

AWS  
re:Invent

**DAT309-R**

# Amazon Aurora storage demystified: How it all works

## **Murali Brahmadesam**

Director of Engineering,  
Amazon Aurora  
Amazon Web Services

## **Tobias Ternstrom**

Director of Product Mgmt,  
Amazon Aurora  
Amazon Web Services

# Agenda

- What is Amazon Aurora?
- Quick recap: Database internals & motivation for building Aurora
- Cloud-native database architecture
- Durability at scale
- Performance results

## Features & demos

- Global databases
- Fast database cloning
- Database backtrack

# What is Amazon Aurora ?

Enterprise class cloud native database

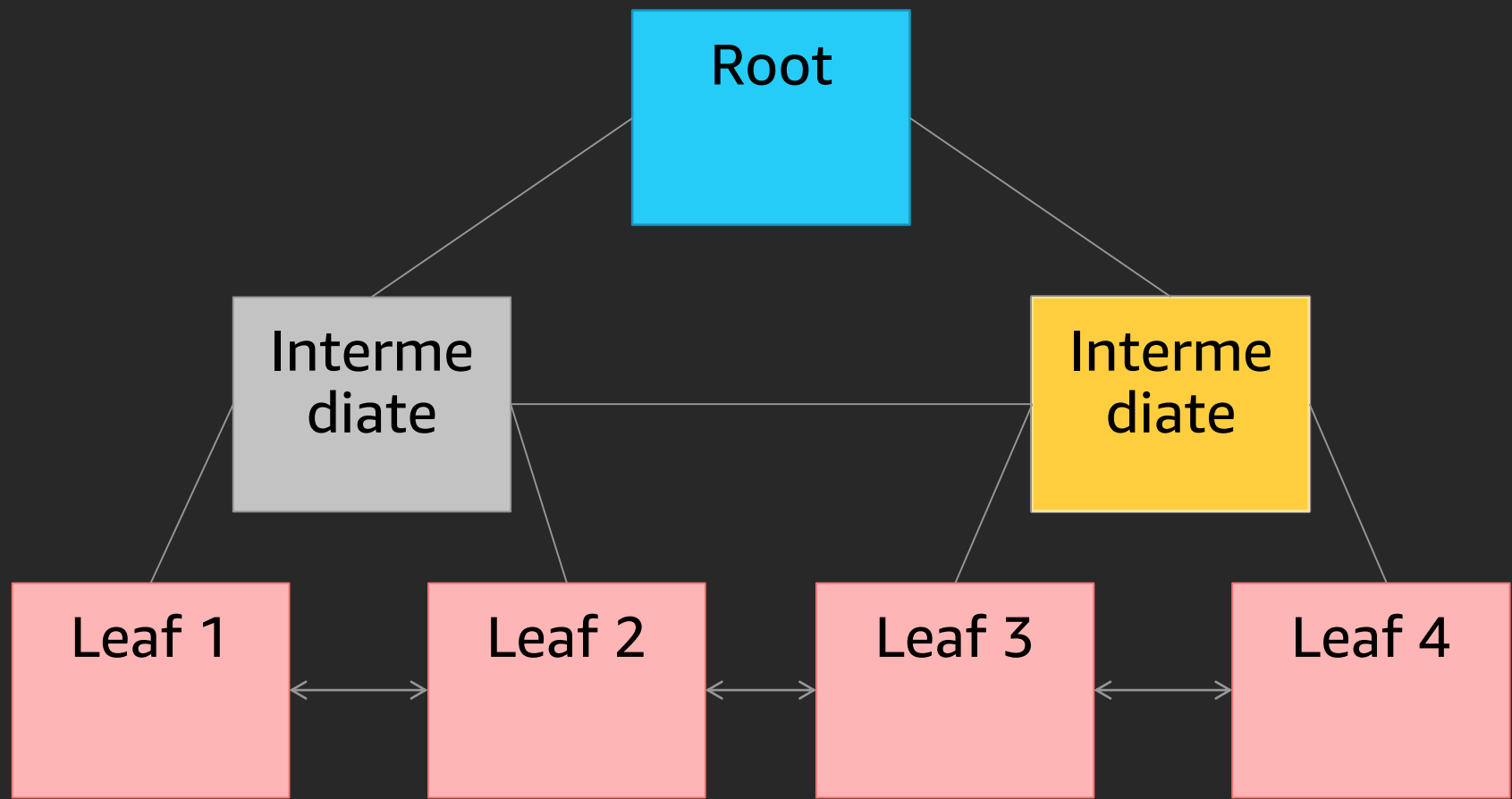


- ✓ **Speed** and **availability** of high-end commercial databases
- ✓ **Simplicity** and **cost-effectiveness** of open-source databases
- ✓ Drop-in **compatibility** with MySQL and PostgreSQL
- ✓ Simple **pay-as-you-go** pricing

Delivered as a **managed** service

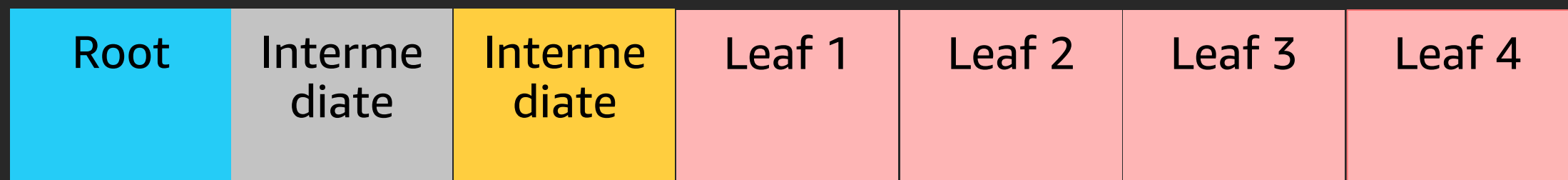
# Quick recap: Database internals

# Quick recap: Database B+ Tree



Data is organized in memory as fixed sized "pages", e.g. 16KB (aka "buffer-pool")

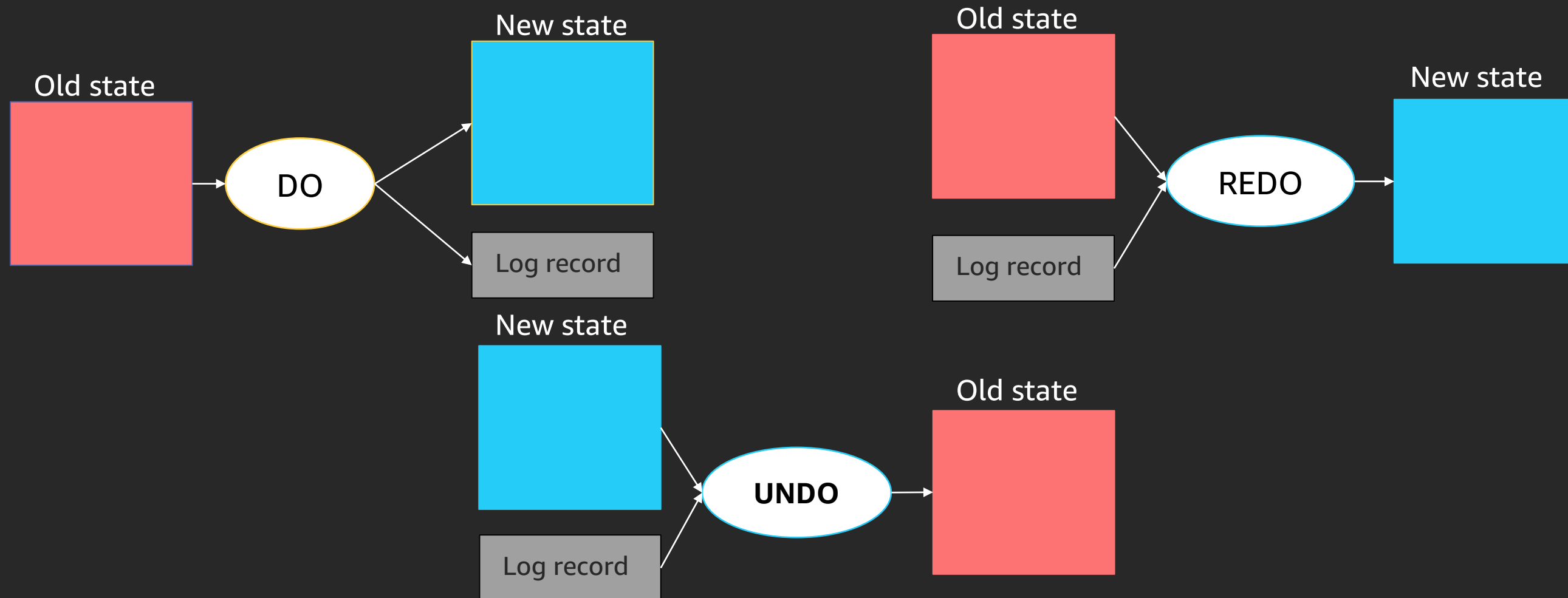
Pages are serialized into durable storage (aka "checkpoint") periodically



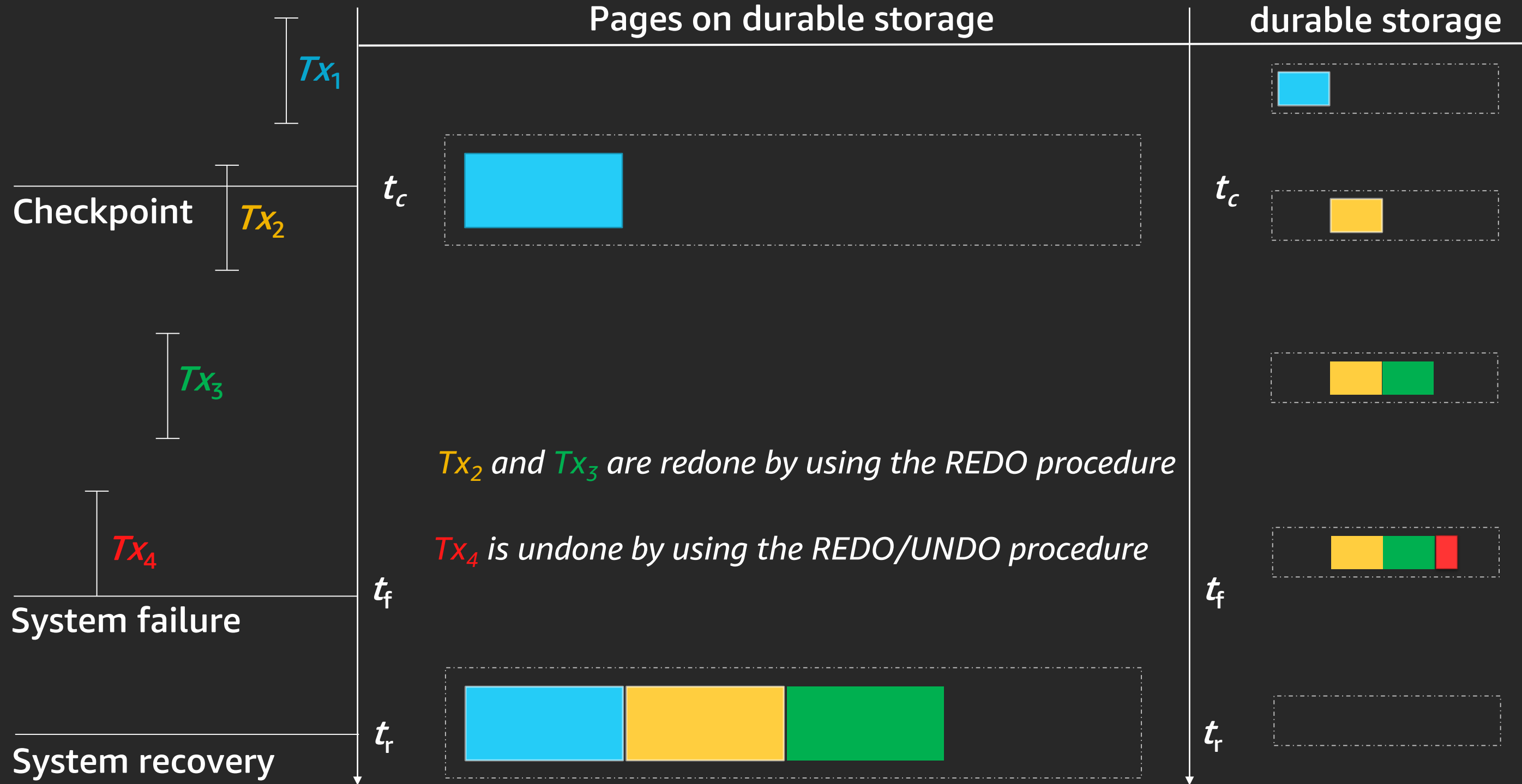
# Quick recap: DO-REDO-UNDO protocol

Data is modified "in-place" in the buffer-pool using a DO/REDO/UNDO operation

Log records with before and after images are stored in a write-ahead log (WAL)



