databricks

Data Ingestion with Lakeflow Connect

Databricks Academy

Course Learning Objectives

- Describe Lakeflow Connect as a scalable and simplified solution for data ingestion into Databricks from a variety of sources.
- Review the benefits of Delta tables and the Medallion architecture.
- Demonstrate how to ingest data from cloud object storage into Delta tables using CREATE TABLE AS, COPY INTO, and Auto Loader, including capturing input file metadata in Bronze layer tables.
- Explain how rescued columns are used during ingestion to manage malformed records.
- Illustrate techniques for ingesting and flattening semi structured JSON data from cloud storage.
- Describe available options for ingesting data from enterprise systems using Lakeflow Connect Managed Connectors.
- Discuss alternative ingestion methods such as MERGE INTO, Delta Sharing and Databricks Marketplace.

Agenda

Course Sections
• Introduction to Data Engineering in Databricks
• Cloud Storage Ingestion with LakeFlow Connect Standard Connectors
• Enterprise Data Ingestion with LakeFlow Connect Managed Connectors
• Ingestion Alternatives

Course Prerequisites (REQUIRED)

Fundamental Knowledge of the Databricks Platform

- Course: **Get Started with Databricks for Data Engineering**

  OR

- Knowledge of Databricks Workspaces, Apache Spark, Delta Lake and the Medallion architecture, Unity Catalog Data Objects

Course Prerequisites (REQUIRED)

Fundamental Knowledge of the Databricks Platform

- Course: **Get Started with Databricks for Data Engineering**

  OR

- Knowledge of Databricks Workspaces, Apache Spark, Delta Lake and the Medallion architecture, Unity Catalog Data Objects

Experience working with a variety of file types

- Parquet
- CSV
- JSON
- TXT and others

Course Prerequisites (REQUIRED)

Fundamental Knowledge of the Databricks Platform

- Course: **Get Started with Databricks for Data Engineering**

  OR

- Knowledge of Databricks Workspaces, Apache Spark, Delta Lake and the Medallion architecture, Unity Catalog Data Objects

Experience working with a variety of file types

- Parquet
- CSV
- JSON
- TXT and others

Proficiency in SQL/Python and Databricks Notebooks

- Experience coding with SQL and Python
- Familiarity with executing code in Databricks notebooks

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!

Introduction to Data Engineering in Databricks

Data Ingestion with with Lakeflow Connect

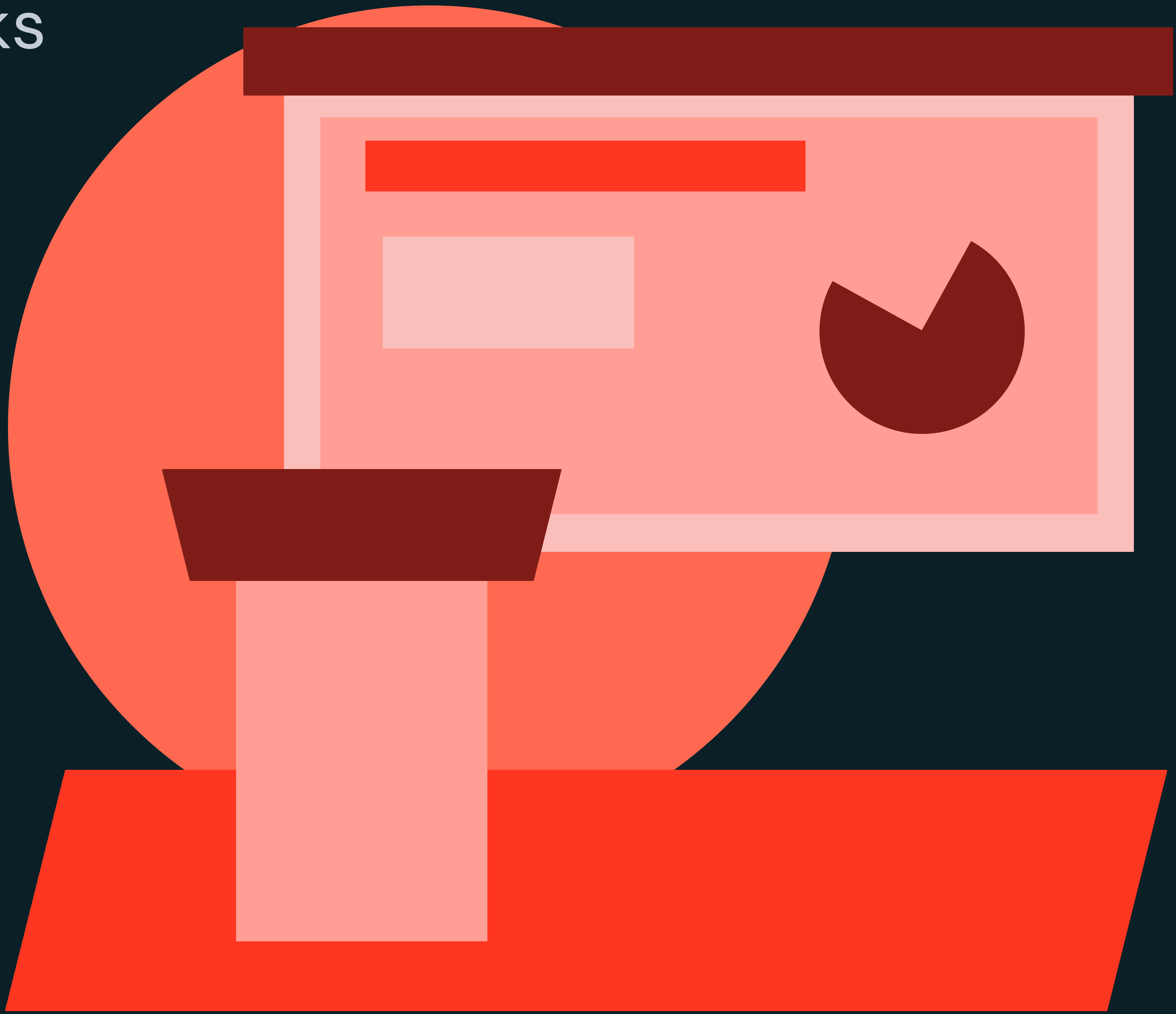
Agenda

Section Overview – Introduction to Data Engineering in Databricks
• Data Engineering in Databricks
• What is Lakeflow Connect?
• Delta Lake Review
• Exploring the Lab Environment

Introduction to Data Engineering in Databricks

LECTURE

Data Engineering in Databricks



Data Engineering in Databricks

databricks LAKEFLOW

UNIFIED DATA ENGINEERING FOR THE DATA INTELLIGENCE PLATFORM

CONNECT	DECLARATIVE PIPELINES (Previously DLT)	JOBS (Previously Workflows)
Efficient ingestion connectors	Accelerated ETL development	Reliable orchestration for analytics and AI

INDUSTRY LEADING DATA PROCESSING ENGINE
(Apache Spark + Structured Streaming)

UNIFIED GOVERNANCE	Unity Catalog
OPTIMIZED STORAGE	DELTA LAKE Parquet ICEBERG

Data Engineering in Databricks

This course will focus on data ingestion with **LakeFlow Connect**

databricks LAKEFLOW

UNIFIED DATA ENGINEERING FOR THE DATA INTELLIGENCE PLATFORM

CONNECT

Efficient ingestion connectors

DECLARATIVE PIPELINES
(Previously DLT)

Accelerated ETL development

JOBS
(Previously Workflows)

Reliable orchestration for analytics and AI

INDUSTRY LEADING DATA PROCESSING ENGINE
(Apache Spark + Structured Streaming)

UNIFIED GOVERNANCE	Unity Catalog
OPTIMIZED STORAGE	DELTA LAKE, Parquet, ICEBERG

databricks

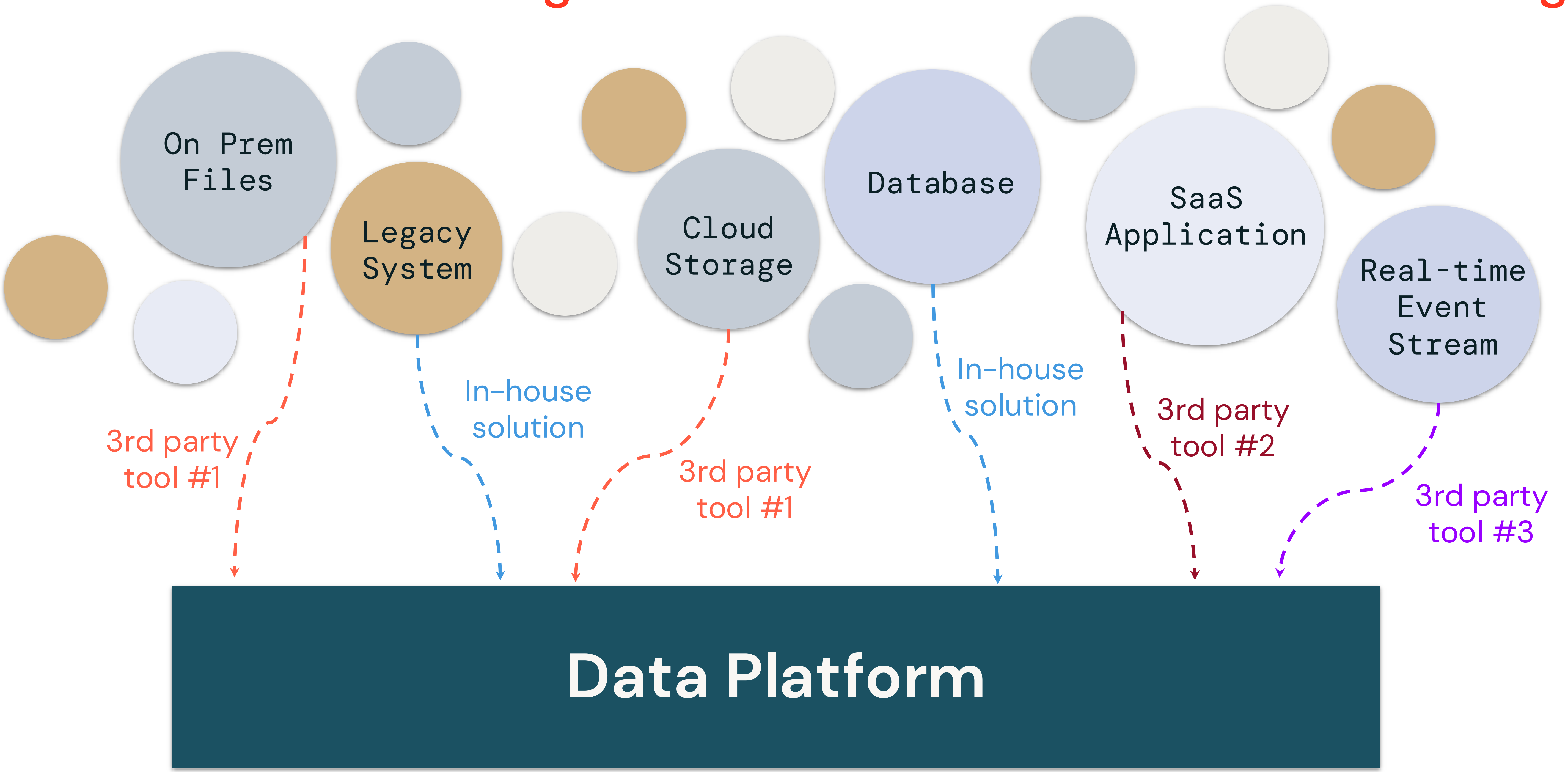
Introduction to Data Engineering in Databricks

LECTURE

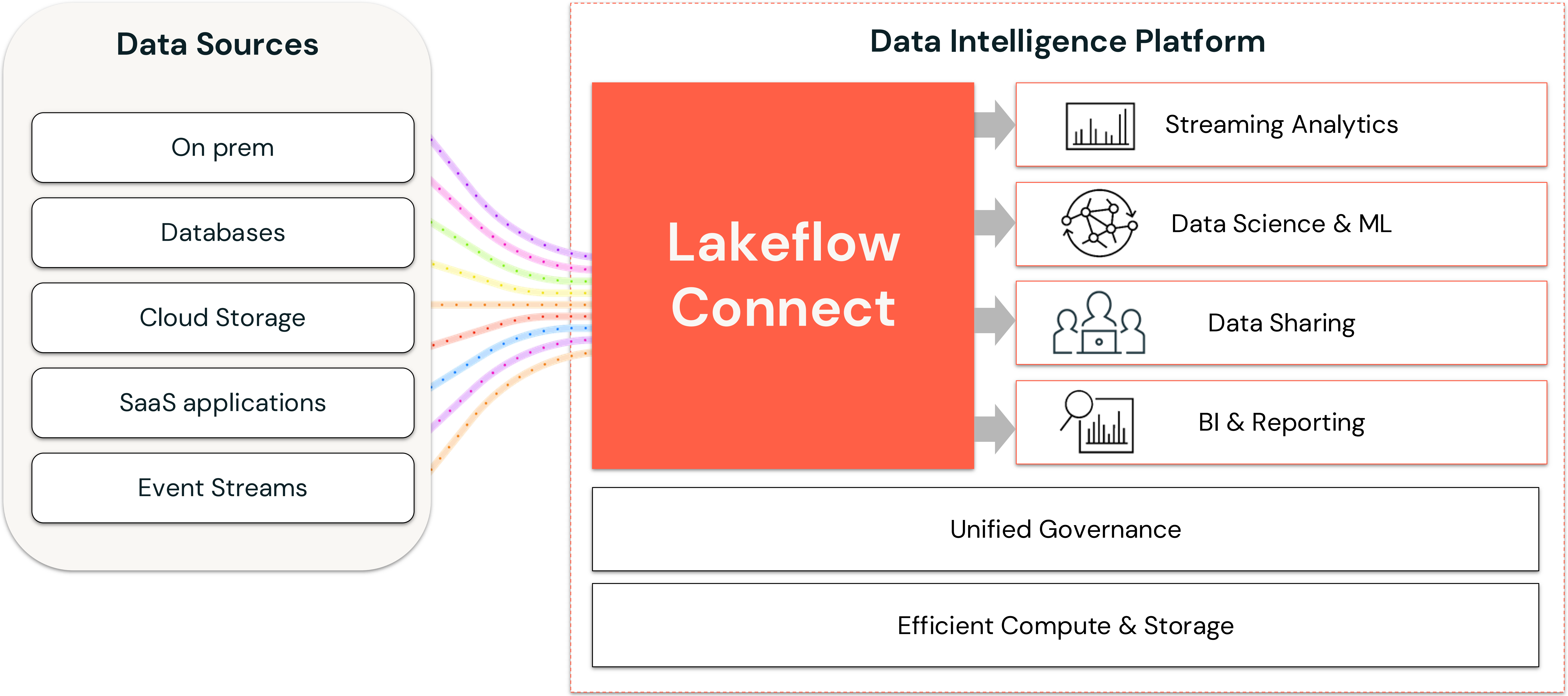
What is Lakeflow Connect?

What is Lakeflow Connect?

