



# Data Lineage with Apache Airflow using OpenLineage

Julien Le Dem and Willy Lulciuc, Datakin | July 2021

# Agenda

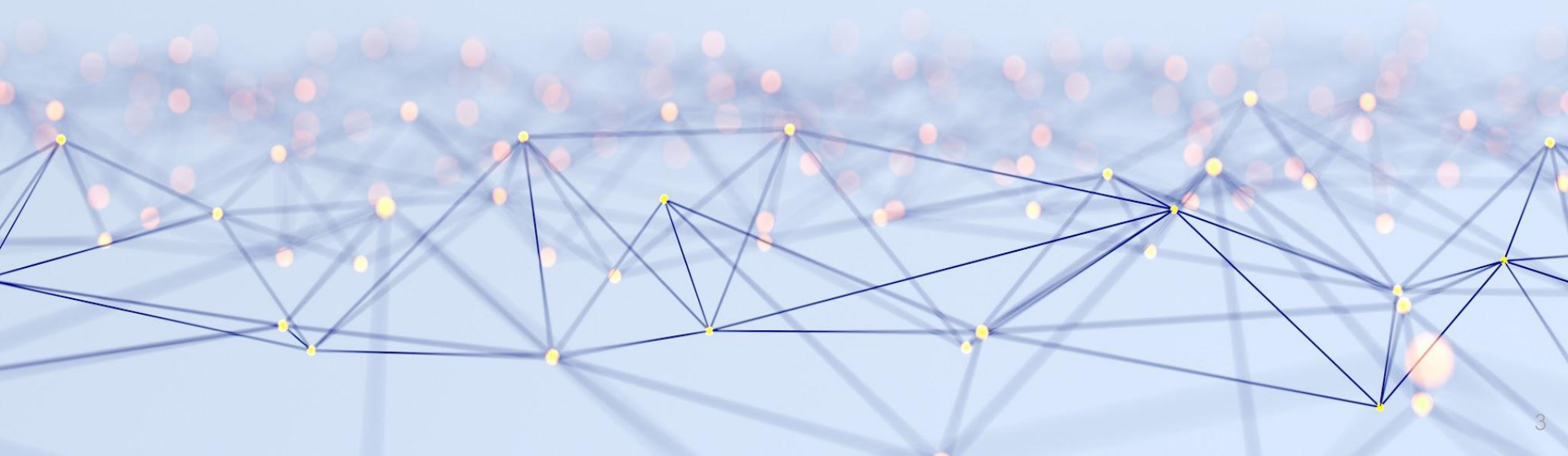
## The need for lineage metadata

## OpenLineage and Marquez

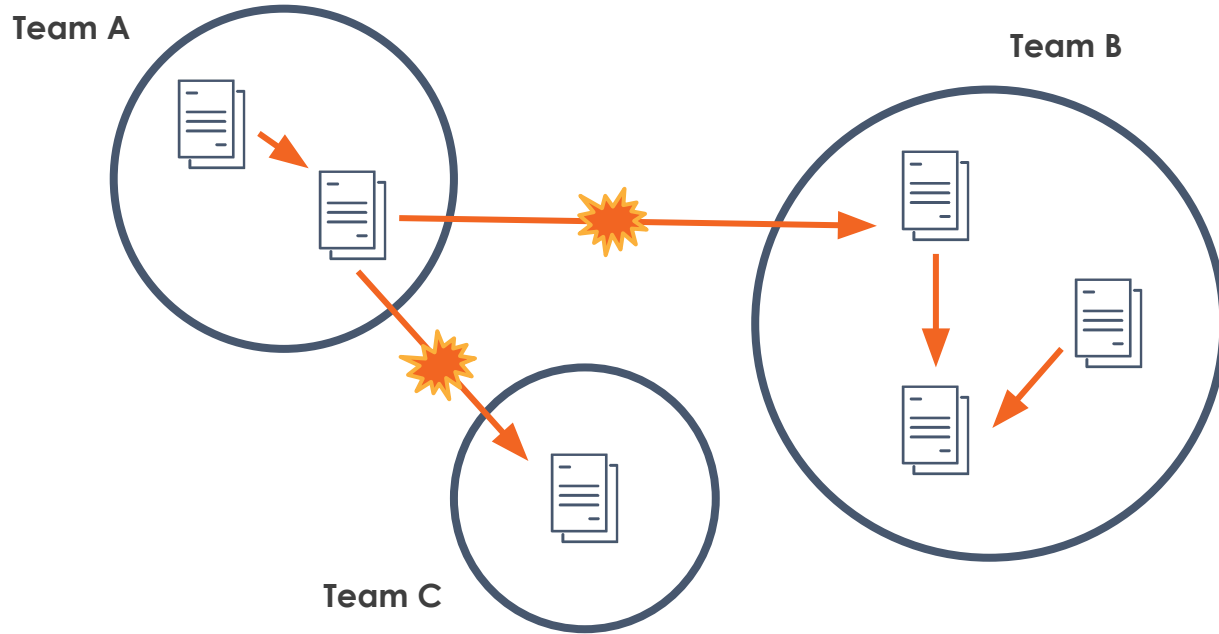
- OpenLineage, an open standard for lineage collection
- Marquez, its reference implementation

