Welcome

Advanced Data Engineering with Databricks

Advanced Data Engineering with Databricks

Learning Objectives

1. Design databases and pipelines optimized for the Databricks Lakehouse Platform.
2. Implement efficient incremental data processing to validate and enrich data driving business decisions and applications.
3. Leverage Databricks-native features for managing access to sensitive data and fulfilling right-to-be-forgotten requests.
4. Manage error troubleshooting, code promotion, task orchestration, and production job monitoring using Databricks tools.

Course Tools

Lecture

Breakout

Rooms

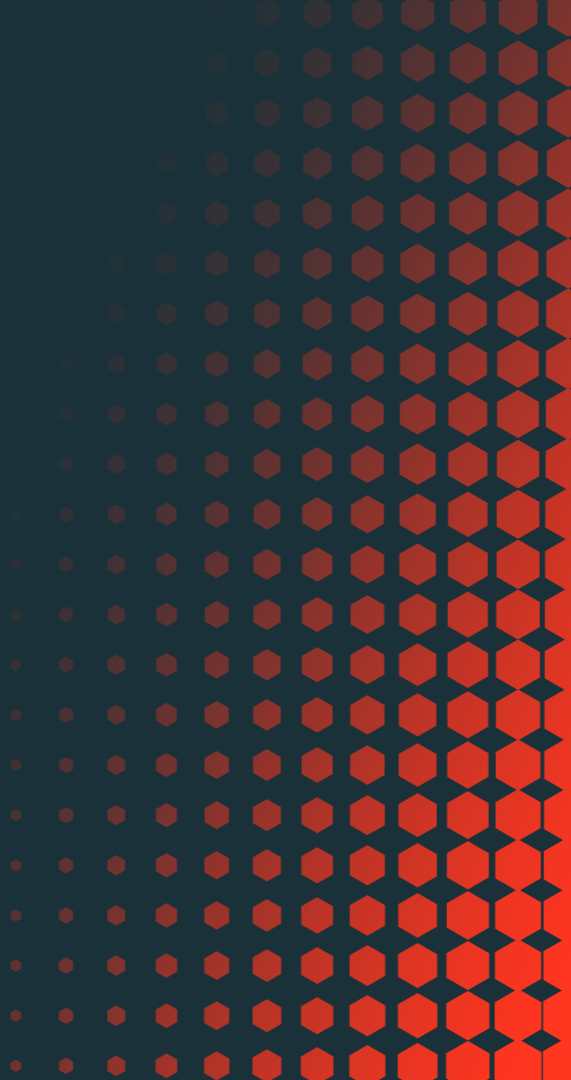
TA Help + Discussion

Resources

Lab Notebooks

Solutions

Agenda



Module 1 - Architecting for the Lakehouse

- Course Intro & Overview
- Adopting the Lakehouse Architecture
- The Lakehouse Medallion Architecture
- Setting Up Tables
- Optimizing Data Storage
- Understanding Delta Lake Transactions
- Delta Lake Isolation with Optimistic Concurrency
- Streaming Design Patterns

Module 2 - Managing Data in Motion

- Clone for Development and Data Backup
- Auto Loader
- Bronze Ingestion Patterns
- Promoting Bronze to Silver
- Streaming Deduplication
- Quality Enforcement
- Slowly Changing Dimensions in the Lakehouse
- Streaming Joins and Statefulness
- Stream-static Joins

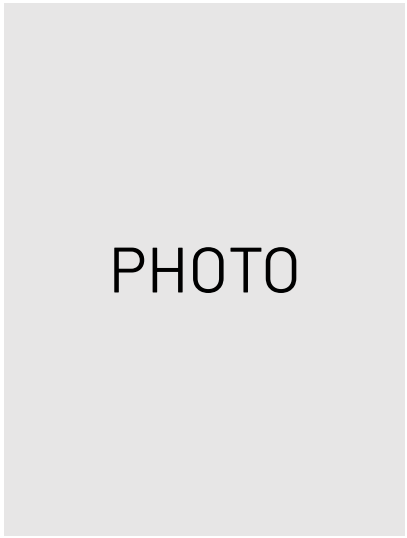
Module 3 - Privacy and Governance for Lakehouse Analytics

- Lakehouse and the Query Layer
- Stored Views
- Materialized Gold Tables
- PII & Regulatory Compliance
- Creating a Pseudonymized PII Lookup Tables
- Storing PII Securely
- Managing ACLS for the Enterprise Lakehouse
- Deidentified PII Access
- Propagating Deletes with Delta Change Data Feed
- Deleting at Partition Boundaries

Module 4 - Databricks in Production

- Orchestration and Scheduling with Multi-Task Jobs
- Monitoring, Logging, and Handling Errors
- Promoting Code with Databricks Repos
- Programmatic Platform Interactions
- Managing Costs and Latency with Incremental Workloads
- Deploying Streaming and Batch Workloads

Welcome!



Your Instructor - Your Name

-
-
-

/in/profile

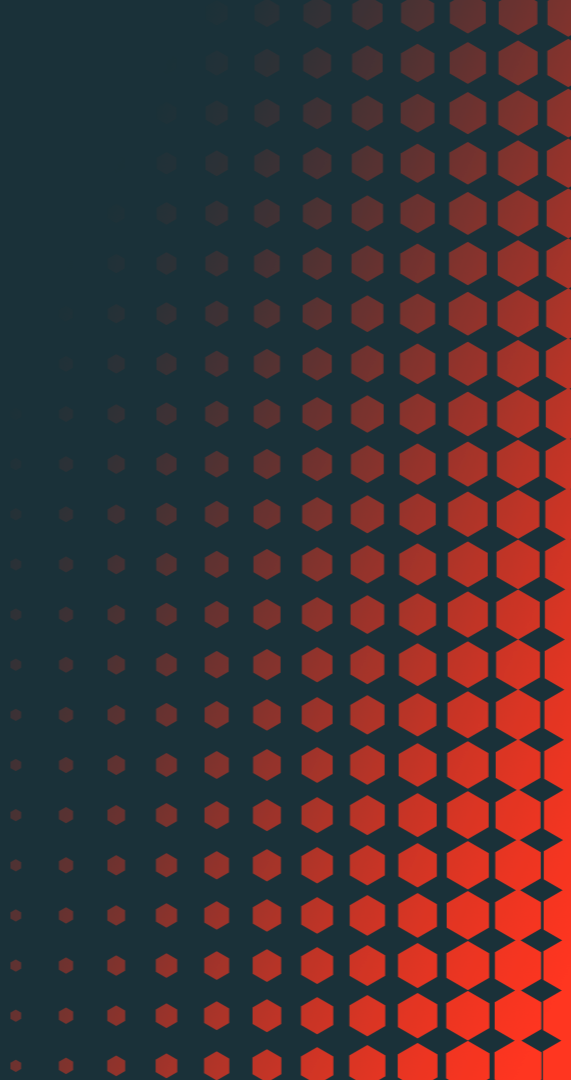
databricks

Welcome!