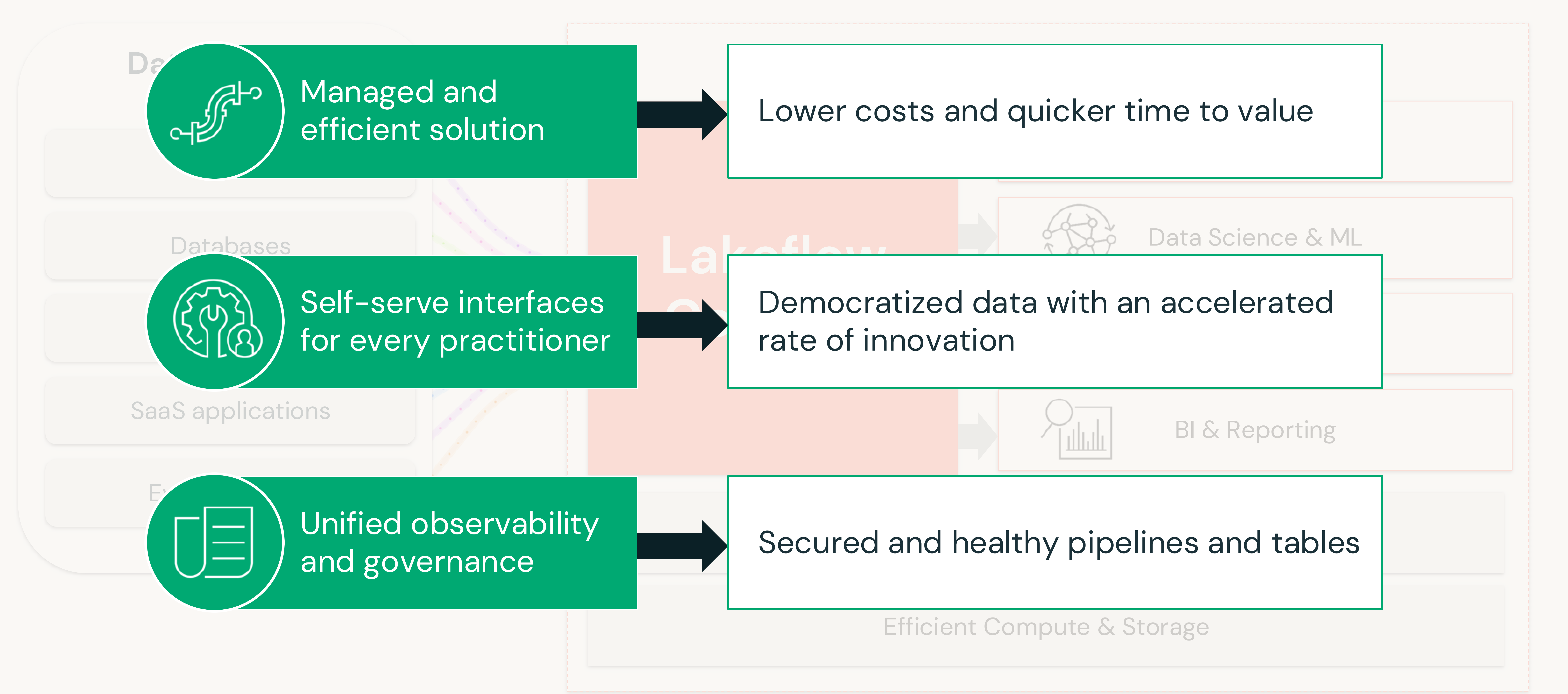
Organizations are Resorting to a Patchwork of Solutions for Data Ingestion



Lakeflow Connect is all Ingestion



Built-in connectors for the Data Intelligence Platform



What is Lakeflow Connect?

Lakeflow Connect - Connectors Overview

Upload Files

- Uploading local files to Databricks
  - Upload a file to a volume
  - Create a table from a local file

Standard Connectors

Ingest data into the lakehouse using various sources and methods:

Supported Sources:

- Cloud Object Storage
- Kafka
- Other Sources

Ingestion Methods:

- Batch
- Incremental Batch
- Streaming

Managed Connectors

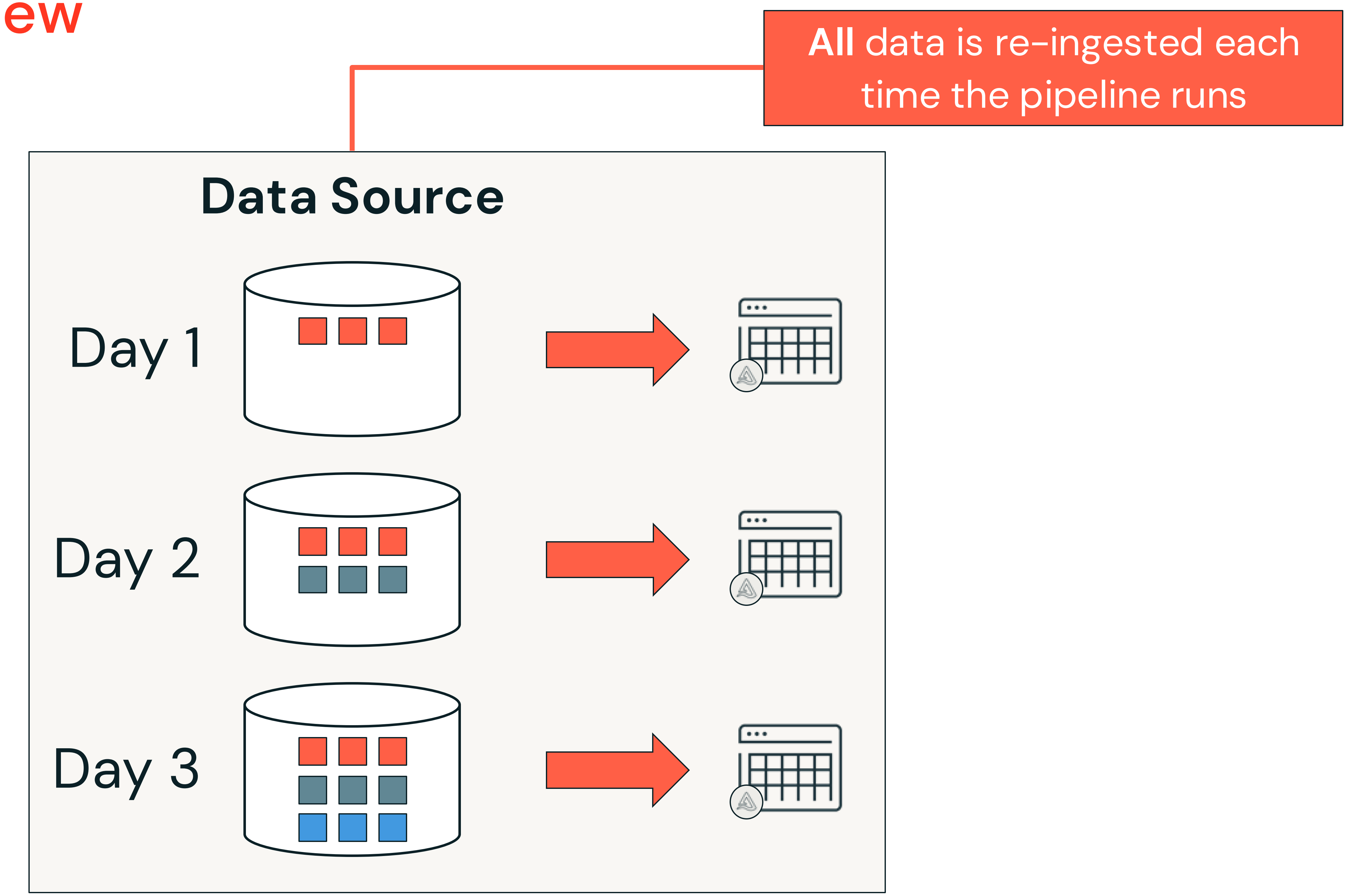
Ingesting data into the lakehouse from:

- Software as a Service (SaaS) applications
- Databases

Leverage efficient **incremental reads** and **writes** to make data ingestion faster, scalable, and more cost-efficient

What is Lakeflow Connect?

Ingestion Methods Overview



Batch

- Load data as **batches of rows into Databricks**, often based on a schedule
- Traditional batch ingestion **processes all records** each time it runs
  - `CREATE TABLE AS (CTAS)`
  - `spark.read.load()`

What is Lakeflow Connect?

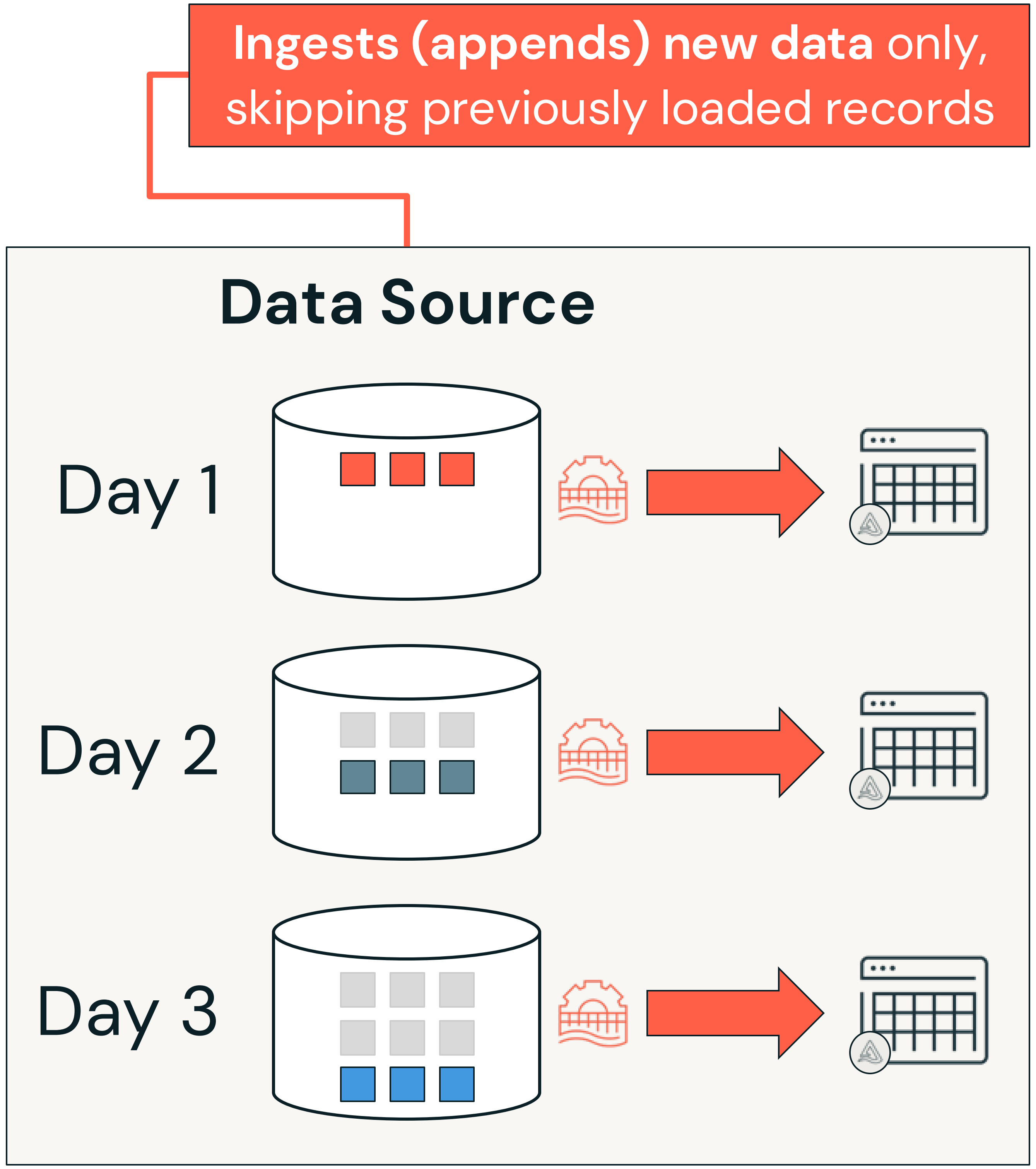
Ingestion Methods Overview

Batch

- Load data as **batches of rows into Databricks**, often based on a schedule
- Traditional batch ingestion **processes all records** each time it runs
  - `CREATE TABLE AS (CTAS)`
  - `spark.read.load()`

Incremental Batch

- **Only new data is ingested**, previously loaded records are **skipped automatically**
- Provides **faster** and more **resource efficient** ingestion by processing less data
  - `COPY INTO`
  - `spark.readStream` (Auto Loader with timed trigger)
  - Declarative Pipelines (`CREATE OR REFRESH STREAMING TABLE`)



What is Lakeflow Connect?

Ingestion Methods Overview

Batch

- Load data as **batches of rows into Databricks**, often based on a schedule
- Traditional batch ingestion **processes all records** each time it runs
  - `CREATE TABLE AS (CTAS)`
  - `spark.read.load()`

Incremental Batch

- **Only new data is ingested**, previously loaded records are **skipped automatically**
- **Faster** ingestion and better **resource efficiency** by processing less data
  - `COPY INTO`
  - `spark.readStream` (Auto Loader with timed trigger)
  - Declarative Pipelines(`CREATE OR REFRESH STREAMING TABLE`)

Streaming

- **Continuously load data** rows or batches of data rows as it is generated so you can query it as it **arrives in near real-time**
- **Micro-batch** processes small batches a **very short, frequent intervals**
  - `spark.readStream` (Auto Loader with continuous trigger)
  - Declarative Pipelines (trigger mode continuous)

databricks

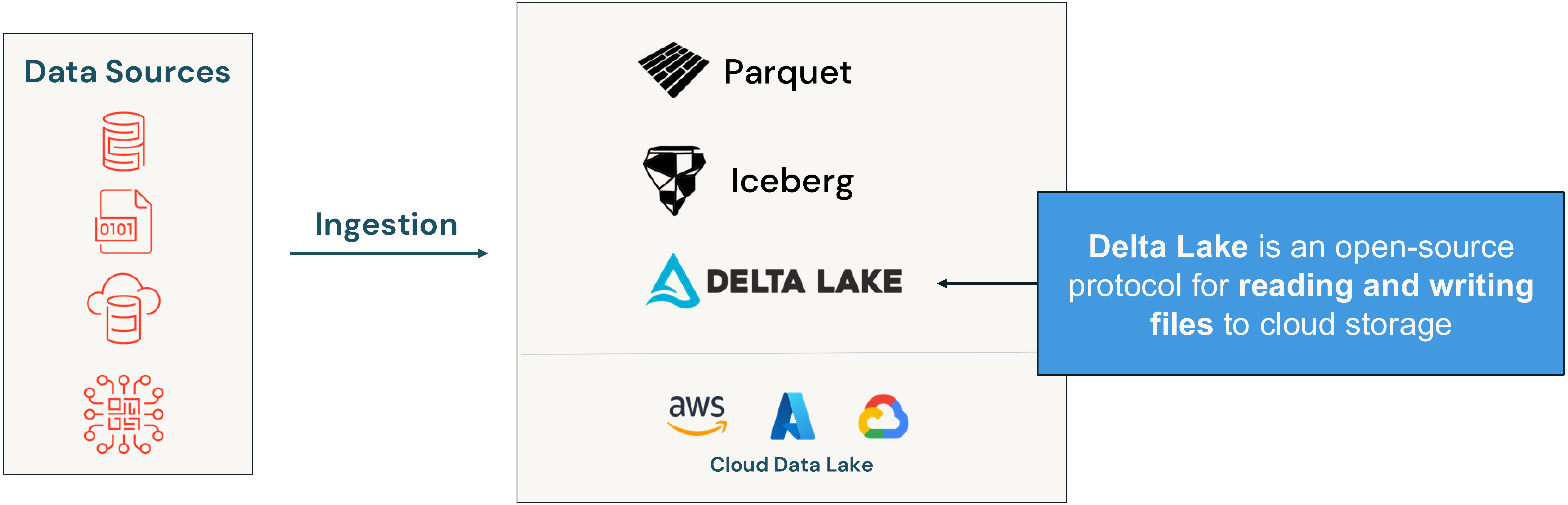
Introduction to Data Engineering in Databricks

