

Apache Airflow workshop





Somkiat Puisungnoen

Search

Somkiat | Home

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามช่างนาฏกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata



somkiat.cc

Page Messages Notifications 3 Insights Publishing Tools Settings Help ▾

Help people take action on this Page. X

+ Add a Button

Home

Posts

Videos

Photos



**[https://github.com/up1/
workshop-apache-airflow](https://github.com/up1/workshop-apache-airflow)**



Topics

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



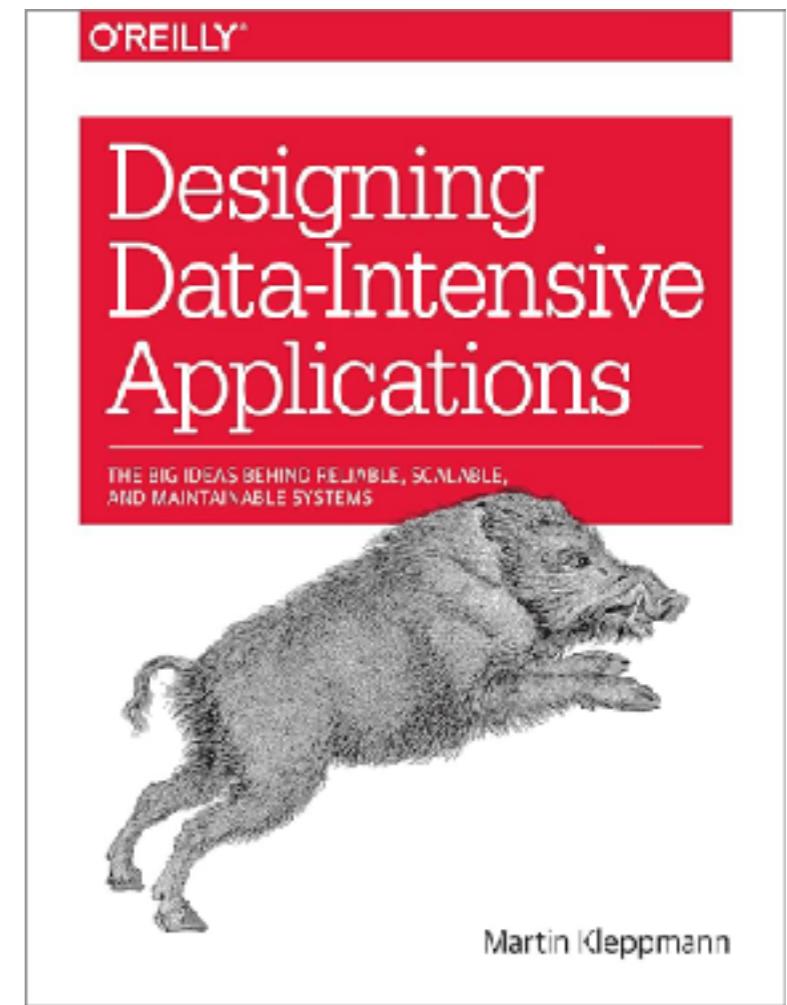
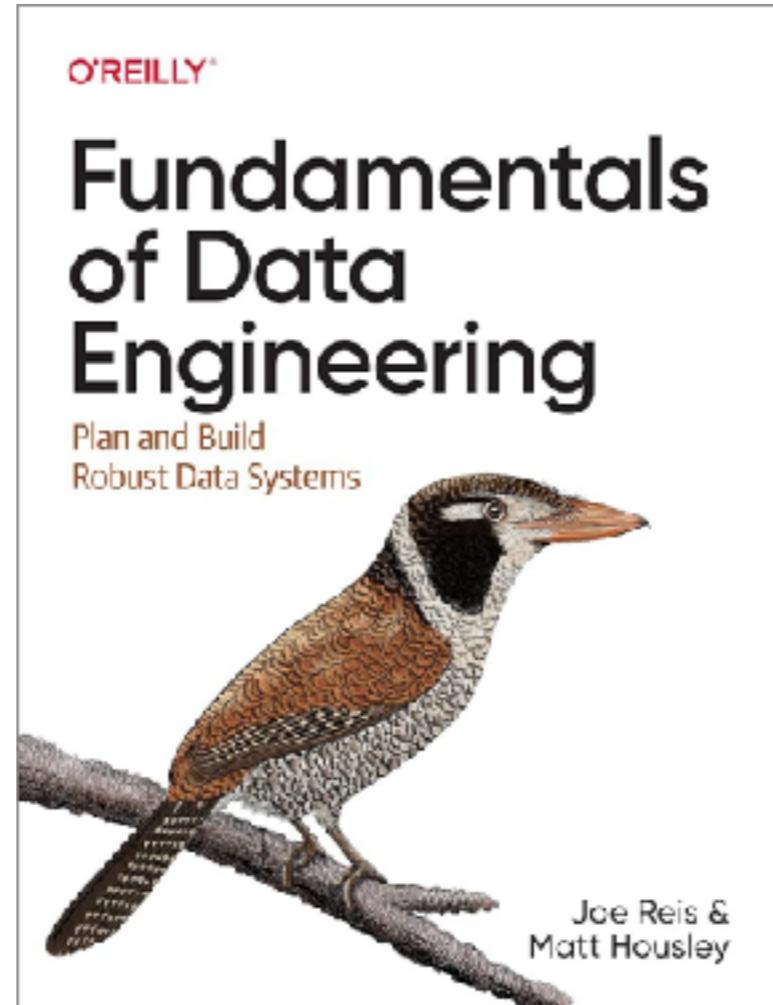
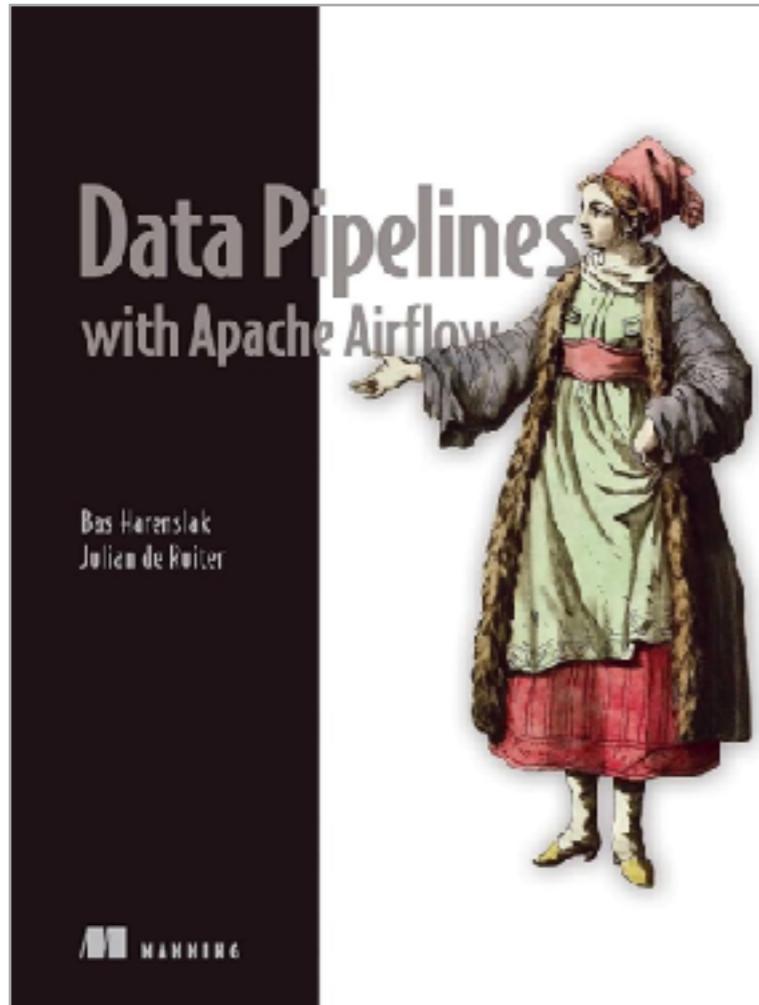


Apache
Airflow

AirBnB + Workflow



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

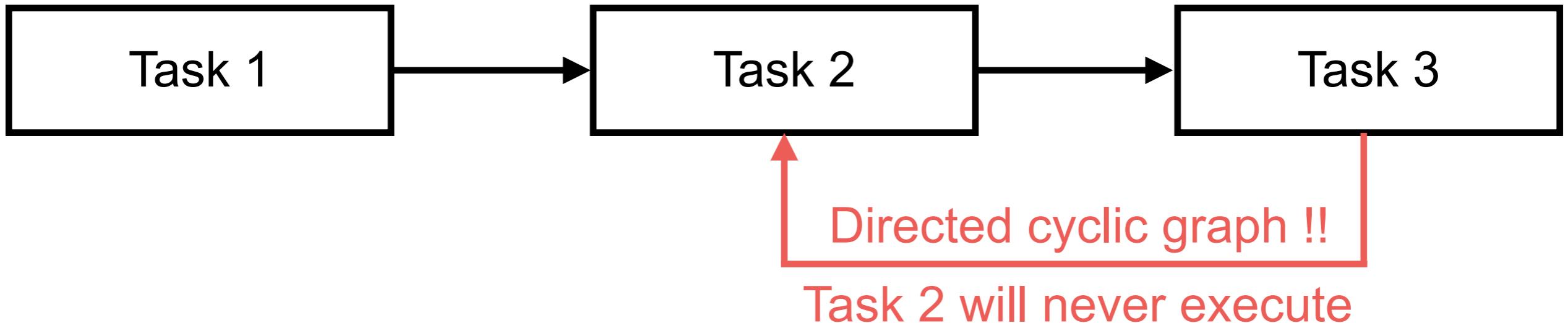


Tasks are represented as nodes in graph



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

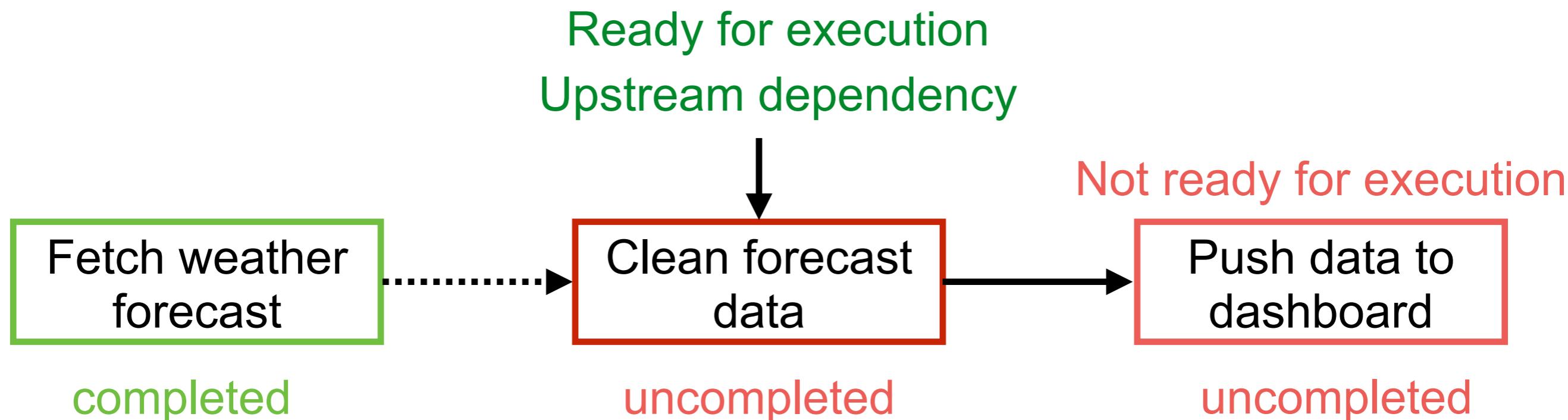
Task ready for execution

No dependency



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

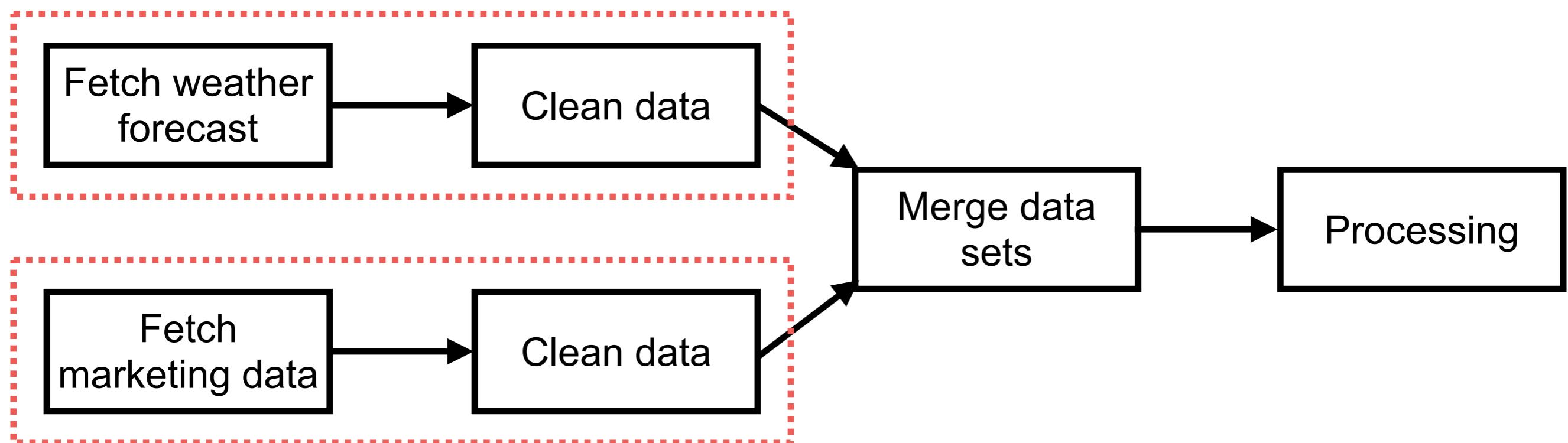


More use cases



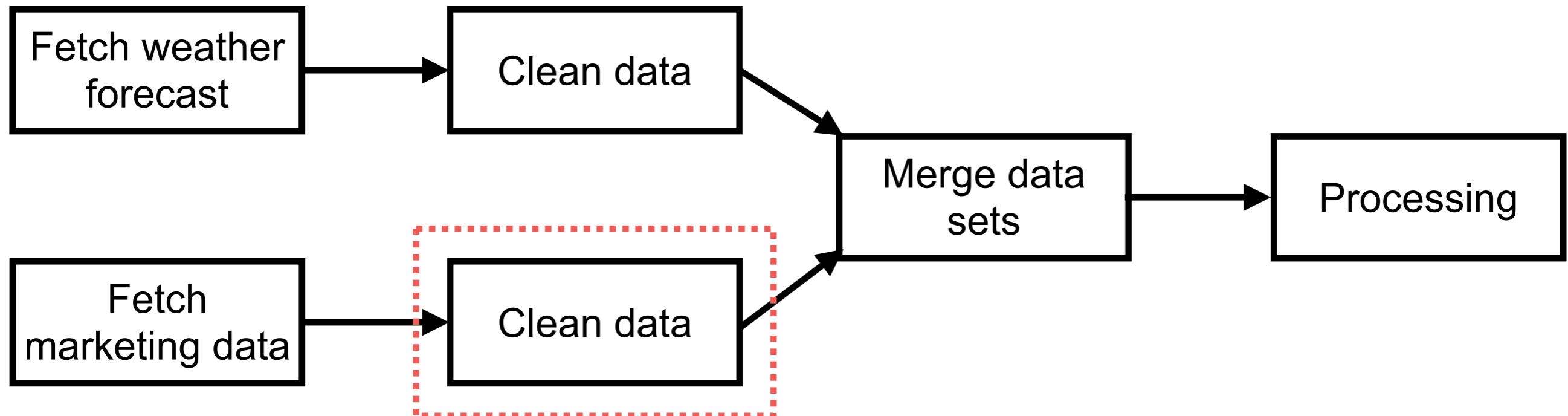
Pipeline graph

Execute each task in parallel process



Pipeline graph

When task failed ?



Rerun only this task



Tools



FIRSTMARK



MAD

LANDSCAPE

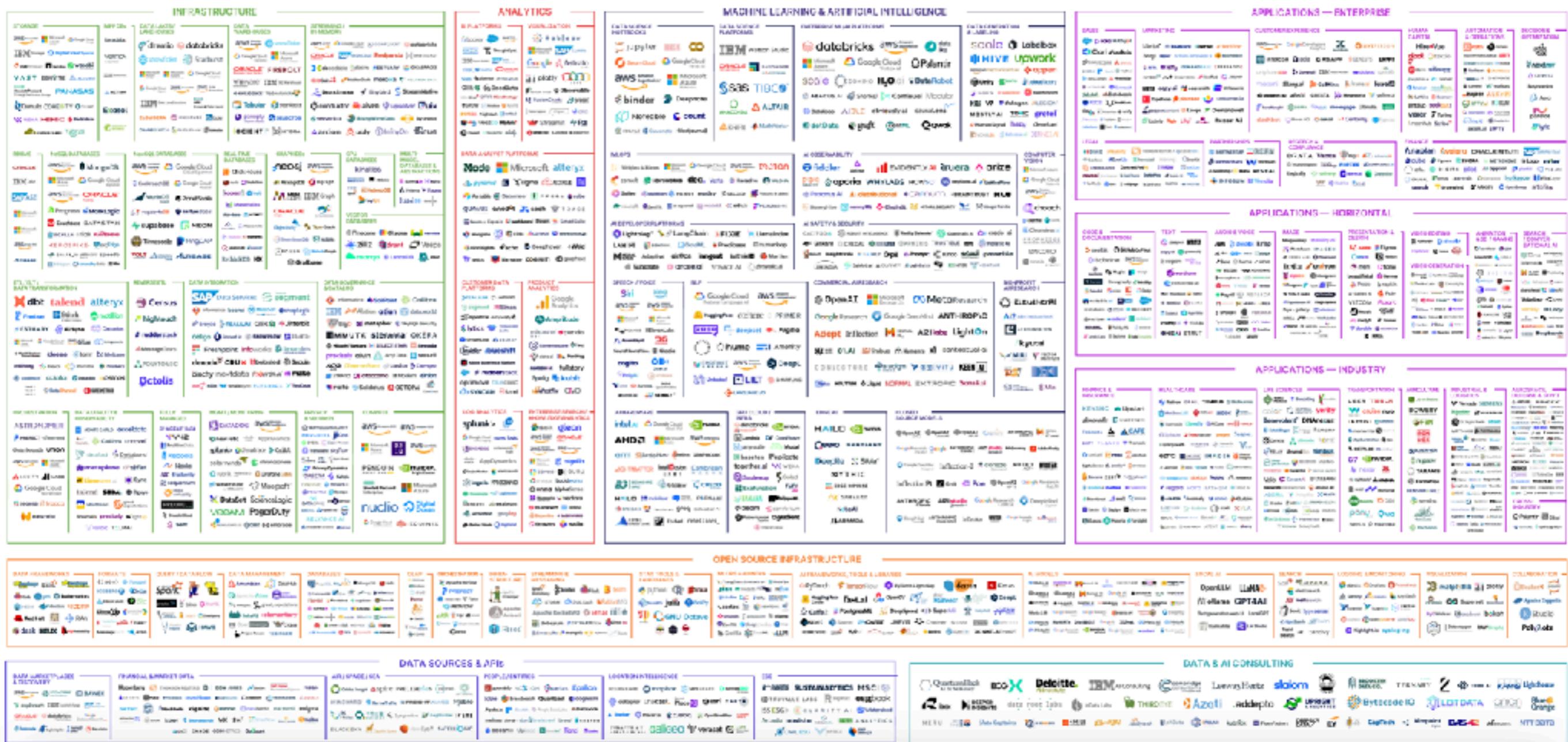
MACHINE LEARNING, AI & DATA

2024

<https://mattturck.com/mad2024/>

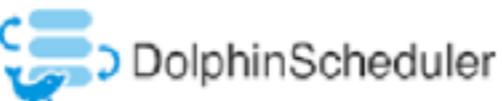


THE 2024 MAD (MACHINE LEARNING, ARTIFICIAL INTELLIGENCE & DATA) LANDSCAPE



Apache Airflow !!

ORCHESTRATION



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



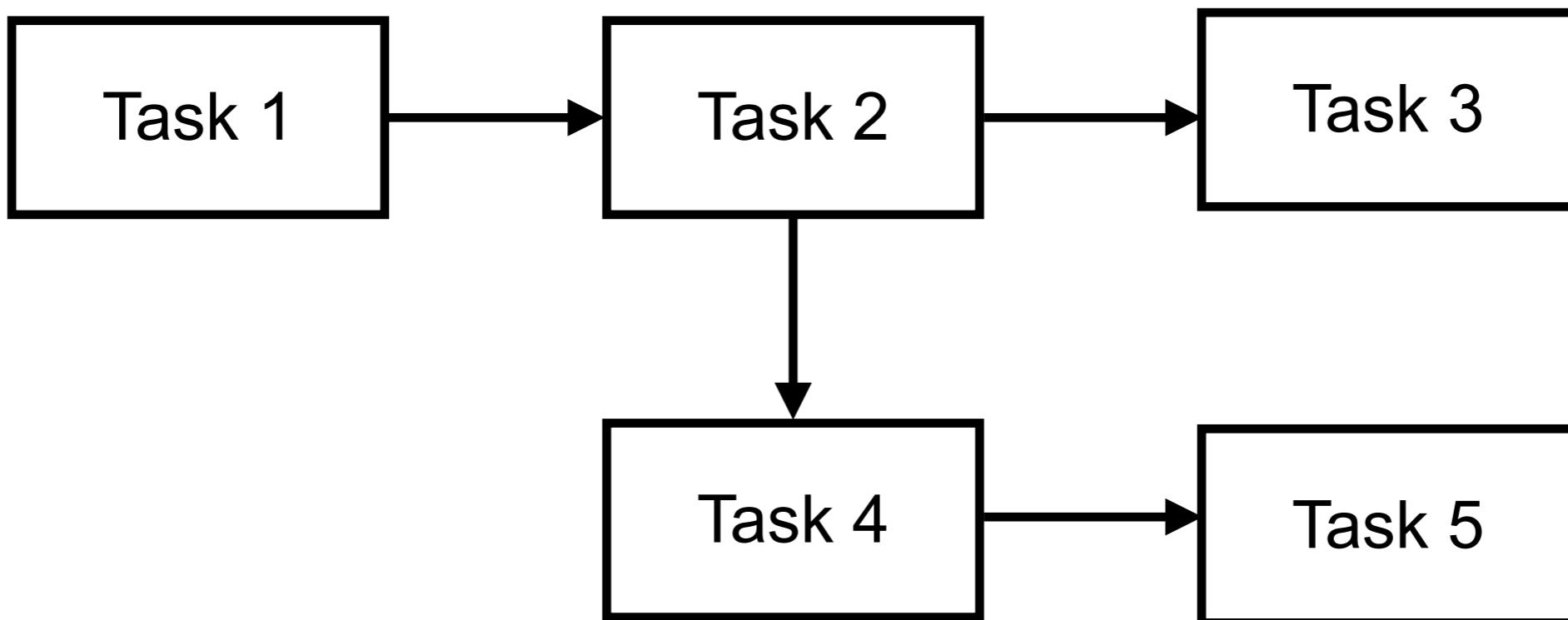
What Airflow ?

