

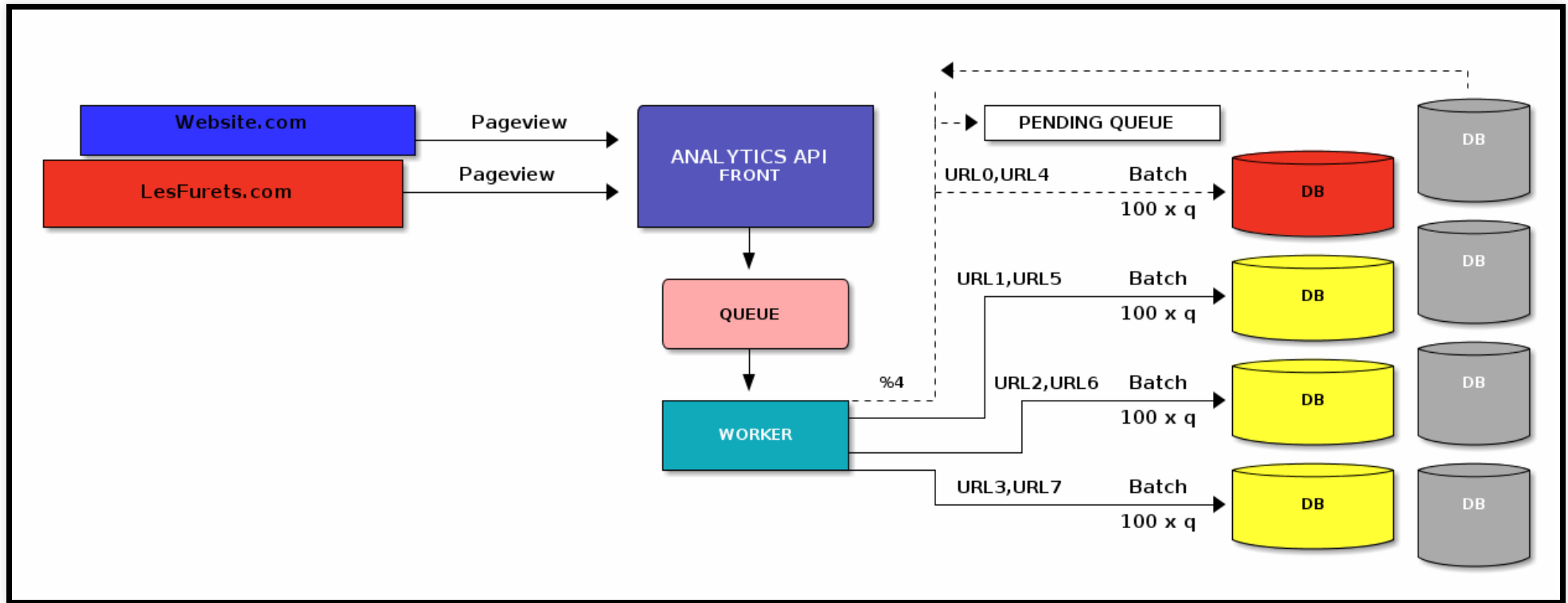
Introduction to Apache Cassandra

Andrei Arion, LesFurets.com, tp-bigdata@lesfurets.com

Plan

- **Motivation**
- Apache Cassandra
 - Partitioning and replication
 - Consistency
- Practice: Tune consistency in Apache Cassandra

Motivation



- do I build new features for customers?
- or just dealing with reading/writing the data?

What went wrong?

- A single server cannot take the load \Rightarrow solution / complexity
 - *Better database*
 - easy to add/remove nodes (**scaling**)
 - transparent data distribution (**auto-sharding**)
 - handle failures (**auto-replication**)

Plan

- Motivation
- **Apache Cassandra**
 - Partitioning and replication
 - Consistency
- Practice: Tune consistency in Apache Cassandra

Apache Cassandra

- started @Facebook inspired by *BigTable* model and *Amazon DynamoDB*
- 2008 Open Source Project
- **Datastax**: commercial offering Datastax Enterprise
 - monitoring(OpsCenter) automating repairs backup...
 - other features: search, analytics, graph, encryption
- 2010 Top Level Apache Project
 - Datastax biggest committer
- 2016 Datastax/ASF disagreements([#Staxit](#))

Apache Cassandra

*Open source, distributed database designed to handle large amounts of data across many commodity servers, providing **high availability** with **no single point of failure**. It offers robust support for clusters spanning **multiple datacenters**, with **asynchronous masterless replication** allowing **low latency** operations for all clients.*

Apache Cassandra

- column oriented NoSQL database
- distributed (data, query)
- resilient (no SPOF)
 - we can query any node \Rightarrow coordinator to dispatch and gather the results
- reliable and simple scaling
- online load balancing and cluster growth

Apache Cassandra users



source: <https://codingjam.it/>

Cassandra terminology

RDBMS

Cassandra

Schema (set of tables)

Keyspace

Table

Table/column family

Row

Row

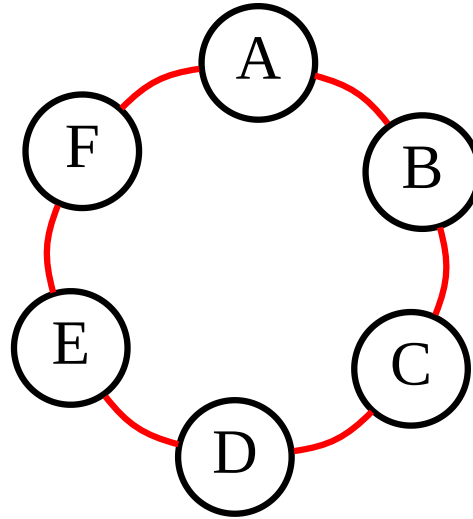
Database server

Node

Master/Slave

Cluster: a set of nodes grouped in one or more datacenters (can span physical locations)

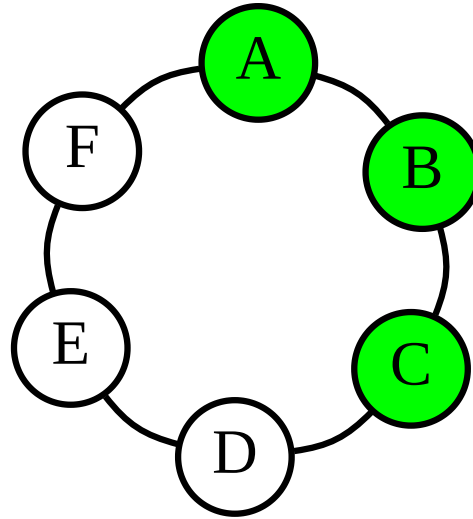
Cassandra ring



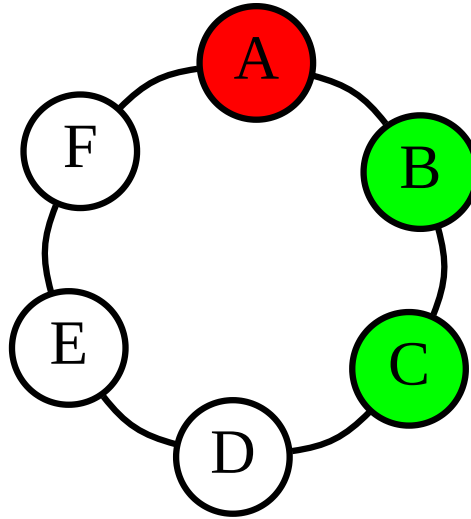
Gossip

- peer-to-peer communication protocol
 - discover and share *location* and *state information* about nodes
 - persist gossip info locally to use when a node restarts
- **seed nodes** \Rightarrow bootstrapping the gossip process for new nodes joining the cluster

