丁来强 wjo1212@163.com

Python高效大数据工作流与任务调度

丁来强 (Lai Qiang Ding)

wjo1212

wjo1212@163.com

LaiQiangDing

About Me

- Farther of a 4 years' boy

About Me

- Worked for 10+ years.
- @Splunk

Agenda

- Background
  - Definition
  - Role in Data Infra
- Requirement
  - Problem
  - Challenges
  - Requirement
- Solutions
  - Overview
  - Luigi
  - Airflow
- Demo

You will learn:

- Role of workflow scheduler for data engineering in ecosystem.
- Challenges and key requirements.
- Solutions and general differences.
- Architecture, design and practices of using Airflow and Luigi **in Python**
- Pitfalls and common patterns in design to use a workflow scheduler

Definition

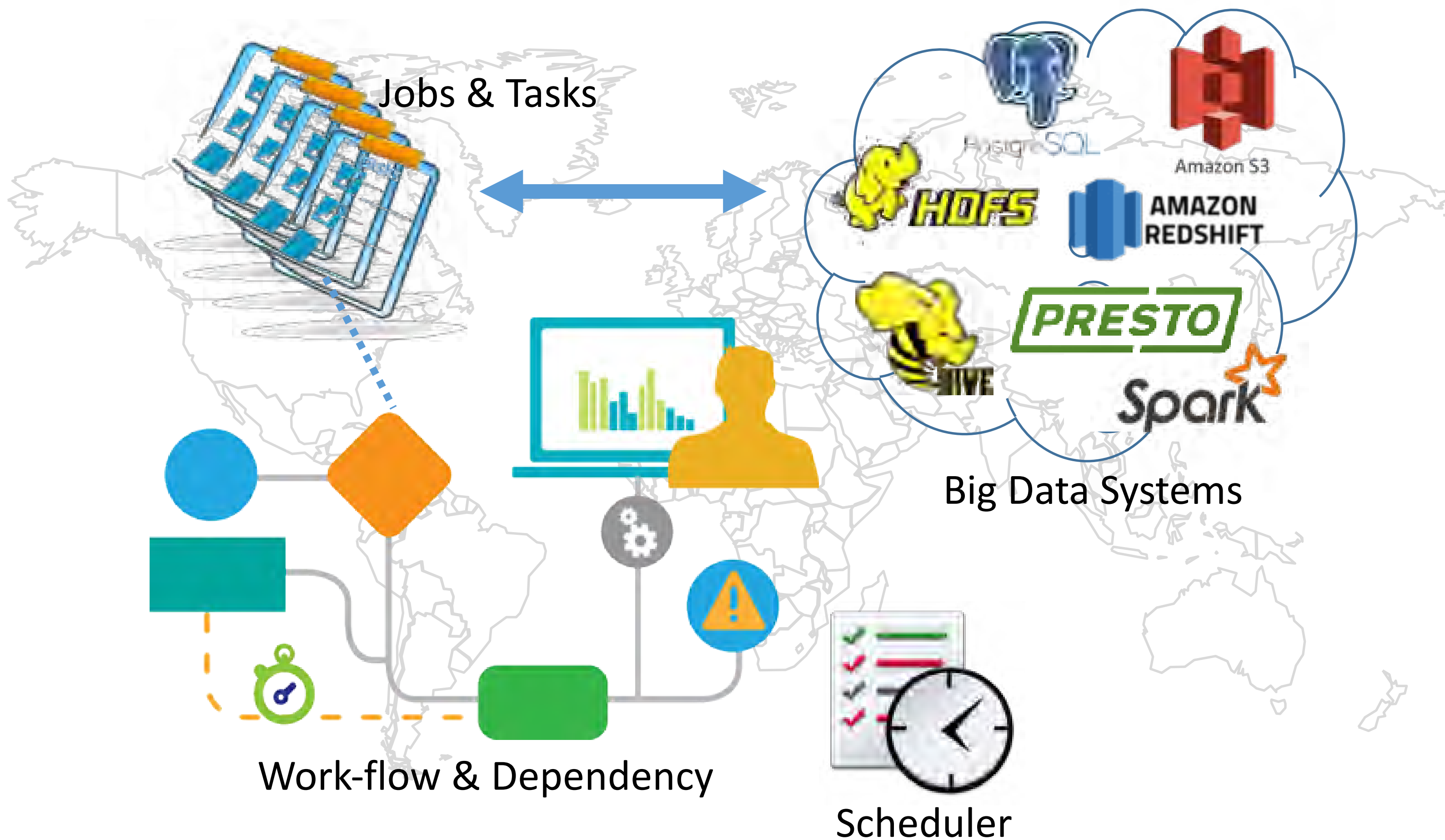
丁来强 wjo1212@163.com

Definition

Big Data Workflow Scheduler

Schedule and manage dependencies of workflow of jobs in data infrastructure, mainly used in offline and near-line system.

Big Data Work-flow Scheduler



丁来强 wjo1212@163.com

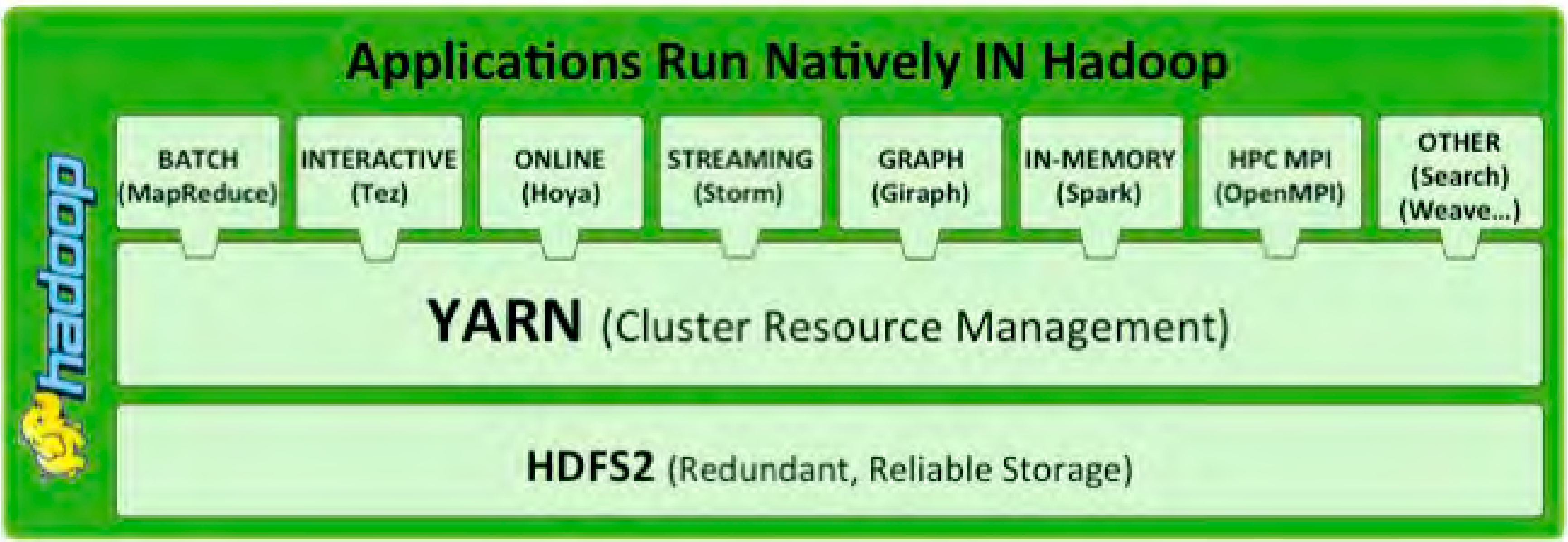
- Different with below categories:

- BPM
  - Like Activiti
- Middleware workflow & SOA
  - Like AWS Simple Workflow
- Pure Data Driven Pipeline/API for Development
  - Like Apache Crunch, Apache Cascading, AWS Data Pipeline, Azure Data Factory
- Pure Streaming Process
  - Like Storm, Spark Streaming

Role in Data Infra

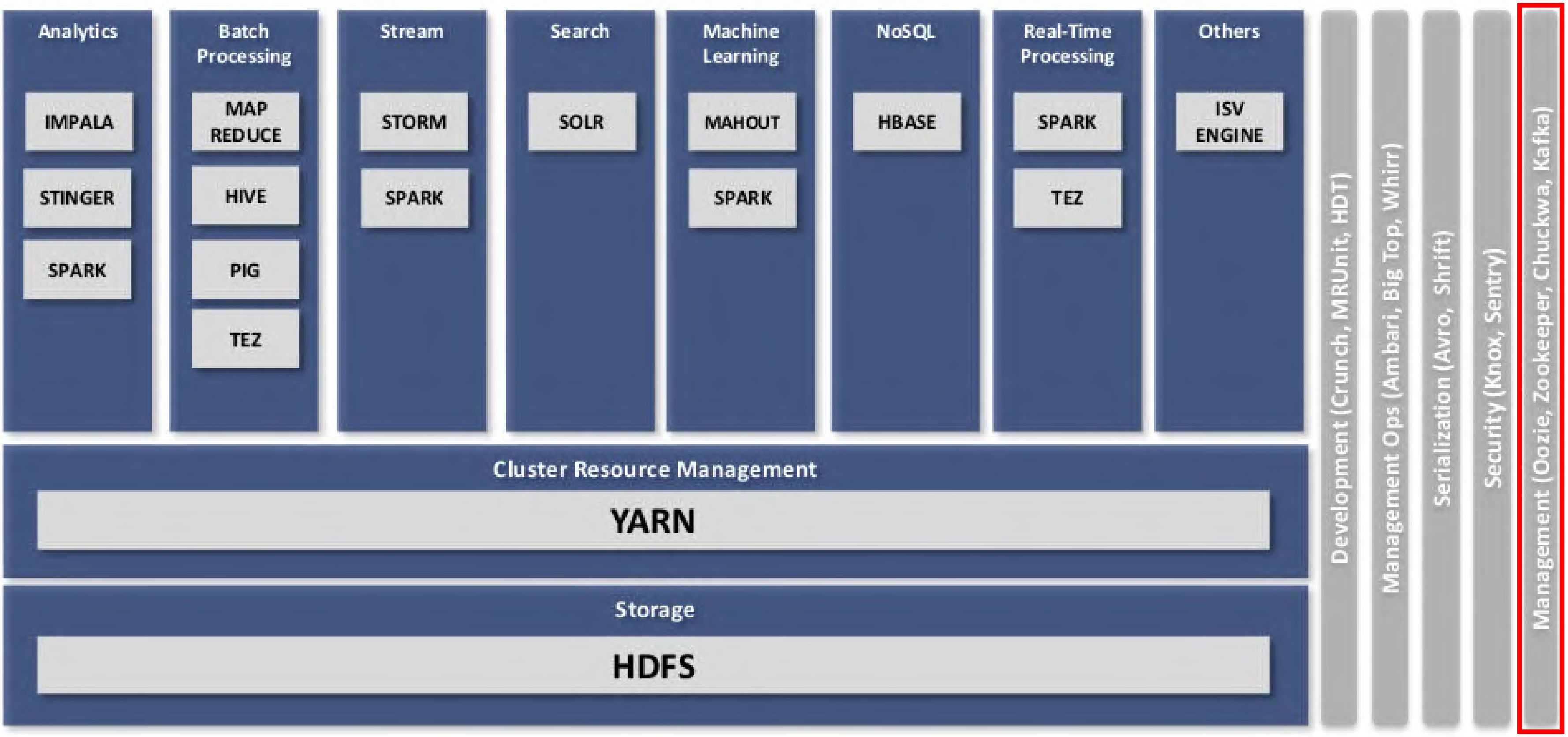
丁来强 wjo1212@163.com

- Hadoop 2.0



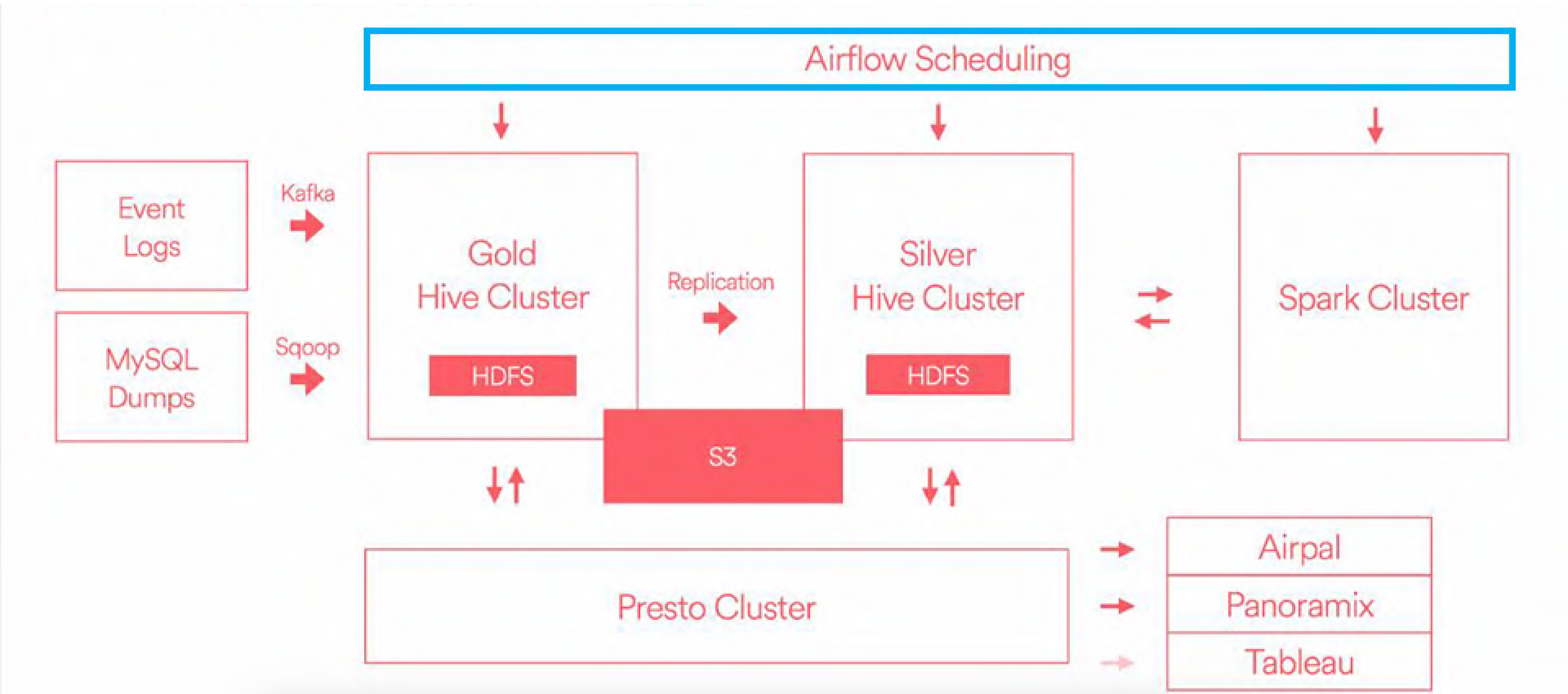
Applications Run Natively IN Hadoop
BATCH (MapReduce)
INTERACTIVE (Tez)
ONLINE (Hoya)
STREAMING (Storm)
GRAPH (Giraph)
IN-MEMORY (Spark)
HPC MPI (OpenMPI)
OTHER (Search) (Weave...)
YARN (Cluster Resource Management)
HDFS2 (Redundant, Reliable Storage)
hadoop

