A CLOSER LOOK AT TODAY'S CLOUD DATA MANAGEMENT SERVICES

CLOUD / DATA CENTER HARDWARE ARCHITECTURES

Cloud data center architecture

- Cloud data centers are clustered in physical locations around the world, called regions.
- Within a Region, there are often several Availability Zones (AZ), each with its own redundant power and networking.

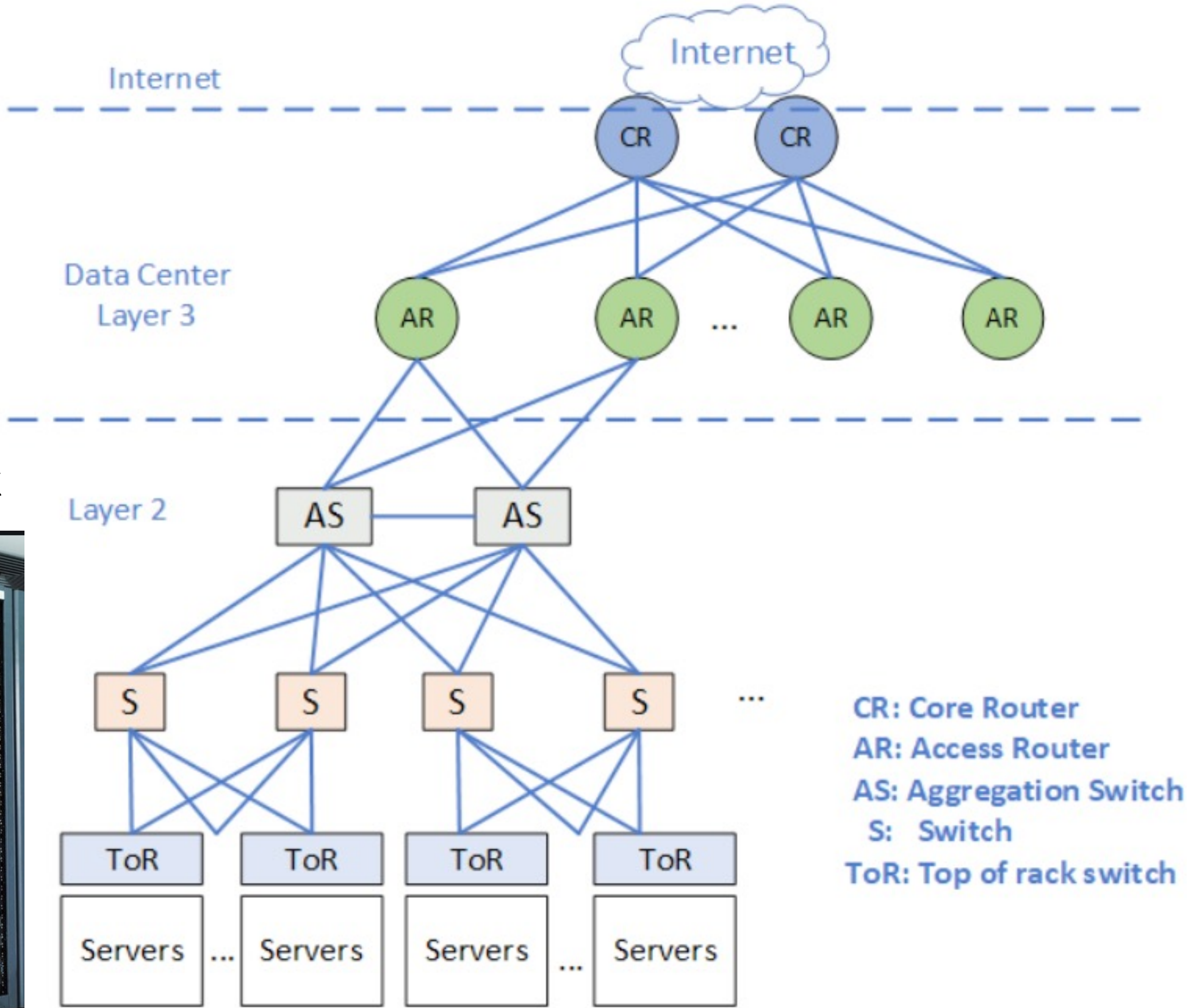
- AZs are physically separated, within a latency-defined parameter (e.g., tens of km)
- All AZ *within* a region are interconnected with high-bandwidth, low-latency network, e.g., few ms round-trip
  - Allows synchronous replication!
  - Increase protection to failure
- Latency *across* regions much higher, e.g., 100 ms

Data center servers

- A data center **<u>server</u>** commonly has
  - Two or more sockets
  - 10s of physical cores per socket
  - 100GB… few TB RAM
  - 10s of TB / local SSD
  - These numbers are constantly evolving
- One such powerful servers is rarely 100% busy with a client task!
  - Thus, multi-tenancy (see later)

On-premises (traditional) data center architecture and networking

Server rack

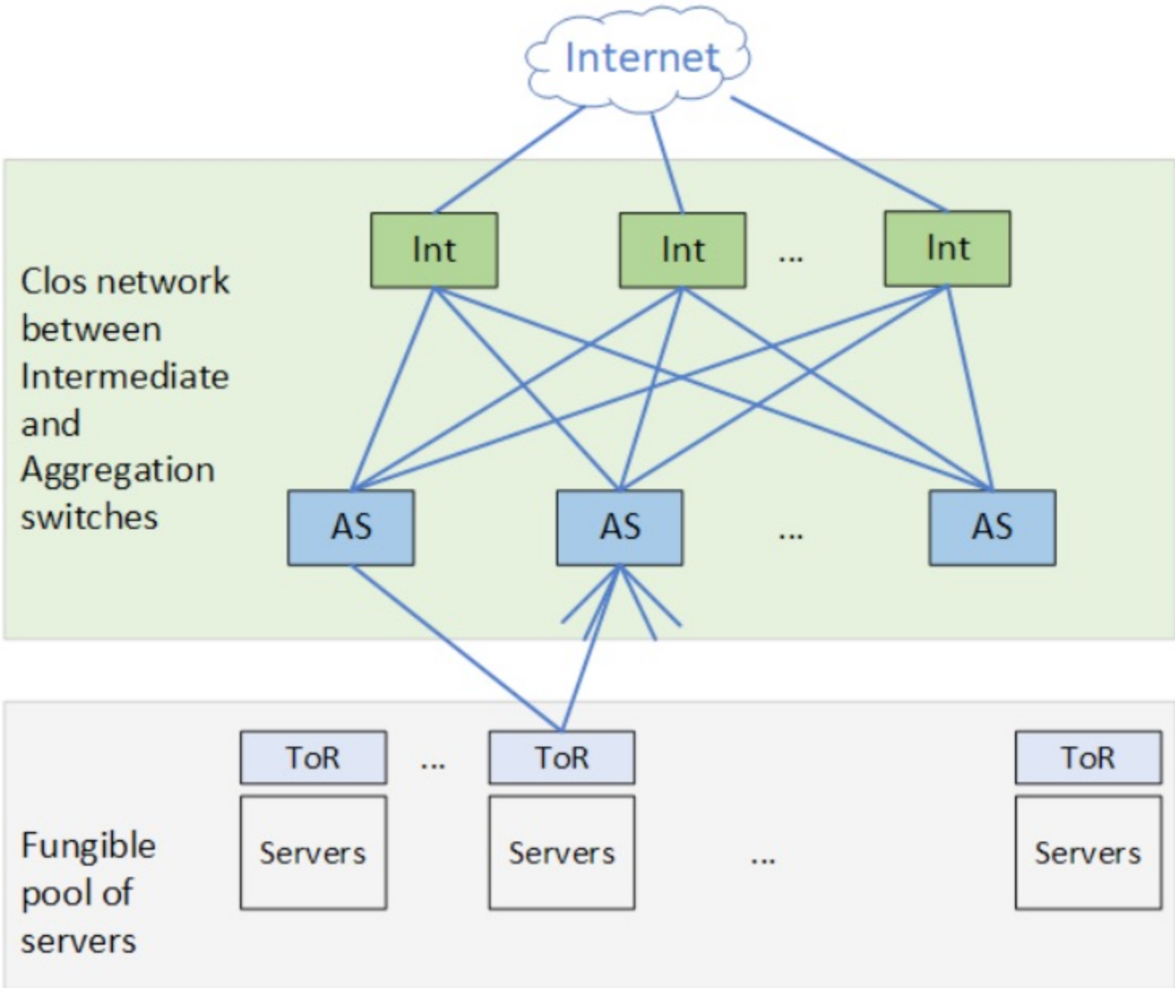


Hierarchical organization

Server-to-server bandwidth is limited

Big Data workloads need quick data transfers across servers!

Modern data center architecture and networking



Clos network (Charles Clos, 1952): network topology allowing any node to exhange data with any other

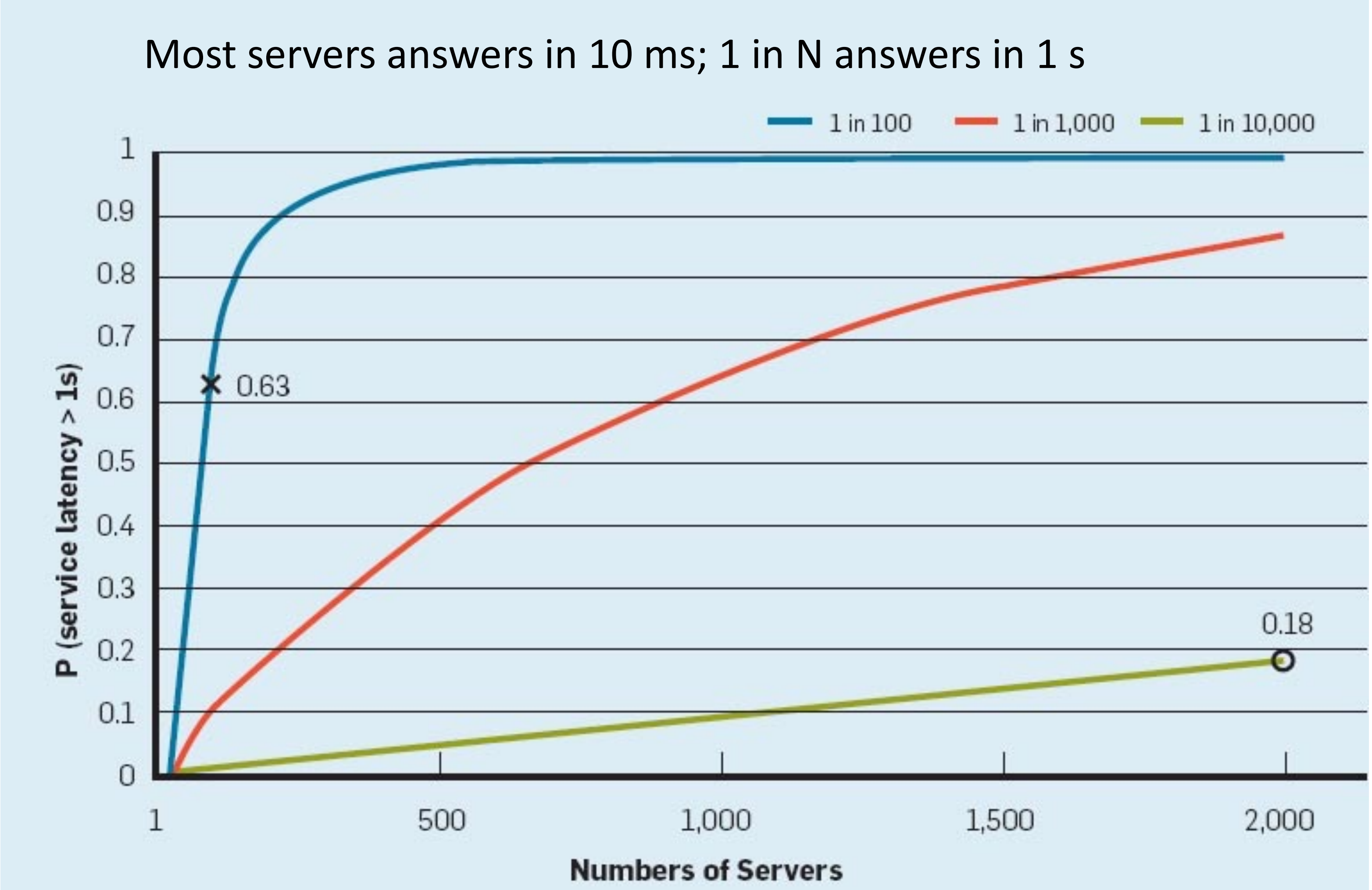
- Overhead only when connection starts (as opposed to packet-switching networks)
- Many paths between any two servers
- Extra techniques to spread traffic across paths

