Aurora Multi-Master

Justin Levandoski

Agenda

- Aurora Fundamentals
- Aurora Multi-Master Architecture
- Aurora Multi-Master Use Cases

Aurora Fundamentals

Amazon Aurora:
A relational database reimagined for the cloud

- ☑ **Speed** and **availability** of high-end commercial databases
- ☑ **Simplicity** and **cost-effectiveness** of open source databases
- ☑ Drop-in **compatibility** with MySQL and PostgreSQL
- ☑ Simple **pay as you go** pricing

Delivered as a **managed** service

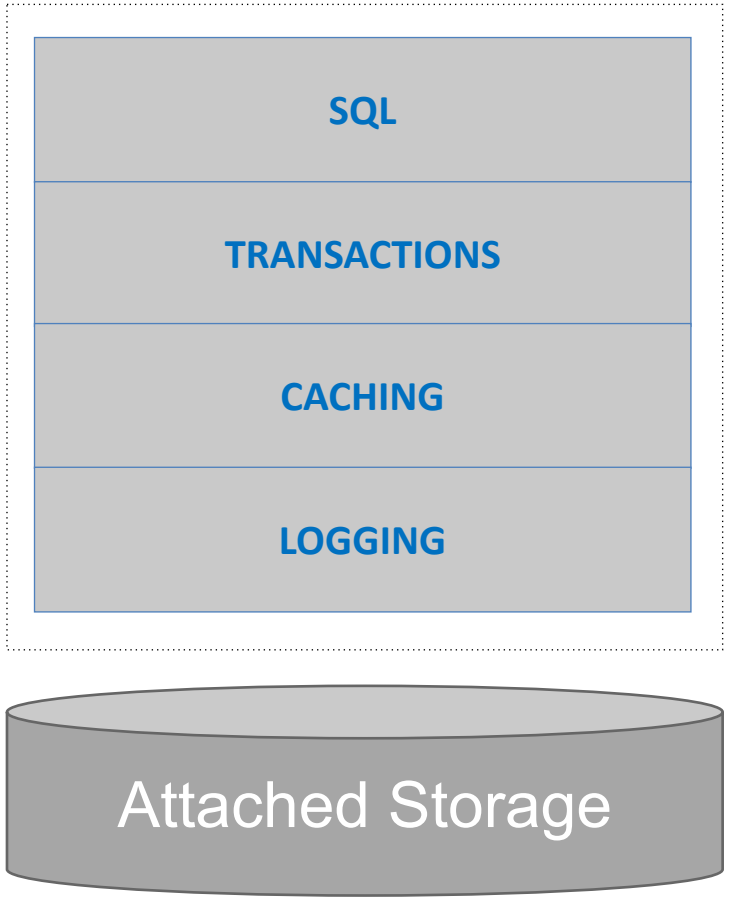
Relational databases were not design for the cloud

- Monolithic architecture
- Large failure blast radius

Databases in the cloud

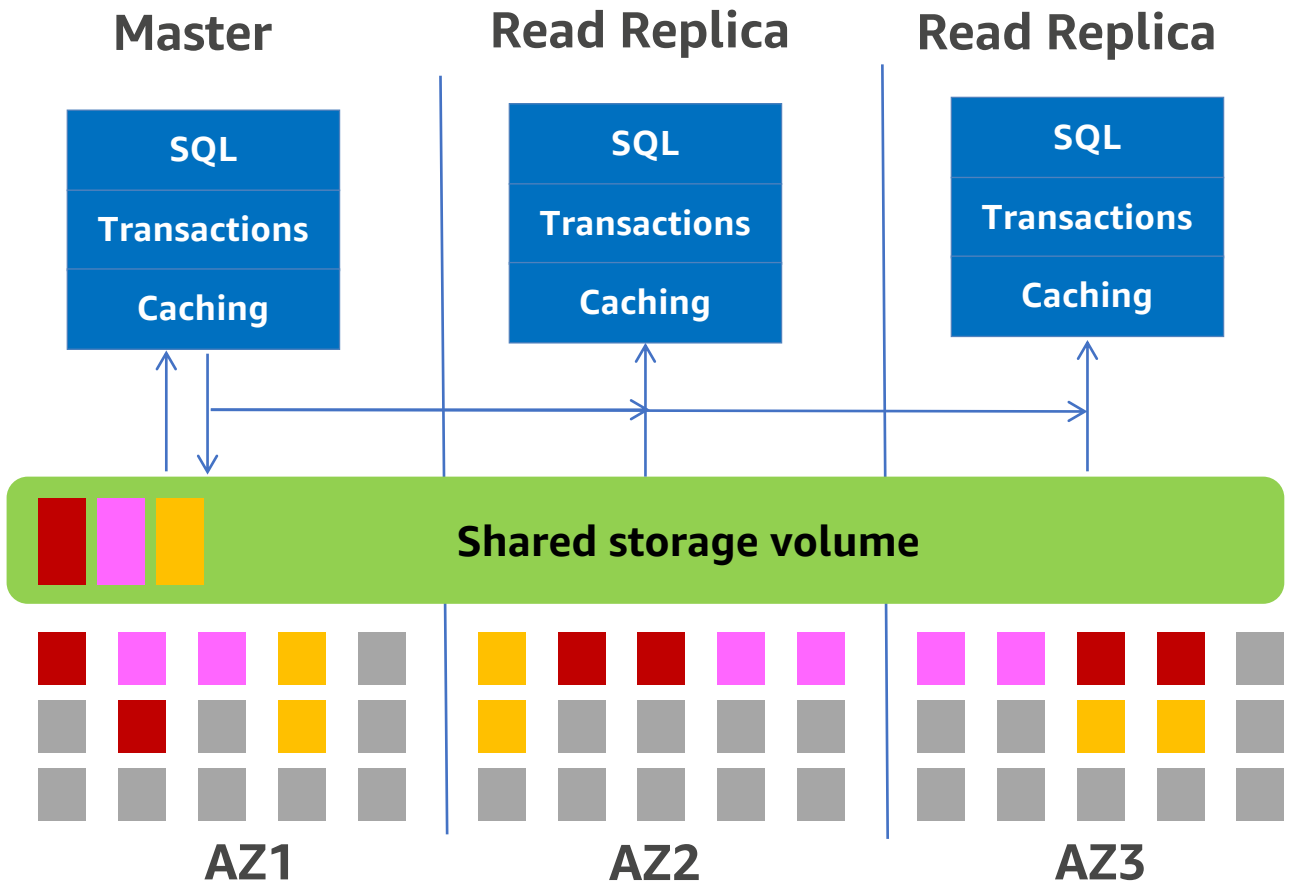
- Compute & storage have different lifetimes
- Instances fail/shutdown/scale up & down
- Instances added to a cluster

Compute & storage are best decoupled for scalability, availability, and durability

