WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

CS 839: Topics in Database Management Systems

Lecture 3: Snowflake

Xiangyao Yu
9/13/2023

Presentation Signup

Too many people signed up for

- Auto-scaling
- Multi-cloud

Please either consider another topic or identify a paper/article in “Paper Signup” sheet **by next Monday**

Need a few more presenters for

- Storage disaggregation for OLAP
- Storage disaggregation for OLTP
- Auto-tuning
- Memory disaggregation

Please sign up for specific papers/articles

- If you are presenting “**disaggregation for OLAP**”. Pick your paper **today!**
- For other topics, try to pick your paper asap

No need to submit review for the lecture where you will present

Group Discussion Summary

Replay in compute node vs. storage node?

	Traditional	**Aurora**	**Log replay as a Service**
Advantages	Simpler design	• Less network traffic • Higher availability (DB server as single point of failure)?	Replay traffic spread across storage nodes
Disadvantages	Replay traffic through a single primary instance	• Storage needs a CPU and is more expensive; • Log replay is a potential bottleneck • Consumes more resources overall	Log service can be a bottleneck

Group Discussion Summary

Multi-master

- Must deal with conflicts across master nodes -> complex concurrency control (potentially happen inside the storage)
- Increased network traffic
- Deals with hotspot better
- More available since some of the masters can fail
- Potentially lower latency (e.g., skewed accesses)

Data partitioning

- Distributed transactions have higher latency due to coordination
- Hard to balance hotspot
- More scalable

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

© 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.

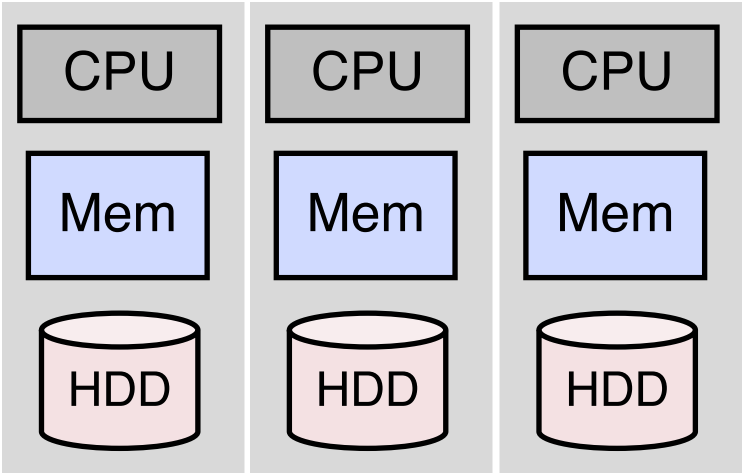
But not only the platform has changed. Data has changed as well. It used to be the case that most of the data in a data warehouse came from sources within the organization: transactional systems, enterprise resource planning (ERP) applications, customer relationship management (CRM) applications, and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

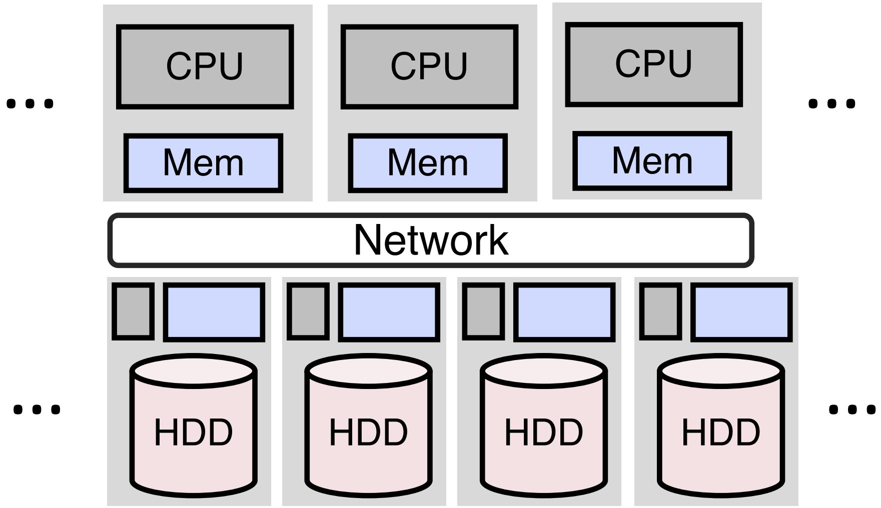
In response to these shortcomings, parts of the data warehousing community have turned to "Big Data" platforms such as Hadoop or Spark [8, 11]. While these are indispensable tools for data center-scale processing tasks, and the open source community continues to make big improvements such as the Stinger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

