Aalto University
School of Science

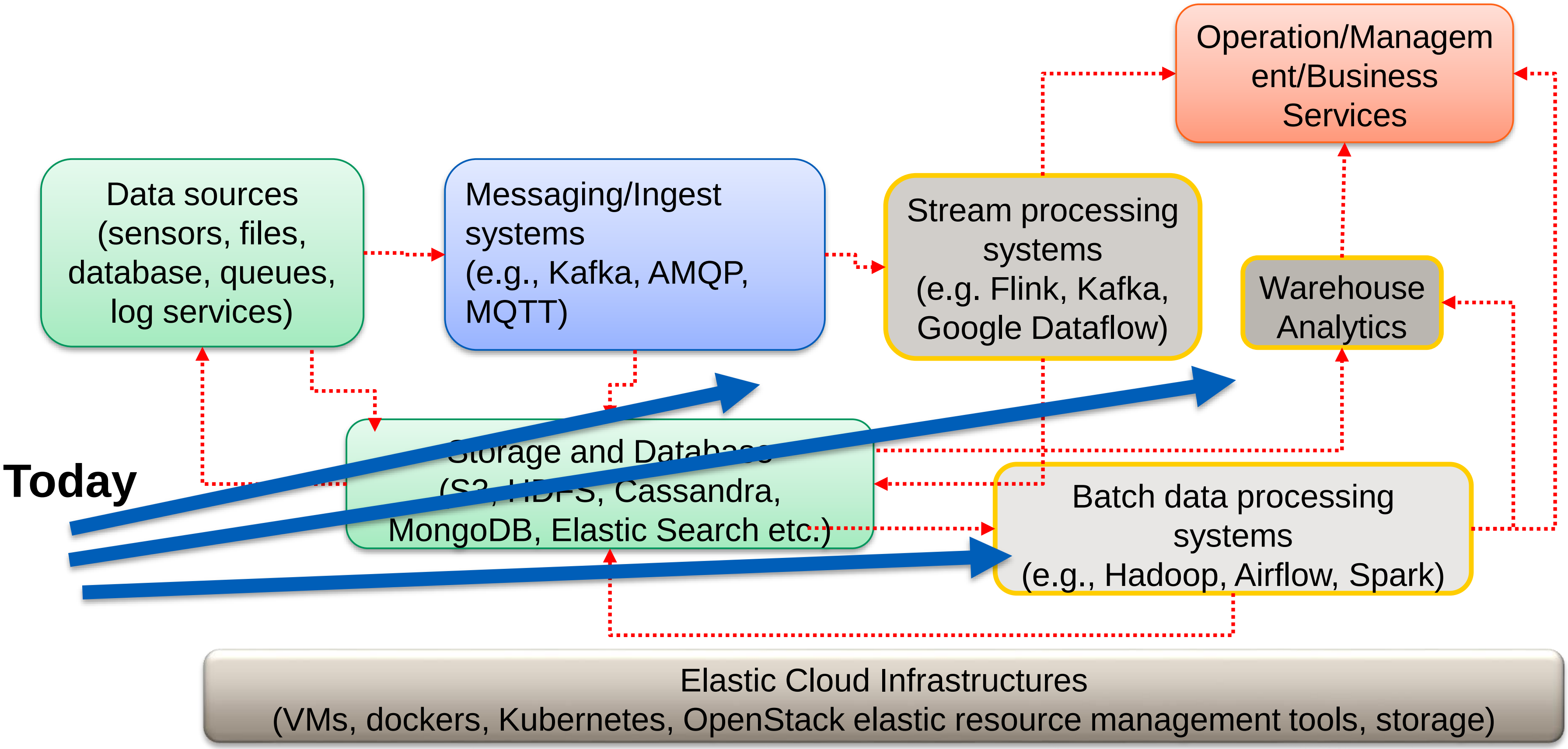
Workflows for Big Data Platforms

Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi, *https://rdsea.github.io*

Schedule

- **The role of workflows in big data platforms**
- **Key concepts about workflows/pipelines**
- **Frameworks**
  - Function-as-a-Service, Apache Airflow
- **Summary**

Big data at large-scale



Tasks in big data platforms

- **Data Processing**
  - data analytics, extraction, transformation, data transfers
- **Machine learning**
  - collecting data, training experiments, serving machine learning algorithms
- **Automation in big platforms**
  - service deployment, elasticity, incident management
- **Service integration with big data platforms**
  - integration with customer service, communications of analytics

Use cases

- **Implementing ETL, data cleansing and backup**
  - access and coordinate many different compute services, data sources, ingestion and extraction applications
- **Implementing complex predictive maintenance**
  - coordination of machine learning pipelines and communication with humans/optimization services
- **Analytics-as a service: metrics, user activities analytics, testing, e.g.**
  - analytics of big data sources

Example of using workflows

Security-related information and metrics from distributed customers

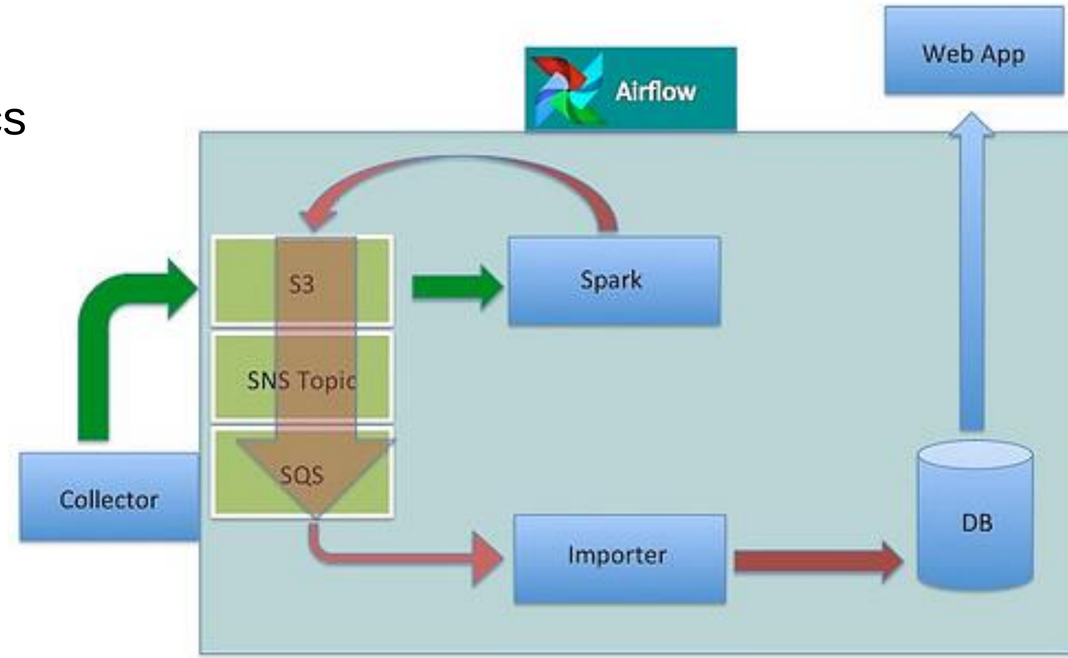
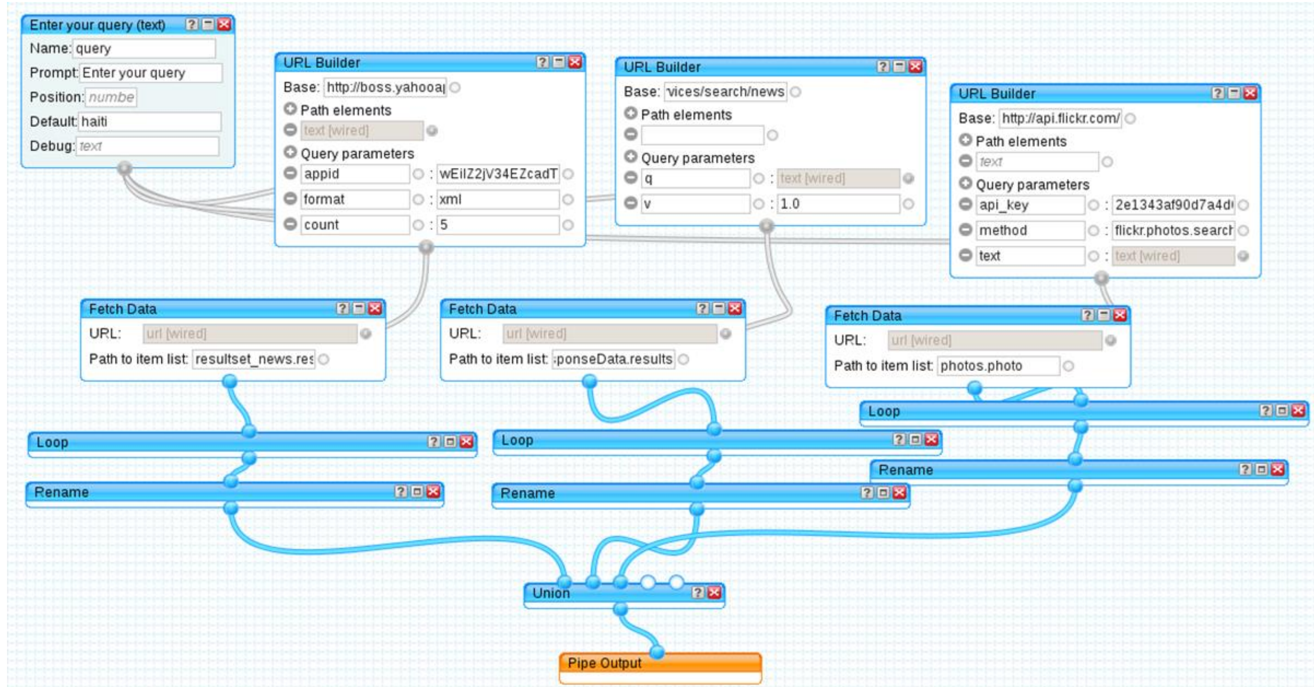


Figure Source: http://highscalability.com/blog/2015/9/3/how-agari-uses-airbnbs-airflow-as-a-smarter-cron.html

A long history – workflows are well-known!

