DATA+AI SUMMIT BY databricks

yipitDATA

Activate Your Lakehouse with Unity Catalog

Anup Segu
Co-Head of Data Engineering, YipitData

Databricks
2023


Photo by Lewis J Goetz

A Great Lakehouse is Active

Business operations should be routinely using the Lakehouse

Fast Time to Insight

Data driven decisions are the norm

Integrates with Broader Stack

Establish source of truth across organization

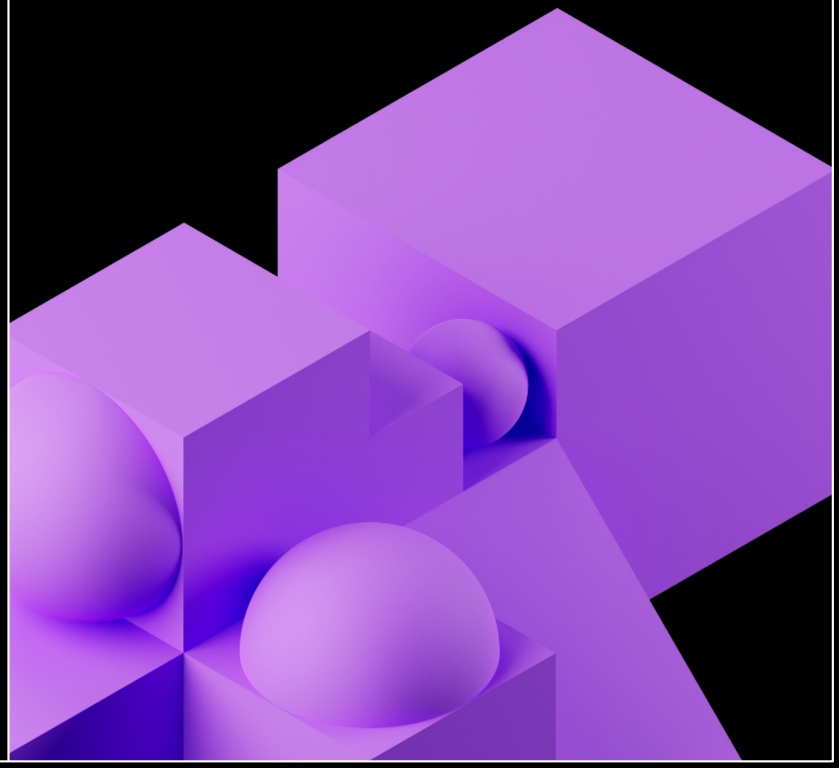
Effective Data Governance

Data assets deployed strategically

This is a story of how we
grew with the Lakehouse

And how Unity Catalog solves
our growing pains

Who are we?

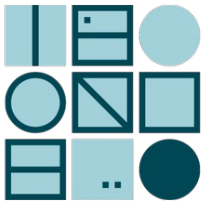


YipitData answers key questions for Investors and Corporations

yipit DATA

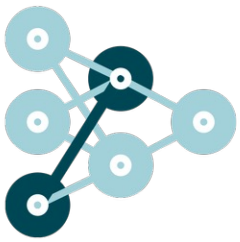
Trusted source of insights using Alternative Data

- Analyzes billions of data points every day to provide accurate, granular insights on **1000+ companies**.
- **470** investment funds and innovative companies rely on us to help them make better business decisions.
- Services include research reports, dashboards, live data feeds, and custom solutions.



Multi-Channel Data Sources

Benchmarking and Data Accuracy

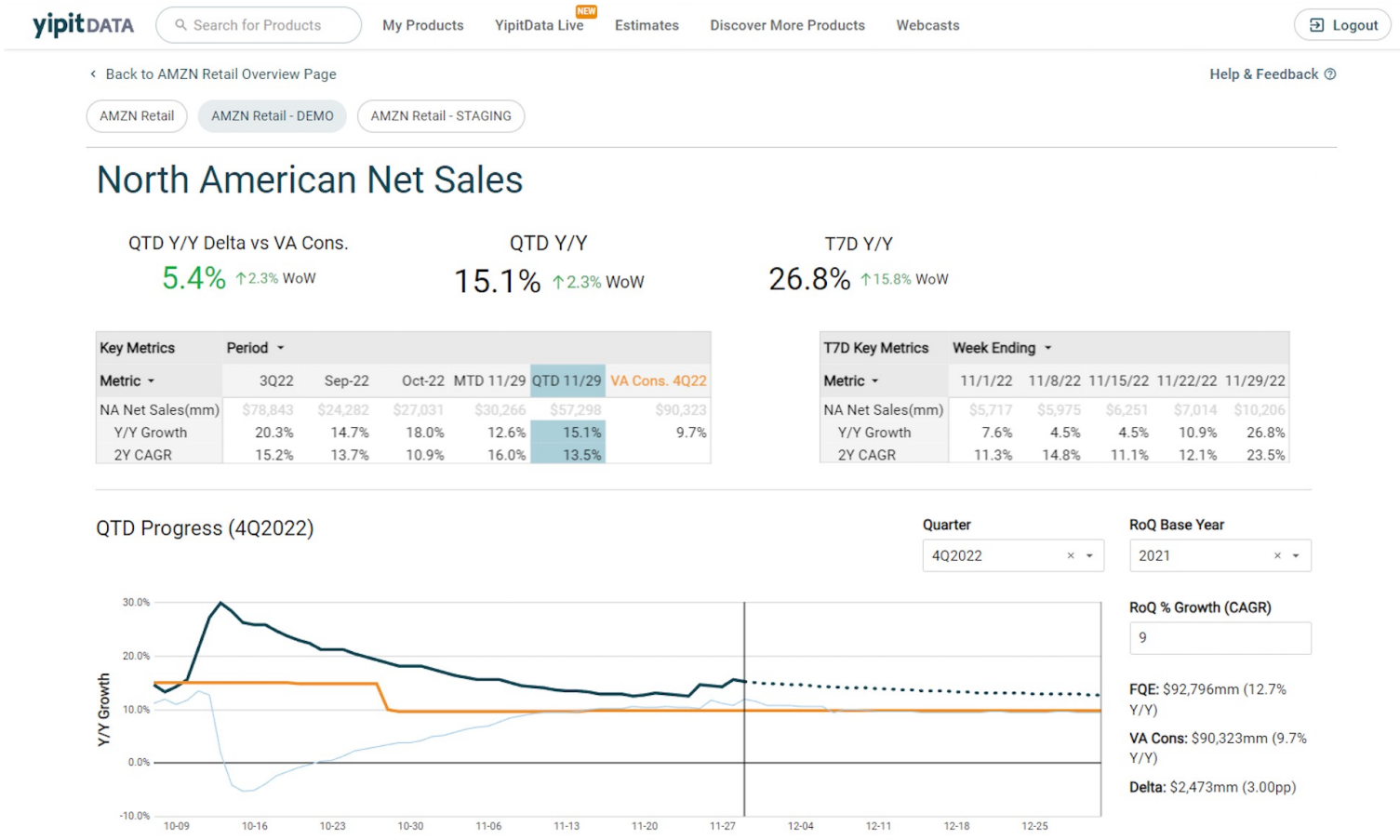


Breakdown of Growth Drivers

Market Share and Category Analytics

Delivering Insights Every Day to Clients

Delivering Insights Every Day to Clients



YipitData Lakehouse Overview

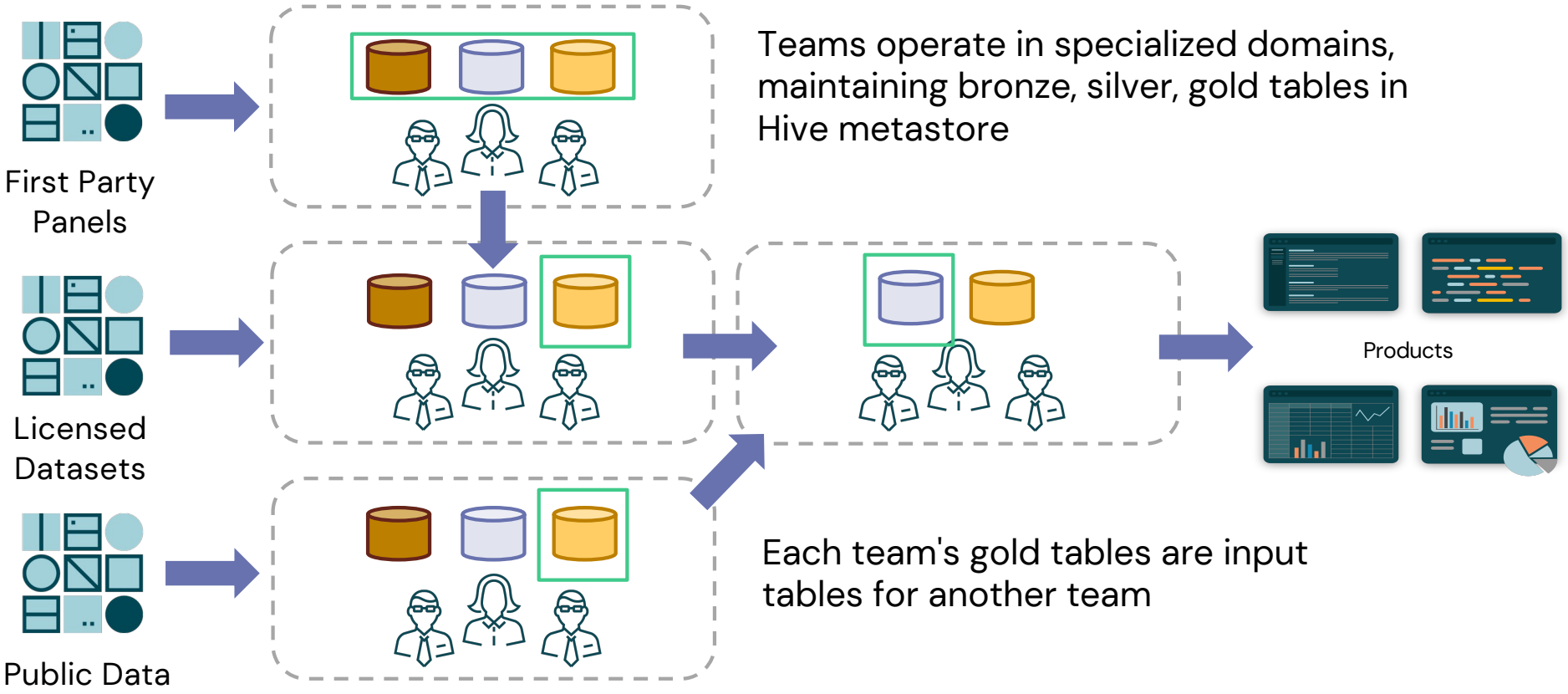
16 PB of Data

2K+ Daily Pipelines

210K Tables

- Bronze, Silver, Gold Medallion Architecture
- 290+ data team independently manage pipelines
- Platform supported by 13 Data Engineers

Self-Service Data Platform at YipitData



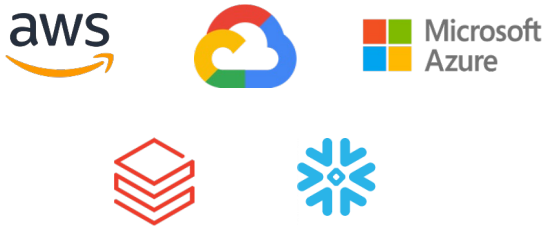
Self-Service Data Platform at YipitData

Broad tooling for analyst personas to operate like data engineers

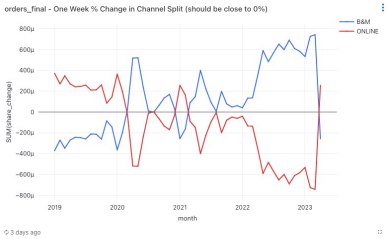
```
from ydbu import create_table

create_table(
    "amzn_gold",
    "na_net_sales",
    df
)
```

Proprietary libraries to standardize and simplify spark code

Preset cluster options and workflow orchestration in Databricks



Data delivery systems to export to multiple clouds



Interactive notebooks and visualizations of Lakehouse data

Manage data infrastructure and ingestion behind the scenes

Decentralized platform was key to being a trusted source of insights for clients

What happened next?

