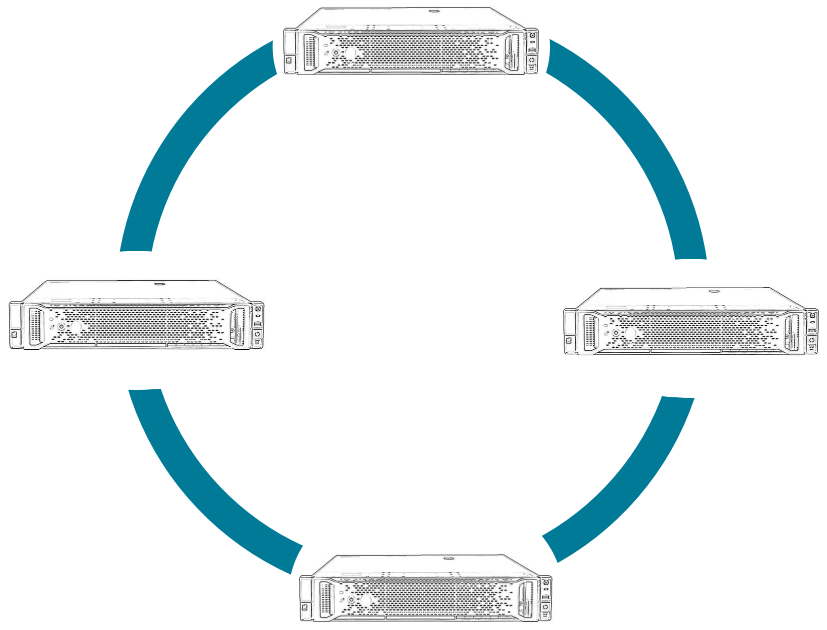
DataStax Enterprise Architecture

Negib Marhoul, Solution Engineer, DataStax

8. November 2017

Agenda

Design Goals and Objectives



- Continuously Available
- Master Less
- Fully Distributed
- Shared-Nothing Architecture
- Build In Replication
- Linear Scalability
- Scale out

DATASTAX

Architecture

Apache Cassandra™ Architecture

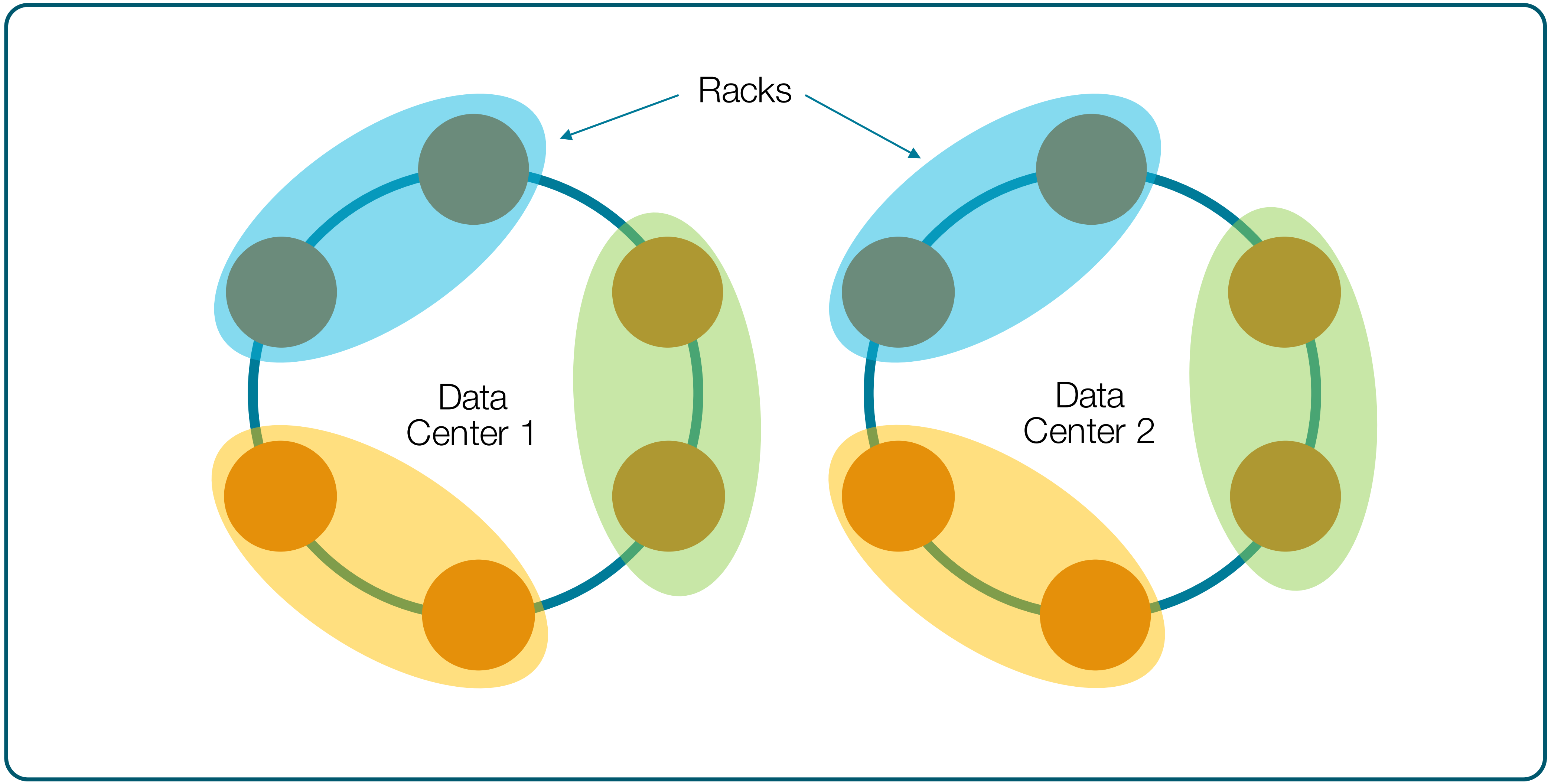
- Cluster layer
-   Amazon DynamoDB paper
-   masterless architecture

- Data-store layer
-   Google Big Table paper
-   Columns/columns family

- All nodes are peers
  - Including seed nodes
  - No master
  - Discovery through gossip

- Built-in replication
  - Simplify your architecture!

