University of BRISTOL

# SWIFT-HEP
# WP5 Analysis Systems

**Sam Eriksen**

10th April 2025

# Overview

- WP5 (Analysis Systems)
  - overview + roadmap
  - Current progress in WP5
  - Future for WP5

**Previous updates**

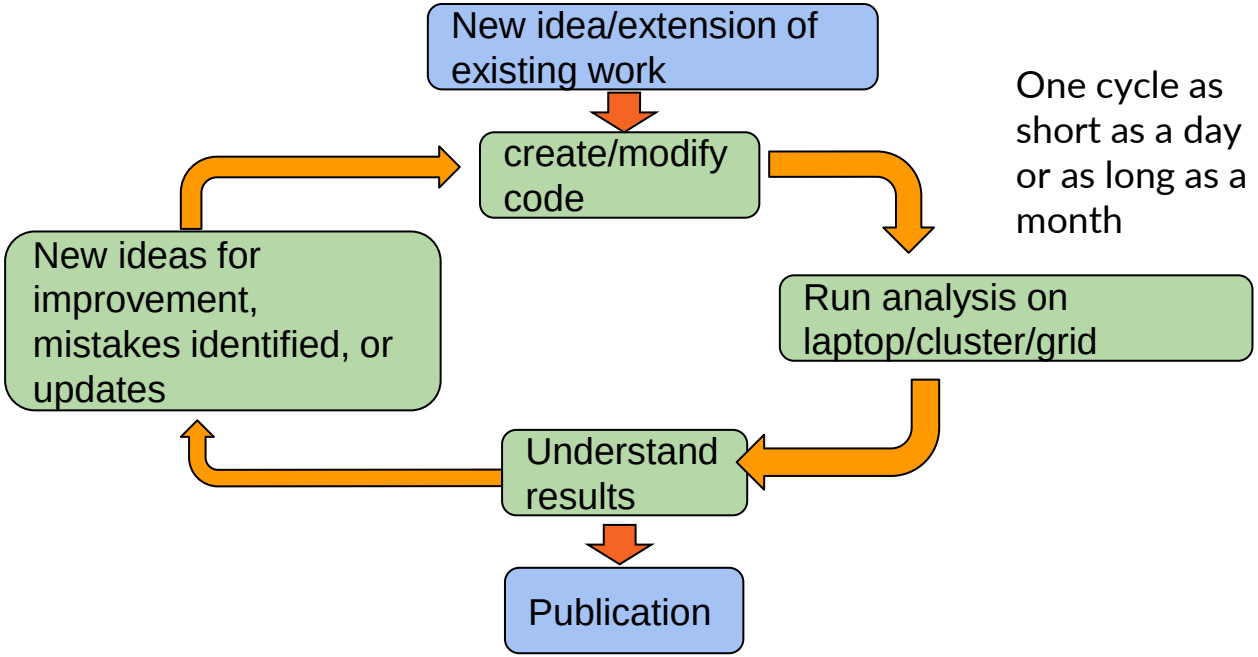
- November 2024
- March 2024
- November 2023
- September 2023
- May 2023
- March 2023
- February 2023

# WP5: Analysis Systems

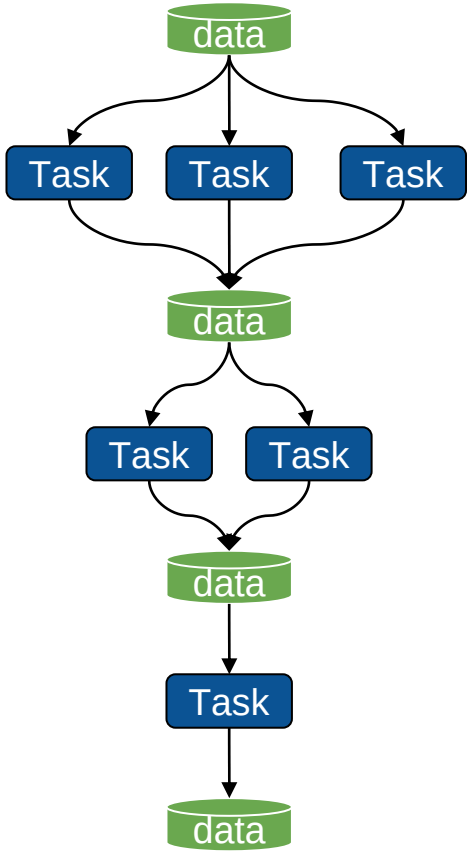
# WP5: Analysis Systems

## Run analysis workloads optimally on distributed resources

# Analysis Anatomy

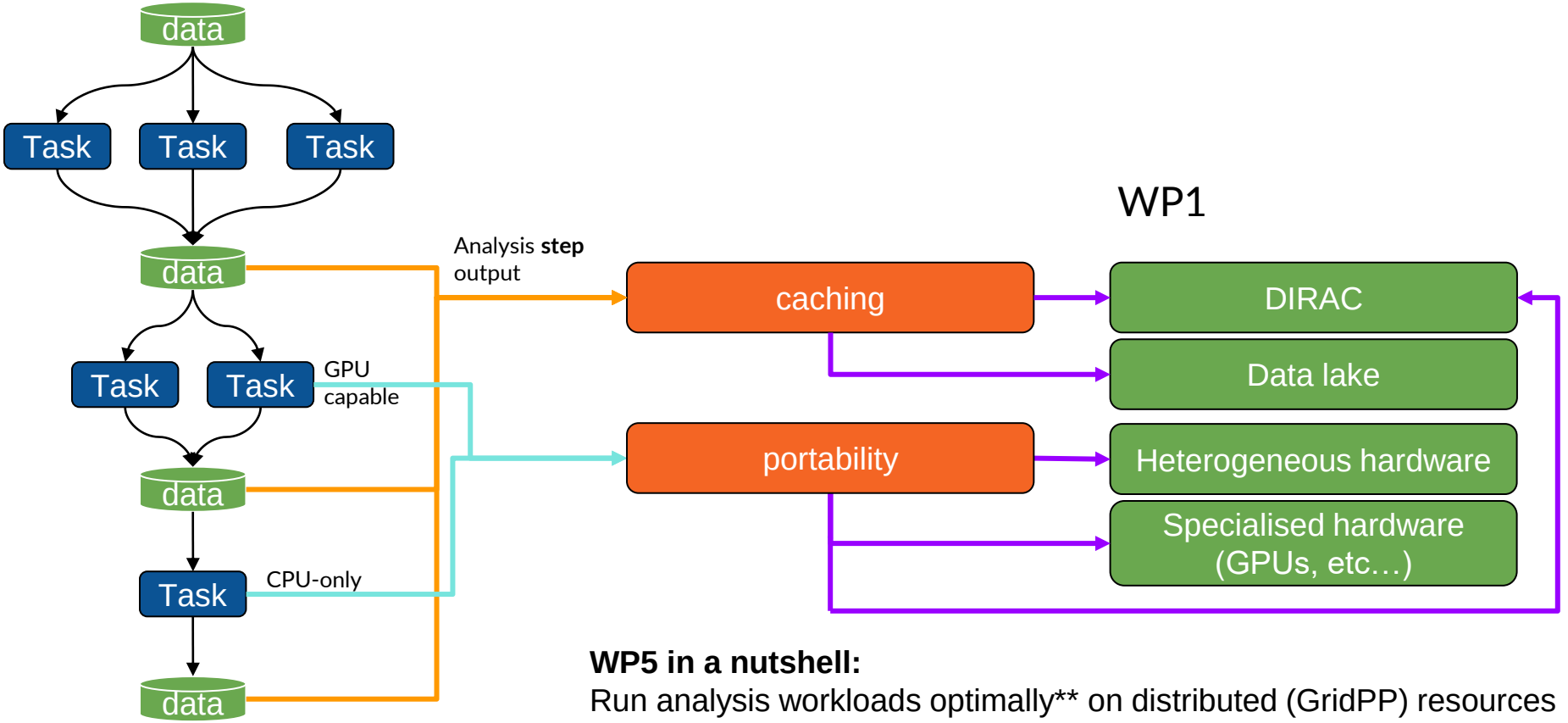



One cycle as short as a day or as long as a month




See talk by Luke Kreczko for some **BIG** Picture

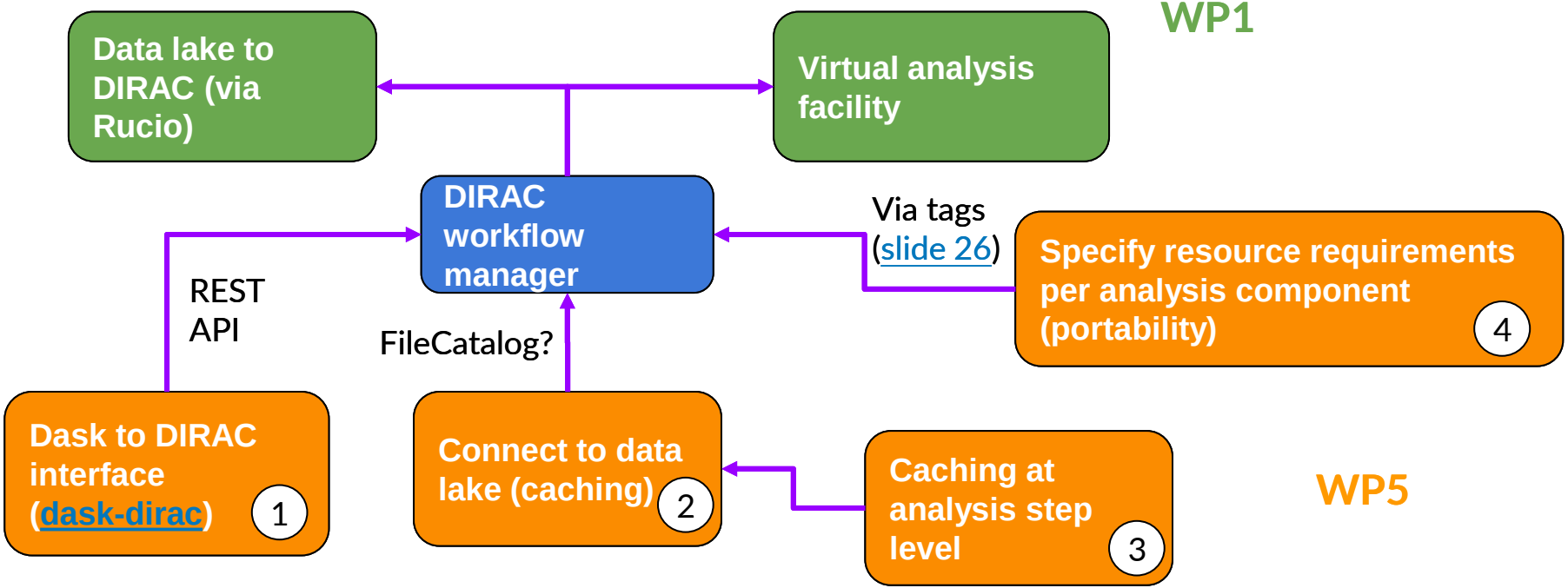
# Analysis Anatomy




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

** balanced between user-experience and computing efficiency

# WP5: Roadmap




More detailed slides

# What have we implimented

# A new cluster: DiracCluster

```
from dask.distributed import LocalCluster, Client
cluster = LocalCluster()
cluster.scale(n_jobs=5) # runs dask worker
client = Client(cluster)
```

