Converting Legacy Schedulers

& Announcing Astronomer Orbiter

Fritz Davenport

Fritz @astronomer.io

Principal Data Engineer

ASTRONOMER

Migrations

Difficult

Autosys

Automic

&

Cronicle

Control-M

Jenkins

&

CRON

JAMS

Necessary...

&

SSIS

AWS Step Functions

Oozie

&

Migrations are...

Difficult Risky Boring Complex Slow **Necessary...**

Various Tools Exist

Control-M to Airflow

https://github.com/GoogleCloudPlatform/dagify

Oozie to Airflow

https://github.com/GoogleCloudPlatform/oozie-to-airflow

https://www.astronomer.io/white-papers/migrating-from-oozie...

Pentaho to Airflow

https://medium.com/@swapnilspra/etl-code-migration-from-pentaho...

& more

Migration Process Overview

1. Setup — networking, infrastructure, authentication, etc
2. Identify, Group — common patterns, stakeholders, related workloads
3. Migrate — migrate migrate migrate migrate...

   a. Translate — Origin System → 🤔 → Airflow
   b. Test, Deploy — 🤞

4. Adopt, Refactor

/input_folder

Structured Workflows

```xml
<coordinator-app name="hello-coord" frequency="${coord:days(1)}"
                 start="2009-01-02T08:00Z" end="2009-01-04T08:00Z"
                 xmlns="uri:oozie:coordinator:0.1">
    <controls>
        <timeout>10</timeout>
        <concurrency>${concurrency_level}</concurrency>
        <execution>${execution_order}</execution>
        <throttle>${materialization_throttle}</throttle>
    </controls>
```

```xml
<workflow-app xmlns="uri:oozie:workflow:1.0" name="demo-wf">
    <start to="cleanup-node"/>

    <action name="cleanup-node">
        <fs>
            <delete path="${nameNode}/user/${wf:user()}/${examplesRoot}/
        </fs>
        <ok to="hdfs-node"/>
        <error to="fail"/>
    </action>

    <action name="hdfs-node">
        <fs>
            <move source="${nameNode}/user/${wf:user()}/${examplesRoot}/
                  target="/user/${wf:user()}/${examplesRoot}/output-data
        </fs>
        <ok to="end"/>
```

Oozie (XML)

/input_folder

Structured Workflows

```
{
    "Defaults" : {
        "Application" : "SampleApp",
        "SubApplication" : "SampleSubApp",
        "RunAs" : "USERNAME",
        "Host" : "HOST",
        "Job": {
            "When" : {
                "Months": ["JAN", "OCT", "DEC"],
                "MonthDays":["22","1","11"],
                "WeekDays":["MON","TUE", "WED", "THU", "FRI"],
                "FromTime":"0300",
                "ToTime":"2100"
            }
        }
    },
    "AutomationAPISampleFlow": {
        "Type": "Folder",
        "Comment" : "Code reviewed by John",
        "CommandJob": {
            "Type": "Job:Command",
            "Command": "echo my 1st job"
        },
        "ScriptJob": {
            "Type": "Job:Script",
            "FilePath":"SCRIPT_PATH",
            "FileName":"SCRIPT_NAME"
        },
        "Flow": {
            "Type": "Flow",
            "Sequence": ["CommandJob", "ScriptJob"]
        }
    }
}
```

Client machine details

Schedule interval

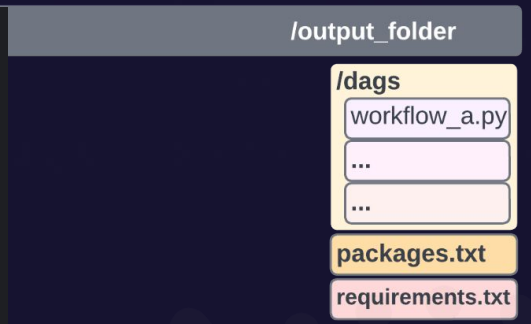
Control-M (JSON, XML)

Job specifics

Dependency

Source: https://airflowsummit.org/slides/2022/b1-choosing-airflow-v2.pdf Author: Parnab Basak

```python
with DAG(
    dag_id="sample_workflow_with_external_event.demo_fin_sample_workflow",
    schedule=None,
    start_date=DateTime( year: 1970, month: 1, day: 1),
    catchup=False,
    tags=["sample_workflow_with_external_event"],
):
    demo_fin_demo_service_task = EmptyOperator(
        task_id="demo_fin_demo_service",
        doc="Sample job flow with multiple branches and using site standards",
    )
    demo_fin_dynamic_event_task = EmptyOperator(
        task_id="demo_fin_dynamic_event",
    )
    demo_fin_job000_task = BashOperator(
        task_id="demo_fin_job000",
        bash_command="sleep 300",
    )
    demo_fin_job001_task = BashOperator(
        task_id="demo_fin_job001",
        bash_command="sleep 300",
        on_success_callback=send_smtp_notification(
            to="alerts@astronomer.io",
            smtp_conn_id="SMTP",
            subject="Error occurred",
            html_content="Job {{ti.task_id}} failed",
        ),
    )
```



Apache Airflow Project

/input_folder

/output_folder

/dags
- workflow_a.py
- ...
- ...

