KubeCon | CloudNativeCon
India 2025

Bridging Data & ML ecosystems: A cloud Native Approach using Kubeflow

Johnu George
Shivji Kumar Jha

About the Speakers

Johnu George
Technical Director
Nutanix AI

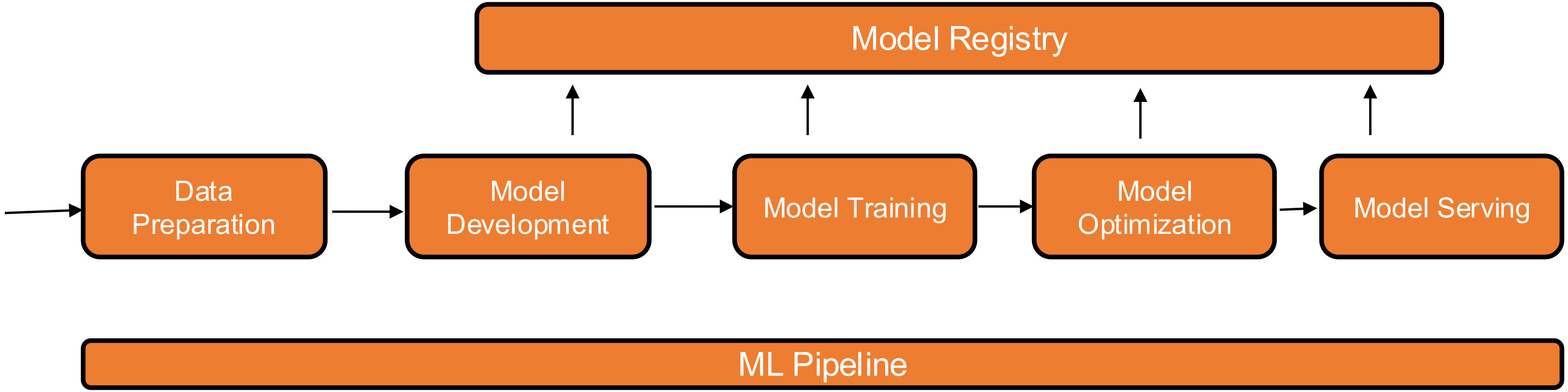
Shivji Kumar Jha
Staff Engineer
Data Platforms at Nutanix

- Kubeflow Steering Committee Member
- Kubeflow Training & AutoML WG chair
- MLCommons Storage Chair

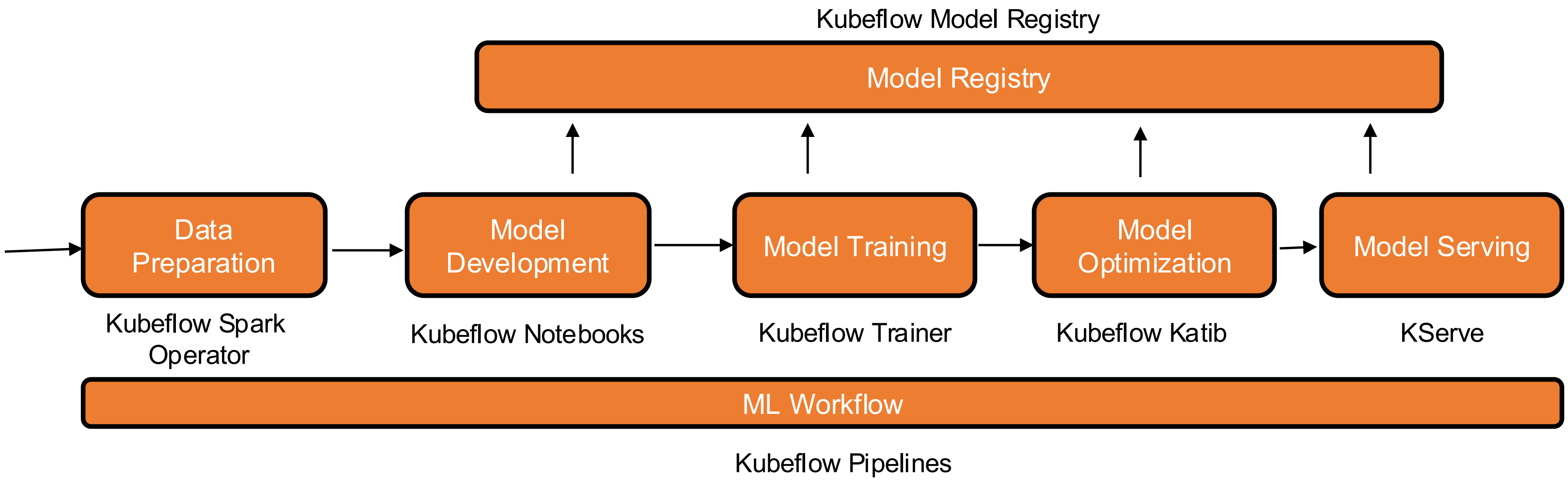
- Software Engineer: Distributed systems / databases
- Contributed code to MySQL, Pulsar, Clickhouse
- Excited about Open-Source Software & Communities
- Past Talks: https://github.com/shiv4289/shiv-tech-talks/

