Security made us do it:

Airflow's new Task Execution Architecture

Who are we?

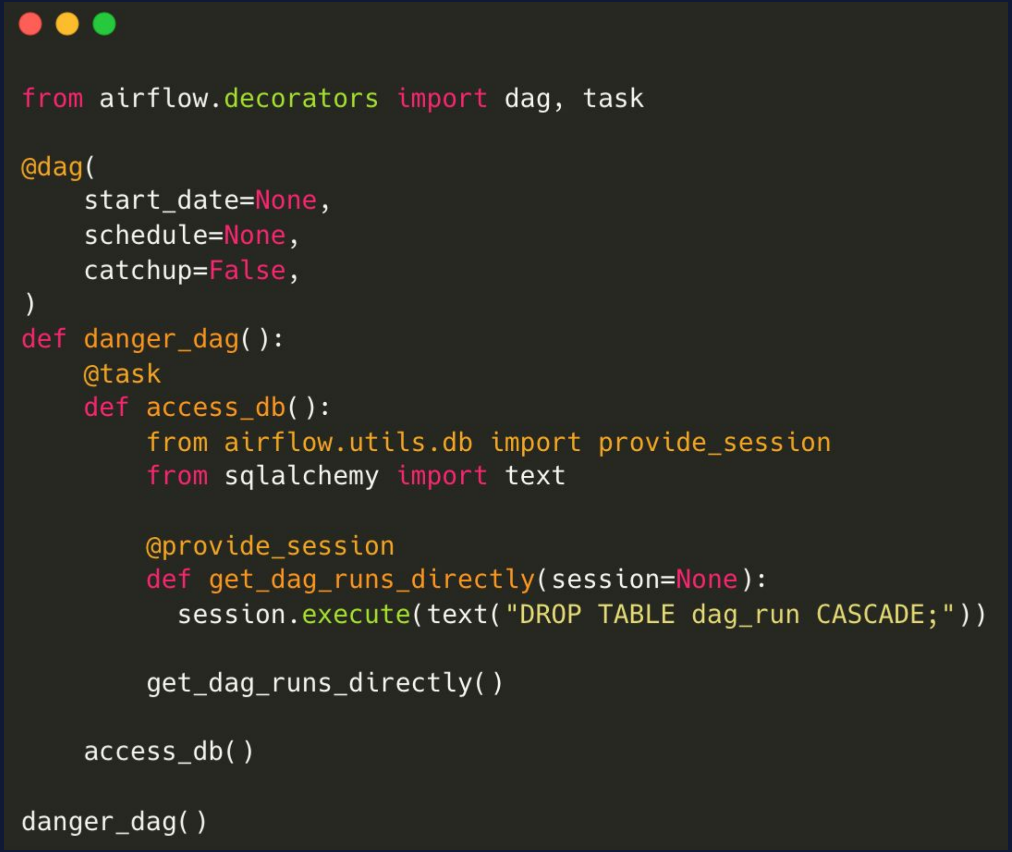
Ash Berlin-Taylor
Airflow Committer & PMC Member
Engineering Leader @ Astronomer

Amogh Desai
Airflow Committer & PMC Member
Senior Software Engineer @ Astronomer

Struggles with Airflow 2

- Tasks can talk to Database.
- Nervous while upgrading.
- Hard to Scale.
- Tasks have to be on same network as DB

What happens if I run this DAG in Airflow 2?



```
from airflow.decorators import dag, task

@dag(
    start_date=None,
    schedule=None,
    catchup=False,
)
def danger_dag():
    @task
    def access_db():
        from airflow.utils.db import provide_session
        from sqlalchemy import text

        @provide_session
        def get_dag_runs_directly(session=None):
          session.execute(text("DROP TABLE dag_run CASCADE;"))

        get_dag_runs_directly()

    access_db()

danger_dag()
```

Disclaimer: Astronomer does not accept any responsibility if you try this at home! Please don't.

Upgrade Challenges (2.x –> 2.y)

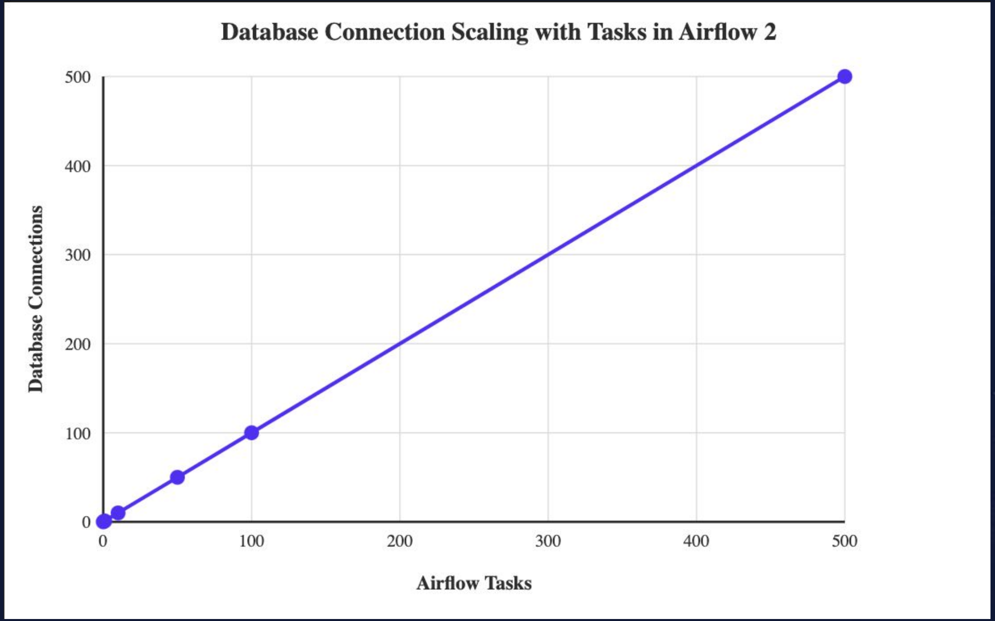
Will this upgrade break my jobs?

