



CS 839 Next Generation Data Systems

Lecture 3: Cloud-native OLAP

Tianyu Li
litianyu@cs.wisc.edu

Reading Check-in

- You have ~10 min (until 11:15) to answer 2 questions
- No electronics

Recap: Cloud-native OLTP

- Very successful systems that re-engineered traditional database systems to be cloud-native
- Specialized/high-performance systems for OLTP exists, but the market has largely decided that Aurora, etc. performant enough for general OLTP use cases
- Can't we use these system for everything?

Analytical/Big Data Workloads

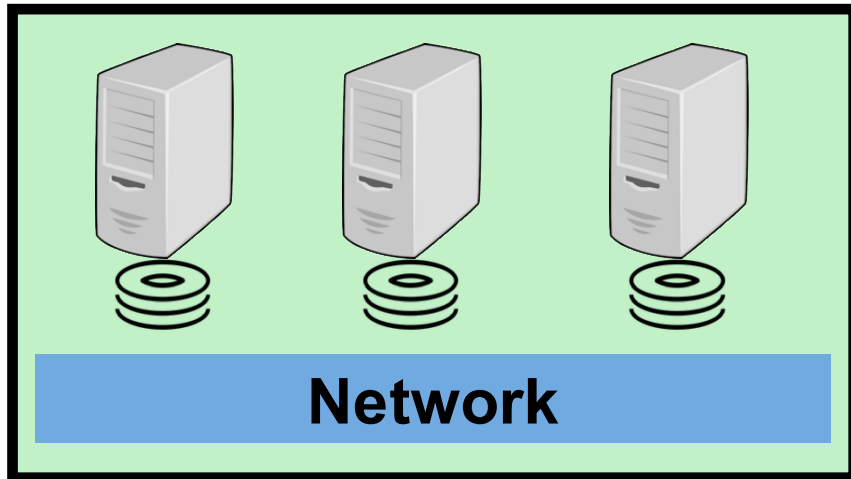
- OLTP Workload: “Lookup this customer, subtract amount from their account balance and update sales statistics”
- Analytical Workload: ”For each product category, compute monthly revenue over the last three years”
 - Large reads, potentially many joins
 - Often only touches a few columns
 - Updates happen, but are often batched

OLAP Systems

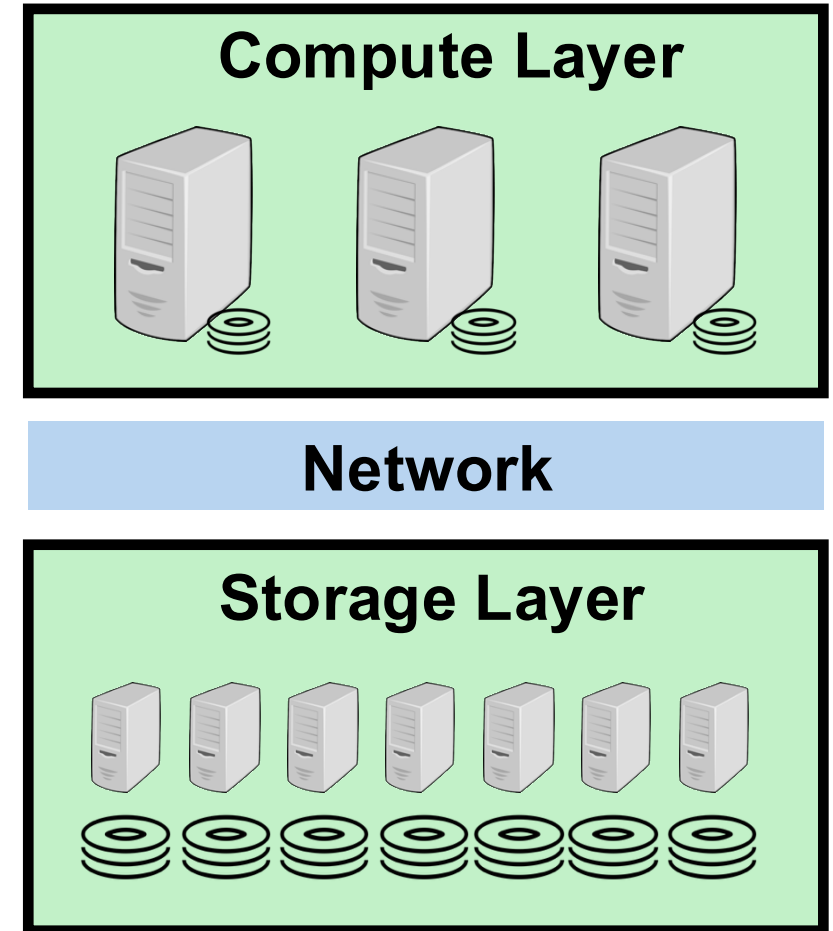
- Key Optimization 1: Columnar Storage
 - Instead of storing data as contiguous rows, store them as groups of contiguous columns
 - Allows for reading only the column that matters
 - Allows for efficient compression / metadata-based skipping
 - Allows for vectorized execution
- Key Optimization 2: Parallel/Distributed Execution
 - Scatter-Gather style query processing
 - Other people know this as MapReduce/Spark
- Separate processing (i.e., ETL) needed to load data from front-end systems (e.g., OLTP)

What's New with DBaaS?