Table of Contents

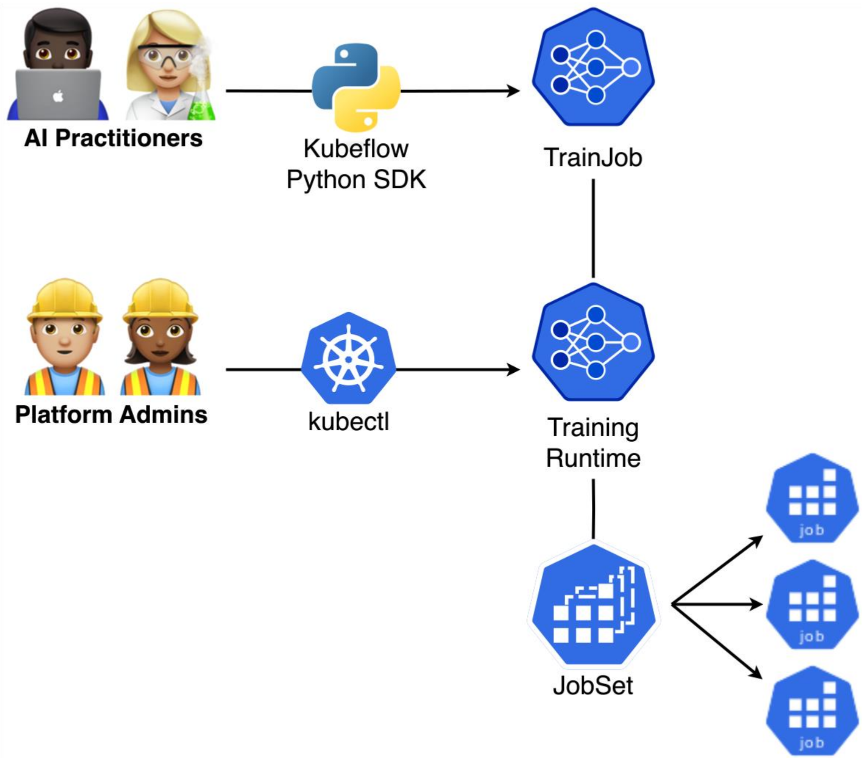
MLOps Pipeline



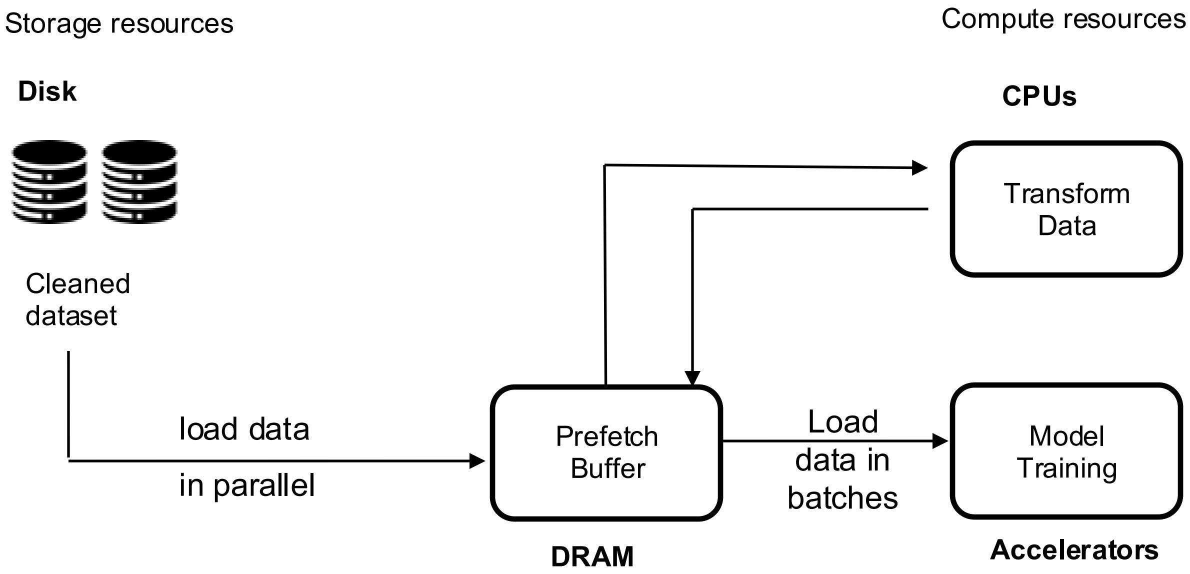
Kubeflow Platform



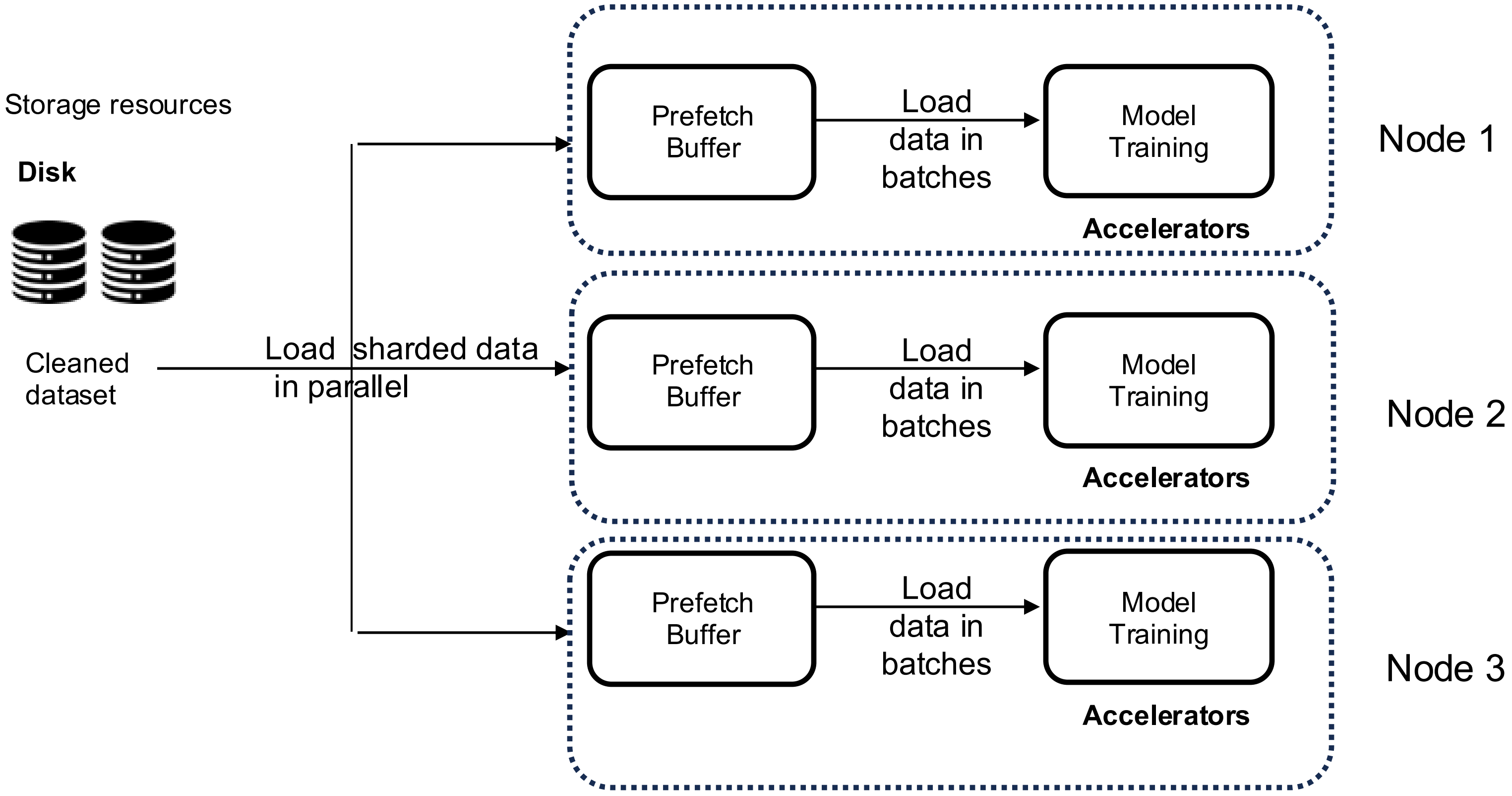
Kubeflow Trainer



Single Node ML training



Multi-Node ML training (DDP)



Challenges in ML stack

- Efficient storage for large datasets
  - Larger dataset, better the model
- GPU under utilization
  - High data throughput requirement to keep GPU busy
- Sharding data across workers
  - Unique subset of data for every worker
- Consistent shuffling of data across epochs
  - Restream data in every epoch
  - Network bandwidth wastage
- Supporting heterogeneous training jobs on same dataset
  - I/O wastage due to static data sharding pattern

Async storage access

```
# Repeat for n epochs
for epoch in range(n_epochs):