We grew fast ..

Alt data market leader with more teams, datasets, and product types

50+
Data Team

4 PB
Lakehouse Data

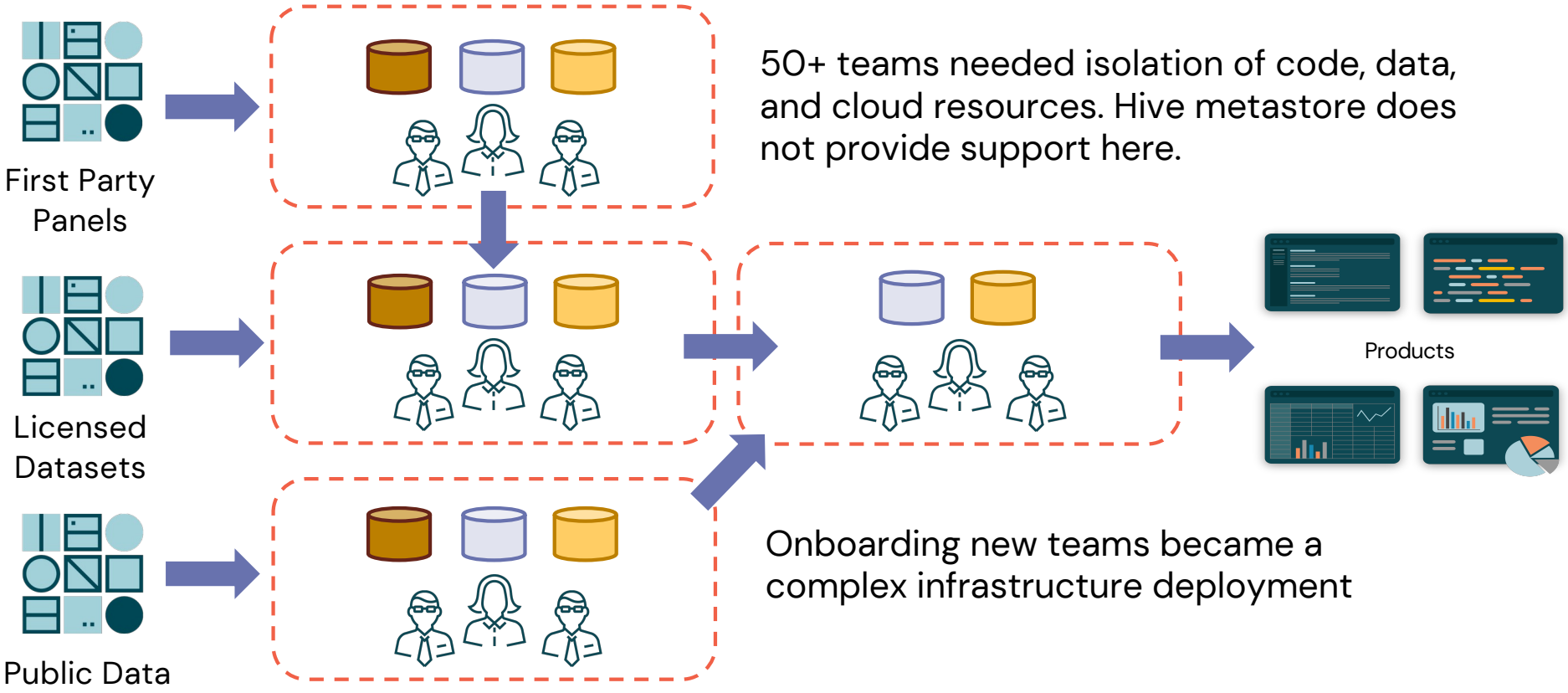
Research
Reports

290+
Data Team

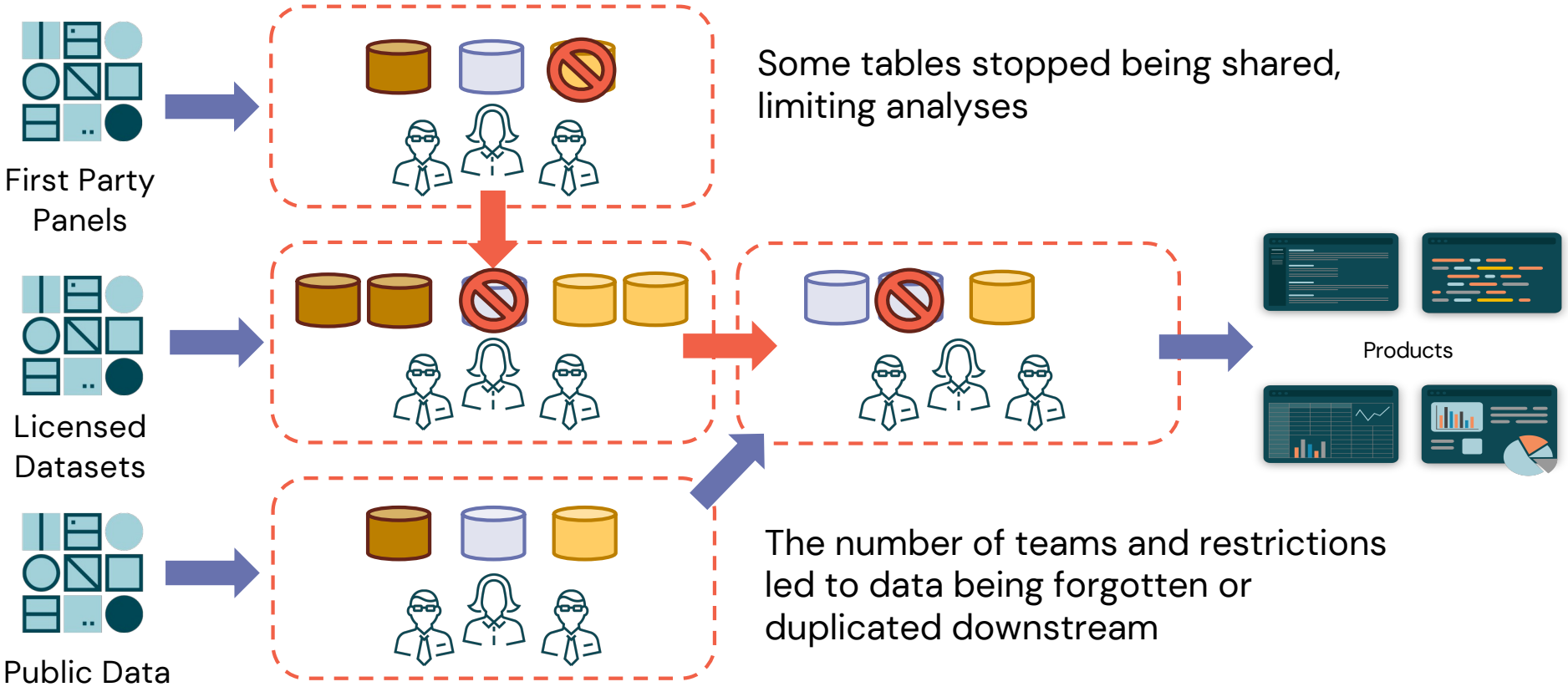
16 PB
Lakehouse Data

Research Reports
Data Feeds
KPI Dashboards
Custom Solutions

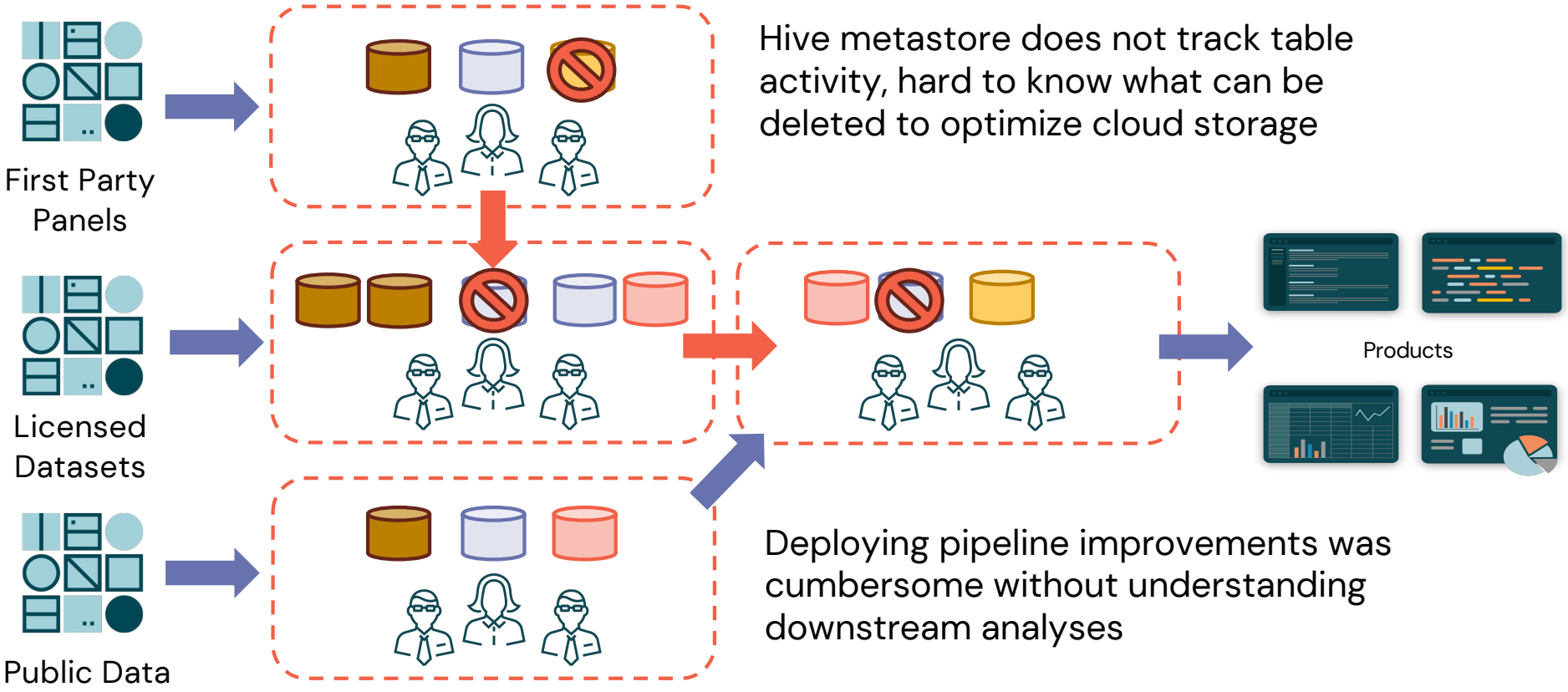
Data Access Became Complex



Data Isolation Hindered Collaboration



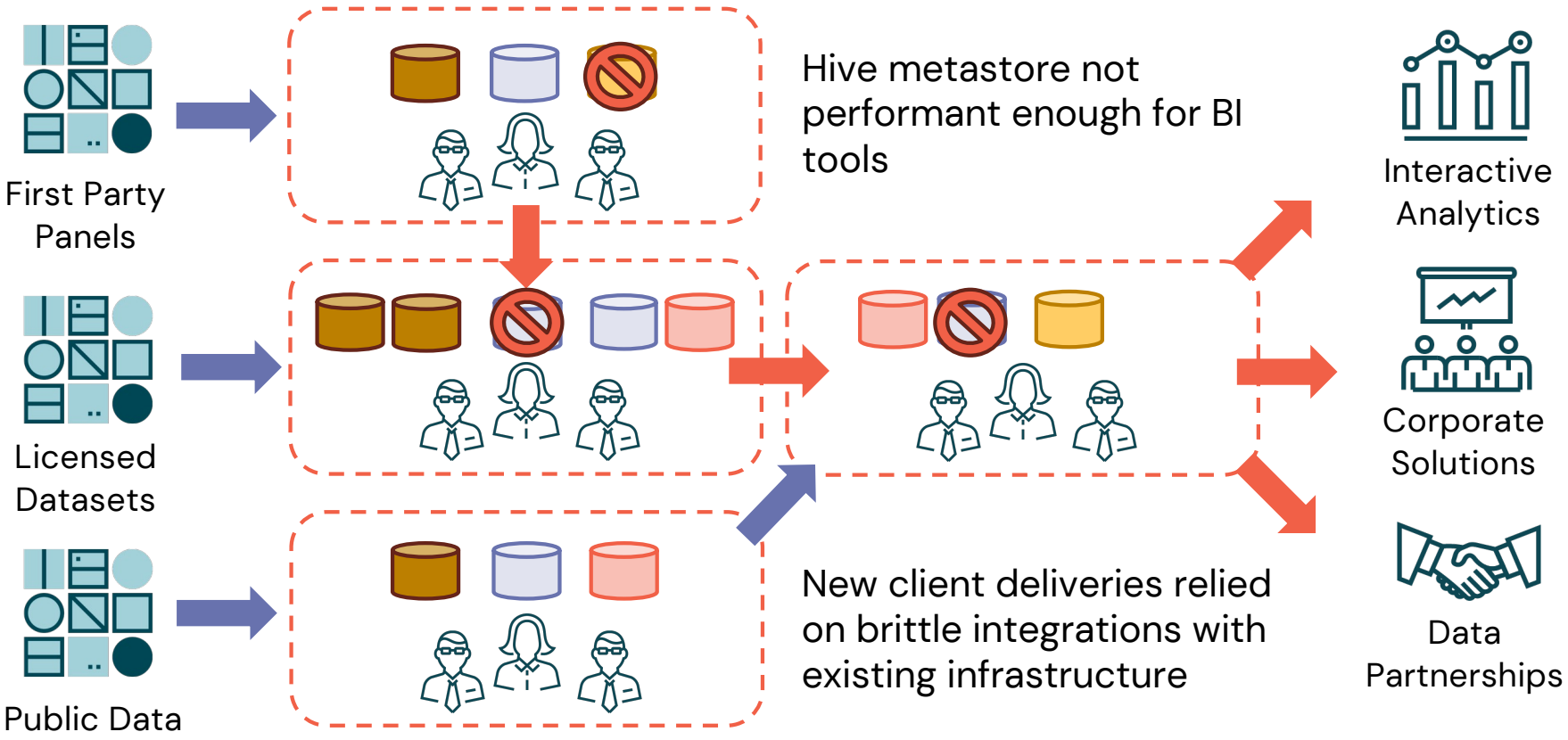
Cost Overhead of Lakehouse



Hive metastore does not track table activity, hard to know what can be deleted to optimize cloud storage

Deploying pipeline improvements was cumbersome without understanding downstream analyses

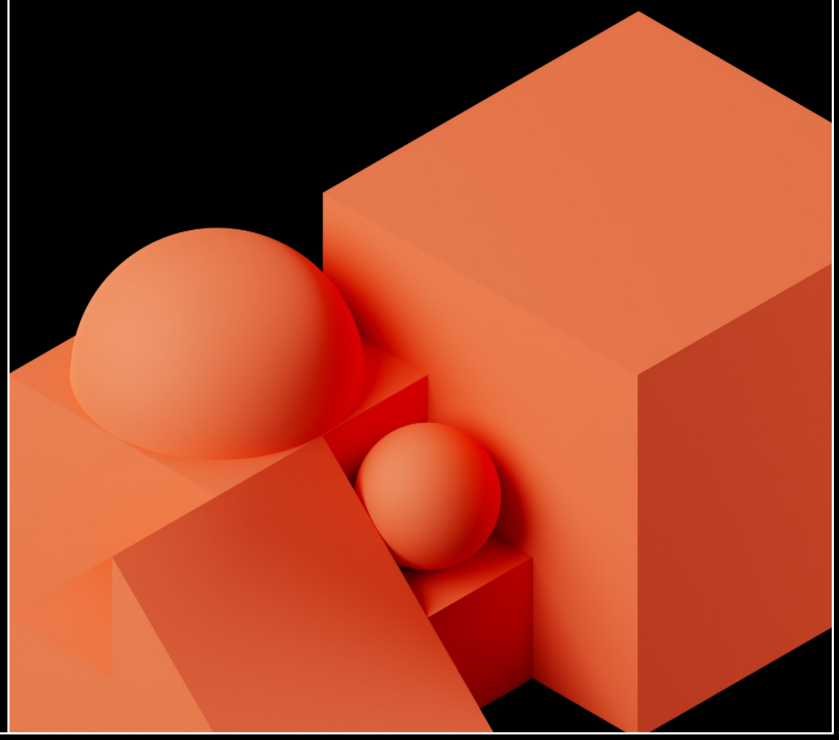
Product Opportunities Blocked



Data governance constraints turned into business challenges.

We needed a new approach to manage data.

Enter Unity Catalog



Unity Catalog Core Features

Data Organization via 3 Level Namespace

```
SELECT
  *
FROM
  prod.revenue.skus
```

Granular permissions through ACL Statements

```
GRANT MODIFY ON
prod.revenue.skus
TO `PG_FINANCE`

GRANT SELECT ON
