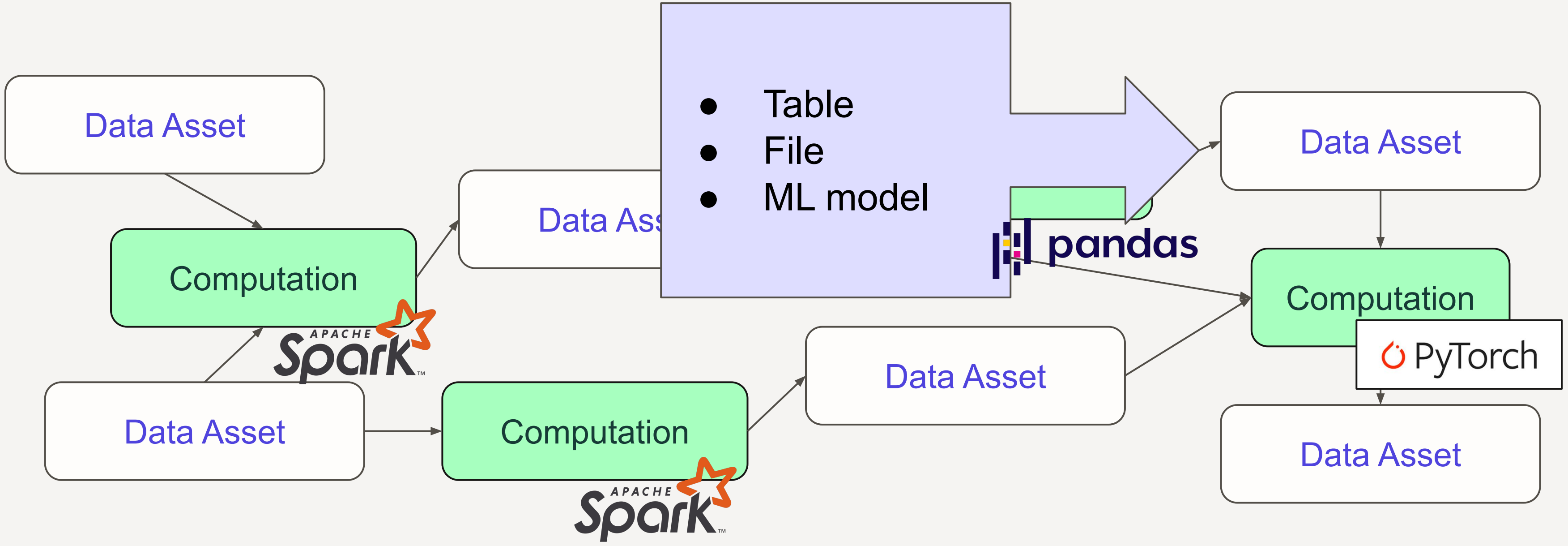
dagster

Asset-Based Data Orchestration

Sandy Ryza (@s_ryz)
Lead Engineer, Dagster Project - Elementl

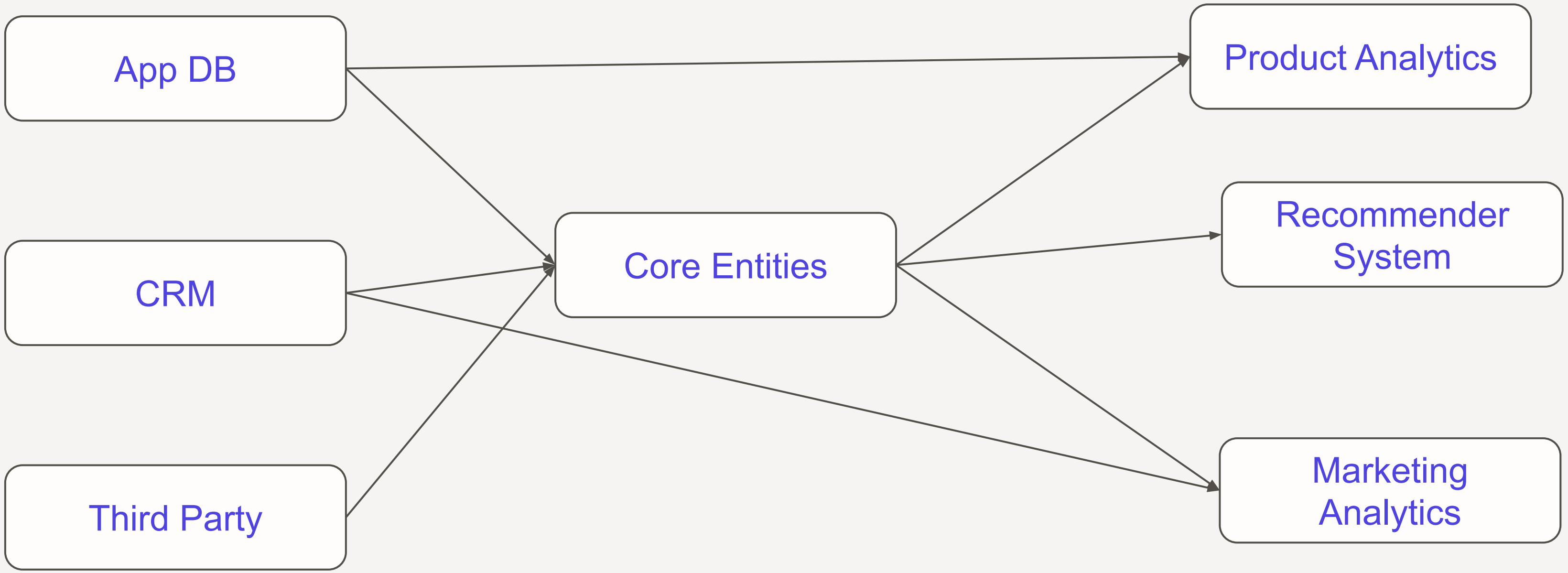
Data practitioners build and maintain data pipelines

What’s a data pipeline?



- Table
- File
- ML model

Data pipelines span entire organizations

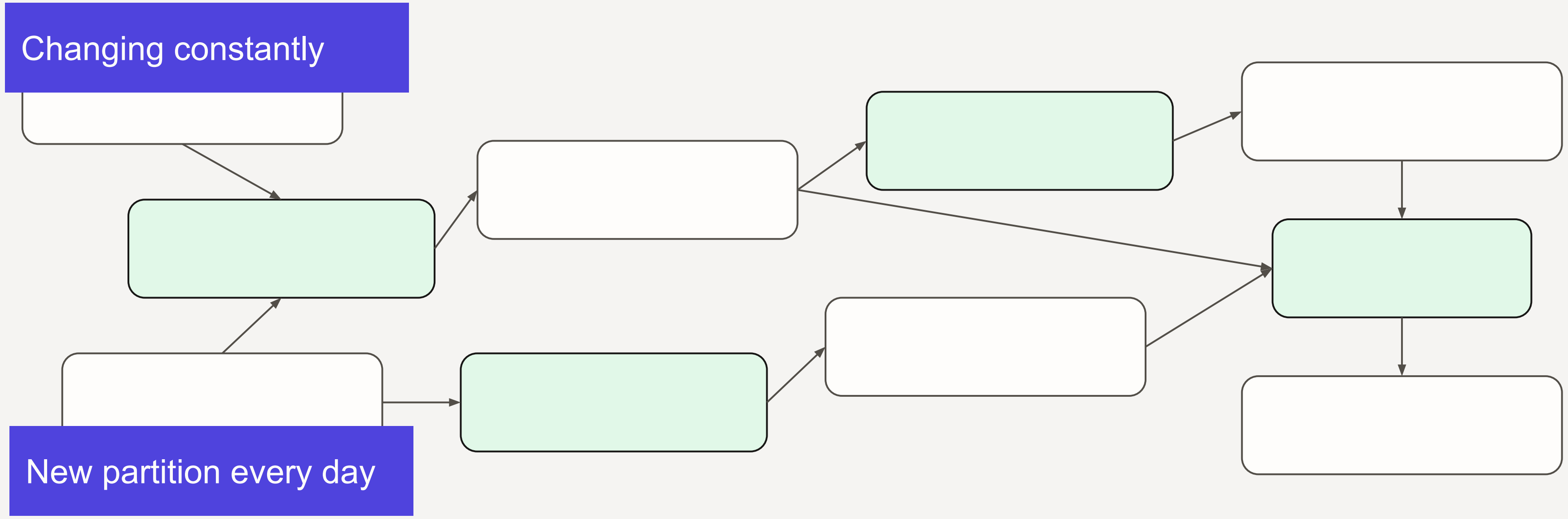


Automatically updating data assets

Why update a data asset?

