WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

CS 764: Topics in Database Management Systems
Lecture 25: Snowflake

Xiangyao Yu
11/30/2022

Announcements

Exam solutions announced on Piazza

Exam grade announced on Canvas

- per-question grades are emailed to individuals

Today’s Paper

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

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

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.

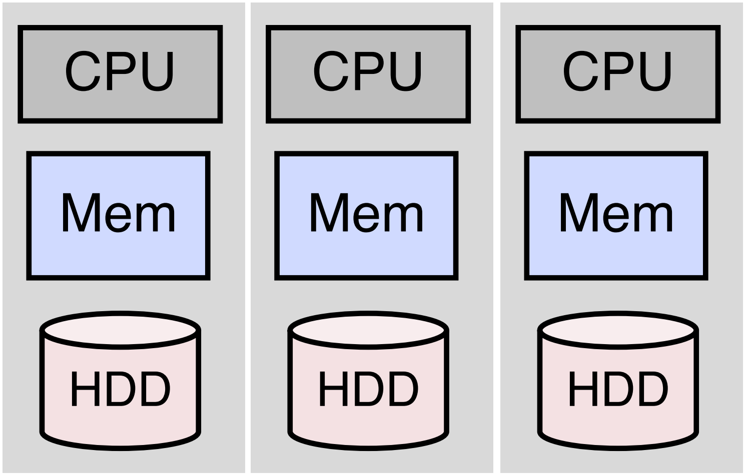
But not only the platform has changed. Data has changed as well. It used to be the case that most of the data in a data warehouse came from sources within the organization: transactional systems, enterprise resource planning (ERP) applications, customer relationship management (CRM) applications, and 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.

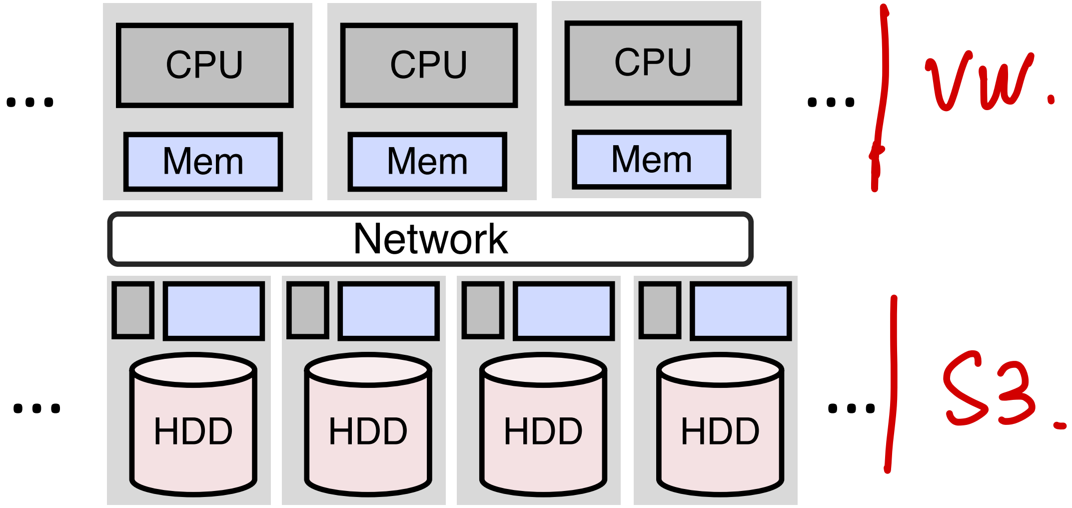
In response to these shortcomings, parts of the data warehousing community have turned to “Big Data” platforms such as Hadoop or Spark [8, 11]. While these are indispensable tools for data center-scale processing tasks, and the open source community continues to make big improvements such as the Stinger 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

SIGMOD 2016

On-Premises vs. Cloud





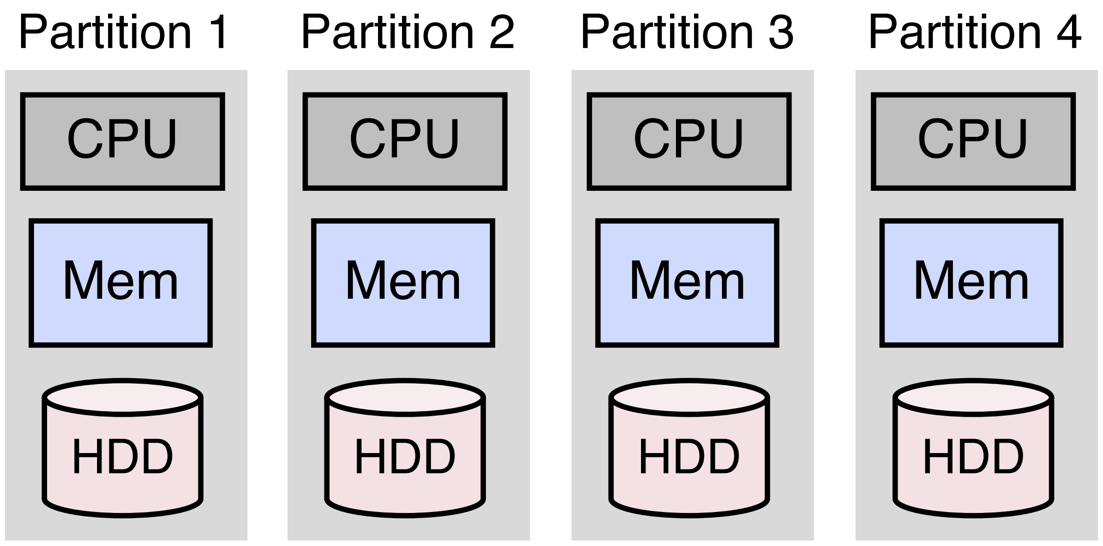
On-premises

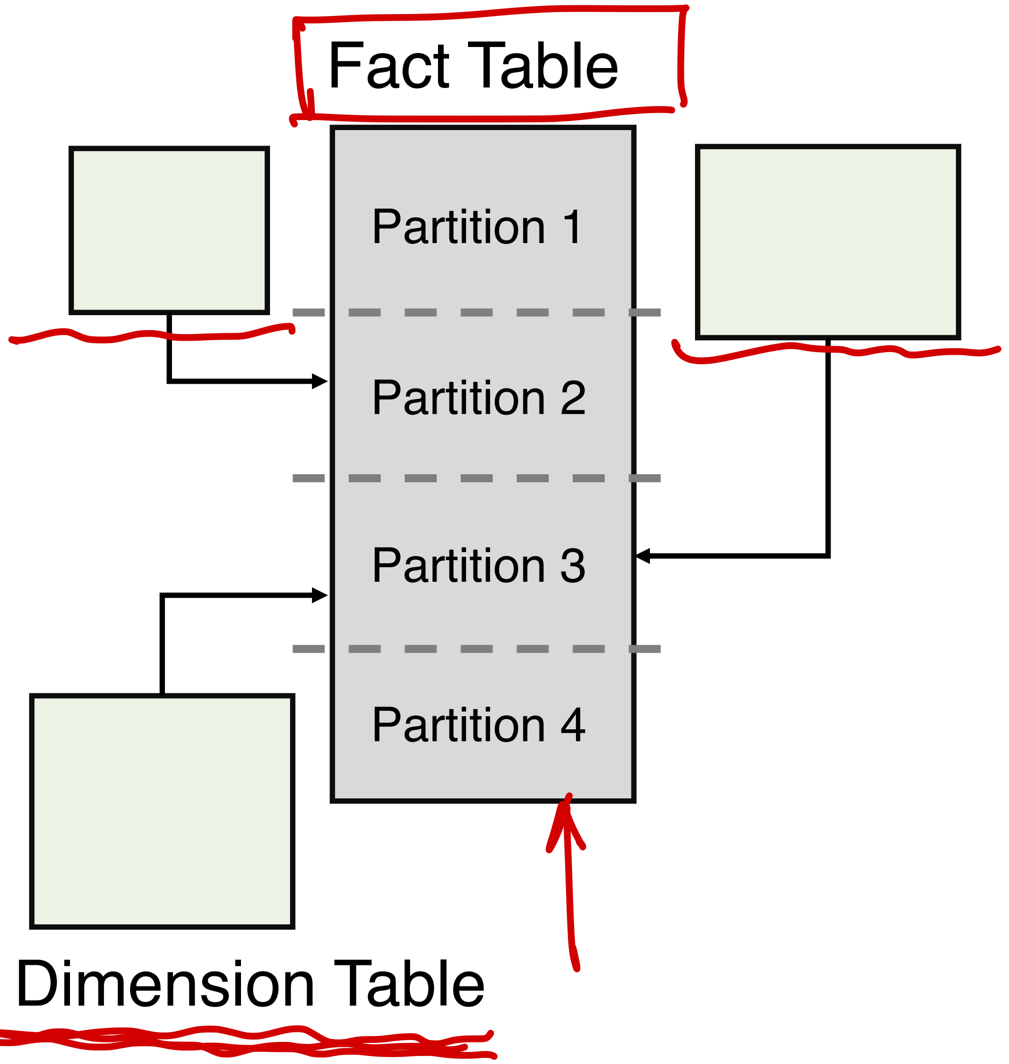
- Fixed and limited hardware resources
- **Shared-nothing** architecture

Cloud

- Virtually infinite computation & storage, Pay-as-you-go price model
- **Disaggregation** architecture