Shared Nothing

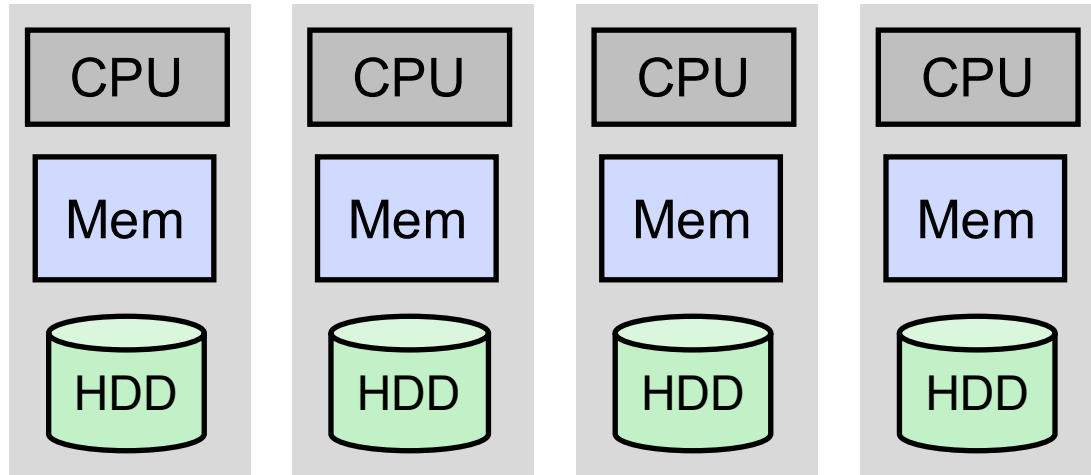
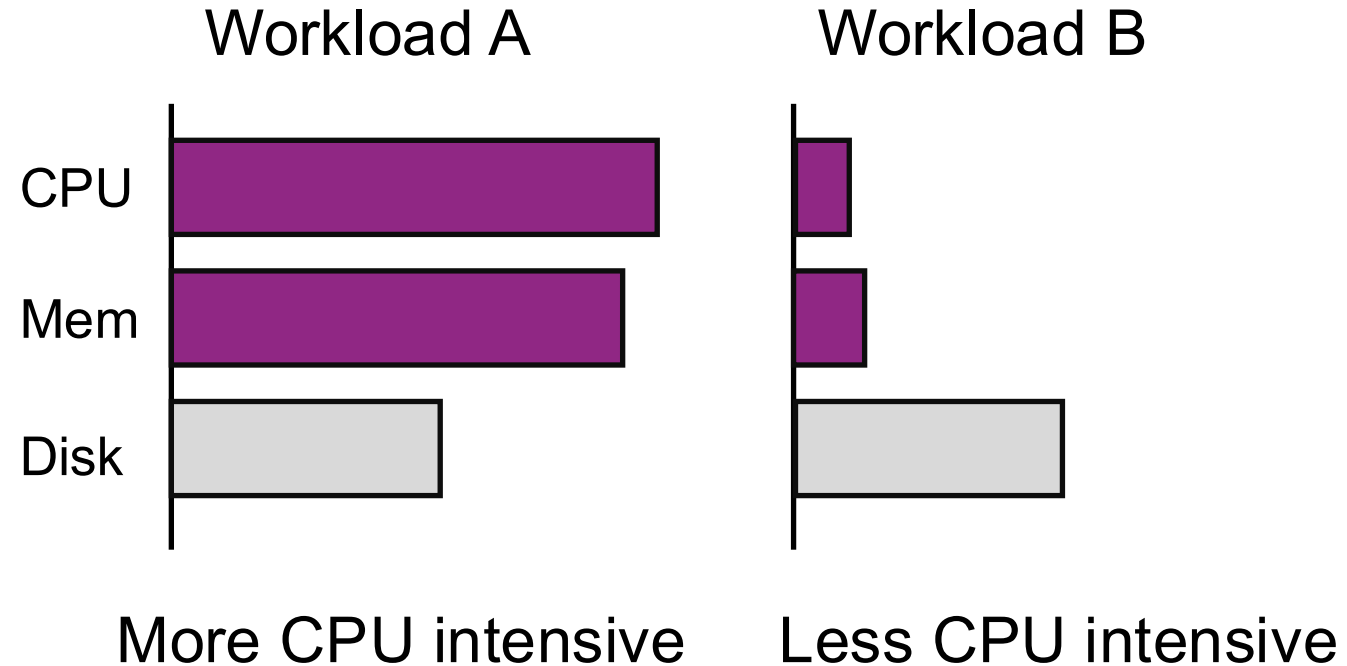


Storage disaggregation



Disaggregation $\approx$ **A cluster of specialized shared-nothing clusters**

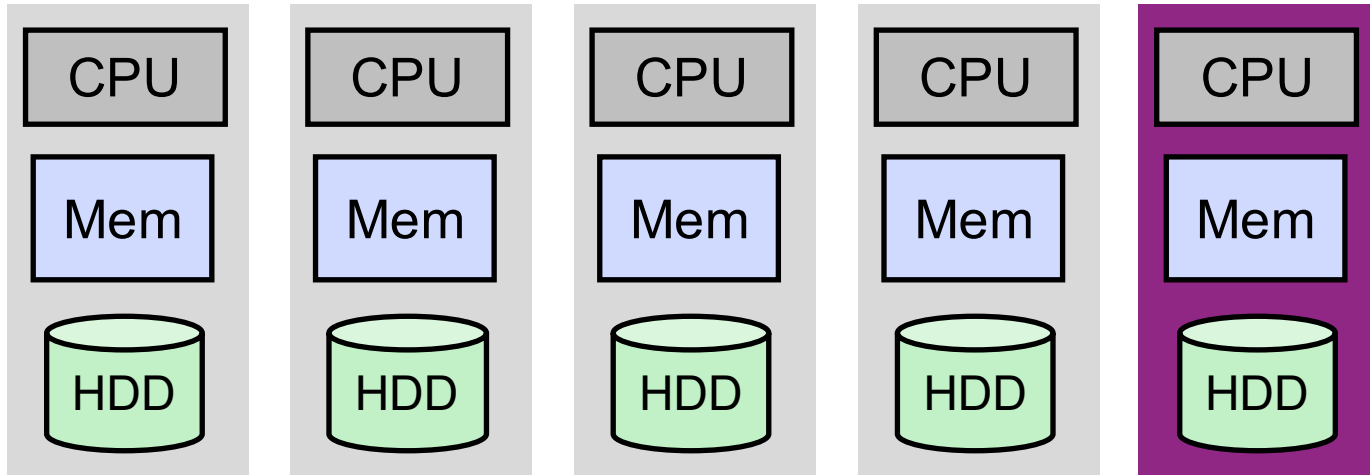
Shared Nothing – Disadvantages



Heterogeneous workload

- Static resource provisioning cannot adjust to heterogeneous workloads
- Must pay for entire cluster even when no queries exist

Shared Nothing – Disadvantages

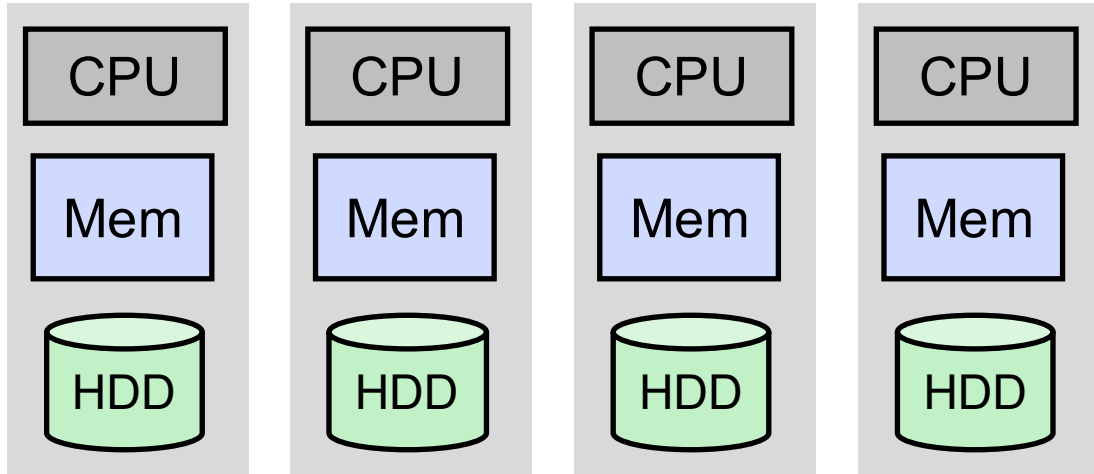


Heterogeneous workload

Membership changes

- Add/Remove a node: data redistribution
- Significant hassle that may cause down time

Shared Nothing – Disadvantages



Heterogeneous workload

Membership changes

Online upgrade

- Similar to membership change but affect all nodes

Storage Disaggregation Architecture

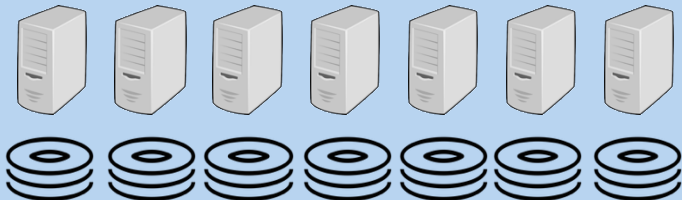
Why disaggregate? Compute and storage behave differently

