High availability in Memgraph

Andi Škrgat

Content of this deck

1. Motivation
2. Architecture
3. Coordinators
4. Automatic failover
5. Routing
6. Authentication
7. Queries

01

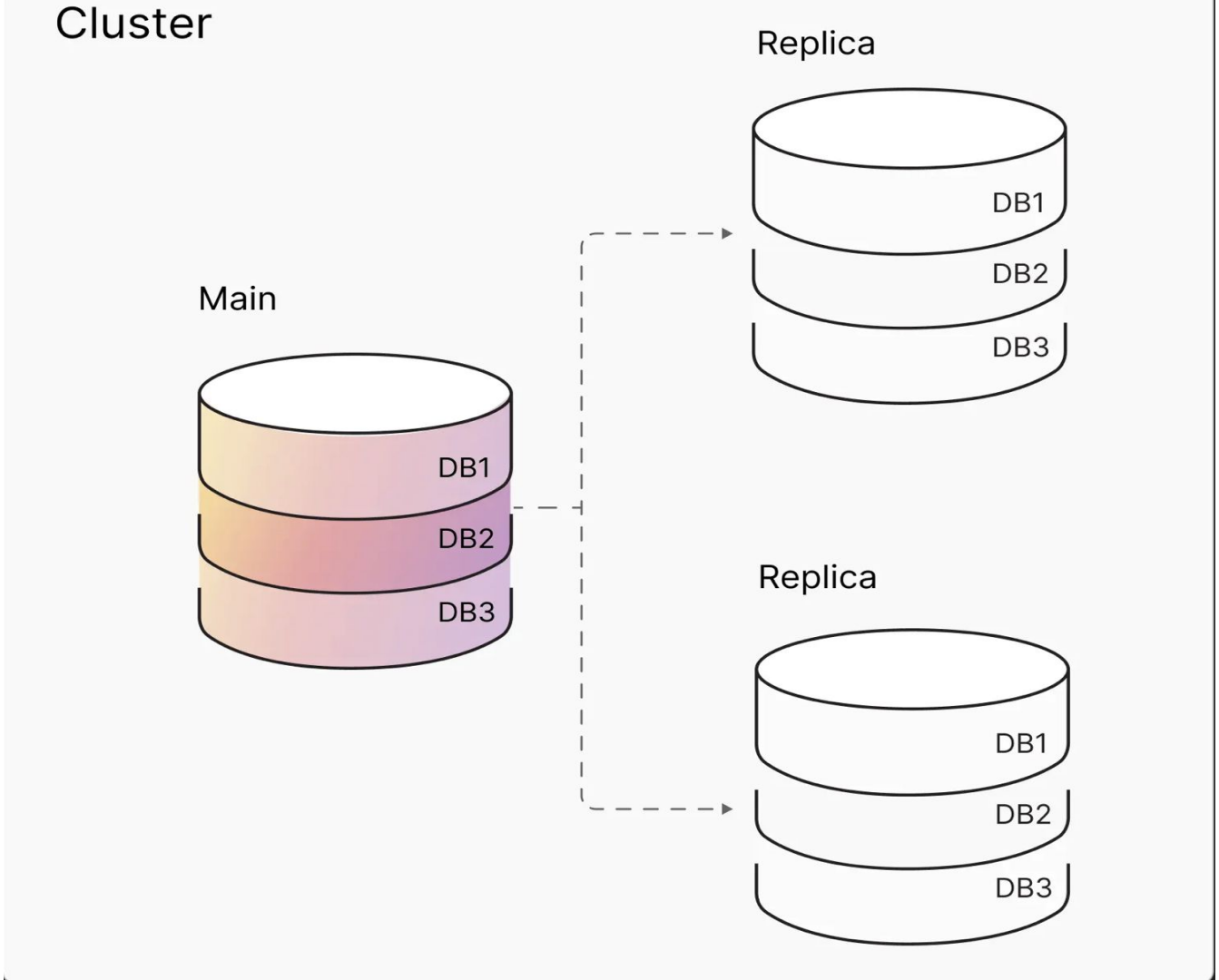
Motivation

Introduction thoughts

- We are talking about primary-backup replication model
- Main is the only writable instance
- Reading both from main and replicas

Replication motivation

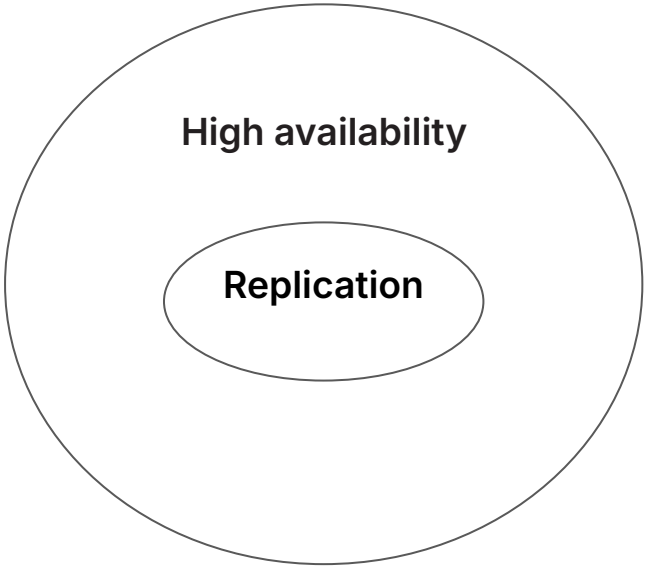
- Multiple copies of data in order to be **fault tolerant**
  - Survive app crashes
  - Storage failures
  - Node outages
  - Network management
- **Scalability** ⇒ distribute read/write queries across the system (horizontal scalability)
- **Performance** ⇒ Data closer to users
- **Transparency** ⇒ Users don’t know whether it is a single instance or a cluster in the background


