WCSS

Data Lineage in High-Performance Computing Environments

Politechnika Wrocławska

The authors of the presentation:
Mateusz Tykierko, Ula Lukierska

WCSS

- Wroclaw Centre for Networking and Supercomputing is organization unit of Wroclaw University of Science and Technology.

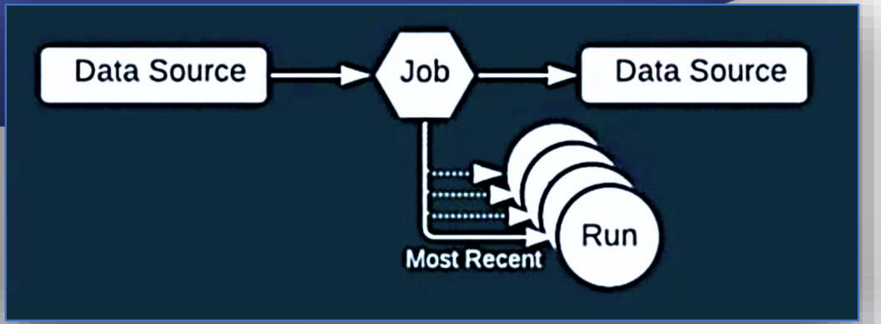
- WCSS was established on 21st of December 1994
- Origins date back to the Wrocław University of Technology Computing Centre, founded in 1972.
- 70+ staff

WCSS

Main tasks:

- operation and development of the Wrocław Academic Computer Network (WASK);
- operation and development of high performance computing services (HPC);
- operation and development of network services for Polish scientific community, industry and public sector
- operation and development of IT security services

the concept of DATA LINEAGE



In the context of data systems,
data lineage helps answer questions like:

- **Where** did this data **come from**?
- How was it **transformed** along the way?
- **Who accessed** or **modified** it?
- What decisions were **made based on this data**?

Imagine you're tracking the **history of a data** from its **initial input** (input file)to **its final** form(output file).
Data lineage is like **tracing that dataset's journey step by step**.
You start by identifying the **original data sources** (like input files),
then follow **each processing step** (like algorithms or transformations- JOB or TASK via RunEvent) until you reach the final output (a report or analysis in output file).

Data LINEAGE –help **managing & overseeing** the use, integrity, security of data

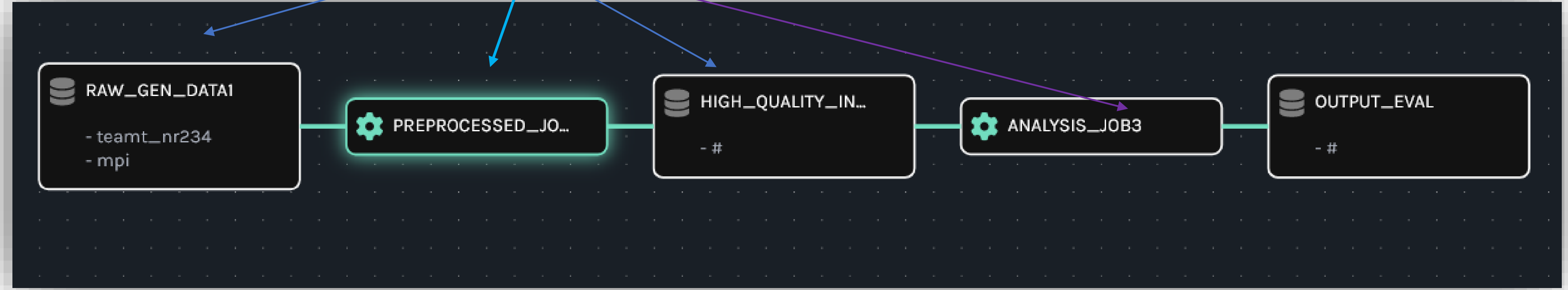
Data lineage is essential for effective **data stewardship** because it provides a clear & **traceable view** of how data is used within an organization with a focus on:

- **Data Provenance** by providing a detailed account of where the data originated, how it has been processed, and where it ends up.
- **Tracking&Transparency** by *documenting* the flow of data through various systems, applications, and processes. /***monitor*** data movements, identify bottlenecks/
- **Compliance&Regulation** by *documenting* data lineage, organizations can demonstrate compliance with regulatory requirements
- **Data Governance**: by providing *visibility* into data flows, dependencies, relationships, it allows to establish and enforce policies, standards, and controls around data usage
- **Auditability&Accountability**: by providing a clear trail of how data is collected, processed, and used. This trail allows organizations to trace back and verify data handling practices for internal and external audits.

User case: Lab data lifecycle

Data Collection and Preprocessing: The lab collects raw genomic data (INPUT DATA) from various sources, such as DNA sequencing machines or public databases.

This data is then **preprocessed** (ONE JOB)to remove noise, errors, and irrelevant information, ensuring **high-quality input**(output – can be input for another JOB) for further **analysis(**another JOB).



Source image: https://marquezproject.ai/about

Politechnika Wrocławska

WCSS

User case: Lab data lifecycle

- **Genomic Analysis:** Scientists perform various analyses(JOBS)on the preprocessed genomic data to extract meaningful insights.

- **Data Lineage in Analysis**: **track** the source of data used in each step. (if a particular genetic variant is identified as significant, scientists need to trace back to the original data source)

- **Publication &Collaboration**: Data lineage helps in providing **integrity, transparency**,& **reproducibility** of data/research by clearly documenting the data sources, preprocessing methods, and analysis techniques used.
  It helps in facilitating collaboration&knowledge sharing within the scientific community.

- **Reproducibility**: Other research teams or collaborators may want to replicate or build upon the findings of the science lab.

Data Lineage - job

- Lineage denotes **the provenance of data**, including its origins, transformations, and movements throughout Data lifecycle.
- In the context of lineage, a **job** typically refers to a specific **task** or **process** that manipulates or interacts with data. This could include ***data processing tasks*** such as data extraction, transformation, loading (ETL), data analysis, model training
- The **lineage of a job** would then describe **the relationship** between the **input data**, **the job itself**, and the resulting **output data**, detailing how the data is transformed or affected by the job's execution.

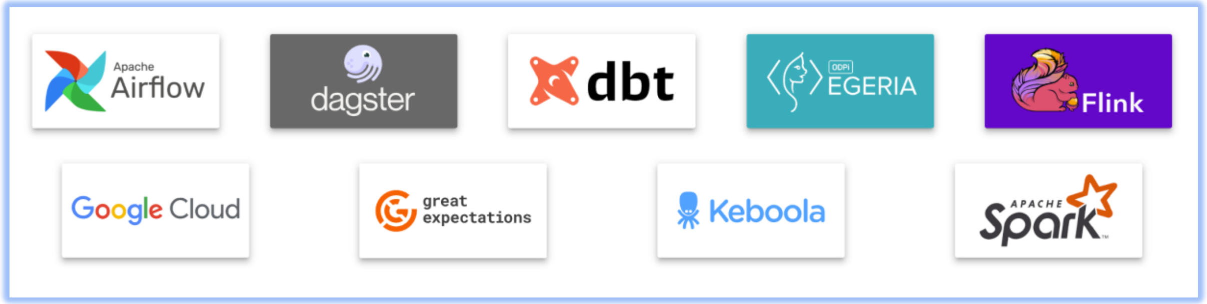
OpenLineage is an *Open Standard* for lineage metadata collection designed to record metadata for a ***job*** in execution

OpenLineage defines a generic model of **dataset, job, run** entities uniquely identified using consistent **naming strategies.**