- Inputs have changed

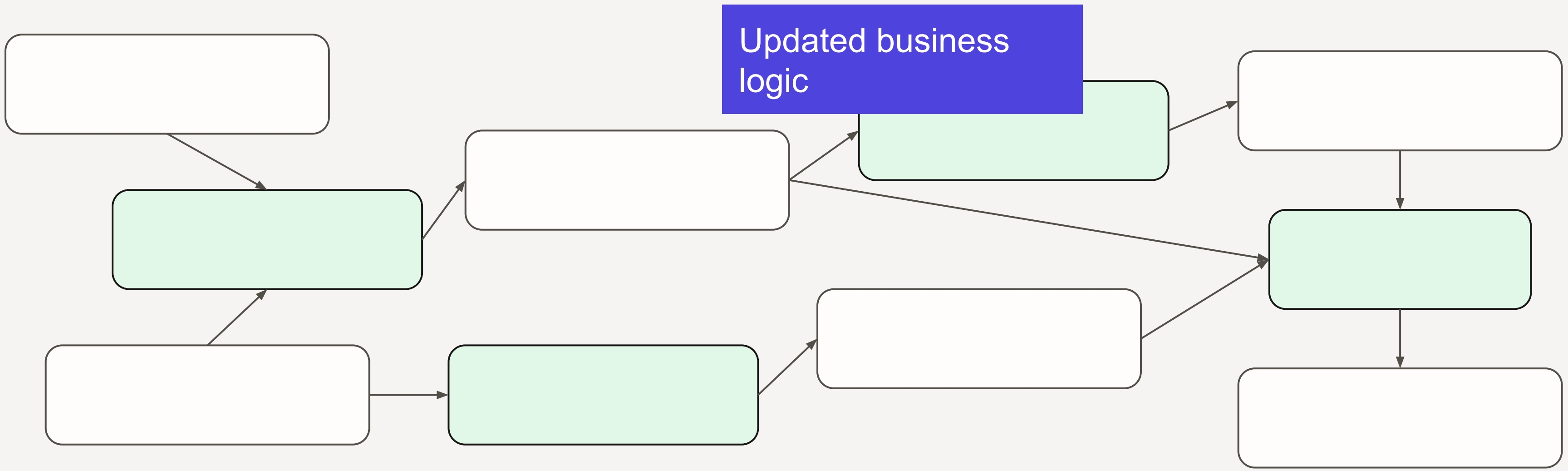
Changing inputs



Why update a data asset?

- Inputs have changed
- Code has changed

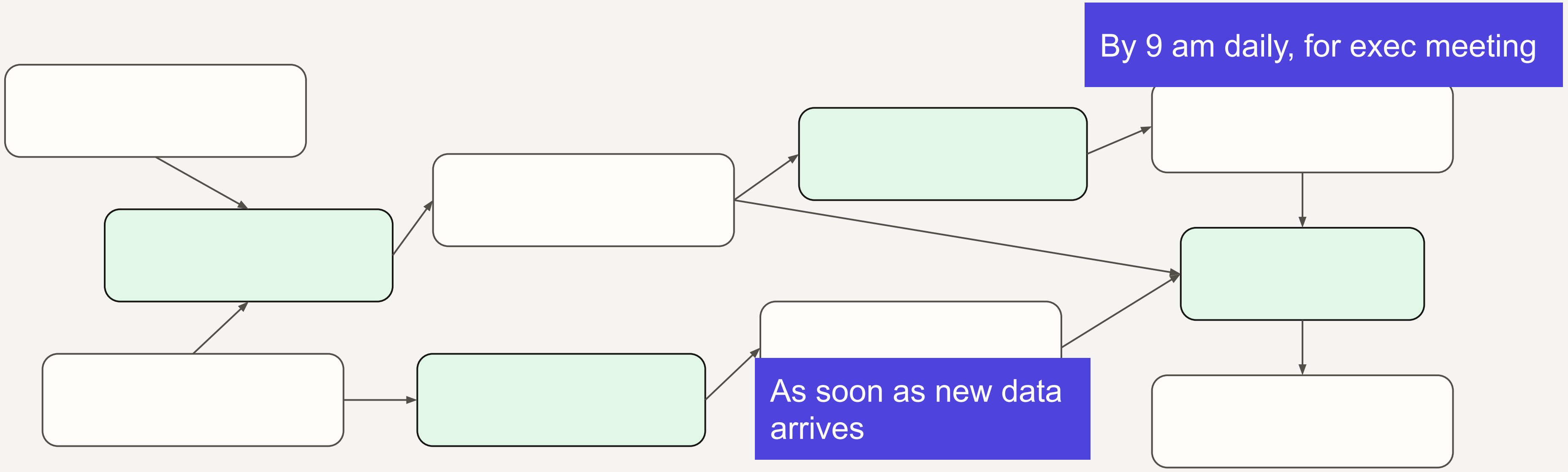
Code changes



Why update a data asset?

- Inputs have changed
- Code has changed
- Fresh data is needed

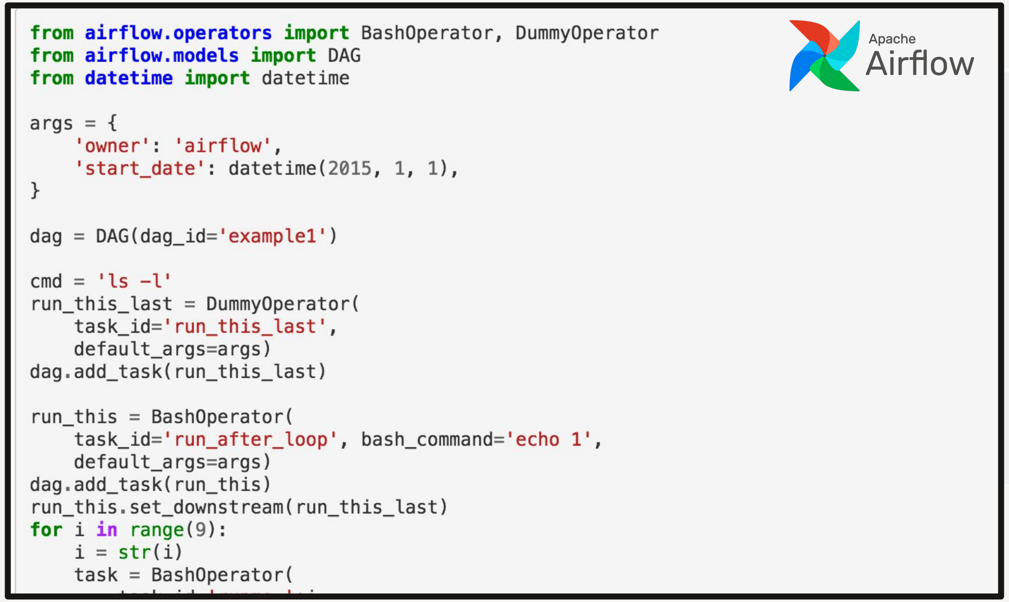
Fresh data is needed



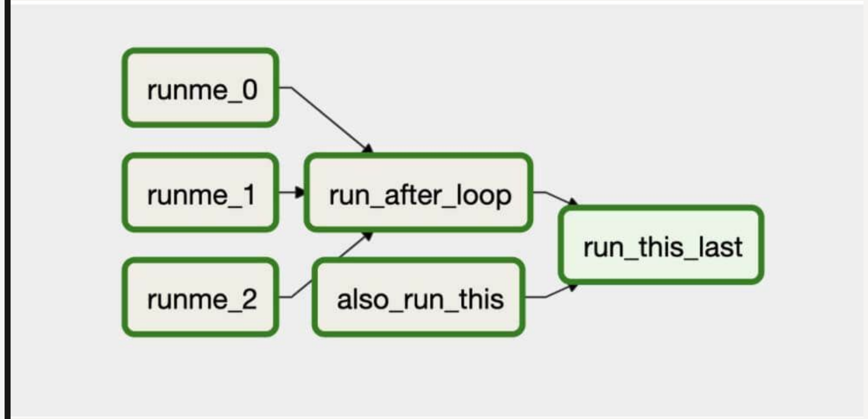
Automatically updating data assets: how?

The status quo: **workflow engines**

- DAG of tasks
- Run the DAG every hour/day/whatever



```
from airflow.operators import BashOperator, DummyOperator
from airflow.models import DAG
from datetime import datetime

args = {
    'owner': 'airflow',
    'start_date': datetime(2015, 1, 1),
}

dag = DAG(dag_id='example1')

cmd = 'ls -l'
run_this_last = DummyOperator(
    task_id='run_this_last',
    default_args=args)
dag.add_task(run_this_last)

run_this = BashOperator(
    task_id='run_after_loop', bash_command='echo 1',
    default_args=args)
dag.add_task(run_this)
run_this.set_downstream(run_this_last)
for i in range(9):
    i = str(i)
    task = BashOperator(
```



Workflow engines: not actually the best way to schedule data pipelines?