# Quick recap: Crash Recovery



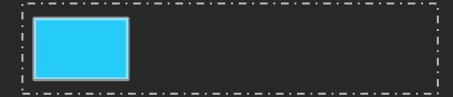


# Quick recap: I/Os required for persistence

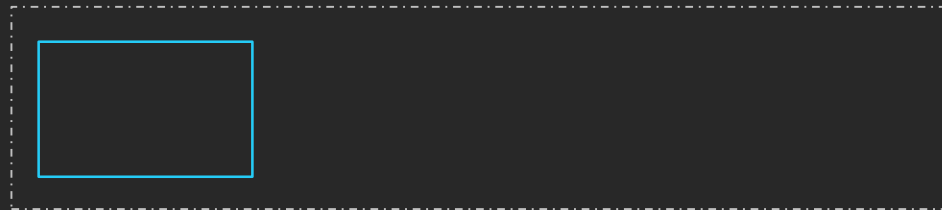
Pages on durable storage

Log records on durable storage

Log record write:  
typically few bytes



Torn page protection write:  
page sized, e.g. 16KB



Checkpoint write:  
page sized, e.g. 16KB



User data change size  $\ll$  I/O size (32KB+)

**Databases are all about I/O**

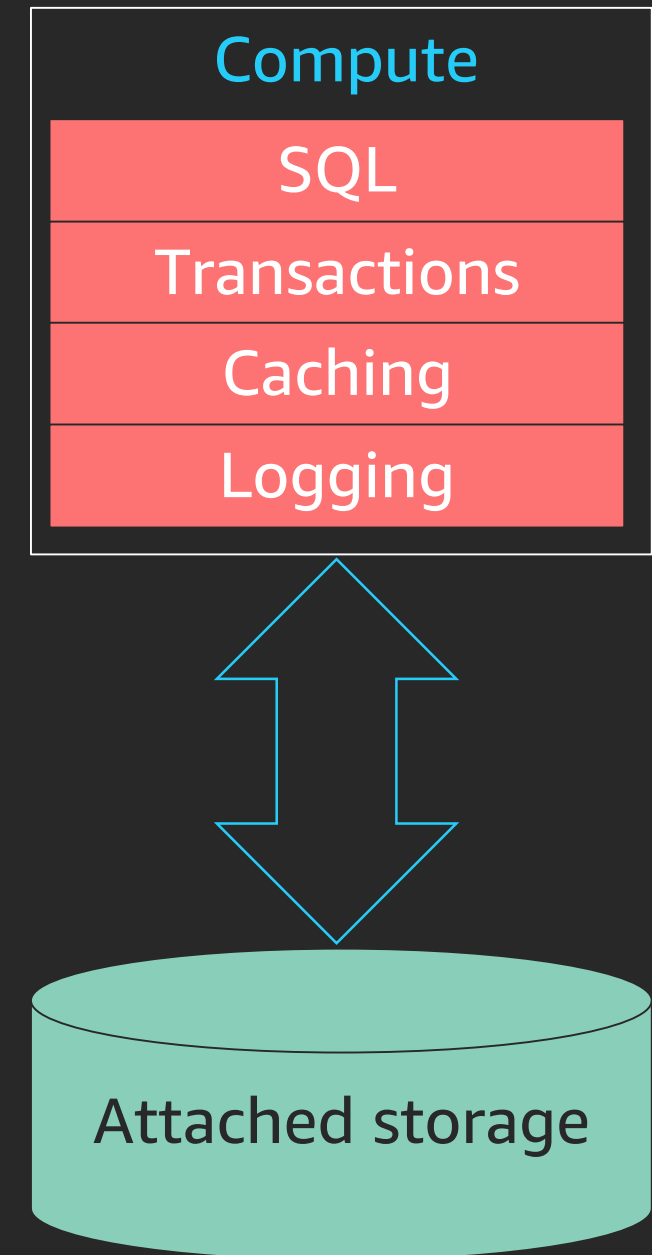
# Cloud native database architecture

# Traditional database architecture

Databases are all about I/O

Design principles for > 40 years

- Increase I/O bandwidth
- Decrease number of I/Os!



# Aurora approach: Log is the database

Log stream from beginning of the database



Any version of a database page can be constructed using the log stream

Blue-page at  $t_5$  can be created using log records from  $t_1$  and  $t_5$



# Aurora approach: Offload checkpointing to the storage fleet

## Problem 1:

Relying only on log stream for page reads is not practical (too slow)

## Solution:

Use periodic checkpoints

## Problem 2:

Database instance is burdened with checkpointing task

## Solution:

Use a distributed storage fleet for continuous checkpointing

# Aurora approach: compute & storage separation

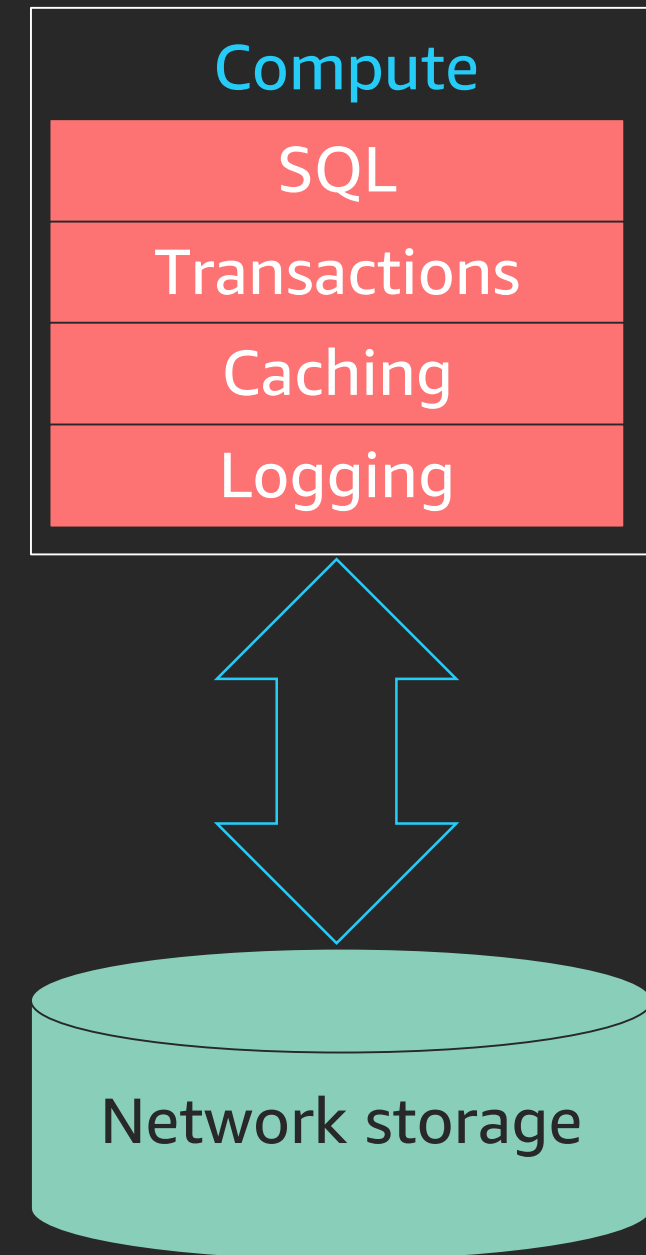
Compute & storage have different lifetimes

## Compute instances

- fail and are replaced
- are shut down to save cost
- are scaled up/down/out on the basis of load needs

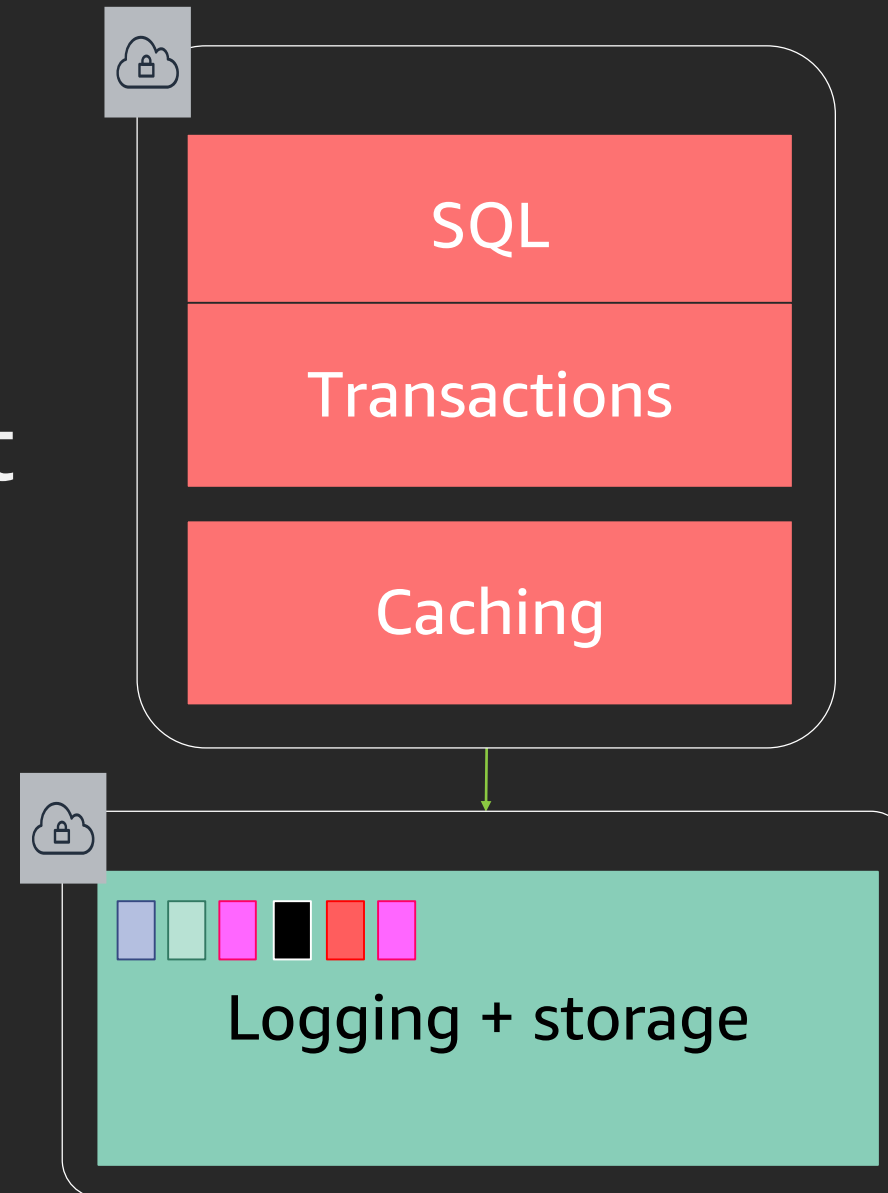
Storage, on the other hand, has to be long-lived

