

Apache Airflow

Extra Class: MLOps



Nguyen-Thuan Duong – TA
Kha-Vi Nguyen - sTA

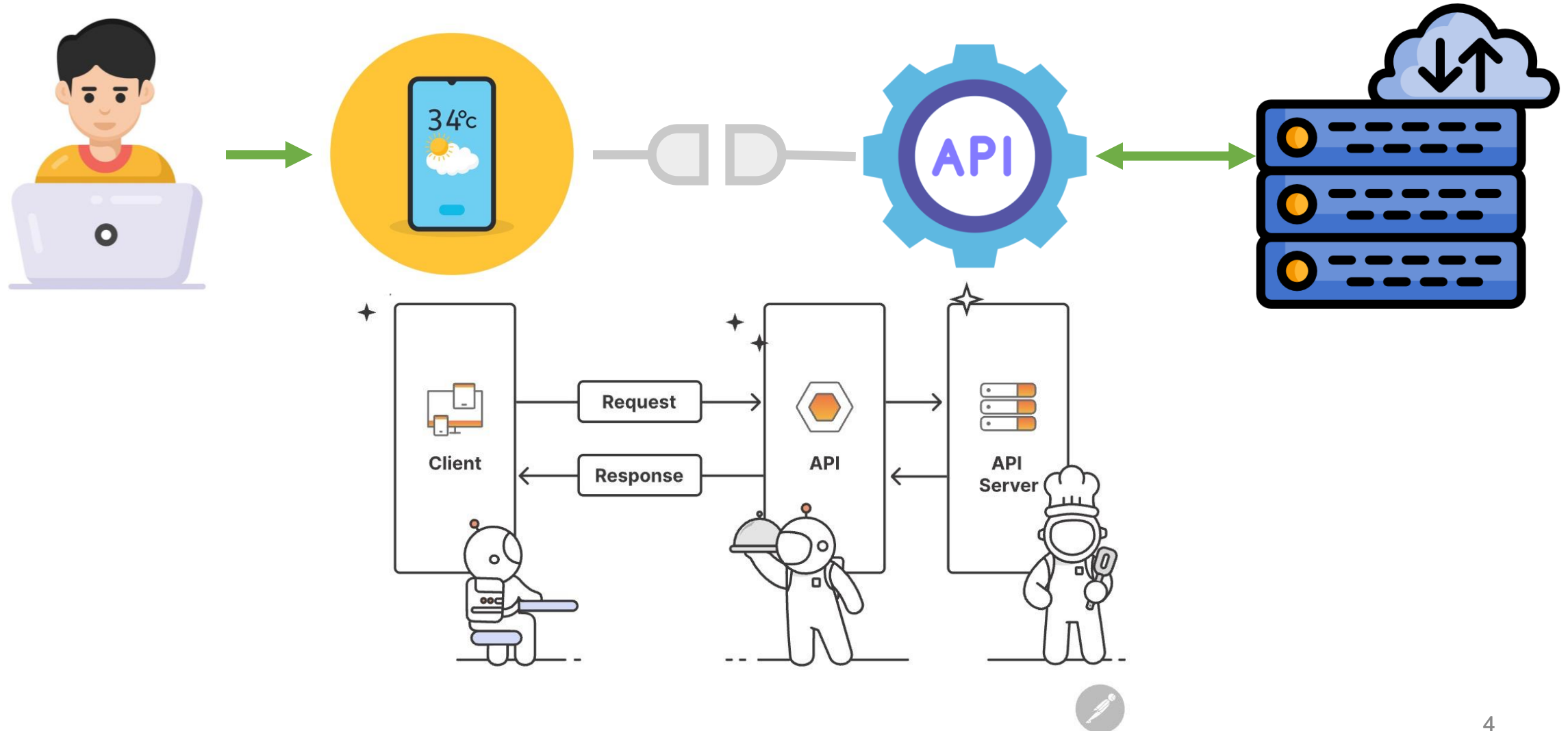
Outline

- Recap
- Introduction
- Airflow
- Pipeline
- Practice
- Question

Recap

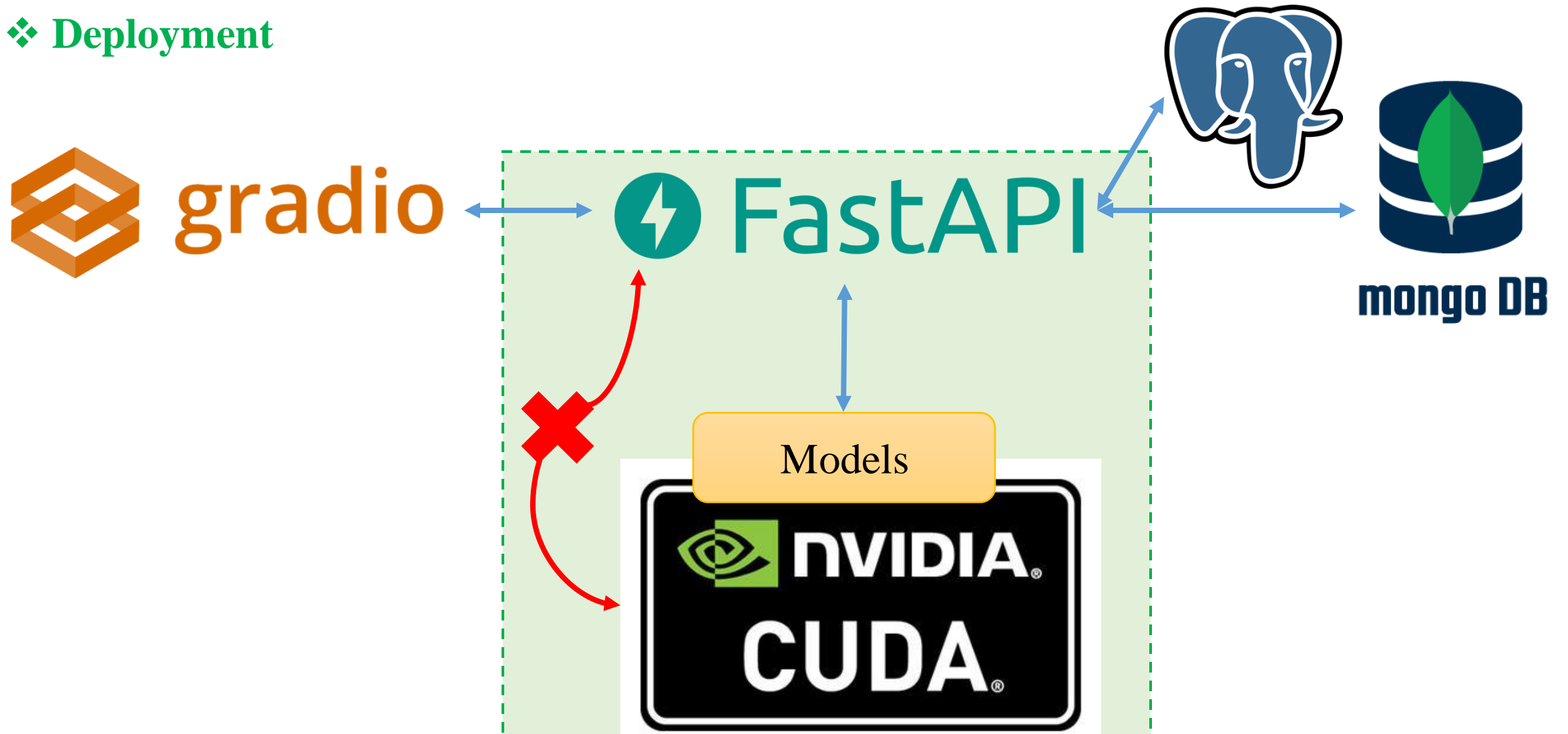
Recap

❖ Deployment



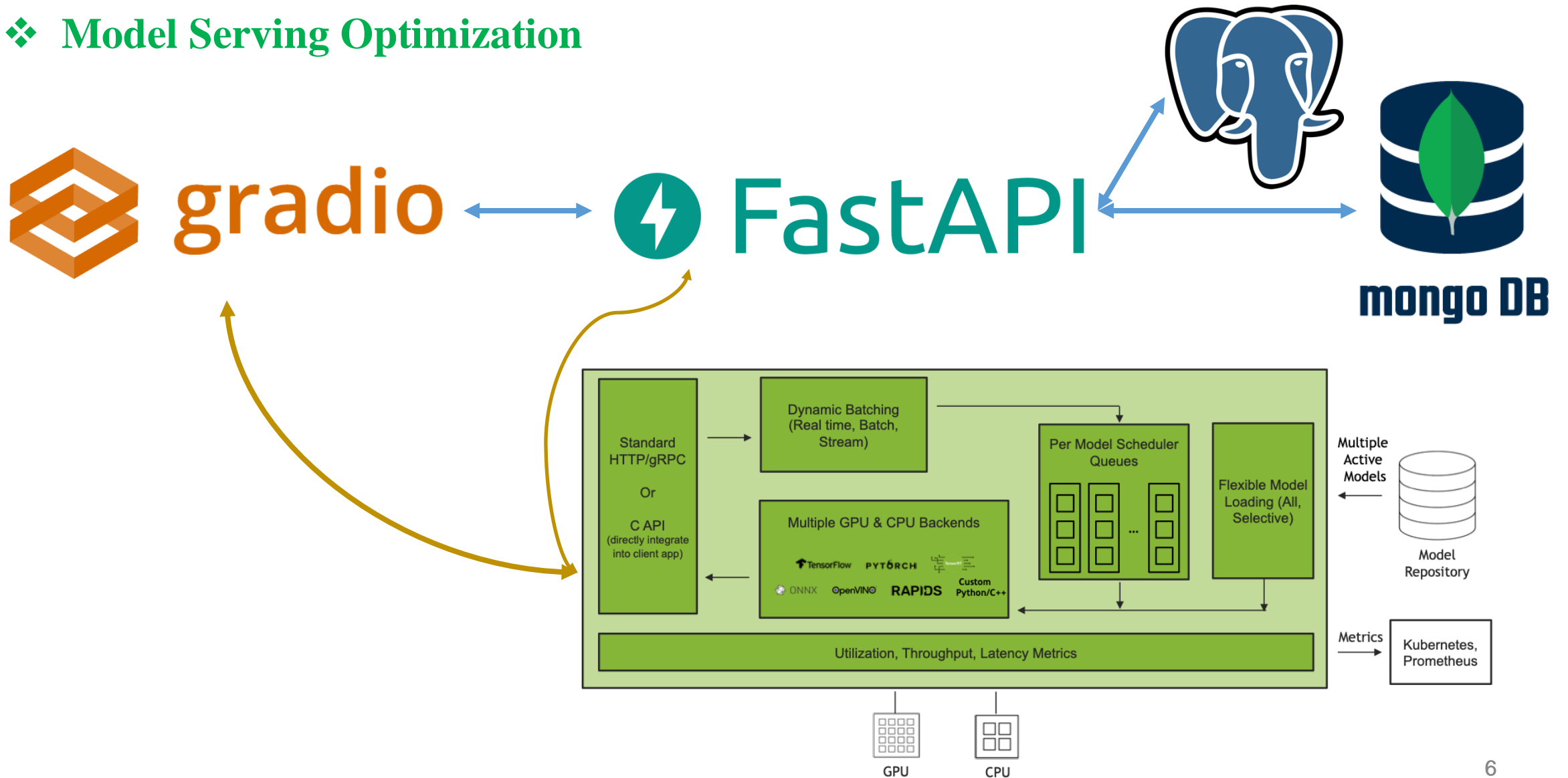
Recap