Based on Yahoo Pipes 2012



Tasks and workflows

- **Diverse types of tasks**
  - task can be simple or complex (e.g., a task running an AI algorithm)
  - tasks are performed by software and humans
    - *triggers and sinks can be humans*
- **Workflow**
  - coordinates many tasks, tasks are not really "carried out" by workflows
  - workflow can be simple, like a pipeline of a sequence of tasks or complex with fork/loop

Workflow and pipeline/data workflow

- **Workflows: a set of coordinated activities**
  - different categories of tasks with control/data dependencies
- **Data workflows → data pipeline**

”a pipeline is a set of data processing elements connected in series, where the output of one element is the input of the next one”

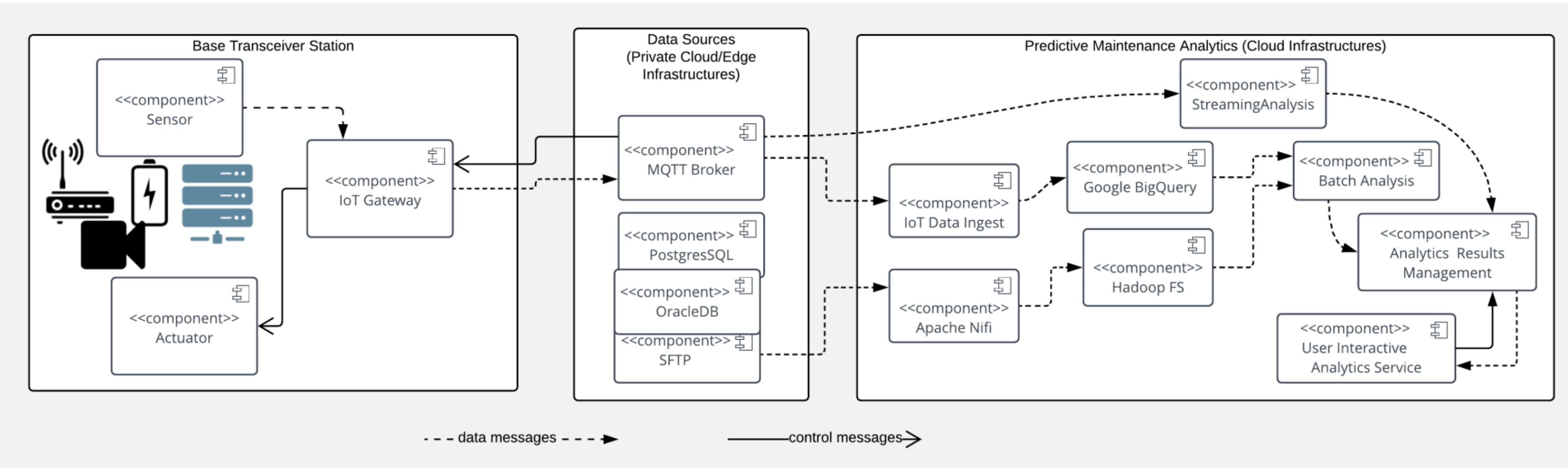
Source: https://en.wikipedia.org/wiki/Pipeline_%28computing%29

Workflow and pipeline/data workflow

- **Two interpretations:**
  - a pipeline is a simple workflow
  - a pipeline coordinates different (sub)workflows
- **In many complex big data designs**
  - at the high-level we have pipelines to glue different (sub)systems using different technologies

  → software component pipeline design

Examples: software component pipeline design in big data



This lecture: we talk about pipelines of "tasks", not component designs

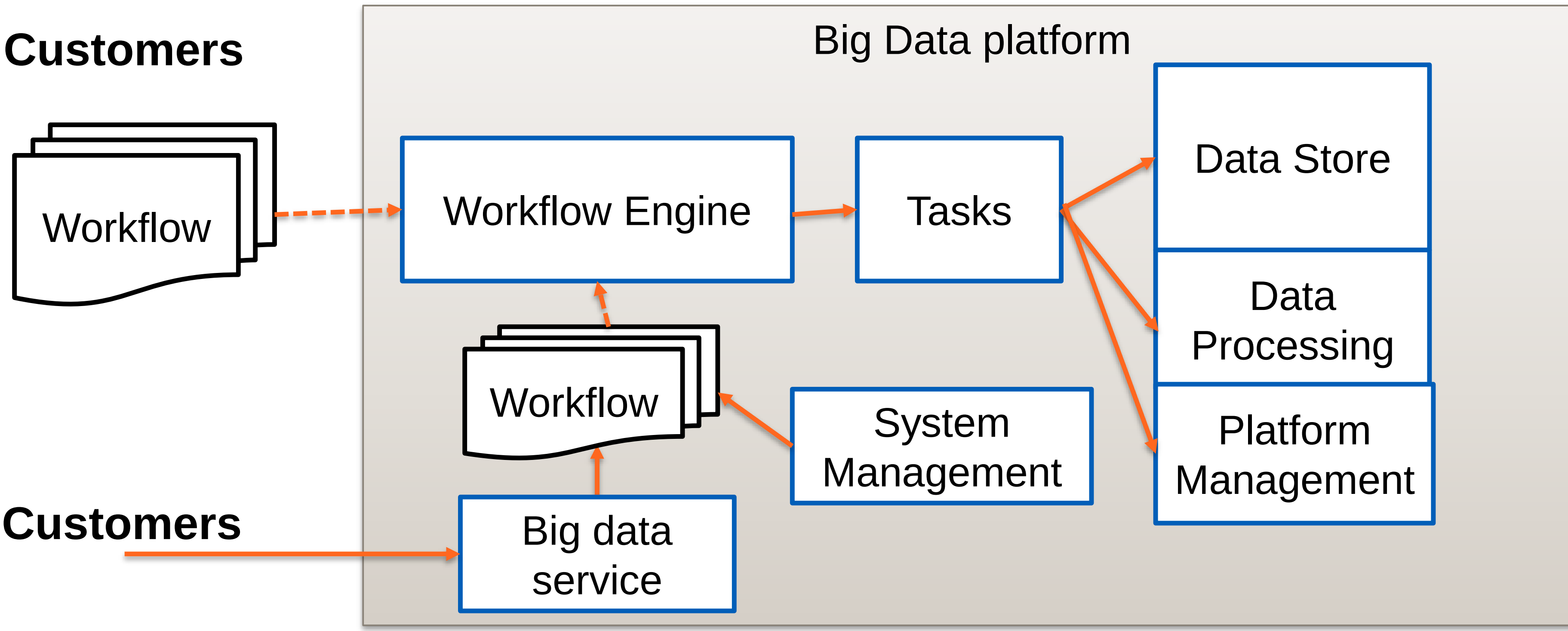
Diverse types of frameworks and capabilities

- **Workflows/pipelines focused on data analytics**
  - big data analytics in generic ways
  - ML pipelines
- **Workflows/pipelines focused on platform management**
  - configuration of services, job managements
- **Workflows/pipelines focused on service integration**
  - integrate different services of analytics, business and humans
- **Application domains**
  - IoT, Industry 4.0, Health data analytics, Science, E-commerce, ...