Decouple compute and storage for scalability, availability, durability



# Aurora uses service-oriented architecture

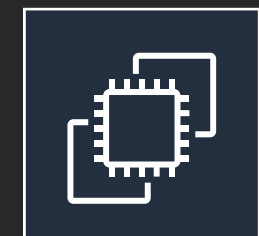
We built a log-structured distributed storage system that is multi-tenant, multi-attach, and purpose-built for databases



Amazon  
DynamoDB



Amazon  
Route 53

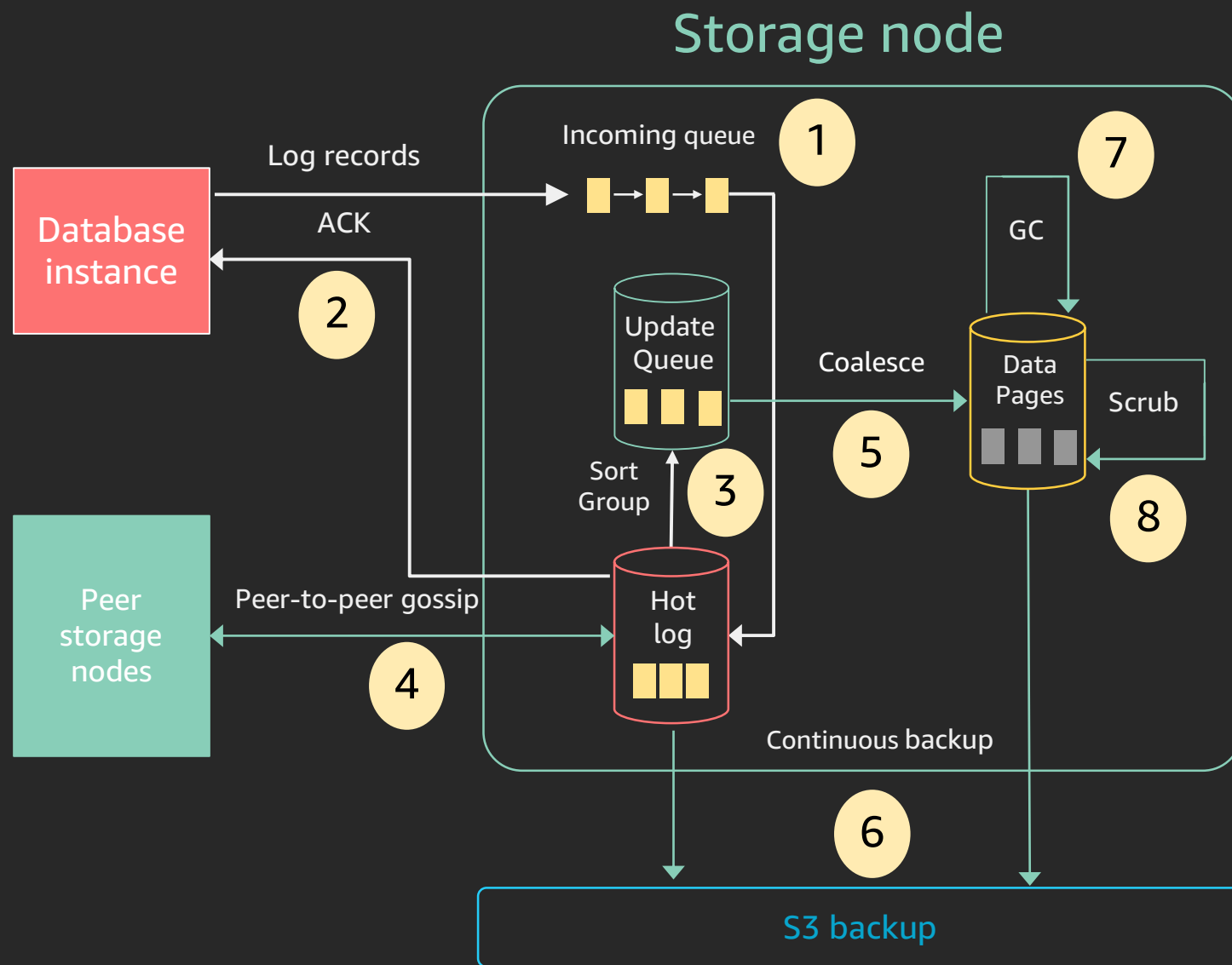


Amazon EC2



Amazon S3

# I/O flow in Amazon Aurora storage node



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

## Note:

- All steps are asynchronous
- Only steps 1 and 2 are in the foreground latency path



# Durability at scale

# Uncorrelated and independent failures

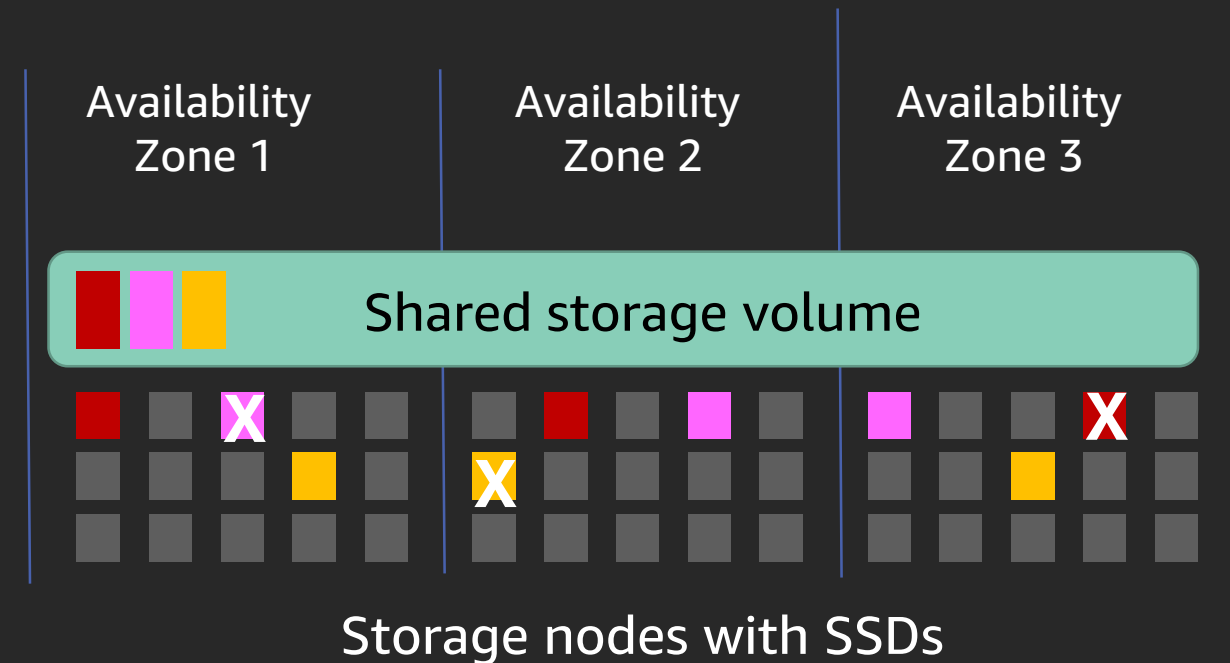
At scale there are continuous independent failures due to failing nodes, disks, and switches.

## The solution is replication

One common straw man:

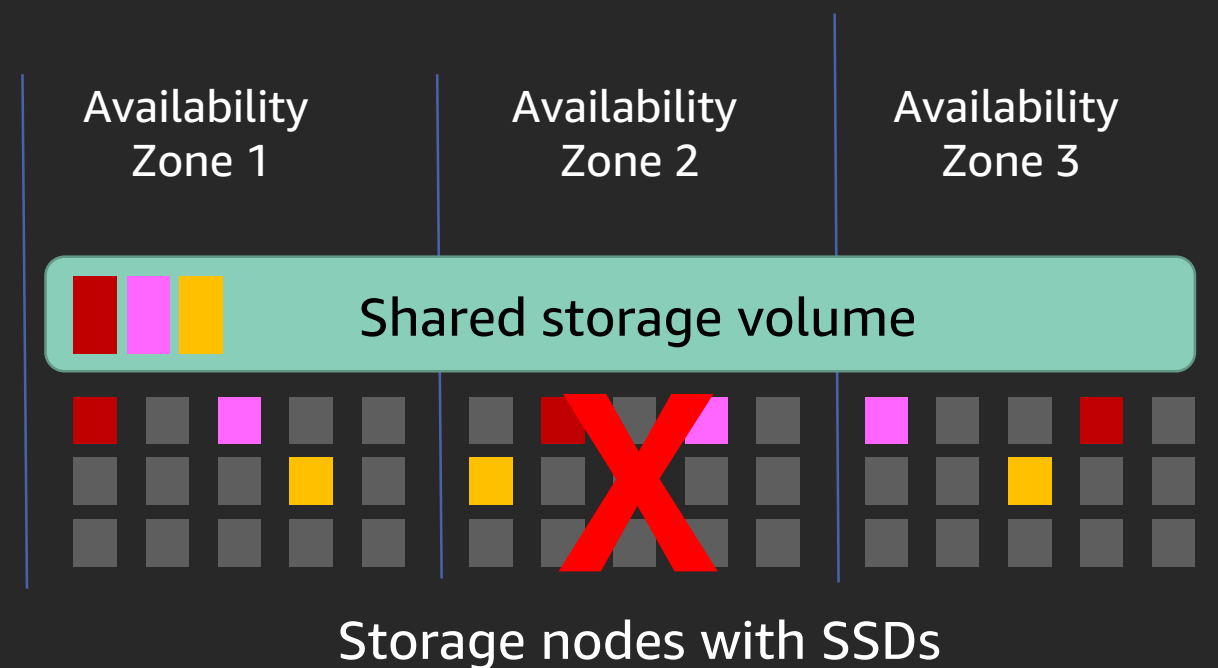
Replicate 3-ways with 1 copy per AZ

Use write and read quorums of 2/3



# What about AZ failure?

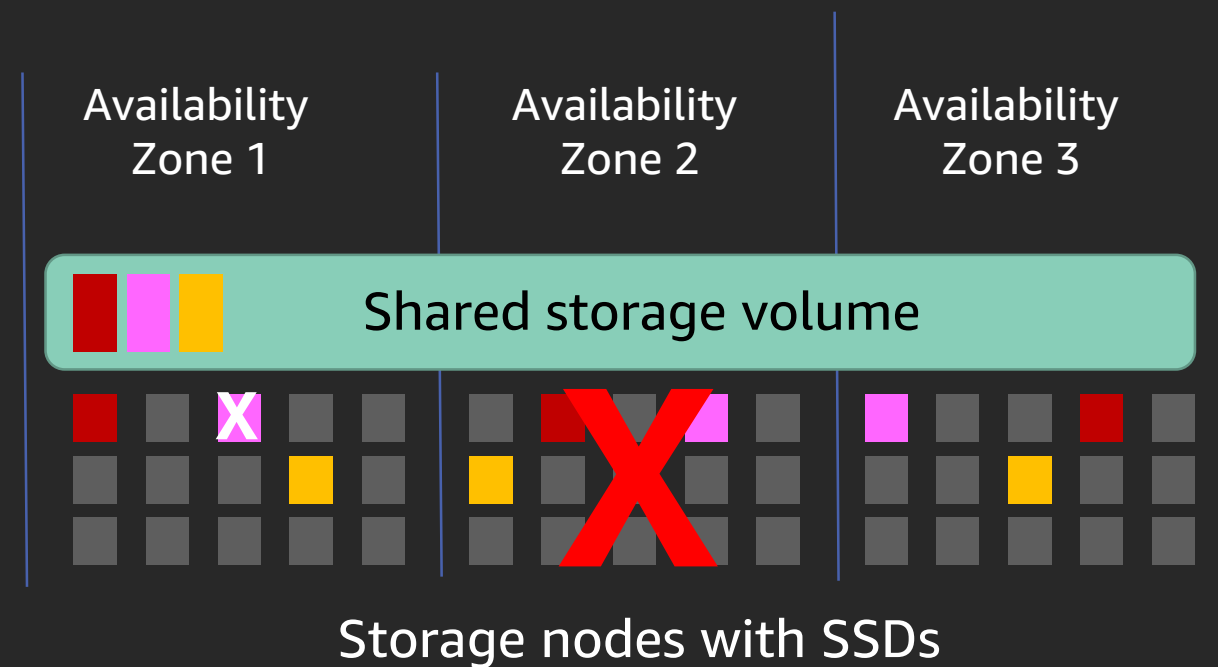
- ⇒ Still have 2/3 copies
- ⇒ Can establish quorum
- ⇒ No data loss



# What about AZ + 1 failures?

Losing 1 node in an AZ while another AZ is down

- ⇒ Lose 2/3 copies
- ⇒ Lose quorum
- ⇒ Lose data



# Aurora tolerates AZ + 1 failures

Replicate 6-ways with 2 copies per AZ

Write quorum of 4/6

What if an AZ fails?