LECTURE

Delta Lake Review

Delta Lake Review

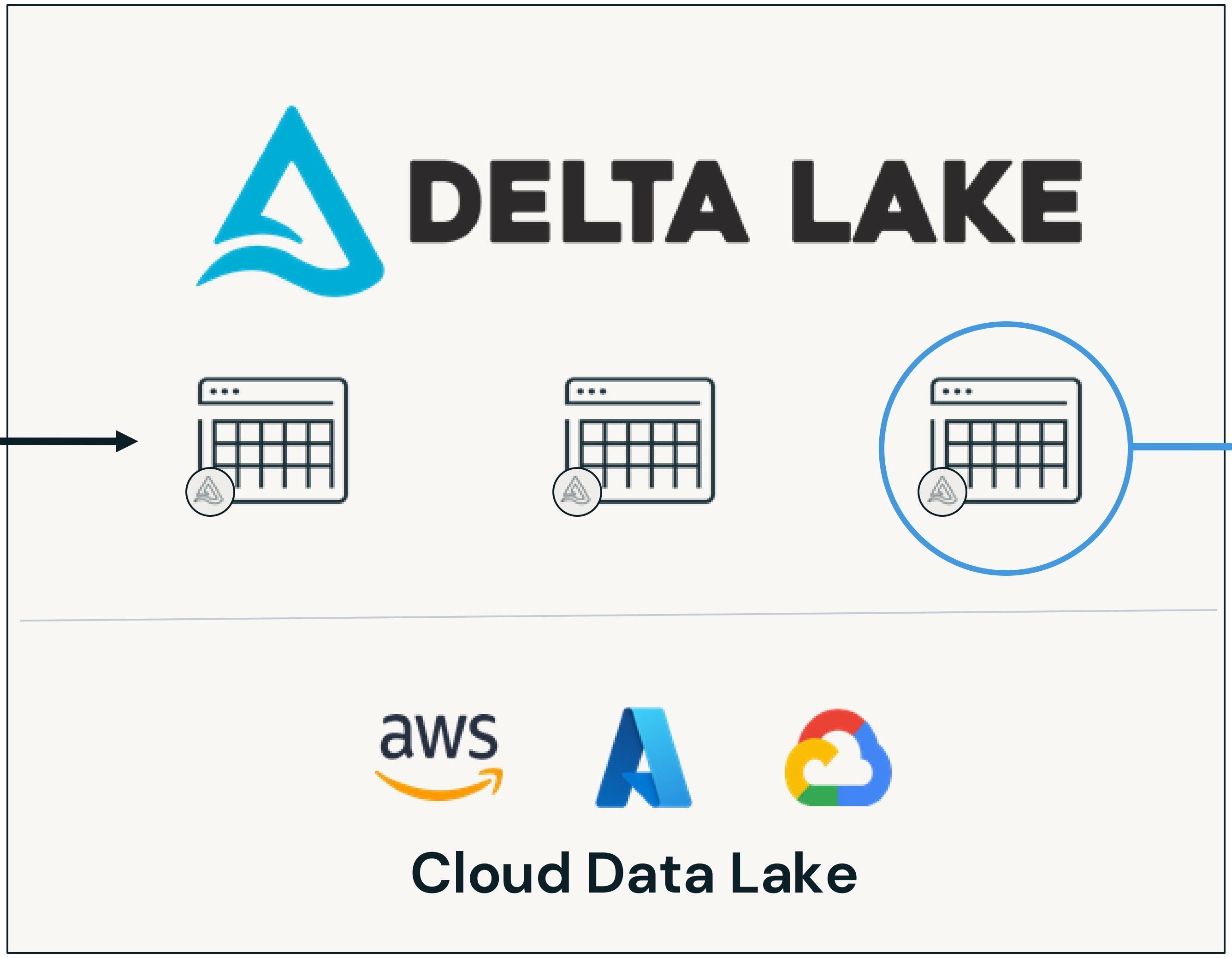
Ingesting Data Into Delta Lake



Delta Lake Review

Delta Table Components Overview

Delta Tables



Files are stored within a directory

DeltaTableName /

…282e.**snappy.parquet**
…432c.**snappy.parquet**
…380c.**snappy.parque**
t
…402z.**snappy.parquet**

_delta_log/
…0000.json
…0001.json
…0002.json

Data is stored as **parquet files** + **Transaction** logs with metadata

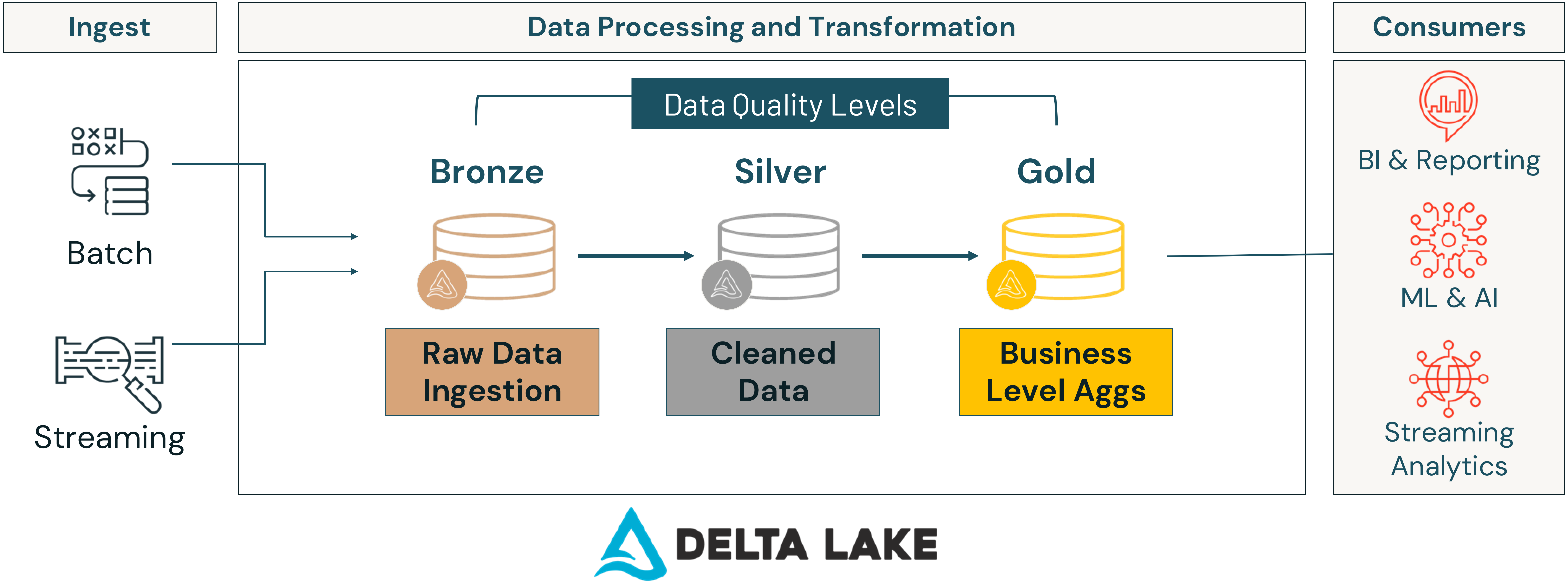
Delta Lake Review

Delta Tables Key Features Review

- ACID Transactions
- Data Manipulation Language (DML)
- Time Travel
- Schema Evolution and Enforcement
- Many more!

Delta Lake Review

Medallion Architecture (Multi Hop) Review

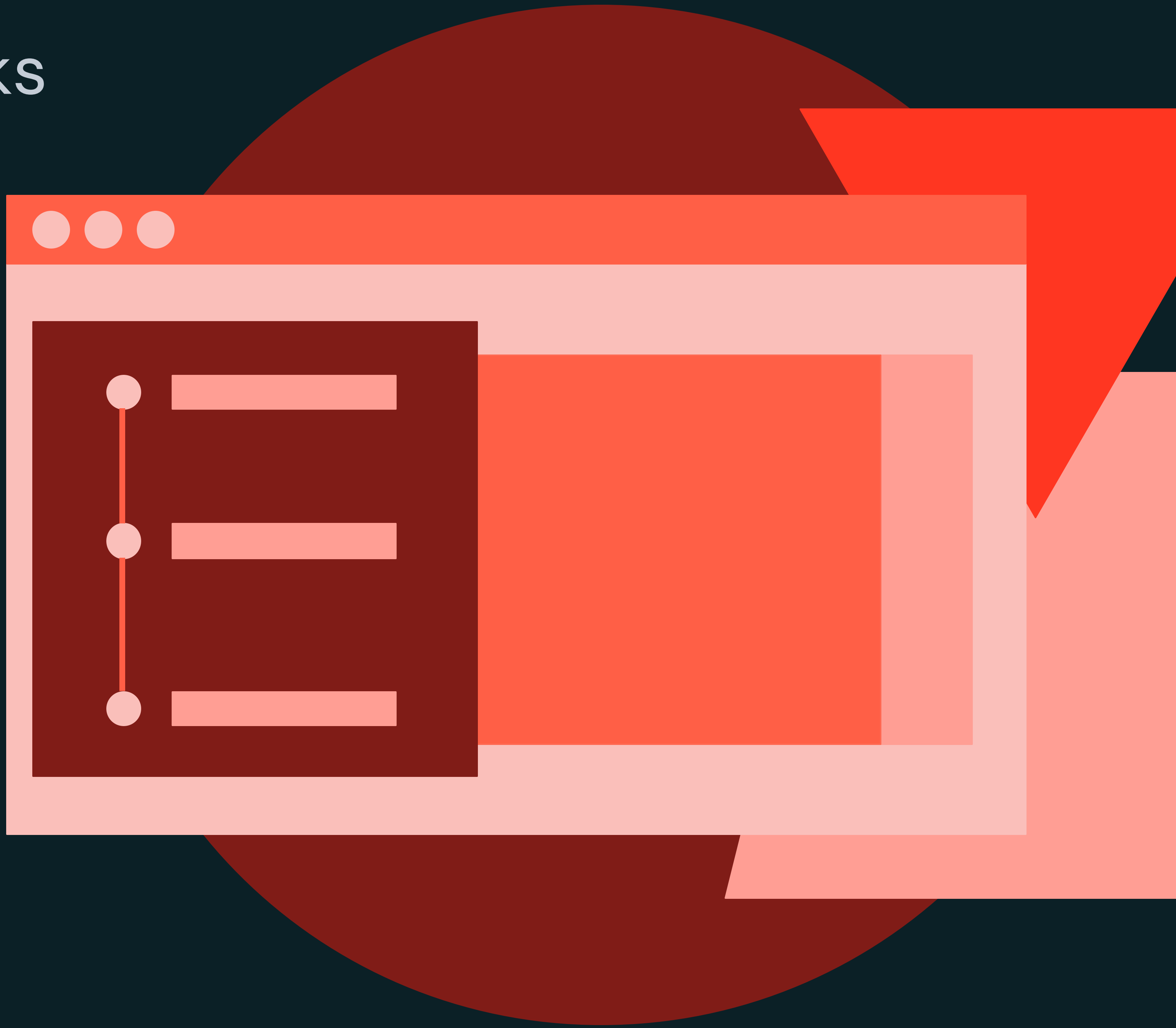


databricks

Introduction to Data Engineering in Databricks

DEMONSTRATION

Exploring the Lab Environment



Notebook: 01 - Exploring the Lab Environment

Cloud Storage Ingestion with LakeFlow Connect Standard Connectors

Data Ingestion with Lakeflow Connect

Agenda

Section Overview – Cloud Storage Ingestion with LakeFlow Connect Standard Connectors
• Introduction to Data Ingestion from Cloud Storage
• Appending Metadata Columns on Ingest
• Working with the Rescued Data Column
• Ingesting Semi-Structured Data: JSON

databricks

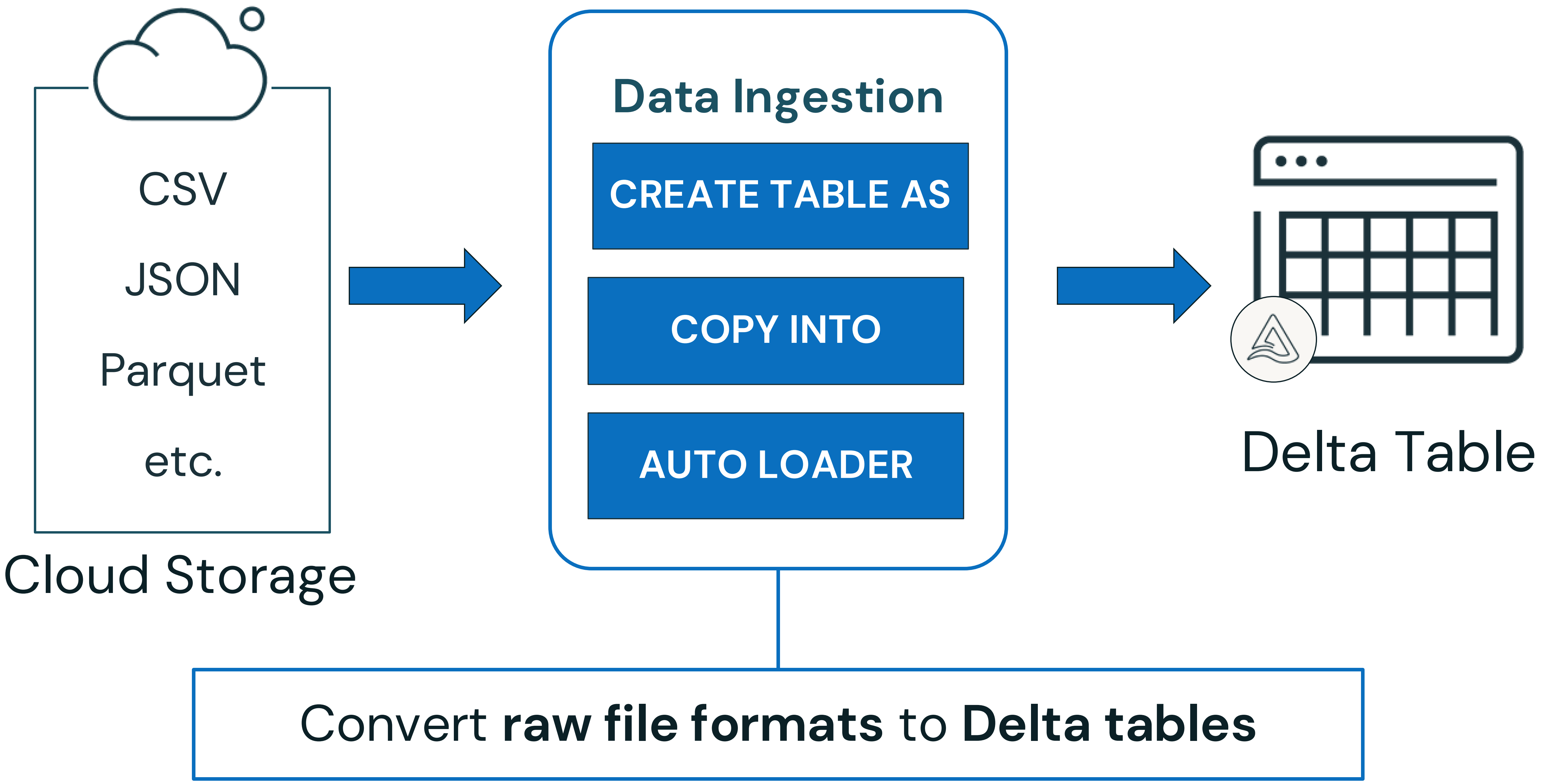
Cloud Storage with LakeFlow Connect Standard Connectors

LECTURE

Introduction to Data Ingestion from Cloud Storage

Data Ingestion from Cloud Storage

Data Ingestion Patterns From Cloud Object Storage



(1 of 3) CREATE TABLE AS (CTAS)

Data Ingestion from Cloud Storage

Method 1 – Batch – CREATE TABLE AS (CTAS)


```
CREATE TABLE new_table AS
SELECT *
FROM read_files(
   <path_to_file(s)>,
    format => '<file_type>',
    <other_format_specific_options>
);
```