Hadoop 2.0



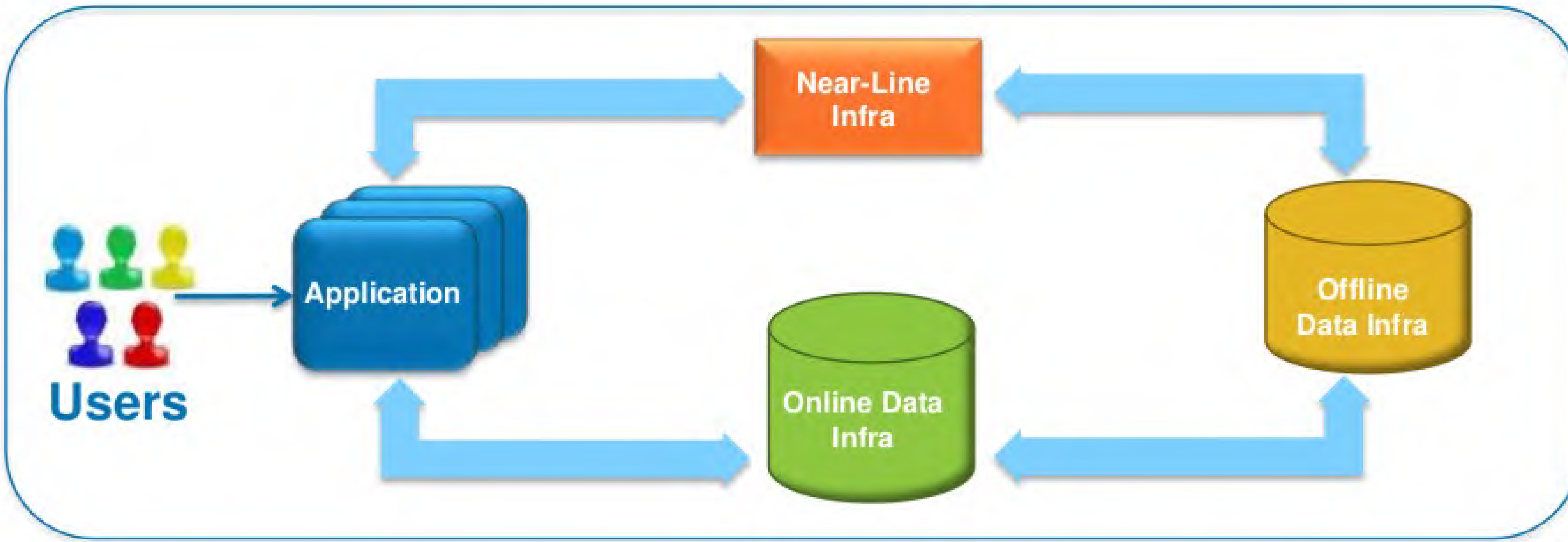
丁来强 wjo1212@163.com

Airbnb Data Infra



丁来强 wjo1212@163.com

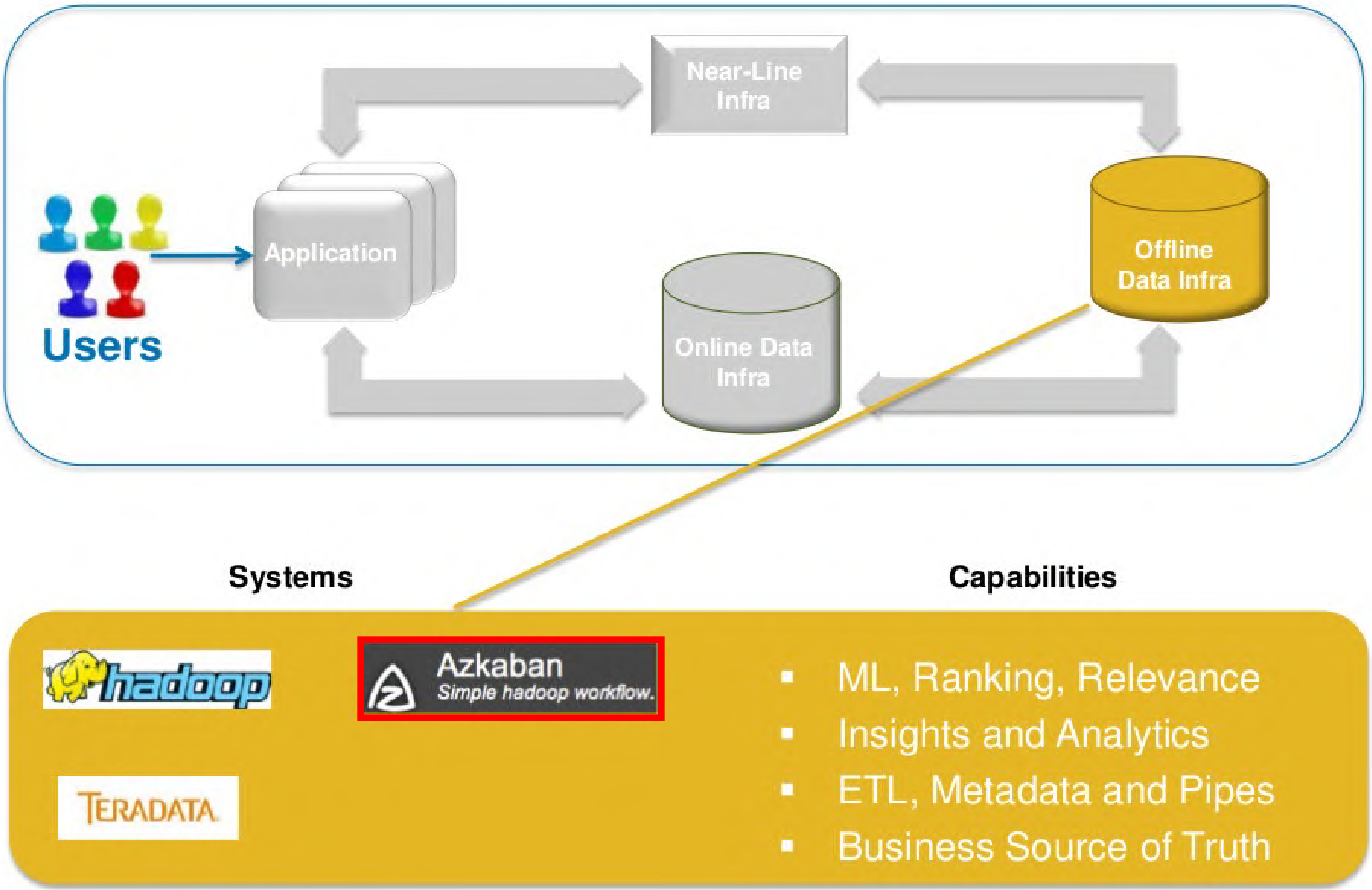
LinkedIn Data Infra



Infrastructure	Latency & Freshness Requirements	Products
Online	Activity that should be reflected **immediately**	• Member Profiles • Company Profiles • Connections • Messages • Endorsements • Skills
Near-Line	Activity that should be reflected **soon**	• Activity Streams • Profile Standardization • News • Recommendations • Search • Messages
Offline	Activity that can be reflected **later**	• People You May Know • Connection Strength • News • Recommendations • Next best idea...

http://www.slideshare.net/amywtang/linkedin-bigdata-yaleoct2012final

LinkedIn Data Infra



http://www.slideshare.net/amywtang/linkedin-bigdata-yaleoct2012final

Data of workflow scheduler in Big Data

airbnb

- 14 boxes dedicated for work-flow system
- 8,000 tasks daily

Airflow

LinkedIn

- Maintain 3 instances of work-flow system
- 2,500 flows, 30,000 jobs daily

Azkaban

Spotify

- 2000+ tasks, 10,000+ Hadoop jobs daily

Luigi

What’s the most important for a Big data workflow scheduler ?

丁来强 wjo1212@163.com

Dead Simple:

