SWIFT-HEP
SoftWare InFrasTructure for High Energy Physics

5 December 2023

Material from the SWIFT-HEP workshop
Work done by Luke Kreczko, Sam Eriksen (Bristol), Tim Noble, Alastair Dewhurst (RAL),
Daniela Bauer, Simon Fayer, Janusz Martyniak (Imperial)

Acronym disambiguation

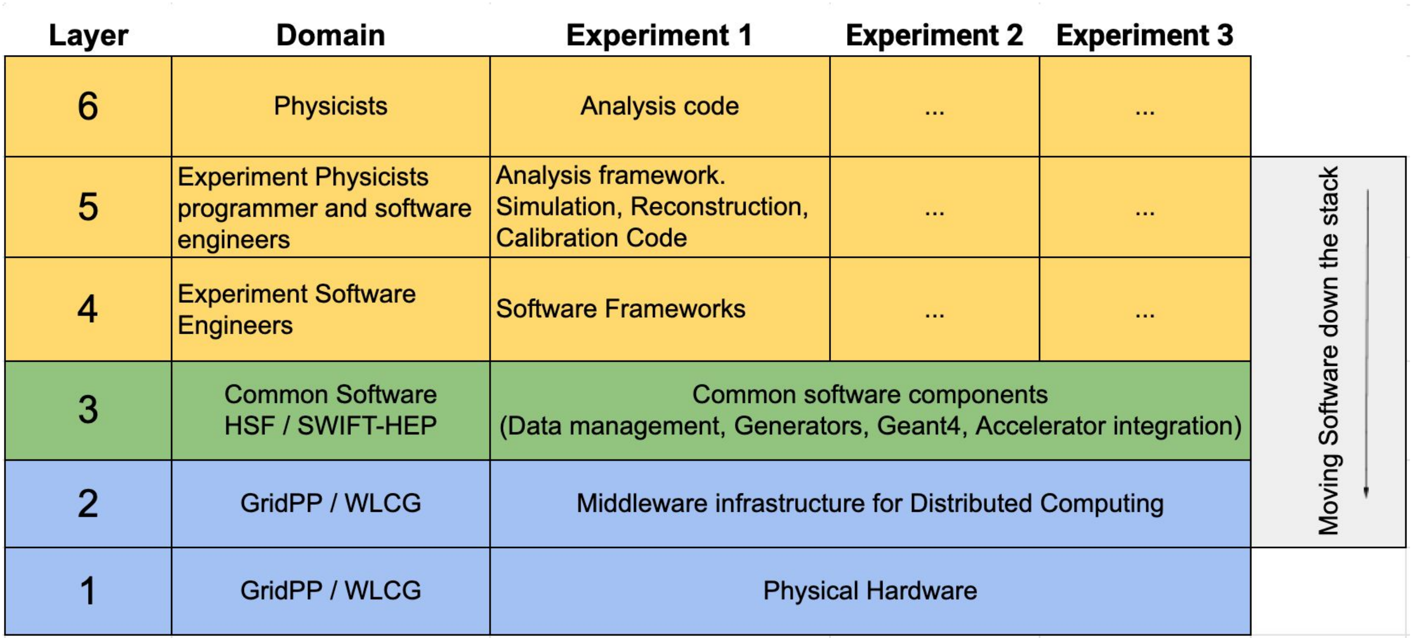
DIRAC = Workflow management system (which is not DiRAC)

IRIS-HEP = Institute for Research and Innovation in Software for High Energy Physics (not IRIS)

Multi-experiment software development

LHCC Review;

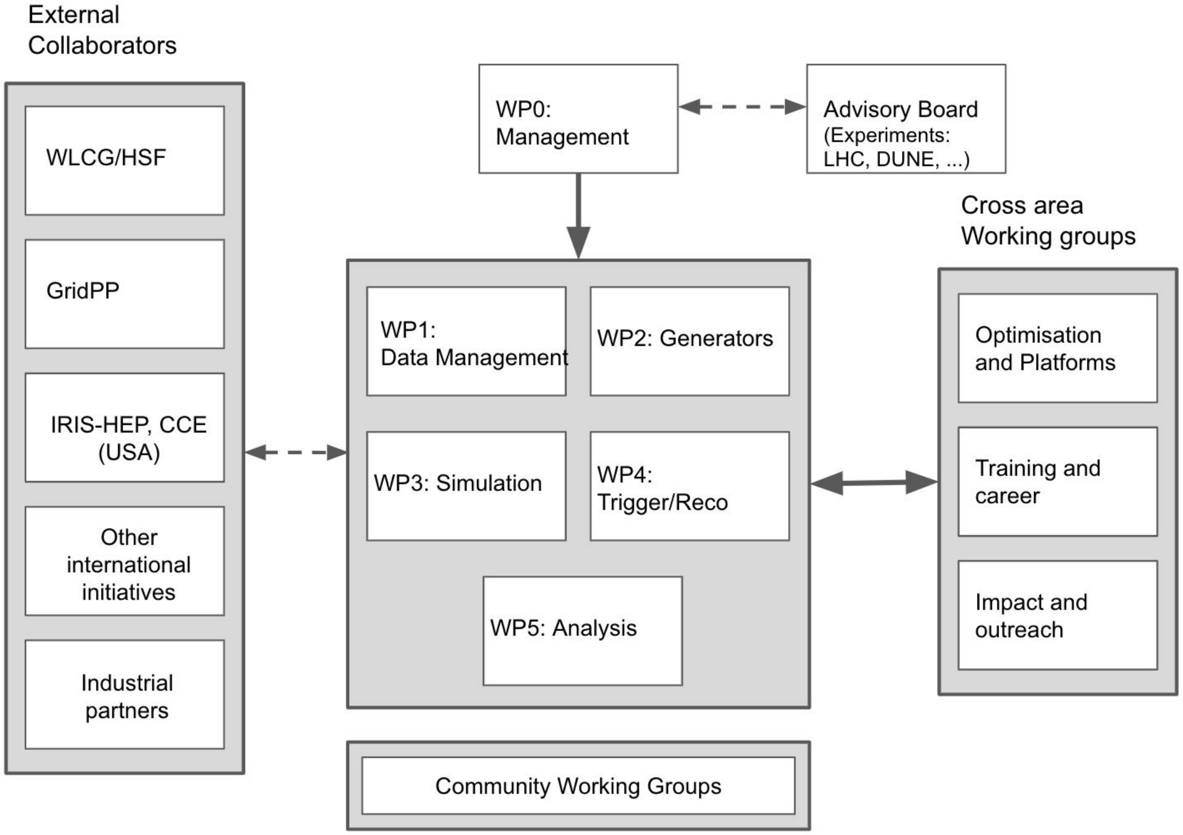
“One area of concern shared by the experiments and WLCG is finding means to ensure that the highly skilled personnel essential for R&D in computing and storage have meaningful career paths within the LHC community to provide for sustainability and the need for continual evolution over the lifetime of HL-LHC. “



