KOENIG
step forward

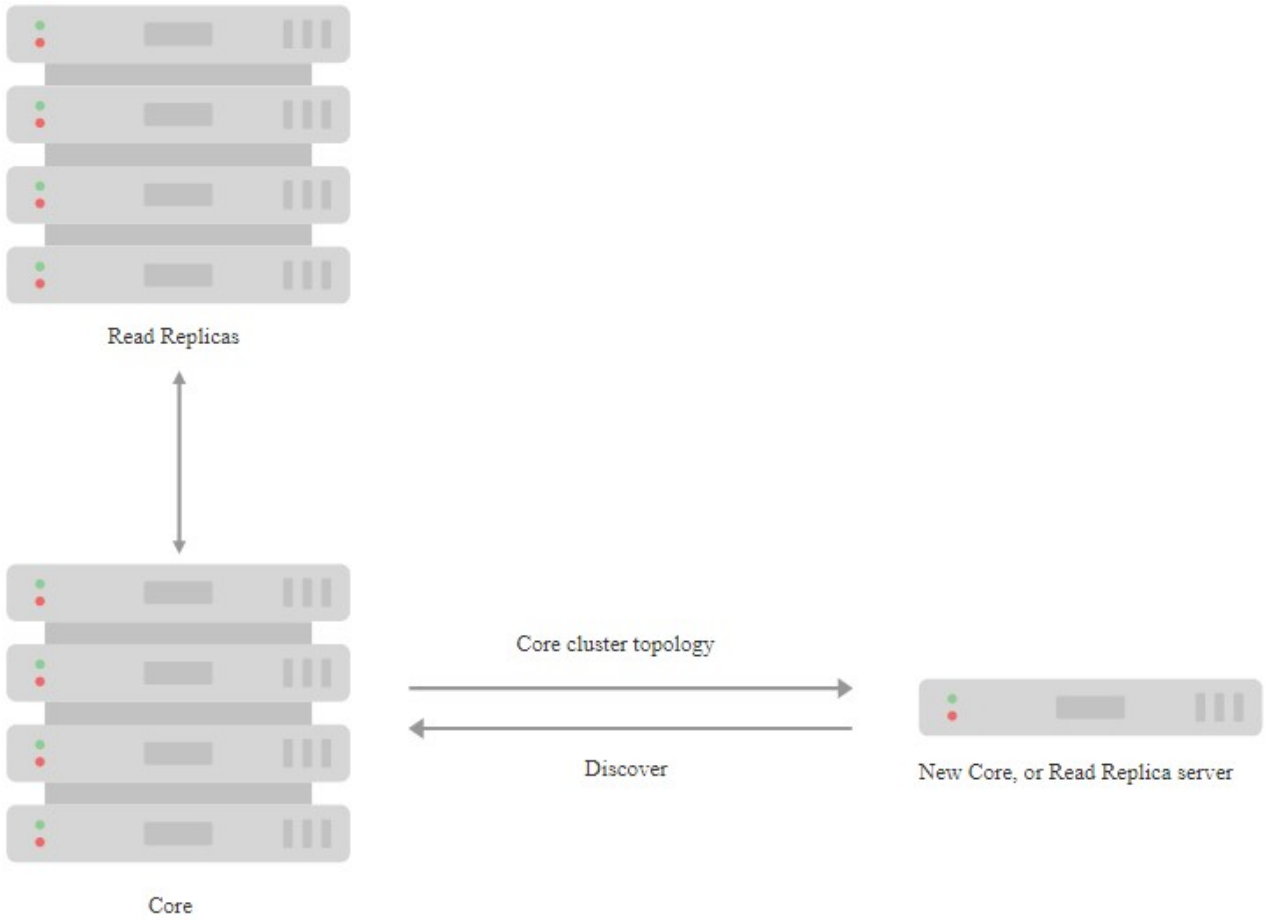
Causal Clustering lifecycle

Introduction

- To develop deeper knowledge of how the cluster operates
- We will follow the lifecycle of a cluster
- We will
  - Boot a Core cluster
  - Add in Read Replicas
    - How they join the cluster
    - Catchup and remain caught up with the Core Servers
  - How backup is used in live cluster environments

Discovery protocol

- The first step in forming a Causal Cluster
- It takes in some information about existing Core cluster servers, and uses this to initiate a network join protocol.
- Using this information, the server will either join an existing cluster or form one of its own.



