Cloud Data Lakes
..Or “LakeHouse”
..Or “Open Data Lake”

Andrew Lamb | Staff Engineer, InfluxData

April 17, 2025, Boston University Guest Lecture, CS-561

influxdata®

Andrew Lamb

Staff Engineer
InfluxData

> 22 😱 years in enterprise software development

Oracle: Database (2 years)

DataPower: XSLT compiler (2 years)

Vertica: DB / Query Optimizer (6 years)

Nutonian/DataRobot: ML Startups (7 years)

InfluxData: InfluxDB 3.0, Arrow, DataFusion (5 years)

MIT VI-2 2002, MEng 2003

Outline

- Why this topic is important
- Database Architectures through the Ages
- Trends driving the move “to the cloud”
- Disaggregated Databases, and common architecture features

influxdata®

Context: 🍿

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].

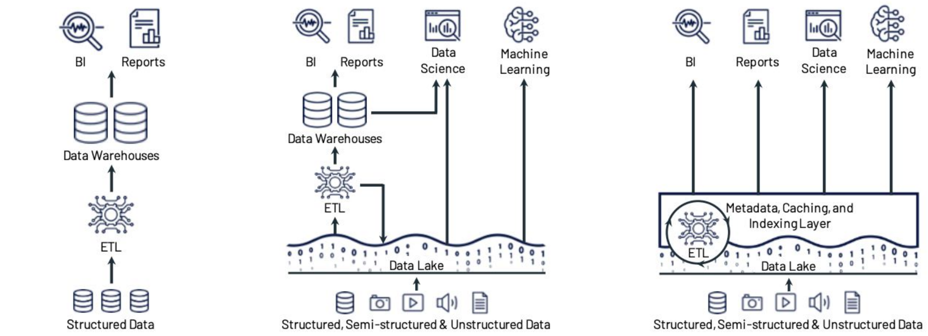
From 2015 onwards, cloud data lakes, such as S3, ADLS and GCS, started replacing HDFS. They have superior durability (often >10 nines), geo-replication, and most importantly, extremely low cost with the possibility of automatic, even cheaper, archival storage, e.g., AWS Glacier. The rest of the architecture 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



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

Figure 1: Evolution of data platform architectures to today's two-tier model (a-b) and the new Lakehouse model (c).

This article is published under the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/). *11th Annual Conference on Innovative Data Systems Research (CIDR '21)*, January 11–15, 2021, Online.

Context: 🍿

databricks.com/blog/2021/11/02/databricks-sets-official-data-warehousing-performance-record.html

Google Maps Gmail Apache Arrow Dat... Tianyu Li

databricks Why Databricks Product Solutions Resources About DATA + AI SUMMIT

Blog / Data Warehousing / Article

Databricks Sets Official Data Warehousing Performance Record

Databricks performan and outperfo

Price

Snowflake

Databricks

Published: November 2, 2021 | Data Warehousing | 10 min read | By Reynold Xin and Mostafa Mokhtar

Today, we are proud to announce that Databricks SQL has set a new world record in 100TB TPC-DS, the gold standard performance benchmark for data warehousing. **Databricks SQL outperformed the previous record by 2.2x**. Unlike most other benchmark news, this result has been formally audited and reviewed by the TPC council.

These results were corroborated by research from Barcelona Supercomputing Center, which frequently runs benchmarks that are derivative of TPC-DS on popular data warehouses. **Their latest research benchmarked Databricks and Snowflake, and found that Databricks was 2.7x faster and 12x better in terms of price performance**. This result validated the thesis that data warehouses such as Snowflake become prohibitively expensive as data size increases in production.

Databricks has been rapidly developing full blown data warehousing capabilities directly on data lakes, bringing the best of both worlds in one data architecture dubbed the data lakehouse. We announced our full suite of data warehousing capabilities as Databricks SQL in November 2020. The open question since then has been whether an open architecture based on a lakehouse can provide the

snowflake.com/en/blog/industry-benchmarks-and-

Google Maps Gmail Apache Arrow Dat... Tiany

snowflake Product Solutions Why S

BLOG CATEGORY

STRATEGY & INSIGHTS NOV 12, 2021

Industry Benchmark Integrity

When we founded Snowflake, we set out to build an inno account what had worked well and what hadn't in prior ar leverage the cloud to rethink the limits of what was possil system that "just worked." We knew there were many opp innovate to lead on performance and scale, simplicity of a

In the same way that we had clarity about many things we didn't want to do. One such thing was engaging in benchr claims divorced from real-world experiences. This practice customers first.

Twenty years ago, the game of leapfrogging benchmark re industry and both of us were on the front line fighting the new world records were being set on a regular basis. Mos special settings, and very specific optimizations that woul Unfortunately, many such changes translated into additio had little or even negative impact on customers' day-to-d Development teams are distracted from focusing on what underserved with more complex technology. Anyone who the reality that the benchmark race became a distraction reason why all the relevant players in the database indust workloads, have largely stopped publishing new results.

Since founding Snowflake, we have focused on our customers and their workloads, and not on synthetic

Proprietary vs open data formats

MPP architecture

Throughput vs latency trade off

databricks Why Databricks Product Solutions Resources About DATA + AI SUMMIT

Blog / Data Warehousing / Article

Snowflake Claims Similar Price/Performance to Databricks, but Not So Fast!

databricks Blog Snowflake Claims Sir Price/Performance t Databricks, but Not S

Published: November 15, 2021 | Data Warehousing | 6 min read | By Mostafa Mokhtar, Reynold Xin and Matei Zaharia

On Nov 2, 2021, we announced that we set the official world record for the fastest data warehouse with our Databricks SQL lakehouse platform. These results were audited and reported by the official Transaction Processing Performance Council (TPC) in a 37-page document available online at tpc.org. We also shared a third-party benchmark by the Barcelona Supercomputing Center (BSC) outlining that Databricks SQL is significantly faster and more cost effective than Snowflake.

A lot has happened since then: many congratulations, some questions, and some sour grapes. We take this opportunity to reiterate that **we stand by our blog post and the results: Databricks SQL provides superior performance and price performance over Snowflake, even on data warehousing workloads (TPC-DS)**.

Snowflake's response: "lacking integrity"?

Snowflake responded 10 days after our publication (last Friday) claiming that our results were "lacking integrity." They then presented their own benchmarks

Keep up wit

Recommen

databric Blog Eliminating for Databas

influxdata®

Table Format wars ⚔️ (preview)

Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores

Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, Michał Świtakowski, Michał Szafrański, Xiao Li, Takuya Ueshin, Mostafa Mokhtar, Peter Boncz[1], Ali Ghodsi[2], Sameer Paranjpye, Pieter Senster, Reynold Xin, Matei Zaharia[3]
Databricks, [1]CWI, [2]UC Berkeley, [3]Stanford University
delta-paper-authors@databricks.com

ABSTRACT

Cloud object stores such as Amazon S3 are some of the largest and most cost-effective storage systems on the planet, making them an attractive target to store large data warehouses and data lakes. Unfortunately, their implementation as key-value stores makes it difficult to achieve ACID transactions and high performance: metadata operations such as listing objects are expensive, and consistency guarantees are limited. In this paper, we present Delta Lake, an open source ACID table storage layer over cloud object stores initially developed at Databricks. Delta Lake uses a transaction log that is compacted into Apache Parquet format to provide ACID properties, time travel, and significantly faster metadata operations for large tabular datasets (e.g., the ability to quickly search billions of table partitions for those relevant to a query). It also leverages this design to provide high-level features such as automatic data layout optimization, upserts, caching, and audit logs. Delta Lake tables can be accessed from Apache Spark, Hive, Presto, Redshift and other systems. Delta Lake is deployed at thousands of Databricks customers that process exabytes of data per day, with the largest instances managing exabyte-scale datasets and billions of objects.

PVLDB Reference Format:
Armbrust et al. Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. *PVLDB*, 13(12): 3411-3424, 2020.
DOI: https://doi.org/10.14778/3415478.3415560

1. INTRODUCTION

Cloud object stores such as Amazon S3 [4] and Azure Blob Storage [17] have become some of the largest and most widely used storage systems on the planet, holding exabytes of data for millions of customers [46]. Apart from the traditional advantages of clouds services, such as pay-as-you-go billing, economies of scale, and expert management [15], cloud object stores are especially attractive because they allow users to scale computing and storage resources separately: for example, a user can store a petabyte of data but only run a cluster to execute a query over it for a few hours.

As a result, many organizations now use cloud object stores to manage large structured datasets in data warehouses and data lakes.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415560

The major open source "big data" systems, including Apache Spark, Hive and Presto [45, 52, 42], support reading and writing to cloud object stores using file formats such as Apache Parquet and ORC [13, 12]. Commercial services including AWS Athena, Google BigQuery and Redshift Spectrum [1, 29, 39] can also query directly against these systems and these open file formats.

Unfortunately, although many systems support reading and writing to cloud object stores, achieving *performant* and *mutable* table storage over these systems is challenging, making it difficult to implement data warehousing capabilities over them. Unlike distributed filesystems such as HDFS [5], or custom storage engines in a DBMS, most cloud object stores are merely key-value stores, with no cross-key consistency guarantees. Their performance characteristics also differ greatly from distributed filesystems and require special care.

The most common way to store relational datasets in cloud object stores is using columnar file formats such as Parquet and ORC, where each table is stored as a set of objects (Parquet or ORC "files"), possibly clustered into "partitions" by some fields (e.g., a separate set of objects for each date) [45]. This approach can offer acceptable performance for scan workloads as long as the object files are moderately large. However, it creates both *correctness* and *performance* challenges for more complex workloads. First, because multi-object updates are not atomic, there is no isolation between queries: for example, if a query needs to update multiple objects in the table (e.g., remove the records about one user across all the table's Parquet files), readers will see partial updates as the query updates each object individually. Rolling back writes is also difficult: if an update query crashes, the table is in a corrupted state. Second, for large tables with millions of objects, metadata operations are expensive. For example, Parquet files include footers with min/max statistics that can be used to skip reading them in selective queries. Reading such a footer on HDFS might take a few milliseconds, but the latency of cloud object stores is so much higher that these data skipping checks can take longer than the actual query.

In our experience working with cloud customers, these consistency and performance issues create major challenges for enterprise data teams. Most enterprise datasets are continuously updated, so they require a solution for atomic writes; most datasets about users require table-wide updates to implement privacy policies such as GDPR compliance [27]; and even purely internal datasets may require updates to repair incorrect data, incorporate late records, etc. Anecdotally, in the first few years of Databricks' cloud service (2014–2016), around half the support escalations we received were due to data corruption, consistency or performance issues due to cloud storage strategies (e.g., undoing the effect of a crashed update job, or improving the performance of a query that reads tens of thousands of objects).

3411

DELTA LAKE

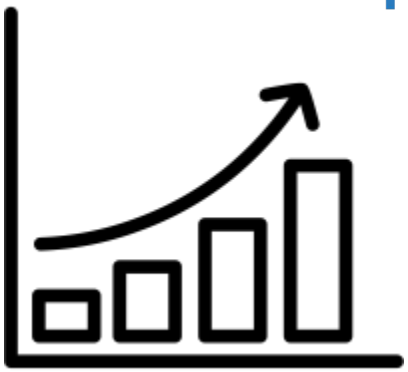
databricks

Delta Lake from databricks

* First mover, better support

* Arguably technically superior

VS



Apache ICEBERG

* much faster / wider adoption

* More Neutral governance

* among other things championed by snowflake

["Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores"]

influxdata

Databricks reportedly paid $2 billion in Tabular acquisition

What if you didn't have to choose a format?

