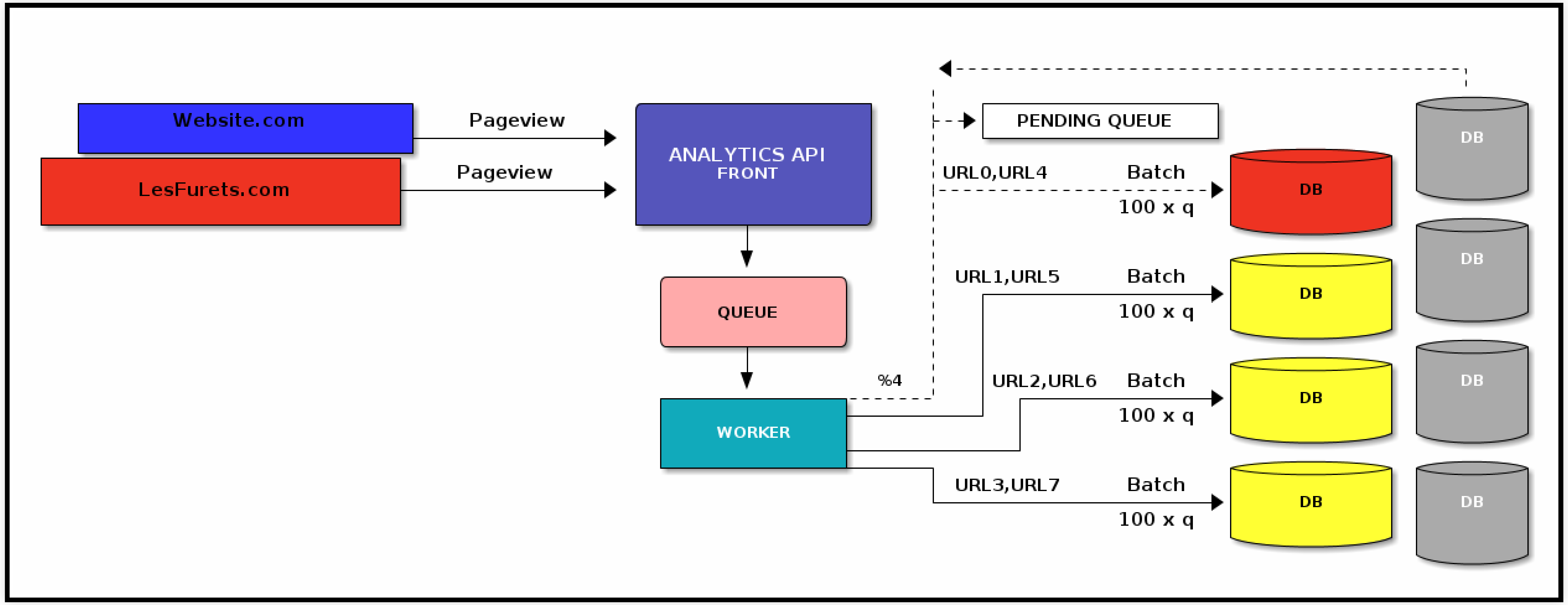
Introduction to Apache Cassandra

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/writting the data?

What went wrong?

- **A single server cannot take the load ⇒** solution / **complexity**
  - *Better database*
    - easy to add/remove nodes (**scalling**)
    - 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

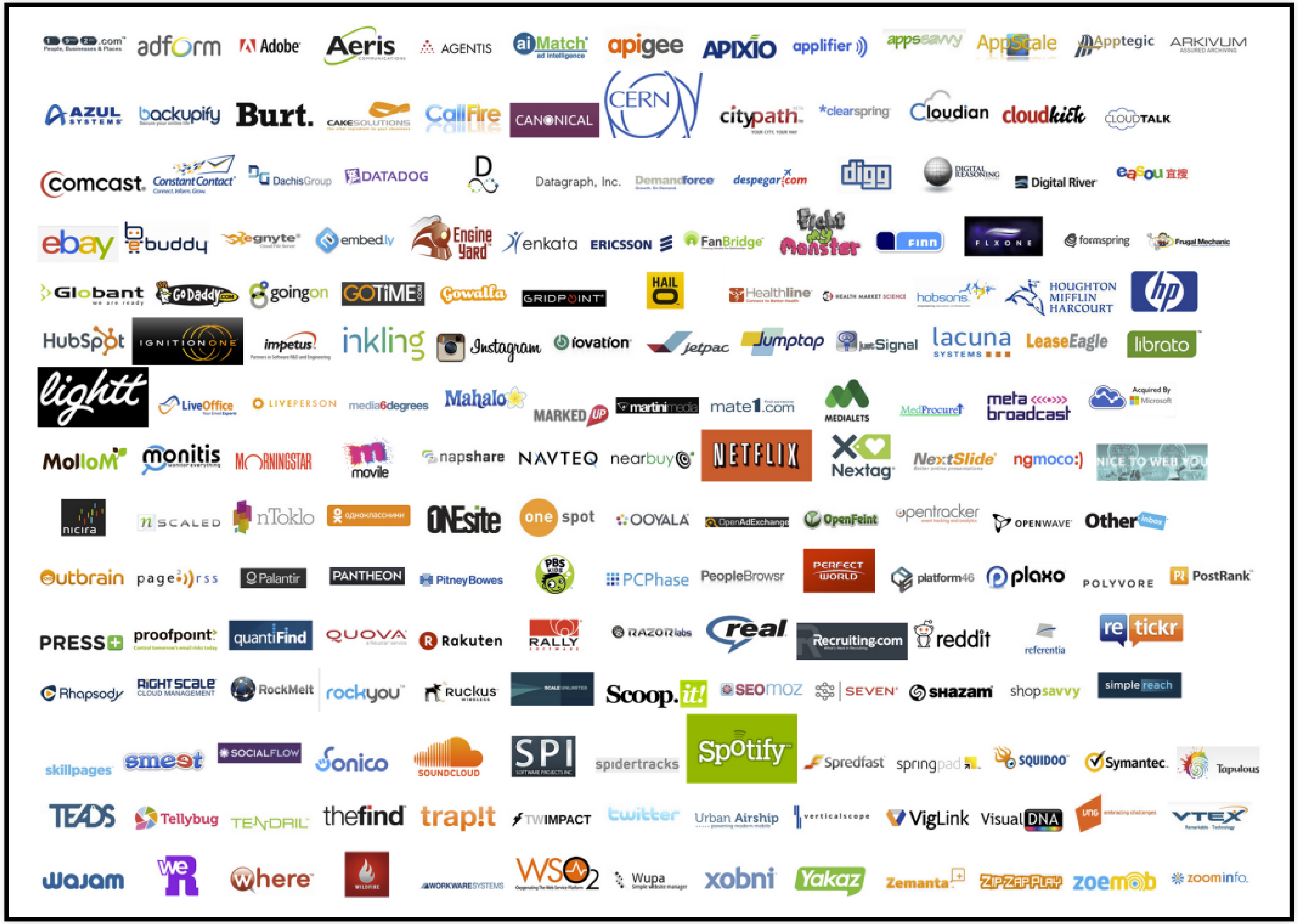
Apache Cassandra

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 ⇒ coordinator to dispatch and gather the results
- reliable and simple scaling
- online load balancing and cluster growth

Apache Cassandra users

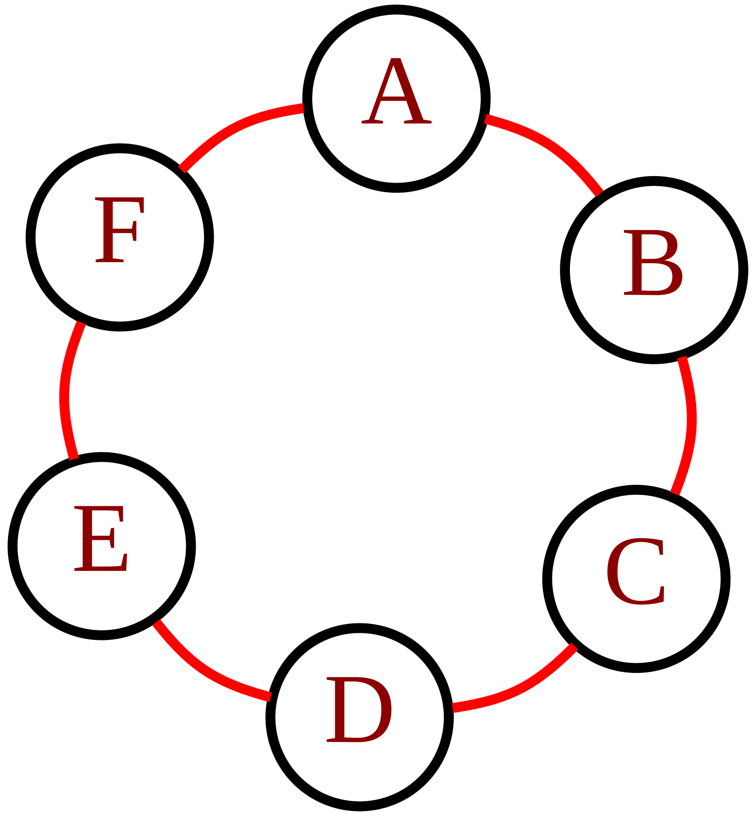


source: https://codingjam.it/

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 groupped in one or more datacenters (can span physical locations)*

