AEROSPIKE

SUMMIT ‘19

AEROSPIKE

Aerospike's Strong Consistency Mode of Operation

Piyush Gupta
Director Customer Enablement
Aerospike

Aerospike's Strong Consistency (SC) Mode

Outline

- What does it get me?
- How does it work?
- Is there a performance hit?
- How do I configure it?

Strong Consistency Mode – What does it get me?

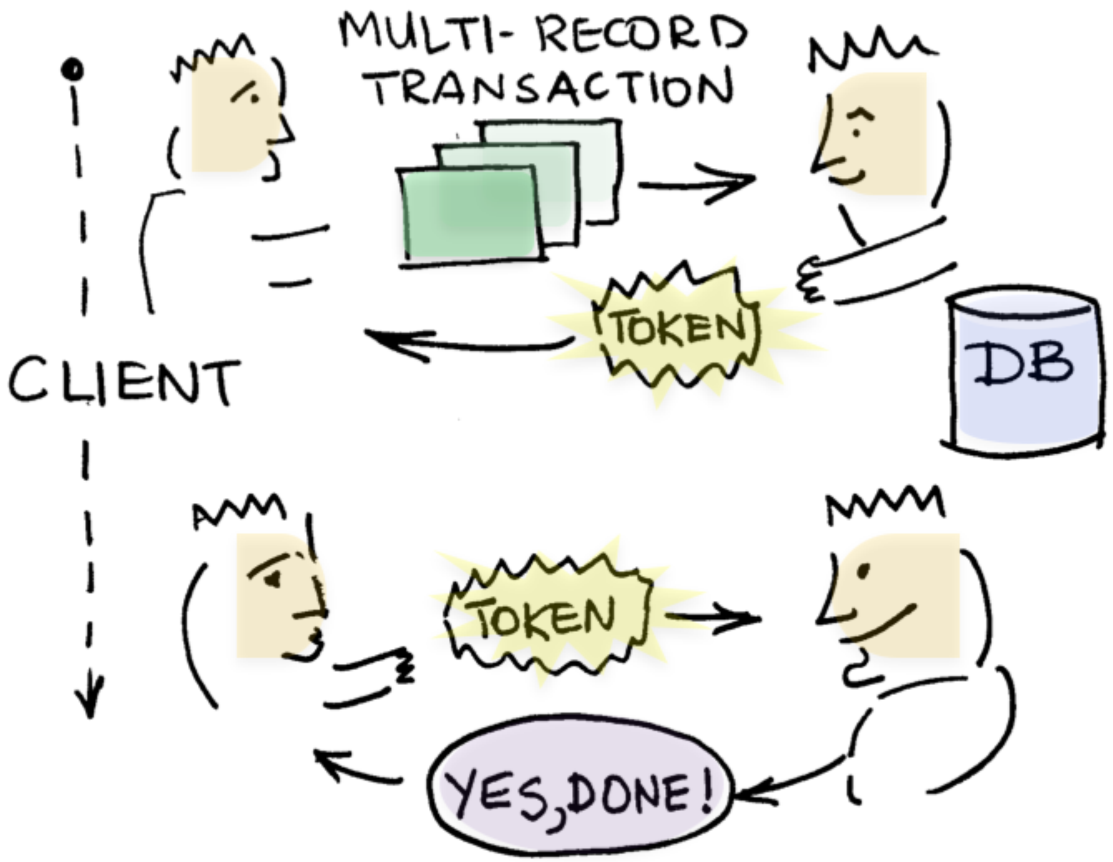
Why Strong Consistency?

Premise: I am going to use Aerospike as a System of Record then normal operations OR during any database failure modes, there should be:

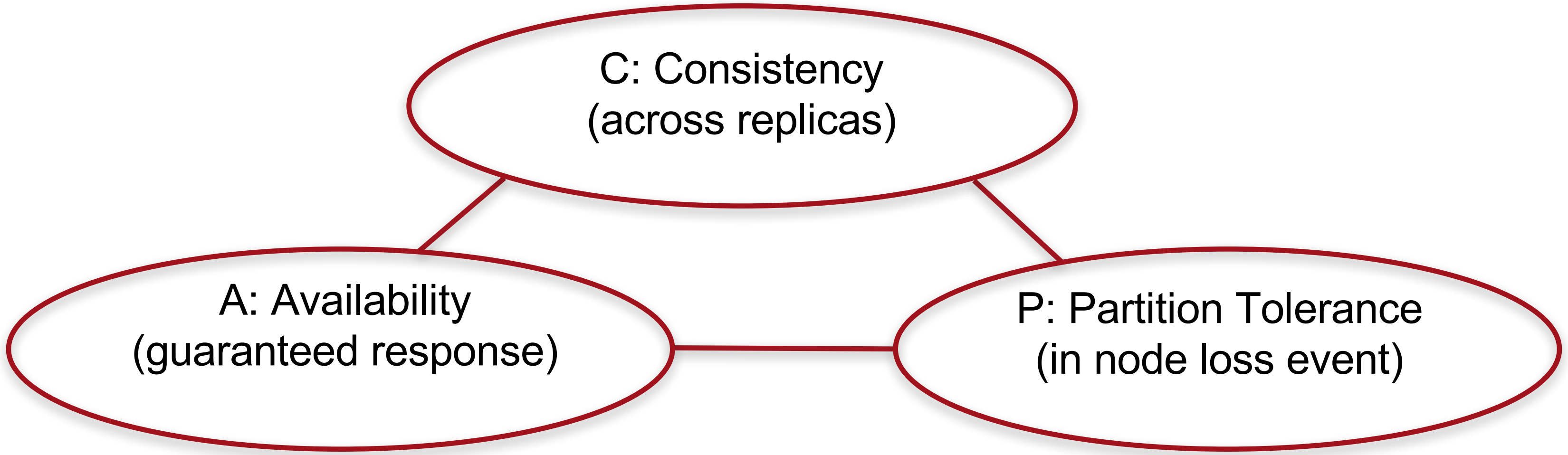
- No Stale Reads
  - Reading data that is not the most recently committed value.
- No Dirty Reads
  - Reading data that is not yet committed.
- No Data Loss
  - Losing data that was committed.

How Relational Databases Address Strong Consistency

- If on single node storage, obviates need for dealing with partition tolerance.
  - Single copy of data.
- Two Phase Commit.
  - Allows rollbacks and multi-record transactions but is slow, very hard to scale up.



CAP Theorem for Distributed Databases



➢ You can only design to guarantee two of the three.

- **CA** – High consistency and availability, *nodes never go down*. (Not a realistic scenario!)

- **Practical Definition:** In case of a node loss, choose between C or A in your distributed database design.
- **AP** – High availability on a cluster of nodes, accept Inconsistency or Eventual Consistency.
- **CP** – Strong consistency and cluster scalability, accept some unavailability during node failure/addition.

Professor Eric Brewer, UC Berkeley, explains the CAP theorem

Strong Consistency Challenge with Distributed DBs

- Distributed databases replicate data to prevent data loss on node failure.

- 3 Copy Consensus based designs with flaky clients can result in:
  - Dirty Reads (client writes to one node only and crashes).
  - Blocked Reads – no consensus (client updates/crashes - different nodes with different values).
  - Blocked Updates (update requiring a read, and, reads are blocked due to consensus failure).

- Distributed Master AP designs can suffer data inconsistency:
  - Under "Split Brain" situations.
  - Clock skew, outages, merges can lead to written data lost even after a read.

Split Brain – It Happens!

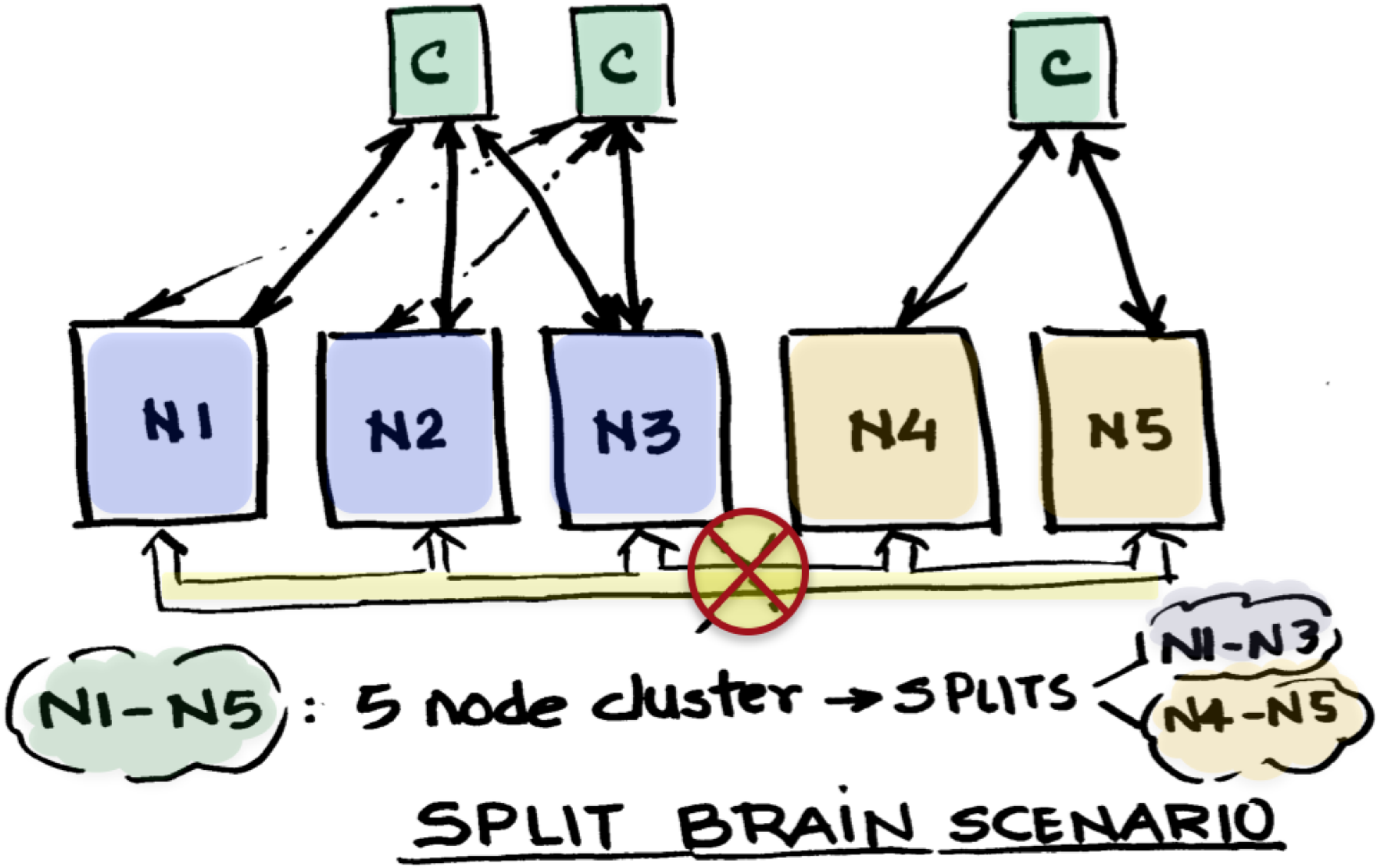
- Split Brain for the purposes of our discussion is defined as a scenario where a cluster splits into two or more separate clusters, each thinking it is **the** cluster.

