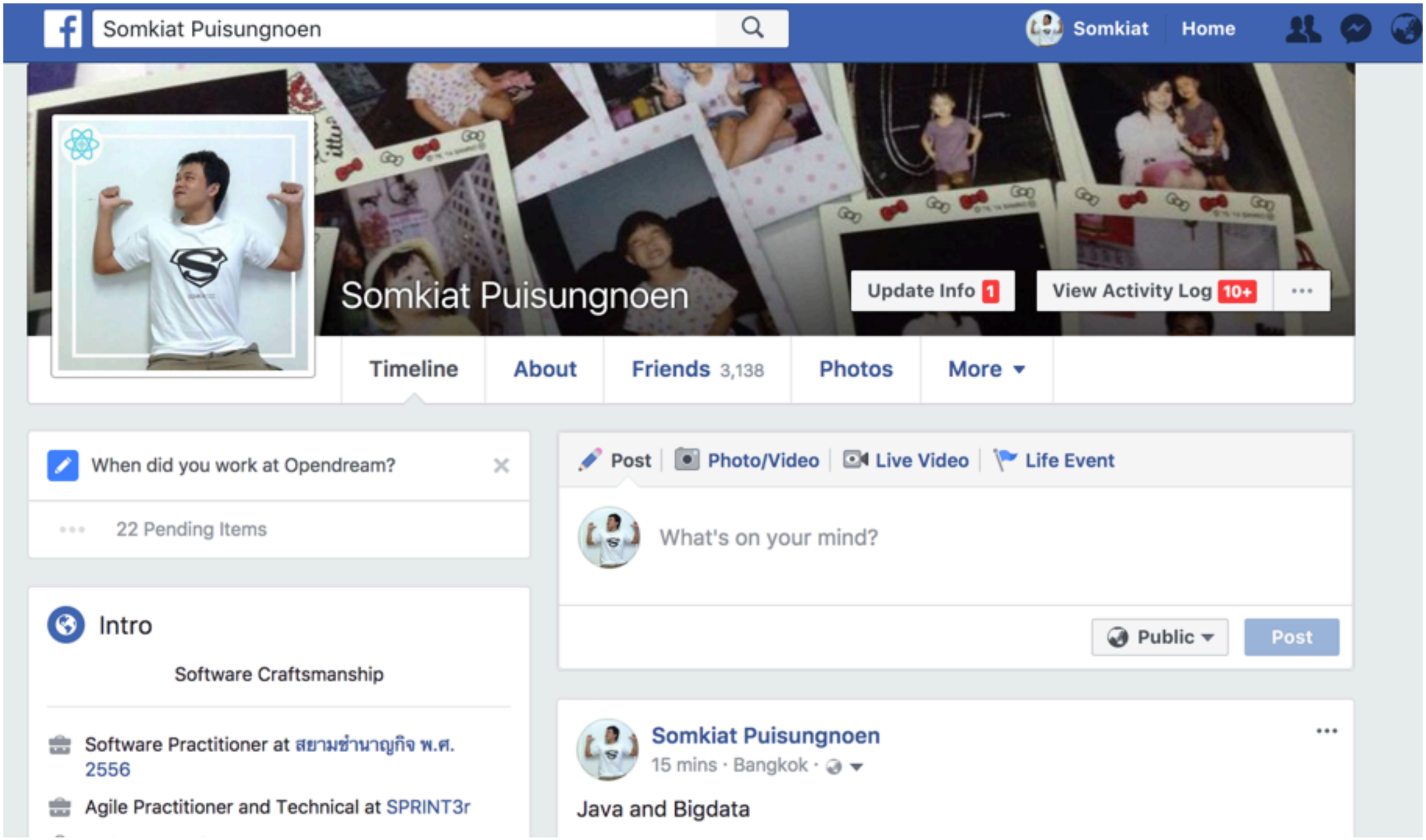
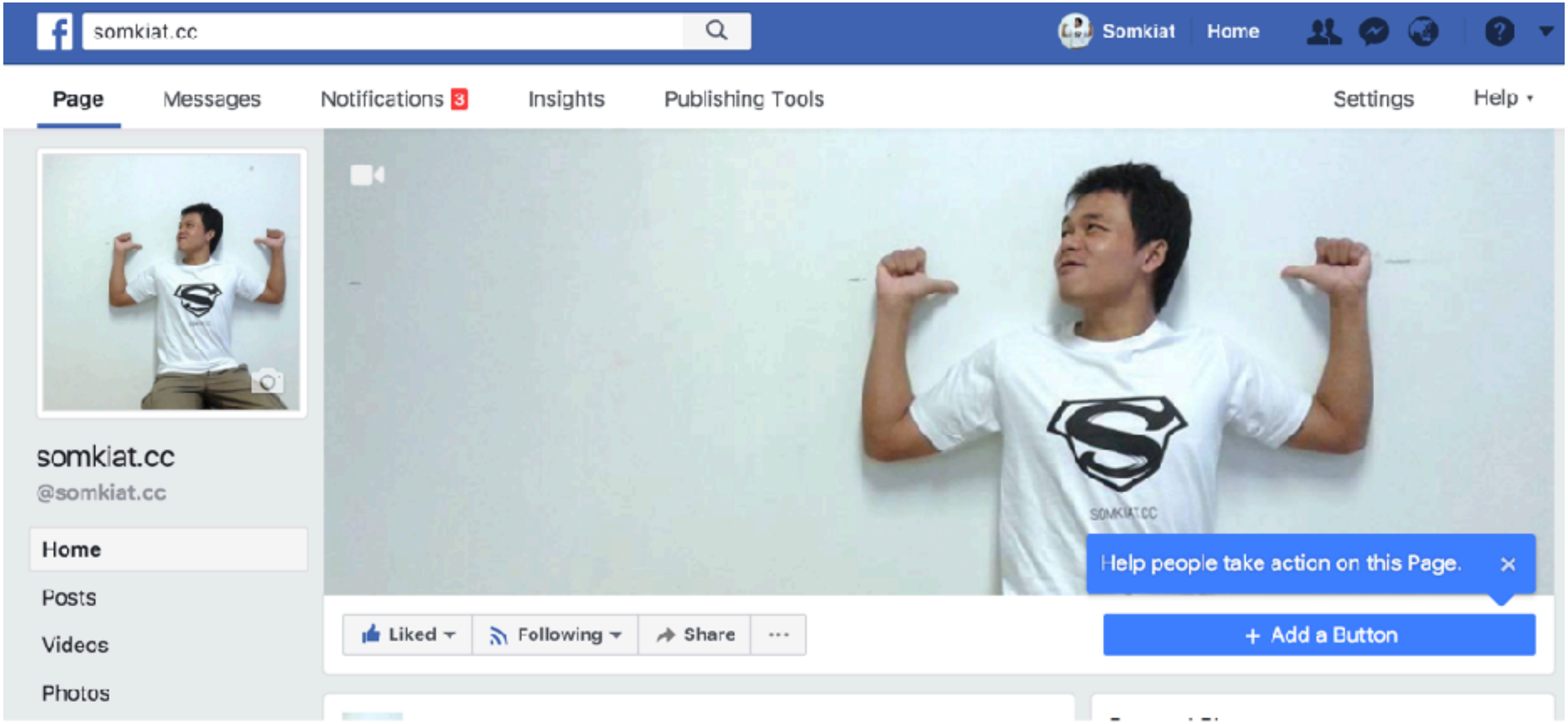
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Apache Airflow workshop





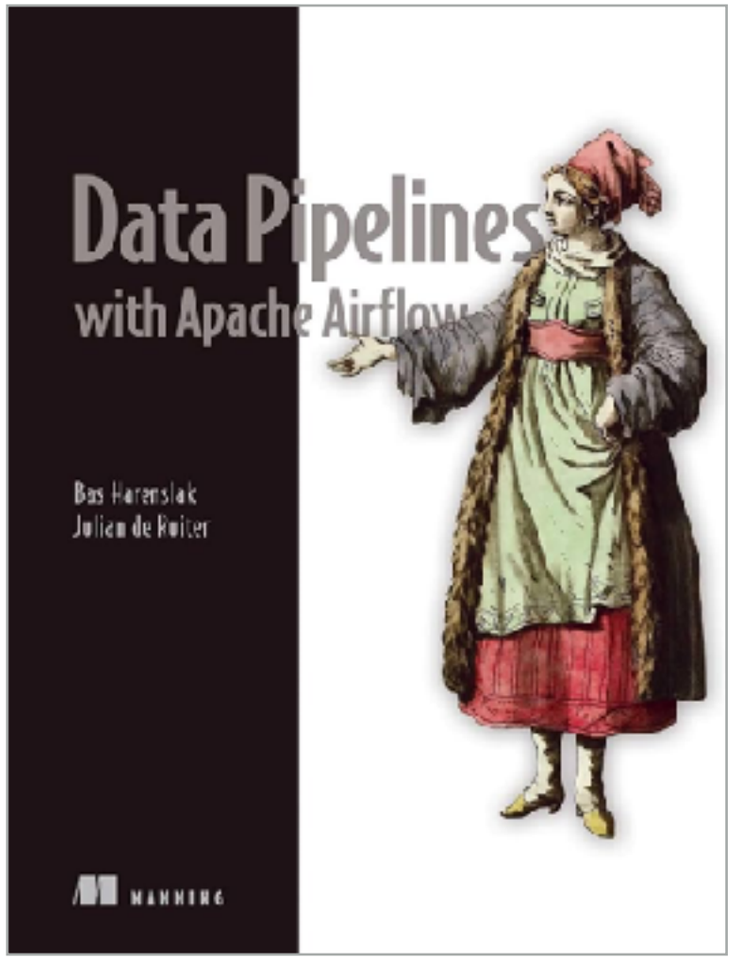
https://github.com/up1/workshop-apache-airflow

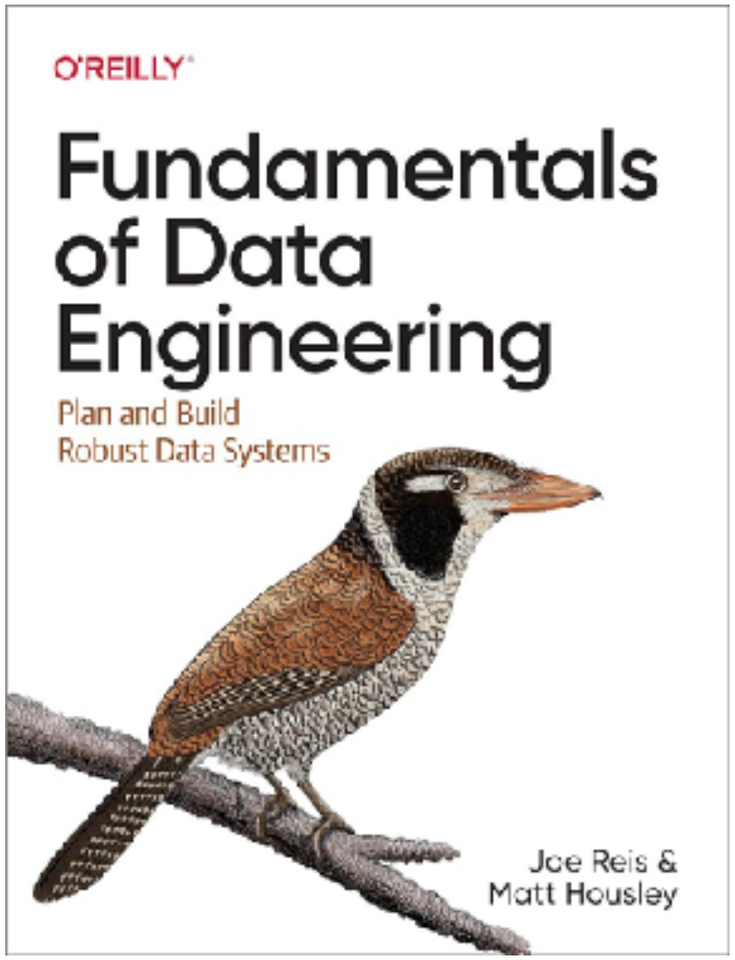
Topics

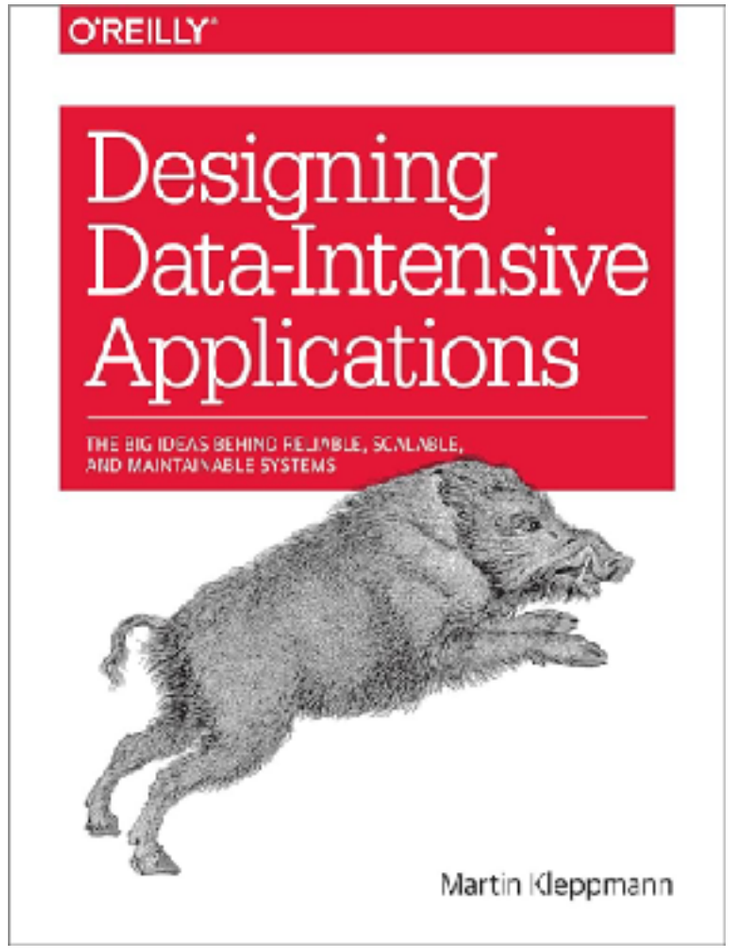
Workflow and Data pipeline
History
Why ?
Introduction of Airflow
Architecture
Design your data pipeline
Workshop

AirBnB + Work**flow**

Books







Data pipeline workflow

Data pipeline

Data pipelines generally consist of several **tasks** or actions that need to be executed to achieve the desired result

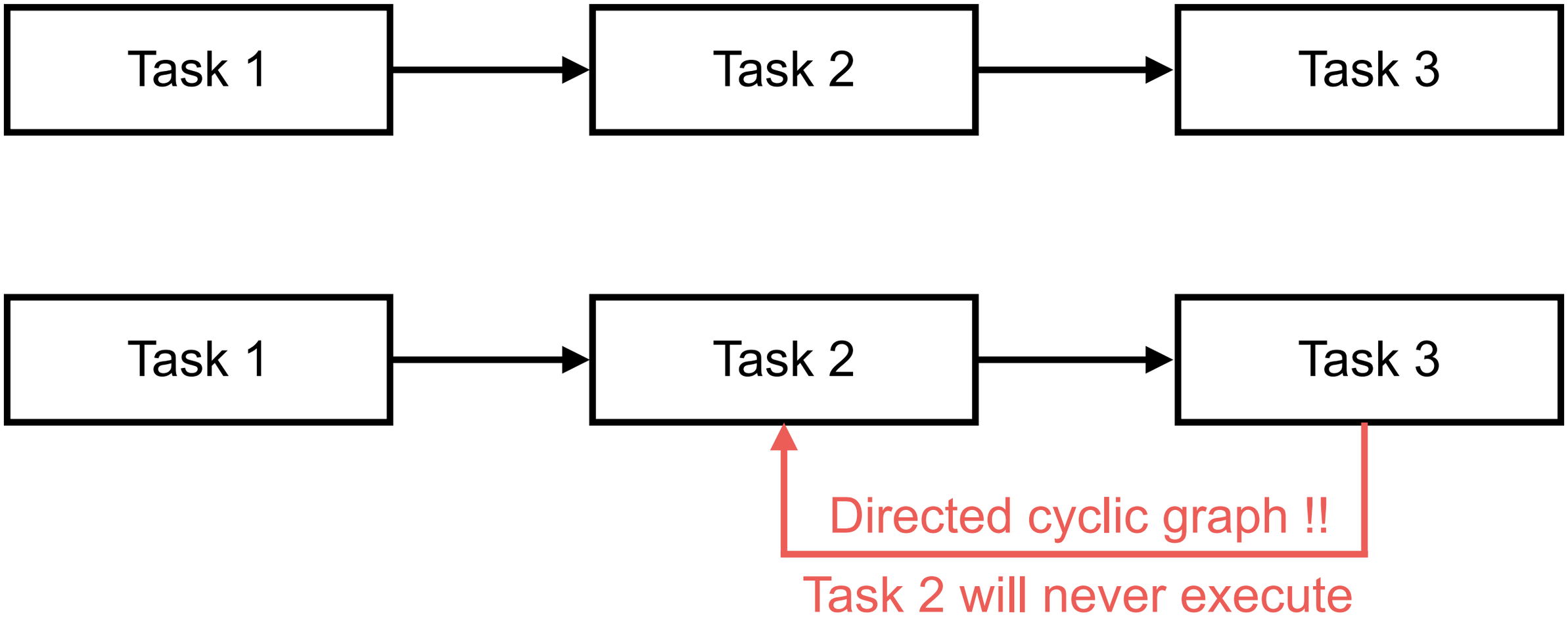
Data pipeline as graph

Represent data pipeline as graph
Easy to understand

Tasks** are represented as nodes in **graph

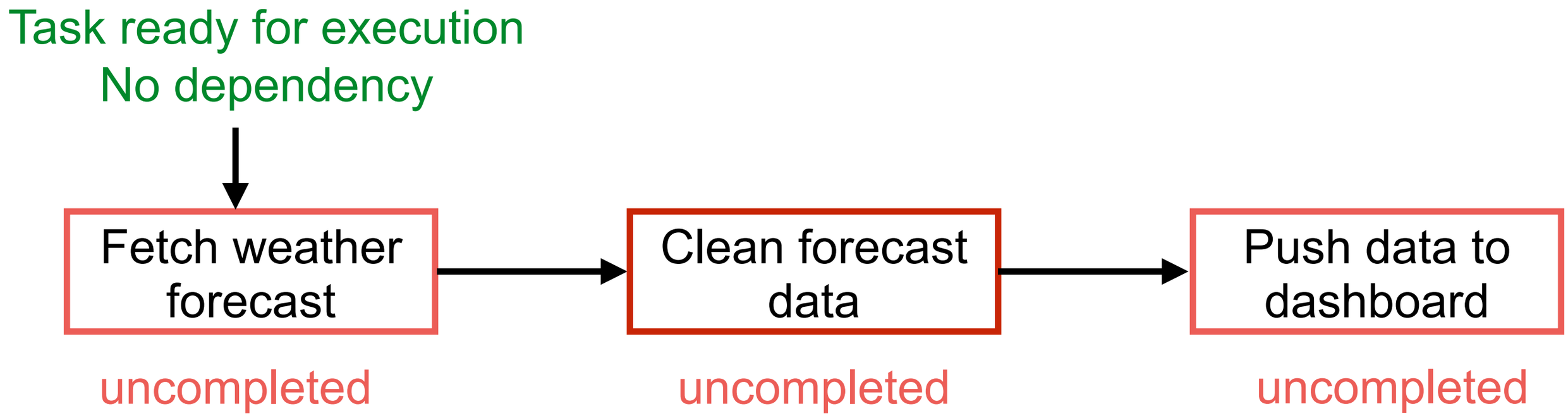
DAGs (Directed Acyclic Graph)

The graph contains directed edges and does not contain any loops or cycles (acyclic)



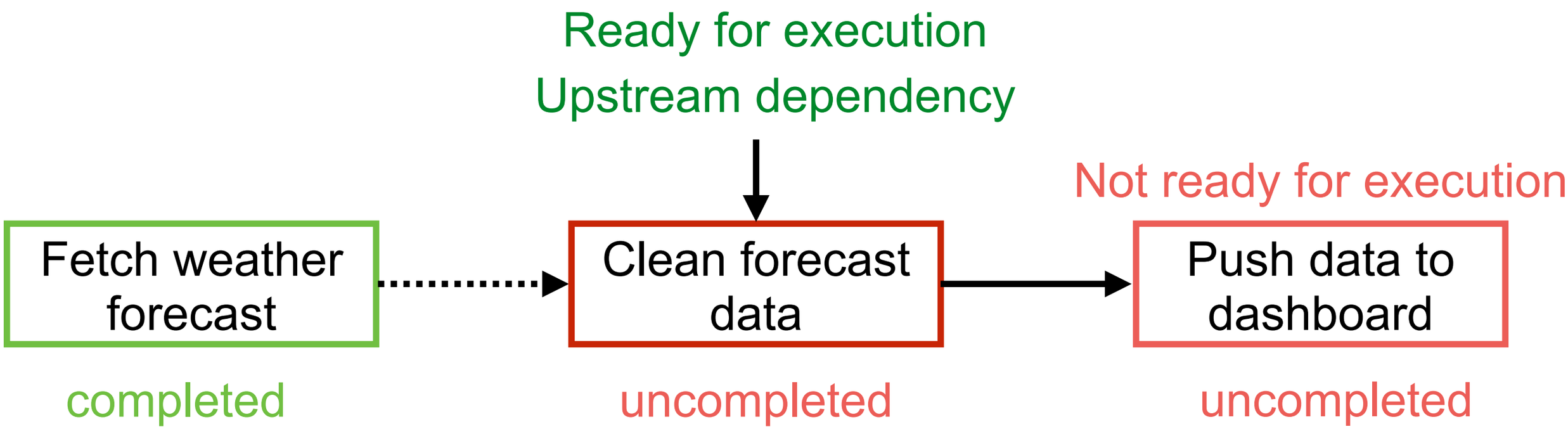
Execute a pipeline graph

Step 1 :: Open uncompleted task in graph



Execute a pipeline graph

Step 2 :: Execute a upstream dependency



Execute a pipeline graph

Step 3 :: Execute a upstream dependency

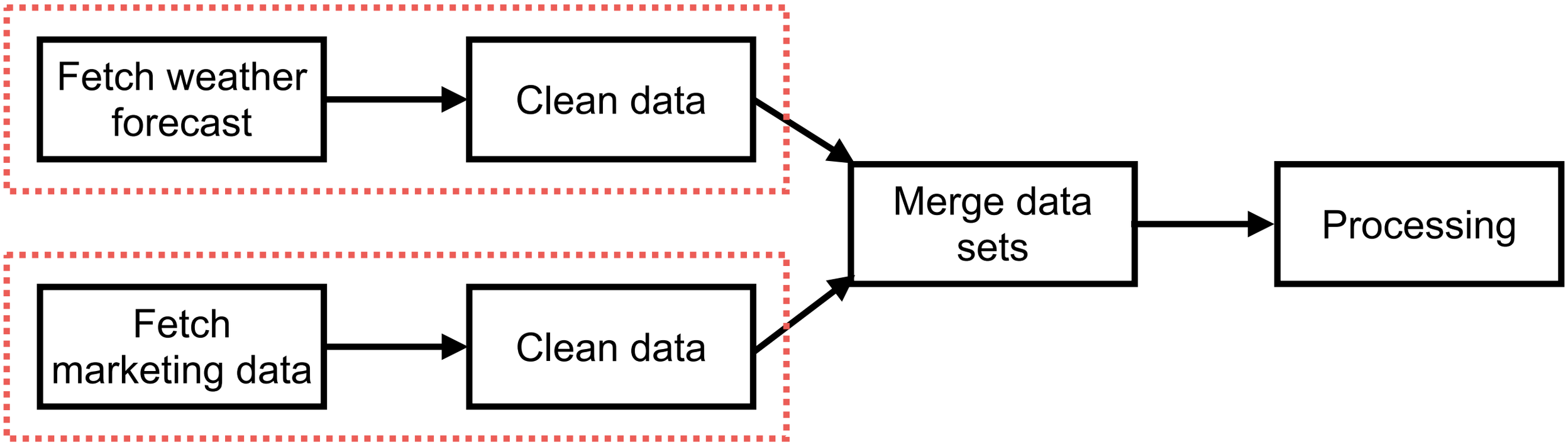
Execute a pipeline graph

Step 4 :: End state

More use cases

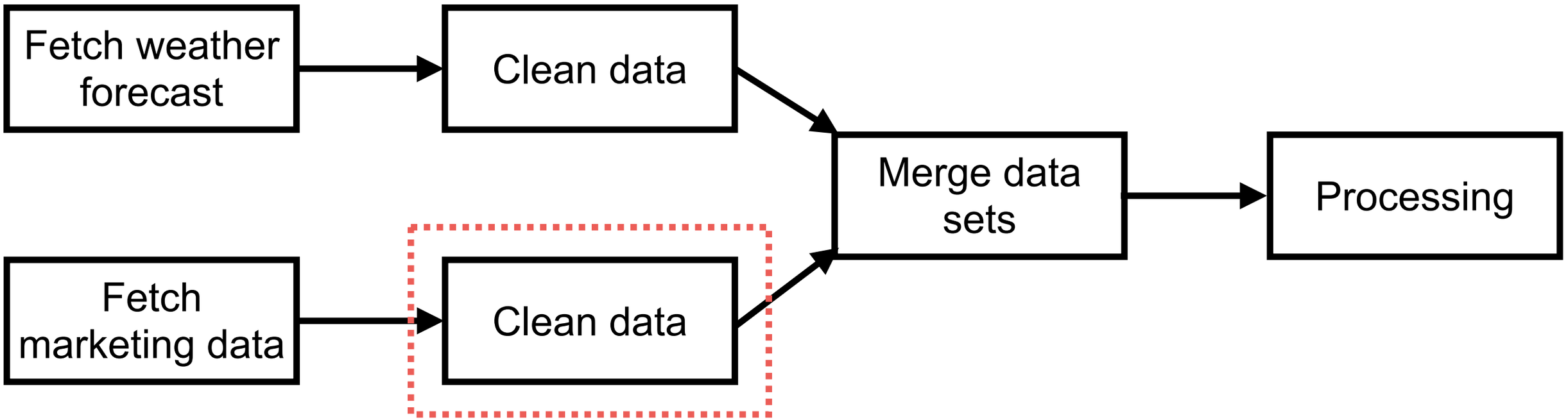
Pipeline graph

Execute each task in parallel process



Pipeline graph

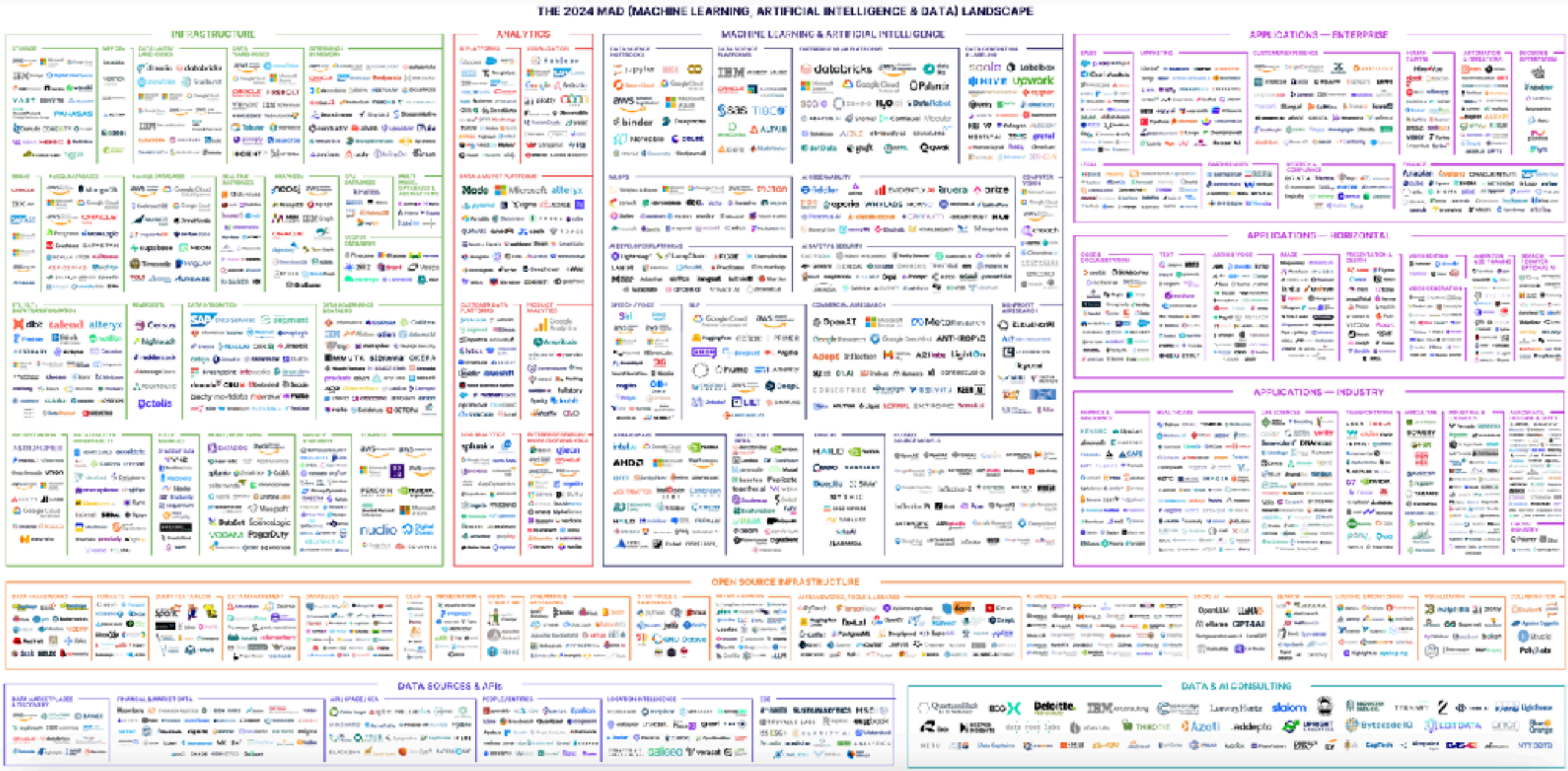
When task failed ?



