29th-30th March 2023
GridPP49 & SWIFT-HEP Workshop

WP5 Overview

Sam Eriksen for SWIFT-HEP WP5

University of Bristol

Overview

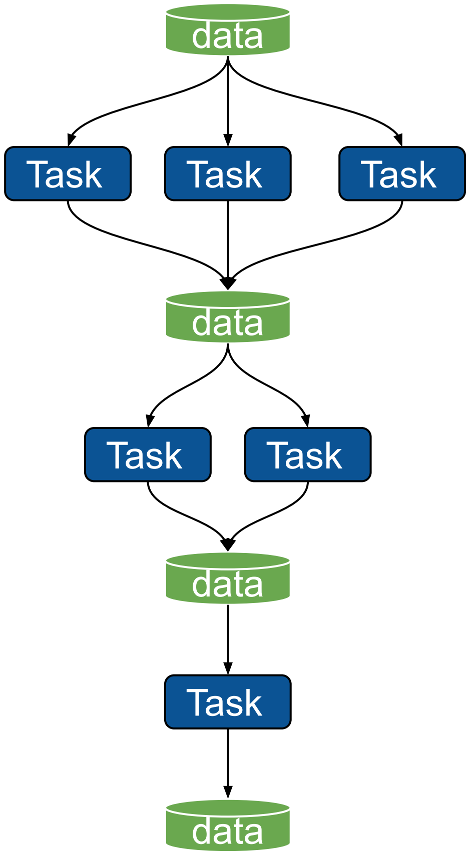
- Overview of WP5 and the roadmap
- Current progress in WP5 (dask-dirac)
- Planned Analysis Grand Challenges

