Wmass analysis framework

Valerio Bertacchi, Lorenzo Bianchini, Elisabetta Manca,
Gigi Rolandi, Suvankar Roy Chowdhury

Milano-Pisa PRIN meeting
05/10/2021

Introduction

- Input of the analysis
- A brief overview of the offline analysis framework developed for the analysis.
- Challenges : complexity and volume.
- Discussion of concepts, not much technical details.

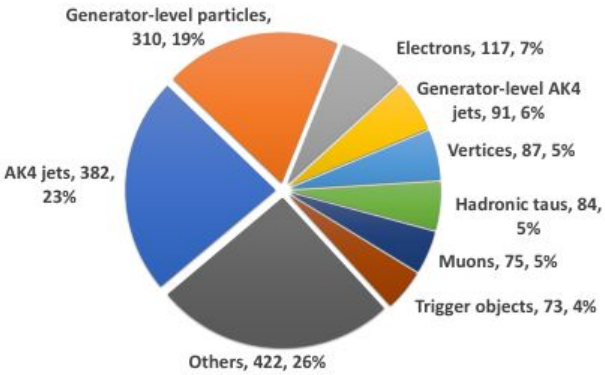
Input data samples

- Input data-tier - NANOAOD
- Contents of the data-tier
- Developments in CMS software framework

NANOAOD II

- The data model employed by CMS has a tiered structure, where each subsequent format contains a more compact summary of the event data than its predecessor. “RAW” event size is ~1 MB
- Our analysis uses the “**NANOAOD**” data-tier.
  - Demand came from ever increasing demand to store CMS data and MC samples as LHC continues to operate
  - Contents are flat ROOT trees one each for Event, Run, Lumi and some meta data
  - For every event, the properties of **high-level objects(e.g. muons)** are stored as arrays in the branches.
  - Size per event is 1-2 kB
- The 17 billion events recorded in Run 2 data-taking period, plus accompanying 60 billion events produced by MC simulations fit in just under 140 TB!

Branch name	Type	Data type	Function	Example
`nObject`	Scalar	Unsigned integer	Number of objects in collection `Object`	`nMuon`, `nJet`, `nGenPart`
`Object_var[i]`	Array	Any	Attribute `var` of the *i*-th object in collection `Object`	`Muon_pt[i]`, `Jet_mass[i]`, `GenPart_pdgId[i]`
`Object_otherIdx[i]`	Array	Signed integer	Index of the object from collection `Other` if it is linked to the *i*-th object from collection `Object`, and −1 otherwise	`Muon_jetIdx[i]`, `Muon_genPartIdx[i]`



NANOAOD II

- The event content of the NANOAOD lacked some of the event content information desired by our analysis
- Our group spent considerable time and effort in finalizing the correct event content in the version of the NANOAOD to be used
  - Store kinematics of generator level leptons at higher floating point precision
  - Finalize the missing muon properties
  - Optimize storage of Generator Weight sets - the biggest challenge.
- All the necessary changes were done in "**CMSSW**" - the cms software framework
- With these changes, we can run every aspect of the analysis starting from the NANOAOD format.
- Beneficial to other precision measurements
- For our analysis, we use CMS the **"SingleMuon"** dataset
  - Contains at least one muon above certain p_T threshold(our case 24 GeV)
- MC samples include the WJets, Drell-Yan, Single Top, Semi-leptonic decays of TTbar.
- Total size for 2016 data and MC ~ 10TB

Offline software

- Requirements
- Choice of underlying tool - RDataFrame
- Analysis graph
- Framework modules
- Framework milestones
- Thinking in higher dimensions
- Performance

Requirements of offline software - I

- Our wishlist - **fast, flexible and robust framework** w.r.t the complexity of the analysis.
- What we are dealing with?
  - Let's take numbers from 2016 only
  - ~10^8 events
  - ~400 nominal histograms
  - ~30K variations of the histograms
  - High dimensionality of histograms
- Different file types
  - Input as root files
  - Meta data(samples, x-sec) as text, json file etc.

Requirements of offline software - I

- Our wishlist - **fast, flexible and robust framework** w.r.t the complexity of the analysis.
- Complexity in terms of histos and their Variations

4D templates (η,pT,mT,iso) for bkg
two fakes processes: one low mT and one high mT
6D templates (η,pT,mT,iso,y,qT) for signal

Variations

Type	N variations
SFactors	8
LHEScale	6
PDFHessian	60
alpha-s	2
Mu p_T variations*	2
JME variations	4