prod.revenue.skus
TO `PG_PRODUCT_OPS`
```

Lakehouse Visibility through Data Explorer

- Data objects can be searched in the UI
- Audit logs to trace lineage and activity of data objects
- REST API to access objects and update permissions

Setting Up Unity Catalog

Enable Unity Catalog for Databricks Account

- Organize groups at the account level
- Create a Unity Catalog Metastore

Configure Cloud Infrastructure

- Deploy a Metastore Role
- Configure cloud storage access to metastore role

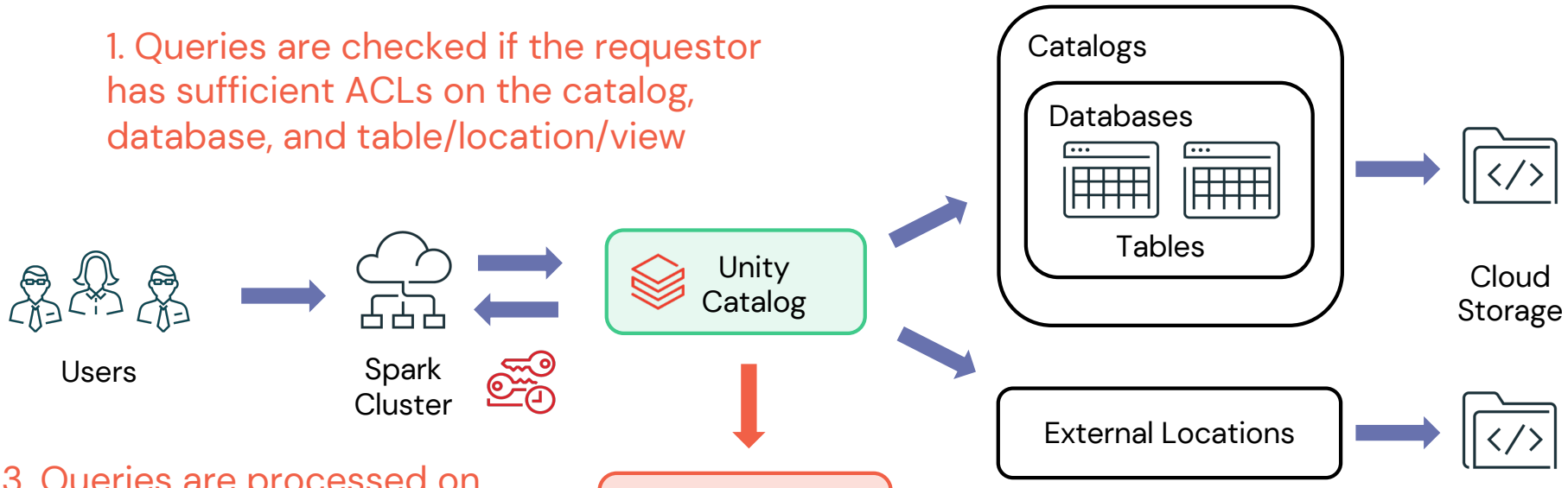
Update Spark Environment

- Upgrade to Databricks Runtime 11.3+
- Use single-user or shared cluster modes

Unity Catalog Architecture

Inspects queries to verify data access and generates pre-signed URLs

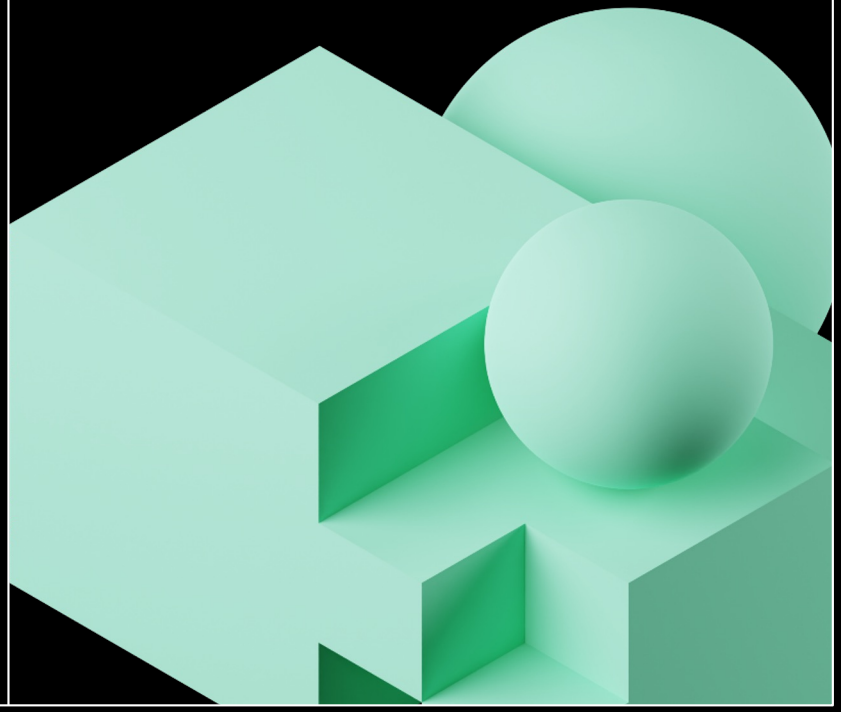
1. Queries are checked if the requestor has sufficient ACLs on the catalog, database, and table/location/view



3. Queries are processed on clusters that fetch data using pre-signed URLs served from Unity Catalog

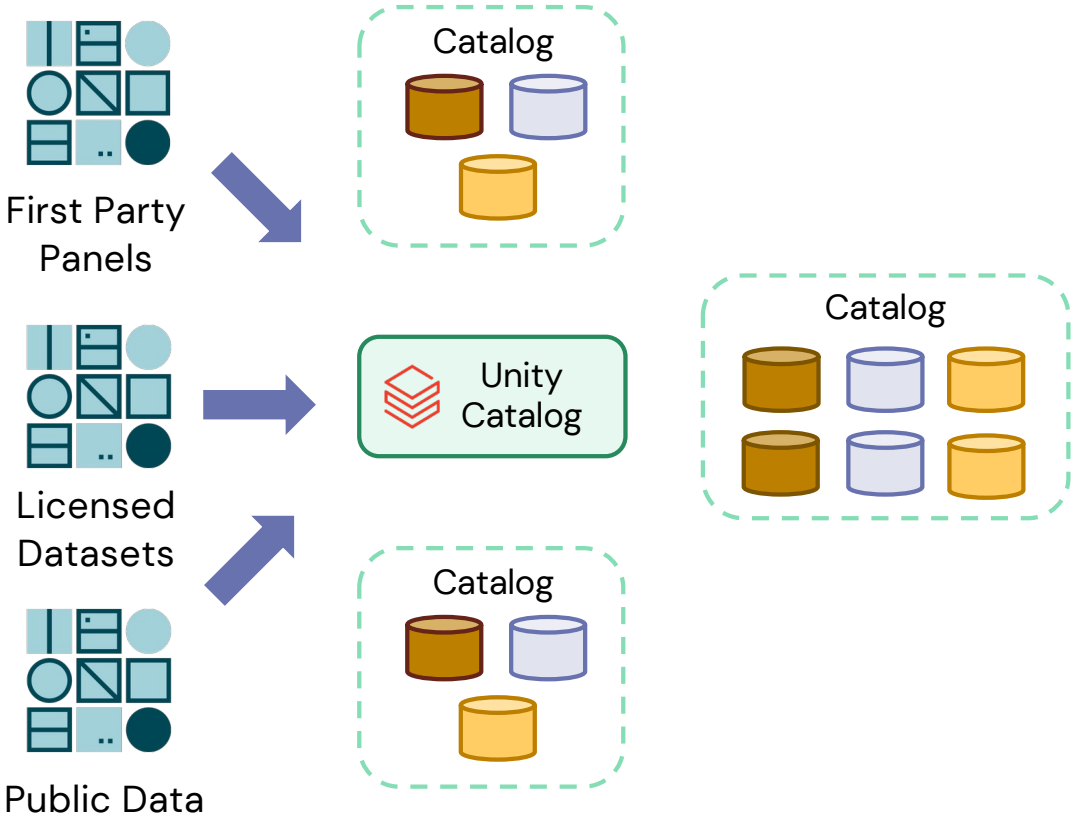
2. Cloud storage access is managed through Unity Catalog via an assigned metastore role

