RDataFrame: a flexible and scalable analysis experience

V. E. Padulano, E. Guiraud, E. Tejedor Saavedra, I. Kabadzhov, Pawan
for the ROOT team
ACAT 2022, 26/10/2022

The HEP analysis landscape as we see it

Analysis life cycle

skimming, ntuple production	quick exploration, first implementation	systematics, scale out	statistical analysis

Check out the latest RooFit talk!

Platforms

laptop or PC	many-core machine	computing cluster + job submission

Analysis languages

- ~50% C++
- ~50% Python

Storage

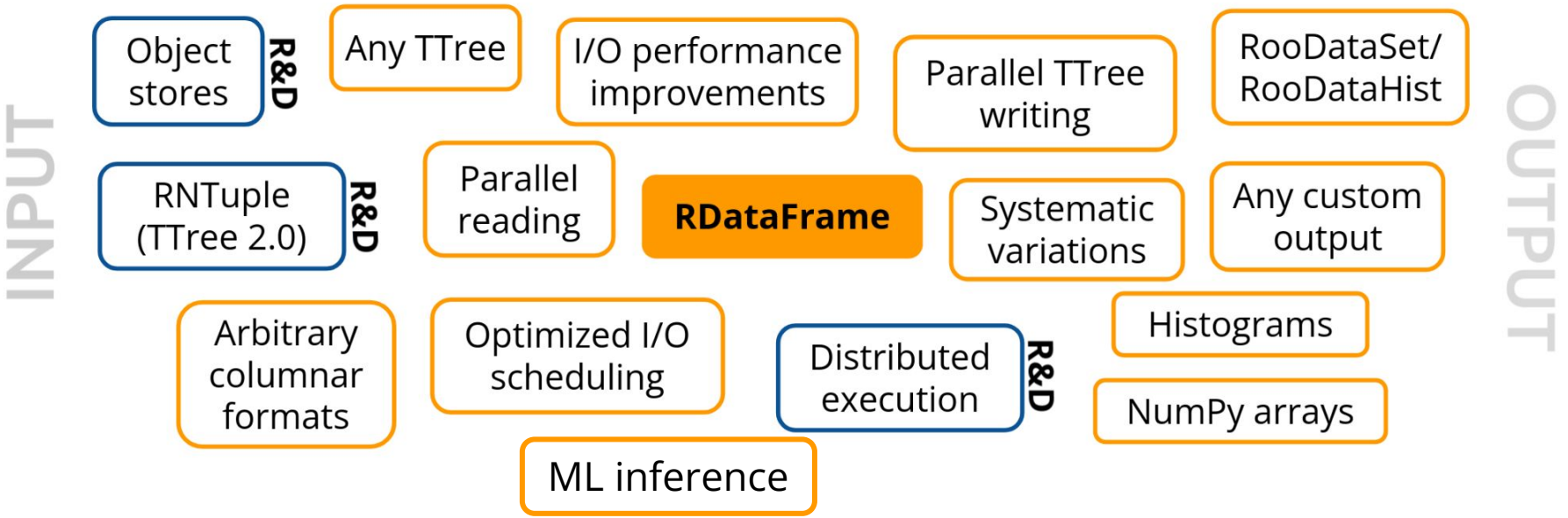
local disk
fast-access network storage
EOS or other not-so-fast backend

Ingredients of an analysis

An analysis needs a matrix of operations to be configured and tracked.

The simpler the software can express this matrix, the easier the task for the analyst.

An entry point to modern ROOT



Inspecting (remote) data

Python

```
df = ROOT.RDataFrame(
  "Events",
  "root://eospublic.cern.ch//eos/root-eos/cms_opendata_2012_nanoaod_skimmed/ZZTo2e2mu.root"
)
df.Display(["Electron_charge", "Electron_pt", "Muon_charge", "Muon_pt"]).Print()
```

Row	Electron_charge	Electron_pt	Muon_charge	Muon_pt
0	-1	15.9803f	-1	39.3238f
	1	24.5981f	1	54.5436f
1	-1	53.3949f	-1	19.7296f
	1	18.2565f	1	22.7590f
2	1	32.4470f	1	61.6184f
	-1	98.6543f	-1	41.6536f
3	-1	27.7553f	-1	9.49819f
	1	26.3404f	1	4.64695f
4	1	24.7105f	1	52.4297f
	-1	24.6616f	-1	60.9041f