❖ Deployment



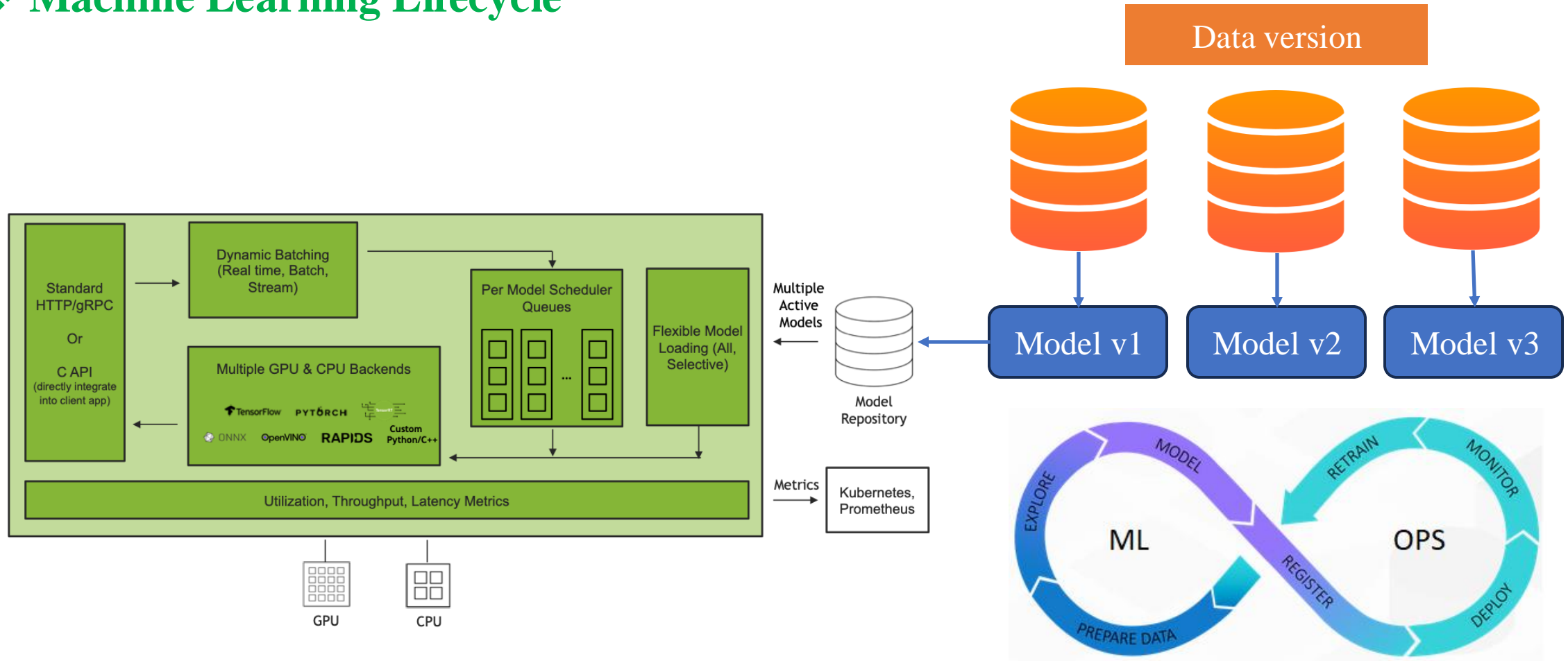
Recap

❖ Model Serving Optimization



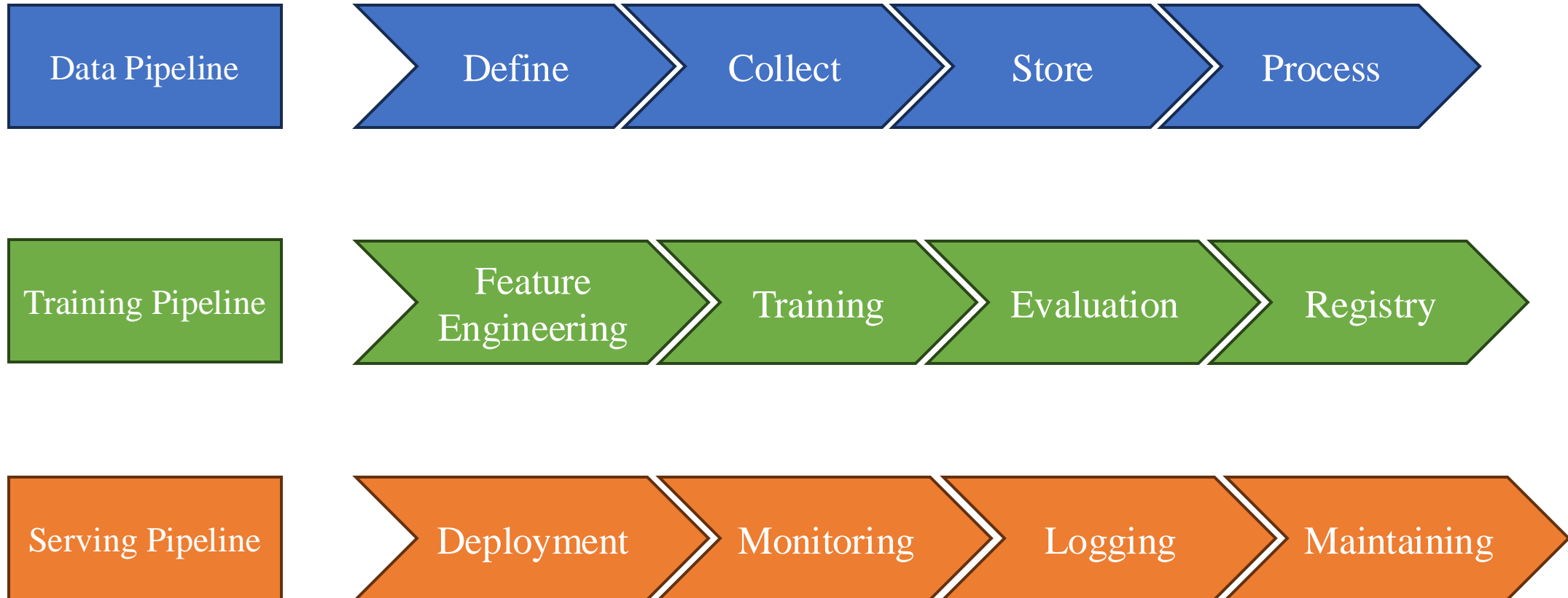
Recap

❖ Machine Learning Lifecycle



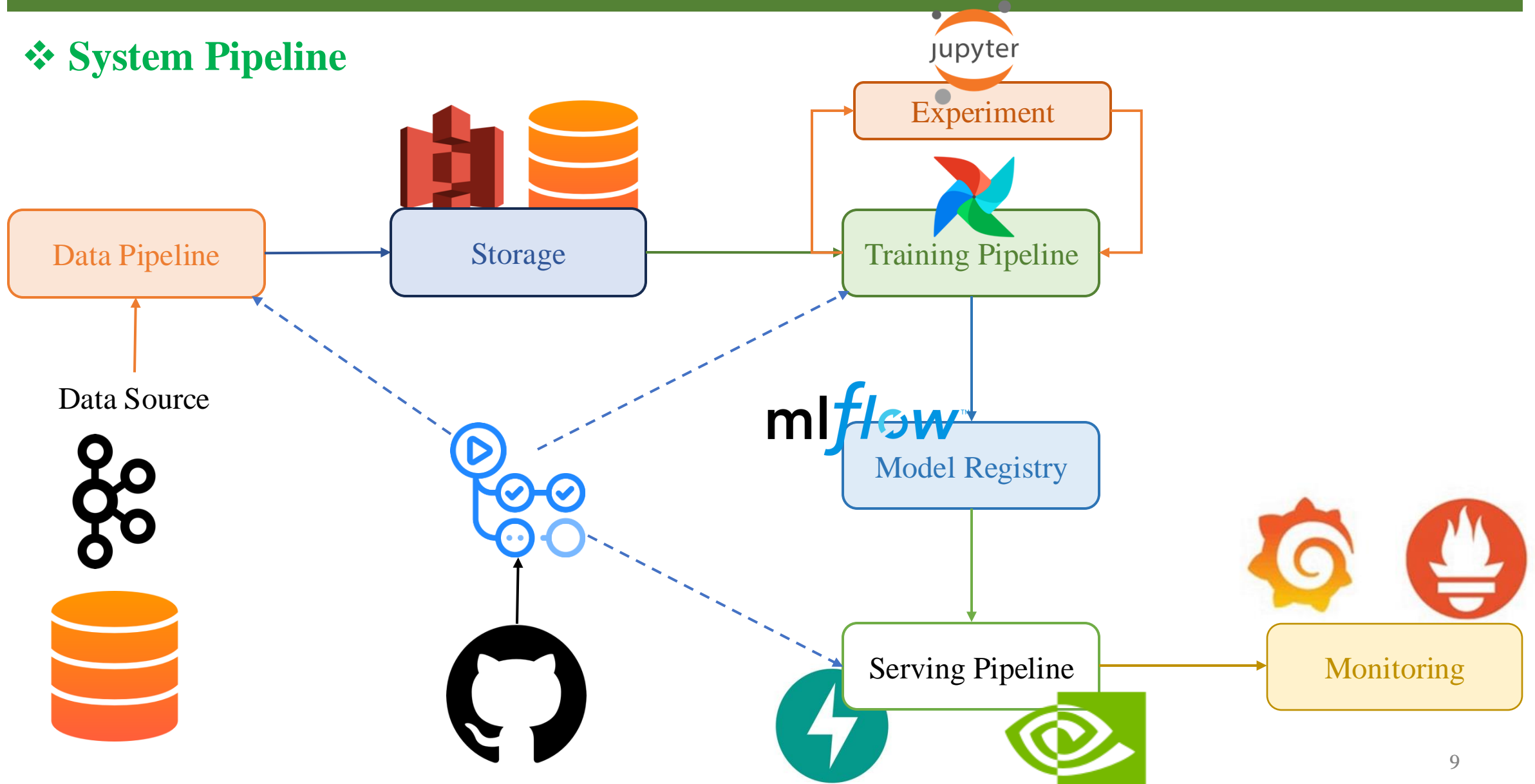
Recap

❖ System Pipeline



Recap

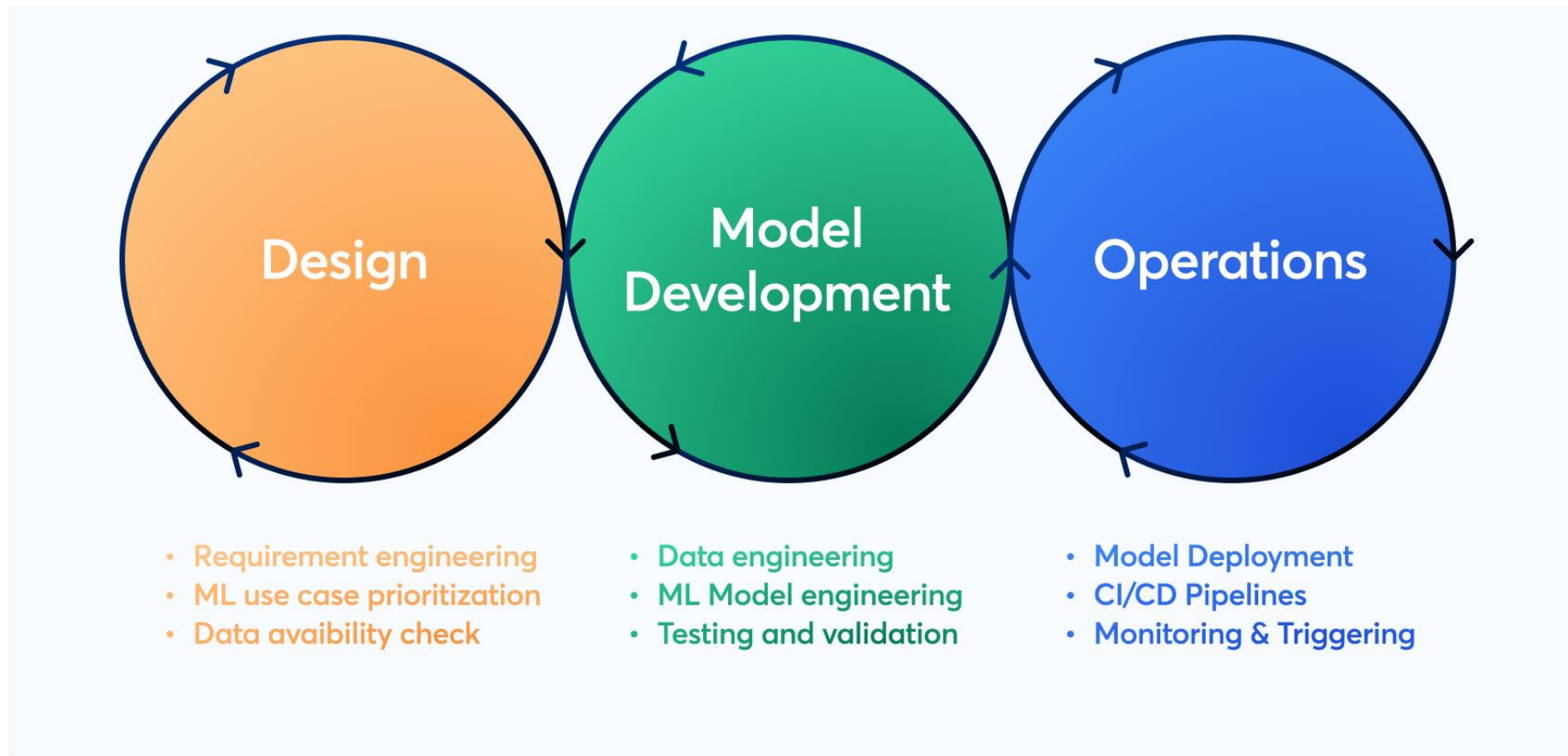
❖ System Pipeline



Introduction

