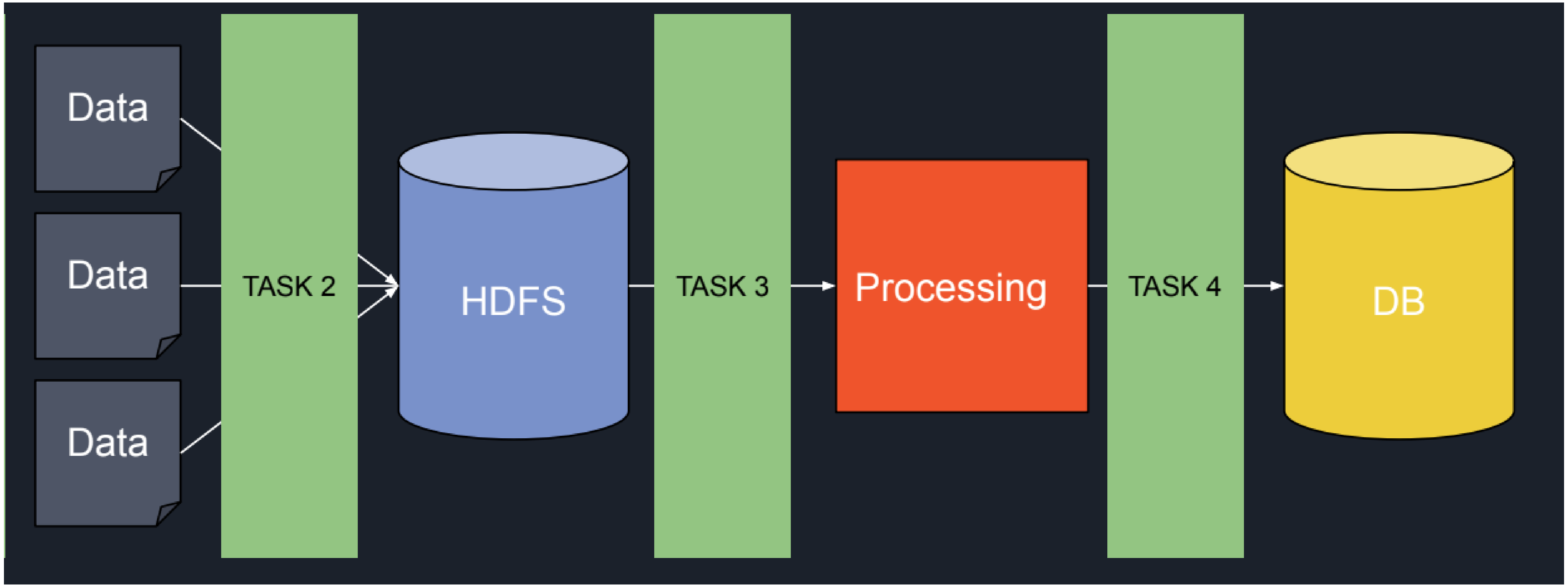
Apache Airflow

Content

- Introduction to the Airflow
- Core components of Airflow. (Web server, scheduler, meta store, executor*, Worker)
- DAG, Operators, Task/Task Instance, Workflow
- How Airflow works ?
- Airflow Useful Features
    - Subdags,
    - Taskgroups,
    - XComs,
    - Connections,
    - Variables,
    - Trigger Rules
    - Task flow API
- Working Demo, Troubleshooting



Why Airflow?

Let's imagine…

You are working for a company and you have a data processing pipeline to run every day at 9 AM which does the following:

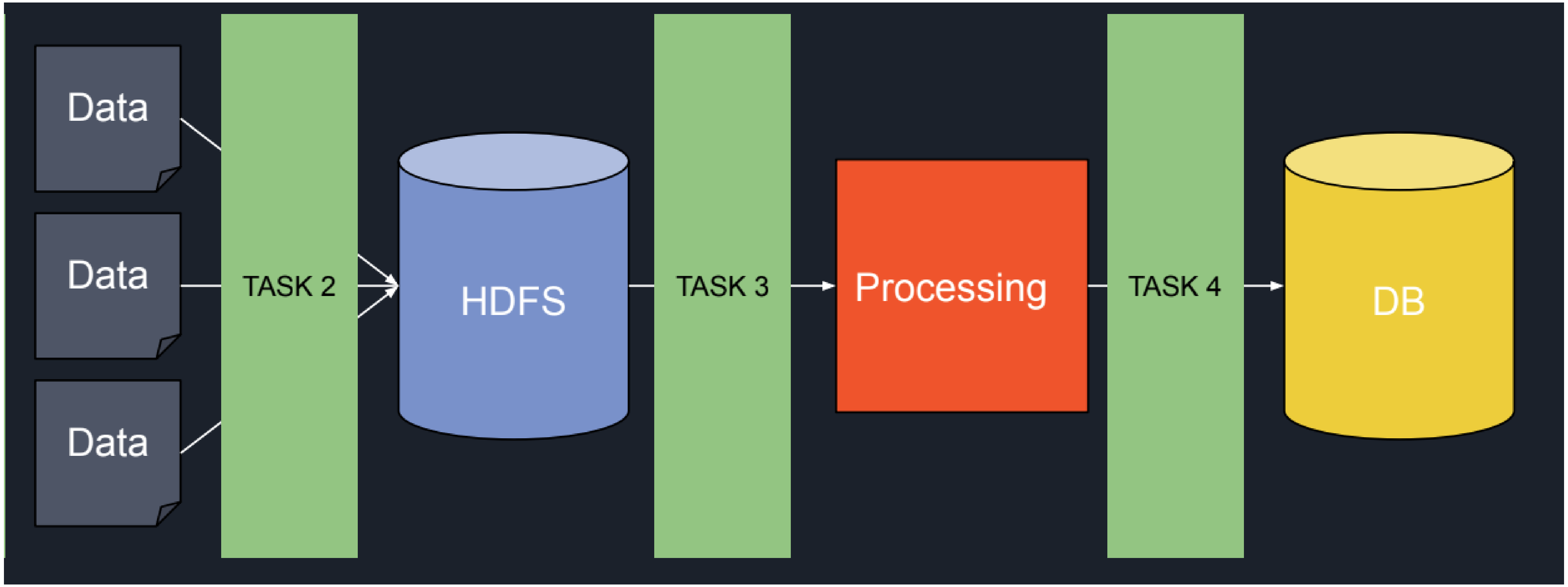
1. Wait for files to come in a specific directory

2. Store all the files into HDFS

3. Run a Spark job to process those files

4. Check the result of the job from the PostgreSQL database

Problems?