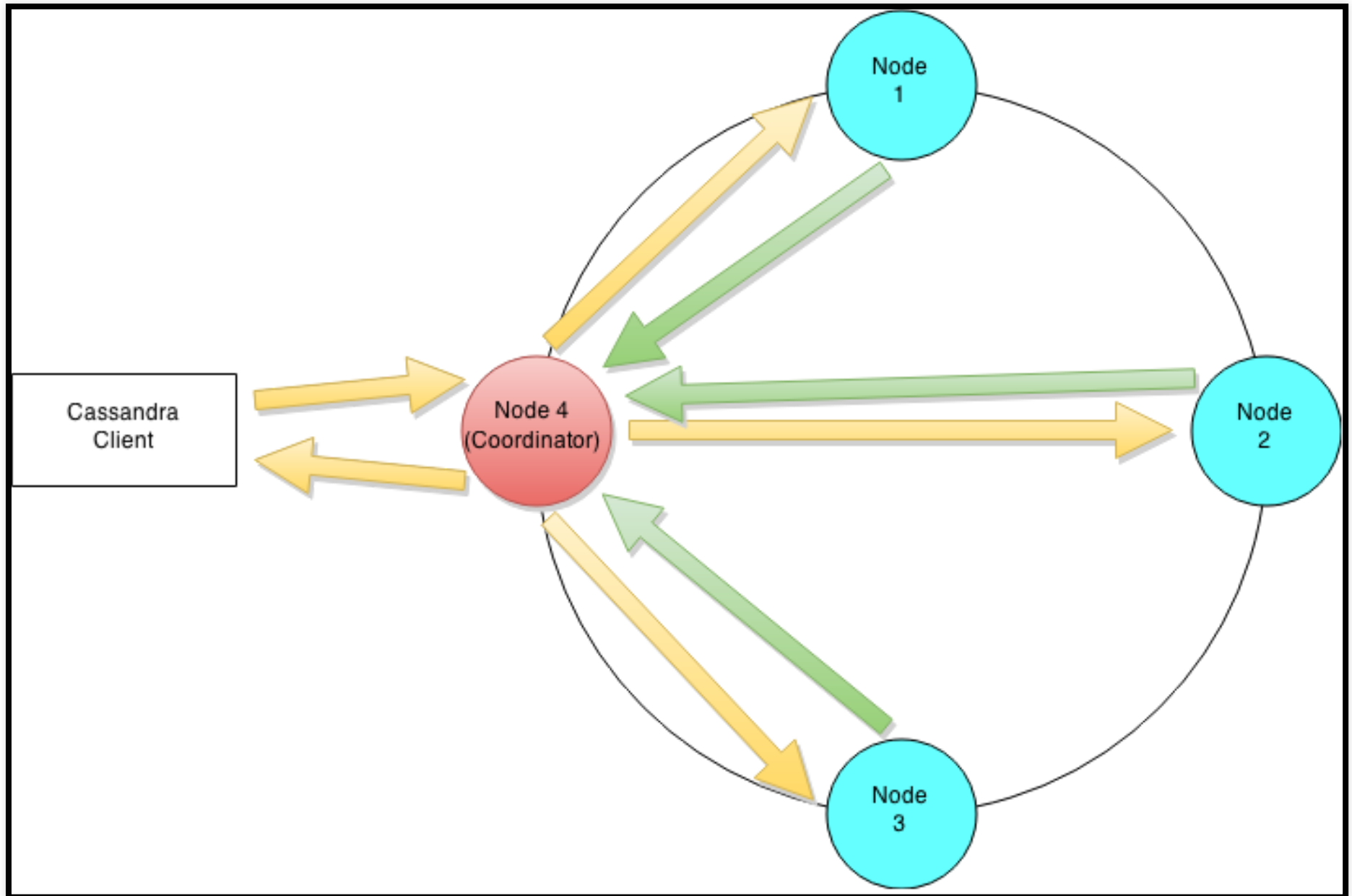
Cassandra replicas



Cassandra node failure



Query

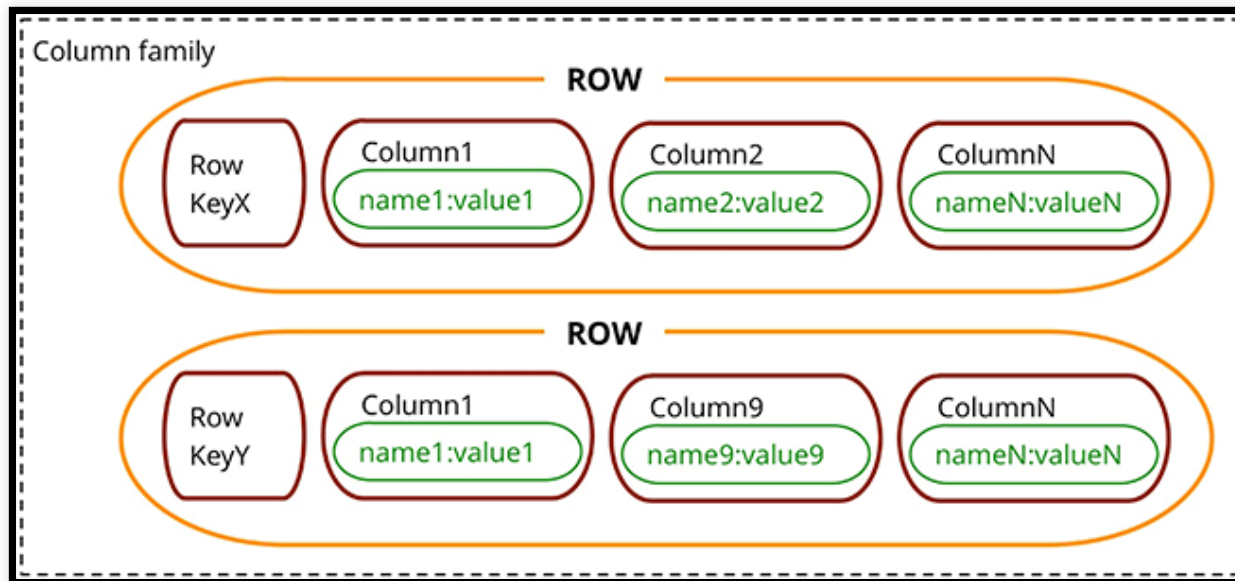


Plan

- Motivation
- Apache Cassandra
 - **Partitioning and replication**
 - Consistency
- Practice: Tune consistency in Apache Cassandra

Cassandra data model

- Stores data in **tables/column families** as rows that have many columns associated to a row key
- *Map<RowKey, SortedMap<ColumnKey, ColumnValue>>*



Data partitioning

- C^* = single logical database spread across a cluster of nodes
- *How to divide data evenly around its cluster of nodes?*
- **Problems:**
 - distribute data **efficiently, evenly**
 - determining a node on which a specific piece of data should reside on
 - minimise the data movements when nodes join or leave the cluster
- Algorithm of **Consistent Hashing**

Mapping data to nodes

- **Problem:** map k entries to n physical nodes
- *Naive hashing* ($NodeID = hash(key) \% n$) \Rightarrow remap a large number of keys when nodes join/leave the cluster
- ***Consistent hashing*:** only k/n keys need to be remapped on average

Consistent Hashing

- **Idea :**
 - use a part of the data as a partition key
 - compute a hash value for each
 - The range of values from a consistent hashing algorithm is a fixed circular space which can be visualised as a ring.

Partitioner

- **hash function** that derives a token from the primary key of a row
- determines which node will receive the *first replica*
- RandomPartitioner, **Murmur3Partitioner**, ByteOrdered

Murmur3Partitionner

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

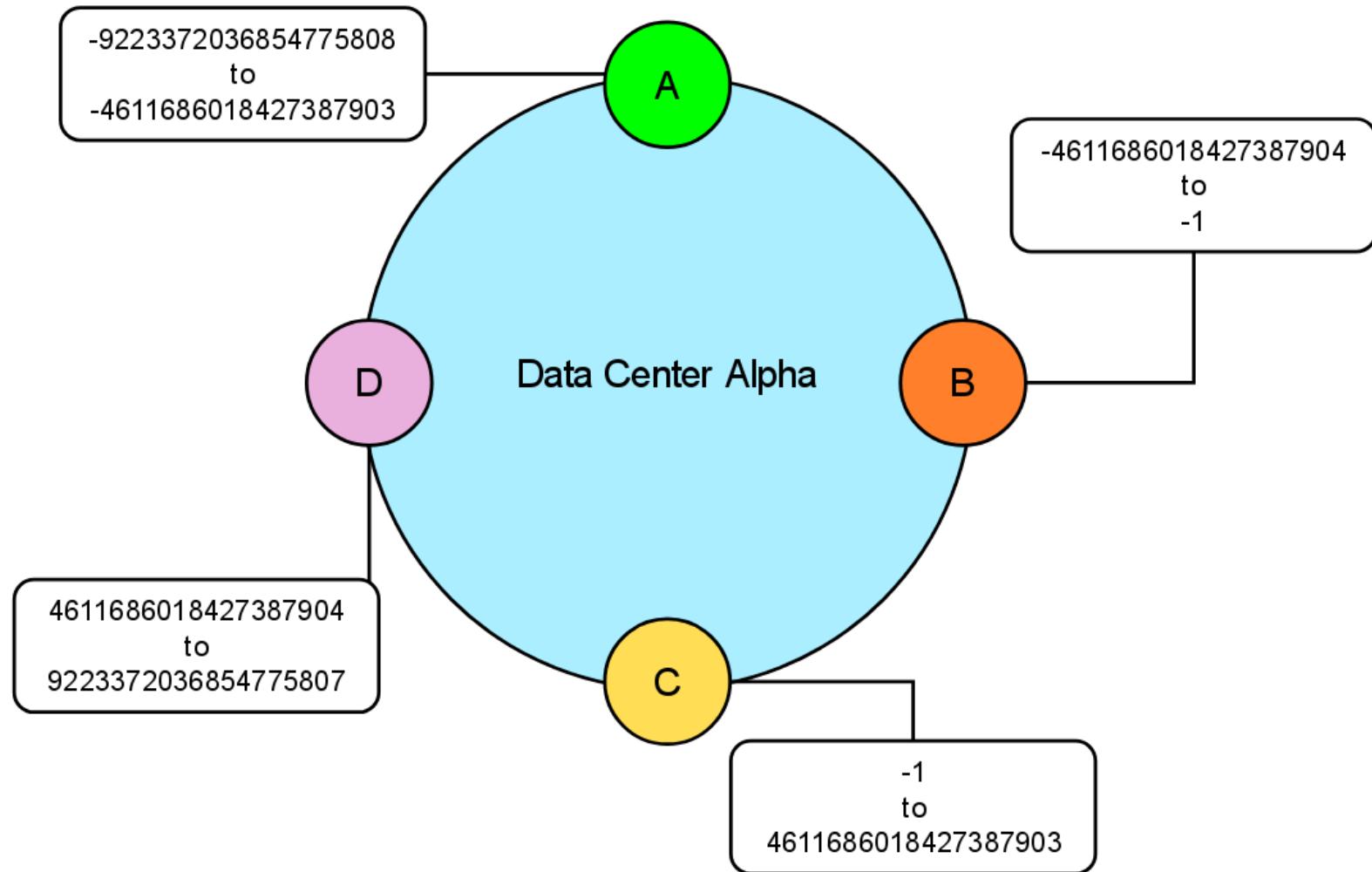
Cassandra assigns a hash value to each partition key:

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

Consistent Hashing: mapping

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

Hash values in a 4 node cluster



Consistent Hashing: mapping

No de	Start range	End range	Part itio n key	Hash value
A	-9223372036854775808	-4611686018427387903	john ny	-6723372854036780875
B	-4611686018427387904	-1	jim	-2245462676723223822
C	0	4611686018427387903	suz y	1168604627387940318
D	4611686018427387904	9223372036854775807	caro l	7723358927203680754

Data Replication

- Create copies of the data, thus avoiding a single point of failure.
- **Replication Factor (RF)** = # of replica for each data
 - set at the *Keyspace* level

Topology informations: *Snitches*

- Inform the database about the *network topology*
 - so requests are routed efficiently
 - and support replication by grouping nodes (racks/datacenters)
- ***SimpleSnitch*** ⇒ does not recognize datacenter or rack information
- ***RackInferringSnitch*** ⇒ infers racks and DC information
- ***PropertyFileSnitch*** ⇒ uses pre-configured rack/DC informations