Shared Nothing – Advantages

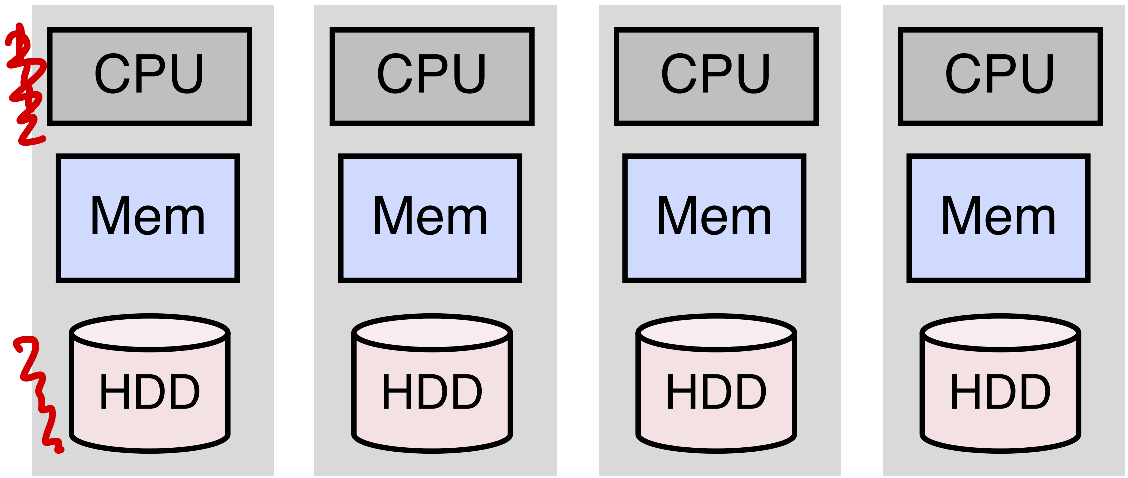


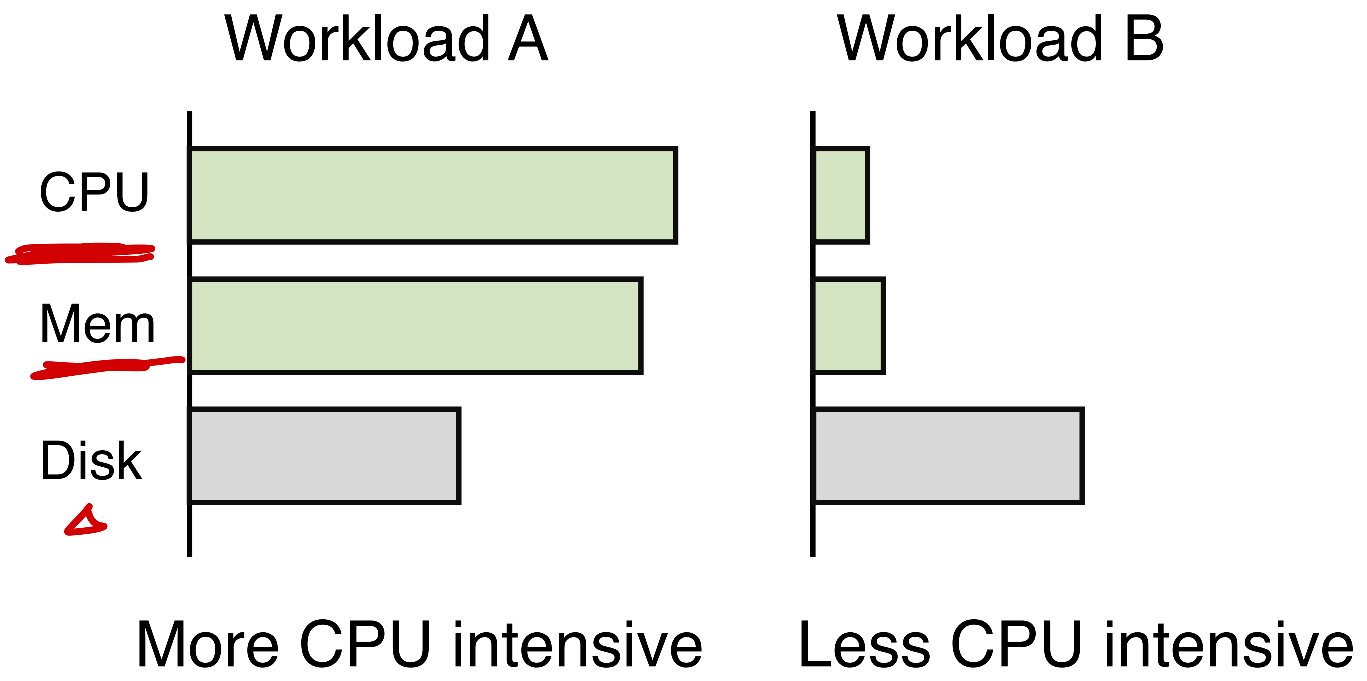


Scalability: horizontal scaling

- Scales well for star-schema queries

Shared Nothing – Disadvantages

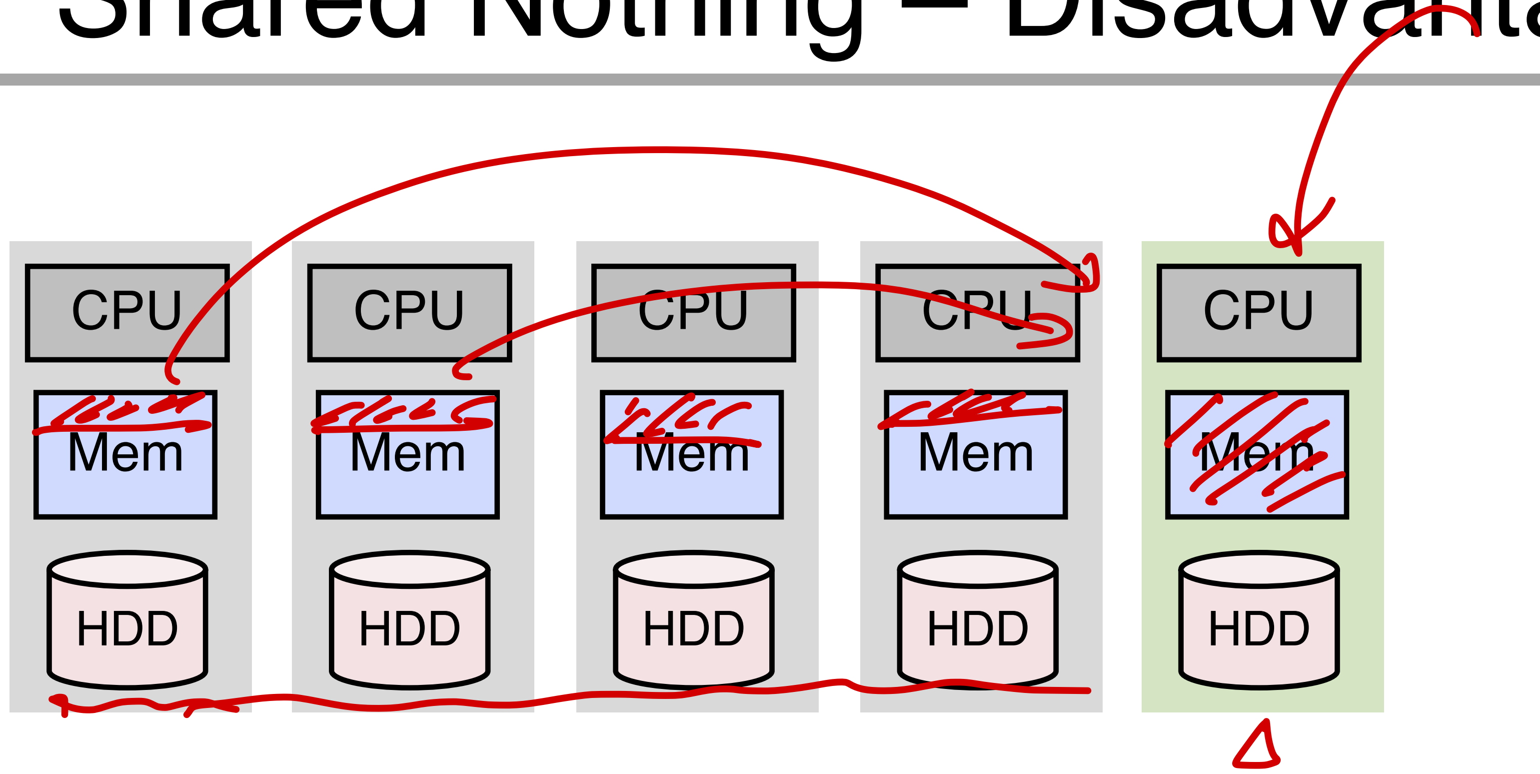




Heterogeneous workload

- Static resource provisioning cannot adjust to heterogeneous workloads

Shared Nothing – Disadvantages

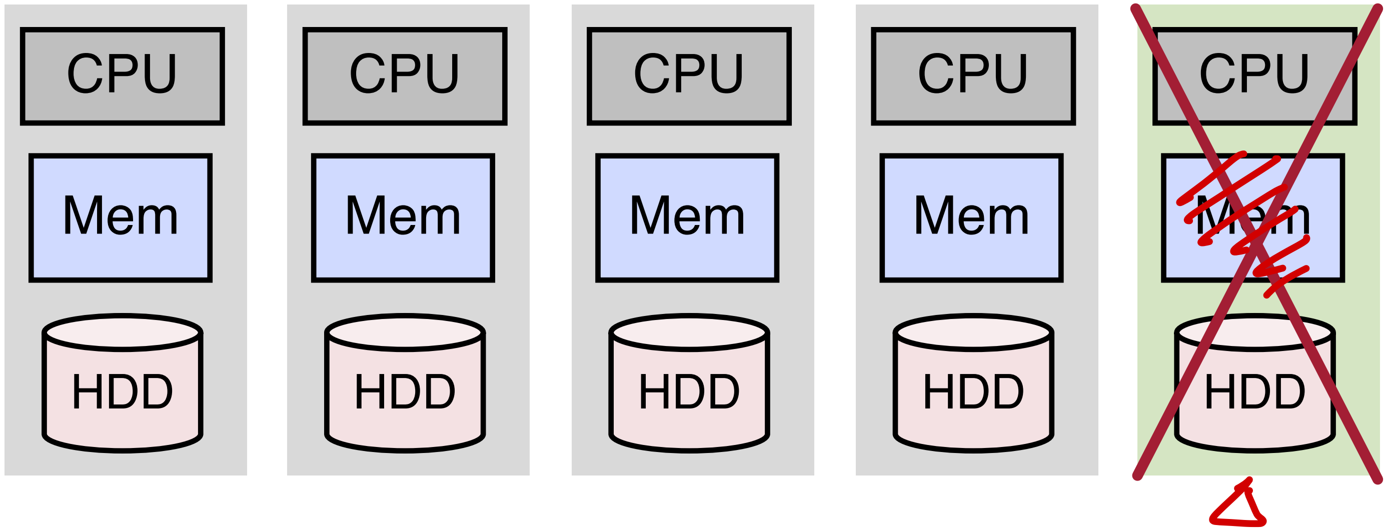


Heterogeneous workload

Membership changes

- Add a node: data redistribution

Shared Nothing – Disadvantages

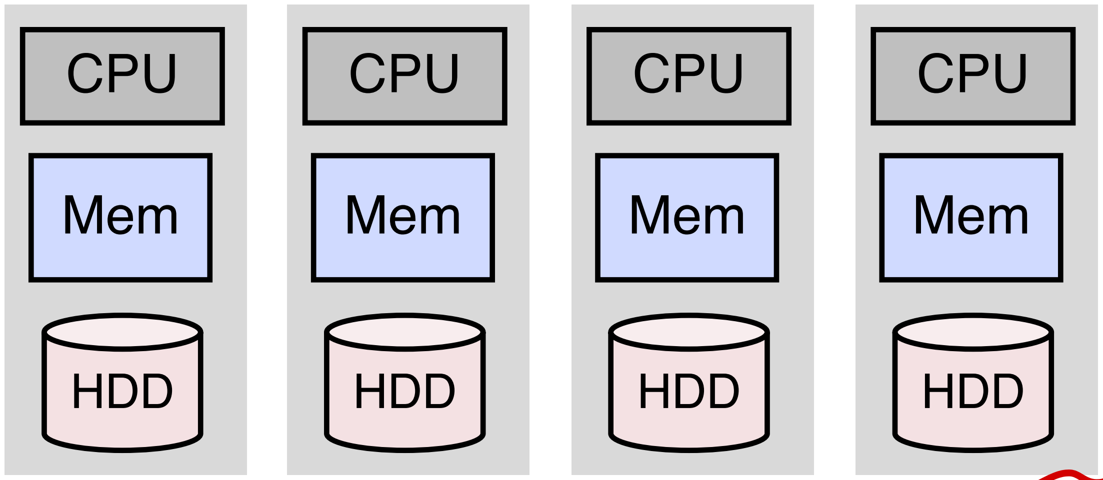


Heterogeneous workload

Membership changes

- Add a node: data redistribution
- Delete a node: similar to the fault tolerance problem