In AP mode, we get:

- *Availability* at the expense of *Consistency*.

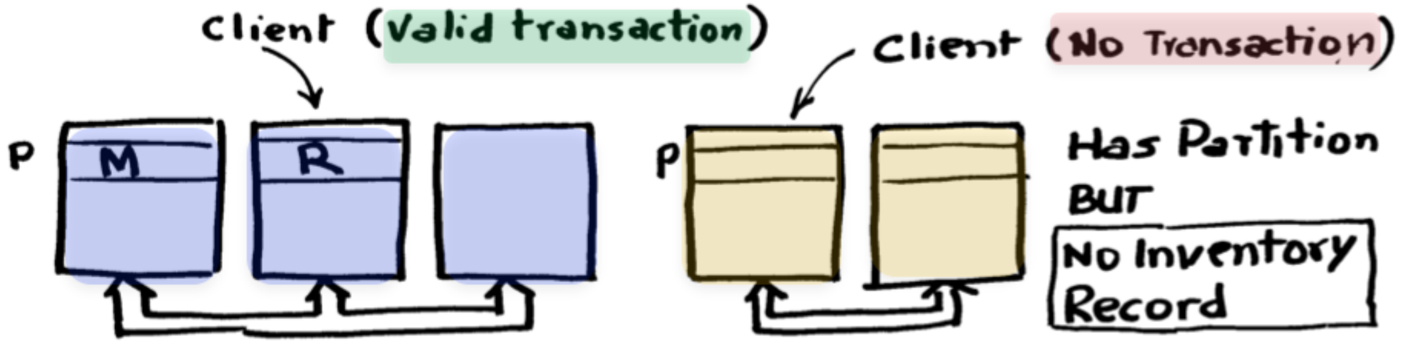
In SC mode, we are rigorous about:

- On the fly cluster changes – nodes going in or out of the cluster.

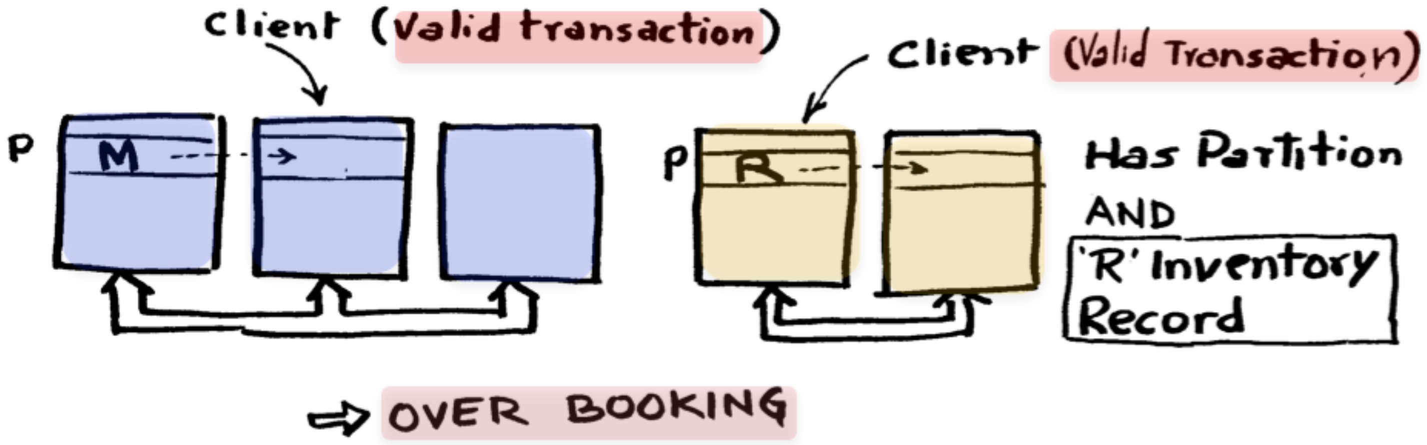


Split Brain AP Case – Inventory Transaction Example

- Case 1: Master & Replica on same split, other split will not have inventory record ➔ Inventory data not found on read but can be added (written).



- Case 2: Master & Replica on split clusters lead to data duplication and create valid over-bookings ➔ Inconsistency.



Aerospike's Strong Consistency Mode for Single Record Transactions

- Configured on a namespace basis.
- Cluster membership is identified as a "roster" of nodes.
- Strong Consistency Mode guarantees:
  - Successful writes or updates are never lost.
  - No Dirty Reads.
  - Client can do 'Linearizable' Reads on a per read basis.
    - With 'linearizable', all clients see the same sequence of updates.
  - Immediate Consistency.
  - Guaranteed to handle up to 27 seconds clock skew between nodes.
    - Implies client never has to merge data or deal with conflicts.

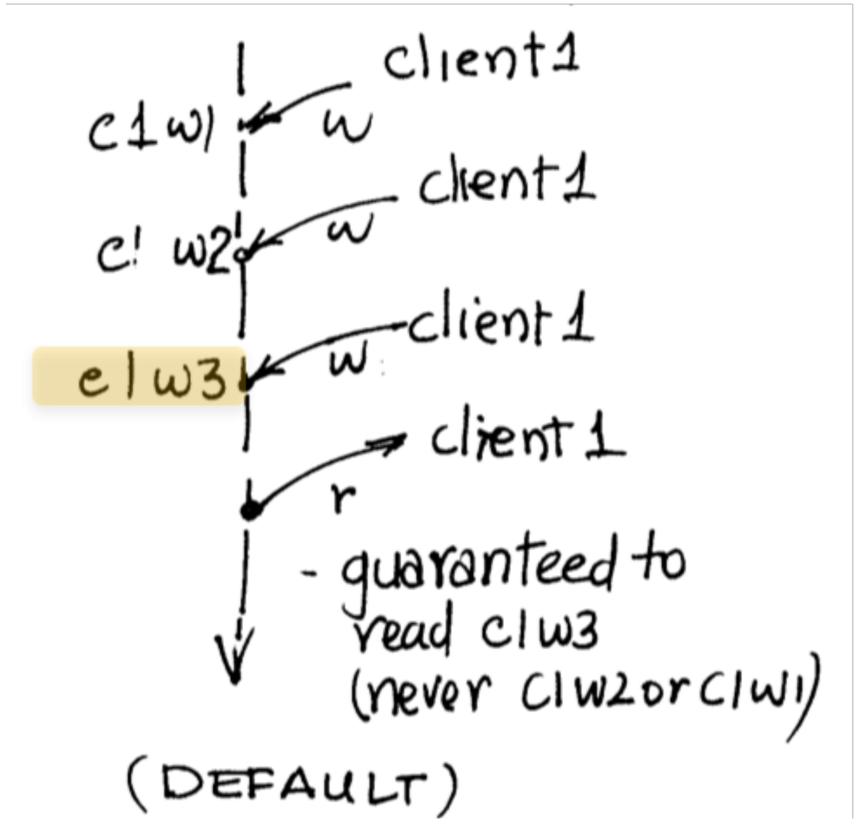
- Aerospike's Strong Consistency Mode passes Jepsen Tests!

Read Linearizability in Strong Consistency Mode

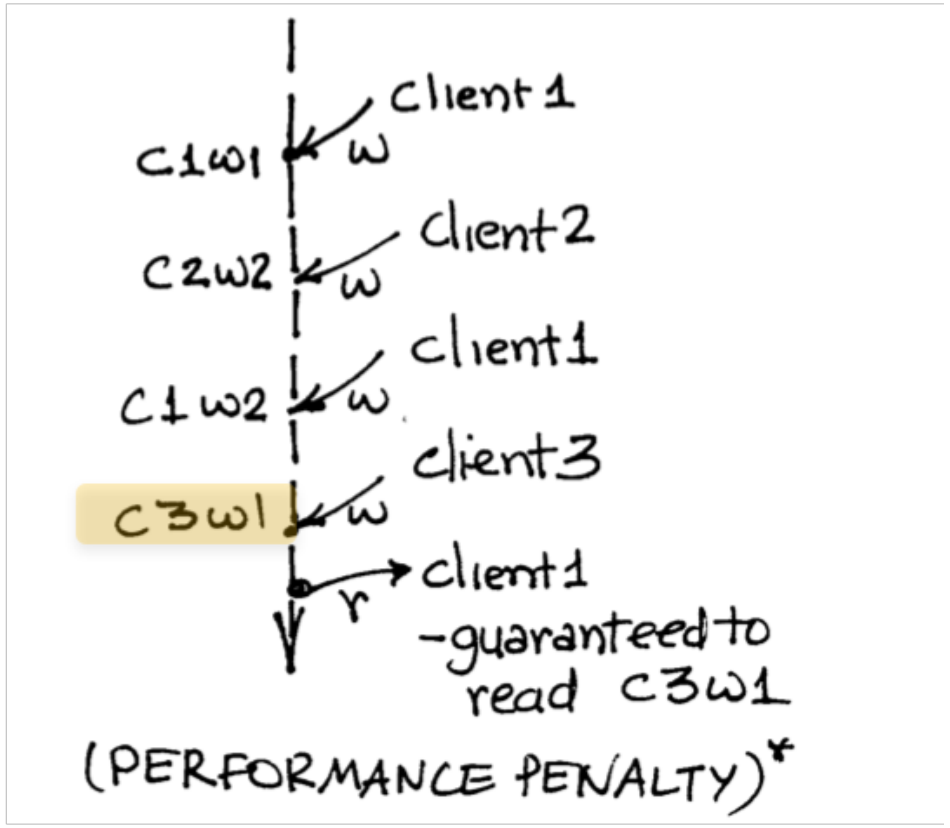
In SC Mode, reads can be configured to never return "stale" reads.

- Session Consistency is the default.
- Global Read Linearizability can be enforced on a per read transaction basis.

