Connections (link)

Хранят информацию о подключении к источнику

- **Создание через WebUI, RESTAPI, CLI, into DAGS**

  - **conn_id:** строковый идентификатор
  - Пароль шифруется с помощью Fernet

- **Используются в соответствующих операторах**

- **Доступны, в случае установленного пакета провайдера**

PostgresOperator

позволяет выполнять SQL-запросы и команды к базе данных PostgreSQL

postgres_conn_id

Идентификатор соединения

sql

Sql-запрос, который нужно выполнить

autocommit

Вкл или выкл автокоммиты

parameters

Параметры, передаваемые в SQL-запрос в качестве значений

PostgresOperator (link)

```python
from airflow import DAG
from airflow.providers.postgres.operators.postgres import PostgresOperator
from airflow.utils.dates import days_ago
from datetime import timedelta

default_args = {
    'owner': 'airflow',
    'start_date': days_ago(1),
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

with DAG(
        dag_id='postgres_example',
        default_args=default_args,
        schedule_interval=None,
        catchup=False,
        tags=['lect2', 'postgres_example']
) as dag:
    create_table = PostgresOperator(
        task_id='create_table',
        postgres_conn_id='postgres_default',
        sql='''
        CREATE TABLE IF NOT EXISTS public.lect_2 (
            name VARCHAR PRIMARY KEY,
            type VARCHAR,
            create_date VARCHAR
        );
        '''
    )

    create_table
```

Sensors (link)

инструменты, которые позволяют задавать условия для запуска задач в DAG (Directed Acyclic Graph) в зависимости от состояния других задач или внешних событий.

poke_interval

Интервал времени между проверками условия. Это время, через которое Sensor будет снова проверять условие.

timeout

Максимальное время ожидания выполнения условия. Если условие не выполнено в течение этого времени, Sensor будет считаться неудачным.

execution_timeout

Время, после которого выполнение сенсора будет прервано, независимо от того, завершилось ожидание или нет.

mode

Режим работы сенсора, который определяет, как будет происходить проверка условия (например, 'poke' для периодической проверки или 'reschedule' для отложенного запуска, если условие не выполнено).

soft_fail

Если установлен в True, то сенсор не будет считаться неудачным, если истекло время ожидания или условие не выполнено. По умолчанию False.

Sensors: example

FileSensor

```python
from airflow import DAG
from airflow.sensors.filesystem import FileSensor
from datetime import datetime, timedelta

default_args = {
    'owner': 'airflow',
    'start_date': datetime(2024, 1, 1),
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

with DAG('file_sensor_example', default_args=default_args, schedule_interval=None, catchup=False) as dag:
    file_sensor_task = FileSensor(
        task_id='wait_for_file',
        filepath='/path/to/your/file.txt',
        poke_interval=300,  # проверять каждые 5 минут
        timeout=600,  # максимальное время ожидания - 10 минут
    )
```

Sensors: example

HTTP Sensor

```python
from airflow import DAG
from airflow.sensors.http import HttpSensor
from datetime import datetime, timedelta

default_args = {
    'owner': 'airflow',
    'start_date': datetime(2024, 1, 1),
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

with DAG('http_sensor_example', default_args=default_args, schedule_interval=None, catchup=False) as dag:
    http_sensor_task = HttpSensor(
        task_id='wait_for_http',
        http_conn_id='http_default',
        endpoint='/health',
        request_params={},  # параметры запроса, если необходимо
        response_check=lambda response: response.status_code == 200,  # проверка успешности ответа
        poke_interval=60,  # проверять каждую минуту
        timeout=600,  # максимальное время ожидания - 10 минут
    )
```

