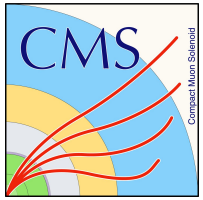


Ad-hoc meeting on distRDF
on INFN AF

September 16th, 2022

DistRDF on INFN AF: performance comparison with respect to legacy approach

D. Ciangottini[1], Daniele Spiga[1], Tommaso Tedeschi[1,2]

Thanks to ROOT/SWAN developers: V. Padulano, E. Guiraud, E. Teejedor

[1]Istituto Nazionale di Fisica Nucleare - Sezione di Perugia, Perugia, Italy
[2]Università degli Studi di Perugia, Perugia, Italy

Disclaimer/Preface

- Working on defining a meaningful architecture for a distribute analysis facility at INFN, we kicked-off an activity to evaluate the performances and the data access patterns of the upcoming analysis tools
  - In particular the comparison w.r.t. a “(typical) legacy” analysis workflow
- The first results (presented today) have been worked on with a tight collaboration with ROOT devs → distributed RDataFrame
  - Started to work on evaluating a workflow with Coffea
- Although we were confident enough to ask for this meeting, the numbers that follow have to be considered preliminary

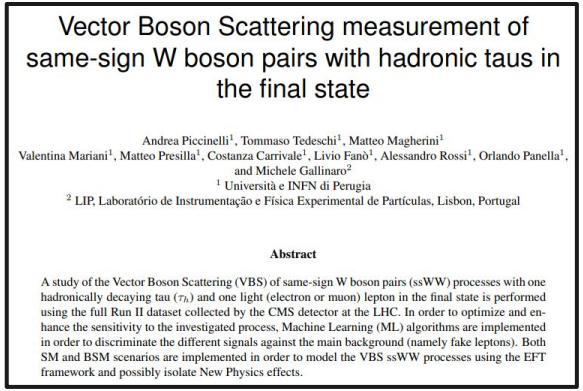
The main objective for us is to get early feedback and cross check suggestion → please ask/suggest also during the presentation, also trying to understand if and what follow-up could be interesting for CMS

Porting of the VBS analysis

VBS SSWW with a light lepton and an hadronic tau in final state: data to be processed for 2017 UL SM analysis: ca. 2TB (Data + MC)

This analysis has been ported from legacy approach (nanoAOD-tools/plain PyROOT-based) to RDataFrame in order to obtain:

- Enhanced **user experience** thanks to the modern high-level declarative interface
- Improved **efficiency** thanks to intrinsic parallelization
- A **unique tool** to be used in a single environment
- **Reduced data layers** and reduced file writing, obtained by merging analysis steps
- Much **faster and easier** definition of different selections to perform **checks**
  - (also thanks to caching)
- Capability of **distribute workflow** on different distributed back-ends (e.g. Dask -> demo on AF@INFN later)



Vector Boson Scattering measurement of same-sign W boson pairs with hadronic taus in the final state

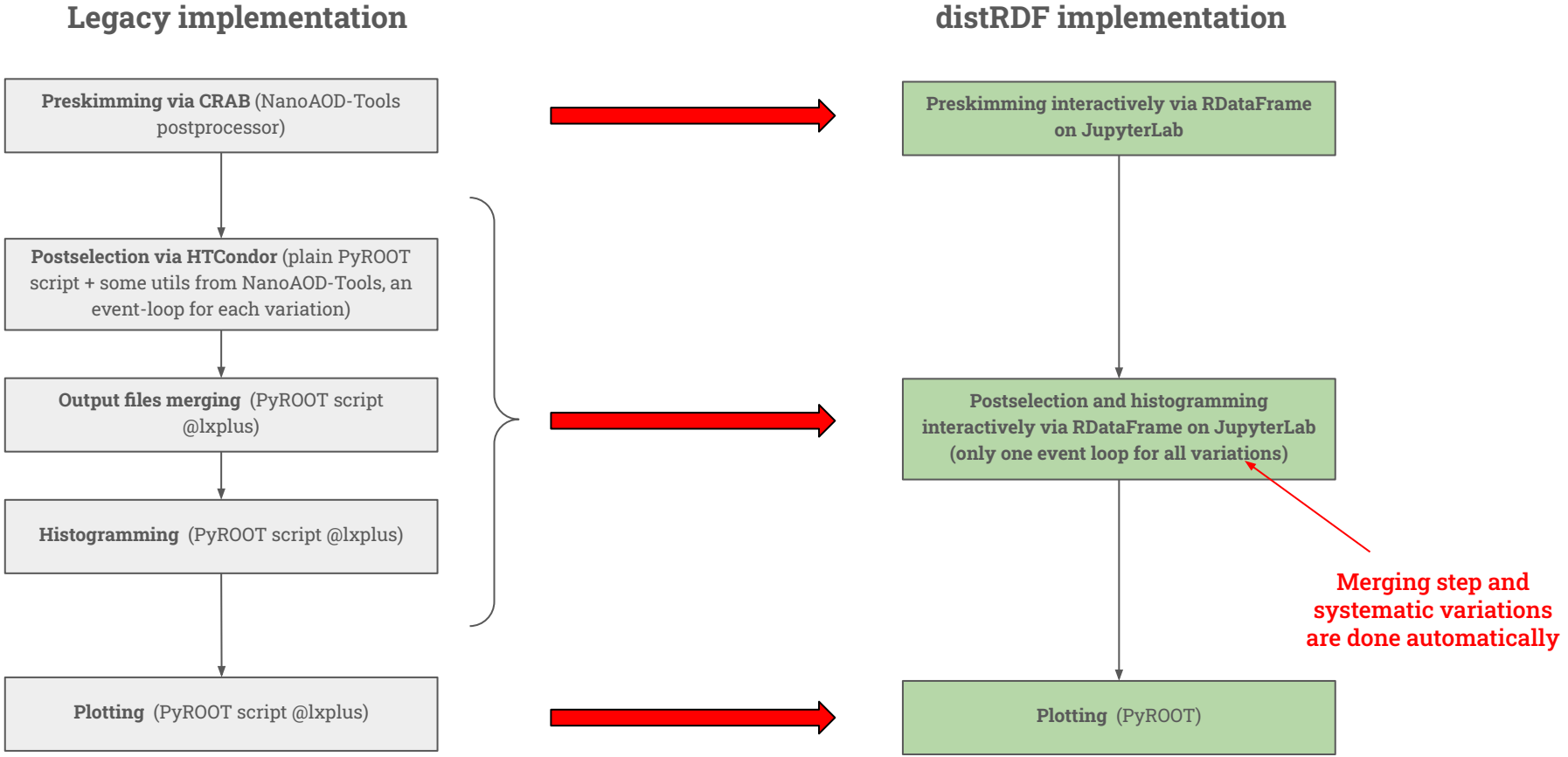
Andrea Piccinelli[1], Tommaso Tedeschi[1], Matteo Magherini[1]
Valentina Mariani[1], Matteo Presilla[1], Costanza Carrivale[1], Livio Fanò[1], Alessandro Rossi[1], Orlando Panella[1], and Michele Gallinaro[2]
[1] Università e INFN di Perugia
[2] LIP, Laboratório de Instrumentação e Física Experimental de Partículas, Lisbon, Portugal

