Finanziato dall'Unione europea
NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

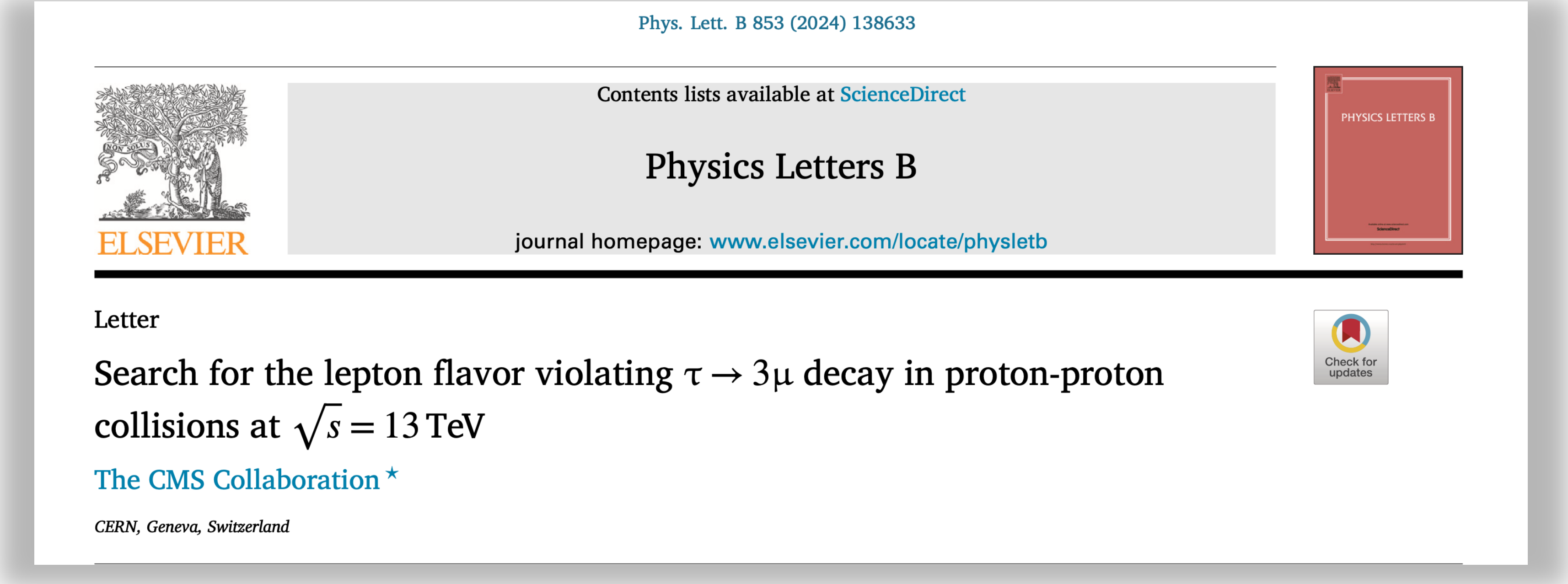
ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

CMS Flavor physics use-case on the ICSC Analysis Facility

Workshop on "Quasi-Interactive Analysis of Big Data with High Throughput", 8-10 Jan 2025, Bologna

Federica Maria Simone
INFN & Polytechnic Bari

Flavor physics use-case: LFV in the charged sector $\tau \to 3\mu$


Phys. Lett. B 853 (2024) 138633

Contents lists available at ScienceDirect

Physics Letters B

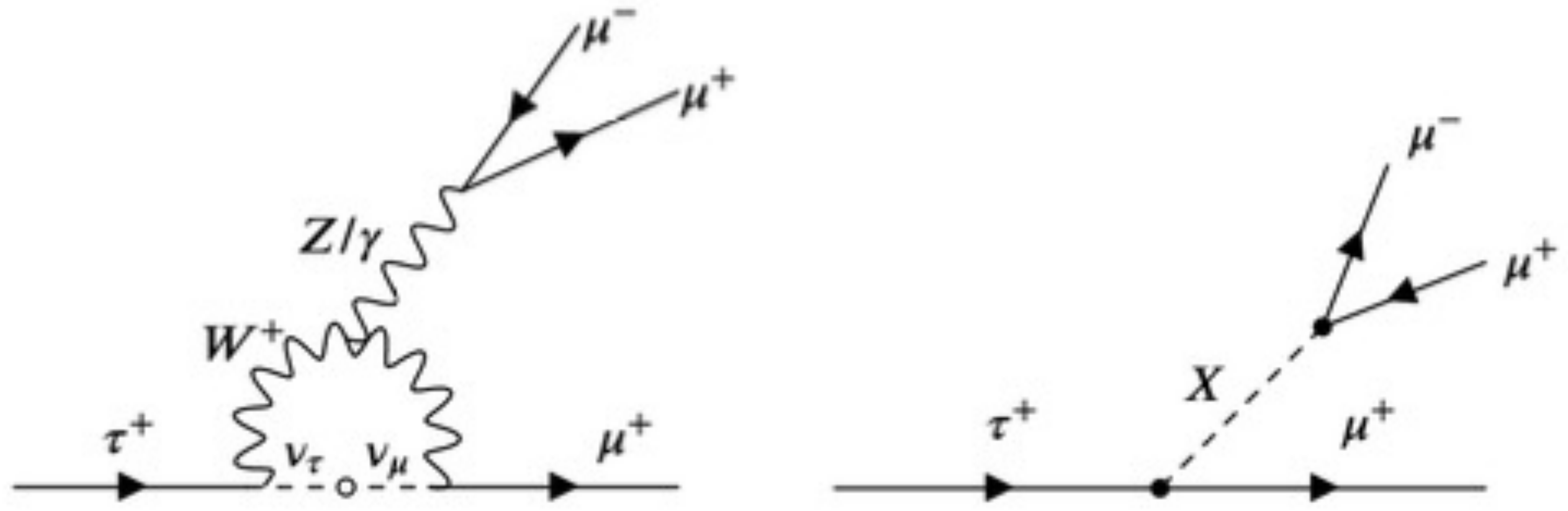
journal homepage: www.elsevier.com/locate/physletb

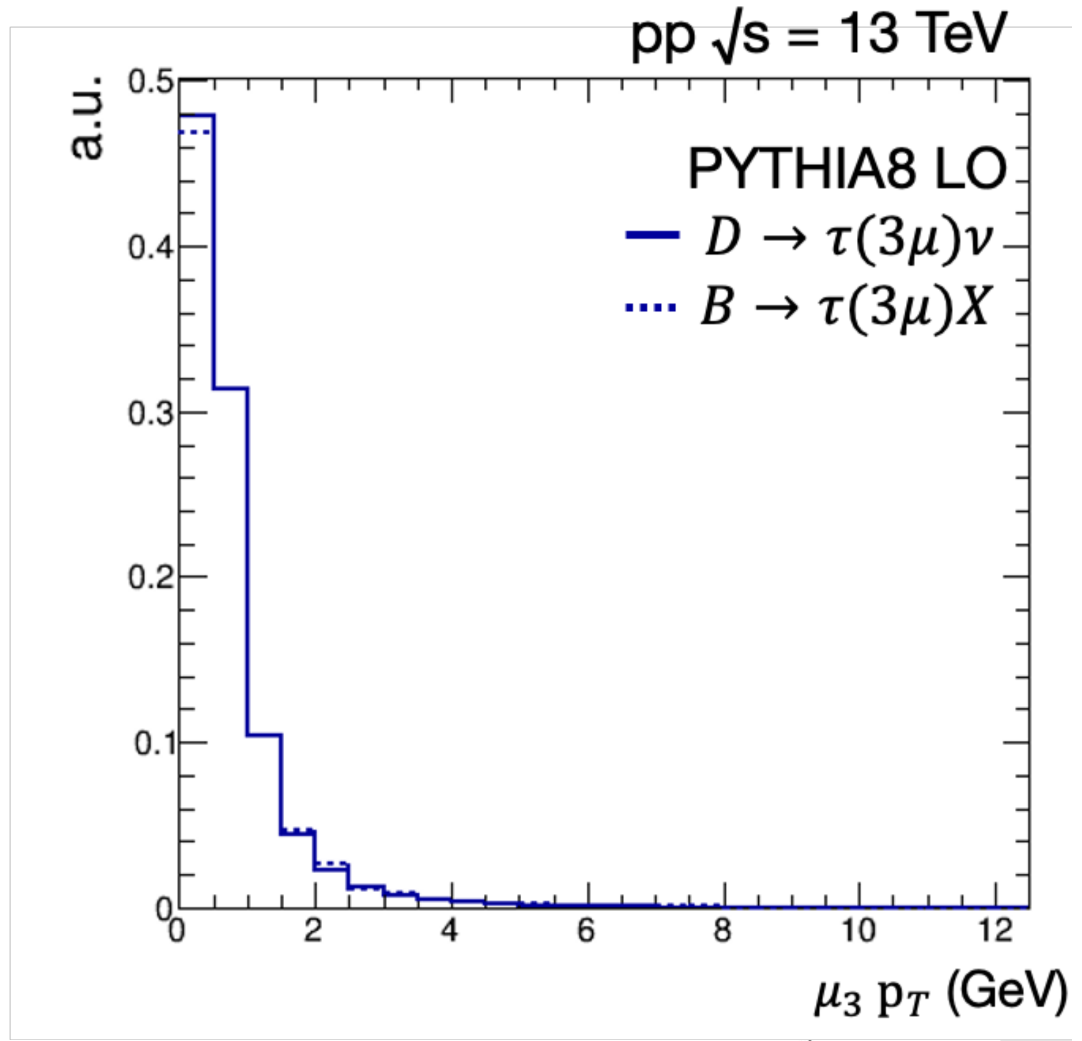
Letter

Search for the lepton flavor violating $\tau \to 3\mu$ decay in proton-proton collisions at $\sqrt{s} = 13$ TeV

The CMS Collaboration *

CERN, Geneva, Switzerland





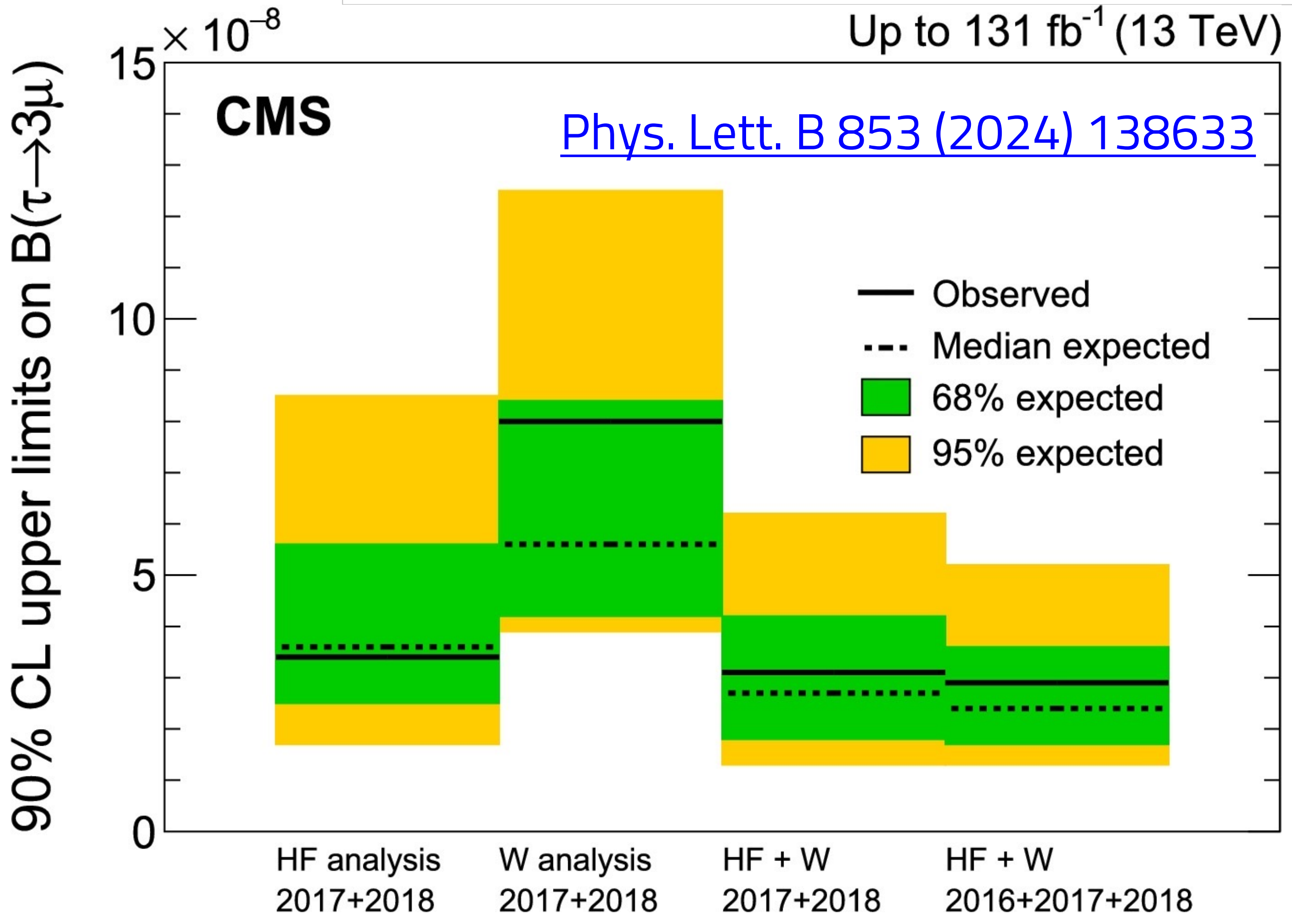
Search for $\boldsymbol{\tau \to 3\mu}$ decays, which have very small SM branching fractions $\mathrm{BR_{SM}} \sim \mathcal{O}(10^{-55})$, while being predicted with sizable BR in several BSM scenarios $\mathrm{BR_{BSM}} \sim \mathcal{O}(10^{-10} \div 10^{-8})$

- $\boldsymbol{\tau}$ leptons produced in D and B meson decays provide large statistics at LHC experiments, but are only accessible with **low-p_T muon triggers**