Question

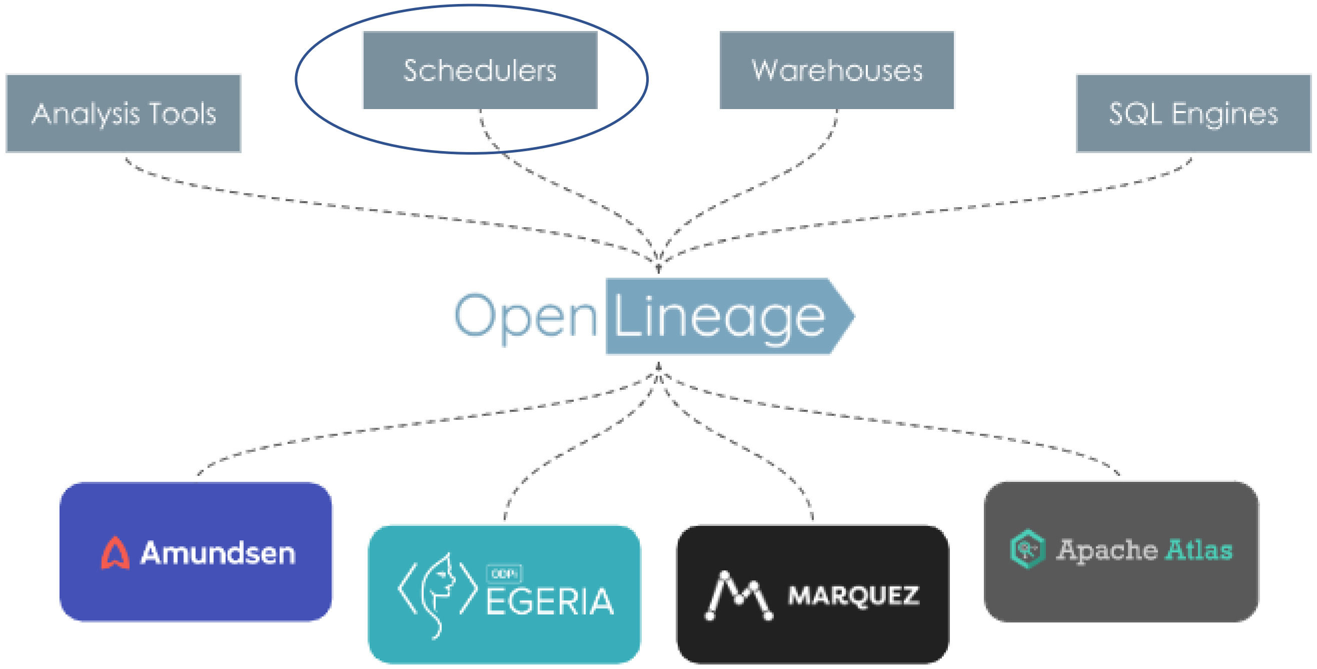
How can OpenLineage be utilized in an HPC environment?

OpenLineage Ecosystem Backend/Producers Data Integration



WITH OPENLINEAGE:

LSF/PBS/SGE/Slurm



Compatibility data sources are known to work with each integration ex.g Airflow

Source image: https://openlineage.io/docs

HPC centers

Polish HPC resources - 2023

CI TASK

TRYTON+
634 nodes, **30 432** cores, **166 TB** RAM,
2,82 PFLOPS

TRYTON
1607 nodes, **38 568** cores, **218 TB** RAM
1,79 PFLOPS

Cyfronet

Ares
788 nodes, **37 824** cores, **200 TB** RAM,
9 nodes 8xGPU V100
4,0 PFLOPS

Athena
7,7PFLOPS

ICM

Okeanos
1084 nodes, **26 016** cores, **138TB** RAM
1,08PFLOS

Topola
223 nodes, **6 244** cores, **18TB** RAM
0,49 PFLOPS

NCBJ

CIS
31 640 cores, **183TB** RAM
1,05 PFLOS

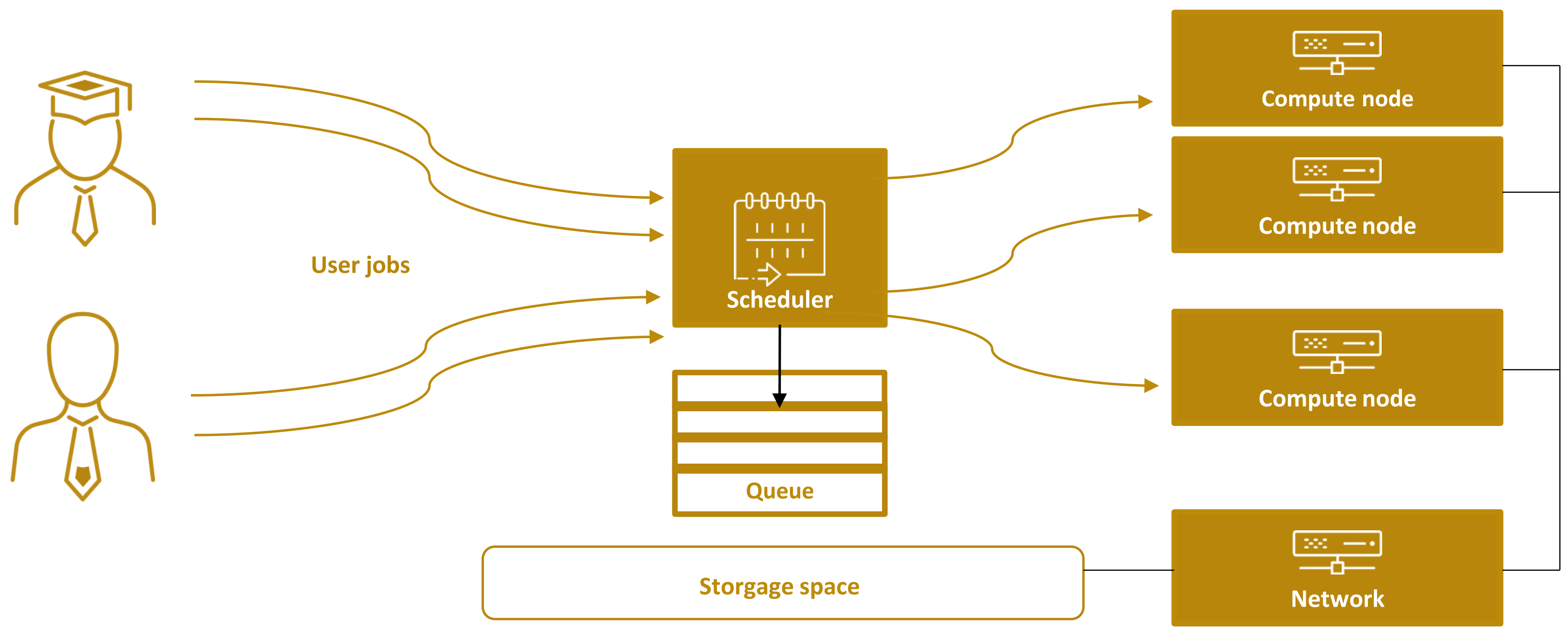
PCSS

ALTAIR
1320 nodes, **63 360** cores, **300 TB** RAM, **9** nodes 8xGPU V100
5,9 PFLOPS

WCSS

Bem2
506 nodes, **24 288** cores, **141 TB** RAM, GPU A100
2,2PFLOPS

HPC cluster


User jobs
Scheduler
Queue
Storgage space
Compute node
Compute node
Compute node
Network

OpenLineage CoreModel

- Each Run State Update can include detail about **the Job**
  A **run** is a particular instance of a job. with a **unique identifier /run uuid/,** that helps unite the events that represent the changes of state through time.
- **Jobs** are identified by a **unique** name **within a** namespace**.**
  Jobs are expected to evolve over time and their changes can be captured through **Run State Updates.**
  Job is a process that consumes or produces **Datasets.** ex.g *Python script is the Job.* or *Rscript.R ,sub-gaussian*
- A **facet** is an atomic piece of metadata attached to one of the core entities.