- Easy to use and configure

Problems with big data job scheduling

丁来强 wjo1212@163.com

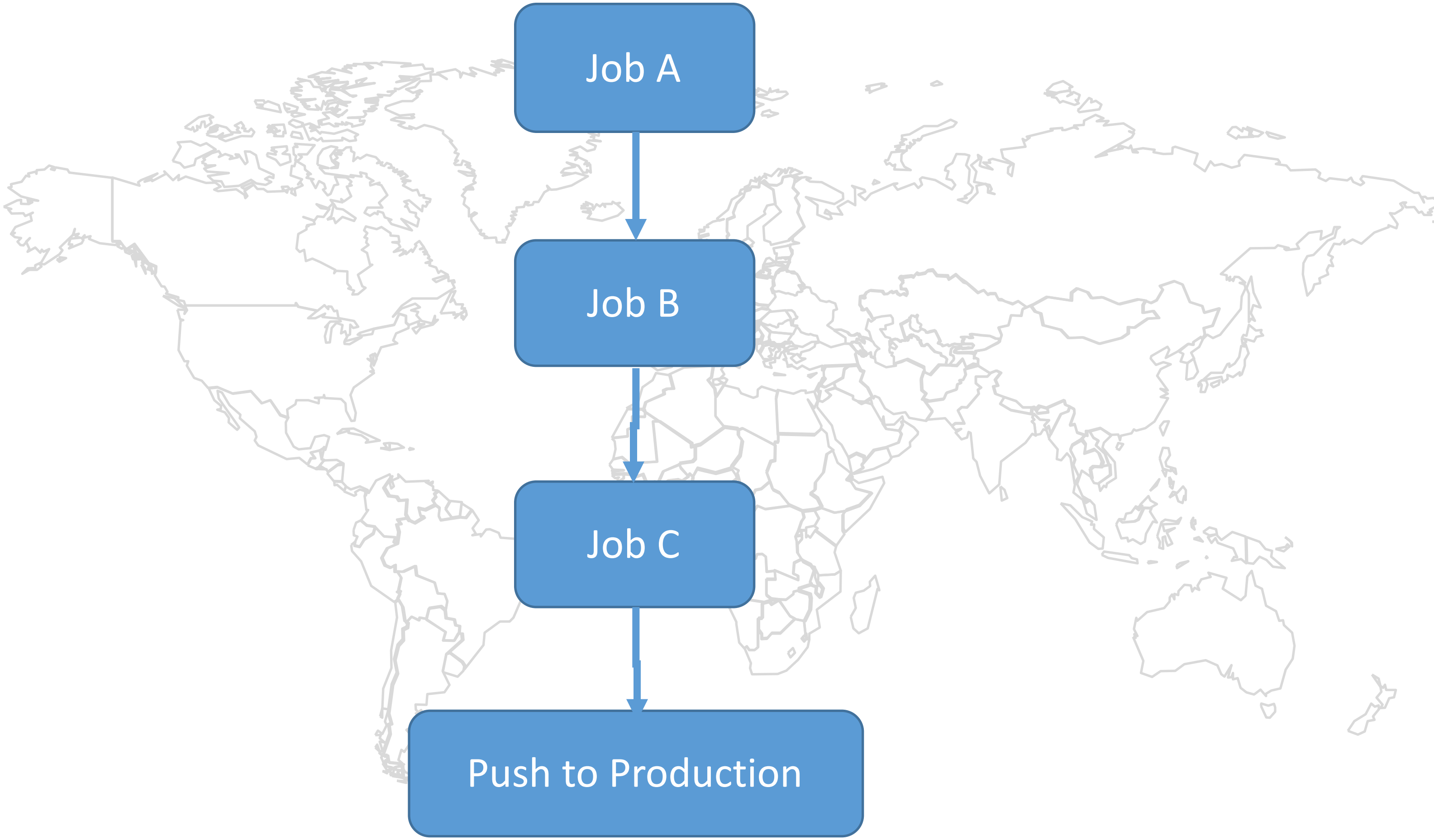
Typical Challenge

- 数据工作流程复杂度越来越高
- 数据分析与批处理数据非常重要
- 大量时间花费在编写任务、检测与排错上

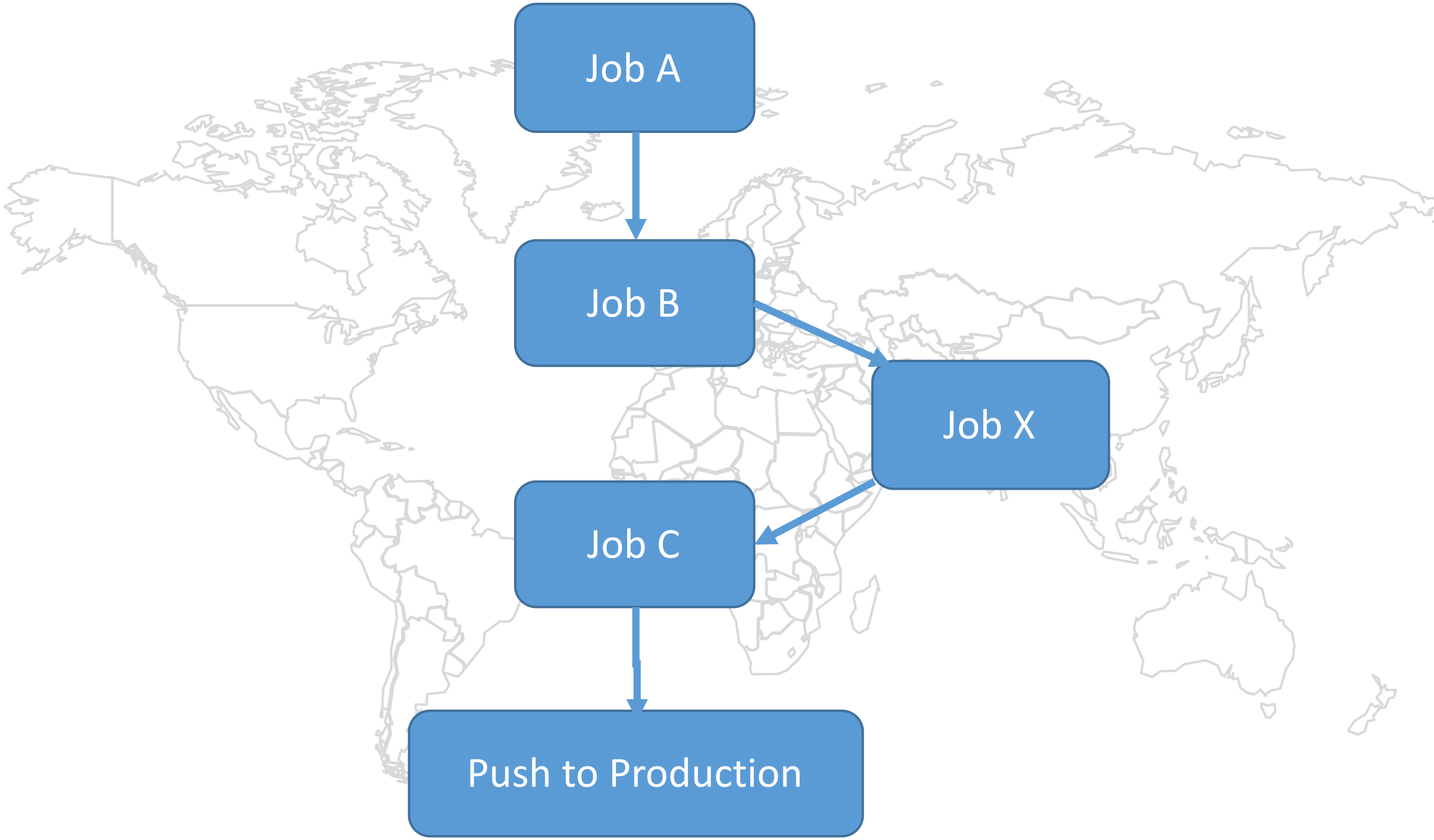
Fragile process

丁来强 wjo1212@163.com

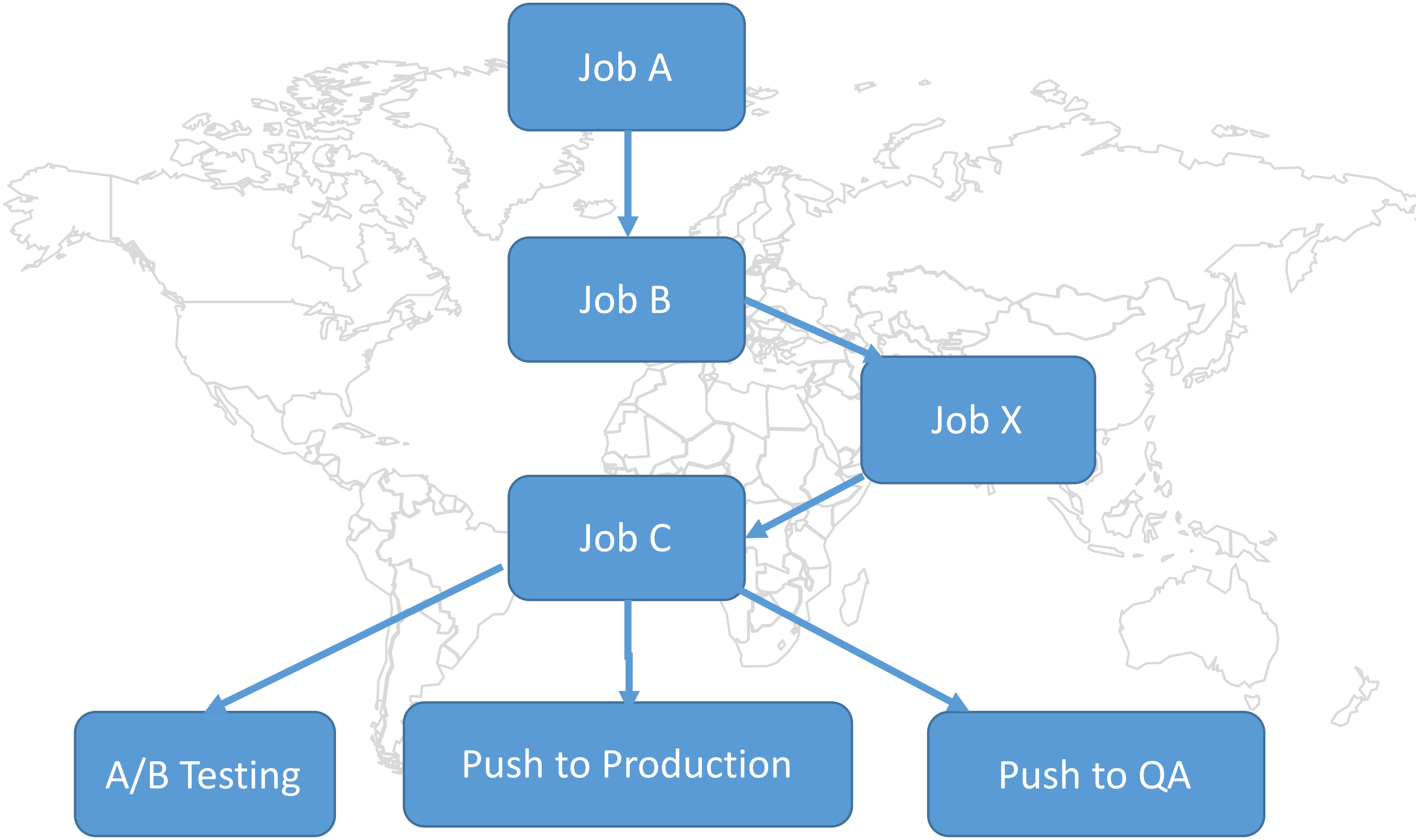
Fragile process



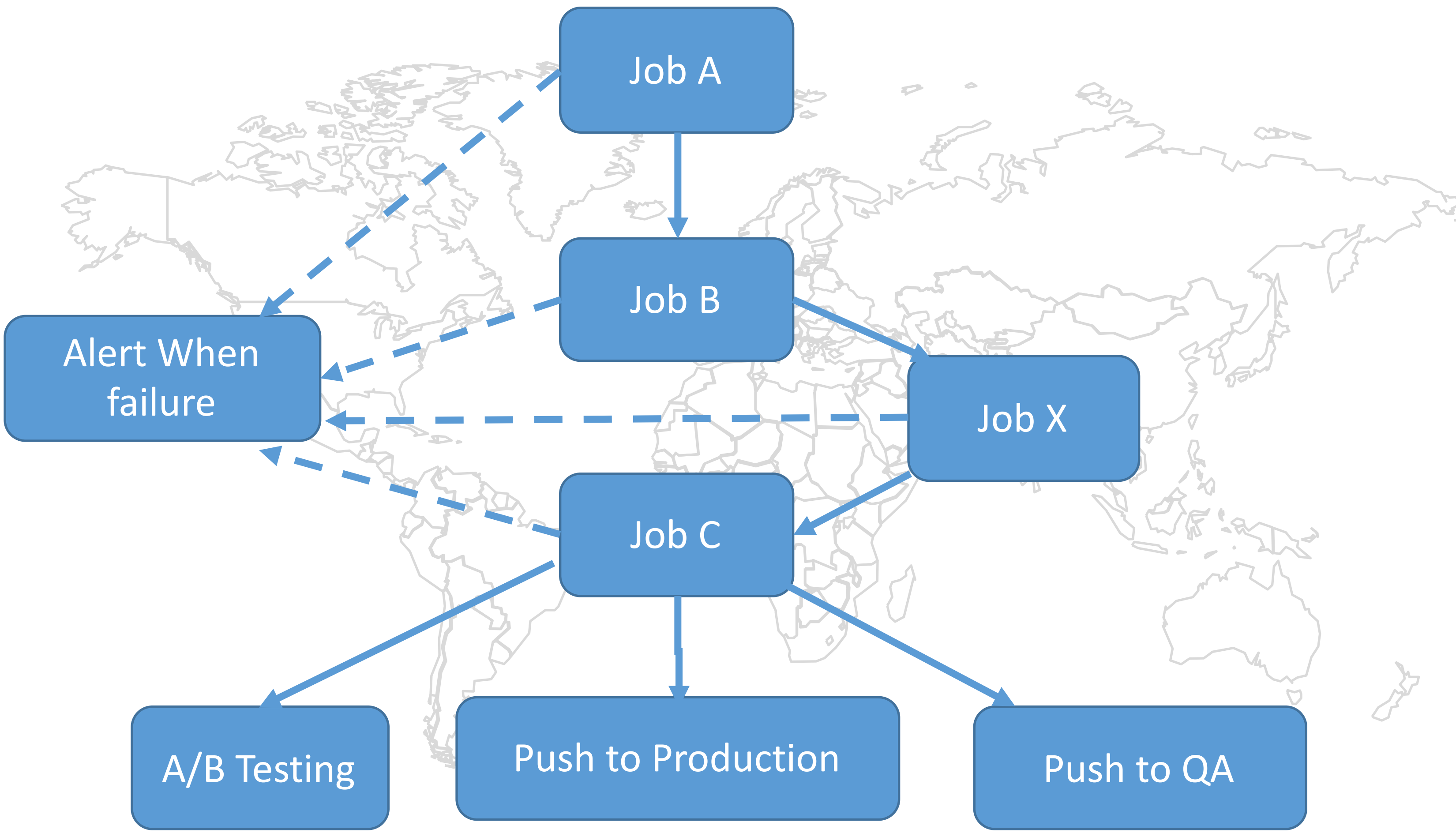
Fragile process



• Fragile process

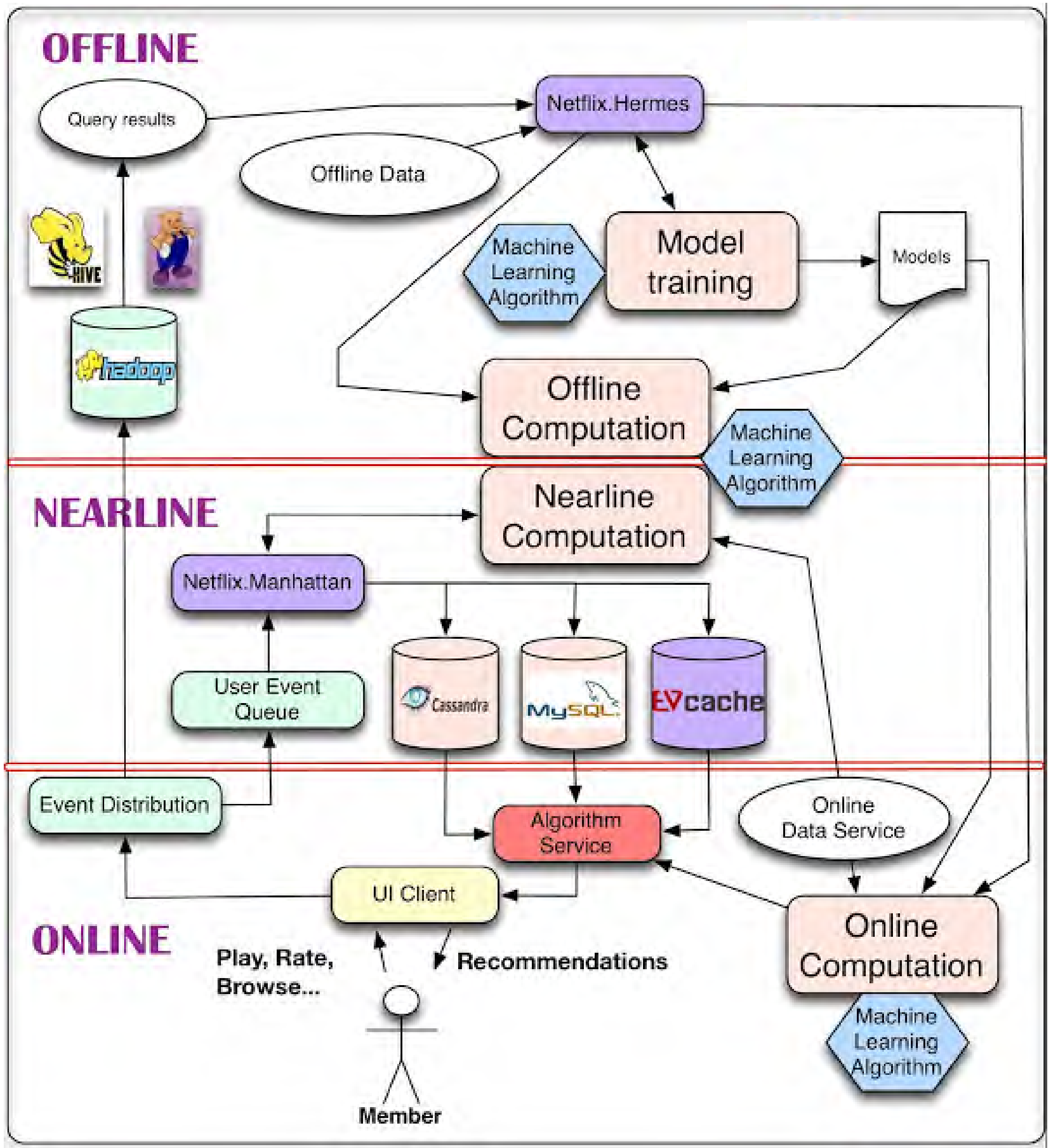


Fragile process



丁来强 wjo1212@163.com

Example: Netflix Recommendation System

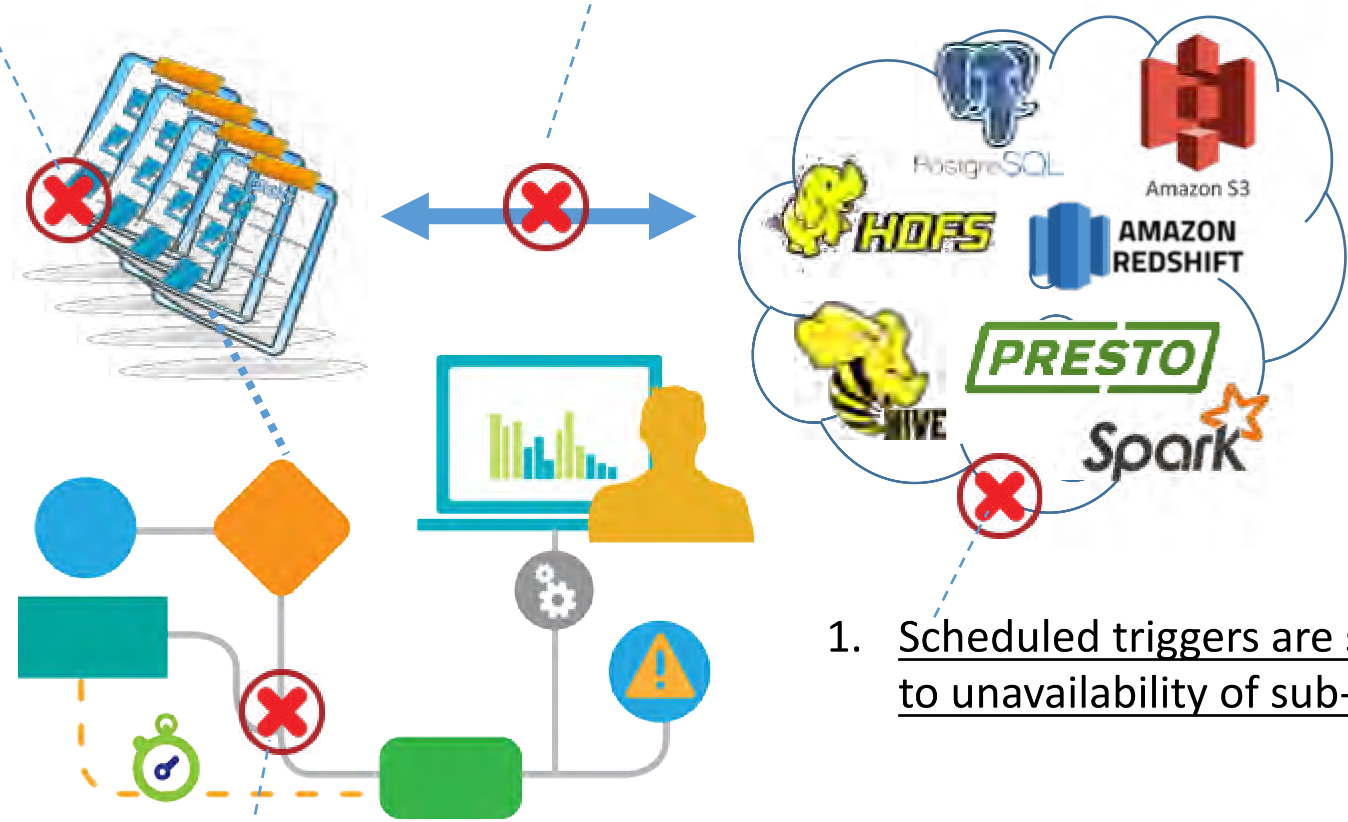


Fragile Failure Handing

丁来强 wjo1212@163.com

3. Some errors or bugs may exist in some jobs' logic

2. Job fails due to system or network may not be temporarily not available



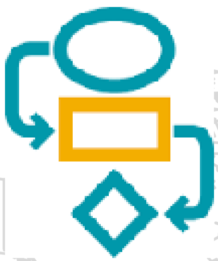
1. Scheduled triggers are skipped due to unavailability of sub-system

4. Performance is slow especially for some critical steps

Requirement

丁来强 wjo1212@163.com

Basic Needs



Work-flow & Dependency

Job Definition

Calendar Based Scheduling

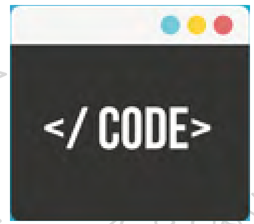
Log Access & Monitoring

Notification

Failure Tolerance & Backfill

Advanced Needs

Complex Rule



Programmatic

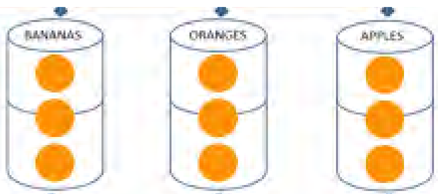
Operator OOB

Scalability

High Availability

SLA Monitor & Alert

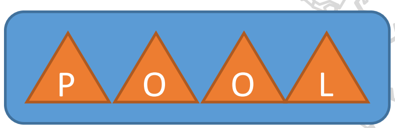
Advanced Needs (cont’)



Queue (Affinity)

Data Profiling

Versioning



Pool (Limit concurrency + priority)

Event Driven Scheduler

Plugins

Solution Overview

丁来强 wjo1212@163.com

Options

Airflow

Azkaban

Luigi

OOZIE

Chronos

cron

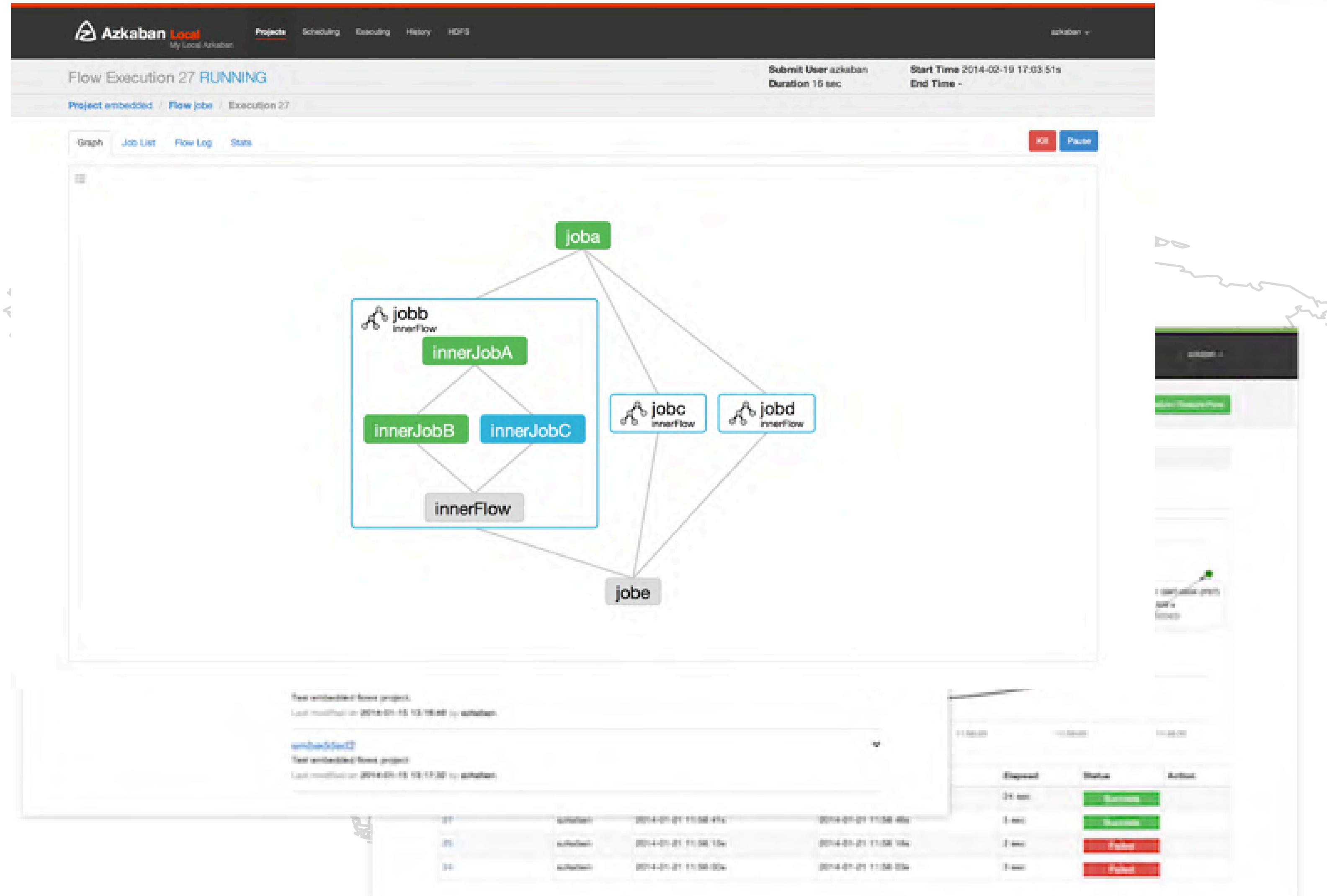
Solution Overview

Basic Info	Luigi	Airflow	Azkaban	Oozie
Language	Python	Python	Java	Java
Github Stars	5,274	3,422	780	354
Contributors	256	178	37	18
Latest Version	2.3.1	1.7.1	3.1	4.2
History	4 years	1+ years	6+ years	6+ years
Invented by	Spotify	Airbnb	LinkedIn	Yahoo
Owned by	Spotify	Apache Incubator	Apache	Apache

Azkaban

- Pros:
  - Born for Hadoop
    - Support all Hadoop, hive, pig versions
  - Easy to use Web UI:
    - Good Job visualization and monitoring
  - Flexible Module structure/Plugins
- Cons:
  - **Properties files based configuration**
  - Web UI only, No CLI and REST interfaces (need 3rd party AzkabanCLI)
  - Limited execution path control

● Azkaban GUI



Azkaban Local
Projects
Scheduling
Executing
History
HDFS
Flow Execution 27 RUNNING
Submit User azkaban
Duration 16 sec
Start Time 2014-02-19 17:03 51s
End Time -
Project embedded / Flow jobe / Execution 27
Graph
Job List
Flow Log
Stats
Kill
Pause
joba
jobb
innerFlow
innerJobA
innerJobB
innerJobC
innerFlow
jobc
innerFlow
jobd
innerFlow
jobe

Oozie

- Pros:
  - Born for Hadoop
  - CLI, HTTP, JAVA API interfaces
  - Support extended Alert integration
- Cons:
  - **Higher learning curve**
  - PDL style XML based configuration
  - Limited Web UI (need Cloudera Hue)
  - No resource control

Luigi

丁来强 wjo1212@163.com

Overview

- Pros:
  - Programmatic by Python
  - Modeling is simple, Code is mature (~20K LOC)
  - Good support Hadoop (MR, logs, dist)
  - Test friendly, support local scheduler
- Cons:
  - Web UI is very limited
  - No built-in trigger (need cron)
  - Not design for large scaling (> 100K tasks)
  - No support distribution of execution

Task Definition

```python
class X(luigi.Task):
    def requires(self): ...
    def run(self): ...
    def output(self): ...
```

Setup Dependencies:
Return one or more Tasks

Logic

Output of the Task:
Return one or more Targets

丁来强 wjo1212@163.com

- # Task Example

```python
import luigi

class MyTask(luigi.Task):
    param = luigi.Parameter(default=42)

    def requires(self):
        return SomeOtherTask(self.param)

    def run(self):
        f = self.output().open('w')
        print >>f, "hello, world"
        f.close()

    def output(self):
        return luigi.LocalTarget('/tmp/foo/bar-%s.txt' % self.param)

if __name__ == '__main__':
    luigi.run()
```