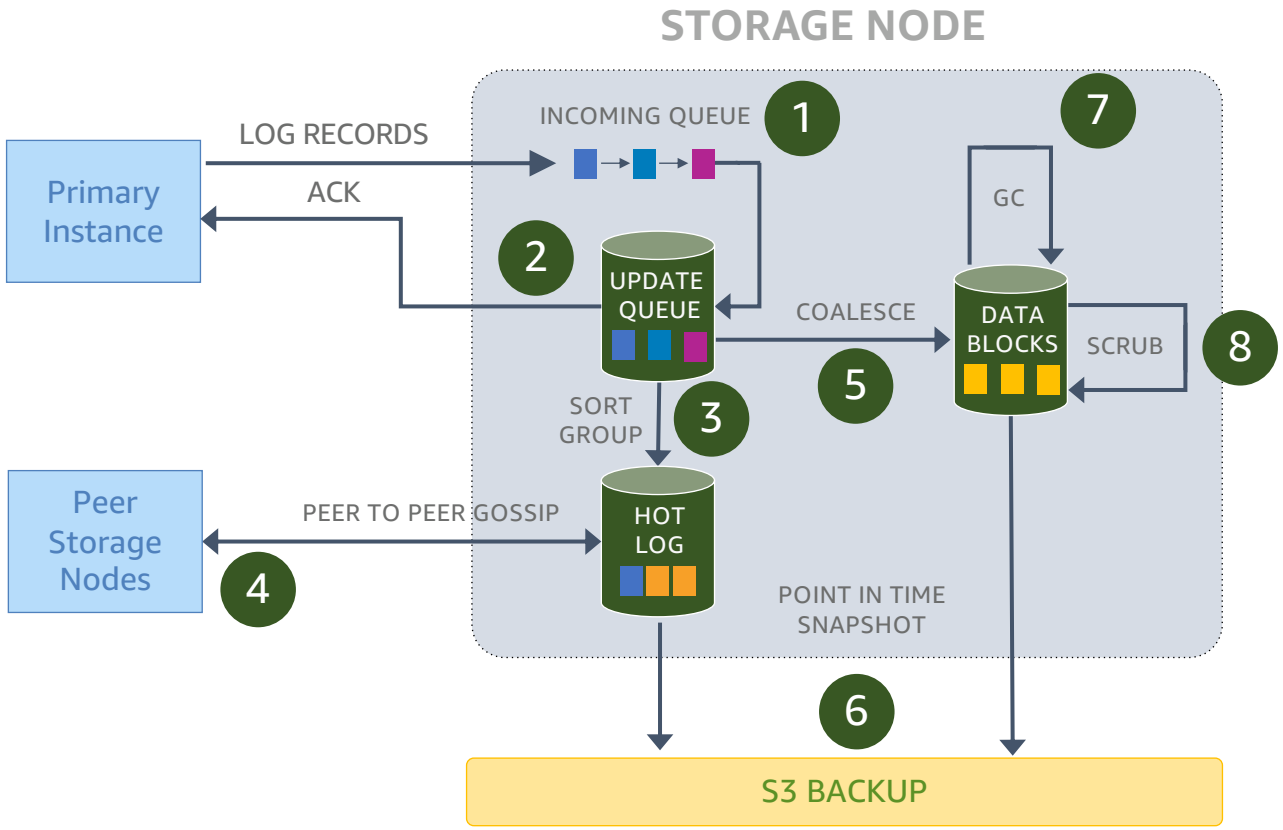


Aurora Architecture

- Database tier
  - Writes redo log to network
  - No checkpointing! *The log is the database*
  - Pushes log application to storage
  - Master replicates to read replicas for cache updates
- Storage tier
  - Highly parallel scale-out redo processing
  - Data replicated 6 ways across 3 AZs
  - Generate/materialize database pages on demand
  - Instant database redo recovery
- 4/6 Write Quorum with Local Tracking
  - AZ + 1 failure tolerance
  - Read quorum needed only during recovery



Aurora Storage Node



SUMMARY

Manages 10GB page segments
10GB = right size for repair/fault tolerance
Use fault tolerance for heat management/machine patching

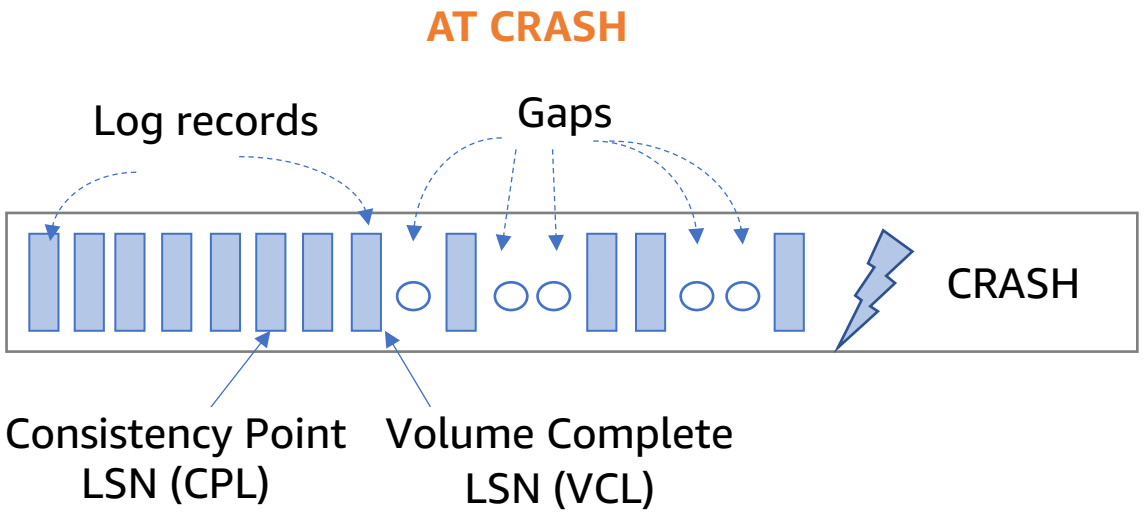
IO FLOW

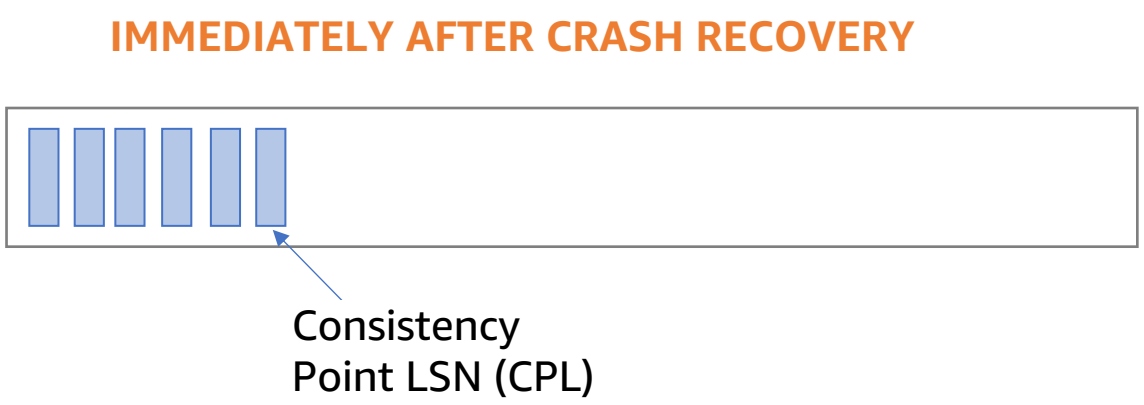
1. Receive record and add to in-memory queue
2. Persist record and ACK
3. Organize records and identify gaps in log
4. Gossip with peers to fill in holes
5. Coalesce log records into new data block versions
6. Periodically stage log and new block versions to S3
7. Periodically garbage collect old versions
8. Periodically validate CRC codes on blocks

OBSERVATIONS

All steps are asynchronous
Only steps 1 and 2 are in foreground latency path
Input queue is **46X** less than MySQL (unamplified, per node)
Favor latency-sensitive operations
Use disk space to buffer against spikes in activity

Crash Recovery





Storage establishes consistency points that increase monotonically + continuously returned to DB

Transactions commit once DB can prove all changes have met quorum

Volume Complete LSN (VCL) is the highest point where all records have met quorum

Consistency Point LSN (CPL) is the highest commit record below VCL.

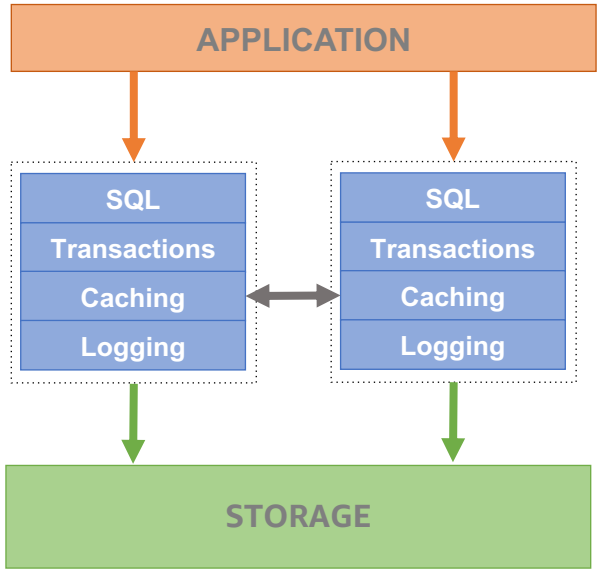
Everything past CPL is deleted at crash recovery

- Removes the need for 2PC at each commit spanning storage nodes.
- No redo or undo processing is required before the database is opened for processing

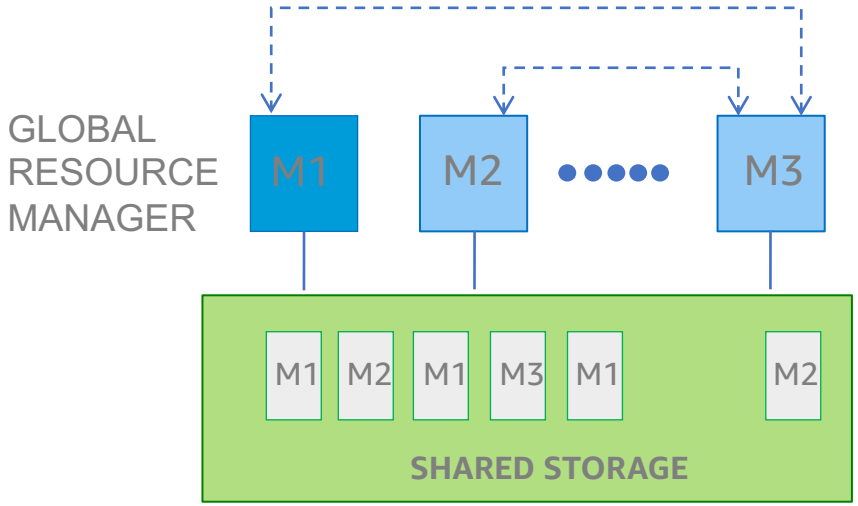
Aurora Multi-Master Architecture

Distributed Lock Manager

SHARED DISK CLUSTER



LOCKING PROTOCOL MESSAGES



Pros

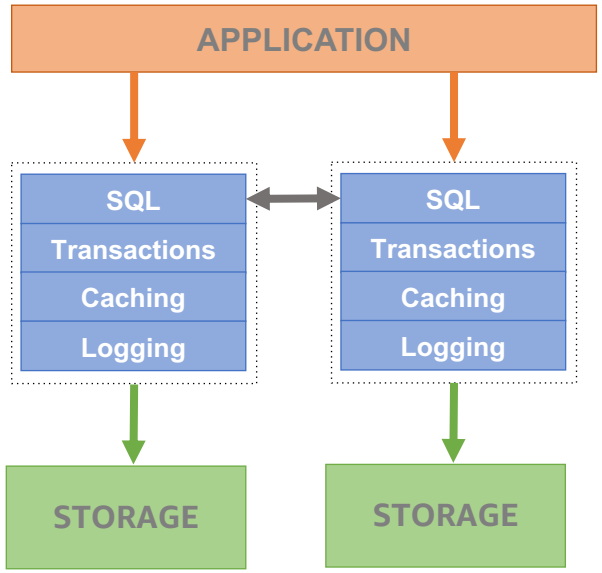
All data available to all nodes
Easy to build applications
Similar cache coherency as in multi-processors