- DAGs authored using Airflow's codebase (tight coupling)
- Workers use Airflow's shared codebase to run tasks
- Worry that upgrade will force provider upgrade too

Change to codebase -

- DAG authors impacted! Update *most* DAGs now
- Workers and scheduler must be on identical version

Scaling Challenges in Airflow 2

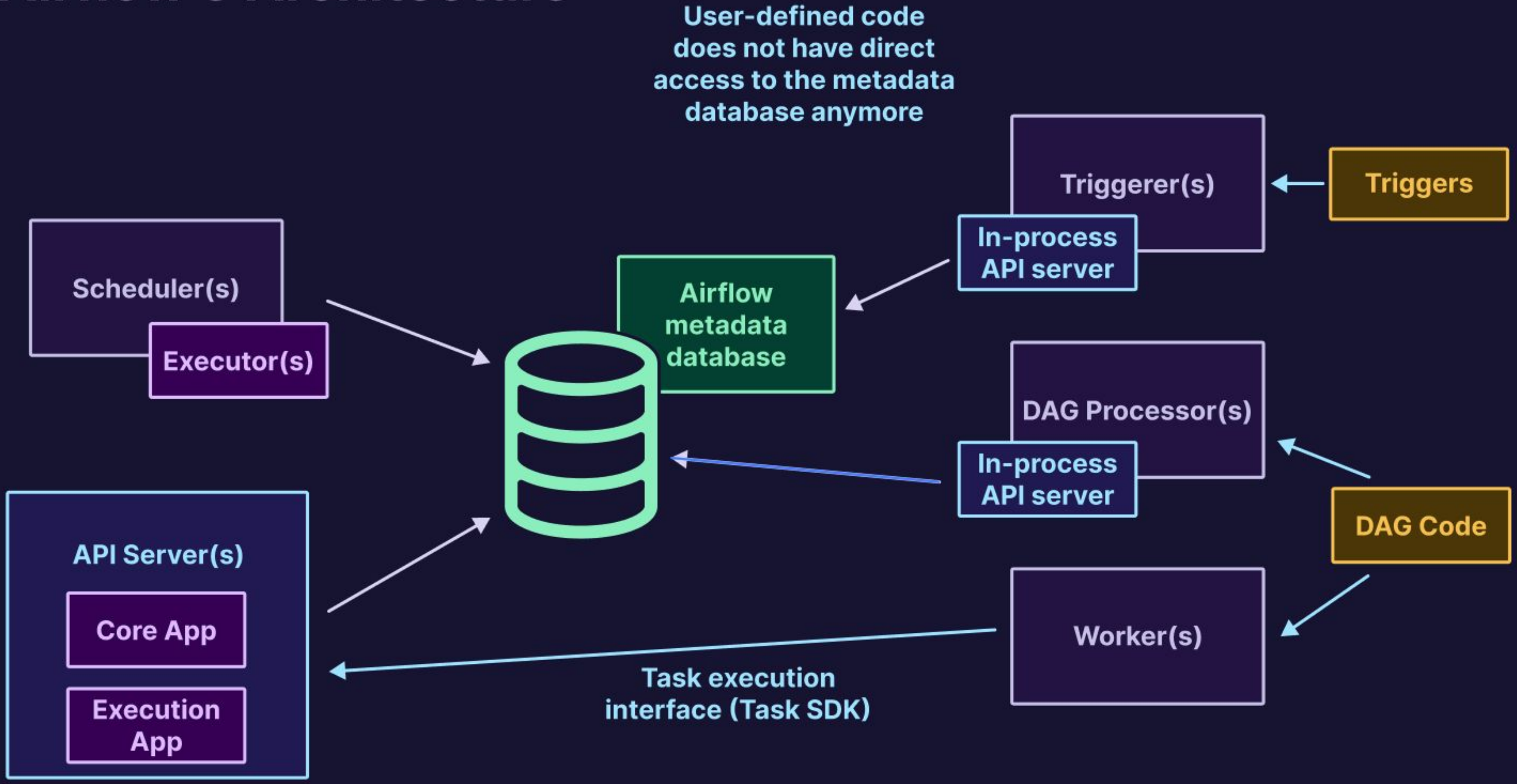


- Each task *can* create DB connections
- *n* tasks ⇒ at least *n* DB connections!
- DB becomes a bottleneck to scaling

The brave new world of

Airflow 3

Airflow 3 Architecture



New Terms

API Server: Airflow Server Component that provides the sole database access point for all Airflow operations for workers.

Task Execution Interface: The REST API that allows workers to communicate with Airflow.

Task SDK: The lightweight package installed on workers that enables them to talk to Airflow.

