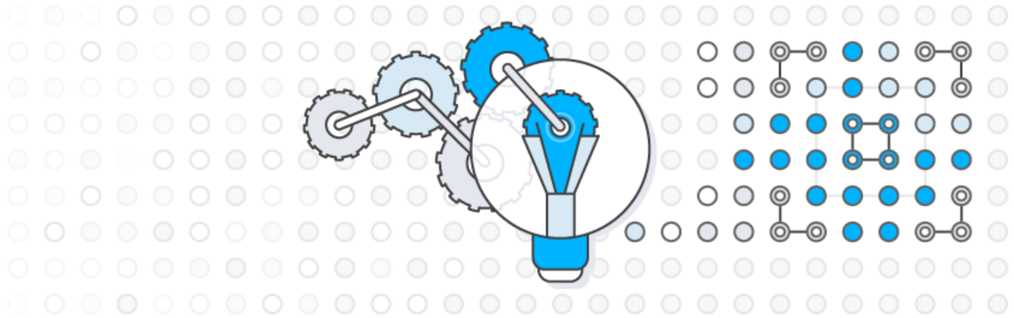


Amazon Aurora & Redshift Spectrum Design Considerations for Cloud-Native Database Management

Anurag Gupta, Sailesh Krishnamurthy, Ippokratis Pandis

Amazon Web Services

HPTS 2017

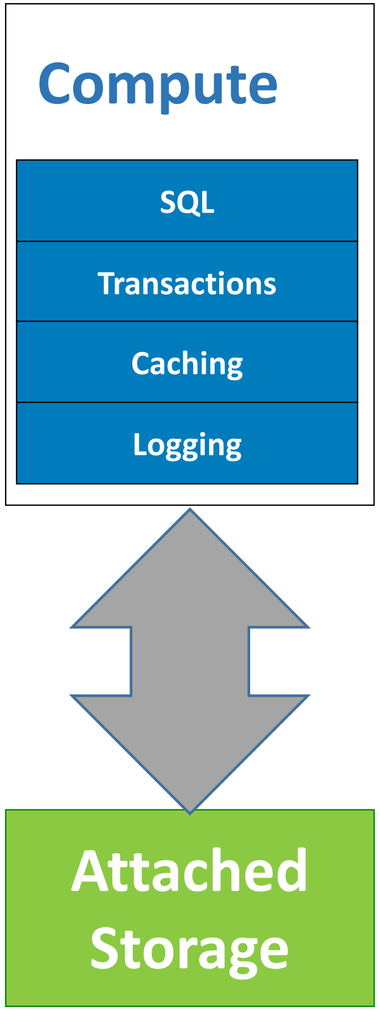
Traditional Database Architecture

Databases are all about I/O

Design principles for 40+ years

Increase I/O bandwidth

Decrease number of I/Os



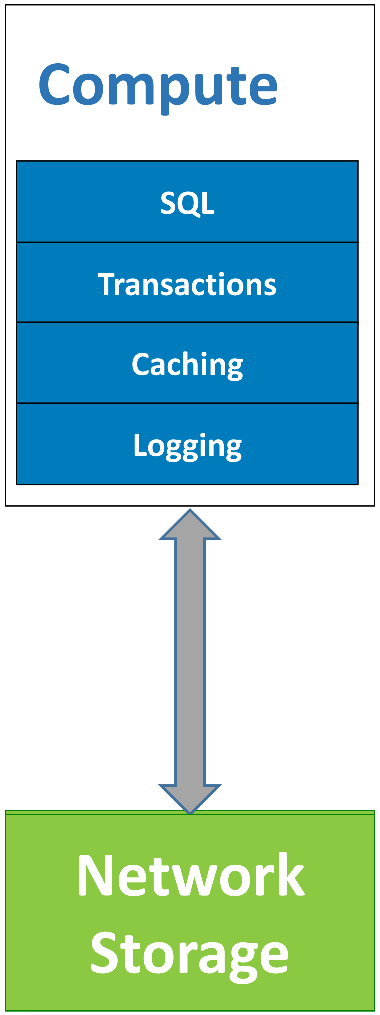
Databases in the Cloud

Compute & Storage have different lifetimes

- Instances fail and may be replaced
- Instances are shut down
- Instances are scaled up/down
- Instances are added to a cluster to scale out

Compute and Storage are best **decoupled**

- For scalability, availability, durability



Databases are data-intensive computing

Compute cycles per byte of data (C:D) is low

I/O operations per byte of data (I:D) is low

But, with scale-out compute, scale-out storage,
data explosion, edge DCs,
network operations per byte of data (N:D) is growing.