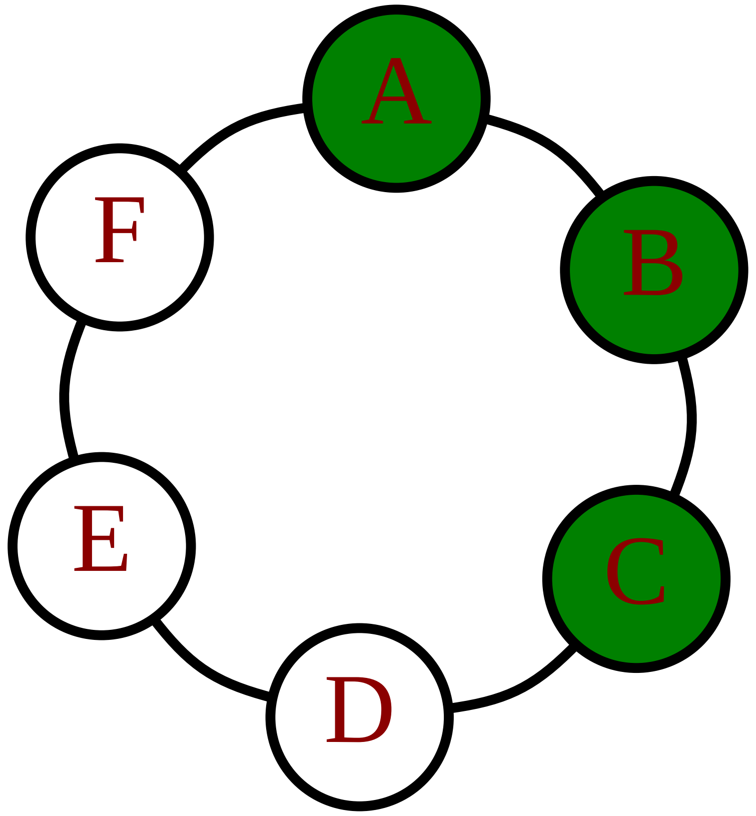
Cassandra ring



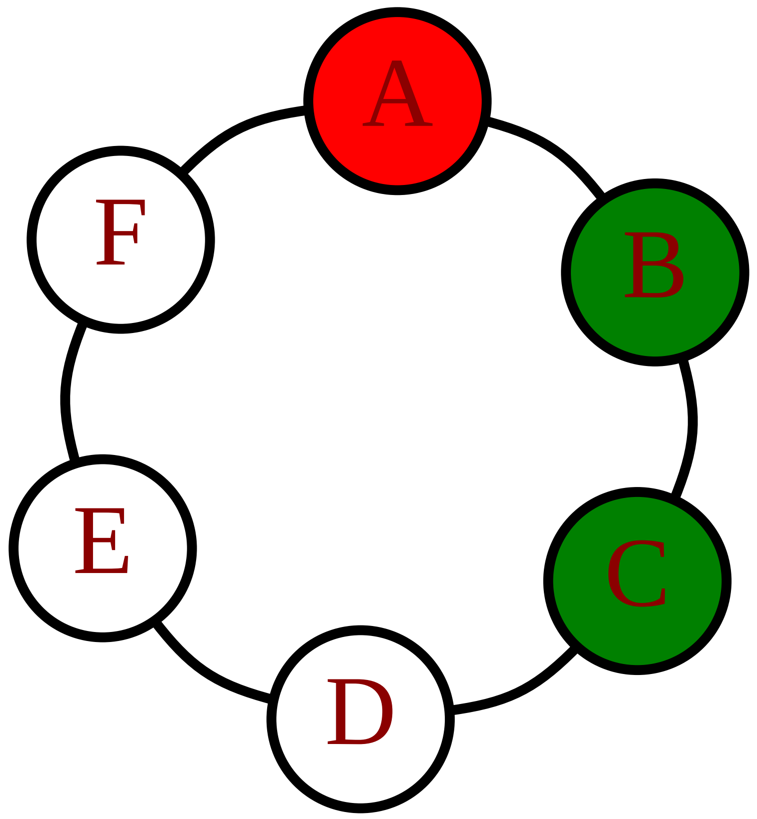
Gossip

- 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** ⇒ bootstrapping the gossip process for new nodes joining the cluster

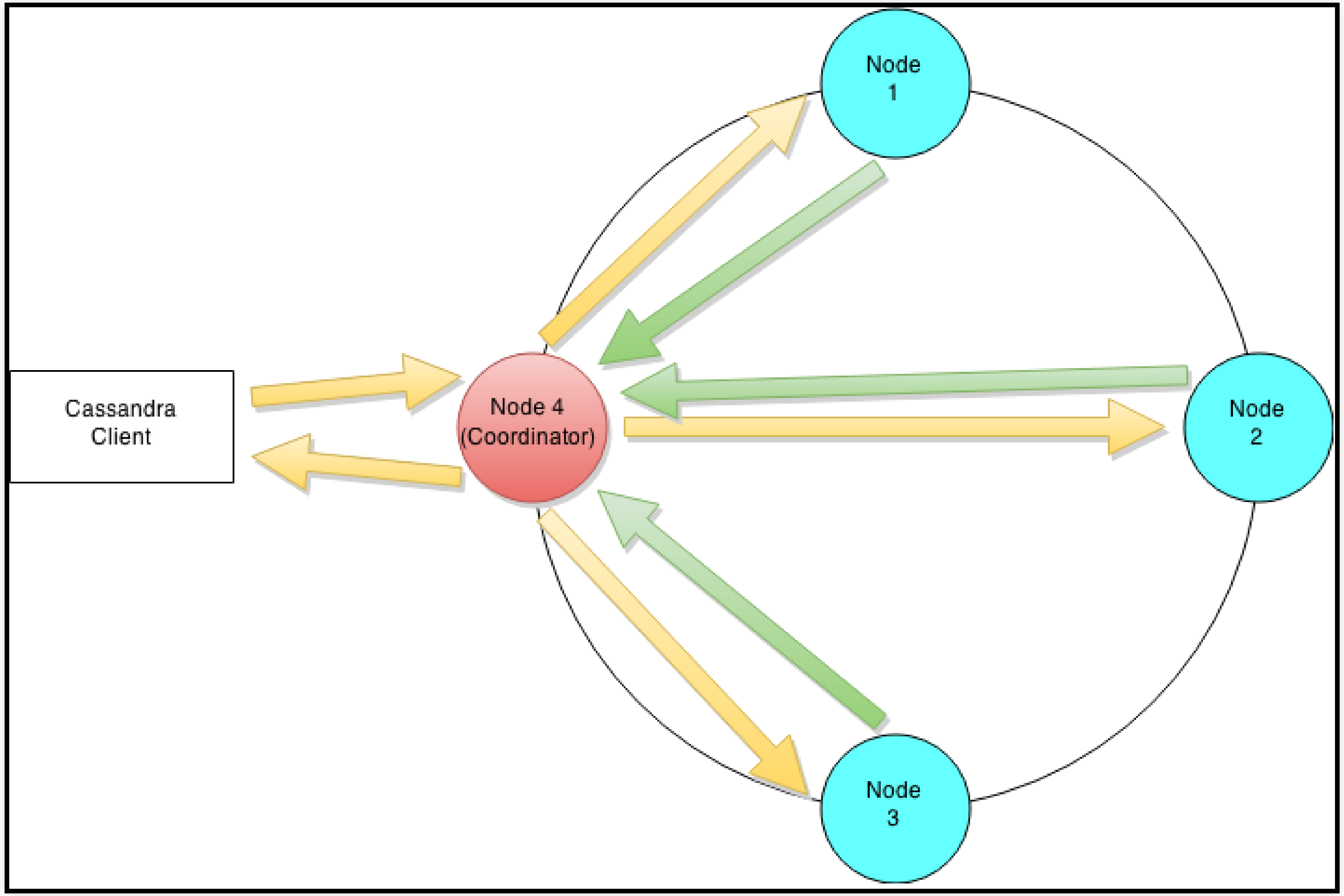
Cassandra replicas



Cassandra node failure



Query

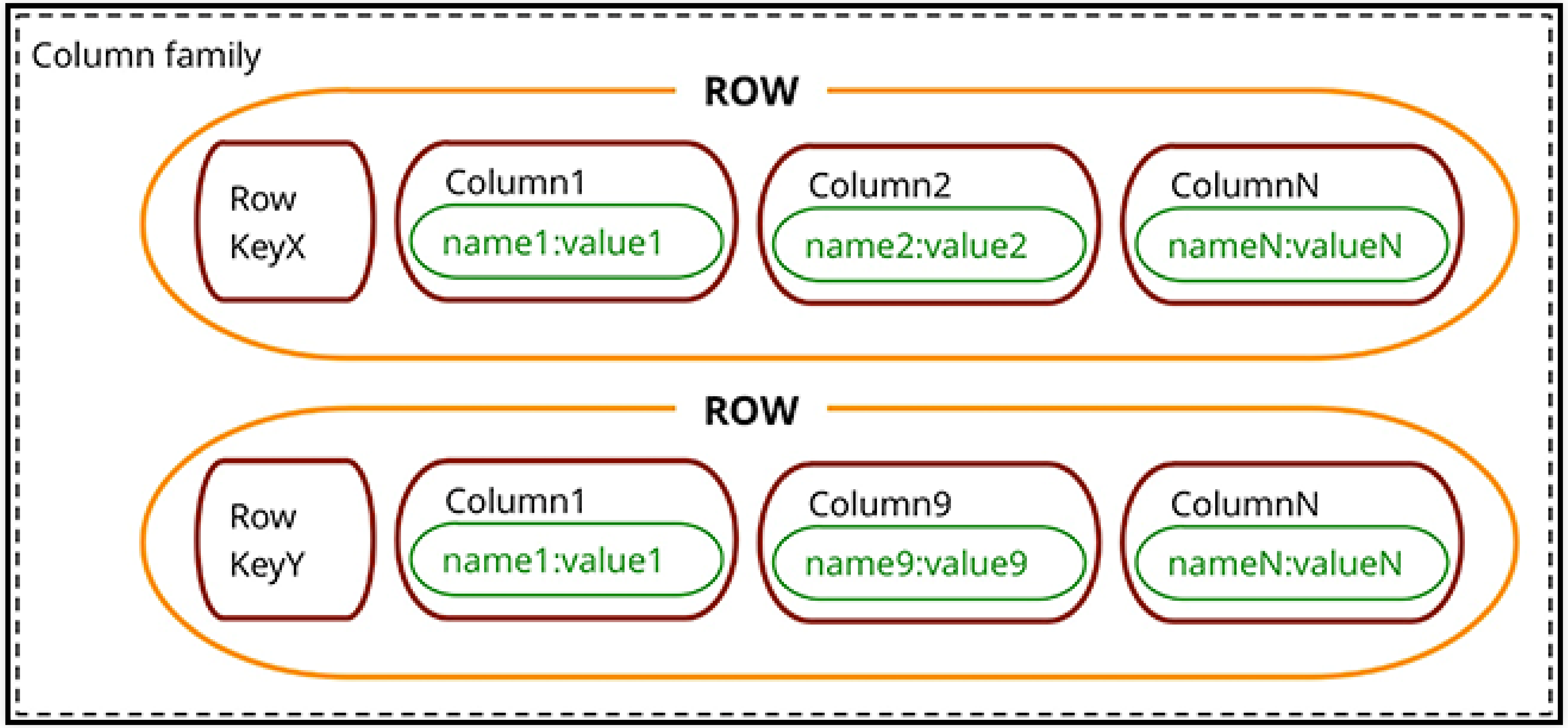


Plan

- 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?***
    - 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)* ⇒ remap a large number of keys when nodes join/leave the cluster
- ***Consistent hashing***: only k/n keys need to be remapped on average