## Airflow observability with OpenLineage

# The need for lineage metadata



# Building a healthy data ecosystem



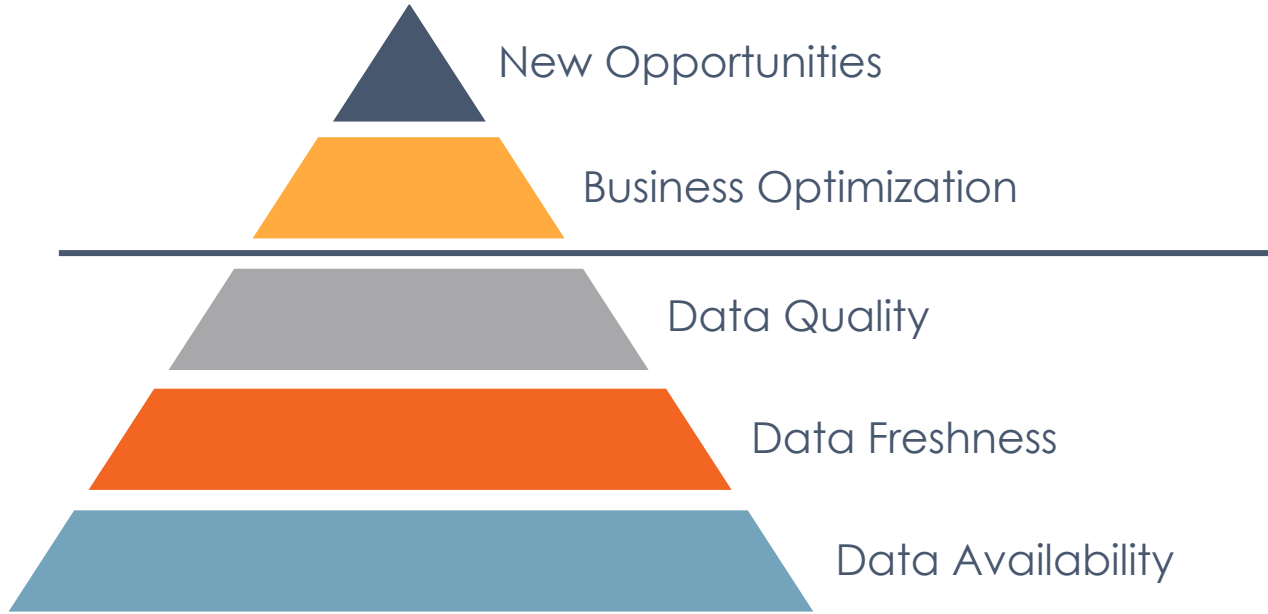
# Today: limited context



DATA

- What is the data source?
- What is the schema?
- Who is the owner?
- How often is it updated?
- Where does it come from?
- Who is using it?
- What has changed?

# ~~Maslow's~~ Data hierarchy of needs





# OpenLineage

# Contributors





# Purpose

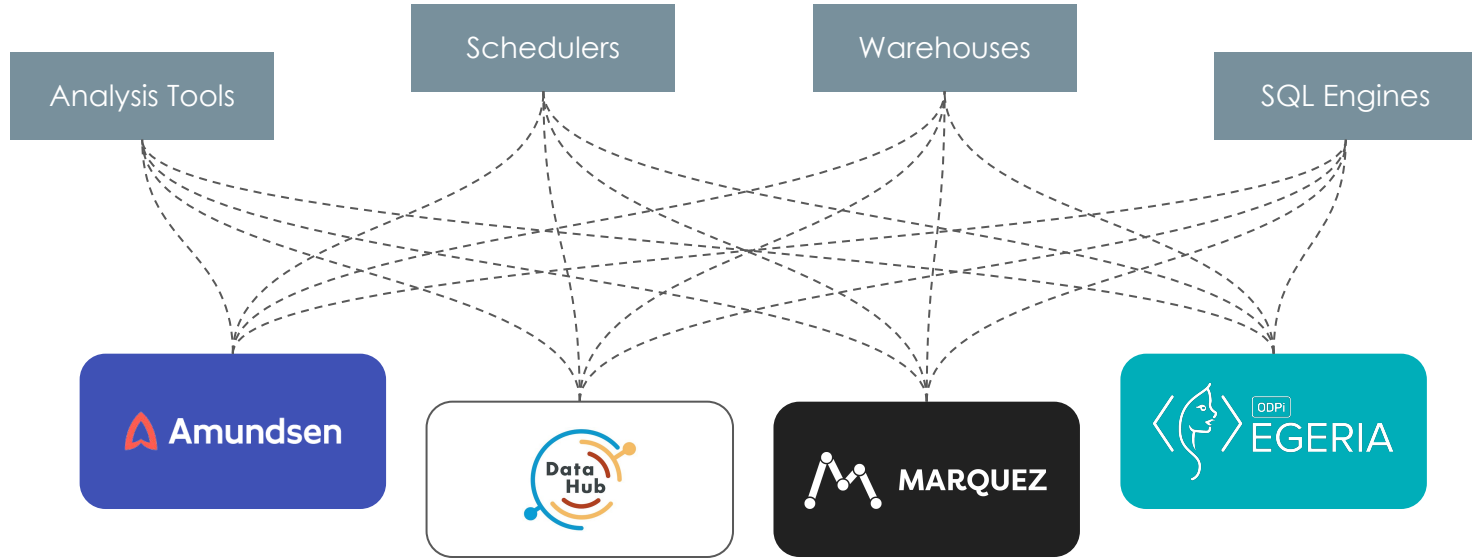
To define an **open standard** for collection of lineage metadata from pipelines **as they are running**.