Goals

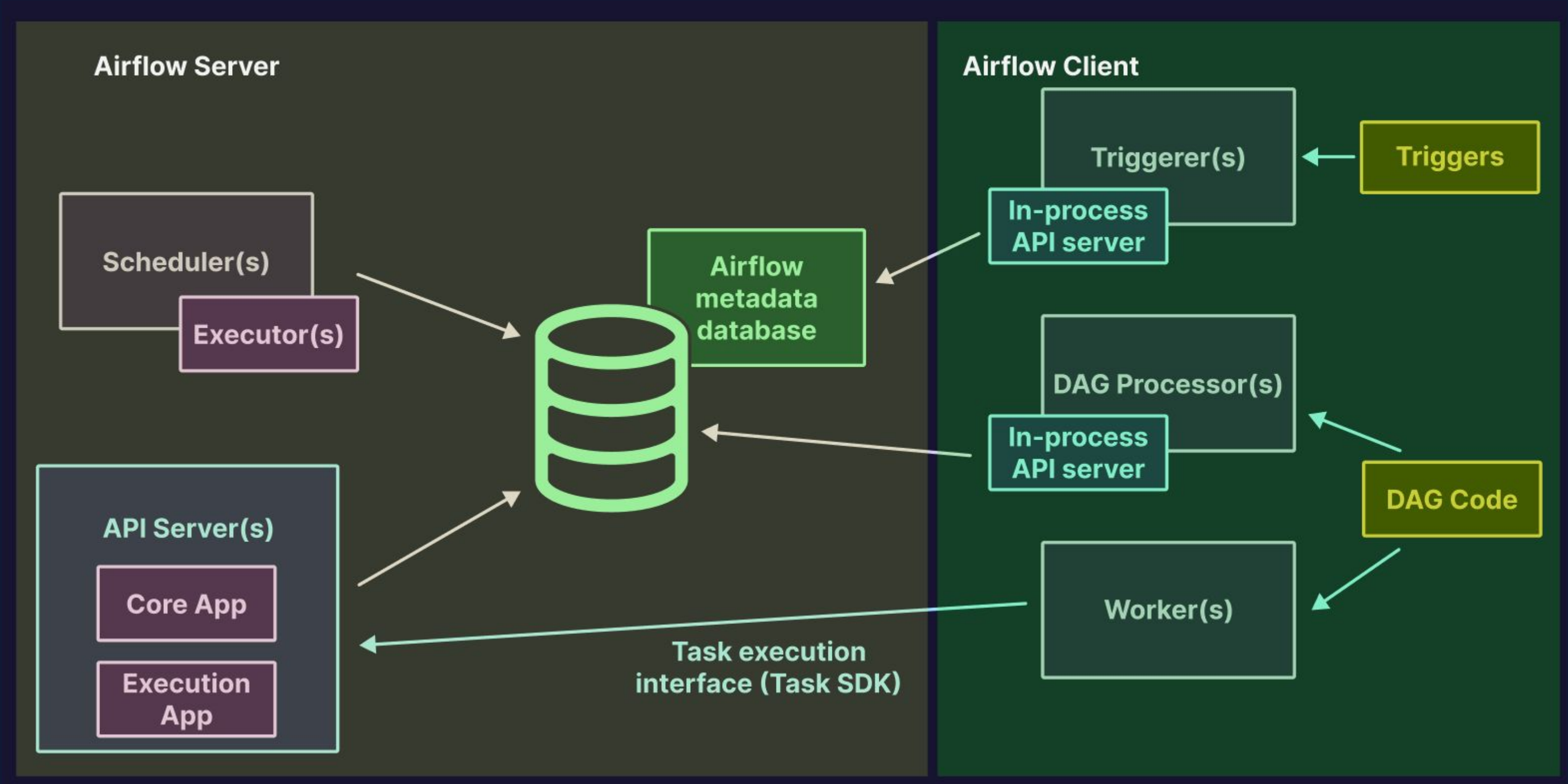
1. Forbid tasks from accessing the metadata DB
2. Workers continue to execute tasks without code change when Deployment is upgraded
3. Enable tasks in multiple languages

Goal #1:
Tasks without direct DB access

Access everything via an API

ASTRONOMER

Server/Client Split



ASTRONOMER

Task SDK

Lightweight package – the thing providers and Dag authors need to use.

Well defined public python interface for Dag and Provider authors

```python
from airflow import DAG
from airflow.decorators import task_group
from airflow.models import Connection, Variable
```

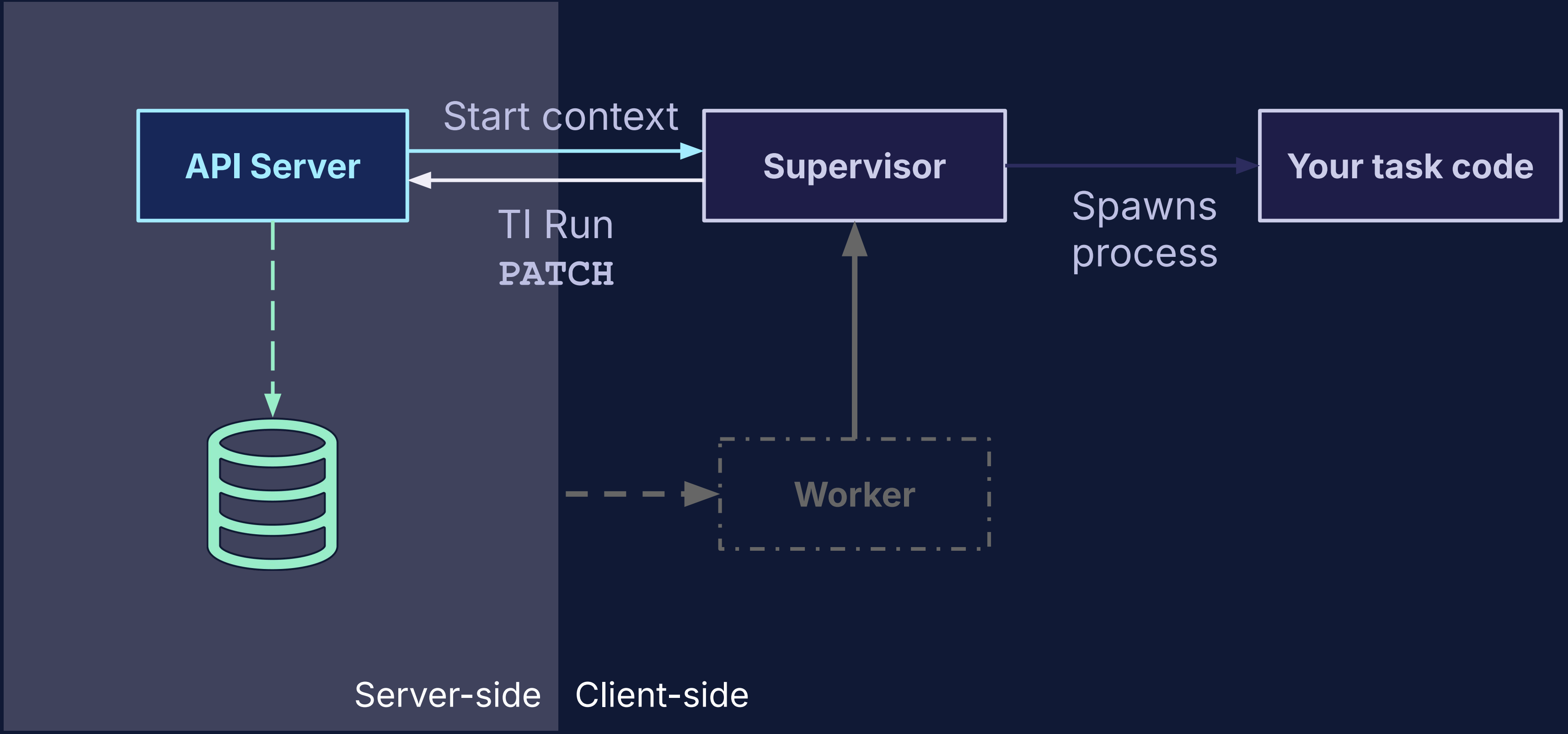
should now be

```python
from airflow.sdk import Connection, DAG, Variable, task_group
```

Compat shims exist, don't worry!