Abstract

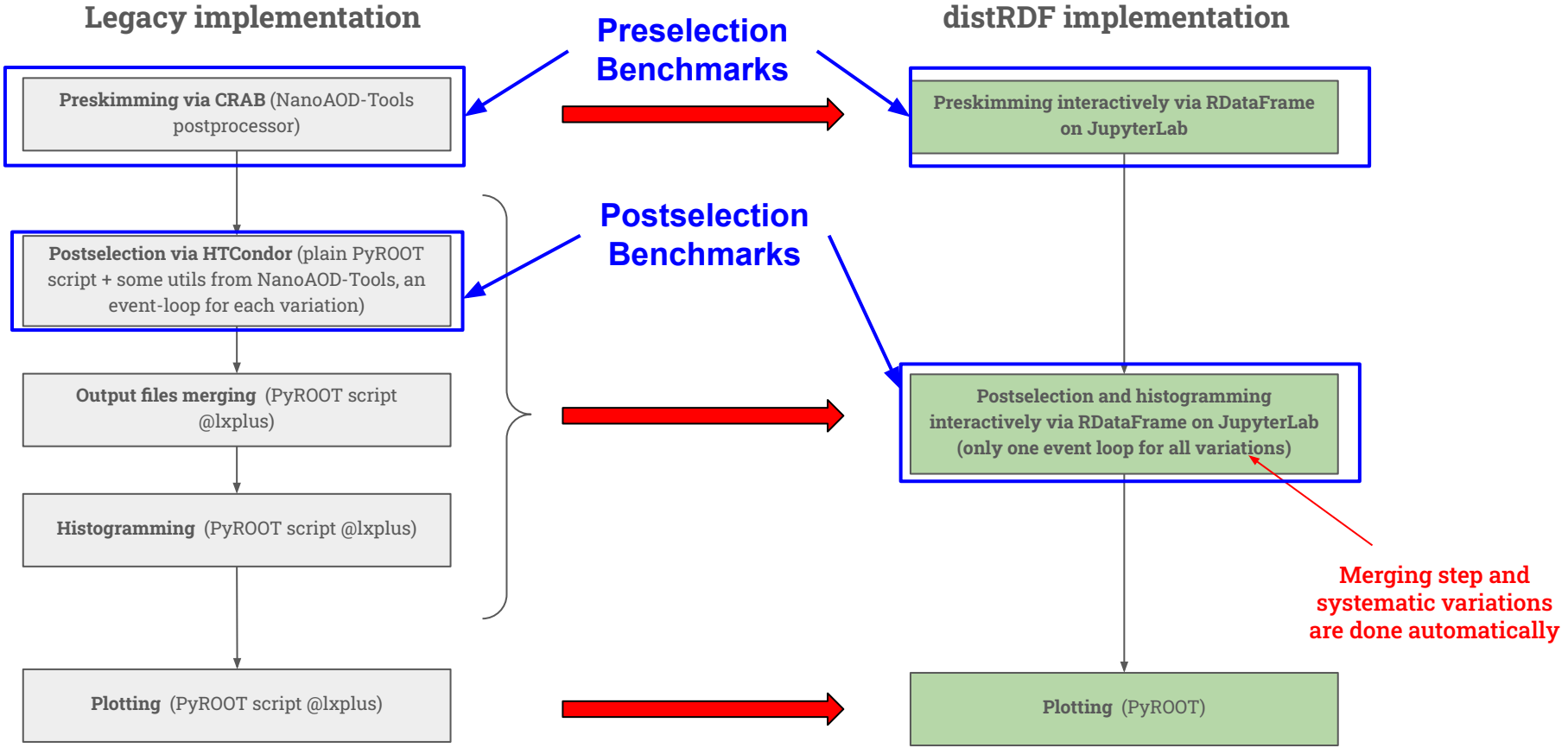
A study of the Vector Boson Scattering (VBS) of same-sign W boson pairs (ssWW) processes with one hadronically decaying tau (τ_h) and one light (electron or muon) lepton in the final state is performed using the full Run II dataset collected by the CMS detector at the LHC. In order to optimize and enhance the sensitivity to the investigated process, Machine Learning (ML) algorithms are implemented in order to discriminate the different signals against the main background (namely fake leptons). Both SM and BSM scenarios are implemented in order to model the VBS ssWW processes using the EFT framework and possibly isolate New Physics effects.