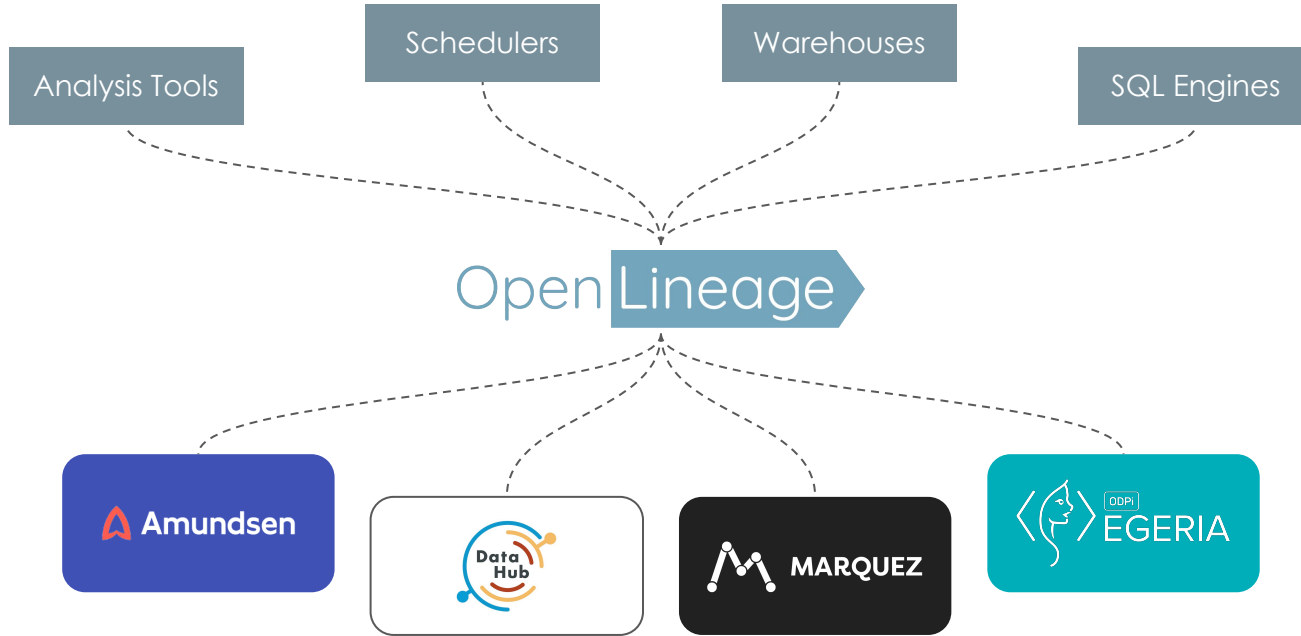
# EXIF for data pipelines



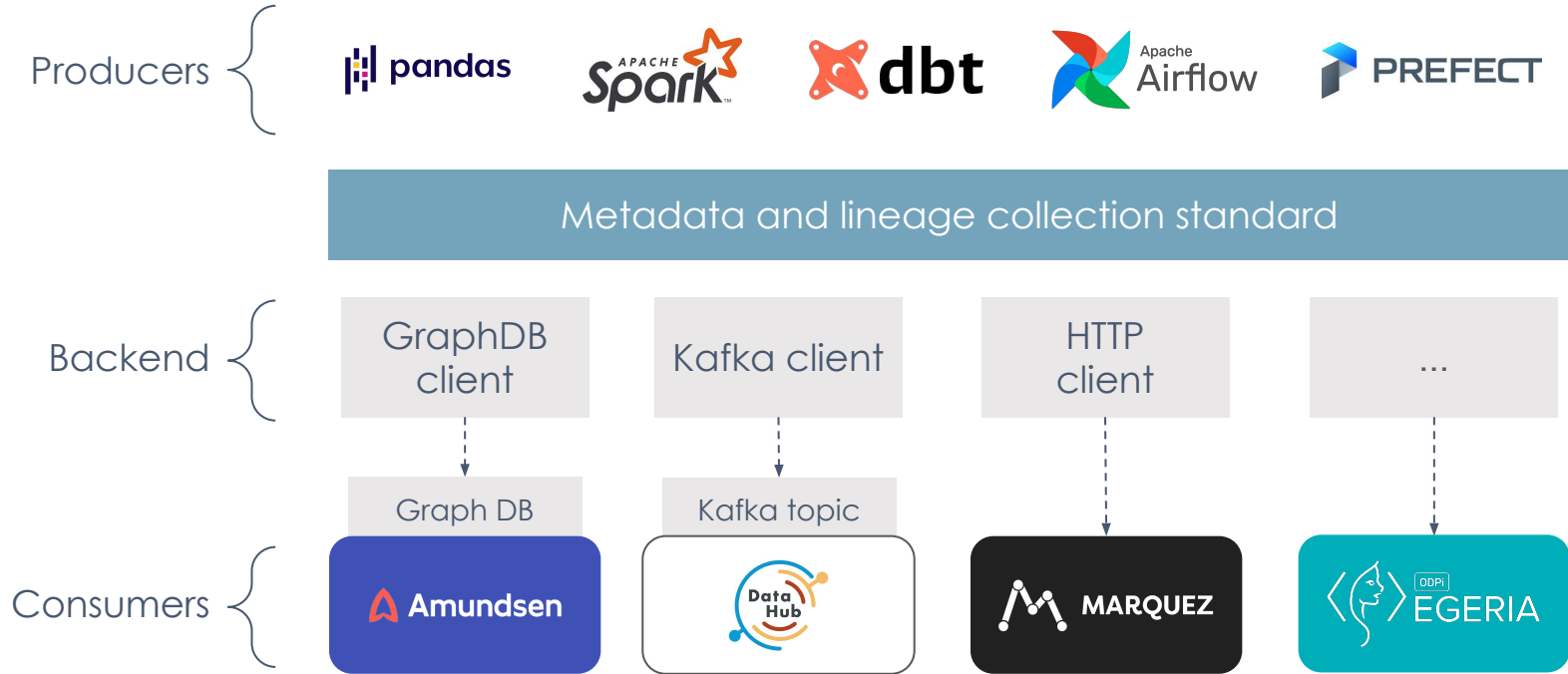
# Before OpenLineage



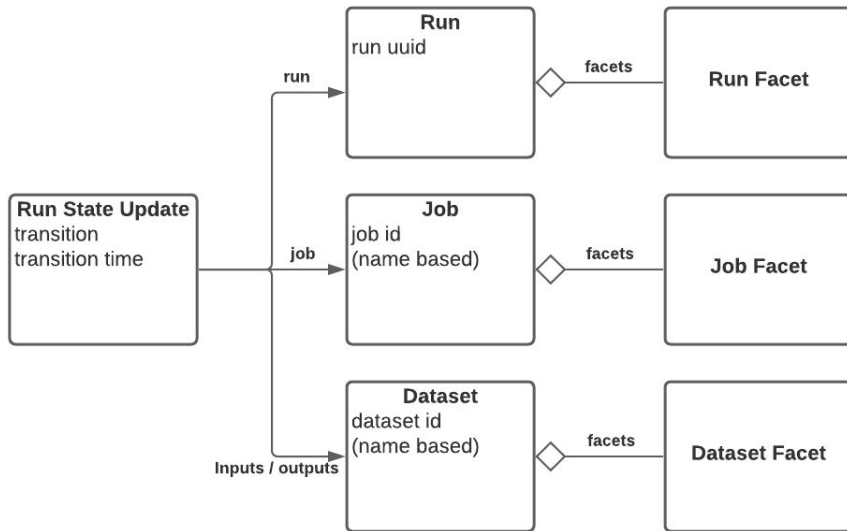
# With OpenLineage



# OpenLineage architecture



# Data model



Built around core entities:  
Datasets, Jobs, and Runs

Defined as a JSONSchema  
spec

Consistent naming for:  
Jobs (*`scheduler.job.task`*)  
Datasets (*`instance.schema.table`*)

# Protocol

Asynchronous events:

- Unique run ID for identifying a run and correlated events

Configurable backend:

- Kafka
- HTTP

## Examples

### **Run Start** event

- source code version
- run parameters

### **Run Complete** event

- input dataset
- output dataset version and schema

## **Extensible**

Facets are atomic pieces of metadata identified by a unique name that can be attached to core OpenLineage entities.

## **Decentralized**

Prefixes in facet names allow the definition of Custom Facets that can be promoted to the spec at a later point.

