Automating Airflow Backfills with Marquez

Willy Lulciuc | Airflow Summit 22’

Hey!

I’m Willy Lulciuc

Software Engineer, Astronomer

Co-creator, Marquez

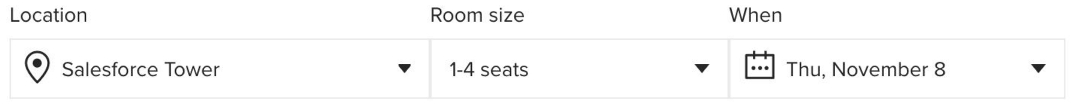
Committer, OpenLineage

@wslulciuc

AGENDA

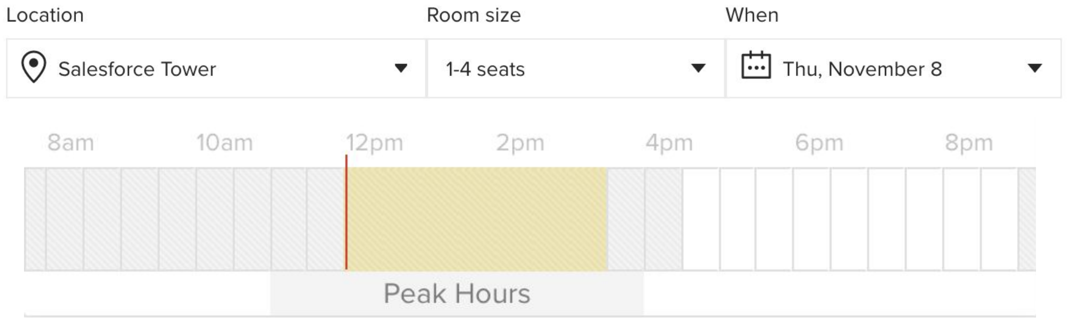
Let’s get booking!