Cluster

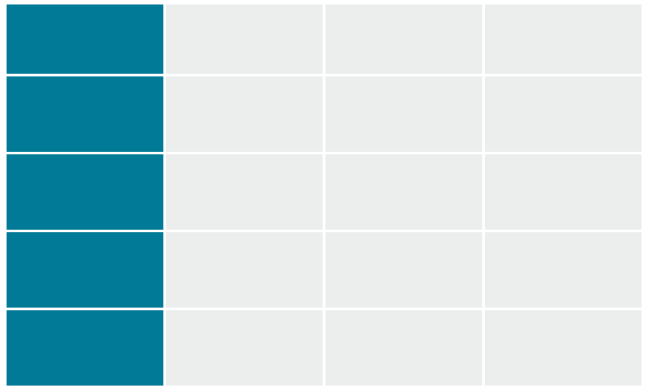


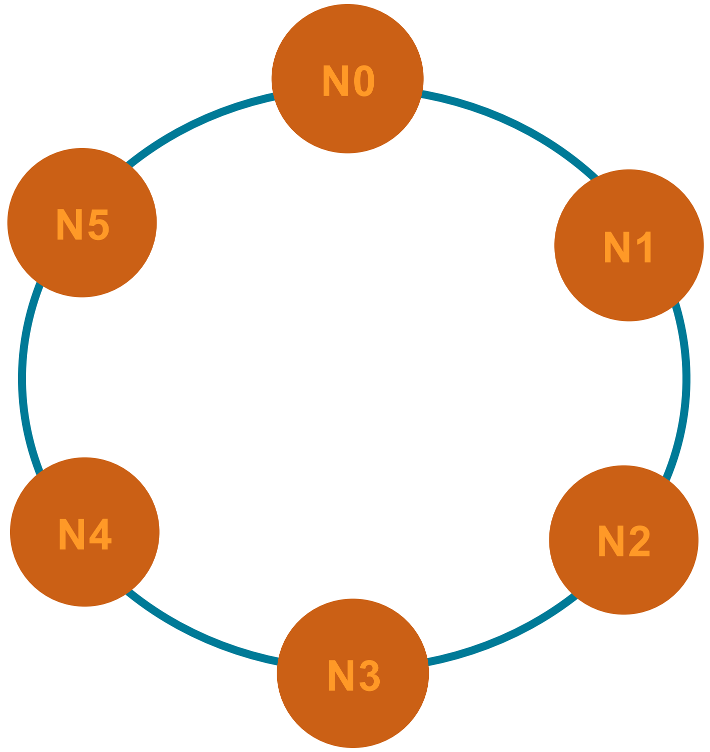
Tokens

Data is partitioned after its partition key

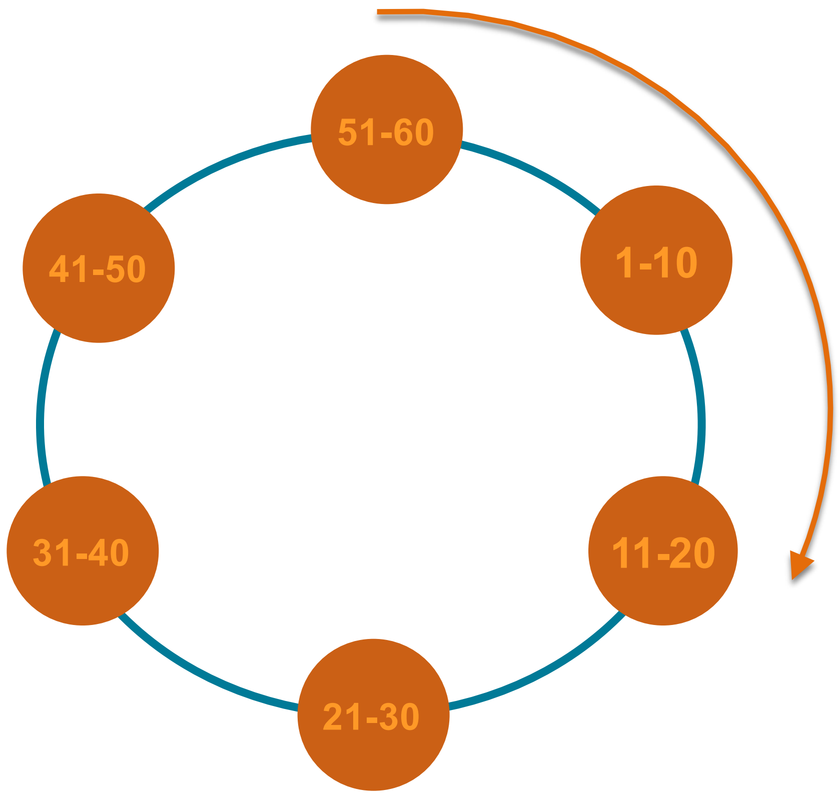
A unique token is allocated to a partition

Token = random hash of #partition





Token Ranges



Token Range : - 2^{63} to 2^{63}

Example with **Replication Factor 3**

N3 will own data for tokens 1 – 30

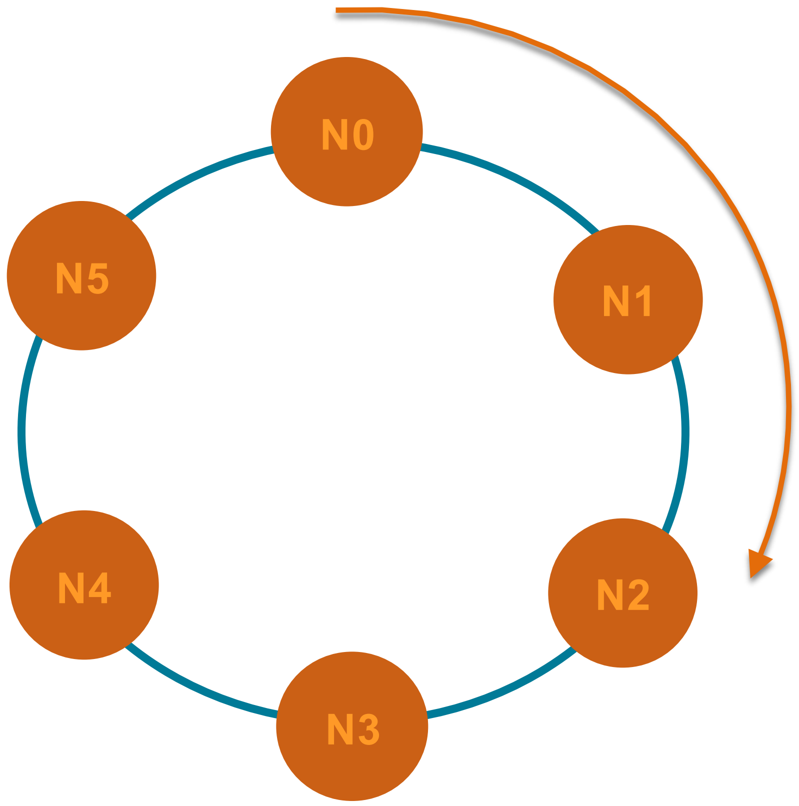
```
Token Range :  1-10, owned by N1,N2,N3
Token Range : 11-20, owned by N2,N3,N4
Token Range : 21-30, owned by N3,N4,N5
```

- Primary key
  - Partition key
  - Clustering columns
- Partitioner
  - Generates unique hash from partition key
- Replication strategy
  - Token hash determines starting point
  - Determines replica placement

Data Distribution

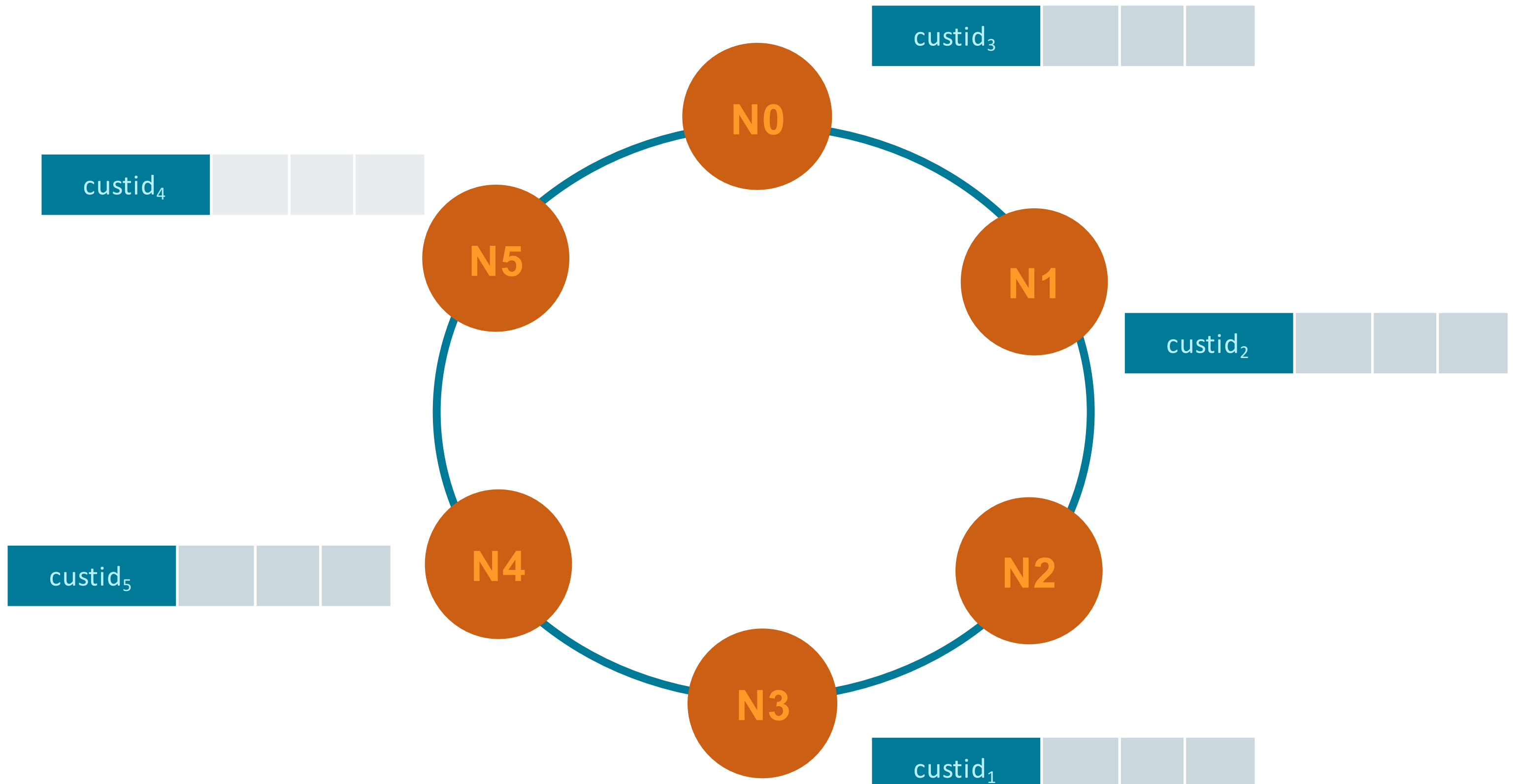
Token = hash of #partition → #node

Token1	custid 1			
Token2	curstid 2			
Token3	curstid 3			
Token4	curstid 4			
Token4	curstid 5			