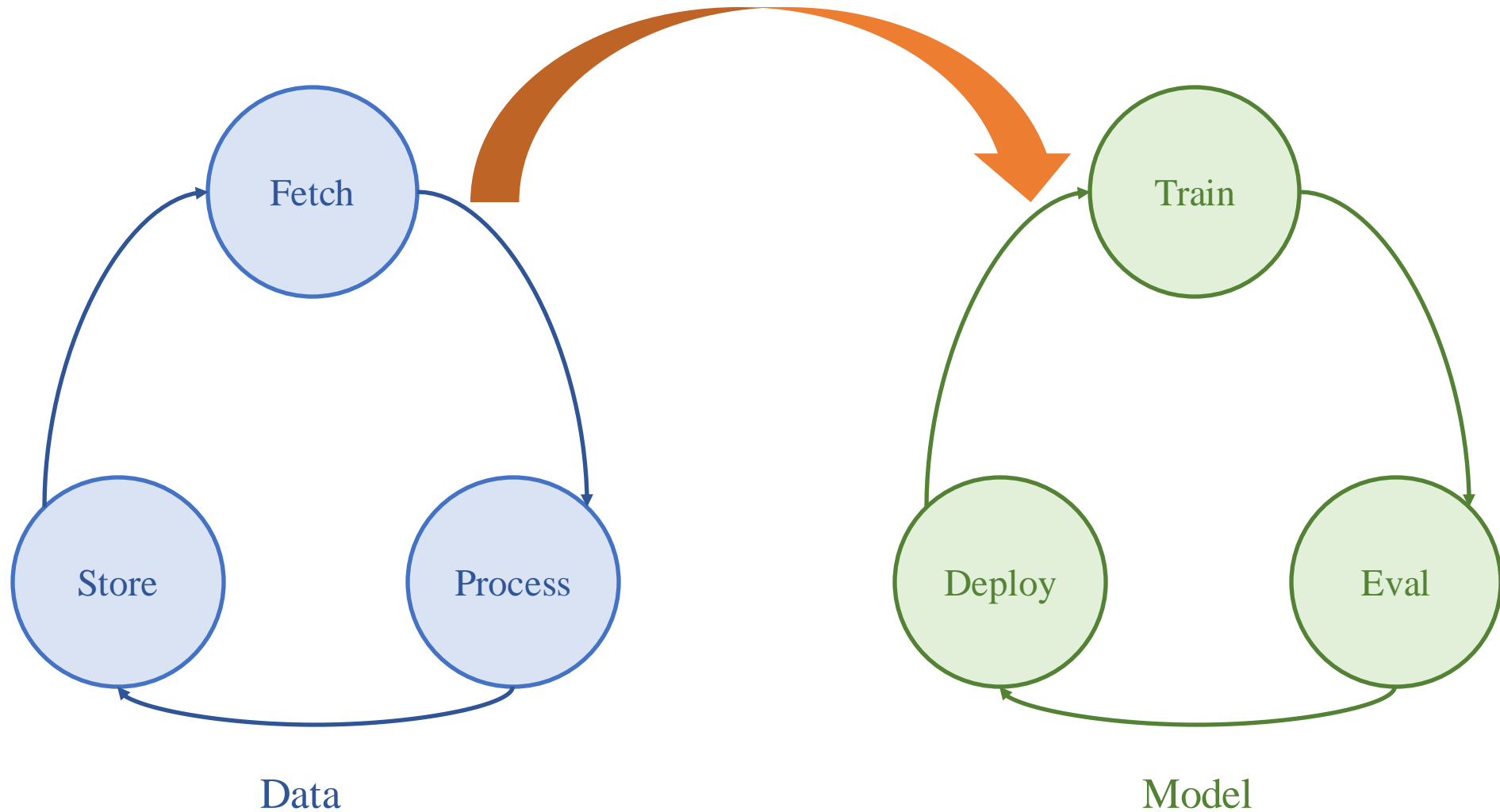
Introduction

❖ Getting Started



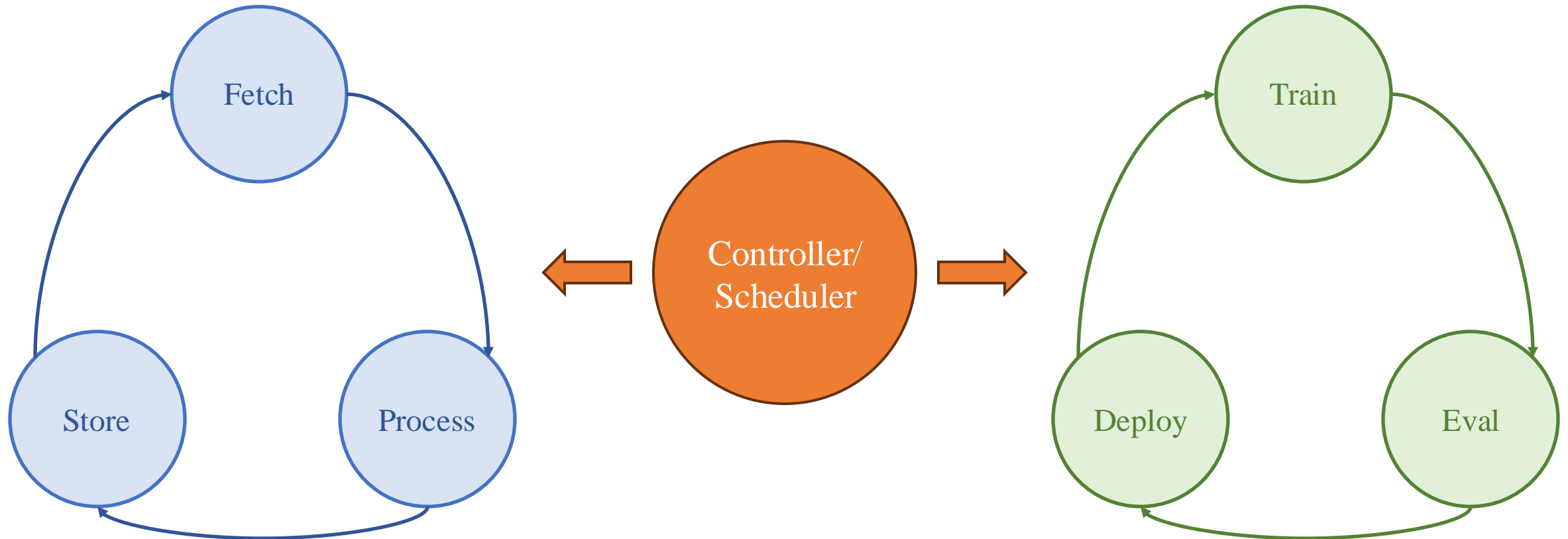
Introduction

❖ Cycle Processing



Introduction

❖ Controller/Scheduler



Airflow

❖ Getting Started

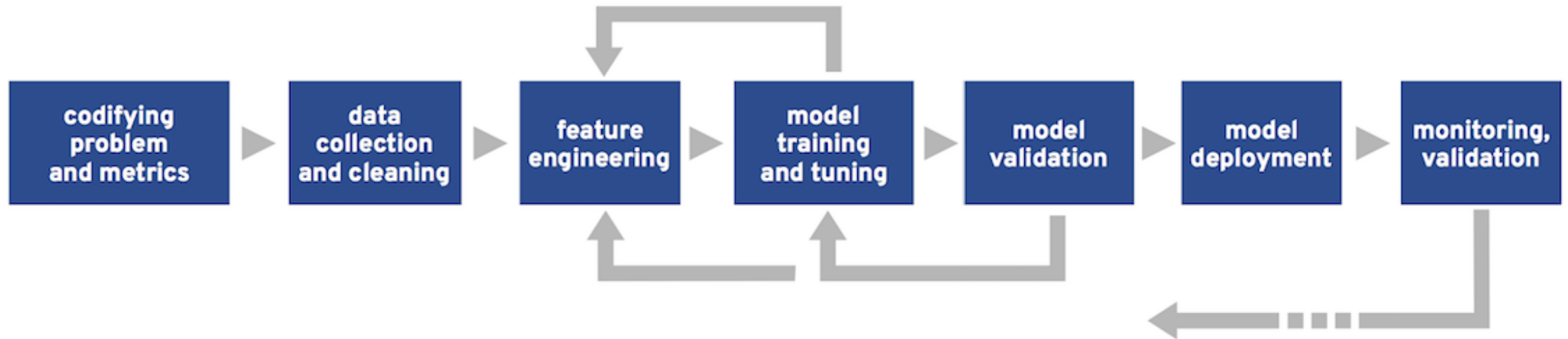
An open-source workflow management platform for data engineering pipelines