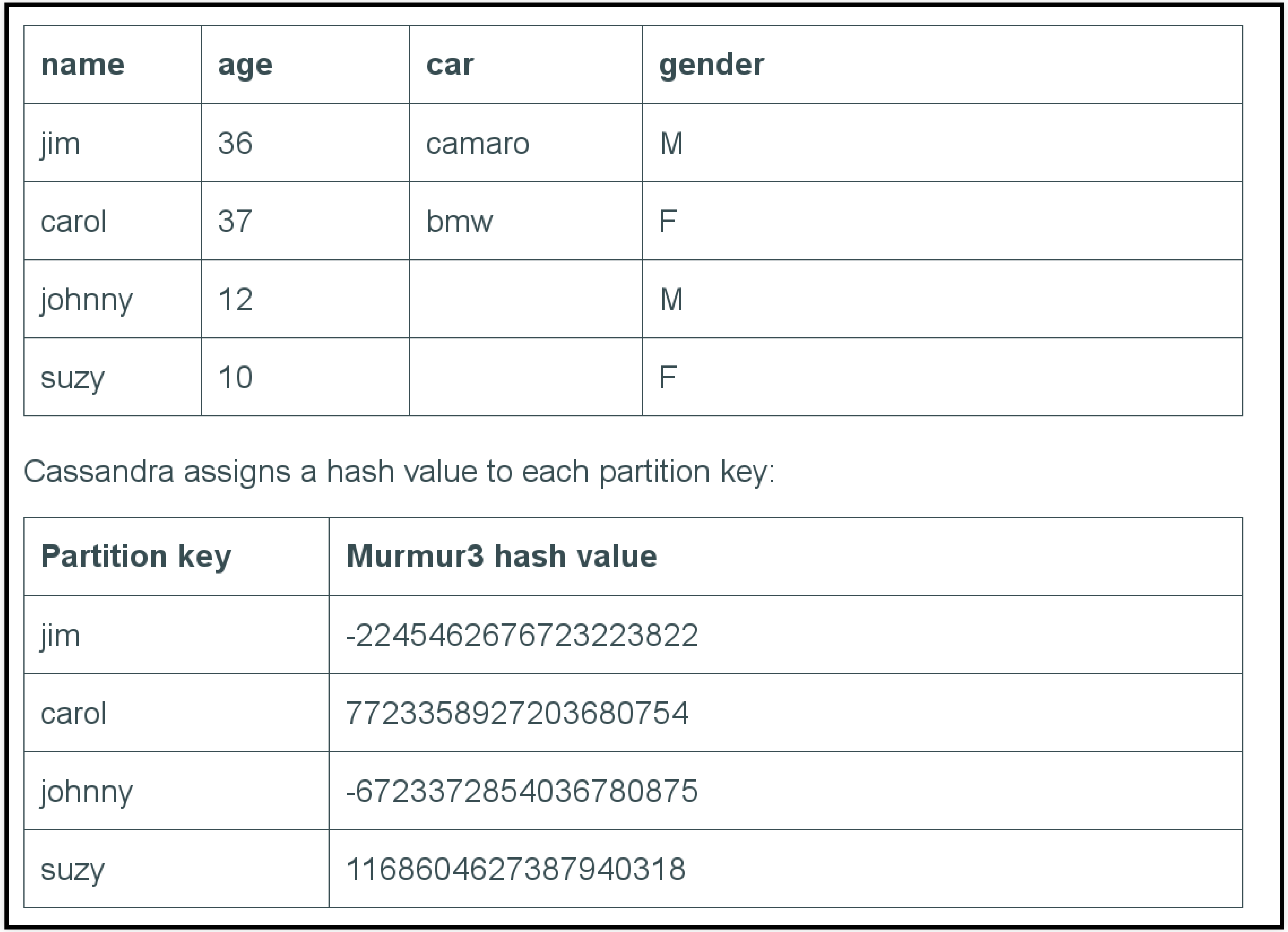
Consistent Hashing

- **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



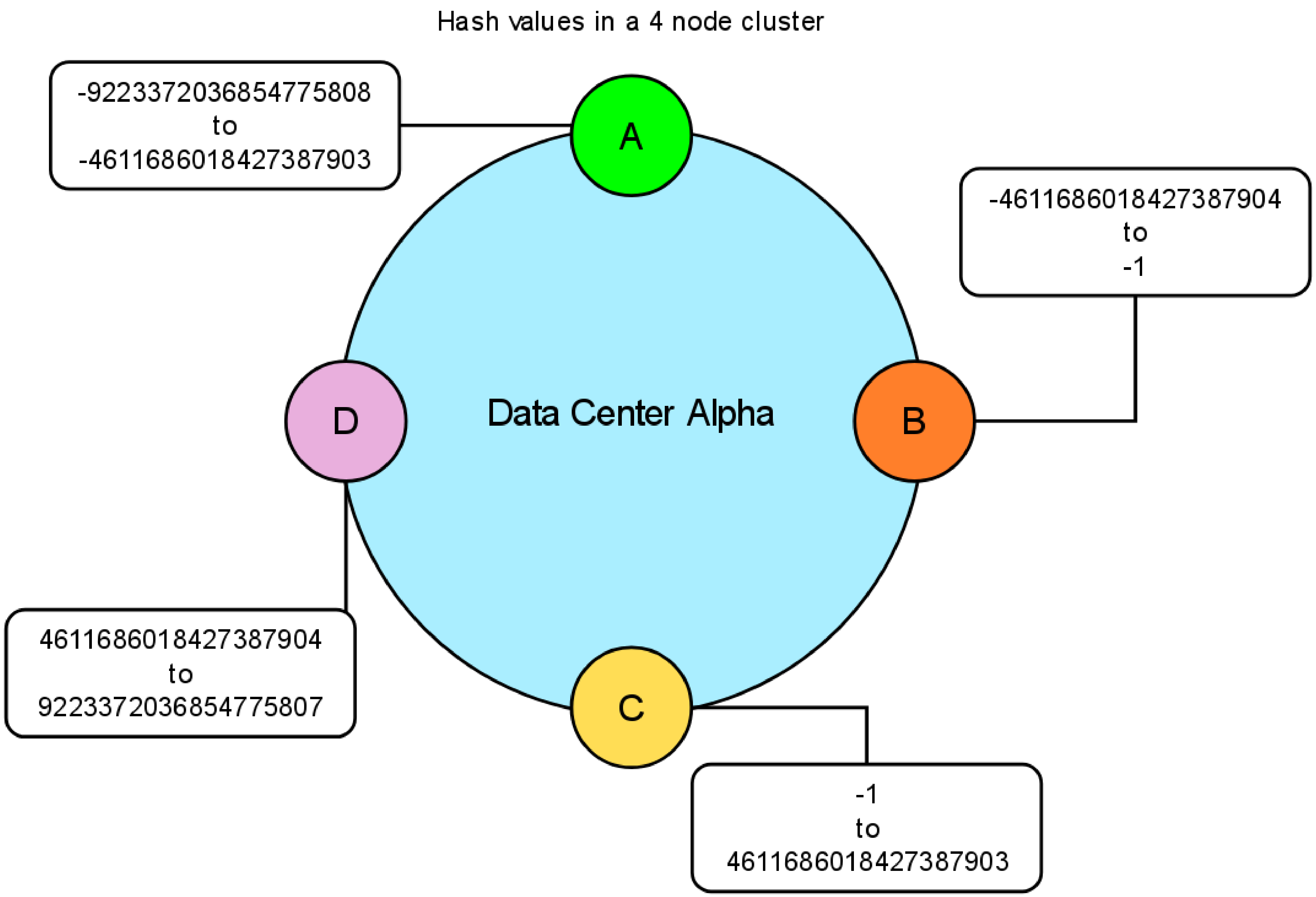
name	age	car	gender
jim	36	camaro	M
carol	37	bmw	F
johnny	12		M
suzy	10		F

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

Data Center Alpha

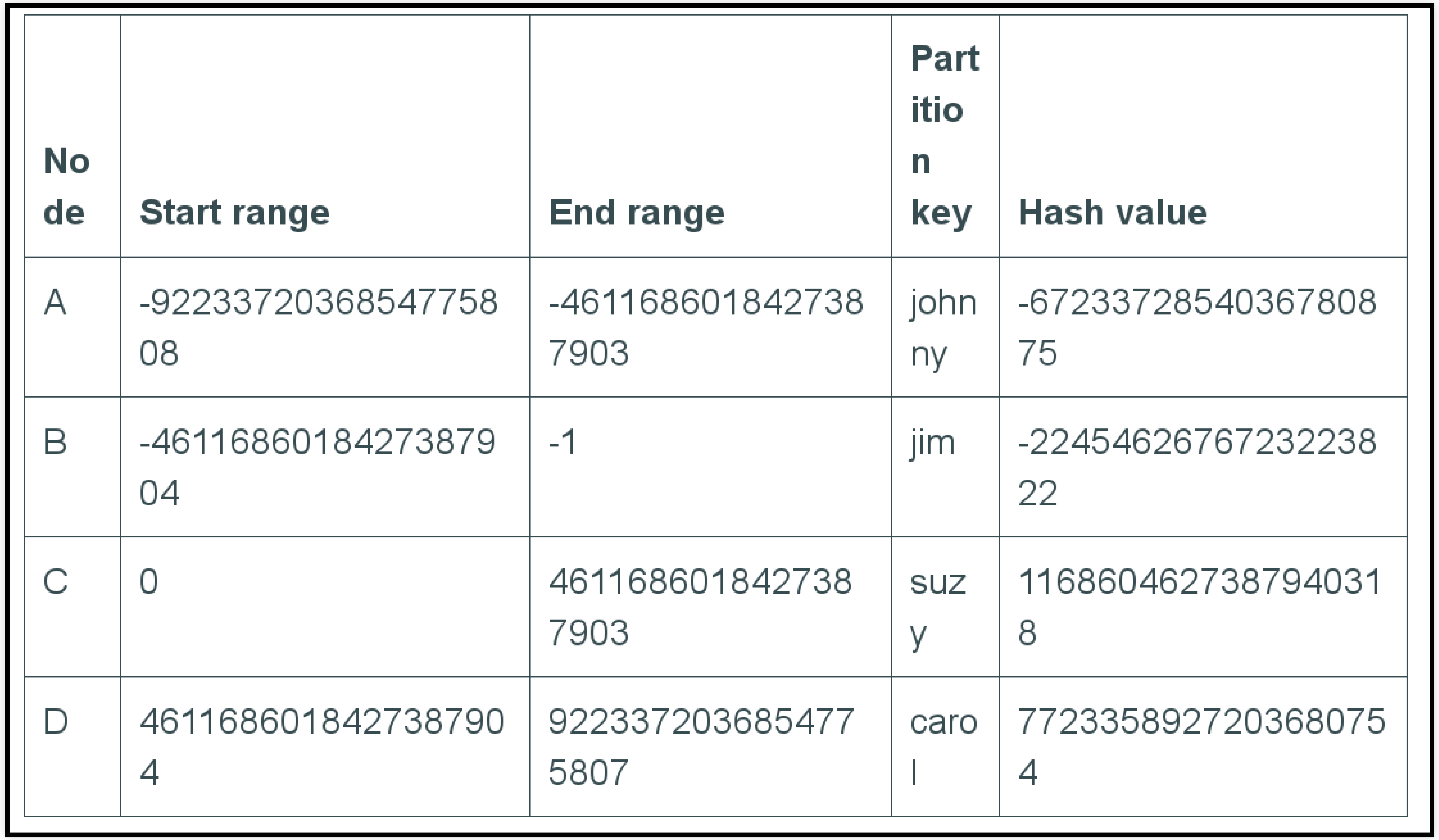
A
-9223372036854775808
to
-4611686018427387903

B
-4611686018427387904
to
-1

C
-1
to
4611686018427387903

D
4611686018427387904
to
9223372036854775807

Consistent Hashing: mapping



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

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***
  - ⇒ requests are routed efficiently
  - ⇒ support replication by groupping nodes (racks/datacenters) and avoid **correlated failures**
- ***SimpleSnitch*** ⇒ does not recognize datacenter or rack information
- ***RackInferringSnitch*** ⇒ infers racks and DC information
- ***PropertyFileSnitch*** ⇒ uses pre-configured rack/DC informations
- ***DynamicSnitch*** ⇒ monitor read latencies to avoid reading from hosts that have slowed down