01 Location + floor


Location
Salesforce Tower
Room size
1-4 seats
When
Thu, November 8

01 Location + floor

02 Open time slot



Location: Salesforce Tower

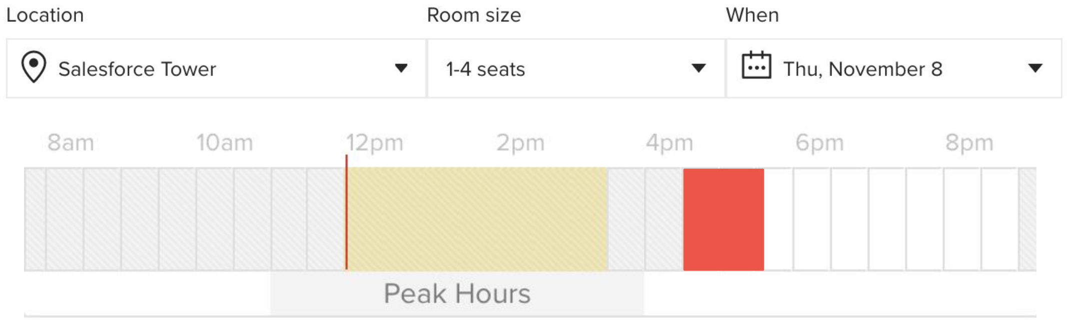
Room size: 1-4 seats

When: Thu, November 8

01 Location + floor

02 Open time slot

03 Duration


Location: Salesforce Tower

Room size: 1-4 seats

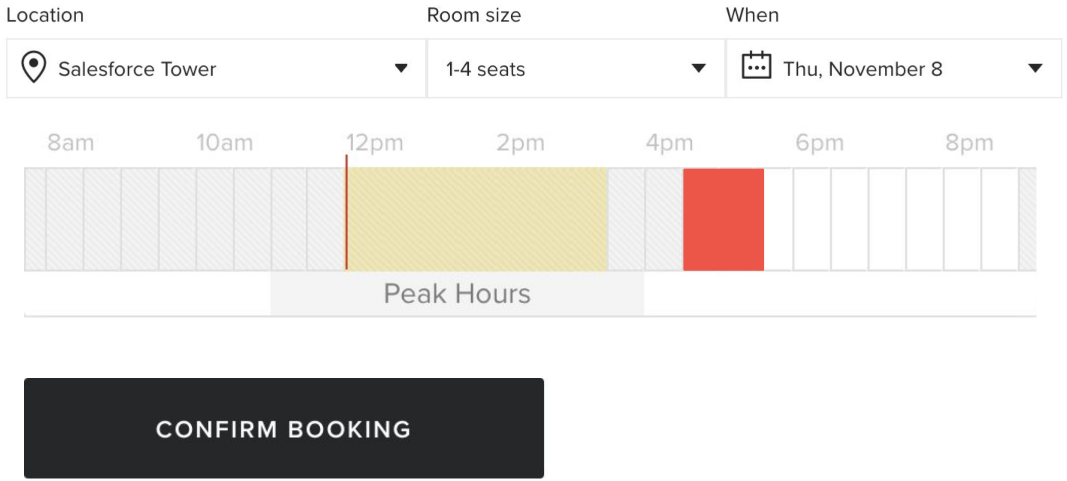
When: Thu, November 8

01 Location + floor

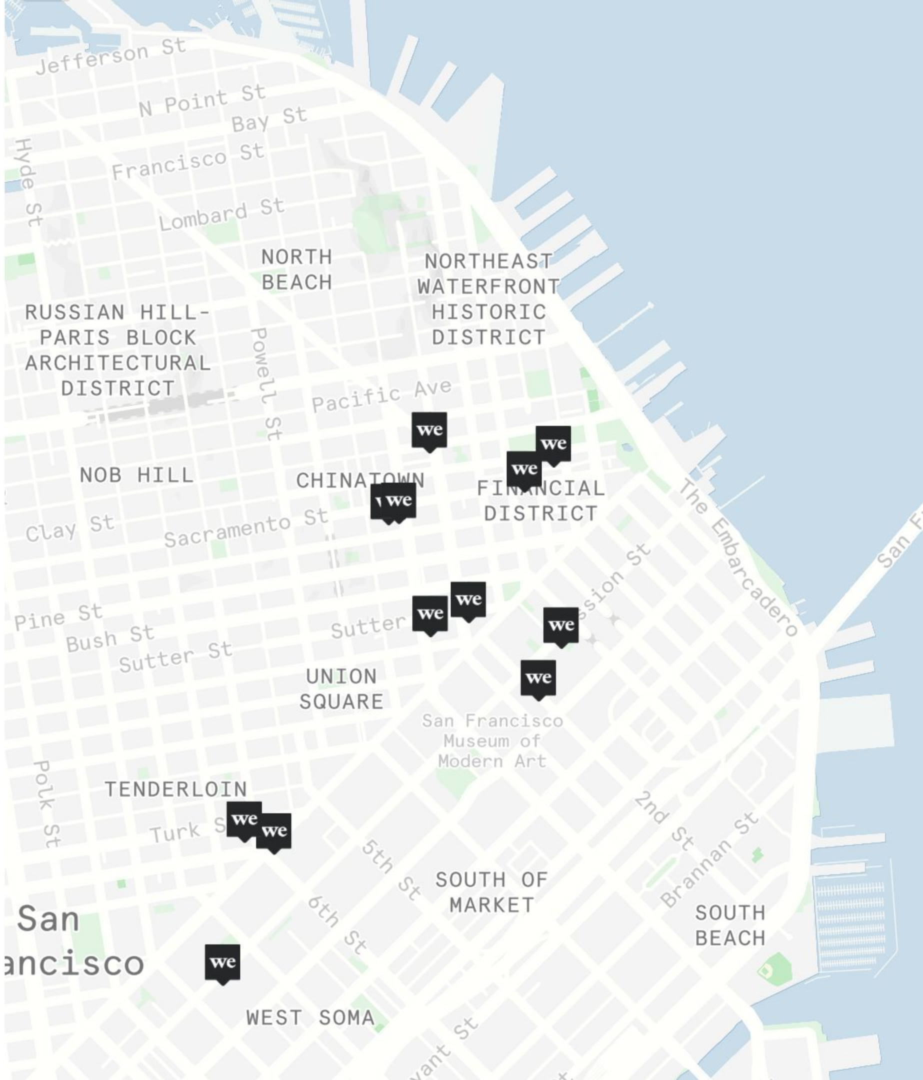
02 Open time slot

03 Duration

04 Confirm


Location
Salesforce Tower
Room size
1-4 seats
When
Thu, November 8

Which location has the most bookings?



```
Set[RoomBooking] ➡ LocationID
```

```
SELECT location,
       COUNT(*) AS bookings,
       booked_by
  FROM room_bookings
 GROUP BY bookings DESC
 LIMIT 1
```

… but, DAGs fail and backfills are a thing

Backfilling

As your organization scales up, so will the amount of data and number of internal data sources. As data outages happen, they become more serious. **Backfilling refers to the process of retroactively processing historical data.** Having a central place to examine and understand DAG dependencies will make your organization more resilient to data outages.

DAG Failures

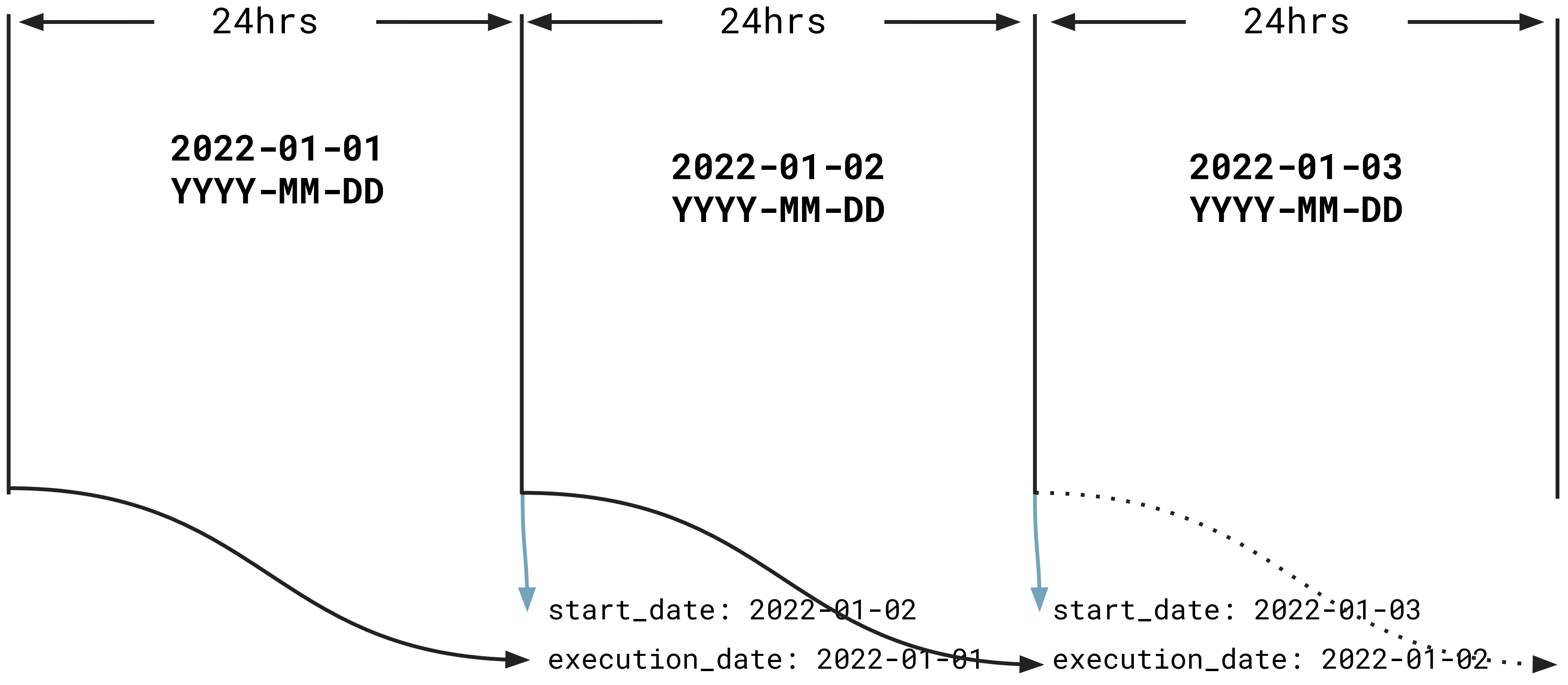
- Data quality
  - Data freshness (incomplete or missing data, etc)
  - Data schema change (column dropped, data type changed)
- Bad code
  - DAG crashes
- DAG dependencies
  - Upstream/downstream DAG failures