Cluster
Replica
DB1
DB2
DB3
Main
DB1
DB2
DB3
Replica
DB1
DB2
DB3

High availability motivation

HA = Everything from replication +
maximize uptime for read and **write queries**

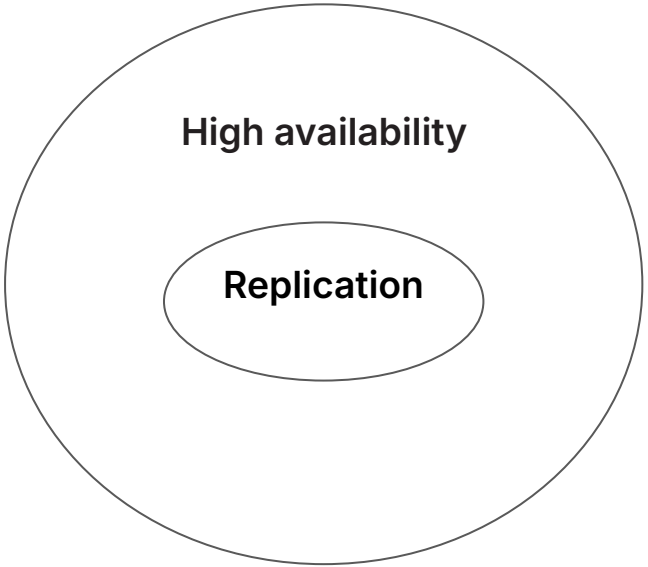
- Cluster management ⇒
  - Instance comes back?
  - **Main goes down?**
  - Main changes?
  - How to better scale?
  - Remove a data instance?
  - ...



High availability motivation

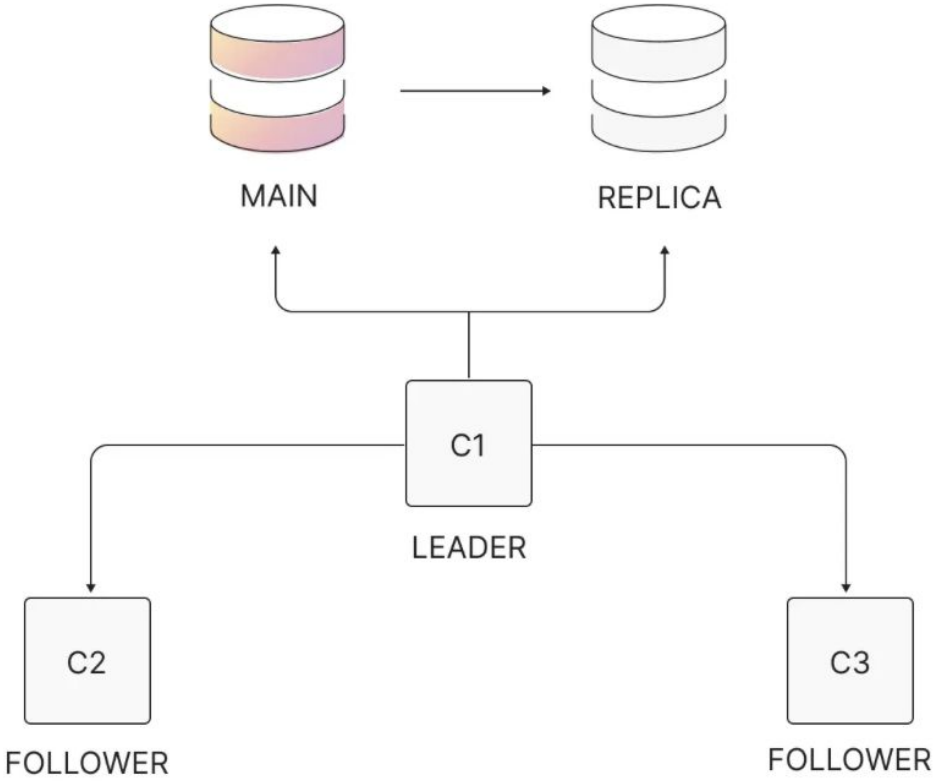
HA = Everything from replication +
maximize uptime for read and **write queries**