Inspecting (remote) data

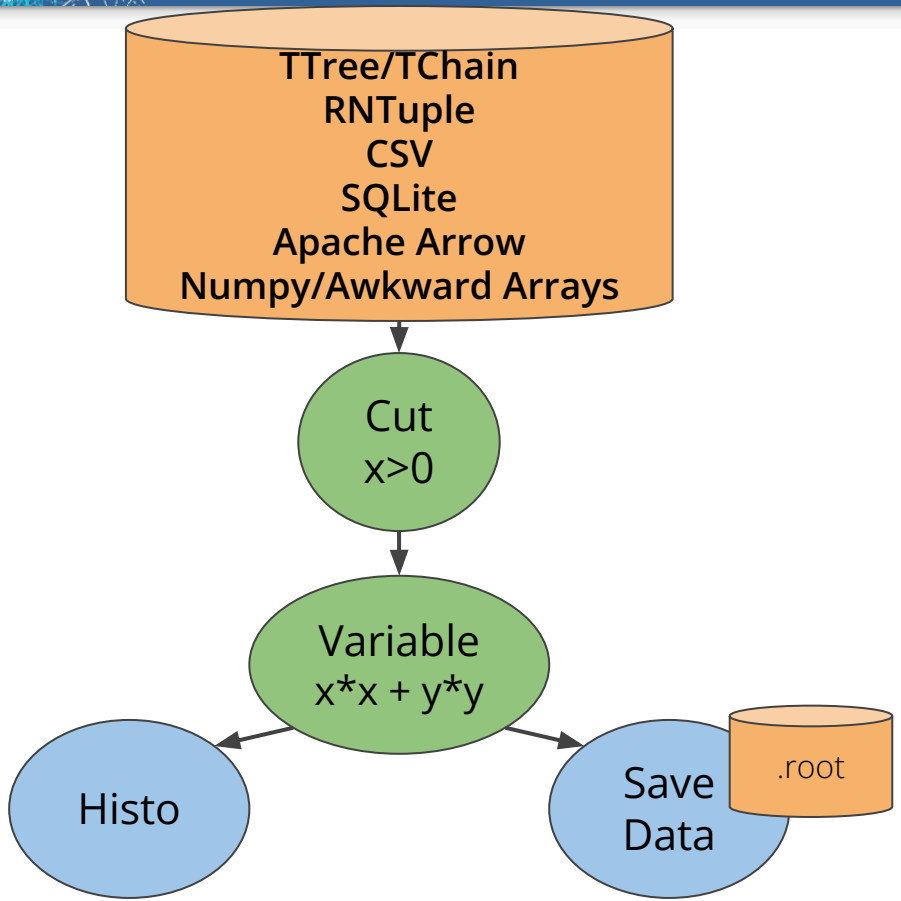
Python

```
df = ROOT.RDataFrame(
  "Events",
  "root://eospublic.cern.ch//eos/root-eos/cms_opendata_2012_nanoaod_skimmed/ZZTo2e2mu.root"
)
df.Describe()
```

```
Property                Value
--------                -----
Columns in total           32
Columns from defines        0
Event loops run             0
Processing slots            1

Column                  Type                            Origin
------                  ----                            ------
Electron_charge         ROOT::VecOps::RVec<Int_t>       Dataset
Electron_dxy            ROOT::VecOps::RVec<Float_t>     Dataset
Electron_dxyErr         ROOT::VecOps::RVec<Float_t>     Dataset
Electron_dz             ROOT::VecOps::RVec<Float_t>     Dataset
Electron_dzErr          ROOT::VecOps::RVec<Float_t>     Dataset
Electron_eta            ROOT::VecOps::RVec<Float_t>     Dataset
Electron_mass           ROOT::VecOps::RVec<Float_t>     Dataset
```