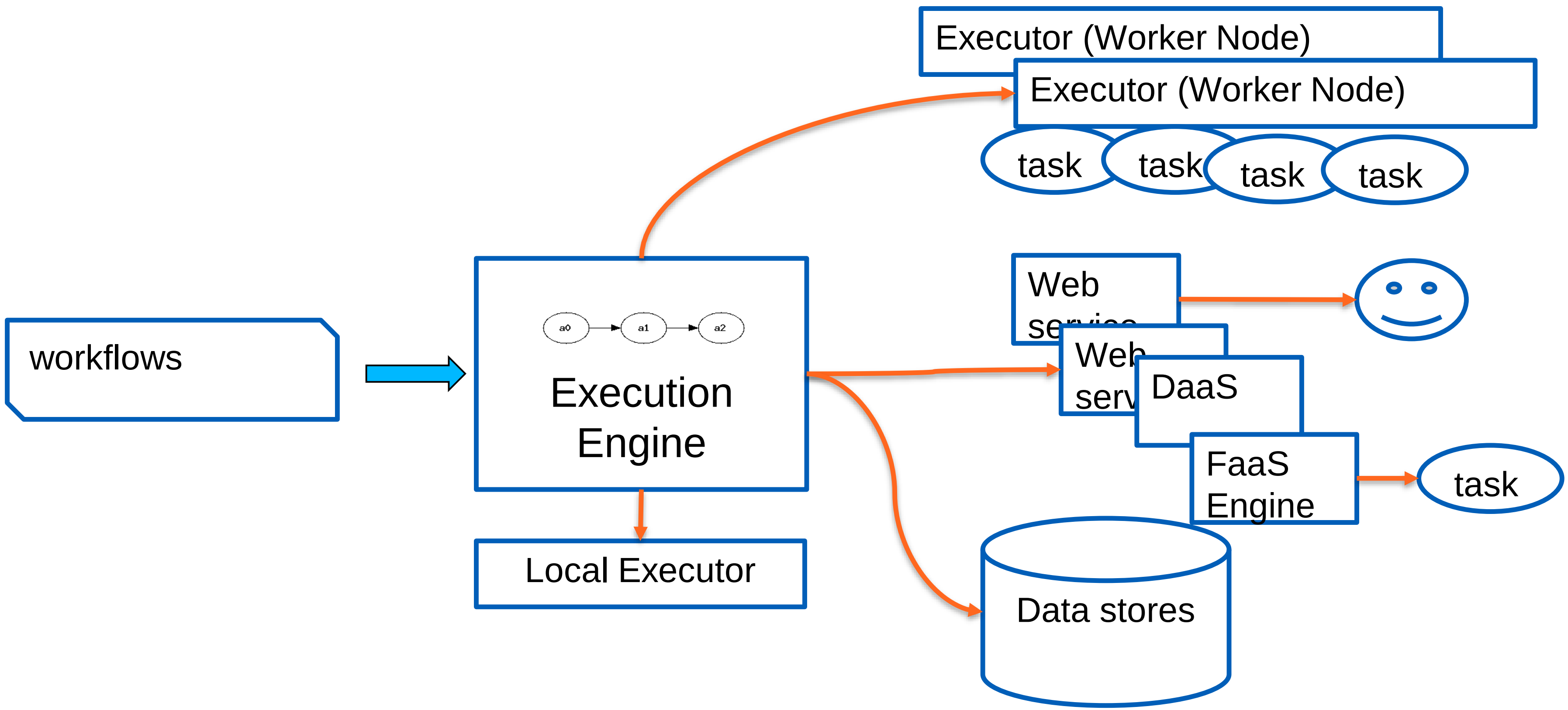
Workflows in big data platforms

- **Often is for big data analytics and ETL**
- **But analytics is not just about data processing tasks**
- **Storage: where is the data from? Where is the sink of data?**
- **Communication of results**
  - is software or human the receiver of the analytics results?
  - software: messaging broker, serverless function, REST API, Webhook?
  - people: email, SMS, Slack?

Workflows in big data platforms: more than analytics



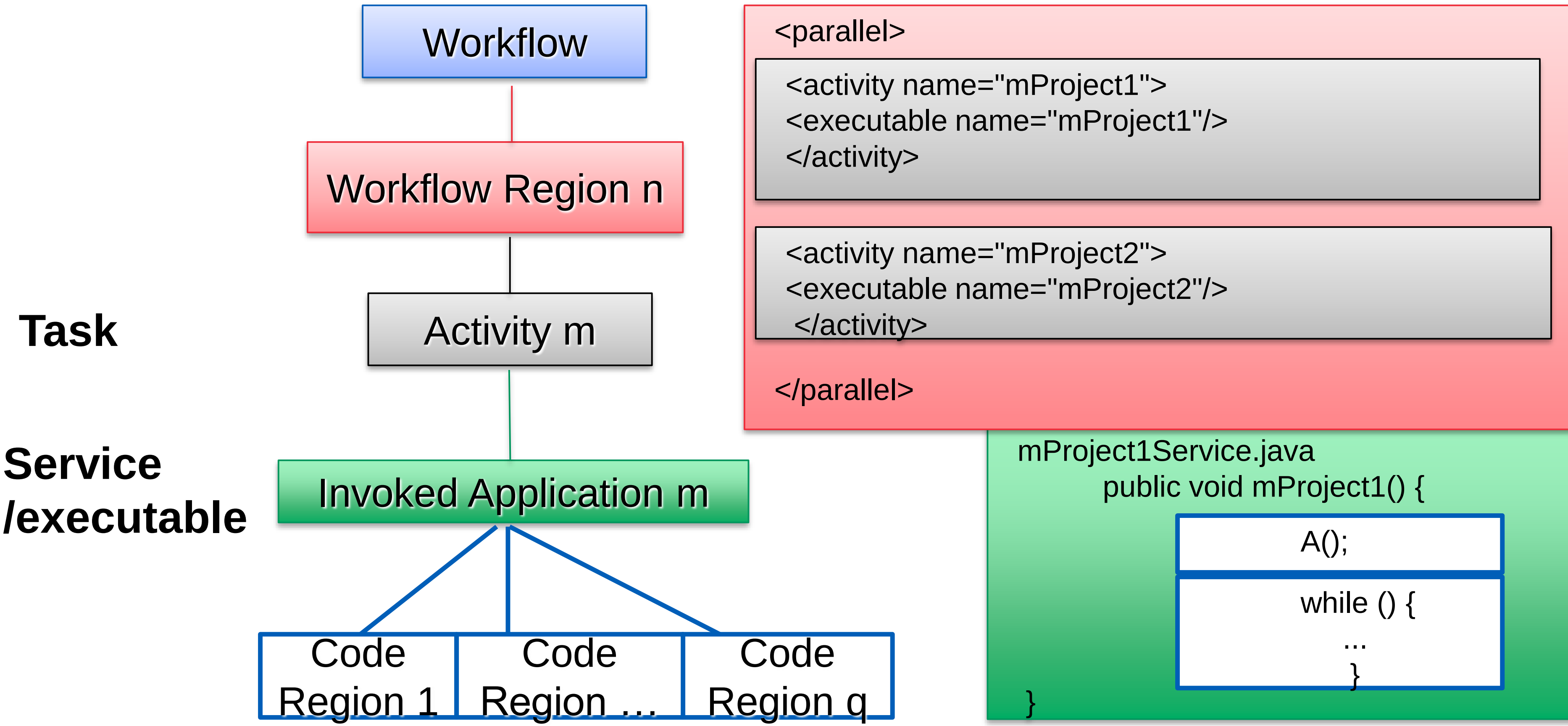
Common workflow execution models



Key components

- **Tasks/Activities**
  - describe a single work (it does not mean small)
  - tasks can be carried out by humans, executables, scripts, batch applications, stream applications and services.
- **Workflow Languages**
  - how to structure/describe tasks, dataflows, and control flows
- **Workflow Engine**
  - execute the workflow by orchestrating tasks
  - usually call remote services to run tasks

Structured view of workflows



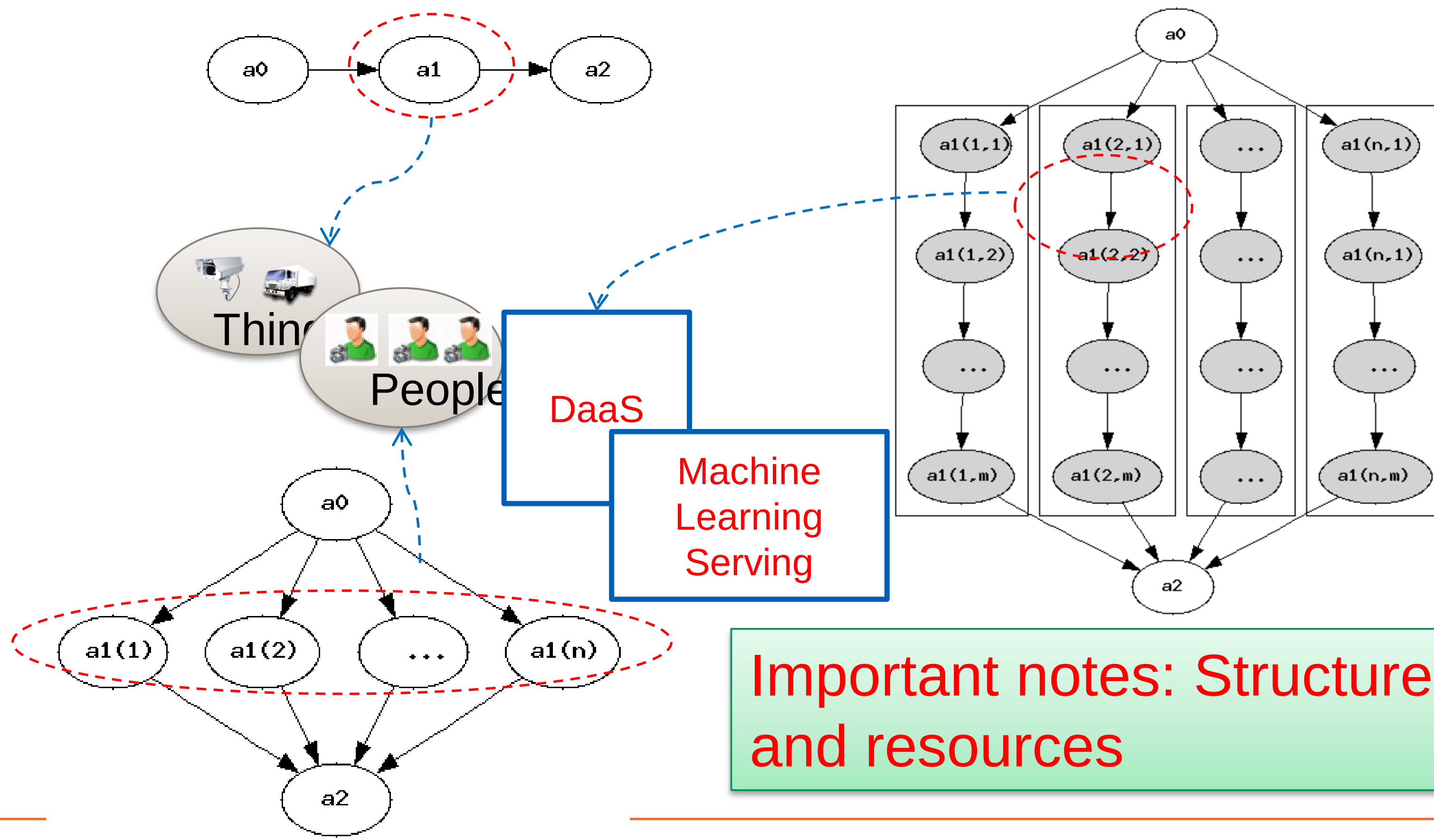
Runtime aspects

- **Parallel and distributed execution**
  - tasks are deployed and running in different machines
  - multiple workflows are running
- **Long running**
  - can be hours or weeks!
- **Checkpoint and recovery**
- **Monitoring and tracking**
  - which tasks are running, where are they?
- **Stateful management**
  - dependencies among tasks w.r.t control and data