Let’s get to know you 

- Name
- Role and team
- Length of experience with Spark and Databricks
- Motivation for attending
- Fun fact or favorite mobile app

Architecting for the Lakehouse



databricks

Module Objectives

1. Describe the nuances of the data lakehouse and guarantees of Delta Lake
2. Design databases and tables optimized for production use cases
3. Design ELT pipelines that leverage the strengths of Delta Lake and Databricks

Agenda

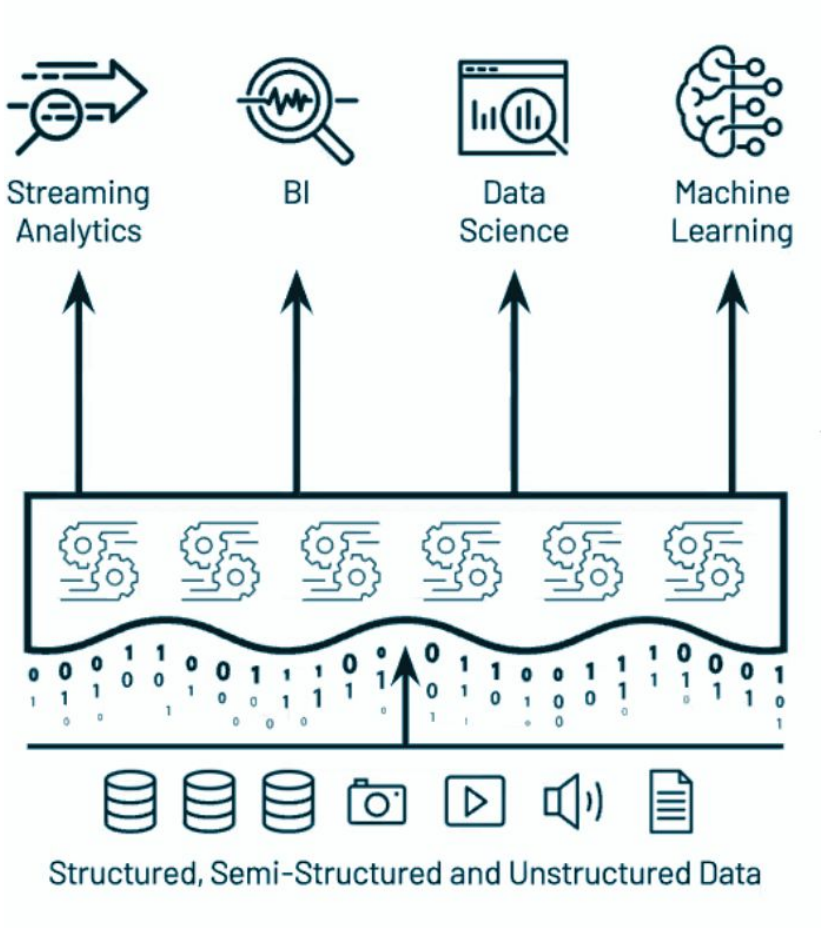
- Course Intro & Overview
- Adopting the Lakehouse Architecture
- The Lakehouse Medallion Architecture
- Setting Up Tables
- Optimizing Data Storage
- Understanding Delta Lake Transactions
- Delta Lake Isolation with Optimistic Concurrency
- Streaming Design Patterns

Course Application and Dataset

Design and implement a multi-pipeline multi-hop architecture to enable the Lakehouse paradigm.

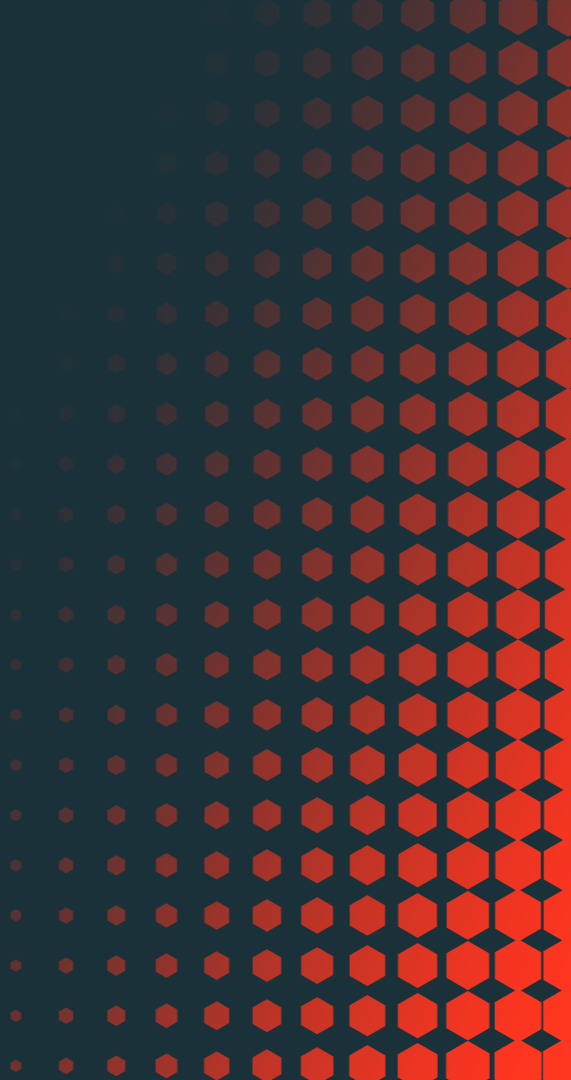
Our Company

moovio



databricks

Adopting the Lakehouse Architecture



databricks

Data Lake

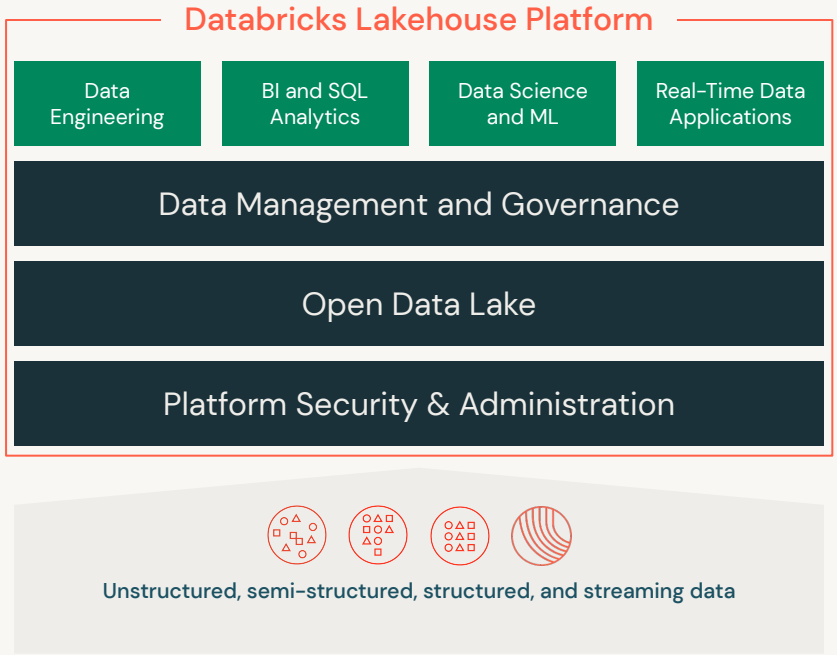
Lakehouse

One platform to unify all of your data, analytics, and AI workloads

Data Warehouse

The Databricks Lakehouse Platform

- Simple
- Open
- Collaborative



Data Lake

DELTA LAKE

An open approach to bringing **data management and governance** to data lakes

Better reliability with transactions

48x faster data processing with indexing

Data governance at scale with fine-grained access control lists

Data Warehouse

Delta Lake brings ACID to object storage

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Delta Lake provides ACID guarantees scoped to tables

Problems solved by ACID transactions in Delta

1. Hard to append data
2. Modification of existing data difficult
3. Jobs failing mid way
4. Real-time operations hard
5. Costly to keep historical data versions