Int: Intermediate switch
AS: Aggregation Switch
ToR: Top of rack switch

Hardware implications

- Traditional (on-premises) data center:
  - **Storage and computing coupled** on same nodes
  - High availability and durability achieved by running multiple “hot” standby database servers
  - Efficient, but expensive! $$$
- Cloud data center
  - Sharing hardware across clients → economy of scale! $
  - File storage much cheaper than own SSDs; provides replication for durability
  - **Computation capacity decoupled from storage**, only booked when needed
  - SSD storage local to compute nodes: only as cache
  - Challenging to achieve high performance, due to network limits
  - Effective data caching crucial for performance

Latency (response time) of parallel processing across several servers



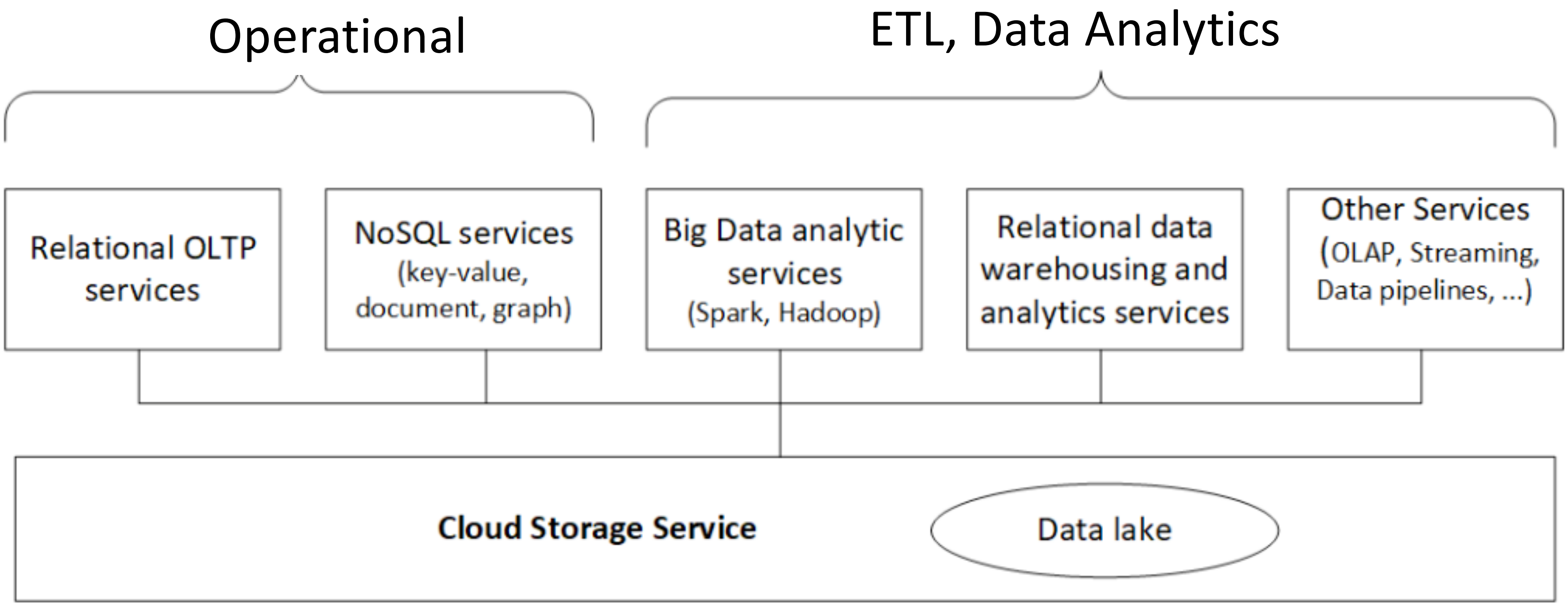
Dean and Barroso (Google), “The tail at scale”, Communications of the ACM, vol. 56 (2013)

CLOUD WORKLOAD CLASSIFICATION

Cloud database services

Services that run on hot data,
facing the users of the cloud client
High responsiveness needed

Services that run on hot and history data
Usually more data is involved
Lower responsiveness requirements



Operational cloud services

- **Relational Online Transaction Processing**
  - Transaction: modifications to the data
  - Online: must be very responsive!
  - Typical example: e-commerce

- **NoSQL workloads**: also OLTP, but on key-value-data, JSON documents, or graphs
  - Typical example: social media

ETL and Data Analytics services

ETL: extract, transform, load ("massage/pre-process" the data): for data integration; before ML…

- **Big Data Analytics services** (Spark, Hadoop)
  – Ingest & process data in a Hadoop or Spark cluster
- **Relational data warehousing & analytics**
  – E.g., analyze sales by brand, category, season, shop
- **Other** (streams, recursive processing, etc.)

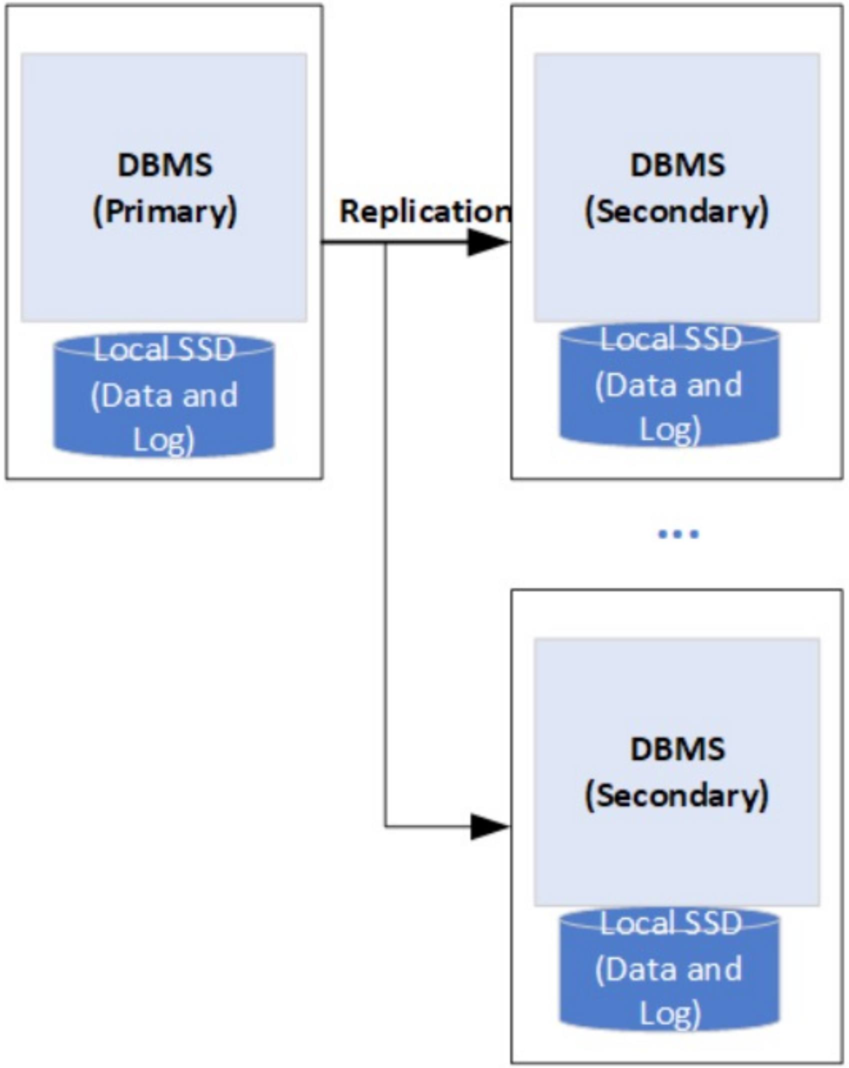
Classes not fully disjoint; active areas of research

ARCHITECTURES FOR CLOUD OLTP SERVICES

Cloud OLTP services

- Requirements:
  - High availability
  - Durability
  - Scalability with data volume
  - Controlling cost
- Two types of architectures:
  - **Coupled** storage and computing (first to appear)
  - Next generation: **decoupled** architectures

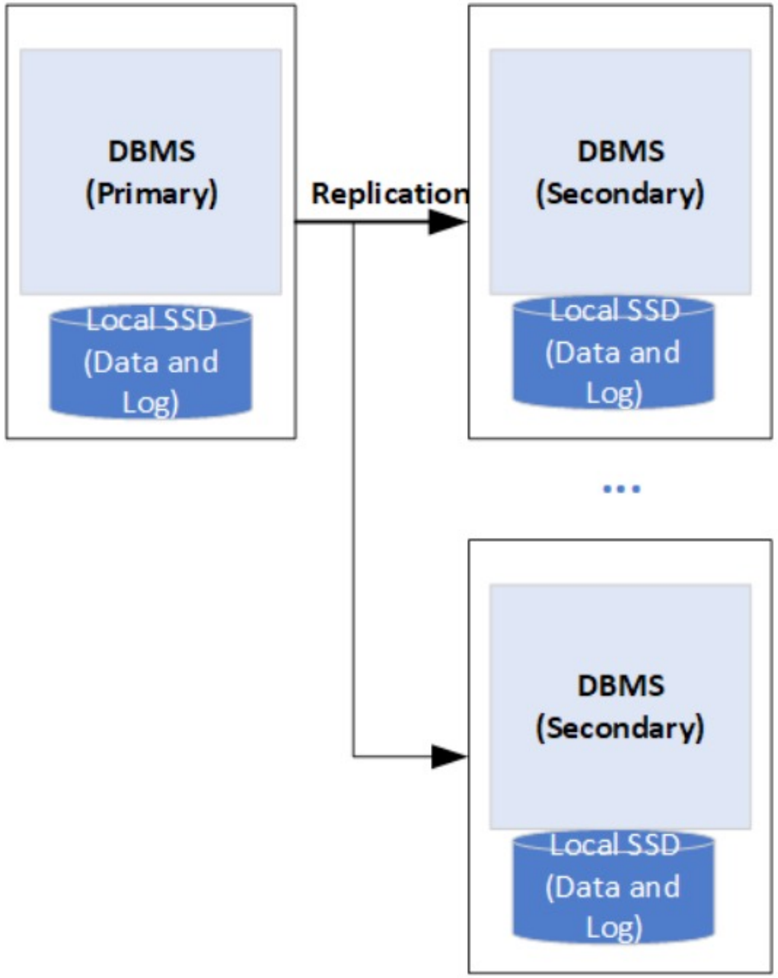
Coupled cloud OLTP architectures



- The DB runs in a **primary server**
- One or more **secondary servers** are hot replicas, in standby
- Because the servers run *transactions*, the log is also completely replicated!
- When the primary fails, *elections* designate a secondary who takes its place, then a new secondary is spawned with a copy of the data
  - For >= 99.99 availability, 3+ secondary servers
- High performance is achieved by using **SSDs** for data and log files

Azure SQL Database Business Critical
Amazon Relational Database Service (RDS)