Compute Layer



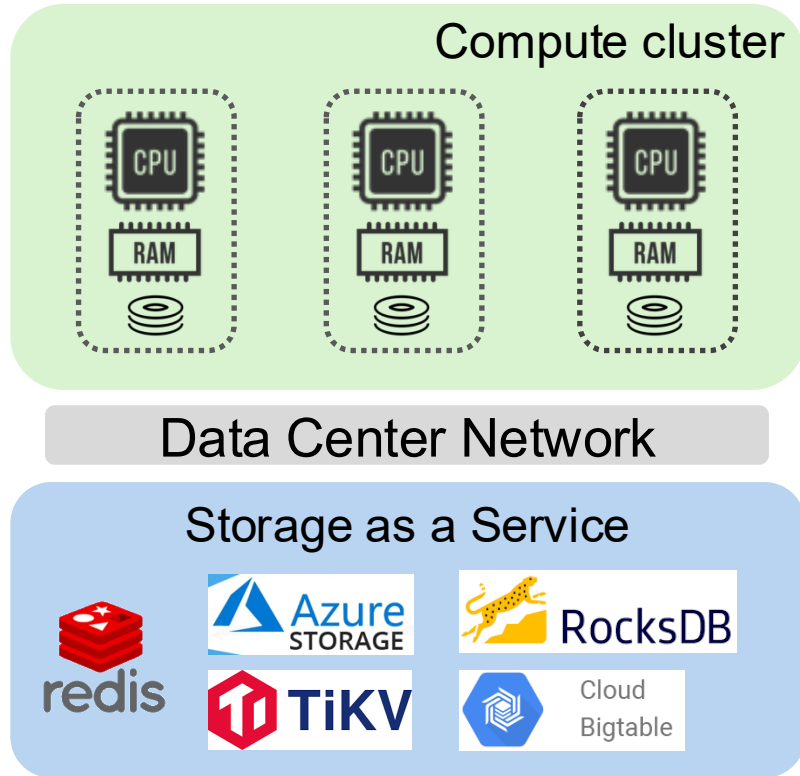
Network

Storage Layer



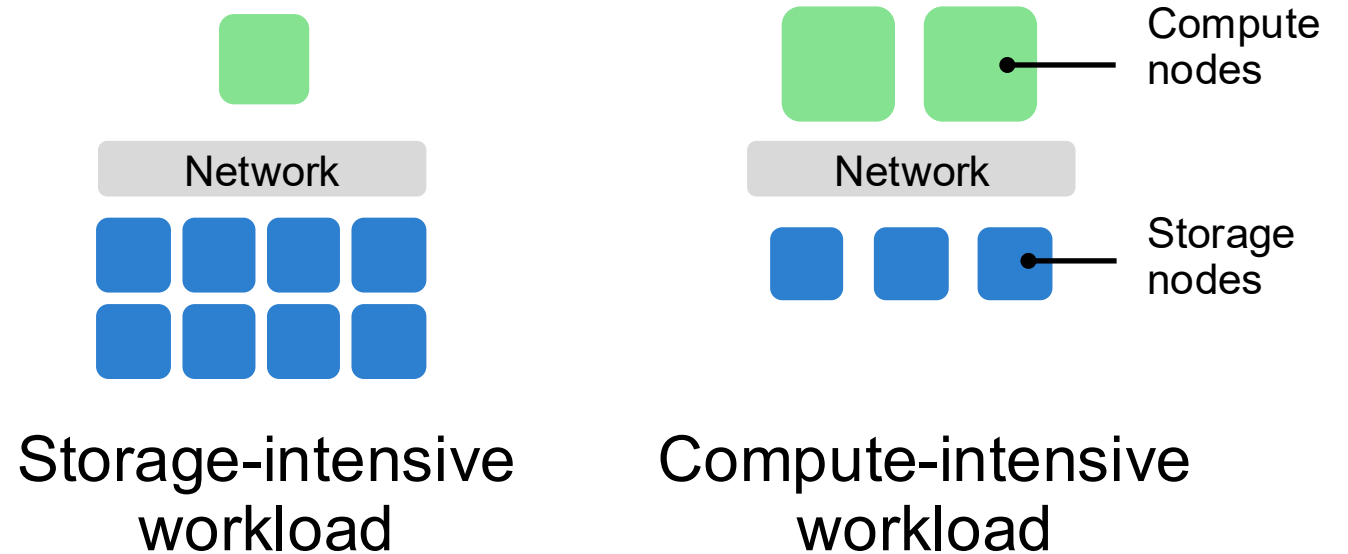
- **Stateless** → easy to scale
 - **Expensive** → VMs cost **\$0.1–10/hour**
 - **Bursty** → demand changes rapidly
-
- **Stateful** → difficult to scale
 - **Low-cost** → S3 costs **\$0.02/GB/month**
 - **Steady** → demand changes slowly

