Reliable Airflow DAG Design when building a Time-series Data Lakehouse

Sung Yun

Enterprise Data Lake, Bloomberg

Airflow Summit

Let's flow together

September 19-21, 2023,
Toronto, Canada

Time-series Data

Disclaimer: dummy data

Date	Company ID	Price	P/E Ratio	Industry Sector
1992-09-16	Company A	15000	5	Tech
1992-09-16	Company B	5000	10	Tech
...	...	...	...	...
2023-09-18	Company A	23600	19	Tech
2023-09-18	Company B	19000	11	Tech
2023-09-19	Company A	23500	20	Tech
2023-09-19	Company B	21000	10	Tech

Human Expectations

“Daily starting Tuesday, September 19th, 2023, between 5:30 PM and 7:30 PM EDT”

Recurrence Pattern

Start Date and Time

Expected Delivery Window

Time-series Analysis

Disclaimer: dummy data

“Daily starting Tuesday, September 19th, 2023, between 5:30 PM and 7:30 PM EDT”

1992-09-16	Company B	5000	10	Tech
…	…	…	…	…
2023-09-18	Company A	23600	19	Tech
2023-09-18	Company B	19000	11	Tech

TechAtBloomberg.com

Bloomberg

Engineering

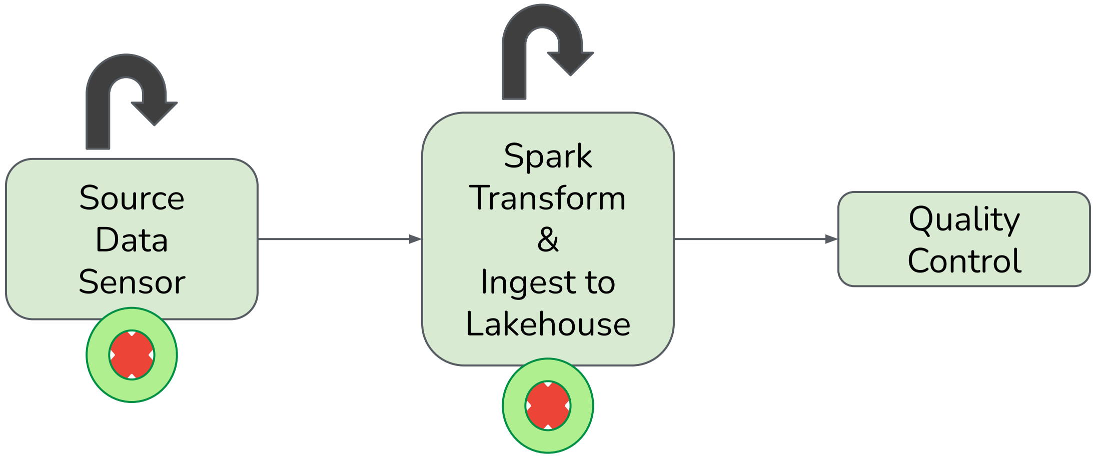
Designing for Reliability

- Recoverability
- Scalability
- Failure and Delay Detection (SLA Miss Detection)

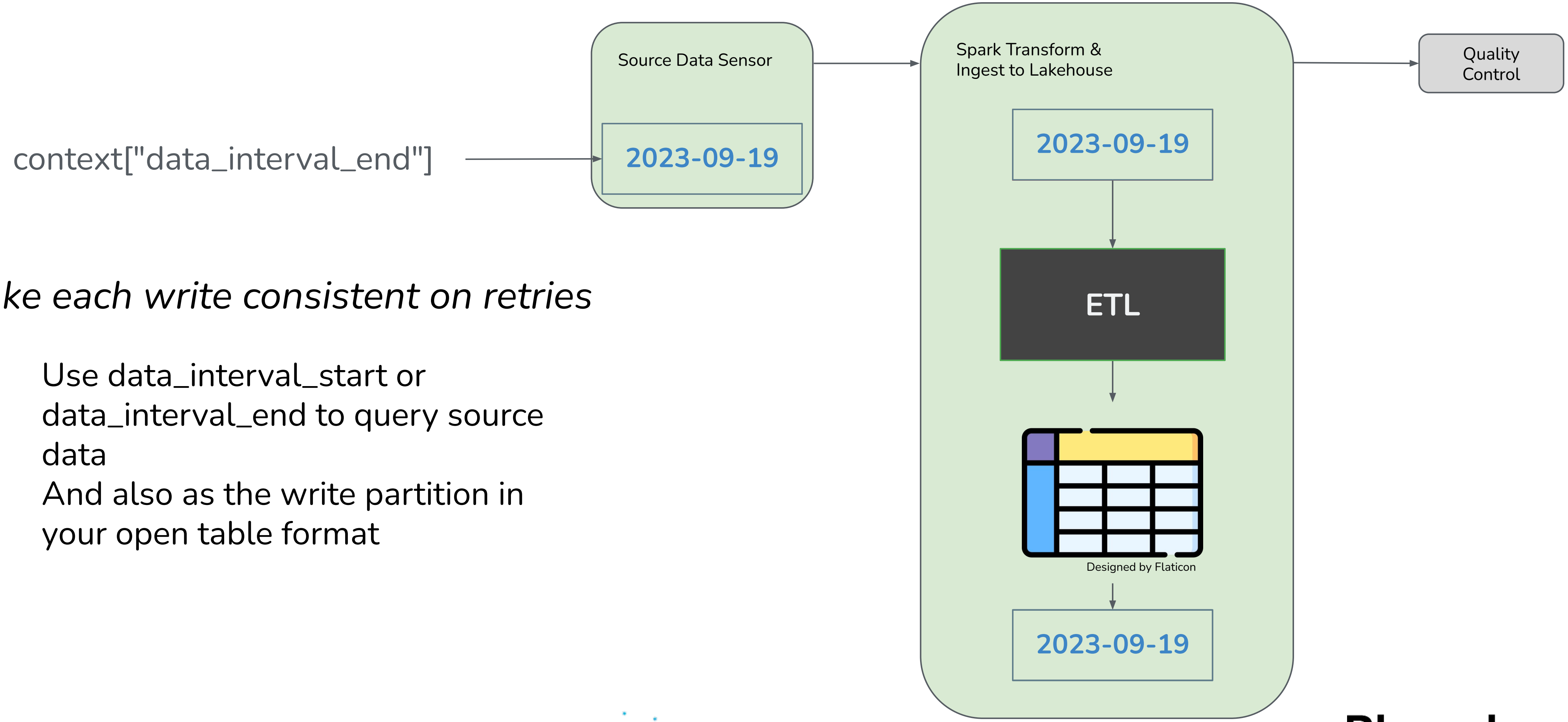
Recoverability

Making tasks safe to re run

Example DAG Run



Making Each Task Safe to Re-run



Make each write consistent on retries

1. Use data_interval_start or data_interval_end to query source data
2. And also as the write partition in your open table format