SIGMOD 2016

Shared-nothing vs. Storage-disaggregation





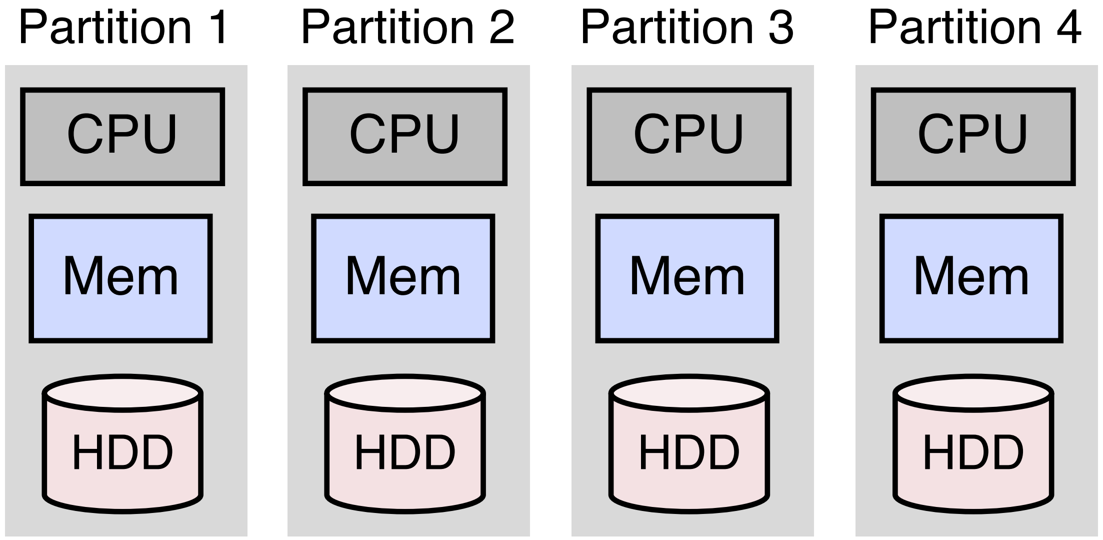
Shared-nothing

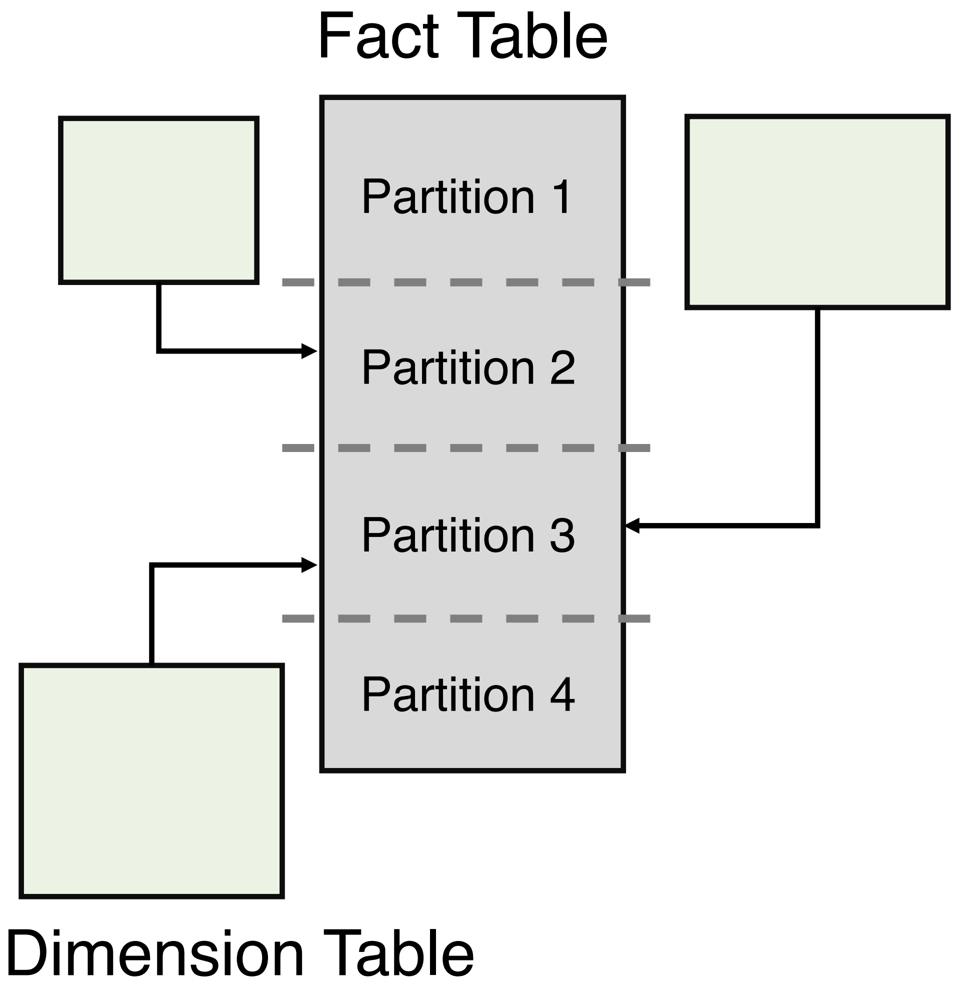
- Fixed and limited hardware resources

Storage disaggregation

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

Shared Nothing – Advantages

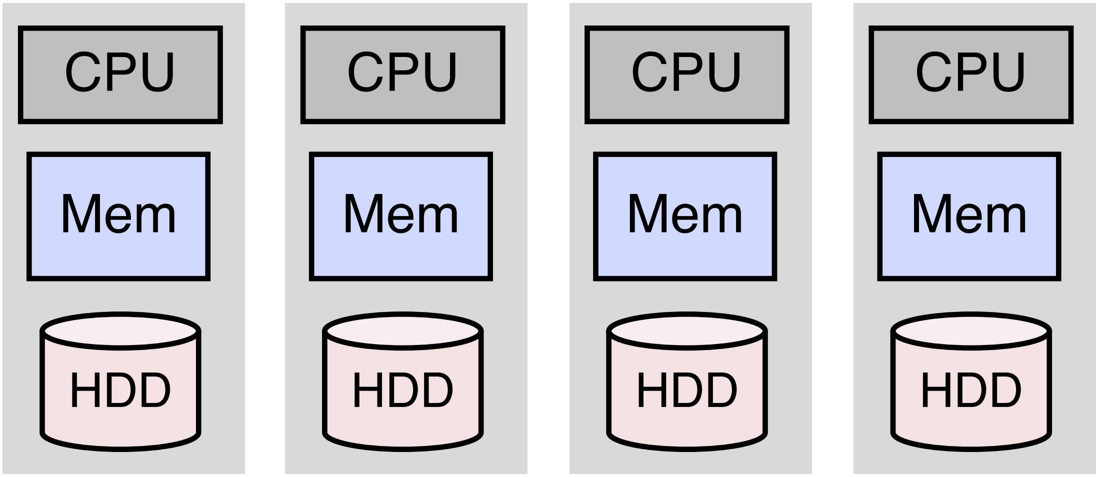


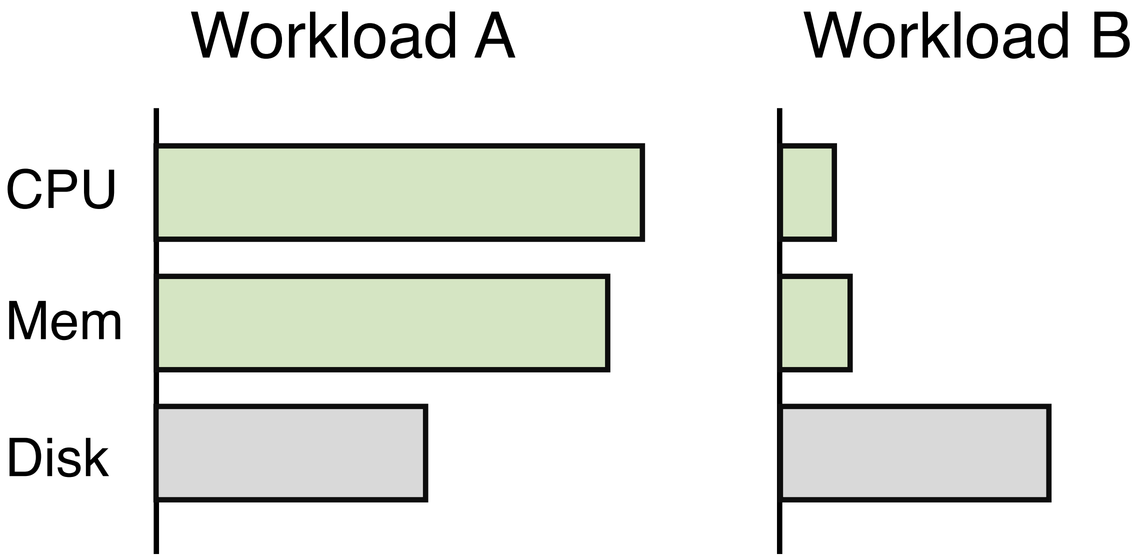


Scalability: horizontal scaling

- Scales well for star-schema queries

Shared Nothing – Disadvantages





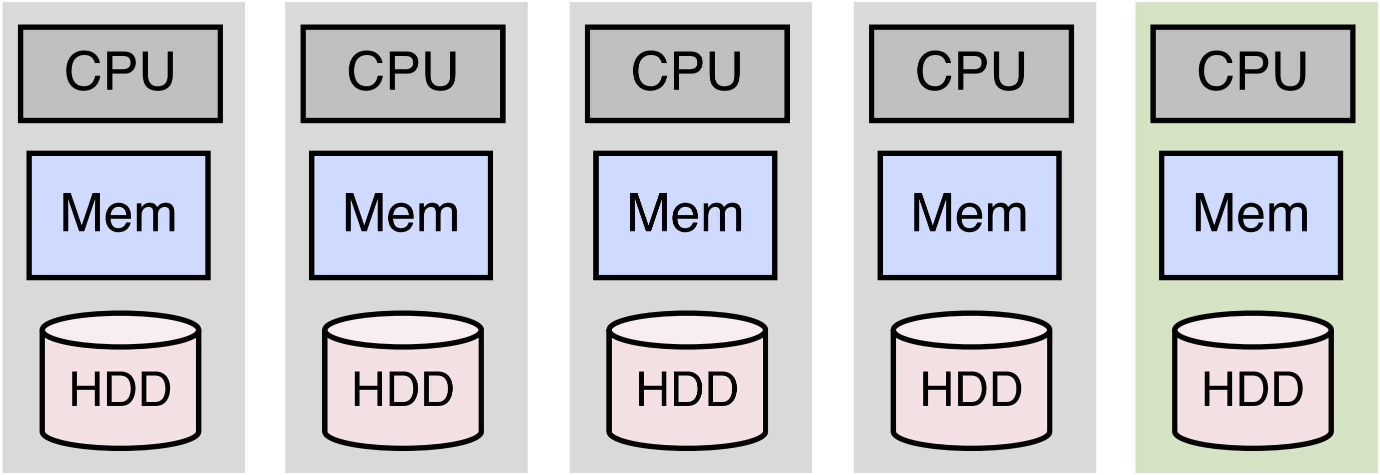
More CPU intensive

Less CPU intensive

Heterogeneous workload

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

Shared Nothing – Disadvantages

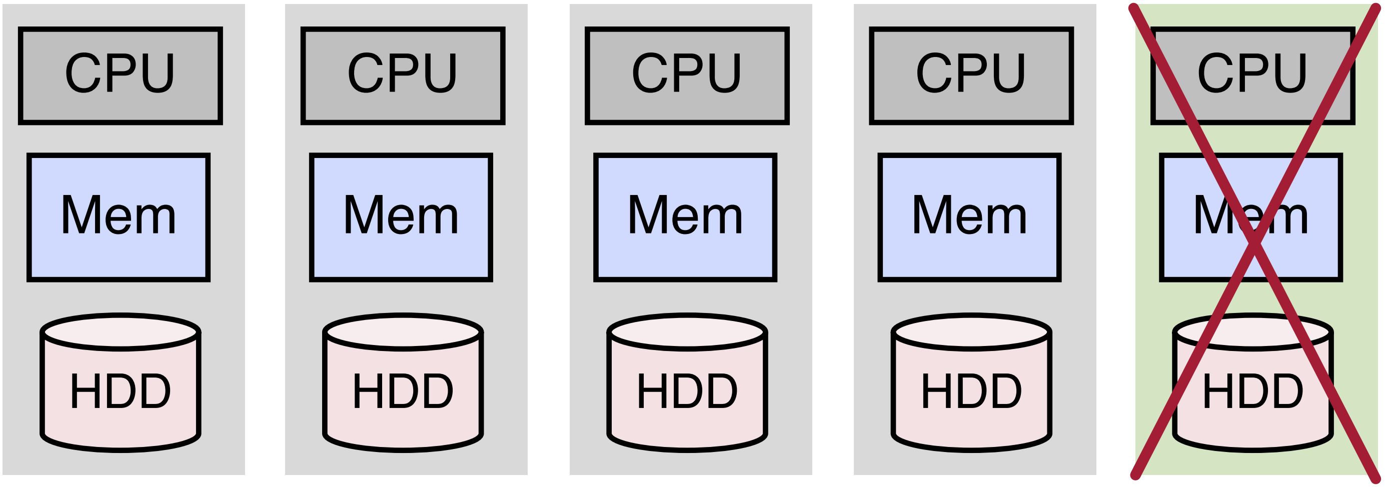


Heterogeneous workload

Membership changes

- Add a node: data redistribution

Shared Nothing – Disadvantages

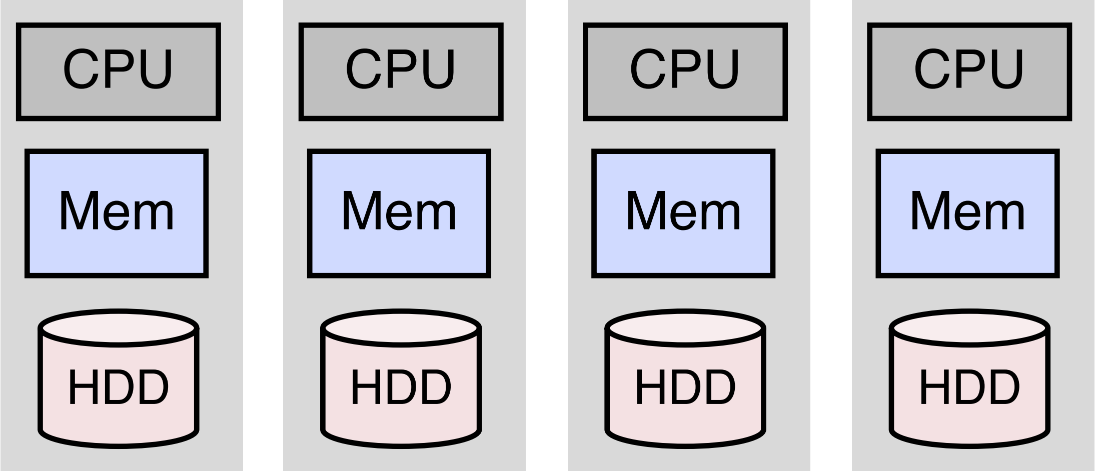


Heterogeneous workload

Membership changes

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

Shared Nothing – Disadvantages



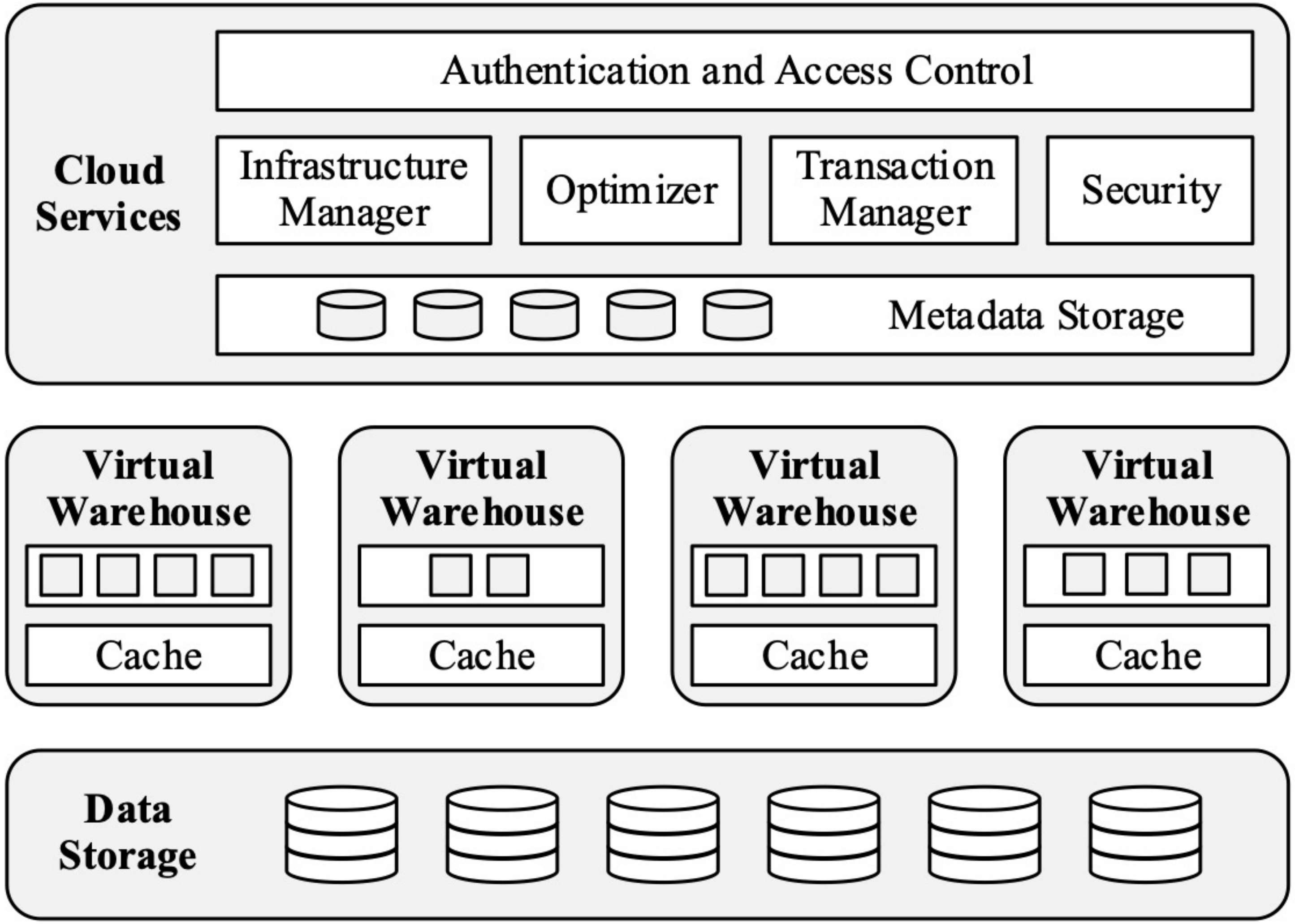
Heterogeneous workload

Membership changes