Describing workflows

- **Programming languages with procedural code**
  - general- and specific-purpose programming languages, such as Java, Python, Swift
  - common ways in big data platforms
- **Descriptive languages with declarative schemas**
  - BPEL and several languages designed for specific workflow engines
  - common in business and scientific workflows

Tasks orchestration



Important notes: Structures and resources

Key requirements for workflow frameworks in big data platforms

- **Rich connectors to various data sources**
- **Big data computation engines**
  - for running different workload: ML and (batch/stream) big data processing
- **Different underlying cloud infrastructures**
- **REST APIs and message broker integration**

Existing frameworks for your study

- **Many workflow frameworks exist**
- **Apache Oozie: http://oozie.apache.org/**
  - designed to work with Hadoop: orchestrating Hadoop jobs
    - *it is important if you need to manage a lot of Hadoop/MapReduce jobs*
  - Fluent Job API & XML-based workflows
- **Serverless-based: Function-as-a-Service**
  - e.g., Microsoft, Google, AWS serverless/function-as-a-service
- **Apache Airflow: a generic workflow framework**
- **Workflows for managing machine learning pipelines**

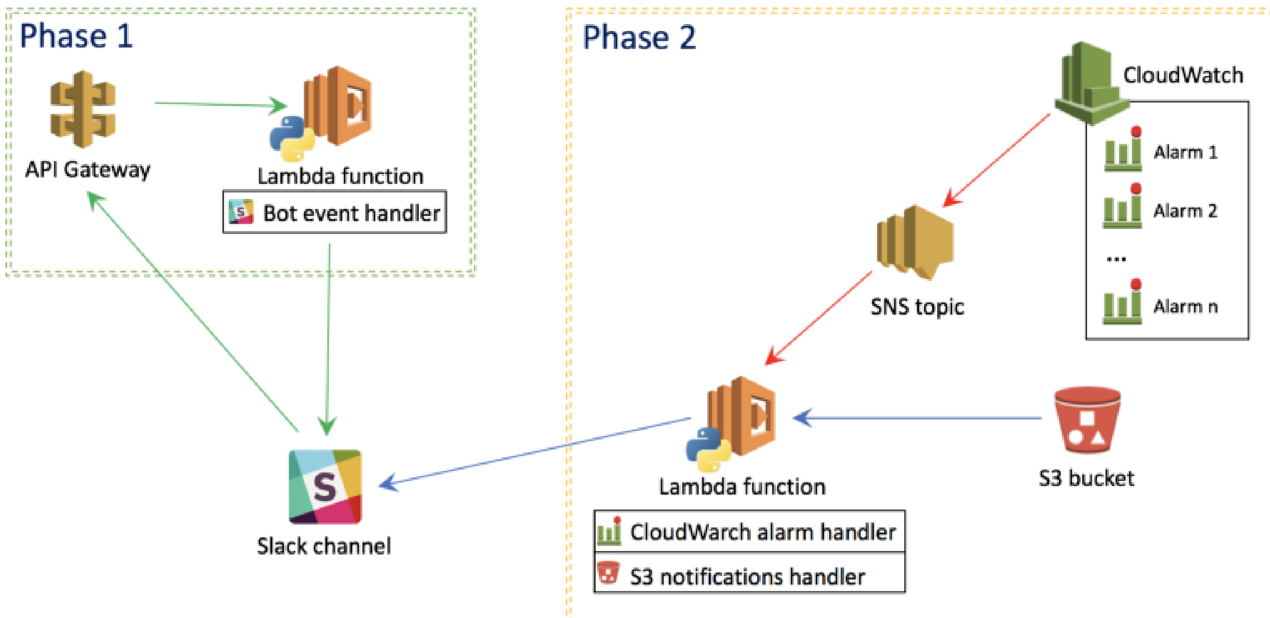
Example with serverless computing/function-as-a-service

Serverless/Function-as-a-service

- **Key principles**
  - running code without managing complex back-end server/application server infrastructures
  - tasks in your application: described as functions
    - *As you typically see in your Java/Python/JavaScript code*
  - functions are uploaded to FaaS engines and will be executed based on different triggers (e.g., direct call or events)
- **Event-driven triggers!**
  - triggered from HTTP calls, messages (brokers), storage events (e.g. new files are stored)

Example: monitoring and human actions

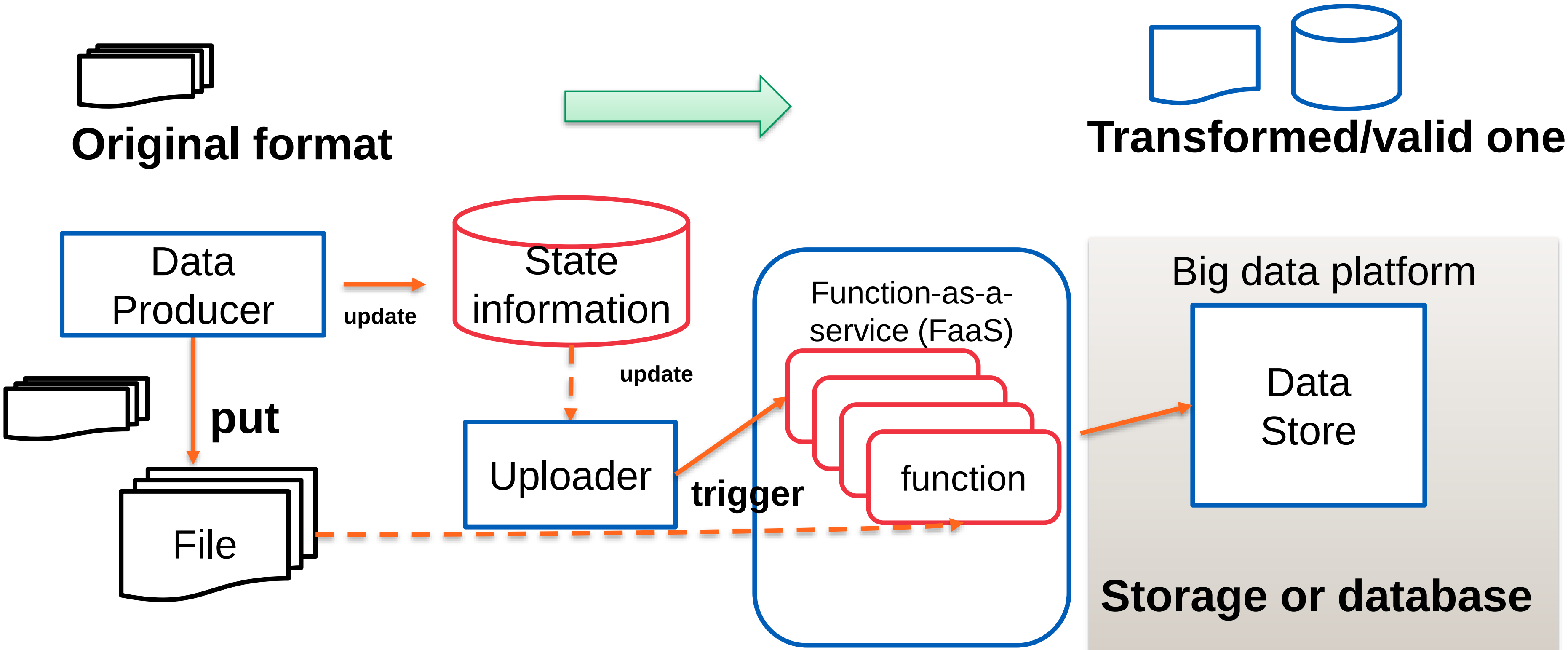
From Anton Chernysh, Source: https://medium.com/devoops-and-universe/serverless-slack-bot-on-aws-vs-azure-getting-notified-instantly-ab0916393e1d



Function-as-a-service and workflows

- **Basic tasks can be implemented as a function**
  - follow the function-as-a-service model (serverless)
  - FaaS engine is a service for executing tasks
- **Workflow coordination with additional features**
  - using coordination engines: explicitly coordinate functions by calling functions executed by FaaS engines
  - implicitly coordinate functions –as a workflows - using connectors and triggers

Recall: data ingestion

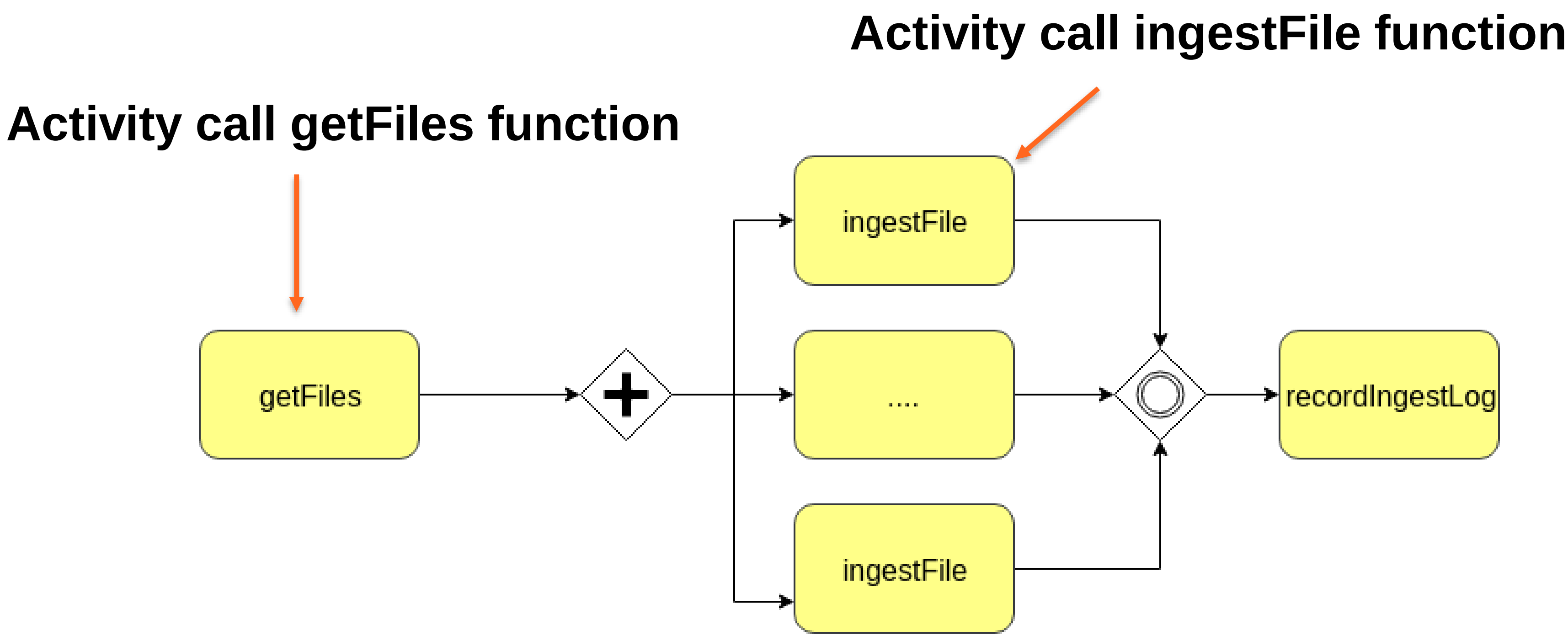


See some possible workflows?

