redislabs
home of redis

Active-Active Geo-Distributed Apps with Redis CRDTs (conflict free replicated data-types)

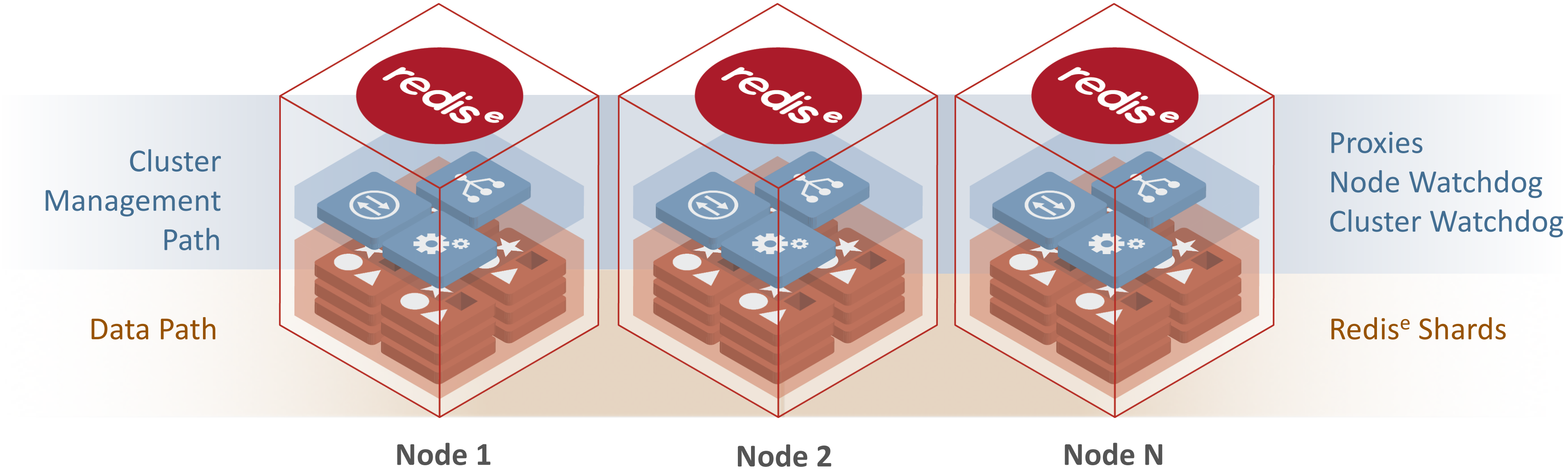
REDISCONF 2018

Agenda

- Quick Intro to Redis Enterprise
- Redis Enterprise Architecture
- Redis CRDTs
  - Introductions to Redis CRDTs
    - CAP Theorem and Redis CRDTs
  - How to develop apps with Redis CRDTs
    - Counters
    - Strings
    - Sets
    - Lists

Introduction to Redis Enterprise

Redis^e Cluster



Cluster Management Path

Data Path

Proxies
Node Watchdog
Cluster Watchdog

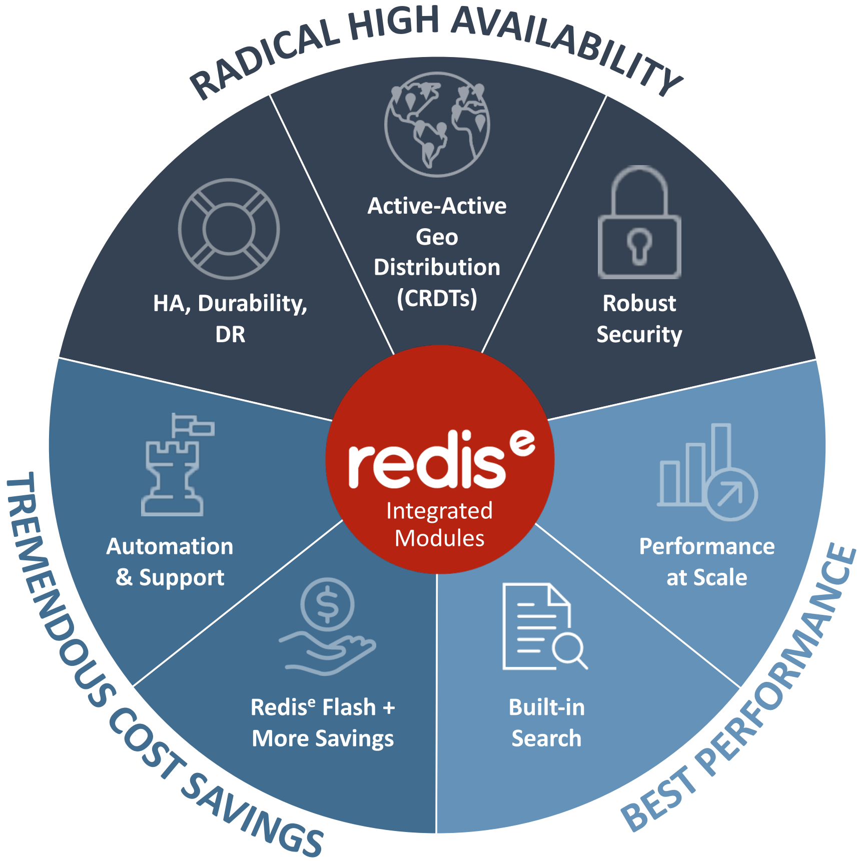
Redis^e Shards

Node 1

Node 2

Node N

Introduction to Redis Enterprise – cont.



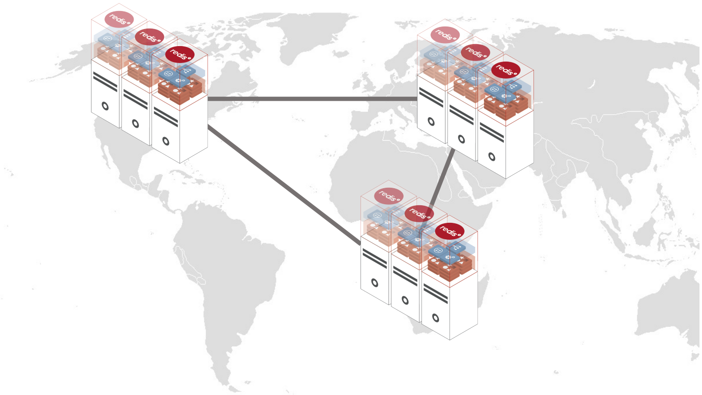
Replication Architecture

Replication Architecture with Redis Enterprise

Low Replication Lag & High Replication Throughput

- Local Replication: Built for LAN
  - Higher bandwidth
  - Lower latency
  - High quality links susceptible to fewer failures and retransmits

- Cross-Geo Replication: Built for WAN
  - Lower bandwidth
  - Higher latency
  - “Noisier” network quality susceptible to more failures and retransmits