LocalCluster
- Scheduler is local
- Workers are local

```
from dask.distributed import SlurmCluster, Client
cluster = SlurmCluster()
cluster.scale(n_jobs=5) # runs srun
client = Client(cluster)
```

SlurmCluster
- Scheduler is local
- Workers are slurm jobs

```
from dask.distributed import PBCluster, Client
cluster = PBCluster()
cluster.scale(n_jobs=5)
client = Client(cluster)
```

PBCluster
- Scheduler is local
- Workers run slurm jobs

```
from dask_dirac import DiracCluster
from dask.distributed import Client
cluster = DiracCluster()
cluster.scale(n_jobs=5) # runs dask-dirac submit
client = Client(cluster)
```

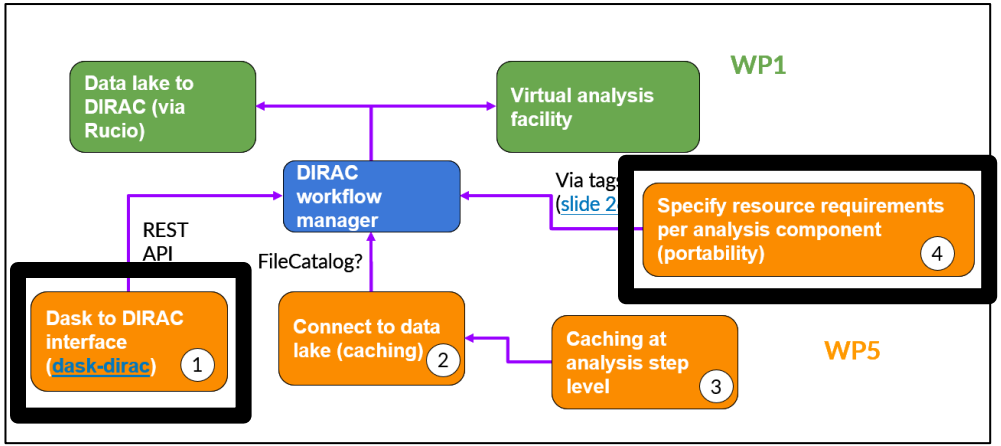
DiracCluster
- Scheduler is local
- Workers are DIRAC

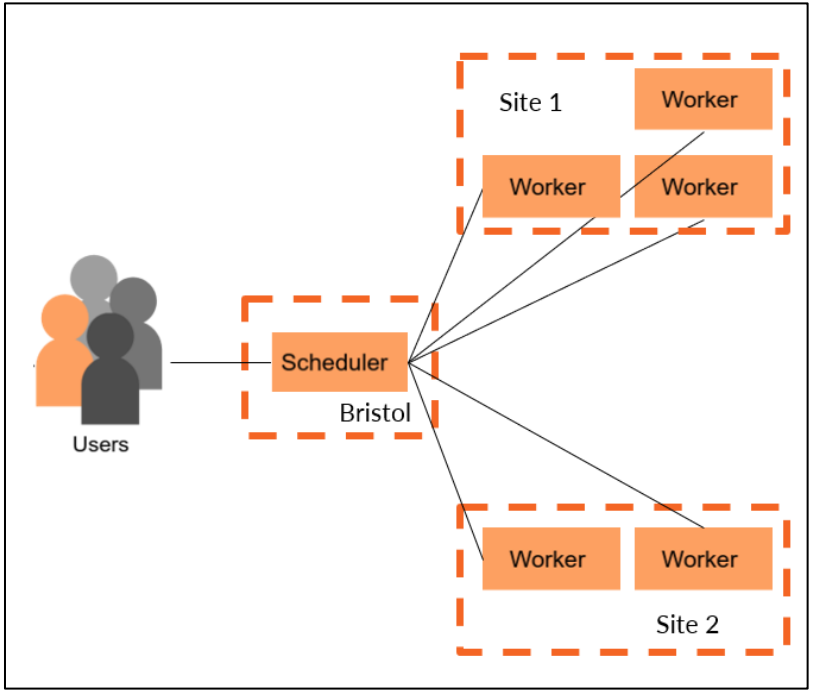
- Requirement;
  - that DIRAC site has an open port
  - local has port open (configurable)

# DiracCluster

```
from dask_dirac import DiracCluster
from dask.distributed import Client
cluster = DiracCluster(sites=[site_list],
                       gpus=<bool>, nthreads=<int>)
cluster.scale(n_jobs=5)
client = Client(cluster)
```

- Can split across multiple sites
  - Requires open port for worker
- Able to require GPU for job
  - More on this later
- Able to select the number of threads (default is all)



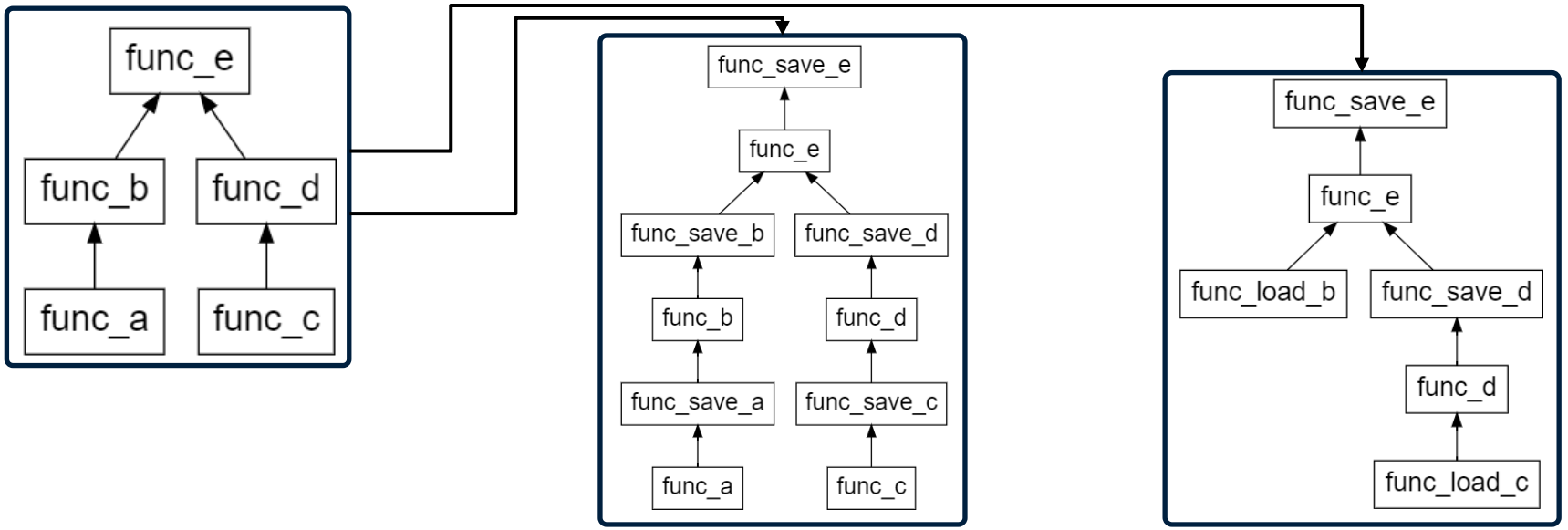
# A new client: DiracClient

```
from dask.distributed import Client
client = Client(cluster)
```

Client
- Job (task graph) split and passed to scheduler

```
from dask_dirac import DiracClient
client = DiracClient(cluster)
```

DiracClient
- Intercepts task graph allowing for caching checks

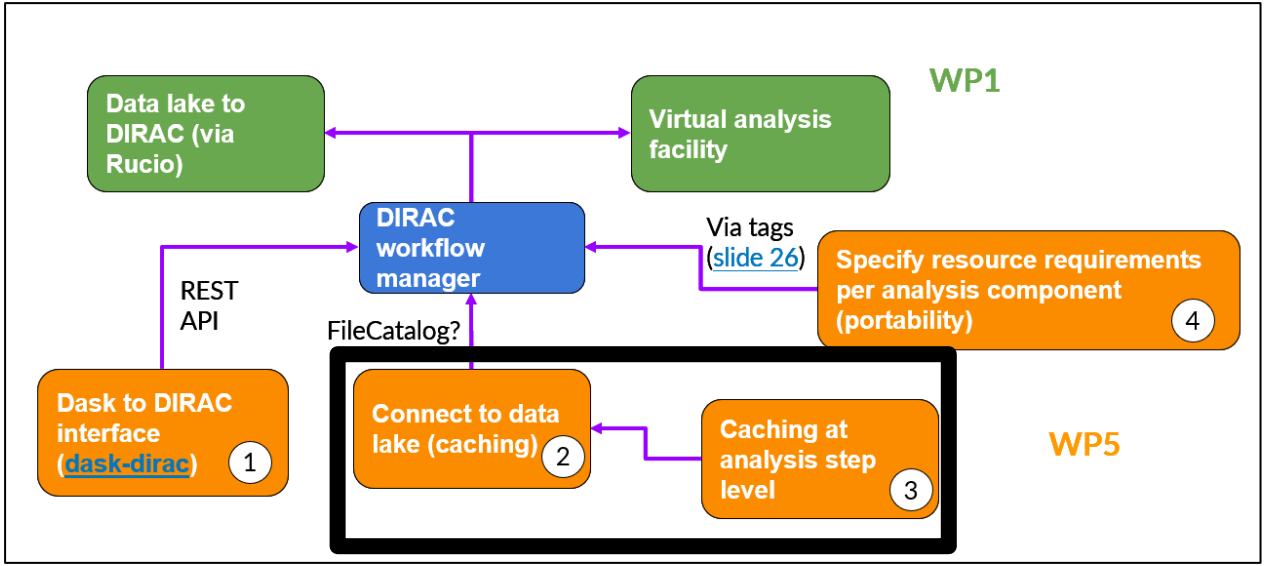


# DiracClient

```
from dask_dirac import DiracClient
client = DiracClient(cluster,
                     cache_location=<path>)
```