        for i, data in enumerate(training_loader):

                # Every data instance is an input + label pair
                inputs, labels = data

                # Zero your gradients for every batch!
                optimizer.zero_grad()

                # Make predictions for this batch
                outputs = model(inputs)

                # Compute the loss and its gradients
                loss = loss_fn(outputs, labels)
                loss.backward()

                # Adjust learning weights
                optimizer.step()
```

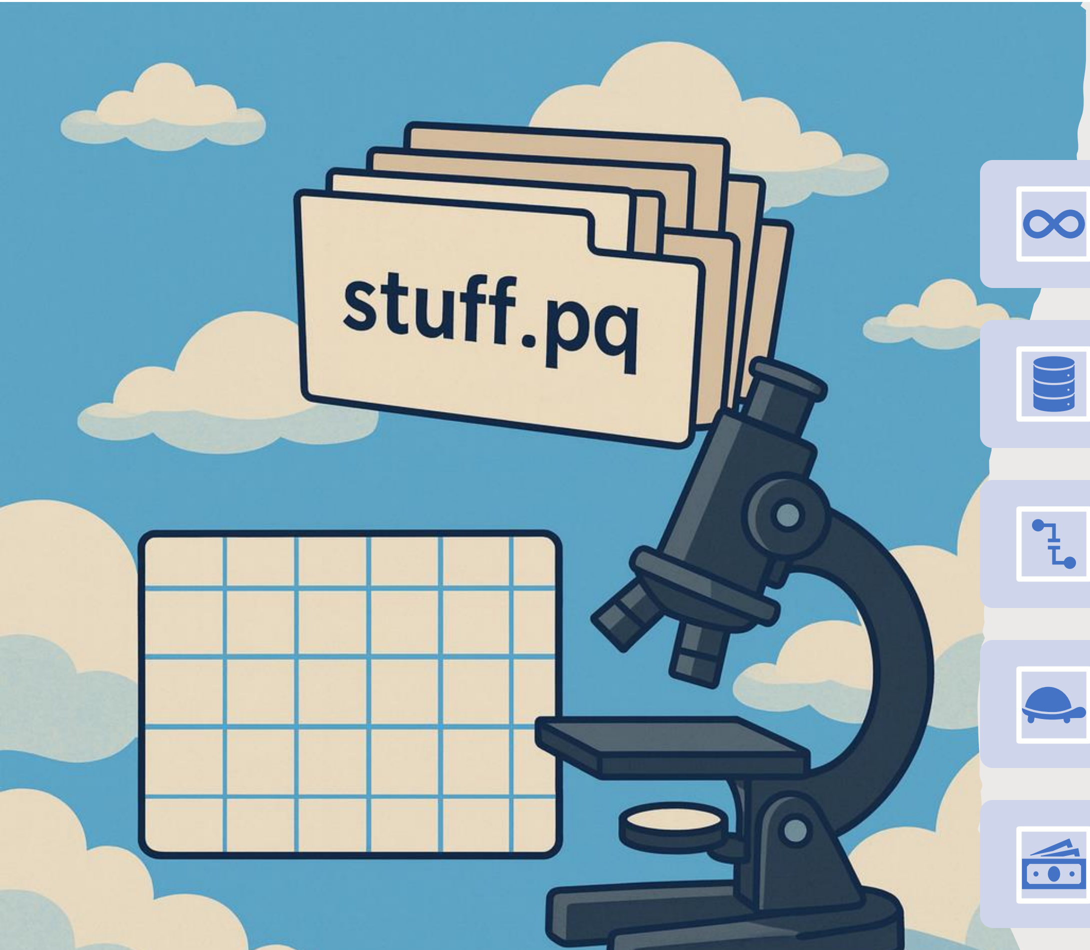
Next-gen data stack

Storage Efficiency

Apache Parquet

- Large data set needs efficient storage
- Parquet is a **compressed, columnar storage format**
- Lower storage & IO cost.