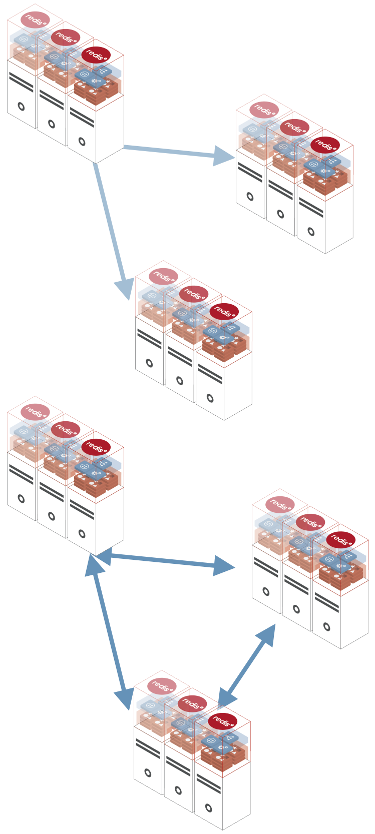


Cross-Geo Replication

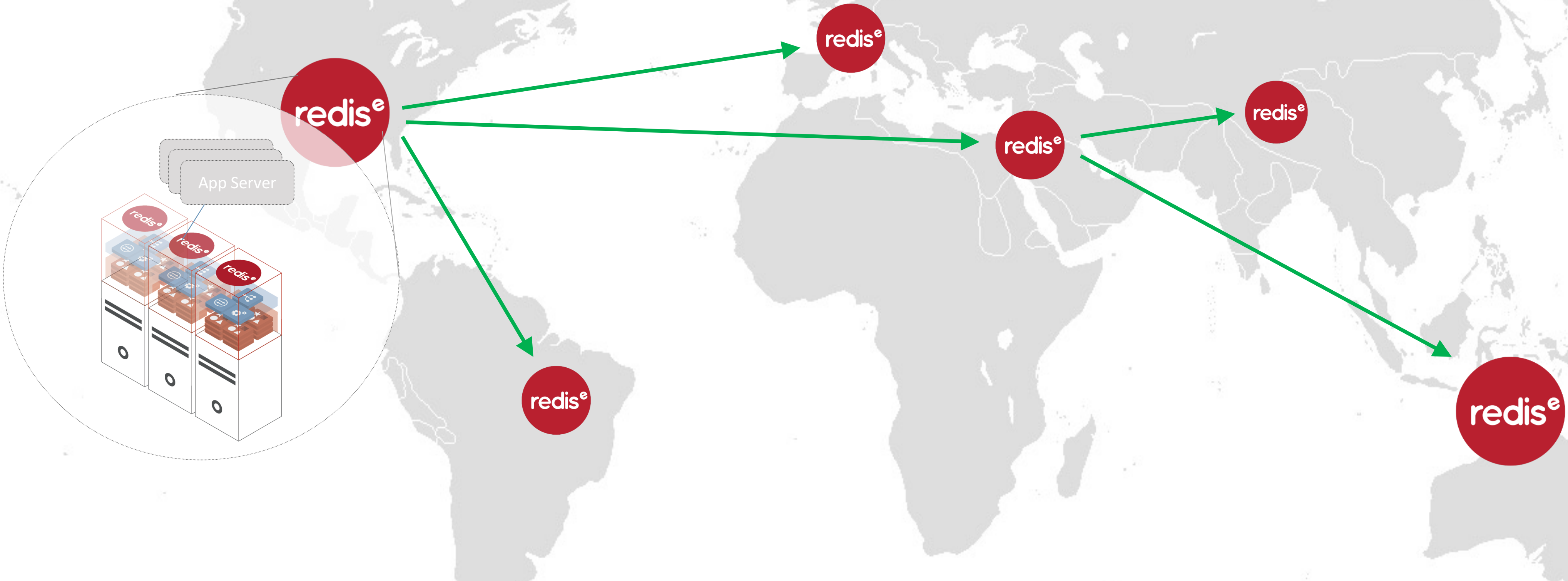
- Unidirectional Replication
  - *Replica Of* – Data Movement between Source to Destination DB
    - Content Distribution to Various Geographies for low latency, local reads
    - Continuous Data Transfer *to* and *from* Other Redis or Redis^{e} Databases

- Bi-Directional Replication
  - *Redis CRDTs* *(Conflict-free Replicated Data Tpes)* - Active-Active Writes & Reads
    - Advanced Protection against Regional Failures
    - Geo-Distributed Apps for Concurrent Active-Active Writes & Reads

Geo-Distributed Clusters



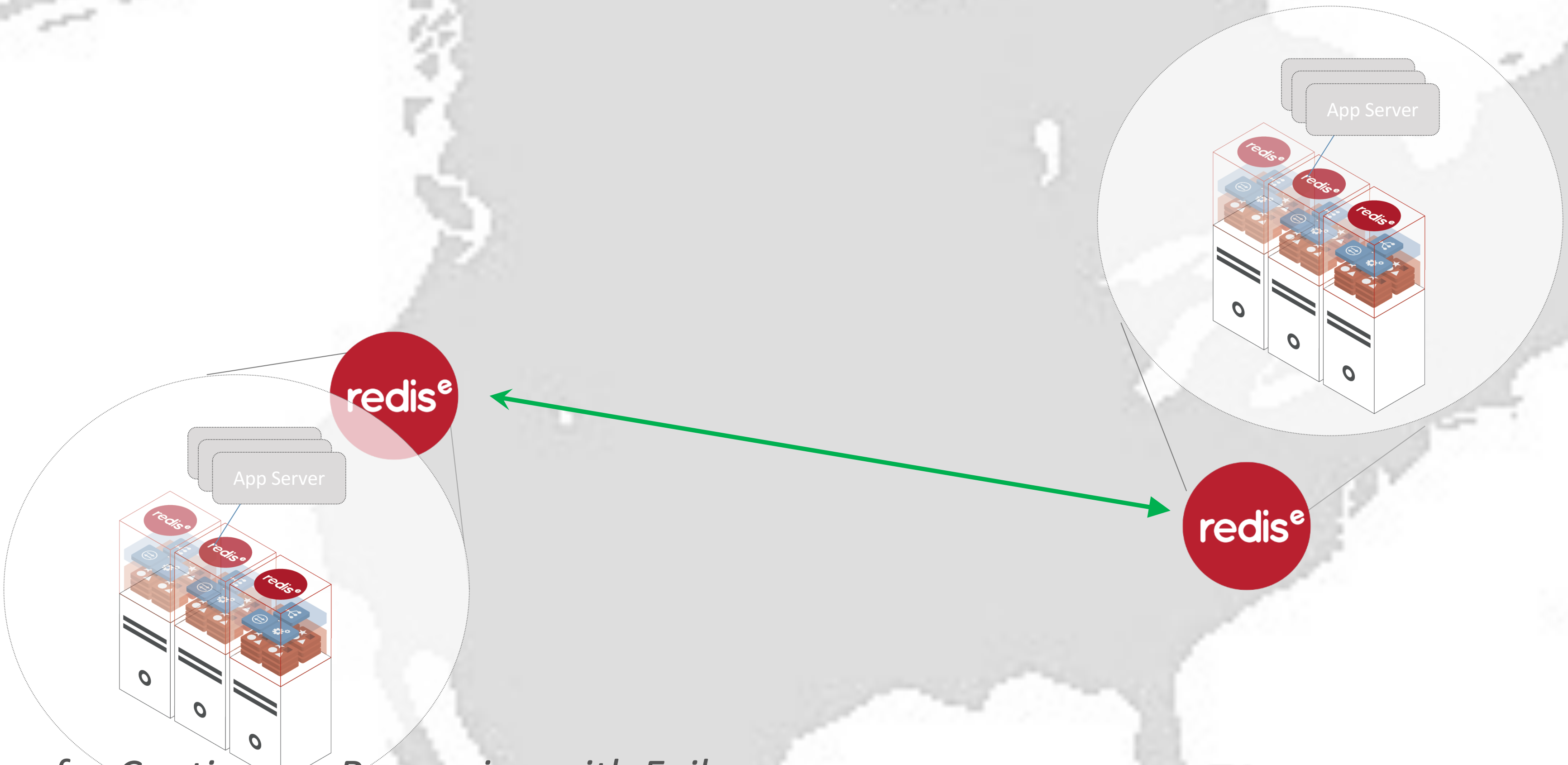
Replica Of: Geo Distribution for Fast Local Data Access