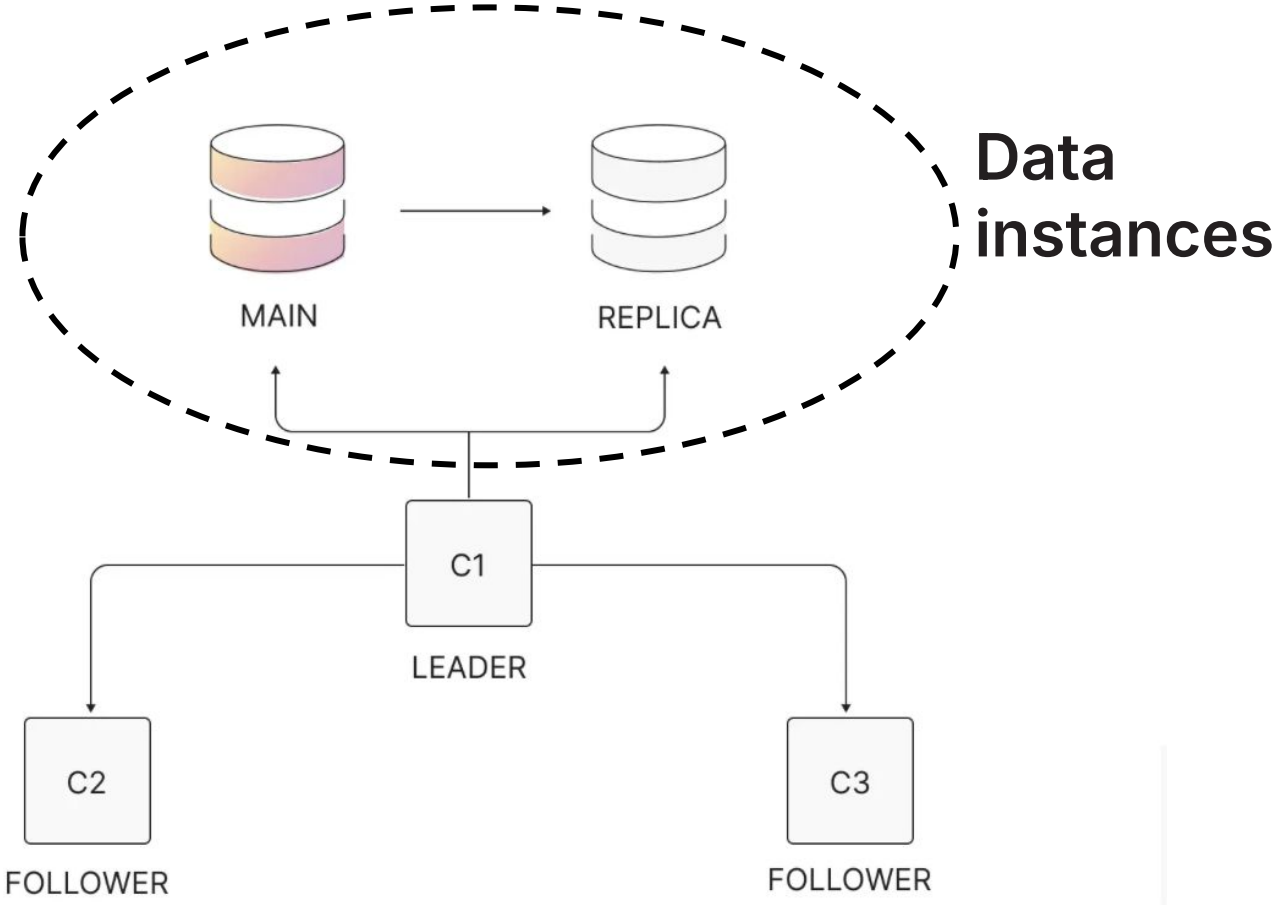
- Cluster management ⇒
  - Instance comes back?
  - **Main goes down? AUTOMATIC FAILOVER**
  - Main changes?
  - How to better scale?
  - Remove a data instance?
  - ...