Sensors: example

HivePartitionSensor

```
with DAG('hive_partition_sensor_example', default_args=default_args, schedule_interval=None, catchup=False) as dag:
    hive_sensor_task = HivePartitionSensor(
        task_id='wait_for_hive_partition',
        table='my_hive_table',  # Название вашей таблицы Hive
        partition_name='ds={{ ds }}',  # Имя и значение партиции, используя шаблон Jinja
        metastore_conn_id='hive_default',  # Идентификатор соединения с метастором Hive в Airflow
        poke_interval=300,  # Проверять каждые 5 минут
        timeout=600,  # Максимальное время ожидания - 10 минут
    )
```

Готовые сенсоры (link)

FileSensor
Наличие файла или директории

PythonSensor
Ф-ия вернула True

HivePartitionSensor
Наличие партиции

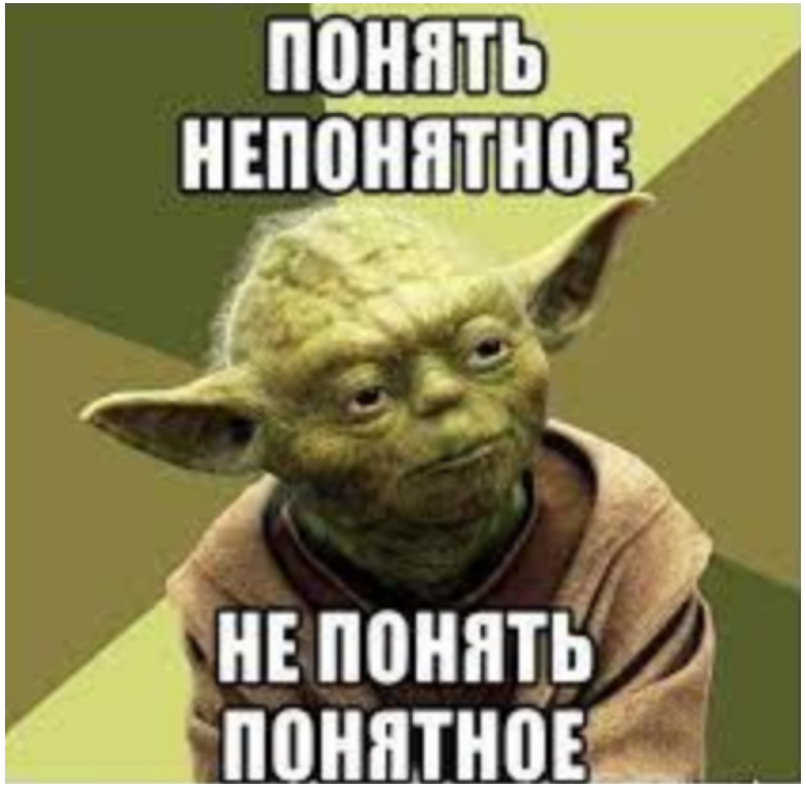
ExternalSensor
Результат выполнения другой задачи

Вопросы?

Sensors [«05_practice»]

