
ALL ABOUT
DEFERRABLES
ANDREW GODWIN // @andrewgodwin

Hi, I'm

Andrew Godwin

- Principal Engineer at ASTRONOMER
- Primary author of Deferrable Operators core
- Somehow at 15 years of writing Python

Why Deferrables?

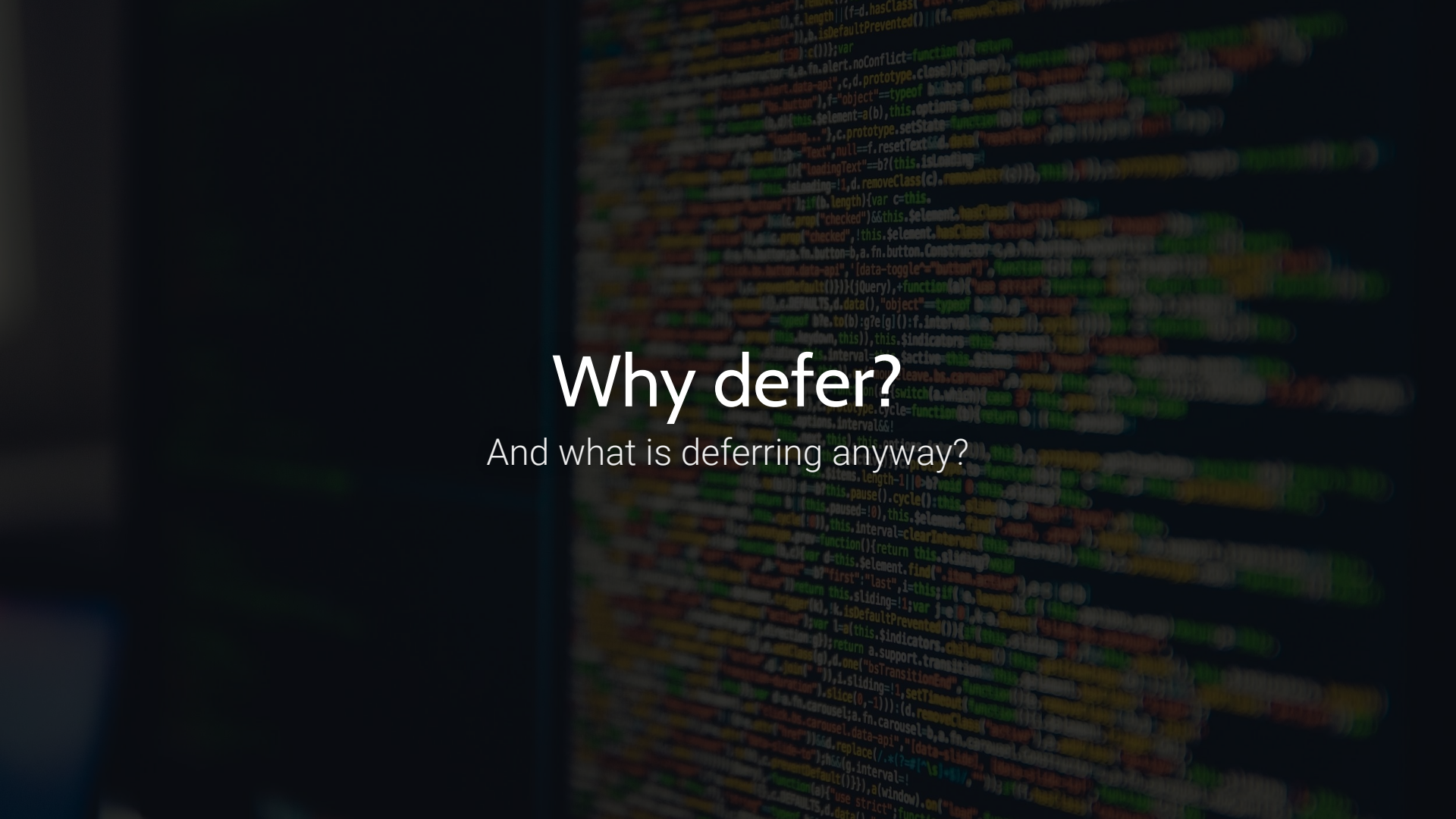
Did we really need a whole new concept?

How They Work

The answer is not just "Very Well, Thank You"

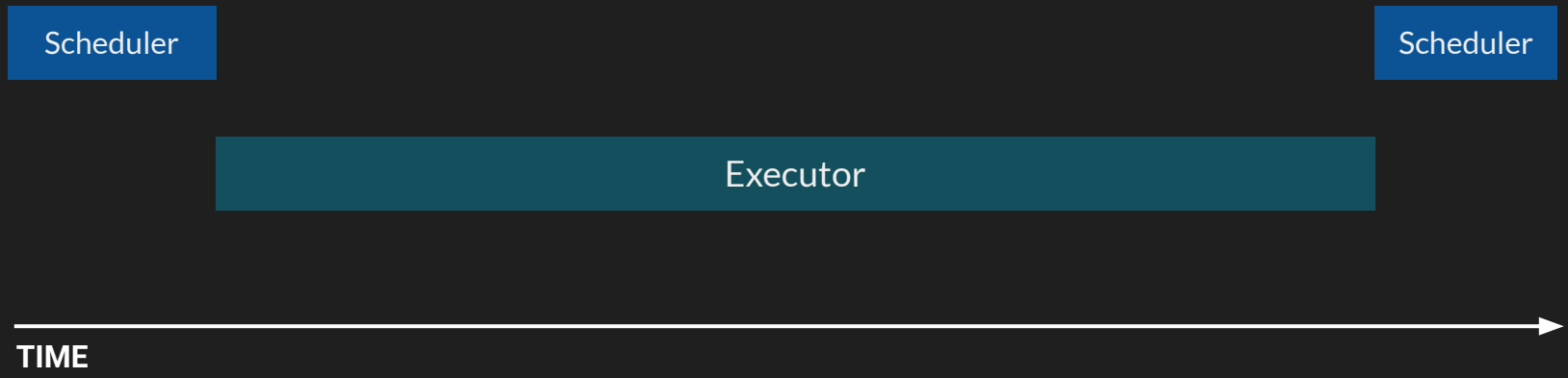
What's next?