The business logic of the task

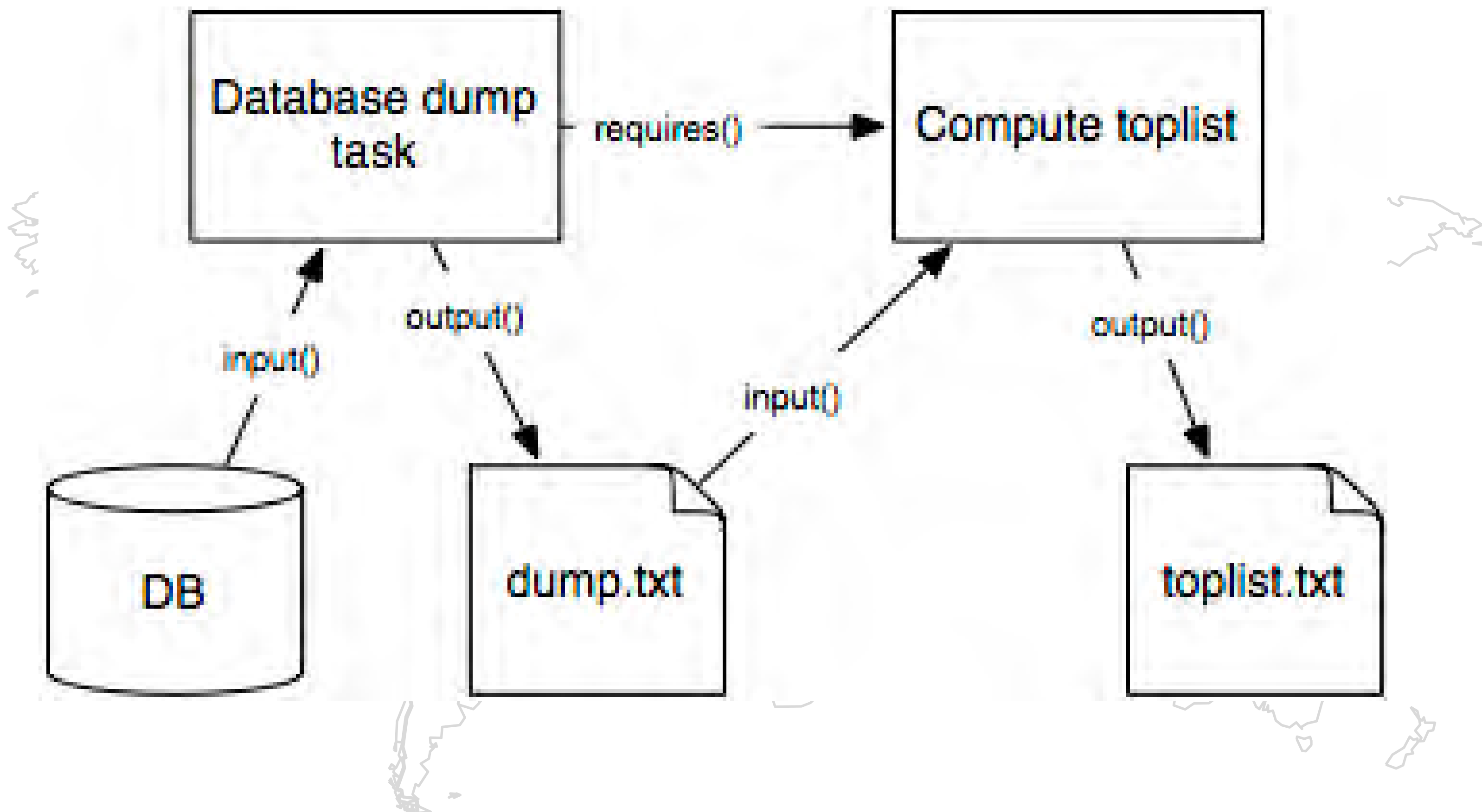
Where it writes output

What other tasks it depends on

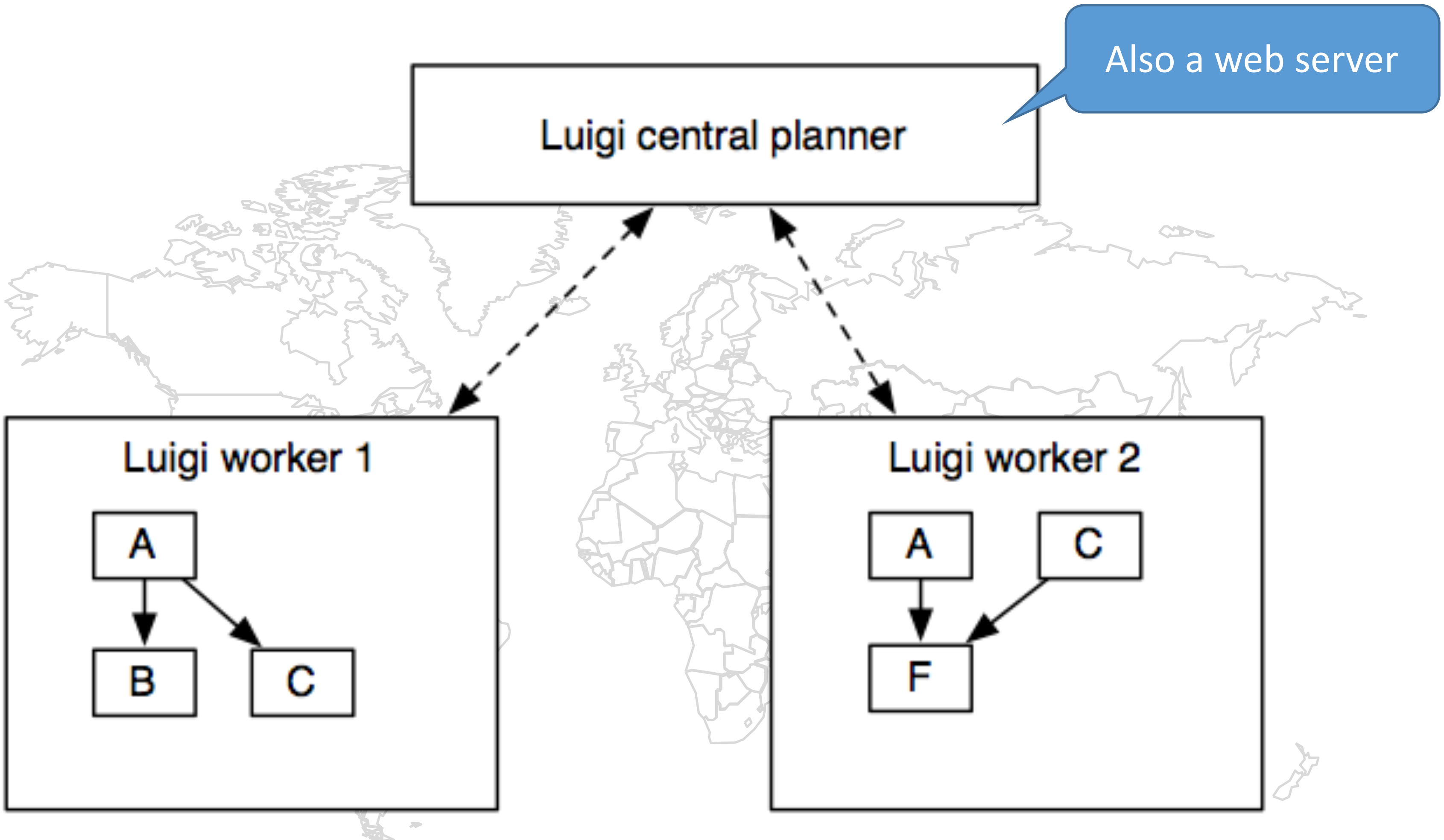
Parameters for this task

丁来强 wjo1212@163.com

Task Execution



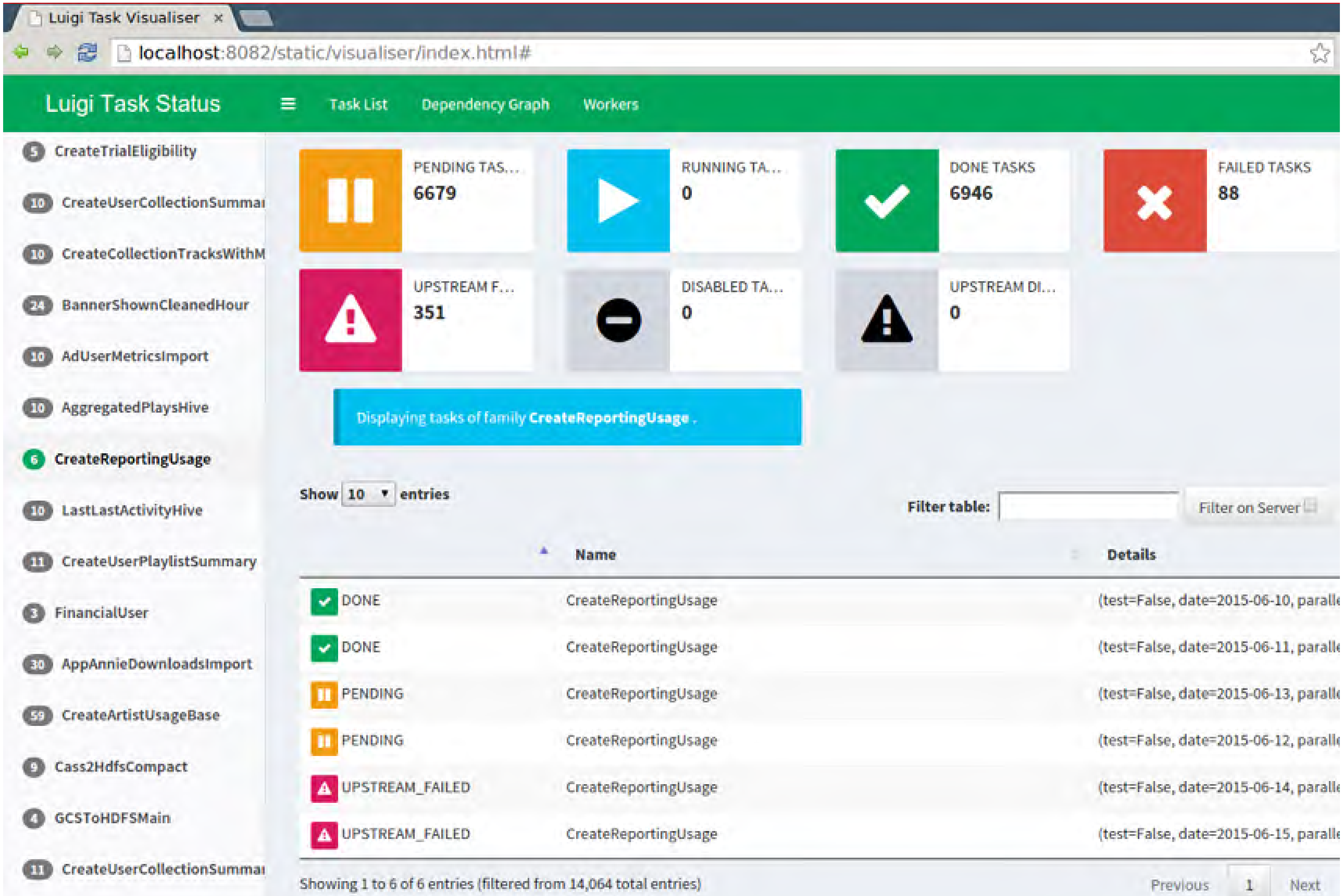
Architecture



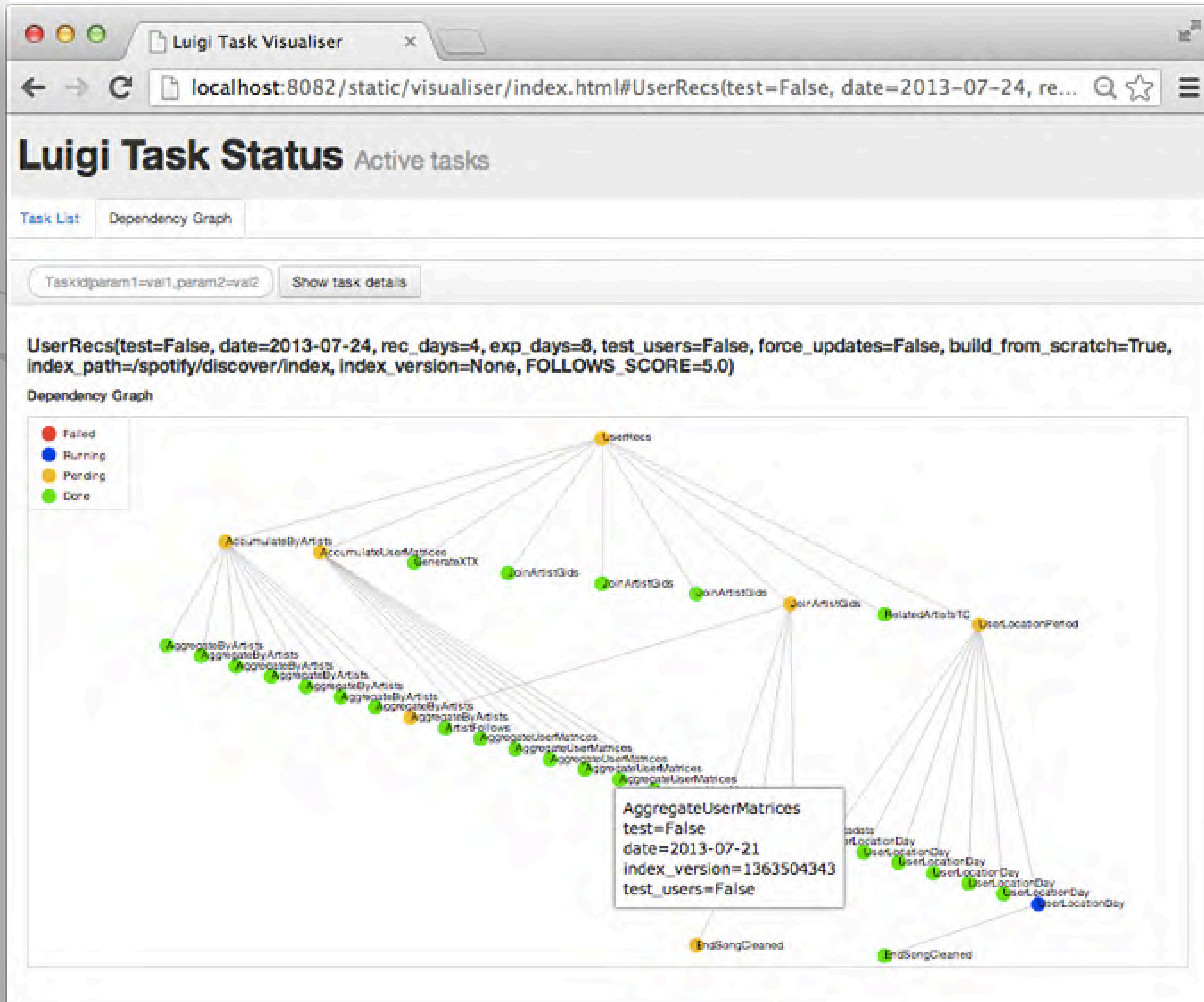
Architecture Notes

- Mainly manage the dependency and de-dup the task running.
- Mainly focus on data pipeline ETL.
- Limitations
  - No calendar trigger
  - Web UI is very limited
  - Too couple between worker and scheduler (not support > 100K tasks)
  - Execution is bundled on specific worker

• Web UI – execution status



- Web UI – DAG visualization



丁来强 wjo1212@163.com

Task and Targets Library

- Google Bigquery
- Hadoop jobs
- Hive queries
- Pig queries
- Scalding jobs
- Spark jobs
- Postgresql, Redshift, Mysql tables
- and more…

Airflow

丁来强 wjo1212@163.com

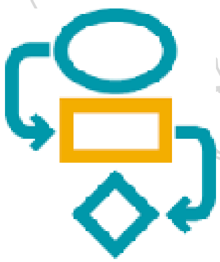
Overview

- Pros (we will see):
  - More General Flexible Architecture
  - Very compelling Web UI
  - Lots of cool features OOB, Rich Operator library
  - Fast growing adoption (30+ companies)
  - Test friendly (test mode and SequentialScheduler)
- Cons:
  - Coding quality is not so mature (UT coverage is not high)
  - No event driven scheduler (same to all others solutions)

Airflow Tech Stack

- Python Code (< 20K LOC)
- DB: SqlAlchemy
- Celery for distributed execution
- Web Server: Flask / gunicorn
- UI: d3.js / Highcharts / Pandas
- Templating: Jinjia2