Cache supported:
- local
- Dirac storage element

Working towards:
- RUCIO

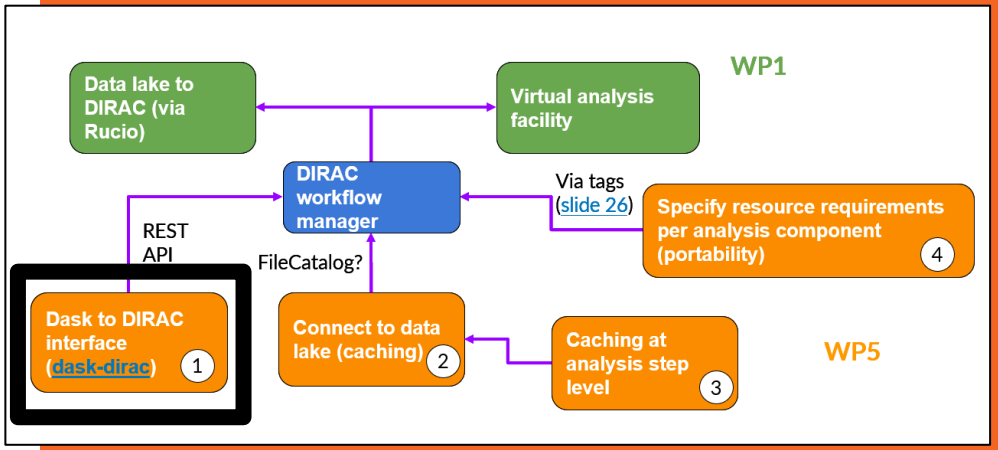


# Practical Examples

# Practical Example 1

# CMS AGC

- Demonstration of using dask-dirac



# Practical Example 1: CMS AGC

**IRIS-HEP Analysis Grand Challenge**
**CMS $t\bar{t}$ using 2015 Open Data**
**doi: 10.22323/1.414.0235**

The AGC is about performing the last step in an analysis pipeline at **scale** to test workflows envisioned for the HL-LHC

Why CMS $t\bar{t}$ ?
- Is a realistic workflow
- Up to ~2TB (~1B events)
- Uses **dask**

For this example, I'm using;
- AGC branch: se_daskdirac
- dask-dirac 2025.3.1
- submission is to **diracdev**
- Using 42.8GB (~20M events)

In order to use dask-dirac, what edits were made?
- Added the **DiracCluster** as an option
- Set af = dirac when running

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

    cluster = DiracCluster(
                    scheduler_options={"port": 8786},
                    dirac_sites="LCG.UKI-SOUTHGRID-RALPP.uk",
                    cert_path="/users/ak18773/SWIFT_HEP/dev_dirac/diracos/etc/grid-security/certificates",
                    owner_group="gridpp_user",
                    user_proxy="/tmp/x509up_u397871",
                    submission_url="https://diracdev.grid.hep.ph.ic.ac.uk:8444",
                    container="docker://sameriksen/dask:cms_agc",
                    nthreads=5,
                )

    cluster.scale(jobs=1)

    print("Workers can take a while to connect. Please be patient.")
    print(f"Cluster dashboard: {str(cluster.dashboard_link)}")

    client = Client(cluster)
```

# Practical Example 1: Step 1

**Get a grid proxy**

```
sam@distributed:~/SWIFT_HEP/dirac_ui/diracos$ pwd
/users/ak18773/SWIFT_HEP/dirac_ui/diracos
sam@distributed:~/SWIFT_HEP/dirac_ui/diracos$ source diracosrc
realpath: '': No such file or directory
realpath: '': No such file or directory
realpath: '': No such file or directory
(base) sam@distributed:~/SWIFT_HEP/dirac_ui/diracos$ dirac-proxy-init -d gridpp_user
Generating proxy...
```

- Exactly the same of any other grid job
- Follows instructions from dirac docs with one difference
- We configure for the development server

```
dirac-configure -F -S GridPP -C dips://diracdev.grid.hep.ph.ic.ac.uk:9135/Configuration/Server -I
```

# Practical Example 1: Step 2

**Run the analysis notebook as normal**

- Analysis is a jupyter notebook
- Set: af = dirac
- Set: number of files

- When we get to defining the executor, this is where dask-dirac is used

```
NanoAODSchema.warn_missing_crossrefs = False # silences warnings about branches we will not use here
if USE_DASK:
    cloudpickle.register_pickle_by_value(utils) # serialize methods and objects in utils so that they can be accessed withi
    executor = processor.DaskExecutor(client=utils.clients.get_client(af=utils.config["global"]["AF"]))
else:
    executor = processor.FuturesExecutor(workers=utils.config["benchmarking"]["NUM_CORES"])
```

- In this example, 1 job submitted (see slide 15); visible on https://diracdev.grid.hep.ph.ic.ac.uk:8443/DIRAC

| | JobId ↓ | | Status | MinorStatus | Application... | Site | JobName | LastUpdate[UTC] |
|---|---|---|---|---|---|---|---|---|
| ☐ | 1599 | | Waiting | Pilot Agent Sub... | Unknown | LCG.UKI-S... | dask_dirac: das... | 2025-03-31 13:45:02 |

# Practical Example 1: Step 2

**Run the analysis notebook as normal**

- And once the job starts, workers connect and the analysis begins

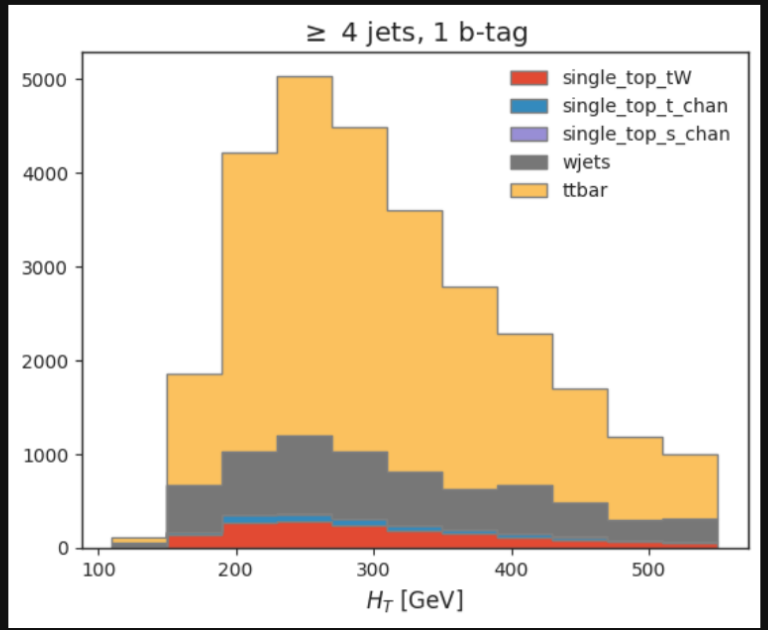
```
# processing
all_histograms, metrics = run(
    fileset,
    treename,
    processor_instance=TtbarAnalysis(USE_INFERENCE, USE_TRITON)
)
```

# Practical Example 1: Step 2

**Run the analysis notebook as normal**

- Once processing if finished, the results are retried from the workers and we can plot the result

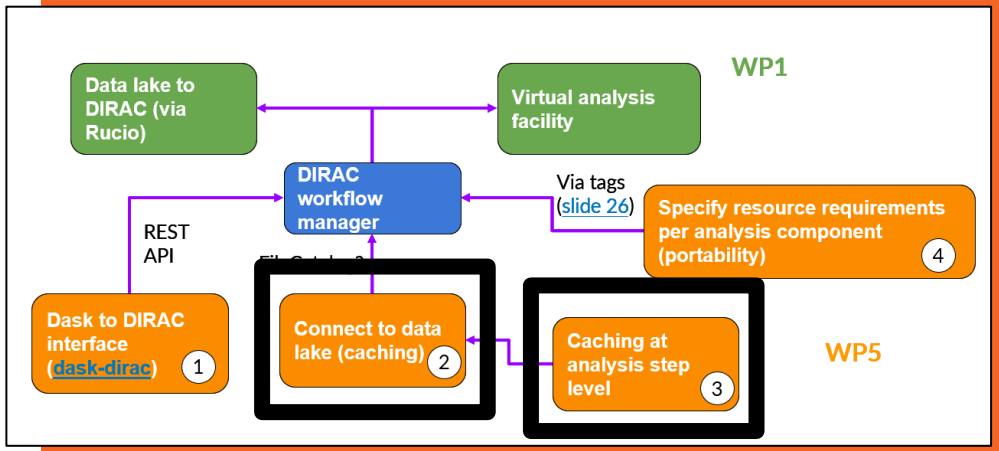
```
all_histograms["hist_dict"]["4j1b"][120j::hist.rebin(2), :, "nominal"].stack("process")[::-1].plot(stack=True, histtype="fi
plt.legend(frameon=False)
plt.title("$\geq$ 4 jets, 1 b-tag")
plt.xlabel("$H_T$ [GeV]");
```



# Practical Example 2

# DiracClient

- Demonstration of using caching



# Practical Example 2: DiracClient

**dask-dirac**
**April2025_workshop.ipynb**

This example is to show caching in action on a simple example.

What is the example?
- LocalCuster (workers are local)
- DiracClient
- Local caching

Where is this example?
- April2025_workshop.ipynb
- dask-dirac 2025.3.1

In order to use the DiracClient, what edits were made?
- Just from dask.distributed.Client to dask_dirac.DiracClient
- Set a cache_location (dirac://, file:///, rucio://, etc…)

```
from dask_dirac import DiracClient, DiracCluster
from dask.distributed import LocalCluster, Client
import dask.array as da

cluster = LocalCluster(n_workers=1)

client = DiracClient(cluster,
                     cache_location="file:///tmp/dask-cache_05022025")
# client = Client(cluster)
```

# Practical Example 2: DiracClient

Local Cluster; workers are on same machine

Only change from conventional approach
Using DiracClient rather than Client

Nothing in the cache location