Online upgrade

- Similar to membership change but affect all nodes

Multi-Cluster Shared-Data Architecture



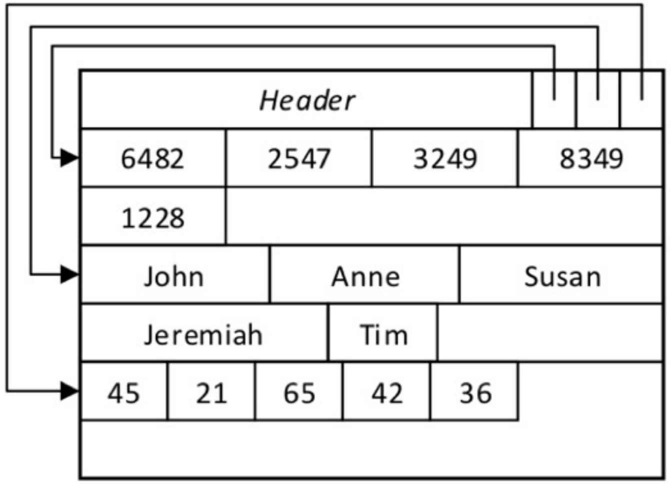
Control layer

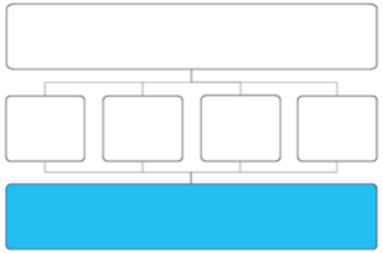
Compute layer

Storage layer

Architecture – Storage

Data format: PAX



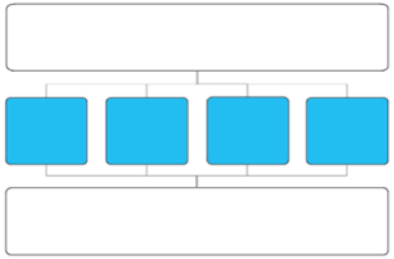


Data horizontally partitioned into immutable files (~16MB)

- 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

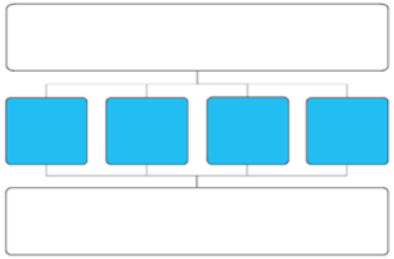


T-Shirt sizes: XS to 4XL

Elasticity and Isolation

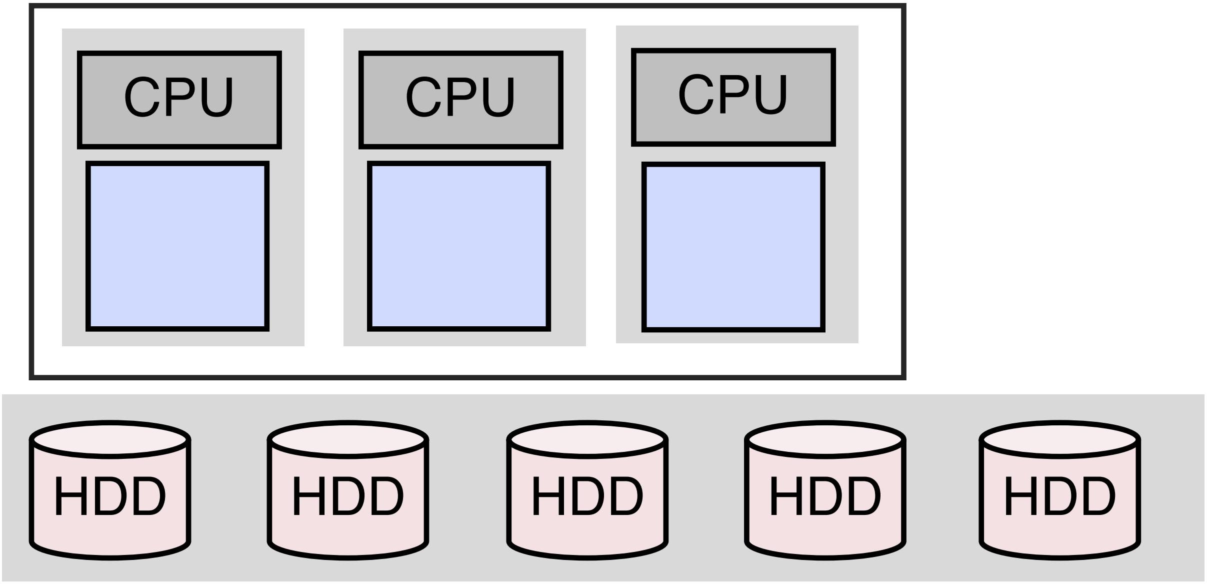
- Created, destroyed, or resized at any point (may shutdown all VWs)
- User may create multiple VWs for multiple queries
- Determine the VW size based on performance and cost requirements

Architecture – Virtual Warehouse

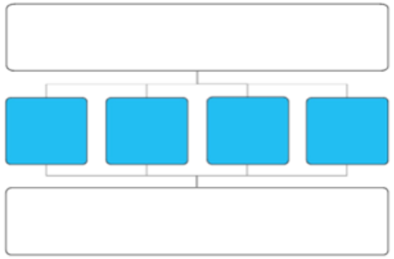


Local caching

- S3 data can be cached in local memory or disk



Architecture – Virtual Warehouse

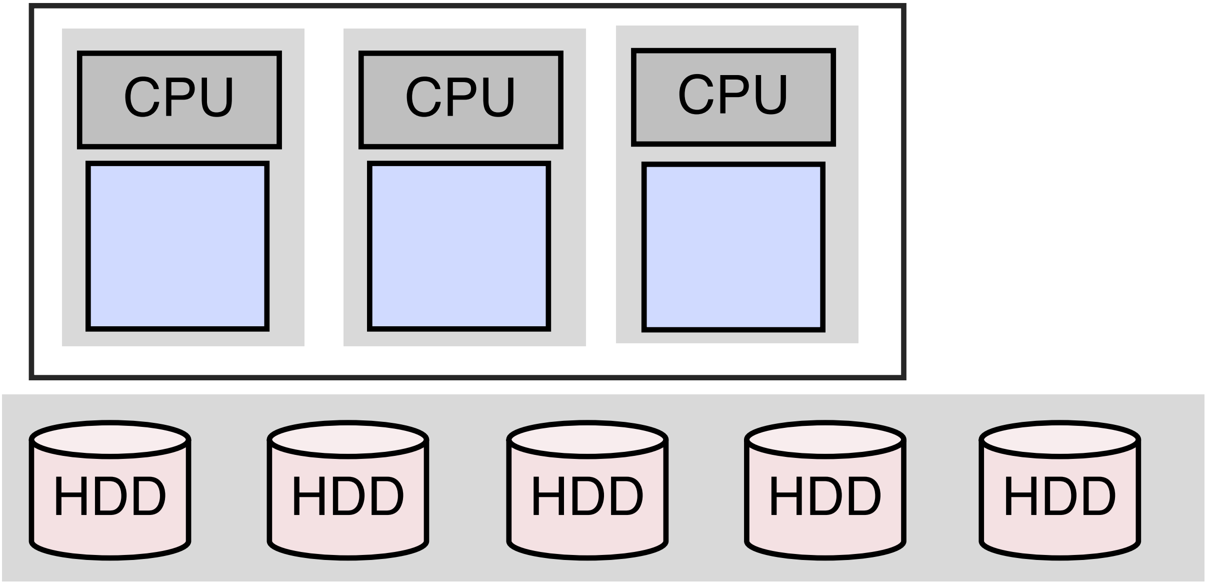


Local caching

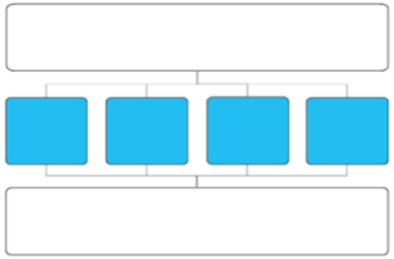
- S3 data can be cached in local memory or disk

