Cloud Data Lake Technologies

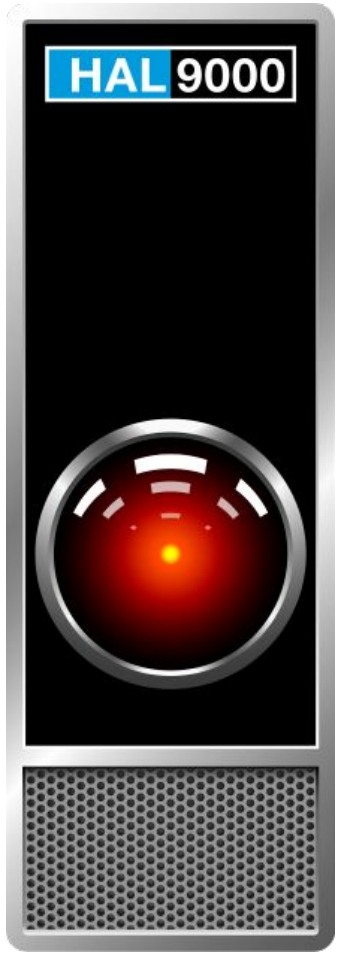
Ben Galewsky
University of Illinois

Nick Manganelli
University of Colorado, Boulder

Burt Holzman
Fermilab

National Center for Supercomputing Applications
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

About Me...



- Former IT Consultant in Industry
  - Led the deployment of big data stack at Kelloggs

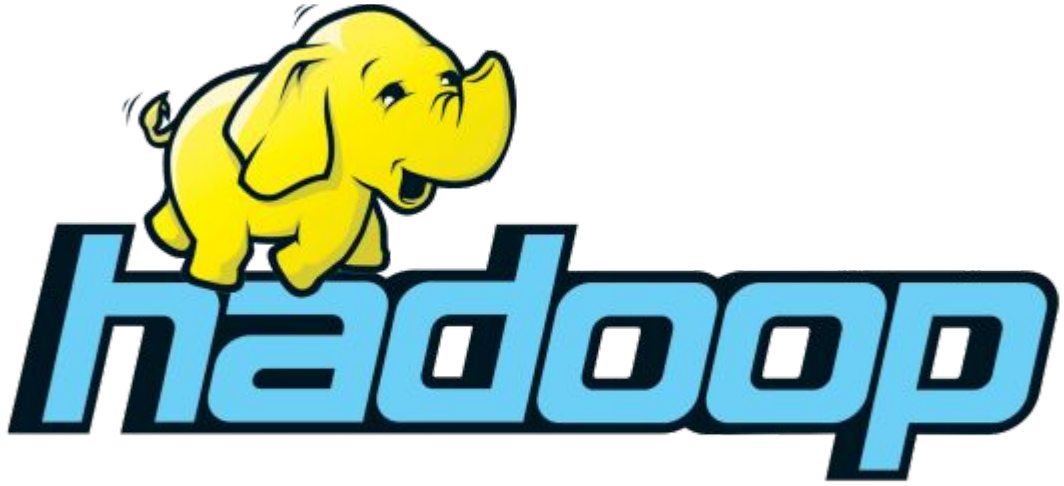
- Software Directorate at NCSA
- Collaborator on IRIS-HEP
  - Lead developer on ServiceX

Overview

- History of cloud data lake technologies
- Current state of the art
- Proof of concept with CMS Data
- Future directions

Big Data 1.0

- Hadoop Popularized big data
- HDFS
- Hive Distributed SQL



Big Data 1.0 - What Went Wrong

- Very complex to deploy and configure
- HDFS: adding more storage adds more compute
- Not a great fit in cloud computing environments