- It started at Airbnb in 2014
- Manage the company's complex workflows
- Workflows as code



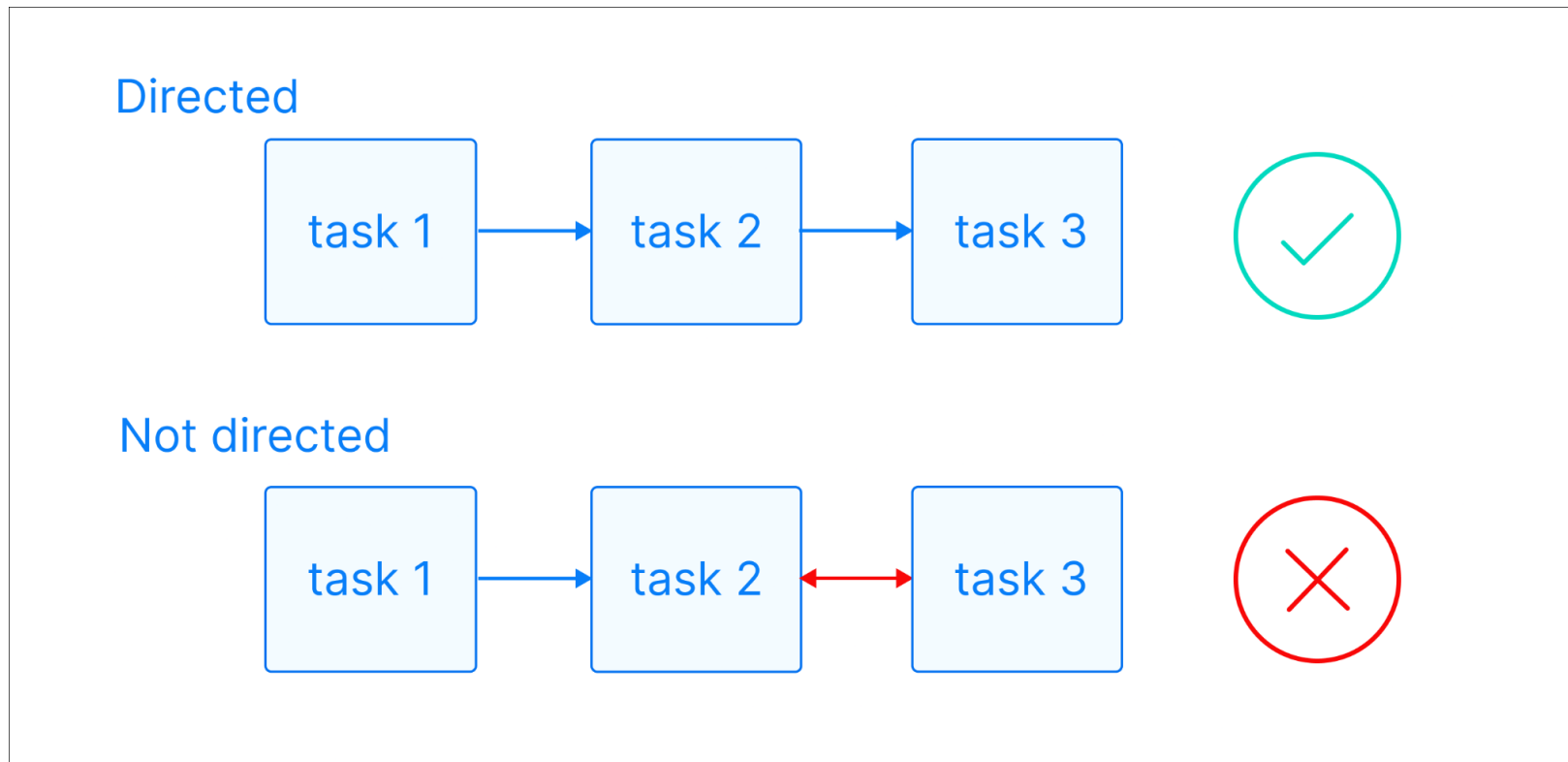
❖ Workflow

A workflow is a series of tasks that are arranged in a DAG. The DAG specifies the order in which the tasks should be executed.



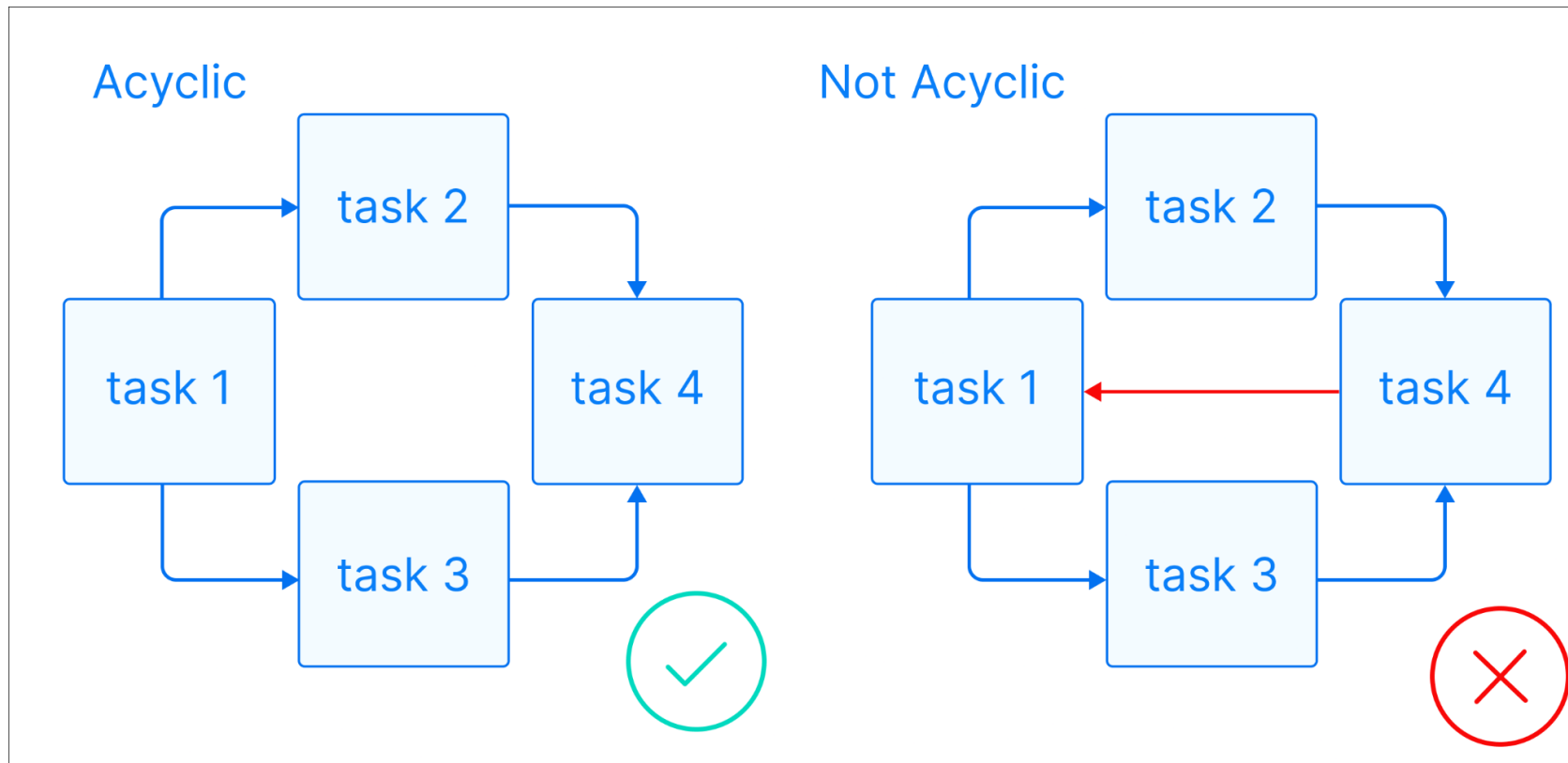
❖ Directed Acyclic Graph

A task can be either upstream, downstream, or parallel to another task (clear direction)



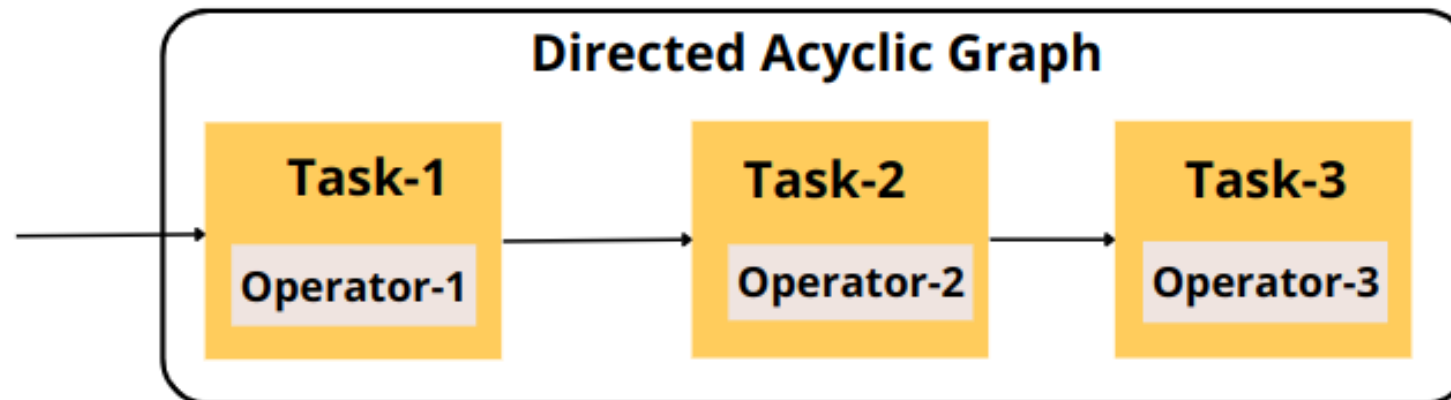
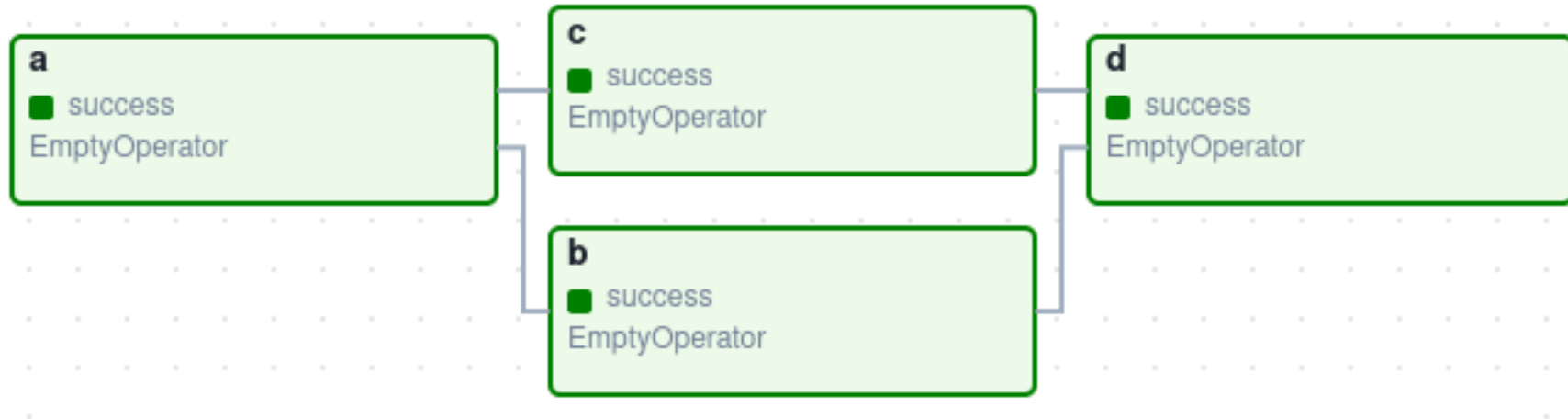
❖ Directed Acyclic Graph