Why Airflow?

Problems

- What if … a file does not arrived in time?
- What if … my Spark job failed?
- What if … I have 1000 pipelines to execute?

Why Airflow

Airflow handles those problems and more

- Cron Replacement
- Fault Tolerant
- Dependency Rules
- Python Code
- Handle Task Failures
- Report / Alert on failures
- Extensible and modulable
- Beautiful UI

Airflow is a perfect tool in order to create, monitor and manage your data pipelines.

What is Airflow

- **Apache Airflow is an open source platform to author, schedule and monitor workflows**

What Airflow is Not?

Airflow is not a data streaming solution

- Airflow is not in the scope of Apache Spark or Storm.
- Primarily built to perform scheduled batch jobs

Cloud providers -

aws

Google Cloud

ASTRONOMER

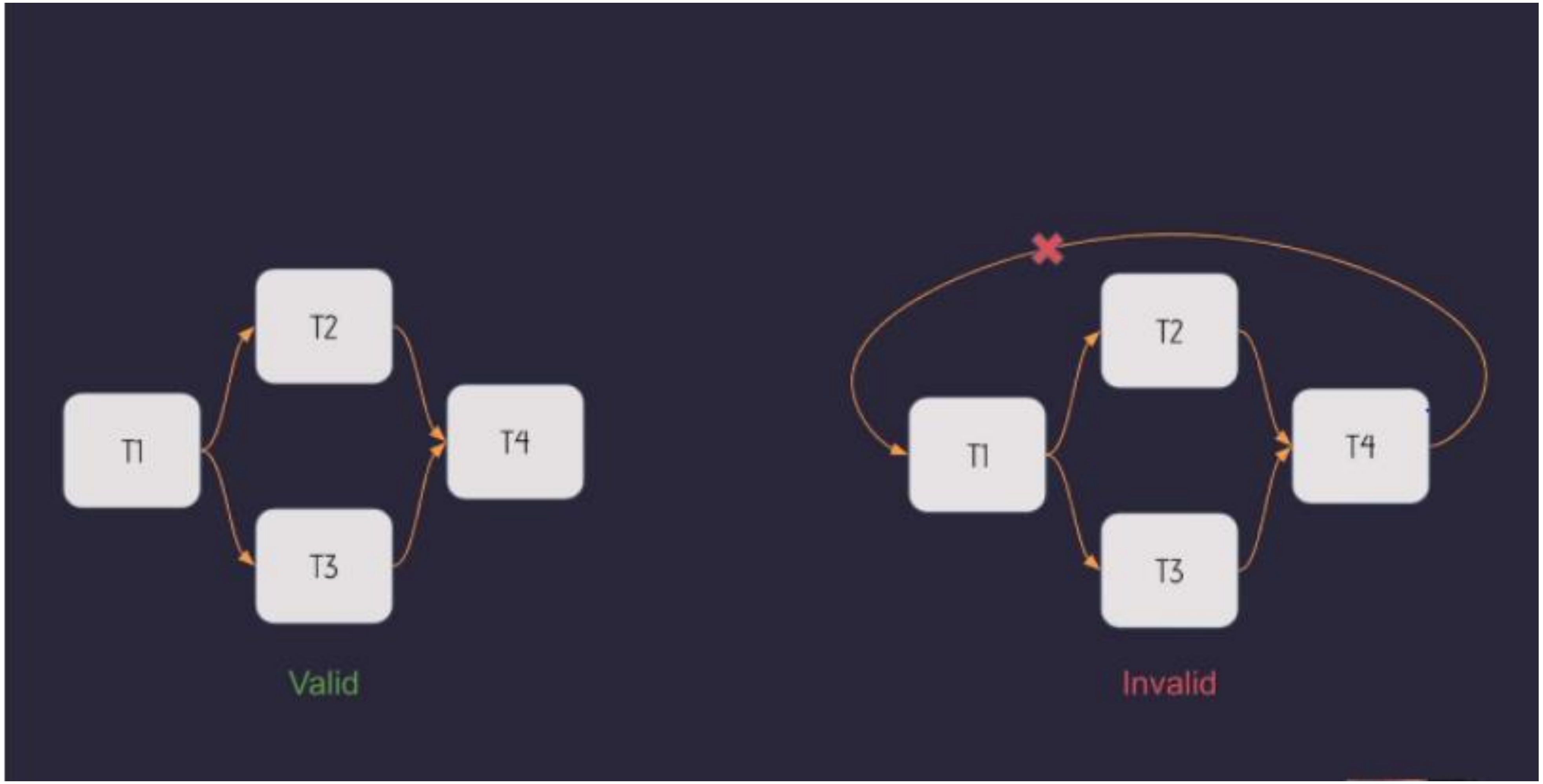
Core Components of Airflow

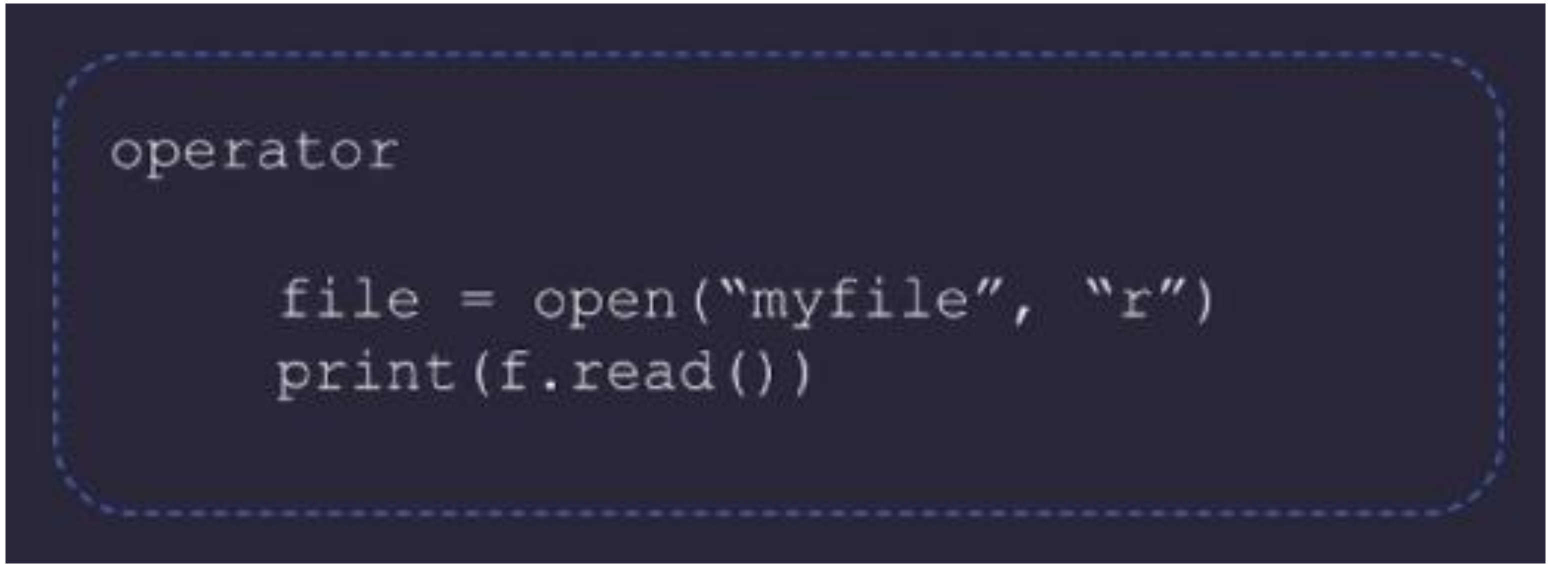
- Web Server
- Scheduler
- Metastore database
- 🟦 Executor
- 🟩 Worker