`CREATE TABLE AS (CTAS)` creates a Delta table **by default** from files in cloud object storage

The **`read_files()`** function reads files under a provided location and returns the data in **tabular form**

- Supports reading **file formats** like JSON, CSV, XML, TEXT, BINARYFILE, PARQUET, AVRO, and ORC file formats.
- Can detect the **file format automatically and infer a unified schema** across all files.
- Specify **specific file format options** to read in the data based on the source file format
- Can be used in **streaming tables** to **incrementally** ingest files into Delta Lake using Auto Loader

(2 of 3) COPY INTO

Data Ingestion from Cloud Storage

Method 2 – Incremental Batch – COPY INTO (legacy)


```
CREATE TABLE new_table;


COPY INTO new_table
FROM '<dir_path>'
FILEFORMAT=<file_type>
FORMAT_OPTIONS(<options>)
COPY_OPTIONS(<options>)
```

1. **Create an empty table to copy data into**
   - You can create an empty table **without** a schema
   - You also can **explicitly** create the table with a schema

Data Ingestion from Cloud Storage

Method 2 – Incremental Batch – COPY INTO (legacy)


```
CREATE TABLE new_table;

COPY INTO new_table
FROM '<dir_path>'
FILEFORMAT=<file_type>
FORMAT_OPTIONS(<options>)
COPY_OPTIONS(<options>)
```

2. COPY INTO for incremental batch ingestion

- Is a **retriable and idempotent operation** and will **skip files** that have already been loaded **(incremental)**
- Supports various common files types like parquet, JSON, XML, etc
- `FROM` specifies the path of the cloud storage location continuously adding files
- `FORMAT_OPTIONS()` control how the source files are parsed and interpreted. The available options depends on the file format
- `COPY_OPTIONS()` controls the behavior of the COPY INTO operation itself, such as schema evolution (**mergeSchema**) and idempotency (**force**)

(3 of 3) AUTO LOADER

Data Ingestion from Cloud Storage

Method 3 – Incremental Batch or Streaming – Auto Loader

- **Incrementally** and **efficiently** processes new data files (in batch or streaming) as they arrive in cloud storage **without any additional setup**
- Auto Loader has support for both **Python** and **SQL (leveraging Declarative Pipelines)**
- You can use Auto Loader to process billions of files
- Auto Loader is built upon **Spark Structured Streaming**
- A deep dive into Auto Loader is **out of scope for this course**, please refer to these links and courses for more in depth information resources:
    - Documentation and Tutorials
        - Stream Processing and Analysis with Apache Spark™ Course
        - Build Data Pipelines with DLT Course
        - Databricks Streaming and DLT (Advanced) Course

Data Ingestion from Cloud Storage

Method 3 – Incremental Batch or Streaming – Auto Loader

Python Auto Loader

```
(spark
.readStream
    .format("cloudFiles")
    .option("cloudFiles.format", "json")
    .option("cloudFiles.schemaLocation", "<checkpoint_path>")
    .load("/Volumes/catalog/schema/files")
.writeStream
    .option("checkpointLocation", "<checkpoint_path>")
    .trigger(processingTime="5 seconds")
    .toTable("catalog.database.table")
)
```

Auto Loader with SQL (Declarative Pipelines)

```
CREATE OR REFRESH STREAMING TABLE catalog.schema.table
SCHEDULE EVERY 1 HOUR
AS
SELECT *
FROM STREAM read_files(
   '<dir_path>',
   format => '<file_type>'
)
```

Data Ingestion from Cloud Storage

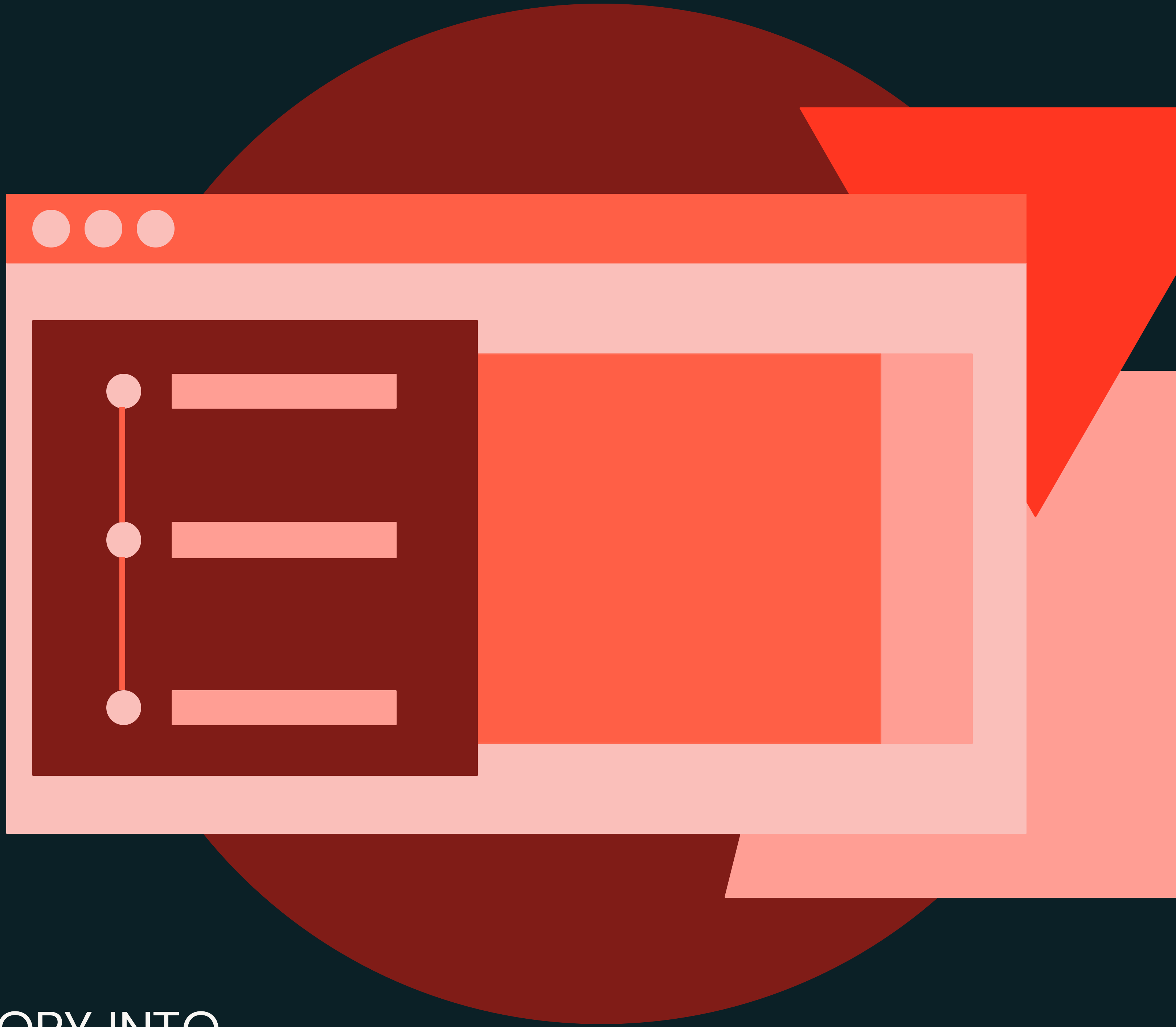
FEATURE	CREATE TABLE AS (CTAS) + spark.read	COPY INTO	Auto Loader
Ingestion Type	Batch	Incremental Batch	Incremental (Batch or Streaming)
Use Cases	Best for smaller datasets	Ideal for thousands of files	Scale to millions+ of files per hour, backfills with billions of files
Syntax/Interface	• Python (spark.read) • SQL (CTAS)	SQL	• Python (spark.readStream) • SQL with Declarative Pipelines (CREATE OR REFRESH STREAMING TABLES) ○ Use streaming tables in Databricks SQL
Idempotency	No	Yes	Yes
Schema Evolution	Manual or inferred during read	Supported with options	Auto Loader automatically detects and evolves schemas. It supports loading data without predefined schemas and handles new columns as they appear.
Latency	High	Moderate (scheduled)	Low or high depending configuration
Easy of Use	Simple	Simple and SQL-based	Intermediate to advanced depending on the implementation (Python or SQL, incremental batch or streaming)
Summary	Best for one time, ad hoc ingestion. Can be scheduled to always read and process all data.	Simple and repeatable for incremental file ingestion. Great for schedule jobs or pipelines.	Best for near real-time streaming or incremental ingestion, with high automation and scalability.

databricks

Cloud Storage with LakeFlow Connect Standard Connectors

DEMONSTRATION

- **Data Ingestion with CREATE TABLE AS and COPY INTO**
- **Create Streaming Tables with SQL using Auto Loader**



Notebook: 02A - Data Ingestion with CREATE TABLE AS and COPY INTO

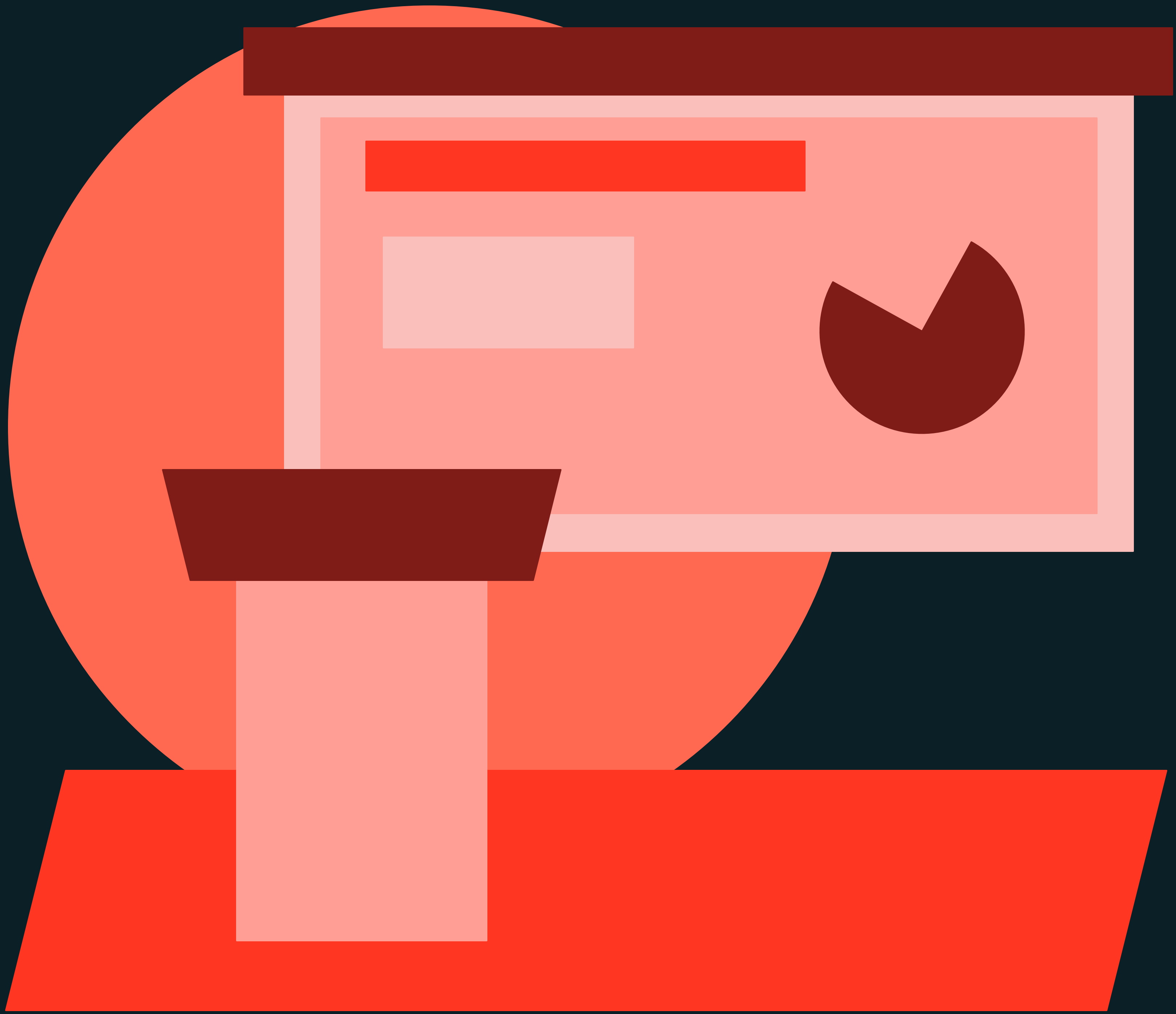
Notebook: 02B - Create Streaming Tables with SQL using Auto Loader

databricks

Cloud Storage with LakeFlow Connect Standard Connectors

LECTURE

Appending Metadata Columns on Ingest



Appending Metadata Columns on Ingest

Adding a Metadata Column

Create a Delta Table

Bronze Table

raw_file

users	unix_ts
peter	1592187804331222
zebi	1592200952155132
...	...

Bronze

Metadata columns

users	unix_ts	last_mod_time	source
peter	1592187804331222	2024-10-01T18:04...	raw_file
zebi	1592200952155132	2024-10-01T18:04...	raw_file
...	...	...	...