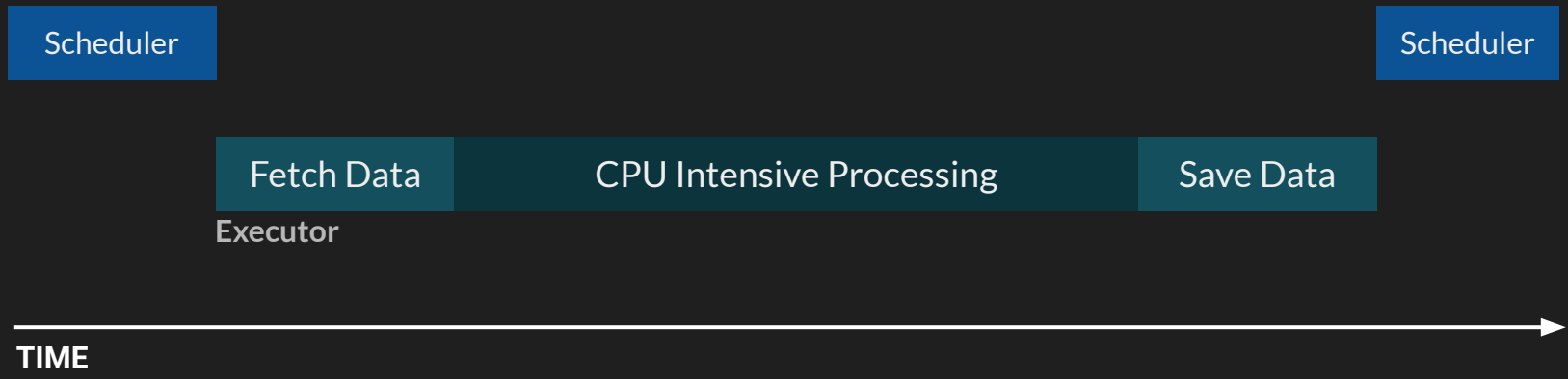
There's a lot more we can do with this

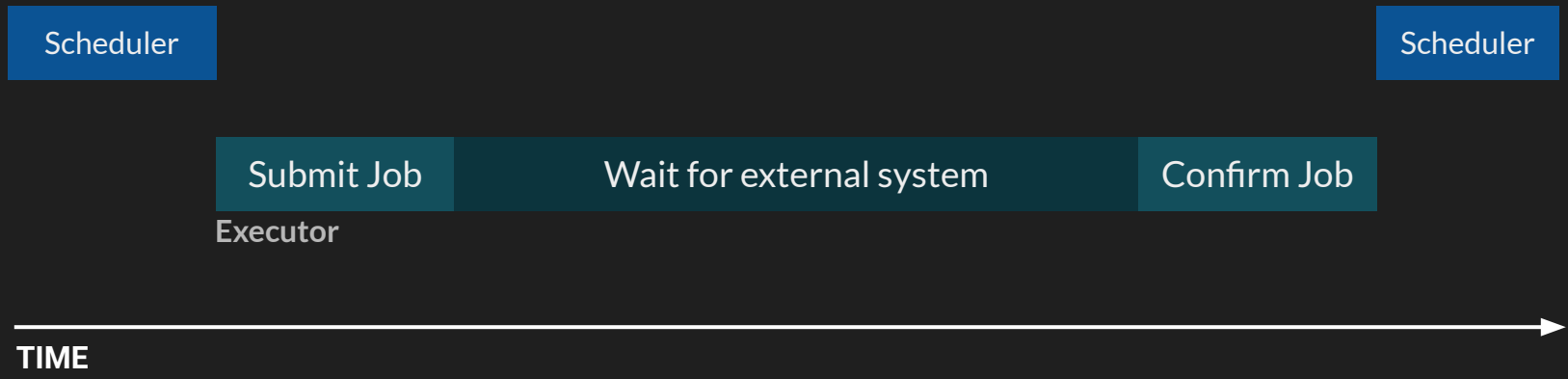


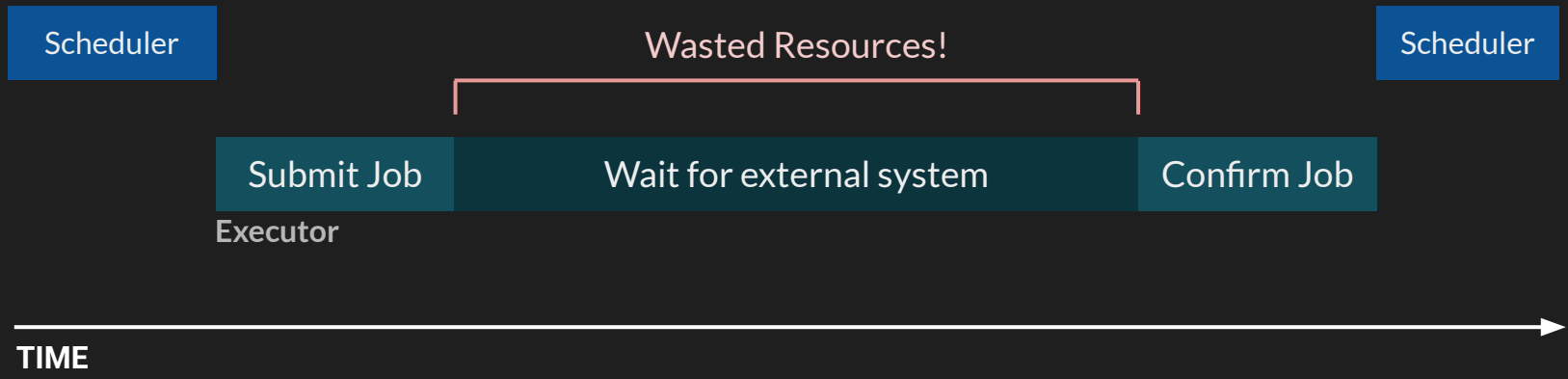
Why defer?

