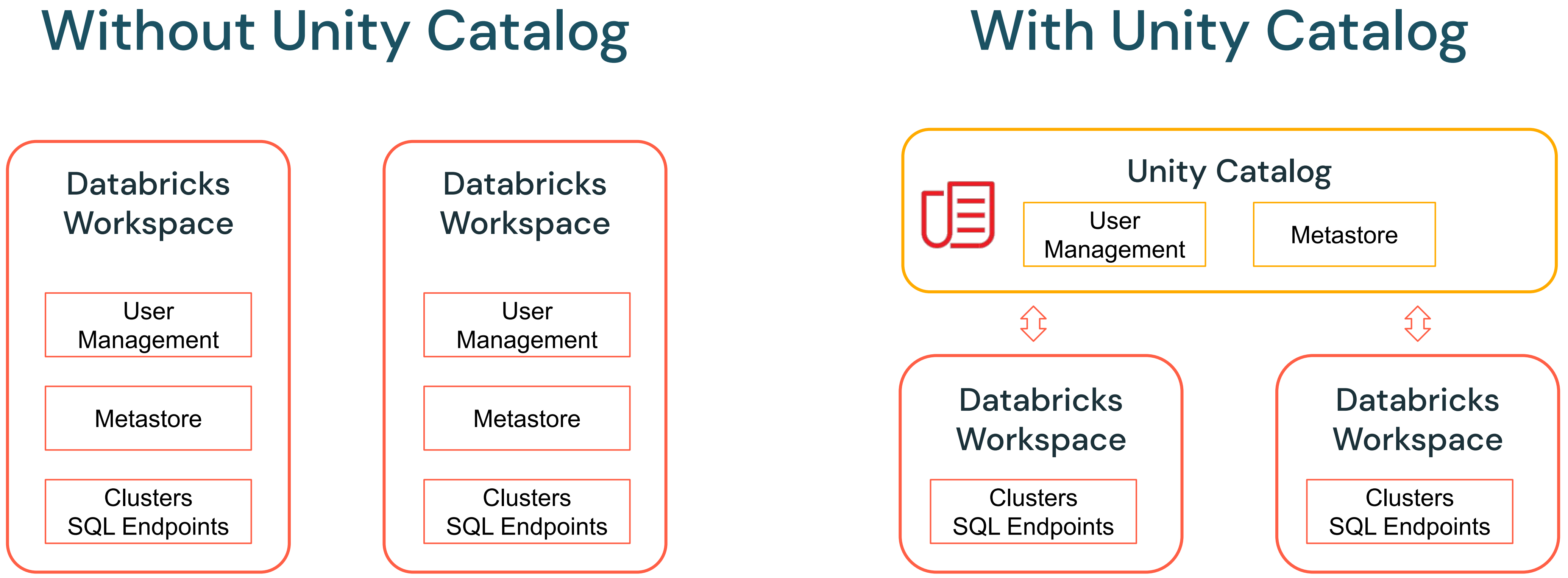
Unity Catalog

Fine-grained governance
for data and AI

Technical Deep Dive

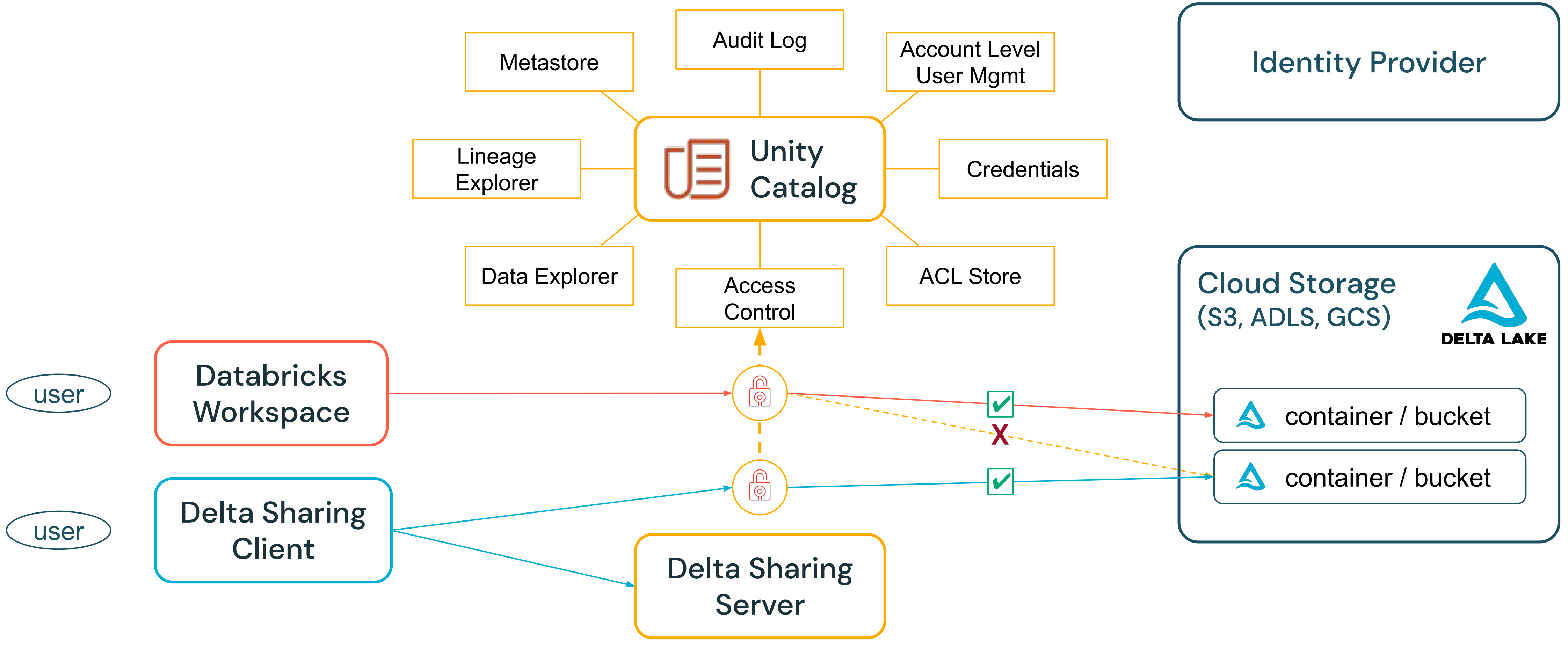
Introducing Unity Catalog

Unified view of your data estate via a centralized metadata and user mgmt



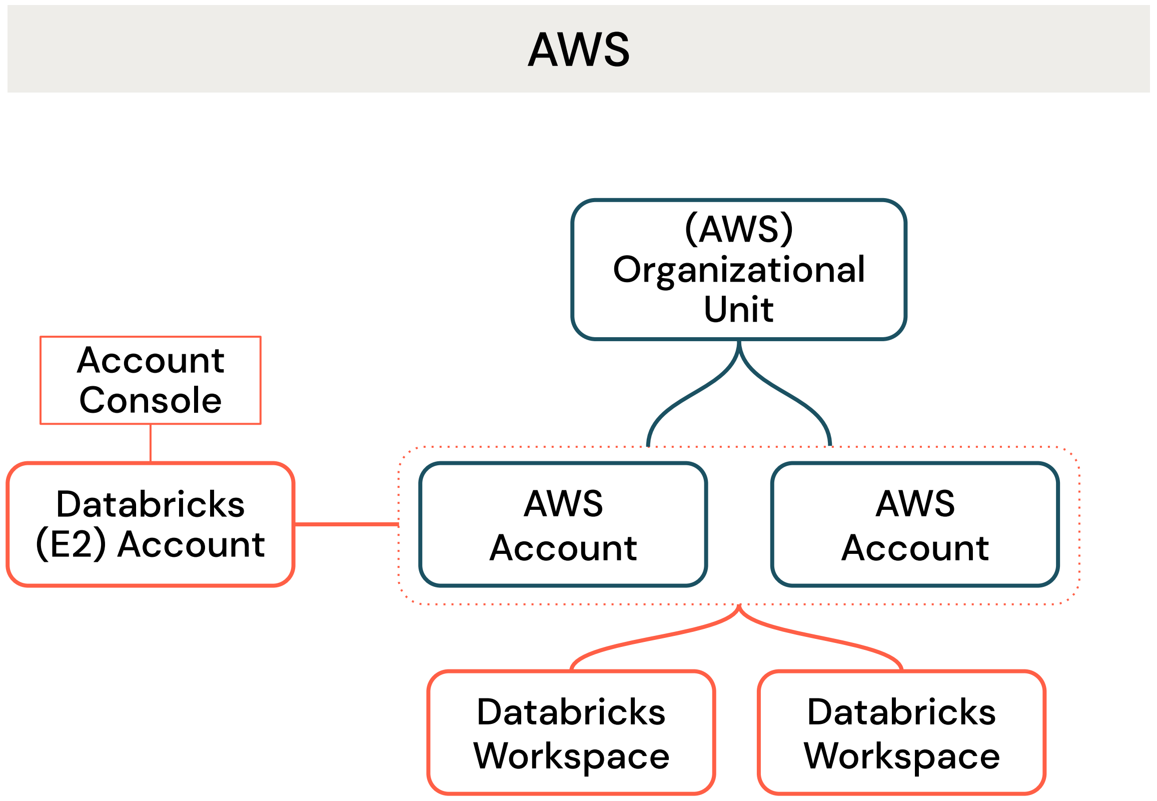
Unity Catalog & Delta Sharing – Components