The I/O bottleneck has moved to the network

Let’s examine how we approach this for a modern transactional database

Amazon Aurora: A Cloud-Native Database

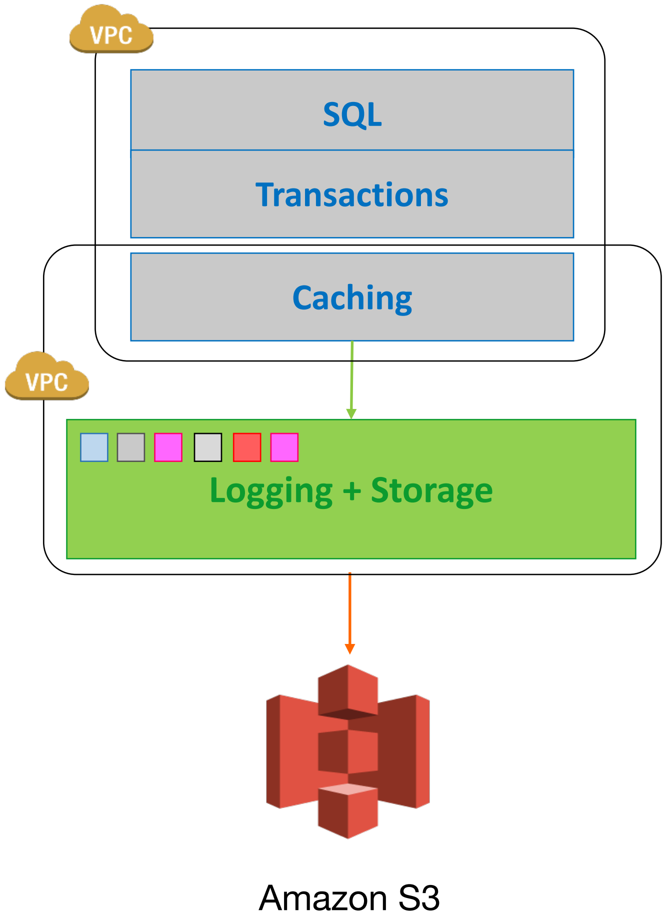
Move redo to multi-tenant storage service

Address the network bottleneck

Improves durability, availability, and jitter

... which are really the same

... over different time scales



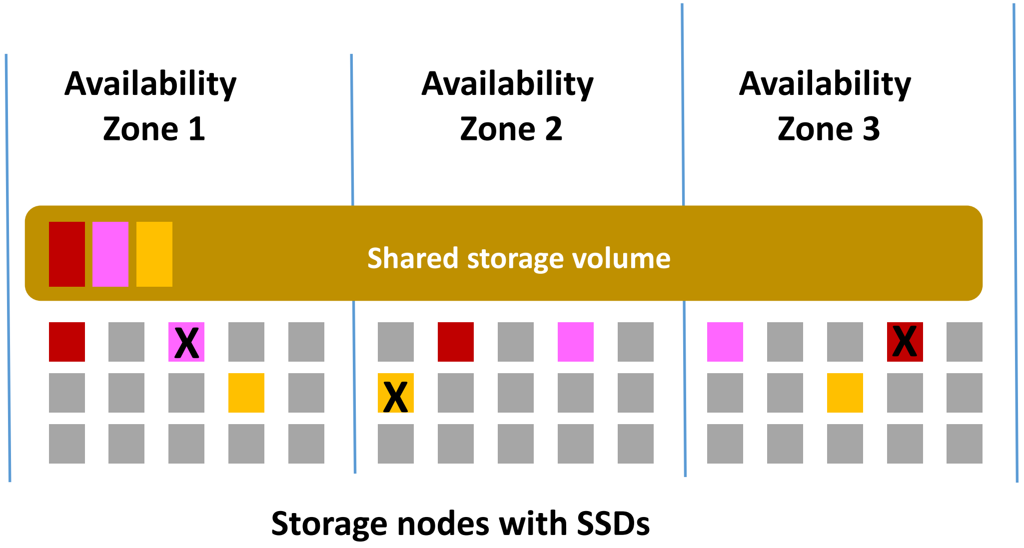
Verbitski et al, Amazon Aurora: Design Considerations for High Throughput Cloud Native Relational Databases, **SIGMOD 2017**

Durability at Scale

Uncorrelated and Independent Failures

At scale, continuous independent failures

- Constant background radiation
- Nodes, disks, switches all fail



Replicate storage for resilience

One common strawman:

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

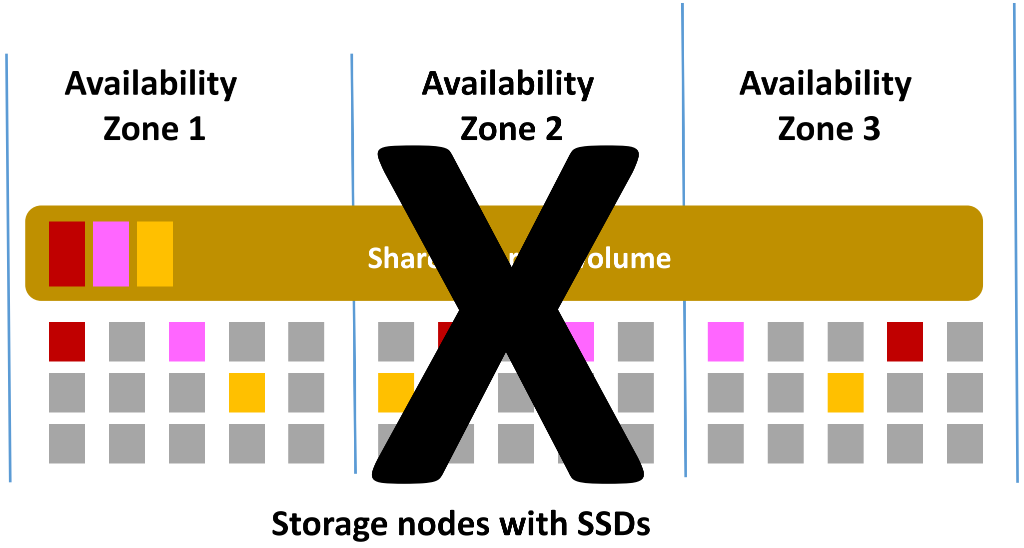
What if an AZ is lost ?

