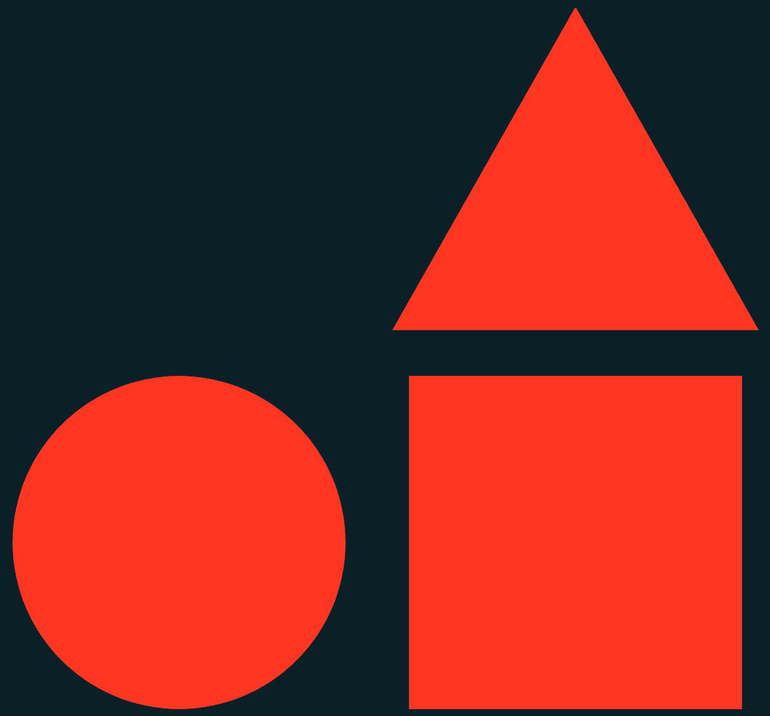
databricks

Databricks Data Privacy



Databricks Academy

Course Learning Objectives

- Store sensitive data appropriately to simplify granting access and processing deletes.
- Perform data masking and configure fine grained access control to configure appropriate privileges to sensitive data.
- Process deletes to ensure compliance with the right to be forgotten.

Databricks Data Privacy

Course Prerequisites

- Ability to perform basic code development tasks using the Databricks Data Engineering & Data Science workspace (create clusters, run code in notebooks, use basic notebook operations, import repos from git, etc)
- Intermediate programming experience with **PySpark**
- Intermediate programming experience with **Delta Lake**
- Beginner experience defining **DLT pipelines** using PySpark and using the DLT UI

Lab Exercise Environment

Technical Details

- Your lab environment is provided by Vocareum.
- It will open in a new tab.
- It has been configured with the permissions and resources required to accomplish the tasks outlined in the lab exercise.
- Third-party cookies must be enabled in your browser for Vocareum's user experience to work properly.
- Make sure to enable pop-ups!

Agenda

Databricks Data Privacy	Time
Storing Data Securely	10 mins
Unity Catalog	60 mins
PII Data Security	40 mins
Streaming Data and CDF	70 mins

databricks

Storing Data Securely

Databricks Data Privacy

Storing Data Securely

Learning Objectives

- Identify common compliance regulations
- Identify Data Privacy Key Aspects and describe optimal approaches to meet compliance requirements
- How Databricks Simplifies Compliance

Agenda

Storing Data Securely
• Regulatory Compliance
• Data Privacy

databricks

Storing Data Securely

LECTURE

Regulatory Compliance

Regulatory Compliance

Overview

- EU = GDPR (General Data Protection Regulation)
- US = CCPA (California Consumer Privacy Act)
- Simplified Compliance Requirements
  - Inform customers what personal information is collected
  - Delete, update, or export personal information as requested
  - Process request in a timely fashion (30 days)

Regulatory Compliance

How Databricks Simplifies Compliance

- Reduce copies of your PII
- Find personal information quickly
- Reliably change, delete, or export data
- Built-in data skipping optimizations **(Z-order)** and housekeeping of obsolete/deleted data **(VACUUM)**
- Use transaction logs for auditing

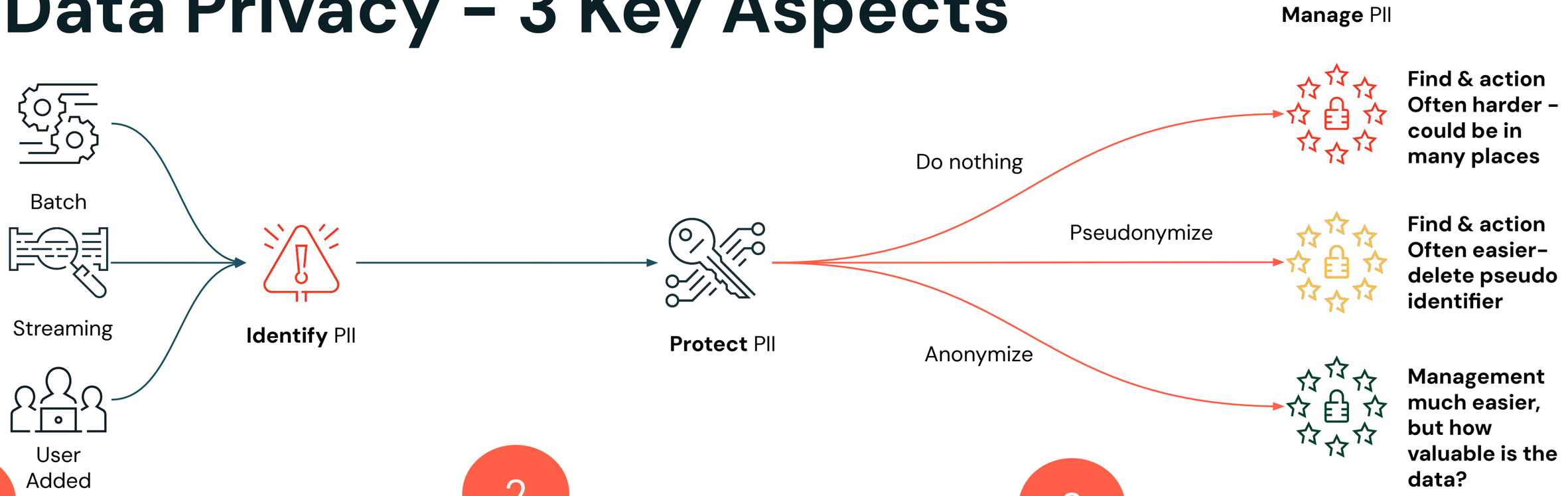
databricks

Storing Data Securely

LECTURE

Data Privacy

Data Privacy – 3 Key Aspects



1

Identify

- How accurate is my detection?
- How do I apply it to all of my data?
- How do I know that it's been applied?
- How performant is it?

2

Protect

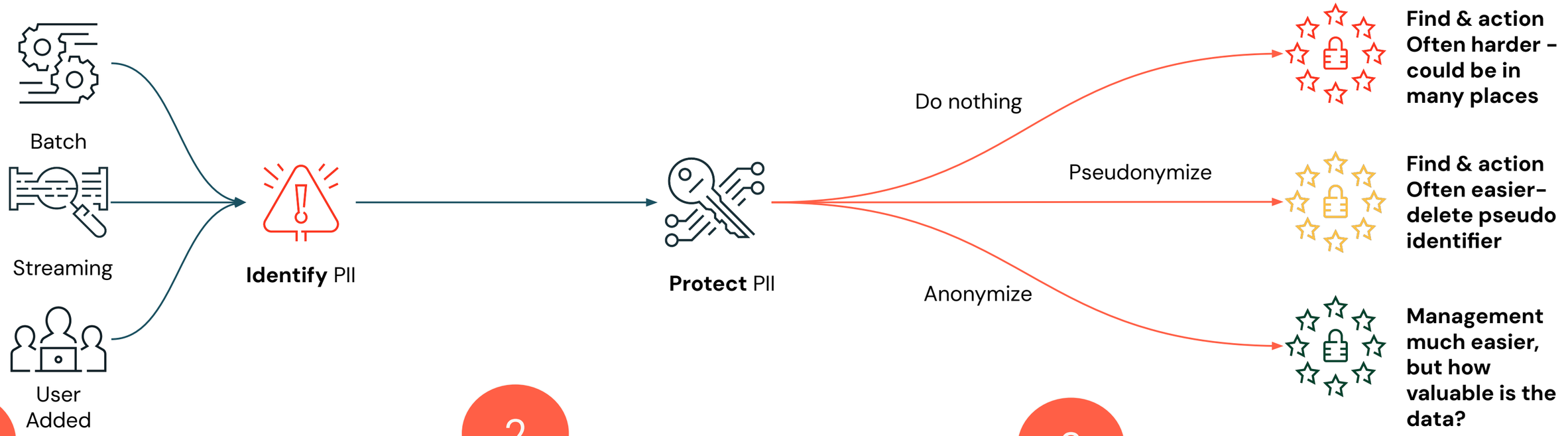
- Do I do nothing, anonymize or pseudonymize?
- If I anonymize, is the data still valuable?
- If I pseudonymize, how do I know it isn't easily reversible?

3

Manage

- How do I search across all of my data?
- How performant is that search?
- How do I apply actions (right to be forgotten, etc?)
- How performant is it when I do?

Data Privacy - How to Address Them



1

Identify

- Data Discovery
- Data Classification
- Data Mapping

2

Protect

- Data Security Measures
- Data Handling
- Consent Management
- Privacy by Design

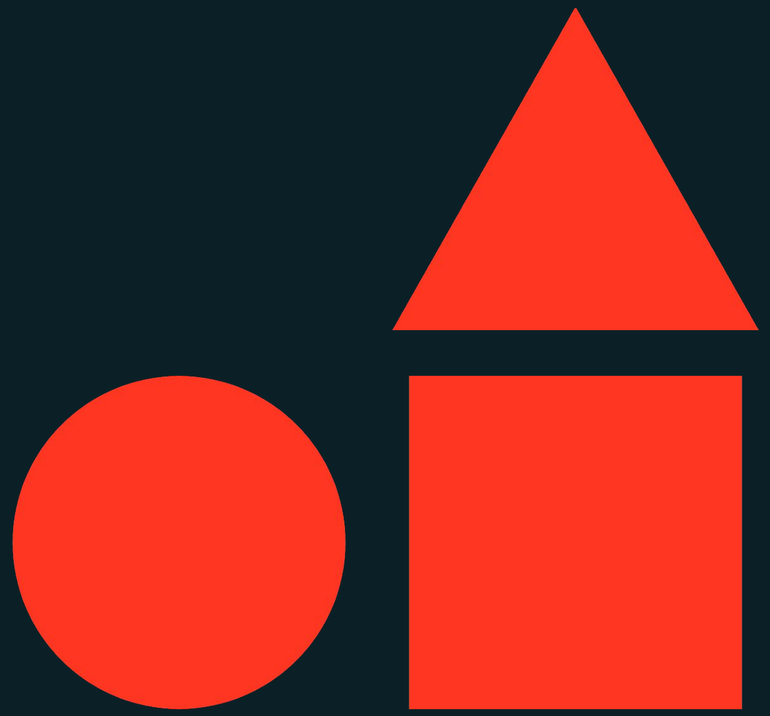
3

Manage

- Data Governance
- Compliance Management
- Monitoring and Auditing
- Data Subject Rights Management
- Incident Response
- Privacy Impact Assessment

databricks

Unity Catalog



Databricks Data Privacy

Unity Catalog

Learning Objectives

- Describe fundamental concepts about using Unity Catalog for data governance
- Describe Dynamic views, Row Filters and Column Masking to perform data masking and control access to rows and columns
- Describe several approaches in how to audit your data
- Evaluate the Unity Catalog security model, and data isolation to implement robust data governance strategies

Agenda

Unity Catalog
• Key Concepts and Components
• Audit Your Data
• Data Isolation
• Securing Data in Unity Catalog (Demo)

