Testing RDataFrame in CMS analysis

Elisabetta Manca (Scuola Normale Superiore & CERN)

benchmark: W properties measurement using full Run2

data taking period	**2016**	**2017**	**2018**
number of events into acceptance	35 fb^{-1} ~100 M	45 fb^{-1} ~130 M	65 fb^{-1} ~185 M

x10 statistics in Montecarlo to have statistical error negligible

benchmark: W properties measurement using full Run2

data taking period	2016	2017	2018
number of events into acceptance	35 fb^{-1} ~100 M	45 fb^{-1} ~130 M	65 fb^{-1} ~185 M

x10 statistics in Montecarlo to have statistical error negligible

do we have the tools to process such a huge number of events?

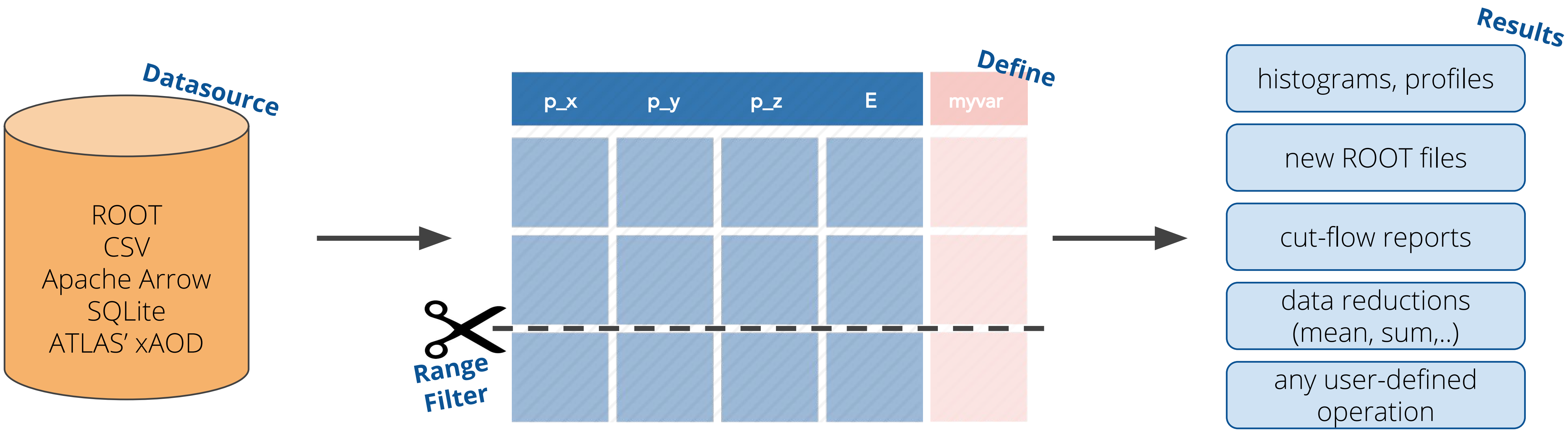
proposed solution: go multithread!

RDataFrame offers the possibility to easily *parallelise* and *optimise* the event loop