The Lakehouse Medallion Architecture

Multi-hop Pipeline

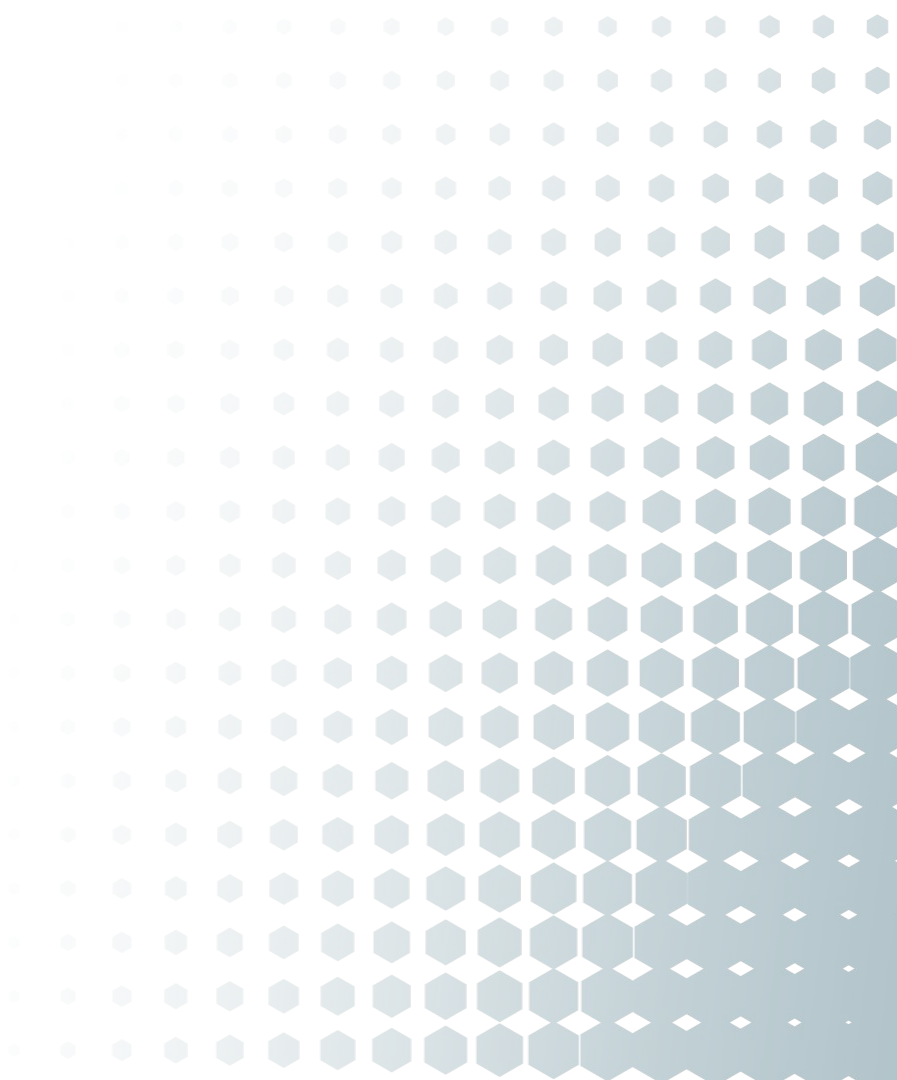
Source:
Files or integrated systems

Bronze:
Raw data and metadata

Silver:
Validated data with atomic grain

Gold:
Refined, aggregated data

Bronze Layer



Why is the Bronze Layer Important?

- Bronze layer replaces the traditional data lake
- Represents the full, unprocessed history of the data
- Captures the provenance (what, when, and from where) of data loaded into the lakehouse
- Data is stored efficiently using Delta Lake
- If downstream layers discover later they need to ingest more, they can come back to the Bronze source to obtain it.

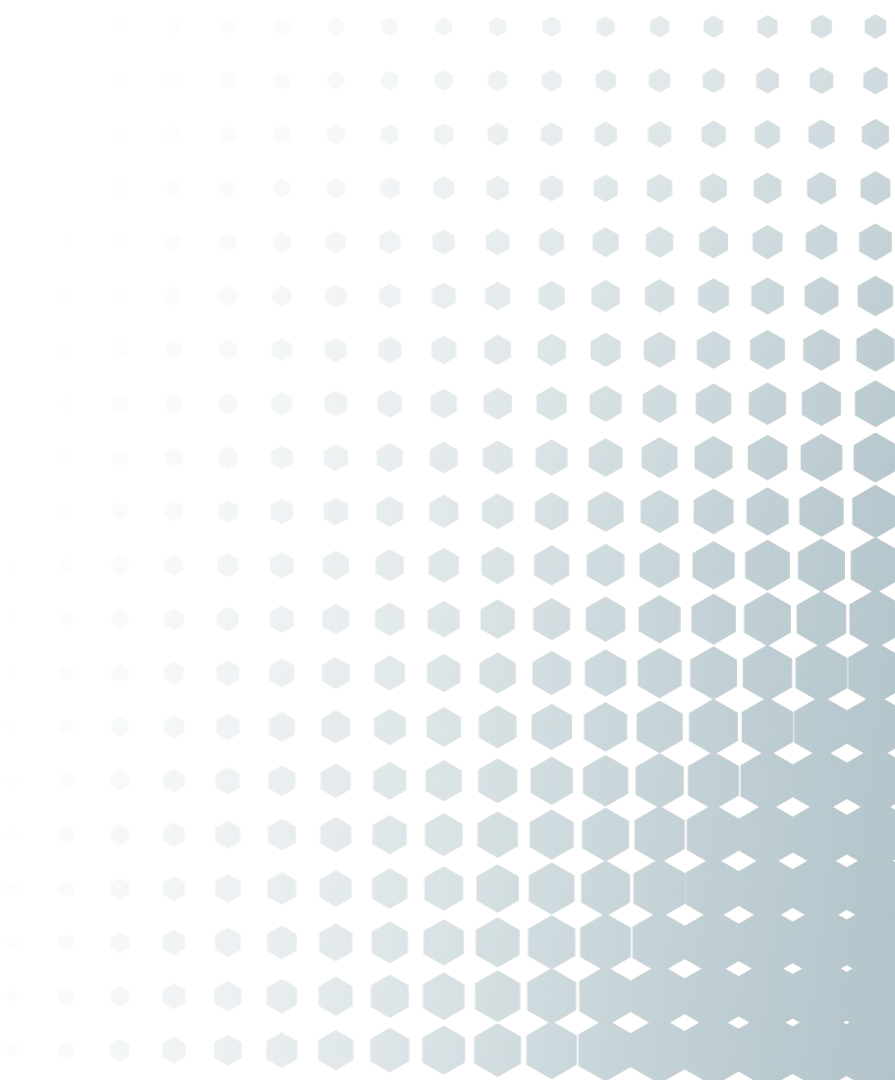
Bronze Layer Guiding Principles

- The goal of this layer is data capture and provenance:
  - Capture exactly what is ingested, without parsing or change.
- Typically a Delta Lake table with these fields in each row:
  - Date received/ingested
  - Data source (filename, external system, etc)
  - Text field with raw unparsed JSON, CSV, or other data
  - Other metadata
- Should be append only (batch or streaming)
- Plan ahead if data must be deleted for regulatory purposes

Processing Deletes

- Retain all records when possible
- Soft-deletes if necessary
- Hard-deletes may be required by regulatory processes

