
CERN

Signal Monitoring Project

Use of Apache Airflow for Historical Data Collection and Signal Monitoring Execution

Kasper Andersen, Zinur Charifoulline, Per Hagen, Michał Maciejewski, Christoph Obermair, Sivert Sagmo, Arjan Verweij
TE-MPE-PE

with technical help from:
TE-MPE-MS: Thibaud Buffet, Jean-Christophe Garnier, Markus Zerlauth
IT-DB-SAS: Prasanth Kothuri, Piotr Mrówczyński
BE-CO-APS: Grigorios Avgitidis, Krzysztof Krynicki, Marcin Sobieszek, Jakub Woźniak, Bartek Urbaniec

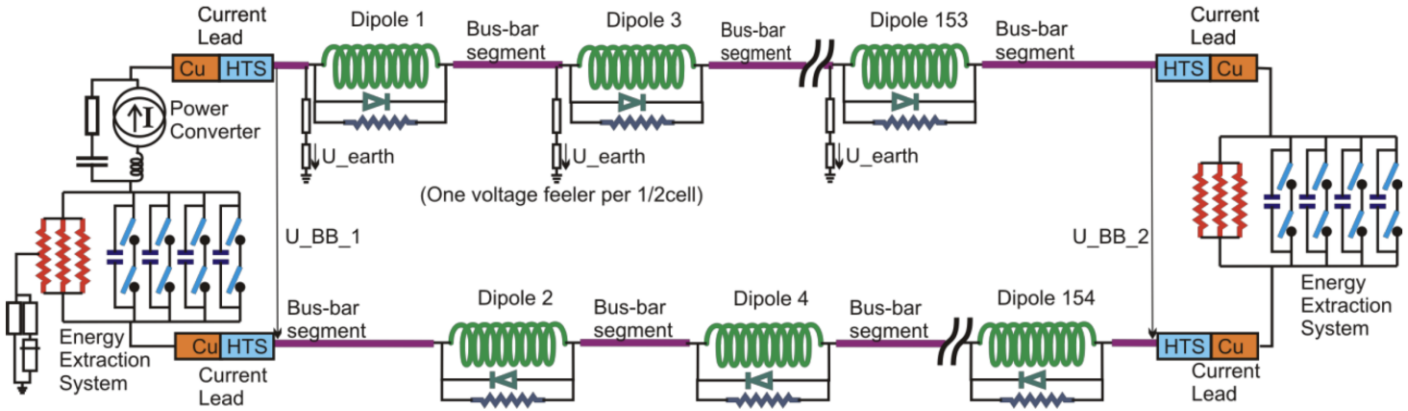
https://cern.ch/sigmon

Scope

We develop a general-purpose framework for monitoring:

1. Superconducting magnets and busbars
2. Circuit and magnet protection systems
3. Grounding networks
4. Current leads
5. …

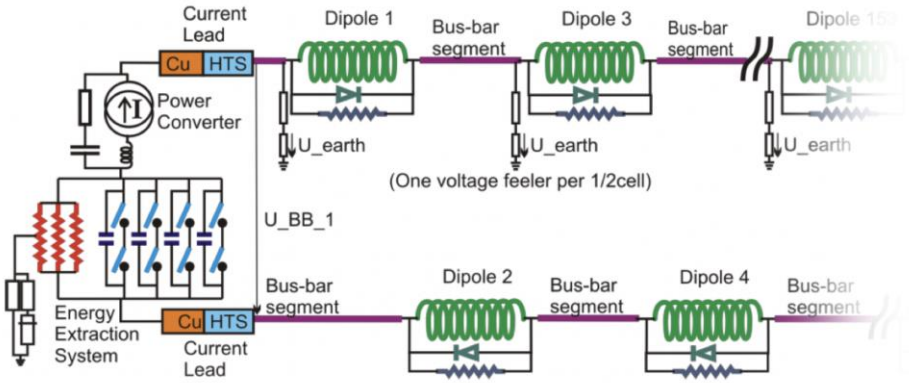
… with potential to extend to any other system with logged signals in PM and NXCALS* (e.g., Power Converters, Cryogenics Equipment, **Beam Instrumentation**).



*NXCALS introduces a **paradigm shift** from local to cluster computing
The data should be processed where it is stored, i.e., on the cluster