```
from dask_dirac import DiracClient, DiracCluster
from dask.distributed import LocalCluster, Client
import dask.array as da
```

✓ 0.0s

```
# Task structure has changed in newer versions, so for this using 2024.5.0
import dask
dask.__version__
```

✓ 0.0s

```
'2024.5.0'
```

```
cluster = LocalCluster(n_workers=1)
```

✓ 1.1s

```
client = DiracClient(cluster,
                     cache_location="file:///tmp/dask-cache_05022025")
#client = Client(cluster)
```

✓ 0.0s

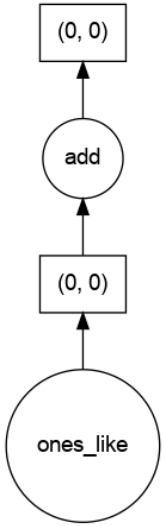
```
# Check the cache location and show what files are there
print(client.cache_location)
!ls {client.cache_location[7:]} # remove file:// at the beginning
```

✓ 0.2s

```
file:///tmp/dask-cache_05022025
```

# Practical Example 2: DiracClient

Processing is adding a number to an array.

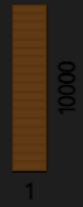



Behind the scenes, task graph has been altered so the two stages are cached

```
# Create a Dask DataFrame directly
dask_array = da.ones((1e4, 1), chunks=(1)) + 20231
dask_array
```

✓ 0.0s

| | Array | Chunk |
|---|---|---|
| Bytes | 78.12 kiB | 8 B |
| Shape | (10000, 1) | (1, 1) |
| Dask graph | 10000 chunks in 2 graph layers | |
| Data type | float64 numpy.ndarray | |




```
result = client.compute(dask_array)
```

✓ 8.5s

```
r = result.result()
```

✓ 19.5s

```
# Check the cache location and show what files are there
print(client.cache_location)
!ls {client.cache_location[7:]} # remove file:// at the beginning
```

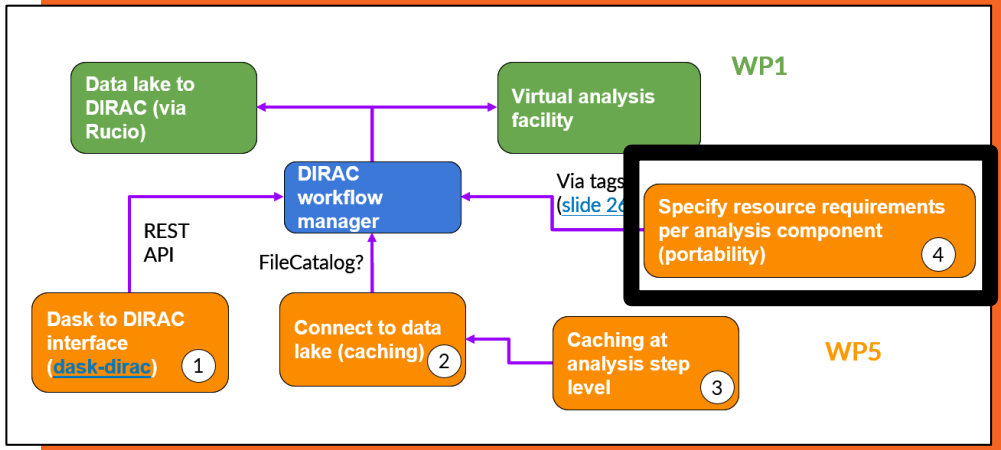
✓ 0.2s

```
file:///tmp/dask-cache_05022025
79ff19291795877767dc8ff7ac8dec3ae6ad1ddd52e93e1c088bc90f24451fe5ecbaec4bbe9a055ab35ab4b70431e16f.parquet
7d41dc42b2d1236a9e67e28103d1fb3bdc6fc748920a8f971a7abe6c9304ff7477f3e402e50face5c1b40c694eee0402.parquet
```

# Practical Example 3

# LZ Analysis

- Demonstration of interchangeability of CPU and GPU in analysis



# Practical Example 3: LZ AGC with GPUs

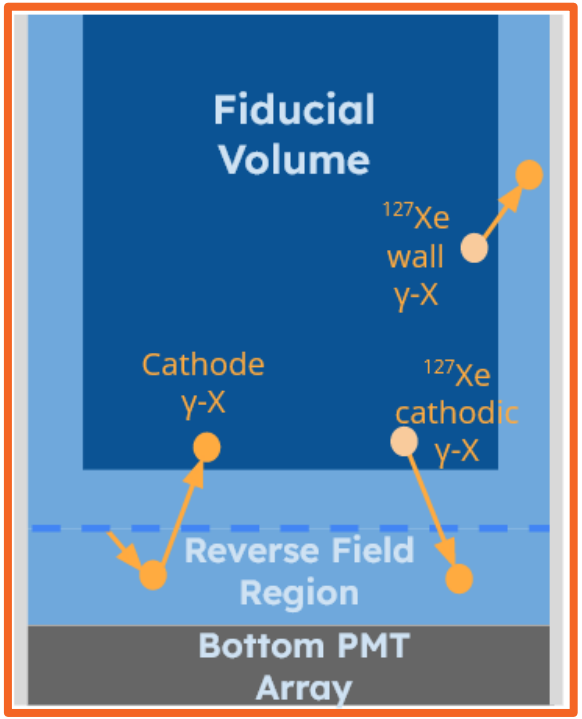
**dask-dirac April2025_workshop.ipynb**

What is the example?
- LZ analysis on simulated data
- Local Cluster
- 6-files, ~1-5 GB each
- ~4 mil events
- In production, it's closer to ~100 TB

Where is this example?
- April2025_workshop.ipynb
- dask-dirac 2025.3.1

What is this example about?
- Now we can access grid-GPUs, can we easily integrate into workflows?
- Offering an LZ analysis

**Context:** One background in LZ are γ-X events, these can look like dark matter, and originate where not all charge is collected. This analysis looks at simulations and determines what rate of these we should expect.



# Practical Example 3: LZ AGC with GPUs

In the core of the analysis; CPU version

Open a file

Make some arrays

Some processing of arrays

```
Tabnine | Edit | Test | Explain | Document
def process_file(file):
    component = file.split('/SS_skim_')[1][:-5]

    tfile = up.open(file)
    scatters = tfile['Scatters']
    truth = tfile['RQMCTruth']

    # Read arrays
    ss_x = scatters['ss.x_cm'].array()
    ss_y = scatters['ss.y_cm'].array()
    ss_dt = scatters['ss.driftTime_ns'].array()
    ss_s1c = scatters['ss.correctedS1Area_phd'].array()
    ss_s2 = scatters['ss.s2Area_phd'].array()
    ss_s2c = scatters['ss.correctedS2Area_phd'].array()

    mc_detS1 = truth['mcTruthVertices.detectedS1Photons'].array()
    mc_detS2 = truth['mcTruthVertices.detectedS2Photons'].array()

    n_events = len(ss_x)

    result = process_events(ss_x, ss_y, ss_dt, ss_s1c, ss_s2, ss_s2c, mc_detS1, mc_detS2)
    n_fv, n_roi, n_fv_roi, n_mssi, n_fv_mssi, n_roi_mssi, n_fv_roi_mssi = result

    event_weight = 1e-6 / 200

    return component, n_events, n_fv, n_roi, n_fv_roi, n_mssi, n_fv_mssi, n_roi_mssi, n_fv_roi_mssi, event_weight
```

# Practical Example 3: LZ AGC with GPUs

Loop over entries in array

Evaluate a bunch of cuts

Count events which pass cuts

```
Tabnine | Edit | Test | Explain | Document
@numba.jit
def process_events(ss_x, ss_y, ss_dt, ss_s1c, ss_s2, ss_s2c, mc_detS1, mc_detS2):
    n_fv, n_roi, n_fv_roi = 0, 0, 0
    n_mssi, n_fv_mssi, n_roi_mssi, n_fv_roi_mssi = 0, 0, 0, 0

    # Loop over events
    for i in range(len(ss_x)):

        # Evaluate cuts
        cut_bool_resistor_fv = fv_resistor(ss_x[i], ss_y[i])
        cut_bool_z = fv_z(ss_dt[i] / 1000)
        cut_bool_r = fv_phi_r(ss_x[i], ss_y[i], ss_dt[i] / 1000)
        fv_cut = cut_bool_resistor_fv and cut_bool_z and cut_bool_r
        roi_cut = roi(ss_s1c[i], ss_s2[i], ss_s2c[i])

        # Examine if MSSI
        nS1, nS2 = 0, 0
        for j in range(len(mc_detS1[i])):
            if mc_detS1[i][j] > 0.:
                nS1 += 1
            if mc_detS2[i][j] > 0.:
                nS2 += 1
        is_mssi = nS1 > nS2

        # Now evaluate cuts and count events
        if is_mssi:
            n_mssi += 1
        if fv_cut:
            n_fv += 1
            if is_mssi:
                n_fv_mssi += 1
            if roi_cut:
                n_fv_roi += 1
                if is_mssi:
                    n_fv_roi_mssi += 1
        if roi_cut:
            n_roi += 1
            if is_mssi:
                n_roi_mssi += 1

    return n_fv, n_roi, n_fv_roi, n_mssi, n_fv_mssi, n_roi_mssi, n_fv_roi_mssi
```

# Practical Example 3: LZ AGC with GPUs

How does CPU and GPU approaches differ?

For most of the cuts, no change is needed

```
"""
FV: Resistor cutout
"""
Tabnine | Edit | Test | Explain | Document
@numba.njit
def fv_resistor(x, y):
    res1X, res1Y, res1R = -69.8, 3.5, 6
    res2X, res2Y, res2R = -67.5, -14.3, 6

    insideRes1 = (x-res1X)*(x-res1X) + (y-res1Y)*(y-res1Y) > res1R**2
    insideRes2 = (x-res2X)*(x-res2X) + (y-res2Y)*(y-res2Y) > res2R**2

    return (insideRes1 & insideRes2)