Rerun only this task

Tools

https://mattturck.com/mad2024/



Apache Airflow !!

Workflow as a Code

What Airflow ?

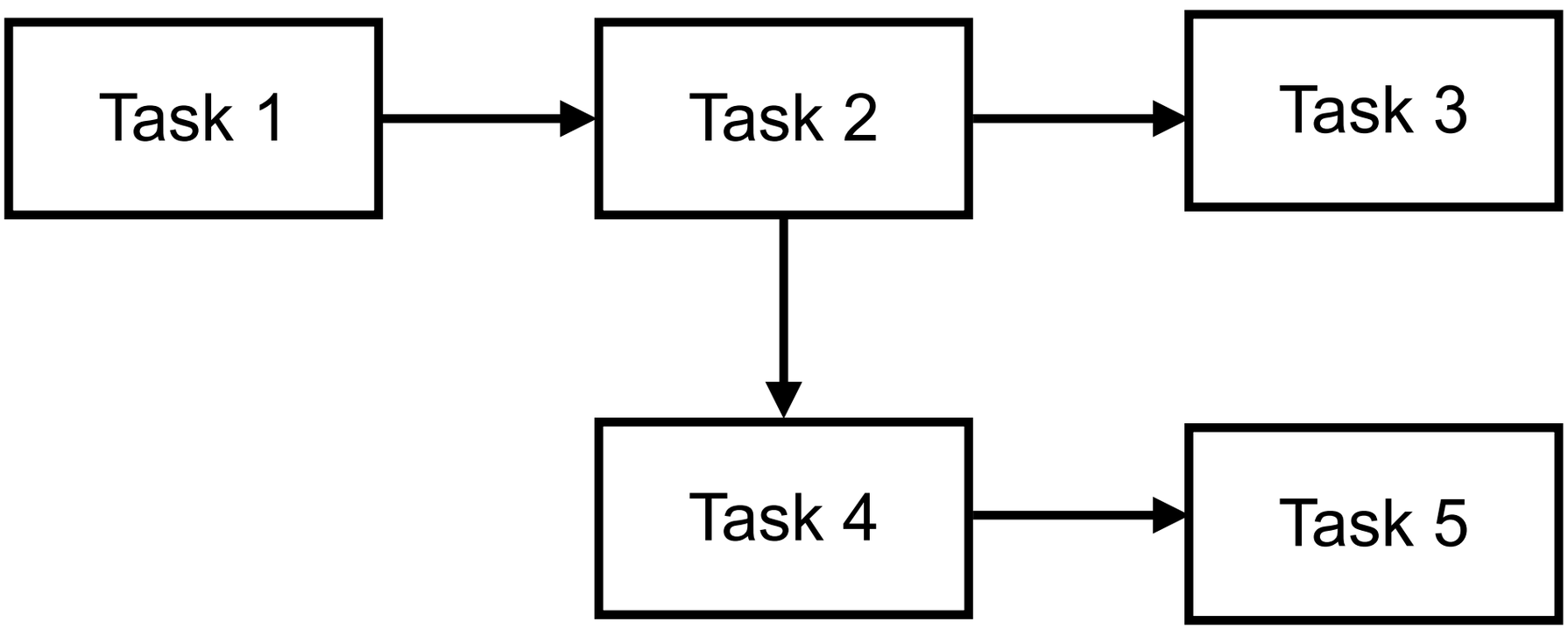
Platform for programmatically authoring, scheduling, and monitoring **workflows.**

It is especially useful for creating and orchestrating complex **data pipelines.**

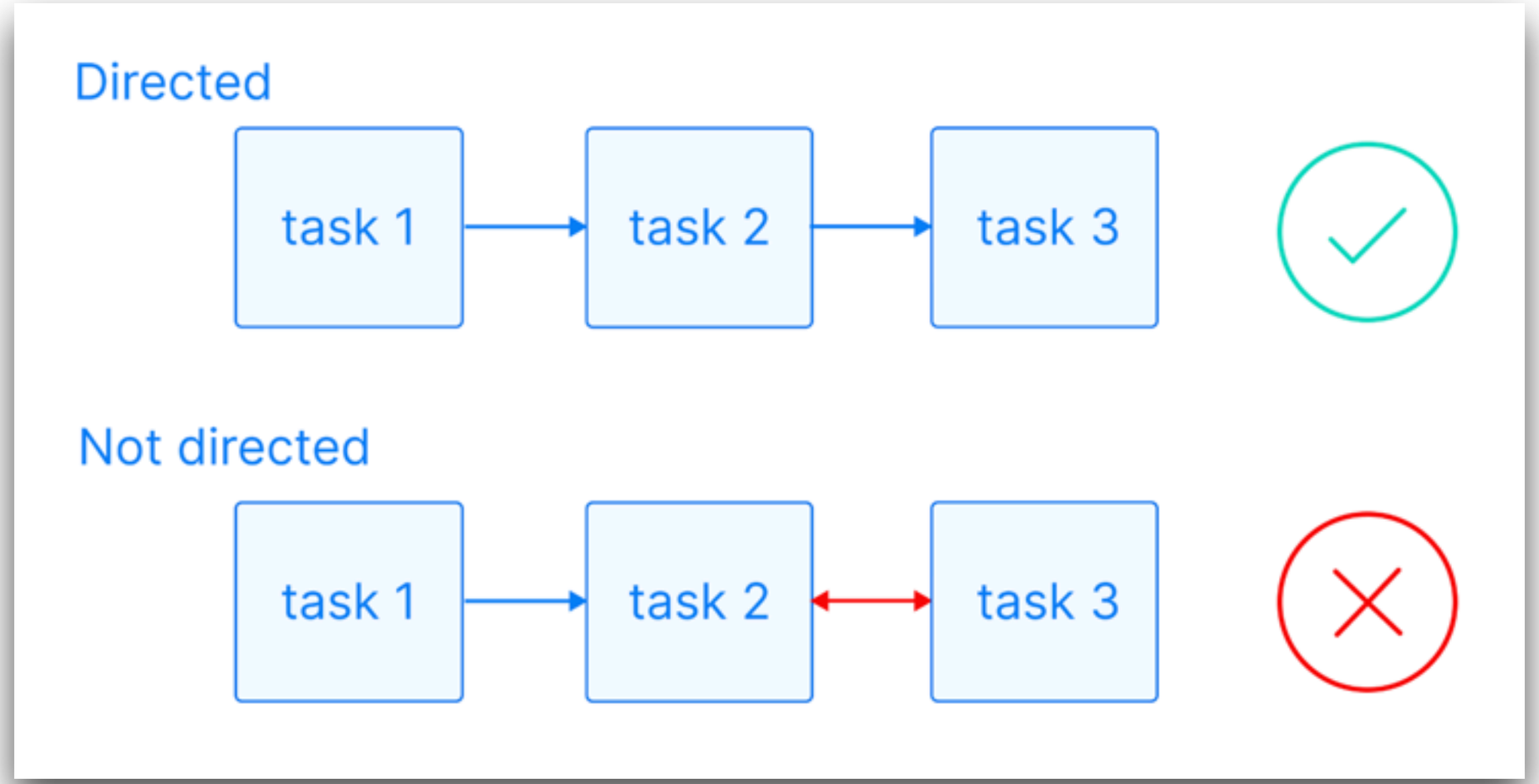
ETL	Business operations	MLOps	Manage infrastructure

What Airflow ?

Starts in Airbnb 2014
Manage complex **workflows**
Sequence of tasks (DAG)
Create workflow with **Python**

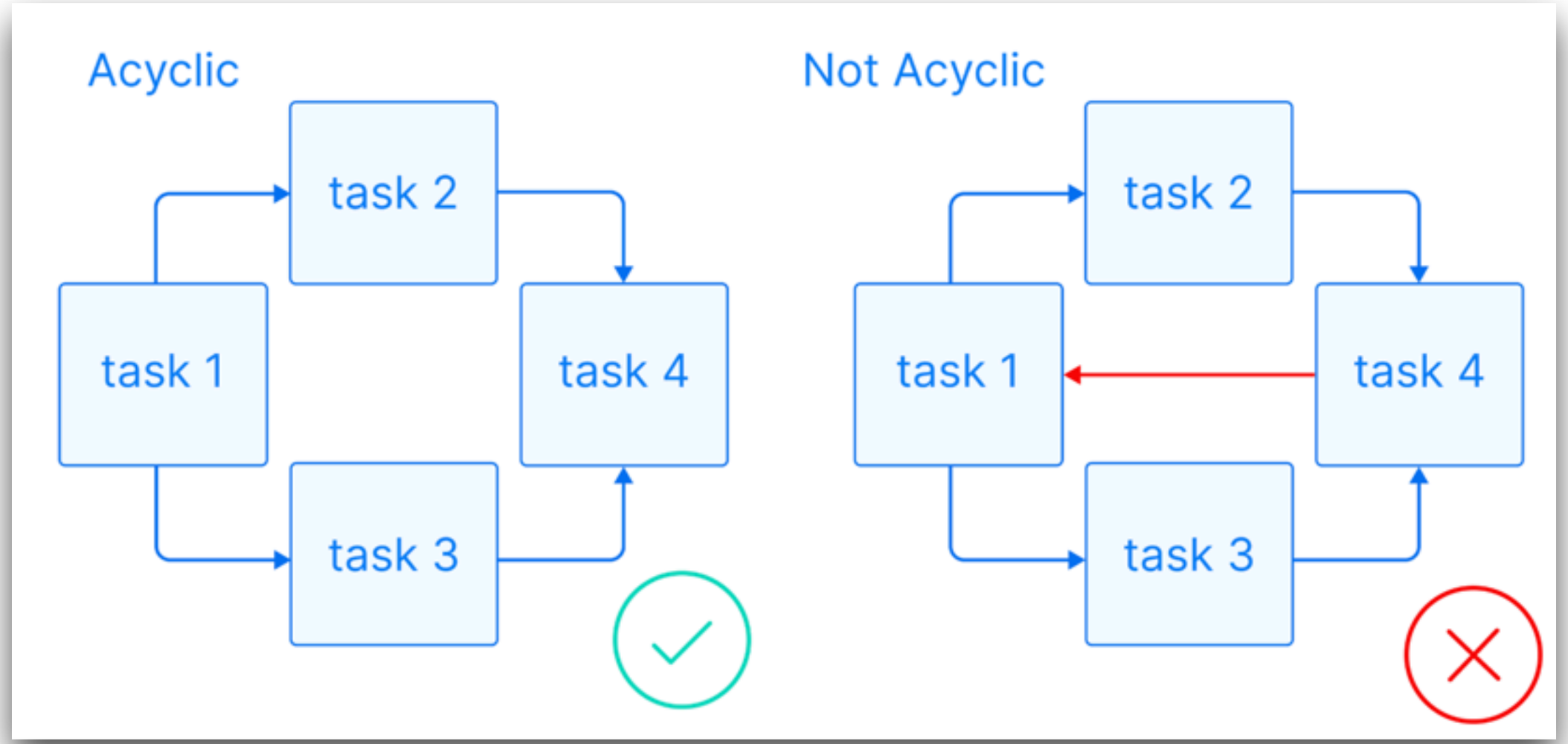


DAG (Directed Acyclic Graph)



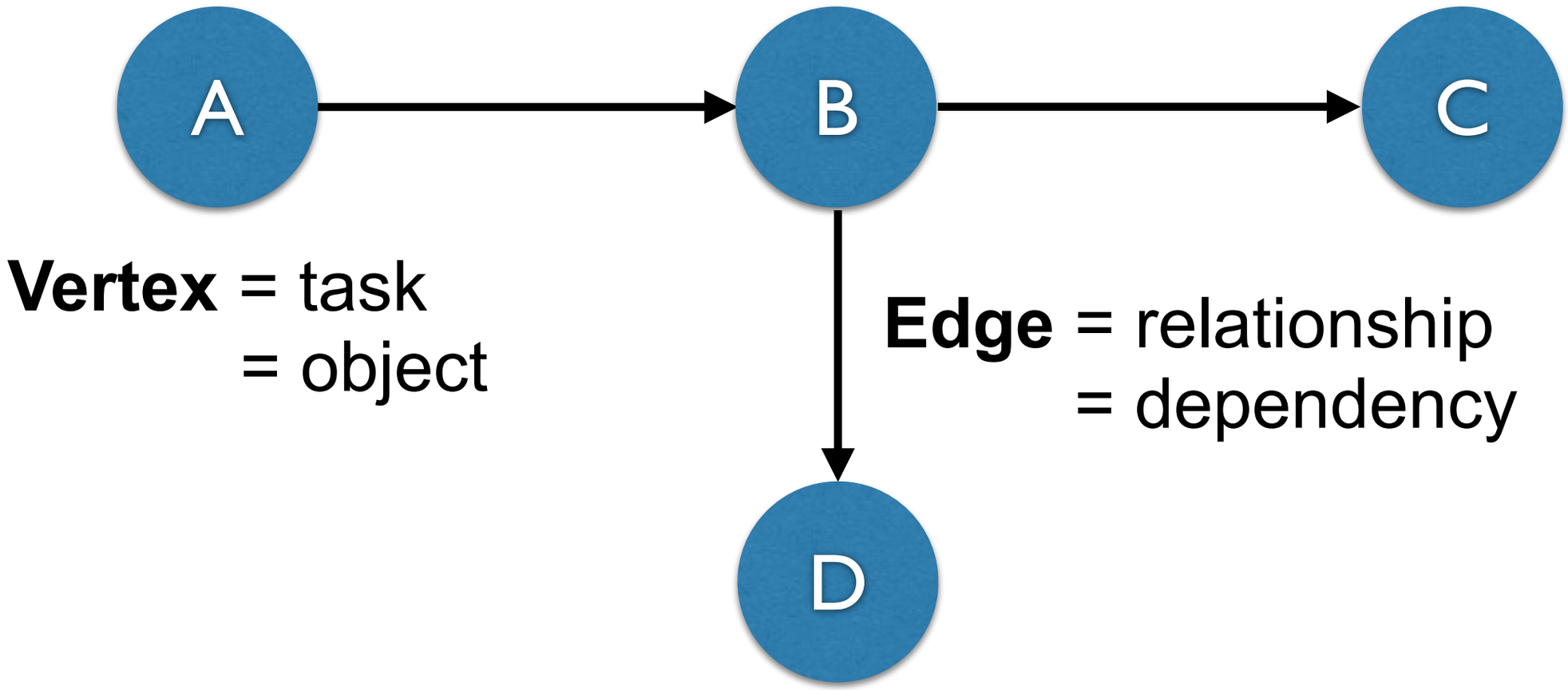
https://www.astronomer.io/docs/learn/dags

DAG (Directed Acyclic Graph)



https://www.astronomer.io/docs/learn/dags

DAG (Directed Acyclic Graph)

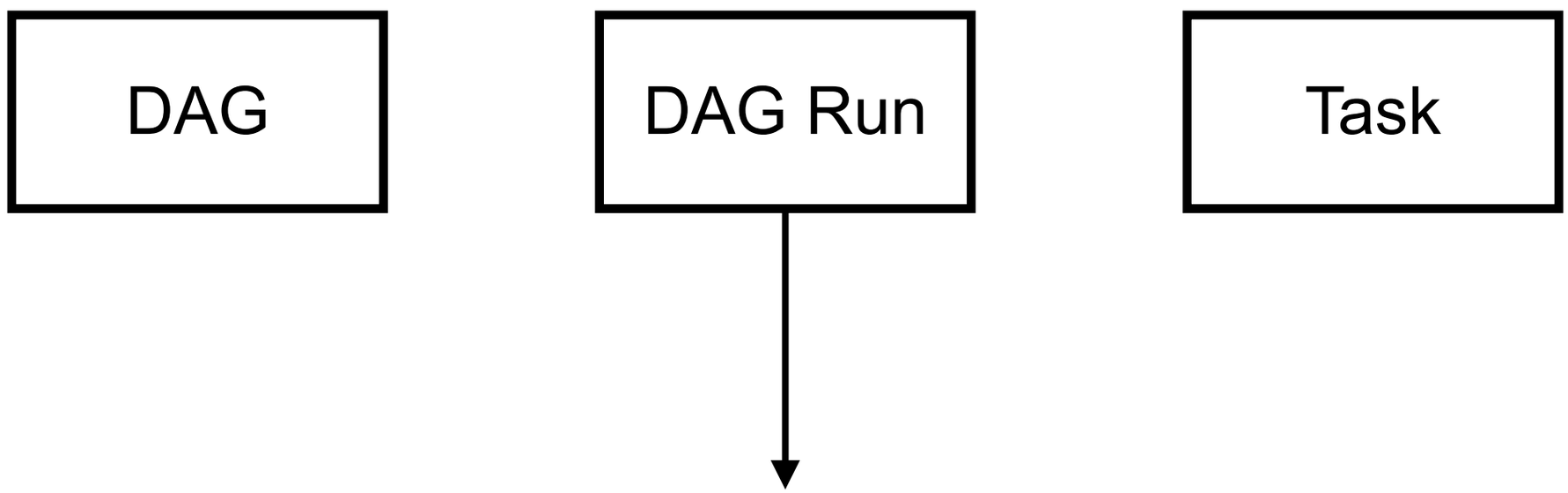


Vertex = task
= object

Edge = relationship
= dependency

Directional = **Directed**
No loops = **Acyclic**

Basic knowledges

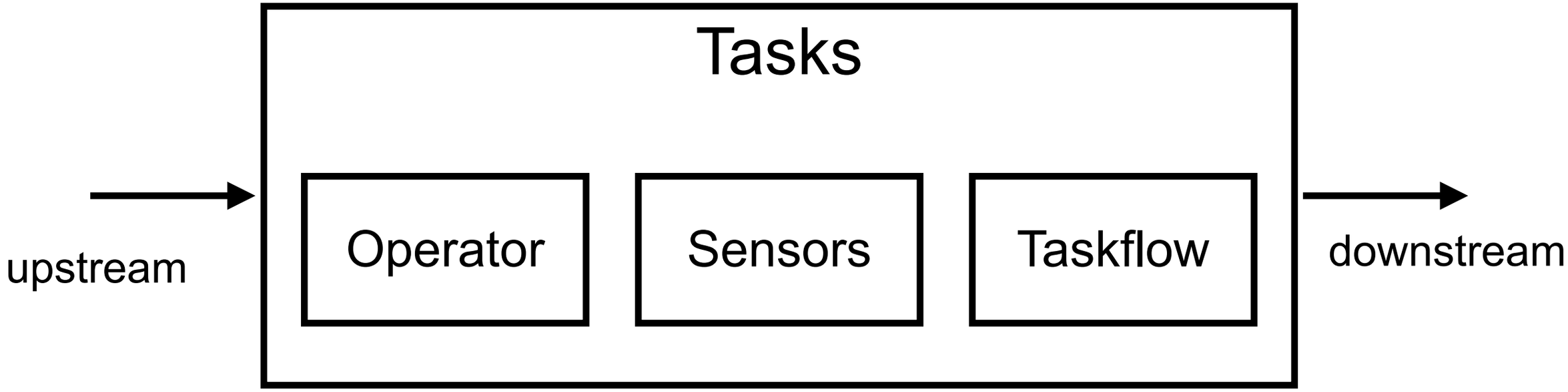


The execution of a DAG at a specific **point in time**

Manual | Scheduled | Dataset trigger | Backfill

Tasks

Unit of work or execution in Airflow
Represent as node in DAG
Isolated form each other by default
Task can have relationship



https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/tasks.html

When to not use ?

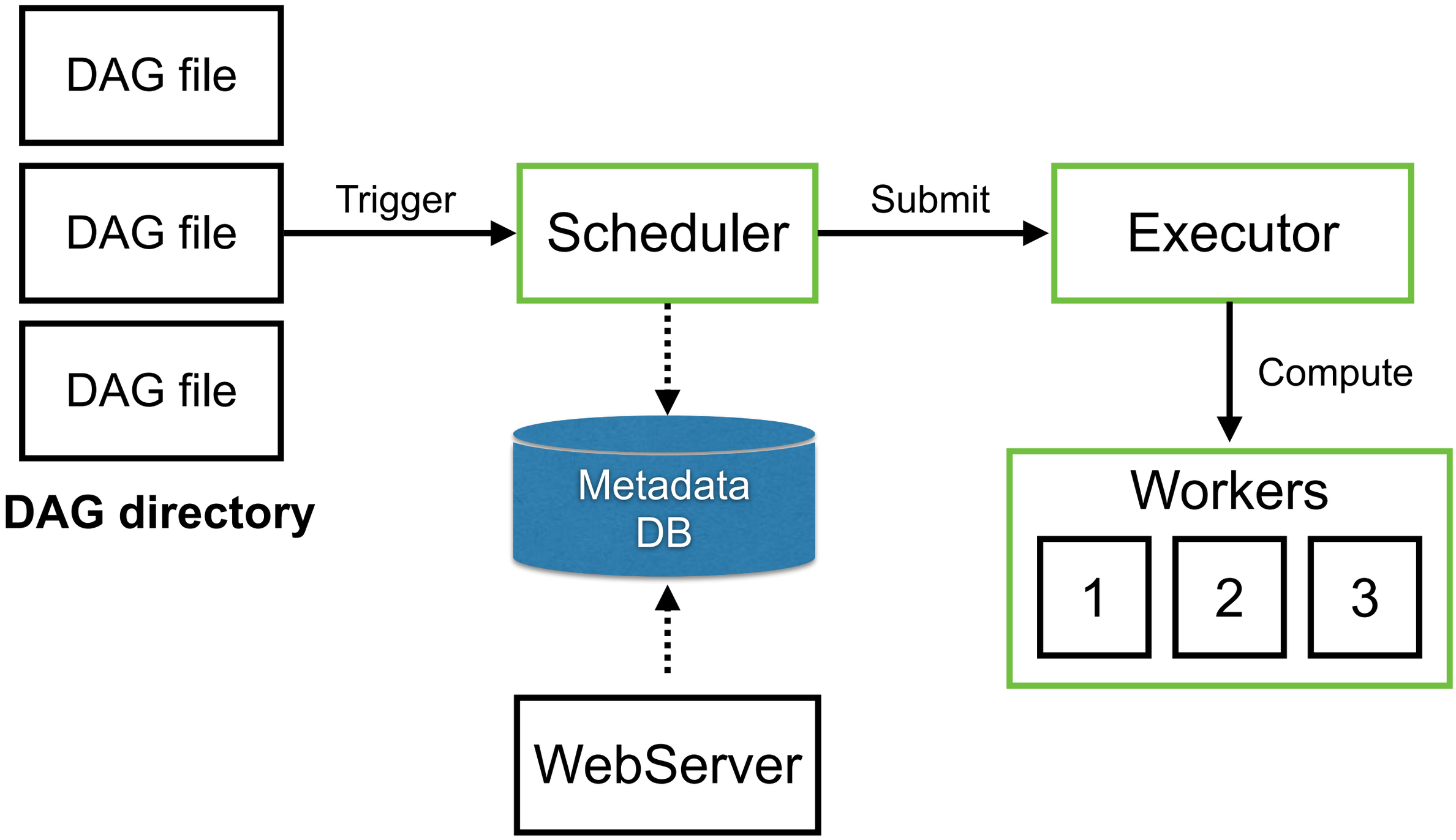
Handling streaming pipeline
Highly dynamic pipeline
Less experience with Python

Design for recurring and batch-oriented tasks

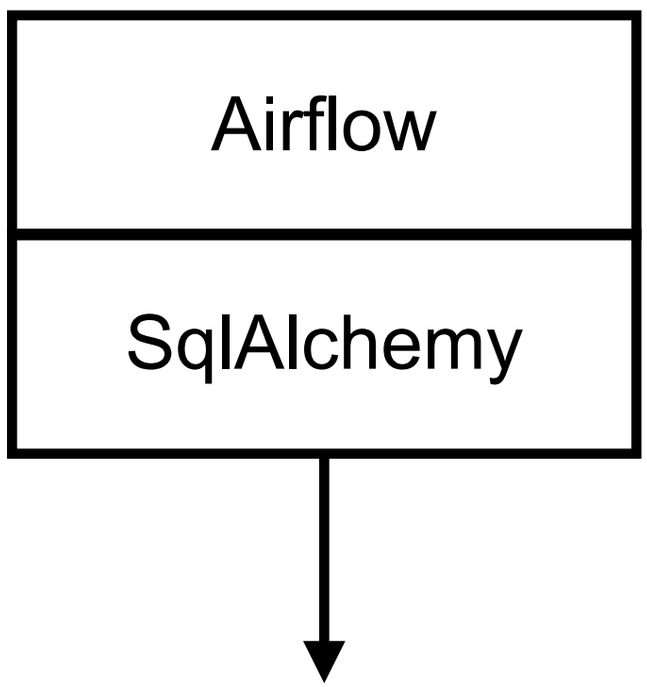
Architecture

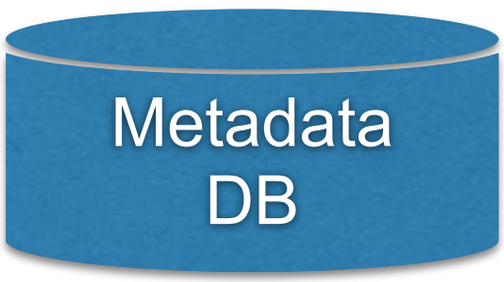
https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html

