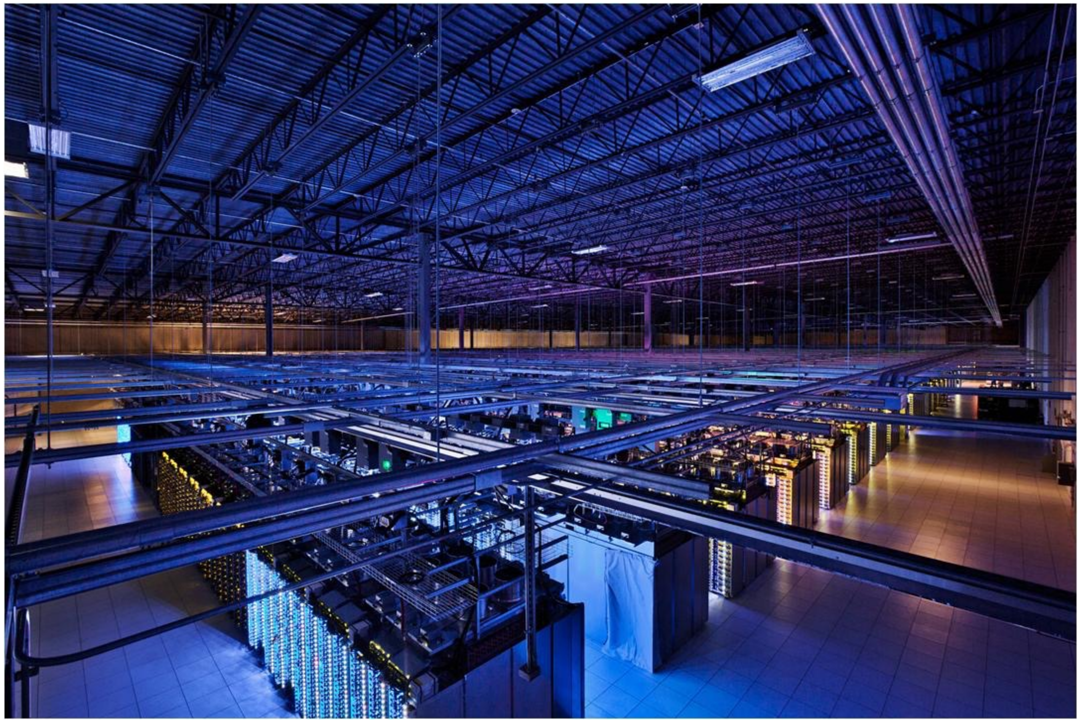
Centrum Wiskunde & Informatica

Large Scale Data Engineering

Cloud Database Systems

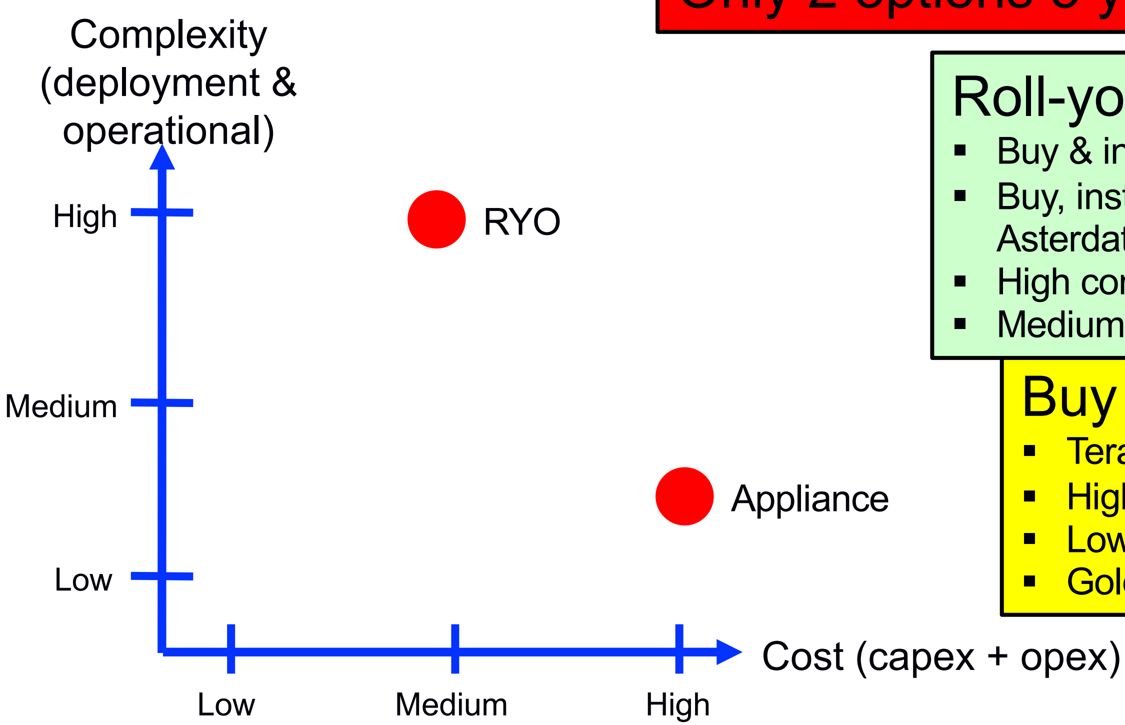


Credits

- David DeWitt & Willis Lang (Microsoft)
  – cloud DW material
- Marcin Zukowski (Snowflake)
- Ippokratis Pandis (Amazon Redshift/Spectrum)
- Steven Bryen (Amazon Aurora)
- Spark Team
  – Matei Zaharia, Xiangrui Meng (Stanford),
  – Ion Stoica, Xifan Pu (UC Berkeley)
  – Reynold Xin, Alex Behm (Databricks)

10,000 ft. view: Complexity vs Cost

Only 2 options 5 years ago!



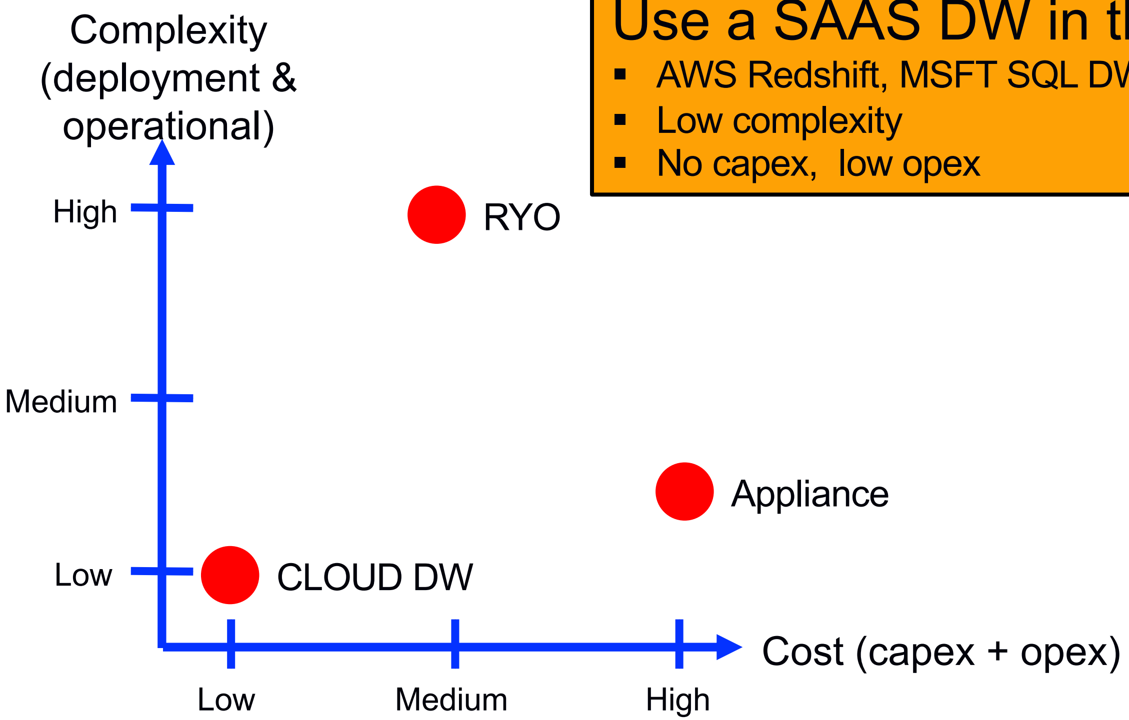
Roll-your-own (RYO)

- Buy & install a cluster of servers
- Buy, install, & configure software (Vertica, Asterdata, Greenplum, …)
- High complexity
- Medium capex and opex

Buy an appliance

- Teradata, Oraclle Exadata, Netezza
- High capex, low opex
- Low complexity
- Gold standard for performance

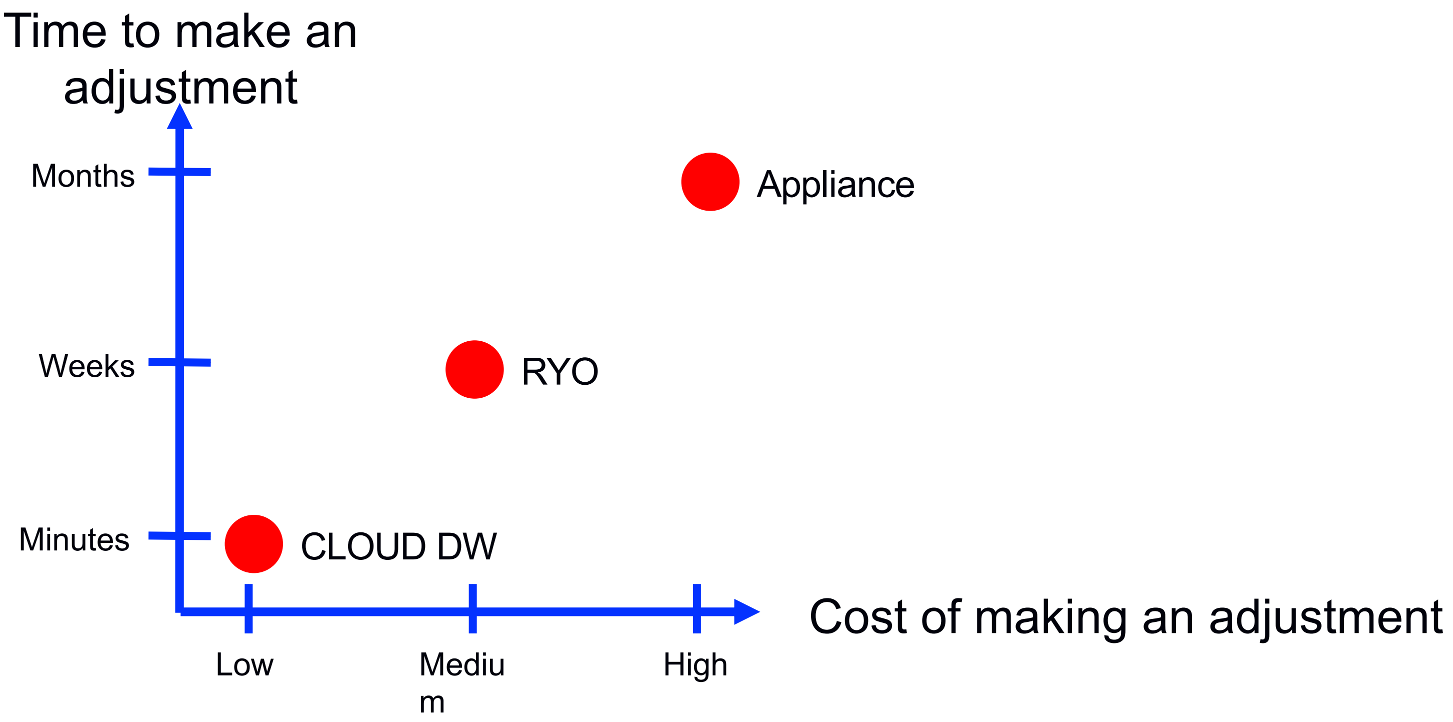
10,000 ft. view: Complexity vs Cost



Use a SAAS DW in the cloud

- AWS Redshift, MSFT SQL DW, Snowflake, BigQuery
- Low complexity
- No capex, low opex

Scalability and the price of agility



Why Cloud DW?