Boils down to losing 1 node

⇒ Still have 2/3 nodes

⇒ Can establish quorum

⇒ No data loss ☺



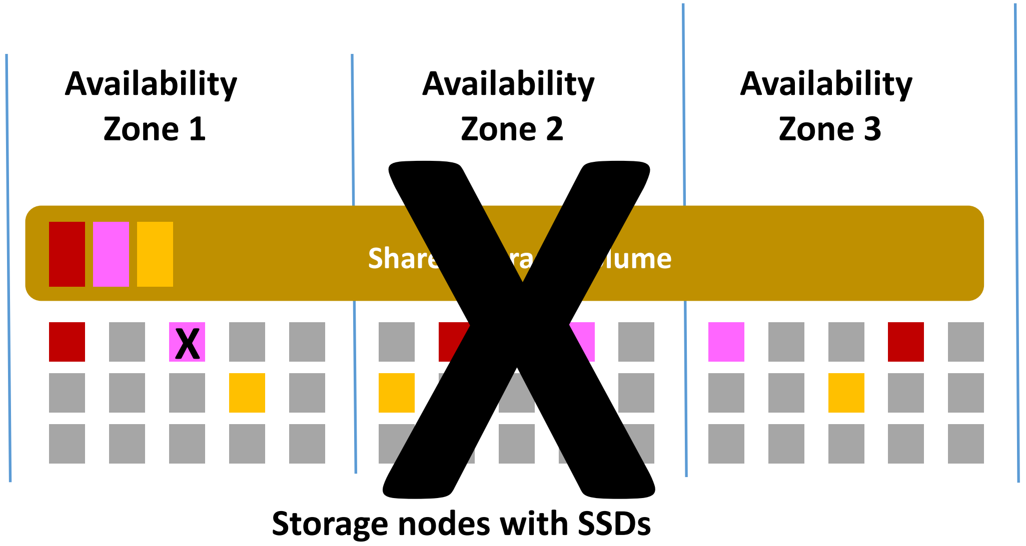
What if another node is *also* lost ?

AZ+1 model: correlated failures!

⇒ Lose 2/3 nodes

⇒ Lose quorum

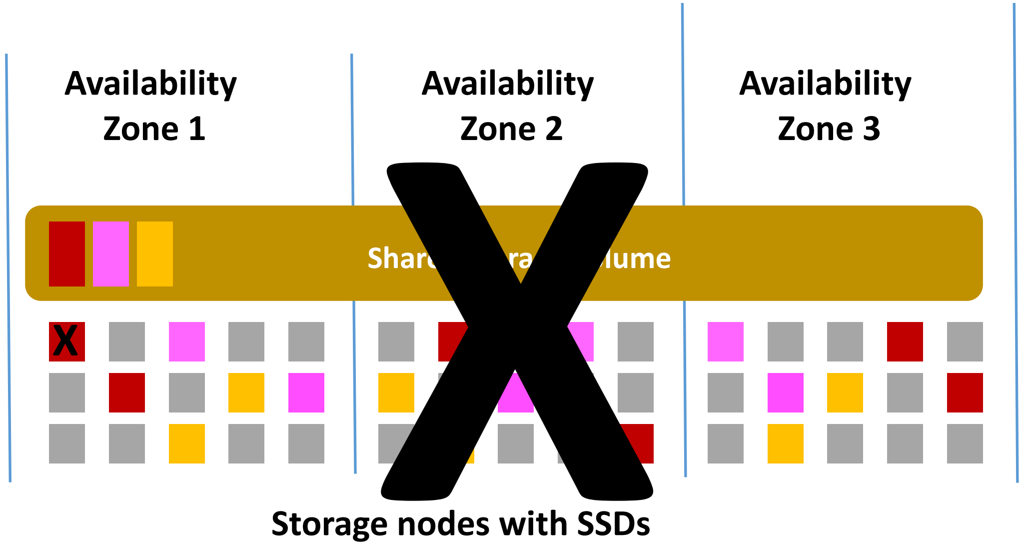
⇒ Lose data ☹



Aurora tolerates AZ+1 failures

Replicate 6-ways with 2 copies per AZ

- Write quorum of 4/6
- Read quorum of 3/6 (only for repair)



What if there is an AZ failure ?

⇒ Still have 4/6 nodes

⇒ Maintain write availability

What if there is an AZ+1 failure ?

