Dynamic hybrid workflows for Deep Learning on HPC infrastructure

Iacopo Colonnelli, Assistant Professor (RTD-A)
Università degli Studi di Torino
Member of the CWL Technical Team

ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

UNIVERSITÀ DI TORINO

ICTP
The Abdus Salam International Centre for Theoretical Physics

Reproducibility and FAIRness

Reproducibility

Reproducible Workflows
- Version control repositories
- Scripts / containerisation
- Recipes
- Accessible forcing data
- Attached DOIs

Standardised Assessment
- Common diagnostics
- Common code base

User/Community Value
- Publishing requirement
- Accelerate debugging & development
- Recognition for "non-standard" outputs
- Democratisation of skills
- Shared knowledge base

Reproducibility in Machine Learning

Improving Reproducibility in Machine Learning Research
(*A Report from the NeurIPS 2019 Reproducibility Program*)

Joelle Pineau JPINEAU@CS.MCGILL.C
School of Computer Science, McGill University (Mila)
Facebook AI Research
CIFAR

Philippe Vincent-Lamarre PHILVLAM@GMAIL.CON
Ecole de bibliothèconomie et des sciences de l'information,
Université de Montréal

Koustuv Sinha KOUSTUV.SINHA@MAIL.MCGILL.C
School of Computer Science, McGill University (Mila)
Facebook AI Research

Vincent Larivière VINCENT.LARIVIERE@UMONTREAL.C
Ecole de bibliothéconomie et des sciences de l'information,
Université de Montréal

Alina Beygelzimer BEYGEL@YAHOO-INC.COM
Yahoo! Research

Florence d'Alché-Buc FLORENCE.DALCHE@TELECOM-PARIS.FR
Télécom Paris,
Institut Polytechnique de France

Emily Fox EBFOX@CS.WASHINGTON.EDU
University of Washington
Apple

Hugo Larochelle HUGOLAROCHELLE@GOOGLE.COM
Google
CIFAR

A Step Toward Quantifying Independently Reproducible Machine Learning Research

Edward Raff
Booz Allen Hamilton

The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

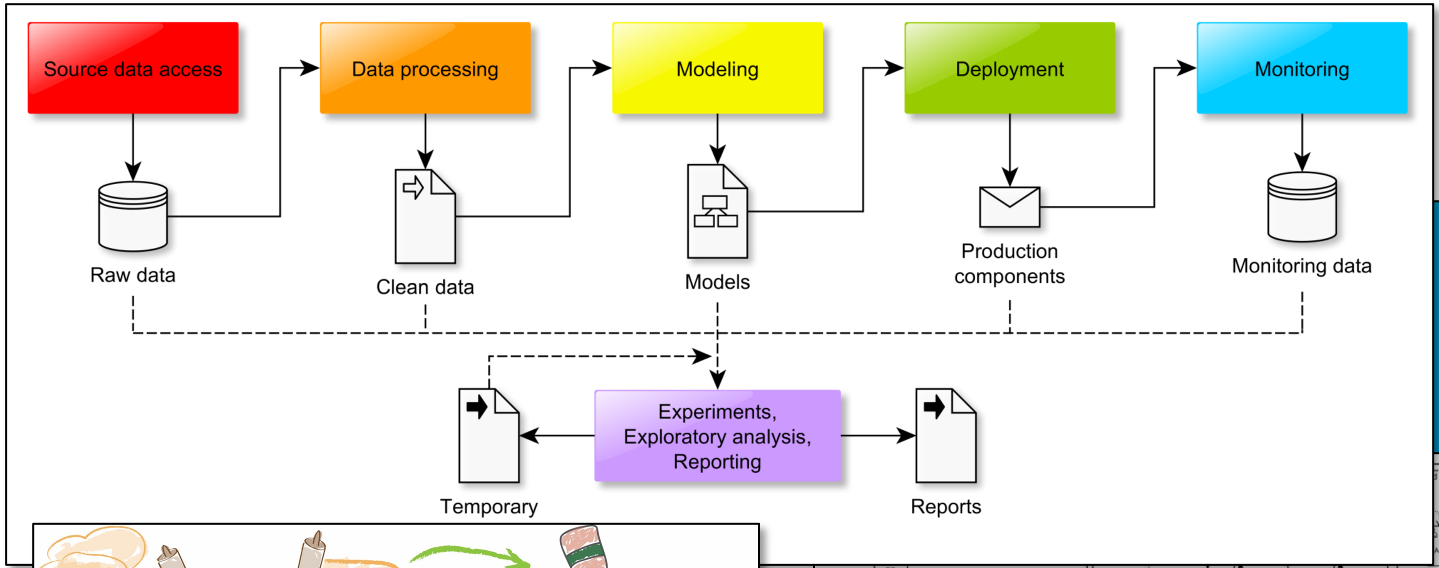
State of the Art: Reproducibility in Artificial Intelligence

Odd Erik Gundersen, Sigbjørn Kjensmo
Department of Computer Science
Norwegian University of Science and Technology

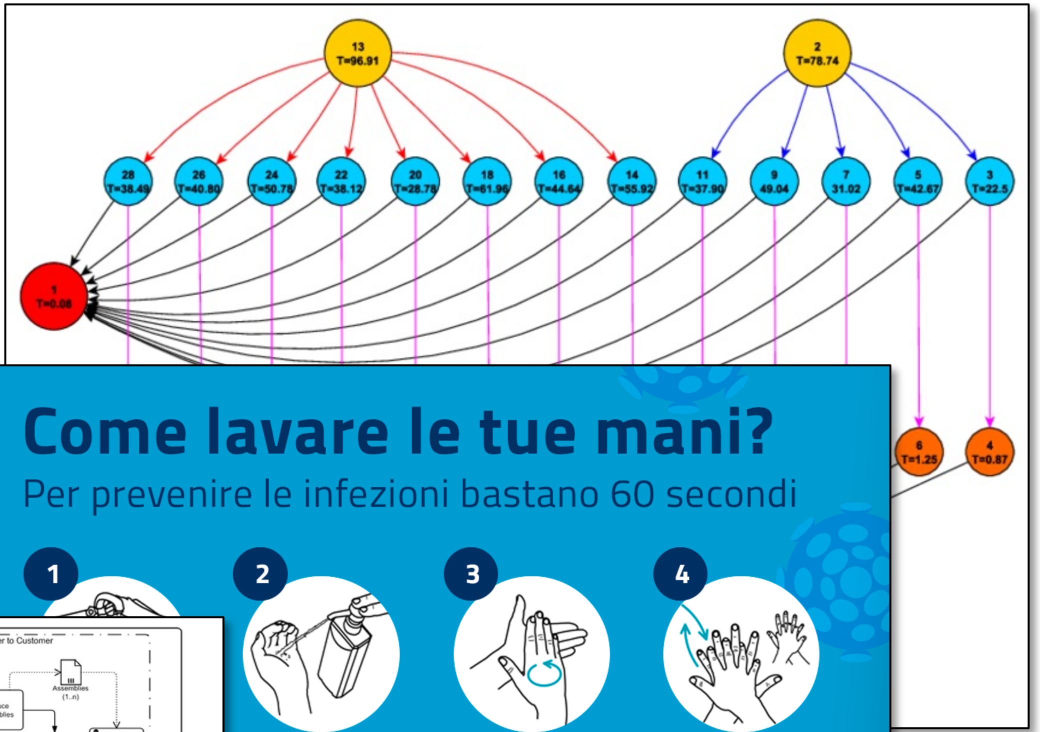
Workflows

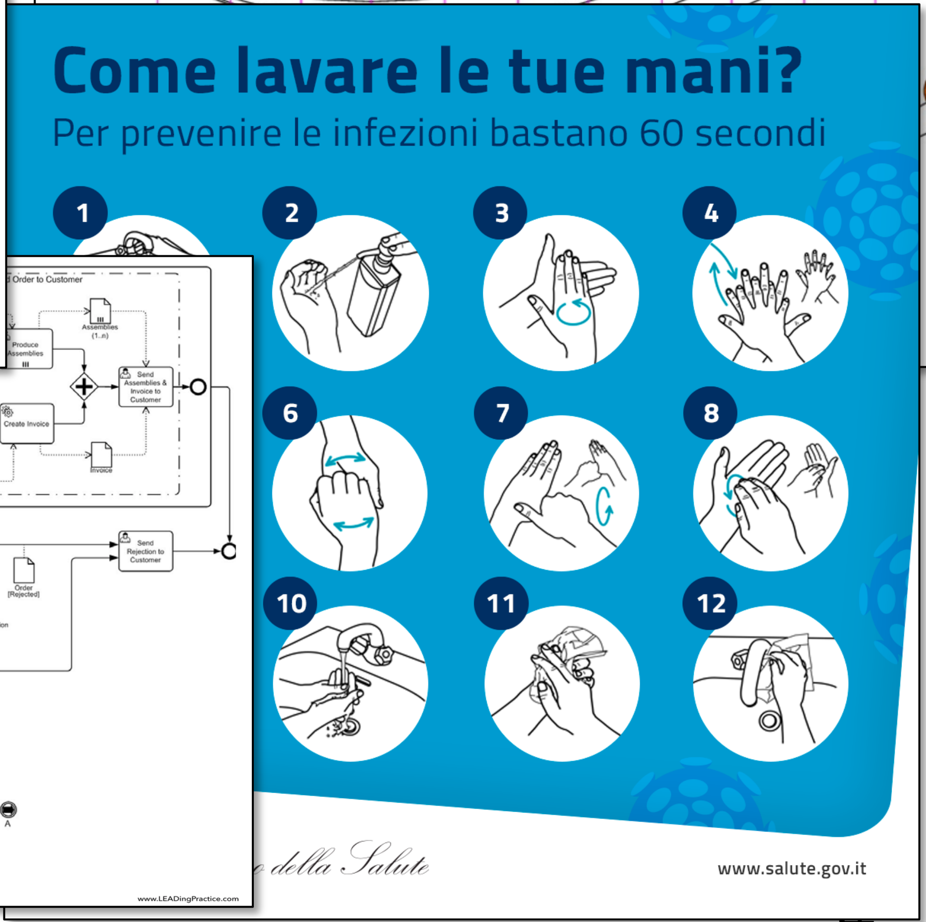
A **workflow** is an abstraction that models a complex and modular working process as a set of **steps** and their **inter-dependencies**.

Workflows

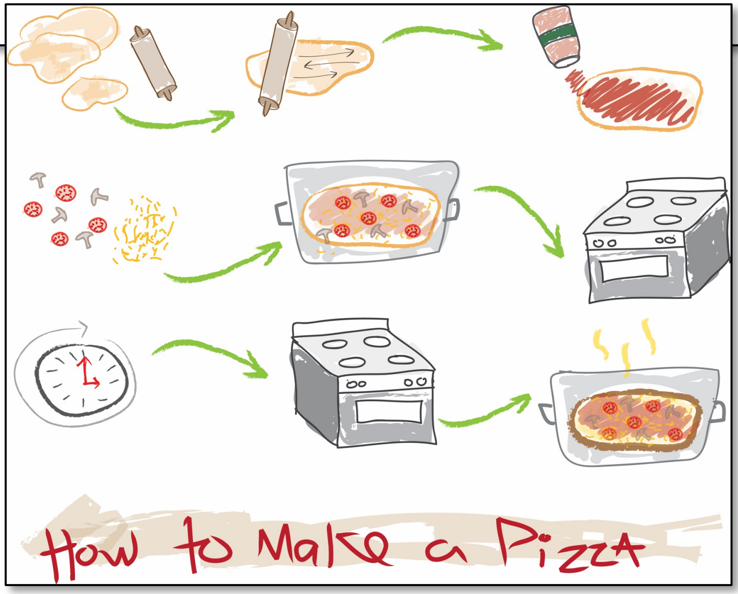


Source data access
Data processing
Modeling
Deployment
Monitoring
Raw data
Clean data
Models
Production components
Monitoring data
Experiments, Exploratory analysis, Reporting
Temporary
Reports