Zach Wilson (@eczachly

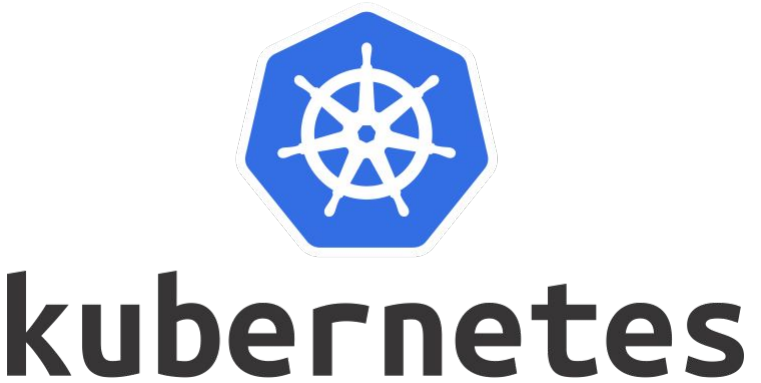
Some Important Developments

- Kubernetes
- Parquet
- Object Store
- Apache Iceberg
- Nessie
- Distributed SQL

Kubernetes

Kubernetes is an open-source container orchestration system that automates the deployment, scaling, and management of containerized applications across a cluster of nodes.

- Automates the distribution and scheduling of containerized application components across the cluster nodes.
- Provides load balancing, service discovery, storage orchestration, self-healing capabilities, and automated rollouts and rollbacks.
- Enables applications to be deployed, scaled, and managed consistently across different environments (on-premises, public cloud, hybrid, or multi-cloud).
- Abstracts away the underlying infrastructure, allowing applications to be portable and easily moved between environments.



perplexity.ai generated summary

Parquet File Format

Parquet Columnar Data Format:

- **Efficient storage**: Parquet files are highly compressed, resulting in significant storage savings compared to other formats like CSV. For example, a 13,193,045 record CSV file took up 8.6 GB, while the same data in Parquet format was only 1.6 GB, a 500% reduction in size
- **Fast data access**: Parquet's columnar storage layout allows for efficient querying, as only the relevant columns need to be read. This minimizes I/O and latency, making queries faster compared to row-oriented formats
- **Supports complex data**: Parquet can handle complex data types like arrays, maps, and nested structures that are difficult to store efficiently in simpler formats

perplexity.ai generated summary

Object Store

- **Scalability**: Object storage scales well and doesn't suffer performance lags as it scales
- **Simplicity**: Object storage systems are popular for their simplicity, ease of use, and cost-effectiveness compared to NAS systems
- **HTTP access**: Object storage is accessed via HTTP, making it very easy to access objects through a range of applications
- **Eventual consistency**: To enable scalability and cost-effectiveness, object stores were designed with eventual consistency, unlike POSIX file systems which require strong consistency

perplexity.ai generated summary

Apache Iceberg

- **Transactional consistency** between multiple applications where files can be added, removed or modified atomically, with full read isolation and multiple concurrent writes
- **Full schema evolution** to track changes to a table over time
- **Time travel** to query historical data and verify changes between updates
- **Partition layout and evolution** enabling updates to partition schemes as queries and data volumes change without relying on hidden partitions or physical directories
- **Ability to handle large datasets** with tens of petabytes and millions of partitions
- **File-level operations** enabling atomic changes to individual records without rewriting entire partitions

perplexity.ai generated summary

Nessie

Nessie is an open-source transactional catalog for data lakes that provides a Git-like experience for managing data tables and views.