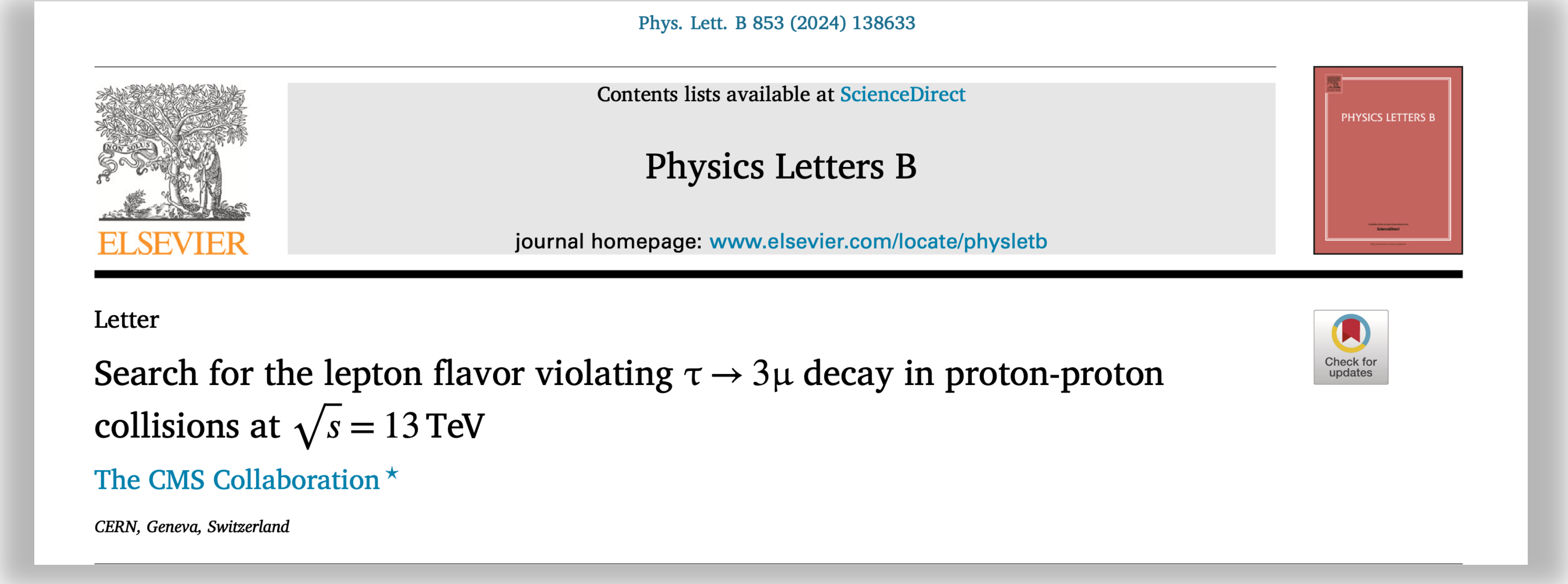
- Analysis of Run 2 data recently published, **stat. limited**

  → benefitting from inclusive low-p_T muon L1 trigger in **Run 3**

  → technical challenge: **new datasets are ×2 ÷ 3 times heavier**



Flavor physics use-case: LFV in the charged sector $\tau \to 3\mu$


Phys. Lett. B 853 (2024) 138633

Contents lists available at ScienceDirect

Physics Letters B

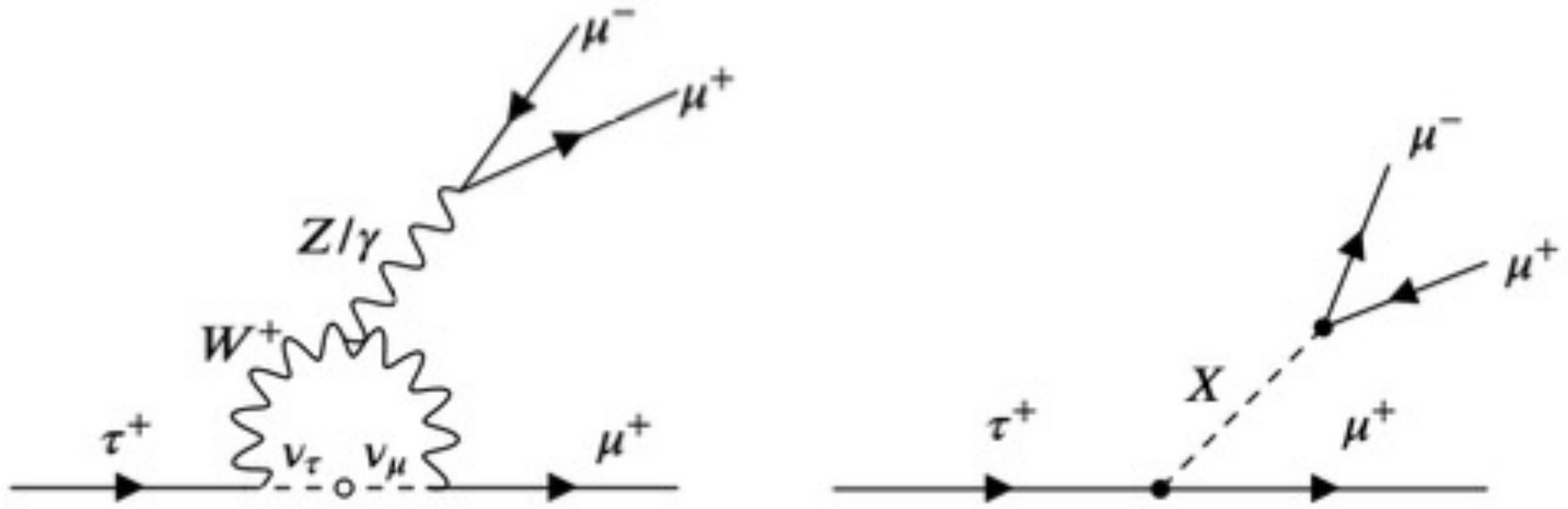
journal homepage: www.elsevier.com/locate/physletb

Letter

Search for the lepton flavor violating $\tau \to 3\mu$ decay in proton-proton collisions at $\sqrt{s} = 13$ TeV

The CMS Collaboration *

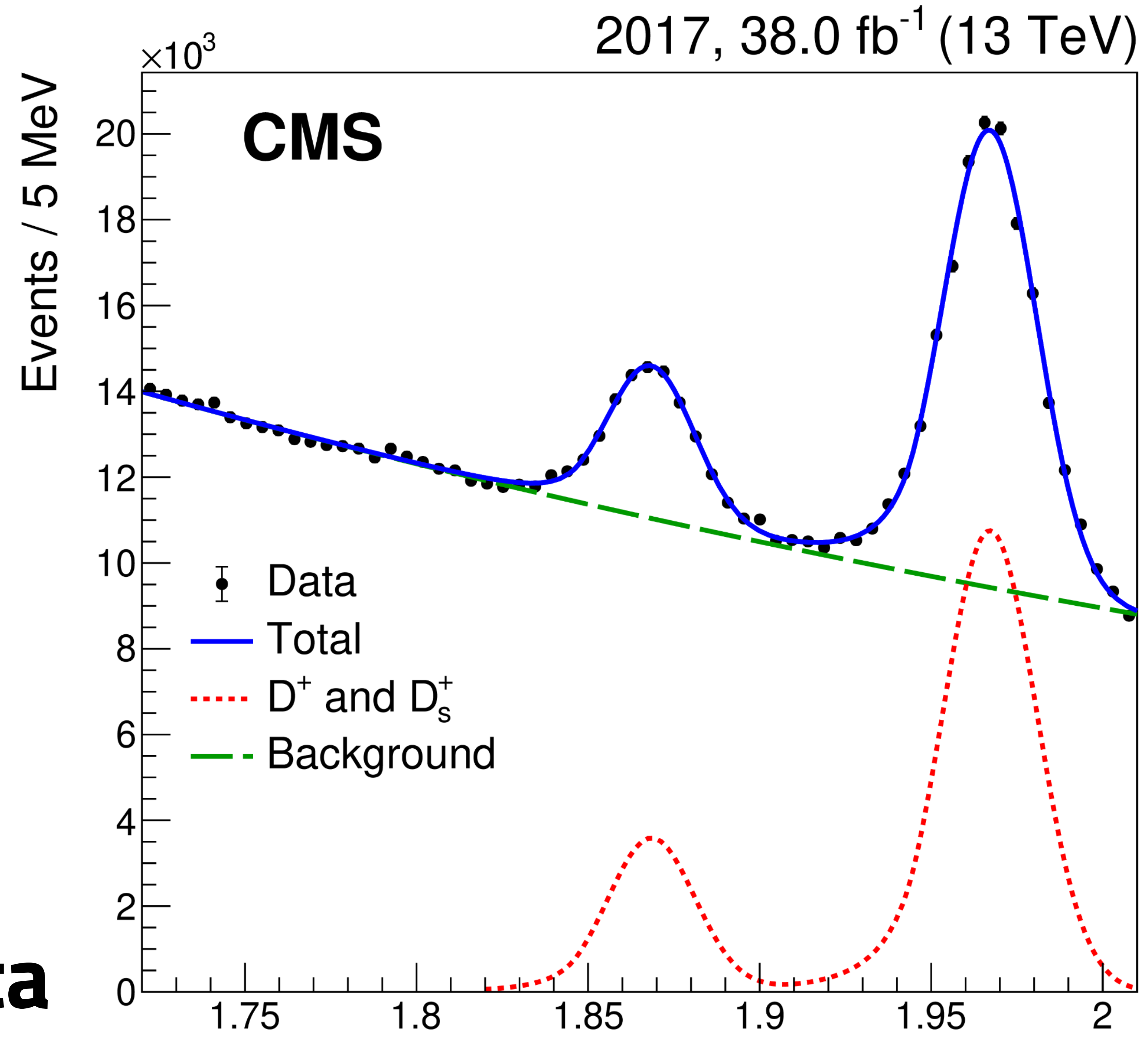
CERN, Geneva, Switzerland



Search for $\boldsymbol{\tau \to 3\mu}$ decays, which have very small SM branching fractions $\mathrm{BR_{SM}} \sim \mathcal{O}(10^{-55})$, while being predicted with sizable BR in several BSM scenarios $\mathrm{BR_{BSM}} \sim \mathcal{O}(10^{-10} \div 10^{-8})$

- $\boldsymbol{\tau}$ leptons produced in D and B meson decays provide large statistics at LHC experiments, but are only accessible with **low-p_T muon triggers**
- The normalisation channel used as a benchmark: $D_s^+ \to \phi(\mu\mu)\pi^+$

  → **cut-based analysis + mass fit for measuring the D_s^+ yield in data**



Phys. Lett. B 853 (2024) 138633

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

$D_s^+ \to \phi(\mu\mu)\pi^+$ analysis workflow

CMS Dataset

- Data collected by low p_T dimuon triggers
- MiniAOD data tier **centrally produced**

ROOT ntuples

- **Skimmed** data, events with 2μ+1track final state
- Saving only physics objects of interest
- **Plain data format**, ~ 5 GB / fb-1, stored on eos

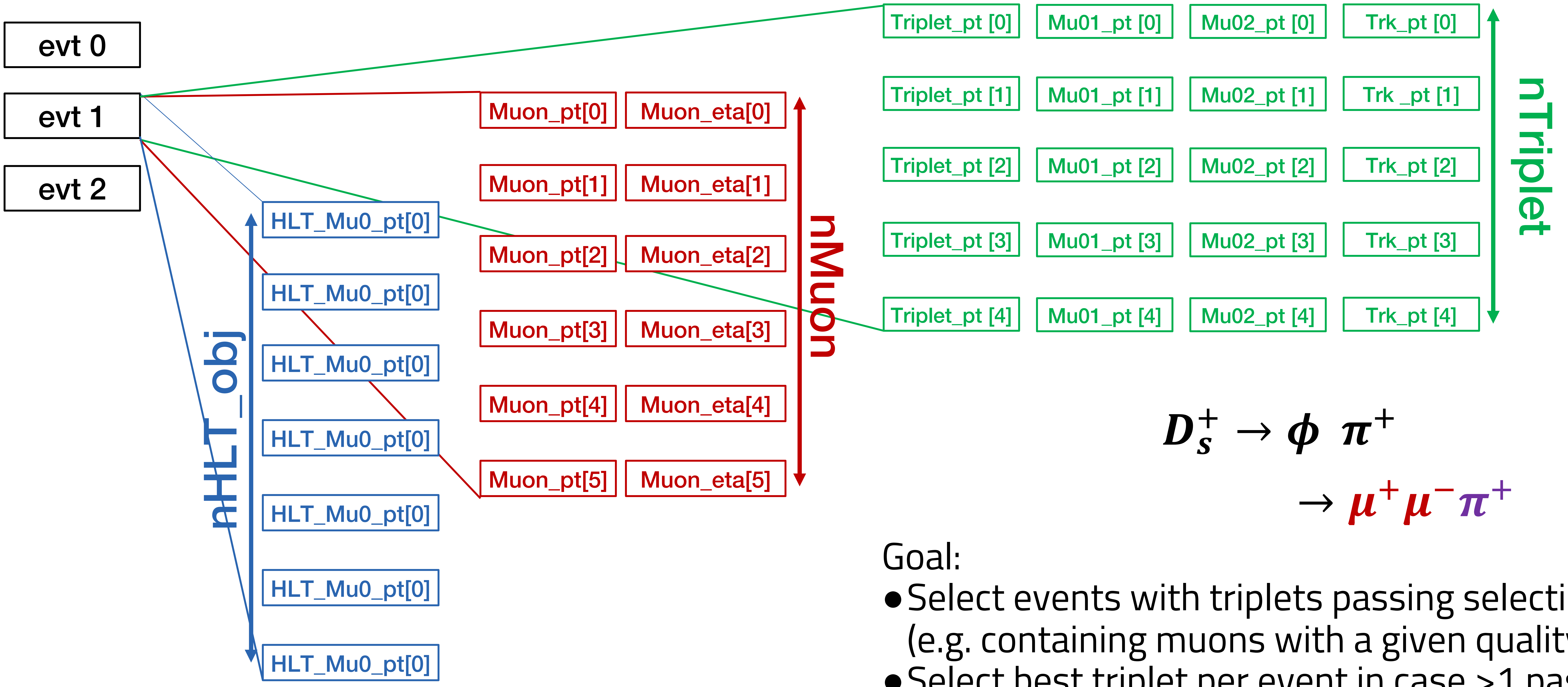
Analysis

- Define **high-level variables**
- Apply **scale factors** and corrections
- Apply **selections**, select best D_s candidate per event
- **Fit** the 2μ+1track invariant mass

- **Legacy: approach** Loop-based analysis implemented using ROOT TTree:MakeClass
  - split computation in batches of input files, run separately as HTCondor jobs, gather the output rootfiles

- **New**: Ntuples read as RDataFrame, almost all operations “lazy” → no loop triggered till the end
  - going distributed using ROOT RDataFrame distributed features, with Dask backend.

