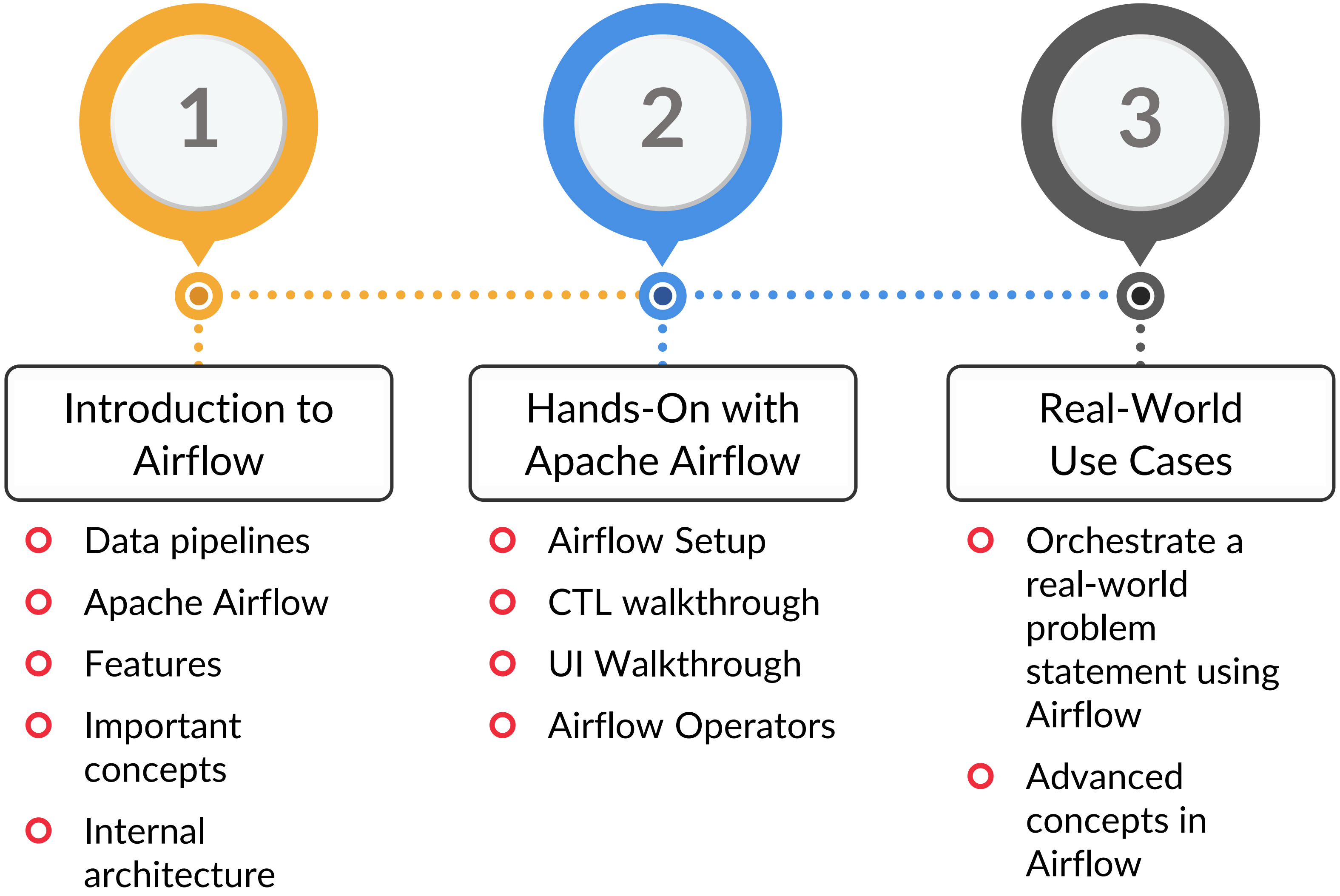
Introduction to Apache Airflow

Session 1

HEALTHCARE EVOLUTION



1

Introduction to Airflow

- Data pipelines
- Apache Airflow
- Features
- Important concepts
- Internal architecture

2

Hands-On with Apache Airflow

- Airflow Setup
- CTL walkthrough
- UI Walkthrough
- Airflow Operators

3

Real-World Use Cases

- Orchestrate a real-world problem statement using Airflow
- Advanced concepts in Airflow

