



CS 839 Next Generation Data Systems

Lecture 2: Cloud-native OLTP

Tianyu Li
litianyu@cs.wisc.edu

Administrivia

- If you are on the wait list, you should have received an email
 - Come talk to me after class
- If you realize this class is not for you, please make a decision early to give others a chance!

Administrivia

- First reading check-in on Thursday (9/10) 11:05 – 11:15
- Expect pen & paper short answer about a couple of the papers you read, no electronics
- Remember, you are encouraged to use AI to help get through the readings
 - Assigned readings intentionally more than usual
 - You should experiment with different approaches and we will ask you about your experience as the semester goes on
 - However, you are responsible for understanding the material

Administrivia

- Team assignment due 9/15
 - Submit on canvas, will be posted soon
- Teams of up to 3 (3 preferred)
 - If you want to work alone, check with me first
- If you can't (or don't want to?) find teammates, submit areas of interest and instructor will assign you

Recap: “Next Generation” Data Systems

- Same question: how do we efficiently and economically support [important data processing workload] using [available computing capabilities]?
- The variables change every few years, but the question and the techniques at our disposal often do not
- Learning Goal: understand key new capabilities, emerging workloads, and past systems to synthesize new ones

Today: Cloud-native OLTP

- Around 10 years in the making and becoming ubiquitous
- Important data processing workload: Online-Transaction Processing (OLTP)
- Available Computing Capabilities: Cloud Computing

What is OLTP?

- Representative Workload: Enterprise Record Management, E-Commerce, Payment Systems, etc.
- Characteristics:
 - Short transactions with high concurrency
 - Often known/highly-predictable read/write sets
 - Often mostly point queries
 - Scale is often limited (bound by real world problems it models)

Example: TPC-C

- Models the workload of a warehouse supply company
 - Tables: Warehouse, District, Customer, Orders, Items, etc.
 - Tunable scale around # Warehouse
 - Workload often skewed
- Workload consists of a few types of short transactions
 - NewOrder (45%): RMW with heavy sequential inserts
 - Payment (43%): RMW with mostly point updates
 - OrderStatus (4%): Point Reads
 - Delivery (4%): (Small) Batch Updates
 - StockLevel (4%): (Small) Batch Reads

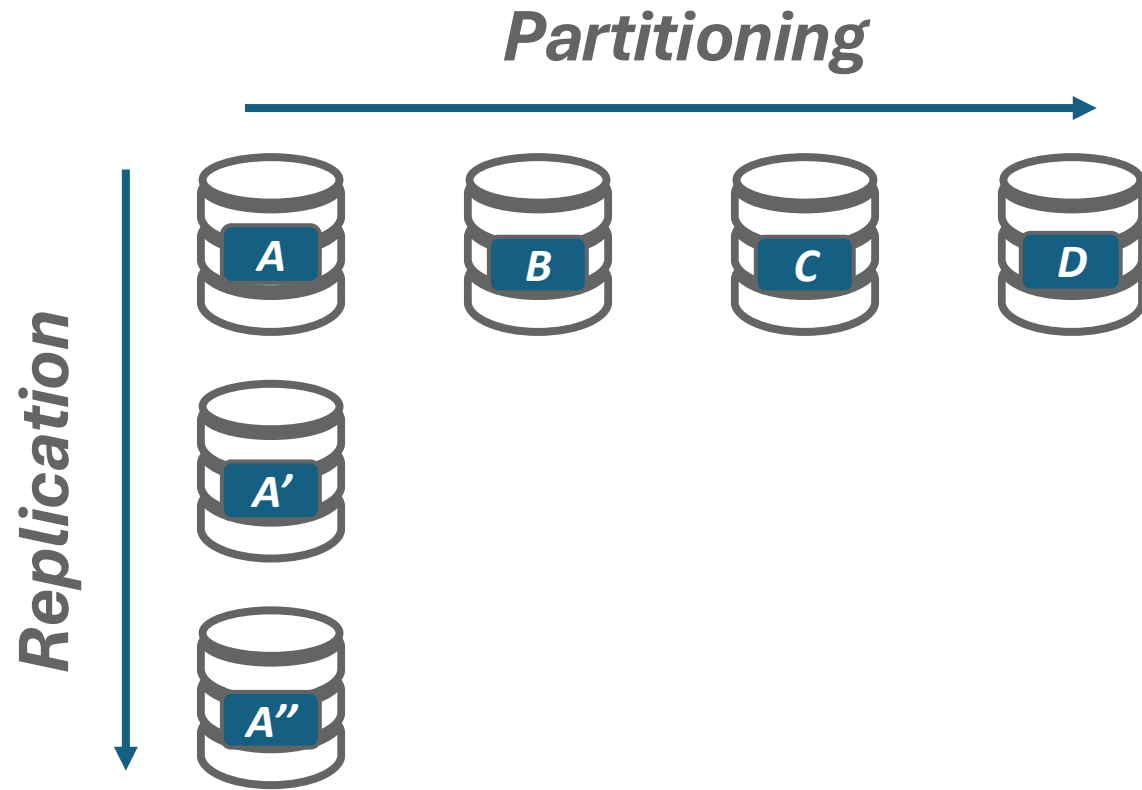
Obvious Solution



Why not just Postgres?

- Most traditional database systems support OLTP very well
 - And many workloads fit onto one (potentially big) node
- Problem – going distributed
 - For availability / disaster recovery
 - For scale
 - For low-latency
 - Sometimes for regulatory purposes

Distributed DBMS



Problems

- Traditional Architecture: Shared-nothing
 - Each node a standalone machine that talks over the network
- Traditional Deployment Model: on-premise databases
 - Host and maintain your own hardware
 - Purchase software licenses / deploy open-source
 - Hire DBAs to deploy and manage daily workloads
- Problem: a lot of technical work
 - Ideally, customers want a lot of this complexity *managed*

Enter the Cloud

- The cloud started as a simple idea: Amazon/Google/Microsoft builds data centers and manage the hardware, you rent capacity from them
 - Benefit from economy of scale and multi-tenancy
- Initially, the cloud sells virtual machines that look and smell like physical servers connected through a network
- Now: often sells managed software service

First: Easy Solutions

- Take DB software, and instead of deploying on-prem, deploy them onto VMs
 - Reasonable solution that is often used
- Problems:
 - Software complexity still exists
 - Performance and fault-tolerance challenges
 - Misses on many opportunities provided by the cloud
- Goal: build managed DB-as-a-Service systems that take advantage of cloud SaaS