Geo Distribution for Local Data Access (CDN Like)

- Read local copy with low latency, instead of crossing borders
- Push updates to all regions with fast, memory based replication

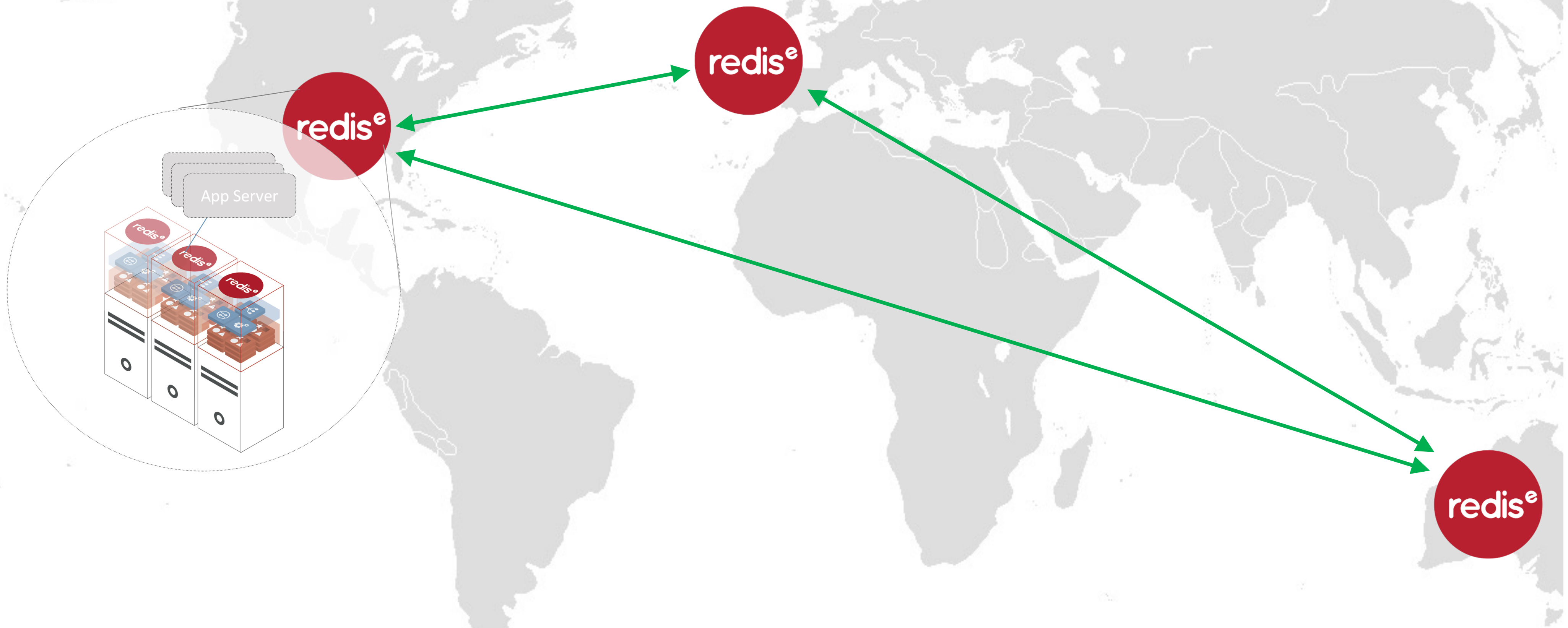
Redis CRDTs: Active-Active Geo Distribution for Geo-Failover



Geo Distribution for Continuous Processing with Failover

- Redis CRDTs for reliable handling of race conditions during geo-failover

Redis CRDTs: Active-Active Geo-Distribution



Active-Active Reads/Writes

Redis Conflict Free Replicated Data Types a.k.a CRDTs With Smart Conflict Resolution

Introducing Redis CRDTs

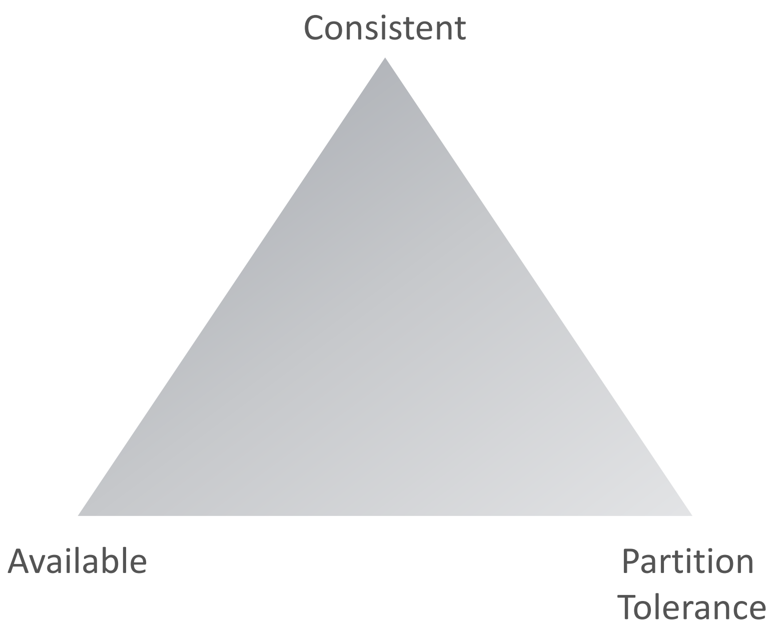
(Conflict-free Replicated Data Types)

redislabs

Active-Active Geo-Distributed Applications Are Hard!

- Consensus Driven Protocols – 2 Phase Commit
  - *Very chatty over LAN and WAN with very low throughput*
  - *Strictly “Consistent” thus not “Available” by nature*
  - *Products: Relational Databases*
- Quorum Based Writes and Reads
  - *Very chatty over LAN and WAN with very low throughput*
  - *Strictly “Consistent” thus not “Available” by nature*
  - *Products: DynamoDB, Cassandra*

Consistent but NOT Available



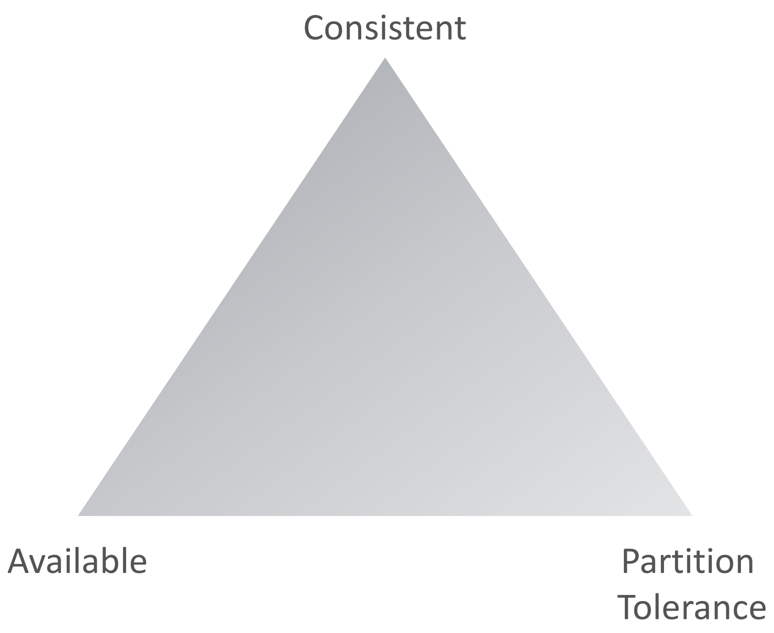
Active-Active Geo-Distributed Applications Are Hard!