Coupled cloud OLTP architectures



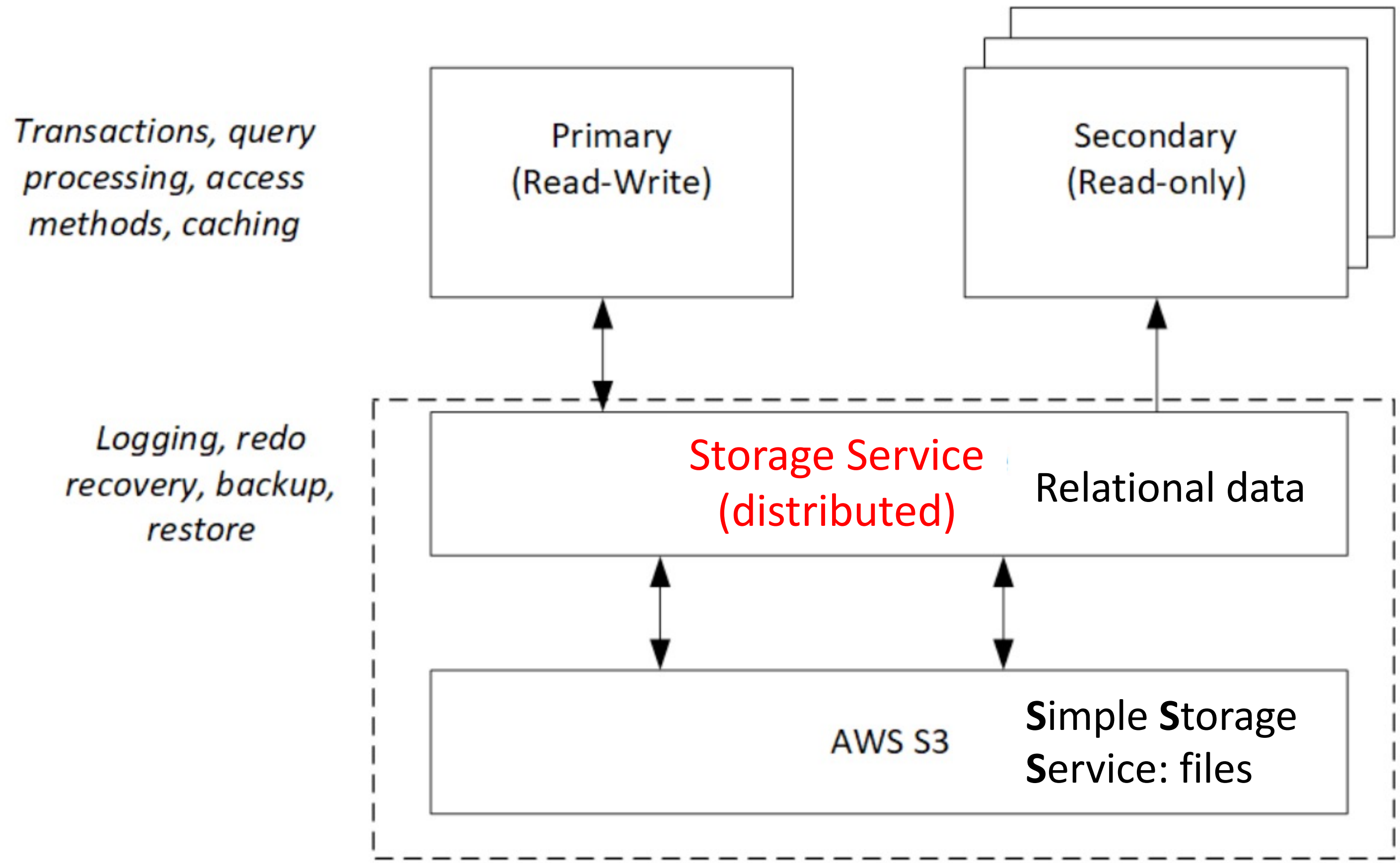
- Scalability ultimately limited by the compute and storage capacity of 1 single node (e.g., 10TB…)
  - Many businesses can fit their data in this budget.
- All primary and secondaries need full SSD storage
  - Quite high storage cost
- Some cost control by chosing how much compute resources (CPU, memory, etc.) to provision
- Smart efficient replication method (at block level, through OS, etc.)

- Some enterprise OLTP applications that require maximum performance still run this way

Disaggregated (decoupled) cloud OLTP architectures

- Decoupling:
  - Data is stored on cheap, replicated **storage server**
  - **Compute servers** are allocated on demand
  - Storage and computation can *independently* scale out
  - The entire database is no longer available on each compute node → aggressive **caching** is needed to offset the latency of data access!
- AWS (Amazon Web Services) Aurora, Azure SQL Hyperscale, Google Cloud Spanner

AWS Aurora (Amazon)



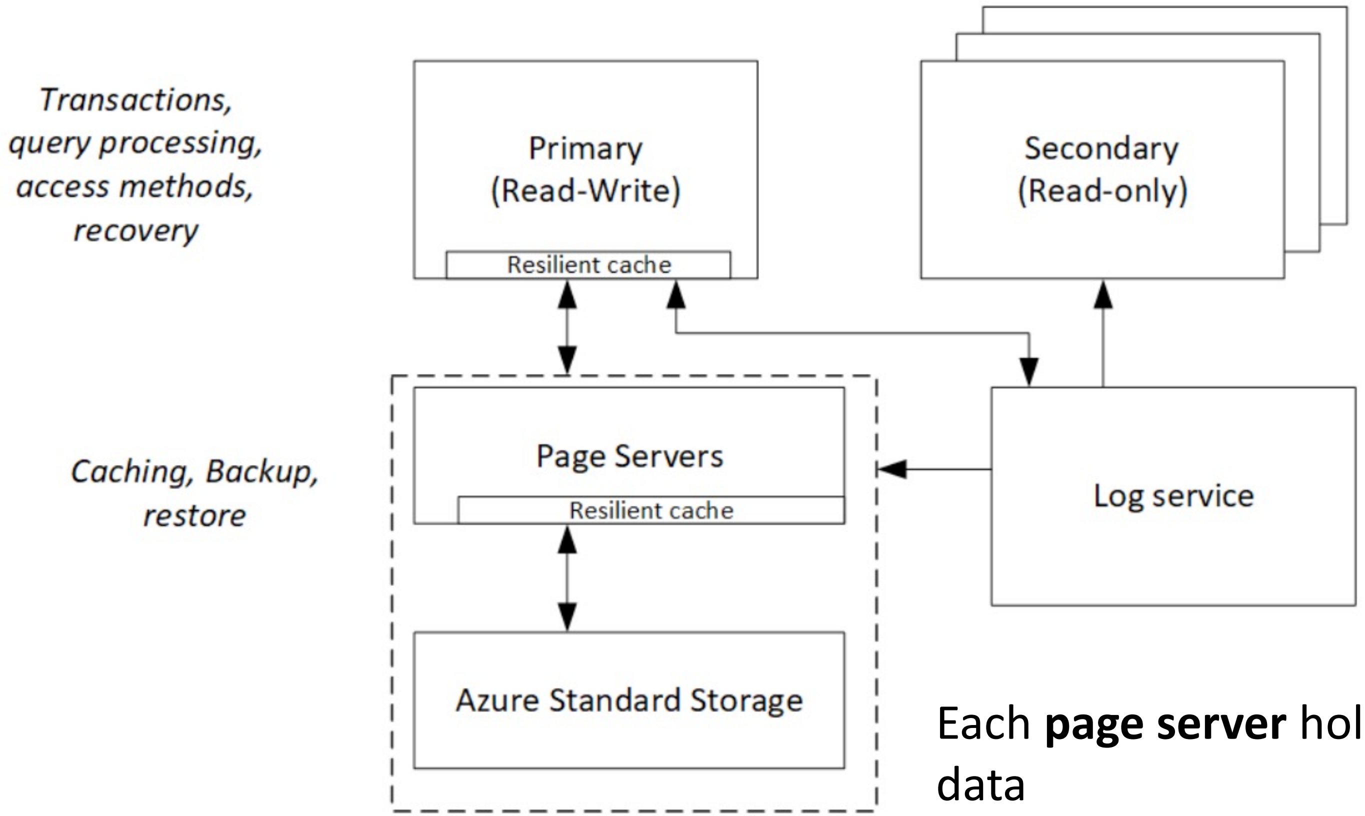
The storage service **replicates** the data across multiple AZs for high availability

The storage service continuously applies log records on all the secondary replicas to **keep them up to date**.

When a compute node requests a page, the storage service returns the current version of the page.

SSD cache on compute and storage service nodes.

Azure SQL Hyperscale (Microsoft)



Log handled separately by dedicated **log service**

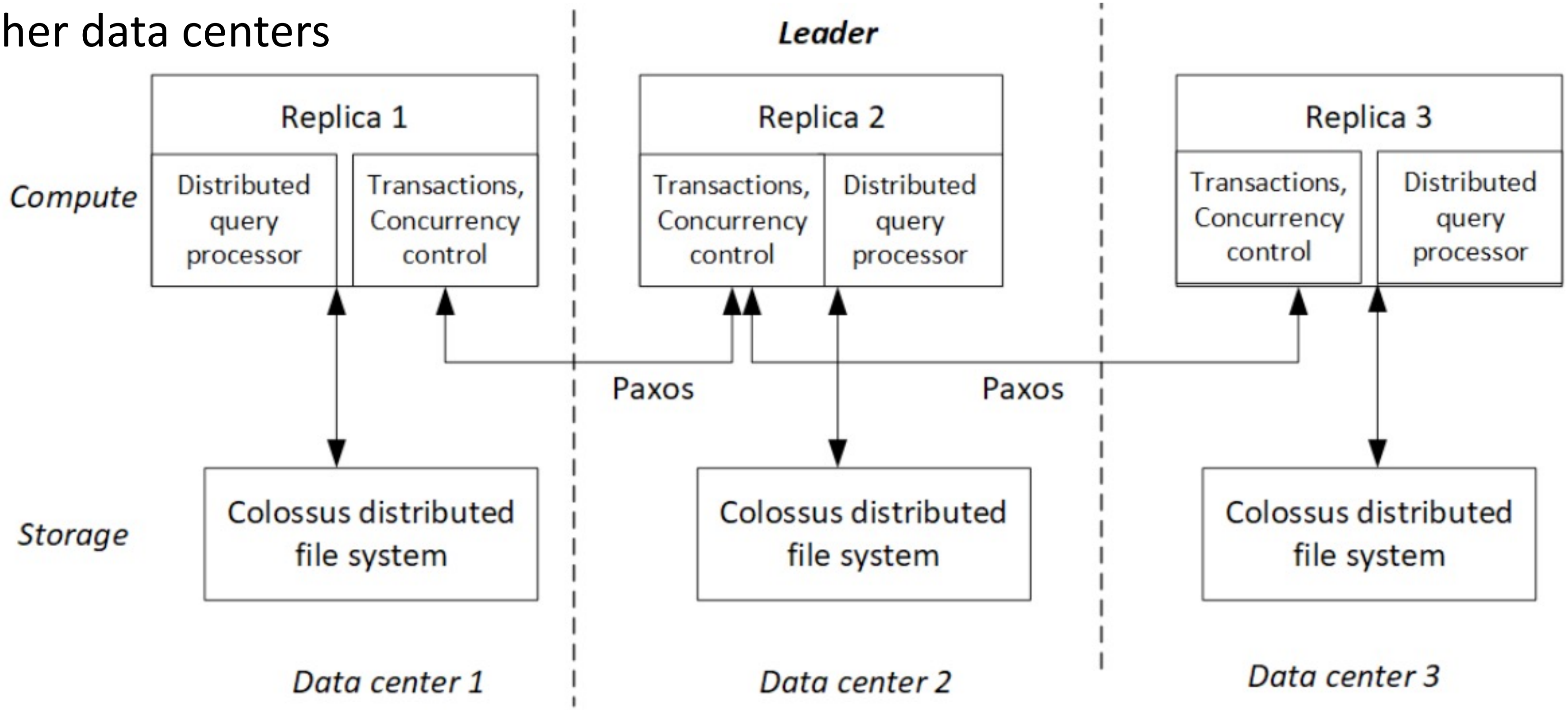
- Multiple replicas + quorum-based protocol for reconciliation
- Keeps recent log records in memory (likely to be neded again), across multiple nodes
- Older log records moved to secondary storage

Each **page server** holds a copy of a partition of the data

- Answers compute nodes' page requests
- Caches the partition on local SSD, warm data in memory
- Checkpoints the data and creates backups

Cloud Spanner (Google)

- **Shared-nothing** architecture, based on append-only Colossus distributed file system
- Each table is **sharded** across a data center, then **replicated** for high-availability in other data centers



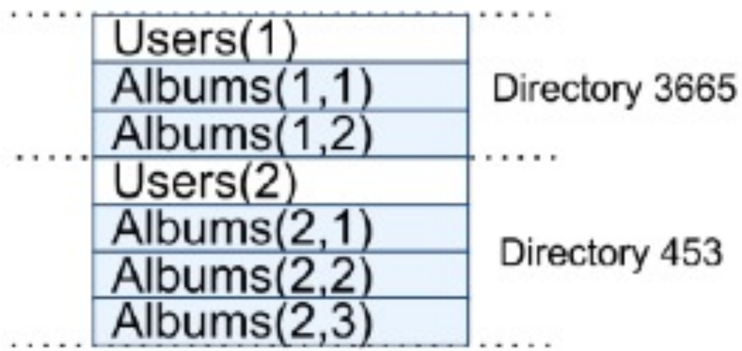
- Transactions use a replicated write-ahead redo log (WAL)
- Paxos consensus algorithm used to reconcile log content.

