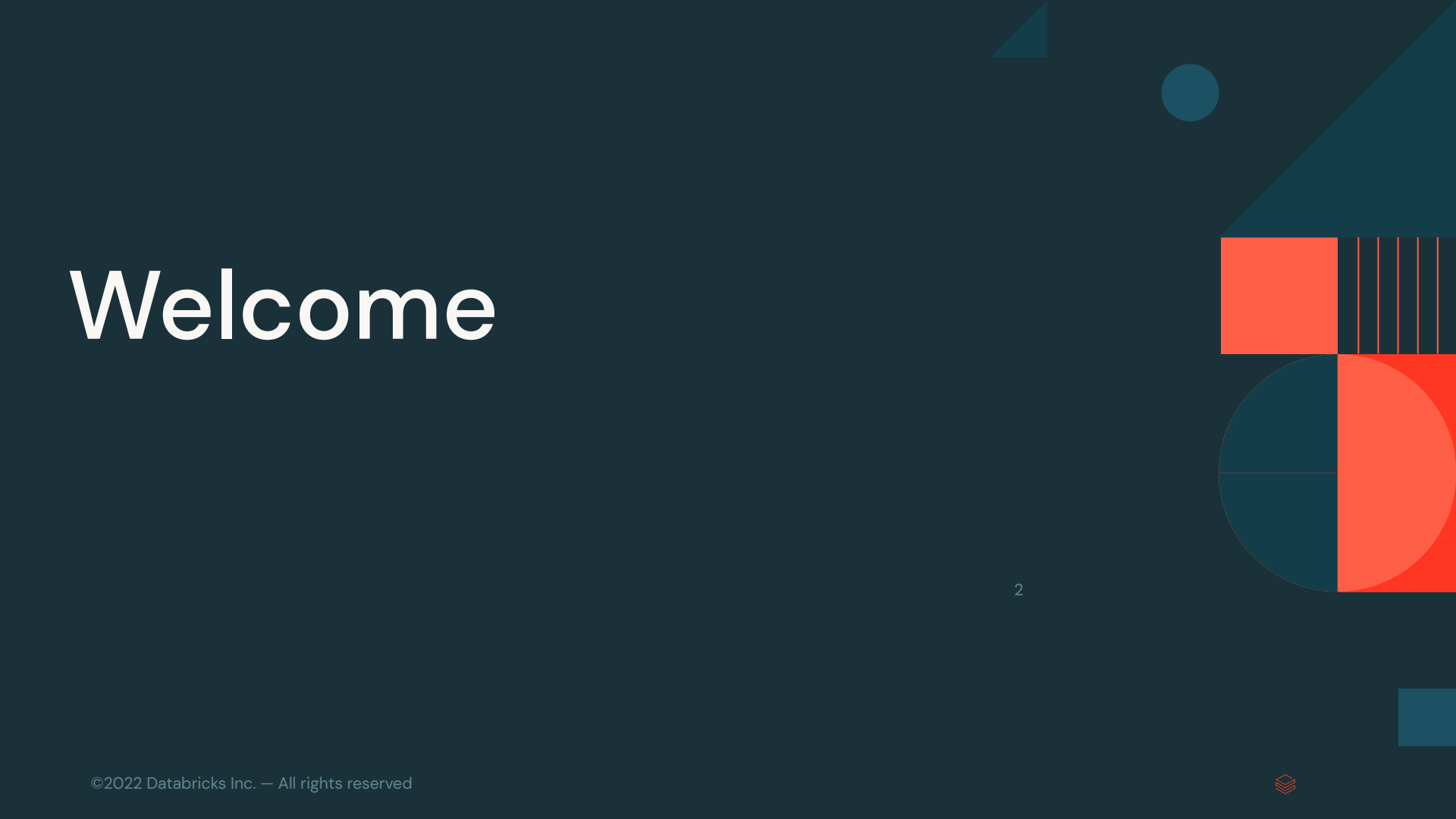
databricks

Apache Spark Programming

With Databricks

Welcome



Course Agenda

Here's where we're headed

DataFrames	Introduction	Databricks Platform	Spark SQL	Reader & Writer	DataFrame & Column
Transformations	Aggregation	Datetimes	Complex Types	Additional Functions	User-Defined Functions
Performance	Spark Architecture	Query Optimization	Partitioning		
Structured Streaming and Delta	Streaming Query	Aggregating Streams	Delta Lake		

Lesson Objectives

By the end of this course, you should be able to:

1. Identify core features of Spark and Databricks
2. Describe how DataFrames are created and evaluated in Spark
3. Apply the DataFrame API to process and analyze data
4. Demonstrate how Spark is optimized and executed on a cluster
5. Apply Delta Lake and Structured Streaming to process data

Module 1

Introductions

Welcome!

Let’s get to know you 

- Name
- Role and team
- Programing experience
- Motivation for attending
- Personal interest

Module 2

Spark Core

databricks

Spark Core

Databricks Ecosystem
Spark Overview

Spark SQL

Reader & Writer

DataFrame & Column

Databricks Ecosystem

Lakehouse

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

Customers

5000+

across the globe

Original creators of:

Apache Spark™ · Delta Lake · mlflow™

Data Lake

Lakehouse

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

Data Warehouse

Data Lake

DELTA LAKE

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

Better reliability with transactions

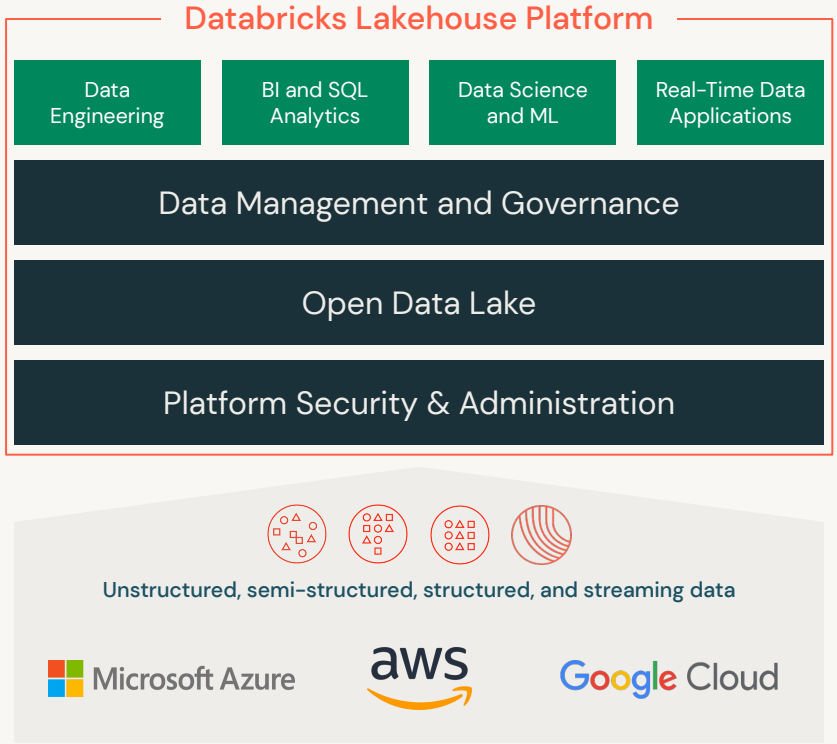
48x faster data processing with indexing

Data governance at scale with fine-grained access control lists

Data Warehouse

The Databricks Lakehouse Platform

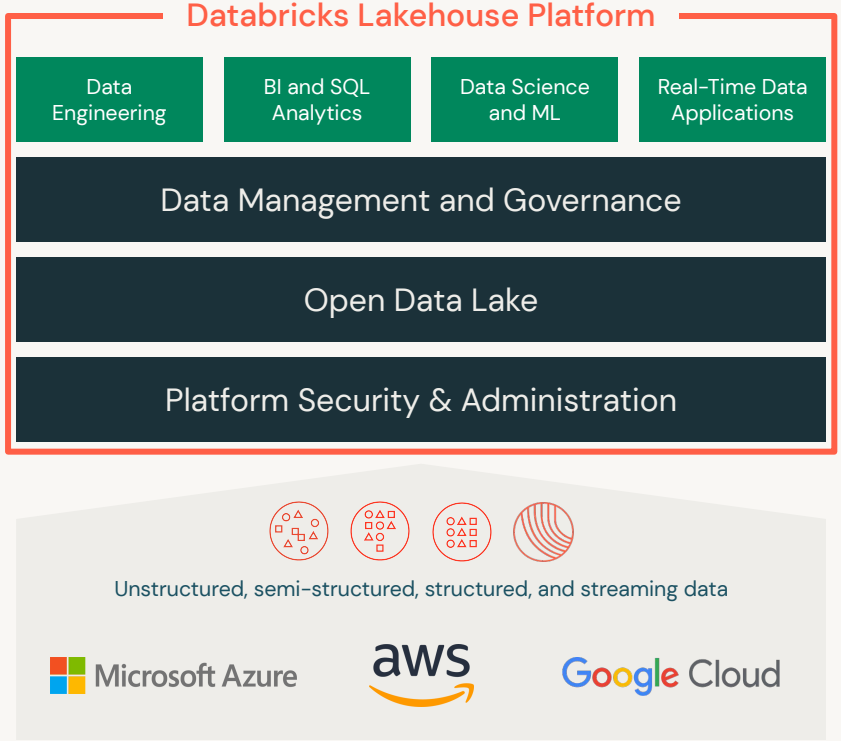
- Simple
- Open
- Collaborative



The Databricks Lakehouse Platform

Simple

Unify your data, analytics, and AI on one common platform for all data use cases



The Databricks Lakehouse Platform

Open

Unify your data ecosystem with open source standards and formats.

Built on the innovation of some of the most successful open source data projects in the world

30 Million+

Monthly downloads

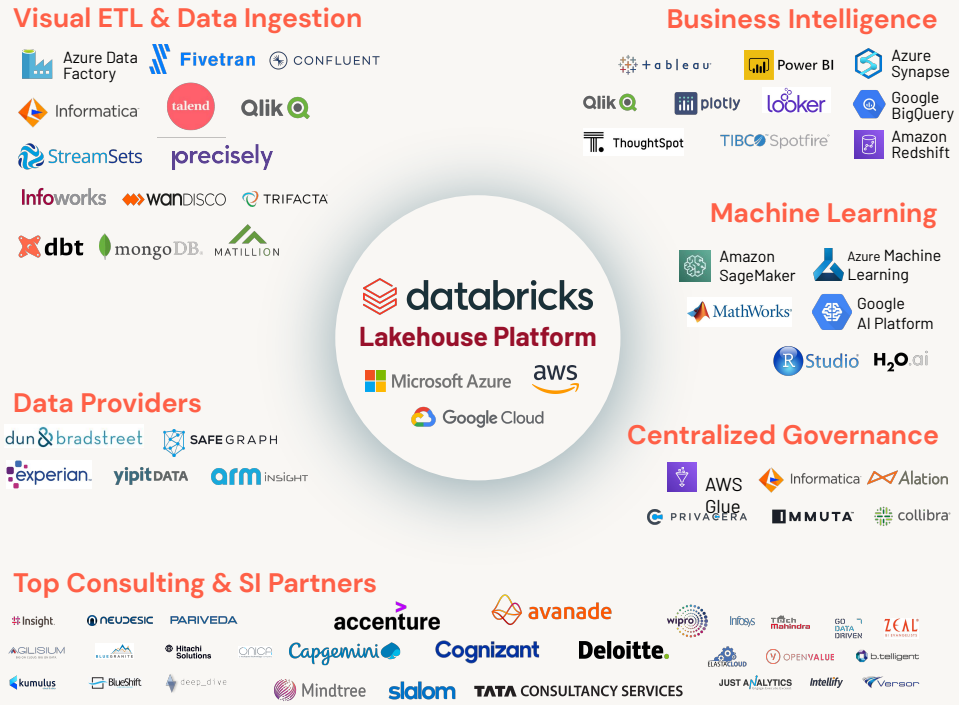
The Databricks Lakehouse Platform

Open

Unify your data ecosystem with open source standards and formats.

450+

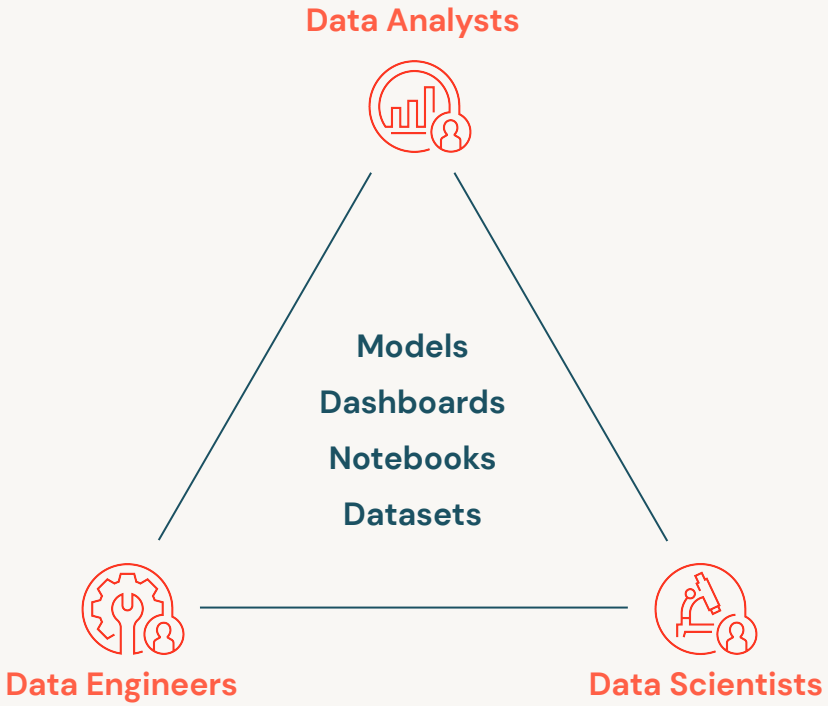
Partners across the data landscape



The Databricks Lakehouse Platform

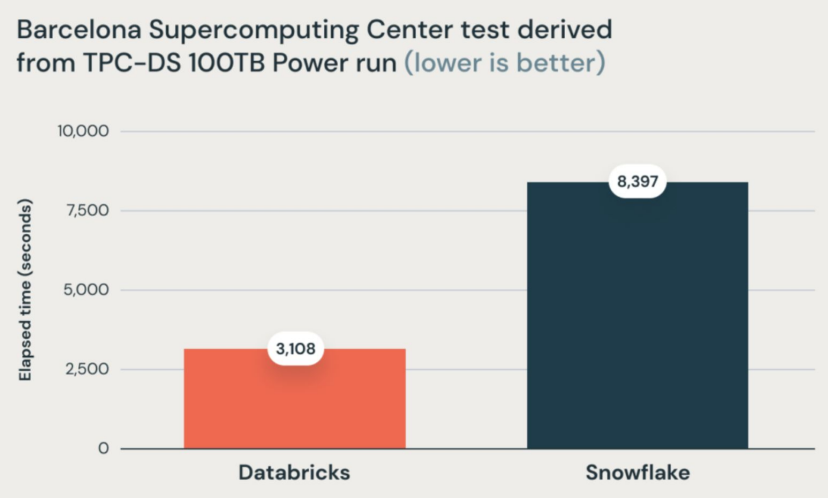
Collaborative

Unify your data teams to collaborate across the entire data and AI workflow



TPC-DS

Databricks SQL set **official data warehousing performance record** – outperformed the previous record by 2.2x.