AN-2021/042

Porting of VBS analysis



Porting of VBS analysis



Metrics & working conditions

- Metrics:
  - Via site monitoring dashboard (on-node telegraf sensors into a dedicated influxDB):
    - Overall execution time
    - Network read
    - Memory occupancy
  - Via script/job:
    - Rate (events/s), both overall (considering initialization time) and event-loop-only
- Legacy and distRDF run on very same resources:
  - Legacy on the same HTCondor pool Dask is deployed on
  - Same network connection
  - Same storage
  - Very same nodes (6 nodes - 96 workers @ T2_LNL_PD)
- Legacy approach has been reviewed and optimized before this benchmark wrt how it was before:
  - Mostly in NanoAOD-tools postprocessor usage (modules ordering, branches to be read, etc..)

Results summary

Benchmark: UL2017 MC analysis (NanoAOD, 1.1 TB, 1274 files, ca. 700mln events):

- Preselection:
  - Filters on trigger, correction computation
- Postselection:
  - Proper event reconstruction:
    - 1st scenario: saving in a Tree all interesting variables (nominal):
      - RDF includes also ML inference
    - 2nd scenario: histograms of 3 relevant variables (varied):
      - RDF -> directly to histograms
      - Legacy -> only HTCondor step considered, a Tree for each systematic variation

Preselection		
	Legacy	**RDF (O2)**
Overall time	3h 40min	25min
Overall rate	1095 Hz	7306 Hz
Event-loop rate	1192 Hz	8473 Hz
Overall network read	488 GB	371 GB
Average RSS per-node	Ca. 13 GB	Ca. 17 GB

Postselection - 1st scenario		
	Legacy	**RDF**
Overall time	0.25h	0.08h
Overall rate	306 Hz	855 Hz
Event-loop rate	412 Hz	1976 Hz
Overall network read	11 GB	10 GB
Average RSS per-node	Ca. 1 GB	Ca. 15 GB

Postselection - 2nd scenario		
	Legacy	**RDF (O2)**
Overall time	1.5h	10 min
Overall rate	35 Hz	337 Hz
Event-loop rate	36 Hz	437 Hz
Overall network read	84 GB	20 GB
Average RSS per-node	Ca. 5 GB	Ca. 15 GB

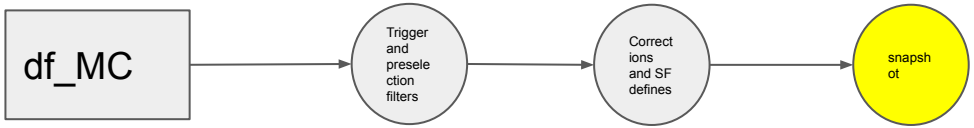
Questions/discussion

Detailed views of the 3 scenarios follow...

Results

Taking into account the analysis of UL2017 MC (1.1 TB, 656978035 events):

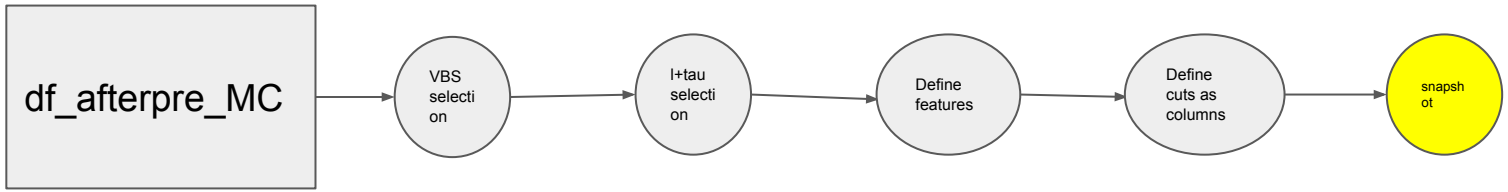
- Preselection:



Results

Taking into account the analysis of UL2017 MC (1.1 TB, 656978035 events):

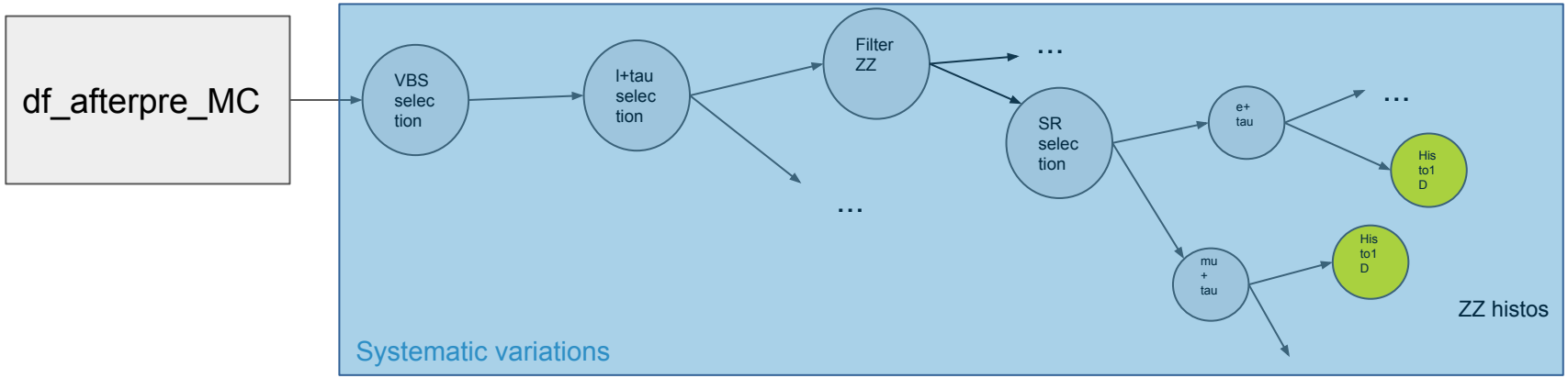
- Postselection 1st scenario:



Results

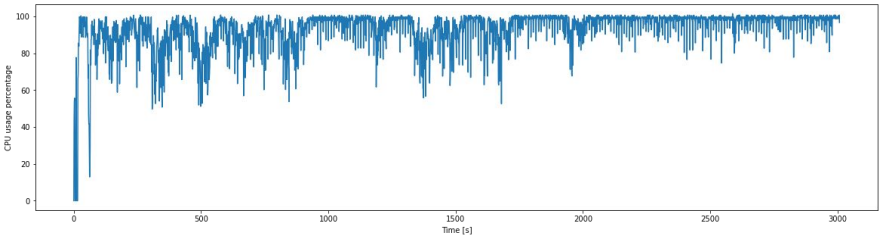
Taking into account the analysis of UL2017 MC (1.1 TB, 656978035 events):

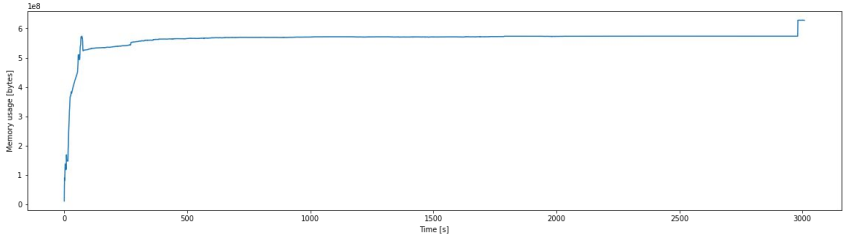
- Postselection 2nd scenario:

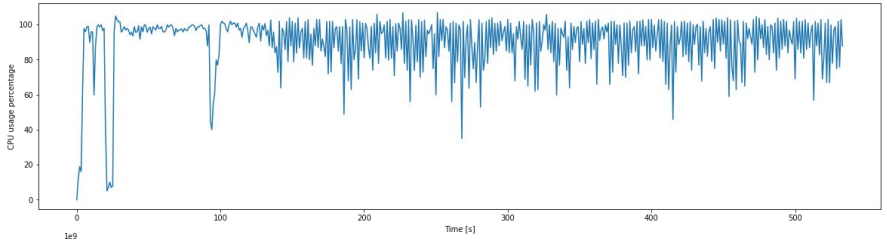


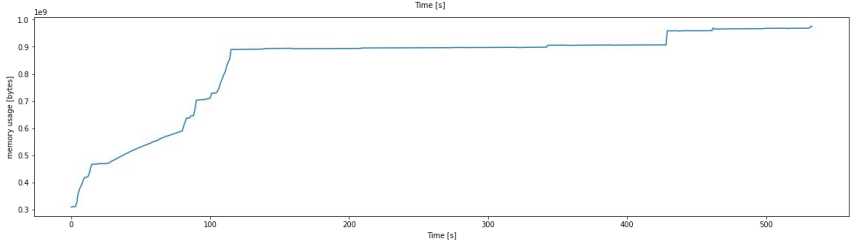
Results - task details

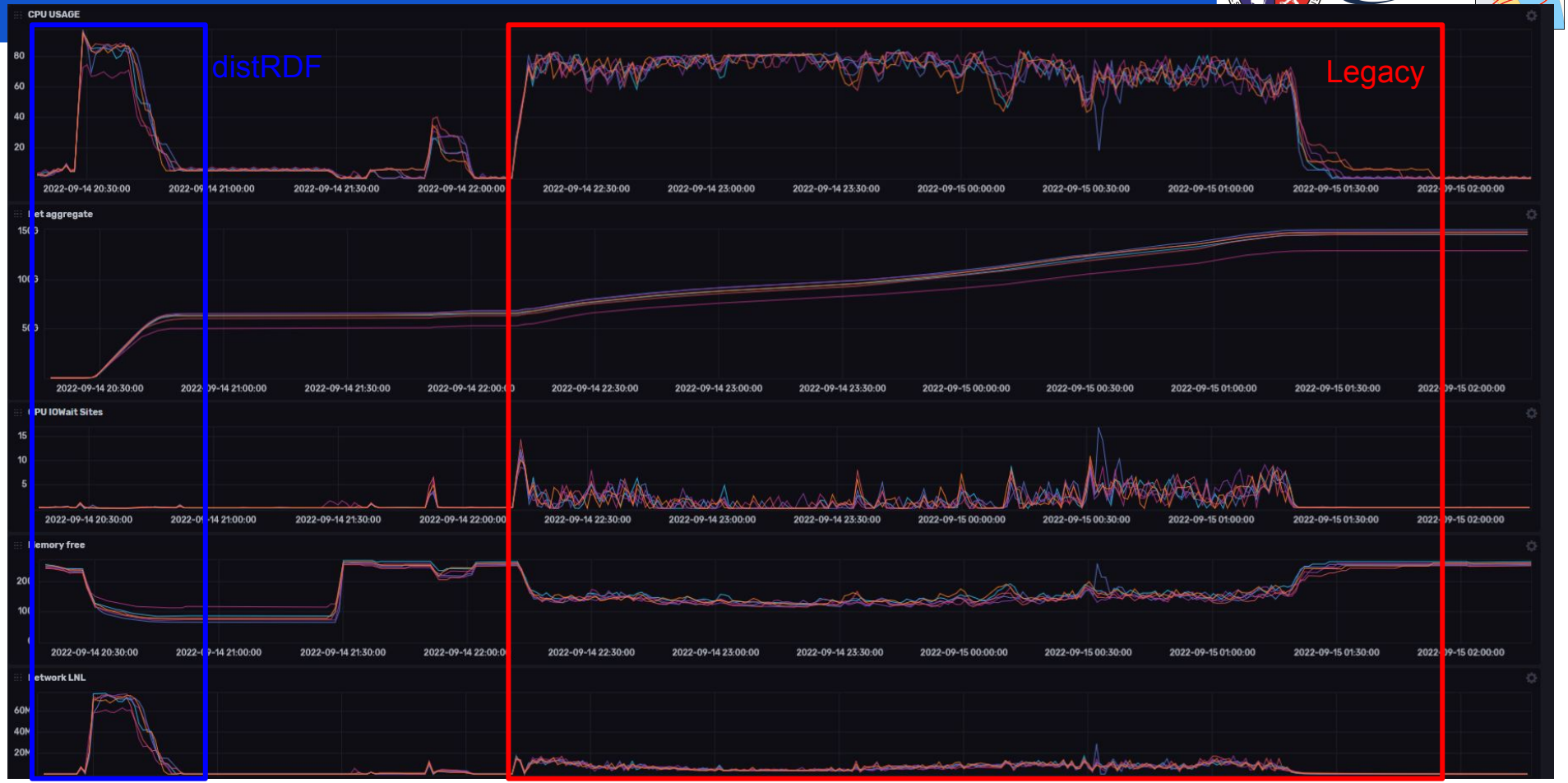
Preselection		
	Legacy	**RDF (O2)**
Overall time	3h 40min	25min
Overall rate	1095 Hz	8252 Hz
Event-loop rate	1192 Hz	9013 Hz
Overall network read	488 GB	371 GB
Average RSS per-node	Ca. 13 GB	Ca. 17 GB