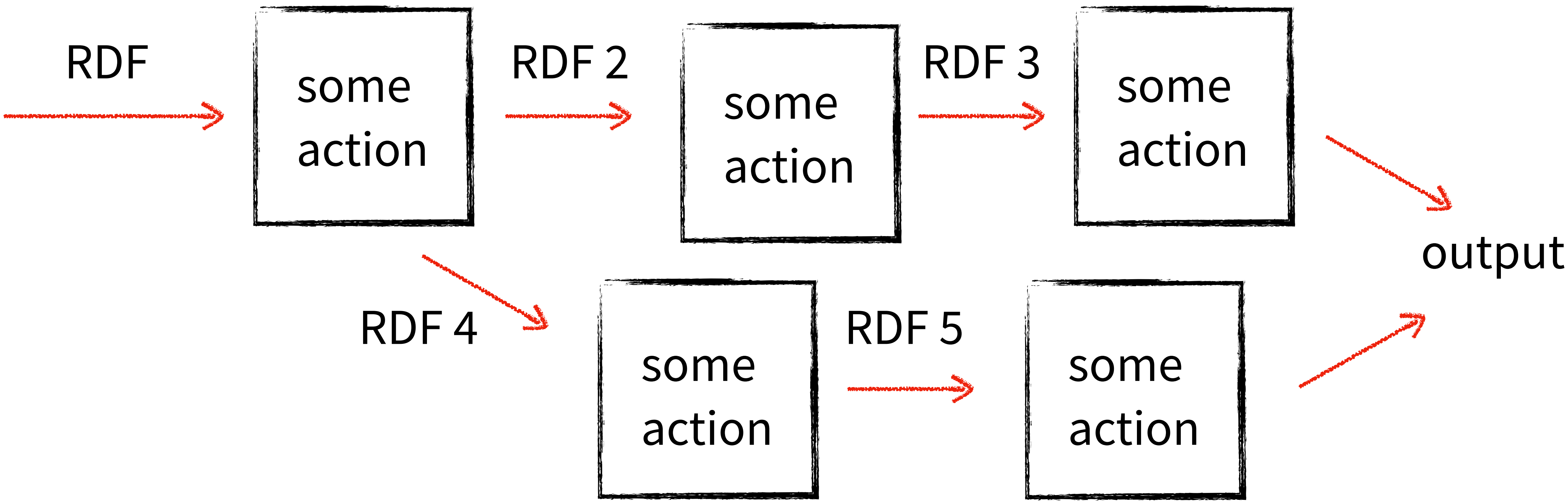
NanoAOD as a new data format distributed by CMS as “centralised ntuples”

credits: ROOT team

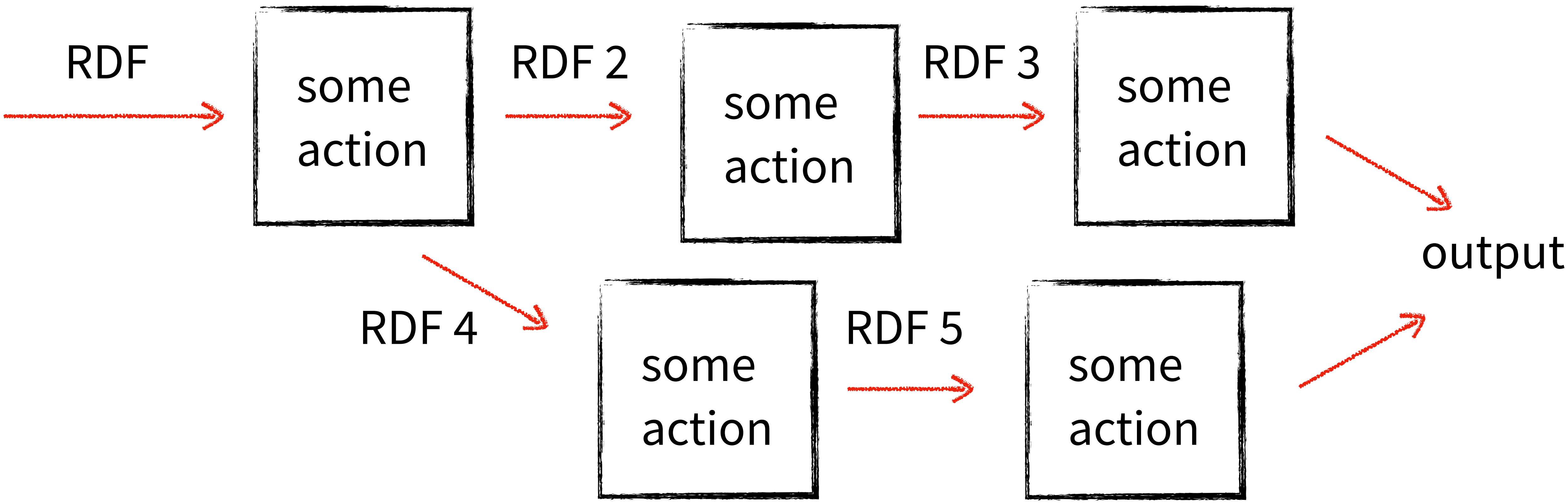
RDataFrame in a nutshell



the idea: wrap RDF into a mini-framework that executes some modules following a graph path



the idea: wrap RDF into a mini-framework that executes some modules following a graph path



