SWIFT HEP
SoftWare InFrastructure and Technology for High Energy Physics

2023.11.22 SWIFT-HEP #6

Analysis Systems introduction

Luke Kreczko for SWIFT-HEP WP5

University of BRISTOL

Key points in HEP Data Analysis

Physics

Last mile of long chain of data recording and processing.
Goals: **gain insight and create new knowledge**

Computing

Analysis workflow (data + software) depends on experiment, analysis group, subset of data (signal + relevant backgrounds), analysis iteration.

Flexibility is paramount.

WP5 key points

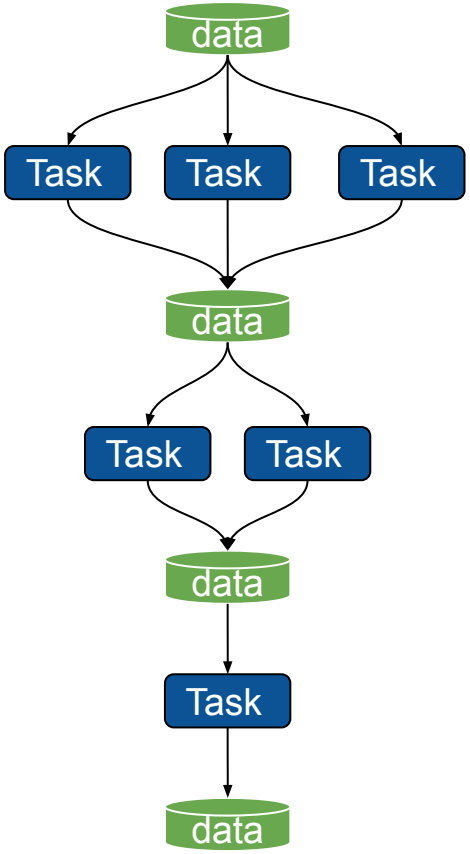
Seamless Access to Computing Resources

Ease of access to distributed computing resources through user-friendly interfaces and “industry standards”

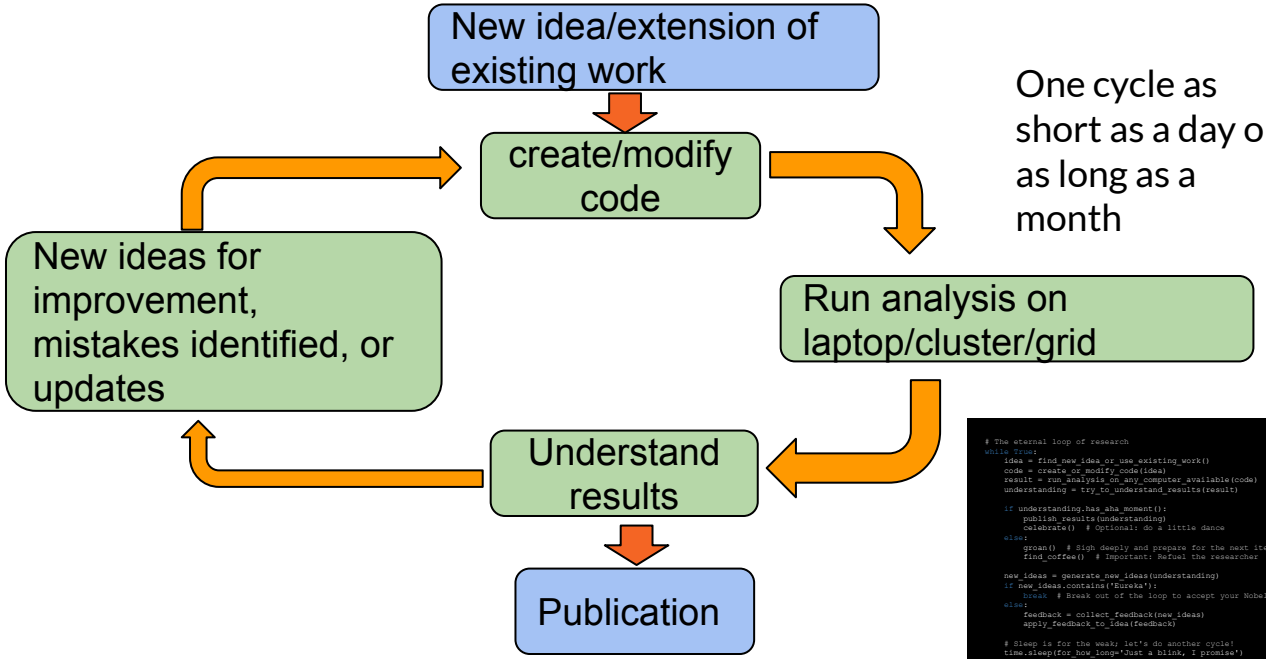
Efficiency

Caching for fast iterations, portability for mapping algorithms to hardware accelerators (GPU, FPGA)

Anatomy of an analysis workflow



The cycle (oversimplified)