Airflow Web



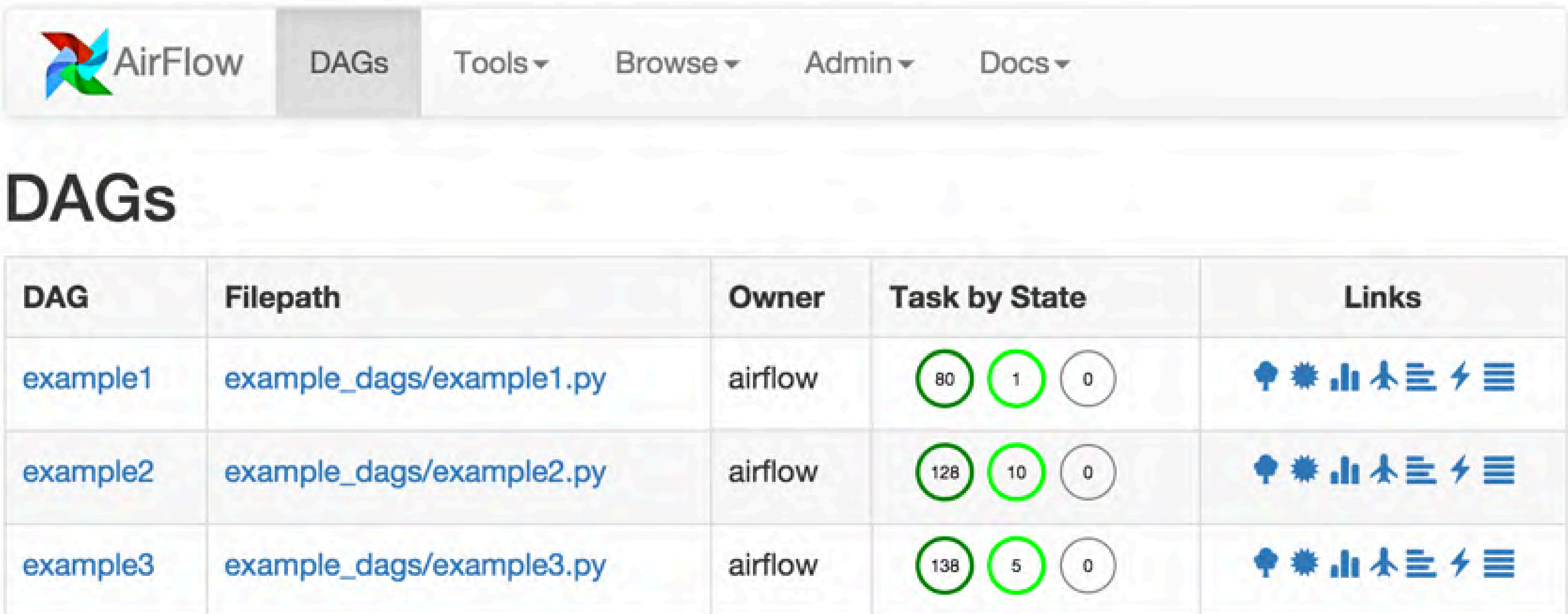
Work-flow & Dependency

Log Access & Monitoring

Data Profiling

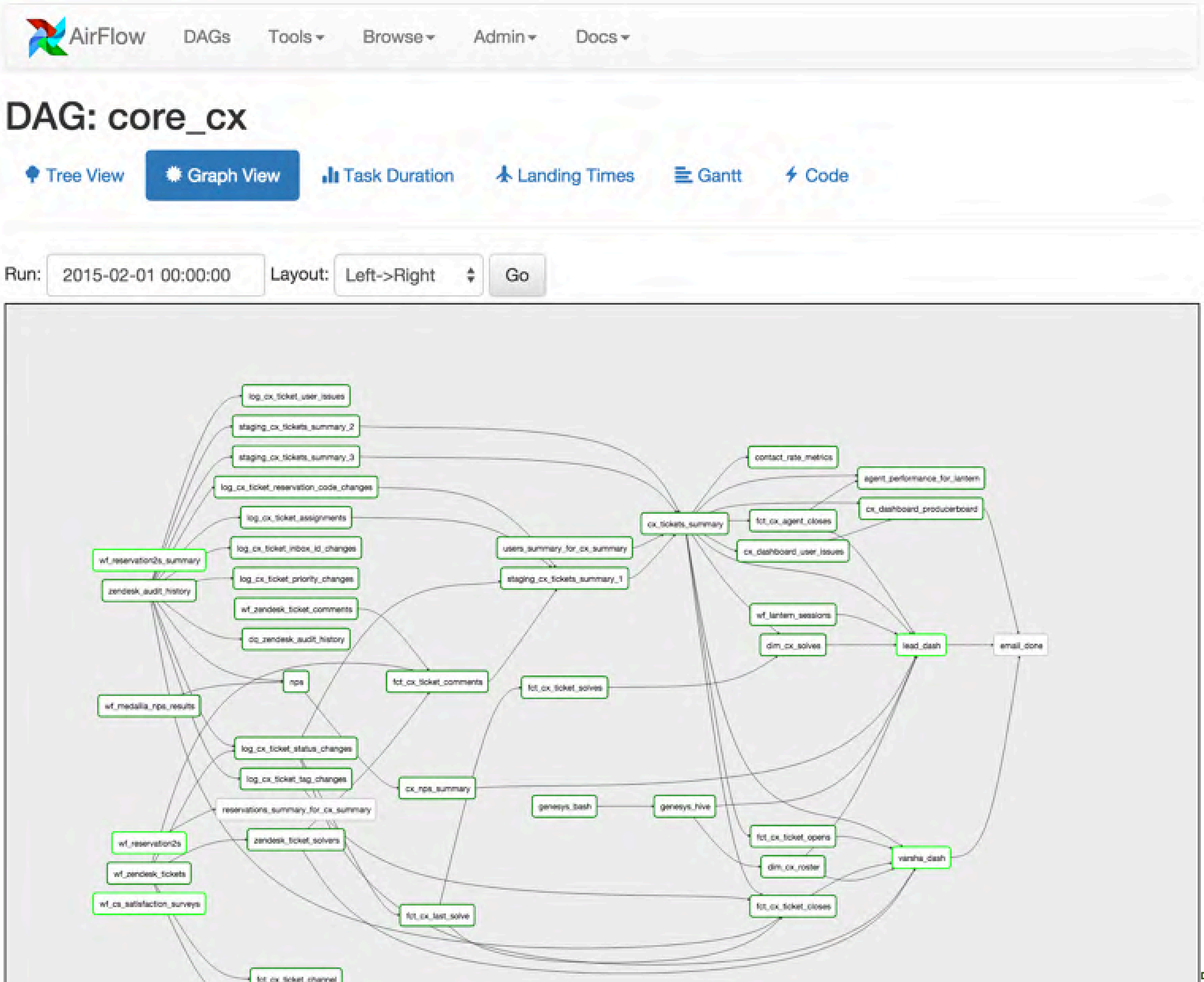
丁来强 wjo1212@163.com

Web UI – Overall status

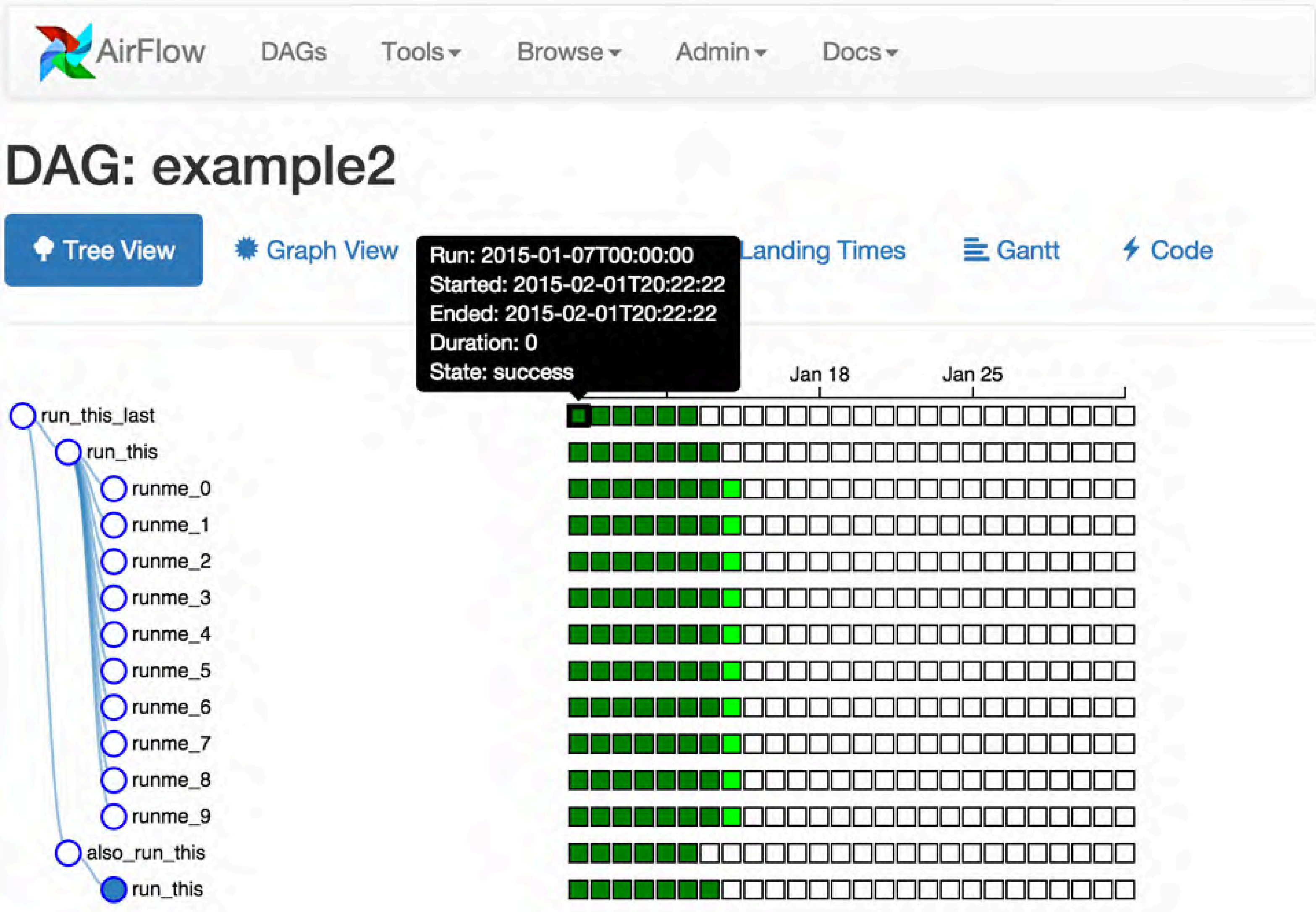


DAG	Filepath	Owner	Task by State			Links
example1	example_dags/example1.py	airflow	80	1	0	
example2	example_dags/example2.py	airflow	128	10	0	
example3	example_dags/example3.py	airflow	138	5	0	

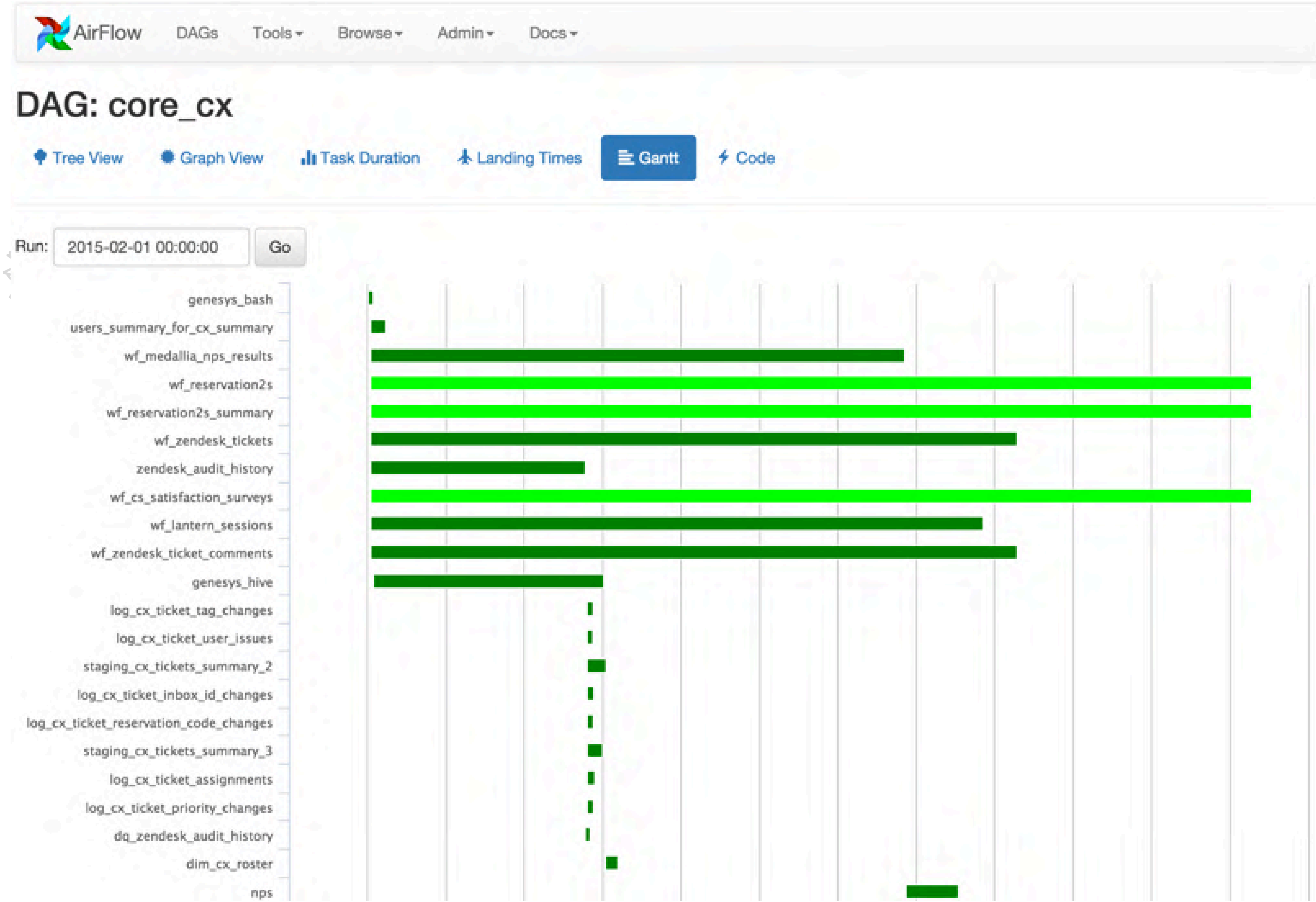
• Web UI – workflow visualization



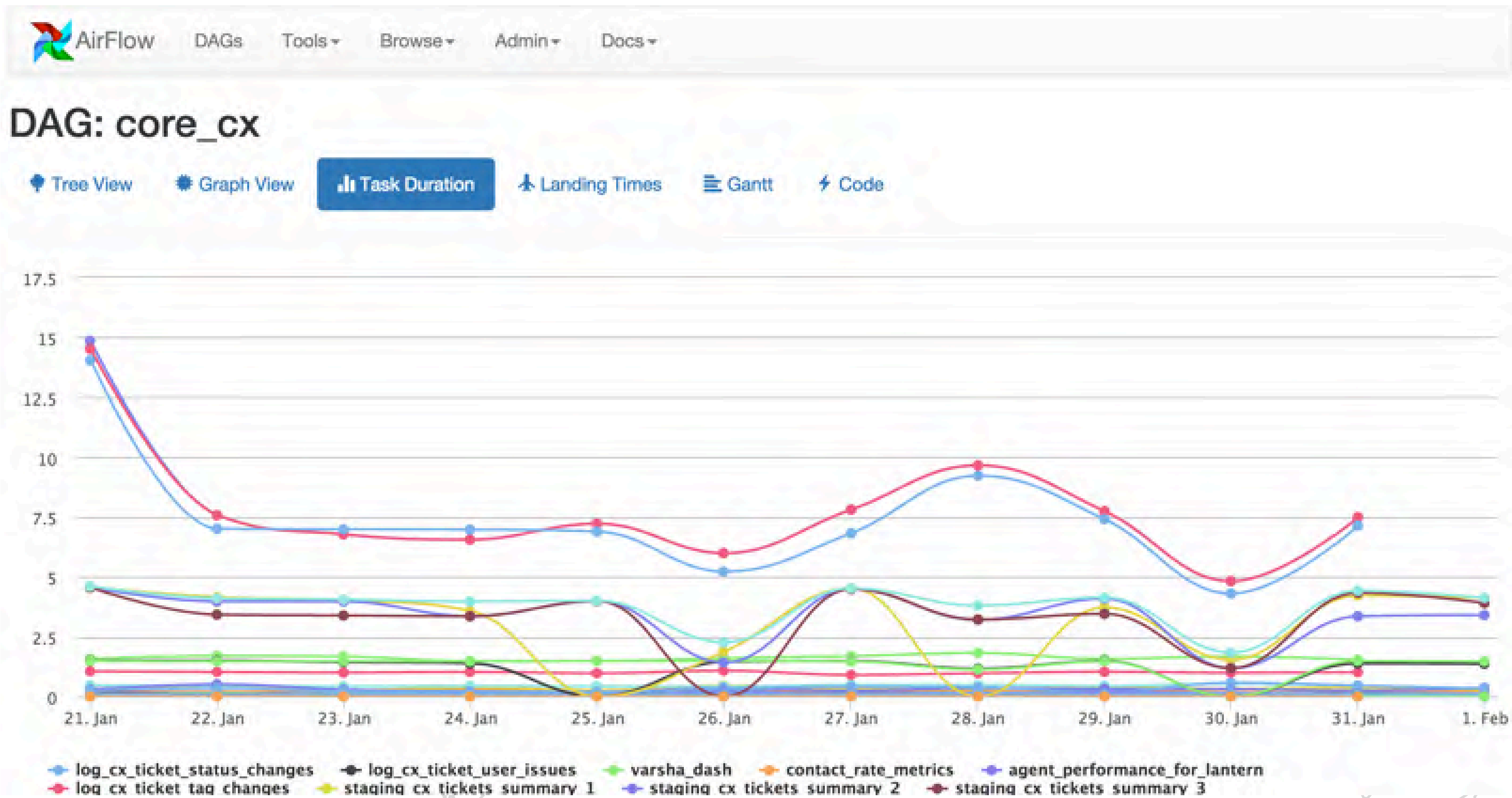
• Web UI – execution history



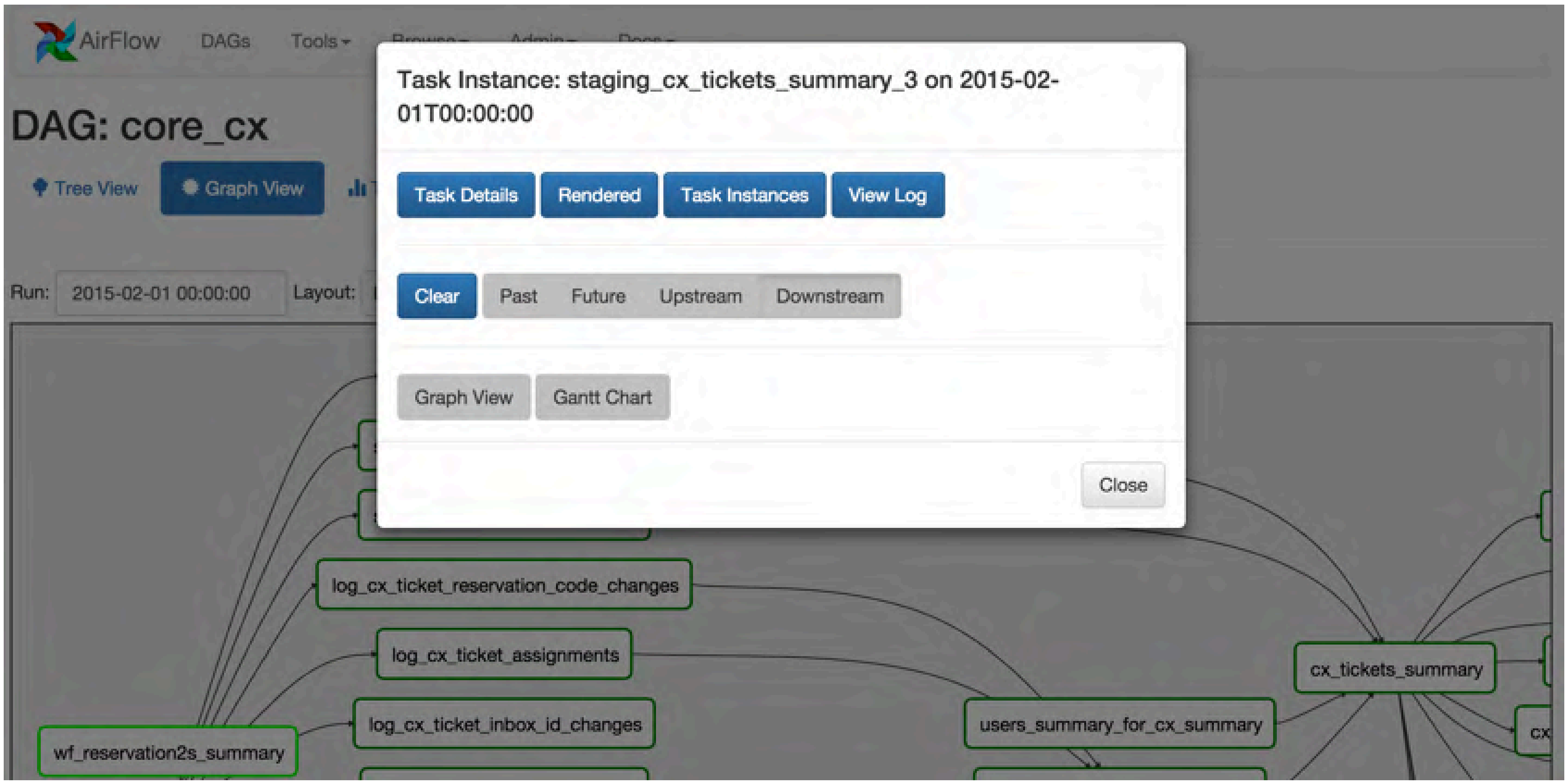
Web UI – performance profile



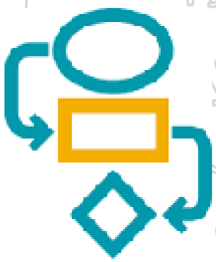
• Web UI – Performance stats over time



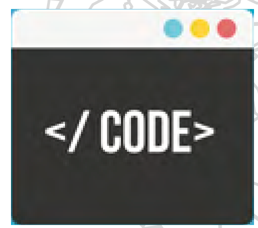
● Web UI – Deep dive for task execution



Airflow Concepts



Work-flow & Dependency



Programmatic

Operator OOB

丁来强 wjo1212@163.com

DAG (Directed Acyclic Graph)

```python
from airflow.operators import PythonOperator
from airflow.models import DAG

dag = DAG(
    dag_id='mydag',
    default_args={ 'owner': 'airflow' },
    schedule_interval='0 0 * * *',
    dagrun_timeout=timedelta(minutes=60))

task1 = BashOperator(
    task_id='mydag_task1',
    bash_command='echo "{{ run_id }}" && sleep 1',
    dag=dag)


def python_logic(param1): pass

task2 = PythonOperator(
    task_id='mydag_task2',
    python_callable=python_logic,
    op_kwargs={'param1': 10},
    dag=dag)

task1.set_upstream(task2)
```