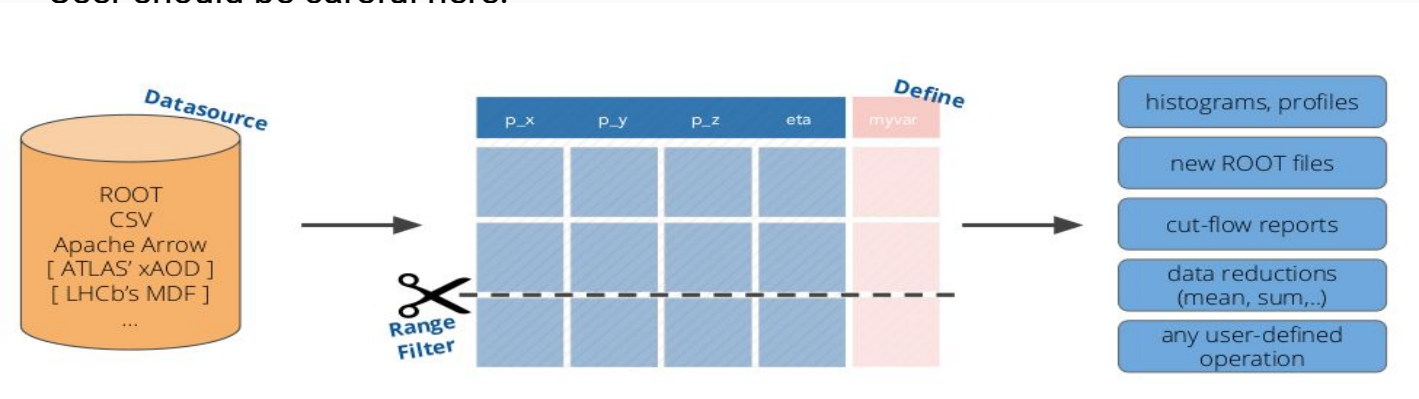
Requirements of offline software - II

- ***Built an analysis*** framework based on **RDataFrame**.
  - https://github.com/emanca/wproperties-analysis

- What RDataFrame offers?
  - Simple but powerful tool to analyse data with modern C++, python.
  - Parallelizable - transparent MT and supported for multi-core machines.
  - Easy to express dependencies on different objects.
  - **Graph style approach**, optimized event loop.
  - Interfaces to different types of data sources
  - User writes analysis - ROOT takes care of optimization.

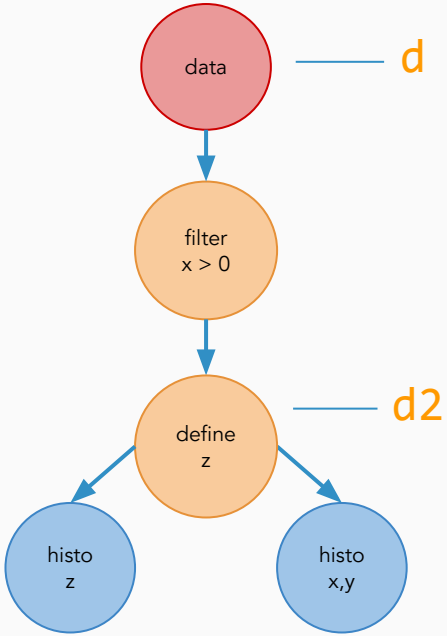
RDataFrame overview

- Three simple steps for an user.
- **Build** a DataFrame object from a source
  - ROOT tree, csv etc.
  - Non-ROOT data can also be used via RDataSource.
- Transformations on the input.
  - **Define**: new columns from existing columns.
  - **Filter**: create ranges of events by applying cuts
- **Apply actions** on the transformed data.
  - e.g., Histogramming
- Event loop is triggered only when the full analysis has been set up and **an first access** to a result has been made.
  - User should be careful here.



Thinking in terms of Graphs

```
// d2 is a new data-frame, a transformed version of d
auto d2 = d.Filter("x > 0")
           .Define("z", "x*x + y*y");

// make multiple histograms out of it
auto hz = d2.Histo1D("z");
auto hxy = d2.Histo2D("x","y");
```



- More intuitive way of visualizing an analysis workflow.

Pisa Framework

- The framework is divided into several smaller packages dedicated to execute certain tasks
  - Core package: https://github.com/emanca/RDFprocessor.git
  - Analysis package: https://github.com/emanca/wproperties-analysis
- Each package contains several modules(or code where algorithm are implemented)
- All the types of modules are written mostly written in C++. Some are written in python
  - Compiled with g++ and dictionaries are generated with reflex enabling them to be loaded in pyroot
- User API is written in python.
  - Loads information on samples, systematic variations, selections etc from json, python dictionaries.
  - Accepts command line inputs.
  - Builds the analysis graph by defining the dependence of various modules.
  - Launches jobs in Multi threading + multi-processing mode for all samples.