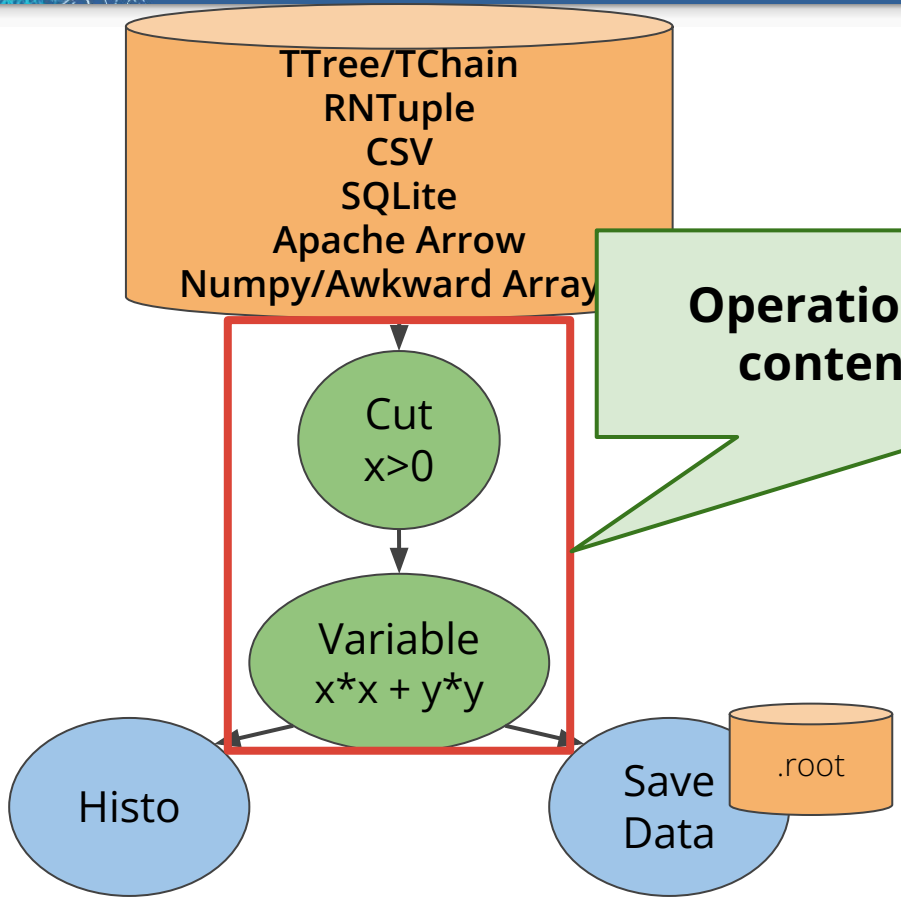
...

First steps with RDF



- Open this dataset
- Make a cut
- Create a new variable
- Make histogram
- Save data to file

First steps with RDF

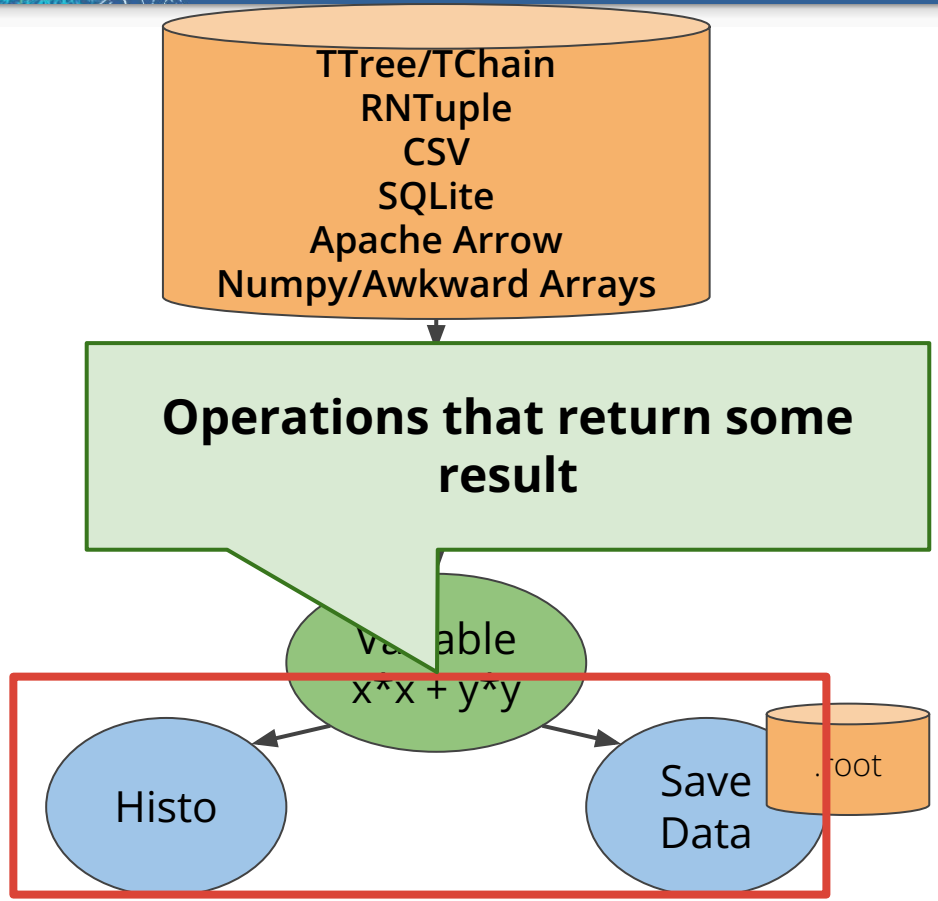


- Open this dataset

Operations that change the contents of the dataset

- Create a new variable
- Make histogram
- Save data to file

First steps with RDF



- Open this dataset
- Make a cut
- Create a new variable
- Make histogram
- Save data to file