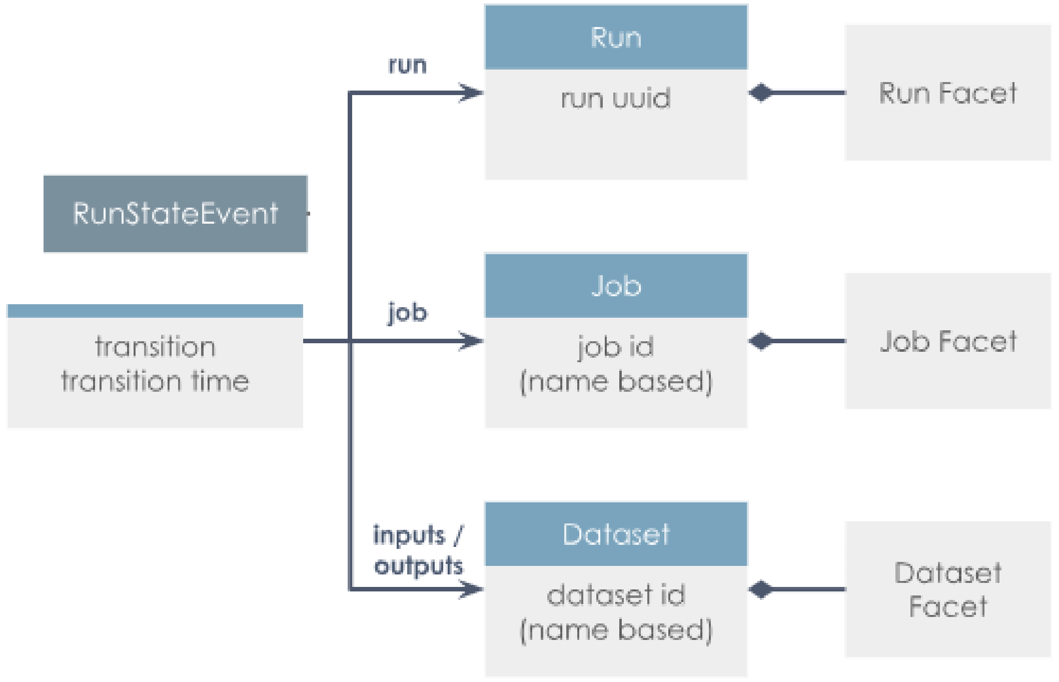


Image src : https://openlineage.io/docs

Key OpenLineage Concepts

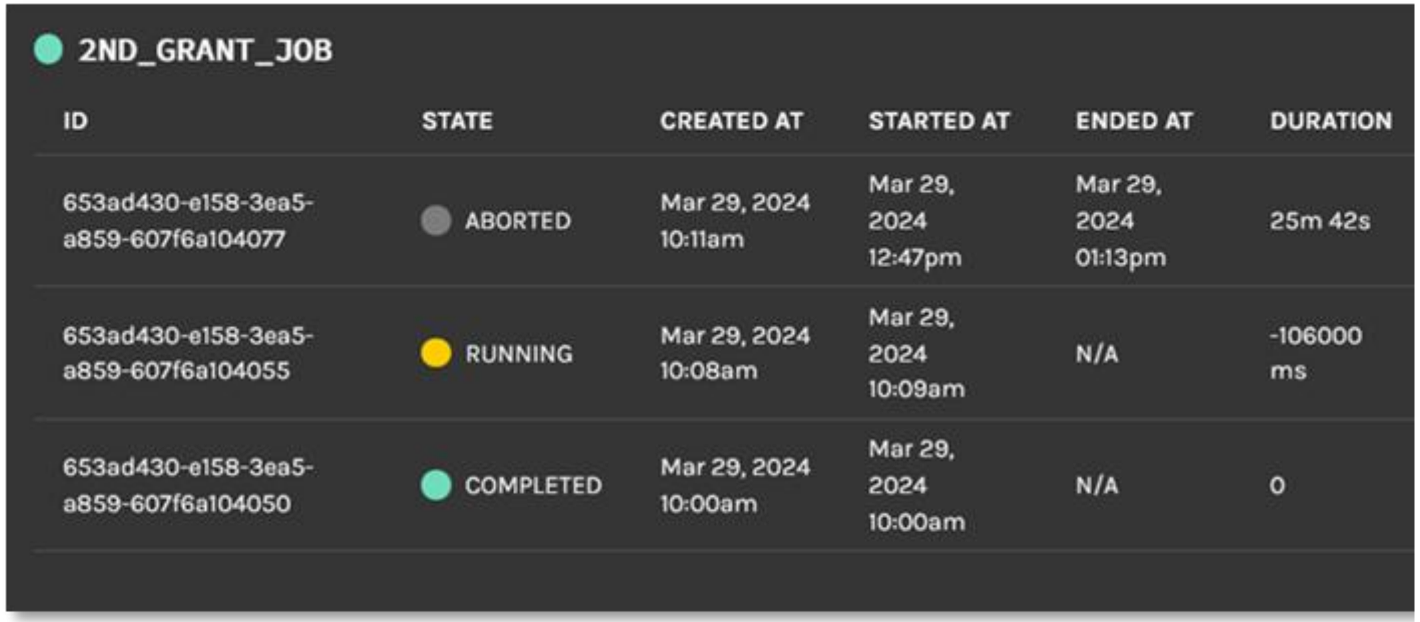
A **job** is the highest level of abstraction, and represents some type of process that produces **datasets**.
A **dataset** is an abstract representation of data.

All jobs have **state.** That is, they progress and change through time, every run begins with a **START** state and ends with a **COMPLETE, ABORT, or FAIL** state.

Finally **facets** are additional metadata that can be attached to either a **job, dataset, or run** to further describe these objects.

Run cycle is likely to have at least **two** Run State Updates

Usually, the first Run State for a Job would be START &the last would be COMPLETE.



2ND_GRANT_JOB

ID	STATE	CREATED AT	STARTED AT	ENDED AT	DURATION
653ad430-e158-3ea5-a859-607f6a104077	● ABORTED	Mar 29, 2024 10:11am	Mar 29, 2024 12:47pm	Mar 29, 2024 01:13pm	25m 42s
653ad430-e158-3ea5-a859-607f6a104055	● RUNNING	Mar 29, 2024 10:08am	Mar 29, 2024 10:09am	N/A	-106000 ms
653ad430-e158-3ea5-a859-607f6a104050	● COMPLETED	Mar 29, 2024 10:00am	Mar 29, 2024 10:00am	N/A	0

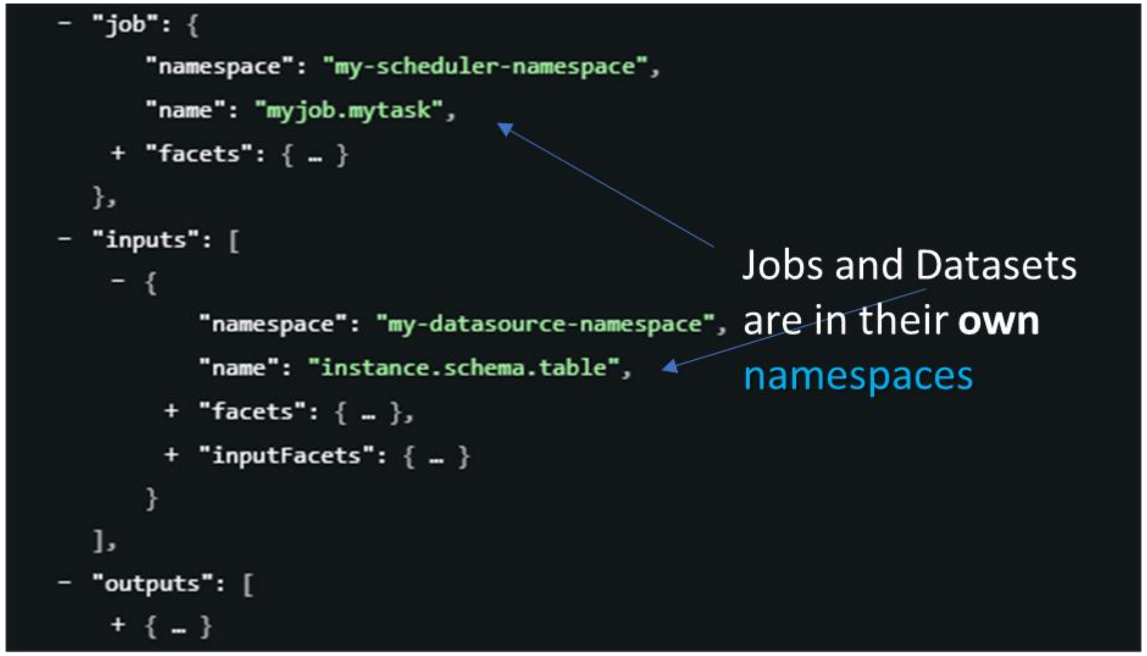
NAMING - is key to the production of useful lineage

How is a job's **namespace** derived?