Easy Access to Parquet Files

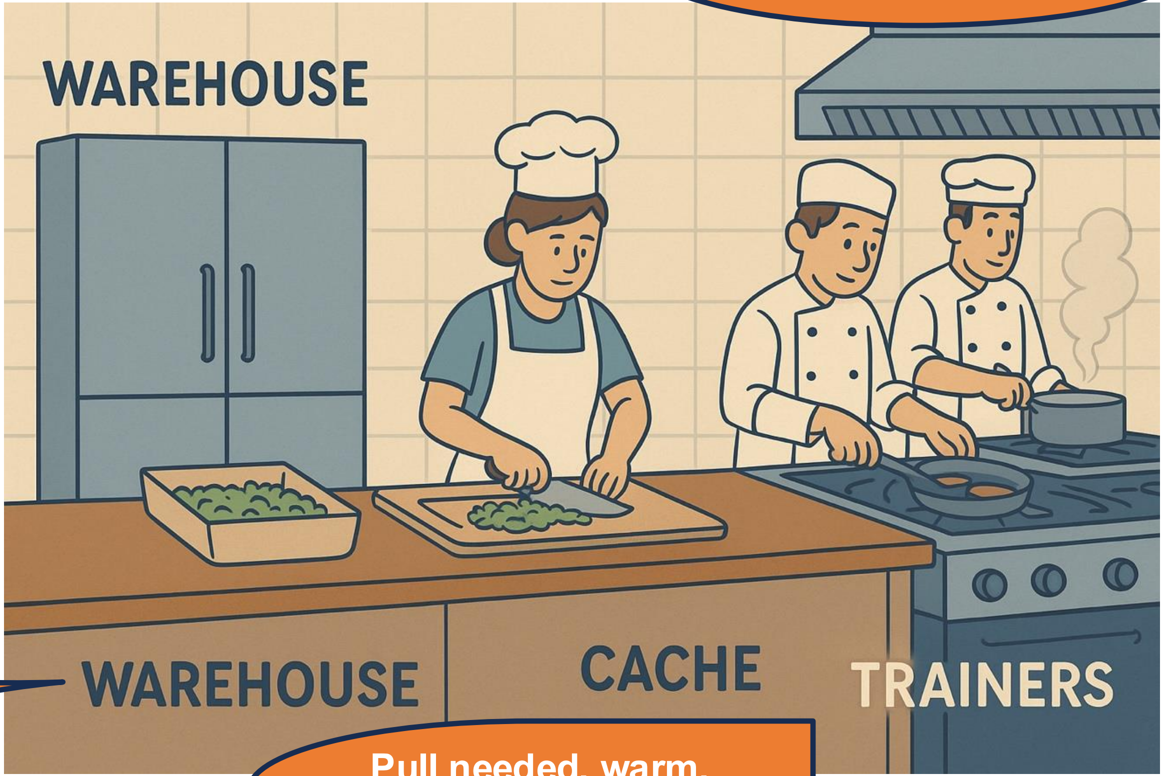


Apache Iceberg

- S3 means - Infinite Storage & good HA
- **Open** Table format (SQL & ACID)
- Lots of integrations -> access patterns
- But… slow access
- And high transfer costs