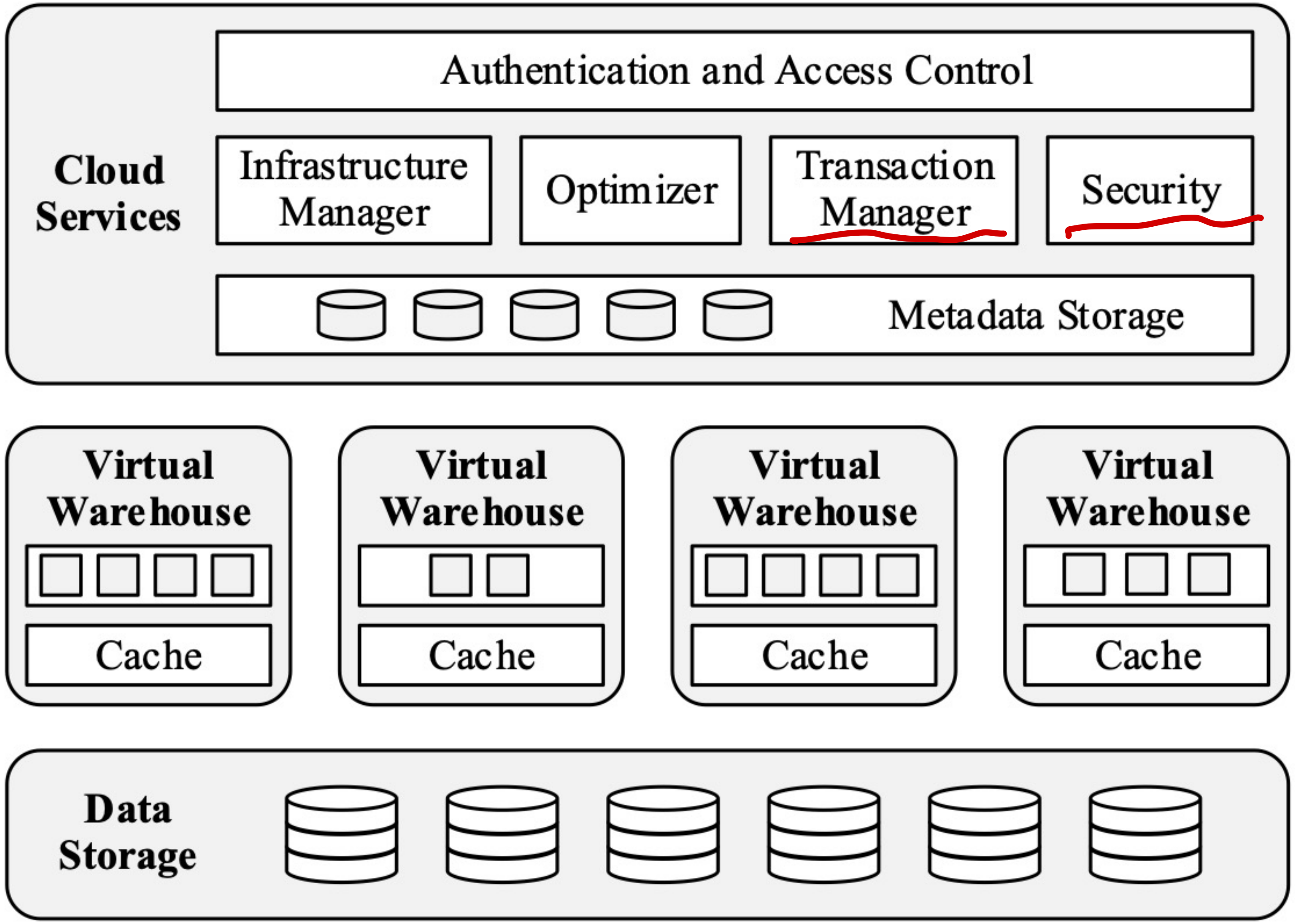
Shared Nothing – Disadvantages



Heterogeneous workload
Membership changes
Online upgrade

- Similar to membership change but affect all nodes

Multi-Cluster Shared-Data Architecture



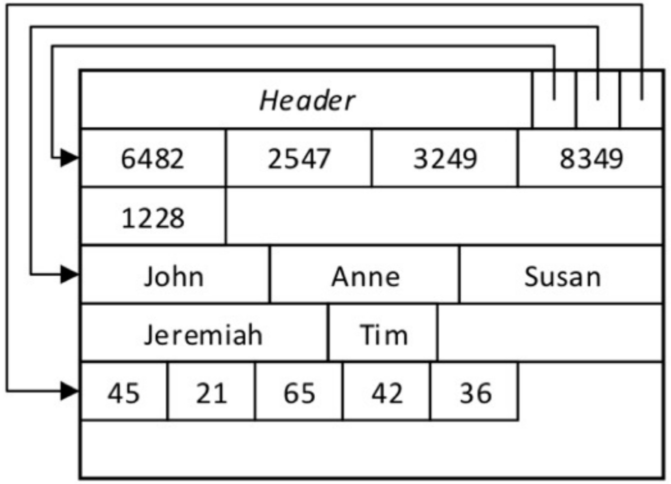
Control layer

Compute layer

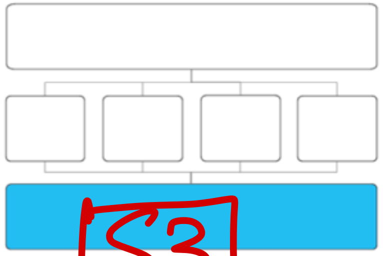
Storage layer

Architecture – Storage

Data format: PAX



5 rows



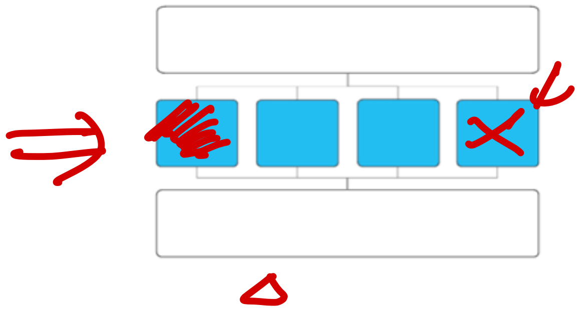
Data horizontally partitioned into immutable files (~16MB)

– An update = remove and add an entire file

– Queries download file headers and columns they are interested in

Intermediate data spilling to S3

Architecture – Virtual Warehouse

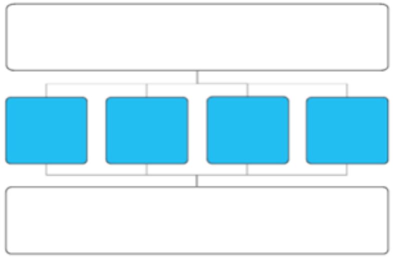


T-Shirt sizes: XS to 4XL

Elasticity and Isolation

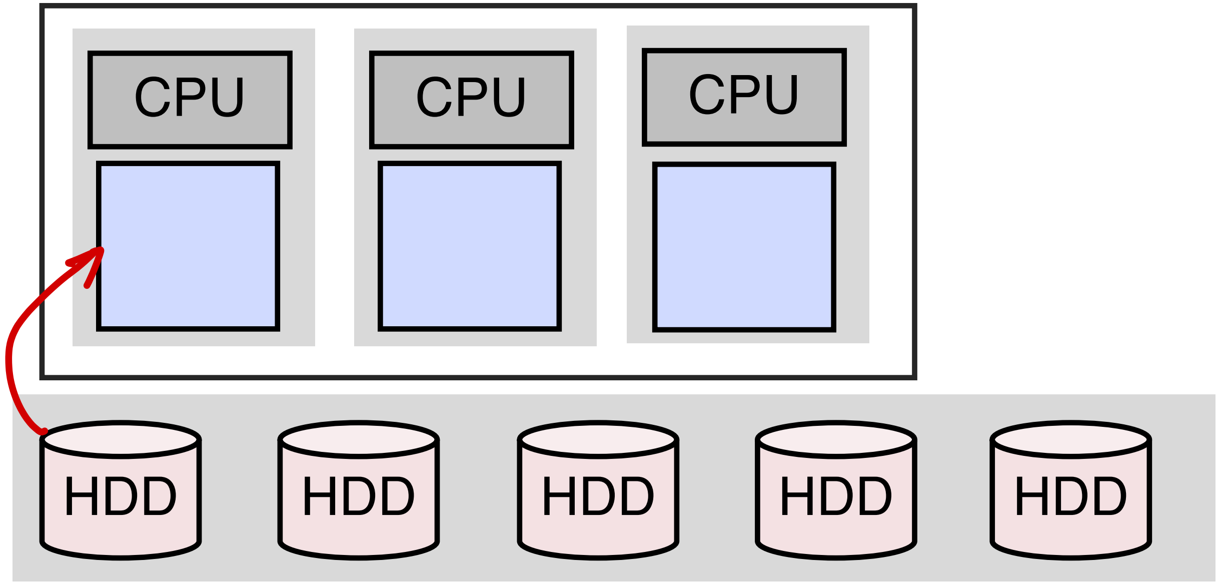
- Created, destroyed, or resized at any point (may shutdown all VWs)
- User may create multiple VWs for multiple queries
- Determine the VW size based on performance and cost requirements

Architecture – Virtual Warehouse

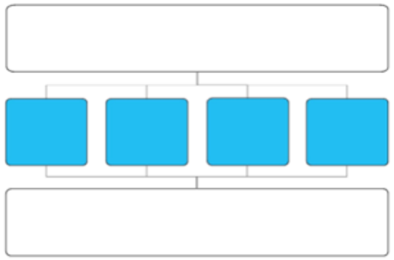


Local caching

- S3 data can be cached in local memory or disk



Architecture – Virtual Warehouse

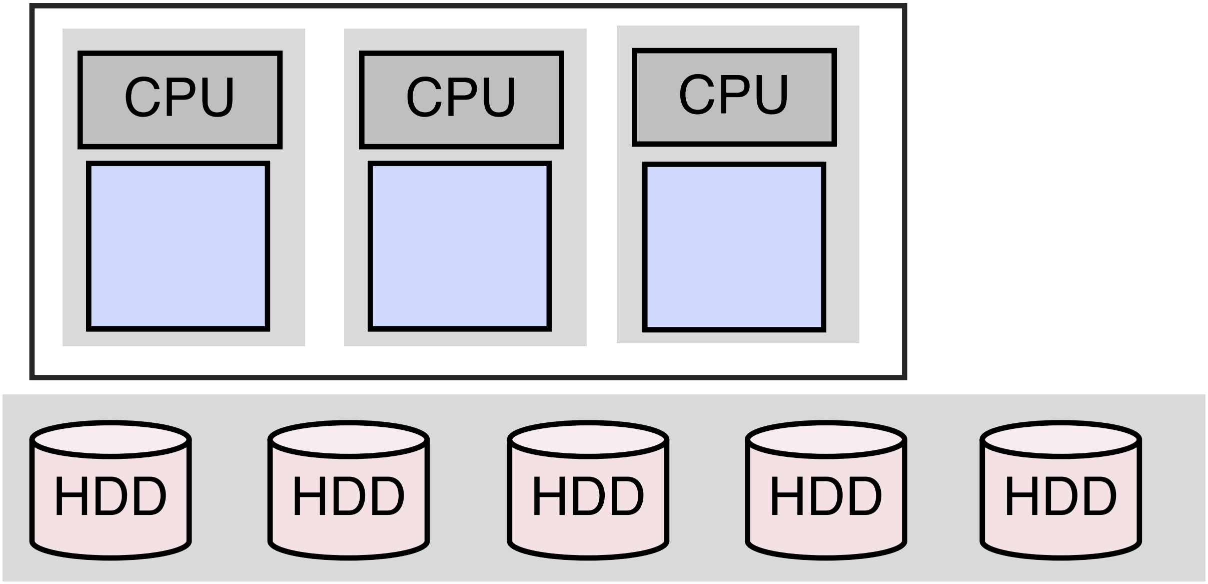


Local caching

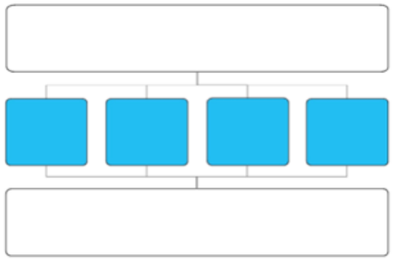
– S3 data can be cached in local memory or disk

Consistent hashing

- When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped



Architecture – Virtual Warehouse

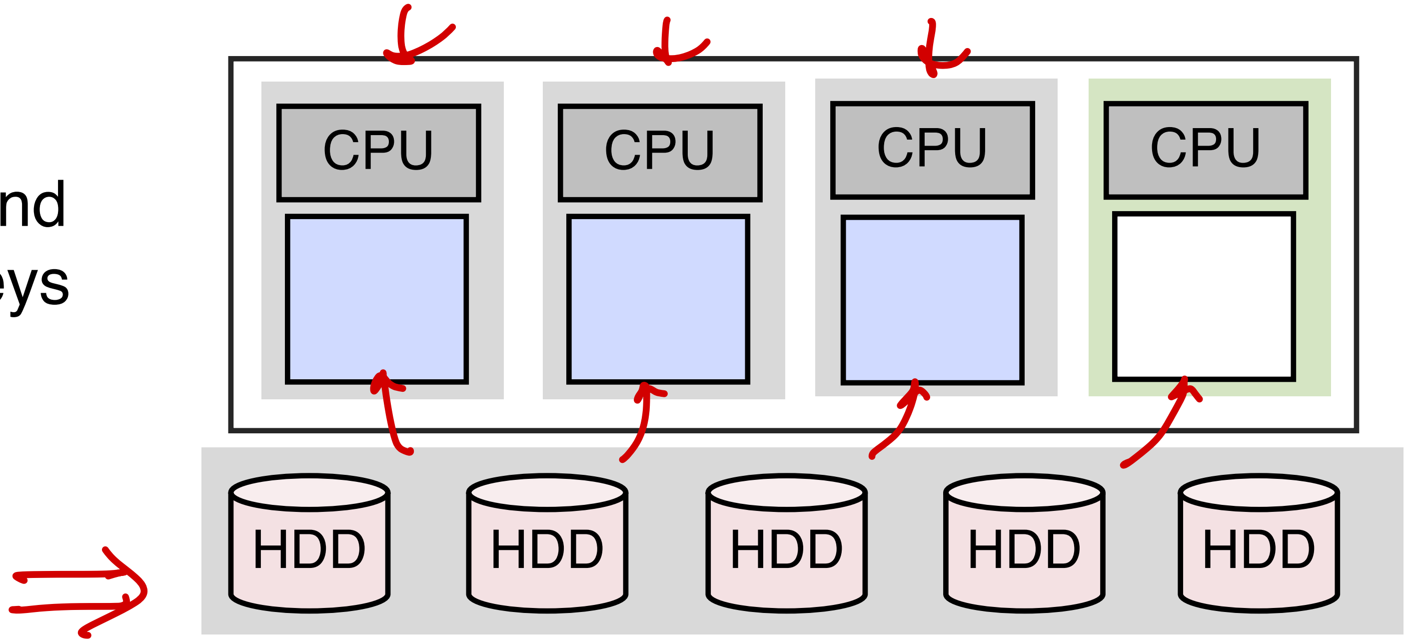


Local caching

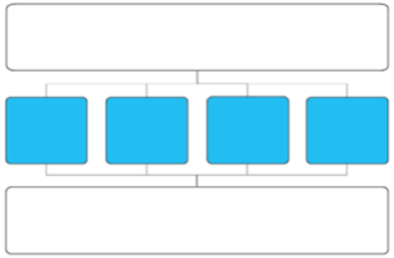
– S3 data can be cached in local memory or disk