Data is evenly distributed and clock wise replicated

Data Distribution



Cassandra Query Language

```
CREATE KEYSPACE retailer WITH replication =
{'class': 'NetworkTopologyStrategy', 'DC1': '3'}
AND durable_writes = true;
```

Replication Factor

```
CREATE TABLE retailer.sales_by_customer (
    custid int,
    salesdt text,
    comment text,
    discount double,
   revenue double,
    PRIMARY KEY (custid, salesdt));
```

Partition Key

```
SELECT * FROM sales_by_customer where custid=1 OR custid=2 AND salesdt >=20160401;
```

DATASTAX

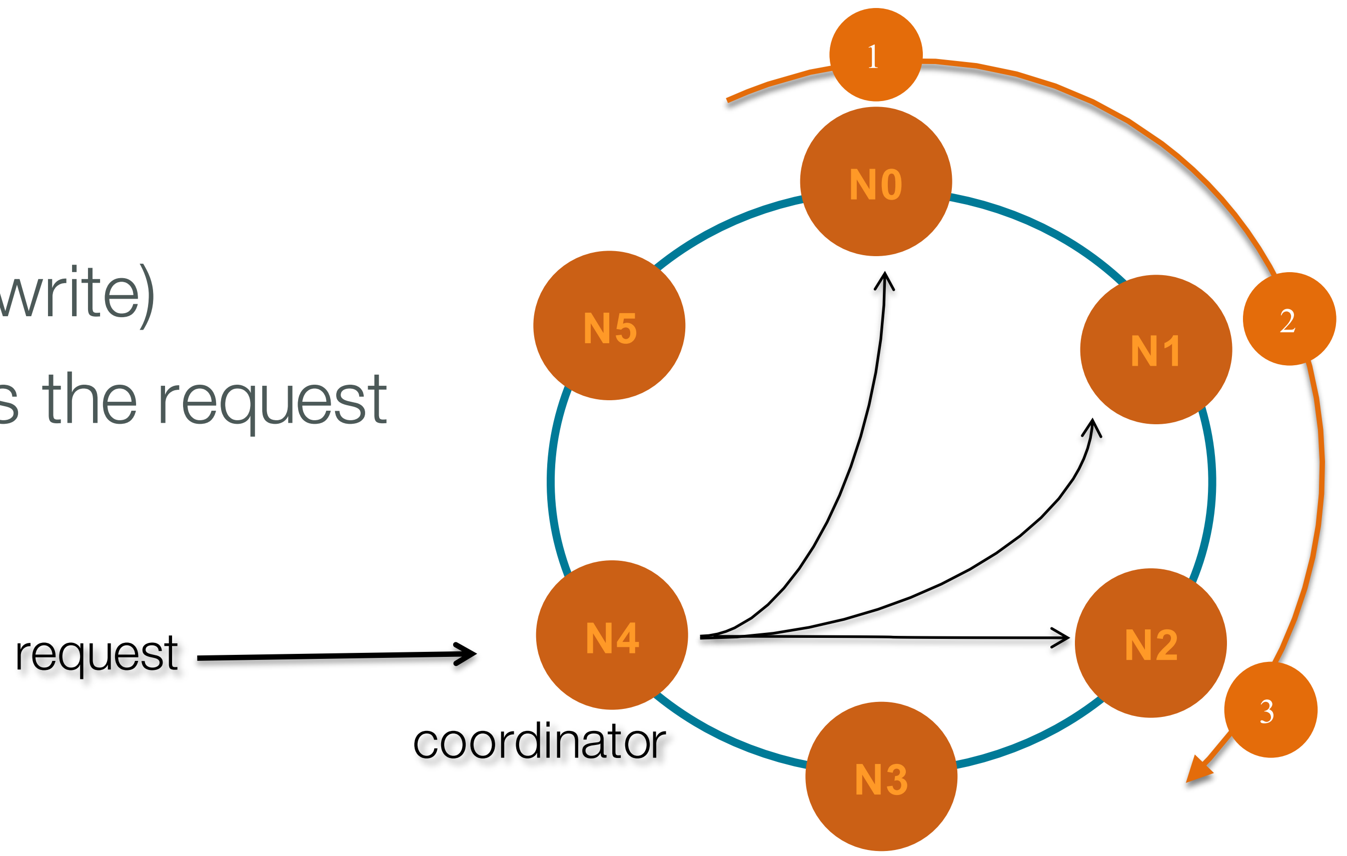
Lab 1 : Accessing the cluster

Tunable Consistency
Read and write request handling

Coordinator node

Incoming requests (read/write)