Layer	Domain	Experiment 1	Experiment 2	Experiment 3
6	Physicists	Analysis code	...	...
5	Experiment Physicists programmer and software engineers	Analysis framework. Simulation, Reconstruction, Calibration Code	...	...
4	Experiment Software Engineers	Software Frameworks	...	...
3	Common Software HSF / SWIFT-HEP	Common software components (Data management, Generators, Geant4, Accelerator integration)		
2	GridPP / WLCG	Middleware infrastructure for Distributed Computing		
1	GridPP / WLCG	Physical Hardware		

Moving Software down the stack

What does SWIFT-HEP do?



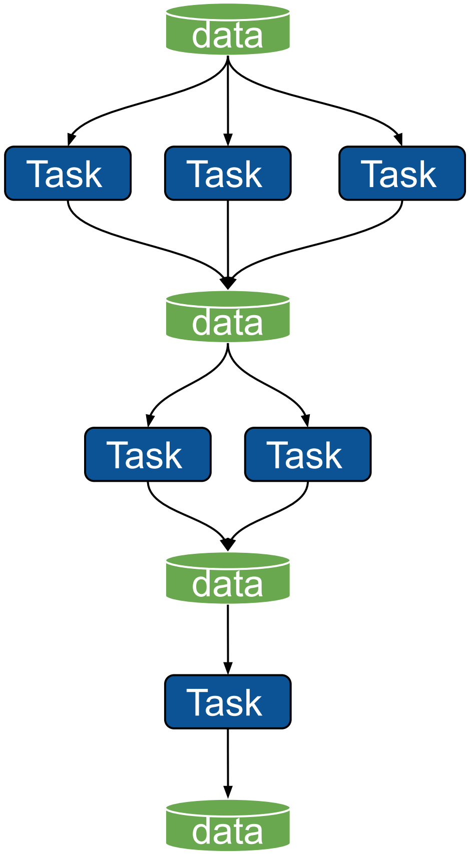
5 FTE. Funded for 3 years in 2021
Extended at flat FTE effort until 2025

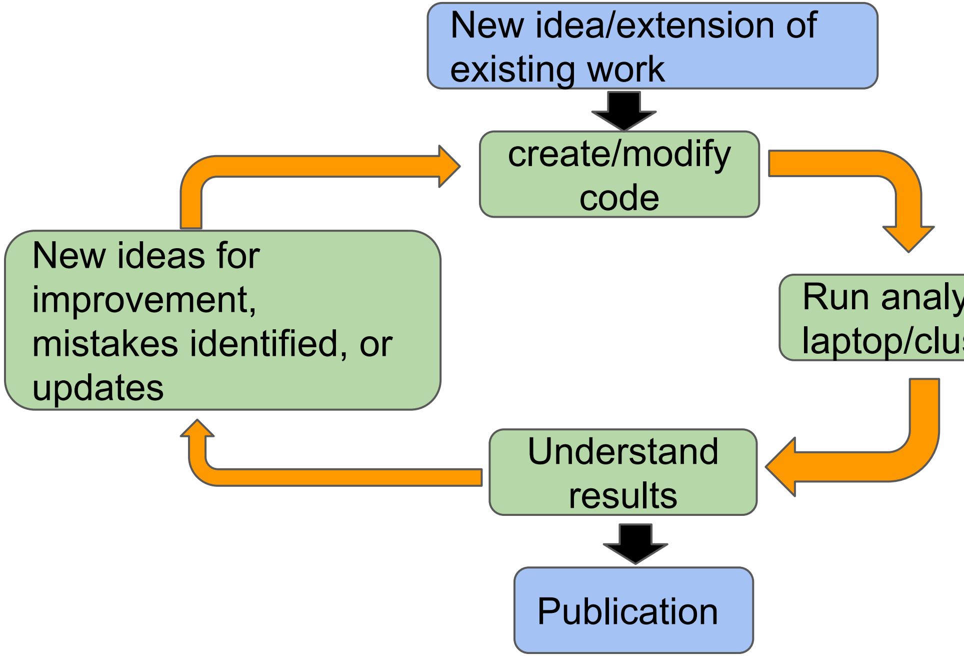
Different from IRIS, mostly staff effort, no capital

WP1: Data and workflow management
WP5: Analysis systems

Strongly connected, merged into a single one at the end of phase 1

Anatomy of an analysis workflow (simplified)