⇒ Still have 3 nodes (read/repair quorum)

⇒ No data loss ☺

⇒ Rebuild failed node by copying from one of other 3

⇒ Recover write availability ☺ ☺

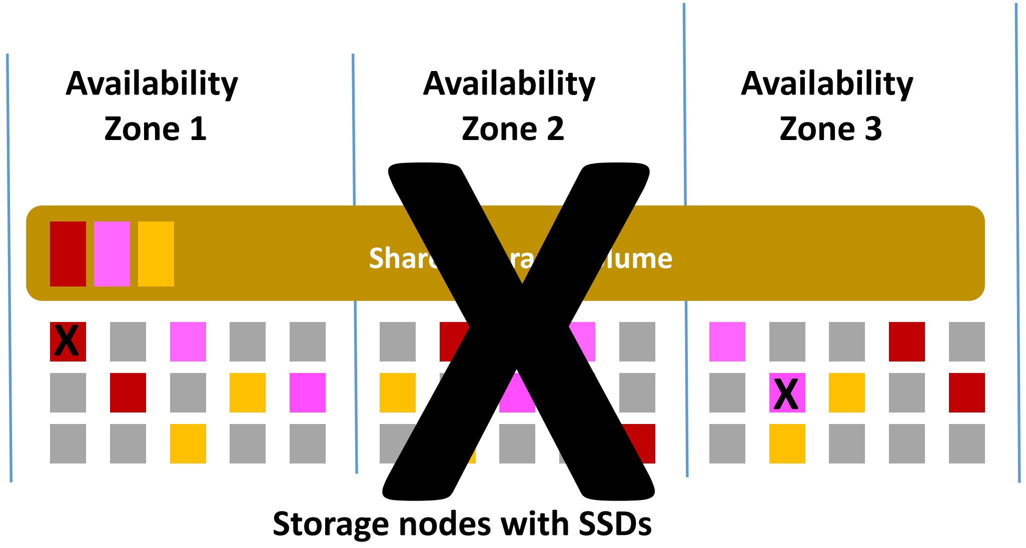
Is a 4/6 quorum sufficient for AZ+1 ?

Depends on repairing a failed node before AZ+1 becomes AZ+2 (double-fault)

P(AZ+2) in repair interval is function of MTTF

- Can only reduce MTTF & P(AZ+2) so much

Instead try to reduce repair interval (MTTR)



Segmented Storage

Partition volume into n fixed size segments

Replicate each segment 6-ways into a Protection Group (PG)

A single PG failing is enough to fail the entire volume

Probability is additive since failures are independent

$$P(Volume\ failure) = \sum_{i=1}^{n} P(Failure\ in\ PG\ i)$$

What is the "Goldilocks" segment size ?

Trade-off between likelihood of faults and time to repair

- If segments are too small then failures are more likely
- If segments are too big then repairs take too long

Choose the biggest size that lets us repair "fast" enough

- We currently picked a segment size of 10GB
- Can repair a 10GB segment in **~10 seconds** on a 10Gbps link

10GB segments are the unit of independent failure & repair

Can we really afford 6 copies ??
(aka Burden of Amplified Writes)

The Log is the Database

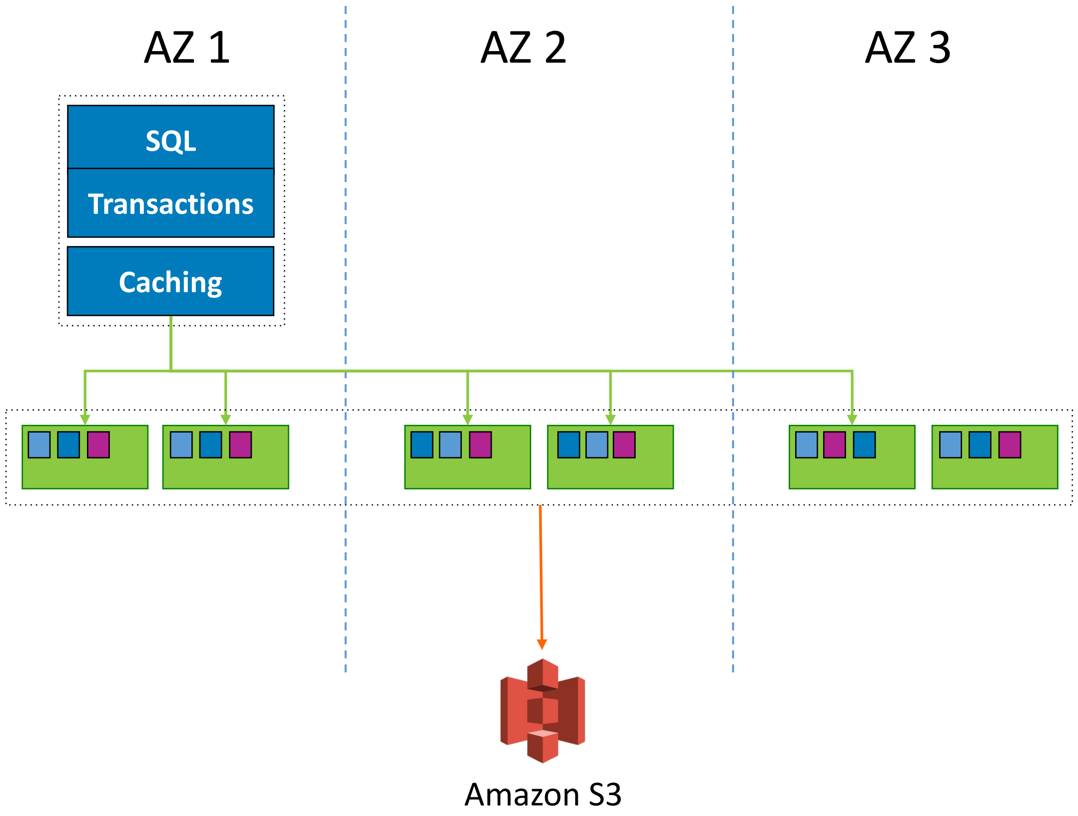
Offload redo processing to storage

- Only write redo log records on network
- Push log applicator to storage tier
- Generate database pages on demand
- Materialize database pages in background
- Continuous backup to S3

Redo and backups are now parallelized

No more full page writes

- Checkpointing, cache eviction, bg writer



How does Aurora handle the network bottleneck

Decorrelating storage and instance failures requires network storage. This creates high Network:Data (N:D).