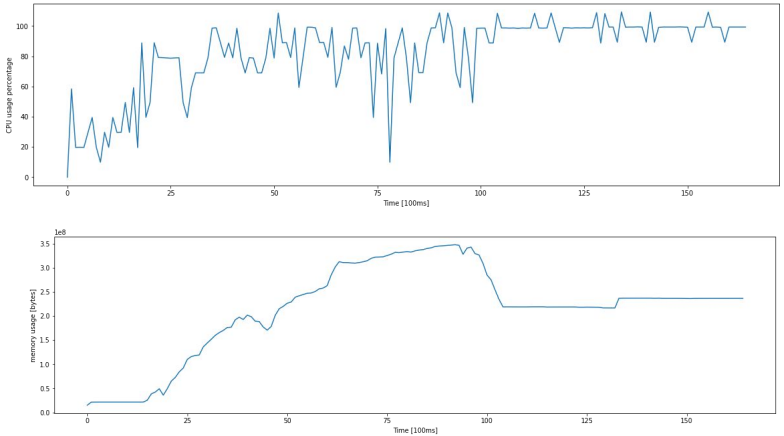


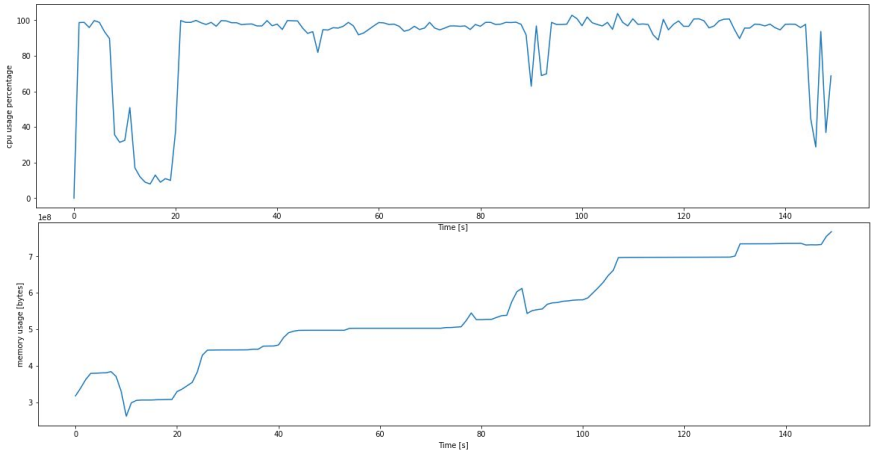



CPU USAGE
distRDF
Legacy
80
60
40
20
2022-09-14 20:30:00
2022-09-14 21:00:00
2022-09-14 21:30:00
2022-09-14 22:00:00
2022-09-14 22:30:00
2022-09-14 23:00:00
2022-09-14 23:30:00
2022-09-15 00:00:00
2022-09-15 00:30:00
2022-09-15 01:00:00
2022-09-15 01:30:00
et aggregate
PU IOWait Sites
15
10
5
emory free
etwork LNL

Results - task details

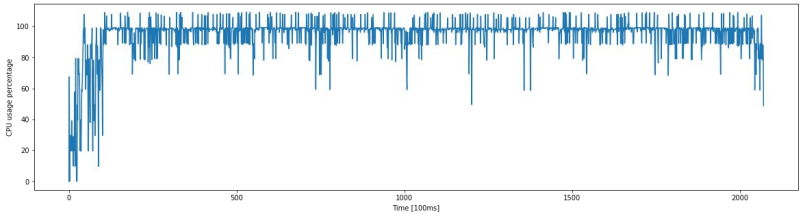
Postselection - 1st scenario		
	Legacy	**RDF**
Overall time	0.25h	0.08h
Overall rate	306 Hz	855 Hz
Event-loop rate	412 Hz	1976 Hz
Overall network read	11 GB	10 GB
Average RSS per-node	Ca. 1 GB	Ca. 15 GB