New idea/extension of existing work

create/modify code

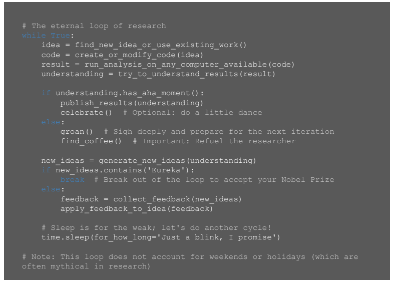
Run analysis on laptop/cluster/grid

Understand results

New ideas for improvement, mistakes identified, or updates

Publication

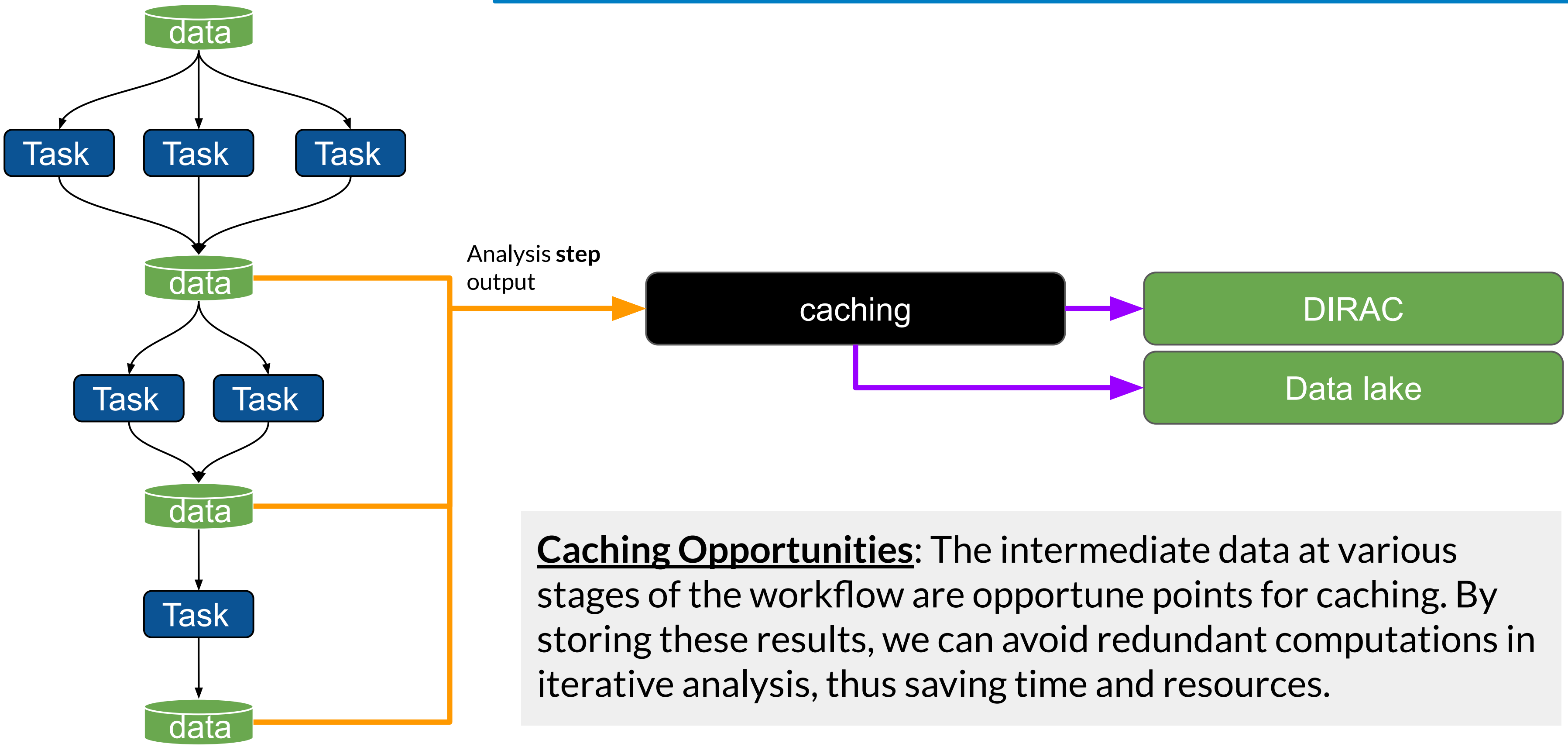
One cycle as short as a day or as long as a month



```
# The eternal loop of research
while True:
    idea = find_new_idea_or_use_existing_work()
    code = create_or_modify_code(idea)
    result = run_analysis_on_any_computer_available(code)
    understanding = try_to_understand_results(result)

    if understanding.has_aha_moment():
        publish_results(understanding)
        celebrate()  # Optional: do a little dance
    else:
        groan()  # Sigh deeply and prepare for the next iteration
        find_coffee()  # Important: Refuel the researcher

    new_ideas = generate_new_ideas(understanding)
    if new_ideas.contains('Eureka'):
        break  # Break out of the loop to accept your Nobel Prize
    else:
        feedback = collect_feedback(new_ideas)
        apply_feedback_to_idea(feedback)

    # Sleep is for the weak; let's do another cycle!
    time.sleep(for_how_long='Just a blink, I promise')

# Note: This loop does not account for weekends or holidays (which are
often mythical in research)
```