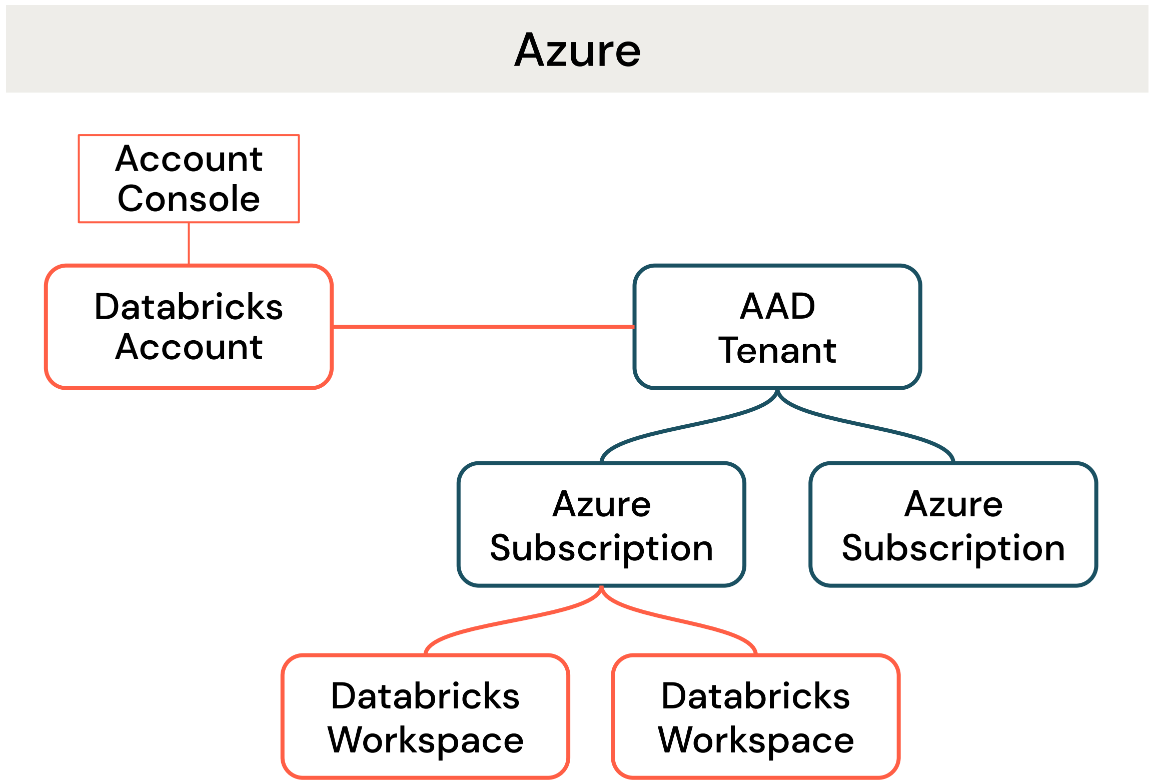
Centralized Governance



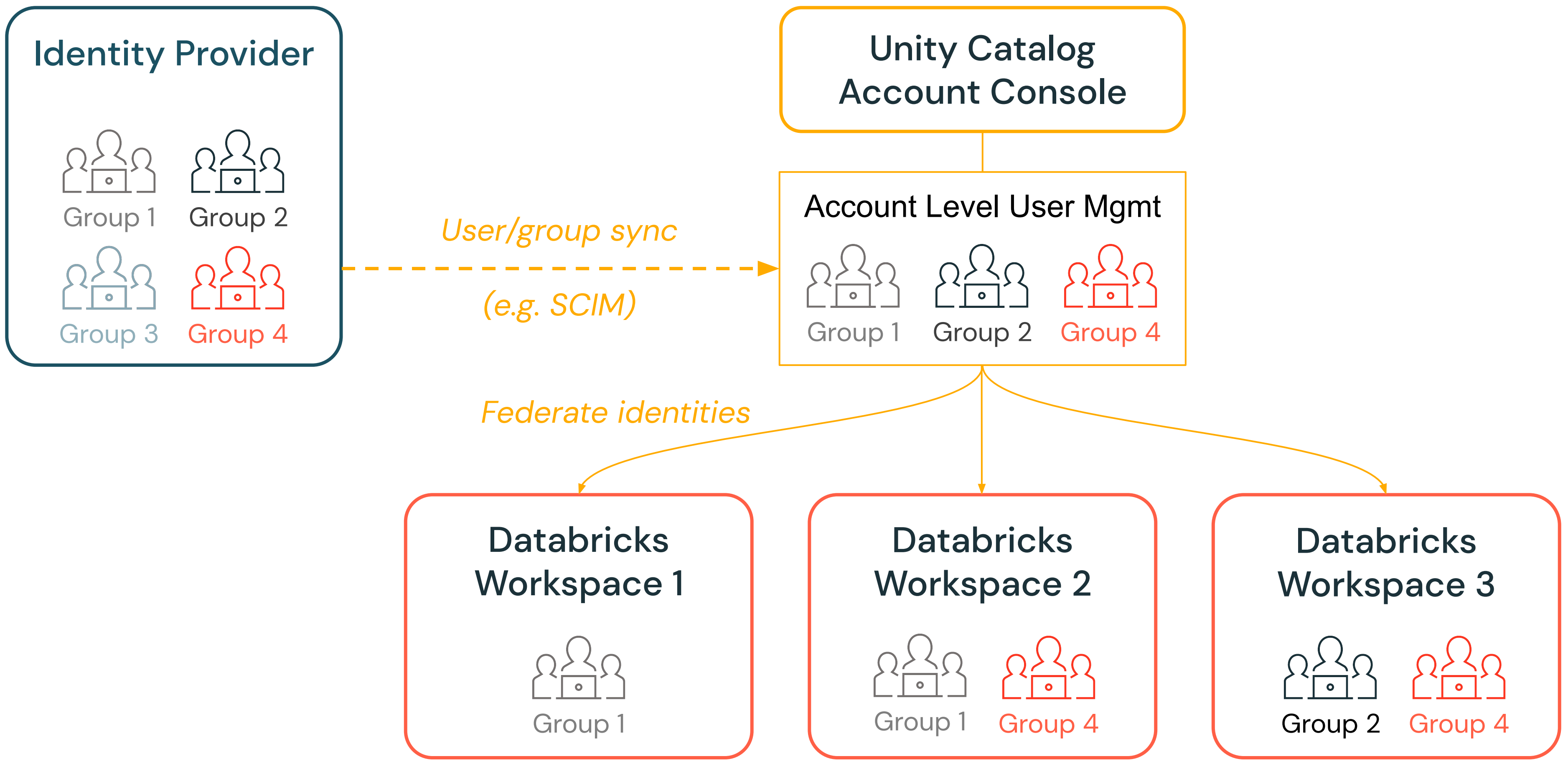
✓ *Short-lived token*
X *Access denied*

Databricks Accounts and the Cloud Provider Hierarchy