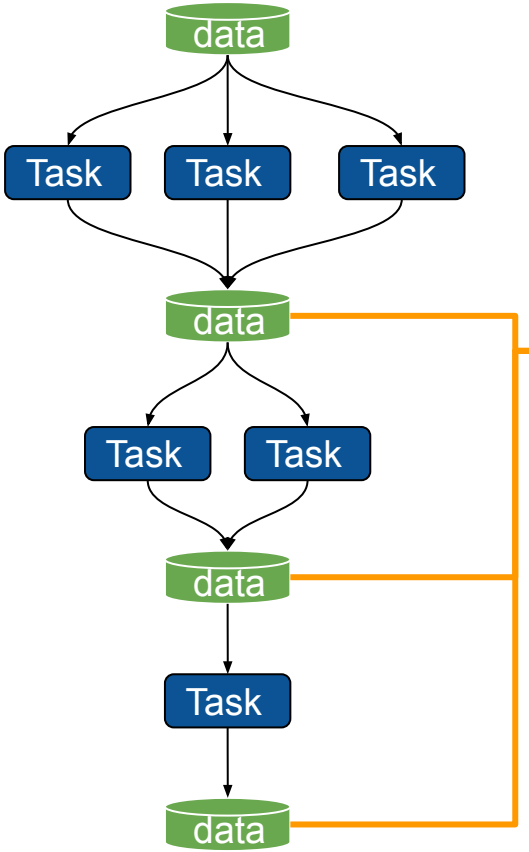


One cycle as short as a day or as long as a month

```
# 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

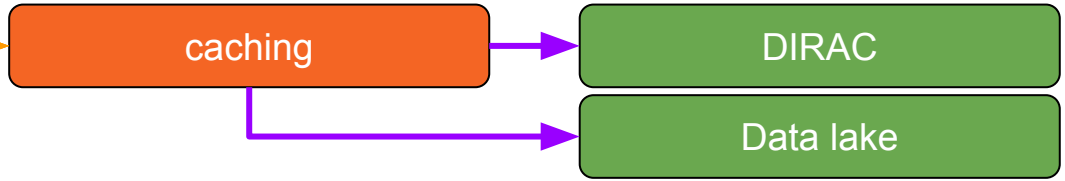
Analysis workflow



WP5

WP1

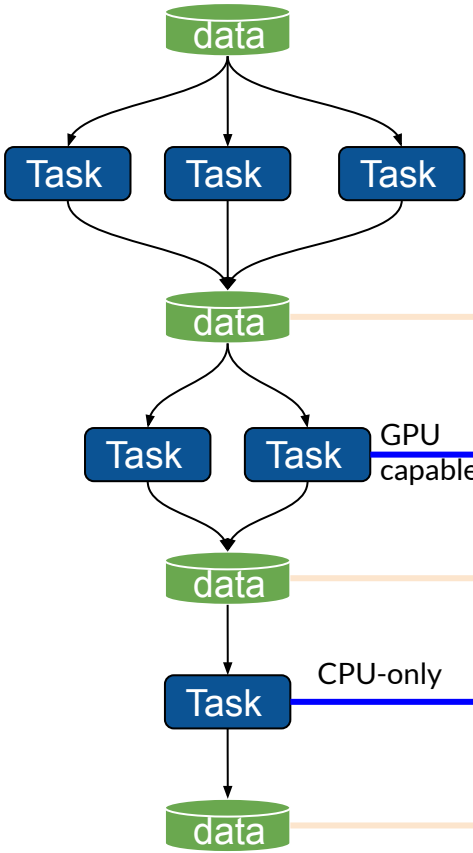
Analysis **step** output



Caching Opportunities: 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.

SWIFT-HEP WP 5 in a nutshell

Analysis workflow



WP5

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.

portability

WP1

DIRAC

Data lake

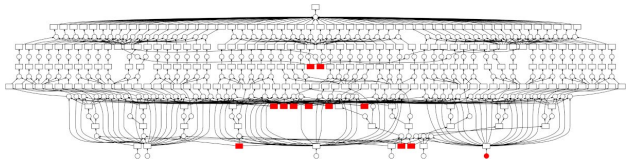
Virtual Analysis Facility

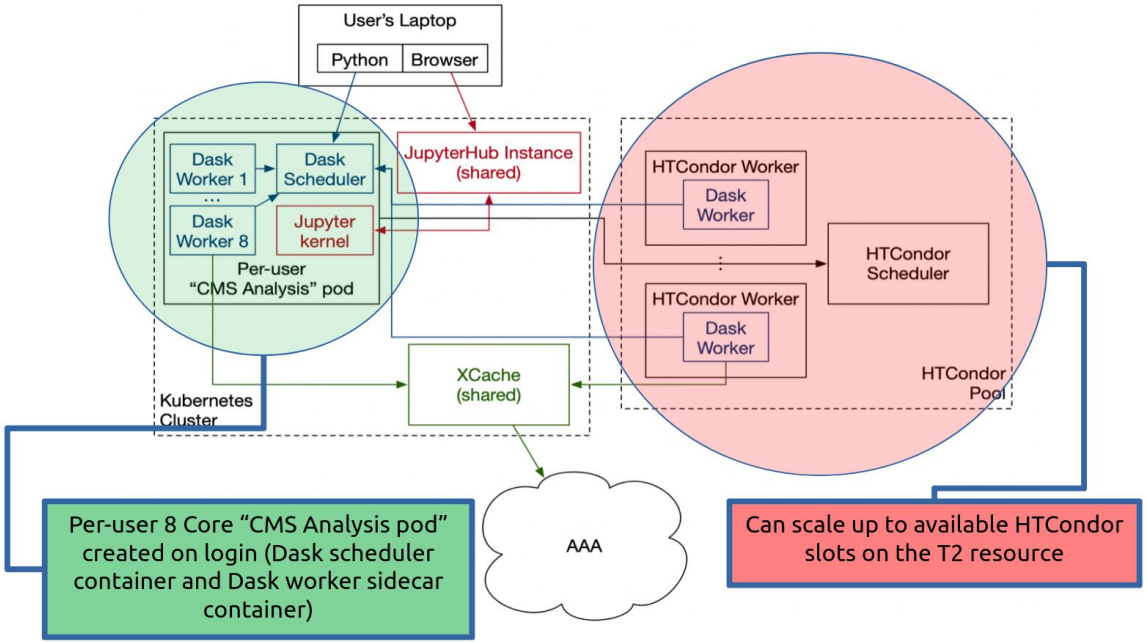
Heterogeneous hardware

Specialised hardware (GPU, FPGA, etc)

Scheduling with coffea-casa