Coordinator node handles the request



Every node can be coordinator → masterless

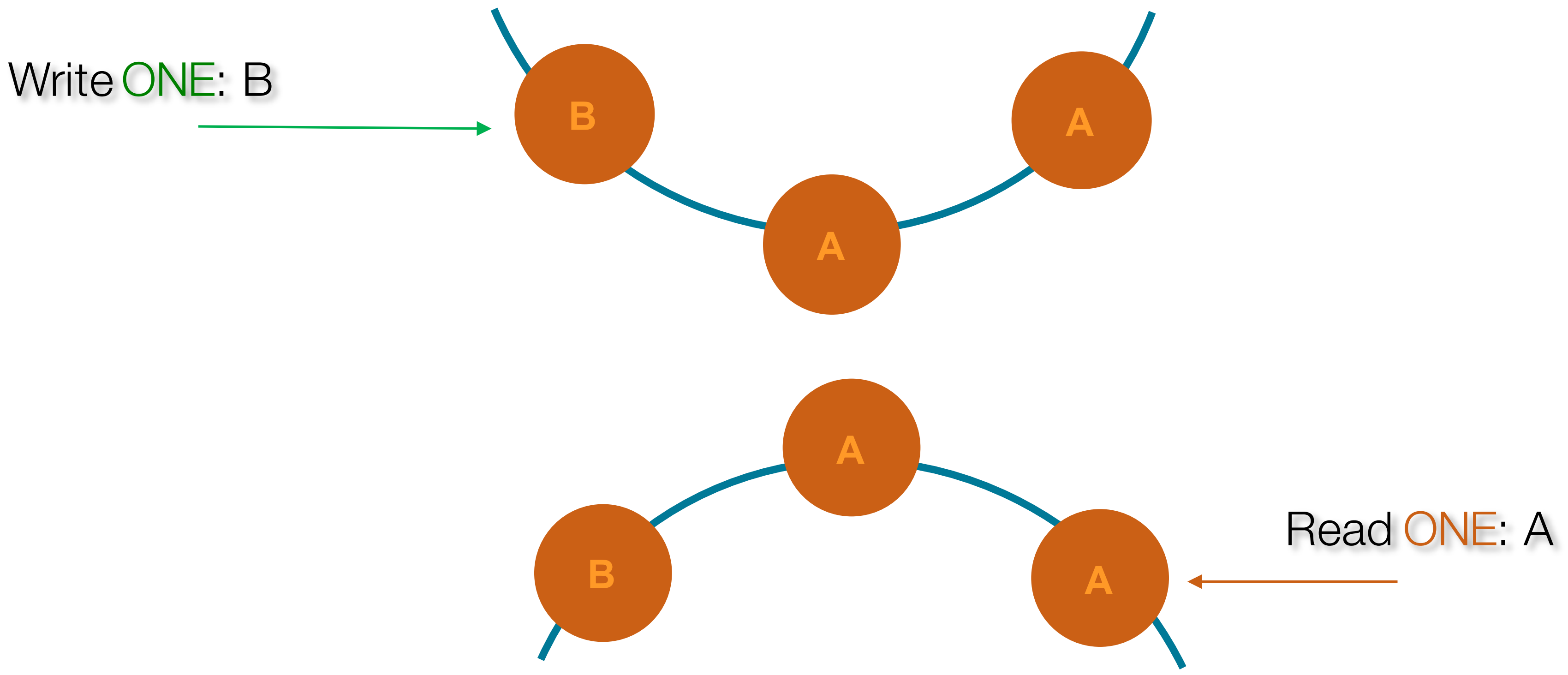
Consistency

Tunable at runtime

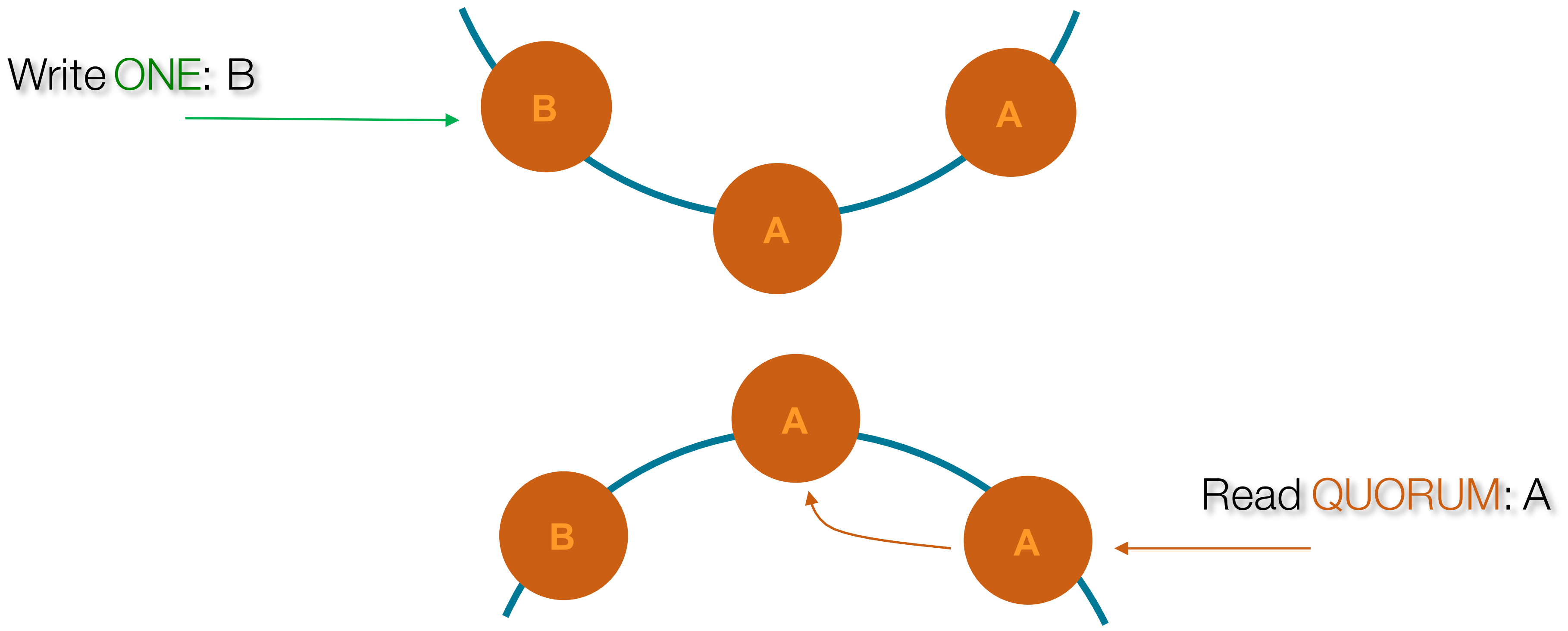
- ONE
- QUORUM (strict majority w.r.t. RF)
- ALL