Consistent hashing

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



Architecture – Virtual Warehouse

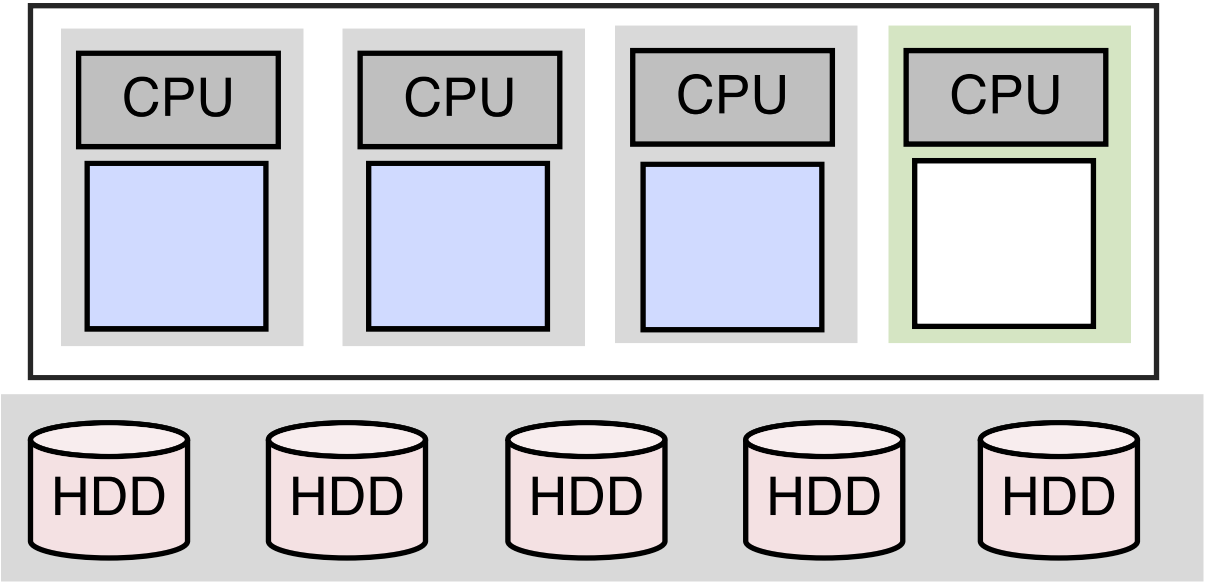


Local caching

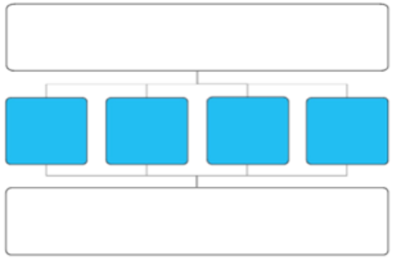
- S3 data can be cached in local memory or disk

Consistent hashing

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



Architecture – Virtual Warehouse

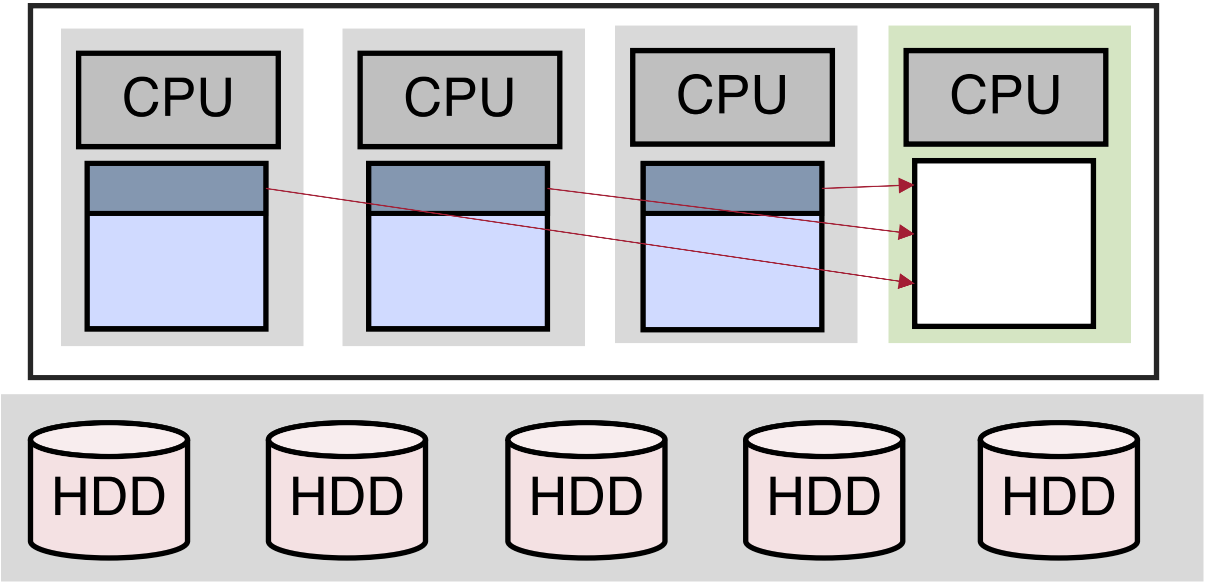


Local caching

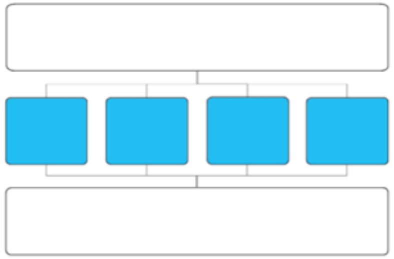
- S3 data can be cached in local memory or disk

Consistent hashing

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



Architecture – Virtual Warehouse

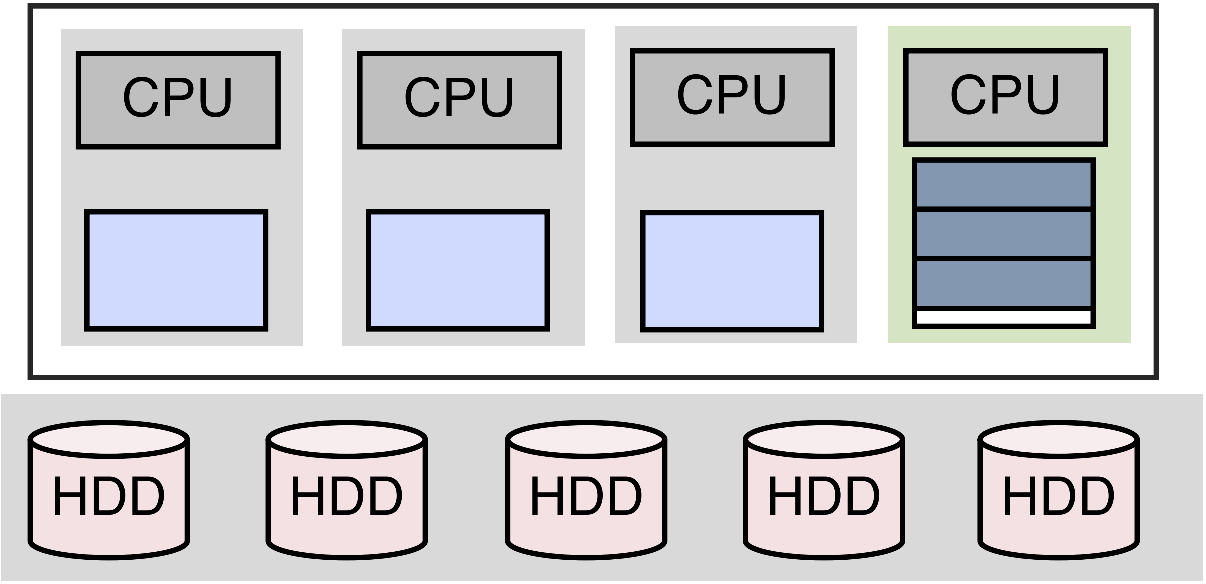


Local caching

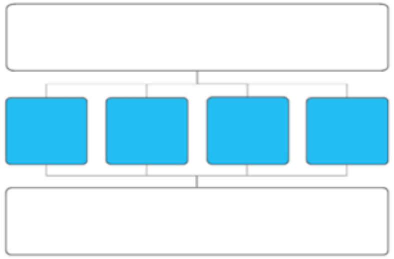
- S3 data can be cached in local memory or disk