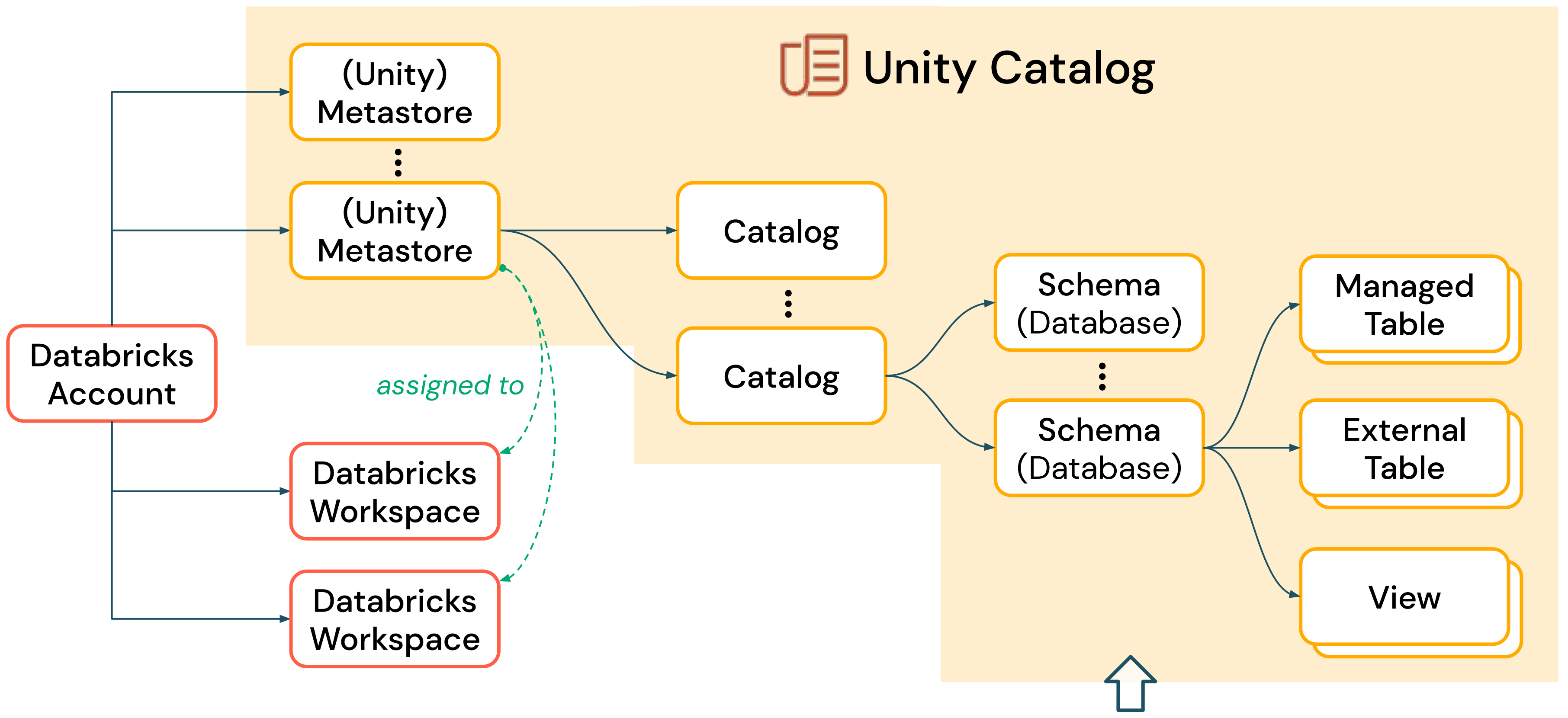




Identity Federation with Unity Catalog

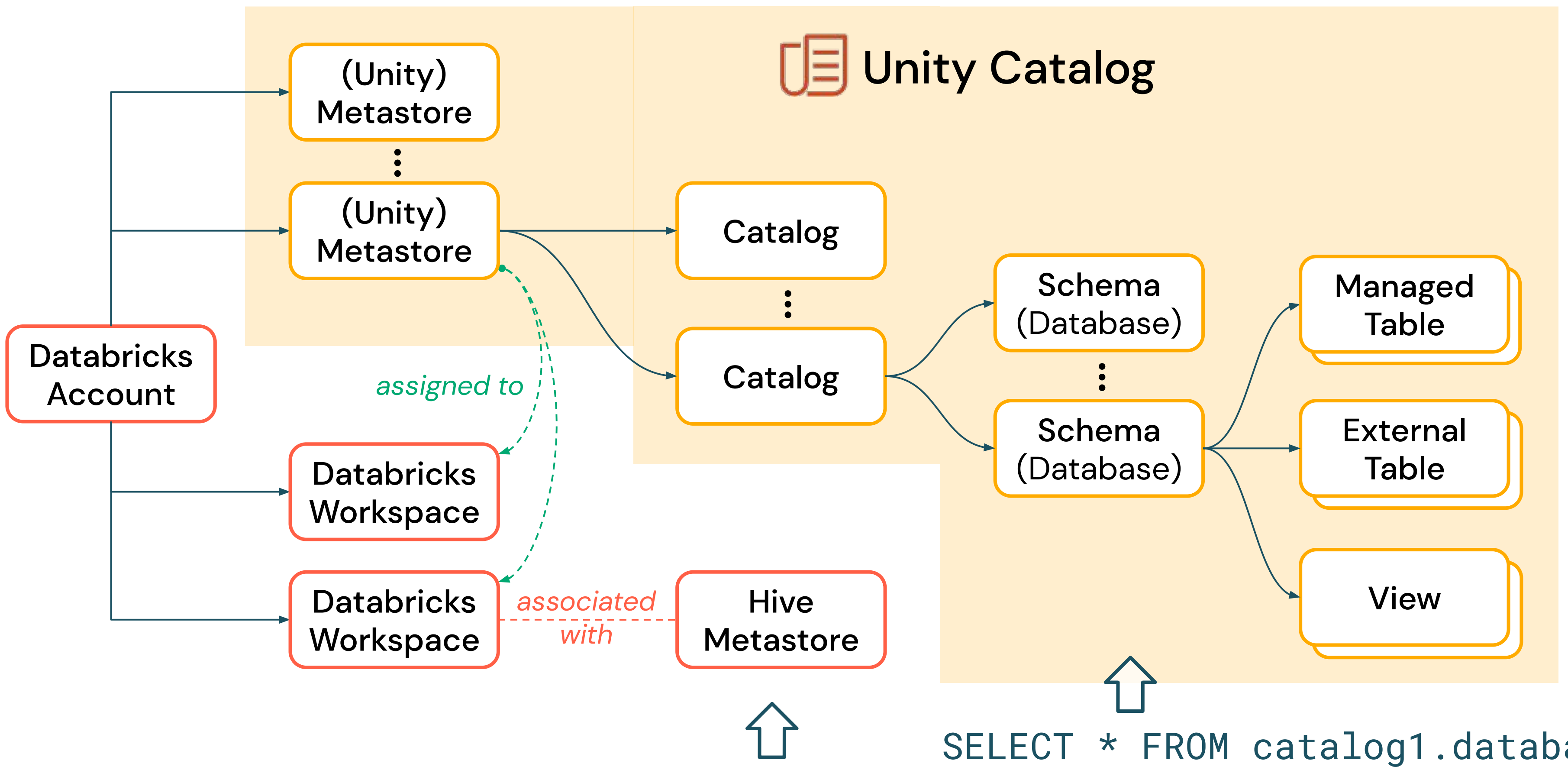


The three level namespace of UC



```
SELECT * FROM catalog1.database1.table1;
```