Replication strategies

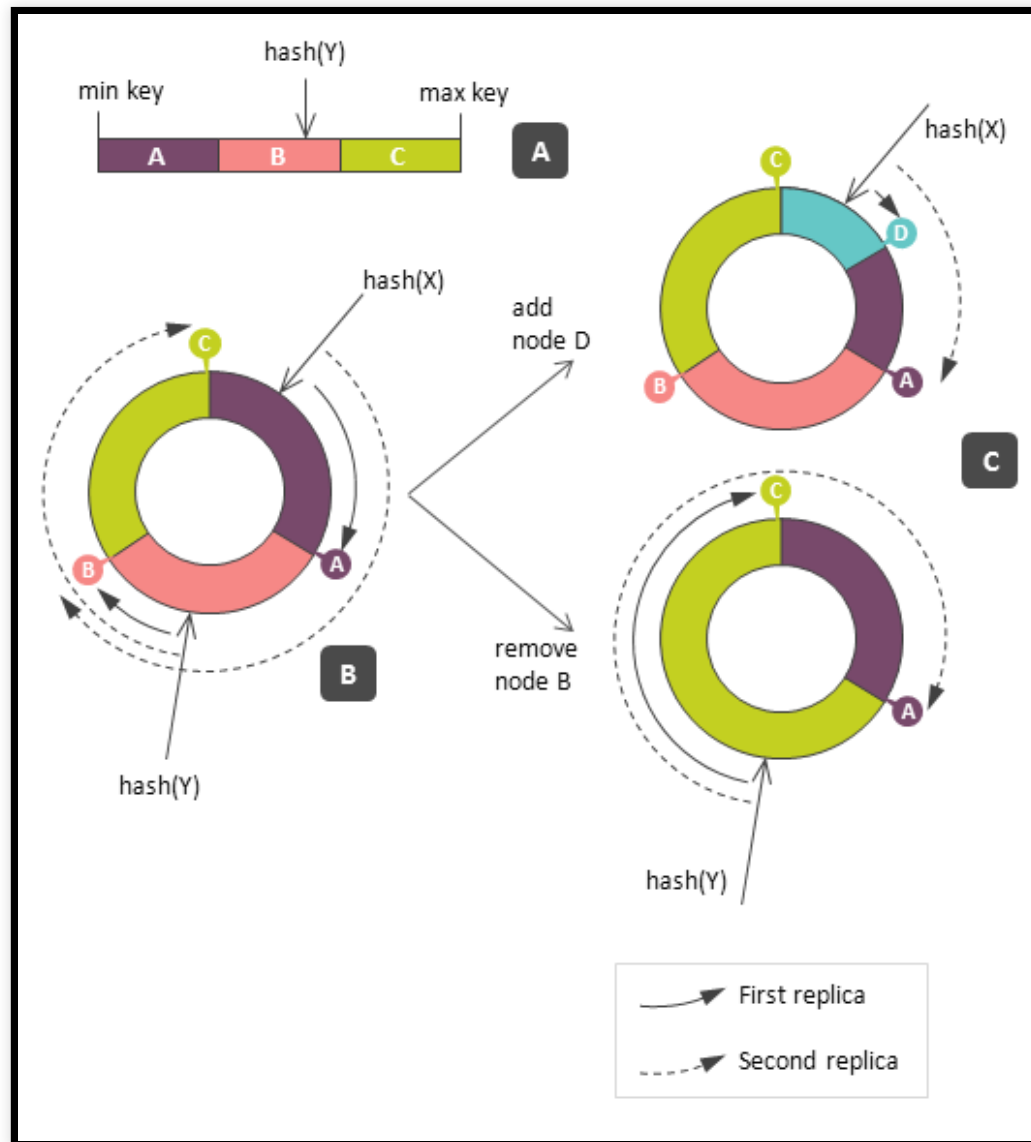
- use proximity information provided by snitches to determine locality of a copy
 - **SimpleStrategy:**
 - use only for a single data center and one rack
 - place the copy to the next available node (clockwise)
 - **NetworkTopologyStrategy:** specifies how many replicas you want in each DC
- defined at *keyspace* level

Replication strategies

```
CREATE KEYSPACE temperature  
WITH replication =  
  {'class': 'SimpleTopologyStrategy', 'replication_factor':'2'};
```

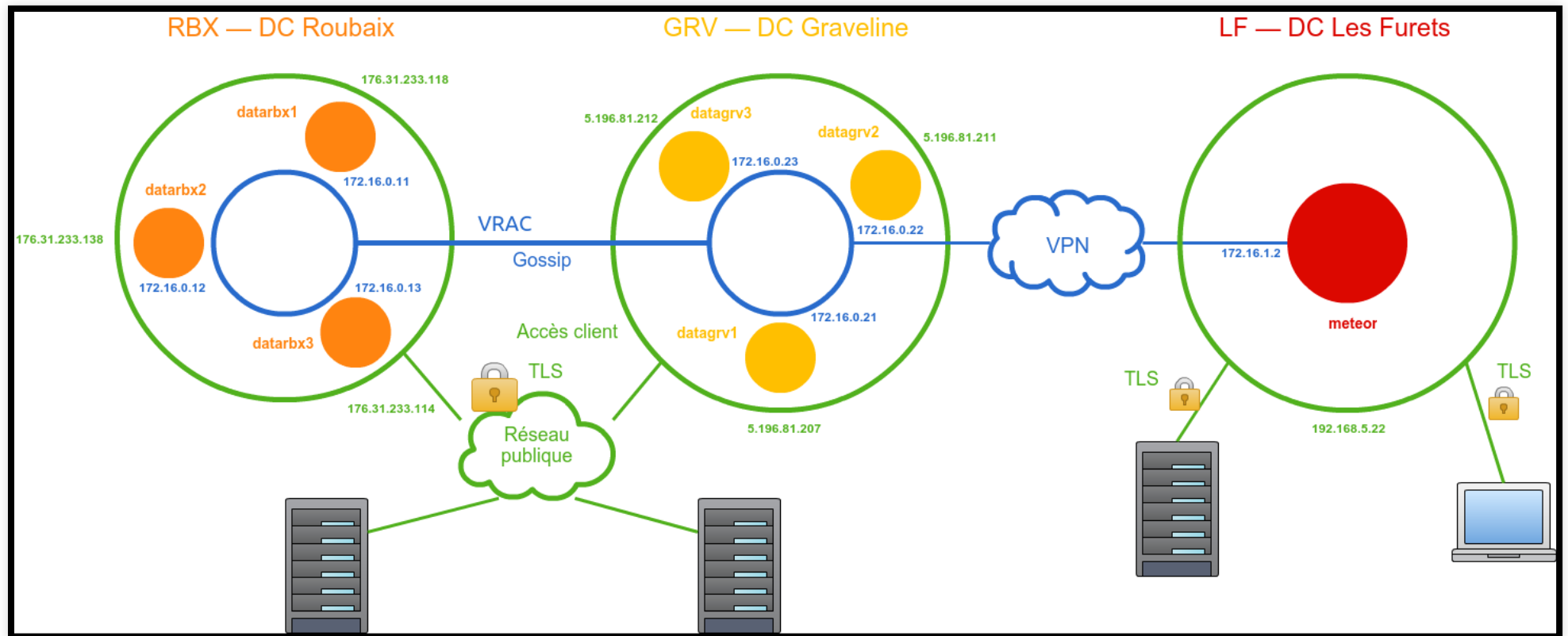
```
CREATE KEYSPACE lesfurets  
WITH replication =  
  {'class': 'NetworkTopologyStrategy', 'RBX': 2, 'GRV':2, 'LF':1};
```

SimpleReplicationStrategy



NetworkTopologyStrategy

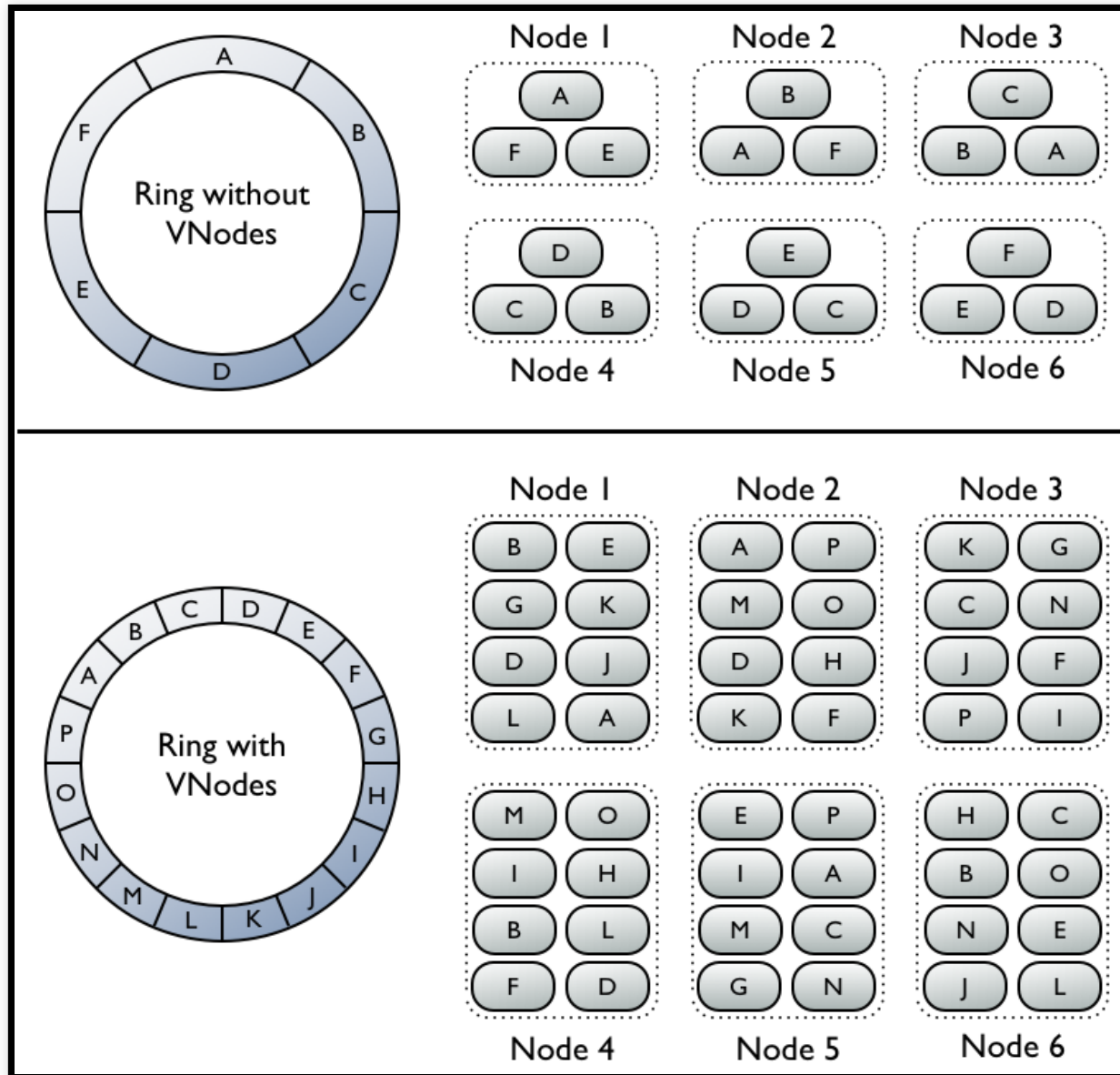
```
CREATE KEYSPACE lesfurets  
WITH replication =  
{'class': 'NetworkTopologyStrategy', 'RBX': 2, 'GRV': 2, 'LF': 1};
```