Uses Dask and dask-jobqueue





From coffea-casa docs

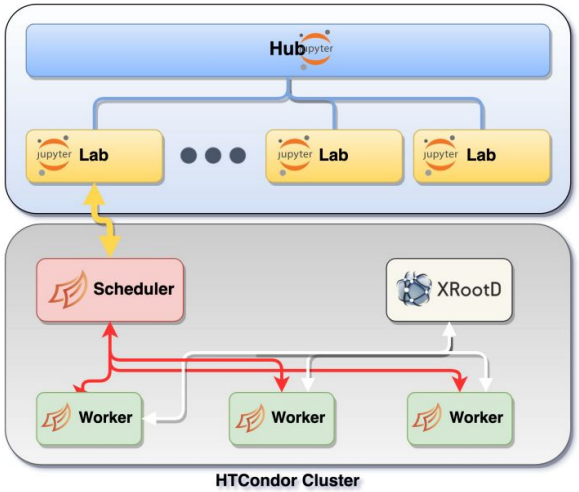
INFN: a CMS project

The overall idea, from where we started

Integration work of well established technologies

- **JupyterHub** (JHub) and **JupyterLab** (JLab) to manage the **user-facing part of the infrastructure**
- **DASK** to introduce the **scaling over a batch system**
- **XRootD as data access protocol** toward AAA:
  - Here we foresee the usage of caching layers (see later)

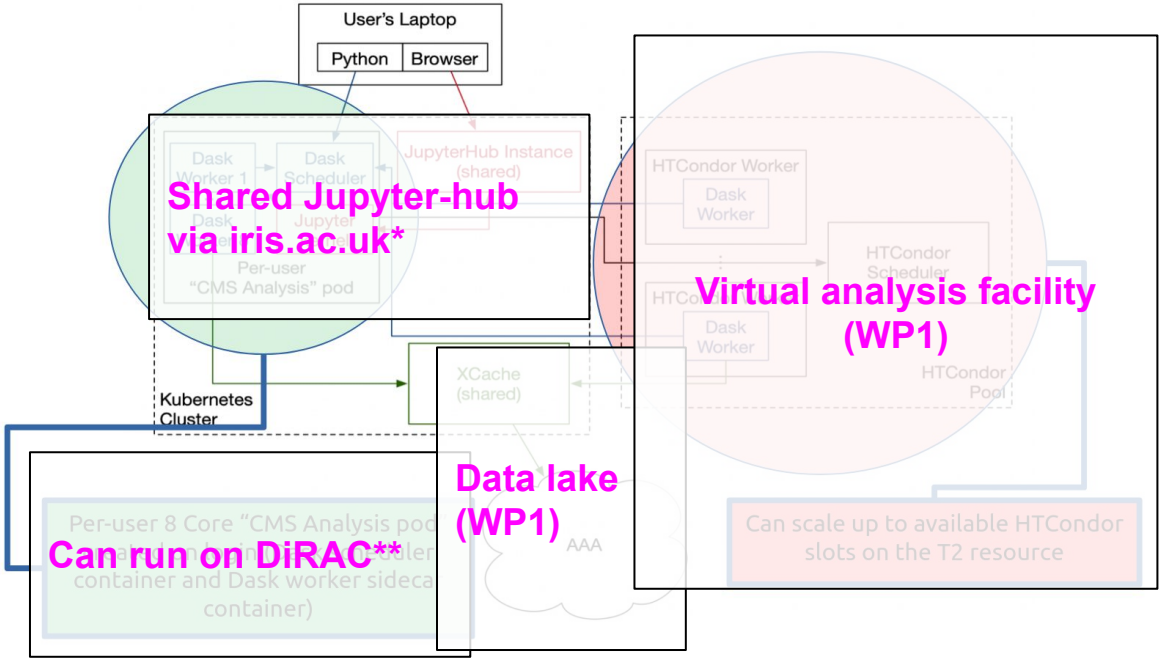
So far we opted for scaling over HTCondor:
⇒ User prioritization and in general configuration tuning is under study



From Analysis facilities forum (28.07.2022)

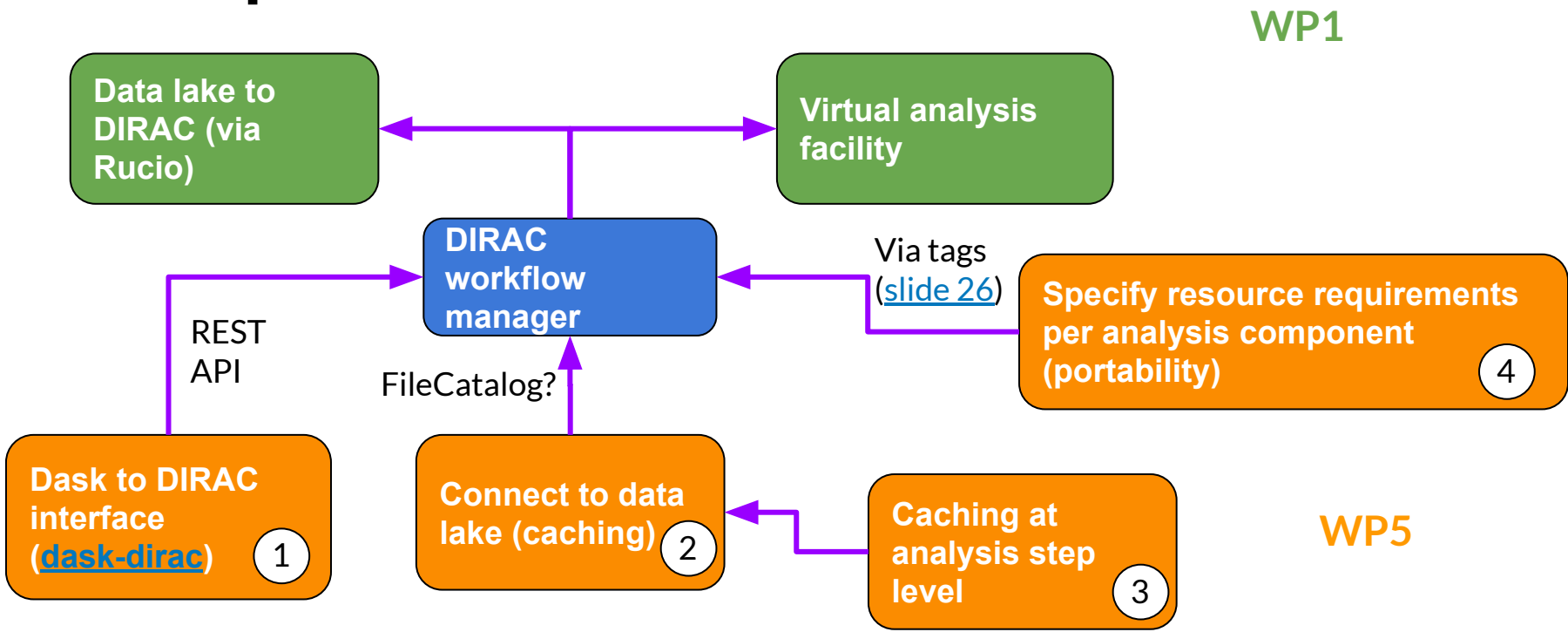