Latency, availability, durability concerns benefit from quorums. Amplifies already high N:D

Correlation of failures require tolerating loss of AZ+1. Further amplifies N:D

Pushing most work down to storage tier greatly reduces N:D

Even so, Aurora is rate-limited by available network PPS.

Lets look at I/O for data warehouses

The network is a shared resource

On a fleet-wide basis,
Amazon Redshift might be doing
1 EB of physical I/O operations daily

(excludes cache hits)

2^60 bytes, or
2^40 1MB block I/Os

If you go over the network,
2^50 1K packets

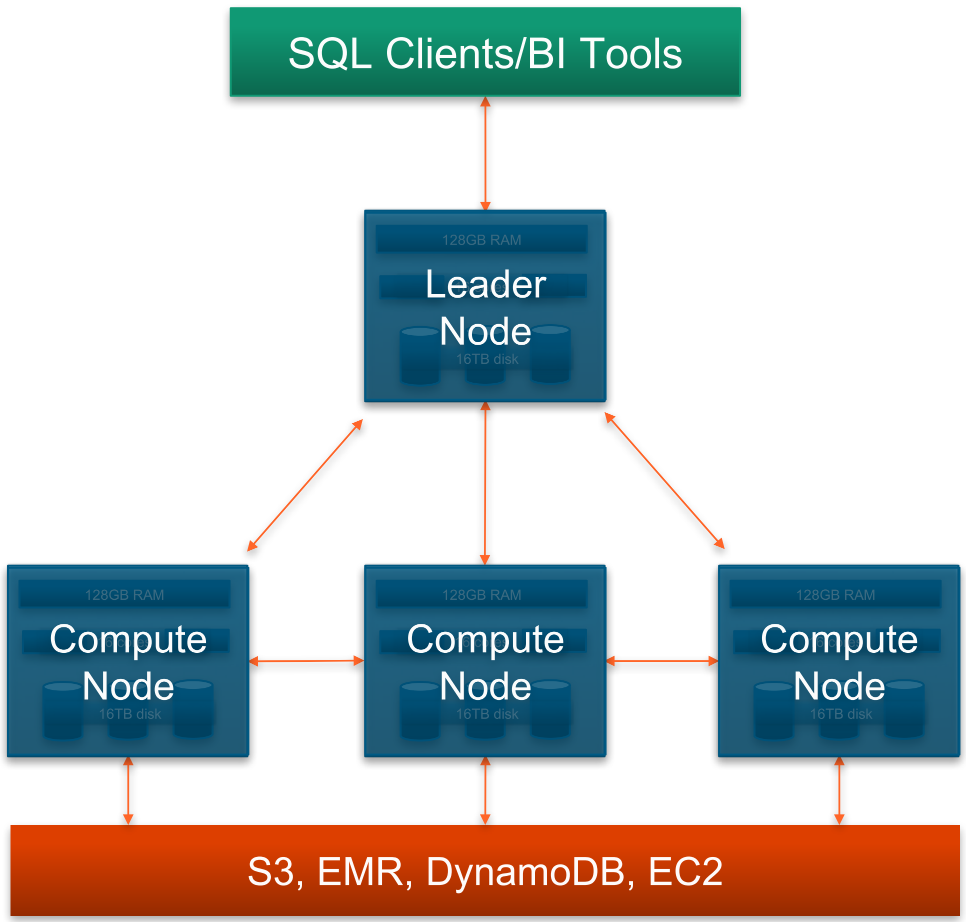
That’s a lot

Data in S3 is encoded
to achieve 11 9’s durability
inexpensively

2^55 network packets before hops, retransmits

Redshift is designed to minimize network

- Like Aurora, most work is pushed down to compute nodes
- Compute nodes have local disks to reduce I/O traffic
- Data is distributed to minimize communication across nodes



Our customers increasingly are moving to Data Lake architectures

Data in open formats in a highly durable, low cost data store (e.g. S3)