Advantages of Storage Disaggregation

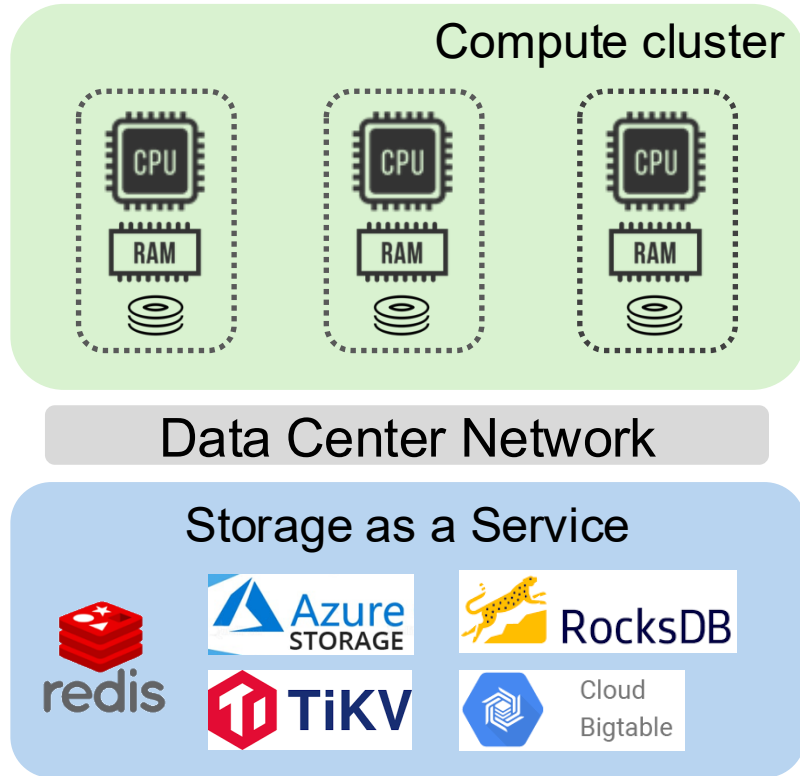


Advantage #1: Independent Scaling

- Compute and storage resources can scale independently



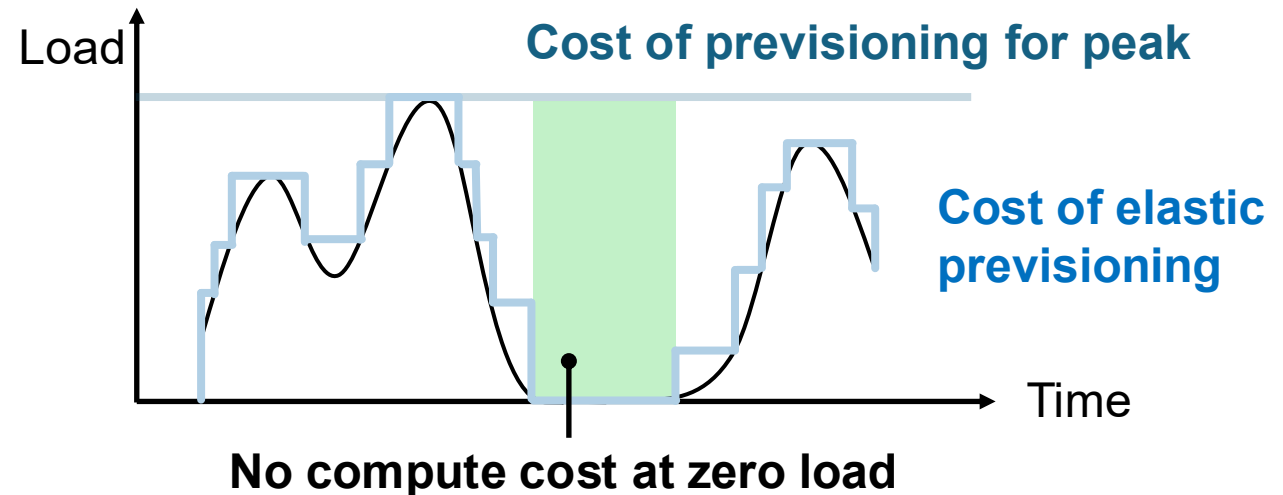
Advantages of Storage-Disaggregation



Advantage #2: Elastic Scaling of Compute

- Storage service can be much cheaper than compute servers

S3 storage price	\$0.02 per GB per month
16 vCPU Virtual Machine	\$0.5 per hour per VM



Snowflake

The Snowflake Elastic Data Warehouse

Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes,
Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel,
Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley,
Peter Povinec, Greg Rahn, Spyridon Triantafyllis, Philipp Unterbrunner

Snowflake Computing

ABSTRACT

We live in the golden age of distributed computing. Public cloud platforms now offer virtually unlimited compute and storage resources on demand. At the same time, the Software-as-a-Service (SaaS) model brings enterprise-class systems to users who previously could not afford such systems due to their cost and complexity. Alas, traditional data warehousing systems are struggling to fit into this new environment. For one thing, they have been designed for fixed resources and are thus unable to leverage the cloud's elasticity. For another thing, their dependence on complex ETL pipelines and physical tuning is at odds with the flexibility and freshness requirements of the cloud's new types of semi-structured data and rapidly evolving workloads.

We decided a fundamental redesign was in order. Our mission was to build an enterprise-ready data warehousing solution for the cloud. The result is the Snowflake Elastic Data Warehouse, or "Snowflake" for short. Snowflake is a multi-tenant, transactional, secure, highly scalable and elastic system with full SQL support and built-in extensions for semi-structured and schema-less data. The system is offered as a pay-as-you-go service in the Amazon cloud. Users upload their data to the cloud and can immediately manage and query it using familiar tools and interfaces. Implementation began in late 2012 and Snowflake has been generally available since June 2015. Today, Snowflake is used in production by a growing number of small and large organizations alike. The system runs several million queries per day over multiple petabytes of data.

In this paper, we describe the design of Snowflake and its novel multi-cluster, shared-data architecture. The paper highlights some of the key features of Snowflake: extreme elasticity and availability, semi-structured and schema-less data, time travel, and end-to-end security. It concludes with lessons learned and an outlook on ongoing work.

Categories and Subject Descriptors

Information systems [Data management systems]: Database management system engines

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGMOD/PODS'16 June 26 - July 01, 2016, San Francisco, CA, USA

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-3531-7/16/06.

DOI: <http://dx.doi.org/10.1145/2882903.2903741>

Keywords

Data warehousing, database as a service, multi-cluster shared data architecture

1. INTRODUCTION

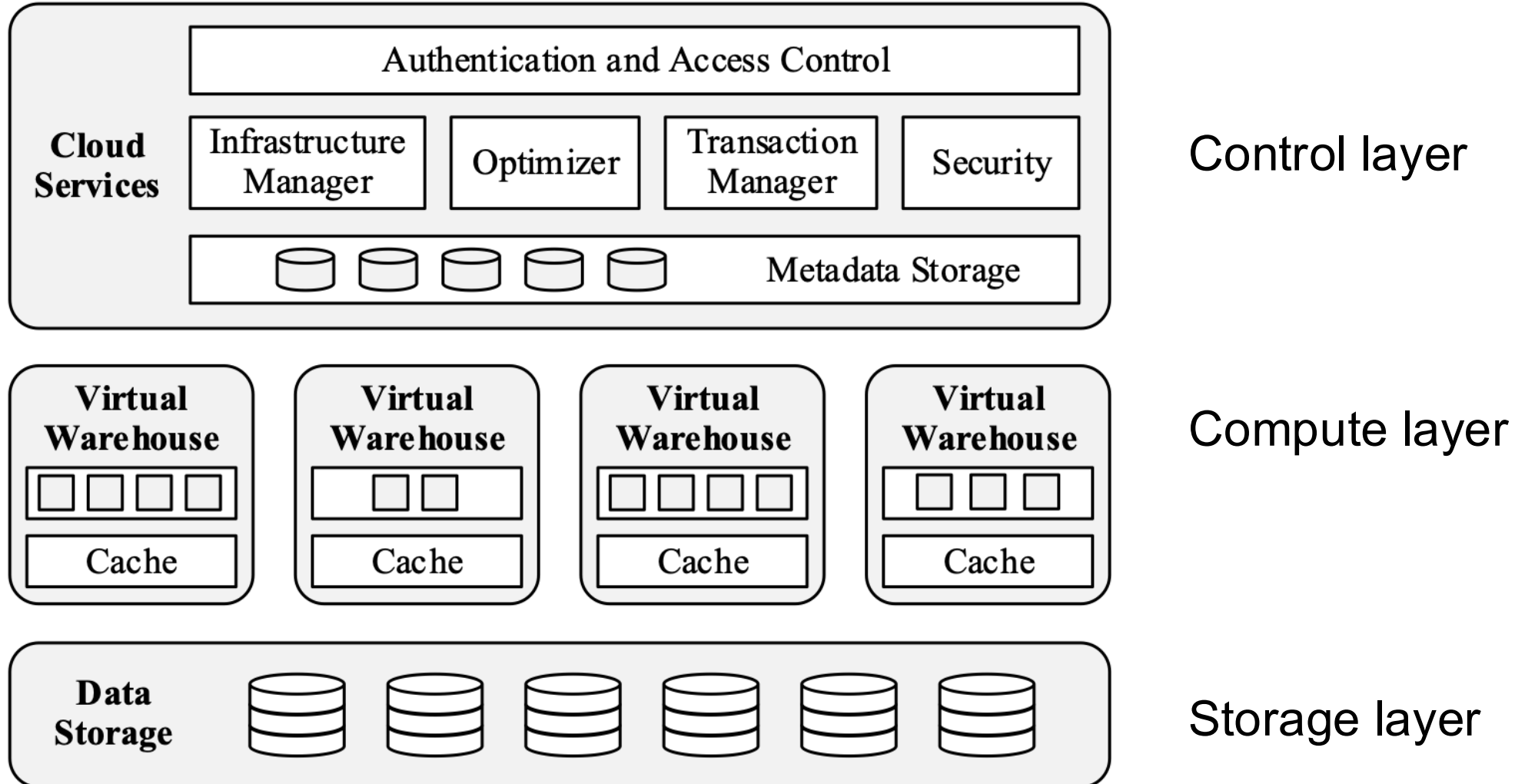
The advent of the cloud marks a move away from software delivery and execution on local servers, and toward shared data centers and software-as-a-service solutions hosted by platform providers such as Amazon, Google, or Microsoft. The shared infrastructure of the cloud promises increased economies of scale, extreme scalability and availability, and a pay-as-you-go cost model that adapts to unpredictable usage demands. But these advantages can only be captured if the *software* itself is able to scale elastically over the pool of commodity resources that is the cloud. Traditional data warehousing solutions pre-date the cloud. They were designed to run on small, static clusters of well-behaved machines, making them a poor architectural fit.