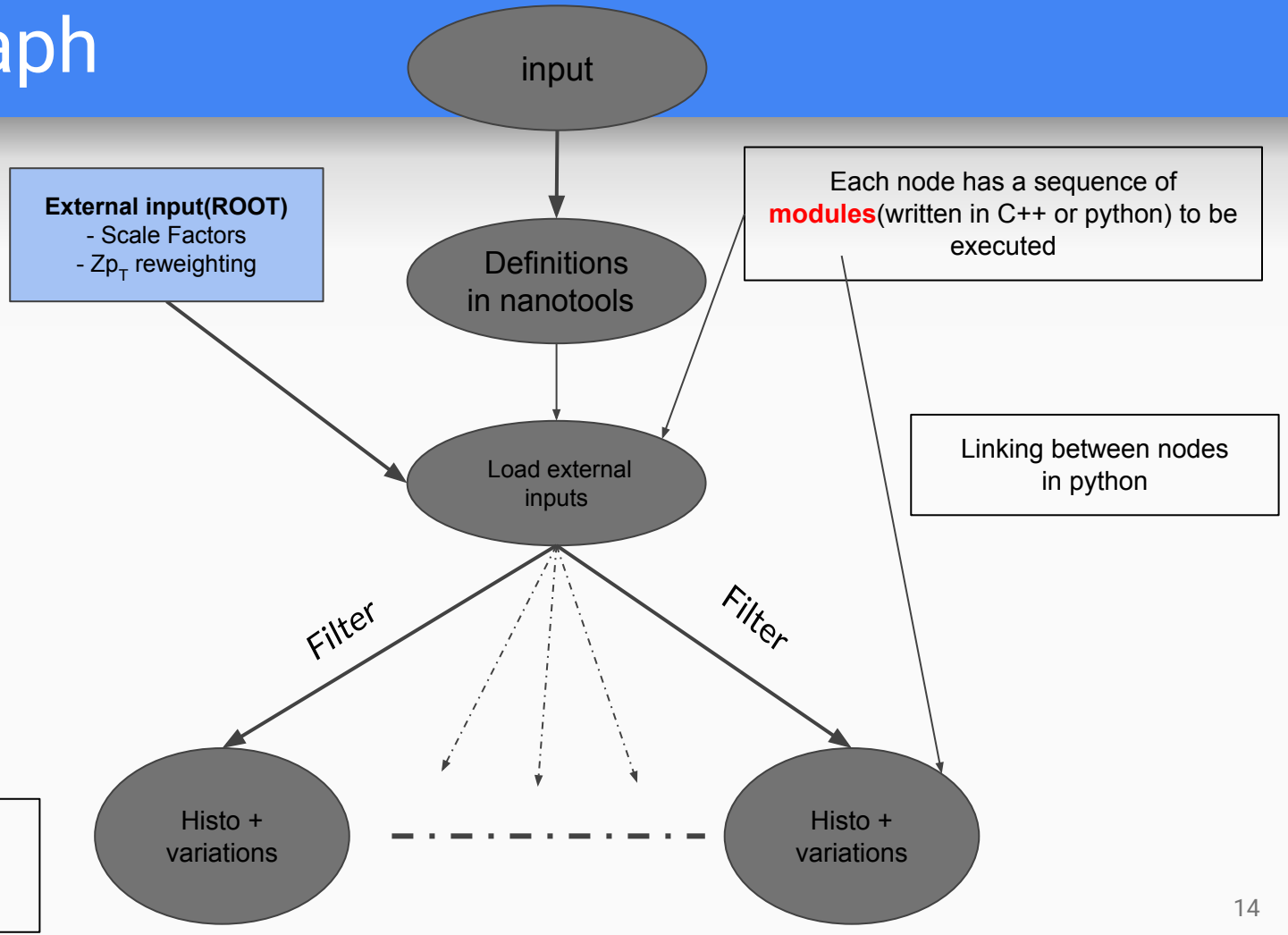
Pisa Framework - Snapshot of the repo

Common	All external inputs like various weights, sample info, plotting tools	
Fit	Fitting tools based on combine-tf	
RDFprocessor @ 804ba7e	Core software which builds and runs the analysis	
config	User API	
nanotools	Creating complex variables on the fly from input NANOAOD	
templateMaker	Applying various selections, loading external inputs, defining histograms	
.gitignore	update gitignore	8 months ago
.gitmodules	adding RDFprocessor as a submodule	2 years ago
Makefile	updating Makefile and adding setup script	8 months ago
README.md	Fix README syntax	8 months ago
setpath.sh	updating Makefile and adding setup script	8 months ago

Common

Analysis graph

- Graph style approach to an analysis.
- Intuitive visualisation of the whole analysis.
- Event loop is run only once.
- parallelizable since it is built on RDF.



Framework - modules I

- definitions : Define every quantity needed later starting from columns available in input.
  - E.g. MT

```
.Define("MT", W_mt, {"Mu1_pt", "Mu1_phi", "MET_pt_nom", "MET_phi_nom"})
```

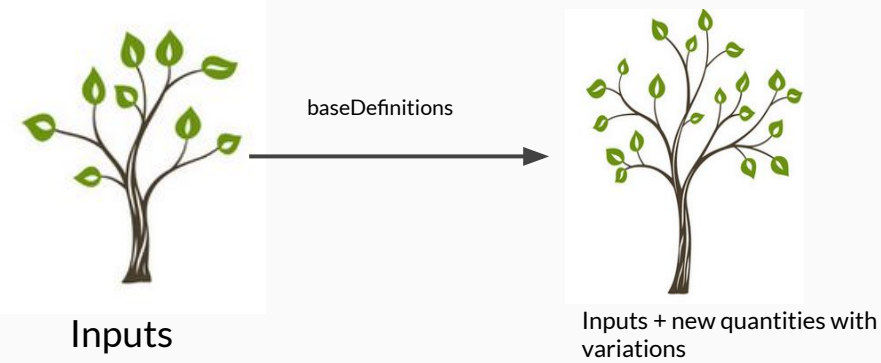
New column

Helper function

Input columns

float W_mt(float,float,float,float)

- Columns for variations of quantities, e.g. Muon_pt_correctedUp/Down, are also defined at this stage.