Requirements

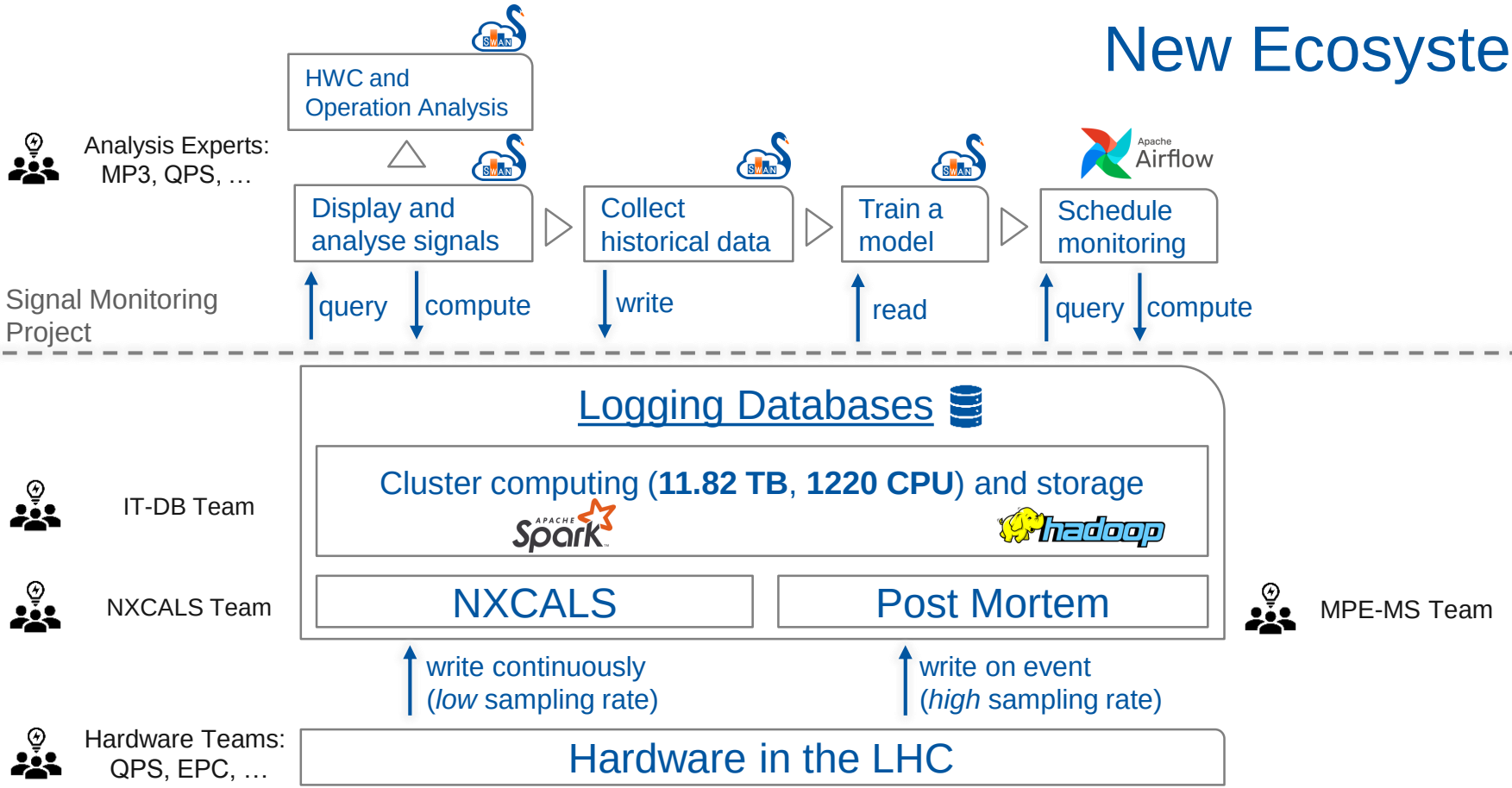
- Many signals vary strongly during the operation cycle (ramp, FPA, flat-top, etc).
- Monitored signals should be compared to warning or alarm thresholds.
- Exceeding these thresholds should trigger an e-mail.
- User-friendly interface is needed to define signals, type of monitoring, time intervals, warning/alarm levels, e-mail address person to be notified.



→ Flexible triggering process
→ Intuitive user-interface

Source: Project Kick-Off Presentation

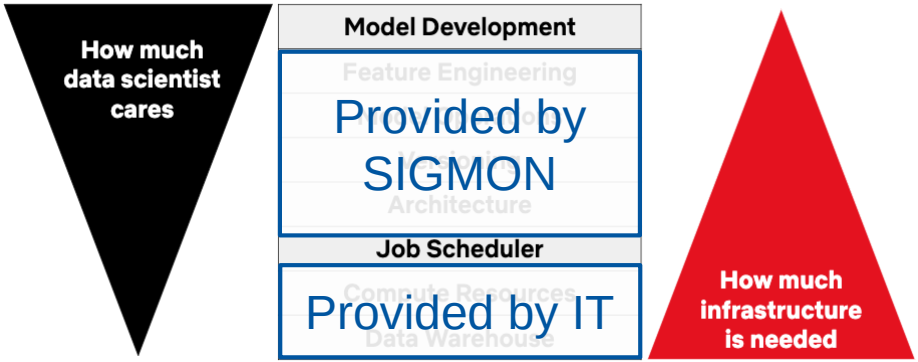
New Ecosystem



We aim at **a coherent approach** over all circuits, all systems, and *all* types of analysis
We use **python** due to the wealth of libraries for data analysis and modelling
We employ **SWAN notebooks** for development and communication across teams

Motivation

*The **new cluster** computing ecosystem and a **growing interest** in data-driven models call for a **standardized workflow and infrastructure** for data collection, modelling, and signal monitoring scheduling.*



Source: https://netflixtechblog.com/open-sourcing-metaflow-a-human-centric-framework-for-data-science-fa72e04a5d9

Complex data workflows contribute to reproducibility crisis in science, Stanford scientists say

The main concerning takeaway from our study is that, given exactly the same data and the same hypotheses, different teams of researchers came to very different conclusions

- Russell Poldrack

While worrisome, Poldrack said the findings can help researchers assess and improve the quality of their data analyses moving forward. **Potential solutions** include ensuring that data is analyzed **in multiple ways**, as well as making data analysis workflows **transparent and openly shared** among researchers.

https://news.stanford.edu/2020/05/20/complex-data-workflows-contribute-reproducibility-crisis/