- Consensus Driven Protocols – 2 Phase Commit
  - *Very chatty over LAN and WAN with very low throughput*
  - *Strictly "Consistent" thus not "Available" by nature*
  - *Products: Relational Databases*
- Quorum Based Writes and Reads
  - *Very chatty over LAN and WAN with very low throughput*
  - *Strictly "Consistent" thus not "Available" by nature*
  - *Products: Cassandra*

Consistent but NOT Available

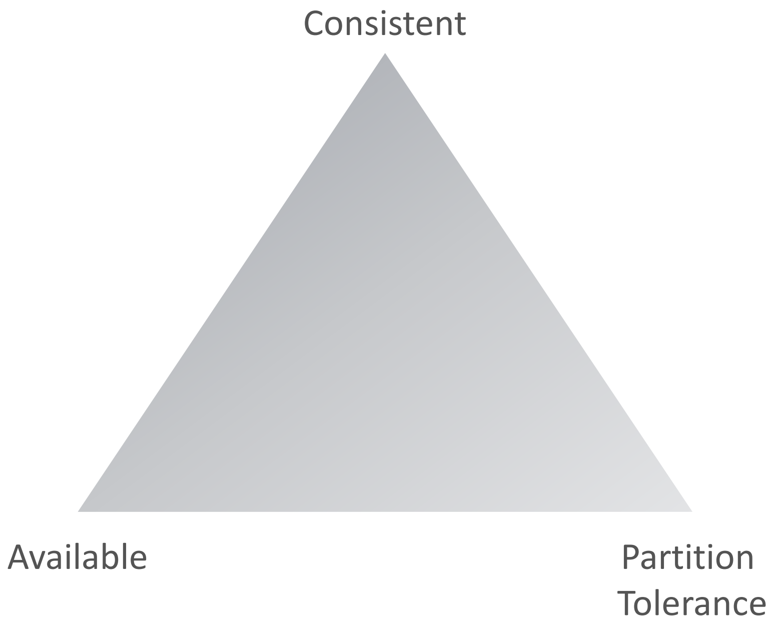
- LWW - Last Writer Wins Conflict Resolution
  - *Inaccurate and Insufficient for many apps (see next slide…)*
  - *Products: Couchbase*
- MVCC - Multi Version Concurrency Control
  - *High overhead and slow throughput due to large metadata*
  - *Complex to program against*
  - *Products: CouchDB*

Available but NOT Consistent



Introducing CRDTs (Conflict-Free Replicated Data Types)

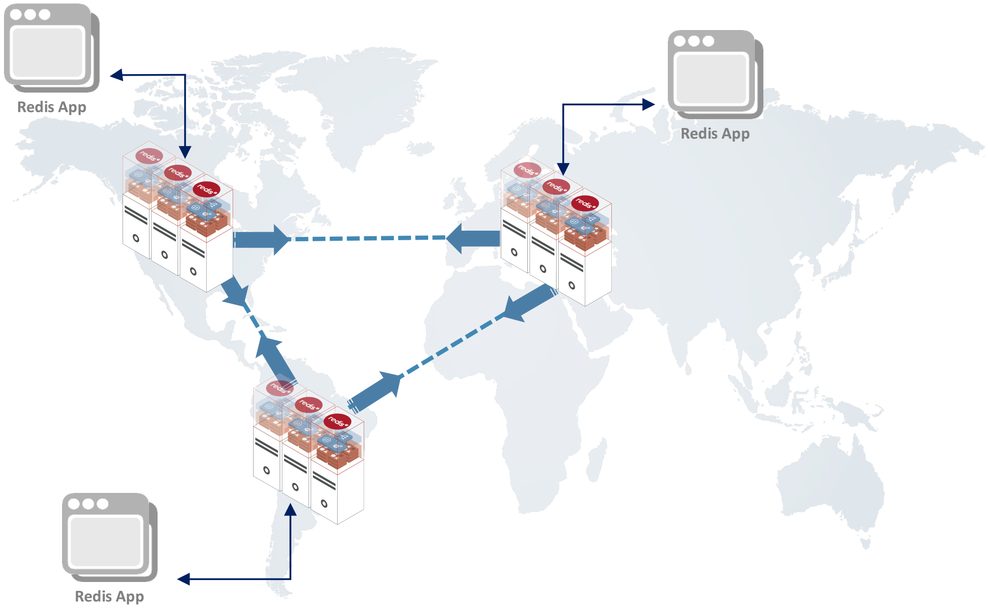
- **Consistency**: Strong eventual consistency + Causal Consistency
  - Eventually Consistent: Any two nodes that receive same but unordered updates will be in the same states
  - Causally Consistent: Events ordered across all participants
- **Availability**: Based on *consensus-free* protocol
  - Available: Local cluster availability is sufficient
  - No expensive multi-step consensus communication
- **Smart conflict resolution** with complex data types
  - Each Redis “type + method” expresses developer intent in resolving conflicts
- Based on proven **Mathematical Models**
  - Eric Brewer “*CAP Twelve Years Later: How the "Rules" Have Changed*”
  - Marc Shapiro “Strong Eventual Consistency and Conflict-free Replicated Data Types”



Active - Active Geo Distributed Apps with Redis CRDTs

Redis CRDTs

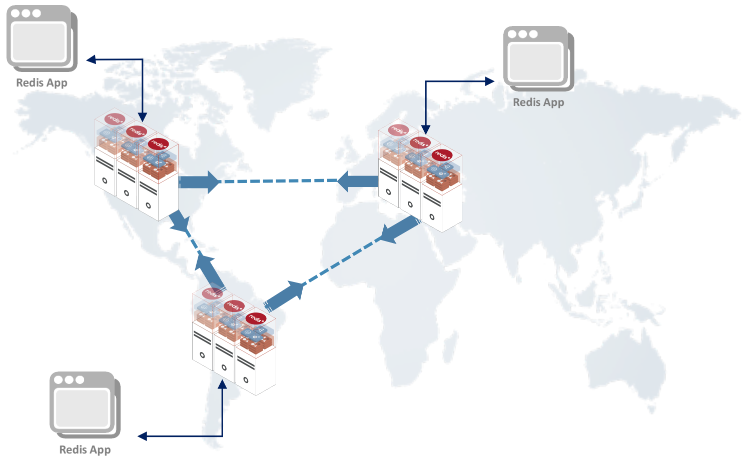
Active-Active Geo-Distributed Redis Apps with Smart Auto-Conflict Resolution and Simplified Development