Key Concepts

- 🟧 DAG
- ⬜ Operator
- 🟨 Task
- 🟦 Task Instance
- 🟩 Workflow

- DAGS





```
operator

    file = open("myfile", "r")
    print(f.read())
```

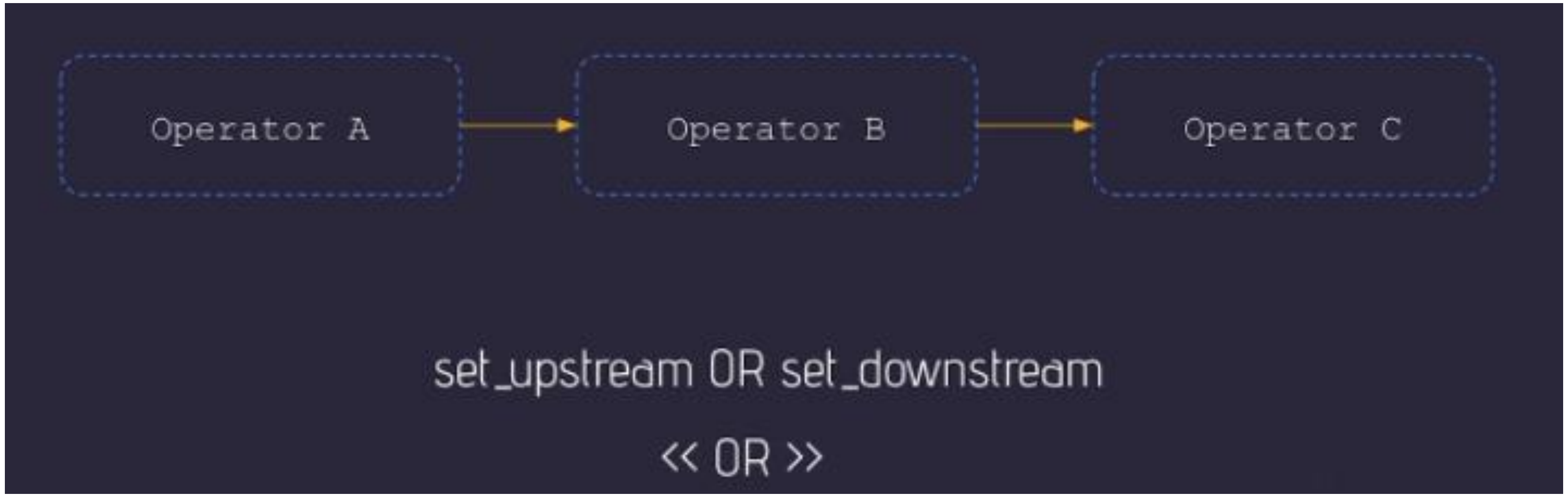
