
AWS
re:Invent

DAT401

Amazon Aurora storage demystified

Anupriya Mathur

Senior Product Manager, Amazon Aurora

AWS

AWS re:Invent

Agenda

What is Amazon Aurora?

Cloud-native database architecture

How storage grows and resizes

Durability at scale

Aurora backups

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

Cloud-native database architecture

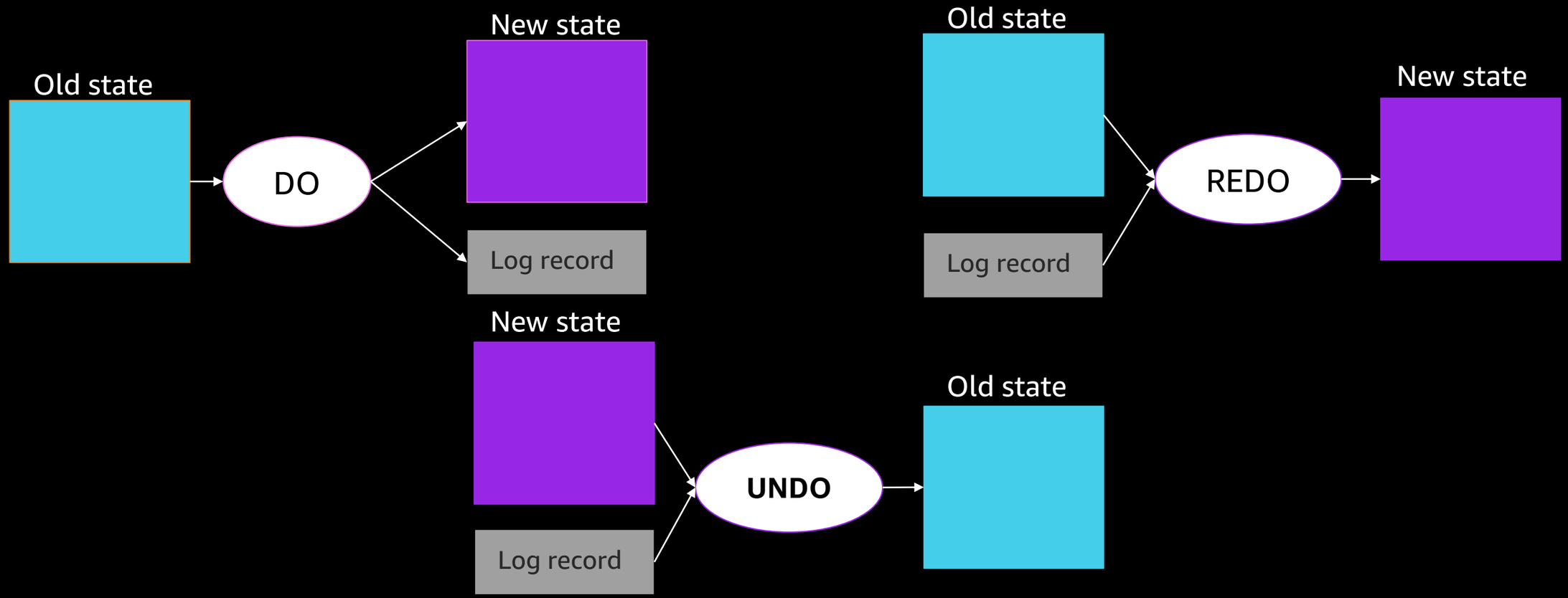
AWS re:Invent

Quick recap: DO-REDO-UNDO protocol

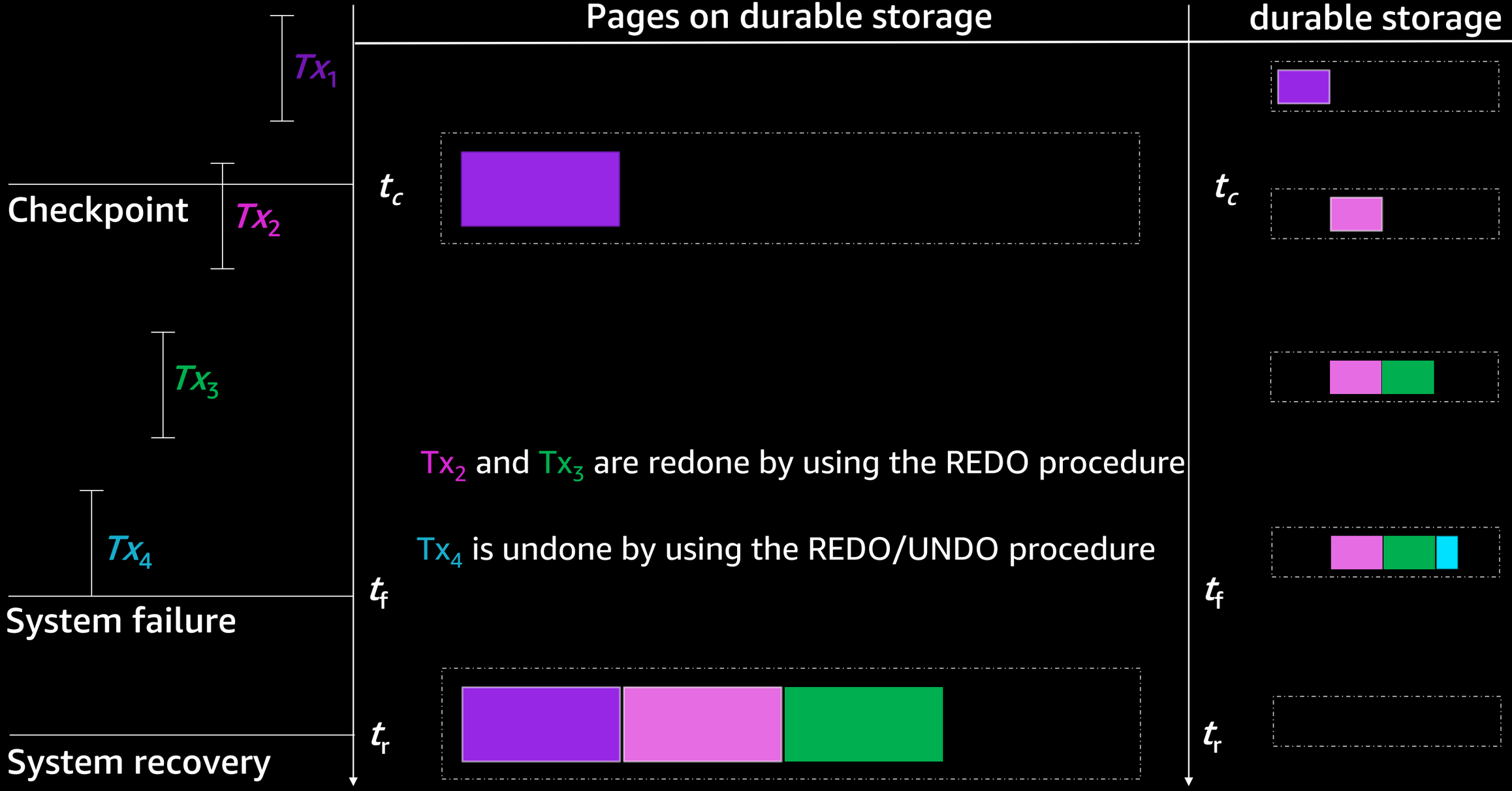