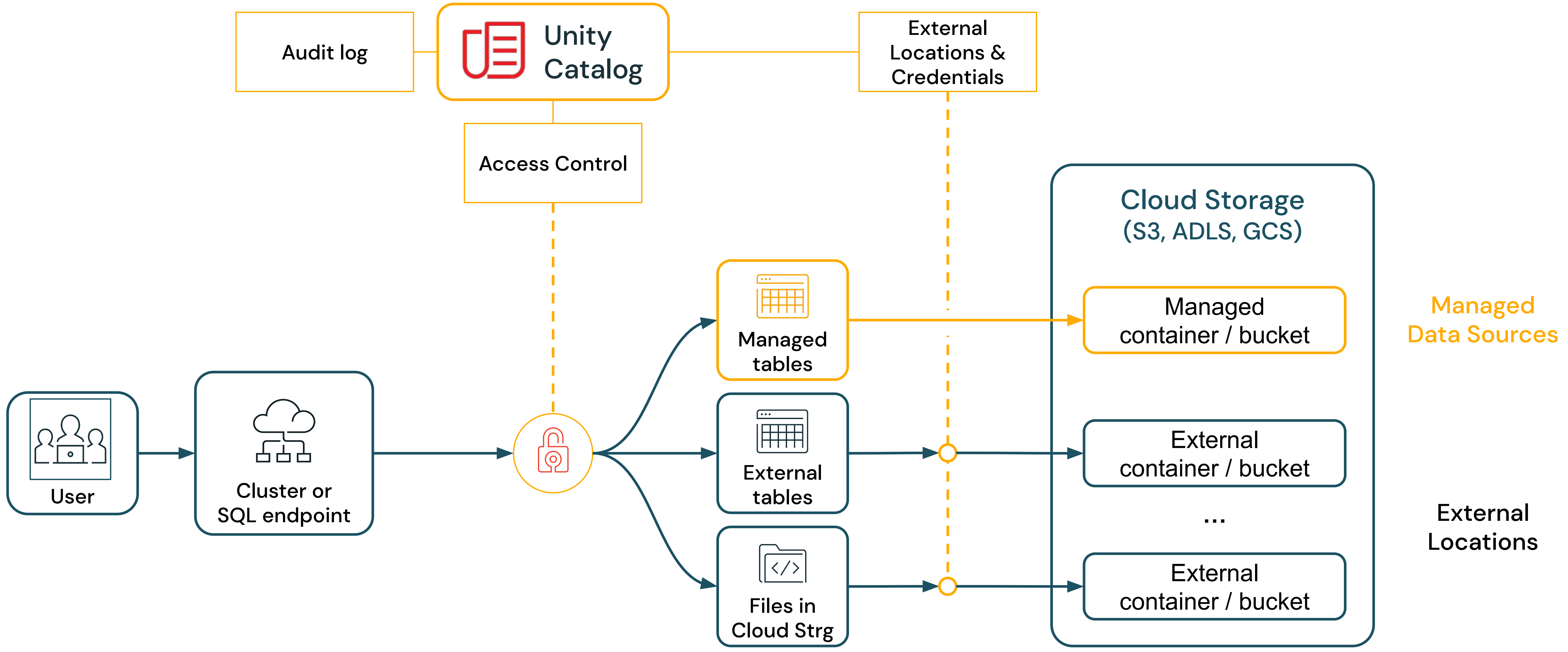
Hive Metastore is integrated into UC



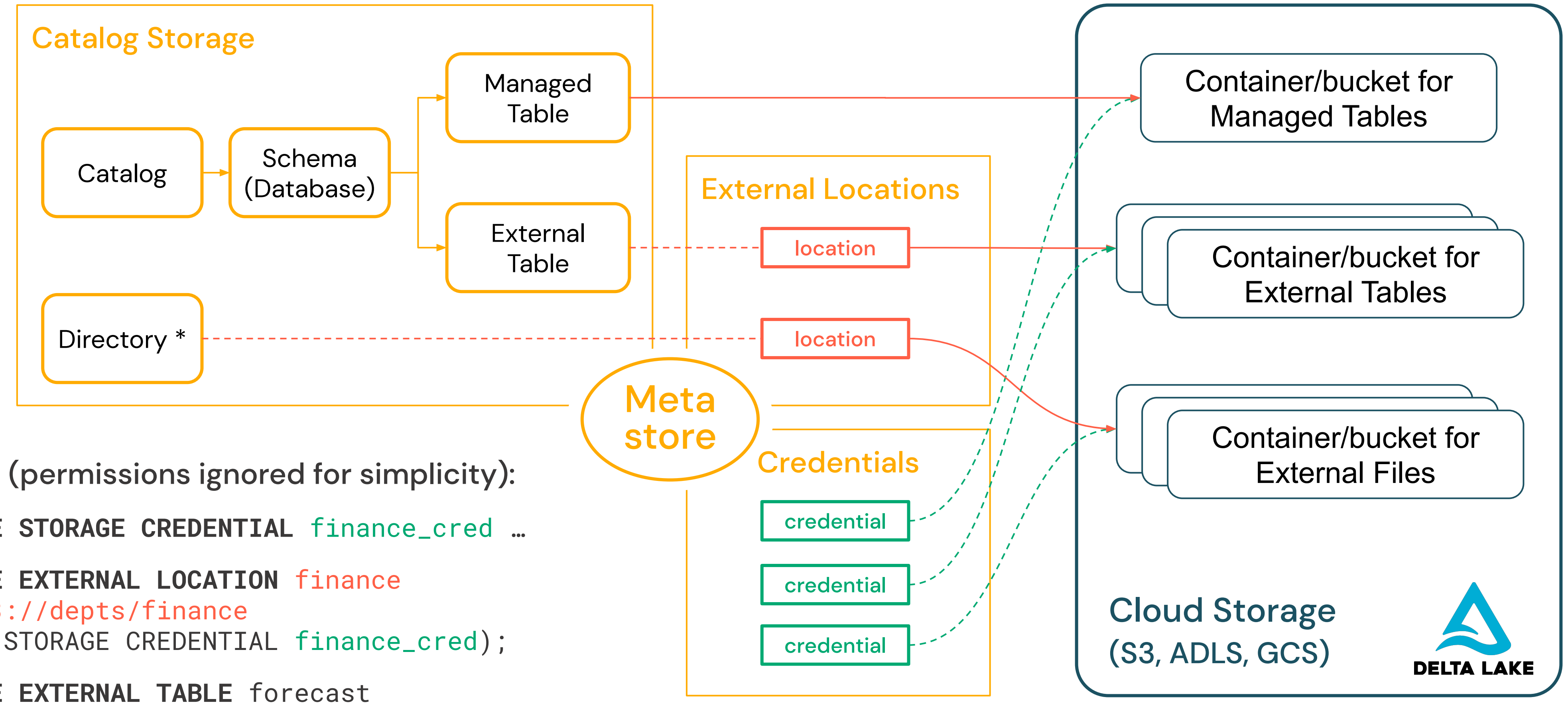
```
SELECT * FROM catalog1.database1.table1;
```

```
SELECT * FROM hive_metastore.database2.table2;
```

Managed Data Sources & External Locations



Metastore, external locations and credentials

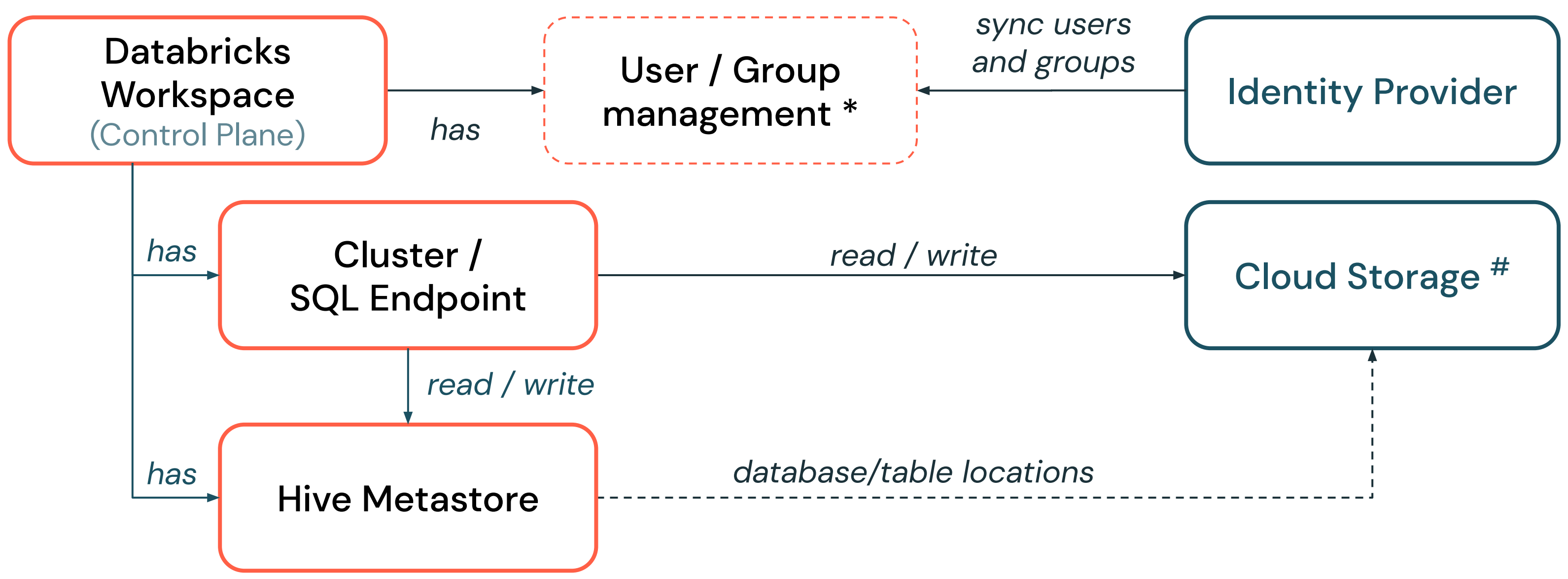


Example (permissions ignored for simplicity):

```
  CREATE STORAGE CREDENTIAL finance_cred …

  CREATE EXTERNAL LOCATION finance
  URL s3://depts/finance
  WITH (STORAGE CREDENTIAL finance_cred);

  CREATE EXTERNAL TABLE forecast
  LOCATION s3://depts/finance/forecast;

* CREATE DIRECTORY eu_invoices
  LOCATION s3:/depts/finance/eu/invoices;
```

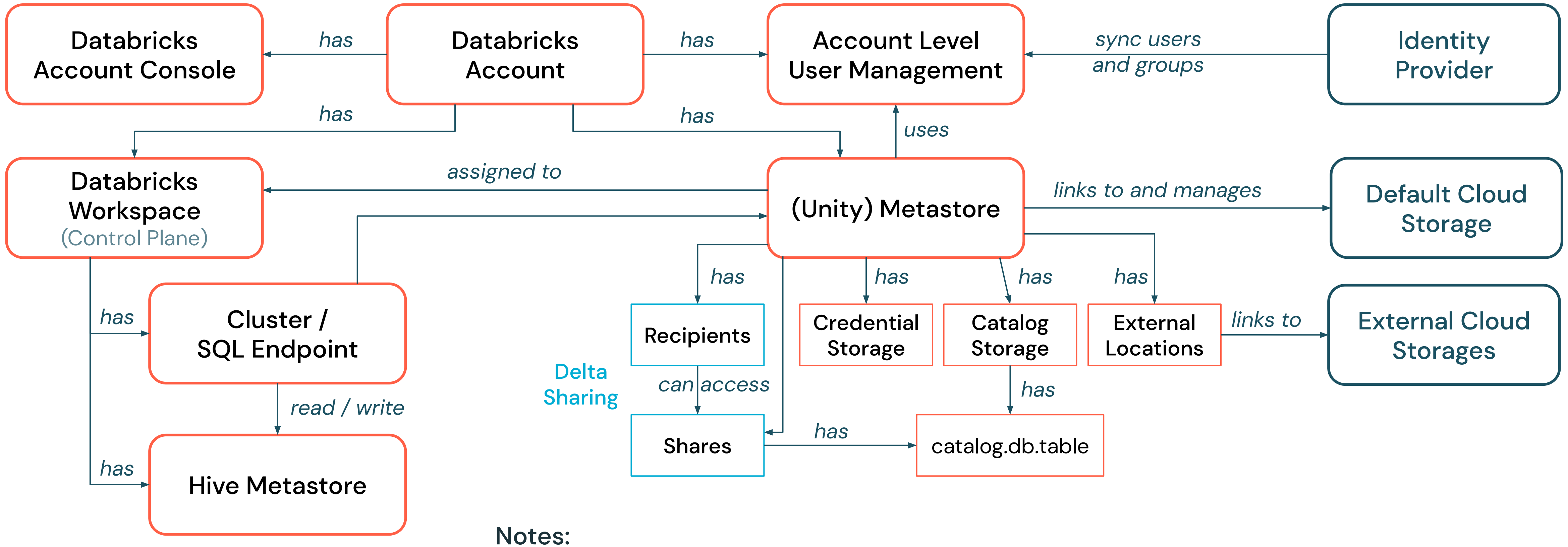
* coming soon

Object relations before Unity Catalog



* An integrated module of every Databricks Workspace
+ Could also be an external Metastore
# Any accessible cloud storage or the root container (dbfs)

Object relations with Unity Catalog (UC)



Notes:

- Using Unity Catalog is optional. Workspaces can still use a Hive Metastore only
- There can be more than one Unity Metastore (UC) per Databricks Account (e.g. for regional isolation or for isolation of lines of business)
- Every workspace can only attach to one UC Metastore, however one Unity Metastore can be assigned to several Workspaces