WP5: Analysis Systems

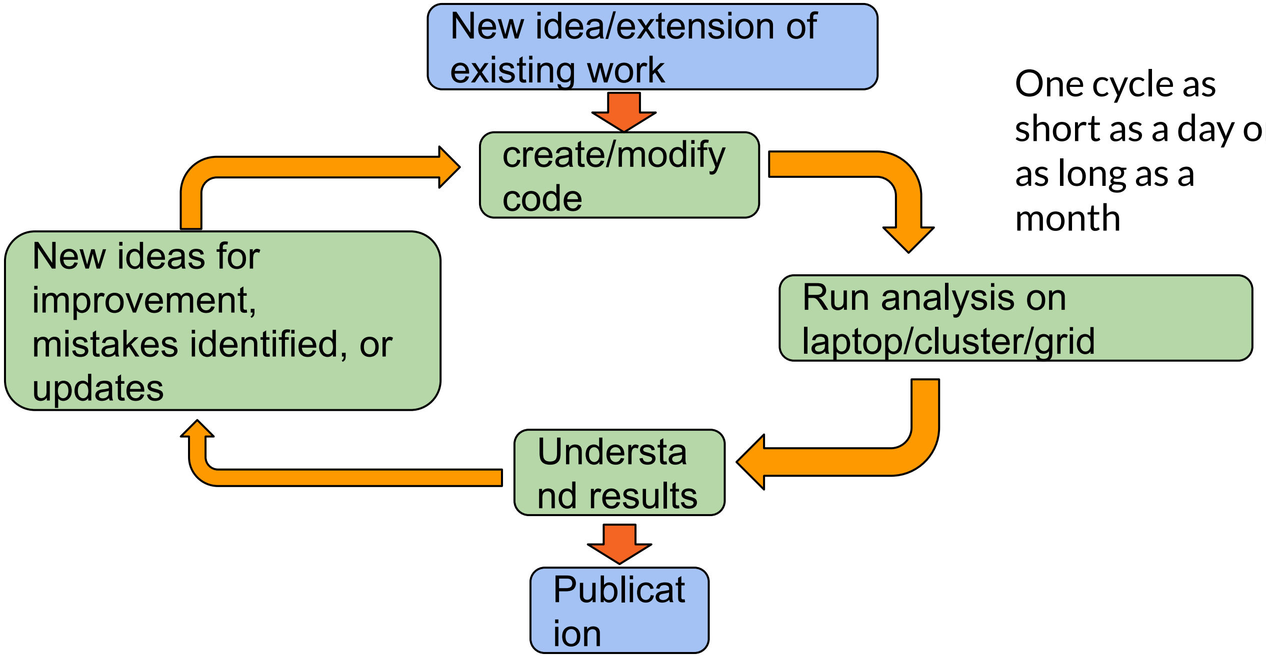
WP5: Analysis Systems

Run analysis workloads optimally on distributed resources

Anatomy of an analysis workflow

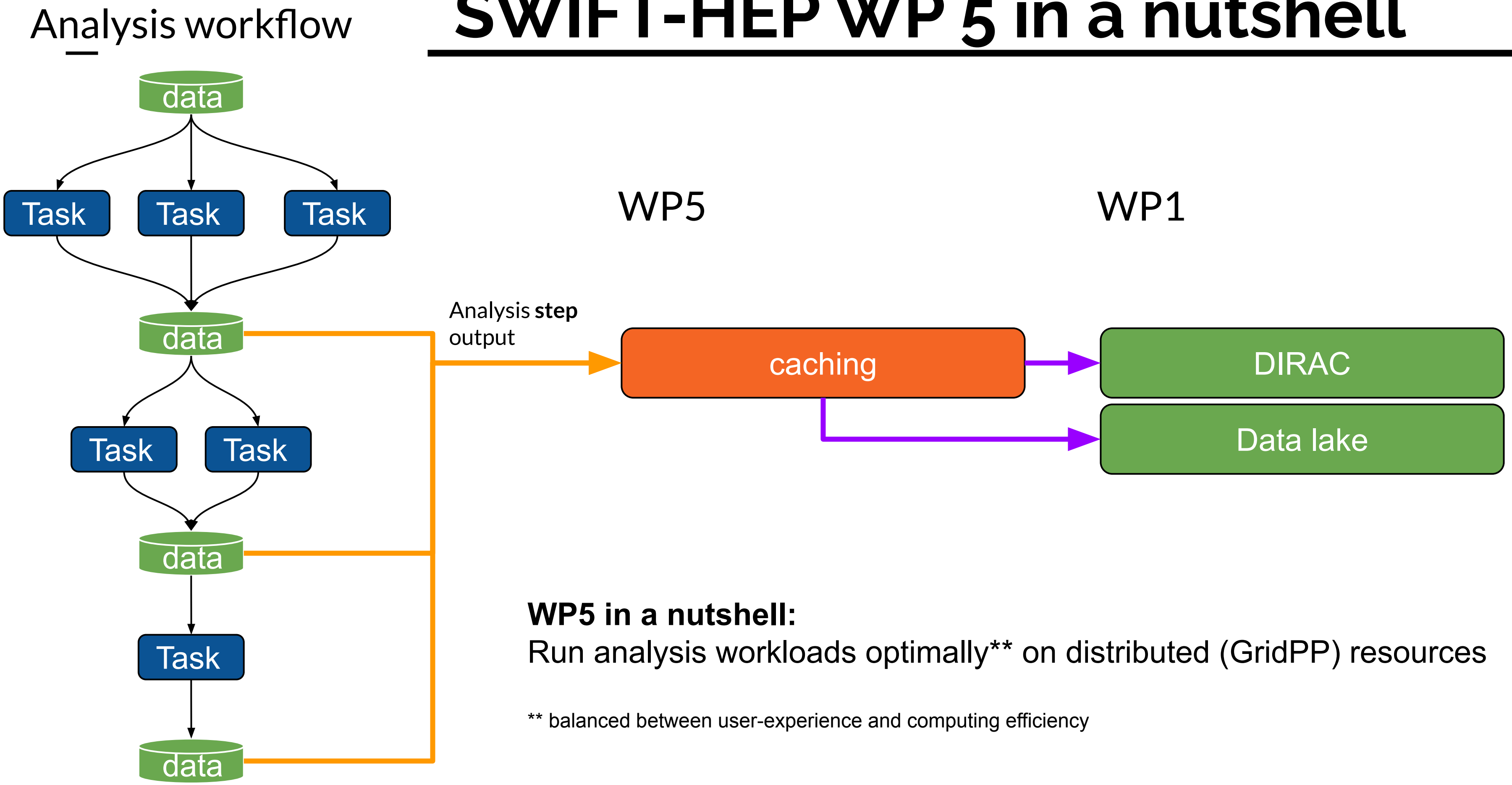


The cycle (oversimplified)



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

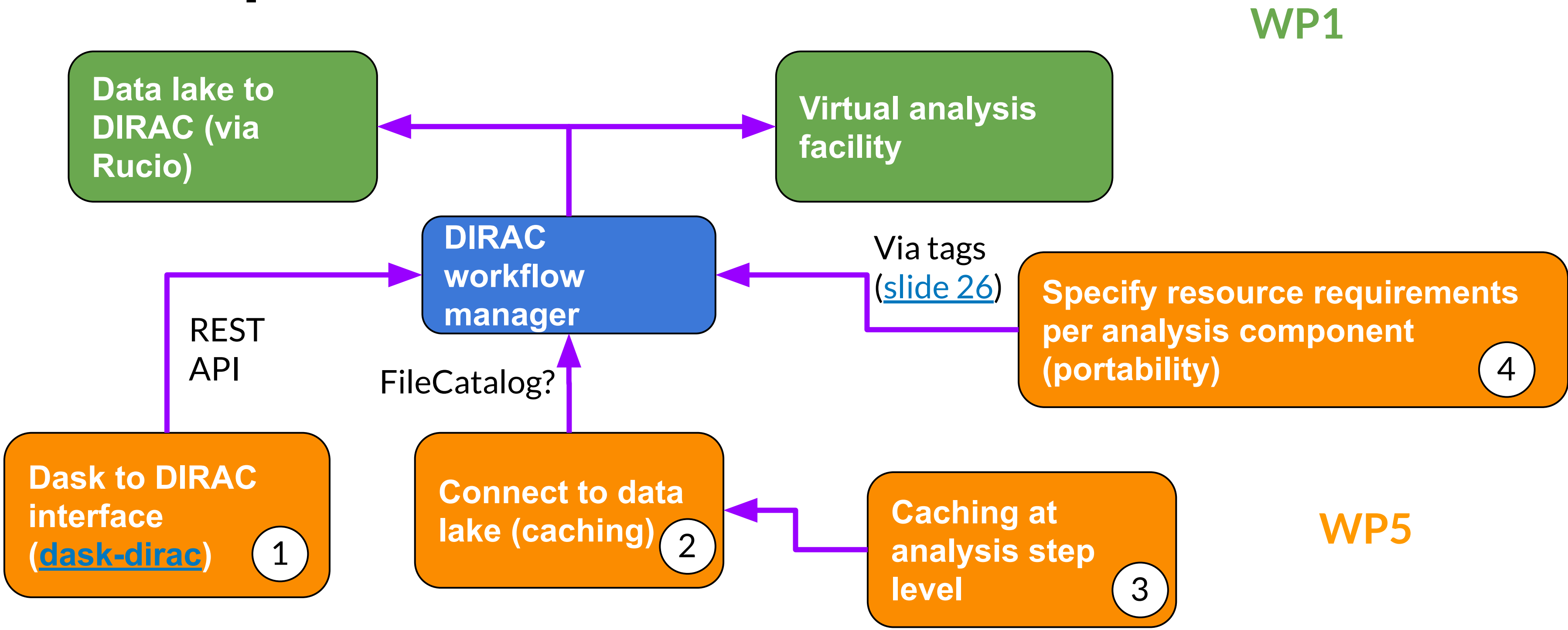
SWIFT-HEP WP 5 in a nutshell



WP5 in a nutshell:
Run analysis workloads optimally** on distributed (GridPP) resources

** balanced between user-experience and computing efficiency

Roadmap overview



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

Intermission: Dask

Scale any Python code

Parallelize any Python code with Dask Futures, letting you scale any function and for loop, and giving you control and power in any situation.

From https://www.dask.org/

Dask can submit to most batch systems all the same - fantastic from users' perspective

Can we use Dask in HEP?

- Coffea (Analysis Grand Challenges) ✓
- RDataframe ✓
- Awkward-array (native support) 

- FAST-HEP (custom graphs) 

If infrastructure can be used via Dask → wide use is possible

SWIFT-HEP Phase 2 (informal planning)

If phase 1 is the prototype, phase 2 will be the production-ready product

1. Scale up: user interface (Jupyter hub, Dask gateway) to accommodate GridPP communities
2. Automate user experience: analysis computing & physics reports
3. Data security and separation: Develop with WP1 data lake
4. Play on GridPP strengths: Include more non-LHC communities
5. Outreach: A simple enough system could be use for masterclass to demonstrate HEP analysis at scale (with real data)

Progress since last workshop

Dask to DIRAC interface (dask-dirac)

