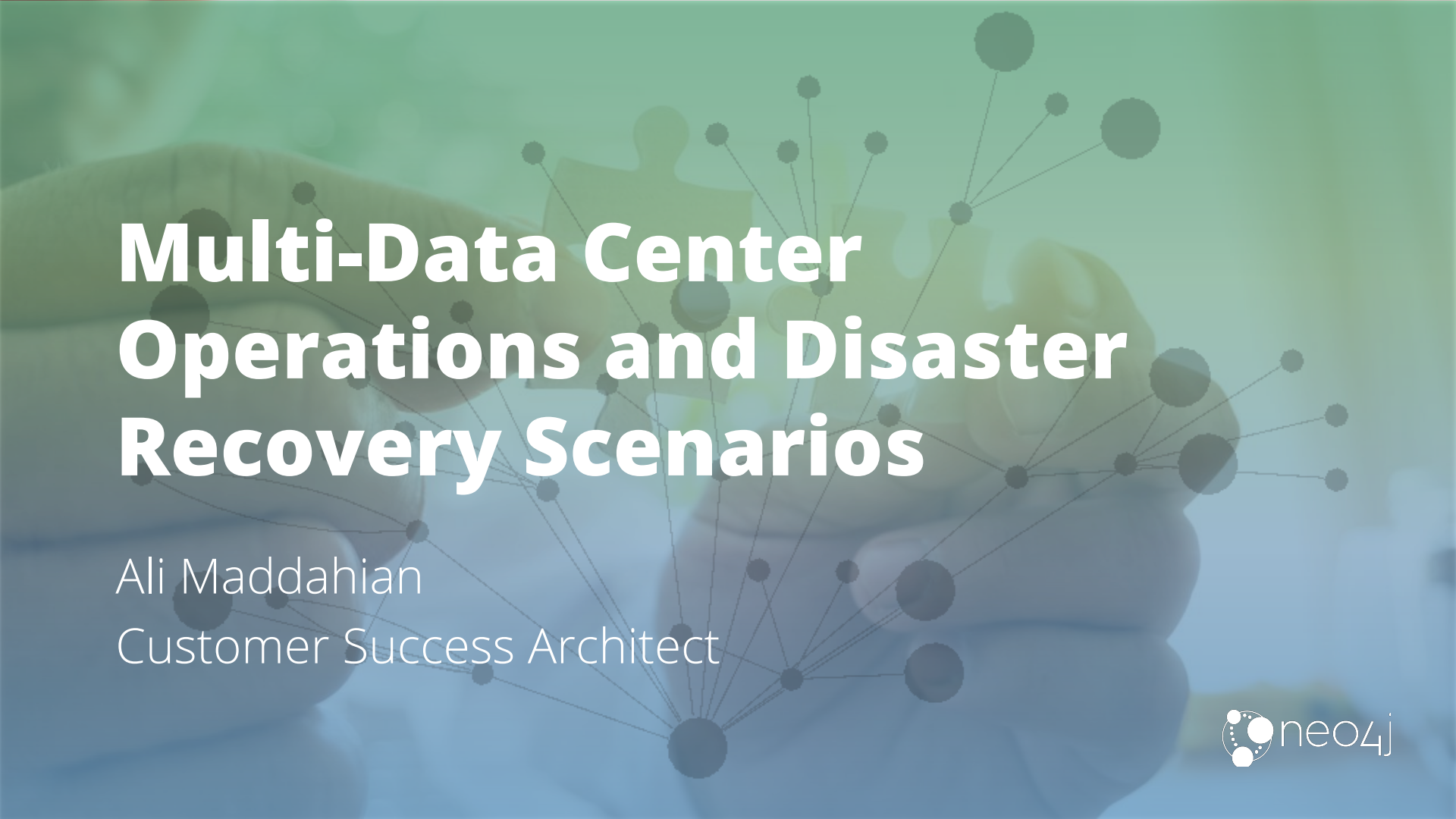


Multi-Data Center Operations and Disaster Recovery Scenarios

Ali Maddahian
Customer Success Architect

neo4j

Agenda:

- Causal Clusters and Fault Tolerance
- Multi Datacenter Operations
- Disaster Recovery Scenarios