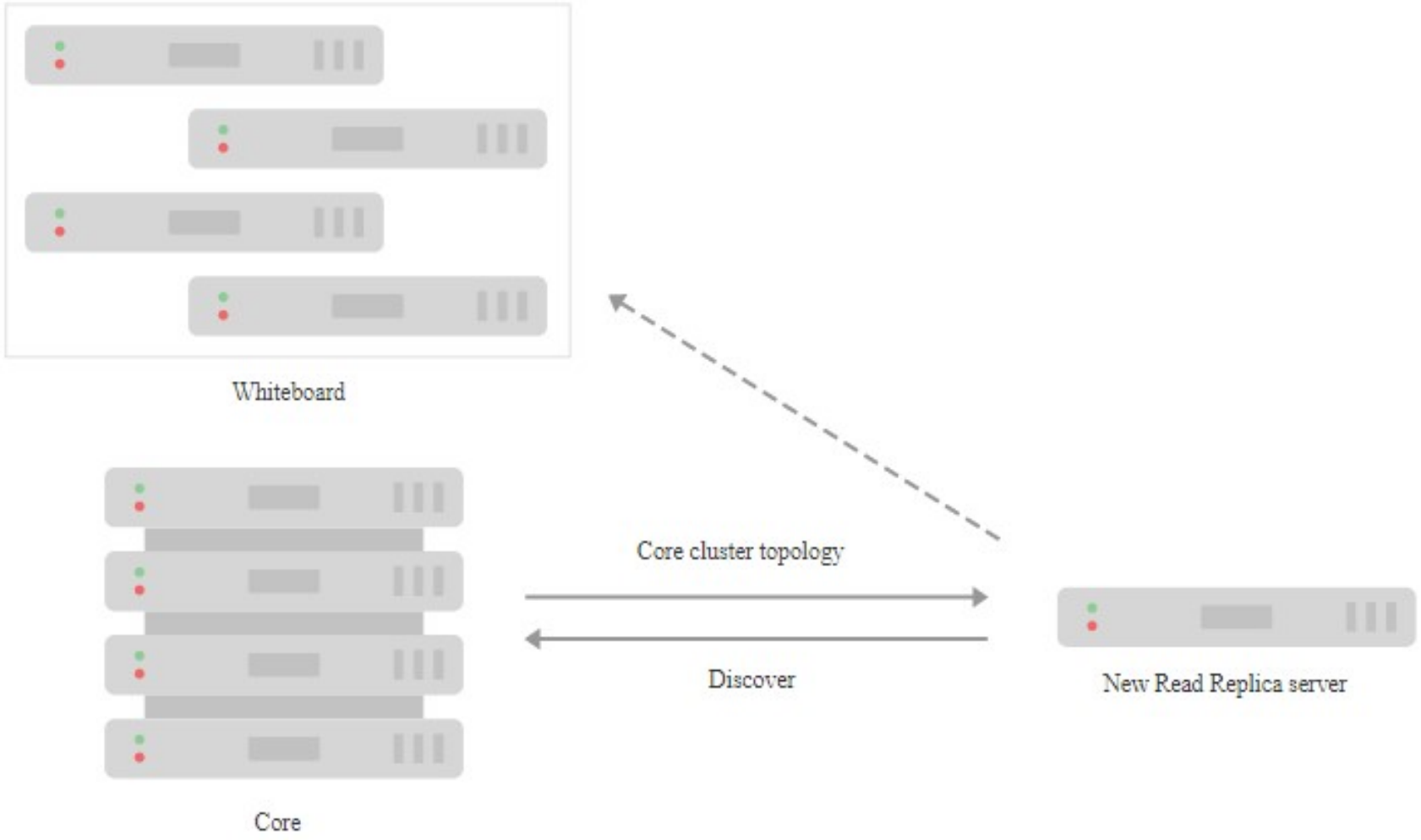
Discovery protocol

- Takes information from
  - causal_clustering.initial_discovery_members
- It lists IP addresses and ports that form the cluster on startup.

Core membership

- If a Core Server is performing discovery
  - Once it has made a connection to the one of the existing Core Servers, it then joins the Raft protocol.
- The new Core Server must also catch up its own Raft logs with respect to the other Core Servers
- There will be a delay before the new Core Server becomes available if it also needs to catch up
- Where a joining Neo4j instance has databases whose names match databases which already exist in the cluster, the database stores on the joining instance must be the same as their counterparts on cluster members
- A new instance may also join a cluster if it does not contain any matching databases.