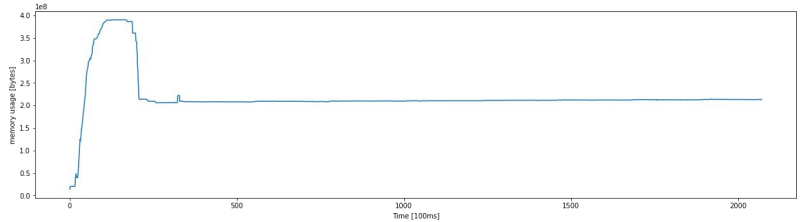


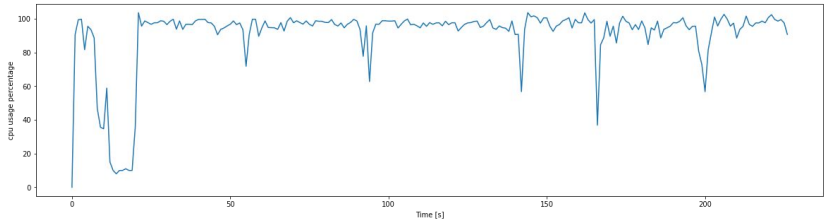


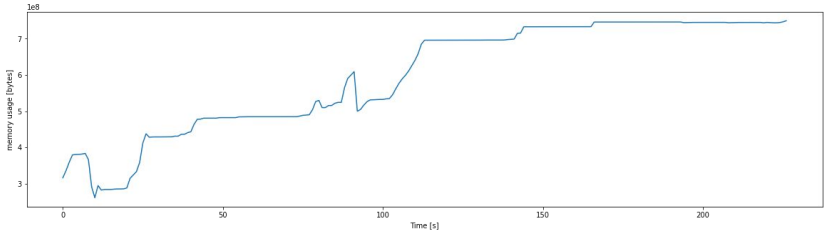
Results - task details

Postselection - 2nd scenario		
	Legacy	**RDF (O2)**
Overall time	1.5h	10 min
Overall rate	35 Hz	337 Hz
Event-loop rate	36 Hz	437 Hz
Overall network read	84 GB	20 GB
Average RSS per-node	Ca. 5 GB	Ca. 15 GB









The analysis tool

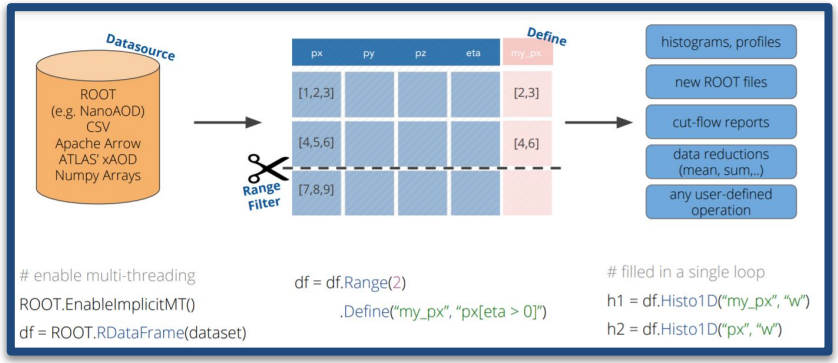
Today's use-case example is based on RDataFrame (**but the infrastructure presented before is NOT RDataFrame-specific**)

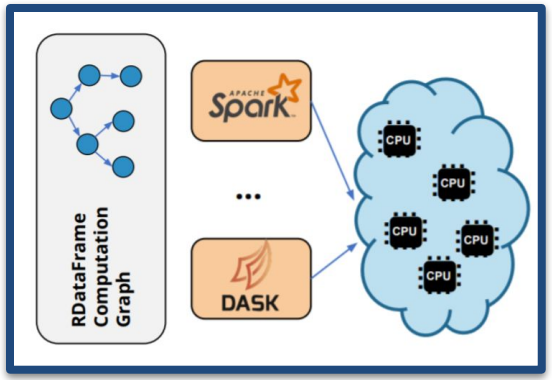
RDataFrame is ROOT's high level interface for analyses of data stored in TTree, CSV's and other data formats.

- multi-threading
- low-level optimizations (parallelization and caching).

Calculations are expressed in terms of a type-safe functional chain of actions and transformations.

Can be executed in parallel on distributed computing frameworks on a set of remote machines.





Images taken from "A Python package for distributed ROOT RDataFrame analysis", by V. Padulano, PyHEP 2021