Step #1 of WP5
Talking to DIRAC via HTTP

DIRAC is adding HTTP support

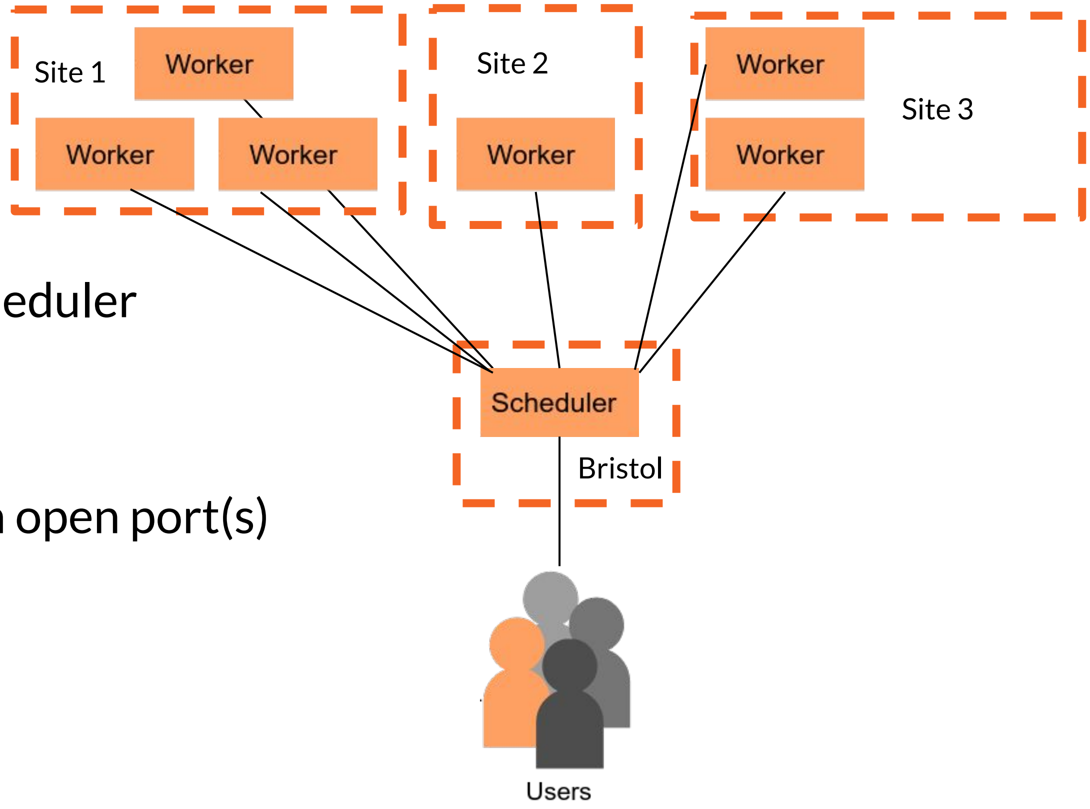
- Version 7.3 brings JobManager, JobMonitor, JobStateUpdate
- Version 8.0 brings more and token support
- Version 8.1: Everything available via HTTP
- Can utilize HTTP to make a “dependency free” DIRAC client
- Github repository
- Once operational, test on Analysis Grand Challenges

dask_dirac

- Within dask_jobqueue replace PBSCluster with DiracCluster

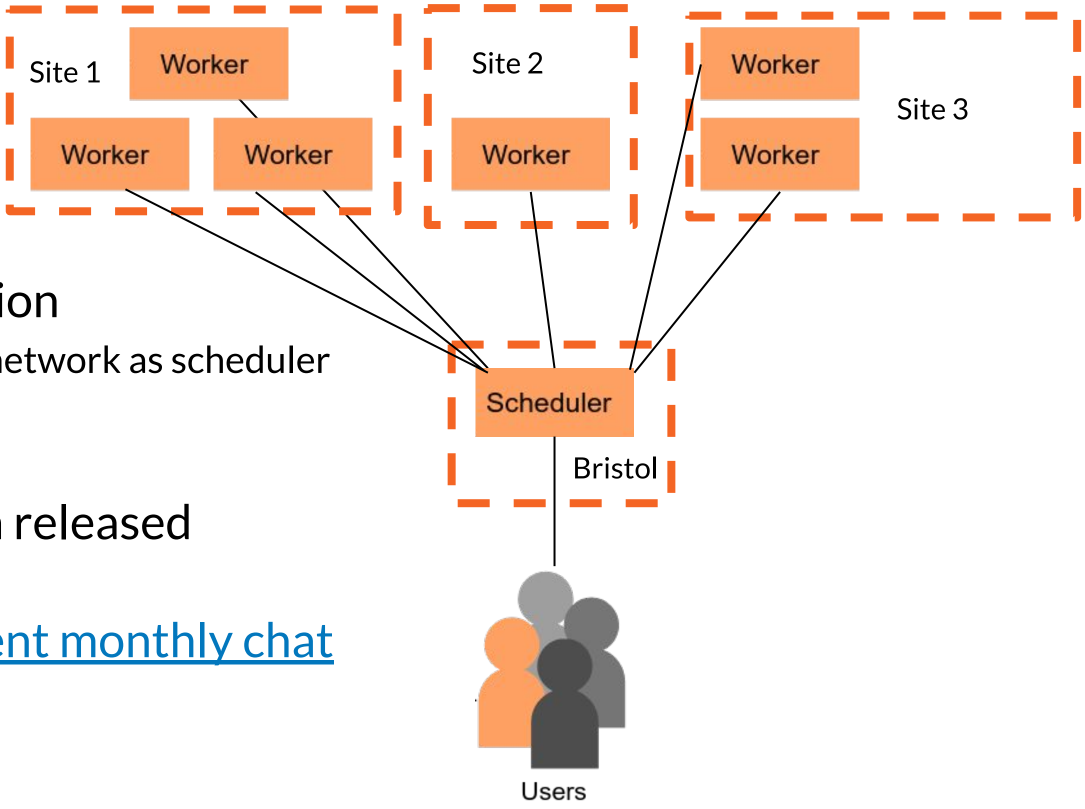
```python
from dask_jobqueue import PBSCluster
cluster = PBSCluster()
cluster.scale(jobs=10)    # Deploy ten single-node jobs

from dask.distributed import Client
client = Client(cluster)  # Connect this local process to remote workers
```

dask_dirac



- Public access required to scheduler
- Setup new VM @ bristol with open port(s)

dask_dirac



- Working version with limitation
  - Workers must be on the same network as scheduler for jobs to run
  -
- This is v 0.1.0 which has been released
  - Github repository
- More details in the most recent monthly chat

https://distributed.dask.org/en/latest/journey.html

dask_dirac

Step 1: Client

Step 2: Arrive in the Scheduler

Step 3: Select a Worker

Step 4: Transmit to the Worker

Step 5: Execute on the Worker

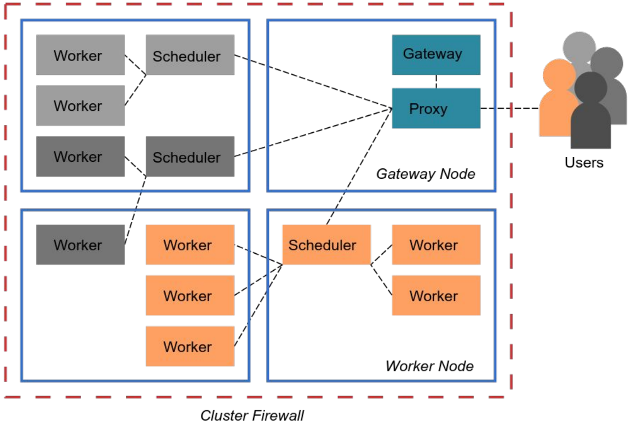
Step 6: Scheduler Aftermath

Step 7: Gather ← Failing to gather

Step 8: Garbage Collection

Issue: Dask worker has no public IP

Looking into dask-gateway



Analysis Grand Challenges

Analysis Grand Challenges

Step 2: Run Analysis Grand Challenges at Brunel via DIRAC

(Github repo)