Each execution of job is captured
as a RunEvent with corresponding **metadata.**

A Run event identifies the **Job** it is an instance of by providing **the job's unique** identifier**.**

The Job identifier is composed of a Namespace and **a Name.**



```
- "job": {
      "namespace": "my-scheduler-namespace",
      "name": "myjob.mytask",
    + "facets": { … }
  },
- "inputs": [
    - {
          "namespace": "my-datasource-namespace",
          "name": "instance.schema.table",
        + "facets": { … },
        + "inputFacets": { … }
      }
  ],
- "outputs": [
    + { … }
```

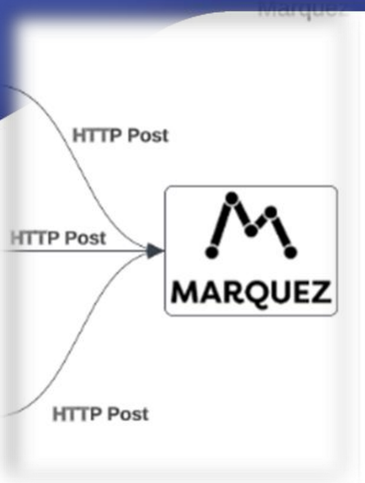
The **Namespace** is the root of the naming hierarchy.

The job **name** is constructed to identify the job within that namespace.

Jobs and Datasets are in their **own namespaces.**

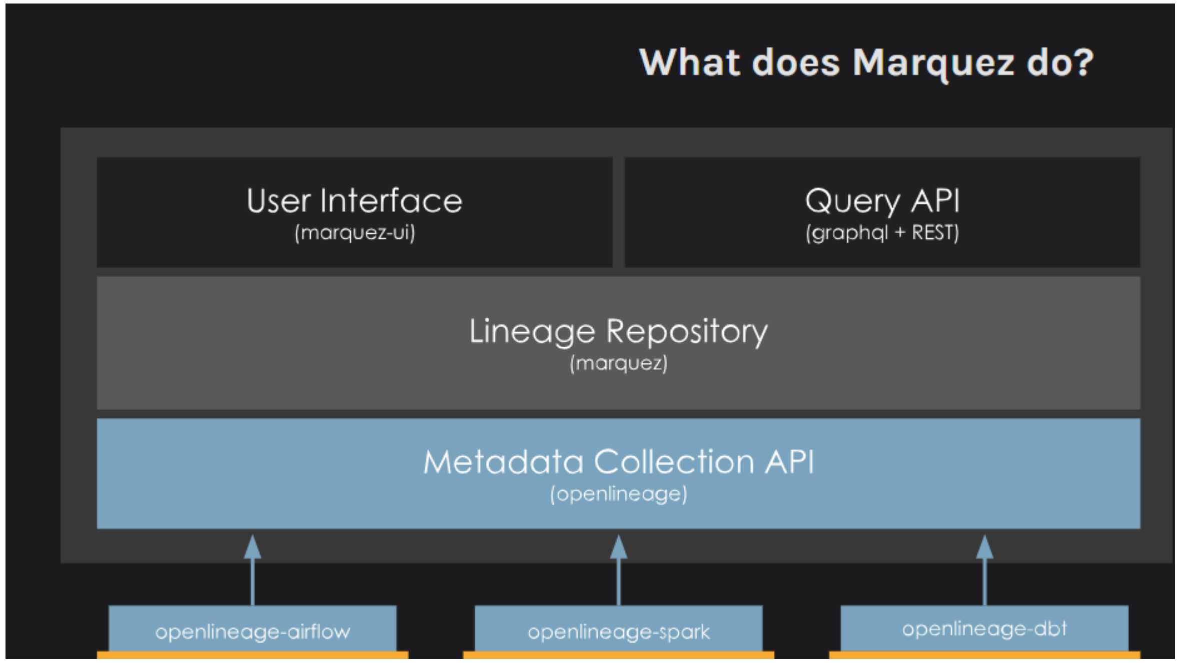
- **Job namespaces** are **related to their schedulers**.
- The **namespace for a dataset** is the **unique name for its datasource**

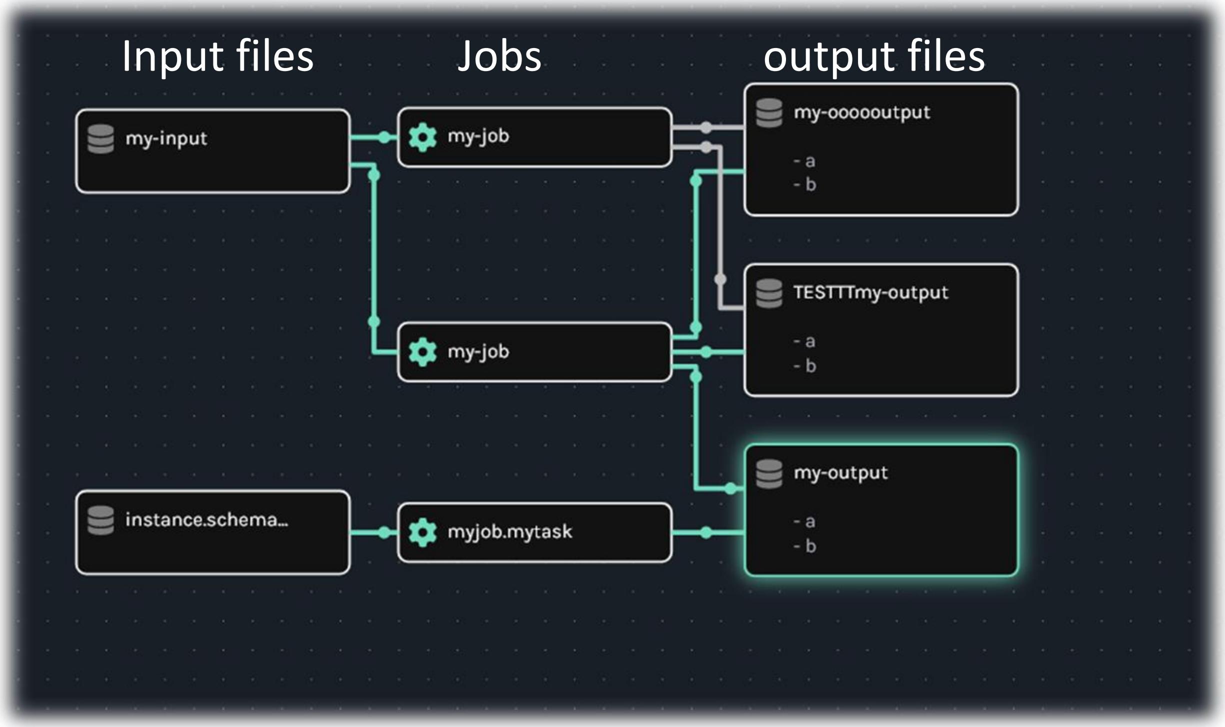
visualize a data ecosystem's metadata



Marquez is an LF AI & DATA Foundation project to **collect, aggregate,** and **visualize** a data ecosystem's metadata.

It is the reference implementation of the OpenLineage API centralizes dataset lifecycle management