Recap: Disk-based DBMS

- Data organized in pages stored on disk
- For performance, disk pages are cached in-memory and only periodically written back (i.e., NO-FORCE)
- Also for performance, dirty disk pages containing uncommitted writes are flushed when necessary (i.e. STEAL)
- To achieve ACID, use a write-ahead log to persist changes without having to flush pages

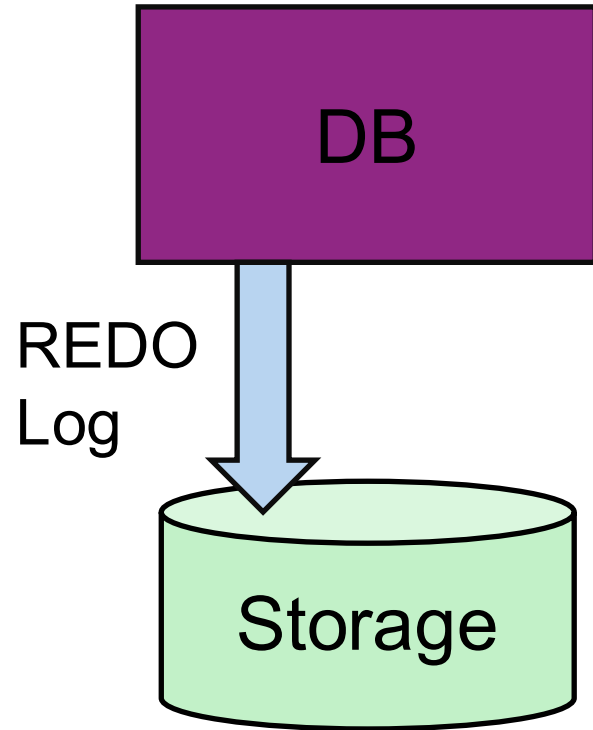
Cloud-native DBMS

- Goal: replace disks with commodity cloud storage
- Challenge: Cloud storage (e.g. EBS) does not behave like normal storage disks
 - On the one hand, it has massive aggregate throughput
 - On the other hand, it is much higher latency and potentially rate limited for single user
- For OLTP workload – must mask latency of naïve migration

Amazon Aurora

- Base System: MySQL or Postgres
- Aurora: modifies just the storage layer of the base DB
 - Buffer Pool reads from cloud storage
 - Buffer Pool discards pages on flush
 - WAL are replicated in cloud storage
 - The storage layer replays the log into data pages

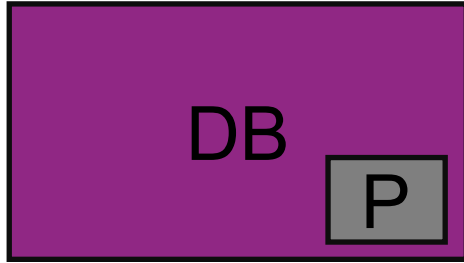
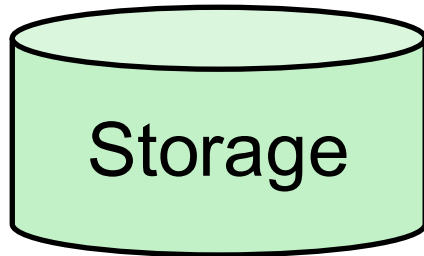
Forward Processing – Write and Commit



Write: flush REDO log record to storage (no UNDO)

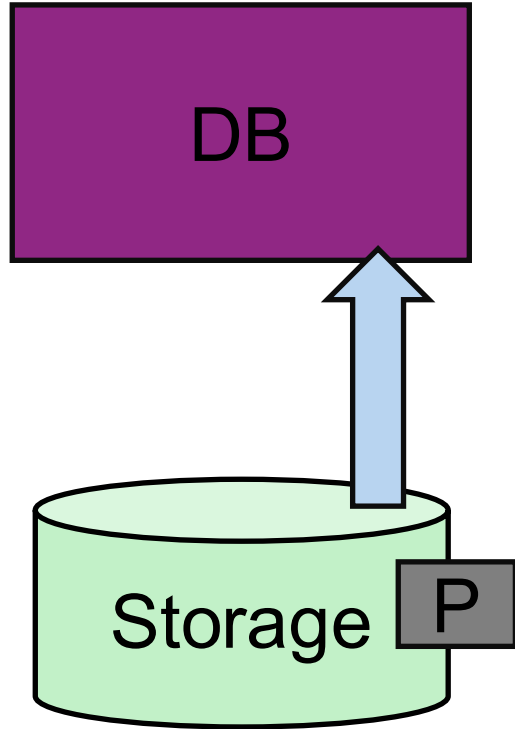
Commit: flush COMMIT log to storage

Forward Processing – Read



Buffer hit: read from main memory of the DB server

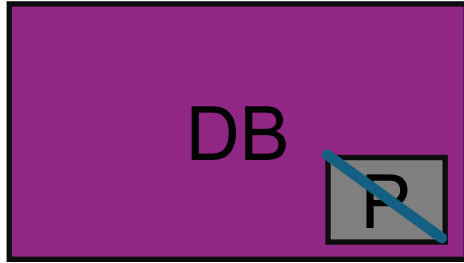
Forward Processing – Read



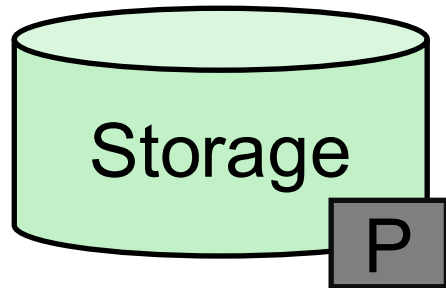
Buffer hit: read from main memory of the DB server