Framework - modules II

- weightDefinitions: takes ROOT files with external inputs, e.g. Scale factors.
- Reads required histos and defines new columns for nominal weights and variation of

```
auto d1 = d.Define("WHSF", defineWHSF, {"Mu1_pt", "Mu1_eta", "Mu1_charge"})
           .Define("WHSFVars", defineWHSFVars, {"Mu1_pt", "Mu1_eta", "Mu1_charge"})
```

```
auto defineWHSF = [](float pt, float eta, float charge){
 int bin = SFhistoPlus->FindBin(eta,pt);//SFhistoPlus or Minus
 return    SFhistoPlus->GetBinContent(bin);
}
```

This returns a vector< float>

Inputs

baseDefinitions

Inputs + new quantities with variations

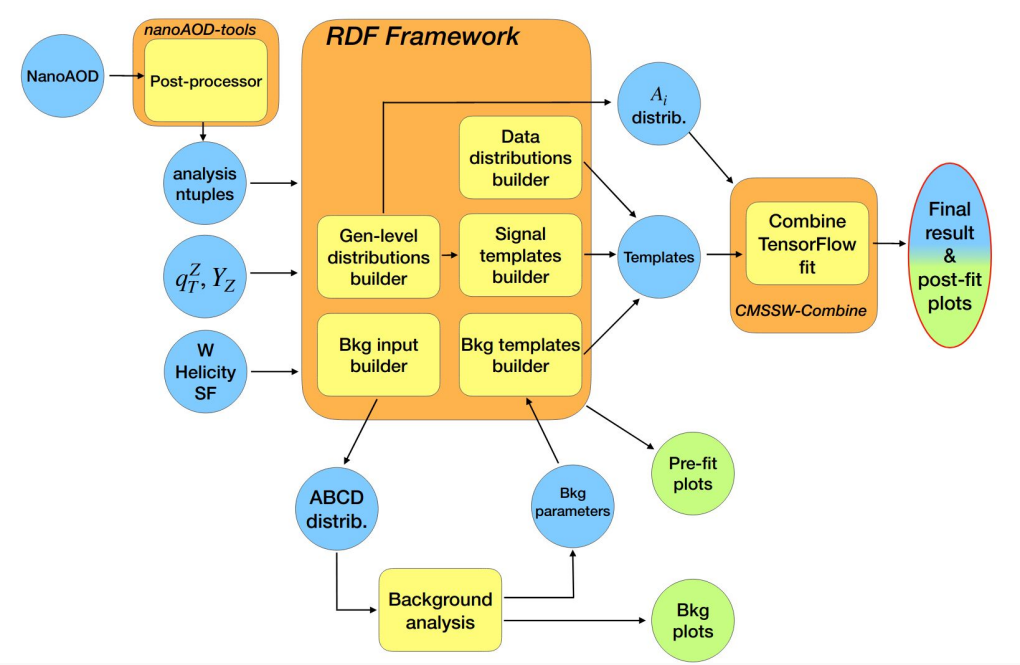
Weight modules

Inputs + new quantities with variations + weights and variations

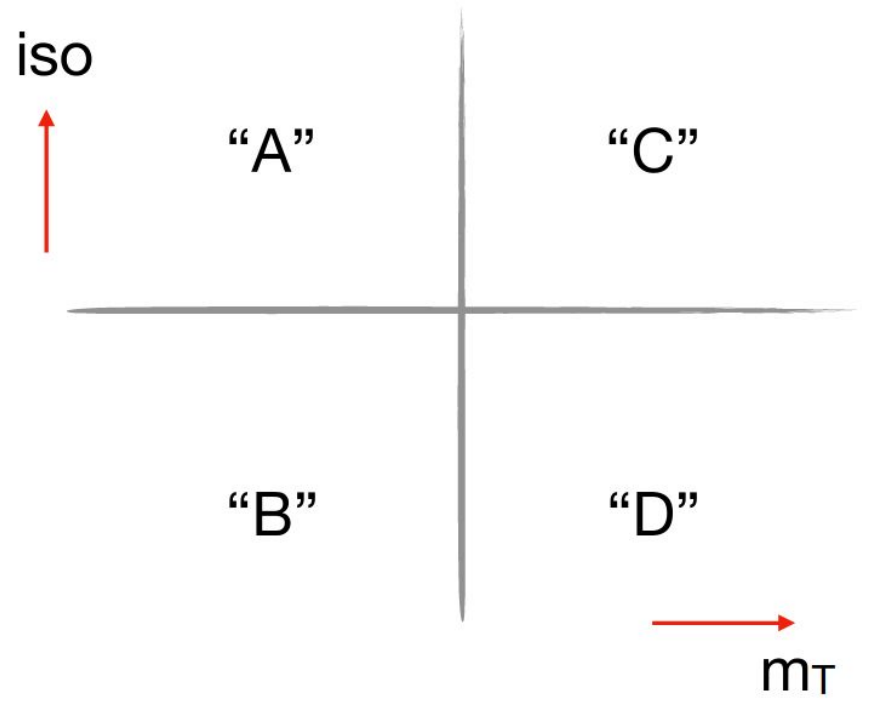
At this point we are ready to make plots etc.