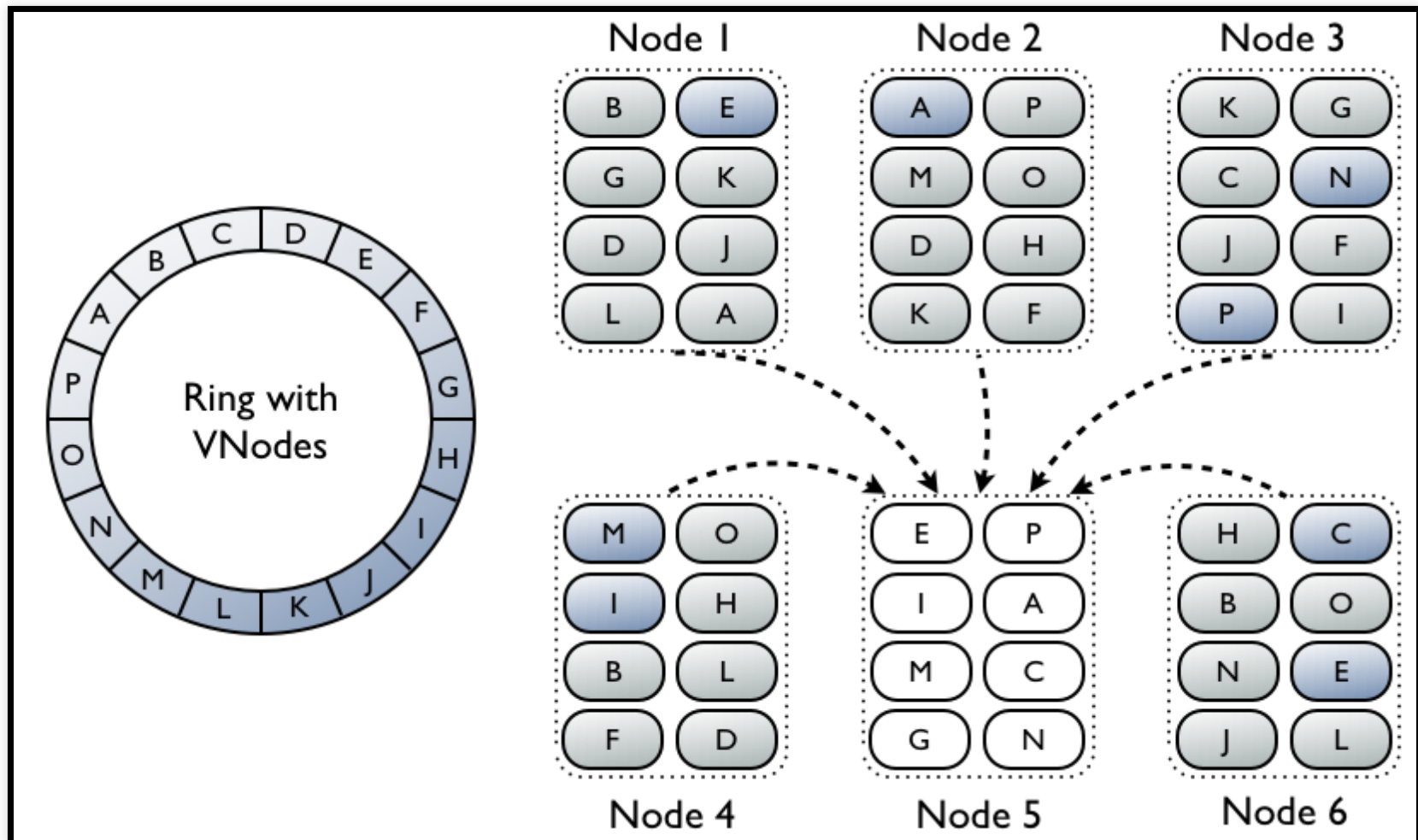
Token allocation

- static allocation (initial-token="-29334..." dans cassandra.yaml)
 - need to be modified at each topology change
- VNODES (num_tokens)
 - random slot allocation (< 3.0)
 - smart (3.0+)

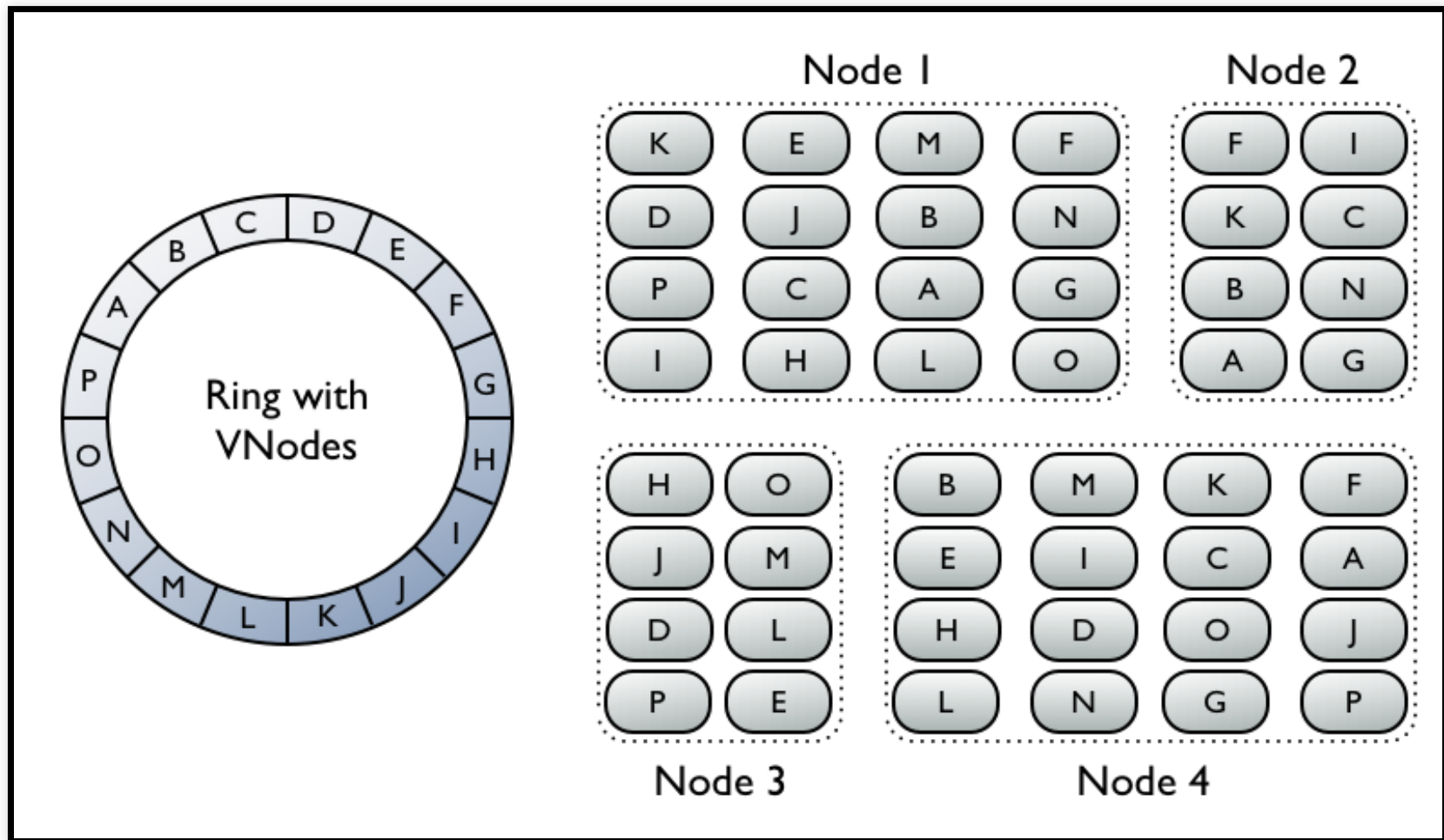
Virtual nodes (VNodes)



VNodes: remapping keys



VNodes: heterogeneous clusters



Plan

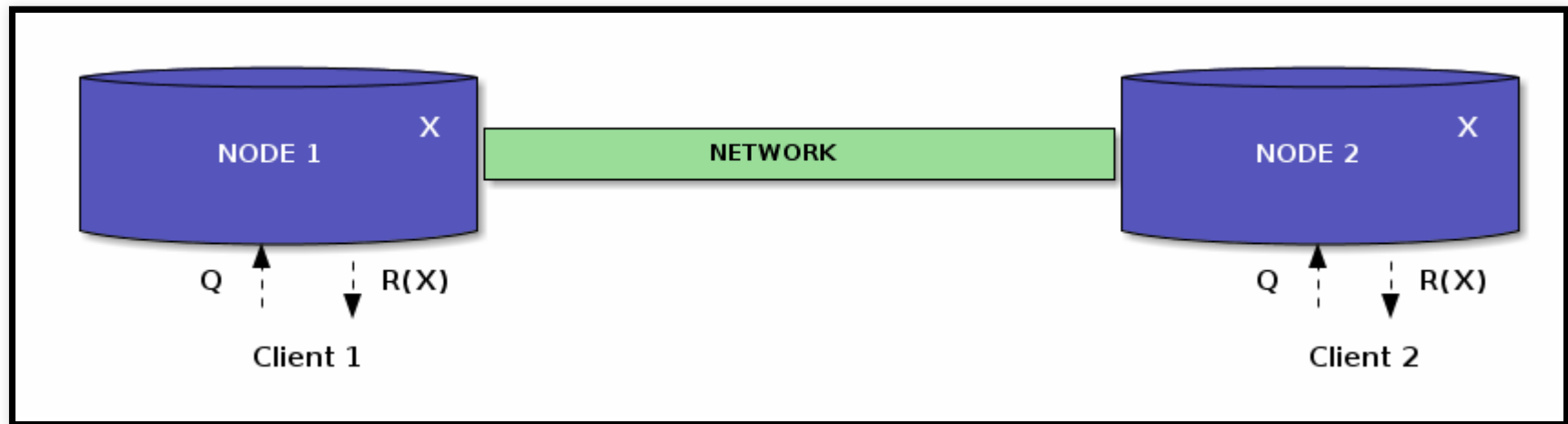
- Motivation
- Apache Cassandra
 - Partitioning and replication
 - **Consistency**
- Practice: Tune consistency in Apache Cassandra

Properties of distributed systems

- **Consistency:** read is guaranteed to return the most recent write for a given client.
- **Availability:** non-failing node will return a reasonable response within a reasonable amount of time (no error or timeout)
- **Partition Tolerance:** the system will continue to function when network partitions occur.