Silver Layer



databricks

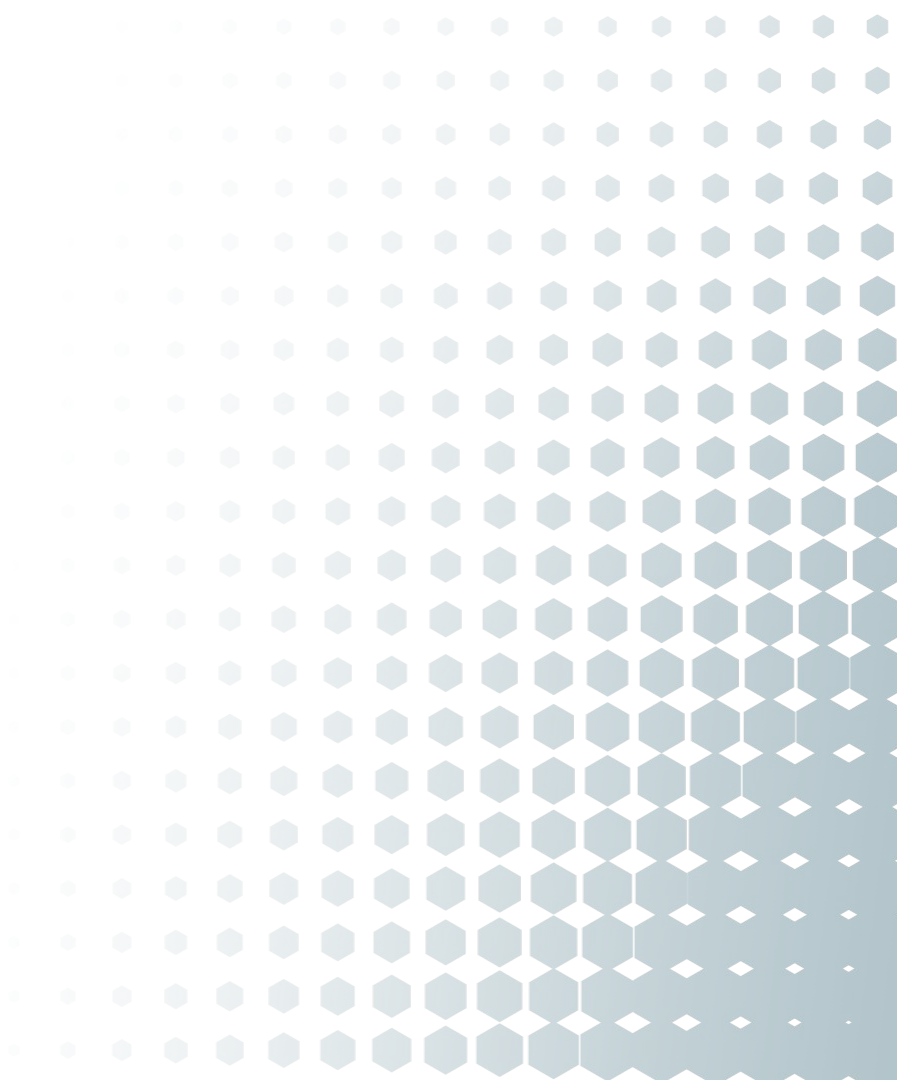
Why is the Silver Layer important?

- Easier to query than the non-curated Bronze “data lake”
  - Data is clean
  - Transactions have ACID guarantees
- Represents the “Enterprise Data Model”
- Captures the full history of business action modeled
  - Each record processed is preserved
  - All records can be efficiently queried
- Reduces data storage complexity, latency, and redundancy
  - Built for both ETL throughput AND analytic query performance

Silver Layer Guiding Principles

- Uses DeltaLake tables (with SQL table names)
- Preserves grain of original data (no aggregation)
- Eliminates duplicate records
- Production schema enforced
- Data quality checks passed
- Corrupt data quarantined
- Data stored to support production workloads
- Optimized for long-term retention and ad-hoc queries

Gold Layer



databricks

Why is the Gold Layer important?

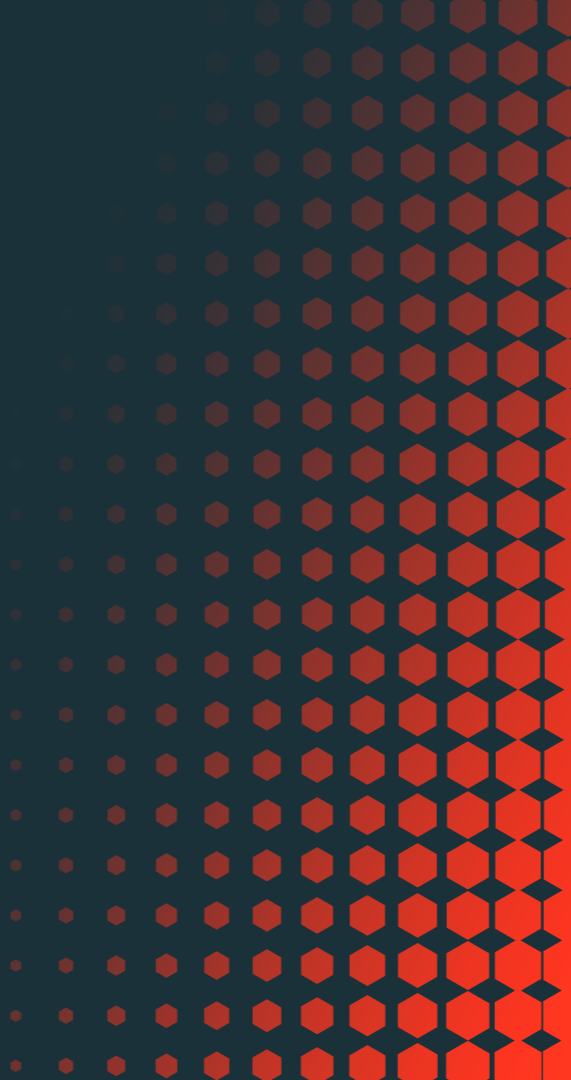
- Powers ML applications, reporting, dashboards, ad hoc analytics
- Reduces costs associated with ad hoc queries on silver tables
- Allows fine grained permissions
- Reduces strain on production systems
- Shifts query updates to production workloads

Notebook: Setting Up Tables

Notebook: Optimizing Data Storage

Notebook: Understanding Delta Lake Transactions

Isolation with Optimistic Concurrency Control



Optimistic Concurrency Control

1. Read
2. Write
3. Validate and commit (fail transaction/update if validation fails)

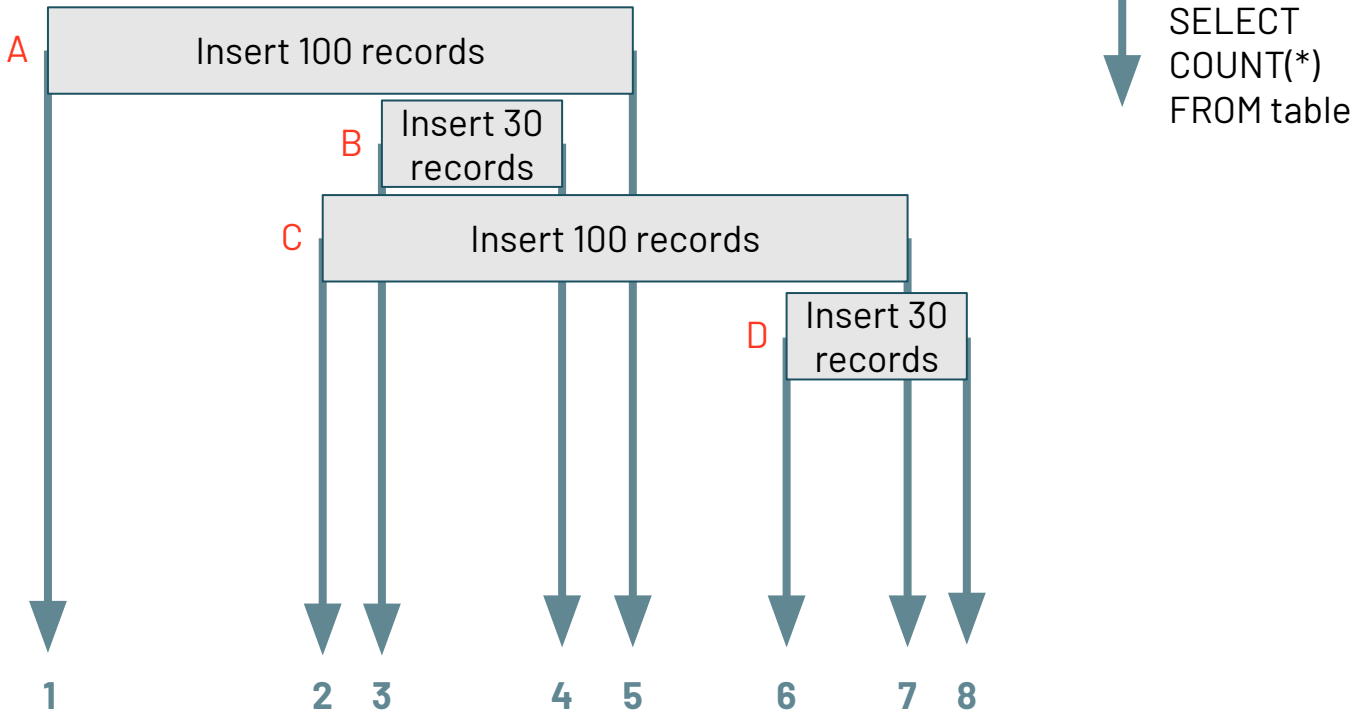
Deadlock on Databricks?