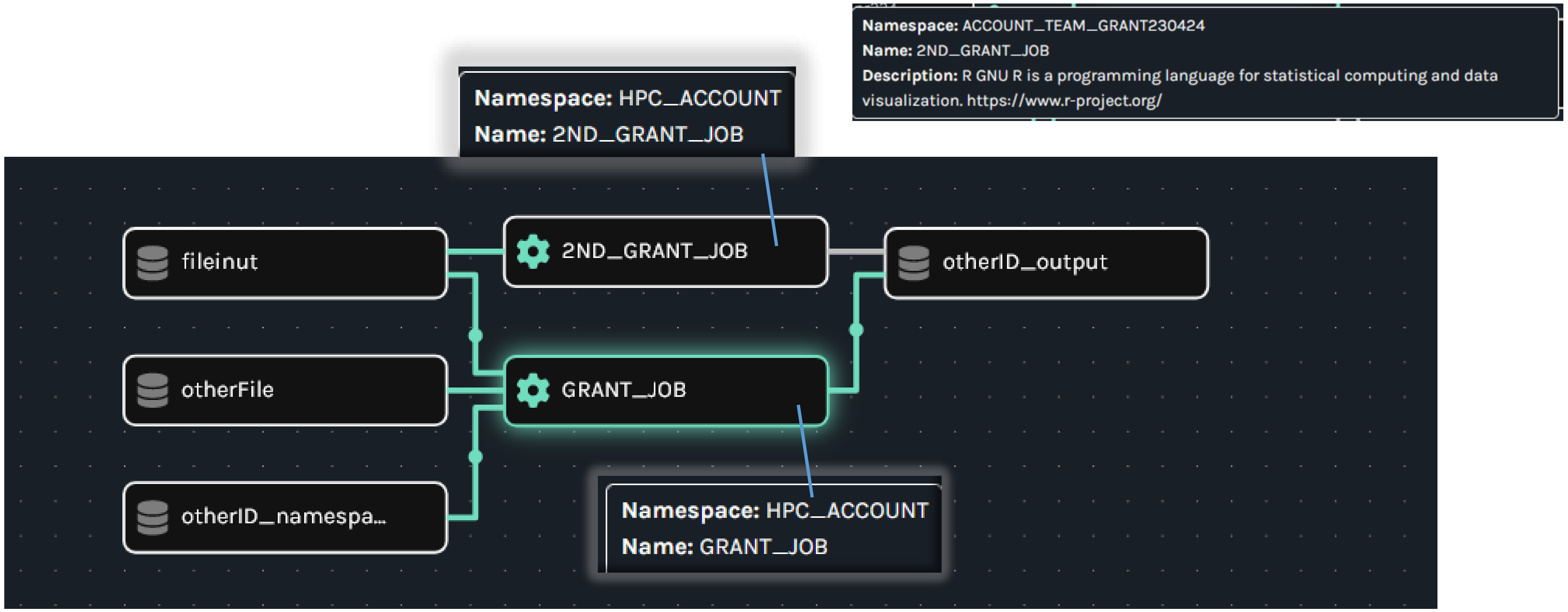




Source image: https://marquezproject.ai/about

OpenLineage the same Job namespace Different Job name

The decision of what will count as a **namespace** and
what as a **name** is a matter of choice – grant, software,account, team
We can specify which software we use in a Documentation facet of the job



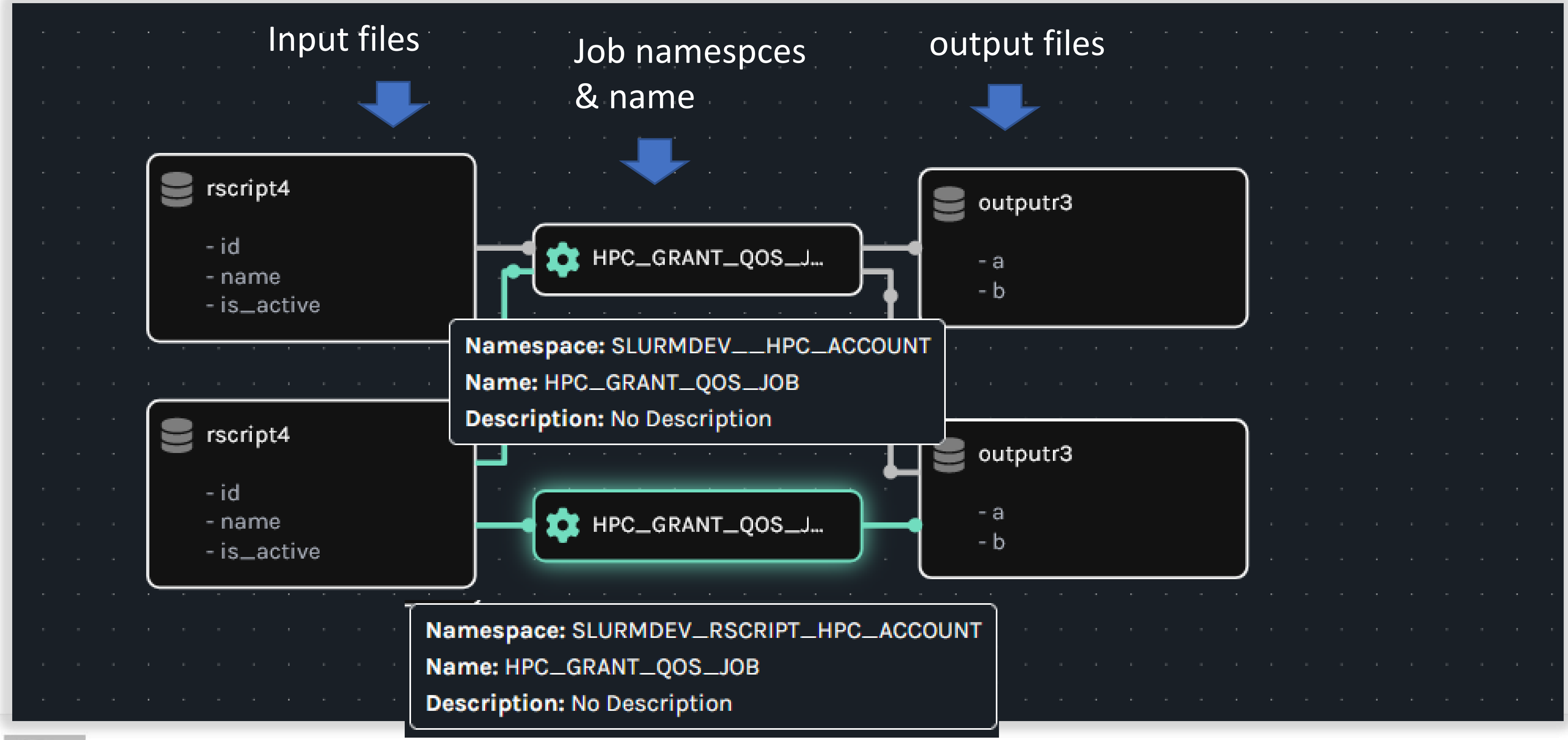
Namespace: ACCOUNT_TEAM_GRANT230424
Name: 2ND_GRANT_JOB
Description: R GNU R is a programming language for statistical computing and data visualization. https://www.r-project.org/

Namespace: HPC_ACCOUNT
Name: 2ND_GRANT_JOB

Namespace: HPC_ACCOUNT
Name: GRANT_JOB

OpenLineage different Jobs- NAMESPACE

Different **Namespaces** – Grants, Users, Account, project Number Id but **the same** NAME JOB (the computational parameters and the job script)