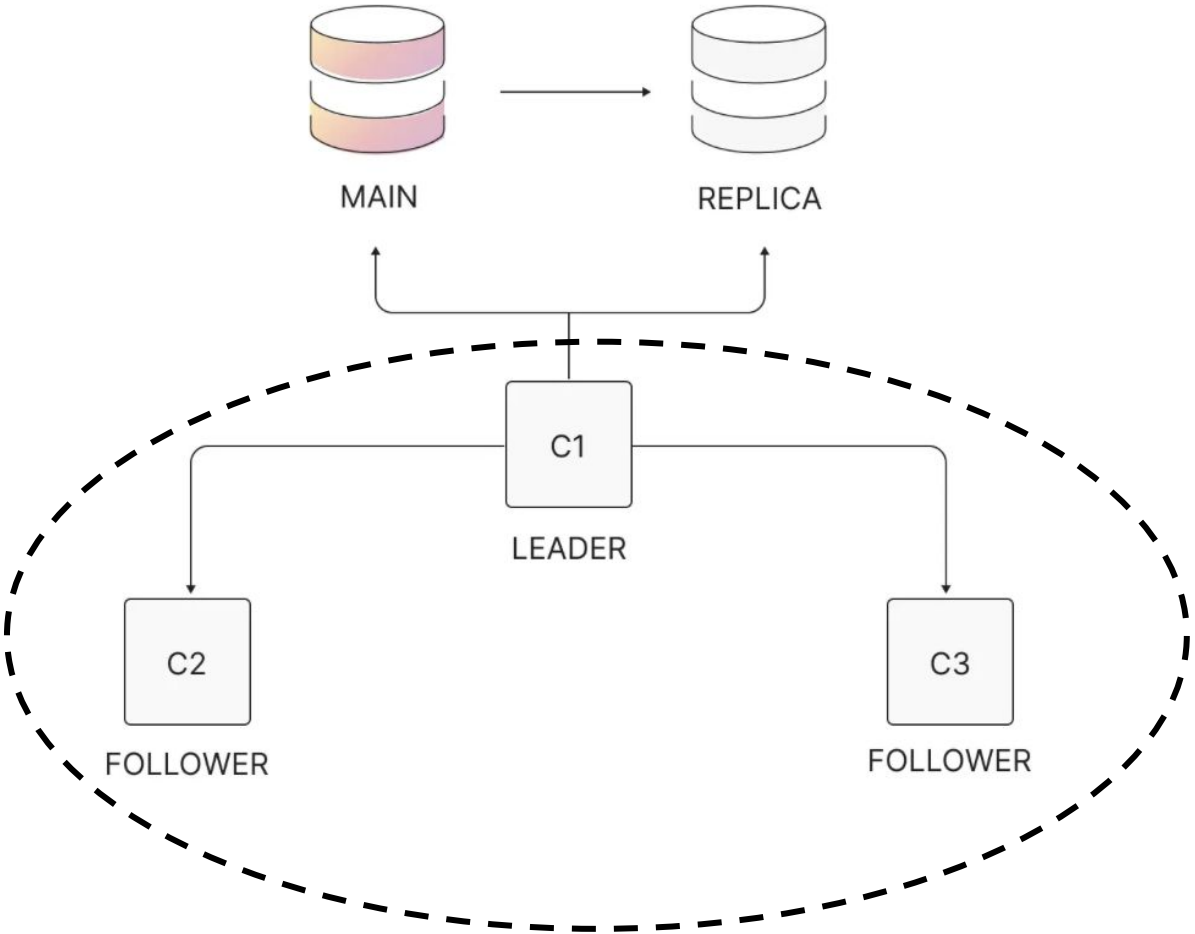
02

Architecture


MAIN
REPLICA
C1
LEADER
C2
FOLLOWER
C3
FOLLOWER



Data instances



Raft coords

03

Coordinators

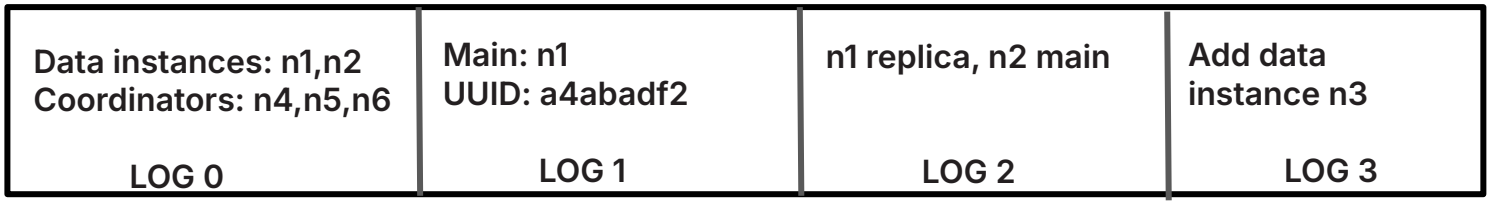
Goals of Raft cluster

- Consensus algorithm ⇒ managing replicated log
  - If any server has applied a log entry to its state machine, than no other server may apply a different command for the same log
- Leader election ⇒ single writable leader that accepts log entries
- Fully functional as long as majority of servers are up
- Simpler than Paxos

- Paper
- Explanation

Cluster state as replicated log

- Each log entry is a delta log ⇒ what is changing with this log instead of saying “this is your whole new state”
- JSON serialization



User interaction

1. REGISTER INSTANCE …
2. Create a delta log entry
3. Coordinators commit

Users runs a query on top of the leader coordinator

Parse the query and convert it to the log

Leader replies to the coordinator that the query is committed as soon as the majority of coordinators accept it.