First steps with RDF

```
df = ROOT.RDataFrame(dataset)


df = df.Filter("x > 0")\
       .Define("r2", "x*x + y*y")
rHist = df.Histo1D("r2")
df.Snapshot("newtree", "out.root")
```

- Open this dataset
- Make a cut
- Create a new variable
- Make histogram
- Save data to file

First steps with RDF

```
df = ROOT.RDataFrame(dataset)

df = df.Filter("x > 0")\
       .Define("r2", "x*x + y*y")

rHist = df.Histo1D("r2")

df.Snapshot("newtree", "out.root")
```

Cuts

Derived quantities/observables

- Open this dataset
- Make a cut
- Create a new variable
- Make histogram
- Save data to file

First steps with RDF

```
df = ROOT.RDataFrame(dataset)

df = df.Filter("x > 0")\
       .Define("r2", "x*x + y*y")

rHist = df.Histo

df.Snapshot("newTree", "out.root")
```

Cuts

Derived quantities/observables

- Open this dataset
- Make a cut
- Create a new variable

Users can inject **arbitrary code** at all steps, which makes this relatively simple API extremely versatile.

RVecs: working with collections

Cut+Variable+Histo: quick one-liner

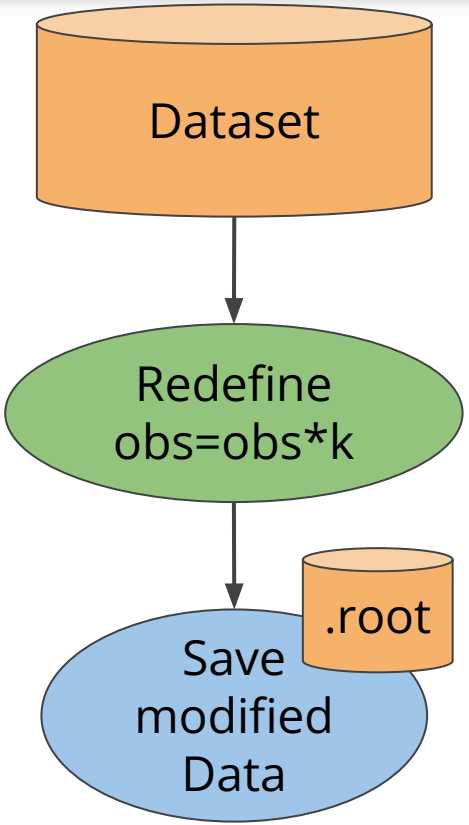
Python
```
h = df.Define("pt", "muon_pt[abs(muon_eta) < 2]").Histo1D("pt")
```

Compiled C++

C++
```
RVecD selectPt(RVecD &pt, RVecD &eta) {
  return pt[abs(eta) < 2];
}

auto h = df.Define("pt", selectPt, {"muon_pt", "muon_eta"})
           .Histo1D<RVecD>("pt");
```

See also the user guide

Re-defining values of observables



Task:

- My dataset comes from some external processing
- I want to rewrite values of a certain observable/column
- The name for the observable should stay the same

Re-defining values of observables

```
df = ROOT.RDataFrame(
        "Events",
        "root://eospublic.cern.ch//eos/root-eos/"
        "cms_opendata_2012_nanoaod_skimmed/ZZTo2e2mu.root"
)

df = df.Redefine(
        "Electron_mass",
        "Electron_mass*correction"
)

massHist = df.Histo1D("Electron_mass")
```

Task:

- My dataset comes from some external processing
- I want to rewrite values of a certain observable/column
- The name for the observable should stay the same

Re-defining values of observables

```
df = ROOT.RDataFrame(
        "Events",
        "root://eospublic.cern.ch//eos/root-eos/"
        "cms_opendata_2012_nanoaod_skimmed/ZZTo2e2mu.root"
)

df = df.Redefine(
        "Electron_mass",
        "Electron_mass*correction"
)

massHist = df.Histo1D("Electron_mass")
```

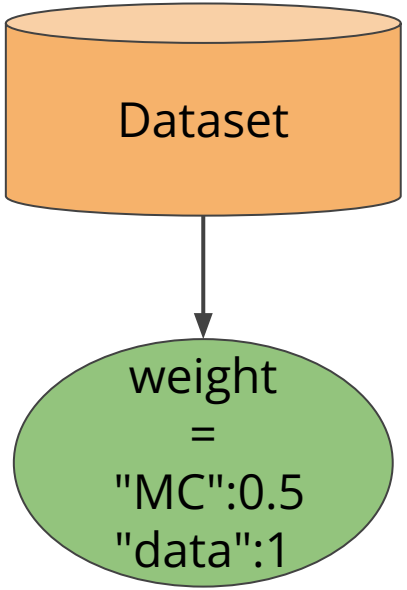
Use previous values to define new ones

Access updated observable with same name

Task:

- My dataset comes from some
  cessing
  rite values of a
  certain observable/column

Variables that depend on the sample



Task:

- I need to define the weight for an observable
- The weight depends on whether the sample is data or simulation

Variables that depend on the sample

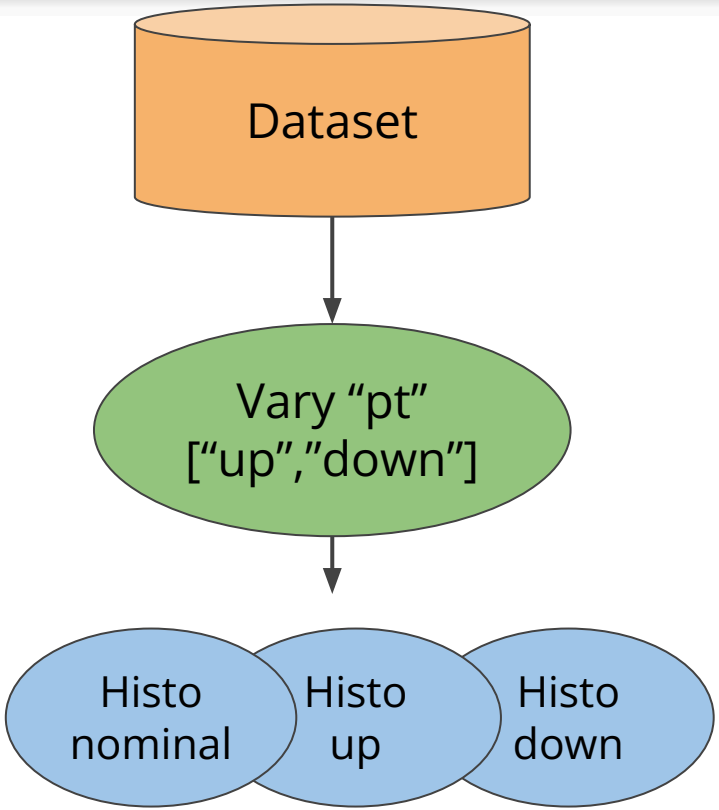
Python

```
df.DefinePerSample(
   "weight",
   'rdfsampleinfo_.Contains("MC") ? 0.5 : 1.'
).Histo1D("value","weight")
```

C++

```
df.DefinePerSample("weight",
      [](unsigned slot, const RDF::RSampleInfo &s) {
           return s.Contains("MC") ? 0.5 : 1.; })
   .Histo1D("value","weight");
```

Systematic variations (1)



Task:

- I need to include systematic variations for one of my observables
- Plot histograms: nominal and varied

Systematic variations (1)

Python

```
nominal_hx = (
   df.Vary("pt", "RVecD{pt*0.9, pt*1.1}",  ["down", "up"])
      .Filter("pt > k")
      .Define("x", someFunc, ["pt"])
      .Histo1D("x")
)
hx = ROOT.RDF.VariationsFor(nominal_hx)
hx["nominal"].Draw()
hx["pt:down"].Draw("SAME")
```

Systematic variations (1)

attach an up/down variation to “pt”

Python

```
nominal_hx = (
   df.Vary("pt", "RVecD{pt*0.9, pt*1.1}",  ["down", "up"])
      .Filter("pt > k")
      .Define("x", someFunc, ["pt"])
      .Histo1D("x")
)
hx = ROOT.RDF.VariationsFor(nominal_hx)
hx["nominal"].Draw()
hx["pt:down"].Draw("SAME")
```

Systematic variations (1)

Python

```
nominal_hx = (
   df.Vary("pt", "RVecD{pt*0.9, pt*1.1}",  ["down", "up"])
      .Filter("pt > k")
      .Define("x", someFunc, ["pt"])
      .Histo1D("x")
)
hx = ROOT.RDF.VariationsFor(nominal_hx)
hx["nominal"].Draw()
hx["pt:down"].Draw("SAME")
```

attach an up/down variation to “pt”

proceed as usual, as if working with nominal values only

Systematic variations (1)

Python

```
nominal_hx = (
   df.Vary("pt", "RVecD{pt*0.9, pt*1.1}",  ["down", "up"])
      .Filter("pt > k")
      .Define("x", someFunc, ["pt"])
      .Histo1D("x")
)
hx = ROOT.RDF.VariationsFor(nominal_hx)
hx["nominal"].Draw()
hx["pt:down"].Draw("SAME")
```

attach an up/down variation to “pt”

proceed as usual, as if working with nominal values only

obtain all variations

Systematic variations (1)