SESSION OVERVIEW

01 Data pipelines in large scale distributed environments

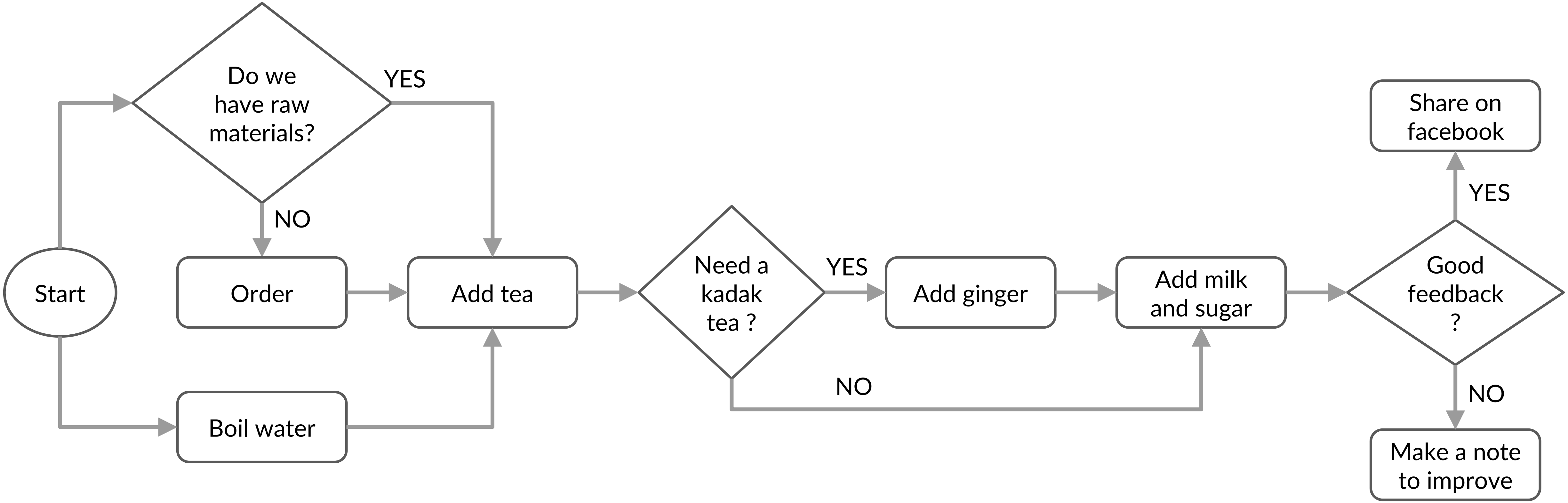
02 Solutions to data pipeline orchestration

03 Apache Airflow and its features

04 The architecture of Airflow

FLOW CHART

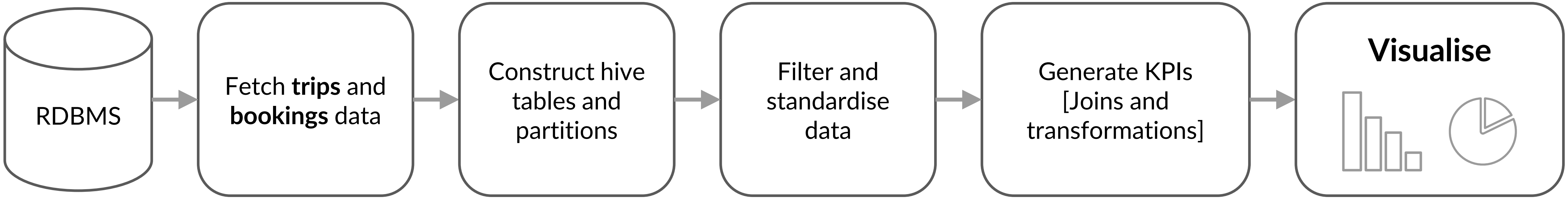
- Visual representation of the flow of actions taken to reach to an output or conclusion
- Easy to understand the logical steps happening in the activity