Spark Overview

- De-facto standard unified analytics engine for big data processing

- Largest open-source project in data processing

- Technology created by the founders of Databricks

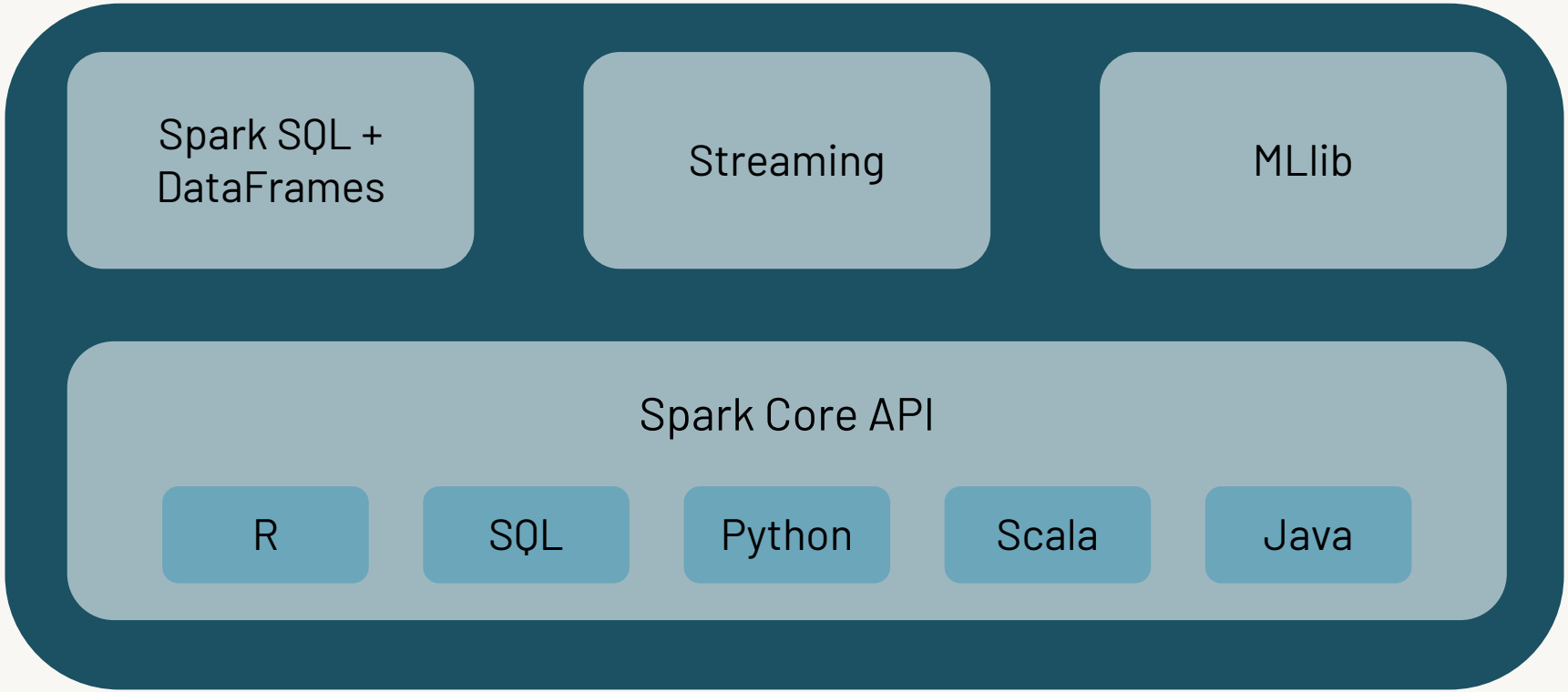
Spark Benefits

Fast

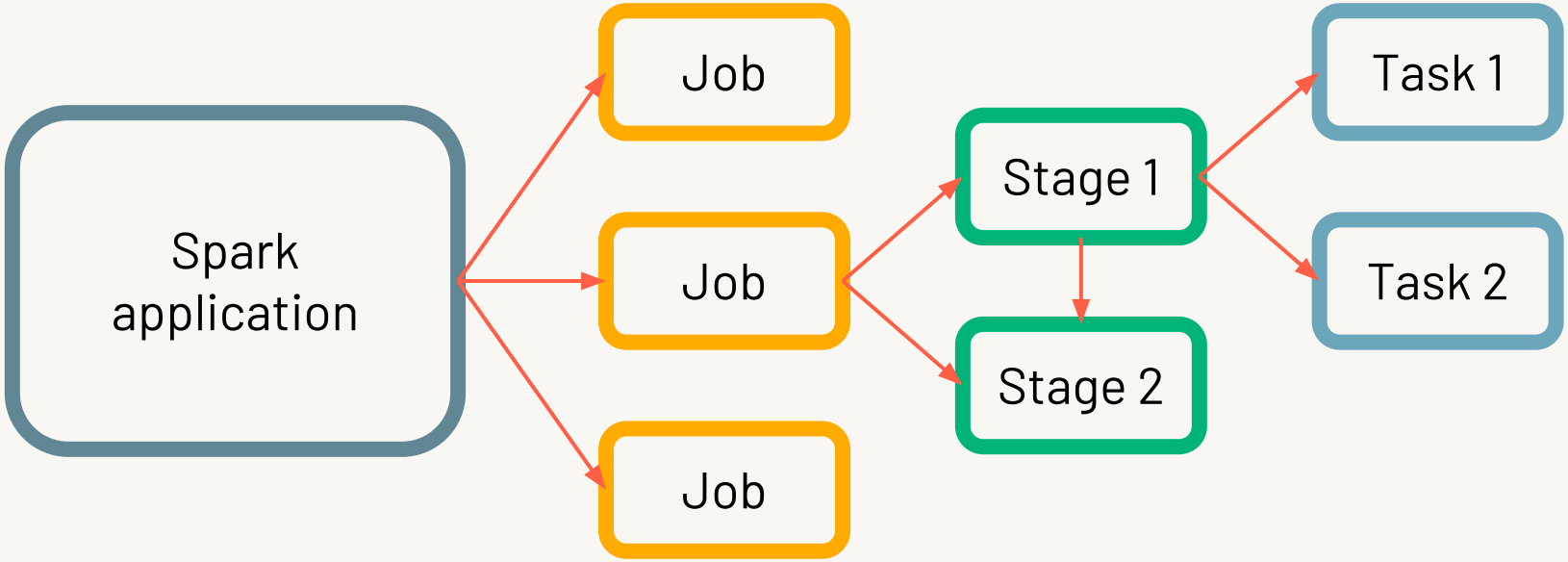
Easy to Use

Unified

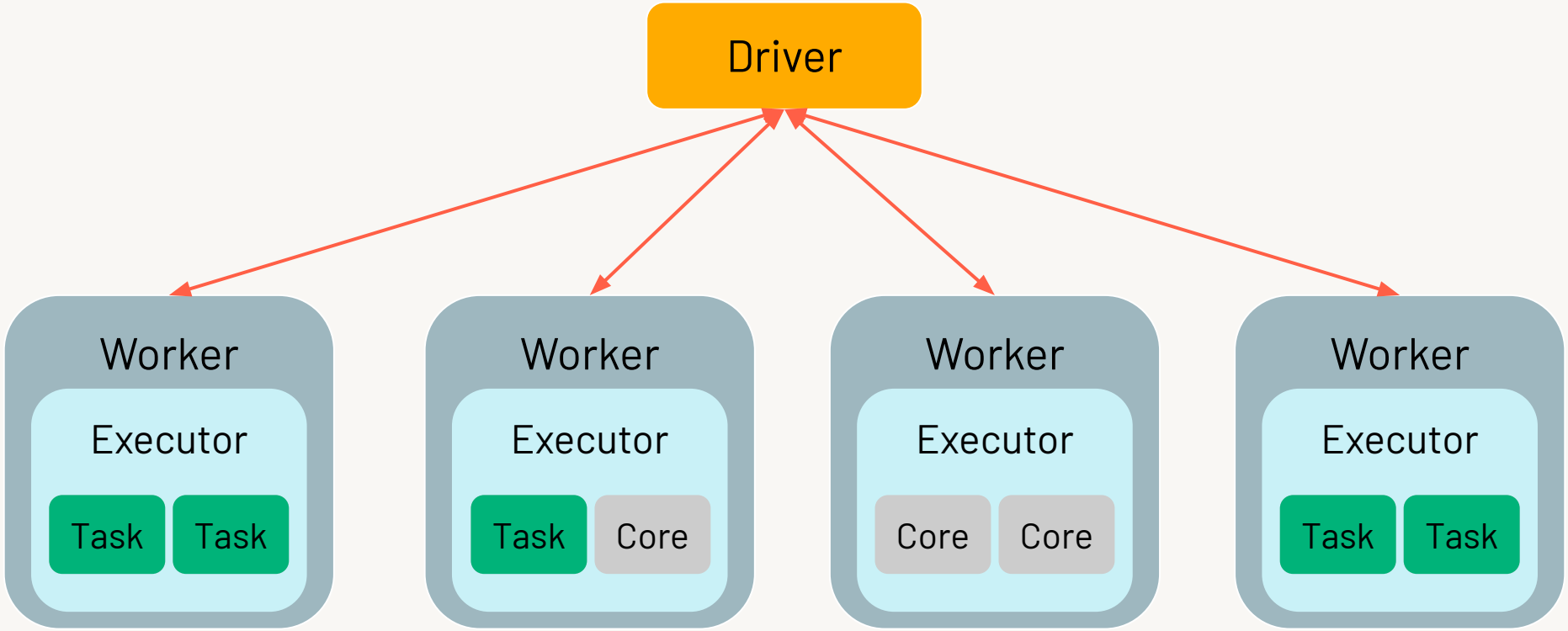
Spark API



Spark Execution



Spark Cluster



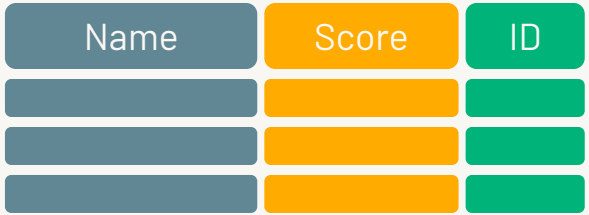
CSV

Comma Separated Values

```
item_id, name, price, qty
M_PREM_Q, Premium Queen Mattress, 1795, 35
M_STAN_F, Standard Full Mattress, 945, 24
M_PREM_F, Premium Full Mattress, 1695, 45
M_PREM_T, Premium Twin Mattress, 1095, 18
```


Parquet

A columnar storage format



Row-Oriented data on disk

Kit	4.2	1	Alex	4.5	2	Terry	4.1	3

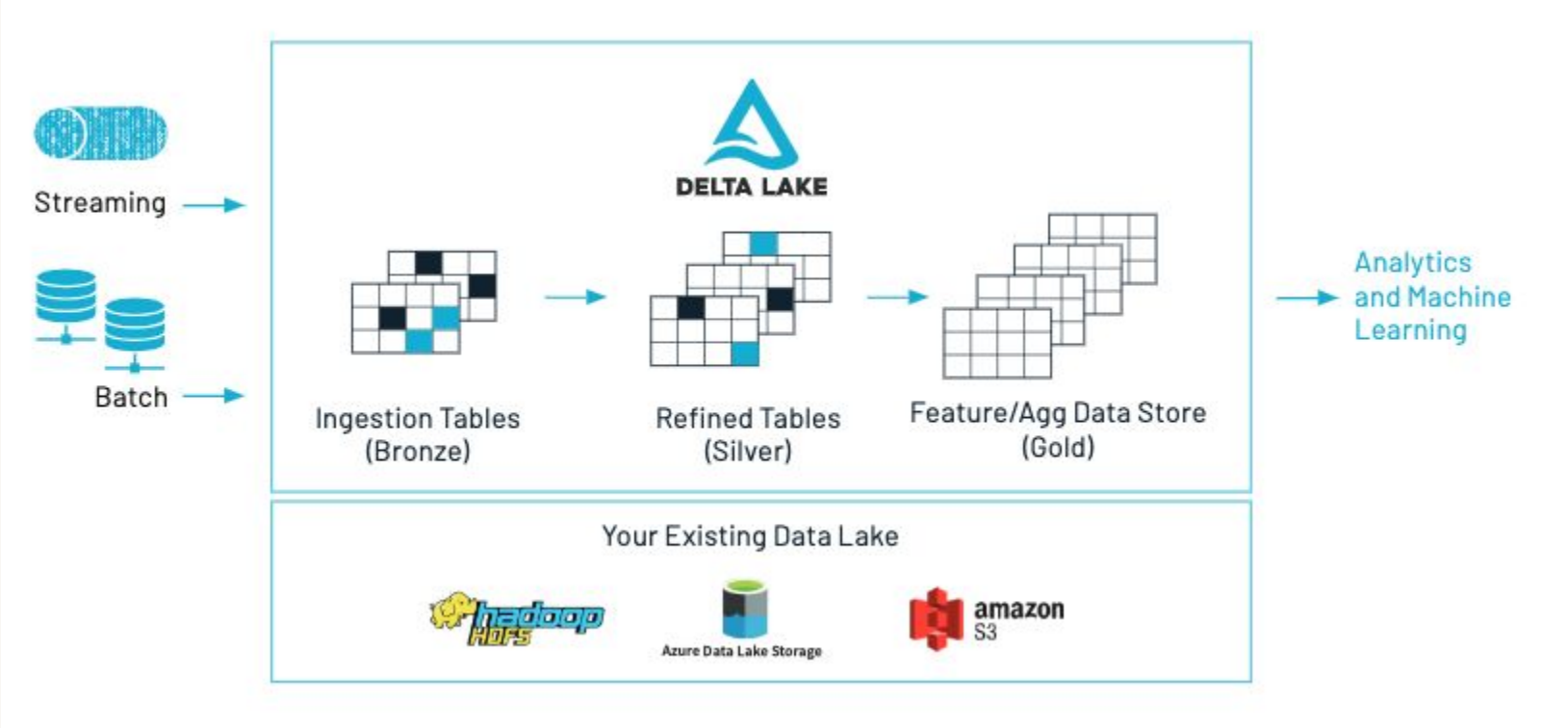
Column-Oriented data on disk

Kit	Alex	Terry	4.2	4.5	4.1	1	2	3

Delta Lake

Technology designed to be used with Apache Spark to build robust data lakes

Open-source Storage Layer



Delta Lake's Key Features

- ACID transactions
- Time travel (data versioning)
- Schema enforcement and evolution
- Audit history
- Parquet format
- Compatible with Apache Spark API