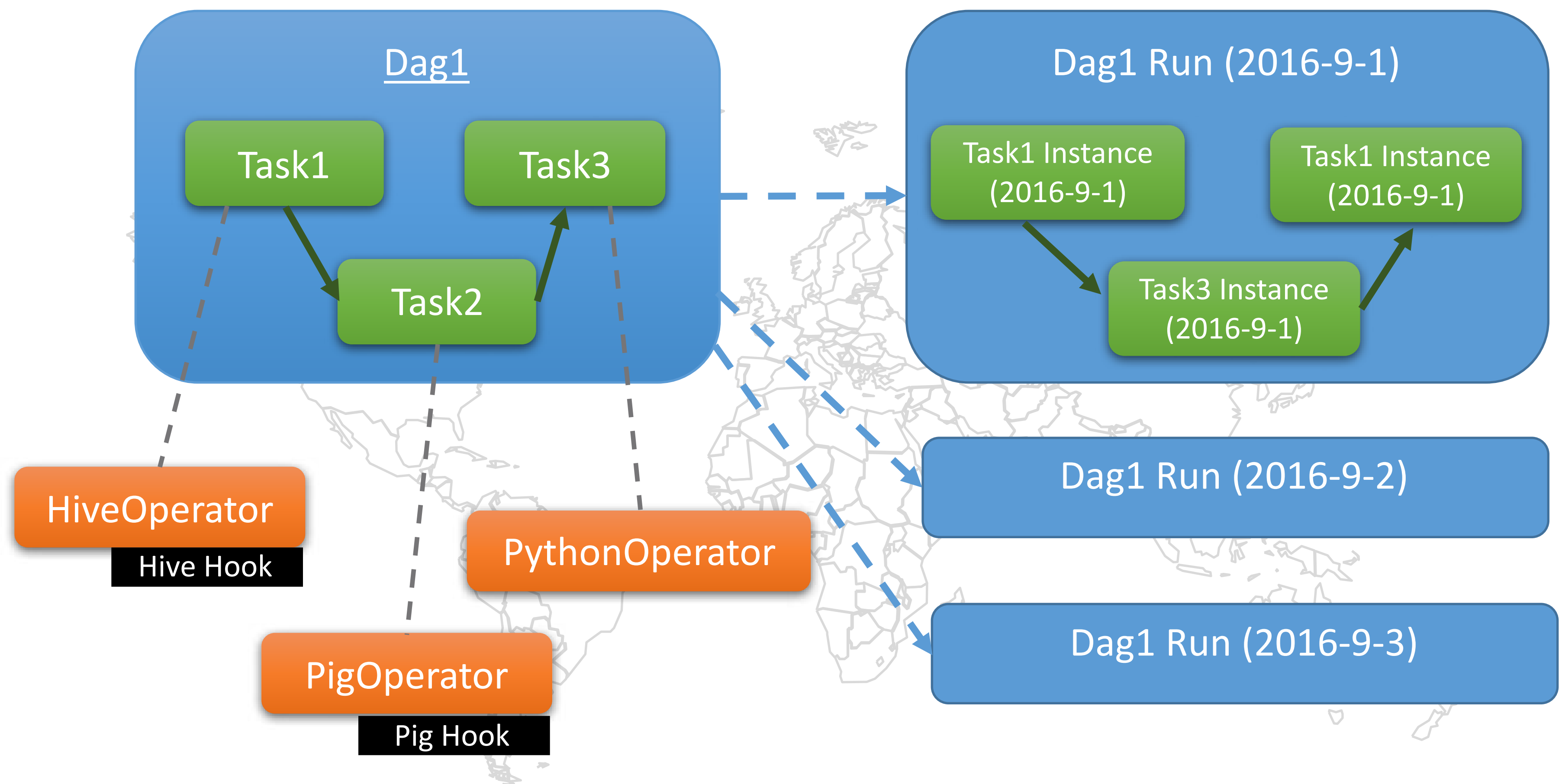
DAG: a collection of tasks w/ scheduling settings

Task: an instance of BashOperator
Support templating

An task of another kind of PythonOperator

Setup the dependencies

DAG execution



Concepts – DAG, DAG Run

- DAG
  - A collection of Tasks
  - Setting of Calendar Scheduling
- Dag Run
  - A run instance of DAG with a scheduled date (ID: dag, start time and interval)

Concepts – Operator, Task and TI

- Operator
  - Task templates
- Task
  - Instance of a Operator
- Task Instance (TI)
  - Belong to Dag Run
  - A run instance of a Task with a scheduled date (id: dag, task, start time and interval)

Concepts - Operator

- Operator
  - Task templates, general categories:
    - Sensor
    - Branching
    - Transformer
  - Settings of Trigger Rules, retry etc.
  - Use Hook for real operation w/ external systems

Operator Library

- Google Bigquery, Could Storage
- AWS S3, EMR
- Spark SQL
- Docker
- Presto
- Sqoop
- Hive jobs
- Vertica
- Qubole
- SSH
- Hipchat, Slack, Email
- Postgresql, Redshift, Mysql, Oracle etc.
- and more…

丁来强 wjo1212@163.com

Parameterized Tasks

- Variables
  - Global parameters
- Connections
  - External system's connection string, confidential, extra parameters etc. Normally used by Hook.
- DAG Parameters/Macros
- Templating
  - Using Jinjia for batch or any places that fit
- Xcom
  - Share data between Tasks

Architecture Insight

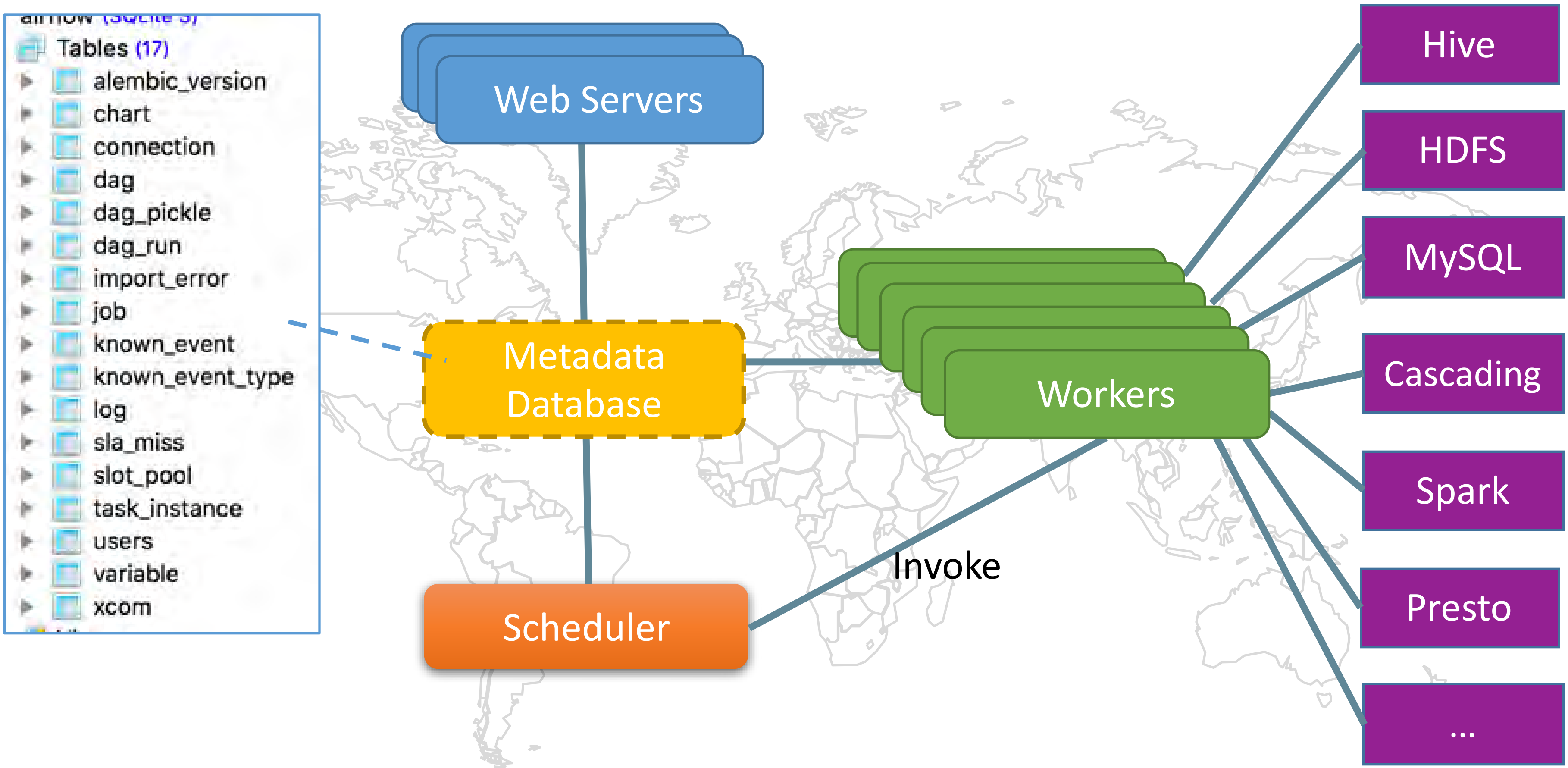
Scalability

High Availability

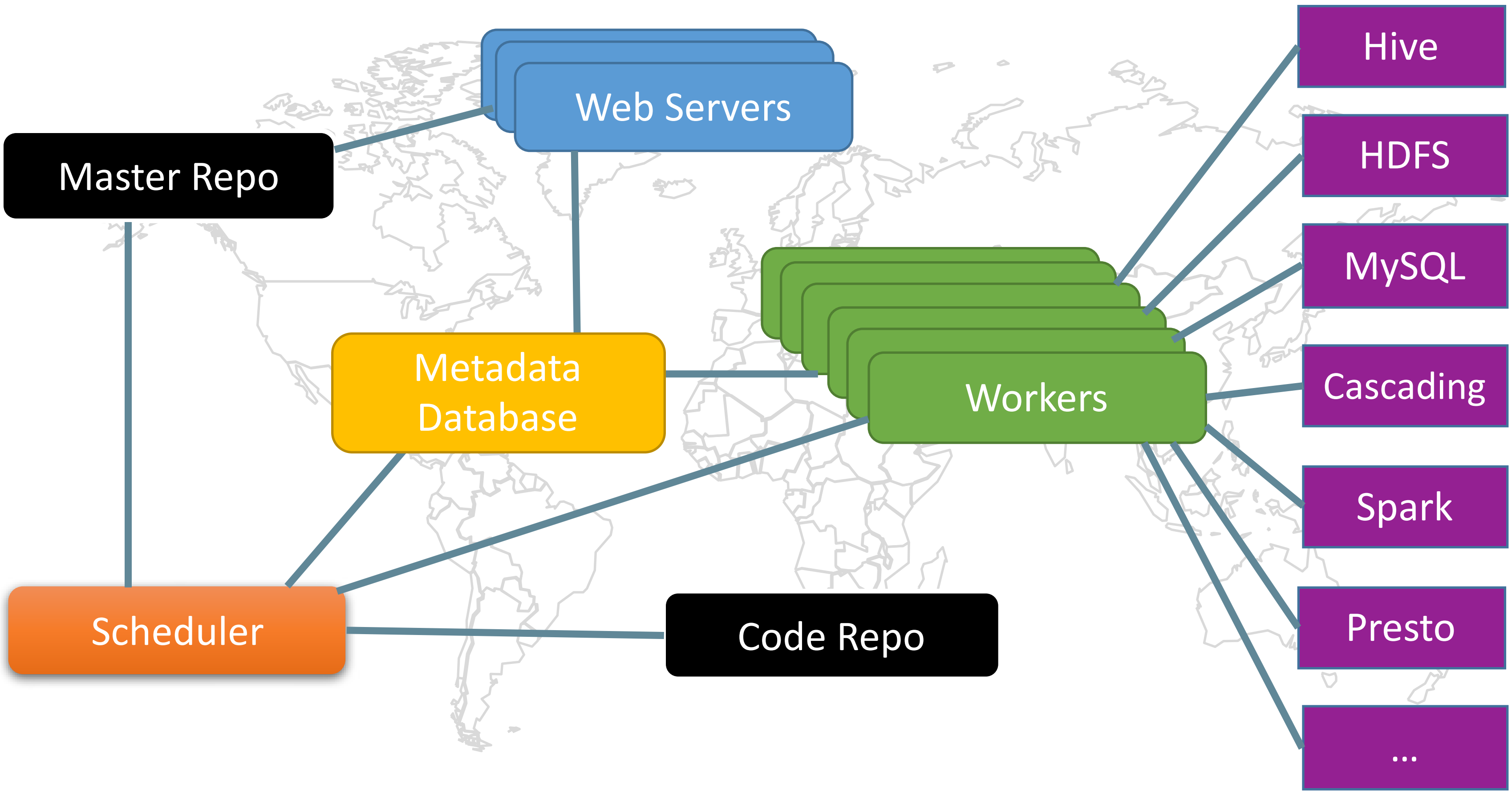
Versioning

丁来强 wjo1212@163.com

• Airflow Architecture (Local Scheduler)