Outline

1. Signal Monitoring
2. Apache Airflow
3. Applications
4. Workflows
5. Summary

Signal Monitoring Workflow

Exploration → **Data Collection** → Modelling → Monitoring

Exploration – getting the signal features *right*

Creation of a notebook to explore a signal and compute characteristic features

	Feature 1	Feature 2	Feature …	Feature n
event 1	0.078	980	…	10.4

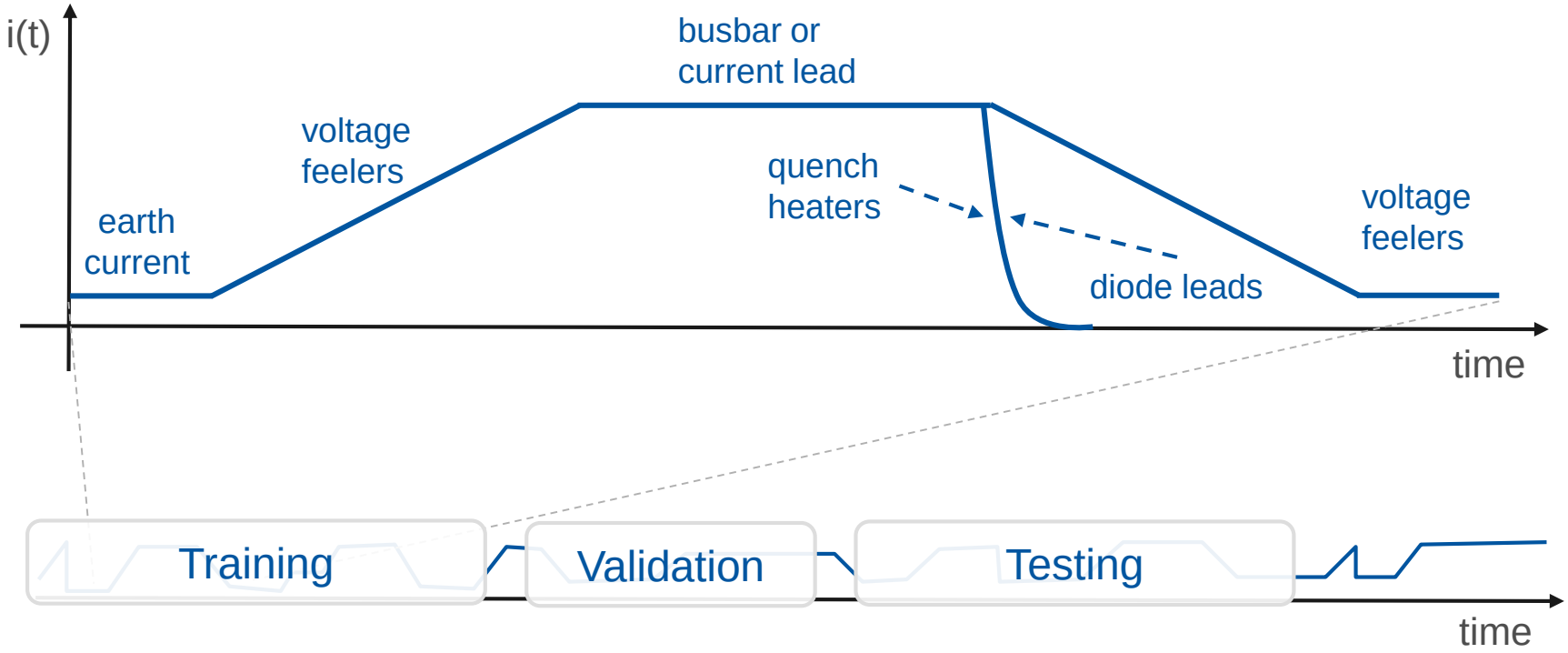
Data Collection – getting the *right* signal features

Execution of a notebook over past operation to collect data for numerical models

	Feature 1	Feature 2	Feature …	Feature n
event 1	0.078	980	…	10.4
event 2	0.081	995	…	9.8
event …	…	…	…	…
event m	.08	1000	…	10.1

Signal Monitoring Workflow

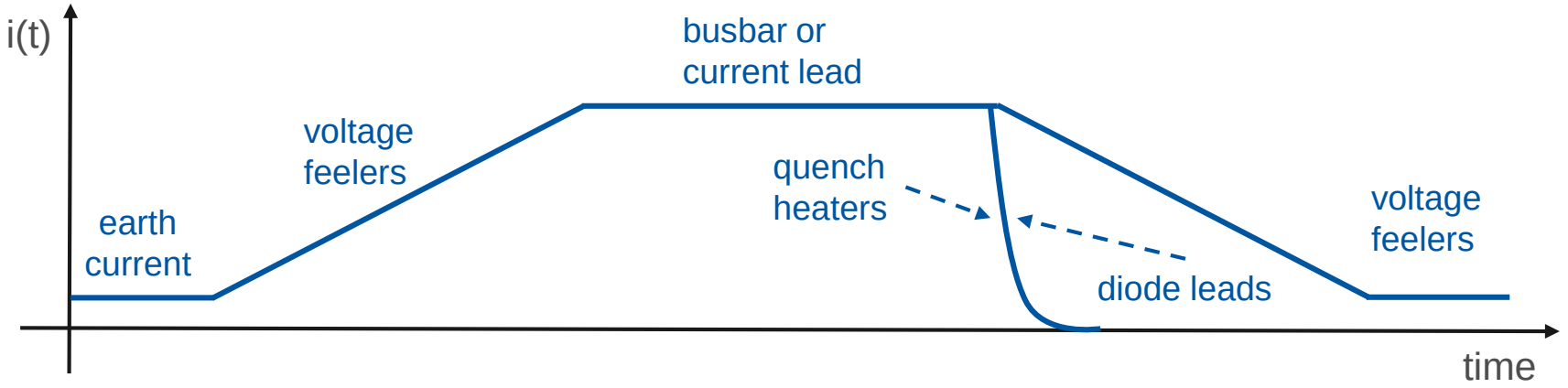
Exploration → Data Collection → **Modelling** → Monitoring



Data-driven models encode historical data in a compact way.

Signal Monitoring Workflow

Exploration → Data Collection → Modelling → **Monitoring**



Automatic execution of monitoring application depends on the state of operation:

- triggered by PM events (PC, QH, MAGNET) and AccTesting (test duration, circuit)
- triggered by change in the beam mode (GND, COLDBB)
- scheduled in regular intervals, e.g., every hour (DFB)

→ Our scheduler (Apache Airflow) can be triggered on demand through REST API

Limitations of SWAN

SWAN is a great tool for signal exploration, however:

- the duration of a user session is limited (data collection takes time)
- connection sometimes breaks
- broken analysis is not automatically restarted
- analysis can not be triggered on demand

We need a dedicated solution for scheduling of historical data collection and monitoring. Development of an in-house tool is beyond the scope and capabilities of our project.

A Solution - Cron

A standard solution for scheduling analysis jobs is cron.

IT department supports cron as a scheduler system.
However, it does not offer any user interface (a terminal application) and requires use of a special scripting language.