Data access patterns

Life of a query without Unity Catalog

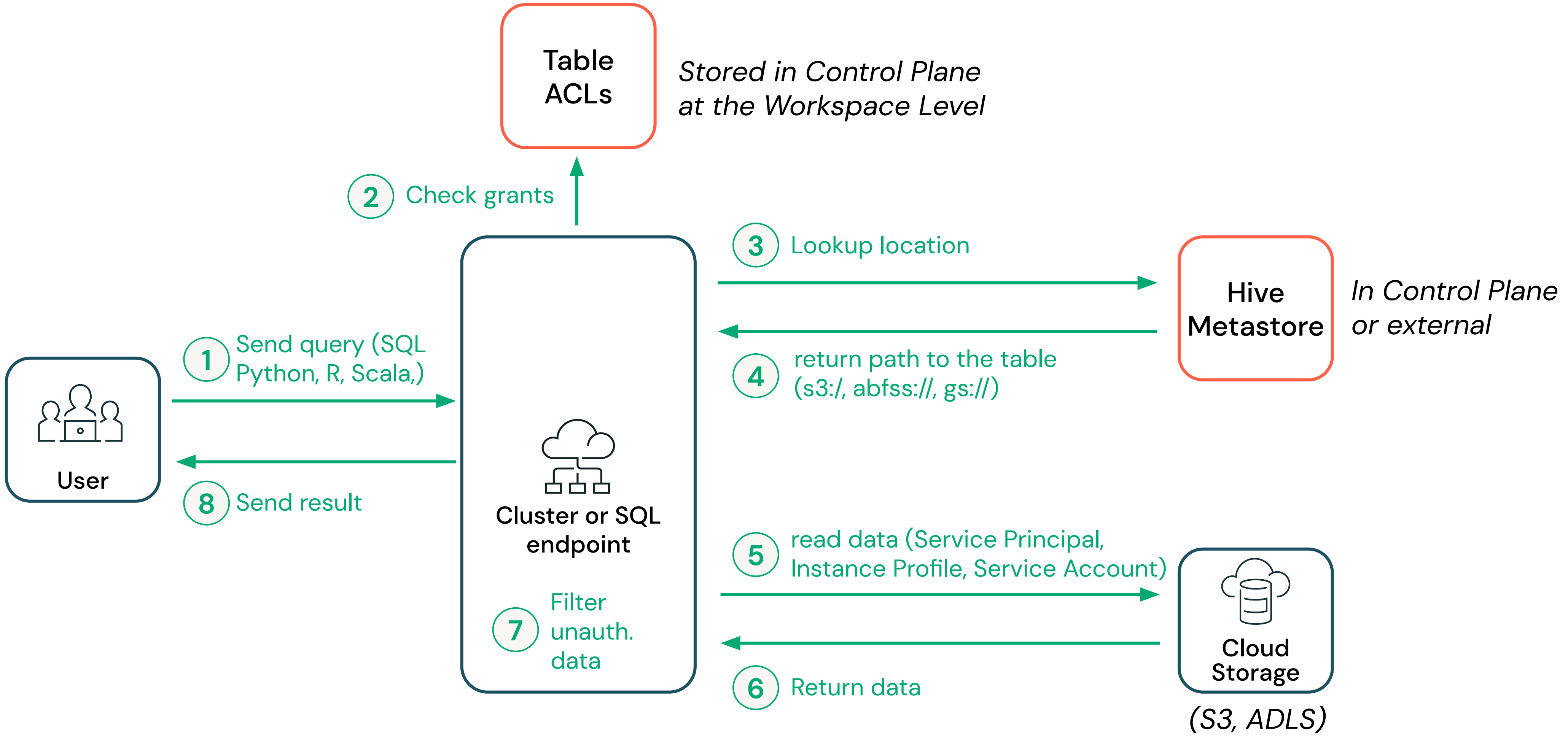
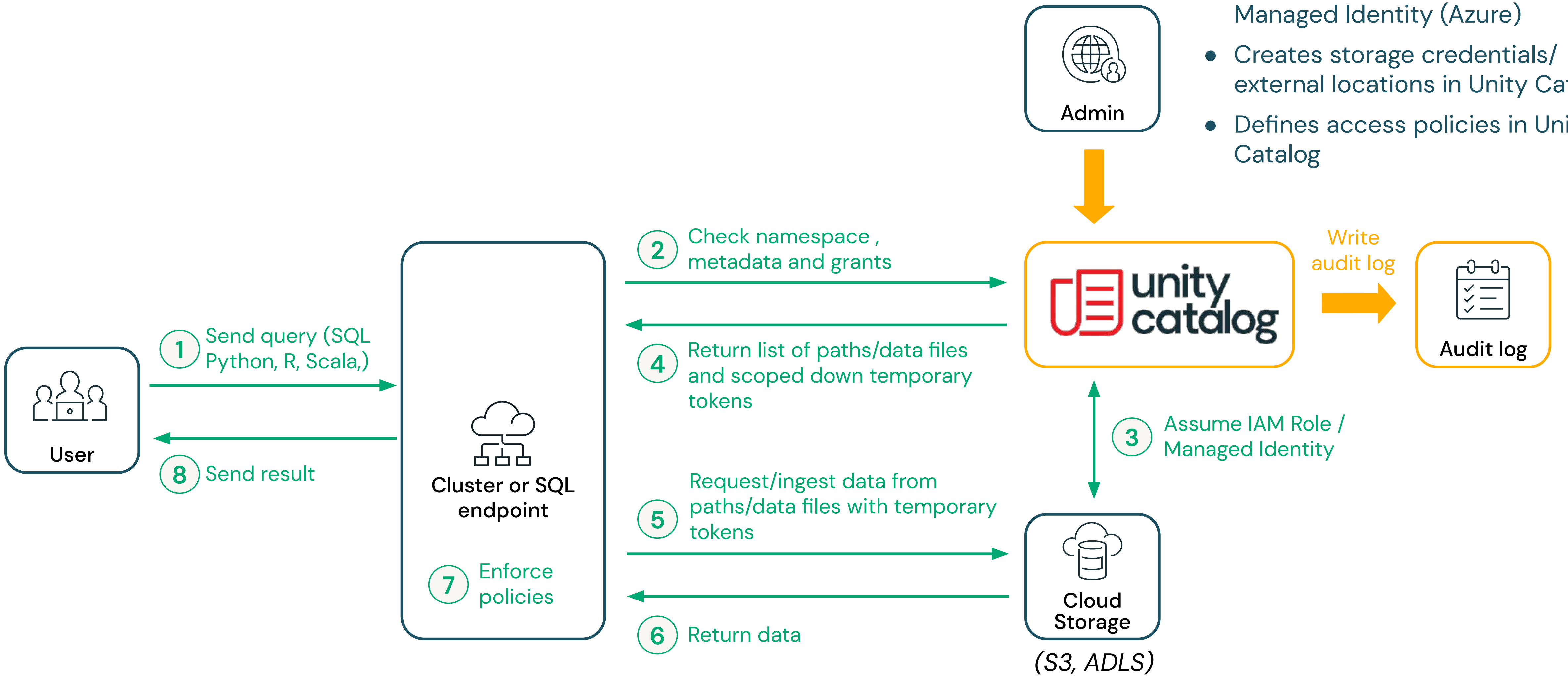


Table ACLs
Stored in Control Plane at the Workspace Level
2 Check grants
3 Lookup location
Hive Metastore
In Control Plane or external
1 Send query (SQL Python, R, Scala,)
4 return path to the table (s3:/, abfss://, gs://)
User
8 Send result
Cluster or SQL endpoint
5 read data (Service Principal, Instance Profile, Service Account)
7 Filter unauth. data
Cloud Storage
6 Return data
(S3, ADLS)

Life of a query with Unity Catalog



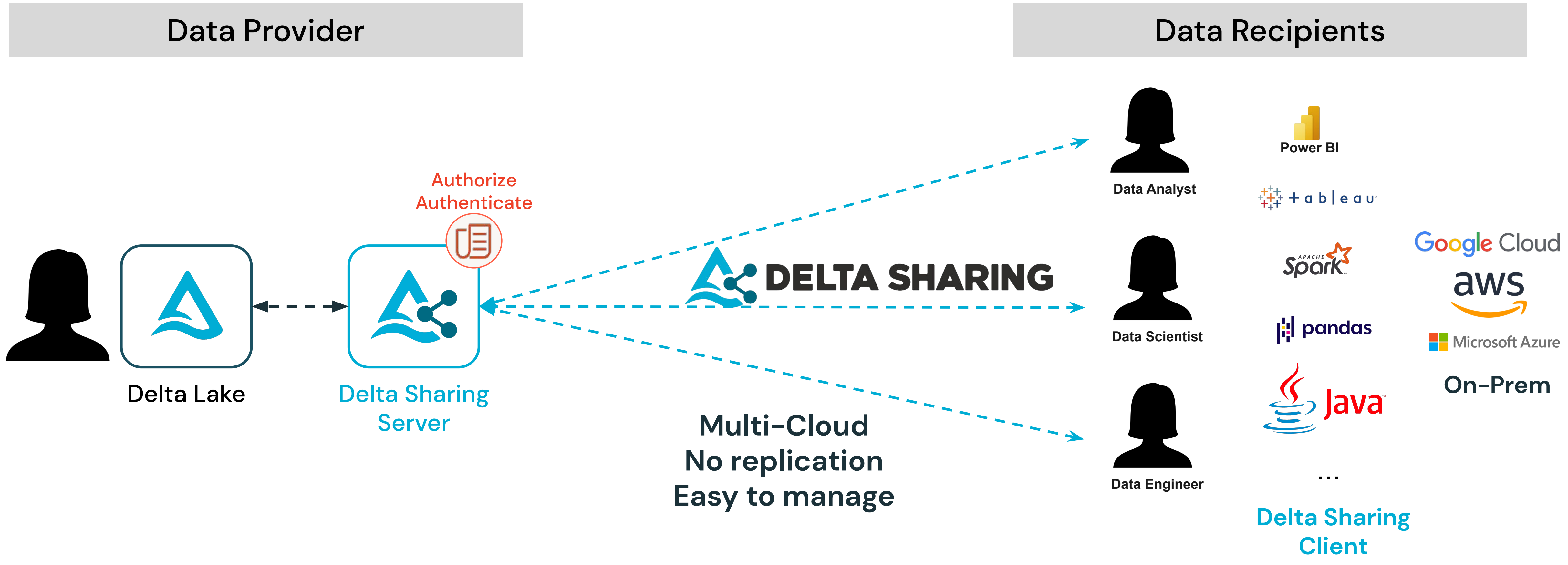
Admin
unity catalog
Write audit log
Audit log
User
1 Send query (SQL Python, R, Scala,)
8 Send result
Cluster or SQL endpoint
7 Enforce policies
2 Check namespace , metadata and grants
4 Return list of paths/data files and scoped down temporary tokens
3 Assume IAM Role / Managed Identity
5 Request/ingest data from paths/data files with temporary tokens
6 Return data
Cloud Storage
(S3, ADLS)