DATA PIPELINE

- Databases good for Online Transactional Processing (OLTP) are not best suited for Online Analytical Processing (OLAP)
- First step in cost effective and scalable solution for data processing is to bring data from OLTP databases to OLAP systems
- Data Pipeline
  - Process of moving data from one system to the other and possibly transforming and validating it in the process

UBER USE CASE



RDBMS

Fetch **trips** and **bookings** data

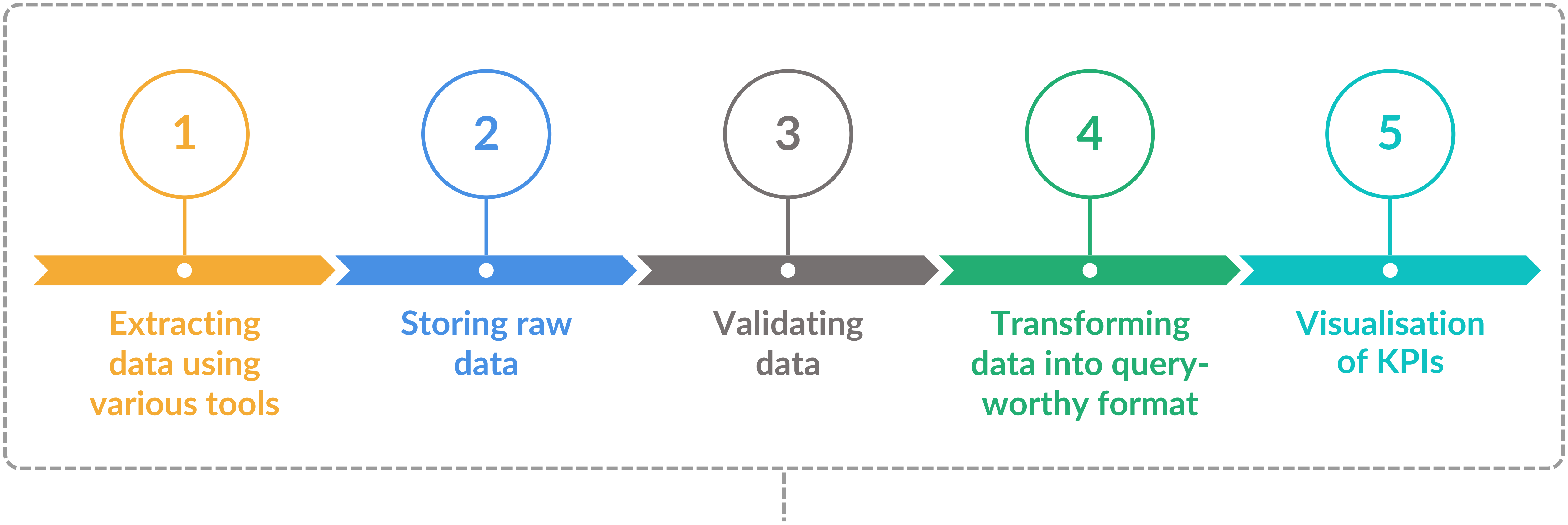
Construct hive tables and partitions

Filter and standardise data

Generate KPIs [Joins and transformations]

Visualise

GENERAL STRUCTURE OF DATA PIPELINES



1. Extracting data using various tools
2. Storing raw data
3. Validating data
4. Transforming data into query-worthy format
5. Visualisation of KPIs

Orchestration of the aforementioned process

IMPORTANCE OF DATA PIPELINE AUTOMATION

- Data pipeline acts as the central nervous system for the **data-driven-decision-making.**
- **Notification** when processed data is available or if something fails
- **Reduced manual effort.** More **focus on business logic.**
- Keep **track of the performance** of different steps in the pipeline. This helps in **identification and resolution of bottlenecks**
- Automating data pipelines allows the company to collect, process and economically use data in **real-time**

POSSIBLE SOLUTIONS TO DATA PIPELINE AUTOMATION

- Manual Orchestration
  - No need to learn any data-pipeline orchestration tool
  - Impractical in a large scale industry environment
  - Against the principle of Do-Not-Repeat

POSSIBLE SOLUTIONS

- Cron Jobs
  - Very simple, comes pre-installed in Linux machines, easy to learn
  - Cumbersome when pipelines are longer
  - No intelligent handling of failed tasks or task retries
  - Lacks in-built alerting and monitoring
  - Not extensible
  - No visualization
  - Difficult to get info about the components of the pipeline or current status or previous runs