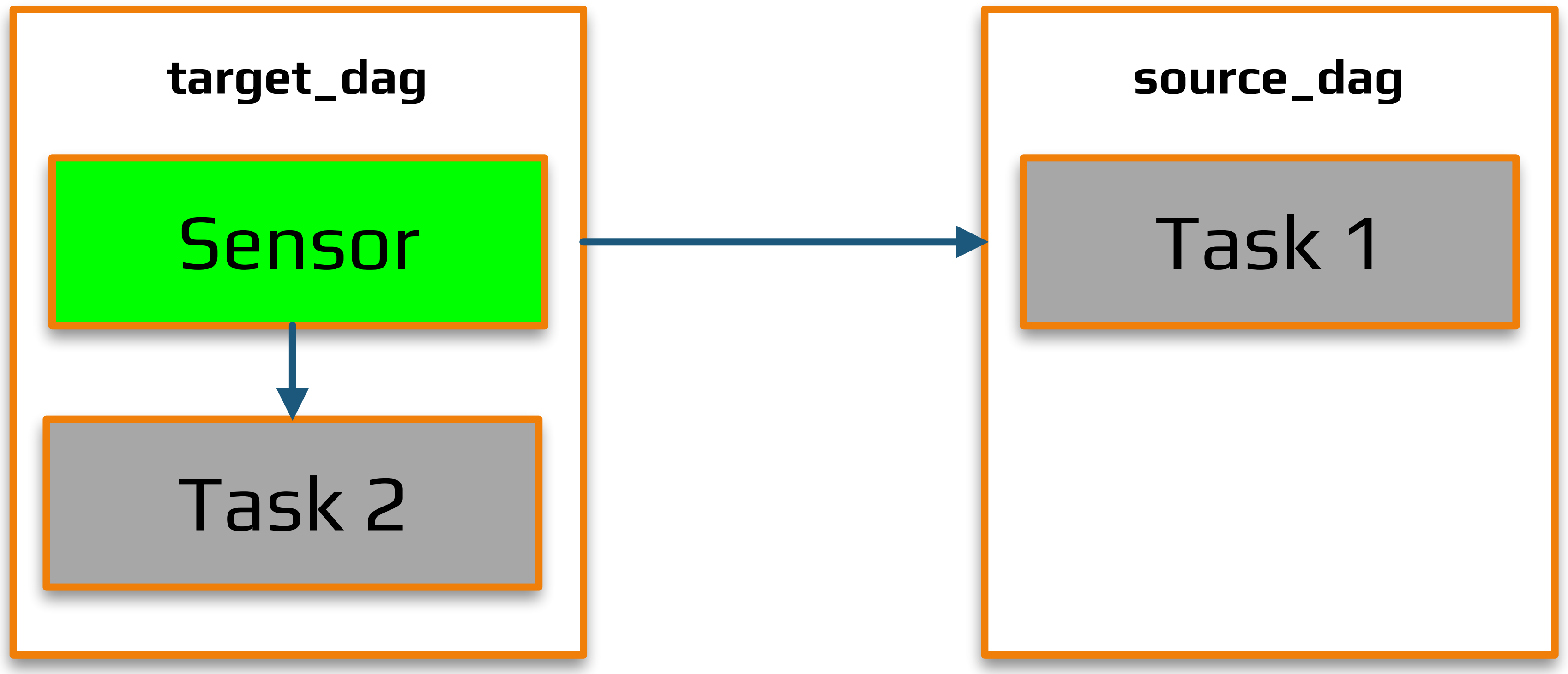
Текст задания находится на:
/practice_md/05_practice.md