- No CapEx and low OpEx
- Quick deployment
- Low storage prices (Azure, AWS S3, GFS)
- Flexibility to scale up/down compute capacity
- Simple upgrade process

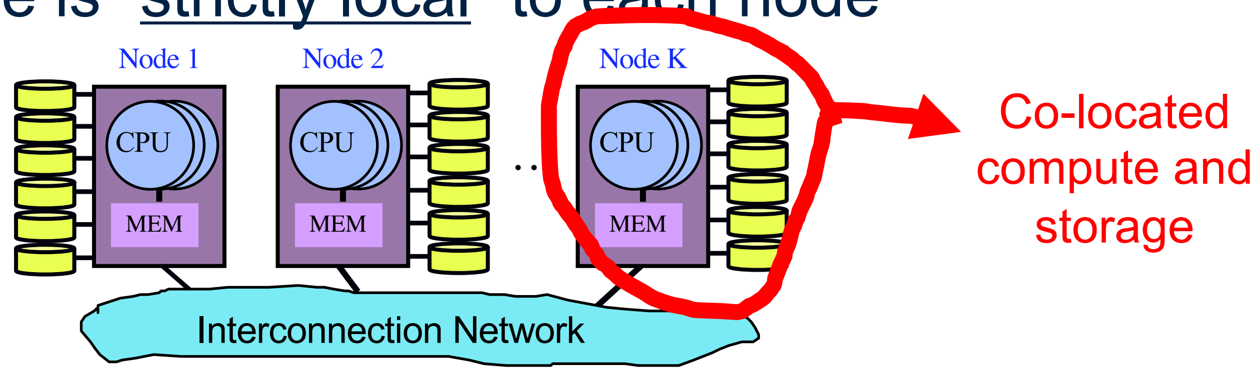
On-Premise Parallel Analytical DBs

- Alternative architectures
  - Shared-memory
  - **Shared-disk/storage**
  - **Shared-nothing**
- Partitioned tables
- Partitioned parallelism

"The Case for Shared Nothing,"
Stonebraker, HPTS '85

Shared-Nothing

- Commodity servers connected via commodity networking
- DB storage is ”strictly local” to each node



- Design scales extremely well

Shared Disk/Storage

- Commodity servers connected to each other and storage using commodity networking
- ote storage” (e.g. a SAN, S3, Azure

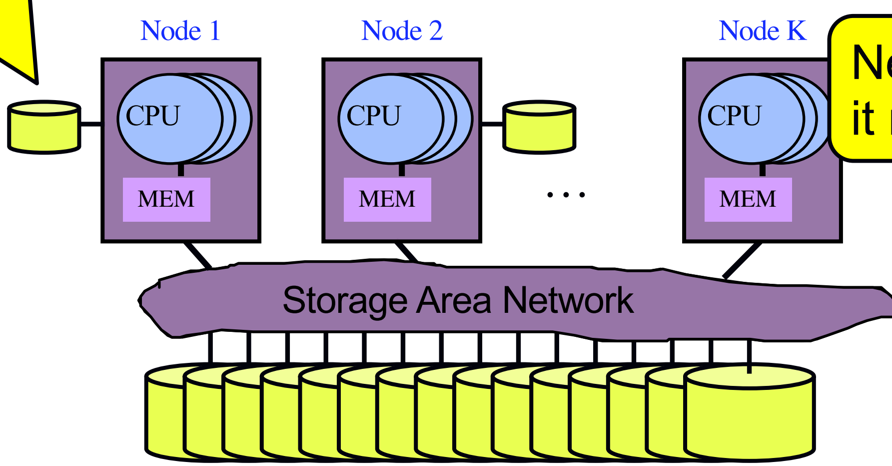


Table Partitioning

What?

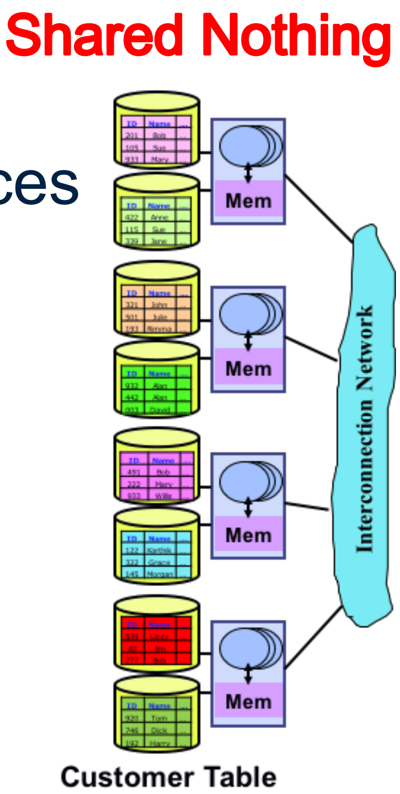
Distribute rows of each table across multiple storage devices

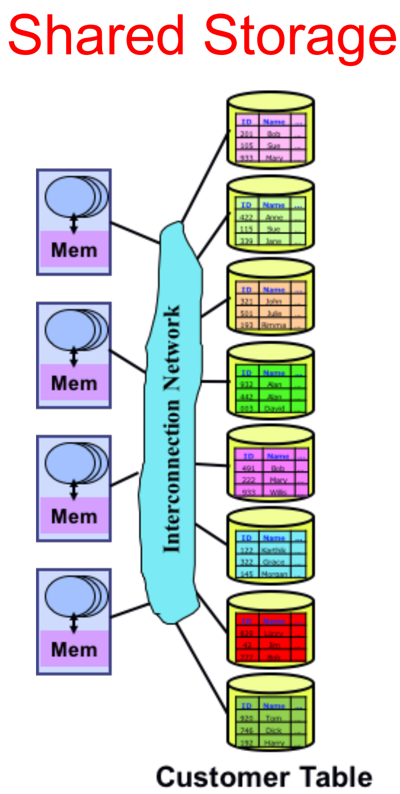
Why?

- spread I/O load
- parallel query execution
- data lifecycle management

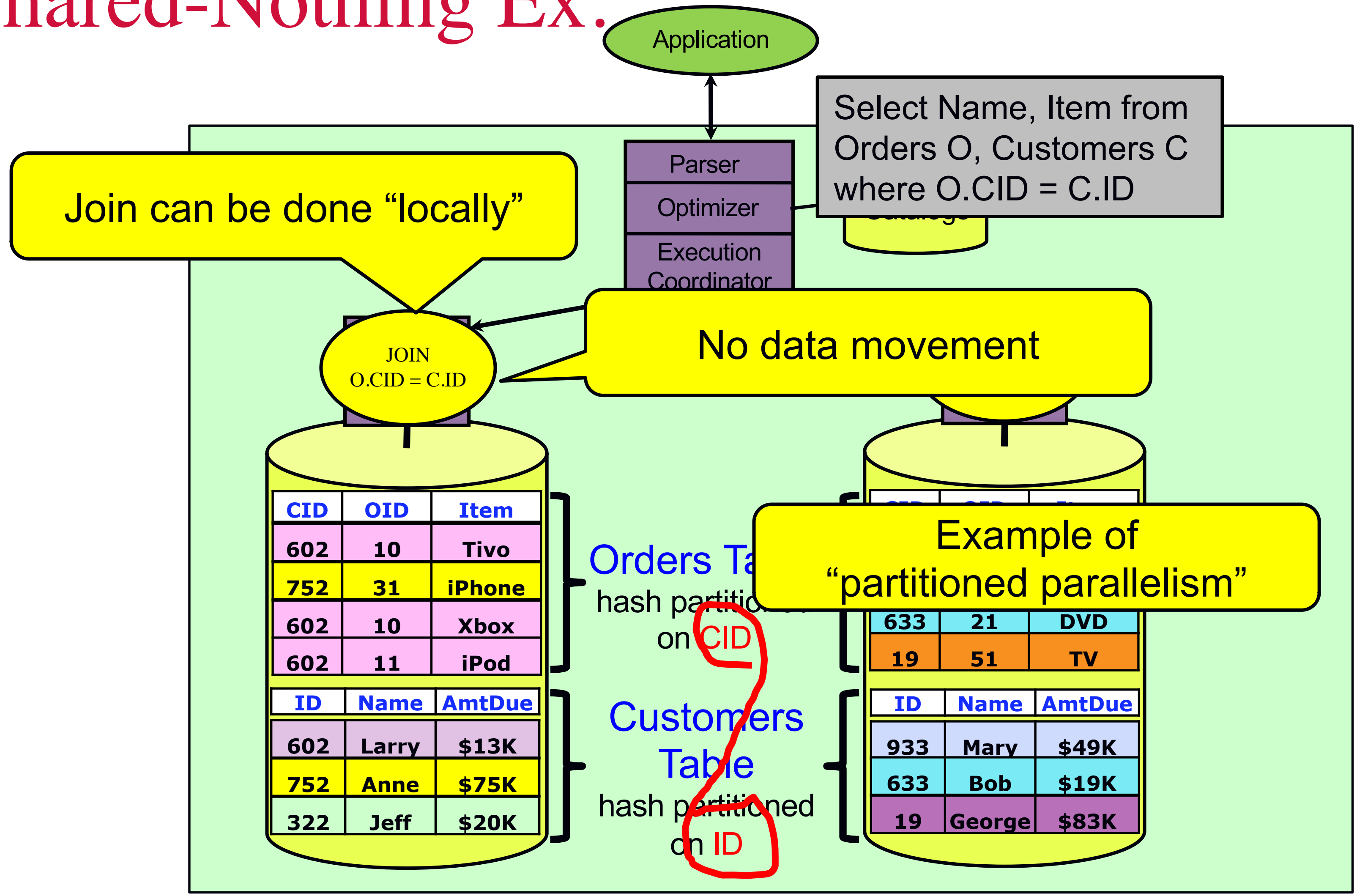
How?