packages.txt

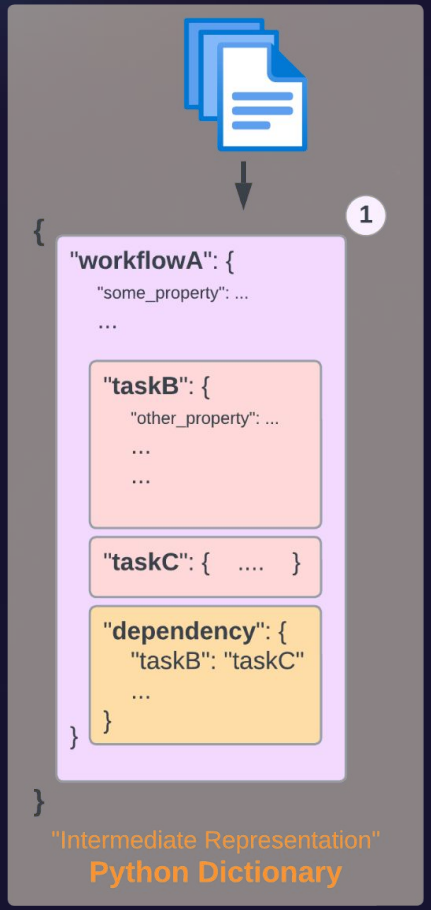
requirements.txt

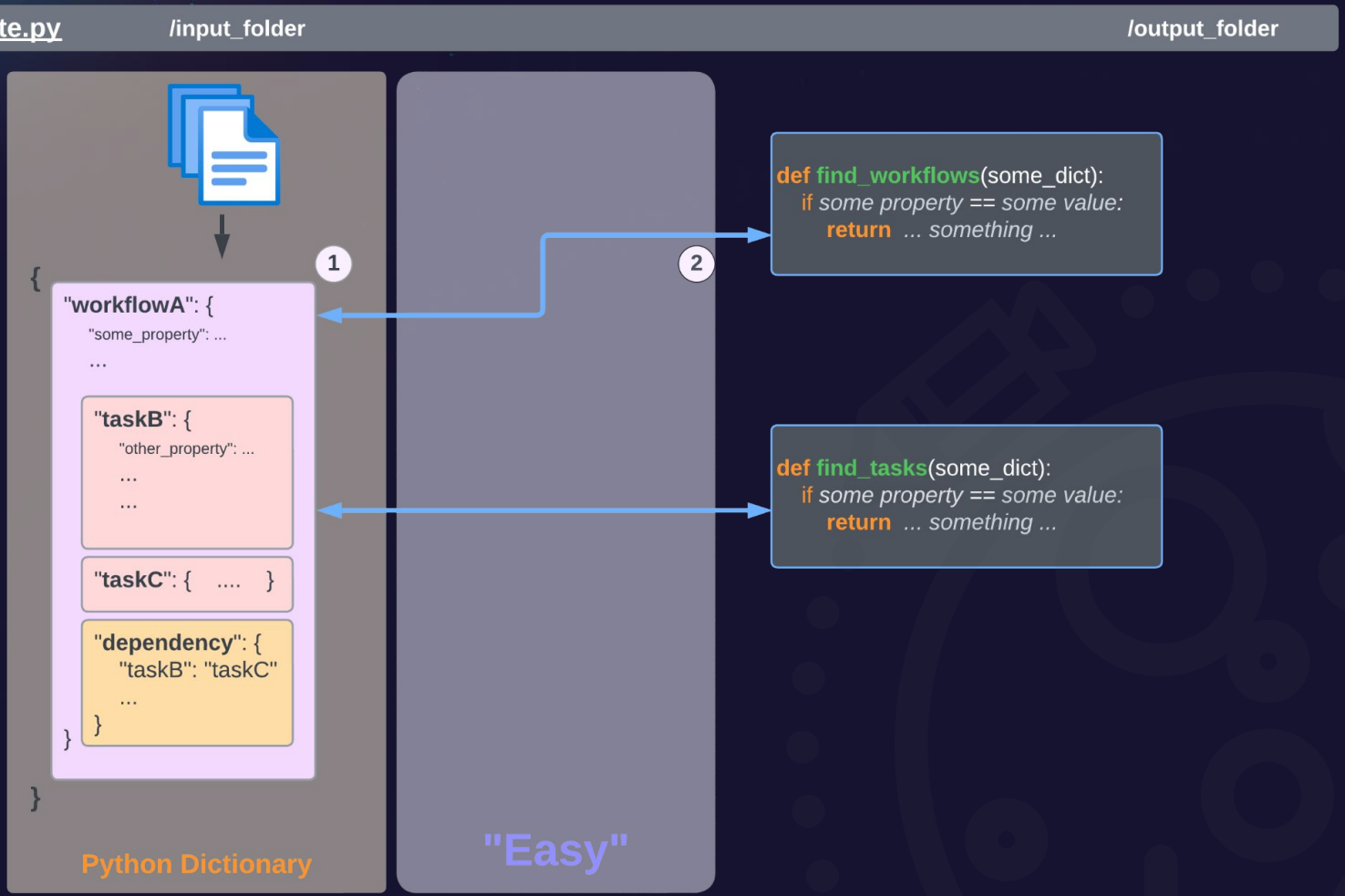
Structured Workflows

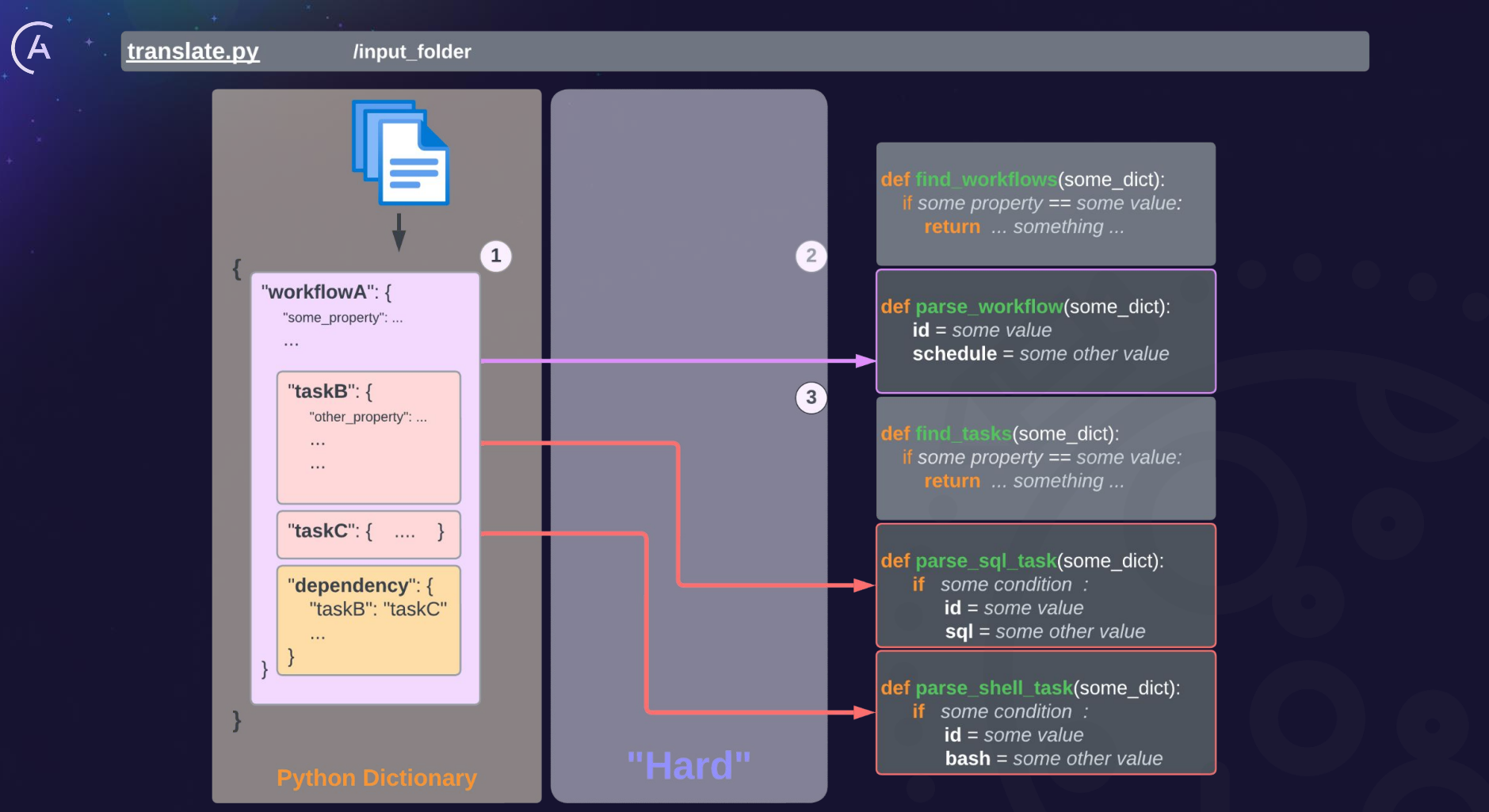
✨Translate✨

Apache Airflow Project

/input_folder /output_folder




translate.py
/input_folder
/output_folder
{
"workflowA": {
"some_property": ...
...
"taskB": {
"other_property": ...
...
...
"taskC": { .... }
"dependency": {
"taskB": "taskC"
...
}
}
}
1
2
def find_workflows(some_dict):
if some property == some value:
return ... something ...
def find_tasks(some_dict):
if some property == some value:
return ... something ...
Python Dictionary
"Easy"


translate.py
/input_folder
{
"workflowA": {
"some_property": ...
...
"taskB": {
"other_property": ...
...
...
"taskC": { .... }
"dependency": {
"taskB": "taskC"
...
}
}
}
Python Dictionary
1
2
3
"Hard"
def find_workflows(some_dict):
if some property == some value:
return ... something ...
def parse_workflow(some_dict):
id = some value
schedule = some other value
def find_tasks(some_dict):
if some property == some value:
return ... something ...
def parse_sql_task(some_dict):
if some condition :
id = some value
sql = some other value
def parse_shell_task(some_dict):
if some condition :
id = some value
bash = some other value

translate.py /input_folder

```
{
    "Defaults" : {
        "Application" : "SampleApp",
        "SubApplication" : "SampleSubApp",
        "RunAs" : "USERNAME",