Databricks Agrees to Acquire Tabular, the Company Founded by the Original Creators of Apache Iceberg

June 4, 2024

Share this post

Databricks and Tabular will work together towards a joint vision of the open lakehouse

SAN FRANCISCO — June 4, 2024 — Databricks, the Data and AI company, today announced it has agreed to acquire Tabular, a data management company founded by Ryan Blue, Daniel Weeks, and Jason Reid. By bringing together the original creators of Apache Iceberg™ and Linux Foundation Delta Lake, the two leading open source lakehouse formats, Databricks will lead the way with data compatibility so that organizations are no longer limited by which of these formats their data is in. Databricks intends to work closely with the Delta Lake and Iceberg communities to bring format compatibility to the lakehouse; in the short term, inside Delta Lake UniForm and in the long term, by evolving toward a single, open, and common standard of interoperability. Databricks and Tabular will work together towards a joint vision of the open lakehouse.

source

Posted: 9:44 AM PDT · August 14, 2024

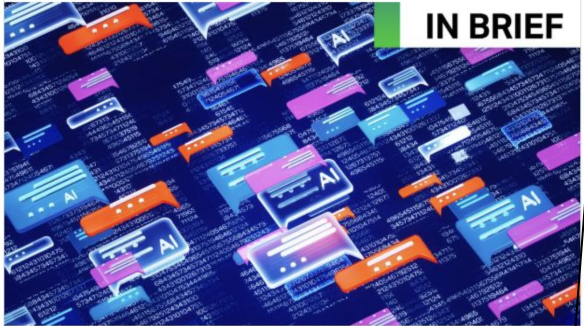


IMAGE CREDITS: ANDRIY ONUFRIYENKO / GETTY IMAGES

Maxwell Zeff

Databricks reportedly paid $2 billion in Tabular acquisition

Analytics and AI giant Databricks reportedly paid nearly $2 billion when it acquired Tabular in June, a startup that was only doing $1 million in annual recurring revenue, according to Bloomberg. That's a pretty outrageous exit multiple, and it was purportedly fueled by a battle between Databricks and Snowflake.

Tabular had over $30 million in funding, backed by Altimeter Capital, Andreessen Horowitz and Zetta Venture Partners, when it was acquired

source

databricks

ON DEMAND

Beyond Lakehouse Table Formats

The original creators of Delta Lake and Apache Iceberg™ take on interoperability

The path to table format interoperability

Choosing the best unified platform for data, analytics and AI is easy — it's lakehouse. Choosing the right open table format for your lakehouse? Not as easy.

For most organizations, it's a daunting decision that delays lakehouse adoption. The holdup hurts your ability to capitalize on analytics and AI.

What if you didn't have to choose a format?

Join us for a conversation about interoperability with Michael Armbrust, original creator of Delta Lake, and Ryan Blue, an original creator of Apache Iceberg. They'll discuss the state of open table formats and how Databricks is solving interoperability.

Access on demand

* First Name

* Last Name

* Company Email

* Company Name

* Job Title

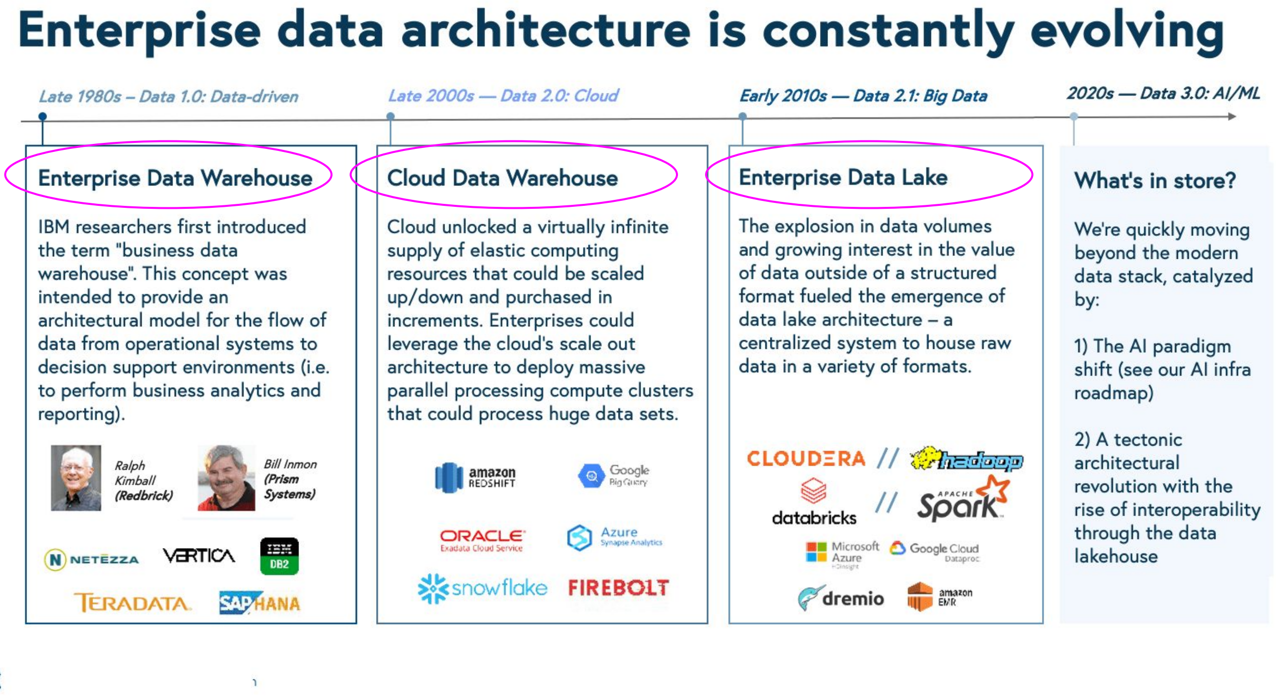
* Phone Number

* Country

United States

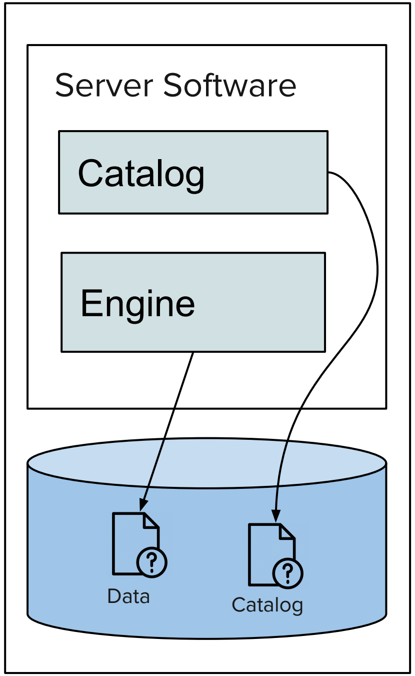
source

Roadmap: Data 3.0 in the Lakehouse Era



[Roadmap: Data 3.0 in the Lakehouse Era - Bessemer Venture Partners](#) (3/25/2025)

Single Node Servers (1990s)



Tightly integrated **engine**, **storage** and **catalog**

CPU + MEM + STORAGE

Data + Catalog *stored in* ***proprietary formats*** *on local file systems*

Microsoft® SQL Server

ORACLE

IBM DB2

PostgreSQL

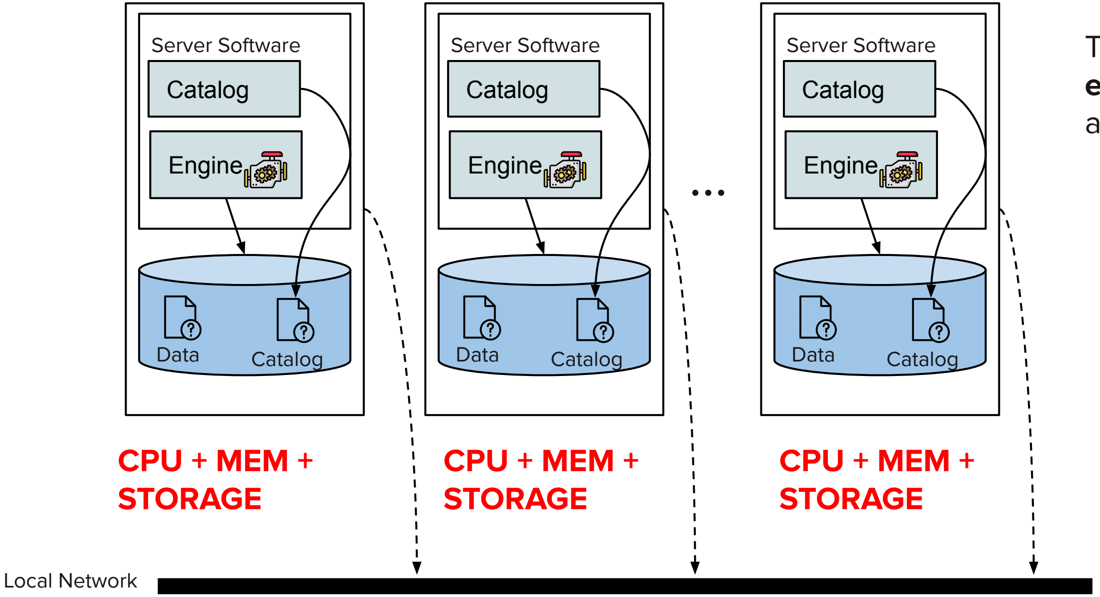
Exemplars

Driven by

- Minicomputer ➔ Servers
- Local hard drive capacity

influxdata®

MPP / Shared Nothing (2000s)



Tightly integrated **engine**, **storage** and **catalog**

Data stored on each node, **next** to compute

Propriety Catalog and Data format

Nodes communicate over commodity network

Exemplars

Driven by

- Fast local networks
- “Inexpensive” (10x cheaper) commodity Linux servers

Arrival of the “Cloud”

Object Stores: What

What: “Infinite FTP server in the sky”

What: Distributed Key/Value stores

Microsoft Azure Blob Storage

AWS S3

MINIO

Google Cloud Storage

Basic CRUD interface:

GET <URL> → Bytes

PUT <URL> Bytes

LIST <PREFIX> (lists keys)