Buffer miss: read page from storage

Forward Processing – Eviction



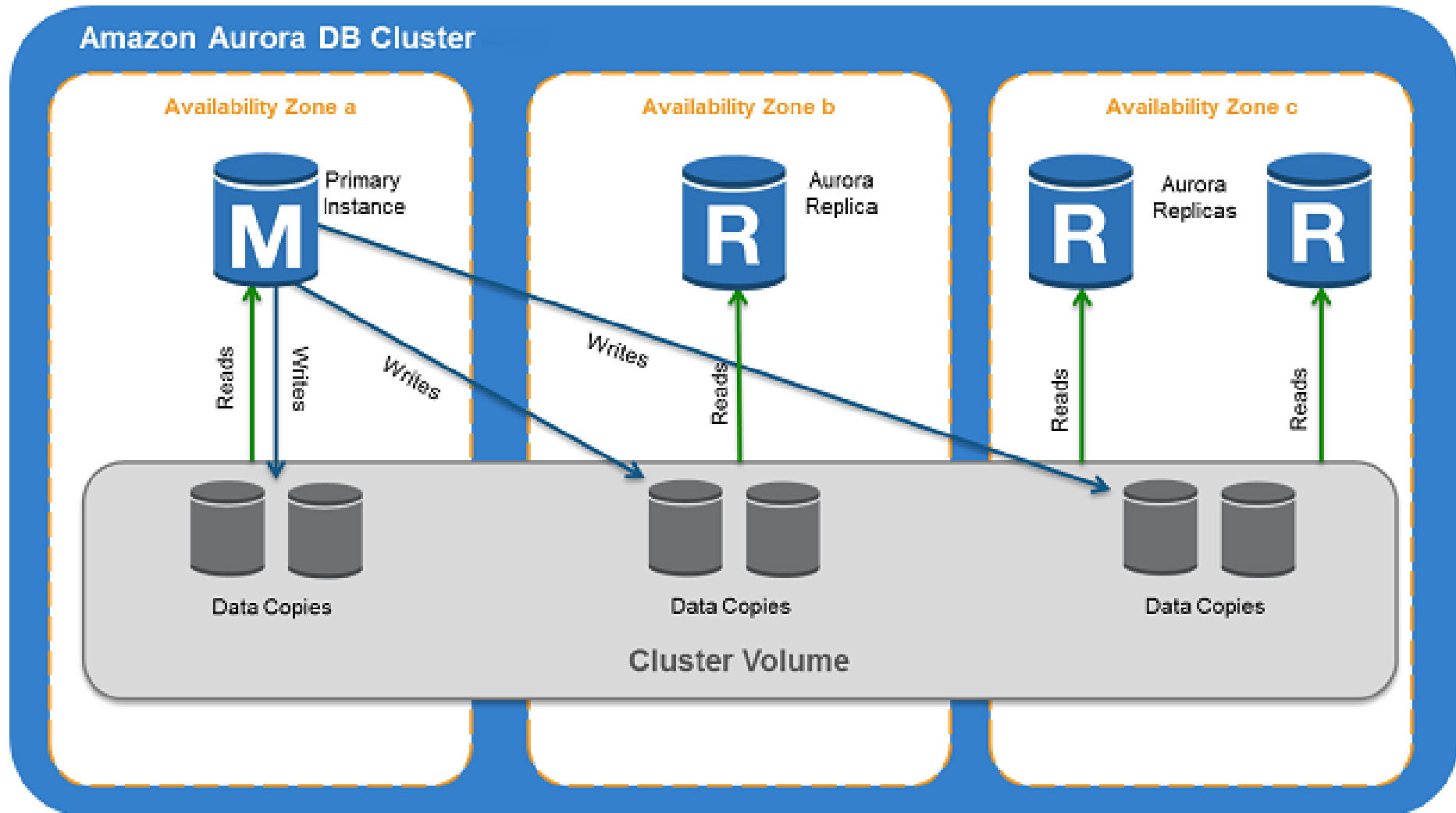
Aurora:
discard dirty
page



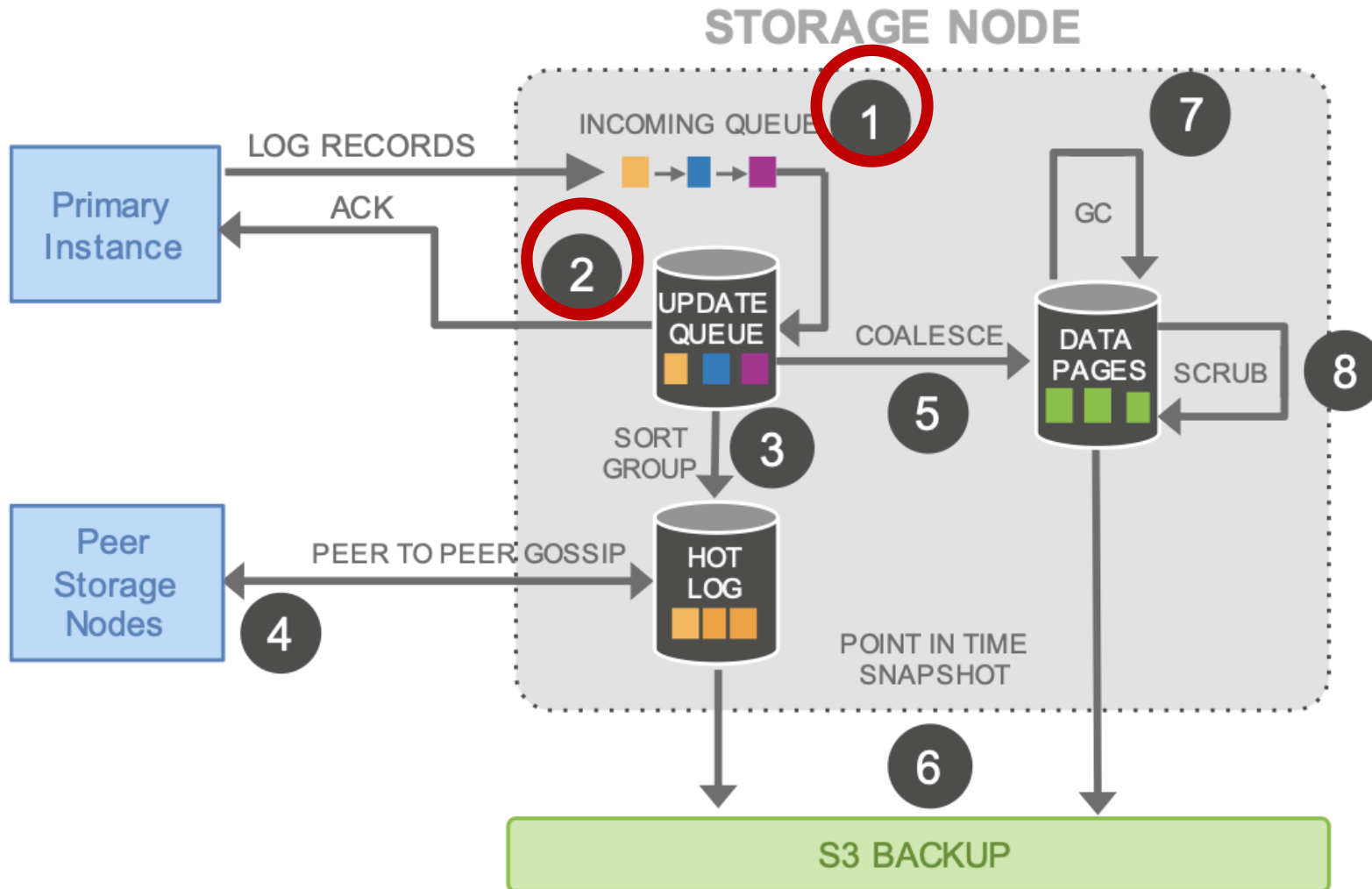
Dirty eviction: discard dirty page (no write back to storage)