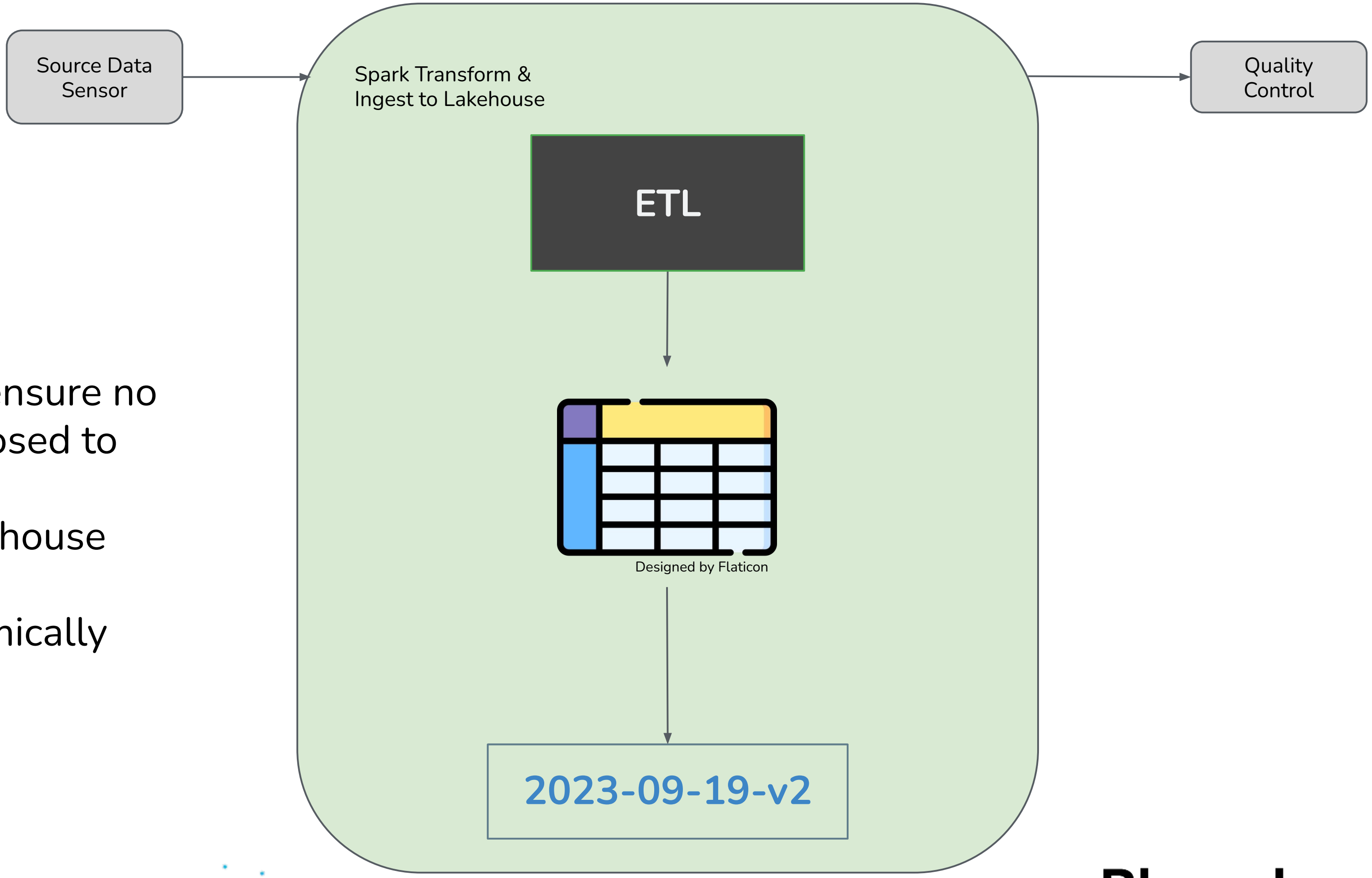
TechAtBloomberg.com

Bloomberg Engineering

Making Each Task Safe to Re-run

Make each write atomic

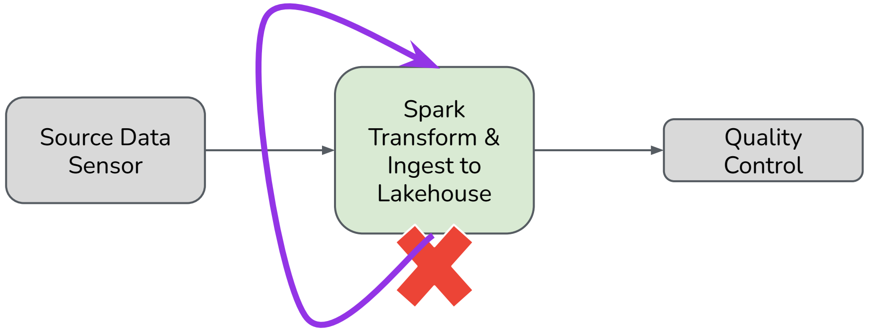
1. Use snapshot isolation to ensure no in-between states are exposed to other writers or readers
2. Use a write-optimized lakehouse partition strategy
3. Utilize operations that atomically replace an entire partition



TechAtBloomberg.com

Bloomberg Engineering

Retry on Failure



Enable Retries

1. Enable automatic task level retries with ‘retries’ parameter
2. Invoke a retry manually by ‘clearing’ a task

FAILED → *QUEUED*

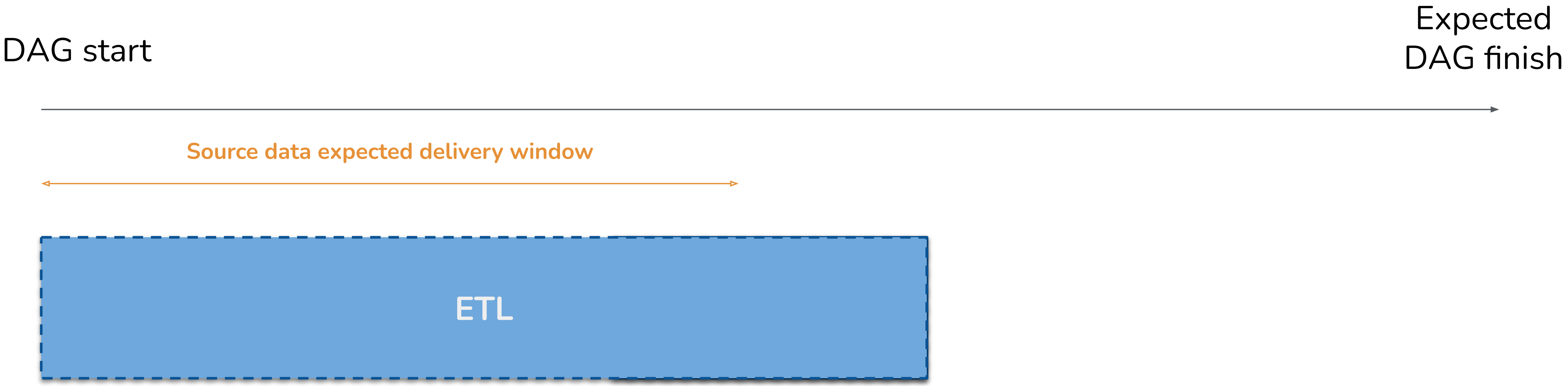
SUCCESS → *QUEUED*

```
with DAG(
    dag_id="my_dag",
    start_date=pendulum.datetime(2016, 1, 1),
    schedule="@daily",
    default_args={"retries": 2},
):
    op = BashOperator(
            task_id="hello_world",
            bash_command="Hello World!"
      )

    op2 = BashOperator(
            task_id="good_bye_cruel_world",
            bash_command="Good Bye!"
      )

    op >> op2
```