Session Consistency

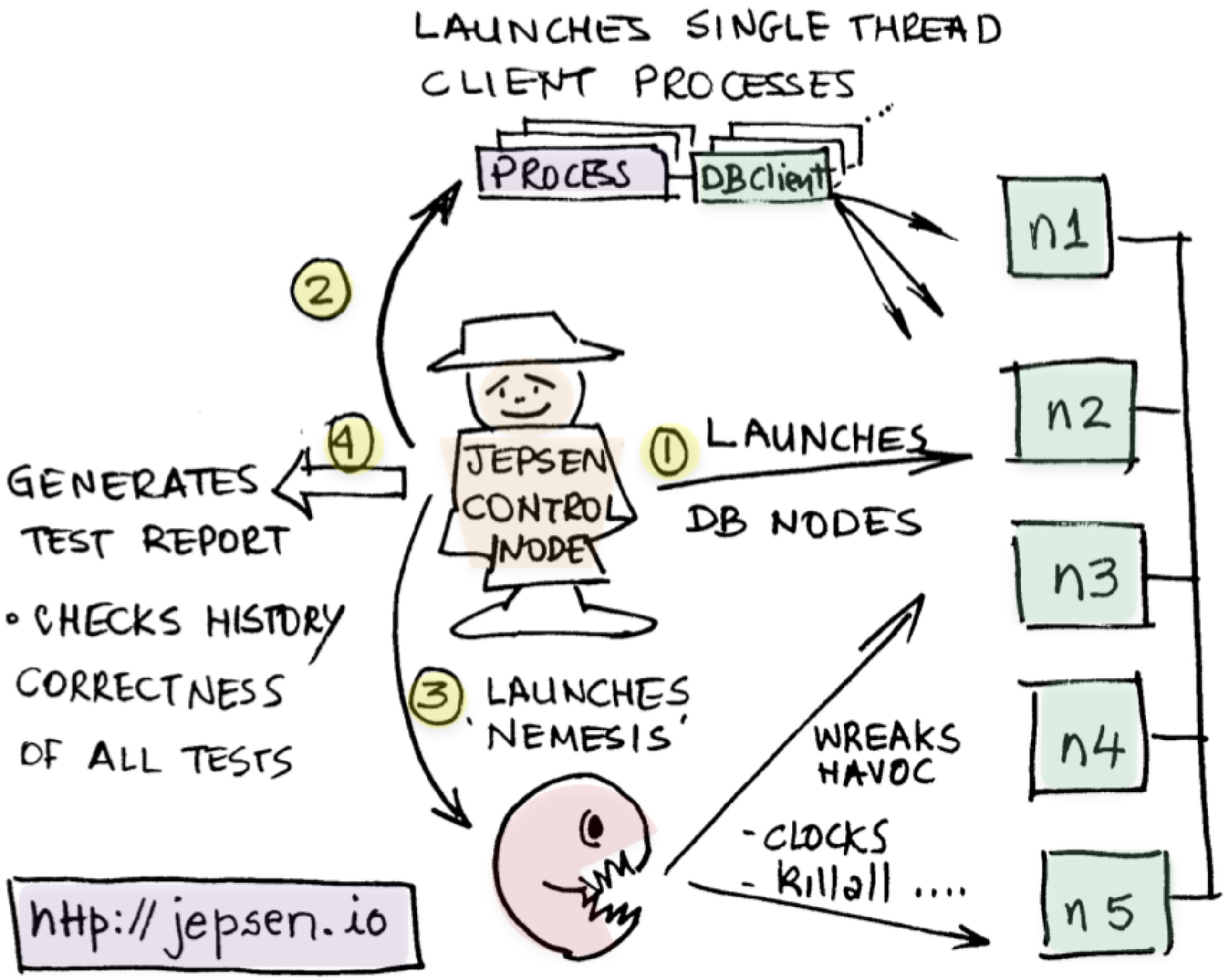


Global Linearizability



What are Jepsen Tests?

- Conducted by http://jepsen.io to find consistency holes in distributed databases.

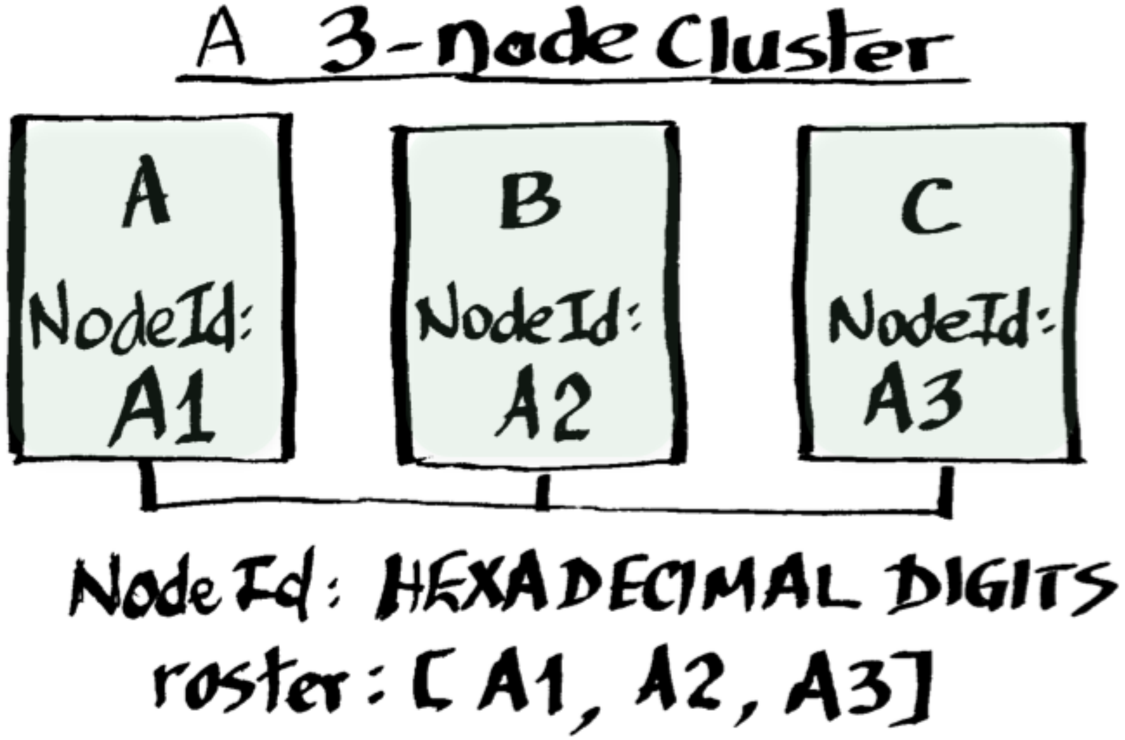


Strong Consistency Mode – How does it work?

Definition of Terms: Cluster vs Roster

Mesh or Multicast – nodes find each other and form a cluster.

- Nodes have Node-IDs.
- Roster: A set of nodes that define a SC Mode namespace.
- Nodes added subsequently to the cluster will not automatically extend an SC namespace without operator intervention. (Must update roster and "recluster:")



Identifying a Roster

Configure on each node:

```
service {
node-id A1   (must use 0-9, A-F only.  i.e. Base 16 digits. )
...
}

namespace ns1 {
replication-factor 2
strong-consistency true
default-ttl 0
...
}
```

Identifying a Roster

Identify the roster for namespace ns1:

```
$ asadm
```

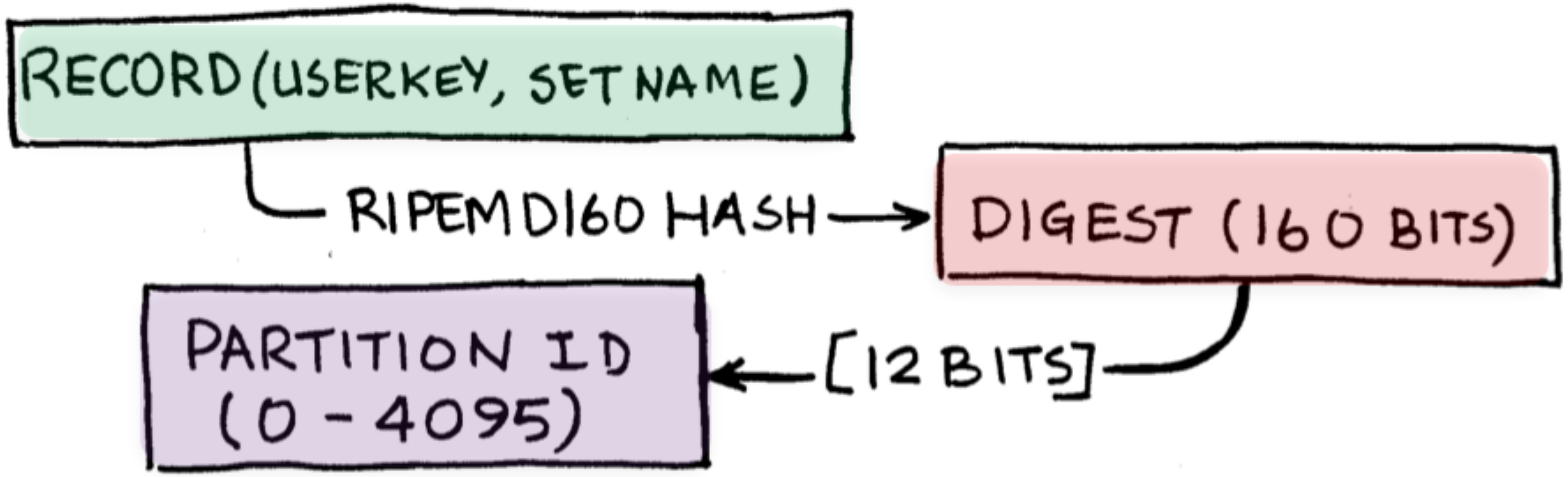
```
Admin> asinfo -v "roster:namespace=ns1"
Admin> asinfo –v "roster-set:namespace=ns1;nodes=A1,A2,A3"
```

Elevate a pending_roster to roster:

```
Admin> asinfo –v "recluster:namespace=ns1"
```