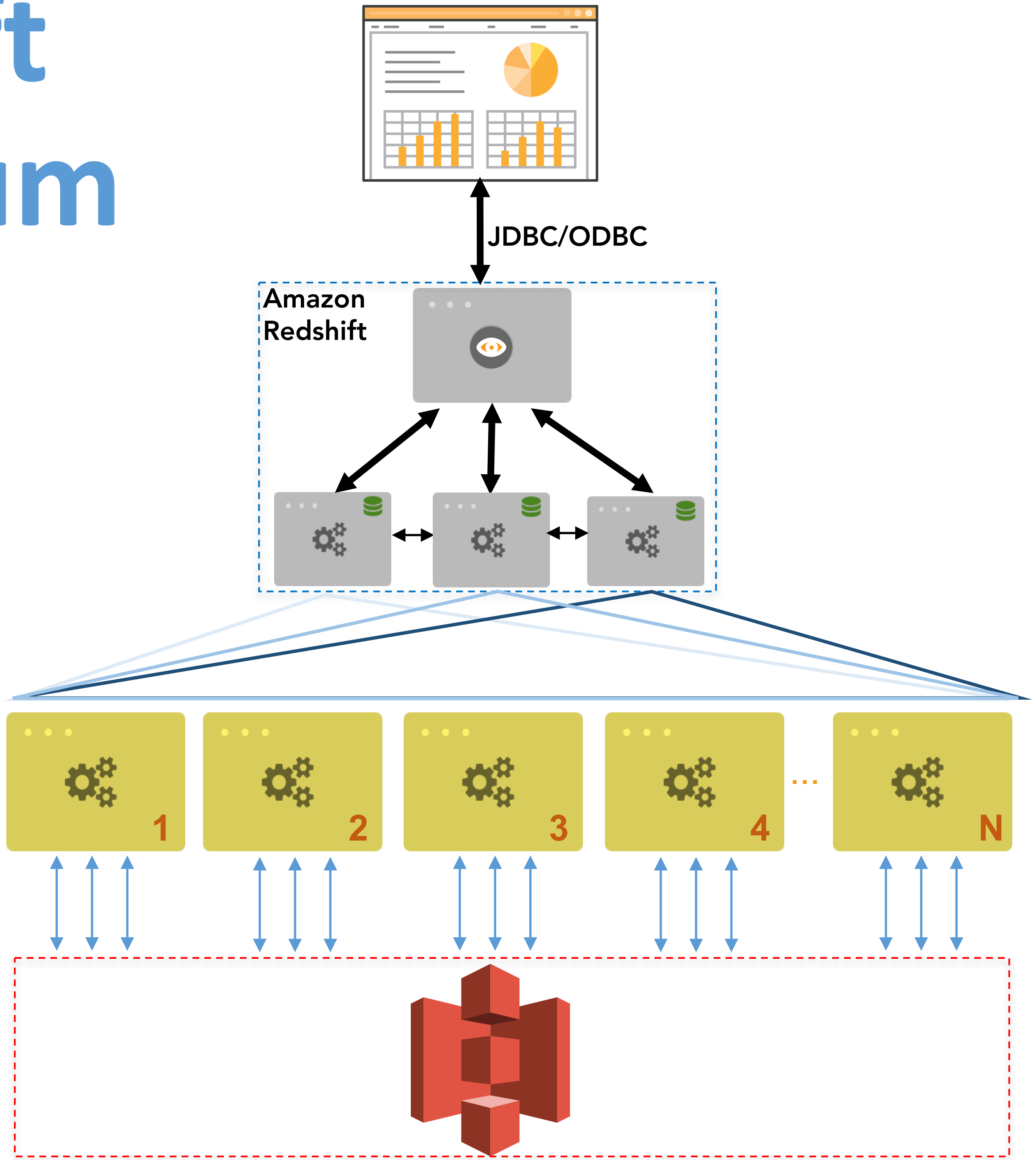
Accessible from their data processing engine of choice (Hadoop, data warehouse, serverless SQL, etc)