DELETE <URL>

influxdata

Object Stores: Why

Microsoft Azure Blob Storage

AWS S3

MINIO

Google Cloud Storage

Why give up nice File system APIs

E.g. can't append or modify parts of objects

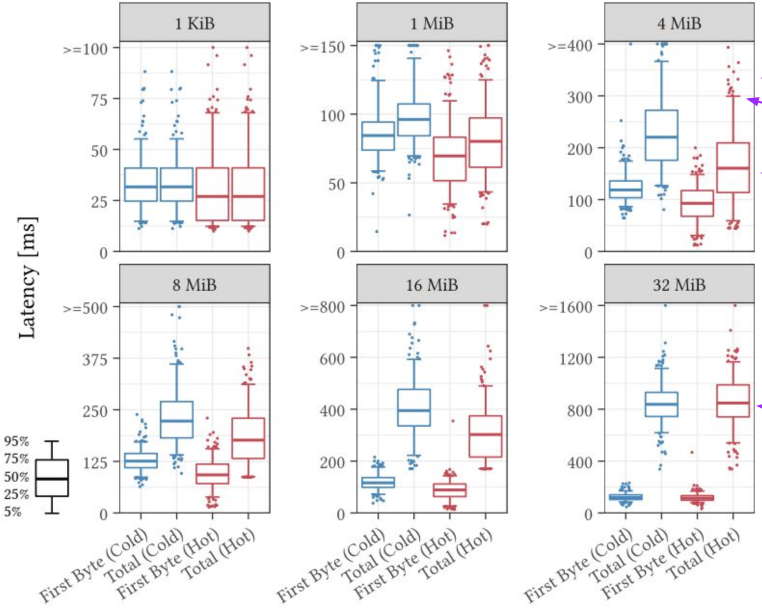
- Durability (3x replication, cross AZ, handled transparently)
- "Infinite" scale + capacity
- cheap ($23/TB/month*)
- Pay per access (not per byte): $0.40/million requests

⇒ Compelling to outsource persistent storage to Object Stores

influxdata

Object Stores: Ugly

Significant latency / latency unpredictability



400+ms for some queries (10MB/s)

300ms avg (P95) 13.3MB/sec

175ms avg (P50) 22.8MB/sec

800ms avg (P50) 40MB/sec

Microsoft Azure Blob Storage

AWS S3

MINIO

Google Cloud Storage

Source: [Exploiting Cloud Object Storage for High-Performance Analytics]

influxdata®

Object Stores: Ugly

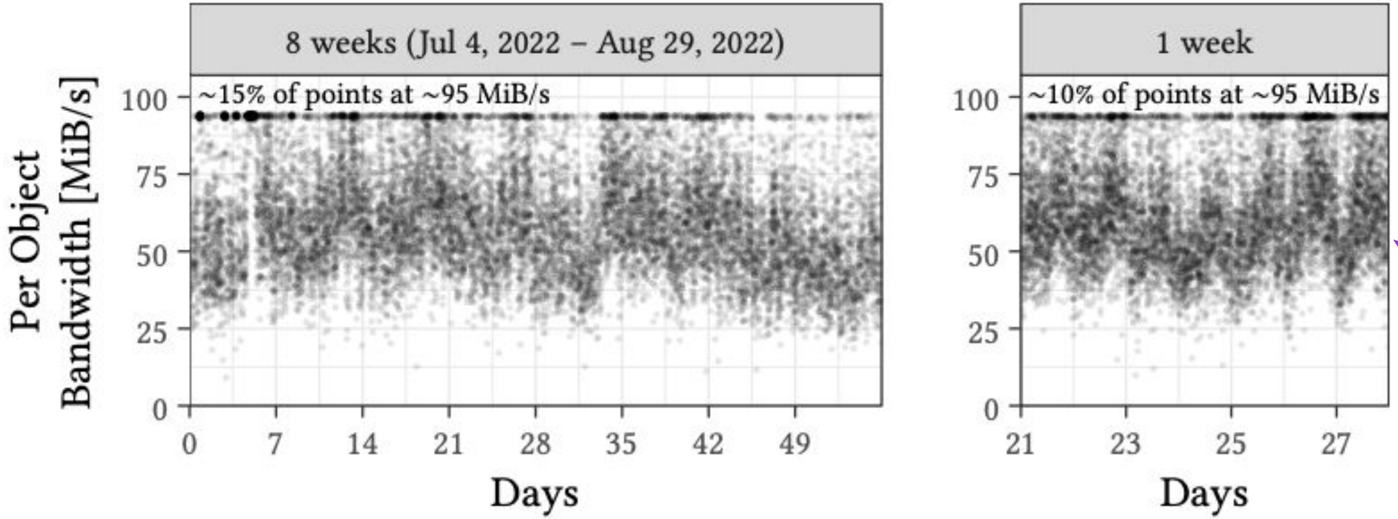
Microsoft Azure Blob Storage

AWS S3

IO

Google Cloud Storage

Significant latency / latency unpredictability



Not isolated from other workloads: observed bandwidth varies cyclically

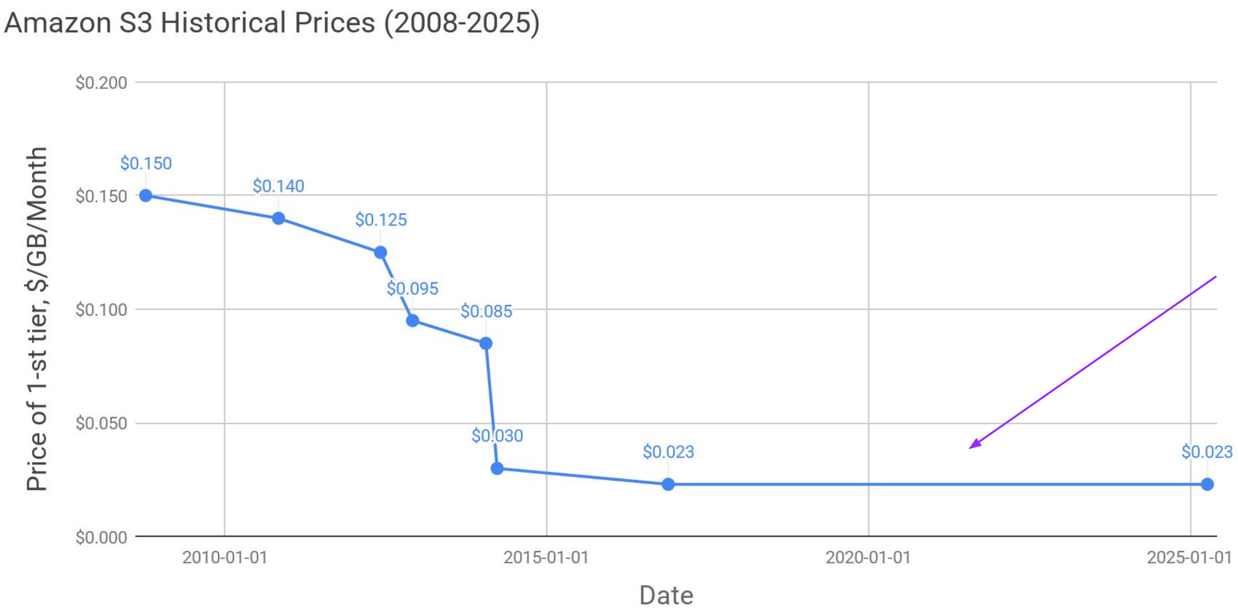
Source: [Exploiting Cloud Object Storage for High-Performance Analytics]

 |

influxdata

Object Stores: Ugly

Isn't as cheap as it turns out, more expensive over time



🤔 no price change in the last 10 years...

Source: [Copy of Amazon S3 Historical Prices (2008-2025)]

Microsoft Azure Blob Storage

AWS S3

MINIO

Google Cloud Storage

influxdata

Elastic Compute: What

GCP Compute Engine

AzureVM

Amazon EC2

What: “Rent VM’s by the day, hour or minute (now)”

Pricing: https://aws.amazon.com/ec2/pricing/on-demand/

Example: t2.xlarge: 4 vCPU 16GB RAM @ $0.1856/hour)

0:00:00 **Start** VM (start billing)

3:25:24 **Stop** VM (stop billing)

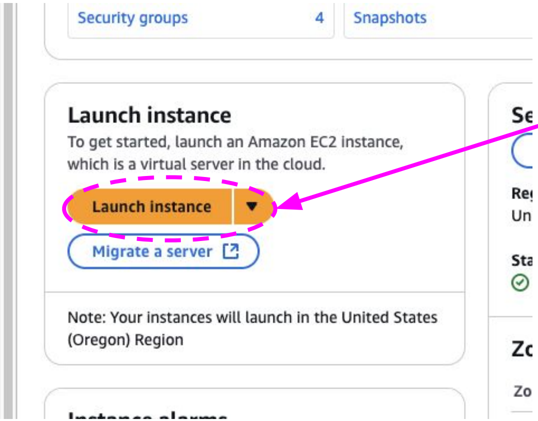
3*60*60 + **25***60 + **24** = 12324 seconds

12324 seconds / 3600 seconds/hour * $0.1856/hour ⇒ $0.64

Elastic Compute: Why

No upfront capital investment

⇒ Much more efficiently use hardware



VS

Personal Anecdote: budgeting ~ $250K 6 months in advance for server clusters to test on

HP DL380s (popular midrange server in late 2000s)

[Ebay "Lot of 21 HP Proliant DL380"]

influxdata

Elastic Compute: The Ugly

GCP Compute Engine

AzureVM

Amazon EC2

Overabundance leads to waste: 💸

Easy to spend

Developers often leave machines running by accident 🤑

Rise of cost optimization software

Personal Anecdote: $1m/month AWS bills

influxdata®

Elastic Compute: The Ugly

GCP Compute Engine

AzureVM

Amazon EC2

Kinda crappy compared to your own machine

Sequential Write Throughput:

- My Macbook Pro 1TB SSD: > **1GB/s**
- GCP VM* 4 SSD @ 1.5TB RAID 0: **815 MB/s**

VS

GCP Compute Engine

* c3-standard-22-lssd (22 vCPUs, 88 GB Memory)

4 “local”SSDs, RAID0, Intel Sapphire Rapids x86_64

```
#  Test the IO throughput using `dd`
dd if=/dev/zero of=/data/test1.img bs=1G count=10 oflag=dsync
# 10737418240 bytes (11 GB, 10 GiB) copied, 13.179 s, 815 MB/s
```