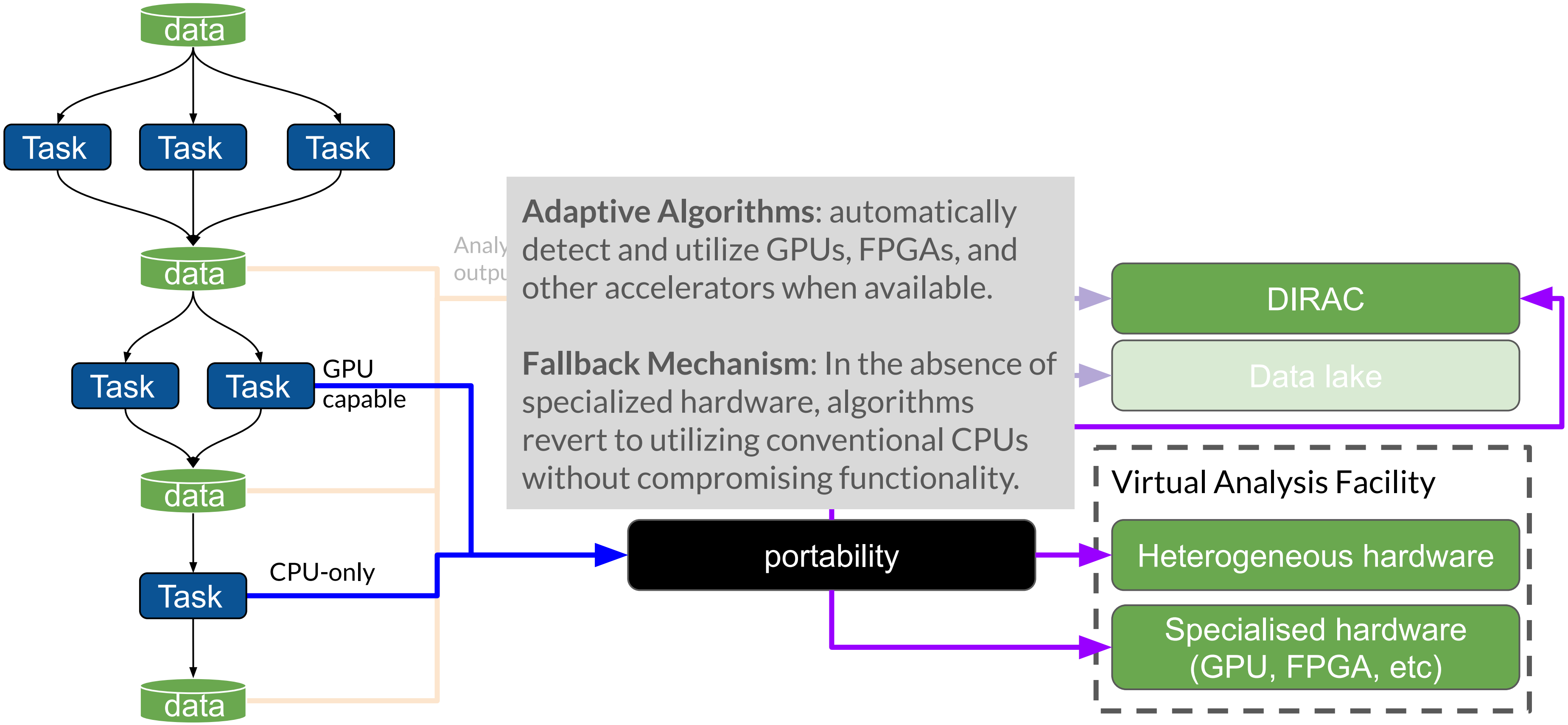
SWIFT-HEP WP 5 in a nutshell

Analysis workflow



<u>Caching Opportunities</u>: The intermediate data at various stages of the workflow are opportune points for caching. By storing these results, we can avoid redundant computations in iterative analysis, thus saving time and resources.

Analysis workflow



Adaptive Algorithms: automatically detect and utilize GPUs, FPGAs, and other accelerators when available.

Fallback Mechanism: In the absence of specialized hardware, algorithms revert to utilizing conventional CPUs without compromising functionality.

Analysis Grand Challenges (IRIS-HEP)

IRIS-HEP are planning to verify work through several analysis grand challenges

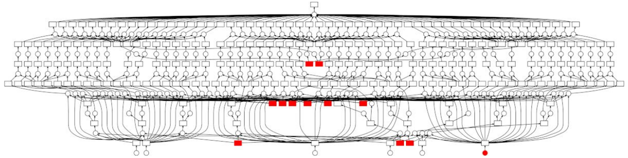
Aiming for a realistic workflow, e.g.

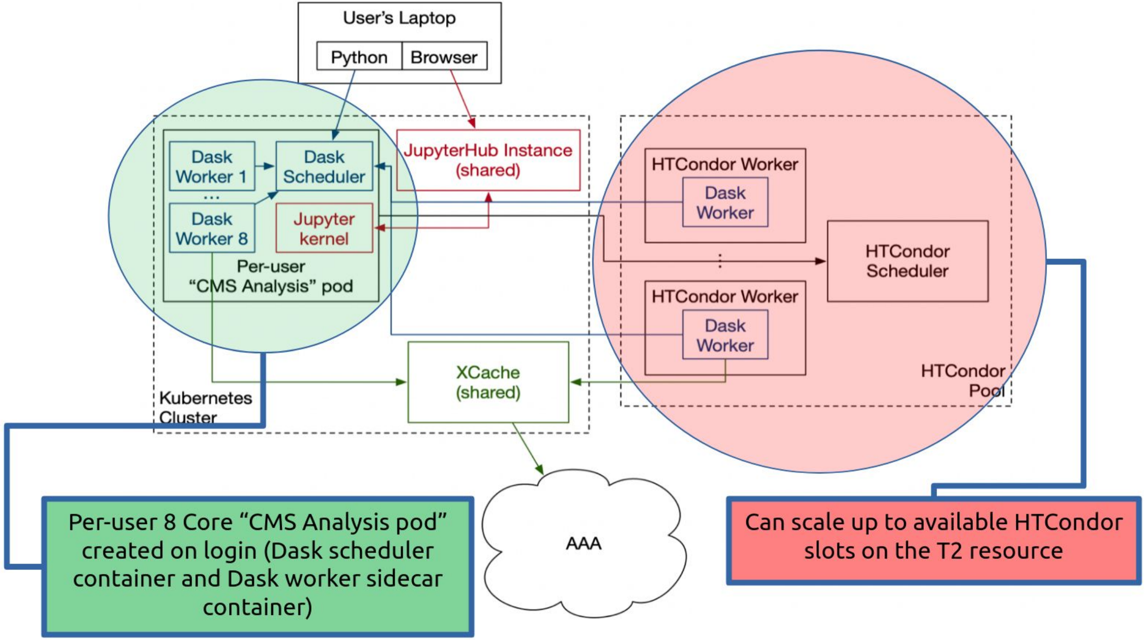
- Existing analysis, their example: Higgs → tau tau
- Approx 200 TB of input data, their example: CMS NanoAOD
- Testing performance (speed, resource usage)
- Outputs: statistical inference, tables, control plots, HEP Data
- Other metrics: reproducibility of results (e.g. with REANA)