But not only the platform has changed. Data has changed as well. It used to be the case that most of the data in a data warehouse came from sources within the organization: transactional systems, enterprise resource planning (ERP) applications, customer relationship management (CRM) applications, and the like. The structure, volume, and rate of the data were all fairly predictable and well known. But with the cloud, a significant and rapidly growing share of data comes from less controllable or external sources: application logs, web applications, mobile devices, social media, sensor data (Internet of Things). In addition to the growing volume, this data frequently arrives in schema-less, semi-structured formats [3]. Traditional data warehousing solutions are struggling with this new data. These solutions depend on deep ETL pipelines and physical tuning that fundamentally assume predictable, slow-moving, and easily categorized data from largely internal sources.

In response to these shortcomings, parts of the data warehousing community have turned to "Big Data" platforms such as Hadoop or Spark [8, 11]. While these are indispensable tools for data center-scale processing tasks, and the open source community continues to make big improvements such as the Stinger Initiative [48], they still lack much of the efficiency and feature set of established data warehousing technology. But most importantly, they require significant engineering effort to roll out and use [16].

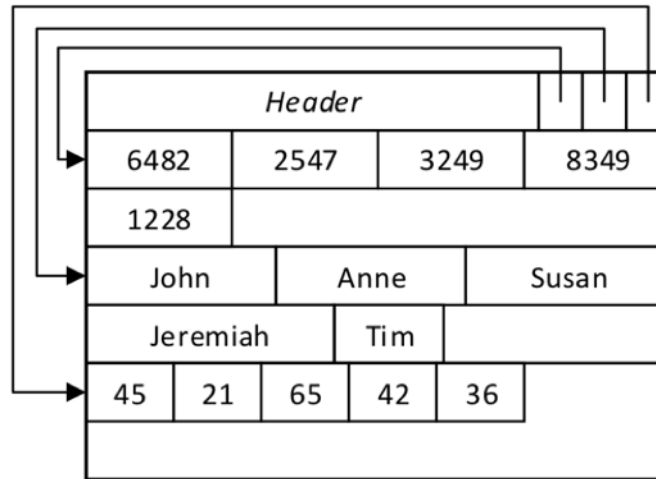
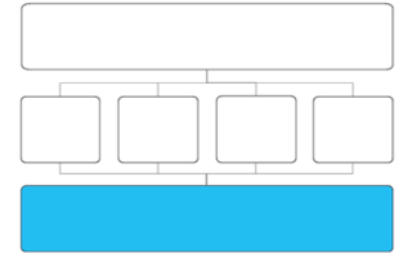
We believe that there is a large class of use cases and workloads which can benefit from the economics, elasticity, and service aspects of the cloud, but which are not well served by either traditional data warehousing technology or

Multi-Cluster Shared-Data Architecture



Architecture – Storage

Data format: PAX

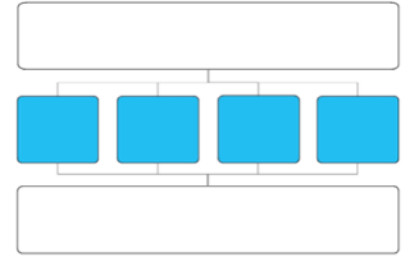


Data horizontally partitioned into immutable files (~16MB)

- An update = remove and add an entire file
- Queries download file headers and columns they are interested in

For our purposes, this is just a standard storage service like S3

Architecture – Virtual Warehouse



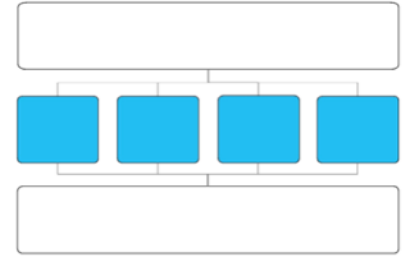
Elasticity and Isolation

- Created, destroyed, or resized at any point (may shutdown all VWs)
- User may create multiple VWs for multiple queries
- Determine the VW size based on performance and cost requirements

Architecture – Virtual Warehouse

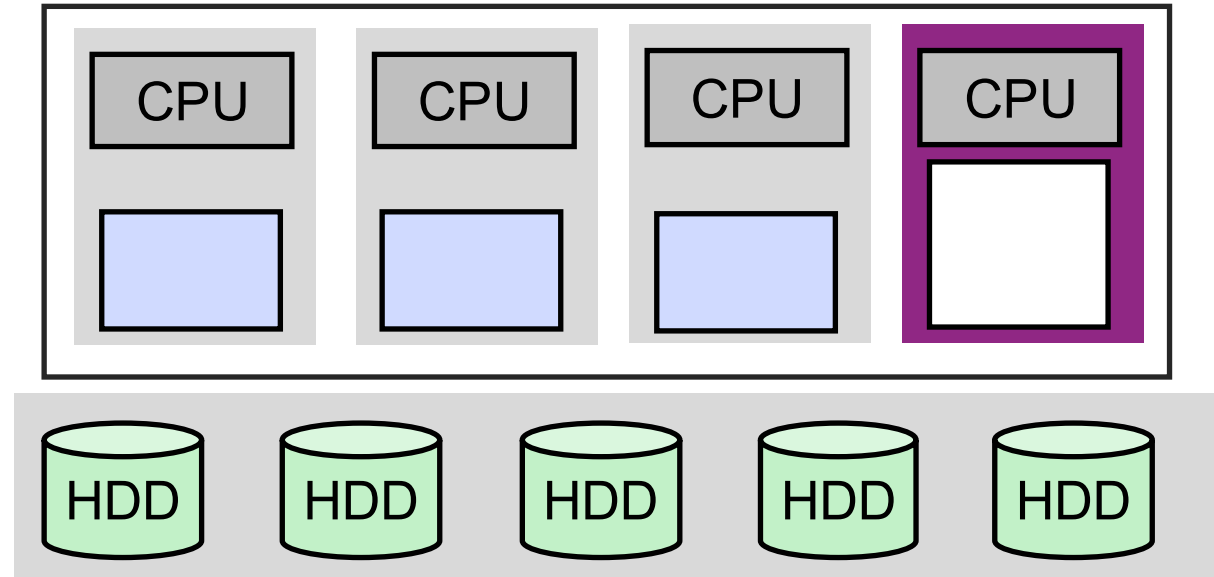
Local caching

- S3 data can be cached in local memory or disk



Consistent hashing

- When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped
- When a VW is resized, **no data shuffle required**; rely on LRU to replace cache content