RDataFrame transparently performs *data* parallelism
and the event loop is run only once
(due to RDF laziness and some output collection optimisation)

some action

= a python class

= a C++ class (inherits from a virtual mother class)

```python
class filter:

    def __init__(self, string):

        self.myTH1 = []
        self.myTH2 = []
        self.myTH3 = []
        self.string = string

    def run(self,d):

        self.d = d.Filter(self.string)
        return self.d

    def getTH1(self):

        return self.myTH1

    def getTH2(self):

        return self.myTH2

    def getTH3(self):

        return self.myTH3
```

```cpp
#include “filter.h"

RNode filter::run(RNode d){

   auto d1=d.Filter("Mystring_");

   return d1
}

std::vector<ROOT::RDF::RResultPtr<TH1D>> filter::getTH1(){

   return _h1List;

}

std::vector<ROOT::RDF::RResultPtr<TH2D>> filter::getTH2(){

   return _h2List;

}

std::vector<ROOT::RDF::RResultPtr<TH3D>> filter::getTH3(){

   return _h3List;

}
```

an example of action: extraction of the “Angular Coefficients” from a WJets Montecarlo

spherical harmonics 2nd order (W has spin 1!)

task: compute

$$m = \frac{\sum P_k(\cos\theta, \phi) w_i}{\sum w_i}$$

$$\sigma_m = \sqrt{\sigma^2_{P_k} \frac{\sum w_i^2}{(\sum w_i)^2}}$$

for each bin of W p_T and y
for each harmonics k = 0,…,7

implementation:

- 1 TH2 filled with *w*

for each harmonics (0 to 7):

- 1 TH2 filled with P_k and weighted with *w*
- 1 TH2 filled with $P_k{}^2$ (to compute variance)

in RDF language:
about 10 Filters and 10 Defines

an example of usage:

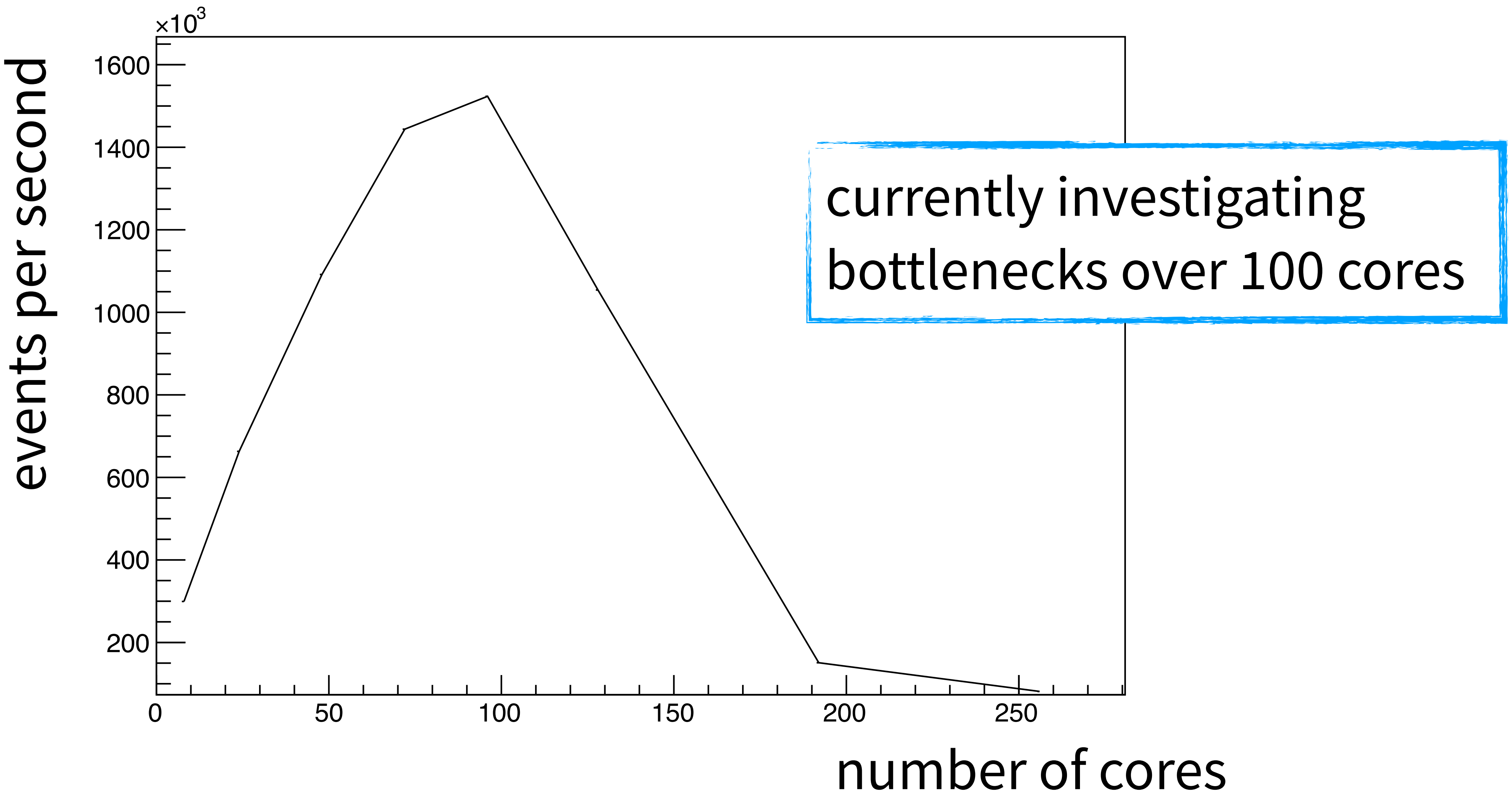
```
import ROOT

from RDFtreeV2 import RDFtree

ROOT.ROOT.EnableImplicitMT(64)

cut = 'pt1>22.0&&pt2>20.0&&abs(eta1)<2.5&&abs(eta2)<2.5&&mass>75&&mass<115'
inputFileD ='/scratch/emanca/WMass/MuonCalibration/KaMuCaSLC7/CMSSW_8_0_20/src/KaMuCa/Derivation/
run/Scale/muonTreeDataZ.root'

calibData = ROOT.string(‘DATA_80X_13TeV')

# data
pD = RDFtree(outputDir = 'TEST', outputFile = "beforeCalibData.root", inputFile = inputFileD, \
modules=[prepareSample(cut=cut, target=90.851,isMC=False),basicPlots(res='Z')], treeName = 'tree')

# execute run() method of a class
pD.run()

# add new classes

pD.branch([ROOT.applyCalibration(calibData),basicPlots(res='Z',
calib=True),prepareSampleAfter(cut,res='Z')],outputFile="afterCalibData.root")

# collect output and trigger event loop
pD.getOutput()
```

a scaling plot:

384 cores Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
Lots of SSD storage, bleeding edge hardware
@ Scuola Normale Superiore
27 GB input data (~4000 cluster)

a scaling plot:

384 cores Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
Lots of SSD storage, bleeding edge hardware
@ Scuola Normale Superiore
27 GB input data (~4000 cluster)



conclusions:

the combo RDF + NanoAOD (or any kind of flat tree) proved to be very successful in reducing execution time in multicore machines

mini-framework is a nice tool to perform an analysis in a compact way.
new features can be added:

ex. can we treat systematic uncertainties in an automatic and efficient way?