POSSIBLE SOLUTIONS

- Apache Oozie
  - One of the early and robust data orchestration tools with
    - scheduling capabilities
    - UI
    - parameterization (using variables)
    - supporting diverse operations (like Email, MapReduce, Hive etc.)
  - Steep learning curve
  - Uses XML for building pipelines, difficult to create dynamic DAGs
  - Difficult to build pipelines with a mediocre GUI
  - Community support is not great

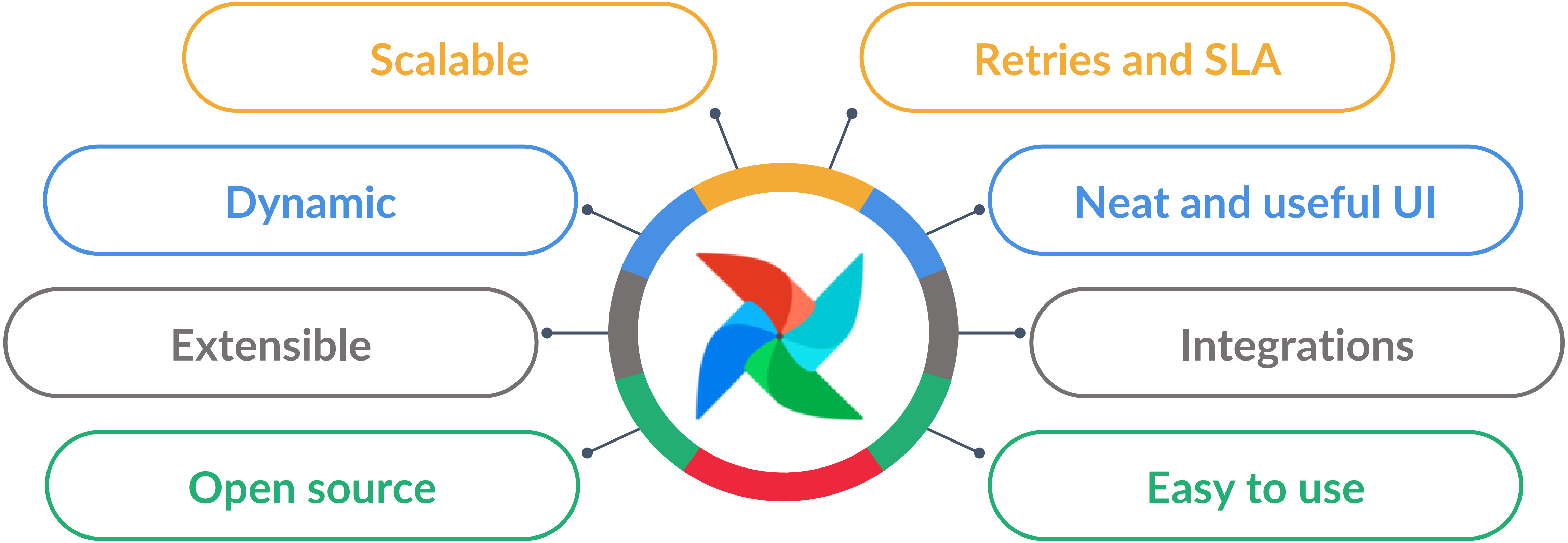
WHY DO WE NEED A NEW TOOL?

- Scalable to **orchestrate** and **schedule** large number of processes
- Efficient **maintenance and monitoring** of pipelines
- **Richer UI** to **visualise** the status and compare against historical runs
- **Retry and timeout feature** for failed tasks
- Easy **SLA** handling
- **Compatibility** with various tools in the data engineering domain
- Strong **community support**

INTRODUCTION TO APACHE AIRFLOW

- Started in October 2014 by Maxime Beauchemin at **Airbnb**
- Aims at meeting the needs of complex data processing pipelines
- Framework to programmatically **create, schedule** and **monitor** data pipelines
- Data pipelines in Airflow
  - Implemented in the form of **DAGs**
  - Created from the Python code
  - Can be generated dynamically

FEATURES OF APACHE AIRFLOW



Scalable

Retries and SLA

Dynamic

Neat and useful UI

Extensible

Integrations

Open source

Easy to use

DAG : DATA PIPELINES IN AIRFLOW

- DAGs or **Directed Acyclic Graphs** form the core structure around which Airflow structures its data pipelines.
- Properties of **DAGs include:**
  - Should be **directed** (i.e., the edges all have a direction indicating which task is dependent on which).
  - Must be **acyclic** (i.e., they can't contain cycles).
  - Are **graph** structures (i.e., a collection of vertices and edges).
- The whole DAG is defined in a Python program.