# OpenLineage Facets



# Facet examples

## Dataset:

- Stats
- Schema
- Version
- Column-level lineage

## Job:

- Source code
- Dependencies
- Source control
- Query plan

## Run:

- Scheduled time
- Batch ID
- Query profile
- Params

# OMG the possibilities are endless

Dependency tracing  
Root cause identification  
Issue prioritization  
Impact mapping  
Precision backfills  
Anomaly detection  
Change management  
Historical analysis  
Compliance

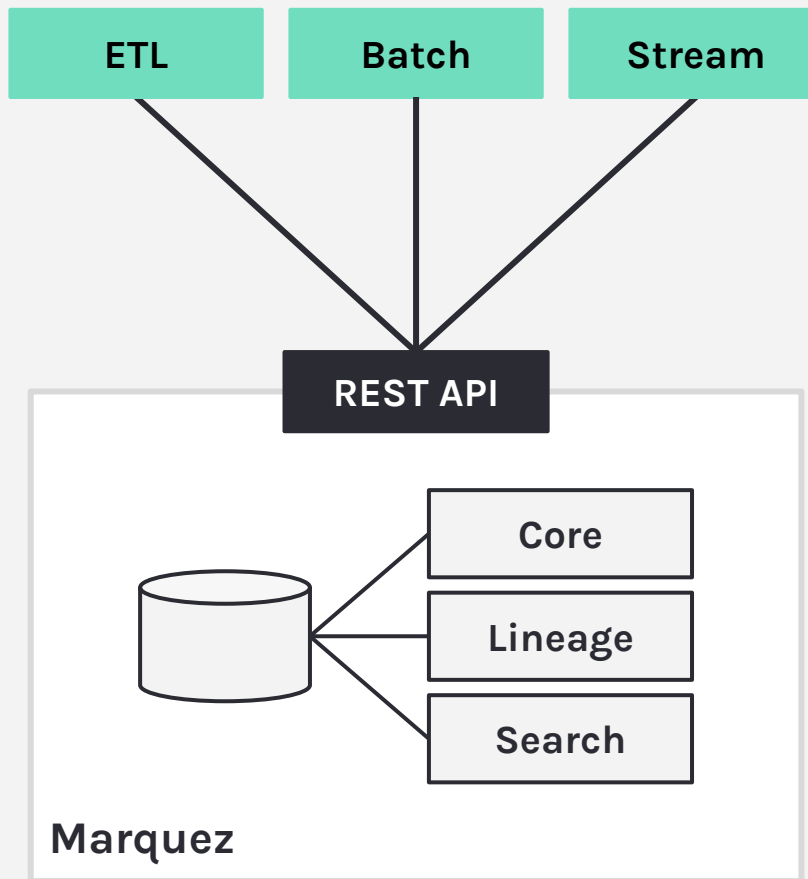


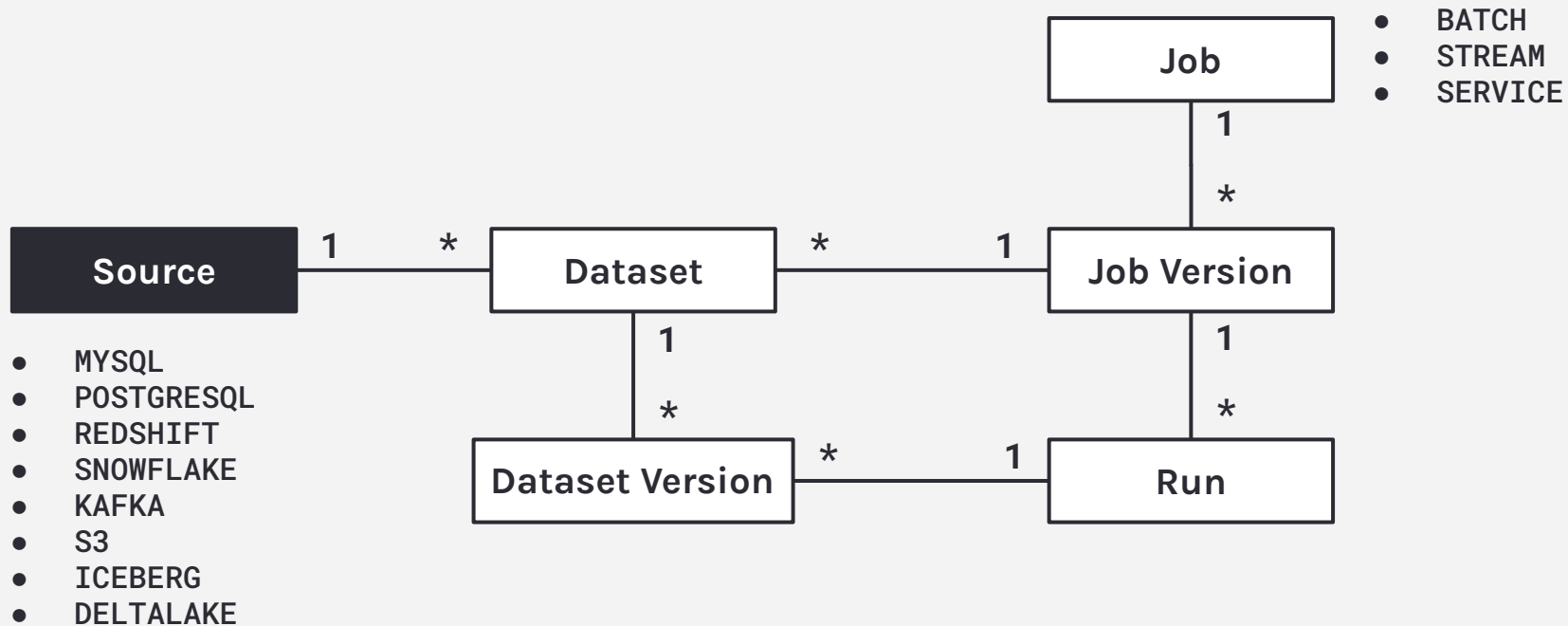


**MARQUEZ**

# Metadata Service

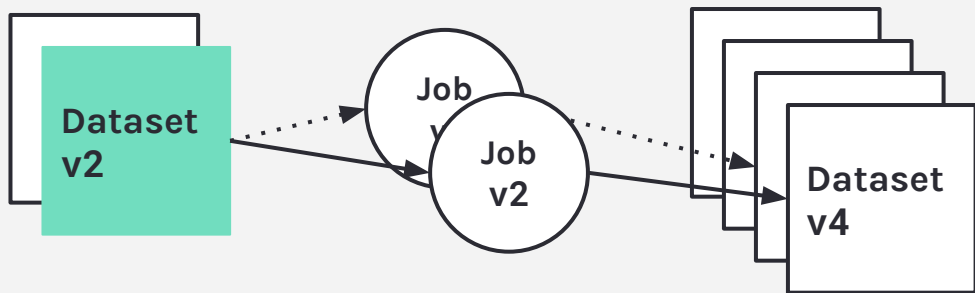
- **Centralized metadata management**