databricks

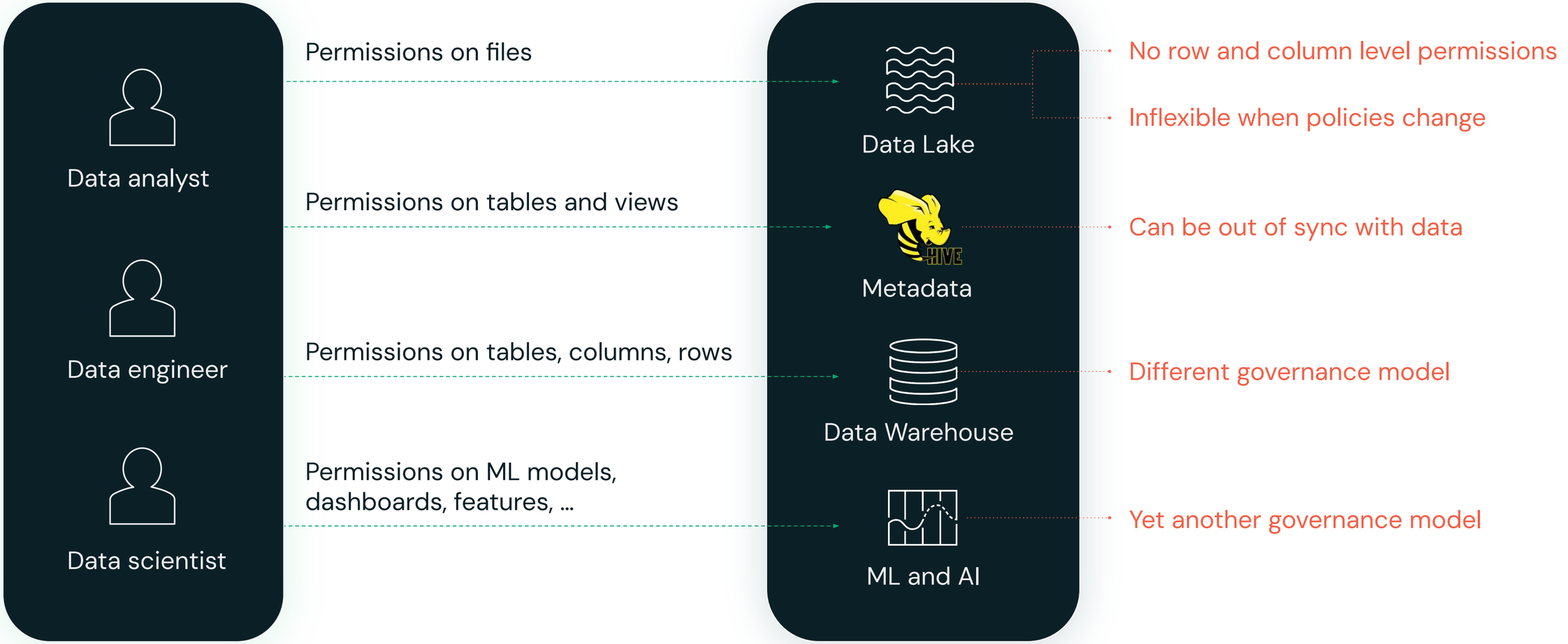
Unity Catalog

LECTURE

Key Concepts and Components

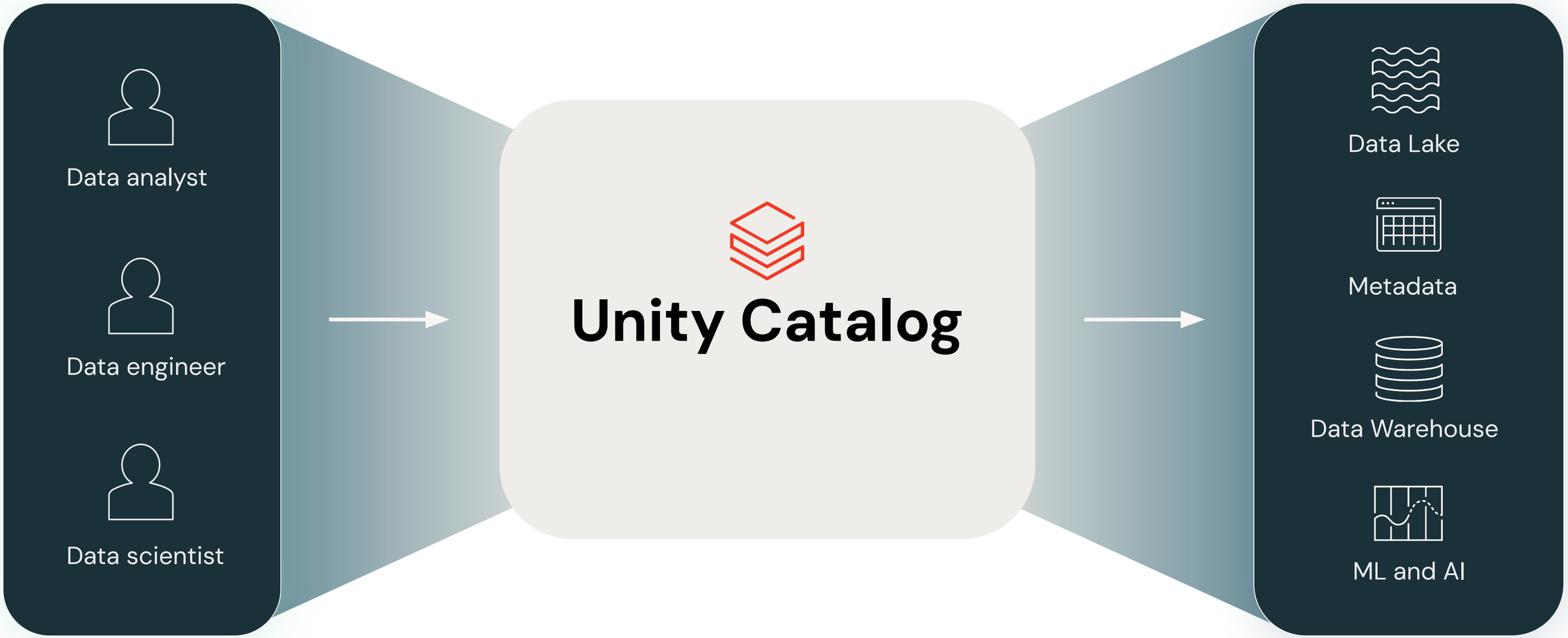
Key Concepts and Components

Governance for Data, Analytics and AI is complex



Databricks Unity Catalog

Unified governance for data, analytics and AI



Data Security Model

Access Control Lists (ACLs)

Privileges

- USE CATALOG
- USE SCHEMA
- CREATE TABLE
- SELECT
- MODIFY
- EXECUTE

Securables

- Metastore
- External Storage Access
- Storage Credential
- External Location
- Catalog
- Schema (Database)
- View
- Table
- Function
- Volume
- Model
- Query Federation
- Foreign Catalog
- Connection
- Delta Sharing
- Share
- Recipient
- Provider

Principals

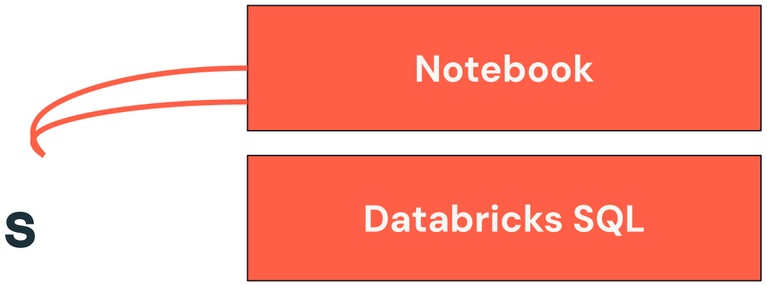
- User
- Service principal
- Group

Data Security Model

Managing ACLs

1. SQL

   ```
   GRANT SELECT ON TABLE t TO analysts
   REVOKE SELECT ON TABLE t FROM analysts
   ```



2. Catalog explorer

   Data Science & Engineering workspace or DBSQL

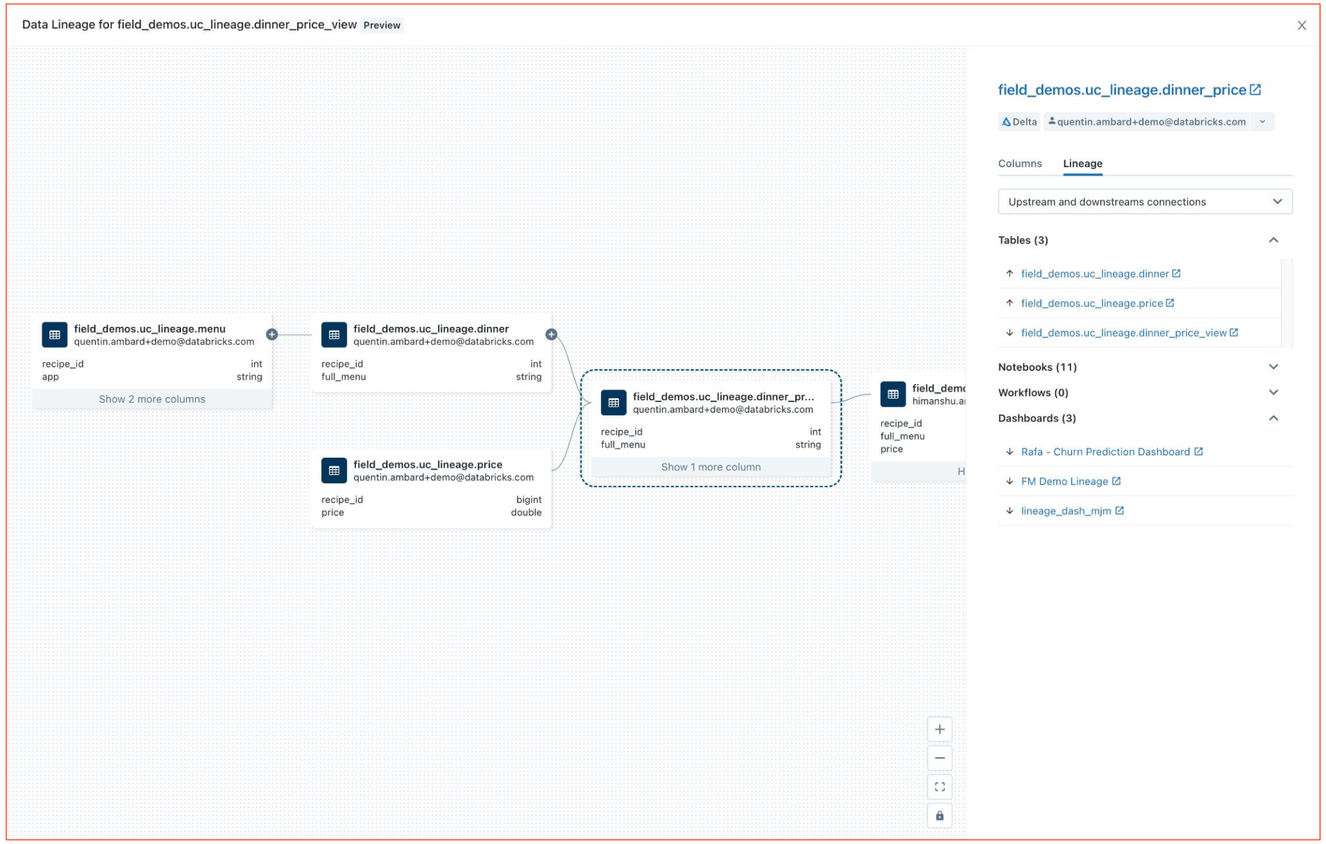
3. Programatically

   Databricks CLI, Terraform, and REST APIs

Automated Lineage for all Workloads

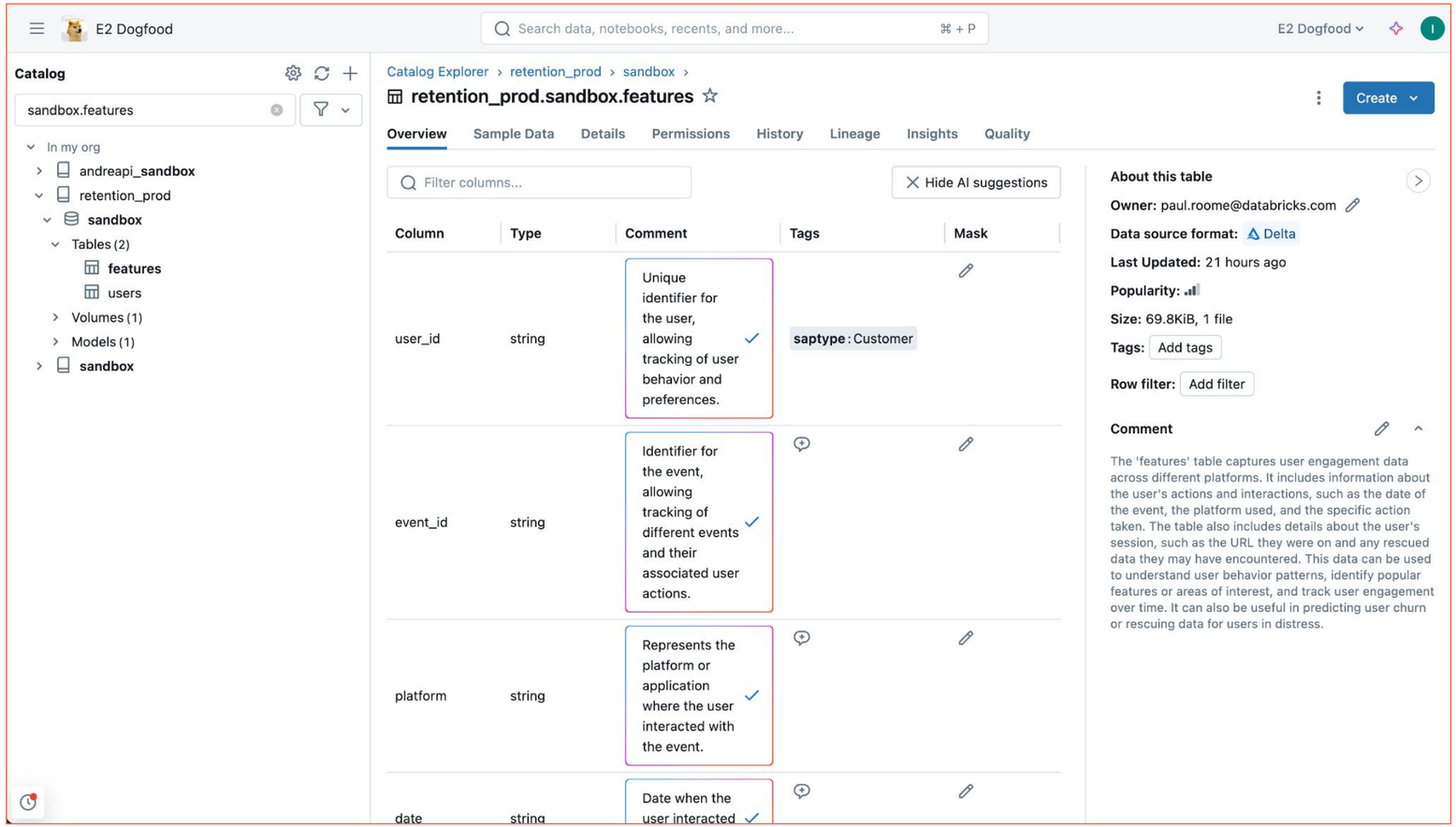
End-to-end visibility into how data flows and consumed in your organization

- Auto-capture runtime data lineage on a Databricks cluster or SQL warehouse
- Leverage common permission model from Unity Catalog
- Lineage across tables, columns, dashboards, Lakeflow Jobs, notebooks, files, external sources, and models



Tagging + AI generated Documentation

- Auto-generate concise and informative **table and column comments** for Unity Catalog
- Document your backlog of data assets with **missing documentation** in minutes
- **Tag** your data for easier discovery and to facilitate tag-based policies

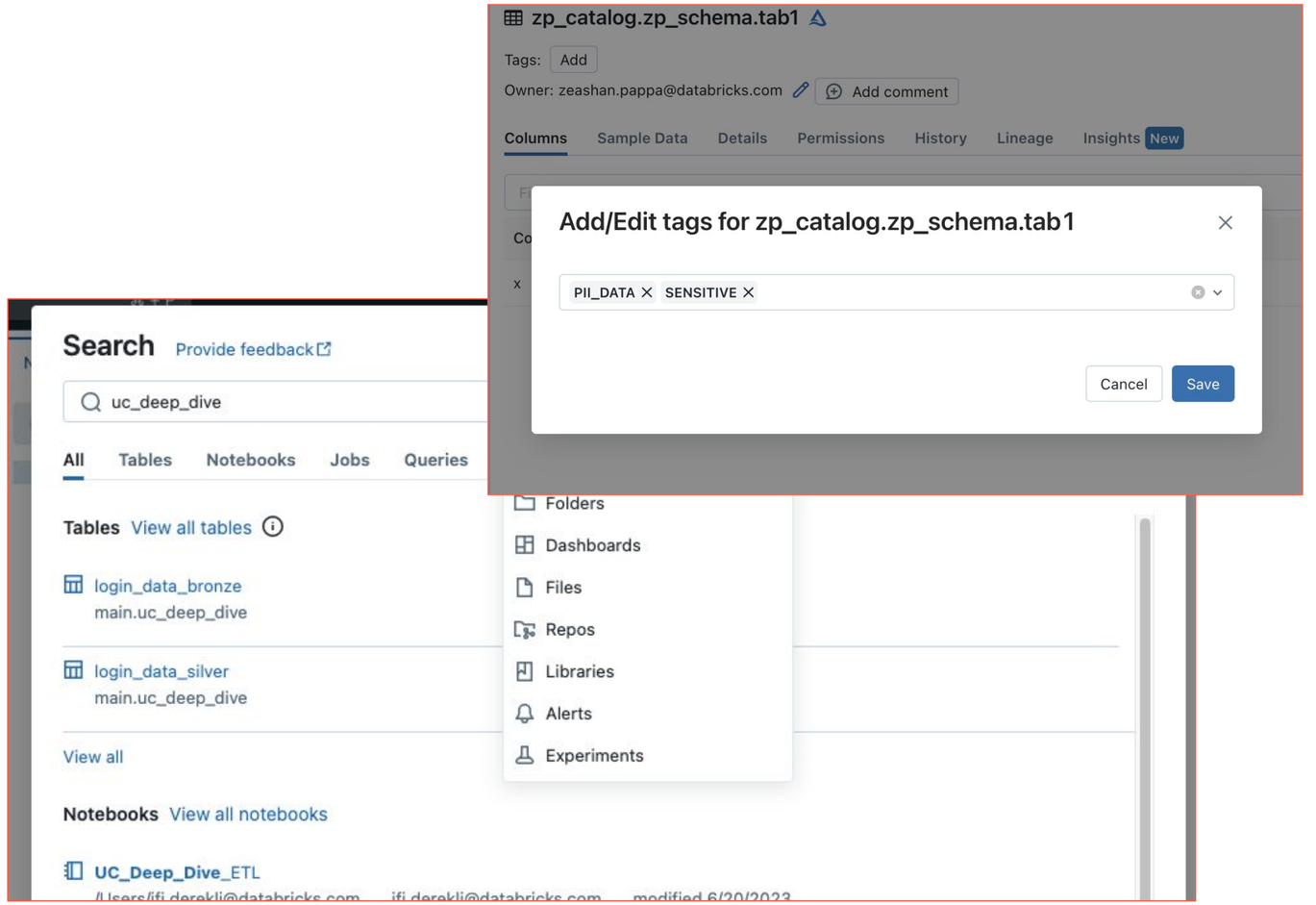


Built in search and discovery

Accelerate time to value with low latency data discovery

- Unified UI to search for data assets stored in Unity Catalog
- Leverage common permission model from Unity Catalog
- Tag Column, Table, Schema, Catalog objects in UC
- Search for objects on tags