Python

```
nominal_hx = (
   df.Vary("pt", "RVecD{pt*0.9, pt*1.1}",  ["down", "up"])
      .Filter("pt > k")
      .Define("x", someFunc, ["pt"])
      .Histo1D("x")
)
hx = ROOT.RDF.VariationsFor(nominal_hx)
hx["nominal"].Draw()
hx["pt:down"].Draw("SAME")
```

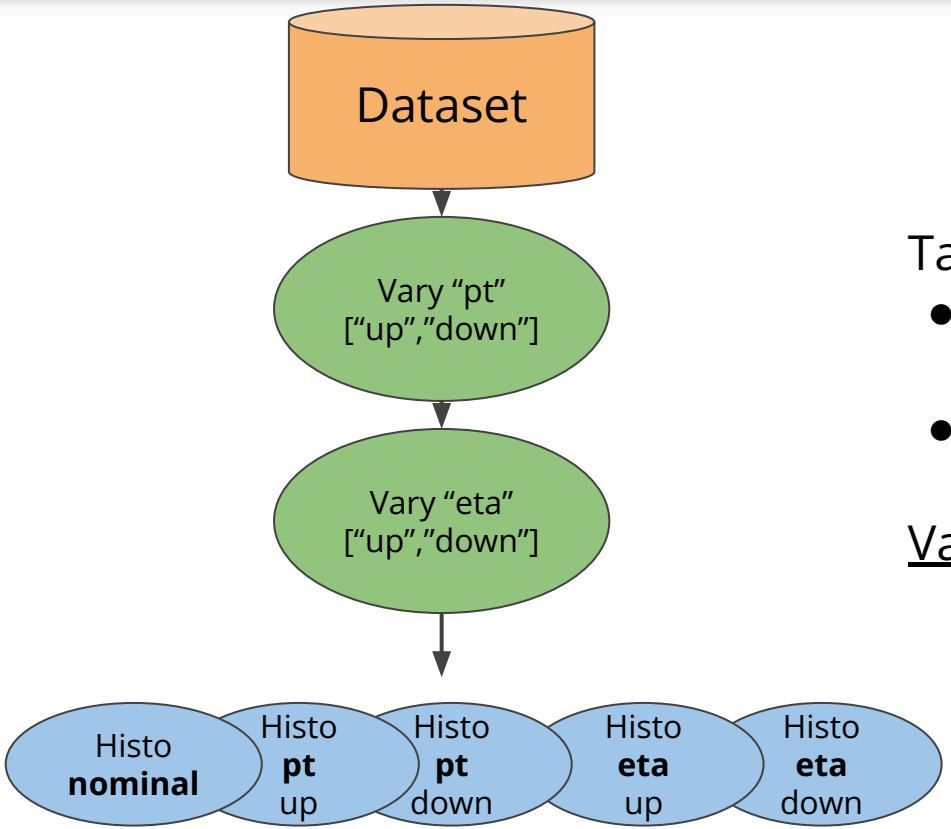
attach an up/down variation to “pt”

proceed as usual, as if working with nominal values only

obtain all variations

Variations automatically propagate to selections, derived quantities and results.
Only needed quantities are re-computed, all in **one event loop**.

Systematic variations (2)



Task:

- I have to include variations for different observables
- Plot again

<u>Variations are applied one at a time</u>

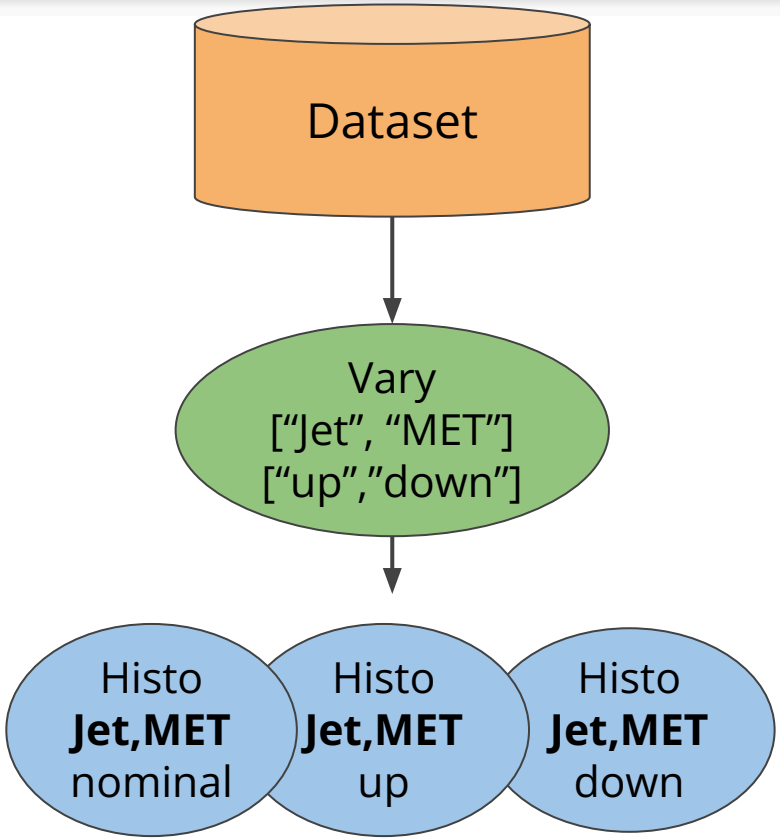
Systematic variations (2)