Scalability

Optimizing resource utilization

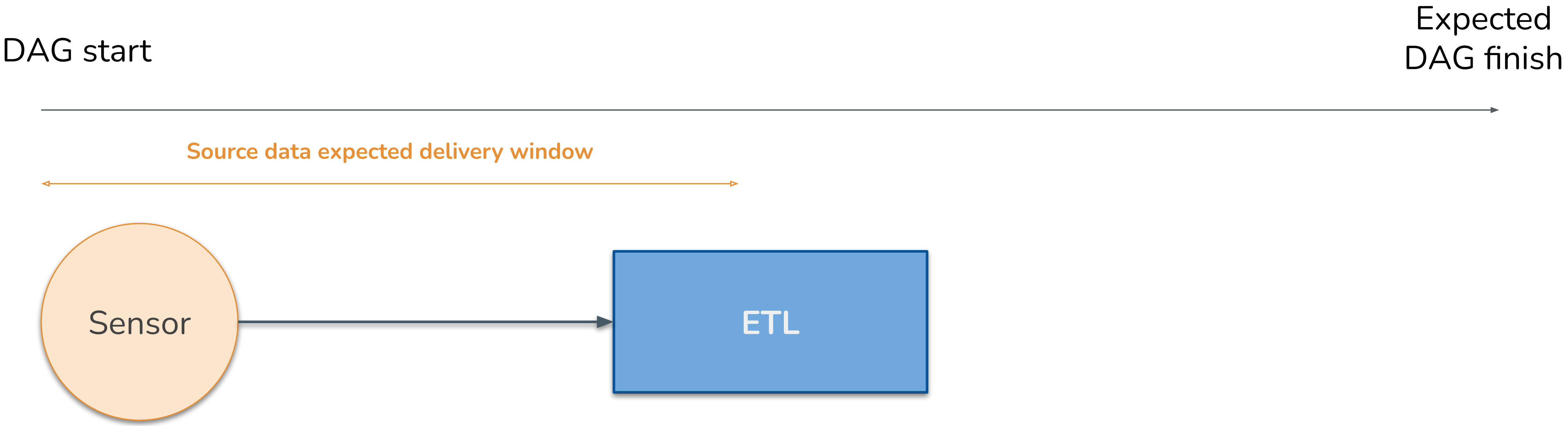
Scalability



TechAtBloomberg.com

Bloomberg Engineering

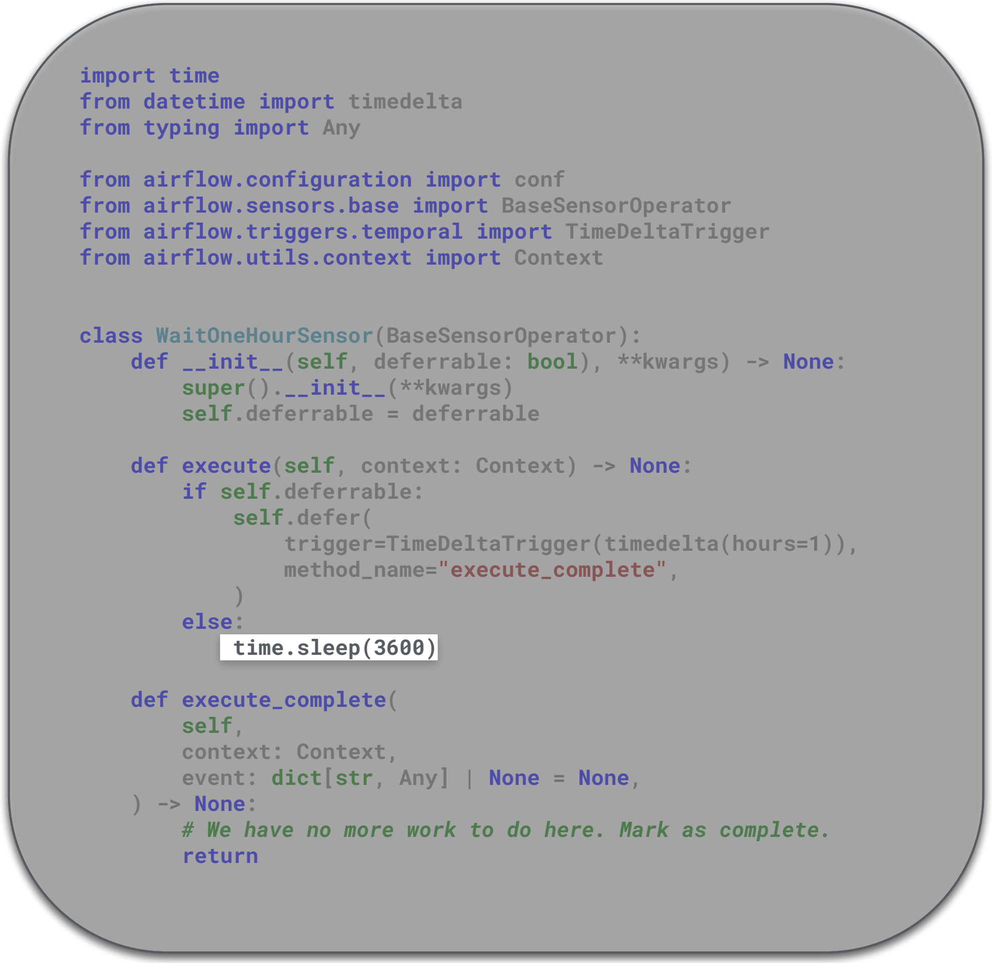
Hogging resources, only when we need them



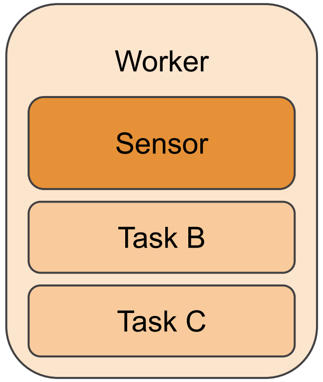
TechAtBloomberg.com

Bloomberg
Engineering

Sensors



```python
import time
from datetime import timedelta
from typing import Any

from airflow.configuration import conf
from airflow.sensors.base import BaseSensorOperator
from airflow.triggers.temporal import TimeDeltaTrigger
from airflow.utils.context import Context


class WaitOneHourSensor(BaseSensorOperator):
    def __init__(self, deferrable: bool), **kwargs) -> None:
        super().__init__(**kwargs)
        self.deferrable = deferrable

    def execute(self, context: Context) -> None:
        if self.deferrable:
            self.defer(
                trigger=TimeDeltaTrigger(timedelta(hours=1)),
                method_name="execute_complete",
            )
        else:
            time.sleep(3600)

    def execute_complete(
        self,
        context: Context,
        event: dict[str, Any] | None = None,
    ) -> None:
        # We have no more work to do here. Mark as complete.
        return
```