Unity Catalog Impact



Data access is simple and universal

Unity Catalog Centralized Access Controls



- Source data is registered as External Locations and ingested as bronze tables
- Catalogs segment 210K+ tables by business units

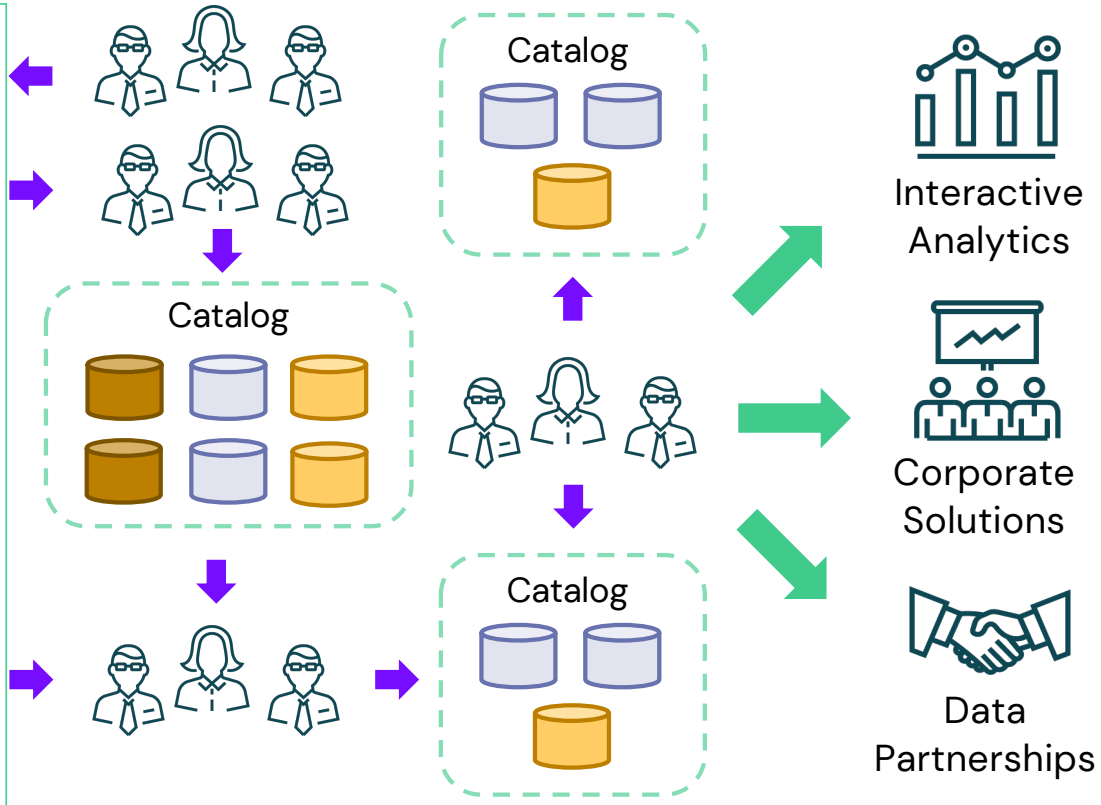
```
CREATE CATALOG yd_prod;

CREATE CATALOG yd_ereceipt;

CREATE CATALOG yd_corporate;
```

Unity Catalog Centralized Access Controls

- Organize teams around the data rather than the other way around.
- Use ACLs to grant groups precise access to data.

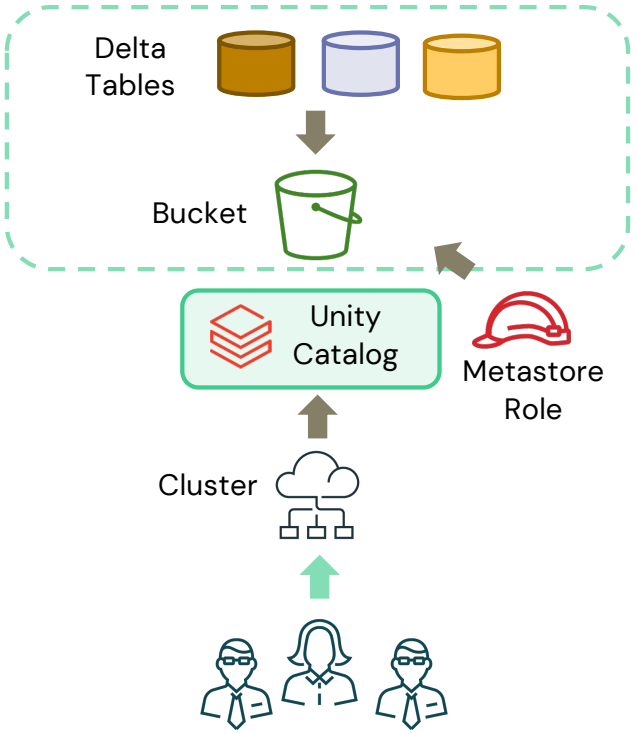
```
GRANT USE SCHEMA, SELECT ON
SCHEMA yd_prod.ereceipts_gold
TO `PG_PRODUCT`;
```

- Corporate Solutions and Partnerships teams rely on isolated catalogs to conduct daily business



Cloud Infrastructure Simplified

Deprecated 70% of cloud resources that managed permissions before

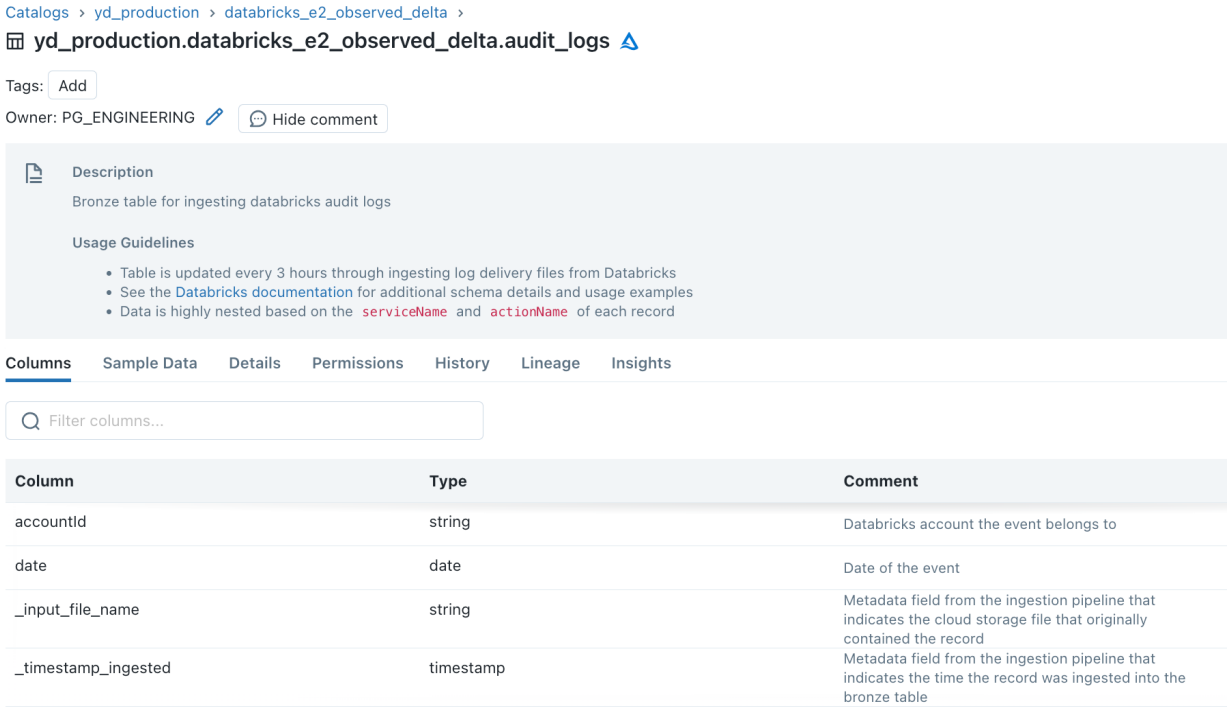


- Unity Catalog connects to clouds using a single "metastore role"
- Data access to teams is granted by Unity Catalog ACLs, regardless of cluster used
- Temporary credentials API can access native cloud storage APIs

Driving collaboration between teams

Drive data adoption with documentation

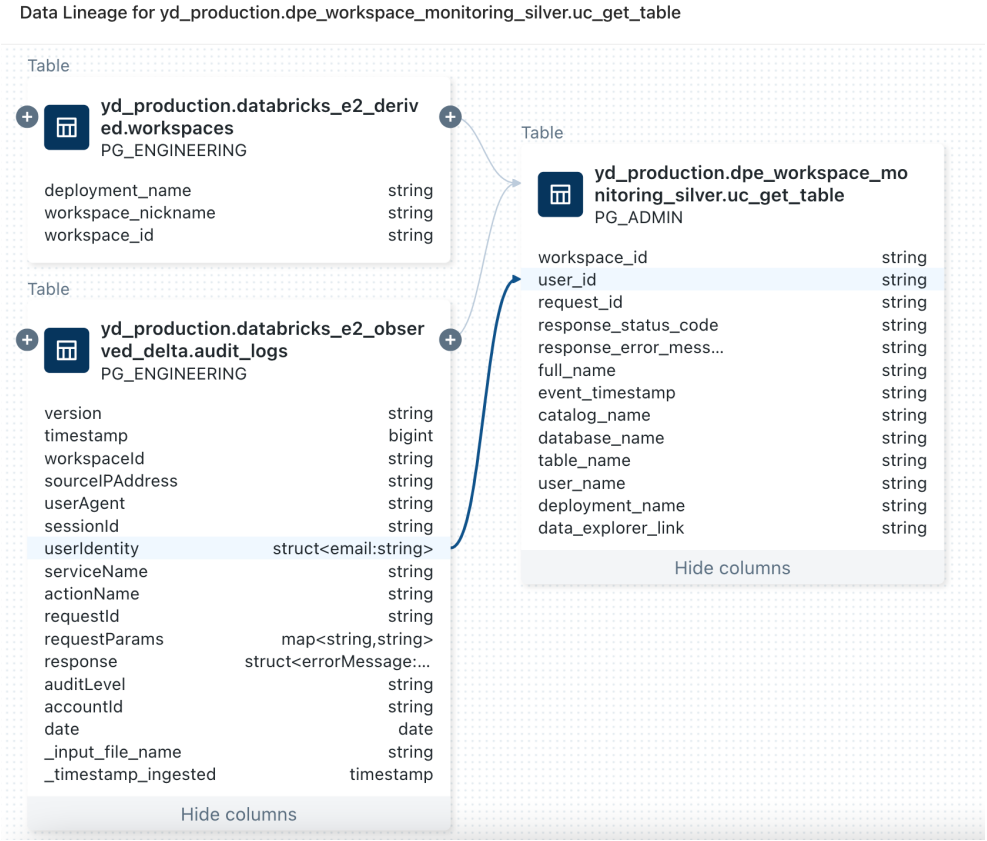
Data Explorer displays comments for catalogs, databases, and tables