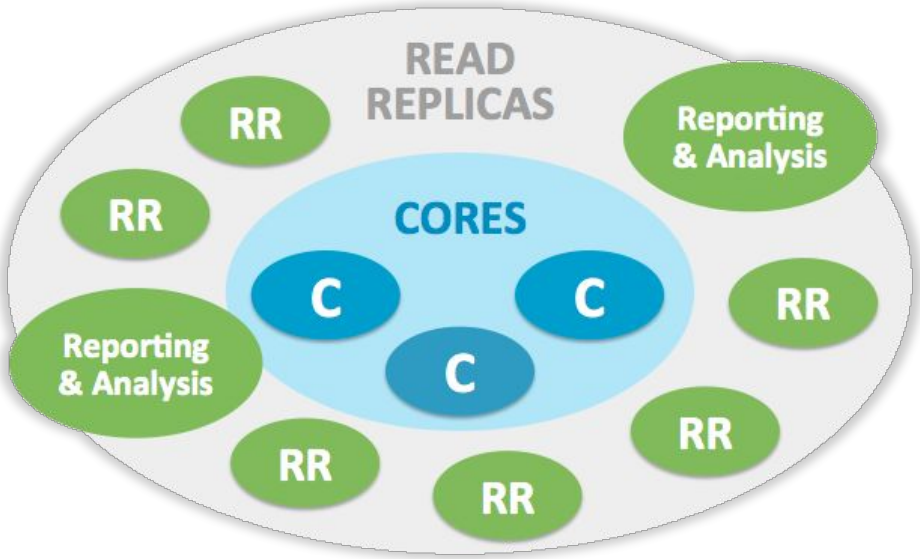


Causal Clusters and Fault Tolerance



Causal Cluster Overview

- RAFT-based architecture
  - Continuous availability
  - Consensus commits
- Seamless load balancing
  - Drivers --> Bolt --> Cluster
- Causal consistency
  - ACID-based consistency
  - Supports “read your own writes”