Ntuples are nanoAOD-like



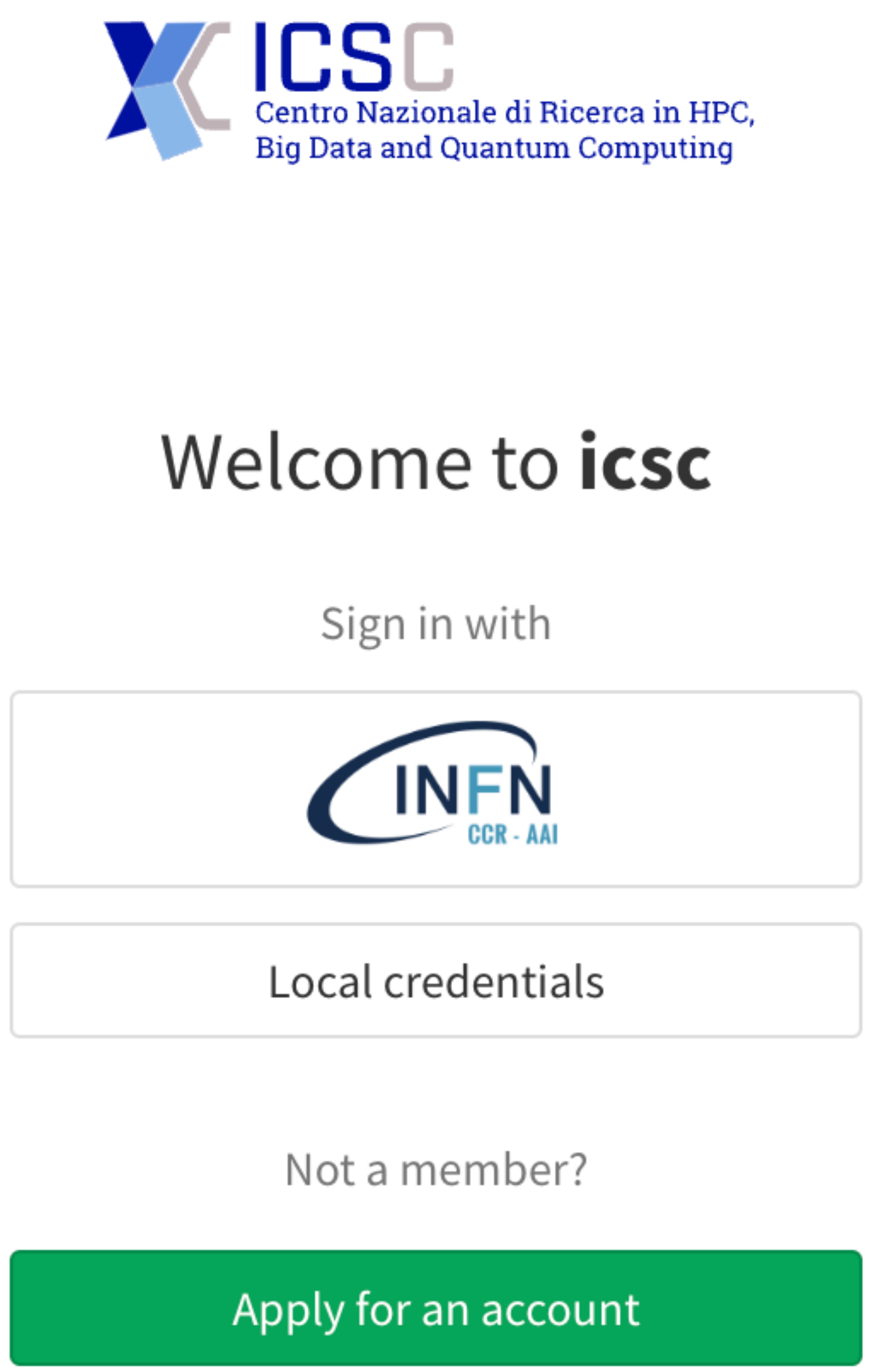
$$D_s^+ \to \phi\ \pi^+$$
$$\to \mu^+\mu^-\pi^+$$

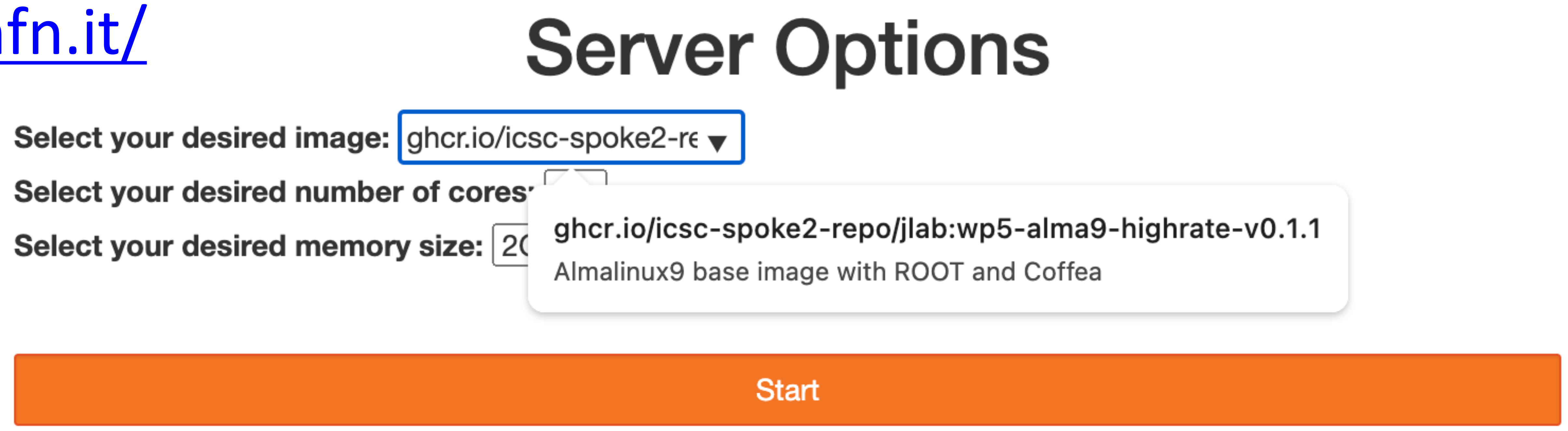
Goal:

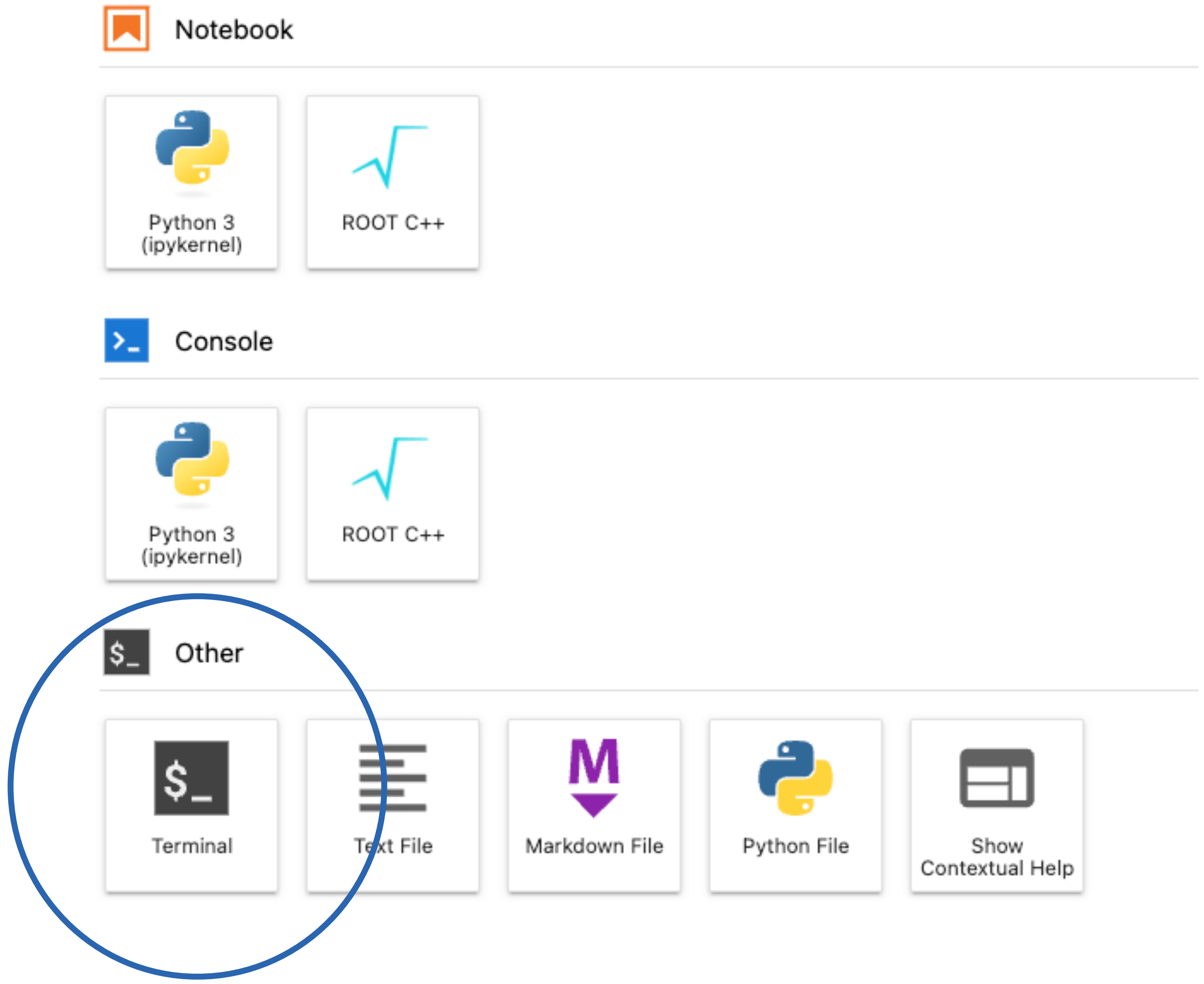
- Select events with triplets passing selections (e.g. containing muons with a given quality)
- Select best triplet per event in case >1 pass

Getting started: access to the AF

https://hub.131.154.98.51.myip.cloud.infn.it/







Get the code for this tutorial

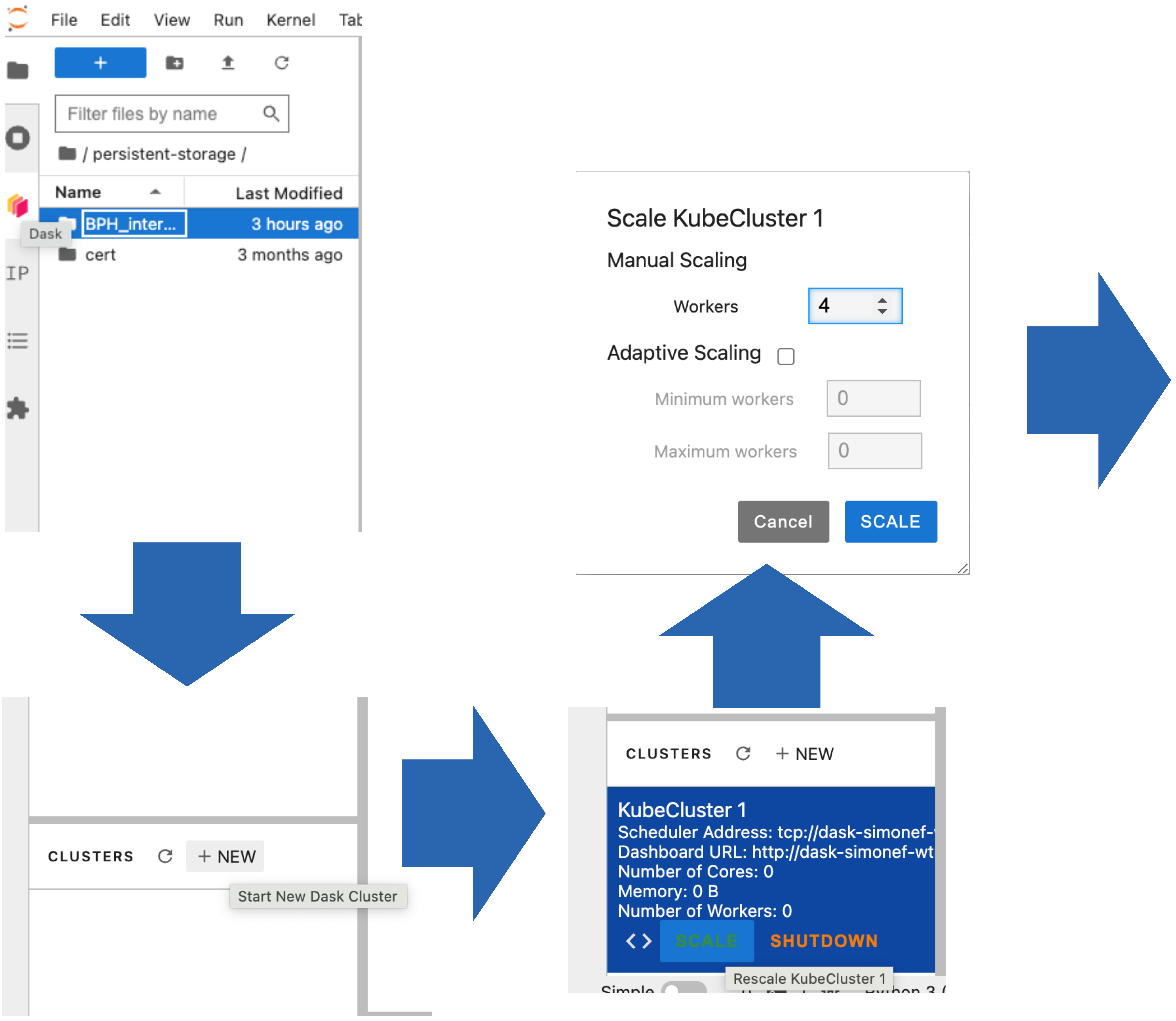
```
cd /opt/workspace/persistent-storage/

git clone https://github.com/fsimone91/BPH_interactive_analysis.git

mkdir ../cert
```