- Teams add comments to data objects in the Data Explorer or via Spark
- Documented tables are self-explanatory, more likely to be used
- Use markdown to link to internal documentation
- History tab summarizes table updates (delta only)

Reduce time to release data changes

Lineage highlights unused tables and columns to de-risk deployments



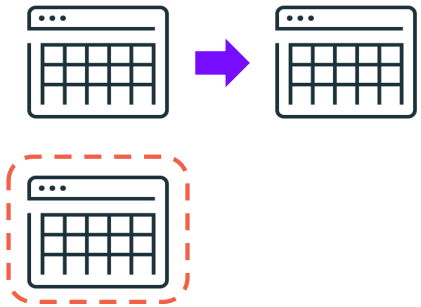
- Lineage data is used to prevent bad releases of data pipelines
- Tables are analyzed to see which fields are used and how often
- Databricks workflows are integrated with lineage to map pipelines to table creation
- Lineage data can be fetched via the REST API for automation and integration

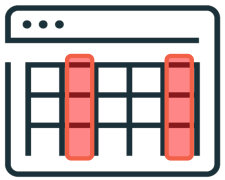
Optimizing Lakehouse operating costs

Pruning Tables Using Lineage

Lineage data revealed 27% of data in storage was inactive

- Analyzing lineage data at scale surfaced many unused tables:
  - Tables without `getTable` events in audit logs
  - Storage costs estimated by joining to inventory reports from cloud provider

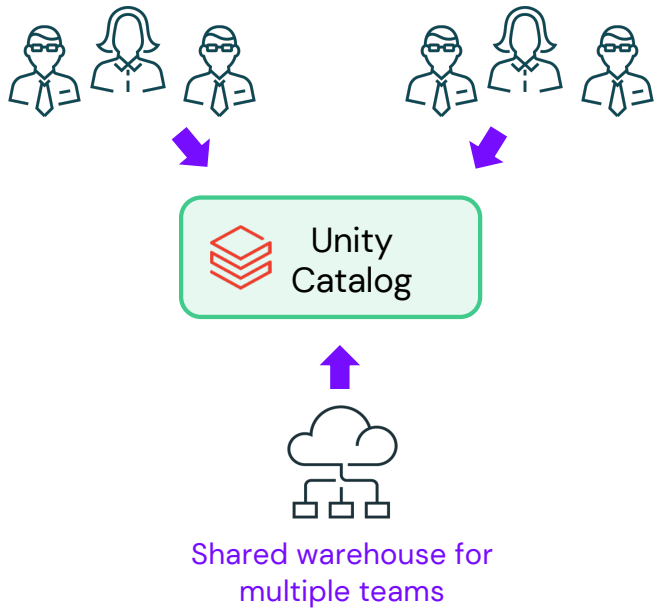




- Column-level lineage examined on large tables:
  - Unused columns are removed from pipelines
  - Used columns are reordered for better query performance

Reducing SQL Warehouse Bill by 45%

Permissions model allows for fewer, multi-team SQL warehouses



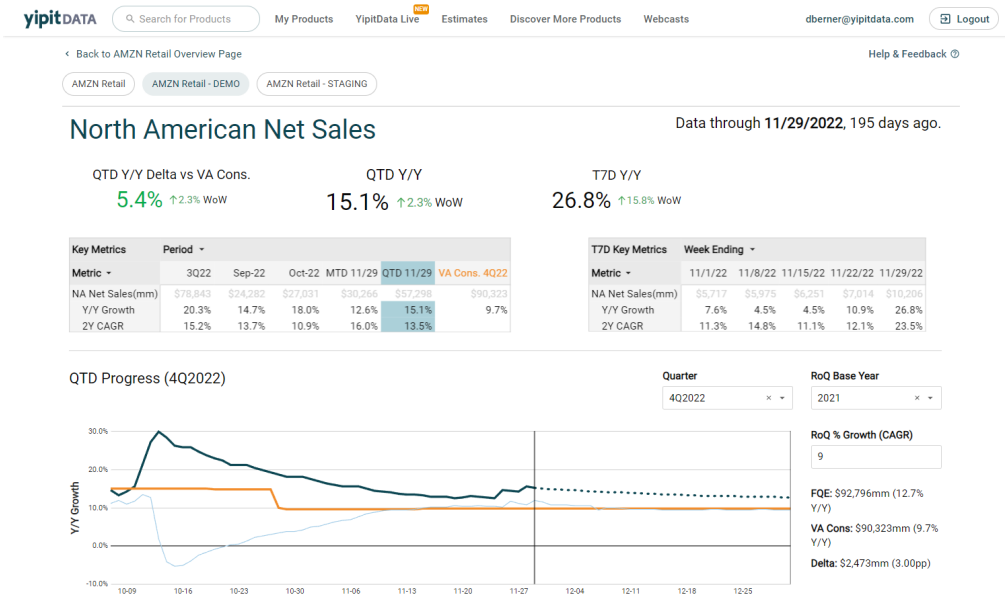
- Unity Catalog separates data permissions from compute layer, allowing for shared compute
- Consolidation had a secondary benefit of better autoscaling, further optimizing spend

Enabling new product opportunities

Opening Up Interactive Analytics

Ms-level latency is critical for rich data experiences

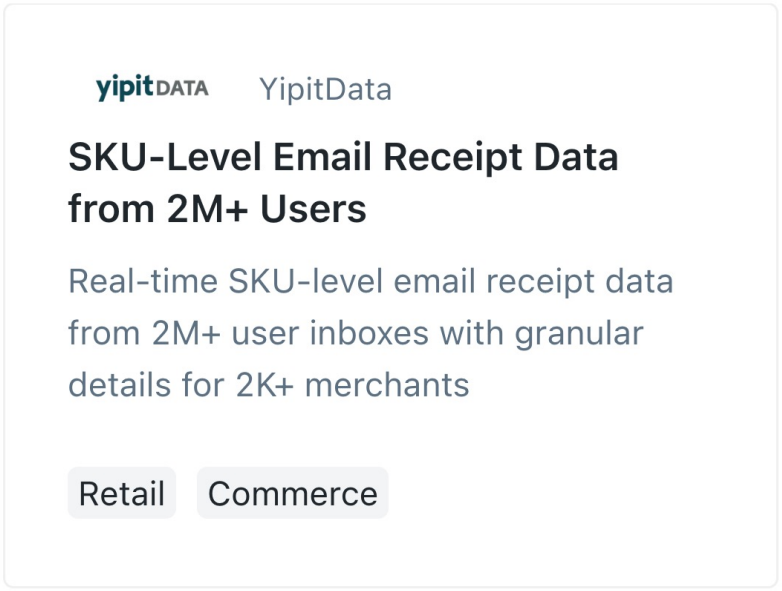
- BI tools require fast metadata operations (ex: `SHOW TABLES`)
- Hive metastore could not scan metadata for 210K tables. Unity Catalog had no issues.
- The performance boost was key to opening up new client experiences



YipitData on the Databricks Marketplace

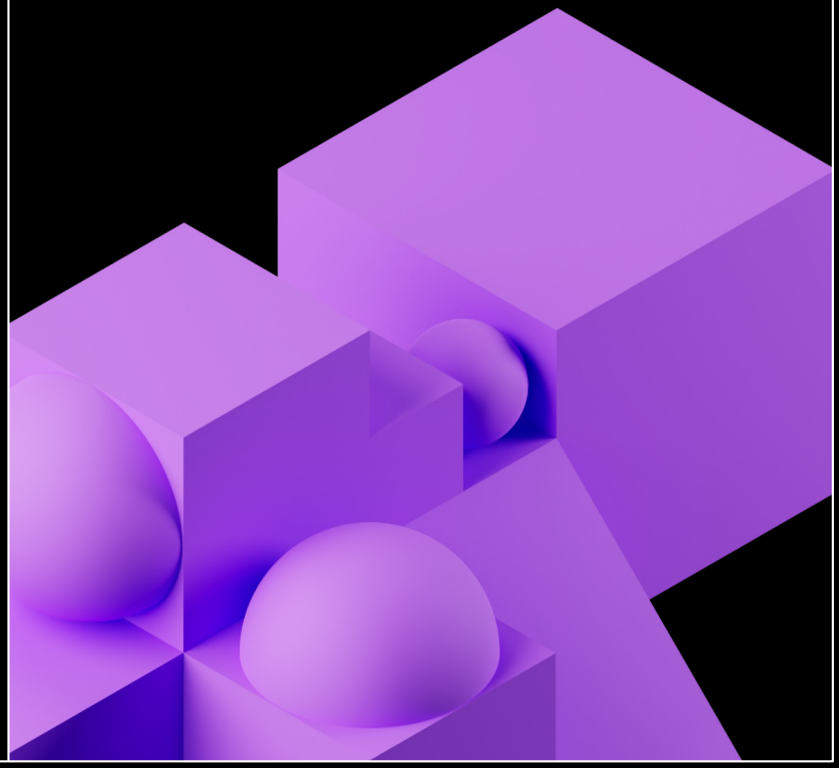
Now connecting to clients via Unity Catalog and Delta Sharing

More listings coming soon!



- Gold tables are packaged as Delta Shares, appear as read-only "catalogs" for clients
- Listed on Databricks marketplace for fast discovery and integration by clients
- Tables shared with no extra infrastructure or compute via Delta Sharing protocol

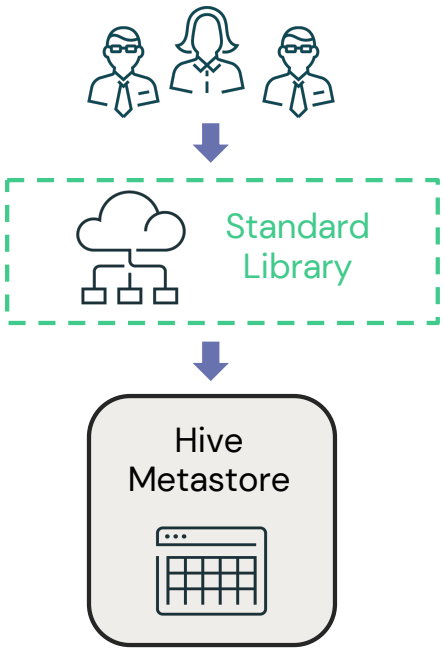
Migrating from Hive to Unity Catalog



Needed an in-place switch to Unity Catalog with no downtime

Migration Strategy

Populate Unity Catalog in the background through standard libraries

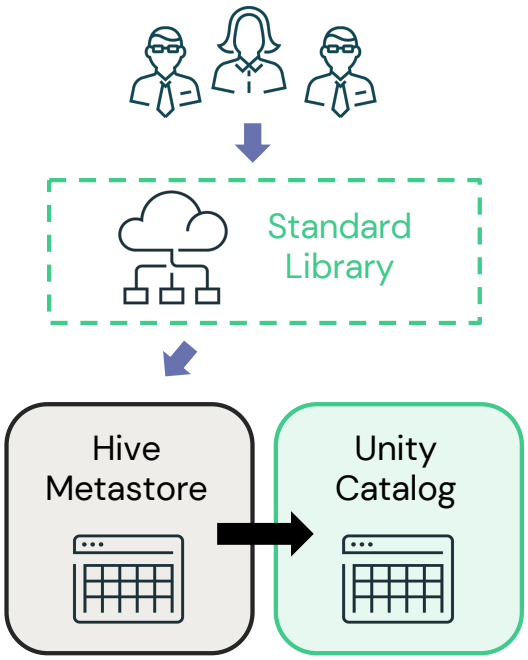


- Teams used spark to create external tables in Hive metastore
- Executed via a standard library function, `create_table`

```
def create_table(database: str, table: str, df: DataFrame):
    (
        df.write.format("delta")
        .mode("overwrite")
        .saveAsTable(f"{database}.{table}")
    )
```