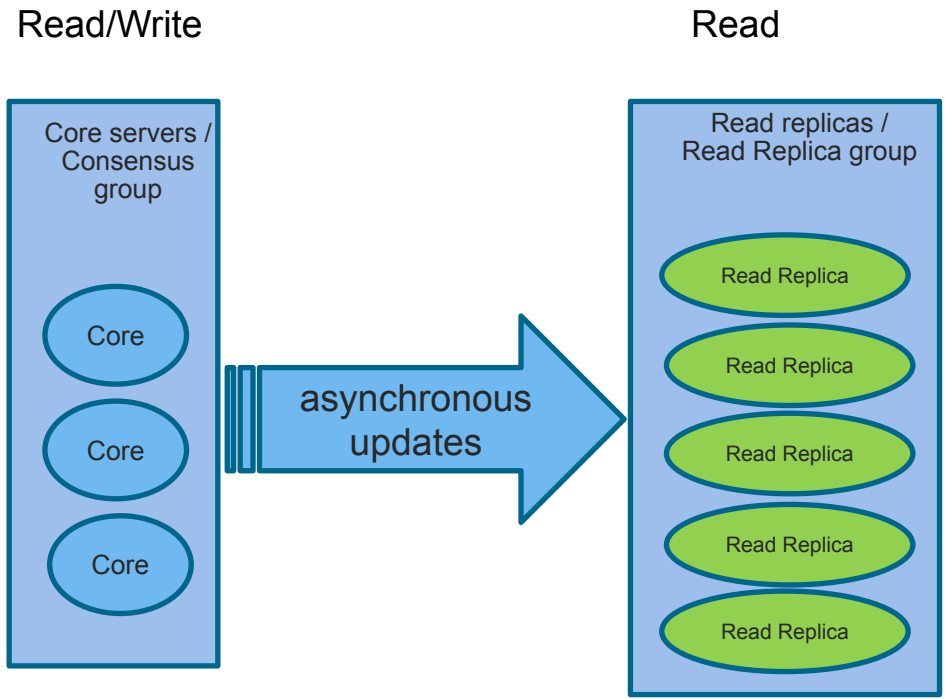
Causal Cluster overview

Cores

- Safety and fault-tolerance
- Consensus on updates
- Few machines

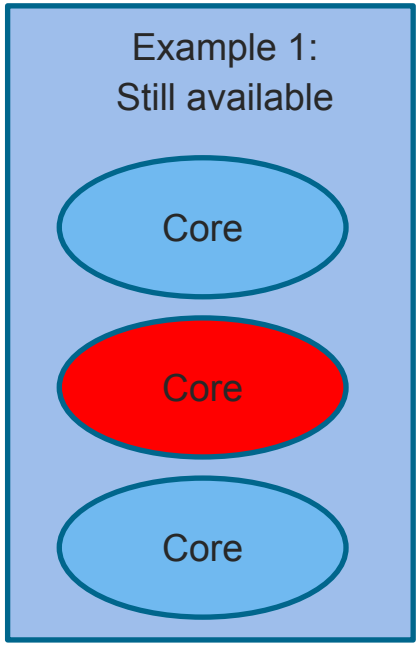
Read replicas

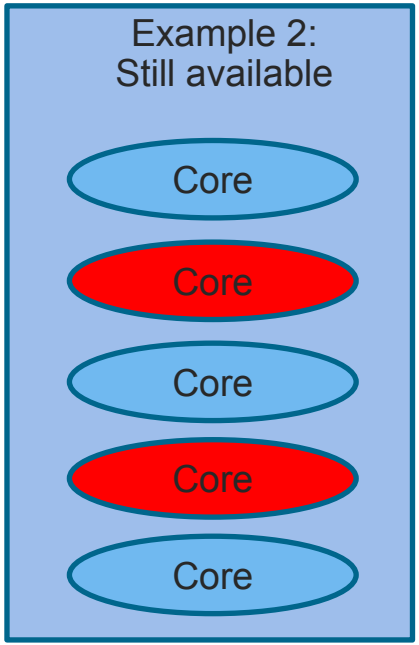
- Consistent read replicas
- Asynchronously updated
- Many machines
- Doesn’t affect fault-tolerance

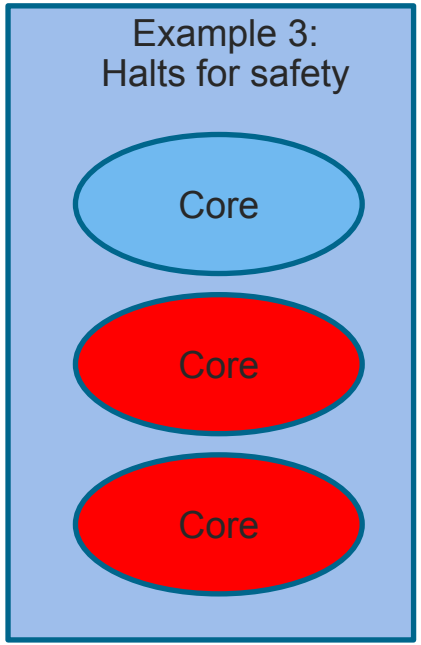


Fault tolerance and safety - Cores

- Fault tolerance for cores (M=2F + 1)
- M is the number of Core Servers required to tolerate F faults.



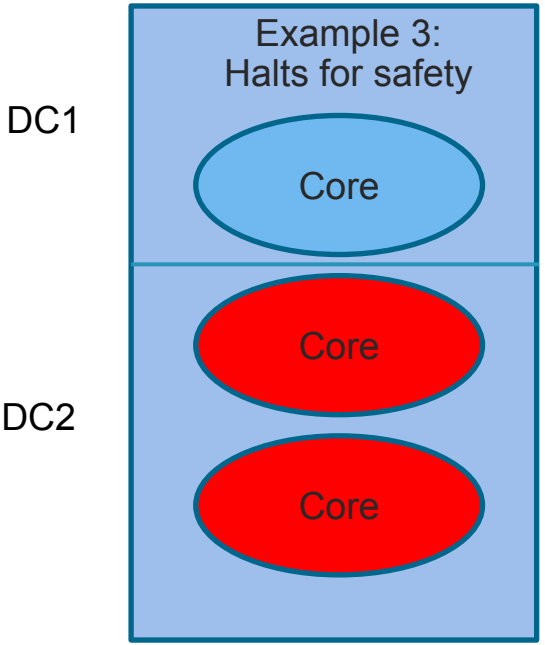




Fault tolerance and safety (cores) – multi DC

Multi DC

- Multi DC is still just one cluster.
- Consider physical limits affecting latency, bandwidth, performance, scalability, fault-tolerance.



https://neo4j.com/docs/operations-manual/current/clustering-advanced/multi-data-center/disaster-recovery/