- **Not** a concern in the Lakehouse
- All reads use Snapshot isolation
- Setting Serializable isolation for Delta Lake tables will prevent concurrent writes from succeeding
- Other issues not seen:
  - Dirty reads, non-repeatable reads, phantom reads

What is Isolation?

- Determines when and how transactions become visible to other users and systems
  - Delta Lake uses the transaction log to control isolation
  - A transaction commits when a log file is written to the transaction log
- Concerned with concurrency effects
- Exists as a spectrum, referred to as “isolation levels”
- Typically defined at database level
- **Delta Lake defines isolation at the table level (doesn’t span multiple tables)**

Concurrent Inserts



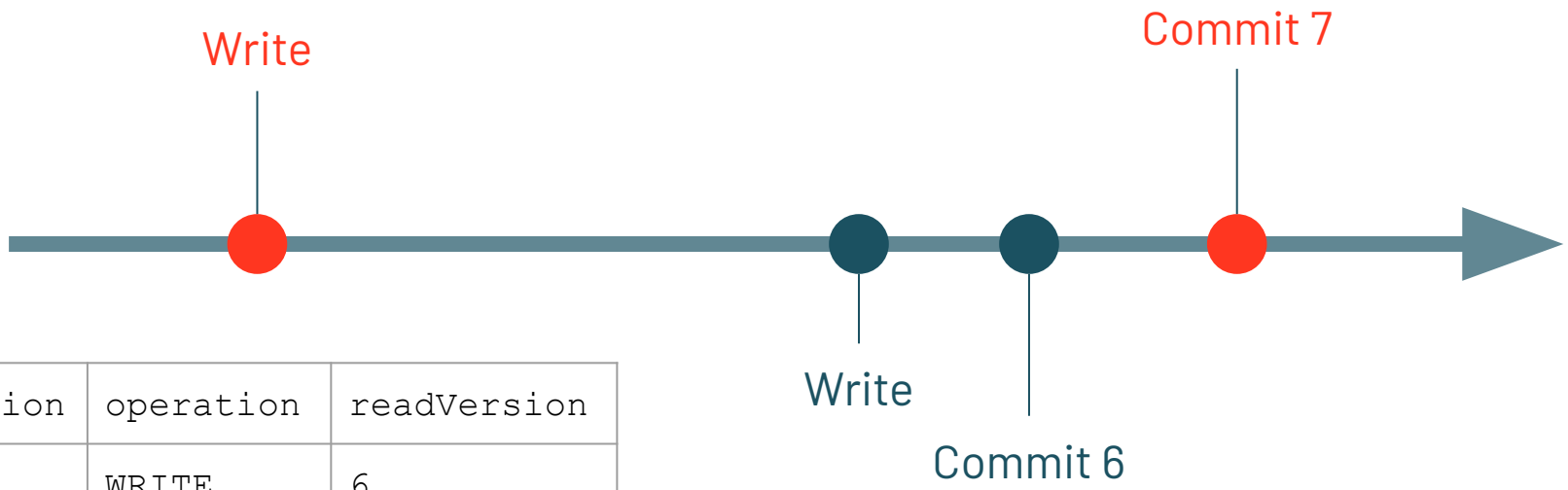
Write Conflicts with WriteSerializable

	INSERT	UPDATE, DELETE, MERGE INTO	OPTIMIZE
INSERT	**Cannot conflict**		
UPDATE, DELETE, MERGE INTO	**Cannot conflict**	**Can conflict**	
OPTIMIZE	**Cannot conflict**	**Can conflict**	**Can conflict**

Write Conflicts with Serializable

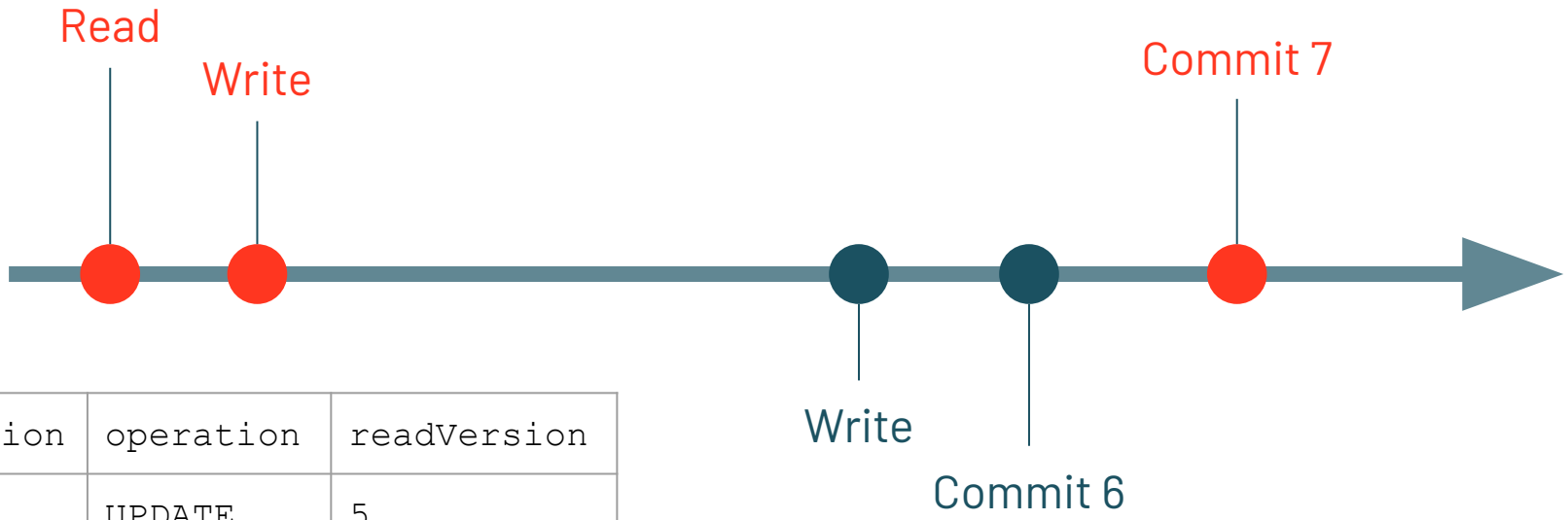
	INSERT	UPDATE, DELETE, MERGE INTO	OPTIMIZE
INSERT	**Cannot conflict**		
UPDATE, DELETE, MERGE INTO	**Can conflict**	**Can conflict**	
OPTIMIZE	**Cannot conflict**	**Can conflict**	**Can conflict**