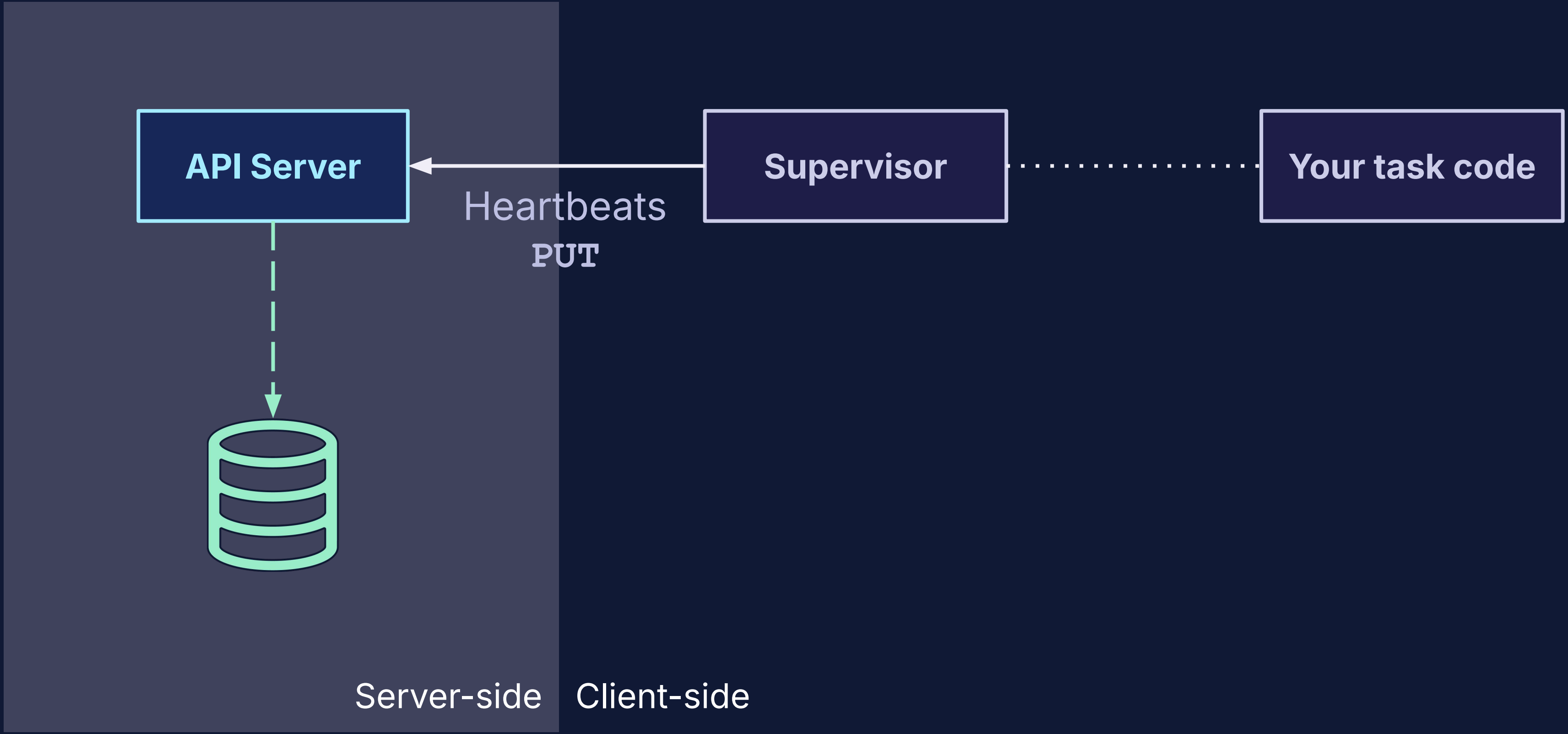
ASTRONOMER

Traffic Flows: Starting a task

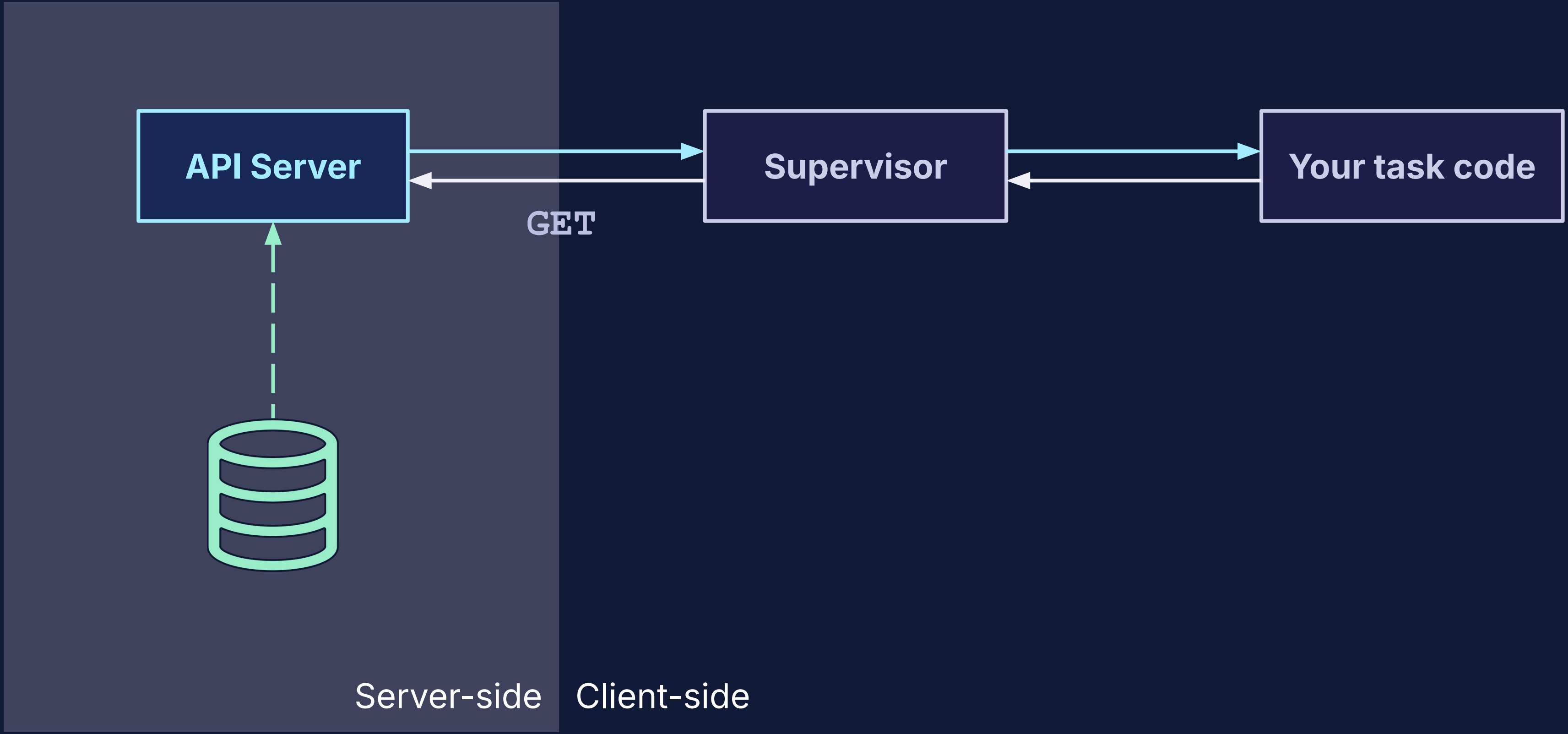


ASTRONOMER

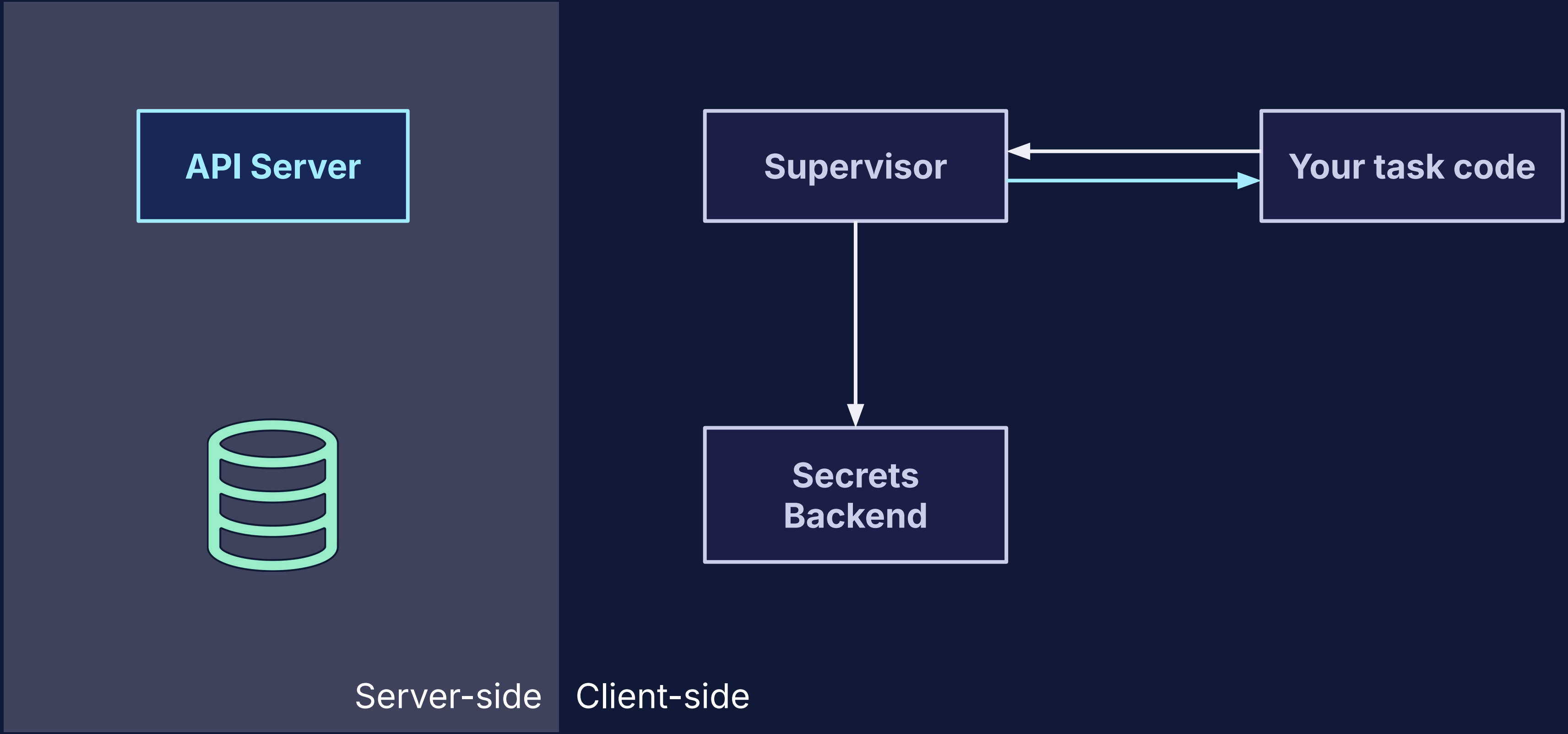
Traffic Flows: TI Heartbeat



Traffic Flows: Variables/XCom/Connection

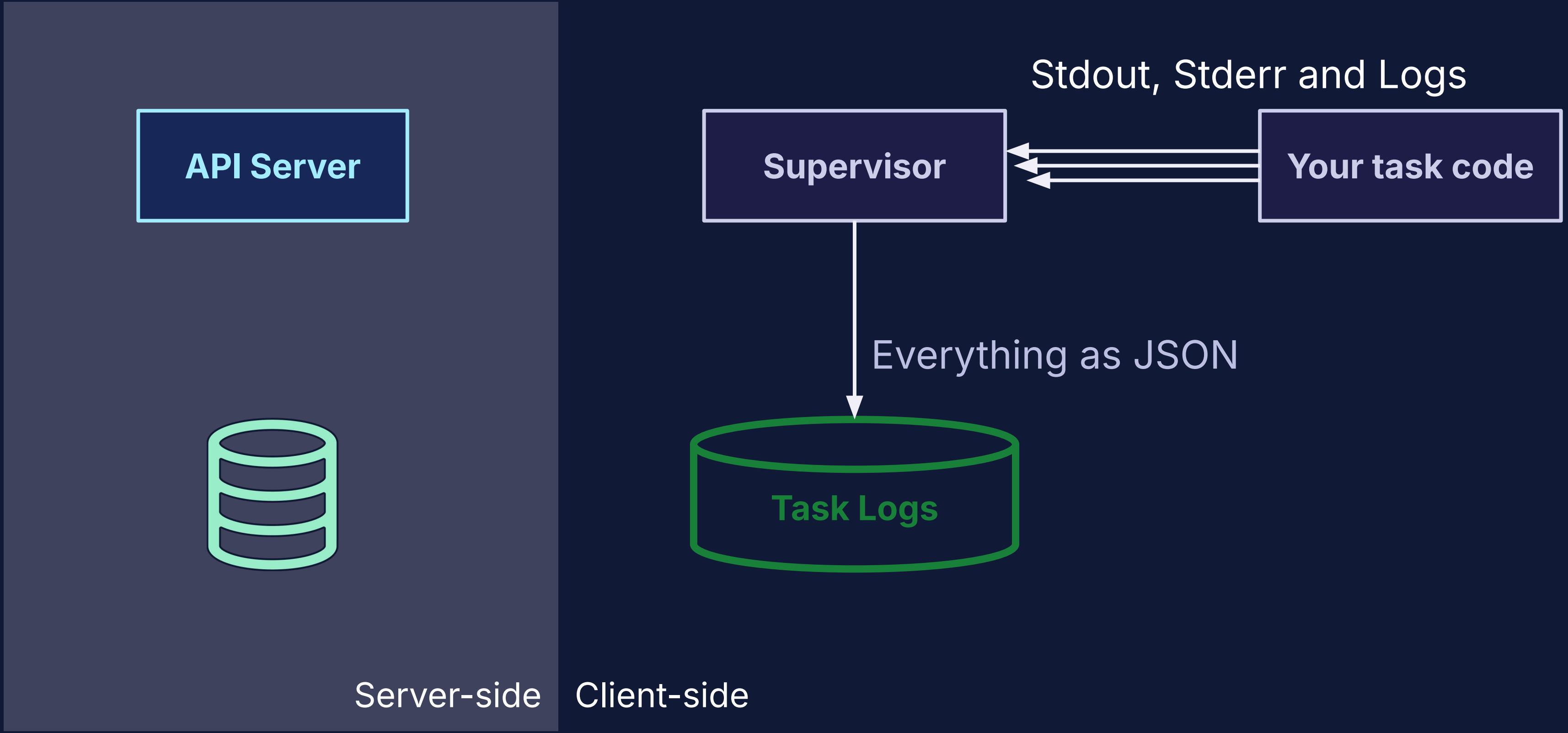


Traffic Flows: Conn w/ Secrets Backend

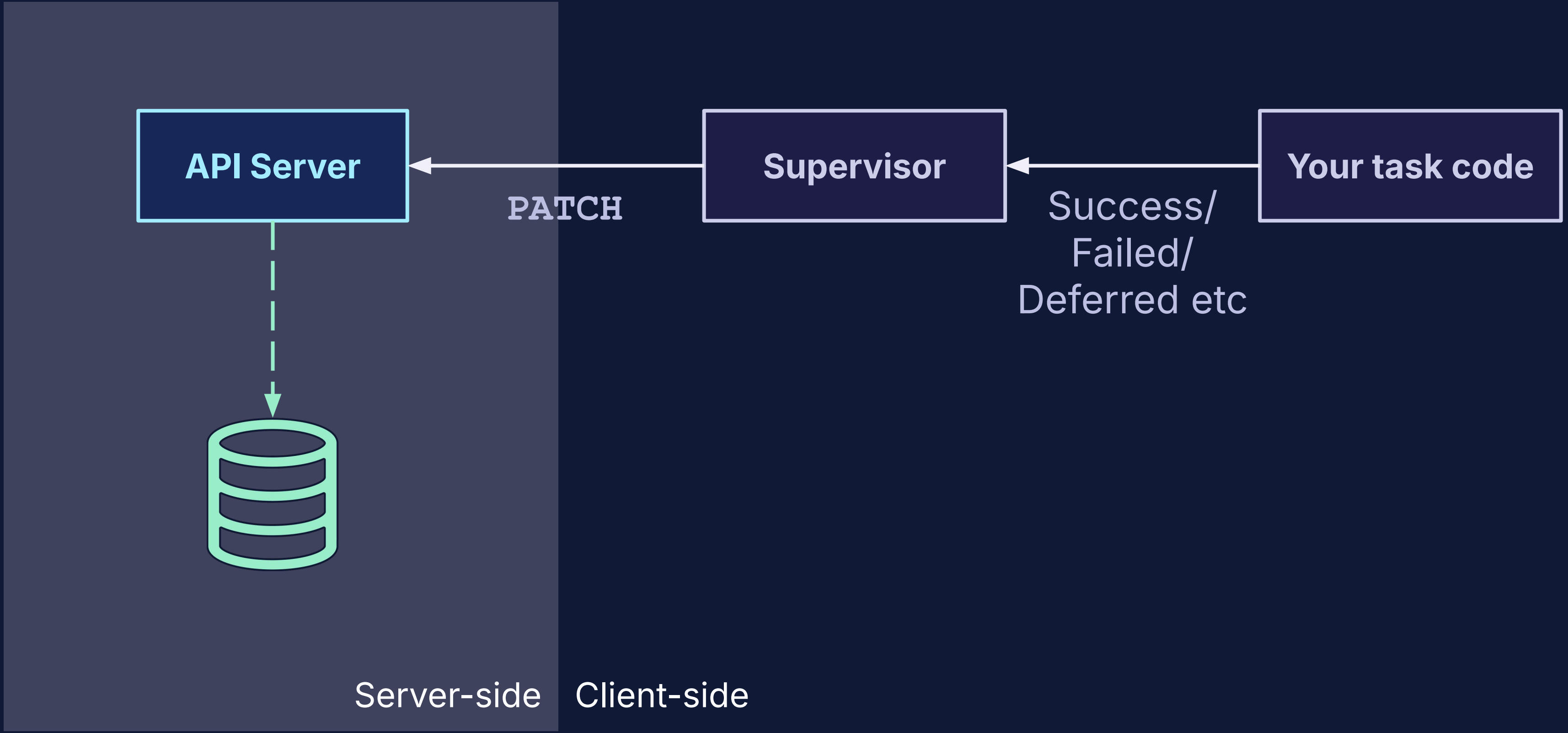


ASTRONOMER

Traffic Flows: Output and Logs