Starts in Airbnb 2014

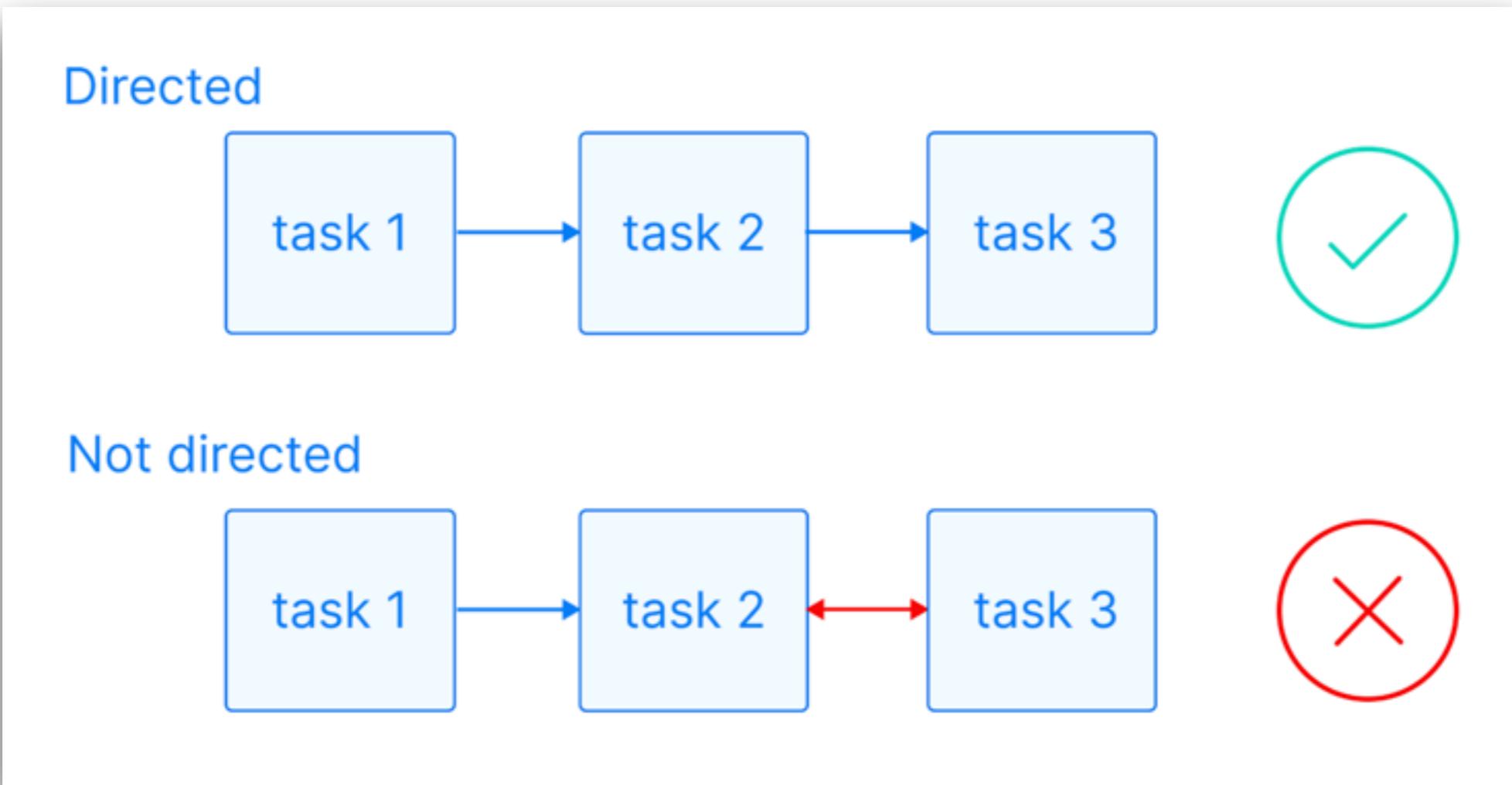
Manage complex **workflows**

Sequence of tasks (DAG)

Create workflow with **Python**



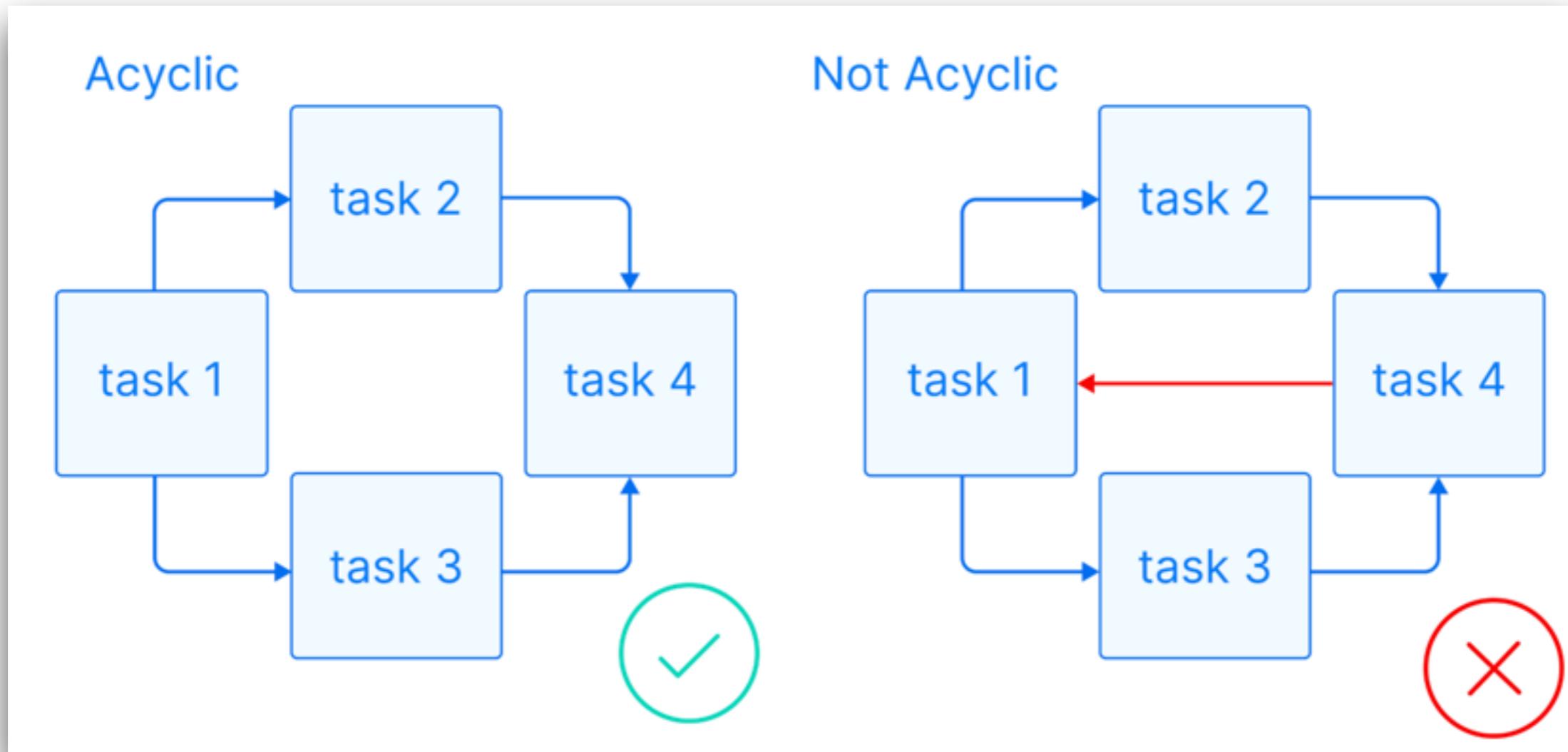
DAG (Directed Acyclic Graph)



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



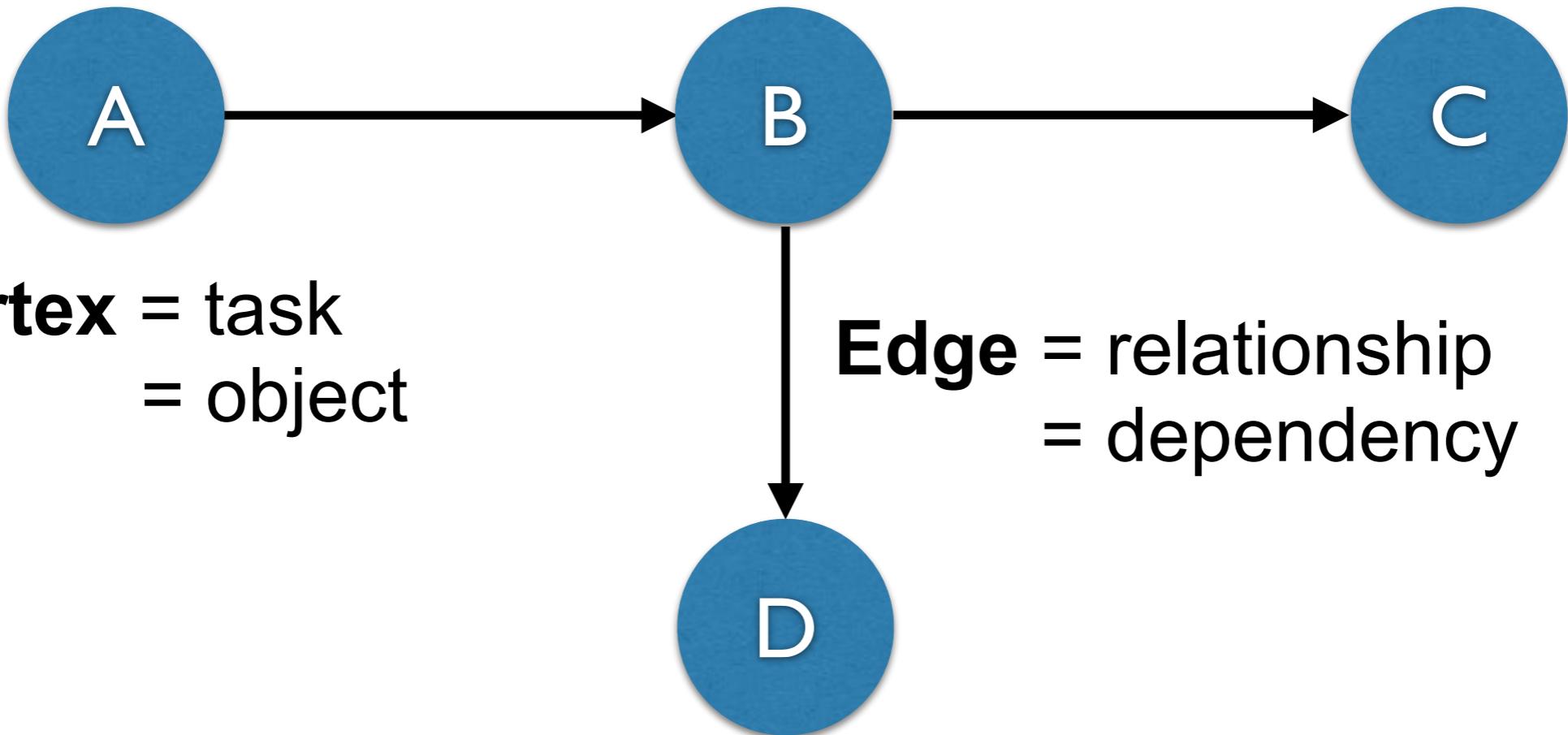
DAG (Directed Acyclic Graph)



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



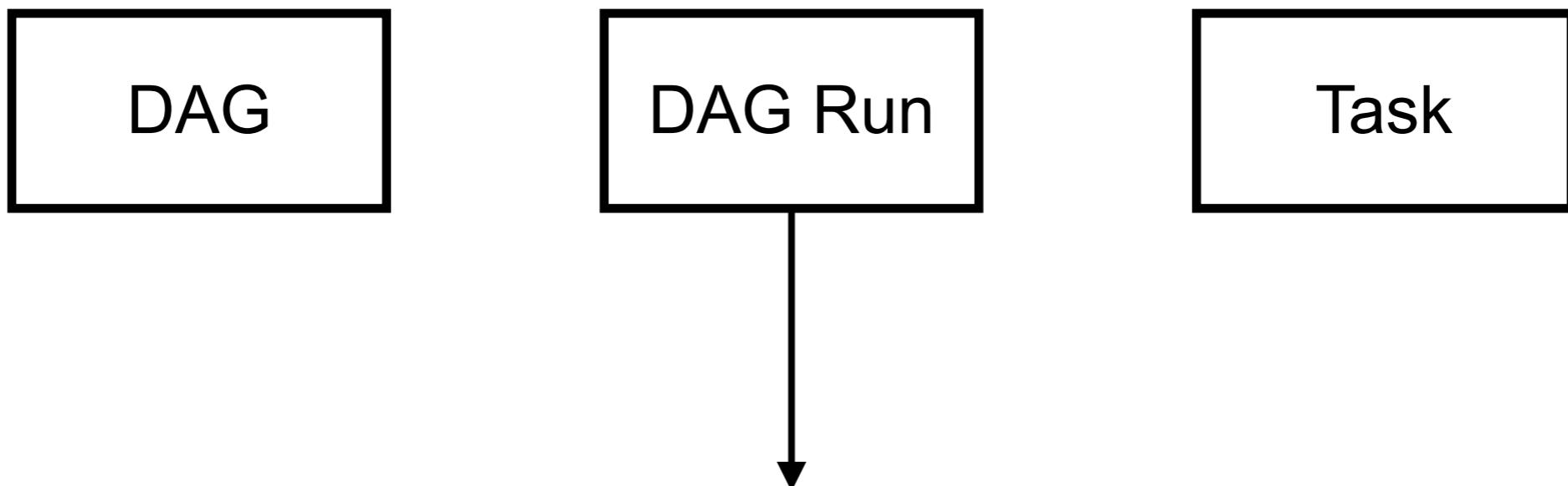
DAG (Directed Acyclic Graph)



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



Basic knowledges



Manual

Scheduled

Dataset trigger

Backfill



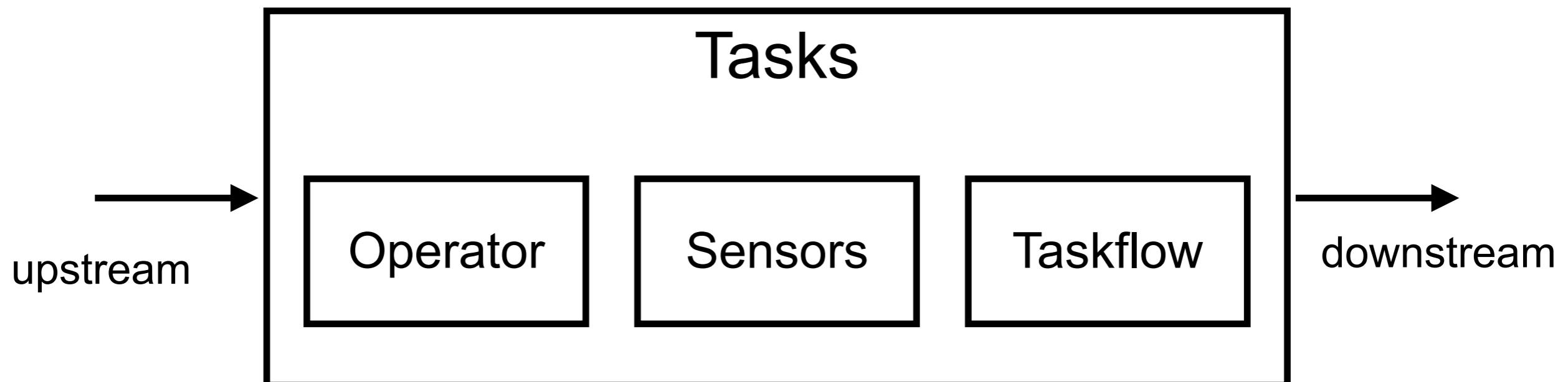
Tasks

Unit of work or execution in Airflow

Represent as node in DAG

Isolated from 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

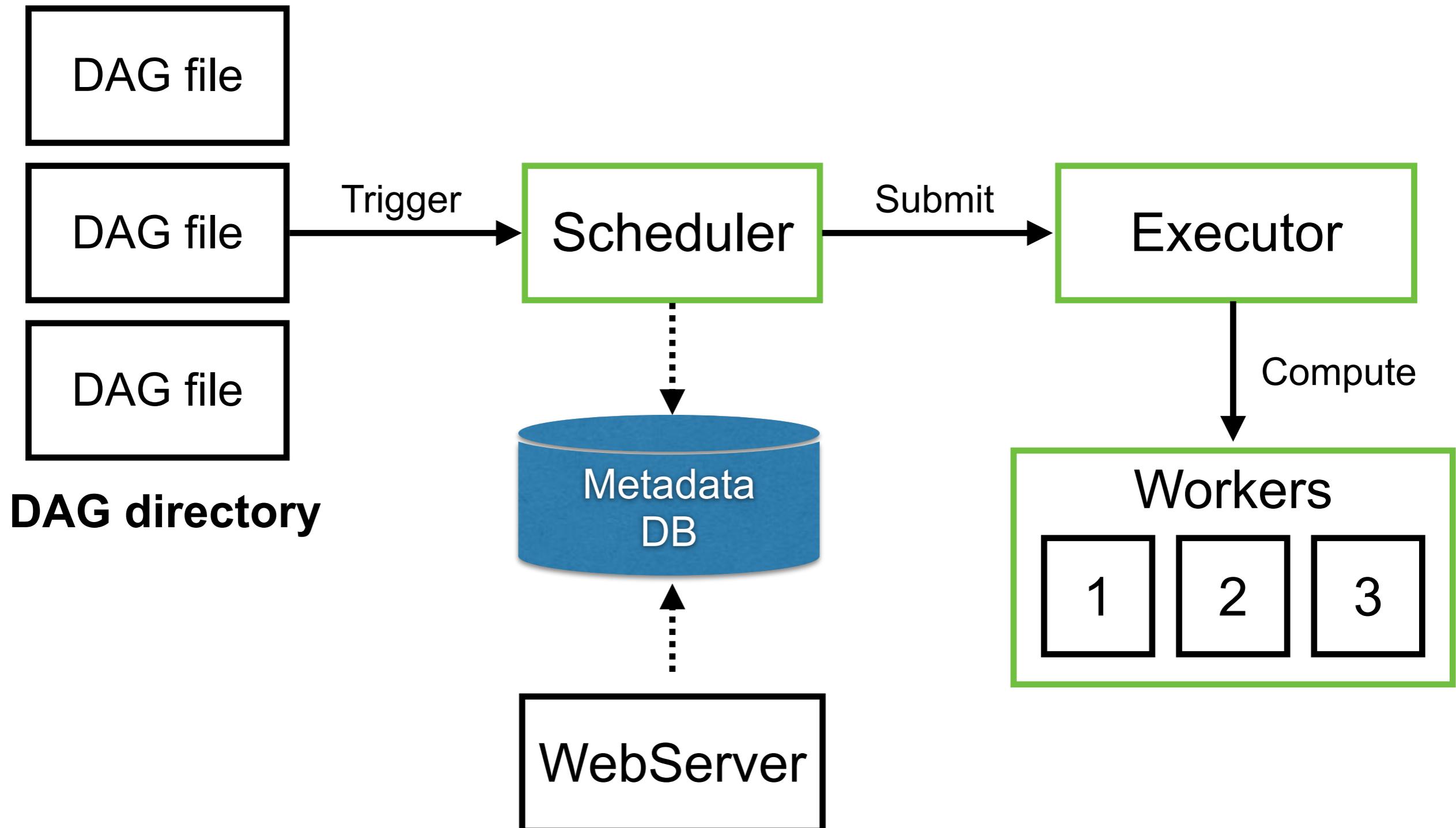


Architecture

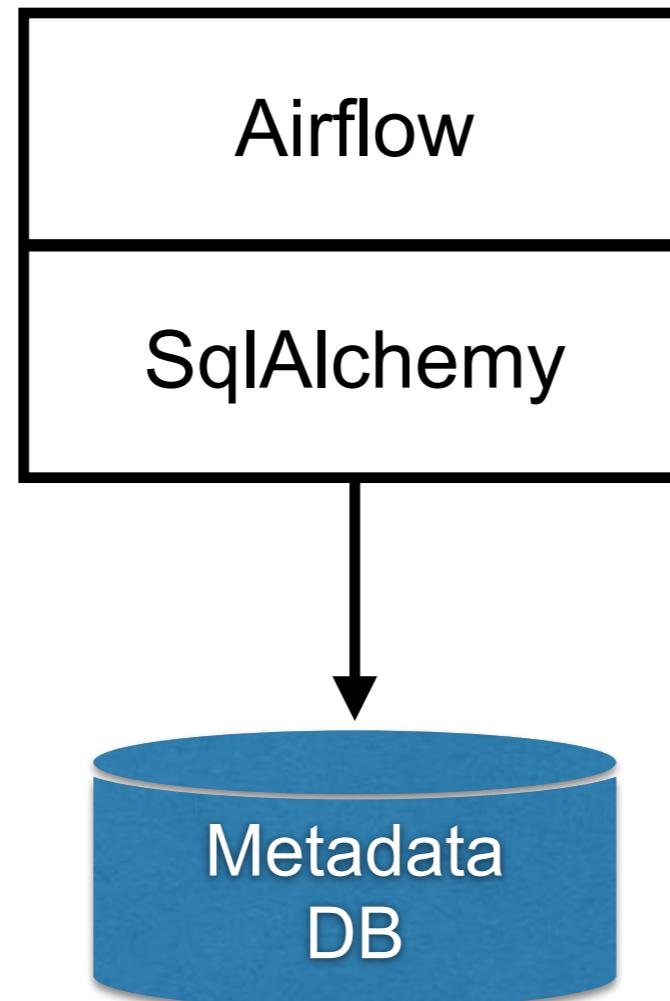
<https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html>