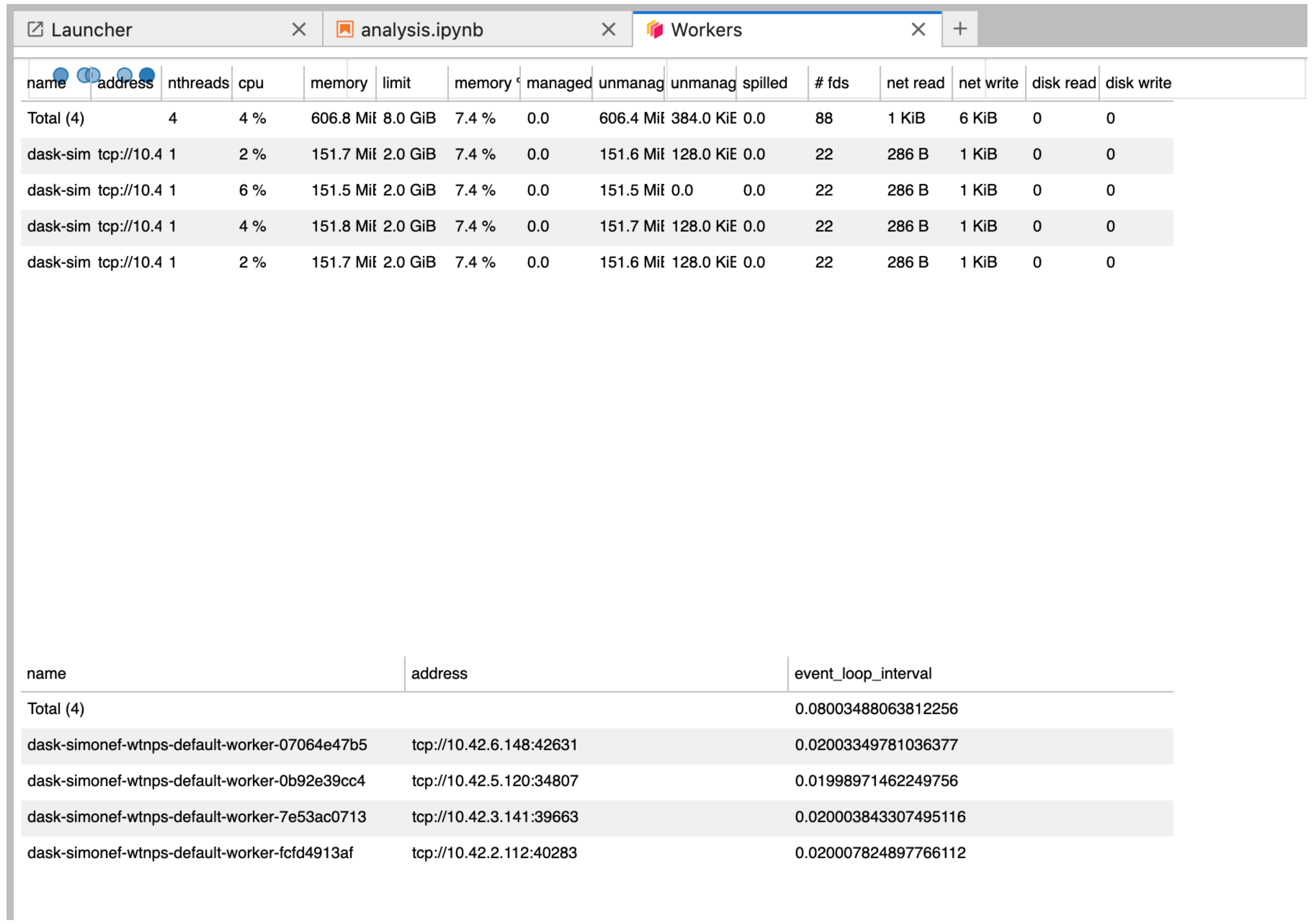
… copy here your `userkey.pem` and `usercert.pem` files, then generate your X509 user proxy certificate

```
voms-proxy-init -cert ../cert/usercert.pem -key ../cert/userkey.pem
```

Start the scheduler

- Start a New Dask Cluster
- Scale it
- Check the Dashboard!





name	address	nthreads	cpu	memory	limit	memory '	managed	unmanag	unmanag	spilled	# fds	net read	net write	disk read	disk write
Total (4)		4	4 %	606.8 MiE	8.0 GiB	7.4 %	0.0	606.4 MiE	384.0 KiE	0.0	88	1 KiB	6 KiB	0	0
dask-sim	tcp://10.4	1	2 %	151.7 MiE	2.0 GiB	7.4 %	0.0	151.6 MiE	128.0 KiE	0.0	22	286 B	1 KiB	0	0
dask-sim	tcp://10.4	1	6 %	151.5 MiE	2.0 GiB	7.4 %	0.0	151.5 MiE	0.0	0.0	22	286 B	1 KiB	0	0
dask-sim	tcp://10.4	1	4 %	151.8 MiE	2.0 GiB	7.4 %	0.0	151.7 MiE	128.0 KiE	0.0	22	286 B	1 KiB	0	0
dask-sim	tcp://10.4	1	2 %	151.7 MiE	2.0 GiB	7.4 %	0.0	151.6 MiE	128.0 KiE	0.0	22	286 B	1 KiB	0	0

name	address	event_loop_interval
Total (4)		0.08003488063812256
dask-simonef-wtnps-default-worker-07064e47b5	tcp://10.42.6.148:42631	0.02003349781036377
dask-simonef-wtnps-default-worker-0b92e39cc4	tcp://10.42.5.120:34807	0.01998971462249756
dask-simonef-wtnps-default-worker-7e53ac0713	tcp://10.42.3.141:39663	0.020003843307495116
dask-simonef-wtnps-default-worker-fcfd4913af	tcp://10.42.2.112:40283	0.020007824897766112

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

To the notebook: call the Dask Client using the python API

analysis.ipynb

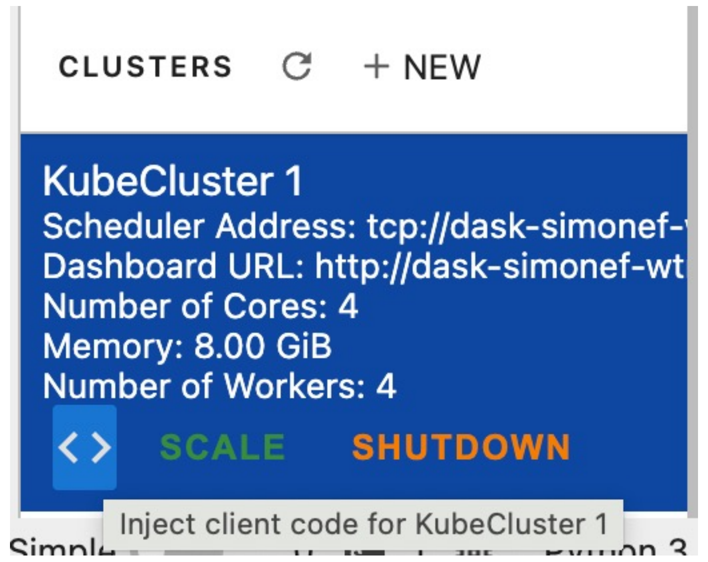
Basic imports

```
[1]: import sys, os, time
     start = time.time()
     import json
     import ROOT
```

Dask scheduler

```
[2]: from dask.distributed import Client, performance_report
```

```
[3]: Local = False

     if Local:
         from dask.distributed import LocalCluster
         cluster = LocalCluster()
         client = Client(cluster.scheduler.address)
```



- Define a Dask Client, either Local or Distributed

Now start new Dask cluster, scale the number of workers

```
[4]: from dask.distributed import Client

     client = Client("tcp://dask-simonef-wtnps-scheduler.jhub:8786")
     client
```

[4]: **Client**

Client-91e4b50c-ccda-11ef-864e-866b7cfbeaf6

Connection method: Direct

Dashboard: http://dask-simonef-wtnps-scheduler.jhub:8787/status

Launch dashboard in JupyterLab

▾ Scheduler Info

Scheduler

Scheduler-04cd4cca-000d-4905-87e5-09f7881756fb

Comm: tcp://10.42.7.128:8786	**Workers:** 4
Dashboard: http://10.42.7.128:8787/status	**Total threads:** 4
Started: 41 minutes ago	**Total memory:** 8.00 GiB

▸ Workers

Almost ready: X509 user proxy

- Upload your X509 proxy file to the workers
- Set environmental variables in the workers

X509 proxy configuration

The `/tmp/x509up_u` file should be generated prior running the notebook using `voms-proxy-init -cert ../cert/usercert.pem -key ../cert/userkey.pem`

```
[9]: from distributed.diagnostics.plugin import UploadFile
client.register_worker_plugin(UploadFile("/tmp/x509up_u0"))
```

```
/tmp/ipykernel_676/2847743139.py:2: DeprecationWarning: `Client.register_worker_plugin` has
been deprecated; please use `Client.register_plugin` instead
  client.register_worker_plugin(UploadFile("/tmp/x509up_u0"))
```

```
[9]: {'tcp://10.42.0.75:43259': {'status': 'OK'},
 'tcp://10.42.0.76:42581': {'status': 'OK'},
 'tcp://10.42.1.93:32875': {'status': 'OK'},
 'tcp://10.42.1.94:42729': {'status': 'OK'},
 'tcp://10.42.2.68:41105': {'status': 'OK'},
 'tcp://10.42.2.69:41157': {'status': 'OK'},
 'tcp://10.42.3.92:36155': {'status': 'OK'},
 'tcp://10.42.3.93:34933': {'status': 'OK'},
 'tcp://10.42.3.94:43541': {'status': 'OK'},
 'tcp://10.42.4.76:42331': {'status': 'OK'},
 'tcp://10.42.4.77:38303': {'status': 'OK'},
 'tcp://10.42.4.78:43961': {'status': 'OK'},
 'tcp://10.42.4.79:45559': {'status': 'OK'},
 'tcp://10.42.6.89:34275': {'status': 'OK'},
```

```
[10]: def set_proxy(dask_worker):
    import os
    import shutil
    working_dir = dask_worker.local_directory
    proxy_name = 'x509up_u0'
    os.environ['X509_USER_PROXY'] = working_dir + '/' + proxy_name
    os.environ['X509_CERT_DIR']="/cvmfs/grid.cern.ch/etc/grid-security/certificates/"
    return os.environ.get("X509_USER_PROXY"), os.environ.get("X509_CERT_DIR")
```

```
[11]: client.run(set_proxy)
```

```
[11]: {'tcp://10.42.0.75:43259': ('/tmp/dask-scratch-space/worker-ee12y_1w/x509up_u0',
  '/cvmfs/grid.cern.ch/etc/grid-security/certificates/'),
 'tcp://10.42.0.76:42581': ('/tmp/dask-scratch-space/worker-lp8yzbsh/x509up_u0',
  '/cvmfs/grid.cern.ch/etc/grid-security/certificates/'),
```

Almost ready: X509 user proxy

```
[21]: def ls(dask_worker):
          import os
          working_dir = dask_worker.local_directory
          return os.listdir(working_dir)

      def clear_nodes(dask_worker):
          import os
          os.popen('rm ./*.root')
          return True
```

```
[23]: myfile_path = 'Efficiency_muon_trackerMuon_Run2018_UL_ID.json'
      client.register_plugin(UploadFile(myfile_path))
      client.run(ls)
```

```
[23]: {'tcp://10.42.2.114:35371': ['storage',
        'Efficiency_muon_trackerMuon_Run2018_UL_ID.json',
        'x509up_u0'],
       'tcp://10.42.3.144:39451': ['storage',
        'Efficiency_muon_trackerMuon_Run2018_UL_ID.json',
        'x509up_u0'],
       'tcp://10.42.5.122:38909': ['storage',
        'Efficiency_muon_trackerMuon_Run2018_UL_ID.json',
        'x509up_u0'],
       'tcp://10.42.6.150:37019': ['storage',
        'Efficiency_muon_trackerMuon_Run2018_UL_ID.json',
        'x509up_u0']}
```

- Read and write from/to the workers

- For example, we upload to the workers a json file containing some scale factors / corrections used in the analysis and check they are there

$D_s^+ \to \phi(\mu\mu)\pi^+$ analysis code: auxiliary functions

Declare custom C++ functions

[7]:
```
text_file = open("Utilities.h", "r")
data = text_file.read()

import correctionlib
correctionlib.register_pyroot_binding()

def my_initialization_function():
    ROOT.gInterpreter.Declare('{}'.format(data))
    ROOT.gInterpreter.Declare('auto corrSet = correction::CorrectionSet::from_file("Efficiency_muon_trackerMuon_Run2017_UL_ID_schemaV2.json");')
    # obtained from https://gitlab.cern.ch/cms-muonPOG/muonefficiencies/-/raw/master/Run2/UL/2017/2017_Jpsi/Efficiency_muon_trackerMuon_Run2017_UL_ID.json
    ROOT.gInterpreter.Declare('auto corr = corrSet->at("NUM_MediumID_DEN_TrackerMuons");')

ROOT.RDF.Experimental.Distributed.initialize(my_initialization_function)
```

- Utilities.h contains C++ functions used to do operations over RVec<T> objects or to perform auxiliary computations
- Optional: the correctionlib library allows for handling json files containing corrections. In this case, I am using the xPOG recommended v2 schema for muon SF provided by the Muon POG. Find some more examples here (by CAT)