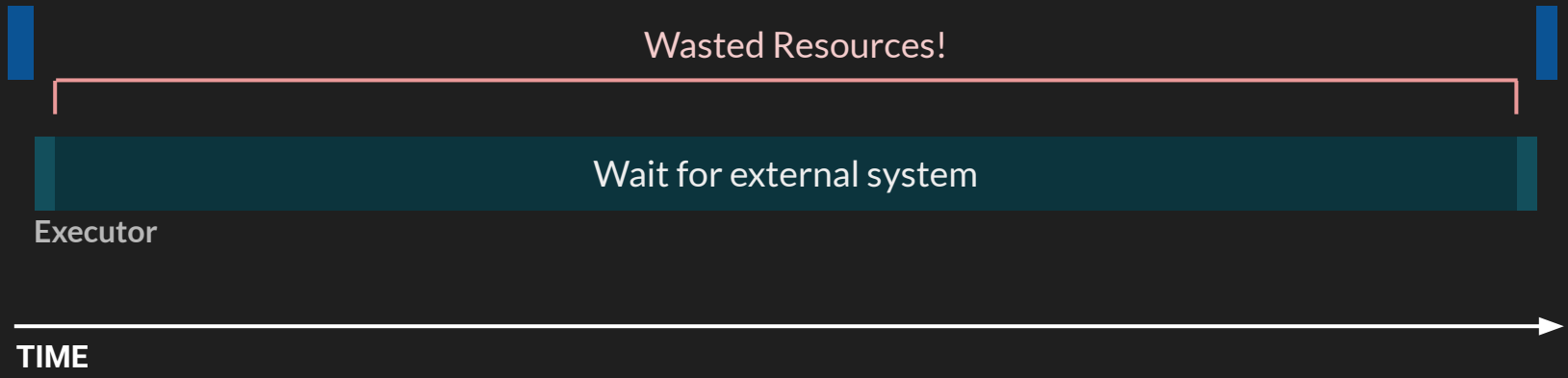
And what is deferring anyway?


Scheduler
Scheduler
Executor
TIME


Scheduler
Scheduler
Fetch Data
CPU Intensive Processing
Save Data
Executor
TIME

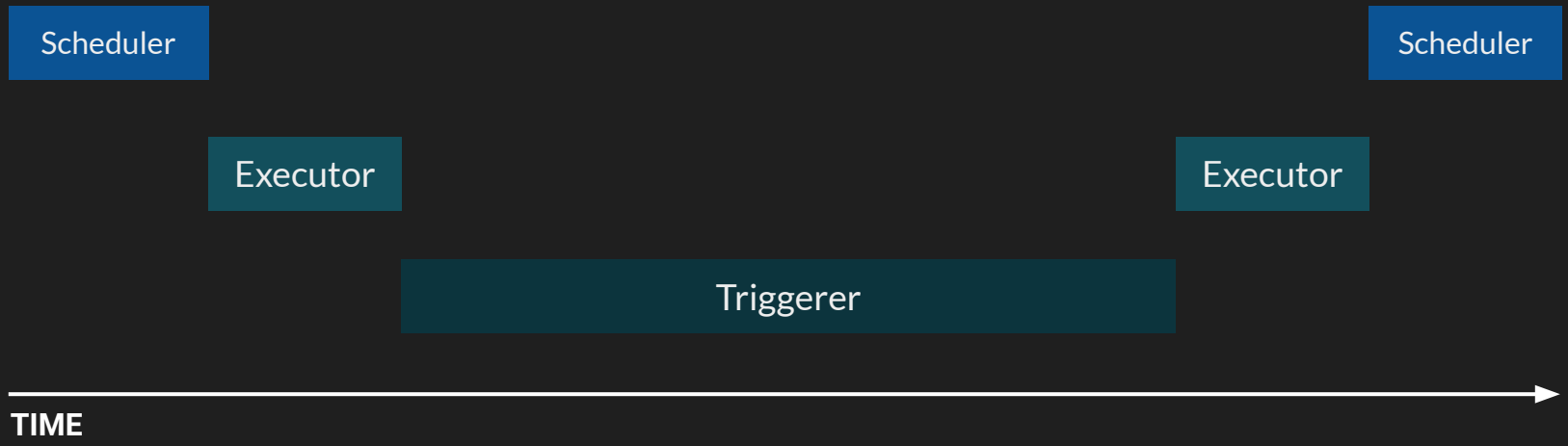

Scheduler
Scheduler
Submit Job
Wait for external system
Confirm Job
Executor
TIME