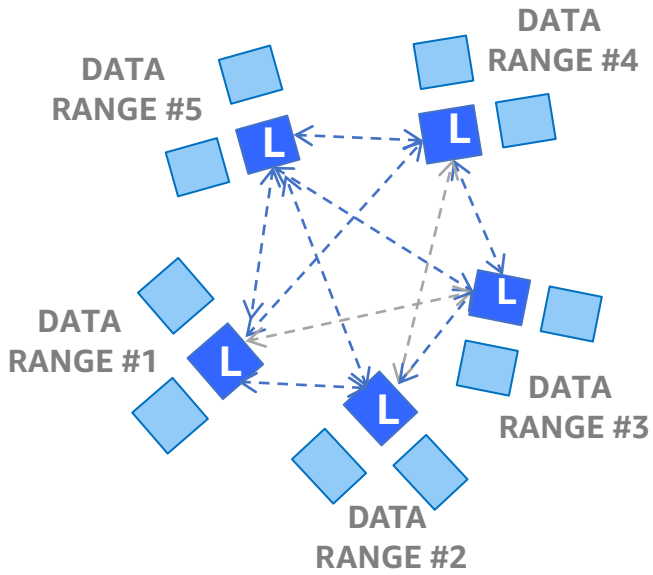
Cons

Heavyweight cache coherency traffic on per-lock basis
Networking can get expensive
Negative scaling with hot blocks

Partitioned with Consensus

SHARED NOTHING





Pros

Query broken up and sent to data nodes

Less coherence traffic – only for commits

Can scale to many nodes

Cons

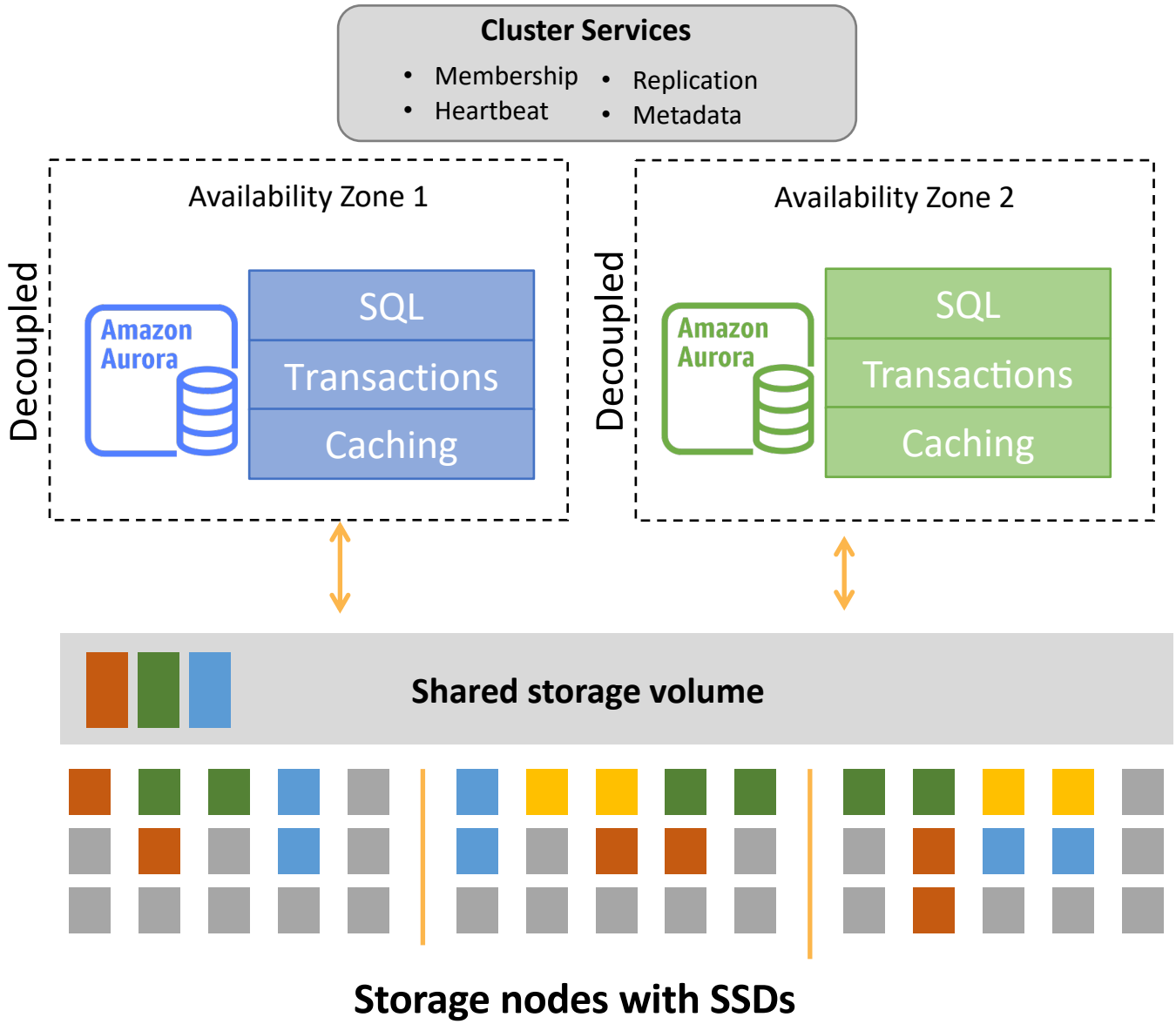
Heavyweight commit and membership change protocols

Can result in hot partitions = expensive repartitioning

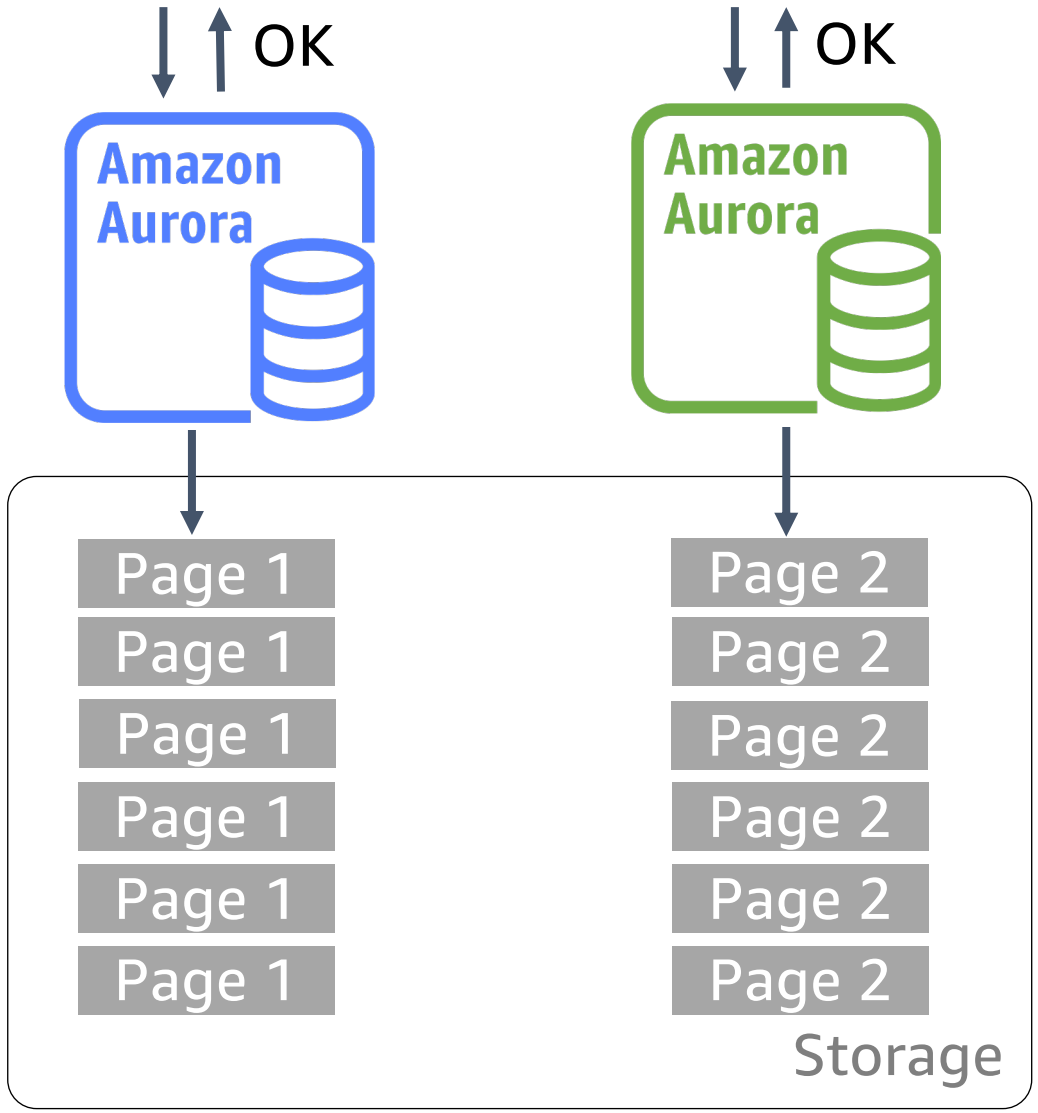
Cross partition operations expensive; better at small requests