  - Sources
  - Datasets
  - Jobs
- **Features**
  - Data governance
  - Data lineage
  - Data discovery + exploration





# Design benefits

- Debugging
  - What **job version(s)** produced and consumed **dataset version X**?

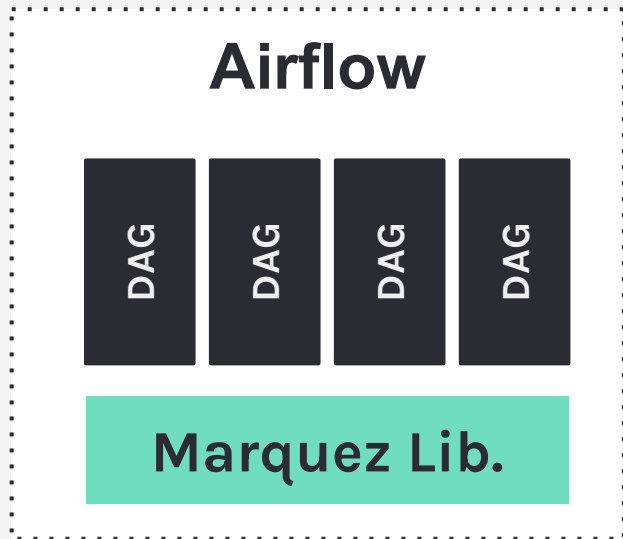


- Backfilling
  - Full / incremental processing

The background of the slide is white and decorated with numerous small, semi-transparent squares in red, green, and blue. Scattered throughout are several pinwheel logos, each composed of four overlapping triangles in red, green, blue, and cyan. The title text is centered in the middle of the slide.