Replication strategies

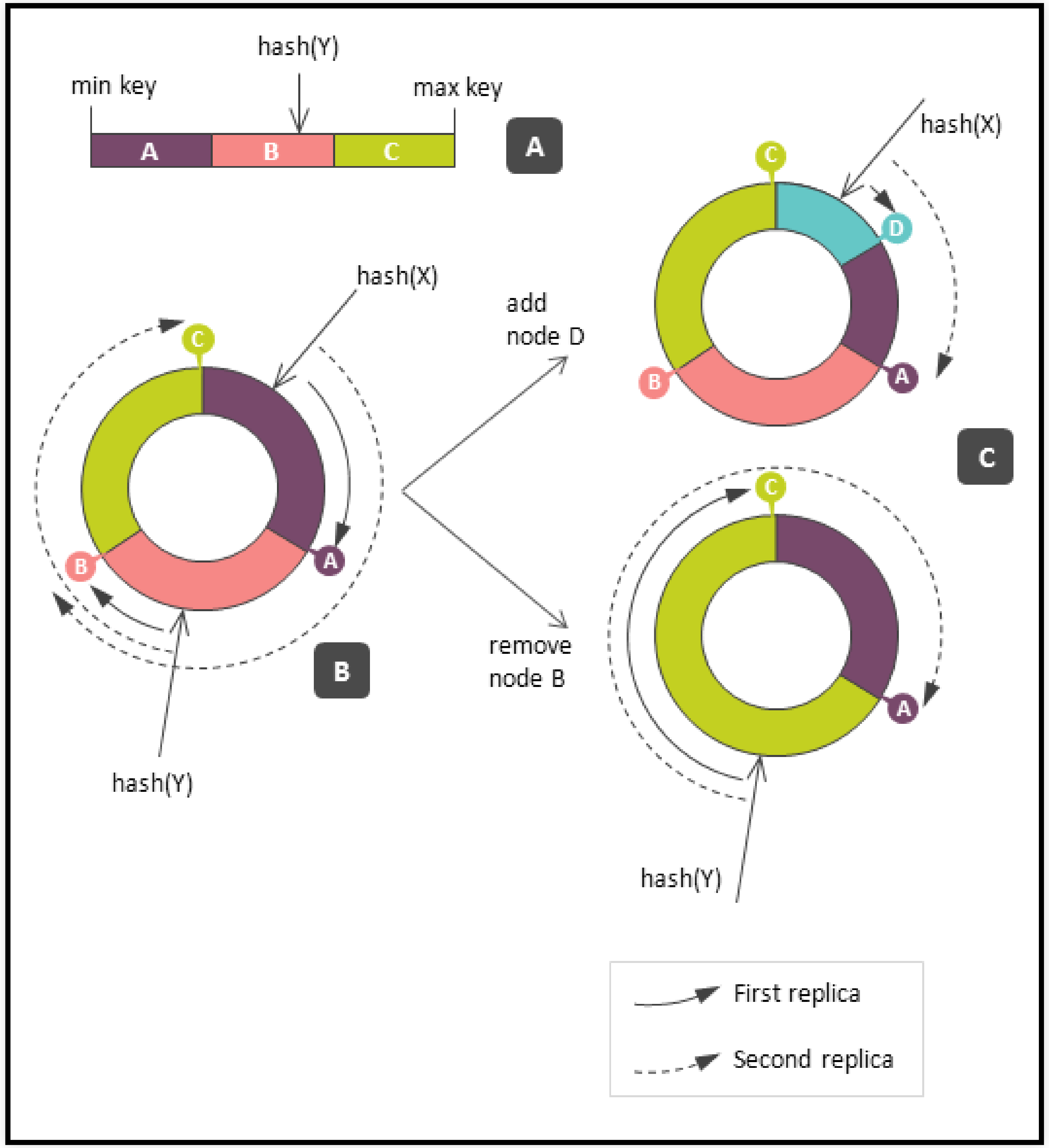
- use proximity information provided by snitches to determine locality of a copy
  - **SimpleStrategy**:
    - use only for a single data center and one rack
    - place the copy to the next available node (clockwise)
  - **NetworkTopologyStrategy**: specifies how many replicas you want in each DC
-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

Replication strategies

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

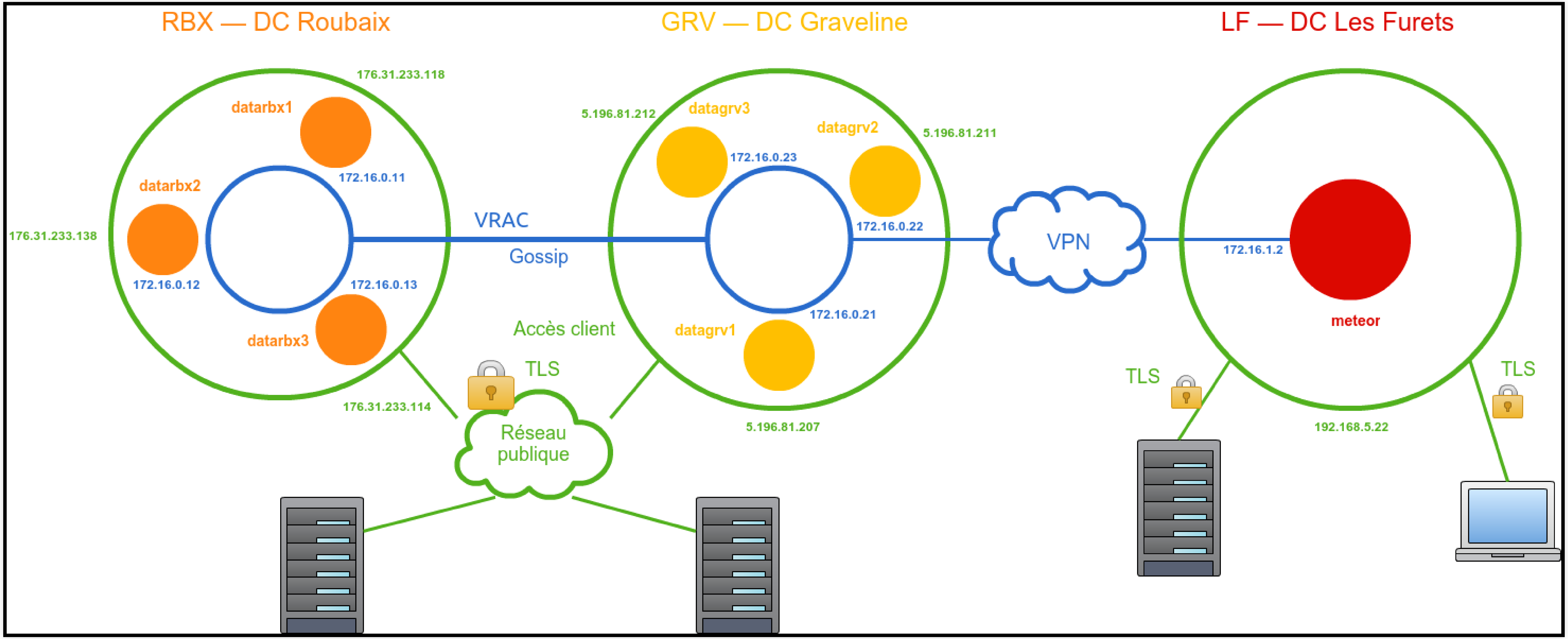
```
CREATE KEYSPACE lesfurets
WITH replication =
{'class': 'NetworkTopologyStrategy', 'RBX': 2,'GRV':2,'LF':1};
```

SimpleReplicationStrategy



NetworkTopologyStrategy

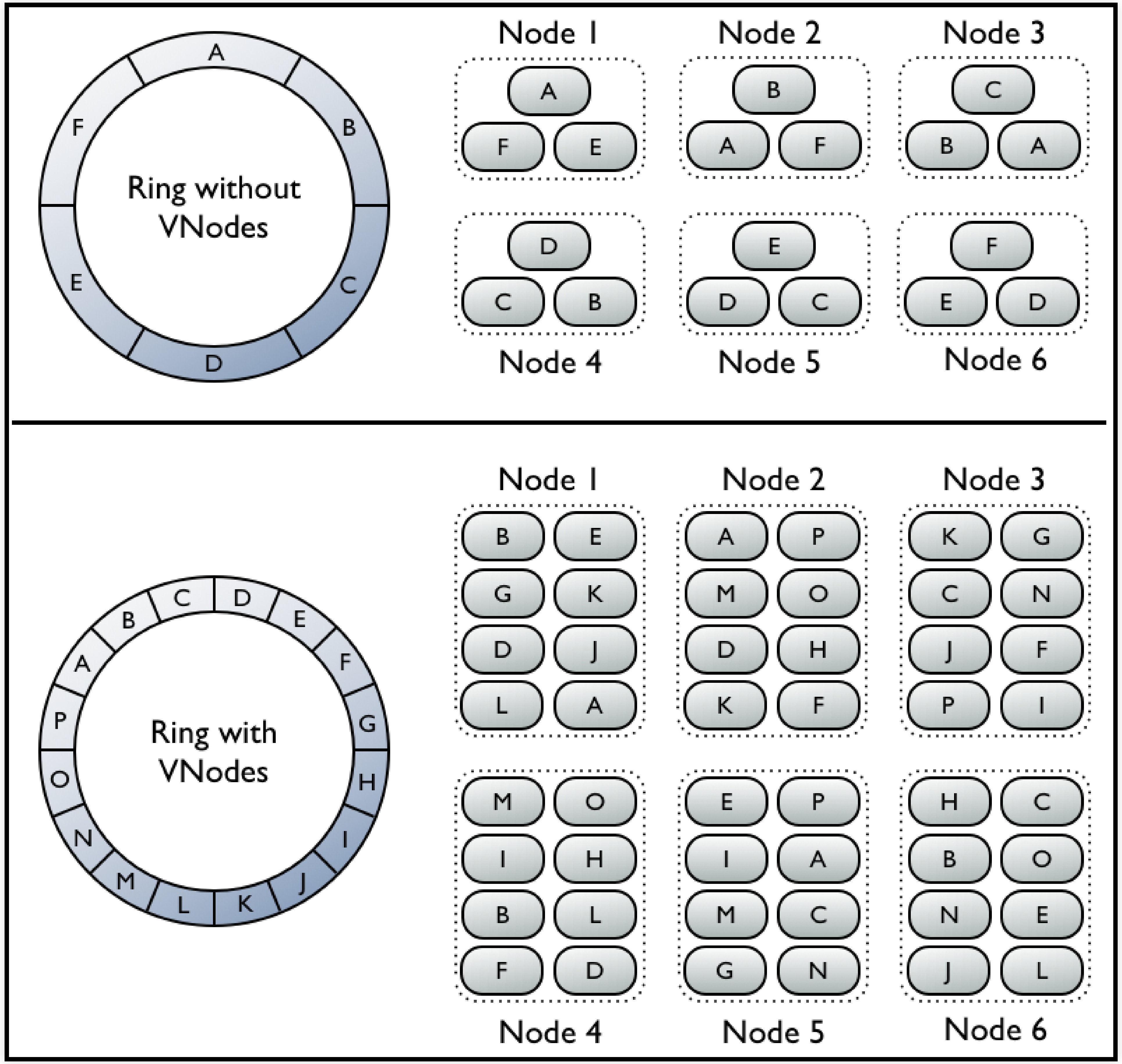
```
CREATE KEYSPACE lesfurets
WITH replication =
{'class': 'NetworkTopologyStrategy', 'RBX': 2,'GRV':2,'LF':1};
```