Each line of a crontab file represents a job, and looks like this:

```
# ┌──────────────── minute (0 - 59)
# │ ┌────────────── hour (0 - 23)
# │ │ ┌──────────── day of the month (1 - 31)
# │ │ │ ┌────────── month (1 - 12)
# │ │ │ │ ┌──────── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                  7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
```

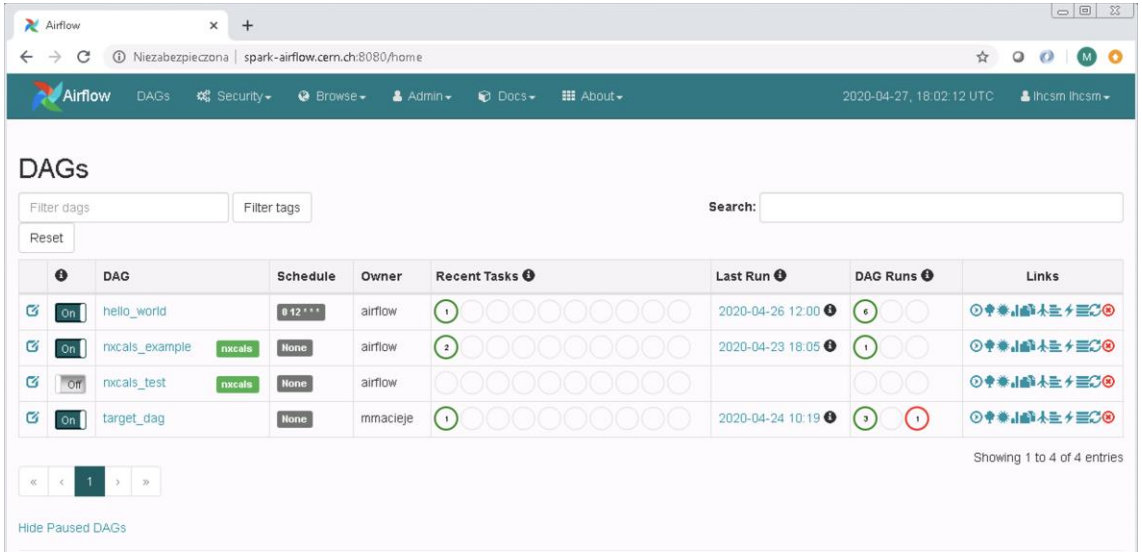
Source: https://en.wikipedia.org/wiki/Cron

A Solution – Apache Airflow

Airflow is designed under the principle of "configuration as code".

While other "configuration as code" use markup languages (XML), Airflow allows developers to import python libraries and classes.

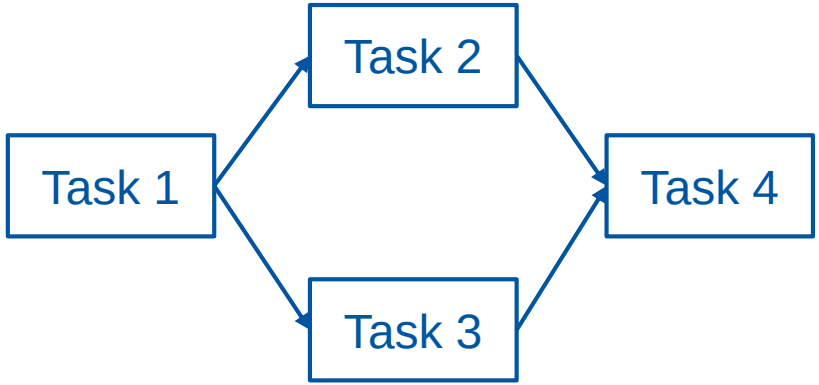
In addition, Airflow provides an intuitive WebUI.



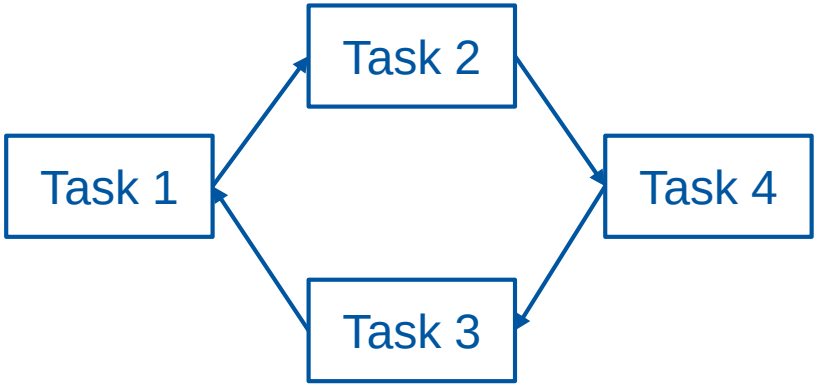
Source: https://en.wikipedia.org/wiki/Apache_Airflow

Apache Airflow

Apache Airflow is based on a concept of direct acyclic graphs to represent stages of application execution.



This is a DAG



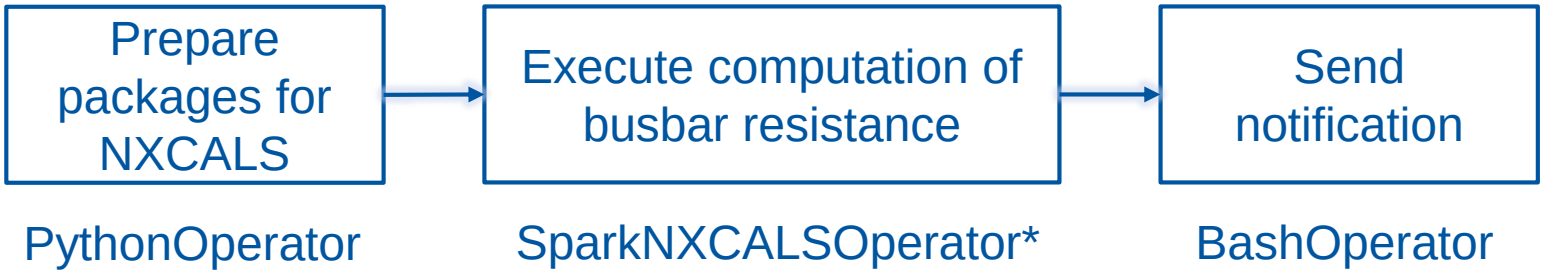
This is not a DAG

Each task is based on an operator, e.g. PythonOperator

For more details, please consult: https://airflow.apache.org

Sample DAG

Apache Airflow provides python scripting capabilities with an intuitive way for editing script parameters and an integration with the NXCALS cluster*.