        "Host" : "HOST",
        "Job": {
            "When" : {
                "Months": ["JAN", "OCT", "DEC"],
                "MonthDays":["22","1","11"],
                "WeekDays":["MON","TUE", "WED", "THU", "FRI"],
                "FromTime":"0300",
                "ToTime":"2100"
            }
        }
    },
    "AutomationAPISampleFlow": {
        "Type": "Folder",
        "Comment" : "Code reviewed by John",
        "CommandJob": {
            "Type": "Job:Command",
            "Command": "echo my 1st job"
        },
        "ScriptJob": {
            "Type": "Job:Script",
            "FilePath":"SCRIPT_PATH",
            "FileName":"SCRIPT_NAME"
        },
        "Flow": {
            "Type": "Flow",
            "Sequence": ["CommandJob", "ScriptJob"]
        }
    }
```

```
def find_workflows(some_dict):
    if some property == some value:
        return ... something ...
```

```
def parse_workflow(some_dict):
    id = some value
    schedule = some other value
```

```
def find_tasks(some_dict):
    if some property == some value:
        return ... something ...
```

```
def parse_sql_task(some_dict):
    if some condition :
        id = some value
        sql = some other value
```

```
def parse_shell_task(some_dict):
    if some condition :
        id = some value
        bash = some other value
```

translate.py /input_folder

Rule = Filter or map input (e.g. this *specific input* converts to SQL Operator)