01 Backfills (naive)

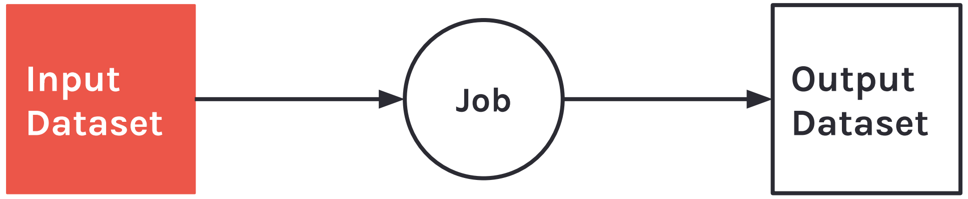
Airflow `backfill`

```
$ airflow backfill \
    --start-date <START_DATE> \
    --end-date <END_DATE> \
    <DAG_ID>
```

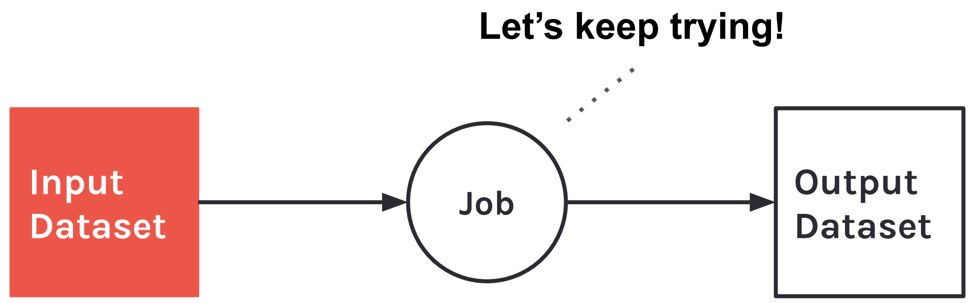
Airflow `execution_date`



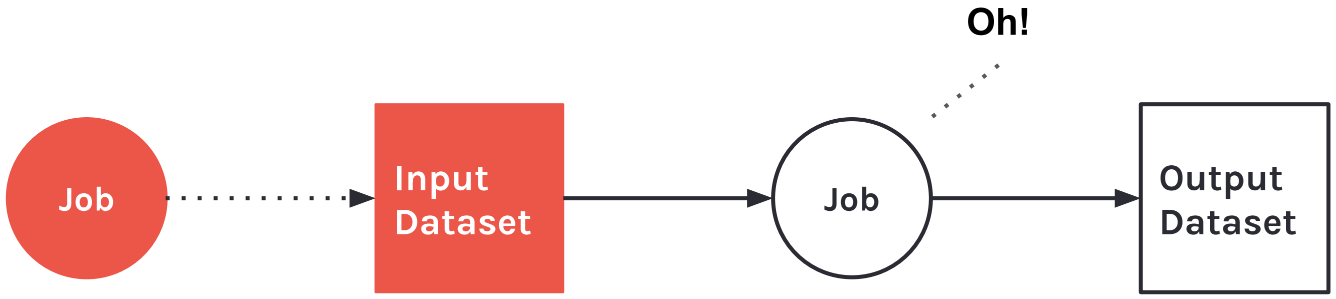
Data Quality Failures



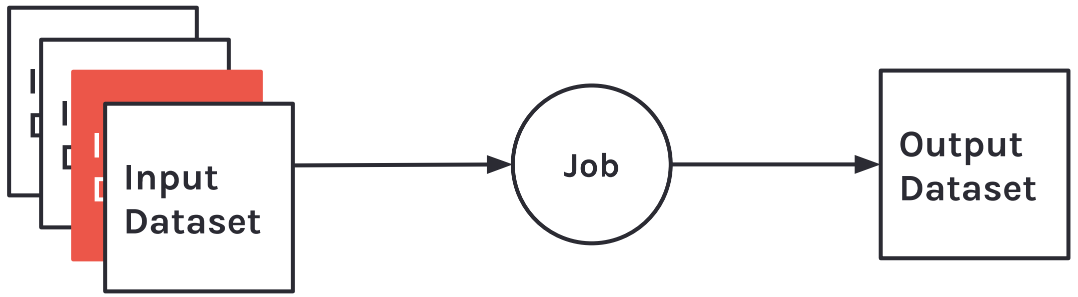
Retries!



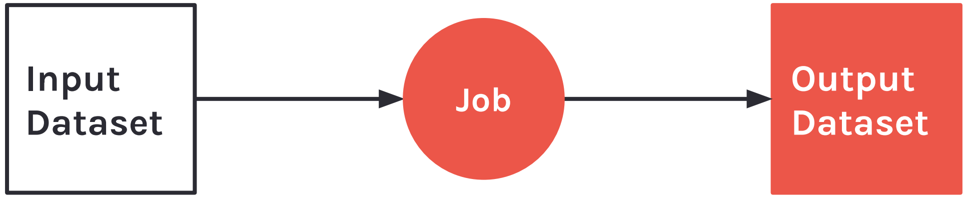
Upstream Dependency Failures



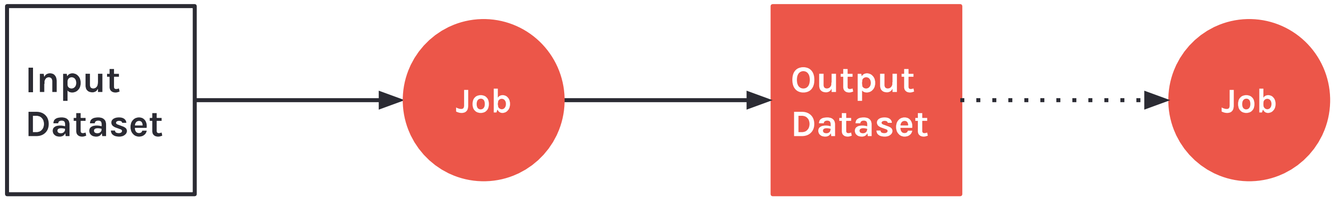
One Bad Datapoint



Bad Code Failures



Downstream Failures



Backfilling is tough…

- How quickly can data quality issues be identified and explored?
- What alerting rules should be in place to notify downstream DAGs of possible upstream processing issues or failures?
- What effects (if any) would upstream DAGs have on downstream DAGS id dataset consumption was delayed?