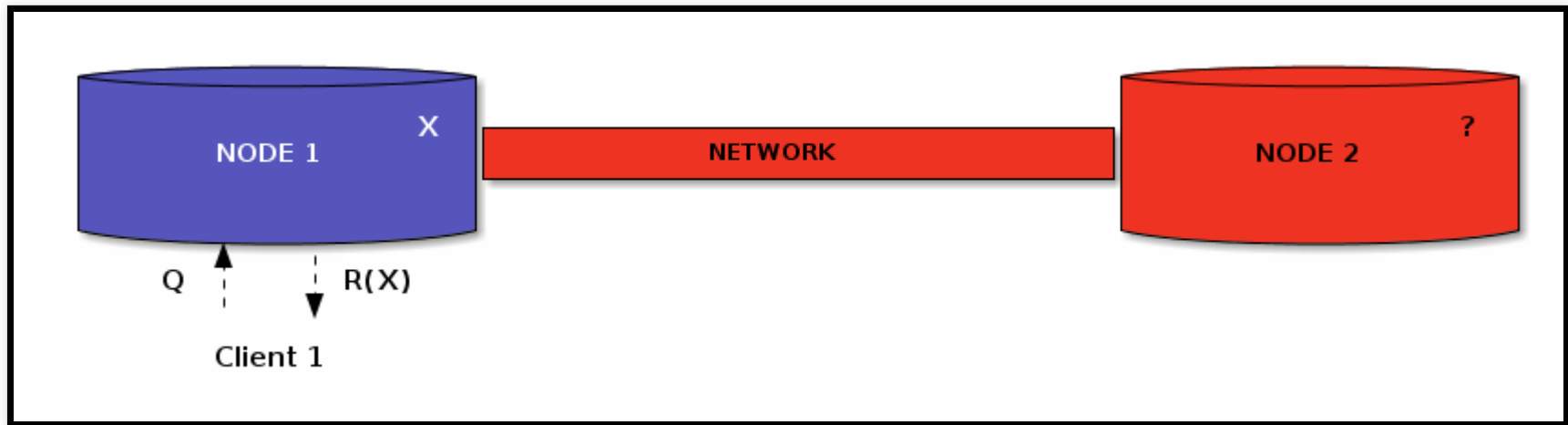
Consistency

- *a read returns the most **recent** write*
- *eventually consistent* : guarantee that the system will evolve in a consistent state
 - provided there are no new updates, all nodes/replicas will eventually return the last updated value (~DNS)



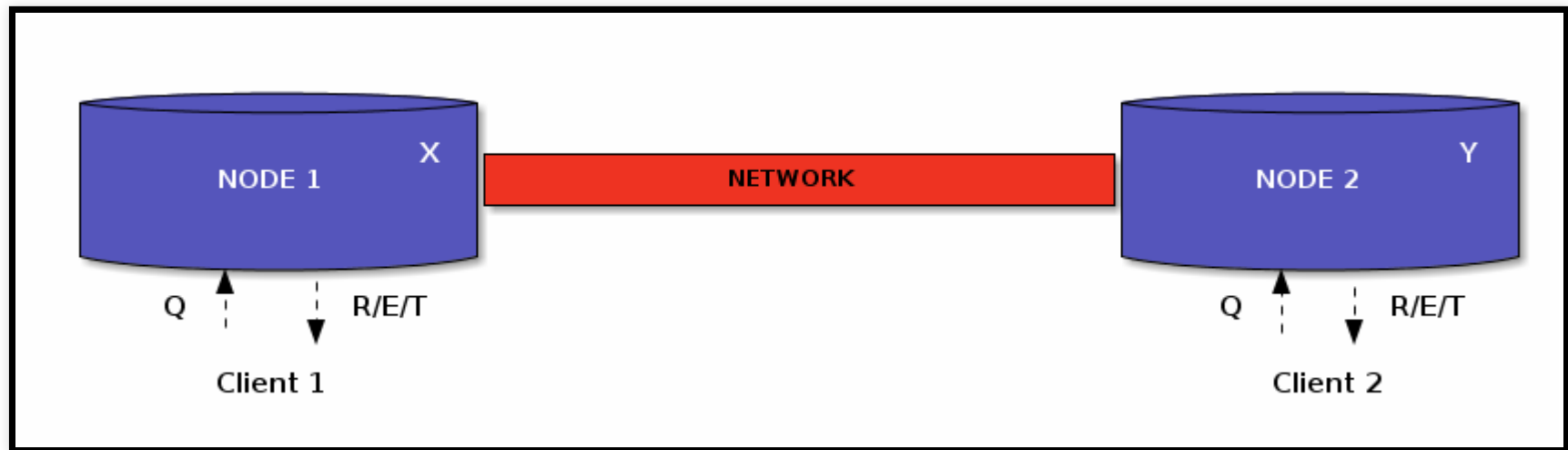
Availability

- *a non-failing node will return a reasonable response (no error or timeout)*



Partition tolerance

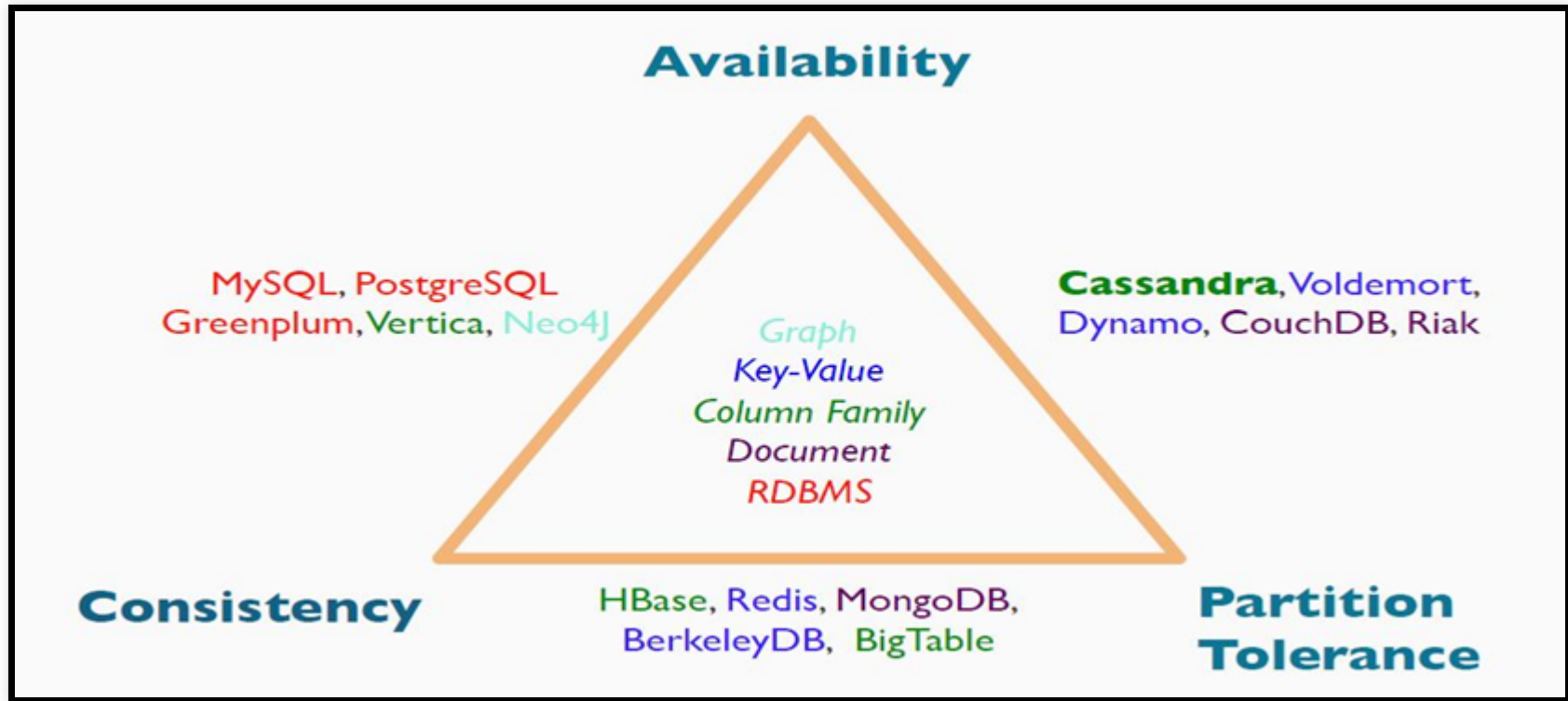
- *ability to **function** (return a response, error, timeout) when network partitions occur*



Partition tolerance

- network is unreliable
- **you can choose** how to handle errors
 - return an old value
 - wait and eventually timeout, or return an error at once
- in practice: choice between AP and CP systems

CAP theorem



Cassandra consistency

- **AP** system
 - *eventually consistent*
 - without updates the system will converge to a consistent state due to *repairs*
 - *tunable consistency* :
 - Users can determine the consistency level by tuning it during **read** and **write** operations.

BASE

- RDBMS: **A**tomic, **C**onsistent, **I**solated, **D**urable
- Cassandra: **B**asically **A**vailable, **S**oft state, **E**ventually consistent

Consistency Level (CL)

- mandatory **protocol-level** parameter for each query (read/write),
- #replicas in a cluster that must acknowledge the **read / write**
 - **write consistency - R**: #replicas on which the write *must succeed** before returning an acknowledgment to the client application.
 - **read consistency - W**: #replicas that must *respond to a read* request before returning data to the client application
- default level: *ONE*
- most used: *ONE, QUORUM, ALL, ANY ... (LOCAL_ONE, LOCAL_QUORUM...)*

Which consistency level ?

W : Niveau de cohérence en écriture (nb de nœuds)

R : Niveau de cohérence en lecture (nb de nœuds)

RF : Facteur de réplication

Cohérence immédiate si :