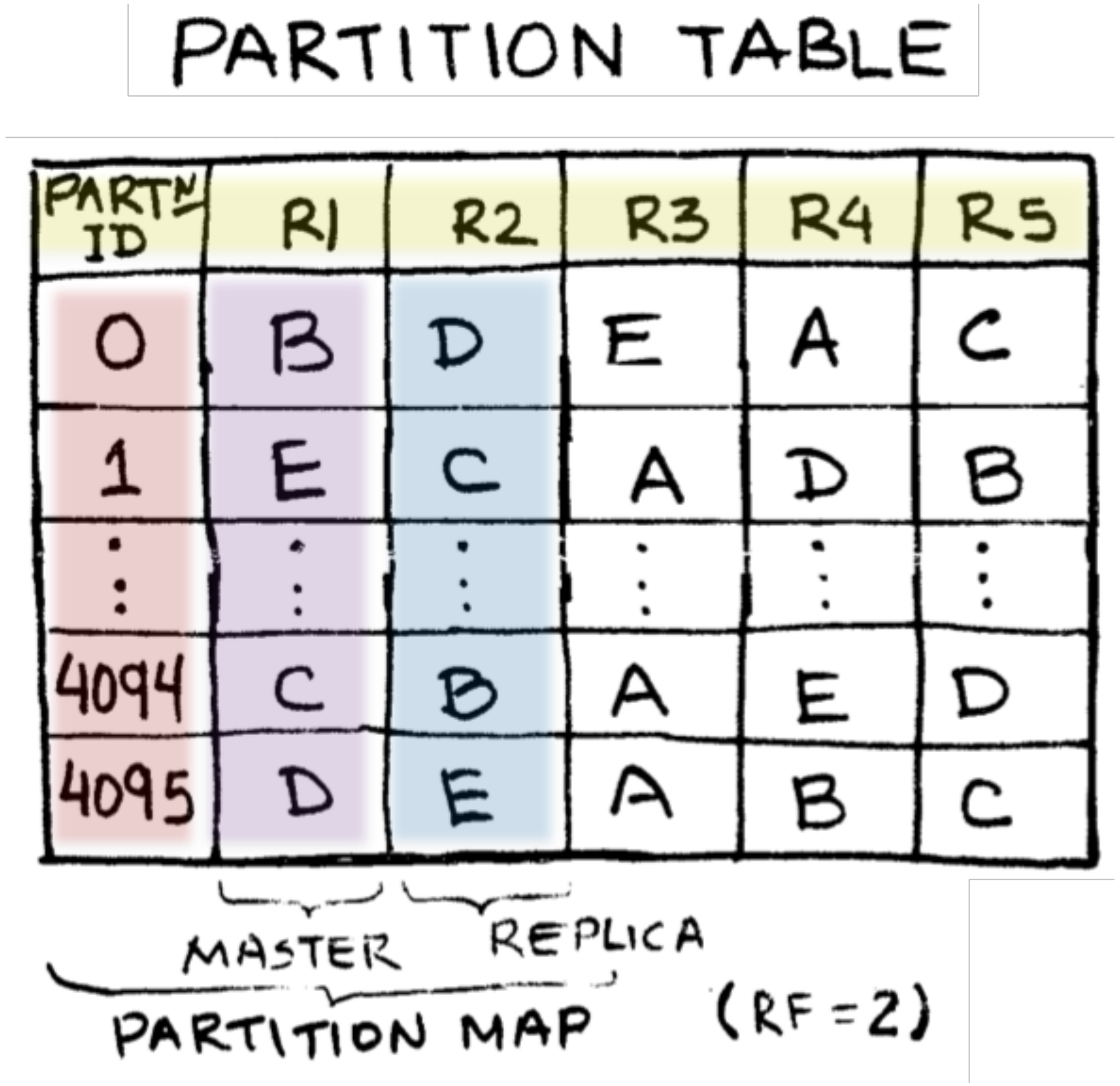
Recap: Record Distribution and Partitions

- Set name + key type(1byte) + User key is hashed into a 20 byte value (40 Hexadecimal digits) using the RIPEMD160 hash function.
- This hash + additional data (fixed 64 bytes) are stored in Primary Index in RAM.
- 12 bits of this hash are used to compute the partition id. 2^{12}=4096.



Recap: Record Distribution and Partition Map

- 4096 partitions per namespace. Each namespace has its own Partition Map.
- Partition Map maps Partition id to Node id based on cluster membership.
- Partition map generated when cluster forms or changes.
- A namespace, set name & user key locate a record to a partition on a node.
- **In SC Mode, a Partition 'Q' may be:**
- **ACTIVE, UNAVAILABLE or DEAD.**



PARTITION TABLE

PARTN ID	R1	R2	R3	R4	R5
0	B	D	E	A	C
1	E	C	A	D	B
⋮	⋮	⋮	⋮	⋮	⋮
4094	C	B	A	E	D
4095	D	E	A	B	C

MASTER
REPLICA
PARTITION MAP
(RF = 2)

SC Mode Terms: Roster-Master, Roster-Replica, Master, Replica

With all nodes defined in the roster up and running, we have:

Roster Master `(rm)` Designated master node for a partition in the roster.

Roster Replica `(rr)` Designated replica node(s) for a partition in the roster.

- One (RF=2) or more (RF>2) nodes that are identified as the Replicas for a Partition.

- `rm` and `rr` designations do not change, even if node goes down, until the operator changes the roster & "reclusters". **(Different from AP mode where cluster changes are dynamic.)**

Master `(m)` A Node currently the working Master for a Partition.

e.g.: A "full" Roster Replica that got promoted to Master on a cluster split.

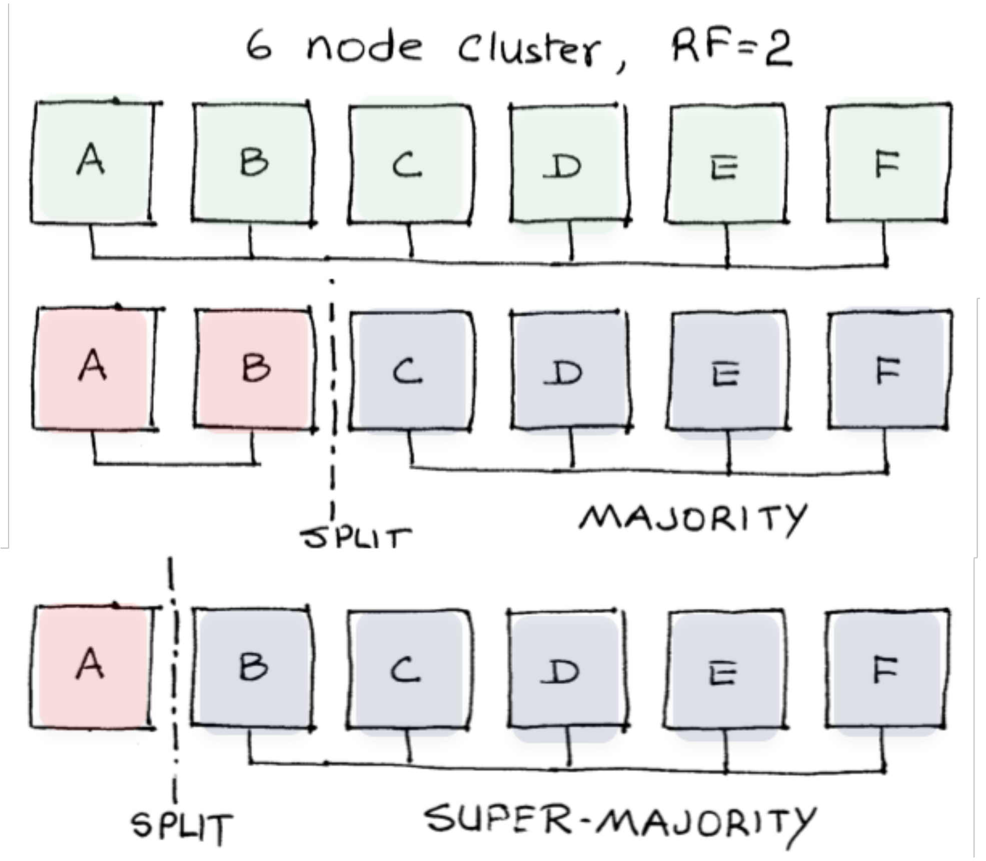
Replica `(r)` A node currently the working Replica for a Partition.

e.g.: A node that got designated Replica due to a cluster split (receives copy of partition from Roster-Master or current Master on rebalance).

SC Mode Terms: Strict Majority vs Super-majority

If a cluster splits into sub-clusters:

- Strict Majority sub-cluster: **Number of nodes in sub-cluster > (Roster Size / 2).**
- Super-Majority sub-cluster: **Less than "RF # of nodes" are down or split.**
  - #nodes present should be >RF
  - A super-majority sub-cluster has the full data, all partitions.
  - So, if RF>1, under Rolling upgrade we will always have a super-majority sub-cluster.



Partition State: Full Partition vs Subset Partition

A node is said to have a:

- Full Partition: `(f)` It has a full copy of the data for that partition (prior to a split, normal operating cluster, fully rebalanced).

- Subset Partition: `(s)`Node may be either a master or replica of a partition *but only has a subset of the data*. Needs to receive the rest of the data from rebalance in progress or needs to update its data from Duplicate Resolution (situation discussed later), in progress.

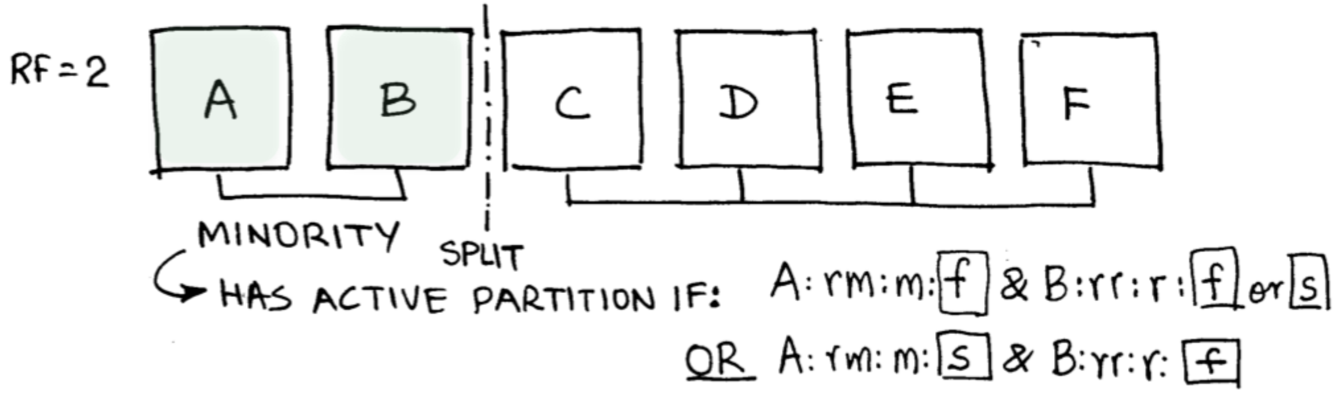
- Rule: A node that leaves the cluster (even if it was at "full" status), always rejoins the cluster with "subset" status. After rebalancing is complete, it can regain "full" status for that partition.

Next, Partition Availability: a) ACTIVE, b) UNAVAILABLE and c) DEAD.