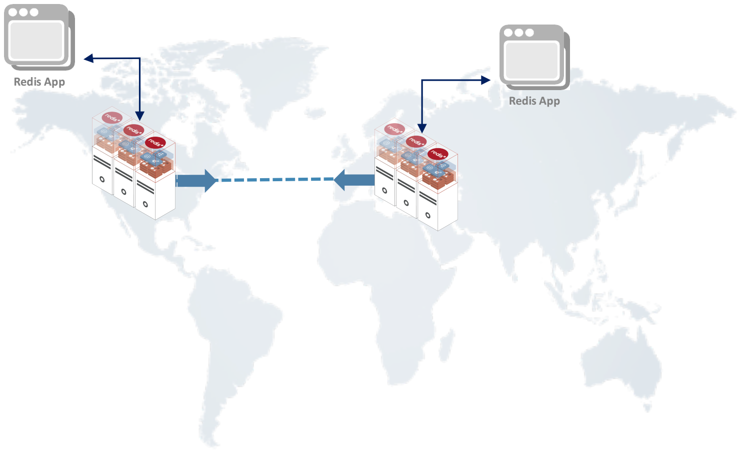
- Geo-Replicated Active-Active Deployments
- Smart & Transparent Conflict Resolution
- Based on CRDT Technology (*conflict free replicated data types*)

Redis CRDTs

- Geo-Replicated Active-Active Deployments
  - Global Database distributed across clusters in multiple regions
  - Fast bi-directional replication across data-centers
  - Low latency local reads & writes: *sub-millisecond latencies*



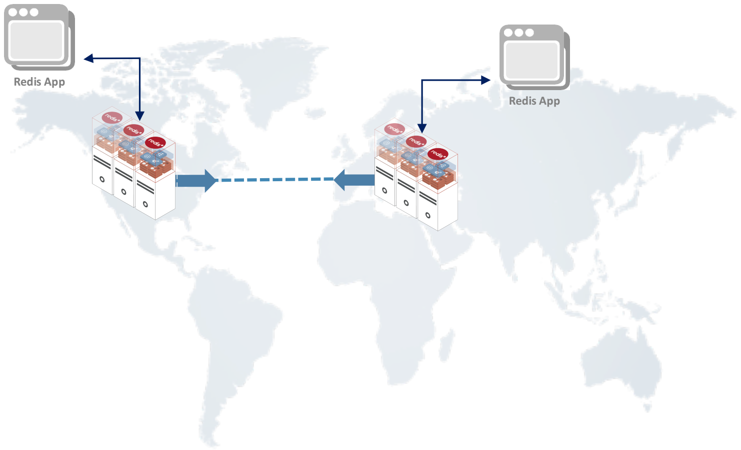
Redis CRDTs



- Smart & Transparent Conflict Resolution
  - CRDBs power concurrent writes across geographies
  - CRDBs handle failover conflicts safely!

time	US Data Center	EU Data Center
t1	SADD cart1 “costume”	

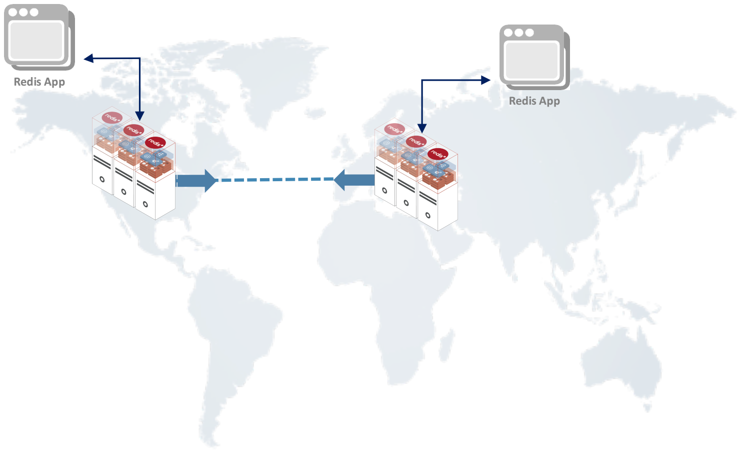
Redis CRDTs



- Smart & Transparent Conflict Resolution
  - CRDBs power concurrent writes across geographies
  - CRDBs handle failover conflicts safely!

time	US Data Center	EU Data Center
t1	SADD cart1 "costume"	
t2	US Data Center Fails – Sync Fails	

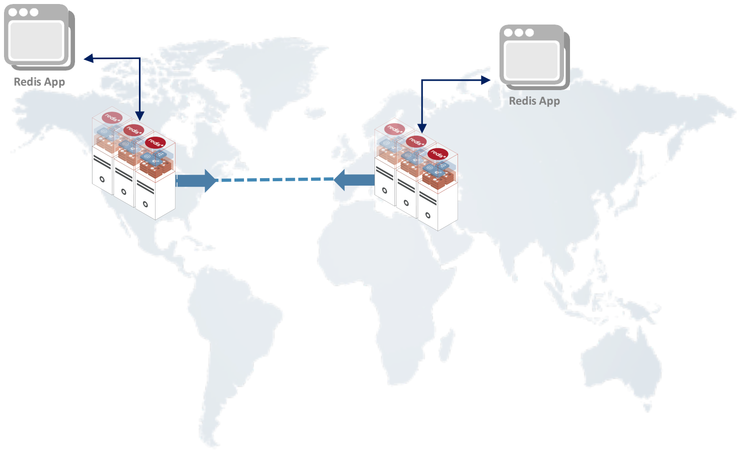
Redis CRDTs



- Smart & Transparent Conflict Resolution
  - CRDBs power concurrent writes across geographies
  - CRDBs handle failover conflicts safely!

time	US Data Center	EU Data Center
t1	SADD cart1 “costume”	
t2	US Data Center Fails – Sync Fails	
t3		SADD cart1 “mask”

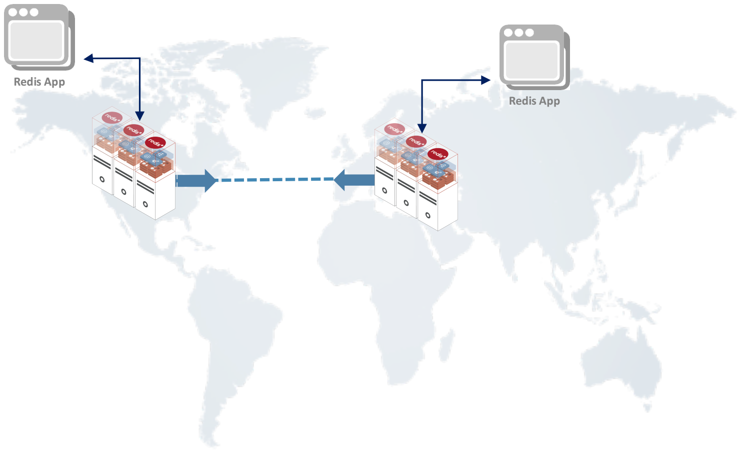
Redis CRDTs



- Smart & Transparent Conflict Resolution
  - CRDBs power concurrent writes across geographies
  - CRDBs handle failover conflicts safely!

time	US Data Center	EU Data Center
t1	SADD cart1 “costume”	
t2	US Data Center Fails – Sync Fails	
t3		SADD cart1 “mask”
t4	US Data Center Recovers – Resume Sync	