Background thread applies logs to the in-memory state.

Code components

- CoordinatorLogStore ⇒ log management (in-memory and on-disk)
- CoordinatorStateManager ⇒ cluster configuration management
- CoordinatorStateMachine ⇒ state machine management
    - CoordinatorClusterState ⇒ in-memory representation of the cluster state

Delta log

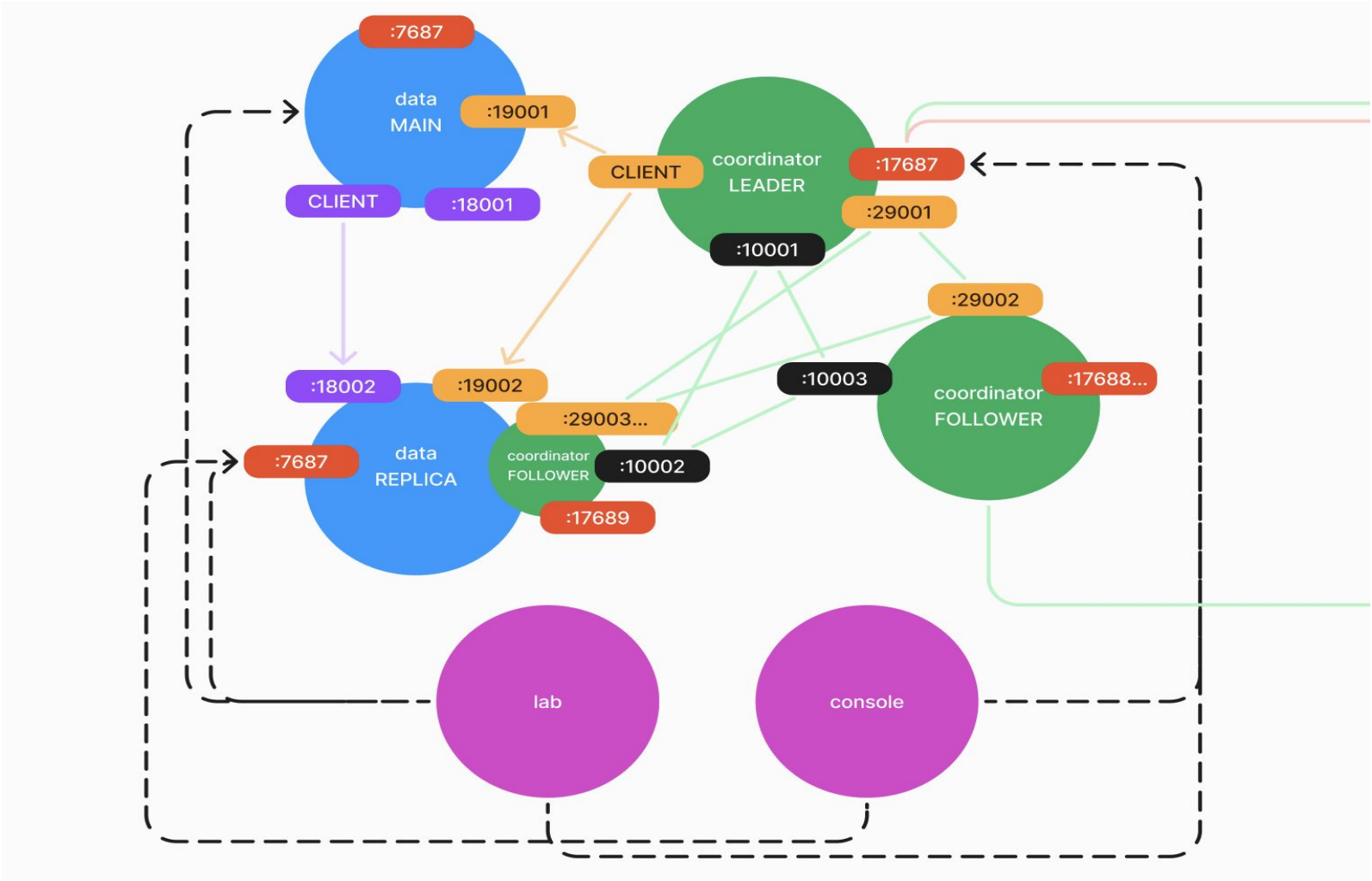
```
Andi Skrgat, 3 months ago | 1 author (Andi Skrgat)
struct CoordinatorClusterStateDelta {
  std::optional<std::vector<DataInstanceContext>> data_instances_;
  std::optional<std::vector<CoordinatorInstanceContext>> coordinator_instances_;
  std::optional<utils::UUID> current_main_uuid_;
  std::optional<bool> enabled_reads_on_main_;
  std::optional<bool> sync_failover_only_;
  std::optional<uint64_t> max_failover_replica_lag_;
  std::optional<uint64_t> max_replica_read_lag_;        feat: Add support for rout

  bool operator==(const CoordinatorClusterStateDelta &other) const = default;
};
```