- *Recommendation: Use comments and Tag your Data Assets on Ingest*

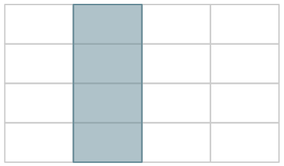


Fine-grained Access Control

Common Use Cases

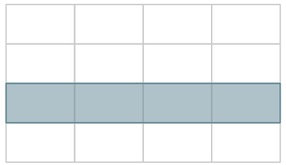
Limit Access to Columns

Omit column values from output



Limit Access to Rows

Omit rows from output



Data Masking

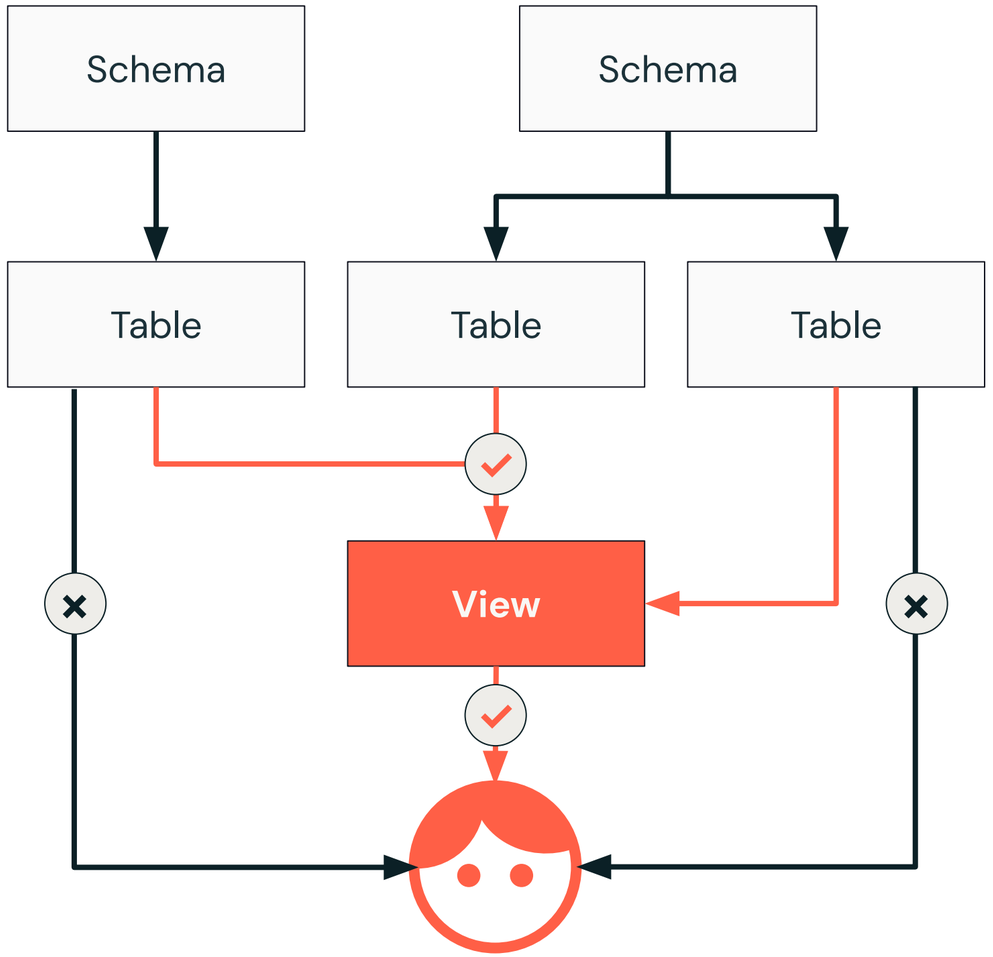
Obscure data

●●●●●●@databricks.com

Can be conditional on a specific user/service principal or group membership through Databricks-provided functions

Fine-grained Access Control

Dynamic Views



Row Level Security and Column Level Masking

Provide differential fine grained access to file based datasets and foreign tables

Only show specific rows

```
CREATE FUNCTION <name> ( <parameter_name >
<parameter_type> .. )
RETURN {filter clause whose output must be a boolean}


CREATE FUNCTION us_filter(region STRING)
RETURN IF(IS_MEMBER('admin'), true, region="US");

ALTER TABLE sales SET ROW FILTER us_filter ON region;
```

Test for group membership

Assign reusable filter to table

Specify filter predicates

Mask or redact sensitive columns

```
CREATE FUNCTION <name> (<parameter_name>,
<parameter_type>, [, <column>...])
RETURN {expression with the same type as the first
parameter}

CREATE FUNCTION ssn_mask(ssn STRING)
RETURN CASE WHEN IS_MEMBER('admin') THEN ssn
            ELSE '*****'
       END;

ALTER TABLE users ALTER COLUMN table_ssn SET MASK ssn_mask;
```

Test for group membership

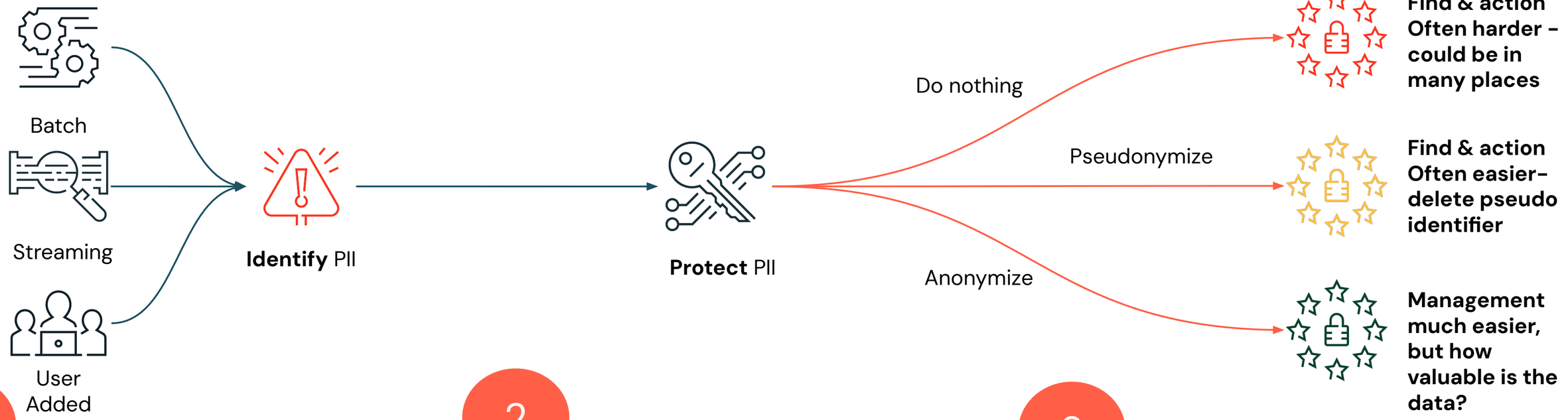
Assign reusable mask to column

Specify mask or function to mask

Data Governance – Key Considerations

What data assets do we have?	**Centralized Catalog**
What data needs to be secured? Who can access the data?	**Centralized ACL**
Who has been accessing the data?	**Auditing**
How is the data flowing across the estate?	**Data Lineage**
How is the data used? Where can we cleanup?	**Data Insights**
How is my data quality over time?	**Lakehouse Monitoring**

Data Privacy – Solutions



1

Identify

- **Data Discovery**
- **Data Classification**
- **Data Mapping**

2

Protect

- **Data Security Measures**
- **Data Handling**
- Consent Management
- Privacy by Design

3

Manage

- **Data Governance**
- **Compliance Management**
- **Monitoring and Auditing**
- Data Subject Rights Management
- Incident Response
- Privacy Impact Assessment

databricks

Unity Catalog

LECTURE

Audit Your Data

System Tables: Object Metadata

Answer questions about the state of objects in the catalog

What tables are in the sales catalog?

```
SELECT table_name
FROM system.information_schema.tables
WHERE table_catalog="sales";
```

Who has access to this table?

```
SELECT grantee, table_name, privilege_type
FROM system.information_schema.table_privileges
WHERE table_name = "login_data_silver";
```

Who last updated the gold tables and when?

```
SELECT table_name, last_altered_by, last_altered
FROM system.information_schema.tables
WHERE table_schema = "churn_gold"
ORDER BY 1, 3 DESC;
```

Who owns this gold table?

```
SELECT table_owner
FROM system.information_schema.tables
WHERE table_catalog = "retail_prod" AND
table_schema = "churn_gold" AND table_name =
"churn_features";
```

System Tables - https://docs.databricks.com/en/admin/system-tables/index.html
Information Schema - https://docs.databricks.com/en/sql/language-manual/sql-ref-information-schema.html

PREVIEW

System Tables: Audit Logs

Near-realtime, see who accessed what, and when

Who accesses this table the most?

```
SELECT user_identity.email, count(*)
FROM system.access.audit
WHERE request_params.table_full_name =
"main.uc_deep_dive.login_data_silver"
AND service_name = "unityCatalog"
AND action_name = "generateTemporaryTableCredential"
GROUP BY 1 ORDER BY 2 DESC LIMIT 1;
```

Who deleted this table?

```
SELECT user_identity.email
FROM system.access.audit
WHERE request_params.full_name_arg =
"main.uc_deep_dive.login_data_silver"
AND service_name = "unityCatalog"
AND action_name = "deleteTable";
```

What has this user accessed in the last 24 hours?

```
SELECT request_params.table_full_name
FROM system.access.audit
WHERE user_identity.email = "ifi.derekli@databricks.com"
AND service_name = "unityCatalog"
AND action_name = "generateTemporaryTableCredential"
AND datediff(now(), event_time) < 1;
```

What tables does this user access most frequently?

```
SELECT request_params.table_full_name, count(*)
FROM system.access.audit
WHERE user_identity.email = "ifi.derekli@databricks.com"
AND service_name = "unityCatalog"
AND action_name = "generateTemporaryTableCredential"
GROUP BY 1 ORDER BY 2 DESC LIMIT 1;
```

Audit Logs - https://docs.databricks.com/en/admin/system-tables/audit-logs.html

PREVIEW

System Tables: Lineage Data

Query upstream and downstream sources in one place

What tables are sourced from this table?

```
SELECT DISTINCT target_table_full_name
FROM system.access.table_lineage
WHERE source_table_name = "login_data_bronze";
```

What user queries read from this table?

```
SELECT DISTINCT entity_type, entity_id,
source_table_full_name
FROM system.access.table_lineage
WHERE source_table_name = "login_data_silver";
```

Table Lineage - https://docs.databricks.com/en/admin/system-tables/lineage.html
Column Lineage - https://docs.databricks.com/en/admin/system-tables/lineage.html#column-lineage-table