Apply both to read & write

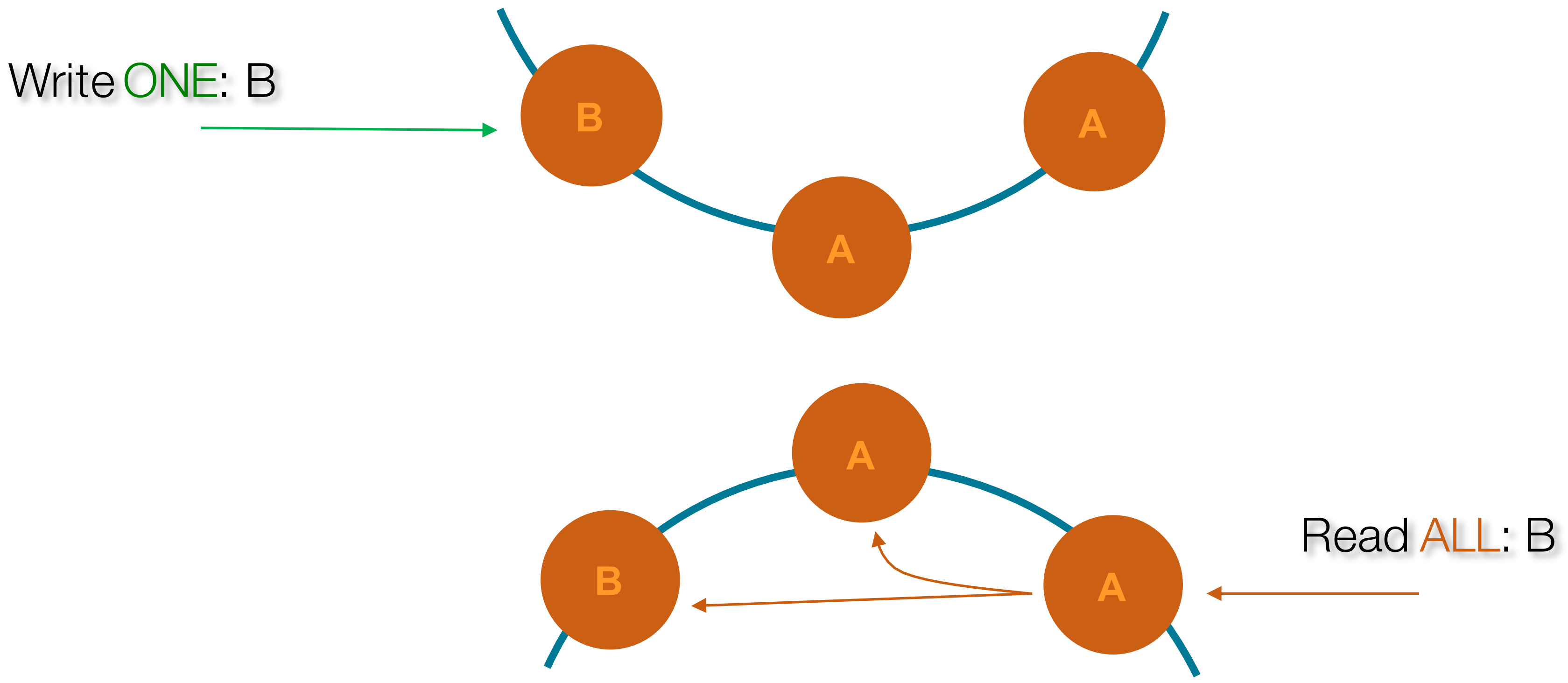
- RF = 3, Write ONE, Read ONE



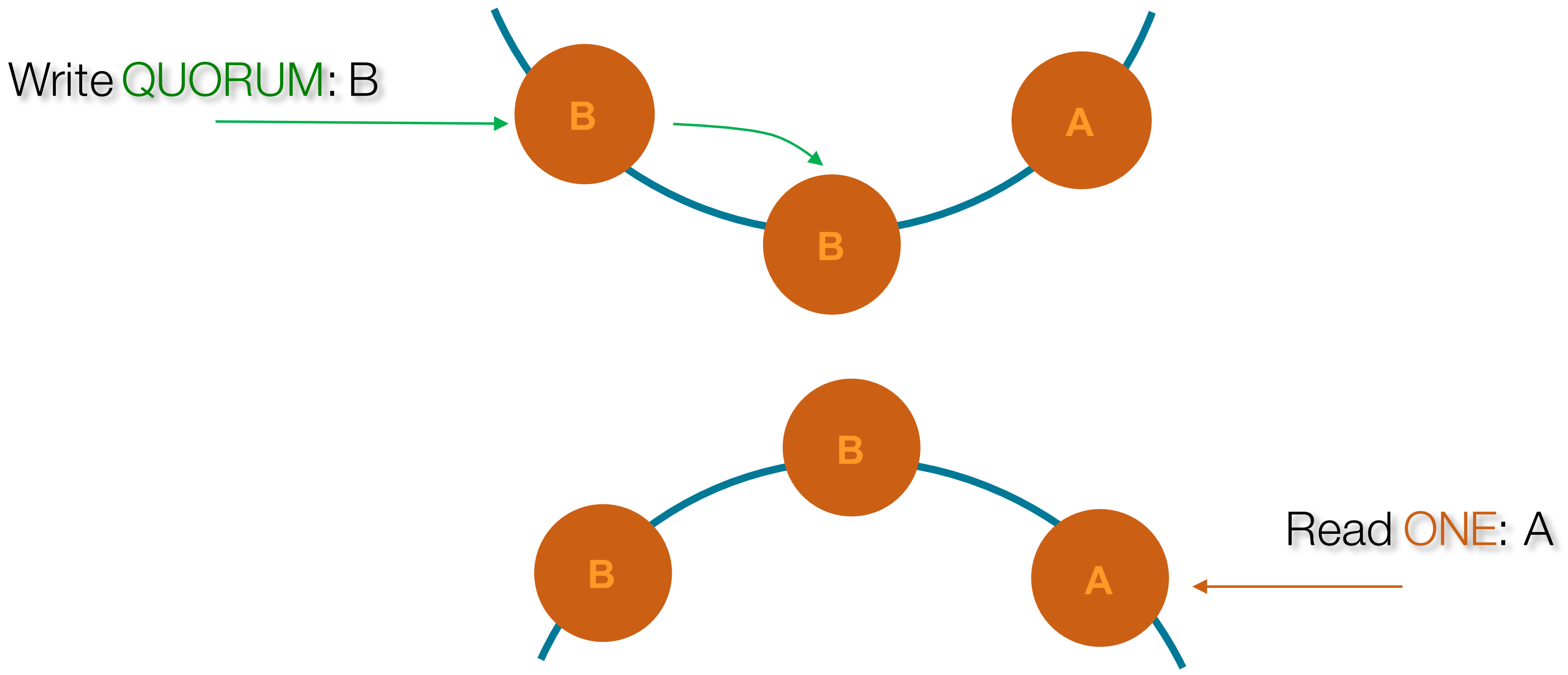
- RF = 3, Write ONE, Read QUORUM



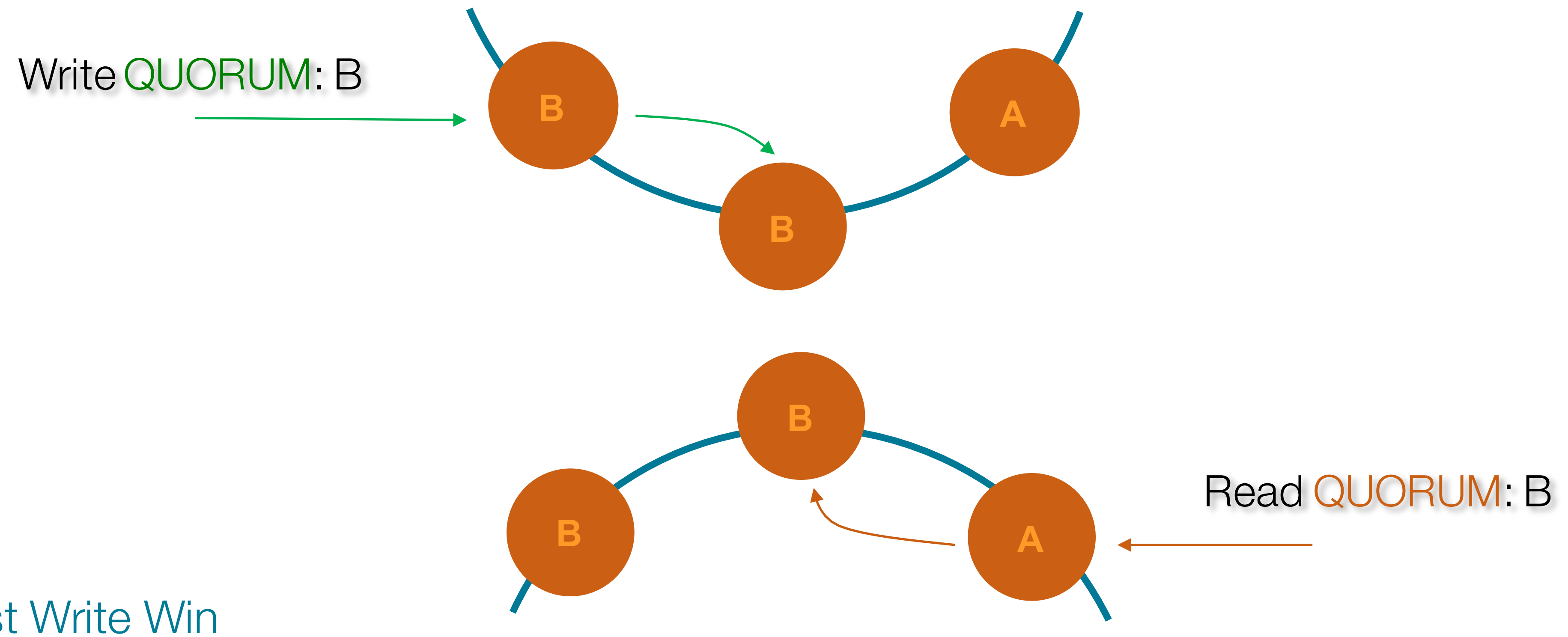
- RF = 3, Write ONE, Read ALL



- RF = 3, Write QUORUM, Read ONE

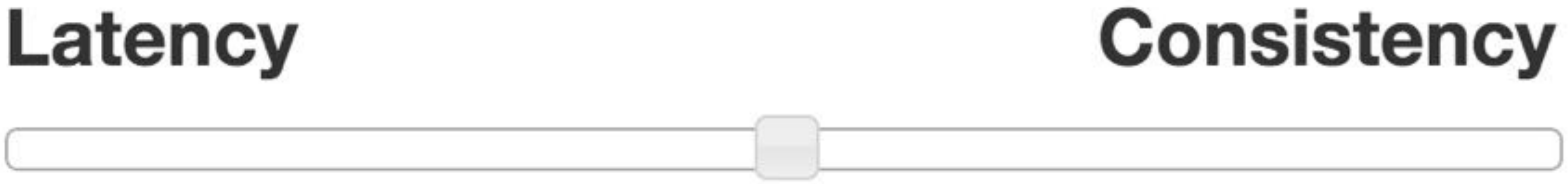


- RF = 3, Write QUORUM, Read QUORUM



- Last Write Win
- R+W > RF = immediate consistency
- Background vs. foreground Read Repair. See more…