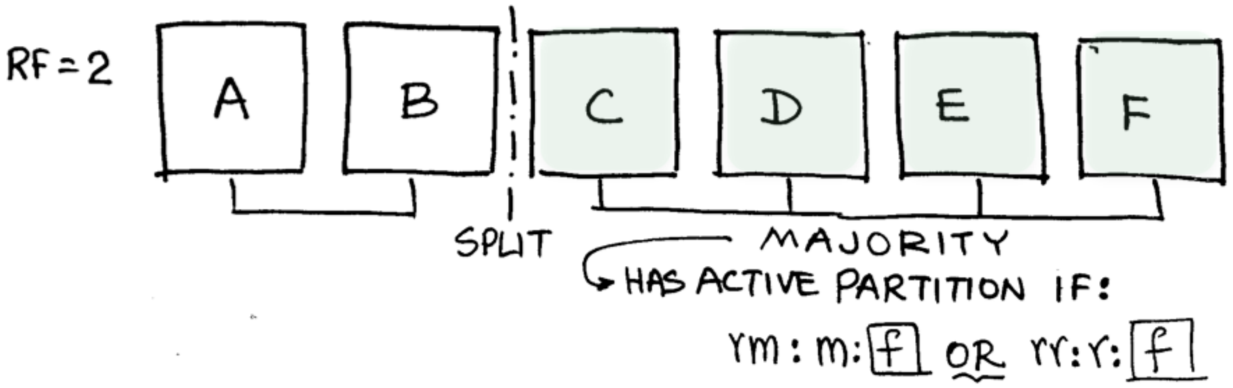
Active Partition in Sub-Cluster: Rules

Partition in a Sub-Cluster is considered Active for reads and writes if:

- Sub-Cluster (regardless of size) has all Roster-master & Roster-replica(s) and at least one is "full". (Rule 1)

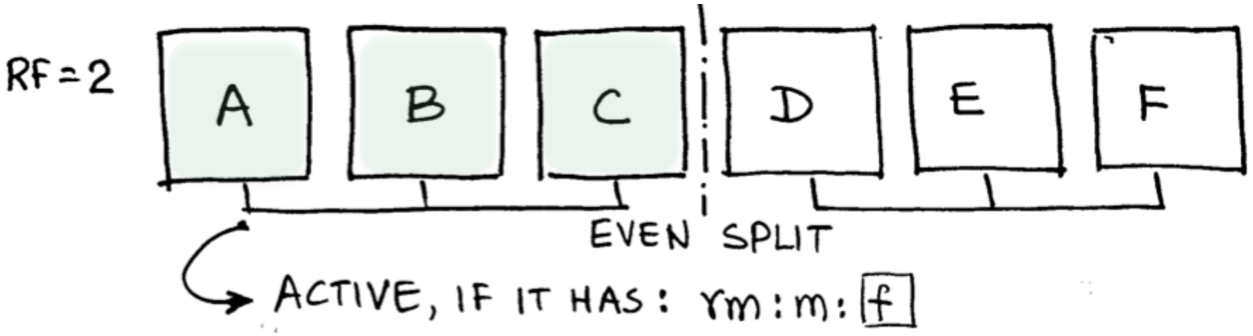


- **OR** Sub-Cluster has a strict majority of nodes & at least one "full" of Roster-master or Roster-replica(s). (Rule 2a)

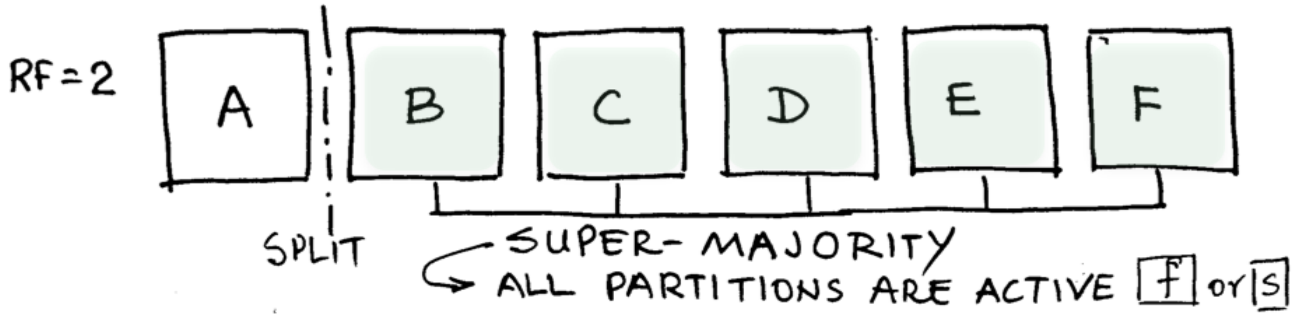


Active Partition in Sub-Cluster: Rules (cont.)

- **OR** If cluster splits evenly (not a strict majority) into two sub-clusters, the sub-cluster holding the Roster-master *and* it must be "full" has the active partition. (Rule 2b)



- OR Super-Majority Cluster – "full" or "subset" – all partitions are active.
  - Rolling Upgrade is Super-Majority. (Rule 3)



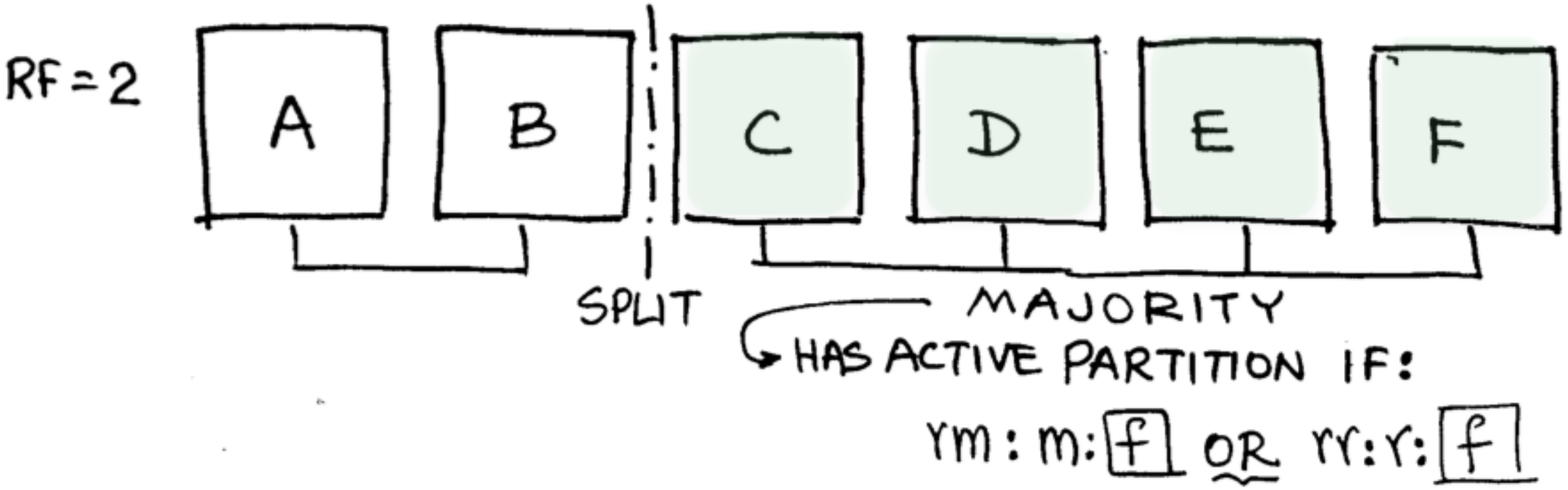
➔There can be only one sub-cluster that has the partition master at any given time.

➔Reads and Writes succeed only if the sub-cluster has at least RF number of nodes.

"Unavailable" Partitions

A Partition is considered "Unavailable" in a Sub-Cluster if:

- It is not Active in that Sub-Cluster
  - For e.g., if C-D-E-F have rm:m:f of Partition Q, then Q is unavailable in A-B



- If greater than or equal to "RF # of nodes" go out, with persistent storage, some partitions will become unavailable till those nodes rejoin the cluster. With data-in-memory only, you will lose the partition data. ➔ In SC Mode, using in-memory only namespaces is not recommended.

"Dead" Partitions

A Partition is considered "Dead" if the partition is not available with full roster of nodes up:

- Some or all of underlying record storage is detected as lost while all nodes are still running and connected. This can happen if:
  - In storage-engine-memory, node(s) restarted under certain conditions. e.g.:: restart >= RF # of nodes simultaneously.
  - One or more SSD drives storing the data, fail or are erased under certain states of the cluster.

➔Once a partition is dead, cluster must be revived (operator intervention).

```
Admin> asinfo –v "revive:namespace=ns1"
Admin> asinfo –v "recluster:"
```

Note: When you revive a dead partition, you may or may not have lost data.

Transactions in SC Mode

Writes in SC Mode

SC Mode Promise: Writes reported as "succeeded" to client are never lost by the server.