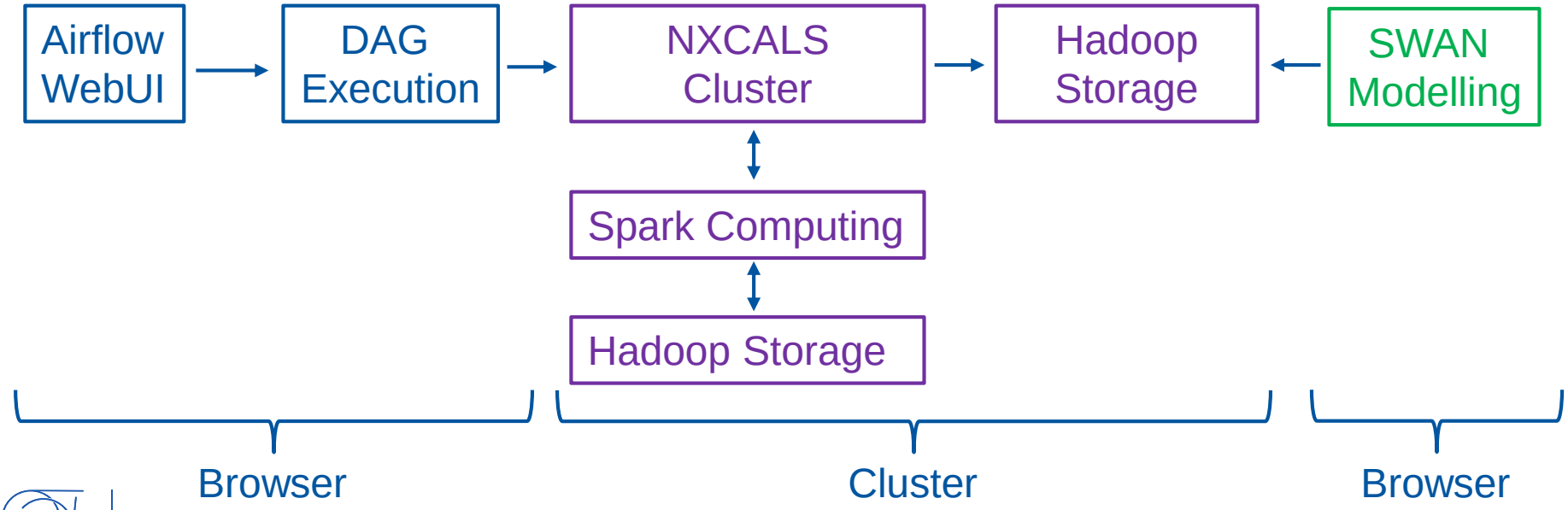
With the SparkNXCALSOperator we can execute operations just like with SWAN.

* https://gitlab.cern.ch/db/spark-service/airflow/-/blob/master/plugins/nxcals_plugin/operators/nxcals_operator.py

For more details please check a presentation from Piotr Mrówczyński:
https://cernbox.cern.ch/index.php/s/8ypLPt6JTCIGIc7

How does it work?

- ✓ A DAG is triggered from Airflow manually, on-demand, in regular intervals.
- ✓ An NXCALS computation is executed on the cluster (outside of Airflow environment).
- ✓ The computation stores results in Hadoop, which is accessible from SWAN



Demo

1. Hello World
2. NXCALS example
3. On-demand analysis
4. Computation of busbar resistance historical data
5. Computation of BLM historical data

- User-friendly interface is needed to define signals, type of monitoring, time intervals, warning/alarm levels, e-mail address person to be notified.