You need a lot of network pipes to pull data

You can’t afford to keep them when not in use

You need to avoid East-West traffic

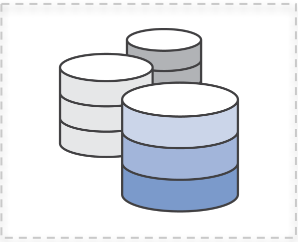
Redshift Spectrum



Amazon S3

Exabyte-scale object storage

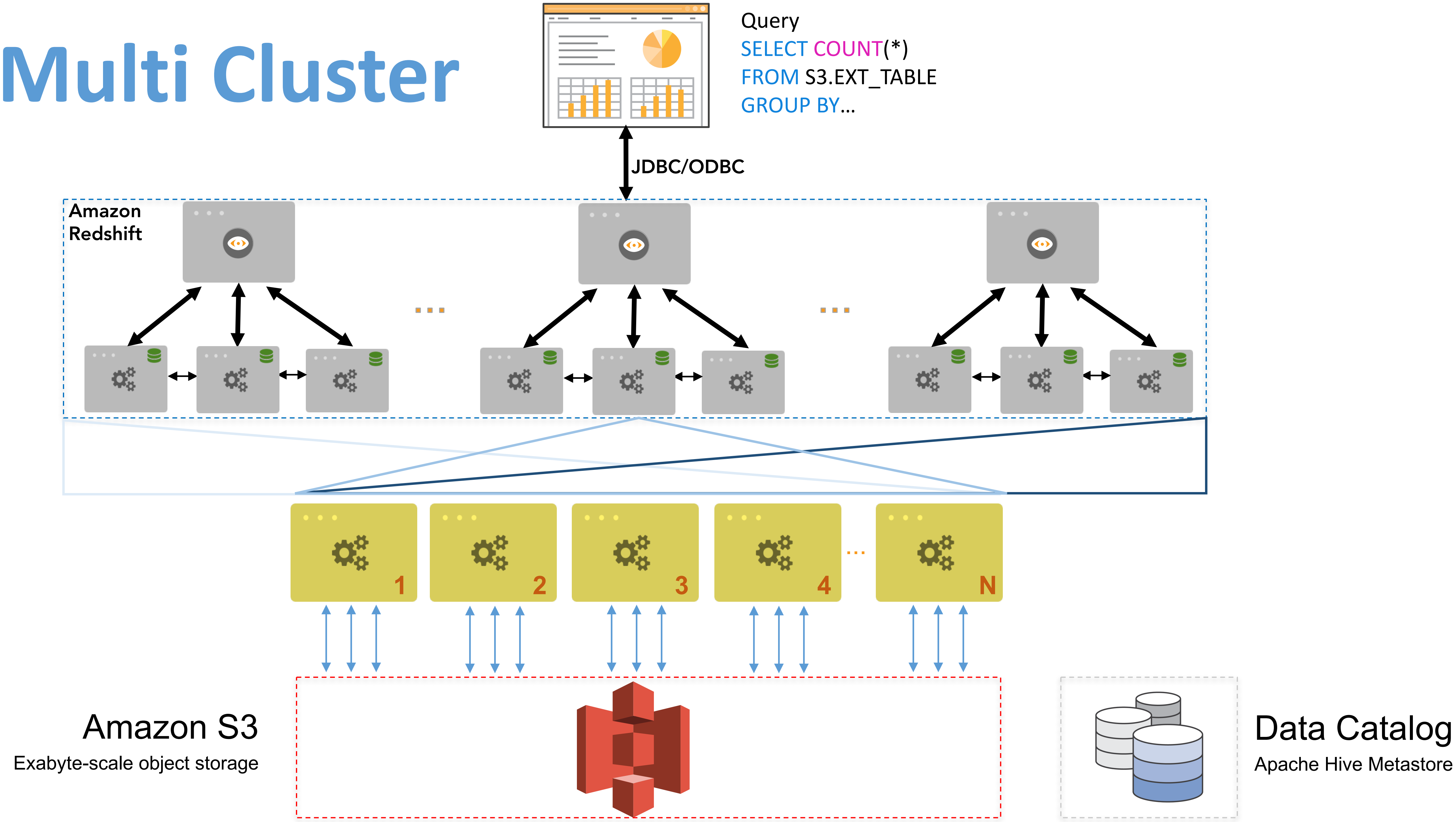
- Full query optimizer and SQL semantics,
- Scale-out resource pool with pay-per-query allocation of nodes to pull data from S3
- Push down scan, filter, aggregate, group…