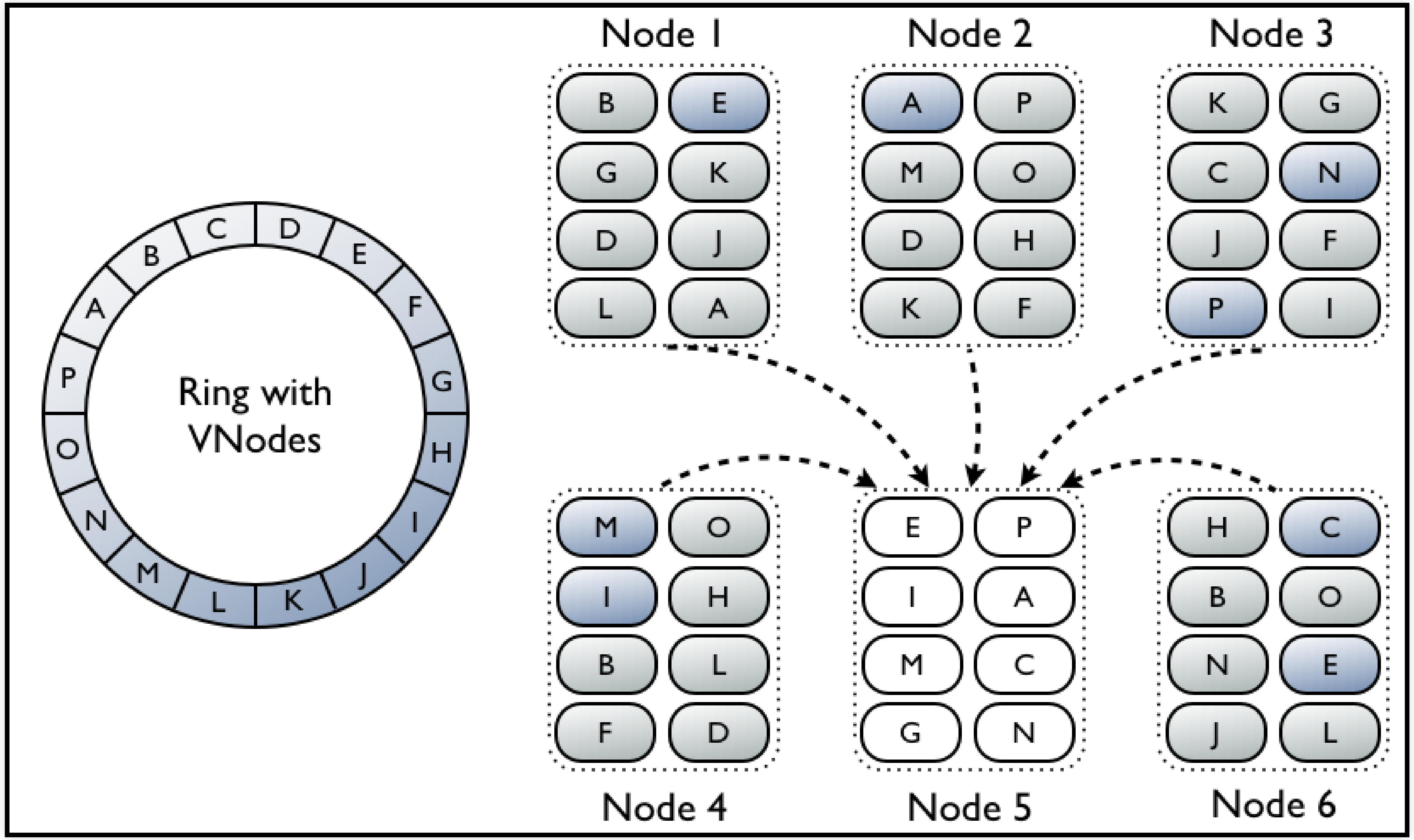
Token allocation

- 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+)

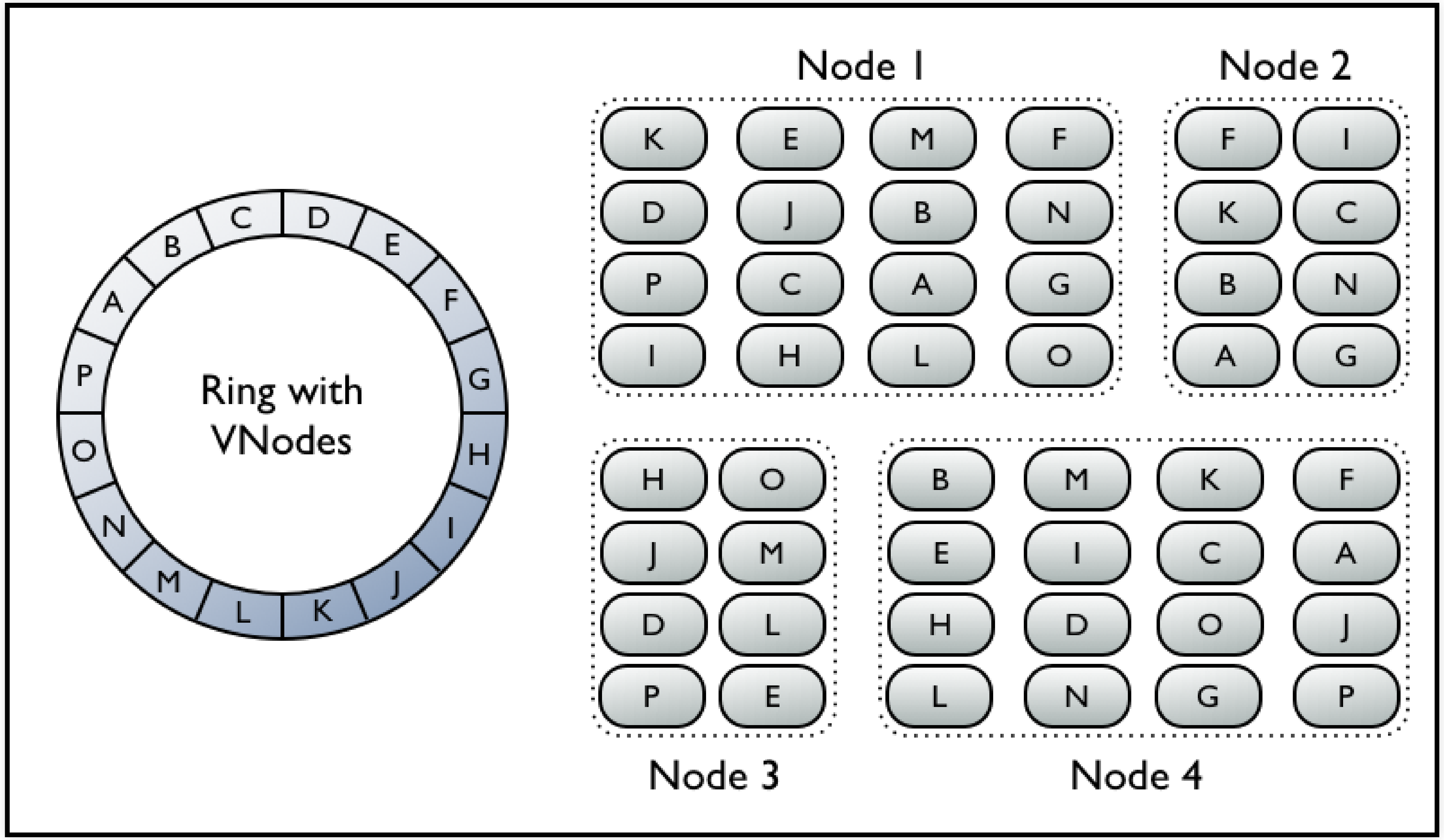
Virtual nodes (VNodes)



VNodes: remapping keys



VNodes: heteregenous clusters



Plan

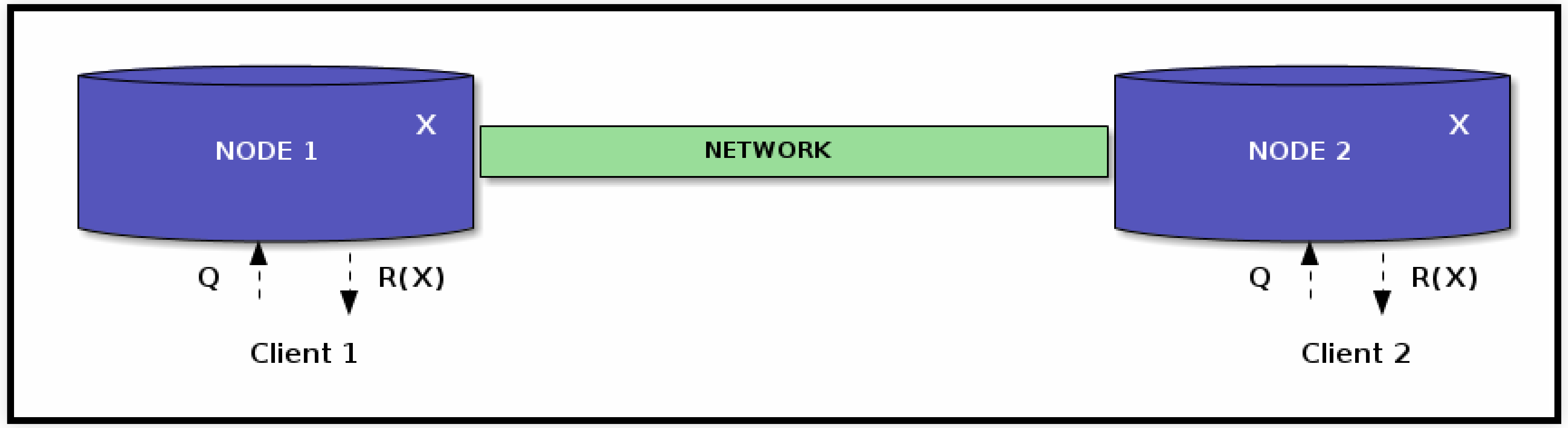
- Motivation
- Apache Cassandra
  - Partitioning and replication
  - **Consistency**
- Practice: Tune consistency in Apache Cassandra

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.

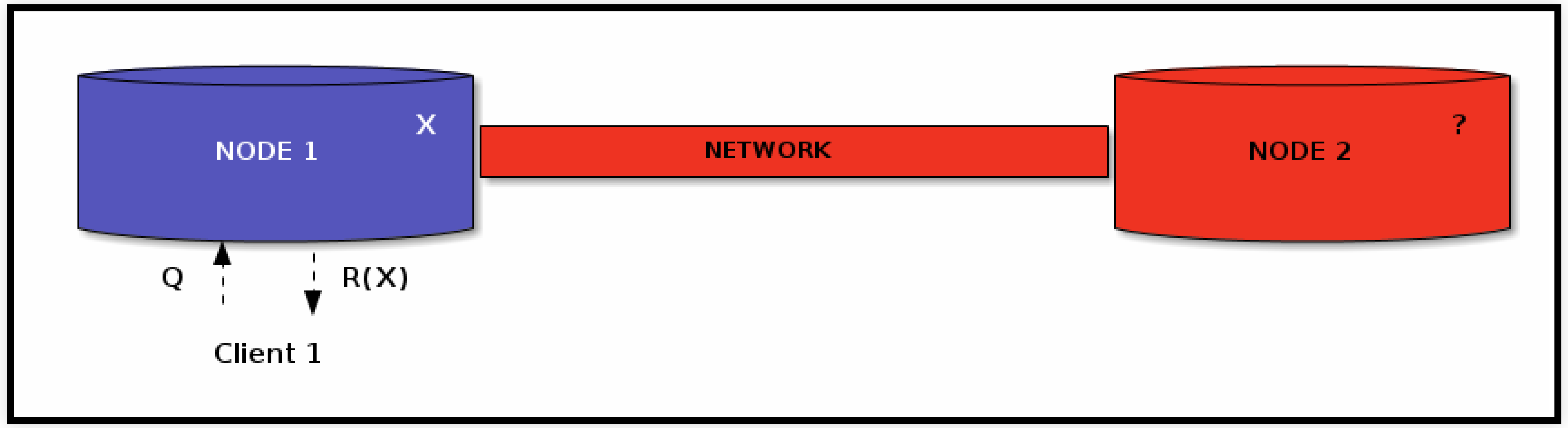
Consistency

- *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)



