Amazon Aurora storage

Purpose built storage for databases

Murali Brahmadesam

Director of Engineering, 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

What is Amazon Aurora ?

Enterprise class cloud native database

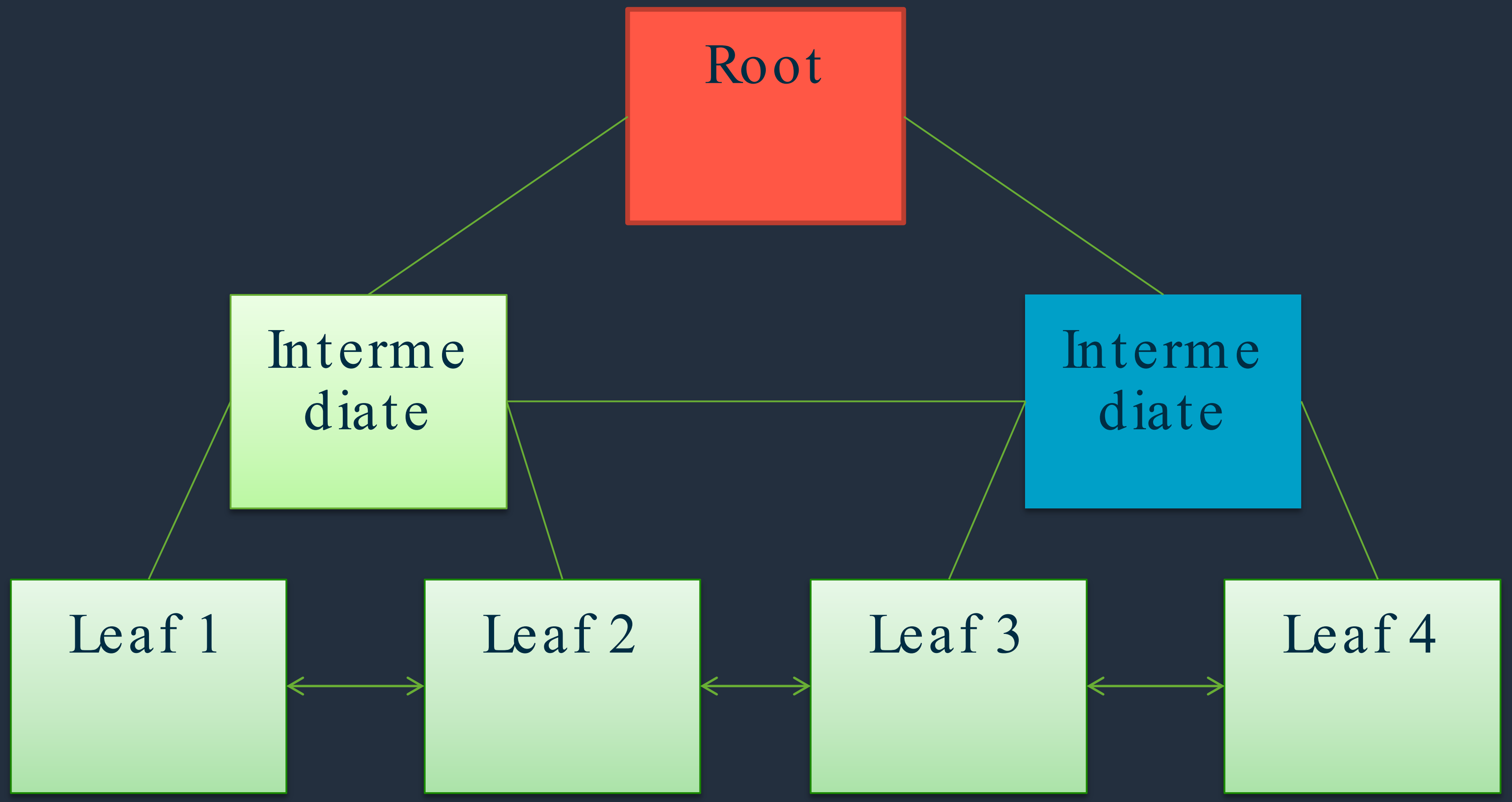
Amazon Aurora

- ☑ 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

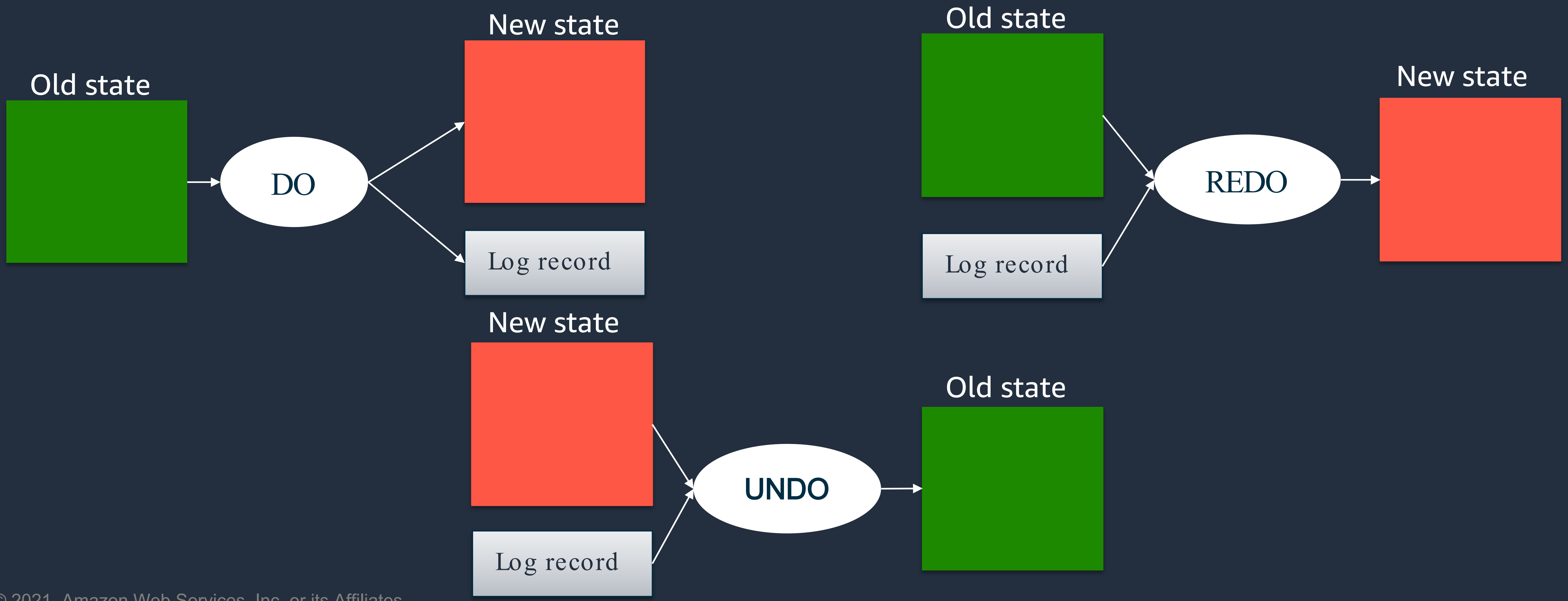
(aka “buffer-pool”)