Pisa analysis framework V1

- The framework has seen 2 major milestones
- Work of Valerio's thesis is based on the first version
- Background analysis is run outside RDF
- ROOT histogramming
- Needed an additional processing of NANOAOD with "post-processor"to add some missing branches



Thinking in higher dimensions

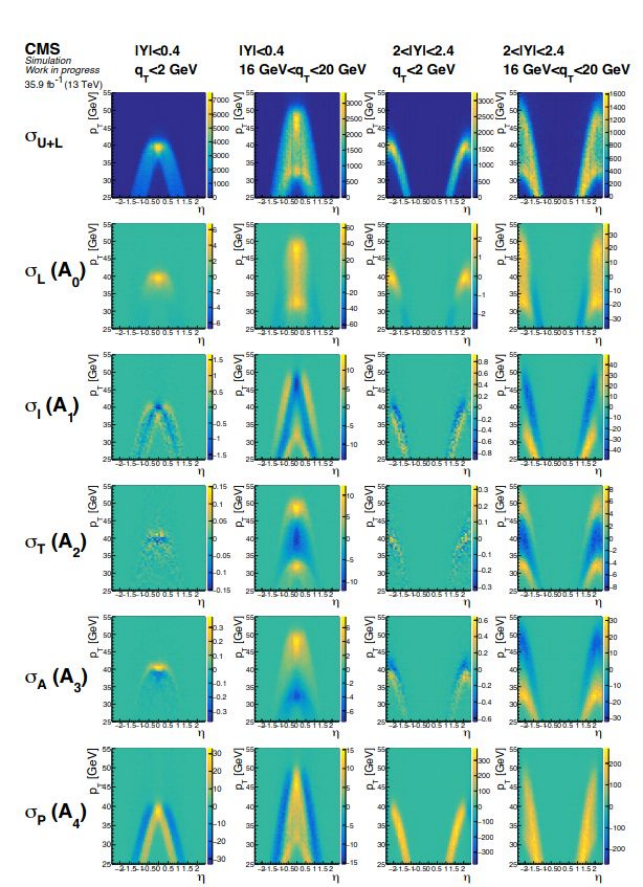


Inside each region get a template of η-p_T of the muon

- in bins of W rapidity and W p_T
- for each helicity cross section
- for different mass hypothesis

Thinking in higher dimensions

Many templates!