DAG ATTRIBUTES

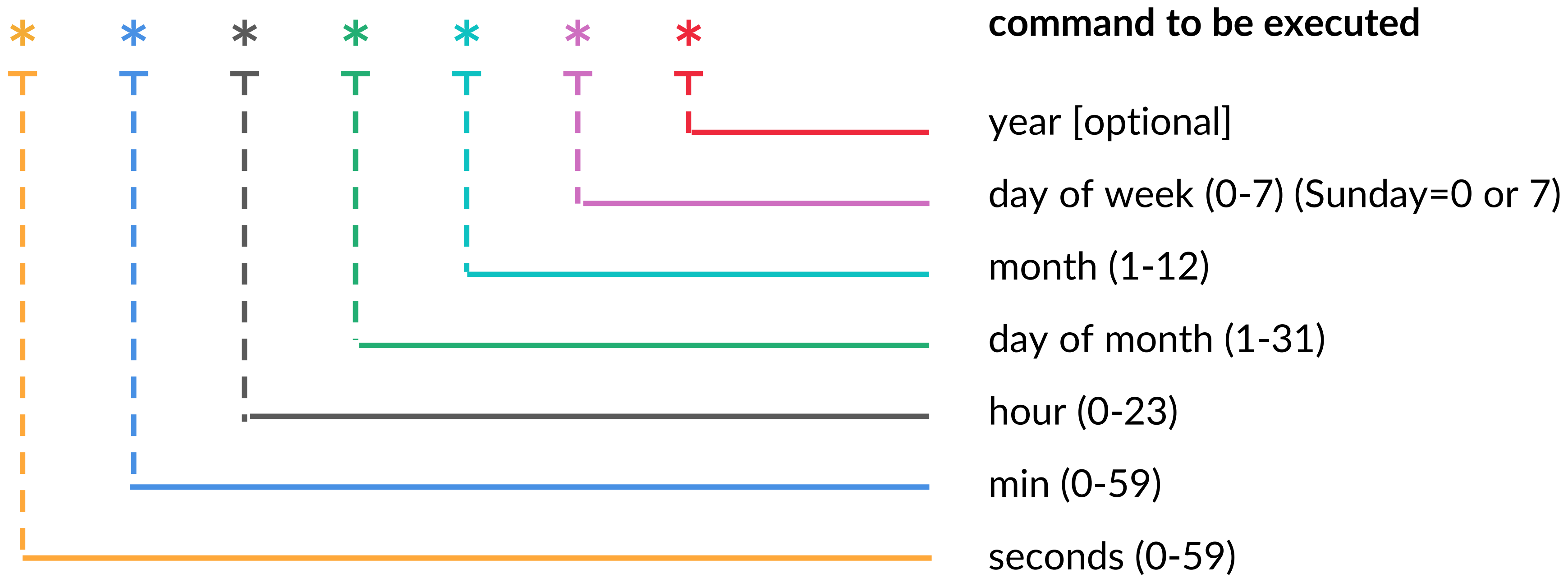
- When creating a DAG we require :
  - ID
  - Description
  - Schedule (can be a cron expression or shorthand string notation like @once, @hourly etc.)
  - Start Date
  - Configs/default arguments
    - Owner
    - Retries

DAG DEFINITION

```
from datetime import datetime
from airflow import DAG

dag_default_configs = {
    'start_date': datetime(2016, 1, 1),
    'owner': 'airflow'
}
dag_object = DAG('my_dag',
              default_args = dag_default_configs,
              description='Sample DAG',
              schedule_interval='0 12 * * *')
```

CRON EXPRESSION GUIDE



* * * * * * * **command to be executed**

year [optional]

day of week (0-7) (Sunday=0 or 7)

month (1-12)

day of month (1-31)

hour (0-23)

min (0-59)