- The page in storage will be updated through replaying the REDO log

Aurora – Single Master

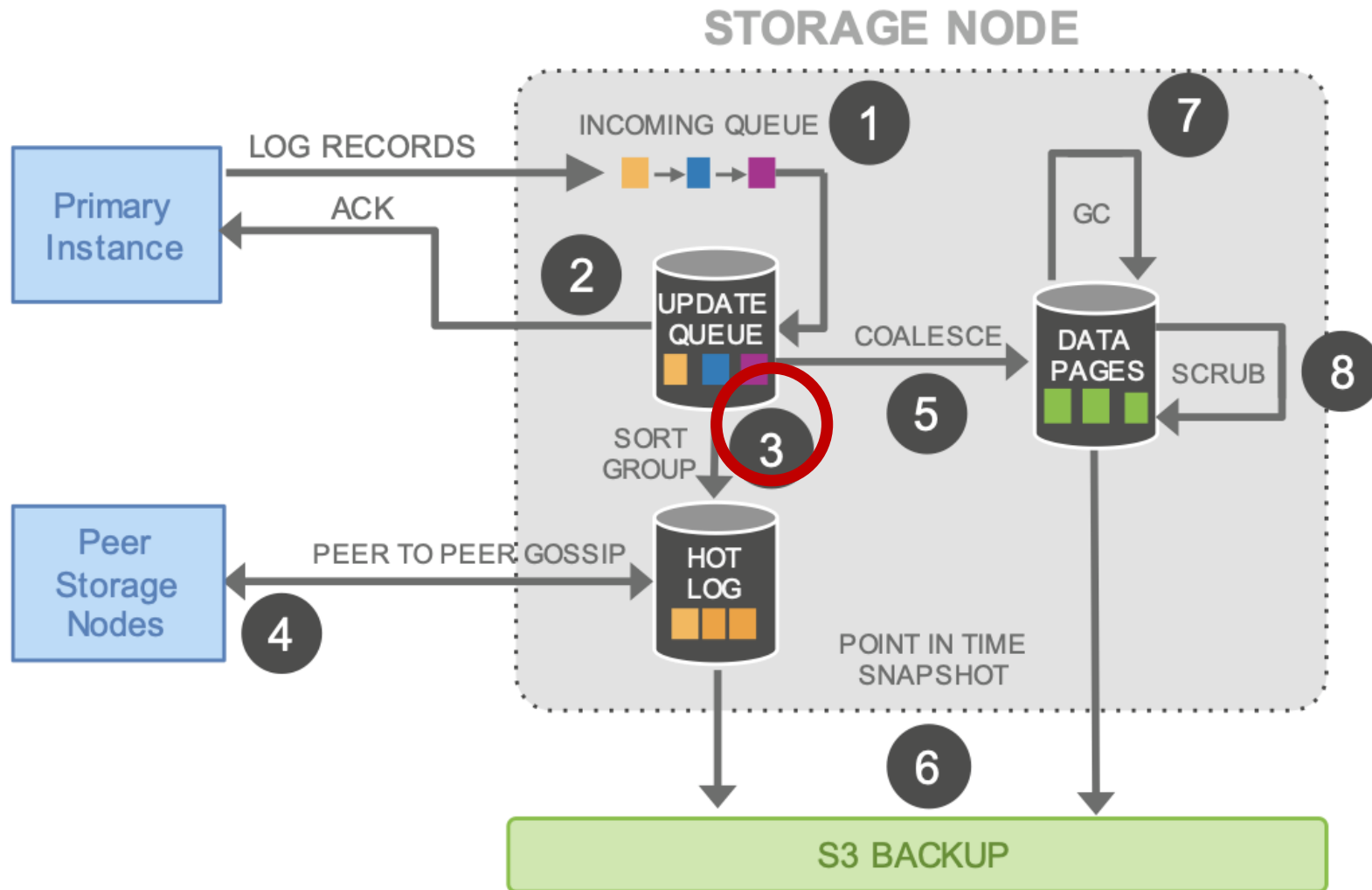


Storage Node



Only Steps 1 & 2 are in the foreground path