Using function-as-a-service in clouds

- **Azure**
  - Azure Function for tasks
  - Azure durable functions (https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-orchestrations)
- **Amazon Web services**
  - AWS Lambda for tasks
  - AWS Step Functions (https://aws.amazon.com/step-functions/)

Example of Azure durable functions



Code

```
module.exports = df.orchestrator(function* (context,req) {
    const tasks = [];

    //....
    //....

    const files = yield context.df.callActivity("getFiles", username);
    for (const file in files) {
        console.log("Deal with file "+file);
        tasks.push(yield context.df.callActivity("ingestFile", file));
    }
    return tasks;

});
```

Pros and cons

- **Pros:**
  - work well in big data ecosystems offered with cloud providers
  - easy to integrate with other cloud services
- **Cons**
  - lock-in with big cloud providers
    - *Difficult if you have to change services in your ecosystem*
  - long running problems: a task is usually short
  - limited coordination/workflow engines compared with well-known scientific/data-intensive analytics workflows

Example with Apache Airflow

https://airflow.apache.org

Airflow overview

- **Originally from Airbnb**
- **Features**
  - Dynamic, extensible, scalable workflows
  - Programmable language-based workflows
    - *Write workflows as procedural code*
- **Good and easy to study to understand concepts of workflows/data pipeline**
- **Google Cloud Composer is a cloud-provided version of Airflow**
  - https://cloud.google.com/composer/

Many connectors

async	`pip install 'apache-airflow[async]'`	Async worker classes
celery	`pip install 'apache-airflow[celery]'`	CeleryExecutor
cloudant	`pip install 'apache-airflow[cloudant]'`	Cloudant hook
crypto	`pip install 'apache-airflow[crypto]'`	Encrypt connection p
devel	`pip install 'apache-airflow[devel]'`	Minimum dev tools re
devel_hadoop	`pip install 'apache-airflow[devel_hadoop]'`	Airflow + dependenci
druid	`pip install 'apache-airflow[druid]'`	Druid related operato
gcp	`pip install 'apache-airflow[gcp]'`	Google Cloud Platfor
github_enterprise	`pip install 'apache-airflow[github_enterprise]'`	GitHub Enterprise au
google_auth	`pip install 'apache-airflow[google_auth]'`	Google auth backend
hdfs	`pip install 'apache-airflow[hdfs]'`	HDFS hooks and ope
hive	`pip install 'apache-airflow[hive]'`	All Hive related opera
jdbc	`pip install 'apache-airflow[jdbc]'`	JDBC hooks and oper
kerberos	`pip install 'apache-airflow[kerberos]'`	Kerberos integration

kubernetes	`pip install 'apache-airflow[kubernetes]'`	Kubernetes Executor
ldap	`pip install 'apache-airflow[ldap]'`	LDAP authentication
mssql	`pip install 'apache-airflow[mssql]'`	Microsoft SQL Server
mysql	`pip install 'apache-airflow[mysql]'`	MySQL operators and
oracle	`pip install 'apache-airflow[oracle]'`	Oracle hooks and ope
password	`pip install 'apache-airflow[password]'`	Password authentica
postgres	`pip install 'apache-airflow[postgres]'`	PostgreSQL operator
qds	`pip install 'apache-airflow[qds]'`	Enable QDS (Qubole
rabbitmq	`pip install 'apache-airflow[rabbitmq]'`	RabbitMQ support as
redis	`pip install 'apache-airflow[redis]'`	Redis hooks and sens
s3	`pip install 'apache-airflow[s3]'`	`S3KeySensor`, `S3Prefi`
samba	`pip install apache-airflow[samba]'`	`airflow.operators.hiv`
slack	`pip install 'apache-airflow[slack']`	`airflow.operators.sla`
ssh	`pip install 'apache-airflow[ssh]'`	SSH hooks and Opera
vertica	`pip install 'apache-airflow[vertica]'`	Vertica hook support

From https://airflow.apache.org/installation.html#extra-packages

Cloud integration and big data support

- **Several supports with known cloud providers**
  - Microsoft Azure
  - Amazon Web Services
  - Databricks
  - Google Cloud Platform
- **Big data supports**
  - Hadoop, Hive, Druid, Presto
- **Distributed execution**

Airflow workflow structure

- **Workflow is a DAG (Direct Acyclic Graph)**
  - a workflow consists of a set of activities represented in a DAG
  - workflow and activities are programed using Python – structures described in code
- **Workflow activities are described by Airflow operator objects**
  - tasks are created when instantiating operator objects

Airflow operators/tasks

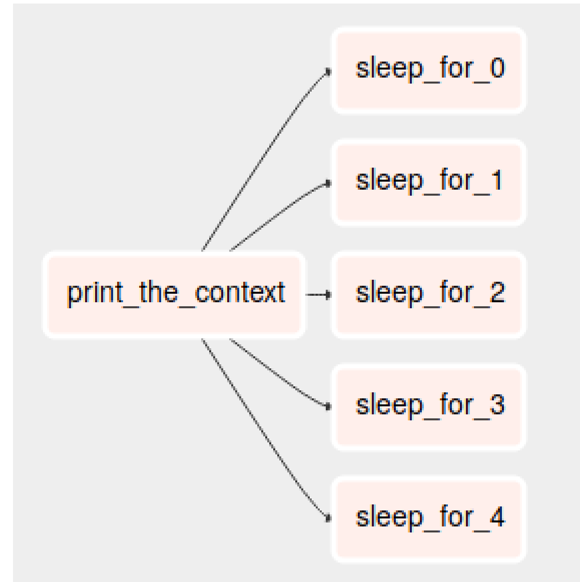
- **Tasks are implemented using operators**
- **Rich set of operators**
  - we can program different kinds of tasks and integrate with different systems

- **Different Types of operators for workflow activities**
  - BashOperator, PythonOperator, EmailOperator, HTTPOperator, SqlOperator, Sensor, DockerOperator, HiveOperator, S3FileTransferOperator, PrestoToMysqlOperator, SlackOperator

Example of operators

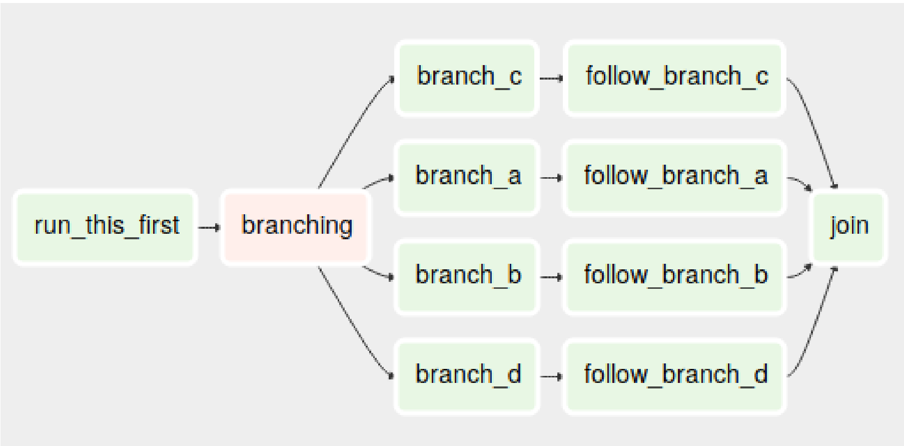
- **High-level structure is mapped to python and suitable operators**

```
for i in range(5):
    task = PythonOperator(
        task_id='sleep_for_' + str(i),
        python_callable=my_sleeping_function,
        op_kwargs={'random_base': float(i) / 10},
        dag=dag,
    )
```



Code and figures captured from Airflow UI:
DAG: example_python_operator
schedule: None

Example of branching




```python
run_this_first = DummyOperator(
    task_id='run_this_first',
    dag=dag,
)

options = ['branch_a', 'branch_b', 'branch_c', 'branch_d']

branching = BranchPythonOperator(
    task_id='branching',
    python_callable=lambda: random.choice(options),
    dag=dag,
)
run_this_first >> branching

join = DummyOperator(
    task_id='join',
    trigger_rule='one_success',
    dag=dag,
)
```