Consistent hashing

- When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped



Architecture – Virtual Warehouse

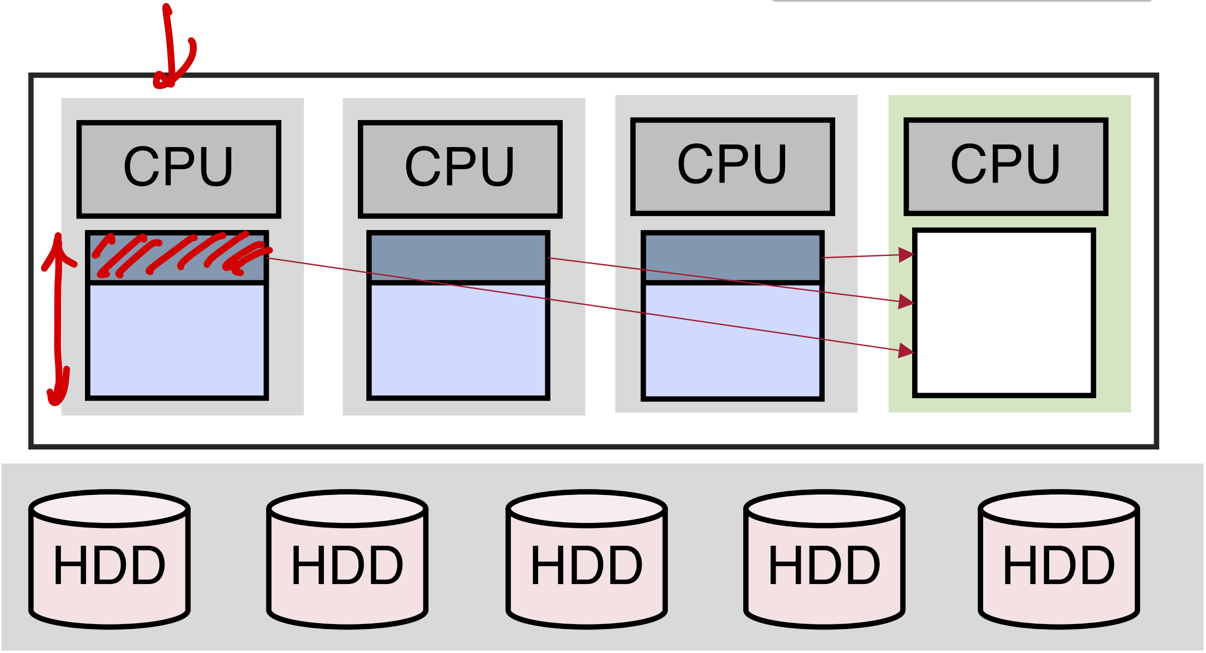


Local caching

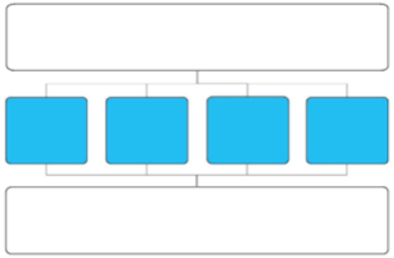
– S3 data can be cached in local memory or disk

Consistent hashing

- When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped



Architecture – Virtual Warehouse

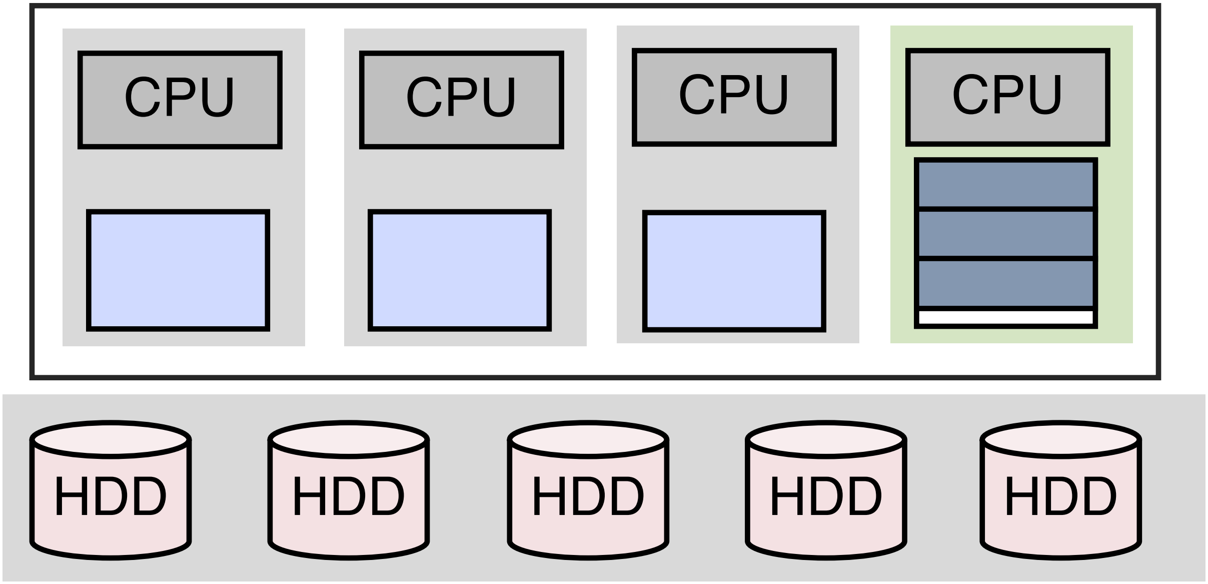


Local caching

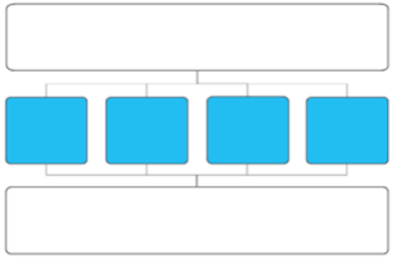
– S3 data can be cached in local memory or disk

Consistent hashing

- When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped



Architecture – Virtual Warehouse

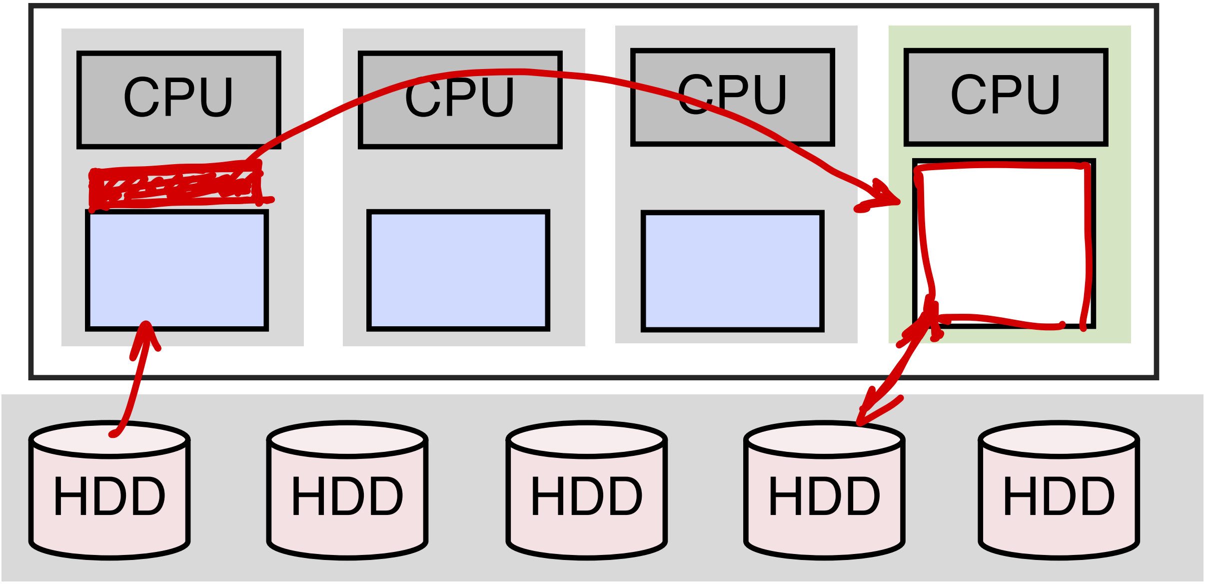


Local caching

– S3 data can be cached in local memory or disk

Consistent hashing

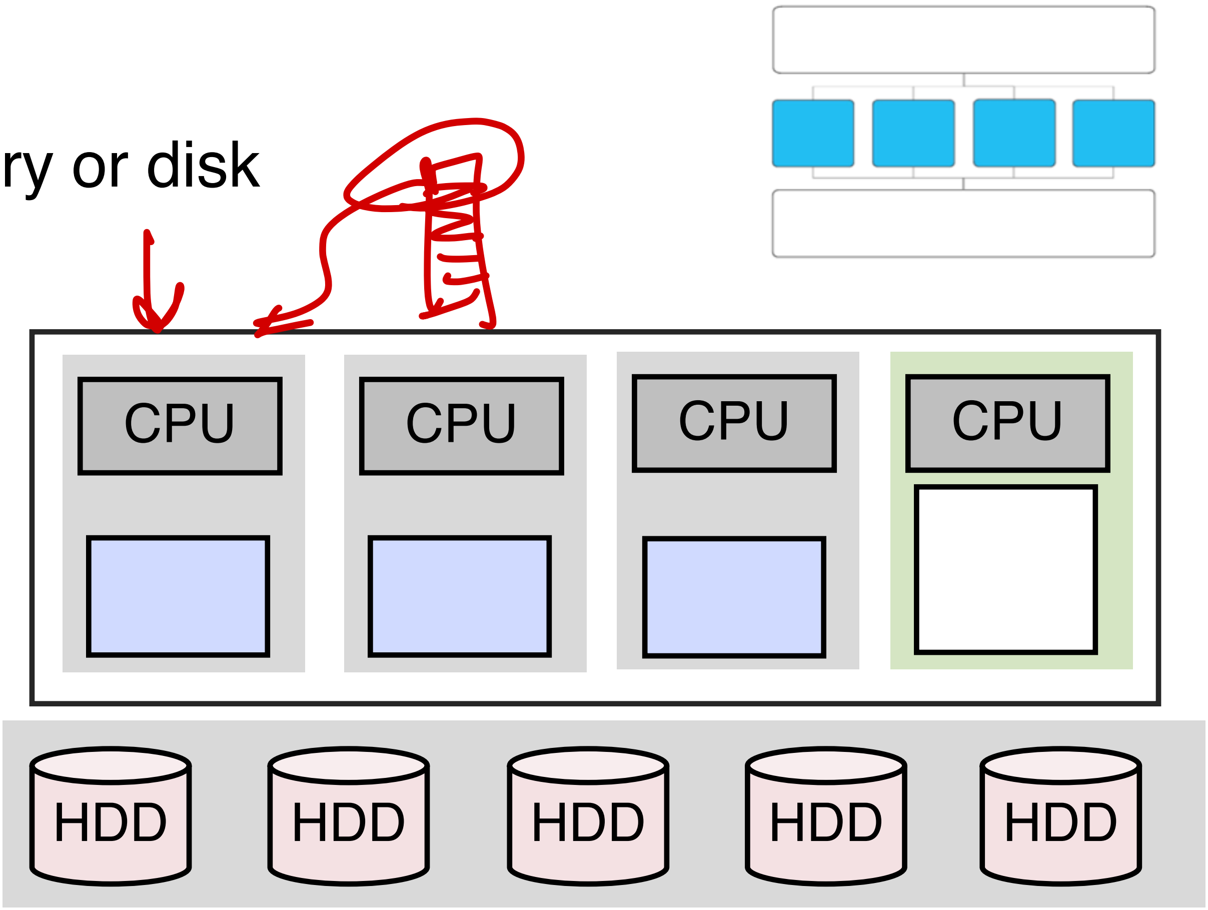
- When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped
- When a VW is resized, **no data shuffle required**; rely on LRU to replace cache content



Architecture – Virtual Warehouse

Local caching

– S3 data can be cached in local memory or disk

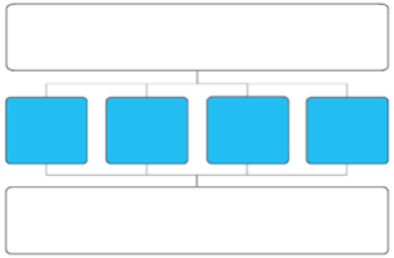


Consistent hashing

- 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

File stealing to tolerate skew

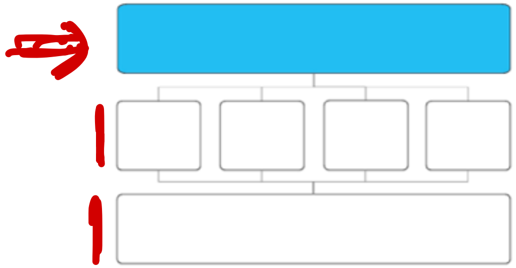
Architecture – Virtual Warehouse



Execution engine

- Columnar: SIMD, compression
- Vectorized: process a group of elements at a time
- Push-based