Pages are serialized into durable storage (aka “checkpoint”) periodically

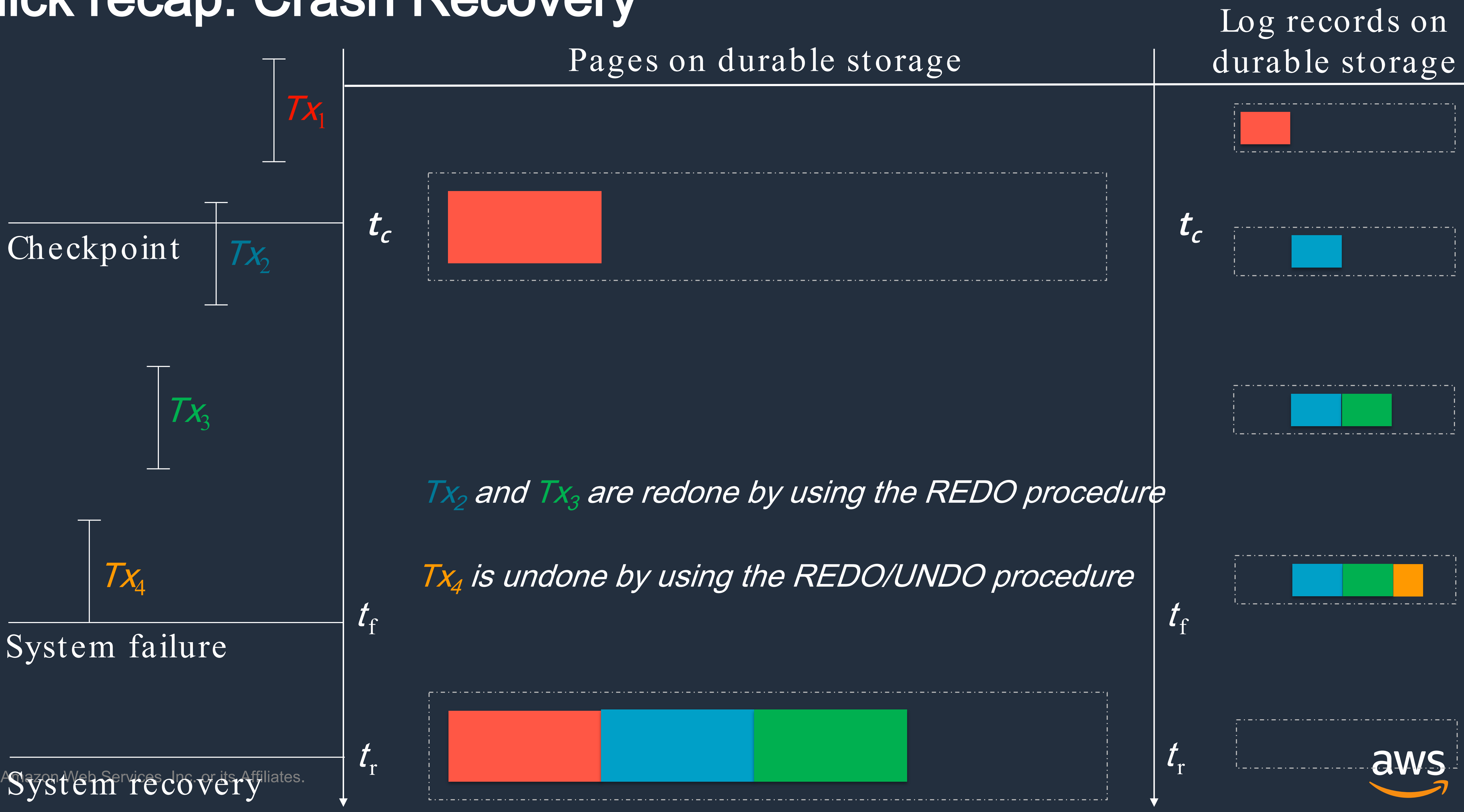
Quick recap: DO -REDO-UNDO protocol

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

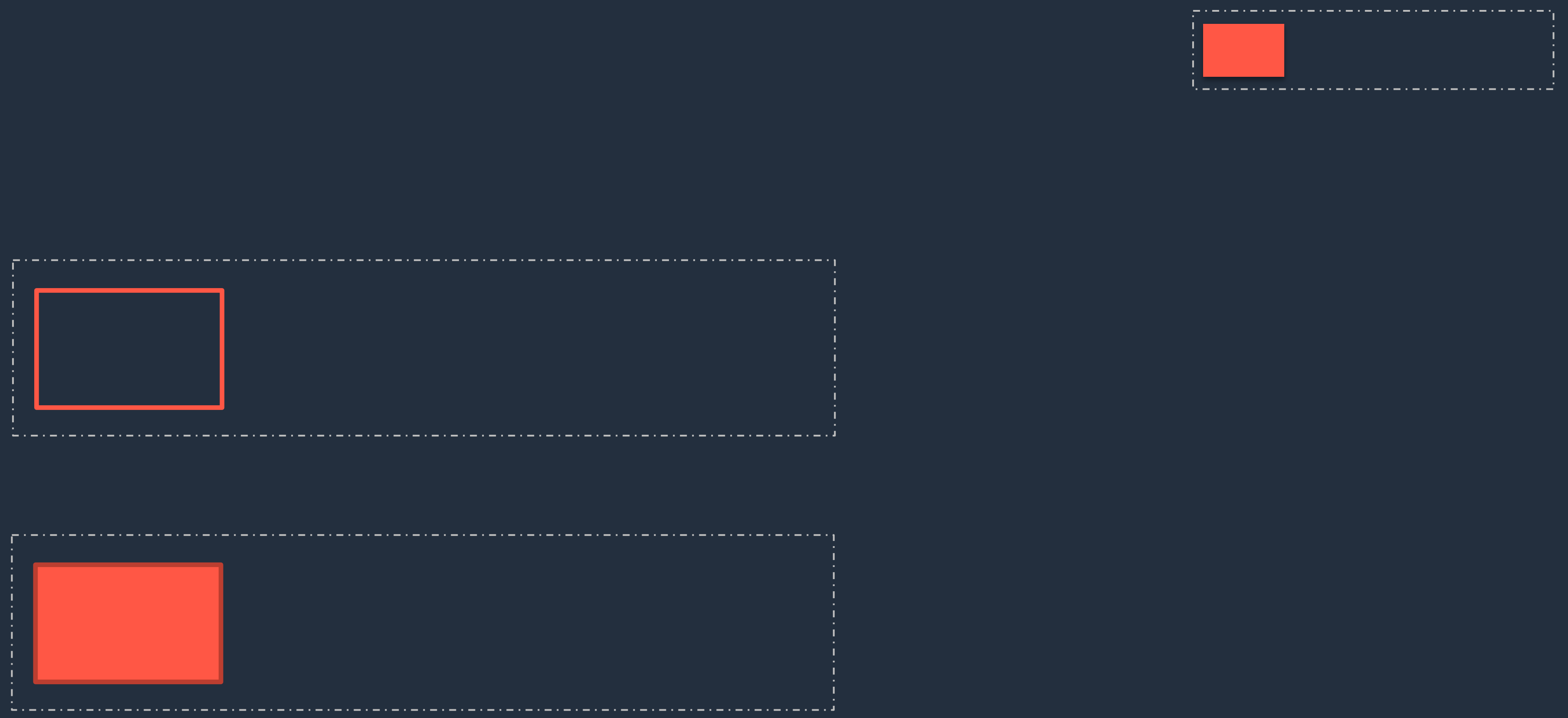
Pages on durable storage

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 << I/ O size (32KB+)