ExternalTaskSensor

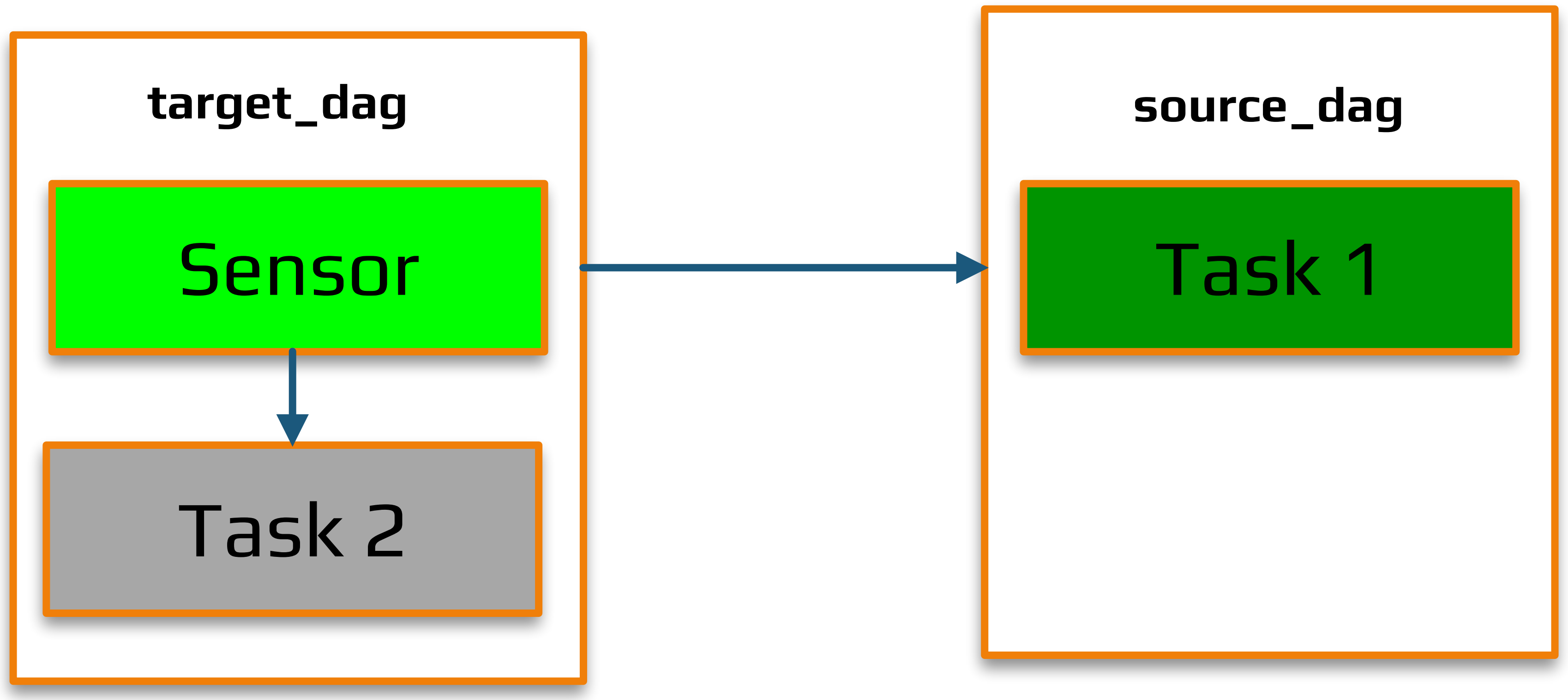


ExternalTaskSensor

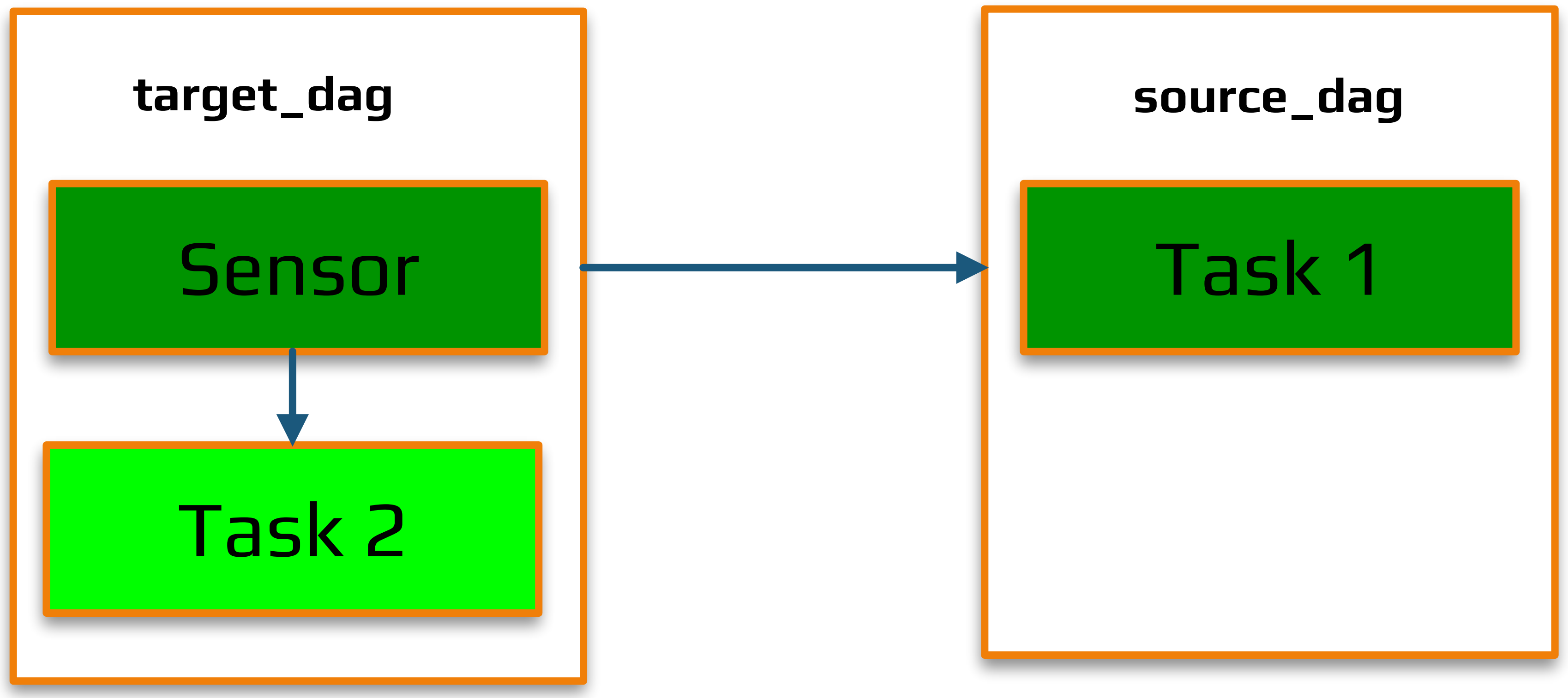
start_date=22:00