⇒ Still have 4/6 copies

⇒ Maintain write availability

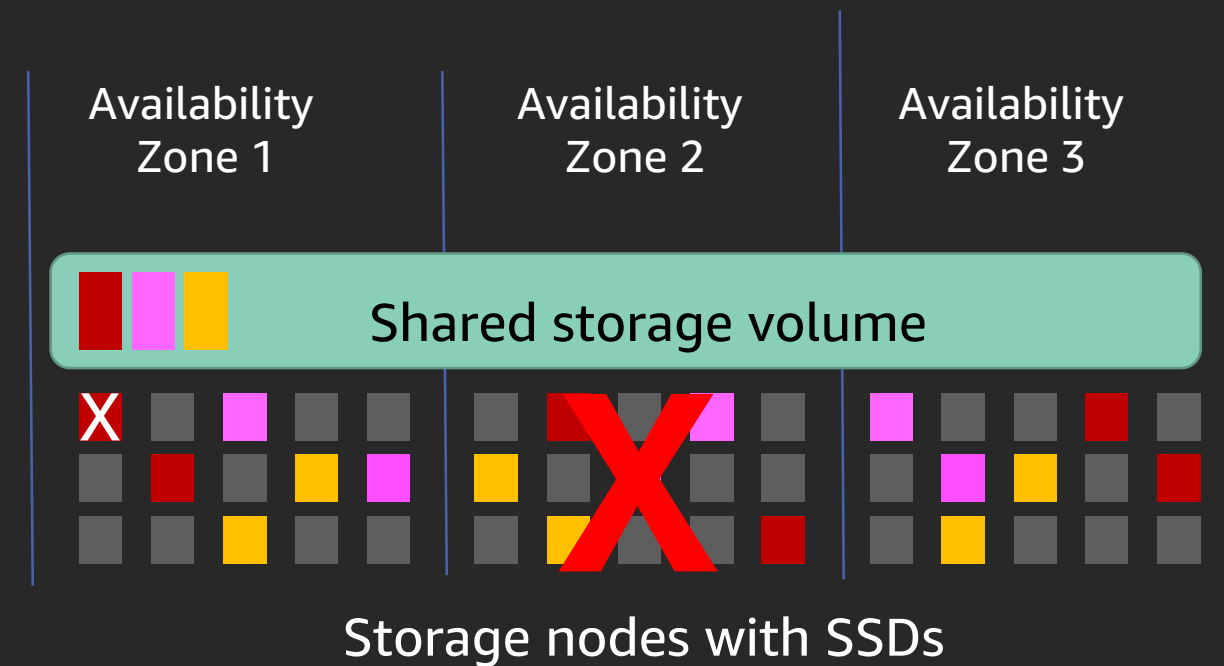
What if there is an AZ + 1 failure ?

⇒ Still have 3 copies

⇒ No data loss

⇒ Rebuild failed copy by copying from 3 copies

⇒ Recover write availability



# Aurora uses segmented storage

- Partition volume into  $n$  fixed-size segments
  - Replicate each segment 6 ways into a protection group (PG)
- Trade-off between likelihood of faults and time to repair
  - If segments are too small, failures are more likely
  - If segments are too big, repairs take too long
- Choose the biggest size that lets us repair “fast enough”
  - We currently picked a segment size of 10 GB, as we can repair a 10-GB segment in less than a minute

# Fast and reversible membership changes

## Use quorum sets, and epochs to

- Enable quicker transitions with epoch advances
- Create richer temporary quorums during changes
- Reverse changes by more quorum transitions



Epoch 1: All nodes are healthy



Epoch 2: Node F is in a suspect state; second quorum group is formed with node G; both quorums are active



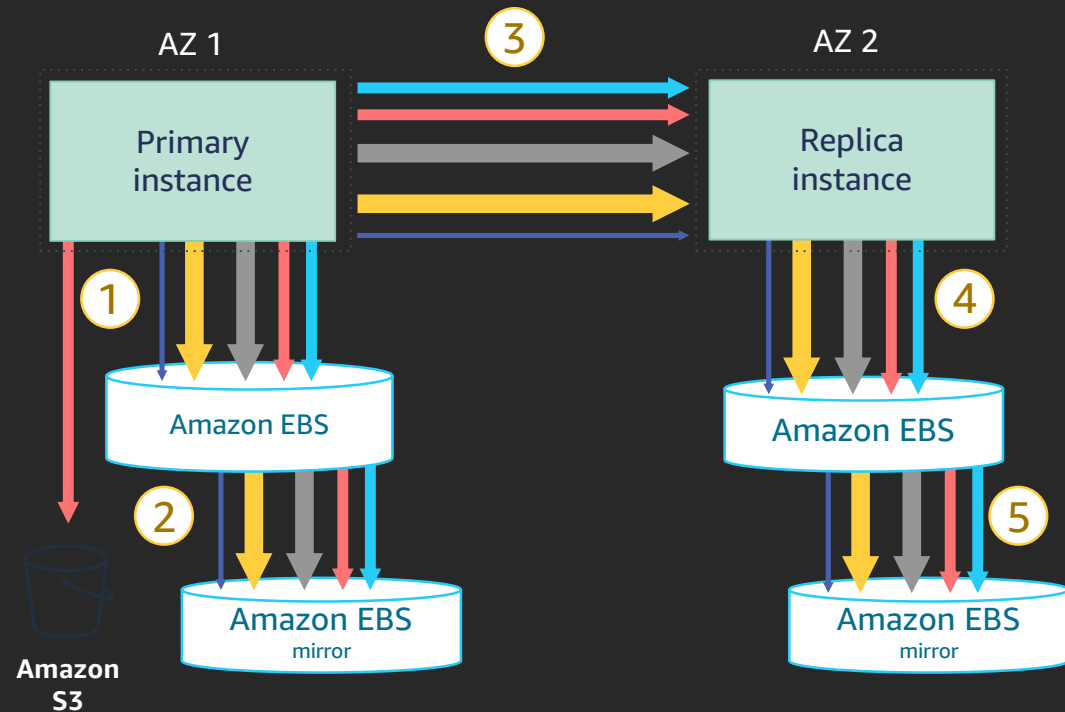
Epoch 3: Node F is confirmed unhealthy; new quorum group with node G is active