Spanner tables

- Each table has a **primary key** (one or more attributes)
- Tables can be organized in **hierarchies**
  - Tables whose primary key **extends the key of the parent** can be stored **interleaved** with the parent
  - Example: photo album metadata organized first by the user, then by the album

```
CREATE TABLE Users {
  uid INT64 NOT NULL, email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
  uid INT64 NOT NULL, aid INT64 NOT NULL,
  name STRING
} PRIMARY KEY (uid, aid),
  INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```

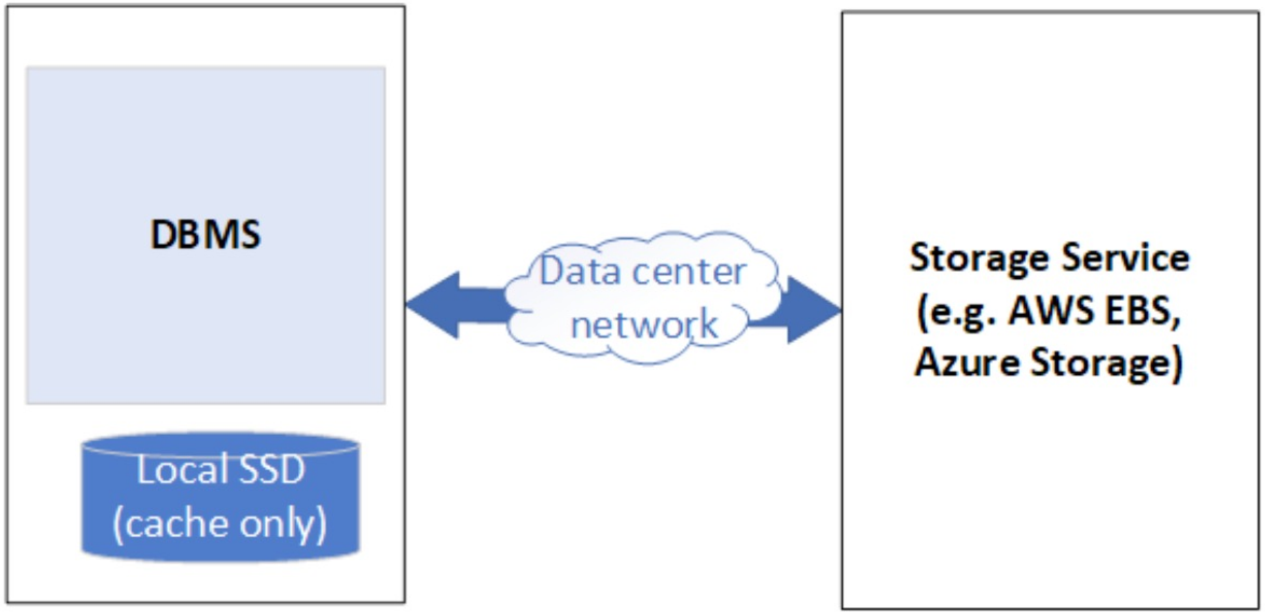


Spanner query processing

- ❑ Distributed SQL query processing engine
- ❑ Optimization such as:
  - ❑ Pruning partitions that are not relevant for a given query
  - ❑ Key-foreign key joins exploiting shard colocation…
- ❑ If a node fails during query processing, the query is automatically restarted
  - ❑ Simplifies application development
  - ❑ Allows to handle node upgrades

Low-cost cloud architectures

- Low-cost = low performance



- Run 1 DBMS attached to storage and log on (slow) inexpensive storage
- Azure SQL Database General Purpose
- Failure → DBMS restart (after downtime)

ARCHITECTURES FOR DATA ANALYTICS SERVICES

Data Analytics services in the cloud

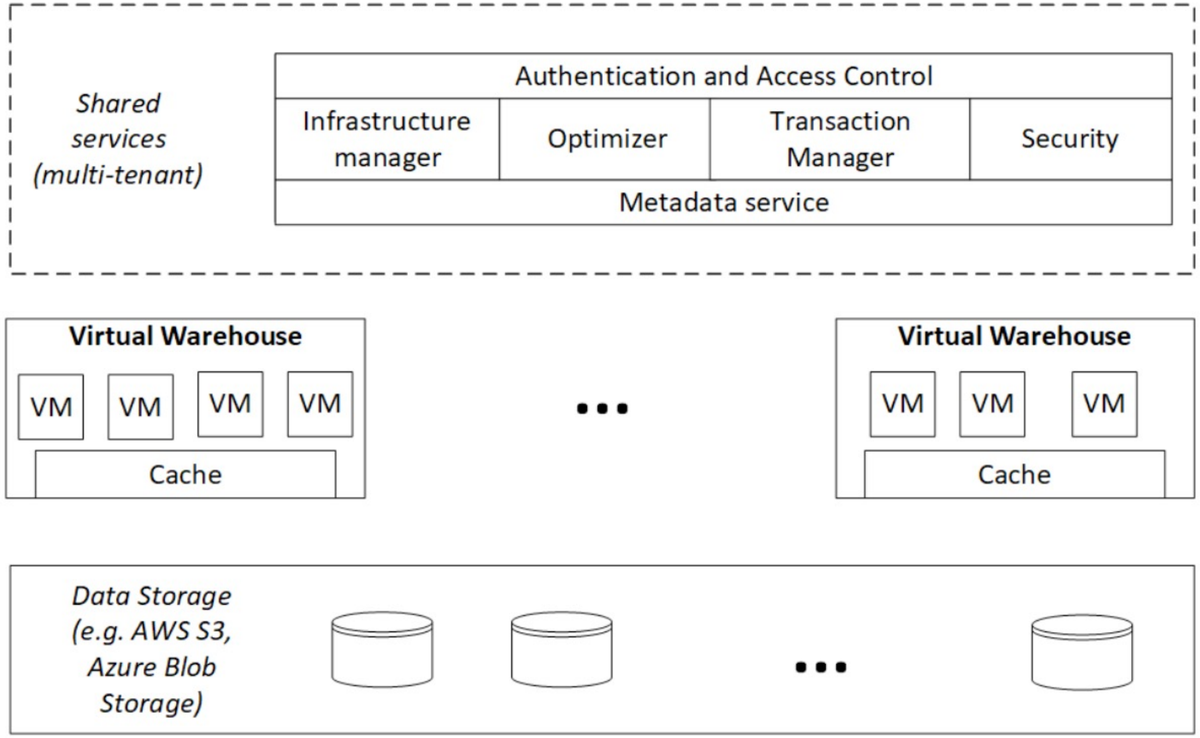
- **Data warehousing (DW)**
  - Data is *loaded before it can be queried*
  - Performance optimizations enabled by indexes, materialized views, data partitioning
- **Big Data Analytics** services allow analyzing data residing in a storage subsystem, e.g., HDFS on premises, or blog storage in the cloud
  - *No need to load the data in advance*
  - Typically much cheaper, much larger scale than DW
  - Heterogeneous data sources: data lake

Dimensions of Cloud Data Analytics services in the cloud

1. Shared nothing vs. shared data
2. Programming API: SQL vs. MapReduce
3. Pre-loaded data vs. in-situ querying
4. Interactive vs. batch querying
5. Sophistication of the query optimizer

DW cloud service: Snowflake

Shared data in a remote storage; SQL API; interactive querying
Pre-loaded data (and *statistics* computed for each partition during loading, managed by the metadata service, in particular for query optimization)



Each **virtual machine (VM)** is a complete database

The VM caches data on local SSD

A **Virtual Warehouse (VW)** is used by 1 client; scale up by adding VMs

No indexes (bad for queries; simplifies transaction processing)

Query evaluation in Snowflake

1. Selective data access
   - Each table is stored as as set of **shards**
   - Inside each shard, data is stored **as a set of (compressed) columns**
   - **Headers** built for each column within the shard
     - Minimum and maximum values
     - No need to read a shard if the query predicate is incompatible with the header information

2. Query optimizer
   - Cost- and statistic-based
   - Headers computed even on intermediary results
   - Some decisions taken at runtime

3. Intermediary query results written in node local disks, then (if needed) to S3

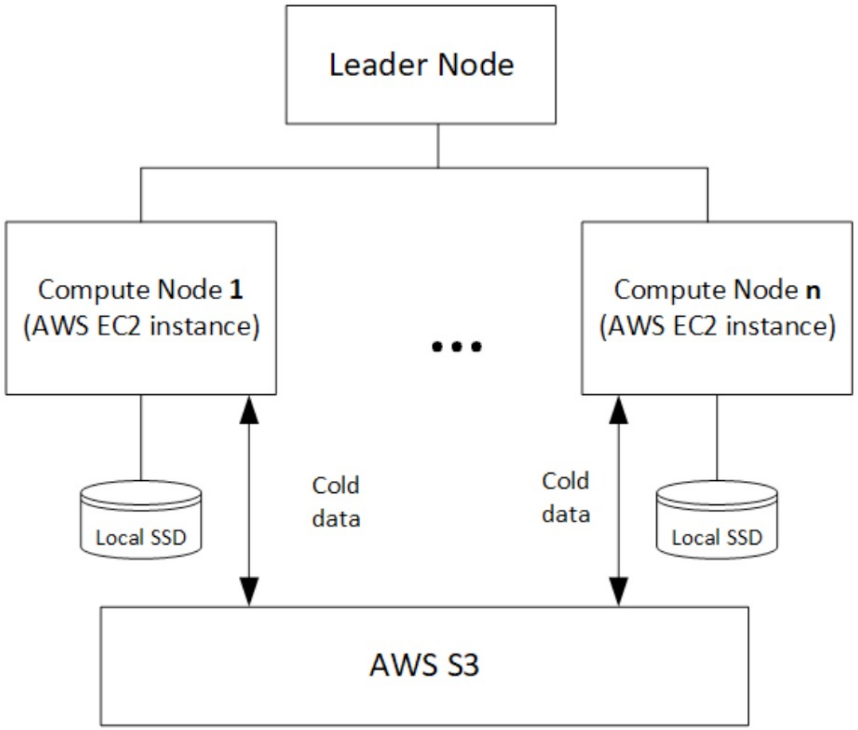
Concurrency control in Snowflake

- Handled globally using fine-granularity data store
- An update creates a new version of a table (multi version concurrency control, MVCC): no finer-granularity update
- Each version has a timestamp
- Possible to explicitly query *the version at or after a certain timestamp*
- Each version stays available 90 days after deletion

DW cloud service: AWS Redshift

Shared-nothing; SQL API; pre-loaded data; interactive querying

Shared-nothing MPP architecture



Cluster = 1 **leader** + n **compute** nodes

Leader coordinates query exec.

A cluster hosts databases (sets of tables).

A table can be:

- **Distributed** across the compute nodes by specifying a distribution key
- **Replicated** to all the compute nodes

Efficient scale-up is difficult since adding nodes requires redistributing the data (costly!)

Recent optimizations: automatic move of cold data to S3, to reduce costs

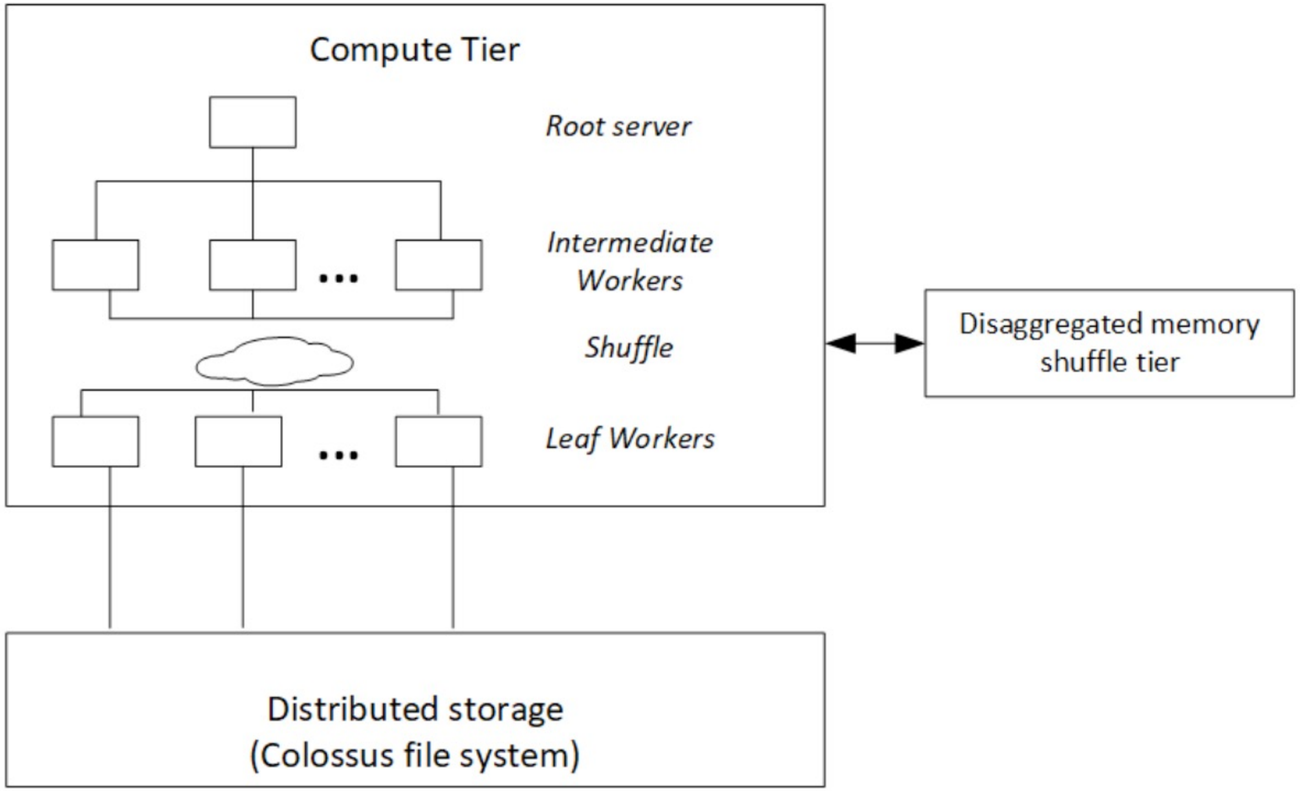
DW cloud service: Google BigQuery

SQL dialect on *nested* relational data

Data either pre-loaded or processed from files

Efficient column-oriented format (Capacitor)

Data automatically sharded and loaded in Colossus



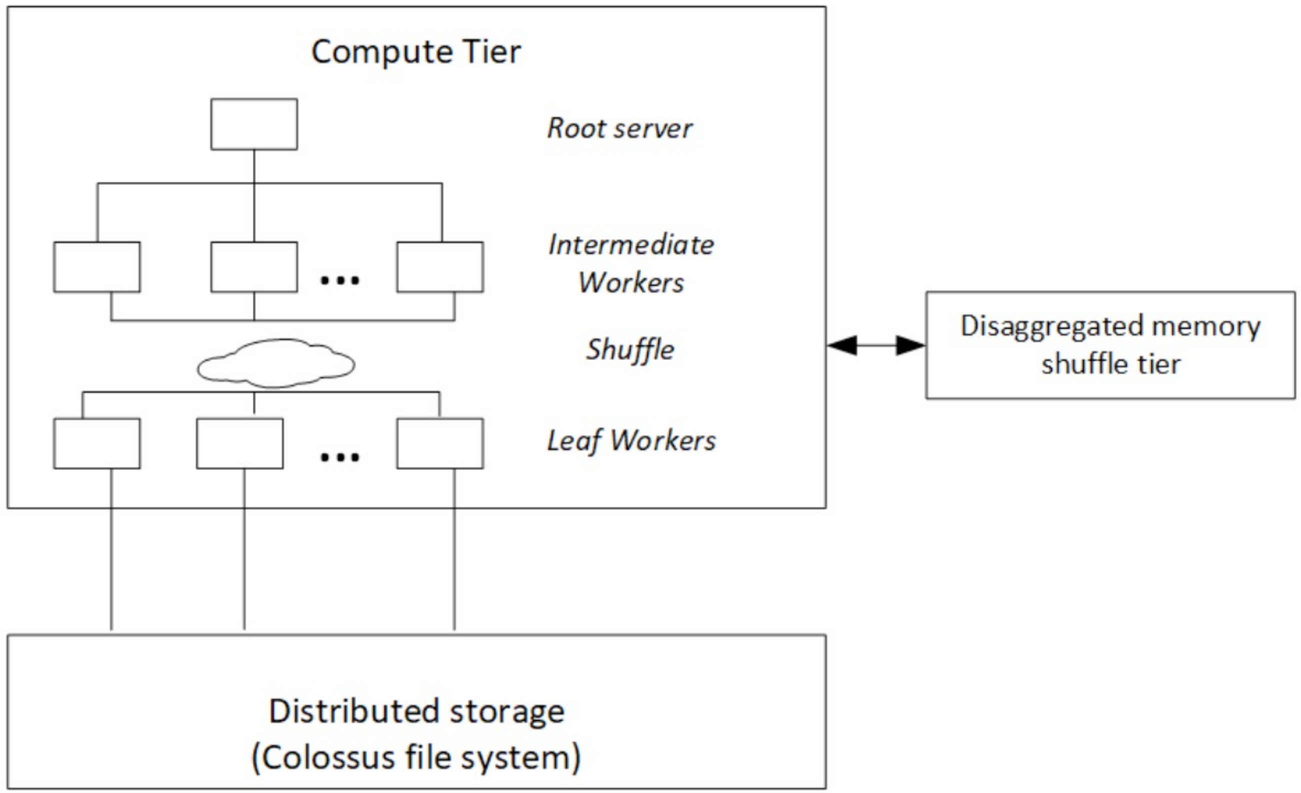
Query processing in Google BigQuery

Started with **1-table queries** over large sharded tables

- Irrelevant partition skip
- Skip indexes to read only part of a partition

Added **distributed joins** → shuffle!

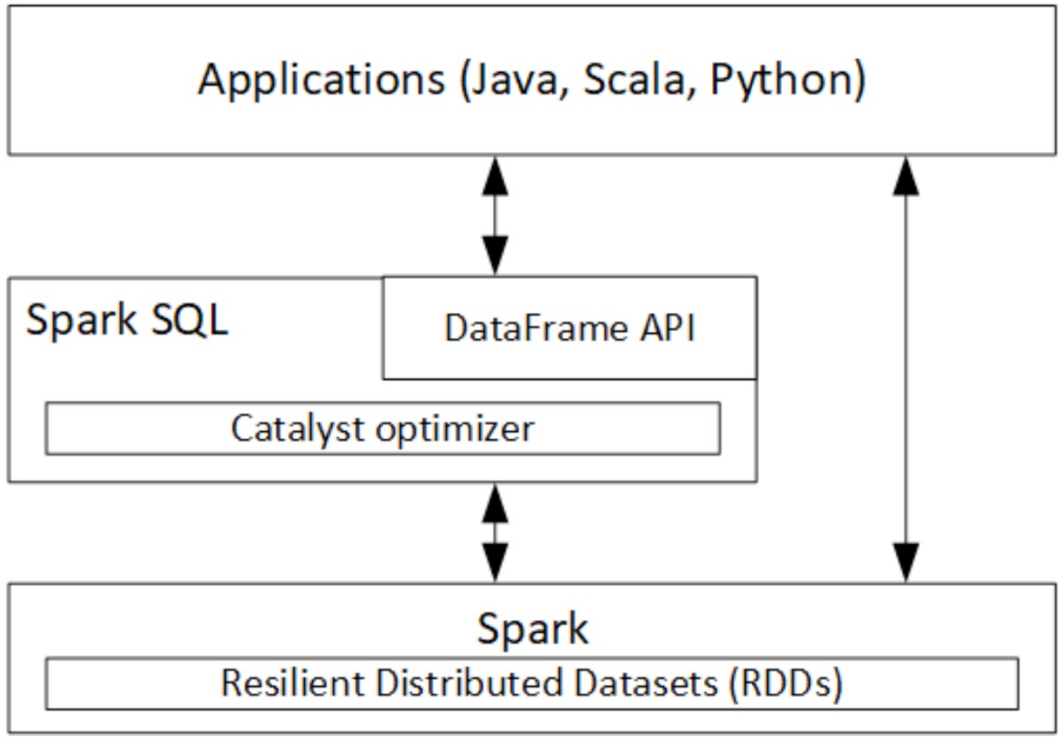
- Distributed, efficient transient storage for the shuffled data (~ memory!)
- Serves also as checkpoint
- More flexibility for scheduling queries



DW cloud service: Spark

Spark:

- ❑ Shared-data (distributed file system, e.g., HDFS, or cloud, e.g., AWS S3 or Azure blob)
- ❑ MapReduce API
- ❑ Batch-oriented
- ❑ Programming model based on RDD
- ❑ Spark SQL: SQL extra layer



Spark: brief overview

- Extremely popular Big Data management framework
- Main concept: Resilient Distributed Datasets (RDD) until v2.0; then just **Dataset** (more optimizations supported)
- A Dataset can be created from a (distributed) file, or through processing. Sample snippets using pySpark:

```
>>> textFile = spark.read.text("README.md")
>>> textFile.count() # Number of rows in this DataFrame
126
>>> textFile.first() # First row in this DataFrame
Row(value=u'# Apache Spark')
>>> linesWithSpark = textFile.filter(textFile.value.contains("Spark"))
>>> textFile.filter(textFile.value.contains("Spark")).count()
15
```

Spark programming

Apache Spark 3.3.0

- Extremely popular Big Data management framework
- Main concept: Resilient Distributed Datasets (RDD) until v2.0; then just **Dataset** (more optimizations supported)
- A Dataset can be created from a (distributed) file, or through processing. Sample snippets using pySpark:

```
>>> textFile = spark.read.text("README.md")
>>> textFile.count() # Number of rows in this DataFrame
126
>>> textFile.first() # First row in this DataFrame
Row(value=u'# Apache Spark')

>>> linesWithSpark = textFile.filter(textFile.value.contains("Spark"))
>>> textFile.filter(textFile.value.contains("Spark")).count()
15
```

Could be distributed!

Dataset transformation

Aggregation

Spark: more complex programming

A Dataset can be created from a (distributed) file, or through processing. Sample snippets using pySpark:

```
## Find the row having the most words:
>>> textFile.select(size(split(textFile.value, "\s+")).name("numWords"))
              .agg(max(col("numWords"))).collect()
[Row(max(numWords)=15)]

## Compute the frequencies of all words, MapReduce style:
>>> wordCounts = textFile.select(explode(split(textFile.value, "\s+"))
                         .alias("word")).groupBy("word").count()
>>> wordCounts.collect()
[Row(word=u'online', count=1), Row(word=u'graphs', count=1), ...]

