Why is Argo different from the others?

Budapest Data Forum

Riccardo Mocchetti - June 2022

What this presentation is not about

- Is Argo the best workflow manager out there?
- Pros and Cons of different orchestrators

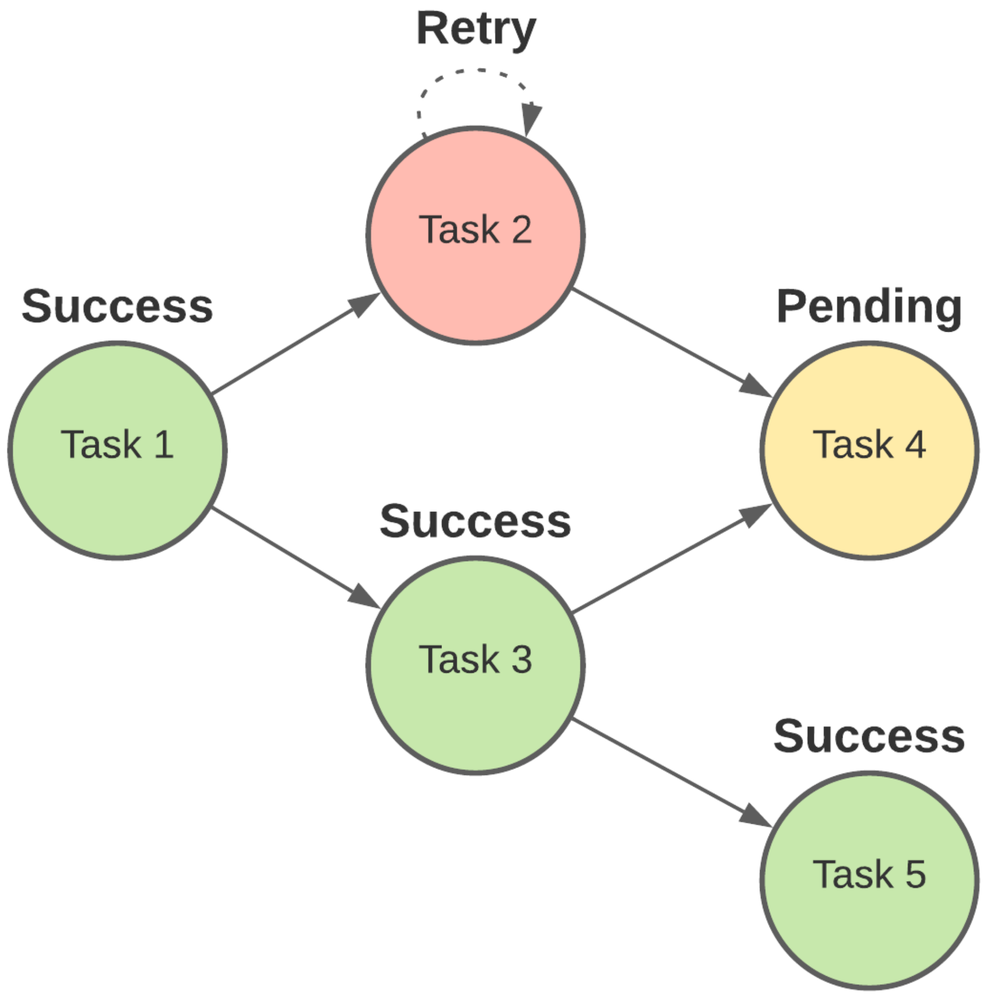
Context

- Data pipelines with a software engineering mindset
- Everything is code
- Automated processes for testing and releasing to production
- Increased reliability

Agenda

- “Workflows as code” orchestrators
- Thinking at different abstraction levels
- Container-Native Orchestration
- Event based workflows