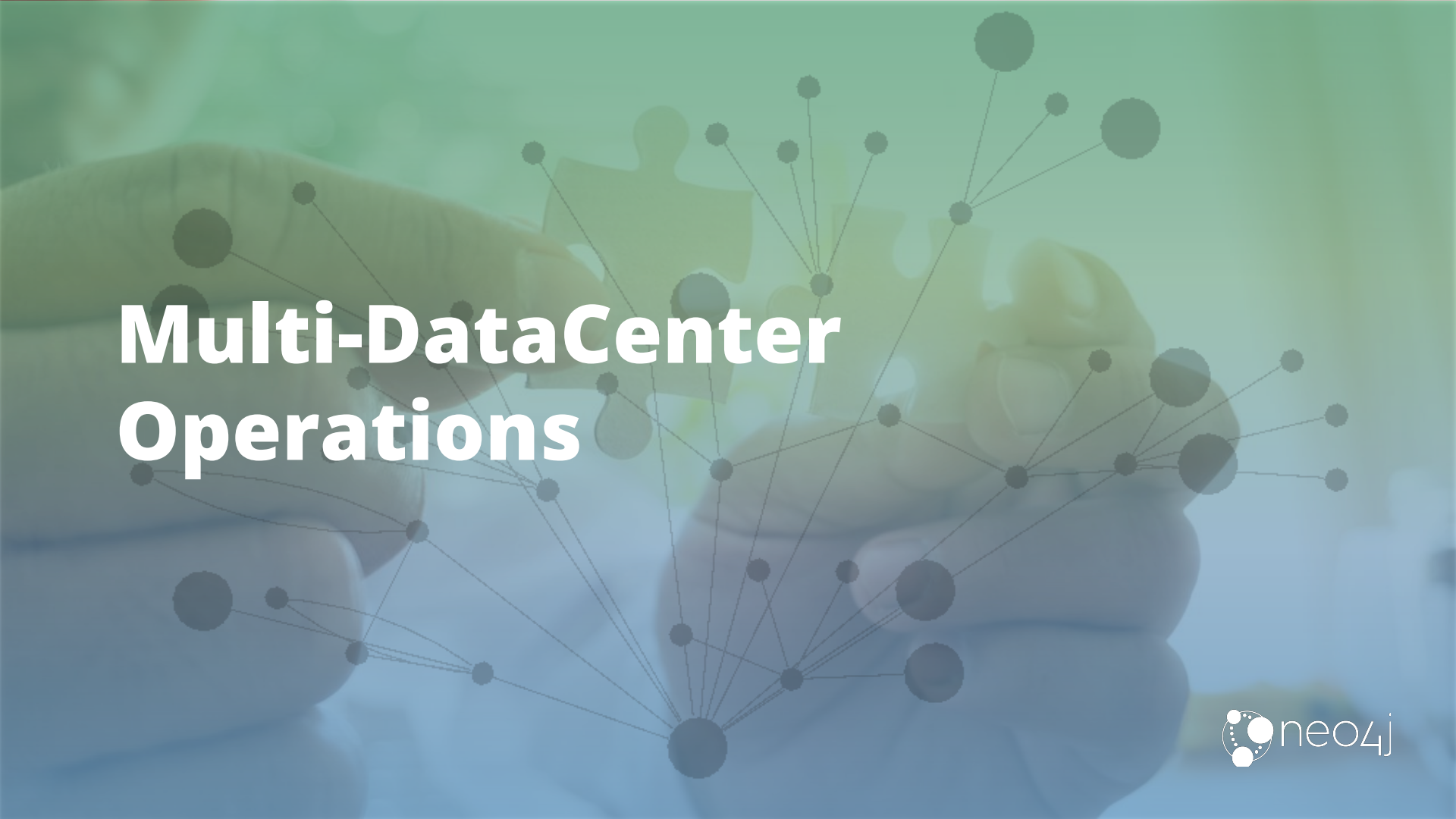
Fault tolerance and safety (cores) – multi DC
Quorum commit - (Bad) Deployment example



1. Assume heavy write load
2. Quorum commit -> committed when US cores have appended.
3. Asia falls behind…
4. Asia eventually require store copy -> fault tolerance reduced.

Conclusion:

- Close cores = fast system!
- Non-equal conditions for cores may indirectly reduce fault tolerance.
- Write is only possible on leader, no reason to distribute cores. Use Read Replicas!
- Leader switch is not a problem, but consecutive leader switches may indicate a problem with the system. Often GC, perhaps a network problem.
- What about disasters? Flooding, power loss etc?


