines and data centers
- Fast read performance for known queries
- Fast, scalable writes