>>> linesWithSpark.cache()
```

Explicit cache control

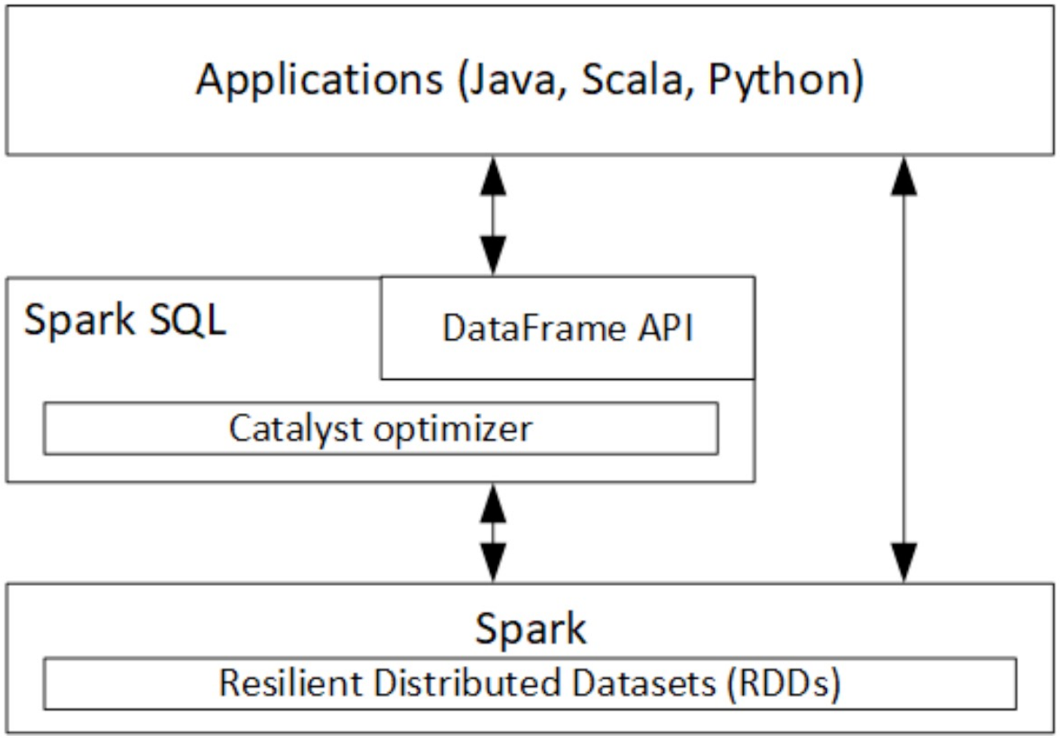
Spark optimizer: Catalyst

Optimizes users' queries for massively parallel processing

0. Use **cached** Datasets,
if possible

1. **Rule-based optimizations**:
push selections,
projections,
transitive equalities,
etc.

2. **Cost-based optimization**

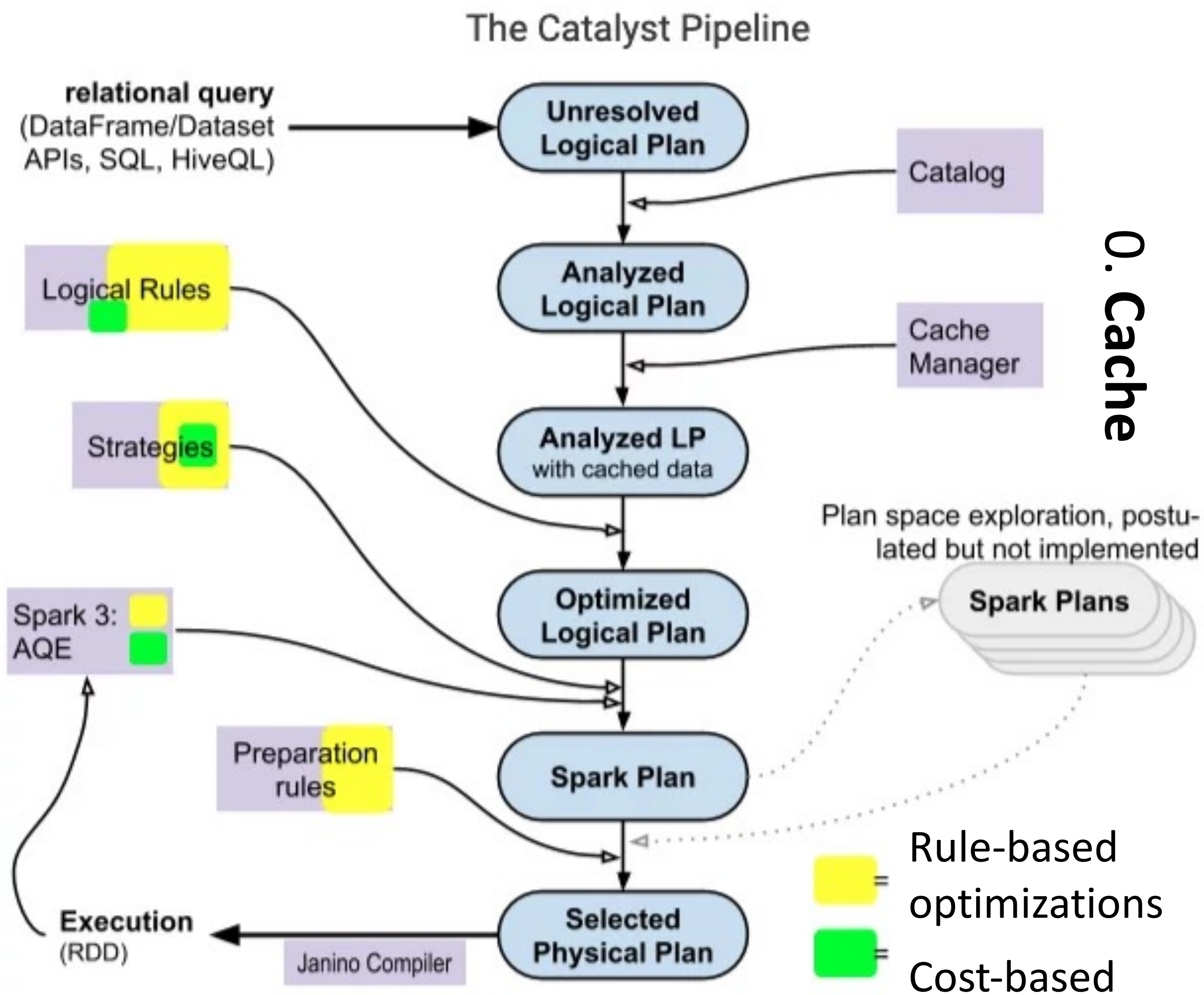


Spark optimizer: Catalyst

Apache Spark 3.3.0

1. **Rule-based optimizations**: push selections, projections, transitive equalities, etc.

2. **Cost-based optimization**



0. Cache

Rule-based optimizations

Cost-based optimizations

https://www.unraveldata.com/resources/catalyst-analyst-a-deep-dive-into-sparks-optimizer/

PRICING AND SLA:
FINANCIALS OF CLOUD SERVICES

What does the bill look like? Pricing models

Storage costs by far dominated by **compute costs**, cost discussion mostly focused on the latter

Two main classes of pricing models

- **Provisioned capacity**
  - The client books a set of compute nodes and keeps them always on, whether or not they are used
- **On demand** (aka **serverless)**
  - Clients only book the resources they need and release them when the work is finished

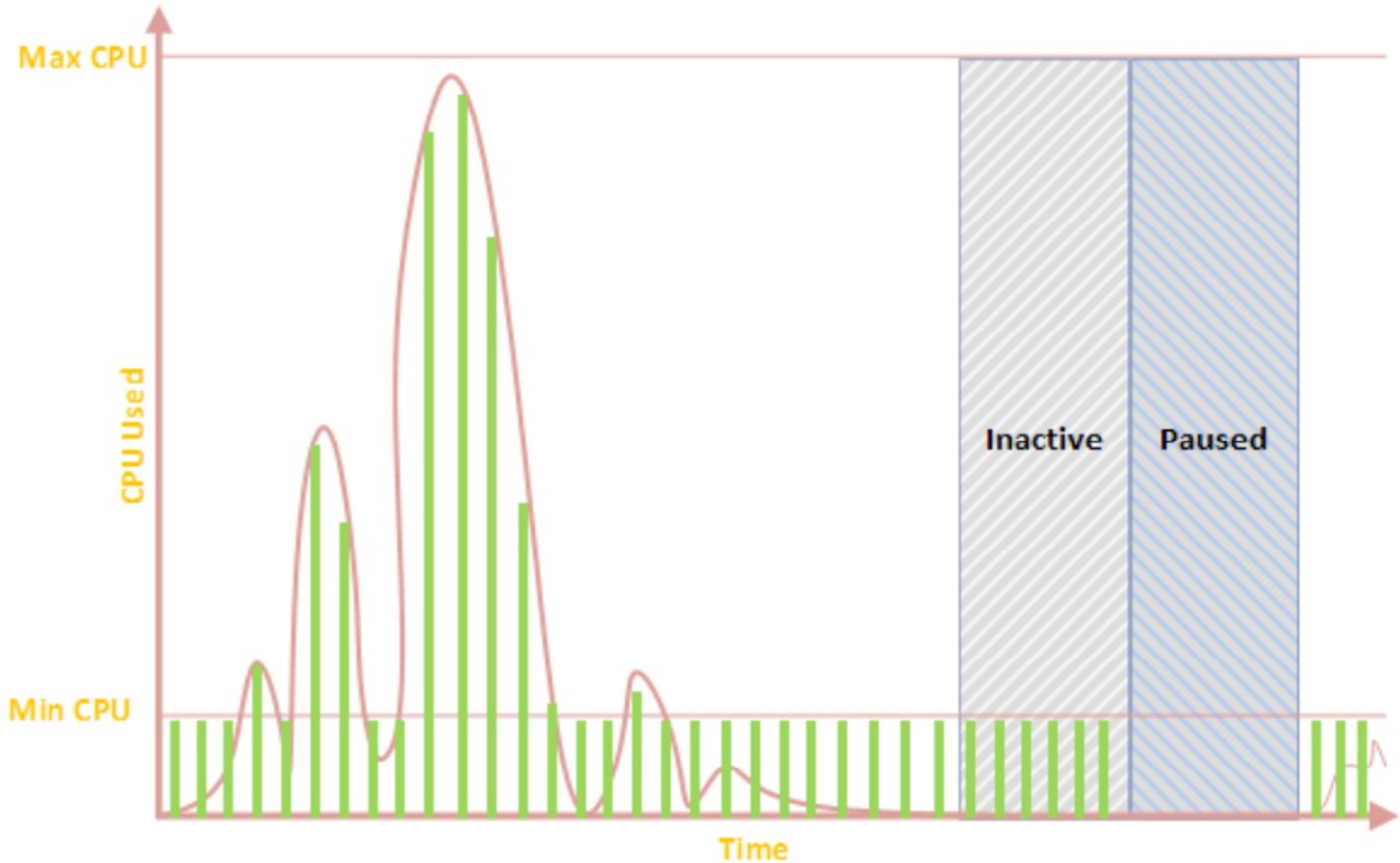
What am I paying for? Quality of Service (QOS) guarantees

- Also called **Service Level Agreement (SLA)**
  - The service level is described by a set of metrics, aka **Key Performance indicators (KPIs),** aka Service-level indicators
- A **Service-Level Objective (SLO)** is a target value or range for a KPI
  - E.g., “availability>=99.99%” for expensive nodes, or “availability>=99.9%” for less expensive ones
  - “2 nines” (10^{-2} unavailability) vs “1 nine”
- Metrics and SLOs are checked internally at every new release or proposed evolution of a product
- An SLO contractually promised to a client is an SLA

Resource-level SLAs

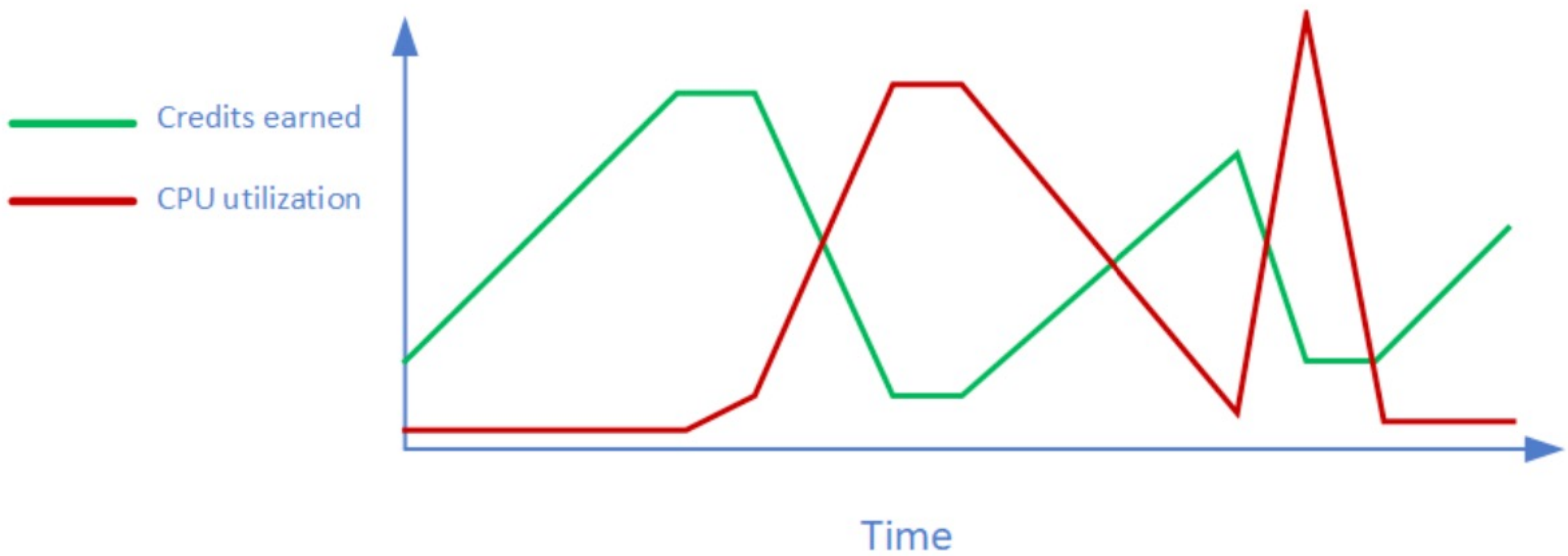
- **Fixed resource SLA**: fixed promises made to tenant (=cloud service user)
- **Min-Max SLA**:
  - A minimum amount of resources are guaranteed to every database + an upper limit per database
  - Once all the databases have received the minimum, the remaining capacity is allocated according to some policy, e.g., a weight of each database

Example: pricing model in Azure SQL database serverless



Resource-level SLAs

- **Burstable SLA**
  - Tenants are given credits per time when they do <u>not</u> run
  - Tenants spend credits by running tasks
  - Appropriate for low-average, bursty workflows, e.g., testing



Pricing incentives

How to make sure cloud capacity is never wasted?

- Make **reserved instances** cheaper to encourage long bookings.

- **Spot prices**:
  - The cloud provider publishes a price updated every 5 minutes
  - Tenants **bid** on how much they are willing to pay
  - If the bid exceeds the price, the VM is allocated immediately
  - Spot priced instances can be 90% cheaper; terminated by the service provider
  - Appropriate for short-lived tasks, when the loss of work in case of termination is not problematic

- **Pre-emptible compute**: cheaper, e.g., by 80%, but could be stopped by the provider with 30 sec notice to save work

Sample cost-service trade-offs in the cloud

Table 1: Selection of EC2 instances (June 2020, us-east-1)

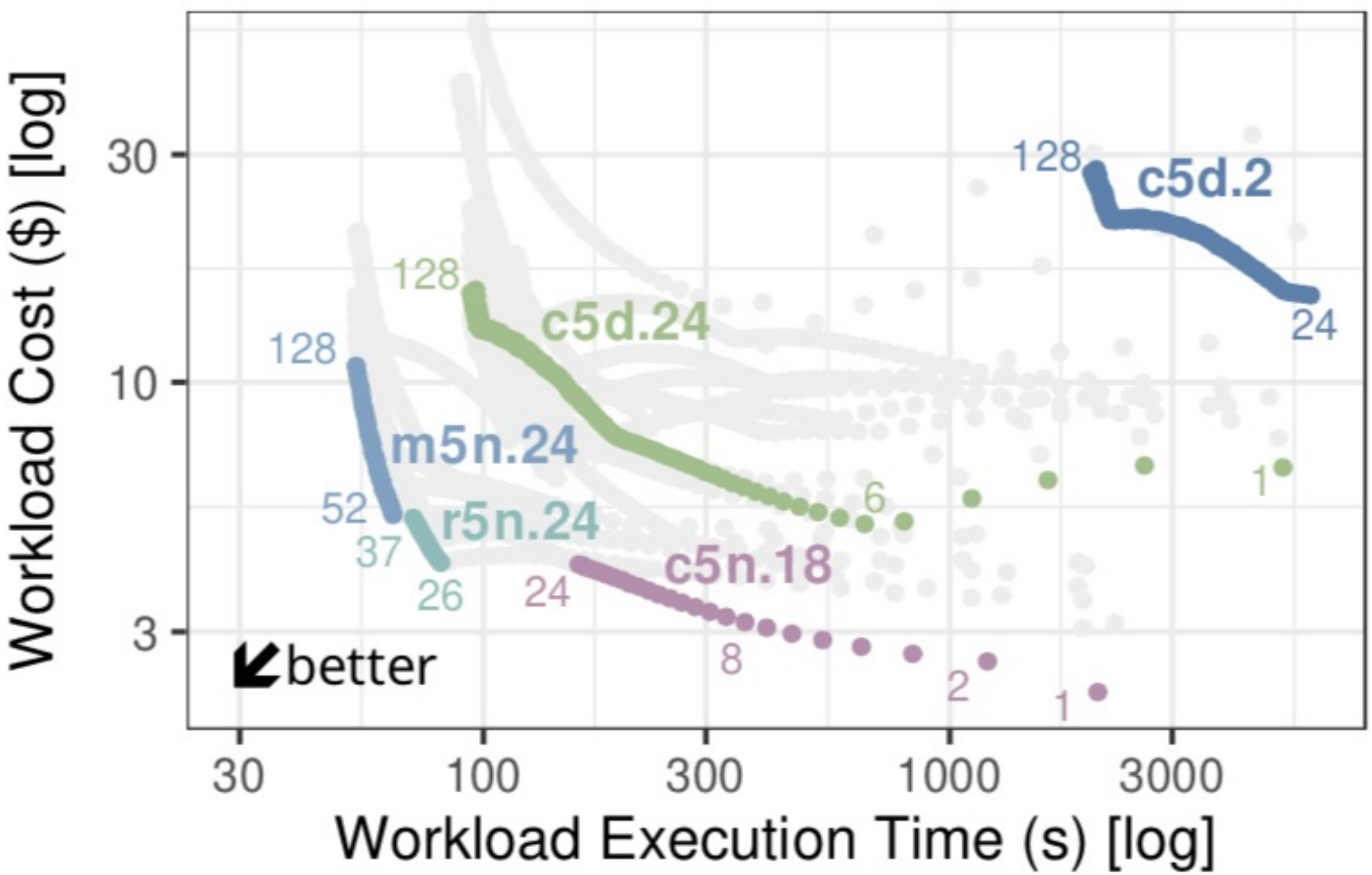
inst.	cores		DRAM		SSD		network		cost ↑
name	#	/$	GB	/$	TB	/$	Gbit/s	/$	$/h
c5n.18	36	9.3	192	49.4	-	-	100	**25.7**	3.89
c5.24	48	**11.8**	192	47.1	-	-	25	6.1	4.08
c5d.24	48	10.4	192	41.7	4×0.9	0.78	25	5.4	4.61
m5.24	48	10.4	384	83.3	-	-	25	5.4	4.61
i3.16	32	6.4	488	97.8	8×1.9	3.04	25	5.0	4.99
m5d.24	48	8.8	384	70.8	4×0.9	0.66	25	4.6	5.42
m5n.24	48	8.4	384	67.2	-	-	100	17.5	5.71
r5.24	48	7.9	768	127.0	-	-	25	4.1	6.05
m5dn.24	48	7.4	384	58.8	4×0.9	0.55	100	15.3	6.53
r5d.24	48	6.9	768	111.1	4×0.9	0.52	25	3.6	6.91
r5n.24	48	6.7	768	107.4	-	-	100	14.0	7.15
r5dn.24	48	6.0	768	95.8	4×0.9	0.45	100	12.5	8.02
i3en.24	48	4.4	768	70.8	8×7.5	**5.53**	100	9.2	10.85
x1e.32	64	2.4	3,904	**146.3**	2×1.9	0.14	25	0.9	26.69

V. Leis and M. Kuschewski, "Towards Cost-Optimal Query Processing in the Cloud", CIDR 2021