Architecture – Cloud Services



Multi-tenant layer shared across multiple users

Query optimization

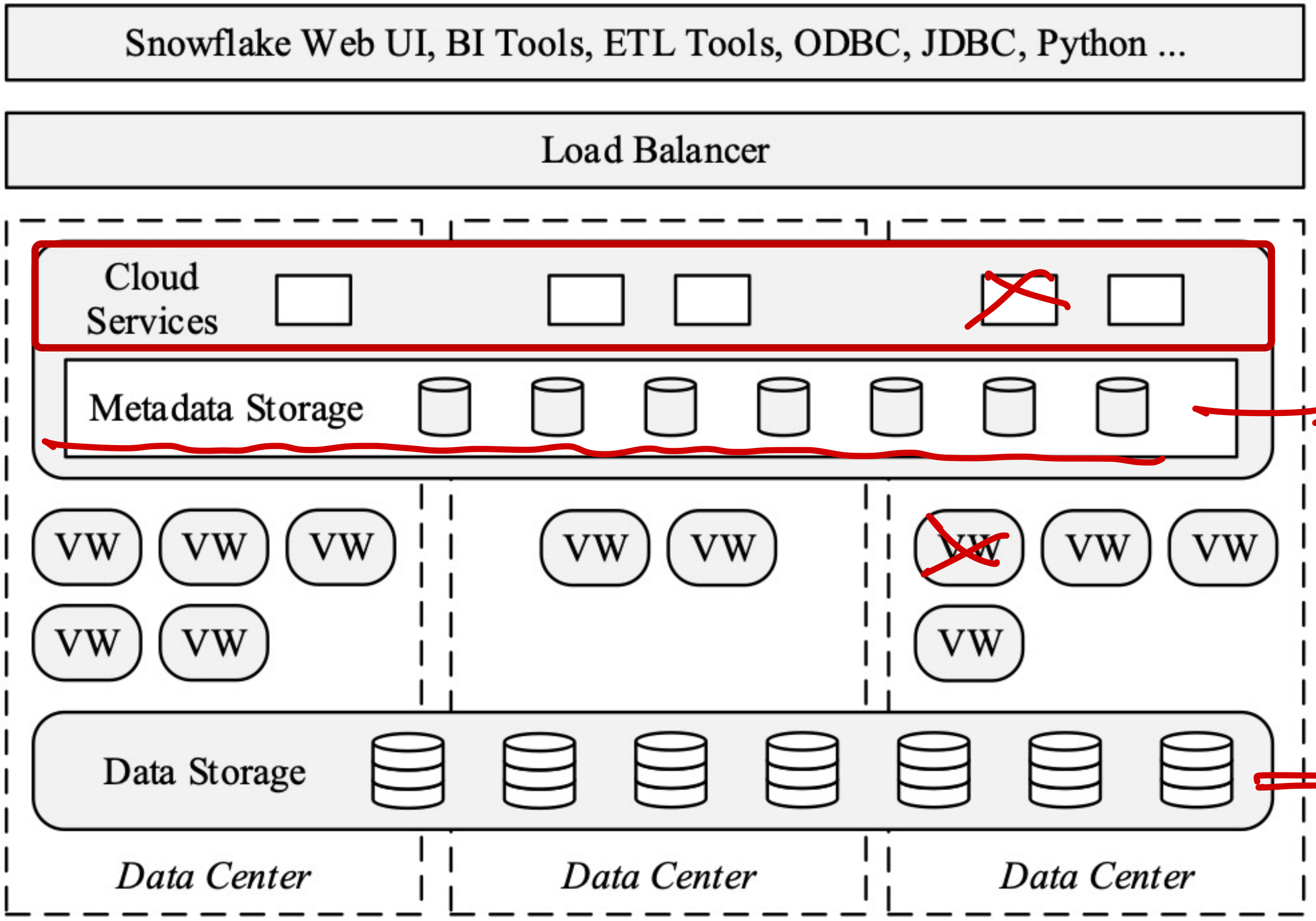
Concurrency control

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

Pruning Zonemap.

- 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

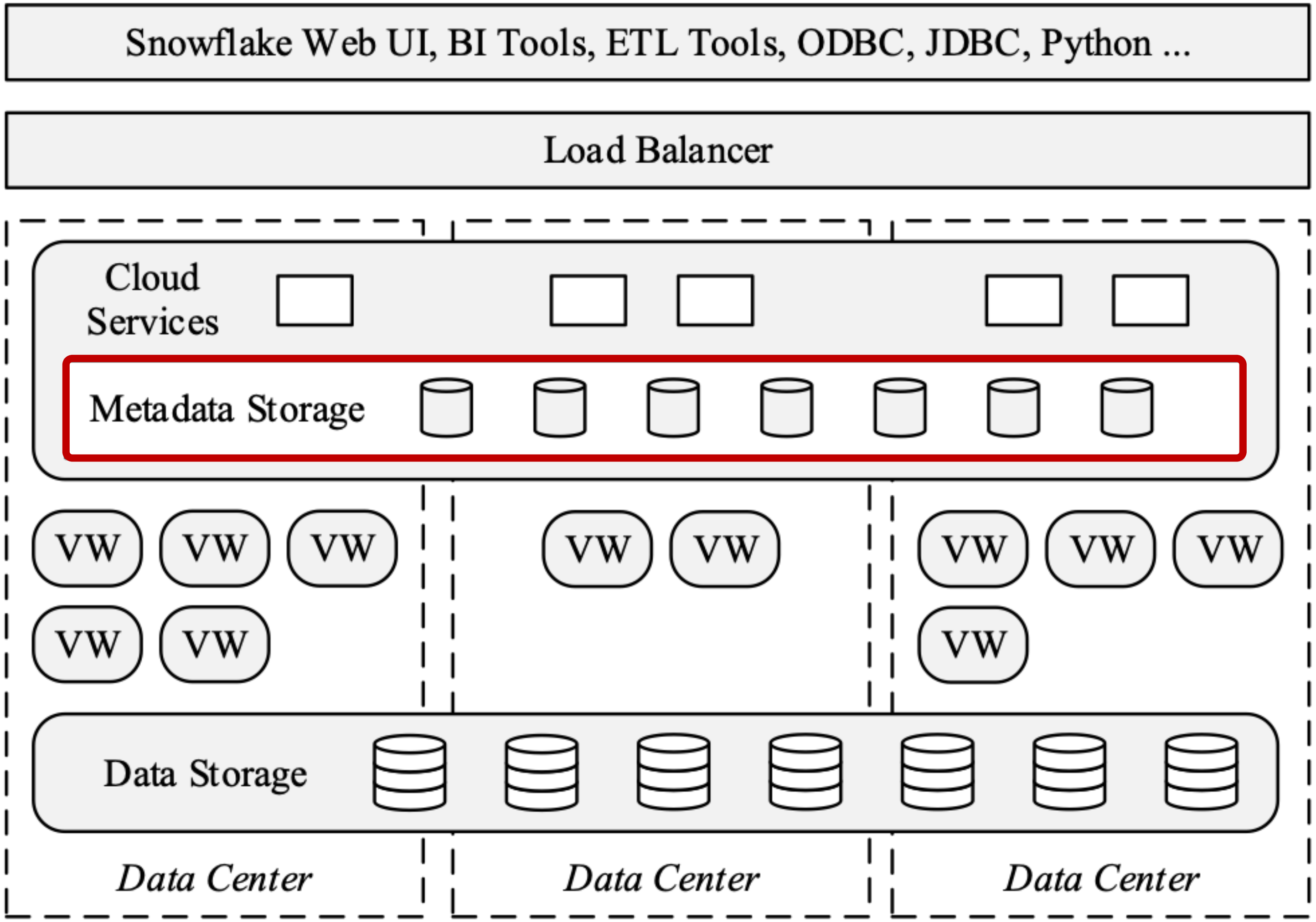


Stateless services

foundation DB.

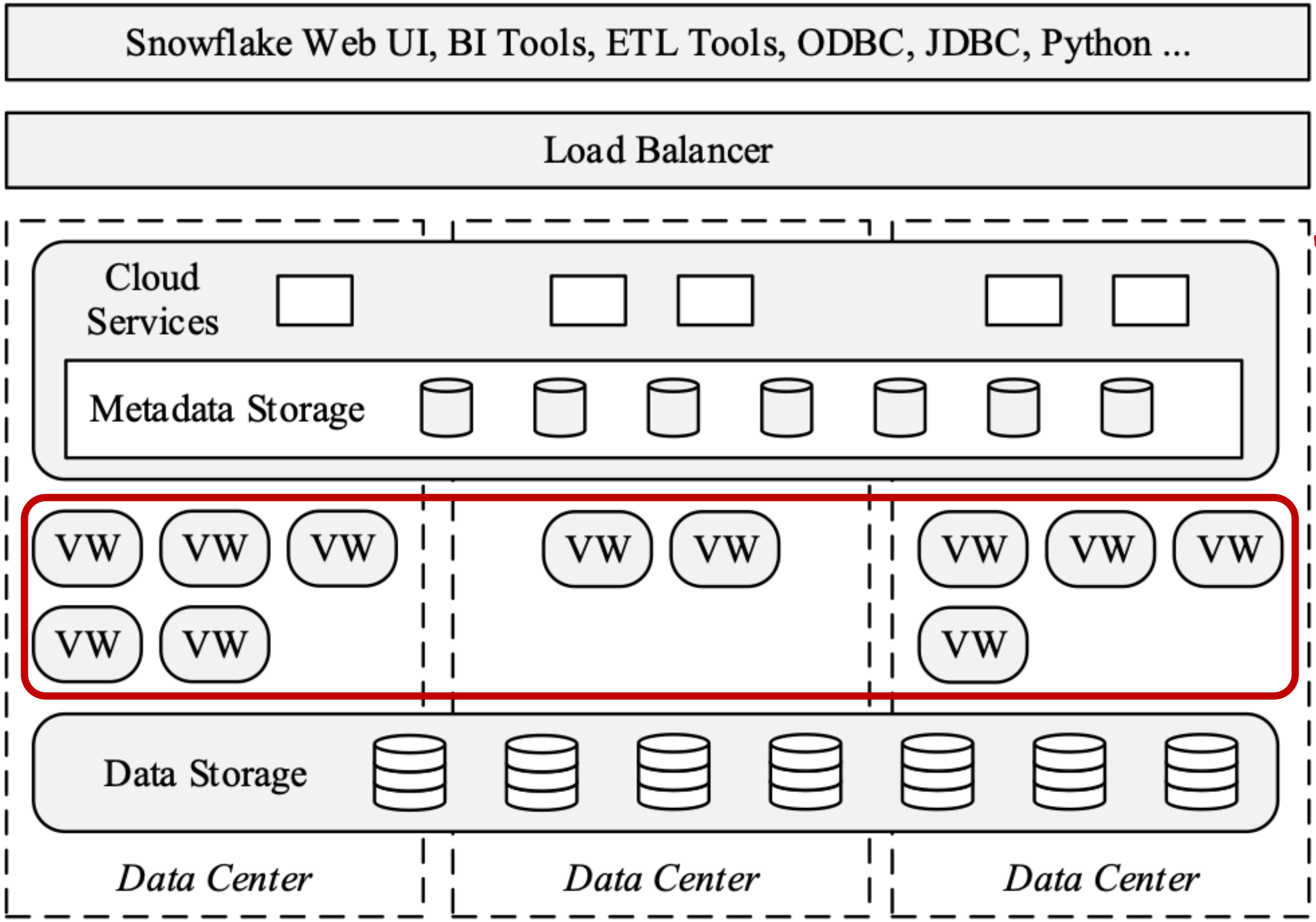
S3.