Consistency trade-off



Consistency summary

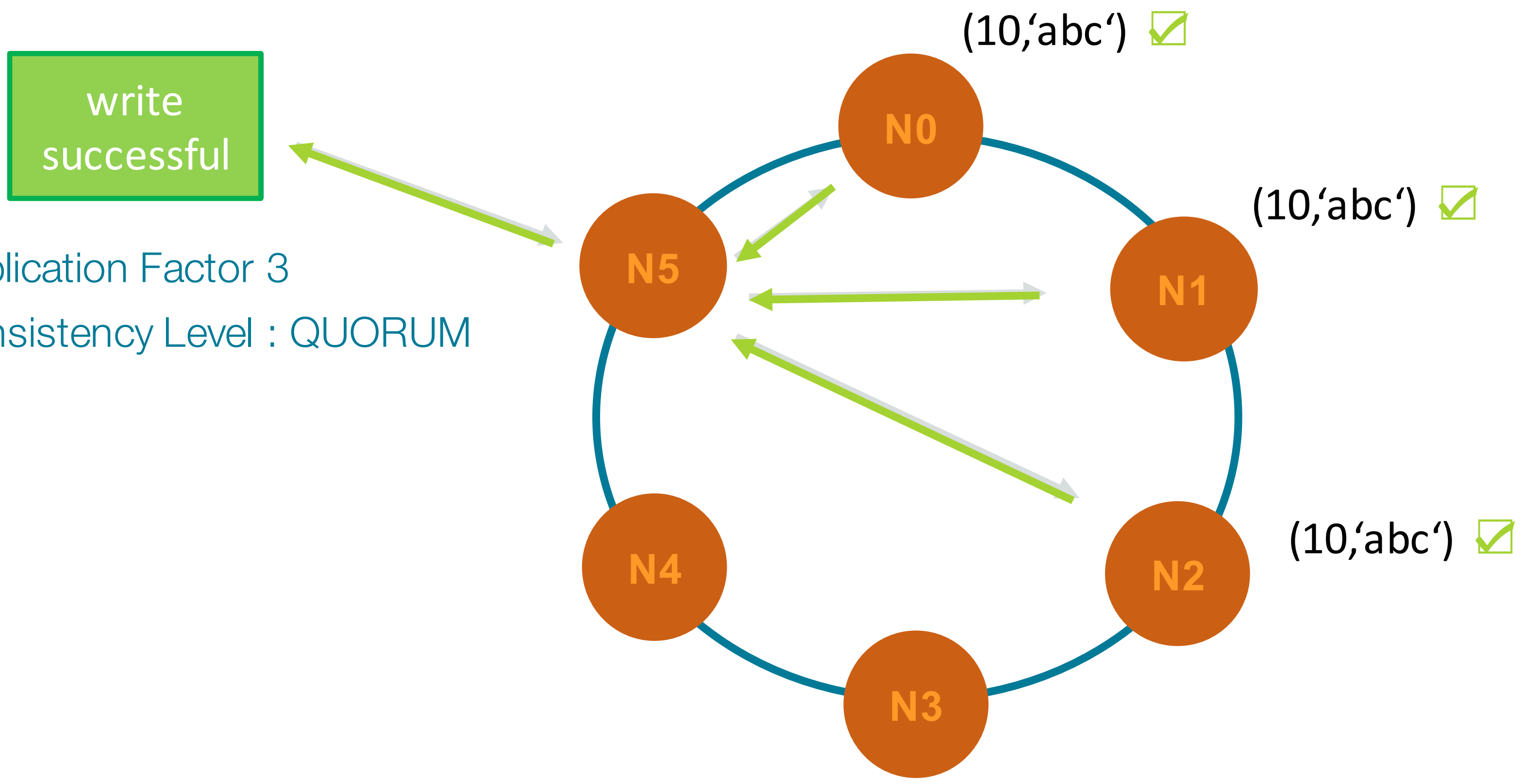
$\text{ONE}_{\text{Read}} + \text{ONE}_{\text{Write}}$