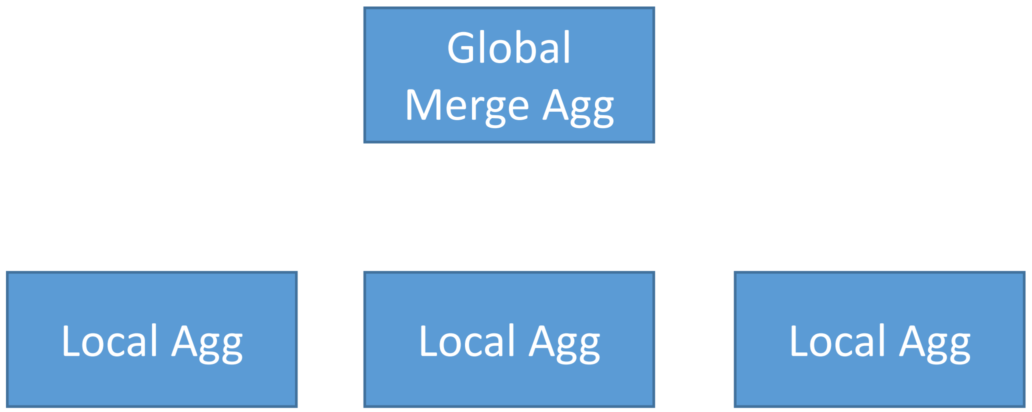
Data Catalog

Apache Hive Metastore

Multi Cluster

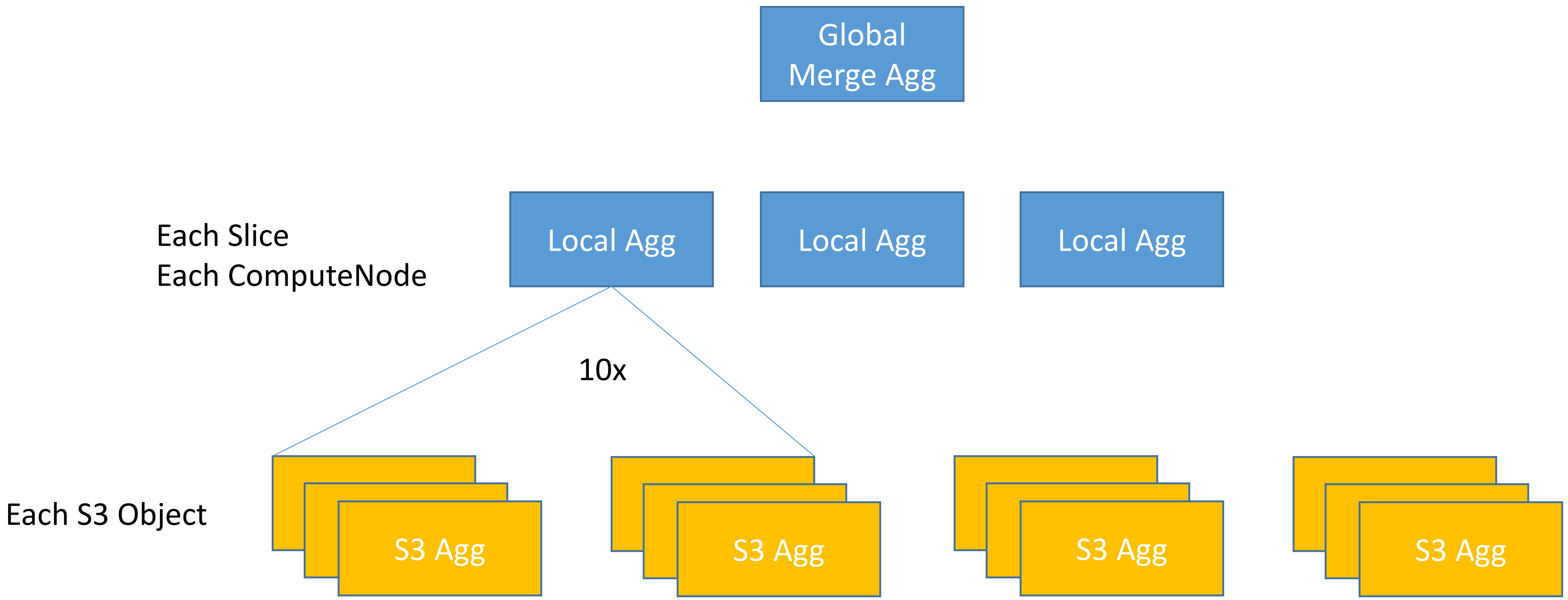


Large-scale SCAN-and-AGG

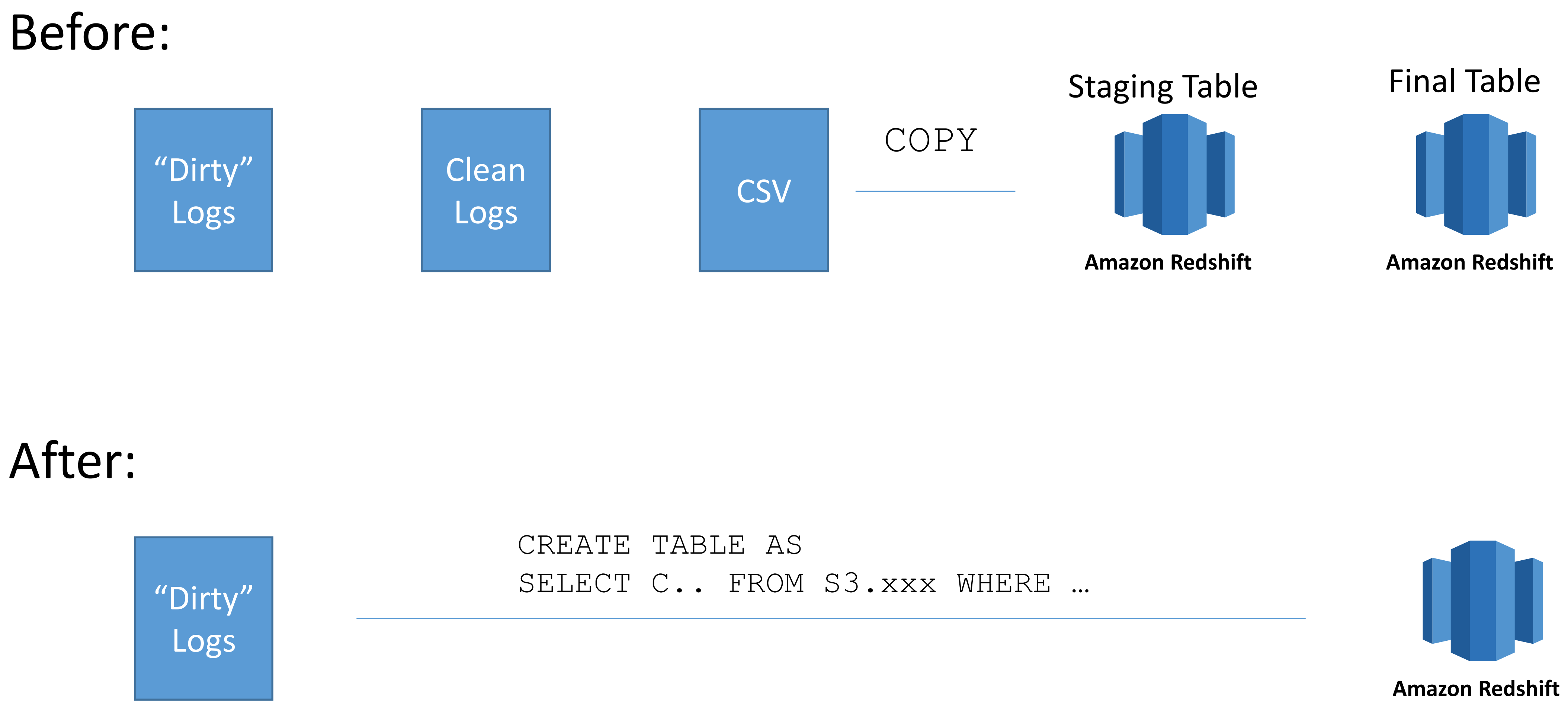


Each Slice
Each ComputeNode

Large-scale SCAN-and-AGG



Simplified Ingestion Pipelines



Future work

- How do we push query processing down into the data lake itself?
- What does it mean for encoding schemes traditionally optimized for high durability at low cost?
- How do we manage transactionality across data processing engines?
- How do we manage access control across data processing engines?