Hash, Round Robin, Range

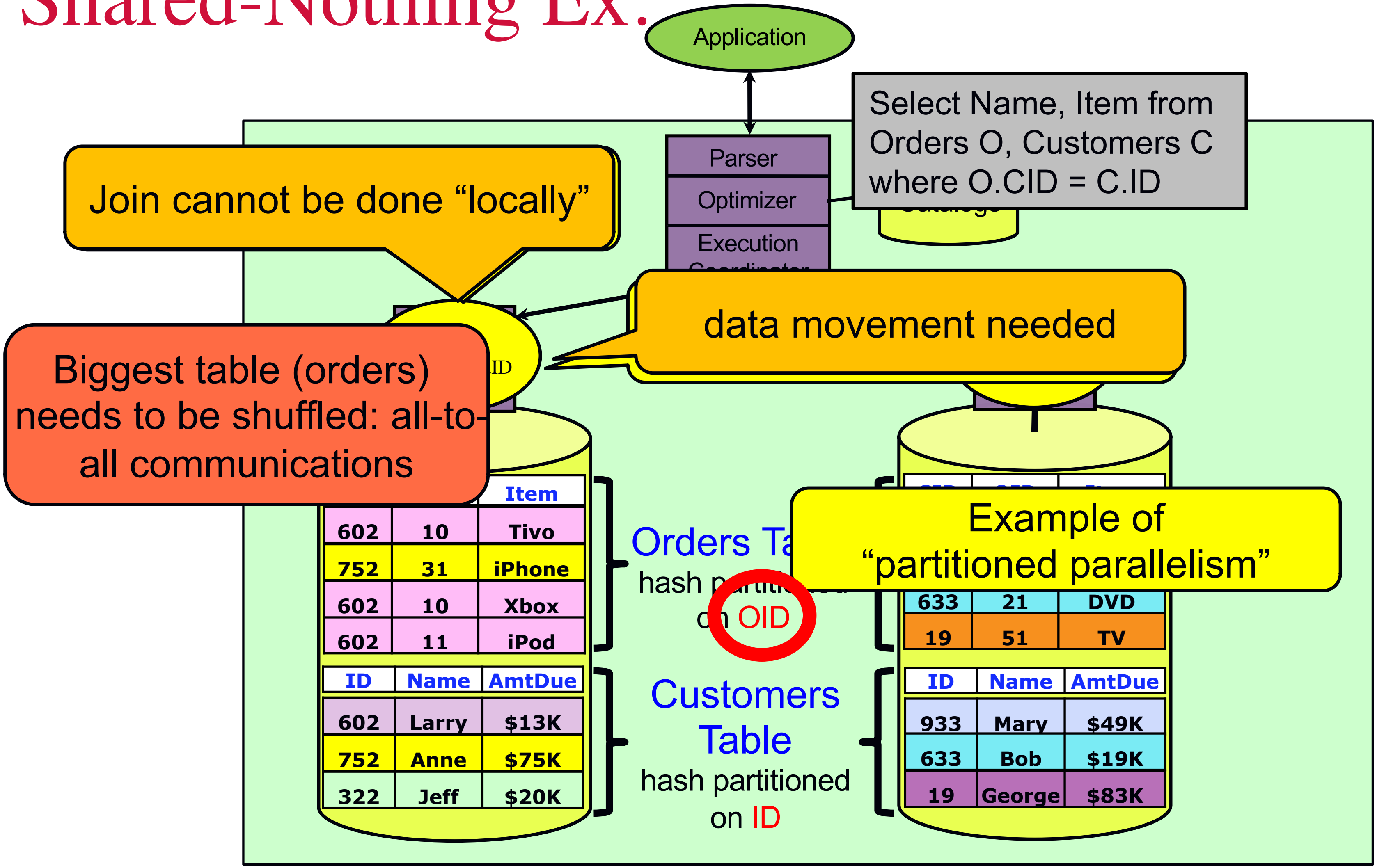




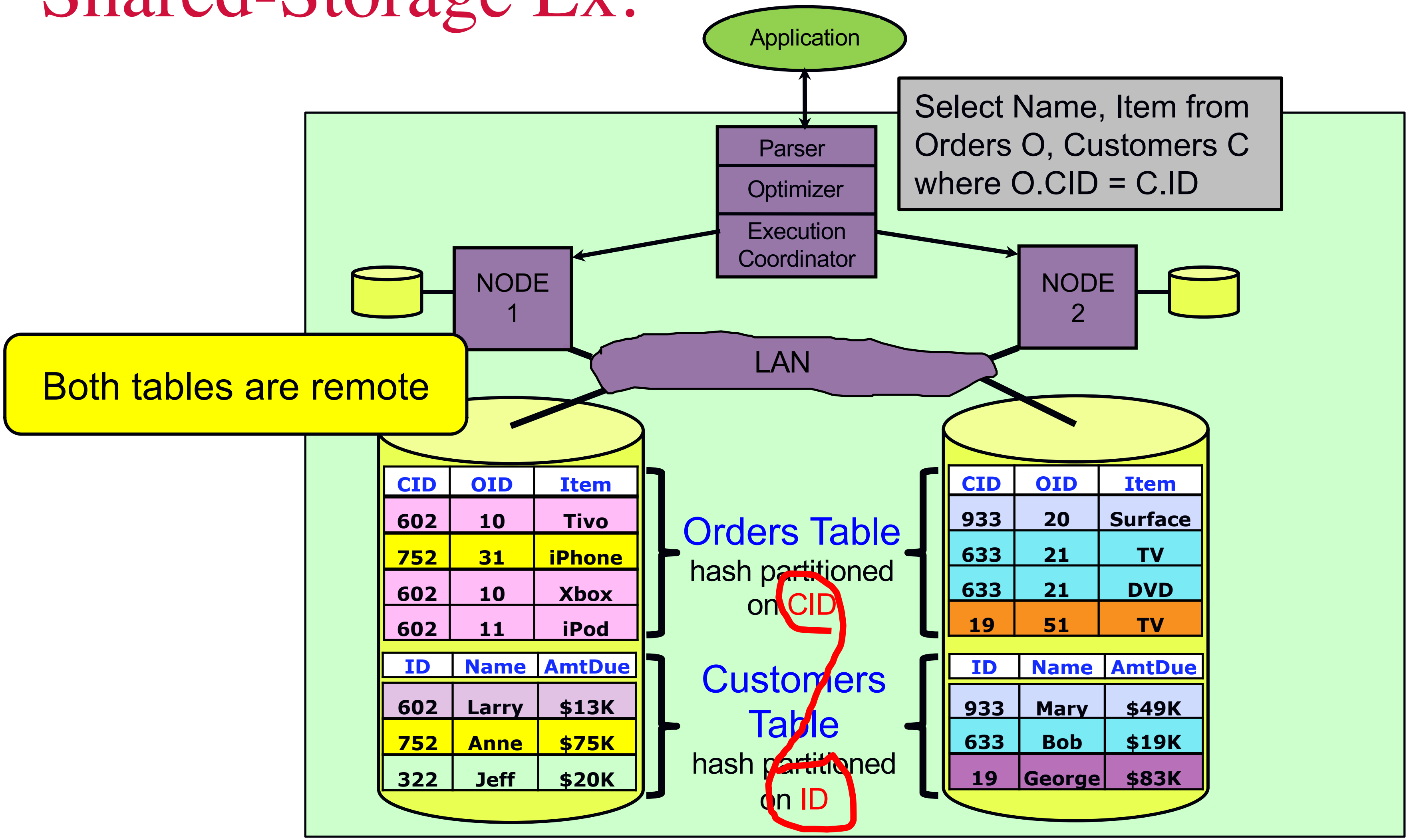
Shared-Nothing Ex.



Shared-Nothing Ex.



Shared-Storage Ex.



CID	OID	Item
602	10	Tivo
752	31	iPhone
602	10	Xbox
602	11	iPod

ID	Name	AmtDue
602	Larry	$13K
752	Anne	$75K
322	Jeff	$20K

CID	OID	Item
933	20	Surface
633	21	TV
633	21	DVD
19	51	TV

ID	Name	AmtDue
933	Mary	$49K
633	Bob	$19K
19	George	$83K

CWI

Centrum Wiskunde & Informatica

For 30+ years

- Shared-nothing has been "gold standard"
  - Teradata, Netezza, DB2/PE, SQLserver PDW, ParAccel, Greenplum
- Simplest design
- Excellent scalability
- Minimizes data movement
  - Especially for DBs with a star schema design
- The "cloud" has changed the game
  - shared nothing:

CWI
Centrum Wiskunde & Informatica

Outline

- Part 1: Intro
- Part 2: Analytic Databases in the Cloud
  - Snowflake
  - Amazon Redshift
  - Google BigQuery
  - Databricks
- Part 3: Transactional Databases in the Cloud
  - Amazon RDS ➔ Aurora

Snowflake Elastic DW

- Shared-storage design (cloud storage = “shared storage”)
  - Compute decoupled from storage, Highly elastic
- Columnar Compressed Store
  - Native data format
  - Stored internally by Snowflake
- Leverages AWS
  - Tables stored in S3 but dynamically cached on local storage
    Clusters of EC2 instances used to execute queries
- Rich data model
  - Schema-less ingestion of JSON documents
  - Regular parts automatically converted to column-store