Databases are all about I/O

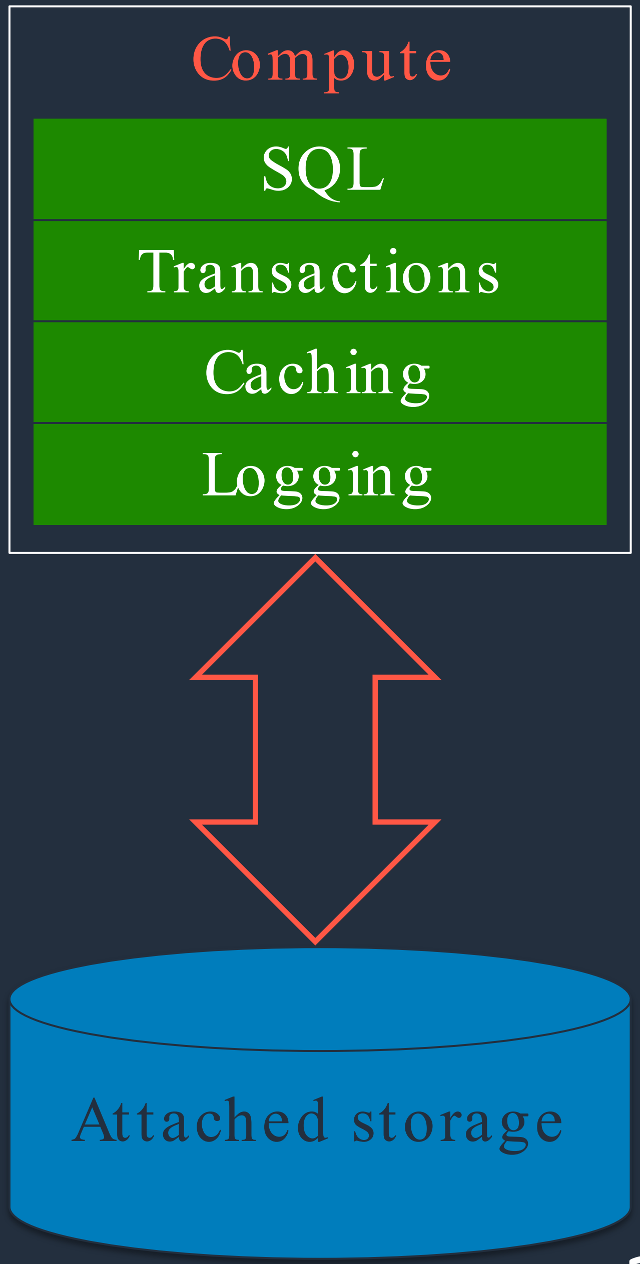
Cloud native database architecture

Traditional database architecture

Databases are all about I/O

Design principles for > 40 years

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

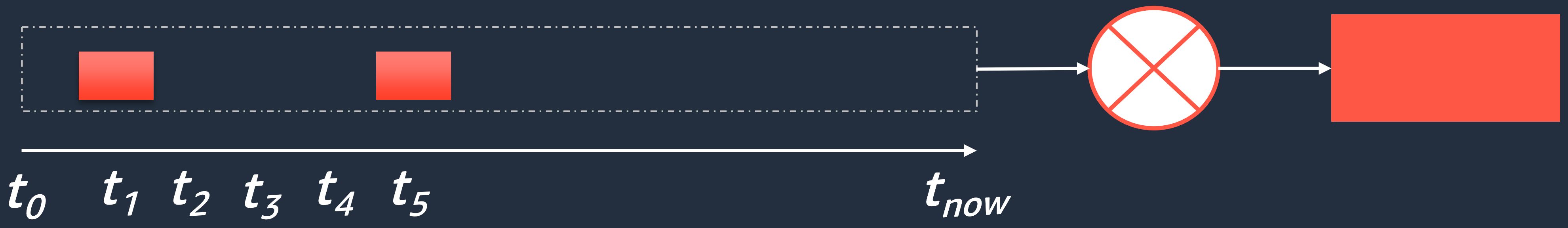


Aurora approach: Log is the database

Log stream from beginning of the database

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

Red-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

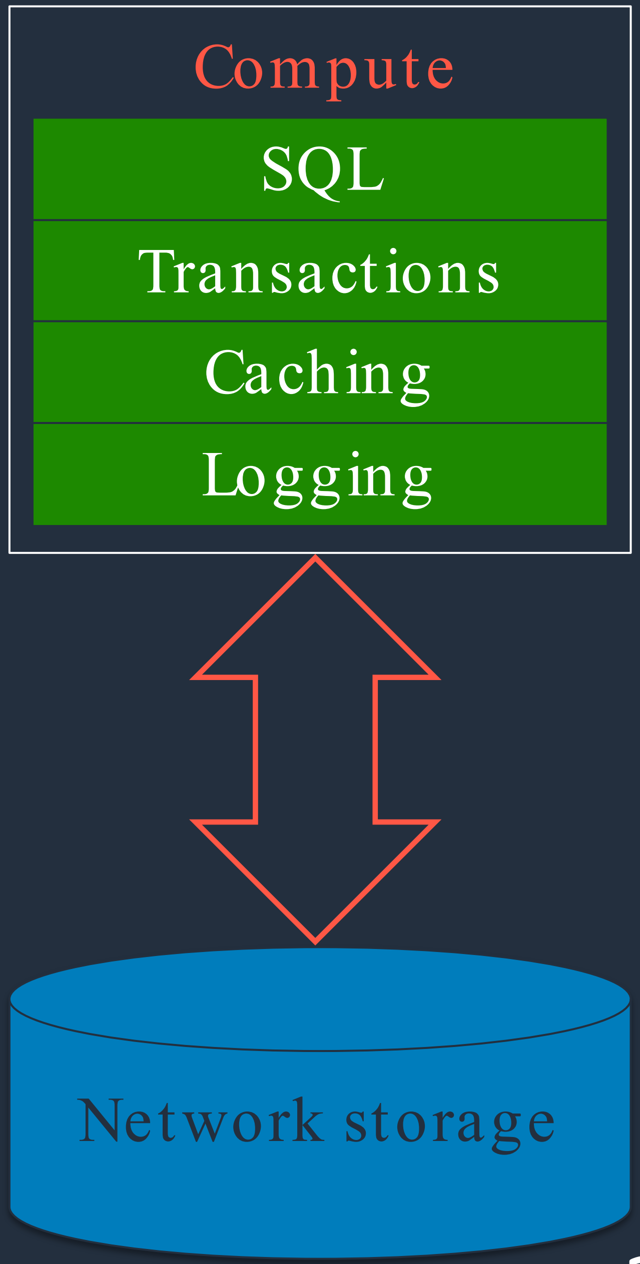
Compute & storage have different lifetimes

Compute instances

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

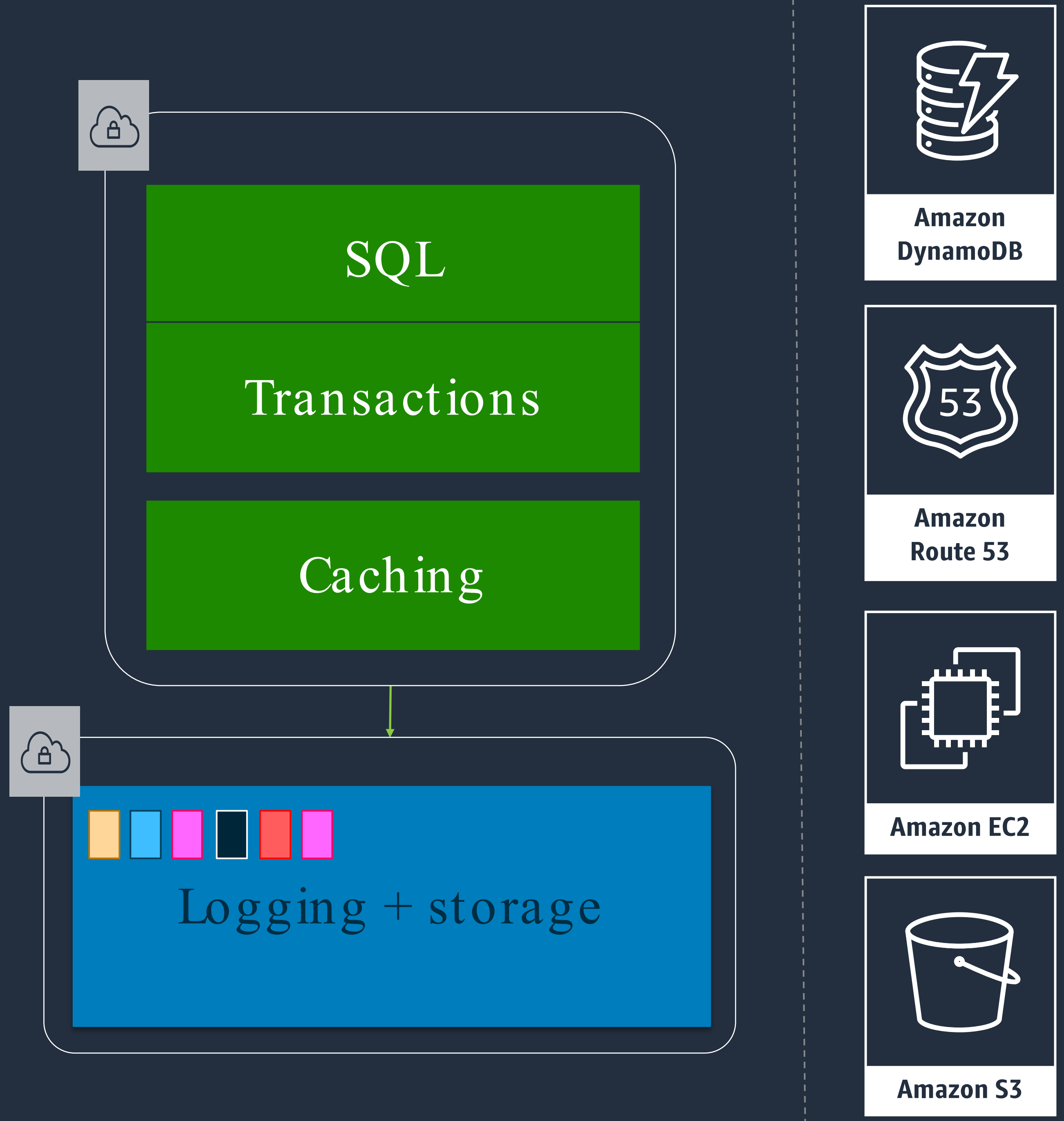
Storage, on the other hand, has to be long-lived