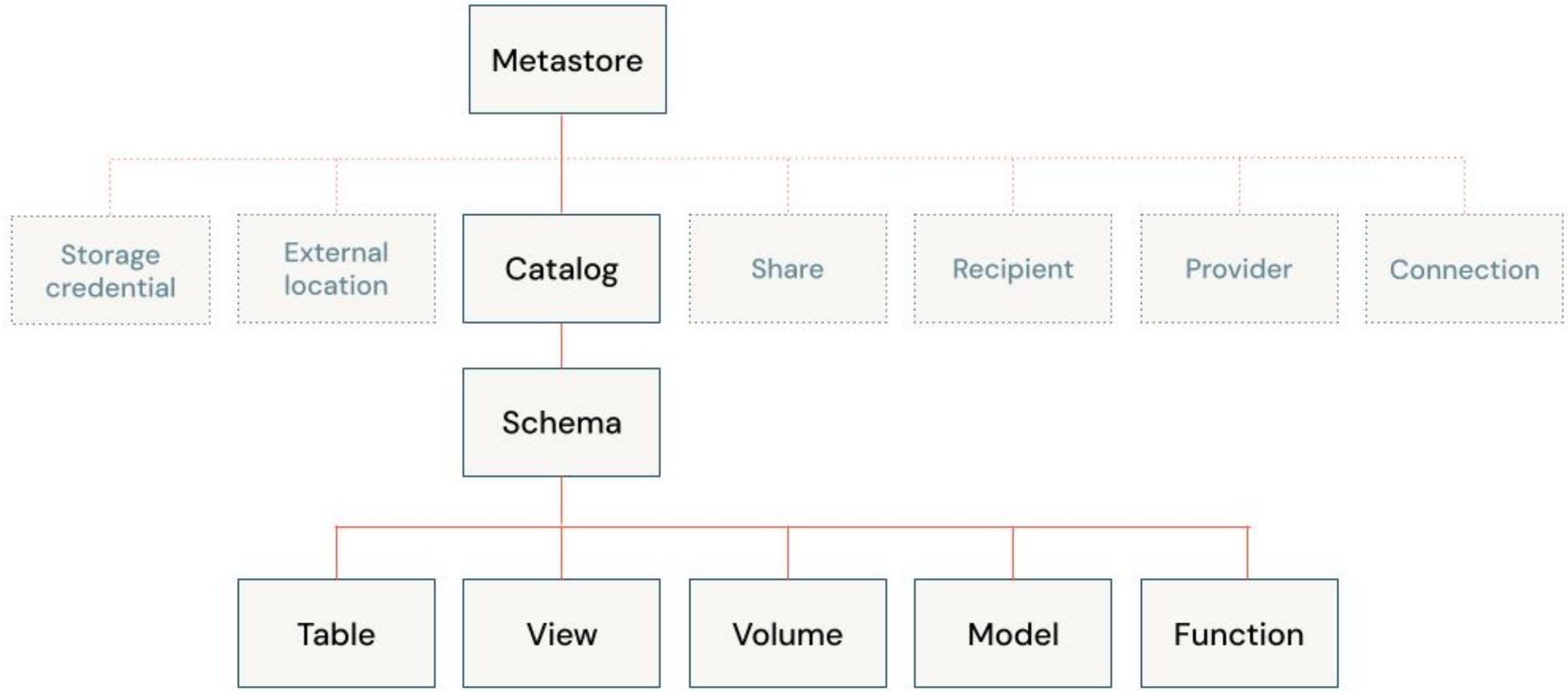
databricks

Unity Catalog

LECTURE

Data Isolation

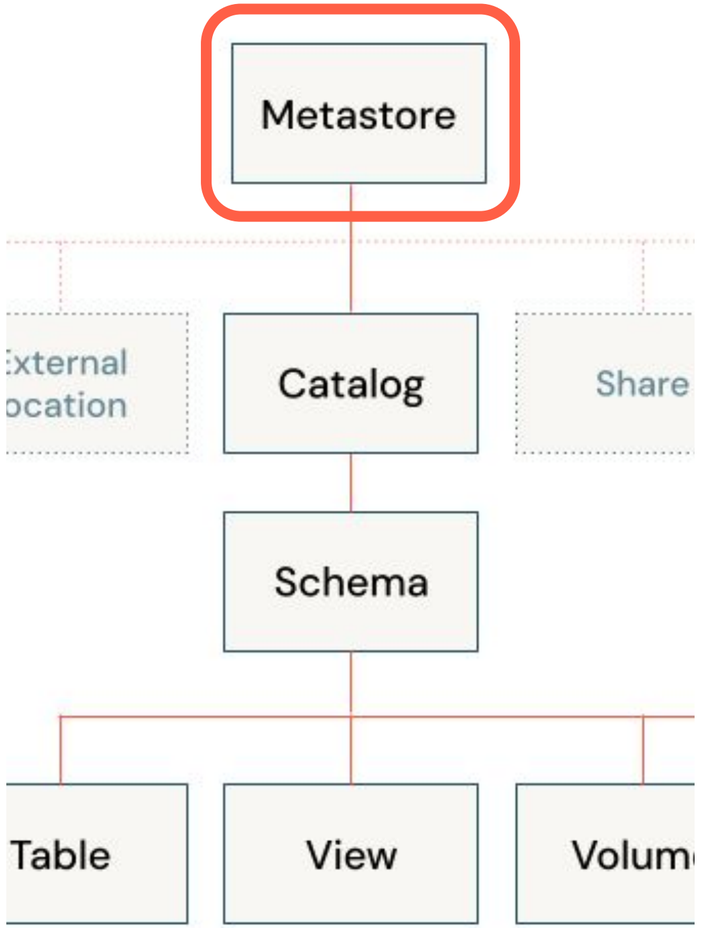
The Data Hierarchy in Unity Catalog



Note: Some securable objects are grayed out to emphasize the hierarchy of objects managed under catalogs

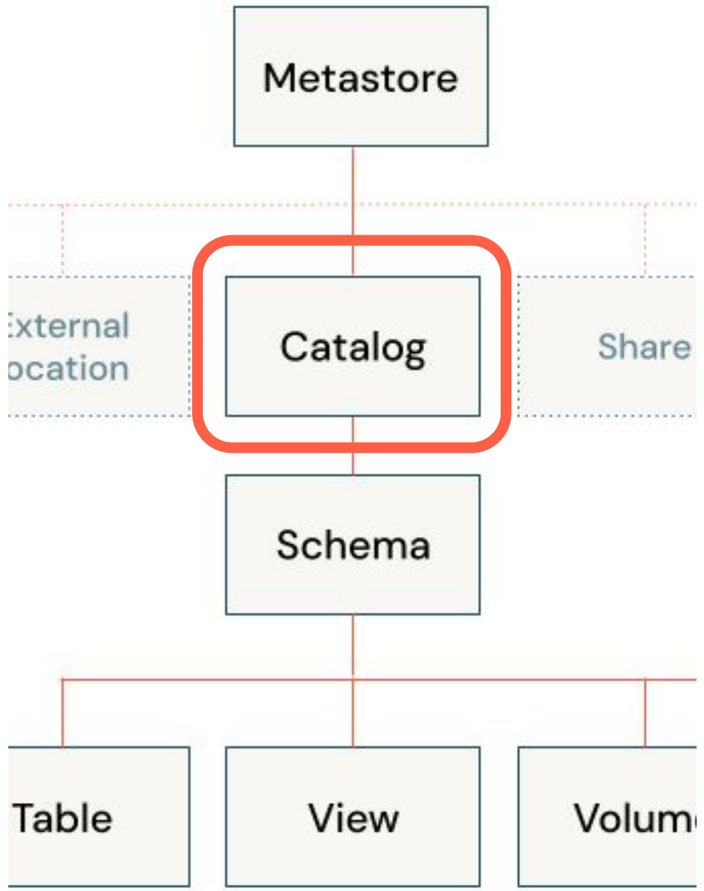
Metastores

- Manage data assets (tables, views, and volumes) and the permissions that govern access to them.
- Admins create **one metastore per region**
- Metastores are mapped to one or more **workspaces within same region**
- Metastores provide regional isolation but are **not intended as units of data isolation**



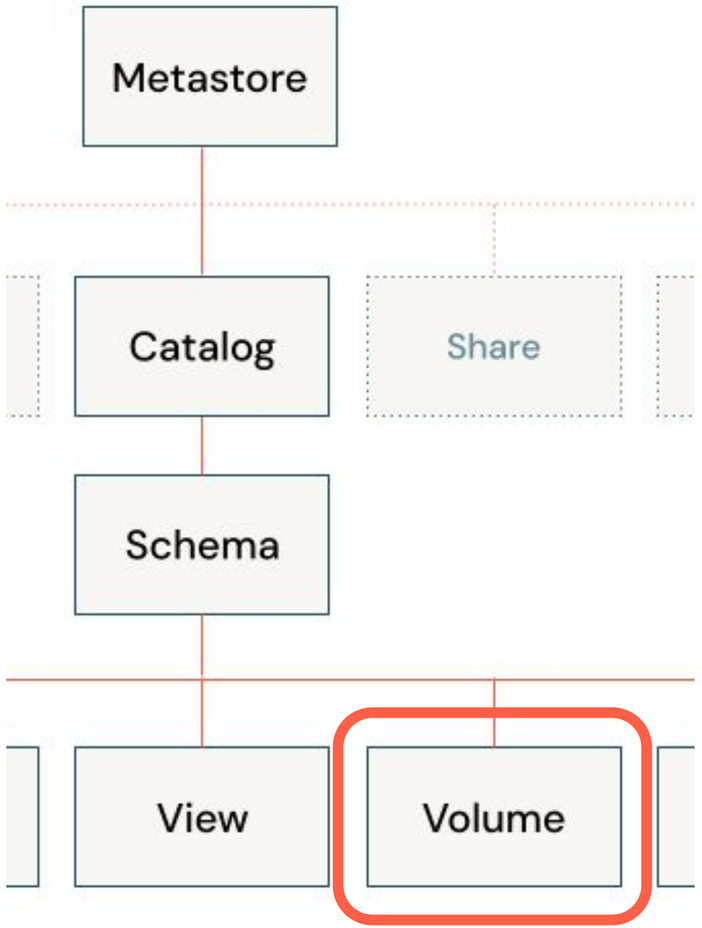
Catalogs

- Intended as the **primary unit of data isolation**
- Often mirror organizational units or software development lifecycle scopes
- Can be stored at the metastore level, or **stored separately from the rest of the parent metastore (preferred)**
- Can be bound to specific workspaces
- Ideal spot to set **inherited permissions**

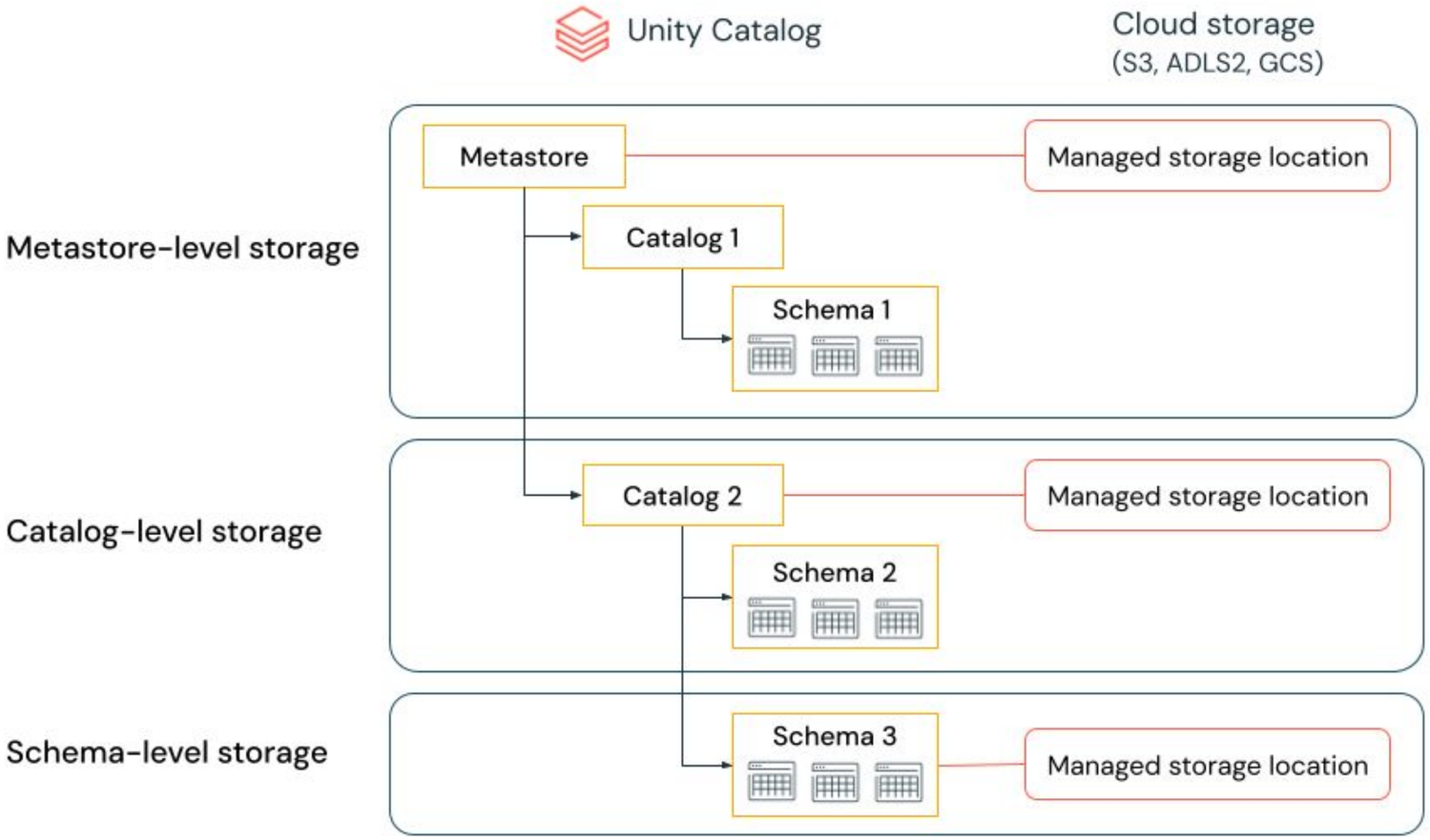


Volumes

- Used for **non-tabular data**
- Any data - structured, semi-structured and unstructured - can be stored here
- Perfect for **libraries, configurations, checkpoint** folders
- Data in Volumes **cannot be registered** as Tables

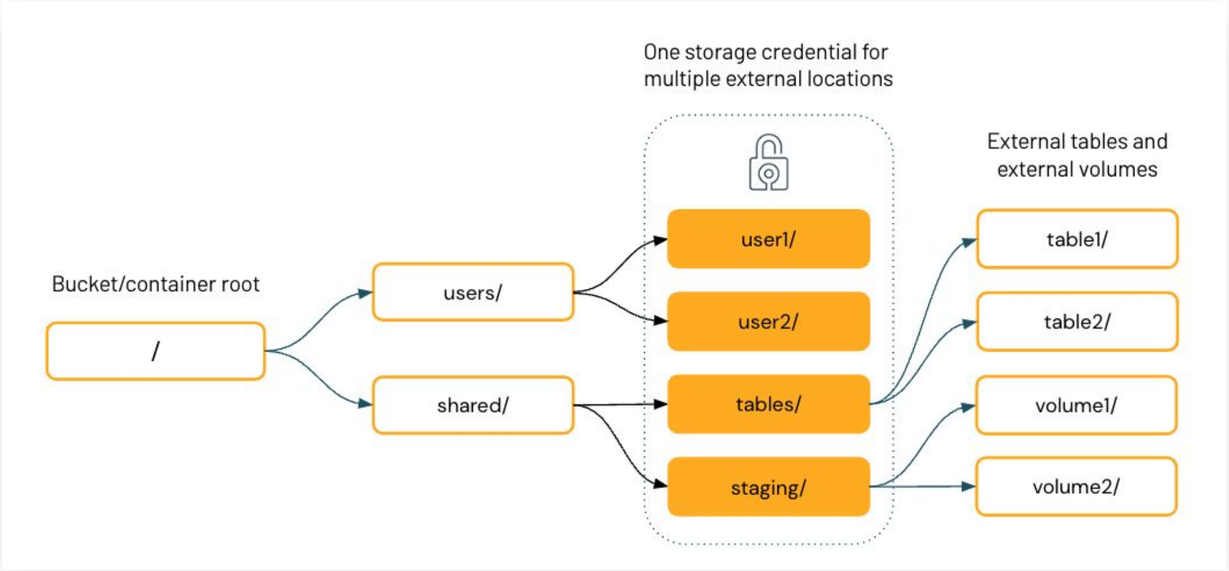


Physically Separating Data



External Locations and Storage Credentials

- External locations are defined as a **path to cloud storage, combined with a storage credential** that can be used to access that location.