Aurora Multi-Master

- Each instance can execute write transaction with no coordination with the others
- Instances share a distributed storage volume
- Nodes fail and recover independently
- Optimistic Page-Based Conflict Resolution
- No Pessimistic Locking
- No Global Commit Coordination
- Writer instances in two availability zones provide continuous availability
- GA August 2019



Non-Conflicting Writes

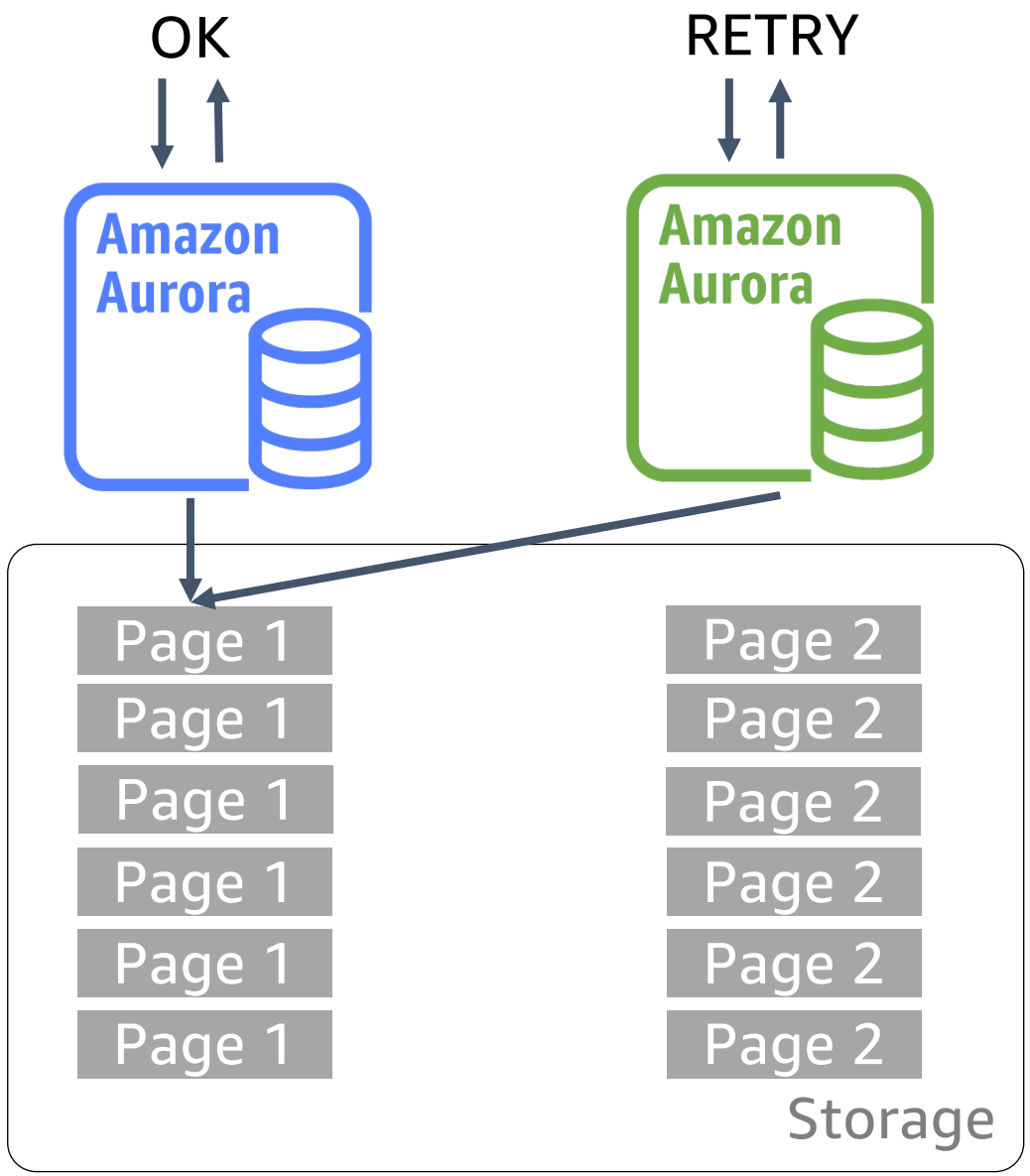


Non-conflicting writes originating on different masters on different tables

Time	Blue Master	Green Master
1	Begin Trx (BT1)	Begin Trx (OT1)
2	Update (table1)	Update (table2)
3	Commit (BT1)	Commit (OT1)

No Synchronization

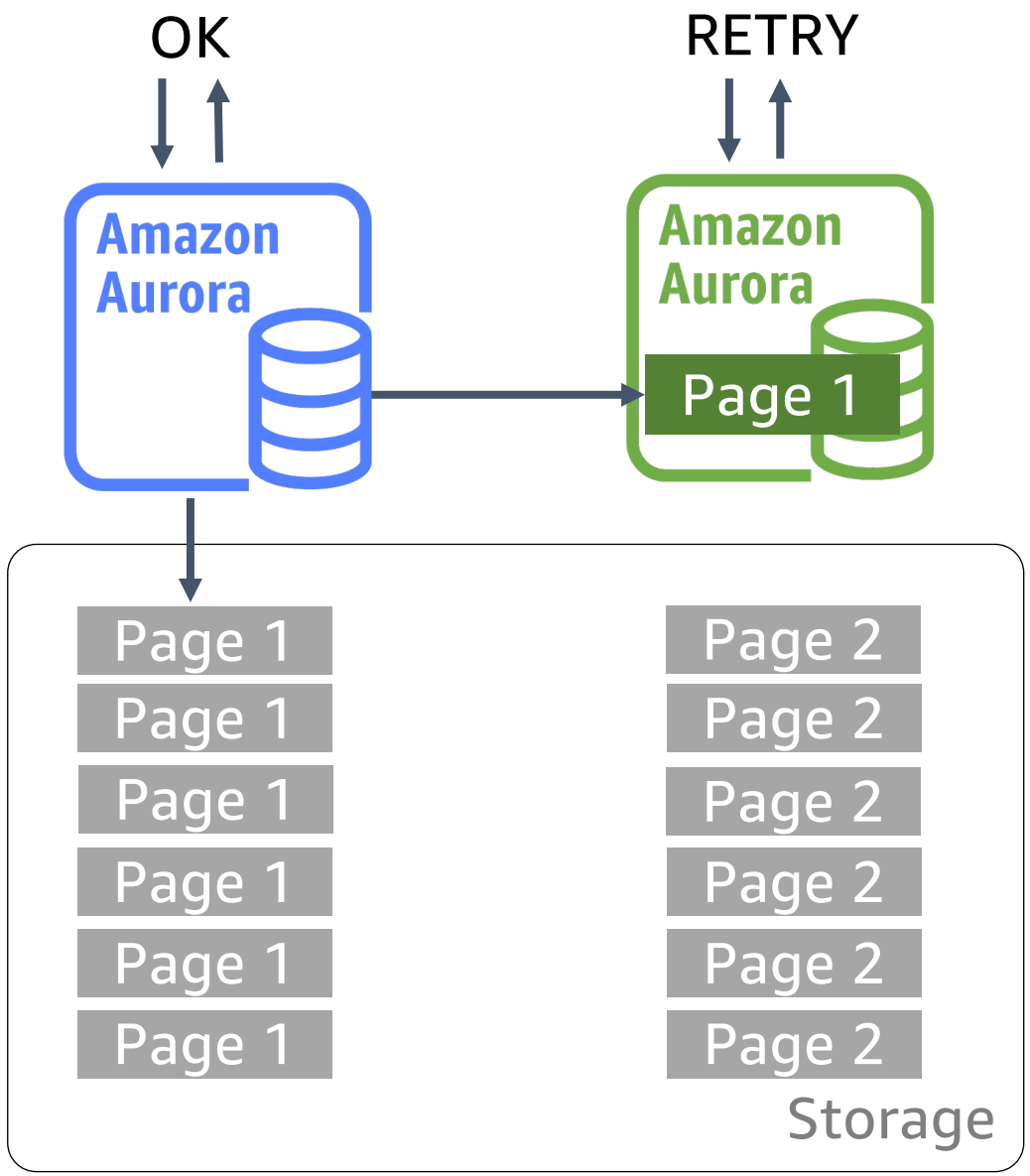
`Conflicting Write



Conflicting writes originating on different masters on the same table

Time	Blue Master	Green Master
1	Begin Trx (BT1)	Begin Trx (OT1)
2	Update (row1, table1)	Update (row1, table1)
3	Commit (BT1)	Rollback (OT1)