Airflow components



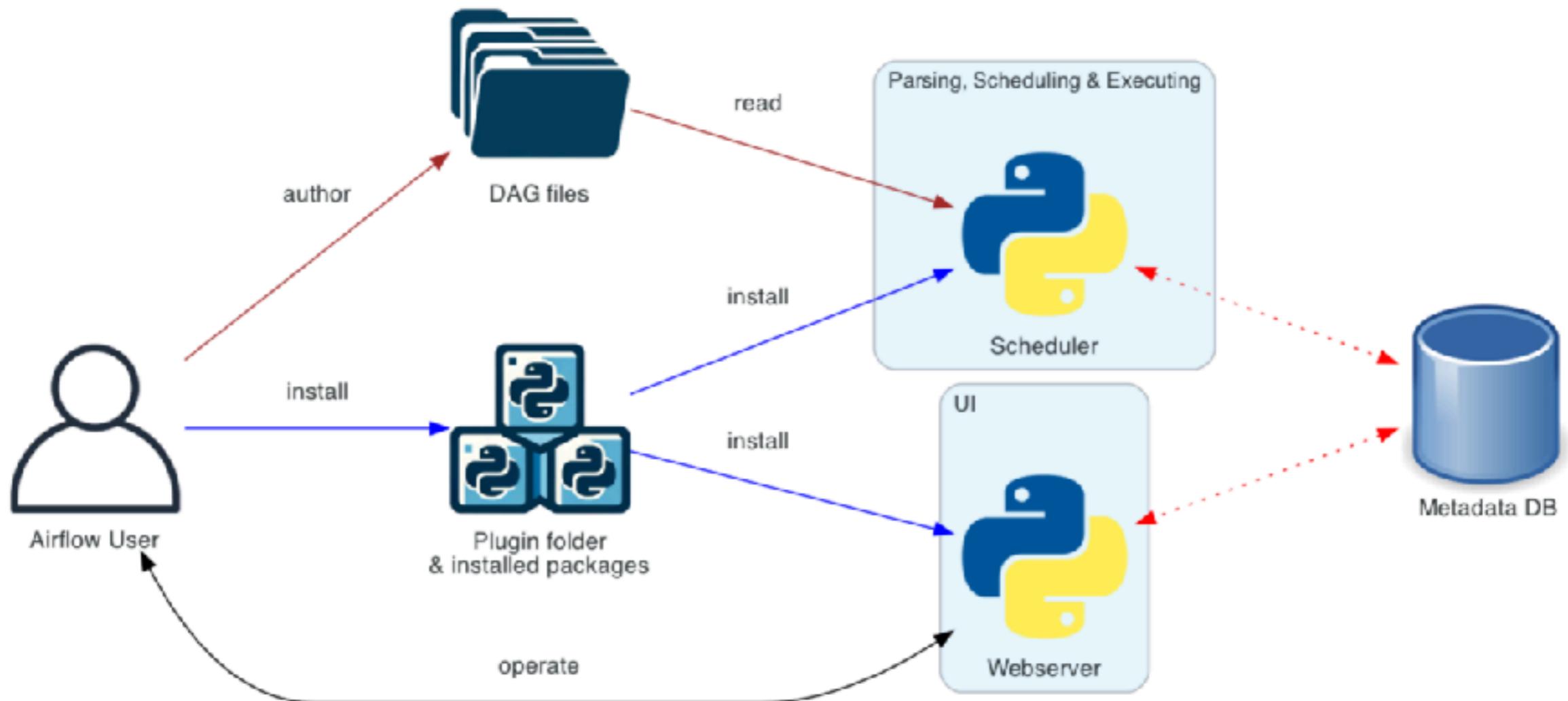
Database



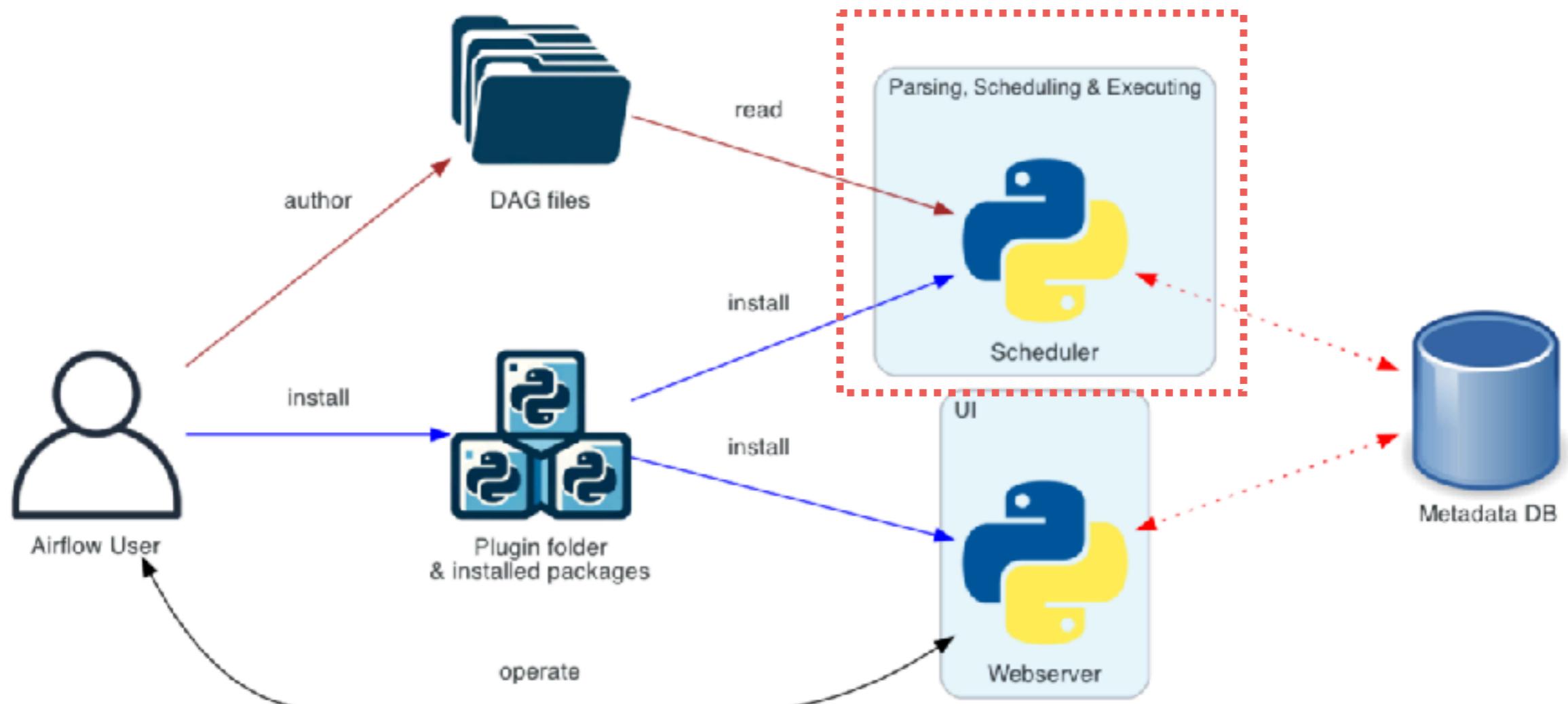
<https://airflow.apache.org/docs/apache-airflow/stable/howto/set-up-database.html>



Simple Airflow Deployment



Schedule and Executor



Types of Executor

Local

Remote

Sequential (default)