Multi-DataCenter
Operations
neo4j

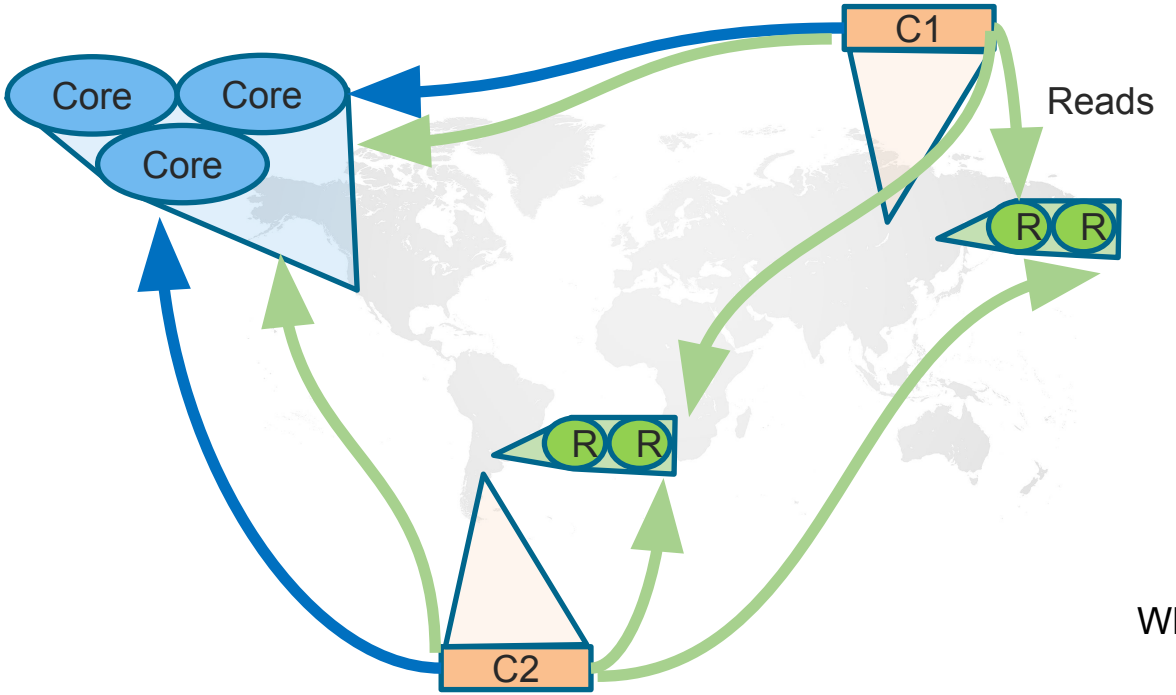
Multi-Data Center Licensing

- Implementing a cluster across multiple datacenters does NOT need any special licensing.
- However, some use cases present high needs for availability, redundancy, locality of client applications, or simply scale. In these cases it is important that the cluster is aware of its physical topology so that it can optimize for workload. *(i.e causal_clustering.multi_dc_license=true)*
- Defining hierarchies of local subclusters to speed the delivery of updates from the cluster core all the way to its most distant replica by maximizing the opportunity for local core-to-replica and replica-to-replica updates.
- To subdivide clusters such that specifics workload is directed only to the desired data center(s). This is useful for applications that need to use one or more data centers for write activity, and another one or more data centers to support read workload.

Multi-Data Center Operations

- Server Groups
  - Membership can map to data centers, availability zones
  - Each server in a Causal Cluster can belong to zero or more server groups. Server Groups can also overlap.
  - Example: *causal_clustering.server_groups=us,us-east*
- Strategy Plugins - Sets of rules that define how RR's contact servers in the cluster pull transaction logs
  - Pre-defined Strategies
    - Example: *causal_clustering.upstream_selection_strategy=connect-randomly-to-server-group,typically-connect-to-random-read-replica*
  - Configuring user-defined strategies
    - *causal_clustering.upstream_selection_strategy=user-defined*
    - *causal_clustering.user_defined_upstream_strategy=groups(north2); groups(north); halt()*
  - Building upstream strategy plugins using Java - Configure in arbitrary ways: load, subnet, machine size, or anything else accessible from the JVM