Optimistic Conflict Resolution



Conflicting writes originating on different masters on the same table

Time	Blue Master	Green Master
1	Begin Trx (BT1)	Begin Trx (OT1)
2	Update (row1, table1)	
3		Update (row1, table1) and rollback (OT1)
4	Commit (BT1)	

Logical Conflict Detection

Mechanics From the Head Node

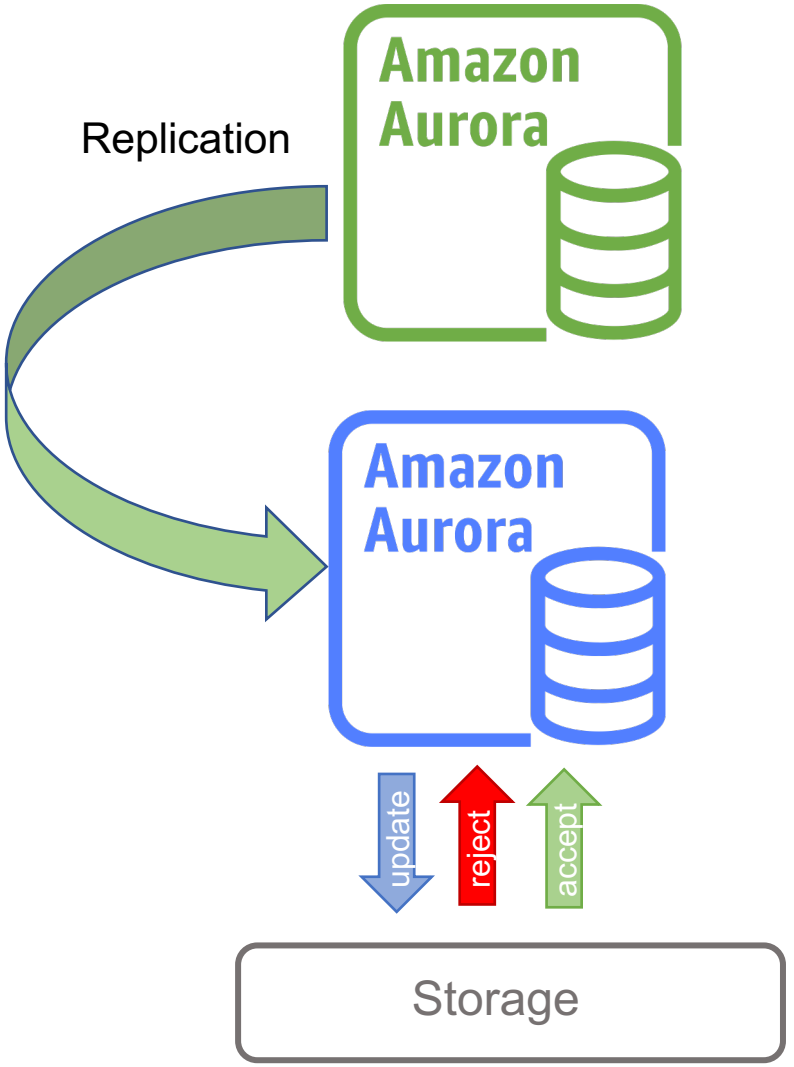
Partitioned LSN and transaction id space

Durability and *resolution point* at storage constantly increasing (creating new page versions)

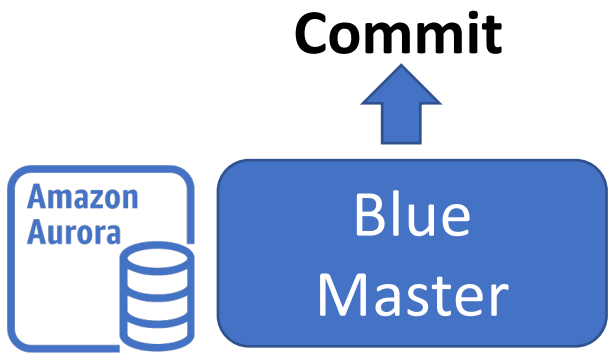
Incoming replication from other masters

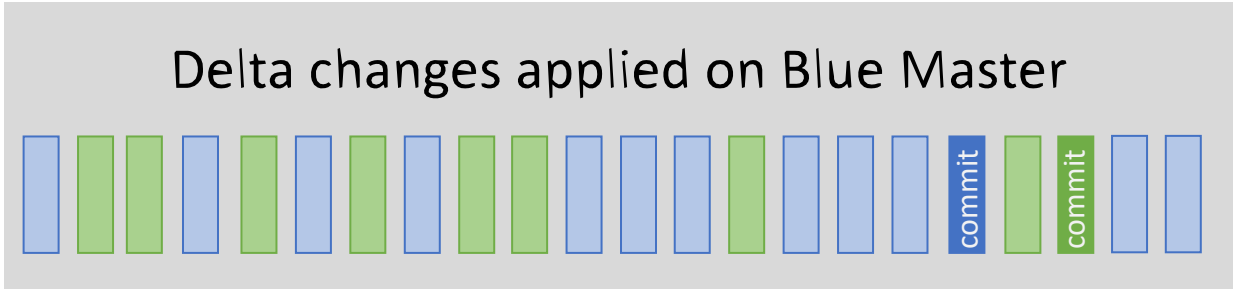
Database engine must handle rejected write to storage

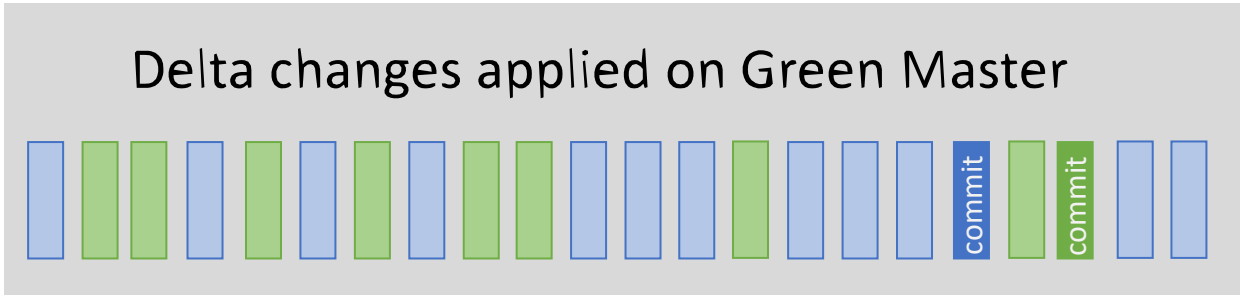
- Transaction rollback
- B-tree structure modifications
- Etc…