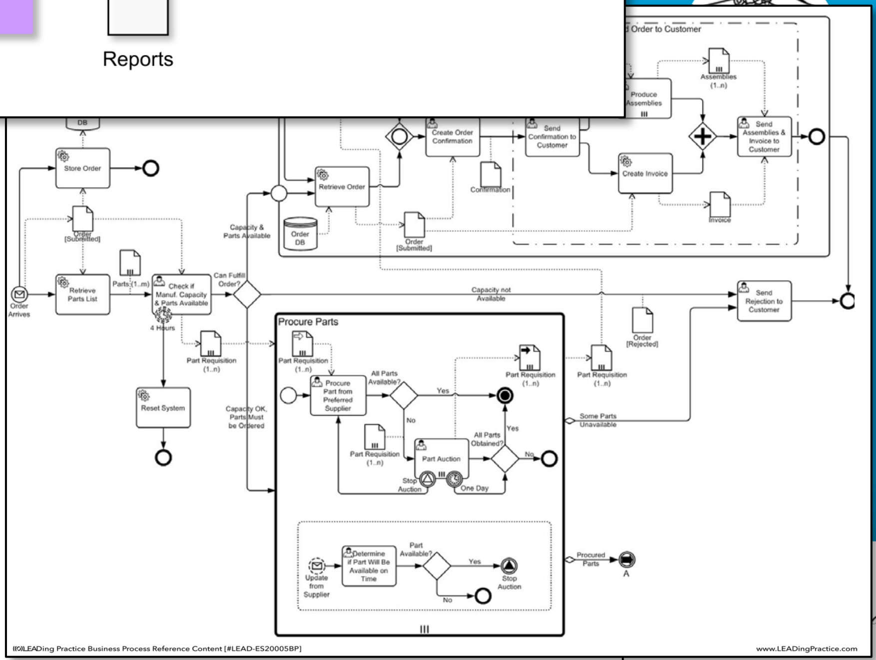




Come lavare le tue mani?
Per prevenire le infezioni bastano 60 secondi
1
2
3
4
6
7
8
10
11
12
www.salute.gov.it

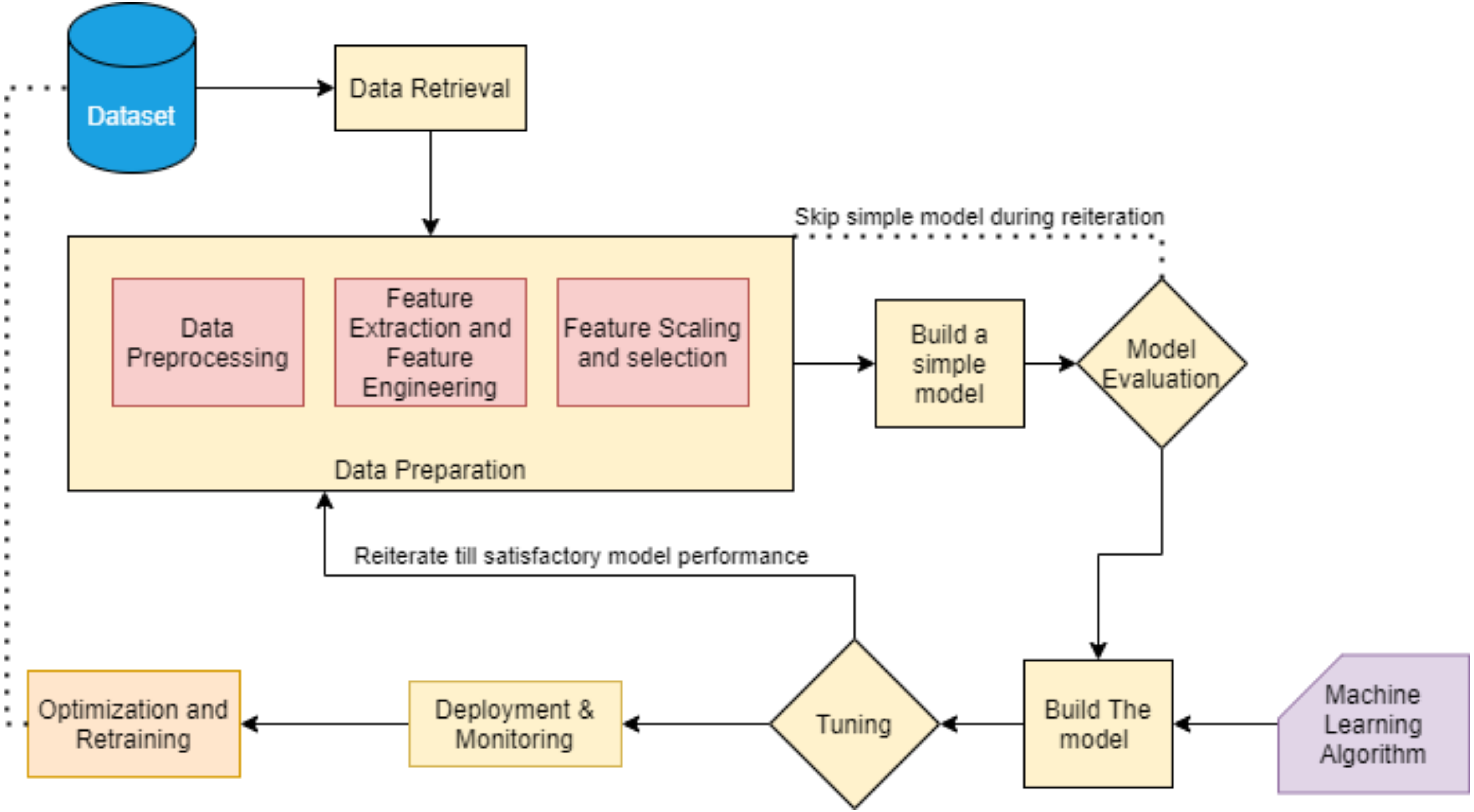


How to Make a Pizza



Procure Parts
www.LEADingPractice.com

Workflows


Dataset
Data Retrieval
Data Preprocessing
Feature Extraction and Feature Engineering
Feature Scaling and selection
Data Preparation
Skip simple model during reiteration
Build a simple model
Model Evaluation
Reiterate till satisfactory model performance
Optimization and Retraining
Deployment & Monitoring
Tuning
Build The model
Machine Learning Algorithm

Workflow Semantics

- **Host semantics** define the subprogram in each workflow step, usually expressed in a general-purpose programming language (e.g., Java, Python, C++) or as a shell script
- **Coordination semantics** define the interactions between steps. Coordination semantics can be either interleaved with host semantics or expressed through a declarative markup syntax, an imperative Domain Specific Language (DSL), or a graph-based modelling interface

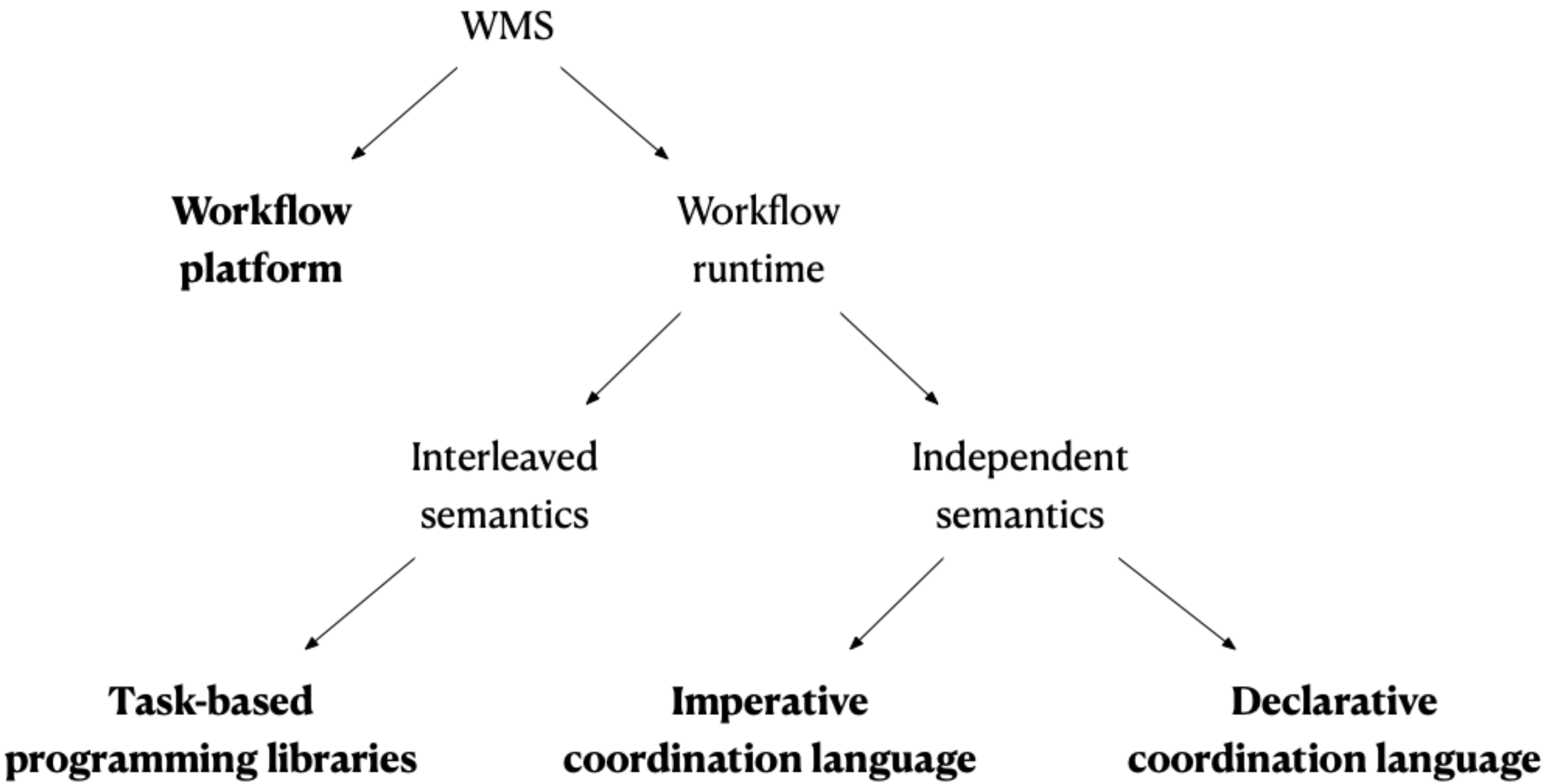
Workflow Lifecycle

- During the **design phase**, a domain expert describes the different functional components of an application and their dependencies as a workflow model
- During the **runtime phase**, a WMS deploys and manages the computational units required for the workflow execution

Workflow Management Systems (WMSs)

Tools in charge of **exposing coordination semantics** to the users and **orchestrating workflows**

Workflow Management Systems (WMSs)



Workflow Platforms

- Support **both design and runtime phases**, usually through advanced Graphical User Interfaces (GUI)
- Support **all aspects of workflow management**, e.g., provenance collection, catalogs, and fault-tolerance
- Usually tightly coupled with a **specific underlying architecture** (e.g., the Grid), without focusing on portability
- They are usually **complex to be installed and properly configured**, as they rely on low-level external libraries that must be independently managed (e.g., HTCondor or GAP interface)

Workflow Platforms

Pegasus

Kepler

Apache Taverna

Galaxy

ASKALON

Task-based Programming Libraries

- Users identify and annotate functions that can be executed as **asynchronous remote tasks**
- Synchronicity is typically implemented with the **futures** paradigm
- The workflow execution plan, typically a layered dataflow model, is **built just-in-time** by the runtime engine
- Privilege **performance over accessibility**, exposing a low-level programming model directly to the user
- Task-based programming libraries commonly offer support for a **limited set of host languages**, resulting in limited reusability and extensibility

Task-based Programming Libraries

Imperative Coordination Language

- Workflow are described using an **imperative DSL**, which is commonly a subset of a general-purpose programming language
- Apache Airflow and Snakemake are essentially **Python scripts** extended by **declarative code** that can be executed on distributed infrastructures
- Makeflow exposes a technology-neutral syntax similar to **Make**
- The Nextflow framework builds on the **Unix pipe** concept to expose an explicit dataflow model
- Toil and DagOnStar model workflows as **pure Python scripts**, through dedicated APIs

Imperative Coordination Language

Apache Airflow

nextflow

snakemake
A framework for reproducible data analysis

TOIL
Massively Parallel Workflows
(and fire-breathing dragon slugs)

Makeflow

Declarative Coordination Language

- The **Common Workflow Language** (CWL) is an open standard for describing workflow DAGs following a JSON or YAML syntax
- Other examples of workflow modelling open standards are the **Workflow Description Language** (WDL) and the **Serverless Workflow Specification**
- Declarative coordination languages are commonly **less expressive** than imperative ones, but it's easier for WMSs to apply rewriting techniques for optimization, improving **performance portability**
- Also, declarative languages are usually **product-agnostic**, improving portability and reusability. An exception is **DAX**, the Pegasus' XML-based low level representation of DAGs