Tabnine | Edit | Test | Explain | Document
@cuda.jit
def fv_resistor_cuda(x, y):
    res1X, res1Y, res1R = -69.8, 3.5, 6
    res2X, res2Y, res2R = -67.5, -14.3, 6

    insideRes1 = ((x - res1X) ** 2 + (y - res1Y) ** 2) > res1R**2
    insideRes2 = ((x - res2X) ** 2 + (y - res2Y) ** 2) > res2R**2

    return insideRes1 and insideRes2
```

```
"""
FV: Z
"""
Tabnine | Edit | Test | Explain | Document
@numba.njit
def fv_z(dt):
    return (dt > 71) & (dt < 1030)

Tabnine | Edit | Test | Explain | Document
@cuda.jit
def fv_z_cuda(dt):
    return (dt > 71) and (dt < 1030)

"""
ROI: S1 and S2
"""
Tabnine | Edit | Test | Explain | Document
@numba.njit
def roi(s1c, s2, s2c):
    return (s1c > 3.) and (s1c < 80.) and (s2 > 14.5 * 44.5) and (s2c < 10 ** 4.5)


Tabnine | Edit | Test | Explain | Document
@cuda.jit
def roi_cuda(s1c, s2, s2c):
    return (s1c > 3.) and (s1c < 80.) and (s2 > 14.5 * 44.5) and (s2c < 10 ** 4.5)
```

# Practical Example 3: LZ AGC with GPUs

How does CPU and GPU approaches differ?

copy to GPU

```
coeffs = np.array([[-1.78880746e-13, 4.91268301e-10, -4.96134607e-07, 2.26430932e-04, -4.71792008e-02, 7.33811298e+01],
           [-1.72264463e-13, 4.59149636e-10, -4.59325165e-07, 2.14612376e-04, -4.85599108e-02, 7.35290867e+01],
           [-3.17099156e-14, 7.26336129e-11, -6.99495385e-08, 3.85531008e-05, -1.33386004e-02, 7.18002889e+01],
           [-6.12280314e-14, 1.67968911e-10, -1.83625538e-07, 1.00457608e-04, -2.86728022e-02, 7.22754350e+01],
           [-1.89897962e-14, 1.52777215e-11, -2.79681508e-09, 1.25689887e-05, -1.33093804e-02, 7.17662251e+01],
           [-2.32118621e-14, 7.30043322e-11, -9.40606298e-08, 6.29728588e-05, -2.28150175e-02, 7.22661091e+01],
           [-8.29749194e-14, 2.31096069e-10, -2.47867121e-07, 1.27576029e-04, -3.24702414e-02, 7.26357609e+01],
           [-2.00718008e-13, 5.44135757e-10, -5.59484466e-07, 2.73028553e-04, -6.46879791e-02, 7.45264998e+01],
           [-7.77420021e-14, 1.97357045e-10, -1.90016273e-07, 8.99659454e-05, -2.30169916e-02, 7.25038258e+01],
           [-5.27296334e-14, 1.49415580e-10, -1.58205132e-07, 8.00275441e-05, -2.13559394e-02, 7.23995451e+01],
           [-6.00198219e-14, 1.55333004e-10, -1.60367908e-07, 7.97754165e-05, -1.94435594e-02, 7.22714399e+01],
           [-8.89919309e-14, 2.40830027e-10, -2.57060475e-07, 1.33002951e-04, -3.32969110e-02, 7.28696020e+01]])

Tabnine | Edit | Test | Explain | Document
@numba.njit
def perform_poly(dt, c):
    poly_results = coeffs[c][0]*dt**5 + coeffs[c][1]*dt**4 + coeffs[c][2]*dt**3 + coeffs[c][3]*dt**2 + coeffs[c][4]*dt + coeffs[c][5]
    return poly_results

# copy coeffs to cuda shared memory
cuda.to_device(coeffs)

Tabnine | Edit | Test | Explain | Document
@cuda.jit
def perform_poly_cuda(dt, c):

    return (
        coeffs[c][0] * math.pow(dt, 5) + coeffs[c][1] * math.pow(dt, 4) + coeffs[c][2] * math.pow(dt, 3) +
        coeffs[c][3] * math.pow(dt, 2) + coeffs[c][4] * dt + coeffs[c][5]
    )
```

# Practical Example 3: LZ AGC with GPUs

How does CPU and GPU approaches differ?

- Arrays have to be copied to GPU

```
Tabnine | Edit | Test | Explain | Document
def process_file(file):
    component = file.split('/SS_skim_')[1][:-5]

    tfile = up.open(file)
    scatters = tfile['Scatters']
    truth = tfile['RQMCTruth']

    # Read arrays
    ss_x = scatters['ss.x_cm'].array()
    ss_y = scatters['ss.y_cm'].array()
    ss_dt = scatters['ss.driftTime_ns'].array()
    ss_s1c = scatters['ss.correctedS1Area_phd'].array()
    ss_s2 = scatters['ss.s2Area_phd'].array()
    ss_s2c = scatters['ss.correctedS2Area_phd'].array()
```

```
Tabnine | Edit | Test | Explain | Document
def process_file_cuda(file):
    component = file.split('/SS_skim_')[1][:-5]

    tfile = up.open(file)
    scatters = tfile['Scatters']
    truth = tfile['RQMCTruth']

    # Read arrays and copy to GPU
    ss_x = cuda.to_device(scatters['ss.x_cm'].array())
    ss_y = cuda.to_device(scatters['ss.y_cm'].array())
    ss_dt = cuda.to_device(scatters['ss.driftTime_ns'].array())
    ss_s1c = cuda.to_device(scatters['ss.correctedS1Area_phd'].array())
    ss_s2 = cuda.to_device(scatters['ss.s2Area_phd'].array())
    ss_s2c = cuda.to_device(scatters['ss.correctedS2Area_phd'].array())
```

# Practical Example 3: LZ AGC with GPUs

How does CPU and GPU approaches differ?

- Result has to be copied back

```
result = process_events(ss_x, ss_y, ss_dt, ss_s1c, ss_s2, ss_s2c, mc_detS1, mc_detS2)
n_fv, n_roi, n_fv_roi, n_mssi, n_fv_mssi, n_roi_mssi, n_fv_roi_mssi = result
```

```
# Launch kernel
n_events = len(scatters['ss.x_cm'].array())
block_size = 256
n_blocks = (n_events + block_size - 1) // block_size

process_events_cuda[n_blocks, block_size](ss_x, ss_y, ss_dt, ss_s1c, ss_s2, ss_s2c, mc_detS1, mc_detS2,
                                          n_fv, n_roi, n_fv_roi, n_mssi, n_fv_mssi, n_roi_mssi, n_fv_roi_mssi)

# Copy result back
cuda.synchronize()
n_fv_host = n_fv.copy_to_host()[0]
n_roi_host = n_roi.copy_to_host()[0]
n_fv_roi_host = n_fv_roi.copy_to_host()[0]
n_mssi_host = n_mssi.copy_to_host()[0]
n_fv_mssi_host = n_fv_mssi.copy_to_host()[0]
n_roi_mssi_host = n_roi_mssi.copy_to_host()[0]
n_fv_roi_mssi_host = n_fv_roi_mssi.copy_to_host()[0]
```

# Practical Example 3: LZ AGC with GPUs

```
cluster = LocalCluster()
client = Client(cluster)
client
```

[5] ✓ 1.7s

**Dask**

- This can still easily be parallelised with dask

```
delayed_results = [dask.delayed(process_file)(file) for file in files]
futures = client.compute(delayed_results)

# monitor the progress
progress(futures)
```

[7] ✓ 0.0s

**Finished:** 4.7s

6 / 6 process_file

```
# Once complete, retrieve the results
results = client.gather(futures)
results_df = pd.DataFrame(results, columns=['Source', 'nSS', 'nSS FV', 'nSS ROI', 'nSS FV ROI', 'nMSSI', 'nMSSI FV', 'nMSSI ROI', 'nMSSI FV ROI', 'eventWeight'])
results_df
```

[8] ✓ 0.0s

| | Source | nSS | nSS FV | nSS ROI | nSS FV ROI | nMSSI | nMSSI FV | nMSSI ROI | nMSSI FV ROI | eventWeight |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Co60_CalibrationSourceTubes | 1400000 | 132410 | 938 | 2 | 799336 | 68986 | 755 | 1 | 5.000000e-09 |
| 1 | Co60_DomePMTs | 1507911 | 874756 | 189 | 9 | 1472377 | 848548 | 189 | 9 | 5.000000e-09 |
| 2 | K40_BottomTruss | 160904 | 61547 | 31 | 0 | 126295 | 44703 | 26 | 0 | 5.000000e-09 |
| 3 | K40_DomePMTs | 107034 | 62464 | 10 | 2 | 84704 | 45883 | 10 | 2 | 5.000000e-09 |
| 4 | Th232-early_BottomTPCPMTBodies | 491017 | 241293 | 75 | 2 | 418065 | 194826 | 73 | 2 | 5.000000e-09 |
| 5 | Th232-late_BottomTPCPMTBases | 260607 | 133536 | 18 | 0 | 255214 | 129945 | 18 | 0 | 5.000000e-09 |