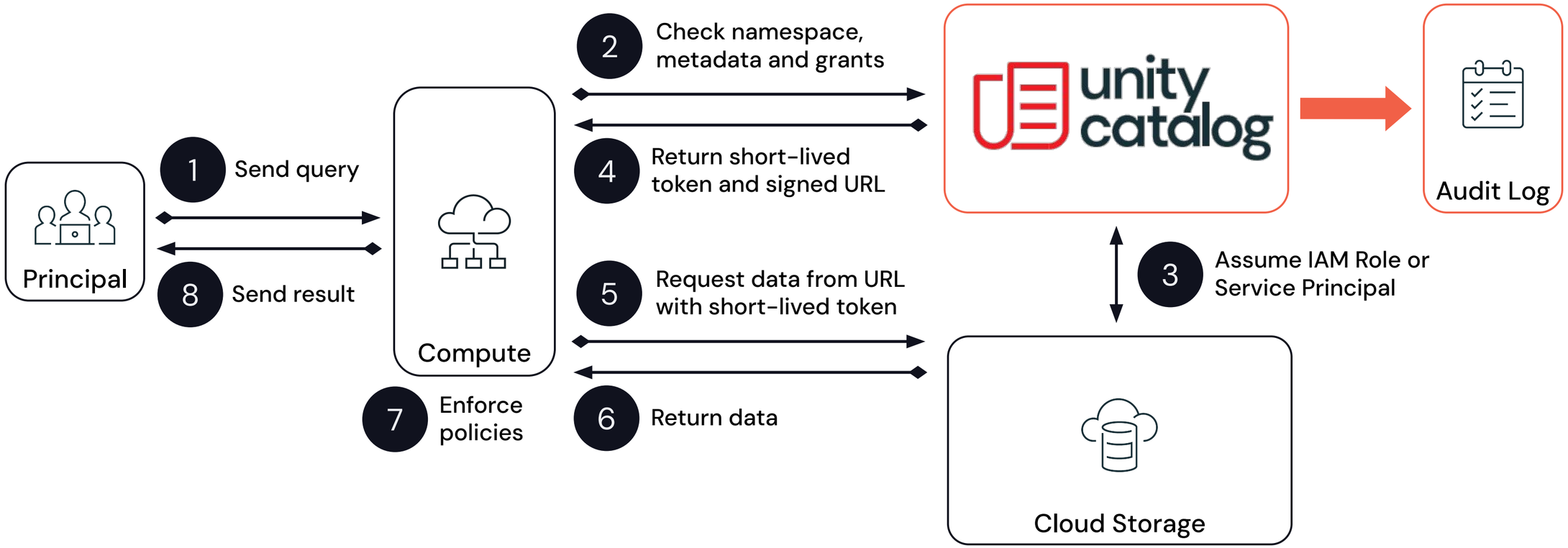


- Allow Unity Catalog to **read and write data on your cloud tenant**

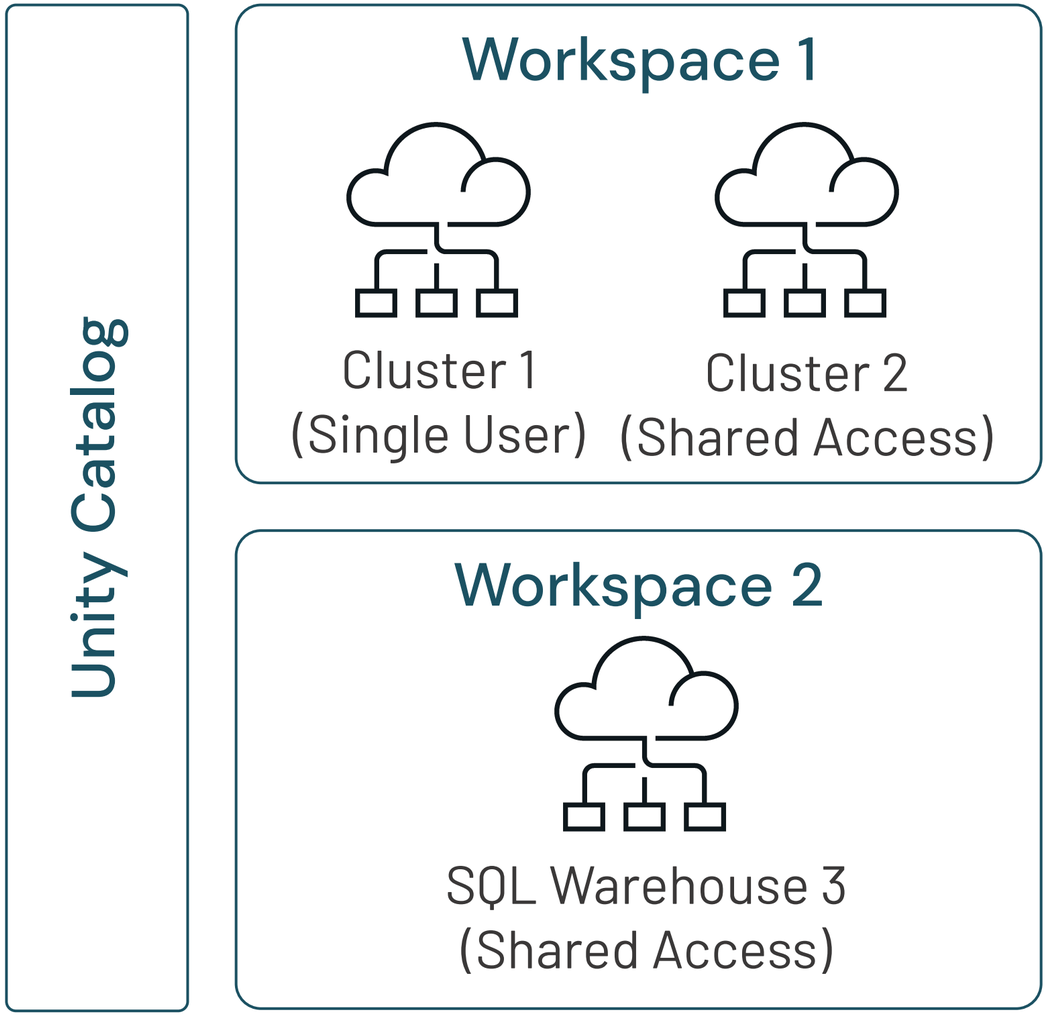
- Use external locations to **register external tables and external volumes** in Unity Catalog.

Accessing Data Securely

The Unity Catalog Security Model



Data and Workload Isolation



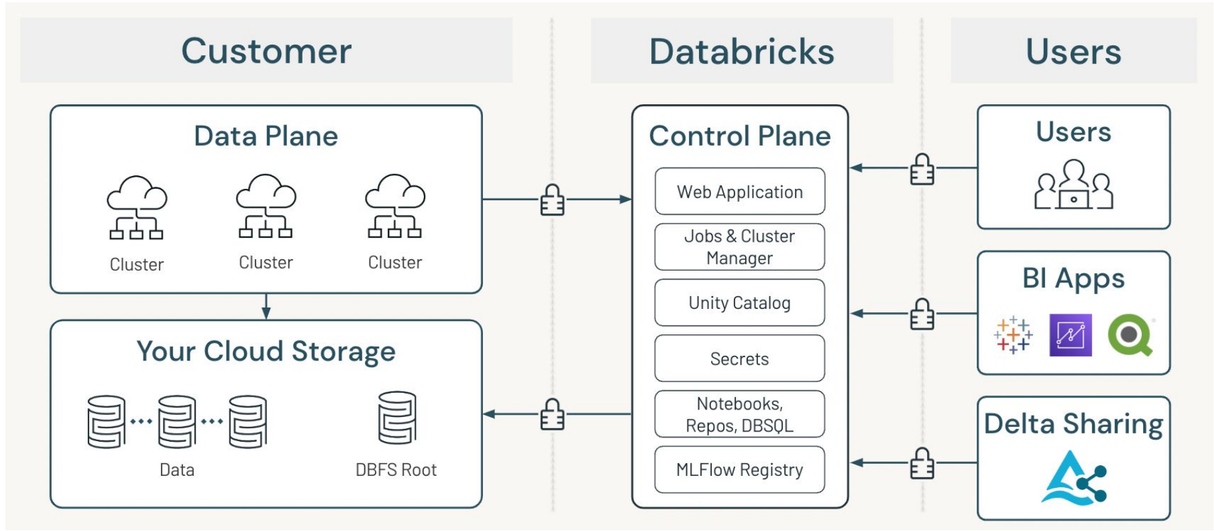
Easily isolate based on your needs

1. **Single-user clusters** provide data protection
2. **Multi-user clusters** that safely support users with different privileges, such as Shared access mode or SQL Warehouses.
3. **Multiple workspaces** isolate groups who won't collaborate

Encryption by Default

All of the following are encrypted by default

- All traffic between control plane and data plane
- All traffic between the user and the control plane
- All storage in the Databricks account
- All EBS volumes
- All traffic to AWS APIs

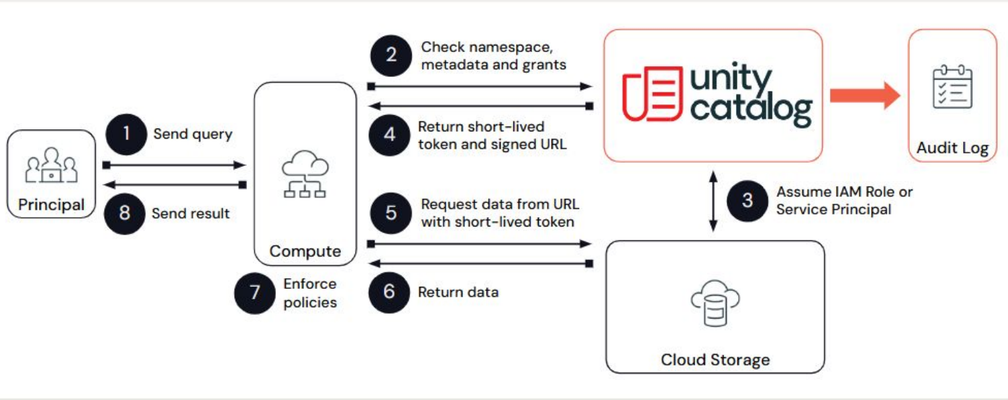


Manage Access to PII

- Control access to storage locations with cloud permissions
- Limit human access to raw data
- Unity Catalog row and column filtering
- Pseudonymize records on ingestion
- Use table ACLs to manage user permissions
- Configure dynamic views for data redaction
- Remove identifying details from demographic views

New Best Practices

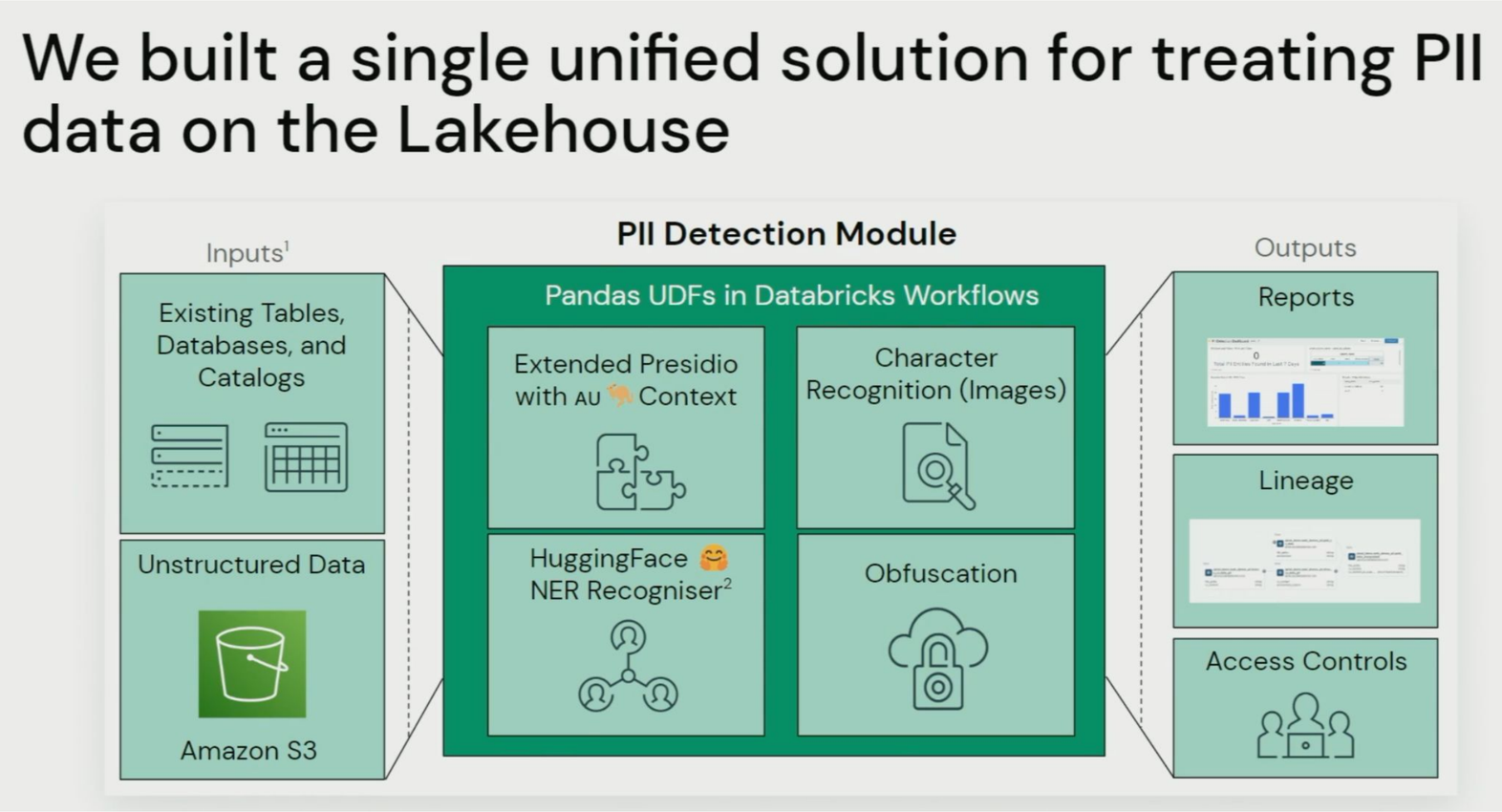
1. **Avoid direct access to object stores**
   a. Less keys => Less leaks
   b. No more credentials in code or secret scopes
   c. UC mediates all access to data
   d. UC maintains audit trail of all access requests
   e. Only infra admins will have object store access



2. **Avoid having data assets in hive metastore**
   a. Hive metastore are not secure enough. All new assets go into UC.
3. **Leverage Workspace–local model registries**
   a. All Model assets will now be stored with schemas in UC.
4. **Avoid DBFS usage**
   a. All unstructured data (checkpoints, libraries, config files etc) should be placed in Volumes