- Creates an IAM role (AWS)/ Managed Identity (Azure)
- Creates storage credentials/ external locations in Unity Catalog
- Defines access policies in Unity Catalog

UC and Delta Sharing

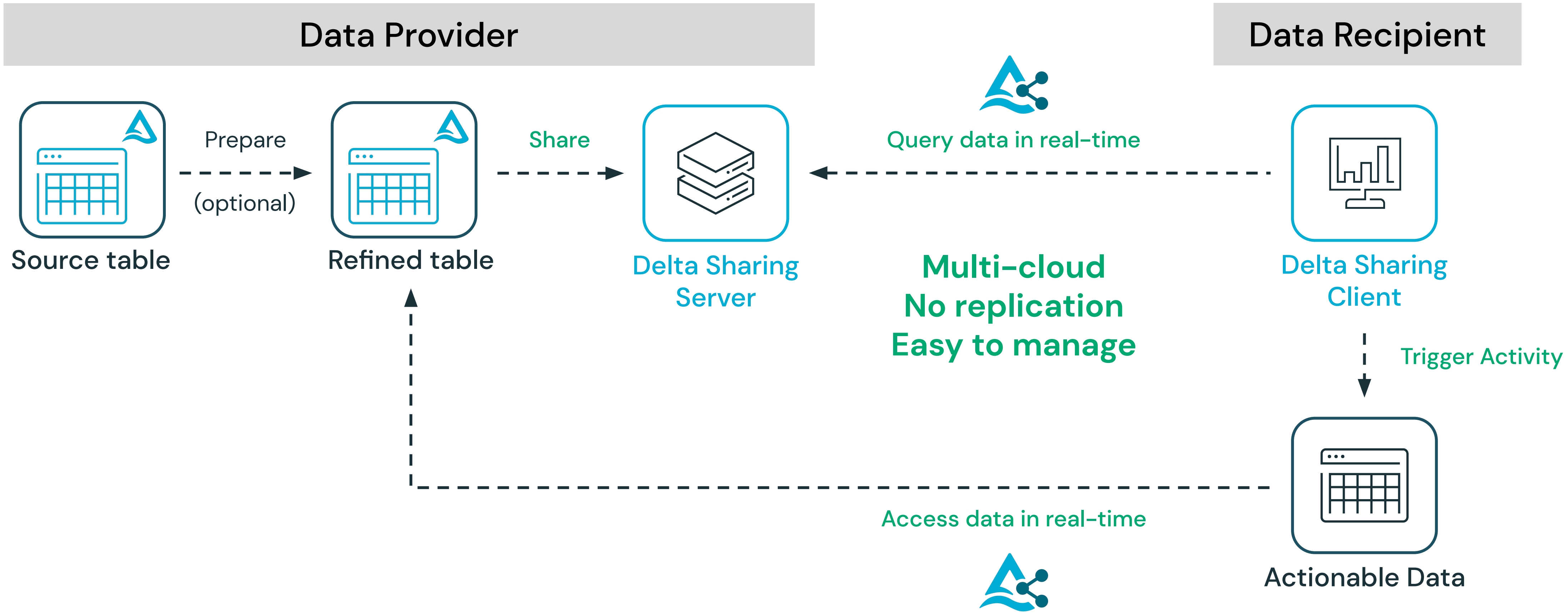
A simple, open and easy approach to data sharing

Reduce data sharing and collaboration from days to real-time

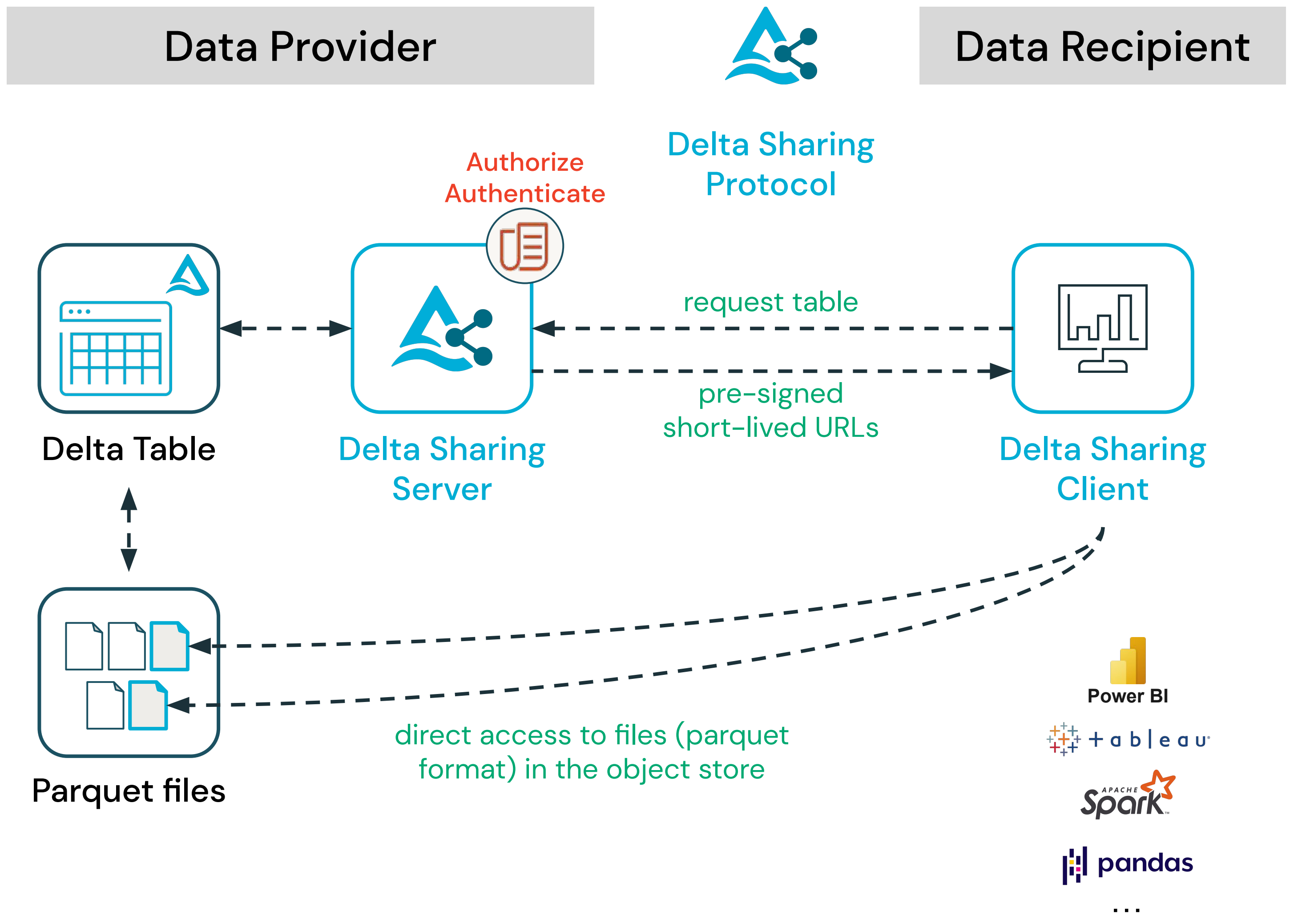


Streamlined Sharing with Delta Sharing

Delta Sharing cuts collaboration time with partners from days to real-time.



Under the Hood



Delta Sharing Protocol:

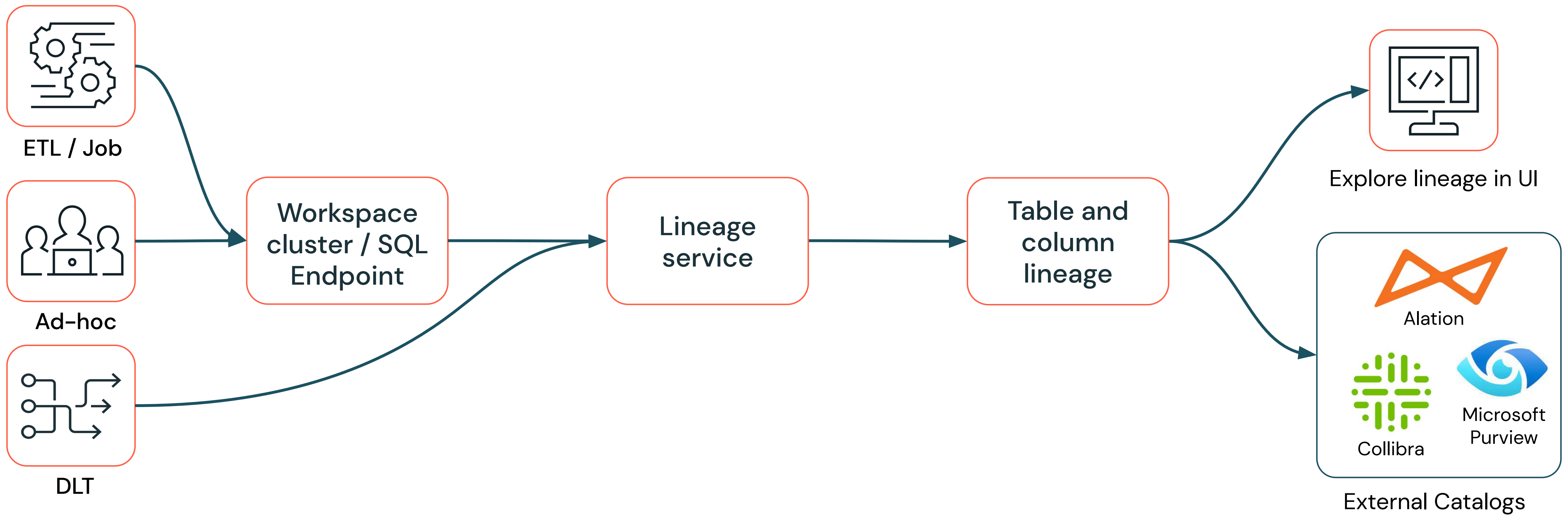
- Client authenticates to Sharing Server
- Client requests a table (including filters)
- Server checks access permissions
- Server generates and returns pre-signed short-lived URLs
- Client uses URLs to directly read files from object storage