IRIS-HEP currently provides

- ATLAS H→ZZ
- CMS ttbar

What SWIFT-HEP could provide

- CMS Higgs analysis (Imperial)
- LZ Analysis (Bristol)

To start with analyses need to be able to use Dask.

Later also custom graphs (caching, portability).

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)

→ more info IRIS-HEP AGC Tools workshop, 25th of April 2022

Analysis Grand Challenges (SWIFT-HEP)

In SWIFT-HEP we can copy the main test with little extra effort*

```
elif af == "DIRAC":
    from dask_dirac import DiracCluster
    from dask.distributed import Client

    cluster = DiracCluster(cores=1, memory="2GB", scheduler_options={"port":8786},
                           user_proxy="/tmp/x509up_u397871",
                           cert_path="/users/ak18773/SWIFT_HEP/DIRACOS/diracos/etc/grid-security/certificates")

    cluster.scale(jobs=10)
    print("Please allow up to 10 minutes for the first worker to connect")
    print(f"Cluster dashboard: {str(cluster.dashboard_link)}")

    client = Client(cluster)
```

*by swapping the dask-jobqueue configuration: HTCondor → DIRAC

Analysis Grand Challenges (SWIFT-HEP)

In SWIFT-HEP we can copy the main test with little extra effort*

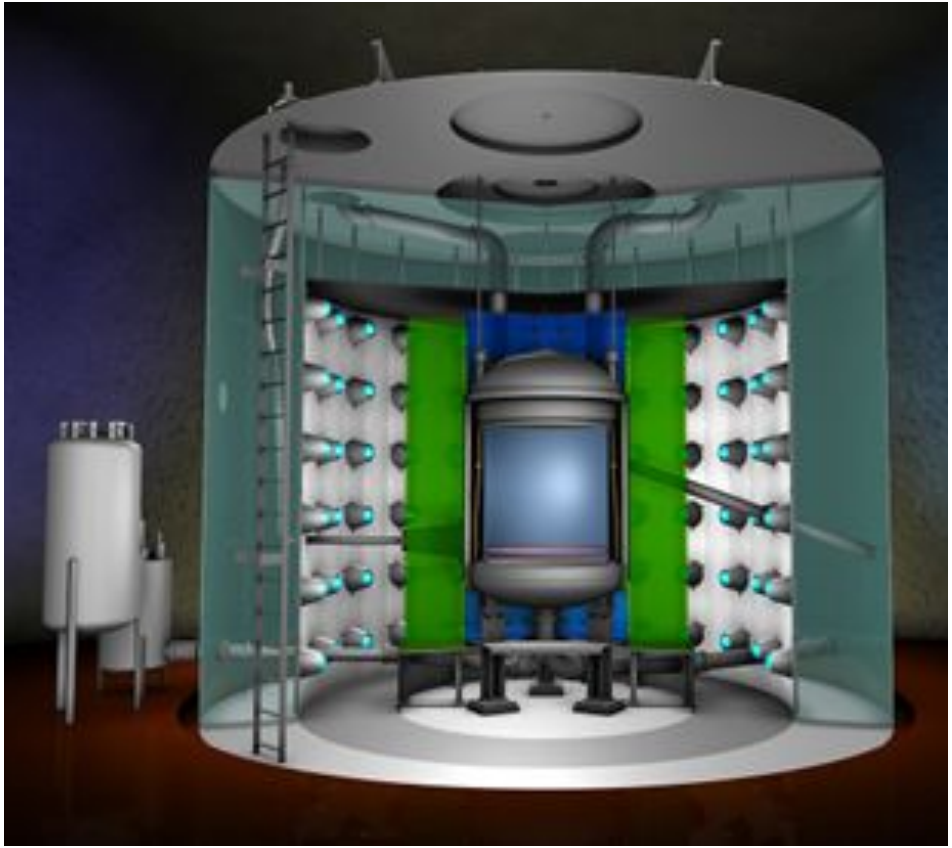
- First test to be run at Brunel
- Will likely need some iterations

What could be a special UK contribution?

- We support many small & medium size experiment → add their examples to the mix?
- I'm also on LZ → could be the first addition

LUX-ZEPLIN

- Direct Dark Matter exp.
- ~250 large collab
- Based at Sanford Underground Research Facility, South Dakota, USA
- (same place as Dune)



LUX-ZEPLIN Analysis

- From 5-years of running, expect ~ 300TB. First run was 30TB
- Most published analysis is single-core C++ looping over events though some recent effort has been made to use python and move away from reliance on single cluster (NERSC)
- Requires multi-tree analysis with isn't currently supported by coffea
- All data + sims are private, but can be persuaded to share some

Summary and Outlook

SWIFT-HEP WP5 has started and things have happened

International scene is testing Dask workflows for ATLAS and CMS analysis

SWIFT-HEP can use existing efforts for testing and contribute non-ATLAS/CMS workflows

First release of dask-dirac interface

Backup

chatGPT on SWIFT-HEP

SWIFT-HEP is a project focused on developing and deploying software tools and infrastructure for high-energy physics (HEP) research on distributed computing platforms, such as HPC, cloud, and Grid computing. The SWIFT-HEP project is a collaboration between researchers from several HEP institutions and computer science departments, including Fermilab, CERN, University of California San Diego, University of Wisconsin, University of Manchester, and others.

The main goal of SWIFT-HEP is to create a unified and flexible framework for HEP data processing, analysis, and simulation on distributed computing resources. The SWIFT-HEP software stack consists of several components, including the SWIFT workflow system, which provides a flexible and user-friendly interface for defining, executing, and managing complex HEP workflows on distributed computing resources. Other components include tools for data management, workflow monitoring, and resource allocation, as well as interfaces to popular HEP software frameworks such as ROOT, Geant4, and Pythia.

SWIFT-HEP is designed to be scalable and adaptable to different computing platforms, allowing HEP researchers to take advantage of the latest computing technologies and architectures. The project is also focused on developing best practices and standards for distributed HEP computing, with the goal of making it easier for researchers to collaborate and share data and resources across different institutions and projects.