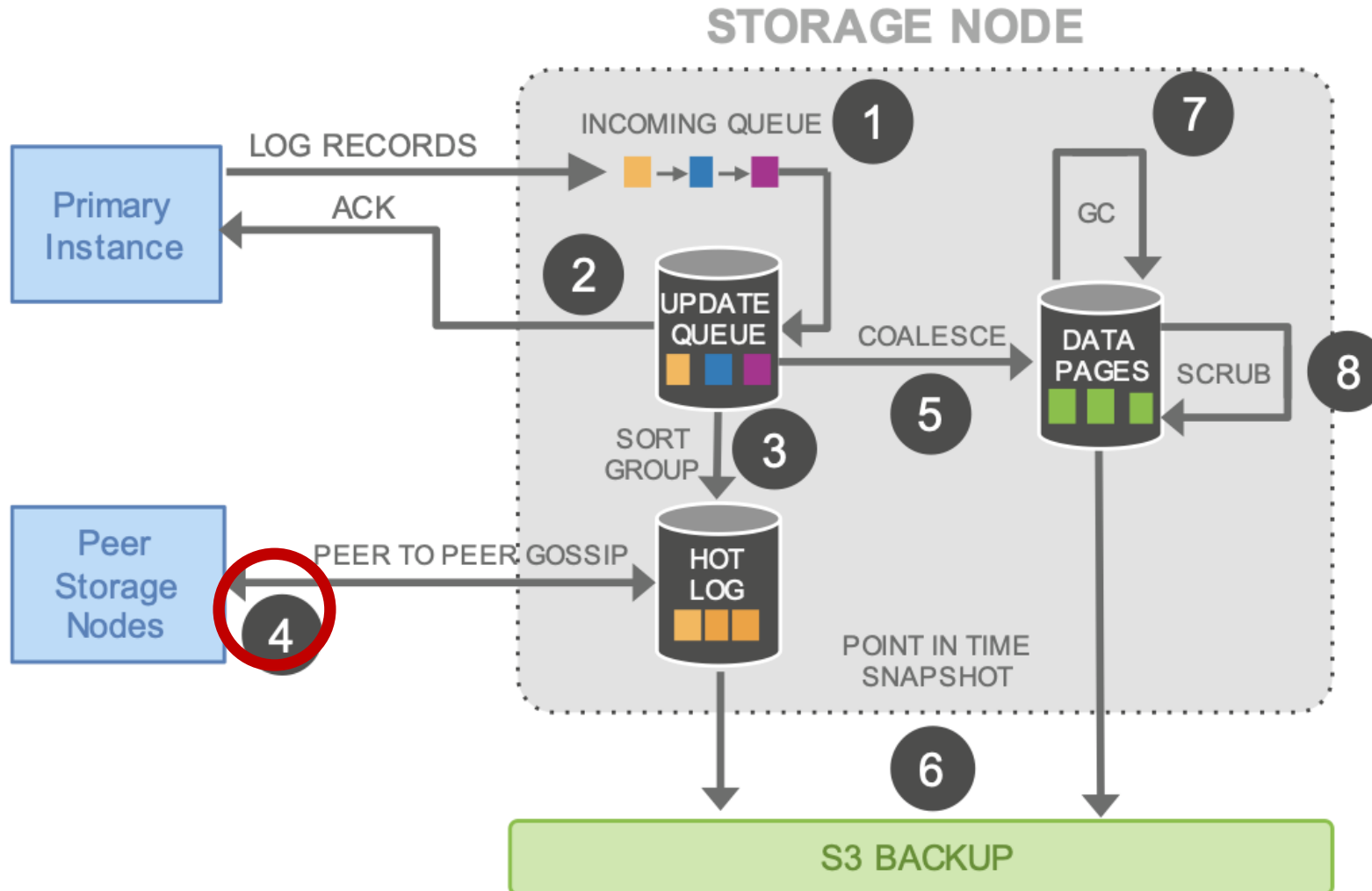
Storage Node



Identify gaps in the log



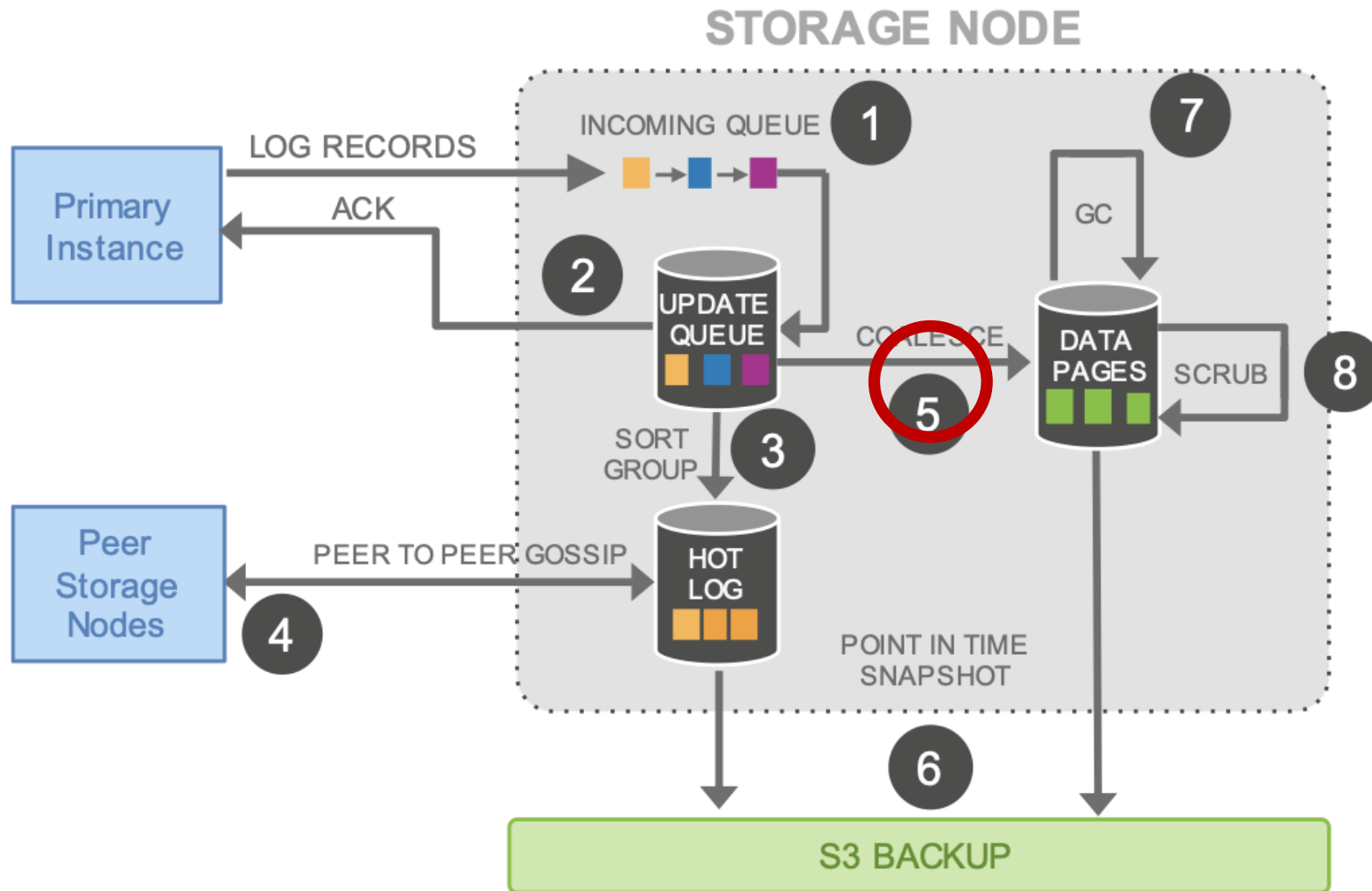
Storage Node



Gossip with peers to fill gaps

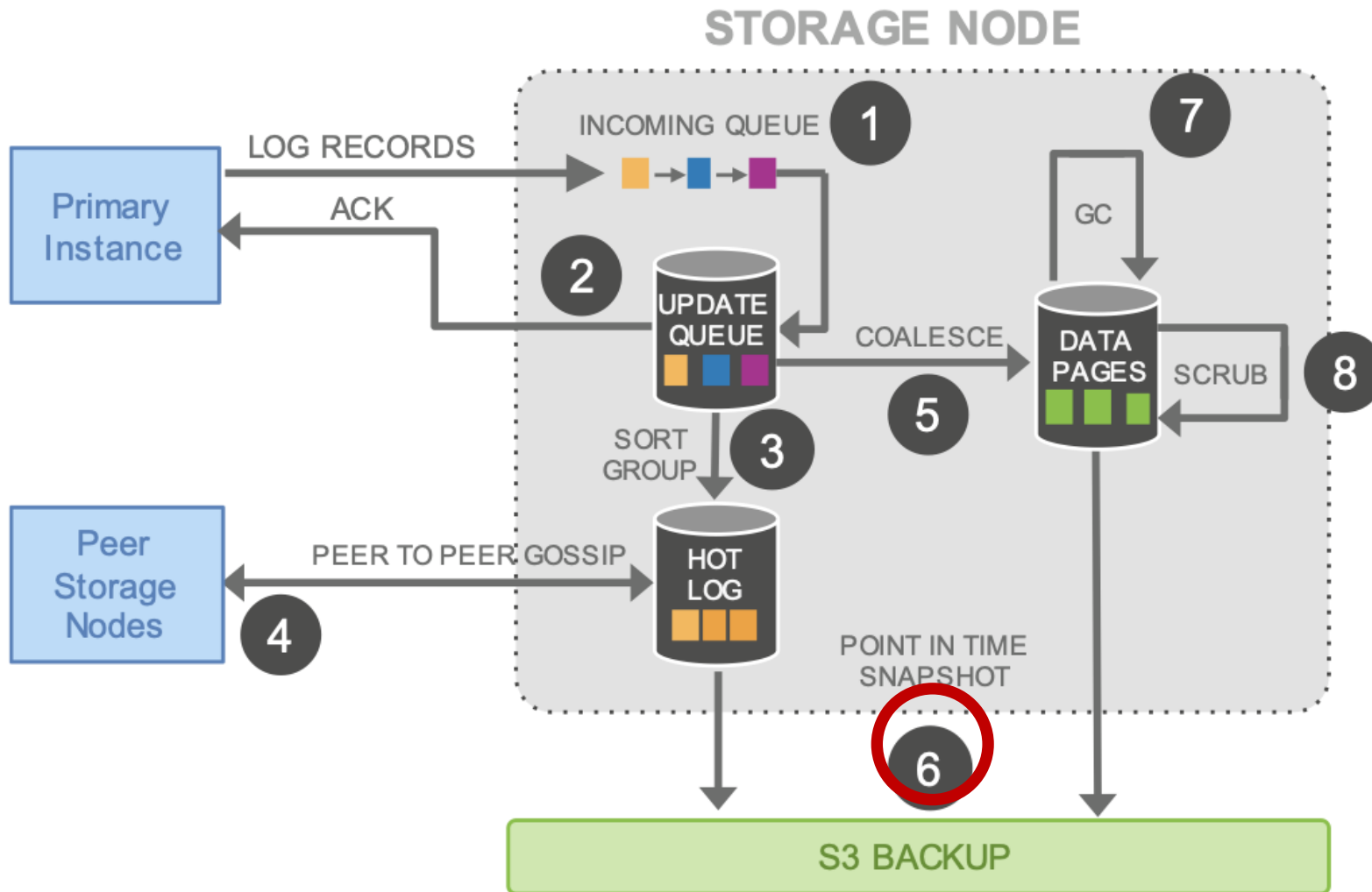