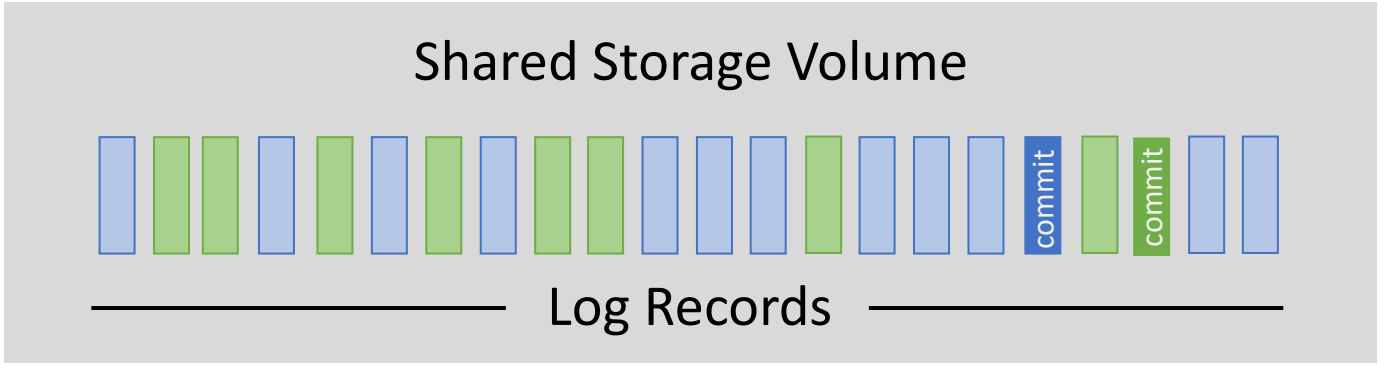
Multi-Master Commit







Transaction is committed after all writes are confirmed



Each node sees its own write

Writes from other nodes are subject to replication delay

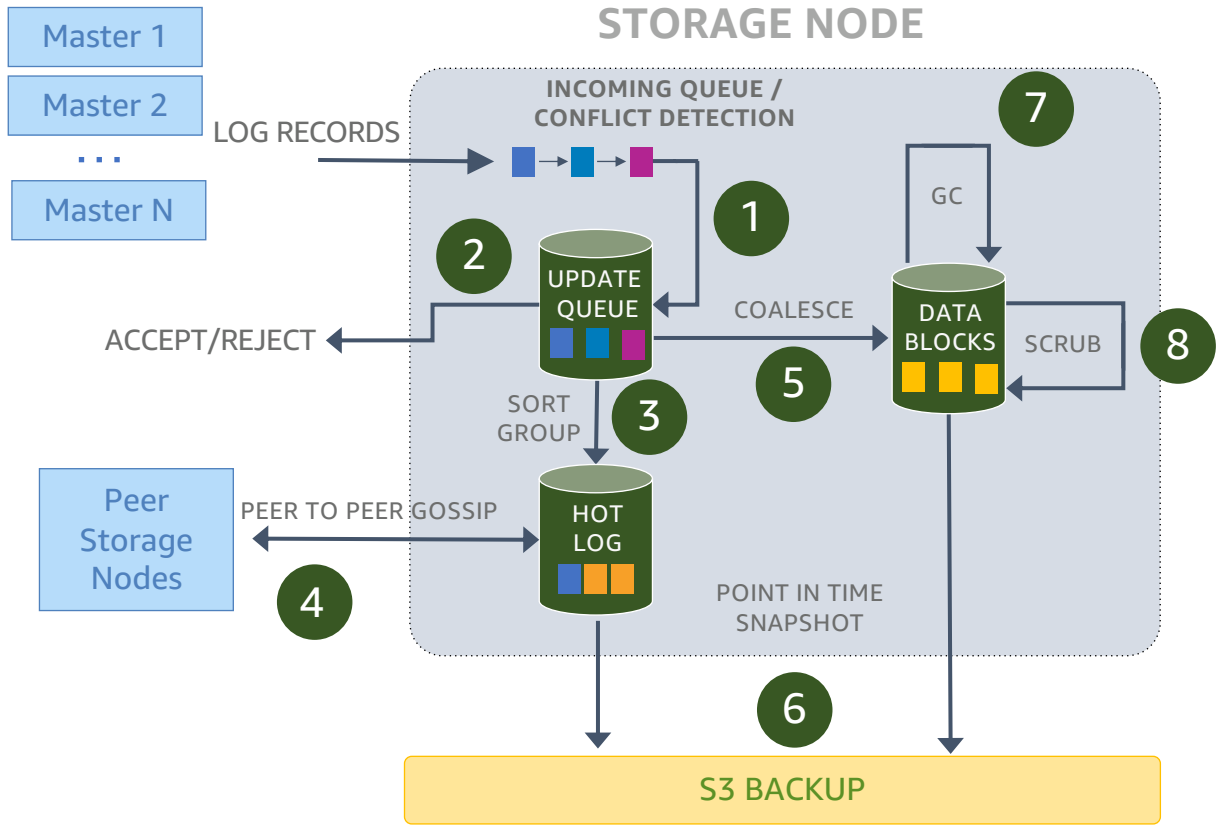
Mechanics From the Storage Node

Log records written by multiple masters

Quorum commit log records like before, fills in log chain, etc

Detects conflicting writes from other nodes

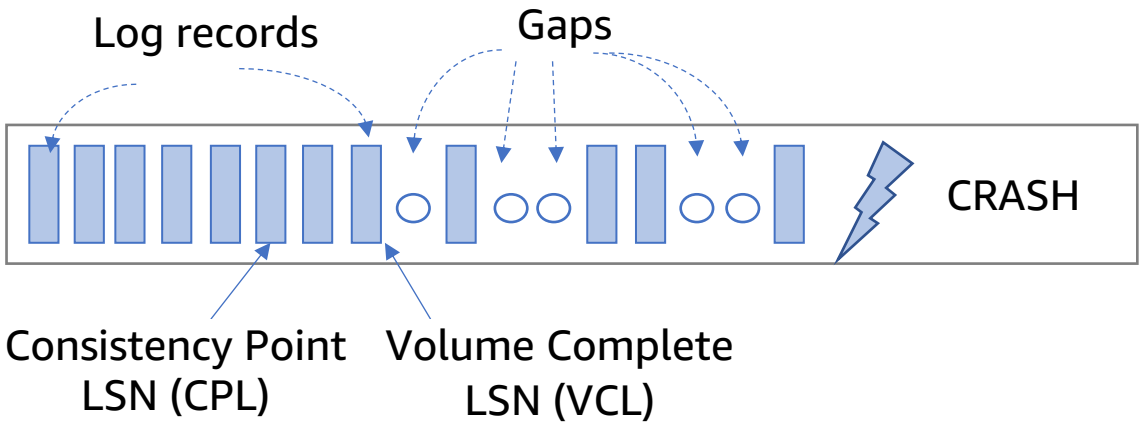
Returns rejection to log write on conflict