Notes:

- Sharing happens on Delta part files, supporting full tables, partitions, delta versions, …
- Client is system independent, just needs to be able to read parquet files
- In Databricks Sharing Server and ACL checks are integrated with Unity Catalog

Lineage

Lineage flow



- Code (any language) is submitted to a cluster or SQL endpoint or DLT executes data flow

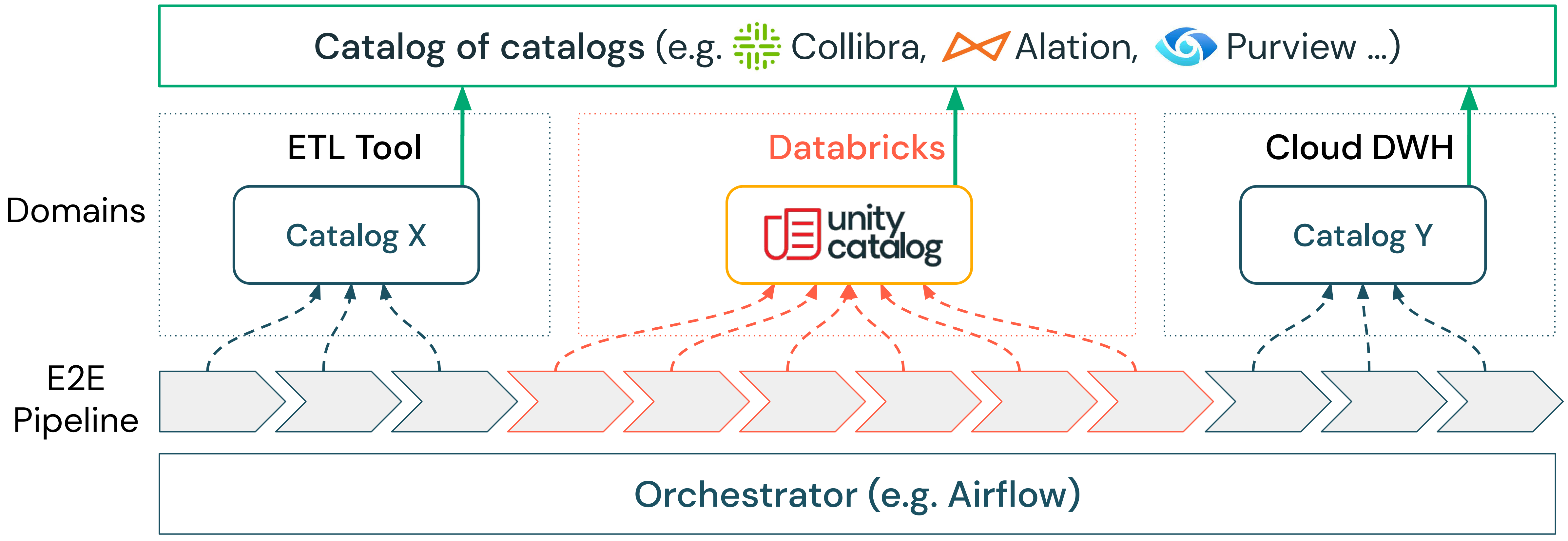
- Lineage service analyzes logs emitted from the cluster, and pulls metadata from DLT
- Assembles column and table level lineage

- Presented to the end user graphically in Databricks
- Lineage can be exported via API and imported into other tool

UC and Partners

Unity Catalog and Catalog Partners

Better together



Catalog of catalogs (e.g. Collibra, Alation, Purview …)
Domains
ETL Tool
Catalog X
Databricks
unity catalog
Cloud DWH
Catalog Y
E2E Pipeline
Orchestrator (e.g. Airflow)

Lineage information flow:

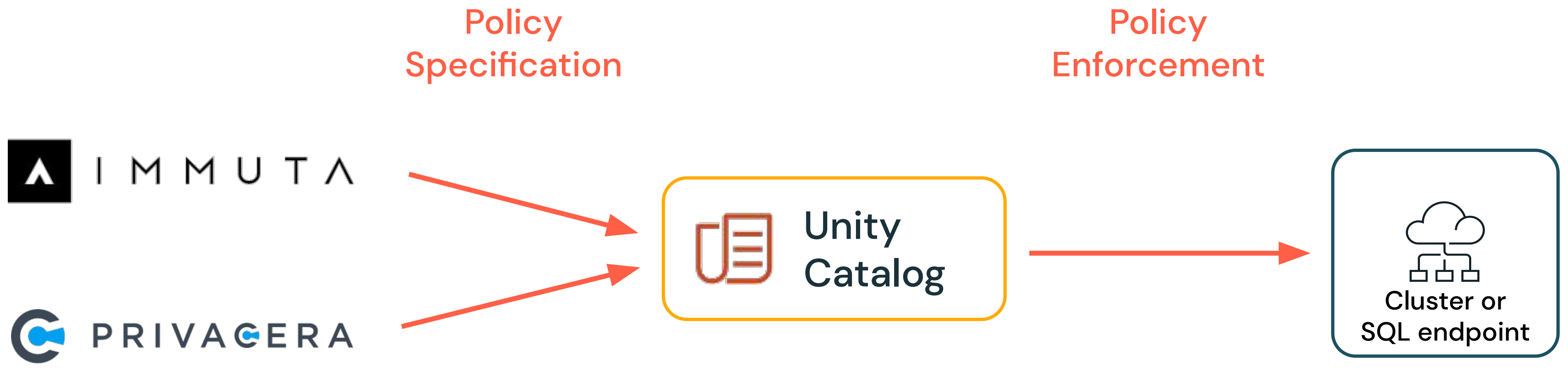
- Pipeline step sending lineage to domain's catalog (e.g. UC)
- Domain's catalog to global catalog of catalogs

Unity Catalog and Governance Partners

Better together

Greatly improves the experience in Immuta and Privacera:

- No longer limits the languages that these products can work in
- No longer limits the APIs that your users can use
- Improves performance and robustness
- Adds a common enforcement layer



Best Practices for UC

Clusters/endpoints with Unity Catalog

Standard clusters with User Isolation mode

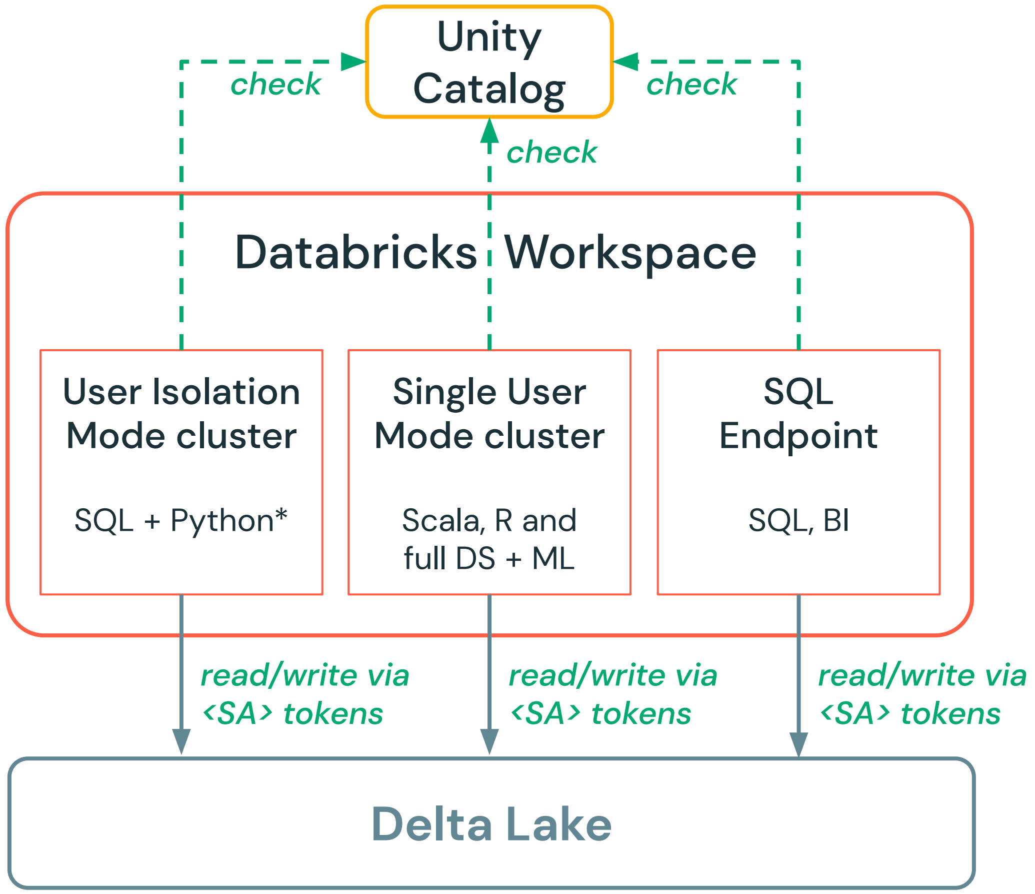
- Use User isolation mode for general workloads using SQL (ETL, data exploration, …).
- ML DBR will not be supported at the beginning