Deployment Example - Default

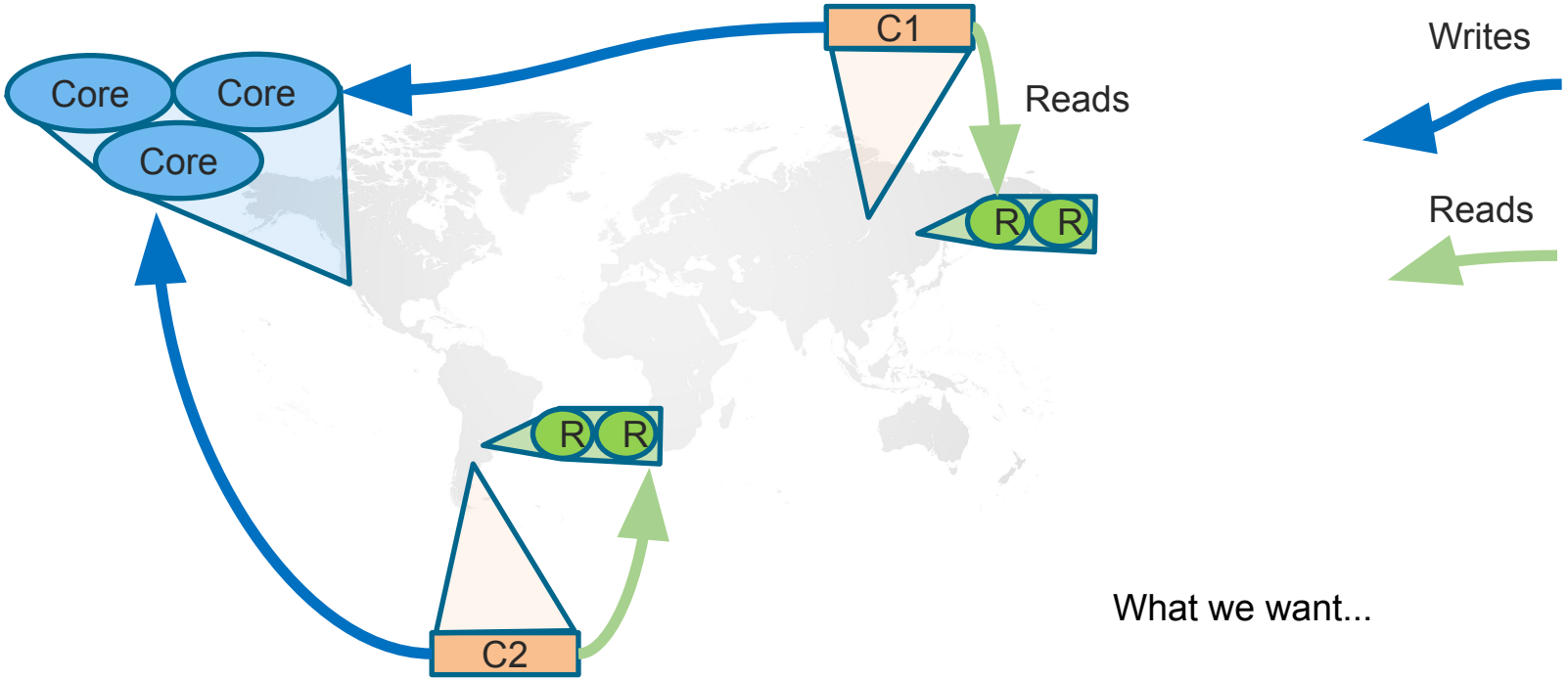


Writes

Reads

What we get by default!

Deployment Example - Configured routing via multi-DC option



Example - Configured routing

- Configure server groups:
  - South america:

    causal_clustering.multi_dc_license=true // Don't forget!
    causal_clustering.server_groups=SOUTH_AMERICA // Create a Server group, multiple groups are allowed!
    causal_clustering.load_balancing.config.server_policies.south_america=groups(SOUTH_AMERICA) // create a load balancing policy for south america (will fallback to default group if south america becomes unavailable)
  - Central asia