schedule_interval = */5 * * * *



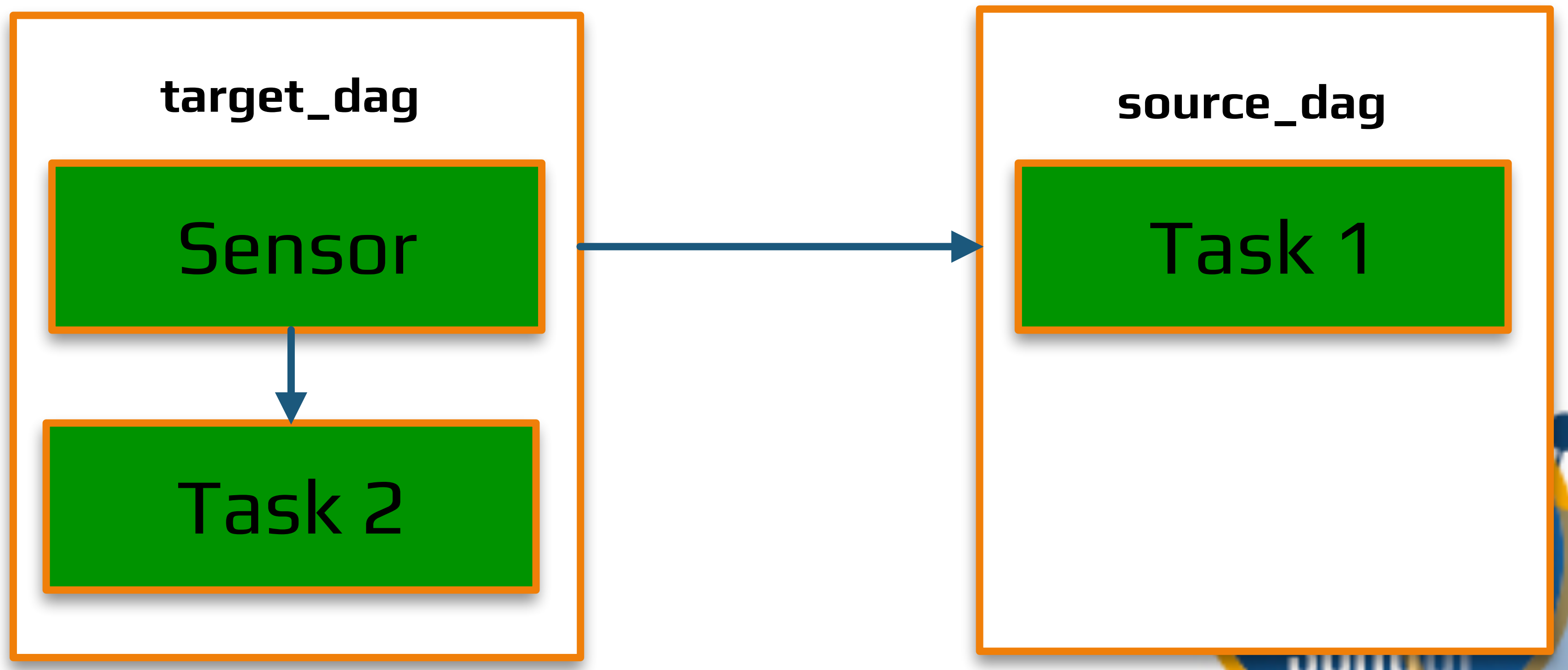
start_date=22:00
schedule_interval = */5 * * * *