```
delayed_results = [dask.delayed(process_file_cuda)(file) for file in files]
futures = client.compute(delayed_results)
progress(futures)
```

[9] ✓ 0.0s

**Finished:** 10.9s

6 / 6 process_file_cuda

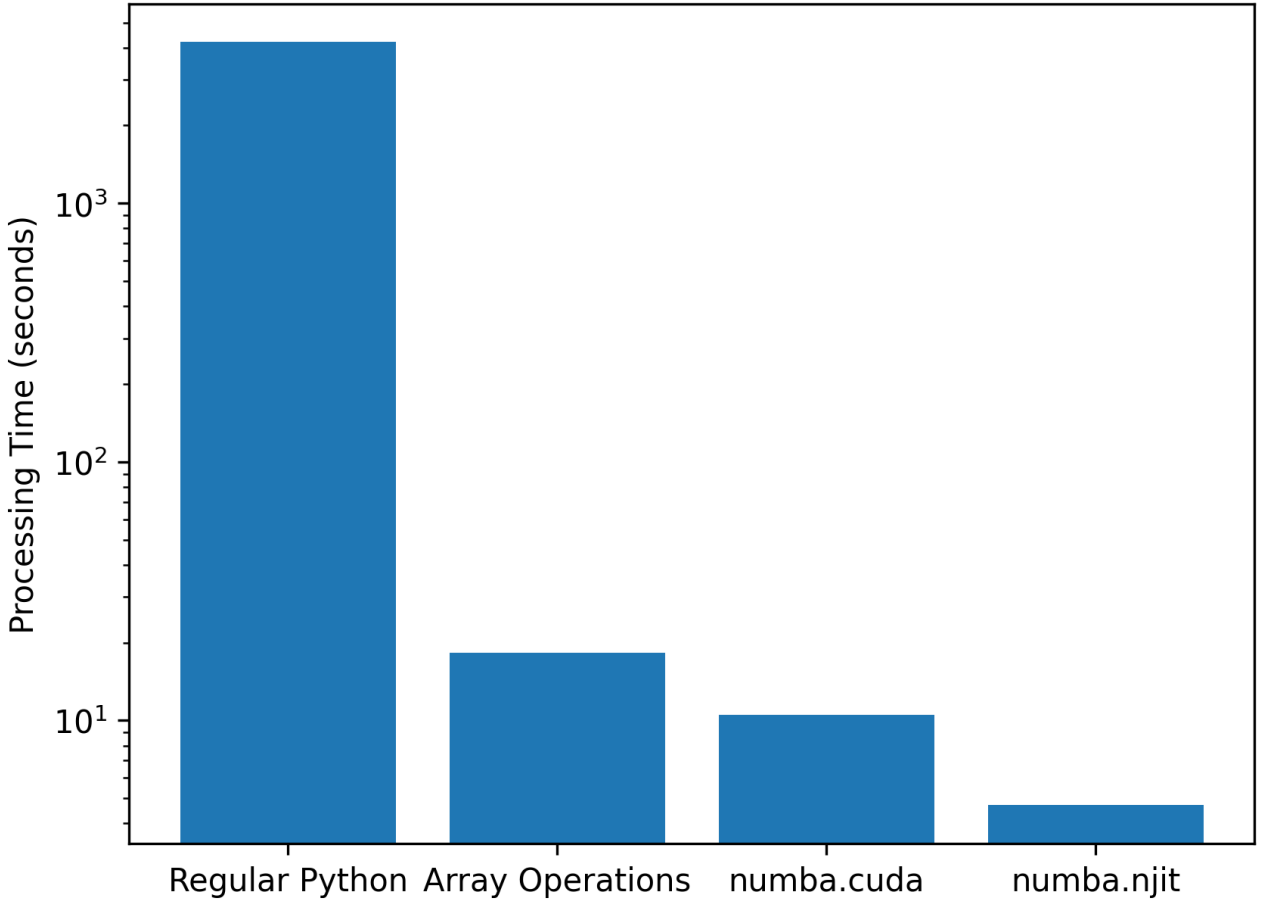
# Practical Example 3: Time Improvement

**Summary**

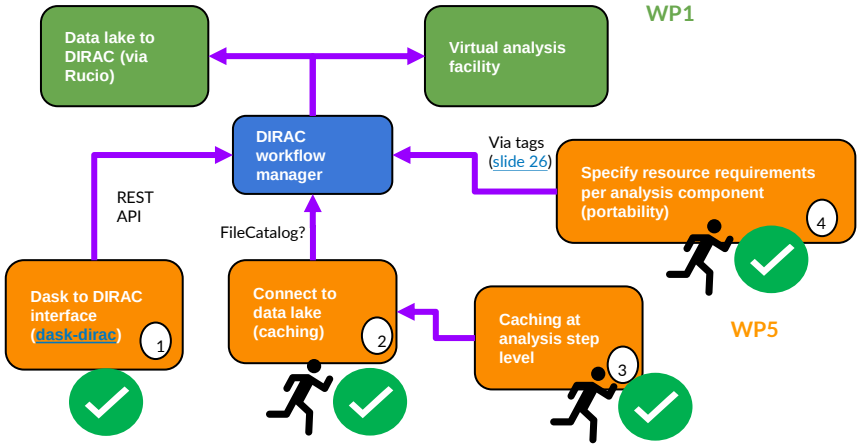
- We think in ‘regular python’ and ‘array operations’
- Minor modifications are required for GPU use
- In this example, GPU overhead is greater than operation speed improvement

**Can this be improved further?**

- The point of this example is to make grid GPUs more accessible
- For GPU there are replacements: CuPy for numpy and cuDF for pandas



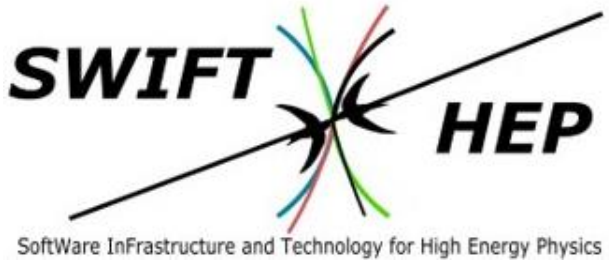
# Summary and plan going forward




- We have a dask-dirac interface
  - Effectively splits up work across these sites
  - We can use request and use GPUs
  - Lowers barrier to DIRAC!
- We low-level cache
  - Between processing steps and persistent
  - Some really complex graphs may not work
- Going forward
  - Will integrate RUCIO when it's ready
  - Write up this work + improve documentation -> invite others to use
  - **This is still a prototype -> need to get production ready**

University of BRISTOL

# Questions?



# BACKUP

# WP5: Roadmap

1. **Dask to DIRAC interface (dask-dirac)**
   - Add extension to dask
   - Dask is able to parallelize any python code
2. **Connect to data lake (caching)**
   - Add the ability to save output after dask instance has closed
3. **Caching at analysis step level**
   - Avoid having to re-run analysis steps
   - Persistent caching
4. **Specify resource requirements per analysis component (portability)**
   - E.g. Let some stages run on GPUs

See talk by Luke Kreczko for some future planning

# 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)

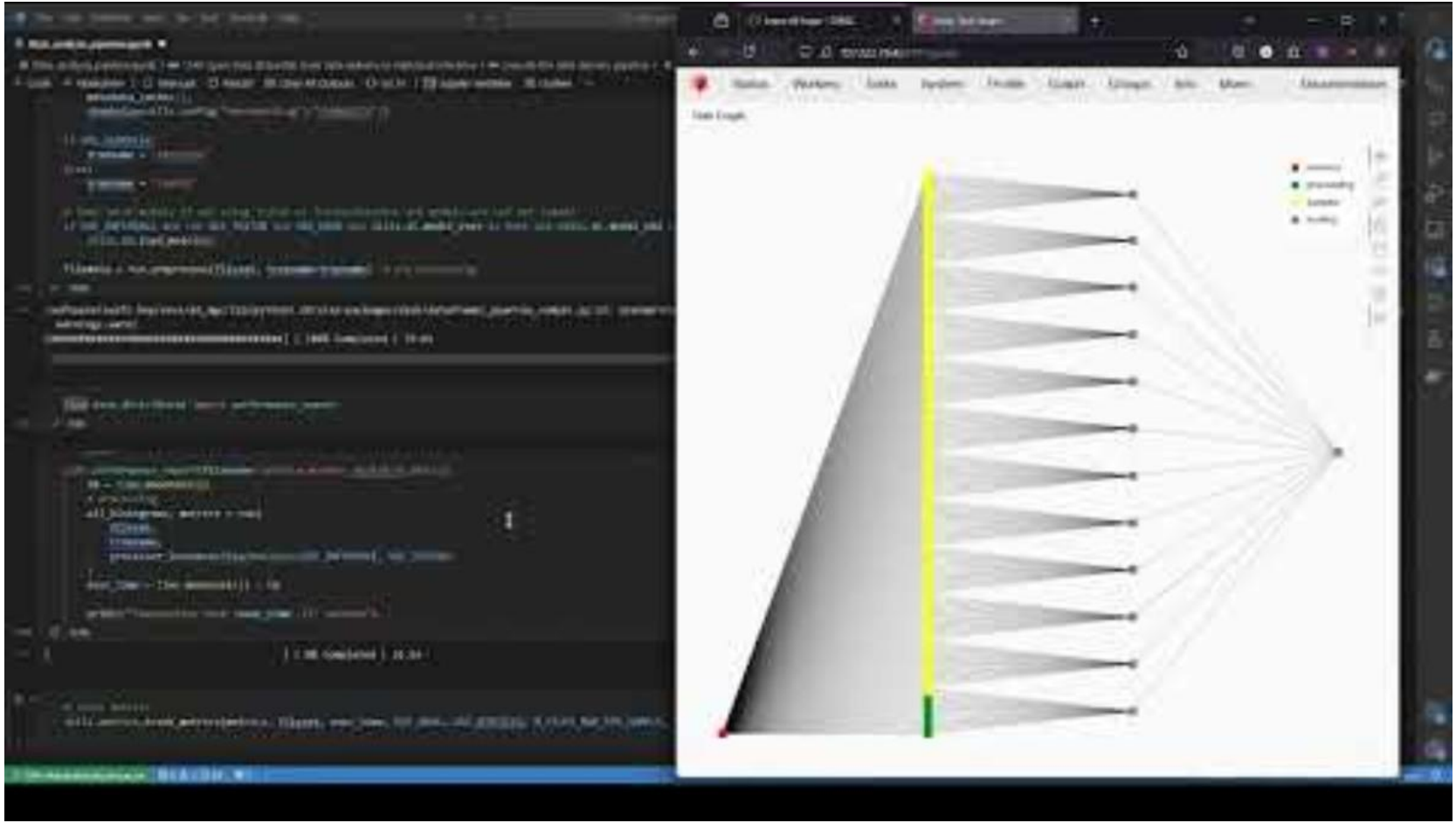
RDataframe

Awkward-array (native support)


FAST-HEP (custom graphs)

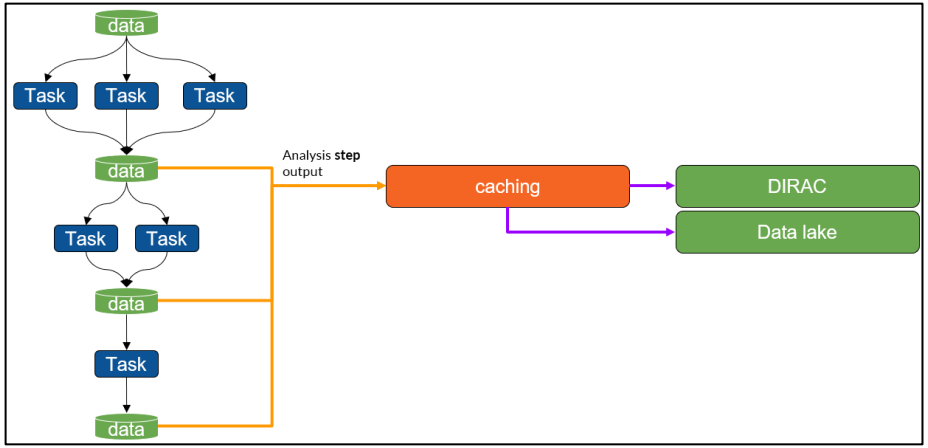

If infrastructure can be used via Dask → wide use is possible

# Practical Example 1: CMS AGC Running example



# What have we been focusing on

**Caching at analysis step level**




## What do we want to demonstrate

- Imagine a complex task that you want to run.
- If you know you are going to have to run this multiple times, but are only modifying one of the functions, you can think of a few ways to reduce the reprocessing;
- Most obvious would be to break up the graph and submit each task separately and save the outputs, and if you are rerunning, load the bits you haven't changed.
- We want that to automatically happen when you submit something.

# Modifying a task graph: Practical example

**Caching at analysis step level**

**This is what we have to play with;**

Task is;

- Calculate hash
- Check if hash exists somewhere
- If exists; replace with load
- If doesn't; add save

hash('func_b', hash('func_a', 2))