Access input ntuples and define a RDF

Define chain of rootfiles to analyze

ntuples corresponding to different datasets and eras are defined in a configuration json file

```
[15]: configjson = "input.json"

      with open(configjson, "r") as f:
          config = json.loads(f.read())
```

Set here which dataset to use

```
[26]: dataset_list = ["2017B"]

      for dataset in dataset_list:
        path = "root://eosuser.cern.ch//eos/user/f/fsimone/lustre/"+config[dataset]["rootpath"]
        print(path)
        treename = config[dataset]["treename"]
        print(treename)

root://eosuser.cern.ch//eos/user/f/fsimone/lustre/SkimPhiPi_UL2017_Run2017B_ModFilter_Mini_v2/210131_221715/0000/
Tree3Mu/ntuple
```

```
[27]: #generating the list of all .root files
      chain = []
      nfiles = 0
      for dataset in dataset_list:
          files = config[dataset]["files"].split()
          for file in files:
              path = "root://eosuser.cern.ch//eos/user/f/fsimone/lustre/"+config[dataset]["rootpath"]
              chain.append(path+"/"+file)

      nfiles = len(chain)
      print(chain)
      print(nfiles)

['root://eosuser.cern.ch//eos/user/f/fsimone/lustre/SkimPhiPi_UL2017_Run2017B_ModFilter_Mini_v2/210131_221715/0000/
imPhiPi_UL2017_Run2017B_ModFilter_Mini_v2/210131_221715/0000//Tree_PhiPi_2.root', 'root://eosuser.cern.ch//eos/user
5/0000//Tree_PhiPi_3.root', 'root://eosuser.cern.ch//eos/user/f/fsimone/lustre/SkimPhiPi_UL2017_Run2017B_ModFilter_
os/user/f/fsimone/lustre/SkimPhiPi_UL2017_Run2017B_ModFilter_Mini_v2/210131_221715/0000//Tree_PhiPi_5.root', 'root:
Filter_Mini_v2/210131_221715/0000//Tree_PhiPi_6.root', 'root://eosuser.cern.ch//eos/user/f/fsimone/lustre/SkimPhiPi
'root://eosuser.cern.ch//eos/user/f/fsimone/lustre/SkimPhiPi_UL2017_Run2017B_ModFilter_Mini_v2/210131_221715/0000//
8
```

- Input.json contains a list of files for each data subera
- We use xrootd to access files on eos
- We then define our ROOT dataframe using the Distributed.Dask constructor, *connecting* it to our Dask Client

```
[28]: numWorkers= len(client.scheduler_info()['workers'])
      print("Number of workers is: {}".format(numWorkers))

      df = ROOT.RDF.Experimental.Distributed.Dask.RDataFrame(treename, chain, daskclient=client)

Number of workers is: 4
```

$D_s^+ \to \phi(\mu\mu)\pi^+$ analysis code

```
# Selections on triplets
# 2 -> 2mu+track candidate mass in (1.62-2.02)GeV
# 3 -> at least 2 track associated with PV
# 4 -> Significance of BS-SV distance in the transverse plane > 2
triplet_selection = "Triplet2_Mass>1.62 && Triplet2_Mass<2.02 && \
                     RefittedPV2_NTracks > 1 && \
                     FlightDistBS_SV_Significance > 2 "

# Events with at least one good candidate
df = df.Define("triplet_mask1", triplet_selection).Filter("ROOT::VecOps::Sum(triplet_mask1) >0")
```

```
# 5 -> Muons and track within CMS acceptance
acceptance_selection = "((abs(Mu01_Eta)<1.2 && Mu01_Pt>3.5) || (abs(Mu01_Eta)>=1.2 && abs(Mu01_Eta)<2.4 && Mu01_Pt>2.0)) &&\
                        ((abs(Mu02_Eta)<1.2 && Mu02_Pt>3.5) || (abs(Mu02_Eta)>=1.2 && abs(Mu02_Eta)<2.4 && Mu02_Pt>2.0)) &&\
                        Tr_Pt>1.2"
# Events with at least one good candidate
df = df.Define("triplet_mask2", acceptance_selection).Filter("ROOT::VecOps::Sum(triplet_mask2)>0")

# Compute dR and dZ between muons/tracks
df = df.Define("dR12", "deltaR_vec(Mu01_Eta, Mu02_Eta, Mu01_Phi, Mu02_Phi)")
df = df.Define("dR13", "deltaR_vec(Mu01_Eta, Tr_Eta, Mu01_Phi, Tr_Phi)")
df = df.Define("dR23", "deltaR_vec(Mu02_Eta, Tr_Eta, Mu02_Phi, Tr_Phi)")

# 6 -> min and max deltaR requirement
dR_selection = "dR12>DELTAR_MIN && dR13>DELTAR_MIN && dR23>DELTAR_MIN &&\
                dR12<DELTAR_MAX && dR13<DELTAR_MAX && dR23<DELTAR_MAX"
df = df.Define("triplet_mask3", dR_selection).Filter("ROOT::VecOps::Sum(triplet_mask3)>0")
```

Some steps of the analysis:

- Apply selections on branches with size nTriplet → easy!

$D_s^+ \to \phi(\mu\mu)\pi^+$ analysis code

[21]:
```python
# Find index in "Muon_" and "Track_" branches
df = df.Define("Mu01_index", "match(MuonPt, Mu01_Pt)")
df = df.Define("Mu02_index", "match(MuonPt, Mu02_Pt)")
df = df.Define("Tr_index",   "match(MuonPt, Tr_Pt)")

# 7 -> Apply Muon ID Global and Particle Flow
df = df.Define("Mu01_ID", "muon_id(Mu01_index, Muon_isGlobal && Muon_isPF)")
df = df.Define("Mu02_ID", "muon_id(Mu02_index, Muon_isGlobal && Muon_isPF)")
```

Some steps of the analysis:

- Other selections might require the matching between $D_s^+ \to \phi(\mu\mu)\pi^+$ candidates (nTriplet size) and the general Muon_ collection (nMuon size)

Utilities.h

```cpp
RVec<int> match(ROOT::VecOps::RVec<double> branch1, ROOT::VecOps::RVec<double> branch2){
//returns vector of indeces such that branch2[index]=branch1
    RVec<int> index;
    for(unsigned i = 0; i<branch1.size(); i++){
      auto idx = std::find(branch2.begin(), branch2.end(), branch1.at(i));
      if( idx != branch2.end()) index.push_back(std::distance(branch2.begin(), idx));
      else index.push_back(-99);
    }
    return index;
}
```

```cpp
RVec<bool> muon_id(RVec<int> index, RVec<bool> branch2){
//returns booleans from branch2, aligned to branch1 based on index
    RVec<bool> out;
    for(unsigned i = 0; i<index.size(); i++){
      out.push_back(branch2.at(i));
    }
    return out;
}
```

$D_s^+ \to \phi(\mu\mu)\pi^+$ analysis code

[21]:
```
# Find index in "Muon_" and "Track_" branches
df = df.Define("Mu01_index", "match(MuonPt, Mu01_Pt)")
df = df.Define("Mu02_index", "match(MuonPt, Mu02_Pt)")
df = df.Define("Tr_index",   "match(MuonPt, Tr_Pt)")

# 7 -> Apply Muon ID Global and Particle Flow
df = df.Define("Mu01_ID", "muon_id(Mu01_index, Muon_isGlobal && Muon_isPF)")
df = df.Define("Mu02_ID", "muon_id(Mu02_index, Muon_isGlobal && Muon_isPF)")
```

Some steps of the analysis:

- Other selections might require the matching between $D_s^+ \to \phi(\mu\mu)\pi^+$ candidates (nTriplet size) and the general Muon_ collection (nMuon size)

```
# 8 -> IP(track, BS) z direction < 20 cm and xy direction < 0.3 cm
df = df.Define("Tr_IPcut", "muon_id(Tr_index, (Track_dz<20 && Track_dxy<0.3) )")
df = df.Define("triplet_mask4", "Mu01_ID && Mu02_ID && Tr_IPcut").Filter("ROOT::VecOps::Sum(triplet_mask4)>0")

# 9 -> dimuon mass compatible with phi(1020)
df = df.Define("Dimu_mass", "dimu_mass(RefTrack1_Pt, RefTrack1_Eta, RefTrack1_Phi, RefTrack2_Pt, RefTrack2_Eta, RefTrack2_Phi)")
df = df.Define("triplet_mask5", "Dimu_mass>1.0 && Dimu_mass<1.04").Filter("ROOT::VecOps::Sum(triplet_mask5)>0")

# 10 -> Trigger Matching
df = df.Define("triplet_mask6", "Mu1_dRtriggerMatch_2017<0.03 && Mu2_dRtriggerMatch_2017<0.03").Filter("ROOT::VecOps::Sum(triplet_mask6)>0")

# Keep best candidate based on vertex chi2
df = df.Define("BestTriplet_index", "bestcandidate(TripletVtx2_Chi2)")
df = df.Define("BestTriplet_mass", "flattening(Triplet2_Mass, BestTriplet_index)")
```

$D_s^+ \to \phi(\mu\mu)\pi^+$ analysis code

```
[21]: # Find index in "Muon_" and "Track_" branches
df = df.Define("Mu01_index", "match(MuonPt, Mu01_Pt)")
df = df.Define("Mu02_index", "match(MuonPt, Mu02_Pt)")
df = df.Define("Tr_index",   "match(MuonPt, Tr_Pt)")

# 7 -> Apply Muon ID Global and Particle Flow
df = df.Define("Mu01_ID", "muon_id(Mu01_index, Muon_isGlobal && Muon_isPF)")
df = df.Define("Mu02_ID", "muon_id(Mu02_index, Muon_isGlobal && Muon_isPF)")
```

Some steps of the analysis:

- Other selections might require the matching between $D_s^+ \to \phi(\mu\mu)\pi^+$ candidates (nTriplet size) and the general Muon_ collection (nMuon size)

```
# 8 -> IP(track, BS) z direction < 20 cm and xy direction < 0.3 cm
df = df.Define("Tr_IPcut", "muon_id(Tr_index, (Track_dz<20 && Track_dxy<0.3) )")
df = df.Define("triplet_mask4", "Mu01_ID && Mu02_ID && Tr_IPcut").Filter("ROOT::VecOps::Sum(triplet_mask4)>0")

# 9 -> dimuon mass compatible with phi(1020)
df = df.Define("Dimu_mass", "dimu_mass(RefTrack1_Pt, RefTrack1_Eta, RefTrack1_Phi, RefTra
df = df.Define("triplet_mask5", "Dimu_mass>1.0 && Dimu_mass<1.04").Filter("ROOT::VecOps::

# 10 -> Trigger Matching
df = df.Define("triplet_mask6", "Mu1_dRtriggerMatch_2017<0.03 && Mu2_dRtriggerMatch_2017

# Keep best candidate based on vertex chi2
df = df.Define("BestTriplet_index", "bestcandidate(TripletVtx2_Chi2)")
df = df.Define("BestTriplet_mass", "flattening(Triplet2_Mass, BestTriplet_index)")
```

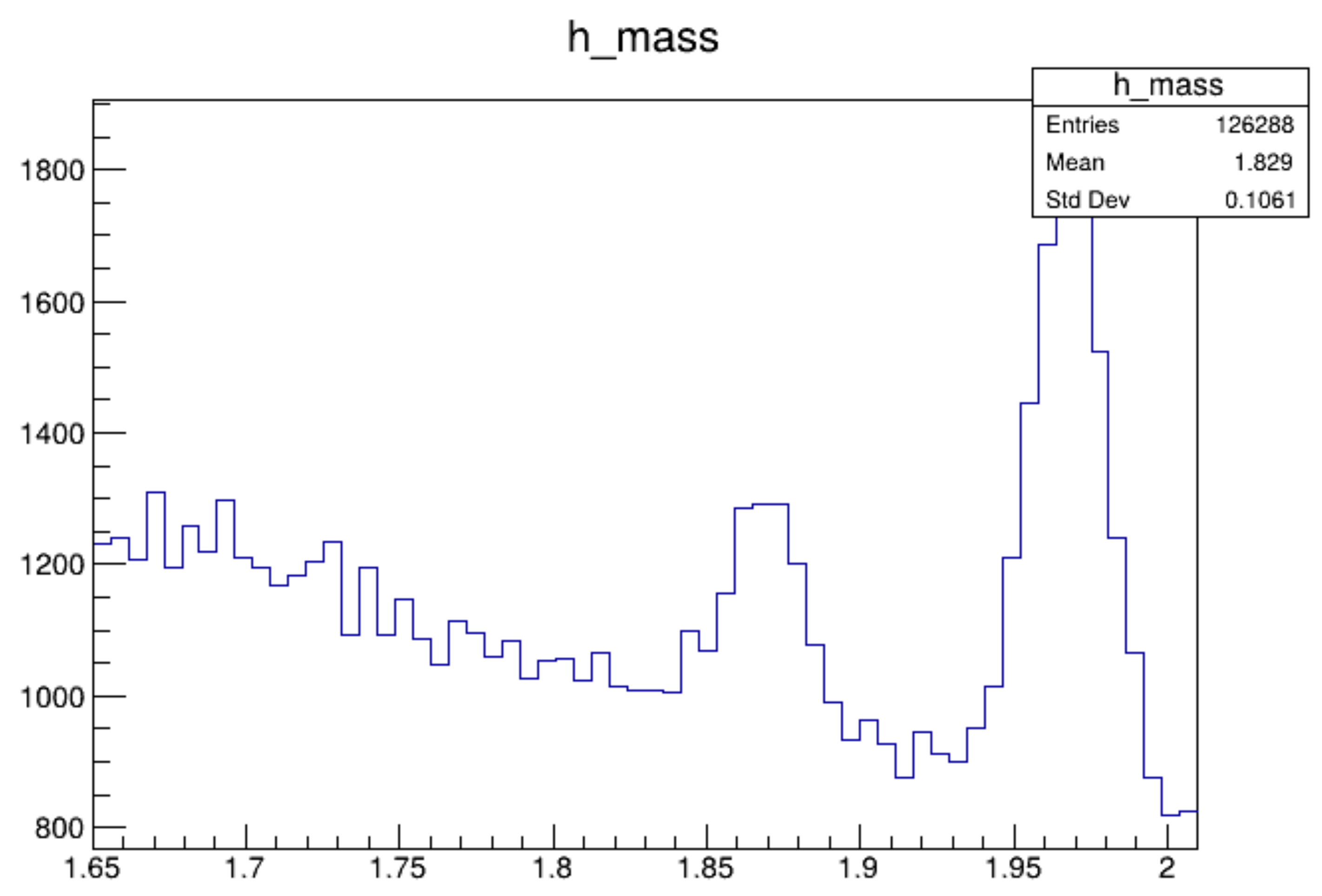
```
int bestcandidate(RVec<double> TripletChi2){
//returns index pointing to best candidate (one per event) based on Vtx fit Chi2
  auto ptr = std::min_element(TripletChi2.begin(), TripletChi2.end());
  return std::distance(TripletChi2.begin(), ptr);
}

double flattening(ROOT::VecOps::RVec<double> var, int index){
    double value = -99;
    try {
        value = var.at(index);
    } catch (const std::out_of_range& e) {
        std::cout << "Not valid index " << std::endl;
        return -99;
    }
    return value;
}
```