Consistent hashing

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



Architecture – Virtual Warehouse

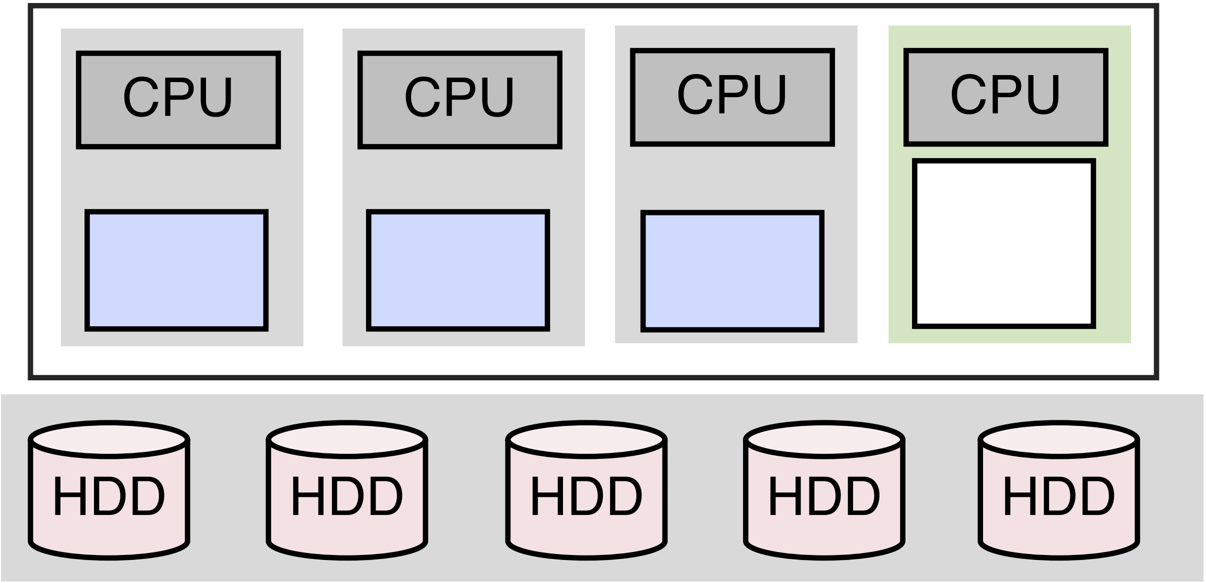


Local caching

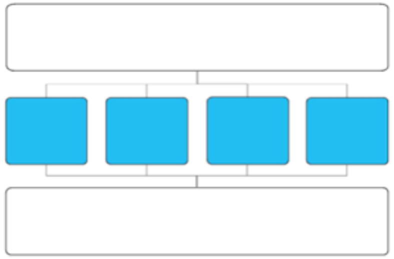
- S3 data can be cached in local memory or disk