# Performance results



# Aurora I/O profile

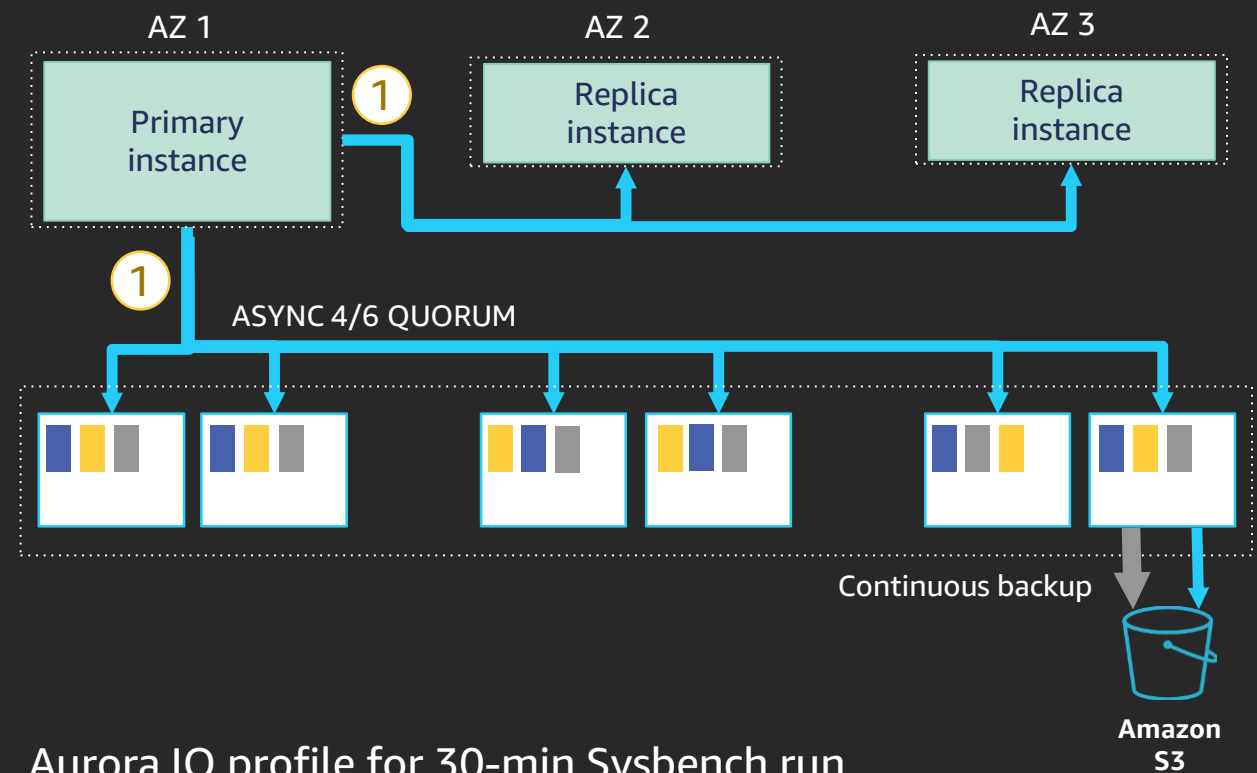
## MySQL with replica



MySQL I/O profile for 30-min Sysbench run

- 780K transactions
- Average 7.4 I/Os per transaction

## Aurora



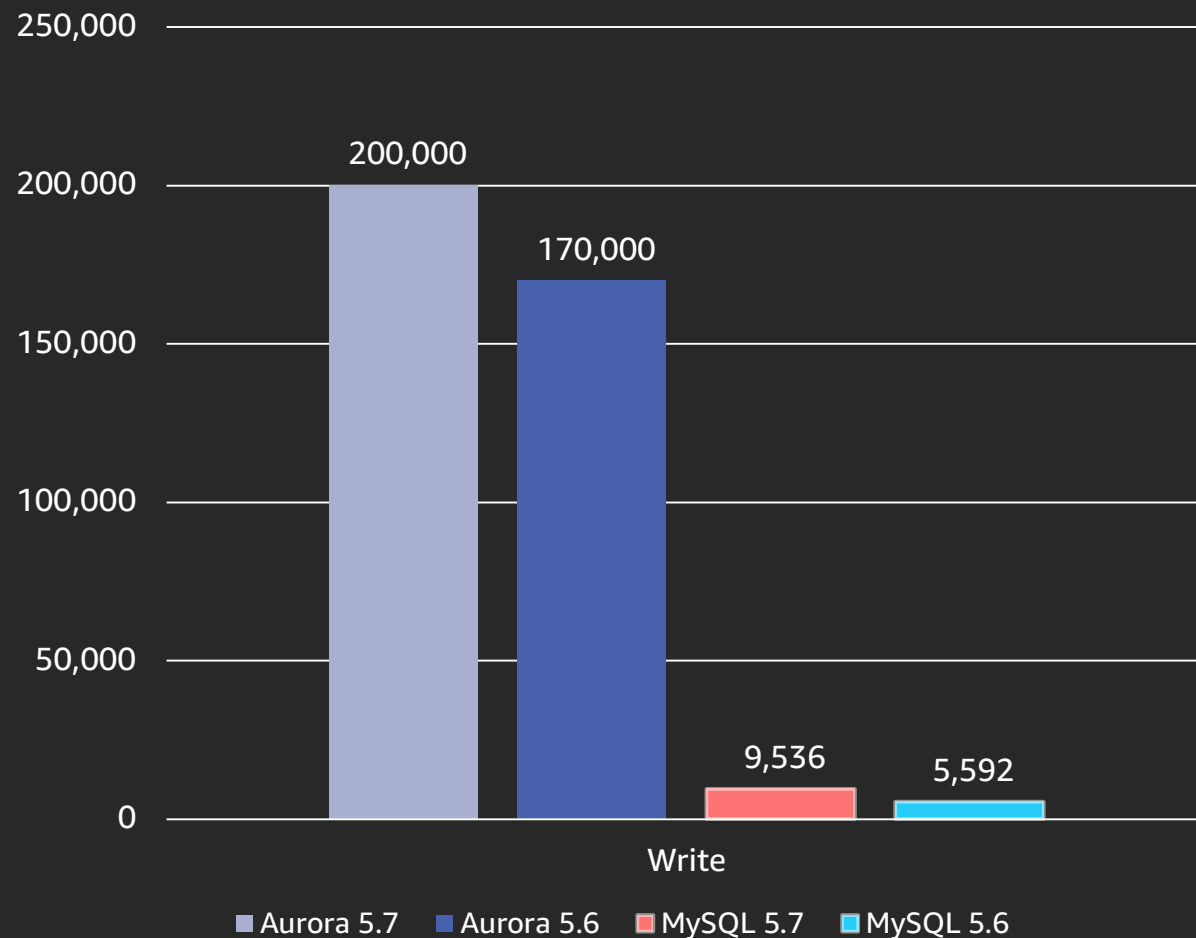
Aurora IO profile for 30-min Sysbench run

- 27M transactions: 35× more
- 0.95 I/Os per transaction (6× amplification): 7.7× less

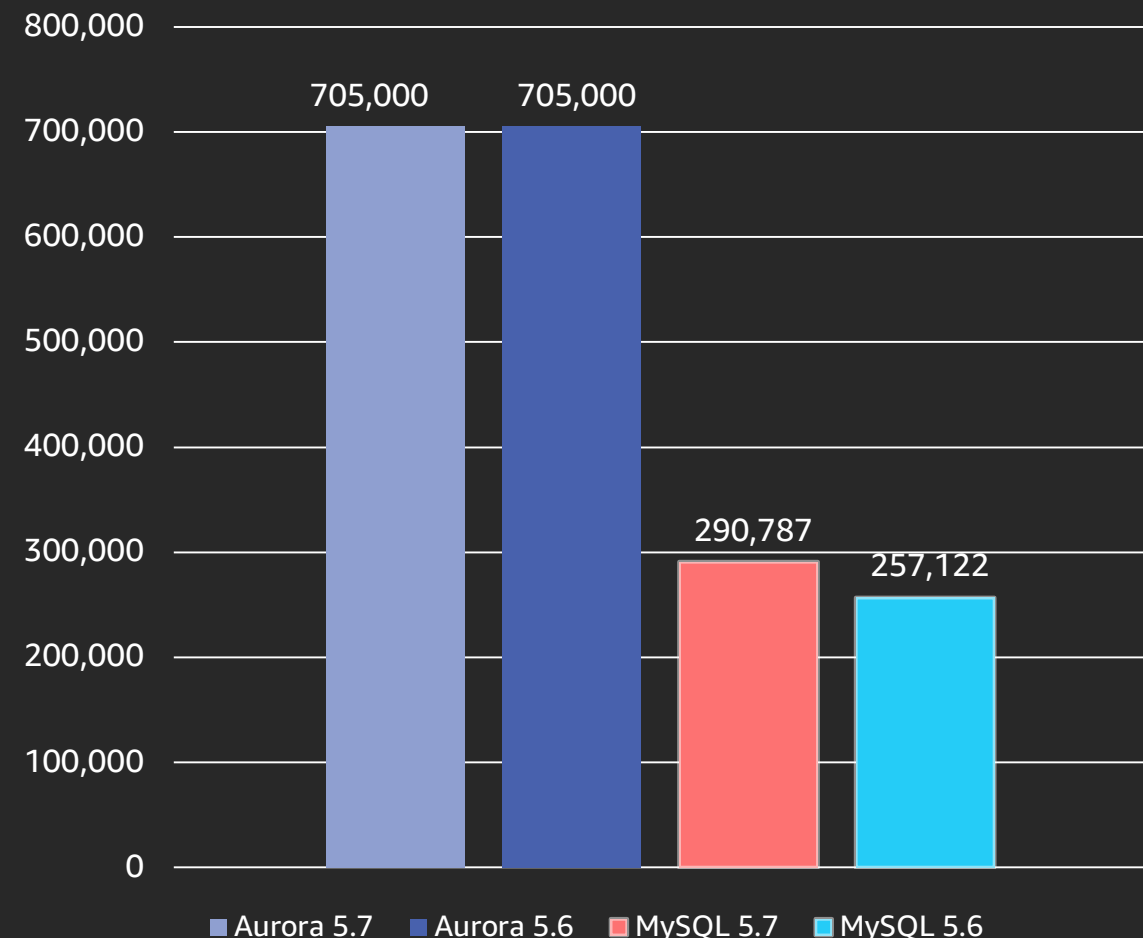


# Write and read throughput

## Aurora MySQL is 5× faster than MySQL



Write throughput



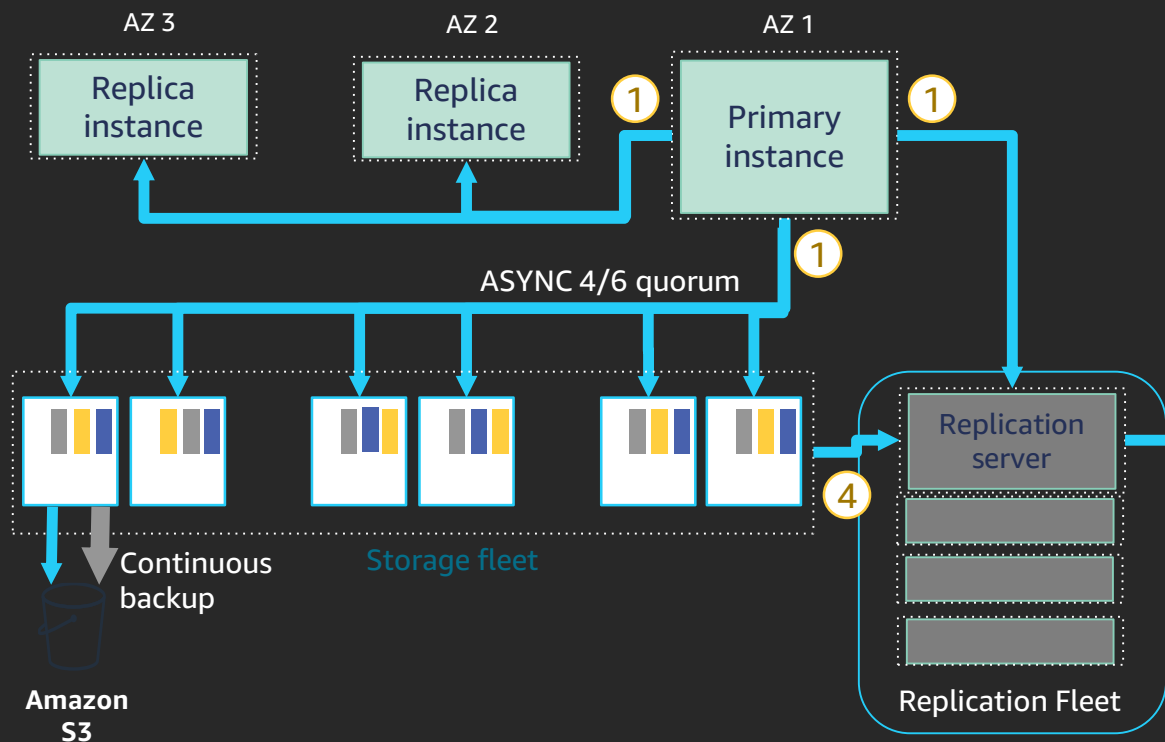
Read throughput

Using Sysbench with 250 tables and 200,000 rows per table on R4.16XL

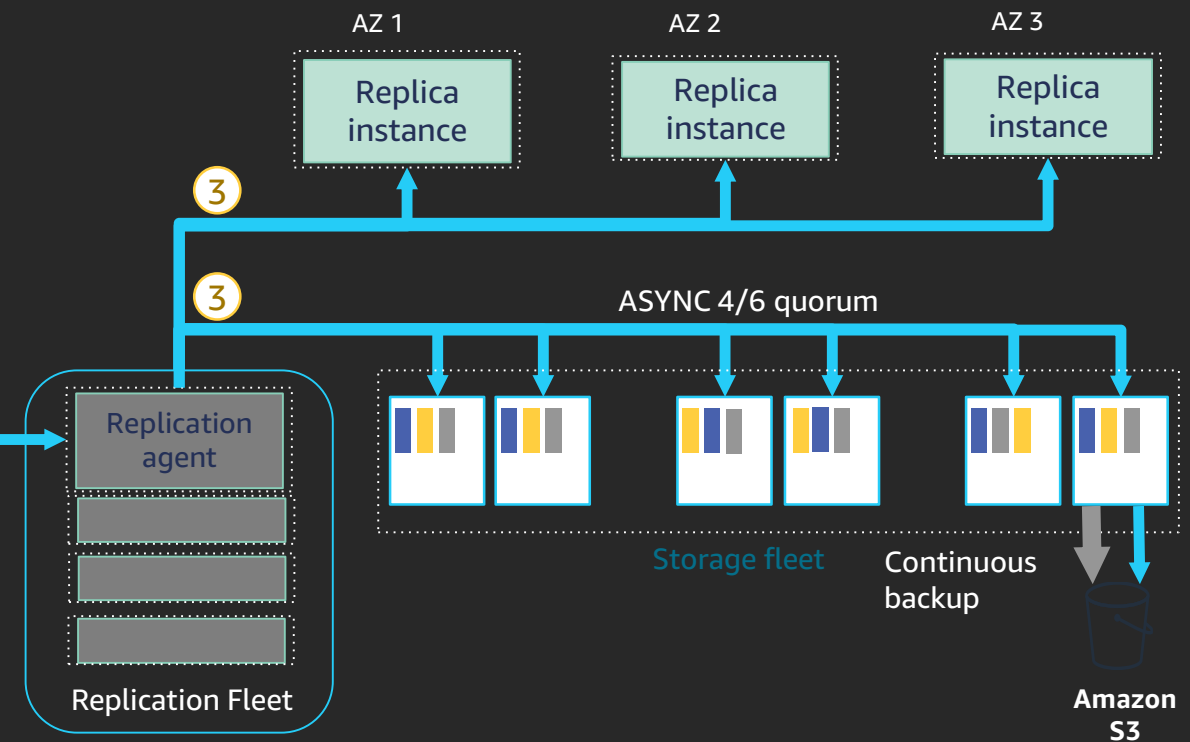
# Global databases

# Global physical replication

## Primary region



## Secondary region



- ① Primary instance sends log records in parallel to storage nodes, replica instances, and replication server
- ② Replication server streams log records to replication agent in secondary region
- ③ Replication agent sends log records in parallel to storage nodes and replica instances
- ④ Replication server pulls log records from storage nodes to catch up after outages