Cluster state

```
private:
 std::vector<DataInstanceContext> data_instances_;
 std::vector<CoordinatorInstanceContext> coordinator_instances_;
 utils::UUID current_main_uuid_;
 // The option controls whether reads are enabled from the main instance. The default is set to false so we don't
 // overwhelm main with both read and write requests
 bool enabled_reads_on_main_{false};
 // The option controls whether it is allowed to failover only on sync and strict_sync replicas or async replicas
 // are also supported.
 bool sync_failover_only_{true};
 // The option controls what is the maximum lag allowed on any replica's database so it could be considered a valid
 // failover candidate
 uint64_t max_failover_replica_lag_{std::numeric_limits<uint64_t>::max()};
 // The option controls what is the maximum lag allowed on the replica for the specific database when the routing
 // table is requested.
 uint64_t max_replica_read_lag_{std::numeric_limits<uint64_t>::max()};
 mutable utils::ResourceLock app_lock_;
};
```

04

Automatic failover


:7687
data
MAIN
:19001
CLIENT
:18001
CLIENT
coordinator
LEADER
:17687
:29001
:10001
:29002
:18002
:19002
:10003
:17688...
coordinator
FOLLOWER
:29003...
:7687
data
REPLICA
coordinator
FOLLOWER
:10002
:17689
lab
console

Servers

- DataInstanceManagementServer
  - On data instances
  - To receive RPC messages from coordinators
- ReplicationServer
  - On data instances
  - To receive RPC messages from data instances
- CoordinatorServer
  - On coordinators
  - Used by NuRaft for their communication
- CoordinatorInstanceManagementServer
  - On coordinators
  - To receive RPC messages from coordinators

Main UUID

- which UUIDs should replicas listen to
- stored as part of the current cluster state in coordinator logs
- prevents split brain issue

- Epochs
  - Serves the same purpose as main UUID and **should be unified** but this one is from old times and not connected to coordinators
- Epoch history
  - Deque of pairs of <epoch_id, ldt> ⇒ not related to coordinators

Success callback

- Leader coordinator is constantly pinging data instances
- **Reconciliation loop model** ⇒ on the failover, we write to Raft that instance M should become MAIN but we don’t send immediately RPC (actor model idea)
- Inspiration from K8s ⇒ we have a centralized place where the source of the truth is stored and then we constantly compare current state with the target state stored in Raft logs (ArgoCD also)

When to react?

- Instance is replica but should be main
- Main has writes disabled
- UUID on Main is wrong
- Instance is main but should be replica
- Instance is replica but listens to the wrong main