Subway analogy

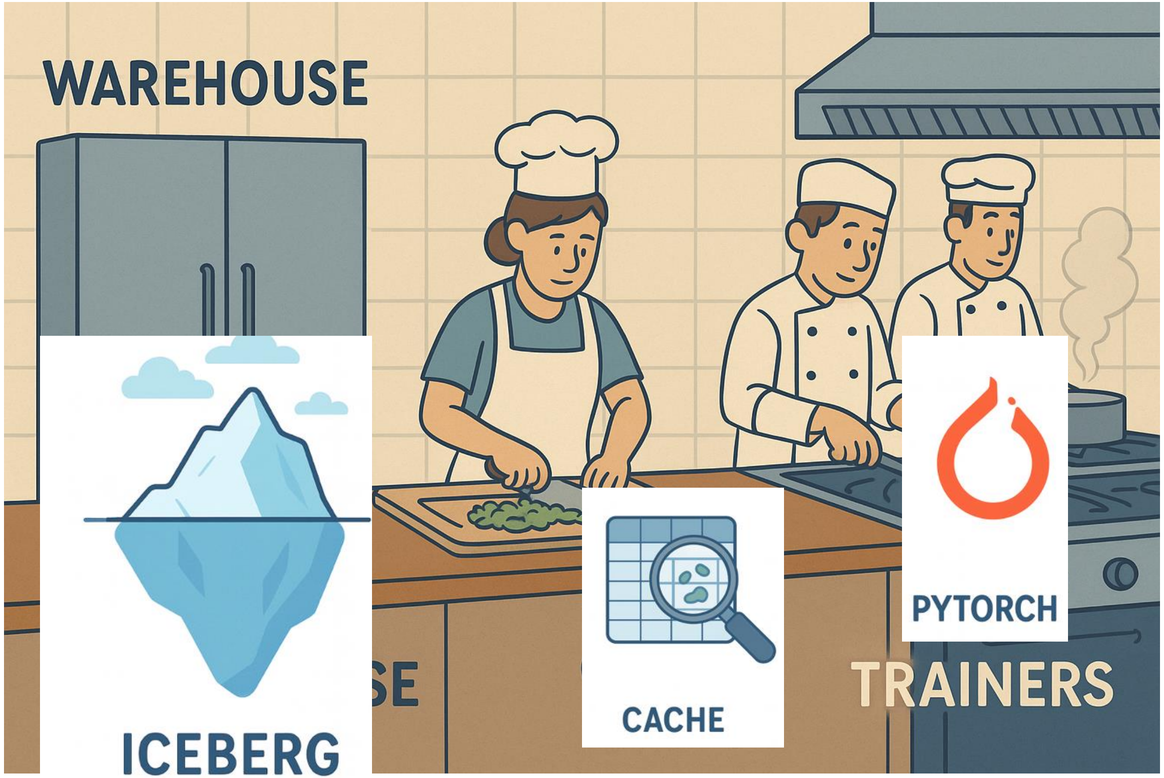
Grab prepped ingredients & cook fast



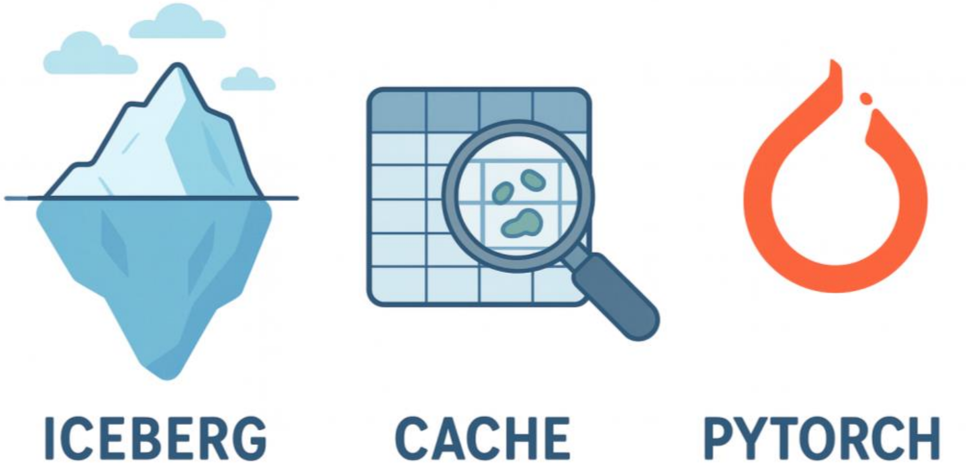
Bulk Storage (Frozen Veggies, meats, sausage)

Pull needed, warm, clean, sort; ready to pick

SUBWAY analogy



Introducing a Cache for trainers

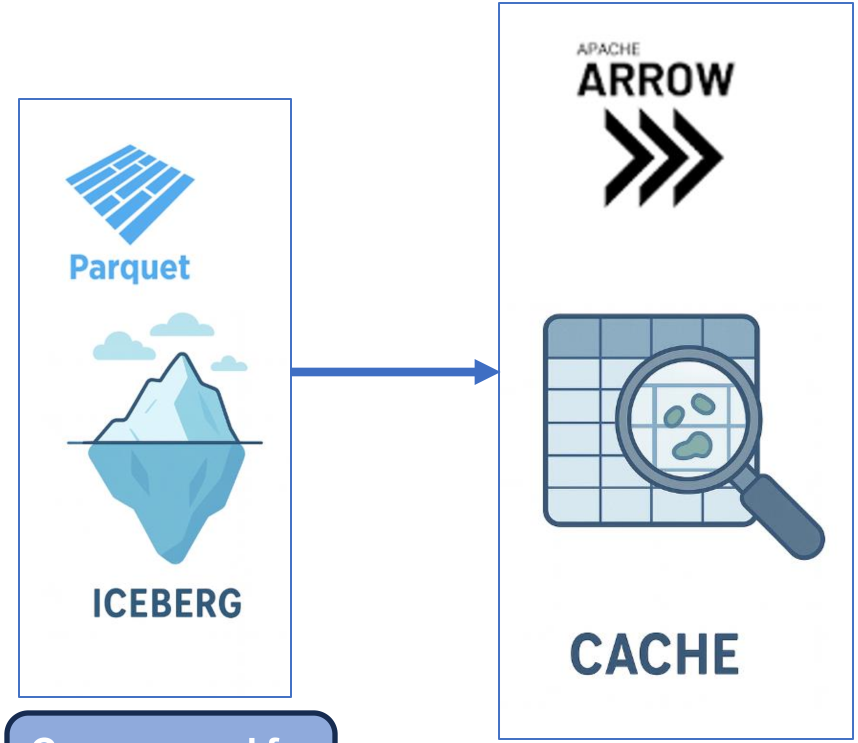


Distributed
GPU-enabled
ML training workers

Store huge datasets in a cost-effective & open format

For easy access of prepped data to training workers

Cache Implementation

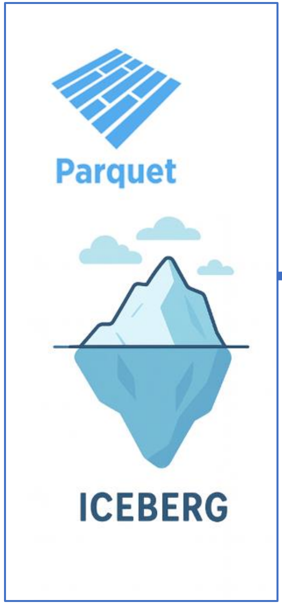


Compressed for efficient storage

Optimized for training

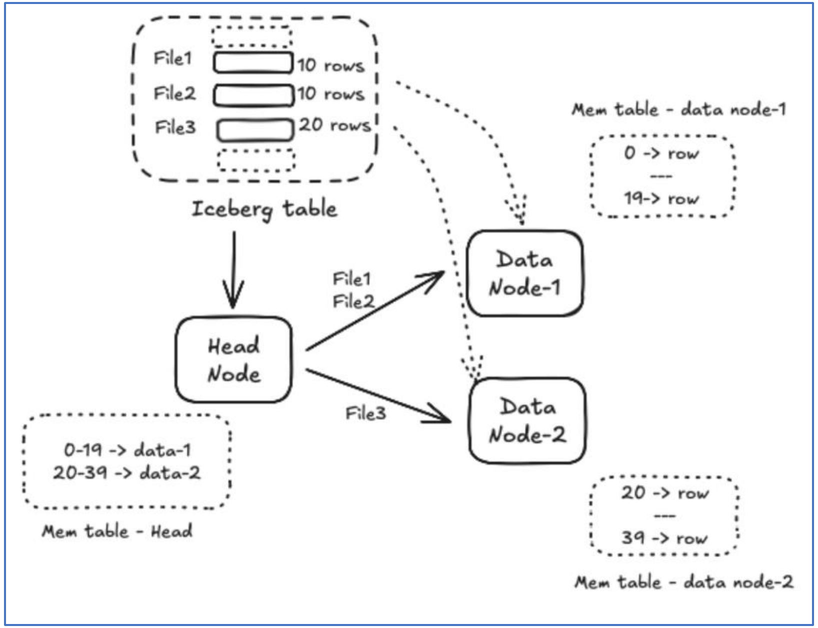
- Cache Data is kept in Arrow format
- Efficient in-memory storage & transport
- Cache nodes convert Parquet from (Iceberg → Arrow RecordBatches)