Storage Node



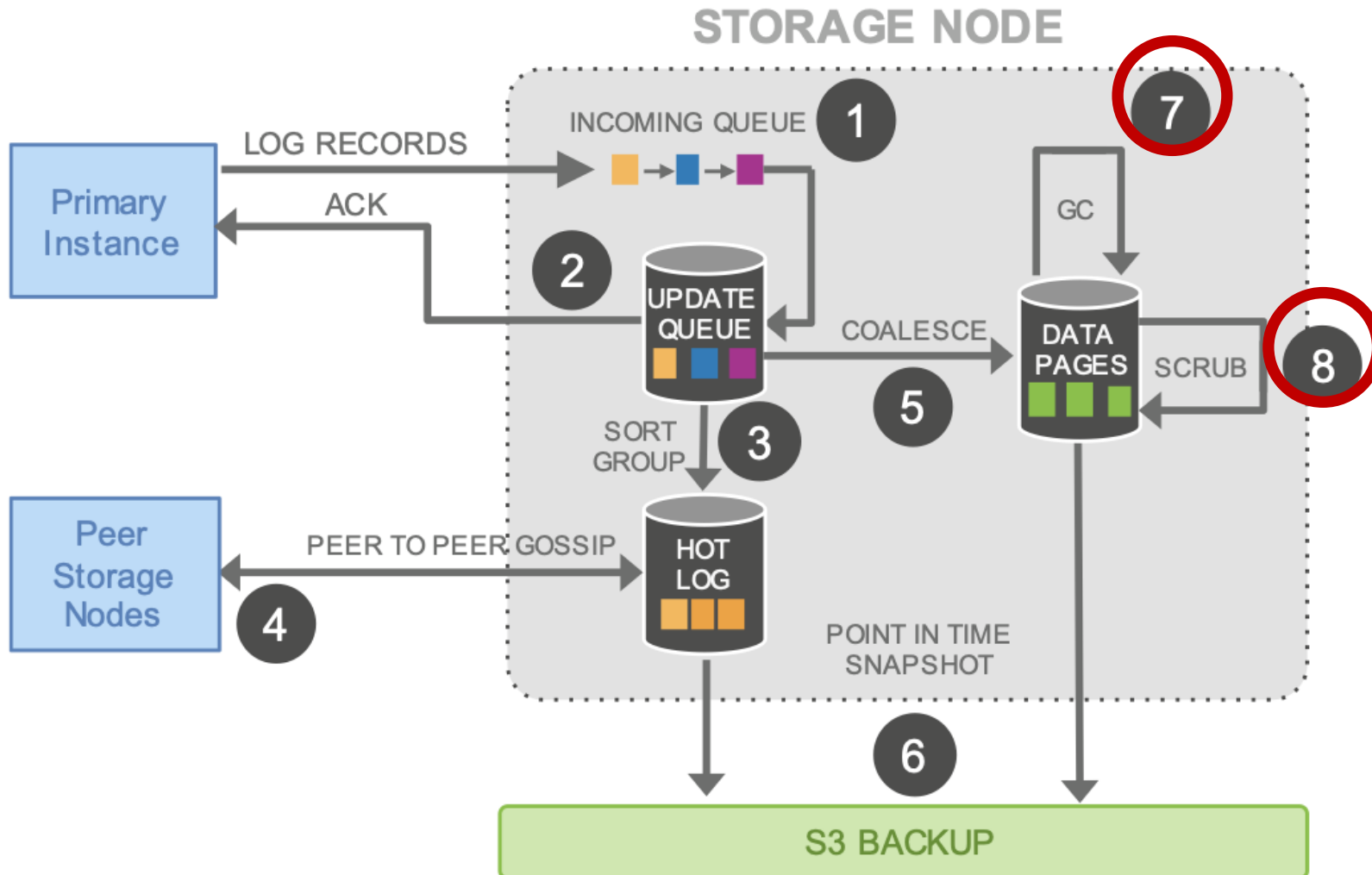
Coalesce log records
into data pages

Storage Node



Periodically stage log
and pages to S3

Storage Node



Periodically garbage collect old versions and