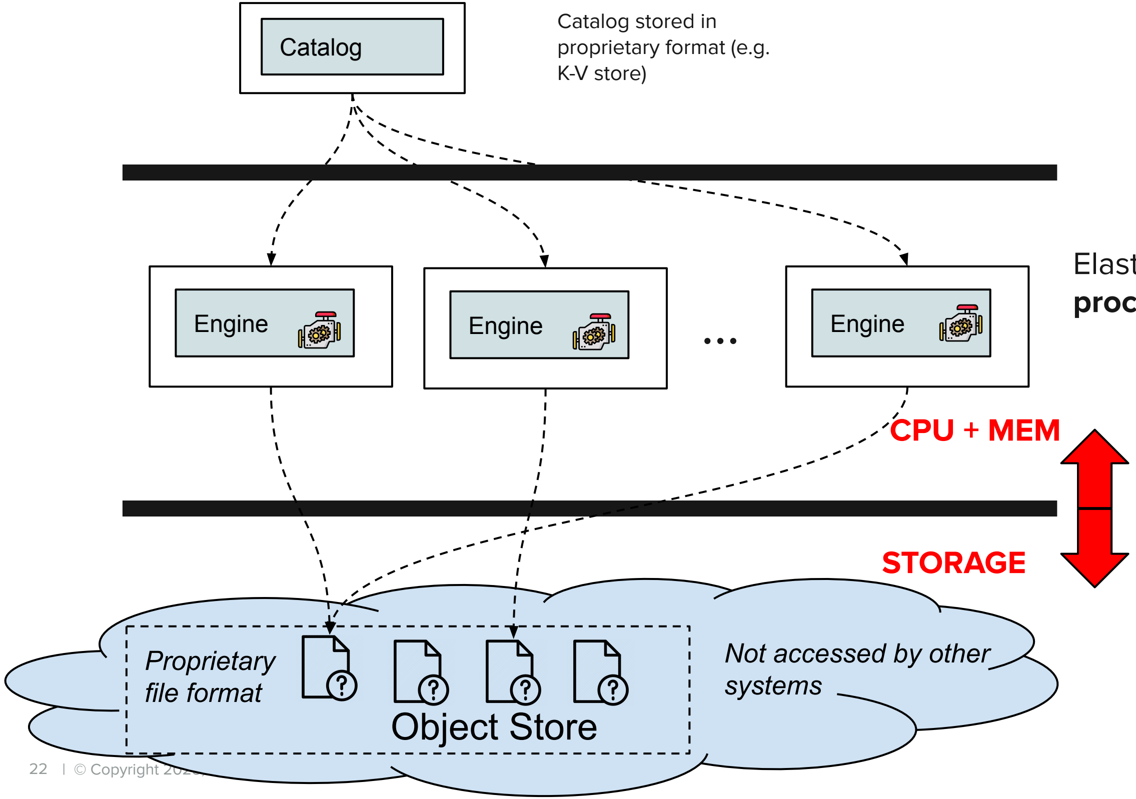
Read more: [tpchgen-rs World’s fastest open source TPC-H data generator, written in Rust - Apache DataFusion Blog]

influxdata

Cloud Database Architecture
“Disaggregated Storage Design”

influxdata®

Disaggregated Architectures (2010s) “Cloud Data Warehouse”



Elastically scalable **processing plane**

Data + Catalog in proprietary formats *on cheap durable object store*

Not *shared across systems*

influxdb 3.0
snowflake
TiDB
Google Big Query
Cloud Spanner

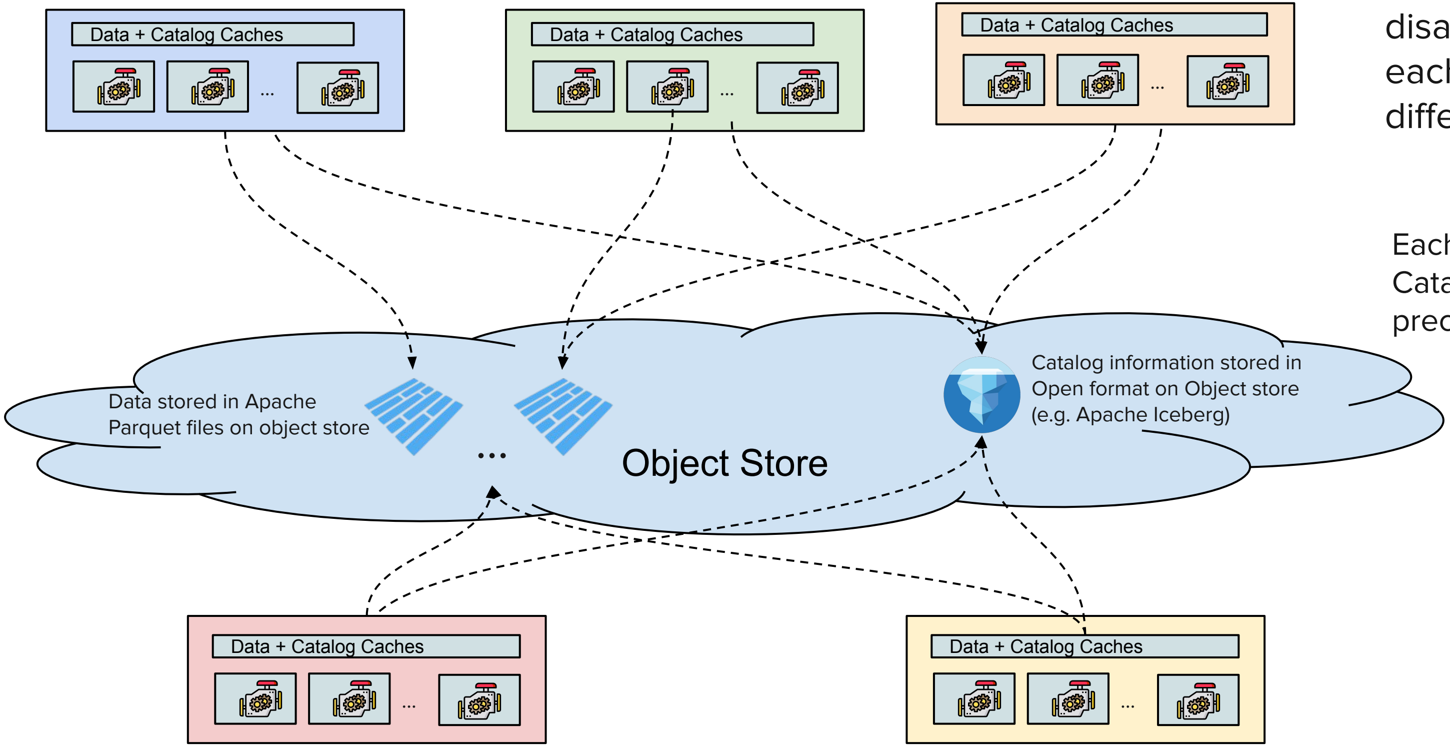
Exemplars

Driven by

- Hourly VM rental
- 10x cheaper storage (AWS S3)

influxdata

Cloud Data Lakes (2020s)



Constellation of disaggregated systems, each focused on different use case

Each includes use case specific Catalog + Data Caching / precomputation / engines

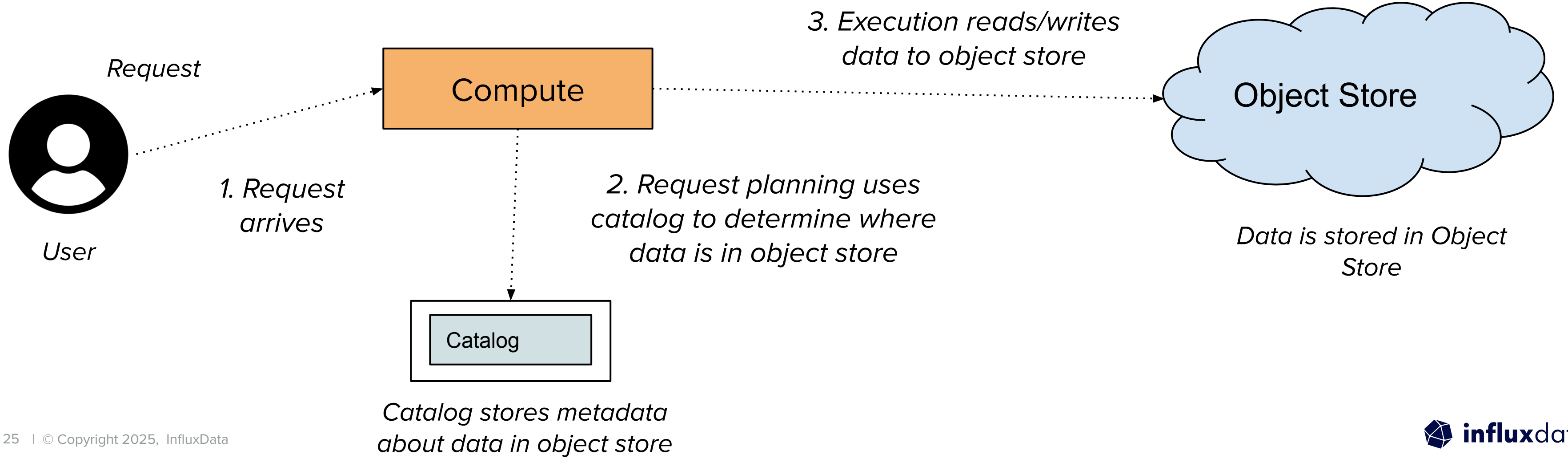
Common Features of Disaggregated Databases

Metadata Store (“Catalogs”)

Object storage latency (100s of ms) is too high for planning for many workloads (both read and write)

No multi-object transactions

⇒ metadata ‘catalog’ describes data layout in object store



influxdata®

Metadata Store ("Catalogs")

Popular choices:

- Key Value store (FoundationDB)
- Traditional transactional SQL systems (postgres)

Typical Contents

- **Schema**: tables, columns, types, etc.
- **Partitioning**: partitions, partition values, etc.
- **File Locations:** paths on object store
- **Pruning:** per-column min/maxes (Small Materialized Aggregates / Zone Maps), Bloom Filters, etc.

Reference: How FoundationDB Powers Snowflake Metadata Forward

Separate Scalable Operations

Separate major responsibilities into separately scalable sets of VMs

Typical Components:

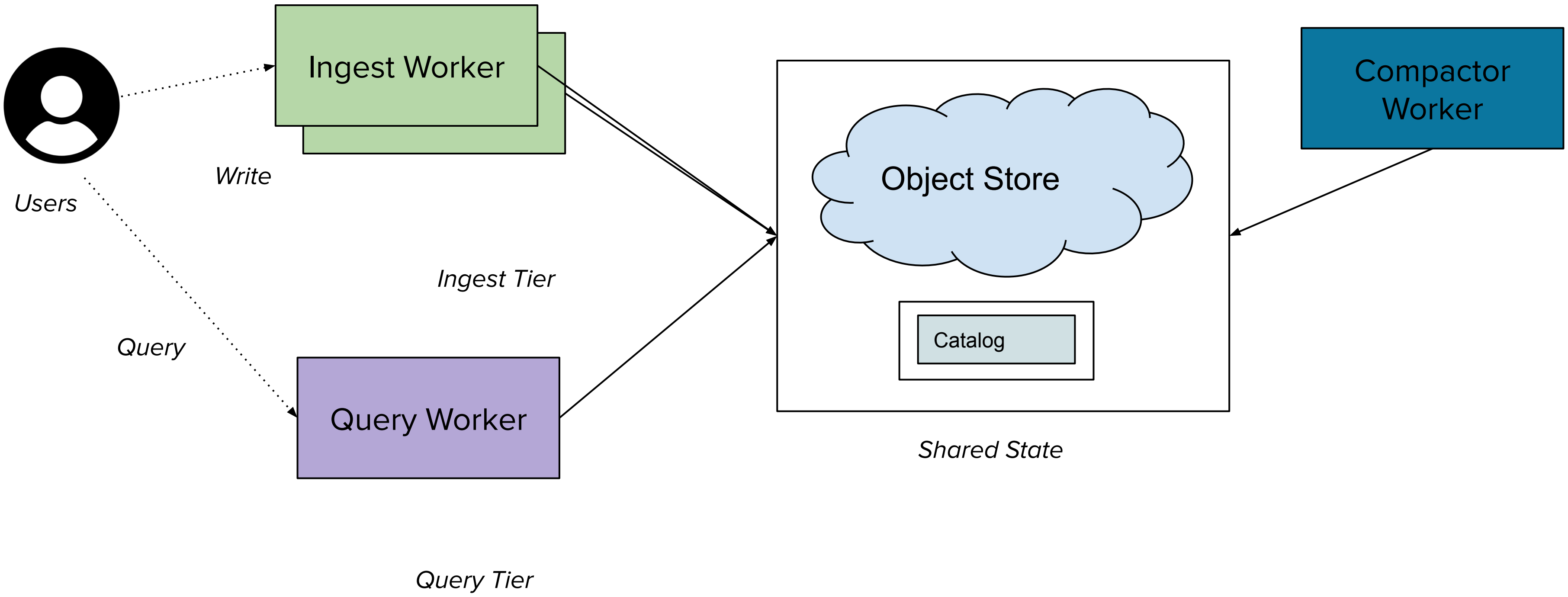
- Write / Ingestion
- Query
- Reorganization (compaction, garbage collection, etc)