Customer Use Case – SEEK

PII Detection at Scale on the Lakehouse

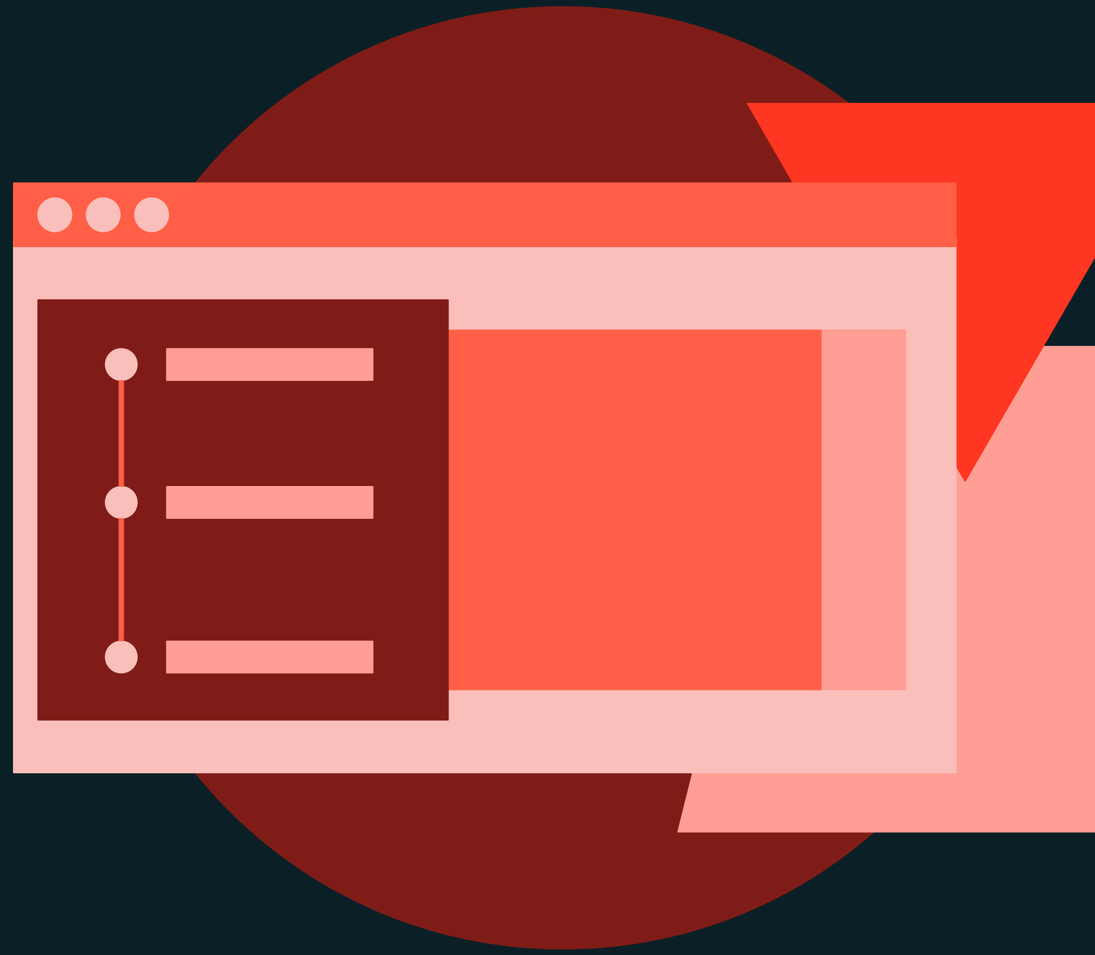


databricks

Unity Catalog

DEMONSTRATION

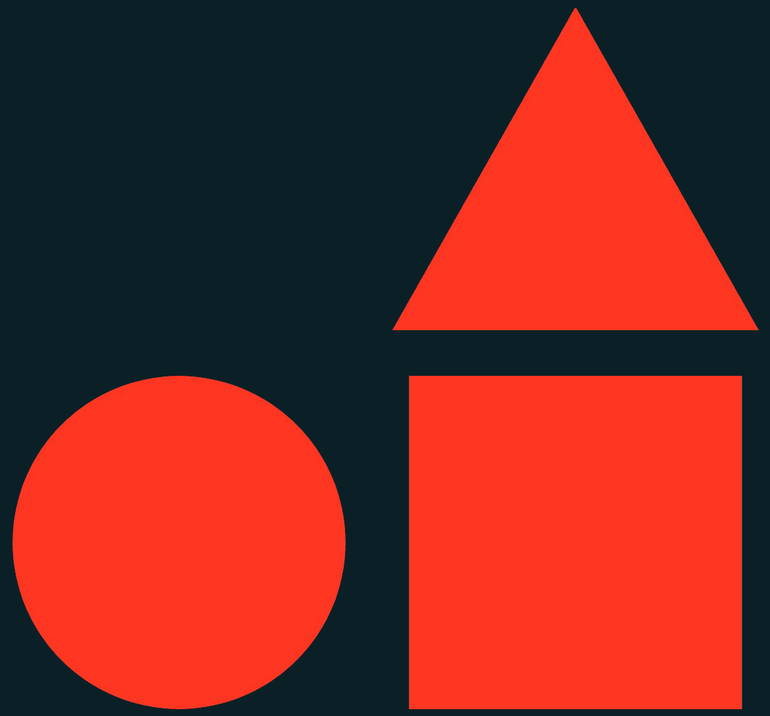
Securing Data in Unity Catalog



DP 1.1 - Securing Data in Unity Catalog

databricks

PII Data Security



Databricks Data Privacy

PII Data Security

Learning Objectives

- Describe and implement Pseudonymization and Anonymization in an automated ETL process
- Describe best practices for handling PII data
- Describe Common Data Protection Techniques

Agenda

PII Data Security
• Pseudonymization & Anonymization
• Summary & Best Practices
• PII Data Security (Demo)

databricks

PII Data Security

LECTURE

Pseudonymization & Anonymization

PII Data Security

Two main data modeling approaches to meet compliance requirements

Pseudonymization

- Protects data at record level
- Re-identification is possible
- Pseudonymised data is still considered PII

Name	John Doe
B_Date	14/04/1987

Name	**User-321**
B_Date	14/04/1987

Anonymization

- Protects entire dataset
- Irreversibly altered
- Non-linkable to original data
- Multiple anonymization methods might be used

Name	John Doe
B_Date	14/04/1987

Name	**********
Age	**20-30**

Pseudonymization

Overview of the approach

- Switches original data point with pseudonym for later **re-identification**
- Only authorized users will have access to keys/hash/table for re-identification
- Protects datasets on **record level** for machine learning
- A pseudonym is still considered to be personal data according to the GDPR
- Two main pseudonymization methods: **hashing** and **tokenization**

Pseudonymization

Method: **Hashing**

- Apply SHA or other hash to all PII
- Add random string "salt" to values before hashing
- Databricks secrets can be leveraged for obfuscating salt value
- Leads to some increase in data size
- Some operations will be less efficient

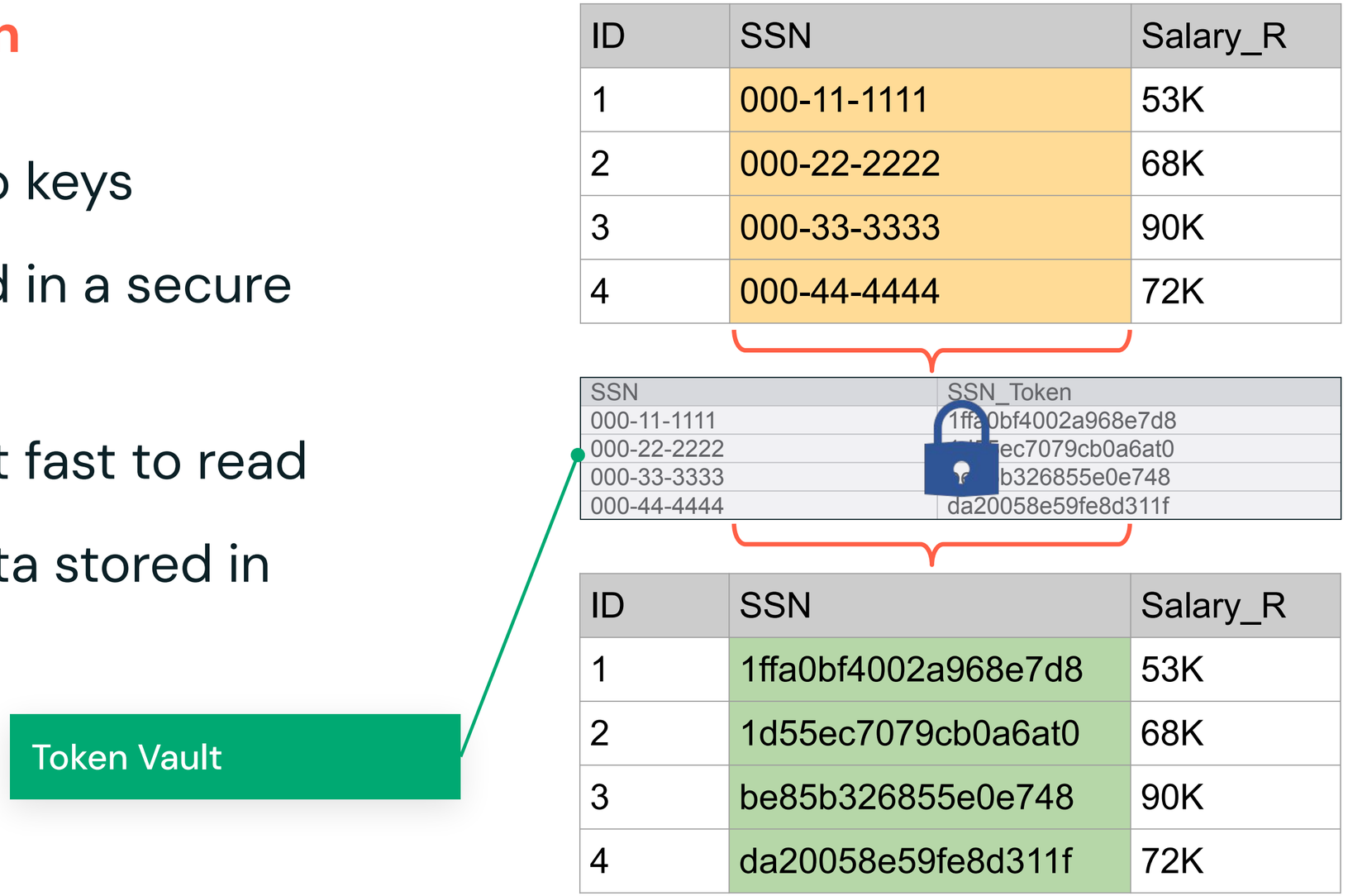
ID	SSN	Salary_R
1	000-11-1111	53K
2	000-22-2222	68K
3	000-33-3333	90K
4	000-44-4444	72K

ID	SSN	Salary_R
1	1ffa0bf4002a968e7d8	53K
2	1d55ec7079cb0a6at0	68K
3	be85b326855e0e748	90K
4	da20058e59fe8d311f	72K

Pseudonymization

Method: **Tokenization**

- Converts all PII to keys
- Values are stored in a secure lookup table
- Slow to write, but fast to read
- De-identified data stored in fewer bytes



Anonymization

Overview of the approach

- Protects **entire dataset** (tables, databases or entire data catalogues) mostly for Business Intelligence
- Personal data is **irreversibly altered** in such a way that a data subject can no longer be identified directly or indirectly
- Usually a combination of more than one technique used in real-world scenarios
- Two main anonymization methods: **data suppression** and **generalization**

Anonymization Methods

Method: **Data Suppression**

- Exclude columns with PII from views
- Remove rows where demographic groups are too small
- Use dynamic access controls to provide conditional access to full data

Source Table		
ID	SSN	Salary_R
1	000-11-1111	53K
2	000-22-2222	68K
3	000-33-3333	90K

View with no PII	
ID	Salary_R
1	53K
2	68K
3	90K

Anonymization Methods

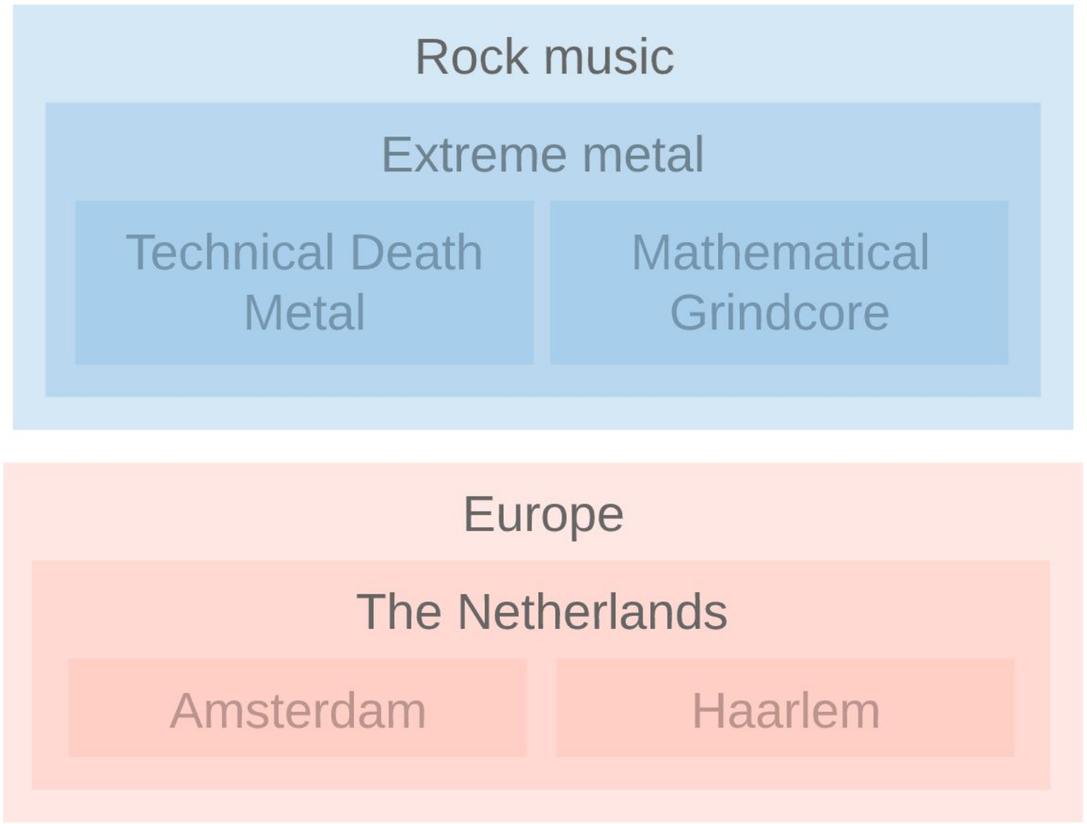
Method: **Generalization**

- Categorical generalization
- Binning
- Truncating IP addresses
- Rounding

Anonymization Methods

Method: Generalization → **Categorical Generalization**

- Removes precision from data
- Move from specific categories to more general
- Retain level of specificity that still provides insight without revealing identity



Anonymization Methods

Method: Generalization → **Binning**

- Identify meaningful divisions in data and group on boundaries
- Allows access to demographic groups without being able to identify individual PII
- Can use domain expertise to identify groups of interest

ID	Department	BirthDate
1	IT	28/09/1997
2	Sales	13/02/1976
3	Marketing	02/04/1985
4	Engineering	19/12/2002

ID	Department	Age_Range
1	IT	20-30
2	Sales	40-50
3	Marketing	30-40
4	Engineering	20-30

Anonymization Methods

Method: Generalization → **Truncating IP addresses**

IP addresses need special anonymization rules;

- Rounding IP address to /24 CIDR
- Replace last byte with 0
- Generalizes IP geolocation to city or neighbourhood level

ID	IP	IP_Truncated
1	10.130.176.215	10.130.176.**0/24**
2	10.5.56.45	10.5.56.**0/24**
3	10.208.126.183	10.208.126.**0/24**
4	10.106.62.87	10.106.62.**0/24**

Anonymization Methods

Method: Generalization → **Rounding**

- Apply generalized rounding rules to all number data, based on required precision for analytics
- Example:
  - Integers are rounded to multiples of 5
  - Values less than 2.5 are rounded to 0 or omitted from reports
  - Consider suppressing outliers

ID	Department	Age_Range	Salary
1	IT	20-30	1245.4
2	Sales	40-50	1300
3	Marketing	30-40	1134

ID	Department	Age_Range	Salary_R
1	IT	20-30	1200
2	Sales	40-50	1300
3	Marketing	30-40	1100

databricks

PII Data Security

LECTURE

Summary & Best Practices

Common Data Protection Techniques

Technique	Description	Example	Example Use Case	Advantages	Disadvantages	Protection
Data Masking	Conceals original data with modified content. Dynamic masking in Databricks allows defining masking rules.	gXXX.dXXXX@databricks.com	Protecting sensitive data while maintaining operational utility.	- Preserves data format. - Some information can remain, while increasing privacy.	- The distribution of data is changed. - Data might be recoverable using information from related columns. - Cannot be used to link data.	Low to Moderate
Pseudo-anonymization	Replace values with pseudonyms or other artificial values.	charles.darwin@databricks.com	Medical studies, where the patient needs to be tracked over time, but identity must be protected.	- Statistical distribution is preserved. - Possible to link multiple datasets.	- Possible to infer actual values from distribution of the pseudo-anonymized values - Data might be recoverable using information from related columns. - Linkage to the original values must be stored securely.	Low to Moderate
Hashing	Transforming data into irreversible string	cf35dd9aafff028e5dcc…	Storage of passwords	- Secure and irreversible - Data linkage possible - Preserves data distribution	- Possible to infer actual values from distribution of the hash values - Data retrieval not possible	Moderate to High
Column Encryption	Encrypts data at the column level. In Databricks, sensitive data is encrypted before storage.	0582e62c284e8ad8d…	Securing specific sensitive data columns.	- High security - Obscures individual values and distributions	- Requires key management. - Increases data size substantially. - Data linkage is difficult. - Distribution is changed.	High
Tokenization	Replacing sensitive data with tokens.	d08fa46b7a79e1201d…	Credit card transactions	- Tokens can replace real data for operations.	- Requires robust tokenization system. - All data is recoverable if the token system is compromised.	High