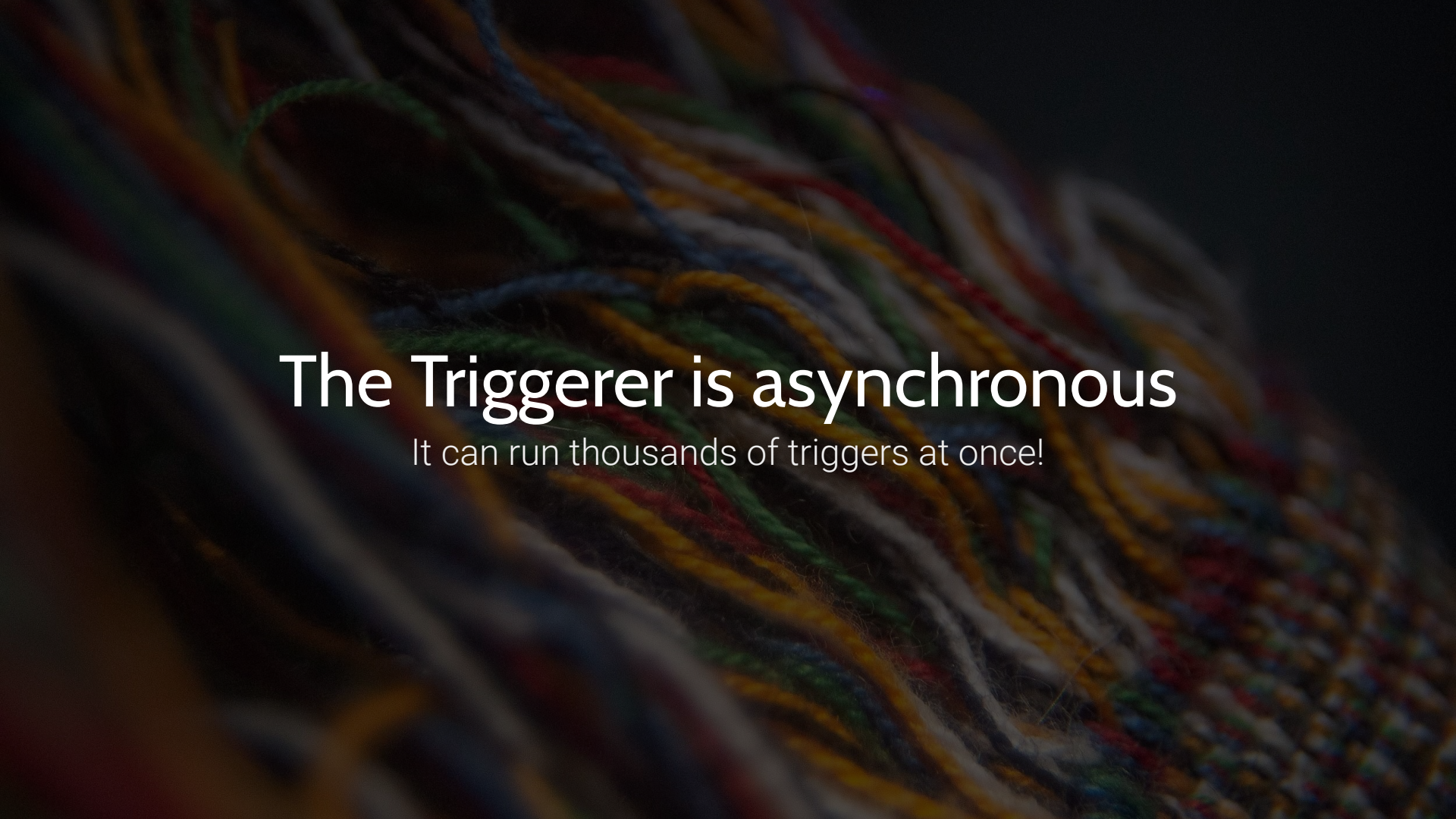

Scheduler
Wasted Resources!
Scheduler
Submit Job
Wait for external system
Confirm Job
Executor
TIME