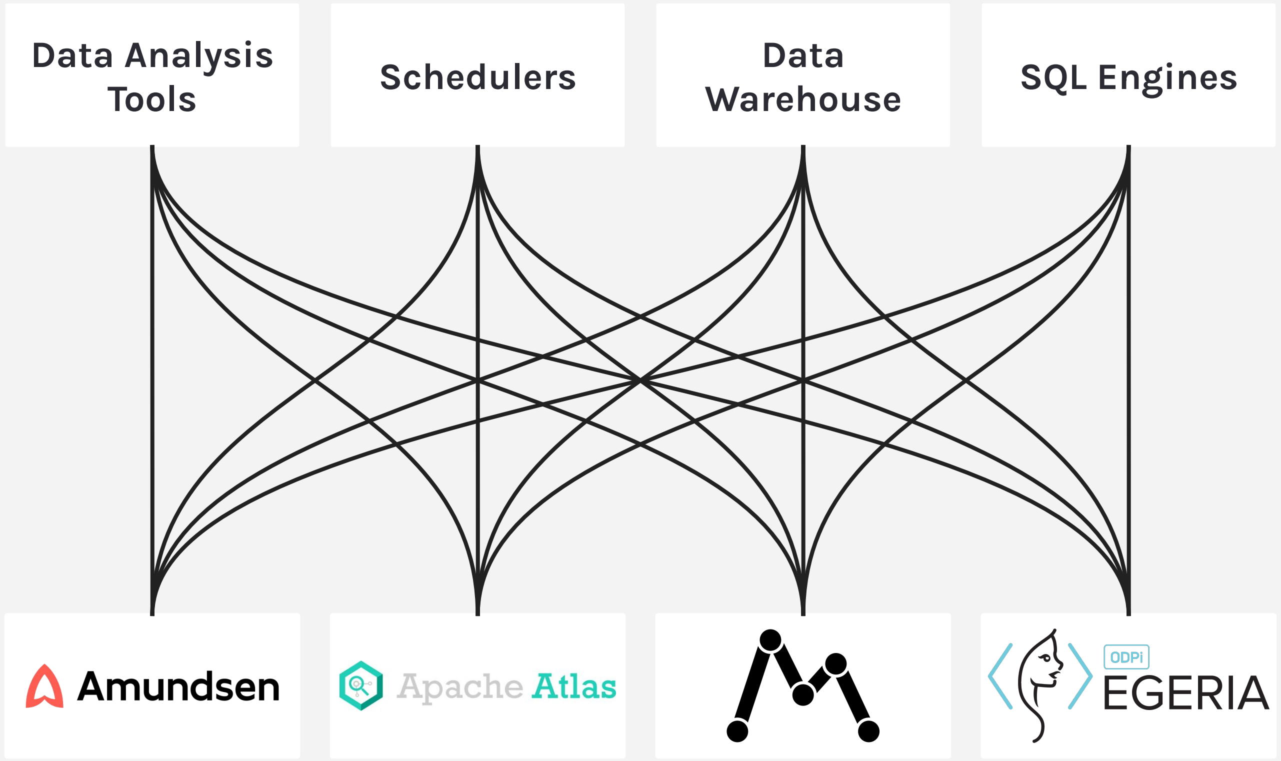
… ugh, backfills shouldn’t be this hard!