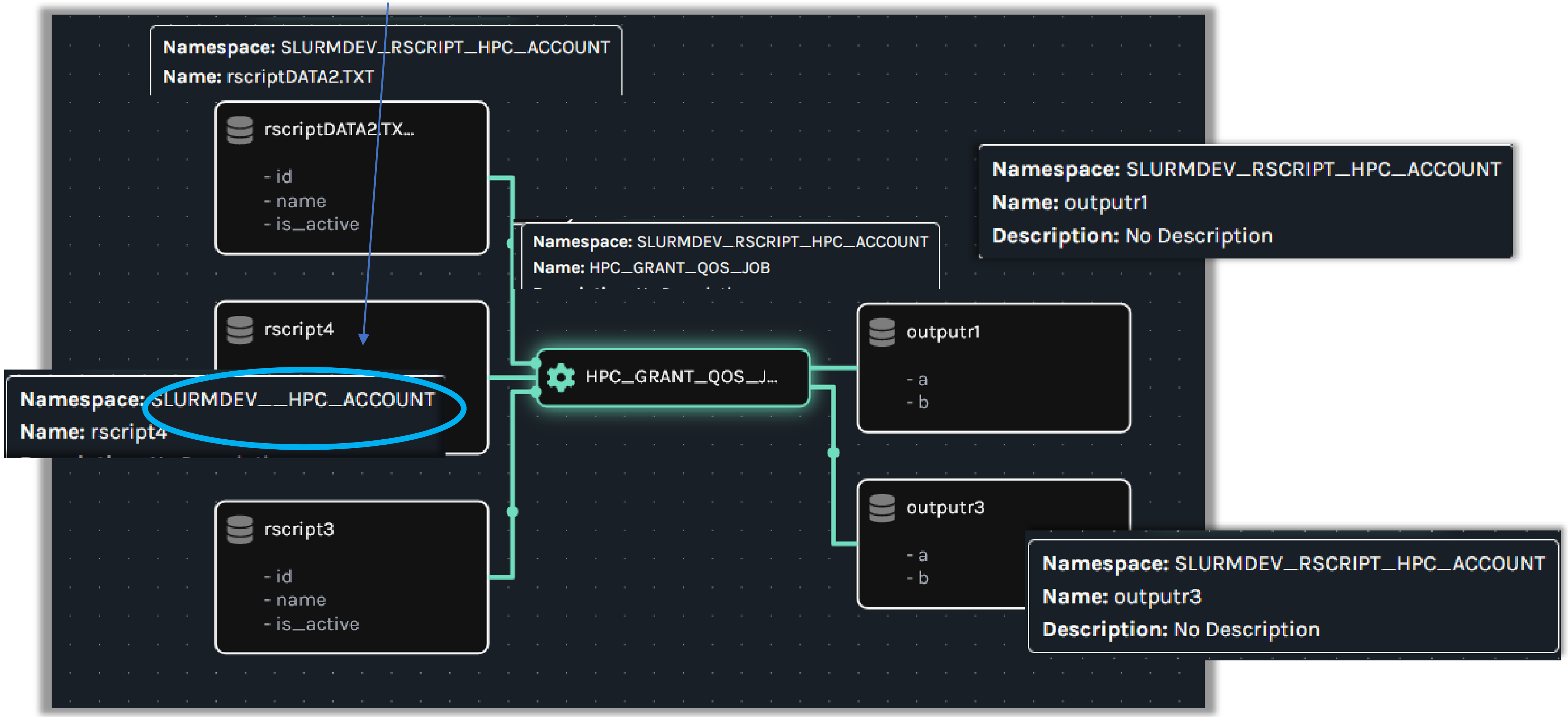


Namespace: SLURMDEV__HPC_ACCOUNT
Name: HPC_GRANT_QOS_JOB
Description: No Description

Namespace: SLURMDEV_RSCRIPT_HPC_ACCOUNT
Name: HPC_GRANT_QOS_JOB
Description: No Description

the same JOB –namespace &name

- Different INPUT data (**different namespaces**) and Output files (**the same** namespace)



Namespace: SLURMDEV_RSCRIPT_HPC_ACCOUNT
Name: rscriptDATA2.TXT

rscriptDATA2.TX...
- id
- name
- is_active

Namespace: SLURMDEV_RSCRIPT_HPC_ACCOUNT
Name: HPC_GRANT_QOS_JOB

Namespace: SLURMDEV_RSCRIPT_HPC_ACCOUNT
Name: outputr1
Description: No Description

rscript4

Namespace: SLURMDEV__HPC_ACCOUNT
Name: rscript4

HPC_GRANT_QOS_J...

outputr1
- a
- b

rscript3
- id
- name
- is_active

outputr3
- a
- b

Namespace: SLURMDEV_RSCRIPT_HPC_ACCOUNT
Name: outputr3
Description: No Description

Politechnika Wrocławska

WCSS

OpenLineage Scope/Ecosystm

the challenges of collecting lineage metadata from schedulers
HPC queuing systems (the most used)SLURM lack native/generic integration with openlineage system (It'd allow for Dataset visualization Table/Column-level lineage)

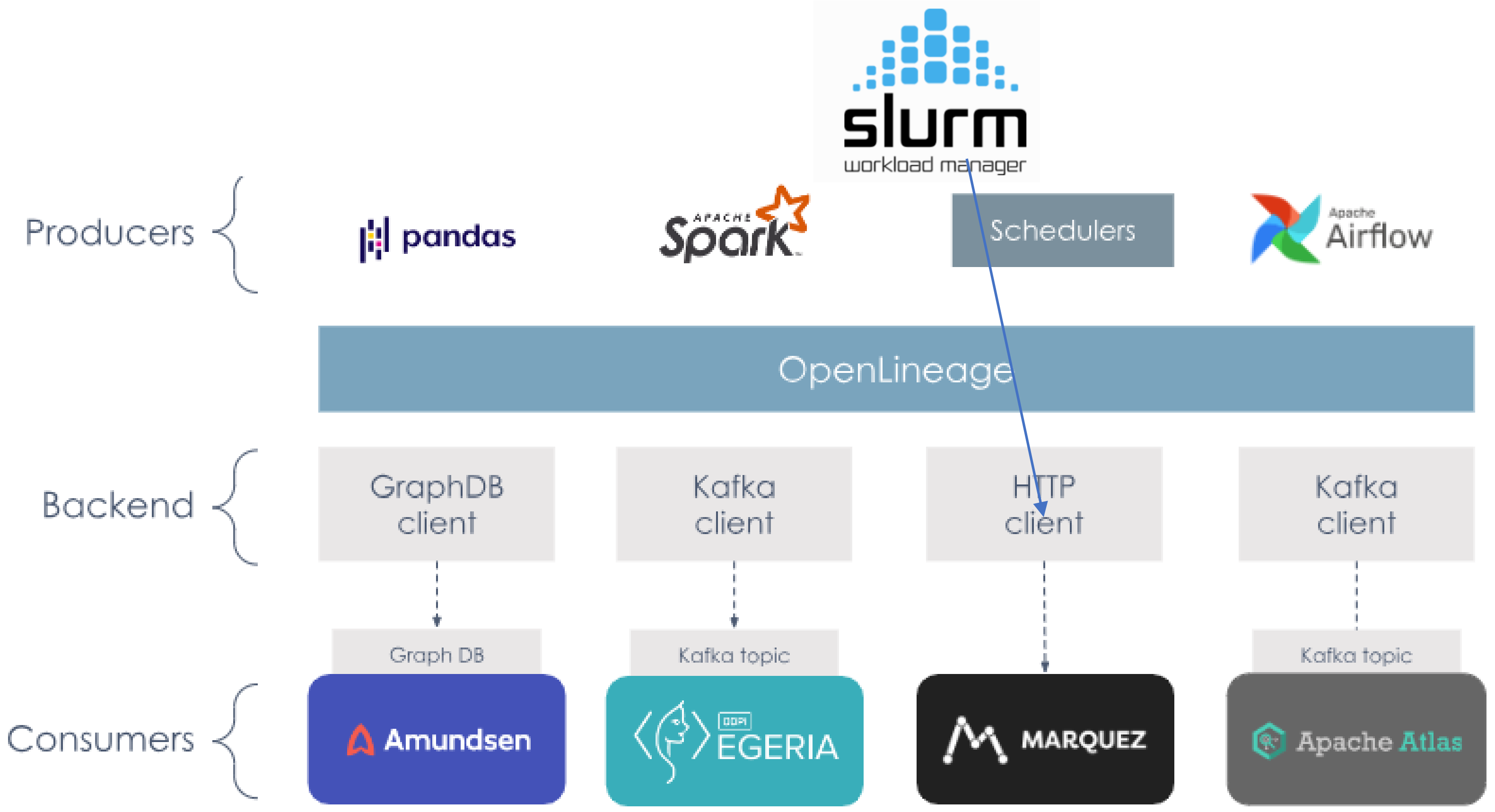
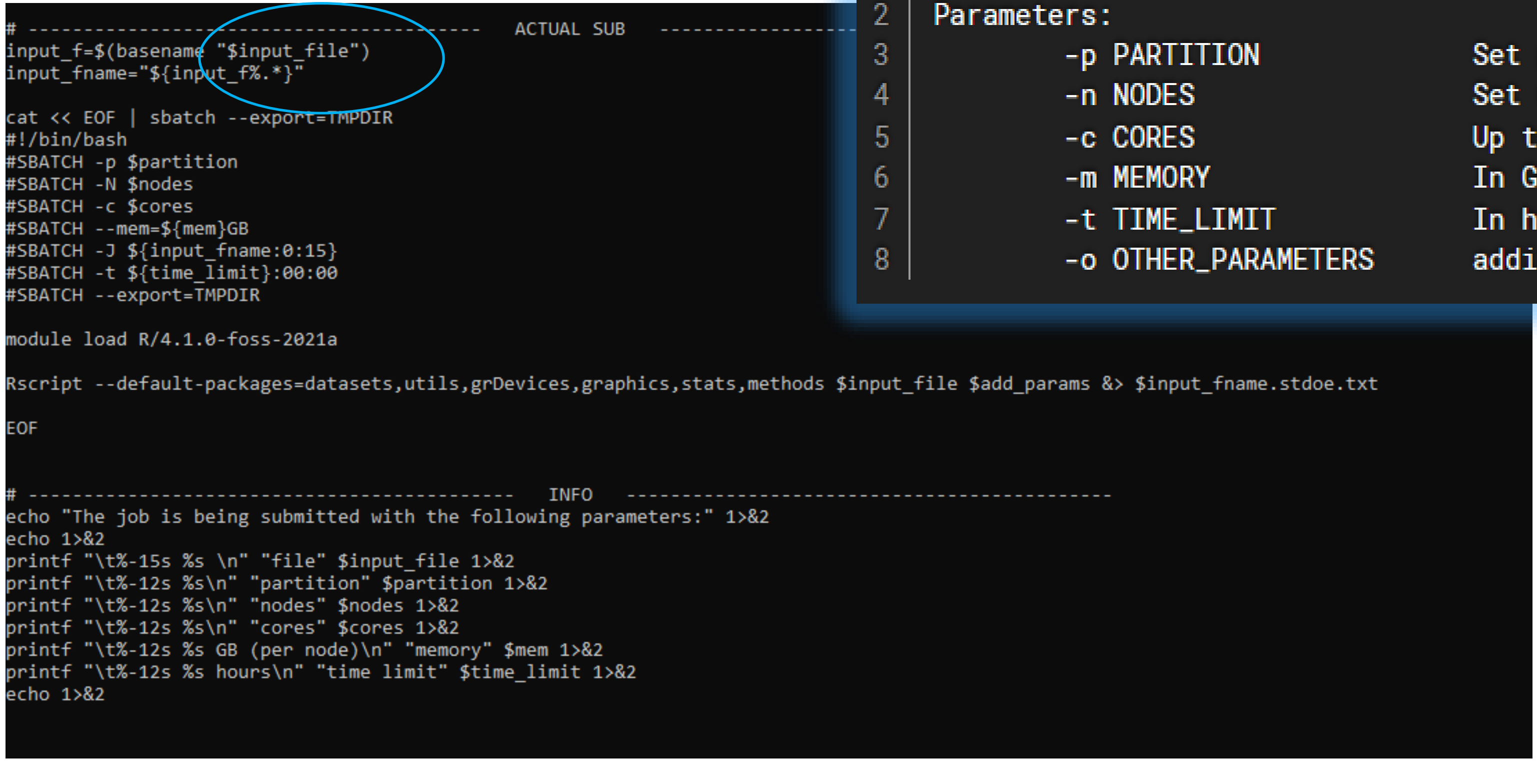
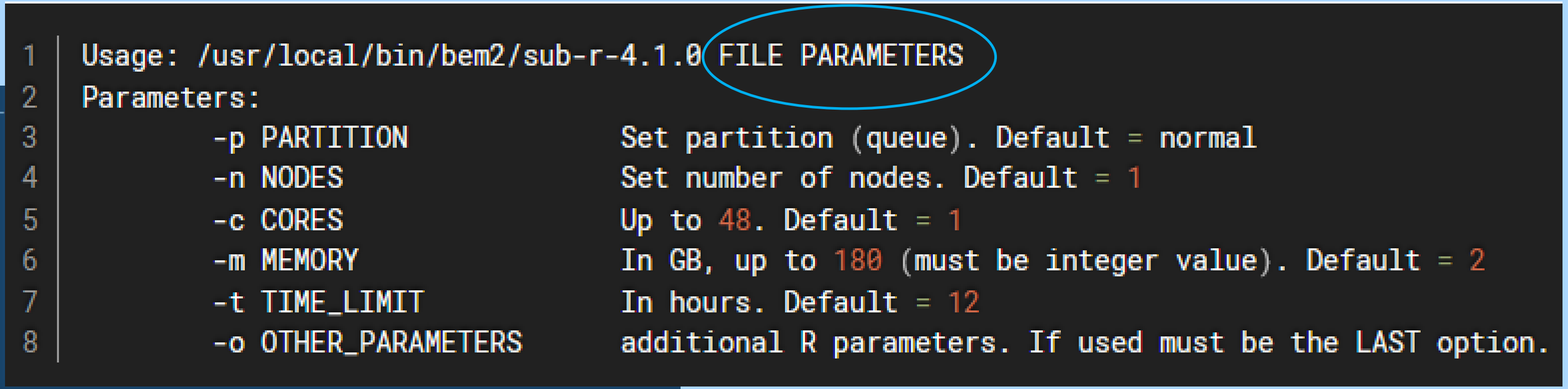