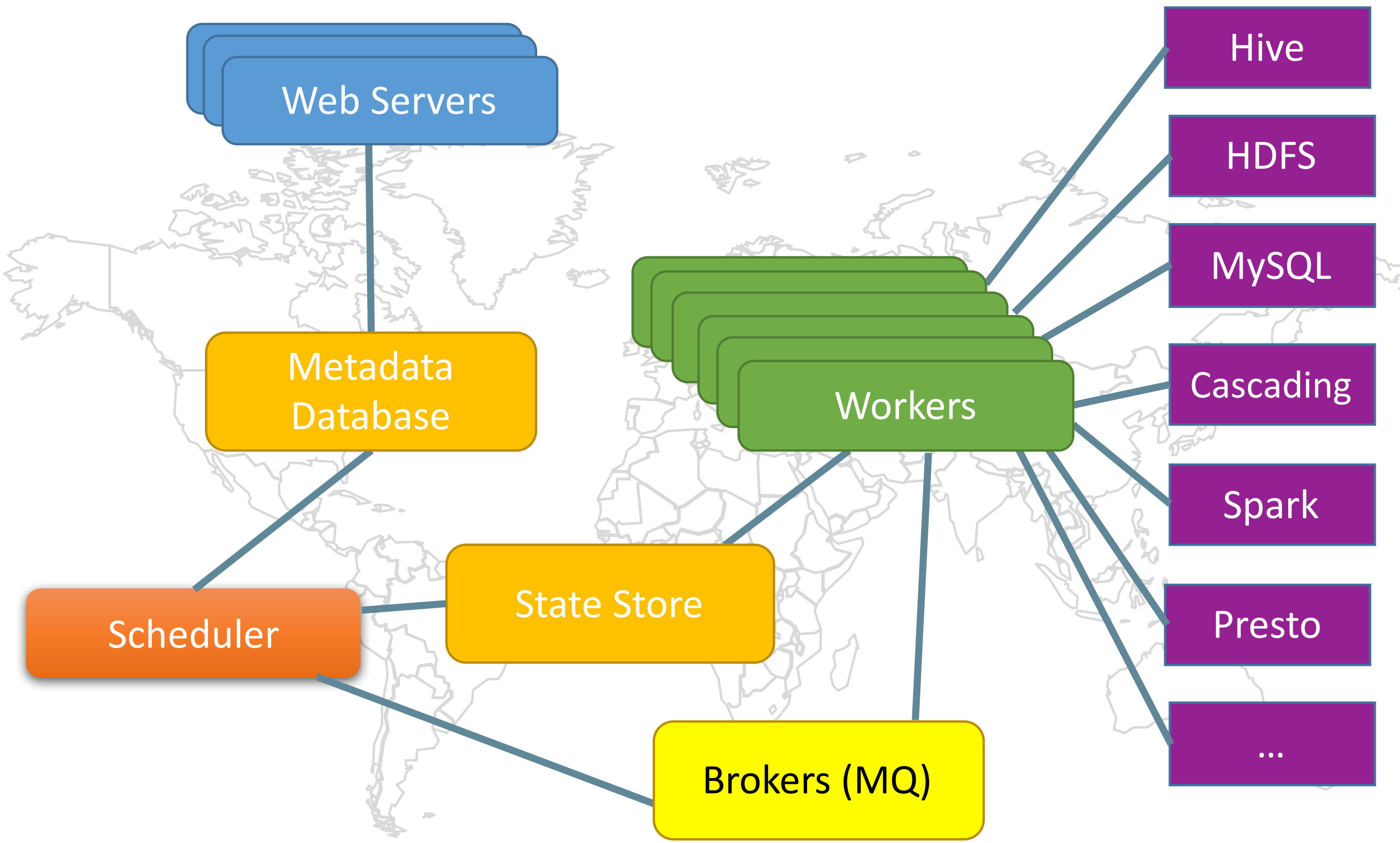


Local Scheduler – w/ version control

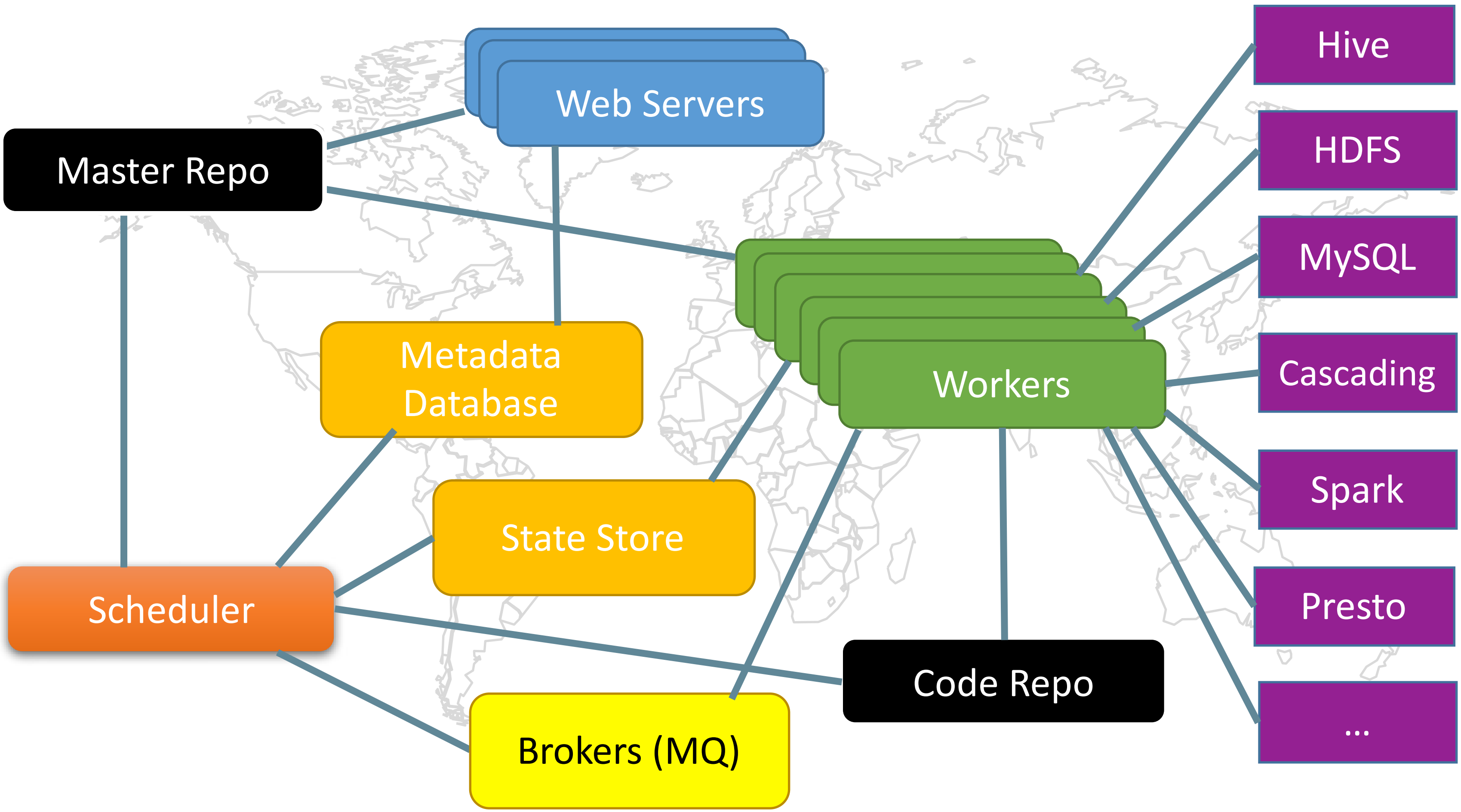


丁来强 wjo1212@163.com

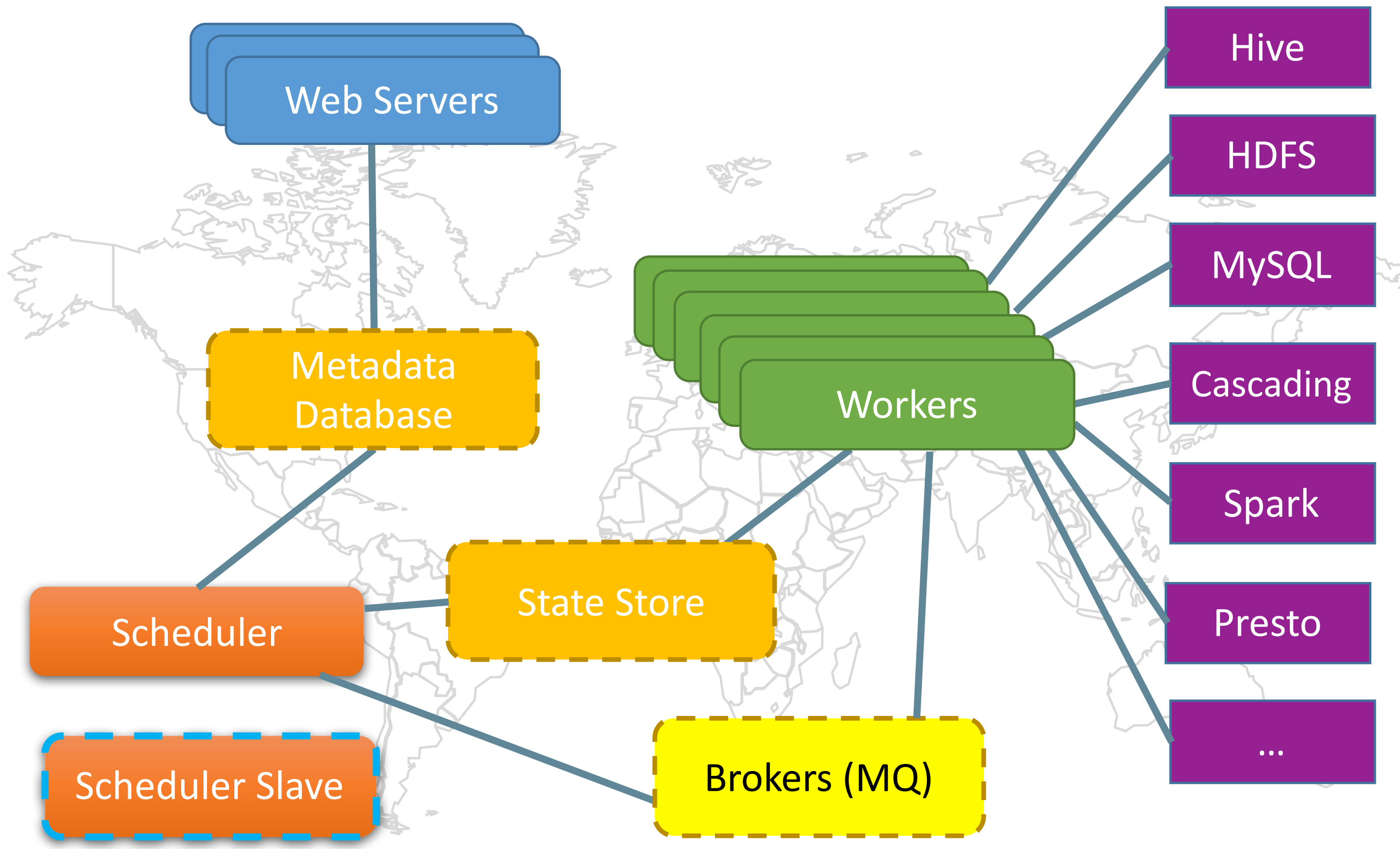
Airflow Architecture (Celery Scheduler)



Celery Scheduler – w/ version control



Airflow Architecture - HA



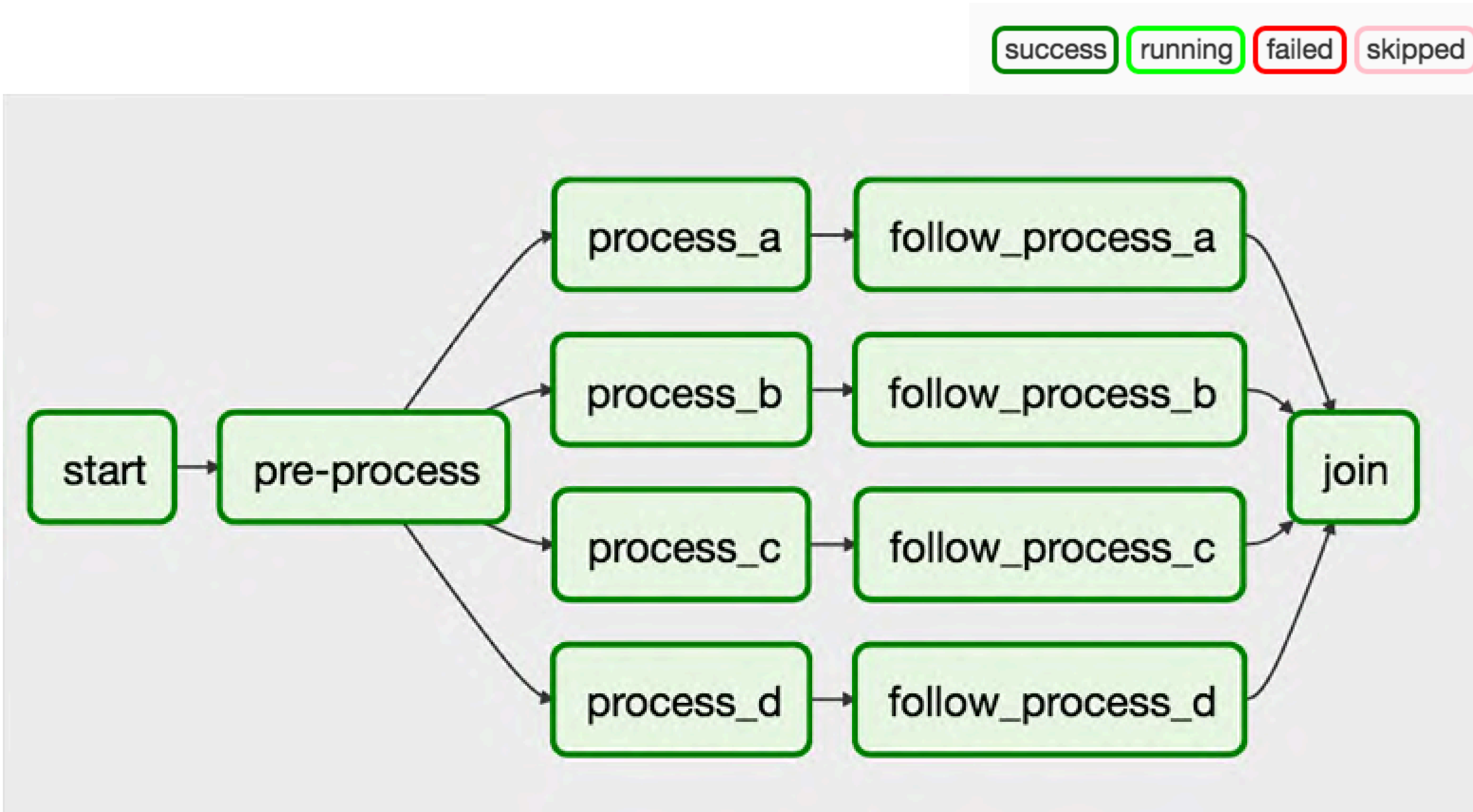
丁来强 wjo1212@163.com

Workflow Patterns

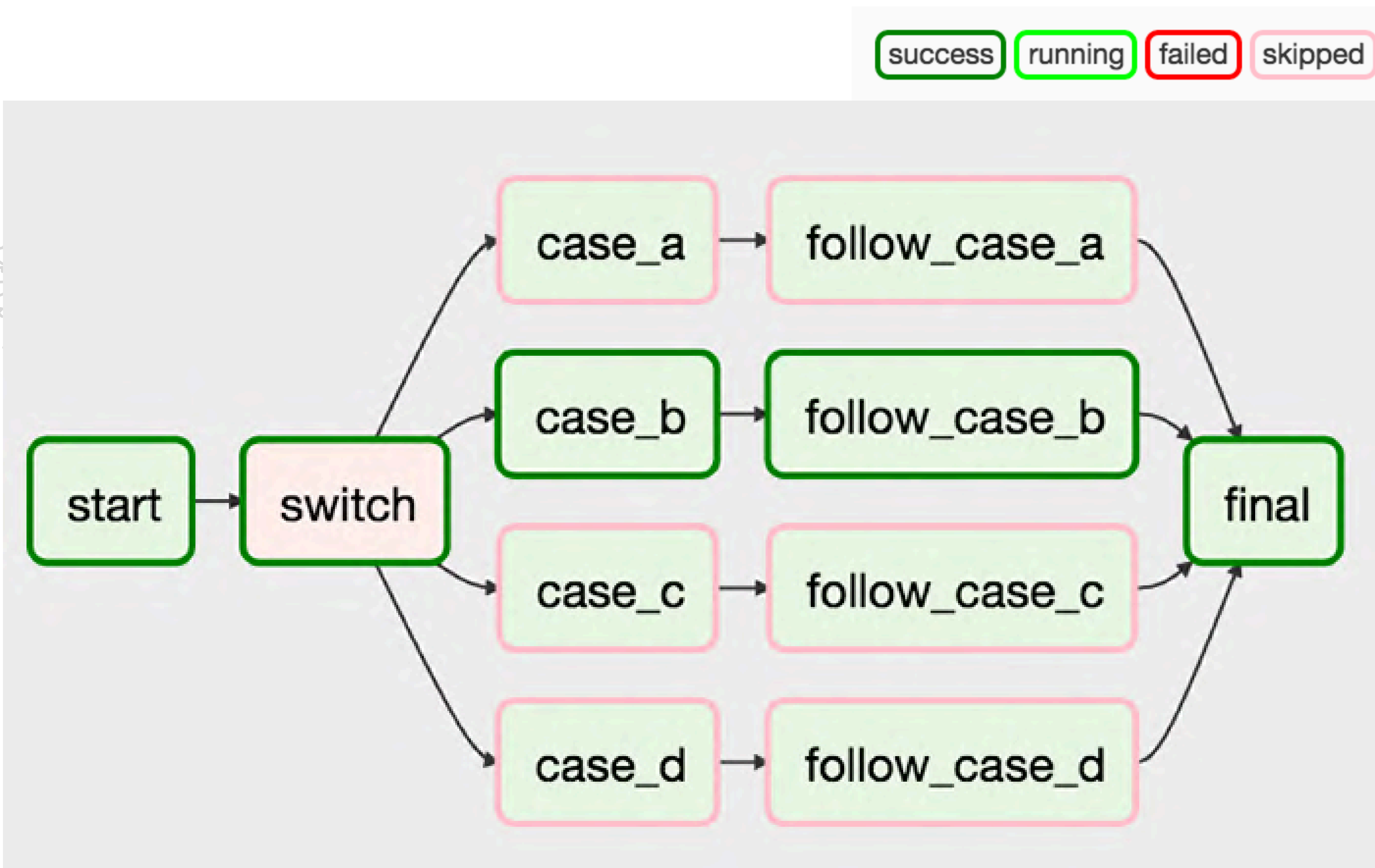
Complex Rule

丁来强 wjo1212@163.com

- Process in parallel

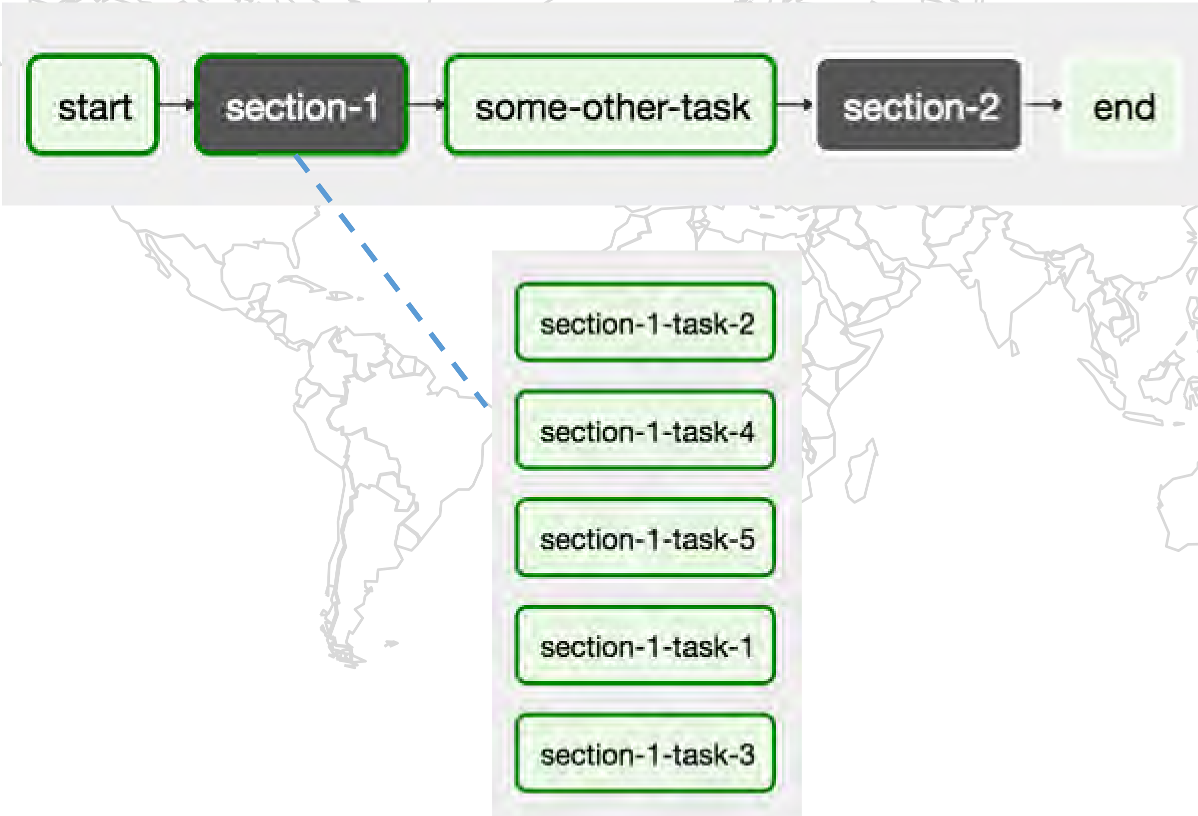


- Switch


success
running
failed
skipped
start
switch
case_a
case_b
case_c
case_d
follow_case_a
follow_case_b
follow_case_c
follow_case_d
final