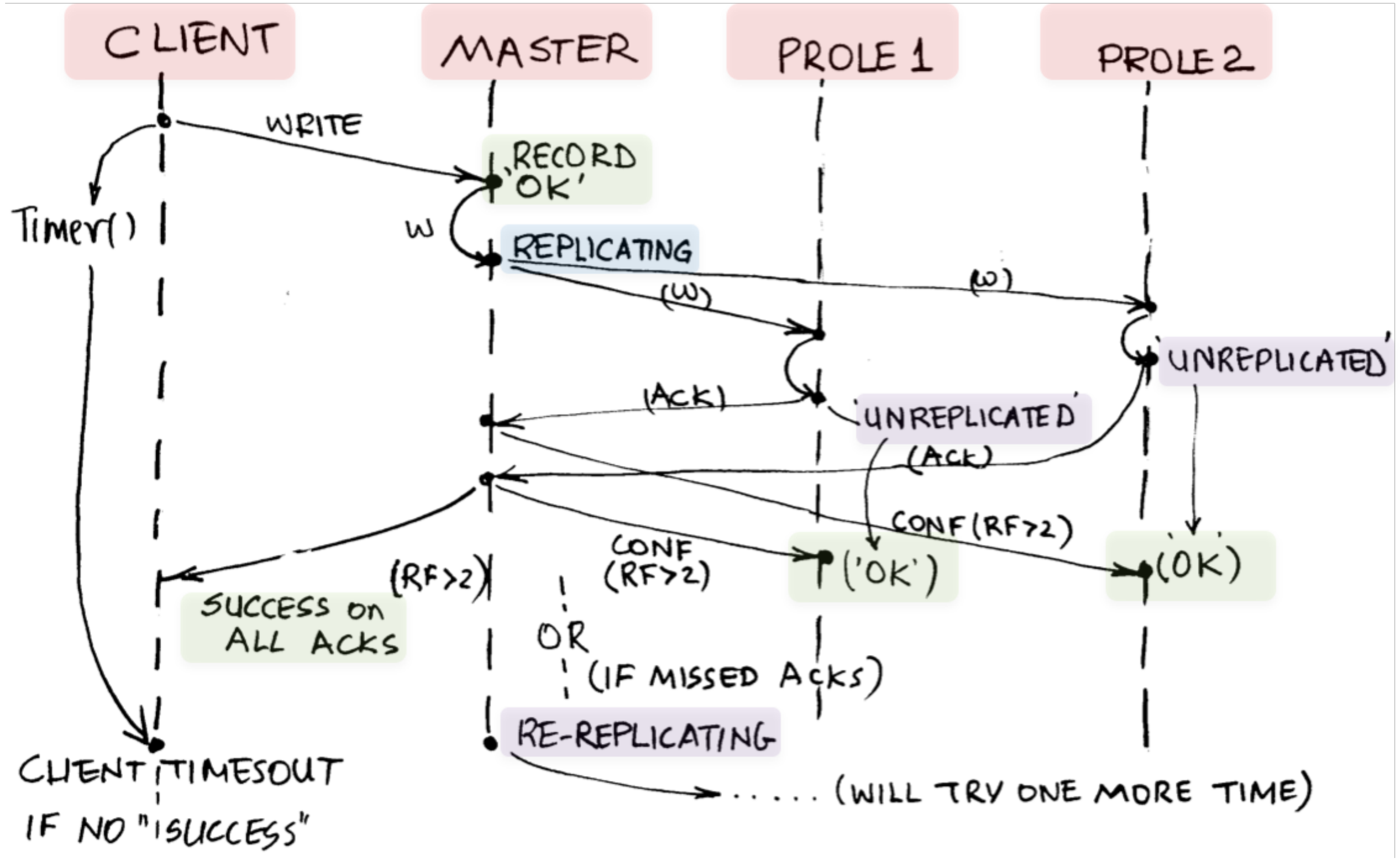
- Only one sub-cluster will take writes – single master for a partition.
- Writes must be acknowledged by all replicas before master acks to client.
- Master acks back to replicas also if RF>2. (Redundant for RF=2).
  - Huge savings on future Duplicate Resolution, if needed.

Record State in SC Mode

- **2 bits in Primary Index** store 4 possible states of a record.
- OK (normal state, *replicated* to RF number of nodes).
- Replicating (State on Master, in the process of writing to Proles).
- Re-Replicating (State on Master, if Acks not received from all Proles).
- Un-Replicated (State on Master if Replication fails, on Prole if RF>2 and Confirmation not received).

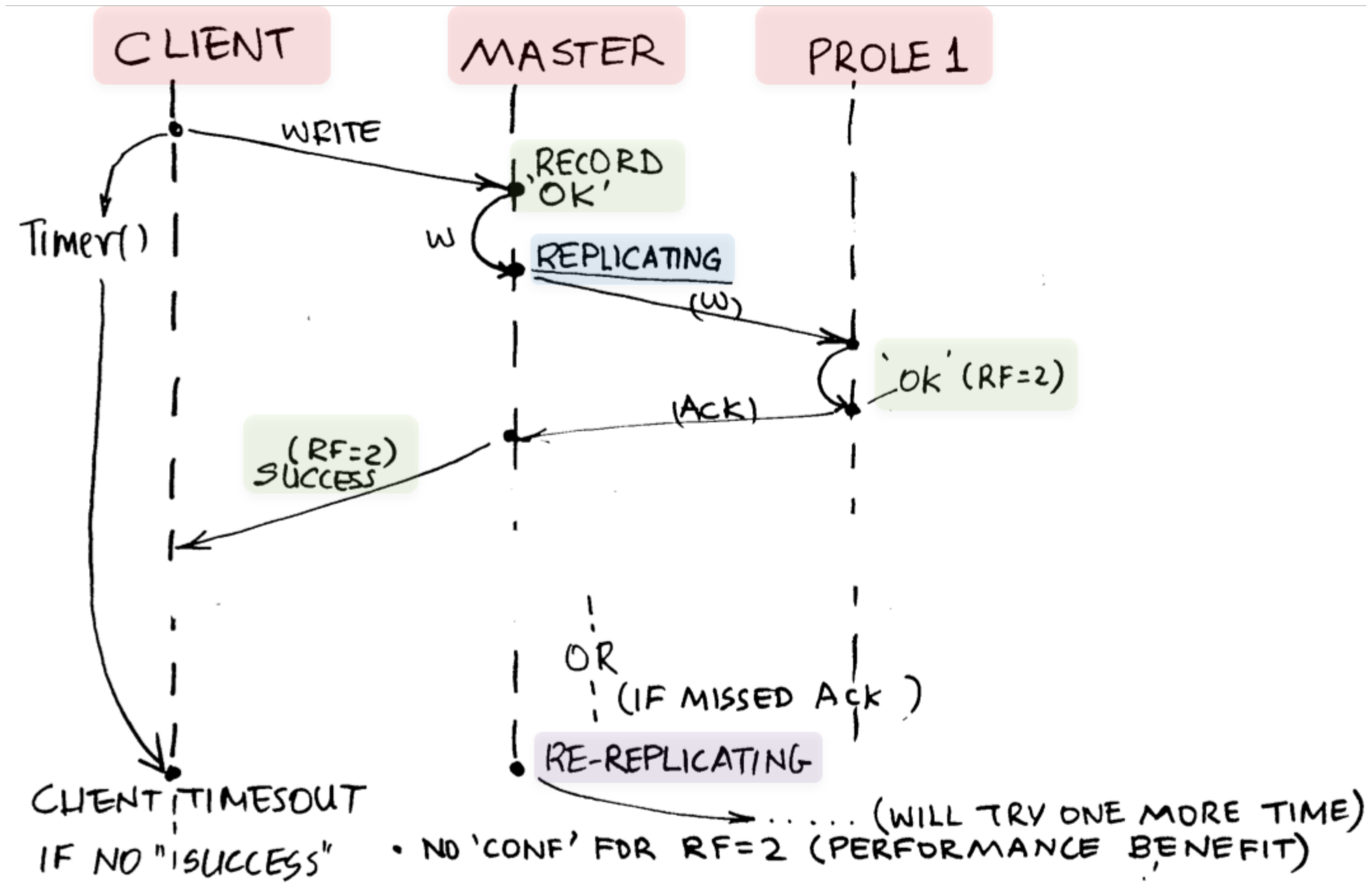
- Let's look at a write transaction for: a) RF>2 vs b) RF = 2 .
- Here, starting state of record on Master is 'OK'.

Write Transaction in SC Mode (RF>2)



- Client TIMEOUT if SUCCESS not received from server.
- Client's view on TIMEOUT of Record status on server – UNKNOWN. Could be on Master or not, OR on Master and Prole1, OR on Master and All Proles...

Write Transaction in SC Mode (RF=2)



- Client TIMEOUT if SUCCESS not received from server.
- Client's view on TIMEOUT of Record status on server – UNKNOWN. Could be on Master or not, OR on Master and Prole...
- Note: No "CONF" messages required for RF=2 (Performance Benefit – fewer fabric msg passing.)

Durability in SC Mode

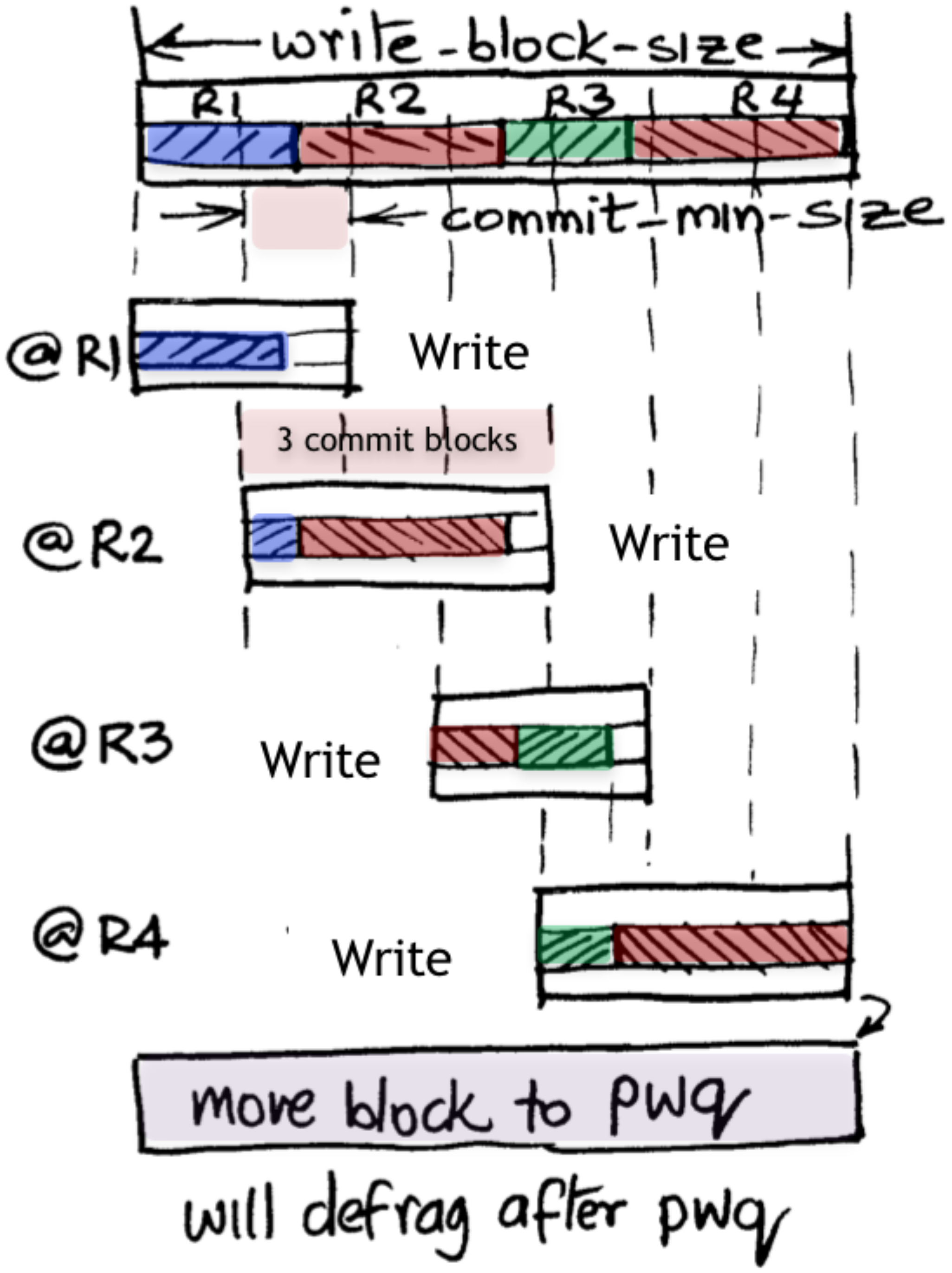
Durability in Strong Consistency Mode

commit-to-device: true Flush to disk before ack to client. (optional)

commit-min-size: (default selected by Aerospike) Set to a power of 2.