Data is modified “in-place” in the buffer-pool using a DO/REDO/UNDO operation

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

Pages from buffer-pool are written to durable storage (checkpoint) periodically



Quick recap: Crash Recovery

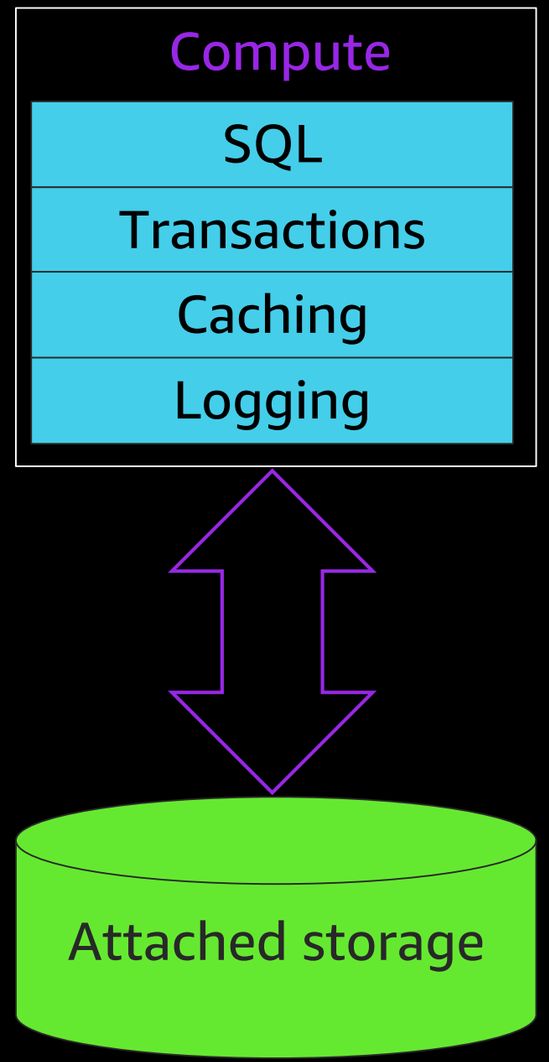


Traditional database architecture

Databases are all about input / output

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

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 the 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