High Availability and Fault Tolerance



Replicated metadata (FoundationDB)

High Availability and Fault Tolerance

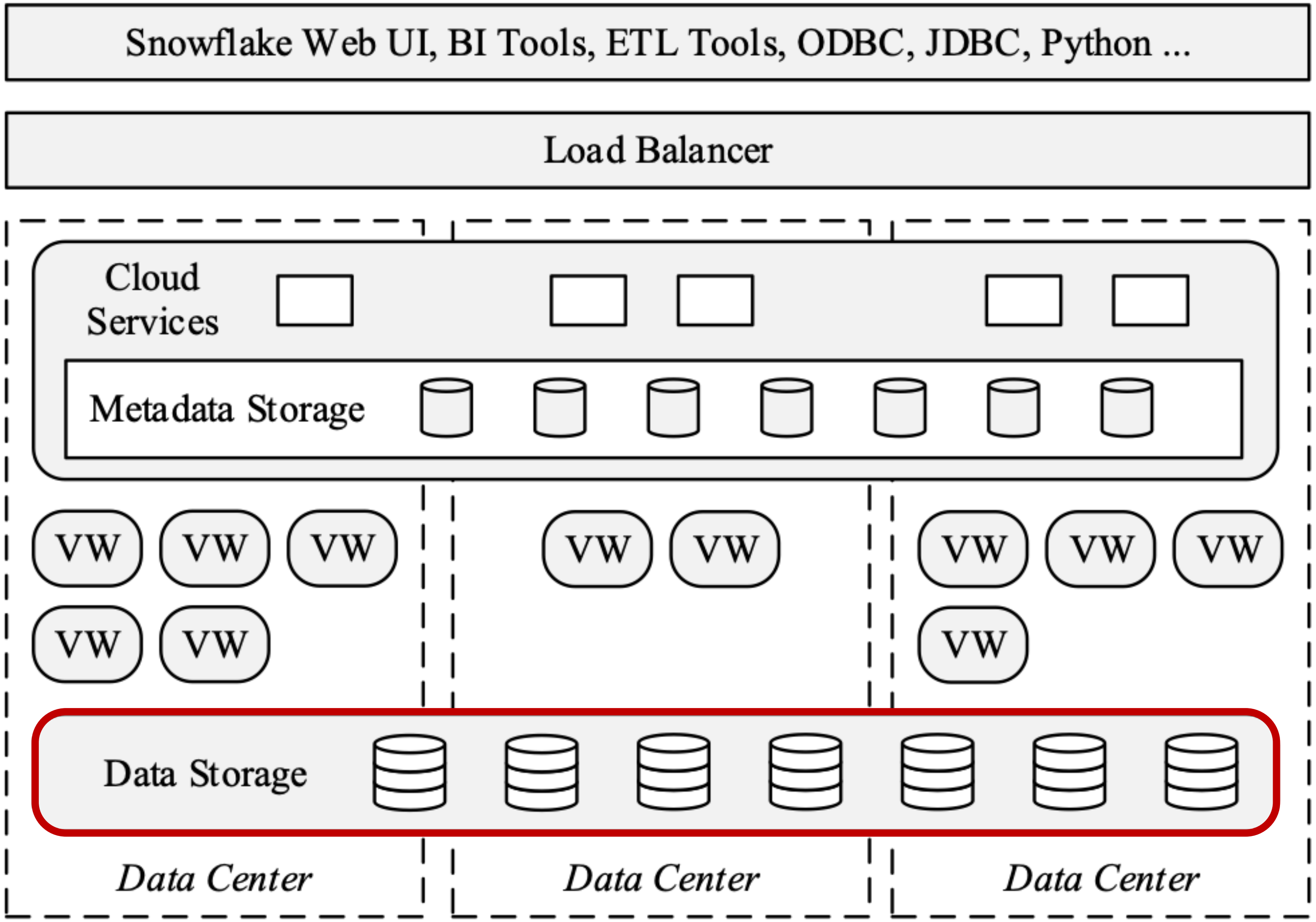


One node failure in VW
- Re-execute with failed node immediately replaced
- Re-execute with reduced number of nodes

Whole AZ failure
- Re-execute by re-provisioning a new VW

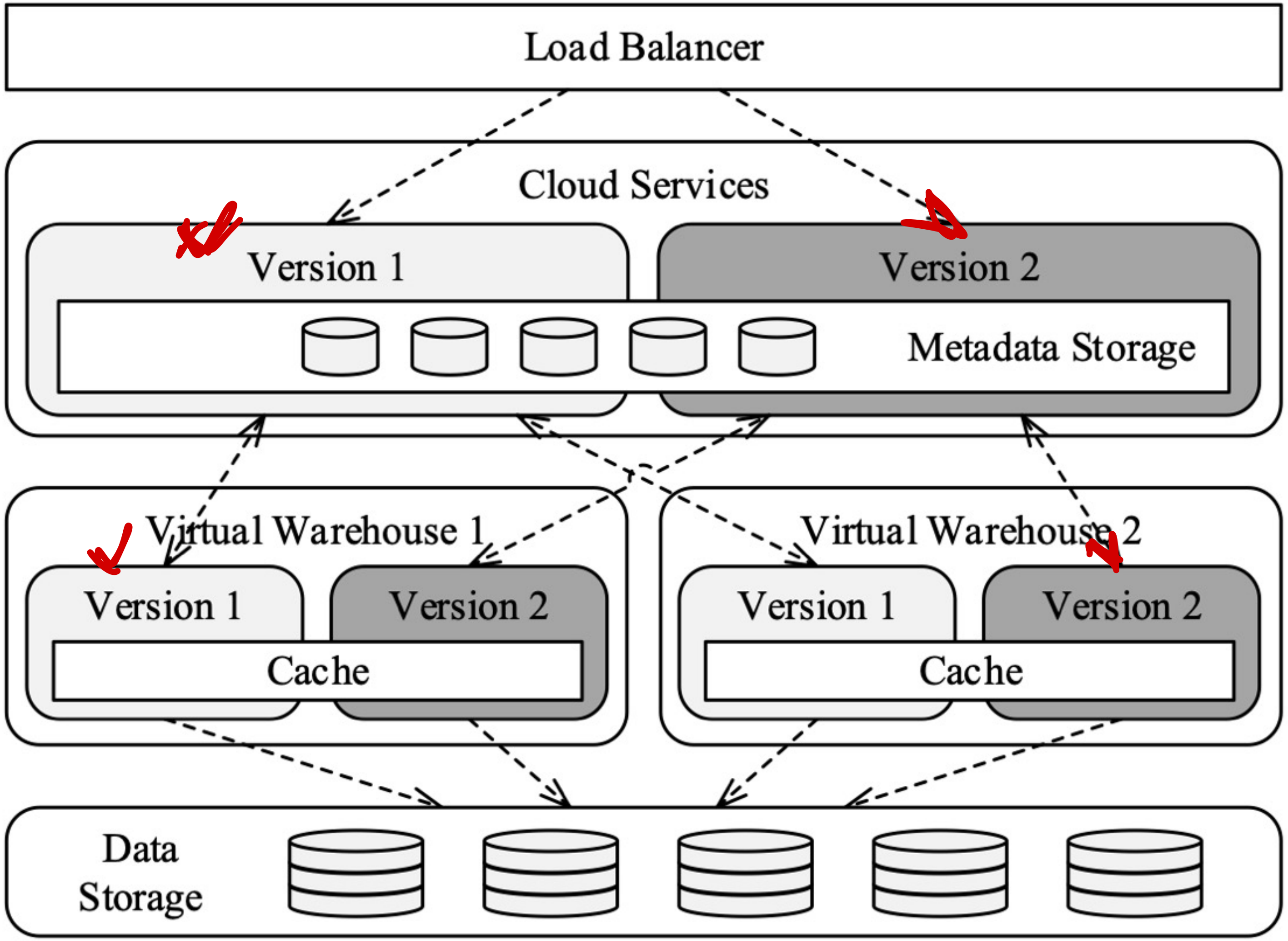
Hot-standby nodes

High Availability and Fault Tolerance



S3 is highly available and durable

Online Upgrade



Deploy new versions of services and VWs

Previous version terminates after active queries finish

Semi-Structured Data

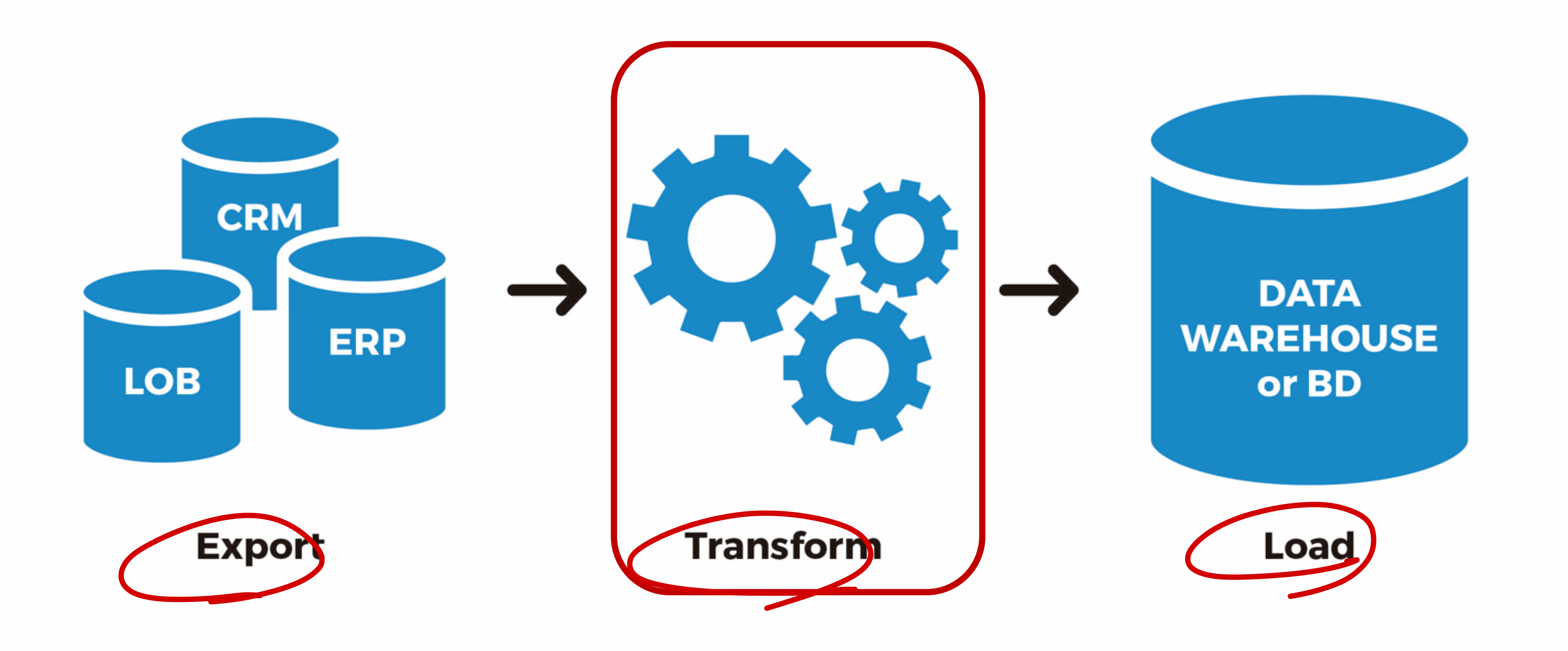
Extensible Markup Language (**XML**)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<customers>
    <customer>
        <customer_id>1</customer_id>
        <first_name>John</first_name>
        <last_name>Doe</last_name>
        <email>john.doe@example.com</email>
    </customer>
    <customer>
        <customer_id>2</customer_id>
        <first_name>Sam</first_name>
        <last_name>Smith</last_name>
        <email>sam.smith@example.com</email>
    </customer>
    <customer>
        <customer_id>3</customer_id>
        <first_name>Jane</first_name>
        <last_name>Doe</last_name>
        <email>jane.doe@example.com</email>
    </customer>
</customers>
```

JavaScript Object Notation(**JSON**)

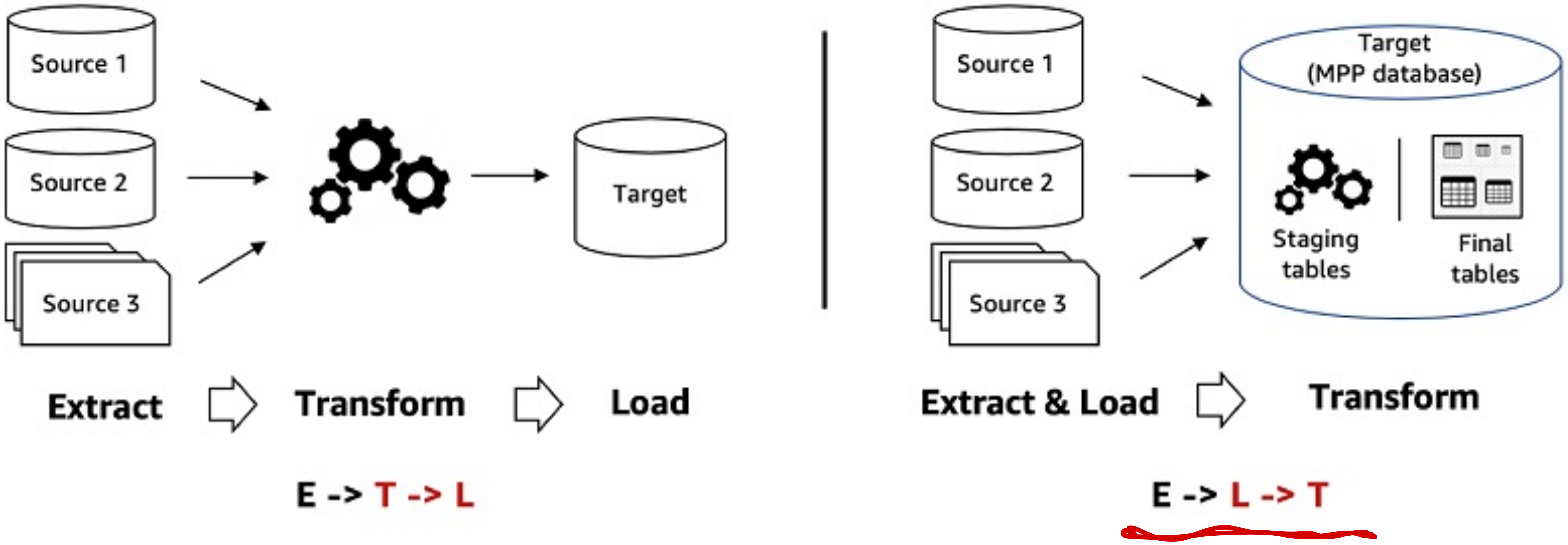
```json
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