- Forces running in lockstep
  - Caught between doing redundant work and stale data
- Code management
  - What DAG should this new data asset be a part of?
  - Monolithic DAG objects
- Alerts when tasks fail vs. when data is late

A different way: Asset-based orchestration

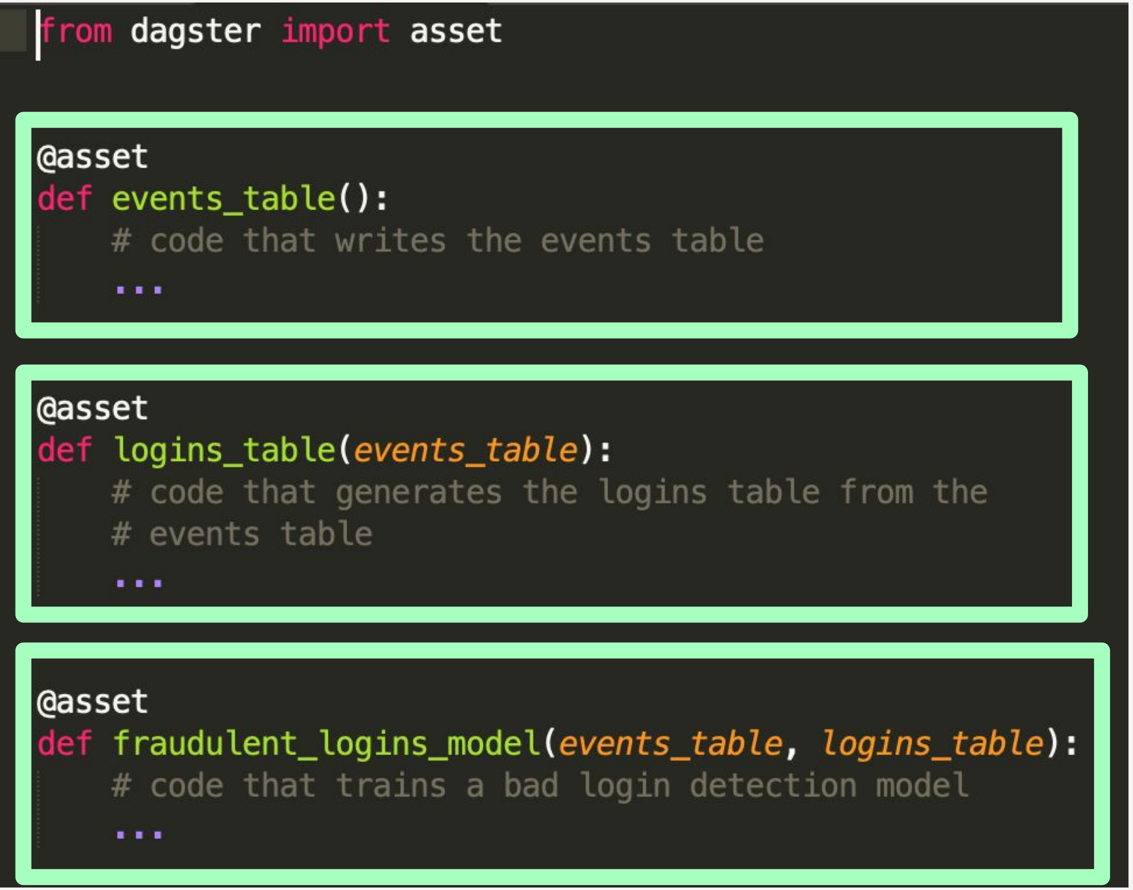
Goals of asset-based orchestration

- Outcomes
  - Make data ready on time
  - Avoid redundant work
- Express scheduling in terms of the data assets
  - When does source data change?
  - How fresh do data assets need to be?
- Understand scheduling decisions

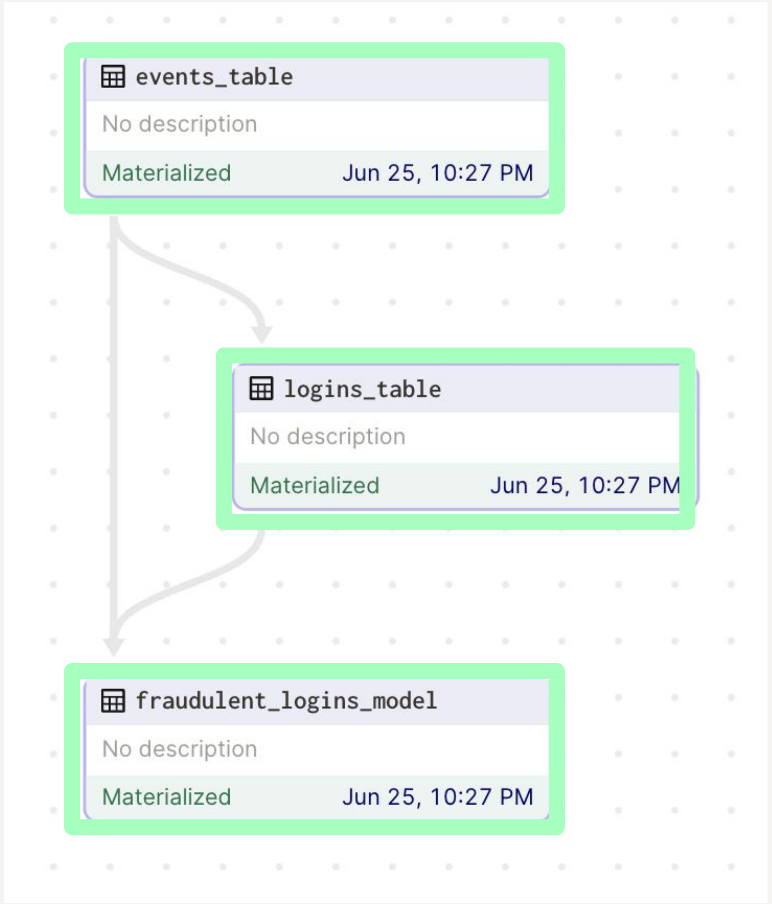
dagster

Building a pipeline

aka defining some data assets



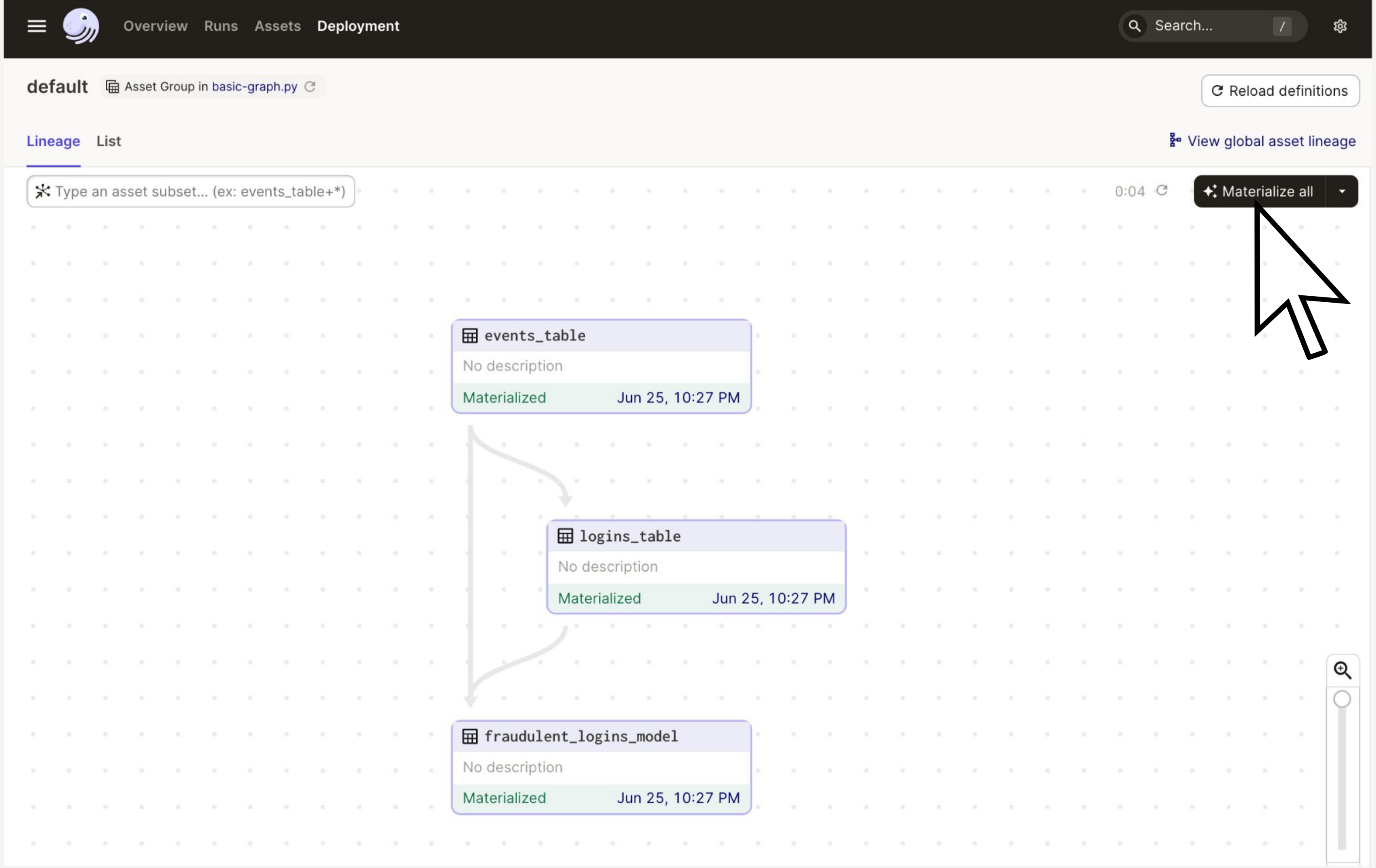
```
from dagster import asset

@asset
def events_table():
    # code that writes the events table
    ...

@asset
def logins_table(events_table):
    # code that generates the logins table from the
    # events table
    ...

@asset
def fraudulent_logins_model(events_table, logins_table):
    # code that trains a bad login detection model
    ...
```