Click to add text

Operator

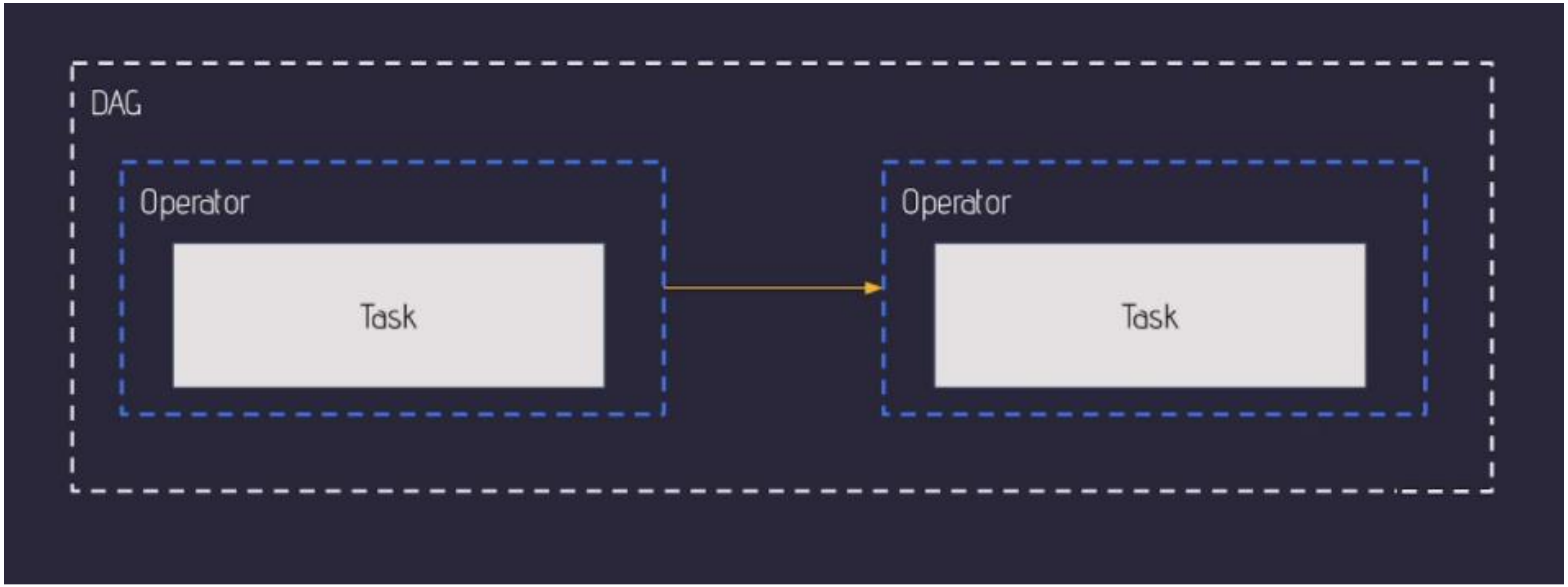
3 Types

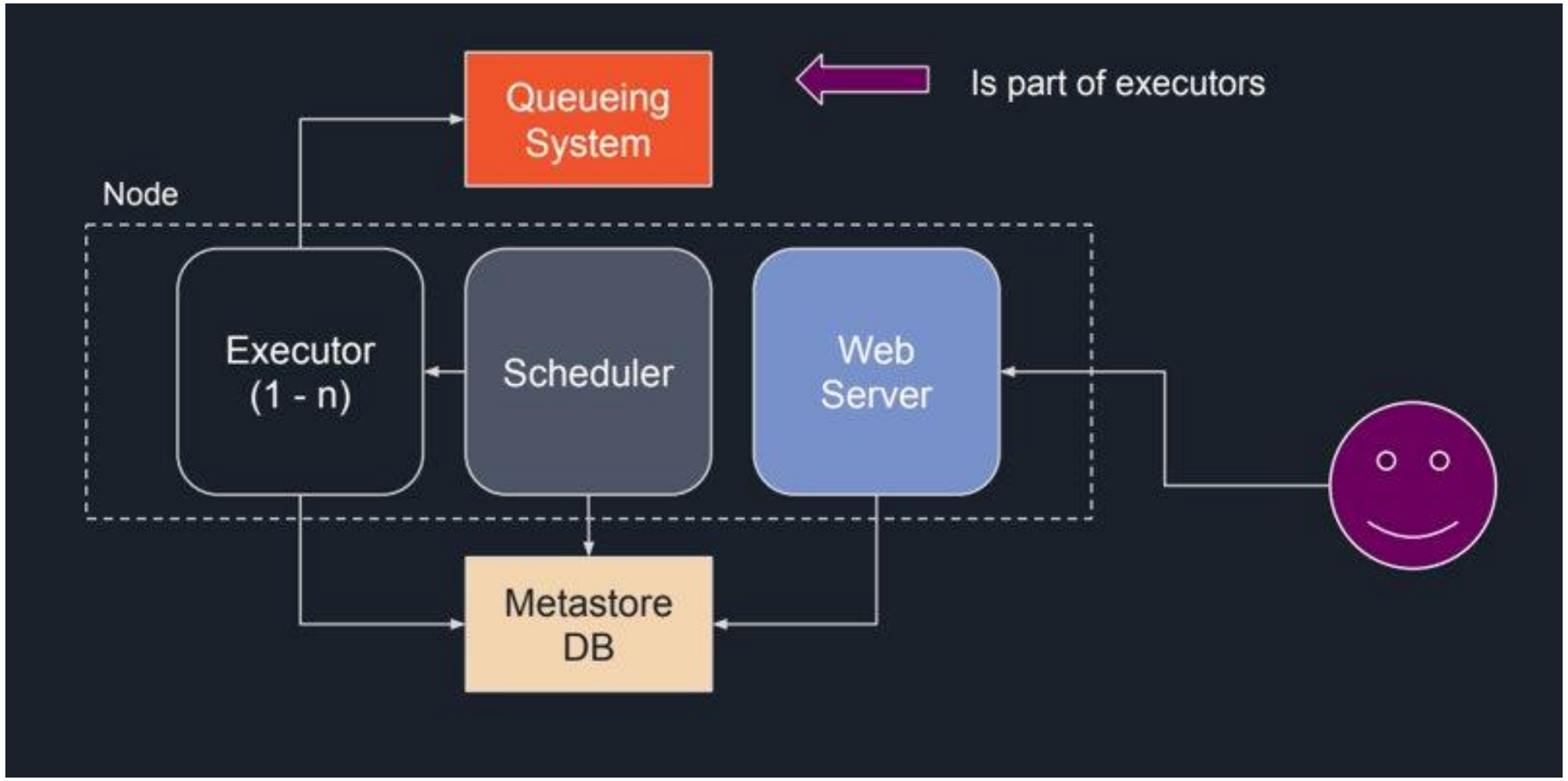
- **Action Operators**
- **Transfer Operators**
- **Sensor Operators**