Decouple compute and storage for scalability, availability, durability

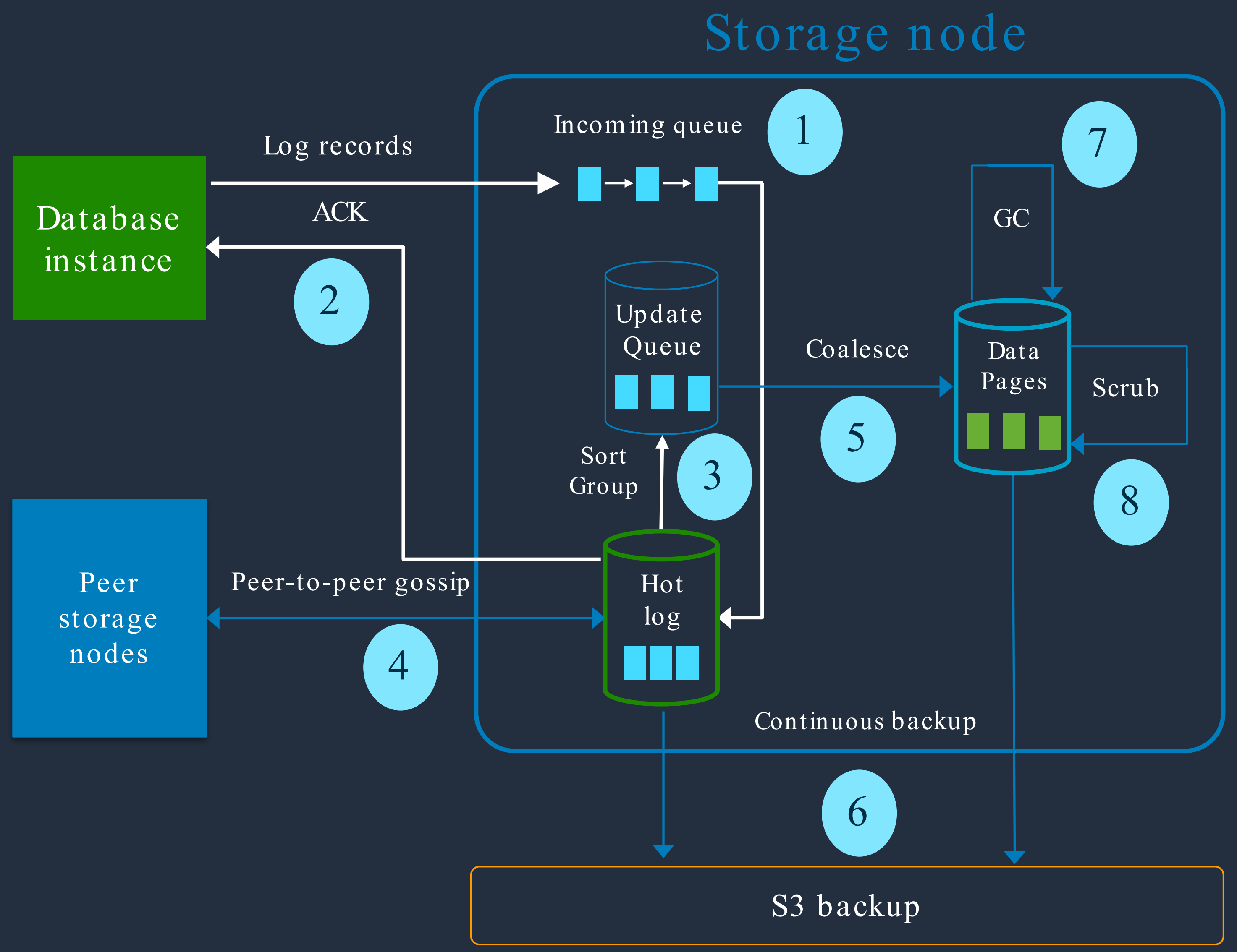


Aurora uses service-oriented architecture

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



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:

- Only steps 1 and 2 are in the foreground latency path rest are asynchronously performed

Durability at scale

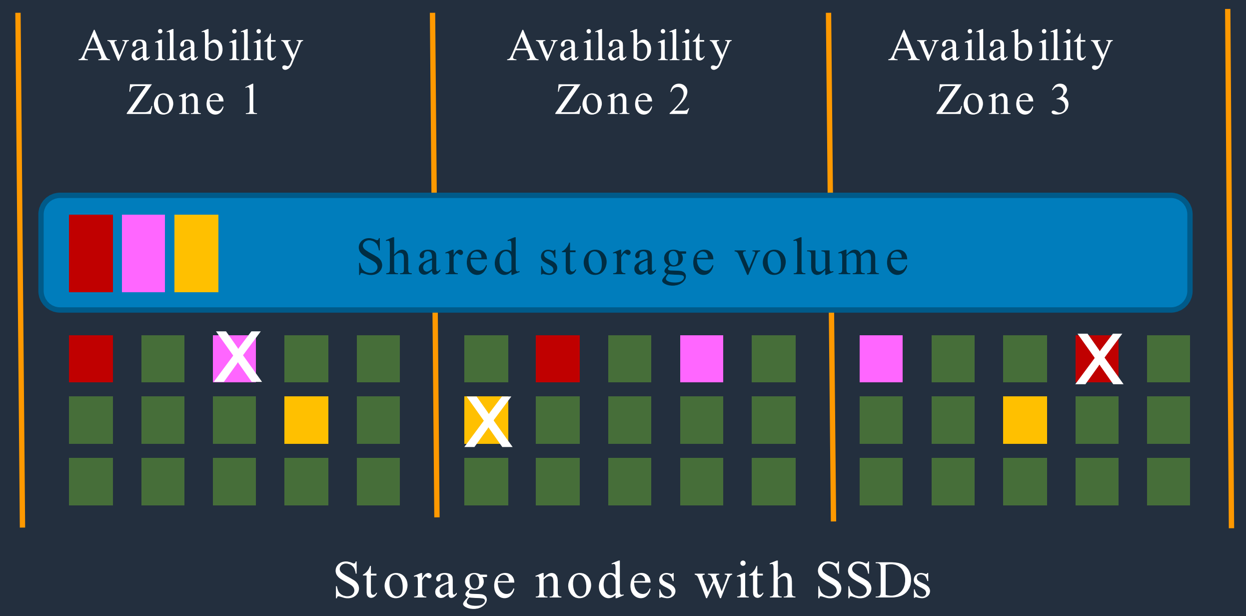
Uncorrelated and independent failures

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

<u>**The solution is replication**</u>

One common straw man:

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

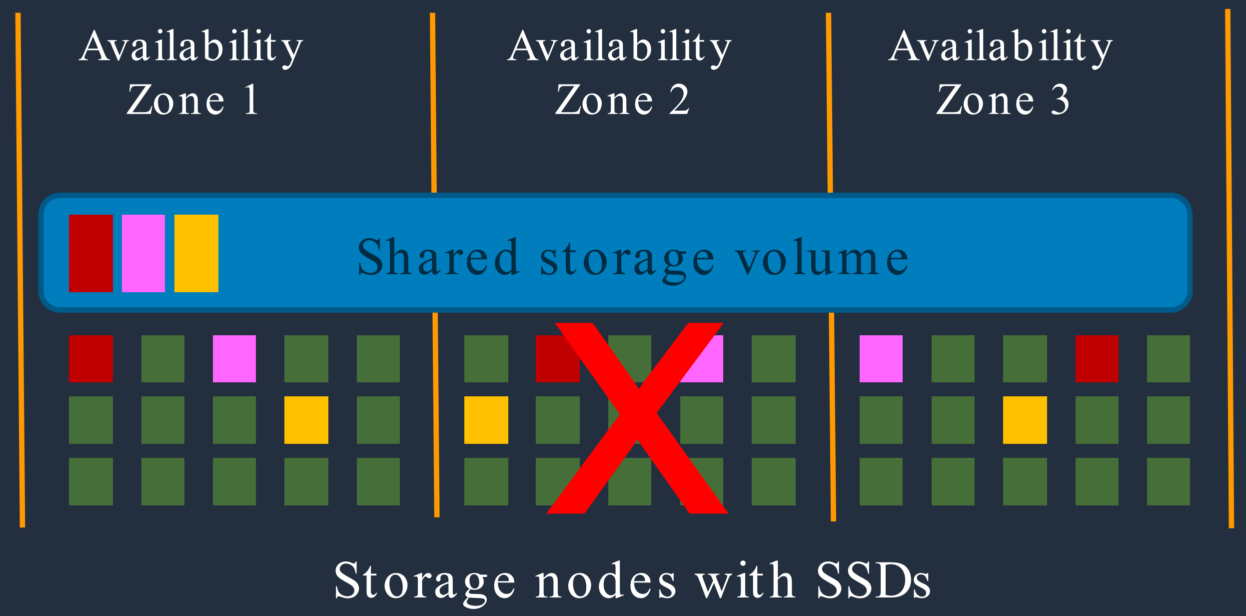


What about AZ failure?

⇒ Still have 2/3 copies

⇒ Can establish quorum

⇒ No data loss



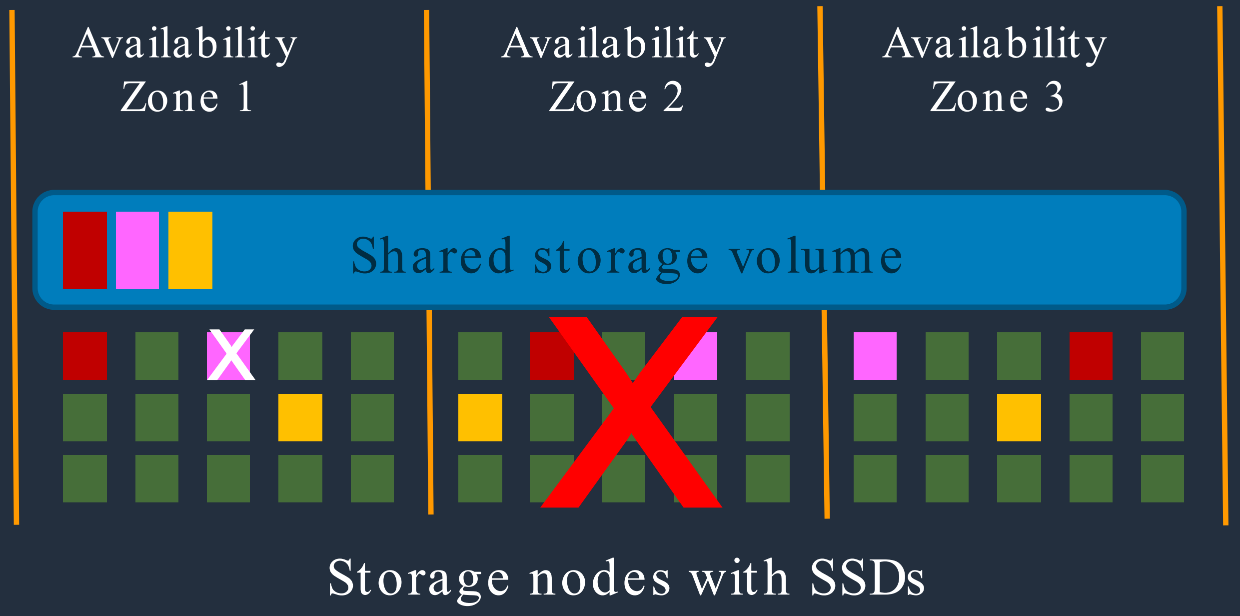
What about AZ + 1 failures?

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

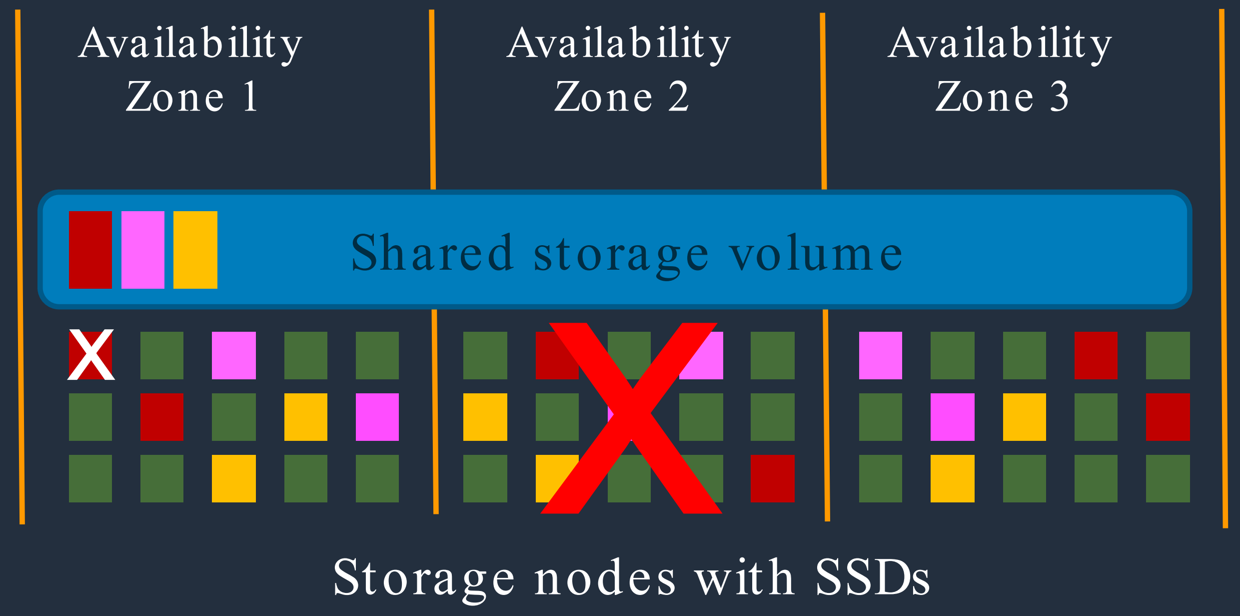
What if there is an AZ + 1 failure ?