→ ACG repo: https://github.com/iris-hep/analysis-grand-challenge

International view: IRIS-HEP

Uses Dask and dask-jobqueue





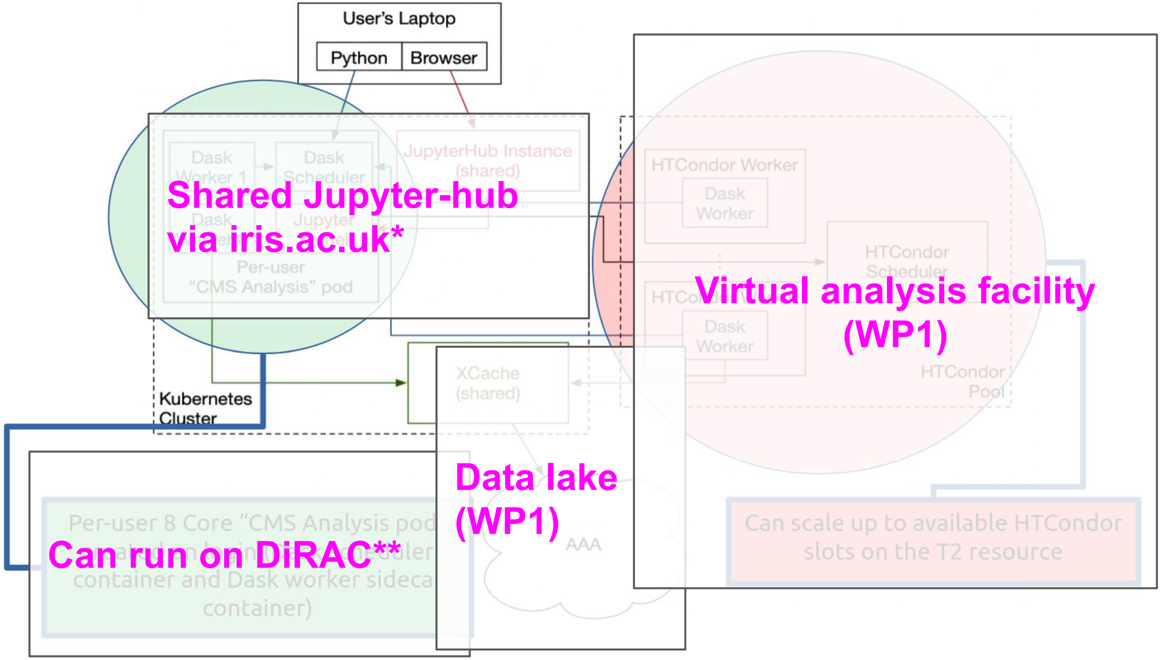
Per-user 8 Core "CMS Analysis pod" created on login (Dask scheduler container and Dask worker sidecar container)

Can scale up to available HTCondor slots on the T2 resource

From coffea-casa docs

International view: IRIS-HEP

As simple as adding DIRAC jobqueue to dask-jobqueue?


User's Laptop
Python
Browser
Dask Worker 1
Dask Scheduler
JupyterHub Instance (shared)
Dask
Jupyter
Per-user "CMS Analysis" pod
Kubernetes Cluster
XCache (shared)
HTCondor Worker
Dask Worker
HTCondor Scheduler
HTCondor
Dask Worker
HTCondor Pool
AAA
Per-user 8 Core "CMS Analysis pod"
container and Dask worker sidecar container)
Can scale up to available HTCondor slots on the T2 resource