Worker

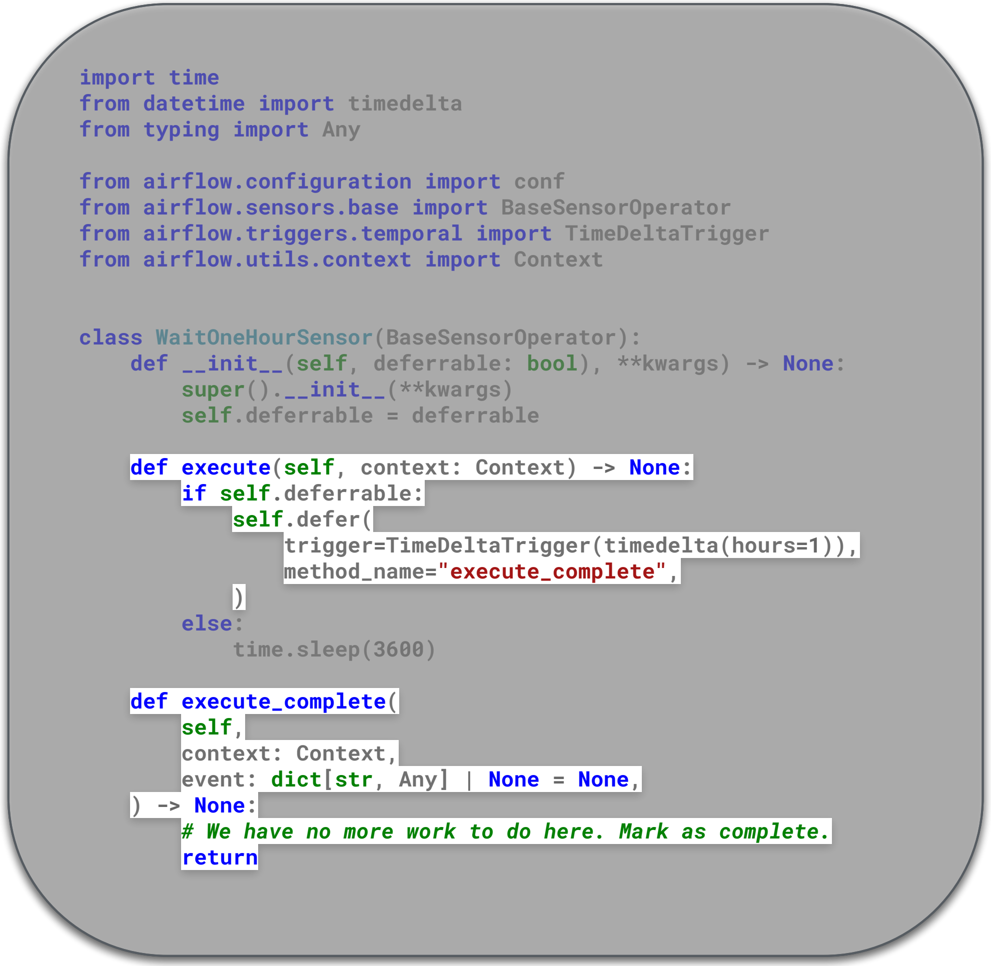
Sensor

Task B

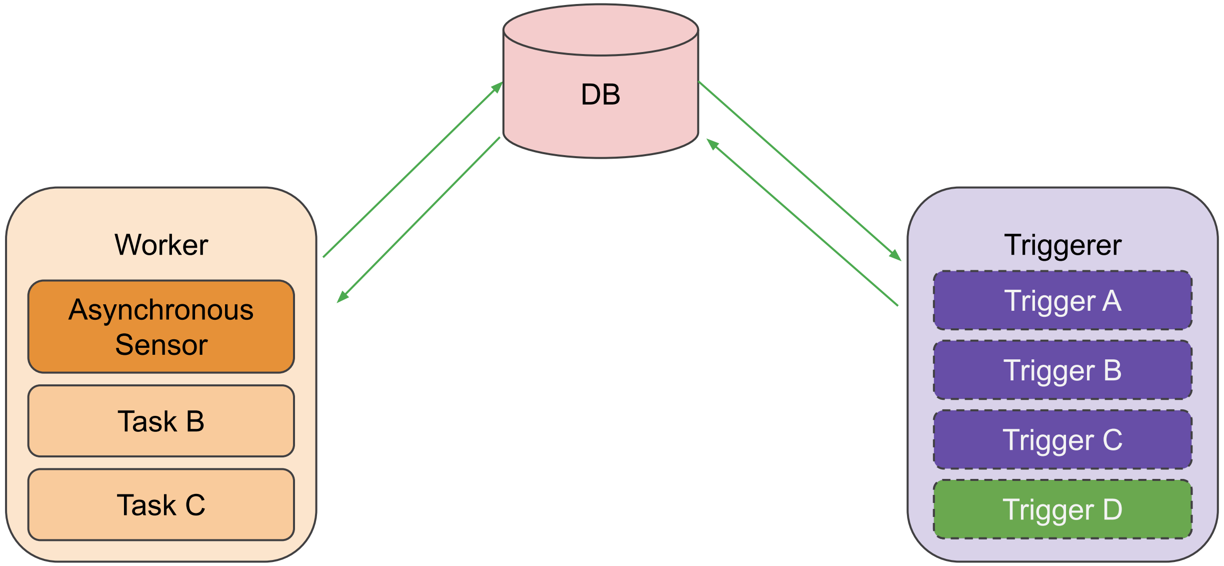
Task C

https://airflow.apache.org/docs/apache-airflow/stable/authoring-and-scheduling/deferring.html#deferrable-operators-triggers

Asynchronous Sensors (Deferrable)



```python
import time
from datetime import timedelta
from typing import Any

from airflow.configuration import conf
from airflow.sensors.base import BaseSensorOperator
from airflow.triggers.temporal import TimeDeltaTrigger
from airflow.utils.context import Context


class WaitOneHourSensor(BaseSensorOperator):
    def __init__(self, deferrable: bool), **kwargs) -> None:
        super().__init__(**kwargs)
        self.deferrable = deferrable

    def execute(self, context: Context) -> None:
        if self.deferrable:
            self.defer(
                trigger=TimeDeltaTrigger(timedelta(hours=1)),
                method_name="execute_complete",
            )
        else:
            time.sleep(3600)

    def execute_complete(
        self,
        context: Context,
        event: dict[str, Any] | None = None,
    ) -> None:
        # We have no more work to do here. Mark as complete.
        return
```



https://airflow.apache.org/docs/apache-airflow/stable/authoring-and-scheduling/deferring.html#deferrable-operators-triggers

TechAtBloomberg.com

Bloomberg

Engineering

Failure and Delay Detection

TechAtBloomberg.com

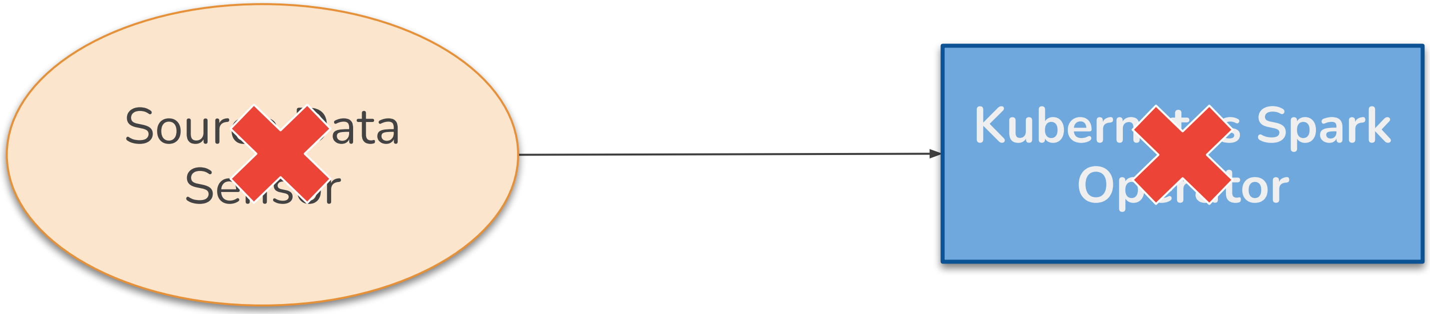
Bloomberg

Engineering

Failure Detection

```
dag.dagrun_timeout: timedelta
task.execution_timeout: timedelta
```

```
task.on_failure_callback: Callable
```



Accountable Owner

Failed:
- ➔ S3 client failure - rate limit
- ➔ Credentials issue

Delay Detection:
- ➔ Source File is not available in expected window

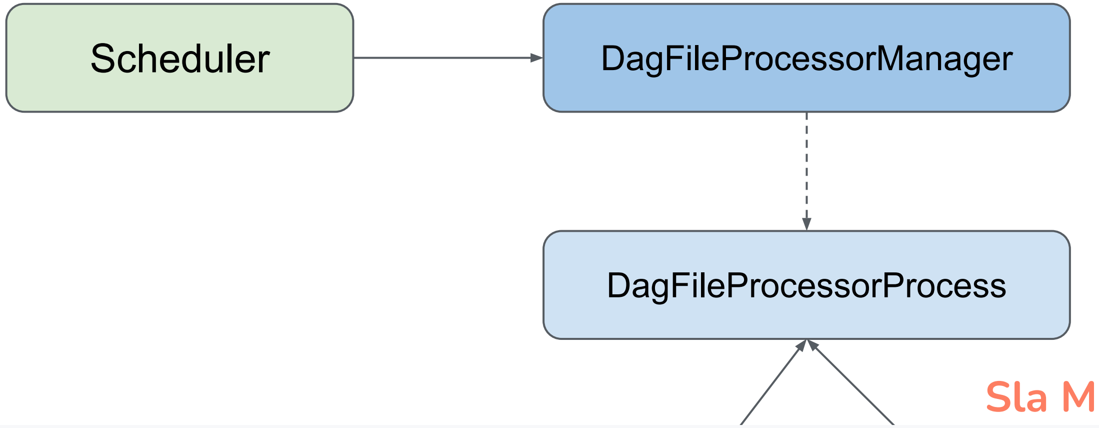
Failed:
- ➔ S3 client failure - rate limit
- ➔ Credentials issue
- ➔ Spark SQL transformation failure