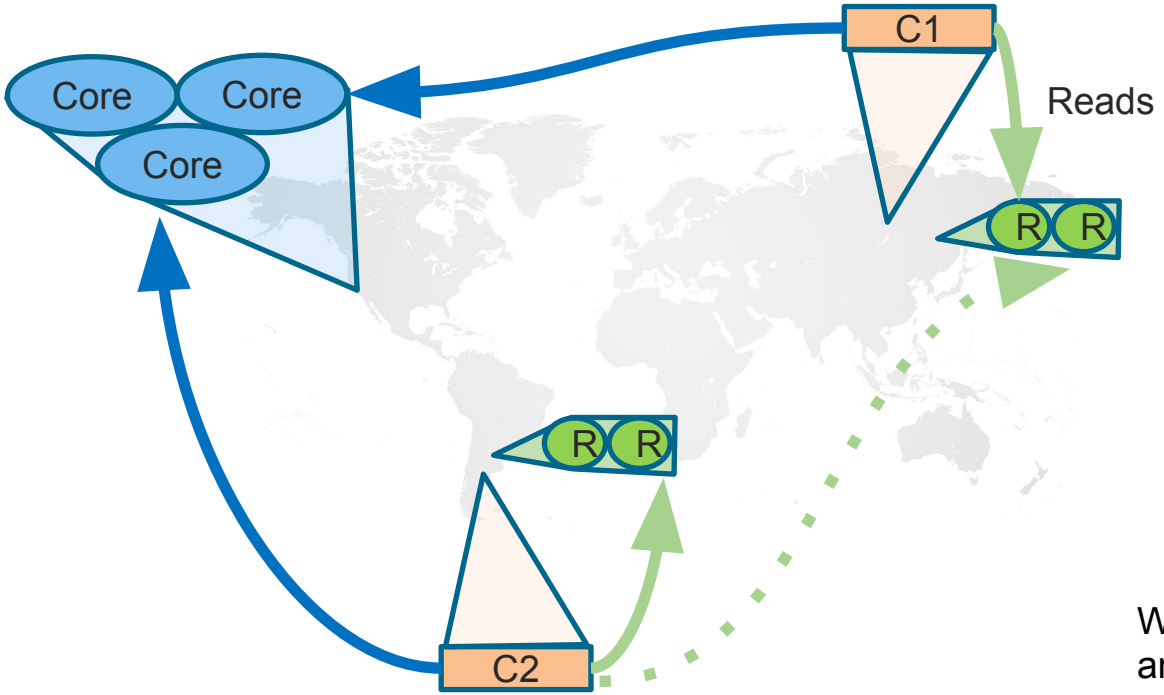
    causal_clustering.multi_dc_license=true
    causal_clustering.server_groups=EAST_ASIA
    causal_clustering.load_balancing.config.server_policies.south_america=groups(EAST_ASIA)
- Configure driver

```
// South America
Driver southAmericaDriver = GraphDatabase.driver( "bolt+routing://server1:7687?policy=south_america" );

// East Asia
Driver eastAsiaDriver = GraphDatabase.driver( "bolt+routing://server2:7687?policy=central_asia" );
```

https://neo4j.com/docs/operations-manual/current/clustering-advanced/multi-data-center/load-balancing/

Example - Configured routing



Writes

Reads

Extra heavy reads

What if some extra heavy analytical transaction should go to east asia because machines there are bigger.

Example - Configured routing

- Configure server groups:
  - South america is same as before
  - Central asia, lets add an extra policy

```
causal_clustering.multi_dc_license=true
causal_clustering.server_groups=EAST_ASIA
causal_clustering.load_balancing.config.server_policies.east_asia=groups(EAST_ASIA)
causal_clustering.load_balancing.config.server_policies.awesome_machines=groups(EAST_ASIA);halt()
```

// **halt()** ensures that we do not fall back to default policy if the specified policy is unavailable. Use with extreme caution, think of all the fault tolerance issues.

- Configure drivers. We create a separate driver for the new policy

```
// South America
Driver southAmericaDriver = GraphDatabase.driver( "bolt+routing://server1:7687?policy=south_america" );
Driver analyticalDriver = GraphDatabase.driver( "bolt+routing://server1:7687?policy=awesome_machines" ); // this is new

// East Asia
Driver eastAsiaDriver = GraphDatabase.driver( "bolt+routing://server2:7687?policy=central_asia" );
```

Disaster Recovery Scenarios



DR Deployment Option 1 – Three DCs

- Topology 1+1+1
  - DC1-A – 1 core
  - DC2-A – 1 core
  - DC3-B – 1 core

- Pros
  - Use 3 instances – No additional instances needed
  - If single instance/DC goes offline
    - *Cluster continues operating*
    - *No Manual Intervention required*

- Cons
  - If 1 DC is in a different region
    - *latency can be an issue to reach quorum after 1 failure in the main region*
  - If an entire region(DC1/DC2) goes offline
    - *Remaining instance will become read-only*
    - *To restore full cluster capabilities, you will need to spun up two new instances and seed them with a copy of the surviving instance, which depending on how big the database is, could take more than one hour to copy/configure databases in this fashion to seed*
    - *Chance for small data-loss in DC3 – as no guarantee that it has received updates from 2 instances in failed region*

DR Deployment Options 2 – Three DCs

- Topology 1+1+1 & 2 RR
  - DC1-A – 1 core
  - DC2-A – 1 core
  - DC3-B – 1 core + 2 RR

- Pros
  - If single instance/DC goes offline
    - *Cluster continues operating*
    - *No Manual Intervention required*