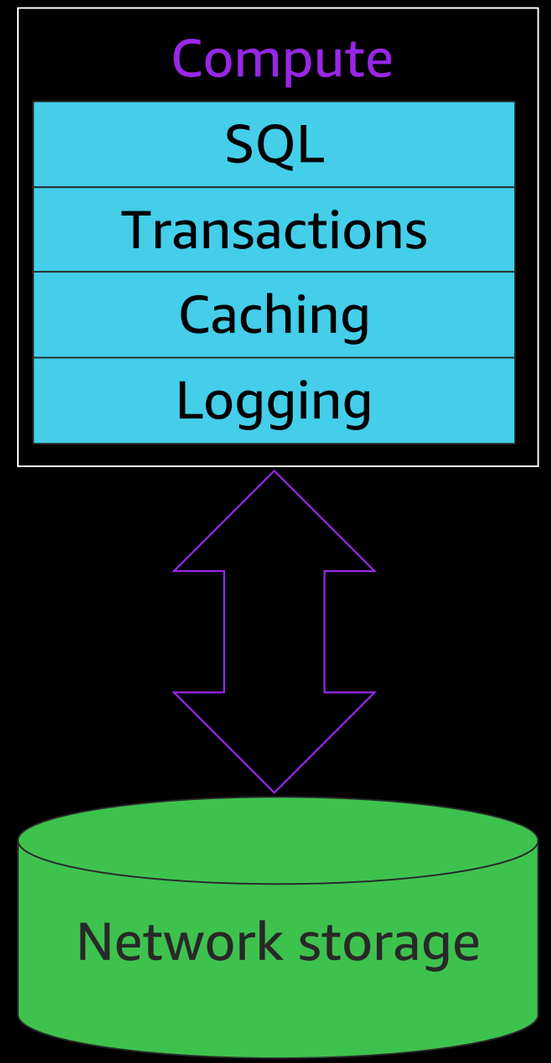
Compute & storage have different lifetimes

Compute instances

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

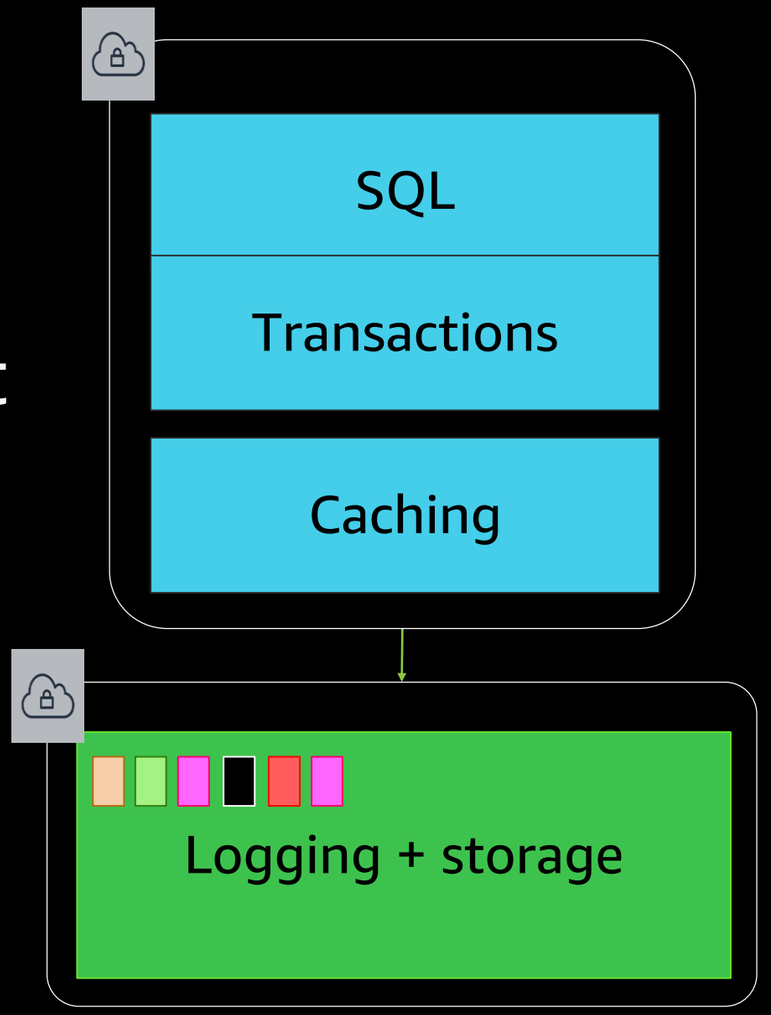
Storage, on the other hand, has to be long-lived