Snowflake Architecture

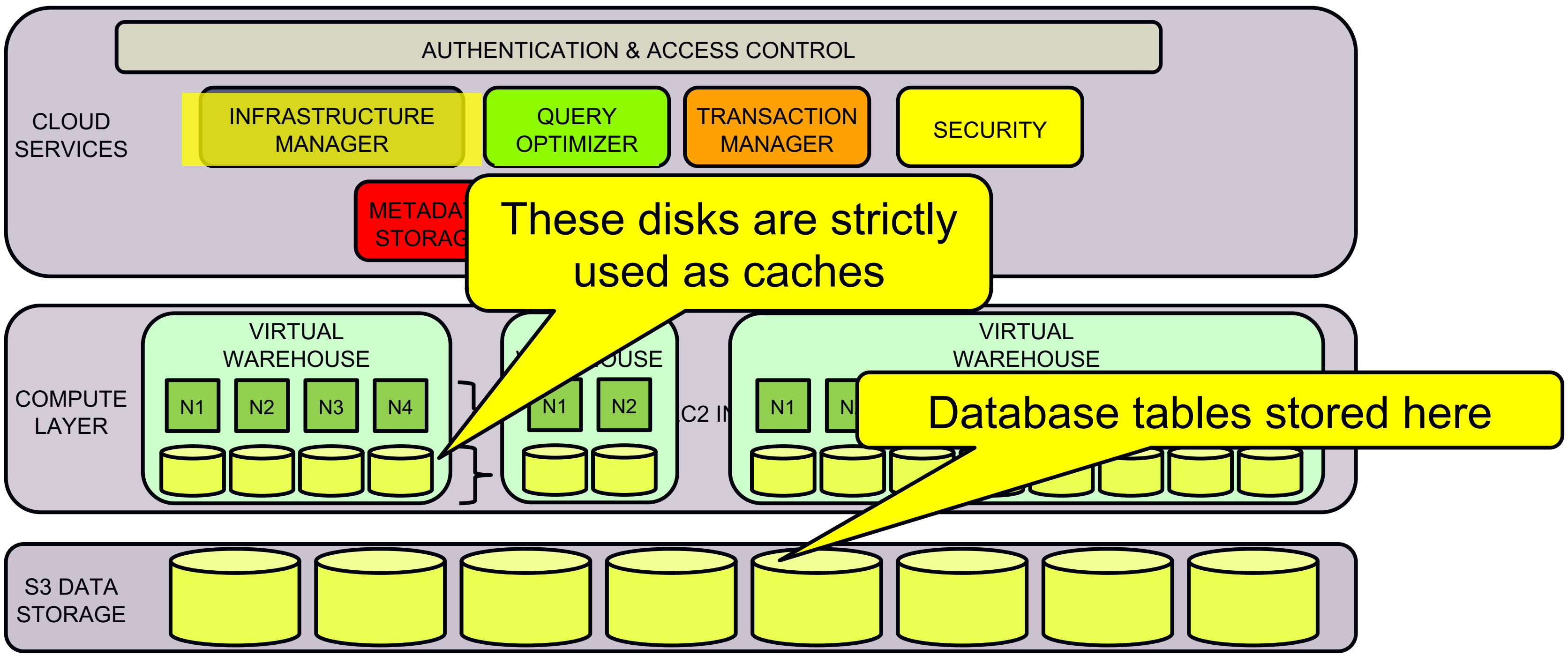
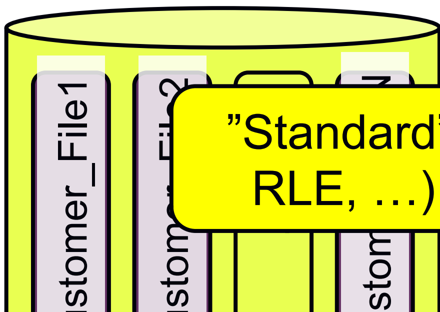


Table Storage

- Rows of each table are stor
  S3 files:

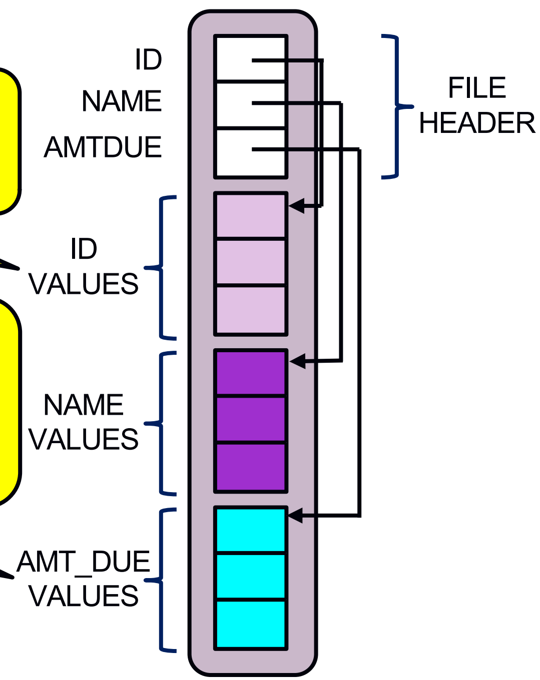
Not able to support hash or RR partitioning as files are created strictly as rows are inserted into table



"Standard" compression (gzip, RLE, …) schemes available

- Each

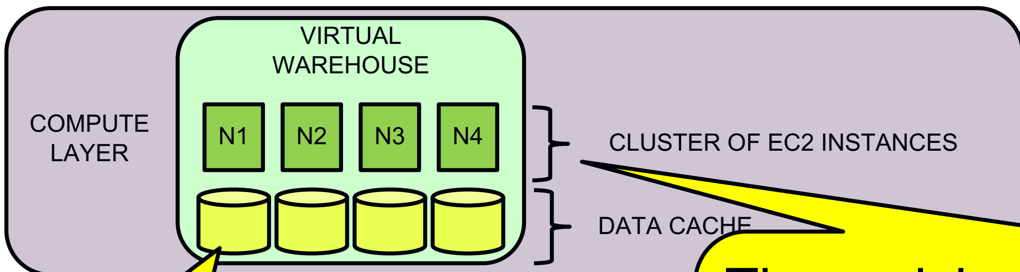
ows stored in columnar fashion



Min & max value of <u>each</u> column of <u>each</u> file of <u>each</u> table are kept in catalog. Used for pruning at run time.

Virtual Warehouses

Dynamically created cluster of EC2 instances

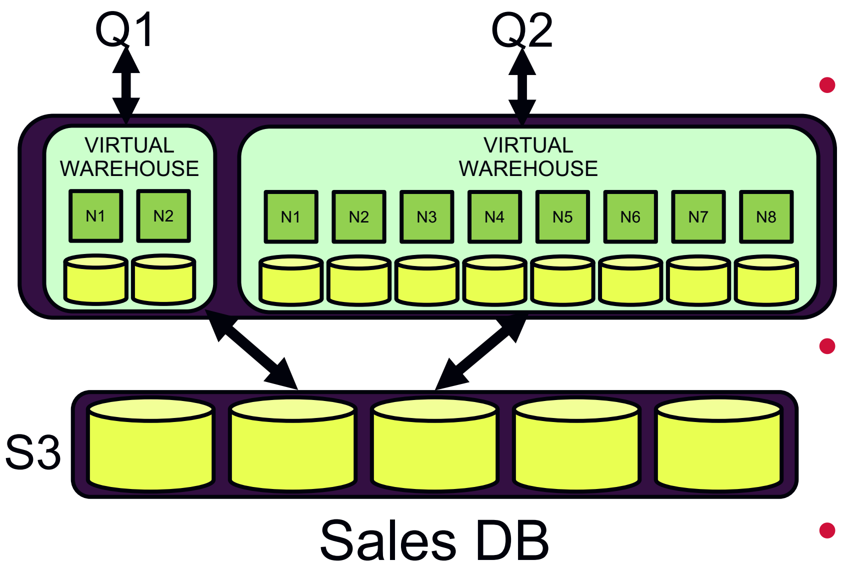


Three sizing mechanisms:

1) Number of EC2 instances
2) ”Size” of each instance (# cores, I/O capacity)
3) Auto-scaling of one virtual warehouse

Local disks cache file headers & table columns

Separate Compute & Storage.

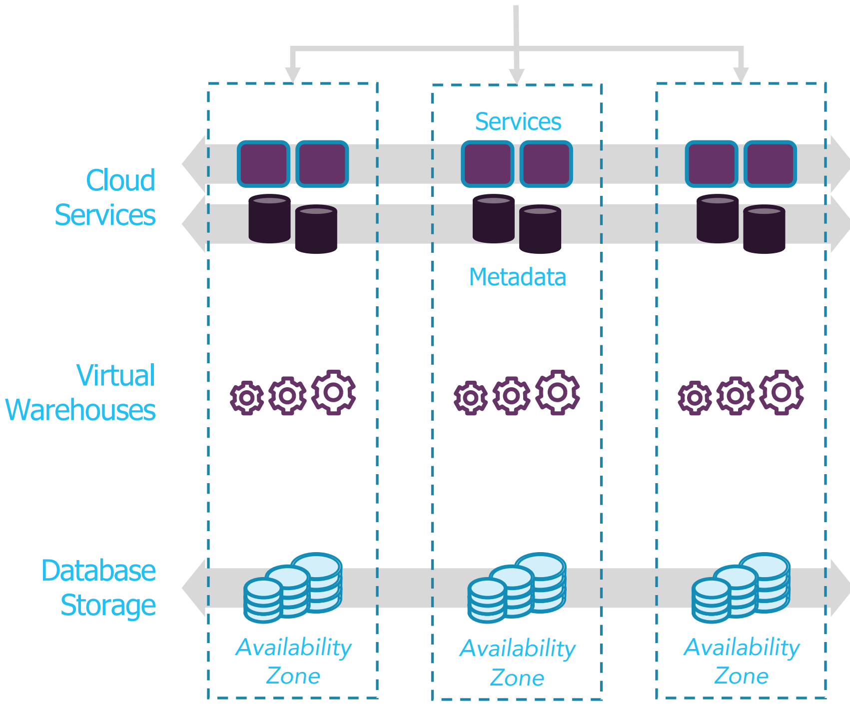


- Queries against the <u>same DB </u>can be given the resources to meet their needs – <u>truly unique idea</u>
- DBA can dynamically adjust number & types of nodes
- This flexibility is simply not feasible with a shared-nothing approach such as RedShift.

Query Execution

- Each query runs on a single virtual warehouse
- Standard parallel query algorithms
- Modern SQL engine:
  - Columnar storage, Vectorized executor
- Updates create new files!
  - Artifact of S3 files not being updatable.
  - But makes **time travel** possible (table "forking" and "cloning")
    - zero-copy: just share certain S3 files

High Availability



Scale-out of all tiers:

metadata, compute, storage

Resiliency across multiple availability zones

- Geographic separation
- Separate power grids
- Built for synchronous replication

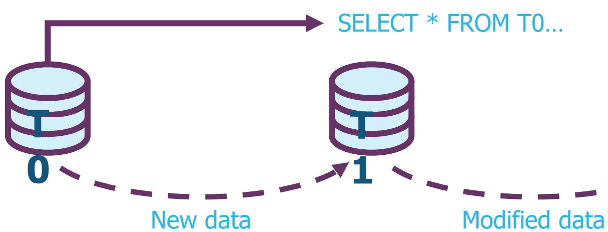
Fully online updates & patches (zero downtime)

Fully managed by Snowflake

High Availability

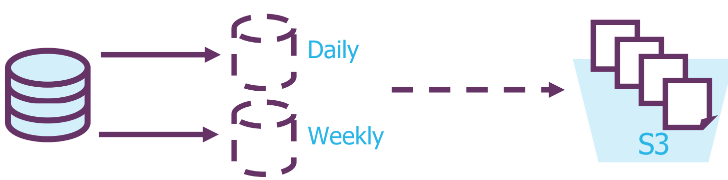
Protection against infrastructure failures

All data transparently & synchronously replicated 3+ ways across multiple datacenters



Protection against corruption & user errors

“Time travel” feature enables instant roll-back to any point in time during chosen retention window