Redis CRDTs



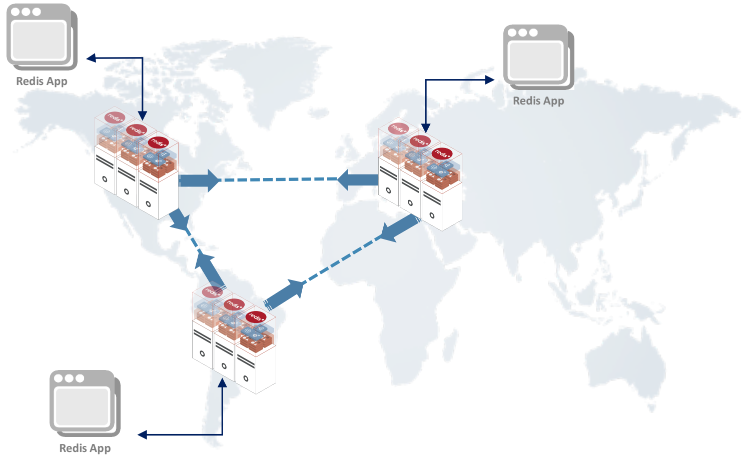
- Smart & Transparent Conflict Resolution
  - CRDBs power concurrent writes across geographies
  - CRDBs handle failover conflicts safely!

time	US Data Center	EU Data Center
t1	SADD cart1 “costume”	
t2	US Data Center Fails – Sync Fails	
t3		SADD cart1 “mask”
t4	US Data Center Recovers – Resume Sync	
t5	SMEMBERS cart1 “costume” “mask”	SMEMBERS cart1 “costume” “mask”

Redis CRDTs

- Based on CRDT Technology
  - Simple to develop with Redis Commands
  - Smarter-conflict resolution based on "developers intent"

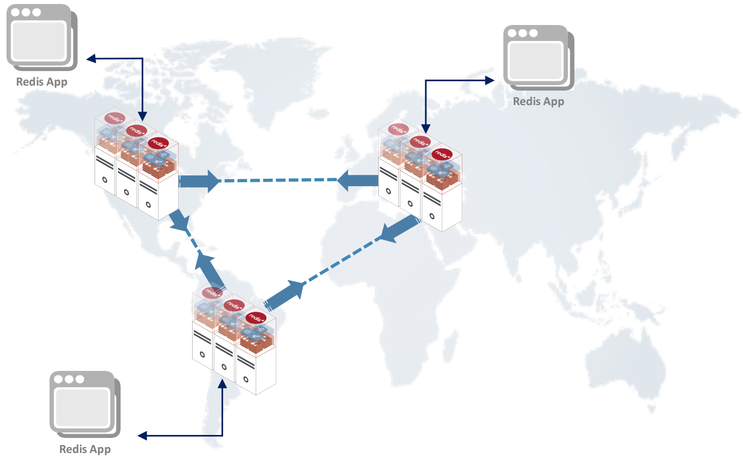
time	US Data Center	EU Data Center
t1	INCR TxCounter1	



Redis CRDTs

- Based on CRDT Technology
  - Simple to develop with Redis Commands
  - Smarter-conflict resolution based on “developers intent”

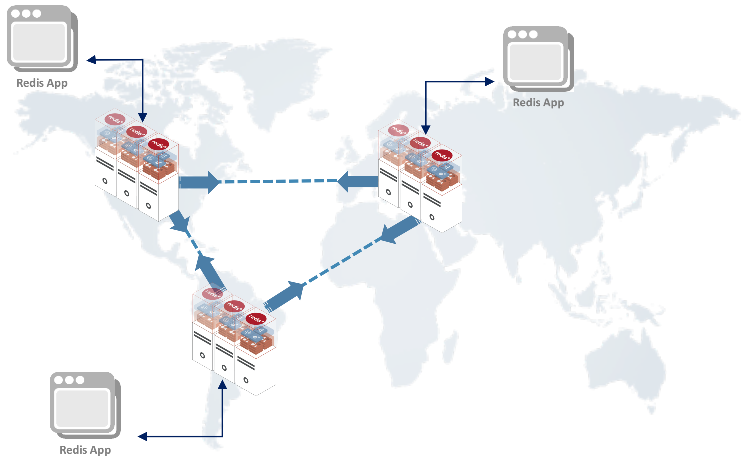
time	US Data Center	EU Data Center
t1	INCR TxCounter1	
t2		INCR TxCounter1



Redis CRDTs

- Based on CRDT Technology
  - Simple to develop with Redis Commands
  - Smarter-conflict resolution based on "developers intent"

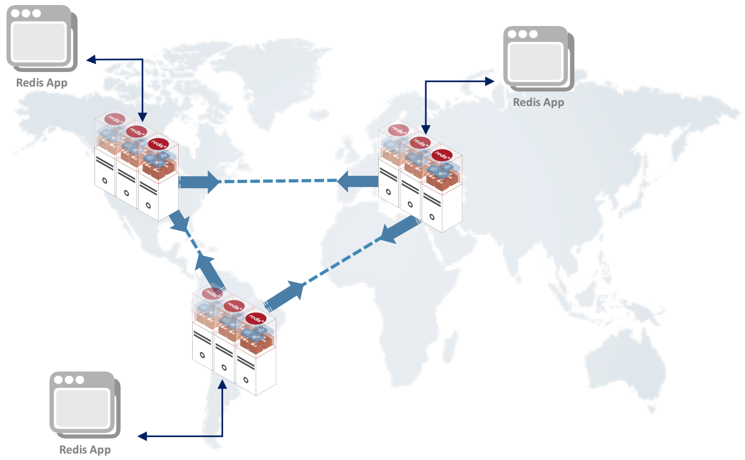
time	US Data Center	EU Data Center
t1	INCR TxCounter1	
t2		INCR TxCounter1
t3	INCR TxCounter1	



Redis CRDTs

- Based on CRDT Technology
  - Simple to develop with Redis Commands
  - Smarter-conflict resolution based on "developers intent"

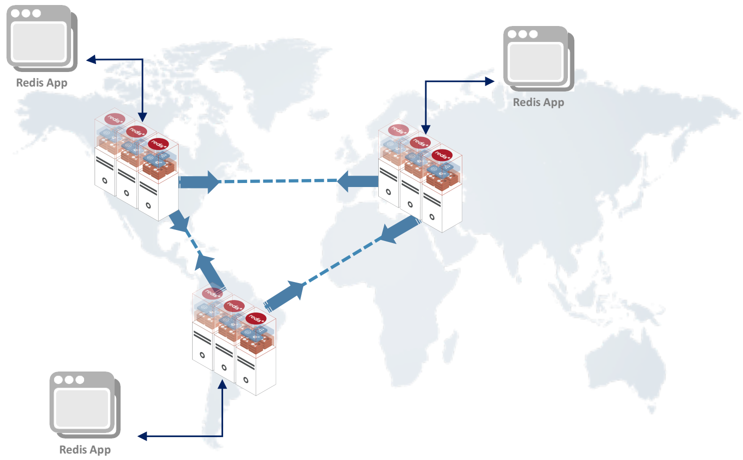
time	US Data Center	EU Data Center
t1	INCR TxCounter1	
t2		INCR TxCounter1
t3	INCR TxCounter1	
t4	Sync	