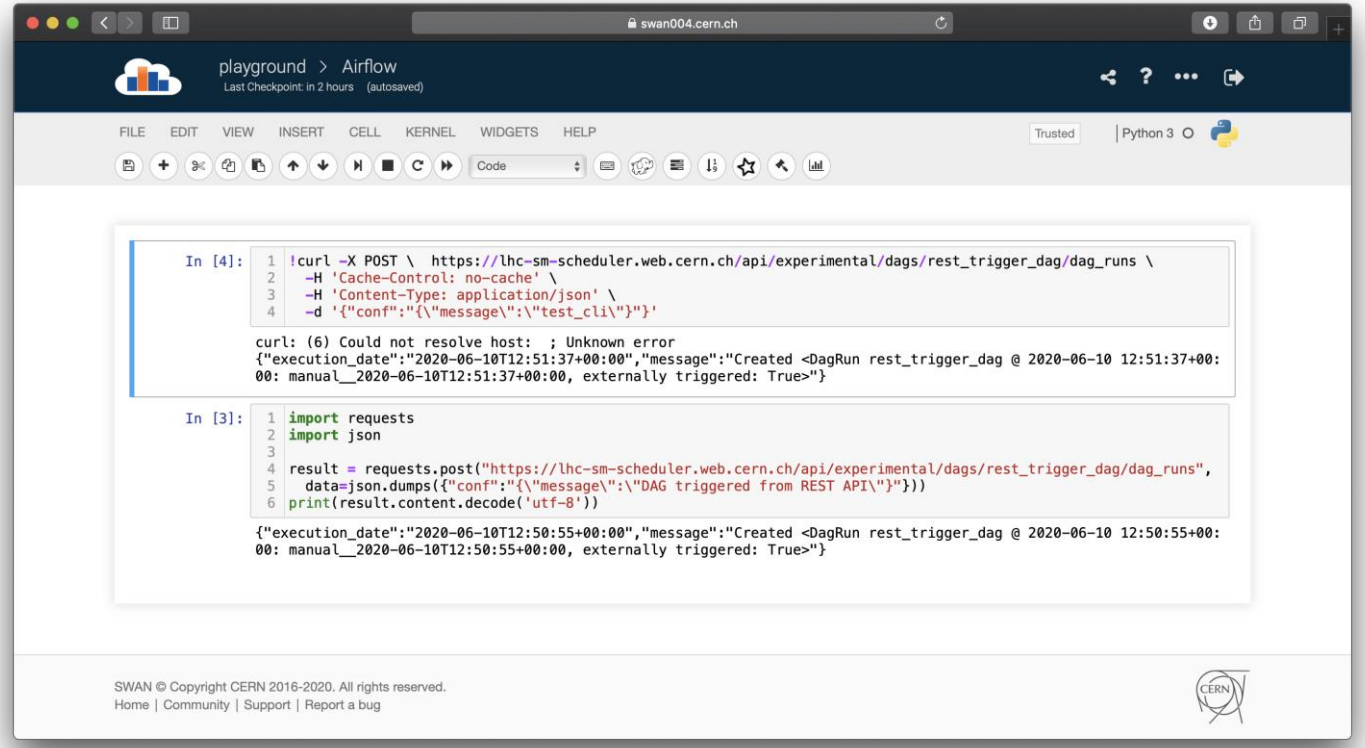
Demo

1. **Hello World**
2. **NXCALS example**
3. On-demand analysis
4. Computation of busbar resistance historical data
5. Computation of BLM historical data (work in progress)

- User-friendly interface is needed to define signals, type of monitoring, time intervals, warning/alarm levels, e-mail address person to be notified.

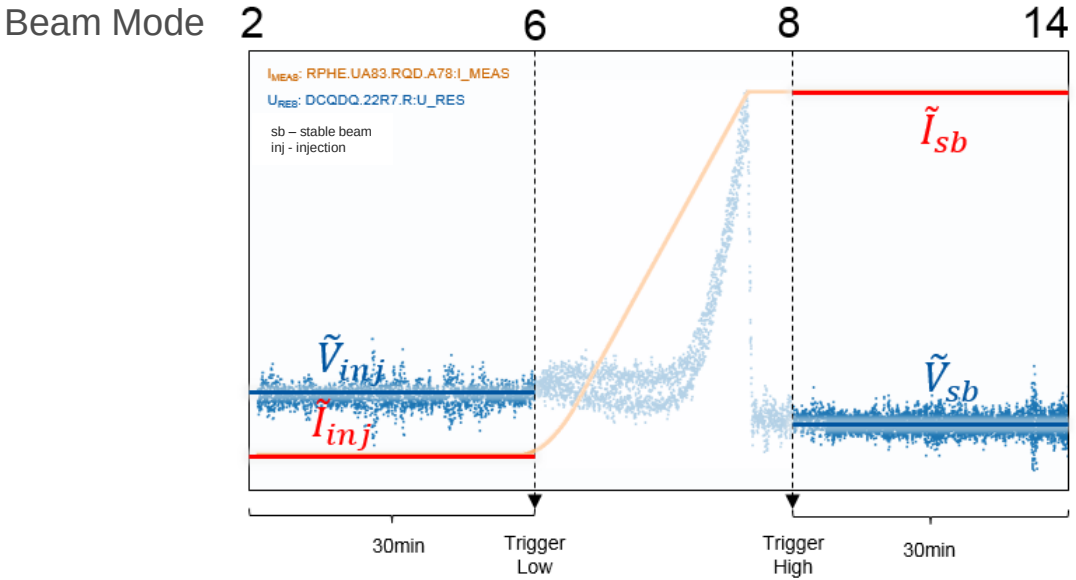
On-Demand Execution

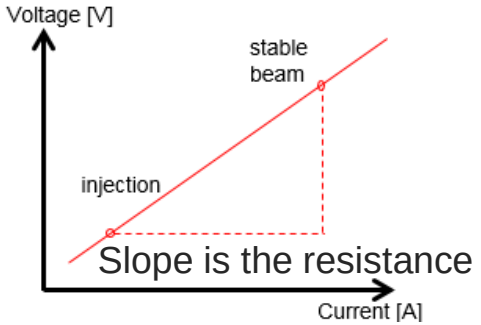
Our analysis can be triggered directly by hardware, e.g. an FPA, a change in the beam mode, etc.



Busbar Resistance

Monitoring of busbar involves calculation of 1248 (RB) and 400 (RQ) resistances
The resistance is calculated from a linear fit of voltage and current at plateaus





Courtesy: C. Obermair

Z. Charifoulline, et al., "Resistance of Splices in the LHC Main Superconducting Magnet Circuits at 1.9 K," IEEE TAS, 28(3), Apr. 2018.
C. Obermair, SUMM Report: http://cds.cern.ch/record/2639871?ln=en

Busbar Resistance

We gather historical data for Run2

```
i_meas_feature_df = QueryBuilder() \
    .with_nxcals(spark) \
    .with_duration(t_start='2018-05-21 12:22:37', t_end='2018-05-21 13:49:12') \
    .with_circuit_type('RB') \
    .with_metadata(circuit_name='*', system='PC', signal='I_MEAS') \
    .feature_query(['mean', 'std'], function=translate_udf).sort_values(by='class').df

u_res_feature_df = QueryBuilder() \
    .with_nxcals(spark) \
    .with_duration(t_start='2018-05-21 12:22:37', t_end='2018-05-21 13:49:12') \
    .with_circuit_type('RB') \
    .with_metadata(circuit_name='*', system='BUSBAR', signal='U_RES', wildcard={'BUSBAR': '*'}) \
    .feature_query(['mean', 'std'], function=translate_udf).sort_busbar_location('RB', circuit_name='*').df

ResistanceBuilder().with_busbar_voltage(u_res_feature_df).with_busbar_current(i_meas_feature_df) \
    .calculate_mean_resistance('RB').convert_to_row(index=t_start_inj)
```

Apache Spark: 10 EXECUTORS 20 CORES Jobs: 36 COMPLETED

	DCBB.8L2.R_mean_inj	DCBB.8L2.R_std_inj	DCBB.8L2.R_mean_sb	DCBB.8L2.R_std_sb	DCBB.8L2.R_R_RES	DCBB.9L2.R_mean_inj	DCBB
1526898157236000000	0.000064	0.000043	0.000076	0.000022	1.164031e-09	-0.000022	

NXCALS cluster computation of **1248** busbar resistances takes approximately as much time as query and local processing of **8** power converter currents

The selection of current plateaus is performed with a *user defined function*

Analysis with Checkpoints

Although we develop a solid software, the infrastructure can break leading to an abort of the computation.