# Airflow observability with OpenLineage

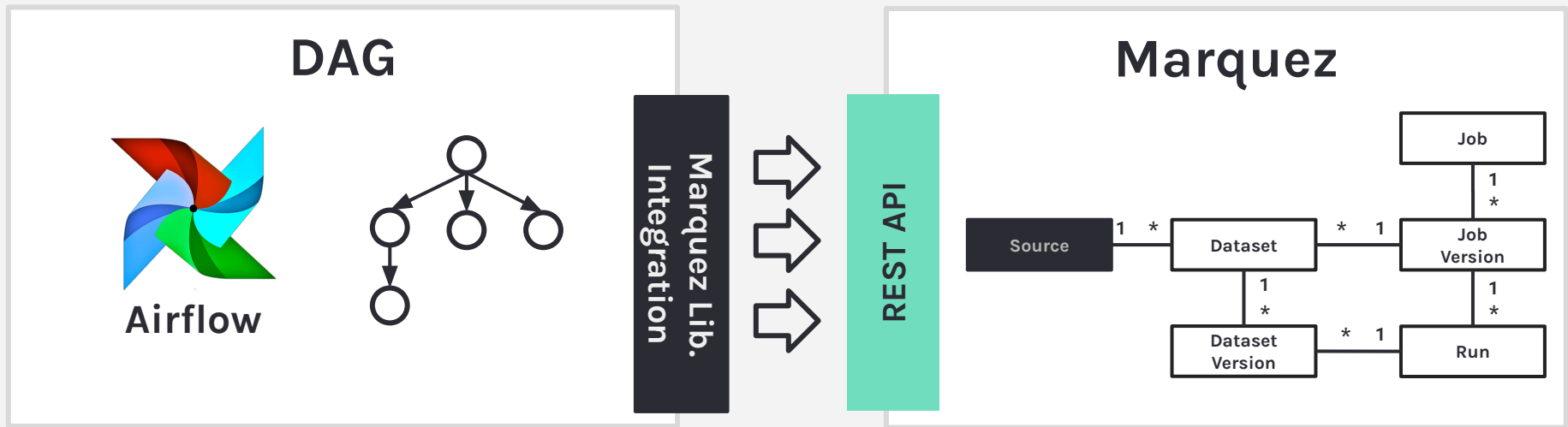
# Airflow support for Marquez





- **Metadata**
  - Task lifecycle
  - Task parameters
  - Task runs linked to **versioned** code
  - Task inputs / outputs
- **Lineage**
  - Track inter-DAG dependencies
- **Built-in**
  - SQL parser
  - Link to code builder (**GitHub**)
  - Metadata extractors



# Capturing task-level metadata in a nutshell

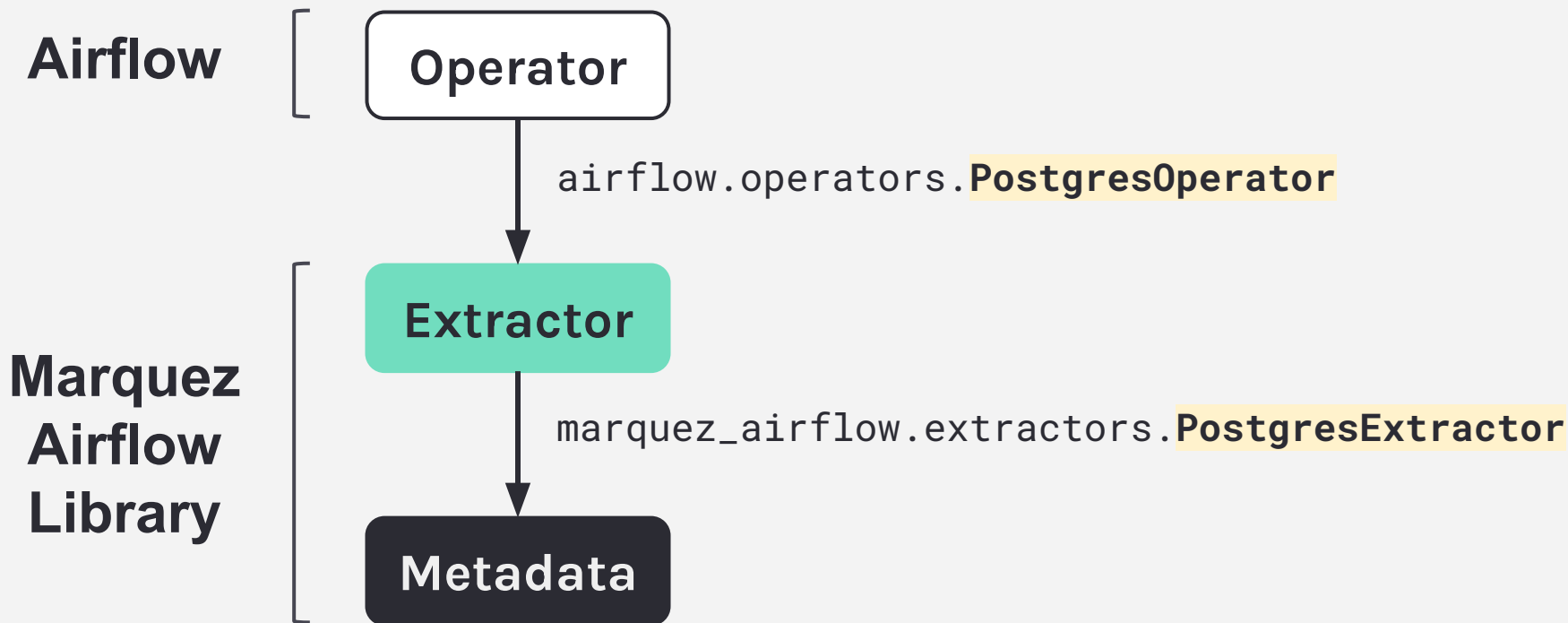


# Marquez Airflow Lib.

- Open source!  
- Enables **global** task-level metadata collection
- Extends Airflow's **DAG** class

room\_bookings\_7\_days\_dag.py

```
from marquez_airflow import DAG
from airflow.operators.postgres_operator import PostgresOperator
...
```