Migration Strategy – Phase 1

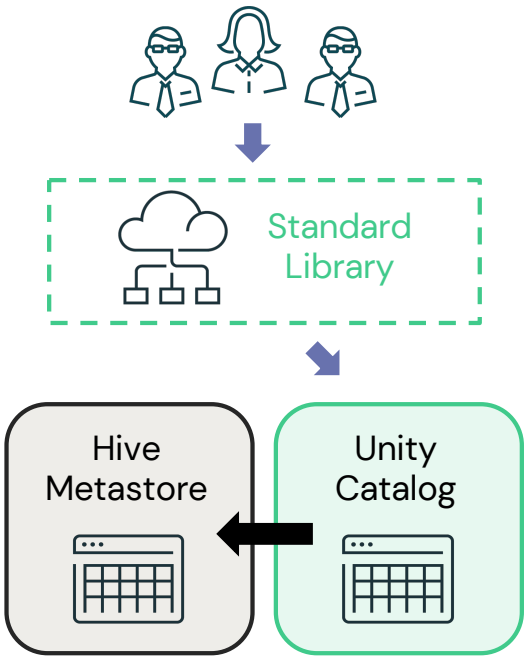
Replicate Hive tables in Unity Catalog using the SYNC operation




```
def create_table(database: str, table: str, df: DataFrame):
    (
        df.write.format("delta")
        .mode("overwrite")
        .saveAsTable(f"hive_metastore.{database}.{table}")
    )

    sql(f"""
    SYNC TABLE prod.{database}.{table}
    FROM hive_metastore.{database}.{table}
    SET OWNER TO `{TEAM}`
    """)
```

Migration Strategy – Phase 2

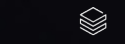
Write directly to Unity Catalog and replicate in Hive as a backup



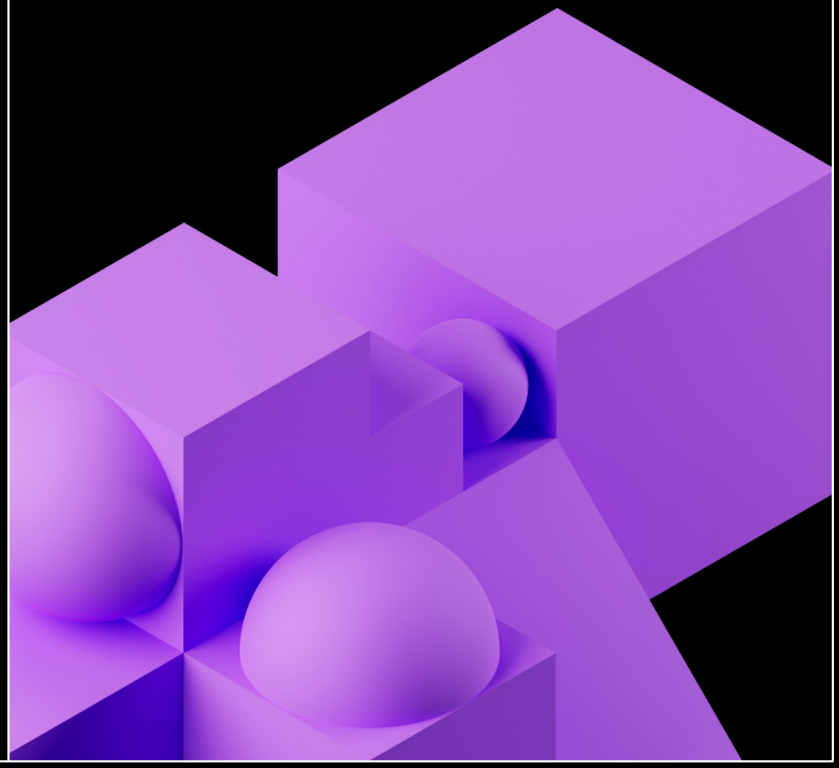

```
def create_table(database: str, table: str, df: DataFrame):
    (
        df.write.format("delta").mode("overwrite")
        .saveAsTable(f"prod.{database}.{table}")
    )

    location = sql(f"DESCRIBE DETAIL prod.{database}.{table}").first().location

    sql(f"""
    CREATE OR REPLACE TABLE hive_metastore.{database}.{table}
    LOCATION '{location}'
    """)
```

Teams switched to Unity Catalog with minimal code changes

Pipelines could "fall back" to hive metastore in case of failures



Conclusion



Lakehouse Growing Pains Resolved

- Data access is complex to maintain with thousands of tables
  - 3 level namespace to organize data and grant precise access to groups
- Data isolation hinders collaboration across teams
  - Browsable data explorer allows for data assets to be found and documented
- Cost overhead of growing, petabyte-scale Lakehouse
  - Lineage is analyzed to eliminate unused data and compute is consolidated
- Externalizing data for new products is difficult
  - Marketplace and interactive analytics are new channels to engage clients

Unity Catalog is more than a governance tool.

Use it to activate your Lakehouse and create value.

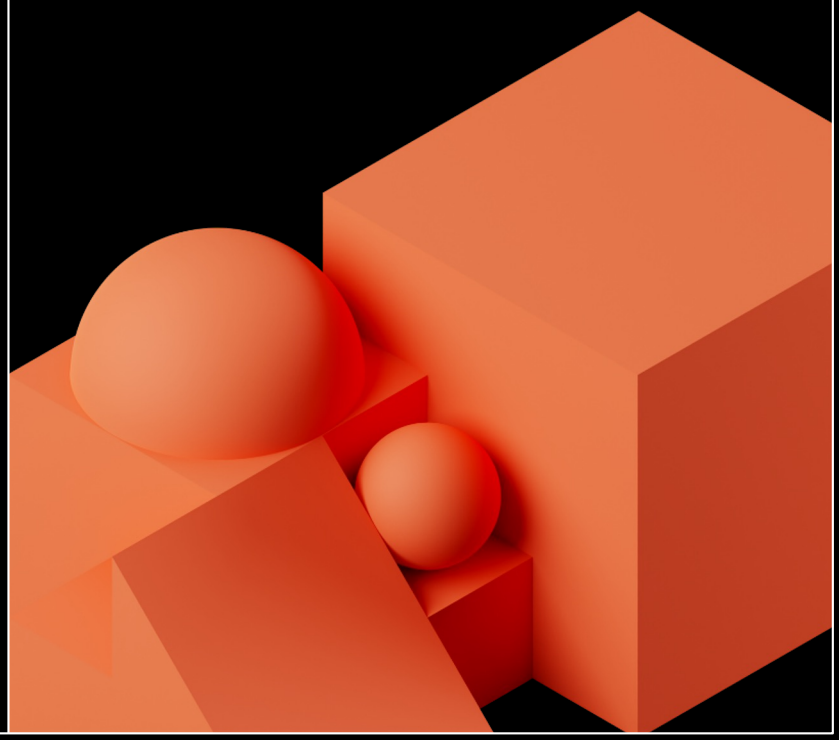

Our Lakehouse is starting
to look like this.
Thanks to Unity Catalog
Photo by Lewis J Goetz

Lakehouse Growing Pains Resolved

- Data access is complex to maintain with thousands of tables
  - 3 level namespace to organize data and grant precise access to groups
- Data isolation hinders collaboration across teams
  - Browsable data explorer allows for data assets to be found and documented
- Cost overhead of growing, petabyte-scale Lakehouse
  - Lineage is analyzed to eliminate unused data and compute is consolidated
- Externalizing data for new products is difficult
  - Marketplace and interactive analytics are new channels to engage clients

APPENDIX

Unity Catalog Tactics

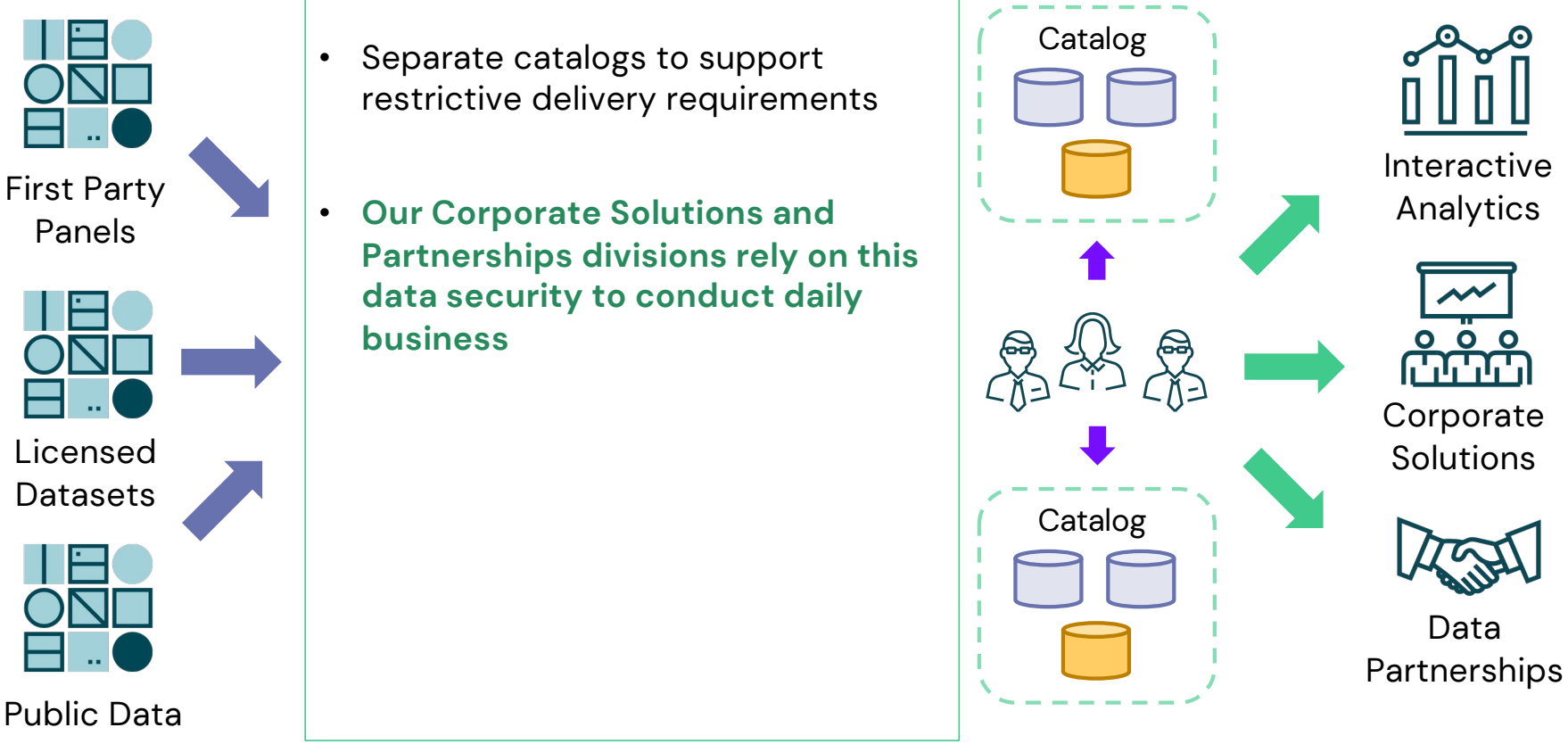


Leverage 3-Level Namespace

Lean on catalogs and databases to categorize data

- Have consistent in table conventions across teams
  - Catalogs serve as the broadest umbrella to organize data
- Permission at catalog or database level to streamline administration
  - Permissions inherited to child tables and views
  - Bronze tables should have the most restrictive permissions
- Grant ACLs and ownership to groups instead of users
  - Simplifies collaboration and team changes

Unity Catalog Centralized Access Controls



Grant Permissions to Groups

Well defined user groups in Databricks streamline access management

- Groups in Databricks should resemble business units
- Define groups at the Databricks account level, not the workspace
- Grant ownership and ACLs to groups. This simplifies collaboration and team changes

```
-- DON'T DO THIS

GRANT SELECT ON SCHEMA prod.sales
TO `user@acme.com`

-- DO THIS

GRANT SELECT ON SCHEMA prod.sales
TO `FINANCE`

ALTER SCHEMA prod.sales OWNER TO
`REVENUE`
```