Decouple compute and storage for scalability, availability, durability

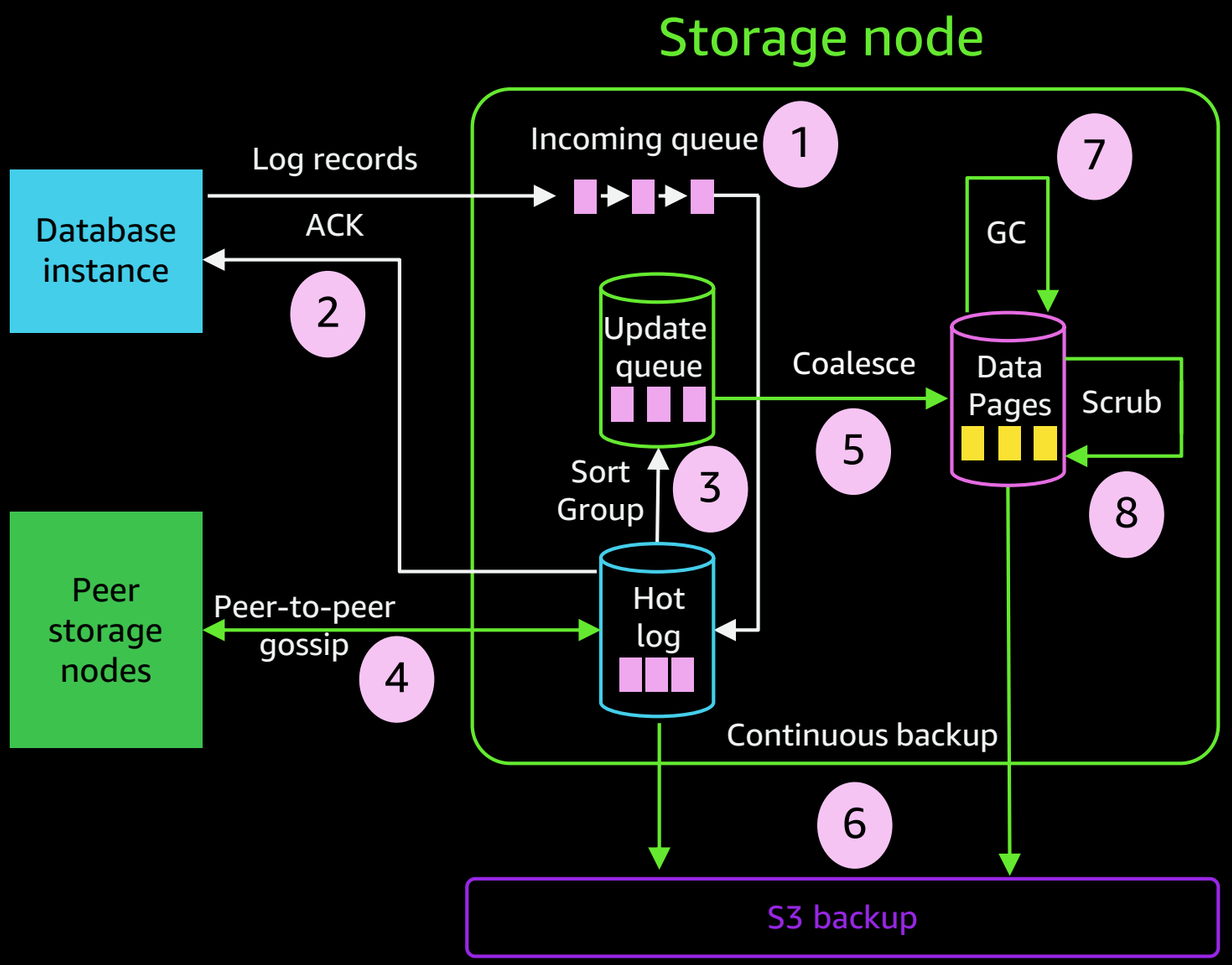


Aurora uses service-oriented architecture

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



I/O flow in Aurora storage node



① Receive log records, 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

How Aurora storage grows and resizes

AWS re:Invent

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, because we can repair a 10 GB segment in less than a minute

Database resizing

Volume size increases when data is written

- Grows in increments of 10 GB
- Maximum size of a DB cluster is 128 TB

Volume size decreases when data is deleted

- Storage is freed at page granularity
- Resizing happens dynamically and as data is deleted