- Only applicable if commit-to-device is true.

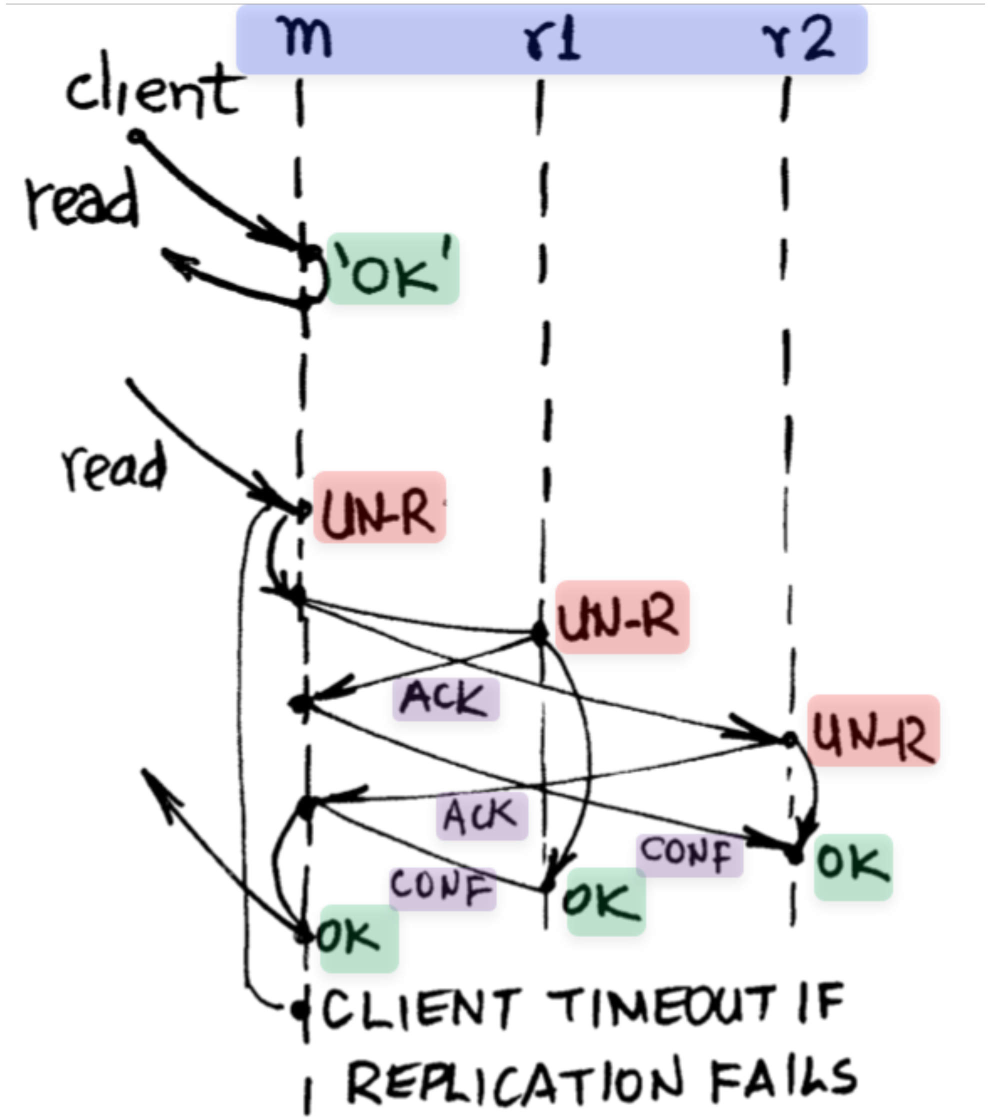
- Records are written to persistent storage on every write at commit-min-size boundaries.
- Commit-to-device ensures that no successful write is ever lost even if master and replica both go down simultaneously (within *flush-max-ms* worst case) before the write-block-buffer has flushed to device.



No Dirty Reads in SC Mode

What are Dirty Reads? Reading data that is not yet committed.

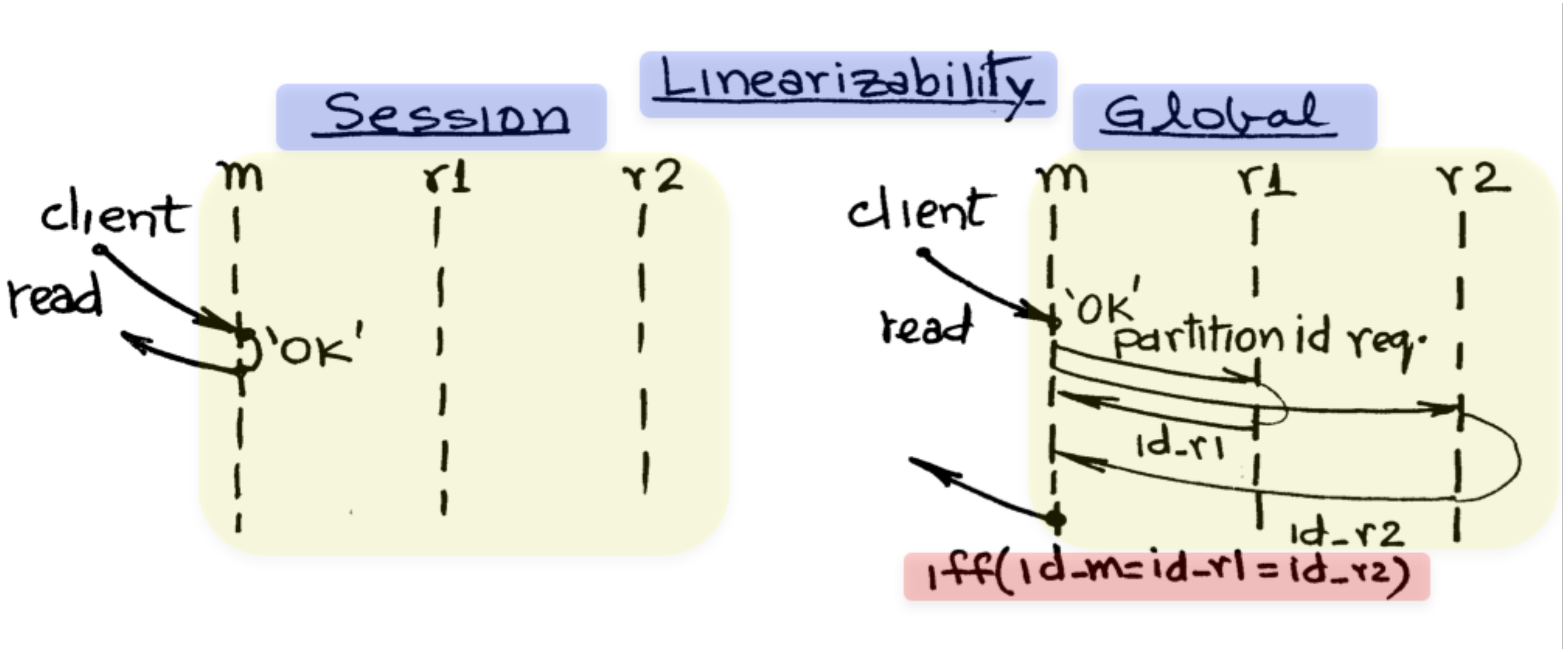
- Reads go to Partition Master
- Normal Cluster, steady state.
  - **Case 1:** Record state = "OK" on master, master returns record to client.
  - **Case 2:** Record state = "UNREPLICATED", master will re-try replication first. If success, return record to client or client will TIMEOUT.



Global Linearized Reads in SC Mode

Global Linearized Read Mode

- No stale reads, requires extra network round-trip (master requests 32 bit partition regime from each replica and compares with its regime value).



In what situation can replica partitions differ from master?

- Split brain heals where all but the master node formed a new cluster but the master node had not observed the split.

Relaxed Reads in Strong Consistency Mode - Ver 4.5.2.1+.

Need For Relaxed Consistency Reads

- Recall, in SC Mode, default session consistent reads always read from Partition Master.

1) Customers using SC Mode with Rack Awareness, 2 racks in separate AZs:
   Client should be able to read from Replica on preferred Rack identified in client.

- This saves on Network Access Costs on Cloud e.g. AWS, for Read-Heavy workloads.

➔ SIGNIFICANT COST SAVINGS.

2) In SC Mode, client should be able to read from Replica when Master is down.

- This prevents unwanted Timeouts on Reads when Master goes down.
- Overall Client / Server discovery typically takes ~1.8 seconds to promote Replica to Master.