Airflow components



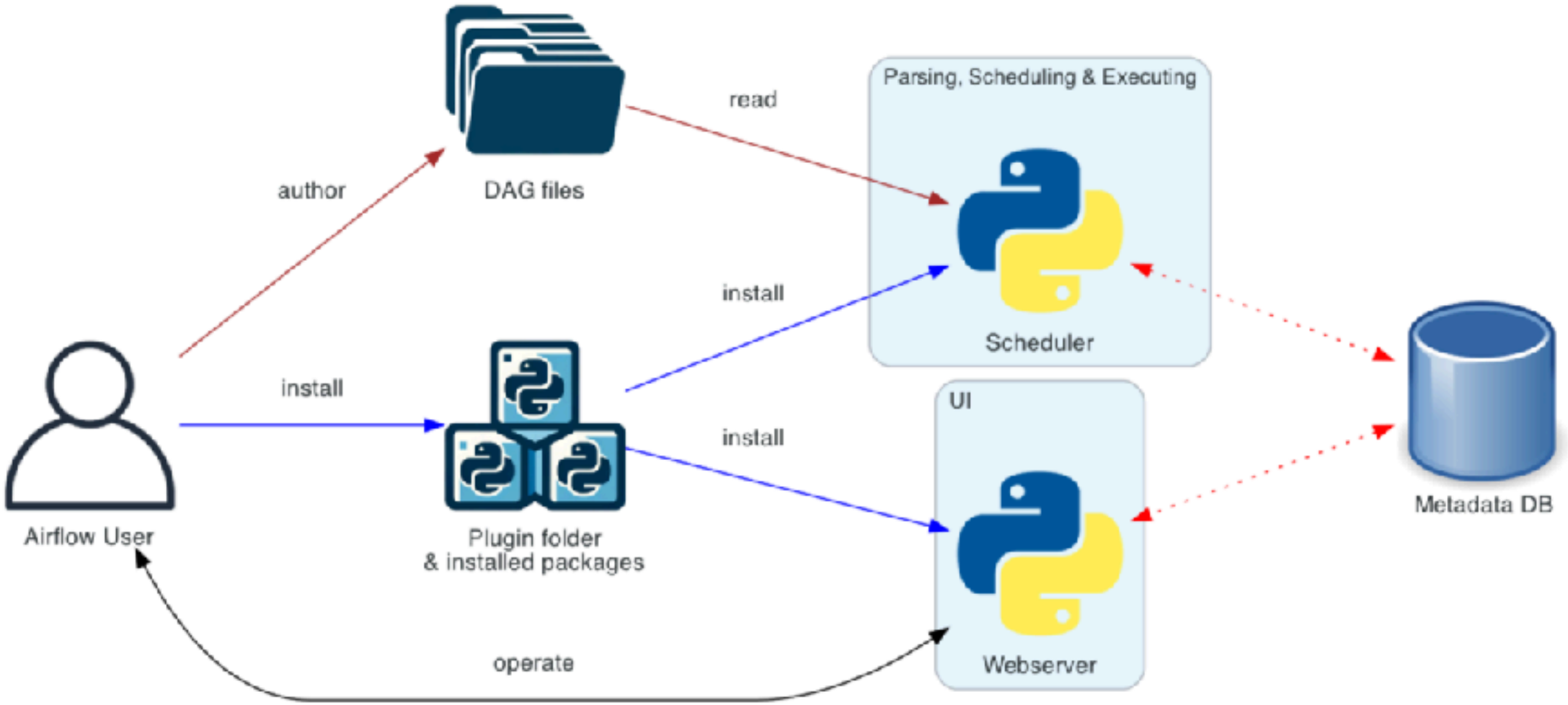
Database



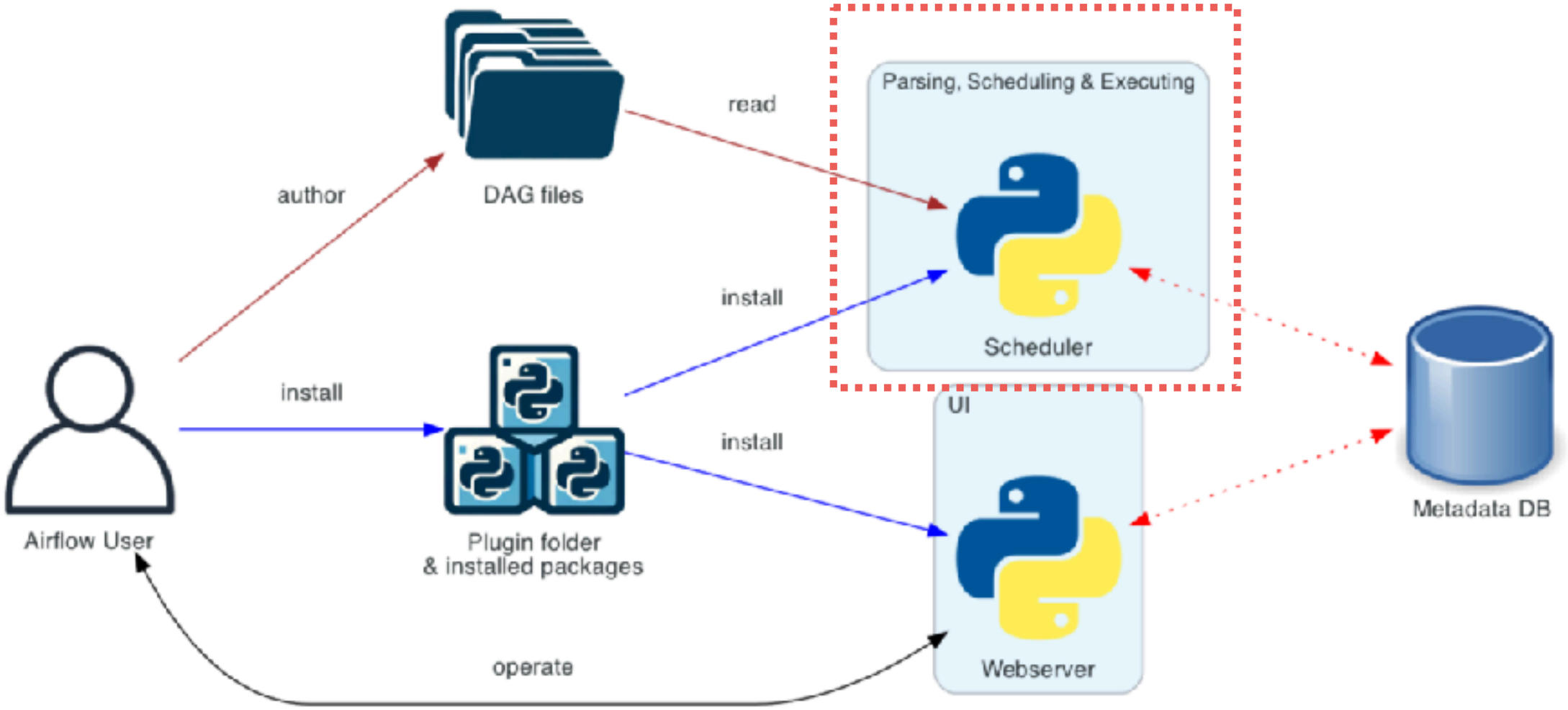


https://airflow.apache.org/docs/apache-airflow/stable/howto/set-up-database.html

Simple Airflow Deployment



Schedule and Executor



Types of Executor

Local	Remote
Sequential (default)	Celery
Local	Batch
	Kubernetes

Airflow 2.10 support multi-executor

https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/executor/index.html

Sequential executor

Default
Run within scheduler
One task instance at a time
SQLite is recommended
Easy to setup and good for testing

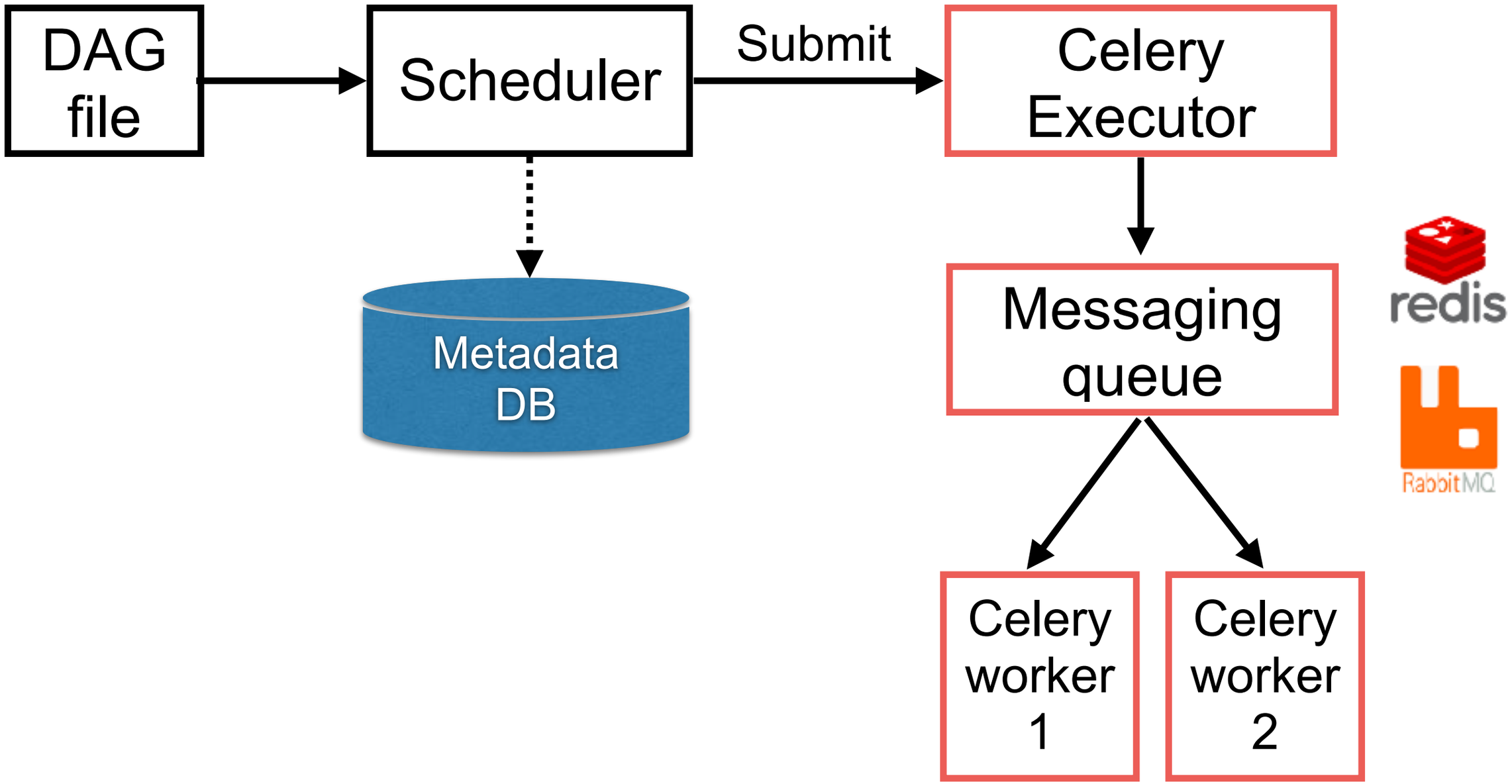
Not scalable, slow, single point of failure

Local executor

Run within scheduler
Multiple task instance at a time
Easy to setup
Good for small workflow

Not scalable, slow, single point of failure

Celery executor

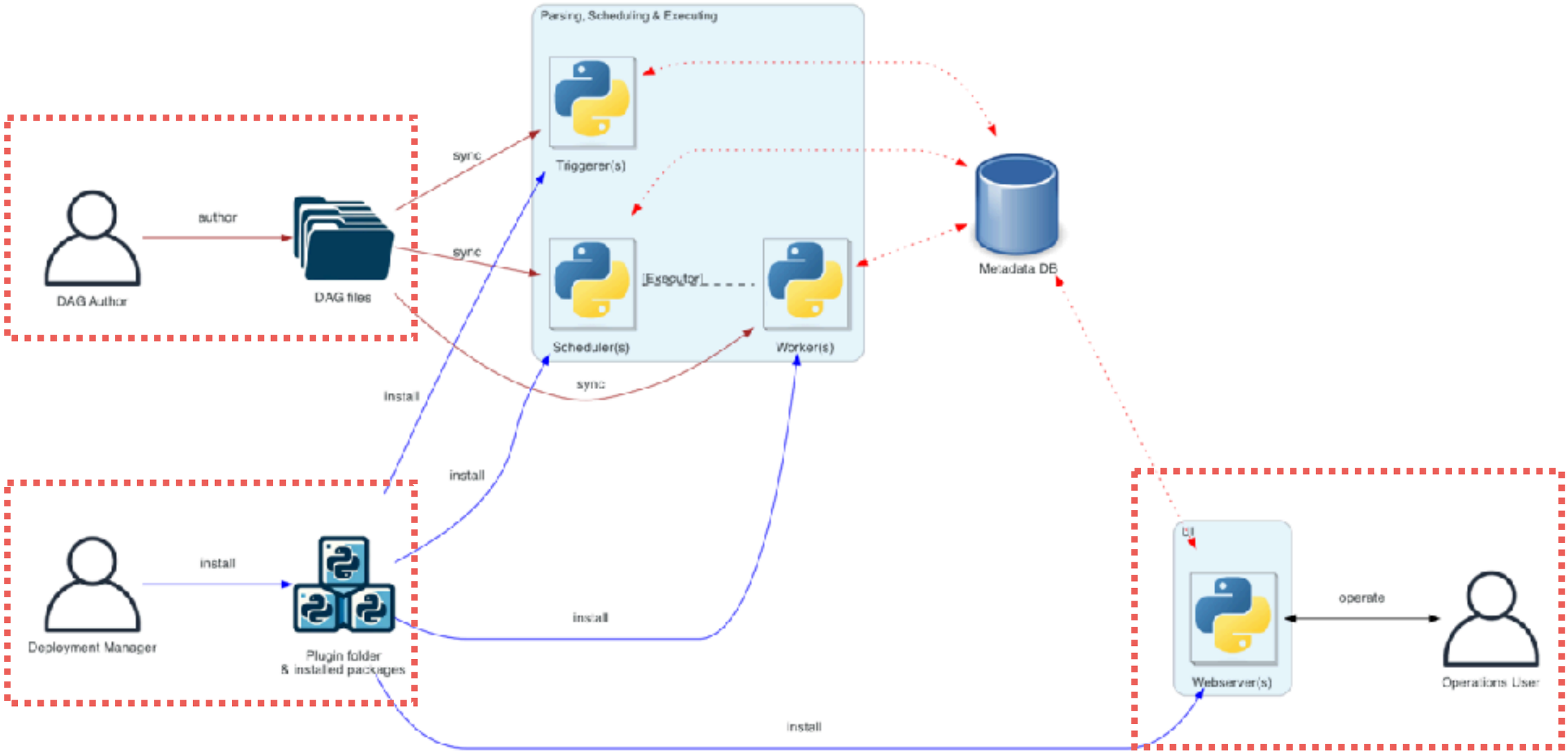


Celery executor

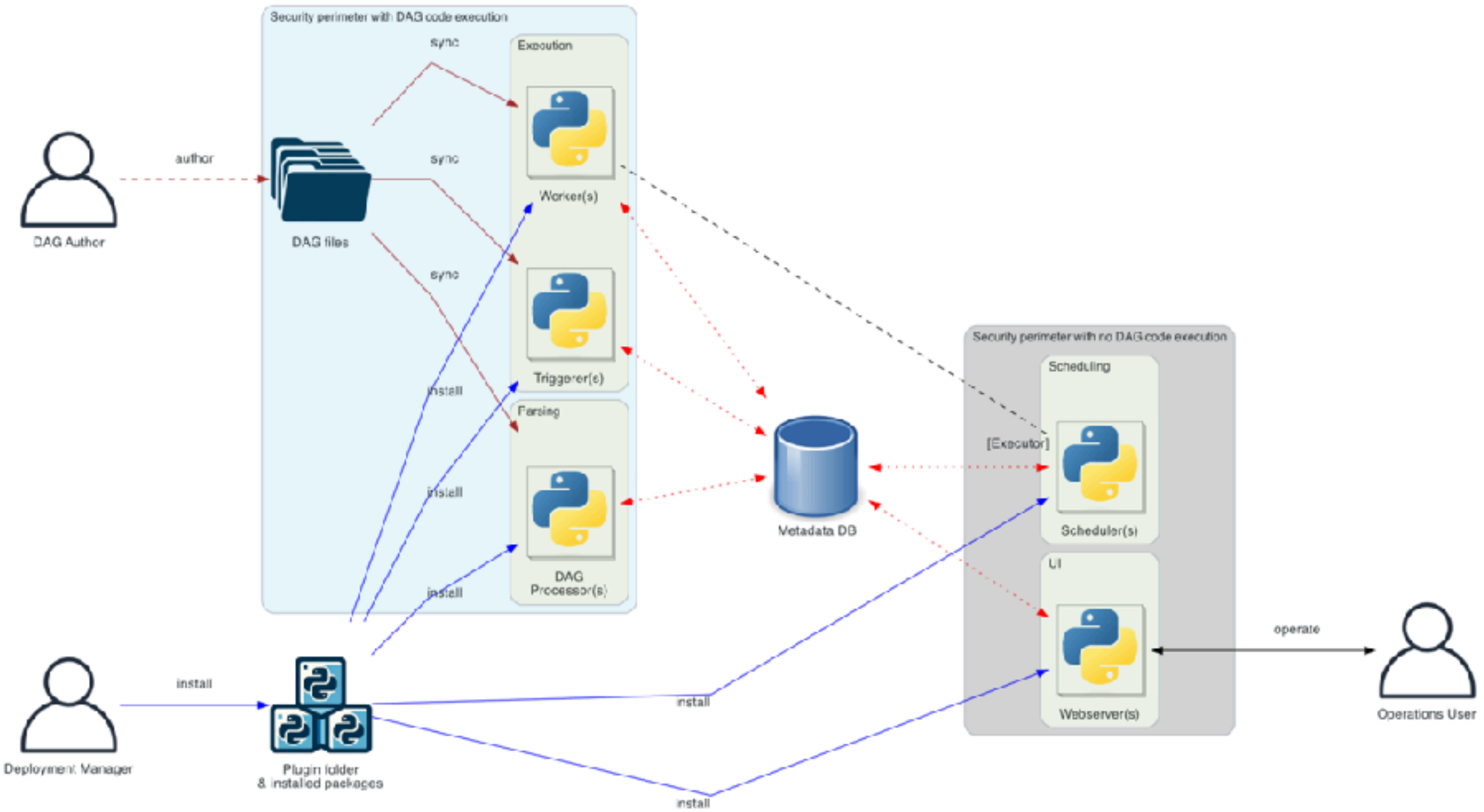
Run tasks on dedicated machine
Distributed task queue
Production ready
Horizontal scale