Declarative Coordination Language

Pegasus

COMMON WORKFLOW LANGUAGE

{wdl}

Serverless Workflow Specification

Common Workflow Language (CWL)

Common Workflow Language (CWL)

- Open standard for describing **analysis workflows and tools**
- Defined with a **schema**, **specification** and **test suite**
- **Portable and scalable** across a variety of software and deployment environments
- Designed to meet the needs of data-intensive science to improve the **FAIRness** of their workflows

open stand
the modern paradigm for standards

software freedom conservancy

COMMON WORKFLOW LANGUAGE

Common Workflow Language (CWL)

- **Human readable** (YAML or JSON)
- CWL file contains a **CommandLineTool** or **Workflow** description

```yaml
cwlVersion: v1.2
class: CommandLineTool

baseCommand: echo

inputs:
  message_text:
    type: string
    inputBinding:
      position: 1

outputs: []
```

```yaml
cwlVersion: v1.2
class: Workflow

inputs:
  rna_reads_fruitfly: File

steps:
  quality_control:
    run: bio-cwl-tools/fastqc/fastqc_2.cwl
    in:
      reads_file: rna_reads_fruitfly
    out: [html_file]

outputs:
  quality_report:
    type: File
    outputSource: quality_control/html_file
```

Common Workflow Language (CWL)

- **Human readable** (YAML or JSON)
- CWL file contains a **CommandLineTool** or **Workflow** description
- Inputs/outputs are explicitly stated

```
cwlVersion: v1.2
class: CommandLineTool

baseCommand: echo

inputs:
  message_text:
    type: string
    inputBinding:
      position: 1

outputs: []
```

```
message_text: Hello world!
```

```
inputs:
  rna_reads_fruitfly: File
```

```
cwlVersion: v1.2
class: Workflow

inputs:
  rna_reads_fruitfly: File

steps:
  quality_control:
    run: bio-cwl-tools/fastqc/fastqc_2.cwl
    in:
      reads_file: rna_reads_fruitfly
    out: [html_file]

outputs:
  quality_report:
    type: File
    outputSource: quality_control/html_file
```

Common Workflow Language (CWL)

- CWL Types: **strings**, **numbers**, **files**, or **records** that combine these; or **arrays** of any of these types
- **Union** and **optional types** too

```
cwlVersion: v1.2
class: CommandLineTool

baseCommand: echo

inputs:
  message_text:
    type: string
    inputBinding:
      position: 1

outputs: []
```

```
inputs:
  rna_reads_fruitfly: File
```

```
message_text: Hello world!
```

Implicit `string` type

```
cwlVersion: v1.2
class: Workflow

inputs:
  rna_reads_fruitfly: File

steps:
  quality_control:
    run: bio-cwl-tools/fastqc/fastqc_2.cwl
    in:
      reads_file: rna_reads_fruitfly
    out: [html_file]

outputs:
  quality_report:
    type: File
    outputSource: quality_control/html_file
```

Common Workflow Language (CWL)

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.2
class: Workflow

requirements:
  ScatterFeatureRequirement: {}

inputs:
  message_array: string[]

steps:
  echo:
    run: hello_world.cwl
    scatter: message
    in:
      message: message_array
    out: []