Why: scale capacity along with demand (e.g scale writers up to handle bursts)

Industrial Examples:

https://docs.snowflake.com/en/user-guide/intro-key-concepts (virtual warehouses)
https://www.datadoghq.com/blog/engineering/introducing-husky/
https://www.influxdata.com/blog/influxdb-3-0-system-architecture/

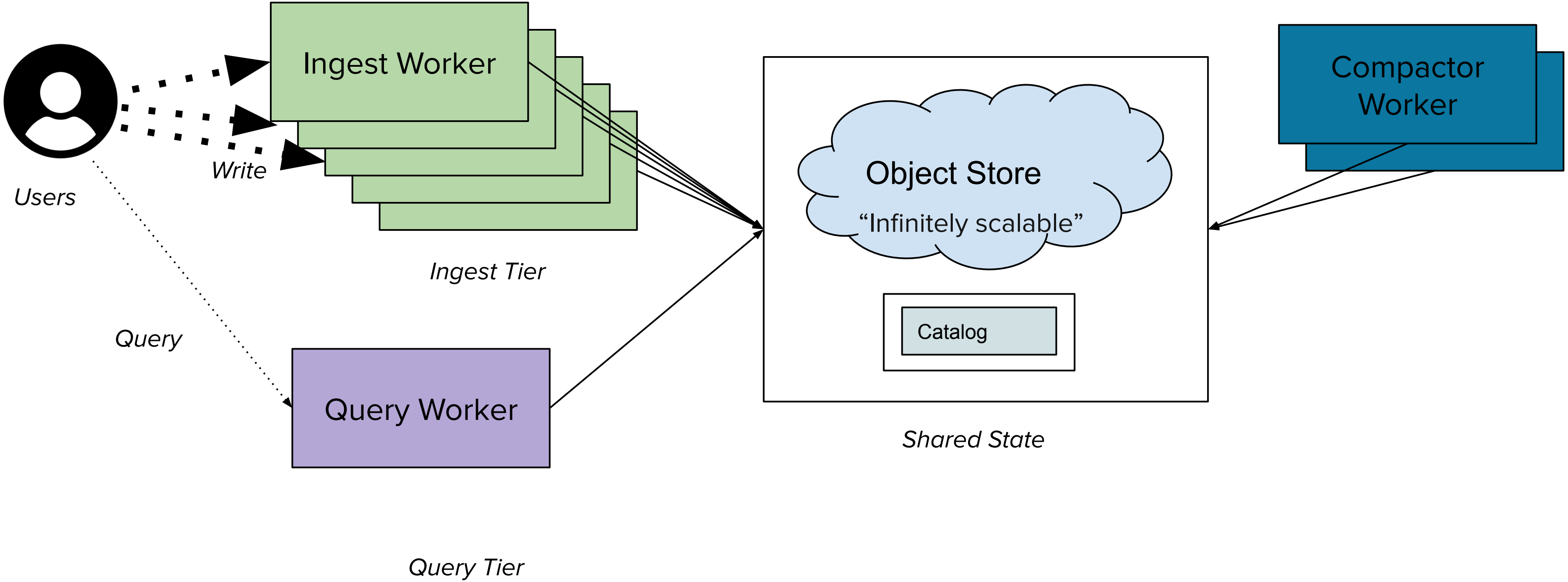
Separate Scalable Operations: Example



 |

influxdata®

Separate Scalable Operations: Example

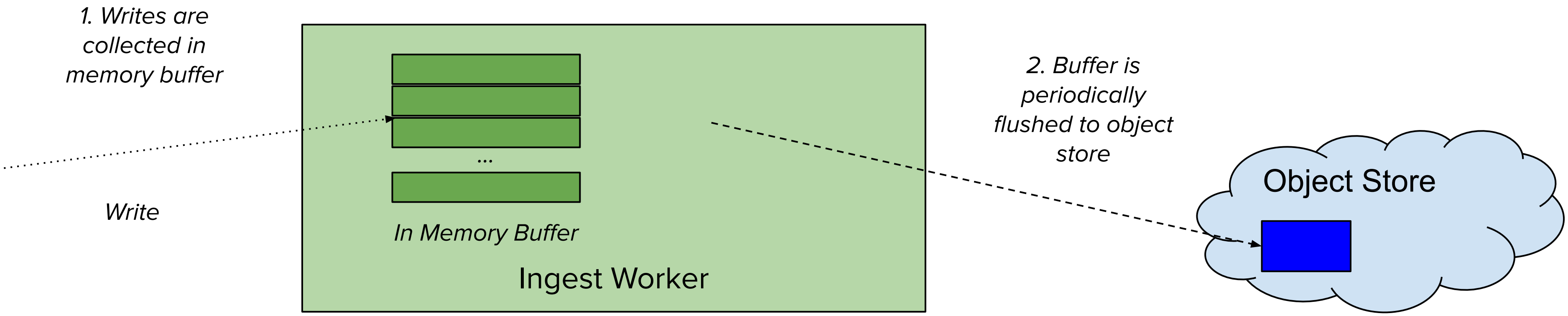


Increase write workload leads to more ingest and compactor workers, no need to increase query tier

Write Buffering

Large per-request overhead to object store ($$ and latency)

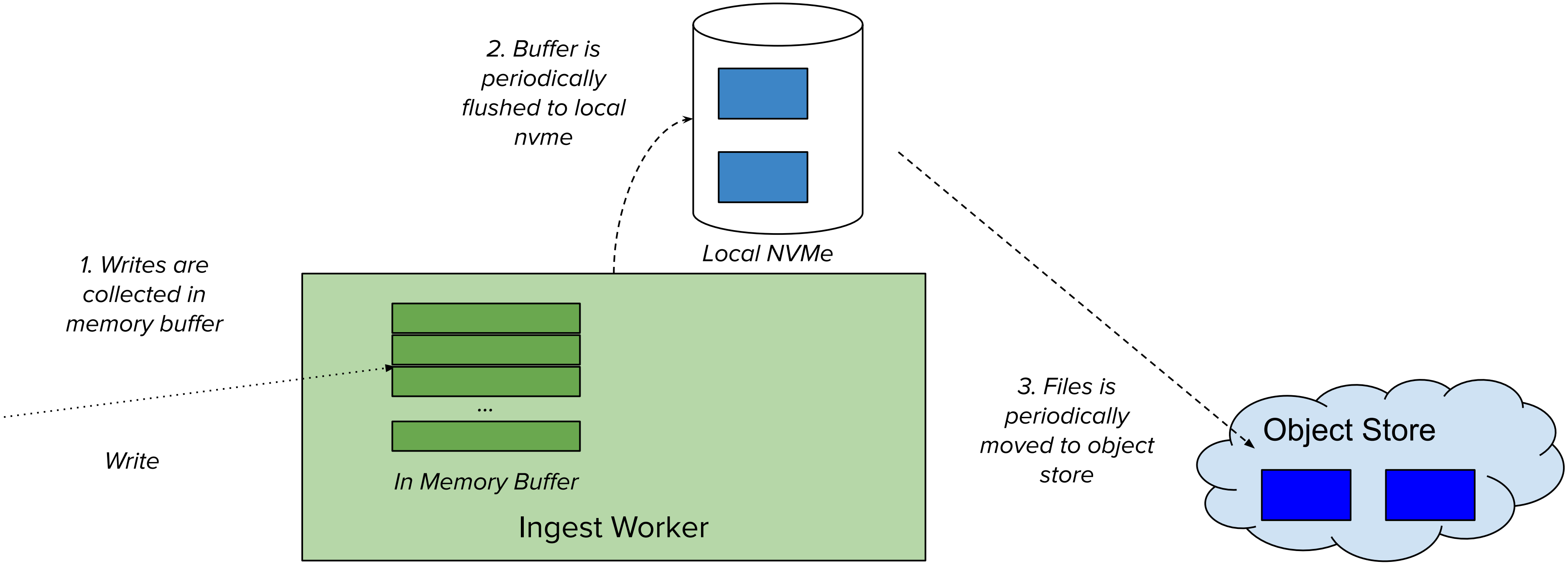
⇒ Buffer in ram + locally to amortize cost across many requests



 |

influxdata

Write Buffering / Local Storage



Write Buffering

Challenges:

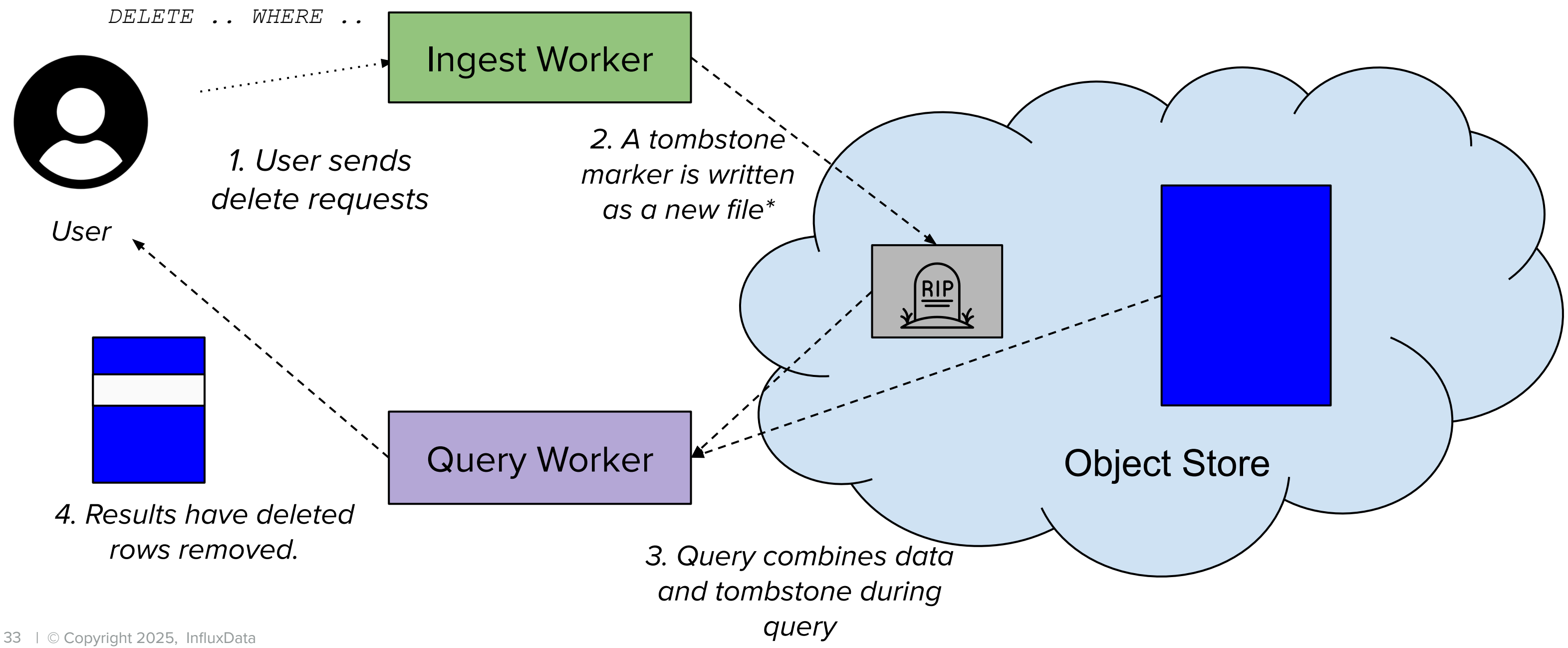
- Durability of data before it is written to object store
- Time to become readable (is memory in buffer readable?)