Add the **last file modification time**

Add the **source file name**

Appending Metadata Columns on Ingest

Common File Metadata Information From the Input Files

Add last file modification timestamp

```
_metadata.file_modification_time
```

2024-10-07T18:04:42.885+00:00

Add input file name

```
_metadata.file_name
```

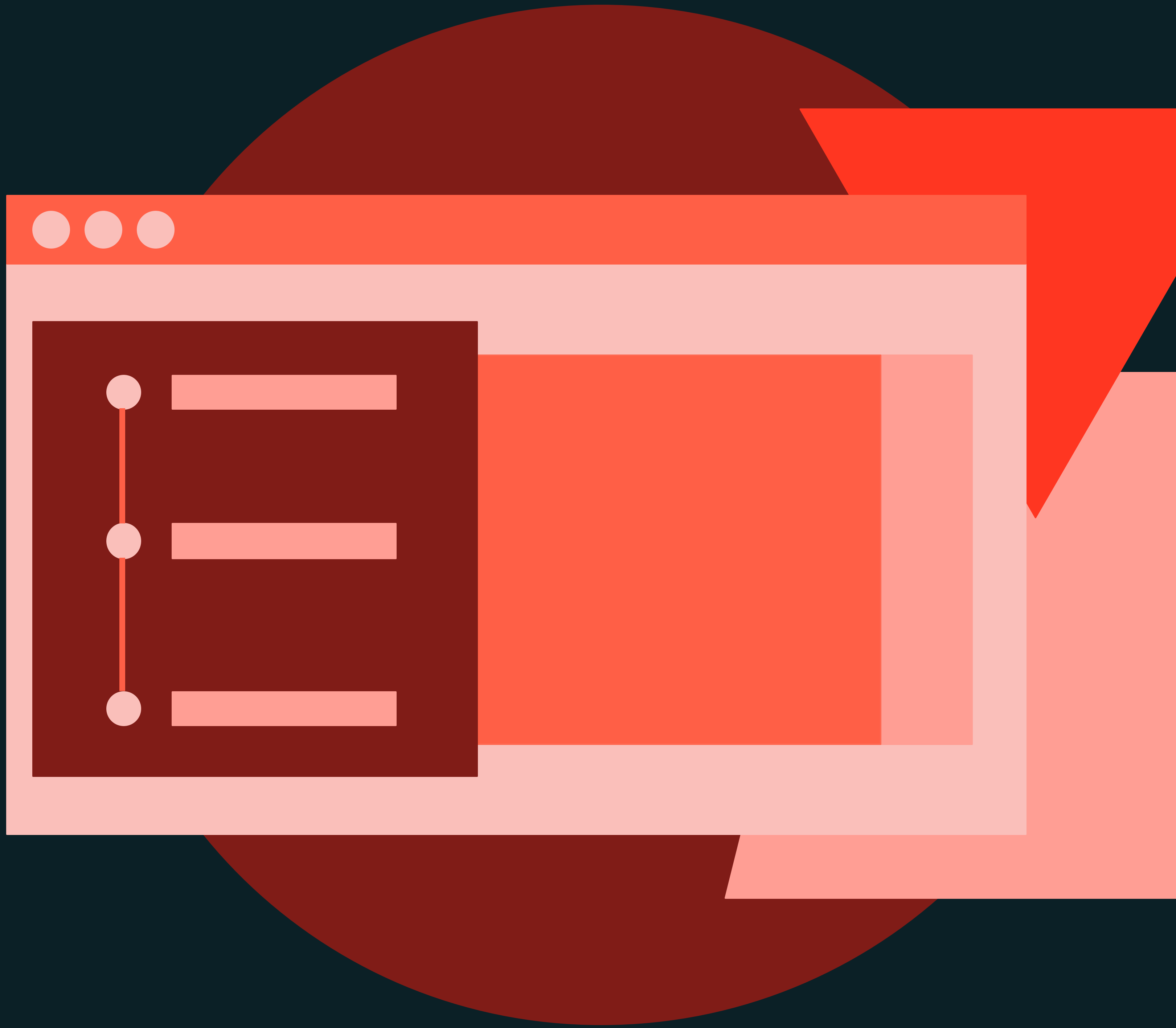
part-00002-7573-1-c000.file_name

databricks

Cloud Storage with LakeFlow Connect Standard Connectors

DEMONSTRATION

Adding Metadata Columns During Ingestion



Notebook: 03 - Adding Metadata Columns During Ingestion

databricks

Cloud Storage with LakeFlow Connect Standard Connectors

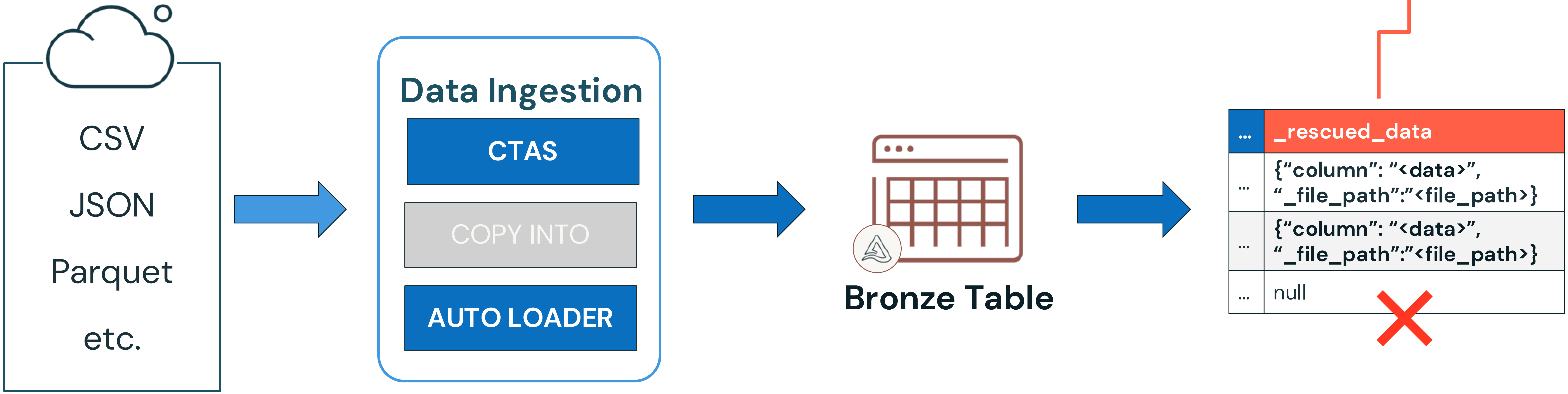
LECTURE

Working with the Rescued Data Column

Working with the Rescued Data Column

Rescuing Malformed Rows on Ingestion

read_files(), spark.read or **Auto Loader** provides a **rescued data column** if the raw data does not match the schema



Working with the Rescued Data Column

Rescued Data Column Example

files

users	cost
peter	$100
zebi	300
…	…

Bronze Table

users (STRING)	cost (BIGINT)	_rescued_data
peter	null	{"cost": "$100", "_file_path":"<file_path>}
zebi	300	null
…	…	…

null since the **cost** is of the correct data type (BIGINT)

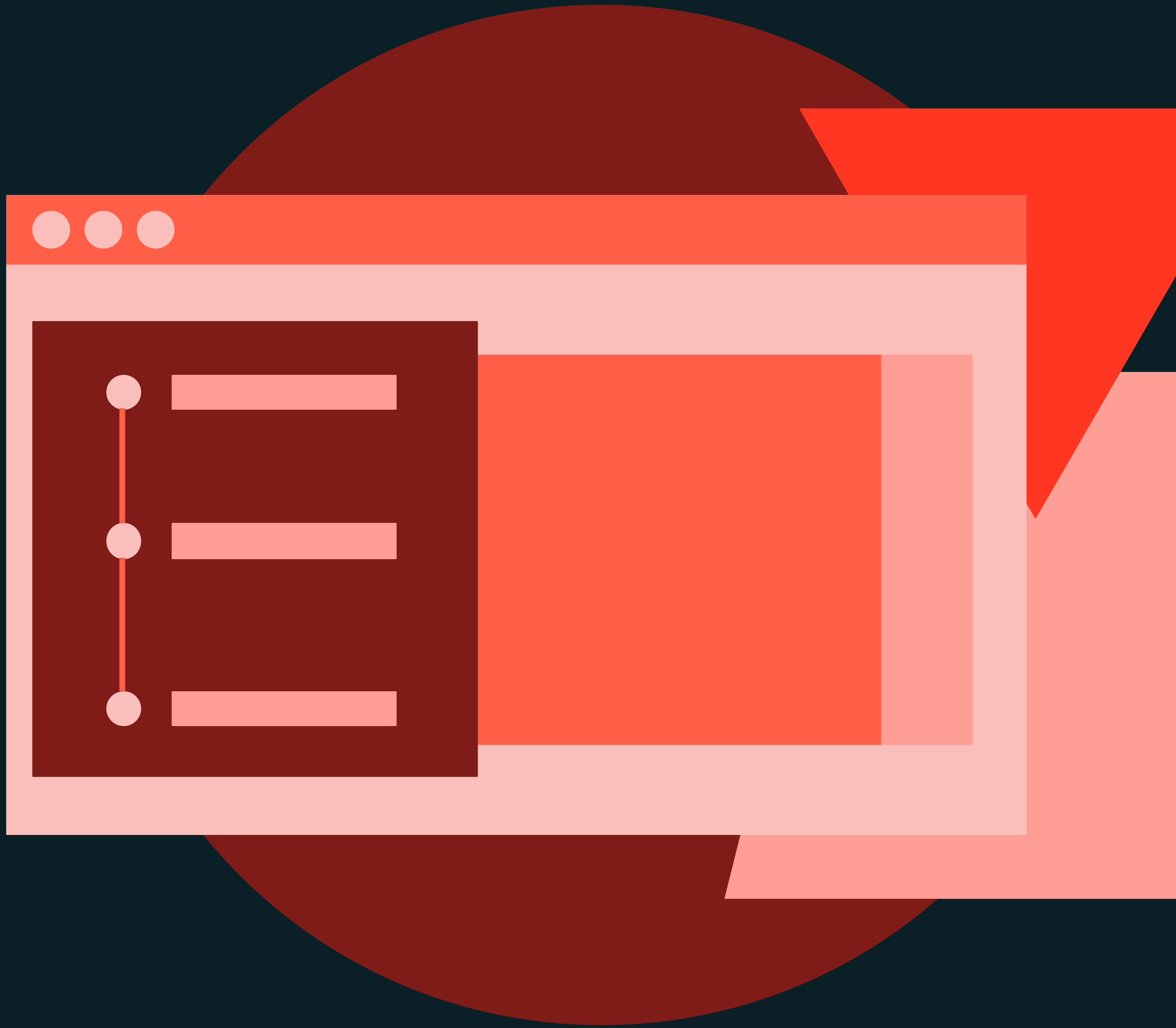
The malformed row is rescued in the column **_rescued_data** as a **JSON formatted string**

databricks

Cloud Storage with LakeFlow Connect Standard Connectors

DEMONSTRATION

Handling CSV Ingestion with the Rescued Data Column



Notebook: 04 - Handling CSV Ingestion with the Rescued Data Column

databricks

Cloud Storage with LakeFlow Connect Standard Connectors

LAB EXERCISE

Creating Bronze Tables from CSV Files

Notebook: 05L - Creating Bronze Tables from CSV Files

databricks

Cloud Storage with LakeFlow Connect Standard Connectors

LECTURE

Ingesting Semi-Structured Data: JSON

Ingesting Semi-Structured Data: JSON

JSON Overview

JSON **objects** are enclosed in brackets

```
{
  "name": "John Doe",
  "age": 35,
  "address": {
    "city": "Anytown",
    "state": "CA"
  },
  "children": [
    {
      "name": "Owen",
      "age": 10
    },
    {
      "name": "Eva",
      "age": 8
    }
  ]
}
```

Ingesting Semi-Structured Data: JSON

JSON Overview

Keys
enclosed in quotation marks

```
{
    "name": "John Doe",
    "age": 35,
    "address": {
        "city": "Anytown",
        "state": "CA"
    },
    "children": [
        {
            "name": "Owen",
            "age": 10
        },
        {
            "name": "Eva",
            "age": 8
        }
    ]
}
```

Ingesting Semi-Structured Data: JSON

JSON Overview

Values

- String
- Number
- Boolean
- Array
- Object

```
{
    "name": "John Doe",
    "age": 35,
    "address": {
        "city": "Anytown",
        "state": "CA"
    },
    "children": [
        {
            "name": "Owen",
            "age": 10
        },
        {
            "name": "Eva",
            "age": 8
        }
    ]
}
```

STRING

NUMERIC

OBJECT

ARRAY of OBJECTS

Ingesting Semi-Structured Data: JSON

Working with a JSON-Formatted STRING Column

json_column
'{"name": "John Doe", "age": 35, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Owen", "age": 10}, {"name": "Eva", "age": 8}]}'
'{"name": "Kristi Doe", "age": 40, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Steve", "age": 10}]}'
...

Columns in tables can hold **JSON formatted strings** as values

Ingesting Semi-Structured Data: JSON

Working with a JSON-Formatted Column as a STRING

json_column
'{"name": "John Doe", "age": 35, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Owen", "age": 10}, {"name": "Eva", "age": 8}]}'
'{"name": "Kristi Doe", "age": 40, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Steve", "age": 10}]}'
...

Working with JSON data can be done using different ***column data types***

1. STRING Data Type

- JSON can be stored as a simple STRING
- Can hold **any JSON STRING** without constraints
- Less **performant**

Use : (colon) syntax to access subfields in JSON formatting strings

```
SELECT json_column:name
```

→ *John Doe*

```
SELECT json_column:address:city
```

→ *Anytown*

Ingesting Semi-Structured Data: JSON

Working with a JSON-Formatted Column as a STRING

json_column
'{"name": "John Doe", "age": 35, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Owen", "age": 10}, {"name": "Eva", "age": 8}]}'
'{"name": "Kristi Doe", "age": 40, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Steve", "age": 10}]}'
…

*Working with JSON data can be done using different **column data types***

1. STRING Data Type

- JSON can be stored as a as a simple STRING
- Can hold **any JSON STRING** without constraints
- Less **performant**

2. STRUCT Data Type

- You can parse JSON data into a STRUCT type, with a **defined schema**
- STRUCT **enforces** the JSON schema
- Is **more efficient** for querying than a JSON formatted STRING

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

JSON String Types	Databricks SQL Data Type
String	**STRING**
Number	**INT/FLOAT/DOUBLE**
Boolean	**BOOLEAN**
Object	**STRUCT <>**
Array	**ARRAY <>**

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

```
{
  "name": "John Doe",
  "age": 35,
  "address": {
    "city": "Anytown",
    "state": "CA"
  },
  "children": [
    {
      "name": "Owen",
      "age": 10
    },
    {
      "name": "Eva",
      "age": 8
    }
  ]
}
```

Define the **schema** of the **JSON formatted string**

```
STRUCT<
  name: STRING,
  age: INT,
  address: STRUCT<
    city: STRING,
    state: STRING
  >,
  children: ARRAY<
      STRUCT<
          name: STRING,
          age: INT
          >
   >
>
```

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

```
{
    "name": "John Doe",
    "age": 35,
    "address": {
        "city": "Anytown",
        "state": "CA"
    },
    "children": [
        {
            "name": "Owen",
            "age": 10
        },
        {
            "name": "Eva",
            "age": 8
        }
    ]
}
```

Specify the **STRUCT** data type to hold the JSON formatted string

```
STRUCT<
    name: STRING,
    age: INT,
    address: STRUCT<
        city: STRING,
        state: STRING
    >,
    children: ARRAY<
        STRUCT<
            name: STRING,
            age: INT
        >
    >
>
```

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

```
{
    "name": "John Doe",
    "age": 35,
    "address": {
        "city": "Anytown",
        "state": "CA"
    },
    "children": [
        {
            "name": "Owen",
            "age": 10
        },
        {
            "name": "Eva",
            "age": 8
        }
    ]
}
```

Specify the STRING and INT data types for the **name** and **age** keys

```
STRUCT<
    name: STRING,
    age: INT,
    address: STRUCT<
        city: STRING,
        state: STRING
    >,
    children: ARRAY<
        STRUCT<
            name: STRING,
            age: INT
        >
    >
>
```

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

```
{
  "name": "John Doe",
  "age": 35,
  "address": {
    "city": "Anytown",
    "state": "CA"
  },
  "children": [
    {
      "name": "Owen",
      "age": 10
    },
    {
      "name": "Eva",
      "age": 8
    }
  ]
}
```

The **address** key holds a **STRUCT** data type with the keys **city** and **state**

```
STRUCT<
  name: STRING,
  age: INT,
  address: STRUCT<
    city: STRING,
    state: STRING
  >,
  children: ARRAY<
      STRUCT<
          name: STRING,
          age: INT
          >
    >
>
```

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

```
{
   "name": "John Doe",
   "age": 35,
   "address": {
      "city": "Anytown",
      "state": "CA"
   },
   "children": [
      {
  1      "name": "Owen",
         "age": 10
      },
      {
  2      "name": "Eva",
         "age": 8
      }
   ]
}
```

The **children** key holds an **ARRAY** of **STRUCTS**

```
STRUCT<
   name: STRING,
   age: INT,
   address: STRUCT<
      city: STRING,
      state: STRING
   >,
   children: ARRAY<
         STRUCT<
               name: STRING,
               age: INT
               >
      >
>
```

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

```
{
  "name": "John Doe",
  "age": 35,
  "address": {
    "city": "Anytown",
    "state": "CA"
  },
  "children": [
    {
      "name": "Owen",
      "age": 10
    },
    {
      "name": "Eva",
      "age": 8
    }
  ]
}
```

We can **derive the schema** of the JSON-formatted STRING column with `schema_of_json`

1

```sql
SELECT schema_of_json('sample-json-string')
```

The function returns the **structure** of the JSON formatted string

```
STRUCT<
  name: STRING,
  age: INT,
  address: STRUCT<
    city: STRING,
    state: STRING
  >,
  children: ARRAY<
      STRUCT<
          name: STRING,
          age: INT
          >
    >
>
```

Ingesting Semi-Structured Data: JSON

Converting JSON Formatted Strings as STRUCTS

```
STRUCT<
   name: STRING,
   age: INT,
   address: STRUCT<
      city: STRING,
      state: STRING
   >,
   children: ARRAY<
         STRUCT<
                name: STRING,
                age: INT
                >
     >
>
```

2

```
SELECT from_json(json_col, 'json-struct-schema>') AS struct_column
FROM table
```

The `from_json` function returns a **struct column** using the JSON string and specified schema

Ingesting Semi-Structured Data: JSON

Working with a JSON-Formatted Column as a STRING

json_column
'{"name": "John Doe", "age": 35, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Owen", "age": 10}, {"name": "Eva", "age": 8}]}'
'{"name": "Kristi Doe", "age": 40, "address": {"city": "Anytown", "state": "CA"}, "children": [{"name": "Steve", "age": 10}]}'
...