periodically validate CRC code on pages

* Cyclic redundancy check (CRC) is an error-detecting code

Recovery

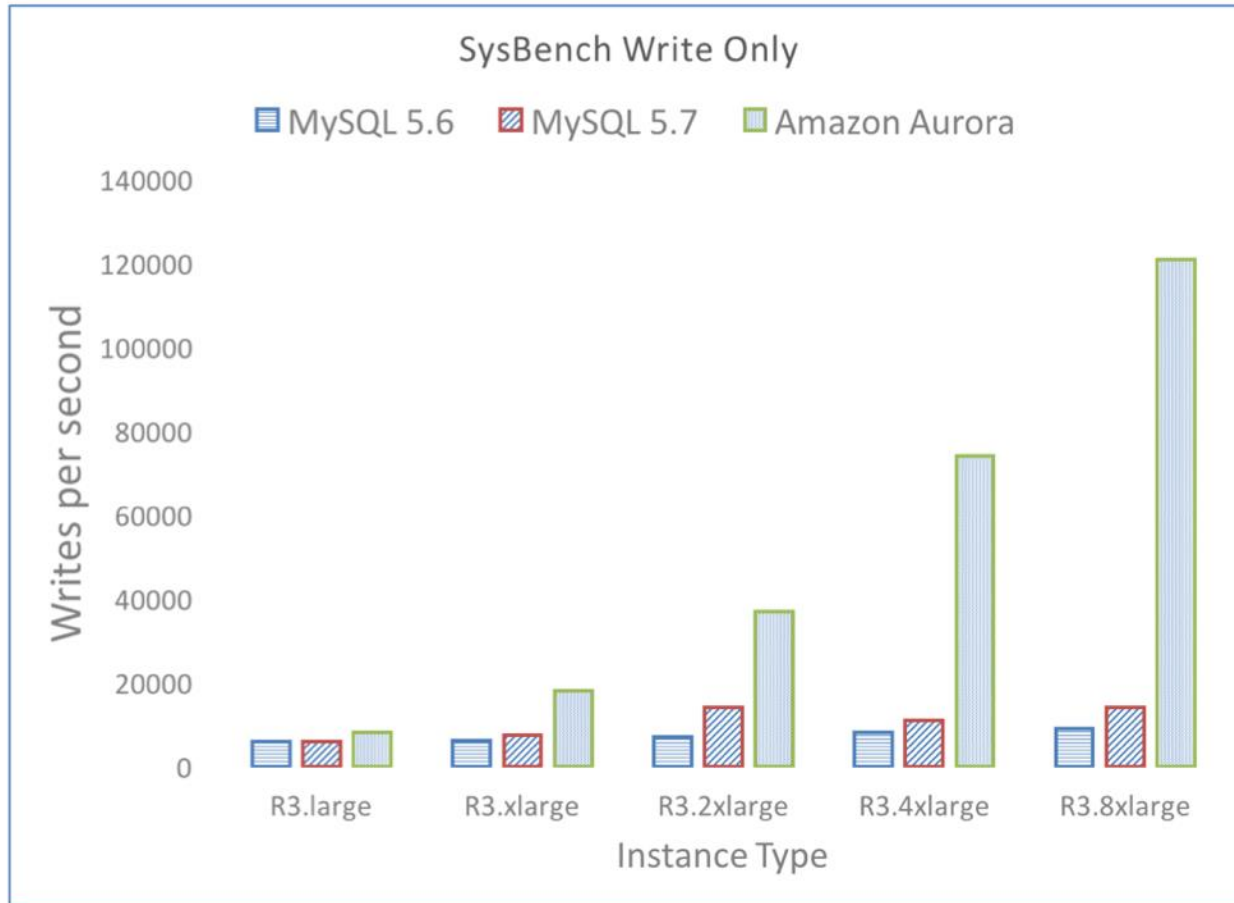
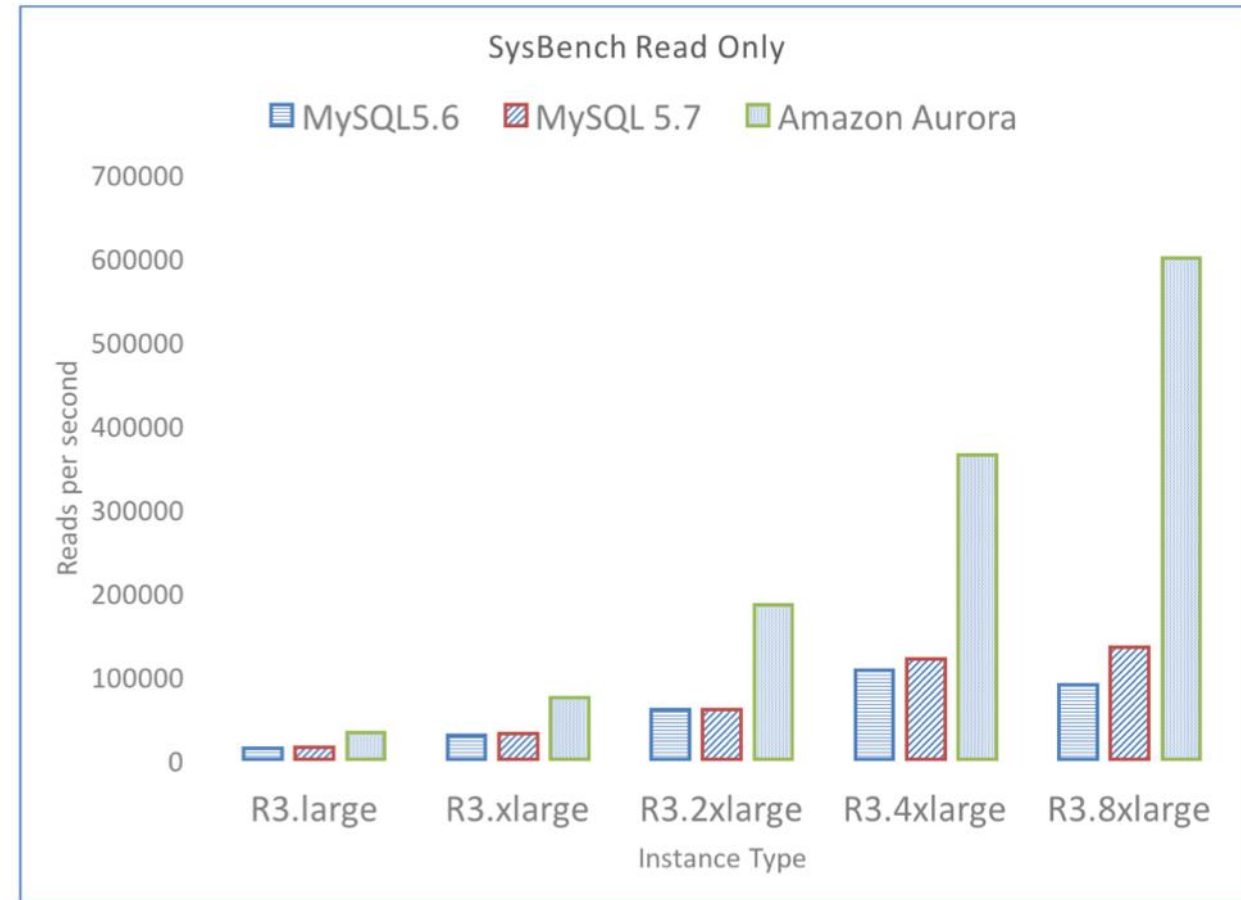
For Compute Node:

- Issue quorum read for the log tail, discard everything after
- Database is immediately ready for operations with some background cleanup

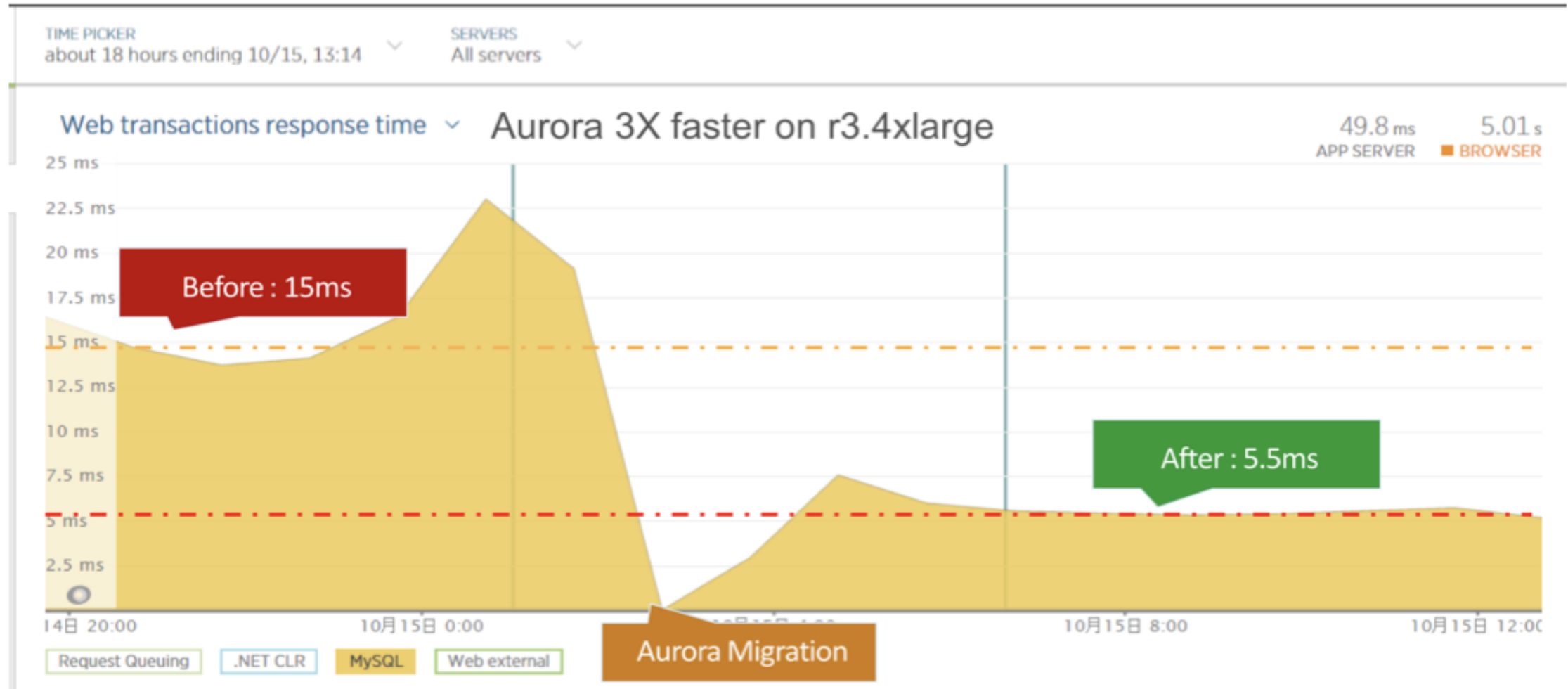
For Storage Node:

- Leader Election etc. required

Evaluation – Aurora vs. MySQL



Evaluation – Real Customer Workloads



Discussion

- Disaggregation is often thought of as trading off performance with higher-level abstraction – why did Aurora perform better with disaggregation?
- How is Socrates different from Aurora, and what do you think that means for performance and engineering complexity?
- What problem does Aurora DSQL solve compared to vanilla Aurora? What does it sacrifice?
- Are there interesting new capabilities that the Aurora architecture enables that is not previously possible?

Next Lecture

- Cloud-Native OLAP & Lakehouses
- Remember to arrive on time for reading check-in!