Delay Detection:
- ➔ Undefined hanging Kubernetes Spark Job

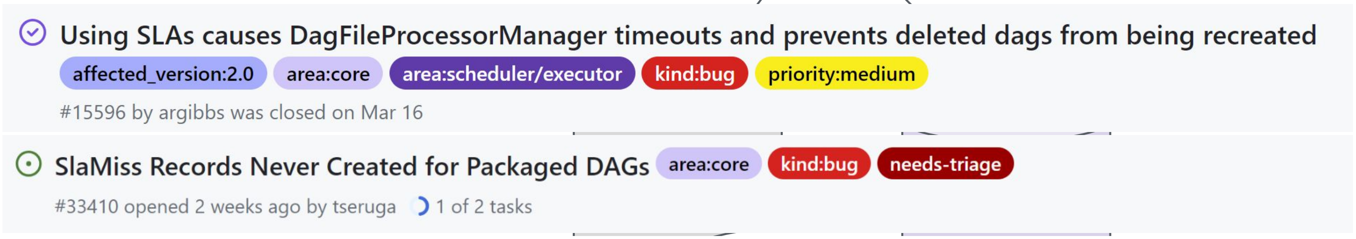
SLAs in Airflow (expected time of completion)

sort of works... with a lot of confusion... and with a lot of flaws

```
if start_date + sla < timezone.utcnow():
    sla_missed = True
```



Sla Missed??

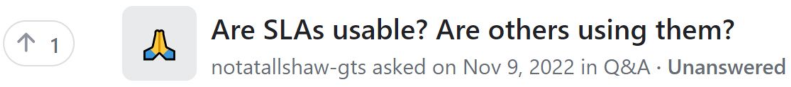


Using SLAs causes DagFileProcessorManager timeouts and prevents deleted dags from being recreated affected_version:2.0 area:core area:scheduler/executor kind:bug priority:medium

#15596 by argibbs was closed on Mar 16

SlaMiss Records Never Created for Packaged DAGs area:core kind:bug needs-triage

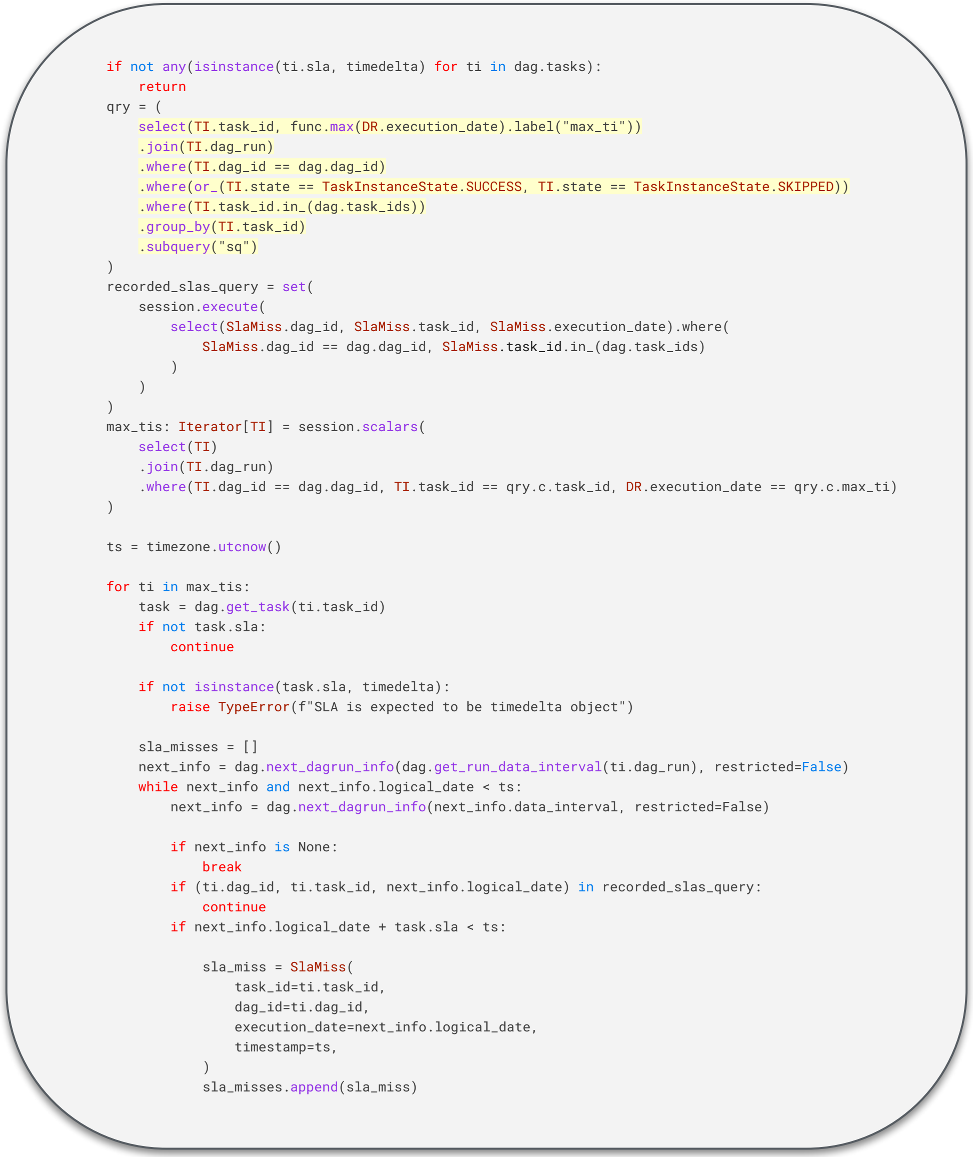
#33410 opened 2 weeks ago by tseruga 1 of 2 tasks



↑ 1 🙏 Are SLAs usable? Are others using them?

notatallshaw-gts asked on Nov 9, 2022 in Q&A · Unanswered

airflow.dag_processing.processor



```
if not any(isinstance(ti.sla, timedelta) for ti in dag.tasks):
    return
qry = (
    select(TI.task_id, func.max(DR.execution_date).label("max_ti"))
    .join(TI.dag_run)
    .where(TI.dag_id == dag.dag_id)
    .where(or_(TI.state == TaskInstanceState.SUCCESS, TI.state == TaskInstanceState.SKIPPED))
    .where(TI.task_id.in_(dag.task_ids))
    .group_by(TI.task_id)
    .subquery("sq")
)
recorded_slas_query = set(
    session.execute(
        select(SlaMiss.dag_id, SlaMiss.task_id, SlaMiss.execution_date).where(
            SlaMiss.dag_id == dag.dag_id, SlaMiss.task_id.in_(dag.task_ids)
        )
    )
)
max_tis: Iterator[TI] = session.scalars(
    select(TI)
    .join(TI.dag_run)
    .where(TI.dag_id == dag.dag_id, TI.task_id == qry.c.task_id, DR.execution_date == qry.c.max_ti)
)

ts = timezone.utcnow()

for ti in max_tis:
    task = dag.get_task(ti.task_id)
    if not task.sla:
        continue

    if not isinstance(task.sla, timedelta):
        raise TypeError(f"SLA is expected to be timedelta object")

    sla_misses = []
    next_info = dag.next_dagrun_info(dag.get_run_data_interval(ti.dag_run), restricted=False)
    while next_info and next_info.logical_date < ts:
        next_info = dag.next_dagrun_info(next_info.data_interval, restricted=False)

        if next_info is None:
            break
        if (ti.dag_id, ti.task_id, next_info.logical_date) in recorded_slas_query:
            continue
        if next_info.logical_date + task.sla < ts:

            sla_miss = SlaMiss(
                task_id=ti.task_id,
                dag_id=ti.dag_id,
                execution_date=next_info.logical_date,
                timestamp=ts,
            )
            sla_misses.append(sla_miss)
```

DAG or Task-level Feature?