Local

Celery

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

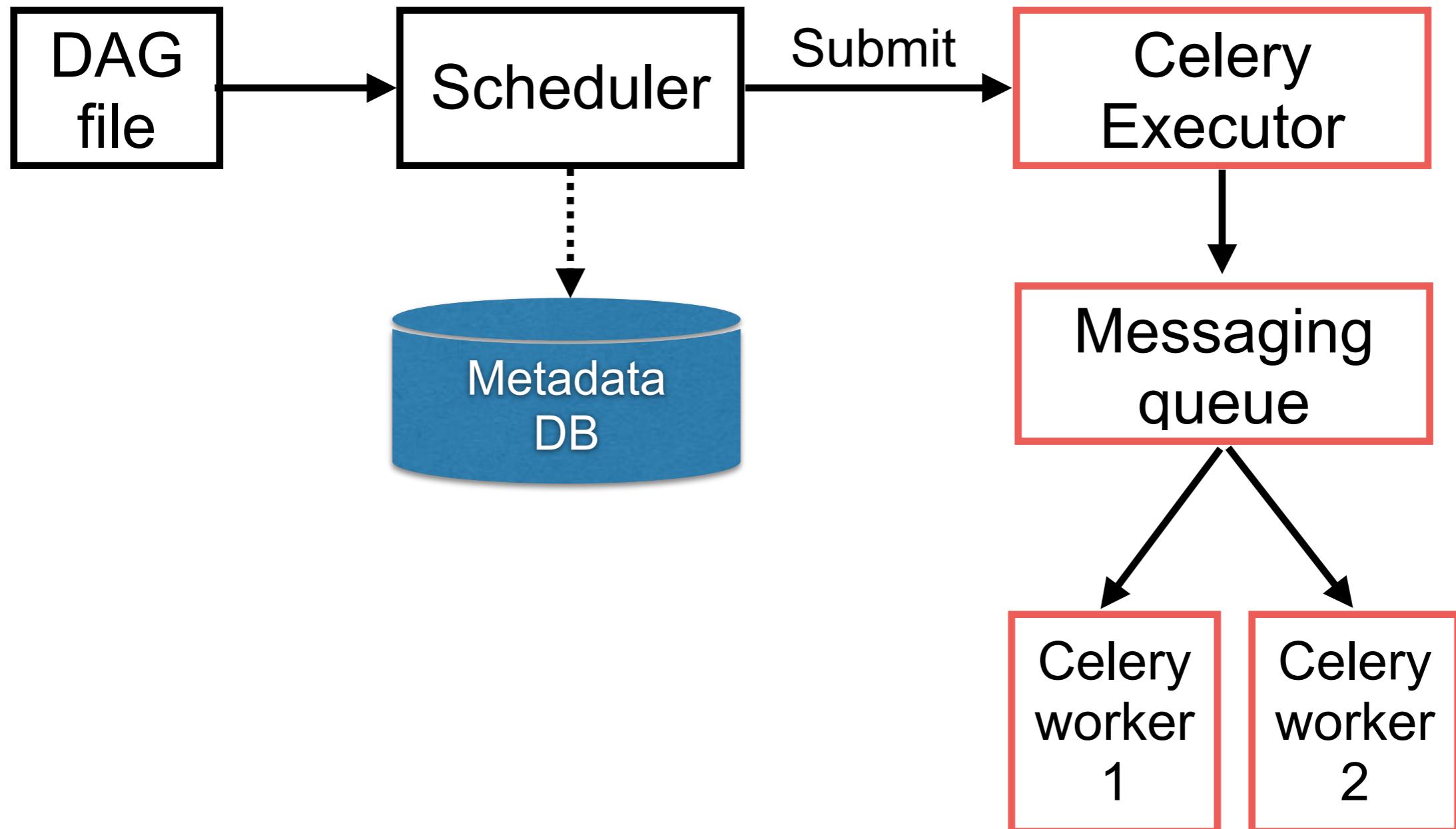
Easy to setup

Good for small workflow

Not scalable, slow, single point of failure



Celery executor



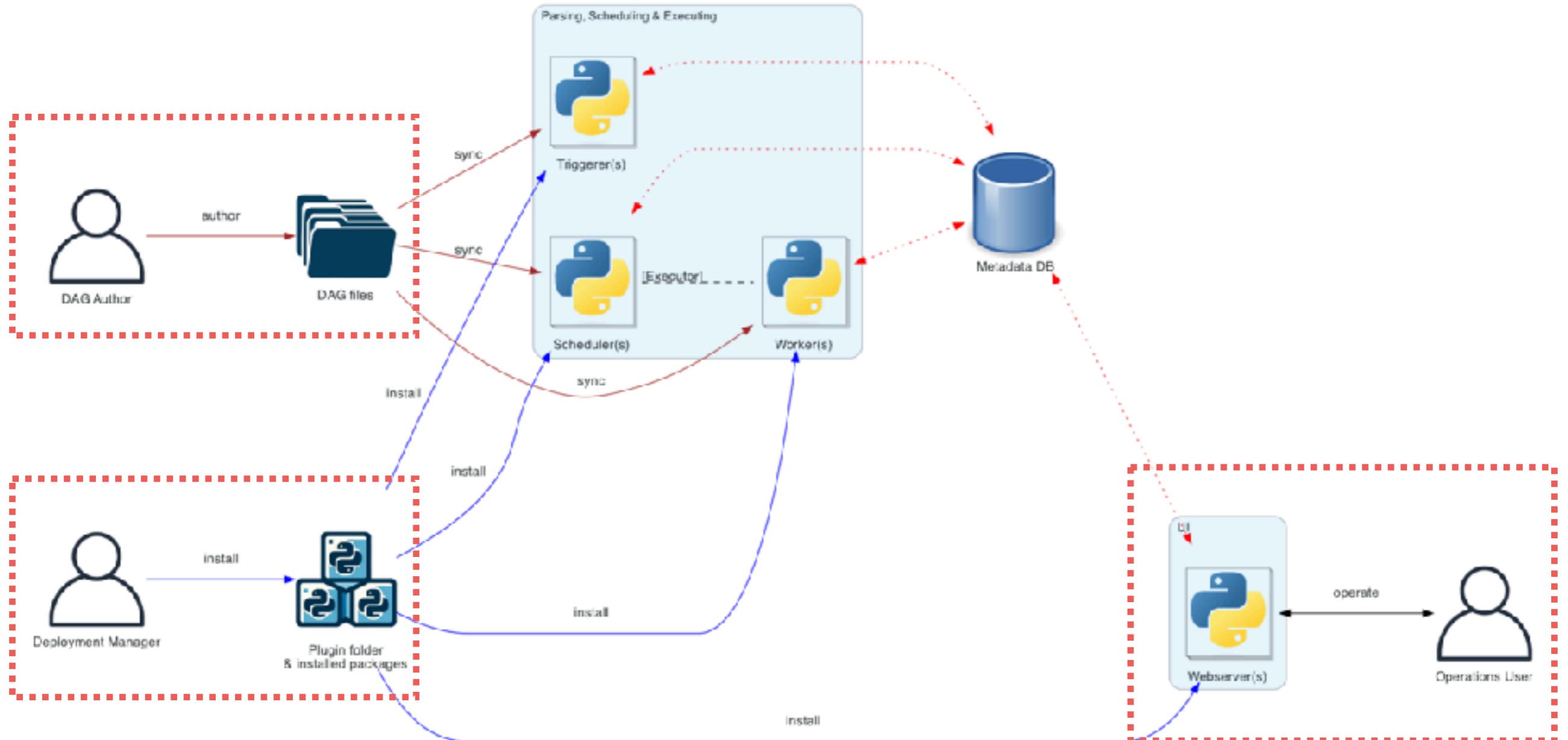
Celery executor

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

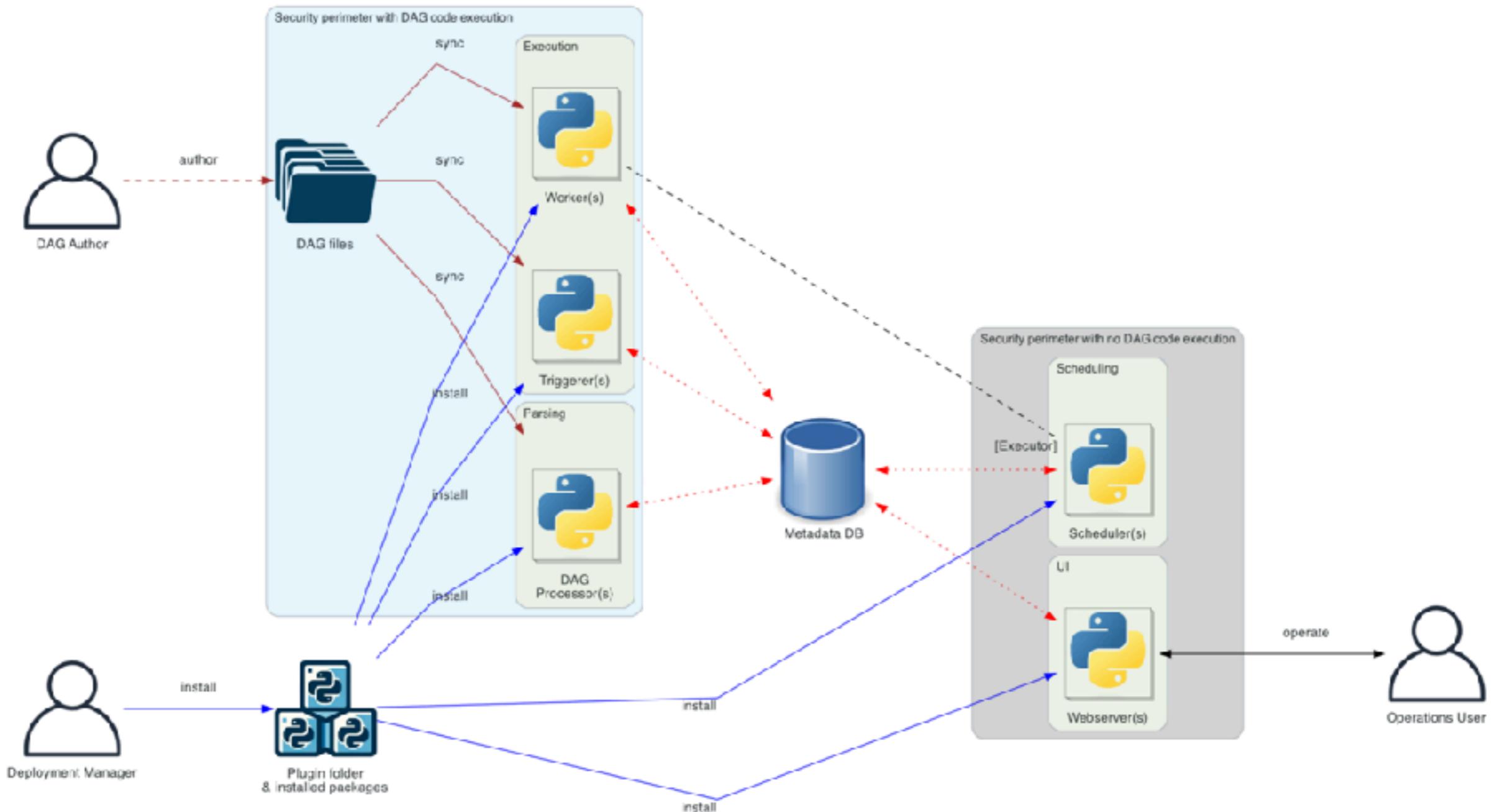
Take time to setup, use more resources



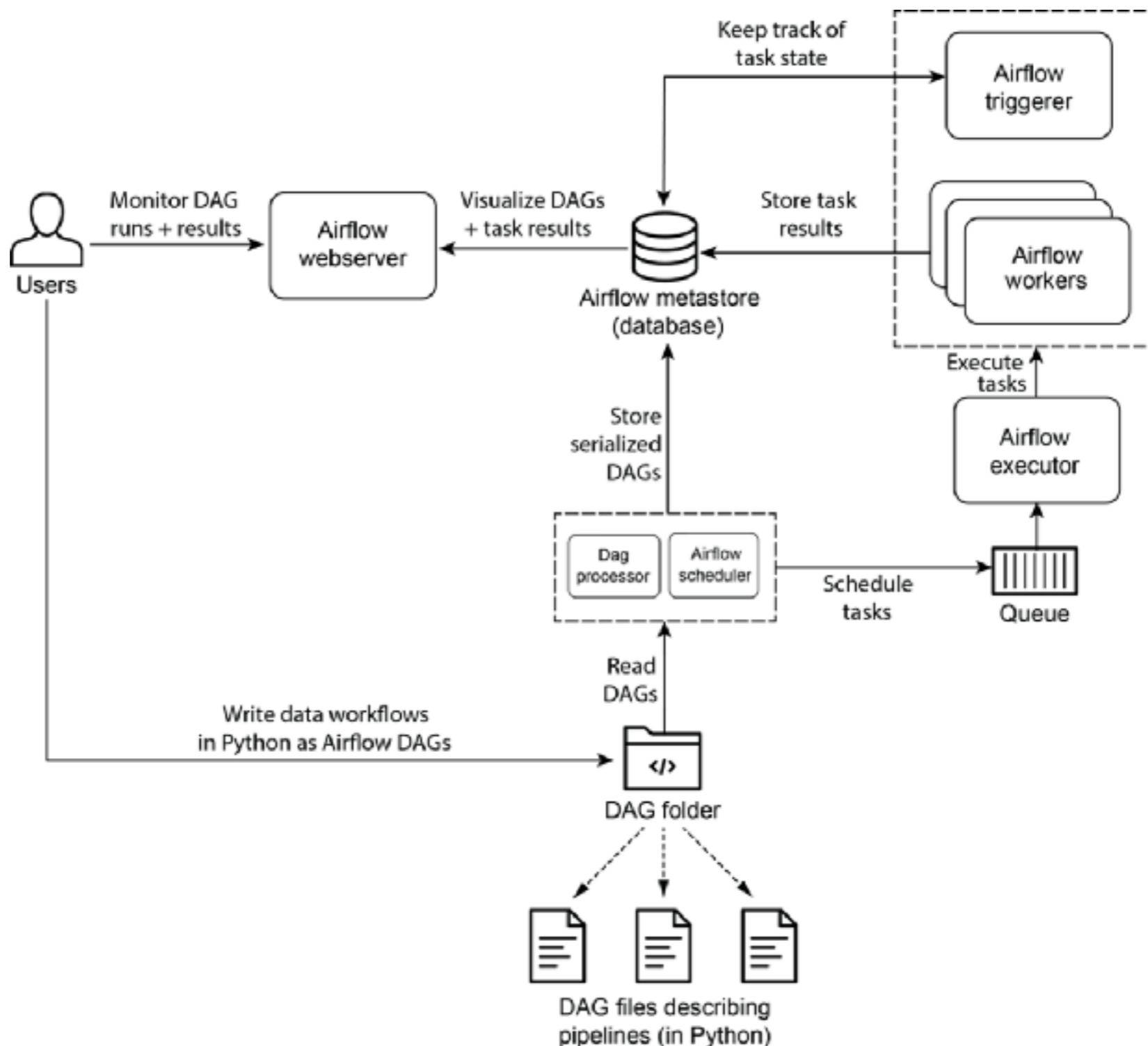
Distributed Airflow Deployment



Separate DAG processing



Airflow Components



3 Main Components

Scheduler

Worker

Trigger

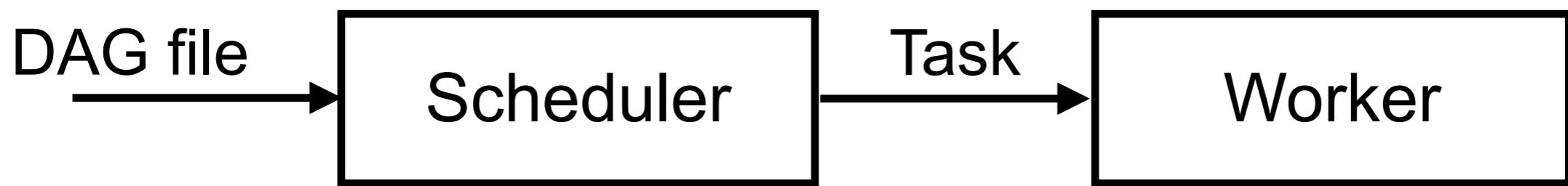
Dag processor

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



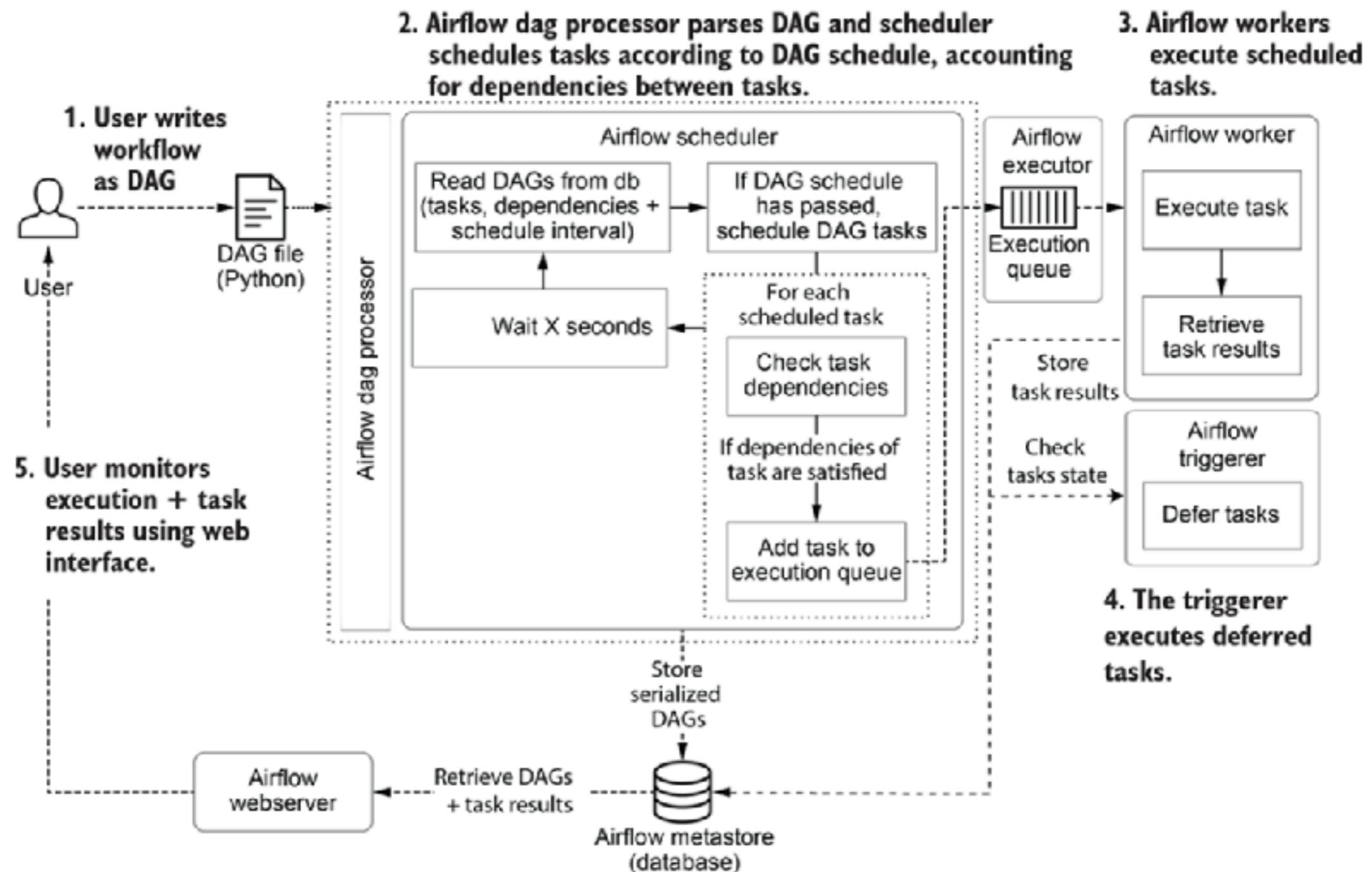
Trigger

Checks task completion status for tasks that support asynchronous processing

Allows to check for specific conditions in the background



Execution pipeline of DAG



Apache Airflow Deployment



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



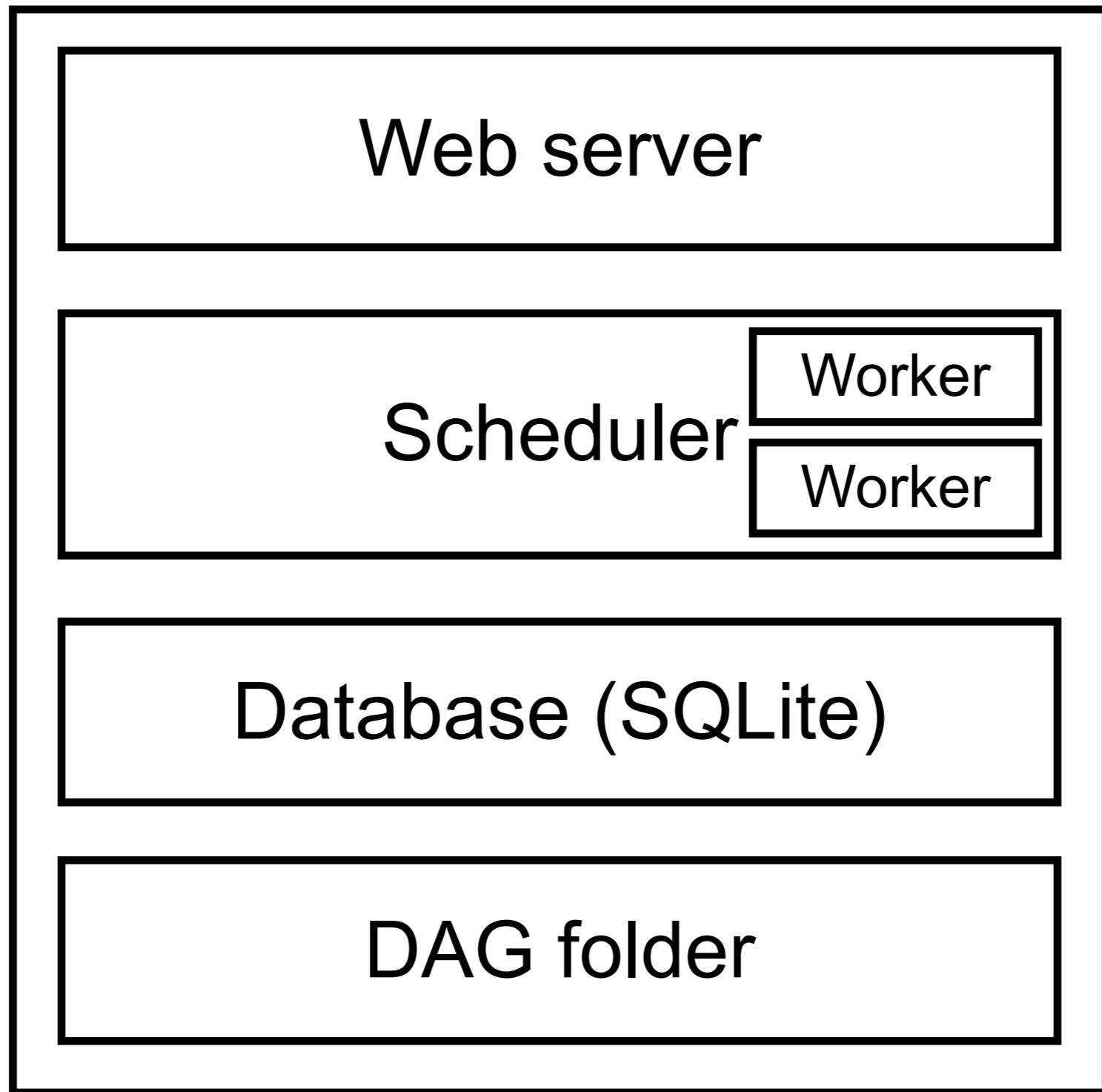
Deployment Airflow

Single
machine

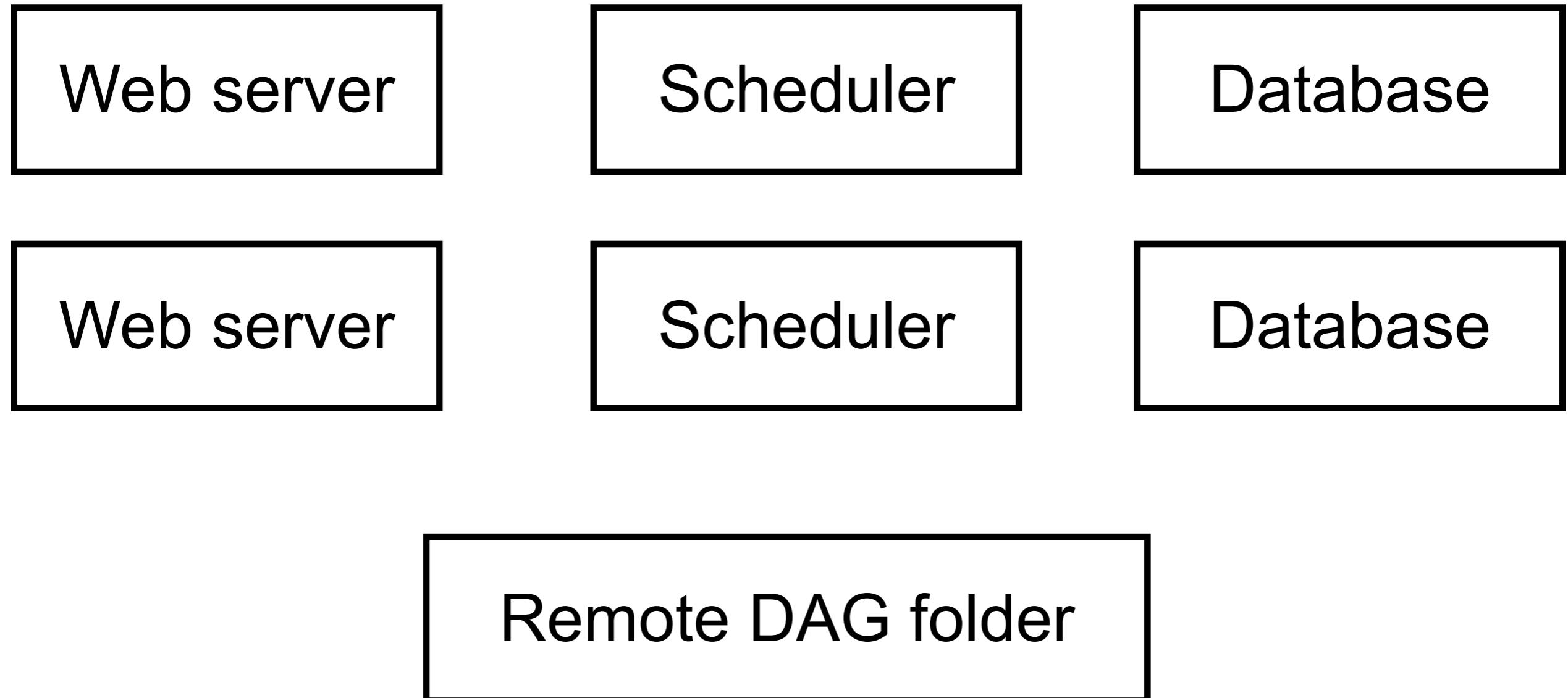
Distributed



Single machine



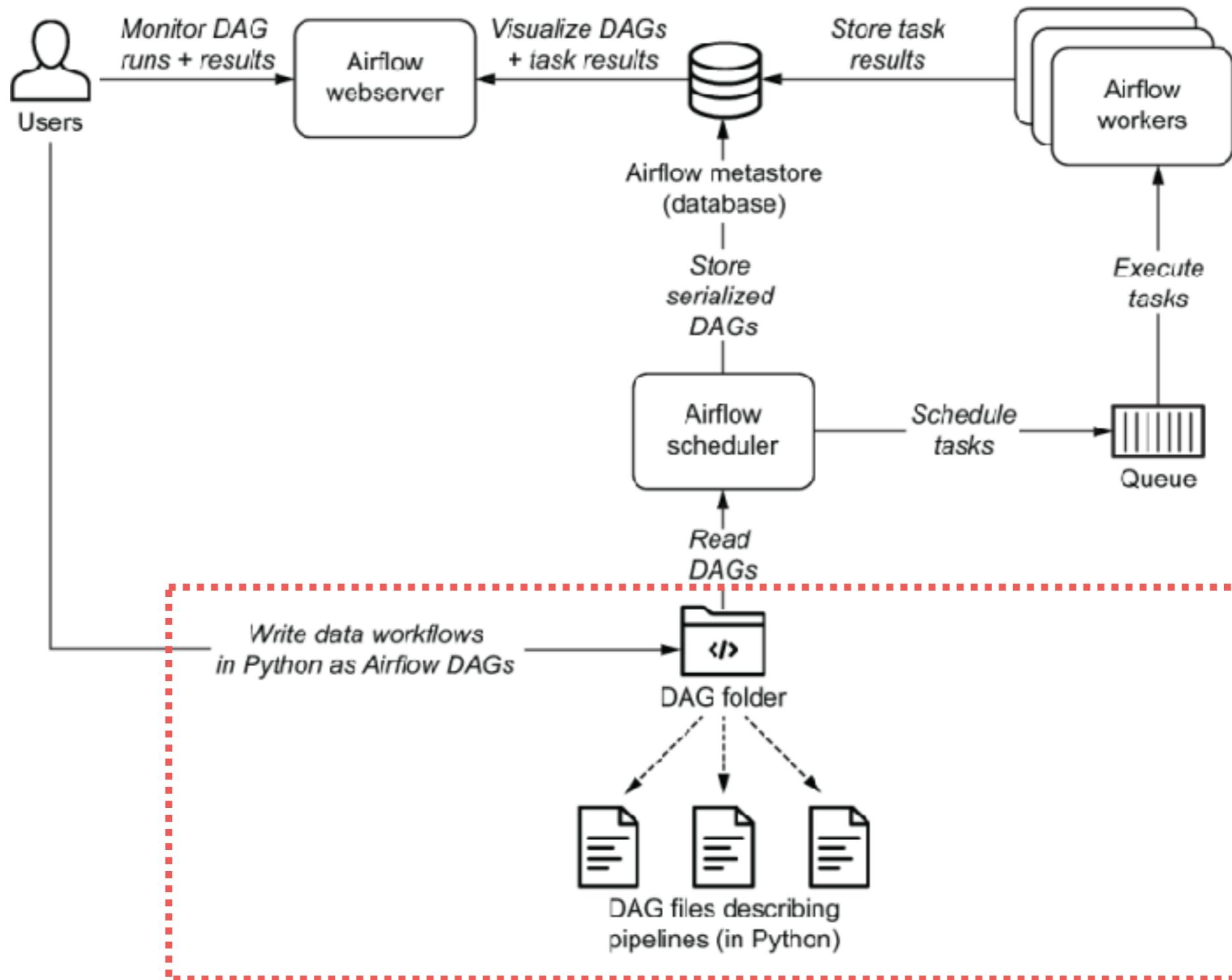
Distributed deployment



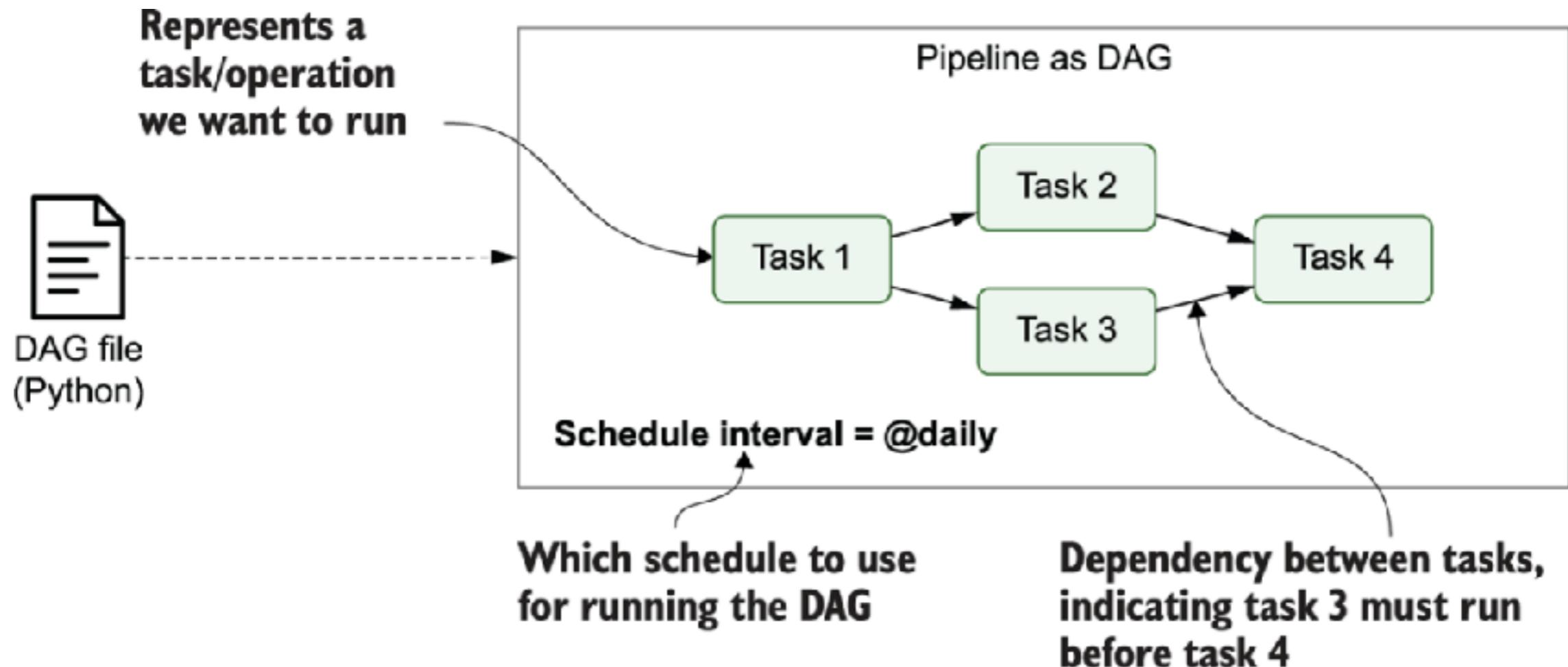
Back to DAG and Tasks



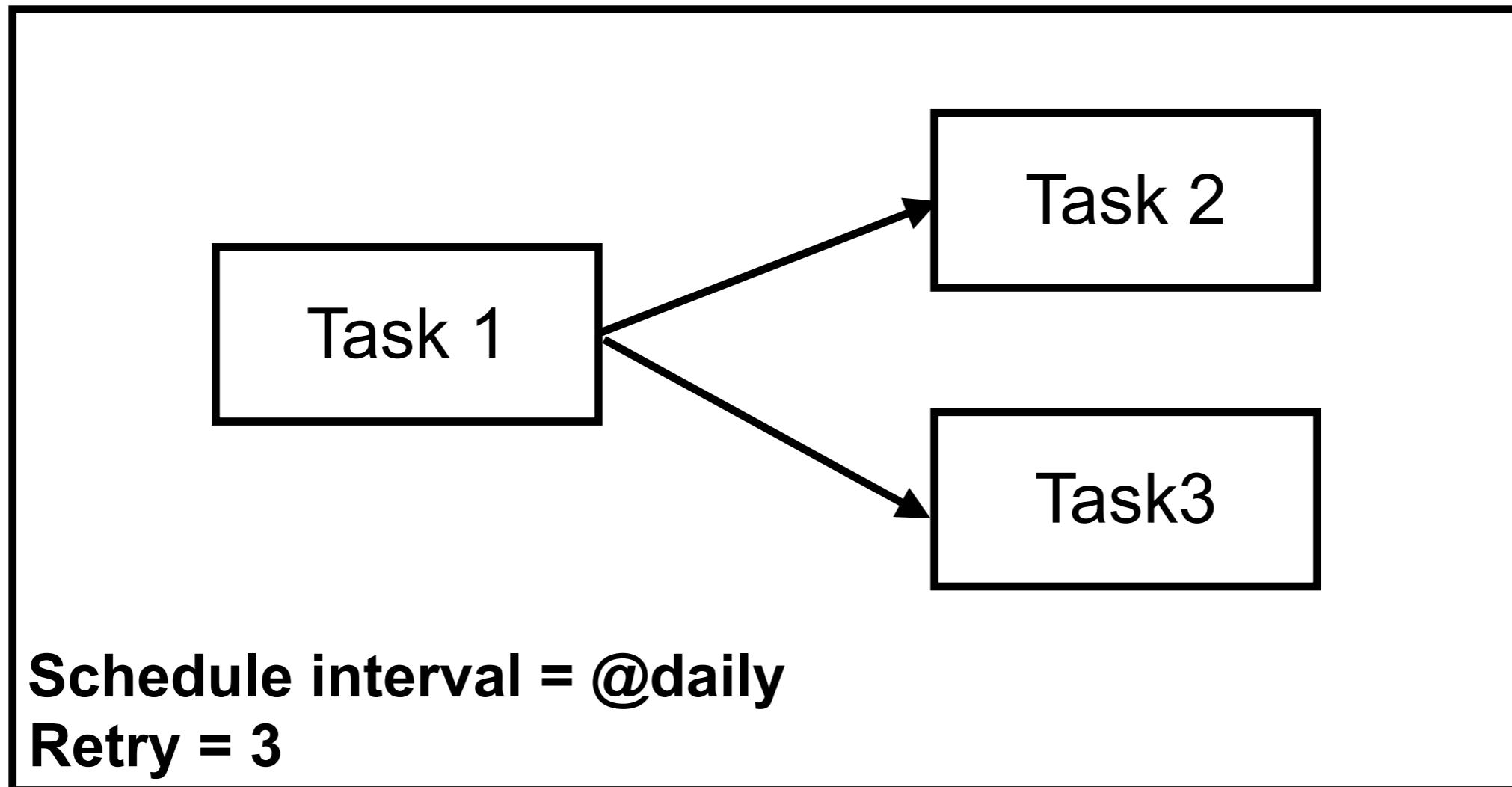
Create DAG file



Hello DAG



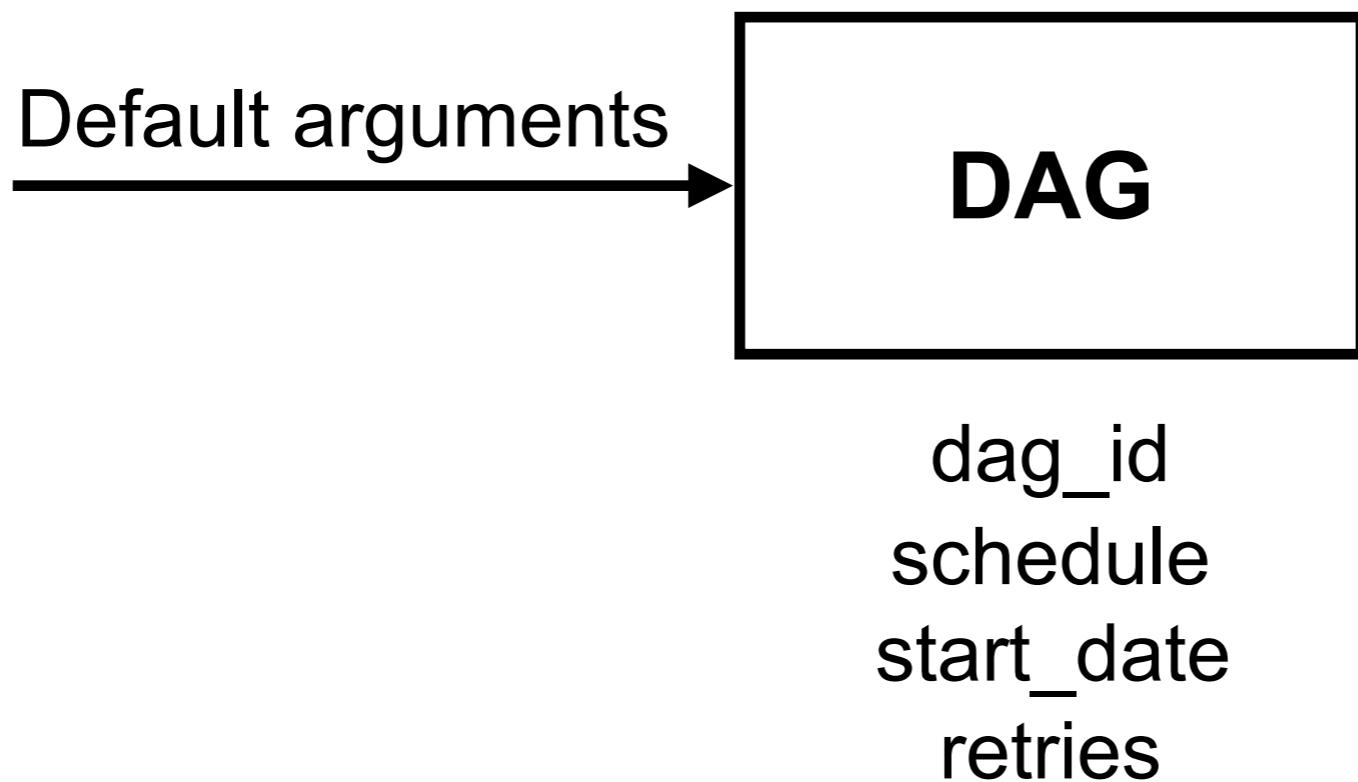
Hello DAG



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

```
@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



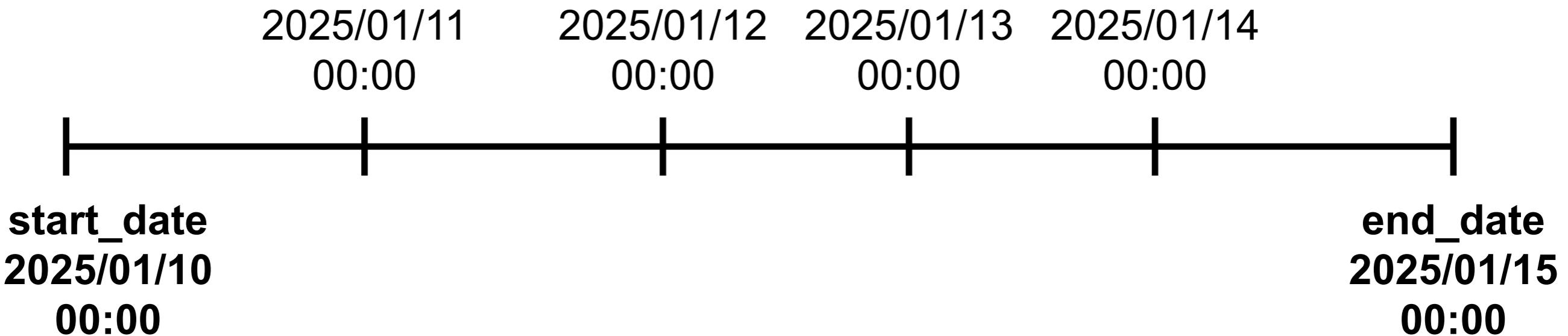
Scheduler presets

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



Example

start_date, end_date and @daily