- TASK
- TASK INSTANCE
- DEPENDENCIES

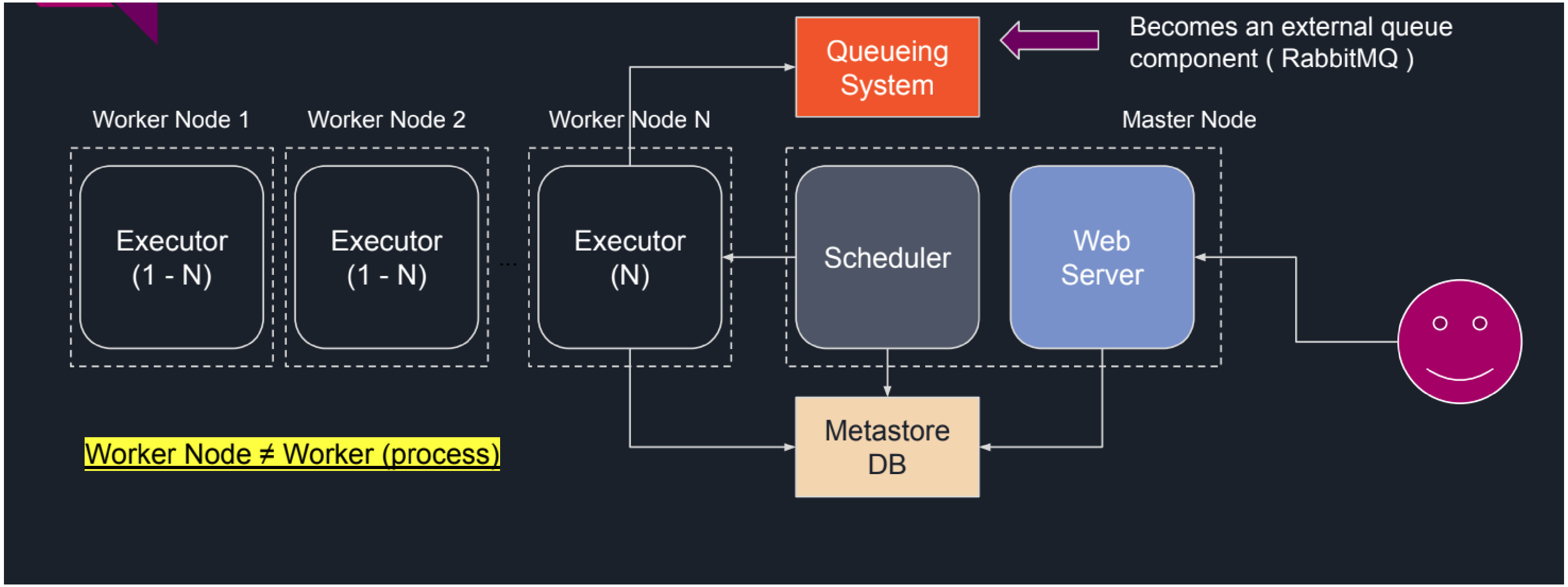


- WORKFLOW

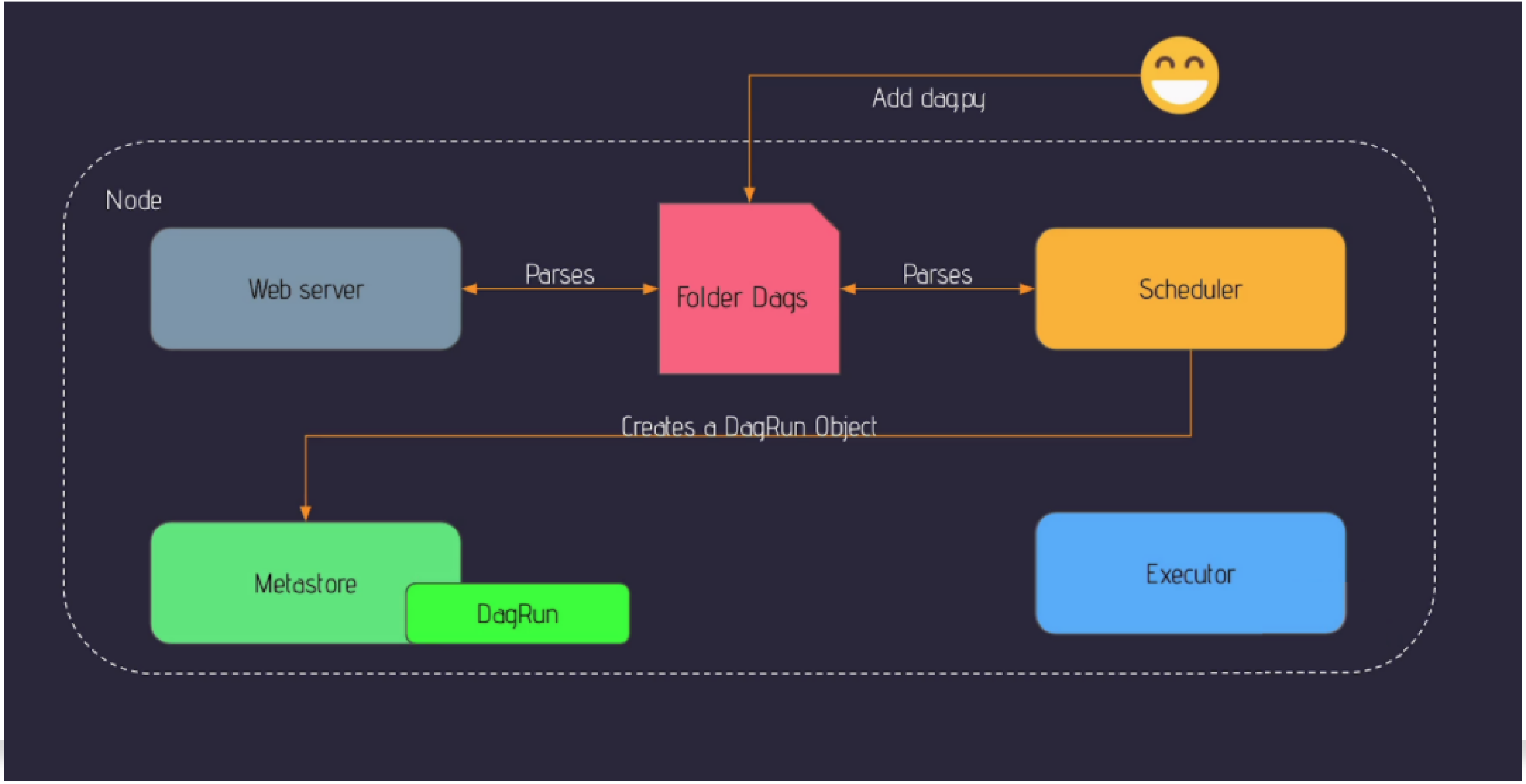




Airflow Single Node Architecture

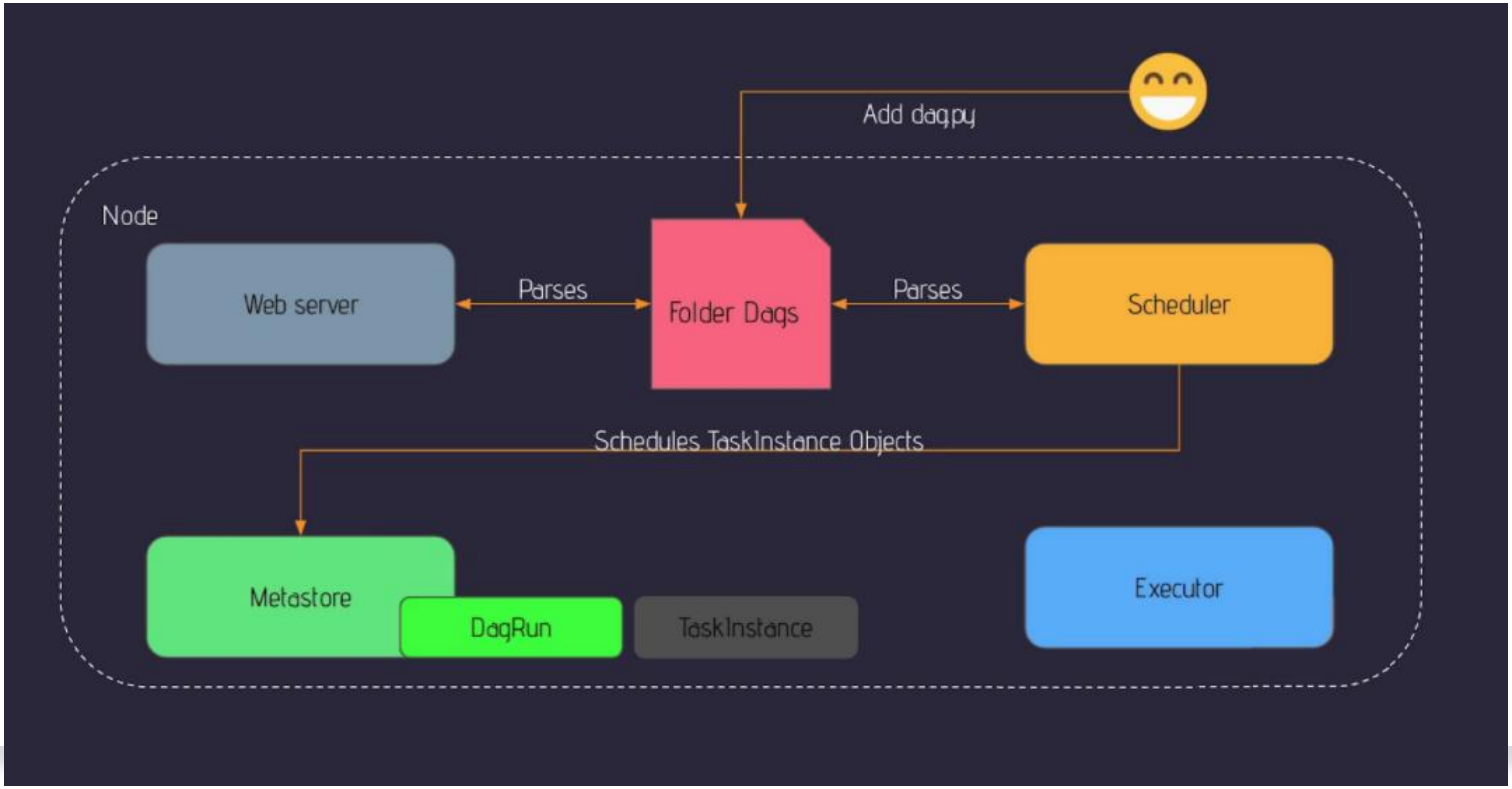