Sample cost-service trade-offs in the cloud

Fixed workload

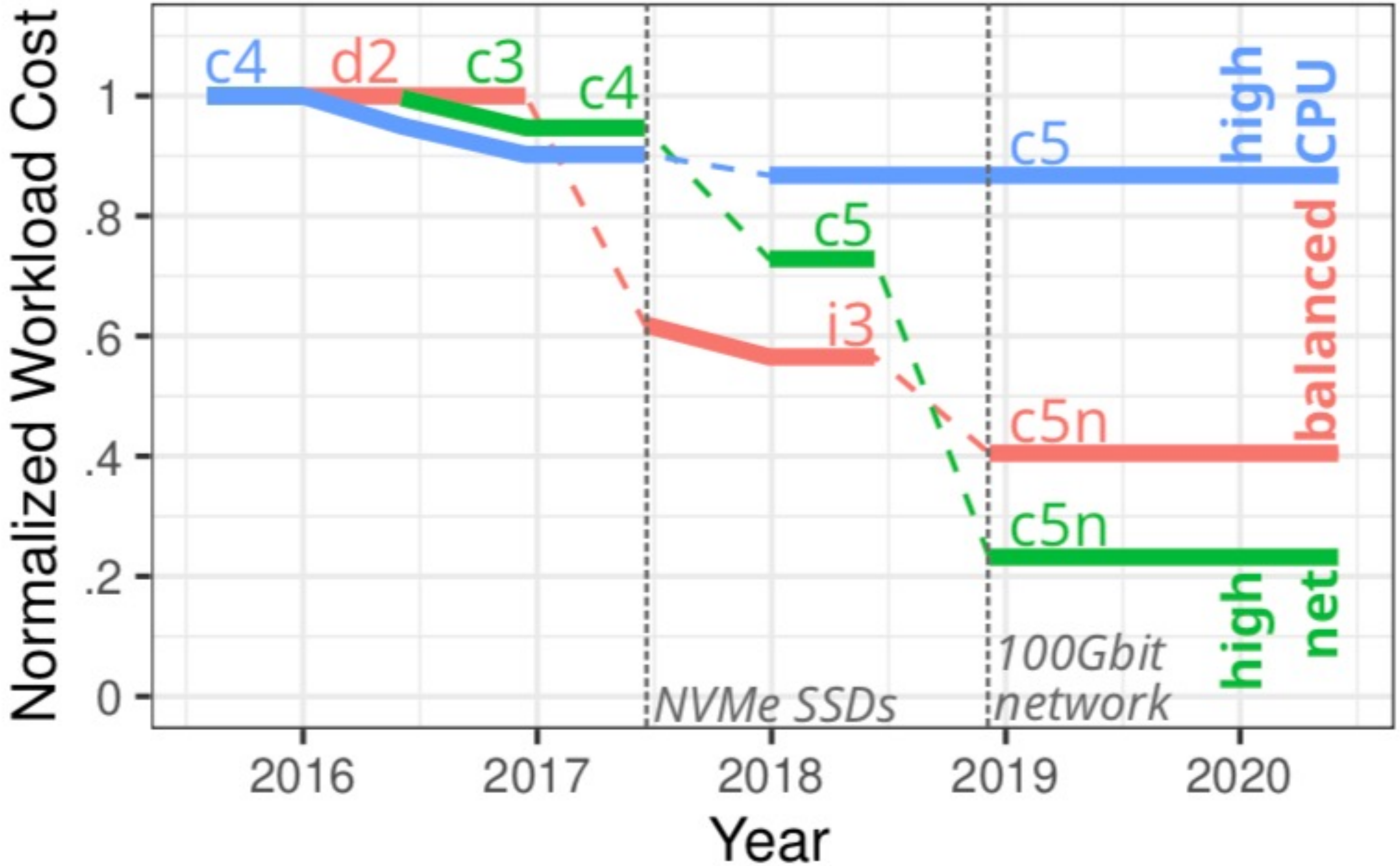
For each cloud configuration, e.g., c5d.2, vary the number of nodes



V. Leis and M. Kuschewski, "Towards Cost-Optimal Query Processing in the Cloud", CIDR 2021

Sample cost-service trade-offs in the cloud

The best offer varies over time, (also) as new configurations are proposed



V. Leis and M. Kuschewski, "Towards Cost-Optimal Query Processing in the Cloud", CIDR 2021

MULTI-TENANCY

Multi-tenancy objective and challenges

- **Degree of consolidation**: the number of databases (=software services) that are hosted on a single server or cluster (=hardware)
  - The greater the consolidation, the larger reduction in costs

- But: integrating databases (or tenants) closely can
  - *Ruin performance* for each of them
  - Expose the applications to *security risks*

- Solution: **virtualize** the available resources to facilitate consolidation while preserving performance and security

Key aspects impacted by virtualization

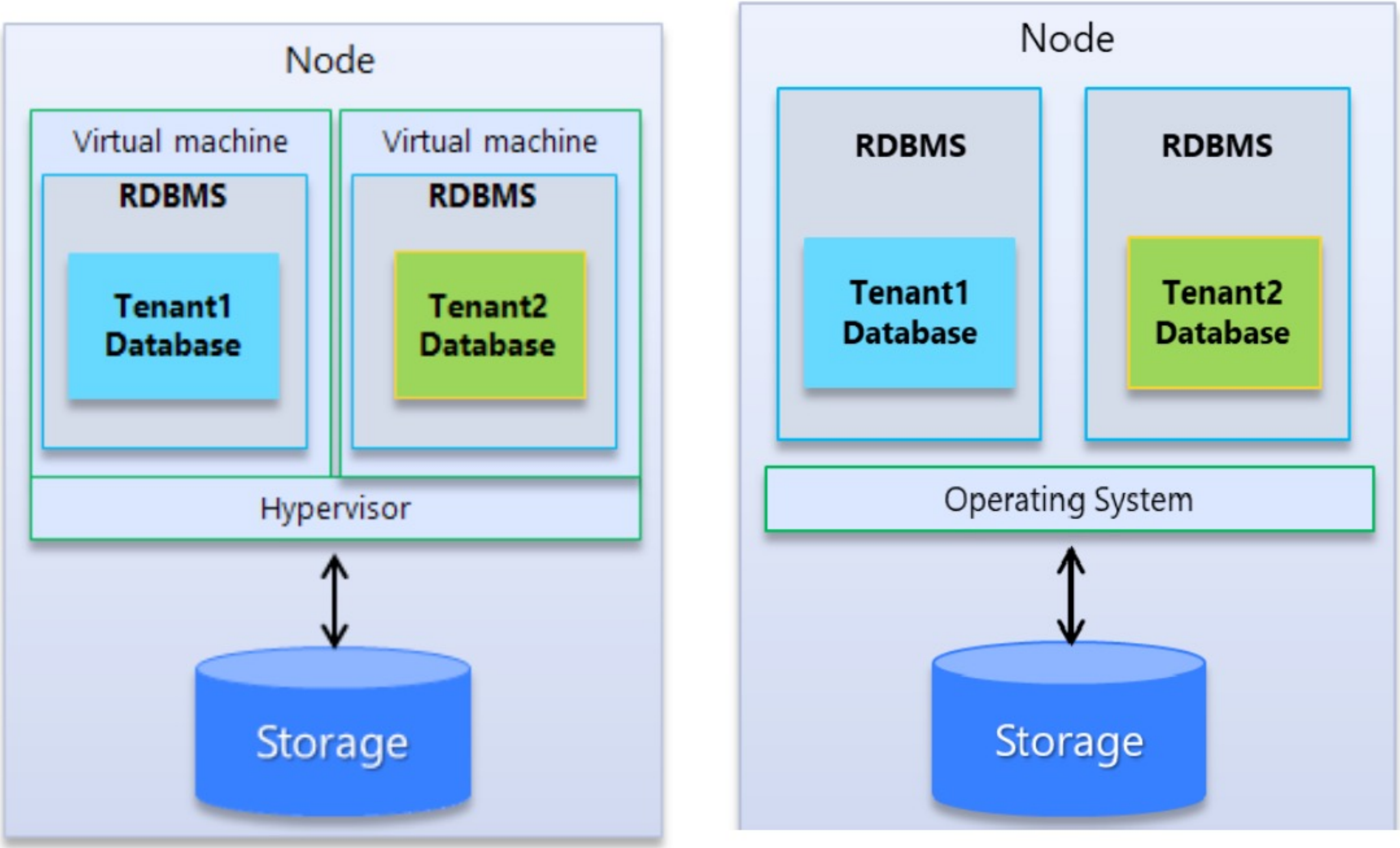
- **Degree of consolidation**: the more we can virtualize from the execution stack (bottom=hardware → … up to the application), the greater the degree of consolidation

- **Degree of isolation**: the lower down the stack is virtualization supported, the greater security and performance offered to tenants

- **Ease of provisioning**: the time taken to create a new database or upsize/downsize is lower if virtualization implemented up the stack

- **Impact of failures**: depending on where failures occur, a single failure may afect 1 or >1 tenant

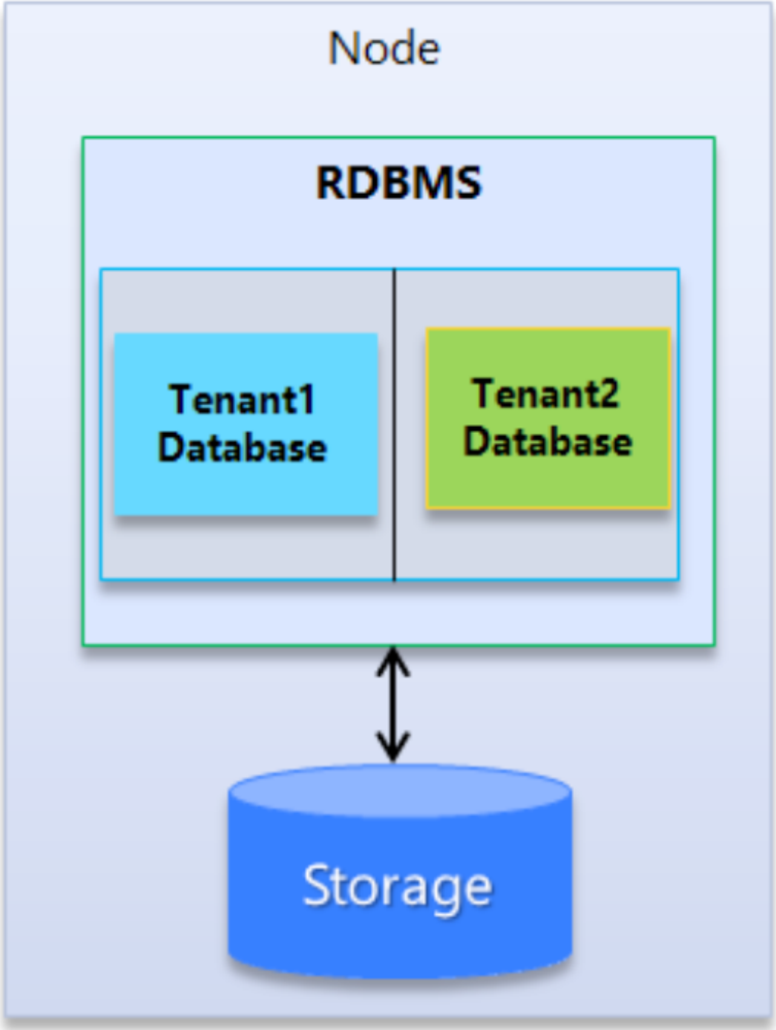
Virtualization models (1)



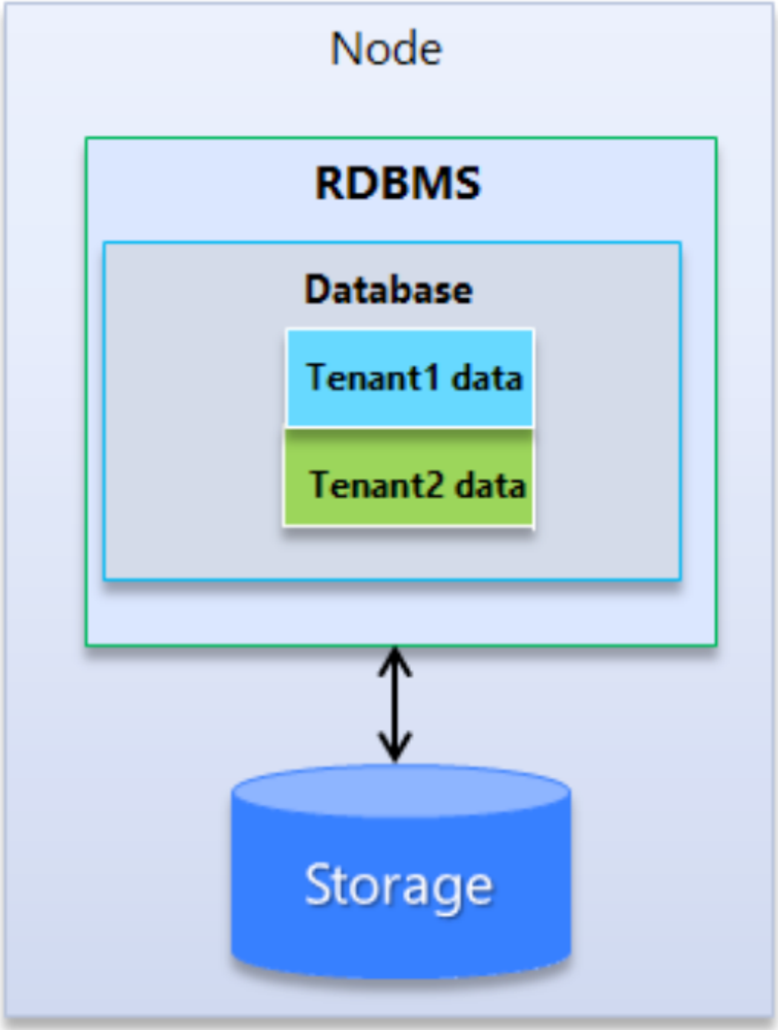
(a) Shared Hypervisor, aka Virtual Machines

(b) Shared Operating System, aka Process-Groups

Virtualization models (2)

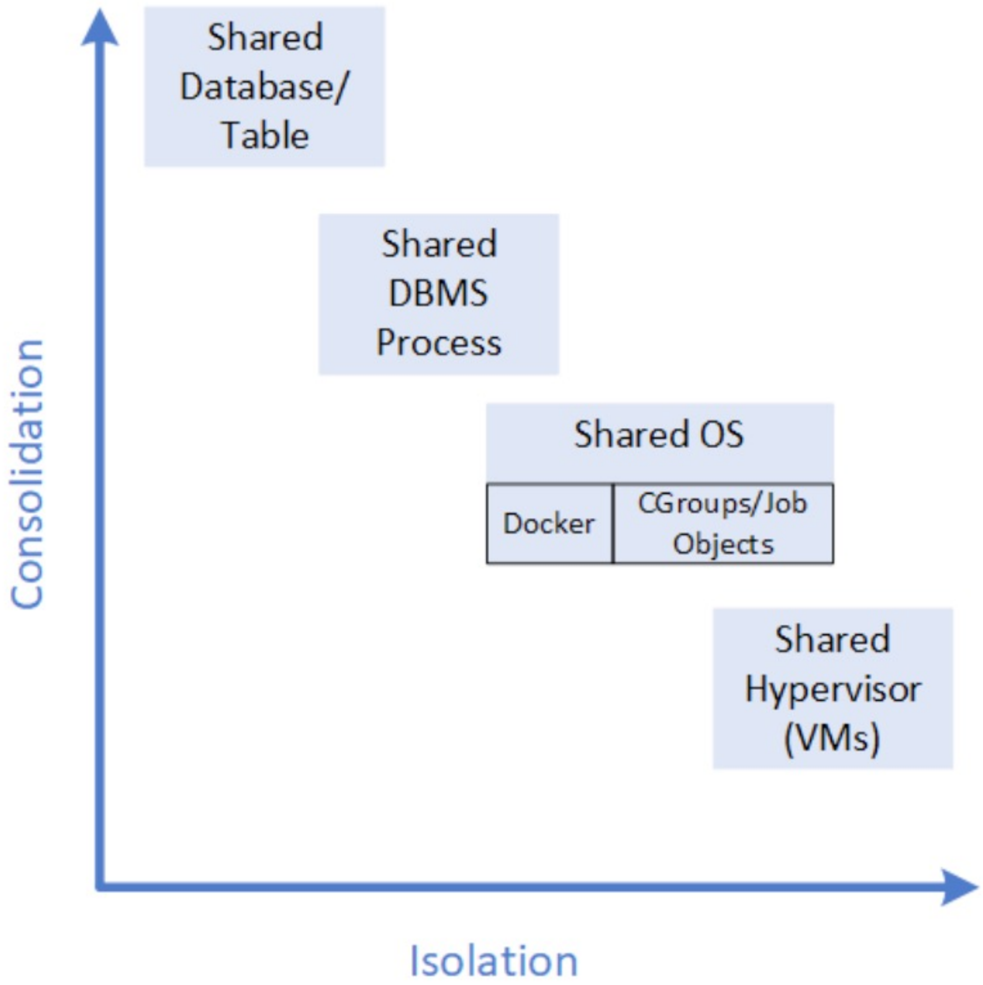


(c) Shared Process



(d) Shared Database/Table

Virtualization models: consolidation/isolation trade-off



CONCLUDING REMARKS