outputs: []
```

CWL Supports *Scatter/Gather* parallel patterns at the step level since v1.0.

If **scatter** declares more than one input parameter, **scatterMethod** describes how to decompose the input into a discrete set of jobs (*dotproduct*, *nested crossproduct*, or *flat crossproduct*).

Common Workflow Language (CWL)

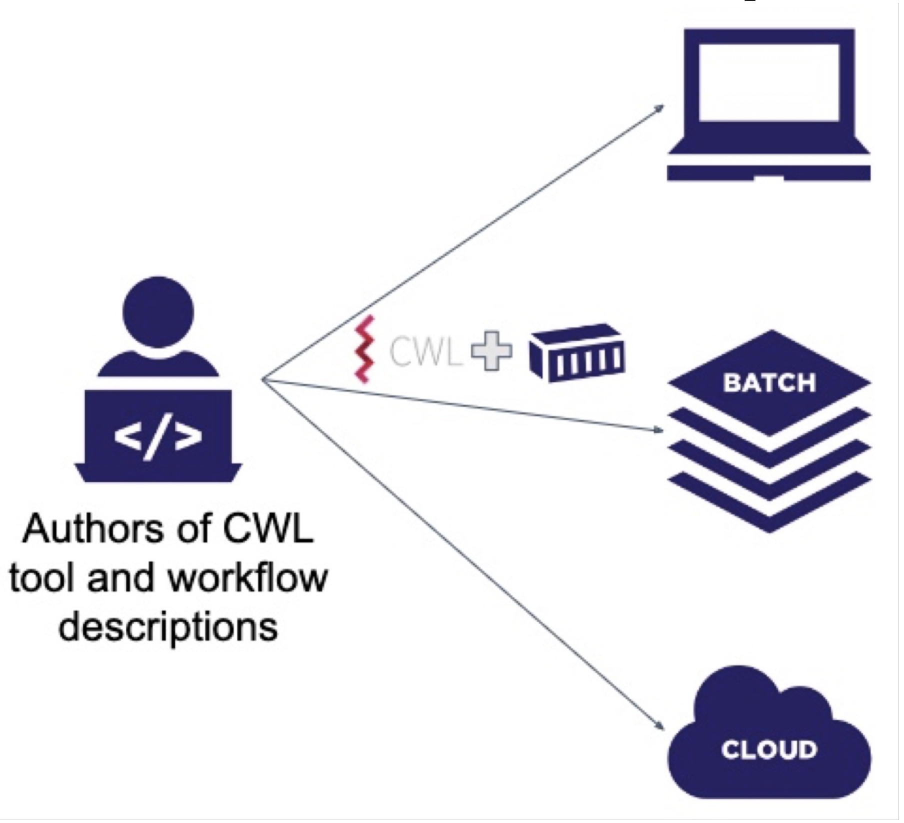
```
#!/usr/bin/env cwl-runner
cwlVersion: v1.2
class: CommandLineTool
baseCommand: node
hints:
  DockerRequirement:
    dockerPull: node:slim
inputs:
  src:
    type: File
    inputBinding:
      position: 1
outputs:
  example_out:
    type: stdout
stdout: output.txt
```

CWL Supports *software containers* at the CommandLineTool level since v1.0.

The **DockerRequirement** directive allows users to specify the Docker image that should execute the command

Multiple CWL implementations can support different container runtimes

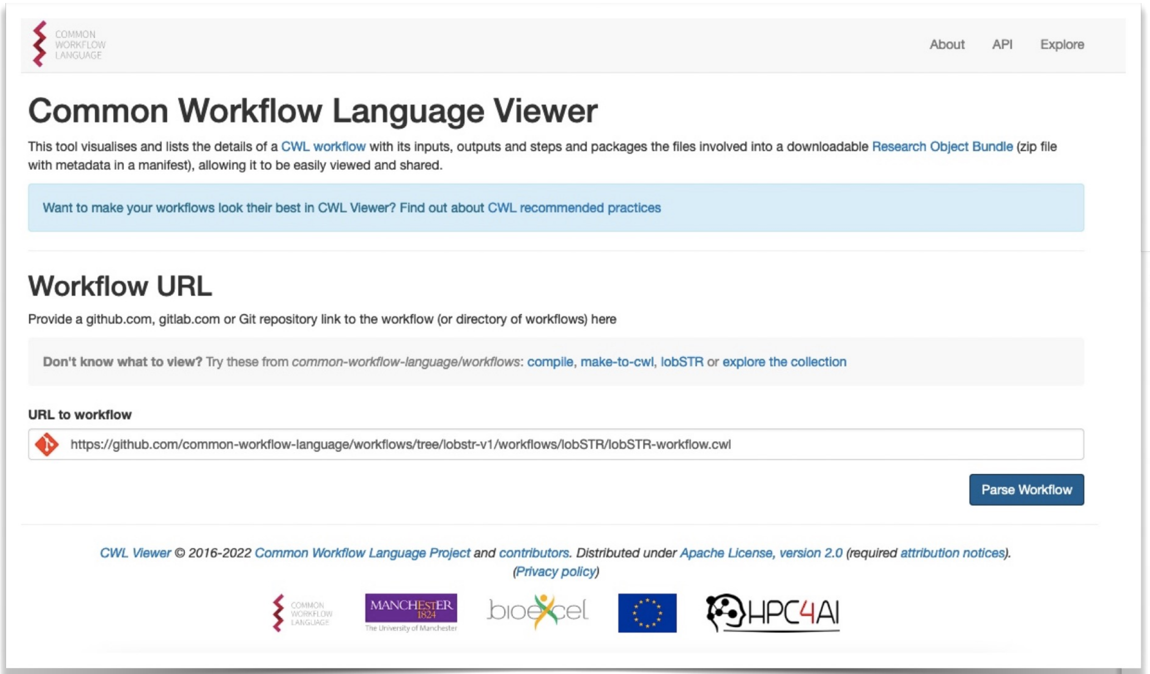
CWL Portability

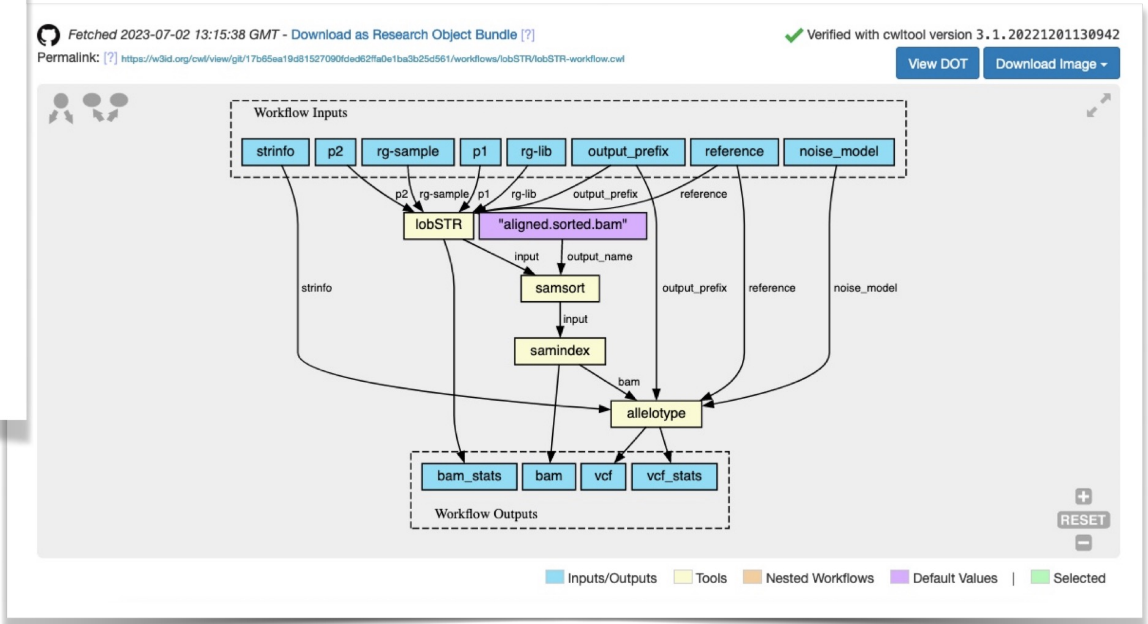


HTCondor
IBM Spectrum LSF
grid engine
slurm workload manager
PBS Works
Apache MESOS
kubernetes
Google Cloud
openstack
amazon webservices EC2
Azure

CONDA
ENVIRONMENT MODULES

CWL Ecosystem – CWL Viewer





https://view.commonwl.org

CWL Ecosystem – WorkflowHub

WorkflowHub

FAIR Workflow Registry

Launched Sept 2020
EOSC service provided by ELIXIR, EOSC-Life, The University of Manchester
Open Development

283 workflows

11 system types

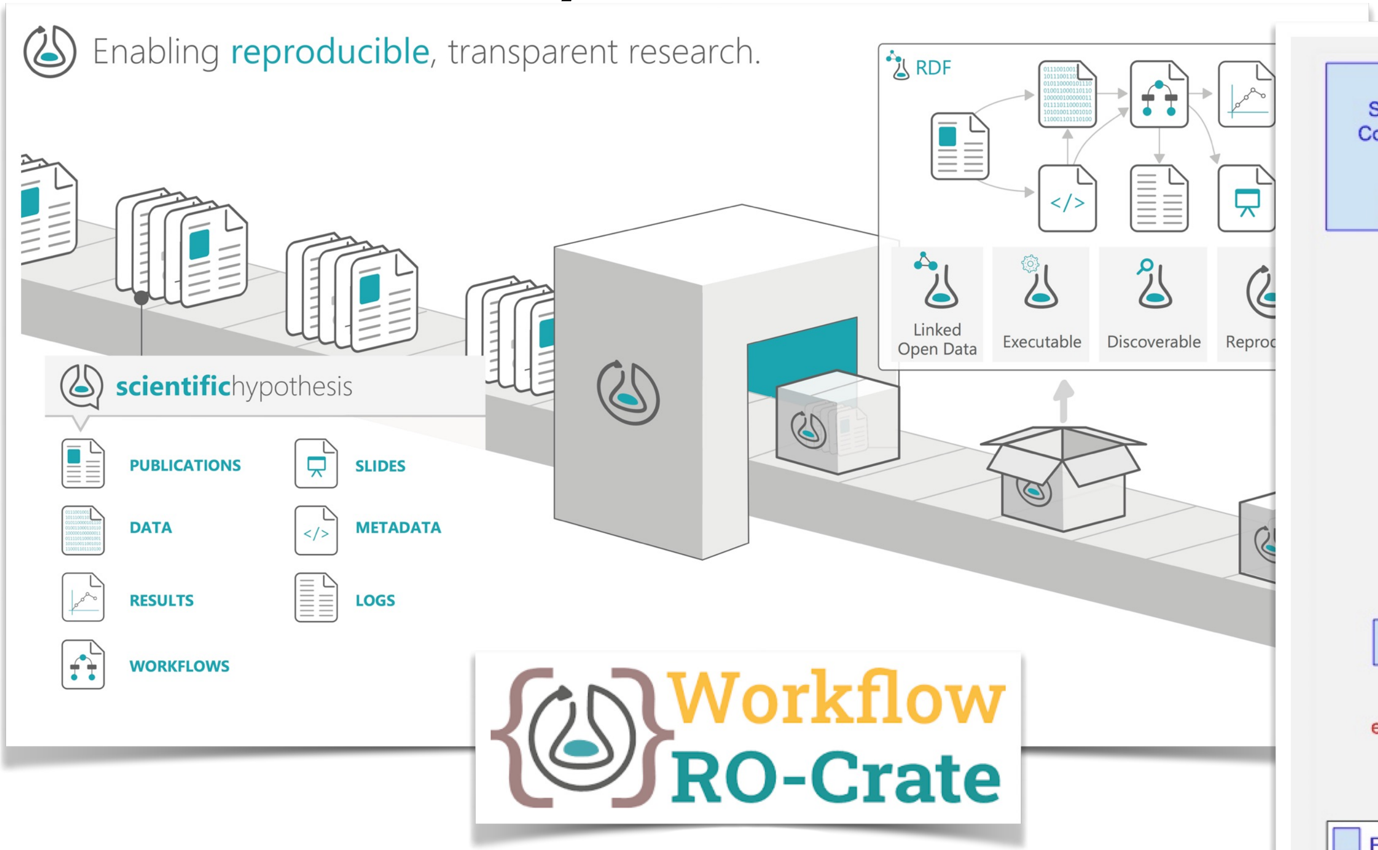
110 teams

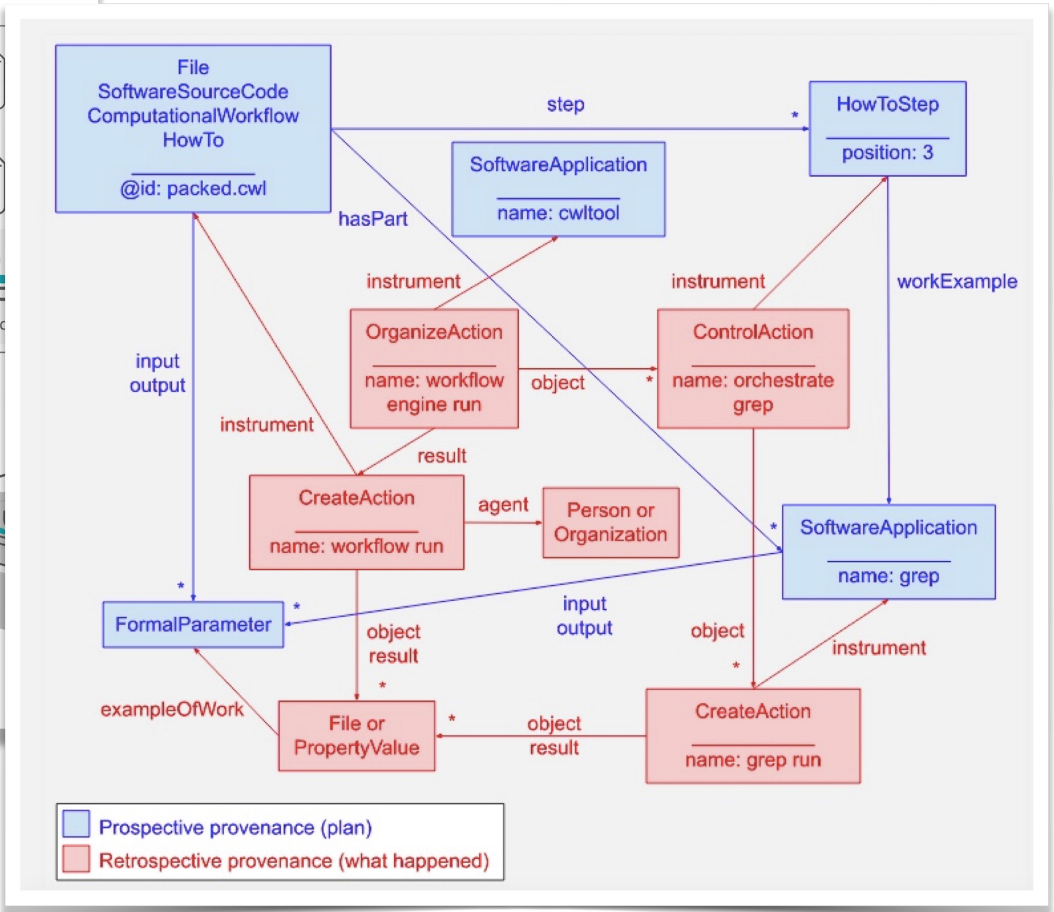
103 organisations

360 people

https://workflowhub.eu

CWL Ecosystem – Workflow Run RO-Crate

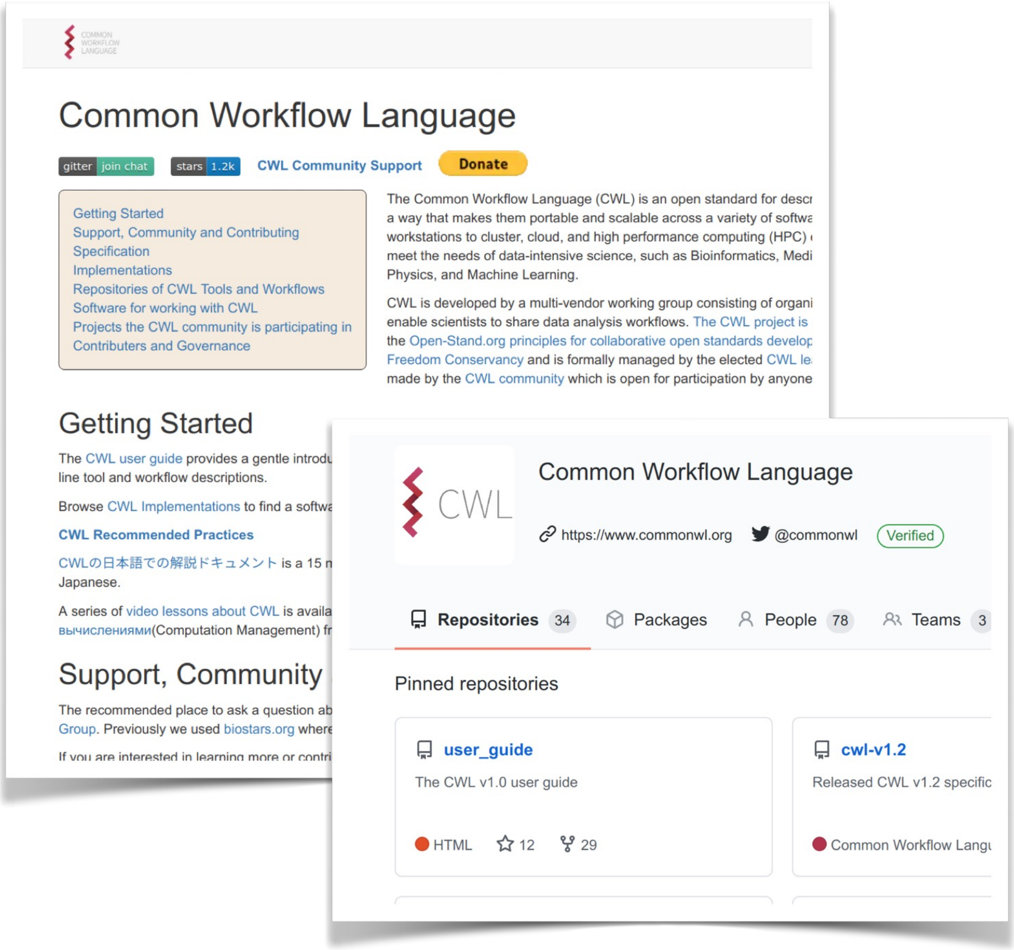




https://www.researchobject.org/workflow-run-crate/

CWL Community

- Website: https://www.commonwl.org/
- User guide: https://www.commonwl.org/user_guide/
- Forum: https://cwl.discourse.group/