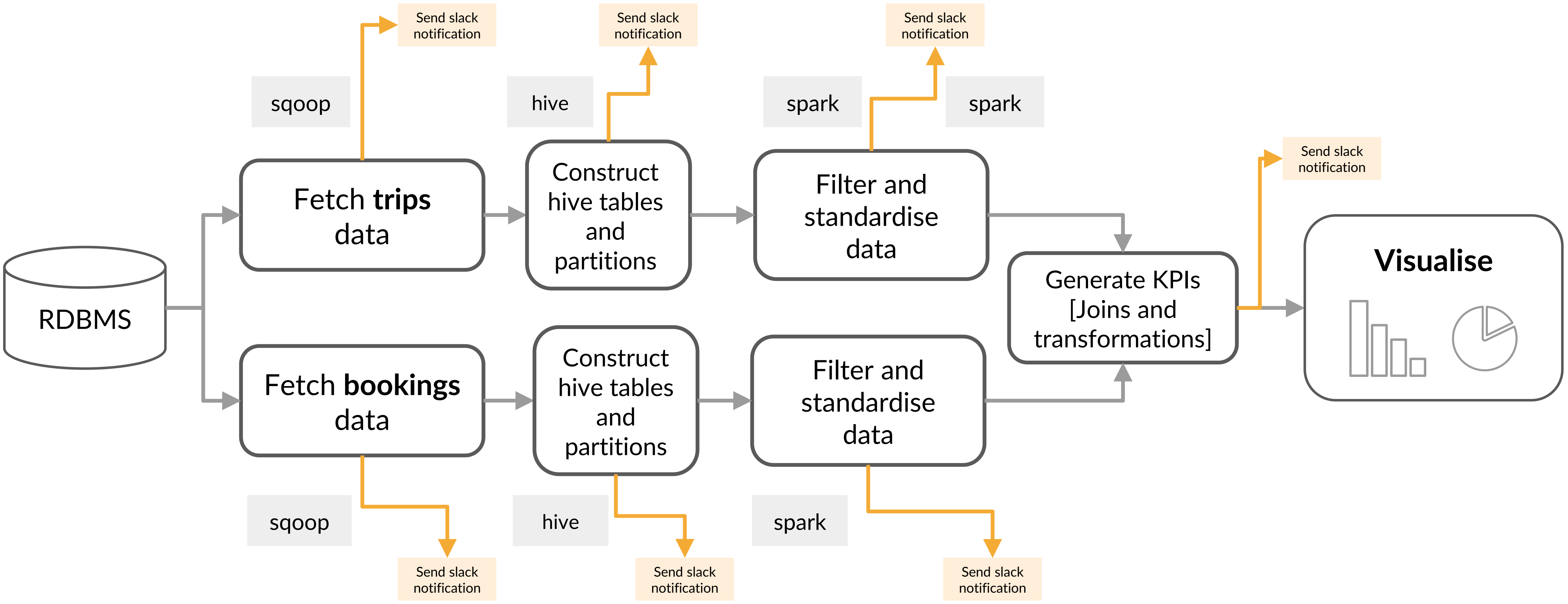
seconds (0-59)

COMPONENTS OF A DAG

A DAG consists of **tasks** and **dependencies** between them.

- Tasks :
  - A Task defines a unit of work within a DAG
  - It is represented as a node in the DAG graph
  - Operators determine what actually gets done by a task.
  - Each task is a python process running on the same or different machines (called worker)
  - E.g: calling a python function, executing a shell script, running a hive query, running a spark job etc.
- Task dependencies :
  - They define the order in which the tasks in a DAG are executed.