Redis CRDTs

- Based on CRDT Technology
  - Simple to develop with Redis Commands
  - Smarter-conflict resolution based on "developers intent"

time	US Data Center	EU Data Center
t1	INCR TxCounter1	
t2		INCR TxCounter1
t3	INCR TxCounter1	
t4	-- Sync --	
t5	GET TxCounter1 3	GET TxCounter1 3

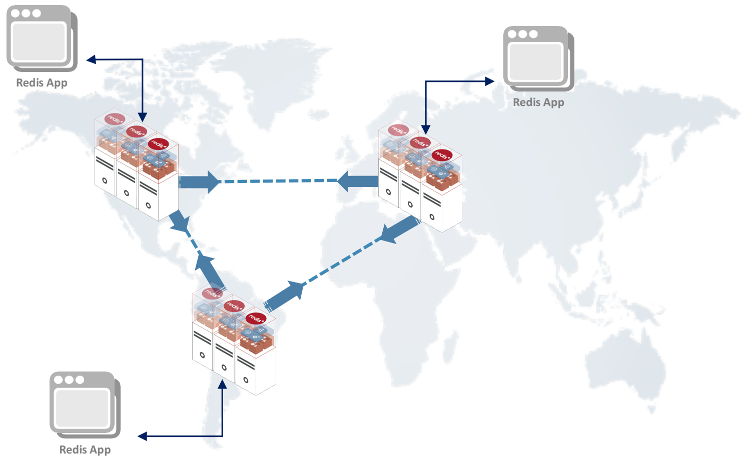


redislabs
home of redis

How Does Redis CRDTs Work?

Redis CRDTs == CRDBs (Conflict free replicated databases)

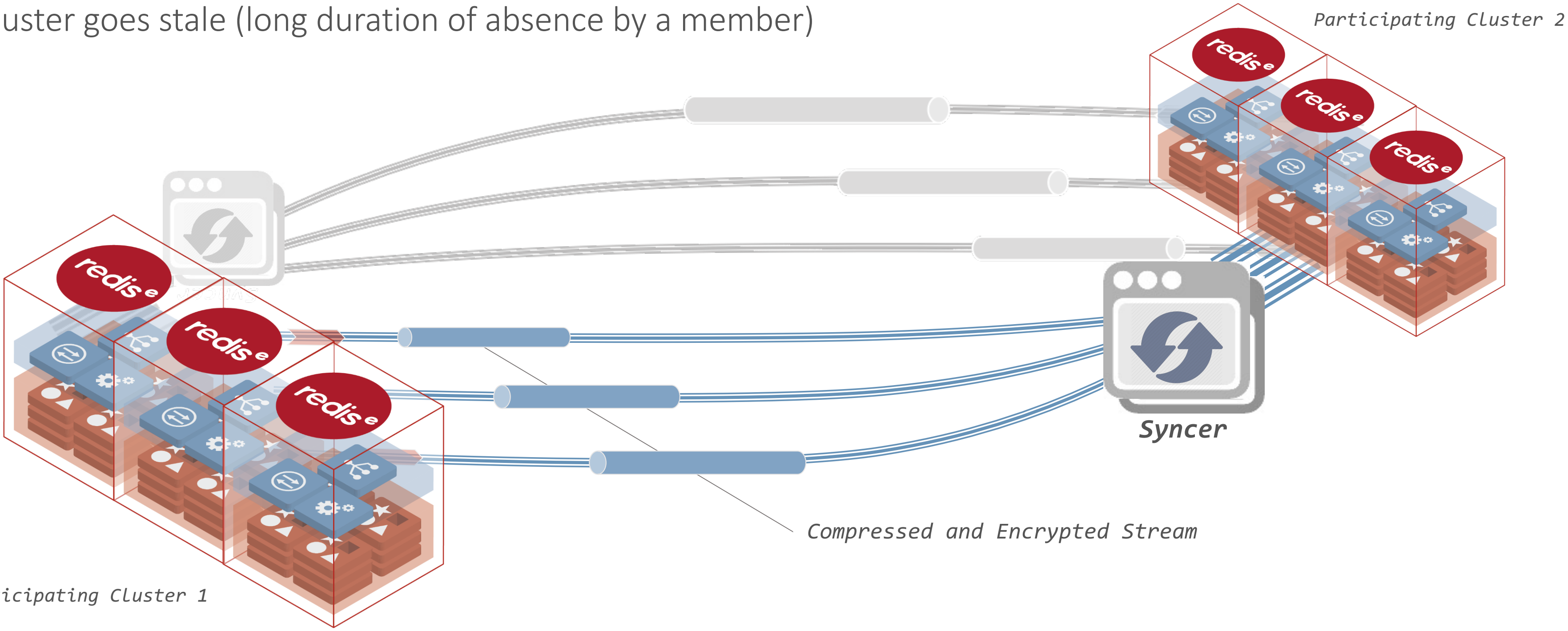
- CREATING a CRDB
  - Set global CRDB options
  - Initialize member CRDB on each participating cluster
    - In case of Error, Rollback
  - Establish bi-directional replication among all members



CRDB Architecture

Bi-directional Replication

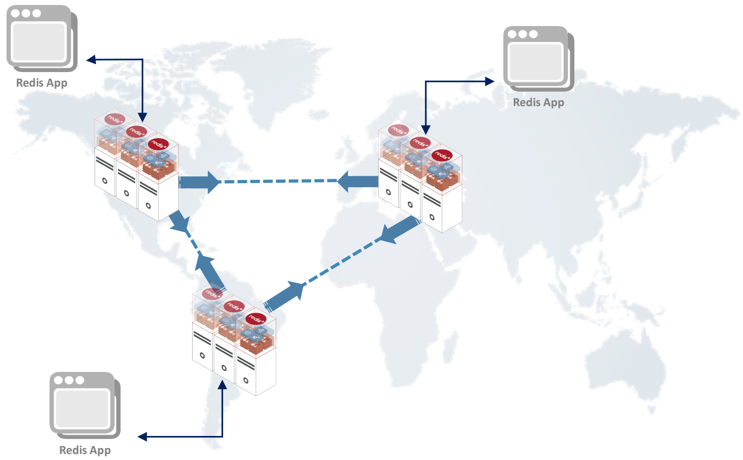
- *Syncer* uses replicas to replicate operations in a streaming fashion
- Resume-able replication under failures but may resync if a participating cluster goes stale (long duration of absence by a member)



How to Develop Applications with CRDBs?