Best Practices for handling PII data

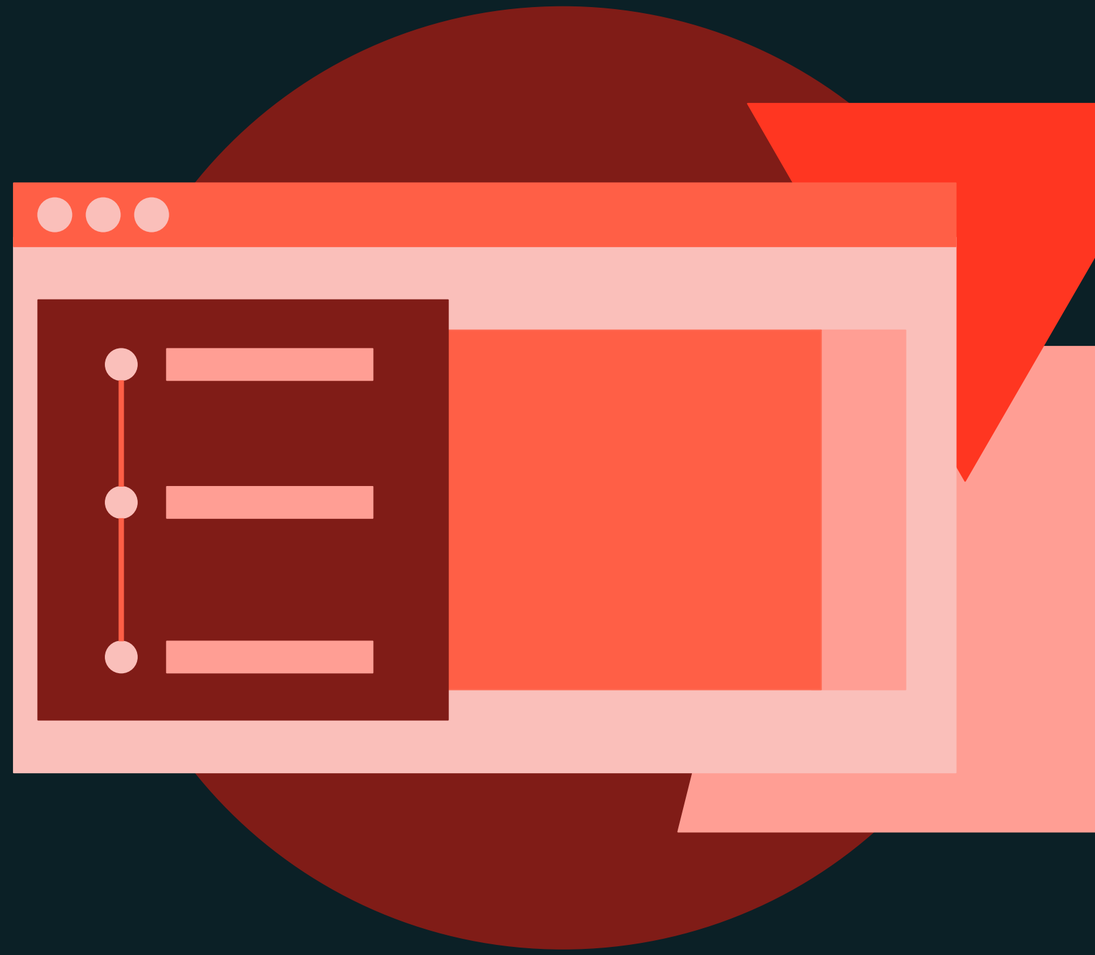
1. Having no PII is always better than having PII
2. Anonymisation is always > than pseudo-anonymization is always > than cleartext
3. Always try to maintain a healthy paranoia around the protections you have applied
4. Always apply the 3 facts rule
5. Always consider how datasets could be combined to allow for re-identification
6. Always ensure your data teams are appropriately trained on applicable privacy laws
7. Not all PII is created equal
8. Conduct PIAR reviews
9. Always isolate environments that process PII
10. Your life will be easiest if you isolate the environment that protects PII

databricks

PII Data Security

DEMONSTRATION

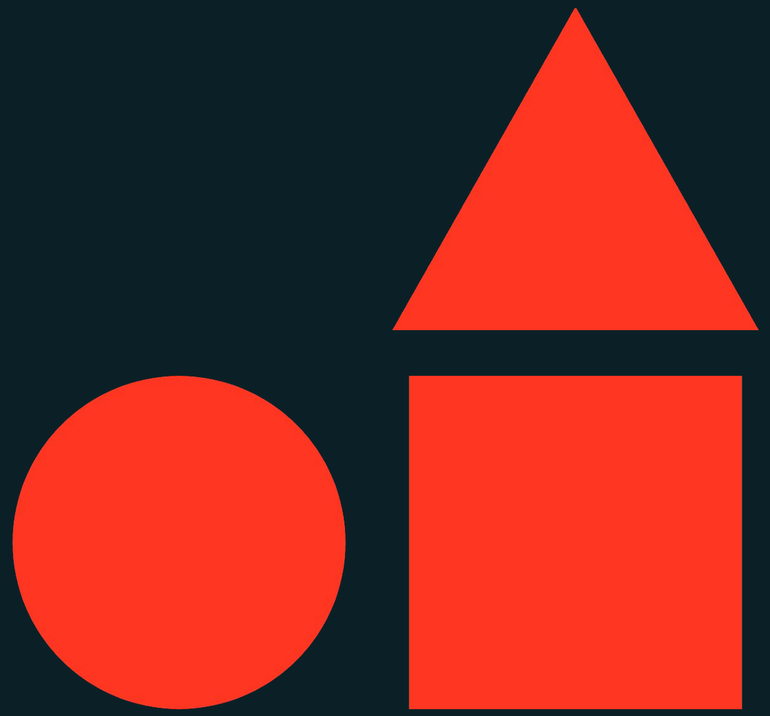
PII Data Security



DP 1.2 - PII Data Security

databricks

Streaming Data and CDF



Databricks Data Privacy

Streaming Data and CDF

Learning Objectives

- Explain how CDF is enabled and how it works
- Discuss why ingesting from CDF can be beneficial for streaming data
- Articulate multiple strategies for using Streaming data to create CDF
- Discuss how CDF addresses past difficulties propagating updates and deletes
- Explain the use case of CDF for removing data in downstream tables
- Describe various methods of recording important data changes
- Discuss how CDF can be leveraged to ensure deletes are committed fully

Agenda

Streaming Data and CDF
• Capturing Changed Data
• Deleting Data in Databricks
• Processing Records from CDF and Propagating Changes (Demo)
• Propagating Changes with CDF Lab (Lab)

databricks

Streaming Data and CDF

LECTURE

Capturing Changed Data

Streaming Data and Data Changes

Updates and Deletes in streaming data

- In Structured Streaming, a data stream is treated as a table that is being **continuously appended**. Structured Streaming expects to work with data sources that are **append only**.
- Changes in existing data (updates and deletes) breaks this expectation!
- We need a **deduplication logic** to identify updated and deleted records.

- **Note:** Delta transaction logs tracks files than rows. Updating a single row will point to a new version of the file.

Solution 1: Ignore Changes

Prevent re-processing by ignoring deletes, updates and overwrites

Ignore Deletes

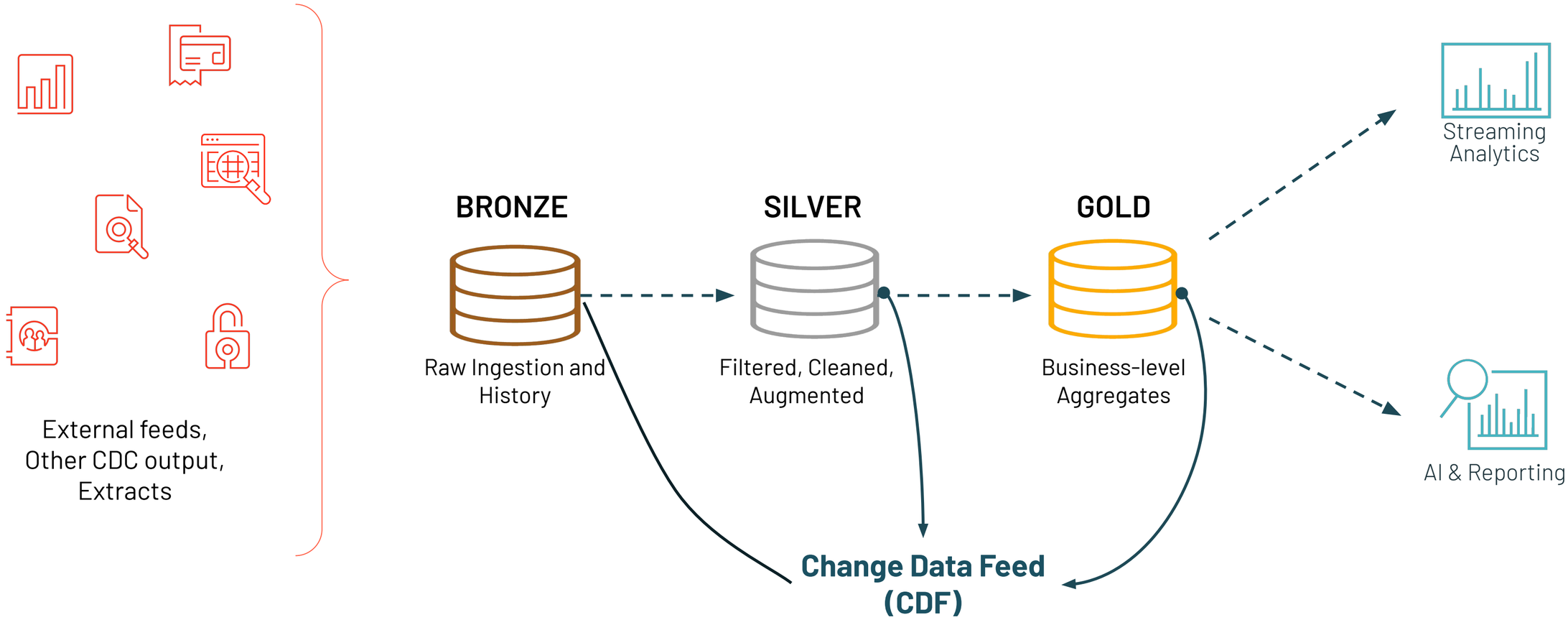
- Ignore transactions that delete data at partition boundaries
- No new data files are written with full partition removal
- ```
  spark.readStream.format("delta")
  .option("ignoreDeletes", "true")
  ```

Ignore Changes

- Allows stream to be executed against Delta table with upstream changes
- Must implement logic to avoid processing duplicate records
- Subsumes ignoreDeletes
- ```
  spark.readStream.format("delta")
  .option("ignoreChanges", "true")
  ```

Solution 2: Change Data Feeds (CDF)

Propage incremental changes to downstream tables

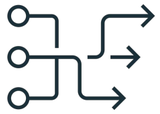


What Delta Change Data Feed Does for You

Benefits and use cases of CDF

Improve ETL Pipelines

Process less data during ETL to increase efficiency of your pipelines by processing only row-level changes



Unify batch and streaming

Common change format for batch and streaming updates, appends, and deletes

BI on your data lake

Incrementally update the data supporting your BI tool of choice

Meet regulatory needs

Full history available of changes made to the data, including deleted information

Comparison CDF vs CDC

Feature	Change Data Feed (CDF)	Change Data Capture (CDC)
Scope	Specific to **Delta Lake tables**	General concept applicable across systems
Functionality	Tracks row-level changes within Delta tables	Captures data changes for synchronization across systems
Implementation	Enabled on Delta tables; uses `**_change_data**` folder and `**table_changes**` function.	Implemented using DLT and APIs like **APPLY CHANGES**
Efficiency	Processes only changed rows for operations.	Synchronizes incremental changes from source databases
Use Case	Tracking changes within Databricks	Capturing changes from external sources

How Does Delta CDF Work?

Sample CDF data schema

Original Table (v1)

	PK	B
	A1	B1
	A2	B2
	A3	B3

1

Change data (Merged as v2)

	PK	B
	A1	B1
	A2	Z2
	~~A3~~	~~B3~~
	A4	B4

2

Change Data Feed Output

PK	B	Change Type	Time	Version
A2	B2	**Preimage**	12:00:00	2
A2	Z2	**Postimage**	12:00:00	2
A3	B3	Delete	12:00:00	2
A4	B4	Insert	12:00:00	2

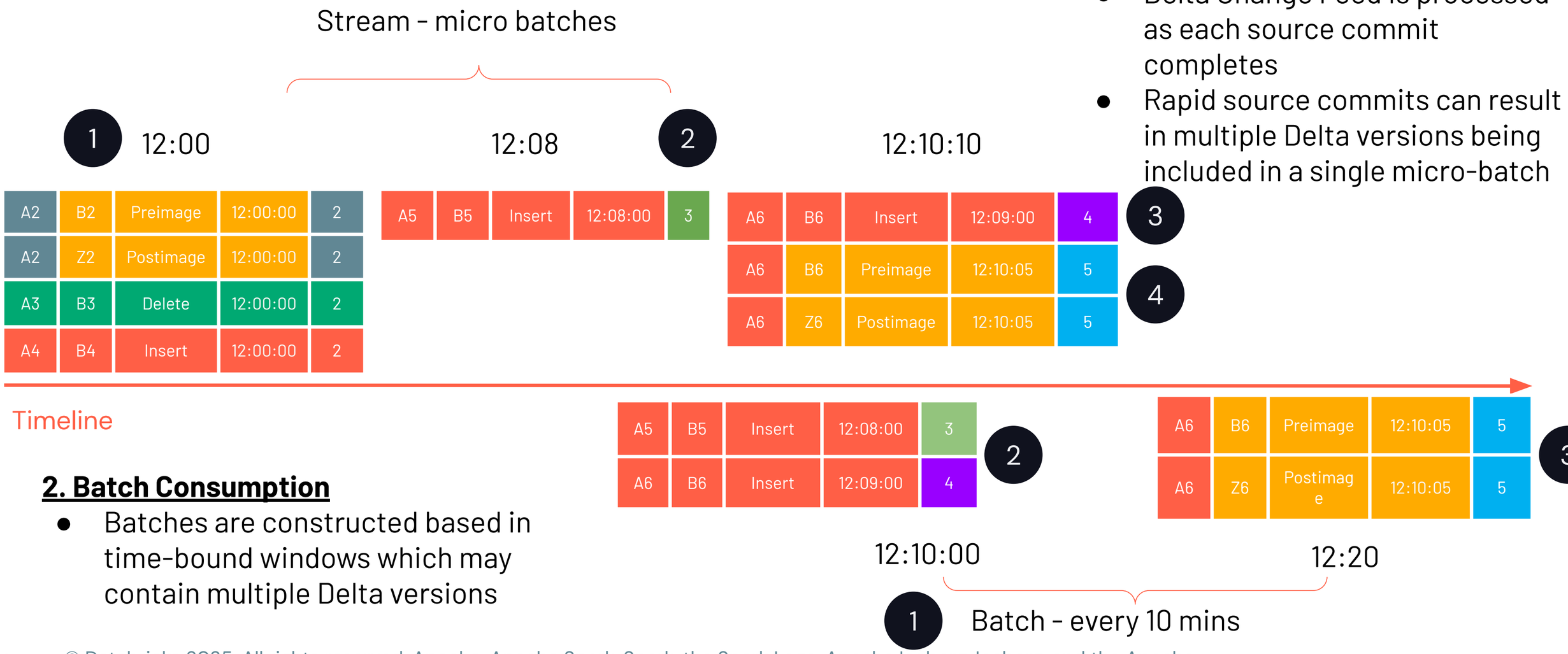
3

4

A1 record did not receive an update or delete.
So it will not be output by CDF.