Shared Jupyter-hub via iris.ac.uk*

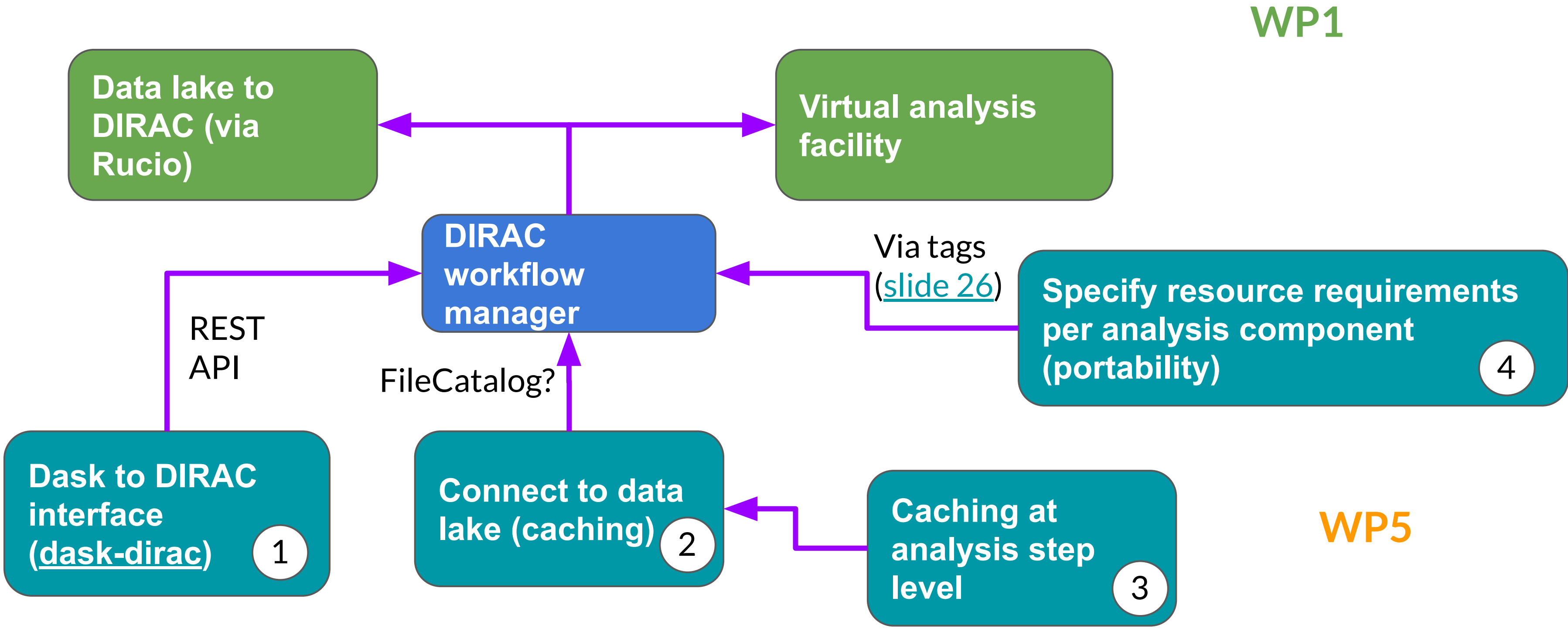
Virtual analysis facility (WP1)

Data lake (WP1)

Can run on DiRAC**

*no relation to IRIS-HEP; **no relation to DIRAC

Roadmap overview

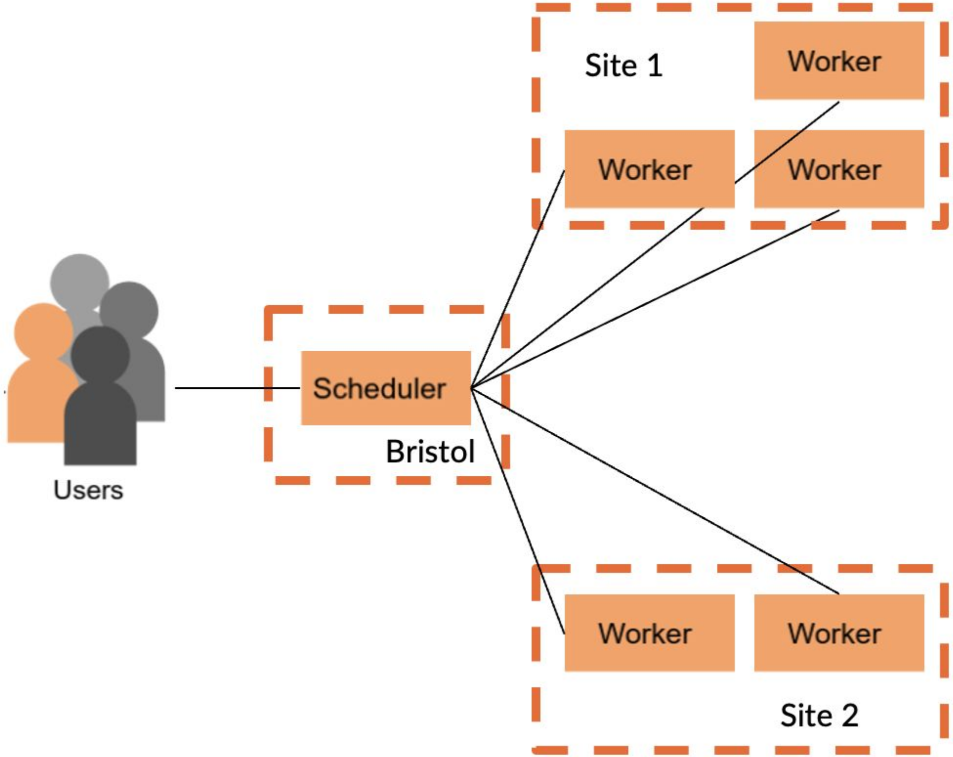

WP1
Data lake to DIRAC (via Rucio)
Virtual analysis facility
DIRAC workflow manager
Via tags (slide 26)
Specify resource requirements per analysis component (portability) 4
REST API
FileCatalog?
Dask to DIRAC interface (dask-dirac) 1
Connect to data lake (caching) 2
Caching at analysis step level 3
WP5

Closes example of what we want to achieve: Dask-based Distributed Analysis Facility (kubernetes slides)

Going beyond a single site

Use the Analysis Grand Challenge as a test case (CMS data)

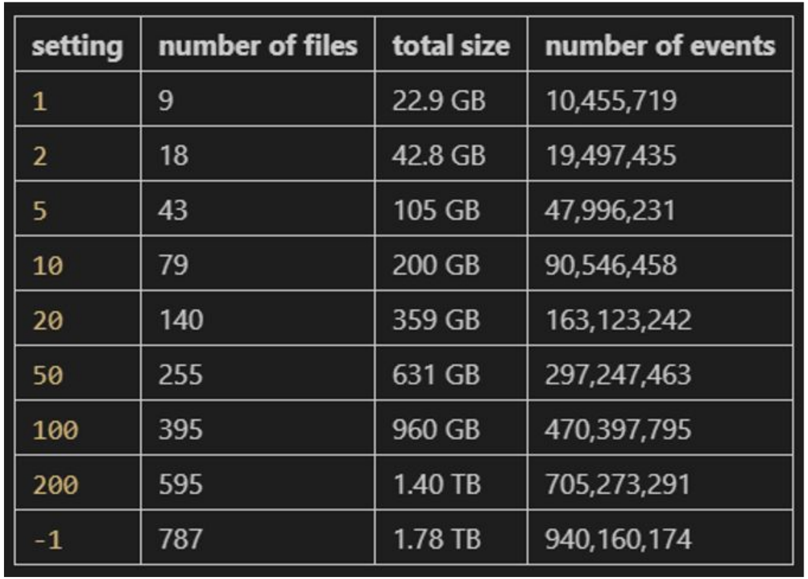
Workers and Scheduler needed to be able to talk to each other -> firewalls stopped this at other sites