Working with JSON data can be done using different ***column data types***

1. STRING Data Type

- JSON can be stored as a as a simple STRING
- Can hold **any JSON STRING** without constraints
- Less **performant**

2. STRUCT Data Type

- You can parse JSON data into a STRUCT type, with a **defined schema**
- STRUCT **enforces** the JSON schema
- Is **more efficient** for querying than a JSON formatted STRING

3. VARIANT Data Type

- Can store any type of data, including JSON, and **is ideal for semi-structured** data
- Highly **flexible**
- Improved **performance** over existing methods

Public Preview as of 2025 Q2

databricks

Cloud Storage with LakeFlow Connect Standard Connectors

DEMONSTRATION

Ingesting JSON files with Databricks

Notebook: 06 – Ingesting JSON Files with Databricks

databricks

Cloud Storage with LakeFlow Connect Standard Connectors

LAB EXERCISE

Creating Bronze Tables from JSON Files

Notebook: 07L - Creating Bronze Table from JSON Files

Enterprise Data with LakeFlow Connect Managed Connectors

Data Ingestion with with Lakeflow Connect

Agenda

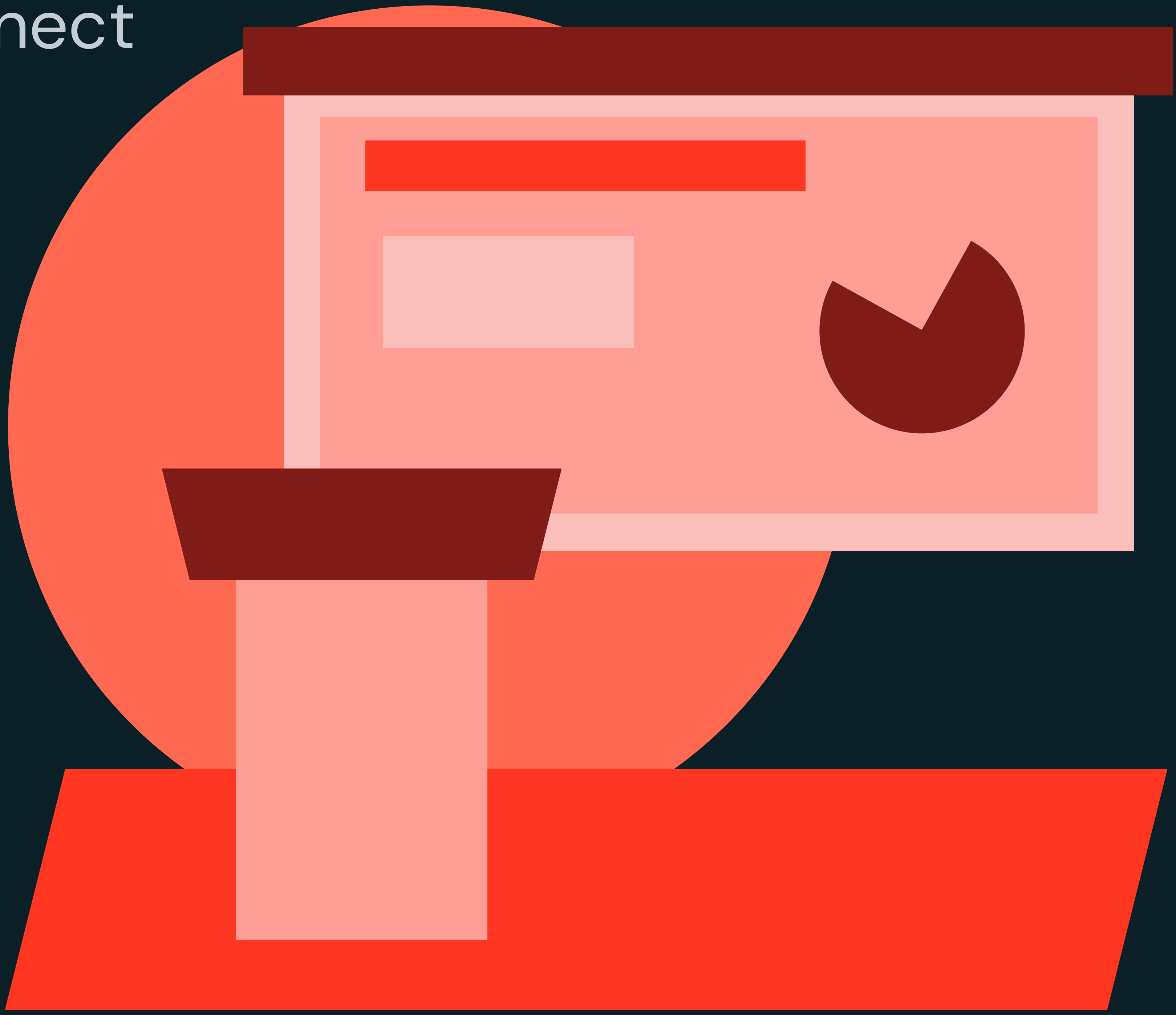
Section Overview – Enterprise Data with LakeFlow Connect Managed Connectors
• Ingesting Enterprise Data into Databricks Overview
• Enterprise Data Ingestion with Lakeflow Connect

databricks

Ingesting Enterprise Data with LakeFlow Connect

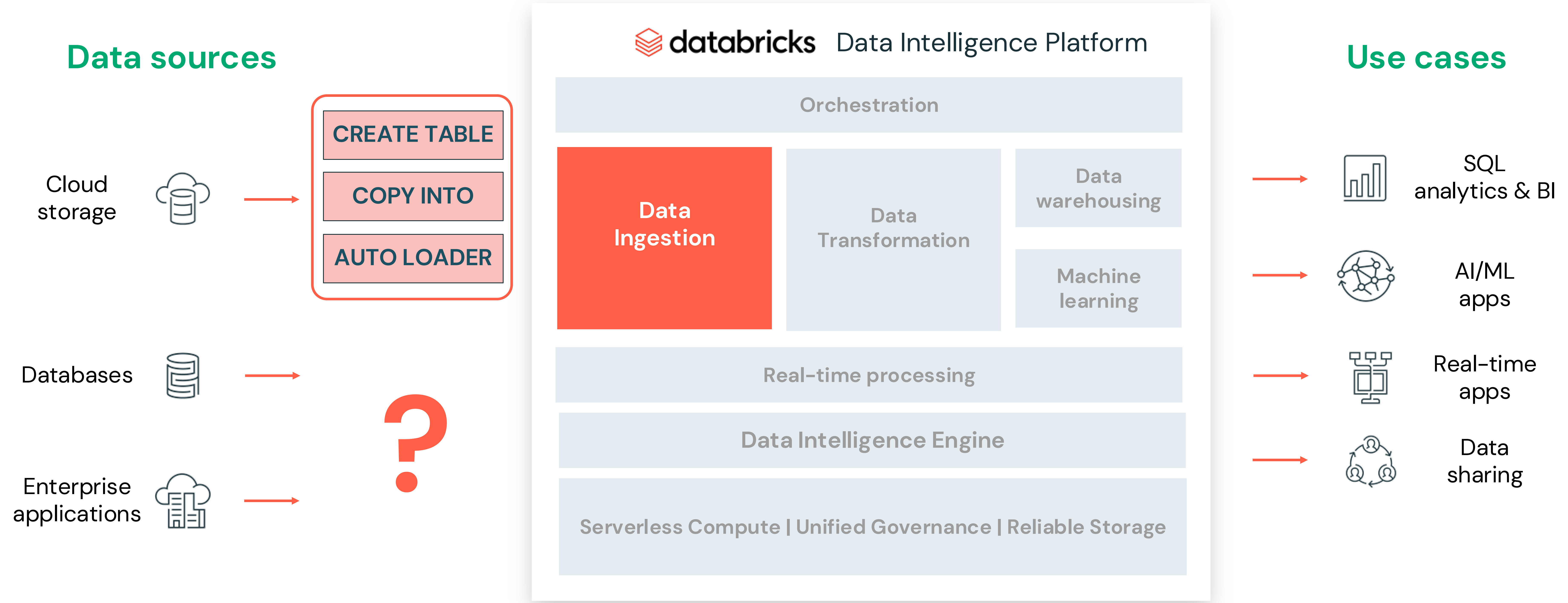
LECTURE

Ingesting Enterprise Data Overview



Ingesting Enterprise Data Overview

Data Ingestion to Databricks Overview



Ingesting Enterprise Data Overview

Lakeflow Connect Managed Connectors

Lakeflow Connect
Managed Connectors

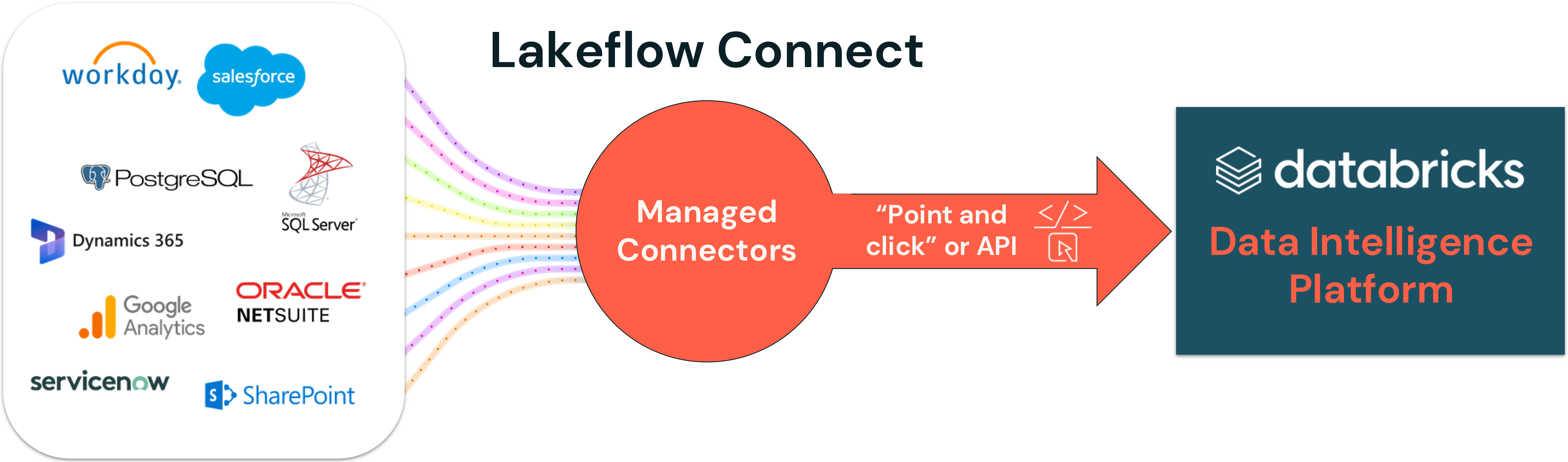
databricks

BENEFITS

- **Simplify** the process of ingestion data from a wide variety of enterprise databases and applications
- Provide a easy to use **user interface (UI)** (or you can use the API)
- **Fully managed by Databricks**, reducing the need for manual configuration or custom code