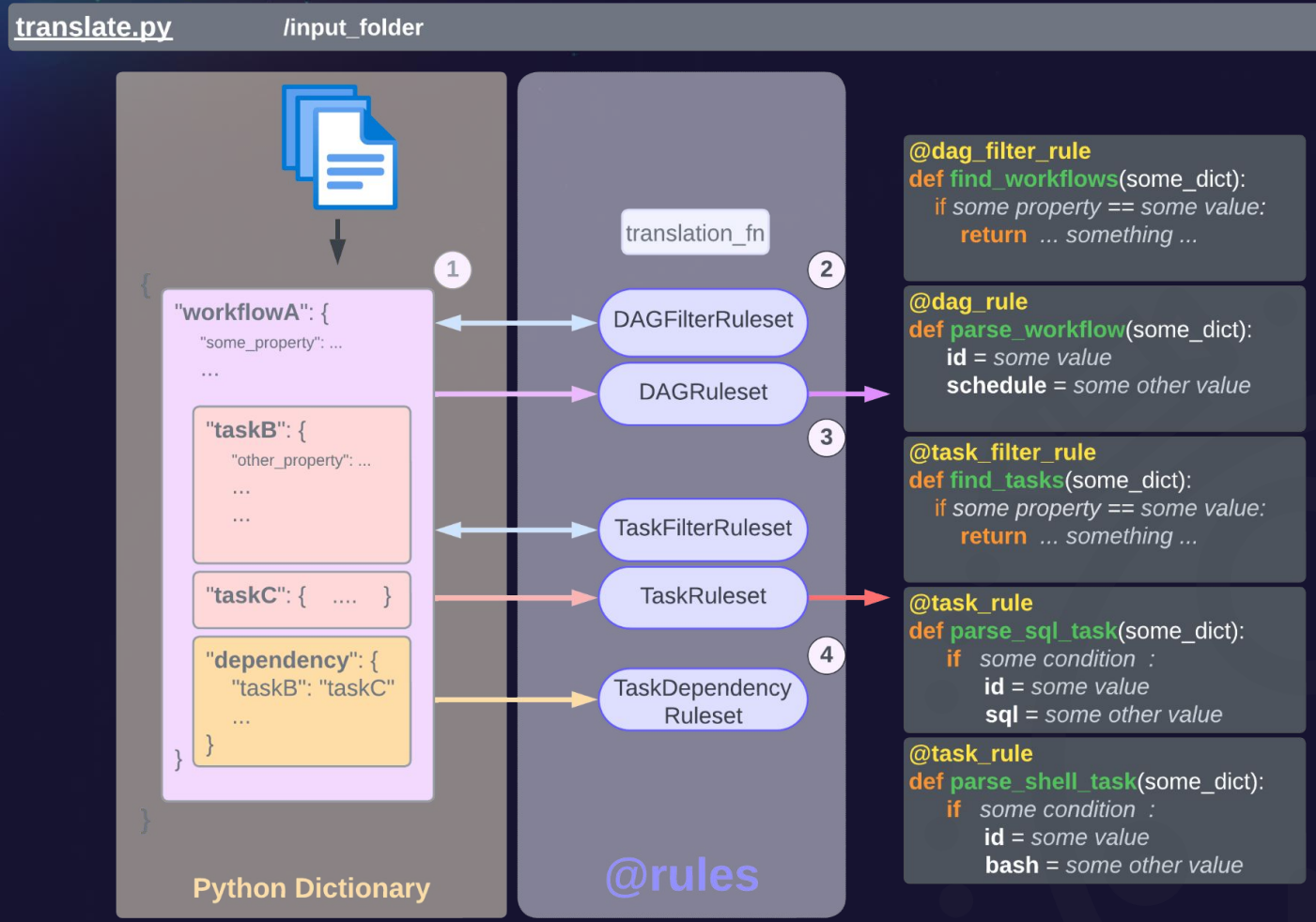
Ruleset = Collection of rules (e.g. these rules filter to something DAG-like)

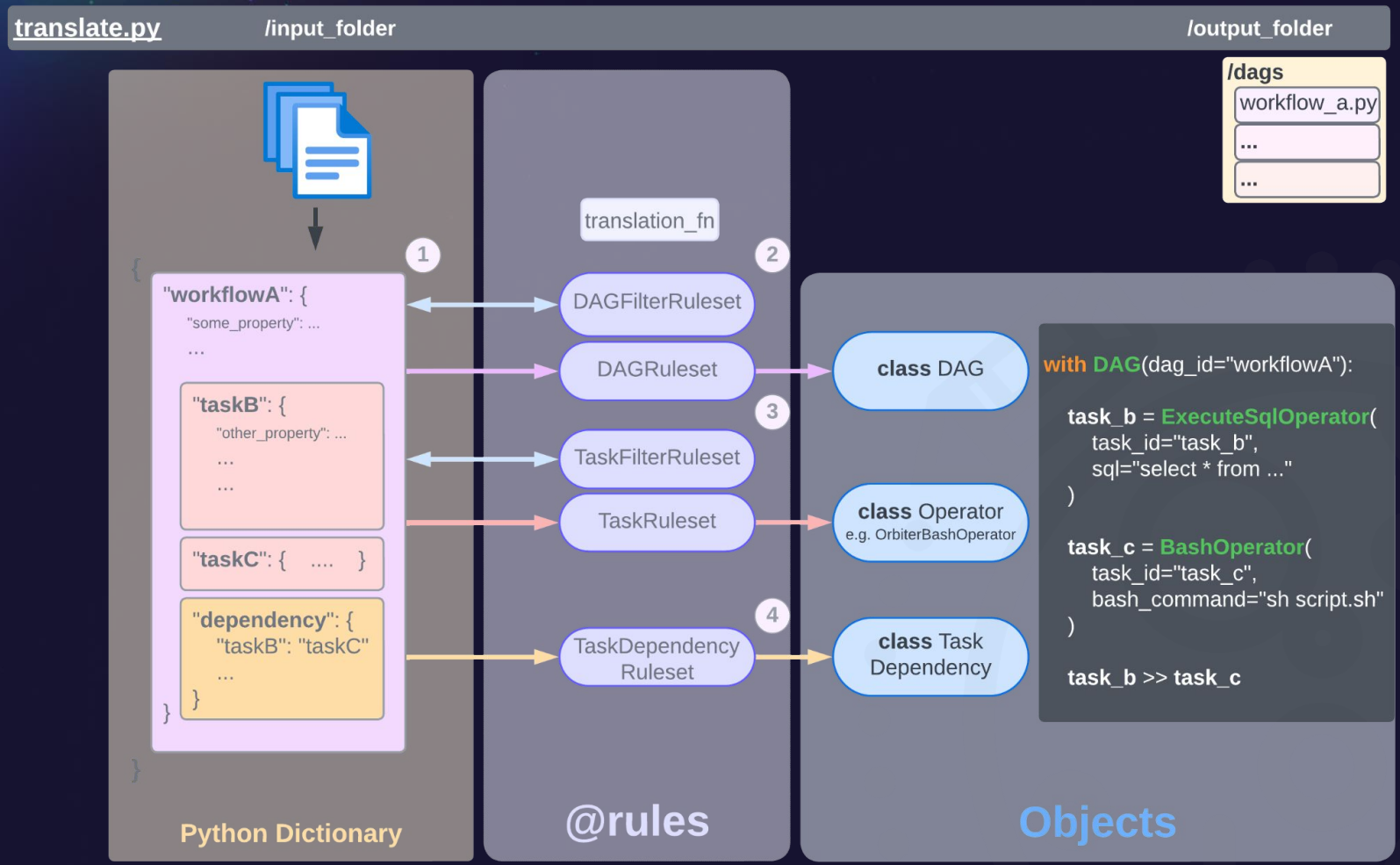
```
@dag_filter_rule
def find_workflows(some_dict):
```