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

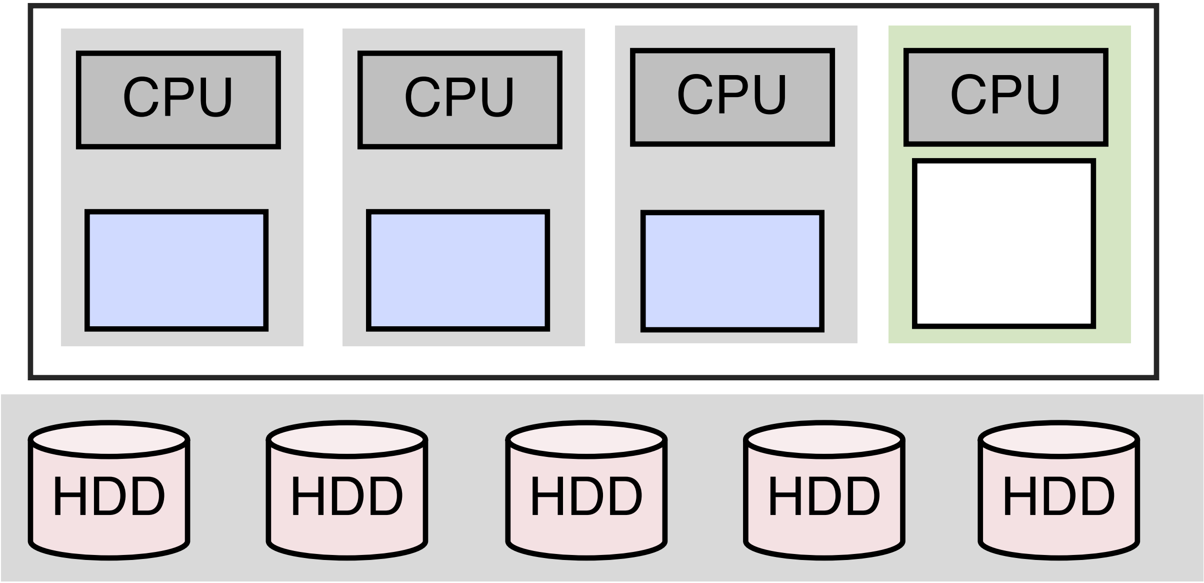


Local caching

– S3 data can be cached in local memory or disk

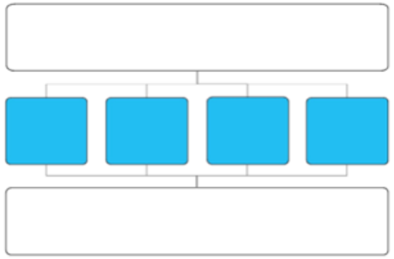
Consistent hashing

- 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

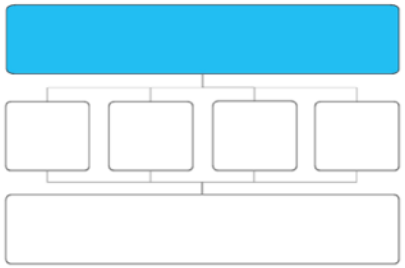
Architecture – Virtual Warehouse



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

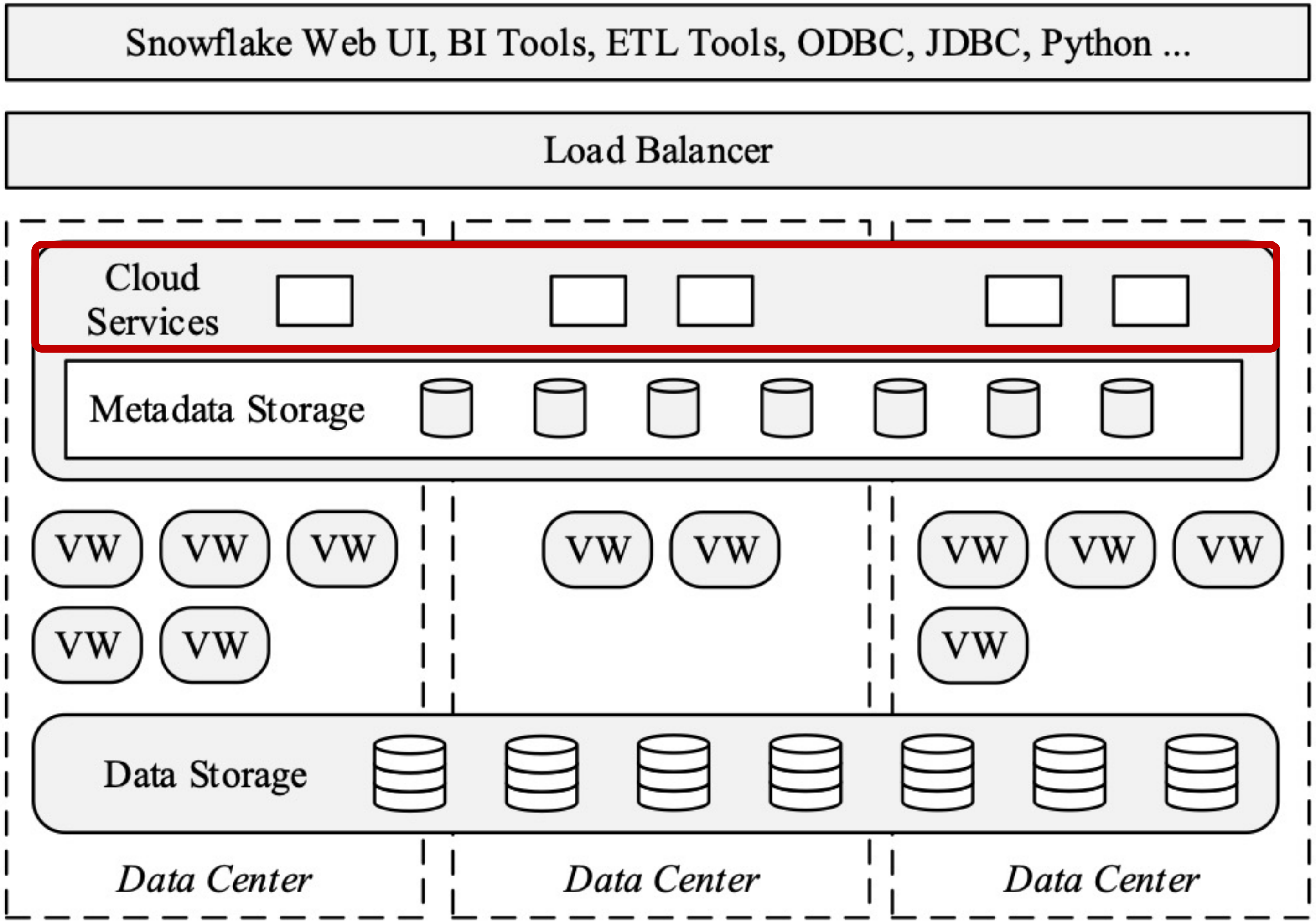
Concurrency control

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

Pruning

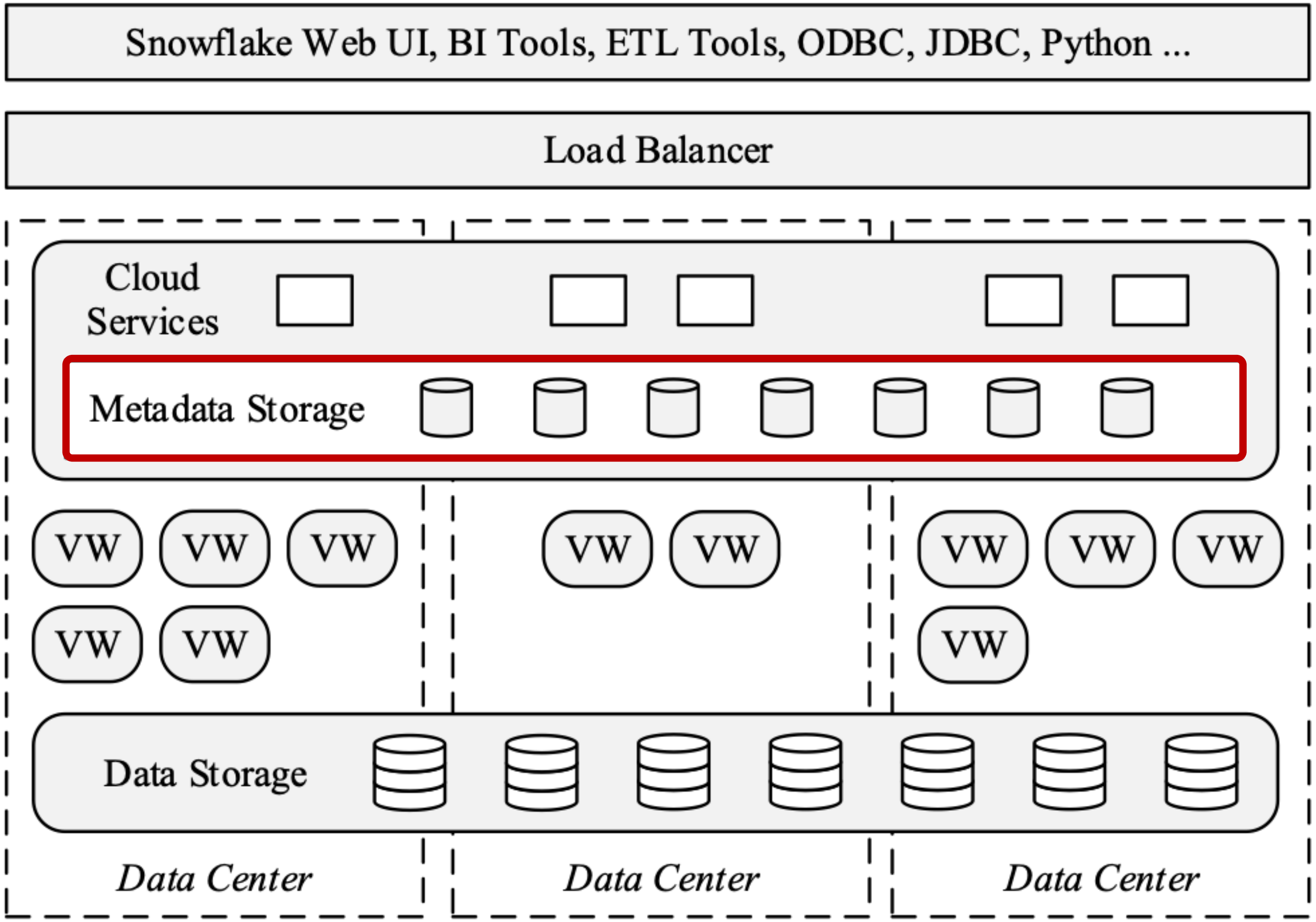
- Snowflake has no index (same as some other data warehousing systems)
- Min-max based pruning: store min and max values for a data block

High Availability and Fault Tolerance



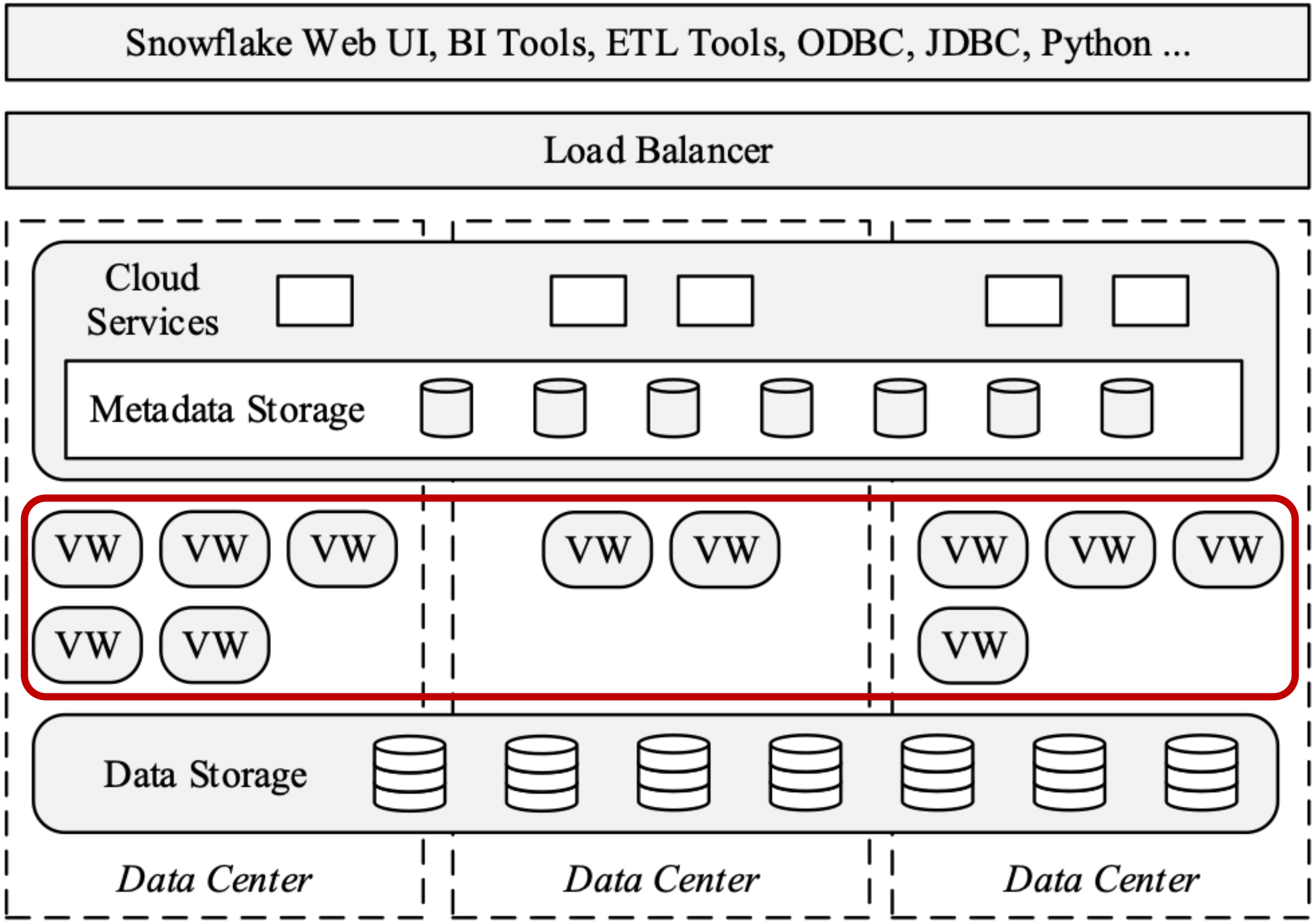
Stateless services

High Availability and Fault Tolerance



Replicated metadata (FoundationDB)

High Availability and Fault Tolerance



One node failure in VW

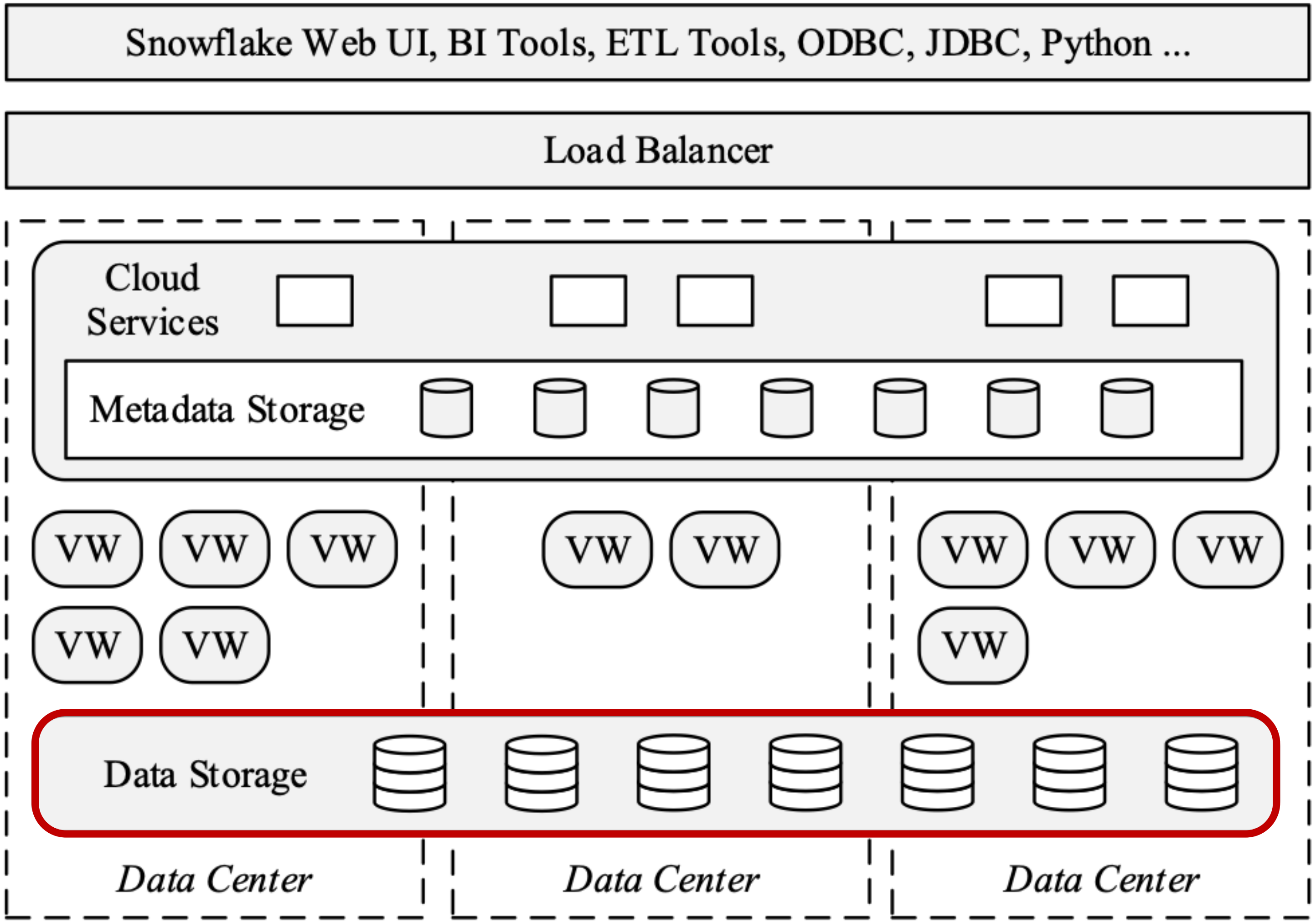
- Re-execute with failed node immediately replaced
- Re-execute with reduced number of nodes