⇒ 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

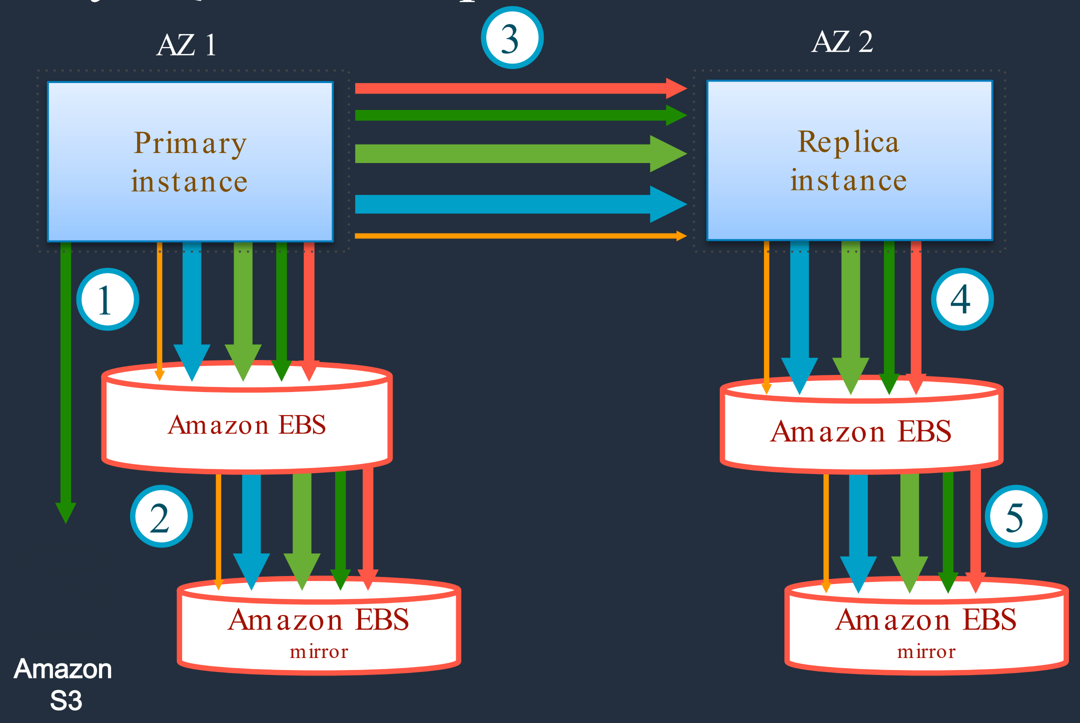
Epoch 2: Node F is in a suspect state; second quorum group is formed with node G; both quorums are active