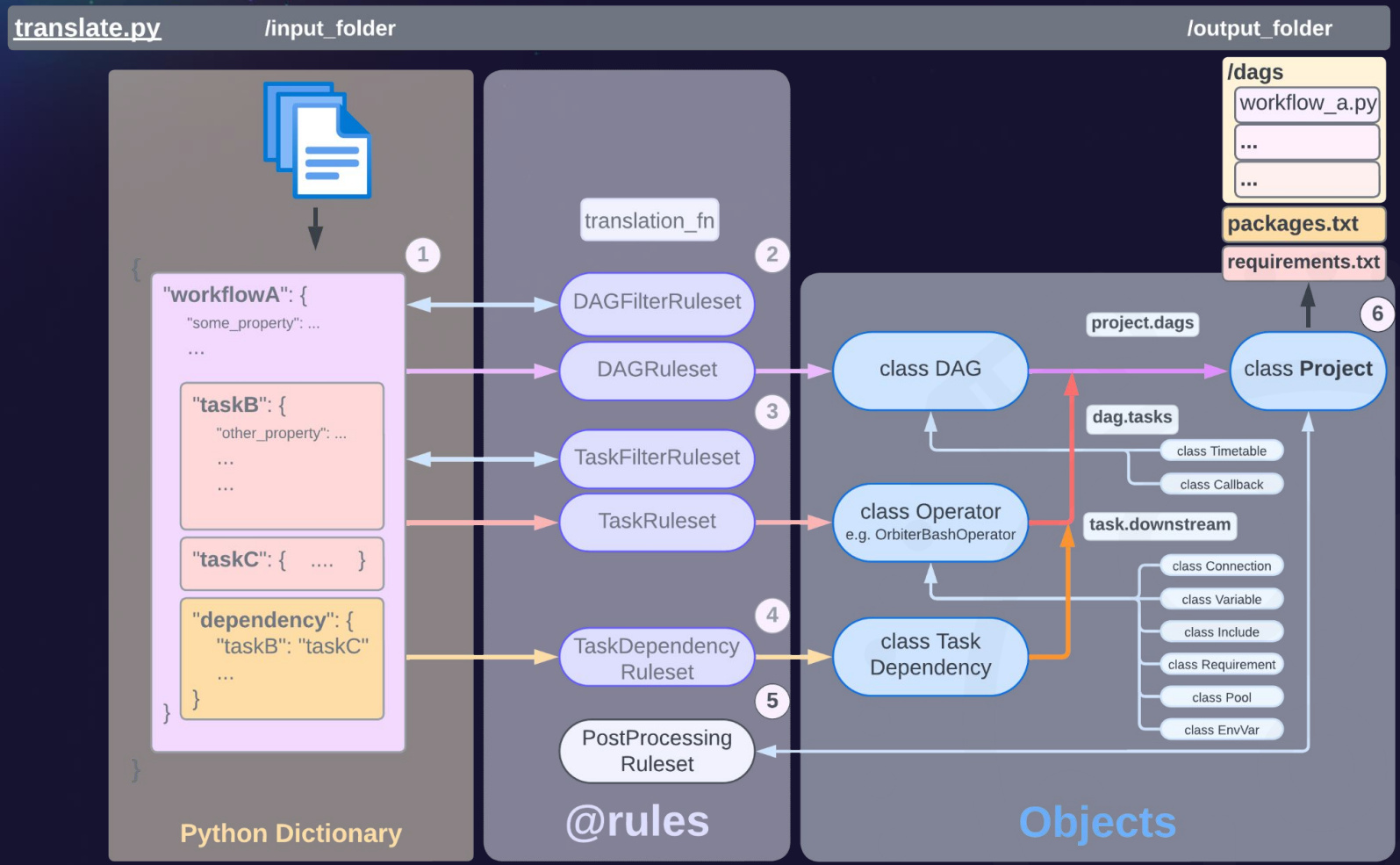
```
def parse_shell_task(some_dict):
    if  some condition  :
        id = some value
        bash = some other value
```