$$W + R > RF$$

W : ONE

R : ONE

Cohérence : à terme

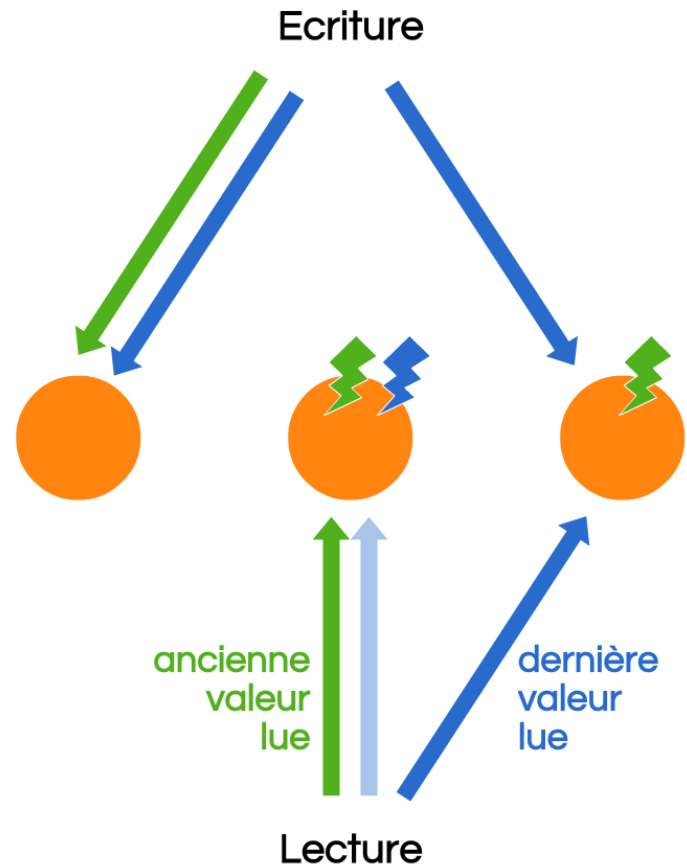
Disponibilité : 2 nœuds en panne

W : QUORUM

R : QUORUM

Cohérence : immédiate

Disponibilité : 1 nœud en panne



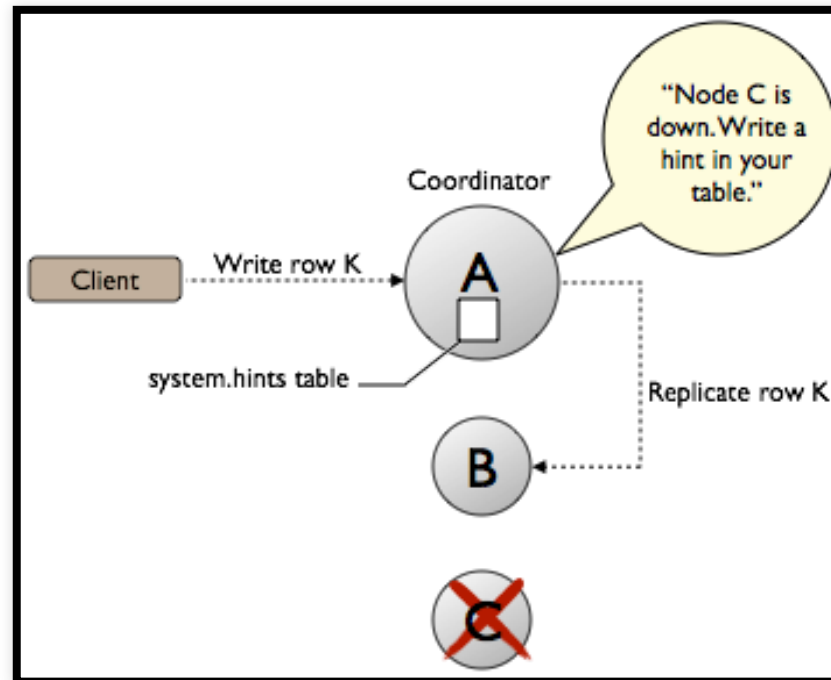
⚡ nœud en panne pendant l'écriture

Consistency mechanisms

- *writes* \Rightarrow *hinted handoff*
- *reads* \Rightarrow *read repairs*
- *maintenance* \Rightarrow *anti-entropy repair* (*nodetool repair*)

Hinted handoff

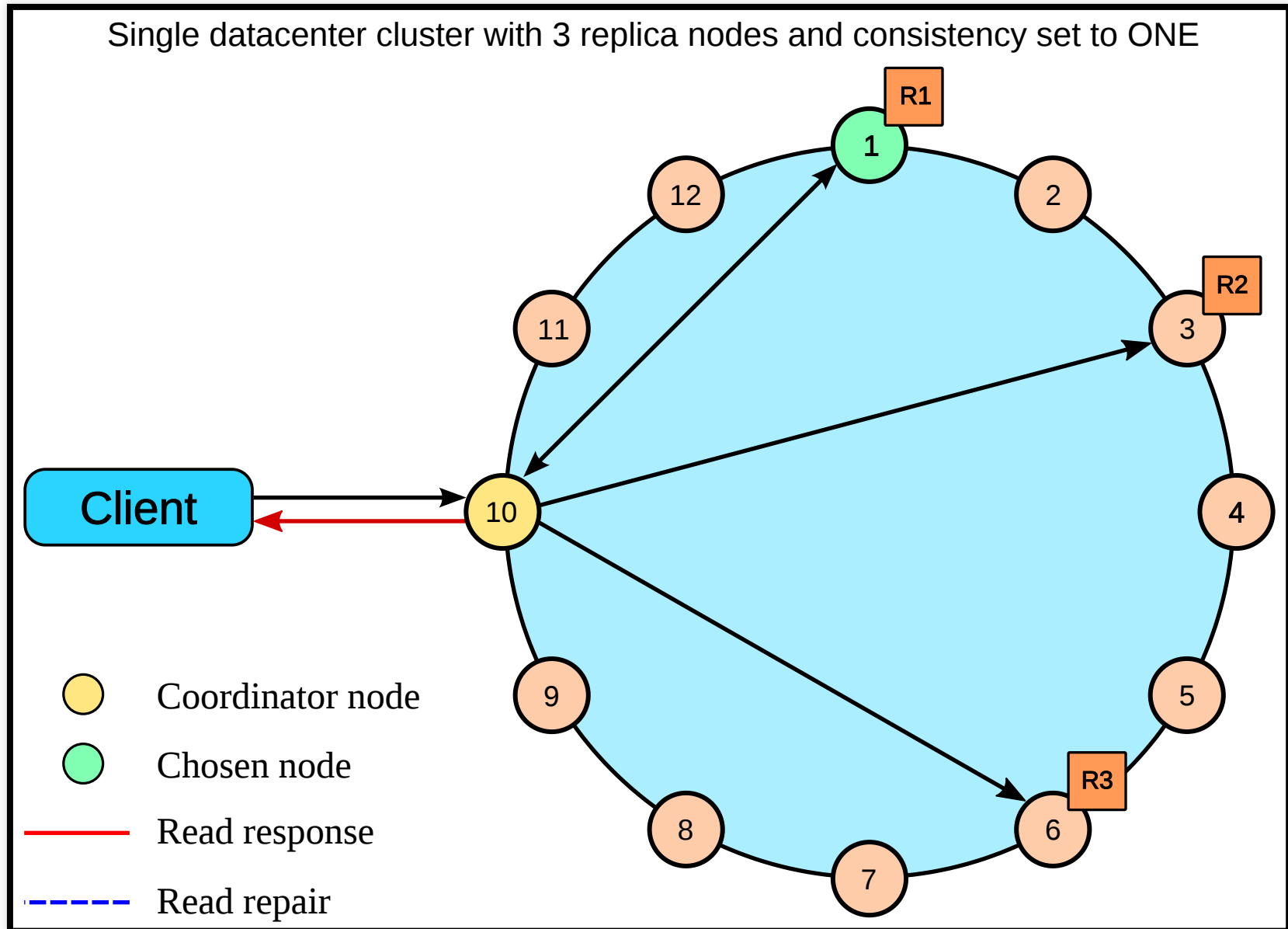
- ONE/QUORUM vs ANY (any node may ACK even if not a replica)
- if one/more replica(s) are down \Rightarrow **hinted handoff**



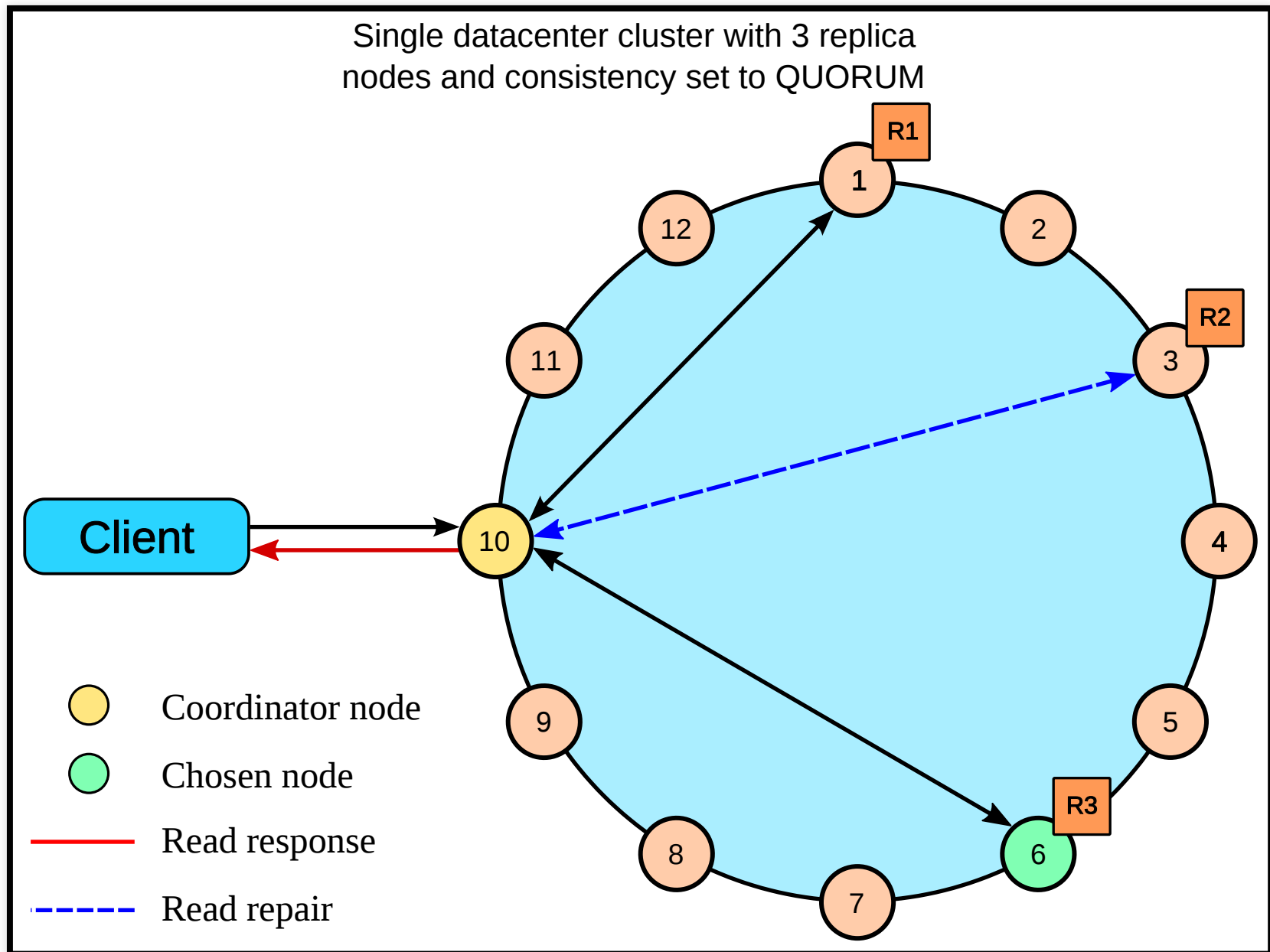
Read repairs

- Goal: detect and fix inconsistencies during reads
- $CL = ONE \Rightarrow$ no data is repaired because no comparison takes place (unless `read_repair_chance > 0`)
- $CL > ONE \Rightarrow$ repair participating replica nodes in the foreground before the data is returned to the client.
 - C^* sends a digest request to each replica not directly involved in the read
 - Cassandra compares all replicas and writes the most recent version to any replica node that does not have it.