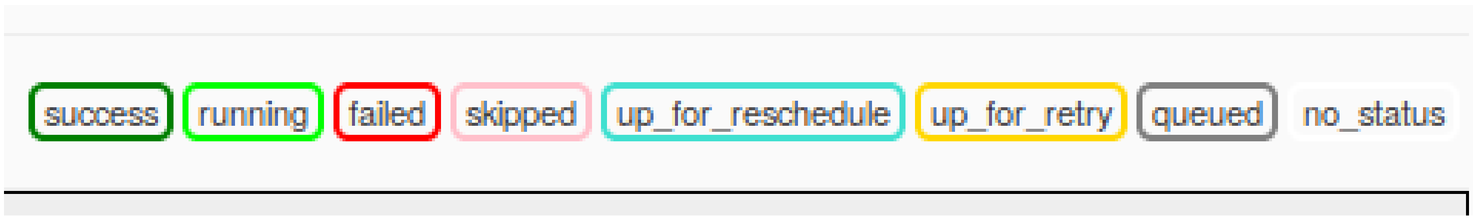
Code and figures captured from Airflow UI
DAG: example_branch_operator
schedule: @daily

Scheduling and execution

- **You can schedule the workflow like a cron job**
  - execute once, every minutes, hours, …
- **Trigger from external**
  - tasks can be triggered as normal (upstream tasks finishes, dependencies)
  - or specific triggers
- **Very suitable ingestion and batch analytics job managements**
  - the ingestion and analytics are done within tasks

Task lifecycle

Different states

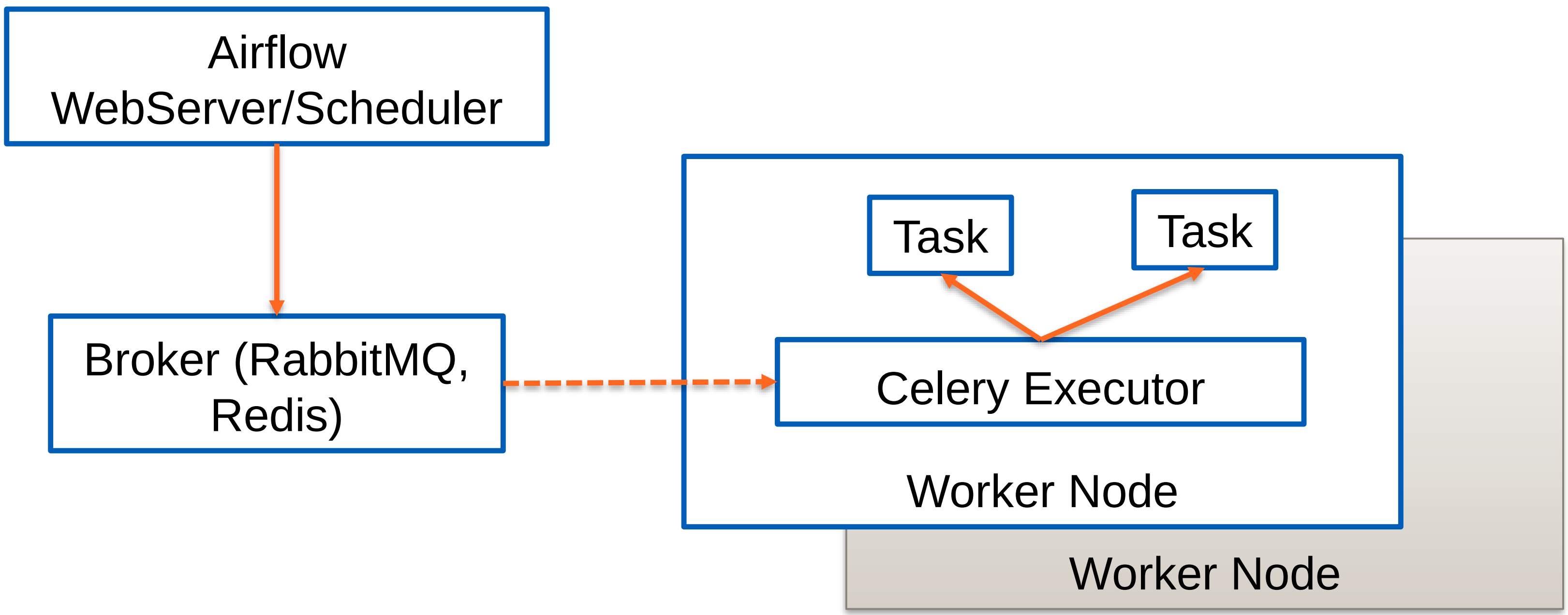


- **Performance metrics can be determined based on states and structures**

- **Interesting in performance analytics?**
  - **Check https://doi.org/10.1016/j.future.2007.01.003**

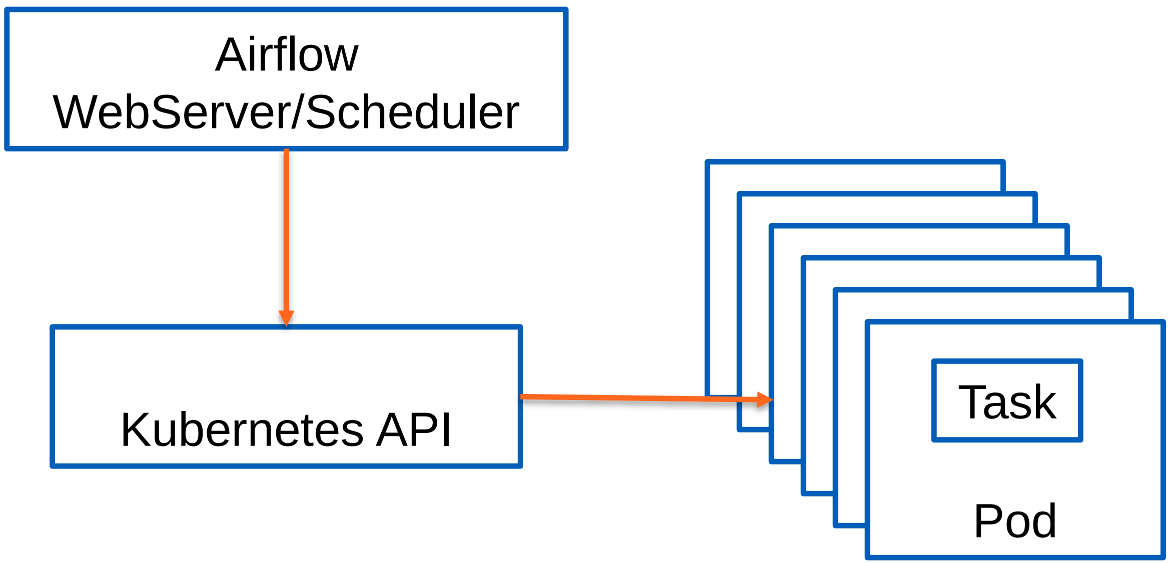
Distributed tasks

You can scale Airflow using workers deployed in different nodes managed by Celery (http://www.celeryproject.org/)



Distributed tasks

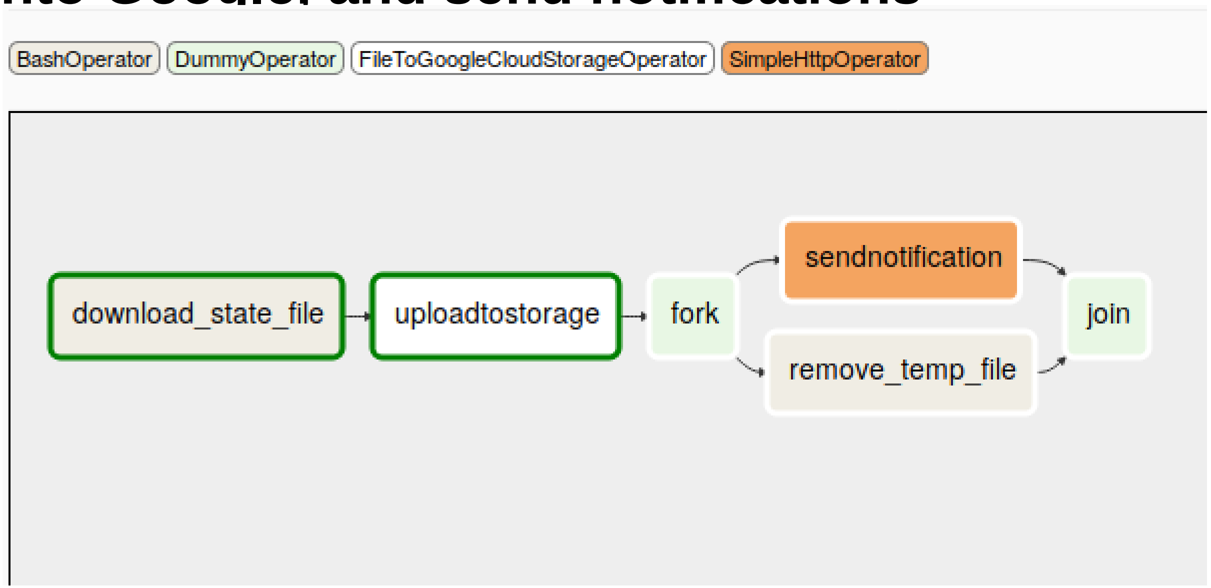
You can scale Airflow to run tasks in Kubernetes



Google Cloud Composer: use Kubernetes

Example

Scenarios: scan various local servers, obtain log files, store log files into Google, and send notifications



Example for uploading state logs

```
fork = DummyOperator(
    task_id='fork',
    trigger_rule='one_success',
    dag=dag
    )
join = DummyOperator(
    task_id='join',
    trigger_rule='one_success',
    dag=dag
    )

t_downloadlogtocloud=  BashOperator(
    task_id="download_state_file",
    bash_command=downloadlogscript,
    dag = dag
    )

t_removefile =  BashOperator(
    task_id='remove_temp_file',
    bash_command=removetempfile,
    dag=dag,
    )
```

```
## change it suitable to your setting
t_analytics=  FileToGoogleCloudStorageOperator(
    task_id="uploadtostorage",
    src=destination_file,
    dst=gcsdir,
    bucket='mybdpairflow',
    google_cloud_storage_conn_id='gcsmybdp',
    dag = dag
    )
## change it suitable for your setting
t_sendresult =SimpleHttpOperator(
    task_id='sendnotification',
    method='POST',
    http_conn_id='notificationserver',
    endpoint='api/logUpdate',
    data=json.dumps({"source_file": source_file}),
    headers={"Content-Type": "application/json"},
    dag = dag
    )
```