Extract-Transform-Load (ETL)



Transform (e.g., converting to column format) adds latency to the system

ETL vs. ELT



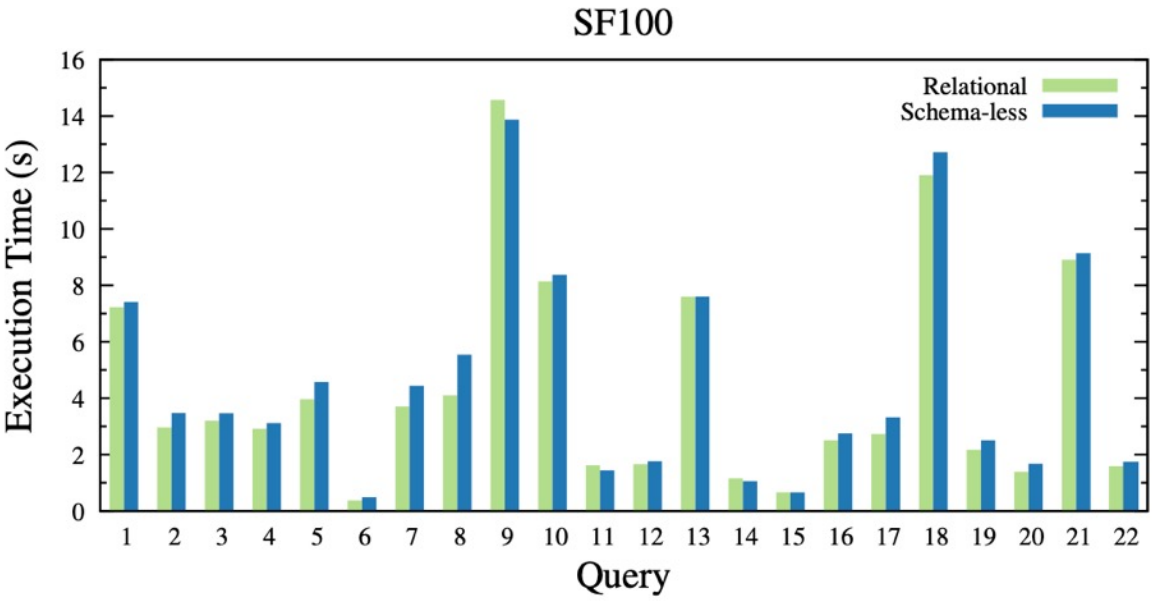
Picture from https://aws.amazon.com/blogs/big-data/etl-and-elt-design-patterns-for-lake-house-architecture-using-amazon-redshift-part-1/

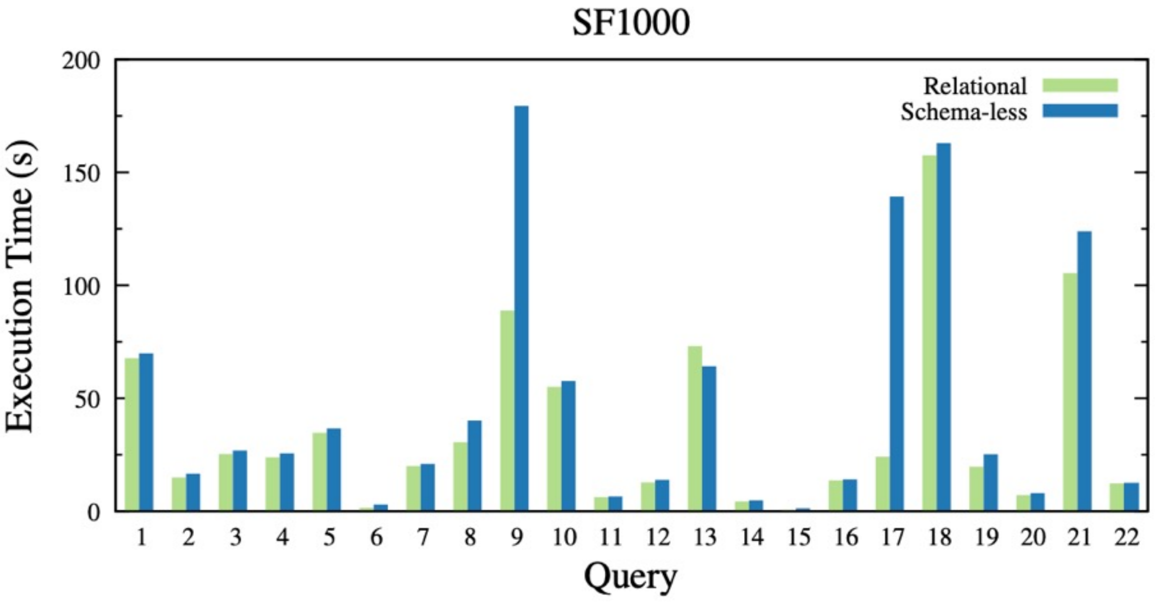
Optimization for Semi-Structured Data

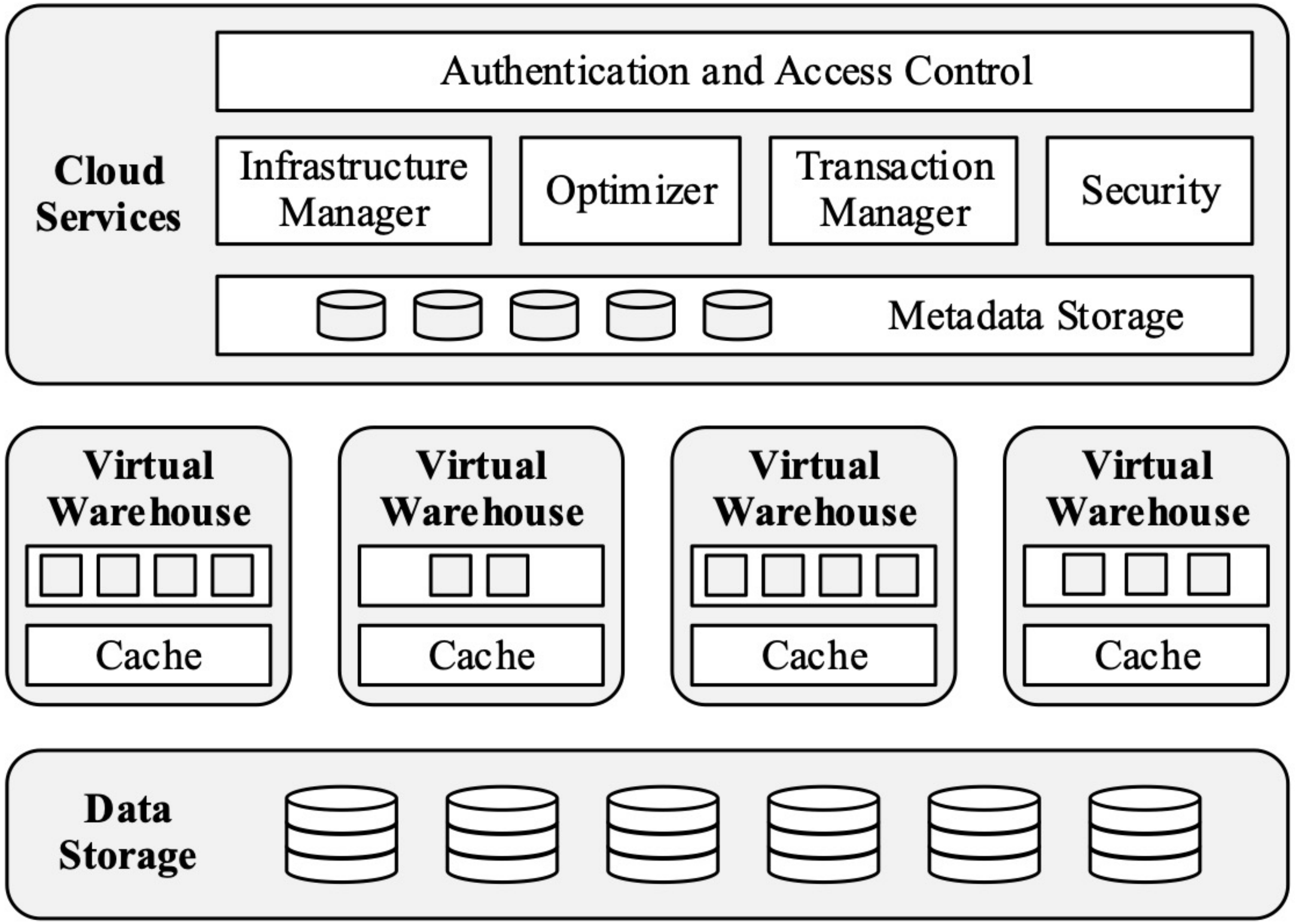
Automatic type inference

Hybrid columnar format

- Frequently paths are detected, projected out, and stored in separate columns in table file (typed and compressed)
- Collect metadata on these columns for optimization (e.g., pruning)

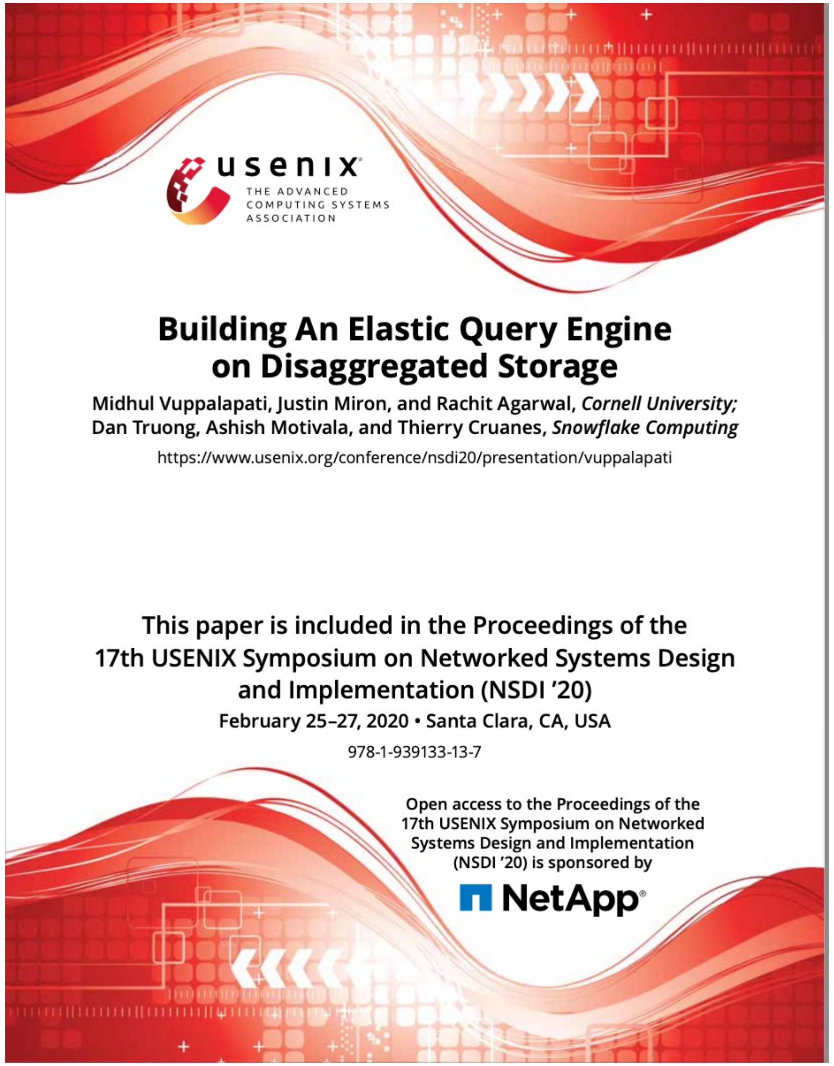






Q: What are the limitations of Snowflake's design?