Architecture – Cloud Services

Multi-tenant layer shared across multiple users

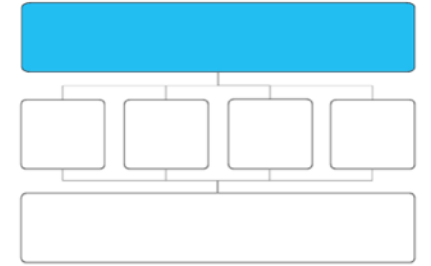
Query optimization

Concurrency control

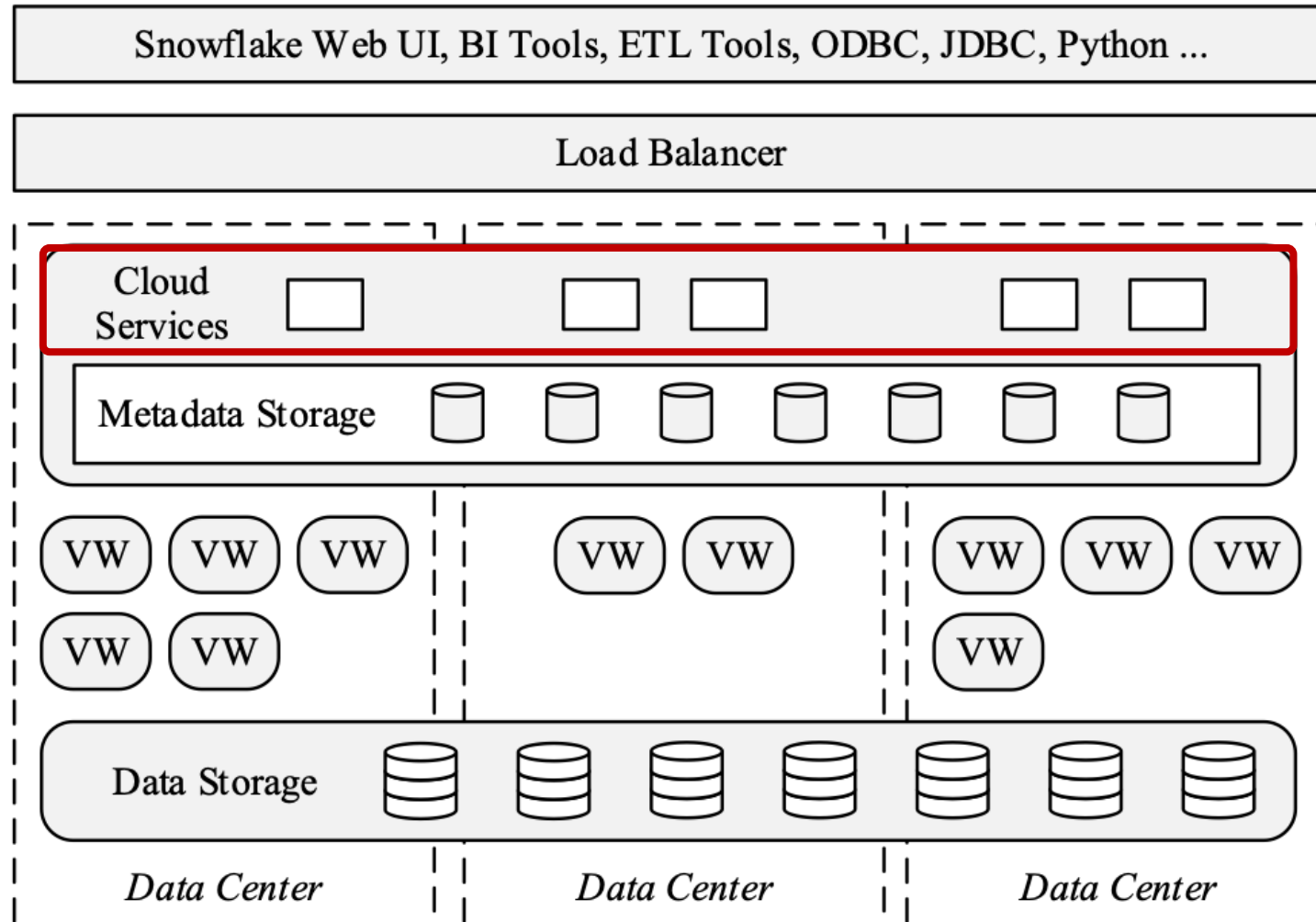
- Isolation: snapshot isolation (SI)
- S3 data is immutable, update entire files with MVCC
- Versioned snapshots used for time traveling
- Assumption: bulk data loading rather than OLTP

Pruning

- Snowflake has no index (same as some other data warehousing systems)
- Min-max based pruning: store min and max values for a data block