events_table
No description
Materialized Jun 25, 10:27 PM

logins_table
No description
Materialized Jun 25, 10:27 PM

fraudulent_logins_model
No description
Materialized Jun 25, 10:27 PM


Overview
Runs
Assets
Deployment
Search...
default
Asset Group in basic-graph.py
Reload definitions
Lineage
List
View global asset lineage
Type an asset subset... (ex: events_table+*)
0:04
Materialize all

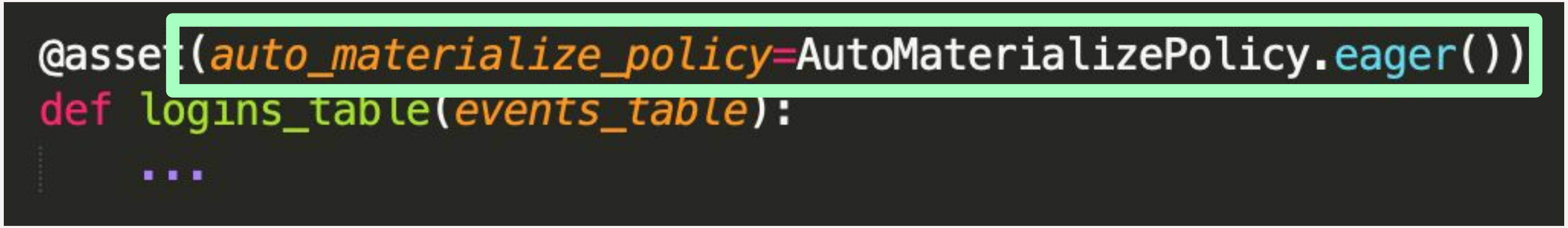
events_table
No description
Materialized Jun 25, 10:27 PM

logins_table
No description
Materialized Jun 25, 10:27 PM

fraudulent_logins_model
No description
Materialized Jun 25, 10:27 PM

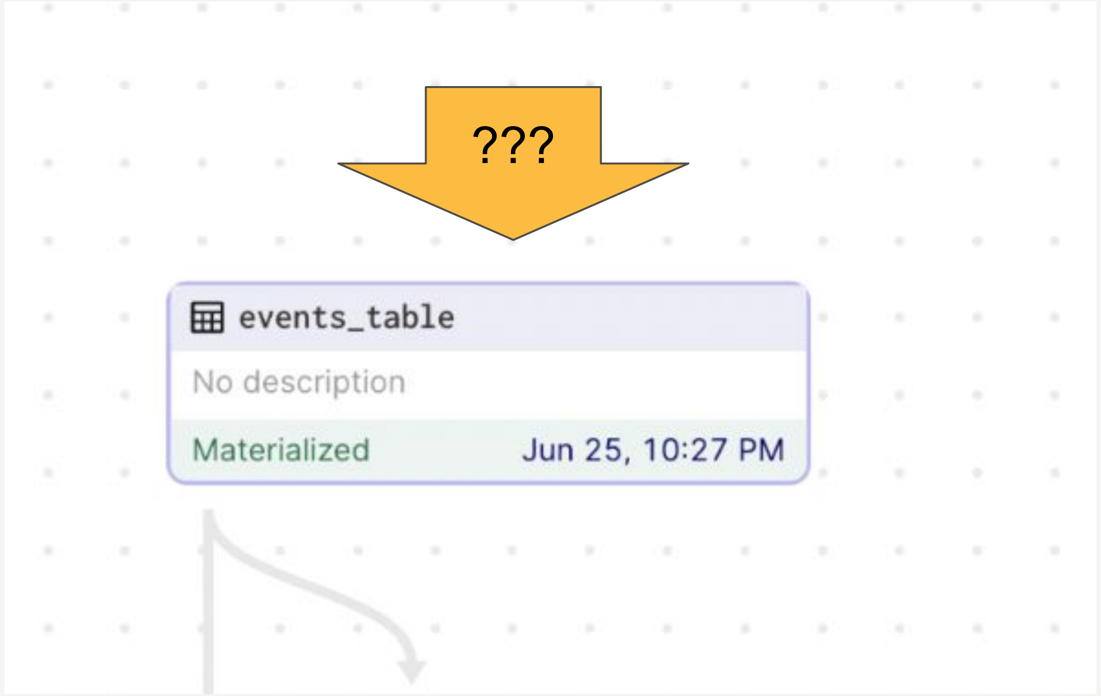
Asset-based orchestration in Dagster

Auto-materialize policies



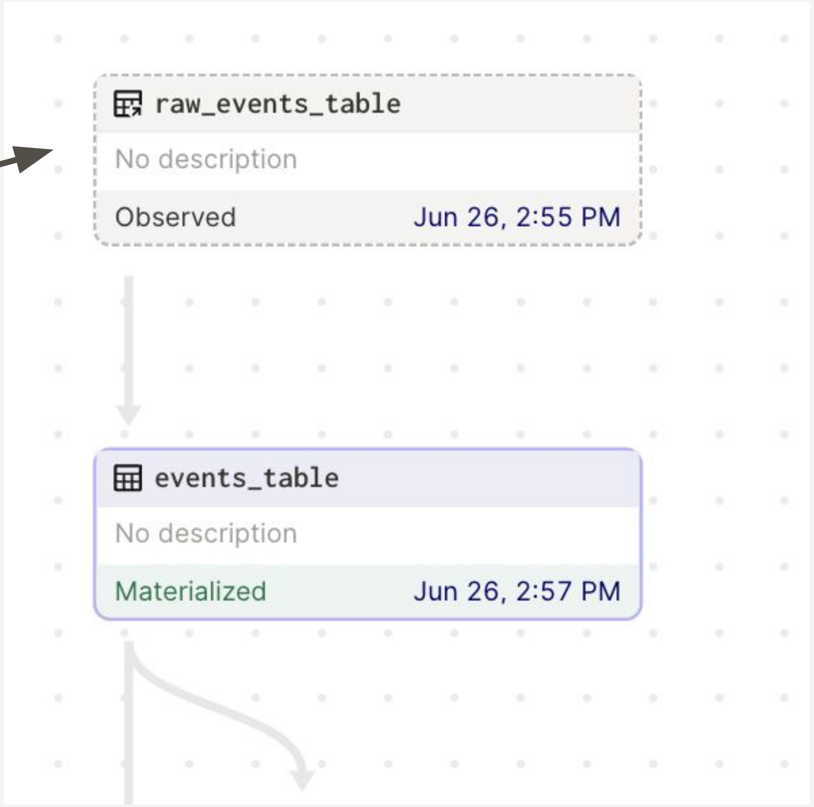
```
@asset(auto_materialize_policy=AutoMaterializePolicy.eager())
def logins_table(events_table):
    ...
```