Monitoring

- Amazon CloudWatch metrics to monitor storage usage: Volume Bytes Used

Example: Database resizing

```
[21:46:49][rdsop][~]$ LD_LIBRARY_PATH=/rdsdbbin/aurora/lib /rdsdbbin/aurora/bin/psql -U rdsadmin -p $(awk '/^port = [0-9]+$/ {print $3}' /rdsdbdata/config/postgresql.conf)
Password for user rdsadmin:
psql (11.8)
Type "help" for help.

rdsadmin=# create database vsTest;
CREATE DATABASE
rdsadmin=# CREATE TABLE t (id int, pkpad char(64) not null, x int, r4k int, r1m int, did int, idxpad varchar(256), vc varchar) with (fillfactor = 10);
CREATE TABLE
rdsadmin=# INSERT INTO t SELECT generate_series(1, 132000000, 1), ' ', 0, floor(4096*random()), floor(1048576*random()), floor(2147483646*random()), ' '::varchar(128), 'X'::varchar(256);
INSERT 0 132000000
rdsadmin=# \q
[22:17:52][rdsop][~]$ LD_LIBRARY_PATH=/rdsdbbin/aurora/lib /rdsdbbin/aurora/bin/psql -U rdsadmin -p $(awk '/^port = [0-9]+$/ {print $3}' /rdsdbdata/config/postgresql.conf)
Password for user rdsadmin:
psql (11.8)
Type "help" for help.

rdsadmin=# CREATE TABLE t1 (id int, pkpad char(64) not null, x int, r4k int, r1m int, did int, idxpad varchar(256), vc varchar) with (fillfactor = 10);
CREATE TABLE
rdsadmin=# INSERT INTO t1 SELECT generate_series(1, 132000000, 1), ' ', 0, floor(4096*random()), floor(1048576*random()), floor(2147483646*random()), ' '::varchar(128), 'X'::varchar(256);
INSERT 0 132000000
rdsadmin=# INSERT INTO t1 SELECT generate_series(1, 132000000, 1), ' ', 0, floor(4096*random()), floor(1048576*random()), floor(2147483646*random()), ' '::varchar(128), 'X'::varchar(256);
INSERT 0 132000000
rdsadmin=# INSERT INTO t SELECT generate_series(1, 132000000, 1), ' ', 0, floor(4096*random()), floor(1048576*random()), floor(2147483646*random()), ' '::varchar(128), 'X'::varchar(256);
INSERT 0 132000000
rdsadmin=# INSERT INTO t1 SELECT generate_series(1, 132000000, 1), ' ', 0, floor(4096*random()), floor(1048576*random()), floor(2147483646*random()), ' '::varchar(128), 'X'::varchar(256);
INSERT 0 132000000
rdsadmin=# drop table t1;
DROP TABLE
rdsadmin=# drop table t;
DROP TABLE
```

Example: Database resizing

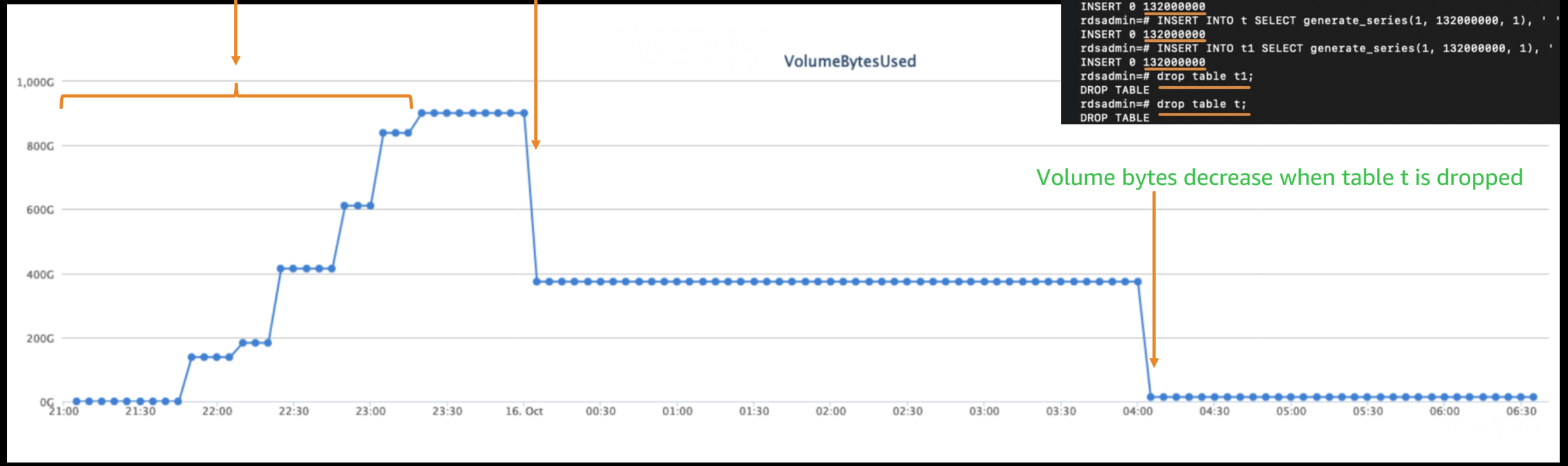
Volume bytes increase with inserts in tables t and t1