OpenLineage

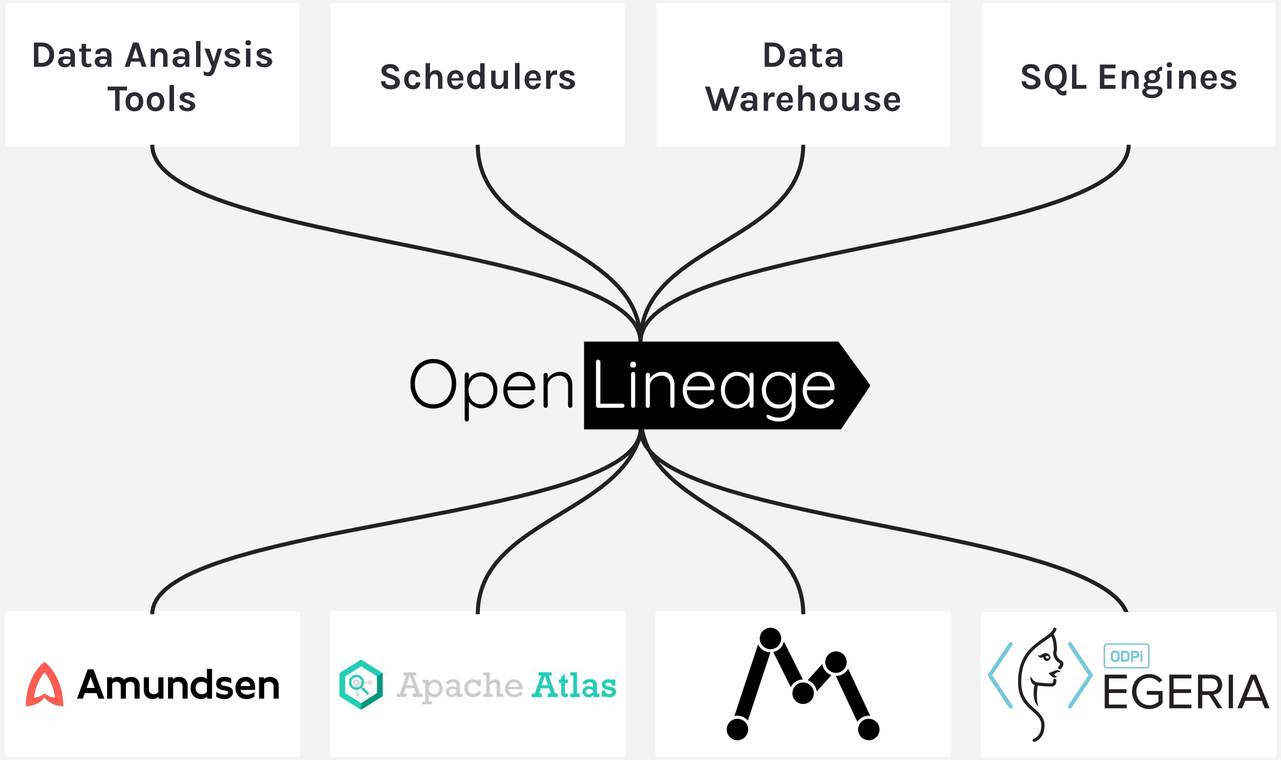
LF AI & DATA

SANDBOX PROJECT

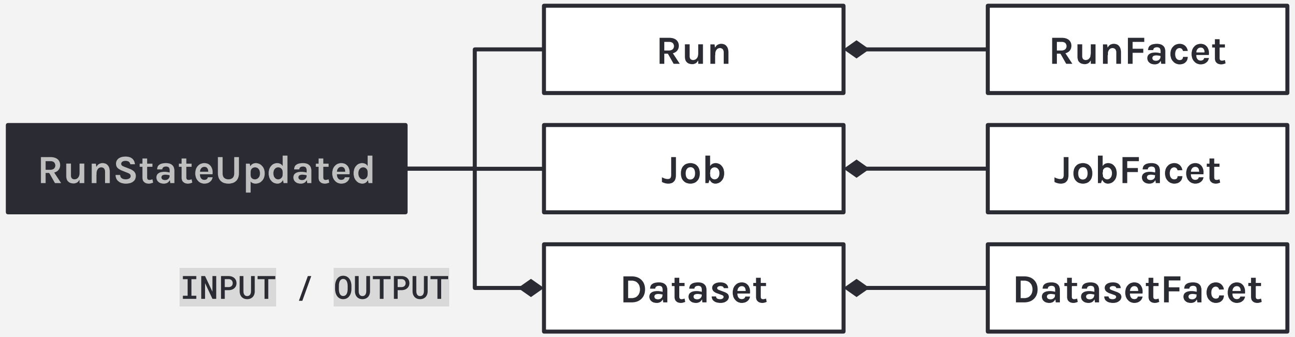
OpenLineage: Intro



OpenLineage: Intro



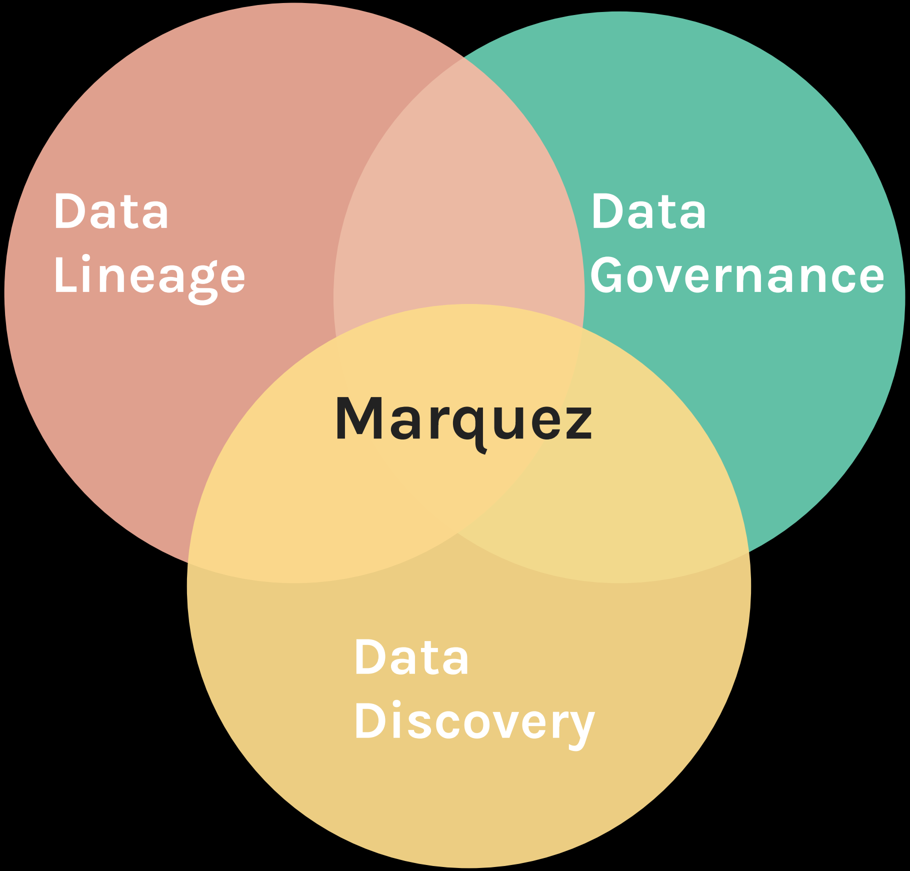
OpenLineage: Data model



- A **open standard** with a *specification* for collecting **lineage** metadata
- Focuses on **job-level** execution
  - *Datasets*
  - *Jobs*
  - *Runs*
- **Event-based** metadata collection
- **Extensible** model via **facets**