start_date=22:00
schedule_interval = */5 * * * *



start_date=22:00
schedule_interval = */5 * * * *



ExternalTaskSensor

```
ets = ExternalTaskSensor(
    external_task_id="task",
    external_dag_id="dag"
)
```

Идеальный мир

Не должен запускаться вручную

Должен запускаться в одну и ту же дату, что и внешний даг

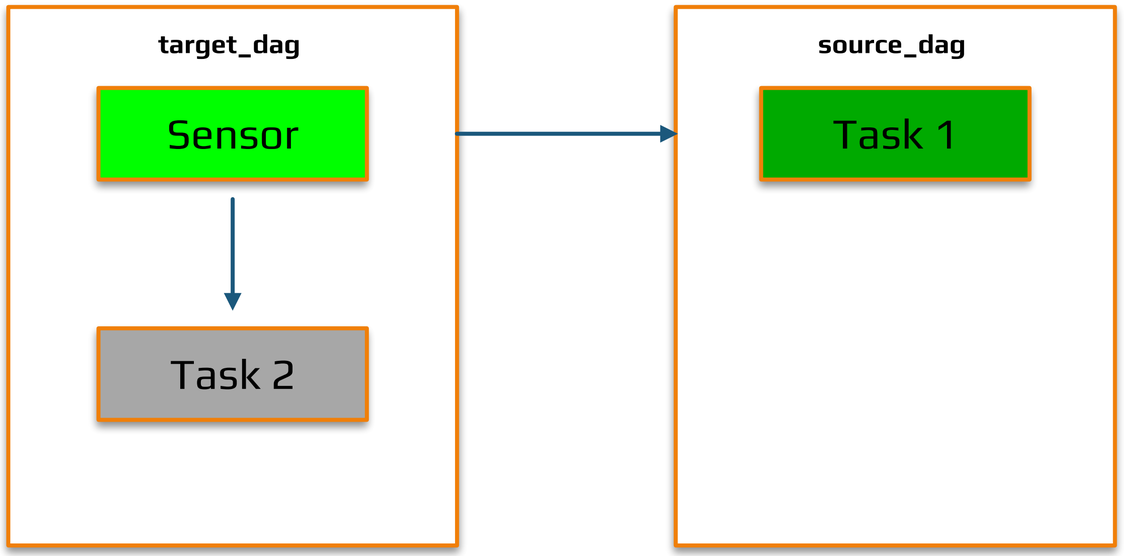
Должен иметь одинаковый интервал планирования со внешним дагом