Take time to setup, use more resources

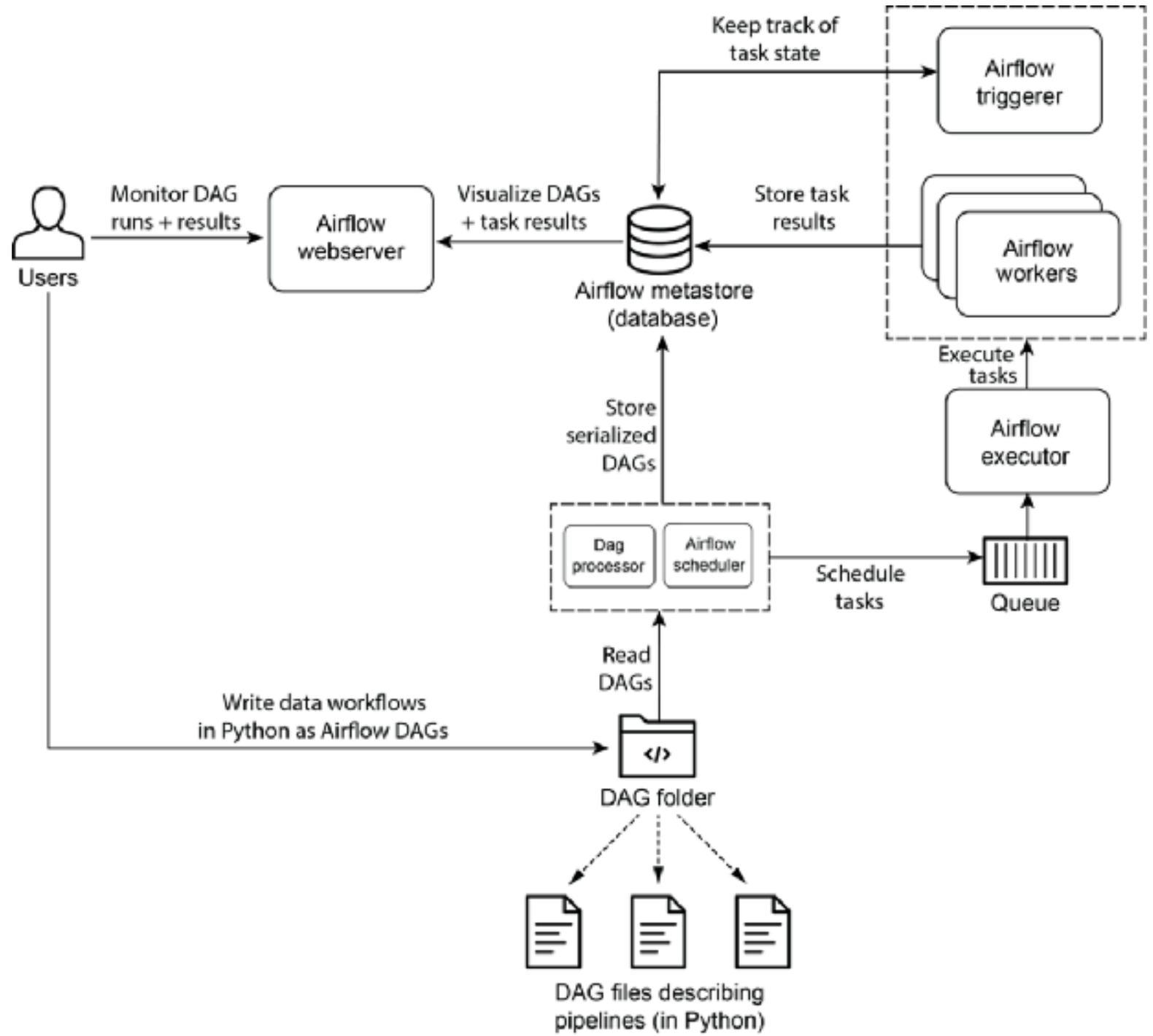
Distributed Airflow Deployment



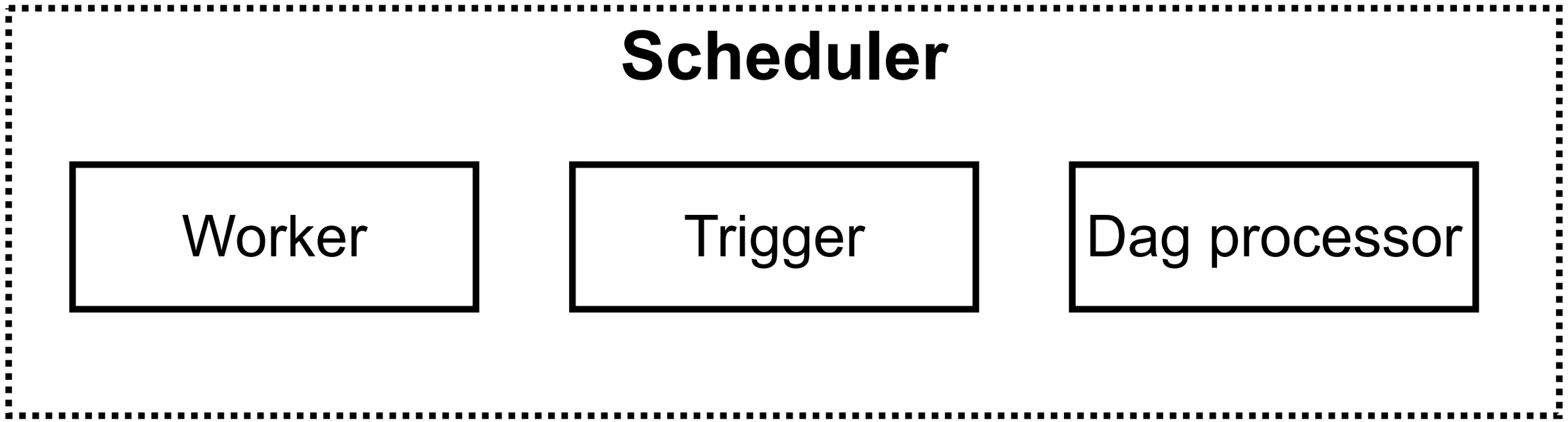
Separate DAG processing



Airflow Components



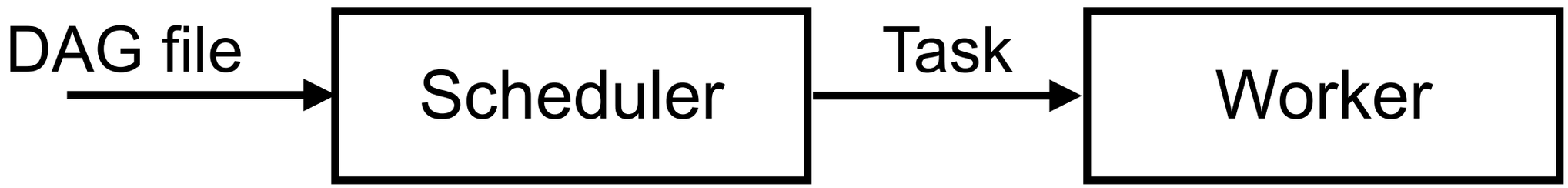
3 Main Components



Scalability and performance

Scheduler

Parse DAG files
Check schedule of DAG and start
Send tasks to worker



Worker

Picks up tasks that are scheduled for execution and execute them

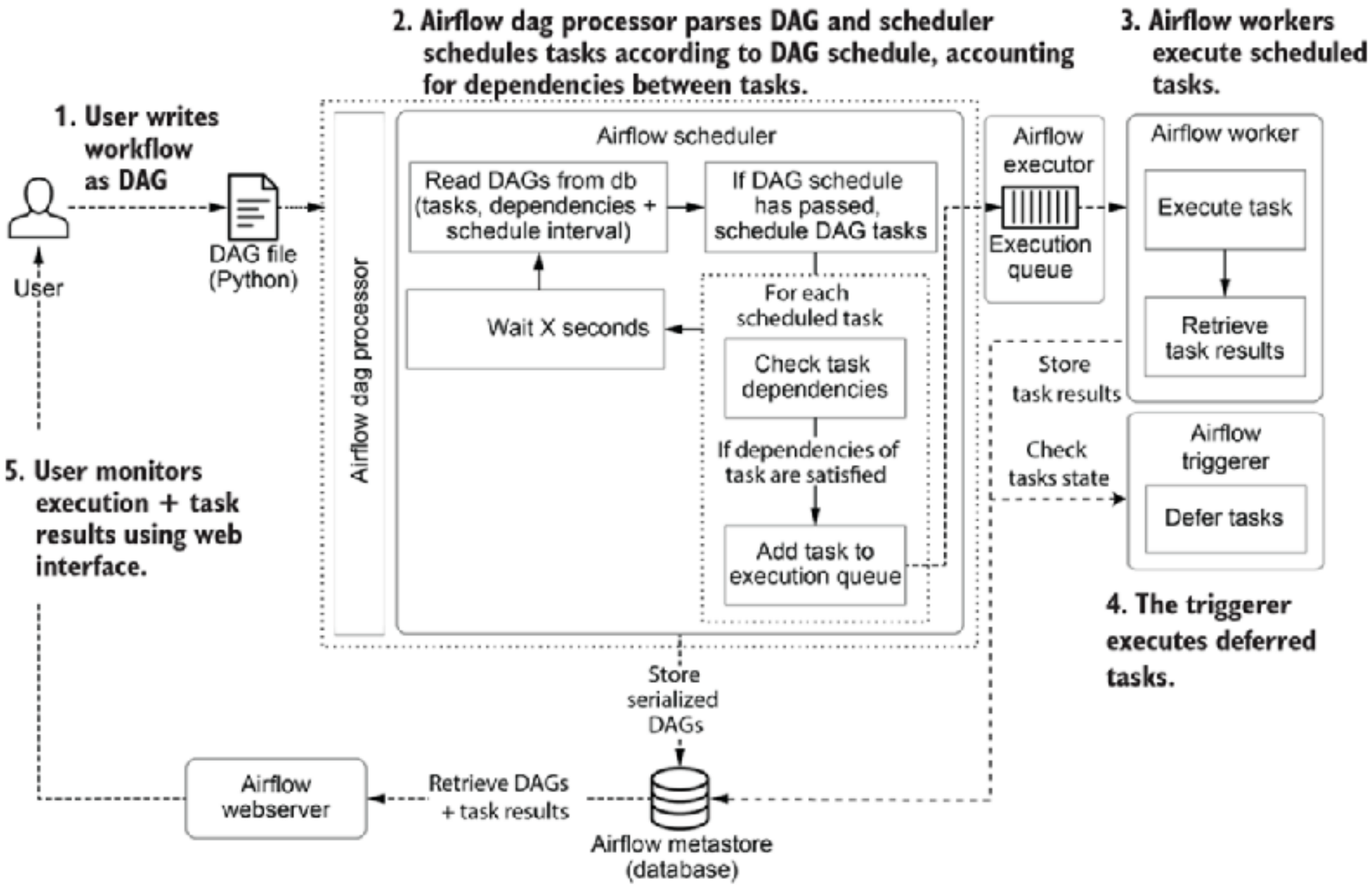
Doing the work

Trigger

Checks task completion status for tasks that support asynchronous processing

Allows o check for specific conditions in the background

Execution pipeline of DAG



Apache Airflow Deployment

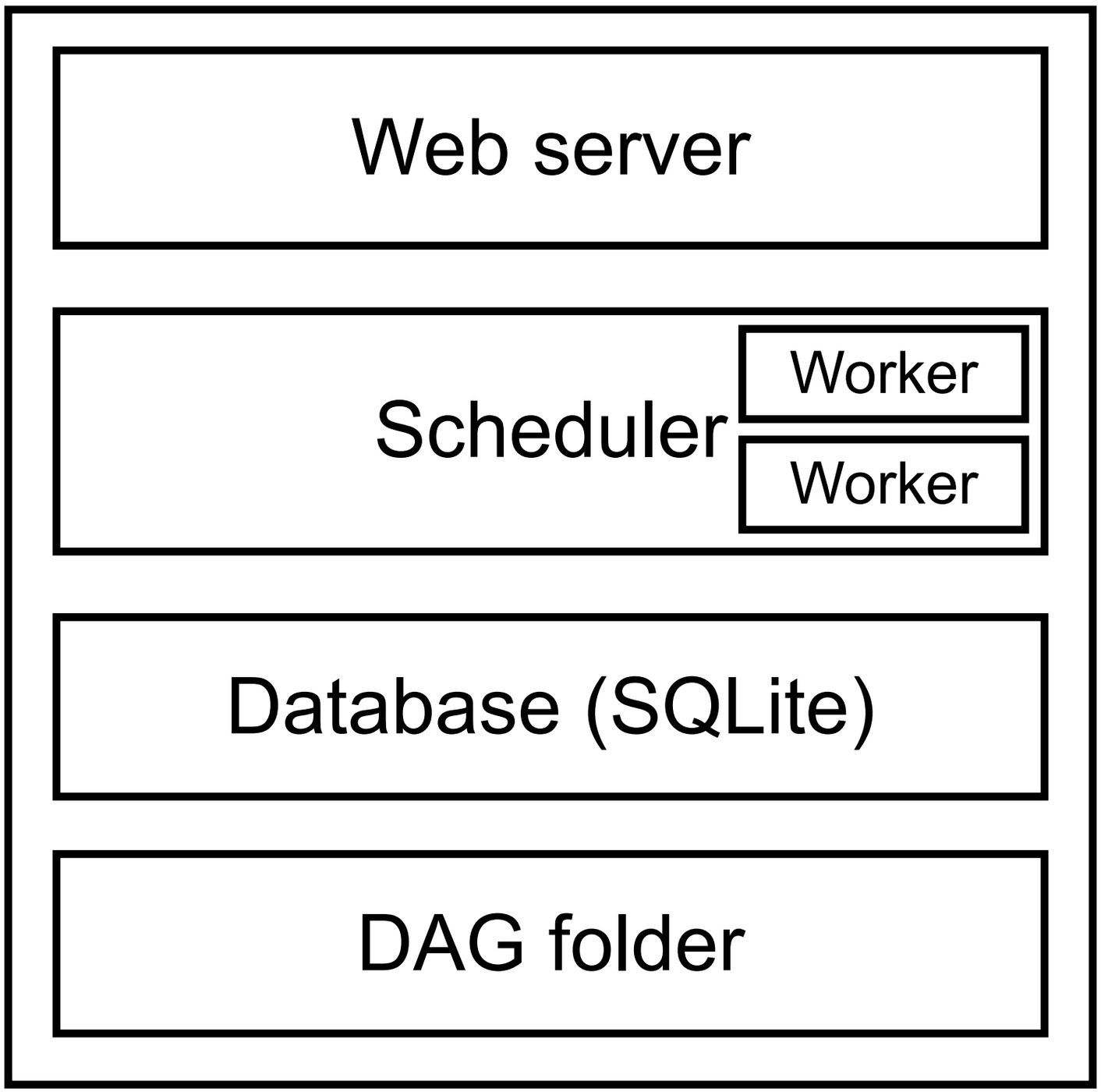
https://github.com/up1/workshop-apache-airflow/wiki/Installation

Deployment Airflow

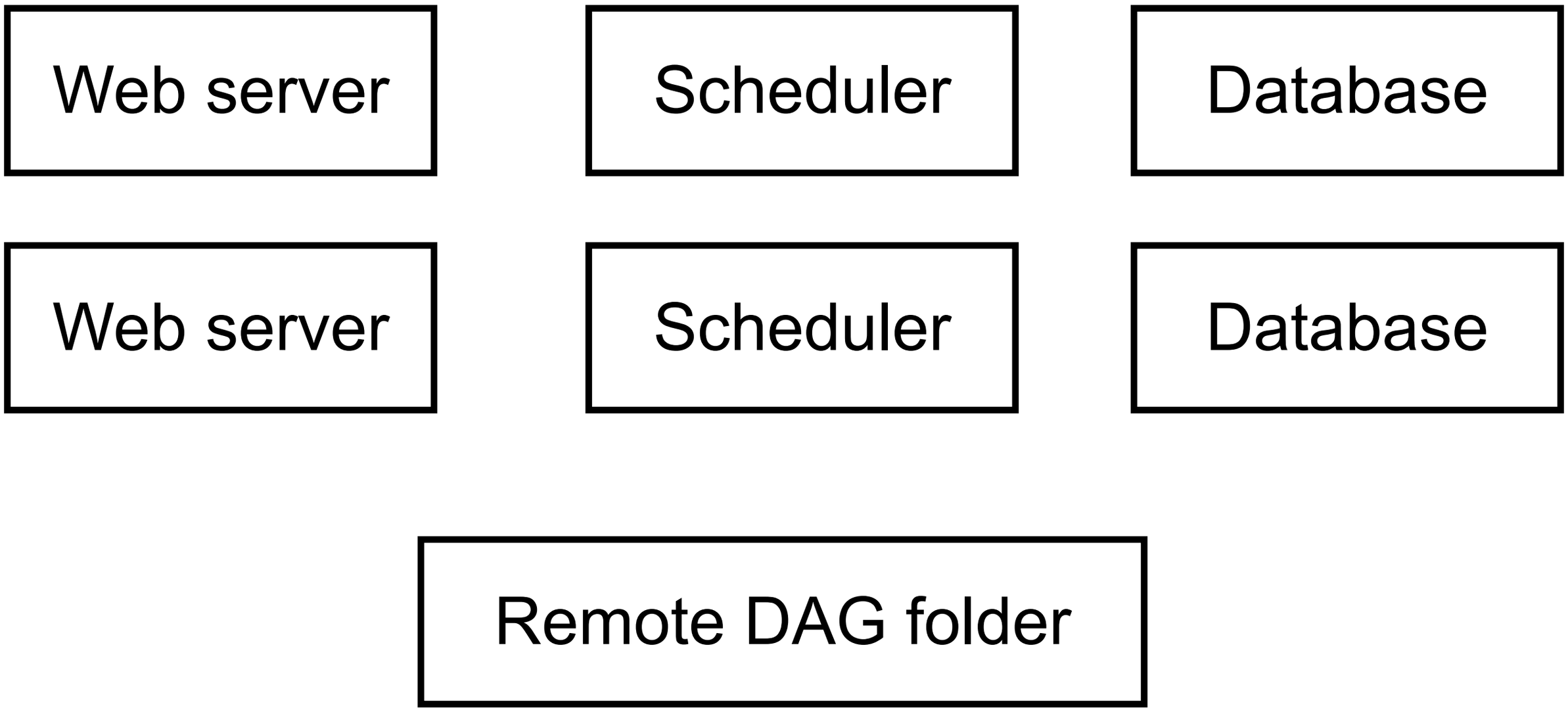
Single machine

Distributed

Single machine

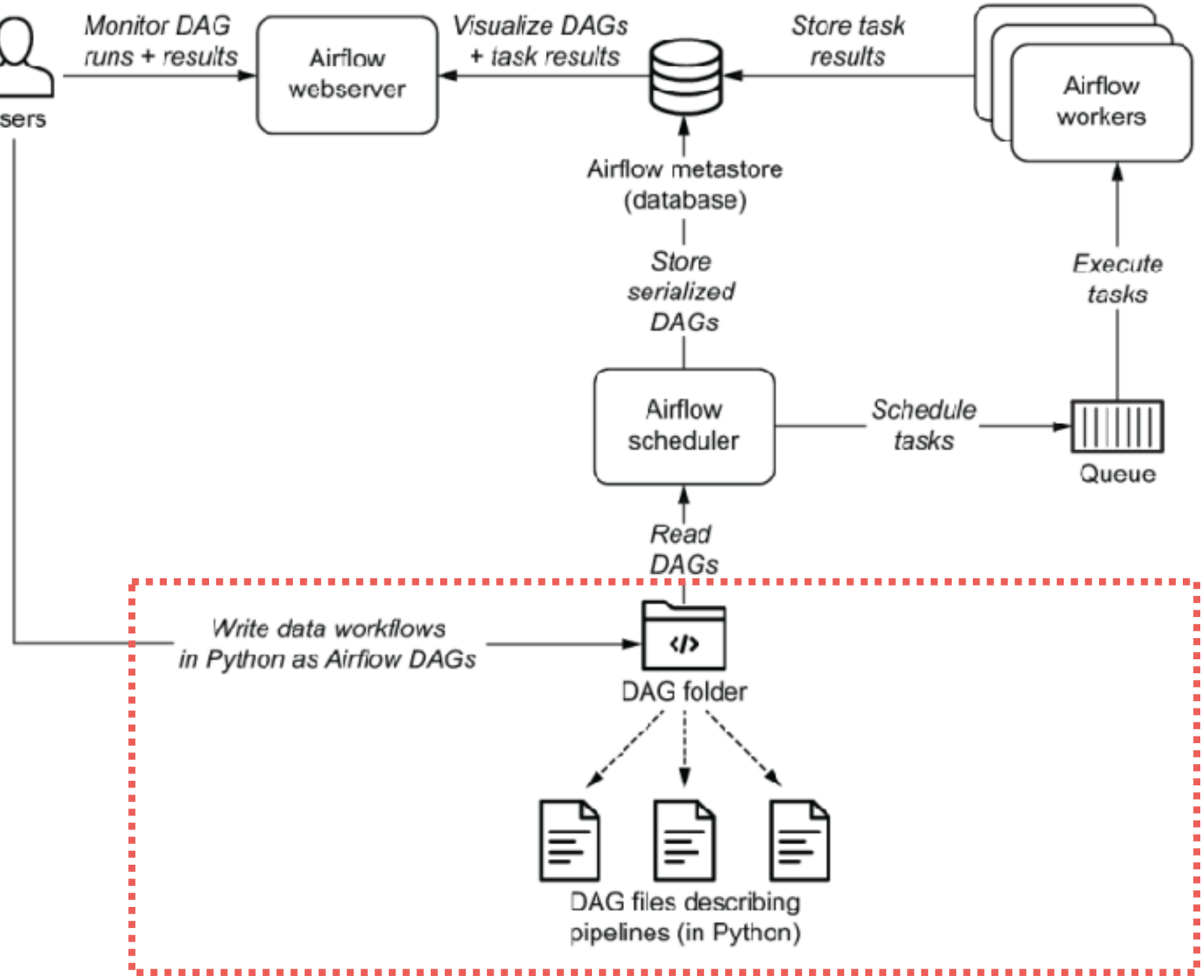


Distributed deployment

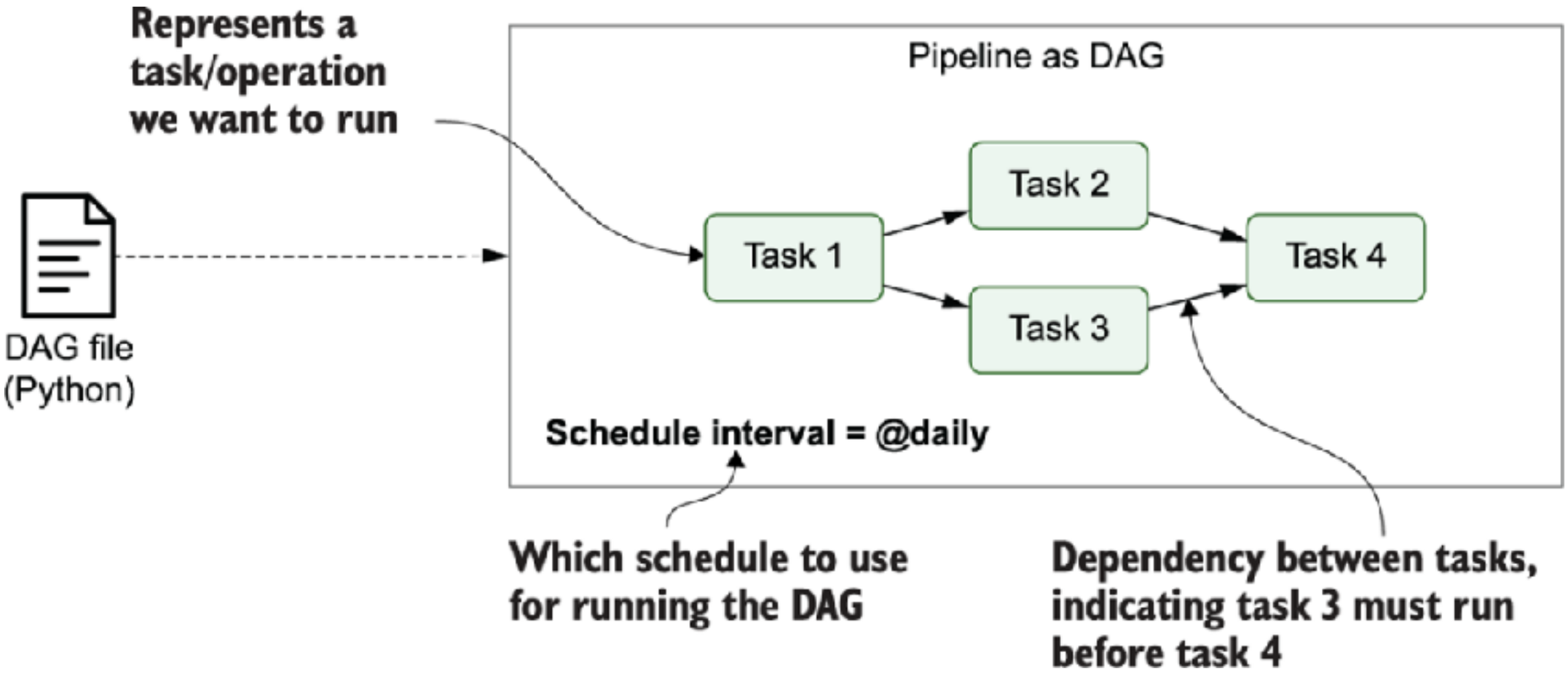


Back to DAG and Tasks

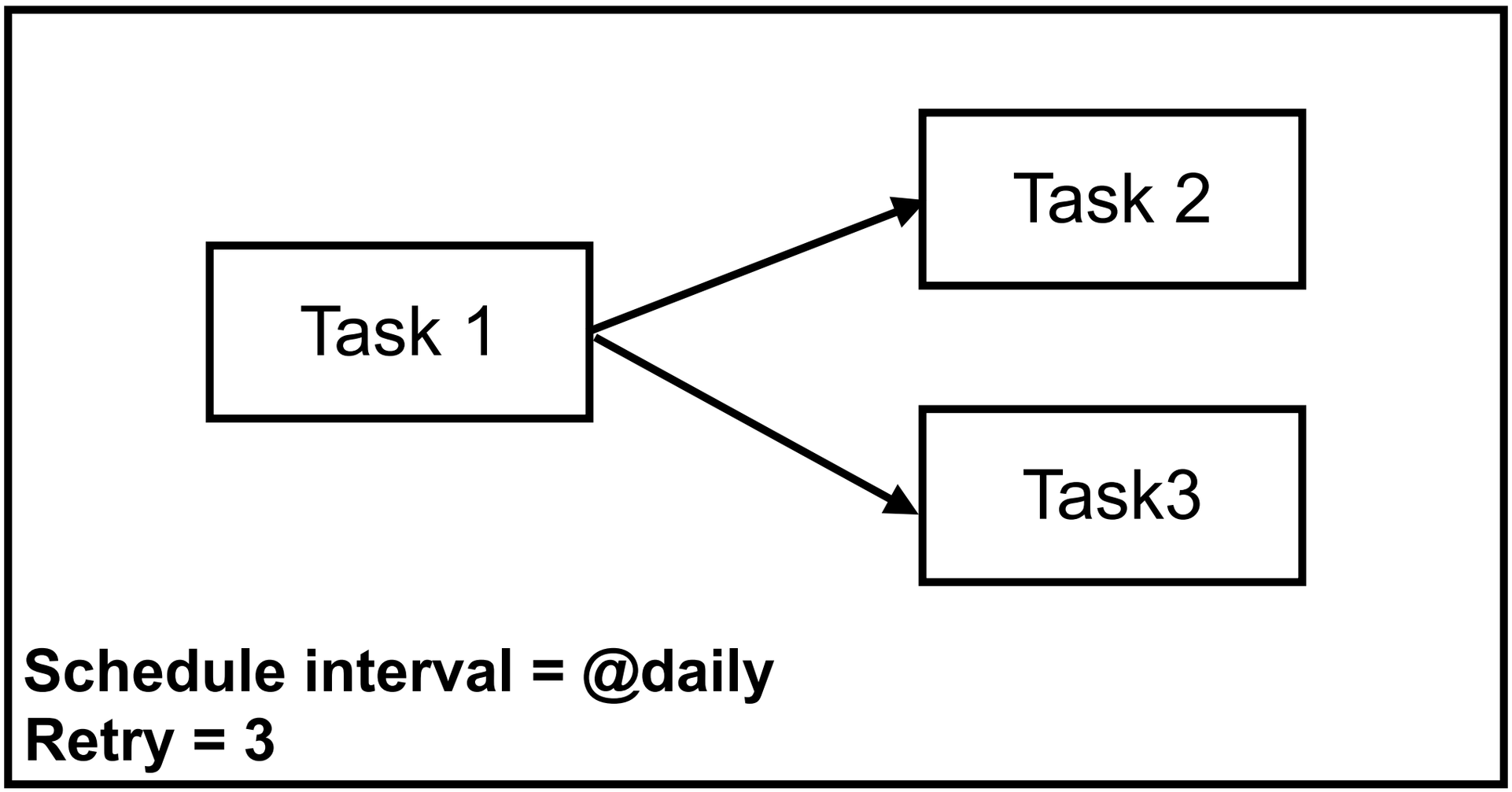
Create DAG file



Hello DAG

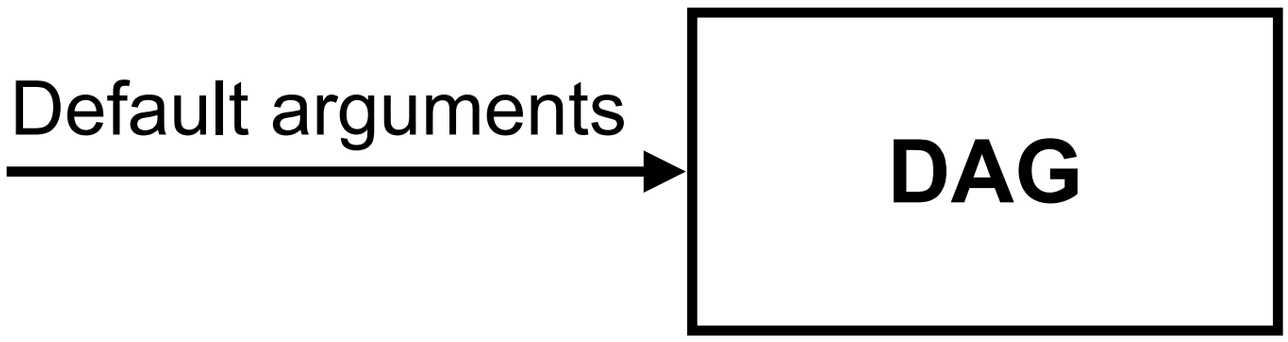


Hello DAG



Pipeline as a Code

Create DAG



Declare DAG (1)

แบบที่ 1

```
with DAG(
    dag_id="my_dag_name",
    start_date=datetime.datetime(2021, 1, 1),
    schedule="@daily",
):
    EmptyOperator(task_id="task")
```

แบบที่ 2

```
my_dag = DAG(
    dag_id="my_dag_name",
    start_date=datetime.datetime(2025, 1, 1),
    schedule="@daily",
)

EmptyOperator(task_id="task", dag=my_dag)
```

https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html

Declare DAG (2)

แบบที่ 3

```python
@dag(start_date=datetime.datetime(2025, 1, 1), schedule="@daily")
def generate_dag():
    EmptyOperator(task_id="task")


generate_dag()
```

https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html

Schedule ?

Schedule ?

```
default_args = {
    'owner': 'demo-01',
    'retries': 5,
    'retry_delay': timedelta(minutes=2)
}

with DAG(
    dag_id='hello_dag',
    default_args=default_args,
    description='This is our first dag that we write',
    start_date=datetime(2024, 12, 31),
    end_date=datetime(2025, 1, 31),
    schedule='@daily'
) as dag:
```

Scheduler

Schedule_interval or Schedule

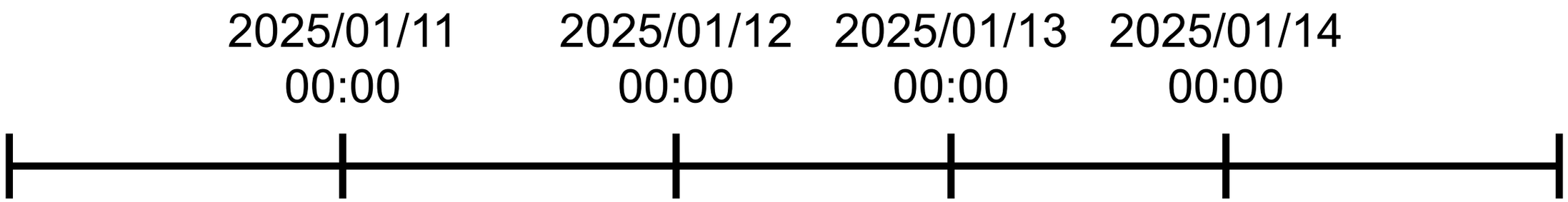
None	@hourly	@weekly
@daily	@monthly	@once
@continuous	Crontab	

Scheduler presets

Preset	Description
None	Don’t schedule (manual or external trigger)
@once	Only once
@continuous	Run as soon as the previous run finish
@daily @quaterly	Run at midnight

Example

start_date, end_date and @daily



start_date
2025/01/10
00:00

end_date
2025/01/15
00:00

```python
with DAG(
    dag_id="daily_schedule",
    schedule="@daily",
    start_date=datetime(2025, 1, 10),
    end_date=datetime(2025, 1, 15)
):
```

Example (2)

Run every 3 days

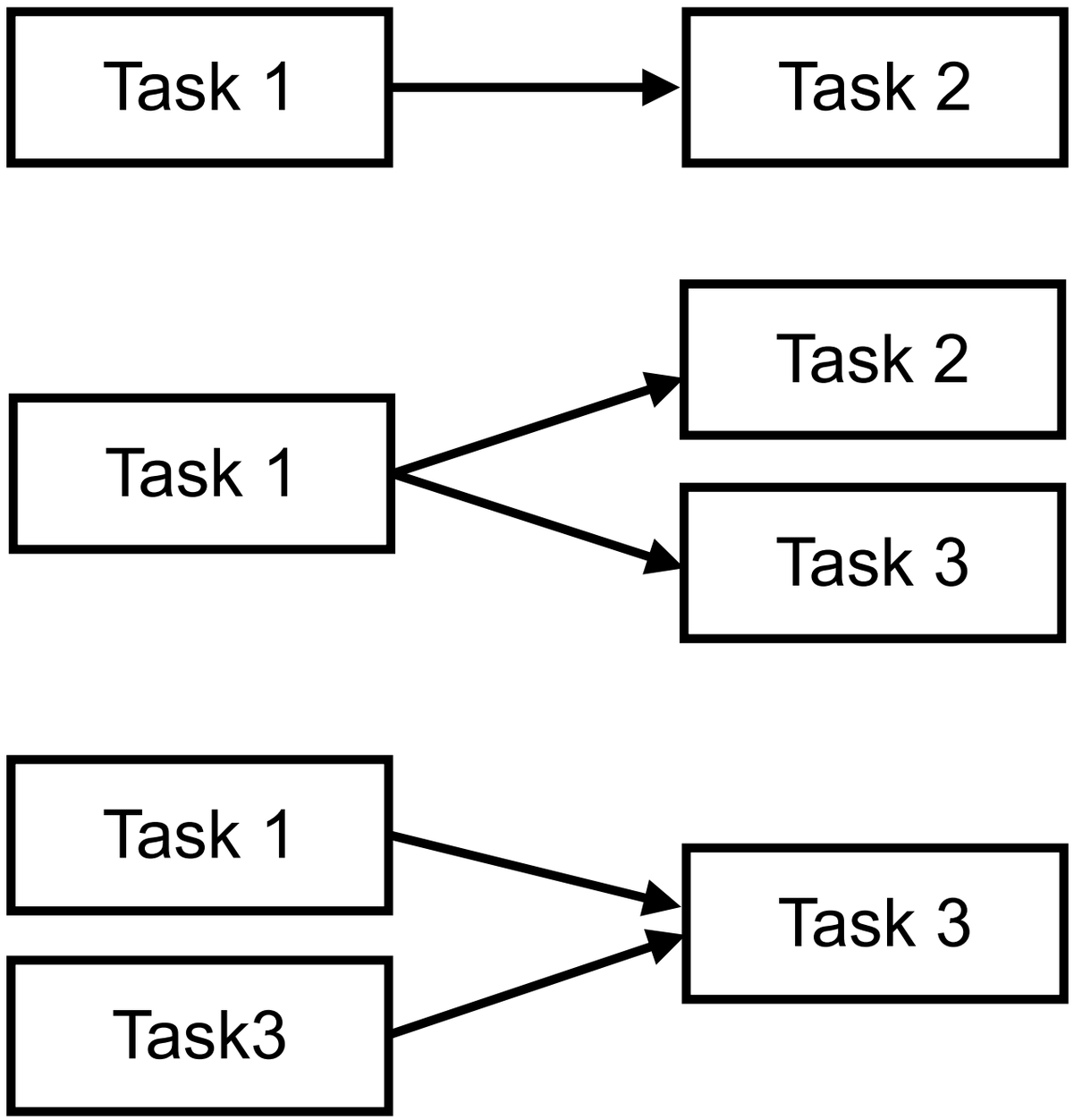
```python
from airflow import DAG
import pendulum
import datetime

with DAG(
    dag_id="03_timedelta",
    schedule=datetime.timedelta(days=3),
    start_date=pendulum.datetime(2025, 1, 1),
    end_date=pendulum.datetime(2025, 1, 10),
):
```