SWIFT-HEP + GridPP == experiment agnostic?

As simple as adding DIRAC jobqueue to dask-jobqueue?



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

Roadmap overview



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

Roadmap overview

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

Analysis Grand Challenges

The medium for testing

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

SWIFT-HEP Phase 2

Phase 2?

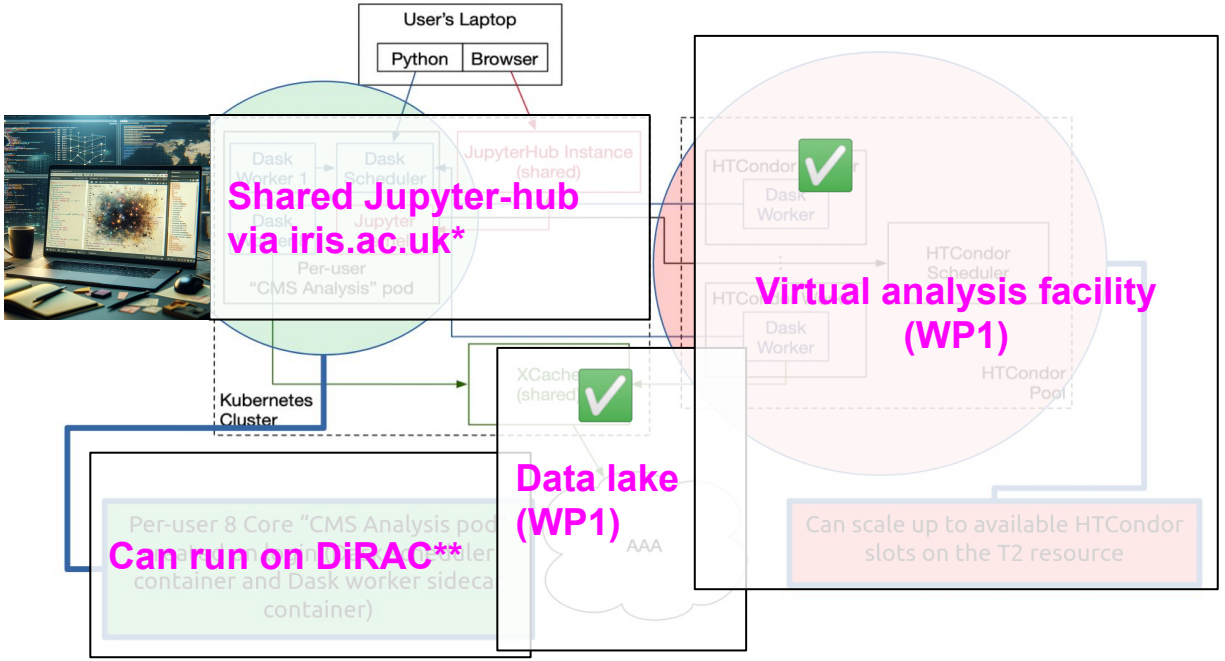
Phase 1 is a technology test/prototype

Phase 2 would be the production system:

Jupyter-hub with IRIS IAM on STFC cloud***

Sample of analyses beyond ACGs

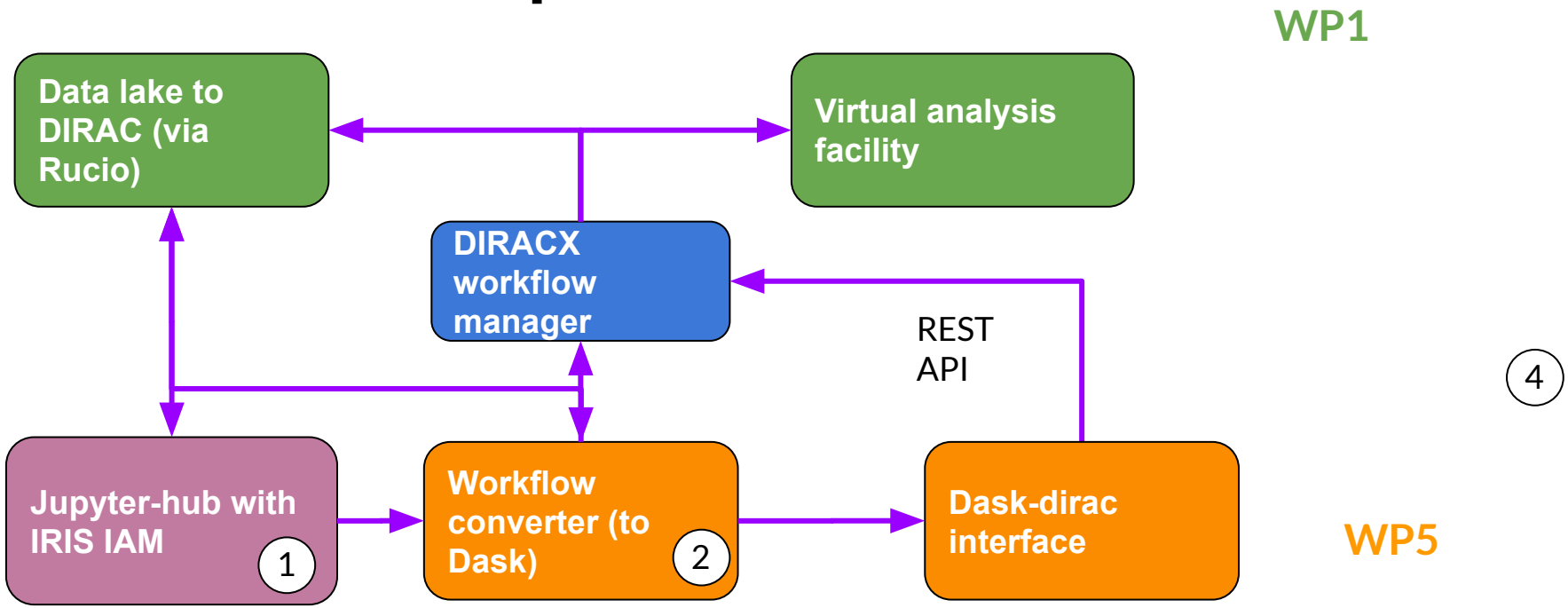
Test phase 1 at scale