available for read/write even (N-1) replicas down

$\text{QUORUM}_{\text{Read}} + \text{QUORUM}_{\text{Write}}$

available for read/write even 1+ replica down

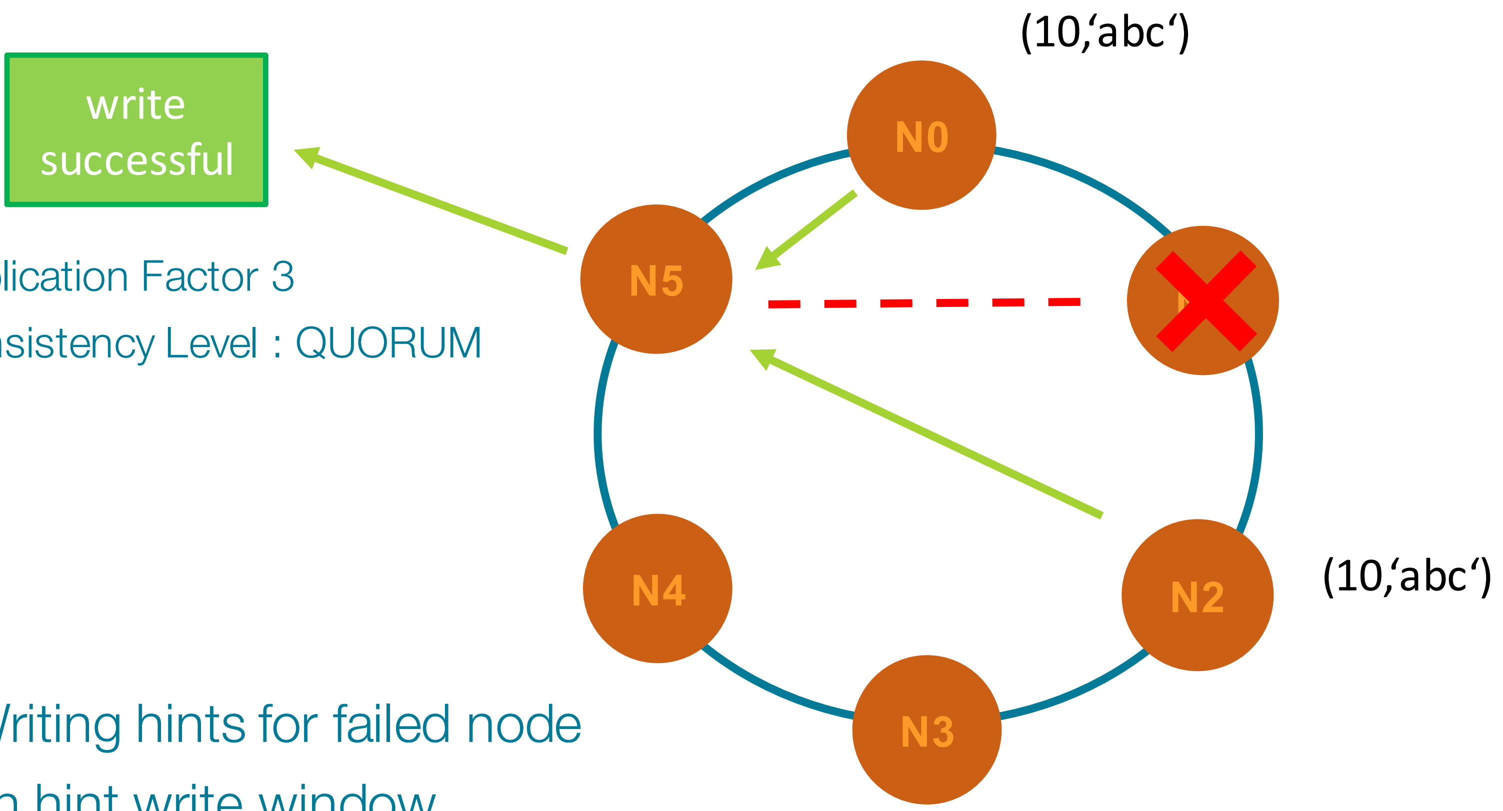
Write request handling



Replication Factor 3

Consistency Level : QUORUM

Write request handling

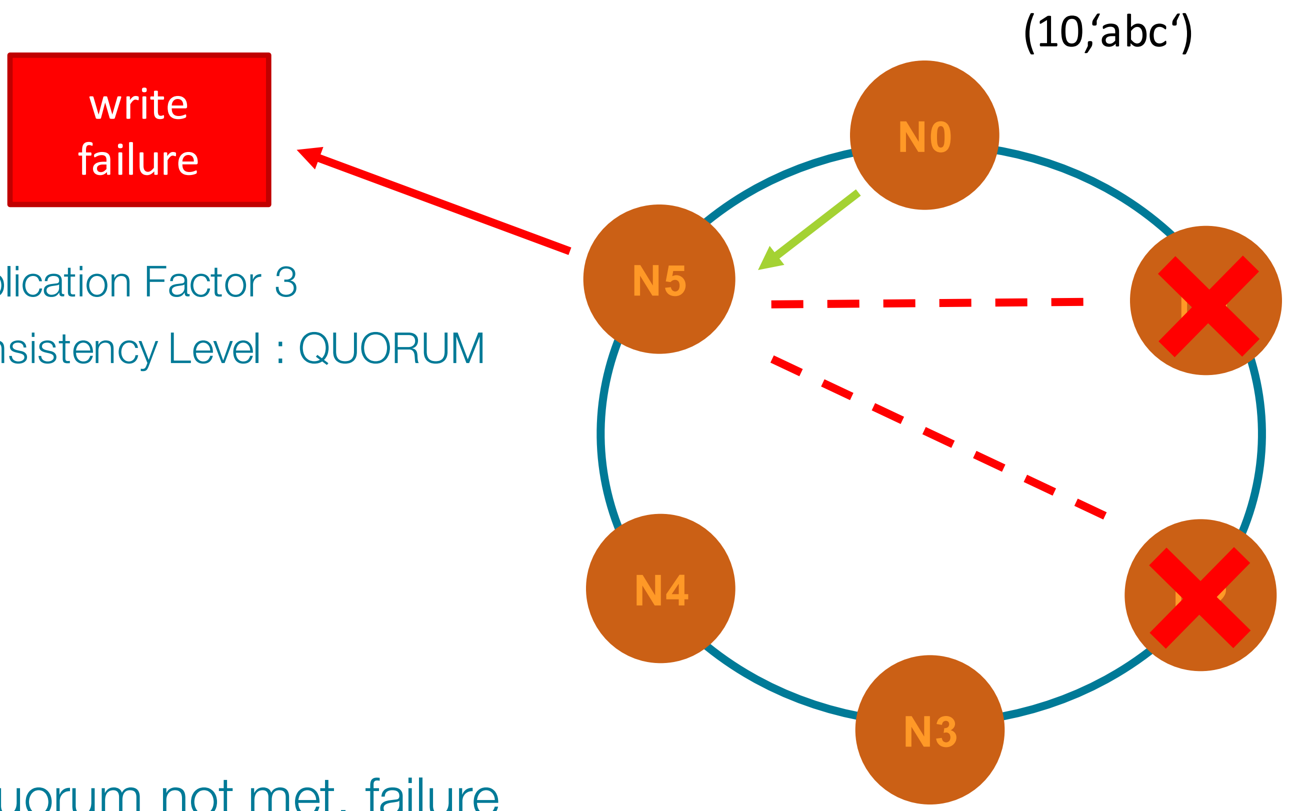


Replication Factor 3

Consistency Level : QUORUM

- Writing hints for failed node
- 3h hint write window
- CL: LOCAL_ALL will fail

Write request handling

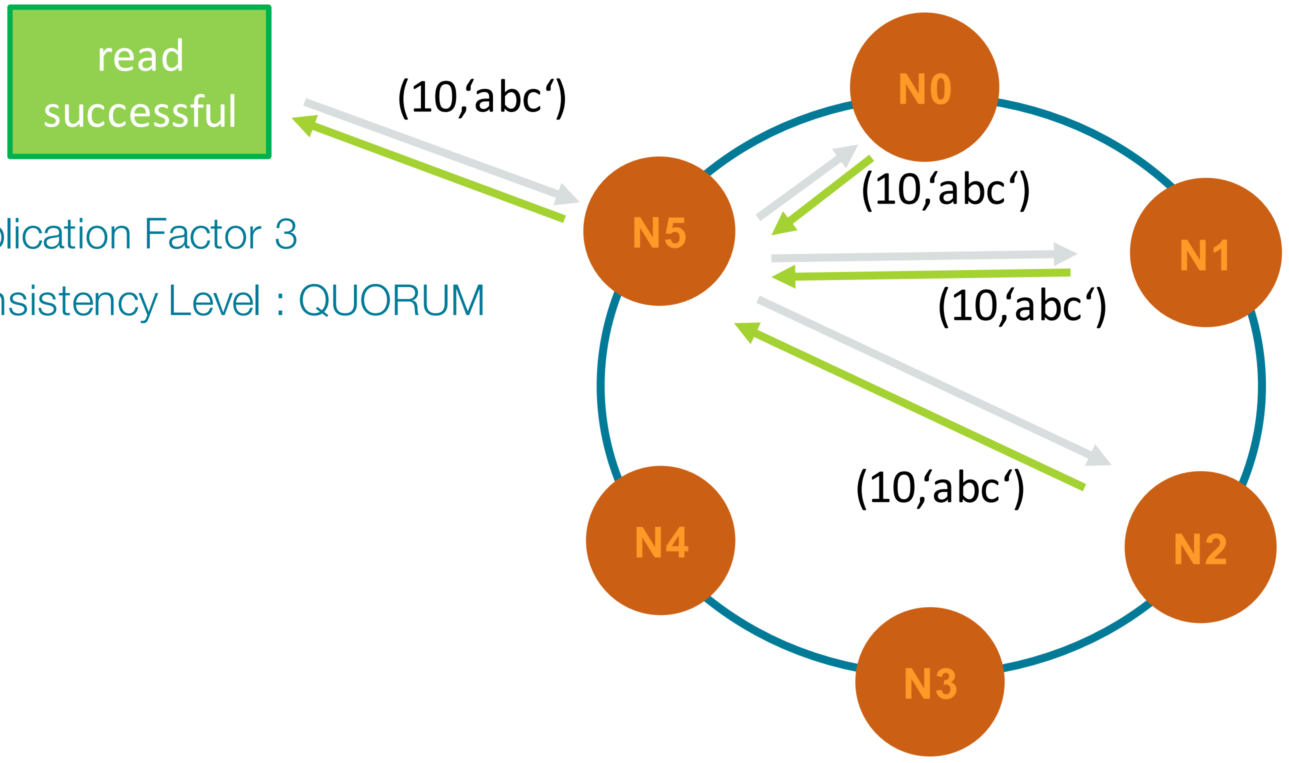


Replication Factor 3

Consistency Level : QUORUM

- quorum not met, failure
- CL: LOCAL_ONE will succeed

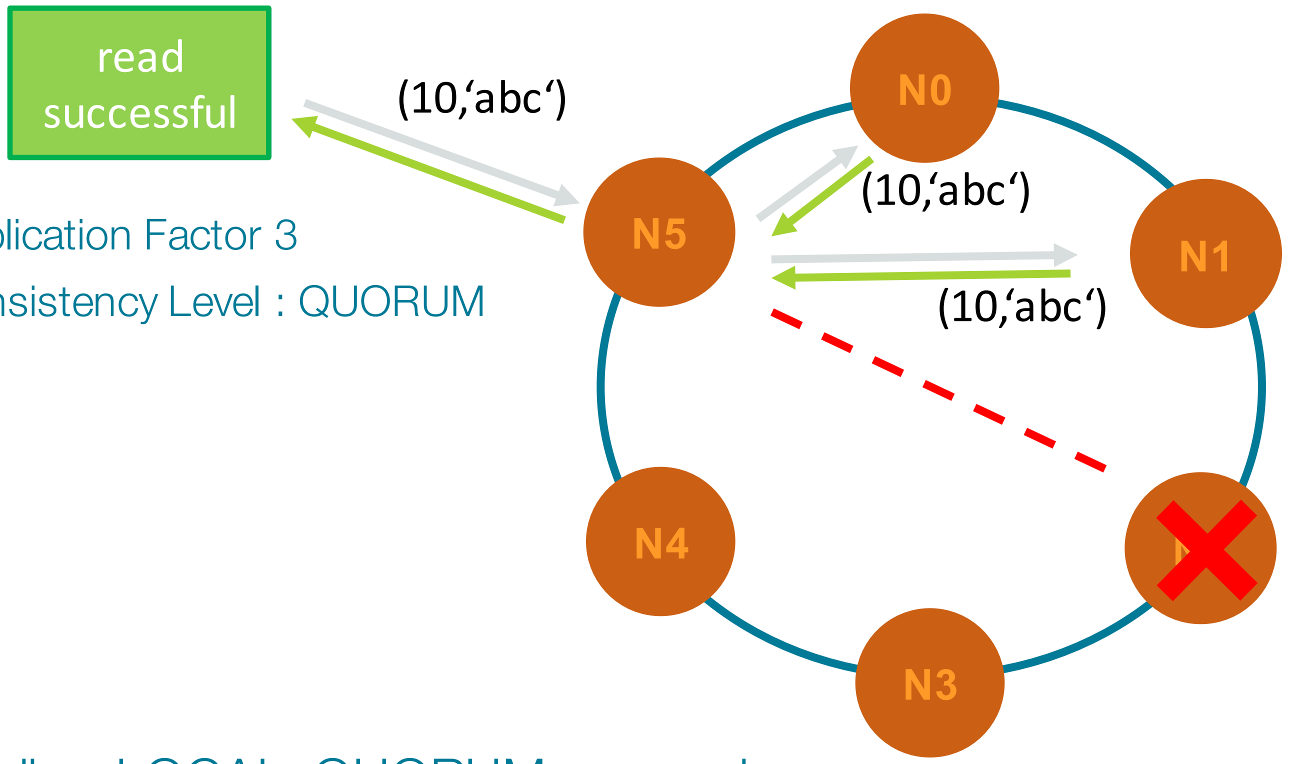
Read request handling



Replication Factor 3

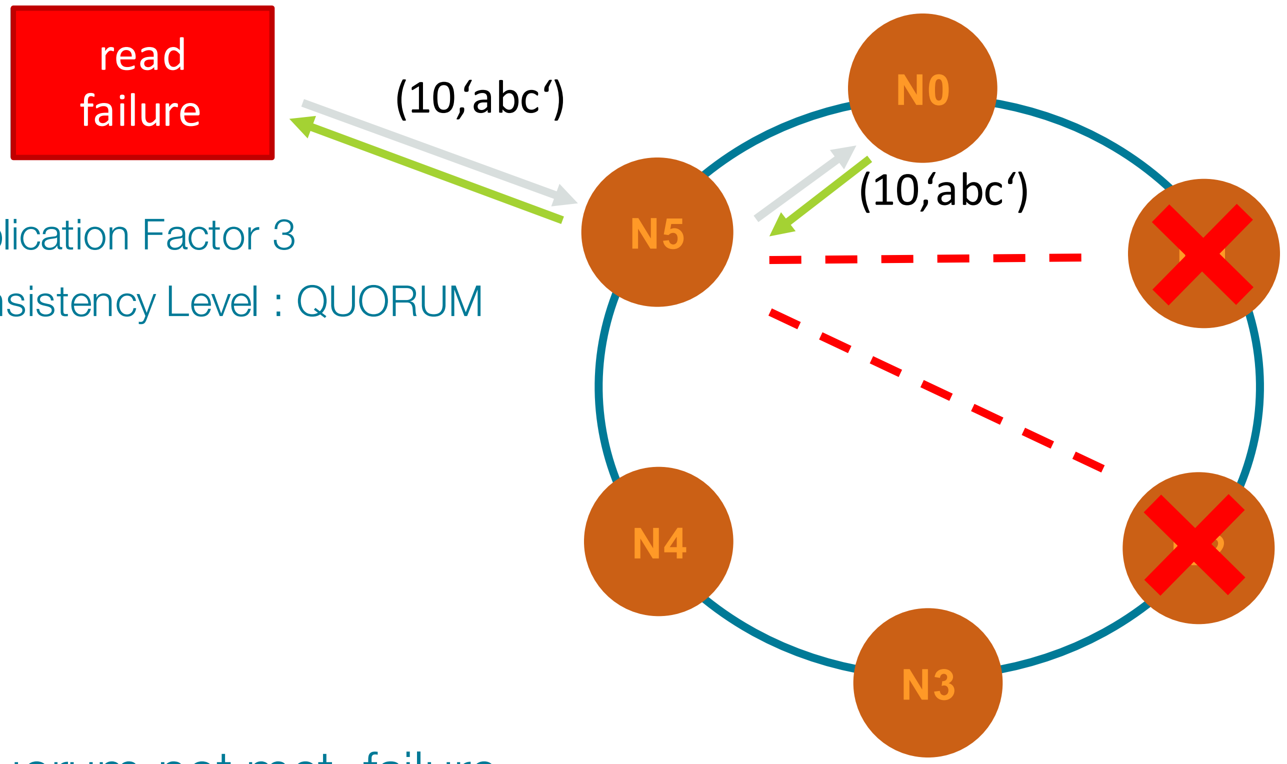
Consistency Level : QUORUM

Read request handling



- Reading LOCAL_QUORUM succeeds
  CL: LOCAL_ALL will fail

Read request handling

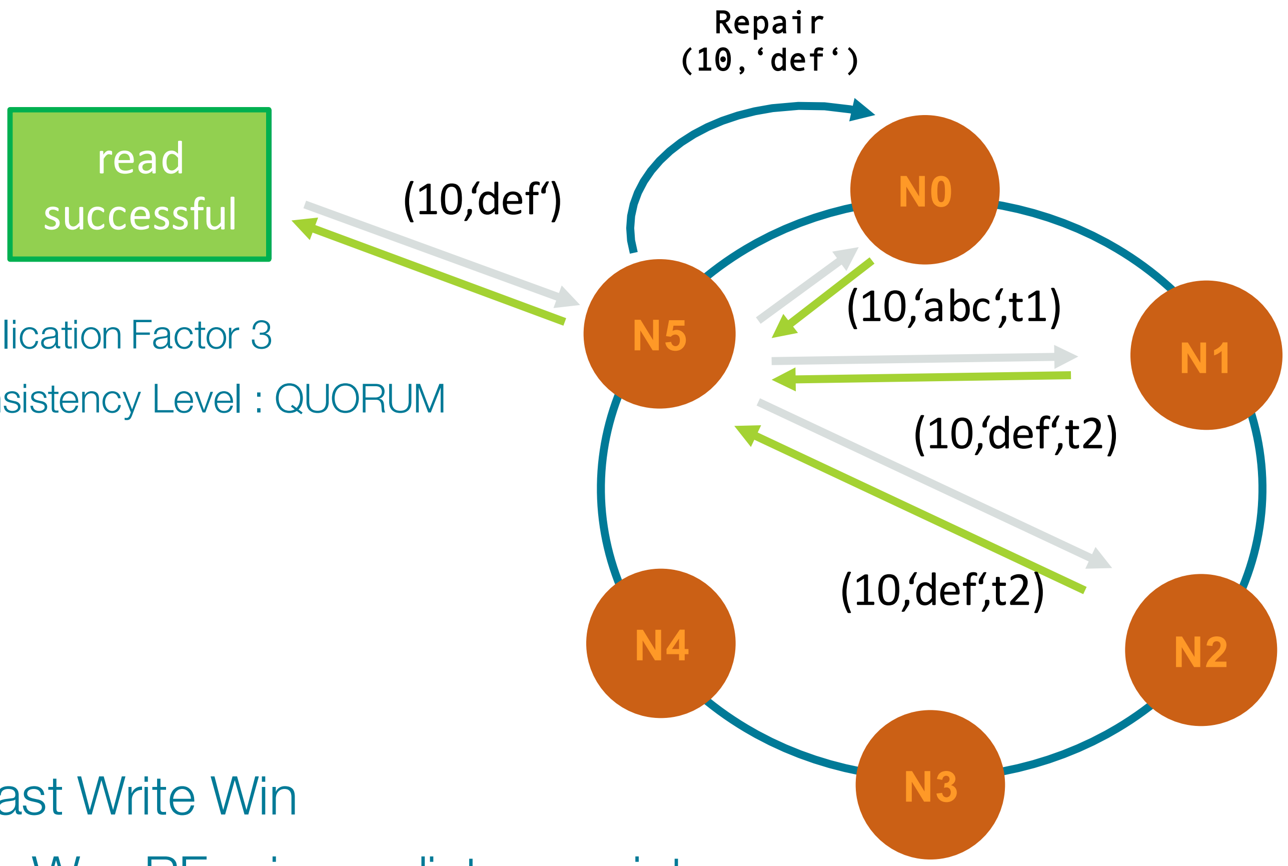


Replication Factor 3

Consistency Level : QUORUM

- quorum not met, failure
- CL: LOCAL_ONE will succeed. See more...

Read request handling – Read Repair



Replication Factor 3

Consistency Level : QUORUM

- Last Write Win
- R+W > RF = immediate consistency
- Background vs. foreground Read Repair. See more…