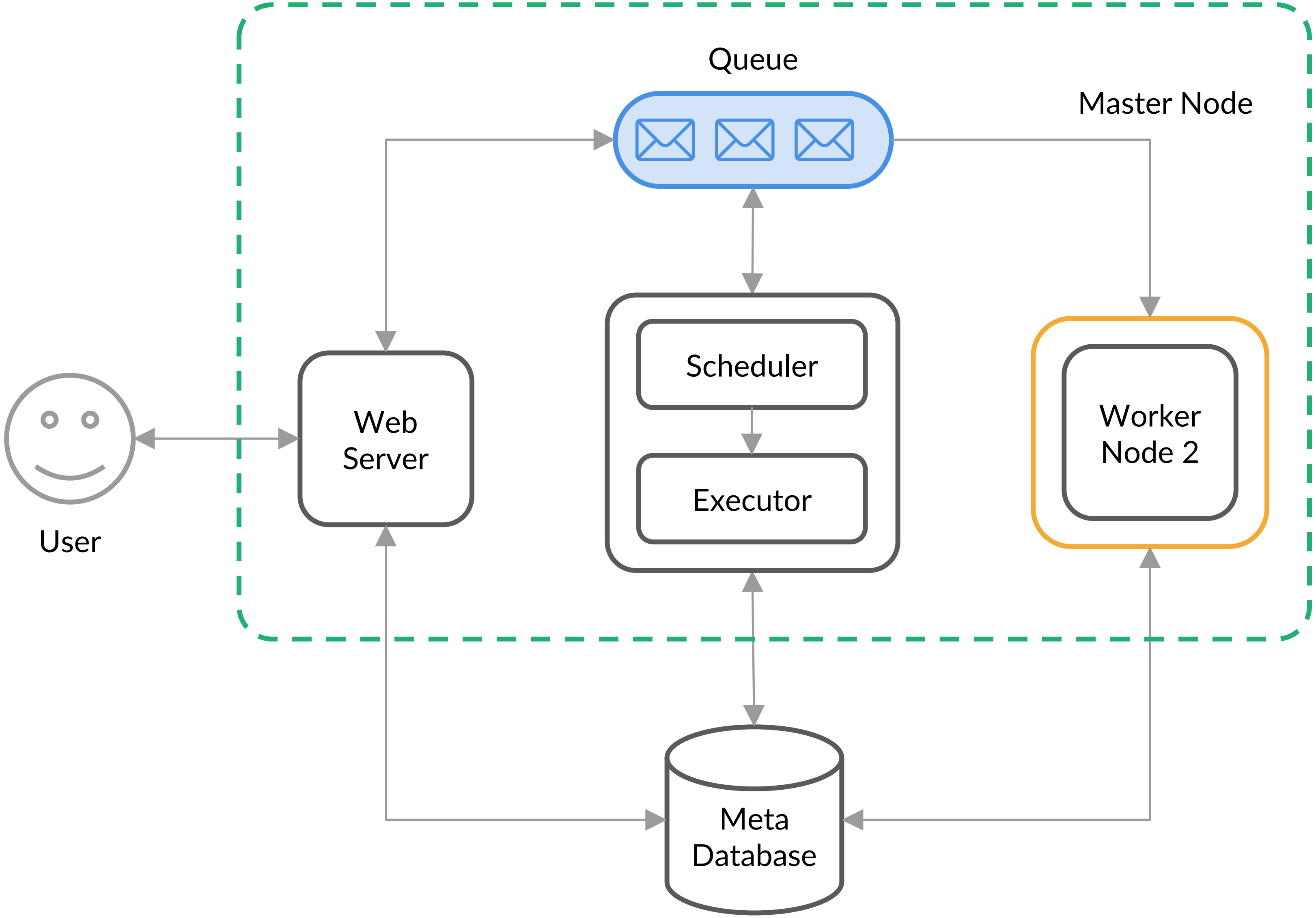
CONSTRUCTING DAG FOR UBER USE CASE



COMPONENTS OF AIRFLOW

- **Web server** serves as the front-end
- **Scheduler** orchestrates various DAGs and their tasks
- **Executor** is the mechanism by which task instances get run
- **Workers** perform the actual work
- **Metadata Database** is used to keep track of task job statuses and other persistent(meta) information