Concurrent Inserts Can’t Conflict



version	operation	readVersion
7	WRITE	6
6	WRITE	5

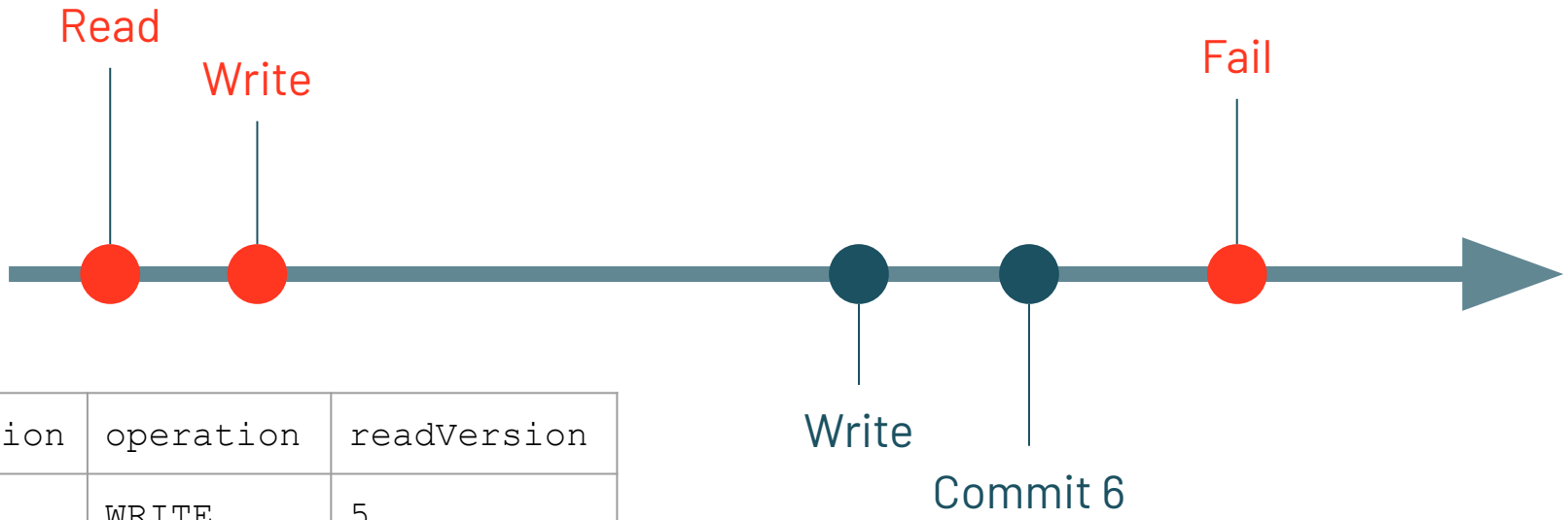
Concurrent Insert/Update in WriteSerialiable



version	operation	readVersion
7	UPDATE	5
6	WRITE	5

With Write-Serializable isolation:
The transaction log shows 6 then 7 (commit order), but the logically correct serializable order is 7 then 6 because 7's update doesn't update 6's new data.

Concurrent Insert/Update in Serializable



version	operation	readVersion
6	WRITE	5

With Serializable isolation:
The second transaction to attempt to commit may fail.

Delta Lake WriteSerializable

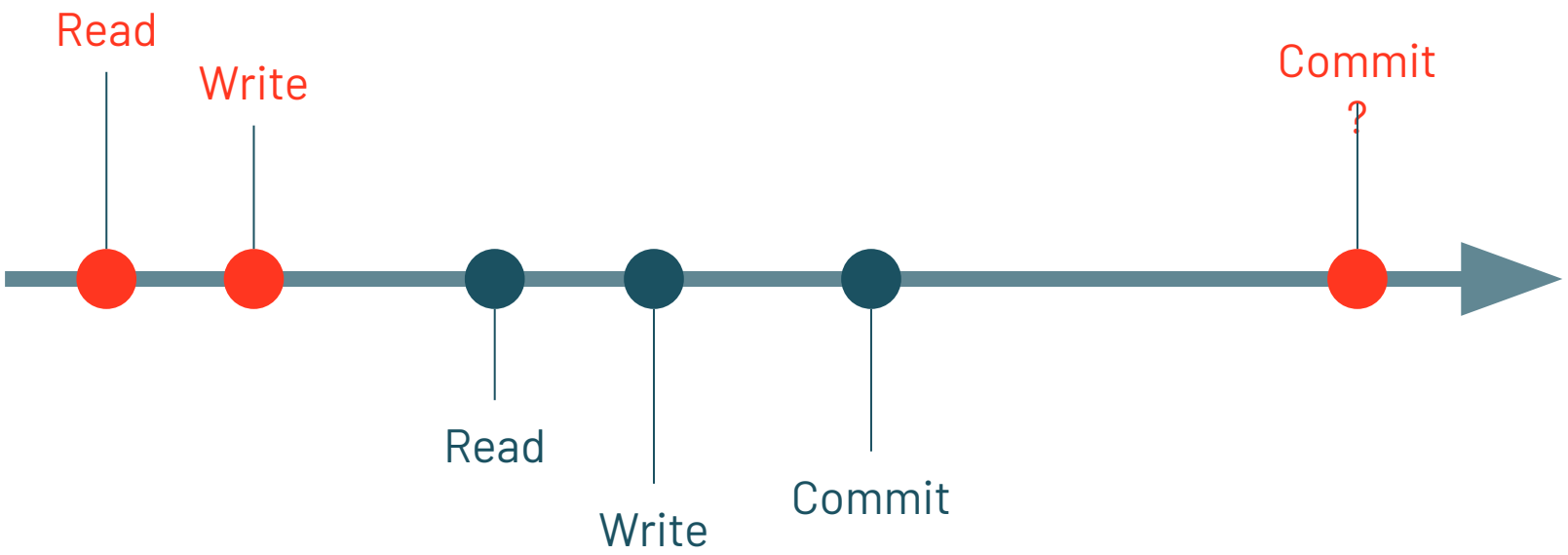
- Slightly relaxes strict serial guarantees to increase throughput
- Append operations will never conflict or block concurrent execution
- Default in Delta Lake

Delta Lake Serializable

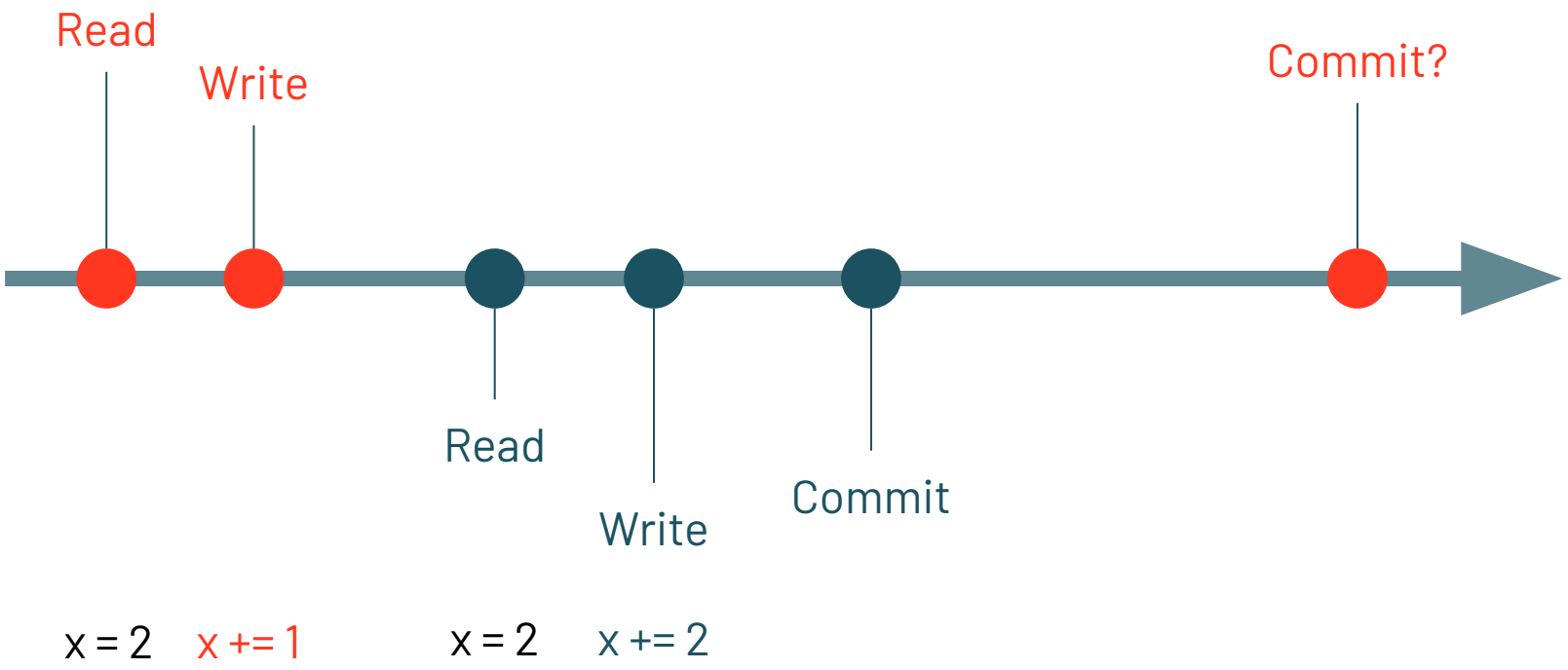
- Ensures committed write operations and all reads are Serializable
- Only allows operations that could be executed one-at-a-time to achieve same outcome as seen in table
- Prevents incorrectly ordered operations through commit failure

```
ALTER TABLE <table-name> SET TBLPROPERTIES
  ('delta.isolationLevel' = 'Serializable')
```