Examples:

- Monarch: Google's Planet-Scale In-Memory Time Series Database
- Architecture | WarpStream
- Architecture | SlateDB

Deletes (+ Updates) via Tombstones

Write once (no updates) storage ⇒ Delete / Update writes new things



 |

Deletes (+ Updates) via Tombstones

Variations:

- **Delete Vectors** (row ids / offsets deleted)
  - Often stored in objects
  - Time consuming to create / Faster query execution
- **Stored Predicates**
  - Often stored in meta store
  - Fast to create / Potentially slower query execution
  - Tied to predicate expressions
  - Slower as number of deletes increase

Challenges

- Sequencing deletes with inserts
- Performance
- Eventually reclaiming Storage

Offset
21
31
67
104

Delete Vector with resolved row ids

```
user_id IN (123,456)
```

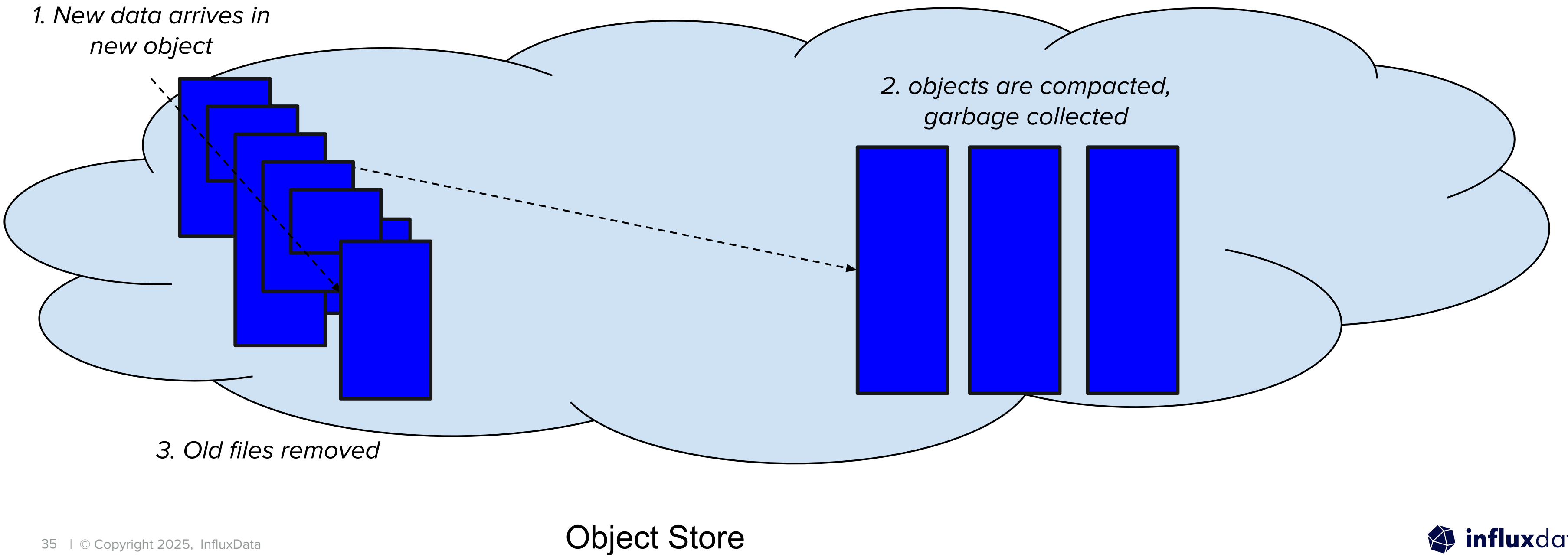
Delete Predicate

 |

Data Layout optimization

Object Storage is write once: write new objects, but not modify existing

⇒ Rewrite objects overtime (better organized, garbage collect, etc)



influxdata®

“Table Formats”

Metadata catalogs / stores are proprietary, add operational overhead.

Use object store to store metadata (cost of increased planning latency)

⇒ Standardize describing what files make up a table

Examples:

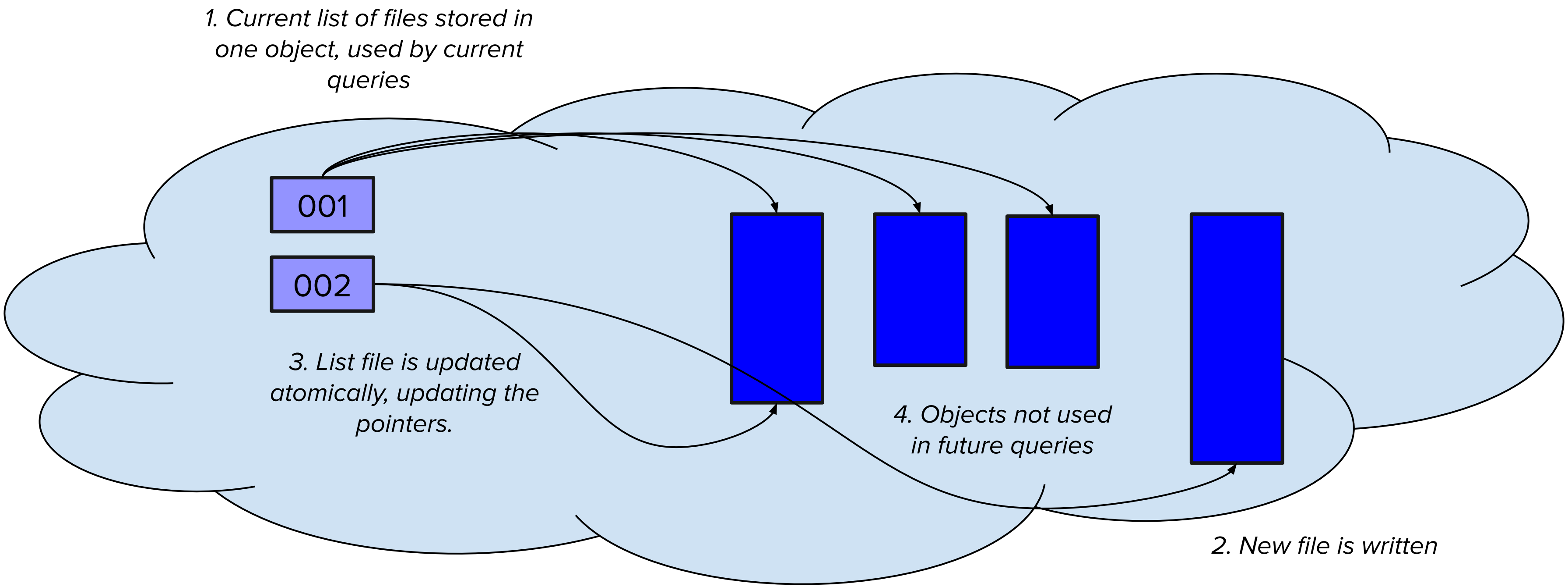
- Apache ICEBERG
- DELTA LAKE
- Apache hudi

“Table Formats”

Separate Data from Metadata

“Table Formats”

Classic case of solve the problem with a layer of indirection



influxdata®

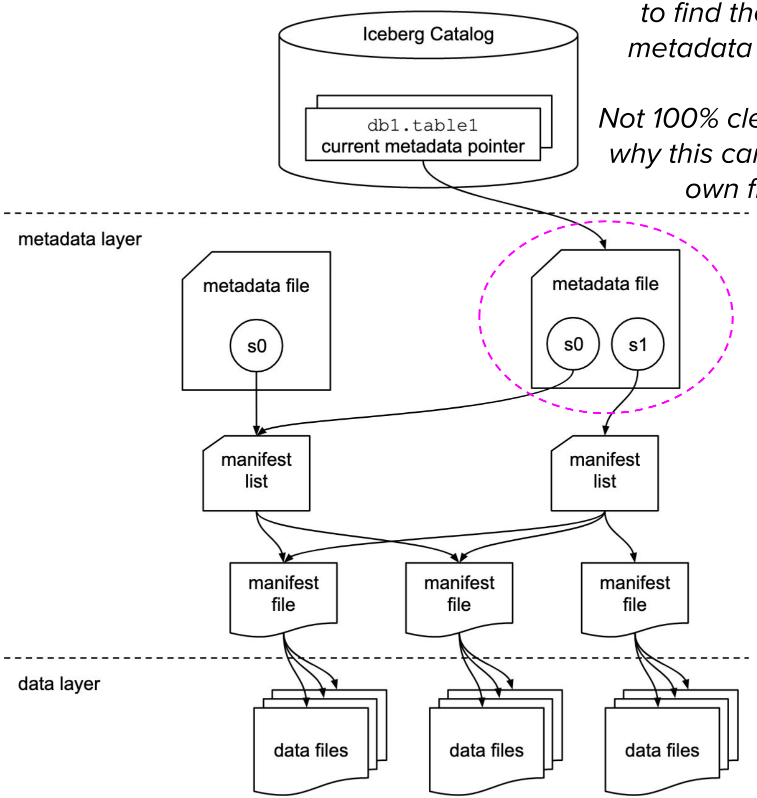
Table Formats

Apache
ICEBERG

Adds extra layer(s) of indirection

Still needs a catalog to find the root metadata file 🤔

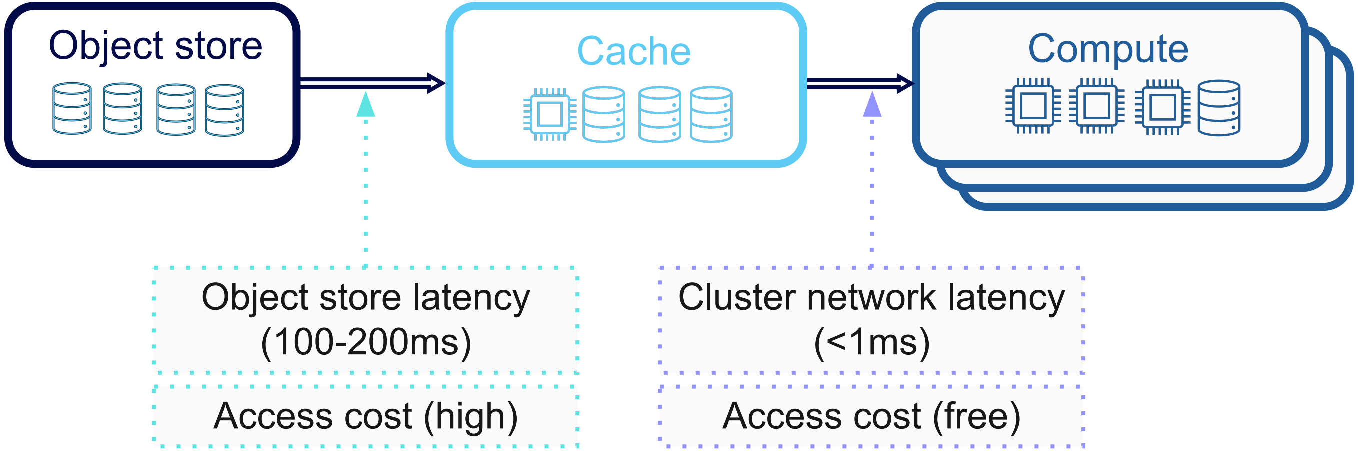
Not 100% clear to me why this can't be its own file