Crontab expression

https://crontab.guru/

Task dependencies

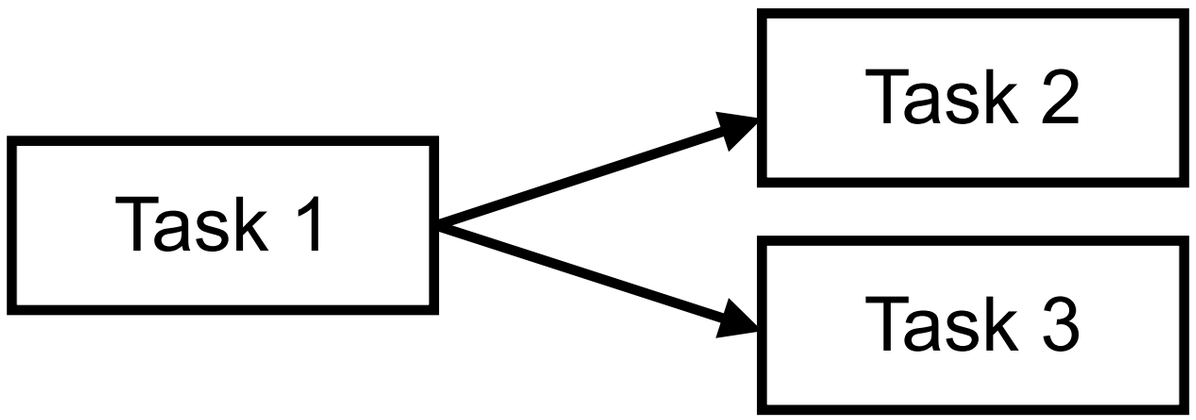
Task dependencies



https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/tasks.html#relationships

Task dependencies

```python
# Task dependency method 1
task1.set_downstream(task2)
task1.set_downstream(task3)

# Task dependency method 2
task1 >> task2
task1 >> task3

# Task dependency method 3
task1 >> [task2, task3]
```



https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/tasks.html#relationships

Control Flow

Control Flow

Branching
Trigger rules
Setup and teardown
Latest only
Depend on past

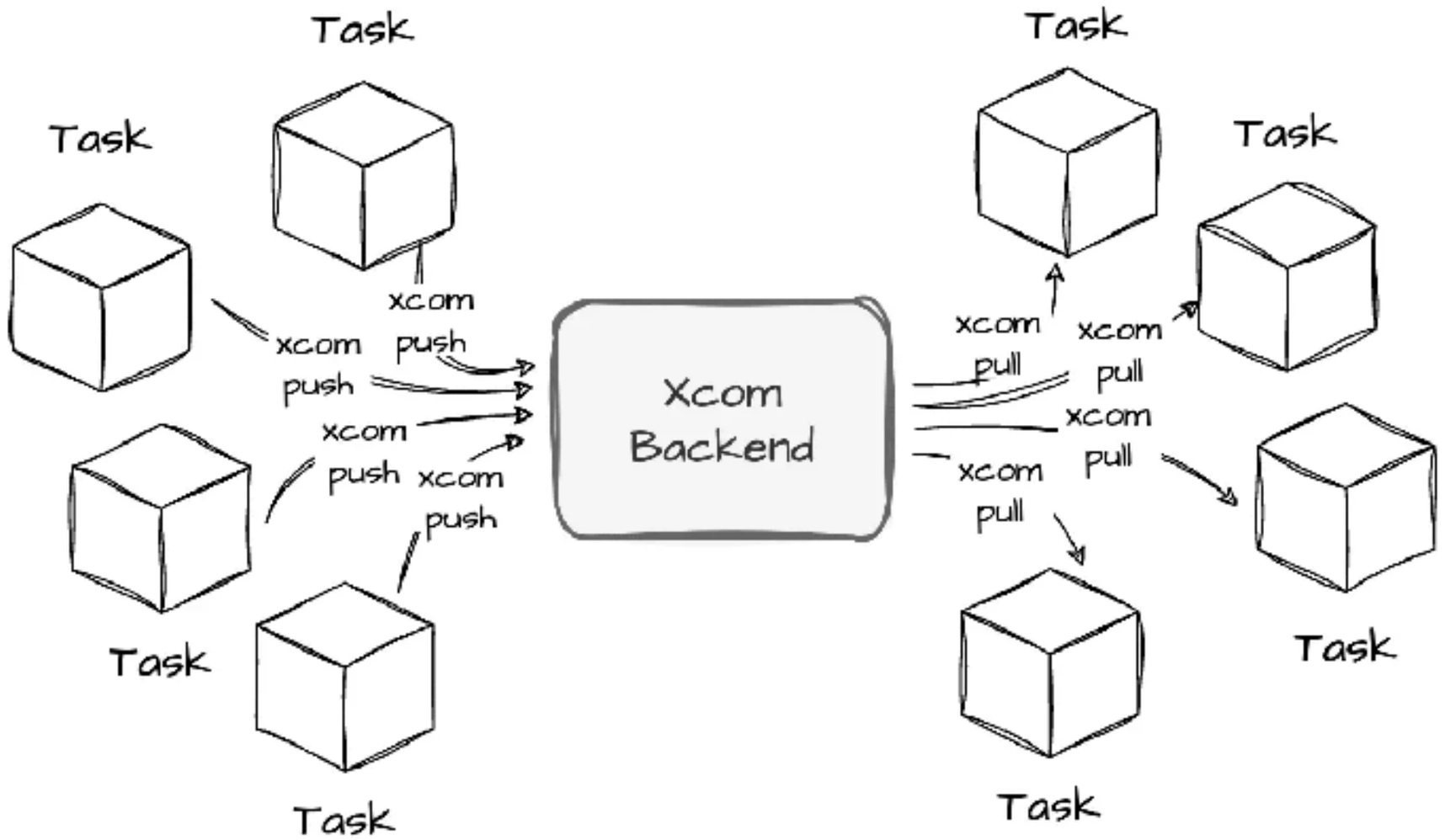
https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html#control-flow

Exchange data between tasks

Sharing data

Exchange data between tasks

Using XCom (Cross Communication)



https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/xcoms.html

Using XCom (Cross Communication)

Using XCom (Cross Communication)

Task Operators

Task with operators

Empty/Dummy Operator

Bash Operator

Python Operator

Kubernetes Pos Operator

Postgres Operator

TaskFlow @task

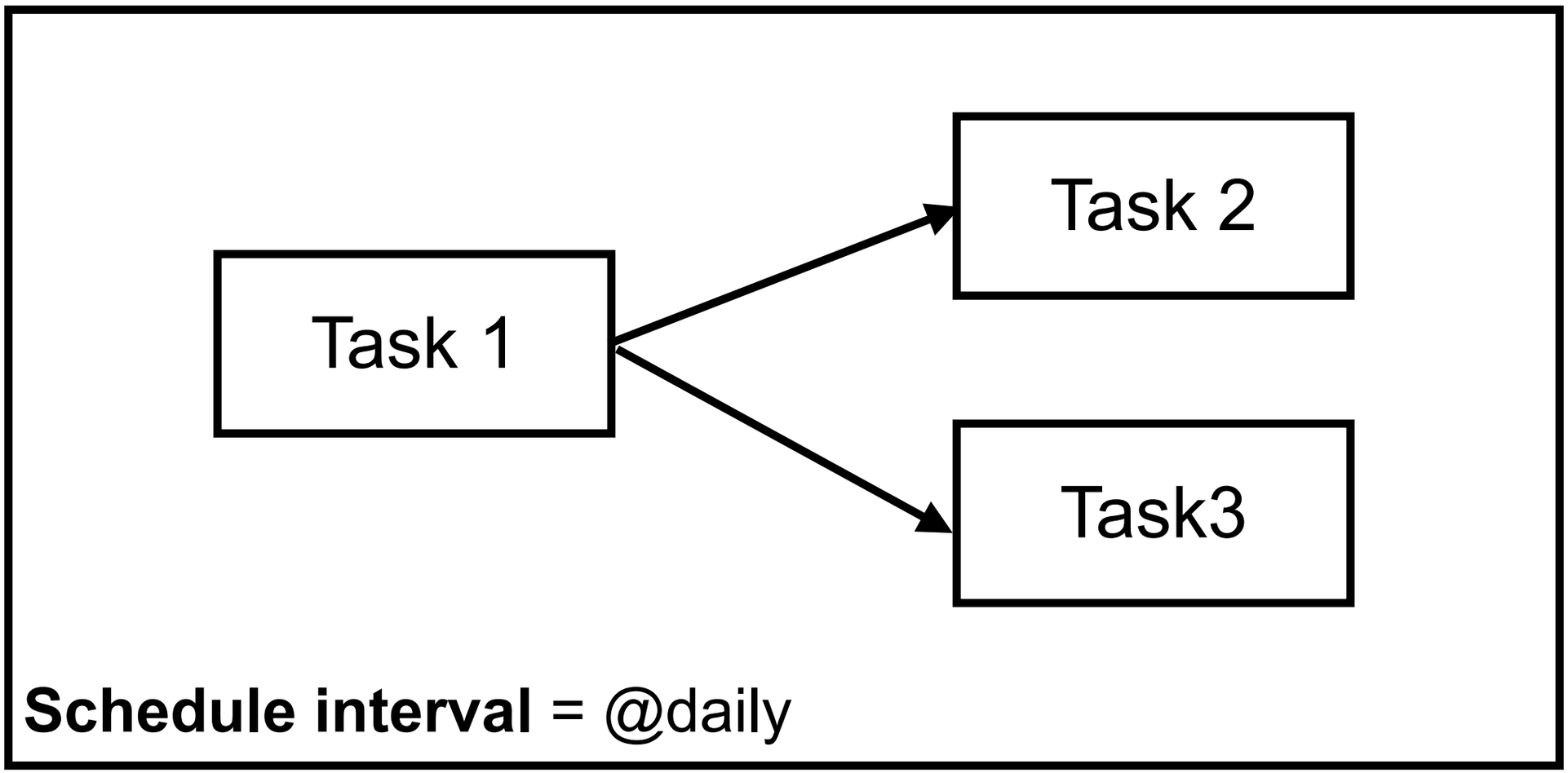
Sensors
(Wait something to happen)

Empty vs Dummy

Feature	EmptyOperator	DummyOperator
Purpose	Used to indicate an empty task that does nothing and is considered lightweight.	Used to create a placeholder or no-op task, often for testing or DAG structure.
Introduced in Version	Airflow 2.4+	Earlier versions of Airflow (pre-2.0).
Execution Behavior	Executes as a lightweight no-operation task without a log message.	Executes as a no-operation task with minimal logging.
Best Suited For	Lightweight task with no overhead; replaces DummyOperator in new DAGs.	Placeholder tasks in older versions or for backward compatibility.
Deprecation Status	Actively supported and recommended.	Deprecated in Airflow 2.4+ in favor of EmptyOperator.
Logging	Minimal logging for better performance.	Logs basic execution messages.
Performance	More efficient with less overhead.	Slightly less efficient than EmptyOperator.
Use Case Example	Representing a task that does nothing in modern DAGs.	Creating task placeholders in legacy DAGs.
Import Path	`from airflow.operators.empty import EmptyOperator`	`from airflow.operators.dummy import DummyOperator`

Hello DAG (1)

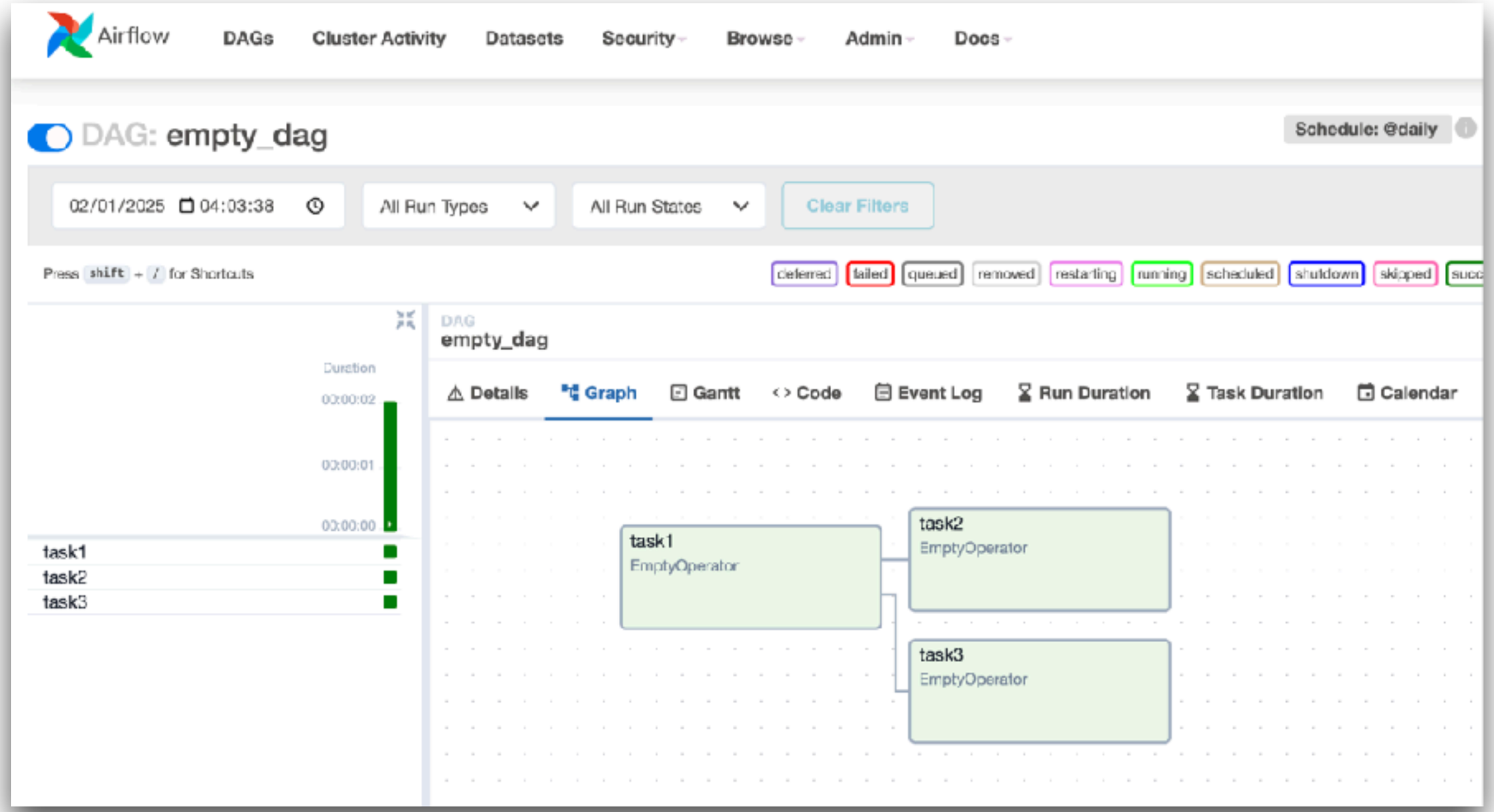
Working with Empty/DummyOperator



EmptyOperator

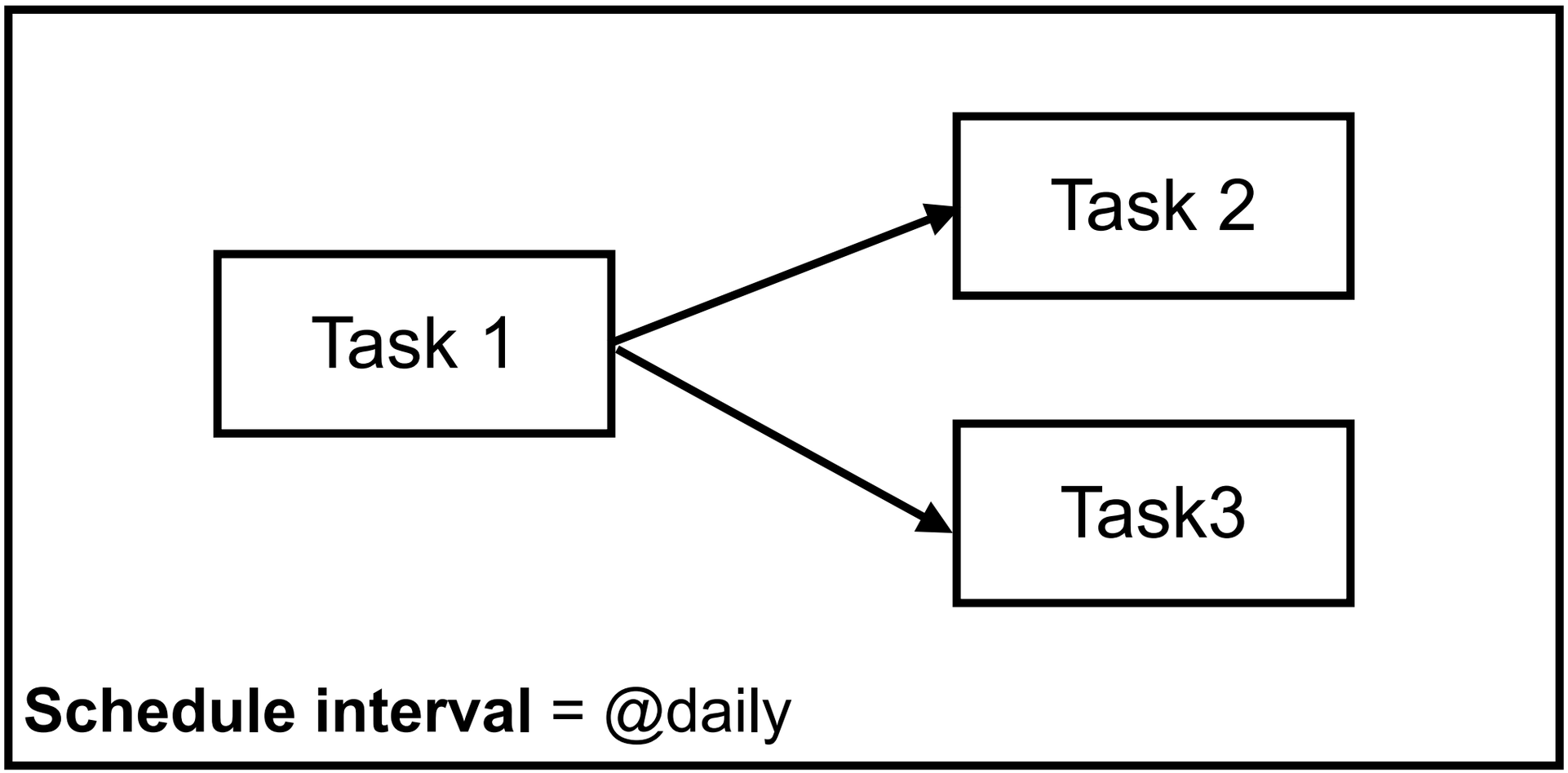
```python
from datetime import datetime
from airflow import DAG
from airflow.operators.empty import EmptyOperator

with DAG (
    dag_id='empty_dag',
    start_date=datetime(2025, 1, 2),
    schedule='@daily'
):

    task1 = EmptyOperator(
        task_id='task1'
    )
    task2 = EmptyOperator(
        task_id='task2'
    )
    task3 = EmptyOperator(
        task_id='task3'
    )
    task1 >> [task2, task3]
```

Run in Web UI



Hello DAG (2)

Working with BashOperator

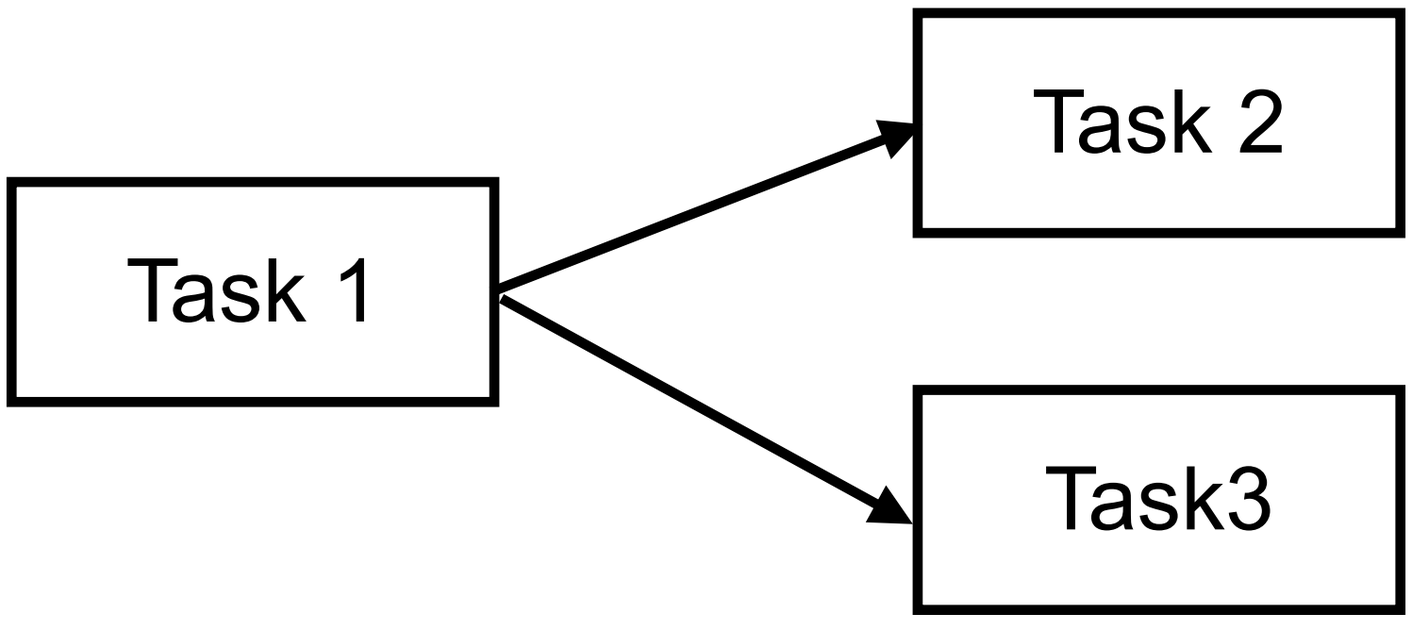


BashOperator

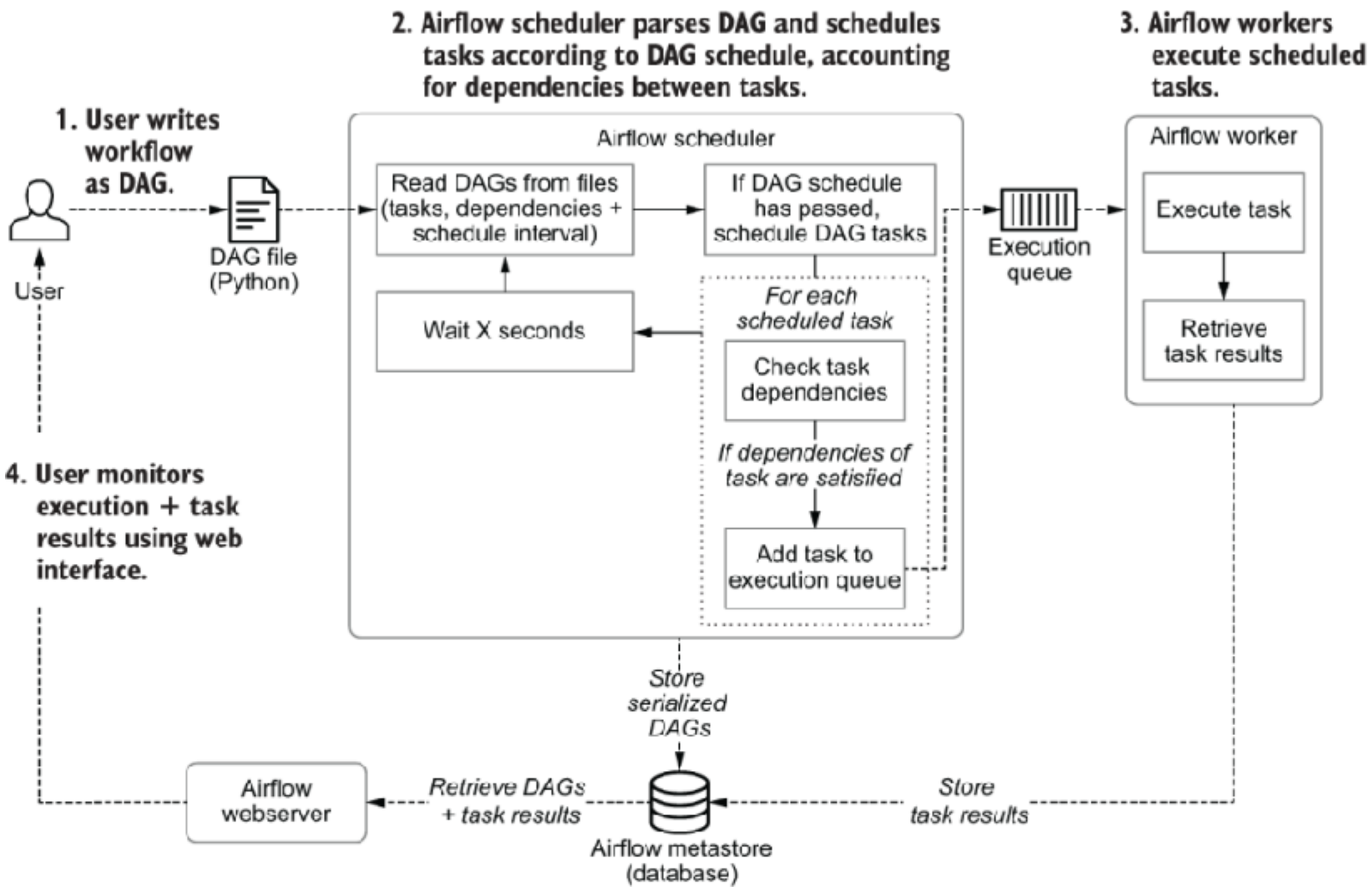
```
with DAG(
    dag_id='hello_dag_v5',
    default_args=default_args,
    description='This is our first dag that we write',
    start_date=datetime(2024, 12, 31),
    end_date=datetime(2025, 1, 31),
    schedule='@daily'
) as dag:
    task1 = BashOperator(
        task_id='first_task',
        bash_command="echo hello world, this is the first task!"
    )

    task2 = BashOperator(
        task_id='second_task',
        bash_command="echo hey, I am task2"
    )

    task3 = BashOperator(
        task_id='thrid_task',
        bash_command="echo hey, I am task3"
    )
```

Hello DAG (3)