The root of the graph

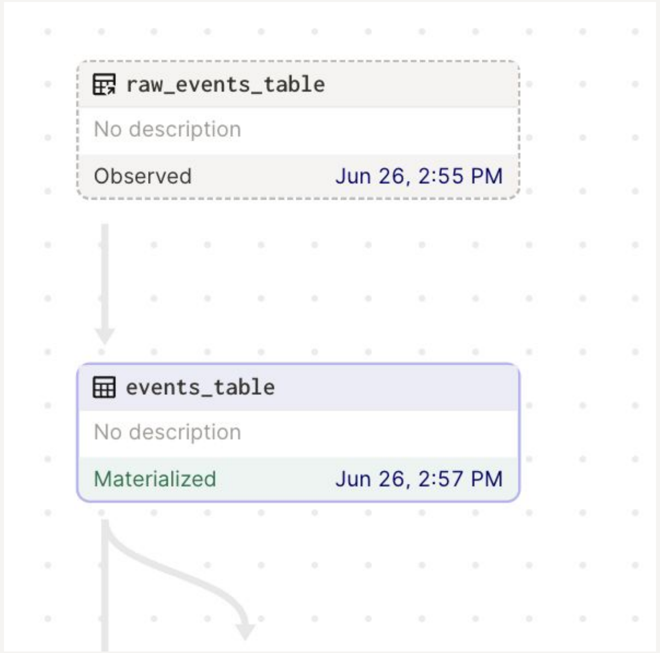


Source assets

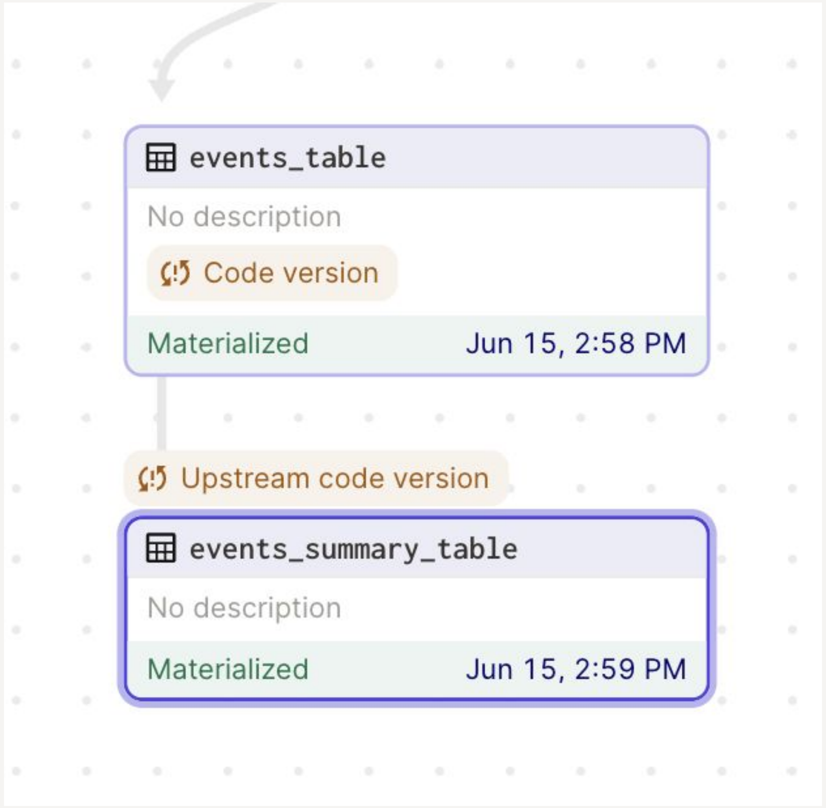
source asset



```python
@observable_source_asset
def raw_events_table():
    path = "raw_data_bucket/raw_events.parquet"
    last_modified_datetime = get_last_modified_datetime(path)
    return DataVersion(str(last_modified_datetime))
```



What about code changes?



events_table

No description

Code version

Materialized Jun 15, 2:58 PM

Upstream code version

events_summary_table

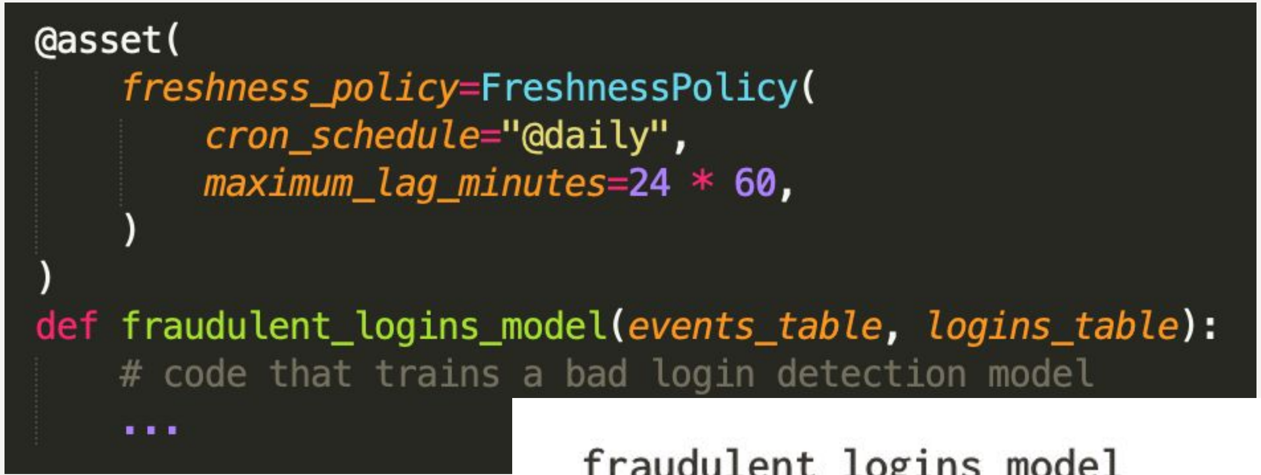
No description

Materialized Jun 15, 2:59 PM

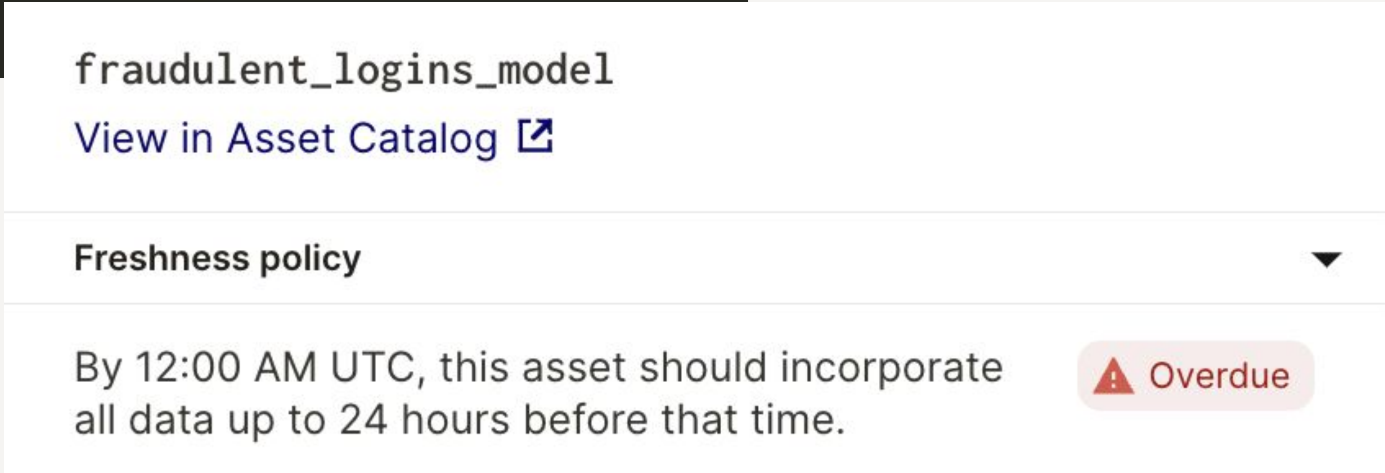
Lazy auto-materialization

$$\begin{tikzcd}\boxed{\text{Upstream asset}} \arrow[r] & \boxed{\text{Downstream asset}}\end{tikzcd}$$

Freshness policies



```
@asset(
    freshness_policy=FreshnessPolicy(
        cron_schedule="@daily",
        maximum_lag_minutes=24 * 60,
    )
)
def fraudulent_logins_model(events_table, logins_table):
    # code that trains a bad login detection model
    ...
```