Overall, SWIFT-HEP is an important initiative in the HEP community, helping to improve the efficiency and effectiveness of data processing and analysis for cutting-edge experiments such as the Large Hadron Collider (LHC) at CERN.

Analysis key points

Physics

Last mile of long chain of data recording and processing.

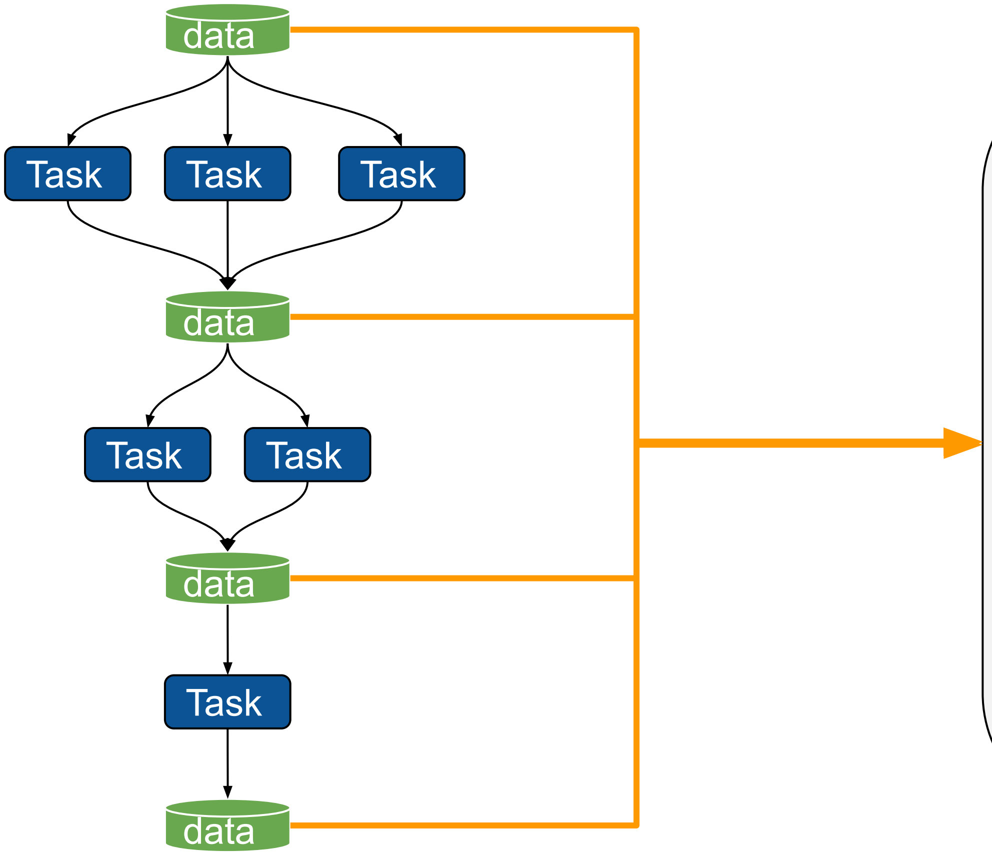
Goals: gain insight and create new knowledge

Computing

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

Flexibility is paramount.

Anatomy of an analysis workflow

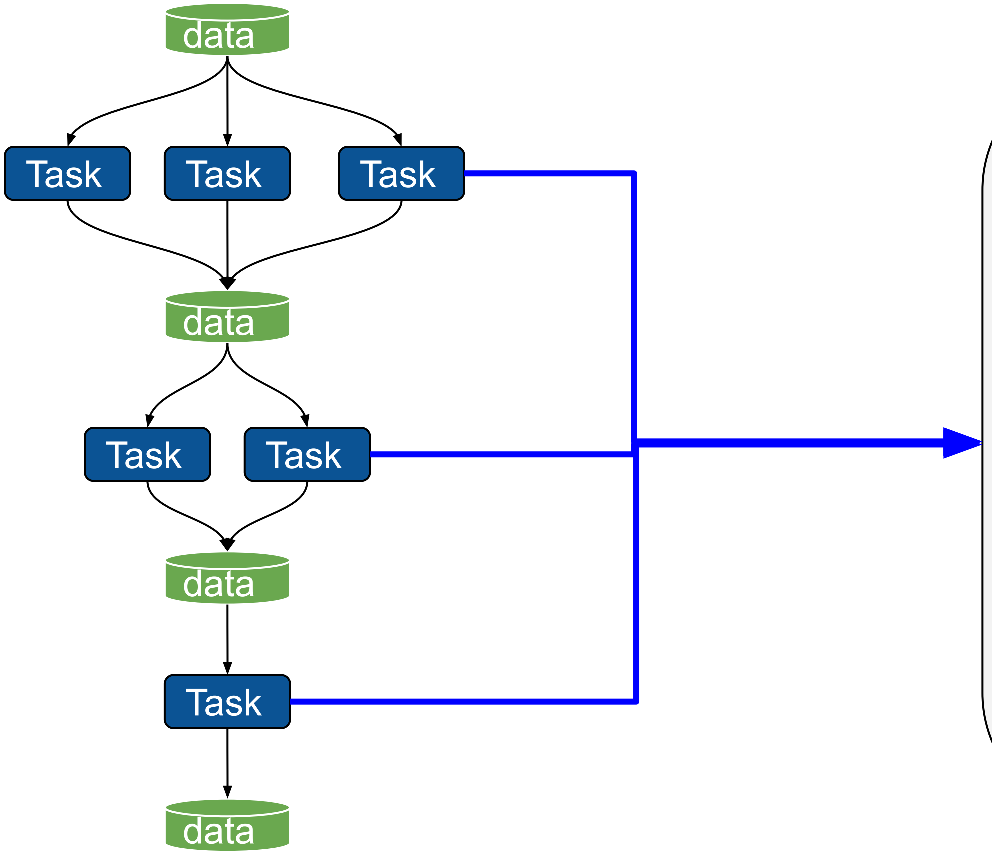


Data Formats

ROOT files: standard for input/output
Internally: Experiment/analysis specific

HDF5	npz
pandas	datacard
parquet	other?