In order to avoid calculating the same thing twice, we use checkpoints.

index	status
1	✓
2	✓
3	✓
4	
5	
6	

Analysis aborted →

index	status
1	✓
2	✓
3	✓
4	✓
5	
6	

Analysis aborted →

index	status
1	✓
2	✓
3	✓
4	✓
5	✓
6	✓

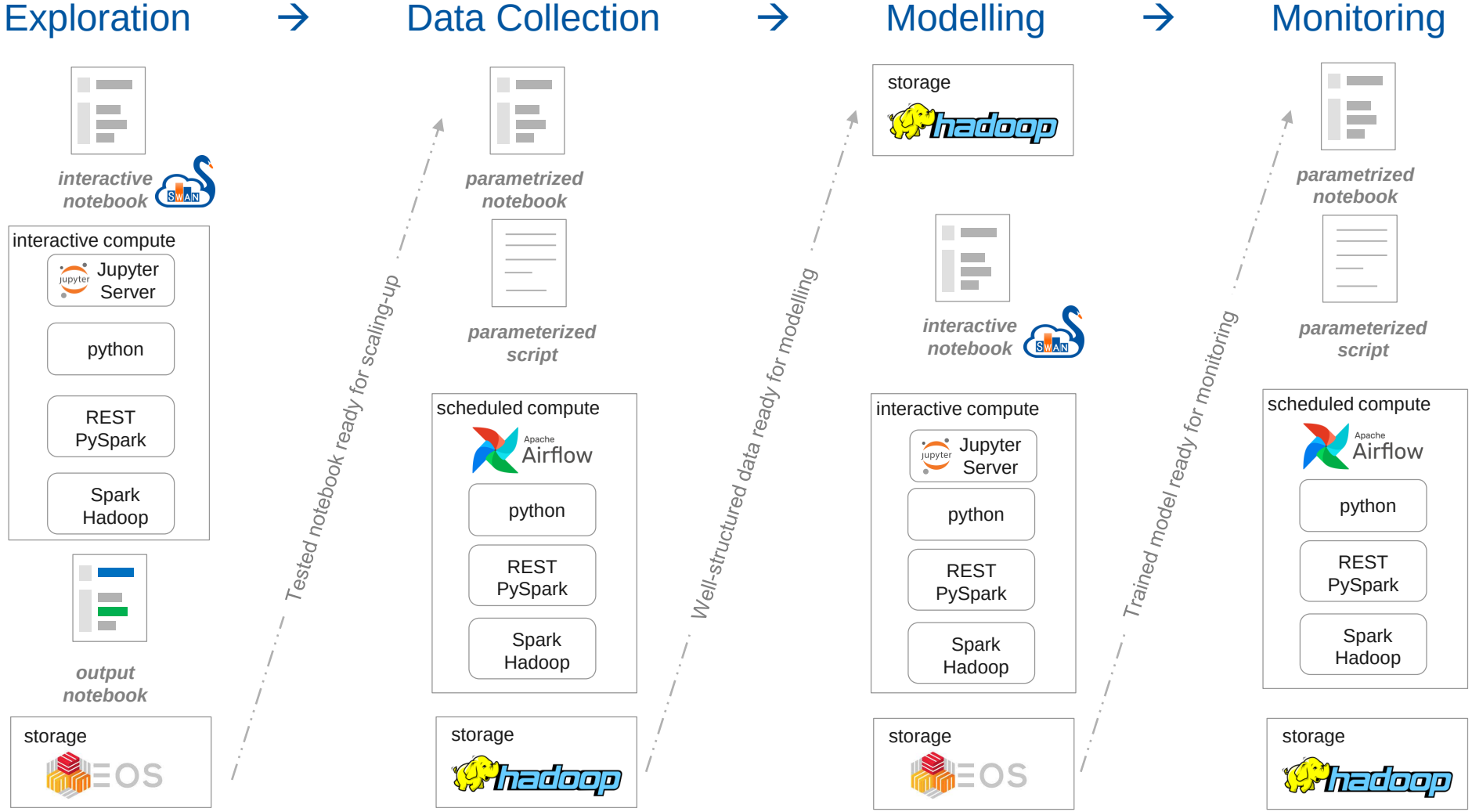
Saving Data with Partitions

While performing data collection over long periods of time, e.g. Run2 (2015-2018), it is advised to split the output dataset into partitions per year.