- Chat: https://matrix.to/#/#cwl:matrix.org
- GitHub: https://github.com/common-workflow-language/
- Weekly video chat: https://groups.google.com/forum/#!forum/common-workflow-language-videochat-invites



CWL Hands-on Session

COMMON WORKFLOW LANGUAGE

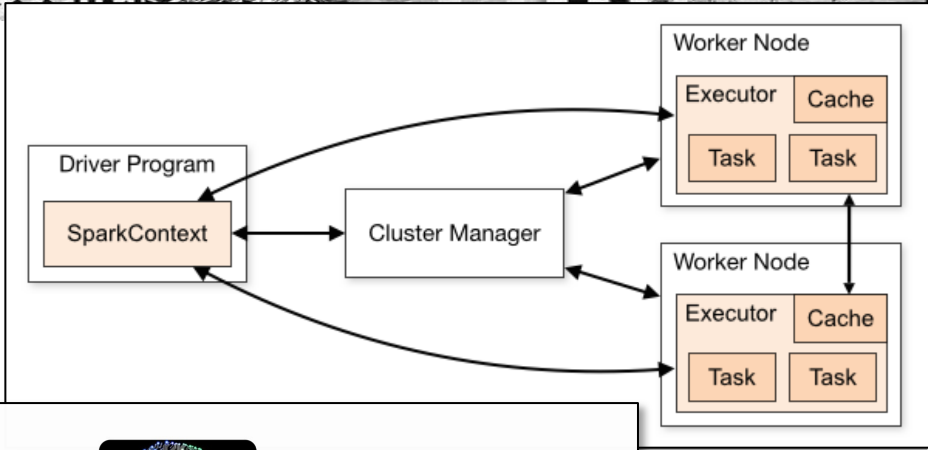
https://github.com/Sera91/SMR3941-ICTP/blob/main/Day2/Tutorial-Workflow/README.md

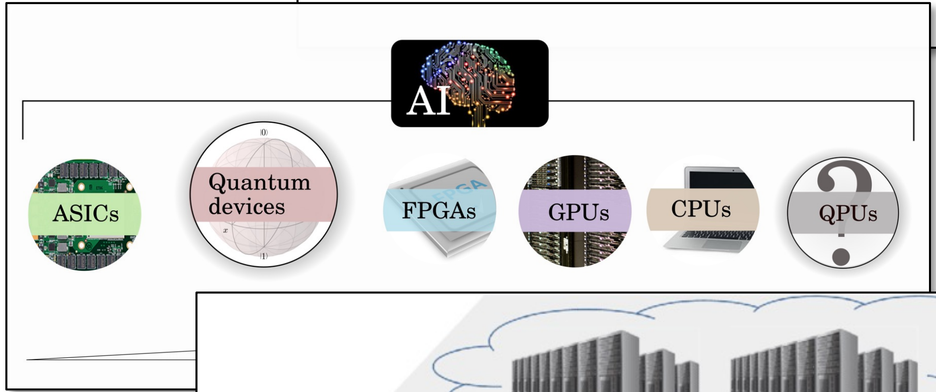
Hybrid workflows

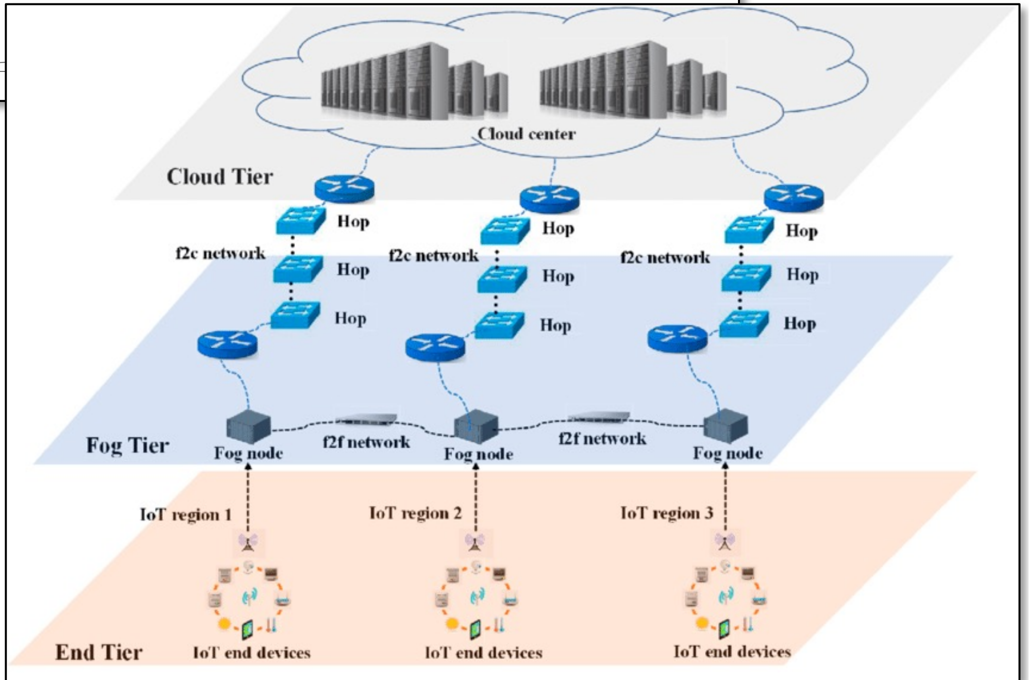
Scientific Workflows

CHALLENGES:

- Each step of a distributed application can require **multiple intercommunicating agents** (e.g., a Spark cluster or a micro-services architecture);
- Large-scale architectures can be **heterogeneous** (e.g., Cloud+HPC environments and Classical+Quantum computing);
- Large-scale architectures can be **modular**, and modules can be **independent** of each other (e.g., modular HPC and infrastructure federations)







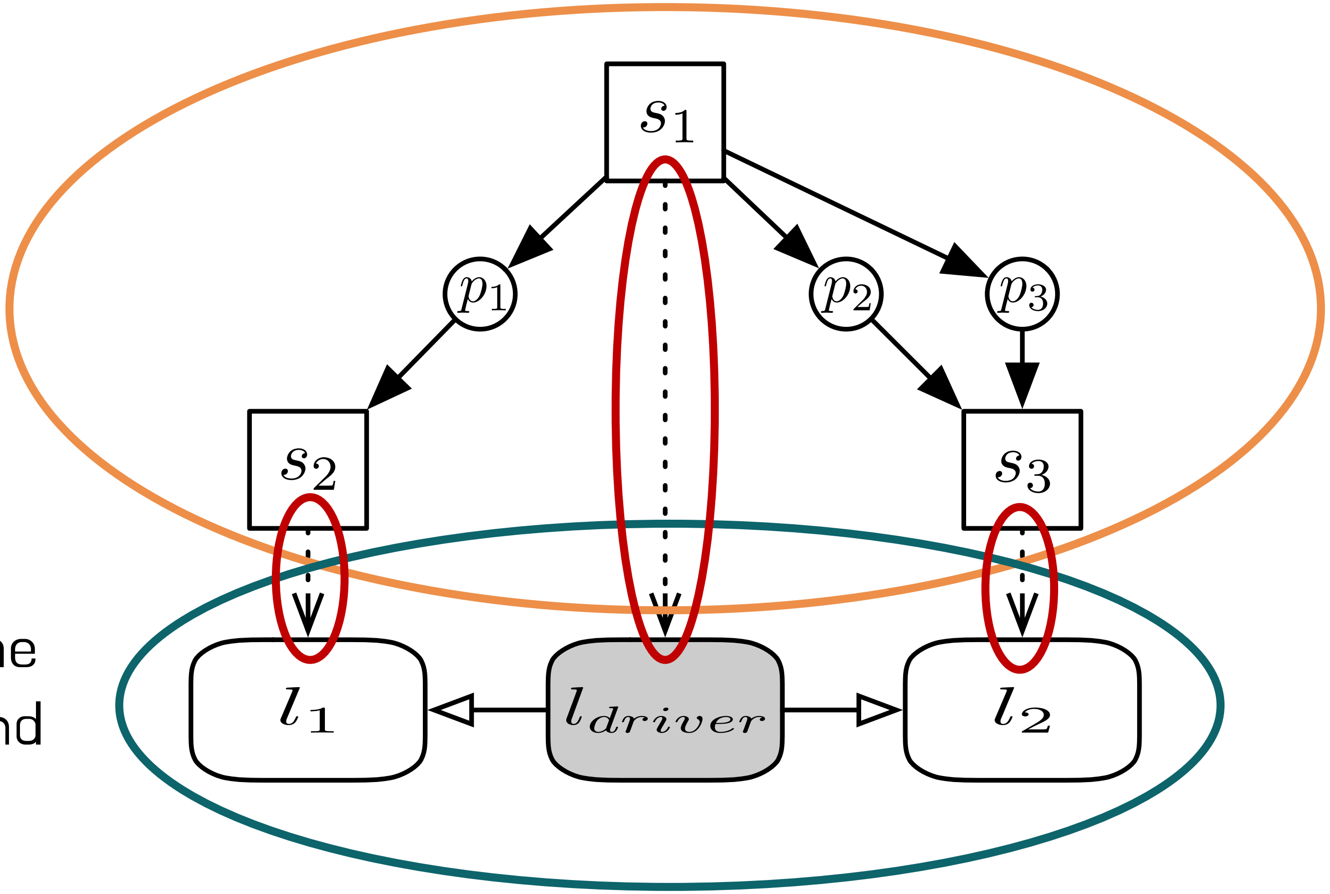
Hybrid Workflows

Workflow model

A directed bipartite graph encoding executable **steps**, data **ports** and **dependencies** between them

Topology of deployment locations

A directed graph where the nodes are the **locations** in charge of executing steps and the links are directed **communication channels** between locations



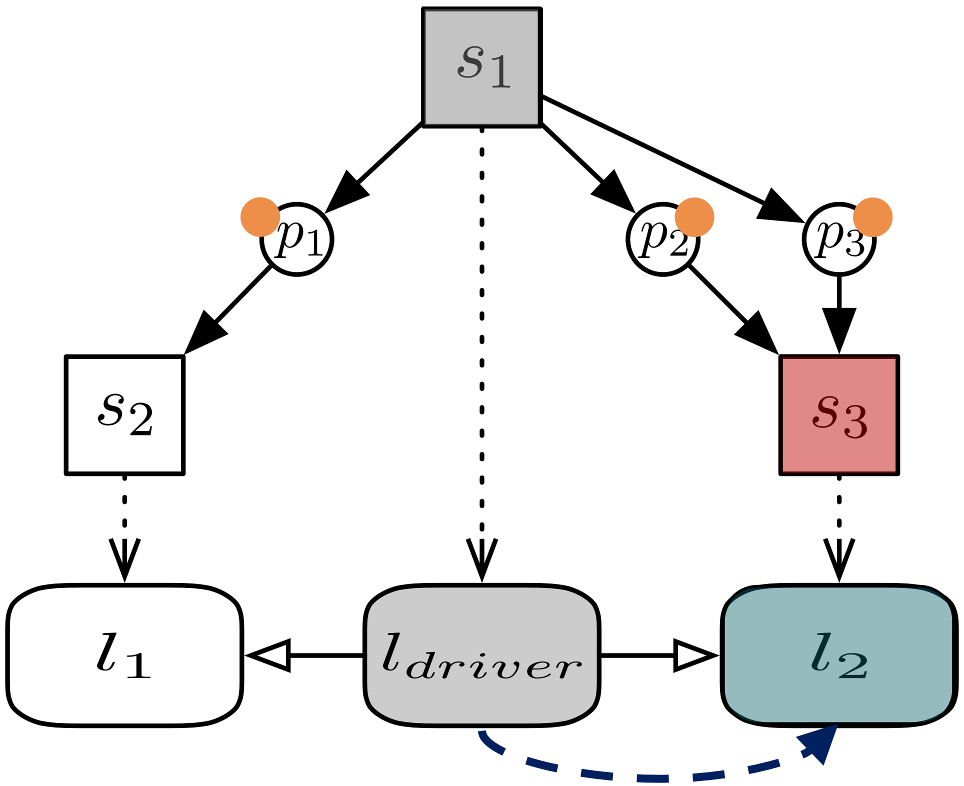
Mapping relations

Many-to-many relations stating which locations are in charge of executing each workflow step

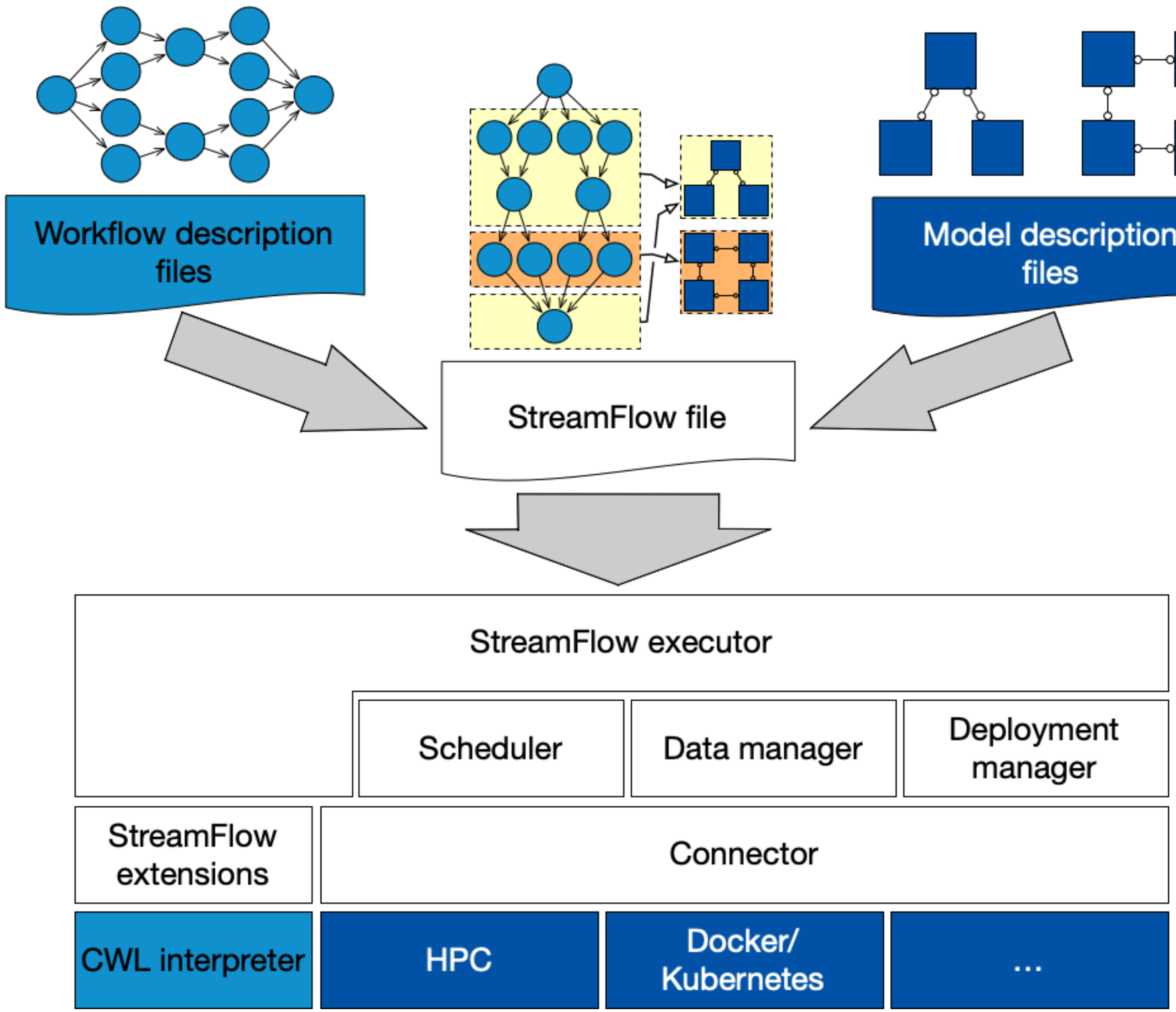
Model Interpretation

A step s becomes **fireable** (ready for execution) when:

- Each input port $In(s)$ contains the right number of **tokens**
- Its related location is **deployed**
- All its input data have been **transferred** on that location



The StreamFlow WMS



https://streamflow.di.unito.it

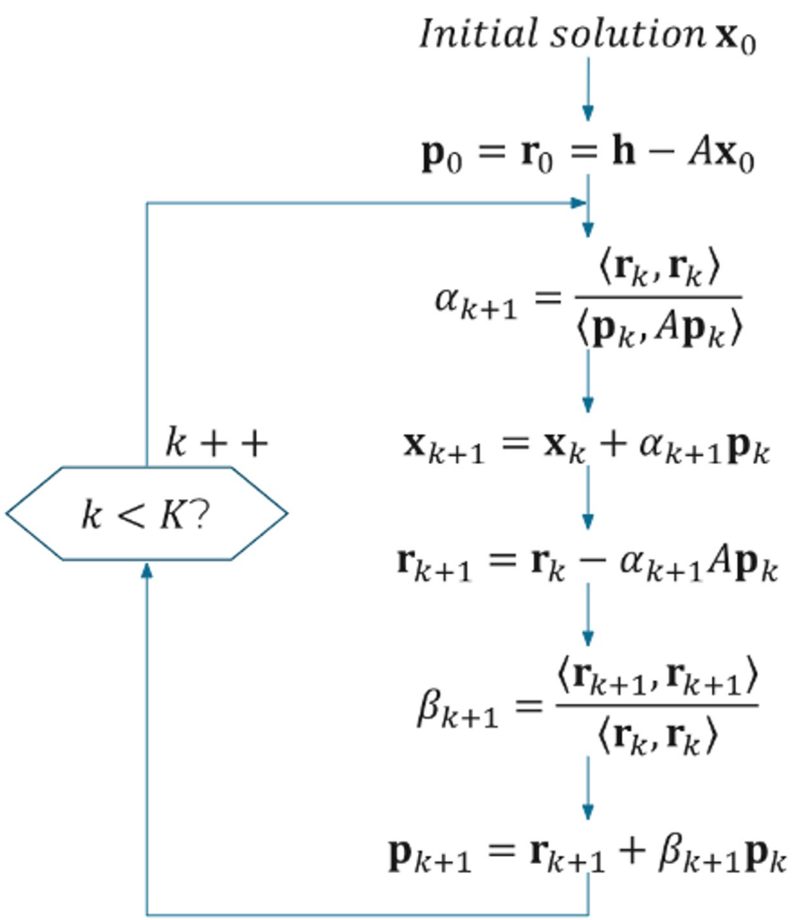