Volume bytes decrease when table t1 is dropped

```
[21:46:49][rdsop][~]$ LD_LIBRARY_PATH=/rdsdbbin/aurora/lib /rdsdbbin/
Password for user rdsadmin:
psql (11.8)
Type "help" for help.

rdsadmin=# create database vsTest;
CREATE DATABASE
rdsadmin=# CREATE TABLE t (id int, pkpad char(64) not null, x int, r4
CREATE TABLE
rdsadmin=# INSERT INTO t SELECT generate_series(1, 132000000, 1), ' '
INSERT 0 132000000
rdsadmin=# \q
[22:17:52][rdsop][~]$ LD_LIBRARY_PATH=/rdsdbbin/aurora/lib /rdsdbbin/
Password for user rdsadmin:
psql (11.8)
Type "help" for help.

rdsadmin=# CREATE TABLE t1 (id int, pkpad char(64) not null, x int, r
CREATE TABLE
rdsadmin=# INSERT INTO t1 SELECT generate_series(1, 132000000, 1), '
INSERT 0 132000000
rdsadmin=# INSERT INTO t1 SELECT generate_series(1, 132000000, 1), '
INSERT 0 132000000
rdsadmin=# INSERT INTO t SELECT generate_series(1, 132000000, 1), ' '
INSERT 0 132000000
rdsadmin=# INSERT INTO t1 SELECT generate_series(1, 132000000, 1), '
INSERT 0 132000000
rdsadmin=# drop table t1;
DROP TABLE
rdsadmin=# drop table t;
DROP TABLE
```