Working with PythonOperator and TaskFlow



High level

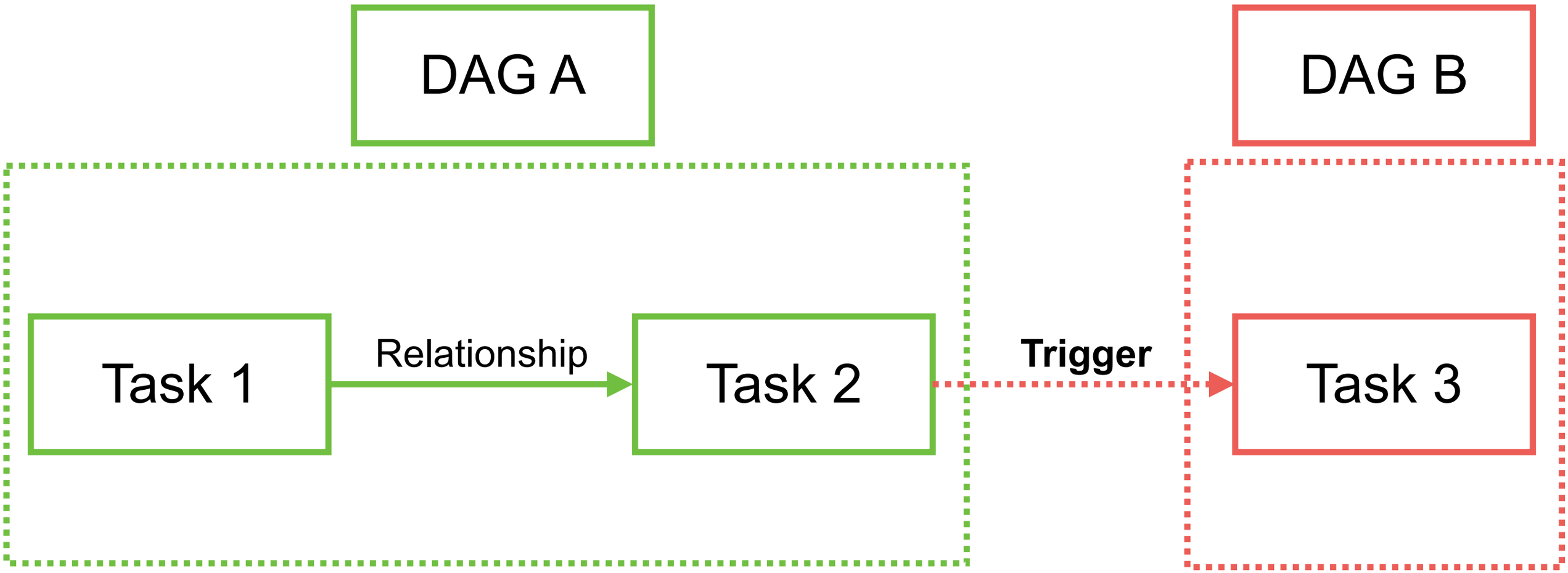


Trigger DAG

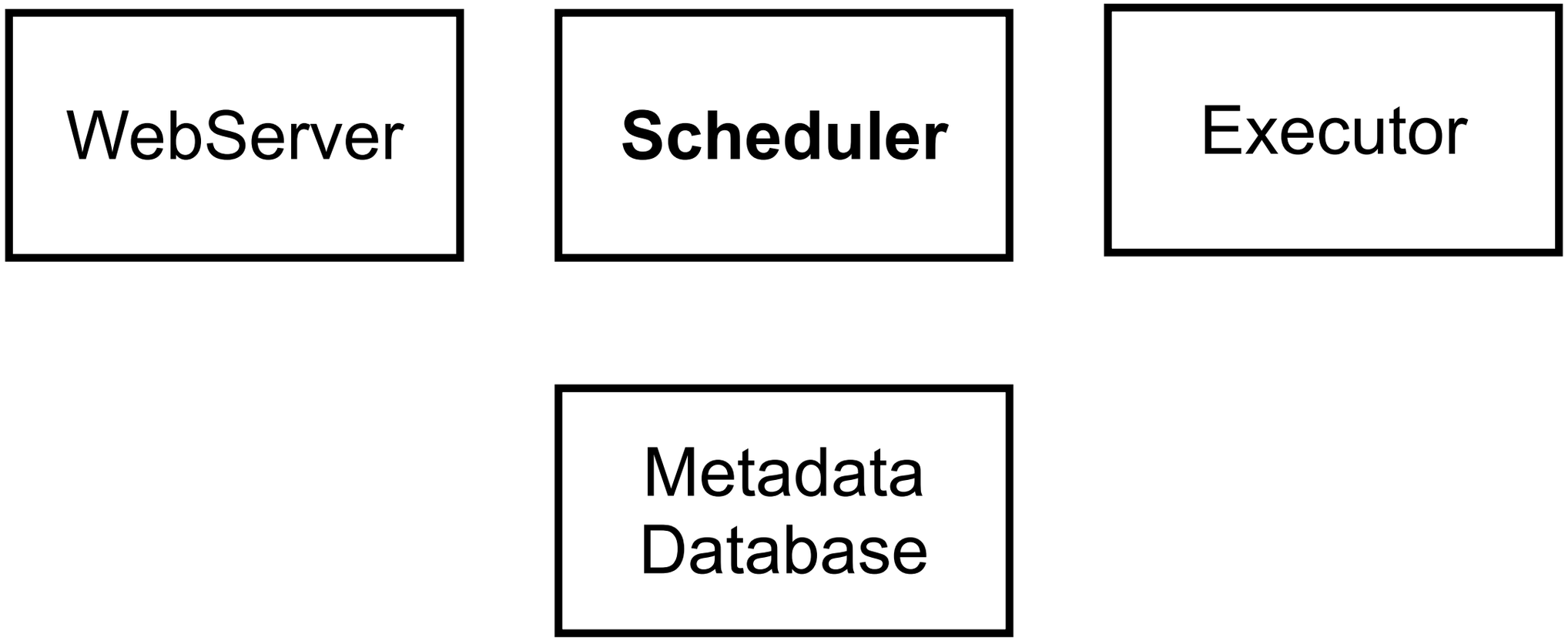
Airflow Trigger

On a schedule from DAG
Manual trigger (UI or CLI or API)
External trigger

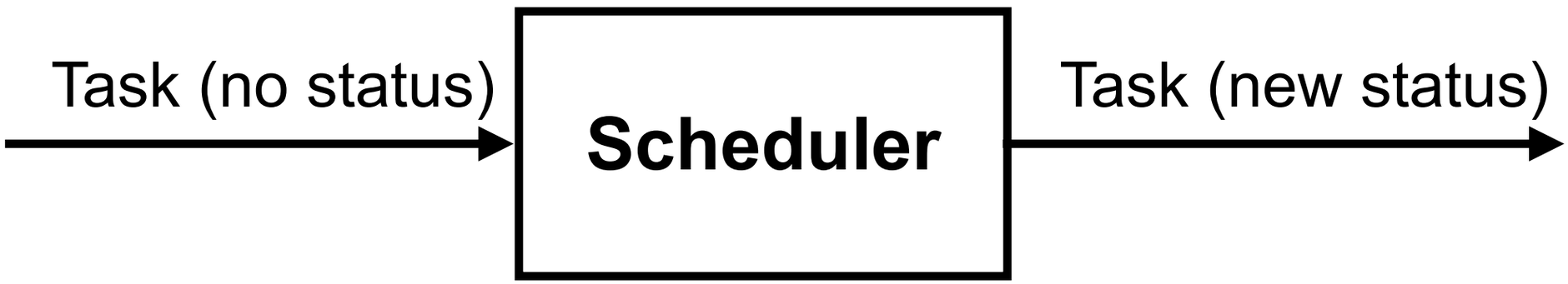
External Trigger



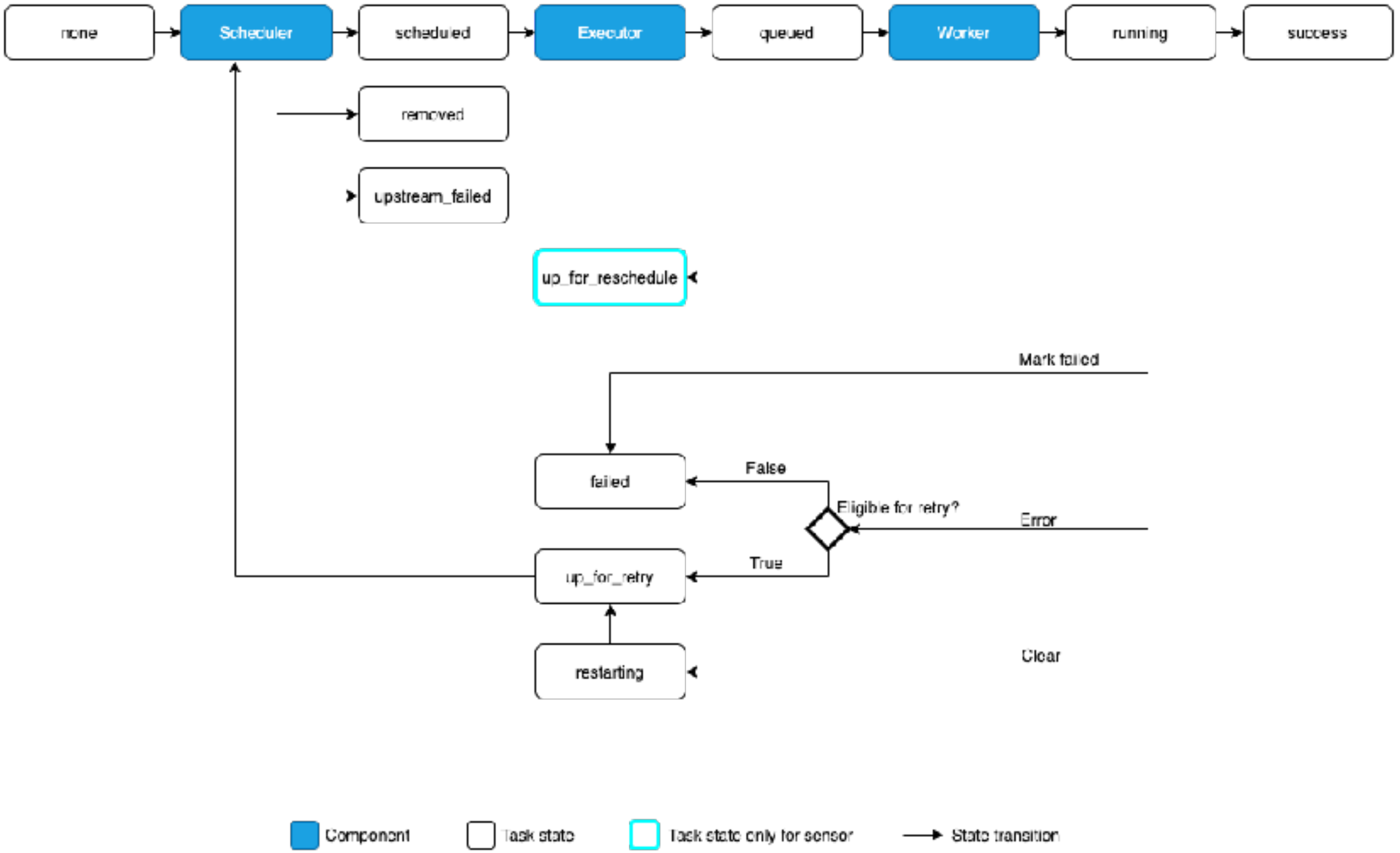
Simple Airflow architecture



Task Life Cycle

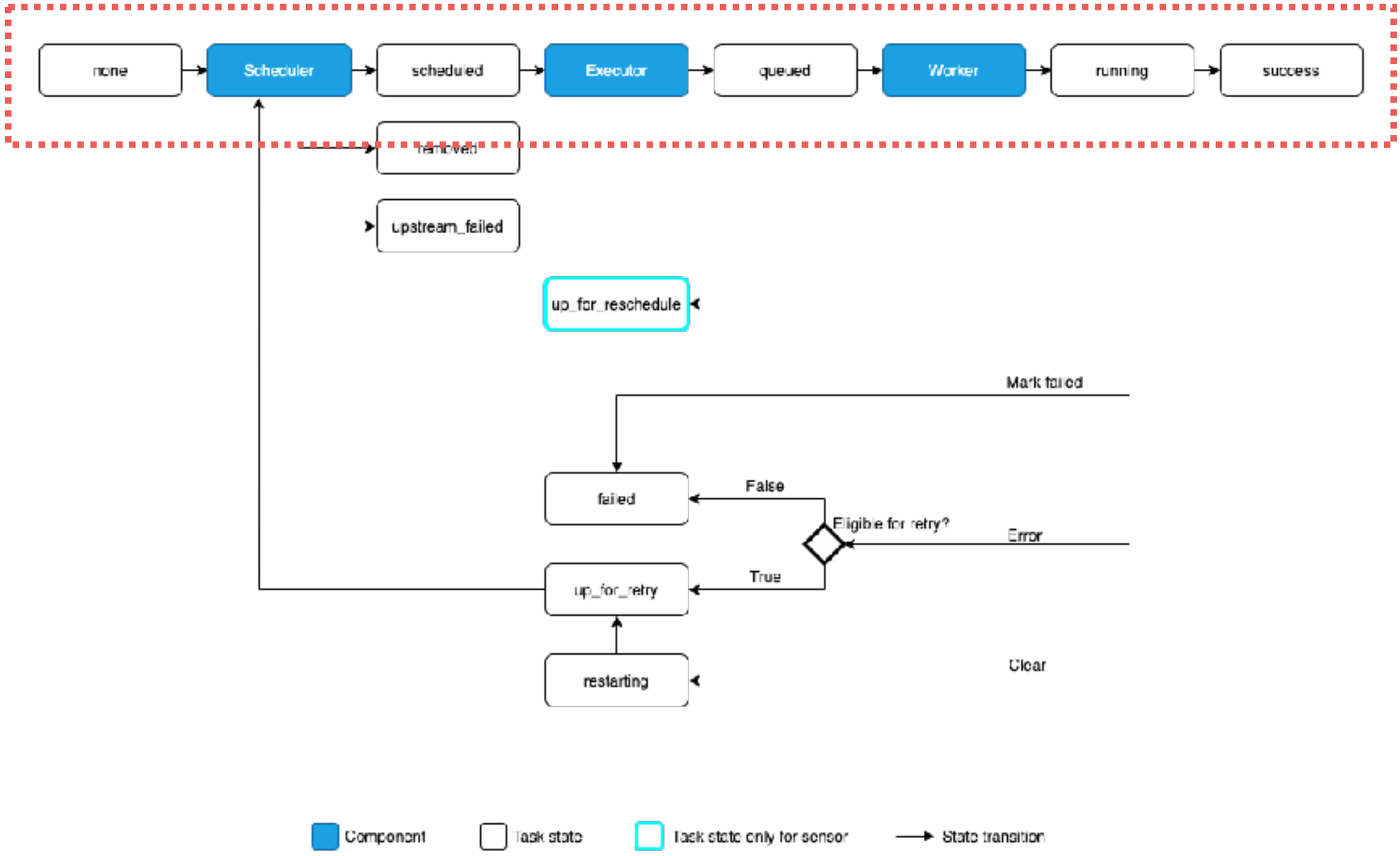


Task Life Cycle



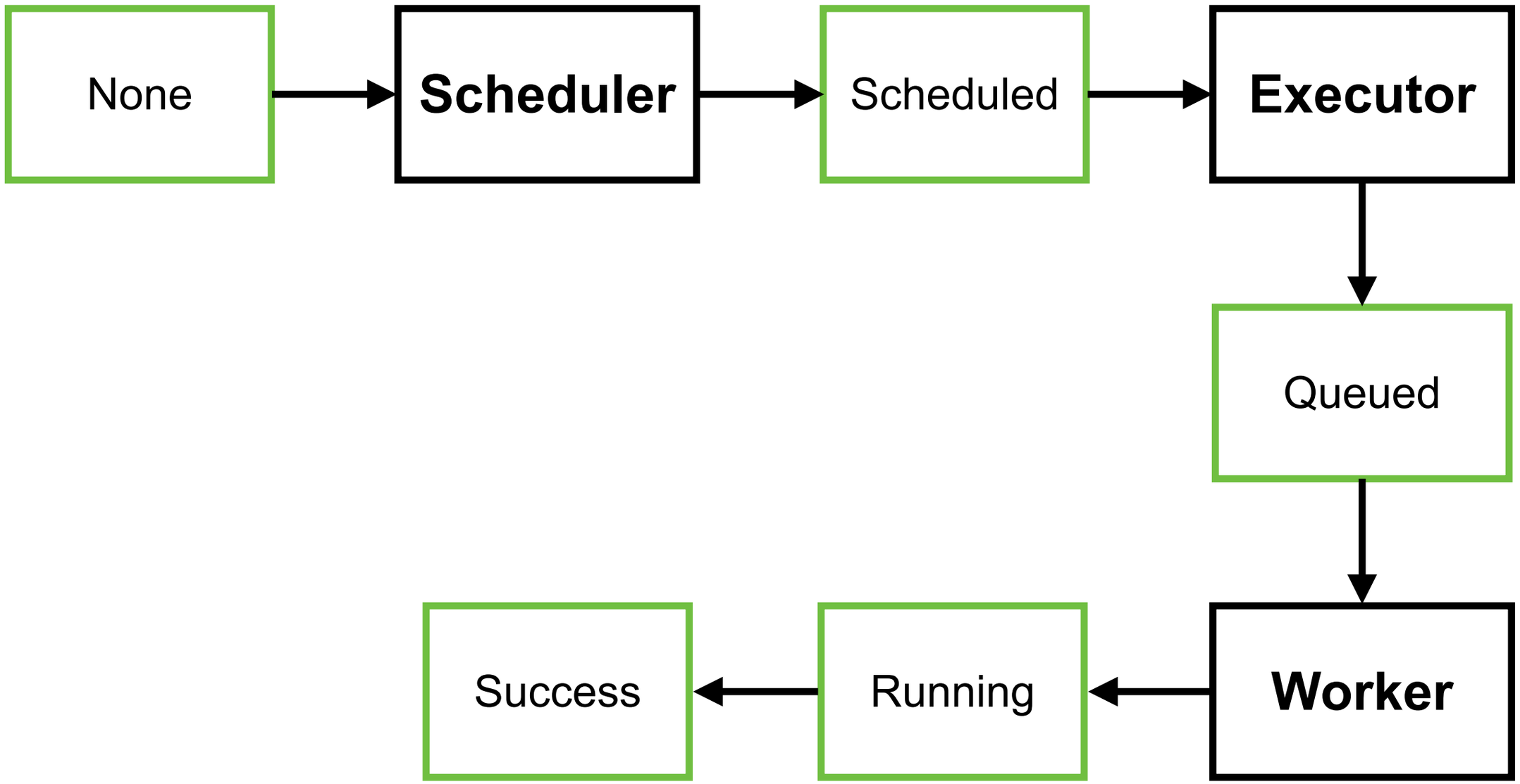
https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/tasks.html

Task Life Cycle (Normal flow)



https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/tasks.html

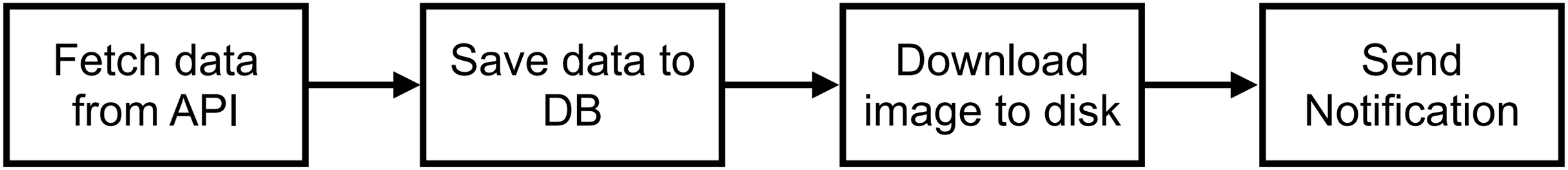
Task Life Cycle (Normal flow)



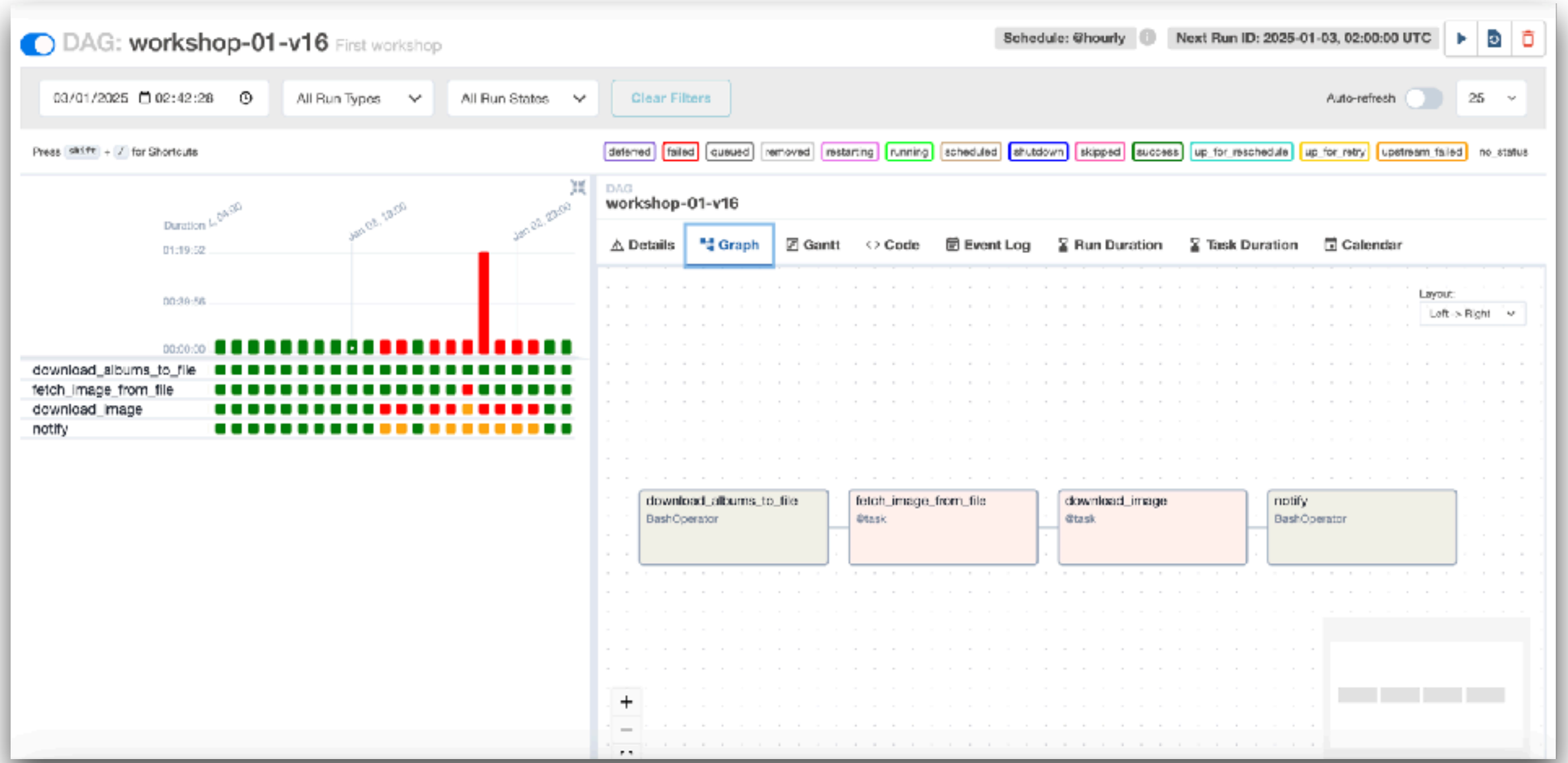
DAG workshop

https://github.com/up1/workshop-apache-airflow/wiki/DAG-workshop

DAG workshop

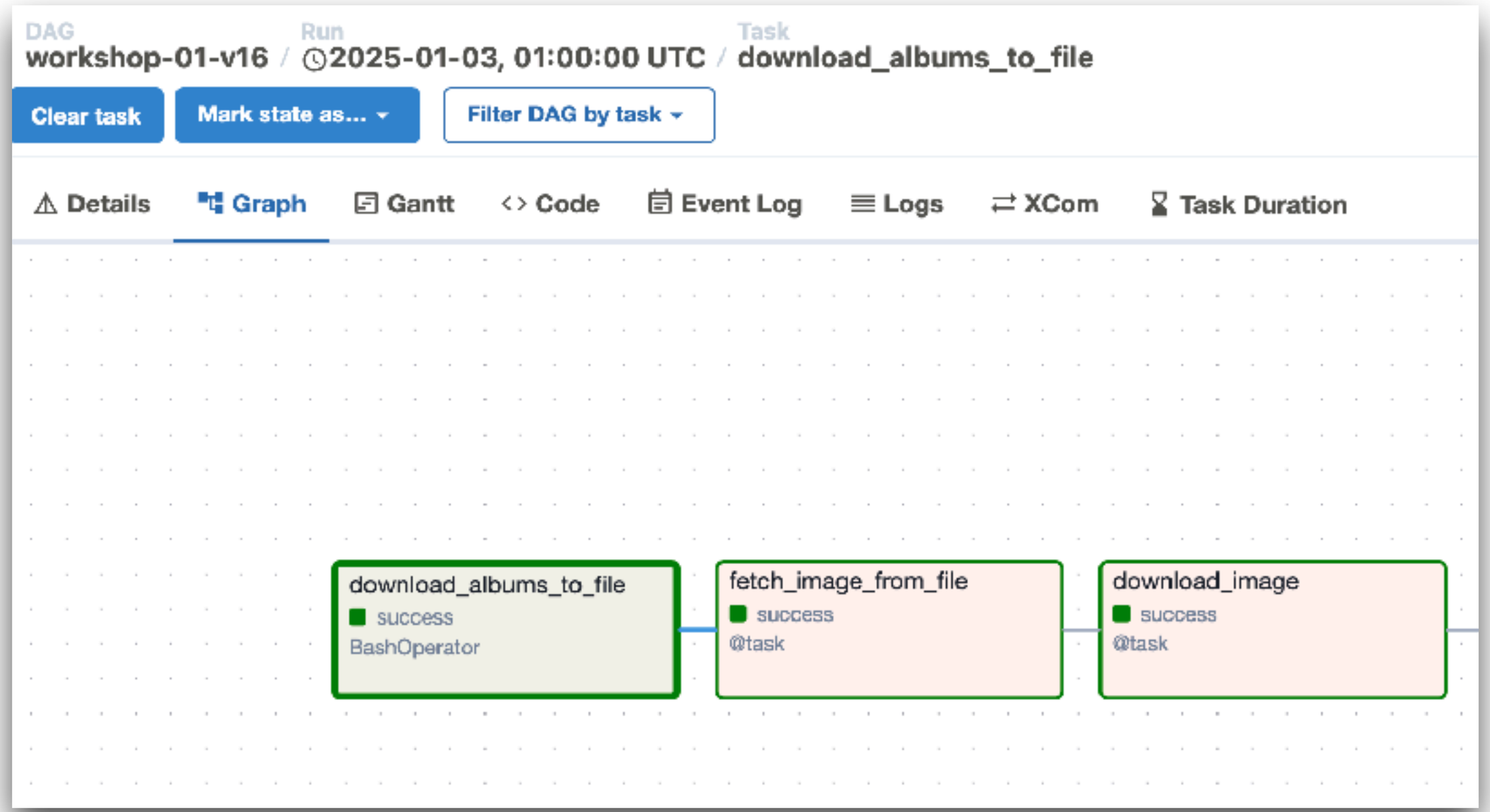


Airflow UI (1)