DAGs


Retry
Task 2
Success
Pending
Task 1
Task 4
Success
Task 3
Success
Task 5

Airflow

```python
# Defining a dag
@dag(start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
     schedule_interval="@daily", catchup=False)
def generate_dag():
    op = EmptyOperator(task_id="task")

dag = generate_dag()

# Defining dependencies
first_task >> [second_task, third_task]
third_task << fourth_task
```

- Designed for long running tasks
- Everything goes through the scheduler (and the database)
- Airflow workers execute python

https://airflow.apache.org/docs/apache-airflow/stable/concepts/dags.html

Prefect

```python
from prefect import task, Task, Flow
import random

@task
def random_number():
    return random.randint(0, 100)

@task
def plus_one(x):
    return x + 1

with Flow('My Functional Flow') as flow:
    r = random_number()
    y = plus_one(x=r)
```

- Designed for short lived tasks
- Passing state between tasks and dynamic task generation
- Flows can be run in isolation

https://docs.prefect.io/core/concepts/flows.html#functional-api

DBT

```
select
  orders.id,
  orders.status,
  sum(case when payments.payment_method = 'bank_transfer' then
payments.amount else 0 end) as bank_transfer_amount,
  sum(case when payments.payment_method = 'credit_card' then
payments.amount else 0 end) as credit_card_amount,
  sum(case when payments.payment_method = 'gift_card' then
payments.amount else 0 end) as gift_card_amount,
  sum(amount) as total_amount

from {{ ref('base_orders') }} as orders
left join {{ ref('base_payments') }} as payments on payments.order_id
= orders.id
```

https://docs.getdbt.com/docs/introduction

- Each select statement becomes a model (task)
- DAGs are defined implicitly
- Limited by SQL

https://rittmananalytics.com/blog/2020/5/28/introducing-the-ra-warehouse-dbt-framework-how-rittman-analytics-does-data-centralization