Airflow Multinode Architecture



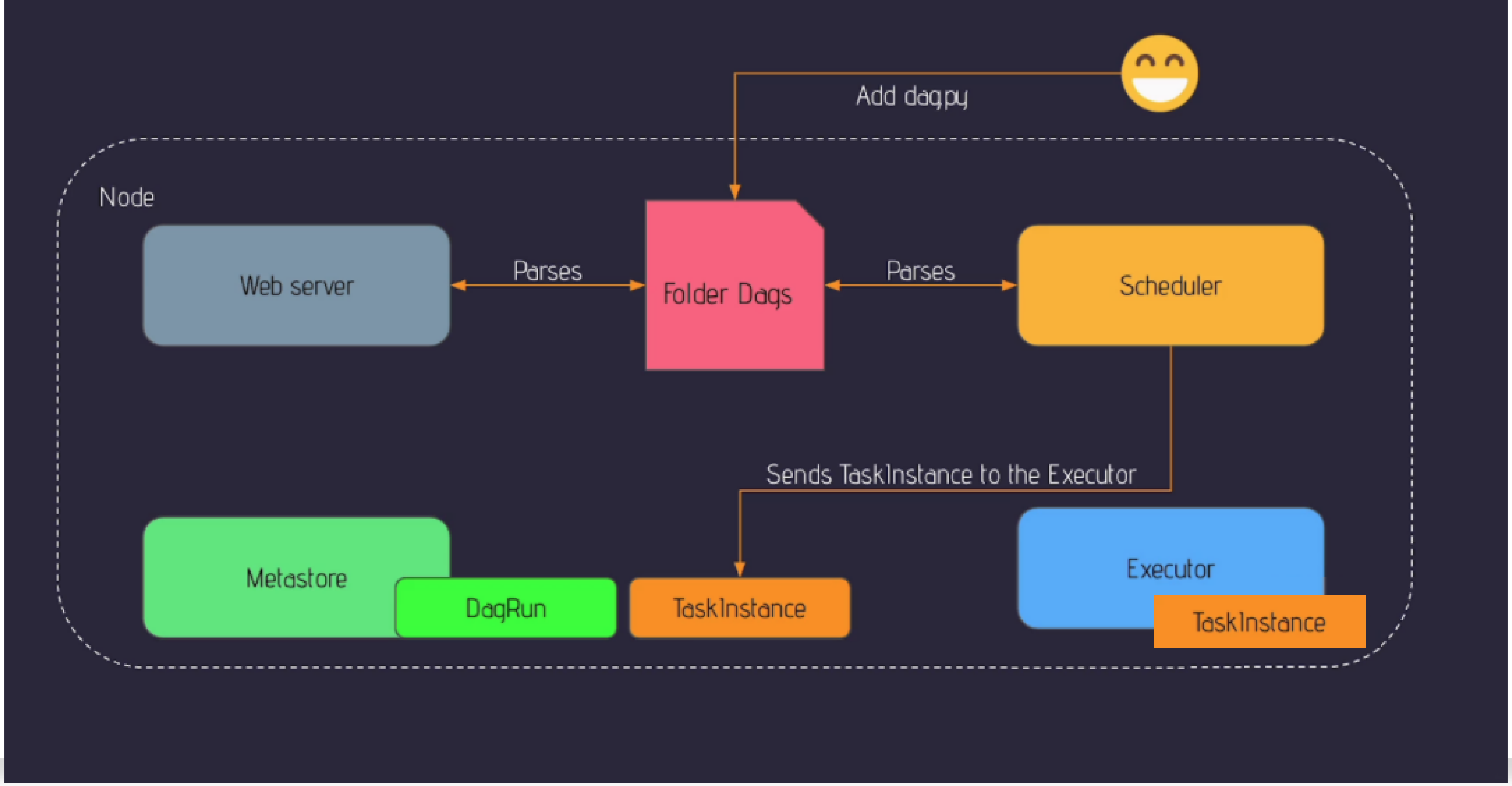
How Airflow Works?

- Upload
- DAG Parsing – webserver and scheduler
- DagRun Object in Metadata