I. Colonnelli, B. Cantalupo, I. Merelli and M. Aldinucci, "StreamFlow: cross-breeding cloud with HPC," in *IEEE Transactions on Emerging Topics in Computing*, vol. 9, iss. 4, p. 1723-1737, 2021. doi: 10.1109/TETC.2020.3019202.

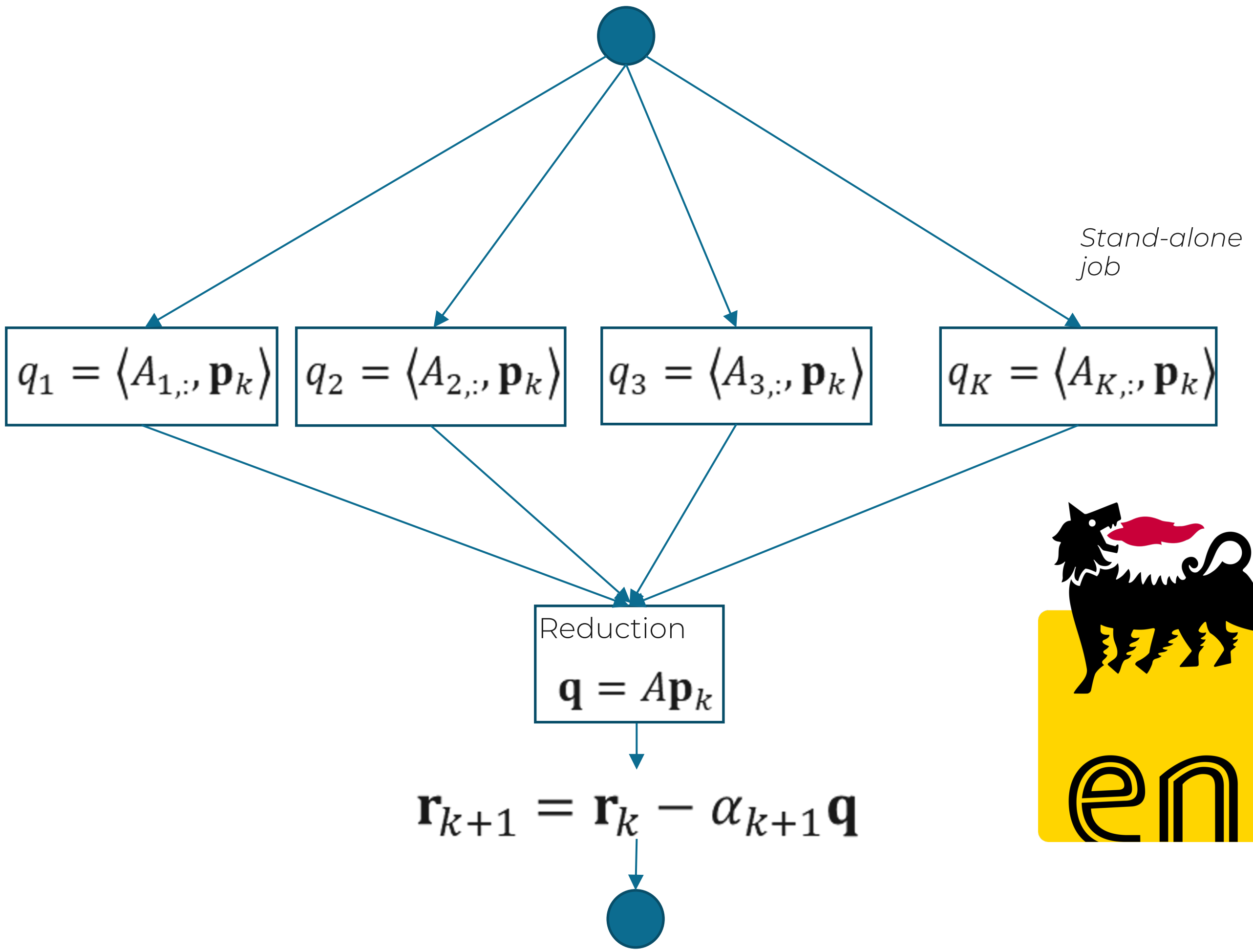
The StreamFlow WMS

StreamFlow is listed as a **production-ready implementation** of CWL. It has also been used as a software laboratory to experiment new CWL extensions in the **CWL4HPC Working Group** (e.g., the Loop extension for iterative workflows)

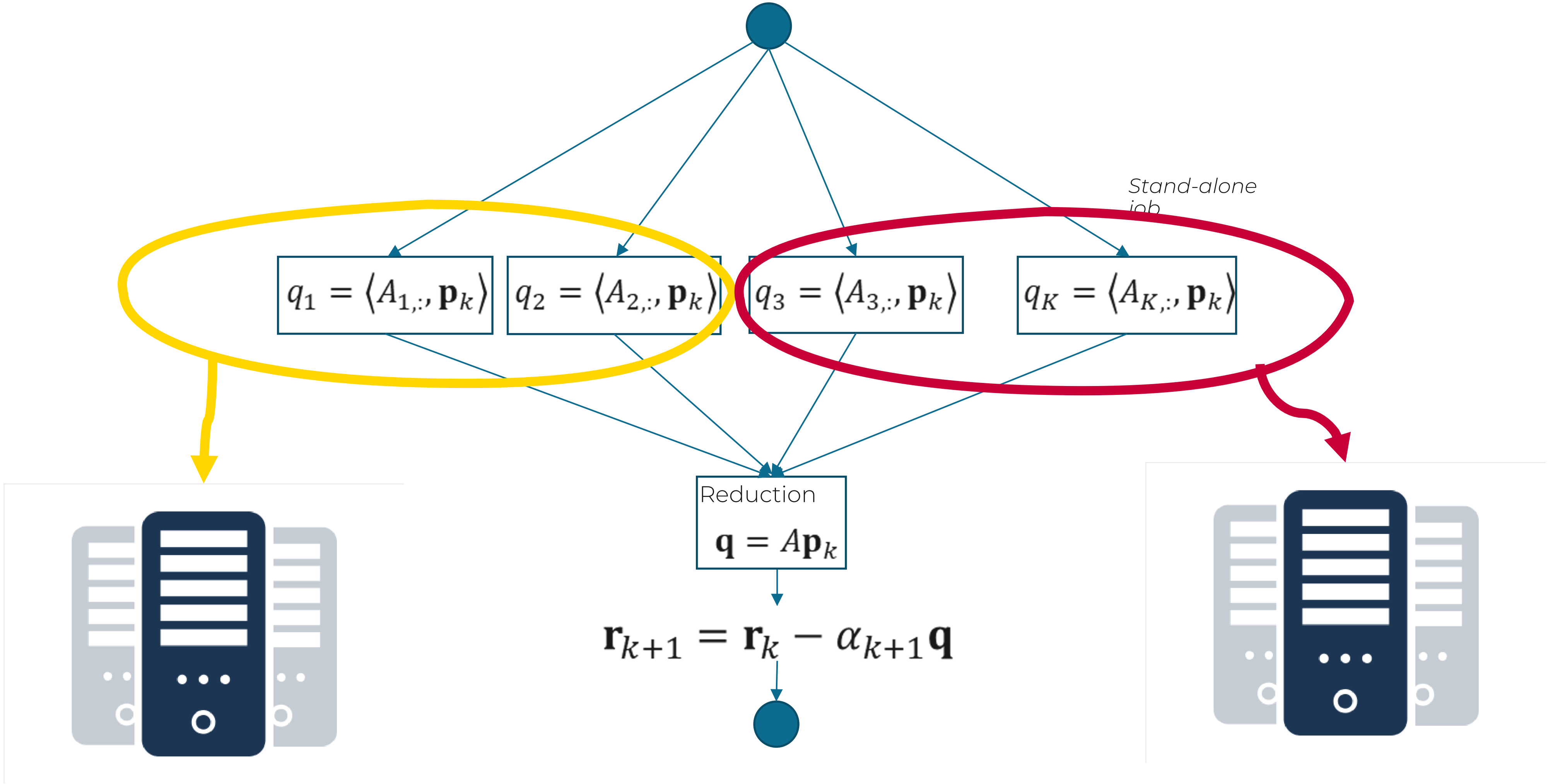
Software	Description	Self-Reported Compliance	Platform support
cwltool	Reference implementation of CWL	CWL v1.0 - v1.2	Linux, OS X, Windows, local execution only
Arvados	Distributed computing platform for data analysis on massive data sets. Using CWL on Arvados	CWL v1.0 - v1.2 required 100%	AWS, GCP, Azure, Slurm, LSF
Toil	Toil is a workflow engine entirely written in Python.	CWL v1.0 - v1.2	AWS, Azure, GCP, Grid Engine, HTCondor, LSF, Mesos, OpenStack, Slurm, PBS/Torque
CWL-Airflow	Package to run CWL workflows in Apache-Airflow (supported by BioWardrobe Team, CCHMC)	CWL v1.0 - v1.1	Linux, OS X
StreamFlow	Workflow Management System for hybrid HPC-Cloud infrastructures	CWL v1.0 - v1.2 required 100% (and nearly all optional features)	Kubernetes, HPC with Singularity (PBS, Slurm), Occam, multi-node SSH, local-only (Docker, Singularity)

Case study: Distributed Conjugate Gradient





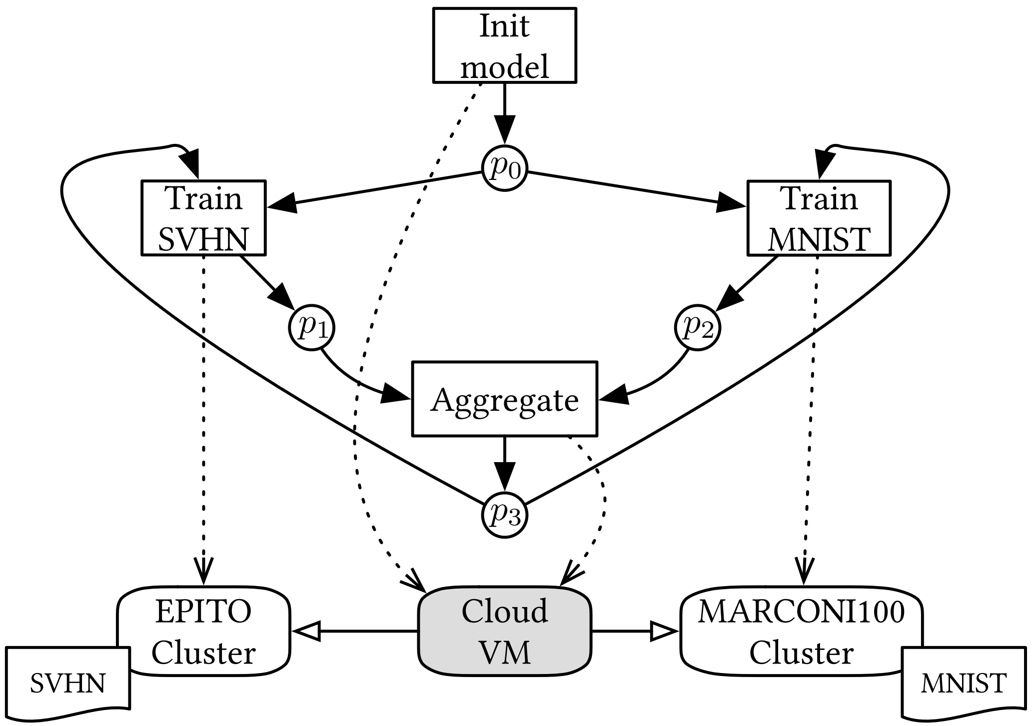
Case study: Distributed Conjugate Gradient



HPC Facility A

HPC Facility B

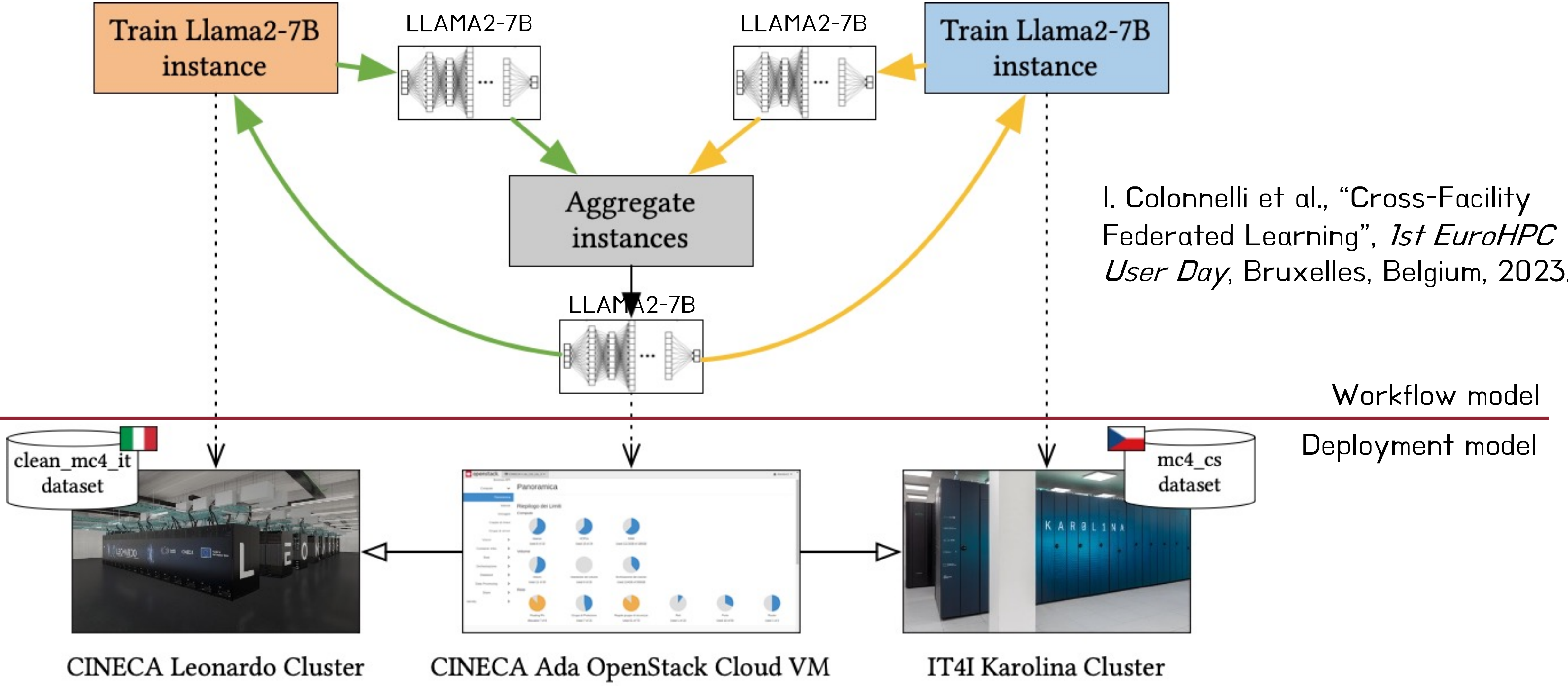
Case study: Cross-Facility Federated Learning



I. Colonnelli, B. Casella, G. Mittone, Y. Arfat, B. Cantalupo, R. Esposito, A. R. Martinelli, D. Medić and M. Aldinucci, "Federated Learning meets HPC and cloud," in *Astrophysics and Space Science Proceedings*, vol 60, 2023, p. 193-199. doi: 10.1007/978-3-031-34167-0_39