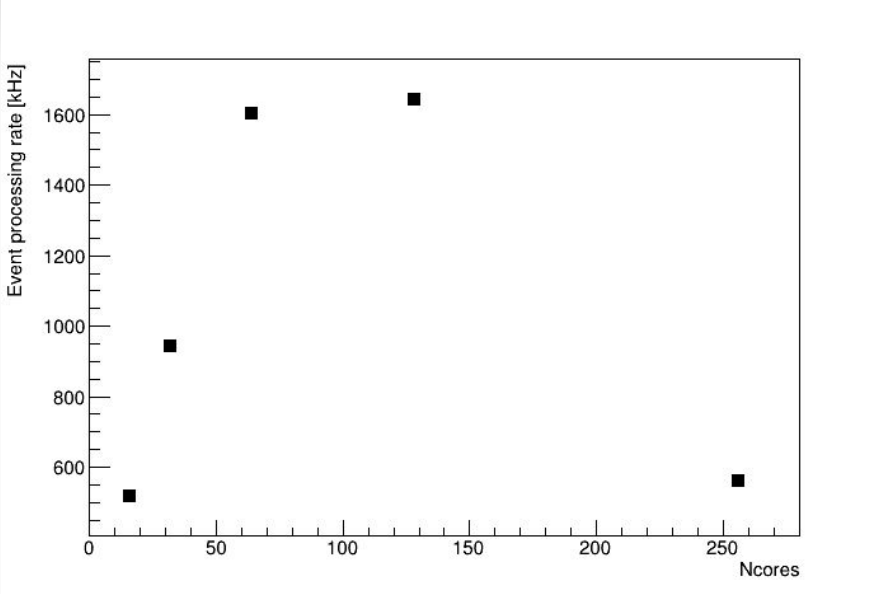


Pisa analysis framework V2

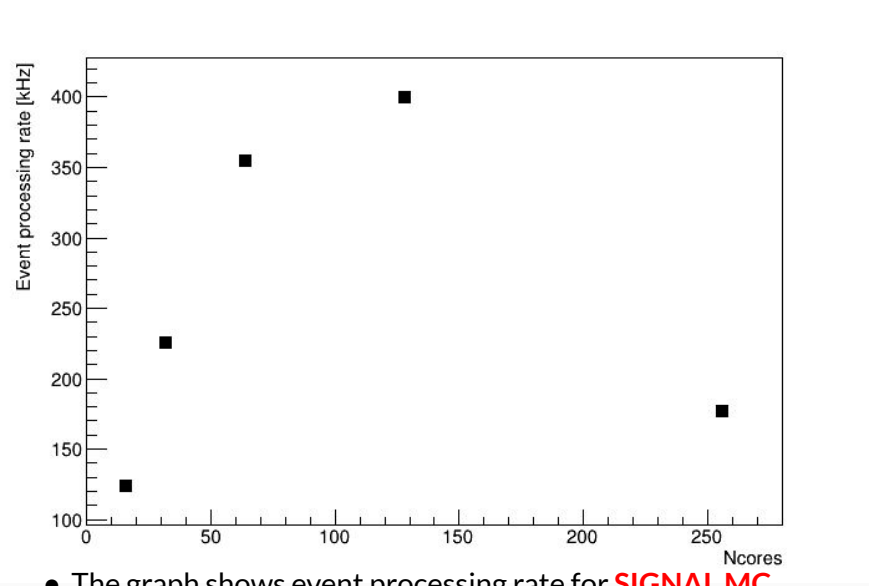
- If we want to fit the QCD shape and normalisation from the 4 regions simultaneously, we have to deal with 7D histograms per helicity per nuisance !
- To achieve this, we switched to BOOST histogramming.
- Advantages
  - It's possible to supply a vectorized array of weights to a histogram - easier to implement systematics.
  - Faster disk I/O
  - Major development was done to enable parallelized filling higher dimensional histos via shared memory
- We can now process these complex objects at ~ MHz level
- Getting the inputs for the fit takes some minutes thanks to our optimised framework
- Drop usage post-processing tools - run directly from NANOAOD

Performance Scaling

On Pisa server with AMD EPYC 7742 processor, 256 cores, 54 TB SSD nvme



- The graph shows event processing rate for **DATA.**
  - Total 55.9 GB input file size read from SSD.
  - 429124293 events
- Histograms written out : 13



- The graph shows event processing rate for **SIGNAL MC**
  - Total 397 GB input file size read from SSD.
  - 652538760 events
- Histograms written out : 6351

Adding a factor 500 histos but pay a factor 4 in speed.

Outlook

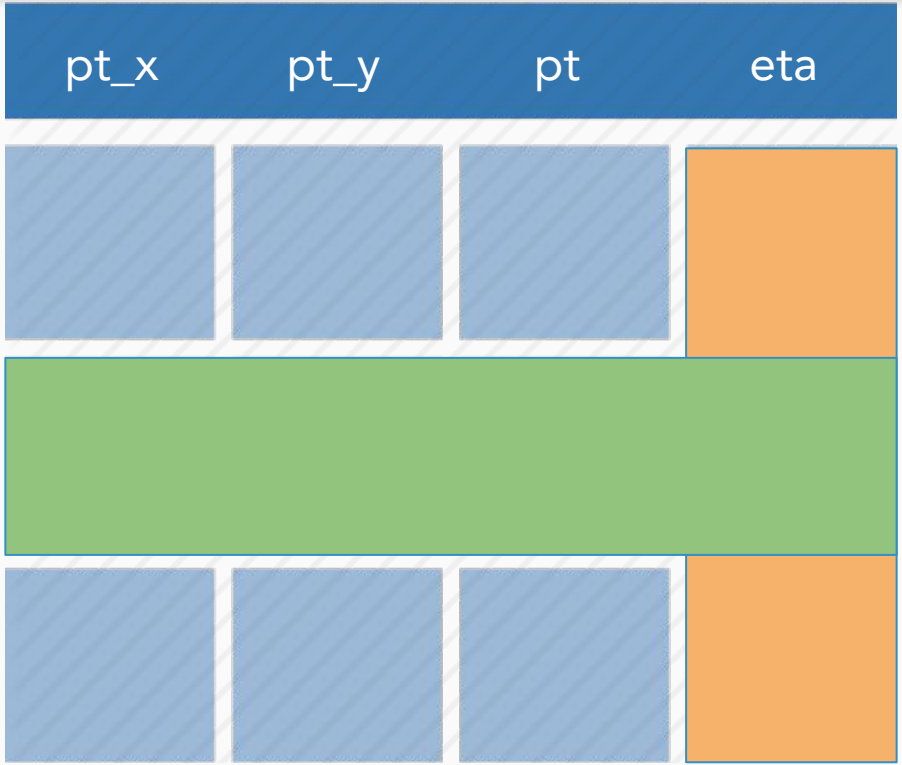
- We have developed an analysis framework based on ROOT DataFrame.
  - Lot of brainstorming from our side in the last years.
- A flexible framework to handle the large complexity of the analysis.
  - RDataFrame + Boost histogramming allows data processing at MHz level
- Encouraging performance results already obtained.