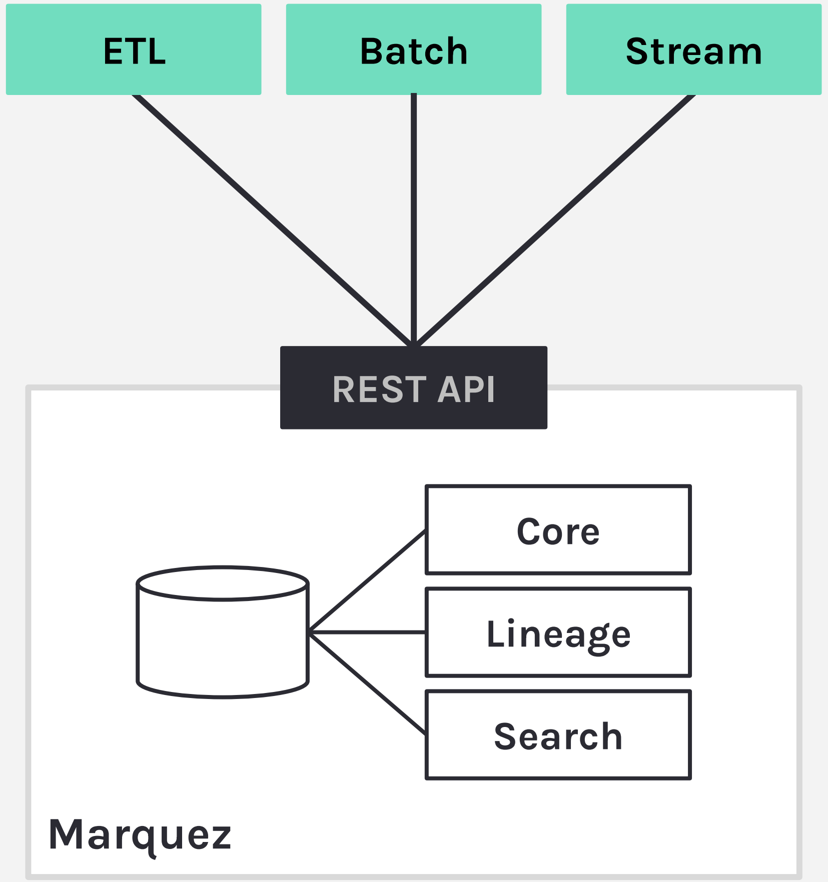
LF AI & DATA

INCUBATION PROJECT

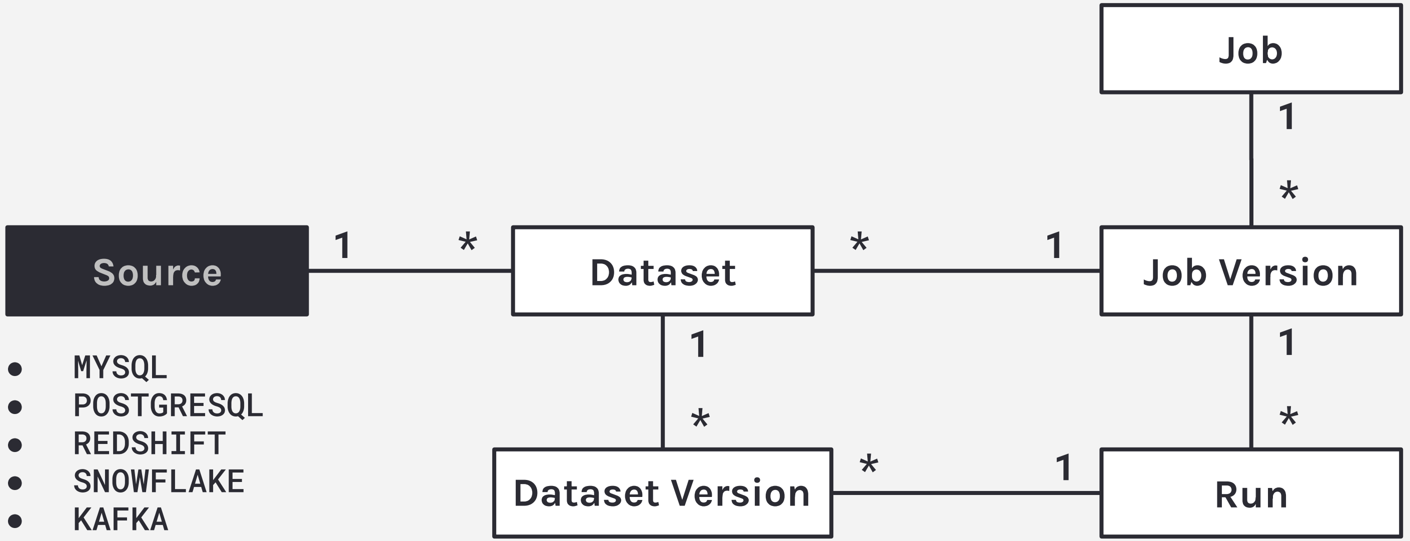

Data
Lineage
Data
Governance
Marquez
Data
Discovery

Metadata Service

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



Marquez: Data model

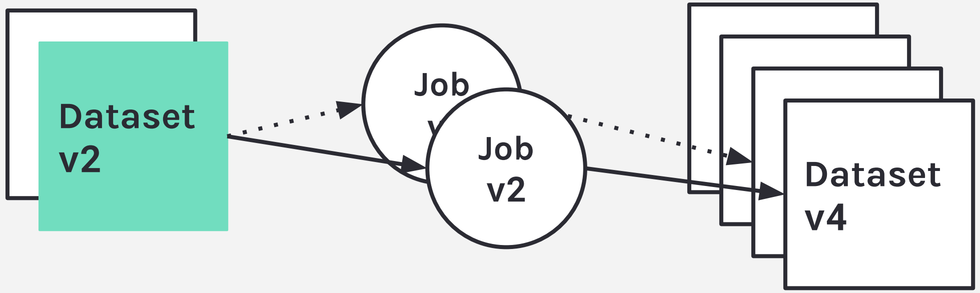


- BATCH
- STREAM
- SERVICE

- MYSQL
- POSTGRESQL
- REDSHIFT
- SNOWFLAKE
- KAFKA
- S3

Design benefits

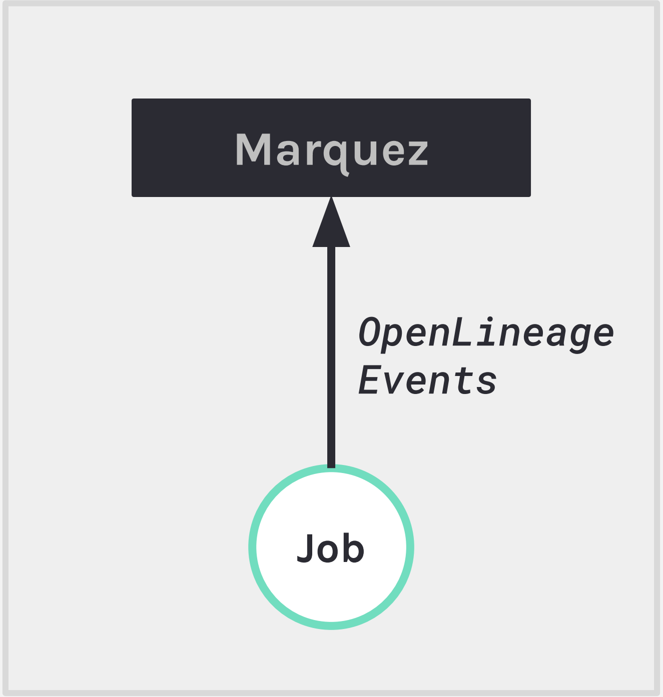
- Debugging
  - What **job version(s)** produced and consumed **dataset version X**?