**High throughput:** Up to 150K writes/second; negligible performance impact

**Low replica lag:** <1 second cross-region replica lag under heavy load

**Fast recovery:** <1 minute to accept full read-write workloads after region failure

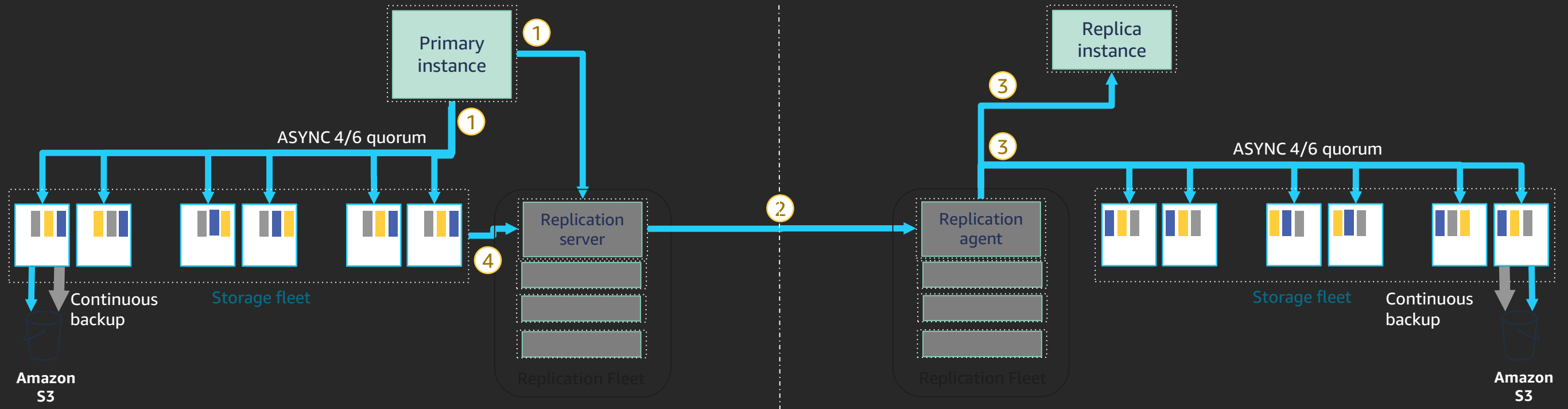


# Global databases - Demo

# Global databases

Primary region (US West)

Secondary region (US East)



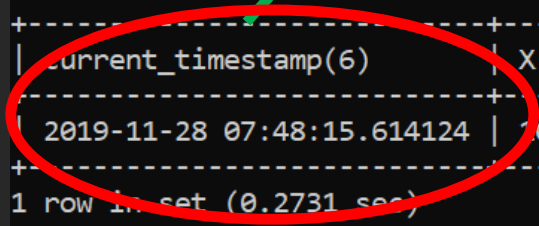
Continuous Inserts

Continuous Reads

```
MySQL Shell
-----+-----+
1 row in set (0.2731 sec)
-----+-----+
| current_timestamp(6) | X |
-----+-----+
| 2019-11-28 07:48:15.602139 | 99960 |
-----+-----+
1 row in set (0.2731 sec)
-----+-----+
| current_timestamp(6) | X |
-----+-----+
| 2019-11-28 07:48:15.605168 | 99970 |
-----+-----+
1 row in set (0.2731 sec)
-----+-----+
| current_timestamp(6) | X |
-----+-----+
| 2019-11-28 07:48:15.608287 | 99980 |
-----+-----+
1 row in set (0.2731 sec)
-----+-----+
| current_timestamp(6) | X |
-----+-----+
| 2019-11-28 07:48:15.611065 | 99990 |
-----+-----+
1 row in set (0.2731 sec)
-----+-----+
| current_timestamp(6) | X |
-----+-----+
| 2019-11-28 07:48:15.614124 | 100000 |
-----+-----+
1 row in set (0.2731 sec)
Query OK, 0 rows affected (0.2731 sec)
MySQL global-reinvent-1-instance-1 reinvent SQL >
```

**U S West  
(Writer)**

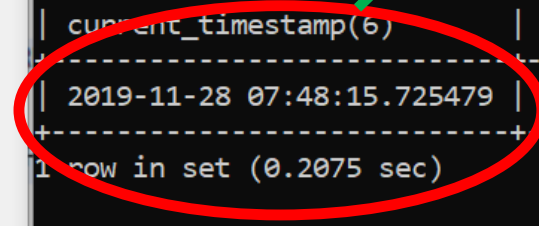
**2019-11-28  
07:48:15.614.124**



```
MySQL Shell
-----+-----+
1 row in set (0.2075 sec)
-----+-----+
| current_timestamp(6) | loopCount | @max |
-----+-----+
| 2019-11-28 07:48:15.592589 | 3798 | 99709 |
-----+-----+
1 row in set (0.2075 sec)
-----+-----+
| current_timestamp(6) | loopCount | @max |
-----+-----+
| 2019-11-28 07:48:15.660000 | 3800 | 99719 |
-----+-----+
1 row in set (0.2075 sec)
-----+-----+
| current_timestamp(6) | loopCount | @max |
-----+-----+
| 2019-11-28 07:48:15.693188 | 3801 | 99989 |
-----+-----+
1 row in set (0.2075 sec)
-----+-----+
| current_timestamp(6) | loopCount | @max |
-----+-----+
| 2019-11-28 07:48:15.725479 | 3802 | 100000 |
-----+-----+
1 row in set (0.2075 sec)
Query OK, 0 rows affected (0.2075 sec)
MySQL global-reinvent-1-instance-1-us-east-1a reinvent SQL >
```

**U S East  
(Reader)**

**2019-11-28  
07:48:15.725.479**



**Replication Lag  
~110ms**



# Fast database cloning

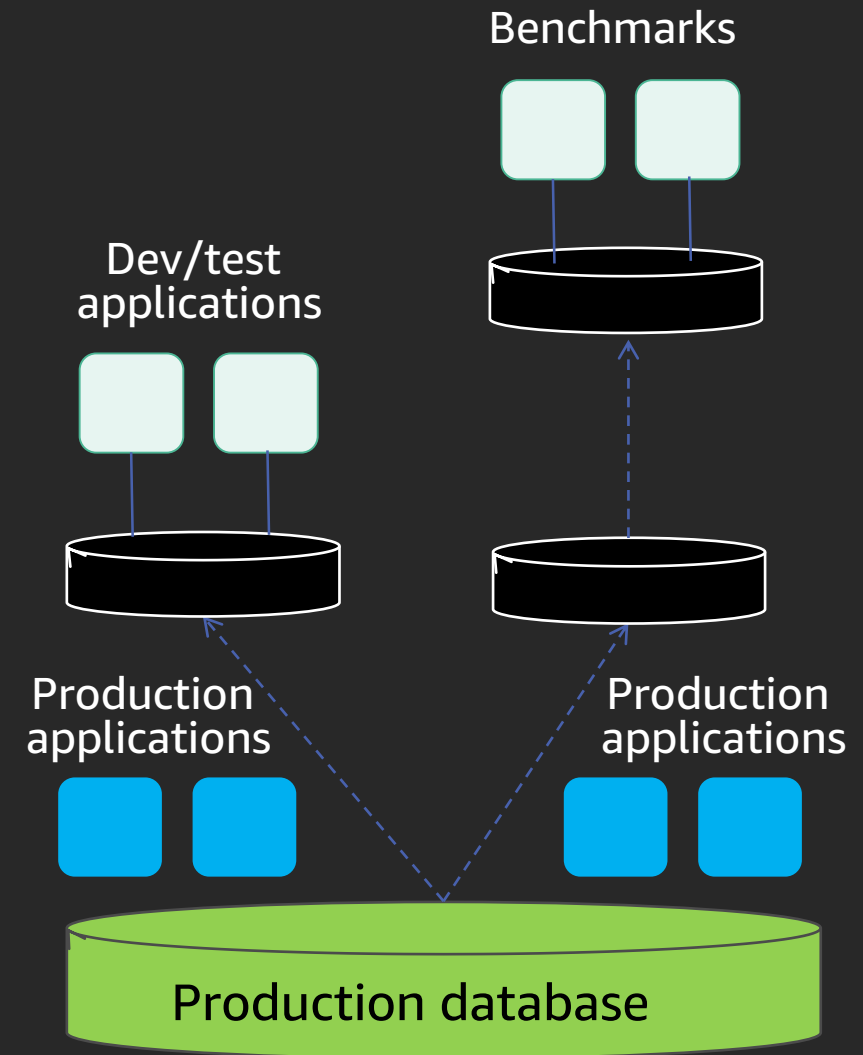
# Fast database cloning

Create a copy of a database without duplicate storage costs

- Creation of a clone is instantaneous because it doesn't require deep copy
- Data copy happens only on write, when original and cloned volume data differ

## Typical use cases

- Clone a production database to run tests
- Reorganize a database
- Save a point-in-time snapshot for analysis without impacting production system

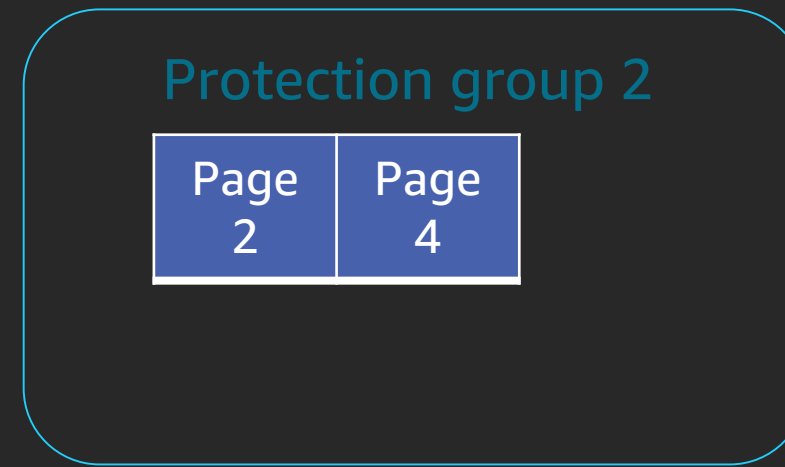
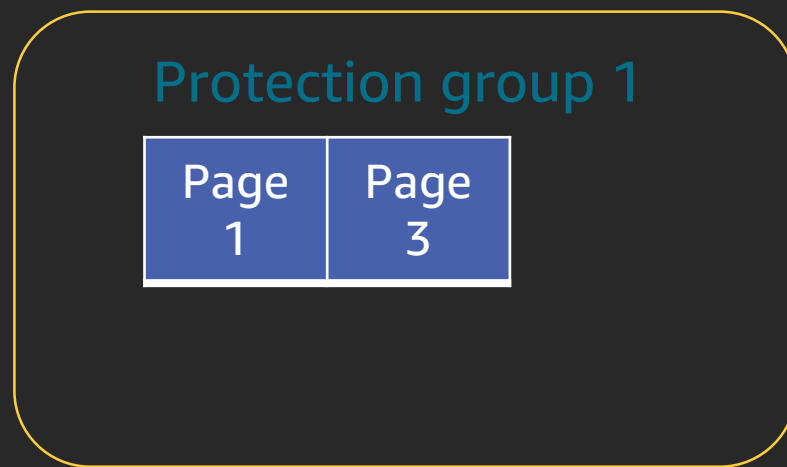


# Database cloning: How does it work?



Both databases reference the same pages on the shared distributed storage system

Shared distributed storage system



# Database cloning: How does it work?



As databases diverge, new pages are added appropriately to each database while still referencing pages common to both databases