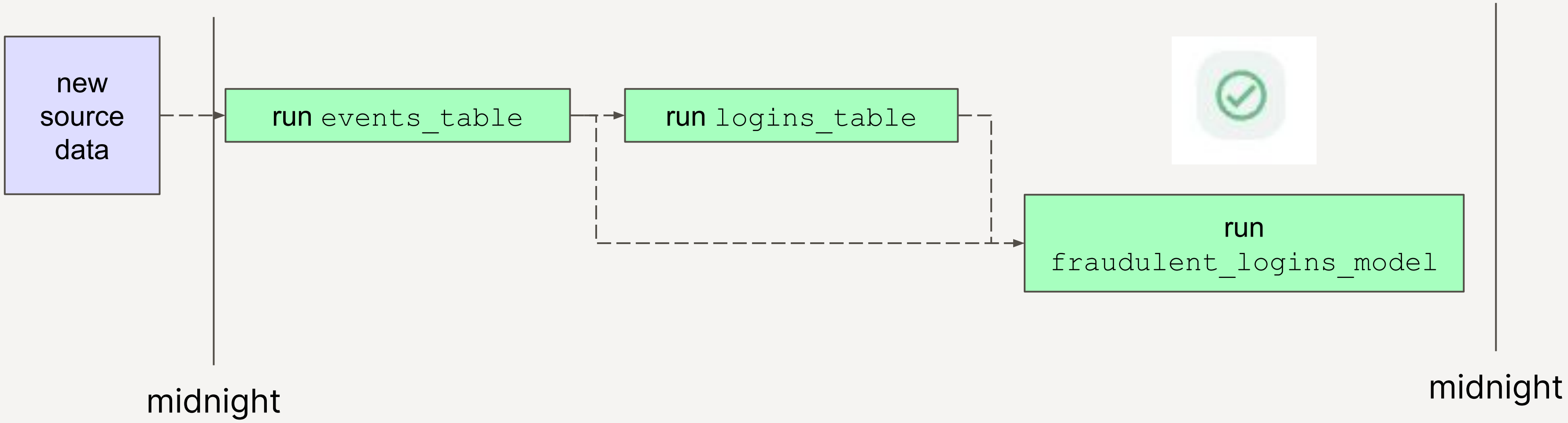
fraudulent_logins_model

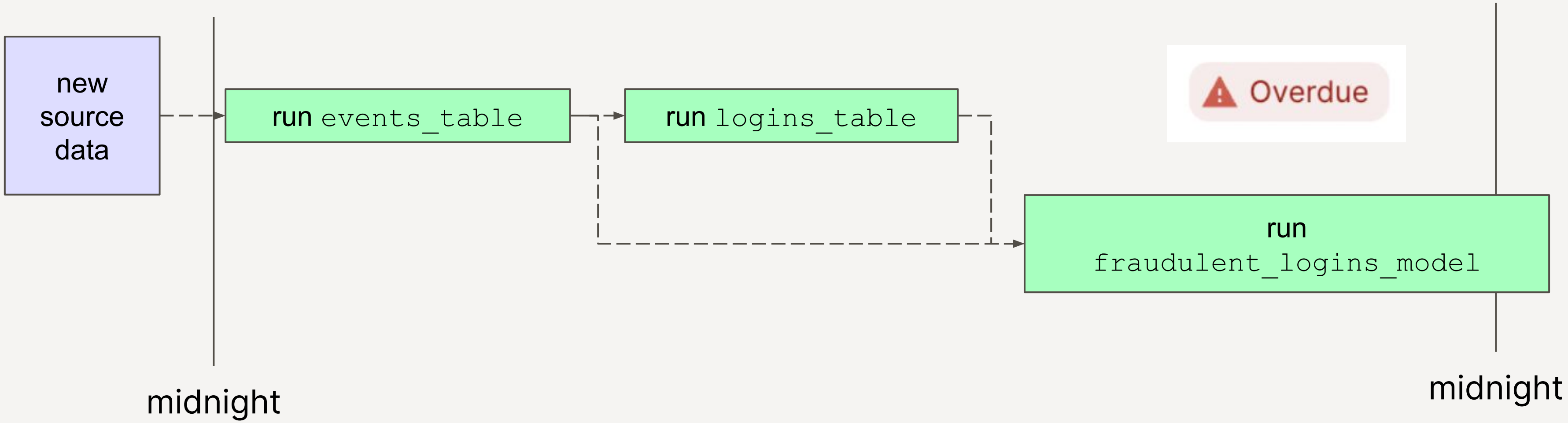
View in Asset Catalog

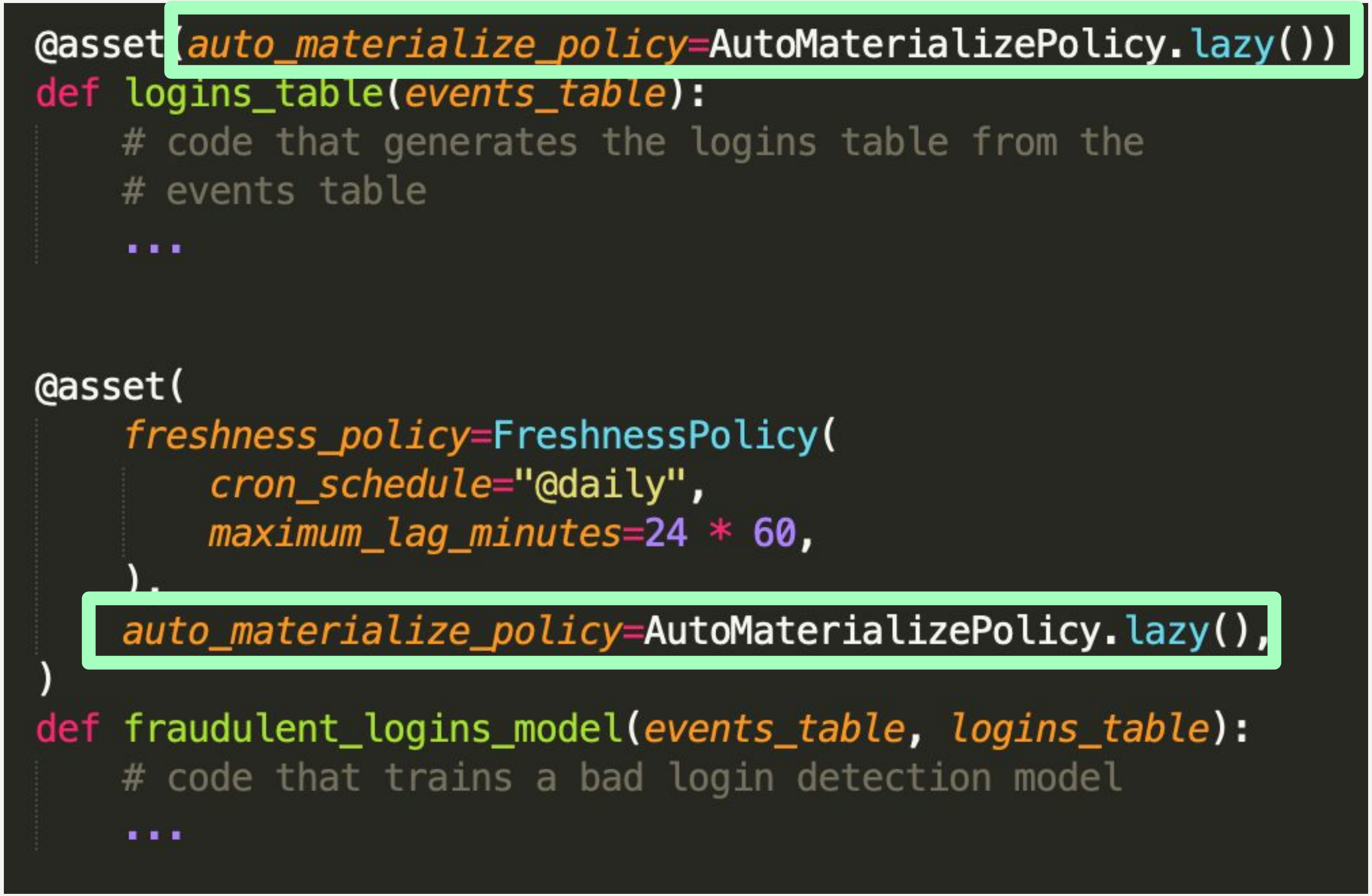
Freshness policy

By 12:00 AM UTC, this asset should incorporate all data up to 24 hours before that time.

Overdue

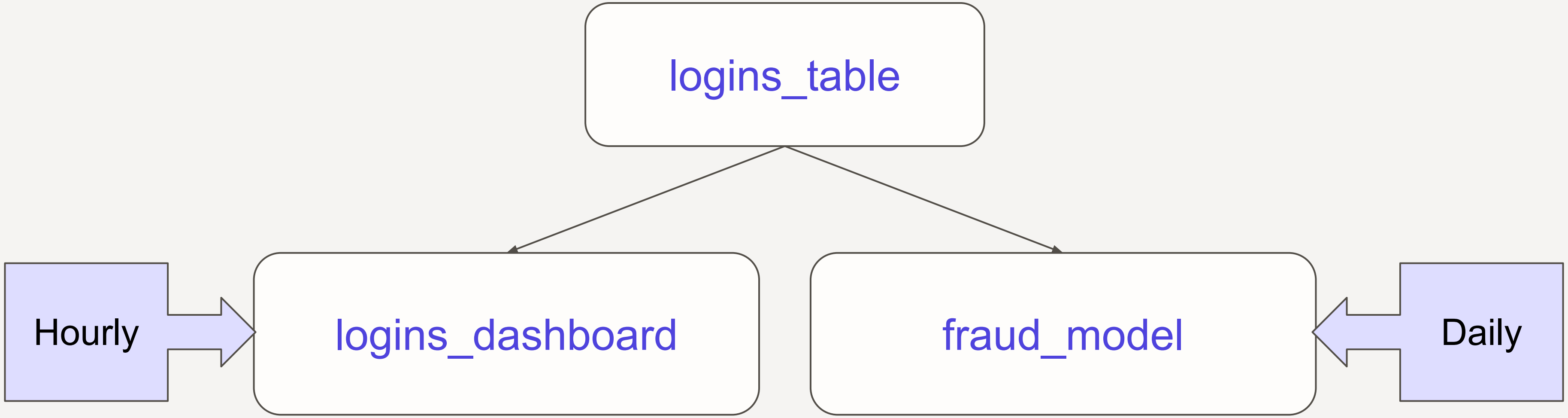

new source data
run events_table
run logins_table
run fraudulent_logins_model
midnight
midnight


new source data
midnight
run events_table
run logins_table
Overdue
run fraudulent_logins_model
midnight