- Background vs. Forground Read Repair
  - Compare digests
    - If any mismatch
  - re-request to same nodes (full data set)
    - compare full data sets, send update
    - block until out-of-date replicas respond
  - Return merged data set to the client

- Consistency Level
  - one, quorum, all
  - local vs. cluster wide

Driver Code

```
Cluster cluster = Cluster.builder()
      .addContactPoint("127.0.0.1")
      .withLoadBalancingPolicy(new TokenAwarePolicy(DCAwareRoundRobinPolicy.builder()
          .withLocalDc("myLocalDC")
          .build())
      .build();


PreparedStatement prepared = session.prepare
      ( "insert into sales_by_customer(custid, salesdt) values (?, ?)”);

BoundStatement bound = prepared.bind(”1", ”20170102");

session.execute(bound);  //
```

Throws UnavailableException If consistency doesn’t met, downgrade is possible with corresponding RetryPolicy. Read More…

Take away

- Data distribution (hash, tokens)
- Data replication (RF)
- All nodes are peer nodes , master less
- Background Read Repairs
- RetryPolicy in driver

Lab 2 : Hands-on DSE CQL

Vielen Dank!

Eventuell Bootstrap, ReBalance, Num Tokens VNodes

- All nodes are peers
  - Including seed nodes
  - No master
  - Discovery through gossip

- Built-in replication
  - Simplify your architecture!