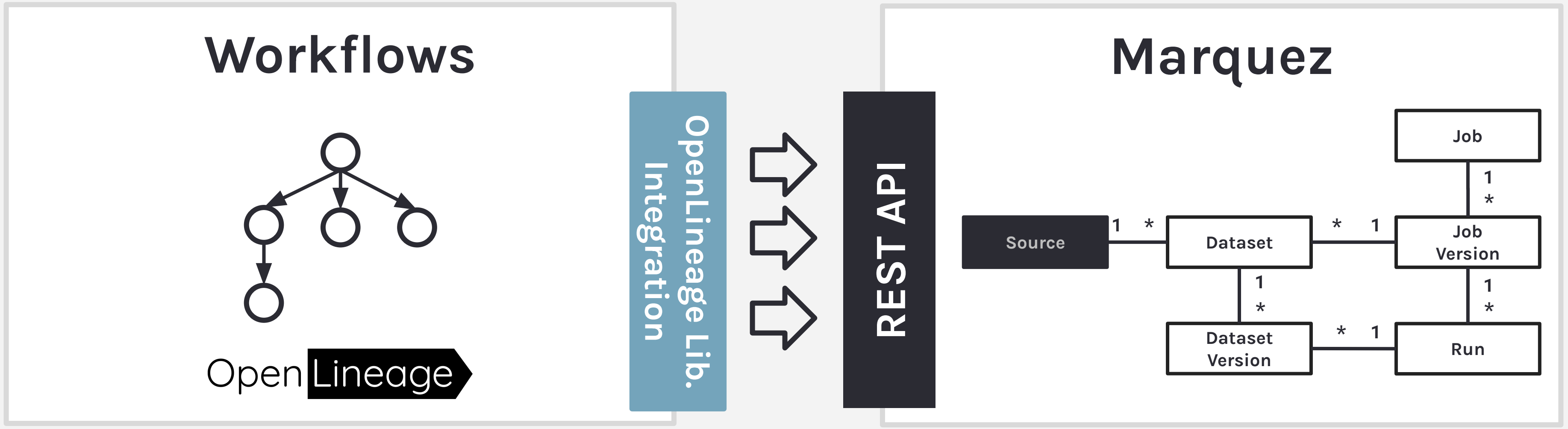
- Backfilling
  - Full / incremental processing

How is metadata collected?

- **Push-based** metadata collection
- REST API
- **OpenLineage** integrations
    - Airflow
    - Spark
    - dbt

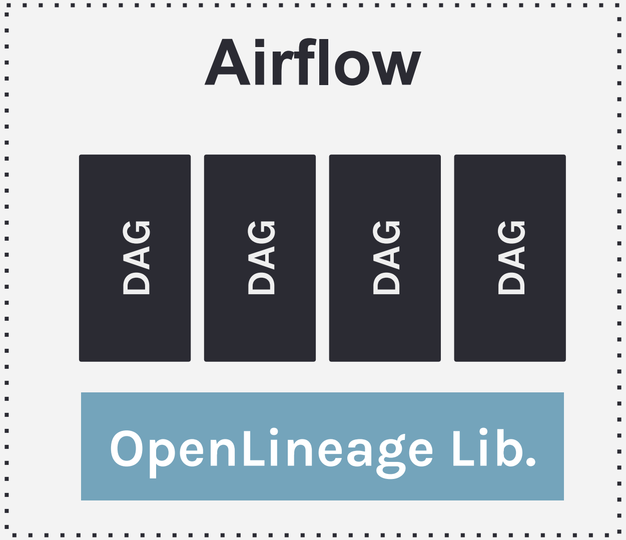


Capturing lineage metadata with Marquez using OpenLineage in a nutshell



OpenLineage + Airflow

OpenLineage support for Airflow



- **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

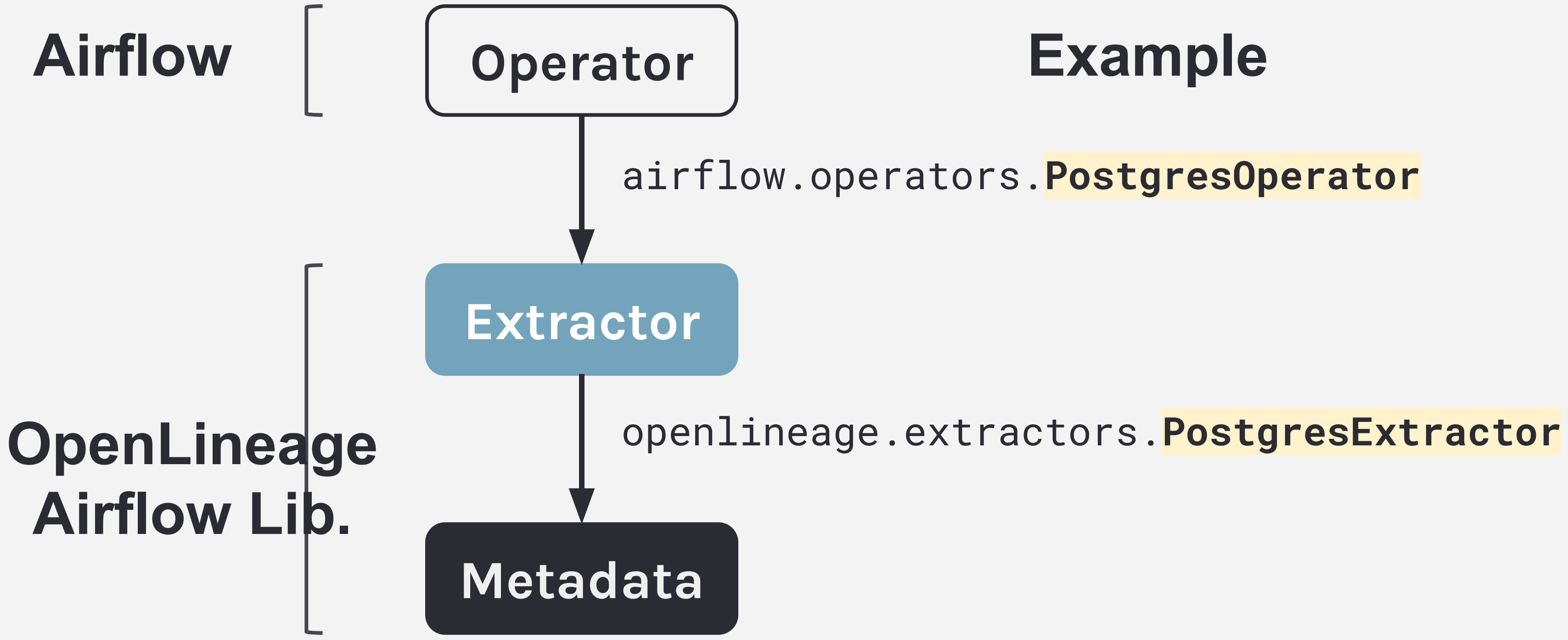
OpenLineage Airflow Lib.