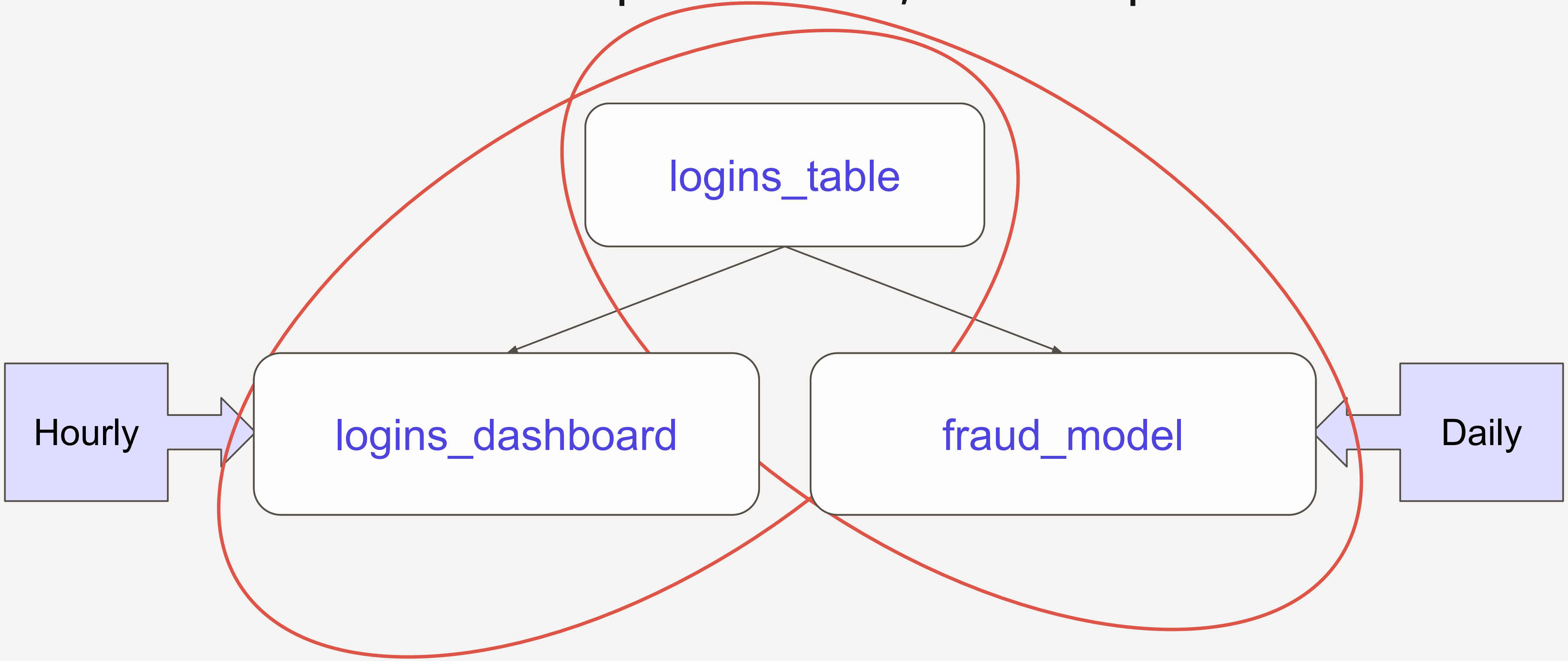


```python
@asset(auto_materialize_policy=AutoMaterializePolicy.lazy())
def logins_table(events_table):
    # code that generates the logins table from the
    # events table
    ...


@asset(
    freshness_policy=FreshnessPolicy(
        cron_schedule="@daily",
        maximum_lag_minutes=24 * 60,
    ),
    auto_materialize_policy=AutoMaterializePolicy.lazy(),
)
def fraudulent_logins_model(events_table, logins_table):
    # code that trains a bad login detection model
    ...
```

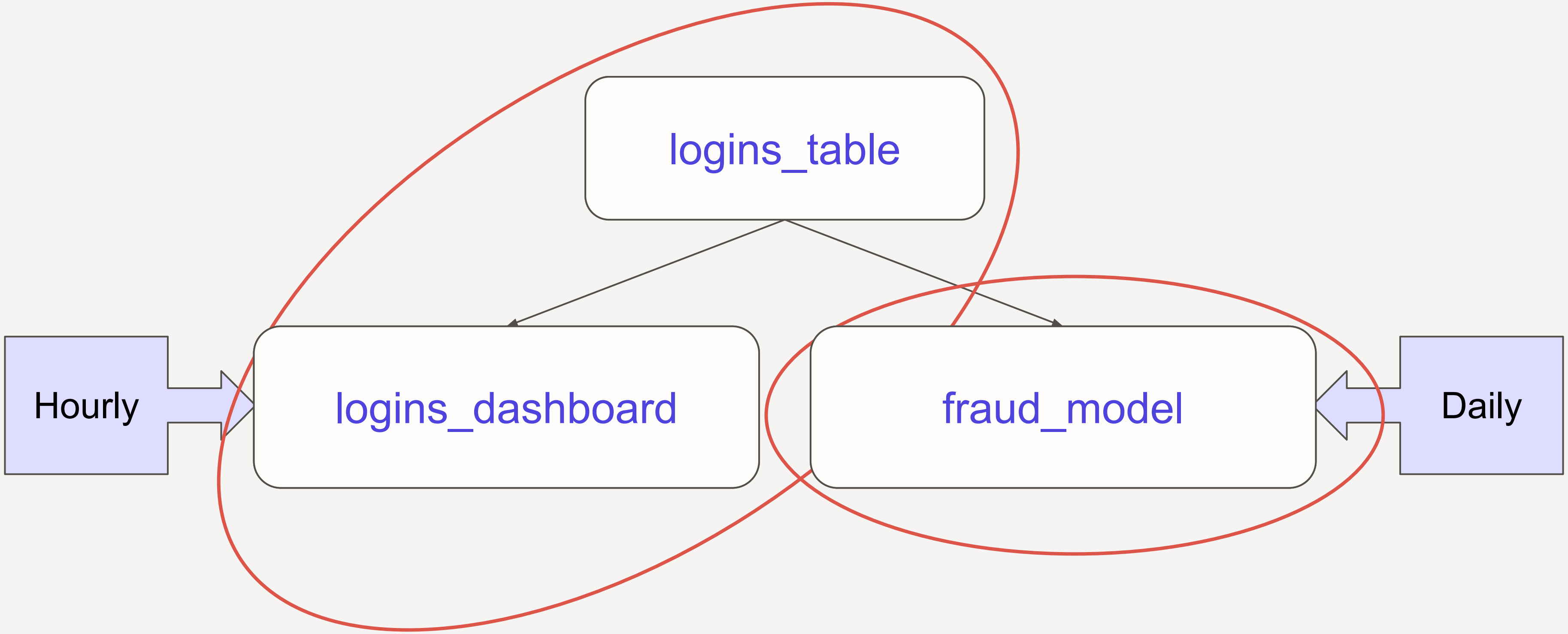
Common scenario:
different freshness requirements, same upstream data