Moving Data from Iceberg to Cache

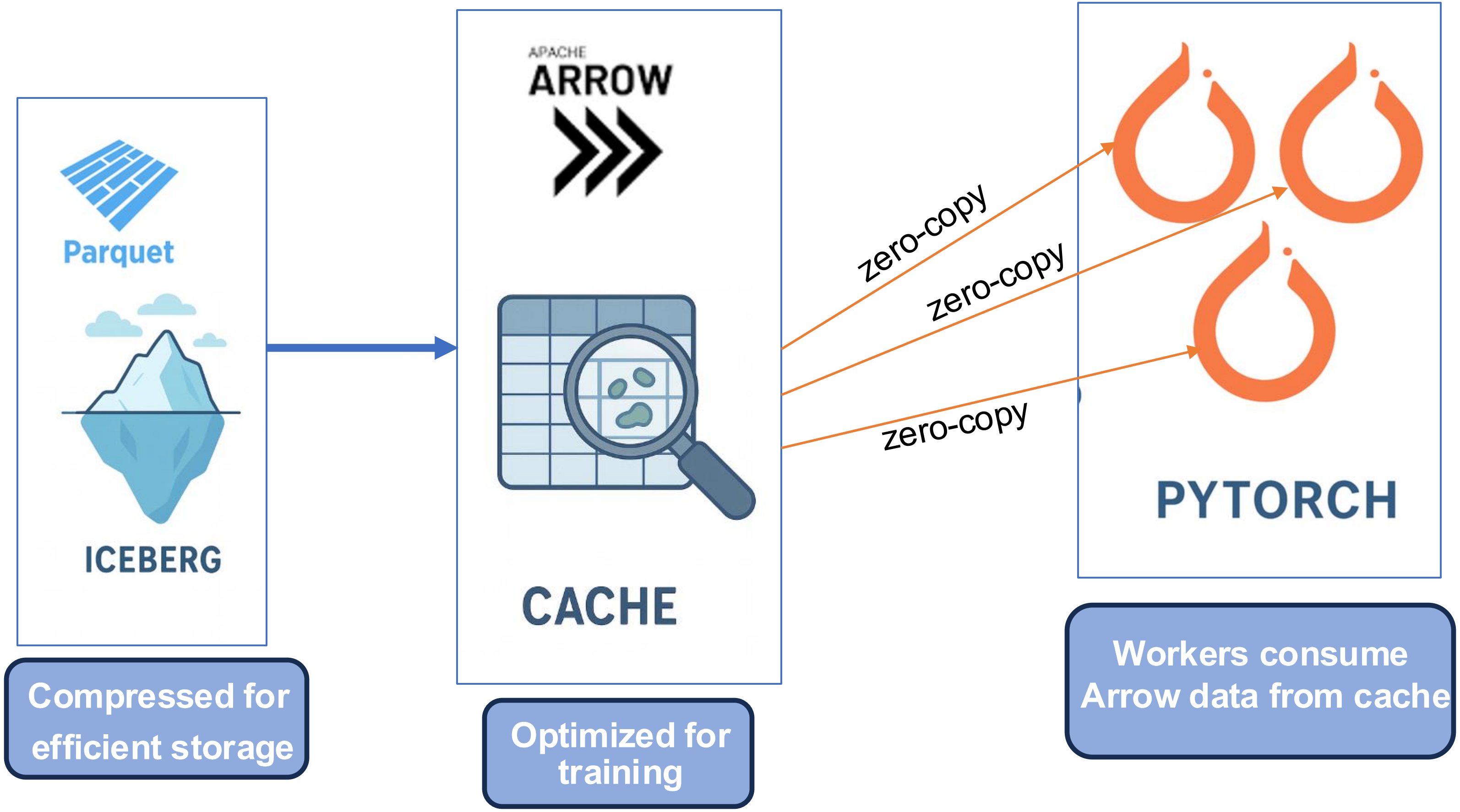


Compressed for efficient storage

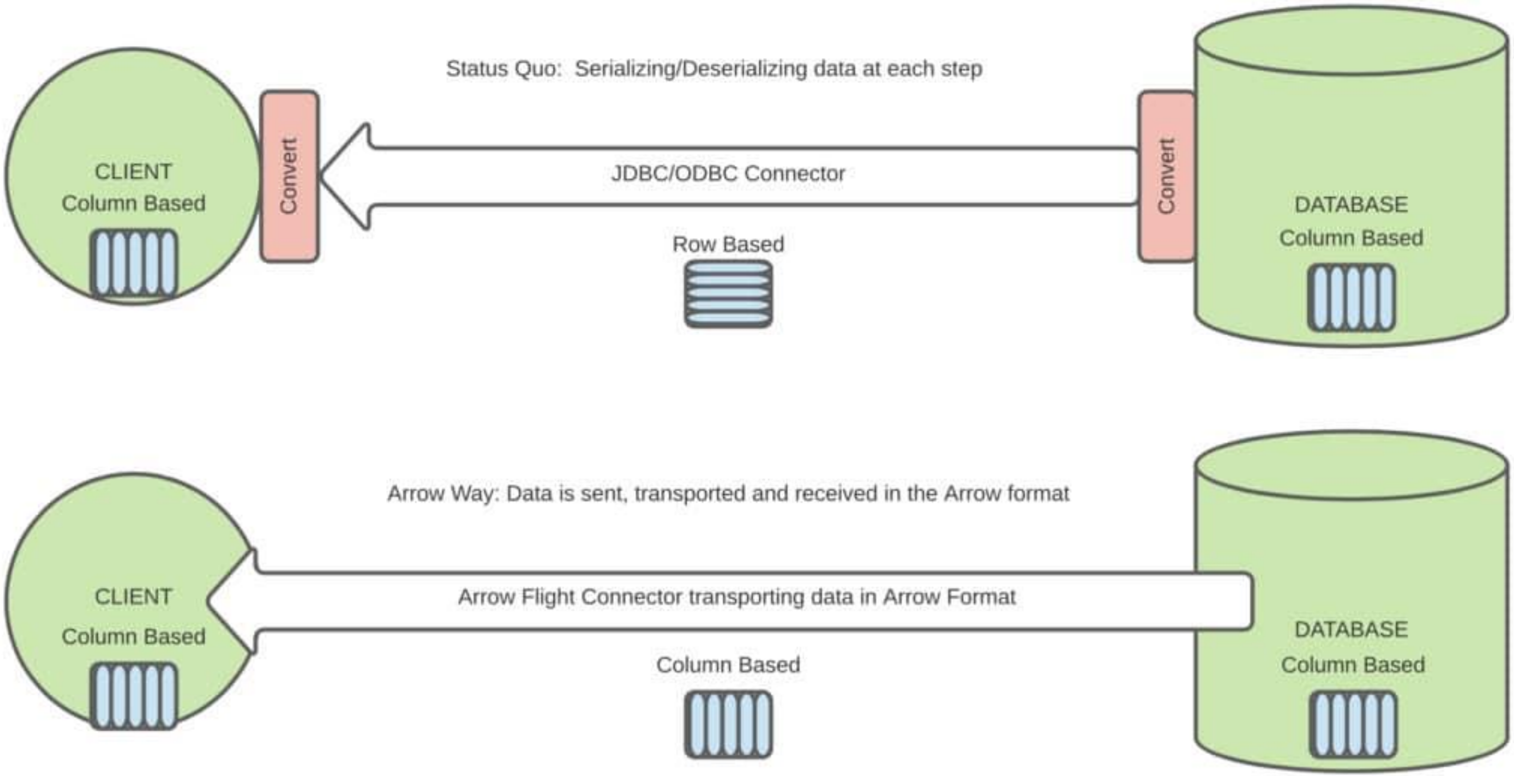
Optimized for training