Python

```
nominal_h = (
  df.Vary("pt", "RVecD{pt*0.9, pt*1.1}",  ["down", "up"])
     .Vary("eta", "RVecD{eta*0.9, eta*1.1}",  ["down", "up"])
     .Histo2D("pt" , "eta")
)

all_h = ROOT.RDF.VariationsFor(nominal_h)
```

Variations are applied one at a time:
the code above creates “universes” nominal, pt:down, pt:up, eta:down, eta:up.

Systematic variations (3)



Task:

- Two observables vary together in my analysis
- Plot again

Systematic variations (3)

Python

```
nominal_hx = df.Vary(
   ["Jet_pt","Jet_mass","MET_pt","MET_phi"],
   getJetMETVariations,
   variationTags=["down", "up"],
   variationName="JetMET"
).Filter("Jet_pt > 30").Histo1D("nJet")

hx = ROOT.RDF.VariationsFor(nominal_hx)
hx["nominal"].Draw()
hx["JetMET:down"].Draw("SAME")
hx["JetMET:up"].Draw("SAME")
```

Composability with the API allows for **simple** definition of **complex observables**

Interoperability with Pythonic arrays

TTree → RDataFrame → Numpy → RDataFrame → TTree

```
df = ROOT.RDataFrame("Events", "file.root")
np_arrs = df.Filter("x > 10").AsNumpy(["x", "y"])
```

```
data = {"x": np.array(...), "y": np.array(...), ...}
df = ROOT.RDF.FromNumpy(data)
```

Awkward arrays → RDataFrame → Awkward Arrays

```
df = ak.to_rdataframe(
    {"x": ak.Array(), "y": ak.Array(), "z": ak.Array()})
```

```
array = ak.from_rdataframe(
    df, columns=("Dimuon_mass"))
```

Related contributions at ACAT:
Poster by Ianna Osborne
Talk by Manasvi Goyal

6.28 Fully Pythonic interface

Support for **Python callables** in RDF through **Numba**

Use external values in RDF event loop

```
k = 10
def cutoff(muon_eta): return muon_eta < k
rdf.Filter(cutoff)
```

Python lambdas

```
rdf.Define("col_c", lambda a,b: a*b)
rdf.Define("col_f", lambda d,e: d*e)
```

Working with collections as numpy arrays

```
def select_pt(muon_pt, muon_eta):
  return muon_pt[np.abs(muon_eta) < 2]

h = df.Define("pt", select_pt).Histo1D("pt")
```

6.28 Easy dataset specification

```
# myspec.json
{
  "sample_a": {
    "files": ["a.root","b.root"],
    "metadata": {
      "w":1., "xsec":21.3, "year":2017}
  },
  "sample_b": {
    "files": ["fb*.root"],
    "metadata": {
      "w":0.5, "type":"MC", "year":2018}
  }
}
```

WIP on defining a common schema for analysis frameworks

Python

```
df = ROOT.RDF.FromJSON("myspec.json")

df.DefinePerSample(
    "weight",
    'rdfsampleinfo_.GetD("w")'
).Histo1D("value","weight")
```

From one core to many cores

Python

```
import ROOT
ROOT.EnableImplicitMT()
# Proceed with the rest of the analysis
```