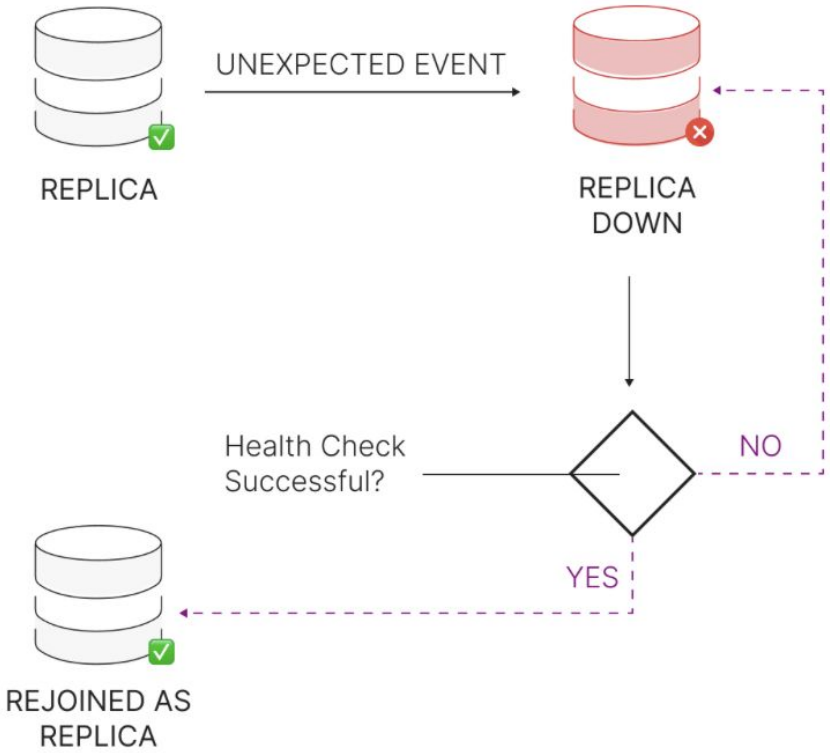
Failure callback

algorithm.txt

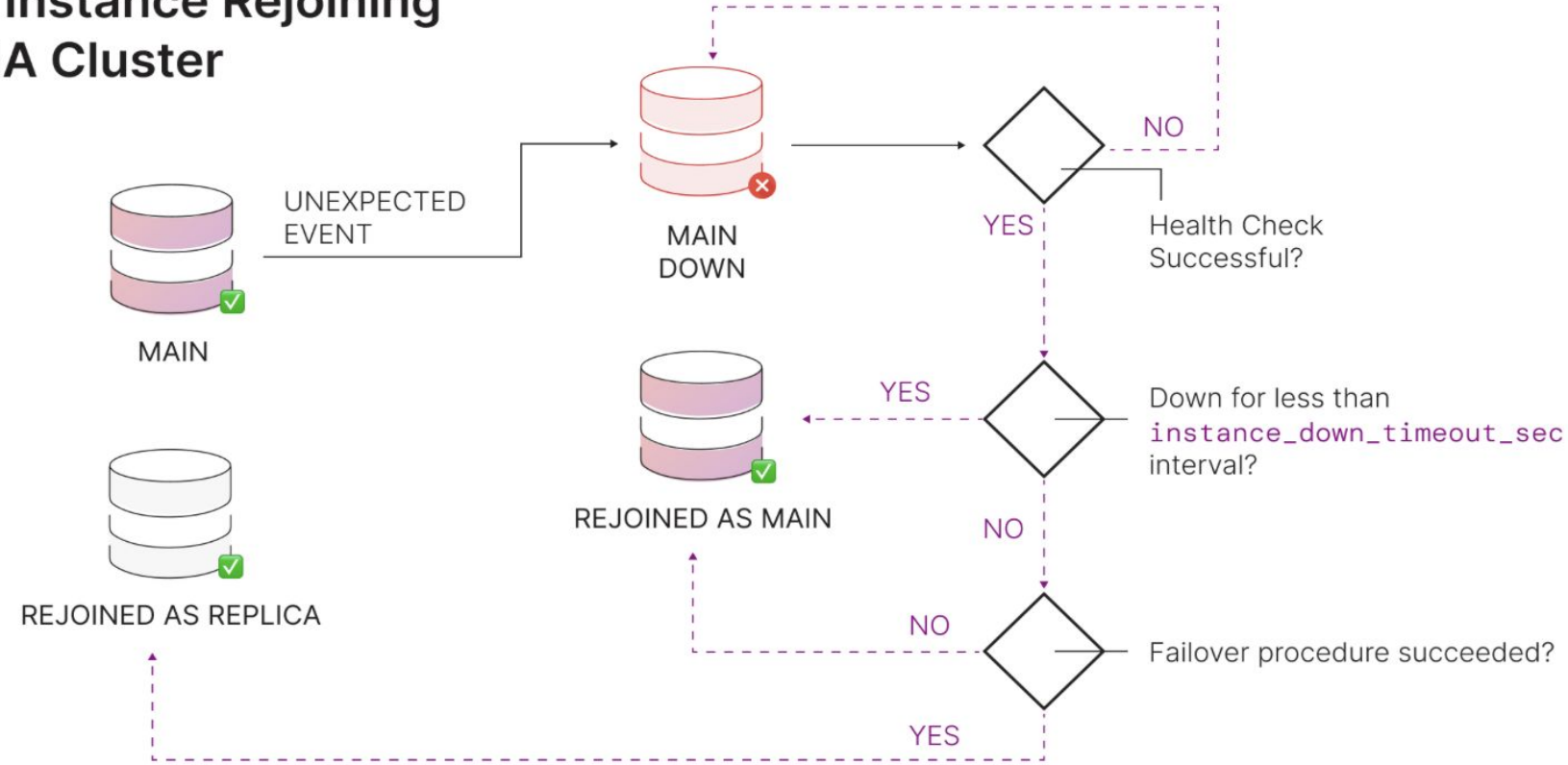
```
1 if instance.response is False:
2   if now() - last_response_time > threshold:
3     if instance is main:
4       TryFailover();
5
```

```
Instance TryFailover() {
  struct InstanceDBInfo {
    std::string db_uuid;
    uint64_t num_committed_txns;
  };

  struct InstanceInfo {
    std::vector<InstanceDBInfo> dbs_info;
    uint64_t last_committed_system_timestamp;
  };

for (instance in data_instances) {
  InstanceInfo instance_info = GetInstanceInfo();
}

auto instances_info = data_instances | std::views::transform(GetInstanceInfo()) | to_map<instance_name, InstanceInfo>;
auto largest_sys_ts_instance = std::ranges::find(instances_info, GetInstanceWithLargestSysTs);
auto dbs_from_largest_sys_ts_instance = largest_sys_ts_instance.dbs;

std::map<instance_name, counter> counts;
for(db in dbs_from_largest_sys_ts_instance) {
  auto newest_instance = FindInstanceThatHasMostData(db);
}
counts[newest_instance]++;

return instance with max counts;
}
```