```
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

```
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

The screenshot shows the crontab.guru editor interface. At the top, it says "crontab guru" and "The quick and simple editor for cron schedule expressions by Cronitor". A green button labeled "Cron Job Monitoring" is visible. Below the title, the text "At 04:05." is displayed in a large, italicized font. Underneath it, the text "next at 2025-01-03 04:05:00" and a "random" link are shown. The main input field contains the cron expression "5 4 * * *". Below the input field is a table of cron syntax rules:

minute	hour	day (month)	month	day (week)
*				any value
,				value list separator
-				range of values
/				step values
@yearly				(non-standard)
@annually				(non-standard)
@monthly				(non-standard)
@weekly				(non-standard)
@daily				(non-standard)
@hourly				(non-standard)
@reboot				(non-standard)

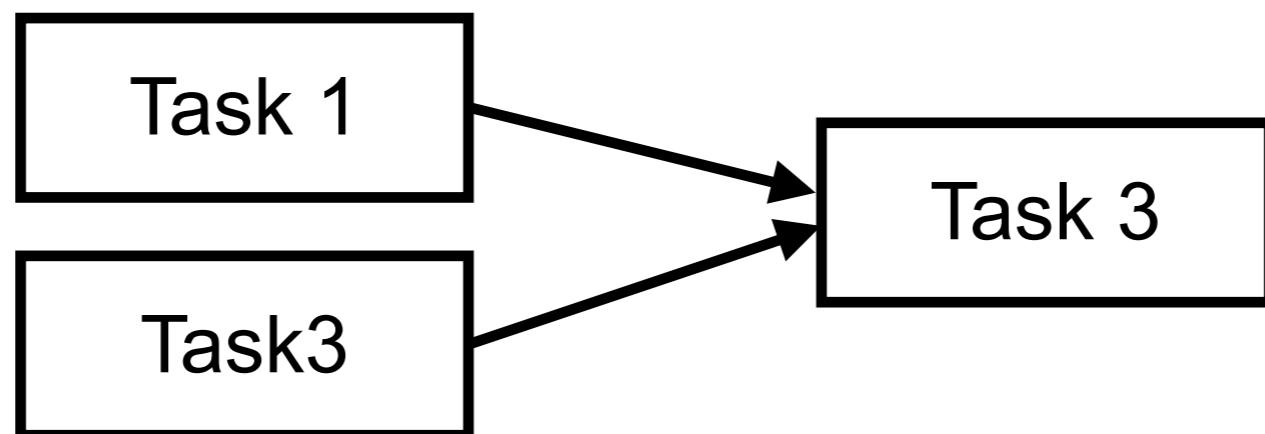
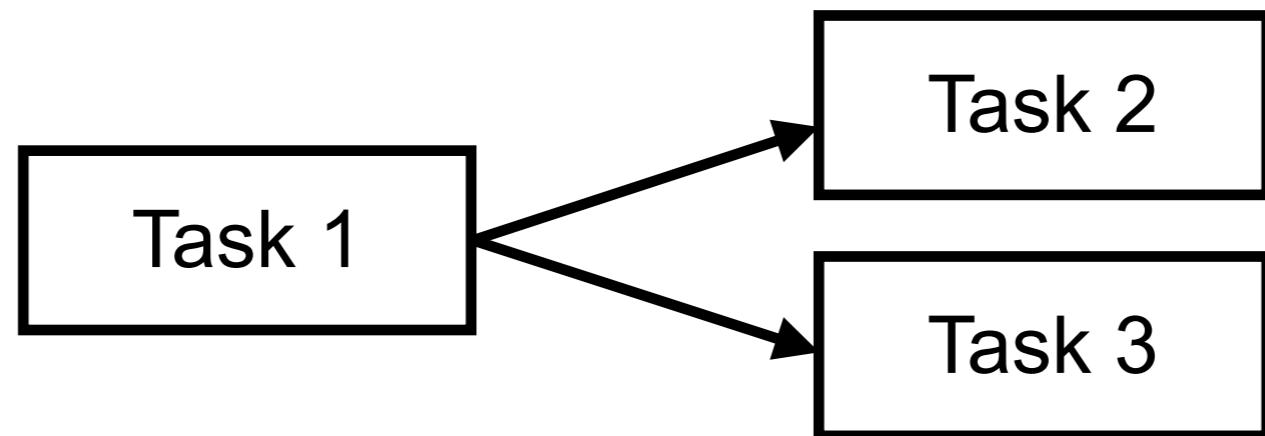
<https://crontab.guru/>



Task dependencies



Task dependencies

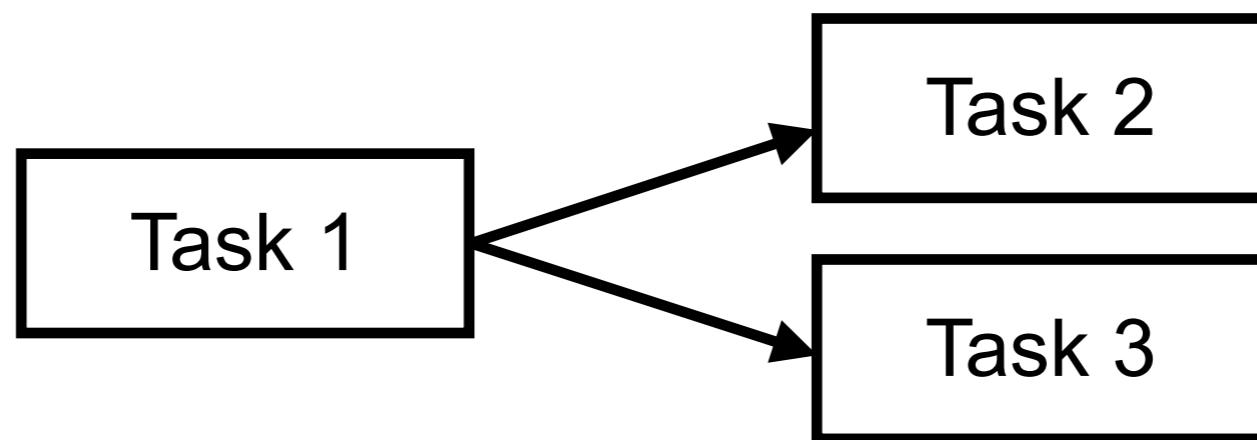


<https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/tasks.html#relationships>



Task dependencies

```
# 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>



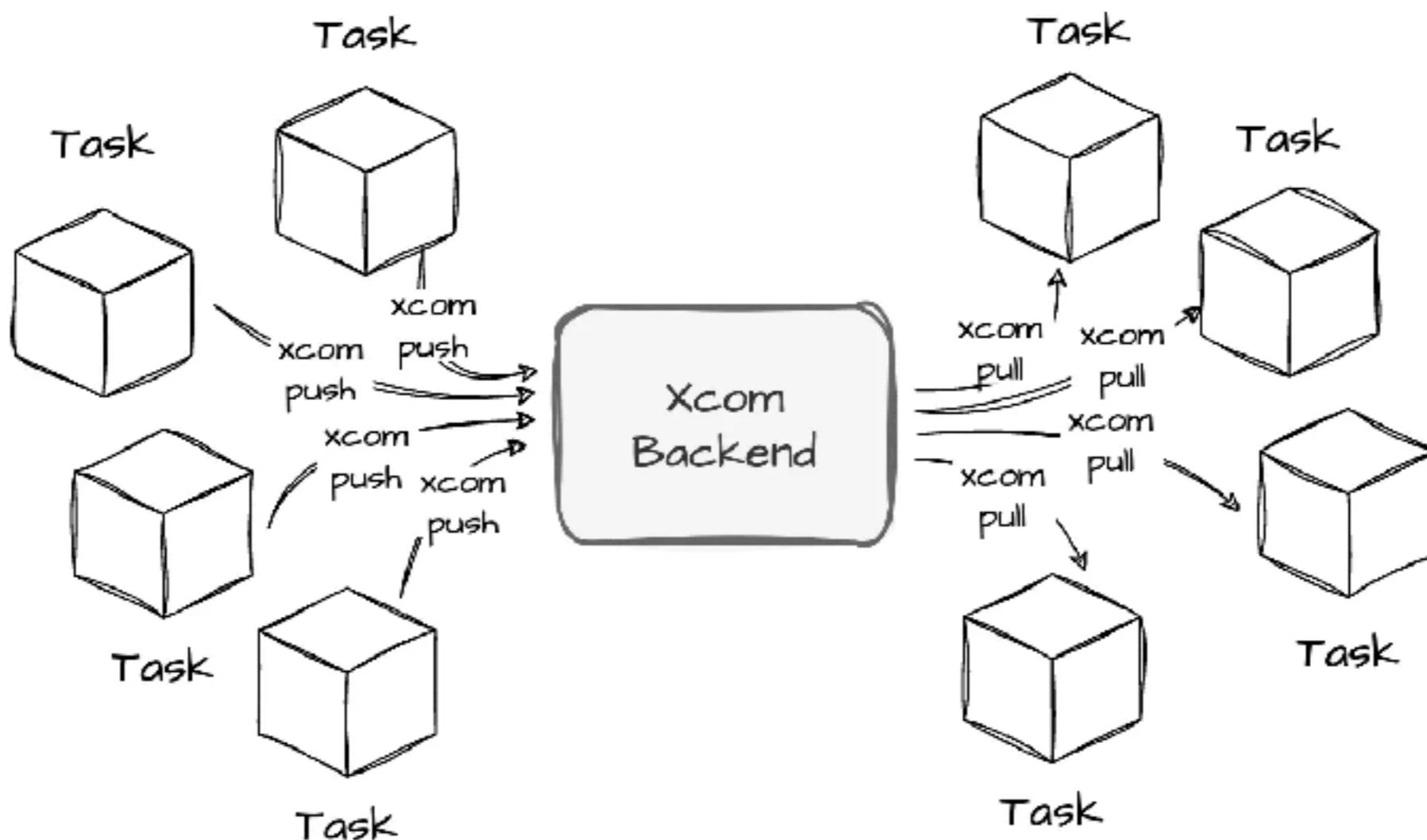
Exchange data between tasks

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)



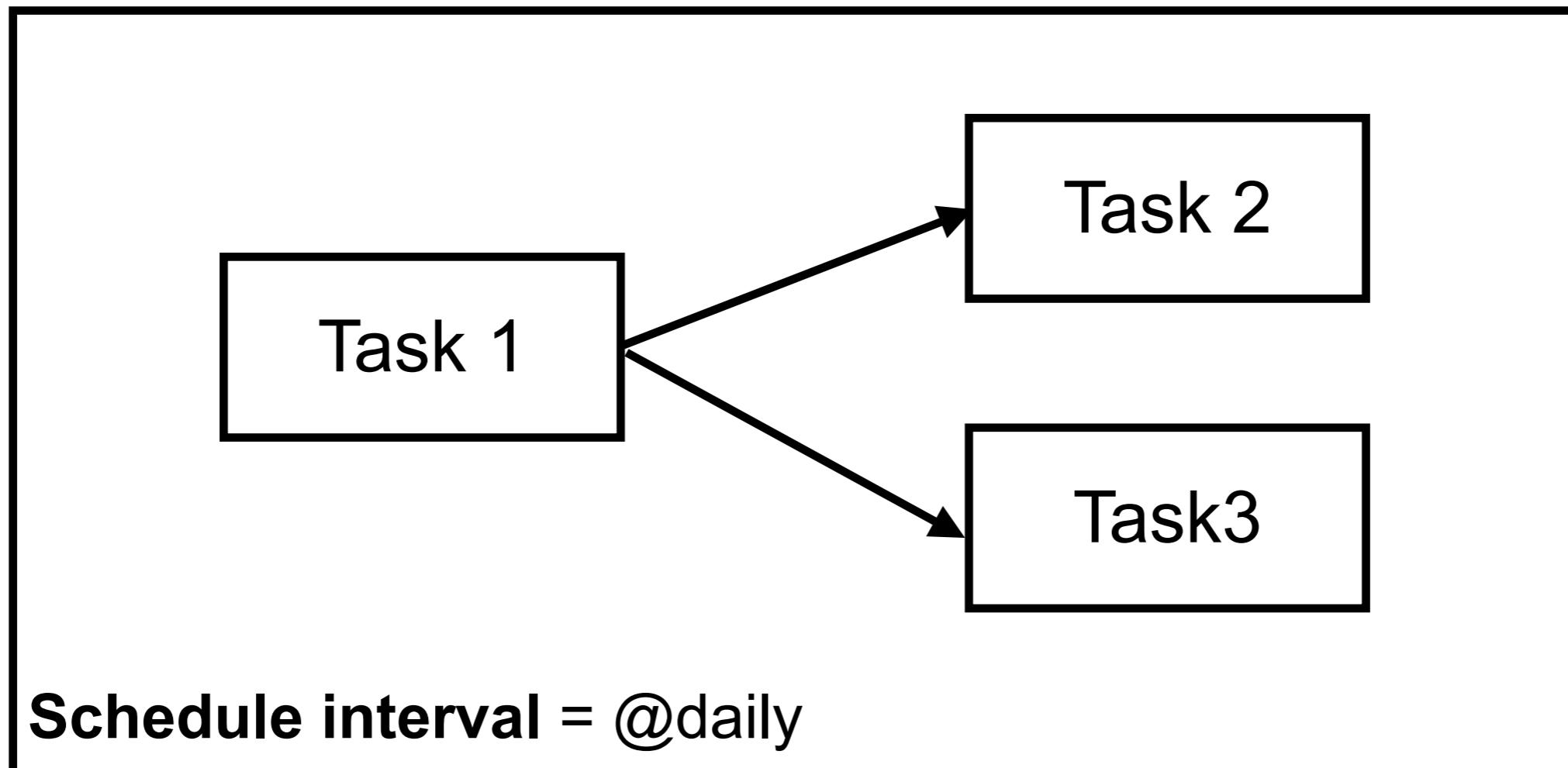
Empty vs Dummy

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	<pre>from airflow.operators.empty import EmptyOperator</pre>	<pre>from airflow.operators.dummy import DummyOperator</pre>



Hello DAG (1)