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

*** These could also be institutional resources with access to both local and GridPP resources

Phase 2 Roadmap overview



Would also include optimising caching and portability as analysis sample size increases.
In other words: phase 2 is a superset of phase 1.

Summary and Outlook

In Phase 1 WP5 aims for a prototype demonstrating the key goals on WP1 deliverables

WP5 is in progress, using AGCs as benchmark - details in Sam's talk

Phase 2 will look towards a production system:

- **Extend** capabilities **beyond ACG** benchmarks to encompass wider analyses
- Integrate JupyterHub or an equivalent platform to provide a **user-friendly interface for researchers**.

Backup slides

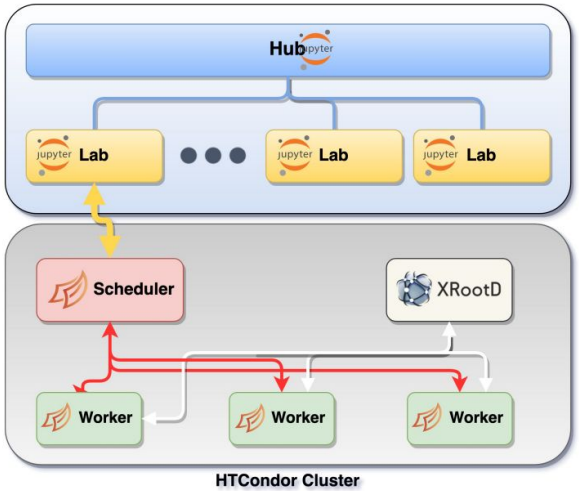
INFN: a CMS project

The overall idea, from where we started

Integration work of well established technologies

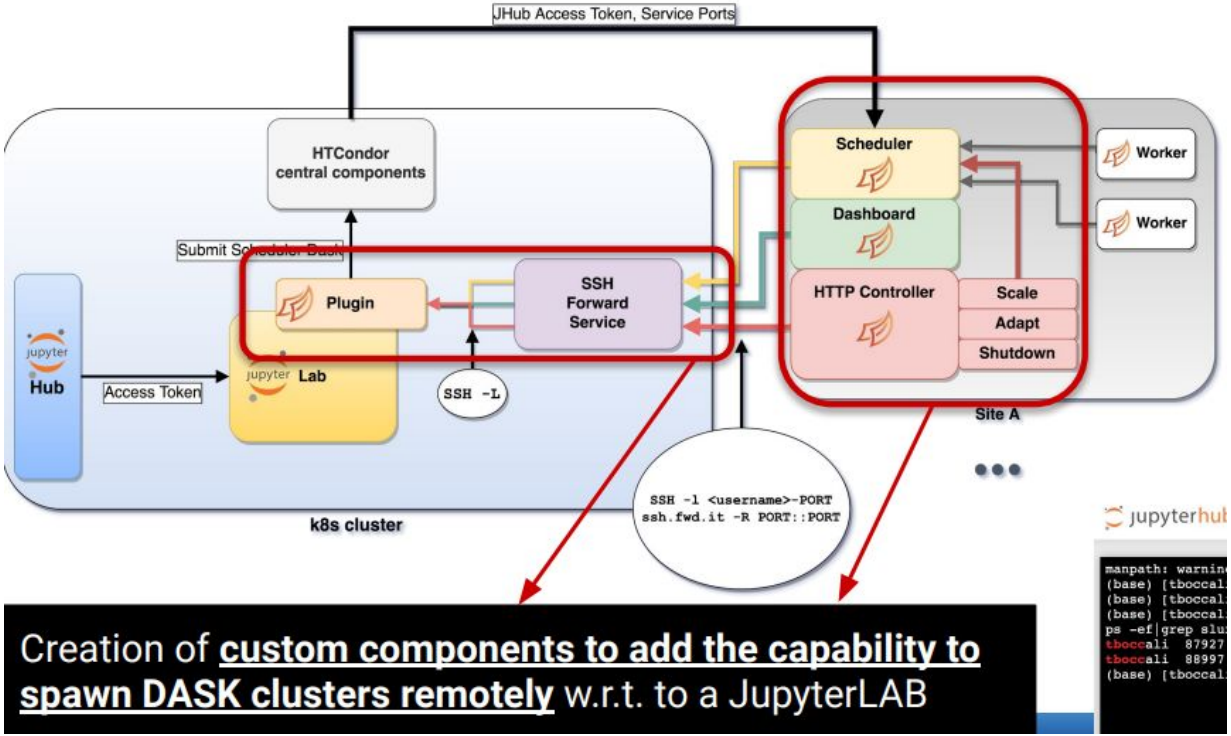
- **JupyterHub** (JHub) and **JupyterLab** (JLab) to manage the **user-facing part of the infrastructure**
- **DASK** to introduce the **scaling over a batch system**
- **XRootD as data access protocol** toward AAA:
  - Here we foresee the usage of caching layers (see later)

So far we opted for scaling over HTCondor:
⇒ User prioritization and in general configuration tuning is under study



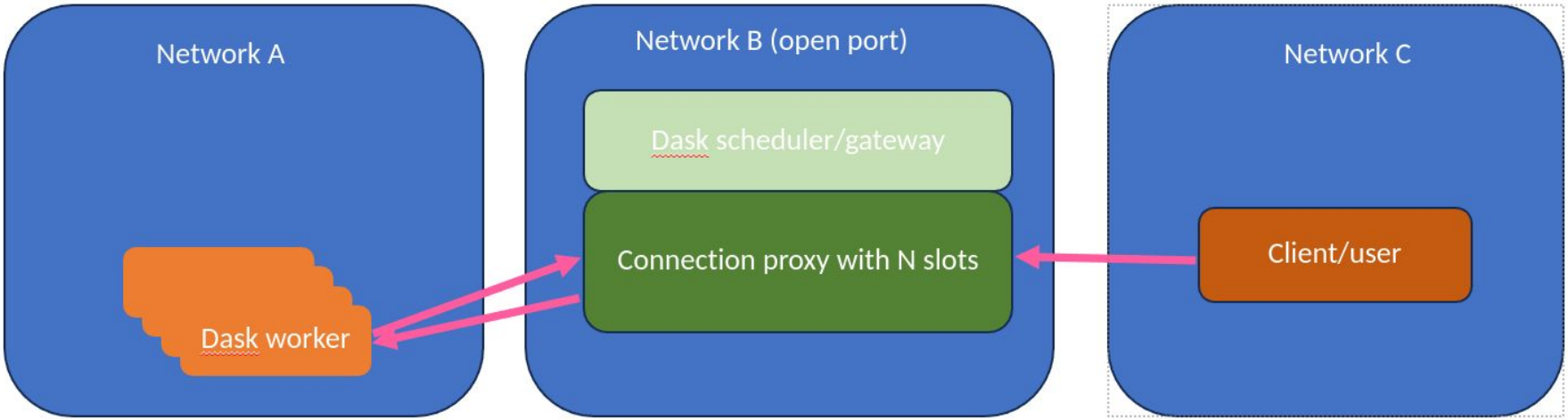
From Analysis facilities forum (28.07.2022)

But:



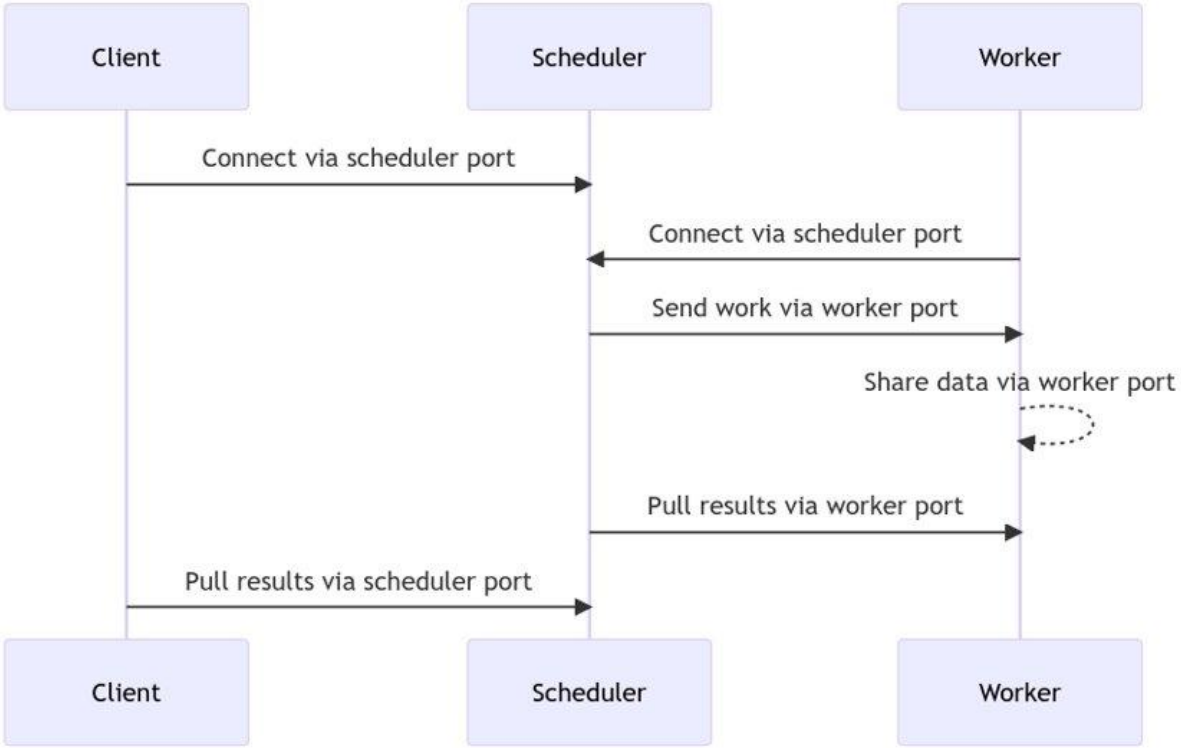