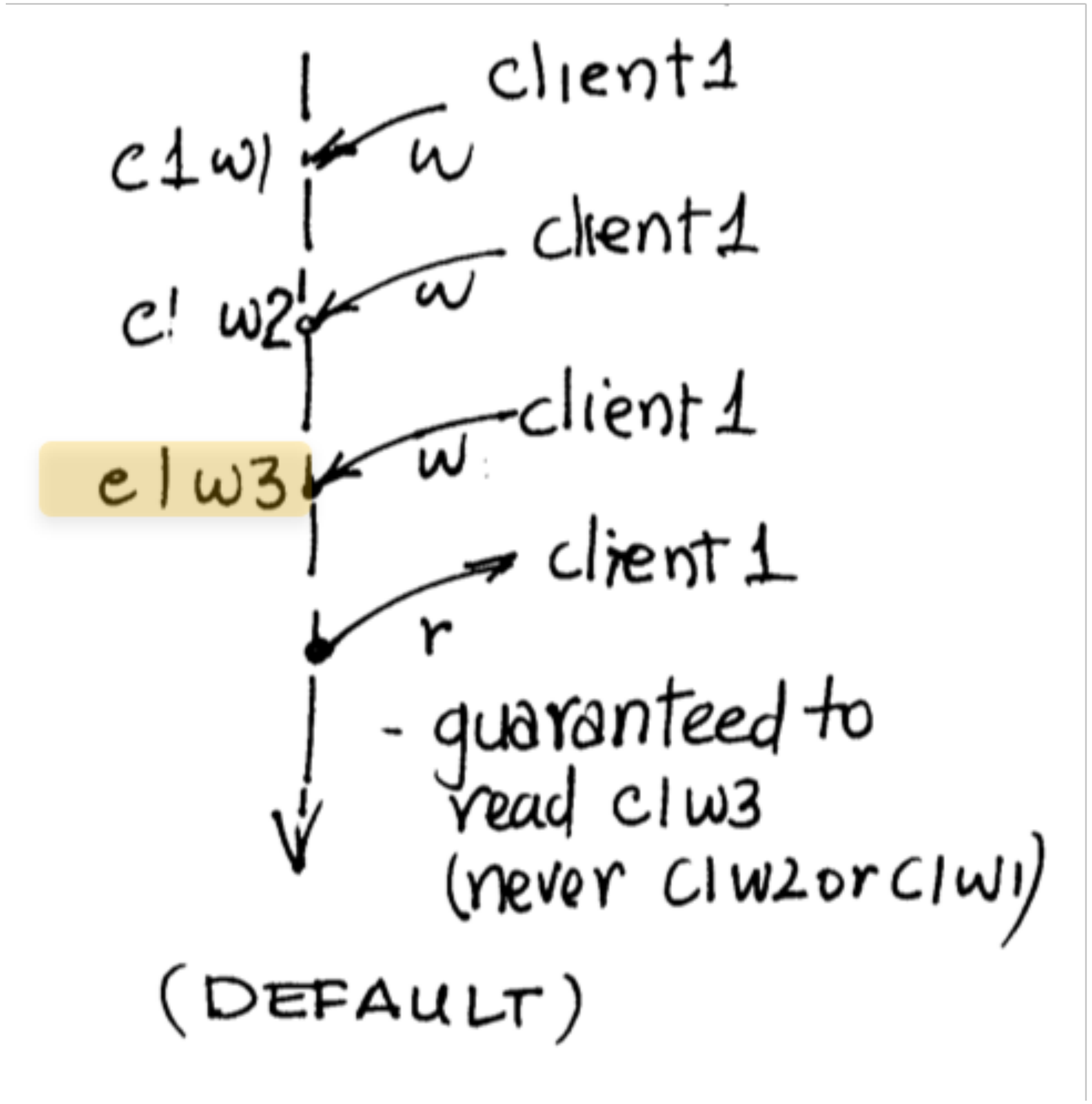
What is the trade-off?

- Relaxed Reads from Replica come at the expense of session consistency guarantee.
- There will be corner cases, *though probability is quite low*, where **even for same session, stale reads** may be returned.

  i.e. c1w2 could be returned instead of c1w3.

- Still, no dirty reads. (i.e. uncommitted writes are not returned.)
- Customers demanding relaxed consistency reads are OK with these trade-offs in their data model.

Session Consistency



New Read Policy Options in SC Mode

Ver 4.5.2.1+, in SC Mode, introduces 4 distinct read policies.

Java Client Example: ReadModeSC settings:

- LINEARIZE - most strict, implements Global Read Linearizability, reads from Master.
- SESSION – *default*, implements session consistency, reads from Master.
- ALLOW_REPLICA – allows reads from Master or any full (non-migrating) Replica. Session Consistency is not guaranteed.
- ALLOW_UNAVAILABLE – "ALLOW_REPLICA" reads even from UNAVAILABLE partitions.

Client Behavior (Updated for Server version 4.5.2.1+)

Which node does the client read from?

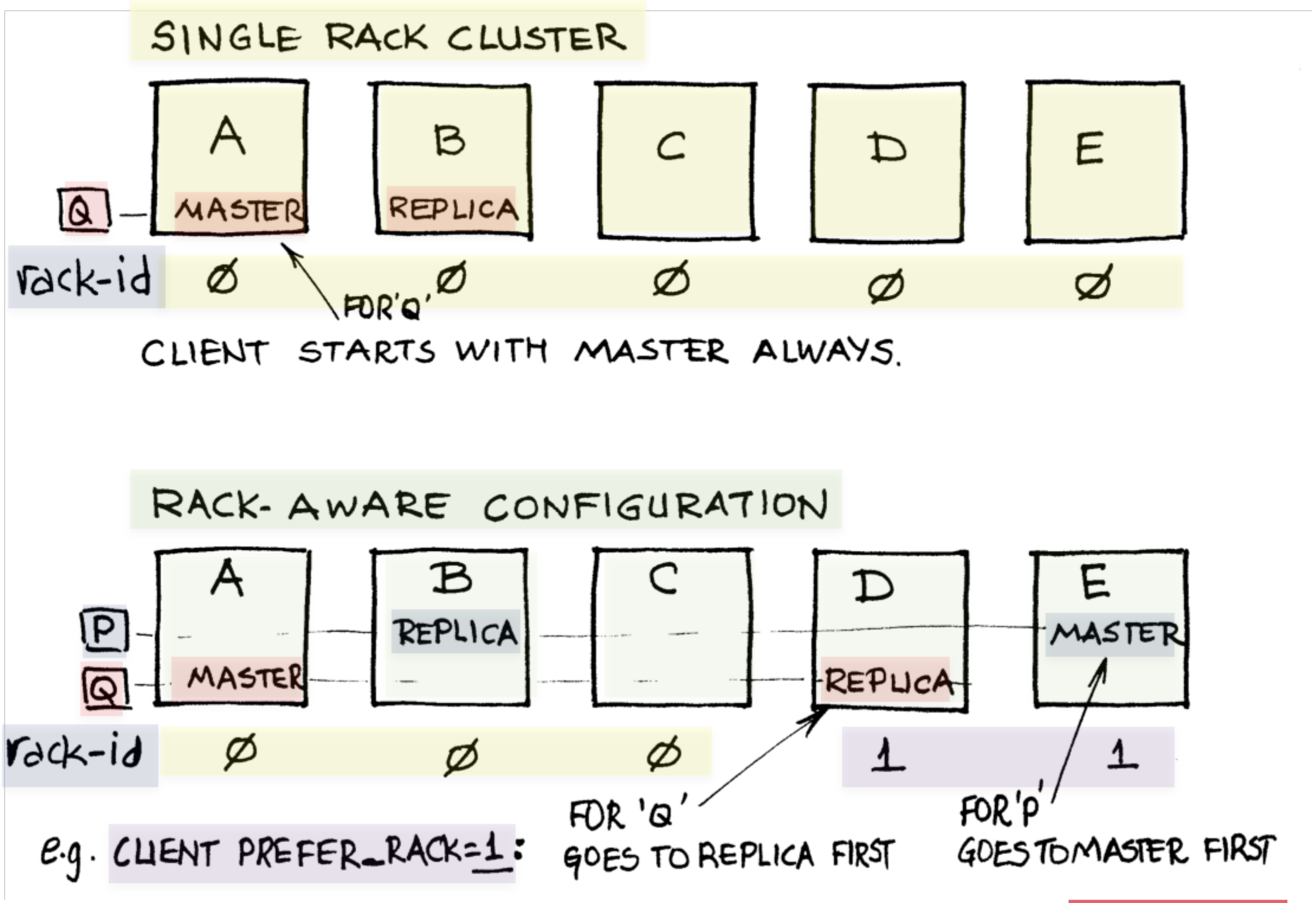
- LINEARIZE
- SESSION

From Master only.

- ALLOW_REPLICA
- ALLOW_UNAVAILABLE.

Single Rack: Master, if fails, then Replica.

Rack-Aware w/Preferred rack Reads: Master or Replica on preferred rack, if fails, Replica or Master on other rack.



SC Mode Performance Comparison

- Comparable reads/writes in session consistency, global linearizability and AP mode.
- In steady cluster state, performance in SC and AP modes is comparable.

	SC Mode Global Linearizablity	SC Mode Session Consistency	AP Mode
Update Latency	630 μs	640 μs	640 μs
Read Latency	548 μs	225 μs	220 μs
OPS	1.87 million	5.95 million	6 million

- 500M records, 8 byte objects, in-memory with persistence, RF=2.

SC Mode Configuration Recommendations

Preferred SC Mode configuration options are:

- Replication Factor = 2
- Suggested cluster size: >= (2 x RF) – 1.
  - Allows up to (Replication Factor – 1) failures.
- Storage engine device preferred. (commit-to-device ensures durability)
- Zero Default TTL preferred. (SC mode records should not need expiration or eviction.)

Rack-Awareness (EE only) in SC Mode

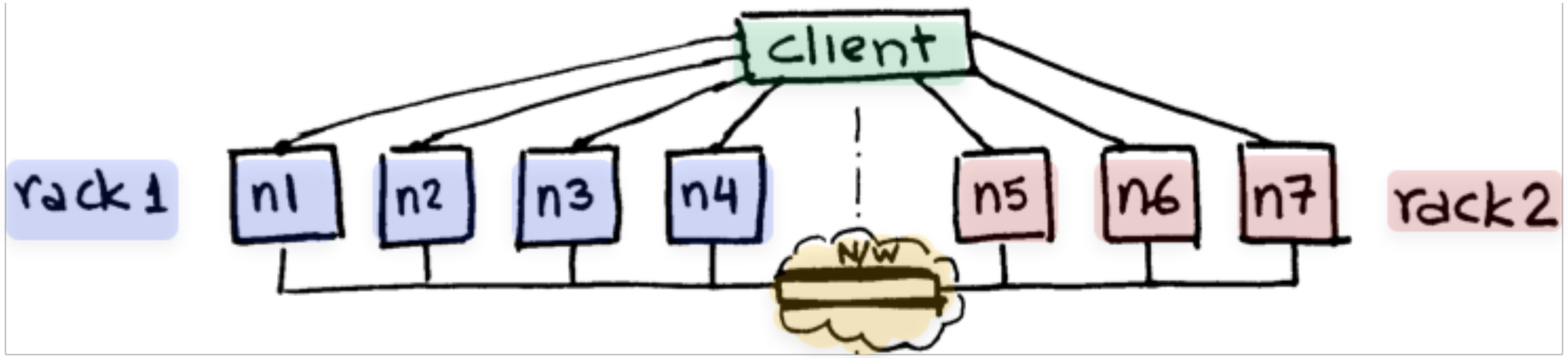
Rack Awareness: Nodes from different racks are assigned to a partition.

- In SC namespace configuration on each node, assign node's rack-id.
- Different SC namespaces, in the same cluster, can specify different rack-id's

  ➔ rack awareness is on a namespace basis.
- By default, we have a single rack (default rack-id = 0) and the initial partition map does not need any adjustment (unless using Uniform Balance).

Rack-Awareness in SC Mode – Commonly Used Pattern

- RF=2, # racks = 2, odd number of nodes



- If rack with minority nodes fails, we will have full availability on the majority rack.
- If rack with majority nodes fails, all availability will be lost.
    - **Rule:** Minority cluster needs both roster-master & roster-replica (full/subset combos)

Q&A?

AEROSPIKE