Traffic Flows: TI Status update



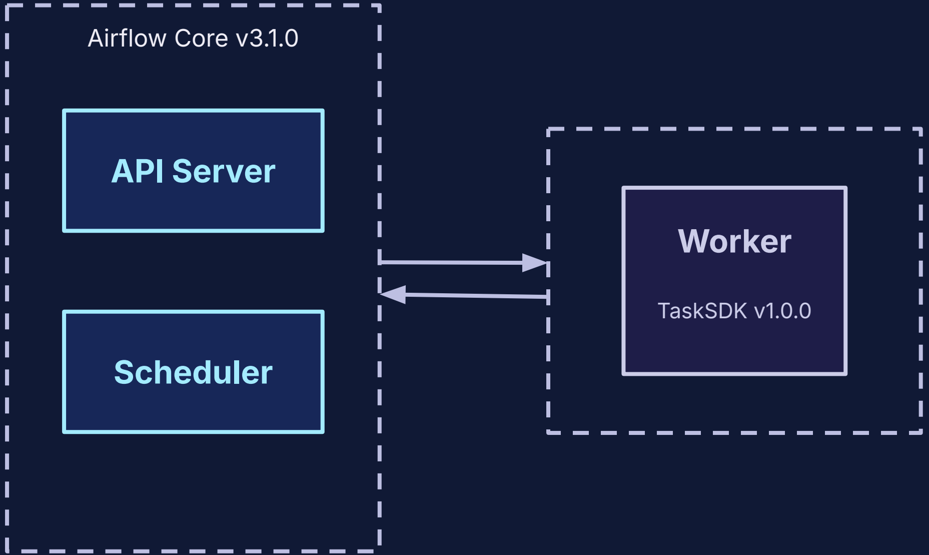
ASTRONOMER

Goal #2:
Support older Workers

With automatic migration of HTTP API request and responses

ASTRONOMER

Why?



OLD WORKER (Task SDK v1.0.0):

"Give me the version_id and task_id for this task"

Expecting: `{"task_id": "123", "version_id": "abc", "max_tries": 3}`

NEW API SERVER (v3.1.0):

"Here you go"