INFINITE LAMBDA

Thinking at a different abstraction level

Difference #1

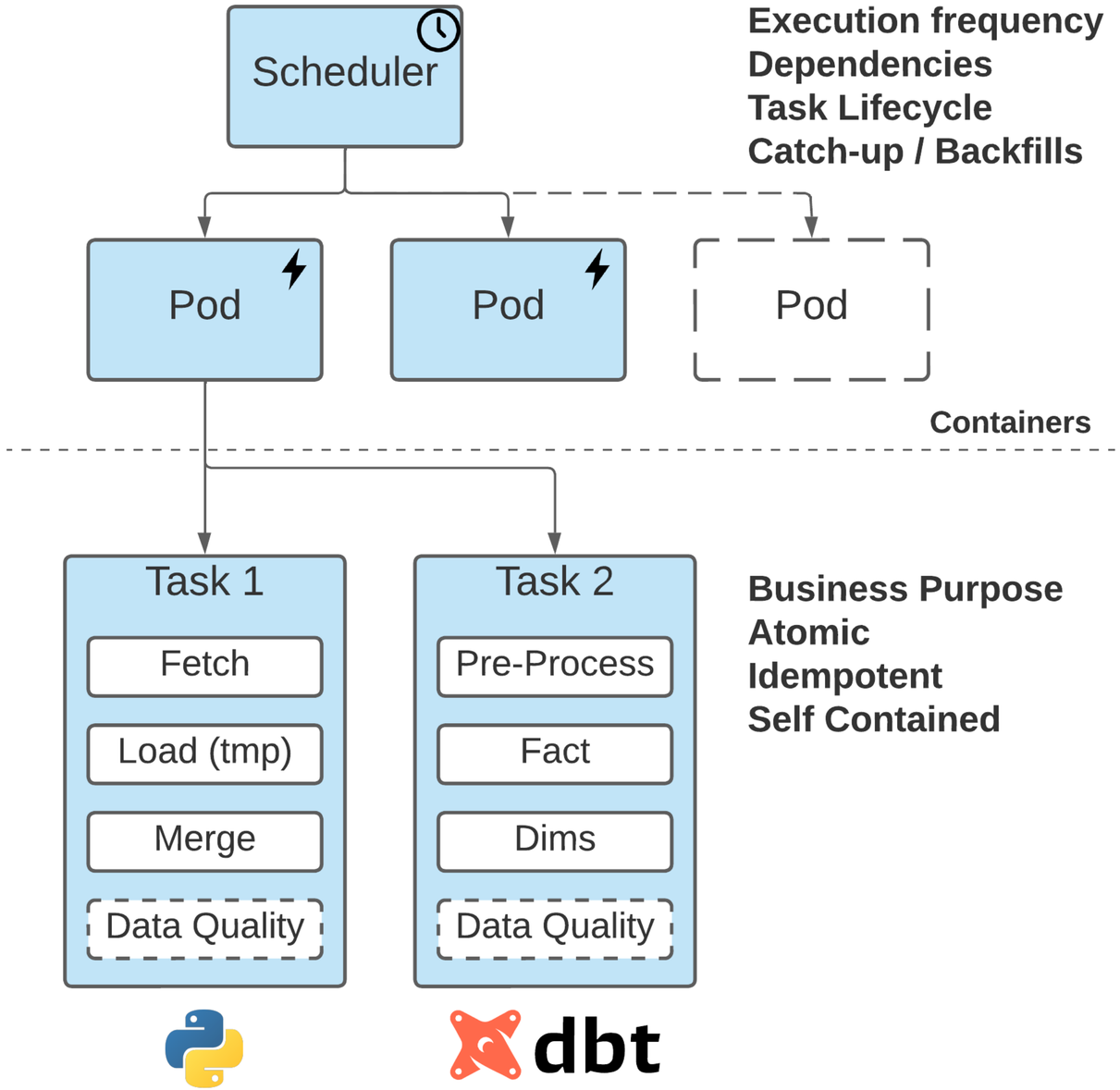
Argo Workflows

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: data-ingestion-
spec:
  entrypoint: data-ingestion
  templates:
  - name: data-ingestion
    container:
      image: my-company/data-ingestion:1.0.1
      command: [python]
      args: ["run.py"]
```

- Containers are the main building blocks
- Containers can be released in isolation
- Encourages to think about business processes

Orchestrator

Tasks



Container-Native Orchestration

Difference #2

Argo Workflows

Argo Workflows is an open source container-native workflow engine for orchestrating parallel jobs on Kubernetes. Argo Workflows is ***implemented as a Kubernetes CRD (Custom Resource Definition).***

https://argoproj.github.io/argo-workflows/#argo-workflows

A little bit about Kubernetes

- Kubernetes is really good at executing long running applications
- Kubernetes is open source, every major cloud supports running Kubernetes as a service
- Kubernetes is highly extensible, one of the reason why it is so popular
- Custom Resource Definitions extend Kubernetes with new resources

Kubernetes Resources

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Kubernetes Resources

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: data-ingestion-
spec:
  entrypoint: data-ingestion
  templates:
  - name: data-ingestion
    container:
      image: my-company/data-ingestion:1.0.1
      command: [python]
      args: ["run.py"]
```

INFINITE LAMBDA

Argo as a CRD

- Argo can leverage all Kubernetes features
  - Install/run
  - High availability and scaling
  - Security and access control
- Can be managed as any other Kubernetes resource
- Cannot be executed outside Kubernetes