REPLICA Instance Rejoining The HA Cluster



MAIN Instance Rejoining The HA Cluster

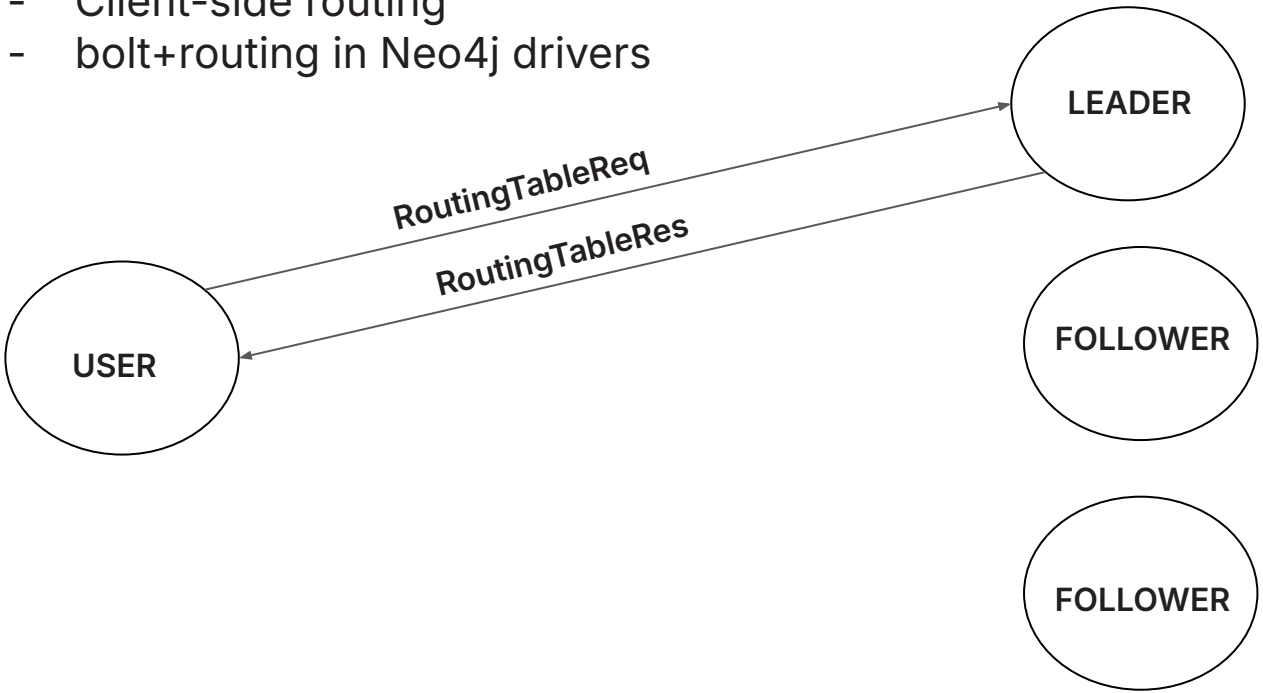


05

Routing

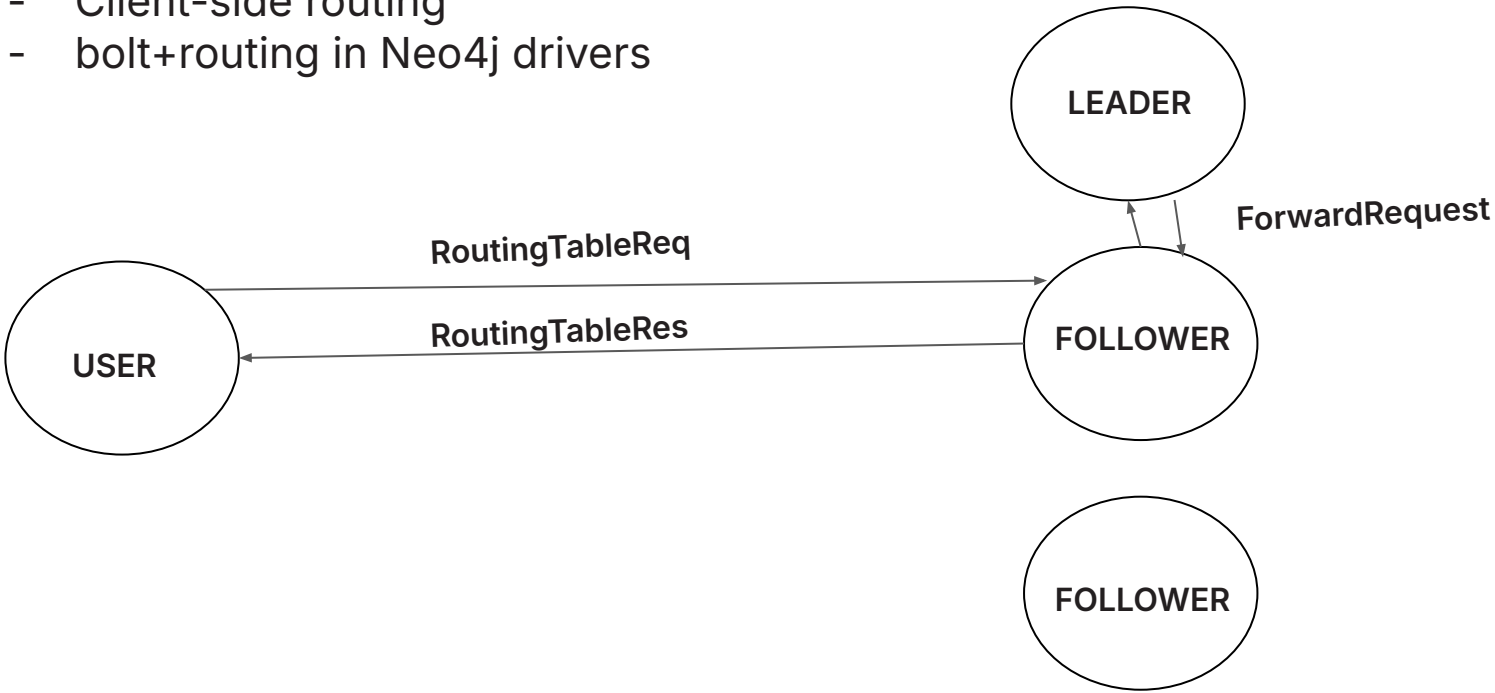
Routing

- Abstraction for users so they don’t need ping each instance if it is alive
- Client-side routing
- bolt+routing in Neo4j drivers

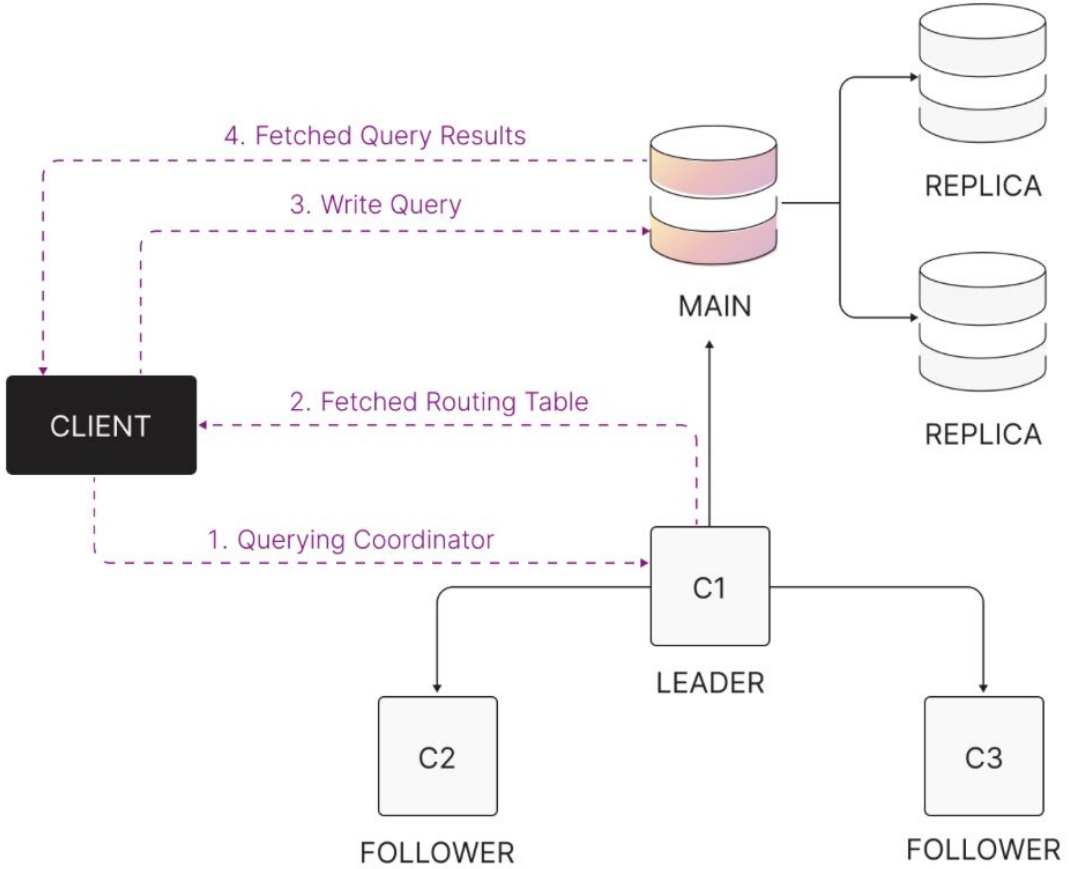


Routing

- Abstraction for users so they don’t need ping each instance if it is alive
- Client-side routing
- bolt+routing in Neo4j drivers



Routing Write Queries With Bolt+Routing



MAIN, REPLICA - Data Instances

C1, C2, C3 - Coordinator Instances

RoutingTableFormat

- READ: n1,n2,n3
- WRITE: n1
- ROUTE: c1,c2,c3

- Routing table changes based on flag values
  - enabled_reads_on_main
  - max_replica_read_lag

06

Authentication

Authentication

- Routing: We allow anyone to connect to coordinators and to request routing table but the proper authentication then happens on data instances ⇒ you need to provide credentials
- Direct bolt: No credentials needed

07

In-Service Software Upgrade (ISSU)

ISSU

1. Do a backup of all of your data
2. Restart replicas
3. Restart main
4. Restart coordinators

07

Queries

Thank you for your time!

MEM GRAPH

www.memgraph.com