Working with Empty/DummyOperator



EmptyOperator

```
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

The screenshot shows the Apache Airflow Web UI interface for the DAG: empty_dag. At the top, there is a navigation bar with links for Airflow, DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. Below the navigation bar, the title "DAG: empty_dag" is displayed, along with a "Schedule: @daily" button.

Below the title, there are several filter and search options:

- Date range: 02/01/2025 to 04:03:38
- Run Types: All Run Types
- Run States: All Run States
- Action buttons: Clear Filters, Press shift + / for Shortcuts, and a row of status filters: deferred, failed, queued, removed, restarting, running, scheduled, shutdown, skipped, succeeded.

The main area displays the DAG structure for "empty_dag". On the left, a sidebar shows the tasks and their execution times:

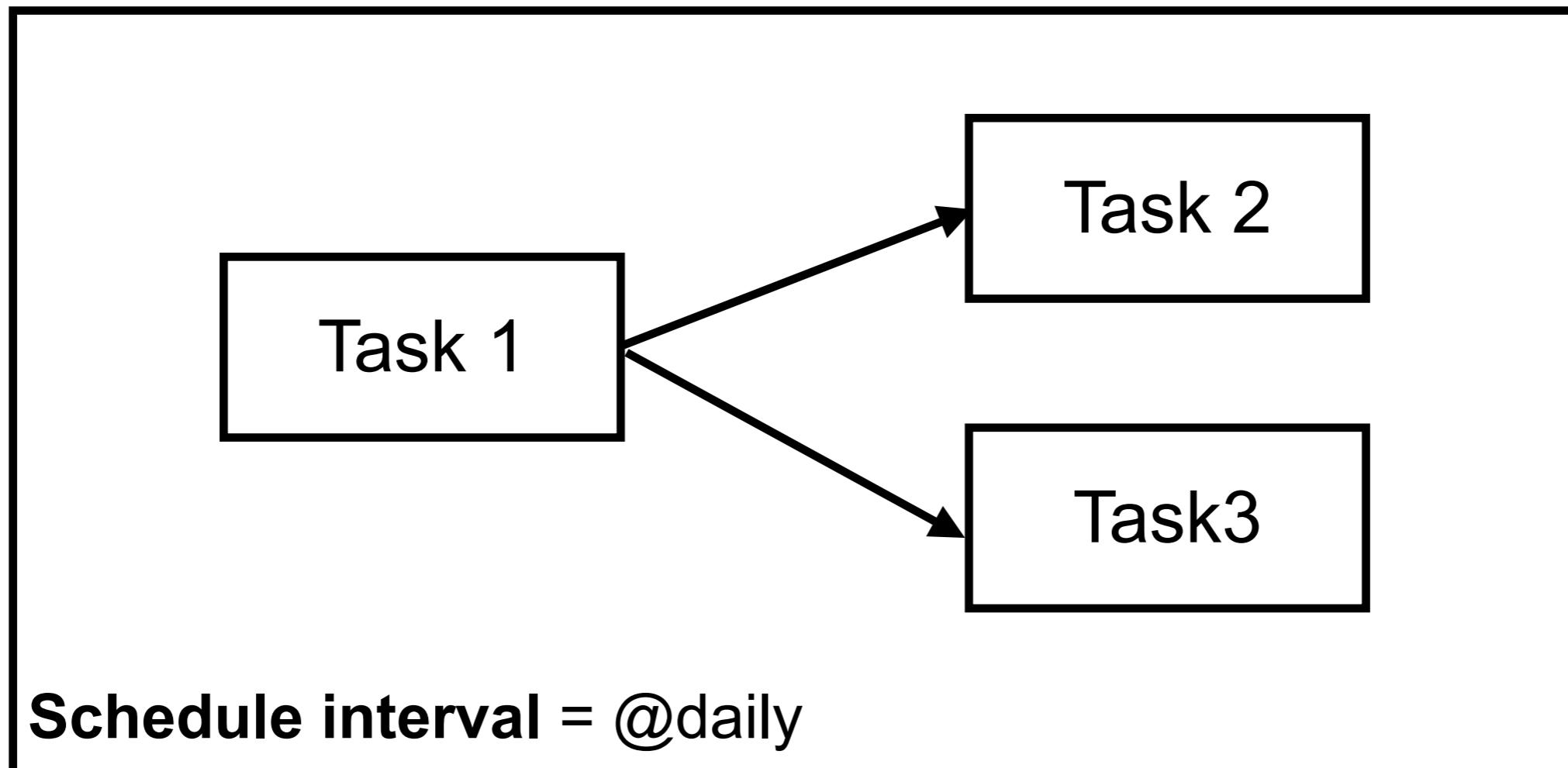
task	Duration
task1	00:00:02
task2	00:00:01
task3	00:00:00

The DAG graph shows three tasks: task1, task2, and task3. Task1 is the root node, connected to task2 and task3. All tasks are labeled "EmptyOperator". The tasks are represented by light green boxes with rounded corners.



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!"
    )

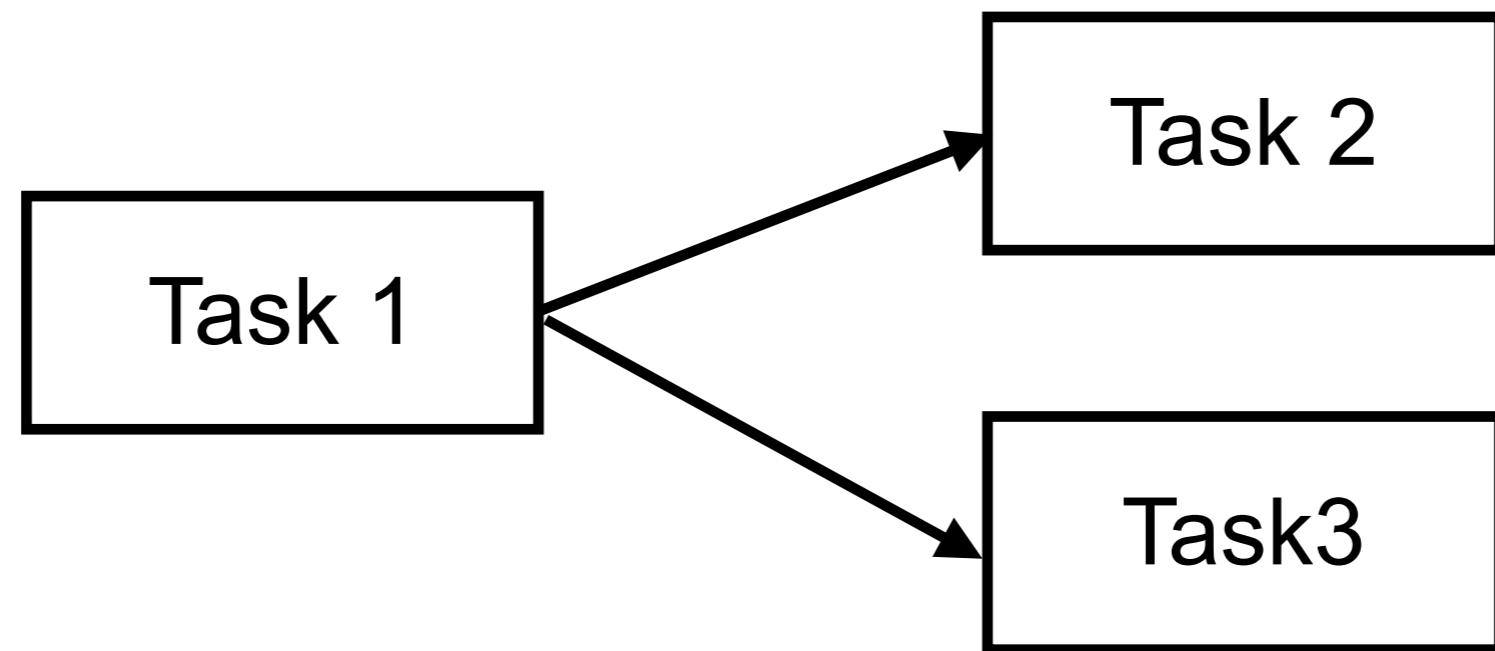
    task2 = BashOperator(
        task_id='second_task',
        bash_command="echo hey, I am task2"
    )

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

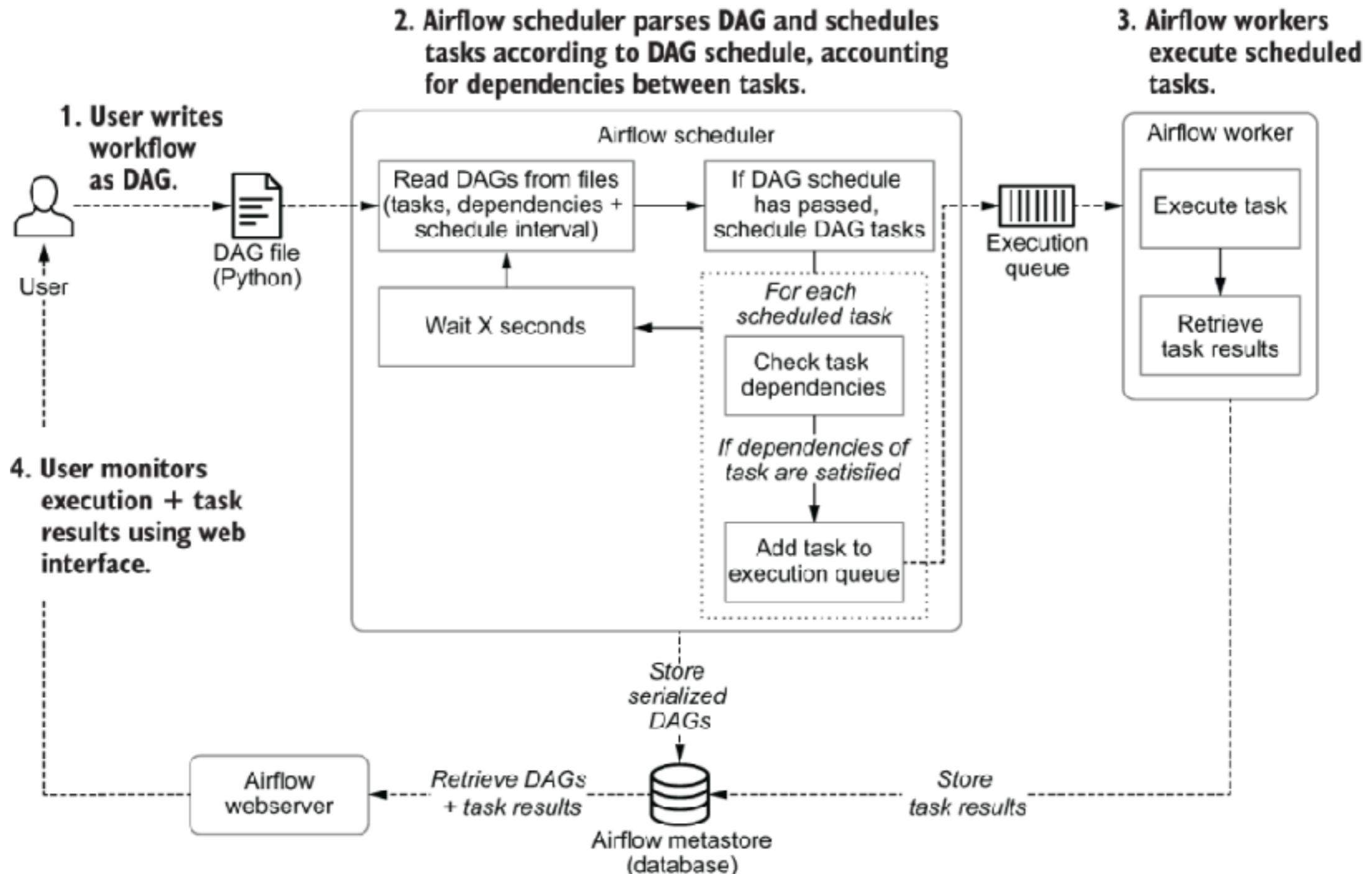


Hello DAG (3)

Working with **PythonOperator** and **TaskFlow**



High level



Trigger DAG

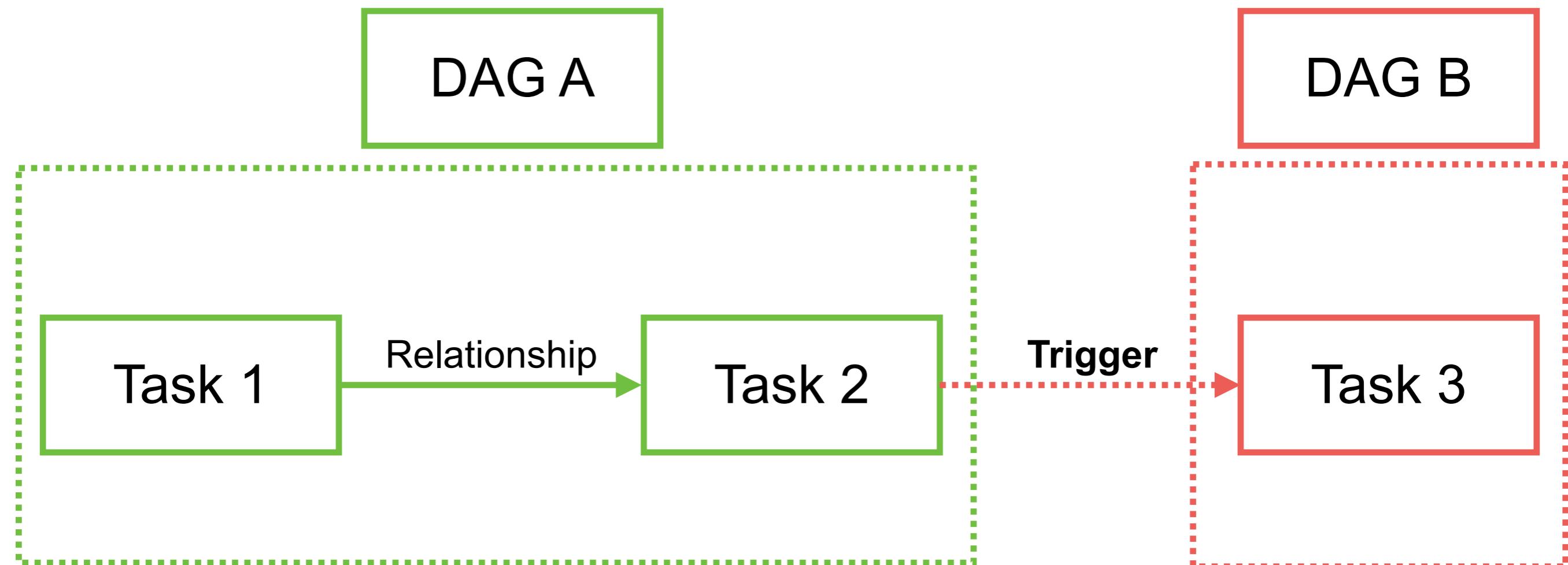


Airflow Trigger

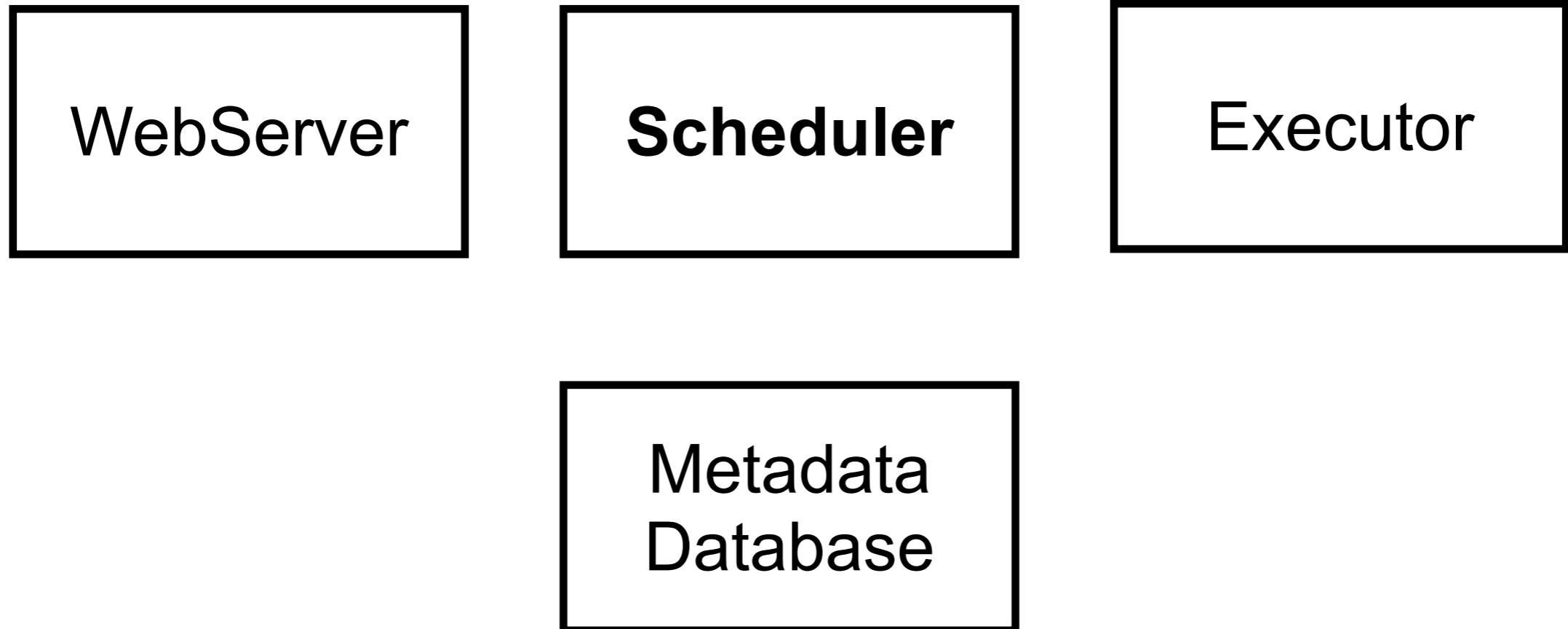
On a schedule from DAG
Manual trigger (UI or CLI or API)
External trigger



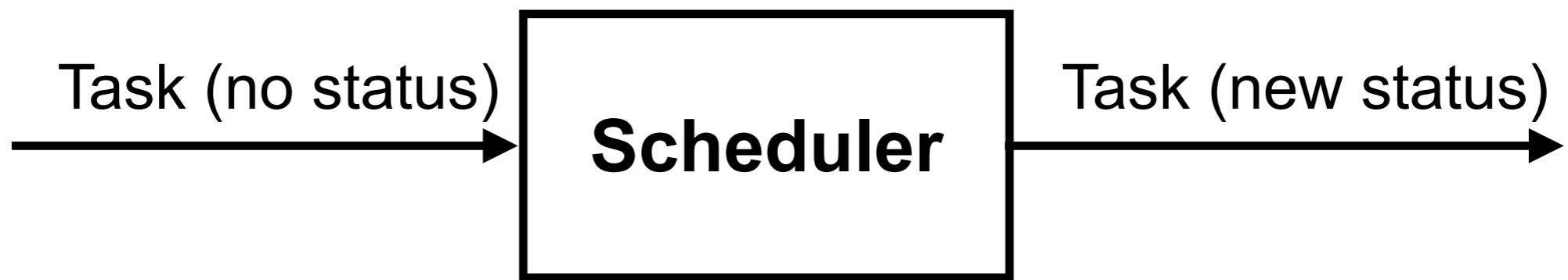
External Trigger



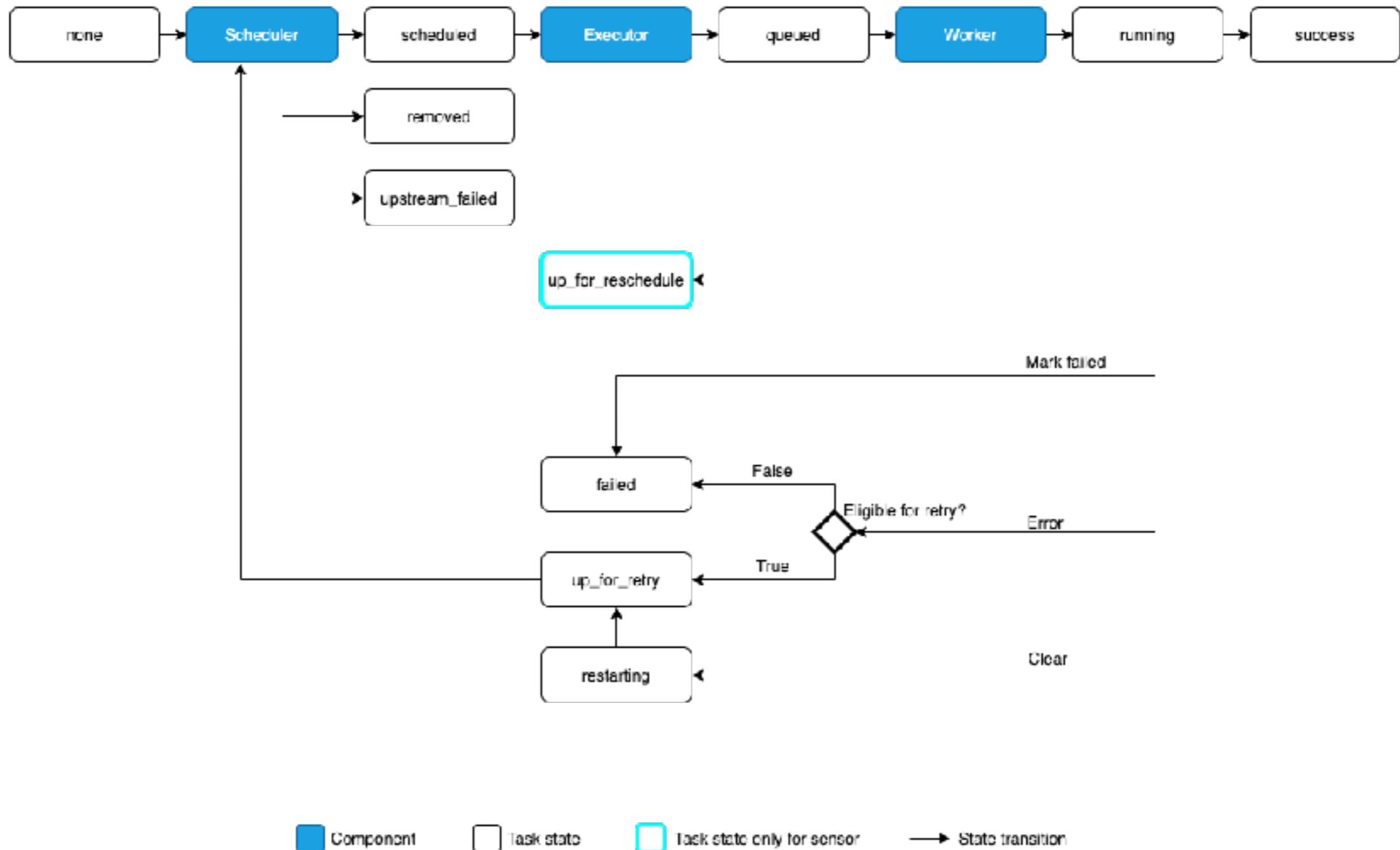
Simple Airflow architecture



Task Life Cycle



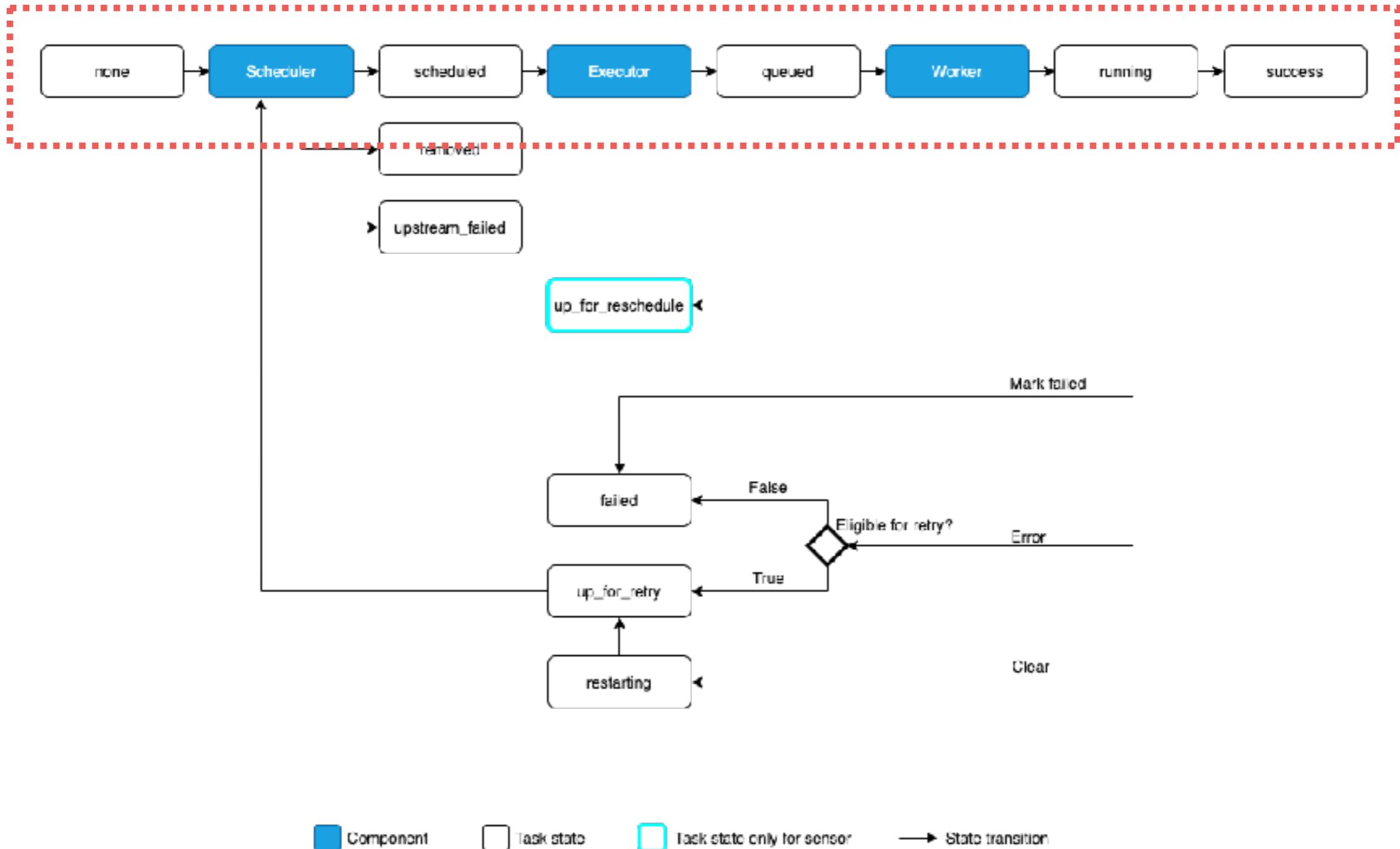
Task Life Cycle



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



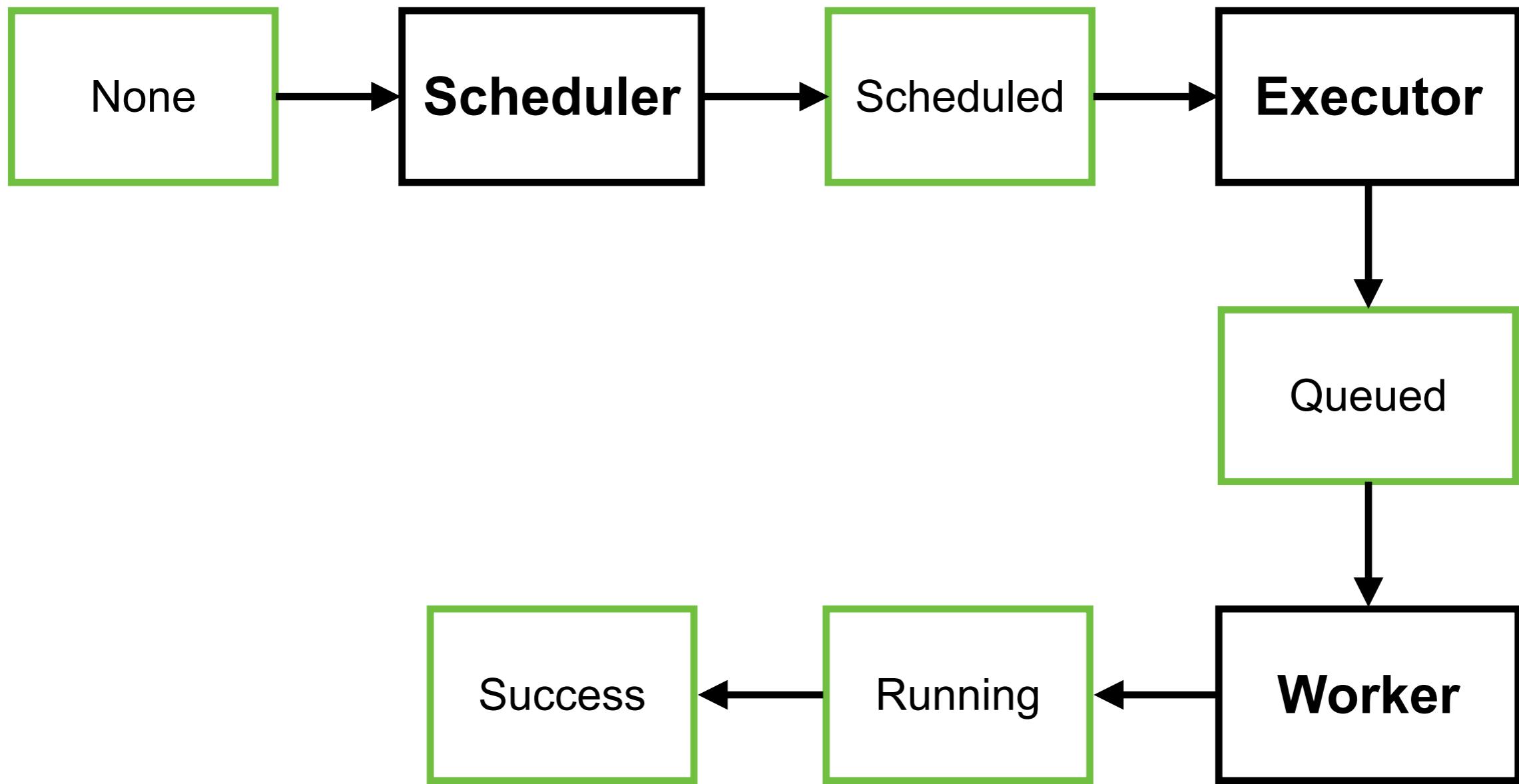
Task Life Cycle (Normal flow)



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



Task Life Cycle (Normal flow)

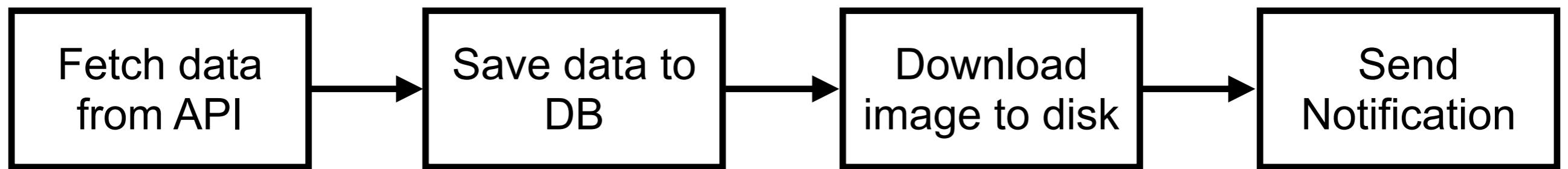


DAG workshop

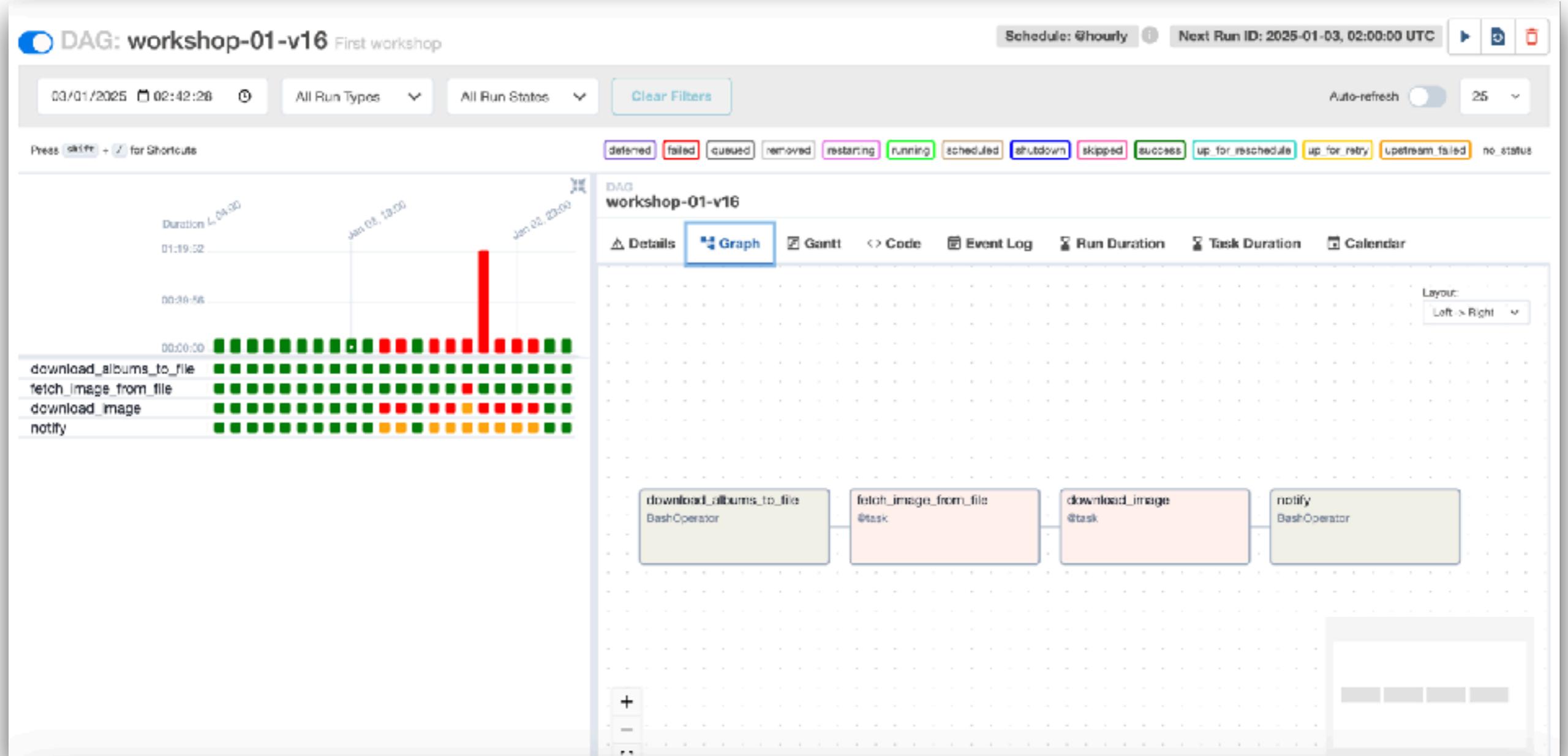
<https://github.com/up1/workshop-apache-airflow/wiki/DAG-workshop>



DAG workshop



Airflow UI (1)



Airflow UI (2)

Detail in each task



Airflow UI (2)

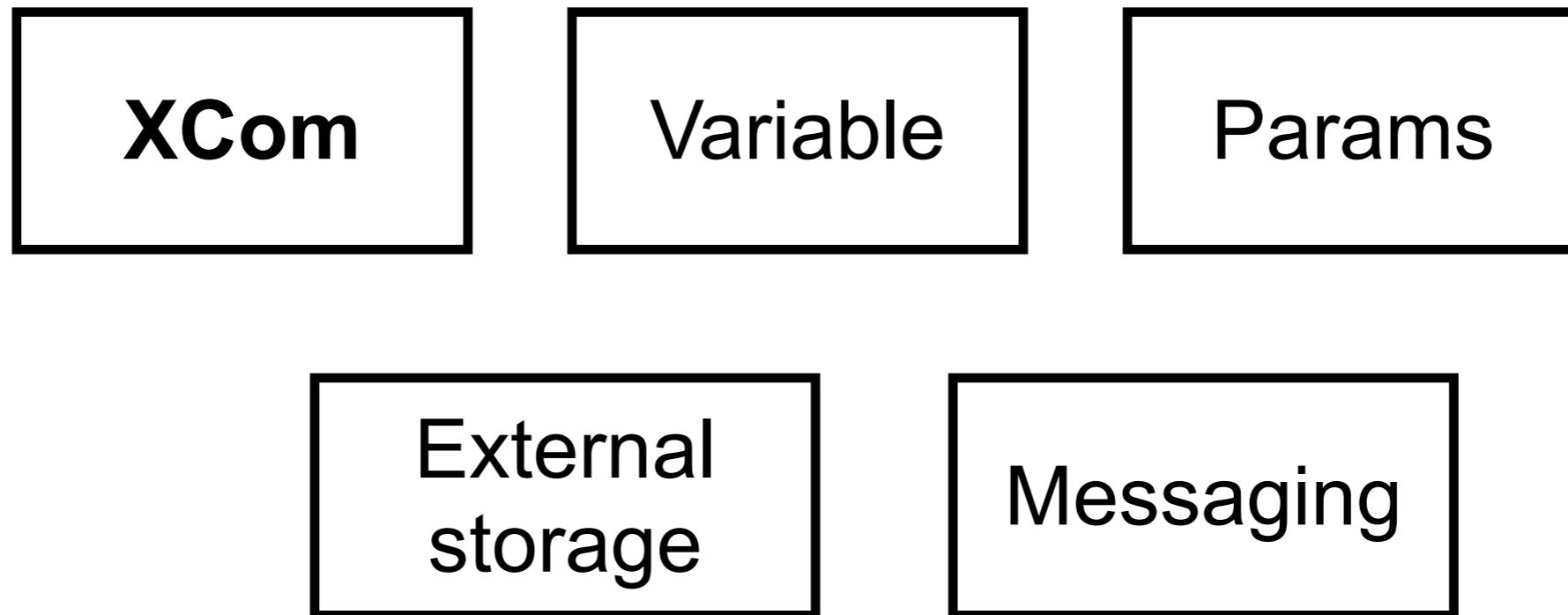
XCom = Cross communication



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



Communication between tasks

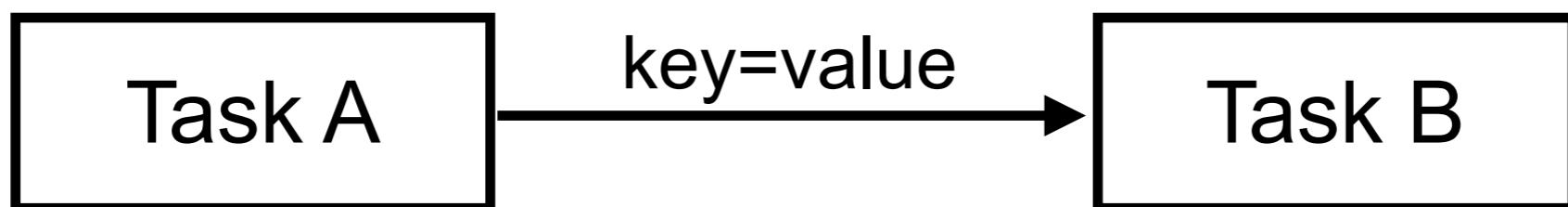


XCom (Cross Communication)

Inter-task communication

Task-level scope

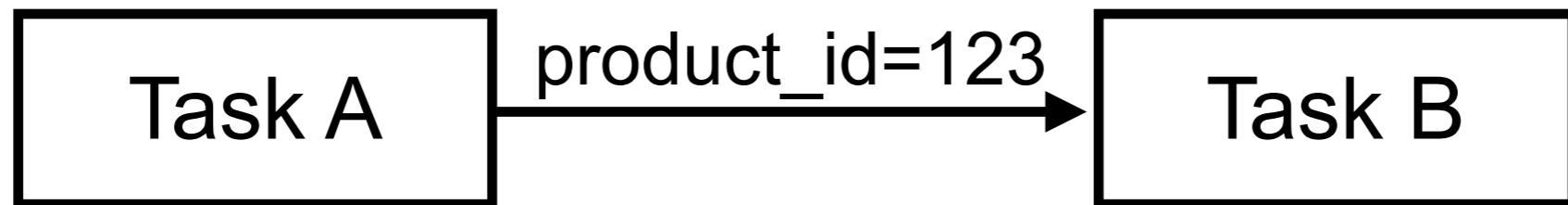
Support any serializable Python object



<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)
```