Anatomy of an analysis workflow



Processing

- Event loop vs vectorized processing
- Monoliths vs compute graphs
- GPU/FPGA capable vs strictly CPU
- Parallelizable vs strictly sequential
- Failure tolerance vs all or nothing
- Time sensitive vs “sometime next week”

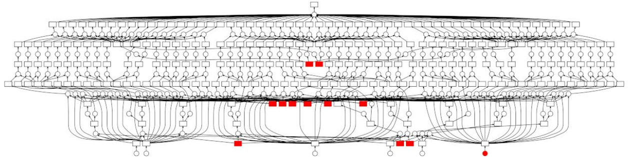
- Varied resource requirements/efficiency

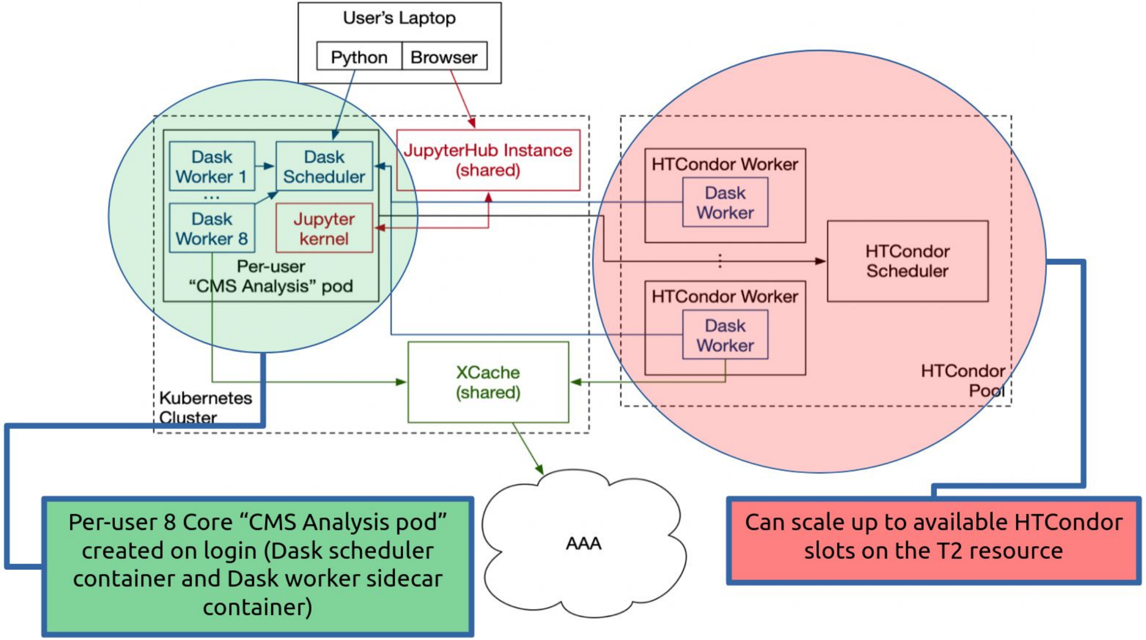
WP1 ↔ WP5

(in practical terms)

Scheduling with coffea-casa

Uses Dask and [dask-jobqueue]



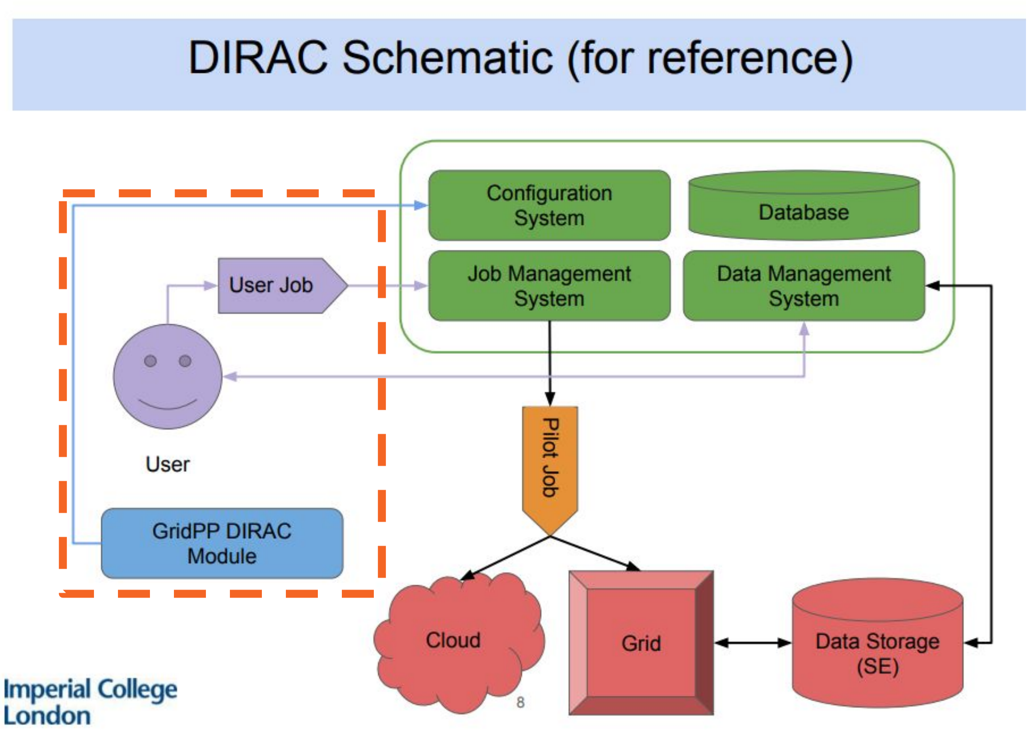


From [coffea-casa docs]

Scheduling with DIRAC

In a nutshell: scheduling across job management systems

Data management system for access to data lake (here caching)



Slide stolen from Janusz's presentation at the SWIFT-HEP May meeting

Concrete [starting] work items (1)

DiracJob and DiracJobQueueCluster in dask-jobqueue*

- Can use DIRAC command-line tools or python library
- In collaboration with DIRAC experts
  - Sensible defaults
  - Best way to communicate extra requirements (e.g. GPU, cached data)

Work here can then easily be migrated to Parsl and/or tested via joblib by volunteers

*we can start as an independent package and merge later

Concrete [starting] work items (2)

Storing (temporary) analysis cache on data lake

- Expiration dates: what is maximally reasonable? What makes sense on average?
- Permissions: users work in (dynamic) groups - What is the best approach for ACLs?
- Xrootd cache: Does it make sense to pre-fill input data based on scheduled DIRAC job?