Read Replica membership



Transacting via Raft protocol

KOENIG
step forward

- Once bootstrapped, each Core Server spends its time processing database transactions
- Updates are reliably replicated around Core Servers via the Raft protocol.
- Updates appear in the form of a (committed) Raft log entry containing transaction commands which is subsequently applied to update the database.

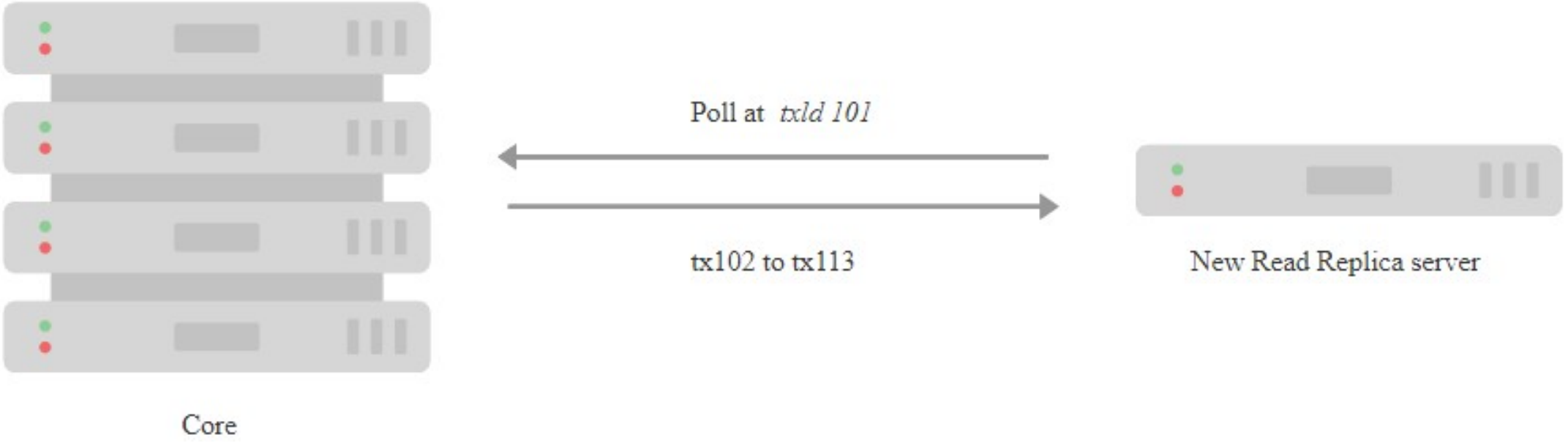
Transacting via Raft protocol

KOENIG
step forward

- The Raft Leader for the current term
  - Appends the transaction to the head of its local log and
  - Asks the other instances to do the same
- When the Leader can see that a majority instances have appended the entry, it can be considered committed into the Raft log
- The client application can now be informed that the transaction has safely committed
- As each database operates within a logically separate Raft group
  - A core server can have multiple roles: one for each database
  - For example, it may be the Leader for database system and at the same time be a Follower for database neo4j.

Catchup protocol

- Read Replicas spend their time
  - Concurrently processing graph queries and
  - Applying a stream of transactions from the Core Servers to update their local graph store.



Read Replica shutdown

- Clean shutdown
  - Invokes the discovery protocol to remove itself from shared whiteboard overview of cluster
- Unclean shutdown
  - Core Servers will notice that the Read Replica's connection has been cut
  - The discovery machinery will initially hide the Read Replica's whiteboard entry
  - On subsequent reboot the Read Replica will rollback any partially applied transactions such that the database is in a consistent state

Core shutdown

KOENIG
step forward

- Handled via the Raft protocol
- Will eventually be voted out from the Raft group
- Cluster has grown smaller, and is therefore less fault tolerant
- For any databases where the leaver was playing the Leader role, each of those leaderships will be transferred to other Core Servers
- Once the new Leader is established, the Core cluster continues
- Raft may only reduce a cluster size to the configured
  - causal_clustering.minimum_core_cluster_size_at_runtime
- Once the cluster has reached this size, it will stop voting out members