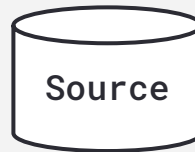


# Operator Metadata

new\_room\_booking\_dag.py

```
t1=PostgresOperator(
    task_id='new_room_booking',
    postgres_conn_id='analyticsdb',
    sql='''
        INSERT INTO room_bookings VALUES(%s, %s, %s)
    ''',
    parameters=... # room booking
)
```

01



# Operator Metadata

new\_room\_booking\_dag.py

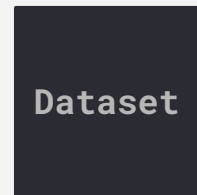
```
t1=PostgresOperator(
    task_id='new_room_booking',
    postgres_conn_id='analyticsdb',
    sql='''
        INSERT INTO room_bookings VALUES(%s, %s, %s)
    ''',
    parameters=... # room booking
)
```

01



Source

02



Dataset

# Operator Metadata

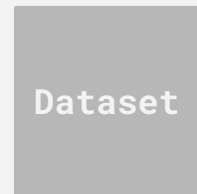
new\_room\_booking\_dag.py

```
t1=PostgresOperator(
    task_id='new_room_booking',
    postgres_conn_id='analyticsdb',
    sql='''
        INSERT INTO room_bookings VALUES(%s, %s, %s)
    ''',
    parameters=... # room booking
)
```

01



02

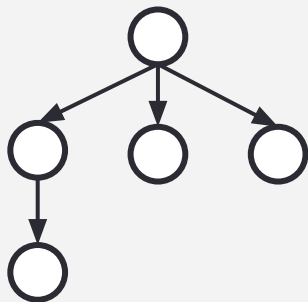


03

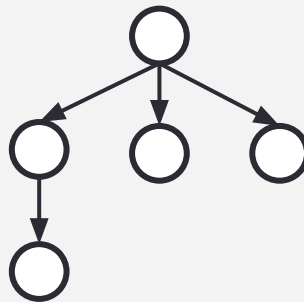


# Managing inter-DAG dependencies

`new_room_bookings_dag.py`



`top_room_bookings_dag.py`

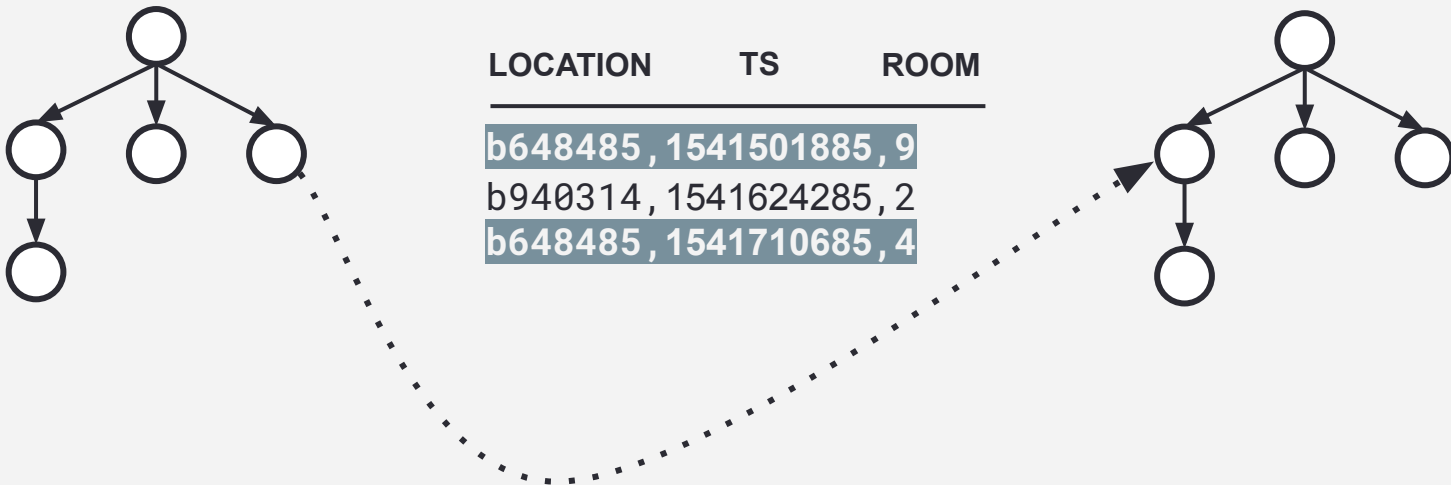


# Managing inter-DAG dependencies

new\_room\_bookings\_dag.py

public.room\_bookings

top\_room\_bookings\_dag.py







Demo

# Join the conversation

OpenLineage

[github.com/openlineage](https://github.com/openlineage)

[openlineage.slack.com](https://openlineage.slack.com)

@openlineage

[groups.google.com/g/openlineage](https://groups.google.com/g/openlineage)



[github.com/marquezproject](https://github.com/marquezproject)

[marquezproject.slack.com](https://marquezproject.slack.com)

@marquezproject