Module 3

Functions

databricks

Functions

Aggregation

Datetimes

Complex Types

Additional Functions

User-Defined Functions

Module 4

Performance

databricks

Performance

Spark Architecture

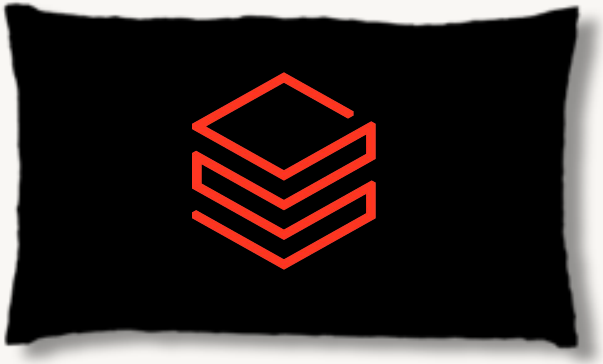
Query Optimization

Partitioning

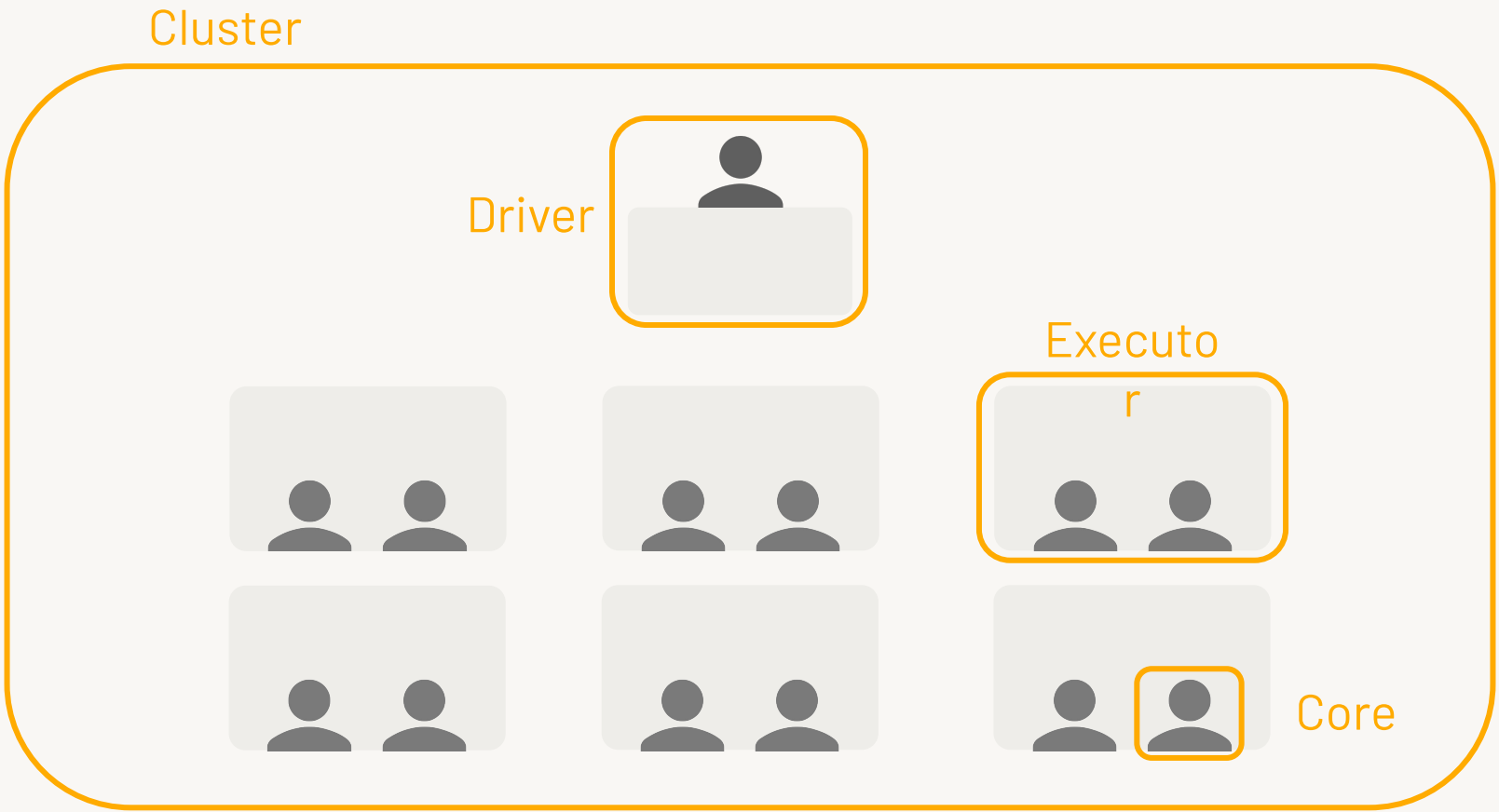
Spark Architecture

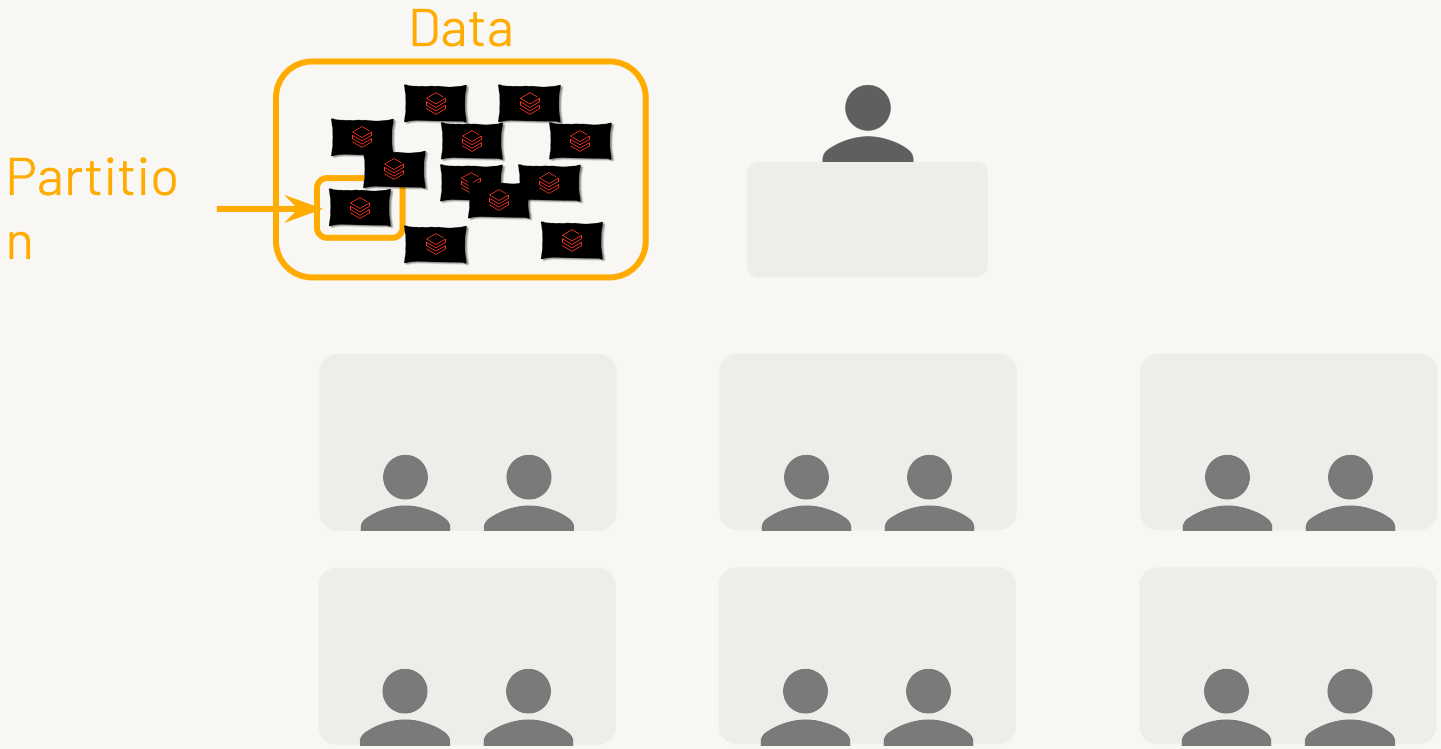
Spark Cluster

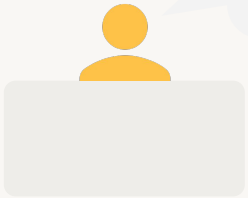
Spark Execution



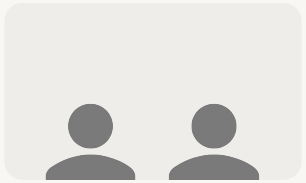
Scenario 1: Filter out brown pieces from candy bags

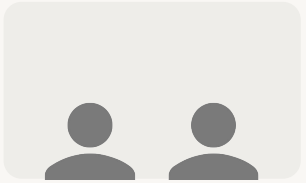


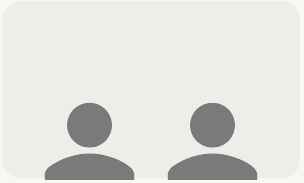


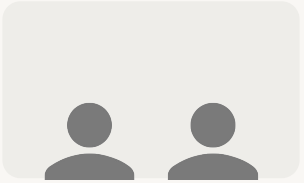


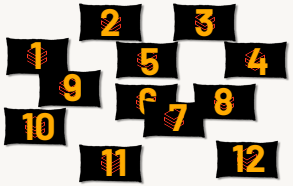
We need filter out brown pieces from these candy bags

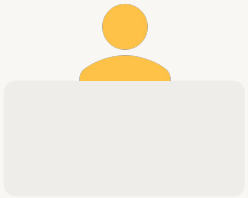


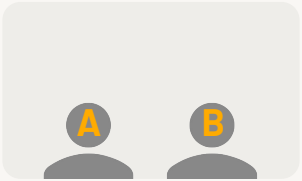


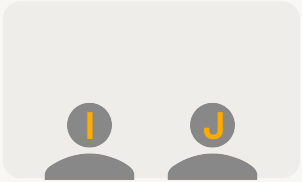


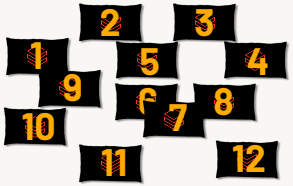


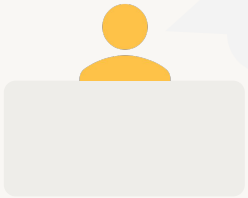




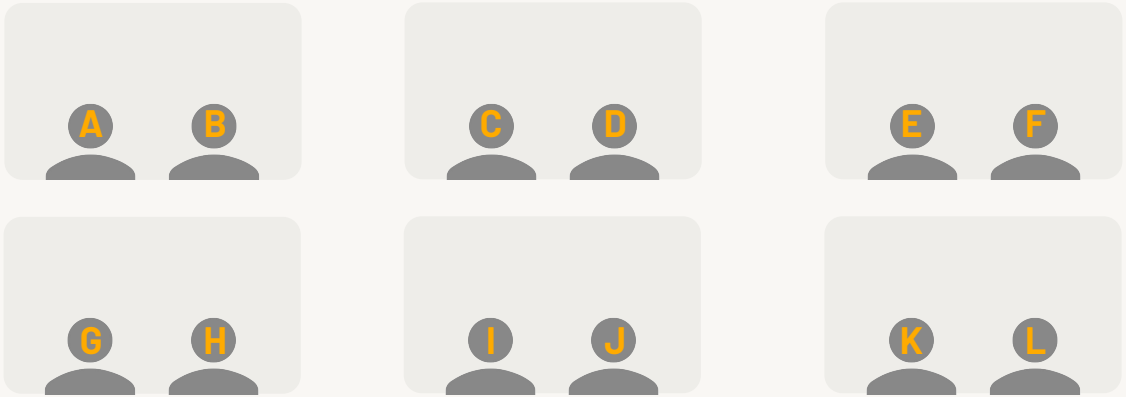


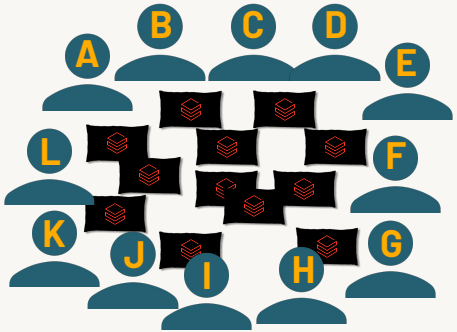


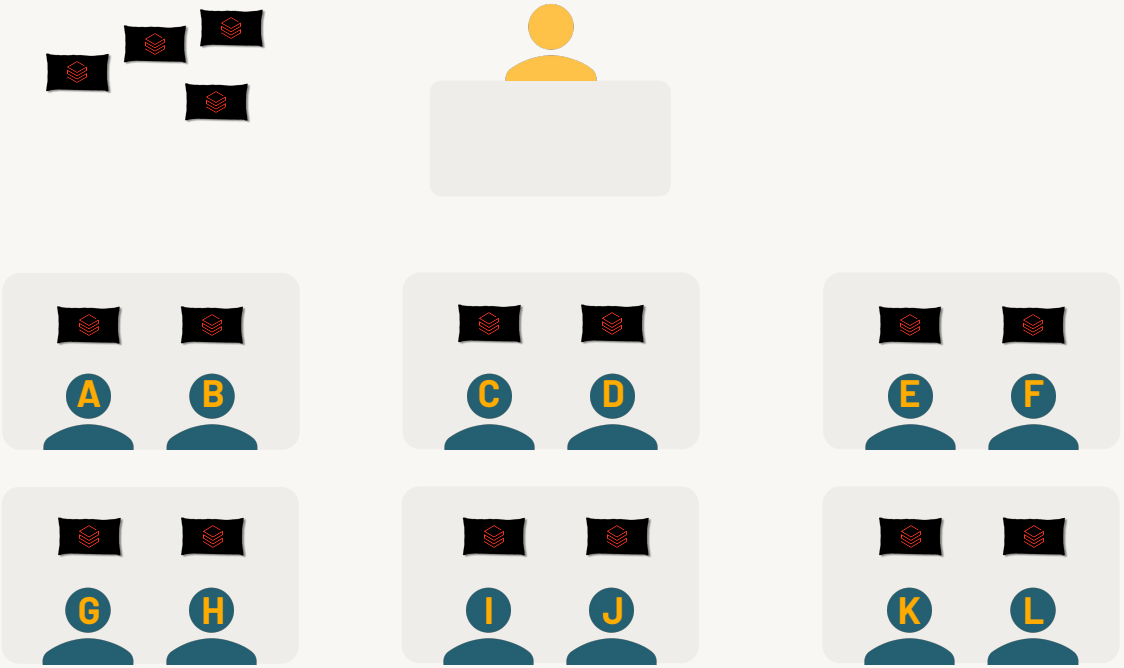


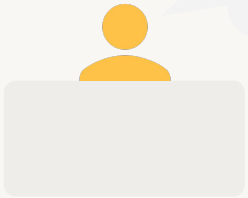


Student A get bag # 1
Student B get bag # 2
Student C get bag # 3
...