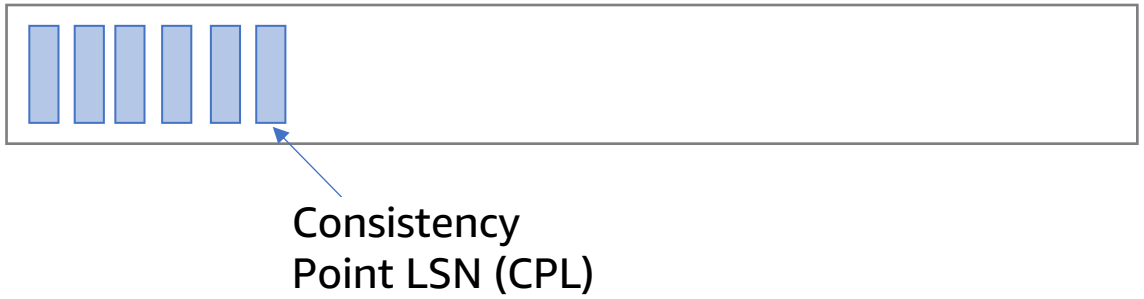
Recovery in Multi-Master

SINGLE MASTER

AT CRASH

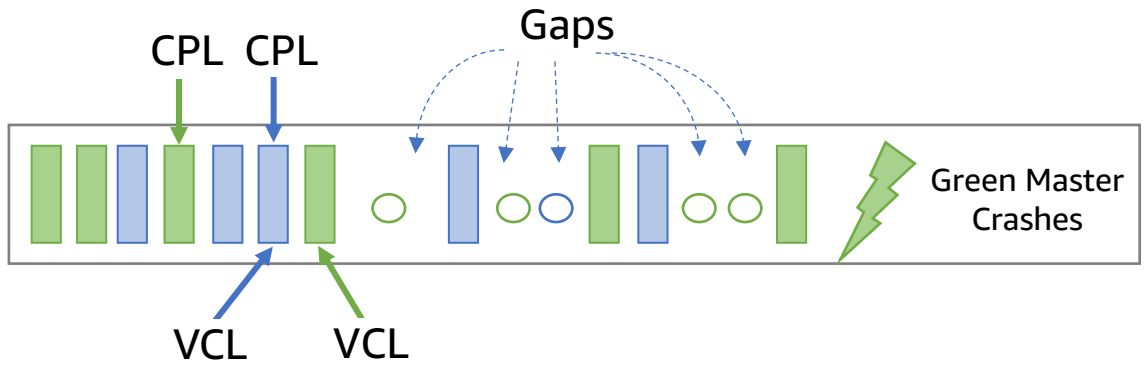


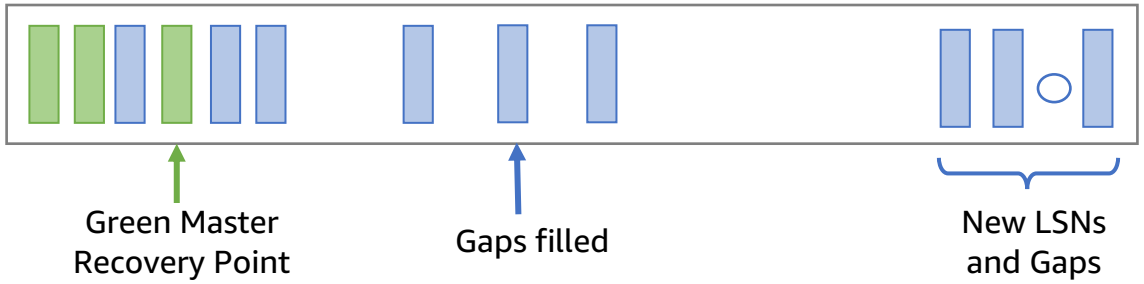
IMMEDIATELY AFTER CRASH RECOVERY



MULTI MASTER

AT CRASH

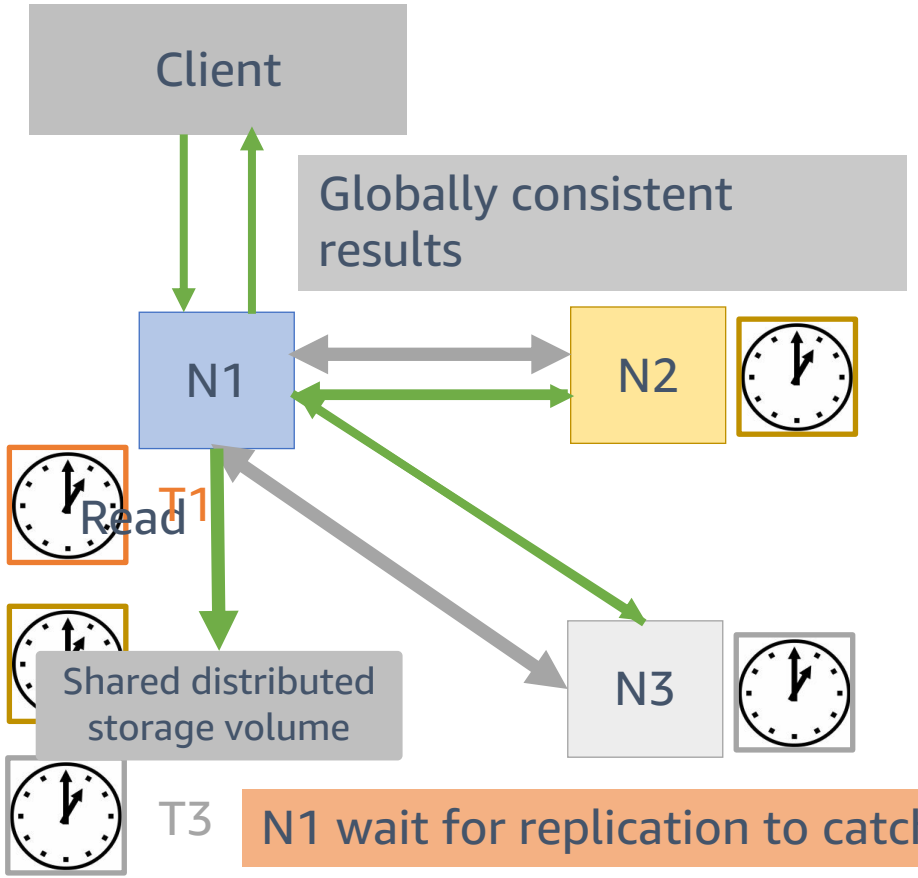




Instance Read-After-Write (INSTANCE_RAW): A transaction can observe all transactions previously committed on this instances, and transactions executed on other nodes, subject to replication lag.

Regional Read-After-Write (REGIONAL_RAW): A transaction can observe all transactions previously committed on all instances in the cluster.

Regional Read-After Write



No waits on the write path

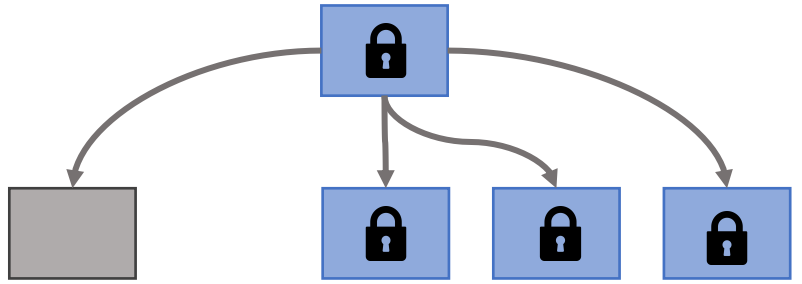
Adds latency ONLY to consistent reads

Configurable per session

Optimistic Execution: Mini-Transactions (3)

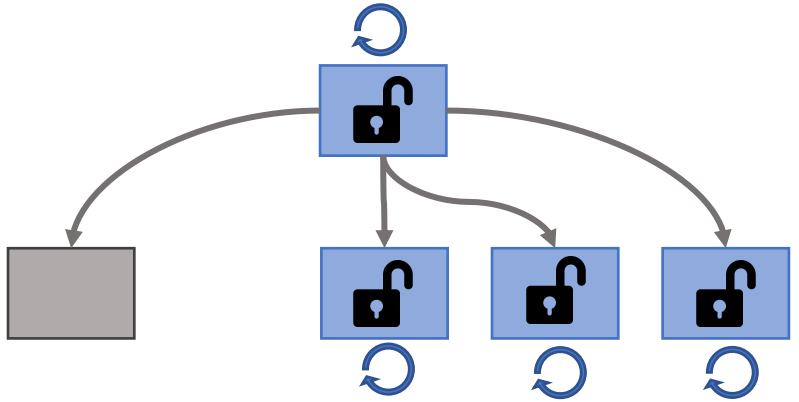
Resolution point constantly advancing

Can pessimistically wait for multi-page mtr to resolve (lack of concurrency/performance)



Aurora optimistically executes multi-page mtr (greater in-memory concurrency)

Rolls back mtr (and all dependent operations) retroactively on conflict



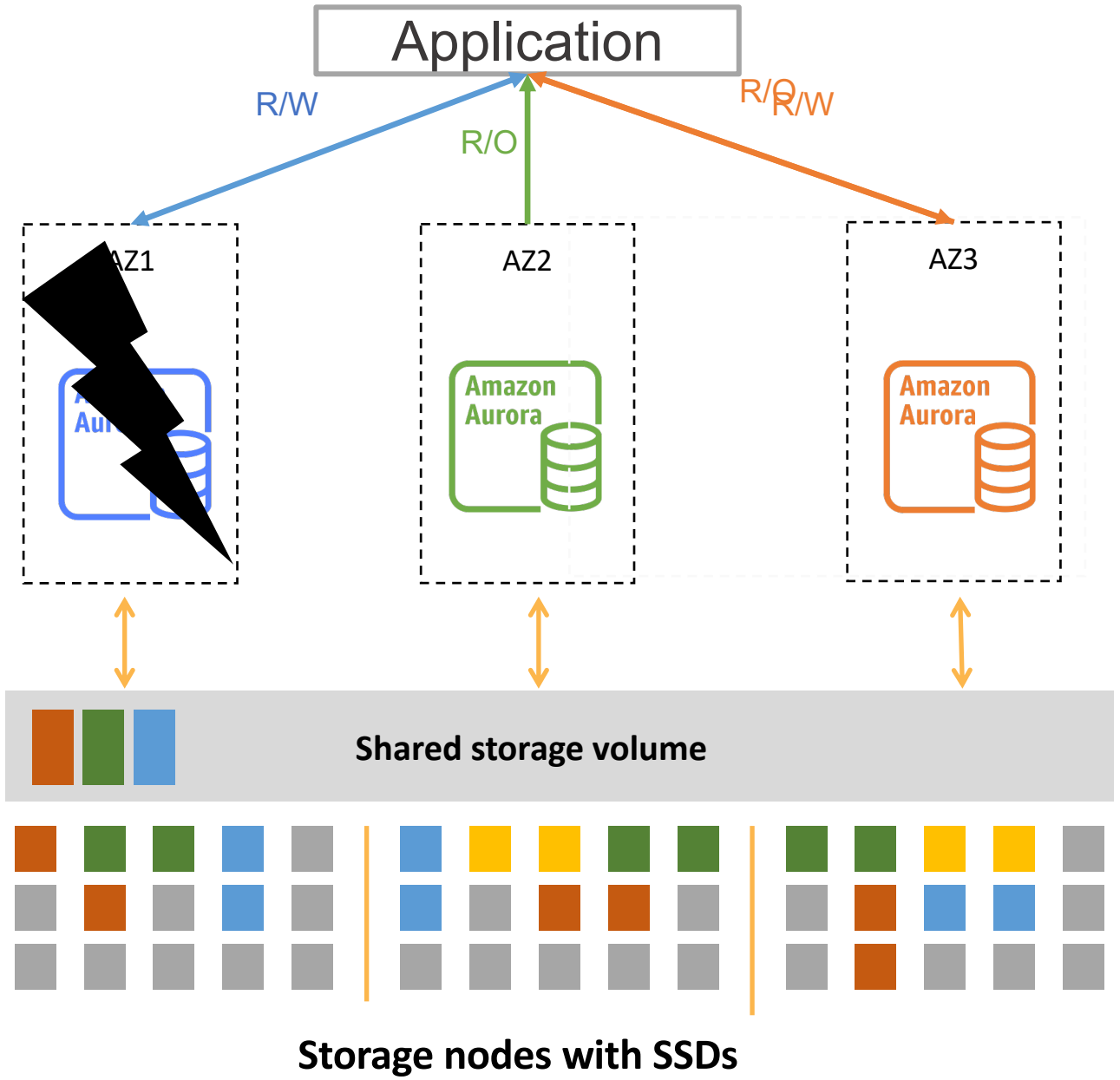
Adaptively switches to pessimistic resolution if high percentage of conflict detected