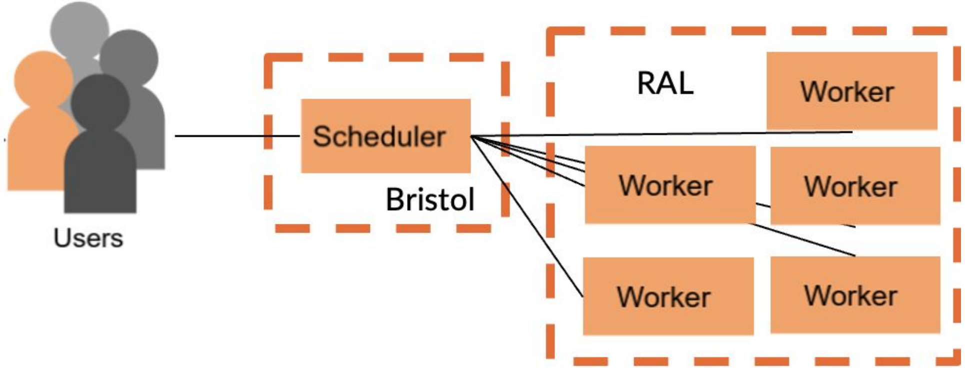
Tests with RAL PPD

Luckily, RAL has nodes which have open ports. Will have to discuss security settings to run the prototype on multiple sites

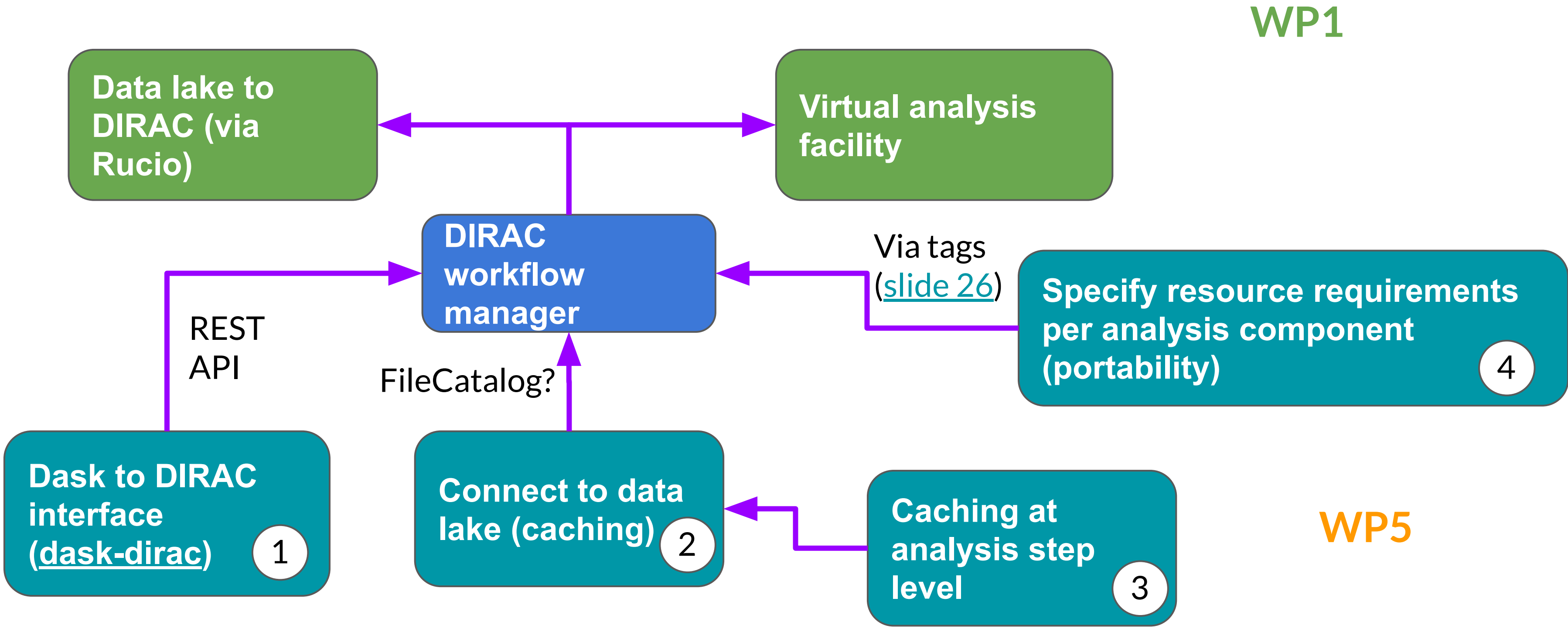
Can look at some basic benchmarking



setting	number of files	total size	number of events
1	9	22.9 GB	10,455,719
2	18	42.8 GB	19,497,435
5	43	105 GB	47,996,231
10	79	200 GB	90,546,458
20	140	359 GB	163,123,242
50	255	631 GB	297,247,463
100	395	960 GB	470,397,795
200	595	1.40 TB	705,273,291
-1	787	1.78 TB	940,160,174



Roadmap overview



WP1

Data lake to DIRAC (via Rucio)

Virtual analysis facility

DIRAC workflow manager

Via tags (slide 26)

Specify resource requirements per analysis component (portability) 4

REST API

FileCatalog?