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

new_room_bookings.py

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

...
```

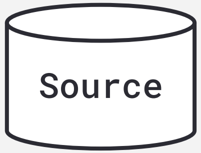

Airflow
Operator
Example
airflow.operators.PostgresOperator
Extractor
openlineage.extractors.PostgresExtractor
OpenLineage Airflow Lib.
Metadata

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
)
```

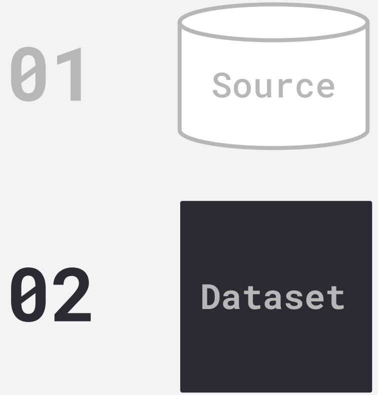
01



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
)
```



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

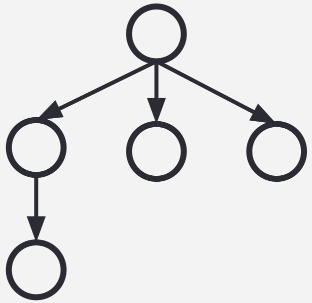
02 Dataset

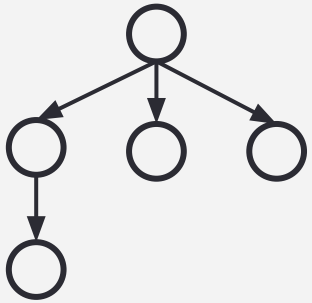
03 Job

Managing inter-DAG dependencies

`new_room_bookings_dag.py`

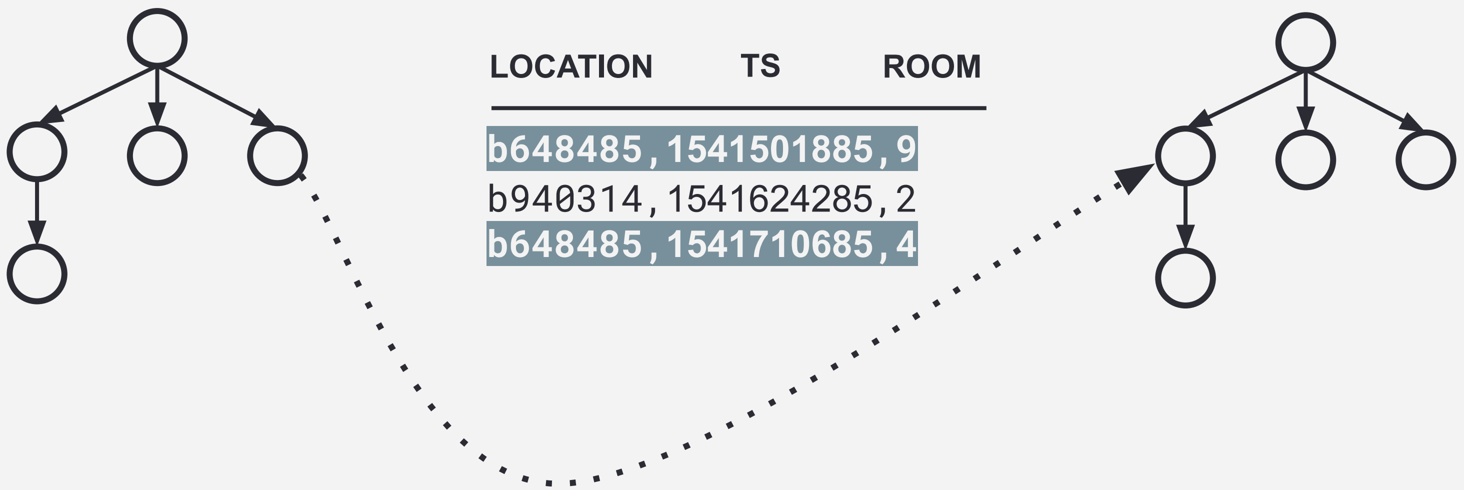
`top_room_bookings_dag.py`





Managing inter-DAG dependencies

`new_room_bookings_dag.py` `public.room_bookings` `top_room_bookings_dag.py`



LOCATION	TS	ROOM
b648485	1541501885	9
b940314	1541624285	2
b648485	1541710685	4

04 Backfills (take 2)

Fail Collaboratively

- **Global View**
  - Lineage metadata allows teams to look at failures across the organization, understanding the impact of the data outage
- **Coordinate**
  - Efforts aren’t duplicated
- **Empower**
  - Give teams the power to resolve data outages completely

05 Future work

Roadmap

- Column-level lineage support
- Job hierarchy and grouping
- Flink integration

Thanks! <o/

Be cool, take the Airflow survey!

bit.ly/AirflowSurvey22

github.com/OpenLineage

@OpenLineage

github.com/MarquezProject

@MarquezProject

Questions?