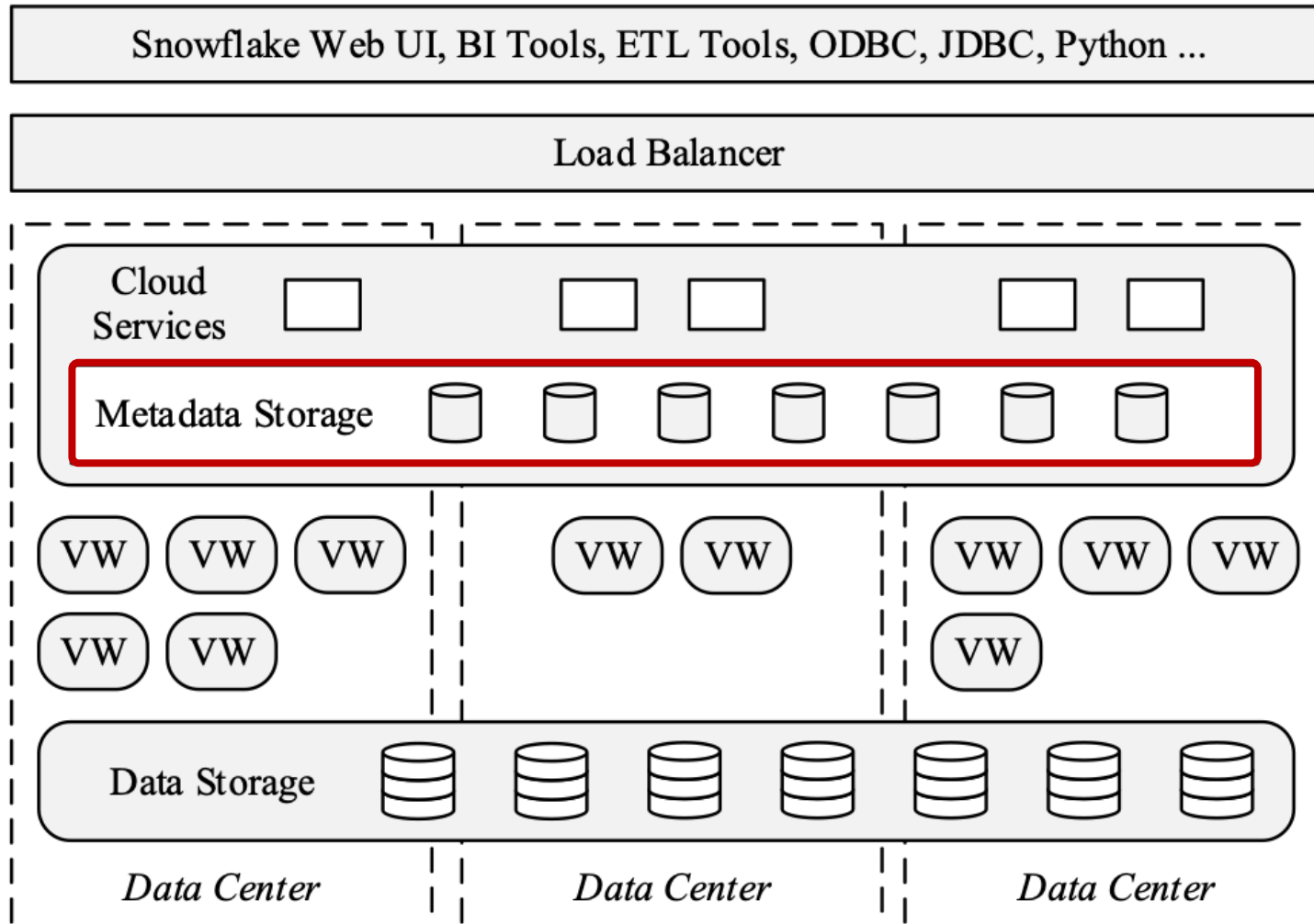


High Availability and Fault Tolerance



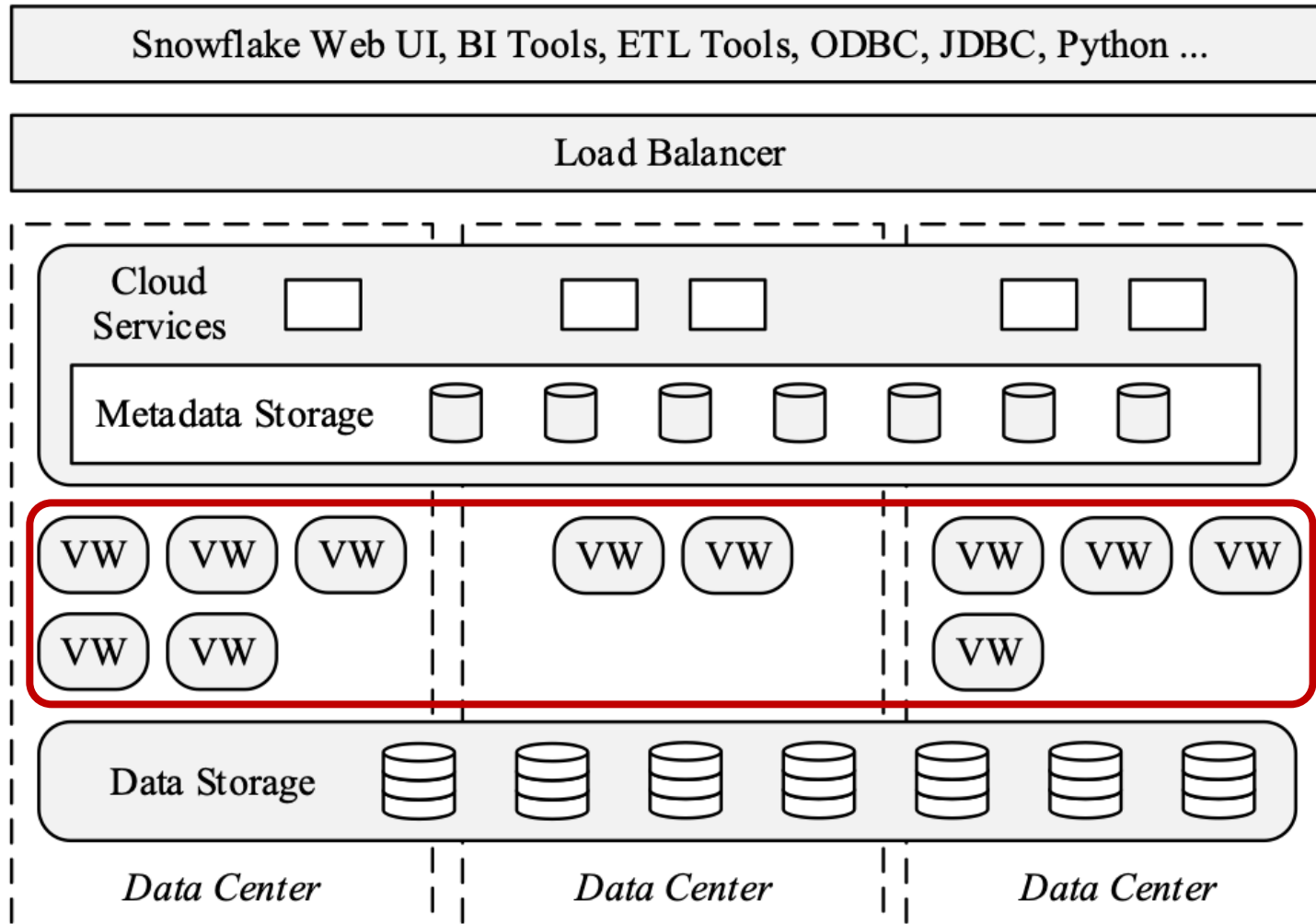
Stateless services

High Availability and Fault Tolerance



Replicated metadata
(FoundationDB)

High Availability and Fault Tolerance



One node failure in VW

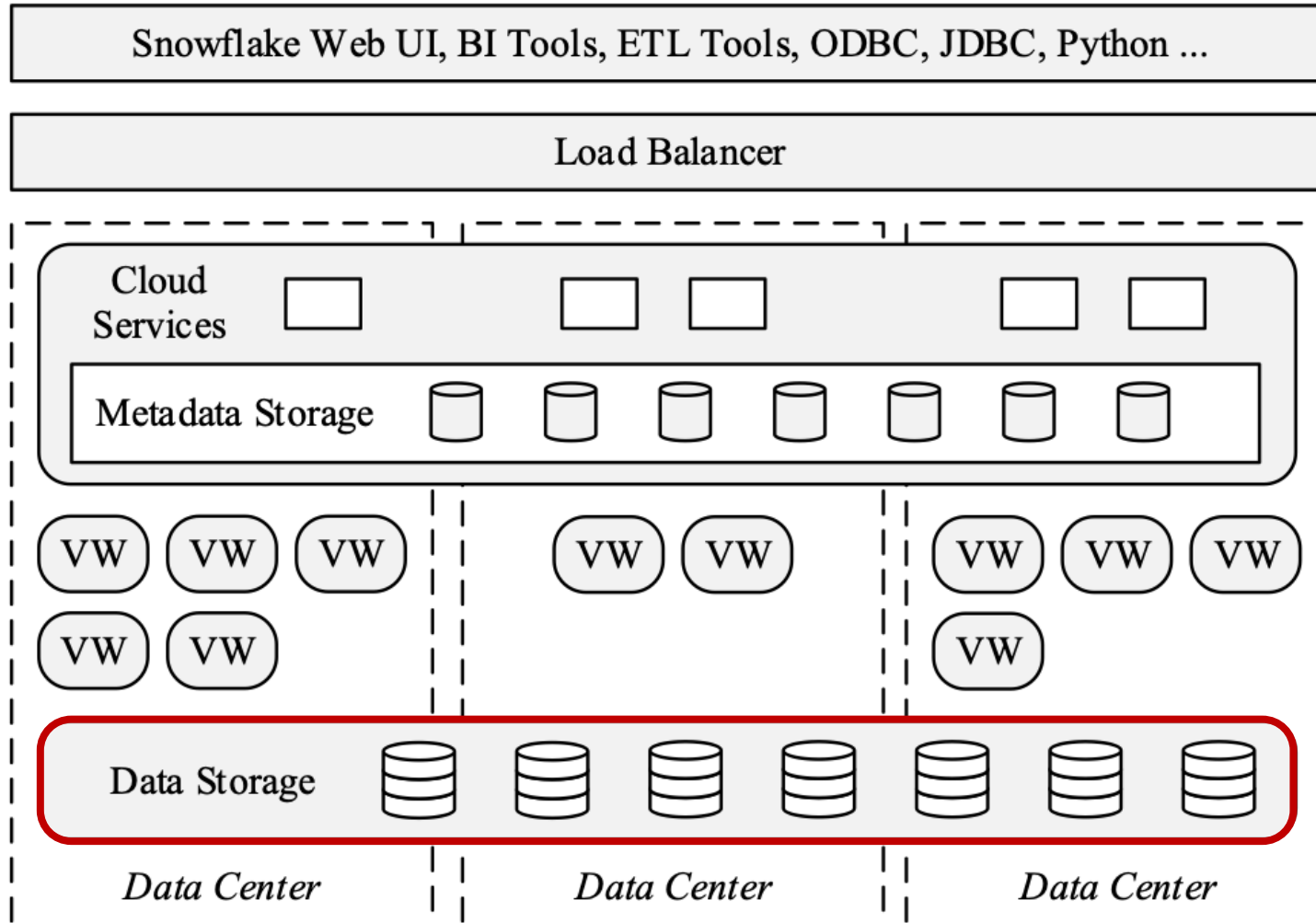
- Re-execute with failed node immediately replaced
- Re-execute with reduced number of nodes