- **Cross-table transactions**: Enables multi-statement transactions across all tables in a data lake, allowing you to make atomic updates to multiple tables at once.
- **Git-like branching and merging**: You can create branches of your entire data catalog to isolate changes, just like Git branches for code. Branches can be merged back to the main branch when changes are ready, enabling better collaboration and quality assurance.
- **Data versioning and time travel**: Since all changes are versioned like Git commits, you can view the state of data at any point in time and revert to previous versions if needed

perplexity.ai generated summary

Trino Distributed SQL

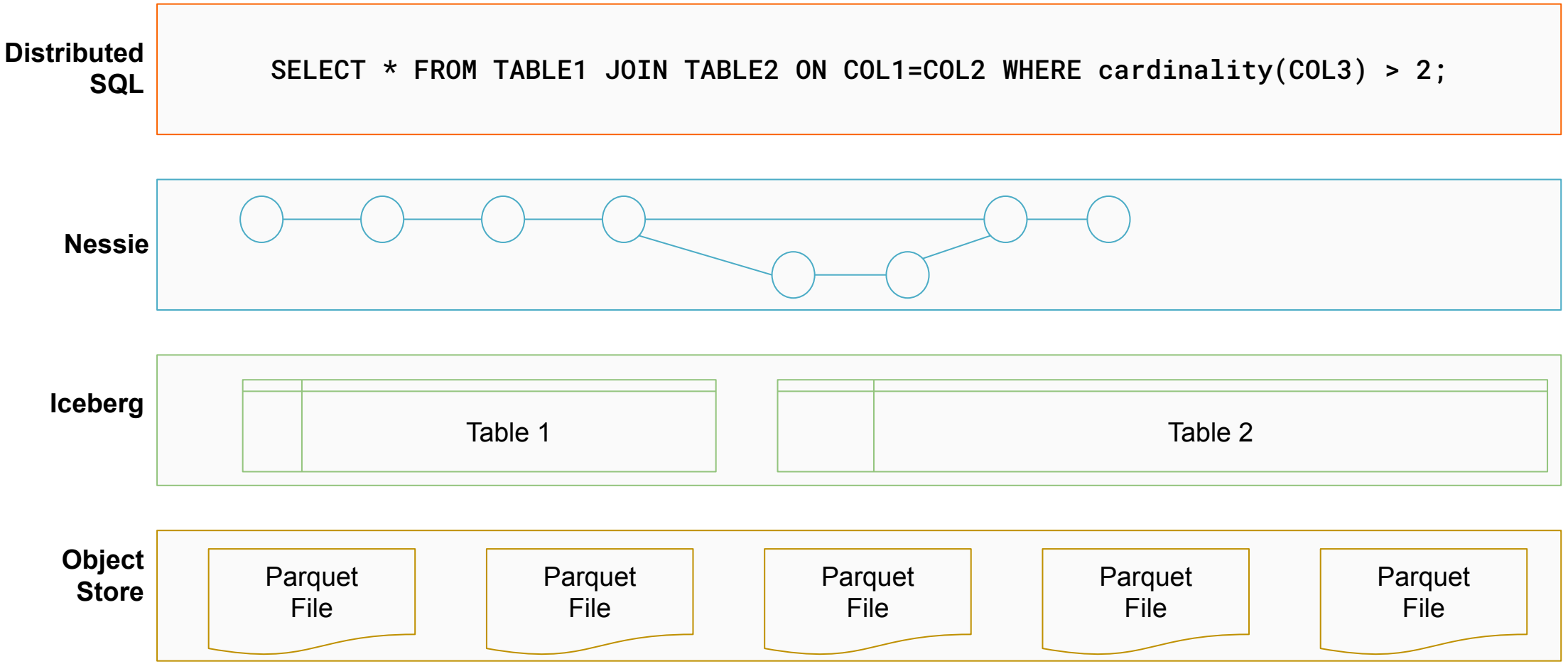
Trino is a distributed SQL query engine designed to efficiently query large data sets ranging from gigabytes to petabytes across multiple heterogeneous data sources.

- Not a database, but rather a SQL query engine that can query data from various sources like object storage (S3, Azure Blob, etc.), relational databases, NoSQL databases, and more without requiring data migration.
- Originally designed at Facebook to efficiently query their hundreds of petabytes of data stored in Hadoop/HDFS, replacing the slower Hive system.
- Leverages distributed query processing techniques like parallel in-memory processing, code generation, and optimized data structures to provide high performance on large data volumes.

trino

perplexity.ai generated summary

Modern Data Stack



How can we apply this to CMS data?