Callback defined at DAG level,

but evaluated for each task

```
@dag(
    schedule="*/2 * * * *",
    start_date=pendulum.datetime(2021, 1, 1,
tz="UTC"),
    catchup=False,
    sla_miss_callback=sla_callback,
    default_args={"email": "email@example.com"},
)
def example_sla_dag():
    @task(sla=datetime.timedelta(seconds=10))
    def sleep_20():
        """Sleep for 20 seconds"""
        time.sleep(20)

    @task
    def sleep_30():
        """Sleep for 30 seconds"""
        time.sleep(30)

    sleep_20() >> sleep_30()


example_dag = example_sla_dag()
```

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

Different Function Signature from Other Callbacks

Others

```python
def task_failure_alert(context):
    print(
         f"Task has failed, "
         "ti_key_str: {context['task_instance_key_str']}"
        )

def dag_success_alert(context):
    print(
         f"DAG has succeeded, "
         "run_id: {context['run_id']}"
        )
```

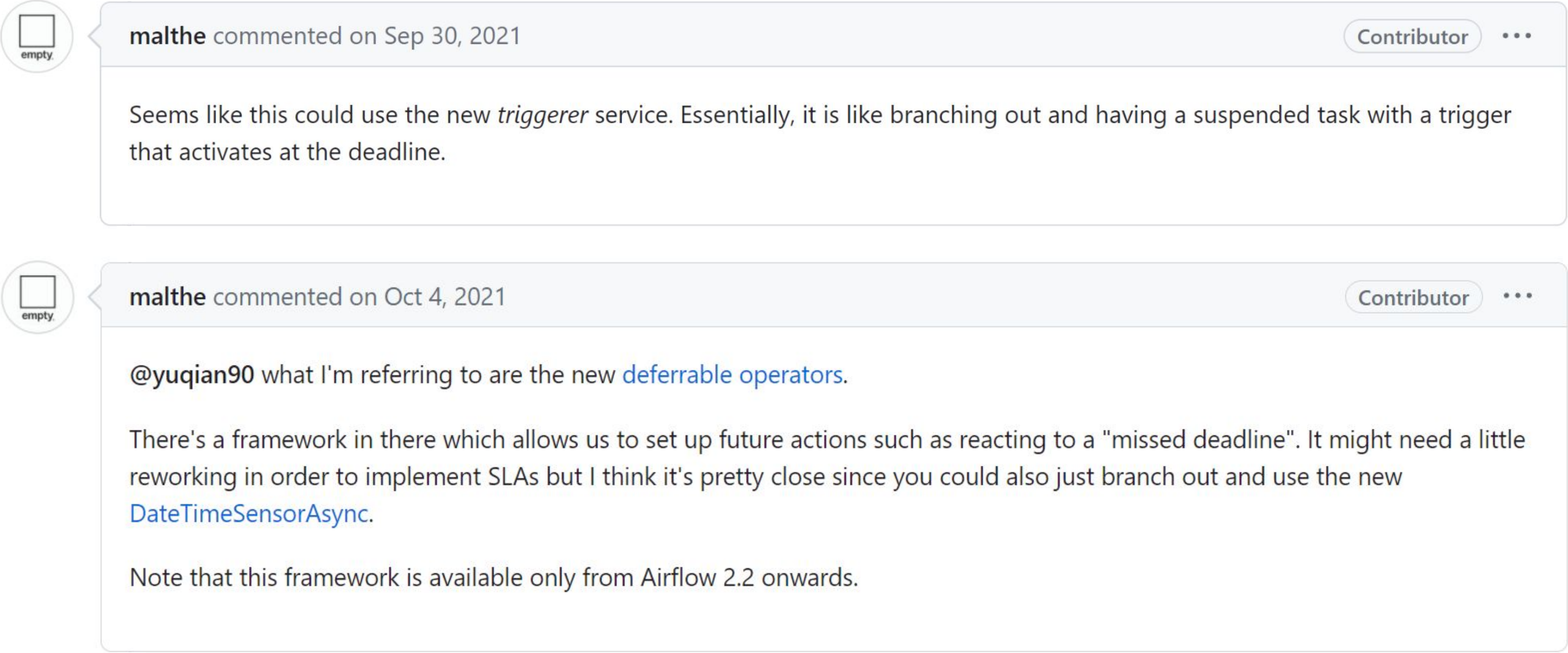
https://airflow.apache.org/docs/apache-airflow/stable/administration-and-deployment/logging-monitoring/callbacks.html#

SLA

```python
def sla_callback(
    dag,
    task_list,
    blocking_task_list,
    slas,
    blocking_tis
):
    print(
        "The callback arguments are: ",
        {
            "dag": dag,
            "task_list": task_list,
            "blocking_task_list": blocking_task_list,
            "slas": slas,
            "blocking_tis": blocking_tis,
        },
       )
```

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

Custom SLAs



malthe commented on Sep 30, 2021

Seems like this could use the new *triggerer* service. Essentially, it is like branching out and having a suspended task with a trigger that activates at the deadline.

malthe commented on Oct 4, 2021

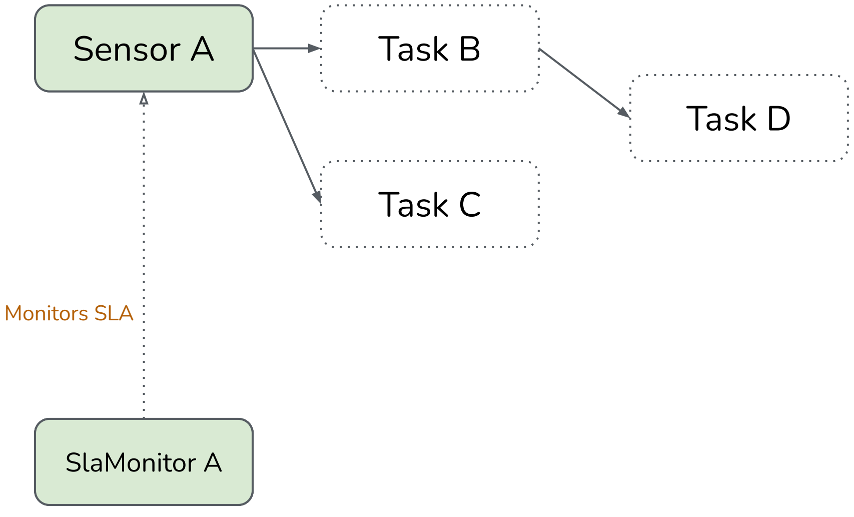
@yuqian90 what I'm referring to are the new deferrable operators.

There's a framework in there which allows us to set up future actions such as reacting to a "missed deadline". It might need a little reworking in order to implement SLAs but I think it's pretty close since you could also just branch out and use the new DateTimeSensorAsync.