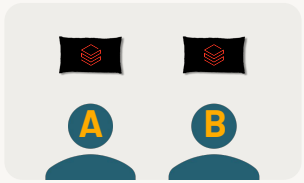


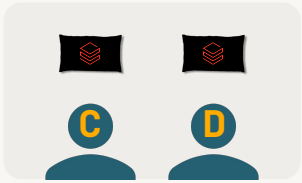


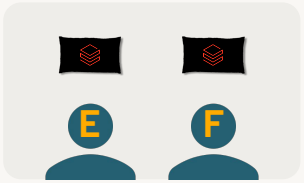


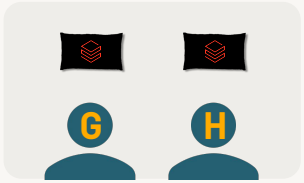


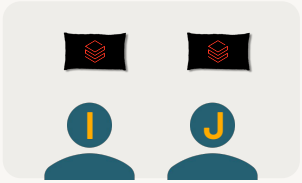
Eliminate the brown candy pieces and pile the rest in the corner

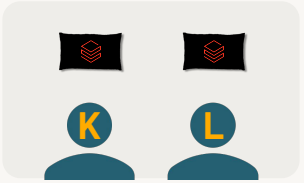


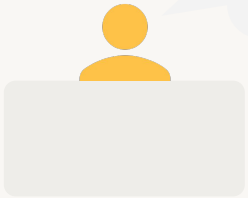




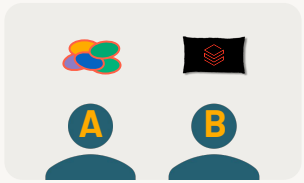


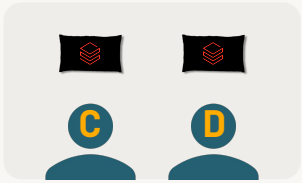


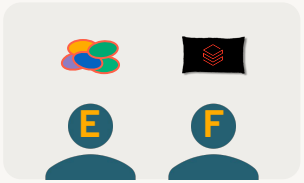


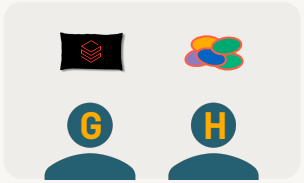


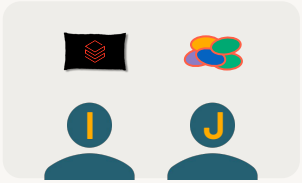
Eliminate the brown candy pieces and pile the rest in the corner