- Cons
  - If 1 DC is in a different region
    - *latency can be an issue to reach quorum after 1 failure in the main region*
  - If an entire region(DC1/DC2) goes offline
    - *Remaining instance will become read-only,*
    - *The two RR's can be quickly configured to cores with scripted approach and full cluster functionality restored.*
      - ***Suitable for larger databases where recovery/seeding time is significant***
    - *Slight Chance for data-loss in DC3 – as no guarantee that it has yet received updates from 2 instances in failed region*

DR Deployment Options 3 – Three DCs

- Topology 1+1+2 & 1 RR
  - DC1-A – 1 core
  - DC2-A – 1 core
  - DC3-B – 2 cores + 1 RR

- Pros
  - If entire region(DC1/DC2) is offline
    - *No data loss as at least one instance in DC3 was part of the quorum (3 out of 4)*

- Cons
  - Latency can be an issue to reach quorum(3/4) between regions
  - If an entire region(DC1/DC2) goes offline
    - *Remaining instances in DC3 will become read-only,*
    - *The RR can be quickly be configured/promoted to core with scripted approach & full cluster functionality restored.*
      - ***Suitable for larger databases where recovery/seeding time is significant***

DR Options 4 – Three DCs

- Topology 1+1+1 & 2+2+2 RR
  - DC1-A – 1 core + 2 RR
  - DC2-A – 1 core + 2 RR
  - DC3-B – 1 core + 2 RR

- Pros
  - If single instance/DC goes offline
    - *Cluster continues operating*
    - *No Manual Intervention required*
  - Tolerate any simultaneous outage of 2 out of 3 DCs
    - *No data loss in the remaining DC*

- Cons
  - Latency can be an issue to reach quorum(2/3) between regions
  - If any two DCs go offline at the same time,
    - *Remaining instances in the remaining DC will become read-only,*
    - *The RRs in that DC can be quickly be configured/promoted to core with scripted approach & full cluster functionality restored.*
      - ***Suitable for larger databases where recovery/seeding time is significant***

DR Options 5 – Two DCs

- Topology 2+1
  - DC2-A – 2 core
  - DC3-B – 1 core

- If DC3 goes offline
  - No Manual Intervention required
- If an instance in DC2 is offline
  - latency can be an issue to reach quorum if DCs are in separate regions

- If DC2 goes offline
  - Remaining instance in DC3 will become read-only
  - Chance for small data-loss in DC3 – as no guarantee that it has received updates from 2 instances in DC2
  - Manual intervention required to seed new cluster - you will need to spun up two new instances and seed them with a copy of the surviving instance, which depending on how big the database is, could take more than one hour to copy/configure databases in this fashion to seed

DR Options 6 – Two DCs

- Topology 2+2
  - DC2-A – 2 core
  - DC3-B – 2 core

- If either DC2 or DC3 goes offline
  - No data loss in remaining DC
  - Remaining instances in DC will become read-only (due to loss of Quorum)
  - Manual intervention required – The 2 remaining instances can be quickly configured/prompted to a 2 node cluster with scripts.
- Latency will always be a factor in this configuration (quorum with 3 out of 4)

DR Options 7 – Two DCs

- Topology 2+2 & 1+1 RR
  - DC2-A – 2 core + 1 RR
  - DC3-B – 2 core + 1 RR

- If either DC2 or DC3 goes offline
  - No data loss in remaining DC
  - Upon a DC going offline, remaining instances will become read-only (due to loss of Quorum)
  - Manual intervention required – The 3 remaining instances can be quickly configured/prompted to a 3 node cluster with scripts.
    - Suitable for larger databases where recovery/seeding time is significant

•Latency will always be a factor in this configuration (quorum with 3 out of 4)

Design Considerations

- Major disruptions such as DR outages can cause irregular traffic patterns
- Plan for the anticipated load spike on the surviving datacenter
- Will all applications have DR support themselves – if only a few will actually fail over, then you might be ok with lower capacity
- As to bringing services online in the DR datacenter, there is a chance the system might get initially flooded as everyone queries for updates and refreshes their data
- **Summary on latency versus scalability:**
    - Issuing read queries to a Core Server generally has the lowest latency in principle but may have the highest contention.
    - Issuing read queries to a Read Replica topologically closest to Core Servers typically has higher latency but also higher scalability.
    - Conversely, issuing read queries to a Read Replica topologically further from Core Servers typically has the highest latency but also the highest scalability.


Thank You!
neo4j