Volume bytes decrease when table t is dropped

Durability at scale

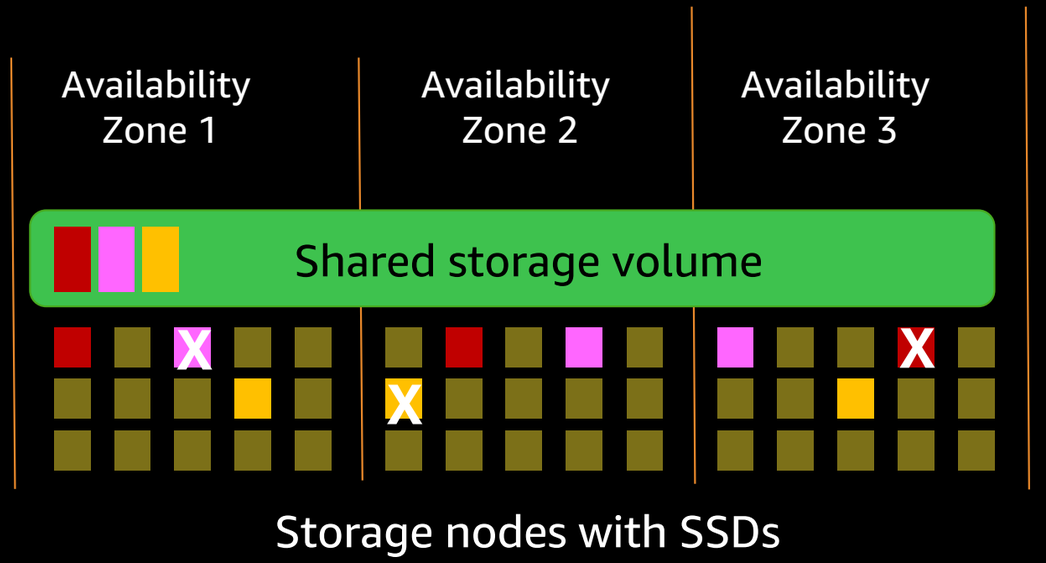
Uncorrelated and independent failures

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

The solution is replication

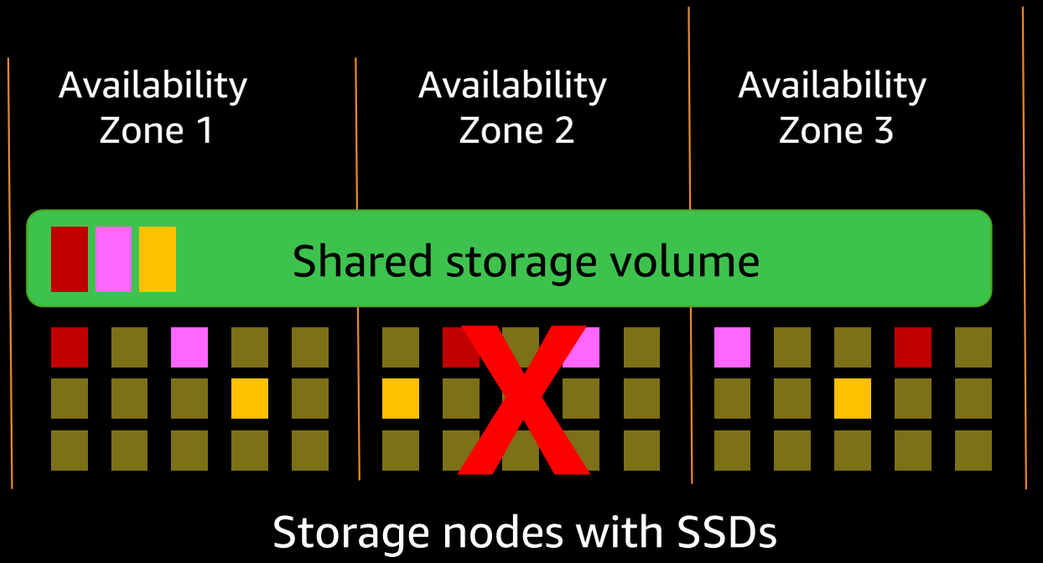
One common straw man

- Replicate 3 ways with 1 copy per AZ
- Use write and read quorums of 2/3



What about AZ failure?

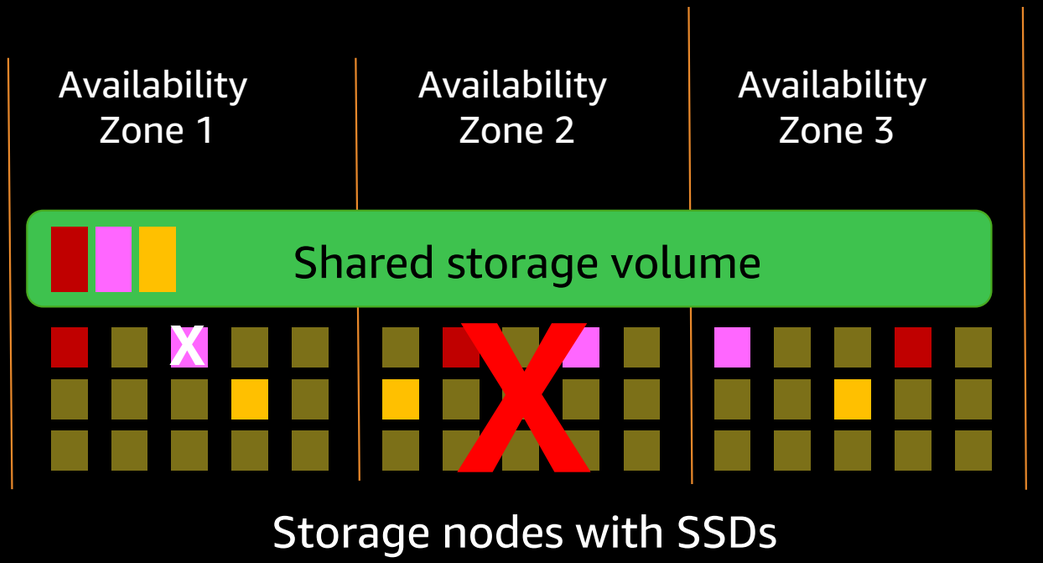
- Still have 2/3 copies
- Can establish quorum
- No data loss