A task cannot depend on itself, nor can it depend on a task that ultimately depends on it (NO circular dependencies)



❖ Directed Acyclic Graph

A DAG is a graph, which is a structure consisting of nodes and edges.

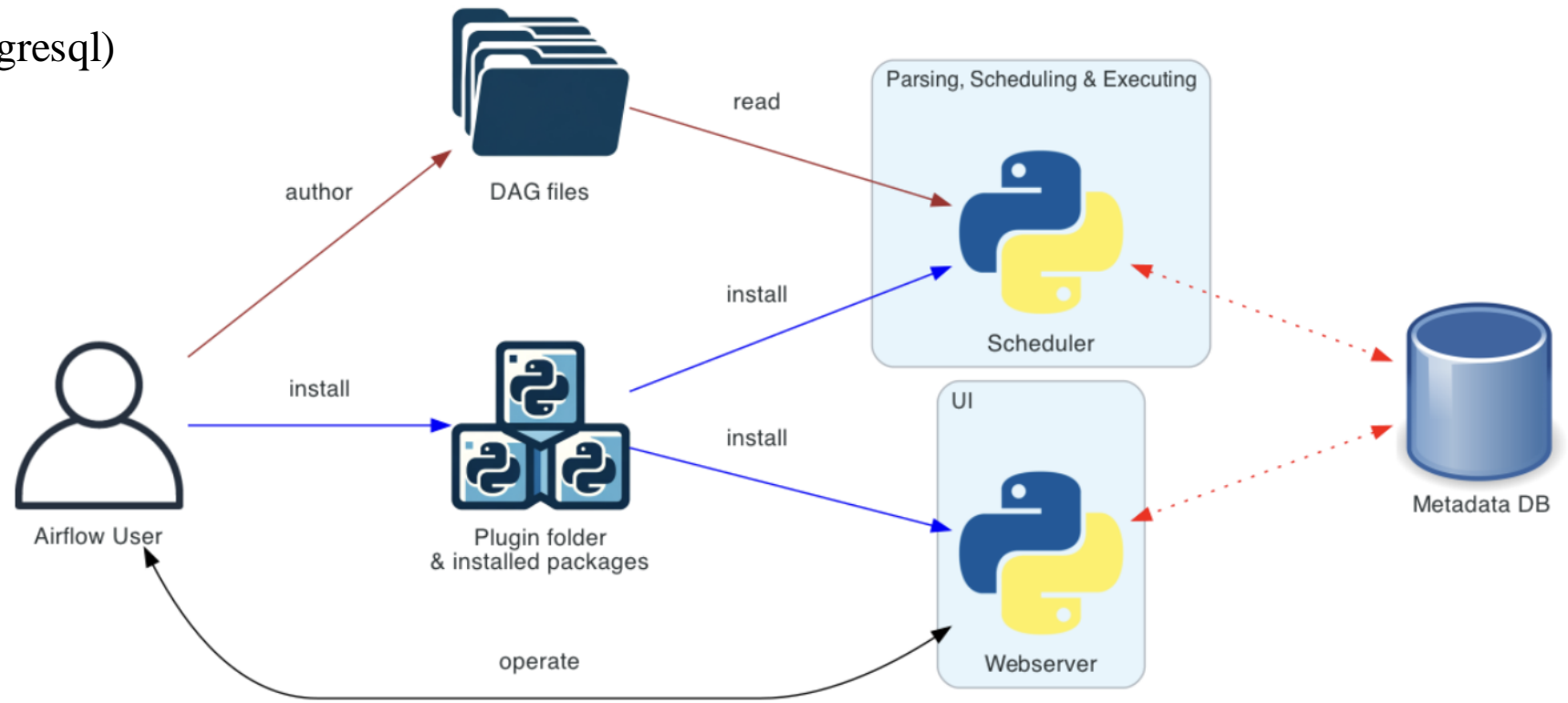


Airflow

❖ Airflow Architecture

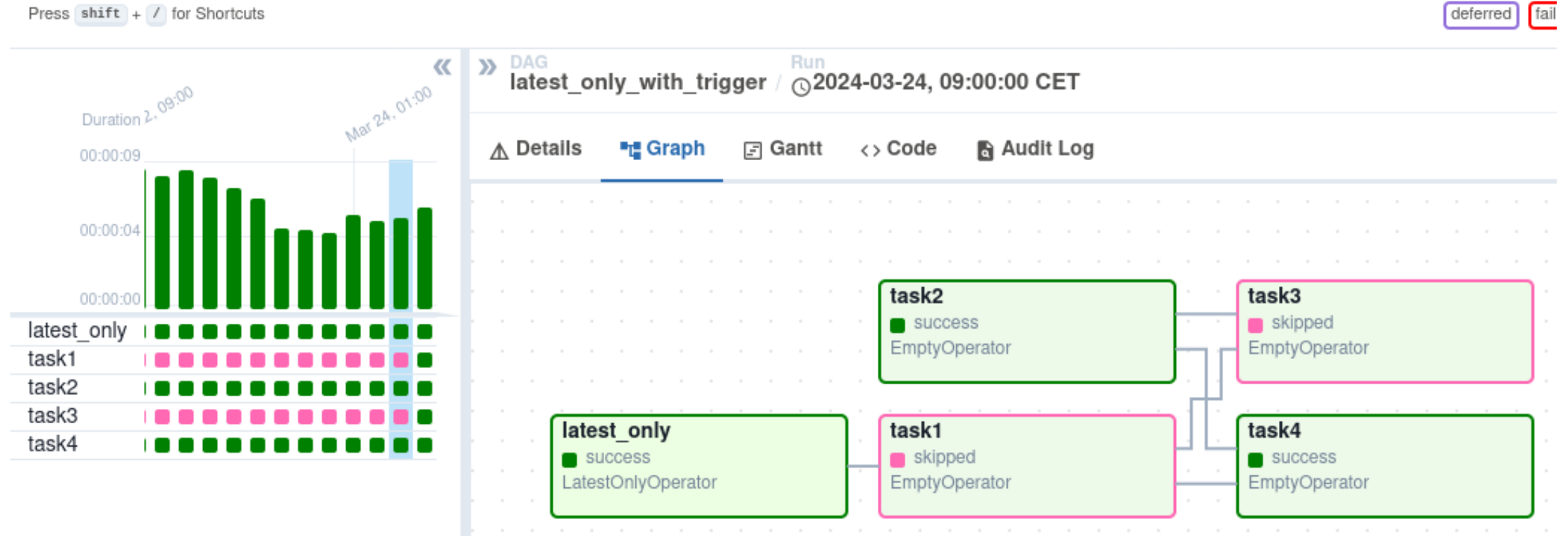
A minimal Airflow installation consists of the following components:

- Scheduler
- Webserver
- A metadata database (Postgresql)
- DAG files
- Message bus (Redis)



❖ Core Concepts

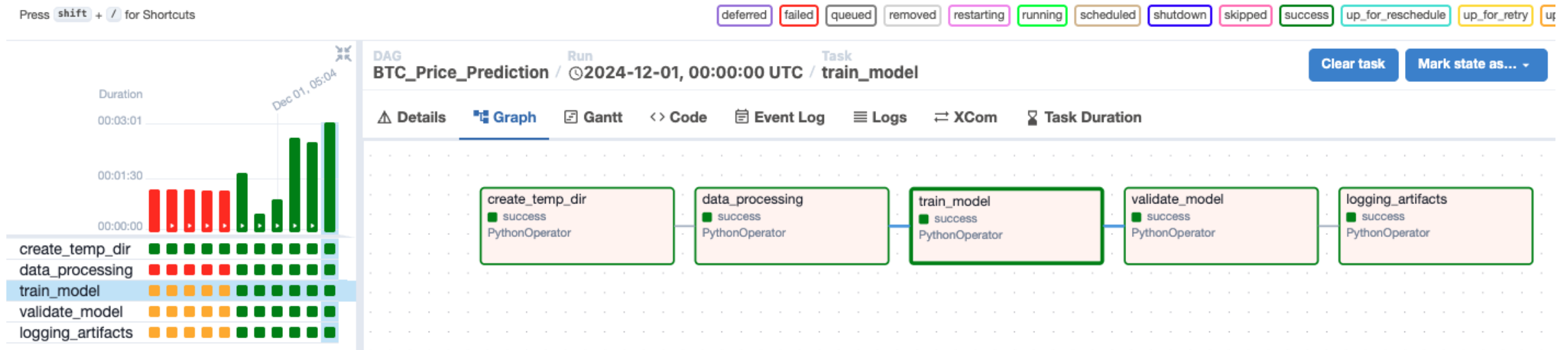
DAGs: Organize tasks with dependencies and relationships to say how they should run.



Airflow

❖ Core Concepts

DAGs: Organize tasks with dependencies and relationships to say how they should run.



❖ Core Concepts

Task: Arranged into DAGs, and then have upstream and downstream dependencies set between them in order to express the order they should run in.

```
1 create_temp_dir_task >> data_processing_task >> train_model_task >> validate_model_task >> logging_artifacts_task
2
```

Individual task dependencies

```
1 create_temp_dir_task \
2 >> [fetch_BTC_data_task, data_processing_task] \
3 >> [fetch_ETH_data_task, data_processing_task] \
4 >> [fetch_Gold_data_task, data_processing_task] \
5 >> train_model_task >> validate_model_task >> logging_artifacts_task
```

Group task dependencies

❖ Core Concepts

Operator: A template for a predefined Task
Eg. BashOperator, PythonOperator, EmailOperator, ...