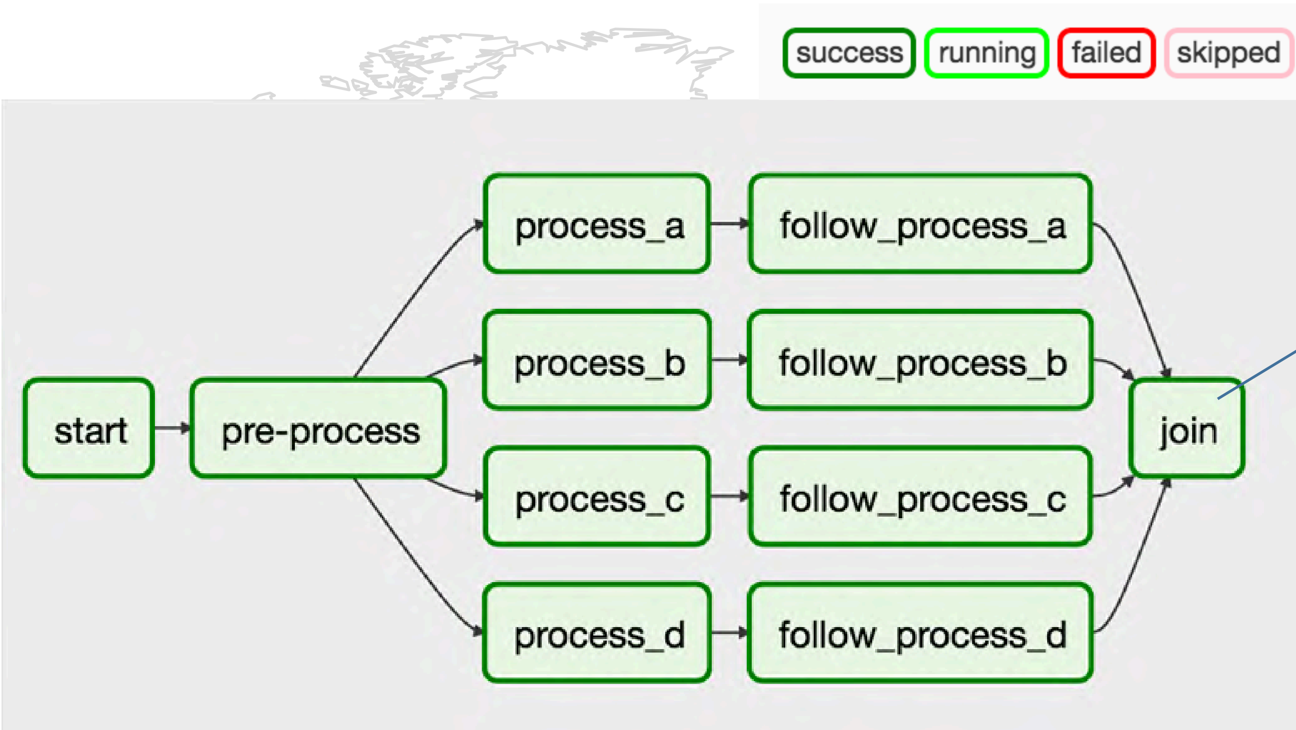
Sub Dag

- Easier to control, re-use and test
- Just like a component in code



丁来强 wjo1212@163.com

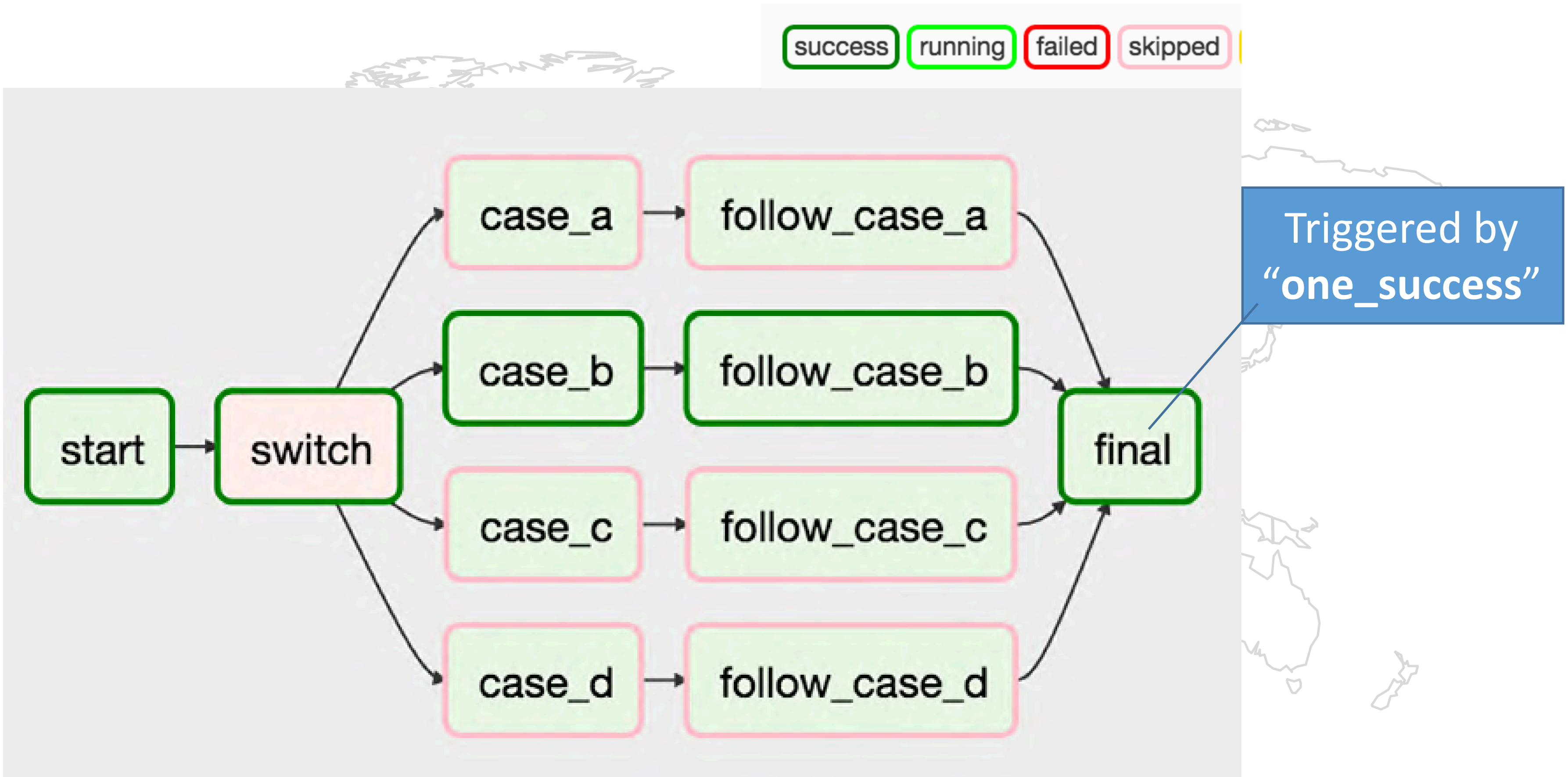
- # Trigger Rule – all success



Triggered by "all_success"

丁来强 wjo1212@163.com

● Trigger Rule – one success

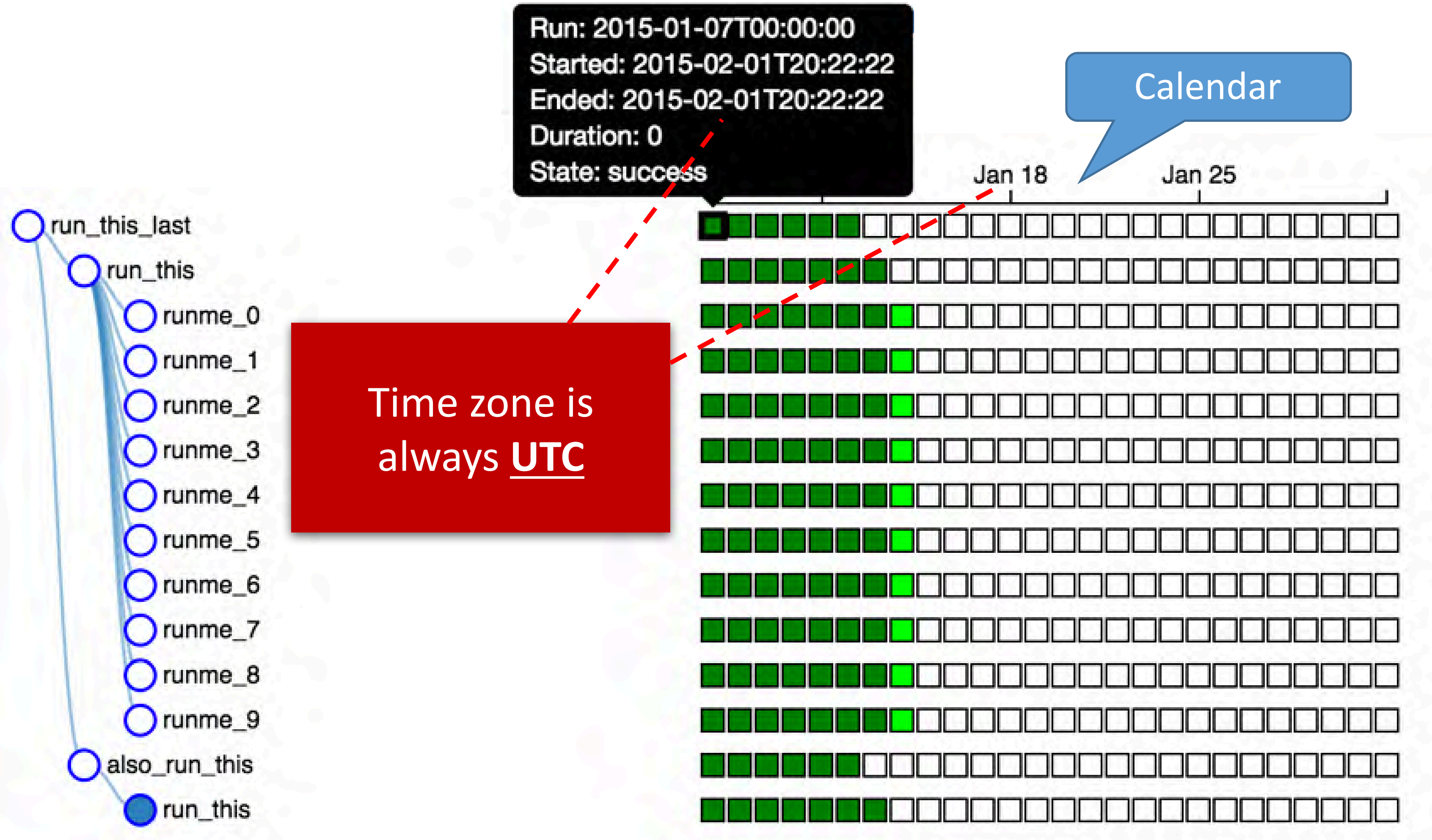


Scheduling Practice

Calendar Based Scheduling

丁来强 wjo1212@163.com

● Calendar based scheduling (UTC)



Scheduler – interval in workflow

```python
dag = DAG(dag_id="mydag",
    default_args= {
        'start_date': datetime(2016,9,8),
    },
    schedule_interval='0 */4 * * *')
```

Run: scheduled__2016-09-09T16:00:00

- ✓ scheduled__2016-09-09T16:00:00
- scheduled__2016-09-09T12:00:00
- scheduled__2016-09-09T08:00:00
- scheduled__2016-09-09T04:00:00
- scheduled__2016-09-09T00:00:00
- scheduled__2016-09-08T20:00:00
- scheduled__2016-09-08T16:00:00
- scheduled__2016-09-08T12:00:00
- scheduled__2016-09-08T08:00:00
- scheduled__2016-09-08T04:00:00
- scheduled__2016-09-08T00:00:00

Run at 04:00:00

Note

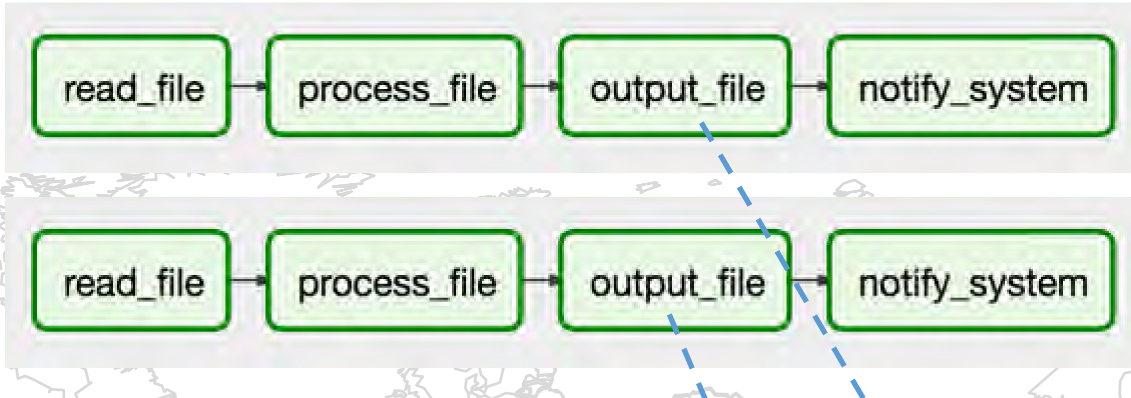
Every Dag Run will only start when next Dag Run's execution time meets

丁来强 wjo1212@163.com

Scheduler – recursive running

Run1 (09-01 <u>00:00</u>)

Run2 (09-01 <u>04:00</u>)



What if ***<u>output_file</u>*** in ***<u>Run1</u>*** *and* ***<u>Run2</u>*** impact each others?

Idea

<u>Try best</u> to avoid this kind of design

丁来强 wjo1212@163.com

Principle when defining Task

Granularity

Task granularity should be proper

- Choose “Right size” for one task
- Task should execute simultaneously

Atomic

Each Task should be atomic

- isolation from concurrent processing
- Either succeed or failure, no grey state
- Failure will not impact the system

• Idempotent Task

```
def do_job(*args, **kwargs):
    make_file1()
    make_file2()
    make_file3()
    do_final_process()

def clean_up(context):
    clean_file_if_exist(
        'file1_path',
        'file2_path',
        'file3_path')

task = PythonOperator(
    task_id='file_operation',
    provide_context=True,
    python_callable=do_job,
    on_retry_callback=clean_up,
    dag=dag)
```

Cleanup env
when failure

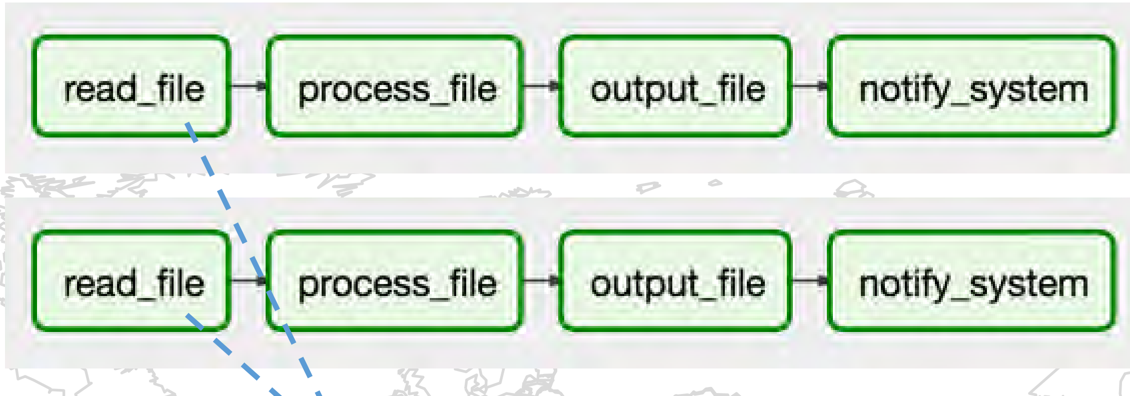
It’s ideal case, in real cases…

丁来强 wjo1212@163.com

Scheduler – recursive dependency

Run1 (09-01 <u>00:00</u>)

Run2 (09-01 <u>04:00</u>)



What if ***read_file*** in ***Run1*** *and* ***Run2*** cannot run in parallel due to external system's limitation

Option 1

Assign a **pool** with **1 Slots** to for task **read_file**

Option 2

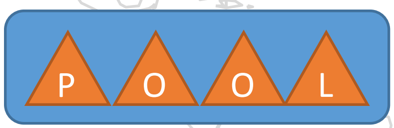
Turn on option "**depends_on_past**" for task **read_file**

丁来强 wjo1212@163.com

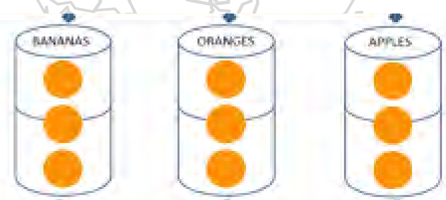
Resource Control

List (2) | Create | With selected▾

		Pool	Slots	Used Slots	Queued Slots
		db connection	10	0	0
		AWS CloudTrial Connections	10	0	0



Pool (Limit concurrency + priority)



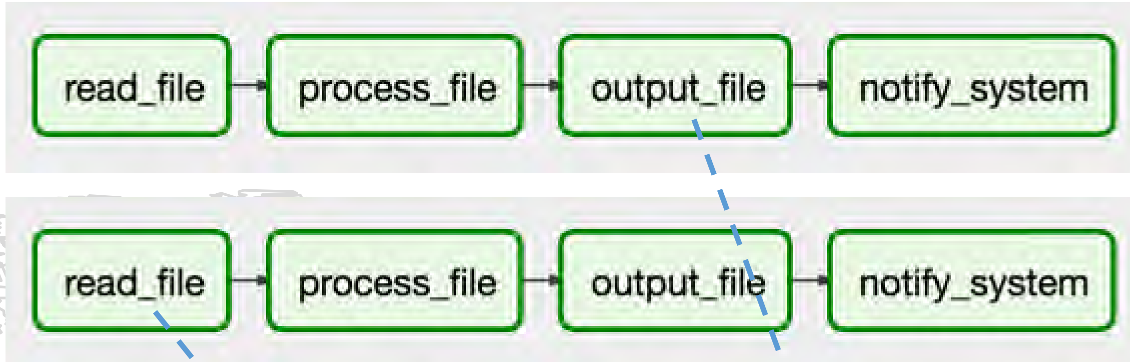
Queue (Affinity)

丁来强 wjo1212@163.com

Scheduler – more recursive dependency

Run1 (09-01 <u>00:00</u>)

Run2 (09-01 <u>04:00</u>)



What if ***<u>read_file</u>*** in ***<u>Run2</u>*** replies on ***output_file*** in ***<u>Run1</u>*** due to restriction or necessary stateful design?

Option

Turn on option “**wait_for_downstream**“ for task **read_file** (This will force to turn on **“depends_on_past”**)

丁来强 wjo1212@163.com

Scheduler – recursive dependency pitfall

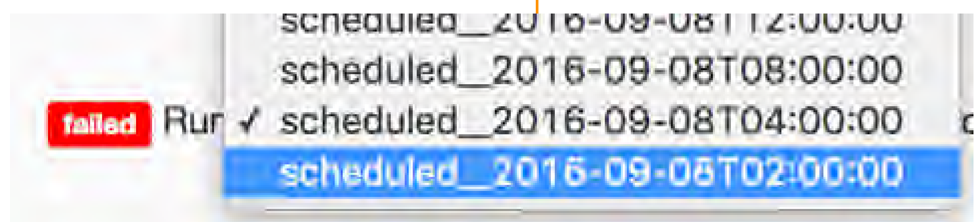
```
dag = DAG(dag_id="mydag",
    default_args= {
        'start_date': datetime(2016,9,8),
    },
    schedule_interval='0 */4 * * *')
```

```
'@once'      just one time
'@hourly': '0 * * * *',
'@daily': '0 0 * * *',
'@weekly': '0 0 * * 0',
'@monthly': '0 0 1 * *',
'@yearly': '0 0 1 1 *',
```

Note

start_date and **schedule_interval** should be aligned

2016-09-08 00:00:00 is aligned
2016-09-08 **<u>02:00:00</u>** is NOT aligned



This will make the DAG failure if the option "**depends_on_past**" is turned on

Some other notes

- Update the dag id when changing the logic inside
- Using SLA alert for critical tasks
- Feature in plan:
  - Event Driven Scheduler
  - Mesos Scheduler
  - More operators
  - More syntax sugar

Demo

丁来强 wjo1212@163.com

Now you've learned:

- Definition and ecosystem.
- Challenges and key requirements.
- Solutions and general comparisons.
- Most important part of Airflow and Luigi
  - Architecture, design, patterns, pitfalls and practices etc.

丁来强 wjo1212@163.com

谢谢观看

wjo1212

wjo1212@163.com

LaiQiangDing