Python Dictionary

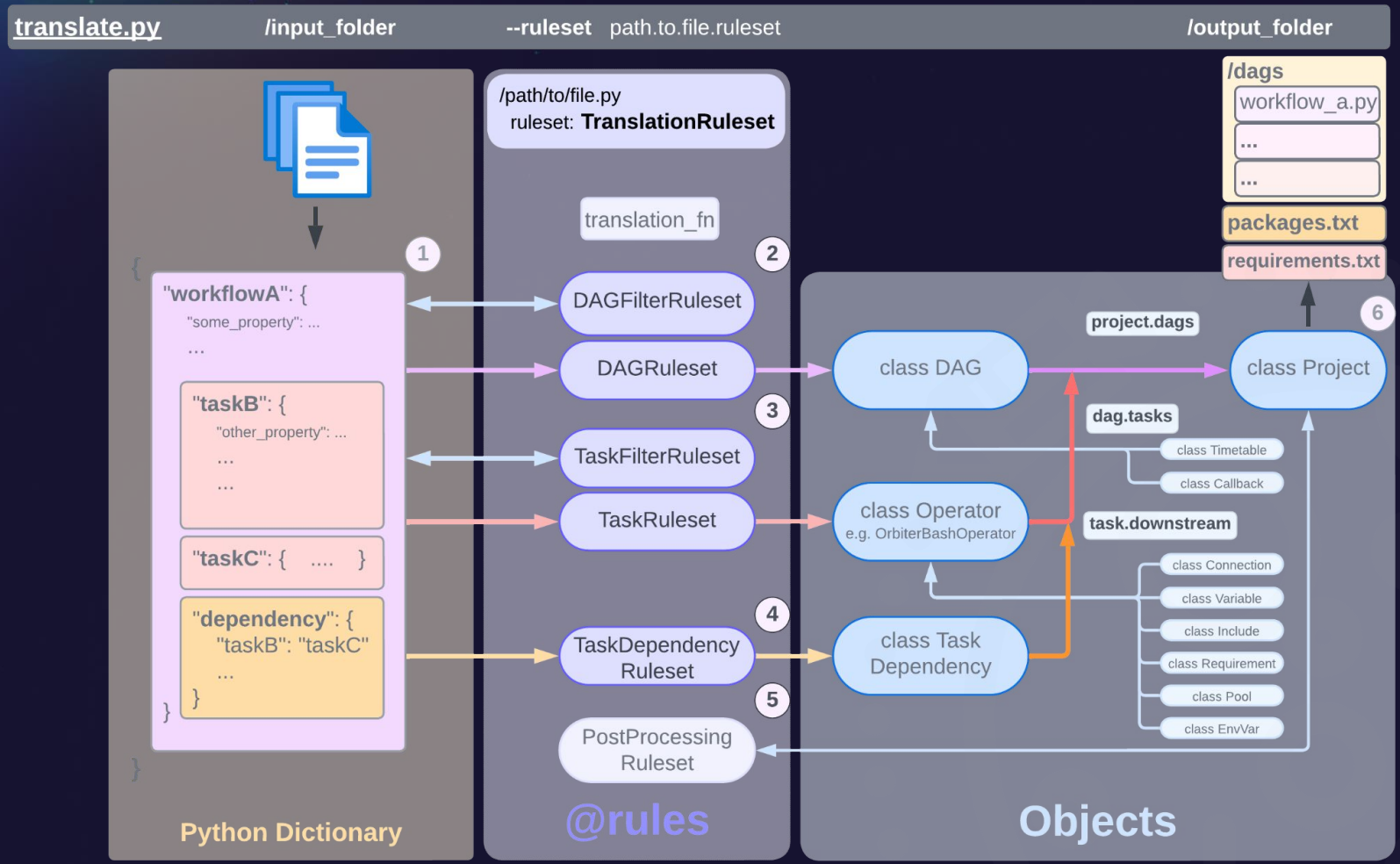
@rules


translate.py
/input_folder
translation_fn
1
2
3
4
"workflowA": {
"some_property": ...
...
"taskB": {
"other_property": ...
...
...
"taskC": { .... }
"dependency": {
"taskB": "taskC"
...
}
}
Python Dictionary
DAGFilterRuleset
DAGRuleset
TaskFilterRuleset
TaskRuleset
TaskDependency Ruleset
@rules
@dag_filter_rule
def find_workflows(some_dict):
if some property == some value:
return ... something ...
@dag_rule
def parse_workflow(some_dict):
id = some value
schedule = some other value
@task_filter_rule
def find_tasks(some_dict):
if some property == some value:
return ... something ...
@task_rule
def parse_sql_task(some_dict):
if some condition :
id = some value
sql = some other value
@task_rule
def parse_shell_task(some_dict):
if some condition :
id = some value
bash = some other value