Object cache

Object storage latency, unpredictability, and cost per access

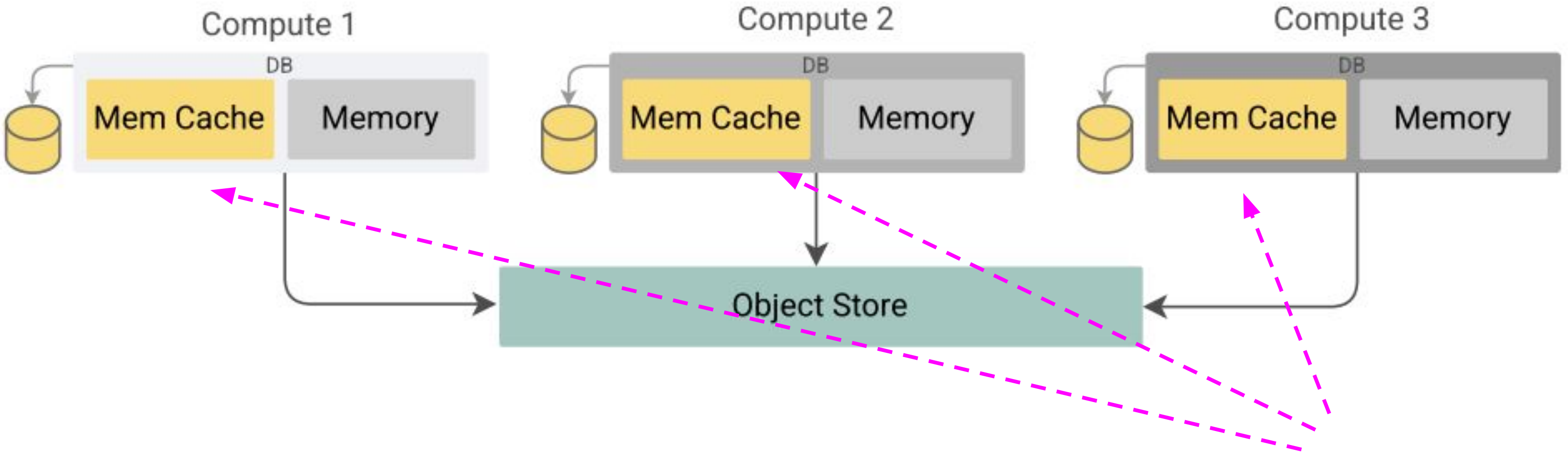
⇒ Reduce via in cluster caches



Credit: [Xiangpeng Hao](), UW Madison

 |

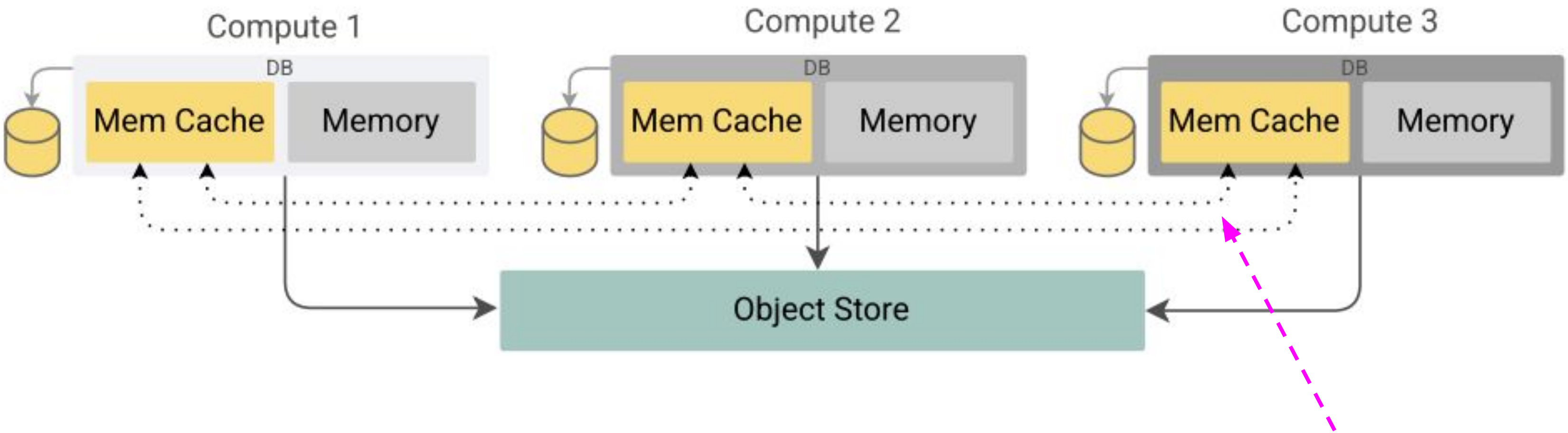
Object cache: Common Topologies



Compute nodes each have their own independent cache

CIDR 2025: [The Five-Minute Rule for the Cloud: Caching in Analytics Systems]()

Object cache: Common Topologies

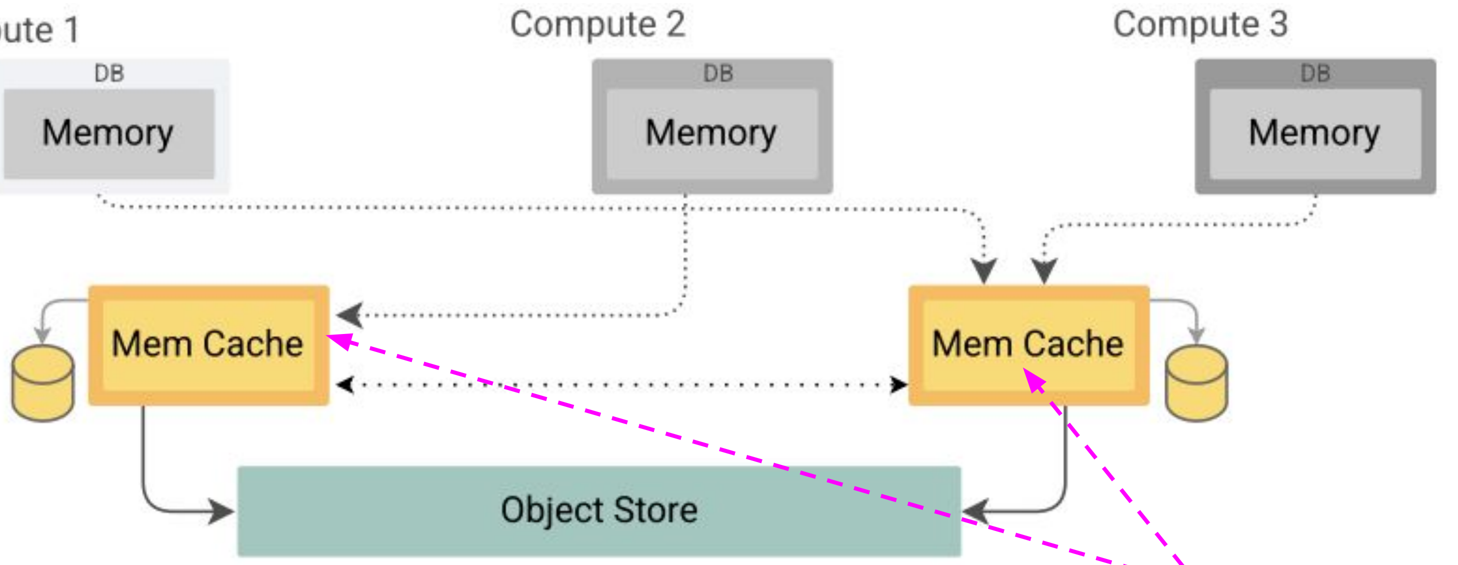


Compute nodes cooperatively manage a distributed cache

CIDR 2025: [The Five-Minute Rule for the Cloud: Caching in Analytics Systems]()

 |

Object cache: Common Topologies



Separate nodes manage the cache in a distributed system

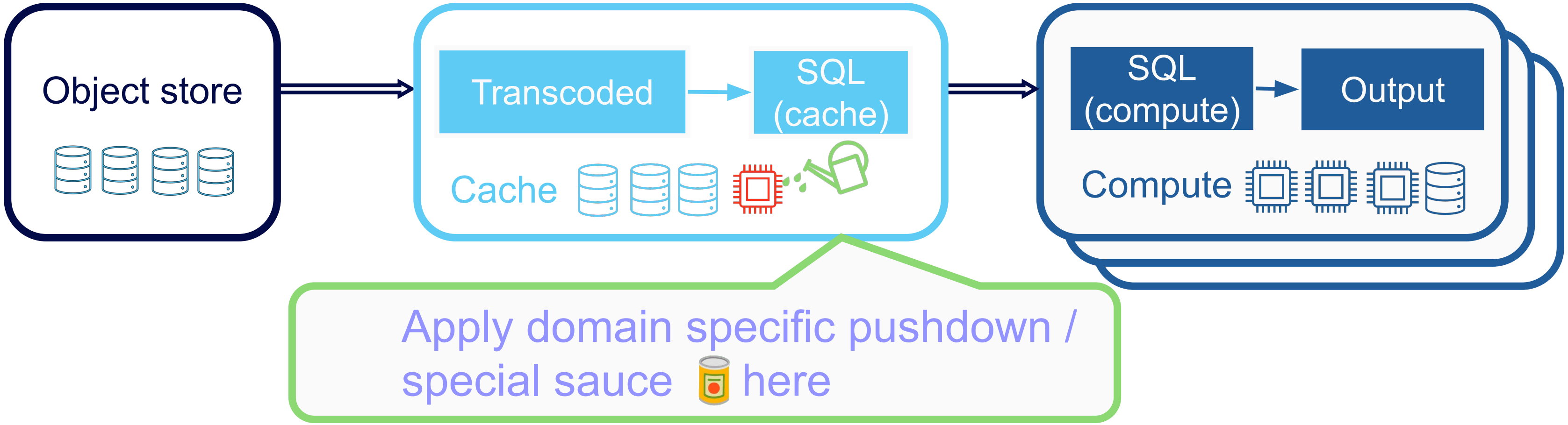
CIDR 2025: [The Five-Minute Rule for the Cloud: Caching in Analytics Systems]