Note that this framework is available only from Airflow 2.2 onwards.

Custom Operator: SLAMonitor

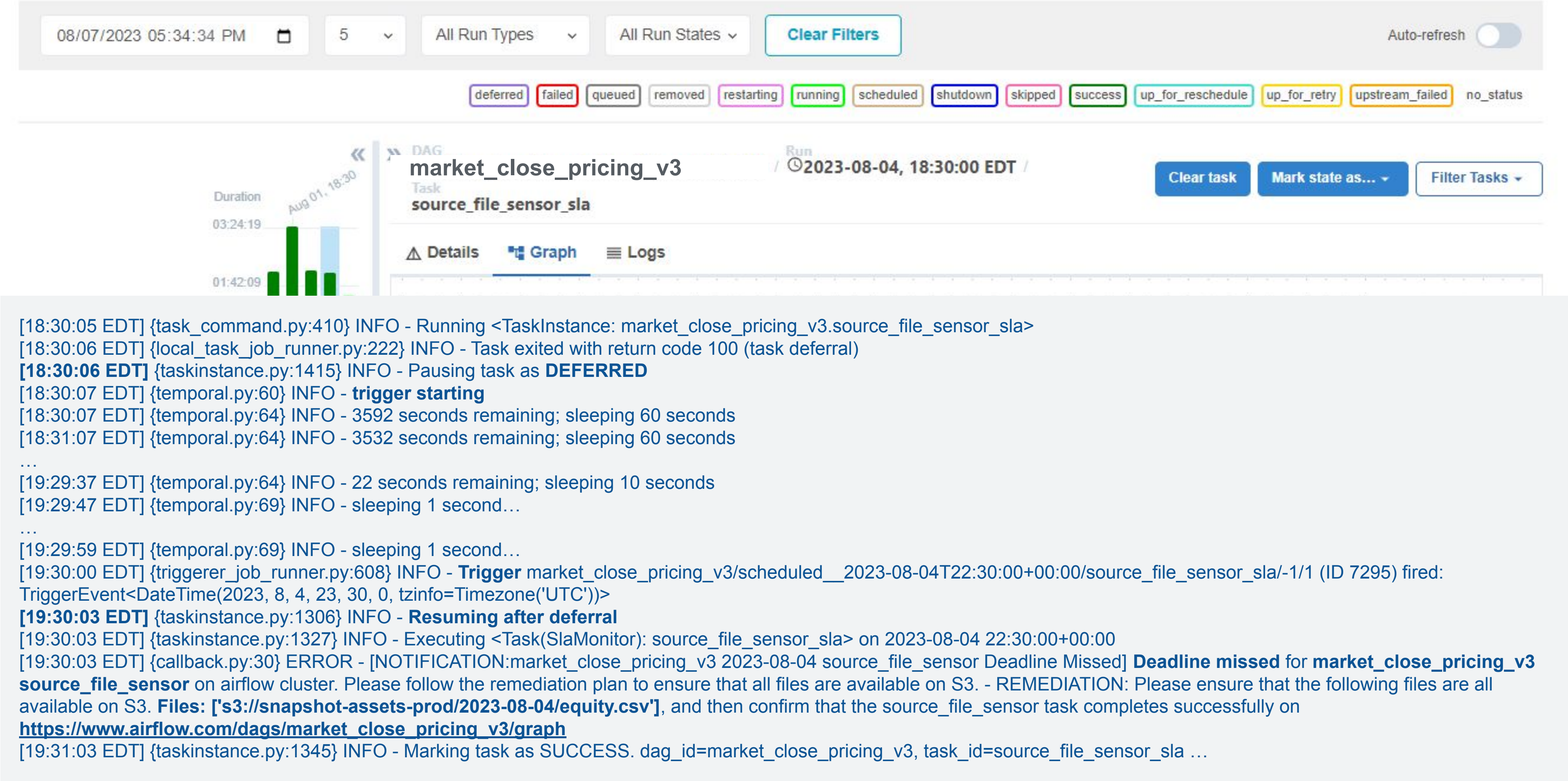
```python
from datetime import timedelta
from typing import Callable

from airflow.models.baseoperator import BaseOperator
from airflow.triggers.temporal import DateTimeTrigger
from airflow.utils.state import TaskInstanceState


class SlaMonitor(BaseOperator):
    def __init__(
        self,
        sla: timedelta,
        target_task_id: str,
        callback: Callable,
        **kwargs,
    ):
        super().__init__(**kwargs)
        self.sla = sla
        self.target_task_id = target_task_id

    def execute(self, context):
        # or define the SLA however else you see fit
        deadline = context[‘data_interval_end’] + self.sla
        self.defer(trigger=DateTimeTrigger(deadline), method_name=’execute_complete’)

    def execute_complete(self, context, event=None):
        ti = context[‘dagrun’].get_task_instance(self.target_task_id)
        if ti and ti.state == TaskInstanceState.SUCCESS:
            self.log.info("SUCCEEDED")
            return
        else:
            # replace this with your choice of sla callback function
            self.log.error("Sla was just missed!!")
            self.callback(context)
            return
```



Executes callback -> “Sla was just missed!!”

Deferred for 1 hour

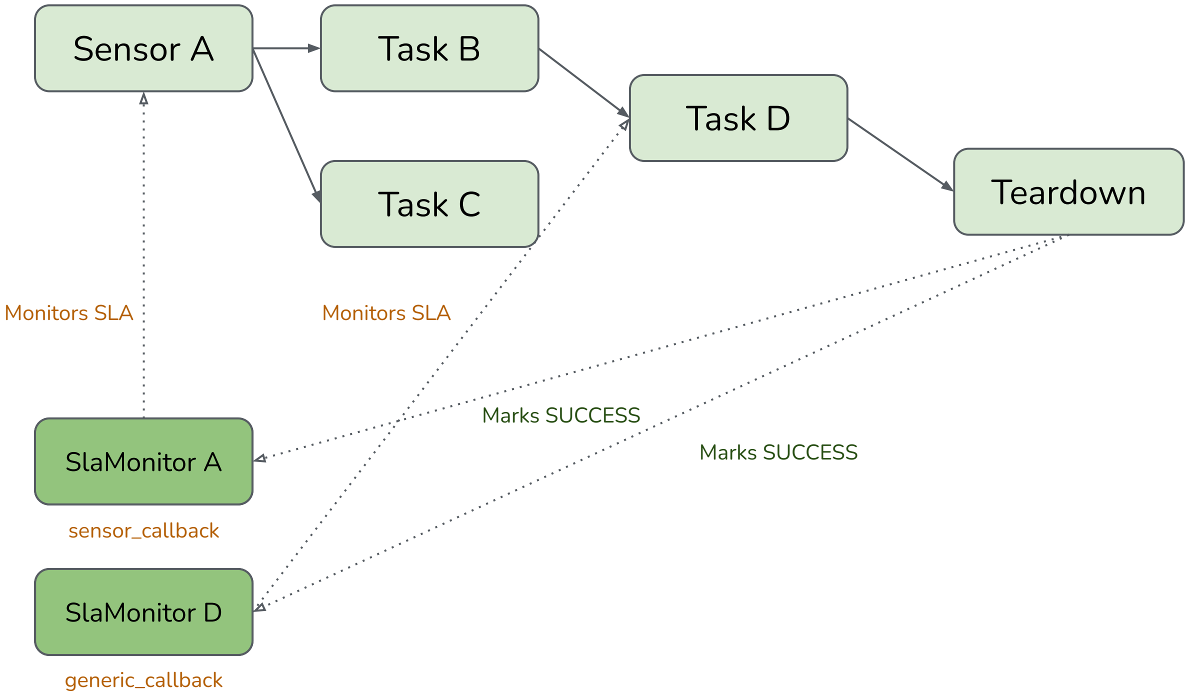
Custom Operator: SLAMonitor



```
[18:30:05 EDT] {task_command.py:410} INFO - Running <TaskInstance: market_close_pricing_v3.source_file_sensor_sla>
[18:30:06 EDT] {local_task_job_runner.py:222} INFO - Task exited with return code 100 (task deferral)
[18:30:06 EDT] {taskinstance.py:1415} INFO - Pausing task as DEFERRED
[18:30:07 EDT] {temporal.py:60} INFO - trigger starting
[18:30:07 EDT] {temporal.py:64} INFO - 3592 seconds remaining; sleeping 60 seconds
[18:31:07 EDT] {temporal.py:64} INFO - 3532 seconds remaining; sleeping 60 seconds
…
[19:29:37 EDT] {temporal.py:64} INFO - 22 seconds remaining; sleeping 10 seconds
[19:29:47 EDT] {temporal.py:69} INFO - sleeping 1 second…
…
[19:29:59 EDT] {temporal.py:69} INFO - sleeping 1 second…
[19:30:00 EDT] {triggerer_job_runner.py:608} INFO - Trigger market_close_pricing_v3/scheduled__2023-08-04T22:30:00+00:00/source_file_sensor_sla/-1/1 (ID 7295) fired:
TriggerEvent<DateTime(2023, 8, 4, 23, 30, 0, tzinfo=Timezone('UTC'))>
[19:30:03 EDT] {taskinstance.py:1306} INFO - Resuming after deferral
[19:30:03 EDT] {taskinstance.py:1327} INFO - Executing <Task(SlaMonitor): source_file_sensor_sla> on 2023-08-04 22:30:00+00:00
[19:30:03 EDT] {callback.py:30} ERROR - [NOTIFICATION:market_close_pricing_v3 2023-08-04 source_file_sensor Deadline Missed] Deadline missed for market_close_pricing_v3
source_file_sensor on airflow cluster. Please follow the remediation plan to ensure that all files are available on S3. - REMEDIATION: Please ensure that the following files are all
available on S3. Files: ['s3://snapshot-assets-prod/2023-08-04/equity.csv'], and then confirm that the source_file_sensor task completes successfully on
https://www.airflow.com/dags/market_close_pricing_v3/graph
[19:31:03 EDT] {taskinstance.py:1345} INFO - Marking task as SUCCESS. dag_id=market_close_pricing_v3, task_id=source_file_sensor_sla …
```