		StreamFlow			OpenFL		
		MNIST acc.	SVHN acc.	Time	MNIST acc.	SVHN acc.	Time
Cloud	100 rounds, 1 epoch/round	99.36%	92.74%	2h40m	97.91%	93.15%	3h06m
	50 rounds, 2 epochs/round	99.37%	92.74%	2h20m	98.88%	94.21%	2h09m
Hybrid	100 rounds, 1 epoch/round	99.29%	93.06%	2h57m	–	–	–
	50 rounds, 2 epochs/round	99.34%	92.85%	1h45m	–	–	–

Case study: Cross-Facility Federated Learning



I. Colonnelli et al., "Cross-Facility Federated Learning", *1st EuroHPC User Day*, Bruxelles, Belgium, 2023.

Literate workflows

Scientific Workflows Adoption

CHALLENGES:

- Learning a new **coordination language** is an extra efforts that often domain experts don't do;
- Dealing with **language syntax and semantics** (although simple and declarative) can be difficult for non IT people;
- People are often more comfortable in **extending their knowledge of a product they already use**, instead of learning something new from scratch.

Computational Notebooks

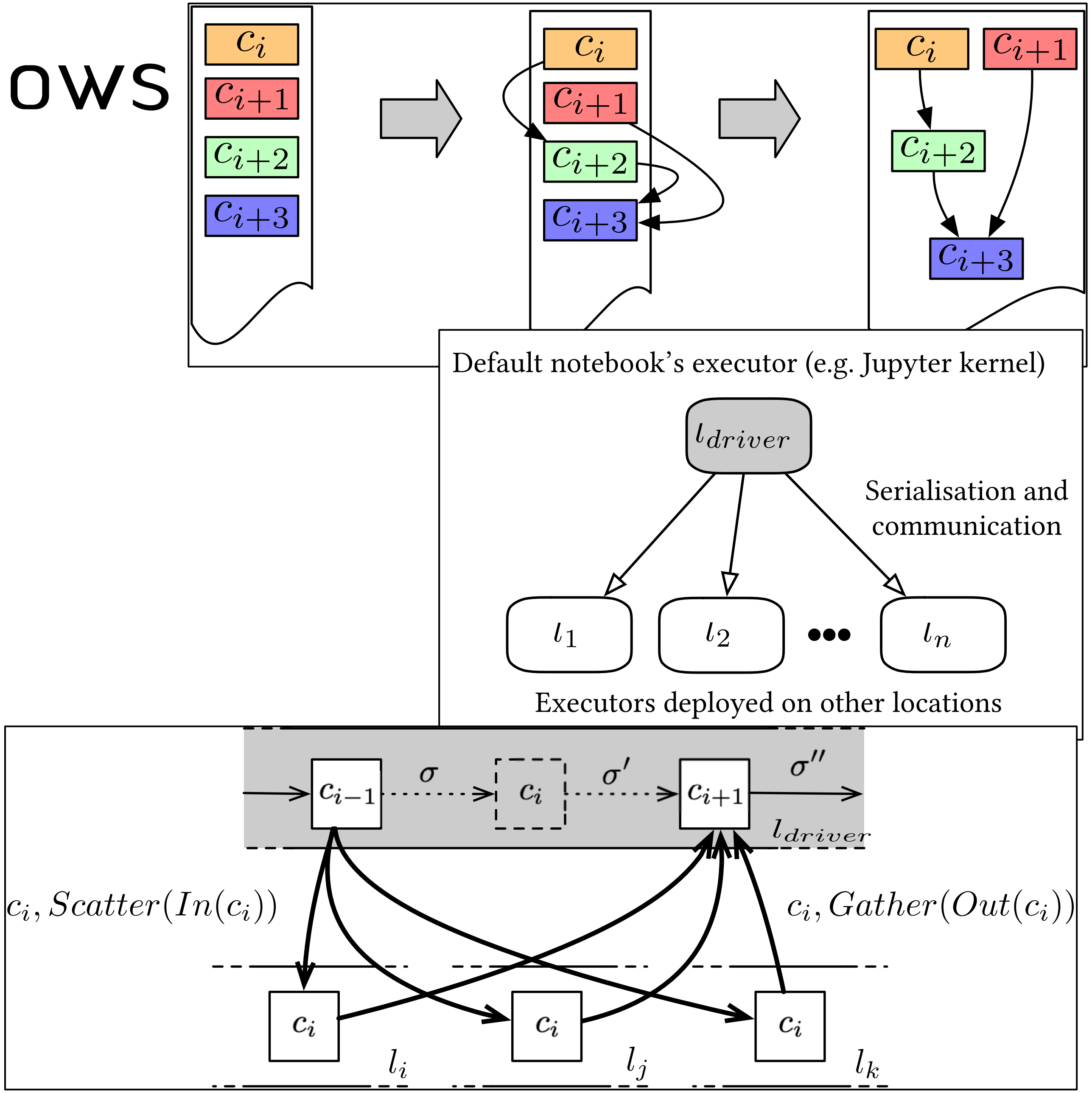
CHALLENGES:

- Notebooks' **purely sequential** execution flow makes it impossible to exploit the inherent concurrency of workflow graphs;
- The lack of a rigorous workflow model prevents to satisfy non-functional requirements like portability, **reproducibility**, **provenance collection**;
- Using Notebooks as a high-level interface to HPC facilities poses crucial **security challenges** due to the lack of support for hybrid topologies.

Hybrid Literate Workflows

REQUIREMENTS:

- Infer inter-cell **true data dependencies** to construct a DAG
- Derive **sequentially equivalent** parallel semantics to extract concurrency from the cells execution;
- Extend the **Notebook metadata format** to describe:
  - Topologies of deployment locations
  - Mapping relations