```
{('func_a-func_b', 0): (<function func_b at 0x7f5d5a957760>,
                        (<function func_a at 0x7f5d5a9576d0>, 2)),
 ('func_b', 0): ('func_a-func_b', 0),
 ('func_c-func_d', 0): (<function func_d at 0x7f5d5a957880>,
                        (<function func_c at 0x7f5d5a9577f0>, 2)),
 ('func_d', 0): ('func_c-func_d', 0),
 ('func_e', 0): (<function func_e at 0x7f5d5a957910>,
                 ('func_d', 0),
                 ('func_b', 0))}
```

Same as above

hash('func_d', hash('func_c', 2))

Same as above

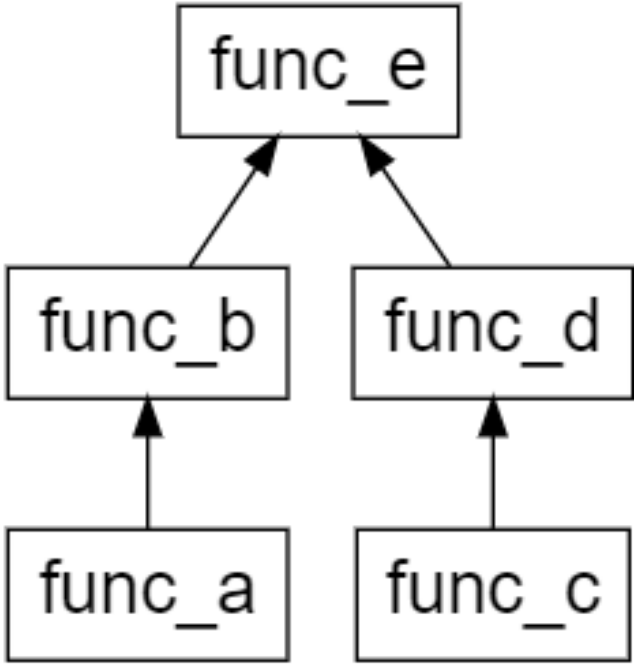
hash('func_e', hash('func_d', hash('hash_c', 2)), hash('func_b', hash('func_a', 2)))

# Modifying a task graph: Practical Example

**Caching at analysis step level**

```python
def func_a(x):
    return x**2

def func_b(x):
    return x**3

def func_c(x):
    return x + 120

def func_d(x):
    return x + 14

def func_e(x, y):
    return x - y
```

```python
# define highlevelgraph
param_a = pd.DataFrame(np.array([2]))
param_b = pd.DataFrame(np.array([2]))

layers = {
    "func_a": {
        ("func_a", 0): (func_a, param_a),
    },
    "func_b": {
        ("func_b", 0): (func_b, ("func_a", 0)),
    },
    "func_c": {
        ("func_c", 0): (func_c, param_b),
    },
    "func_d": {
        ("func_d", 0): (func_d, ("func_c", 0)),
    },
    "func_e": {
        ("func_e", 0): (func_e, ("func_d", 0), ("func_b", 0)),
    },
}
dependencies = {
    "func_a": set(),
    "func_b": {"func_a"},
    "func_c": set(),
    "func_d": {"func_c"},
    "func_e": {"func_b", "func_d"},
}

hlg = HighLevelGraph(layers, dependencies)
```



# Practical Example

**Caching at analysis step level**

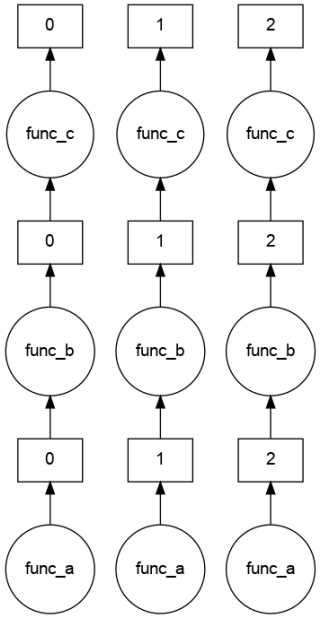
```
{('func_a-func_b', 0): (<function func_b at 0x7f99bb6d3880>,
                        (<function func_a at 0x7f99bb6d37f0>, array([2]))),
 ('func_b', 0): ('func_a-func_b', 0),
 ('func_c-func_d', 0): (<function func_d at 0x7f99bb6d3a30>,
                        (<function func_c at 0x7f99bb6d39a0>, array([2]))),
 ('func_d', 0): ('func_c-func_d', 0),
 ('func_e', 0): (<function func_e at 0x7f99bb6d3ac0>,
                 ('func_d', 0),
                 ('func_b', 0))}
```

```
{('func_a-func_b', 0): (<function func_save at 0x7f99bb6d3be0>,
                        'f84d1e9e761b9b7229bafdf522f347f04b88a65b40568d76f5402ad094b768fd101f3ae957876f0e7bece55676ca102e',
                        (<function func_b at 0x7f99bb6d3880>,
                         (<function func_save at 0x7f99bb6d3be0>,
                          '0776ff206989787bf5a95a35b33716b797efcded6bf71a9e75c1897c5a7ea9ca4f9fb654fe3eba1ebcec7d7911b2dd16',
                          (<function func_a at 0x7f99bb6d37f0>, array([2])))))),
 ('func_b', 0): ('func_a-func_b', 0),
 ('func_c-func_d', 0): (<function func_save at 0x7f99bb6d3be0>,
                        '4c082d6b9b9b918689b81a761abbaae92041057140553ca83e03c4665445a3e78b460e9f0eb8b205e0ef8cb4dd5567f7',
                        (<function func_d at 0x7f99bb6d3a30>,
                         (<function func_save at 0x7f99bb6d3be0>,
                          'f12ce0d8cd6105959827f329b79f2baca8767a4b436cdd843cf008ca50b0ae7754317a320740ecc96d739668308c9d2e',
                          (<function func_c at 0x7f99bb6d39a0>, array([2])))))),
 ('func_d', 0): ('func_c-func_d', 0),
 ('func_e', 0): (<function func_save at 0x7f99bb6d3be0>,
                 '4baeda9eda3c05b98d14b603cea2833cc5b4a3bf1e30cfca90dcaf4f24c9079ea33349840fb93b07c3585cf685dfb2d9',
                 (<function func_e at 0x7f99bb6d3ac0>,
                  ('func_d', 0),
                  ('func_b', 0)))}
```

# Practical Example

**Caching at analysis step level**

```
# Create a Dask Array from the HighLevelGraph
array = Array(hlg, "func_c", shape=(3,), dtype=list, chunks=(1,))
```




```
{('func_a-func_b-func_c', 0): (<function func_c at 0x7efdccb7f6d0>,
                               (<function func_b at 0x7efdccb7f640>,
                                (<function func_a at 0x7efdccb7f5b0>, 2))),
 ('func_a-func_b-func_c', 1): (<function func_c at 0x7efdccb7f6d0>,
                               (<function func_b at 0x7efdccb7f640>,
                                (<function func_a at 0x7efdccb7f5b0>, 3))),
 ('func_a-func_b-func_c', 2): (<function func_c at 0x7efdccb7f6d0>,
                               (<function func_b at 0x7efdccb7f640>,
                                (<function func_a at 0x7efdccb7f5b0>, 4))),
 ('func_c', 0): ('func_a-func_b-func_c', 0),
 ('func_c', 1): ('func_a-func_b-func_c', 1),
 ('func_c', 2): ('func_a-func_b-func_c', 2)}
```

# Practical Example

**Caching at analysis step level**

```
{('func_a-func_b-func_c', 0): (<function func_c at 0x7efdccb7f6d0>,
                               (<function func_b at 0x7efdccb7f640>,
                                (<function func_a at 0x7efdccb7f5b0>, 2))),
 ('func_a-func_b-func_c', 1): (<function func_c at 0x7efdccb7f6d0>,
                               (<function func_b at 0x7efdccb7f640>,
                                (<function func_a at 0x7efdccb7f5b0>, 3))),
 ('func_a-func_b-func_c', 2): (<function func_c at 0x7efdccb7f6d0>,
                               (<function func_b at 0x7efdccb7f640>,
                                (<function func_a at 0x7efdccb7f5b0>, 4))),
 ('func_c', 0): ('func_a-func_b-func_c', 0),
 ('func_c', 1): ('func_a-func_b-func_c', 1),
 ('func_c', 2): ('func_a-func_b-func_c', 2)}
```

```
{('func_a-func_b-func_c', 0): (<function func_c at 0x7fd85844cdc0>,
                               (<function func_b at 0x7fd85844cd30>,
                                (<function func_a at 0x7fd858075870>, 2))),
 ('func_a-func_b-func_c', 1): (<function func_c at 0x7fd85844cdc0>,
                               (<function func_b at 0x7fd85844cd30>,
                                (<function func_a2 at 0x7fd85844c9d0>,))),
 ('func_a-func_b-func_c', 2): (<function func_c at 0x7fd85844cdc0>,
                               (<function func_b at 0x7fd85844cd30>,
                                (<function func_a at 0x7fd858075870>, 4))),
 ('func_c', 0): ('func_a-func_b-func_c', 0),
 ('func_c', 1): ('func_a-func_b-func_c', 1),
 ('func_c', 2): ('func_a-func_b-func_c', 2)}
```

# Worker Plugin example

**Caching at analysis step level**

## Description

- When worker is shutdown/killed or the plugin is disconnected, the directory path and contents are displayed