National Center for Supercomputing Applications
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Motivation

- As the LHC moves towards the HL-LHC era, the volume of data to be stored and processed will grow significantly
  - CMS AOD (450kB/event) - Limited Availability
  - MiniAOD (45kB/event) - Suitable for almost all analyses
  - NanoAOD(4kB/event) - Suitable for half of analyses
- Several competing needs create an impedance mismatch
  - Disk space comes at a premium
  - High throughput requires high availability and duplication across sites around the world
  - Traditional analysis workflows tend to duplicate information from large data-tiers via custom “Ntuples”, in order to create more streamlined but self-contained input data - for the half of analyses able to use NanoAOD, it’s nearly optimal and can obviate the need for intermediate Ntuples

Typical Problems

- An analysis may be able to use NanoAOD, but must store expensive ML outputs
  - Duplicate all necessary input data from NanoAOD + added ML information
- An analysis may have 90% of data needs met by NanoAOD, but the additional requirements drive it to use MiniAOD or AOD
  - Custom NanoAOD variant (superset of central variation) or custom NTuple format created, duplicating a significant amount of centrally-stored events

Use SQL Joins!

cmsopendata2015_ttbar
run
luminosityBlock
event
Electron_pt
Electron_eta
Electron_phi

```
SELECT * FROM cmsopendata2015_ttbar
JOIN ttbar_infer ON
   cmsopendata2015_ttbar.run = ttbar_infer.run
AND
   cmsopendata2015_ttbar.luminosityBlock=ttbar_infer.luminosityBlock
AND
   cmsopendata2015_ttbar.event = ttbar_infer.event
```

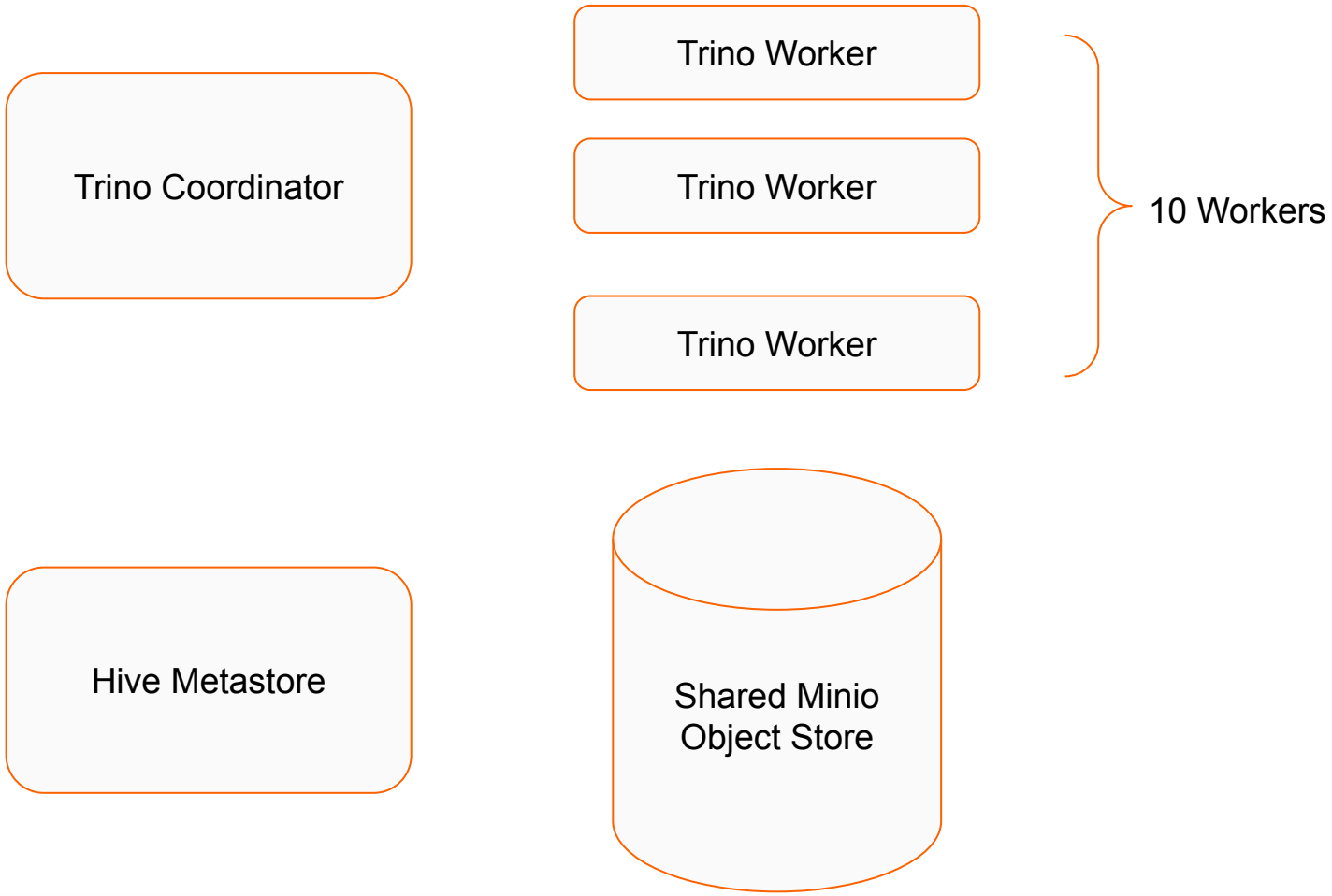
ttbar_infer
run
luminosityBlock
event
GNN_p1
GNN_p2