  - **Explicit intra-cell data-parallel constructs** (e.g. scatter/gather)



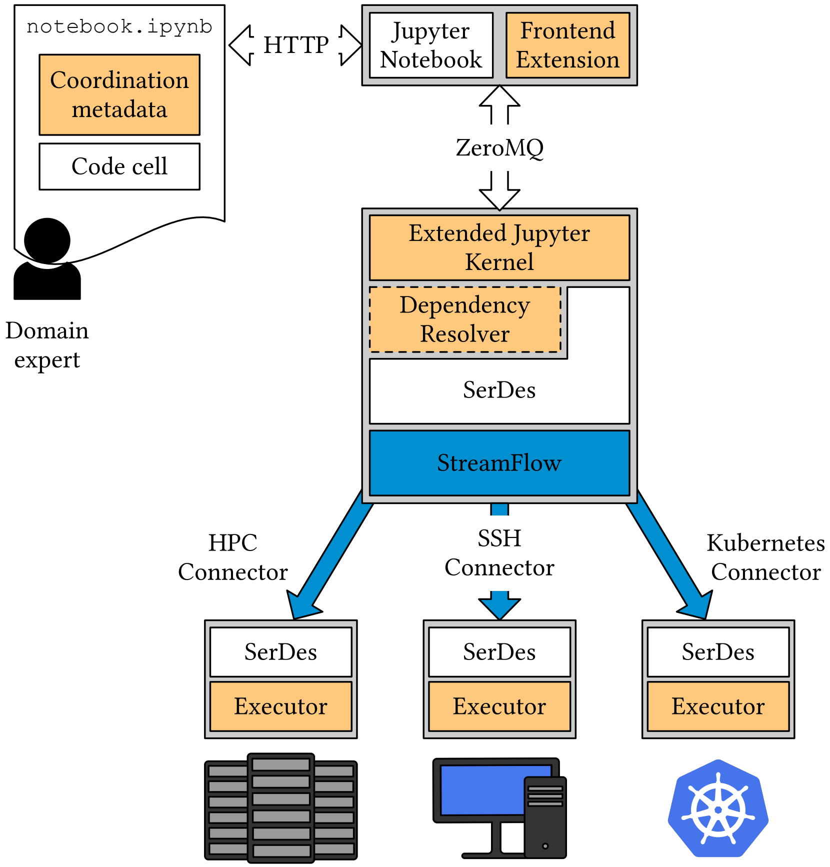
Jupyter Workflow

The **Jupyter Workflow** kernel extends the IPython software stack to support hybrid literate workflows in the Jupyter stack.

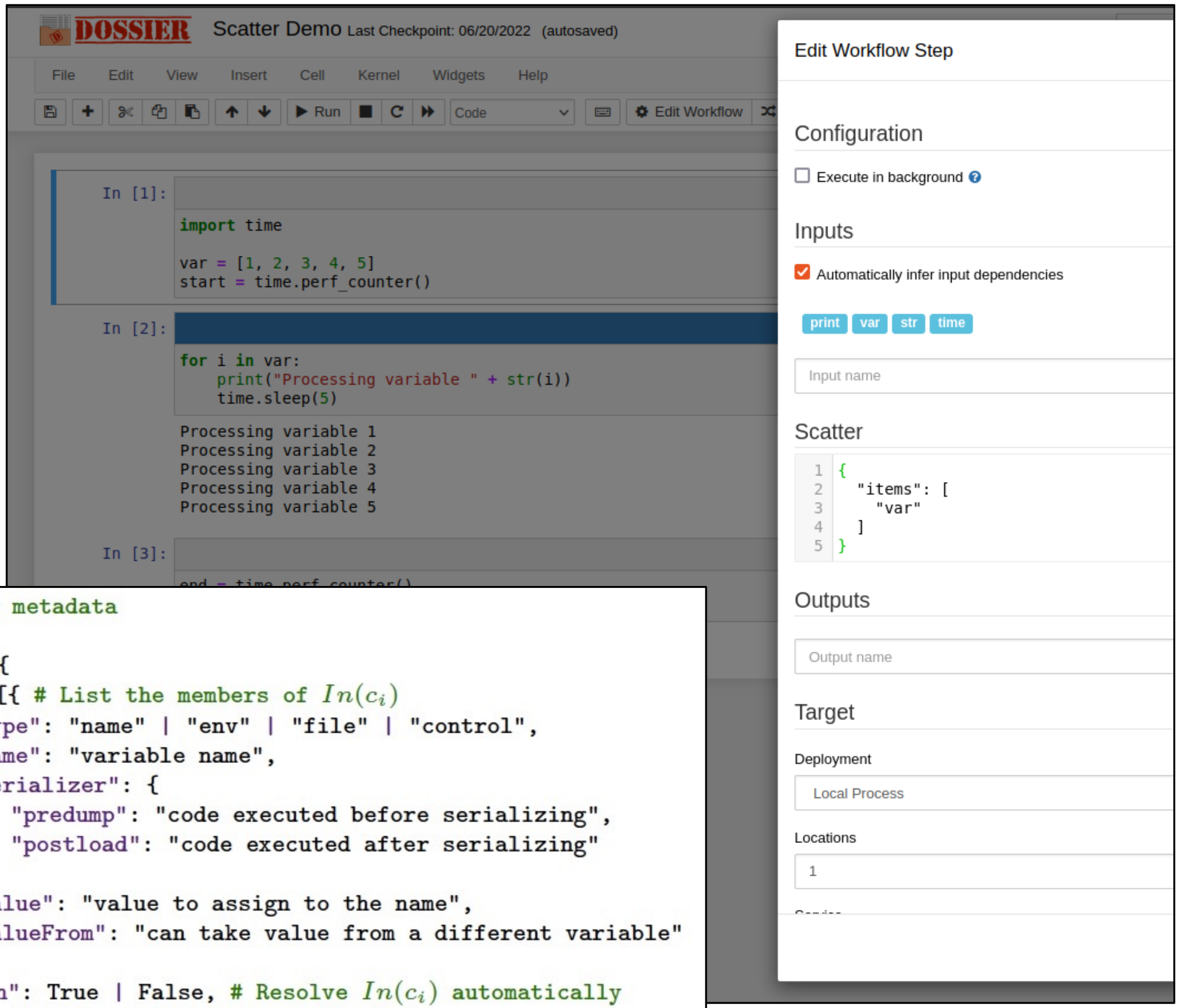
It consists of three main components:

- A **coordination metadata format** to model global cells configurations and location topologies;
- A **dependency resolver component** to help users identify the input dependencies of each cell;
- A **Jupyter stack extension** to handle coordination metadata, execute cells remotely and manage data transfers (through StreamFlow).

https://jupyter-workflow.di.unito.it

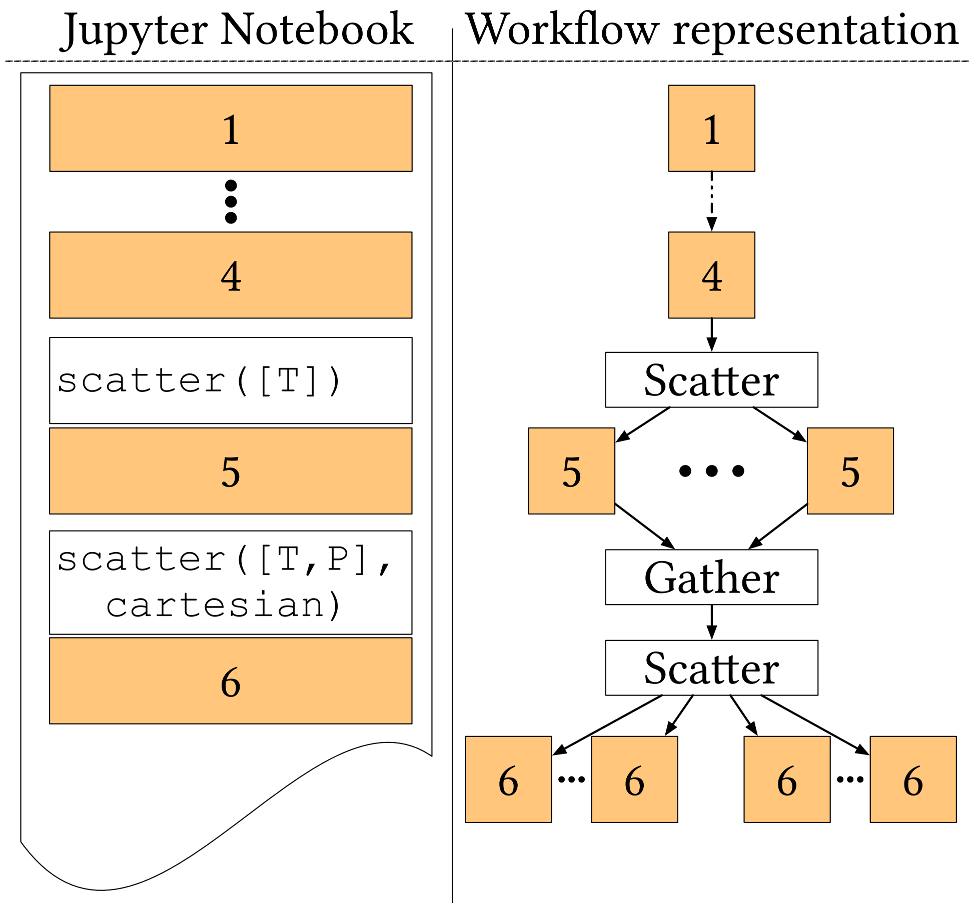


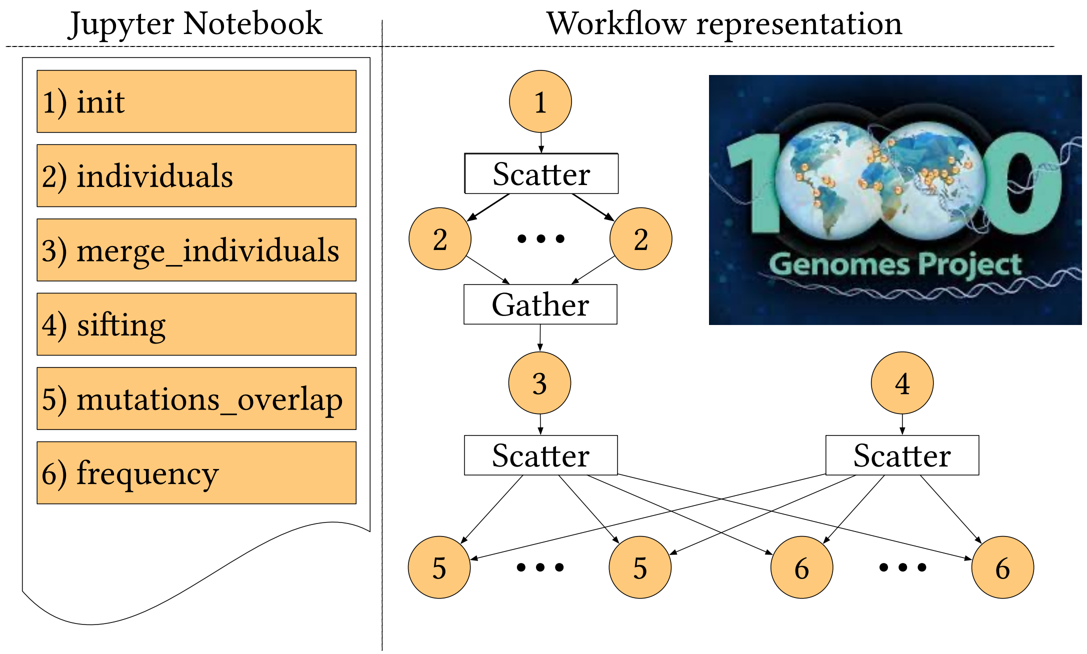
https://jupyter-workflow.di.unito.it




```
# Workflow metadata
{
  "step": {
    "in": [{ # List the members of In(c_i)
        "type": "name" | "env" | "file" | "control",
        "name": "variable name",
        "serializer": {
            "predump": "code executed before serializing",
            "postload": "code executed after serializing"
        },
        "value": "value to assign to the name",
        "valueFrom": "can take value from a different variable"
    }],
    "autoin": True | False, # Resolve In(c_i) automatically
    "out": [ # List the members of Out(c_i)
        ...
    ],
    "scatter": {
        "items": ["variable name" | "scatter subscheme" ],
        "method": "dotproduct" | "cartesian" | ...
    }
  },
```

Jupyter Workflow





I. Colonnelli, M. Aldinucci, B. Cantalupo, L. Padovani, S. Rabellino, C. Spampinato, R. Morelli, R. Di Carlo, N. Magini, and C. Cavazzoni, "Distributed workflows with Jupyter," *Future generation computer systems*, vol. 128, pp. 282-298, 2022.

https://jupyter-workflow.di.unito.it

QUANTUM ESPRESSO