Custom Operator: SLAMonitor

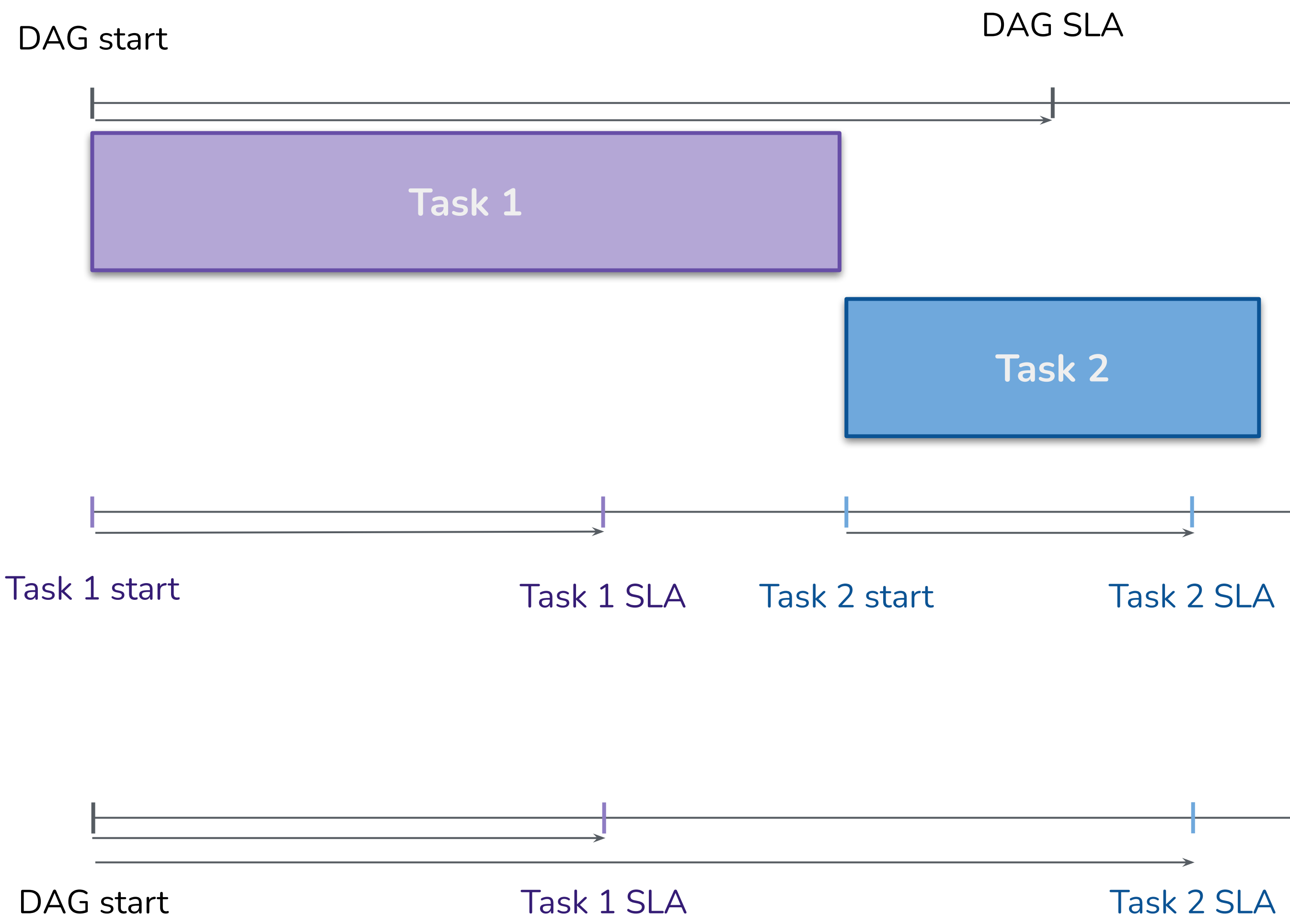
```python
from datetime import timedelta
from typing import Callable

from airflow.models.baseoperator import BaseOperator
from airflow.triggers.temporal import DateTimeTrigger
from airflow.utils.state import TaskInstanceState


class SlaMonitor(BaseOperator):
    def __init__(
        self,
        sla: timedelta,
        target_task_id: str,
        callback: Callable,
        **kwargs,
    ):
        super().__init__(**kwargs)
        self.sla = sla
        self.target_task_id = target_task_id

    def execute(self, context):
        # or define the SLA however else you see fit
        deadline = context[‘data_interval_end’] + self.sla
        self.defer(trigger=DateTimeTrigger(deadline), method_name=’execute_complete’)

    def execute_complete(self, context, event=None):
        ti = context[‘dagrun’].get_task_instance(self.target_task_id)
        if ti and ti.state == TaskInstanceState.SUCCESS:
            self.log.info("SUCCEEDED")
            return
        else:
            # replace this with your choice of sla callback function
            self.log.error("Sla was just missed!!")
            self.callback(context)
            return
```



Questions?

https://www.linkedin.com/in/sung-yun-33451688

Future of SLAs?



DAG SLA measured within a DagRun

Task SLA measured within a Task Instance

Other definitions of SLAs as Custom Operators:
Costly to manage in Airflow Core

TechAtBloomberg.com

Bloomberg
Engineering