```python
from distributed.diagnostics.plugin import WorkerPlugin
from distributed.diagnostics.plugin import forward_stream
import contextlib
import os

class ForwardOutput(WorkerPlugin):

    tabnine: test | explain | document | ask
    def setup(self, worker):
        self._exit_stack = contextlib.ExitStack()
        self._exit_stack.enter_context(forward_stream("stdout", worker=worker))
        self._exit_stack.enter_context(forward_stream("stderr", worker=worker))

    tabnine: test | explain | document | ask
    def teardown(self, worker):
        print("{}: {} ".format(os.getcwd(), os.listdir(os.getcwd())))
        self._exit_stack.close()
```

```
plugin = ForwardOutput()
✓ 0.0s

client.register_plugin(plugin, name='my-plugin')
✓ 0.0s

{'tcp://130.246.45.124:50000': {'status': 'OK'}}
```

```
# remove plugin
client.unregister_worker_plugin(name='my-plugin')
✓ 0.0s

/scratch/condor/dir_112165/406MDmIA544n8FVDjqQVj3jqav4PzpABFKDmon9MDmXIFKDmzPLhKm/DIRAC_mMYY6bpilot/1300: ['job.info', 'std.err', 'std.out']
2024-03-14 19:01:02,044 - distributed.worker - INFO - Removing Worker plugin my-plugin

{'tcp://130.246.45.124:50000': {'status': 'OK'}}
```

# Worker Plugin example

**Caching at analysis step level**

## Description

- When worker is transitioning between states, the state is printed and the number of transitions are counted

```python
class MyPlugin(WorkerPlugin):
    def __init__(self, logger):
        self.worker = None
        self.logger = logger
        self.n_transitions = 0

    def setup(self, worker):
        self.worker = worker

    def transition(self, key, start, finish, *args, **kwargs):
        self.n_transitions += 1
        print('state: {}    n_transitions: {}'.format(finish, self.n_transitions))
```

```
state: executing    n_transitions: 223
state: memory    n_transitions: 224
state: released    n_transitions: 225
state: forgotten    n_transitions: 226
state: waiting    n_transitions: 227
state: ready    n_transitions: 228
state: executing    n_transitions: 229
state: released    n_transitions: 230
state: forgotten    n_transitions: 231
state: memory    n_transitions: 232
state: waiting    n_transitions: 233
```

# Scheduler Plugin example

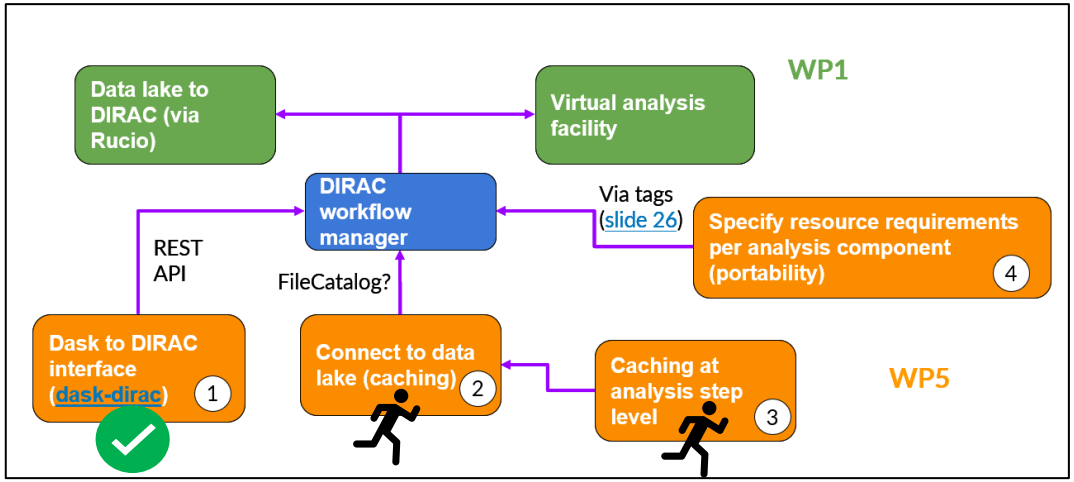
**Caching at analysis step level**

## Description

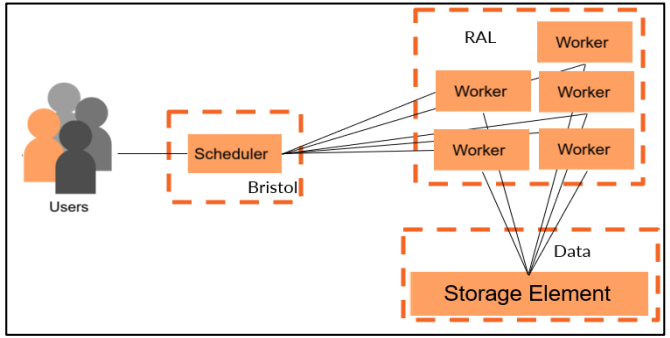
- Remove a task from the graph if it has already been executed

```python
class MySchedulerPlugin(SchedulerPlugin):

    global previous_executed_tasks

    tabnine: test | explain | document | ask
    def update_graph(self, scheduler, keys=None, tasks=None, restrictions=None, **kwargs):
        current_time = datetime.now().strftime('%H:%M:%S')
        print("-------------------------------------------")
        print(f"Current time: {current_time}")
        print(f"Tasks submitted: {tasks}")
        print(f"Prevously Executed Tasks: {previous_executed_tasks}")
        print(f"Keys: {keys}")
        print(f"scheduler tasks: {scheduler.tasks}")
        print(f"scheduler unrunnable: {scheduler.unrunnable}")

        tasks_to_execute = []
        for task in tasks:
            if task in previous_executed_tasks:
                print(f"Task {task} has already been executed - removing from task list")
            else:
                tasks_to_execute.append(task)
                previous_executed_tasks.append(task)

        tasks = tasks_to_execute
        keys = set(tasks)

        print(f"Tasks to be executed: {tasks_to_execute}")
        print(f'keys: {keys}')

        # Update the scheduler's tasks and keys
        scheduler.tasks = {key: scheduler.tasks[key] for key in keys}
        scheduler.unrunnable = {key: scheduler.unrunnable[key] for key in keys if key in scheduler.unrunnable}
        print(f"scheduler tasks: {scheduler.tasks}")
        print(f"scheduler unrunnable: {scheduler.unrunnable}")

    tabnine: test | explain | document | ask
    def transition(self, key, start, finish, *args, **kwargs):
        print("-------------------------------------------")
        print(f"Task {key} is transitioning from {start} to {finish}")
```

# Where we left things last time




**Caching at analysis step level**




```python
class MyPlugin(WorkerPlugin):
    def __init__(self, logger):
        self.worker = None

    def setup(self, worker):
        self.worker = worker

    def transition(self, key, start, finish, *args, **kwargs):
        if finish == "memory":
            # Add data
            requests.request("POST", url, json=payload, headers=headers)
```

- Began looking at intermediate result caching using scheduler plugins
- Allows for tasks to be run when a worker connects/disconnects or transitions (e.g. from running to memory) a job
- Idea: perform check (ideally via RUCIO) if result exists before a task starts

# Practical Example 3: LZ AGC with GPUs

```
Tabnine | Edit | Test | Explain | Document
@numba.jit
def process_events(ss_x, ss_y, ss_dt, ss_s1c, ss_s2, ss_s2c, mc_detS1, mc_detS2):
    n_fv, n_roi, n_fv_roi = 0, 0, 0
    n_mssi, n_fv_mssi, n_roi_mssi, n_fv_roi_mssi = 0, 0, 0, 0

    # Loop over events
    for i in range(len(ss_x)):

        # Evaluate cuts
        cut_bool_resistor_fv = fv_resistor(ss_x[i], ss_y[i])
        cut_bool_z = fv_z(ss_dt[i] / 1000)
        cut_bool_r = fv_phi_r(ss_x[i], ss_y[i], ss_dt[i] / 1000)
        fv_cut = cut_bool_resistor_fv and cut_bool_z and cut_bool_r
        roi_cut = roi(ss_s1c[i], ss_s2[i], ss_s2c[i])

        # Examine if MSSI
        nS1, nS2 = 0, 0
        for j in range(len(mc_detS1[i])):
            if mc_detS1[i][j] > 0.:
                nS1 += 1
            if mc_detS2[i][j] > 0.:
                nS2 += 1
        is_mssi = nS1 > nS2

        # Now evaluate cuts and count events
        if is_mssi:
            n_mssi += 1
        if fv_cut:
            n_fv += 1
            if is_mssi:
                n_fv_mssi += 1
            if roi_cut:
                n_fv_roi += 1
                if is_mssi:
                    n_fv_roi_mssi += 1
        if roi_cut:
            n_roi += 1
            if is_mssi:
                n_roi_mssi += 1

    return n_fv, n_roi, n_fv_roi, n_mssi, n_fv_mssi, n_roi_mssi, n_fv_roi_mssi
```

```
Tabnine | Edit | Test | Explain | Document
def process_events(ss_x, ss_y, ss_dt, ss_s1c, ss_s2, ss_s2c, mc_detS1, mc_detS2):
    # Evaluate cuts
    cut_bool_resistor_fv = fv_resistor(ss_x, ss_y)
    cut_bool_z = fv_z(ss_dt / 1000)
    cut_bool_r = fv_phi_r(ss_x, ss_y, ss_dt / 1000)
    fv_cut = cut_bool_resistor_fv & cut_bool_z & cut_bool_r
    roi_cut = roi(ss_s1c, ss_s2, ss_s2c)

    nS1 = ak.sum(mc_detS1 > 0., axis=1)
    nS2 = ak.sum(mc_detS2 > 0., axis=1)
    is_mssi = nS1 > nS2


    # Now apply cuts and count number of events
    n_mssi = sum(is_mssi)
    n_fv_mssi = sum(is_mssi & fv_cut)
    n_roi_mssi = sum(is_mssi & roi_cut)
    n_fv_roi_mssi = sum(is_mssi & fv_cut & roi_cut)

    n_fv = sum(fv_cut)
    n_roi = sum(roi_cut)
    n_fv_roi = sum(fv_cut & roi_cut)

    return n_fv, n_roi, n_fv_roi, n_mssi, n_fv_mssi, n_roi_mssi, n_fv_roi_mssi
```