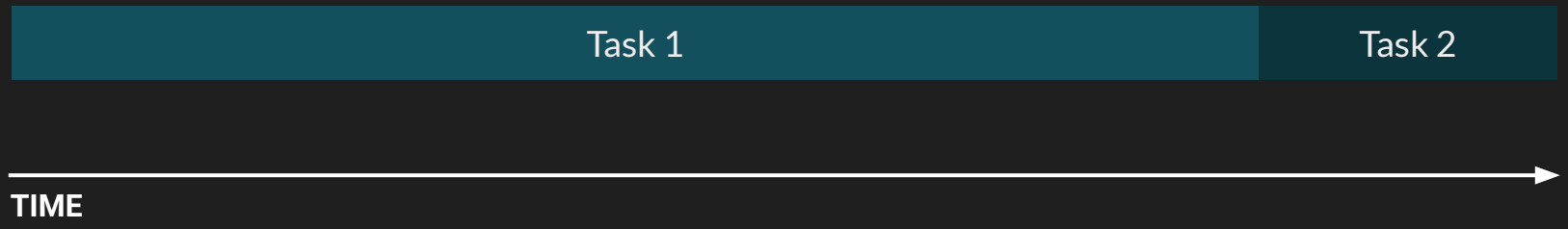

Wasted Resources!
Wait for external system
Executor
TIME

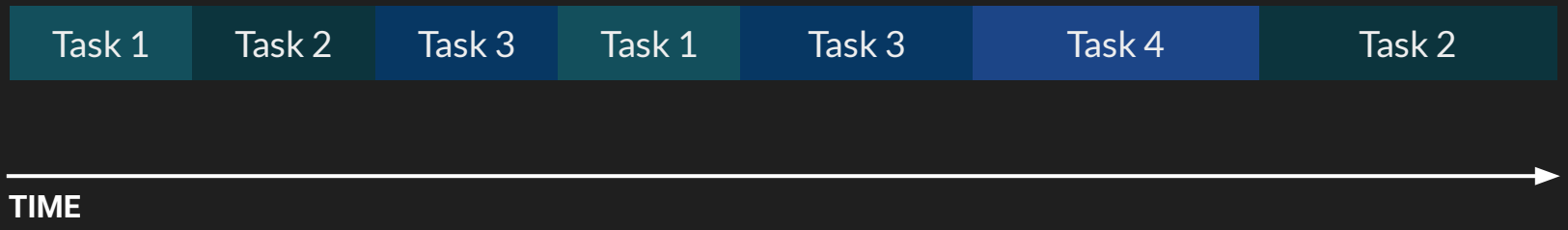
Opportunity Cost

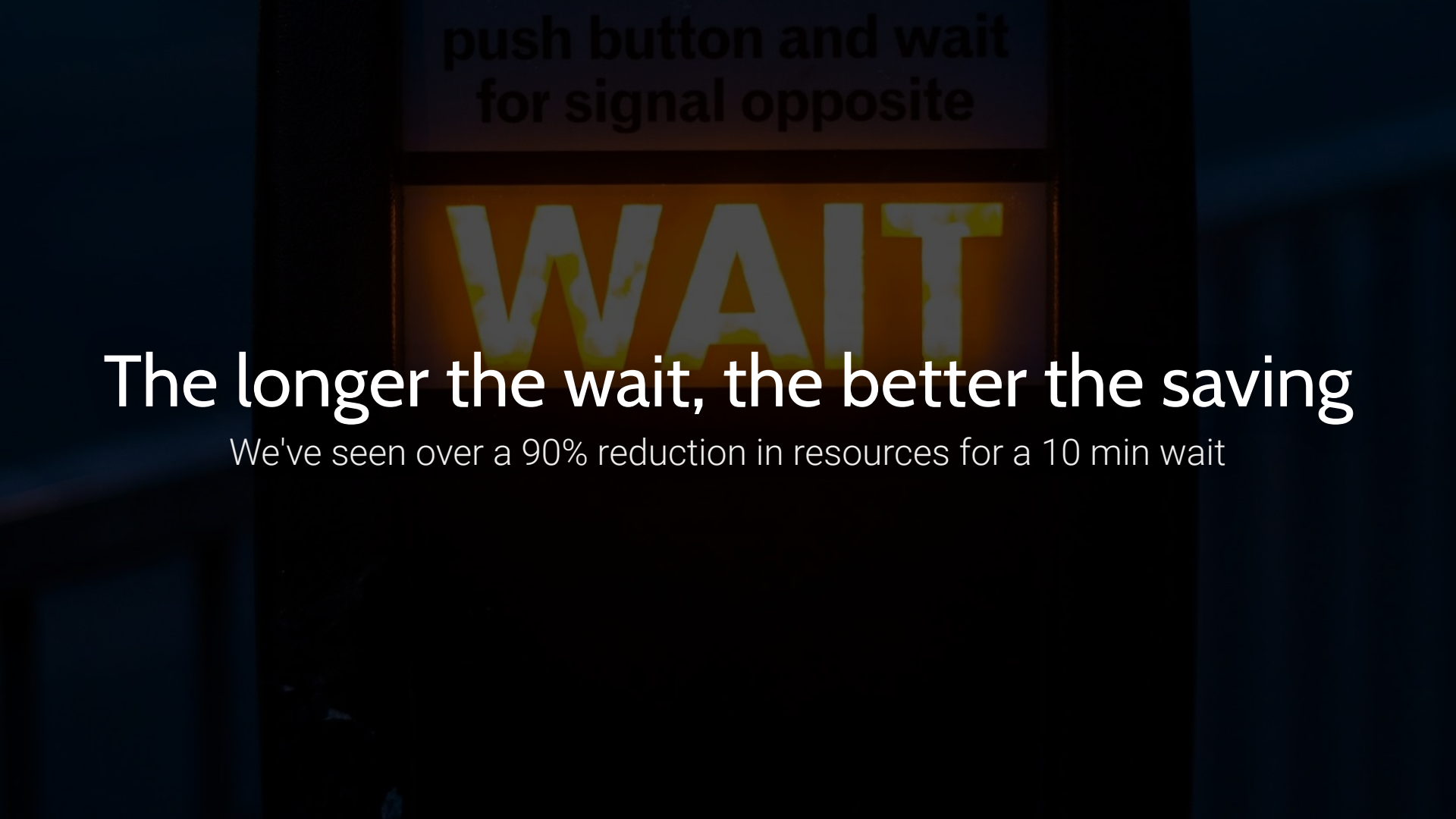
A wasted slot on Celery, wasted reservation on Kubernetes


Scheduler
Scheduler
Executor
Executor
Triggerer
TIME


The Triggerer is asynchronous
It can run thousands of triggers at once!