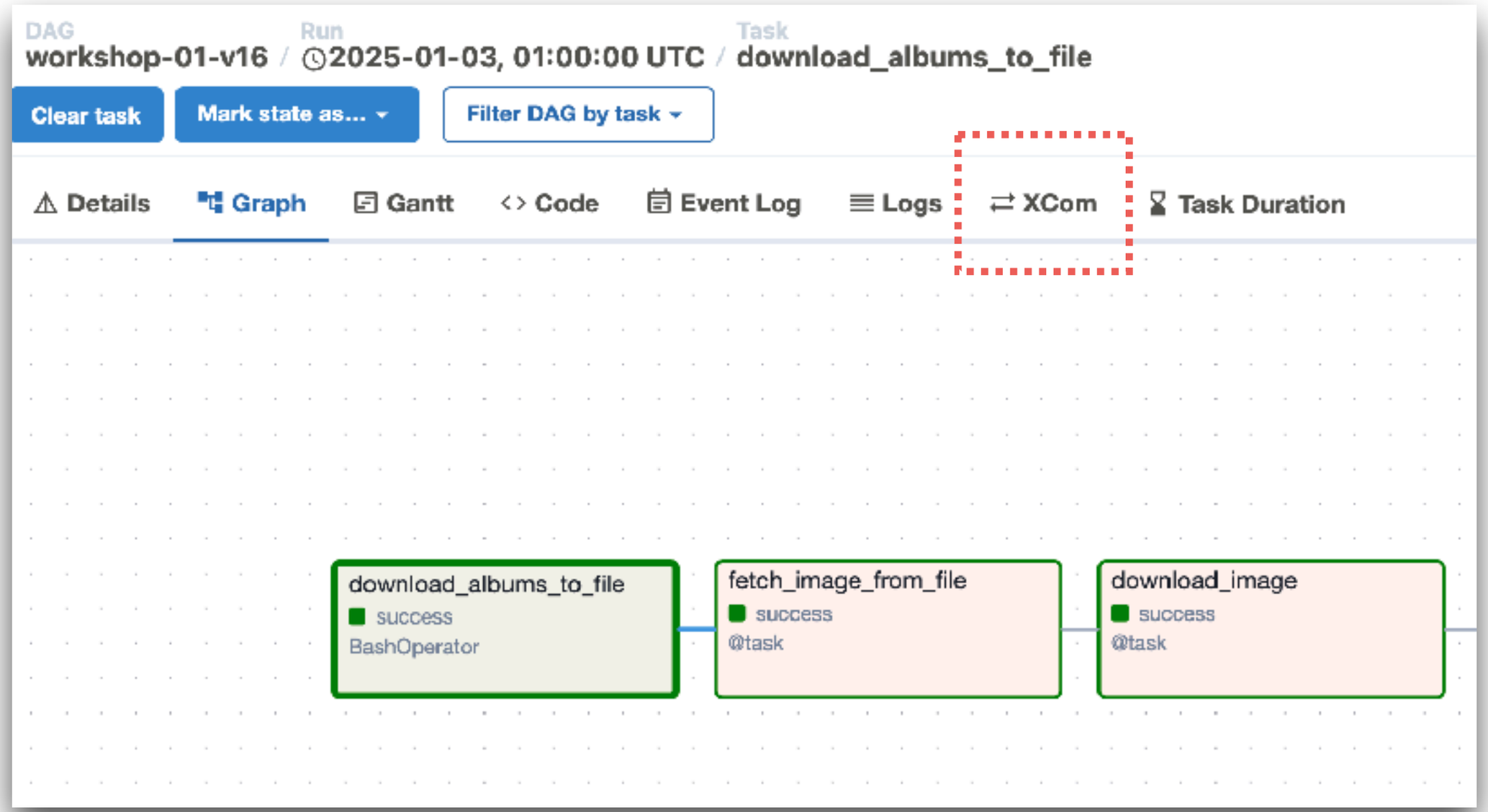
Airflow UI (2)

Detail in each task



Airflow UI (2)

XCom = Cross communication



https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/xcoms.html

Communication between tasks

XCom

Variable

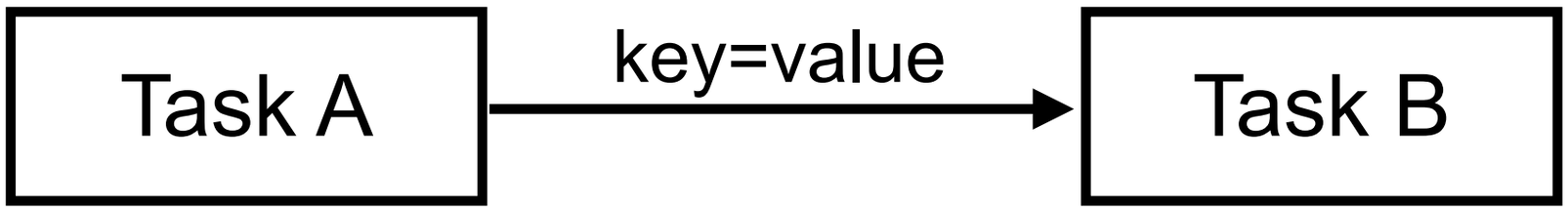
Params

External storage

Messaging

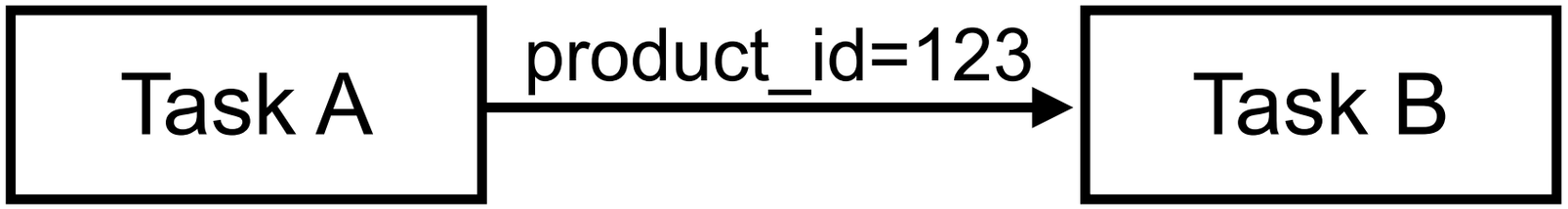
XCom (Cross Communication)

Inter-task communication
Task-level scope
Support any serializable Python object



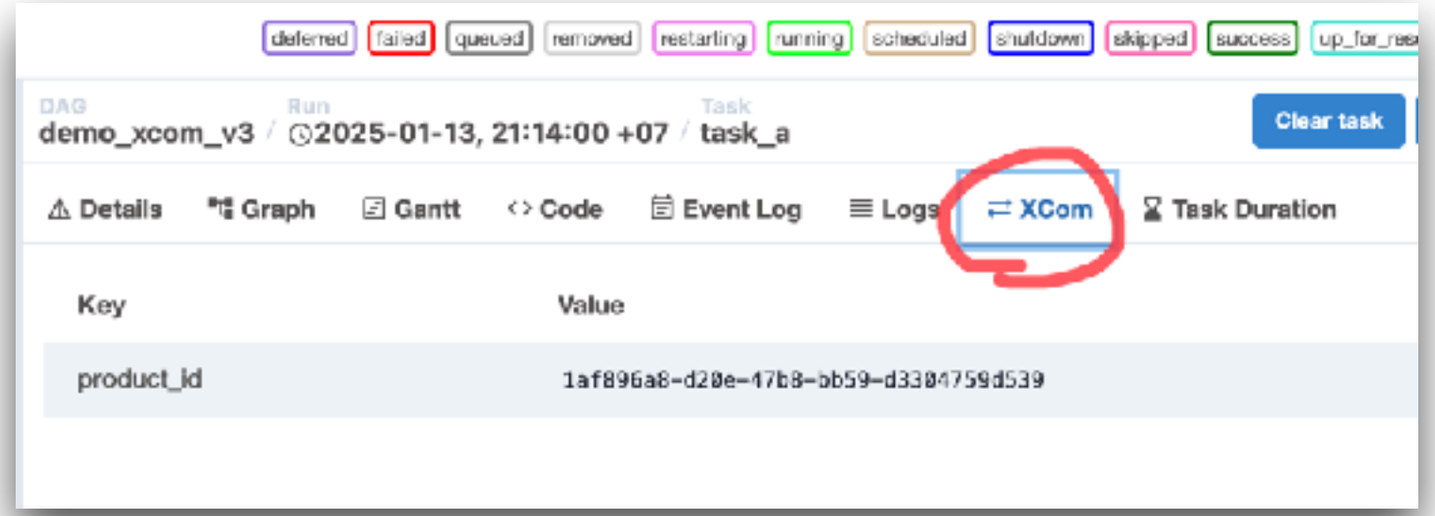
https://github.com/up1/workshop-apache-airflow/wiki/Communication-between-tasks

XCom (Cross Communication)




```
def _called_task_a(**context):
    product_id = str(uuid.uuid4())
    context["task_instance"].xcom_push(
            key="product_id", value=product_id)
```

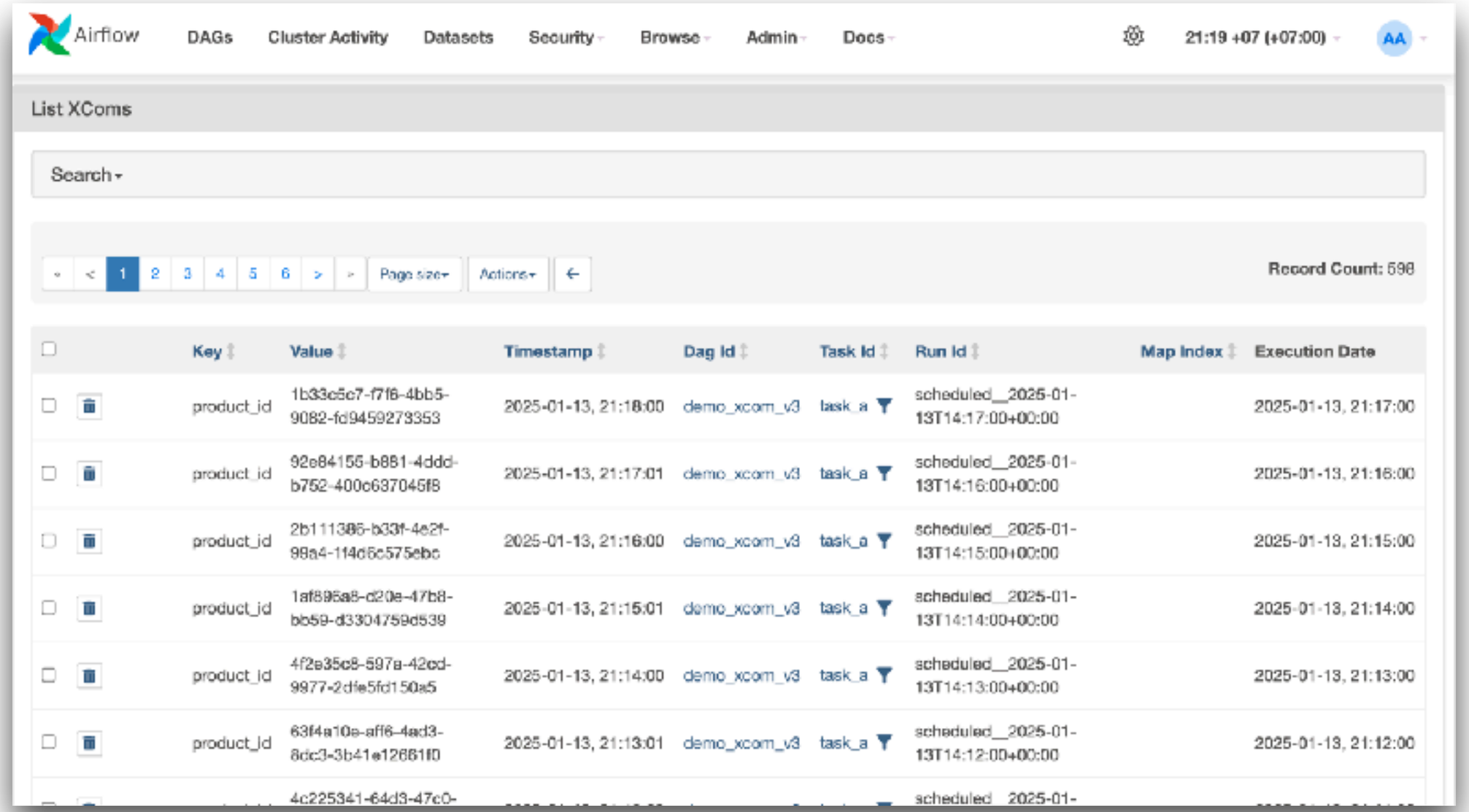
```
def _called_task_b(**context):
    product_id = context[“task_instance"].xcom_pull(
          key="product_id")
    print(product_id)
```



https://github.com/up1/workshop-apache-airflow/wiki/Communication-between-tasks

List of XComs

Admin -> XComs

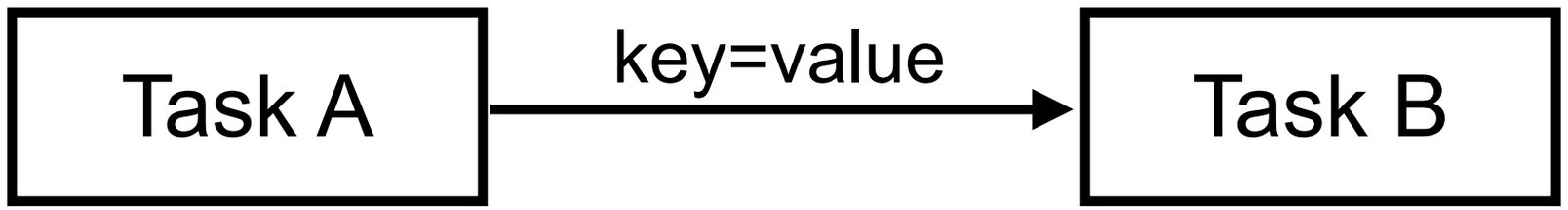


XCom data store in Database

Database	Type and Size
SQLite	BLOB, 2GB
PostgreSQL	BYTEA, 1GB
MySQL	BLOB, 64 KB

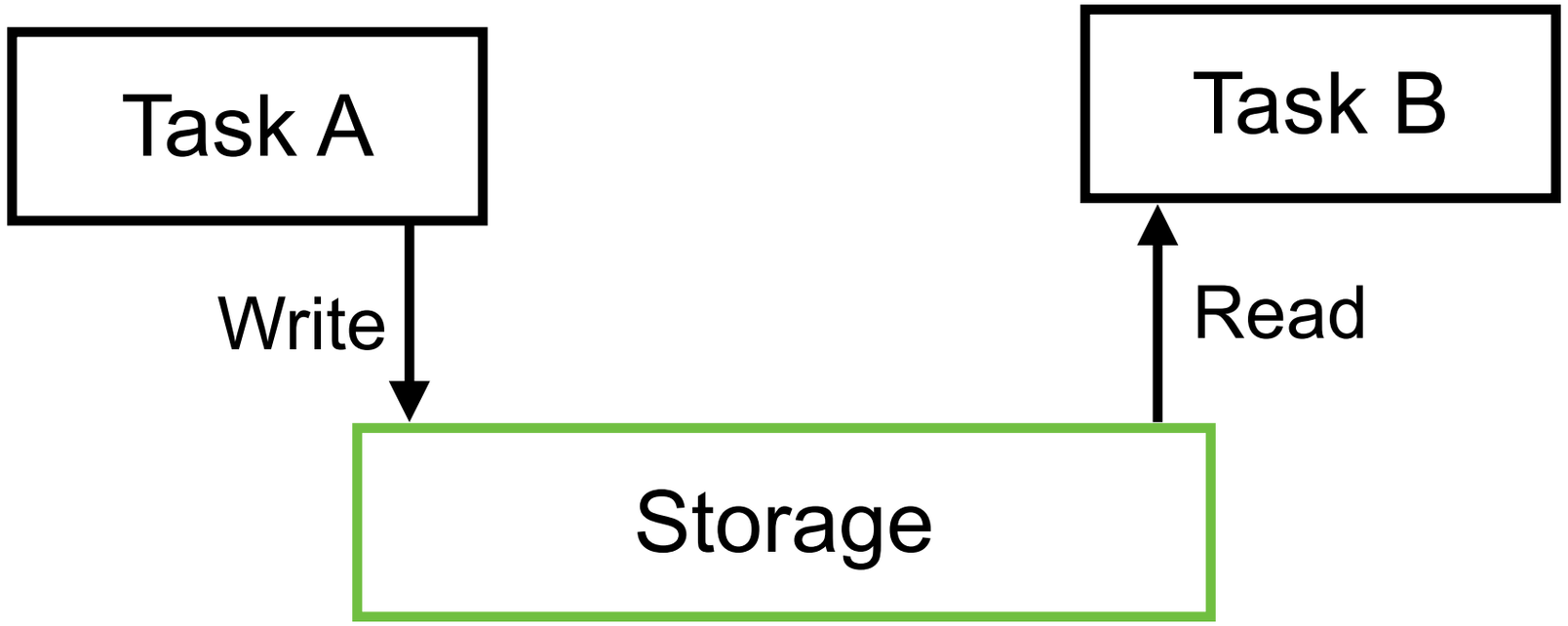
XCom (Cross Communication)

Tight coupling !!
Idempotency !!



Use intermediary data storage

Write data into file
Save data in database
SEnd data to the messaging server



Airflow Sensor

https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/sensors.html

Airflow Sensor

Waiting for a file to exist
Waiting for a certain amount of time to pass
Waiting for task in other DAG to finish

Poke mode

Reschedule

Designed for Event-Driven

Types of Sensor

File

Time

External task

HTTP

SQL

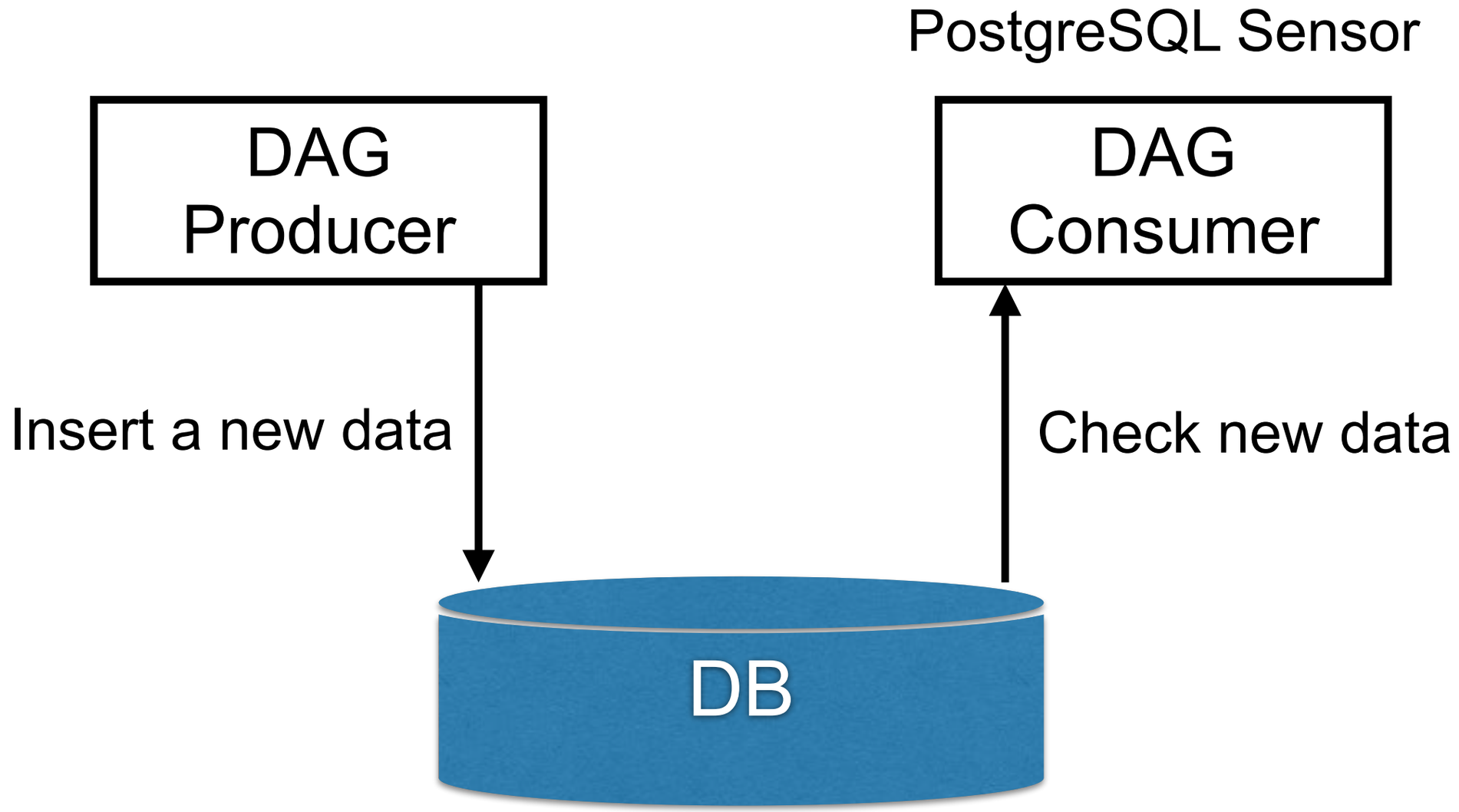
Customize from BaseSensorOperator class

Workshop

PostgreSQL Sensor

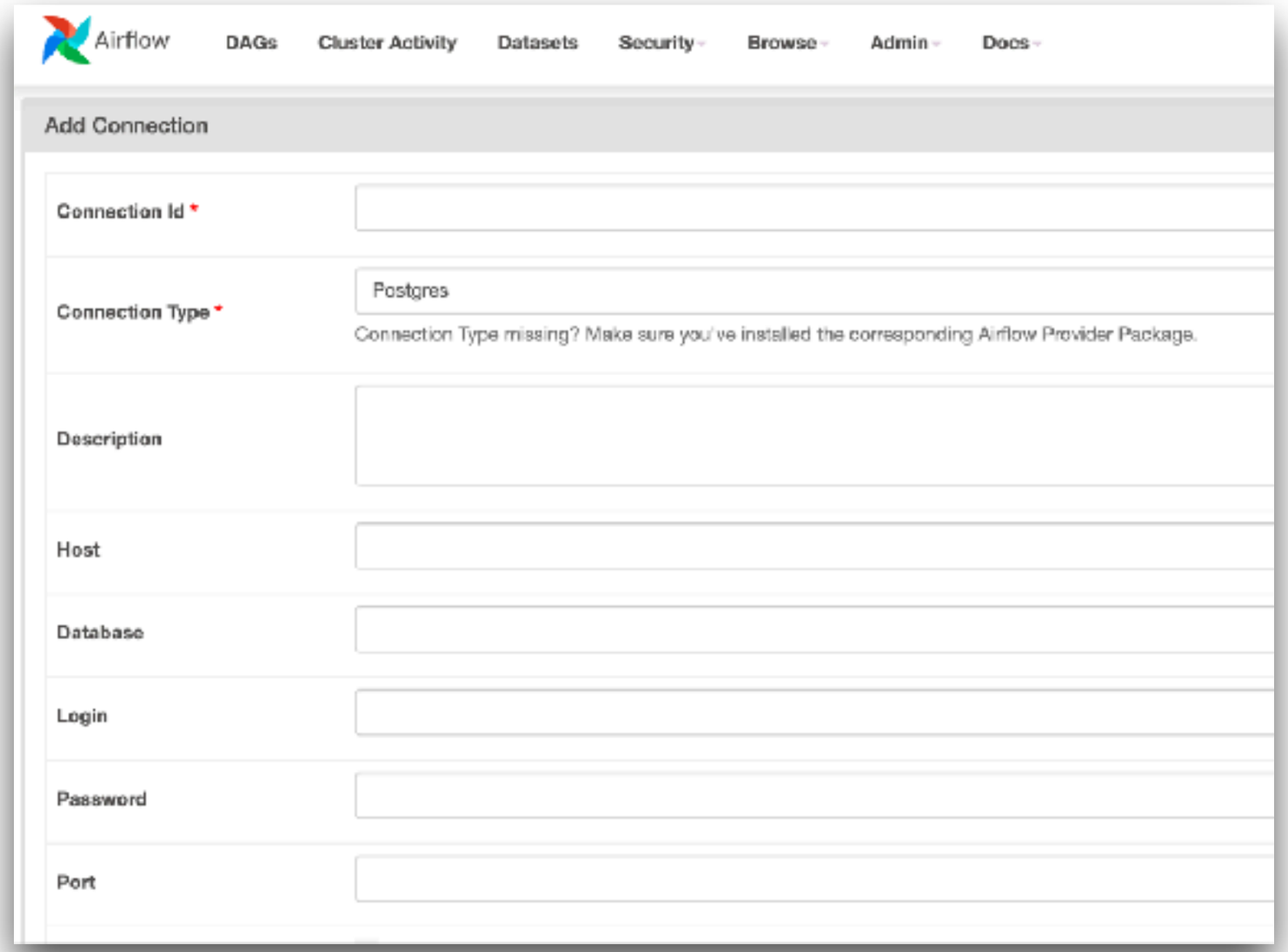
https://github.com/up1/workshop-apache-airflow/wiki/Working-with-Sensor