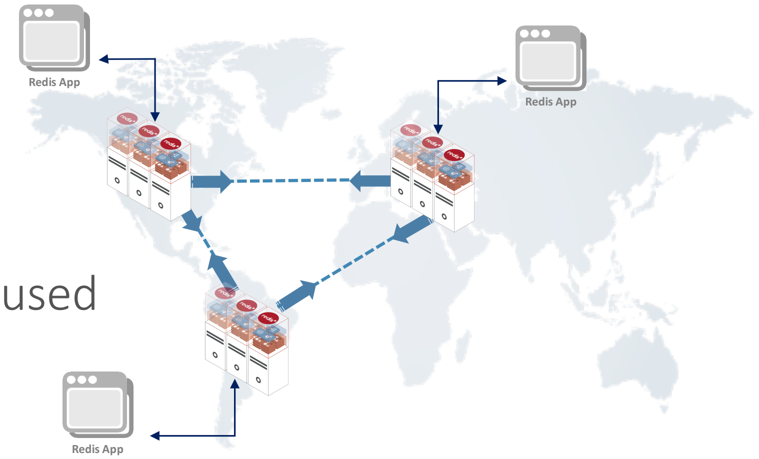
Developing Apps with a CRDB

- App Topology
  - All apps connect to the local member CRDB in the participating cluster
  - Reads/Writes occur on local member CRDB
    - Outcome of concurrent writes are predictable and based on a set of rules
  - Bi-directional replication automatically syncs changes
    - No application code required to resolve conflicts



Developing Applications with CRDBs

- Simply use Redis with Redis Client SDKs!
- Conflict Resolution is smart and automatic
  - Counters - using INCR or INCRBY, INCRBYFLOAT and so on
  - Sets & Sorted Sets – SADD/SREM and so on (new)
  - Hash – HSET/HDEL and so on
  - Strings - using SET/APPEND and so on
  - Lists – using LPUSH/POP and so on (new)
- Conflict Resolution is specific to data type + commands used
  - Total increments since last sync for Counters
  - Add wins with observed remove for Sets
  - Last writer wins for Strings
  - Counter or String semantics for Hash
  - Lists – using item IDs
  - And so on.



Conflicting Writes

- No conflicts here…

time	US Data Center	EU Data Center
t1	SET key1 “value1”	
t2	-- Sync --	
t3		SET key1 “value2”
t4	-- Sync --	
t5	SET key1 “value3”	GET key1 => value2
t6	-- Sync --	
t7	GET key1 => “value3”	GET key1 => “value3”

redislabs
home of redis

Counters in CRDBs

REDIS CRDTS IN ACTION!

Counters

- Counter: New numeric type in CRDTs
  - #type returns Counter
- Value is 5 bit shorter, i.e.
  - Redis supports <= 9,223,372,036,854,775,807 (64 bit signed)
  - CRDB supports <= 288,230,376,151,711,743 (59 bit signed)

Conflicting Writes with Counters - Truly Conflict Free

- Counter value is the SUM() of all operations

time	**US Data Center**	**EU Data Center**
t1	INCRBY key1 10	INCRBY key1 50
t2	-- Sync --	
t3	GET key1 => 60	GET key1 => 60
t4	DECRBY key1 60	INCRBY key1 60
t5	-- Sync --	
t6	GET key1 => 60	GET key1 => 60

Delete vs Update Conflict - Observed Remove

- Conflicting Update vs Delete
  - DEL logically resets the counter.

time	US Data Center	EU Data Center
t1	INCRBY key1 100	
t2	-- Sync --	
t3	DEL key1	INCRBY key1 10
t4	-- Sync ---	
t5	GET key1 => 10	GET key1 => 10

redislabs
home of redis

Strings in CRDBs

REDIS CRDTS IN ACTION!

String Add vs Delete - Add Wins

- Concurrent APPEND and DEL.
  - Add (including Update) Wins.

time	US Data Center	EU Data Center
t1	SET key1 “Hello”	
t2	-- Sync ---	
t3	APPEND key1 “There”	
t4		DEL key1
t5	-- Sync --	
t6	GET key1 => “HelloThere”	GET key1 => “HelloThere”

Key Concurrent Expiration – Longer TTL Wins

- Concurrent Expiration
  - Longer TTL wins (*Non volatile keys have infinite TTL*)

time	US Data Center	EU Data Center
t1	SET key1 “val1”	
t2	-- Sync ---	
t3	EXPIRE key1 10	EXPIRE key1 30
t4	-- Sync --	
t5	TTL key1 => 30	TTL key1 => 30

String APPEND vs. APPEND

- Concurrent APPEND operations
  - LWW (last writer wins)

Time	US Data Center	EU Data Center
t1	SET key1 “Hello”	
t2	-- Sync ---	
t3	APPEND key1 “There”	
t4		APPEND key1 “World”
t5	-- Sync --	
t6	GET key1 => “HelloWorld”	GET key1 => “HelloWorld”

redislabs
home of redis

Sets in CRDBs

REDIS CRDTS IN ACTION!

Sets SADD vs SADD

- Concurrent SADD Operation
  - Preserve merged items

time	US Data Center	EU Data Center
t1	SADD cart1 “costume”	
t2	US Data Center Fails – Sync Fails	
t3		SADD cart1 “mask”
t4	US Data Center Recovers – Resume Sync	
t5	SMEMBERS cart1 “costume” “mask”	SMEMBERS cart1 “costume” “mask”

redislabs
home of redis

Hashes in CRDBs

REDIS CRDTS IN ACTION!

Hashes HSET vs HSET

- Concurrent HSET Operation
  - Preserve merged items

time	US Data Center	EU Data Center
t1	HSET k1 f1 “a”	
t2	-- Sync --	
t3	HSET k1 f2 “b”	
		HSET k1 f3 “123”
t4	-- Sync --	
t5	HGETALL k1 “f1” “a” “f2” “b” “f3” “123”	HGETALL k1 “f1” “a” “f2” “b” “f3” “123”

redislabs
home of redis

Lists in CRDBs

REDIS CRDTS IN ACTION!

Lists LPUSH vs LPOP

- Concurrent LPUSH vs LPOP Operation
  - Preserve merged items

time	US Data Center	EU Data Center
t1	LPUSH l1 “a” “b” “c”	
t2	-- Sync --	
t3	LPUSH l1 “d”	
		LPOP l1
t4	-- Sync --	
t5	LRANGE l1 -100 100 “b” “c” “d”	LRANGE l1 -100 100 “b” “c” “d”

redislabs
home of redis

PUB/SUB in CRDBs

REDIS CRDTS IN ACTION!

PUB/SUB and Eventual Consistency

- Multiple Publishers and Subscribers
  - Message order received can be different per subscriber

time	US Data Center	EU Data Center	ASIA Data Center
t1	PUBLISH c1 “a”	SUBSCRIBE c1	SUBSCRIBE c1
t2	-- Sync --		-- X --
t3	SUBSCRIBE c1	”a”	
		PUBLISH c1 “b”	
t4	-- Sync --		
t5	“b”		“b” “a”

PUB/SUB with Causal Consistency

- Multiple Publishers and Subscribers
  - Causal Consistency Ensure Ordering of Messages

time	US Data Center	EU Data Center	ASIA Data Center
t1	PUBLISH c1 “a”	SUBSCRIBE c1	SUBSCRIBE c1
t2	-- Sync --		-- X --
t3	SUBSCRIBE c1	”a”	
		PUBLISH c1 “b”	
t4	-- Sync --		
t5	“b”		“a” “b”

redislabs
home of redis

Demo!

REDIS CRDTS IN ACTION!

Recap

Active - Active Geo Distribution

Redis Enterprise with Redis CRDTs (Conflict Free Replicated Data-types)

Build Fast Active-Active Geo-Distributed Redis Apps

with Smart Auto-Conflict Resolution

- Geo-Replicated Active-Active Deployments
  - Fast bi-directional across data-centers
  - Low latency local reads & writes : sub-millisecond latencies
- Smart & Transparent Conflict Resolution
  - Build safe regional failover and geo-distributed apps
  - Each Redis Data Type have smart conflict resolution based on “developer intent”
- Based on CRDT Technology (conflict free replicated data types)
  - Simplifies development

Get Started with Redis CRDTs Today!

Redislabs.com

- Download Redis Enterprise 5.0

- Visit Docker Hub: *redislabs/redis:latest*