Ingesting Enterprise Data Overview

Data Ingestion with Lakeflow Connect Managed Connectors

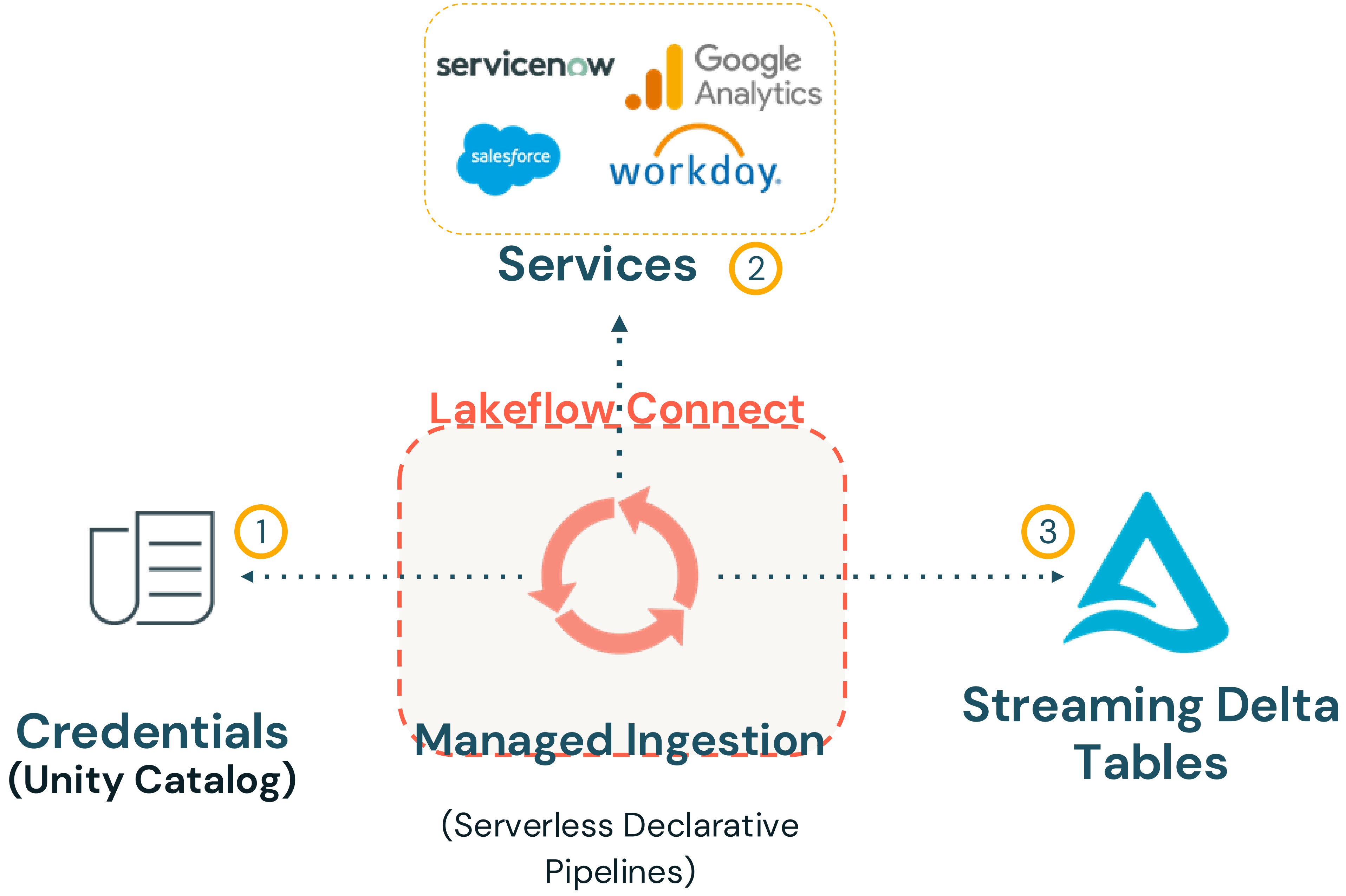


Efficient Databricks Managed Connectors for Ingestion into your Lakehouse

Managed connectors in Lakeflow Connect are in various release states

Ingesting Enterprise Data Overview

Lakeflow Connect Managed Connectors: SaaS Ingestion

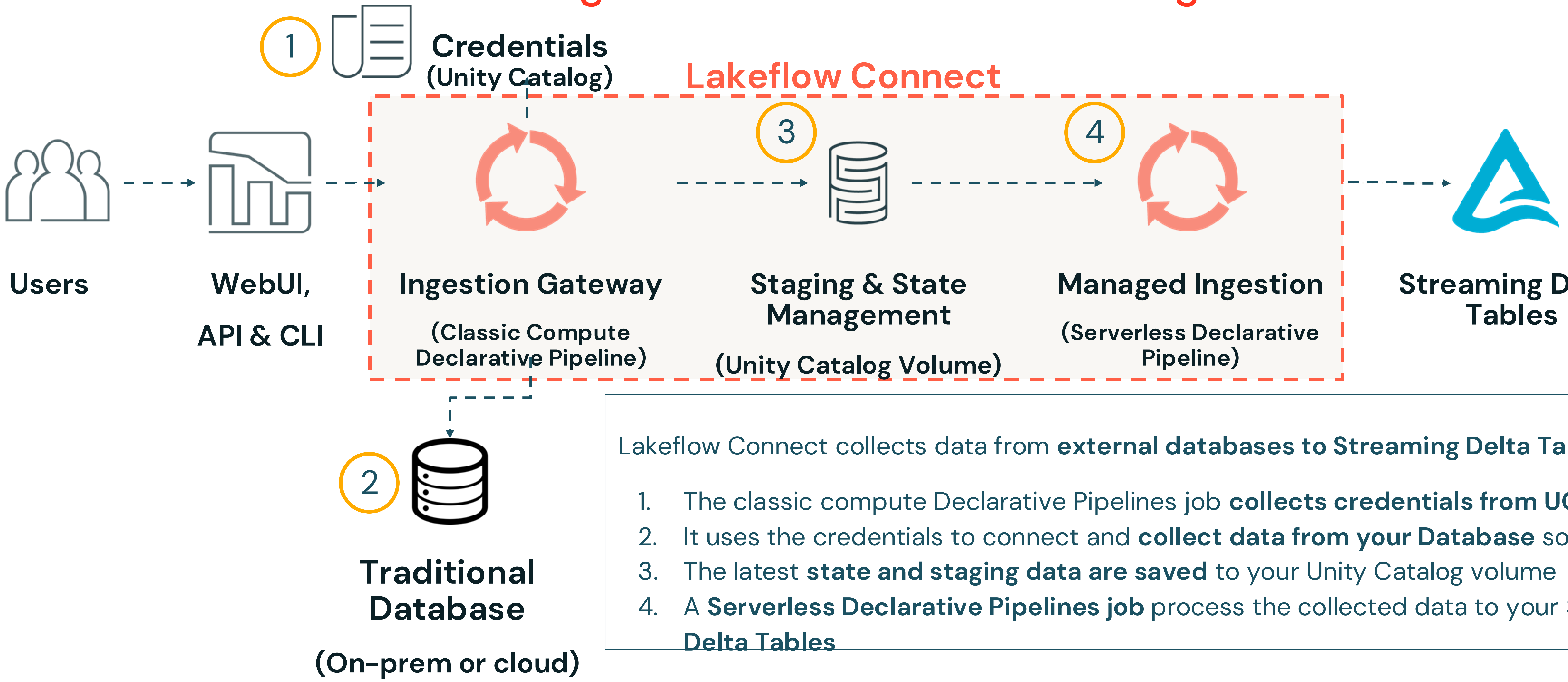


Lakeflow Connect collects data from **external sources to Streaming Delta Tables** using a **serverless compute Declarative Pipelines pipeline**:

1. A Lakeflow Serverless Declarative Pipelines job **collects credentials** from Unity Catalog.
2. The job **reaches out** to the publicly accessible data source (e.g., API, open OLAP port, etc.).
3. The service transforms the data and stores it to a **Streaming Delta Table**.

Ingesting Enterprise Data Overview

Lakeflow Connect Managed Connectors: Database Ingestion



Lakeflow Connect collects data from **external databases to Streaming Delta Tables**

1. The classic compute Declarative Pipelines job **collects credentials from UC**
2. It uses the credentials to connect and **collect data from your Database** sources
3. The latest **state and staging data are saved** to your Unity Catalog volume
4. A **Serverless Declarative Pipelines job** process the collected data to your **Streaming Delta Tables**

Ingesting Enterprise Data Overview

Data Ingestion with Partner Connect

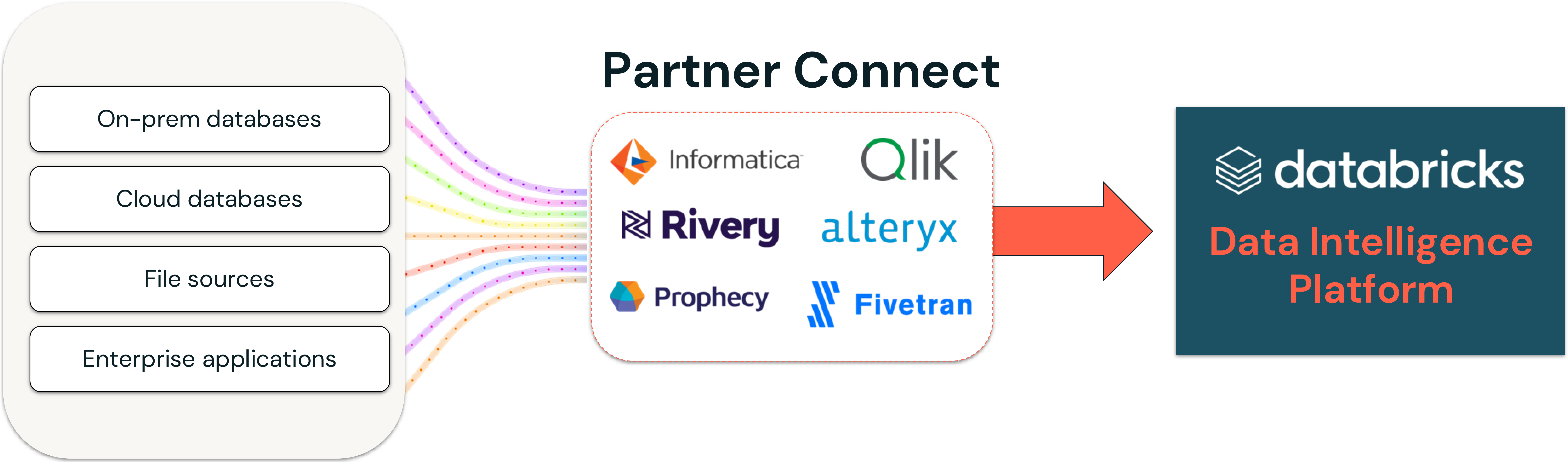
Lakeflow Connect
Managed Connectors

databricks

Partner Connect

Ingesting Enterprise Data Overview

Data Ingestion with Partner Connect

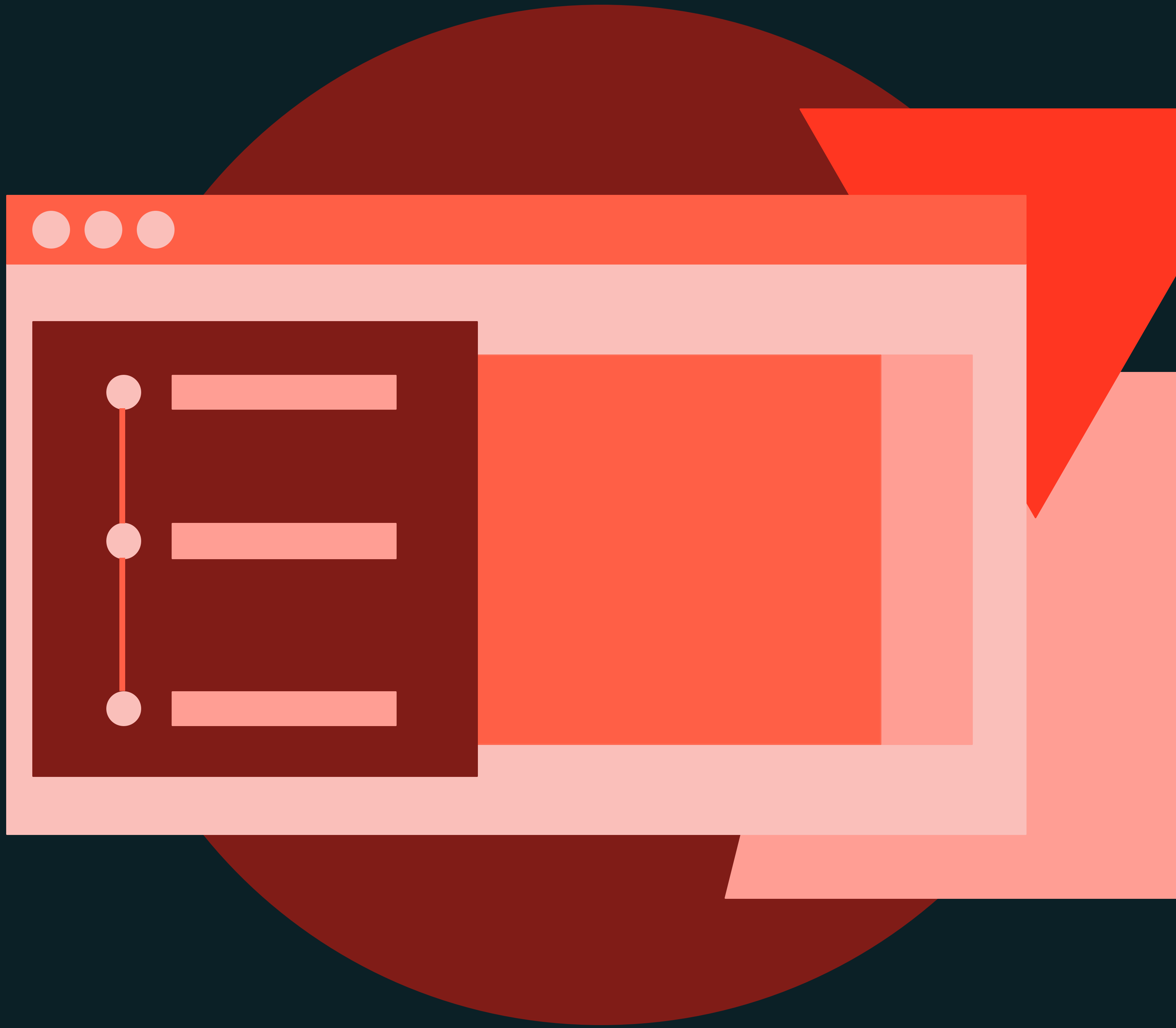


Utilize a rich ecosystem of partner solutions for data ingestion

Data Ingestion and Transformation

DEMONSTRATION

Enterprise Data Ingestion with Lakeflow Connect



Notebook: 08 - Enterprise Data Ingestion with Lakeflow Connect

Ingestion Alternatives (BONUS)

Data Ingestion with with Lakeflow Connect

Agenda

Section Overview – Ingestion Alternatives
• Additional Ingestion Features Overview
• Ingesting into Existing Delta Tables
• Data Ingestion with MERGE INTO

Building Pipelines

LECTURE

Additional Features Overview

Additional Features Overview

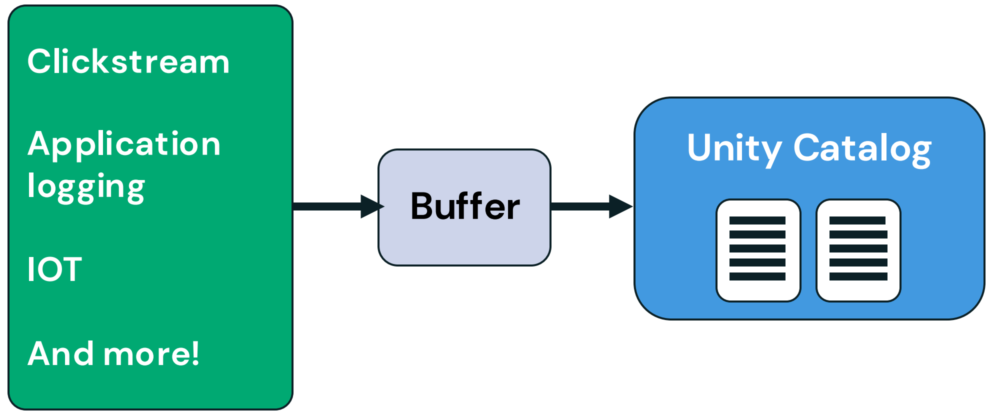
What's Next: Features Outside This Course

Lakehouse Federation

Allows you to **query external data sources** without moving your data

Especially useful for:

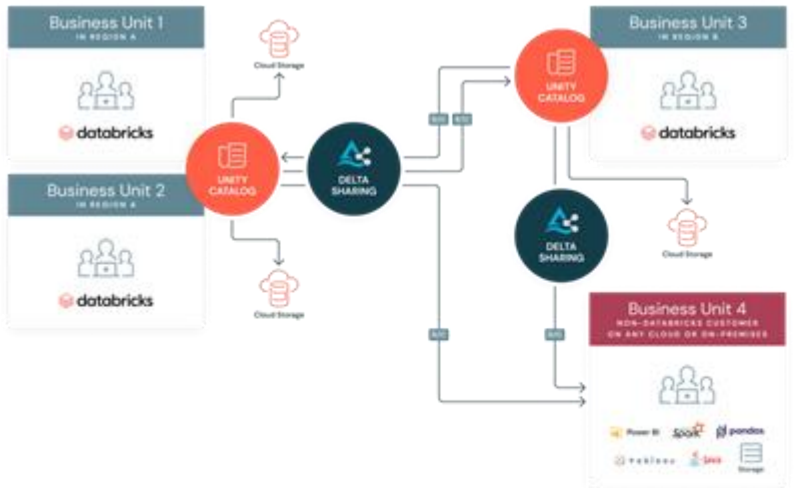
- Ad hoc reporting
- Proof-of-concept work
- The exploratory phase of new ETL pipelines or reports
- Supporting workloads during incremental migration



Zerobus (coming soon)

A **Lakeflow Connect API** that allows developers to **write event data directly to their lakehouse** at very high throughput (100 MB/s) with near real-time latency (<5 seconds)

Simplify ingestion for IOT, clickstreams, telemetry, and more.