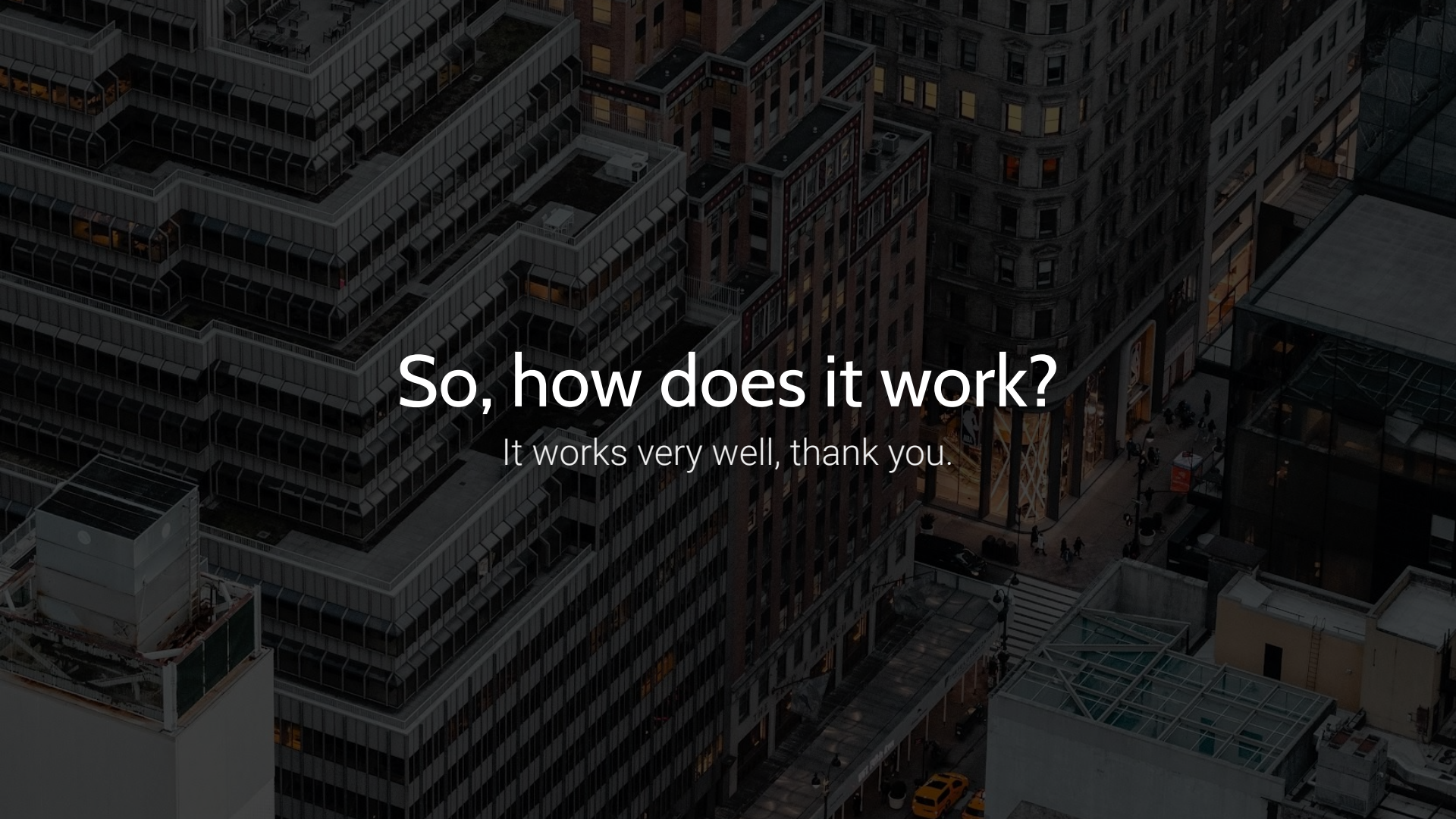

Task 1
Task 2
TIME

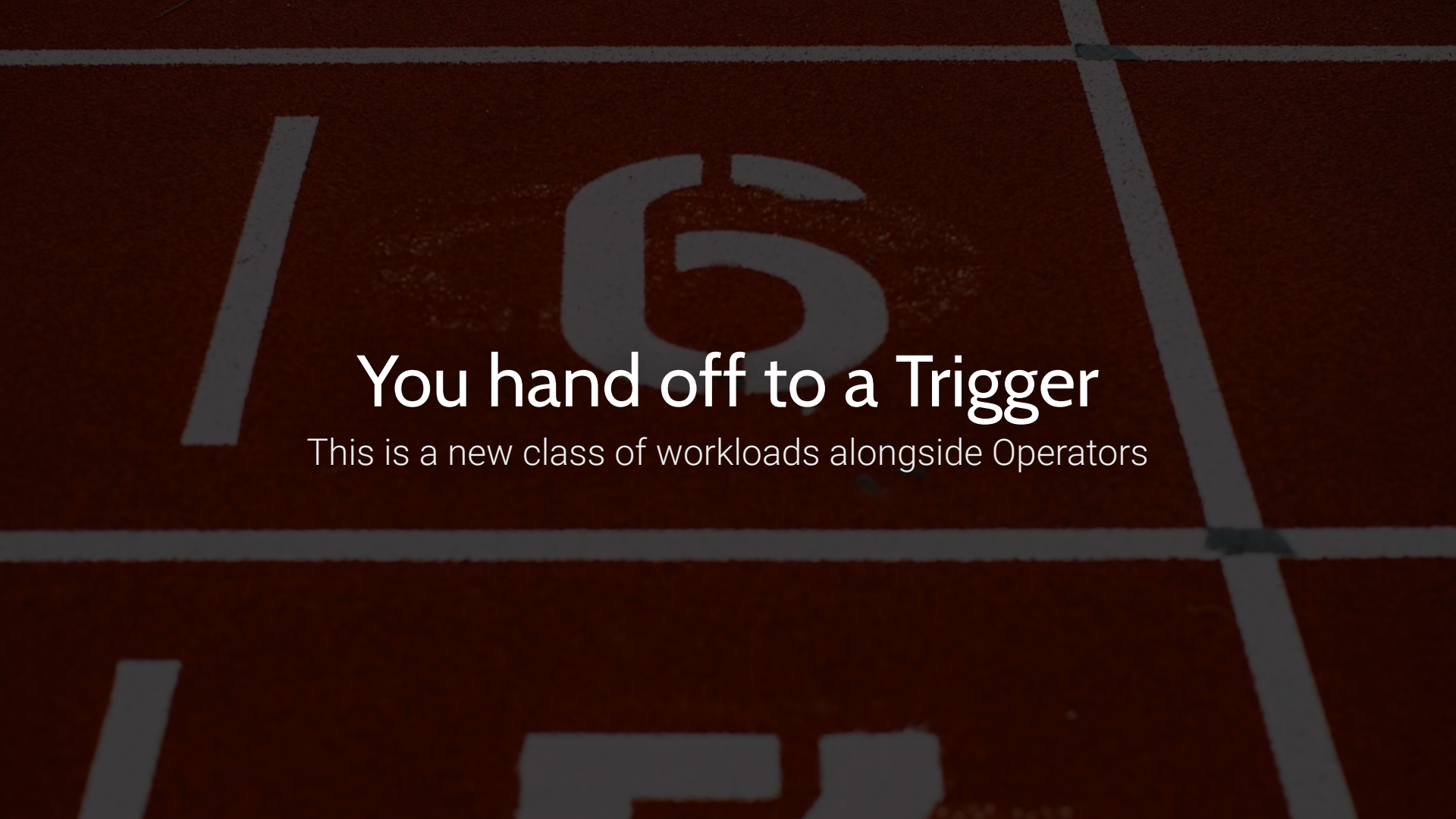

Task 1
Task 2
Task 3
Task 1
Task 3
Task 4
Task 2
TIME

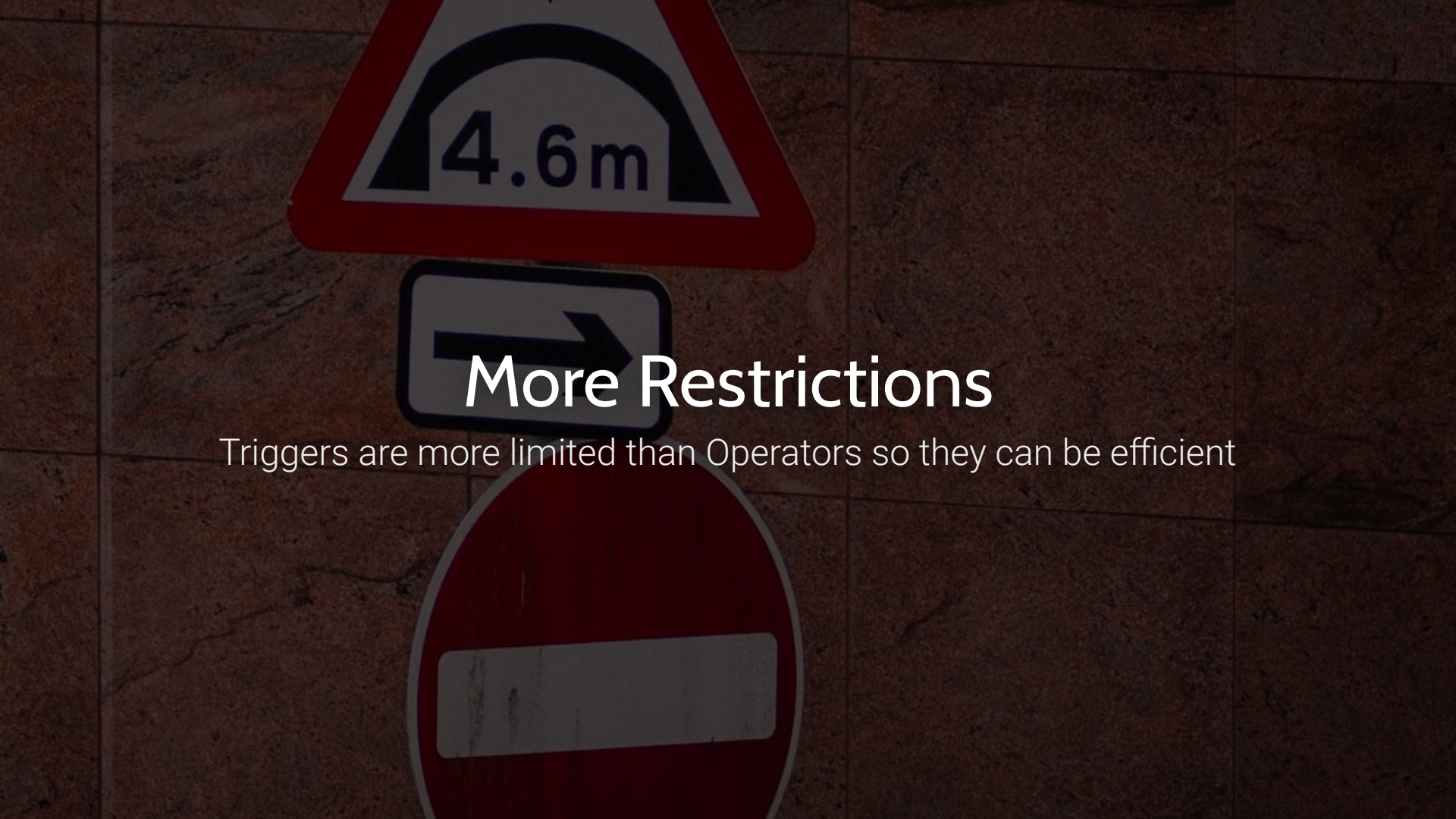


The longer the wait, the better the saving

We've seen over a 90% reduction in resources for a 10 min wait


So, how does it work?
It works very well, thank you.