BashOperator

Executes a bash command

PythonOperator

Calls an arbitrary Python function

DockerOperator

Execute a command inside a docker container

MySqlOperator

Executes sql code in a specific MySQL database

❖ Core Concepts

Operator: A template for a predefined Task
Eg. BashOperator, PythonOperator, EmailOperator, ...

EmailOperator

Sends an email

HTTPOperator

Calls an endpoint on an HTTP system to execute an action

S3FileTransformOperator

Copies data from a source S3 location to a temporary location on the local

SlackAPIOperator

Send a file to a Slack channel

❖ Core Concepts

```
1 default_args = {
2     'owner': 'airflow',
3     'depends_on_past': False,
4     'trigger_rule': 'all_success',
5     'start_date': days_ago(1),
6     'email_on_failure': False,
7     'email_on_retry': False,
8     'retries': 1,
9     'retry_delay': timedelta(minutes=1),
10 }
11
```

```
1 with DAG(
2     'BTC_Price_Prediction',
3     default_args=default_args,
4     description='A simple ML pipeline demonstration',
5     schedule_interval=timedelta(days=1),
6 ) as dag:
7
8     data_processing_task = PythonOperator(
9         task_id='data_processing',
10        python_callable=data_processing,
11        dag=dag,
12    )
13
14    train_model_task = PythonOperator(
15        task_id='train_model',
16        python_callable=train_model,
17        dag=dag,
18    )
19
20    validate_model_task = PythonOperator(
21        task_id='validate_model',
22        python_callable=validate_model,
23        dag=dag,
24    )
```

❖ Pass data between tasks

Sharing data between tasks is a very common use case in Airflow.

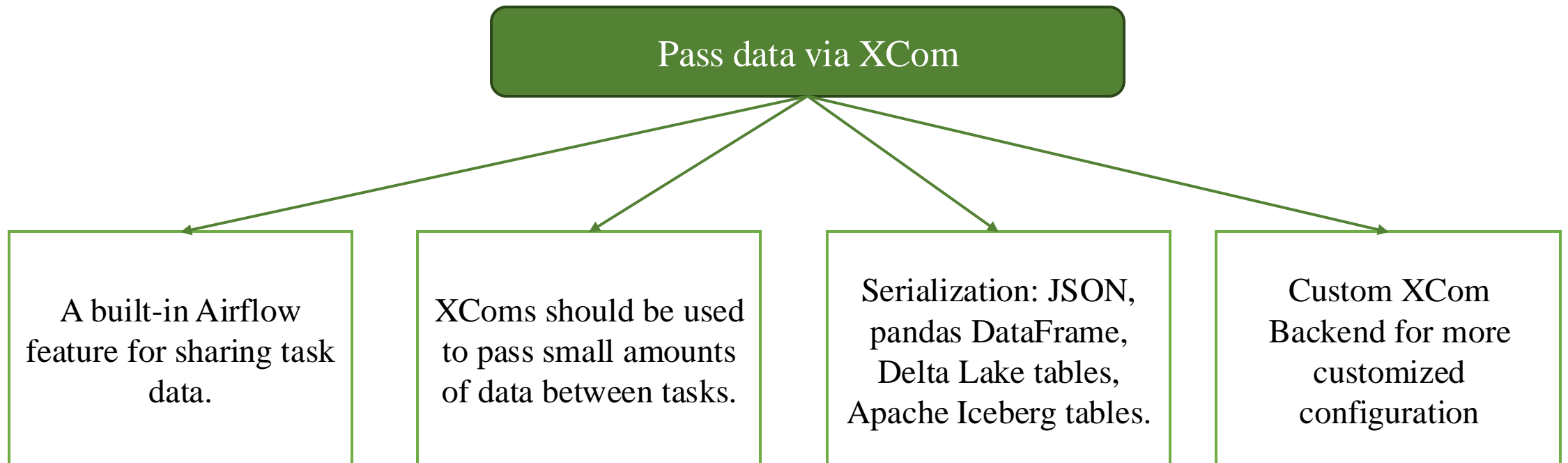
Pass data via XCom



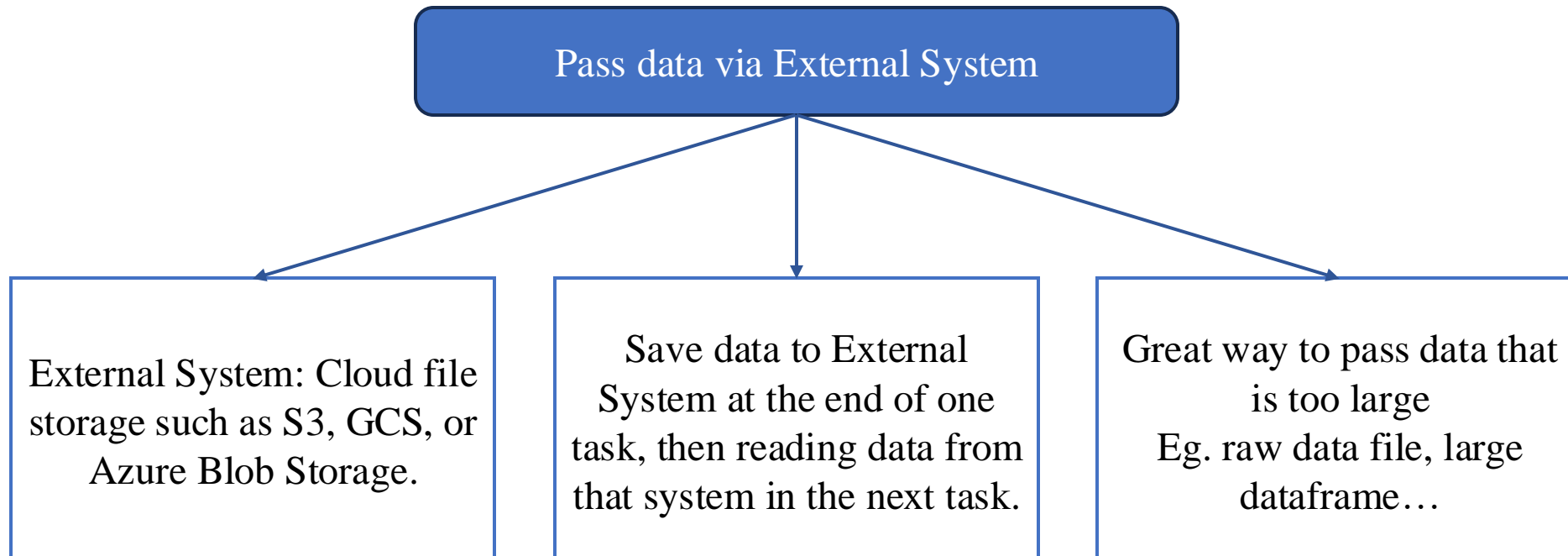
Pass data via External System



❖ Pass data between tasks



❖ Pass data between tasks



Pipeline

Pipeline

❖ Version Control

- Store Airflow code and configuration in Git
- Set up **dev**, **UAT** (User Acceptance Testing), and **production** branches in Git and mapping them to associated Airflow environments
- **Test** and **lint** for all Airflow code before deployment



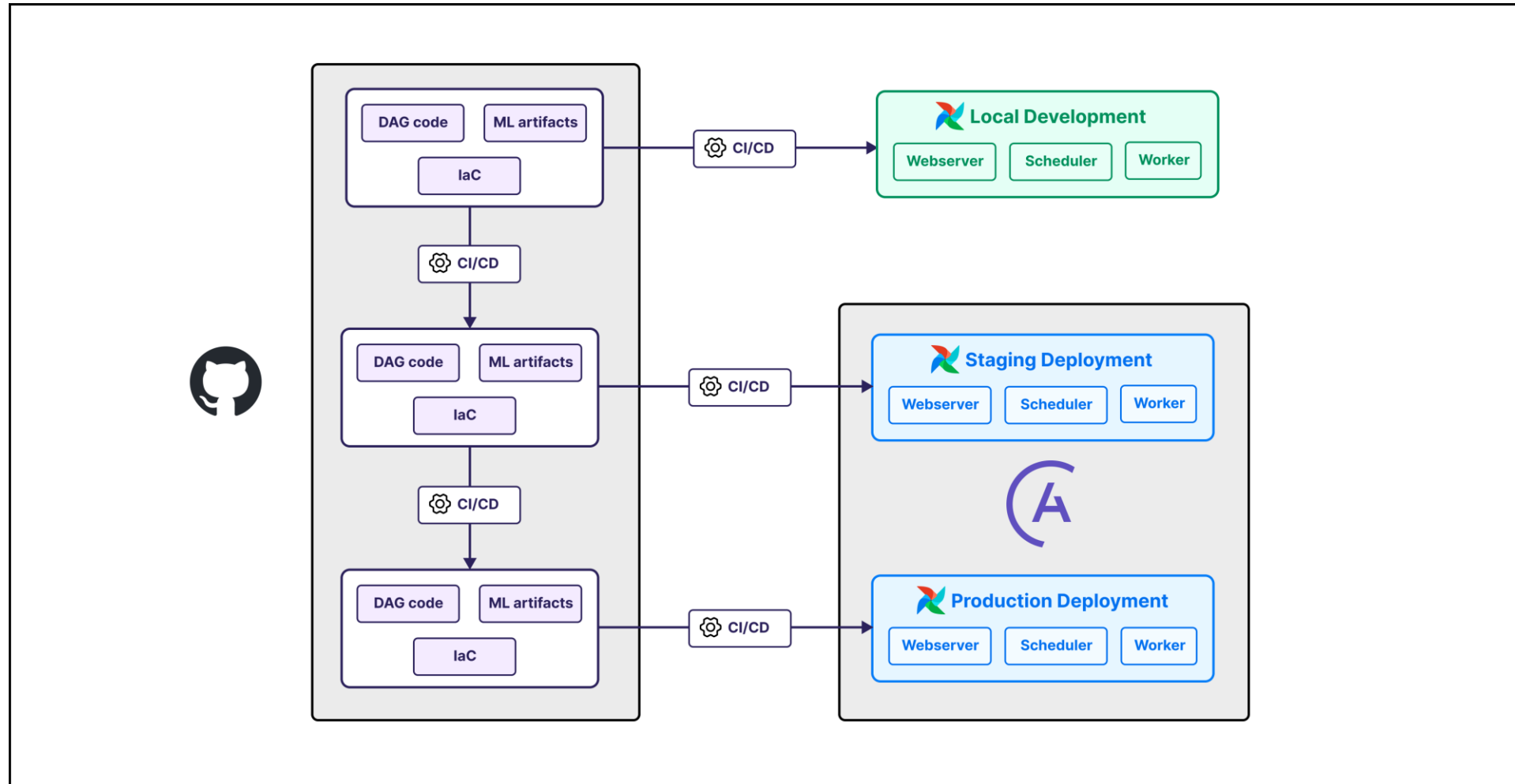
git



Pipeline

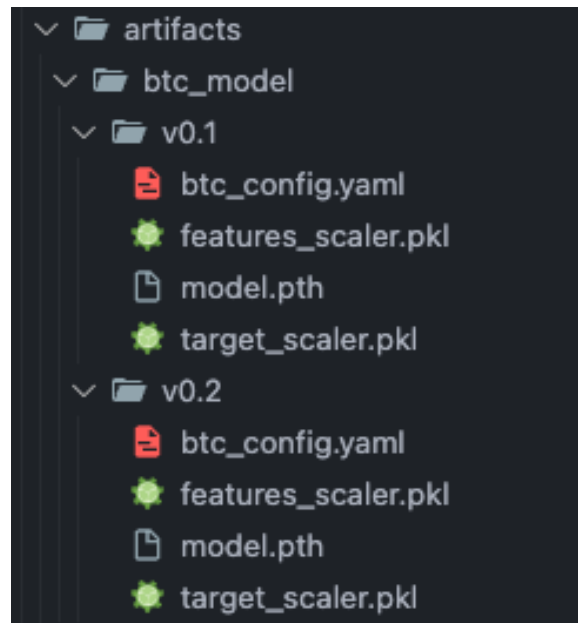
❖ Infrastructure as code (IaC)