Response: `{"task_id": "123", "dag_version_id": "abc", "max_tries": 3}`

ASTRONOMER

Common Approaches

1. Multiple Deployments (very expensive)

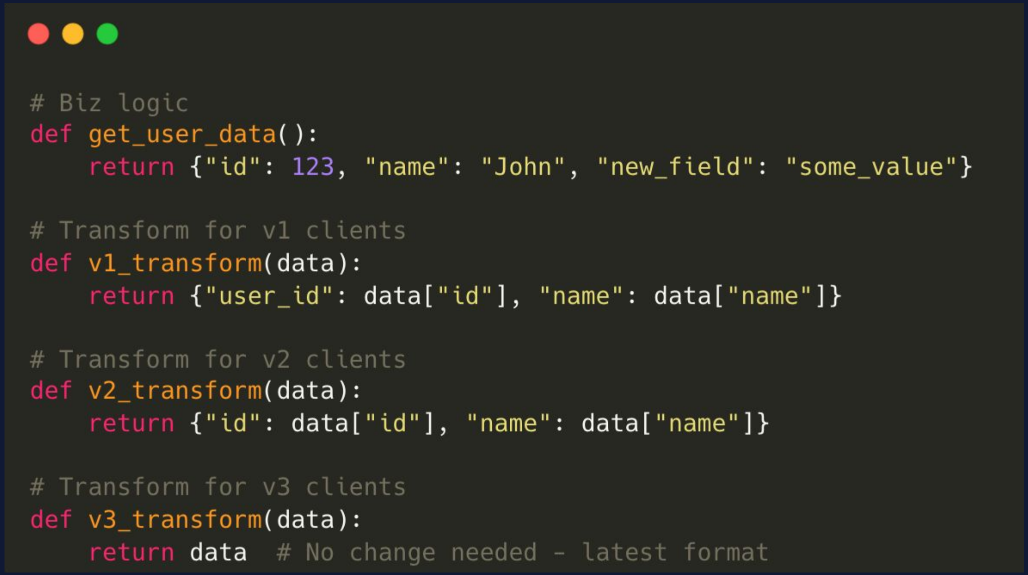
2. Duplicating Endpoints (not scalable)

```
GET /v1/orders  → Old logic
GET /v2/orders  → New logic
GET /v3/orders  → Newest logic
```

ASTRONOMER

Common Approaches

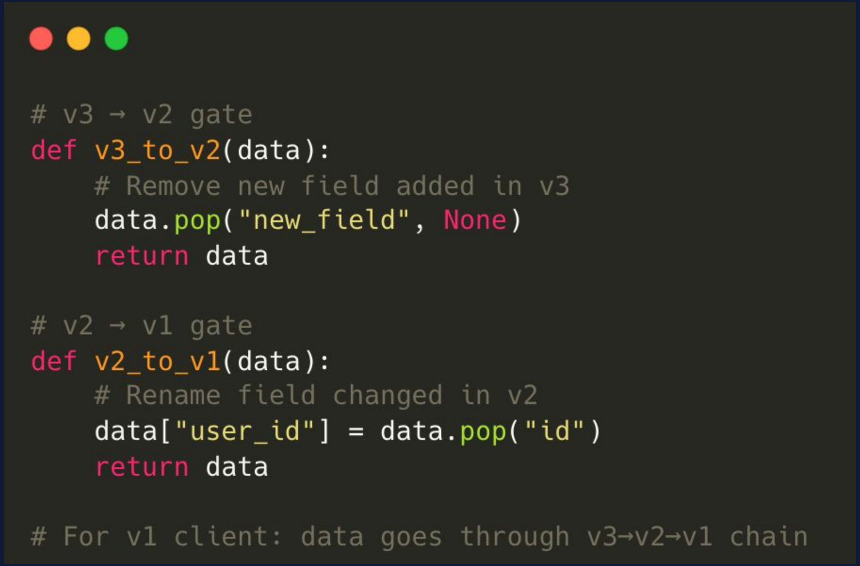
3. Schema-Only Migrations (complex transformation)



```python
# Biz logic
def get_user_data():
    return {"id": 123, "name": "John", "new_field": "some_value"}

# Transform for v1 clients
def v1_transform(data):
    return {"user_id": data["id"], "name": data["name"]}

# Transform for v2 clients
def v2_transform(data):
    return {"id": data["id"], "name": data["name"]}

# Transform for v3 clients
def v3_transform(data):
    return data  # No change needed - latest format
```