DAG demo_xcom_v3 / Run 2025-01-13, 21:14:00 +07 / task_a

Clear task

Details Graph Gantt Code Event Log Logs XCom Task Duration

Key	Value
product_id	1af896a8-d20e-47b8-bb59-d3304759d539

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



List of XComs

Admin -> XComs

Key	Value	Timestamp	Dag Id	Task Id	Run Id	Map Index	Execution Date
product_id	1b33c5c7-f7f6-4bb5-9082-fd9459273353	2025-01-13, 21:18:00	demo_xcom_v3	task_a	scheduled_2025-01-13T14:17:00+00:00		2025-01-13, 21:17:00
product_id	92e84155-b881-4ddc-b752-400c637045fb	2025-01-13, 21:17:01	demo_xcom_v3	task_a	scheduled_2025-01-13T14:16:00+00:00		2025-01-13, 21:16:00
product_id	2b111386-b33f-4e2f-98a4-114d6c575ebc	2025-01-13, 21:16:00	demo_xcom_v3	task_a	scheduled_2025-01-13T14:15:00+00:00		2025-01-13, 21:15:00
product_id	1af896a8-c20a-47b8-bb59-d3304759d539	2025-01-13, 21:15:01	demo_xcom_v3	task_a	scheduled_2025-01-13T14:14:00+00:00		2025-01-13, 21:14:00
product_id	4f2e35c8-597a-42cd-9977-2dfe5fd150a5	2025-01-13, 21:14:00	demo_xcom_v3	task_a	scheduled_2025-01-13T14:13:00+00:00		2025-01-13, 21:13:00
product_id	63f4a10e-aff6-4ad3-8dc3-3b41e12681f0	2025-01-13, 21:13:01	demo_xcom_v3	task_a	scheduled_2025-01-13T14:12:00+00:00		2025-01-13, 21:12:00
	4c225341-64d3-47e0-				scheduled_2025-01-		



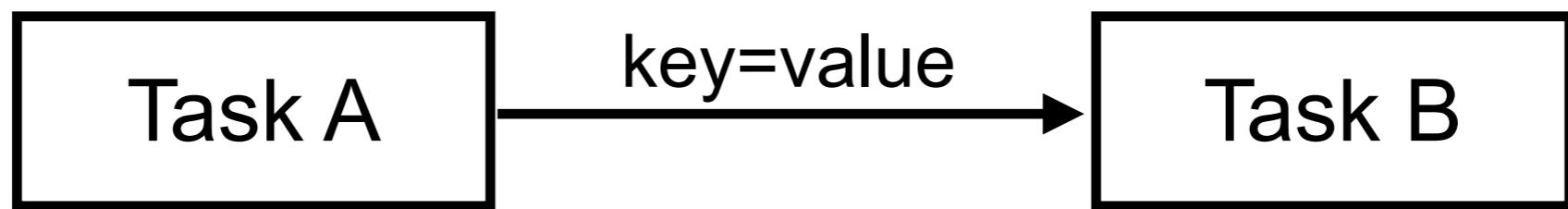
XCom data store in Database

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



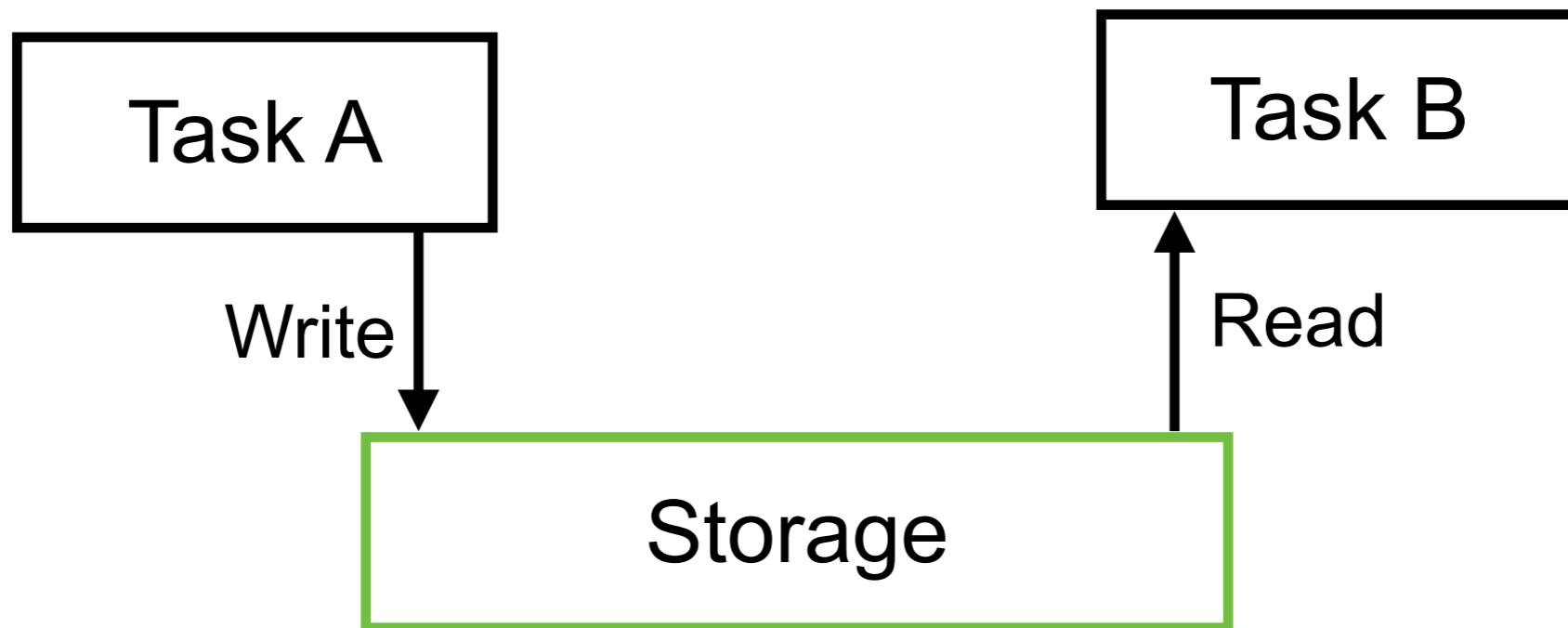
XCom (Cross Communication)

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

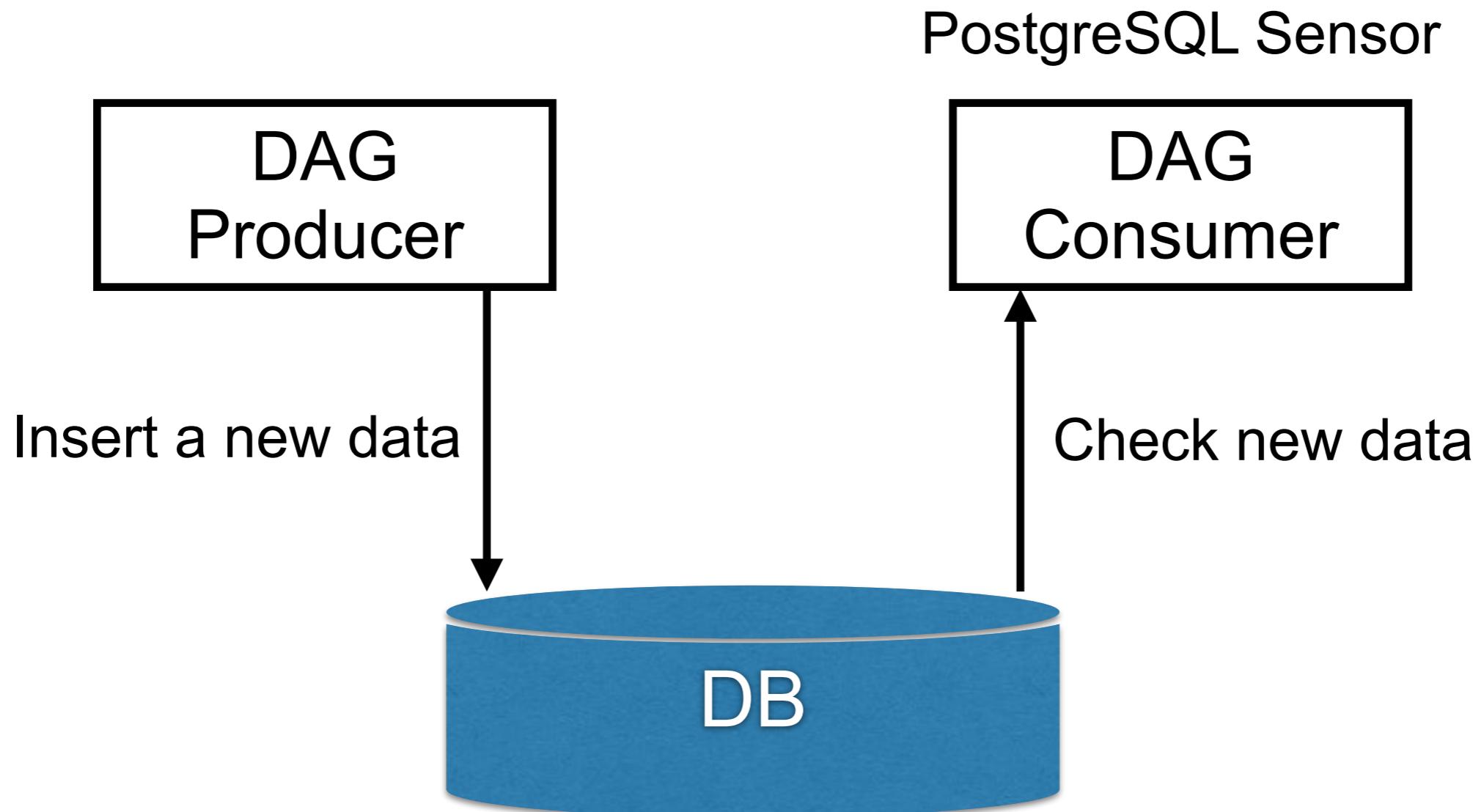
PostgreSQL Sensor



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



Workflows



Create connections

Goto menu Admin -> connections

The screenshot shows the 'Add Connection' page in the Airflow web interface. The top navigation bar includes links for Airflow, DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. The main form is titled 'Add Connection' and contains fields for a new connection:

- Connection Id ***: An input field.
- Connection Type ***: A dropdown menu showing 'Postgres'. A note below it states: 'Connection Type missing? Make sure you've installed the corresponding Airflow Provider Package.'
- Description**: An input field.
- Host**: An input field.
- Database**: An input field.
- Login**: An input field.
- Password**: An input field.
- Port**: An input field.