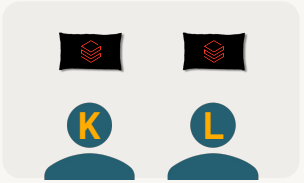


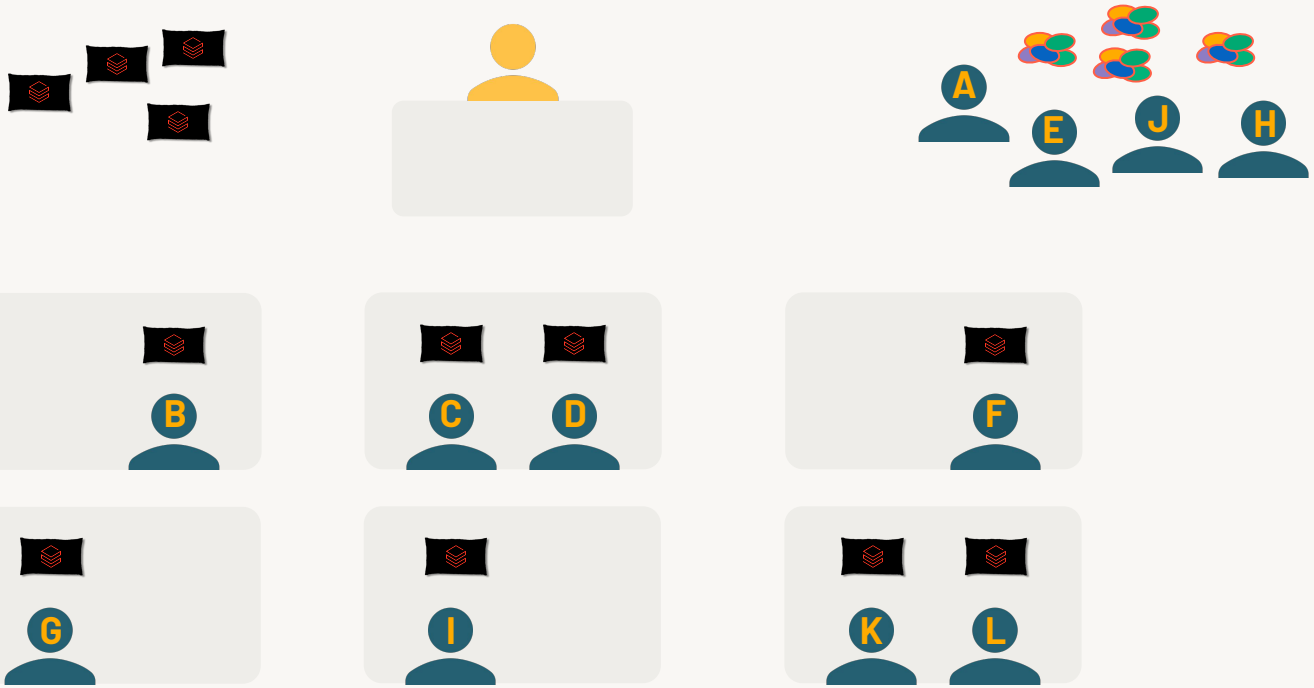


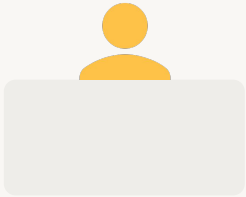


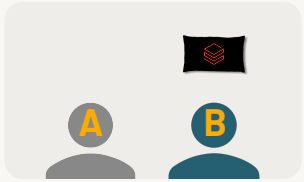


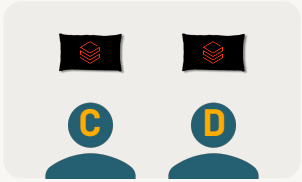


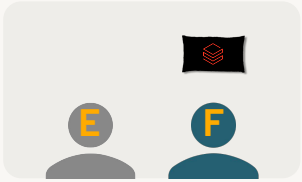


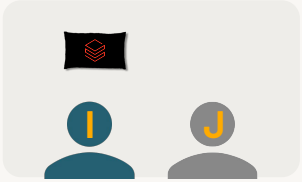


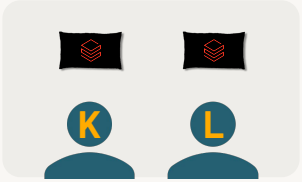


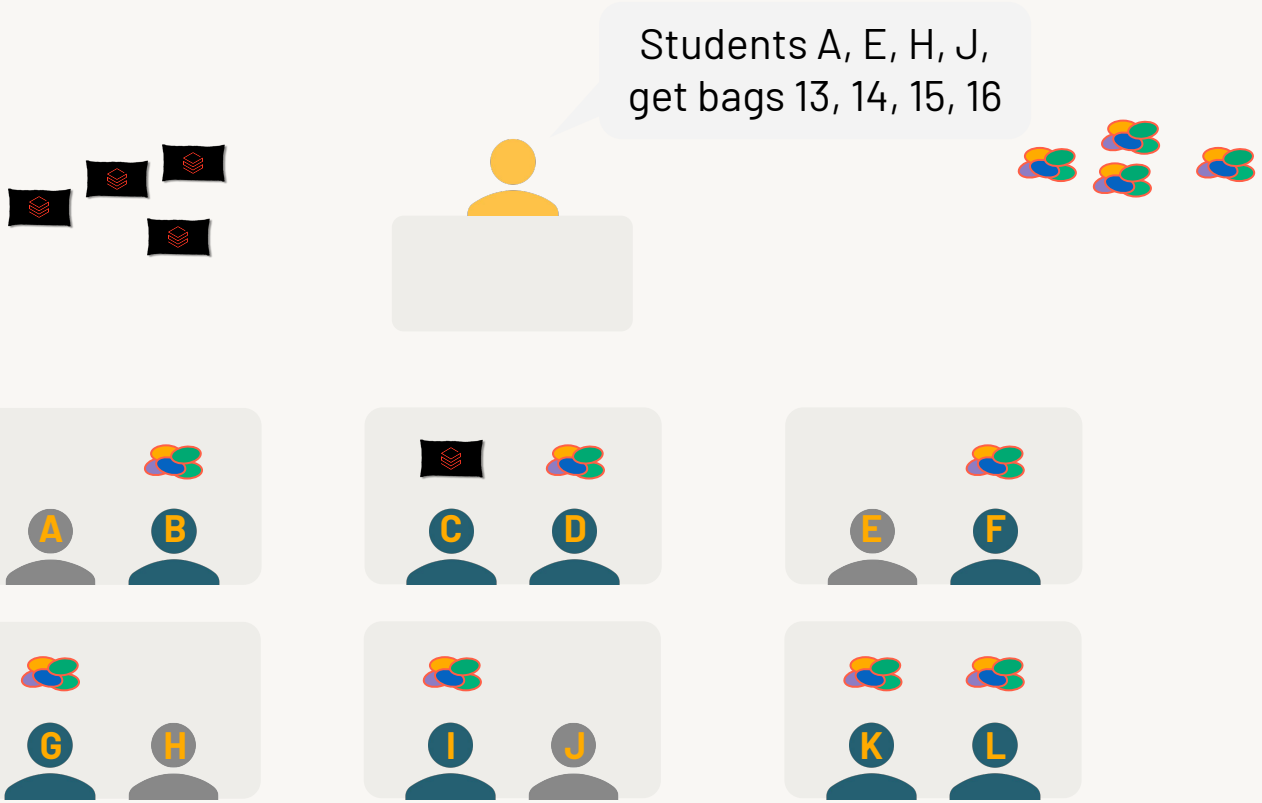


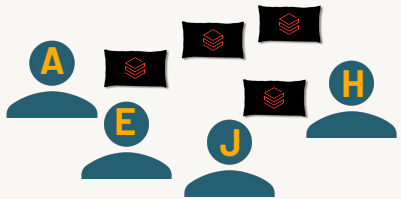


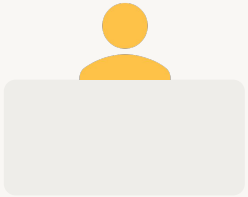


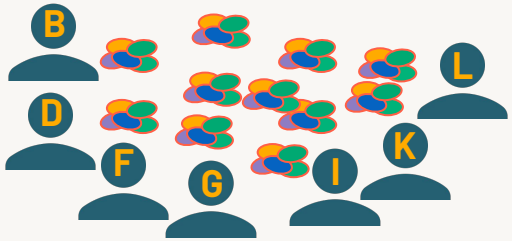


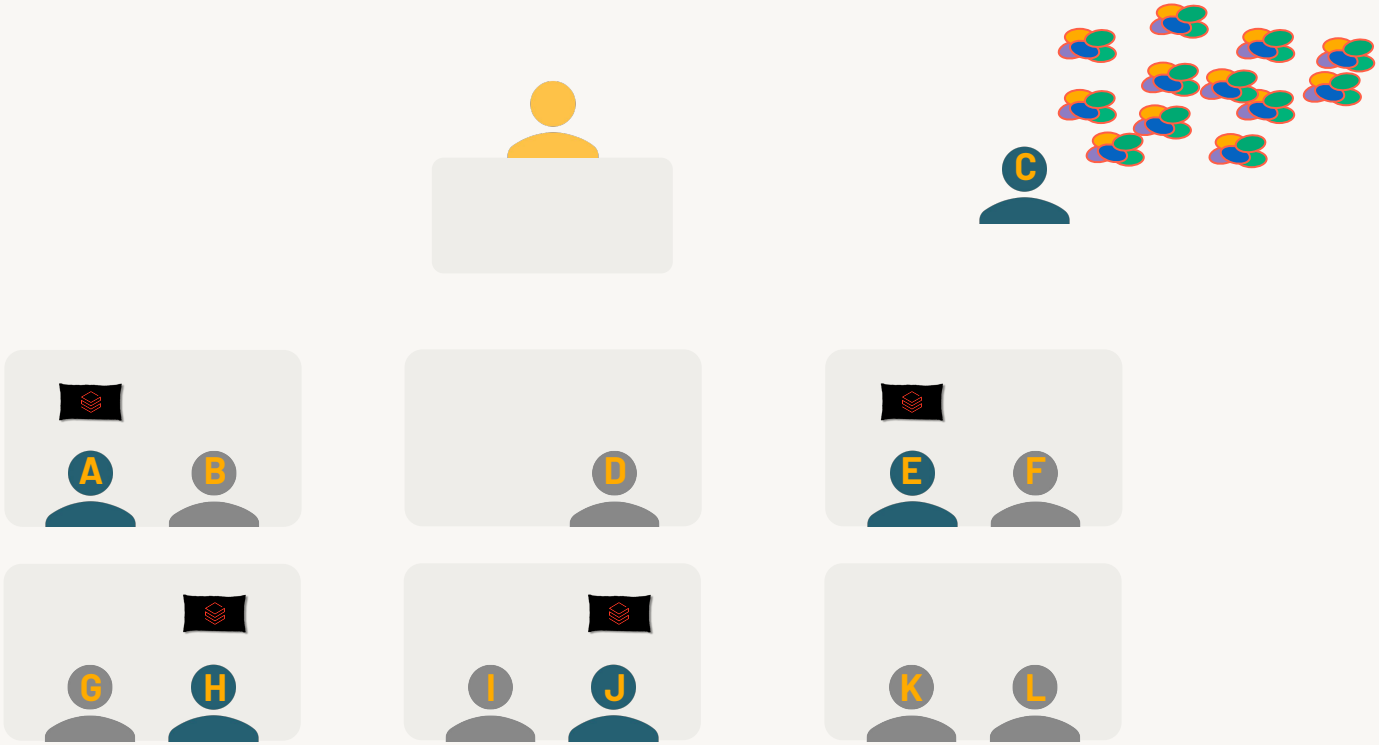


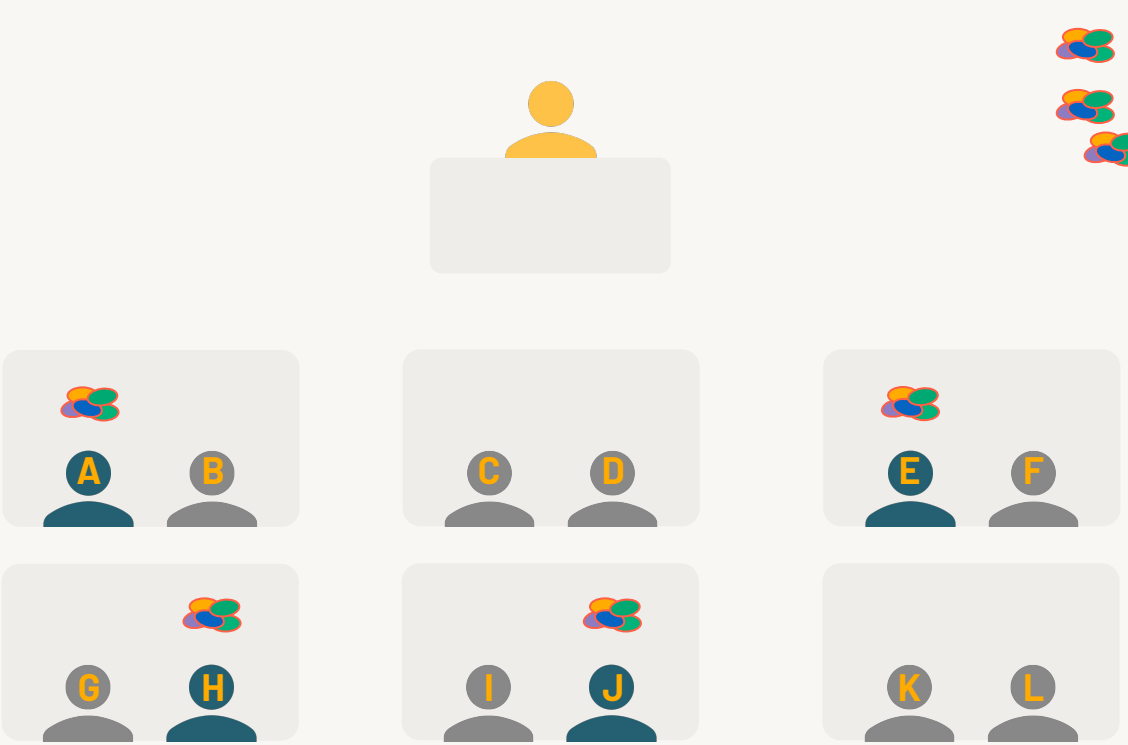


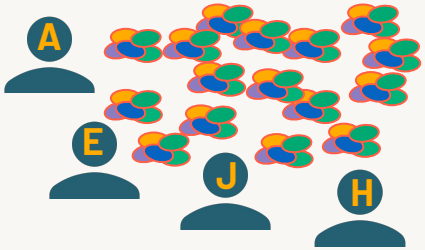


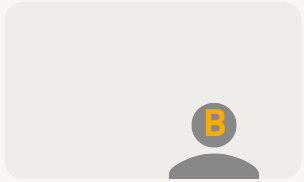


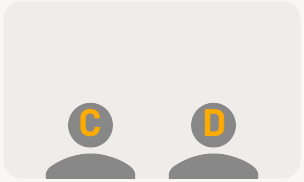


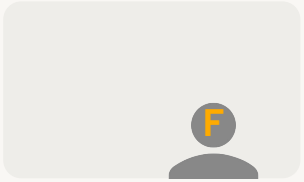


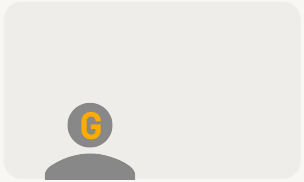


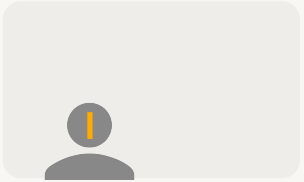


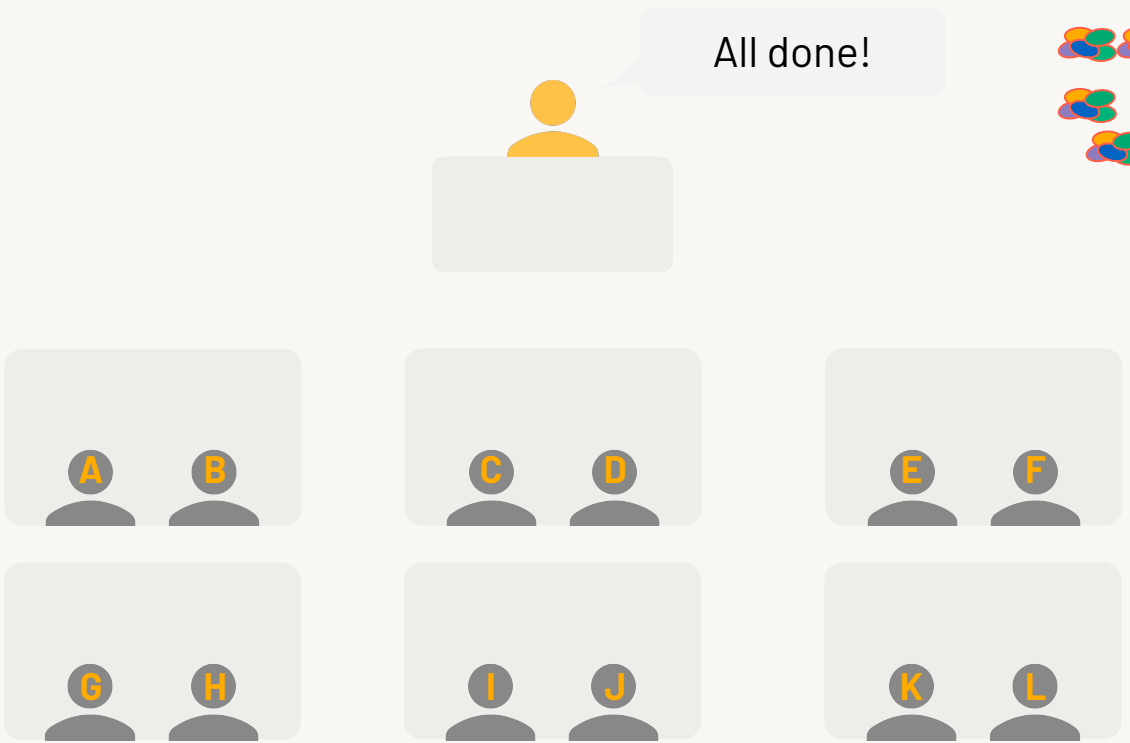


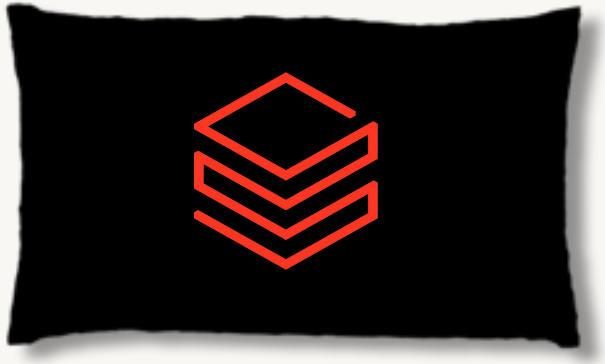








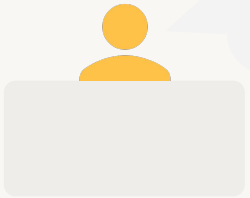




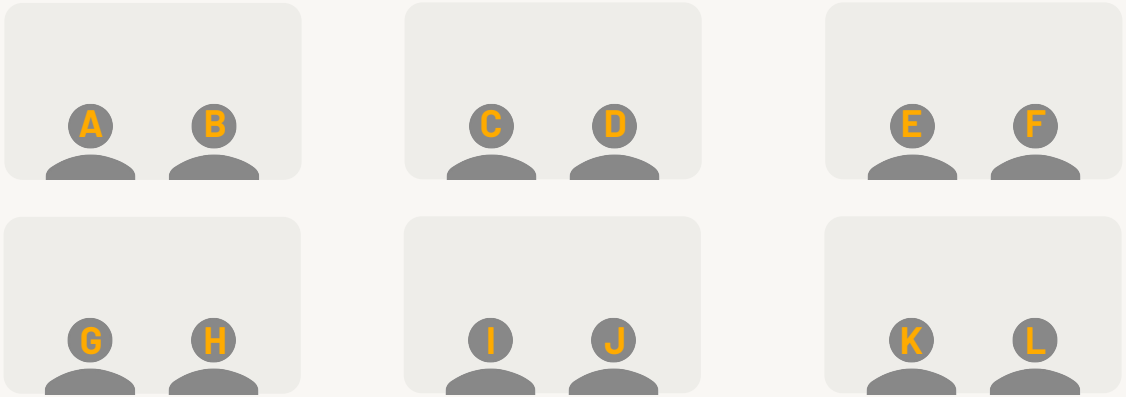
Scenario 2: Count total pieces in candy bags

Stage 1: Local Count

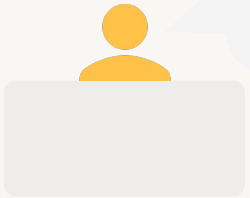
Stage 1: Local Count



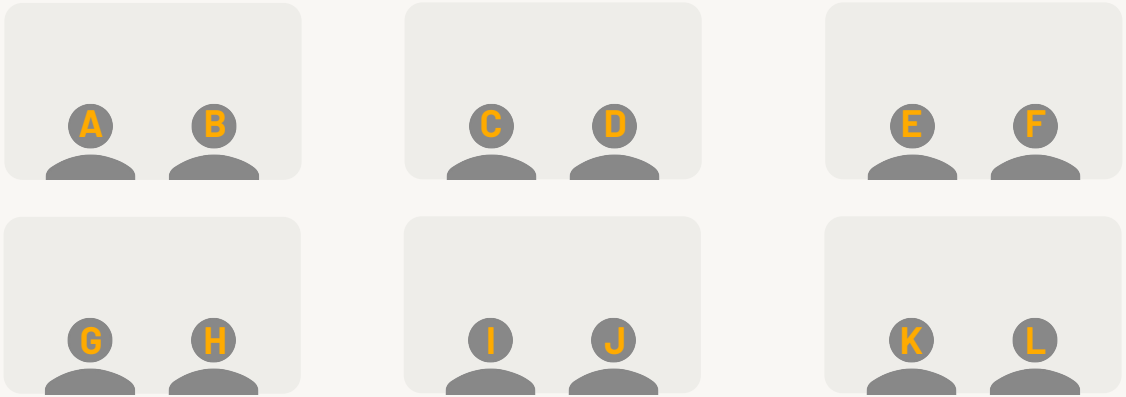
We need to count the total pieces in these candy bags



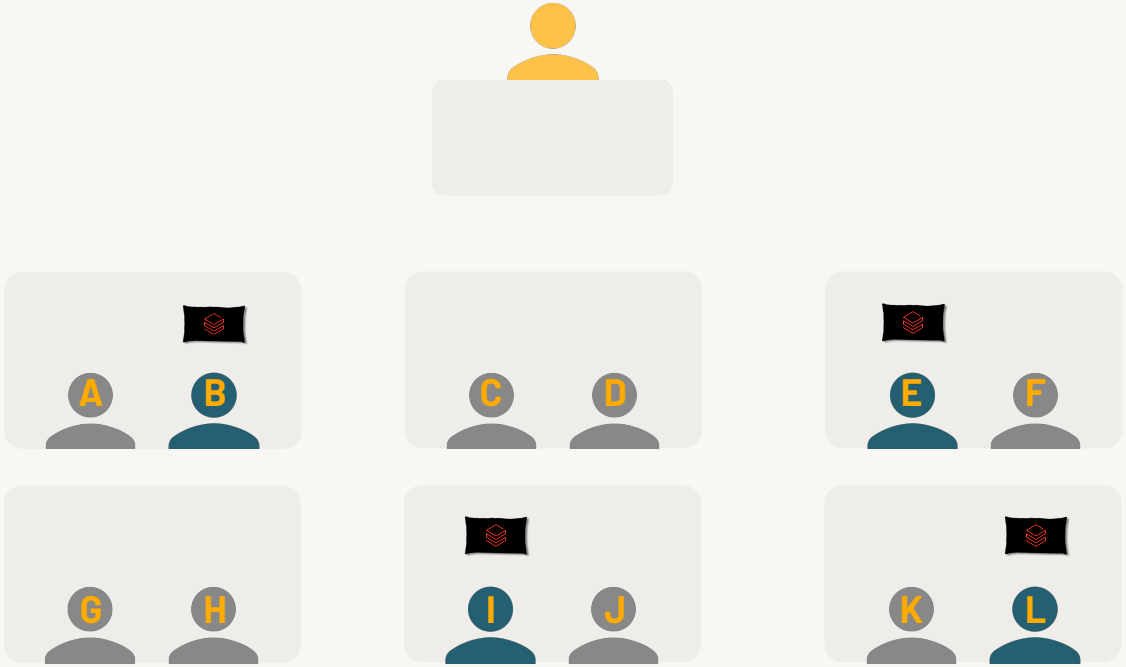
Stage 1: Local Count



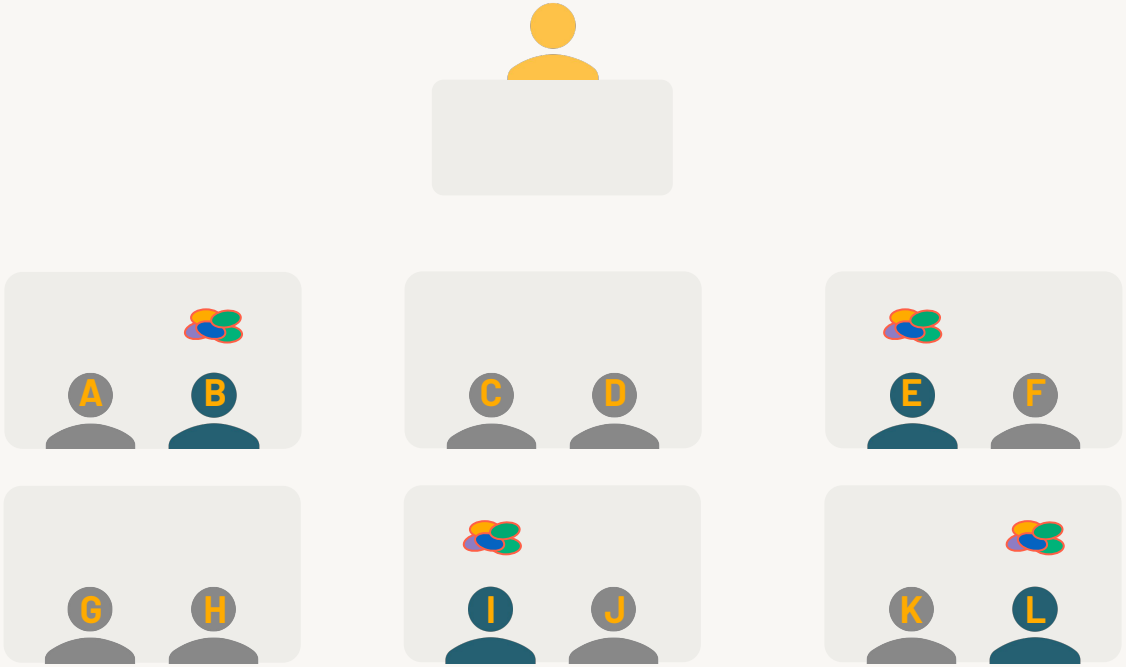
Students B, E, I, L,
get these four bags