Fetching data from Cache

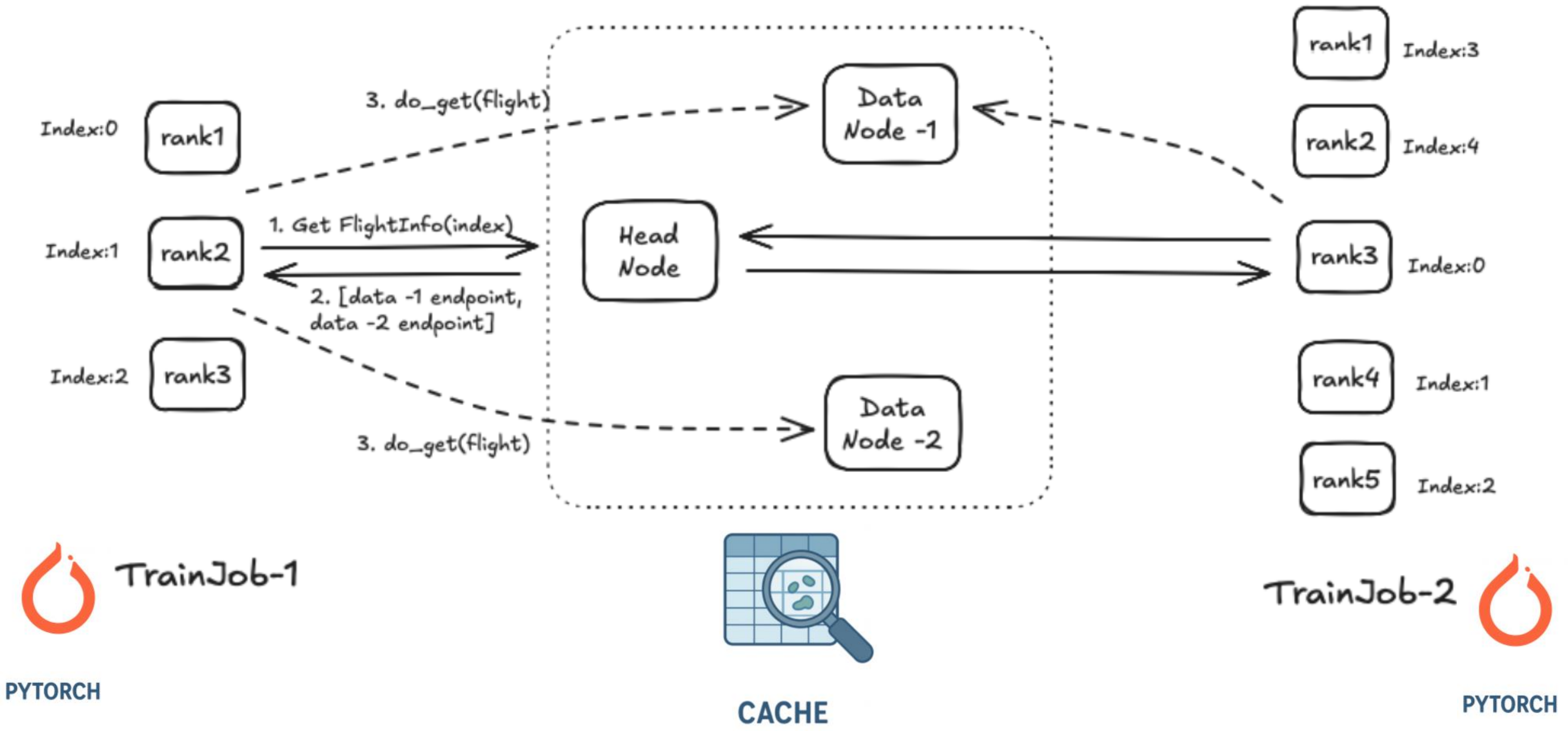


Arrow Flight: zero-copy transport



https://www.dremio.com/blog/an-introduction-to-apache-arrow-flight-sql/#h-apache-arrow-flight