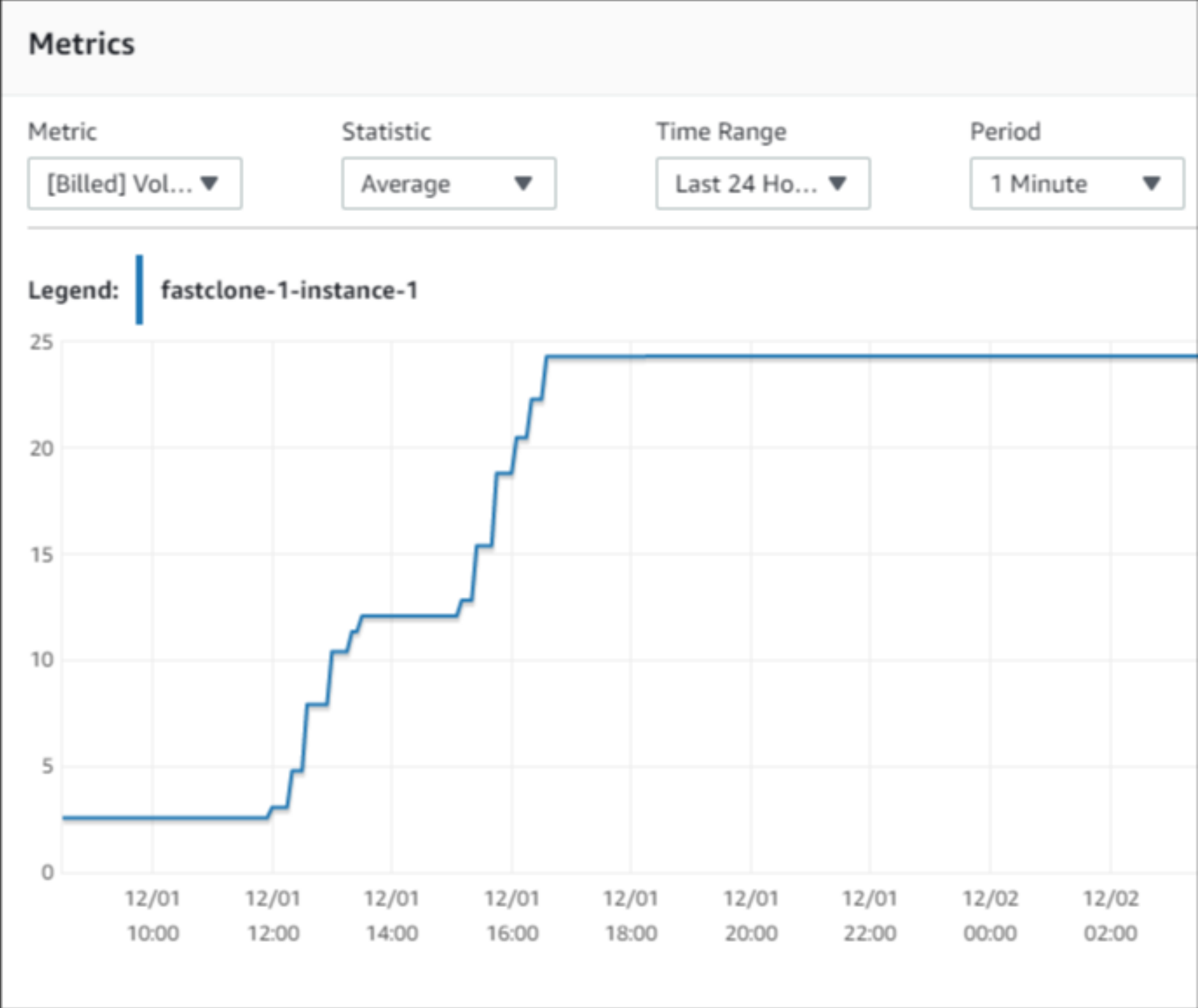
## Shared distributed storage system



# Fast database cloning - Demo

# Database State

- ~40M rows
- ~25GB space used



# Clone

The screenshot displays the AWS Management Console interface for an Amazon RDS instance named 'fastclone-1'. The left-hand navigation pane shows various RDS-related options, with 'Databases' selected. The main content area shows the instance details, including a table of related databases. The 'Monitoring' tab is active, showing a CloudWatch graph for the instance. The 'Actions' dropdown menu is open, and the 'Create clone' option is circled in red.

**Amazon RDS** Services Resource Groups

Admin/kaunanda-Isengard @ 8... N. California Support

RDS > Databases > fastclone-1

## fastclone-1

Modify Actions

Stop  
Delete  
Upgrade now  
Upgrade at next window  
Add reader  
Create cross region read replica  
**Create clone**  
Promote  
Restore to point in time  
Backtrack  
Add replica auto scaling

**Related**

Filter databases

DB identifier	Role	Engine	Region & AZ	Size	Status
fastclone-1	Regional	Aurora MySQL	us-west-1	1 instance	Available
fastclone-1-instance-1	Writer	Aurora MySQL	us-west-1c	db.r5.large	Available

Connectivity & security **Monitoring** Logs & events Configuration Maintenance & backups Tags

CloudWatch (46) Add instance to compare Monitoring Last Hour

Legend: fastclone-1-instance-1

vo

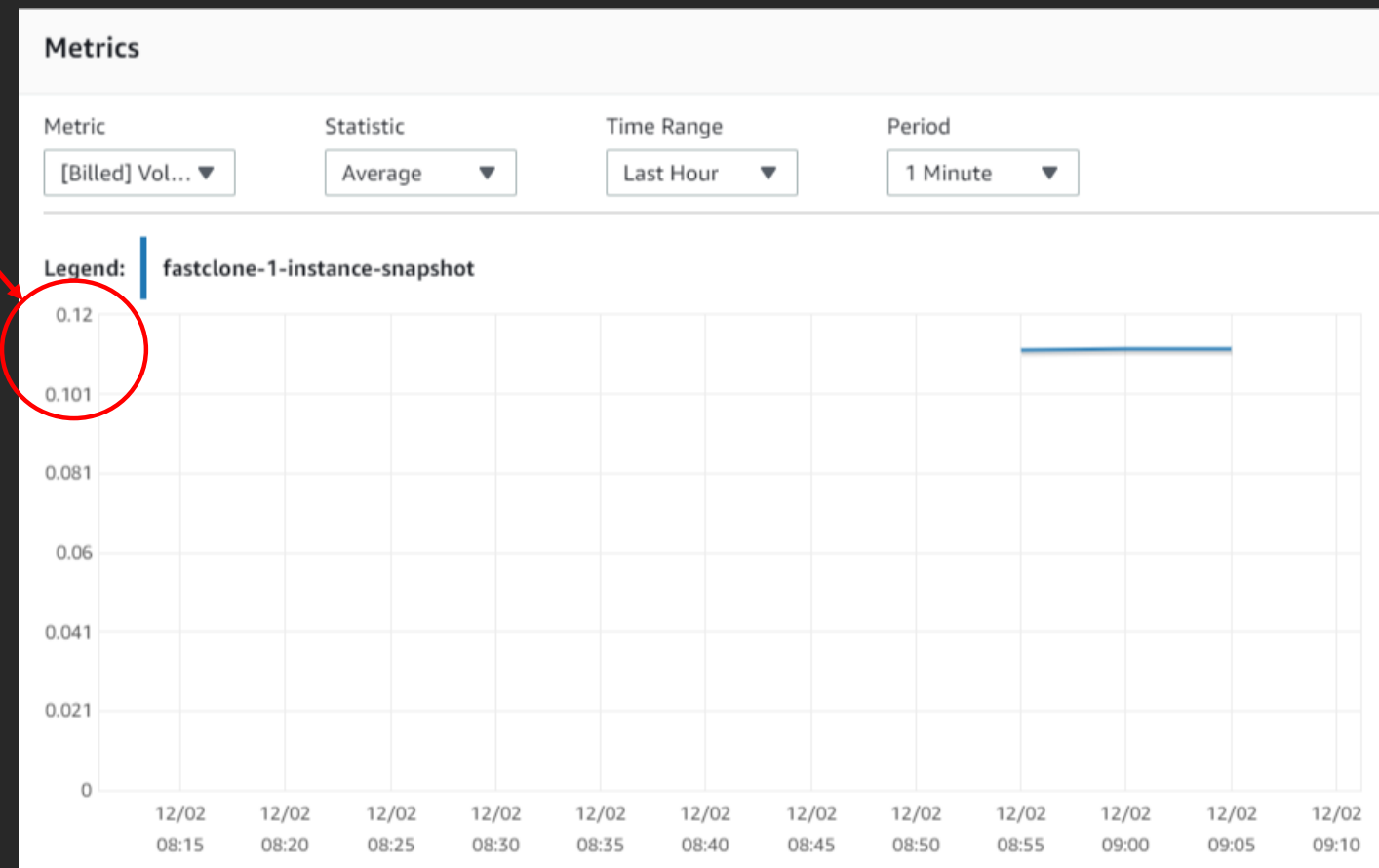
# Database State (Cloned)

- See ~40M rows
- ~0.1GB space used in clone!  
(vs. 25GB)

The screenshot shows the MySQL Workbench interface. The query window contains the following SQL code:

```
1 • show databases;  
2 • use fastclone;  
3 • select count(*) from tw;
```

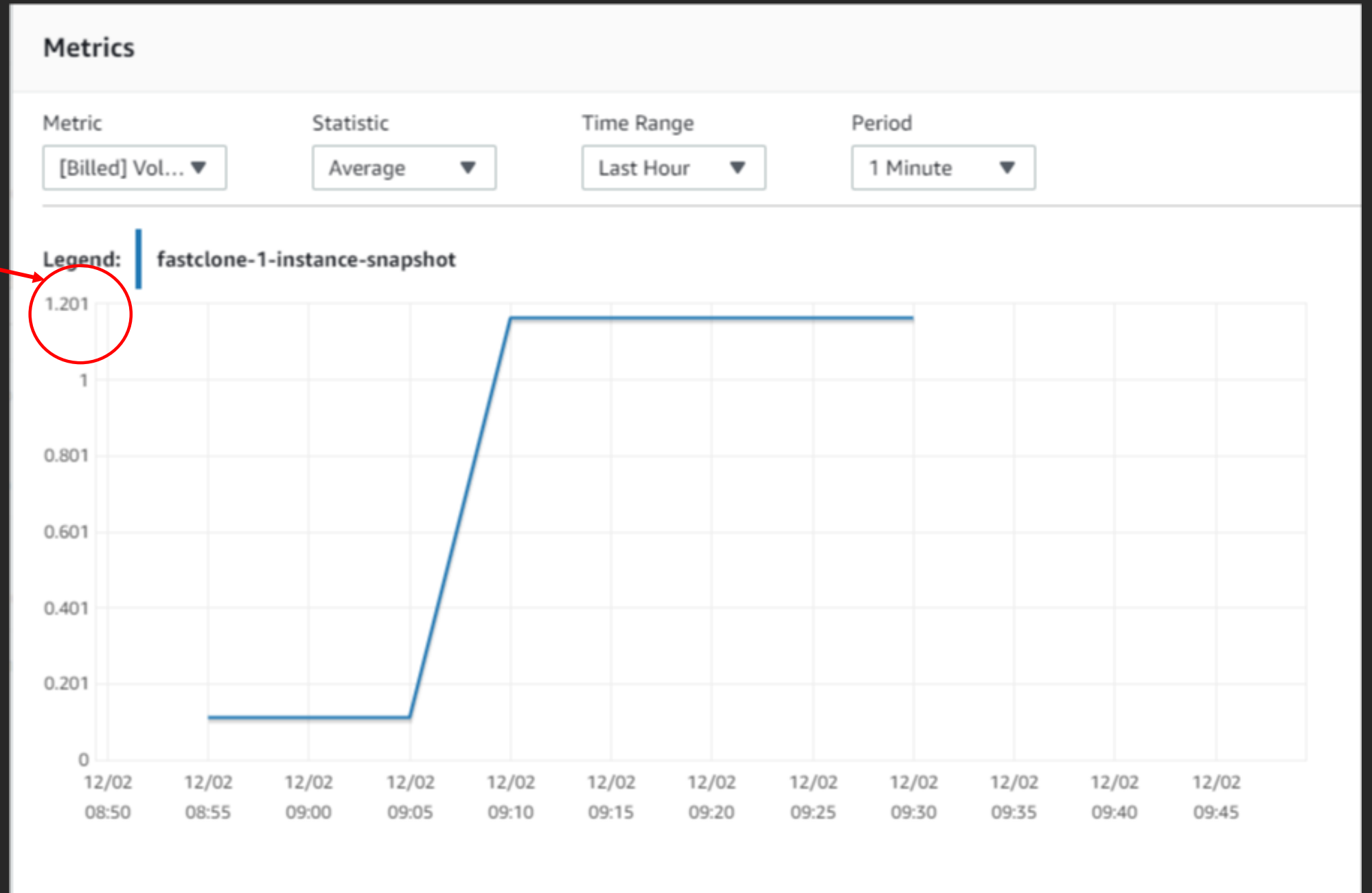
The result grid shows a single row with the value 39999996. A red circle highlights this value, and a red arrow points from the text '~40M rows' in the list to it.