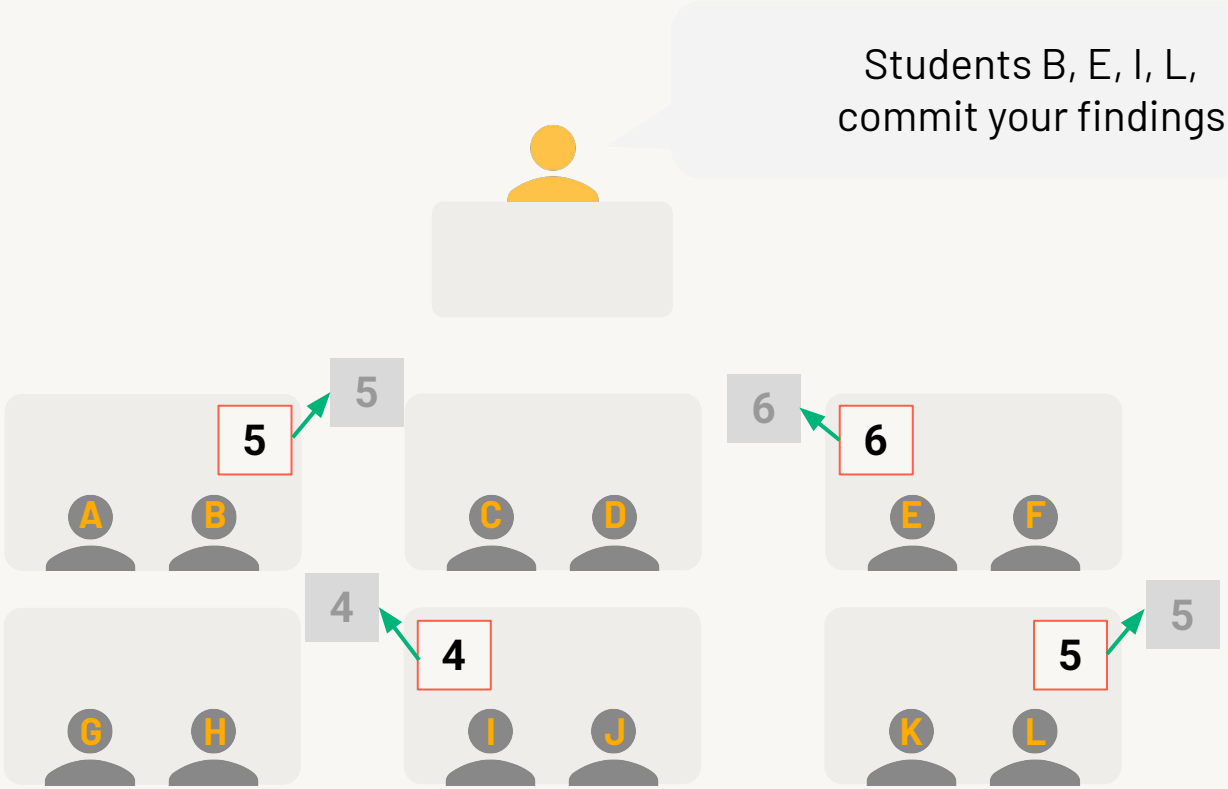
Stage 1: Local Count



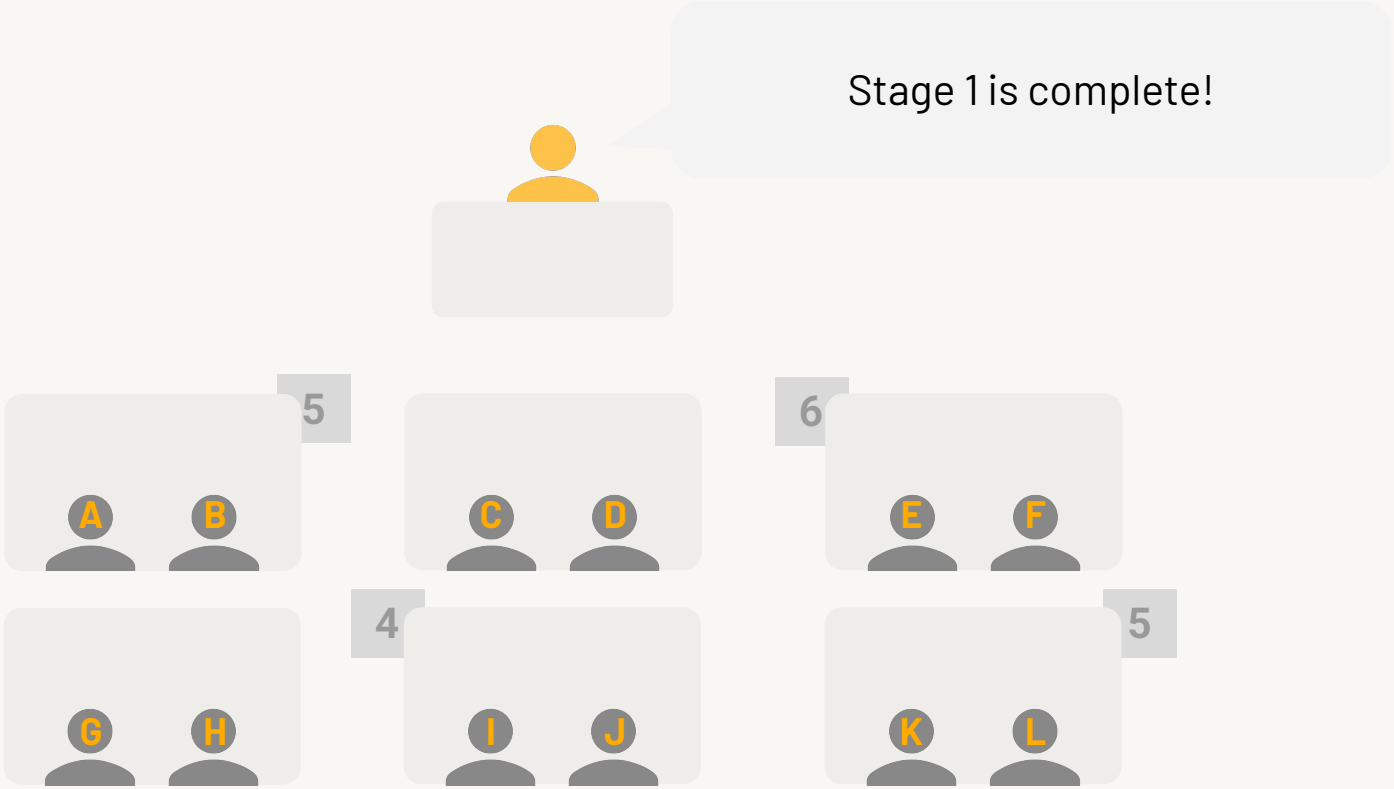
Stage 1: Local Count



Stage 1: Local Count

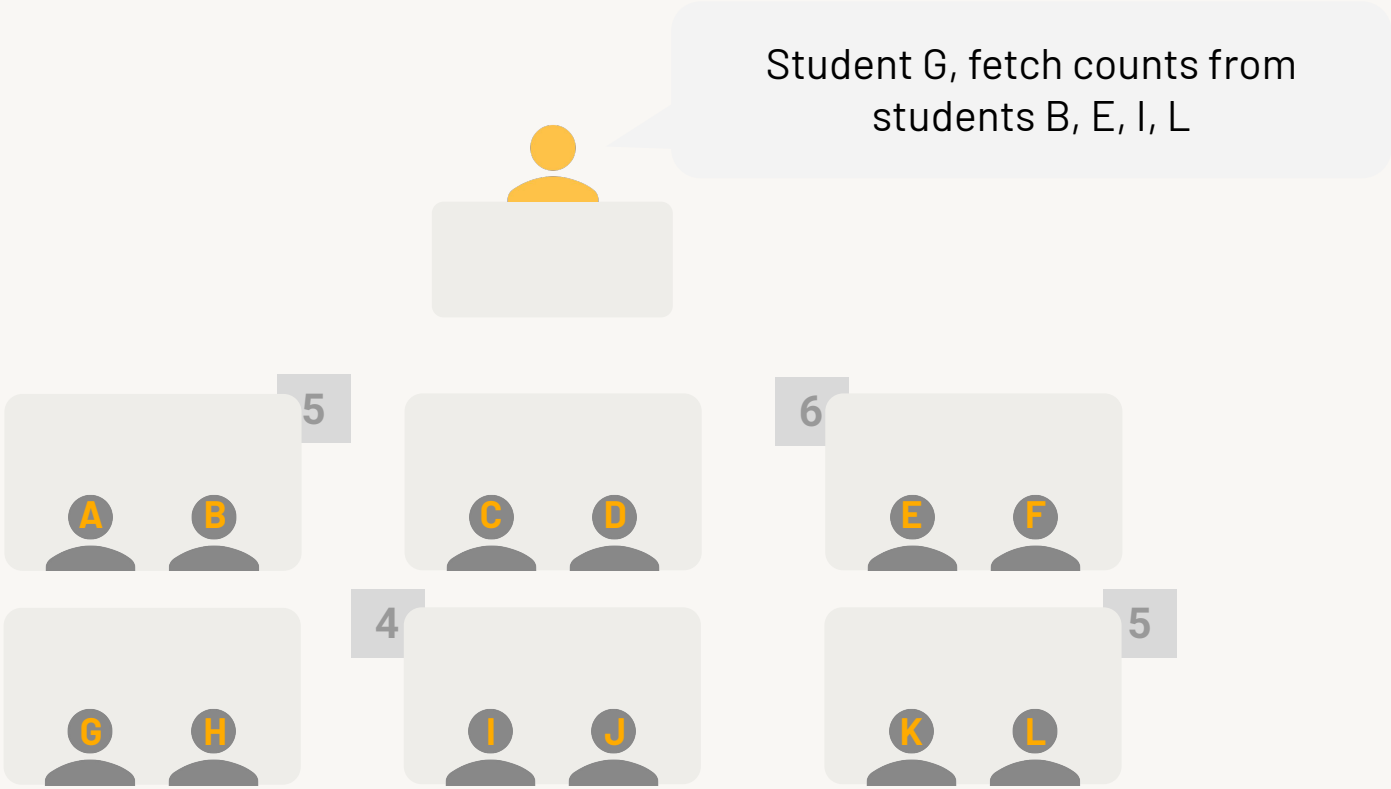


Stage 1: Local Count

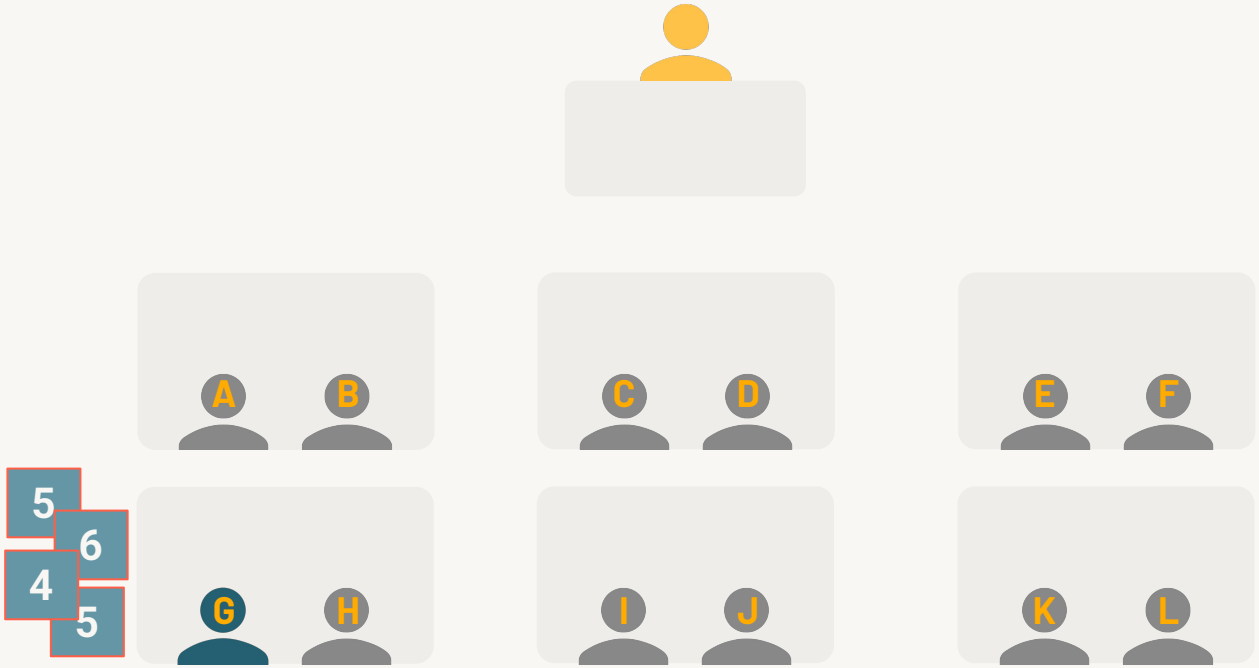


Stage 2: Global Count

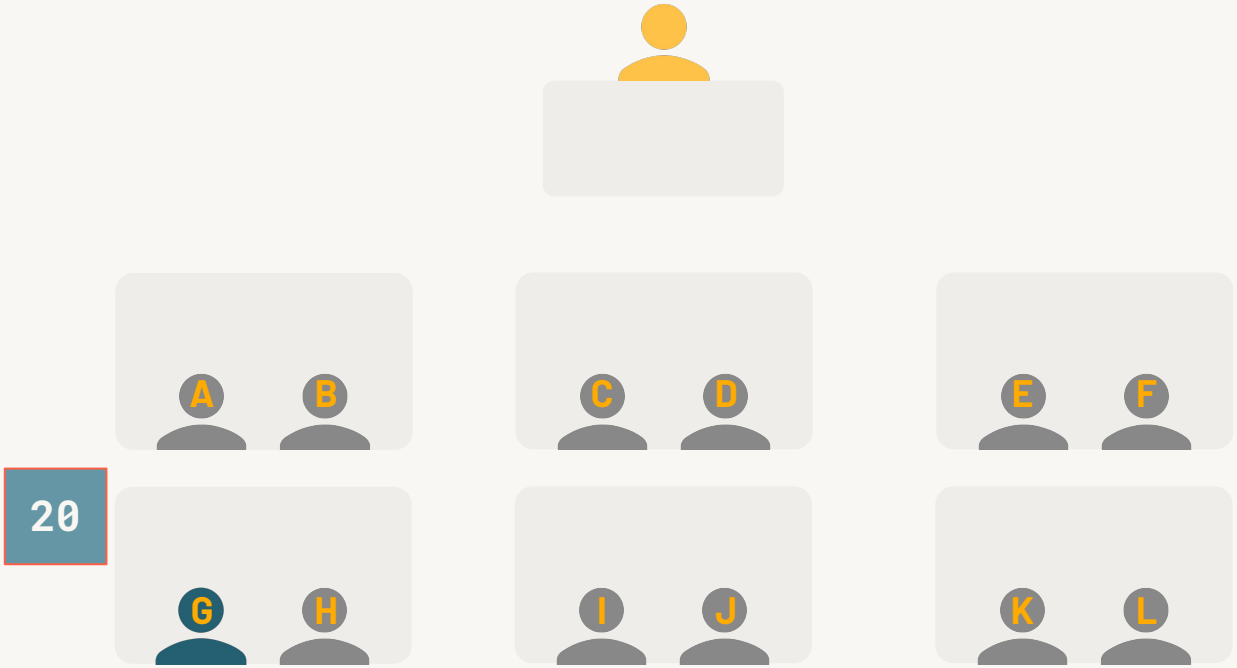
Stage 2: Global Count



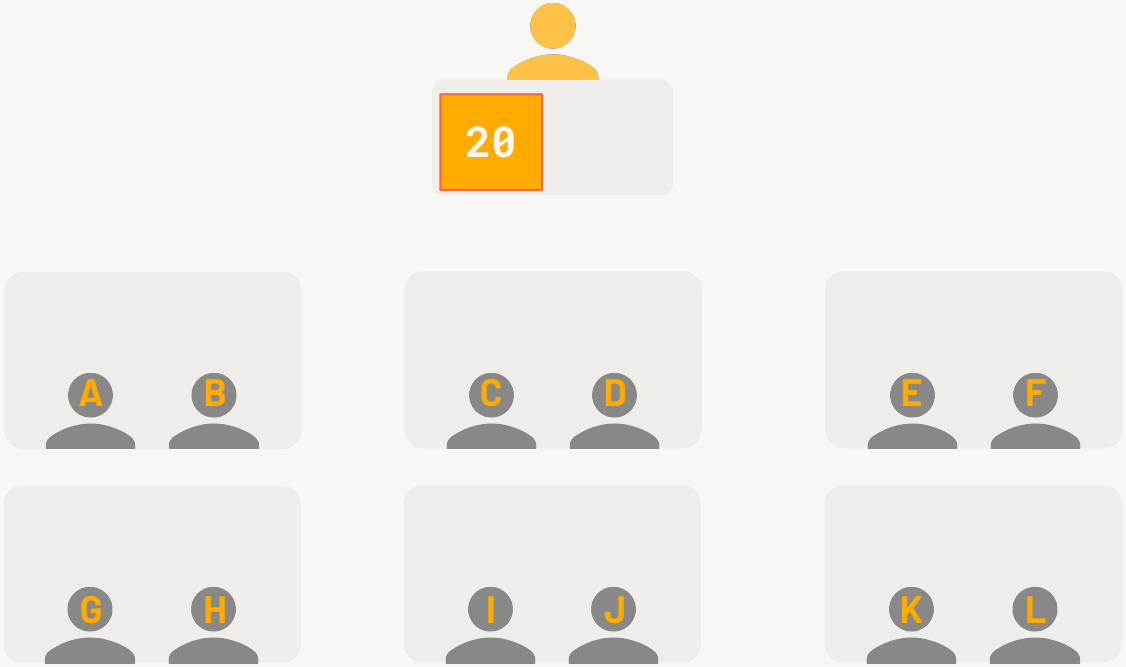
Stage 2: Global Count



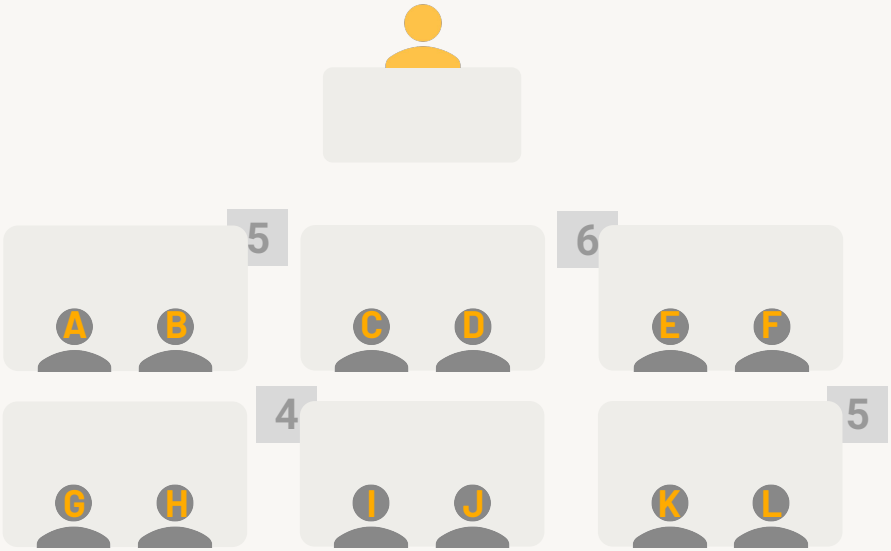
Stage 2: Global Count



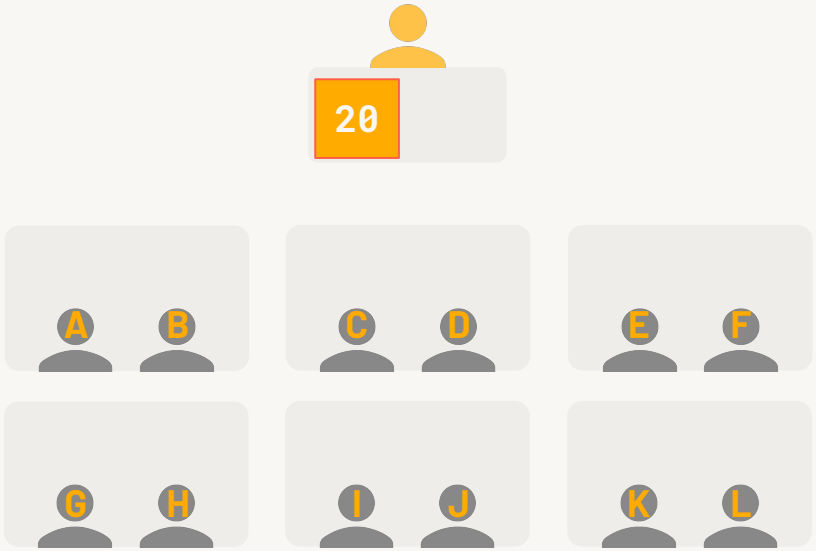
Stage 2: Global Count



Stage 1: Local Count



Stage 2: Global Count

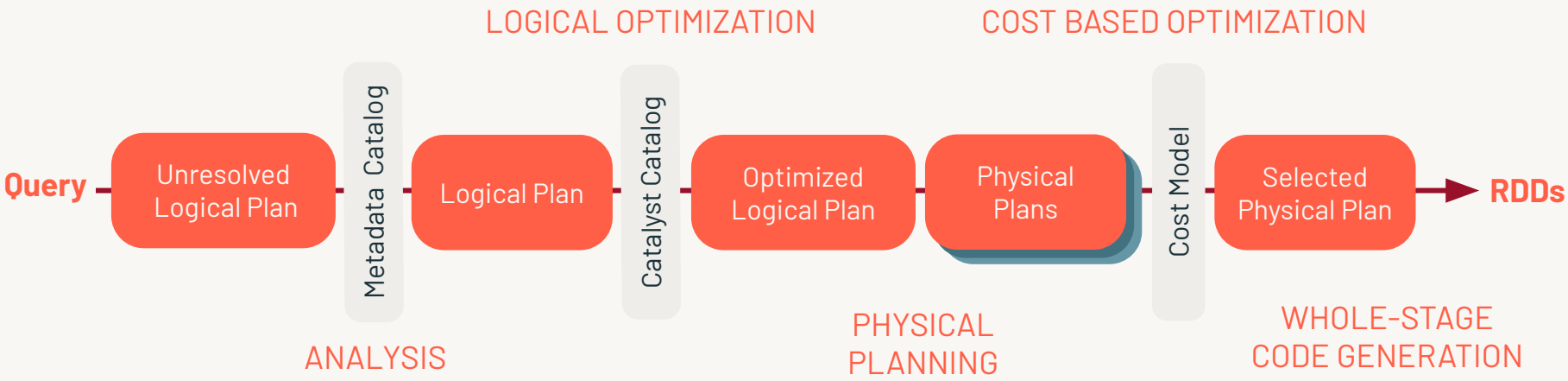


Query Optimization

Catalyst Optimizer

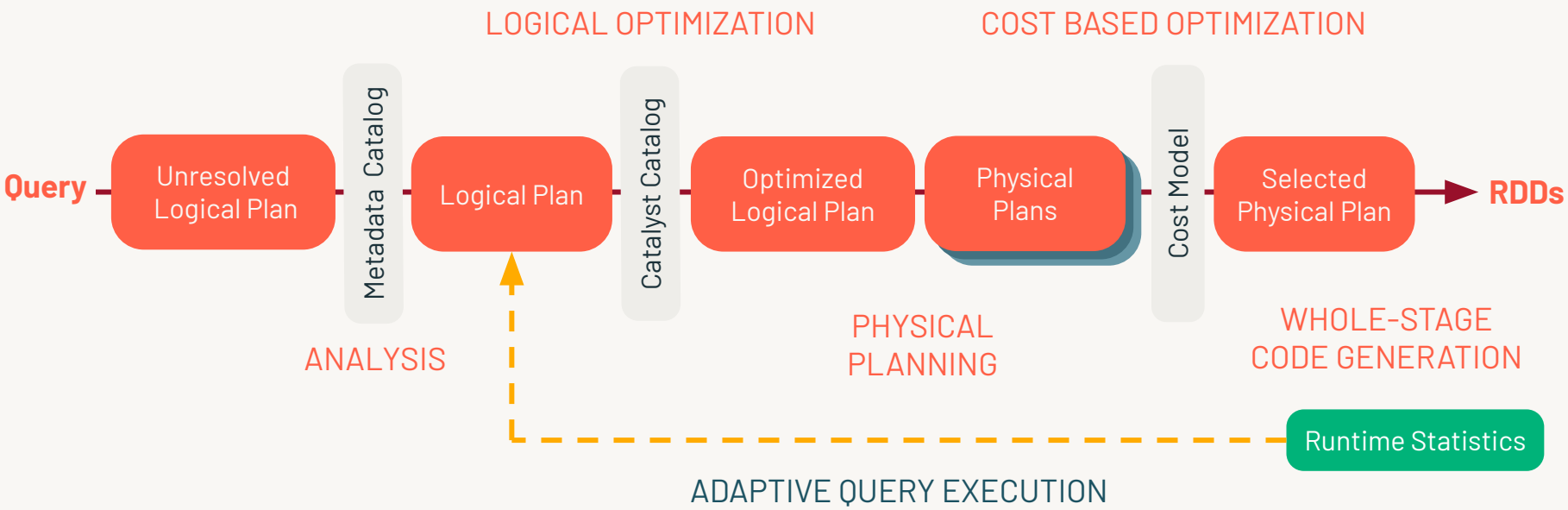
Adaptive Query Execution

Query Optimization



Query Optimization with AQE

New in Spark 3.0, **enabled** by default as of Spark 3.2



Module 5

Structured Streaming

databricks

Structured Streaming

Streaming Query

Stream Aggregations

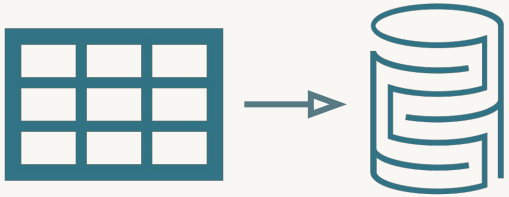
Streaming Query

Advantages

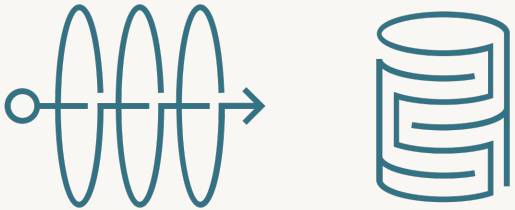
Use Cases

Sources

Batch Processing

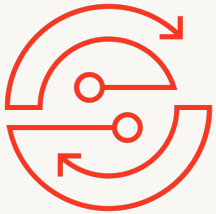


Stream Processing



Advantages of Stream Processing

Lower latency



Efficient Updates