Use the same CI/CD process by defining IaC



Pipeline

❖ ML models registry (next)



```
DAG      Run      Task
BTC_Price_Prediction / 2024-12-01, 00:00:00 UTC / train_model

Details  Graph  Gantt  Code  Event Log  Logs  XCom  Task Duration

All Levels  All File Sources

fec79d124a9f
*** Found local files:
*** * /opt/airflow/logs/dag_id=BTC_Price_Prediction/run_id=scheduled__2024-12-01T00:00:00+00:00/task_id=train_model/attempt=1.log
[2024-12-02, 00:00:07 UTC] {local_task_job_runner.py:123} > Pre task execution logs
[2024-12-02, 00:00:17 UTC] {logging_mixin.py:190} INFO - Epoch 1, train loss: 0.091453, val loss: 0.211709
[2024-12-02, 00:00:24 UTC] {logging_mixin.py:190} INFO - Epoch 2, train loss: 0.064273, val loss: 0.150071
[2024-12-02, 00:00:30 UTC] {logging_mixin.py:190} INFO - Epoch 3, train loss: 0.056405, val loss: 0.122786
[2024-12-02, 00:00:34 UTC] {logging_mixin.py:190} INFO - Epoch 4, train loss: 0.052202, val loss: 0.107706
[2024-12-02, 00:00:40 UTC] {logging_mixin.py:190} INFO - Epoch 5, train loss: 0.048617, val loss: 0.097072
[2024-12-02, 00:00:48 UTC] {logging_mixin.py:190} INFO - Epoch 6, train loss: 0.045361, val loss: 0.087434
[2024-12-02, 00:00:52 UTC] {logging_mixin.py:190} INFO - Epoch 7, train loss: 0.042160, val loss: 0.080800
[2024-12-02, 00:00:56 UTC] {logging_mixin.py:190} INFO - Epoch 8, train loss: 0.038960, val loss: 0.073540
[2024-12-02, 00:01:00 UTC] {logging_mixin.py:190} INFO - Epoch 9, train loss: 0.035670, val loss: 0.066156
[2024-12-02, 00:01:05 UTC] {logging_mixin.py:190} INFO - Epoch 10, train loss: 0.032349, val loss: 0.059031
[2024-12-02, 00:01:12 UTC] {logging_mixin.py:190} INFO - Epoch 11, train loss: 0.028990, val loss: 0.050794
[2024-12-02, 00:01:19 UTC] {logging_mixin.py:190} INFO - Epoch 12, train loss: 0.025542, val loss: 0.044370
[2024-12-02, 00:01:25 UTC] {logging_mixin.py:190} INFO - Epoch 13, train loss: 0.022105, val loss: 0.036882
[2024-12-02, 00:01:30 UTC] {logging_mixin.py:190} INFO - Epoch 14, train loss: 0.018719, val loss: 0.030737
[2024-12-02, 00:01:36 UTC] {logging_mixin.py:190} INFO - Epoch 15, train loss: 0.015475, val loss: 0.023750
[2024-12-02, 00:01:41 UTC] {logging_mixin.py:190} INFO - Epoch 16, train loss: 0.012393, val loss: 0.018332
[2024-12-02, 00:01:46 UTC] {logging_mixin.py:190} INFO - Epoch 17, train loss: 0.009646, val loss: 0.013718
[2024-12-02, 00:01:53 UTC] {logging_mixin.py:190} INFO - Epoch 18, train loss: 0.007274, val loss: 0.009895
[2024-12-02, 00:01:57 UTC] {logging_mixin.py:190} INFO - Epoch 19, train loss: 0.005331, val loss: 0.006432
[2024-12-02, 00:02:01 UTC] {logging_mixin.py:190} INFO - Epoch 20, train loss: 0.003787, val loss: 0.003833
[2024-12-02, 00:02:05 UTC] {logging_mixin.py:190} INFO - Epoch 21, train loss: 0.002645, val loss: 0.002360
[2024-12-02, 00:02:11 UTC] {logging_mixin.py:190} INFO - Epoch 22, train loss: 0.001852, val loss: 0.001307
[2024-12-02, 00:02:19 UTC] {logging_mixin.py:190} INFO - Epoch 23, train loss: 0.001317, val loss: 0.000698
[2024-12-02, 00:02:27 UTC] {logging_mixin.py:190} INFO - Epoch 24, train loss: 0.000994, val loss: 0.000353
[2024-12-02, 00:02:31 UTC] {logging_mixin.py:190} INFO - Epoch 25, train loss: 0.000786, val loss: 0.000224
[2024-12-02, 00:02:37 UTC] {logging_mixin.py:190} INFO - Epoch 26, train loss: 0.000667, val loss: 0.000165
[2024-12-02, 00:02:45 UTC] {logging_mixin.py:190} INFO - Epoch 27, train loss: 0.000598, val loss: 0.000157
[2024-12-02, 00:02:48 UTC] {logging_mixin.py:190} INFO - Epoch 28, train loss: 0.000558, val loss: 0.000166
[2024-12-02, 00:02:51 UTC] {logging_mixin.py:190} INFO - Epoch 29, train loss: 0.000532, val loss: 0.000182
[2024-12-02, 00:02:57 UTC] {logging_mixin.py:190} INFO - Epoch 30, train loss: 0.000517, val loss: 0.000199
[2024-12-02, 00:02:57 UTC] {logging_mixin.py:190} INFO - Training complete
[2024-12-02, 00:02:57 UTC] {python.py:240} INFO - Done. Returned value was: None
[2024-12-02, 00:02:57 UTC] {taskinstance.py:340} > Post task execution logs
```



Pipeline

❖ Mlflow (next)



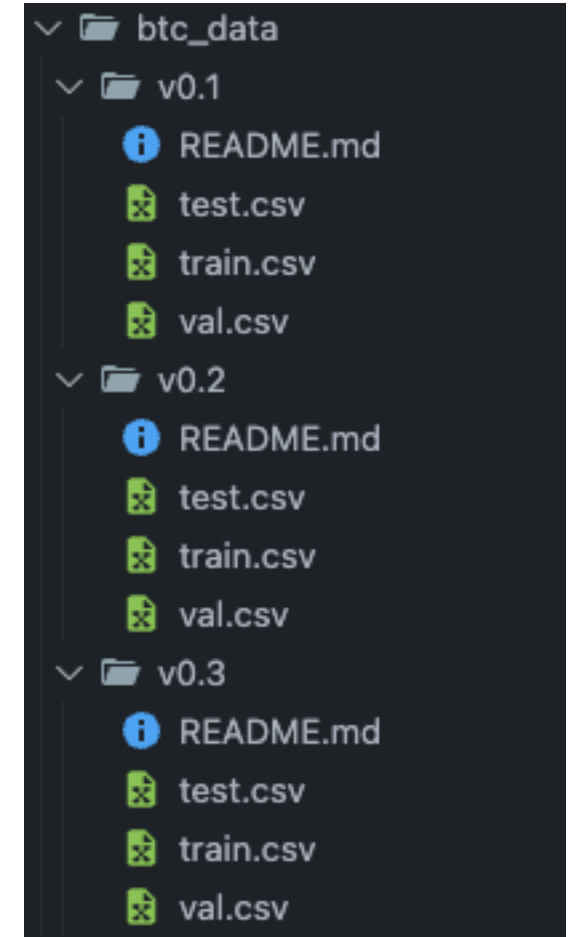
Practice

Practice

❖ Dataset

BTC price data is downloaded from investing.com website. The dataset is collected from 1/2016 to 10/2024 and split to 3 versions: v0.1, v0.2 and v0.3.