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

Performance results

Aurora I/O profile

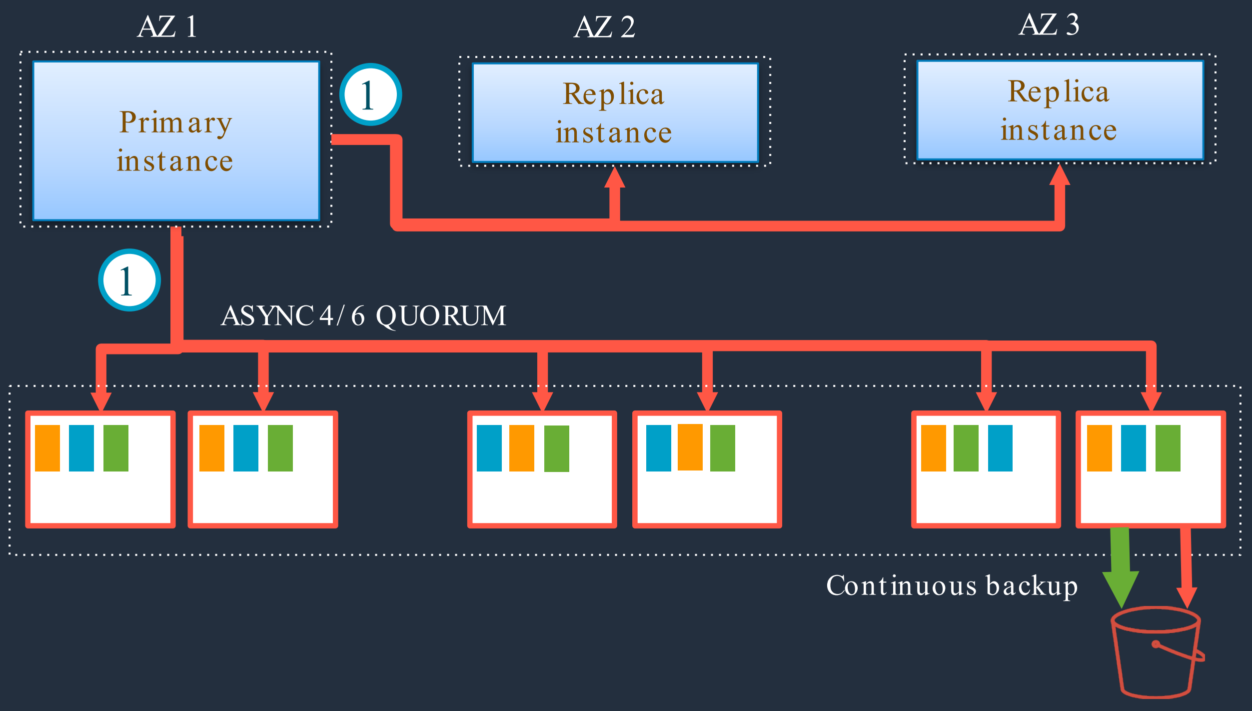
MySQL with replica



MySQL I/O profile for 30-min Sysbench run

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

Aurora



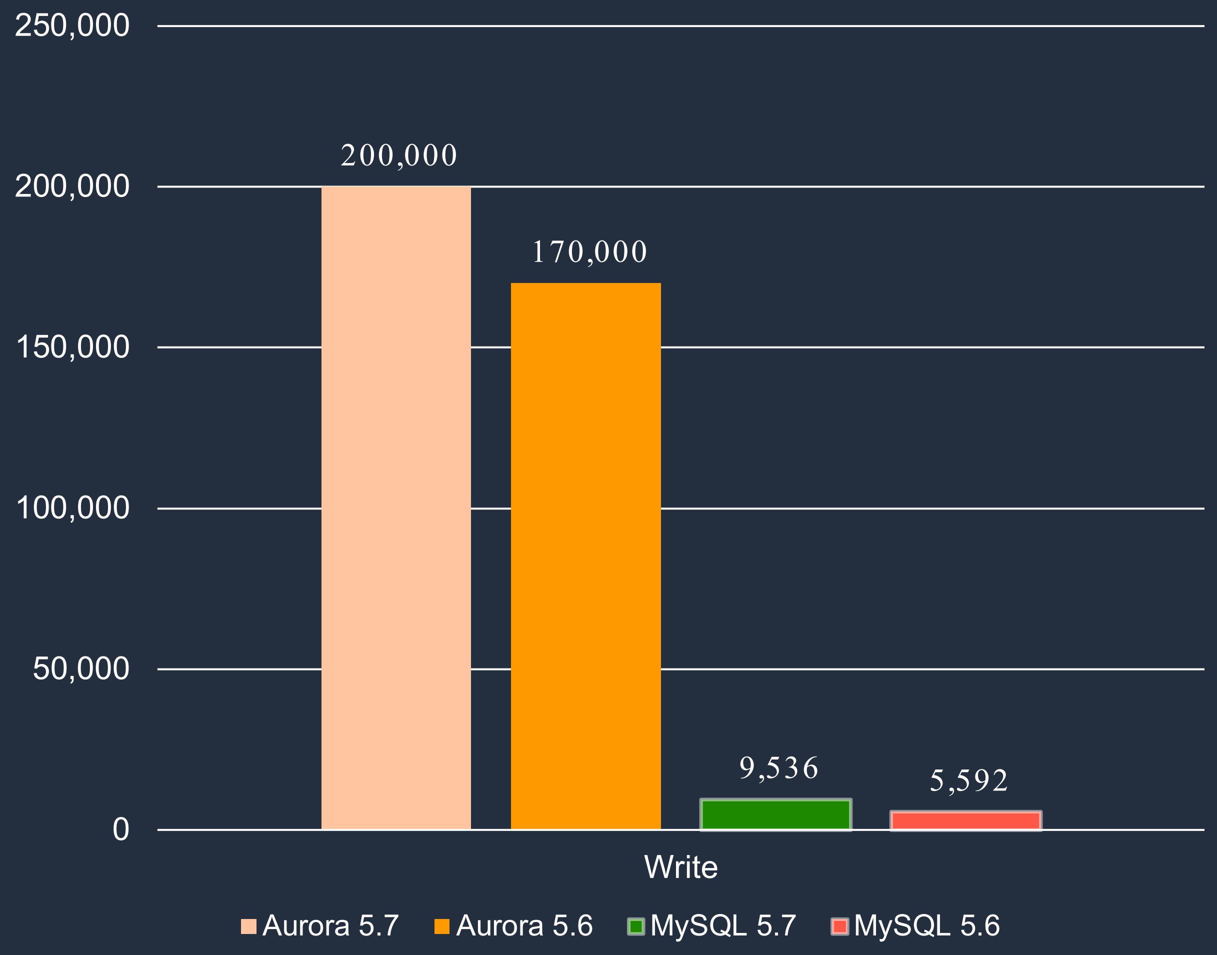
Aurora IO profile for 30-min Sysbench run