Automatic bookkeeping on new data

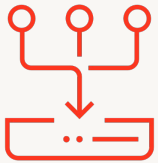
Stream Processing Use Cases

Notifications

Real-time reporting

Incremental ETL

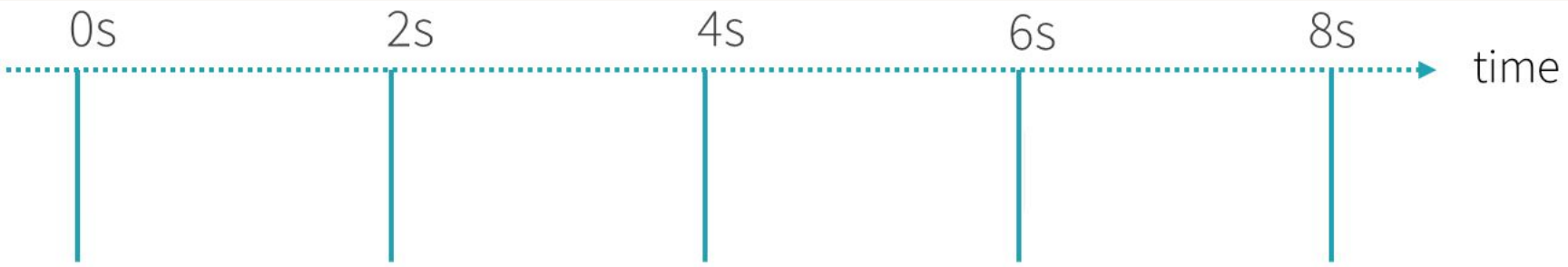
Update data to serve in real-time



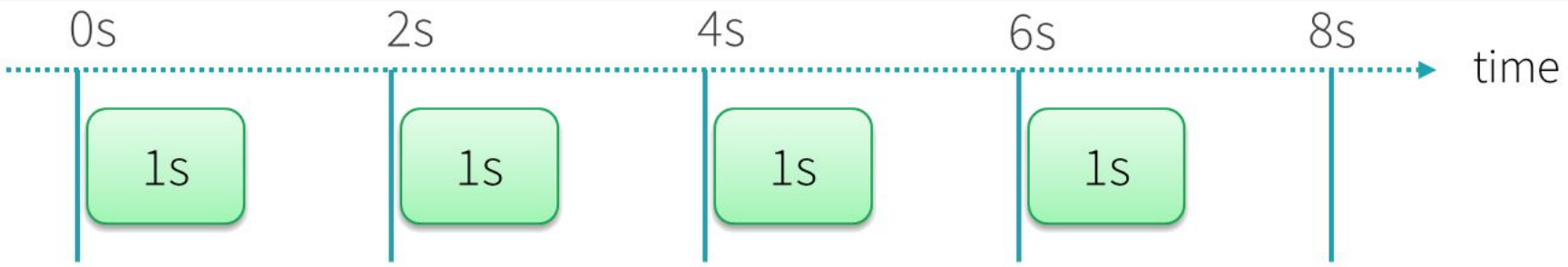
Real-time decision making

Online ML

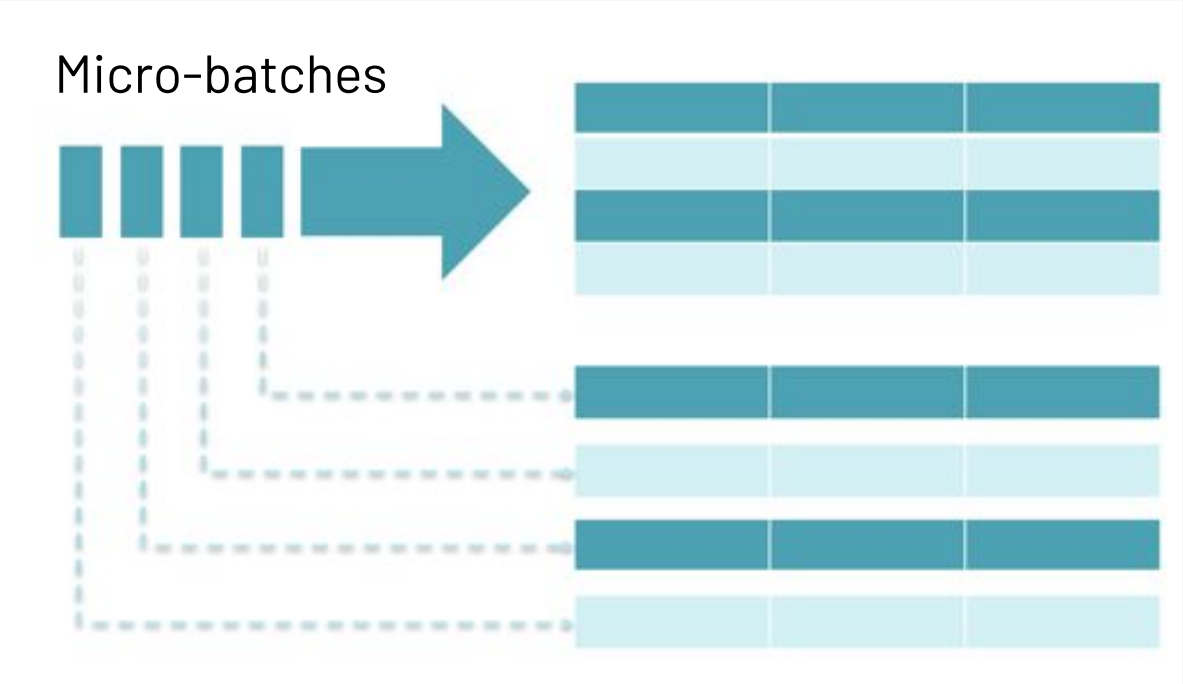
Micro-Batch Processing



Micro-Batch Processing



Structured Streaming



= new rows appended to unbounded table

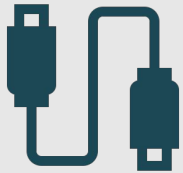
Input Sources

Kafka

Event Hubs

Files

Sockets



Generator

FOR TESTING

Sinks

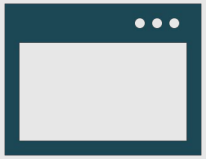
Kafka

Event Hubs

Files

Foreach

Console



Memory

FOR DEBUGGING

Output Modes

APPEND

Add new records only

UPDATE

Update changed records in place

COMPLETE

Rewrite full output

Trigger Types

Default	Process each micro-batch as soon as the previous one has been processed
Fixed interval	Micro-batch processing kicked off at the user-specified interval
One-time	Process all of the available data as a single micro-batch and then automatically stop the query
Continuous Processing	Long-running tasks that continuously read, process, and write data as soon events are available **Experimental*