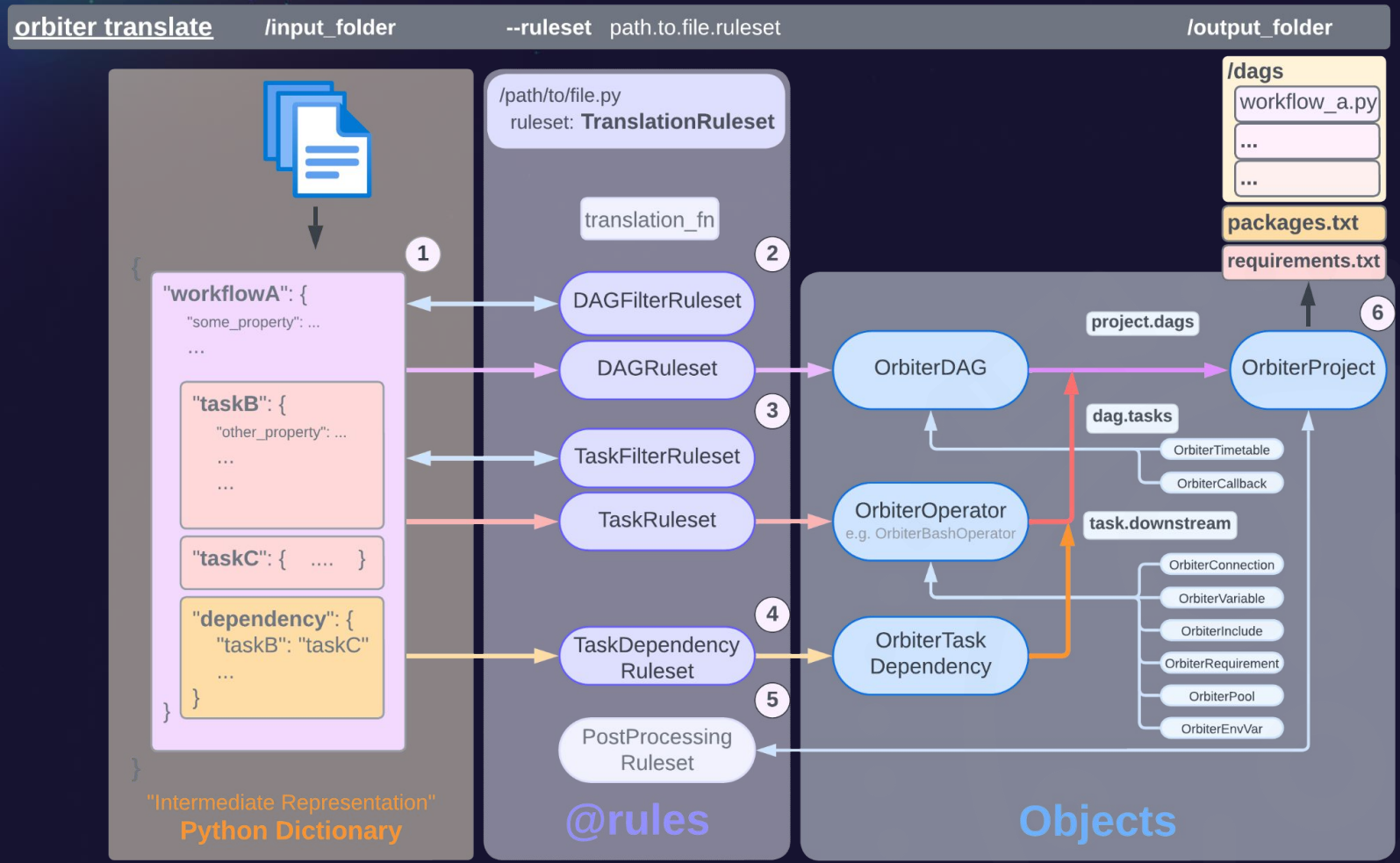

translate.py
/input_folder
/output_folder
/dags
workflow_a.py
...
...
translation_fn
1
2
3
4
"workflowA": {
"some_property": ...
...
"taskB": {
"other_property": ...
...
...
"taskC": { .... }
"dependency": {
"taskB": "taskC"
...
}
}
DAGFilterRuleset
DAGRuleset
TaskFilterRuleset
TaskRuleset
TaskDependency Ruleset
class DAG
class Operator
e.g. OrbiterBashOperator
class Task Dependency
with DAG(dag_id="workflowA"):
task_b = ExecuteSqlOperator(
task_id="task_b",
sql="select * from ..."
)
task_c = BashOperator(
task_id="task_c",
bash_command="sh script.sh"
)
task_b >> task_c
Python Dictionary
@rules
Objects


translate.py
/input_folder
/output_folder
/dags
workflow_a.py
...
...
packages.txt
requirements.txt
translation_fn
1
2
3
4
5
6
"workflowA": {
"some_property": ...
...
"taskB": {
"other_property": ...
...
...
"taskC": { .... }
"dependency": {
"taskB": "taskC"
...
}
}
DAGFilterRuleset
DAGRuleset
TaskFilterRuleset
TaskRuleset
TaskDependency Ruleset
PostProcessing Ruleset
class DAG
class Operator
e.g. OrbiterBashOperator
class Task Dependency
class Project
project.dags
dag.tasks
task.downstream
class Timetable
class Callback
class Connection
class Variable
class Include
class Requirement
class Pool
class EnvVar
Python Dictionary
@rules
Objects

ASTRONOMER


translate.py
/input_folder
--ruleset path.to.file.ruleset
/output_folder
/dags
workflow_a.py
...
...
packages.txt
requirements.txt
/path/to/file.py
ruleset: TranslationRuleset
translation_fn
1
2
3
4
5
6
"workflowA": {
"some_property": ...
...
"taskB": {
"other_property": ...
...
...
"taskC": { .... }
"dependency": {
"taskB": "taskC"
...
}
}
Python Dictionary
DAGFilterRuleset
DAGRuleset
TaskFilterRuleset
TaskRuleset
TaskDependency Ruleset
PostProcessing Ruleset
@rules
class DAG
class Project
project.dags
dag.tasks
class Timetable
class Callback
class Operator
e.g. OrbiterBashOperator
task.downstream
class Connection
class Variable
class Include
class Requirement
class Pool
class EnvVar
class Task Dependency
Objects

ASTRONOMER


orbiter translate
/input_folder
--ruleset path.to.file.ruleset
/output_folder
/dags
workflow_a.py
...
...
packages.txt
requirements.txt
/path/to/file.py
ruleset: TranslationRuleset
translation_fn
1
2
3
4
5
6
{
"workflowA": {
"some_property": ...
...
"taskB": {
"other_property": ...
...
...
"taskC": { .... }
"dependency": {
"taskB": "taskC"
...
}
}
}
"Intermediate Representation"
Python Dictionary
DAGFilterRuleset
DAGRuleset
TaskFilterRuleset
TaskRuleset
TaskDependency Ruleset
PostProcessing Ruleset
@rules
OrbiterDAG
project.dags
OrbiterProject
dag.tasks
OrbiterTimetable
OrbiterCallback
OrbiterOperator
e.g. OrbiterBashOperator
task.downstream
OrbiterConnection
OrbiterVariable
OrbiterInclude
OrbiterRequirement
OrbiterPool
OrbiterEnvVar
OrbiterTask Dependency
Objects

ASTRONOMER

Orbiter

Land legacy workloads safely down in a new home on Apache Airflow!