Analysis Grand Challenges

Analysis Grand Challenges (SWIFT-HEP)

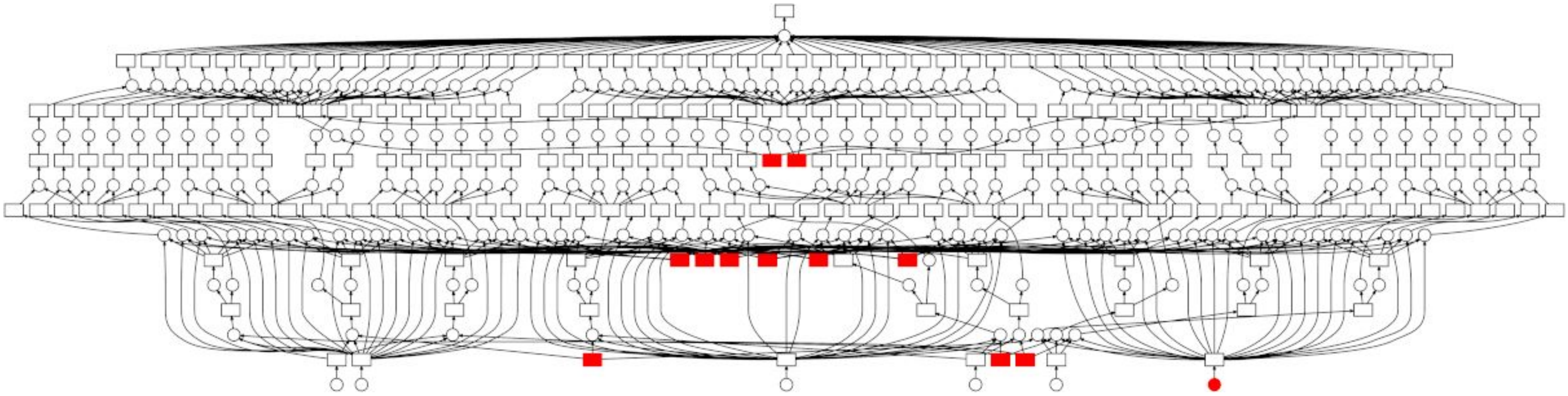
In SWIFT-HEP we can copy the main test with little extra effort*

But: can we involve analysis groups in the UK?

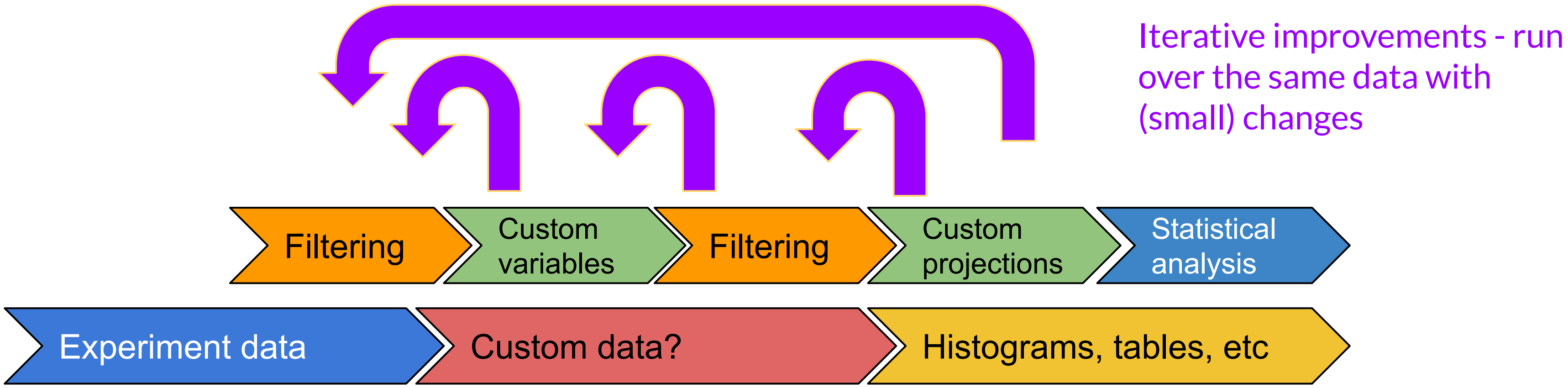
- Would need to provide documentation on the use of DiracJobQueue
- Need to allocate resources per group
- Need to make sure job wrappers and Analysis Facility monitoring capture all metrics (i.e. no additional work for users here)

*by swapping the dask-jobqueue configuration: HTCondor → DIRAC

Analysis workflow example in Dask



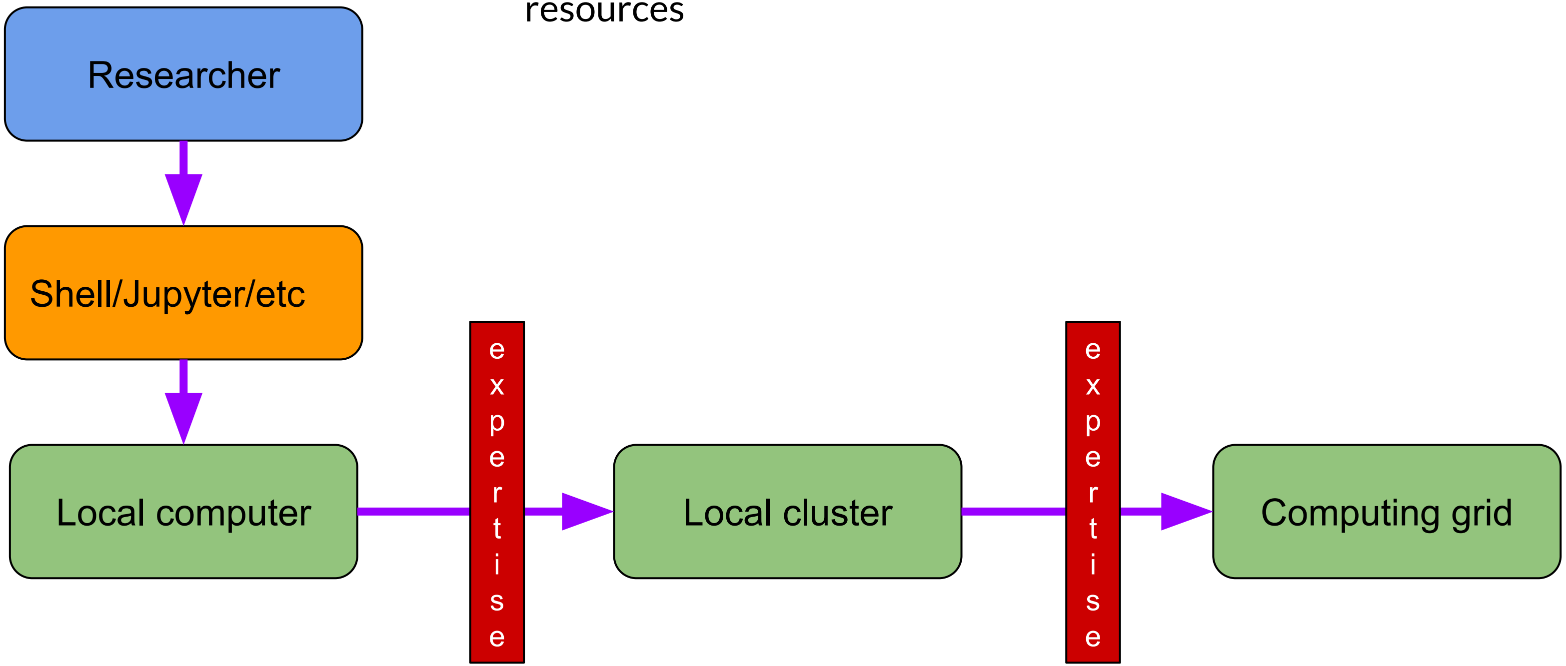
Analysis pipeline example reality might differ