See Structured Streaming Programming Guide

End-to-end fault tolerance

Guaranteed in Structured Streaming by

Checkpointing and write-ahead logs

Idempotent sinks

Replayable data sources

Stream Aggregations

Aggregations
Windows
Watermarking

Real-time Aggregations

Errors in IoT data by device type

Anomalous behavior in server log files by country

Behavior analysis on messages by hashtags

Time-Based Windows

Tumbling Windows

No window overlap

Any given event gets aggregated into **only one** window group

e.g. 1:00–2:00 am, 2:00–3:00 am, 3:00–4:00 am, ...

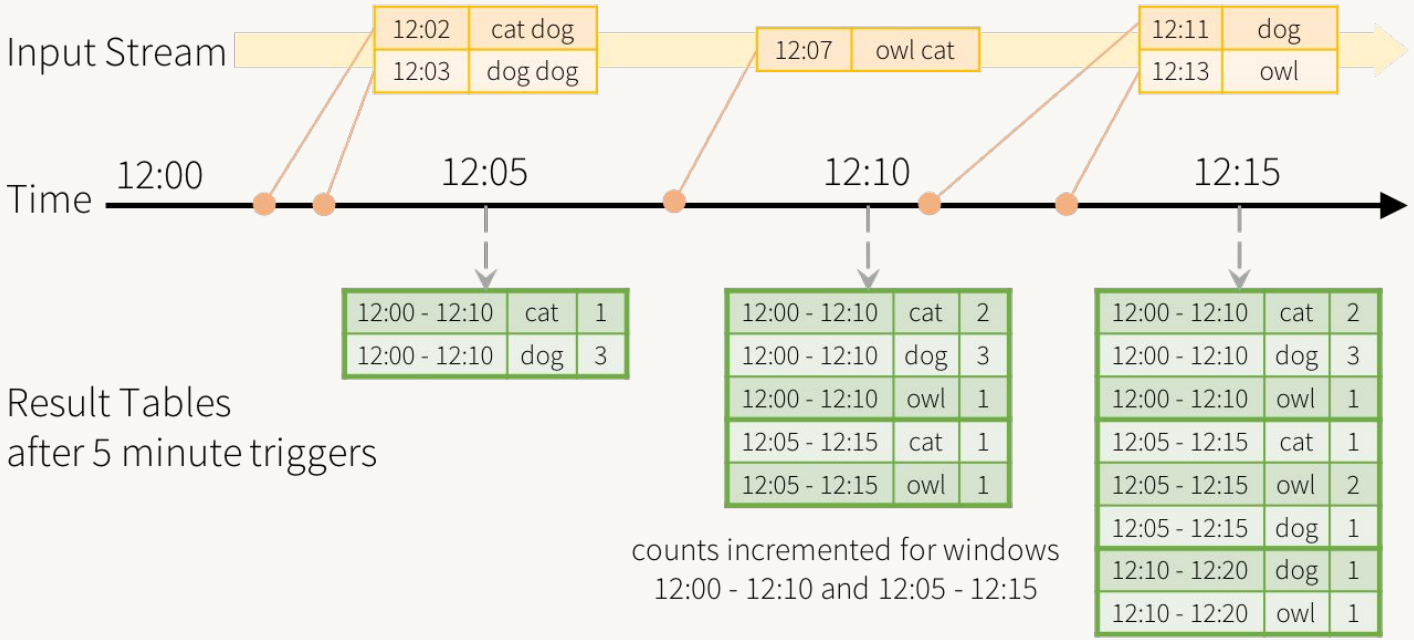
Sliding Windows

Windows overlap

Any given event gets aggregated into **multiple** window groups

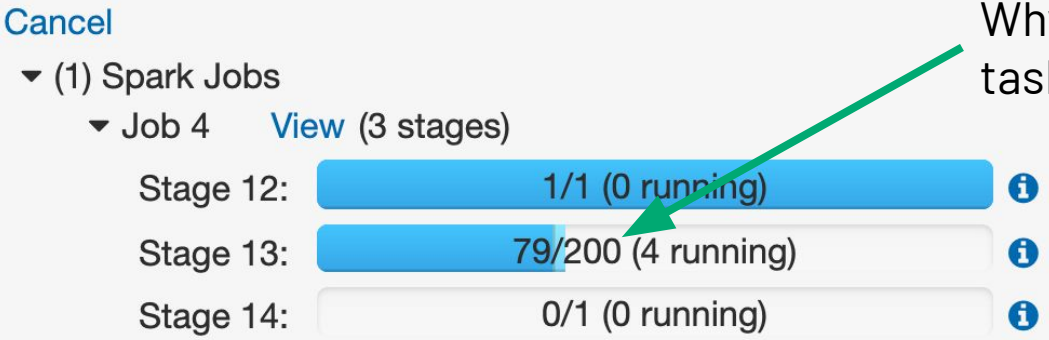
e.g. 1:00–2:00 am, 1:30–2:30 am, 2:00–3:00 am, ...

Sliding Windows Example



Windowing

```
(streamingDF
  .groupBy(col("device"),
           window(col("time"), "1 hour"))
  .count())
```



Why are we seeing 200 tasks for this stage?

Control the Shuffle Repartitioning

```
spark.conf.set("spark.sql.shuffle.partitions",spark.sparkContext.defaultParallelism)
```

Event-Time Processing

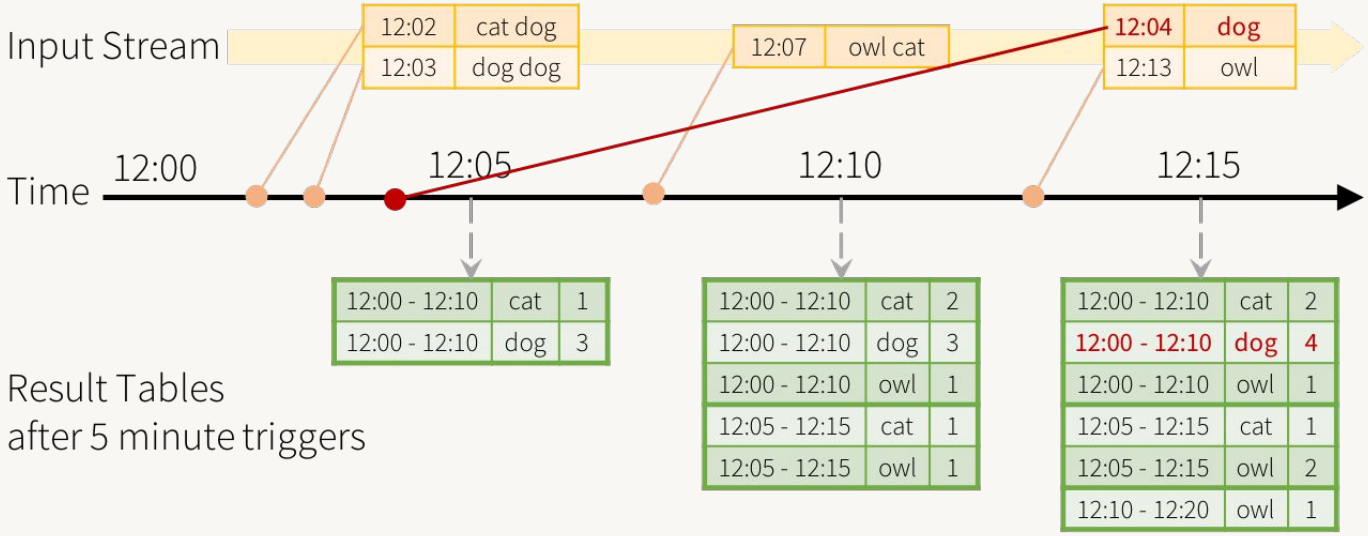
EVENT-TIME DATA

Process based on event-time (time fields embedded in data) rather than receipt time

WATERMARKS

Handle late data and limit how long to remember old data

Handling Late Data and Watermarking



Watermarking

```
(streamingDF
  .withWatermark("time", "2 hours")
  .groupBy(col("device"),
          window(col("time"), "1 hour"))
  .count()
)
```

Module 6

Delta Lake

databricks

Delta Lake

Using Spark with Delta Lake

Delta Lake

Delta Lake Concepts

What is Delta Lake?

- Technology designed to be used with Apache Spark to build robust data lakes
- Open source project at delta.io
- Databricks Delta Lake documentation

Delta Lake features

- ACID transactions on Spark
- Scalable metadata handling
- Streaming and batch unification
- Schema enforcement
- Time travel
- Upserts and deletes
- Fully configurable/optimizable
- Structured streaming support

Delta Lake components

Delta Lake storage layer

Delta tables

Delta Engine

Delta Lake Storage Layer

- Highly performant and persistent
- Low-cost, easily scalable object storage
- Ensures consistency
- Allows for flexibility

Delta tables

- Contain data in Parquet files that are kept in object storage
- Keep transaction logs in object storage
- Can be registered in a metastore (optional)

Delta Engine

Databricks edge feature; not available in OS Apache Spark

- File management optimizations
- Auto-optimized writes
- Performance optimization via Delta caching

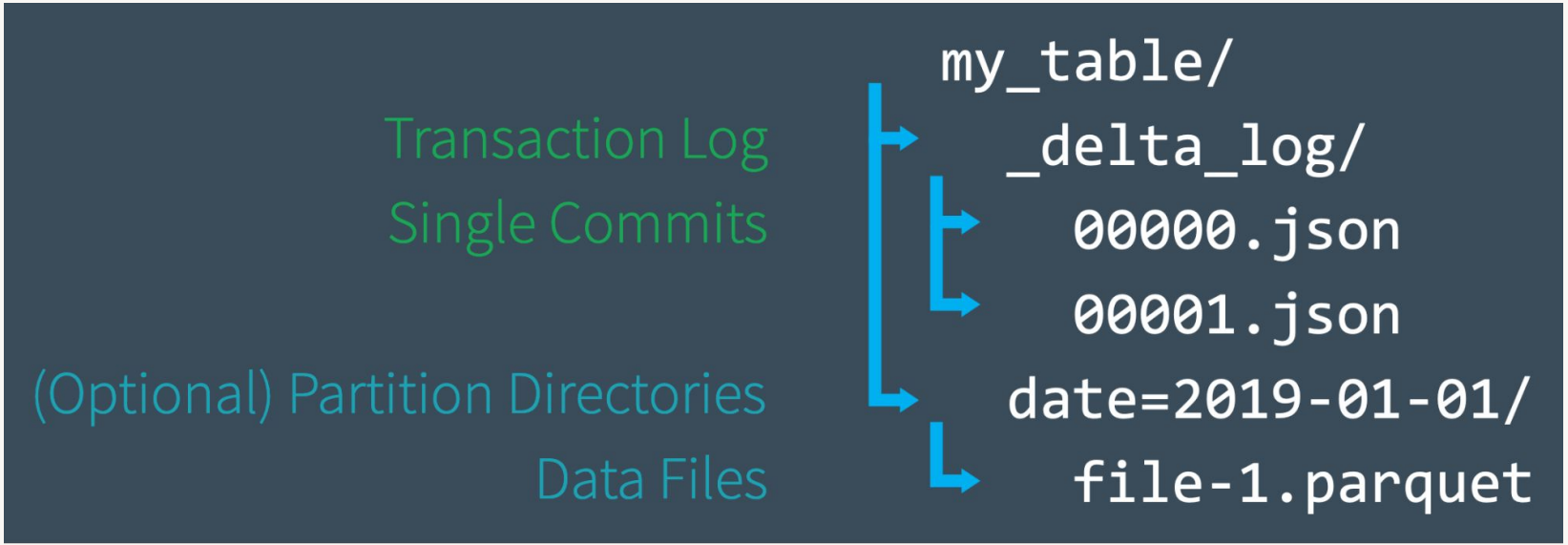
What is the Delta transaction log?

- Ordered record of the transactions performed on a Delta table
- Single source of truth for that table
- Mechanism that the Delta Engine uses to guarantee atomicity

How does the transaction log work?

- Delta Lake breaks operations down into one or more of these steps:
  - Add file
  - Remove file
  - Update metadata
  - Set transaction
  - Change protocol
  - Commit info

Delta transaction log at the file level



Adding commits to the transaction log

`DESCRIBE` Command

- Returns the metadata of an existing table
  - Ex. Column names, data types, comments

`DESCRIBE HISTORY` Command

- Returns a more complete set of metadata for a Delta table
  - Operation, user, operation metrics

Delta Lake Time Travel

- Query an older snapshot of a Delta table
  - Re-creating analysis, reports or outputs
  - Writing complex temporal queries
  - Fixing mistakes in data
  - Providing snapshot isolation

Time Travel SQL syntax

```
SELECT * FROM events TIMESTAMP AS OF timestamp_expression
SELECT * FROM events VERSION AS OF version
```


Time Travel SQL syntax

```
SELECT * FROM events TIMESTAMP AS OF timestamp_expression
```

- `timestamp_expression` can be:
  - String that can be cast to a timestamp
  - String that can be cast to a date
  - Explicit timestamp type (the result of casting)
  - Simple time or date expressions (see the Databricks documentation)

Time Travel SQL syntax

```
SELECT * FROM events VERSION AS OF version
```

- `Version` can be obtained from the output of `DESCRIBE HISTORY events`.

Thank you! Congrats!