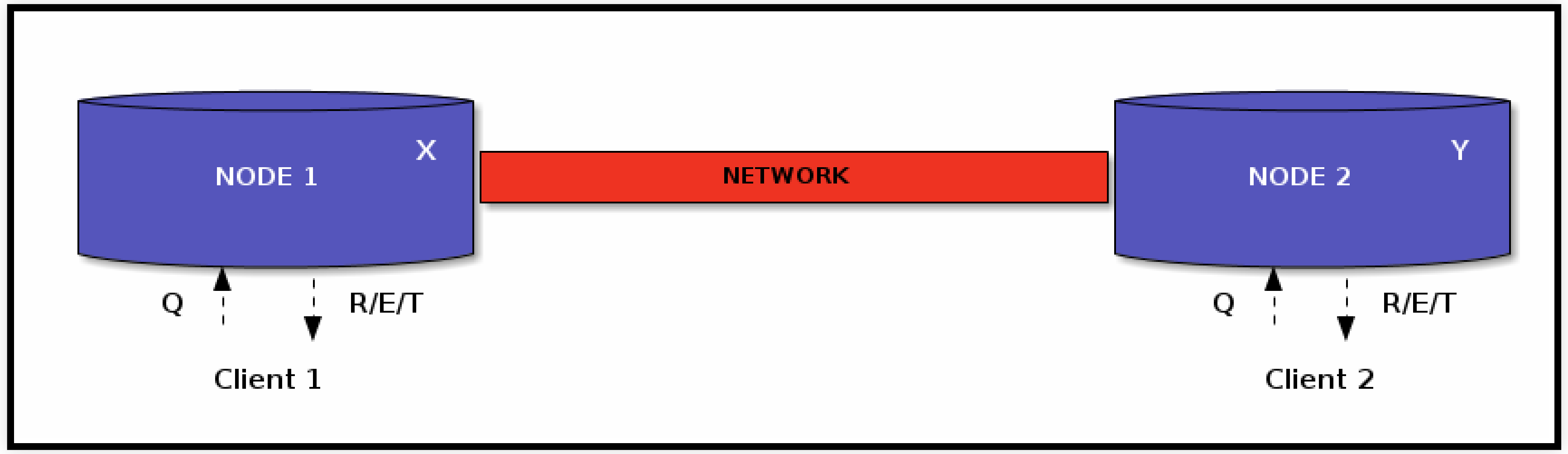
Availability

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



Partition tolerance

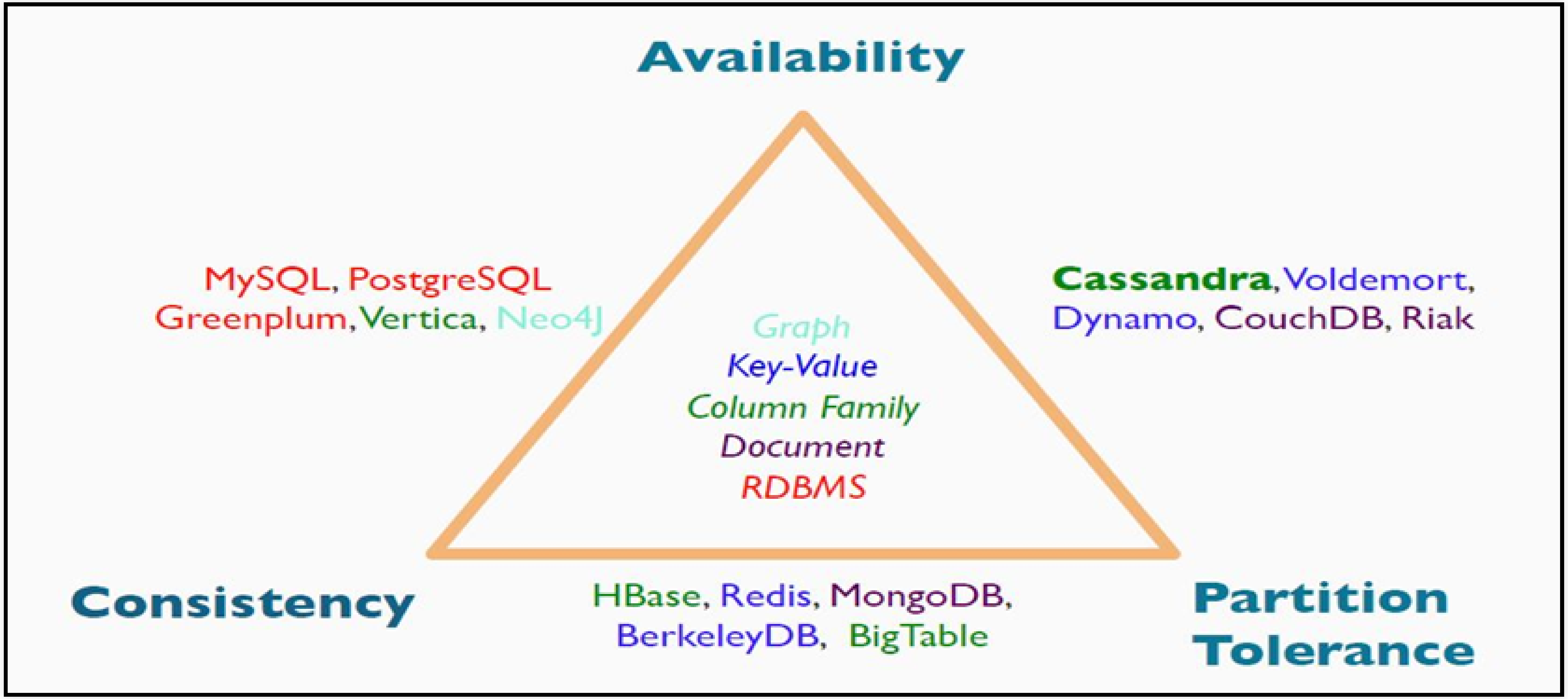
- *ability to **function (return a response, error, timeout)** when network partitions occur*



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



Availability

MySQL, PostgreSQL
Greenplum, Vertica, Neo4J

Cassandra, Voldemort,
Dynamo, CouchDB, Riak

Graph
Key-Value
Column Family
Document
RDBMS

Consistency

HBase, Redis, MongoDB,
BerkeleyDB, BigTable

Partition
Tolerance

BASE

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

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.

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 ?

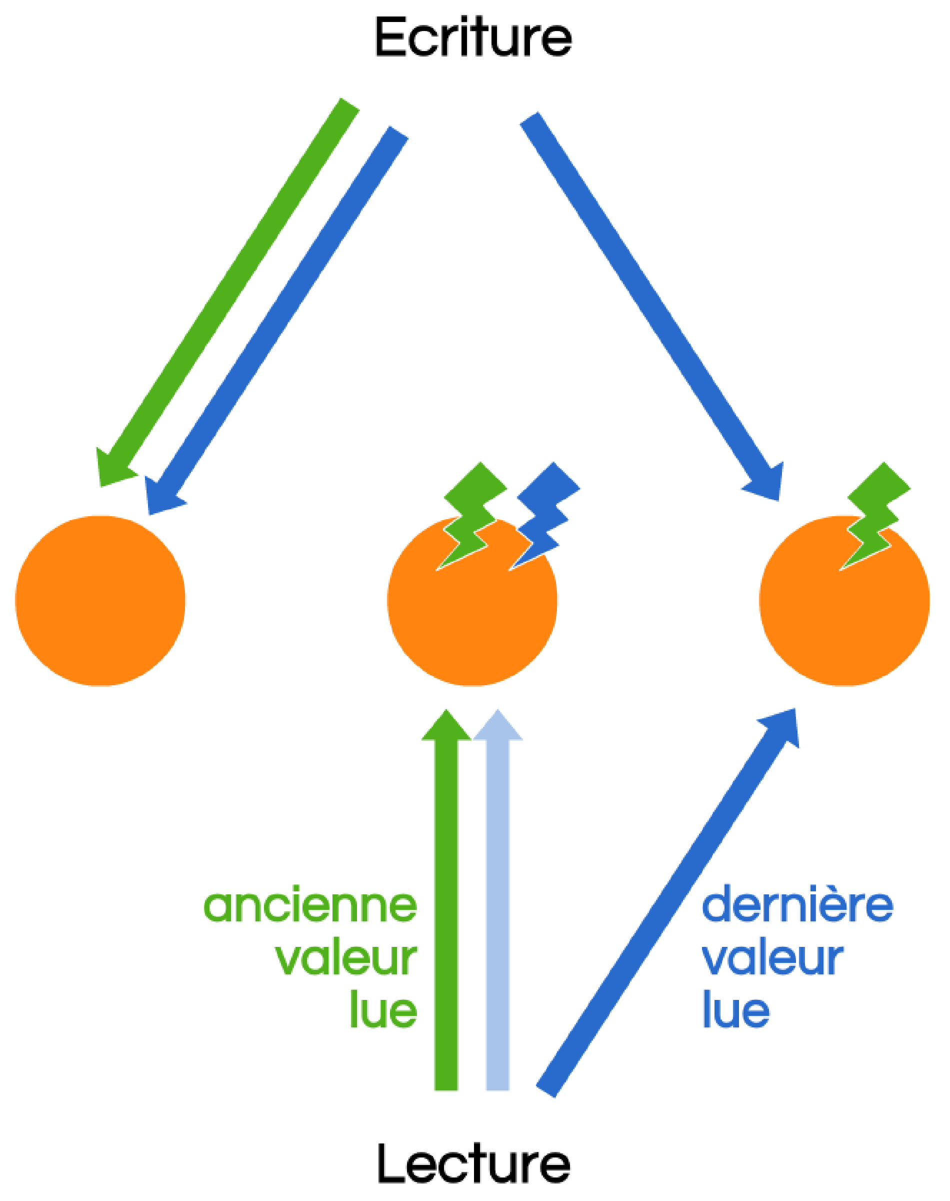
W : Niveau de cohérence en écriture (nb de nœuds)
R : Niveau de cohérence en lecture (nb de nœuds)
RF : Facteur de réplication