Notes

- All datasets must be in Parquet format
  - Use uproot or ServiceX to convert from ROOT
- Parquet and Trino support jagged arrays
  - Perform simple arithmetic operations and filters in query
- Performance is sufficient to make the outputs temporary and reproducible

Exploratory Environment at FNAL

- Deployed in FNAL OpenShift cluster
- Shared Mino Object Store



Some Initial Findings

- NanoAOD Dataset: 13M Events
- GNN Inference: 8 Inference Values
- Simple Join: 7 Seconds
- Output to Parquet file: 24 Seconds

- So far no tuning or attention paid to partitioning
- Output consumed by Coffea on DASK to produce histograms
- Still some limitations in Awkward DASK reading Parquet
- Parquet is as efficient or better than ROOT wrt size/processing time

Future Directions

- Group N-Tuples maintained in Cloud Data Lake.
  - Use Nessie to manage dataset updates without breaking existing analysis
  - SQL based cuts during development
- Use Kafka to stream joined results to *Coffea*
  - Ability to replay
  - Limited TTL to avoid disk space explosion
- Deploy Cloud Data Lake as part of Analysis Facility
  - Maintain Parquet Data Lake of NanoAOD or PHYSLITE datasets

Conclusions

- Cloud Data Lake technologies are mature and highly scalable
- Presents an opportunity to reduce the amount of storage required for modern analysis
- Require a commitment to Parquet
- Some care needed in the structure of the Parquet files
- Ready for testing now

Gratitude

- Lindsey Gray for initial idea and helping assemble team
- My IRIS-HEP colleagues

This project is supported by the National Science Foundation under Cooperative Agreements OAC-1836650 and PHY-2323298. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

This work was performed with support of the U.S. CMS Software and Computing Operations Program under the U.S. CMS HL-LHC R&D Initiative. This work was partially supported by Fermilab operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the Department of Energy, and by the National Science Foundation under grant ACI-1450377 and Cooperative Agreement PHY-1120138. Additional support came from the Department of Energy DE-SC0010005 grant.