Whole AZ failure

- Re-execute by re-provisioning a new VW

Hot-standby nodes

High Availability and Fault Tolerance



S3 is highly available and durable

What's wrong with Cloud Warehouses?

- Traditional Database model: database owns the data, users want answers
- Imagine a modern workload: train a recommendation model using purchases and clickstream events
 - Involves some OLAP to prepare data, however, data feeds into an ML pipeline
 - Data export can be extremely slow in traditional systems

Data Lake

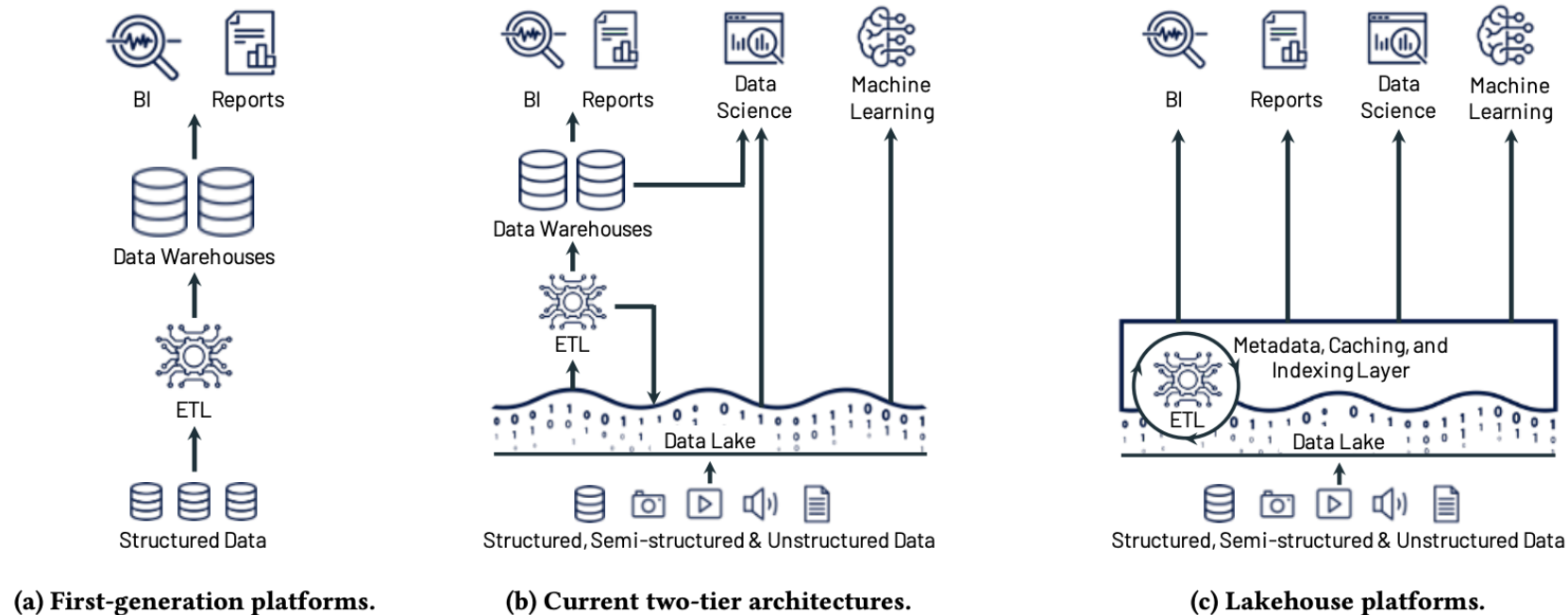
- Grew from insufficiencies of traditional warehouses
 - Reluctance to support semi-/unstructured data
 - High cost to store large datasets
 - Insufficient integration with data science & ML tools
- Fancy name for a pile of data on shared storage (e.g., S3)
 - Often based on open formats (e.g., Apache Parquet)
 - Often include semi-/unstructured data like JSONs, images, etc.
 - Users often bring their own compute (e.g., Spark)
 - Flexible consumption: for SQL, exploratory data science, ML, etc.
 - Specific data may be loaded into warehouses

What's wrong with Data Lakes?

- Key Weakness: Files do not provide database guarantees
 - No centralized governance over updates, schema changes, versions, etc.
 - Readers may see partially published ETL output
 - Access control, discovery, lineage, etc. require additional infrastructure
 - Some efficiency problems: advanced database optimization, data skipping, etc. often unavailable

A Unifying Architecture?

- Databricks proposed the Lakehouse architecture in 2021 in an attempt to add database-like upper layers to the data lake



Lakehouse: Key Ideas

- Metadata Engine on data lakes
 - A transactional engine tracks which files are part of which table version(s)
 - Optimized for batch ingestion, etc., not OLTP
- Efficient Lakehouse Engines
 - Zone maps, query optimization, SSD caches, vectorized execution
- Declarative I/O for data science & ML
 - Direct storage access from ML workloads
 - Integration with DataFrame API, etc.

Meanwhile on the other side...

- Snowflake added support for Parquet & Iceberg
 - Ability to rapidly export from propriety snowflake format
 - Ability to directly access customer-owned storage
- Snowflake added integration with external catalogs
 - i.e., Apache Polaris
- Snowflake launched managed catalog services...
 - i.e., Snowflake Horizon Catalog

Convergence

- Snowflake and Databricks started from two extremes and incrementally added each other's features until they start to look very similar
 - i.e., SoTA OLAP engine over open storage + unified metadata management
- Similar stories with Microsoft/Google/Amazon's offerings
- Perhaps lakehouses are the final answer to cloud data processing?

Discussion

- What is the place of OLTP in this new, unified architecture?
- What are some new management challenges that arise from the lakehouse architecture, and how can we solve them?
- How does AI change the way we access and manage lakehouses?

Next Week

- We will cover vector DBs from the basics to the latest systems
- Reminder to find teammates and think about what to work on!