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 nœuds en panne



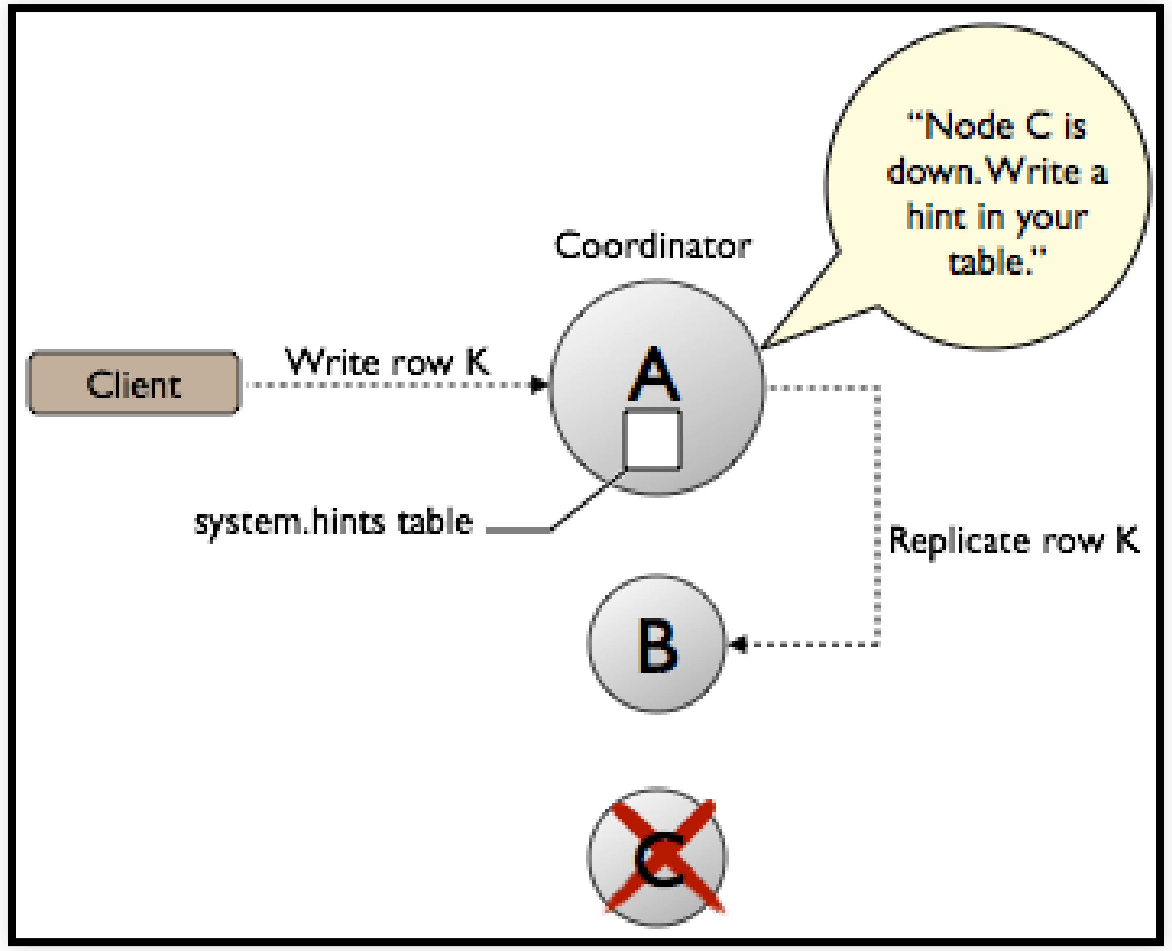
⚡ nœud en panne pendant l'écriture

Consistency mechanisms

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

Hinted handoff

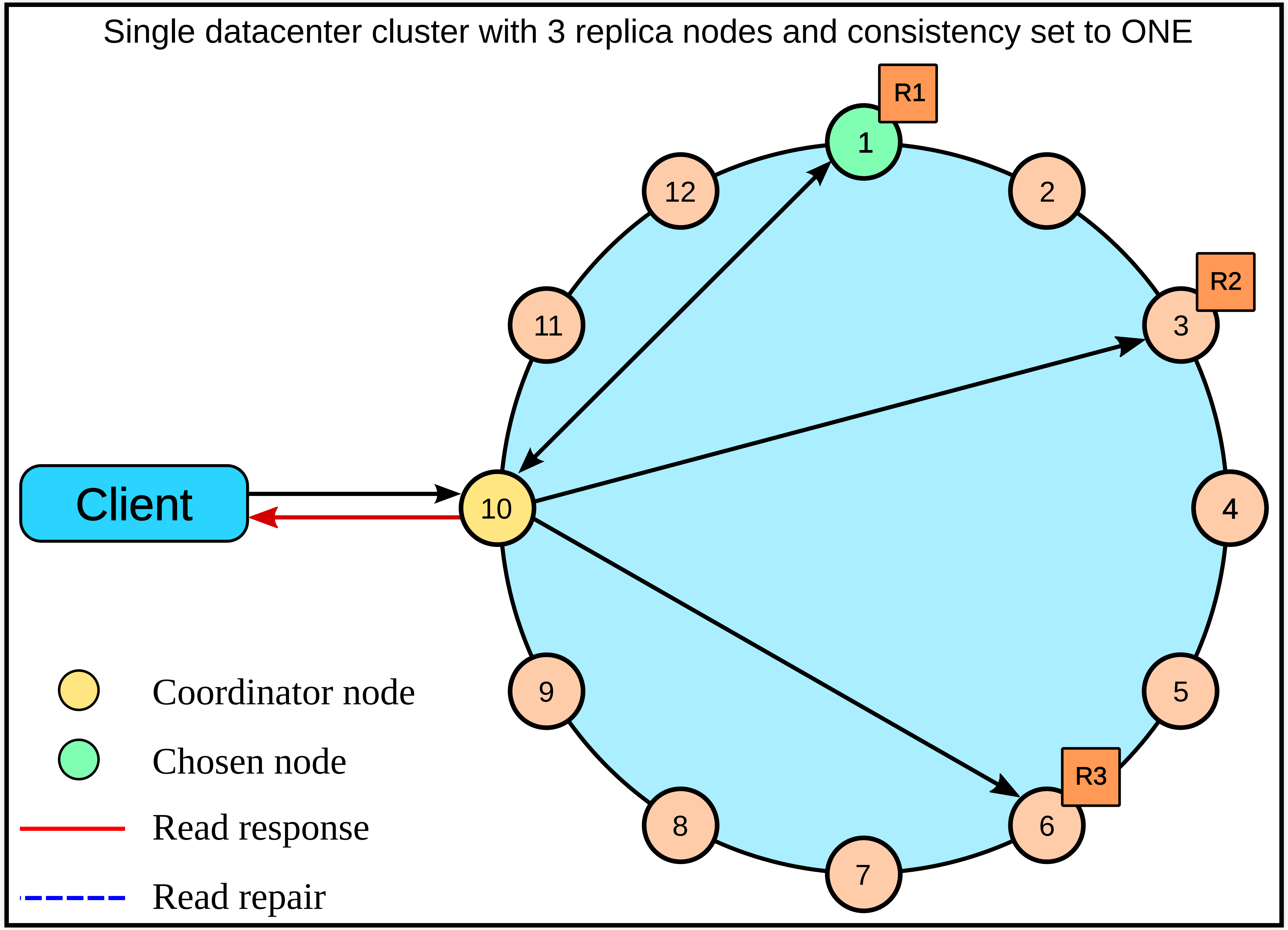
- ONE/QUORUM vs ANY (any node may ACK even if not a replica)
- if one/more replica(s) are down ⇒ **hinted handoff**



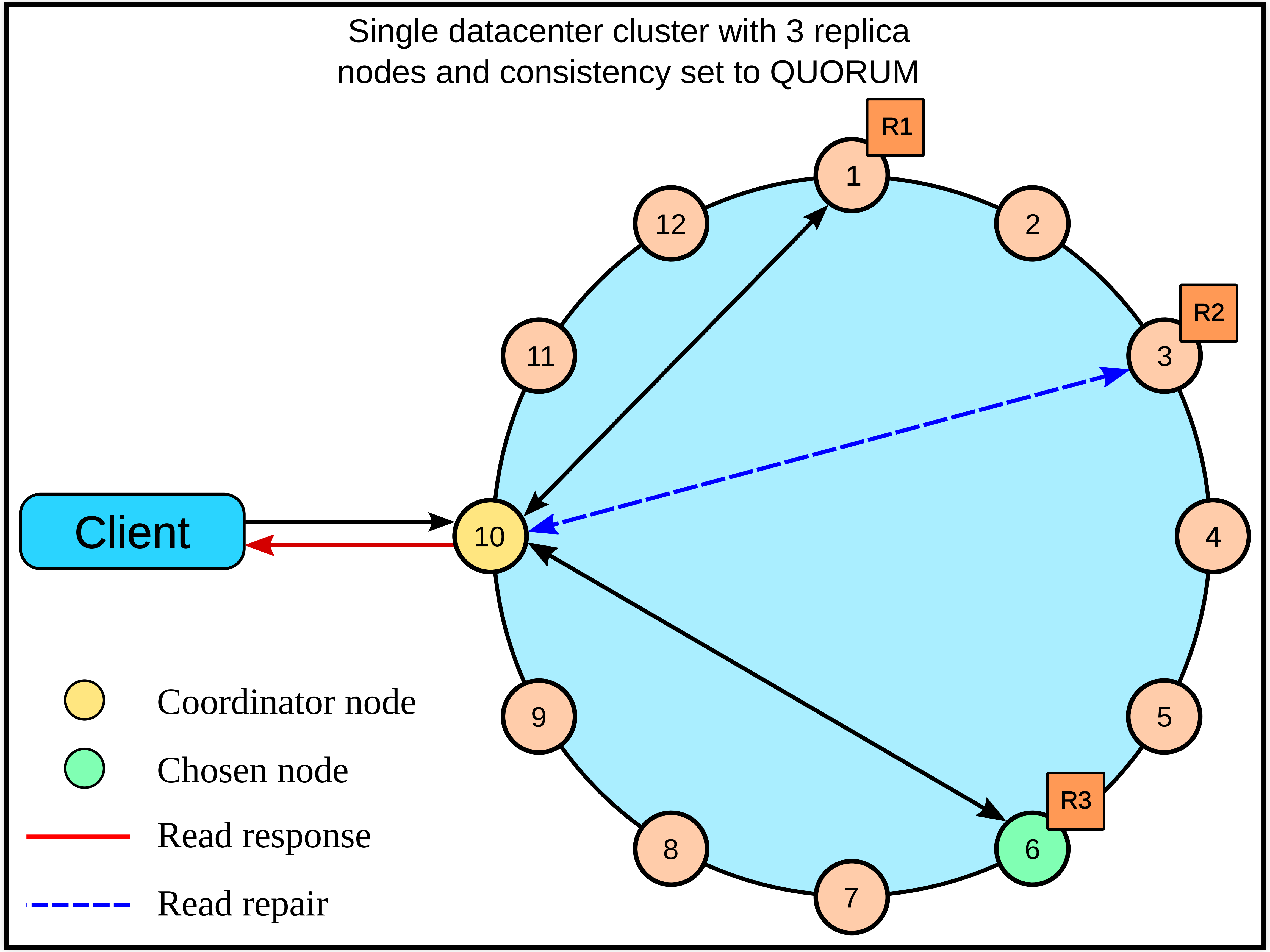
Read repairs

- Goal: **detect and fix inconsistencies during reads**
- **CL** = *ONE* ⇒ no data is repaired because no comparison takes place (unless **read_repair_chance** >0)
- **CL** > *ONE* ⇒ repair participating replica nodes in the foreground before the data is returned to the client.
    - send a direct read request to a chosen node that contains the data (fastest responding)
    - send digest requests to other replicas
    - if digest does not agree send direct request to replicas and determine the **latest** data (*column level!*)
    - writes the most recent version to any replica node that does not have it
    - return the data to the client

Read repairs ONE



Read repairs QUORUM



Read repairs QUORUM DC

Anti-entropy repair (nodetool repair)

- for each token range, read and synchronize the rows
- to insure the consistency this tool must be run regularly !