You hand off to a Trigger
This is a new class of workloads alongside Operators



More Restrictions

Triggers are more limited than Operators so they can be efficient

Must be asynchronous

So we can run thousands per CPU core

No persistent state

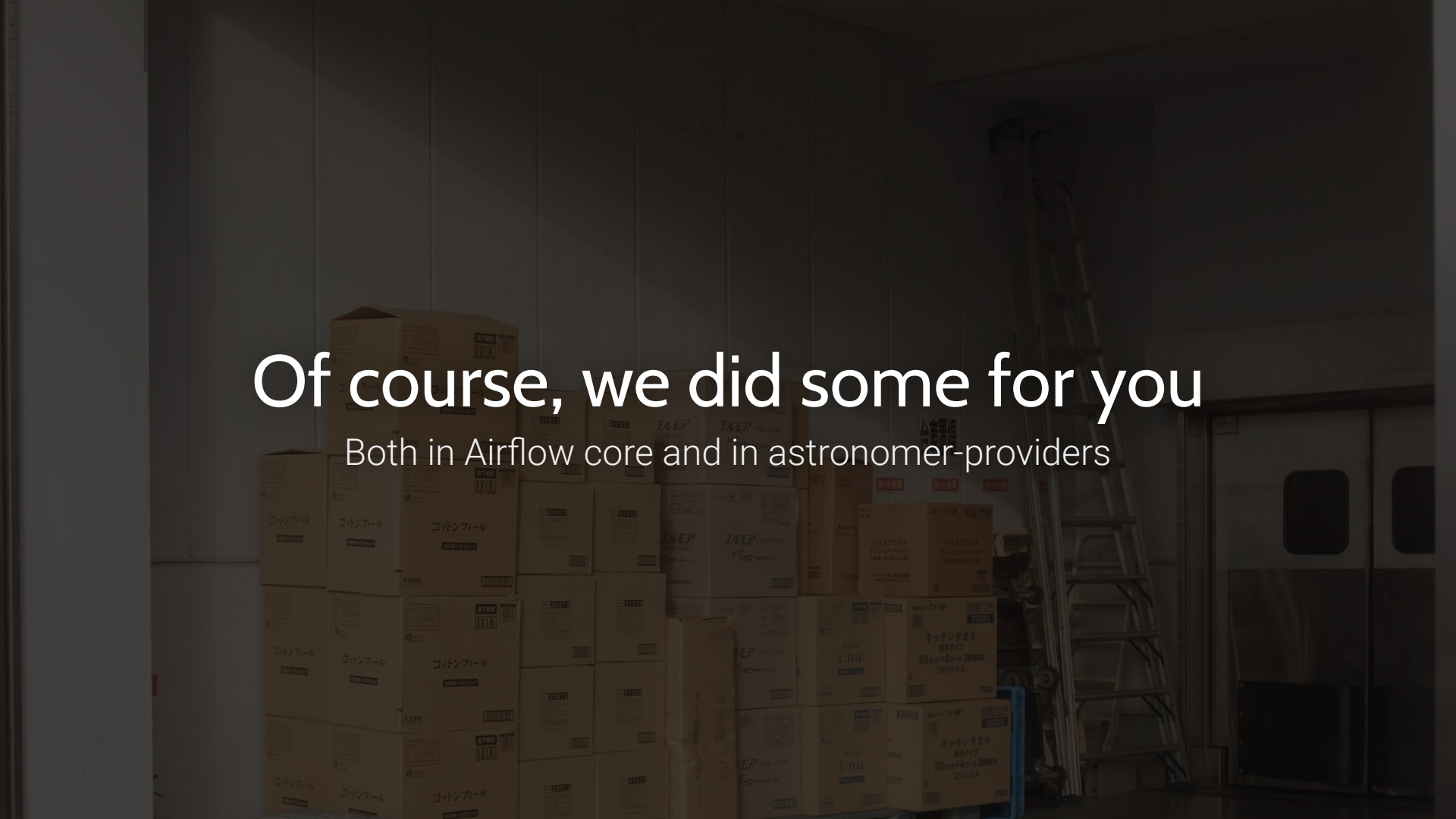
So we can shuttle them around between Triggerers as needed

Must support multiple copies

For reliability during network partitions

```python
class DateTimeTrigger(BaseTrigger):

    def __init__(self, moment: datetime.datetime):
        super().__init__()
        self.moment = moment

    def serialize(self):
        return ("mymodule.DateTimeTrigger", {"moment": self.moment})

    async def run(self):
        while self.moment > timezone.utcnow():
            await asyncio.sleep(1)
        yield TriggerEvent(self.moment)
```

```python
class WaitOneHourSensor(BaseSensorOperator):

    def execute(self, context):
        self.defer(
            trigger=TimeDeltaTrigger(timedelta(hours=1)),
            method_name="execute_complete",
        )

    def execute_complete(self, context, event=None):
        # We have no more work to do here. Mark as complete.
        return
```



Of course, we did some for you

Both in Airflow core and in astronomer-providers

Not everything can be deferred

It must be an external event/system with a portable identifier


What's next?
Turns out, Triggers are generally useful


More operator support for deferring
There's not a lot of reasons *not* to use it

Triggers for DAGs

Will likely play into the new Dataset work

Expanding async workload support

The triggerer should *really* be part of the Executor contract


Making more of Airflow async
It's not just the operators that sit there and idle a lot


Airflow is forged by people like you.
Want to help with any of this? Get in touch!

Thanks.

Andrew Godwin
@andrewgodwin
andrew.godwin@astronomer.io