More on RDF

- Seminar by E. Manca and E. Guiraud.
  - https://indico.cern.ch/event/849610/
- Reference manual
  - https://root.cern/doc/master/classROOT_1_1RDataFrame.html
- Tutorials
  - https://root.cern.ch/doc/master/group__tutorial__dataframe.html

BACKUP

Data representation in RDataFrame



columns
or branches

can contain any kind
of c++ object

entries
or events
or rows

RDataFrame example

Read tree “t” in file “f.root”. For events for which “v2 > 2”, fill histogram “h” with “v1”

```
TFile f("f.root");
TTree *t = nullptr;
f.GetObject("t", t);
t->Draw("v1 >> h", "v2 > 2");
TH1 *h = (TH1F*)(gDirectory->Get("h"));
h->Draw();
```

```
TDataFrame d("t", "f.root");
auto h = d.Filter("v2 > 2")
          .Histo1D("v1");
h->Draw();
```


26

RDataFrame example II

Traditional approach

```
import ROOT

fIn = ROOT.TFile.Open("file.root")
tree = fIn.tree

for event in tree:
    if len(event.muons)<1: continue
    if not event.MET>20: continue

    for muon in event.muons:
        if muon.pt > 25 and  abs(muon.eta)<2.4 \
        and muon.dz<0.1 and muon_dxy<0.01 \
        and muon.relIso<0.5:

         selmuon_pt = muon_pt
```

loop over events

loop over muons

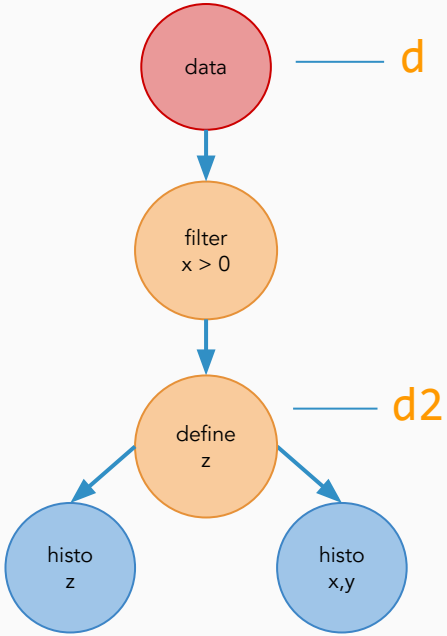
RDataFrame approach

```
import ROOT

ROOT.ROOT.EnableImplicitMT()

RDF = ROOT.ROOT.RDataFrame
d = RDF(treeName, inputFile)

d = d.Filter("nMuon>1 && MET>20")\
      .Define("SelMuon_pt"
      ,"Muon_pt[Muon_pt>25 \
      && abs(Muon_eta)<2.4 \
      && Muon_dz<0.1 && Muon_dxy<0.01 \
      && Muon_relIso<0.5]")
```


27

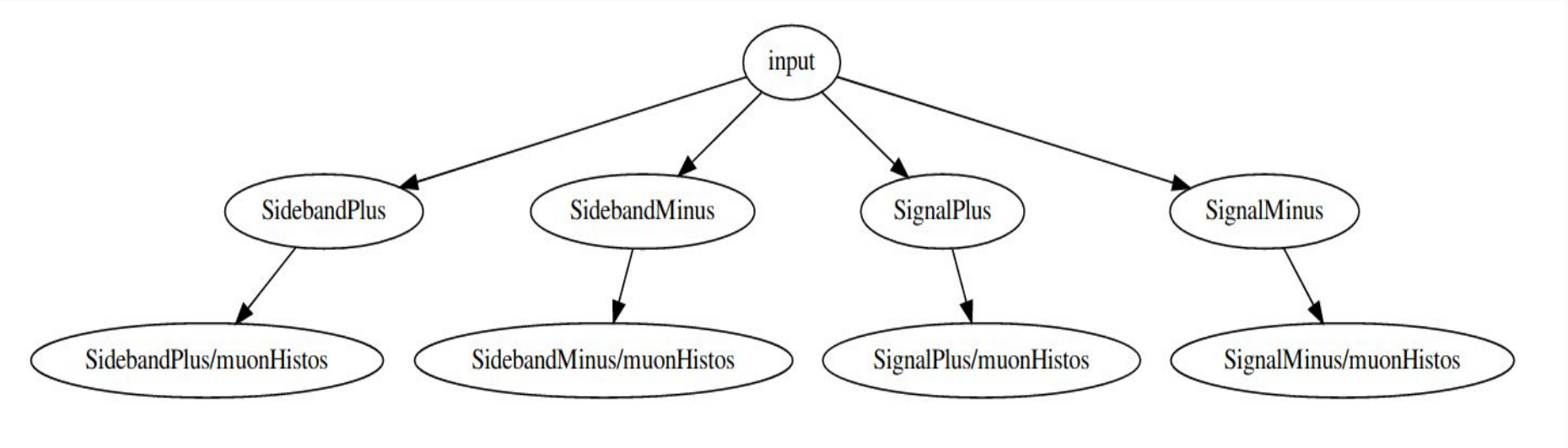
Thinking in terms of Graphs

```
// d2 is a new data-frame, a transformed version of d
auto d2 = d.Filter("x > 0")
           .Define("z", "x*x + y*y");

// make multiple histograms out of it
auto hz = d2.Histo1D("z");
auto hxy = d2.Histo2D("x","y");
```



- More intuitive way of visualizing an analysis workflow.