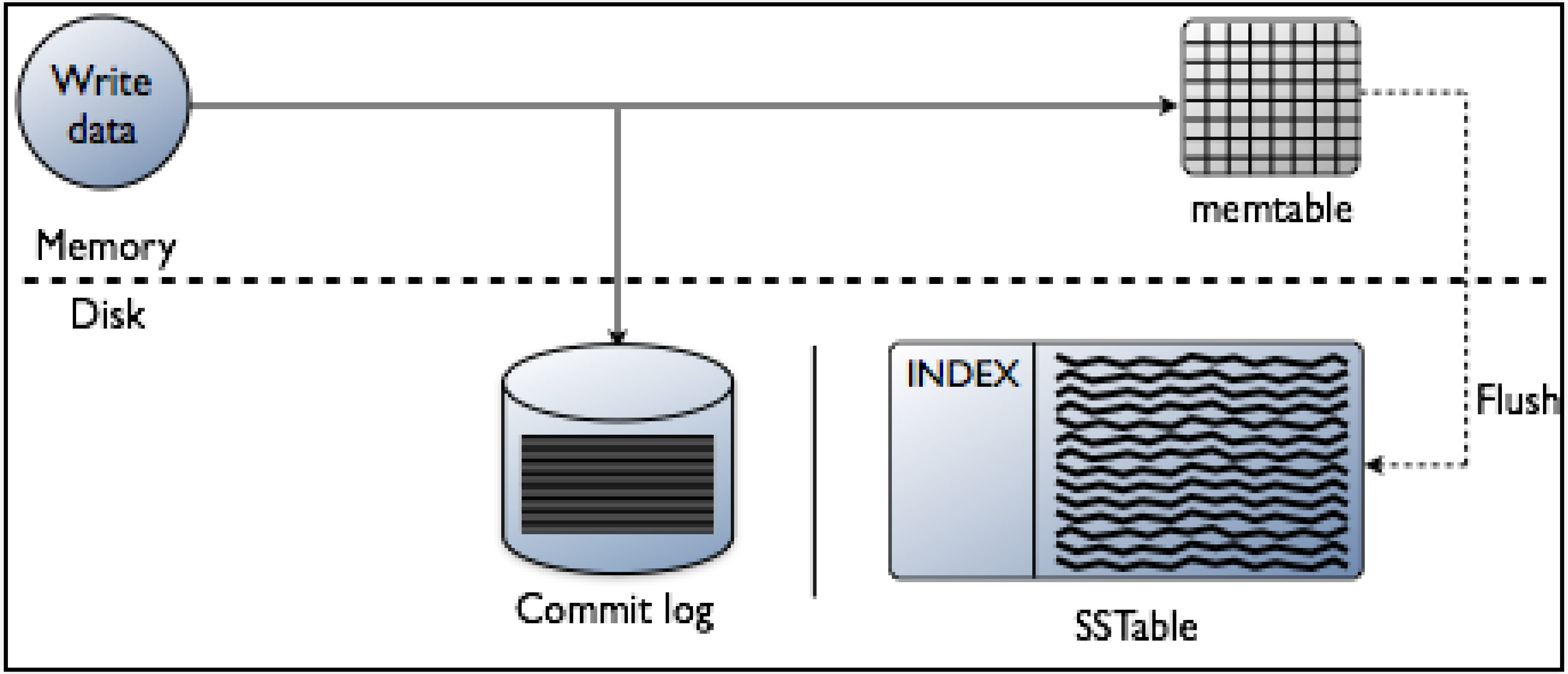
```
nodetool repair [options] [<keyspace_name> <table1> <table2>]

nodetool repair --full
```

- **manual operation, must be scheduled !** (Cassandra Reaper, 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 ⇒ sync before ACK the write

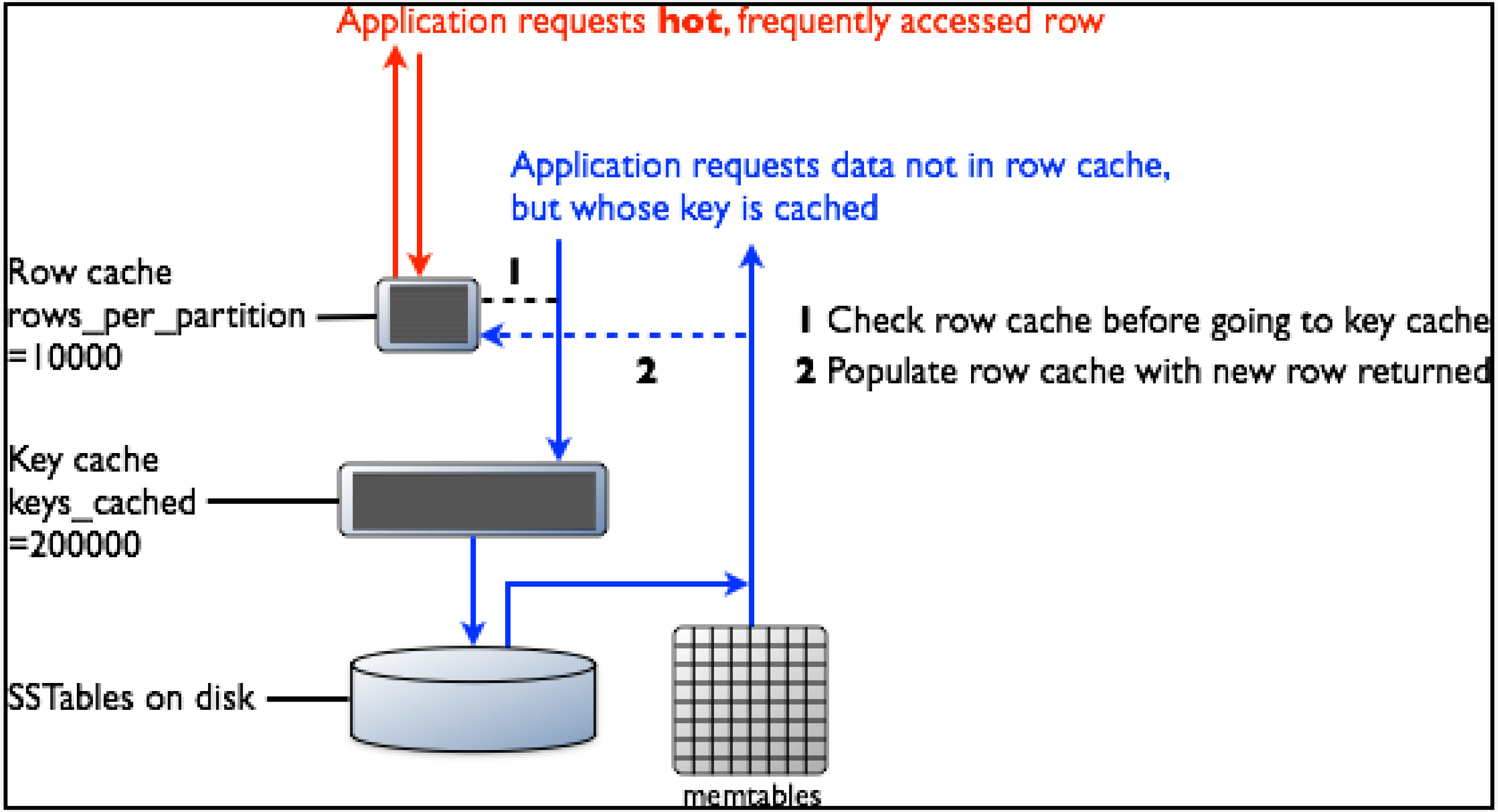
Cassandra write path



Cassandra compactions

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

Cassandra read path (caches)



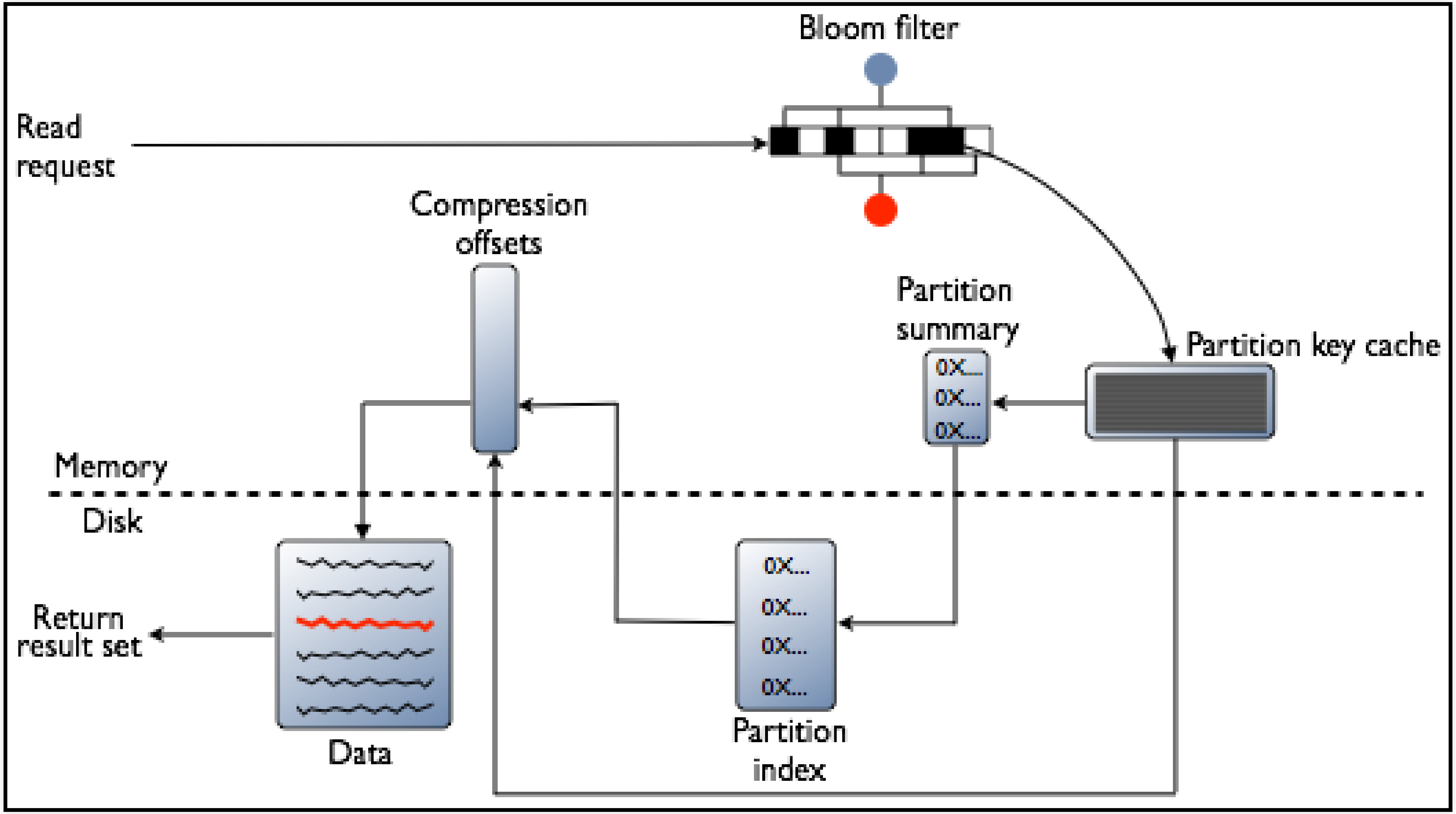
Application requests **hot**, frequently accessed row

Application requests data not in row cache, but whose key is cached

1 Check row cache before going to key cache

2 Populate row cache with new row returned

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 ❶

$ccm node1 stop ❷

$ccm node1 cqlsh ❸
```

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                               Rack
UN  127.0.0.1  47.66 KB   1       33.3%   aaa1b7c1-6049-4a08-ad3e-3697a0e30e10  rack1
UN  127.0.0.2  47.67 KB   1       33.3%   1848c369-4306-4874-afdf-5c1e95b8732e  rack1
UN  127.0.0.3  47.67 KB   1       33.3%   49578bf1-728f-438d-b1c1-d8dd644b6f7f  rack1
```

CQLSh

- standard CQL client

```
[bigdata@bigdata ~]$ ccm node2 cqlsh ❶
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 19.39.0]
Use HELP for help.
cqlsh> SELECT * FROM system.schema_keyspaces ; ❷
```

VSCODE plugin

Ressources:

Apache Cassandra documentation

Datastax documentation

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

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