 |

influxdata®

Object cache++

If you have a cache anyways, opportunities for transcoding

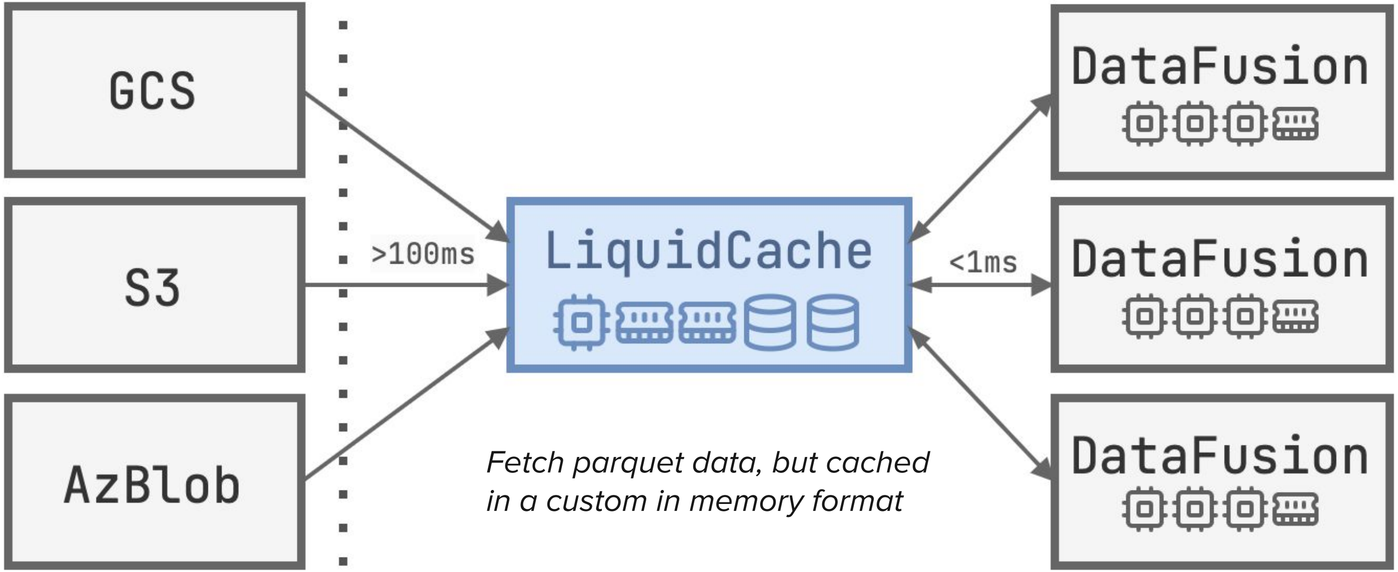


Credit: [Xiangpeng Hao](), UW Madison

 |

Object cache++

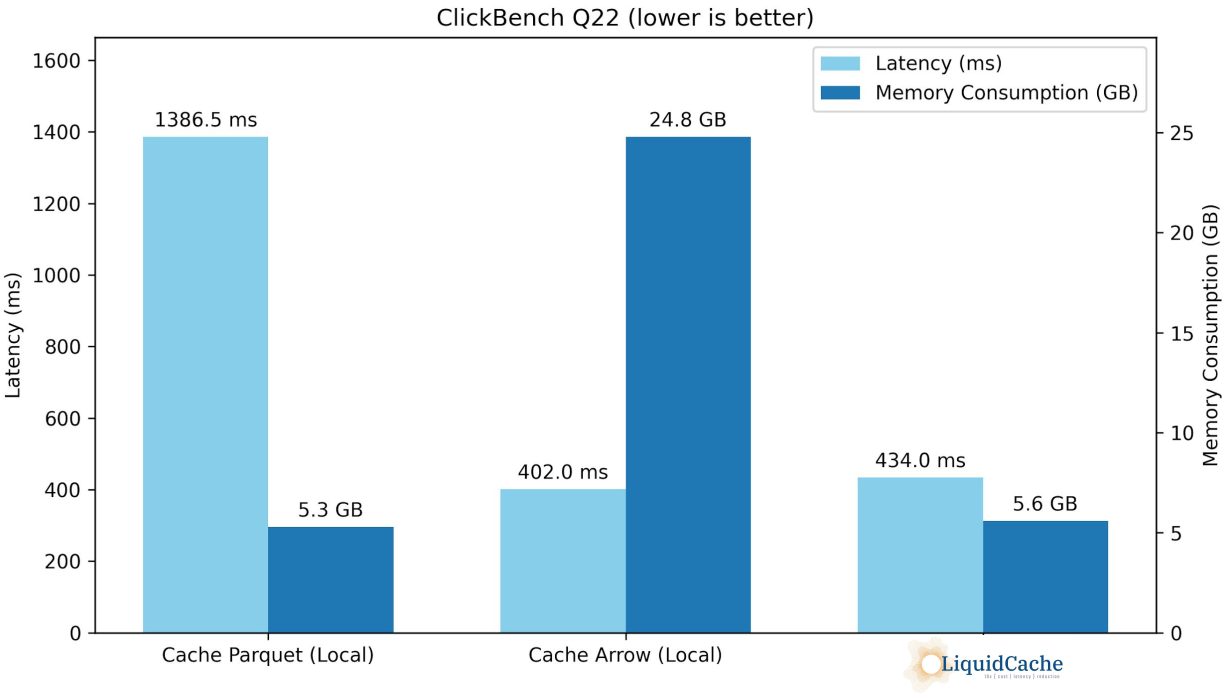
Example: https://github.com/XiangpengHao/liquid-cache



Fetch parquet data, but cached in a custom in memory format

Disclosure: I am an advisor to this project

Liquid Cache – ClickBench Q22

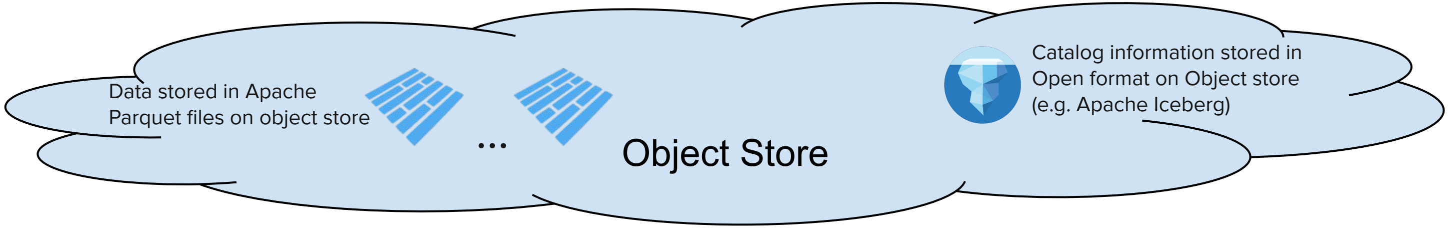



```sql
SELECT
"SearchPhrase",
MIN("URL"),
MIN("Title"),
COUNT(*) AS c,
COUNT(DISTINCT "UserID")
FROM hits
WHERE
"Title" LIKE '%Google%’
AND
"URL" NOT LIKE '%.google.%' AND
"SearchPhrase" <> ’’
GROUP BY "SearchPhrase"
ORDER BY c DESC
LIMIT 10;
```

Future Directions / Predictions

Increased adoption and interest in Open Formats

Specifically: Apache Parquet and Apache Iceberg



Implication: the classic business model of being the data platform that has huge data gravity (hard to move) may be changing

⇒ (Startup) Opportunities for many new specialized engines, etc.

Further disaggregation

Currently have disaggregated storage:

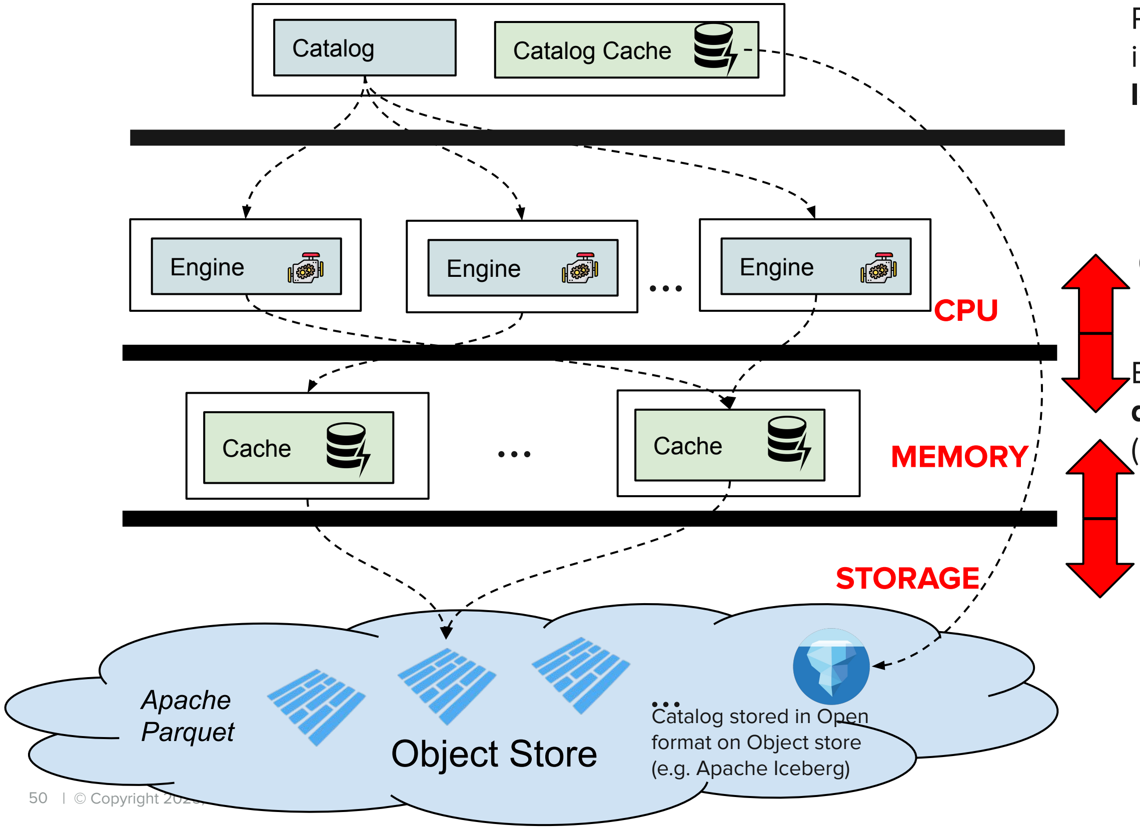
- Storage
- Memory + compute

Predict further disaggregation of memory from compute:

- Storage
- Memory (cache)
- Compute

influxdata®

Disaggregated Memory/Cache (2030s) *Conjecture*



Relevant catalog information **cached locally**

Elastically scalable **processing plane** (CPU intensive)

Elastically scalable **data cache** (memory intensive)

Data + Catalog stored in open formats *shared across systems*

Early signs

Driven by

- Serverless (stateless) processing,
- High bandwidth interconnect
- Demand for interoperability

 |

Thank you!
Questions?

 |