A Cache built for training workers

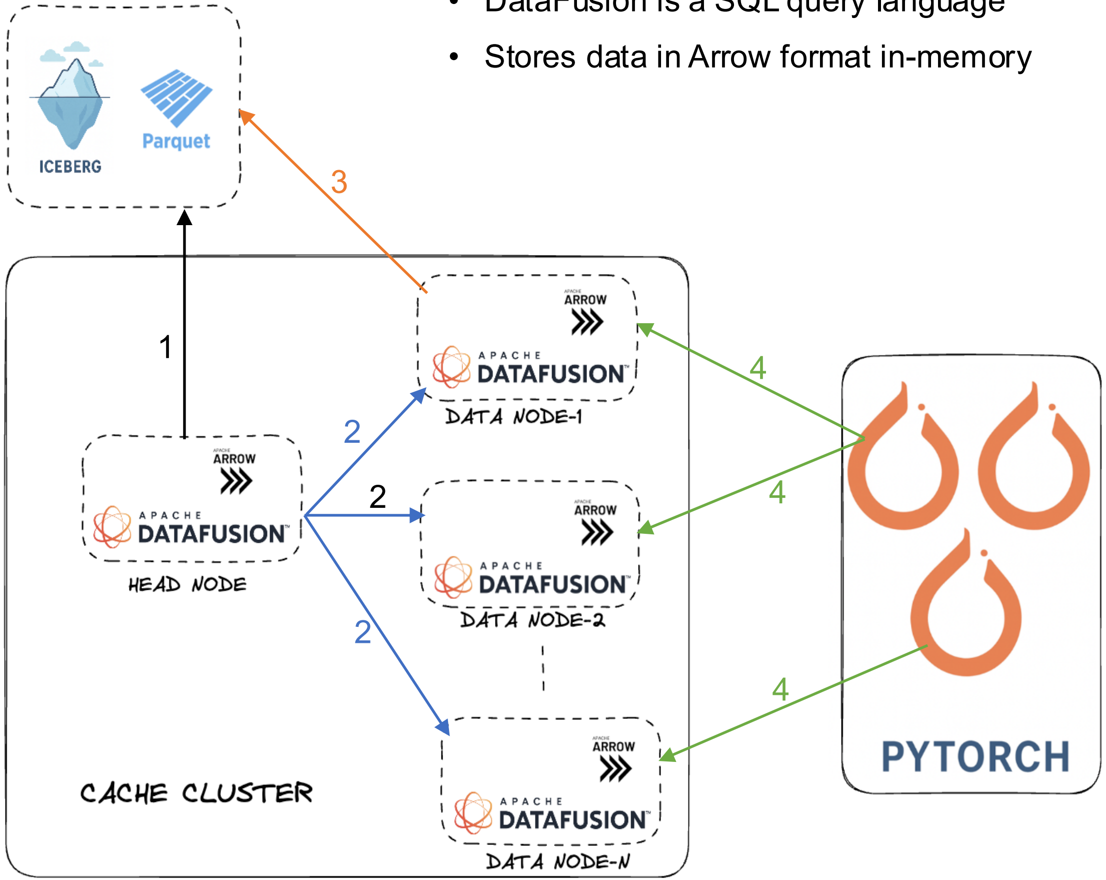


Putting it all together

DataFusion

1. Queries Metadata from Iceberg
2. Head node assigns slices to data nodes.
3. Data nodes fetch, store data (arrow)
4. PyTorch workers fetch required data

- DataFusion is a SQL query language
- Stores data in Arrow format in-memory



Revisiting the challenges

1. Efficient storage for large datasets
   - Larger dataset, better the model

Apache Parquet

2. GPU under utilization
   - High data throughput requirement to keep GPU busy
3. Consistent shuffling of data across epochs
   - Restream data in every epoch
   - Network bandwidth wastage

Arrow Flight

Custom Infra for ML training

4. Sharding data across workers
   - Unique subset of data for every worker

indexed data storage

5. Supporting heterogeneous training jobs on same dataset
   - I/O wastage due to static data sharding pattern

Dynamic assignment based on #workers

Let’s write some code with Kubeflow SDK!

“Hello cache!”: with new data stack


```
client.train(
runtime_ref="torch-distributed-with-cache",
trainer=CustomTrainer(
    func=train_func,
    num_nodes=3,
        resources_per_node={
              "cpu": 3,
              "memory": "6Gi",
              "nvidia.com/gpu": 1,
        },
),
initializer =
  Initializer(dataset=ArrowCacheDatasetInitializer(
        cluster_size="3”,
        metadata_loc="s3a://testbucket/feature_demo/data/yelp_review_v2/metadata/1234.metadata.json",
        table_name="yelp_data_v2",
        schema_name="feature_demo",
        features:[],
          filter:"",
)),
Runtime=torch.runtime
)
```

Join us!

Huge shoutout to Akshay Chitneni, Andrey Velichkevich, Rasik Pandey

Design: Google Doc

Issue Tracker: https://github.com/kubeflow/trainer/issues/2655

Proposal: https://github.com/kubeflow/community/pull/864

Implementation : https://github.com/kubeflow/trainer/pull/2755

Call for contributors

Questions?

Get in touch:

Johnu George - https://www.linkedin.com/in/johnu-george-83036610/

Shivji Kumar Jha - https://www.linkedin.com/in/shivjijha/