Dask to DIRAC interface (dask-dirac) 1

Connect to data lake (caching) 2

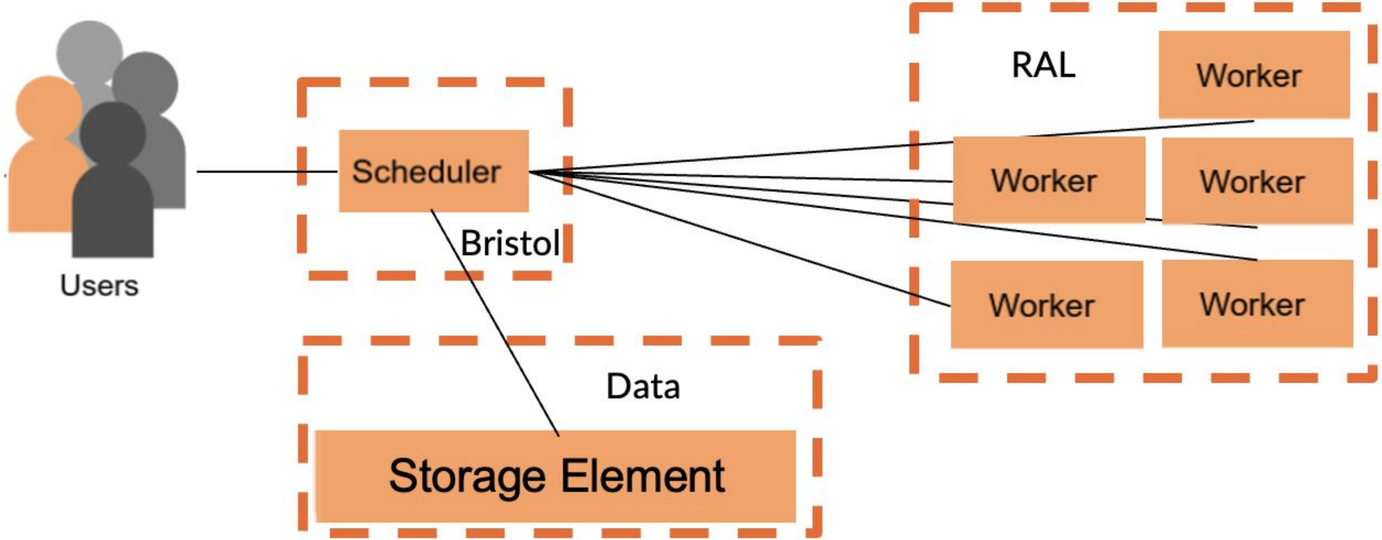
Caching at analysis step level 3

WP5

Closes example of what we want to achieve: Dask-based Distributed Analysis Facility (kubernetes slides)

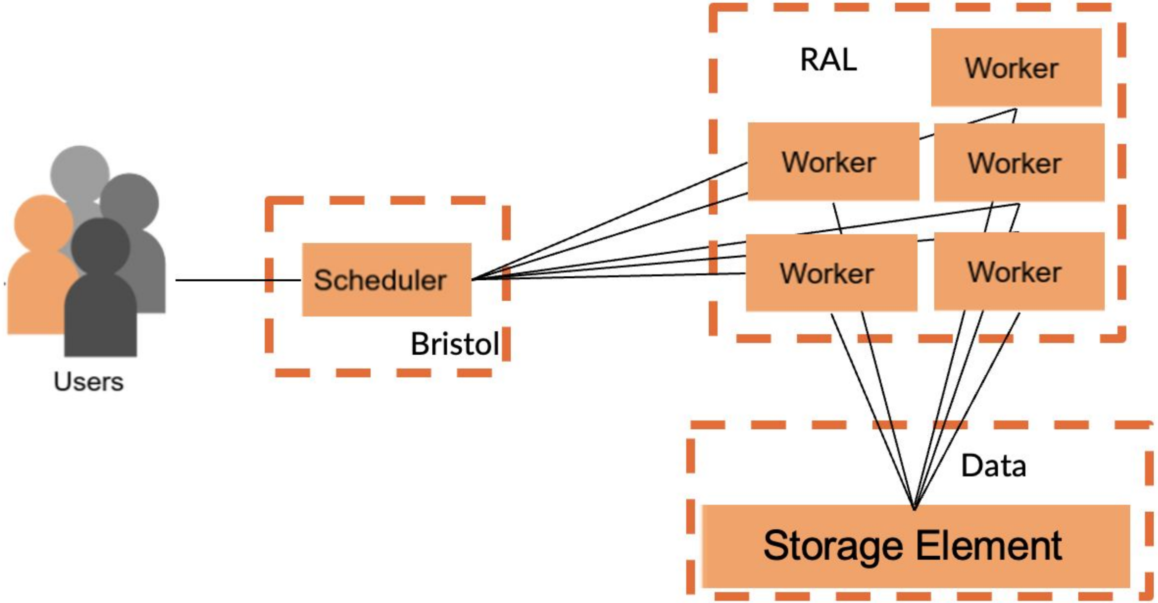
Interacting with the data lake

Currently the scheduler interacts with the storage element directly



Interacting with the data lake

While Rucio should handle this file manipulation



The demonstrator and SWIFT-HEP developments

The final goal is to simplify the interaction with data and compute for users (analysis and smaller communities)

Several challenges, but also several opportunities

- Workflow management: dask interface, multi-site interaction, scaling
- Planned evolution of DiracX
- Data management: data lake deployment across the UK with Kubernetes (achieved) Use of QoS to make sure analysis jobs get best I/O