In our GIT course (tutorials)

Example for uploading state logs

upstream task

```
'''
the dependencies among tasks
'''
t_downloadlogtocloud >> t_analytics
t_analytics >> fork
fork >> t_sendresult
t_analytics >> fork
fork >> t_removefile
t_removefile >> join
t_sendresult >> join
```

downstream task

Monitoring UI

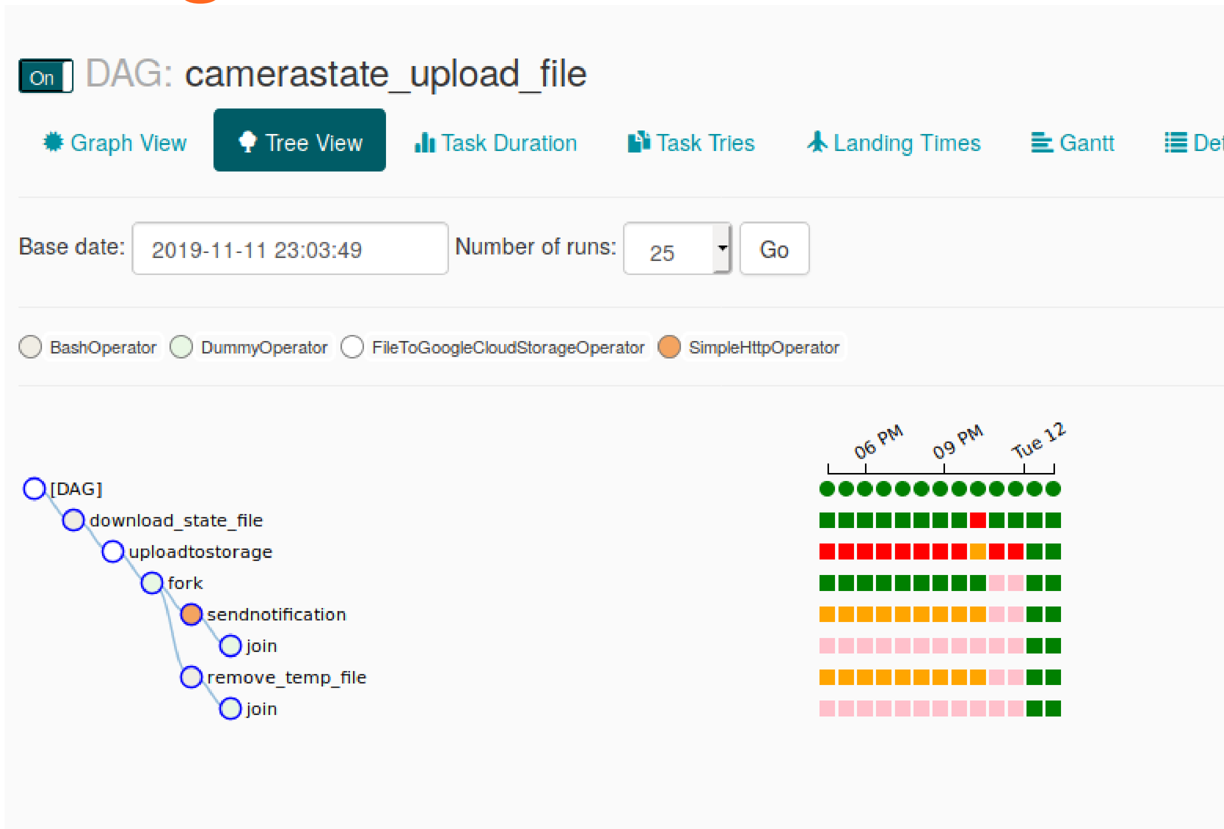
DAGs

Search:

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	On	camerastate_upload_file	None	hong-linh-truong	6	2019-11-11 23:03	13	
	Off	example_bash_operator	0 0 * * *	Airflow	6	2019-11-11 14:47	1	
	Off	example_branch_dop_operator_v3	*/1 * * * *	Airflow				
	Off	example_branch_operator	@daily	Airflow				
	Off	example_http_operator	1 day, 0:00:00	Airflow	6	2019-11-11 14:48	1	
	Off	example_passing_params_via_test_command	*/1 * * * *	airflow				
	Off	example_pig_operator	None	Airflow				
	Off	example_python_operator	None	Airflow				
	Off	example_short_circuit_operator	1 day, 0:00:00	Airflow				
	Off	example_skip_dag	1 day, 0:00:00	Airflow				
	Off	example_subdag_operator	@once	Airflow				
	Off	example_trigger_controller_dag	@once	airflow				
	Off	example_trigger_target_dag	None	Airflow				
	Off	example_xcom	@once	Airflow				
	Off	latest_only	4:00:00	airflow				
	Off	latest_only_with_trigger	4:00:00	airflow				
	Off	test_utils	None	airflow				
	Off	tutorial	1 day, 0:00:00	Airflow				

Showing 1 to 18 of 18 entries

Monitoring UI



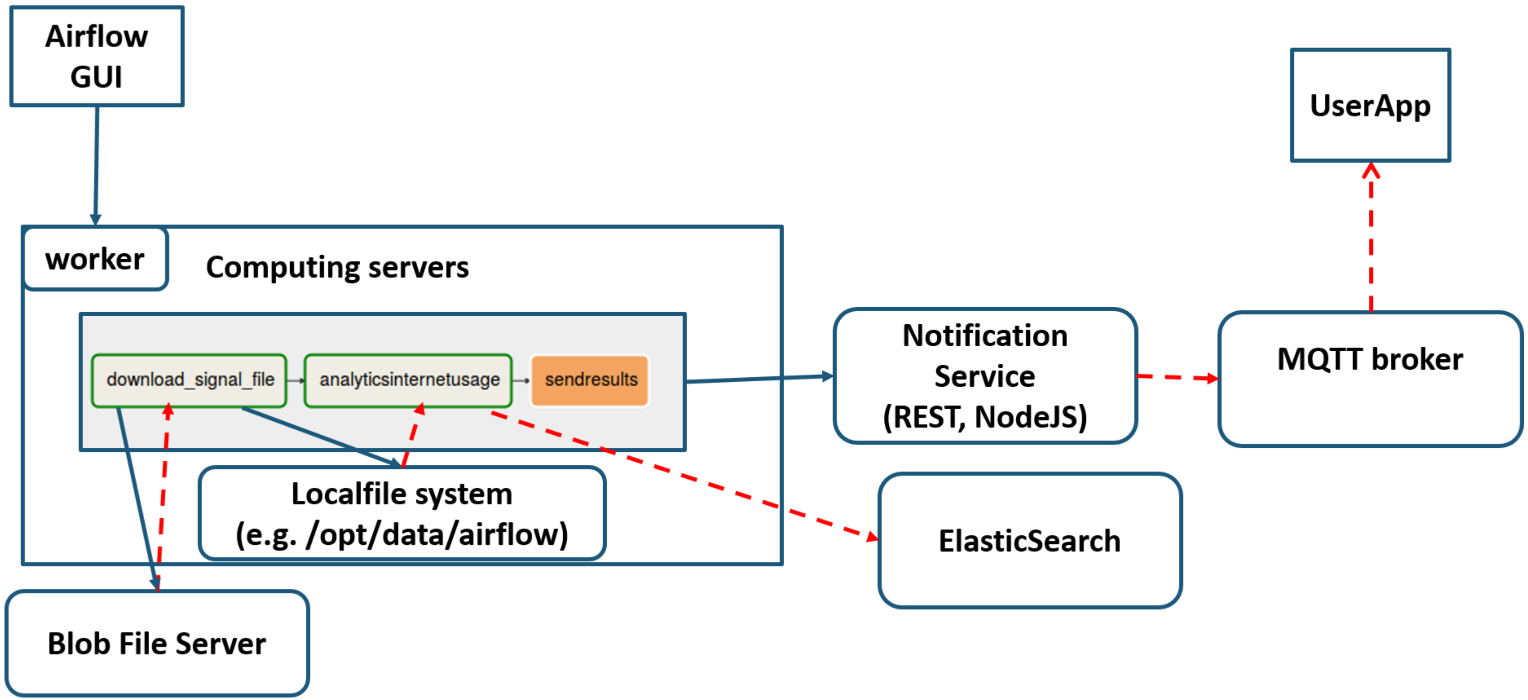
Elasticity control for workflows

- How to scale the workflows?
- Scheduling in a large resource pool (e.g., using clusters)
- Elasticity controls of virtualized resources (VMs/containers/Kubernetes) for executing tasks
- Using distributed task queues, e.g. Celery

Integration with other services in big data platforms

- **Workflows triggers stream analytics ?**
- **Stream analytics triggers serverless functions?**
- **And another way around?**