Workflows



Create connections

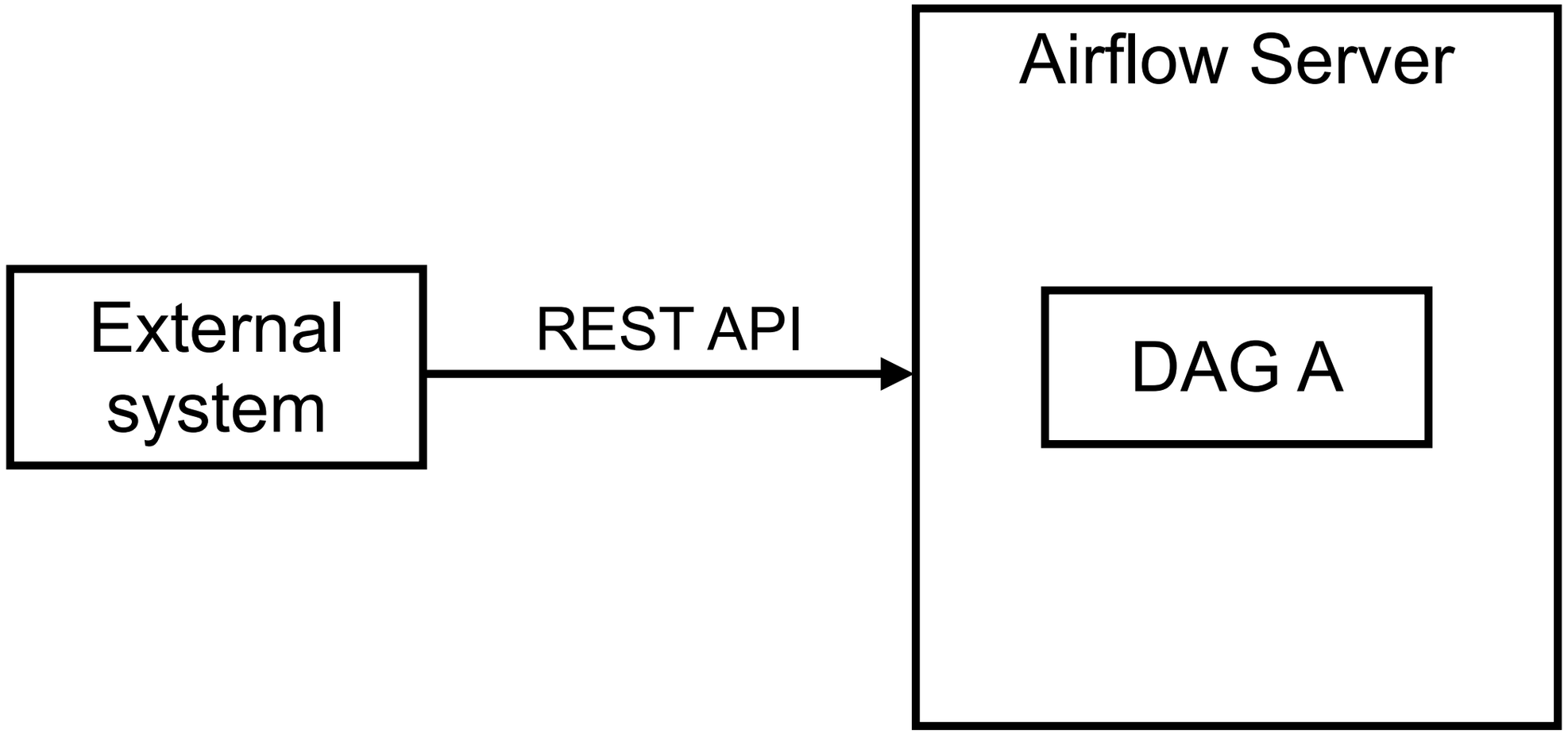
Goto menu **Admin -> connections**



Trigger DAG via APIs

https://airflow.apache.org/docs/apache-airflow/stable/stable-rest-api-ref.html

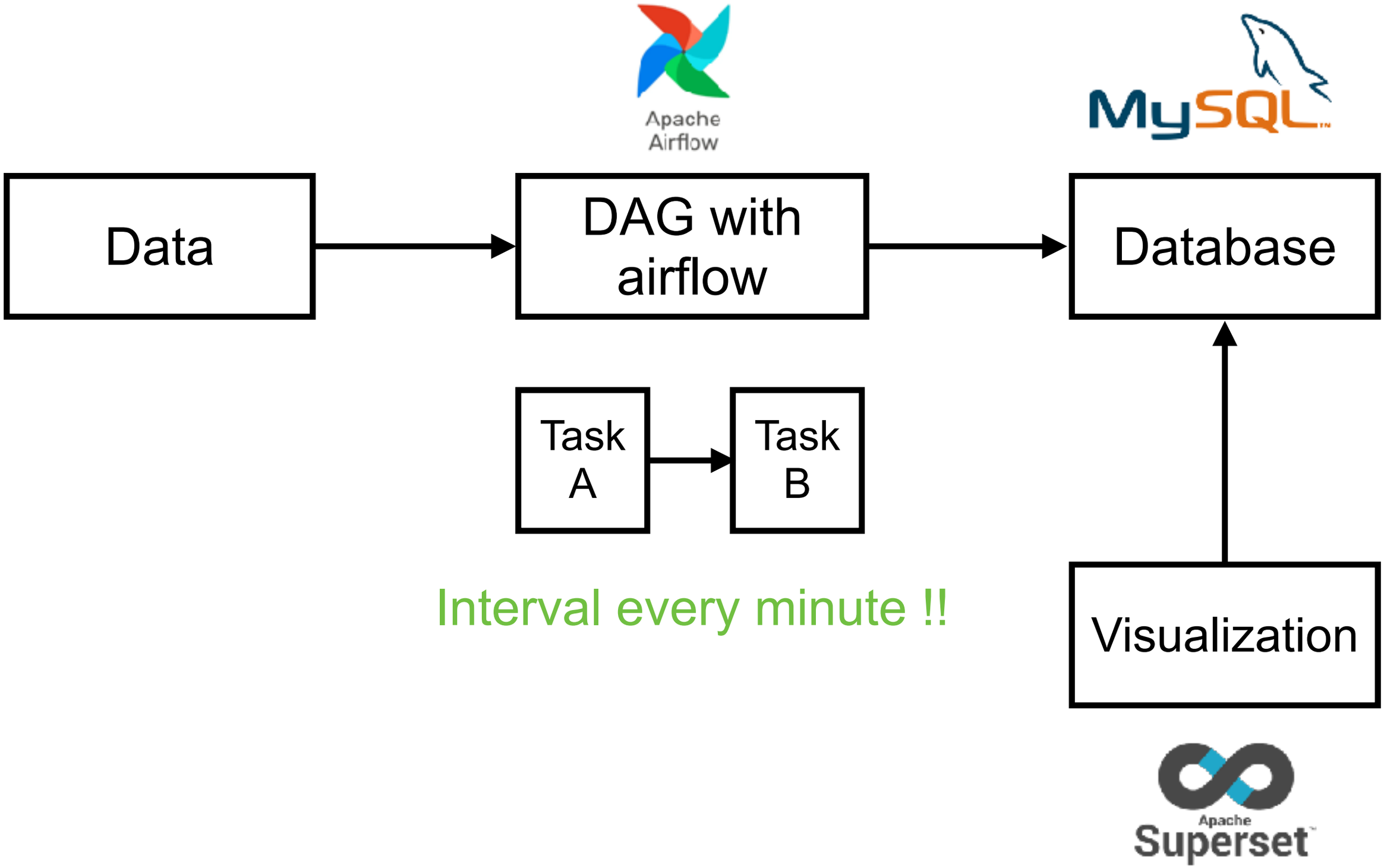
Trigger DAG via APIs



https://github.com/up1/workshop-apache-airflow/wiki/Trigger-DAG-from-External-System-via-REST-API

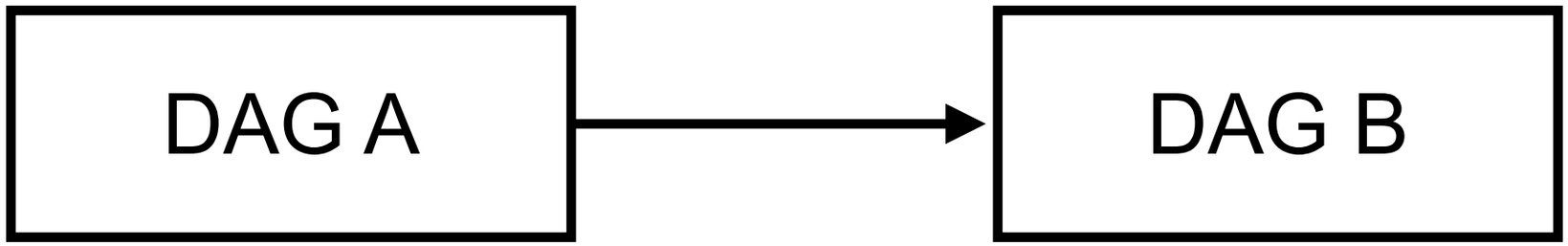
Workshop ETL

Workshop ETL



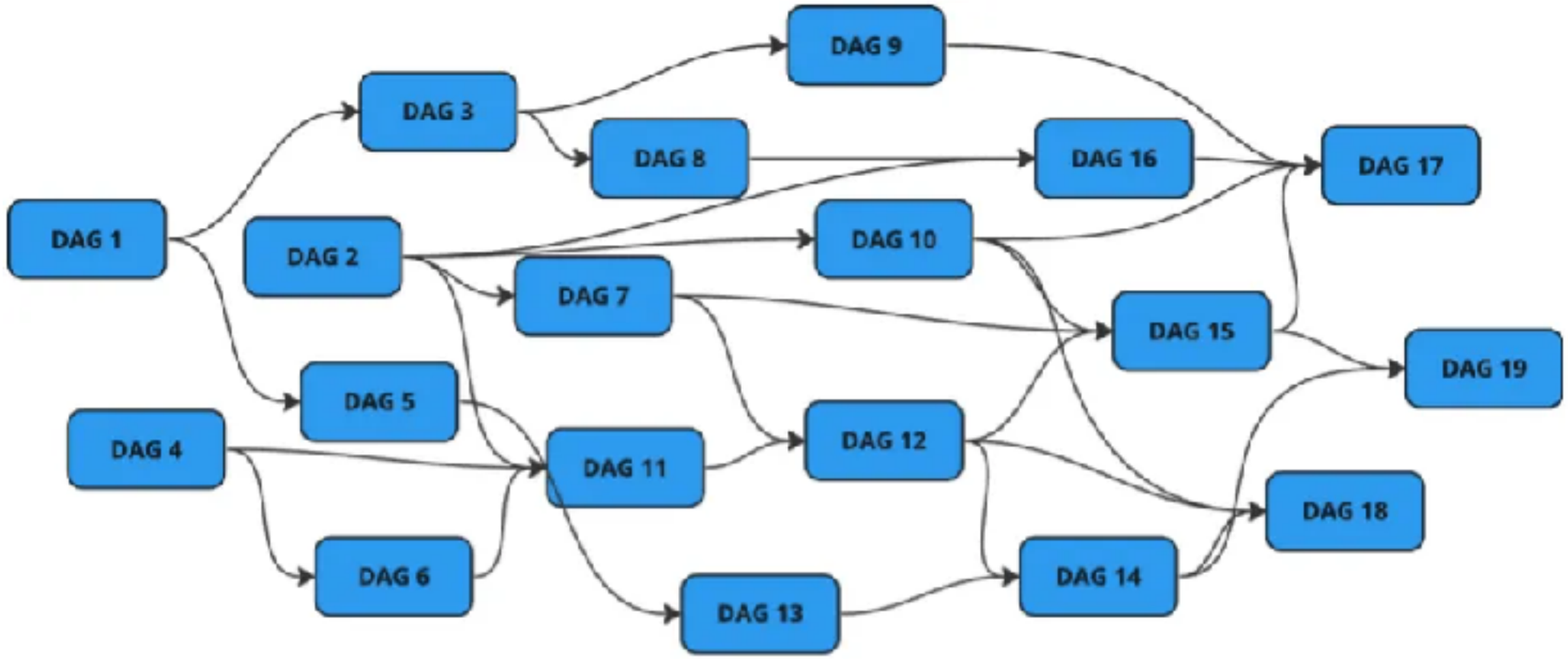
Interval every minute !!

Cross DAG !!



https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/external_task_sensor.html

Cross DAG !!



Unmanaged Cross DAG !!

Data inconsistency
Inefficiency resource usage
Bugs

How to manage Cross DAG ?

Managed Cross DAG

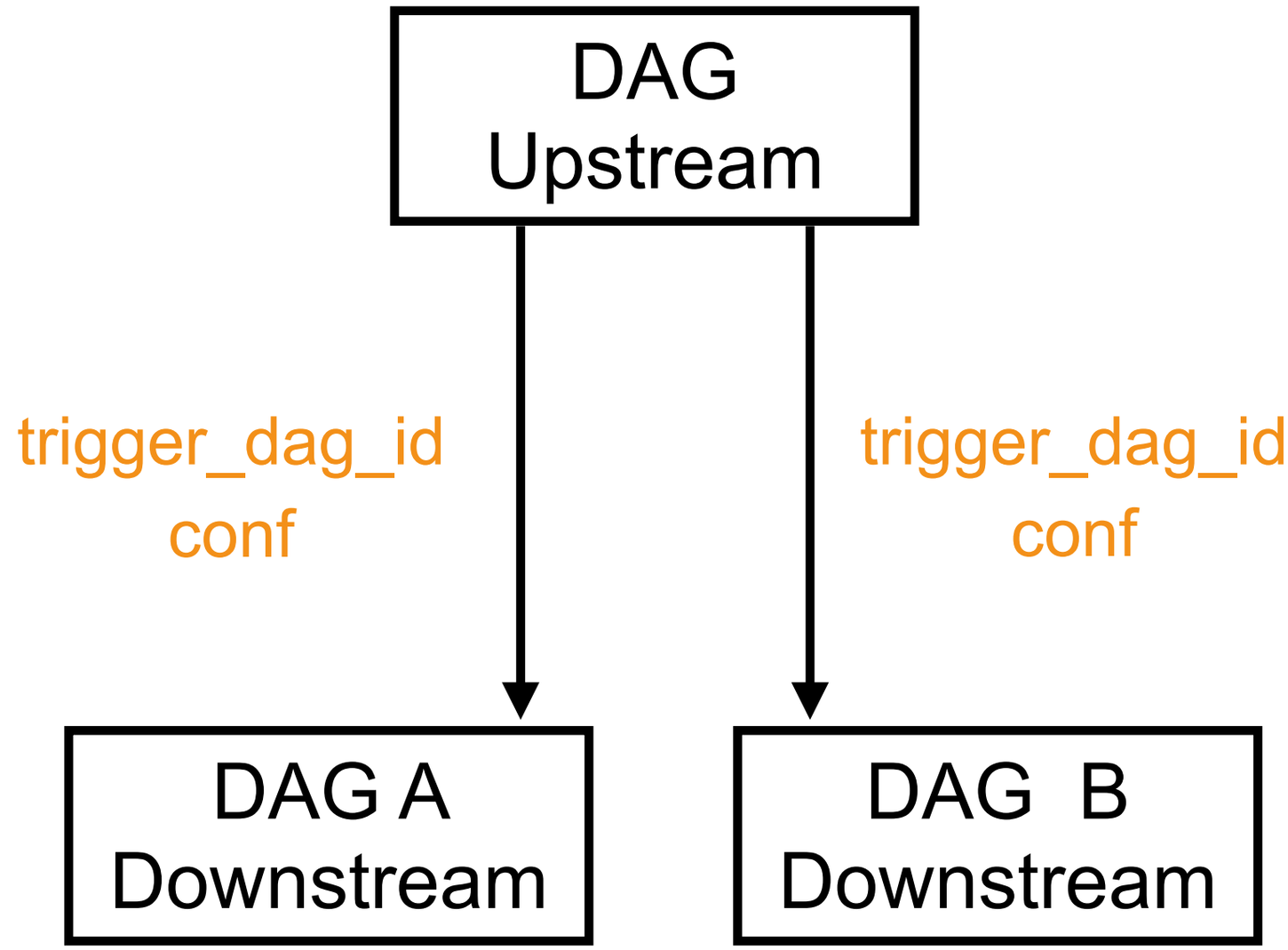
Trigger
Sensor
Dataset
API

Managed Cross DAG

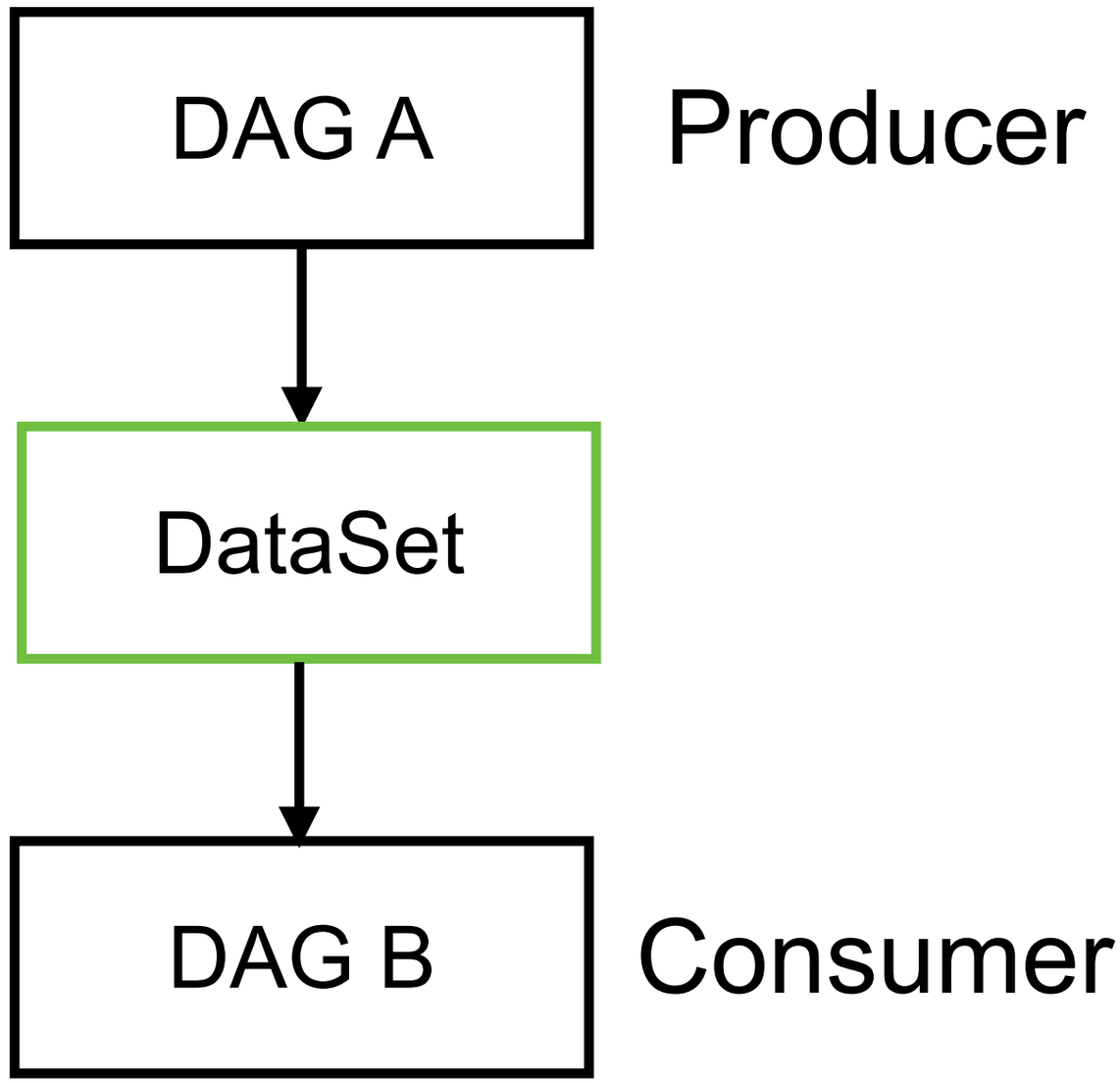
Coupling method	Trigger	Sensor	Dataset	API
Functionality	Upstream triggers downstream	Downstream waits on upstream	Upstream updates dataset and downstream waits on dataset update	Upstream triggers downstream
Coupling principle	Time-driven	Time-driven	Event-driven	Time-driven
Operator	TriggerDagRunOperator	ExternalTaskSensor	/	SimpleHttpOperator
Dependency scope	Task → DAG	Task → Task, Task Group, DAG	Task → DAG	Task → DAG
Usage	1 → Multiple	Multiple → 1	Multiple → Multiple (Non-regular dataset updates)	DAGs in different environments
Coupling time flexibility	Downstream starts immediately after upstream finished	Possibility to add delay between upstream and downstream	Downstream starts immediately after updates finished	Possibility to add delay between upstream and downstream
Minimum required version	1.10	1.10	⚠️ 2.4 ⚠️	1.10
Visualization	Airflow UI	Airflow UI	Airflow UI	/
Data lineage	No	No	Yes	No
Implementation complexity	Medium	Medium	Low	Medium

https://medium.com/datamindedbe/cross-dag-dependencies-in-apache-airflow-a-comprehensive-guide-88cbc0bc68d0

Trigger



Dataset (Event-driven)



https://airflow.apache.org/docs/apache-airflow/stable/authoring-and-scheduling/datasets.html

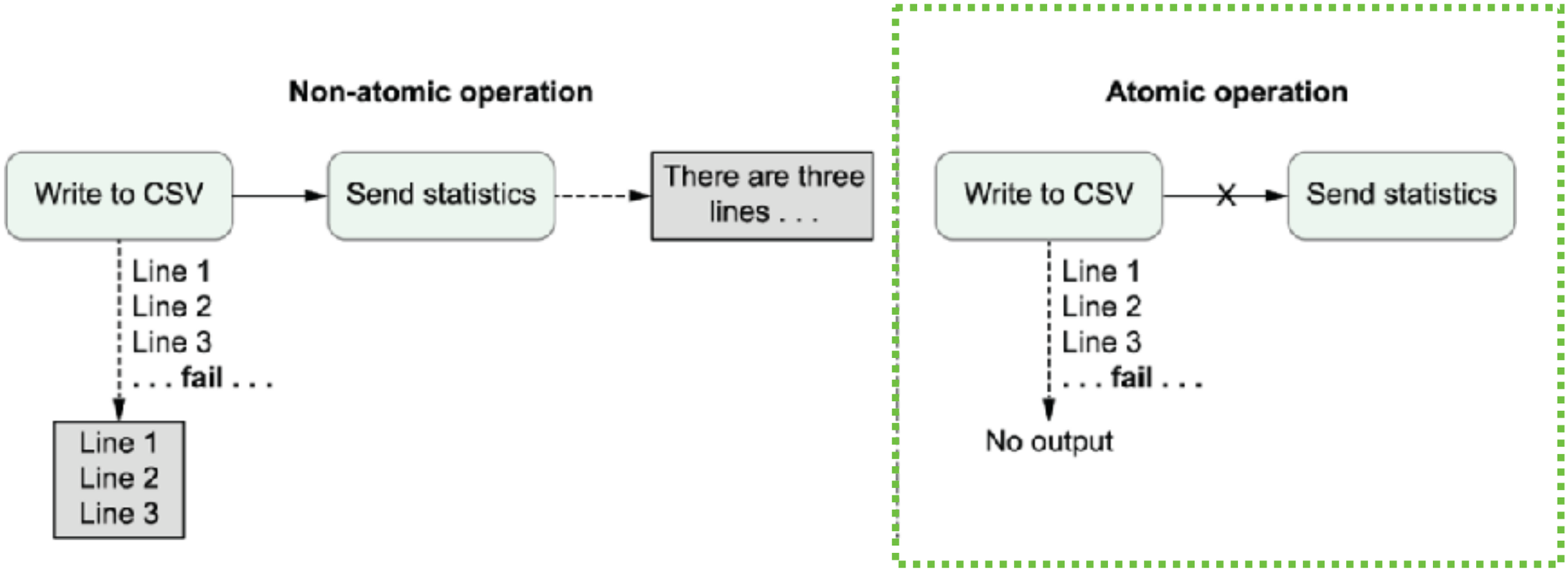
Design Better Tasks

Design Better Tasks

Atomicity
Idempotency

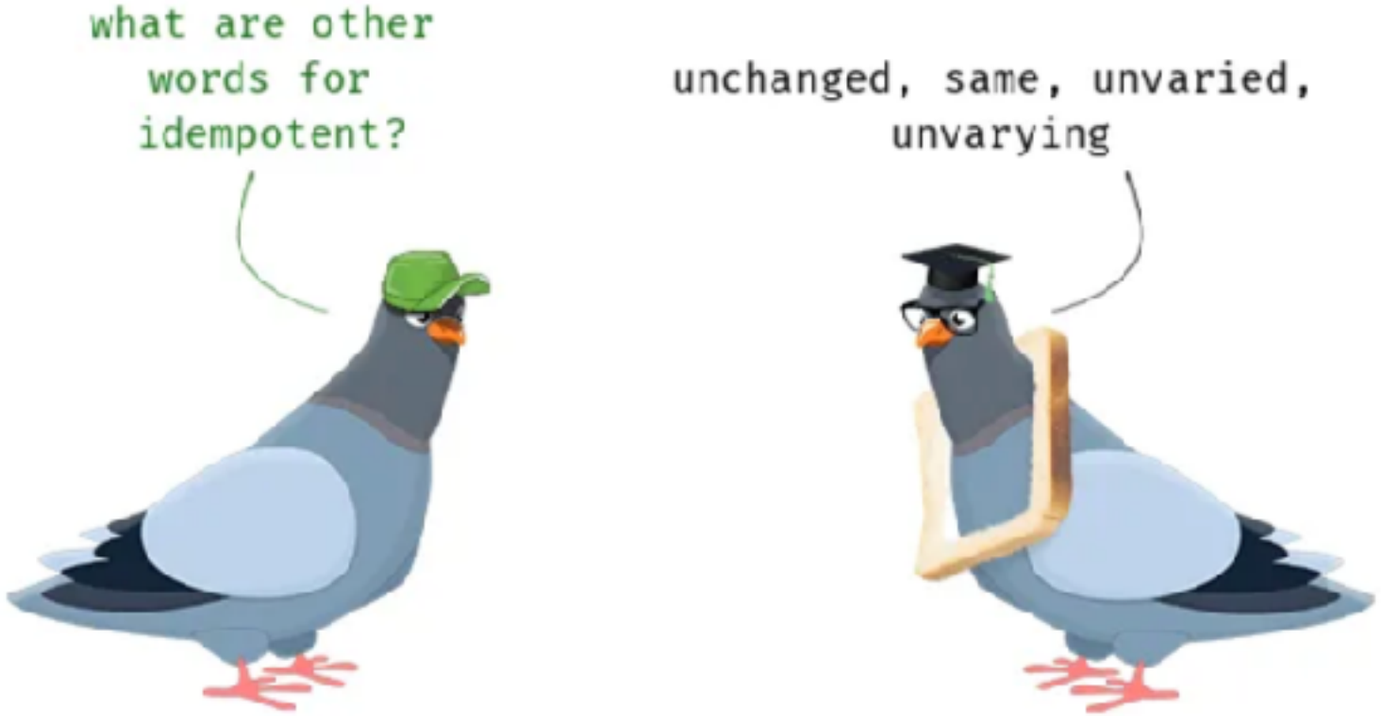
Atomicity

Tasks should be defined so that they either succeed and produce some proper result or fail in a manner that does not affect the state of the system

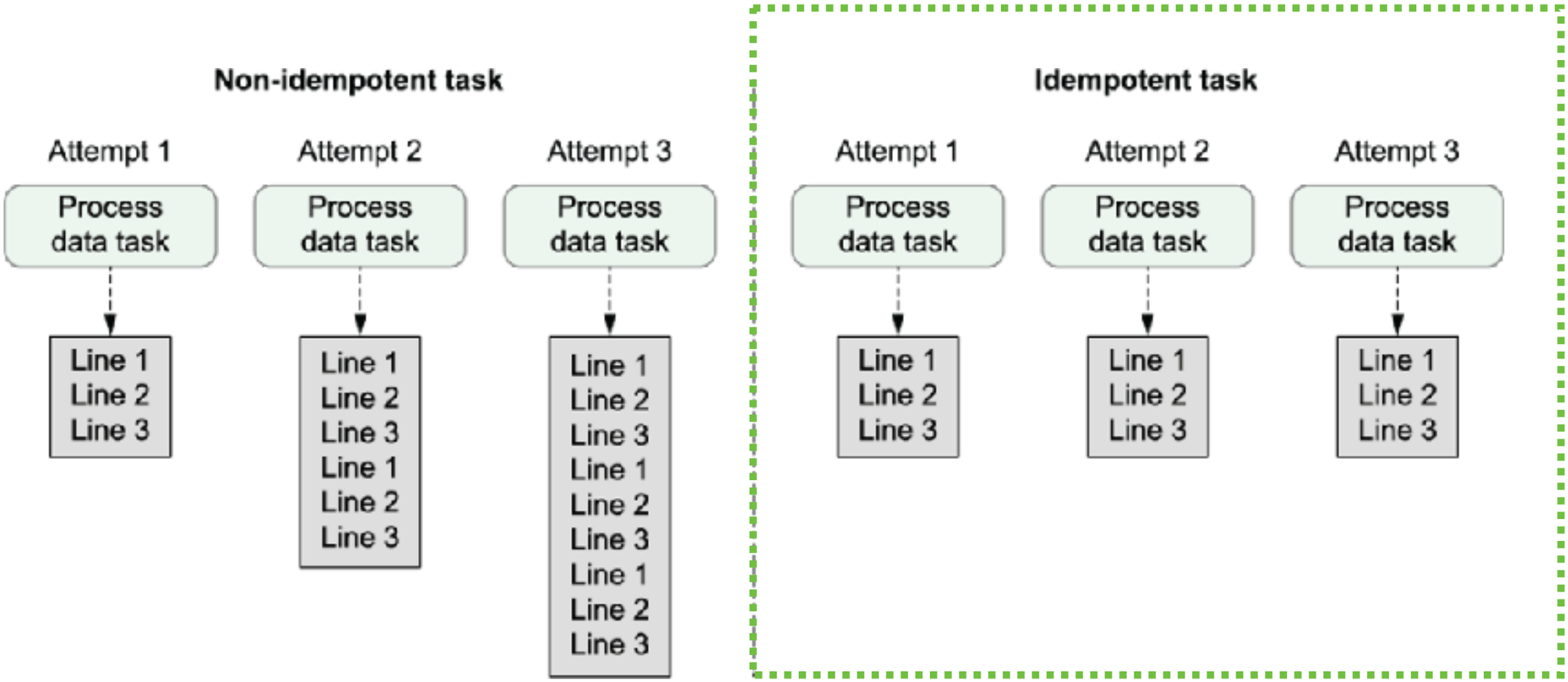


Idempotency

Idempotent Data pipeline is the pipeline that **always return the same result**, no matter when or how many times it runs.



Idempotency



Airflow 3.0 is coming

Anywhere

At any time

In any language