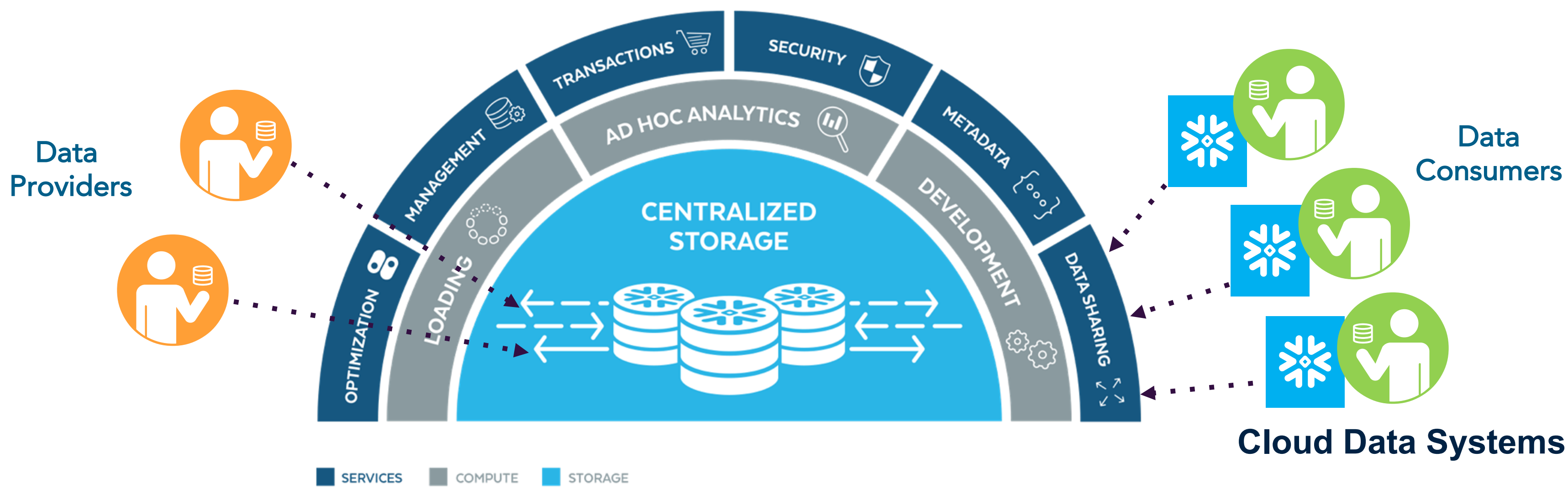
Long-term data protection

Zero-copy clones + optional export to S3 enable user-managed data copies

Secret Weapon: Data Sharing

Providers
- Secure and integrated Snowflake's access control model
- Only pay normal storage costs for shared data
- No limit to the number of consumer accounts with which a dataset may be shared

Consumers
- Get access to the data without any need to move or transform it.
- Query and combine shared data with existing data or join together data from multiple publishers



Snowflake Summary

- Designed for the cloud from conception
- Can directly query unstructured data (json) w/o loading
- Compute and storage independently scalable
  - Virtual warehouses composed of clusters of AWS EC2 instances
  - VWs cache parts of table data. Tables are shared between VWs
- Data Sharing
  - Available across AWS, Azure, Google Cloud
  - Data economy/marketplace
- No management knobs: ease of use
  - No indices, no create/update stats, no distribution keys, …

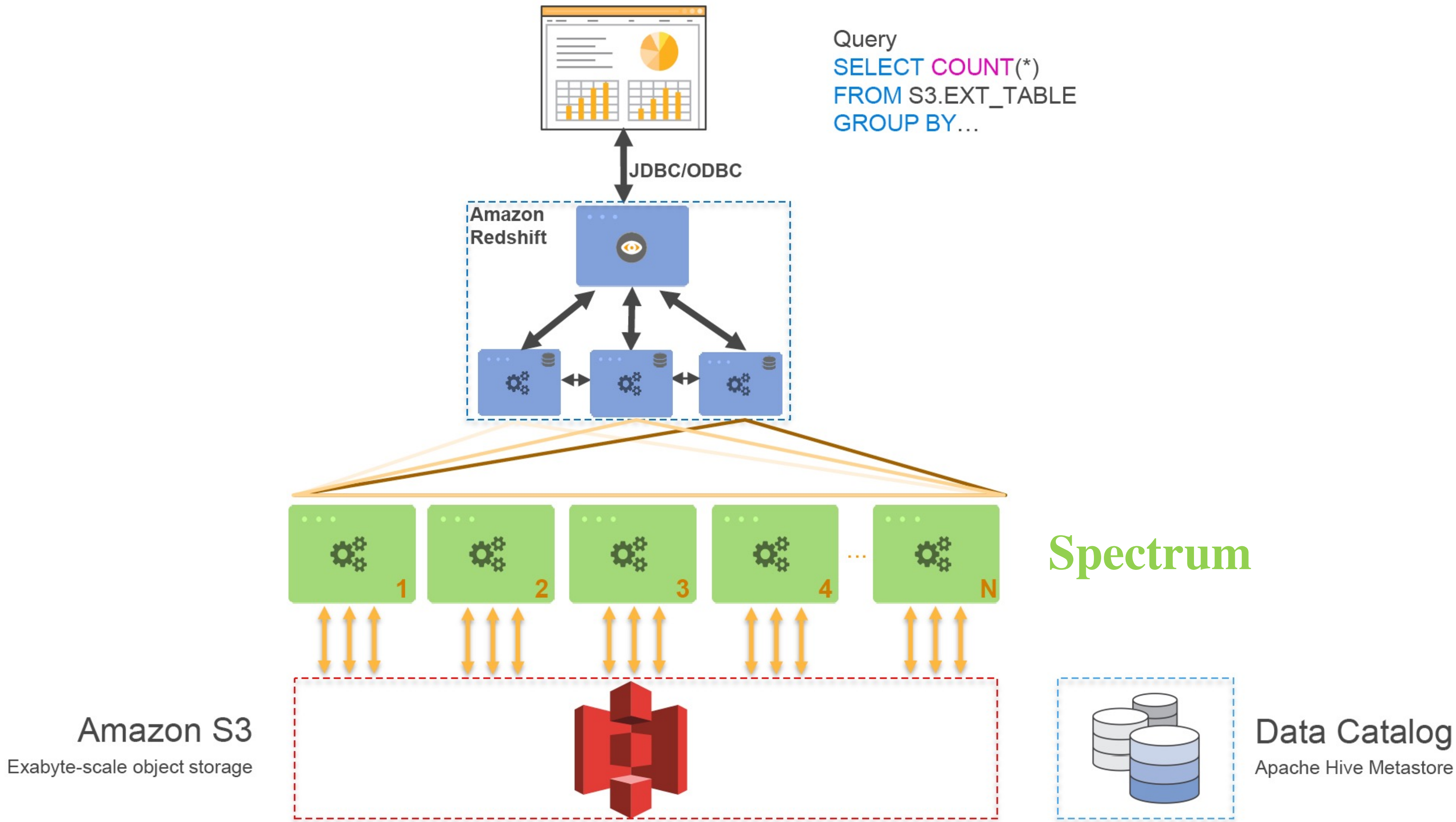
Outline

- Part 1: Intro
- Part 2: Analytic Databases in the Cloud
  - Snowflake
  - Amazon Redshift
  - Google BigQuery
  - Databricks
- Part 3: Transactional Databases in the Cloud
  - Amazon RDS → Aurora

Amazon (AWS) Redshift

- Leader in market adoption (still?)
  - Snowflake is biggest competitor now
- Classic shared-nothing design w. locally attached storage
  - Engine is ParAccel database system (shared-nothing MPP, JIT C++)
  - Native data format (compressed columnar)
  - Leverages local storage, even co-partitioning of tables (local joins)
  - Storage and compute do not scale independently
    - But.. Redshift is becoming more elastic, cloud-centric
- Spectrum subsystem
  - scalable/elastic access external data sources (e.g. Parquet on S3)
  - allows to push data partial queries to external workers

Redshift Spectrum



Outline

- Part 1: Intro
- Part 2: Analytic Databases in the Cloud
  - Snowflake
  - Amazon Redshift
  - Google BigQuery
  - Databricks
- Part 3: Transactional Databases in the Cloud
  - Amazon RDS ➔ Aurora

Google BigQuery

- **Separate storage and compute**
- Leverages Google's internal storage & execution stacks:
  - Collosus distributed file system
  - **DremelX** query executor
  - Jupiter networking stack
  - Borg resource allocator
- No knobs, no indices, …

Serverless ➔ you do not start any machines, Google just runs your queries

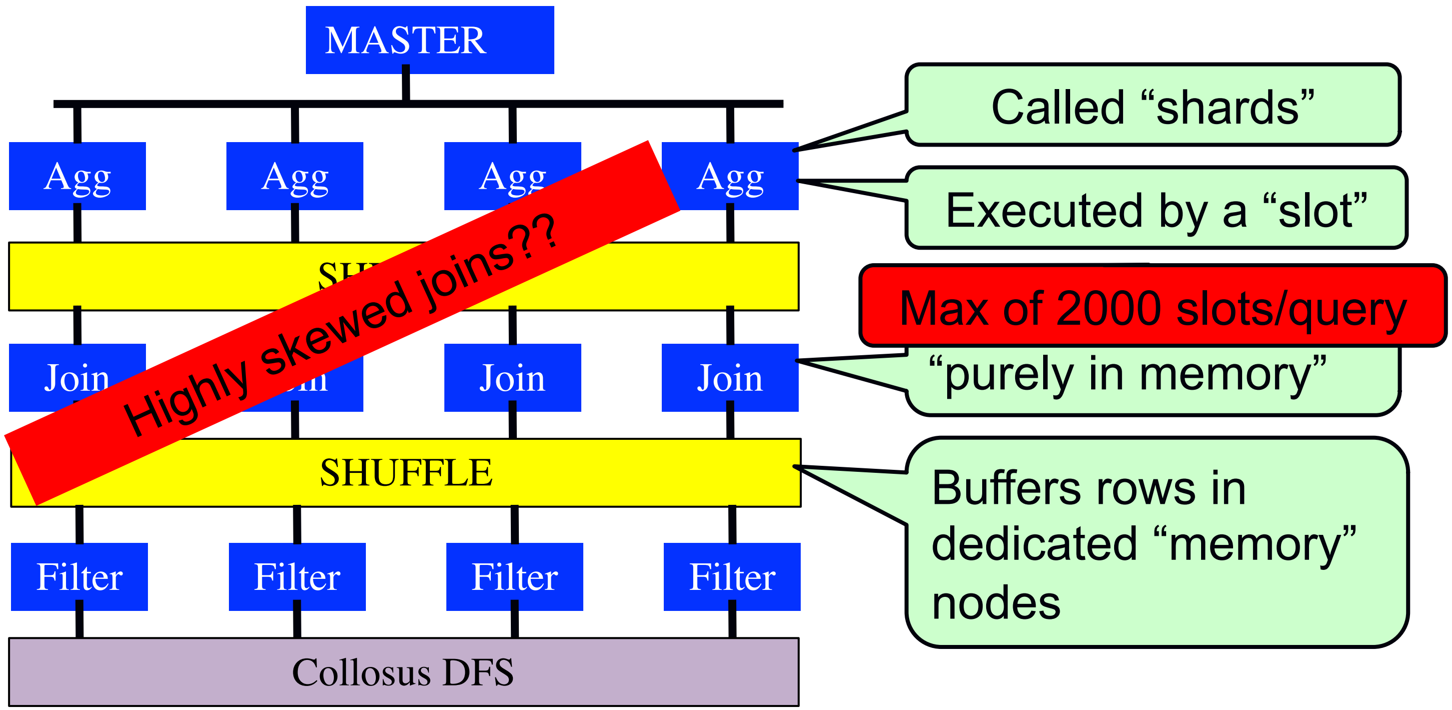
AWS Athena is similar ➔ Serverless Presto in Cloud