Standard clusters with Single User mode

- Use Single User mode for Scala users and for Data Scientists. These clusters support the full feature set of Databricks, e.g. ML DBR, MLflow.
- While only the owner can execute code on this cluster, notebook collaboration via sharing works well. Co-workers see everything, however, they cannot execute cells

SQL endpoints

- Use SQL endpoints for Business Analysts either using Databricks SQL Editor or external BI tools like Power BI, Tableau, …


Unity Catalog
check
check
check
Databricks Workspace
User Isolation Mode cluster
SQL + Python*
Single User Mode cluster
Scala, R and full DS + ML
SQL Endpoint
SQL, BI
read/write via <SA> tokens
read/write via <SA> tokens
read/write via <SA> tokens
Delta Lake

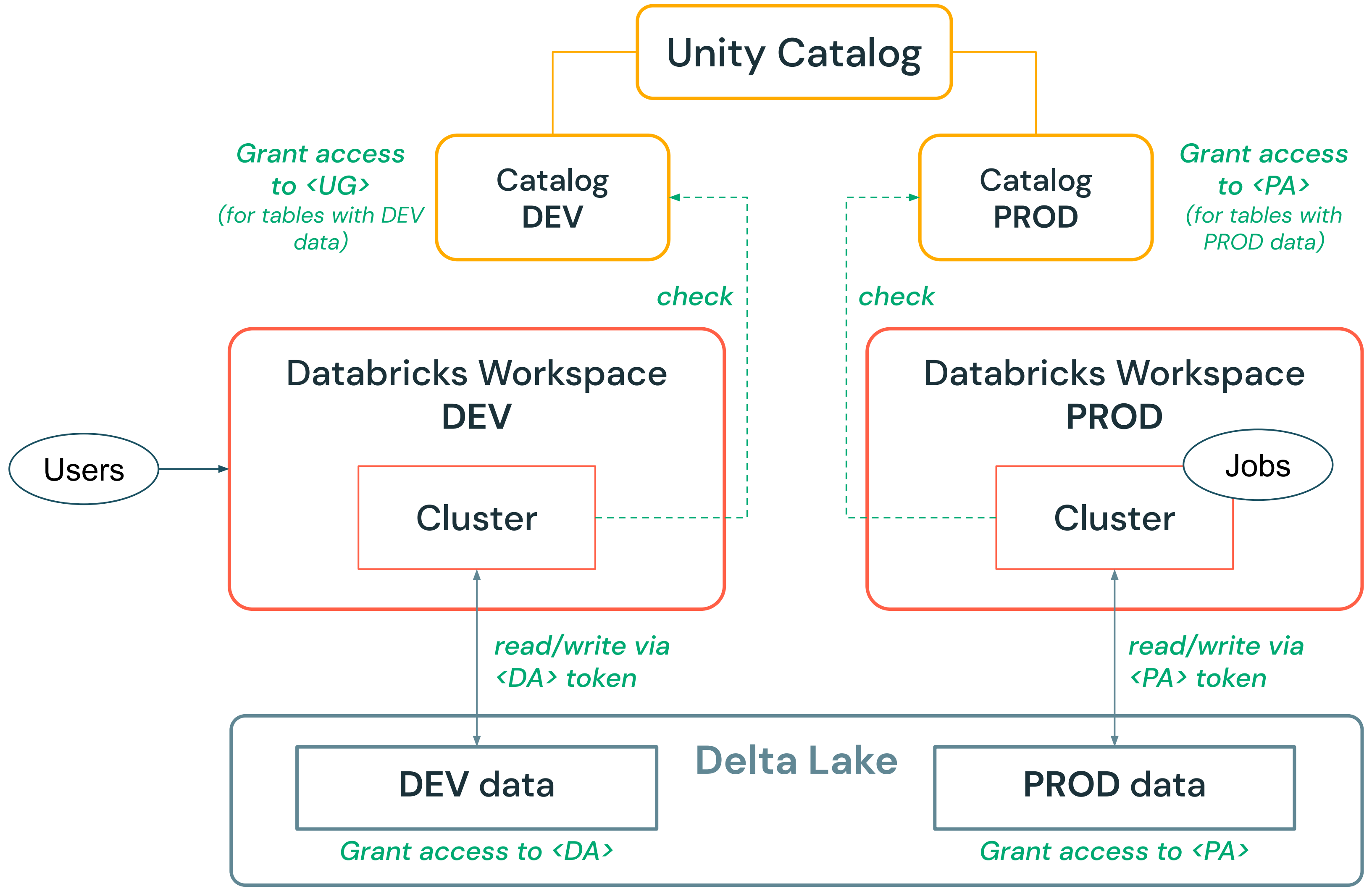
<SA> System Account (Service Principal, Instance Profile, Service Account)

Unity Catalog and Data Science

There are different usage scenarios for Data Science

1. **Single node Data Science**
   Coming from traditional (laptop based) Data Science world, still many problems do not need distributed computing. For single node Data Science a Single User cluster is the best choice
2. **Exploratory Data Science**
   For tasks that involve analysing data, creating reports or finding patterns, a standard DBR can be used on a User Isolation cluster and the needed data exploration libraries (pandas, scikit learn, …) can be installed as notebook scope libraries.
3. **Complex model training**
   When building complex models, especially when distributed model training is needed, a Data Scientist prefers to have full control of the compute resources. Again, Single User clusters are a great choice.

Software Development Lifecycle setup w/ UC


Unity Catalog
Grant access to <UG> (for tables with DEV data)
Catalog DEV
Catalog PROD
Grant access to <PA> (for tables with PROD data)
check
check
Databricks Workspace DEV
Users
Cluster
Databricks Workspace PROD
Jobs
Cluster
read/write via <DA> token
read/write via <PA> token
Delta Lake
DEV data
Grant access to <DA>
PROD data
Grant access to <PA>

Note:

- One of the reasons to have different Workspaces for DEV and PROD is that they could reside in different VNets/VPCs. This is independent of UC, but leads to a setup as it is shown here.

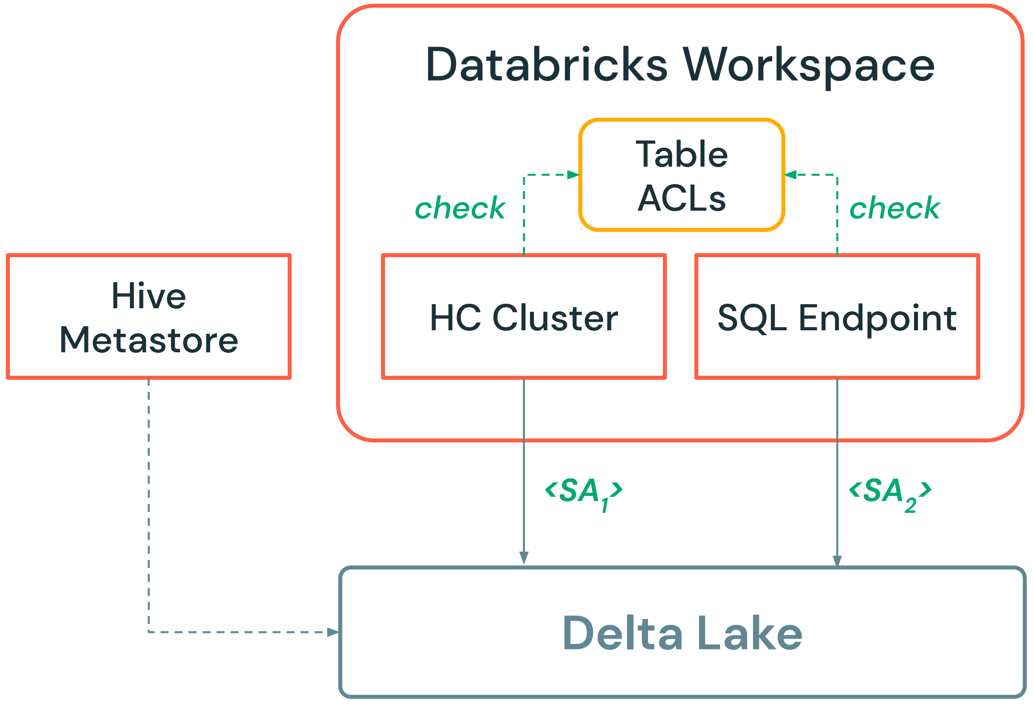
<DA>	DEV System Account (Service Principal, Instance Profile, Service Account)
<PA>	PROD System Account (Service Principal, Instance Profile, Service Account)
<UG>	User Group (Developers, Data Engineers, Data Scientists)

Best Practices before migration to UC

High Concurrency clusters with Table ACLs

- **Don't use DBFS for data**, there is no permission control
- **Use Delta for all data**
- Hive Metastore (Databricks internal or external)
  - **Point default path for managed tables to the Delta Lake**, i.e. do not use DBFS

- Data Science on High Concurrency (HC) clusters with Table ACLs:
  - Single User cluster do not exists with Table ACLs
  - No support for
    - direct file reads/writes (neither dbfs nor object store - even when mounted)
    - ML DBRs or AutoML
    - R
  - Python packages w/o Java components can be installed with %pip and will most probably work.
  - MLflow can be installed via %pip. However, no auto-logging and no support for saving of models or artifacts to DBFS.
  - Artifacts and models can be pushed to the Model Registry



‹SA$_n$› System Account (Service Principal, Instance Profile, Service Account)