Is there a way to standardize this functionality and make it available to everyone?

Dask network layer simplified



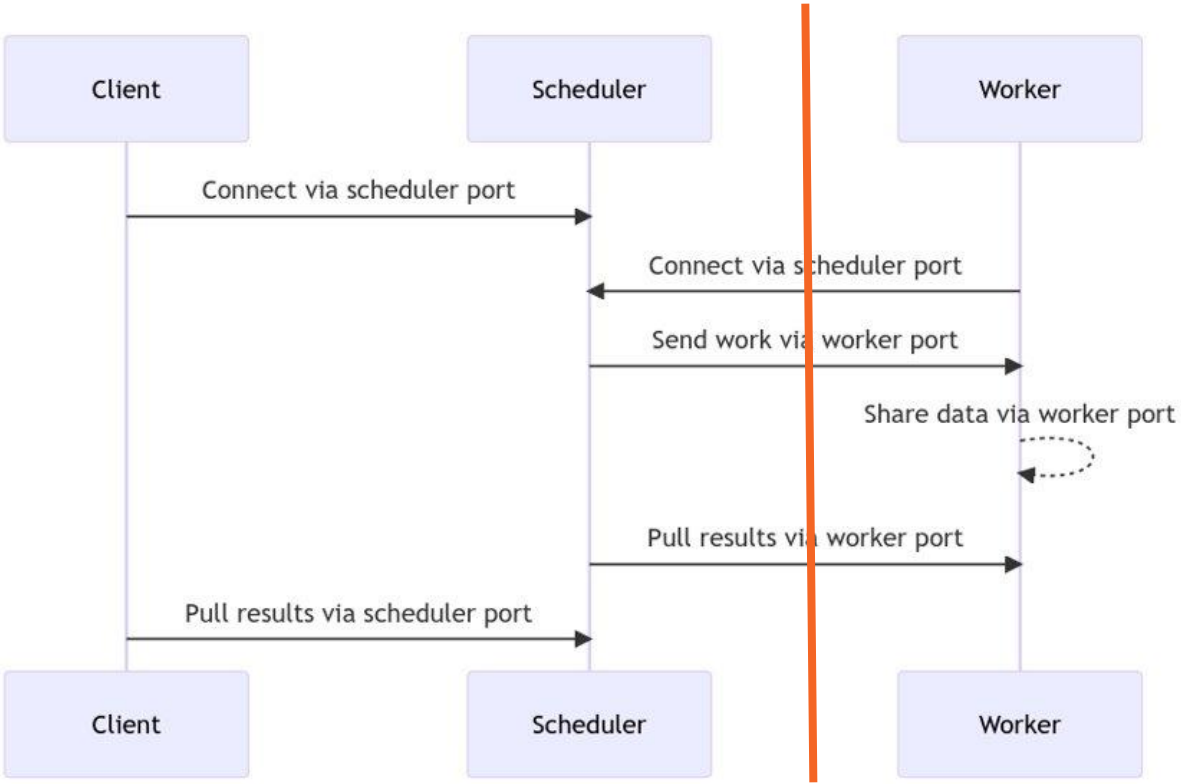
Connection broker can reuse connections

Dask network layer simplified



mermaidjs source

Dask network layer simplified



So where is the problem?

Imagine network boundary between scheduler and workers

Scheduler port is accessible from workers

Worker port is **ONLY** accessible to scheduler if connection is recycled (part of **ESTABLISHED** --> firewall OK)

Default Dask operation: this can happen at **RANDOM** (most likely for small # of workers)

Dask network layer: A general fix

We know connections can be recycled and bypass firewall if they are part of an **ESTABLISHED** connection

We also know of a working solution in our field: The **HTCondor Connection Broker**

Workers, schedulers, etc connect to a **SHARED_PORT**

As long as **SHARED_PORT is open in firewall** on a node accessible to both scheduler and workers --> connection can be established

Most simple solution: Can the **Dask Connection proxy be rewritten to hold worker connections**?

What are the downsides for 100-1000 worker nodes?

CPU vs accelerators

No clear optimal way yet, first draft will require multiple versions of a “stage”:

```
register:
 my_namespace::my_stage:
  cpu: my_module.my_stage
  gpu: my_module.my_stage_gpu
```

GPU version is used if a GPU is detected, CPU version otherwise