Common Approaches

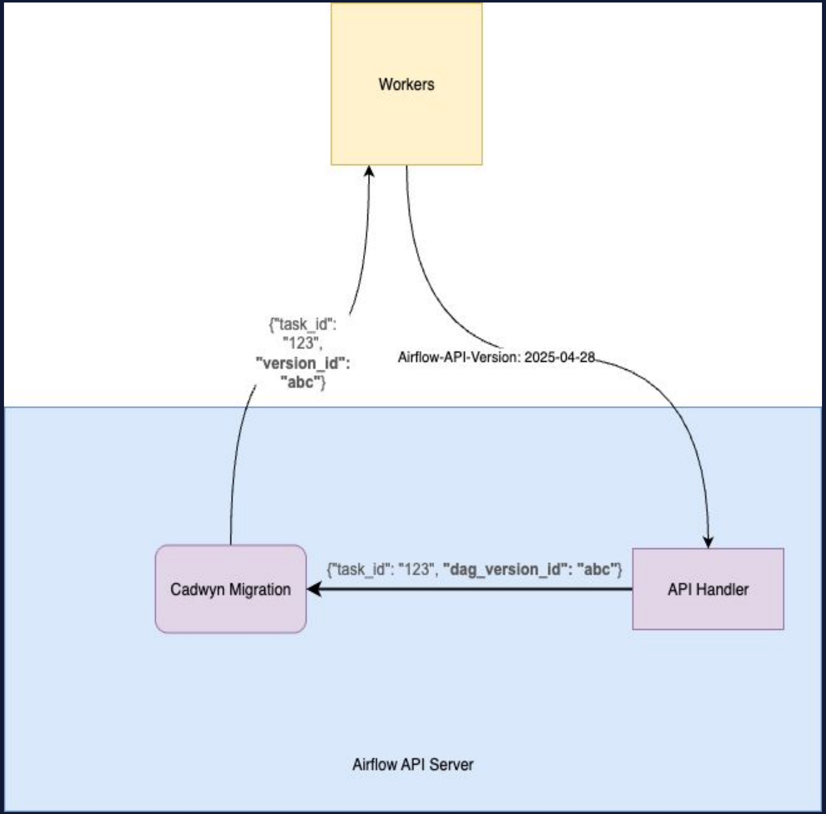
4. Stripe’s Method
   Transformation by exit “gates”



```python
# v3 → v2 gate
def v3_to_v2(data):
    # Remove new field added in v3
    data.pop("new_field", None)
    return data

# v2 → v1 gate
def v2_to_v1(data):
    # Rename field changed in v2
    data["user_id"] = data.pop("id")
    return data

# For v1 client: data goes through v3→v2→v1 chain
```

Scalable, Maintainable

Cadwyn Migrations


Workers
{"task_id": "123", "version_id": "abc"}
Airflow-API-Version: 2025-04-28
{"task_id": "123", "dag_version_id": "abc"}
Cadwyn Migration
API Handler
Airflow API Server

- Cadwyn: Time Travel
- "Undo" all the changes that happened between those dates

Goal:
Support all task SDK versions back to *X* version

Conditions for API Migrations

- Server **must** be ⩾ clients
  - ✅ NEW SERVER → OLD CLIENTS
  - ❌ OLD SERVER → NEW CLIENTS

- Remote Execution:
  - Server Upgrades first
  - Consumers upgrade when ready
  - Cadwyn handles the version mismatch

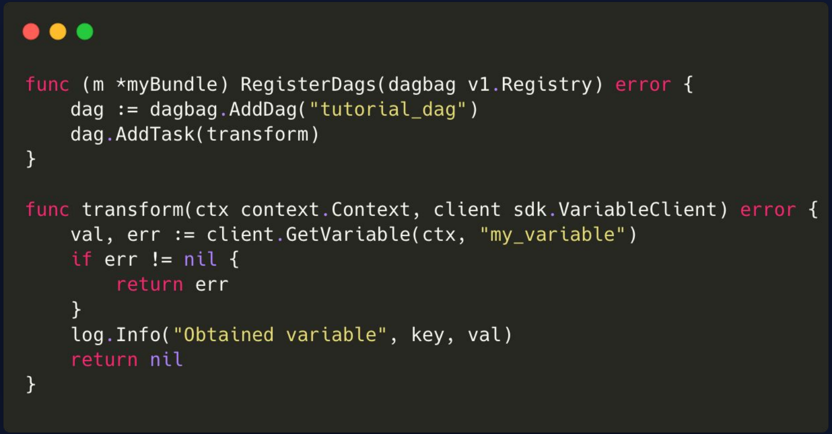
ASTRONOMER

Goal #3:
Tasks in any language

Golang, Java, `$your_choice`…

ASTRONOMER

Benefits: Golang SDK!



```go
func (m *myBundle) RegisterDags(dagbag v1.Registry) error {
    dag := dagbag.AddDag("tutorial_dag")
    dag.AddTask(transform)
}

func transform(ctx context.Context, client sdk.VariableClient) error {
    val, err := client.GetVariable(ctx, "my_variable")
    if err != nil {
        return err
    }
    log.Info("Obtained variable", key, val)
    return nil
}
```

Feels *native* to Go

DAG bundles are compiled into binaries, worker loads them via `hashicorp/go-plugin`

ASTRONOMER

Writing a new language SDK

1. Work out how ExecuteTaskWorkload will get to your workers
   (i.e. implement an Edge executor client)
2. Build a client for the Task Execution API OpenAPI 3.1 spec
3. Workout how you will load your Task functions
   (Dynamically importing? Load plugins? Precompiled in to worker?)
4. Stick it all in a `while True`
5. ???
6. Profit

Tasks as RPC?

A new way of thinking about Airflow Tasks

- Tasks in their own deployment (i.e. an Airflow worker).
  Benefits from code share with main app

- Tasks as RPC: Tasks can run in an existing app deployment!
  i.e. run the Airflow Task handler inside your go webserver process

One Orchestrator, Any Language, Anywhere 🎯

ASTRONOMER