A Follow-up Paper



usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Building An Elastic Query Engine on Disaggregated Storage

Midhul Vuppalapati, Justin Miron, and Rachit Agarwal, *Cornell University;* Dan Truong, Ashish Motivala, and Thierry Cruanes, *Snowflake Computing*

https://www.usenix.org/conference/nsdi20/presentation/vuppalapati

This paper is included in the Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)

February 25–27, 2020 • Santa Clara, CA, USA

978-1-939133-13-7

Open access to the Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20) is sponsored by

NetApp

NSDI 2020

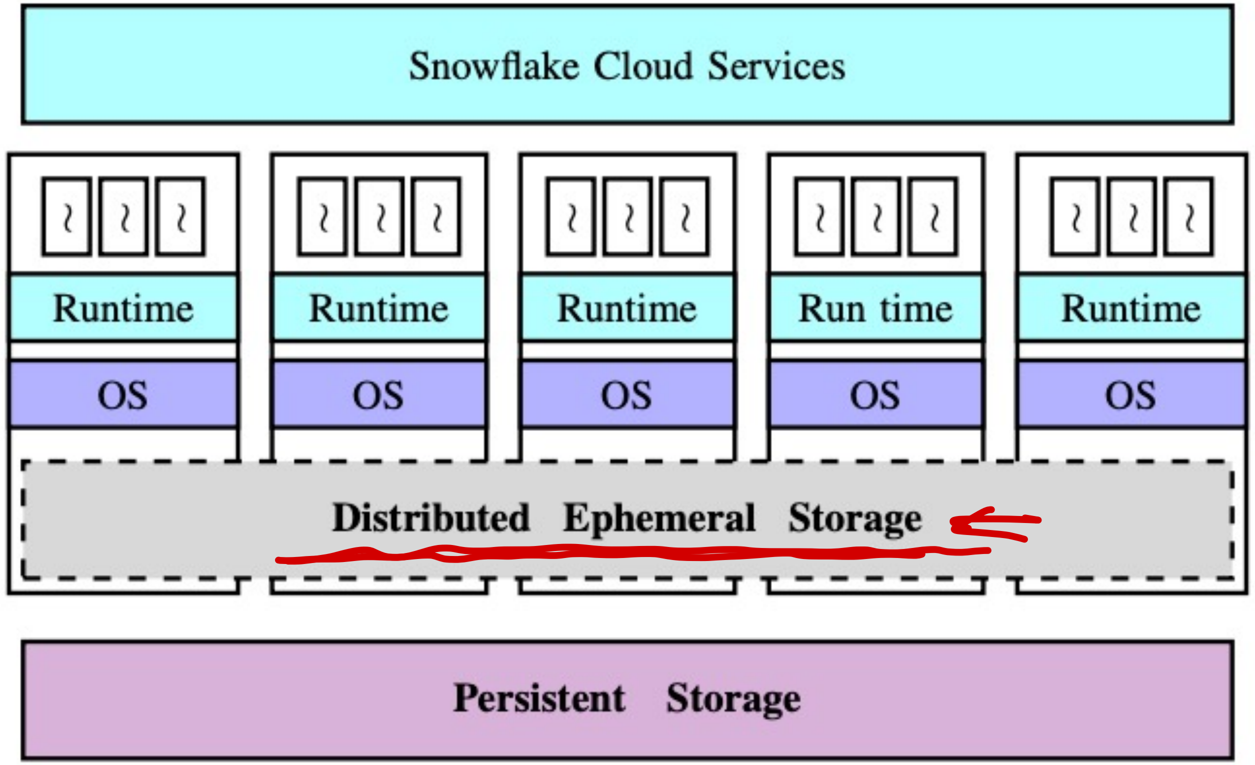
Limitations of current Snowflake design and potential research directions

- Decoupling of compute and ephemeral storage
- Deep storage hierarchy
- Pricing at sub-second timescales

Distributed Ephemeral Storage

Intermediate data is short-lived

- Need low-latency and high throughput
- Strong durability not needed
- Caching of intermediate data vs. persistent data
- Query scheduling: locality-aware task + work stealing



Lakehouse Architecture

Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics

Michael Armbrust[1], Ali Ghodsi[1,2], Reynold Xin[1], Matei Zaharia[1,3]
[1]Databricks, [2]UC Berkeley, [3]Stanford University

Abstract

This paper argues that the data warehouse architecture as we know it today will wither in the coming years and be replaced by a new architectural pattern, the Lakehouse, which will (i) be based on open direct-access data formats, such as Apache Parquet, (ii) have first-class support for machine learning and data science, and (iii) offer state-of-the-art performance. Lakehouses can help address several major challenges with data warehouses, including data staleness, reliability, total cost of ownership, data lock-in, and limited use-case support. We discuss how the industry is already moving toward Lakehouses and how this shift may affect work in data management. We also report results from a Lakehouse system using Parquet that is competitive with popular cloud data warehouses on TPC-DS.

1 Introduction

This paper argues that the data warehouse architecture as we know it today will wane in the coming years and be replaced by a new architectural pattern, which we refer to as the Lakehouse, characterized by (i) open direct-access data formats, such as Apache Parquet and ORC, (ii) first-class support for machine learning and data science workloads, and (iii) state-of-the-art performance.

The history of data warehousing started with helping business leaders get analytical insights by collecting data from operational databases into centralized warehouses, which then could be used for decision support and business intelligence (BI). Data in these warehouses would be written with schema-on-write, which ensured that the data model was optimized for downstream BI consumption. We refer to this as the first generation data analytics platforms.

A decade ago, the first generation systems started to face several challenges. First, they typically coupled compute and storage into an on-premises appliance. This forced enterprises to provision and pay for the peak of user load and data under management, which became very costly as datasets grew. Second, not only were datasets growing rapidly, but more and more datasets were completely unstructured, e.g., video, audio, and text documents, which data warehouses could not store and query at all.

To solve these problems, the second generation data analytics platforms started offloading all the raw data into *data lakes*: low-cost storage systems with a file API that hold data in generic and usually open file formats, such as Apache Parquet and ORC [8, 9]. This approach started with the Apache Hadoop movement [5], using the Hadoop File System (HDFS) for cheap storage. The data lake was a schema-on-read architecture that enabled the agility of storing any data at low cost, but on the other hand, punted the problem of data quality and governance downstream. In this architecture, a small subset of data in the lake would later be ETLed to a downstream data warehouse (such as Teradata) for the most important decision support and BI applications. The use of open formats also made data lake data directly accessible to a wide range of other analytics engines, such as machine learning systems [30, 37, 42].

From 2015 onwards, cloud data lakes, such as S3, ADLS and GCS, started replacing HDFS. They have superior durability (often >10 nines), geo-replication, and most importantly, extremely low cost with the possibility of automatic, even cheaper, archival storage, e.g., AWS Glacier. The rest of the architecture is largely the same in the cloud as in the second generation systems, with a downstream data warehouse such as Redshift or Snowflake. This two-tier data lake + warehouse architecture is now dominant in the industry in our experience (used at virtually all Fortune 500 enterprises).

This brings us to the challenges with current data architectures. While the cloud data lake and warehouse architecture is ostensibly cheap due to separate storage (e.g., S3) and compute (e.g., Redshift), a two-tier architecture is highly complex for users. In the first generation platforms, all data was ETLed from operational data systems directly into a warehouse. In today's architectures, data is first ETLed into lakes, and then again ELTed into warehouses, creating complexity, delays, and new failure modes. Moreover, enterprise use cases now include advanced analytics such as machine learning, for which *neither* data lakes nor warehouses are ideal. Specifically, today's data architectures commonly suffer from four problems:

Reliability. Keeping the data lake and warehouse consistent is difficult and costly. Continuous engineering is required to ETL data between the two systems and make it available to high-performance decision support and BI. Each ETL step also risks incurring failures or introducing bugs that reduce data quality, e.g., due to subtle differences between the data lake and warehouse engines.

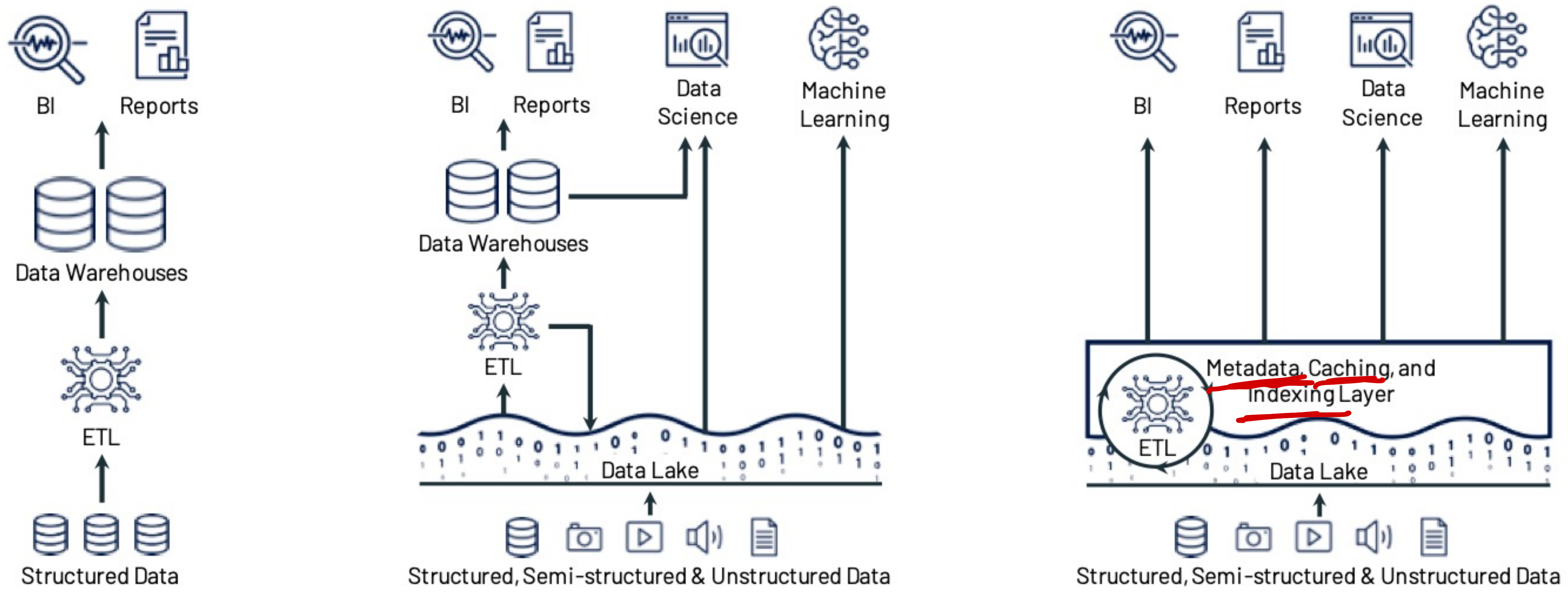
Data staleness. The data in the warehouse is stale compared to that of the data lake, with new data frequently taking days to load. This is a step back compared to the first generation of analytics systems, where new operational data was immediately available for queries. According to a survey by Dimensional Research and Fivetran, 86% of analysts use out-of-date data and 62% report waiting on engineering resources numerous times per month [47].

Limited support for advanced analytics. Businesses want to ask predictive questions using their warehousing data, e.g., "which customers should I offer discounts to?" Despite much research on the confluence of ML and data management, none of the leading machine learning systems, such as TensorFlow, PyTorch and XGBoost, work well on top of warehouses. Unlike BI queries, which extract a small amount of data, these systems need to process large datasets using complex non-SQL code. Reading this data via ODBC/JDBC is inefficient, and there is no way to directly access the internal

 11th Annual Conference on Innovative Data Systems Research (CIDR '21), January 11–15, 2021, Online.

CIDR 2021

Data Warehouse vs. Data Lake



(a) First-generation platforms. **(b) Current two-tier architectures.** **(c) Lakehouse platforms.**

Lakehouse = Data warehouse + data lake

Snowflake – Q/A

Key differences between Snowflake vs. Databricks?

Can we use cloud functions instead of VMs? Starling

Can Snowflake work with other backend like Azure?

Does Snowflake support transactional workloads?

Does spawning VWs for every query incur considerable overhead?

Before Next Lecture

Submit review for

- Yifei Yang, et al., FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS. VLDB, 2021