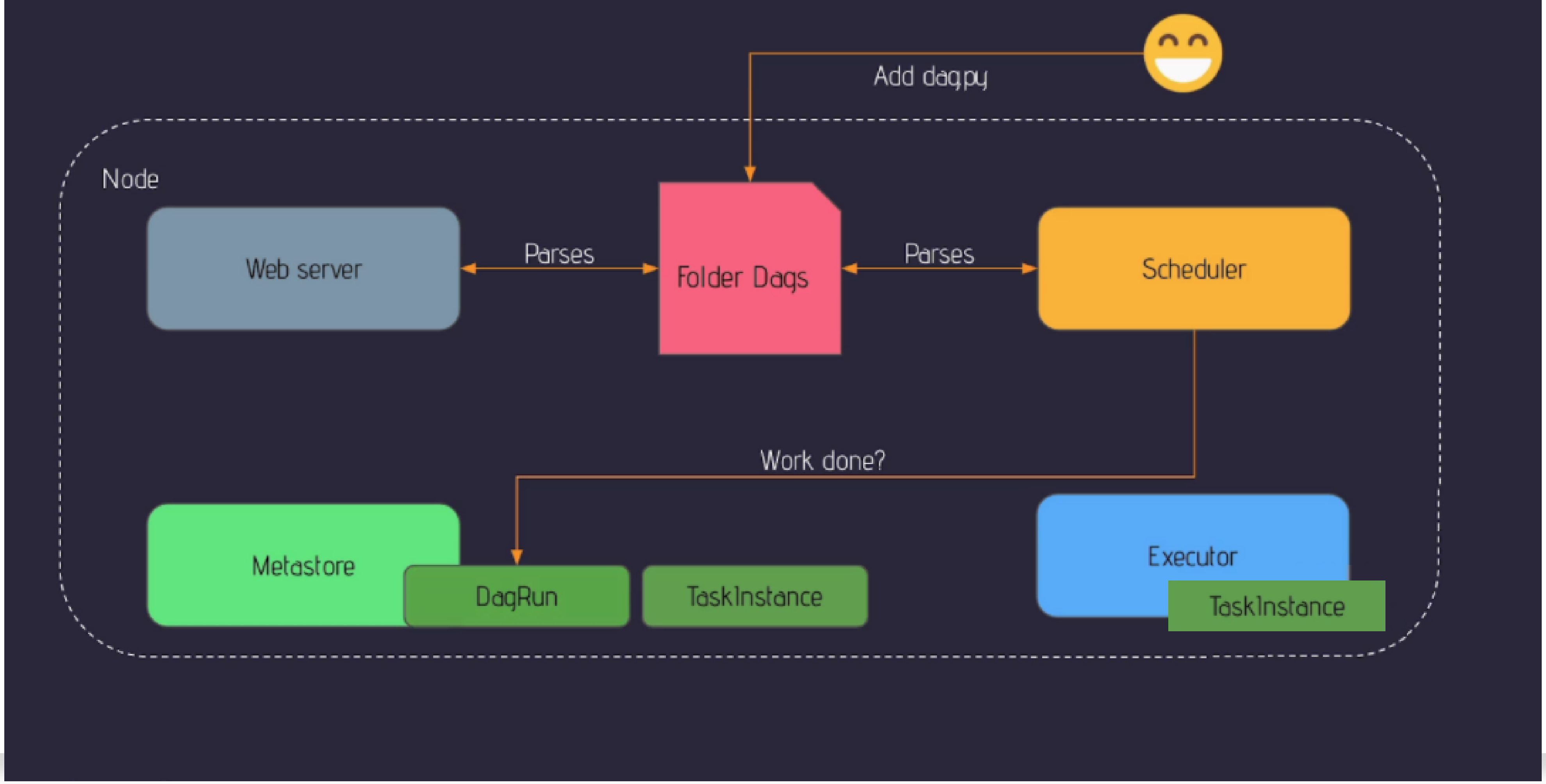
How Airflow Works?

- Task Instance is created by Scheduler,
  - No Status in the beginning.
  - Scheduled, when ready to be triggered



How Airflow Works?