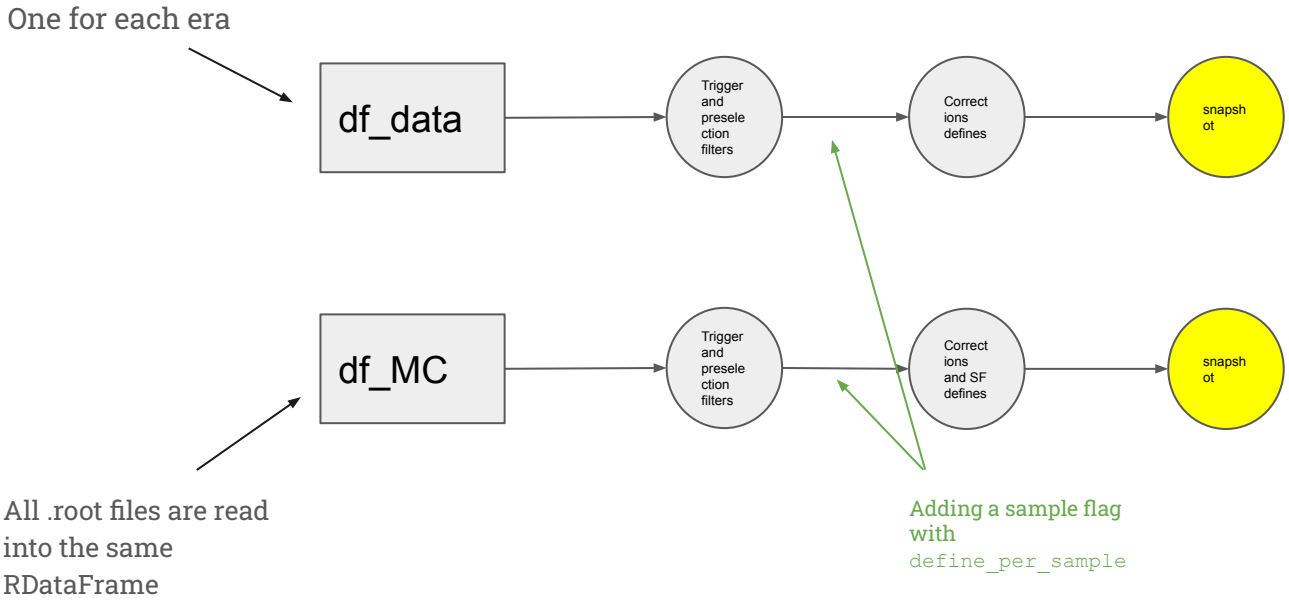
How code looks like

```
def inizialization_function():
    ROOT.gInterpreter.Declare('#include "utils_functions.h"')

df = ROOT.RDF.Experimental.Distributed.Dask.RDataFrame("Events", chain, nPartitions = N, client = client)  #define the dataframe

df_processed = df.Define("column_c", "function(column_a, column_b)")\
                 .Filter("filtering_function(column_d)", "A filter")
                 ...

# book a snapshot (i.e. a saving)-> used in preselection
opts = ROOT.RDF.RSnapshotOptions()
opts.fLazy = True
df_lazy_snapshot = df_processed.Snapshot("treeName", "fileName.root", opts)

# book an histogram -> used in postselection
df_lazy_histo = df_lazy_snapshot.Histo1D("column_a", "weights_column")

# to trigger execution
histos = df_lazy_histo.GetValue()

# to inspect data
df_saved.Display(["column_a", "column_b", "column_c"], nRows = 1).Print()
+-----+----------+----------+----------+
| Row | column_a | column_b | column_c |
+-----+----------+----------+----------+
| 0   | -1       | -1       | -1       |
+-----+----------+----------+----------+
```

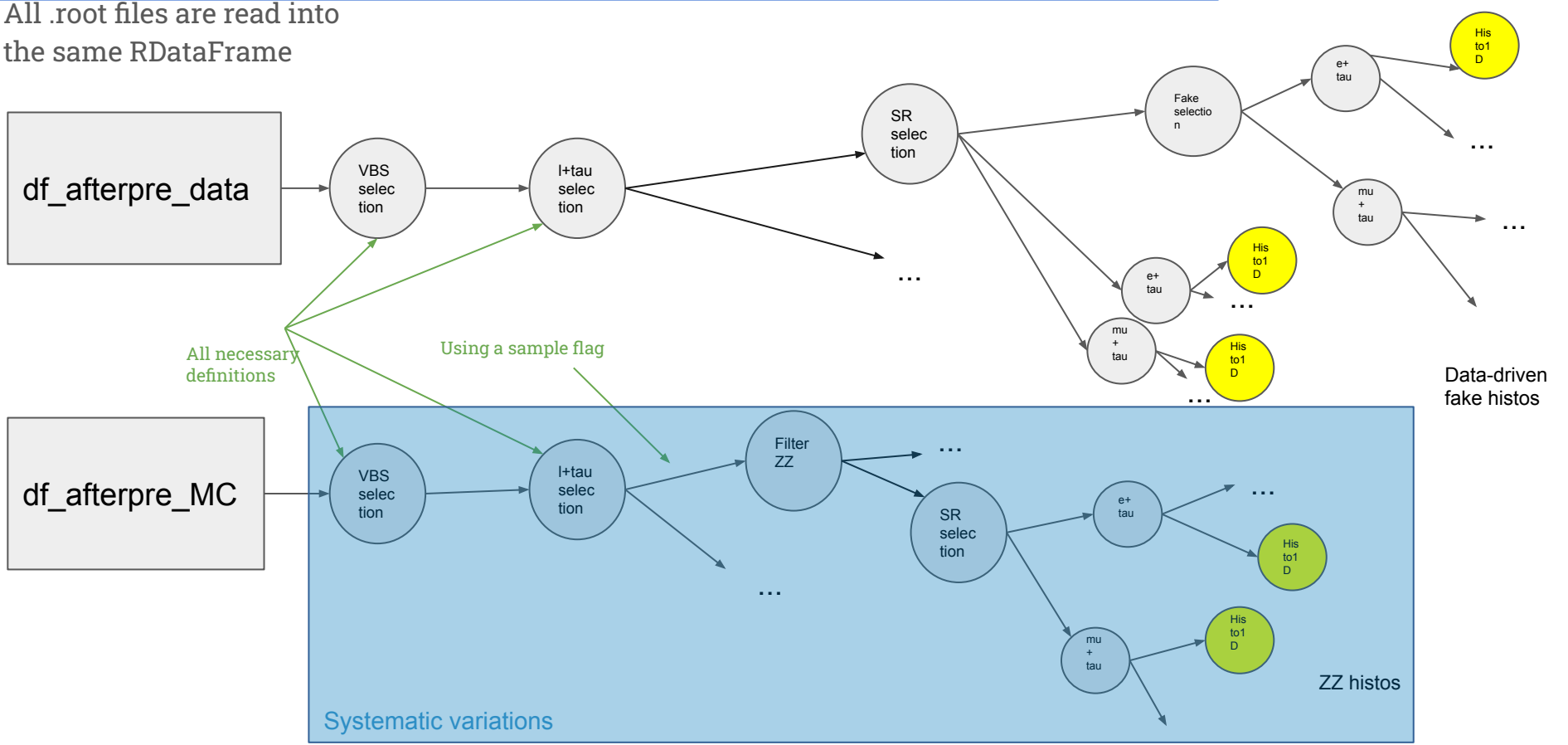
How code looks like

```
def inizialization_function():
    ROOT.gInterpreter.Declare('#include "utils_functions.h"')

df = ROOT.RDF.Experimental.Distributed.Dask.RDataFrame("Events", chain, nPartitions = N, client = client)  #define the dataframe

df_processed = df.Define("column_c", "function(column_a, column_b)")\
                 .Filter("filtering_function(column_d)", "A filter")
                 ...

# book a snapshot (i.e. a saving)-> used in preselection
opts = ROOT.RDF.RSnapshotOptions()
opts.fLazy = True
df_lazy_snapshot = df_processed.Snapshot("treeName", "fileName.root", opts)

# book an histogram -> used in postselection
df_lazy_histo = df_lazy_snapshot.Histo1D("column_a", "weights_column")

# to trigger execution
histos = df_lazy_histo.GetValue()

# to inspect data
df_saved.Display(["column_a", "column_b", "column_c"], nRows = 1).Print()
+-----+----------+----------+----------+
| Row | column_a | column_b | column_c |
+-----+----------+----------+----------+
| 0   | -1       | -1       | -1       |
+-----+----------+----------+----------+
```