Utilities.h

$D_s^+ \rightarrow \phi(\mu\mu)\pi^+$ analysis code

```
[ ]: # Load muon scale factors (can be done with any set of corrections)
df = df.Define("Mu01_SF",
               ("corr->evaluate({ abs(Mu01_Eta), Mu01_Pt, \"nominal\" })"))
df = df.Define("Mu02_SF",
               ('corr->evaluate({ abs(Mu02_Eta), Mu02_Pt, \"nominal\" })'))
```

```
[22]: # Create a histogram from `x` and draw it
with performance_report(filename="my_report.html"):
  h = df.Histo1D(("h_mass", "h_mass", 62, 1.65, 2.01), "BestTriplet_mass")
  c =  ROOT.TCanvas()
  h.Draw("hist")
  c.Draw()
  # Save output for further processing: snapshot saves on workers!
  df_out = df.Snapshot("ntuple", "out.root", ["BestTriplet_mass"])
```



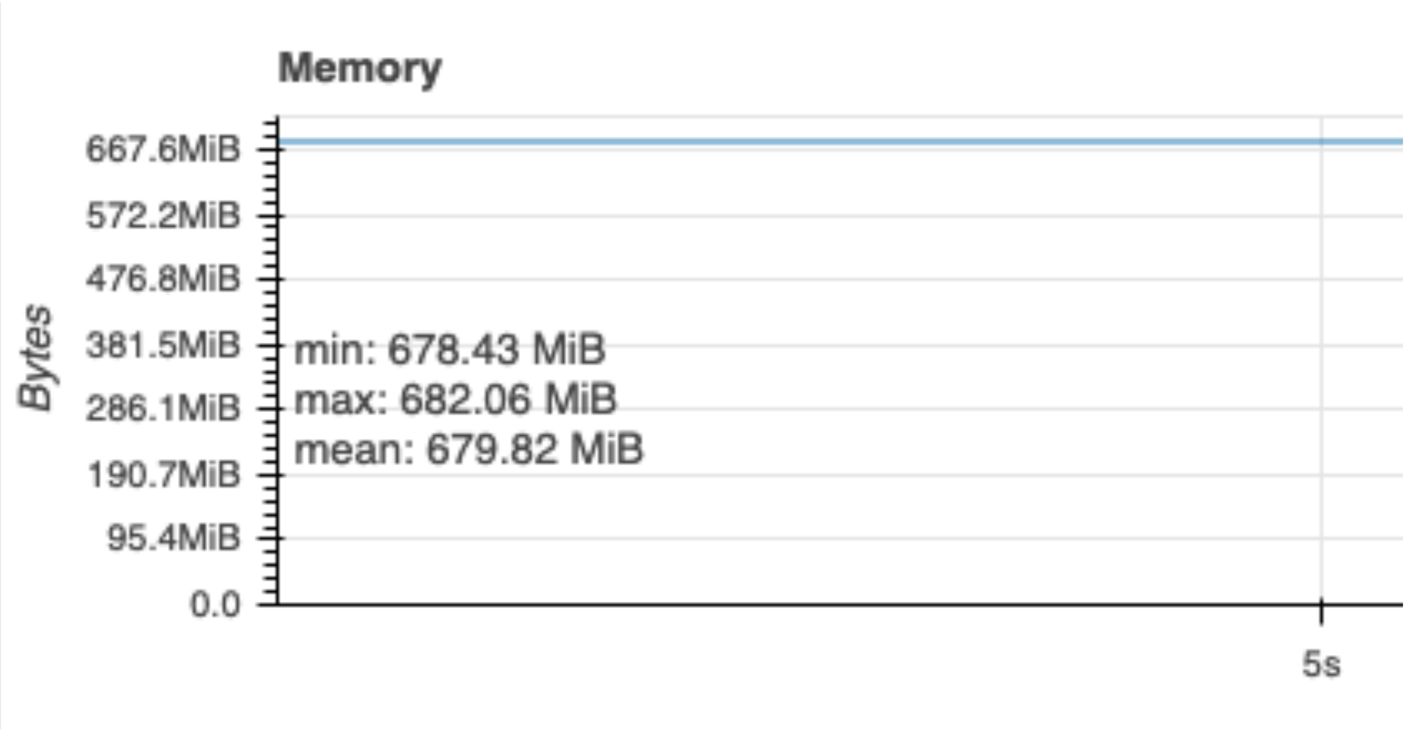
Extra:

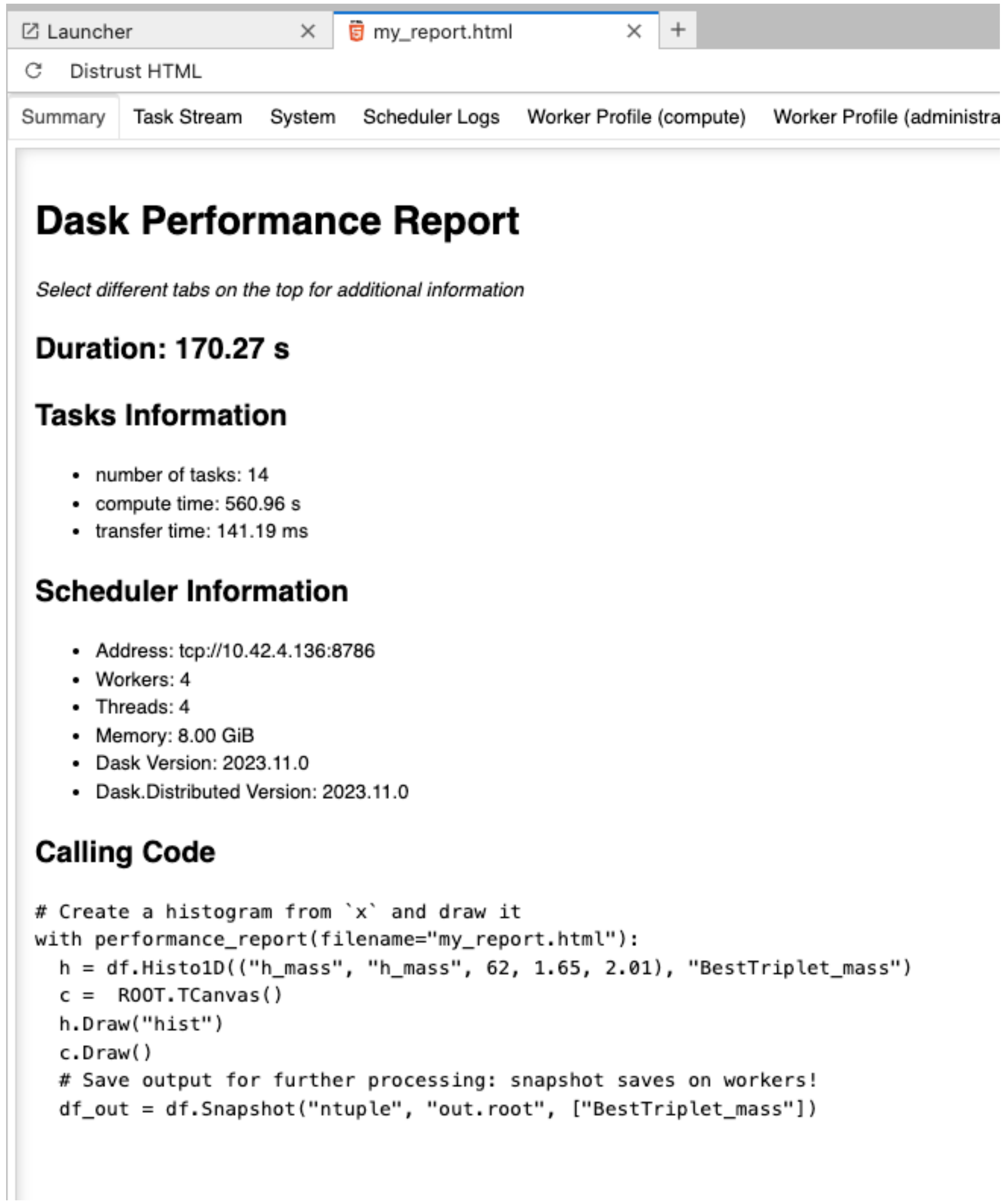
- Create columns containing Muon SF from the POG json file using correctionlib, that can be used for systematic variations

Final step:

- Drawing (or counting) out the final distribution **triggers the computation**
- You can snapshot the final dataframe for further analysis: many “out.root” files are saved in the workers

Dashboard and performance report





Launcher | my_report.html | +

Distrust HTML

Summary | Task Stream | System | Scheduler Logs | Worker Profile (compute) | Worker Profile (administra

Dask Performance Report

Select different tabs on the top for additional information

Duration: 170.27 s

Tasks Information

- number of tasks: 14
- compute time: 560.96 s
- transfer time: 141.19 ms

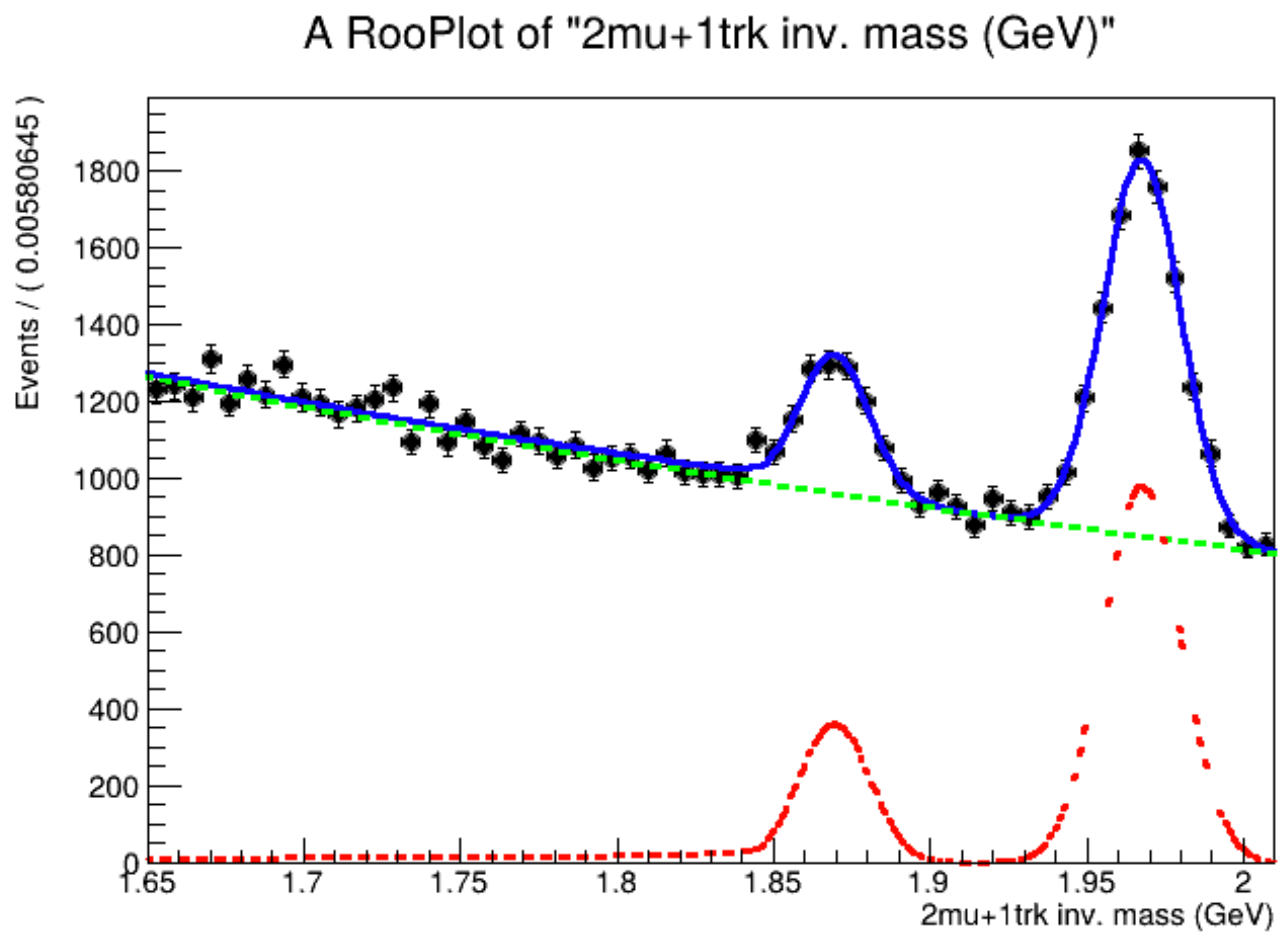
Scheduler Information

- Address: tcp://10.42.4.136:8786
- Workers: 4
- Threads: 4
- Memory: 8.00 GiB
- Dask Version: 2023.11.0
- Dask.Distributed Version: 2023.11.0