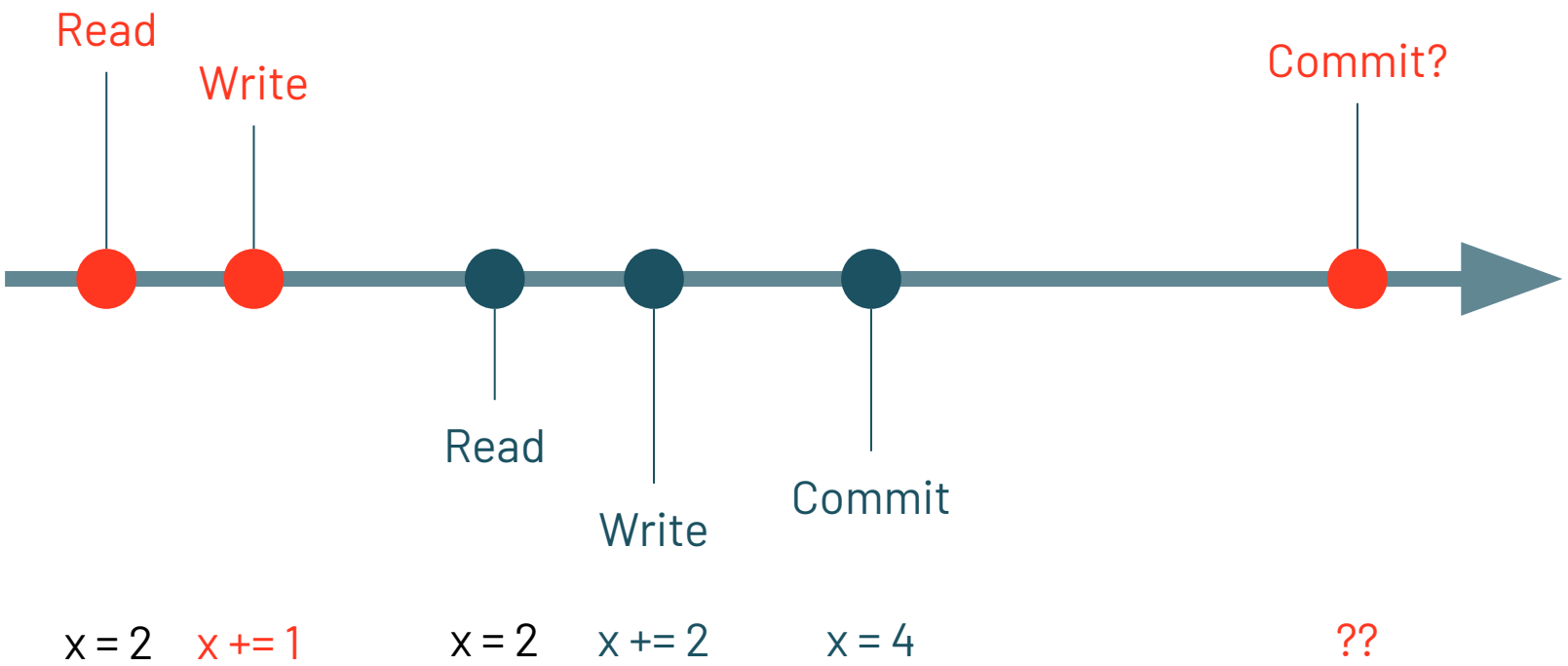
Conflicting Update



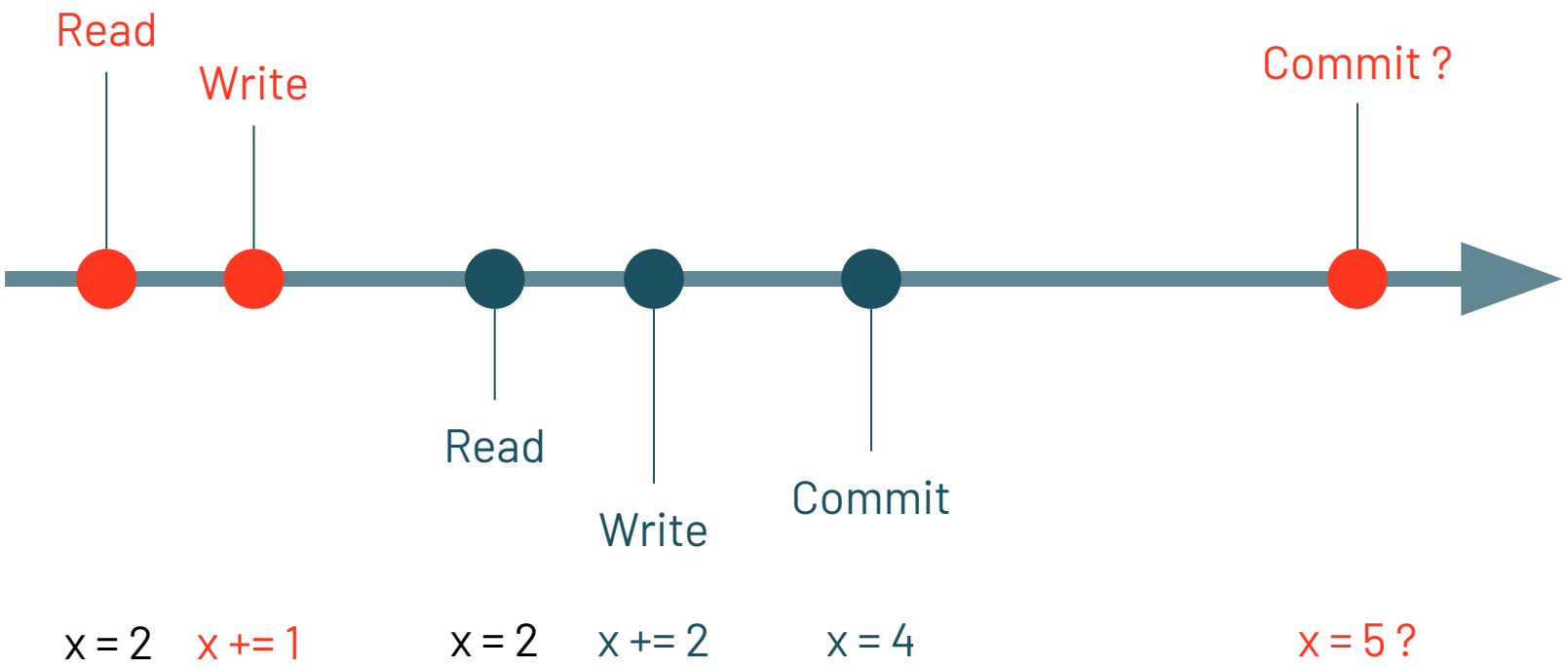
Conflicting Update



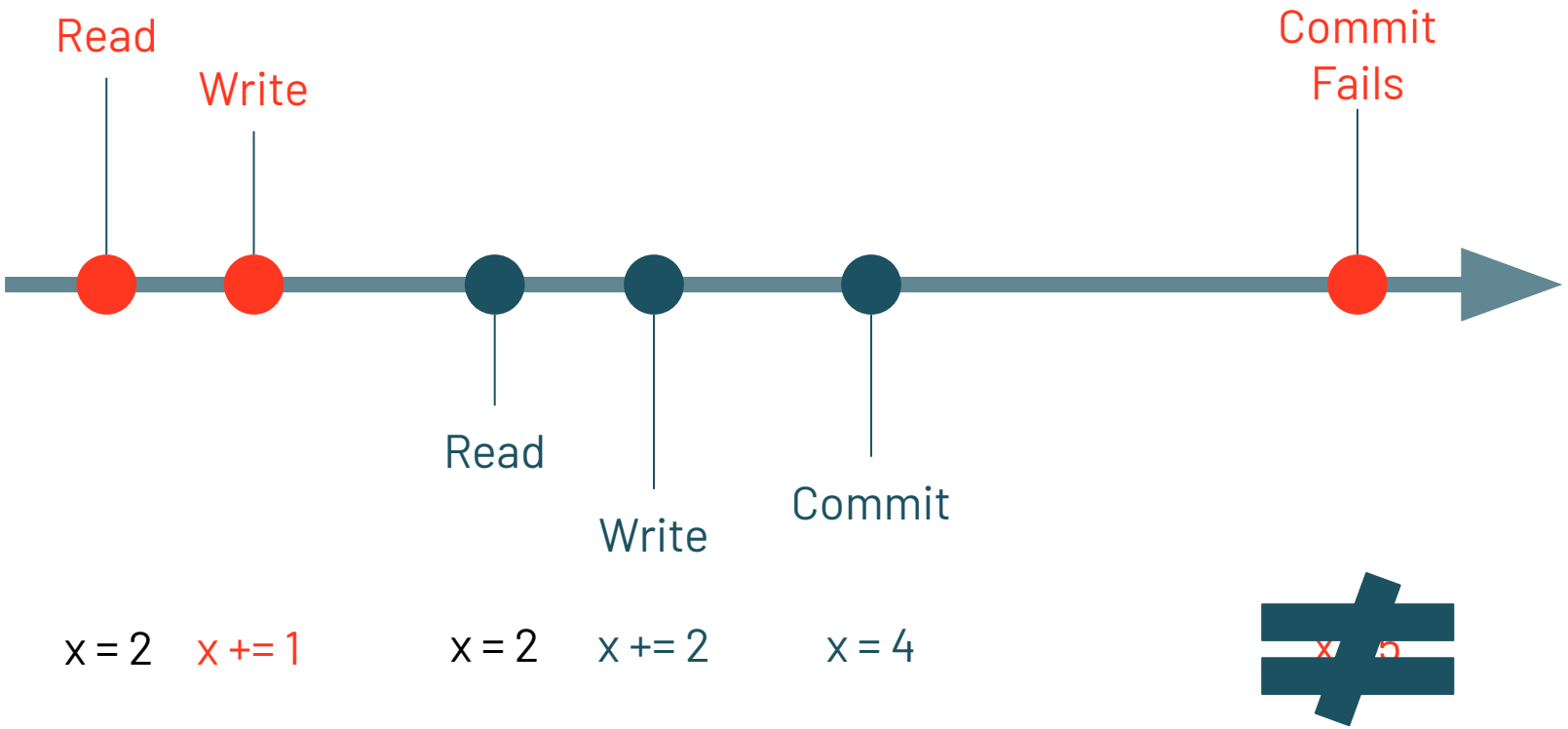
Conflicting Update



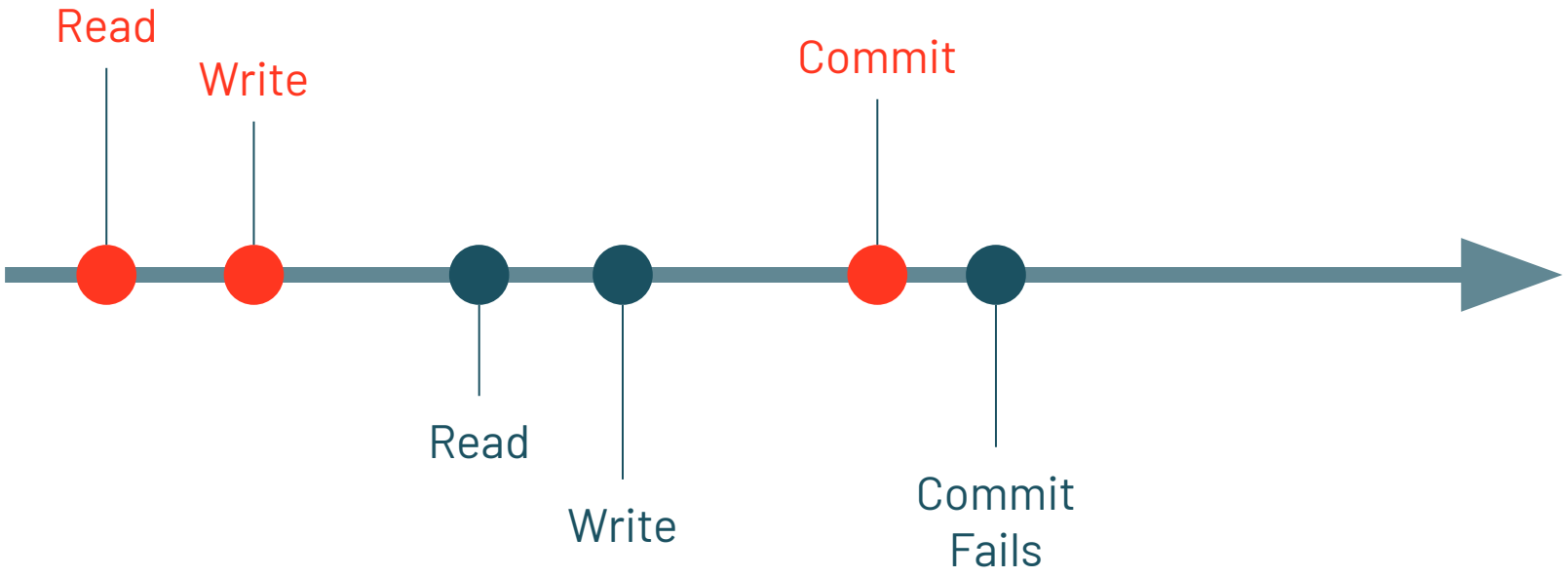
Conflicting Update



Conflicting Update



Conflicting Update (Alternate)



Recommendations to Avoid Conflicts

- Use WriteSerializable instead of Serializable
- Process all changes to a target table through one pipeline
- If multiple jobs must update a target table, avoid scheduling long running or widely scoped updates at the same time
- Turn on retry logic when scheduling jobs to automatically retry on job failure
- Use Delta Live Tables and new DBRs to benefit from latest features

Read the Docs

- Concurrency control
- Isolation levels
- Conflict exceptions

Notebook: Streaming Design Patterns

Design streaming jobs in light of what we've learned about transactions

Module Recap

1. Describe the nuances of the data lakehouse and guarantees of Delta Lake
2. Design databases and tables optimized for production use cases
3. Design ELT pipelines that leverage the strengths of Delta Lake and Databricks