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

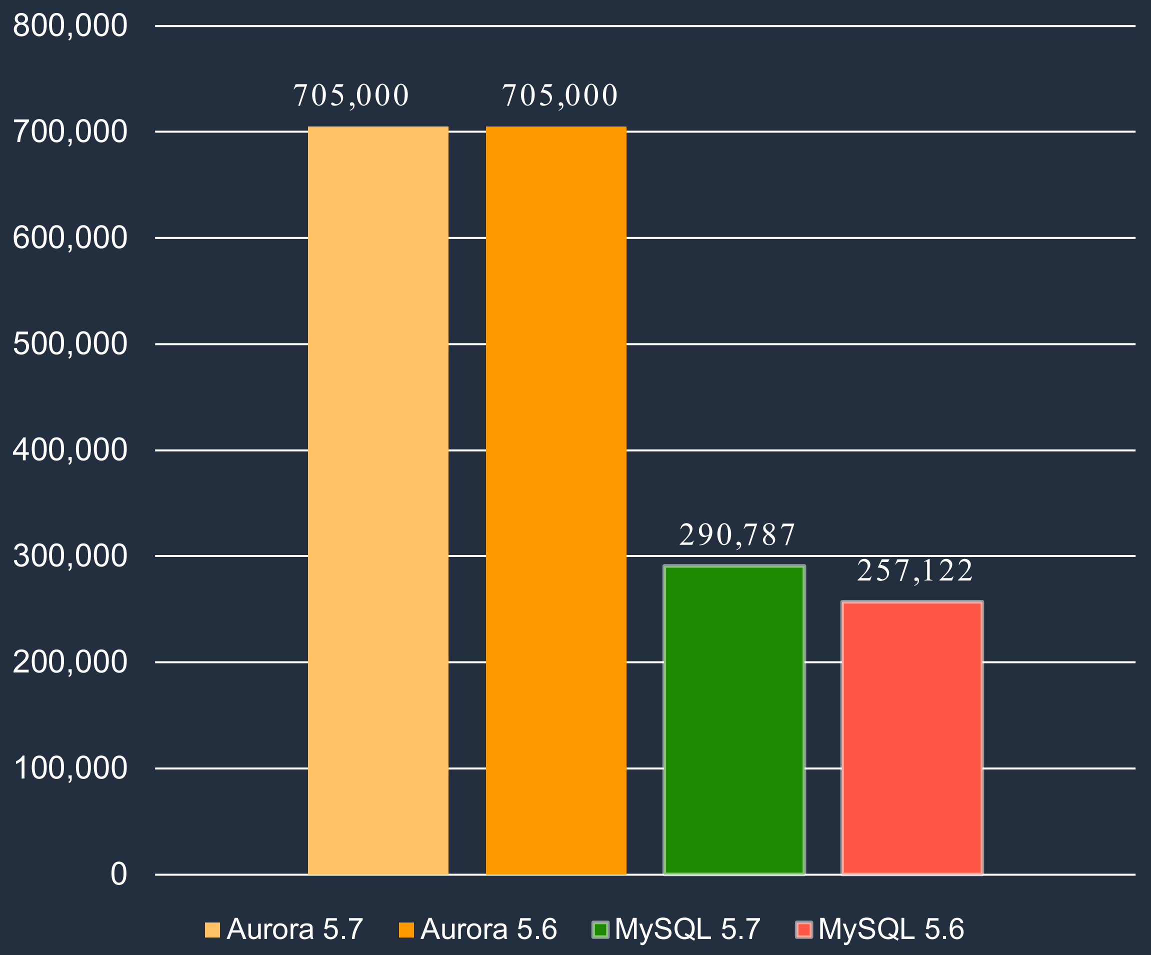
Log — Binlog — Data — Double-write — Frm files

Write and read throughput

Aurora is up to 5× faster than standard MySQL databases



Write throughput



Read throughput

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

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 Brahmadesam