C++ functions that manipulates RVec objects
Target of the porting


One for each era
df_data
Trigger and presele ction filters
Correct ions defines
snapsh ot
df_MC
Trigger and presele ction filters
Correct ions and SF defines
snapsh ot
All .root files are read into the same RDataFrame
Adding a sample flag with
define_per_sample

All .root files are read into the same RDataFrame



Corrections and SFs

PRESELECTION

- PUWeights
- PFCorrections
- bTagSF
- LeptonSF
- Rochester Corrections

→ Ported respective nanoAOD-tools modules in C++

- JMECorrections

→ External C++ RDF-friendly library: https://gitlab.cern.ch/cp3-cms/CMSJMECalculators/-/tree/main

All necessary files are .root, .csv and .txt files read or downloaded locally on the fly via https protocol

Selection

POSTSELECTION

- Jet selection
- Lepton selection and veto
- Tau selection and veto

→ C++ functions

- TauSF
- TES
- FES

→ Ported TauIDSFTool in C++: https://github.com/anpicci/nanoAOD-tools/blob/VBS_PG/python/postprocessing/TauIDSFTool.py

Final output variables

POSTSELECTION

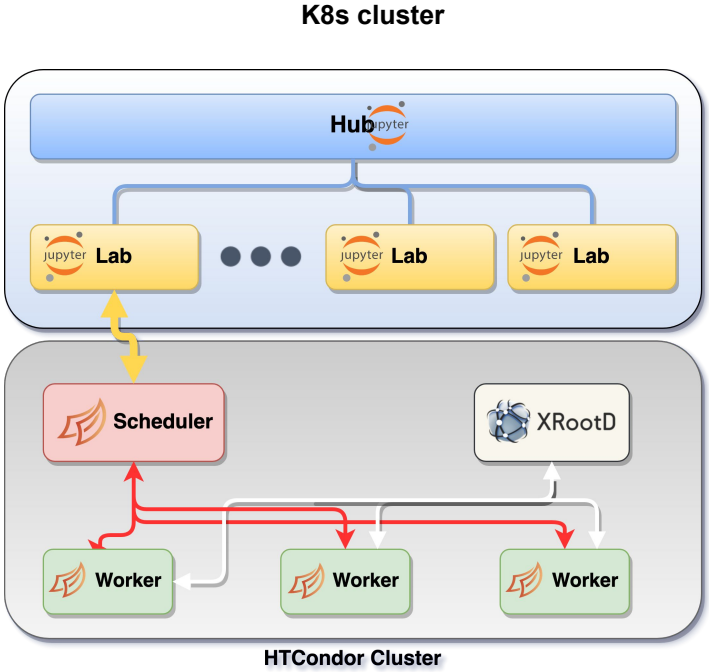
- Every “composite variable” → C++ function for each one
- XGBoost classifier → Externally trained model converted in a format serializable (https://root.cern/doc/master/tmva101__Training_8py.html) and read with the fast tree inference engine offered by TMVA (https://root.cern/doc/master/tmva103__Application_8C.html)
- DNN (trained with scikit-learn scaler + Keras on top of Tensorflow) → Model exported as tensorflow pb model and scaler exported with the usage of https://pypi.org/project/scikinC/ Then inference is done via Tensorflow C++ API (cppflow)

The infrastructure

Three building blocks:

- JupyterHub (JHub) and JupyterLab (JLab) to manage the user-facing part of the infrastructure:
  - This is not exclusive, also accessible via standard UI (“a la batch”)
- DASK to introduce the scaling over a batch system
- XRootD as data access protocol toward AAA:
  - Usage of caching layers

See backup for further implementation details



In the most general case this could be a different site

Infrastructure details