Example for processing mobile signal data in telcos

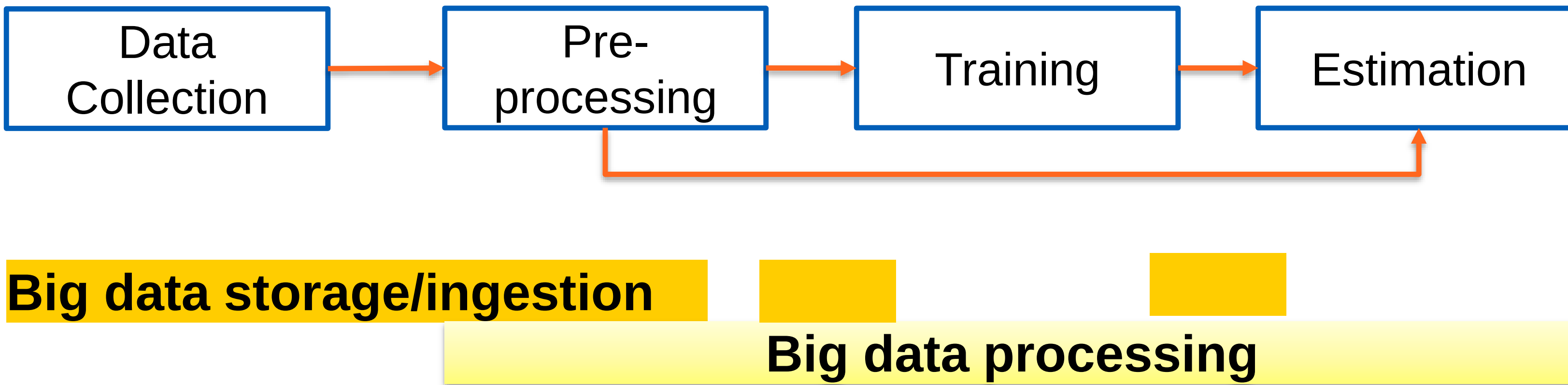


Workflows for machine learning pipelines management

ML pipelines/flows

- **In big data platform we also do need to support many machine learning tasks**
  - machine learning is considering “data processing”
  - but it also requires many other features: data-preparation, data management, experiment management
- **Data in ML pipelines**
  - models, training data, data to be learned!
  - experiment settings and experiment data
  - from the big data platforms viewpoint: they are all data!

Workflows requirements in ML



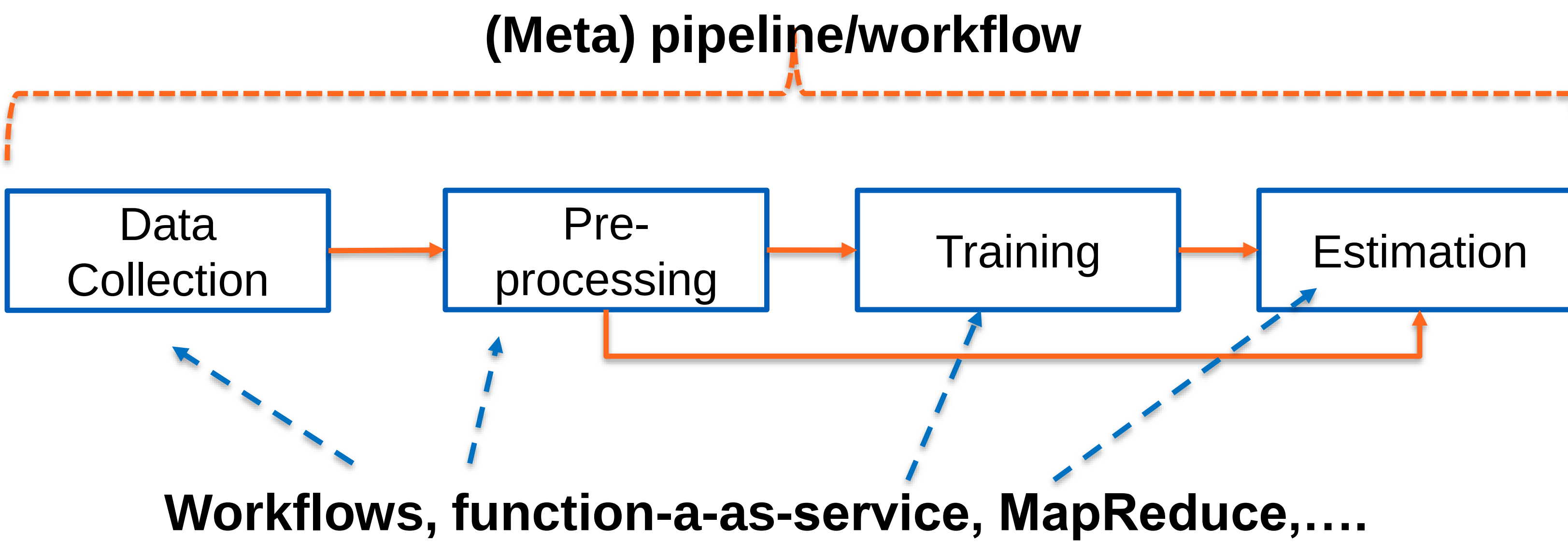
Coordinate these phases (service and data movement)
Experiment management (especially for parameter tuning)

Issues

- **Coordinate different phases in the pipeline**
  - include execution and data movement
  - monitoring and tracking
- **Execution atop multiple computing frameworks suitable for ML**
- **Experiment management**
  - several experiments must be carried out
  - parameter tuning!

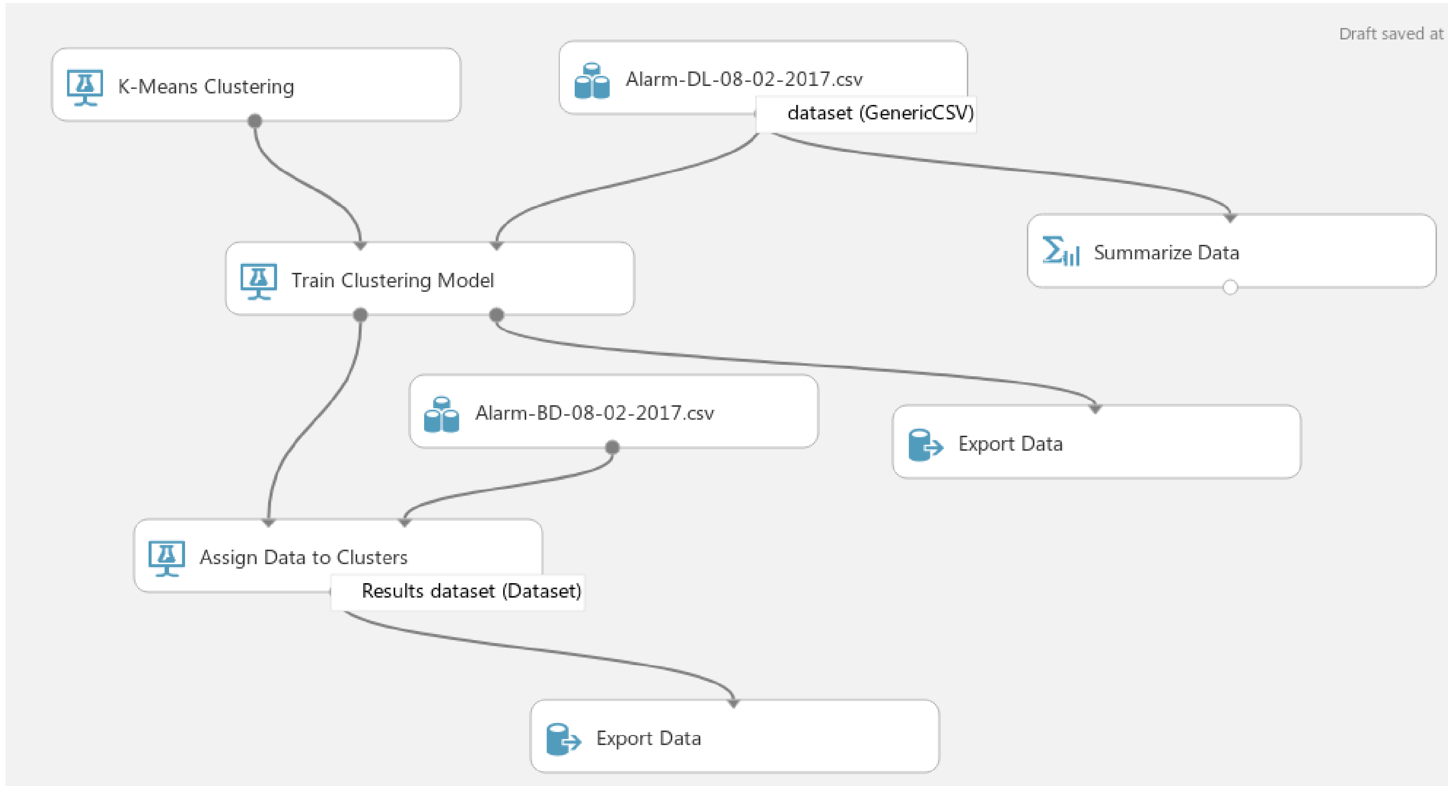
ML workflows

- **Two possible levels:**
  - meta-workflow or pipeline
  - inside each phase: pipeline/workflow or other types of programs



Workflow used in ML pipelines

Azure ML



Other tools

- **KubeFlow (https://www.kubeflow.org/)**
  - build and run ML workflows in Kubernetes
  - end-to-end orchestration of ML workflows
- **MLFlow (https://mlflow.org/)**
  - Projects: capture all utilities, data, etc., for building flows
  - Models: capture machine learning models
  - Pipelines monitoring and experiment management

Summary

- **Facts:**
  - workflows are known techniques
  - widely used in/with big data platforms
  - not just for data analytics: also platform management and integration
- **Thoughts:**
  - need to understand common concepts of workflows
  - generic workflows versus specific workflows (e.g., only for ML)
  - what do we learn from Google Cloud Composer, from a platform provider viewpoint?

Summary

- **Focus:**
  - practical programming with:
    - *Apache Airflow: for data analytics and platform management*
    - *Function-as-a-service: for service integration*
    - *Kubeflow: for machine learning with big data platforms*
- **Action:**
  - hands-on and work on concrete examples
    - *Try to see if you can implement previous use cases/scenarios in your work with workflows*
  - offering workflows as a service in your platform!

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io