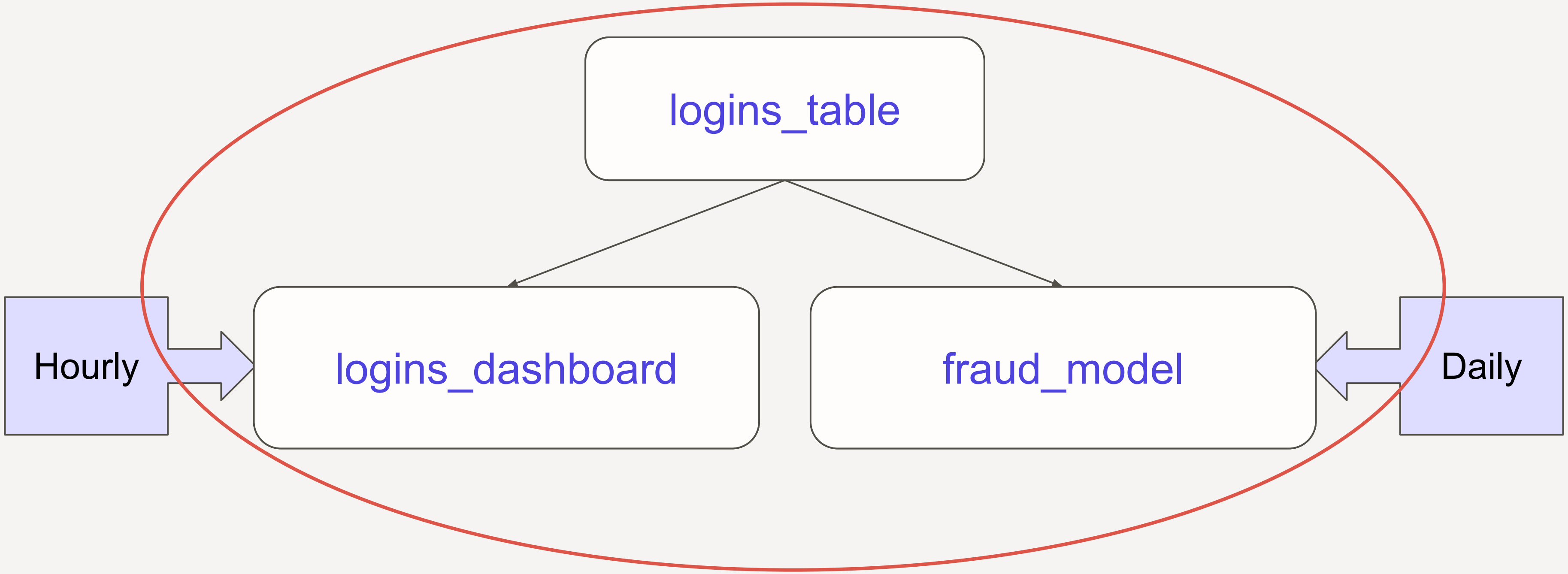
Common scenario:
different freshness requirements, same upstream data



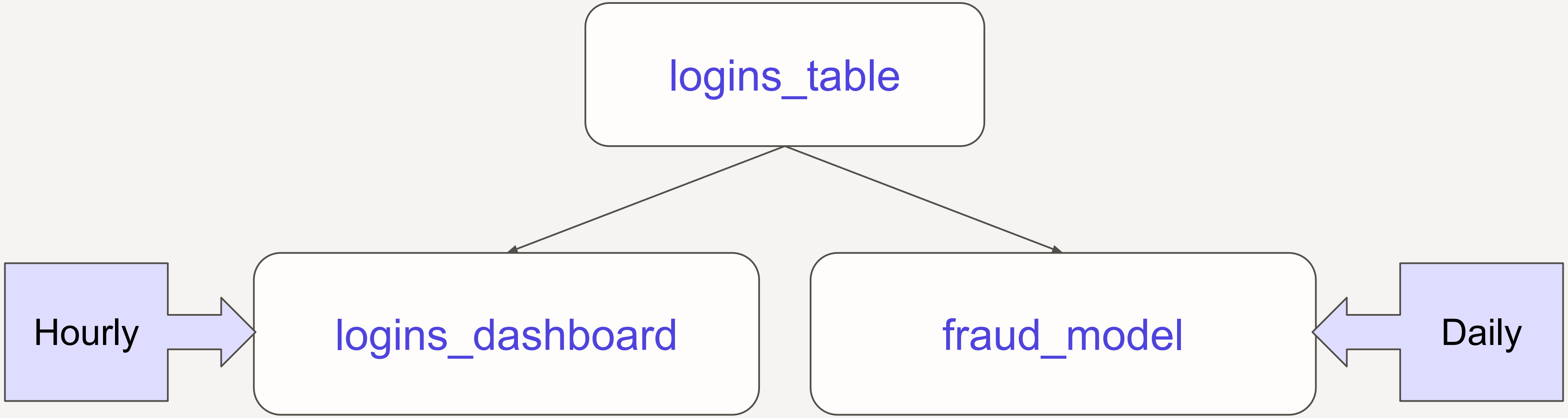
Common scenario:
different freshness requirements, same upstream data



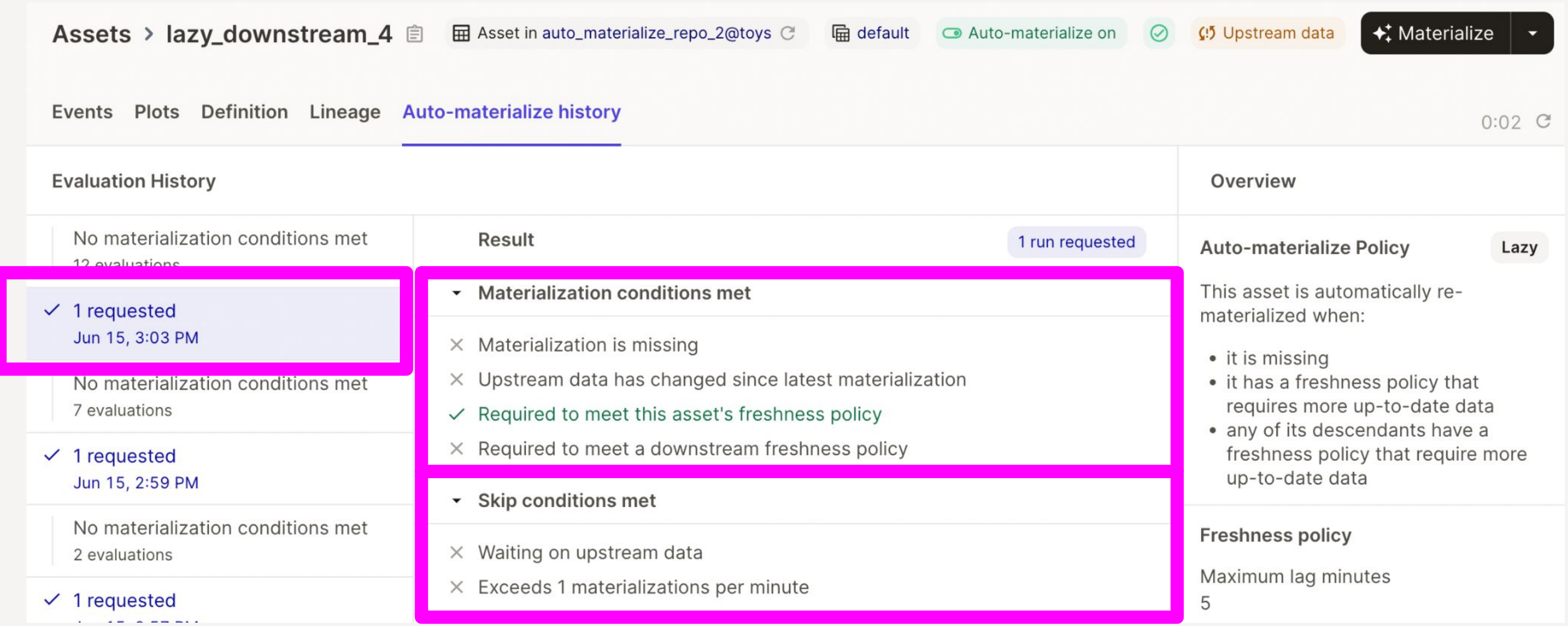
Common scenario:
different freshness requirements, same upstream data



Common scenario:
different freshness requirements, same upstream data



Asset-based orchestration: observability



Assets > lazy_downstream_4

Asset in auto_materialize_repo_2@toys · default · Auto-materialize on · Upstream data · Materialize

Events · Plots · Definition · Lineage · Auto-materialize history · 0:02

Evaluation History

No materialization conditions met
12 evaluations

✓ 1 requested
Jun 15, 3:03 PM

No materialization conditions met
7 evaluations

✓ 1 requested
Jun 15, 2:59 PM

No materialization conditions met
2 evaluations

✓ 1 requested

Result — 1 run requested

Materialization conditions met

- ✗ Materialization is missing
- ✗ Upstream data has changed since latest materialization
- ✓ Required to meet this asset's freshness policy
- ✗ Required to meet a downstream freshness policy

Skip conditions met

- ✗ Waiting on upstream data
- ✗ Exceeds 1 materializations per minute

Overview

Auto-materialize Policy — Lazy

This asset is automatically re-materialized when:

- it is missing
- it has a freshness policy that requires more up-to-date data
- any of its descendants have a freshness policy that require more up-to-date data

Freshness policy

Maximum lag minutes
5

To sum up...

- Data pipeline = graph of data assets connected by computations
- Workflows are not an adequate scheduling abstraction
- Asset-based orchestration
  - Express intentions more clearly
  - Avoid redundant computations
  - Debug scheduling decisions

Thank you

Sandy Ryza
@s_ryz

elementl

dagster