Consuming the Delta CDF

1. **Stream-based Consumption**
   - Delta Change Feed is processed as each source commit completes
   - Rapid source commits can result in multiple Delta versions being included in a single micro-batch



2. **Batch Consumption**
   - Batches are constructed based in time-bound windows which may contain multiple Delta versions

CDF Configuration

Important notes for CDF configuration

- CDF is **not** enabled by default. It can be enabled;
  - At table level: `ALTER TABLE myDeltaTable SET TBLPROPERTIES (delta.enableChangeDataFeed = true)`
  - For all new tables: `set spark.databricks.delta.properties.defaults.enableChangeDataFeed = true;`
- Change feed can be read by;
  - Version
  - Timestamp

Change Tables Function

Tracks row-level changes between versions of a Delta table

- It returns a log of changes to a Delta Lake table with Change Data Feed enabled, including inserts, updates, and deletes
- Leverages metadata columns:
  - ***_change_type***: Specifies the type of change (insert, delete, update_preimage, update_postimage
  - ***_commit_version***: The commit version associated with the change
  - ***_commit_timestamp***: The timestamp of the change
- Syntax:

```
table_changes ( table_str, start [, end ] )
```

databricks

Streaming Data and CDF

LECTURE

Deleting Data in Databricks

Data Deletion in Databricks

Data deletion needs special attention!

- Companies need to handle data deletion requests carefully to main compliance with **privacy regulations** such as GDPR and CCPA.
- PII for users need to be effectively and efficiently handled in Databricks, including deletion.
- These operations usually handled in pipelines separate from ETL pipelines.
- CDF data can be used to propagate deletion action to downstream table.

Recording Important Data Changes

Using commit messages

- Delta Lake supports **arbitrary commit messages** that will be recorded to the Delta transaction log and viewable in the table history. This can help with later **auditing**.
- Commit messages can be;
  - Set at global level
  - Can be specified as part of write operation. For example, data insertion can be labeled based on processing type; manual, automated.

Propagating Data Deletion with CDF

How can CDF be used for propagating deletes?

- Data deletion requests can be streamlined with automated triggers using Structured Streaming.
- CDF can be separately leveraged to **identify records** need to be deleted or modified in downstream tables.

Note: When Delta Lake's history and CDF features are implemented, deleted PII values are still present in older versions of the data.

- Using **Vacuum** command will physically delete PIIs
- **Deleting at a partition boundary** will make the whole process more efficient

CDF Retention Policy

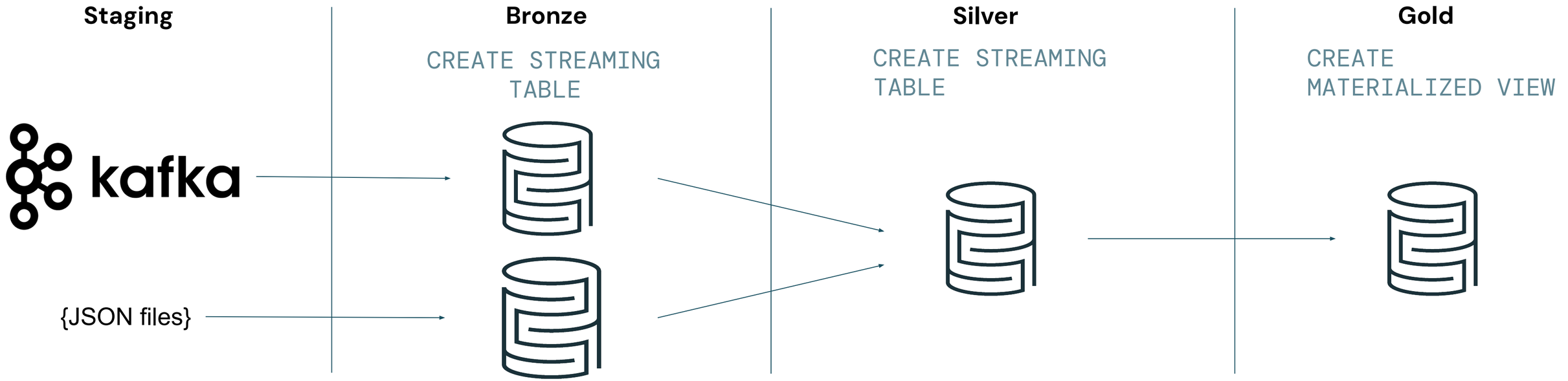
Important notes for CDF configuration

- File deletion will not actually occur until we VACUUM our table!
- CDF records follow the same retention policy of the table. VACUUM command deletes CDF data.
- By default, the Delta engine will prevent VACUUM operations with less than 7 days of retention. To manually run VACUUM for these files;
  - Disable Spark's retention duration check (retentionDurationCheck.enabled)
  - Run VACUUM with DRY RUN to preview files before permanently removing them
  - Run VACUUM with RETAIN 0 HOURS!

Can I perform DML on a streaming Table? (i.e. GDPR)

Example GDPR use case

Using streaming tables for ingestion and materialized views after



Limited Retention

Configurable Retention

Correction / GDPR

Materialized views automatically reflect changes made to inputs

DML works on streaming tables only

Updates, deletes, inserts and merges on streaming tables.

- **Ensure compliance** for retention periods on a table.

```
DELETE FROM my_live_tables.users
WHERE updated < current_time() - INTERVAL 3 years;
```

- **Scrub PII** from data in the lake.

```
UPDATE my_live_tables.users
SET email = hash(email, salt)
WHERE id = 2;
```

- **Append new data**.

```
INSERT INTO my_live_tables.users
VALUES (3, hash(email, salt), current_time());
```

User

id	email	updated
~~1~~	~~****@gmail.com~~	~~01/01/2019~~
2	****@company..	02/01/2024
3	****@hotmail...	02/01/2025

Materialized Views don't support DML

MVs retain data's state from last refresh

- **No direct updates allowed** Insert, Update, Delete, Merge Into.
- **Precomputed results** for faster data retrieval.

```
CREATE OR REFRESH MATERIALIZED VIEW users_by_date AS
  SELECT count(*), updated
  FROM my_live_tables.users
  GROUP BY (updated);
```

- **Refresh mechanism** rather than updating with every query.
  - Manually or in DLT execution

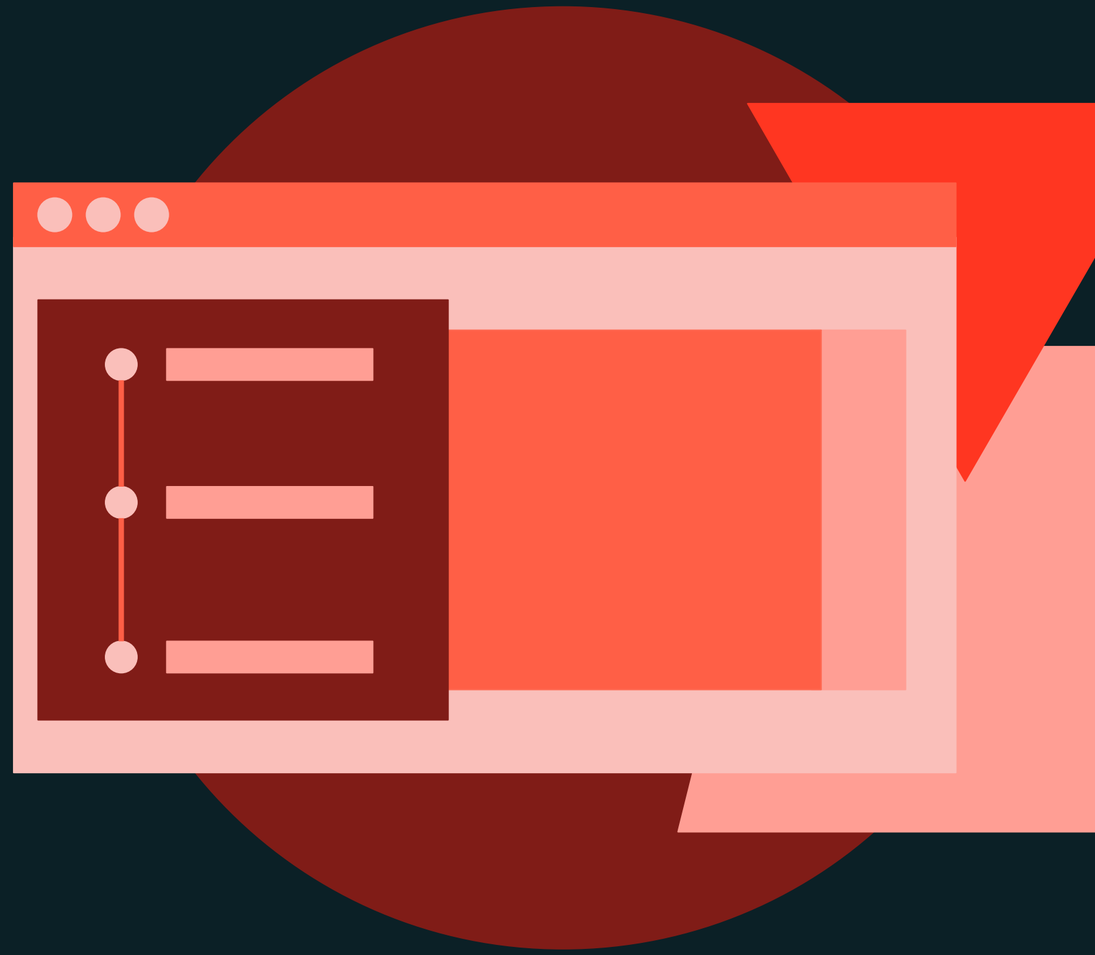
```
REFRESH MATERIALIZED VIEW users_by_date FULL;
```

databricks

Streaming Data and CDF

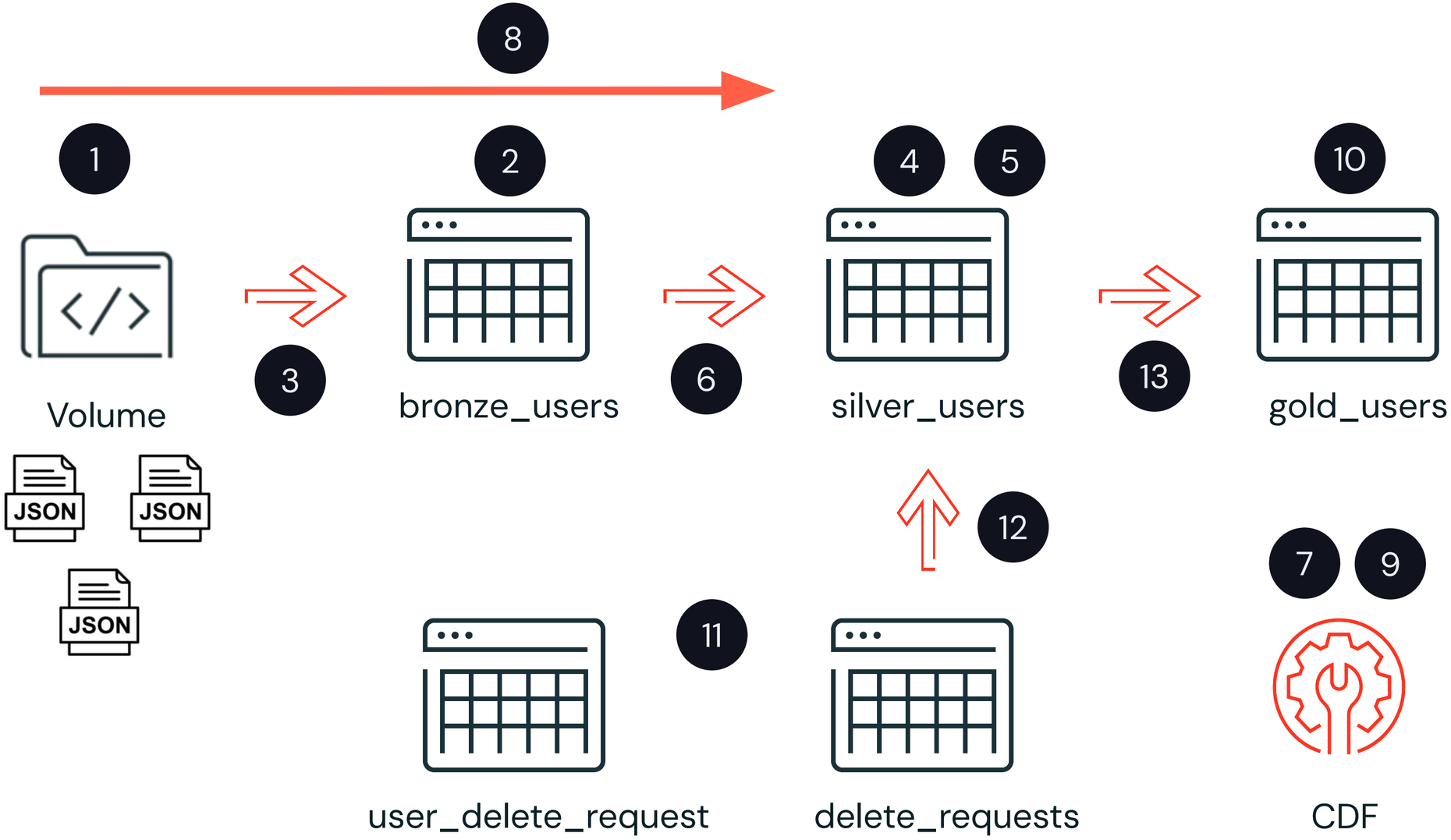
DEMONSTRATION

Processing Records from CDF and Propagating Changes



DP 1.3 - Processing Records from CDF and Propagating Changes

Demo Overview



Steps

1. Create **Volume**
2. Create **Bronze** Users Table
3. Stream from **Volume to Bronze**
4. Create **Silver** Users Table
5. Enable **CDF**
6. Stream and Upsert from **Volume to Silver**
7. Check **CDF in Stream**
8. Stream **more data**
9. Check CDF with **table_changes**
10. Create **Gold** Users Table
11. Create **Requests tables**
12. **Delete** Users in Silver
13. **Propagate** Changes in Gold Table

databricks

Streaming Data and CDF

LAB EXERCISE

Propagating Changes with CDF Lab

DP 1.4L - Propagating Changes with CDF Lab

databricks