- Custom variables **might** include Machine Learning → training and inference on GPU
- **Depending** on underlying tools, statistical analysis can benefit from GPUs as well
- **Depending** on expertise, analysis code might be modular or one big block
- **Depending** on expertise each iteration will use resources efficiently, **or not**

Analysis Workflow: compute

As analysis needs increase, new expertise is needed to use more resources



Analysis Challenge

Large user-driven component → hard to optimize for every case

Inconsistent data use: new data sets, reprocessing of targeted data sets

Ideally, each iteration is as short as possible → "time to insight" low

iterative model == waste of computing resources?

Emerging trend: **interactive analysis**

Jupyter notebooks

Analysis “simplified”

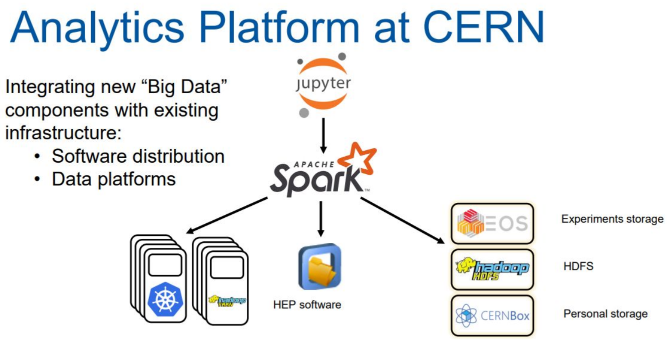
These kinds of workflows seem really desirable by the current generation of PhD students

Shifts a lot of “How to do distributed computing” to “What I want to get done” → declarative approaches are great for research

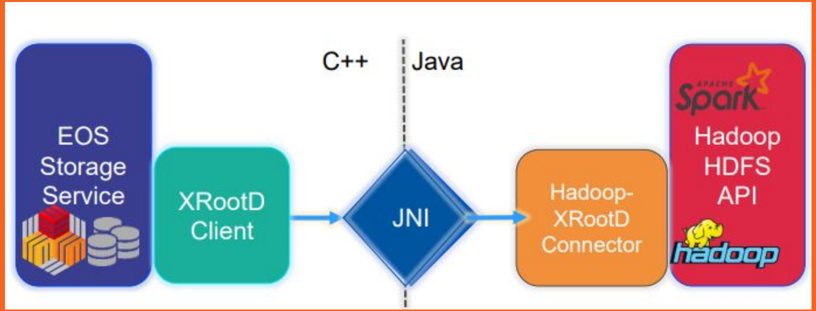
This disconnection allows experts to improve computing infrastructure “behind the scenes”

CERN's analytix cluster

Spark + Hadoop (link)

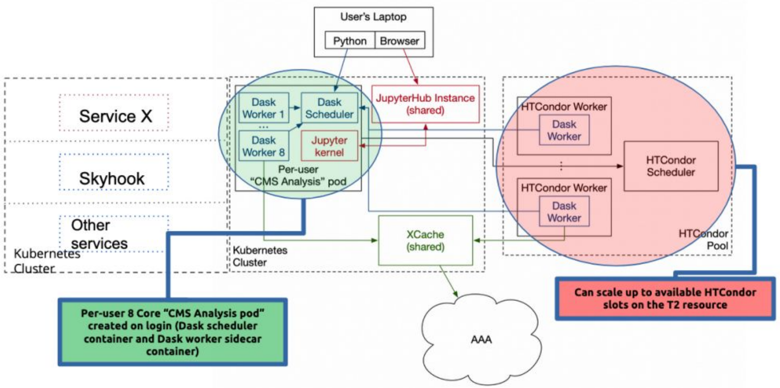


- Initially for log processing on Hadoop
- Can run ROOT analysis on Spark
- Accessible via **CERN's SWAN service (Jupyter)**
- Access to external storage via plugin



IRIS-HEP Coffea-casa

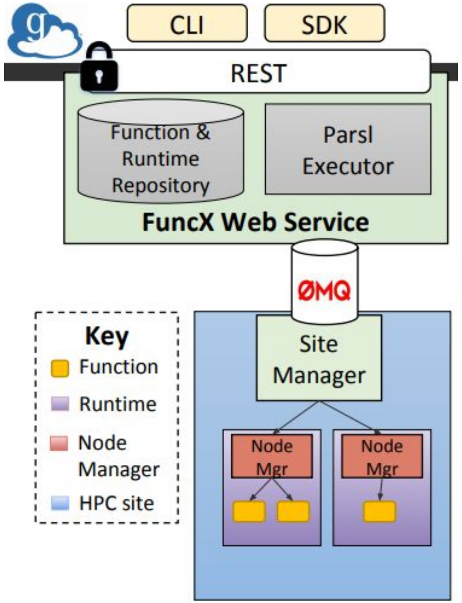
Analysis facility on top of an HTCondor cluster (link)



- Dask as a key component
- Uses TLS proxy (Traefik) to route requests from outside to the Dask cluster
- Dask-jobqueue for submitting to batch system (e.g HTCondor)
- More details in next talk

funcX

Federated function as a service (link)



- “Serverless” approach to compute (similar to FnProject)
- Reduces barriers to access distributed resources
- Low-latency, on-demand
- Can be used to build a catalogue of functions
- Functions can be deployed on special resources → ”binding algorithms to hardware”

Hyper (Lux-Zeplin)

non-LHC analysis via Dask on HPC and HTC (see talk)

“Hyper is an uproot wrapper that lets you execute any Python code easily in parallel”

- Dask as a key component
- Dask-jobqueue for submitting to batch system
- Uses boost_histogram, uproot, numexpr & more
- Tested on a UK cluster and at NERSC
- Example for interactive distributed analysis without a dedicated analysis facility

IRIS-HEP

Analysis Grand Challenges
[AGCs]
(incl. ATLAS, CMS and WLCG)

Related IRIS-HEP workshop

Multiple challenges in the years 2022, 2023, 2025, 2027

Analysis: Demonstrate analysis system can cope with increased data volume while delivering enhanced functionality**

Data volume: realistically sized HL-LHC end-user analysis dataset (~ 200 TB)

Reproducibility and Reinterpretation

Interested in getting more experiments involved to broaden usability

SWIFT-HEP WP 5 in a nutshell