BigQuery Tables

- Stored in Collosus FS
  - Partitioned by day (optionally)
- Columnar storage (Capacitor)
  - RLE compression
  - Sampling used to pick sort order
  - Columns distributed across multiple disks
- Also “external” tables
  - JSON, CSV & Avro formats
  - Google Drive and Cloud Storage

Query Execution

SQL queries compiled into a tree of DremelX operators



CPU Resource Allocation

- Compute resources not dedicated!
  - Shared among other internal and external customers
  - No apparent way to control computational resources used for a query
- # of shards/slots assigned to an operator function of:
  - Estimated amount of data to be processed
  - Cluster capacity and current load

BigQuery Pricing

- Storage: $0.02/GB/month

  (AWS is about $0.023/GB/month)

- Query options

  1) Pay-as-you-go: $5/TB “processed”

     - calculated after column is uncompressed

  (AWS is about $1.60/TB using M4.4Xlarge EC2 instance)

  2) Flat rate: $40,000/month for 2,000 dedicated slots

Outline

- Part 1: Intro
- Part 2: Analytic Databases in the Cloud
  - Snowflake
  - Amazon Redshift
  - Google BigQuery
  - Databricks
- Part 3: Transactional Databases in the Cloud
  - Amazon RDS ➔ Aurora

Databricks

- **Spark**-as-a-service in the cloud ("from the makers of")
  - All data stored in S3, in **open formats**
- Clusters run in the **user** account
  - Data in the **user** account
  - Control plane runs in **Databricks** account
- User can dynamically power up and down clusters
  - Clusters can be grown and shrunk

MLflow: environment for reproducible ML experiments

Delta Lake ("Lakehouse")

- **Delta Tables**: Transactional Cloud Table Storage
  - All data stored in **Parquet, ACID** properties for updates
- **Delta Lake** caches Parquet pages
  - **caching** on fast local disks
    - AWS: local NVMe 500MB/s per core (S3 cloud storage 150MB/s)
    - Azure: bigger perf difference between local and cloud storage
  - **Spark scheduler** schedules jobs with affinity
    - node that likely caches data becomes executor of queries on it
- **Delta Engine** new execution engine (for **SQL queries** only)
  - Replaces previous JIT Java compilation "Tungsten" engine
  - C++ & **Vectorized**: faster & lower-latency

Pay For What You Use

- Snowflake
  - Charged separately for S3 storage and EC2 usage
  - Data resides in Snowflake account
  - works across AWS, Azure, and Google cloud
- Redshift
  - More storage required buying more compute
  - Is gradually becoming more Snowflake-like
- BigQuery
  - Charged separately for GFS storage and TBs “processed”
- Databricks
  - Charged separately for S3 storage and EC2 usage (user account)
  - plus DBUs to Databricks (~EC2 usage)
  - works across AWS, Azure, and Google cloud

Elasticity of the various systems

- Snowflake
  - Query-level control through Virtual Warehouse mechanism
- Redshift
  - Co-located storage and compute constrains elasticity
- BigQuery (…AWS Athena is similar)
  - Serverless: Google decides for you
- Databricks
  - DB-level adjustment (cluster size) – dynamically changeable

Outline

- Part 1: Intro
- Part 2: Analytic Databases in the Cloud
  - Snowflake
  - Amazon Redshift
  - Google BigQuery
  - Databricks
- Part 3: Transactional Databases in the Cloud
  - Amazon RDS ➔ Aurora

Amazon RDS Engines

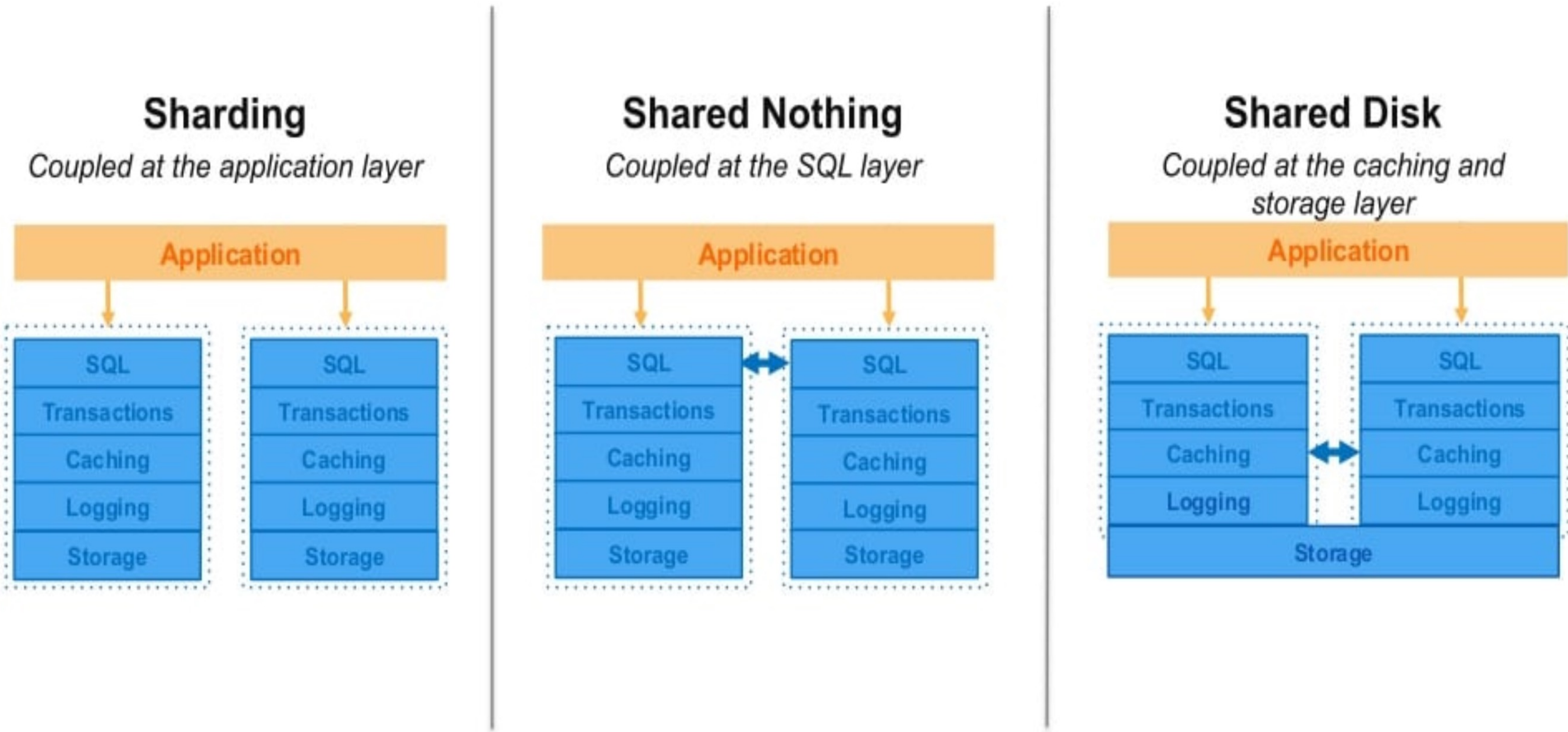
Commercial

Open source

on-premise DBMS
(monolithic architecture)

- RDS = Relational Database Service
  - EC2 instance runs DBMS; SSD or EBS storage; S3 for backup
- Scalability?
  - standard sharding, master/worker log replication

Traditional Ways To Scale a DBMS



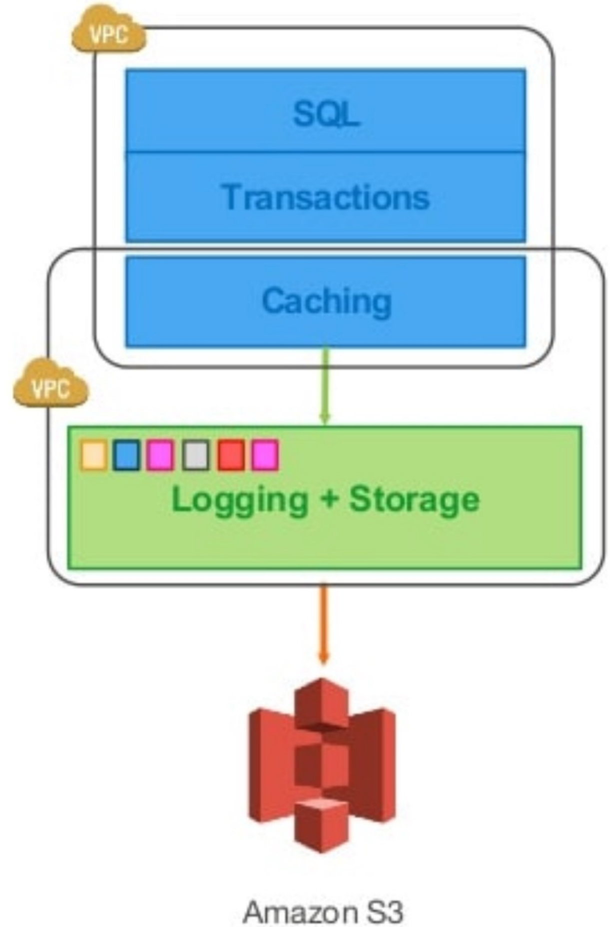
each of these approach is limited by the traditional monolithic architecture

Amazon Aurora Architecture

1. Move the logging and storage layer into a multitenant, scale-out, database-optimized storage service

2. Integrate with other AWS services such as S3, EC2, VPC, DynamoDB, SWF, and Route 53 for control & monitoring

3. Make it a managed service – using Amazon RDS. Takes care of management and administrative functions.



Microsoft now applying this architecture for SQLserver (project Socrates)