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

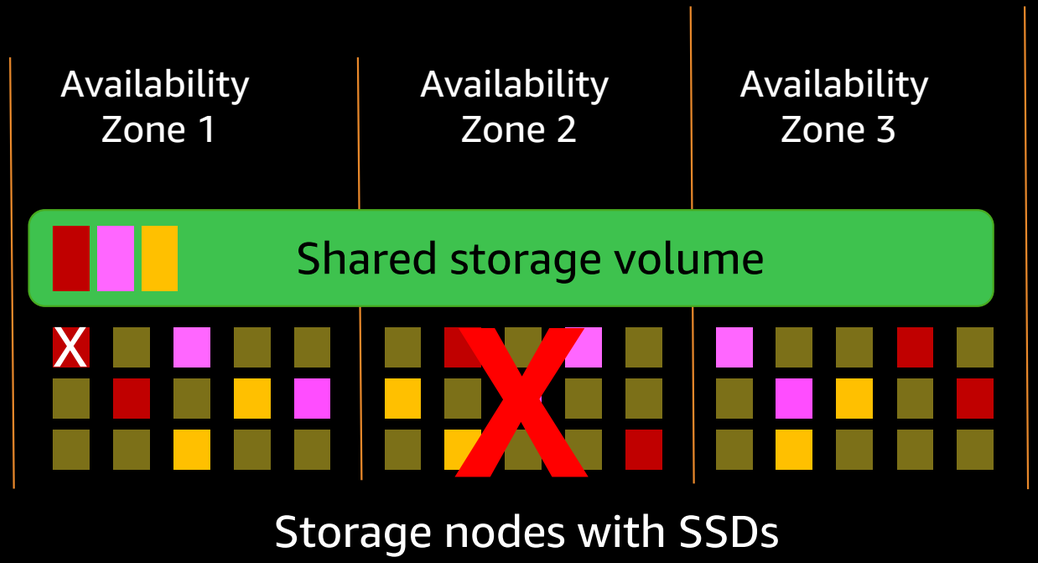
- Write quorum of 4/6

What if an AZ fails?

- Still have 4/6 copies
- Maintain write availability

What if there is an AZ + 1 failure?

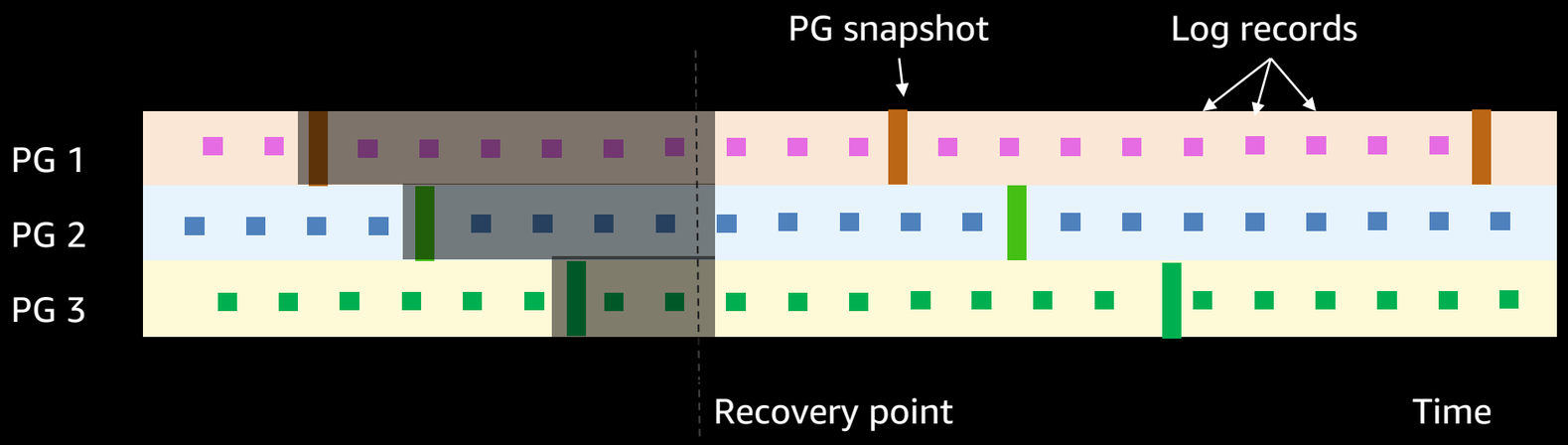
- Still have 3 copies
- No data loss
- Rebuild failed copy by copying from 3 copies
- Recover write availability



Backups

AWS re:Invent

Continuous backups and snapshots



- Take a periodic snapshot of each PG in parallel; stream the redo logs to Amazon S3
- Backup happens continuously without performance or availability impact
- Volume level snapshot is a O(1) operation: a marker in the continuous backup stream
- At restore, retrieve the appropriate PG snapshots and log streams to storage nodes and apply log streams in parallel and asynchronously
- Volume level snapshots can be copied to x-region
- Snapshots can be managed via AWS Backups

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

Related sessions

DAT201

What's new in Amazon Aurora

DAT404

Deep dive on global database for Amazon Aurora

DAT210

Transforming Hilton's reservation system with Amazon Aurora PostgreSQL

AWS re:Invent

Thank you!

Anupriya Mathur

anuma@amazon.com

AWS re:Invent

Please complete the session survey

AWS re:Invent