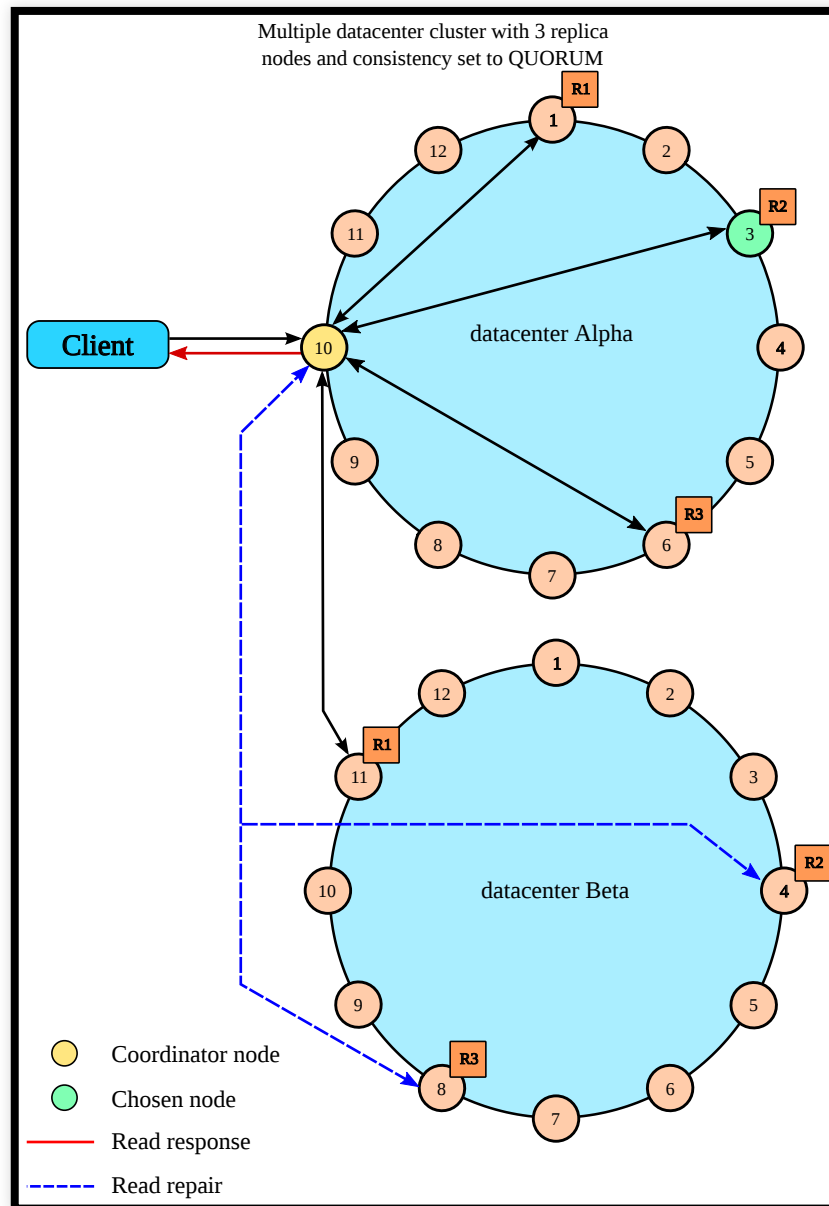
Read repairs ONE



Read repairs QUORUM



Read repairs QUORUM DC



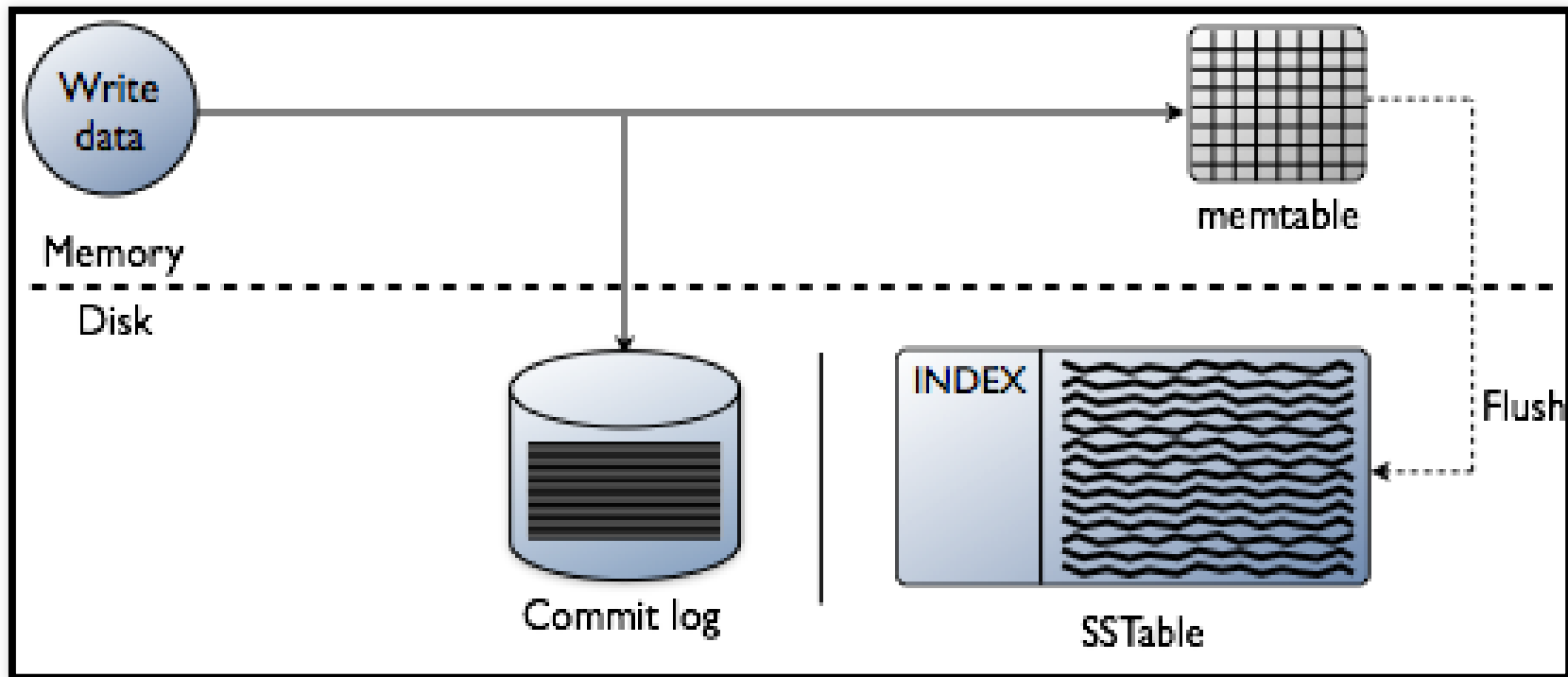
Anti-entropy repair

- for each token range, read and synchronize the rows
- to insure the consistency this tool must be run regularly
!
- not automatised (only on Datastax...)

Durability

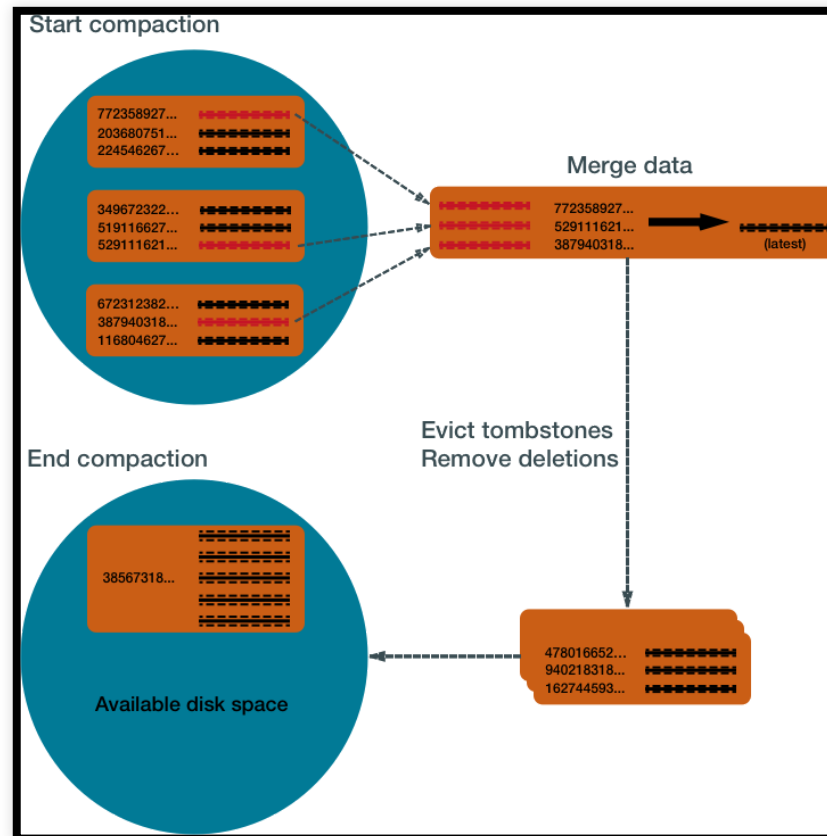
- guarantees that writes, once completed, will survive permanently
 - appending writes to a commitlog first
 - default: flushed to disk every *commitlog_sync_period_in_ms*
 - batch mode \Rightarrow sync before ACK the write