- v0.1: 1/2016 to 12/2023
- v0.2: 1/2016 to 2/2024
- v0.3: 1/2016 to 10/2024



Practice

❖ Model

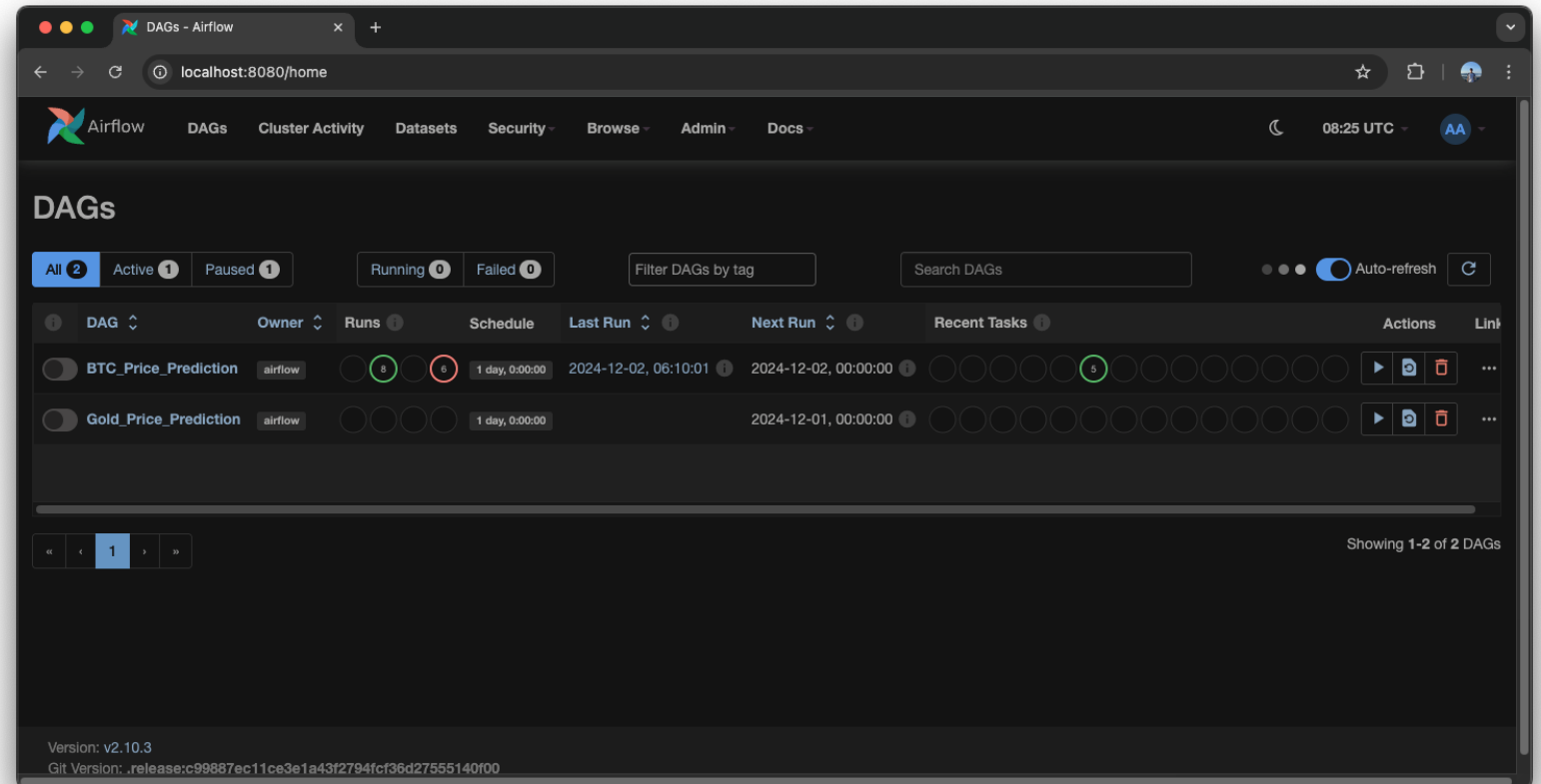
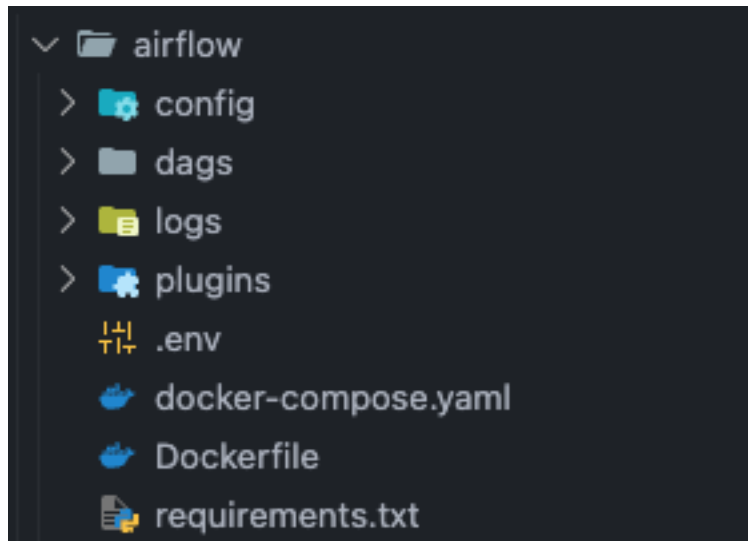
Model is used for training time-series data is RNN. The model contains 2 RNN layer and 2 fully-connected layers to predict the price of BTC in the future.

```
1 class RNN_Model(nn.Module):
2     def __init__(self, input_size, hidden_size, output_size, num_layers):
3         super().__init__()
4         self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True, bidirectional=True)
5         self.fc1 = nn.Linear(hidden_size * 2, hidden_size)
6         self.fc2 = nn.Linear(hidden_size, output_size)
7         self.relu = nn.ReLU()
8
9     def forward(self, x):
10        out, _ = self.rnn(x)
11        x = self.fc1(out[:, -1, :])
12        x = self.relu(x)
13        x = self.fc2(x)
14        return x
```

Practice

❖ Airflow server

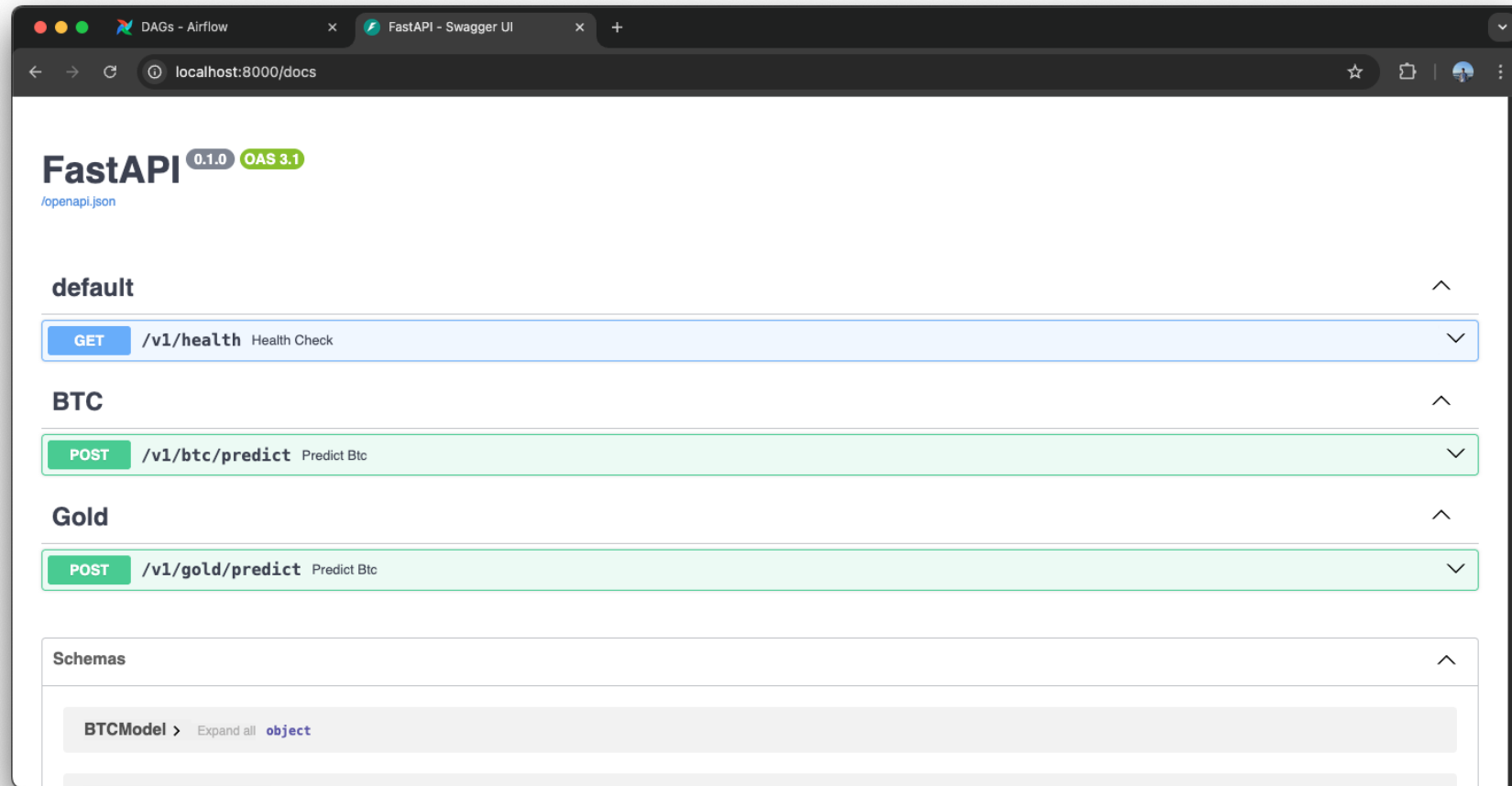
Use Docker and Docker Compose to start Airflow server and related components.



Default user and password of airflow account is: airflow

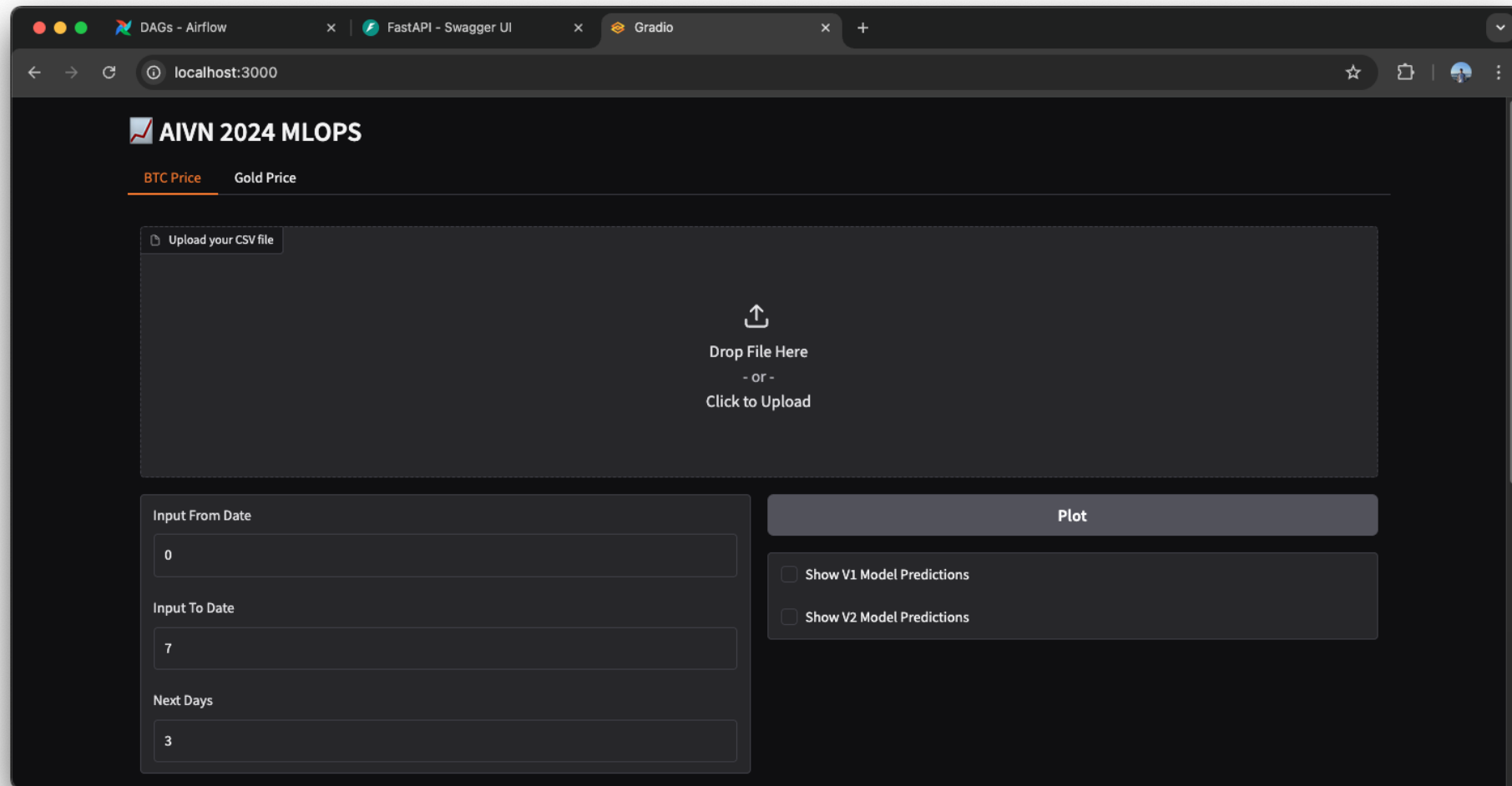
❖ Backend server

FastAPI is used for backend server



❖ Frontend UI

Gradio is used for frontend interface



Question