Thinking in terms of Graphs



- Simplest example from our analysis
- Input split for different regions(filters)
- Single module(muonHistos) with histogram definitions called for each region.

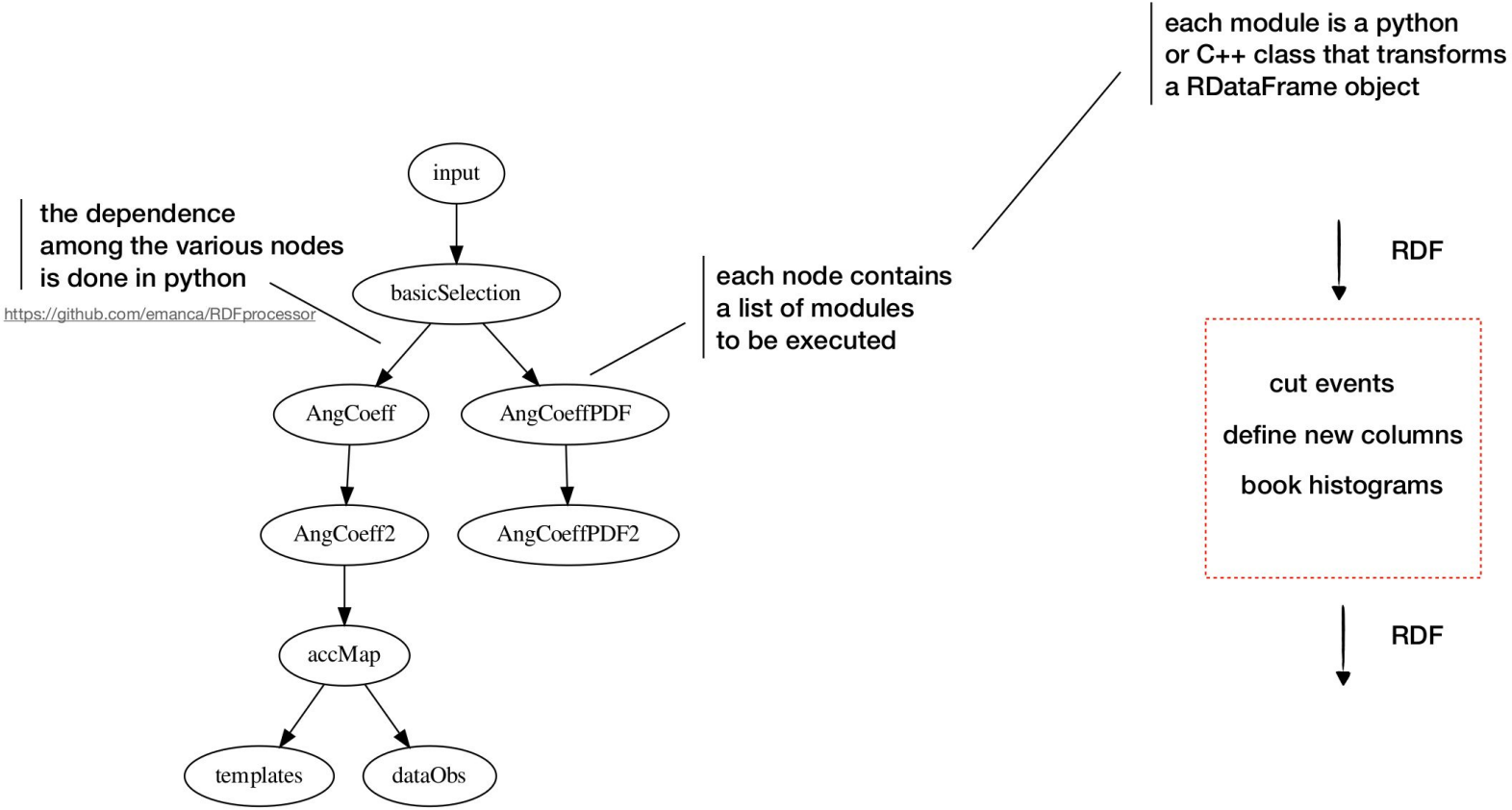
Performance Scaling

- Several trials to optimize the run time.
  - Same code, varied sample size....
- Multi-threading + multiprocessing gives us the best runtimes.
  - Smaller samples(e.g. diboson) - single thread with multi-processing.
  - Large samples - Multi-threaded
- For MT, we decided to use upto 128 cores.
- Table shows runtimes for all the analysis steps starting from the NanoAOD inputs to prefit plotting involving RDF.
- Extraction of signal templates is the most time consuming part.

Step	Runtime (sec)	RDF used
Preparing inputs for background estimation, run on all samples	1817	yes
Fake estimation from data	252	yes
Signal template extraction from WJets	1265	yes

NanoAOD to input for fit- total runtime ~ 1 hour

Thinking in terms of Graphs



Thinking in terms of Graphs