Calling Code

```
# Create a histogram from `x` and draw it
with performance_report(filename="my_report.html"):
  h = df.Histo1D(("h_mass", "h_mass", 62, 1.65, 2.01), "BestTriplet_mass")
  c =  ROOT.TCanvas()
  h.Draw("hist")
  c.Draw()
  # Save output for further processing: snapshot saves on workers!
  df_out = df.Snapshot("ntuple", "out.root", ["BestTriplet_mass"])
```

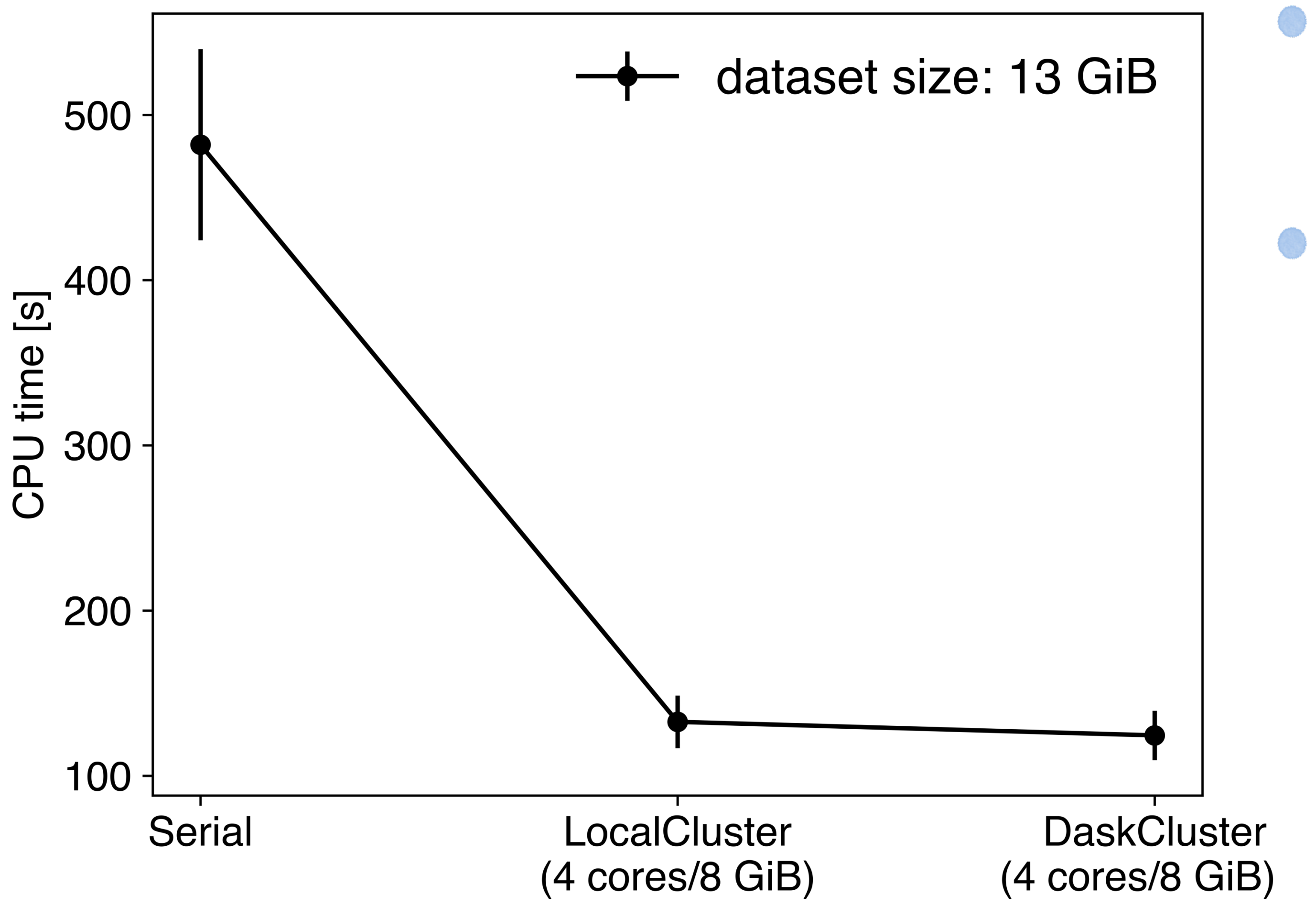
$D_s^+ \to \phi(\mu\mu)\pi^+$ final mass fit



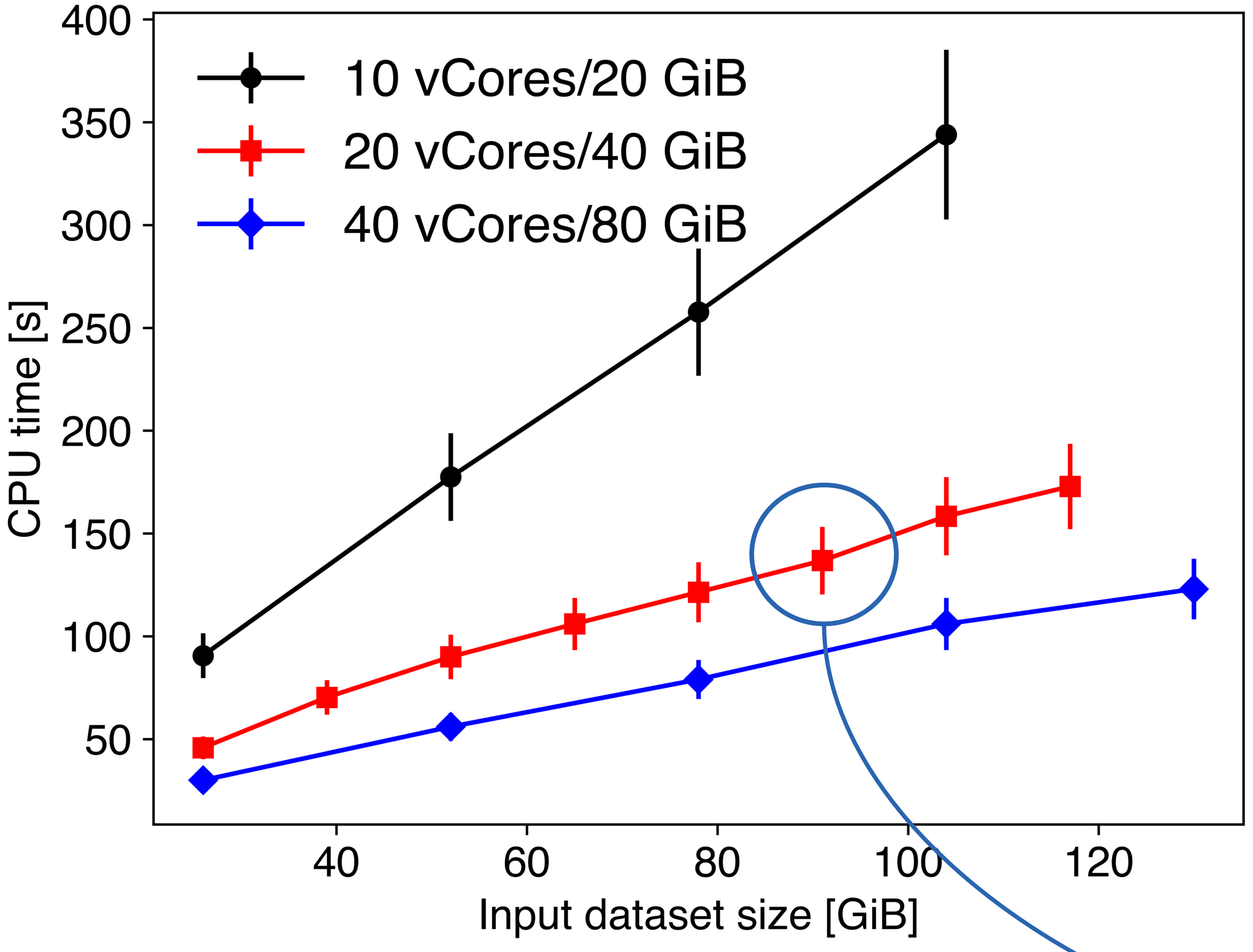

```
[24]: #fitting invariant mass :)
from ROOT import RooRealVar

x = ROOT.RooRealVar("BestTriplet_mass", "2mu+1trk inv. mass (GeV)", 1.65, 2.01)
x.setBins(62)

dh = ROOT.RooDataHist("data", "h_mass", ROOT.RooArgSet(x), Import=h.GetValue())
entries = h.GetValue().GetEntries()

#set ranges
x.setRange("R1",1.93,2.01) #main peak Ds(1.97)
x.setRange("R2",1.83,1.89) #second peak D+(1.87GeV)
x.setRange("R3",1.65,1.84) #background
x.setRange("R4",1.89,1.925) #background
x.setRange("R5",1.99,2.02) #background
x.setRange("R6",1.65,2.01) #full range

meanCB = RooRealVar("mean", "meanCB", 1.97, 1.94, 2.1)
sigmaCB1 = RooRealVar("#sigma_{CB}", "sigmaCB1", 0.02, 0.001, 0.1)
alpha1 = RooRealVar("#alpha1", "alpha1", 1.0, -10, 10)
nSigma1 = RooRealVar("n1", "n1", 1.0, 0.1, 25.0)
sig1 = ROOT.RooCBShape("sig_right", "sig_right", x, meanCB, sigmaCB1, alpha1, nSigma1)
sig1.fitTo(dh, ROOT.RooFit.Range("R1"))

meanCB2 = RooRealVar("mean2", "meanCB2", 1.87, 1.82, 1.89)
sigmaCB2 = RooRealVar("#sigma2_{CB}", "sigmaCB2", 0.05, 0.001, 0.05)
alpha2 = RooRealVar("#alpha2", "alpha2", 1.0, -10, 10)
nSigma2 = RooRealVar("n2", "n2", 1.0, 0.1, 25.0)
sig2 = ROOT.RooCBShape("sig_left", "sig_left", x, meanCB2, sigmaCB2, alpha2, nSigma2)
sig2.fitTo(dh, ROOT.RooFit.Range("R2"))

gamma = RooRealVar("#Gamma", "Gamma", -1, -2.0, -1e-2)
exp_bkg = ROOT.RooExponential("exp_bkg", "exp_bkg", x, gamma)
exp_bkg.fitTo(dh, ROOT.RooFit.Range("R3,R4,R5"))

nSig1 = RooRealVar("nSig1", "Number of signal candidates", entries*0.05, 1.0, entries)
nSig2 = RooRealVar("nSig2", "Number of signal 2 candidates", entries*0.02, 1.0, entries)
nBkg = RooRealVar("nBkg", "Bkg component", entries*0.8, 1.0, entries)
```

```
totalPDF = ROOT.RooAddPdf("totalPDF", "totalPDF", ROOT.RooArgList(sig1, sig2, exp_bkg), ROOT.RooArgList(nSig1, nSig2, nBkg))

r = totalPDF.fitTo(dh, ROOT.RooFit.Extended(ROOT.kTRUE), ROOT.RooFit.Save(ROOT.kTRUE))

c = ROOT.TCanvas()
xframe = x.frame()
#totalPDF.paramOn(xframe, ROOT.RooFit.Parameters(ROOT.RooArgSet(meanCB, meanCB2, sigmaCB1, sigmaCB2, gamma, nSig_right, nSig_left, nBkg)), ROOT
dh.plotOn(xframe)
totalPDF.plotOn(xframe)
totalPDF.plotOn(xframe, ROOT.RooFit.Components(exp_bkg), ROOT.RooFit.LineColor(ROOT.kGreen), ROOT.RooFit.LineStyle(ROOT.kDashed))
totalPDF.plotOn(xframe, ROOT.RooFit.Components(ROOT.RooArgSet(sig1, sig2)), ROOT.RooFit.LineColor(ROOT.kRed), ROOT.RooFit.LineStyle(ROOT.kDashe
xframe.Draw()
c.Draw()
c.SaveAs("mass.png")
```

Performance results (@CHEP2024)



- Significant improvement in execution time *wrt* the standard/serial approach
- The facility allows for dynamically scaling the resources, here testing the performance at fixed #cores and memory, varying the dataset size



- Stress test at high CPU and memory occupancy
- Stable performance, linearly scaling with the input dataset size
- Dataset size ~ 100 GiB is representative of ~15 /fb of Run3 data for this specific analysis

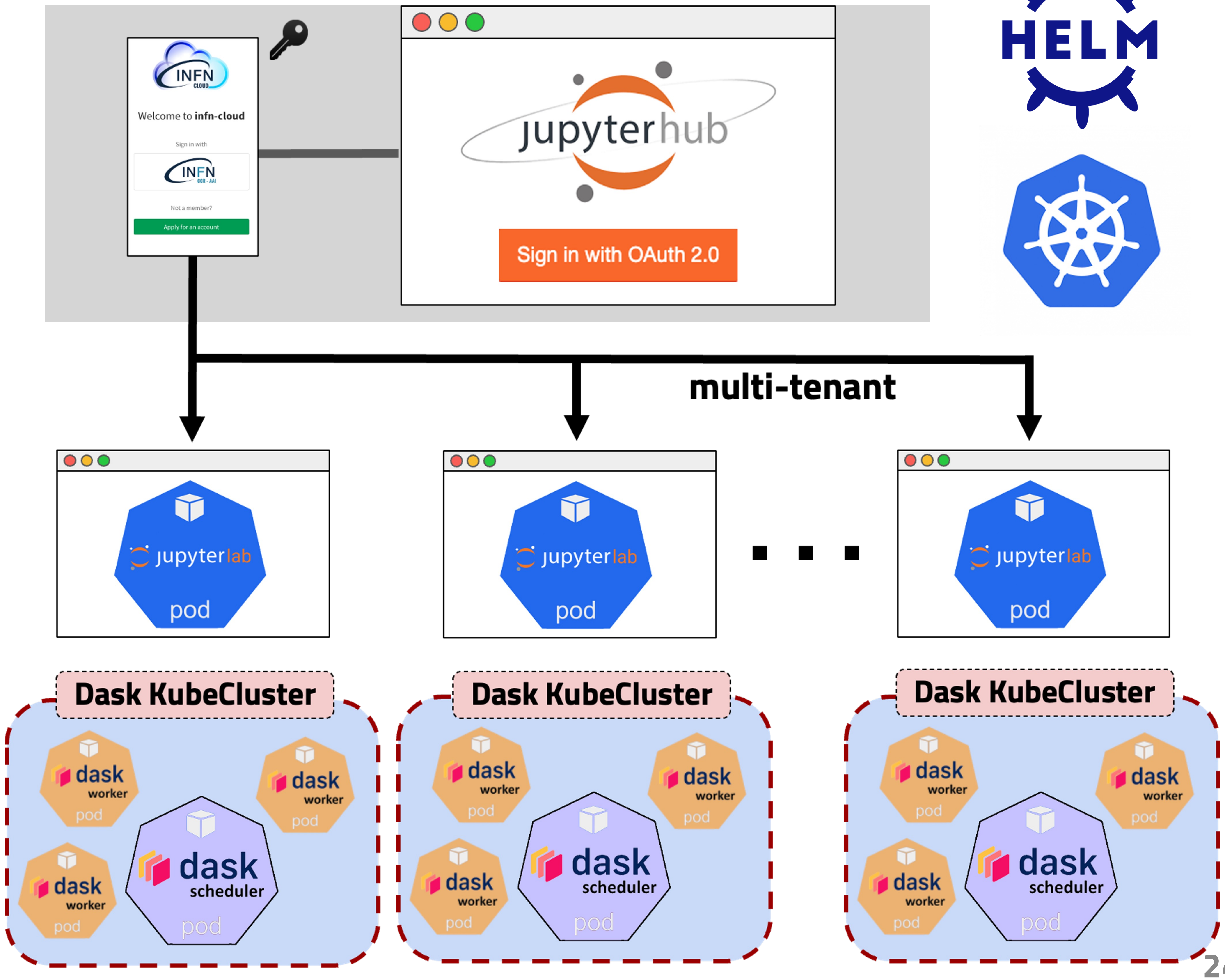
Thank you!