Image src: https://openlineage.io/docs

Example –subscript PARAMETERS



```
# ------------------------------------------   ACTUAL SUB   ----------------
input_f=$(basename "$input_file")
input_fname="${input_f%.*}"

cat << EOF | sbatch --export=TMPDIR
#!/bin/bash
#SBATCH -p $partition
#SBATCH -N $nodes
#SBATCH -c $cores
#SBATCH --mem=${mem}GB
#SBATCH -J ${input_fname:0:15}
#SBATCH -t ${time_limit}:00:00
#SBATCH --export=TMPDIR

module load R/4.1.0-foss-2021a

Rscript --default-packages=datasets,utils,grDevices,graphics,stats,methods $input_file $add_params &> $input_fname.stdoe.txt

EOF


# ---------------------------------------------   INFO   -------------------------------------------
echo "The job is being submitted with the following parameters:" 1>&2
echo 1>&2
printf "\t%-15s %s \n" "file" $input_file 1>&2
printf "\t%-12s %s\n" "partition" $partition 1>&2
printf "\t%-12s %s\n" "nodes" $nodes 1>&2
printf "\t%-12s %s\n" "cores" $cores 1>&2
printf "\t%-12s %s GB (per node)\n" "memory" $mem 1>&2
printf "\t%-12s %s hours\n" "time limit" $time_limit 1>&2
echo 1>&2
```



```
Usage: /usr/local/bin/bem2/sub-r-4.1.0 FILE PARAMETERS
Parameters:
        -p PARTITION            Set partition (queue). Default = normal
        -n NODES                Set number of nodes. Default = 1
        -c CORES                Up to 48. Default = 1
        -m MEMORY               In GB, up to 180 (must be integer value). Default = 2
        -t TIME_LIMIT           In hours. Default = 12
        -o OTHER_PARAMETERS     additional R parameters. If used must be the LAST option.
```

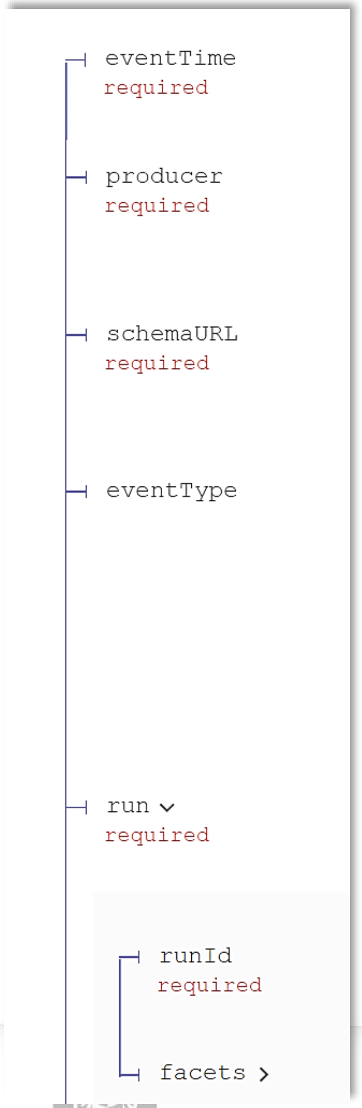
Which parameters must the **user provide/set**, and which can be automatically retrieved from **environmental variables from the Slurm system** regarding the job ?