Whole AZ failure

- Re-execute by re-provisioning a new VW

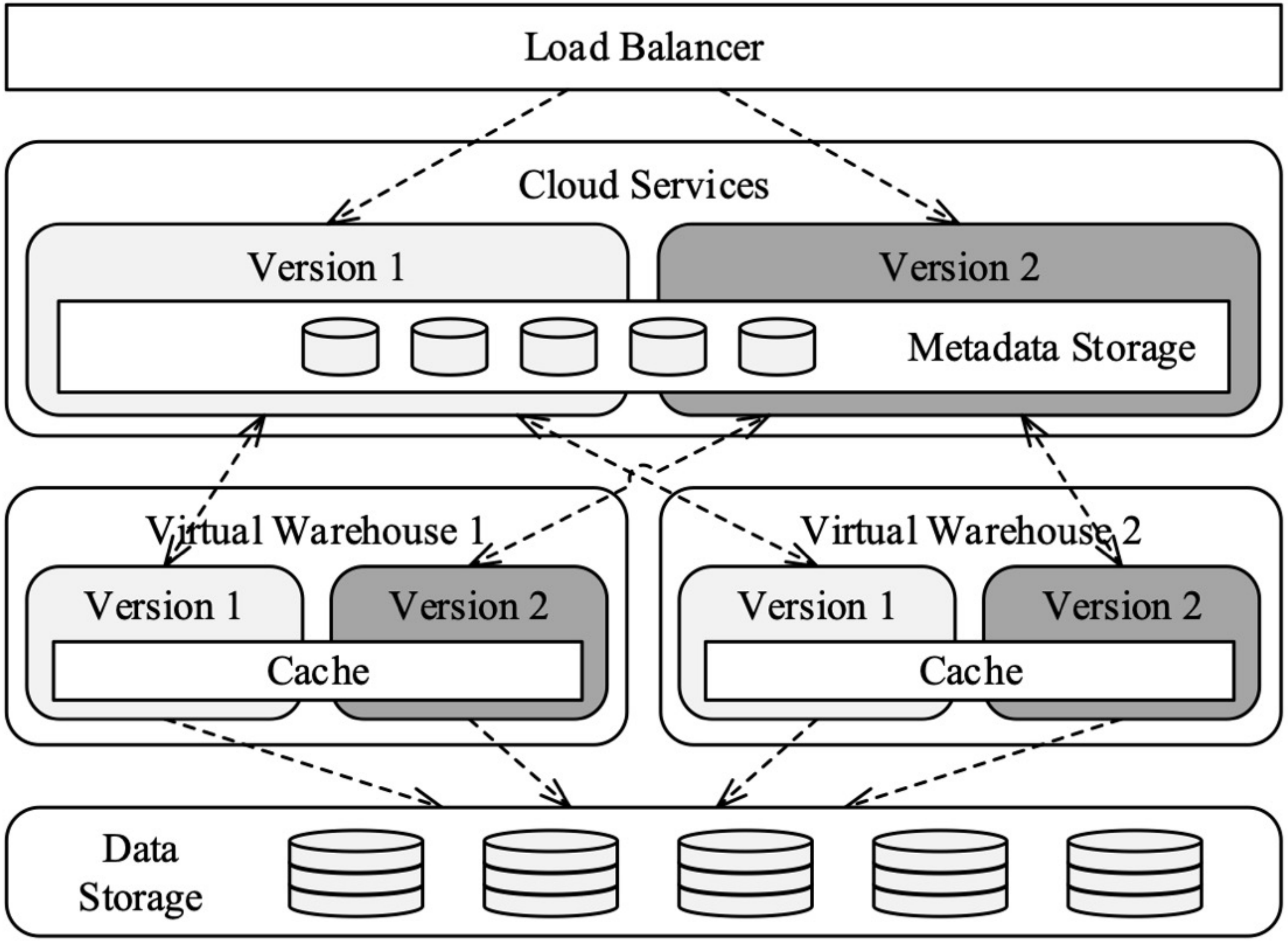
Hot-standby nodes

High Availability and Fault Tolerance



S3 is highly available and durable

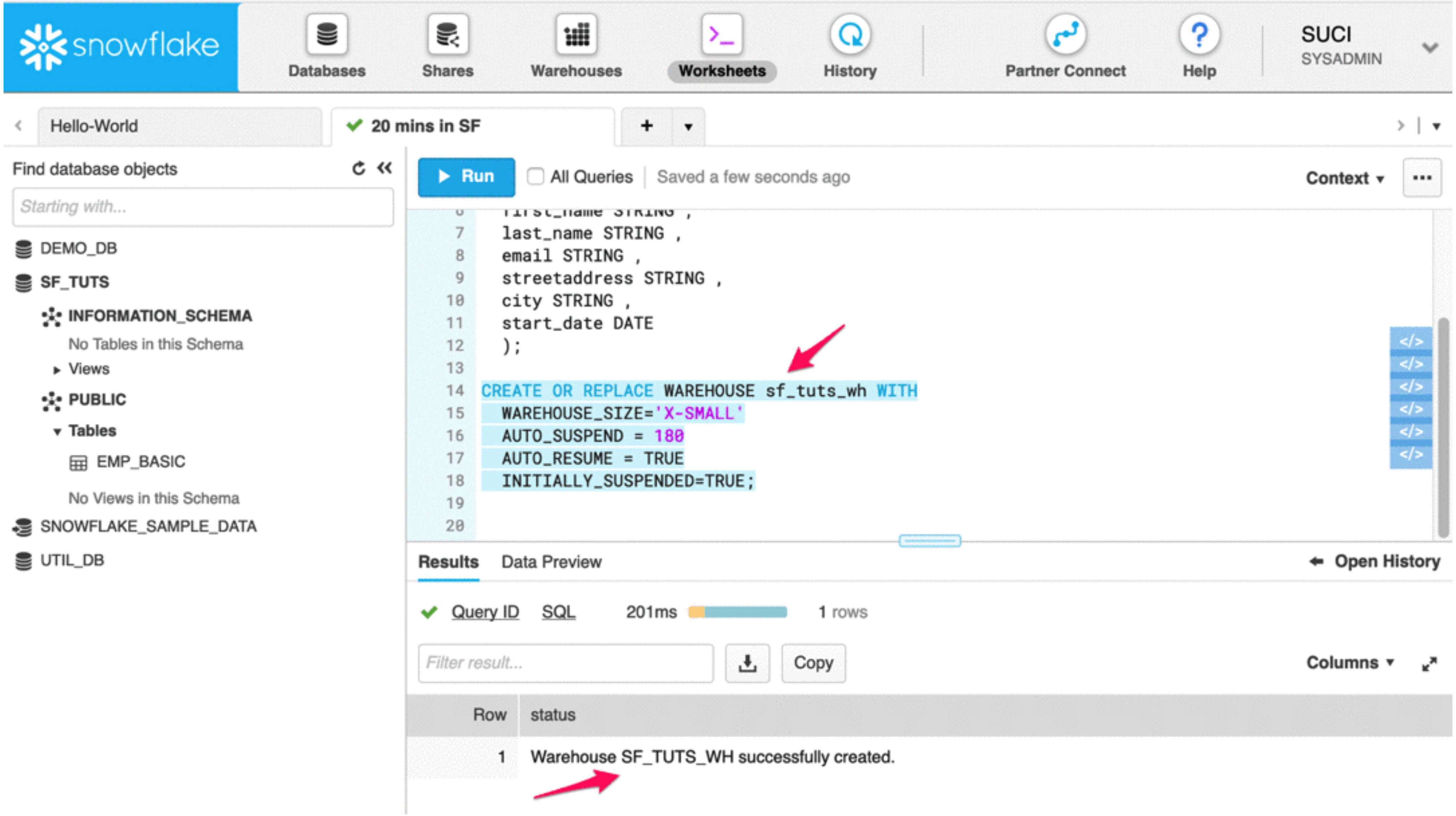
Online Upgrade



Deploy new versions of services and VWs

Previous version terminates after active queries finish

Web User Interface (Serverless)