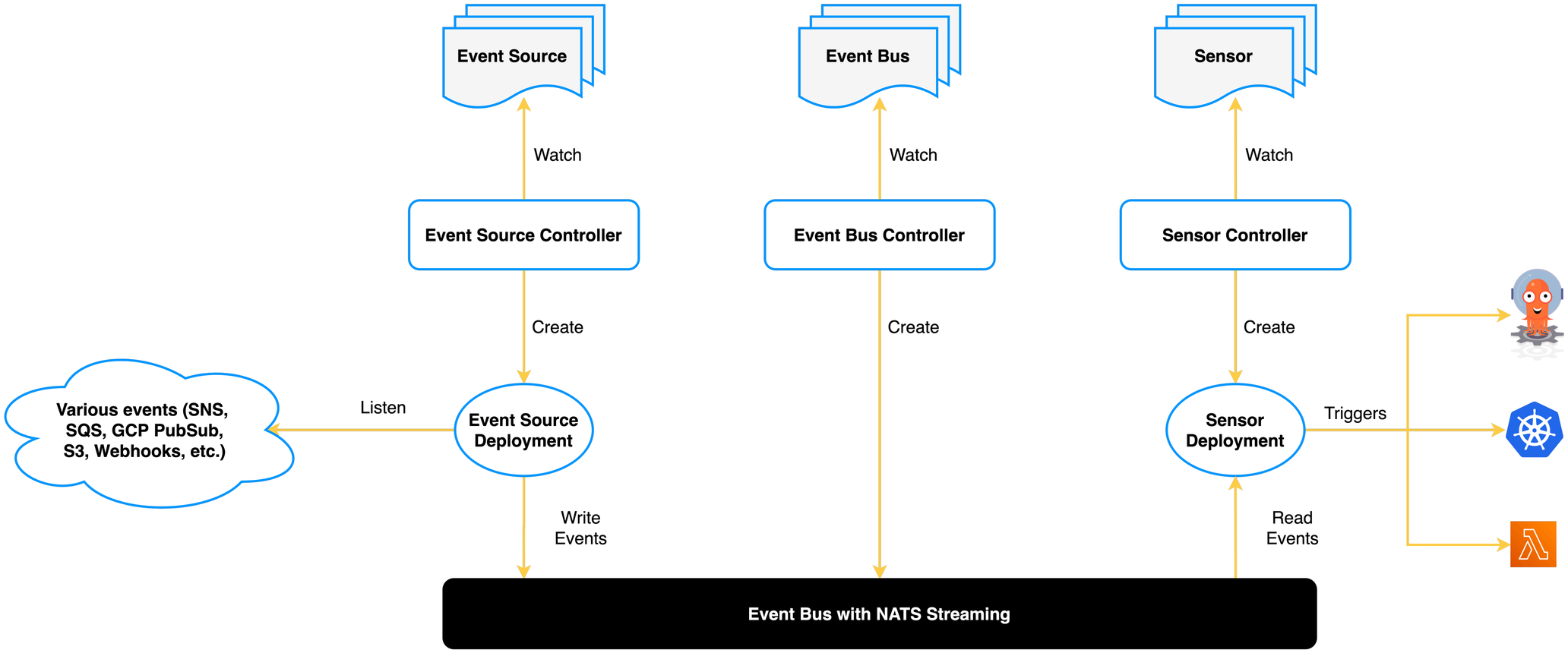
Event Based Workflows

Difference #3

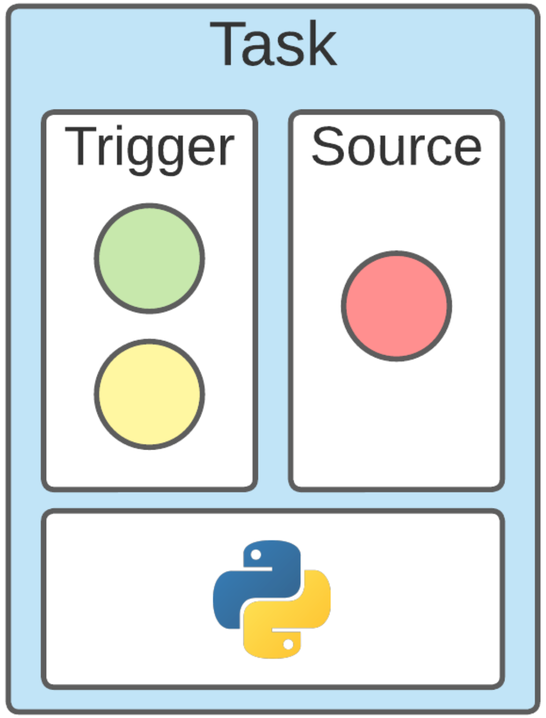
DAGs so far

- DAGs are hardcoded in some kind of specification
- Every change requires a new release
- Difficult to model