Politechnika Wrocławska

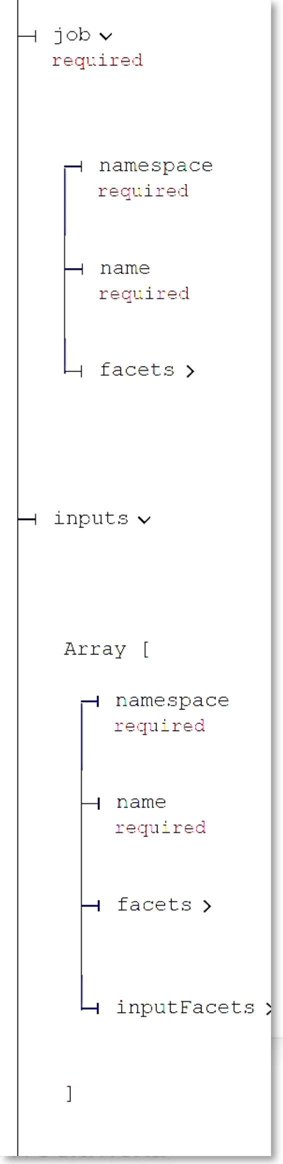
WCSS

Send an RunEvent > API required = fields that cannot be null

- REQUEST BODY SCHEMA: application/json   Record a single EVENT required:



```
┤ eventTime
  required
┤ producer
  required
┤ schemaURL
  required
┤ eventType
┤ run ⌄
  required
    ┤ runId
      required
    └ facets ›
```



```
┤ job ⌄
  required
    ┤ namespace
      required
    ┤ name
      required
    └ facets ›
┤ inputs ⌄
    Array [
      ┤ namespace
        required
      ┤ name
        required
      ┤ facets ›
      └ inputFacets ›
    ]
```

QUESTION which fields can be automatically retrieved from **environmental variables from the SLURM system** regarding the job ?

- **runId**: UUID format 870492da-ecfb-4be0-91b9-9a89ddd3db90
  SLURM_JOBID? 2588280
  SLURM_TASK_PID 1758804
- eventType: "START|RUNNING|COMPLETE|ABORT|FAIL|OTHER / If eventType is null - default OTHER
  It is required to issue 1 START event and 1 of [COMPLETE, ABORT, FAIL] event per run.
  Additional events with *OTHER* eventType can be added to the same run.
  SLURM The typical states are **PENDING, RUNNING, SUSPENDED, COMPLETING, COMPLETED**.
- EventTime ISO 8601 2024-03-25T09:48:06Z
  (conversion from Slurm default format : Unixtimestamp 1711376005
- Job: **Namespace/name** $SLURM_JOB_USER,$SLURM_JOB_ACCOUNT, $SLURM_OB_QOS
- Input//Output **Namespace/name** $SLURM_SUBMIT_DIR
- „**_producer":** "https://github.com/OpenLineage/OpenLineage/blob/v1-0-0/client",
- "**schemaURL**": "https://openlineage.io/spec/1-0 5/OpenLineage.json#/definitions/**RunEvent"**
- producer value is included in an OpenLineage request as a way to know how the metadata was generated.
  It is a URI that links to a source code SHA or the location where a package can be found.

SLURM envs OpenLinege Facets

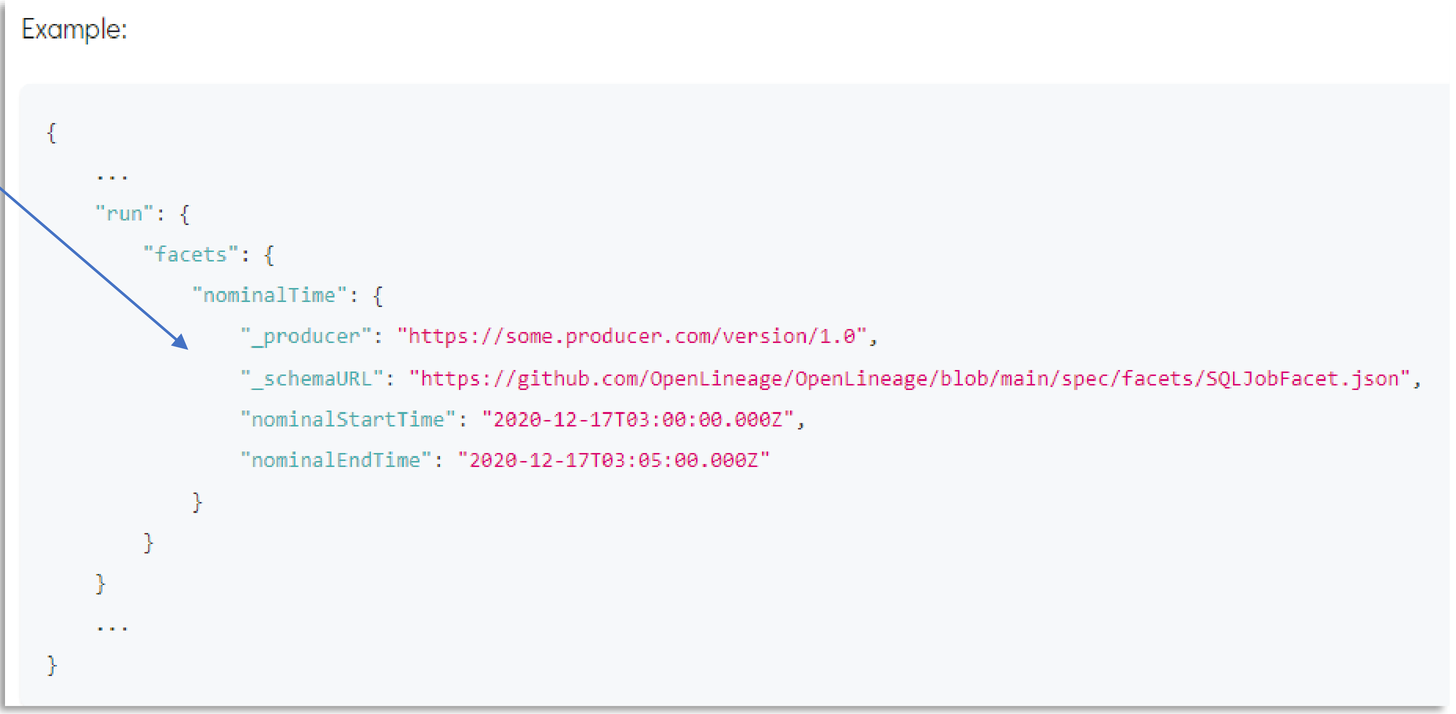
For example Run Facets Error

MessageNominal Time Facet = start/end time of the run. The nominal usually means the time the job run was expected to run (a scheduled time)
No such SLURM env variable exists for the time requested. Within the submission script, you can query the Slurm controller for the information with squeue

```
TIME=$(squeue -j $SLURM_JOB_ID -h --Format TimeLimit)
```

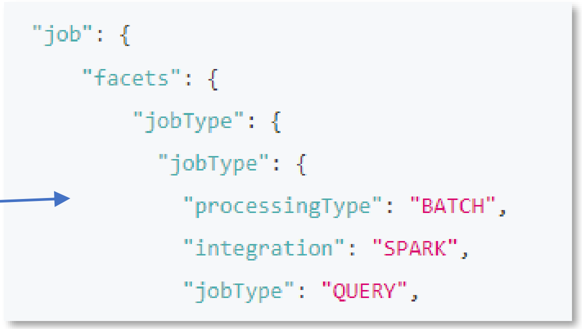
For example Job Facets
SQL/Ownership/Documentation

Job Type BUT **only**
SPARK/AIRFLOW/FLINK/DBT
what about Slurm



Example:

```
{
    ...
    "run": {
        "facets": {
            "nominalTime": {
                "_producer": "https://some.producer.com/version/1.0",
                "_schemaURL": "https://github.com/OpenLineage/OpenLineage/blob/main/spec/facets/SQLJobFacet.json",
                "nominalStartTime": "2020-12-17T03:00:00.000Z",
                "nominalEndTime": "2020-12-17T03:05:00.000Z"
            }
        }
    }
    ...
}
```



```
"job": {
    "facets": {
        "jobType": {
          "jobType": {
            "processingType": "BATCH",
            "integration": "SPARK",
            "jobType": "QUERY",
```

SUMMARY

- SLURM has limited support for data lineage for general-purpose computing
- Namespace and workflow definition are crucial – define in DMP
- Artifacts from SLURM can be used after some tranfromation
- It is possible to provide limited lineage data for every job using job prolog and epilog
- Most of the work has to be done by the user

The authors of the presentation:
Mateusz Tykierko,
Ula Lukierska

WCSS®

Image sources:
https://openlineage.io/docs
https://openlineage.io/docs/spec/facets/run-facets/nominal_time
https://marquezproject.ai/about
https://slurm.schedmd.com/documentation.html

Thank you for your attention.

Politechnika Wrocławska