Delta Sharing

Allows you to securely share data across platforms, clouds, and regions

Additional Ingestion Features Overview

Ingesting Data with Databricks Marketplace

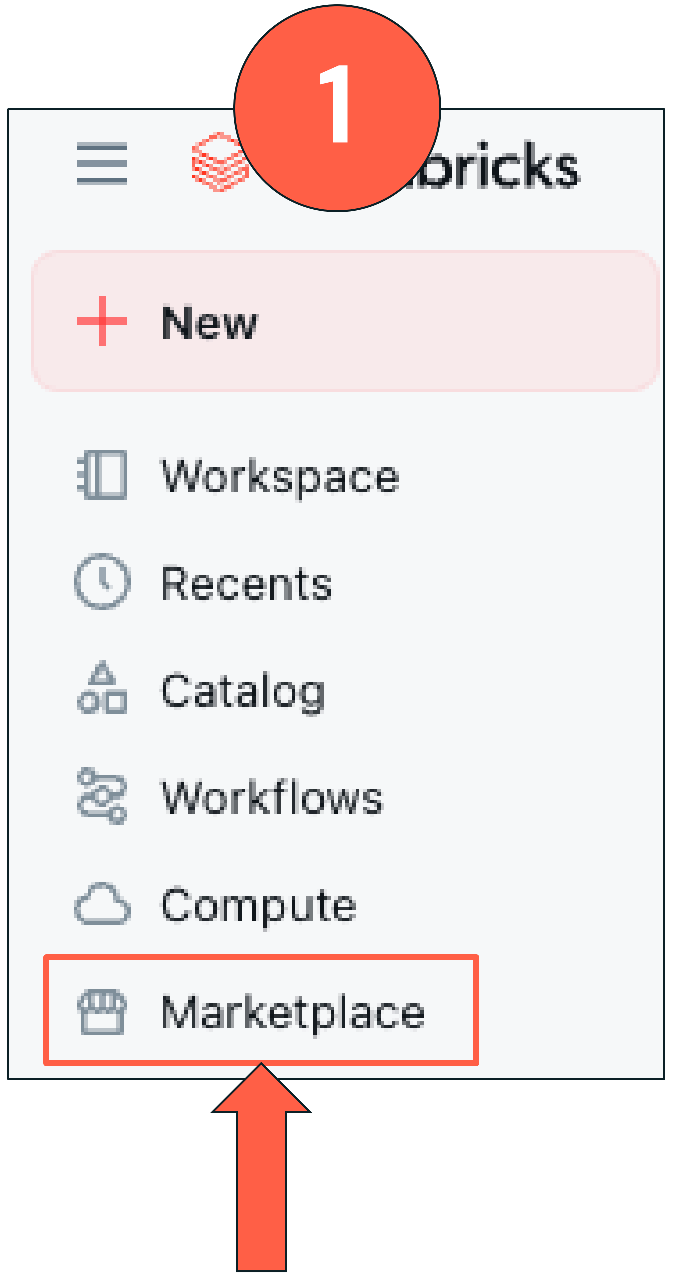
Open exchange for all data products

- Datasets
- Notebooks
- Dashboards
- ML models
- Solutions Accelerators

Powered by Delta Sharing

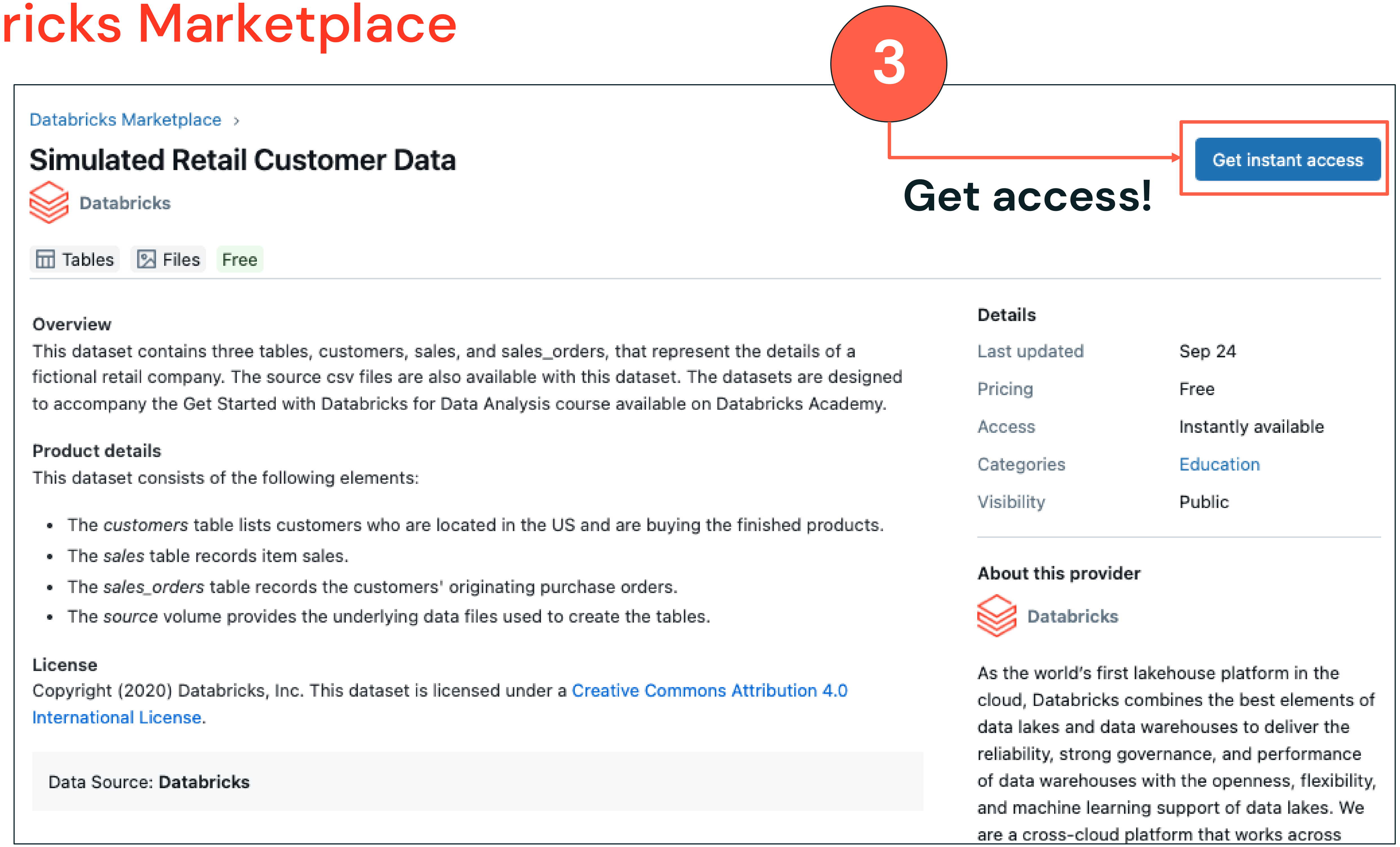
Ingesting Data with Databricks Marketplace

Delta Sharing with Databricks Marketplace



2

Find your assets



databricks

Ingestion Alternatives

LECTURE

Ingesting into Existing Delta Tables

Ingesting into Existing Delta Tables

Updates, Inserts, and Deletes on Delta Tables with MERGE INTO

- Merges a set of **updates, insertions, and deletions** from a **source** table into a **target** Delta table.
- MERGE INTO supports **schema enforcement** or **schema evolution** and allows different actions depending on whether a row is matched between a source and target table:
  - Matched rows: **UPDATE** or **DELETE**
  - Unmatched rows by target: **INSERT**
  - Unmatched rows by source: **UPDATE** or **DELETE**
- This functionality makes MERGE INTO ideal for handling **slowly changing dimensions (SCDs)**, **incremental loads**, and complex **change data capture (CDC)** scenarios.

Ingesting into Existing Delta Tables

Updates, Inserts, and Deletes on Delta Tables with MERGE INTO

target_table

users	email	status
peter	peter@email.com	current
zebi	zebi@email.com	current
...	...	...

Update **target_table** with the **source_table**

source_table

users	email	status
peter	peter@email.com	delete
zebi	**zebi@other.com**	update
samarth	samarth@other.com	new

target_table

users	email	status
zebi	**zebi@other.com**	update
samarth	**samarth@other.com**	**new**
...	...	...

Ingesting into Existing Delta Tables

MERGE INTO

source_table

id	users	email	status
1	peter	peter@email	delete
2	zebi	zebi@email	update
3	samarth	samarth@email	new

Original target_table

id	users	email	status
1	peter	peter@email	current
2	zebi	zebi@email	current
4	matt	matt@email	current

```
MERGE INTO target_table target
USING source_table source
```

Target table (table to update)

Source table (table to merge into target)

Ingesting into Existing Delta Tables

source_table

id	users	email	status
1	peter	peter@email	delete
2	zebi	zebi@email	update
3	samarth	samarth@email	new

Updated target_table

id	users	email	status
1	peter	peter@email	current
2	zebi	zebi@email	current
4	matt	matt@email	current

```
MERGE INTO target_table target
USING source_table source
ON target.id = source.id
```

Specify the condition for merging

Ingesting into Existing Delta Tables

source_table

id	users	email	status
1	peter	peter@email	delete
2	zebi	zebi@email	update
3	samarth	samarth@email	new

Updated target_table

id	users	email	status
1	peter	peter@email	current
2	zebi	zebi@email	update
4	matt	matt@email	current

```
MERGE INTO target_table target
USING source_table source
ON target.id = source.id
WHEN MATCHED AND source.status = 'update' THEN
 UPDATE SET
   target.email = source.email,
   target.status = source.status
```

First WHEN clause for merge

Ingesting into Existing Delta Tables

source_table

id	users	email	status
1	peter	peter@email	delete
2	zebi	zebi@email	update
3	samarth	samarth@email	new

Updated target_table

id	users	email	status
~~1~~	~~peter~~	~~peter@email~~	~~current~~
2	zebi	zebi@email	update
4	matt	matt@email	current

```
MERGE INTO target_table target
USING source_table source
ON target.id = source.id
WHEN MATCHED AND source.status = 'update' THEN
 UPDATE SET
   target.email = source.email,
   target.status = source.status
WHEN MATCHED AND source.status = 'delete' THEN
 DELETE
```

Second WHEN clause for merge

Ingesting into Existing Delta Tables

source_table

id	users	email	status
1	peter	peter@email	delete
2	zebi	zebi@email	update
3	samarth	samarth@email	new

Updated target_table

id	users	email	status
2	zebi	zebi@email	update
4	matt	matt@email	current
3	samarth	samarth@email	new

```
MERGE INTO target_table target
USING source_table source
ON target.id = source.id
WHEN MATCHED AND source.status = 'update' THEN
 UPDATE SET
   target.email = source.email,
   target.status = source.status
WHEN MATCHED AND source.status = 'delete' THEN
 DELETE
WHEN NOT MATCHED THEN
 INSERT (id, first_name, email, sign_up_date,
status)
 VALUES (source.id, source.first_name,
source.email, source.sign_up_date,
source.status);
```

WHEN NOT MATCHED clause for merge

Ingesting into Existing Delta Tables

source_table

id	users	email	status
1	peter	peter@email	delete
2	zebi	zebi@email	update
3	samarth	samarth@email	new

Fully Updated target_table

id	users	email	status
2	zebi	zebi@email	current
4	matt	matt@email	current
3	samarth	samarth@email	new

```
MERGE INTO target_table target
USING source_table source
ON target.id = source.id
WHEN MATCHED AND source.status = 'update' THEN
 UPDATE SET
   target.email = source.email,
   target.status = source.status
WHEN MATCHED AND source.status = 'delete' THEN
 DELETE
WHEN NOT MATCHED THEN
 INSERT (id, first_name, email, sign_up_date,
status)
 VALUES (source.id, source.first_name,
source.email, source.sign_up_date,
source.status);
```

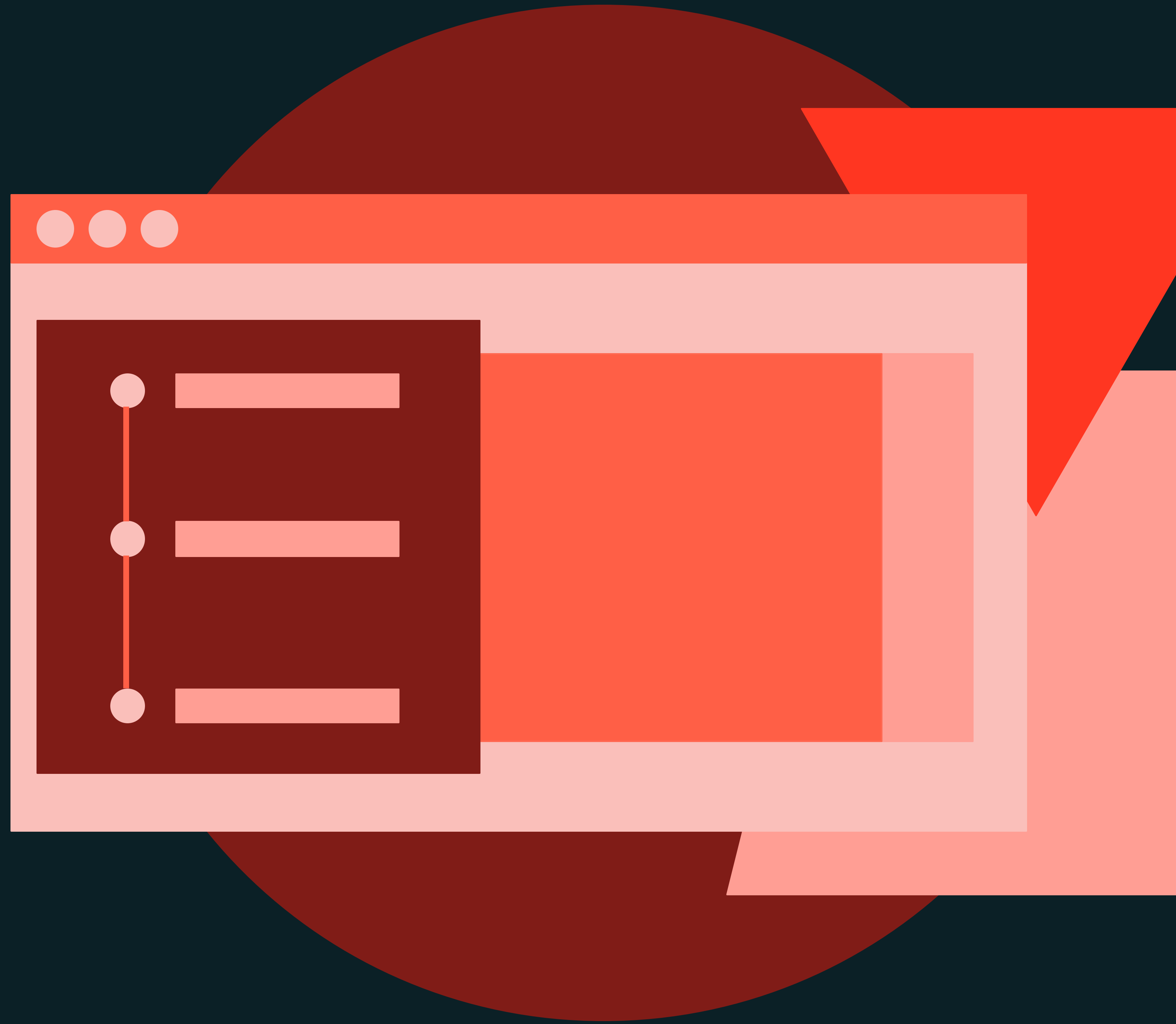
databricks

Ingestion Alternatives

DEMONSTRATION

BONUS – Data Ingestion with MERGE INTO

If time permits, take a look at the fundamentals of MERGE INTO within Databricks for upserting data into another table.



Notebook: 09 – BONUS – Data Ingestion with Merge Into

Summary and Next Steps

Data Ingestion with with Lakeflow Connect

Course Learning Objective Recap

- Describe Lakeflow Connect as a scalable and simplified solution for data ingestion into Databricks from a variety of sources.
- Review the benefits of Delta tables and the Medallion architecture.
- Demonstrate how to ingest data from cloud object storage into Delta tables using CREATE TABLE AS, COPY INTO, and Auto Loader, including capturing input file metadata in Bronze layer tables.
- Explain how rescued columns are used during ingestion to manage malformed records.
- Illustrate techniques for ingesting and flattening semi structured JSON data from cloud storage.
- Describe available options for ingesting data from enterprise systems using Lakeflow Connect Managed Connectors.
- Discuss alternative ingestion methods such as MERGE INTO, Delta Sharing and Databricks Marketplace.

Next Steps

Additional resources for continuing the learning journey.

Data Engineering with Databricks

- Continue your learning through self-paced or instructor-led offerings
- The courses offer hands-on instruction in:
  - Databricks Data Science & Engineering Workspace
  - Databricks SQL
  - Declarative Pipelines
  - Databricks Repos
  - Databricks Task Orchestration
  - Unity Catalog

Data Engineer Associate Certification

- Validate your data and AI skills on Databricks by earning a Databricks credential
- Exam information and exam guide
- The exam covers:
  - Data Intelligence Platform
  - ELT With Spark SQL and Python
  - Incremental Data Processing
  - Production Pipelines
  - Data Governance

databricks