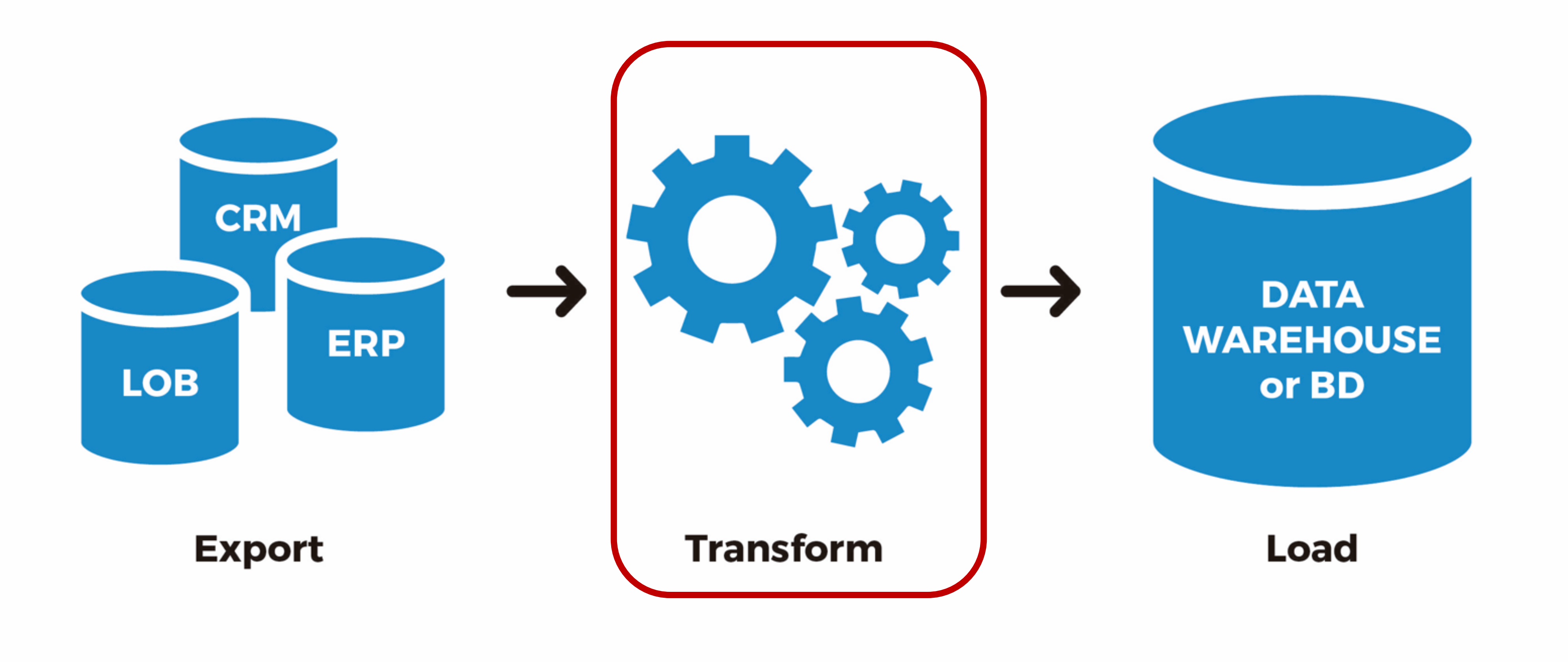


```
  first_name STRING ,
  last_name STRING ,
  email STRING ,
  streetaddress STRING ,
  city STRING ,
  start_date DATE
  );

CREATE OR REPLACE WAREHOUSE sf_tuts_wh WITH
  WAREHOUSE_SIZE='X-SMALL'
  AUTO_SUSPEND = 180
  AUTO_RESUME = TRUE
  INITIALLY_SUSPENDED=TRUE;
```

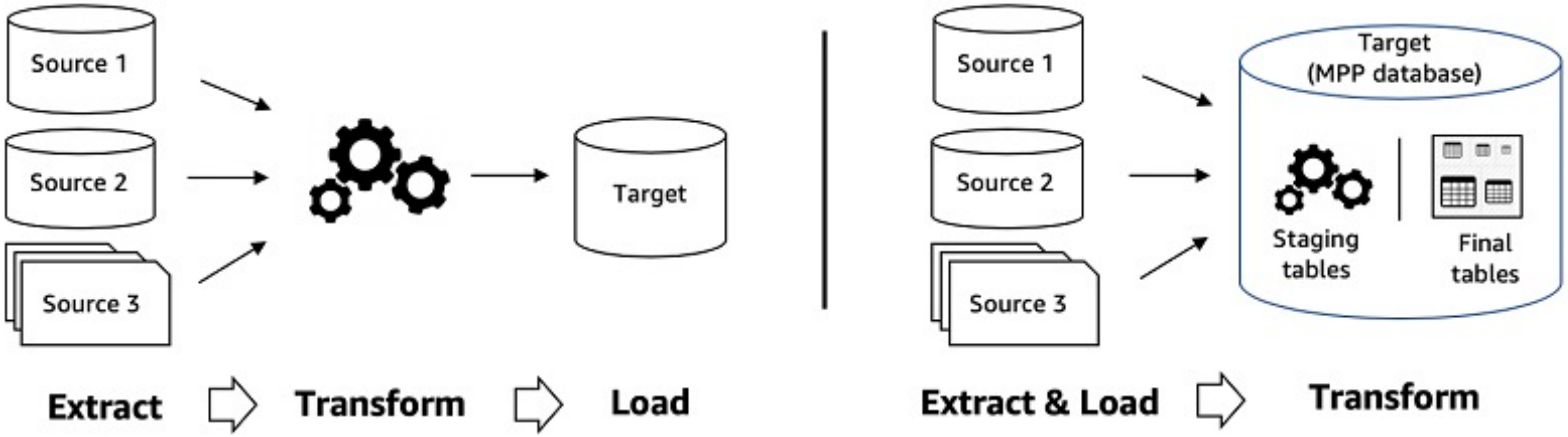
Row	status
1	Warehouse SF_TUTS_WH successfully created.

Extract-Transform-Load (ETL)



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

ETL vs. ELT



E -> T -> L

E -> L -> T

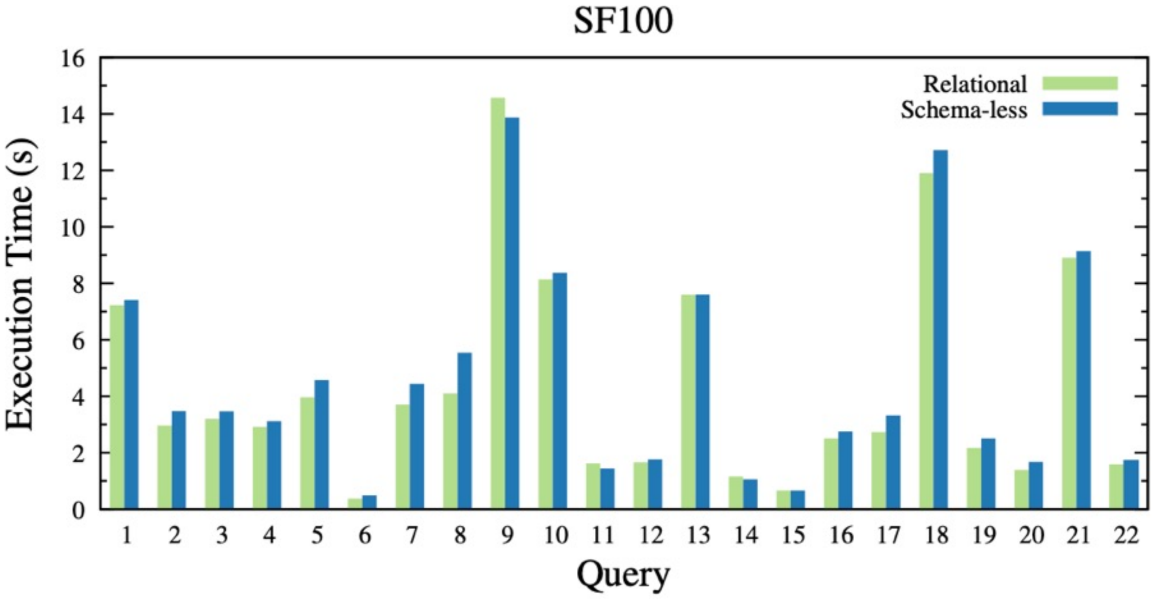
Picture from https://aws.amazon.com/blogs/big-data/etl-and-elt-design-patterns-for-lake-house-architecture-using-amazon-redshift-part-1/

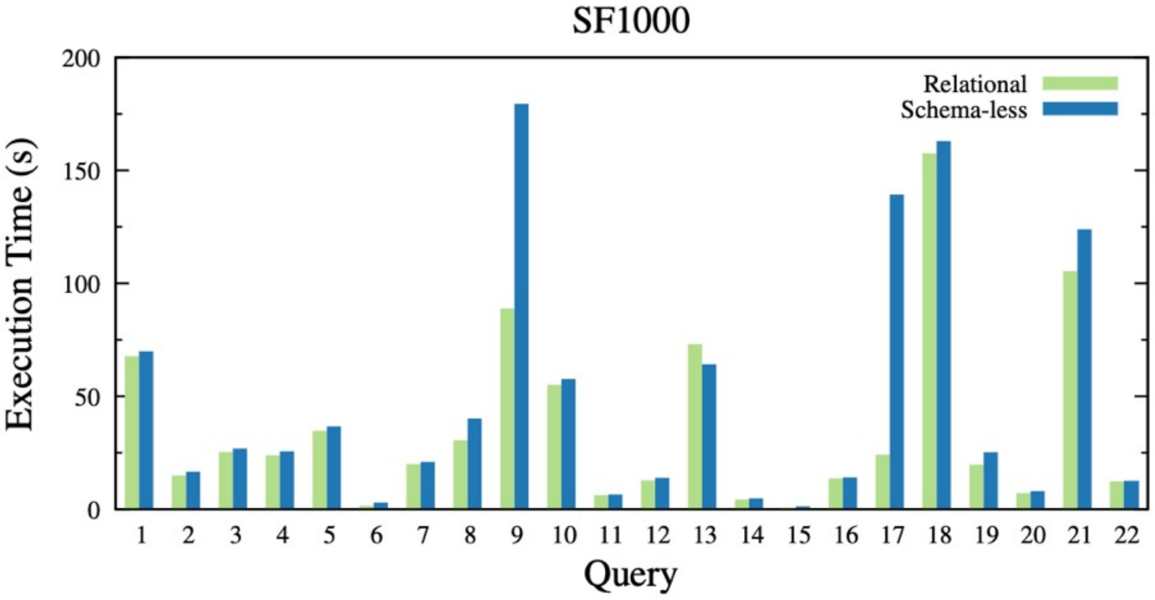
Optimization for Semi-Structured Data

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)





Snowflake – Q/A

No indexing used? min-max filtering

Node failure causes the entire query to fail? More graceful failures like in Spark?

Push-based execution?

Transaction support in Snowflake?

How to deal with large intermediate data?

Performance of Snowflake vs. a shared-nothing deployment?

Discussion Question

For a multi-cloud analytical database where the data is stored across AWS, Azure, and GCP, is Snowflake architecture a good fit? What architecture would you choose in this scenario?

Snowflake is promoting the idea of data marketplace, where Snowflake users can share/trade their data and queries (think of App Store). What new applications can this enable in your opinion?

Please submit your discussion to hotcrp **as a new submission** by the end of Thursday (9/14)

- Title starts with "[Discussion L3]"
- Set authors properly

Before Next Lecture

Review one of “disaggregation for analytical processing” papers