Utilize REST API for Automation

REST API endpoints available for all Unity Catalog resources

- REST API can fetch and modify metadata of data objects without spark compute
- Performing bulk updates of ACLs is faster with the REST API
- Temporary credentials endpoint can access native cloud storage APIs

```python
import requests
import boto3

response = requests.post(
    endpoint="2.1/unity-catalog/temporary-path-credentials",
    params={
        "url": "s3://<bucket>/<path>",
        "operation": "PATH_READ"
    },
)

output = response.json()
session = boto3.session.Session(
    output["aws_temp_credentials"]["access_key_id"],
    output["aws_temp_credentials"]["secret_access_key"],
    output["aws_temp_credentials"]["session_token"],
)

client = session.client("s3")
```

Cloud Provider Recommendations

Unity Catalog should do the heavy lifting for data access

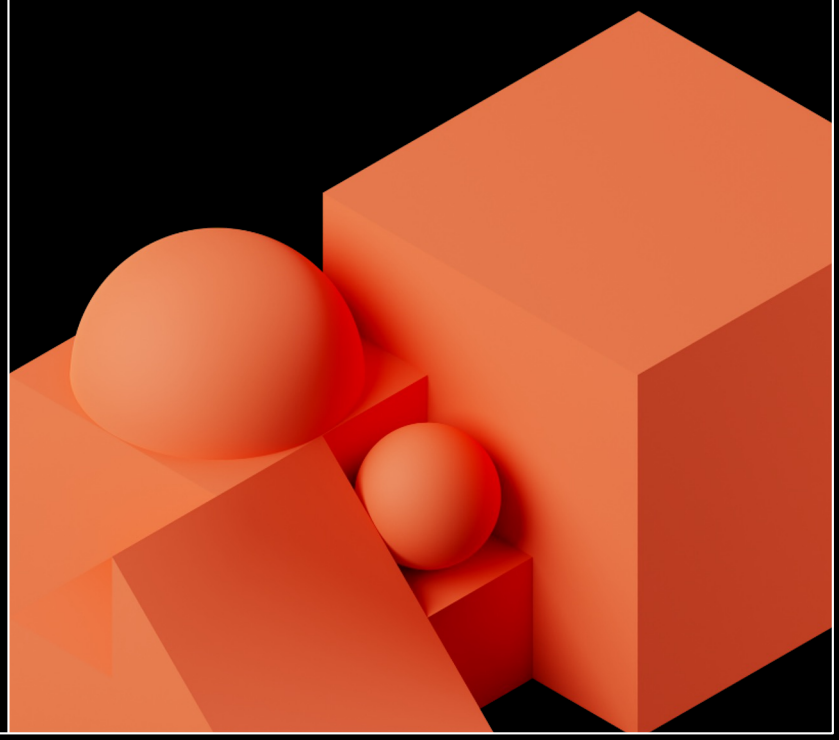
- Minimize the number of storage credentials deployed. Define granular data access via ACLs.
- Because Unity Catalog centralizes data access, it is easier to support multi-account cloud deployments
- Avoid setting data permissions at cluster-level, opportunity to reduce compute footprint

Be Strategic with the Migration

Spend the time upfront to re-organize the lakehouse

- Design catalog and group structure to satisfy business requirements
- First replicate cloud access with external locations and storage credentials
- Move new teams to Unity Catalog, then tackle existing teams
- Use the `hive_metastore` catalog for backwards compatibility and `SYNC` table definitions to Unity Catalog
  - Use `spark.databricks.sql.initial.catalog.name` to set default catalog

Lineage Analysis



Reducing Storage Costs through Lineage

Lineage data revealed 27% of data in storage was inactive

```
WITH read_activity_from_audit_logs AS (
  SELECT
    SPLIT(requestParams.full_name_arg, r'\.')[0] AS catalog_name,
    SPLIT(requestParams.full_name_arg, r'\.')[1] AS schema_name,
    SPLIT(requestParams.full_name_arg, r'\.')[2] AS table_name,
    MIN(FROM_UNIXTIME(timestamp / 1000)) AS first_read,
    MAX(FROM_UNIXTIME(timestamp / 1000)) AS last_read

  FROM
    audit_logs

  WHERE
    serviceName = 'unityCatalog'
    AND actionName = 'getTable'

  GROUP BY
    1, 2, 3
),
```

- Table fetches logged in audit logs as `getTable` events
- Aggregate request parameters in logged events to see activity over time by table

Reducing Storage Costs through Lineage

Lineage data revealed 27% of data in storage was inactive

```
all_tables_in_unity_catalog AS (
  SELECT
    table_catalog AS catalog_name,
    table_schema AS schema_name,
    table_name
  FROM
    system.information_schema.tables
)
```

- System information tables contain details of the entire metastore of Unity Catalog
- Use as a current view of all tables available in the Lakehouse

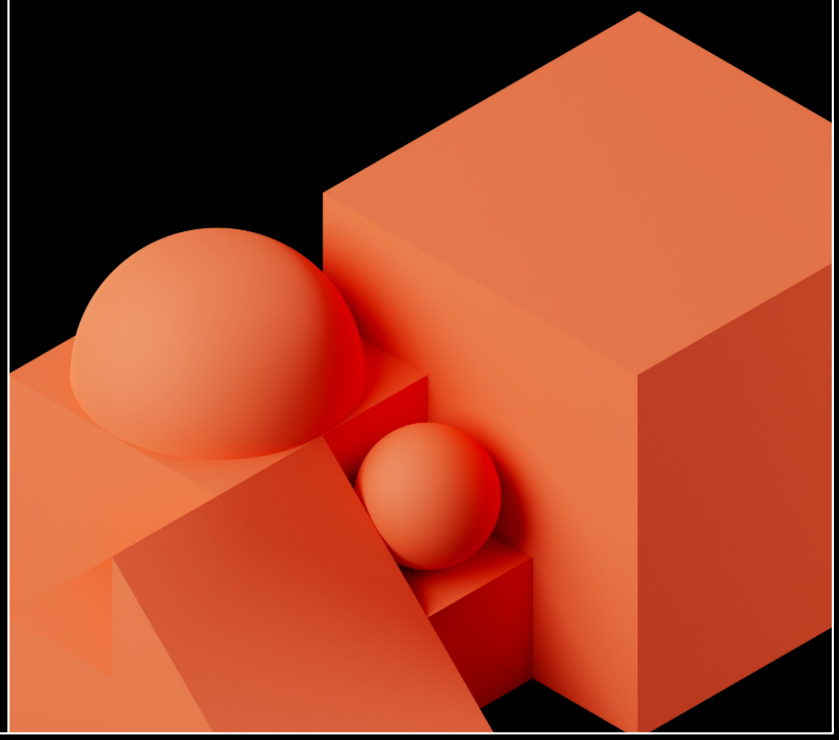
Reducing Storage Costs through Lineage

Lineage data revealed 27% of data in storage was inactive

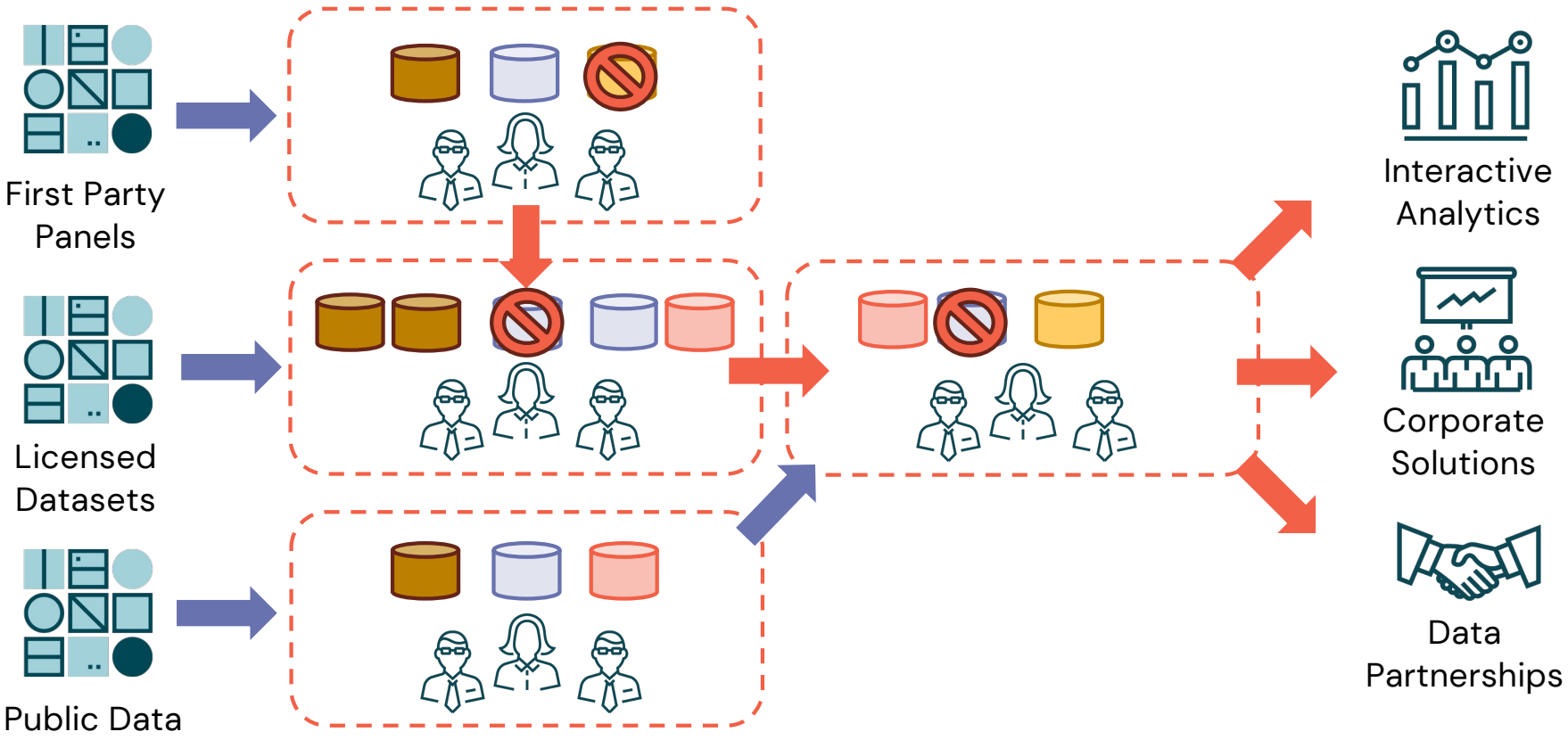
```
SELECT
  catalog_name,
  schema_name,
  table_name
FROM
  all_tables_in_unity_catalog
LEFT ANTI JOIN
  read_activity_from_audit_logs
  USING (catalog_name, schema_name, table_name)
```

- Join current catalog against audit log activity to see unused tables
- Refresh analysis over time to continue to clean up Lakehouse and reduce storage overhead