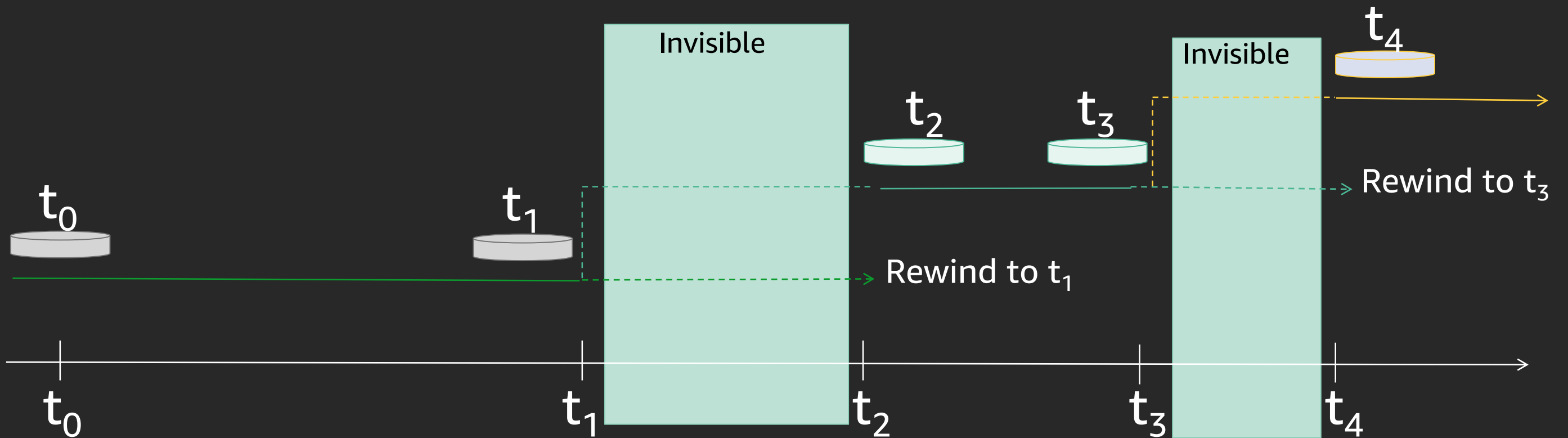
# Database State (Cloned)

- Update ~10M rows
- **Now ~1.2 GB! vs 25GB**



# Database backtrack

# Database backtrack



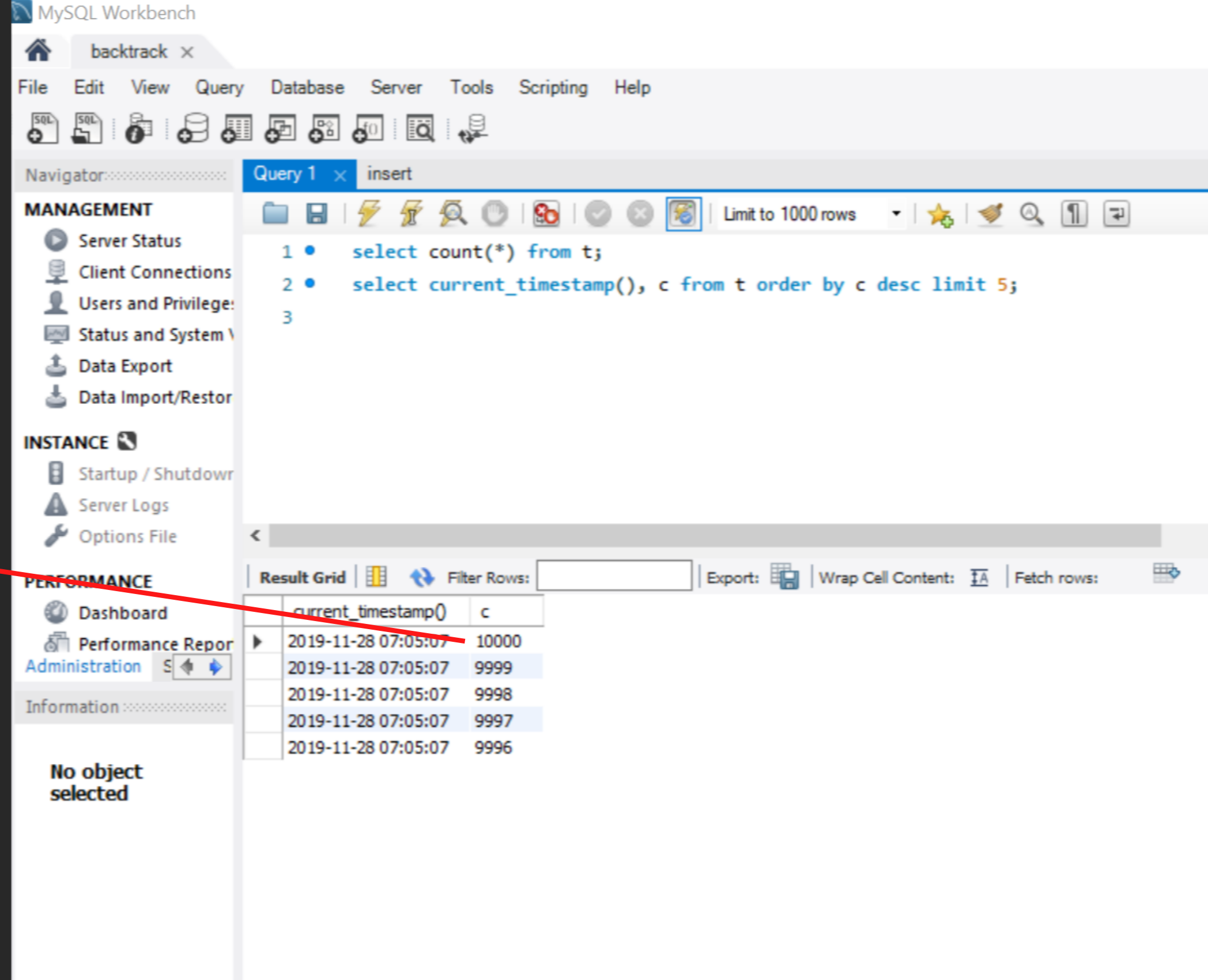
Backtrack is a quick way to bring the database to a particular point in time without having to restore from backups

- Rewinding the database to quickly recover from unintentional DML/DDL operations
- Rewind multiple times to determine the desired point in time in the database state; for example, quickly iterate over schema changes without having to restore multiple times

# Backtrack - Demo

# Database State (@07:05:07)

Table with ~10K  
rows



The screenshot shows the MySQL Workbench interface. The left sidebar contains navigation options under 'MANAGEMENT', 'INSTANCE', and 'PERFORMANCE'. The main window displays a query editor with two SQL statements. The first statement is a query to count rows in table 't'. The second statement is a query to select the current timestamp and column 'c' from table 't', ordered by 'c' in descending order with a limit of 5. Below the query editor, the 'Result Grid' shows the output of the second query, displaying 5 rows of data.

```
1 • select count(*) from t;  
2 • select current_timestamp(), c from t order by c desc limit 5;  
3
```

current_timestamp()	c
2019-11-28 07:05:07	10000
2019-11-28 07:05:07	9999
2019-11-28 07:05:07	9998
2019-11-28 07:05:07	9997
2019-11-28 07:05:07	9996

# Database State (@07:09:58)

- Added a column
- Added two rows

Oops!

MySQL Workbench

backtrack x

File Edit View Query Database Server Tools Scripting Help

Navigator: Query 1 x insert

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Report
- Administration

Information: No object selected

```
1 • select count(*) from t;  
2 • select current_timestamp(), c from t order by c desc limit 5;  
3 • alter table t add (c1 int default 99);  
4 • insert into t values (100001, 82);  
5 • insert into t values (100002, 83);  
6 • select count(*) from t;  
7 • select current_timestamp(), c from t order by c desc limit 5;
```

Result Grid

current_timestamp()	c
2019-11-28 07:09:58	100002
2019-11-28 07:09:58	100001
2019-11-28 07:09:58	10000
2019-11-28 07:09:58	9999
2019-11-28 07:09:58	9998

# Let's Backtrack

RDS > Databases > reinvent-1

## reinvent-1

Modify

Actions ▼

### Related

Filter databases

DB identifier	Role	Engine	Region & AZ	Size	Status
reinvent-1	Regional	Aurora MySQL	us-west-1	2 instances	✔ A
reinvent-backtrack-instance-1	Writer	Aurora MySQL	us-west-1a	db.r5.large	✔ A
reinvent-backtrack-instance-1-us-west-1c	Reader	Aurora MySQL	us-west-1c	db.r5.large	✔ A

- Stop
- Delete
- Upgrade now
- Upgrade at next window
- Add reader
- Create cross region read replica
- Create clone
- Promote
- Restore to point in time
- Backtrack**
- Add replica auto scaling

Connectivity & security

Monitoring

Logs & events

Configuration

Maintenance & backups

Tags

# Let's Backtrack

RDS > Databases > reinvent-backtrack-instance-1 > Backtrack DB cluster

### Backtrack DB cluster

Rewinds the DB cluster to a previous point in time without creating a new DB cluster.  
Earliest restorable time is November 26, 2019 at 8:05:51 PM UTC-8 (Local) ⓘ

Date:  Time:  :  :  UTC-8  
The next available time will be used if the specified time is not available.

⚠ Your DB cluster is unavailable during the Backtrack process, which typically takes a few minutes.

07:06:00

○	[-] reinvent-1	Regional	Aurora MySQL	us-west-1	2 instances	backtracking	-
○	reinvert-backtrack-instance-1	Writer	Aurora MySQL	us-west-1a	db.r5.large	backtracking	▬ 4.00%
○	reinvert-backtrack-instance-1-us-west-1c	Reader	Aurora MySQL	us-west-1c	db.r5.large	backtracking	▬ 3.00%



# Database State (after Backtrack)

Rows added missing  
Single column

The good old  
state!

The screenshot shows the MySQL Workbench interface. The query editor contains the following SQL statement:

```
1 • select current_timestamp(), c from t order by c desc limit 5;
```

The result grid below shows the following data:

current_timestamp()	c
2019-11-28 07:32:08	10000
2019-11-28 07:32:08	9999
2019-11-28 07:32:08	9998
2019-11-28 07:32:08	9997
2019-11-28 07:32:08	9996

The interface also shows a sidebar with navigation options like Server Status, Client Connections, Users and Privileges, Status and System, Data Export, Data Import/Restore, Instance, Performance, and Administration. The bottom status bar indicates 'No object selected'.

# References

# Publications

Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. *In SIGMOD 2017*

Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes. *In SIGMOD 2018*

# Thank you!

**Murali & Tobias**

brahmade@amazon.com

tobiasql@amazon.com



Please complete the session survey in the mobile app.