- **Task Instance** status will be **Queued at this time.**
- Queued tasks are executed into the worker. Task state **running**.
- Executor updates the Task instance, status **running**, **failed** or **success** in the metastore.



How Airflow Works?

- Scheduler checks if more tasks exist? If all completed, DagRun is marked as success/failed
- In parallel to all of this, Web UI is being updated about the status of the task.
- **Task Lifecycle.**

No status → scheduled → queued → running → success/failed.

- Quick Tour of Web UI
- Quick Tour of Airflow CLI
- DAG and Operator definition in python
- Dependencies between tasks
- Connections
- Variables
- Extras and Provider Packages (Operators and hooks)
- XComs
- Branching
- Trigger Rules
- Sub Dags
- Taskgroups
- Task flow API

Quick Tour of Web UI localhost:8080

Quick Tour of Airflow CLI

NOTE : airflow 2.X has updated the CLI usage.

- airflow db init
- airflow db upgrade
- airflow db reset
- airflow webserver
- airflow scheduler
- airflow dags unpause <dag_id>
- airflow dags pause <dag_id>
- airflow dags trigger -e <execution_date> (-e optional)
- airflow dags list
- airflow tasks list <dag_id>
- airflow tasks test <dag_id> <task_id> <execution_date>
- airflow dags backfill -s <start_date> -e <end_date> --reset_dag_runs <dag_id>

DAG and Operator definition in python

Dependencies between tasks

- t1 >> t2
- t2 << t1
- t2.set_upstream(t1)
- t1.set_downstream(t2)
- Chain dependency
- Cross dependency

Airflow Connections

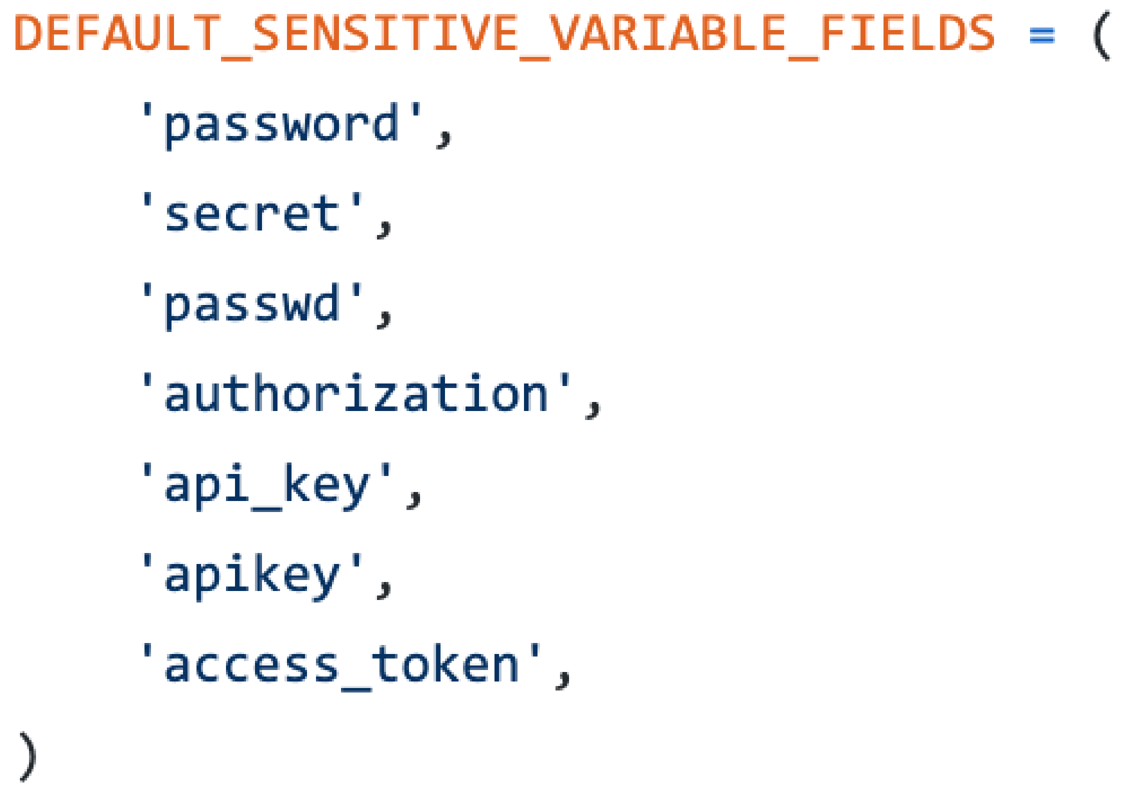
- FROM UI
- FROM REST API
- FROM CLI
  - airflow connections add 'my_prod_db' \
  -     --conn-type 'my-conn-type'
  -     ...
  -
- Export Connections from CLI
  - airflow connections export connections.json
  - airflow connections export /tmp/connections --format yaml
- Store Connection in Environmnet Variables
  - .bashrc
  - Docker.env

Airflow Variables

- How variables work?
- How to SET, GET a variable in Airflow?
- Optimizing variables with the JSON format
- Best practices with variables in Airflow

Airflow Variables

- How to hide the value of a variable?



```
DEFAULT_SENSITIVE_VARIABLE_FIELDS = (
    'password',
    'secret',
    'passwd',
    'authorization',
    'api_key',
    'apikey',
    'access_token',
)
```

PostgreSQL Connection and Operator

- Conn Type missing? Make sure you've installed the corresponding Airflow Provider Package.
- Provider Operator missing?
- Solution :
  - https://airflow.apache.org/docs/apache-airflow/stable/extra-packages-ref.html
  - pip install 'apache-airflow[postgres]'

Runtime Configs and XComs

- From airflow context
- Run time configs can be fetched from Dagrun object

- Xcoms -> Cross communication
- KEY VALUE TIMESTAMP
- Stored in Metadatabase, with an associated
  - Execution_date,
  - Task ID
  - DAG ID

Trigger Rules

- all_success
- all_failed
- all_done
- one_failed
- one_success
- none_failed
- none_failed_or_skipped – 1 parent succeded
- none_skipped
- dummy