HMS Architecture

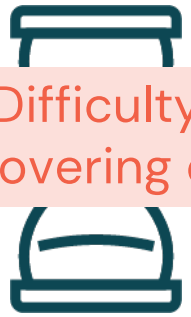


Data Access Was Complex



Lakehouse Growing Pains

Challenges as data volume and organizational complexity creeps in



Difficulty discovering data

Fast Time to Insight

Data dri[illegible]decisions are the norm

Siloed data teams

Integrates with Broader Stack

Establish[illegible]ce of truth across [illegible]nization

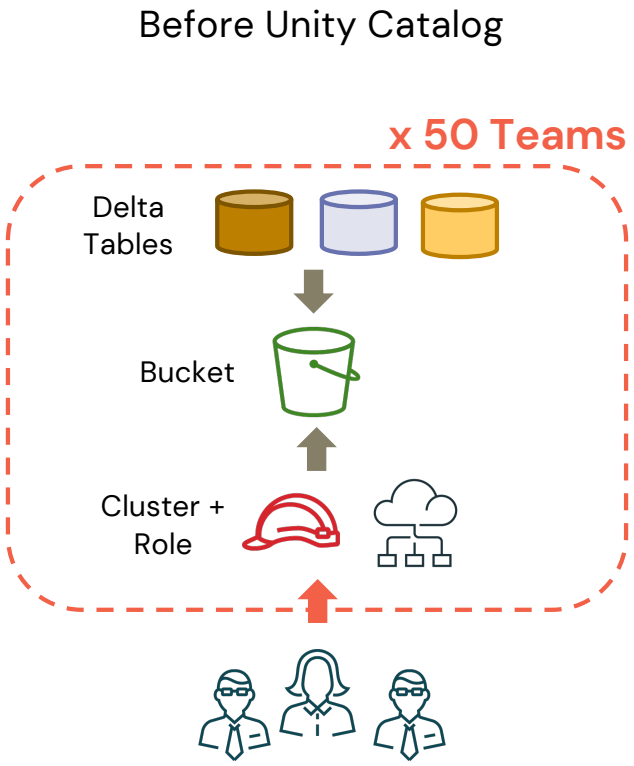
Permission gaps or bottlenecks

Effective Data Governance

Da[illegible]ets deploye[illegible]tegically

Cloud Infrastructure Simplified

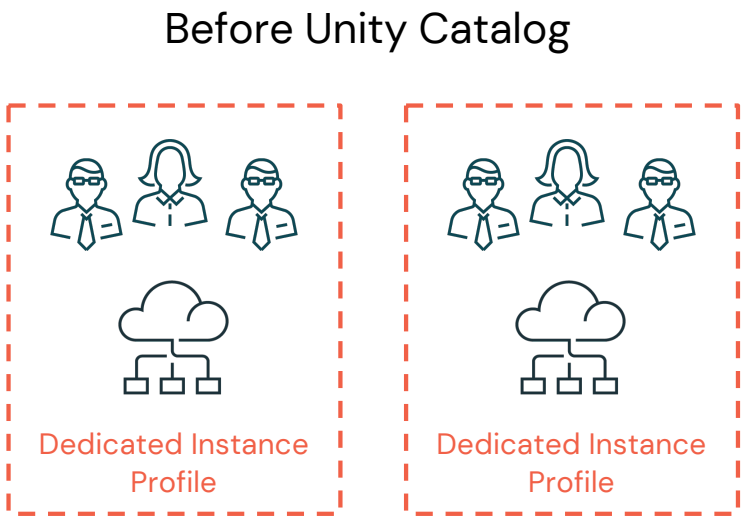
Unity Catalog removes complexity in securing data in the cloud



- Data isolation was implemented by attaching dedicated team instance profiles to clusters
- Each team had their own bucket, instance profile, and cluster to process data
- "Role bloat" was a compounding problem
- Onboarding new teams was complex and slow

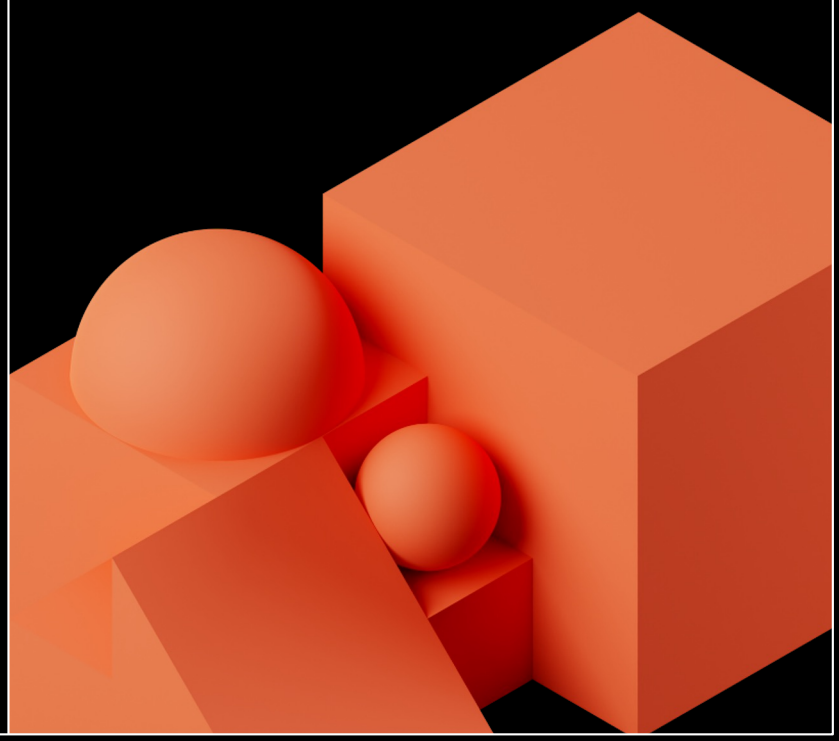
Reducing SQL Warehouse Bill by 45%

Permissions model allows for fewer, multi-team SQL warehouses



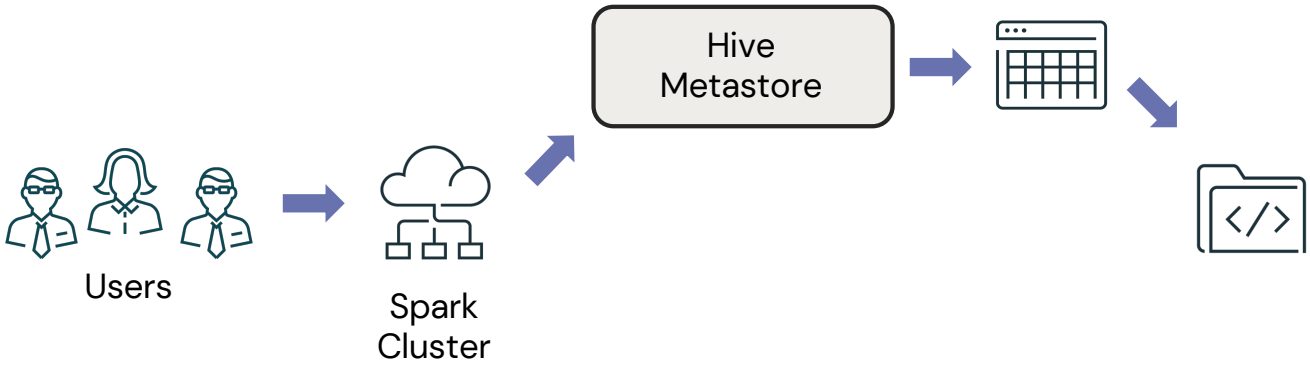
- Each team needed their own warehouse with a dedicated instance profile to support data access
- Warehouses not fully utilized, excess costs for always-on compute

Migration Details



Migration Strategy

In-place switch of Unity Catalog from Hive with zero downtime for teams



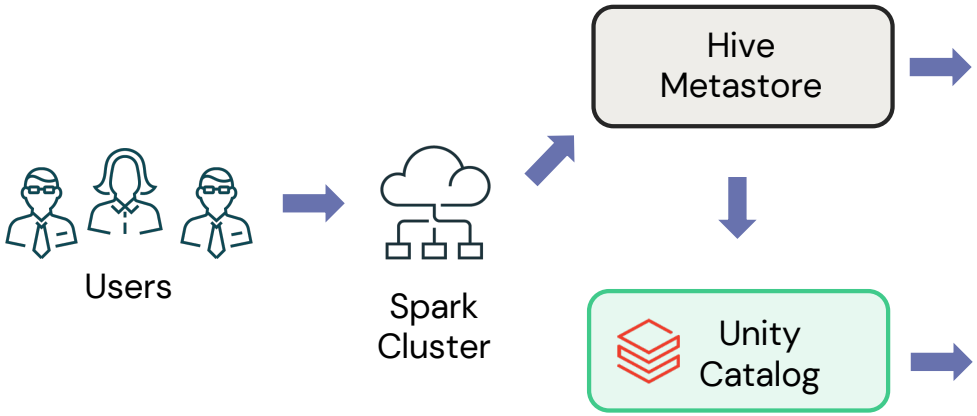
Teams used spark to create external tables through Hive metastore

Executed via a standard library function, `create_table`

```
def create_table(database: str, table: str, df: DataFrame):
    (
        df.write.format("delta")
        .mode("overwrite")
        .saveAsTable(f"{database}.{table}")
    )
```

Migration Strategy

In-place switch of Unity Catalog from Hive with zero downtime for teams



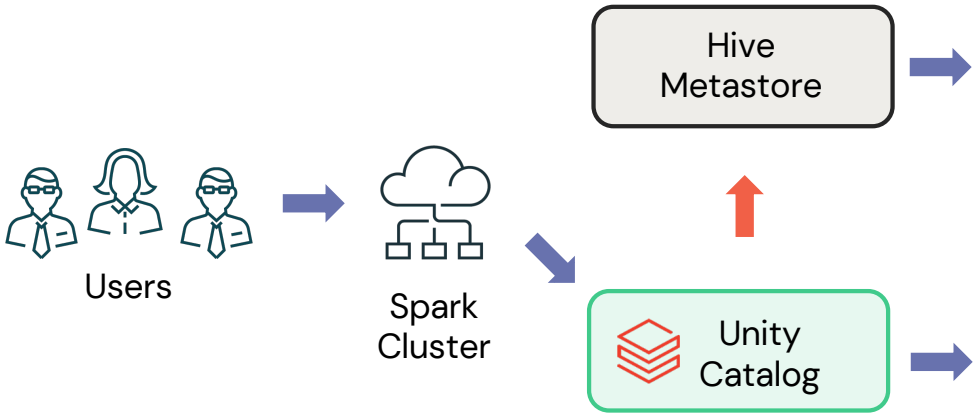

```
def create_table(database: str, table: str, df: DataFrame):
    (
        df.write.format("delta")
        .mode("overwrite")
        .saveAsTable(f"hive_metastore.{database}.{table}")
    )


    spark.sql(f"""
    SYNC TABLE prod.{database}.{table}
    FROM hive_metastore.{database}.{table}
    SET OWNER TO `{TEAM}`
    """)
```

Phase 1 - Table creations processed in Hive and replicated in Unity Catalog, using the SYNC command

External tables point to the same cloud storage files (i.e. no data duplication)

Migration Strategy

In-place switch of Unity Catalog from Hive with zero downtime for teams



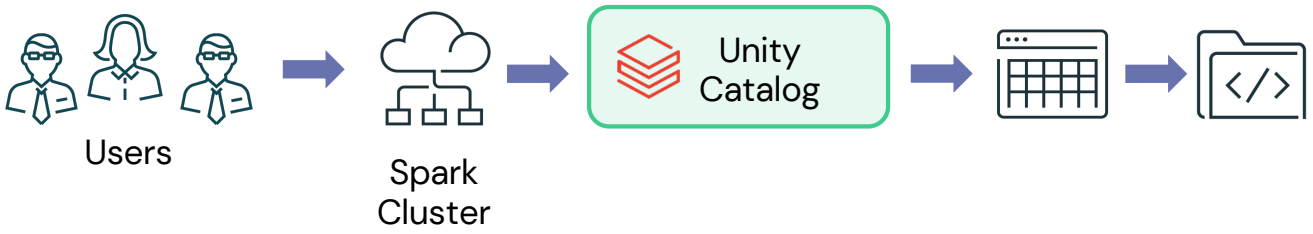

```
def create_table(database: str, table: str, df: DataFrame):
    (
        df.write.format("delta").mode("overwrite")
        .saveAsTable(f"prod.{database}.{table}")
    )

    location = spark.sql(
        f"DESCRIBE DETAIL prod.{database}.{table}"
    ).first().location

    spark.sql(f"""
    CREATE OR REPLACE TABLE hive_metastore.{database}.{table}
    LOCATION '{location}'
    """)
```

Phase 2 - Table creations fulfilled in Unity Catalog, and table definitions copied to Hive to stay consistent

In case of errors, automatically falls back to Phase 1 implementation to avoid downtime

Migration Strategy

In-place switch of Unity Catalog from Hive with zero downtime for teams



After Phase 2, removed the interaction with Hive metastore and migration was complete.

Platform users had to make minimal code changes on their end, as we migrated almost entirely behind the scenes