Any system*

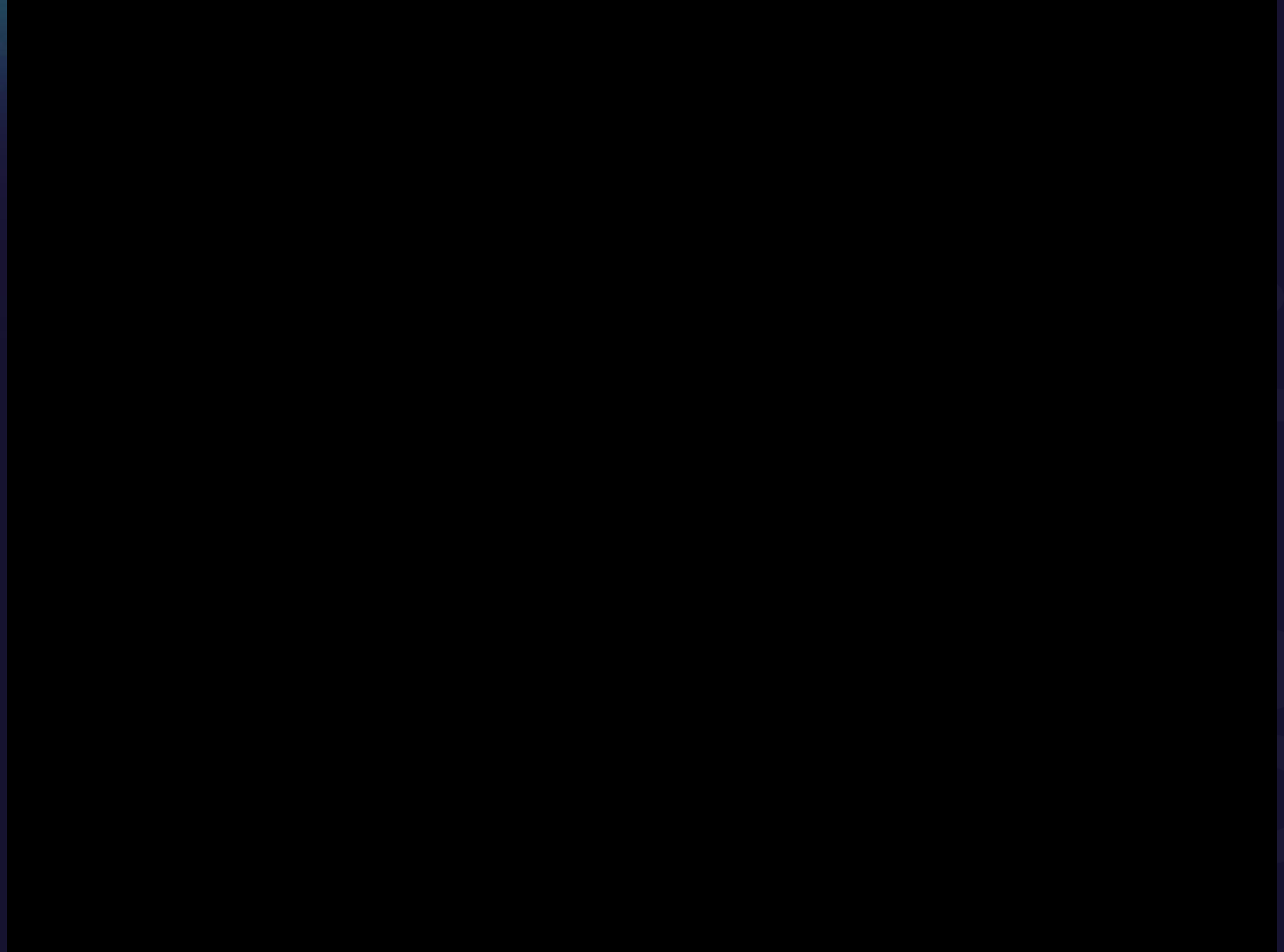
Framework

Extendable & Open Source

Batteries included

1. `pip install astronomer-orbiter`
2. `orbiter **list-rulesets**`
3. `orbiter **install** ...`
4. `orbiter **translate** **_<input dir>_** --ruleset **_<ruleset>_** **_<output dir>_**`

CLI



Orbiter

Migrate (m)any legacy workloads to Airflow!

https://astronomer.github.io/orbiter/

Questions?

Community Translations

Contribute yours!

https://github.com/astronomer/orbiter-community-translations

Astronomer Translations & Assistance

A Team of Airflow Experts to Guide Your Migration!

https://www.astronomer.io/professional-services/

Acknowledgements

"Choosing Apache Airflow over other Proprietary Tools..."
Parnab Basak, Airflow Summit 2022

"Observations from migrating away from Control-M to Airflow"
Arjun Anandkumar

Astronomer Migration White Papers

"DAGify - Enterprise Scheduler Migration Accelerator for Airflow"
Konrad Schieban

& more