Back-up

High throughput data analysis platform

- After connecting to an entrypoint URL, the user reaches a Jupyterhub instance that, after authentication and authorization via INDIGO-IAM, allocates the required resources for the user's working area.

- The jupyterhub is deployed on a Kubernetes (k8s) cluster with **128 vCPUs and 258 GB**, divided into 8 nodes configured via RKE2

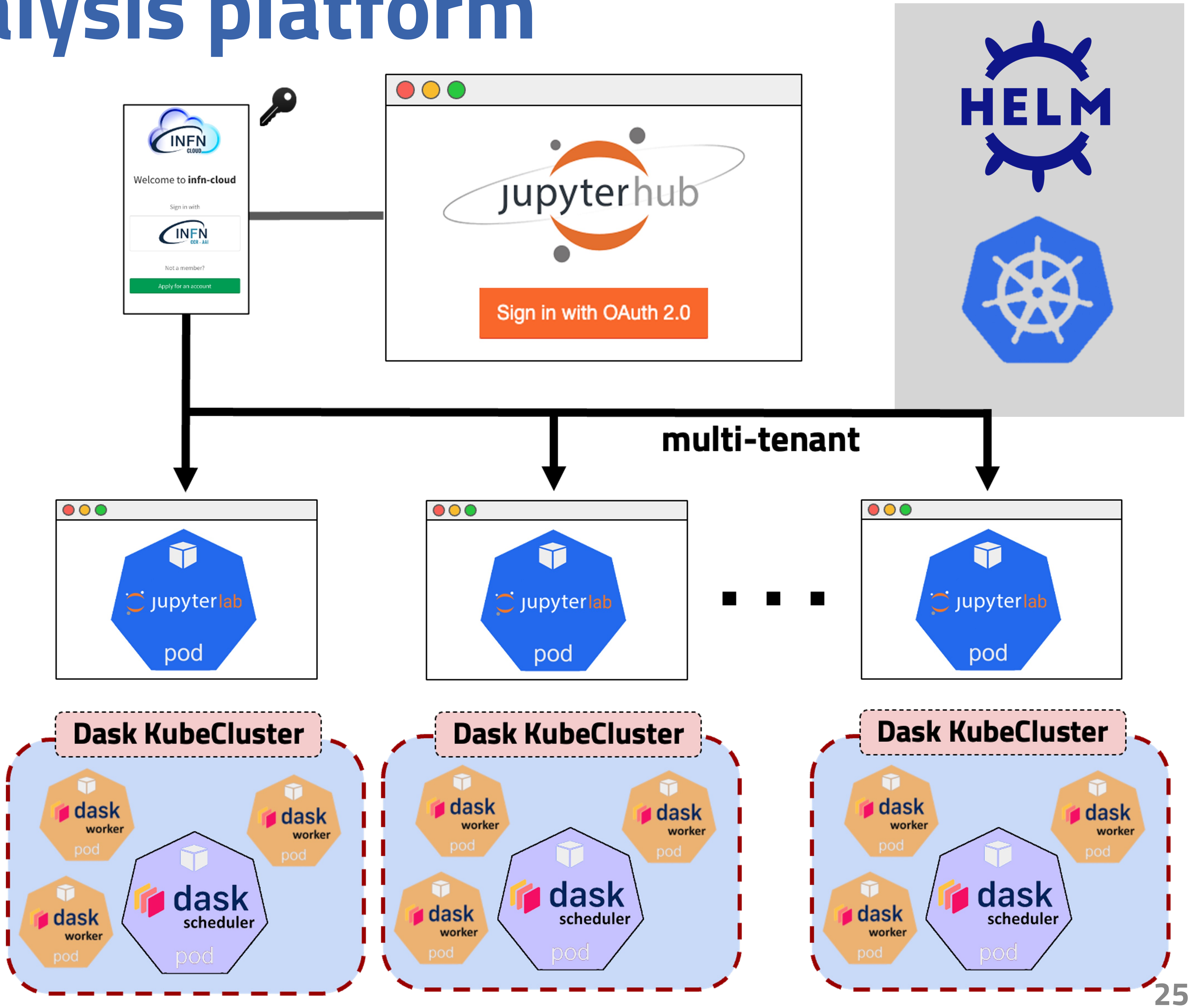


Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

High throughput data analysis platform

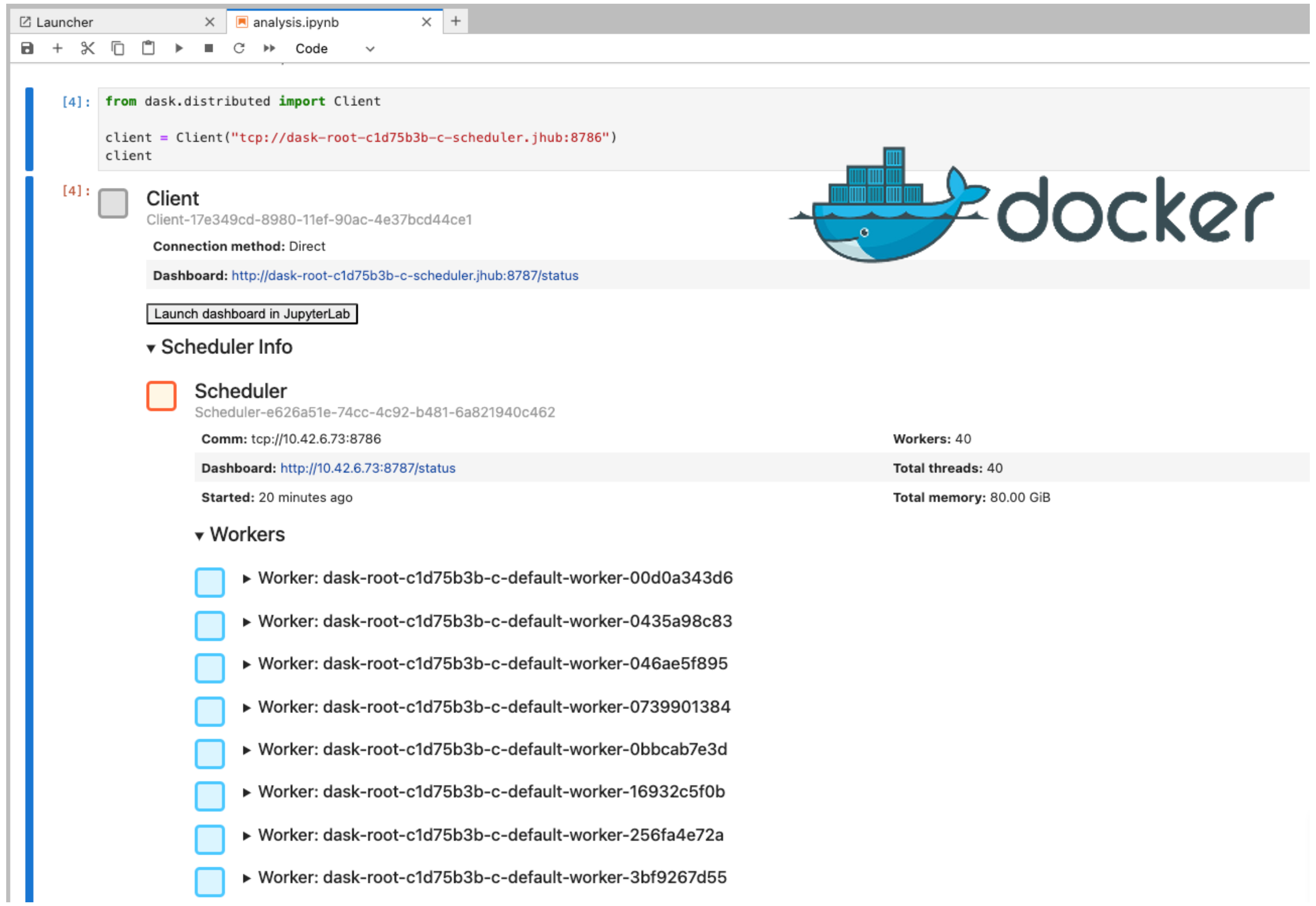
- The deployment of the Kubernetes resources is handled via HELM charts in the official [Spoke2 Jhub HELM repo]

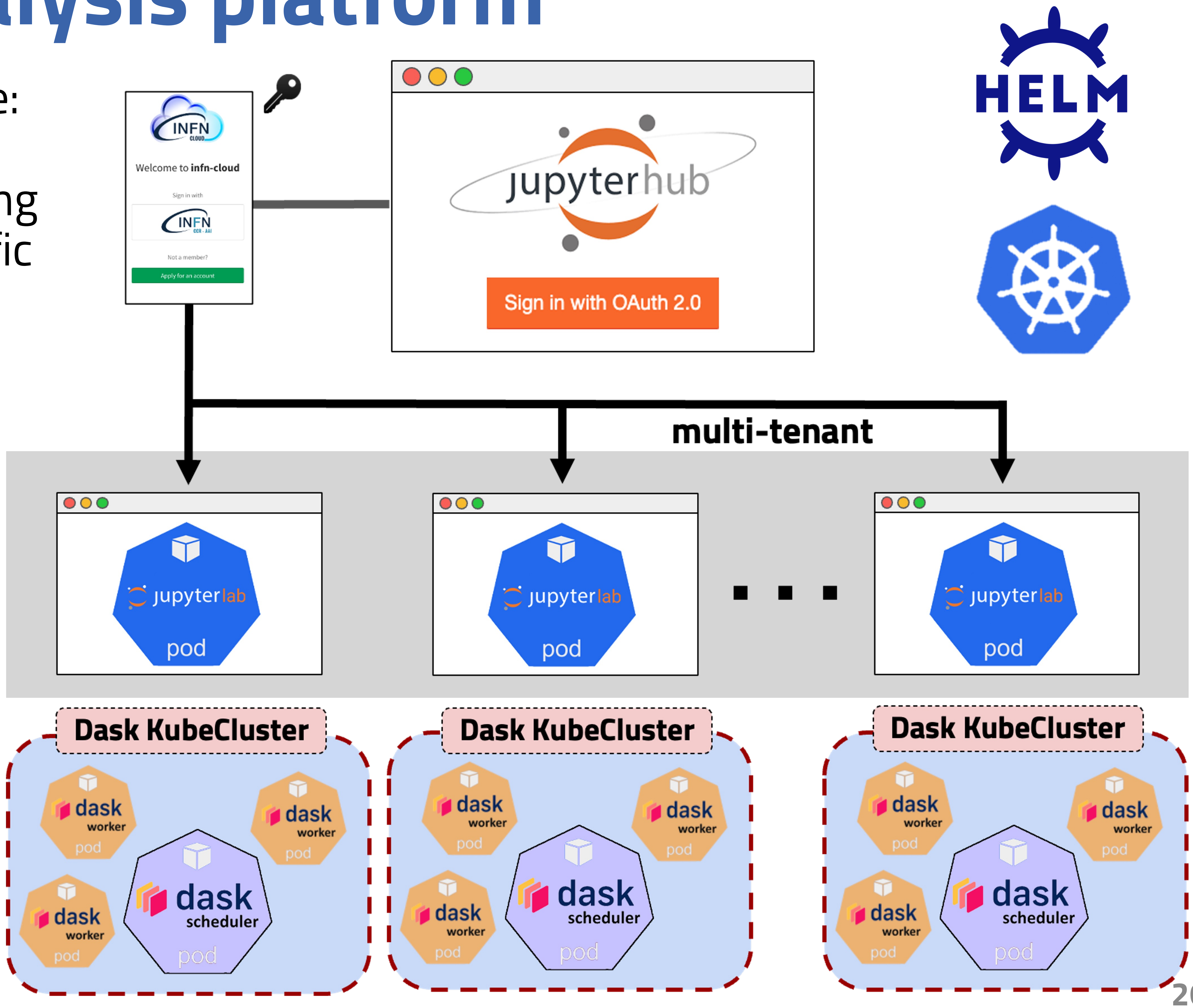
- This allows for a scalable and fault-tolerant deployment of the available resources



High throughput data analysis platform

- Jupyterlab interface is flexible and customizable:
  - Includes specific plugins (e.g. Dask)
  - Working environment highly customizable using Docker containers allowing for experiment specific software





High throughput data analysis platform

- Ideal environment for testing interactive analysis and validating new frameworks, e.g. the multi-threading features of ROOT RDataFrame
- The Dask Labextension provides a user-friendly monitoring dashboard
- More in the official docs!

Dask Dashboard 1 **Monitoring workers** 2 **Cluster map**