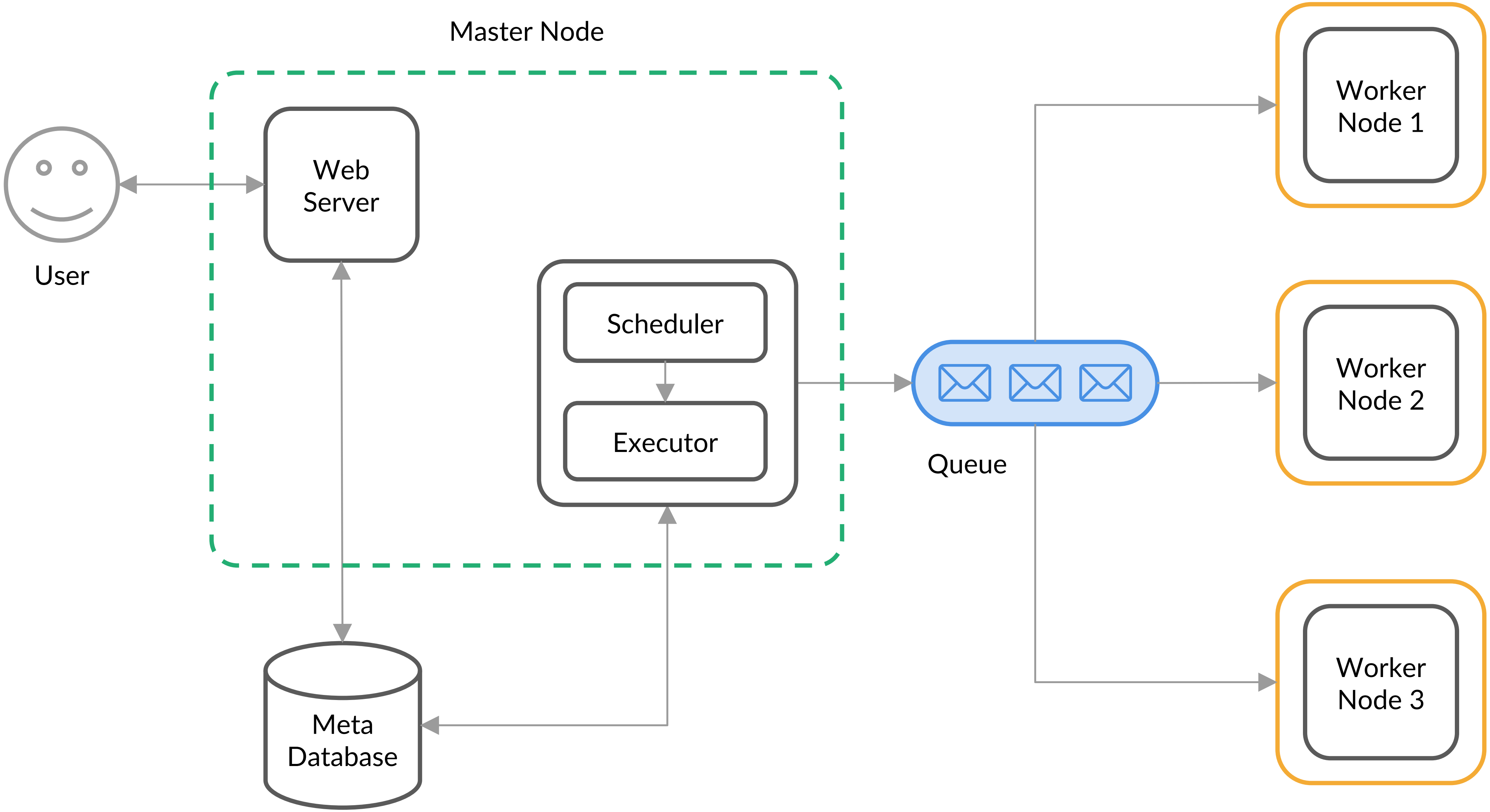
SINGLE NODE ARCHITECTURE



SCALING AIRFLOW: TYPES OF EXECUTORS

- Sequential executor:
  - It is the default executor and will only run one task instance at a time.
  - It is suitable for testing and debugging DAGs before they’re implemented in an industry environment.
- Local executor:
  - It is like a sequential executor with unlimited parallelism.
  - It runs tasks by spawning processes in a controlled fashion in different modes.
- Celery executor:
  - Used in scalable environments
  - Needs RabbitMQ and Redis for configuration.
  - Each worker is in a different node, so it can be easily scaled by adding more nodes.
  - Recommended in production scenario

DISTRIBUTED ARCHITECTURE



SESSION SUMMARY

01 Data pipelines

02 Importance of pipeline orchestration

03 Possible solutions to data pipeline orchestration

04 Introduction to Airflow

05 Airflow Architecture

THANK YOU