Cassandra write path

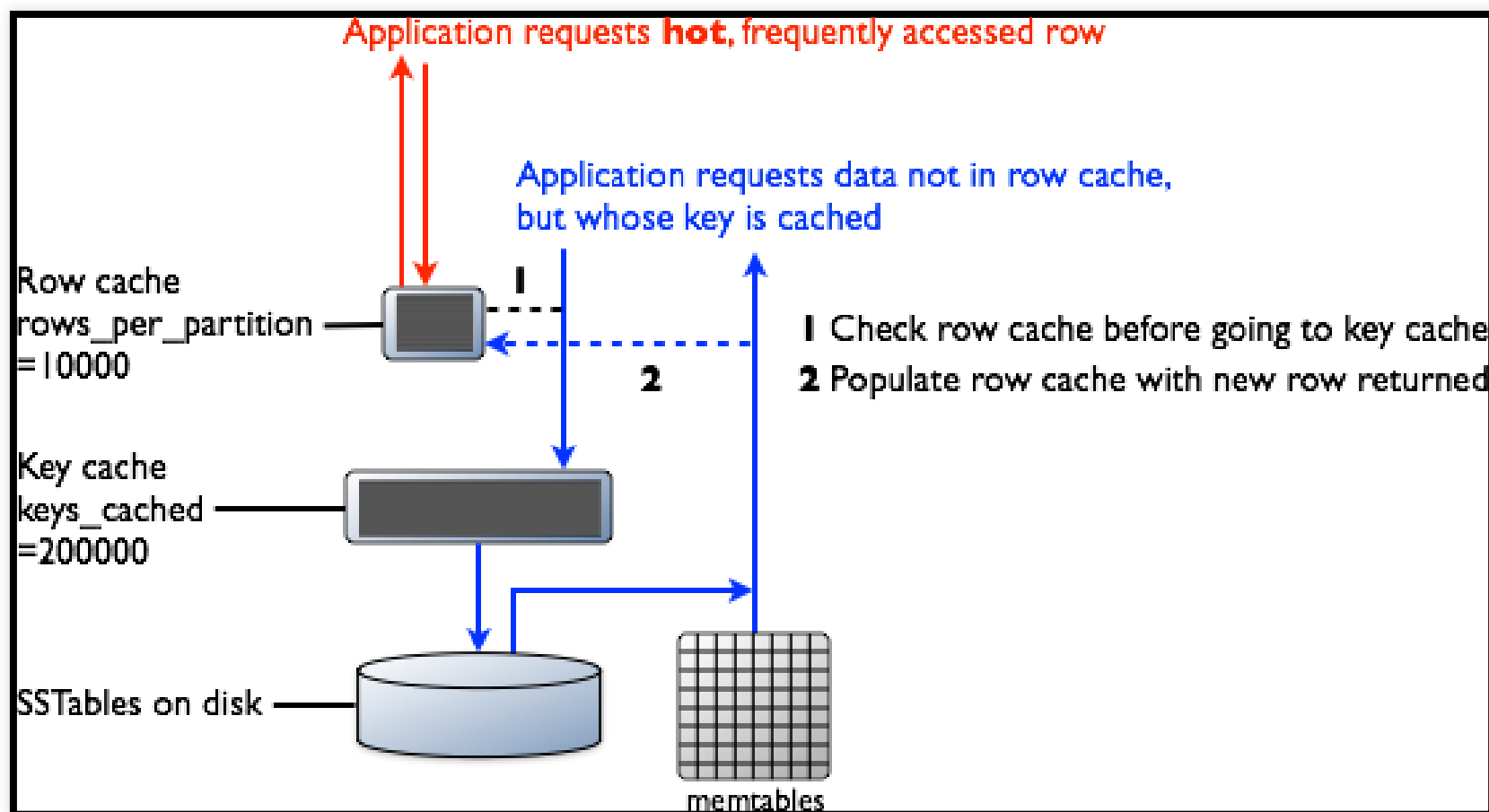


Cassandra compactions

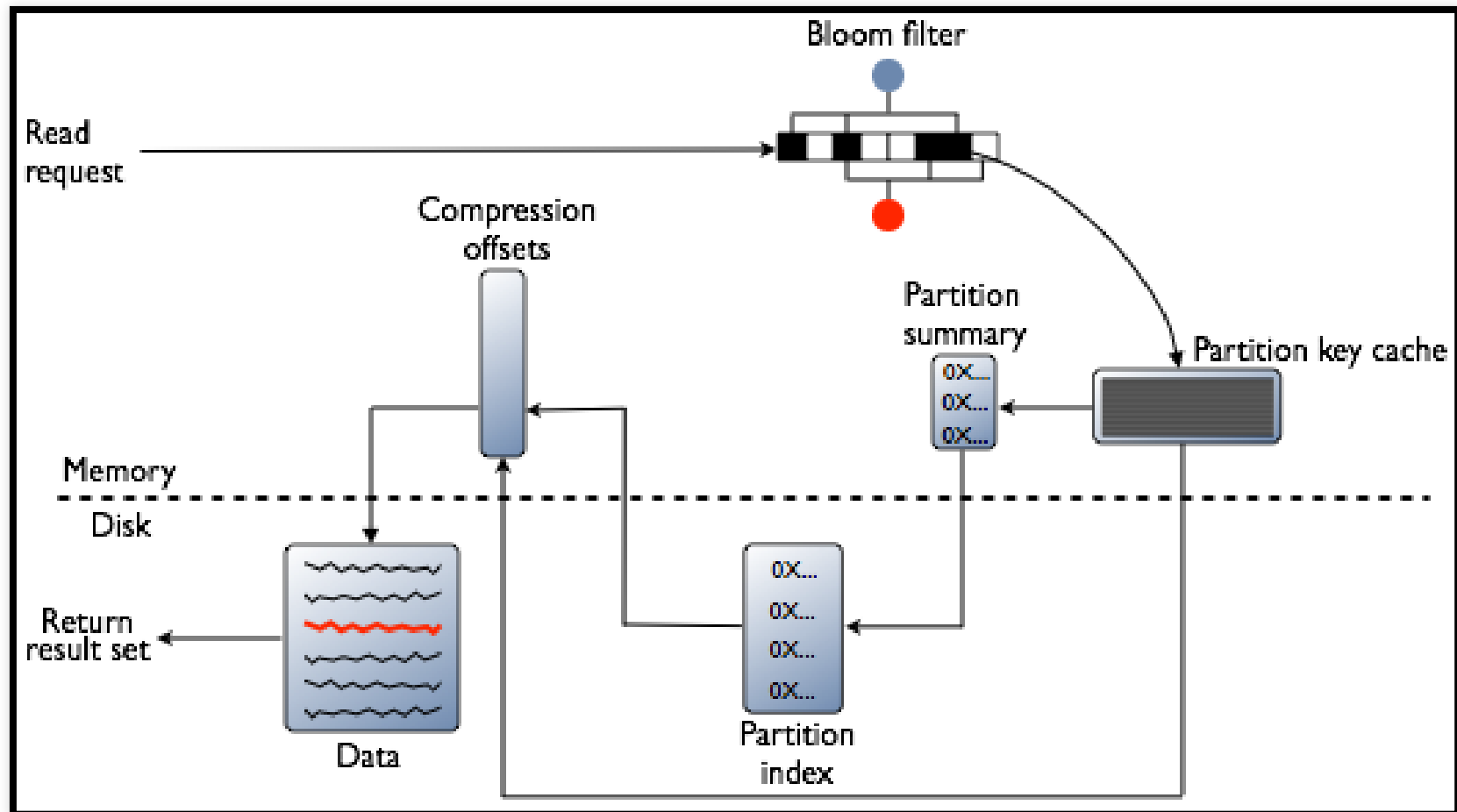
- collects all versions of each unique row
- assembles one complete row (up-to-date)



Cassandra read path (caches)



Cassandra read path (disk)



More info

- [Cassandra database internals documentation](#)

Plan

- Motivation
- Apache Cassandra
 - Partitioning and replication
 - Consistency
- **Practice: Tune consistency in Apache Cassandra**

Practice: Tune consistency in Apache Cassandra

1. create local test clusters
2. explore configuration options and consistency properties

Cassandra cluster manager

- create multi-node cassandra clusters on the local machine
- great for quickly setting up clusters for development and testing

```
$ccm create test -v 2.0.5 -n 3 -s (1)
```

```
$ccm node1 stop (2)
```

```
$ccm node1 cqlsh (3)
```

Nodetool

- a command line interface for managing a cluster
 - explore, debug, performance test
 - maintenance operations, repairs

```
$ccm node1 nodetool status mykeyspace (1)
```

```
Datacenter: datacenter1
```

```
=====
```

```
Status=Up/Down
```

```
|/ State=Normal/Leaving/Joining/Moving
```

--	Address	Load	Tokens	Owns	Host ID
UN	127.0.0.1	47.66 KB	1	33.3%	aaa1b7c1-6049-4a08-ad3e-3697a0e30e10
UN	127.0.0.2	47.67 KB	1	33.3%	1848c369-4306-4874-afdf-5c1e95b87320
UN	127.0.0.3	47.67 KB	1	33.3%	49578bf1-728f-438d-b1c1-d8dd644b6f71

CQLSh

- standard CQL client

```
[bigdata@bigdata ~]$ ccm node2 cqlsh (1)
Connected to test at 127.0.0.2:9160.
[cqlsh 4.1.1 | Cassandra 2.0.5-SNAPSHOT | CQL spec 3.1.1 | Thrift protocol 19
Use HELP for help.
cqlsh> SELECT * FROM system.schema_keyspaces ; (2)
```

VSCODE plugin

The image shows the VS Code plugin interface for database management. The top section is the CQL script editor, and the bottom section is the data table view.

Top Section (CQL Script Editor):

- Unsaved indicator:** A red dot in the tab bar.
- Select cluster:** A dropdown menu showing "US/New York".
- Select keyspace:** A dropdown menu showing "example_keyspace".
- Save script:** A button labeled "SAVE".
- Execute script:** A button labeled "EXECUTE".
- Code editor:** The area containing the CQL script:


```
1 SELECT pk1, pk2, cck1, cck2, c_ascii, c_bigint, c_blob, c_boolean, c_date,  
2 c_decimal, c_double, c_float, c_int  
3 FROM example_keyspace.table_simple  
4 WHERE pk1='abc' AND pk2='abc'  
5 LIMIT 500;
```
- Column clustering key decoration:** A pink line pointing to the `cck1` and `cck2` columns in the `SELECT` statement.
- Partition key decoration:** A pink line pointing to the `pk1` and `pk2` columns in the `WHERE` clause.
- New tab, open CQL script, show history:** A pink line pointing to the tab bar icons.

Bottom Section (Data Table View):

- Partition key column:** A pink box around the `pk1` and `pk2` columns.
- Column clustering key column:** A pink box around the `cck1` and `cck2` columns.
- Cell editor:** A pink box around the `c_bigint` cell in the second row, which is currently being edited.
- Uncommitted change:** A pink line pointing to the `c_bigint` cell in the second row, indicating an uncommitted change.


Uncommitted changes dialog: A dialog box at the bottom showing "1 change pending" and buttons for "COMMIT" and "DISCARD".


Datastax Studio

 DataStax Studio

★ Working with CQL v6.0.0

default localhost








 SCHEMA



CQL Keyspace: not selected

//Table results, containing only basic types(nested types coming up a few cells below!)

SELECT * FROM videodb.video_event;

index ↑	videoid	username	event_timestamp	event	video_timestamp
0	"99051fe9-6a9c-46c2-b949-38ef78858dd0"	tcodd	"6be8f931-ca93-11e7-9ce8-e3d8f7e5f93a"	stop	230000
1	"99051fe9-6a9c-46c2-b949-38ef78858dd0"	tcodd	"6be8ab11-0a93-11e7-9ce8-e3d8f7e5f93a"	start	3000
2	"99051fe9-6a9c-46c2-b949-38ef78858dd0"	tcodd	"6be835e1-0a93-11e7-9ce8-e3d8f7e5f93a"	stop	30000
3	"99051fe9-6a9c-46c2-b949-38ef78858dd0"	tcodd	"6be7c0b0-0a93-11e7-9ce8-e3d8f7e5f93a"	start	0
4	"99051fe9-6a9c-46c2-b949-38ef78858dd0"	tcodd	"66d33161-0906-11e7-97e0-b3490a02b33d"	stop	230000
5	"99051fe9-6a9c-46c2-b949-38ef78858dd0"	tcodd	"66d2e341-0906-11e7-97e0-b3490a02b33d"	start	3000

Displaying 1 - 16 of 16 results for the last statement

< 1 >

Success. 16 elements returned. Duration: 0.012 s.

Ressources:

Datastax documentation

<https://dzone.com/articles/introduction-apache-cassandras>

<https://highlyscalable.wordpress.com/2012/09/18/distributed-algorithms-in-nosql-databases/>