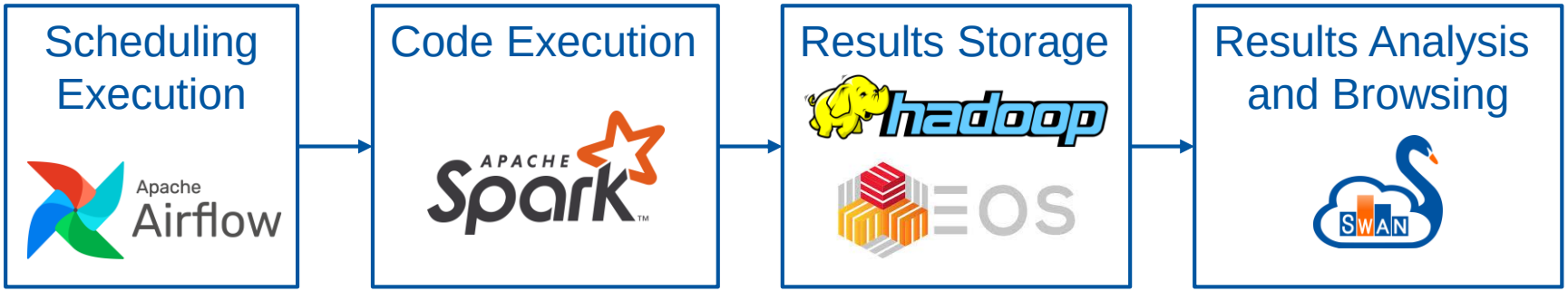
```
bash-4.2$ hdfs dfs -ls /project/lhc_signal_monitoring/busbar/rb
Found 4 items
drwxr-xr-x+  - lhcsm hdfs          0 2020-06-10 01:29 /project/lhc_signal_monitoring/busbar/rb/partition=2015
drwxr-xr-x+  - lhcsm hdfs          0 2020-06-08 14:24 /project/lhc_signal_monitoring/busbar/rb/partition=2016
drwxr-xr-x+  - lhcsm hdfs          0 2020-05-09 06:32 /project/lhc_signal_monitoring/busbar/rb/partition=2017
drwxr-xr-x+  - lhcsm hdfs          0 2020-05-06 03:30 /project/lhc_signal_monitoring/busbar/rb/partition=2018
```

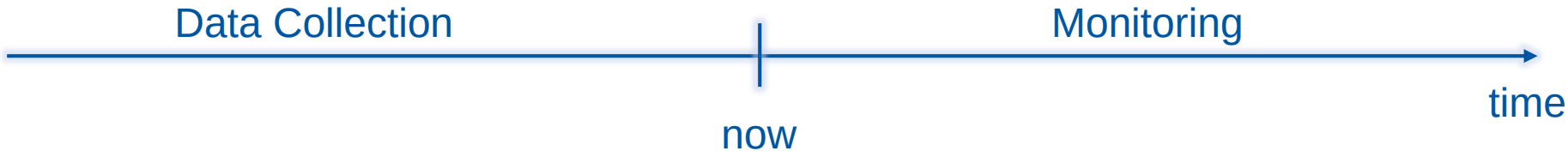
This ensures that

- the table schema is consistent across a year
- each year can be loaded quickly
- all years (provided the schema is the same) can be read at once (by skipping partition)
- the data is stored at HDFS allowing for an immediate processing on Spark cluster


Exploration
Data Collection
Modelling
Monitoring
interactive notebook
interactive compute
Jupyter Server
python
REST PySpark
Spark Hadoop
output notebook
storage
EOS
Tested notebook ready for scaling-up
parametrized notebook
parameterized script
scheduled compute
Apache Airflow
python
REST PySpark
Spark Hadoop
storage
hadoop
Well-structured data ready for modelling
storage
hadoop
interactive notebook
interactive compute
Jupyter Server
python
REST PySpark
Spark Hadoop
storage
EOS
Trained model ready for monitoring
parametrized notebook
parameterized script
scheduled compute
Apache Airflow
python
REST PySpark
Spark Hadoop
storage
hadoop

Signal Monitoring Pipeline





Single Statement Policy:

- at each stage only **one python** statement required to perform an operation
- this is **already** provided by Apache Airflow, Spark, and Hadoop
- we complement it with the **pyeDSL** (single statement to query, compute features, etc.)

The same infrastructure for data collection and monitoring scheduling

pyeDSL speaks AFT

Our pyeDSL provides a generic way of performing PM and NXCALS queries. Lately, the language has been extended to support AFT database:

- Context query

```
QueryBuilder().with_aft(session) \
    .context_query('faults/states')
```

	id	name
0	NON_BLOCKING_OP	Non-Blocking OP

- Fault query

```
QueryBuilder().with_aft(session) \
    .with_duration(t_start='2016-01-13T00:00:00Z', t_end='2016-05-13T00:00:00Z') \
    .fault_query(acceleratorId='LHC', accessNeeded=True)
```

	acceleratorName	acceleratorPropertyInstances	accessNeeded	description	displayLabel	duration	effectiveDuration	endTime
0	LHC	[{'propertyName': 'Time in Fill', 'value': '02...	False	access sectors 4,5,6,7 and LHCb indicate "blue...	None	4890000	4890000	2016-05-12T16:06:12Z

Quick development due to detailed documentation and a solid API: https://aft.cern.ch/docs
An example notebook with all types of queries:
https://gitlab.cern.ch/LHCData/lhc-sm-api/-/blob/master/lhcsmapi/dbsignal/aft/AFT.ipynb

Summary

Apache Airflow provides a user-friendly WebUI for defining (in python) signals, type of monitoring, time intervals, warning/alarm levels, e-mail address of person to be notified. Also:

1. **long-running** data collection jobs (PM, NXCALS, **AFT**) with auto-restart in case of failures
2. **on-demand** execution of monitoring applications
3. **scheduled** execution of monitoring applications
4. *scheduled notebook execution and report generation (e.g. FPA analysis) - in progress*

Find out more at: https://cern.ch/sigmon/about

Summary

Apache Airflow provides a user-friendly WebUI for defining (in python) signals, type of monitoring, time intervals, warning/alarm levels, e-mail address of person to be notified. Also:

1. **long-running** data collection jobs (PM, NXCALS, **AFT**) with auto-restart in case of failures
2. **on-demand** execution of monitoring applications
3. **scheduled** execution of monitoring applications
4. *scheduled notebook execution and report generation (e.g. FPA analysis) - in progress*

The results of data collection are stored in **Hadoop** enabling system modelling in **SWAN**.

Our datasets will be openly available on our website: https://cern.ch/sigmon/datasets

Apache Airflow **is not yet** officially supported by IT. However, the NXCALSOperator and a Docker image are already backed up by our colleagues.

Find out more at: https://cern.ch/sigmon/about


CERN

Notebook-Centric Infrastructure