Aurora Multi-Master Use Cases

Continuous Availability

A lot of work can be done on single Aurora writer

Writable replicas provide instant failover



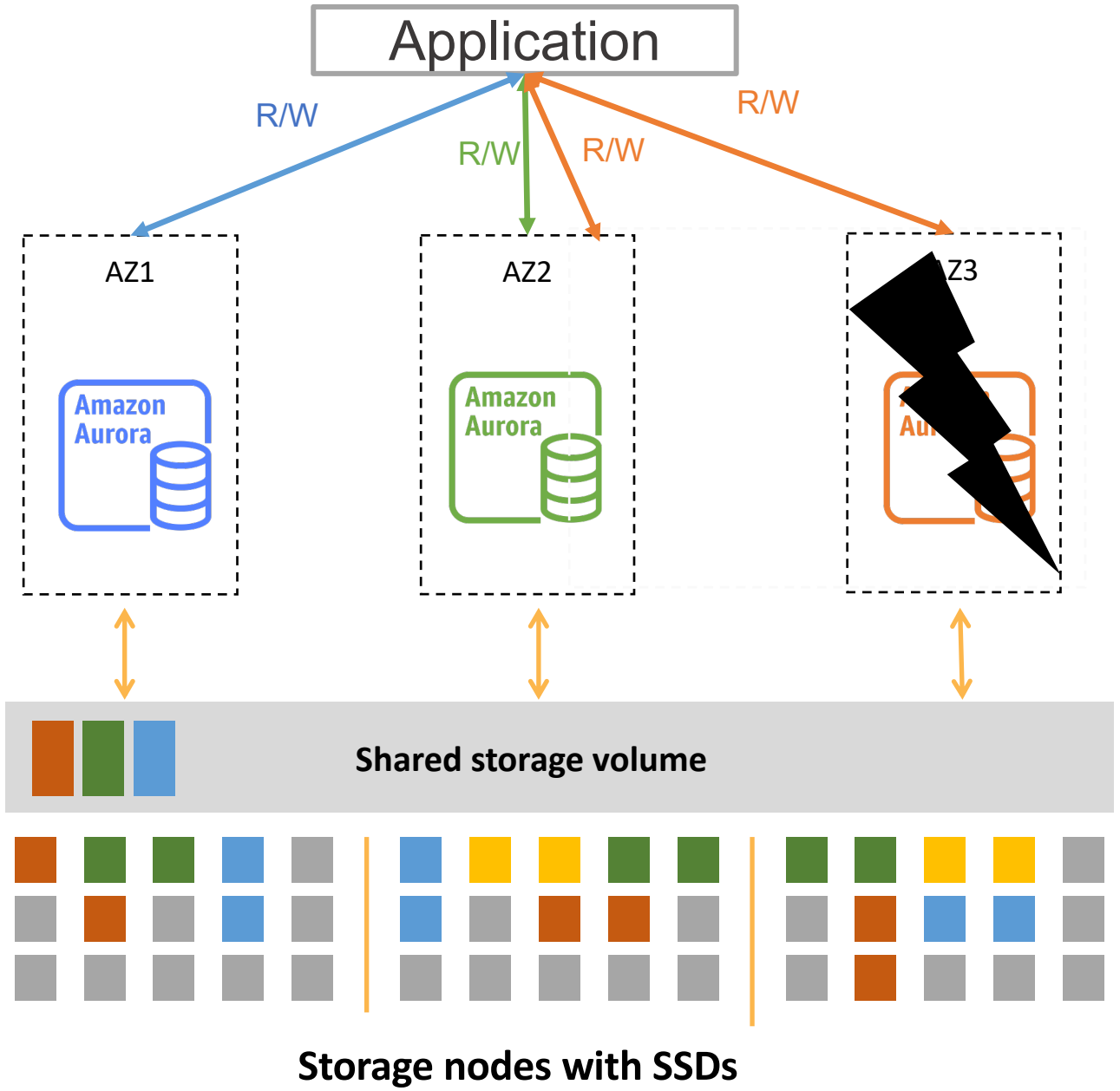
Multi-Writer Configuration

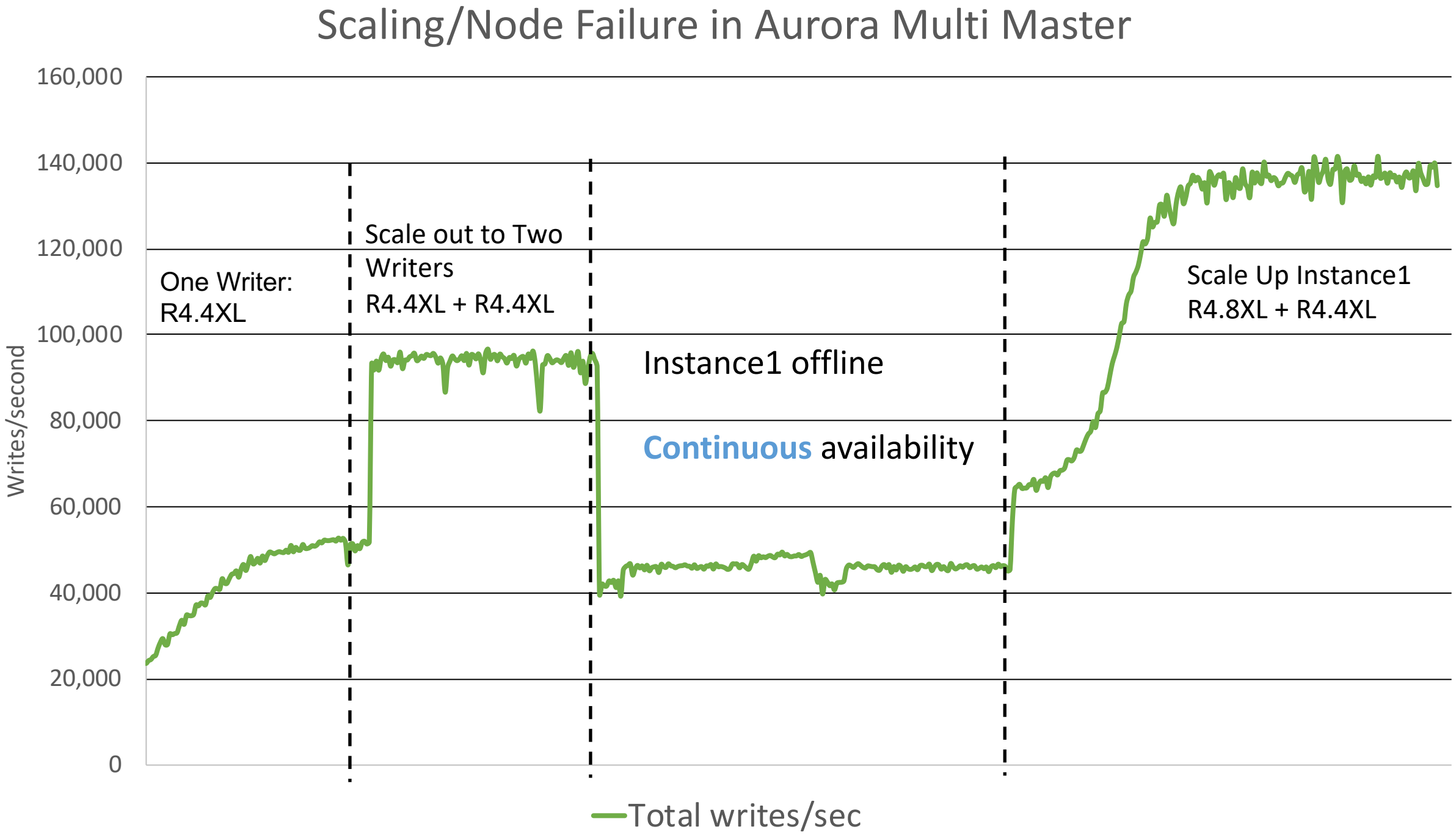
Structure the workload to limit conflicts between database instances.

Prefer partitioning writes per table (or table partition) from a single database instance.

Aurora MM allows customers to “soft partition,” or re-partition on the fly

Continuous availability through failures and planned maintenance




Scaling/Node Failure in Aurora Multi Master
160,000
140,000
120,000
100,000
80,000
60,000
40,000
20,000
0
Writes/second
One Writer:
R4.4XL
Scale out to Two
Writers
R4.4XL + R4.4XL
Instance1 offline
Continuous availability
Scale Up Instance1
R4.8XL + R4.4XL
Total writes/sec

Questions