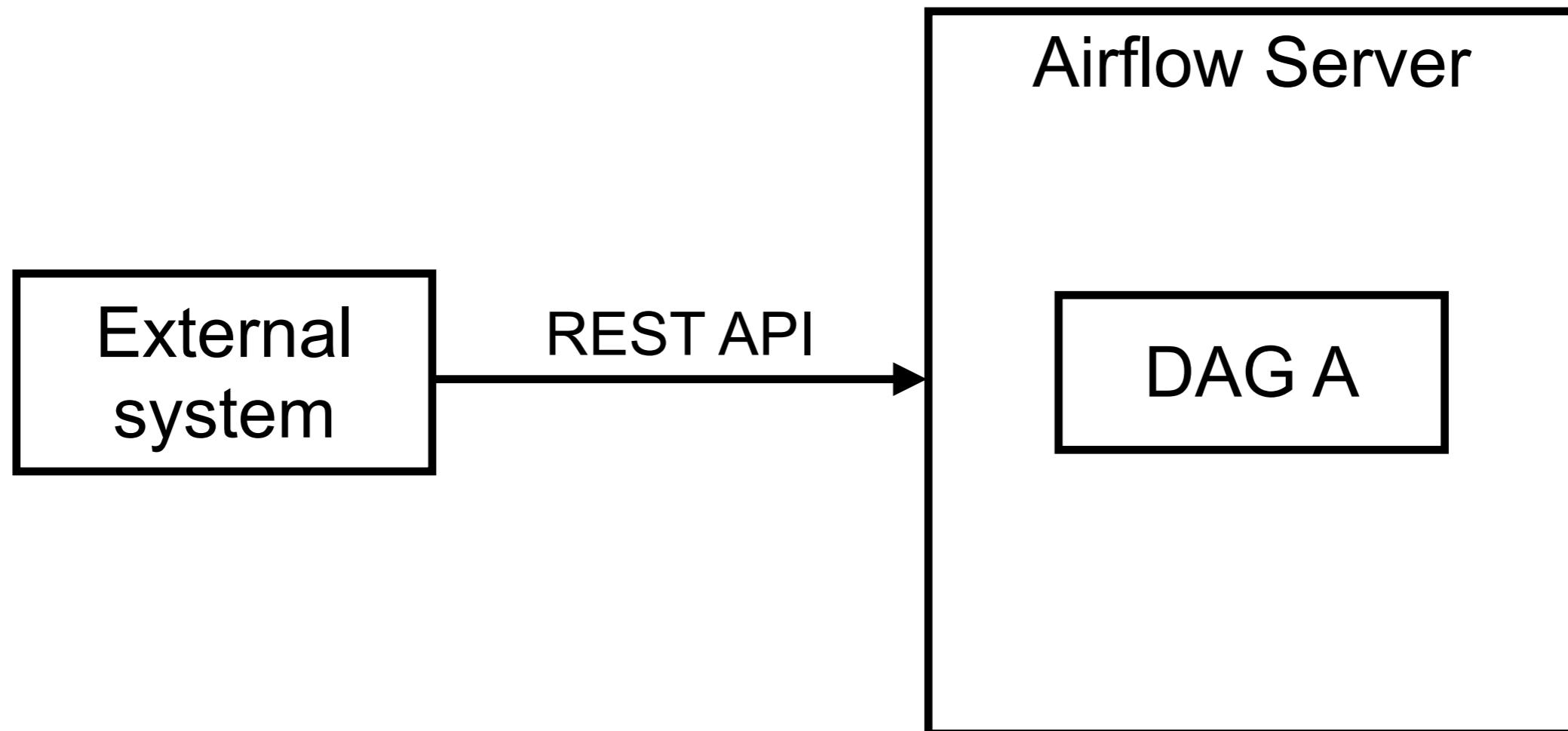


Trigger DAG via APIs

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



Trigger DAG via APIs



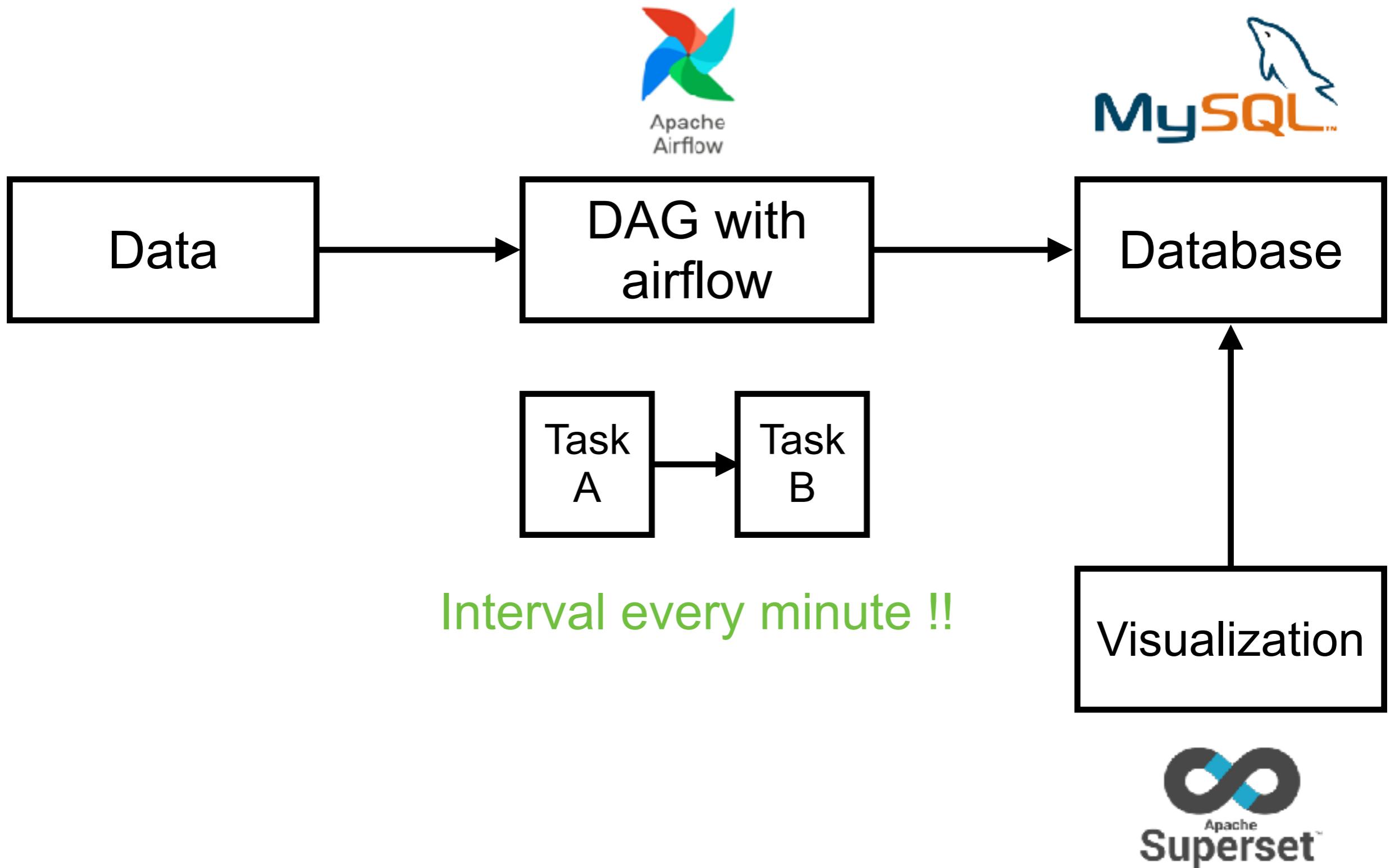
<https://github.com/up1/workshop-apache-airflow/wiki/Trigger-DAG-from-External-System-via-REST-API>



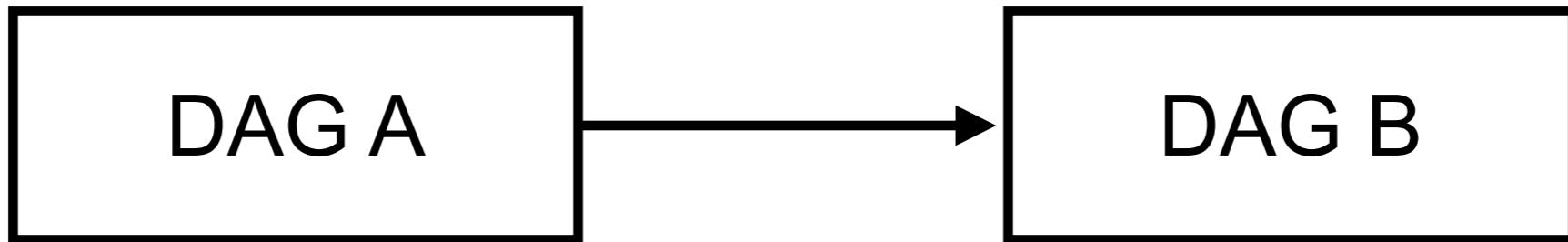
Workshop ETL



Workshop ETL



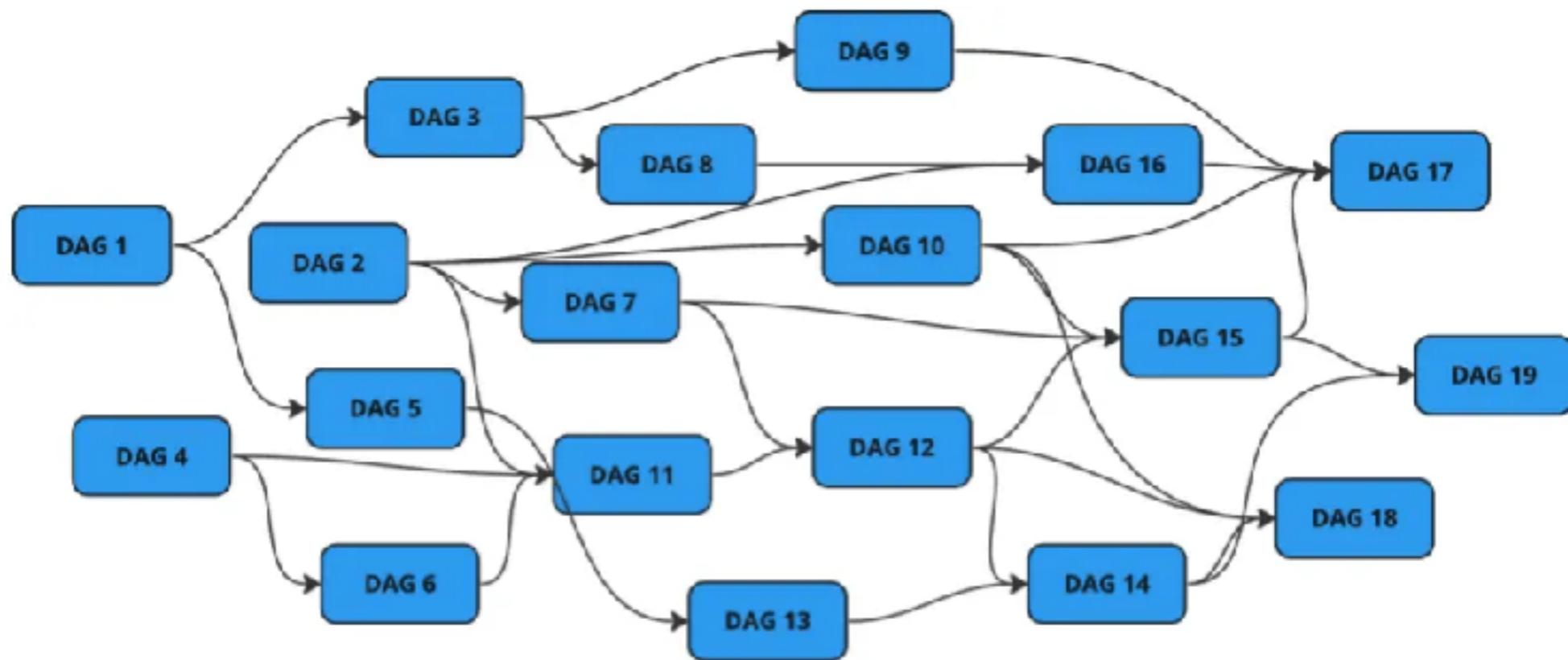
Cross DAG !!



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



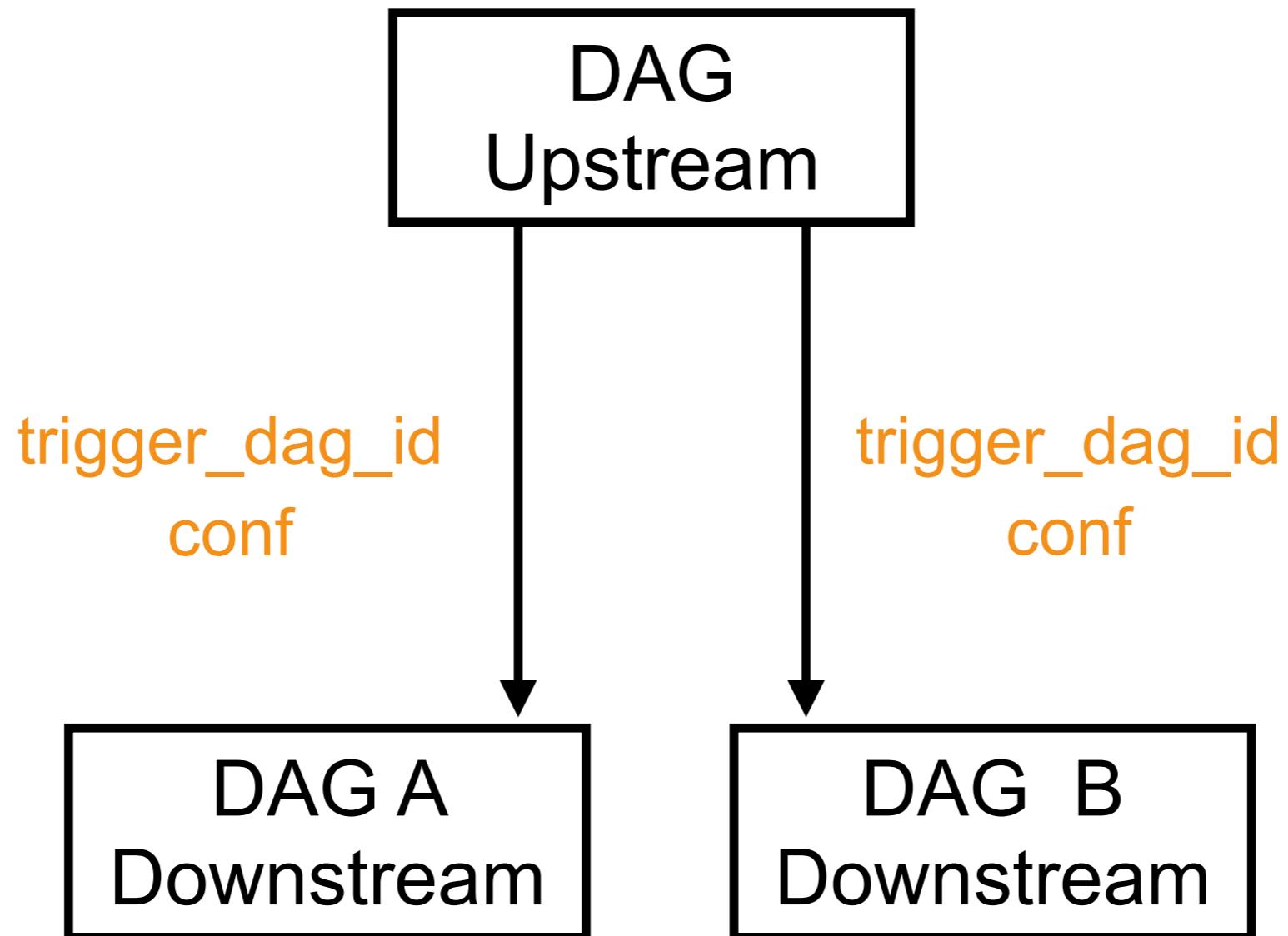
Managed Cross DAG

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

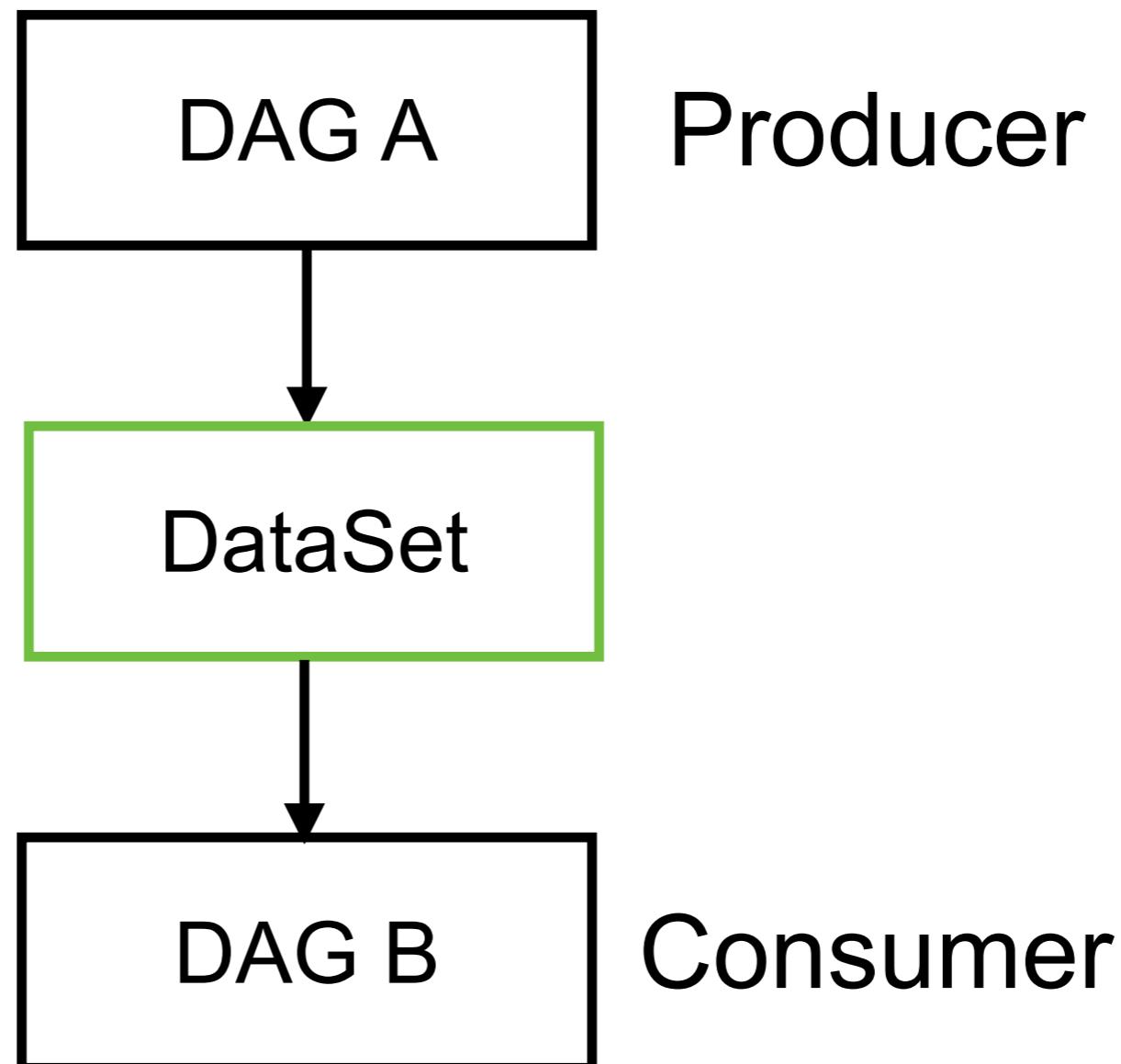
<https://medium.com/datamindedbe/cross-dag-dependencies-in-apache-airflow-a-comprehensive-guide-88cbc0bc68d0>



Trigger



Dataset (Event-driven)



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



Design Better Tasks



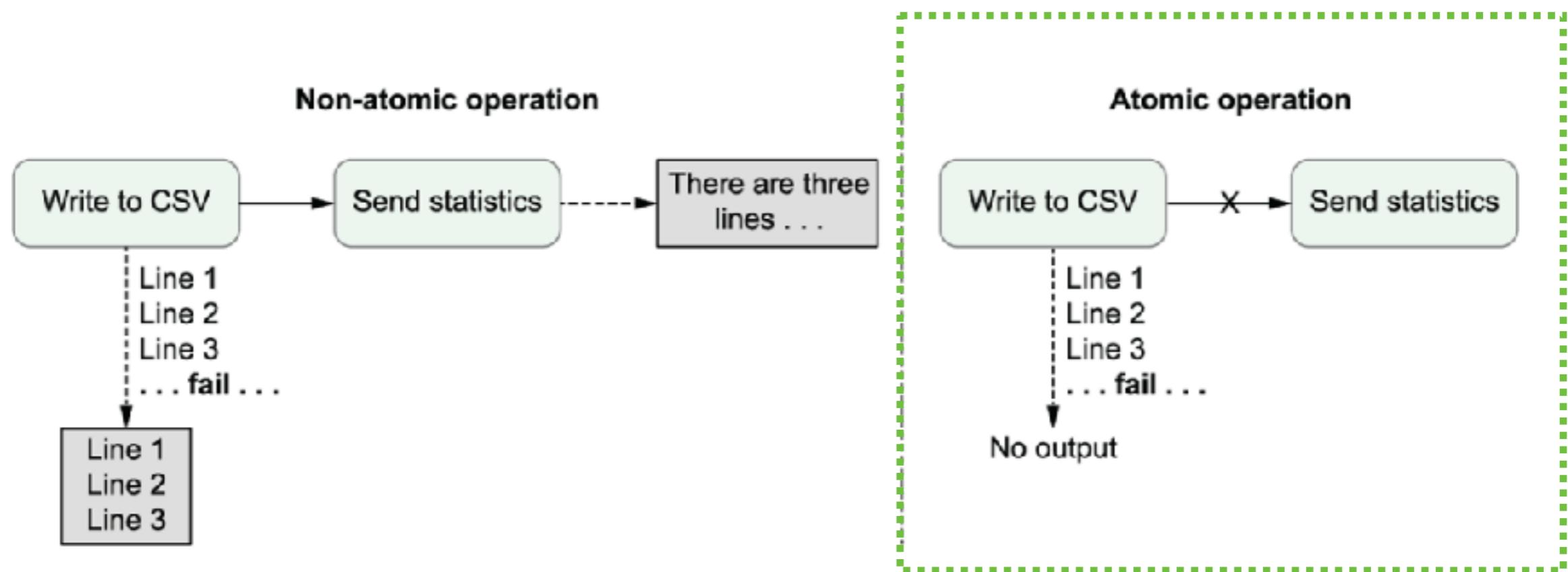
Design Better Tasks

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.

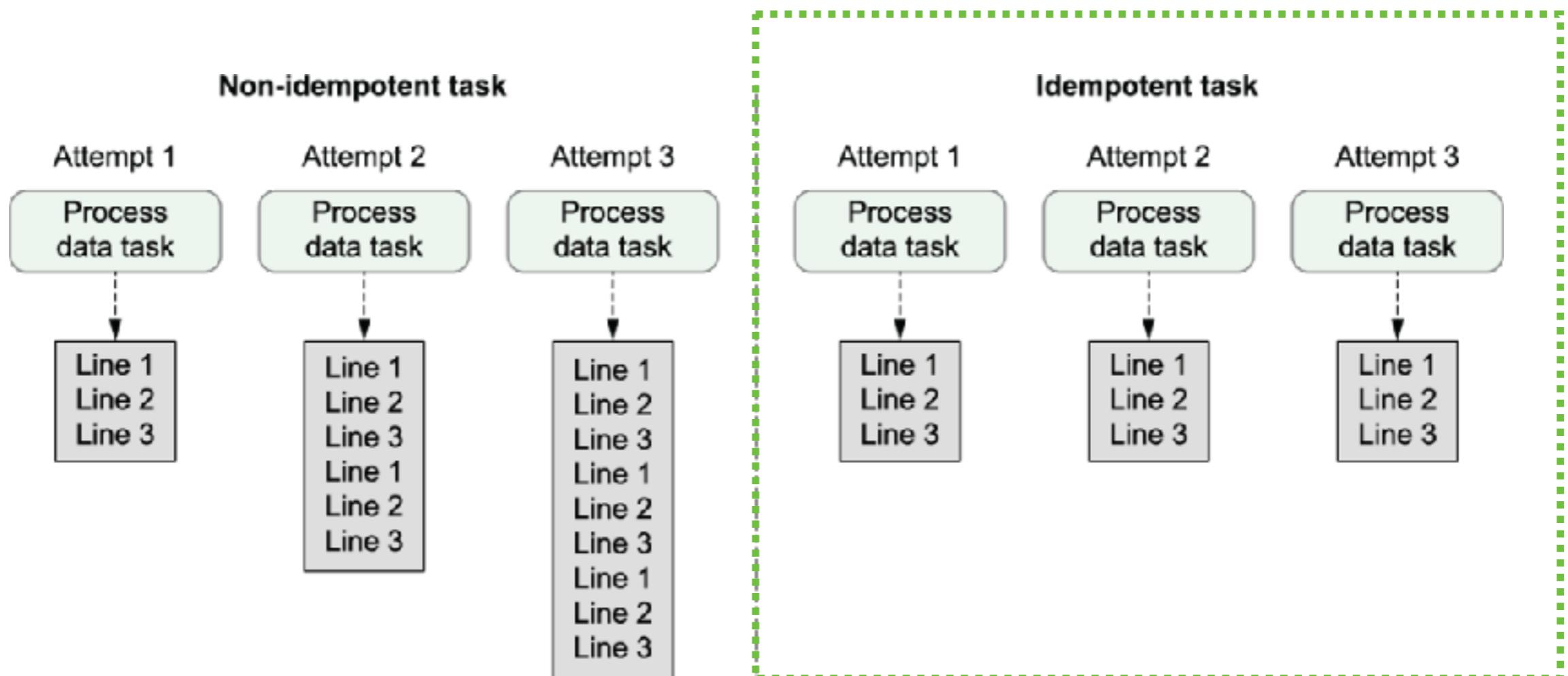
what are other
words for
idempotent?



unchanged, same, unvaried,
unvarying



Idempotency



Airflow 3.0 is coming

Anywhere

At any time

In any language