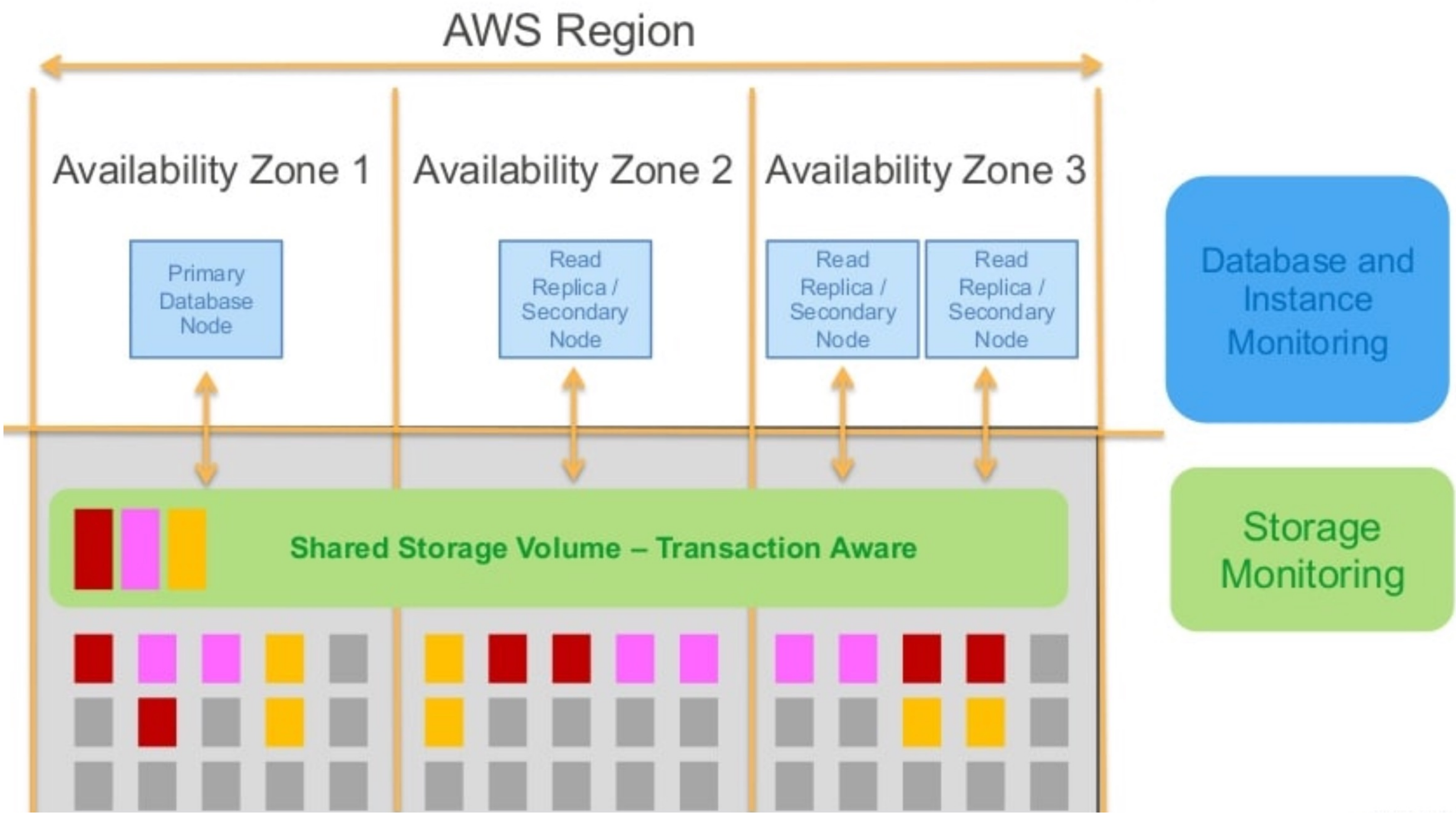
Cloud-scalable PostgreSQL. (&MySQL)

PostgreSQL 9.6 + Amazon Aurora cloud-optimized storage

- Performance: Up to 3x+ better performance than PostgreSQL alone
- Availability: Failover time of <30 seconds
- Durability: 6 copies across 3 Availability Zones
- Read Replicas: Single-digit millisecond lag times on up to 15 replicas

Amazon Aurora Storage

Scalable, Distributed, Log-Structured Storage



CWI
Centrum Wiskunde & Informatica

Aurora Instant Crash Recovery

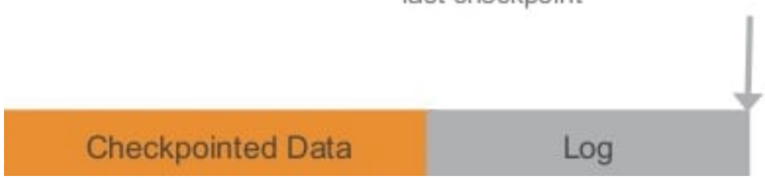
Traditional databases

Have to replay logs since the last checkpoint

Typically 5 minutes between checkpoints

Single-threaded in MySQL and PostgreSQL; requires a large number of disk accesses

Crash at T_0 requires a re-application of the SQL in the log since last checkpoint



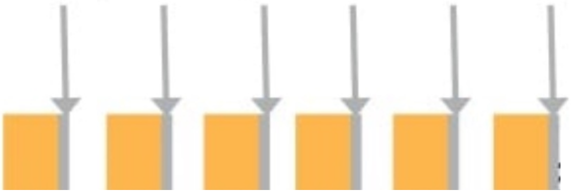
Amazon Aurora

No replay at startup because storage system is transaction-aware

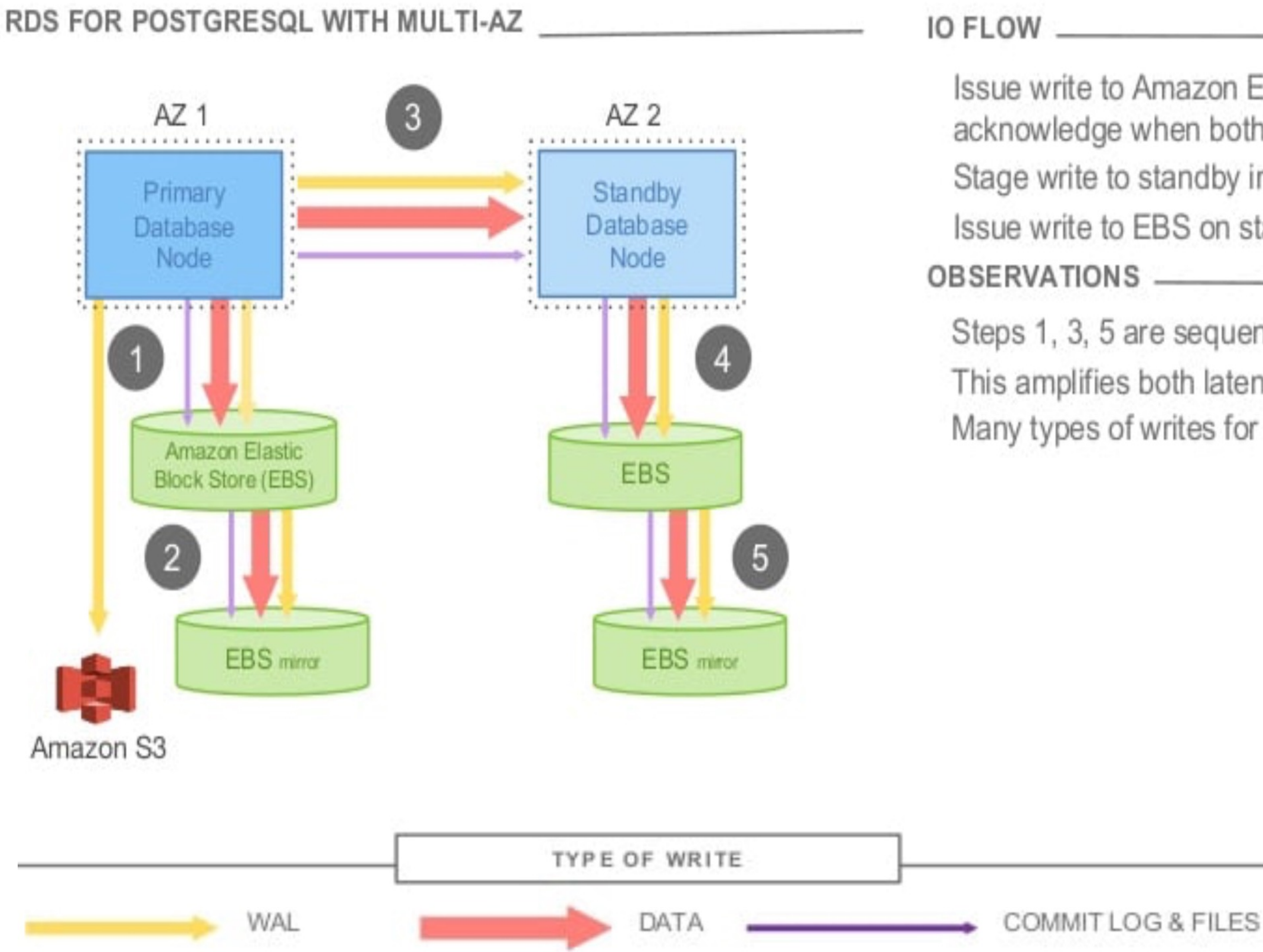
Underlying storage replays log records continuously, whether in recovery or not

Coalescing is parallel, distributed, and asynchronous

Crash at T_0 will result in logs being applied to each segment on demand, in parallel, asynchronously



Amazon RDS: Write I/O Traffic



IO FLOW

Issue write to Amazon EBS, EBS issues to mirror, acknowledge when both done

Stage write to standby instance

Issue write to EBS on standby instance

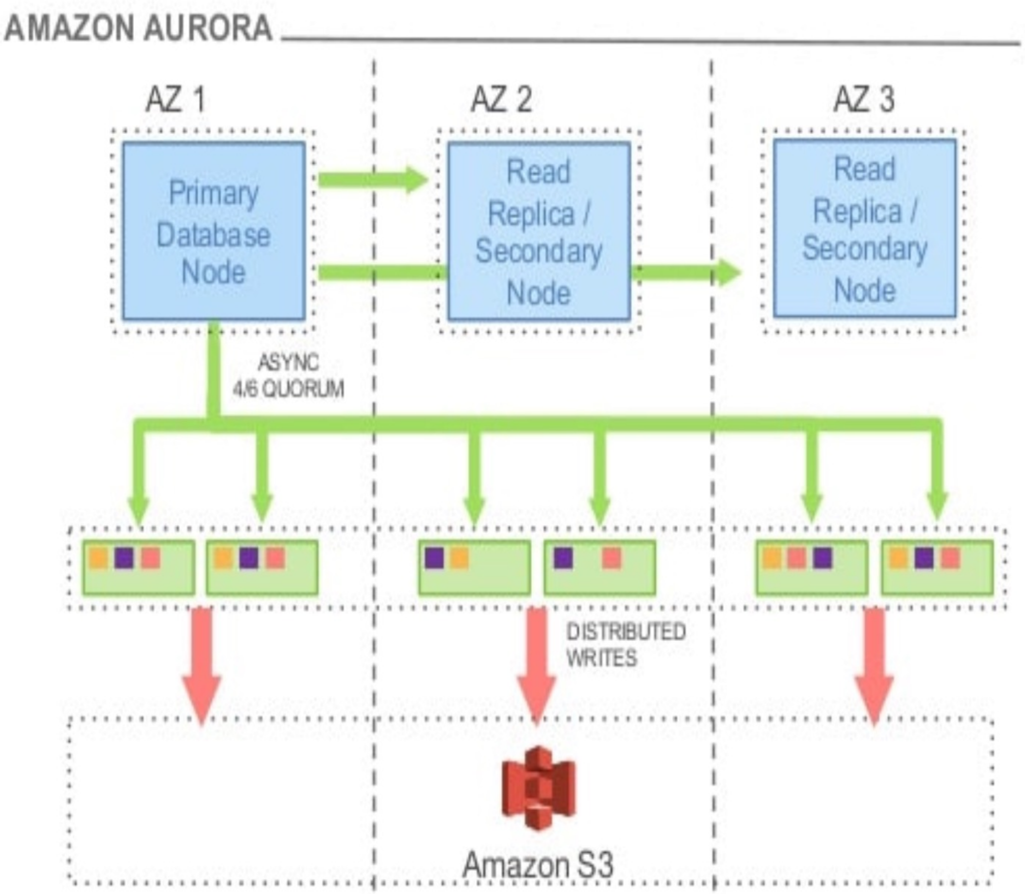
OBSERVATIONS

Steps 1, 3, 5 are sequential and synchronous

This amplifies both latency and jitter

Many types of writes for each user operation

Aurora Database Node: Write I/O Traffic



IO FLOW

Boxcar log records – fully ordered by LSN
Shuffle to appropriate segments – partially ordered
Boxcar to storage nodes and issue writes