ExternalTaskSensor

Но что будет, если:

start_date=22:00
schedule_interval = */5 * * * *

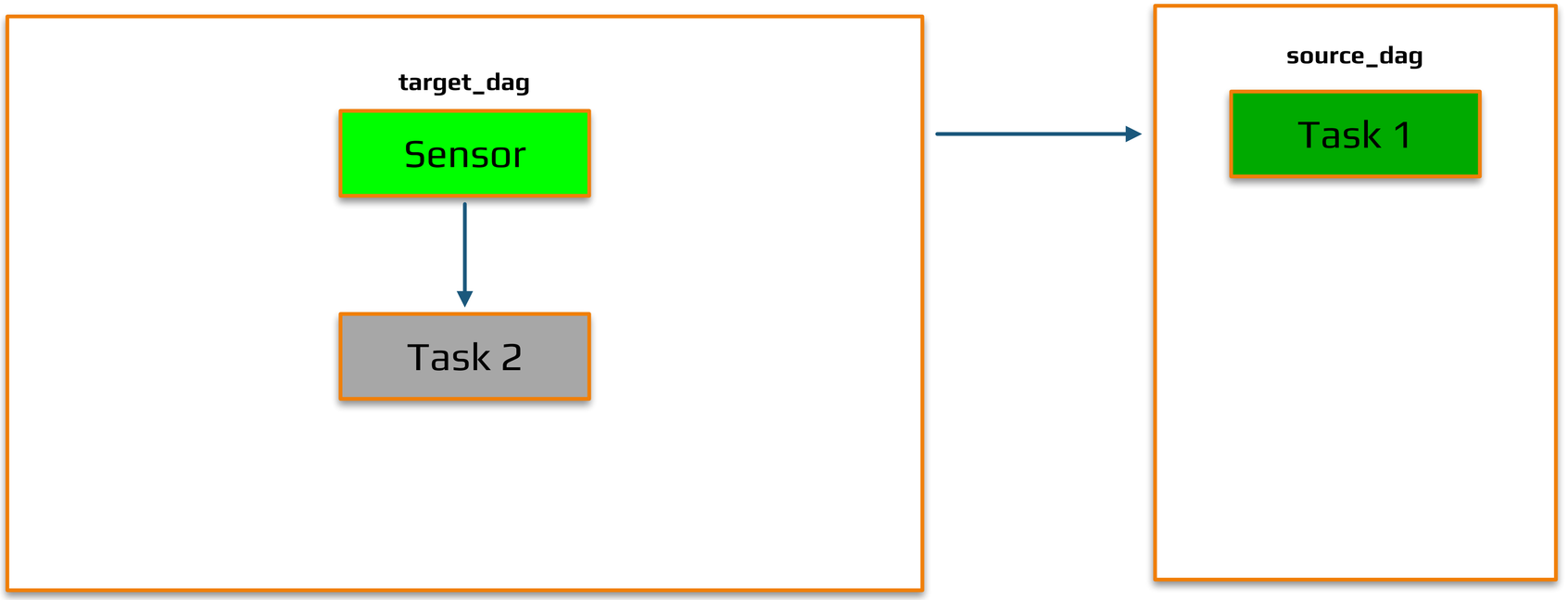
start_date=21:00
schedule_interval = */5 * * * *