Argo Events

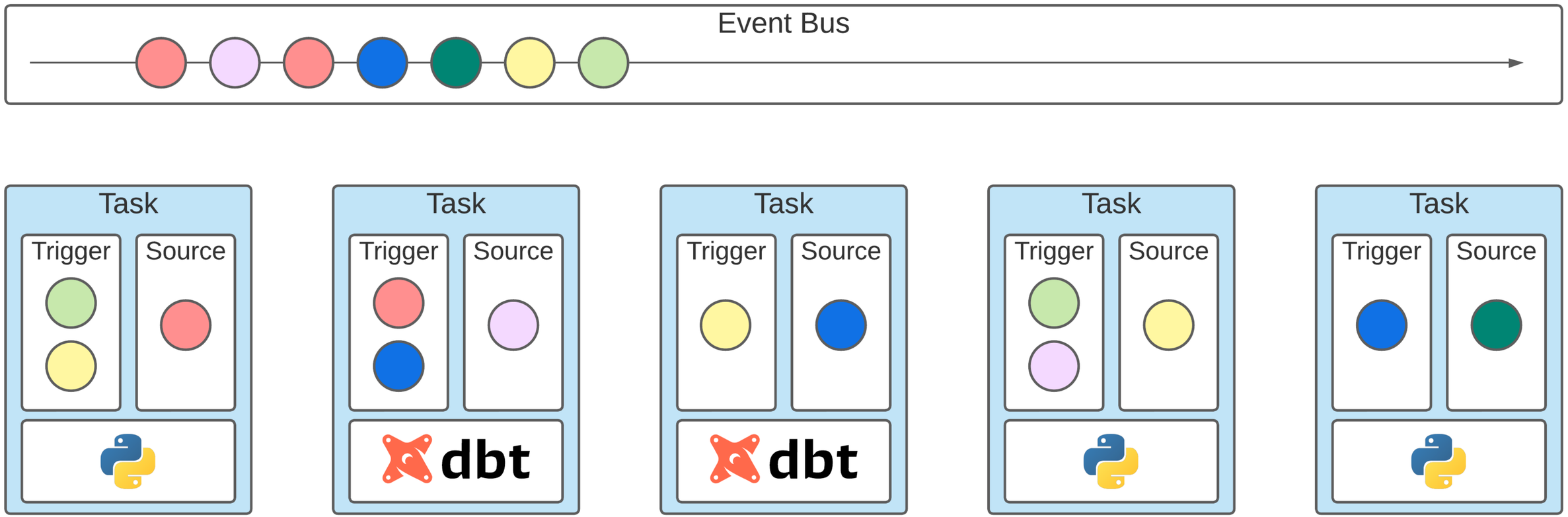

Event Source
Event Bus
Sensor
Watch
Watch
Watch
Event Source Controller
Event Bus Controller
Sensor Controller
Create
Create
Create
Various events (SNS, SQS, GCP PubSub, S3, Webhooks, etc.)
Listen
Event Source Deployment
Sensor Deployment
Triggers
Write Events
Read Events
Event Bus with NATS Streaming

Argo Events - Single Task



- Business purpose
- Atomic
- Idempotent
- Self-contained
- Triggered by green and yellow messages
- Emits red messages

Argo Events - multiple tasks



In conclusion

- “Workflow as code” orchestrators encourage us to think at a low level of abstraction
- Argo Workflows container-native approach encourages us to think in terms of business processes
- Argo Events offers a different approach to tackle tasks dependencies and workflows definitions

Thank you!