OBSERVATIONS

Only write WAL records; all steps asynchronous
No data block writes (checkpoint, cache replacement)
6X more log writes, but 9X less network traffic
Tolerant of network and storage outlier latency

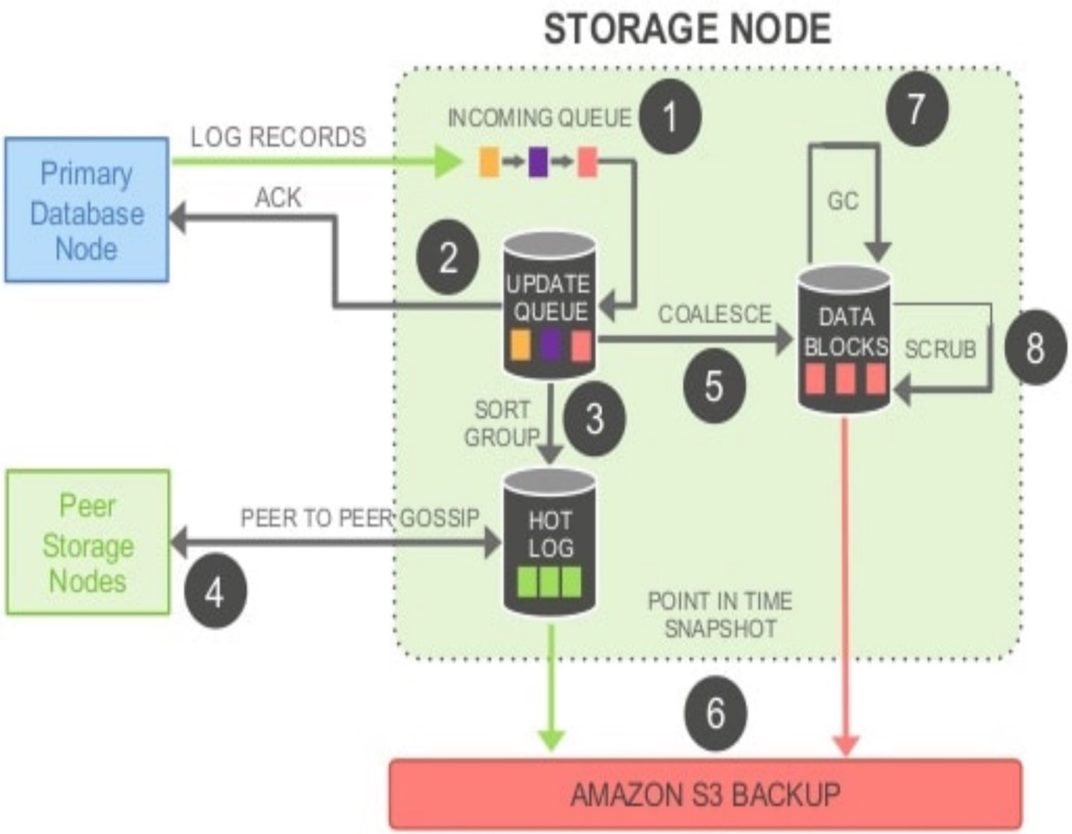
PERFORMANCE

2x or better PostgreSQL Community Edition performance
write-only or mixed read-write workloads

TYPE OF WRITE

AMAZON AURORA + WAL LOG · WAL · DATA · COMMIT LOG

Aurora Storage Node: Write I/O Traffic



IO FLOW

1. Receive record and add to in-memory queue
2. Persist record and acknowledge
3. Organize records and identify gaps in log
4. Gossip with peers to fill in holes
5. Coalesce log records into new data block versions
6. Periodically stage log and new block versions to A S3
7. Periodically garbage collect old versions
8. Periodically validate CRC codes on blocks

OBSERVATIONS

All steps are asynchronous

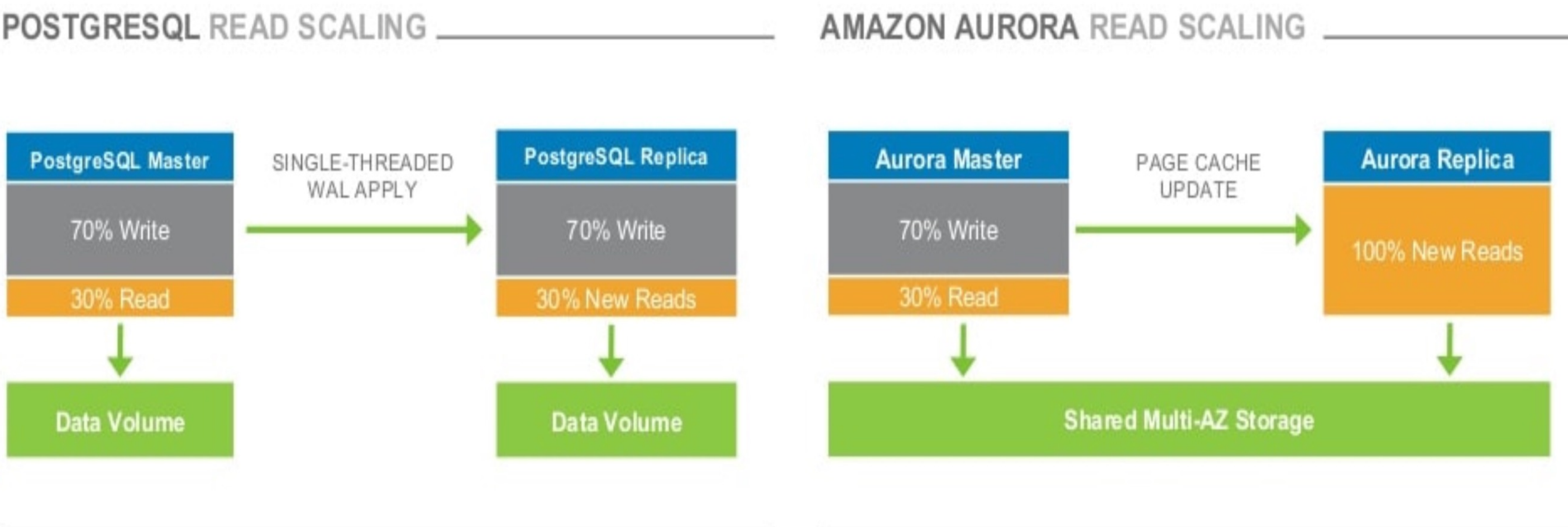
Only steps 1 and 2 are in foreground latency path

Input queue is **far smaller** than standard PostgreSQL

Favors latency-sensitive operations

Uses disk space to buffer against spikes in activity

IO Traffic in Aurora Replicas



Physical: Ship redo (WAL) to Replica

Write workload similar on both instances

Independent storage

Physical: Ship redo (WAL) from Master to Replica

Replica shares storage. No writes performed

Cached pages have redo applied

Advance read view when all commits seen

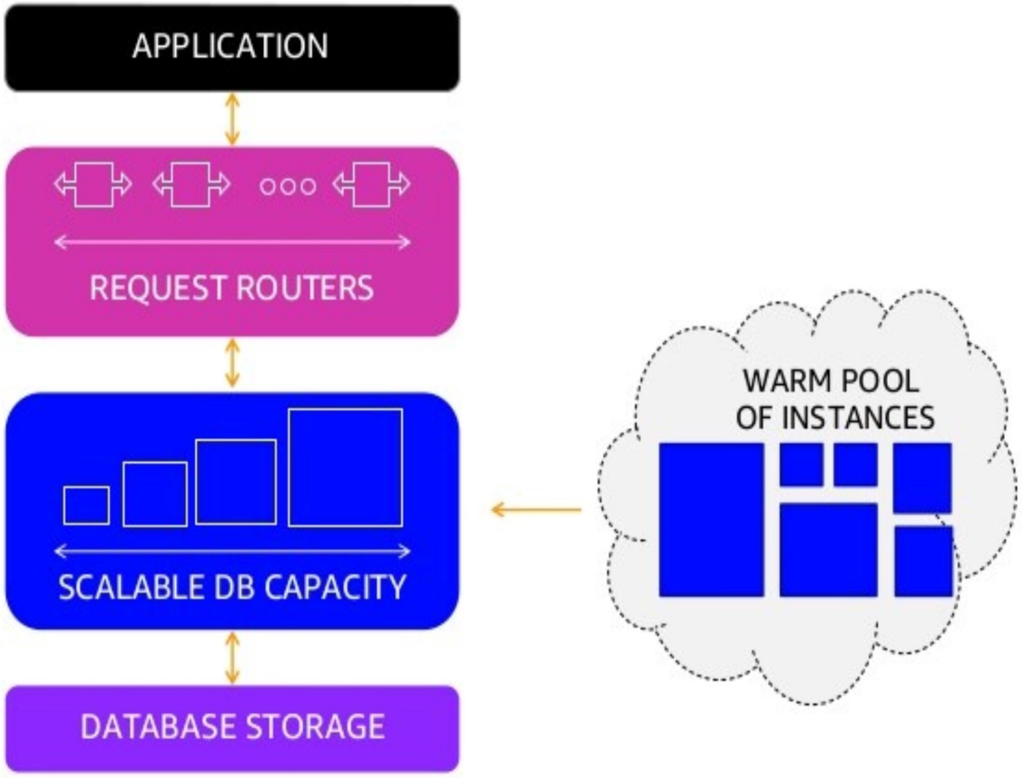
Aurora Serverless

Starts up on demand, shuts down when not in use

Scales up/down automatically

No application impact when scaling

Pay per second, 1 minute minimum



Summary

- Database systems have departed to the cloud
  - No CapEx/low OpEx, Agility, Elasticity, Cost
- Some Analytical Cloud Systems:
  - Redshift: evolved from Parallel DBMS
  - Snowflake: cloud native (virtual warehouses sharing)
  - Databricks: Spark ➔ SQL + MLFlow
  - serverless: BigQuery and Athena
    - all: Vectorized or JIT, Columnar Compressed
- Transactional Cloud Systems: Aurora architecture:
  - fleet of multi-master database nodes ➔ only write log
  - fleet of storage nodes that continuously replay logs (recovery) and create fresh data bocks