ExternalTaskSensor

start_date=22:00
schedule_interval = */5 * * * *

start_date=21:00
schedule_interval = */5 * * * *

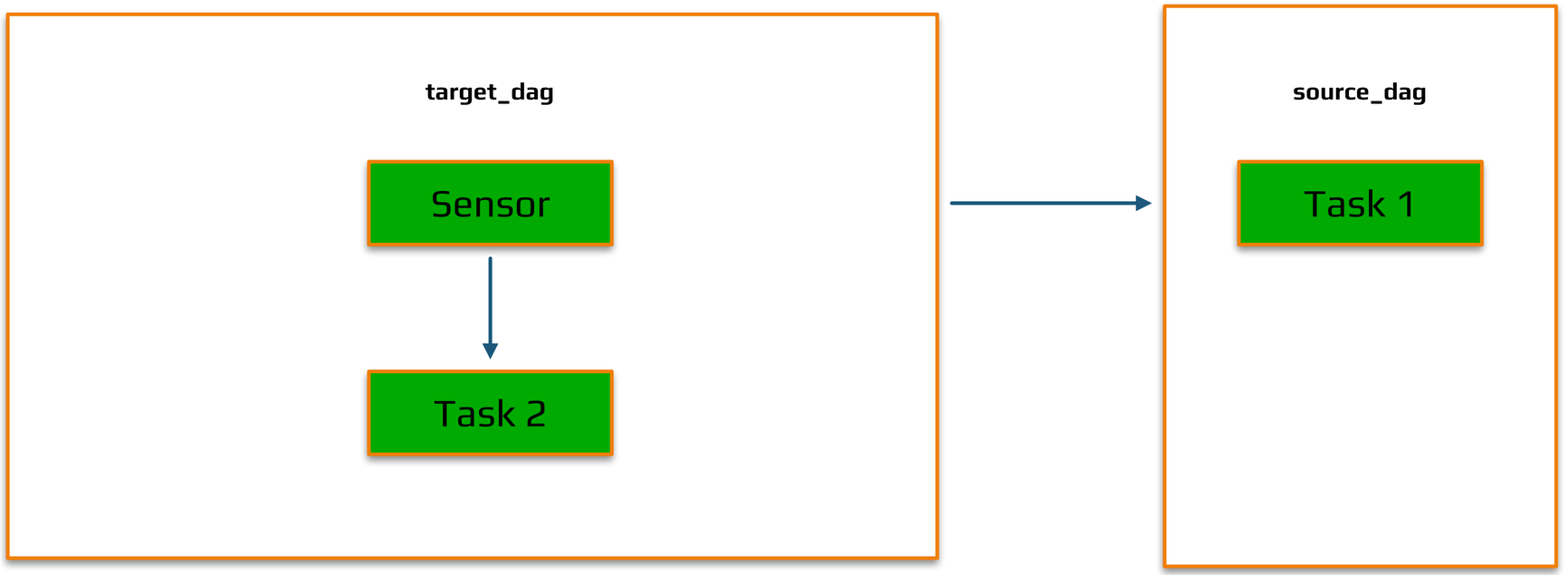


Poking for dag.task on 2023-01-01T22:00:00 ...

ExternalTaskSensor

start_date=22:00
schedule_interval = */5 * * * *
execution_delta=timedelta(hour=-1)

start_date=21:00
schedule_interval = */5 * * * *



Poking for dag.task on 2023-01-01T21:00:00 ...

ExternalTaskSensor

Что в итоге:

По умолчанию ExternalTaskSensor будет искать запуск source_dag, который имеет такой же execution_date, что и текущий запуск target_dag.

Это полезно, когда ваши DAG'и имеют синхронизированные расписания.

Trigger Rules (link)

all_success (default): All upstream tasks have succeeded

all_failed: All upstream tasks are in a **failed** or **upstream_failed** state

all_done: All upstream tasks are **done** with their execution

one_failed: At least one upstream task has **failed** (no wait)

one_success: At least one upstream task has **succeeded** (no wait)

none_failed: All upstream tasks have not **failed** or **upstream_failed** - that is, all upstream tasks have succeeded or been skipped

none_failed_min_one_success: All upstream tasks have **not failed** or **upstream_failed**, and at least one upstream task has **succeeded**.

none_skipped: No upstream task is in a **skipped** state - that is, all upstream tasks are in a success, failed, or upstream_failed state

always: No dependencies at all, run this task at any time

Trigger Rules (link)

Разбор примера

trigger_rules [«06_practice»]

Текст задания находится на:
/practice_md/06_practice.md