From one node to many nodes

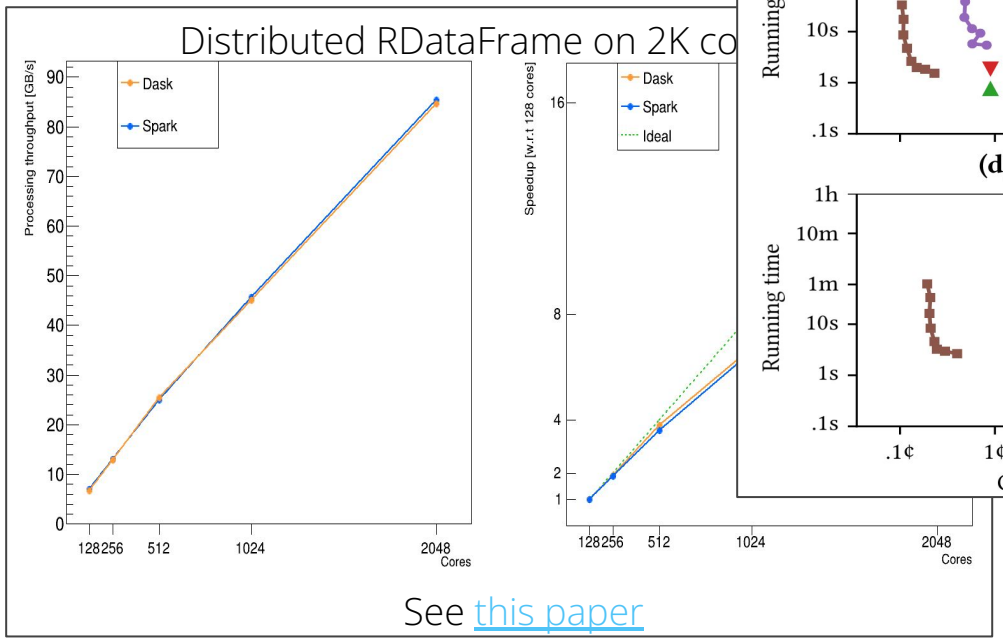
Python

```
cluster = dask_jobqueue.HTCondorCluster(
    n_workers=64, cores=32)

df = RDataFrame(dataset, daskclient=Client(cluster))

# Proceed with the rest of the analysis
```

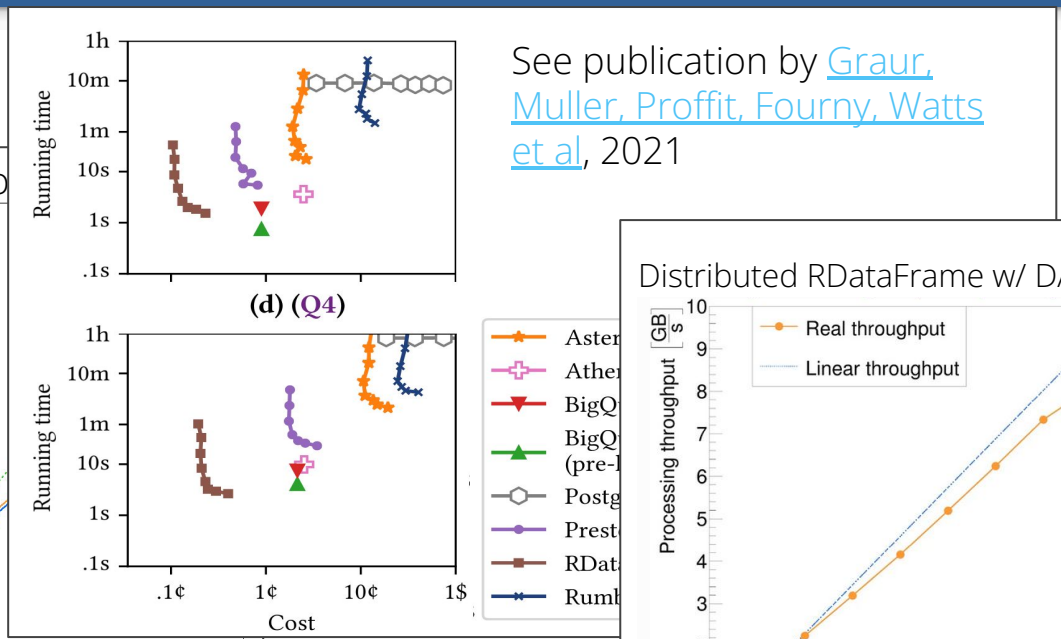
... Also with your (Slurm, Spark, Kubernetes, AWS, bare metal) cluster

tutorial, docs, talk

We care about performance. A lot.



See [this paper]



See publication by [Graur, Muller, Proffit, Fourny, Watts et al], 2021

See [this paper]

See more in the backup slides

...and more

- transparent support for RNTuple, aka TTree 2.0 (faster, smaller) with no code changes
- machine learning inference as part of the event loop (see recent talk about SOFIE)
- support for TTree chains, friends, indexed friends, TEntryLists
- custom aggregations/results
- automatic cut-flow reports
- ...

Concluding remarks

Designed for you, with you

RDataFrame is a battle-tested, fast, versatile interface for modern HEP analysis.

RDataFrame (and ROOT) keeps **evolving**, in **cooperation with the community**.

With an ambitious plan of work, it is **critical to focus on the right features** - with your help!

Where to find us

Documentation

RDF user guide

RDF tutorials

New ROOT manual

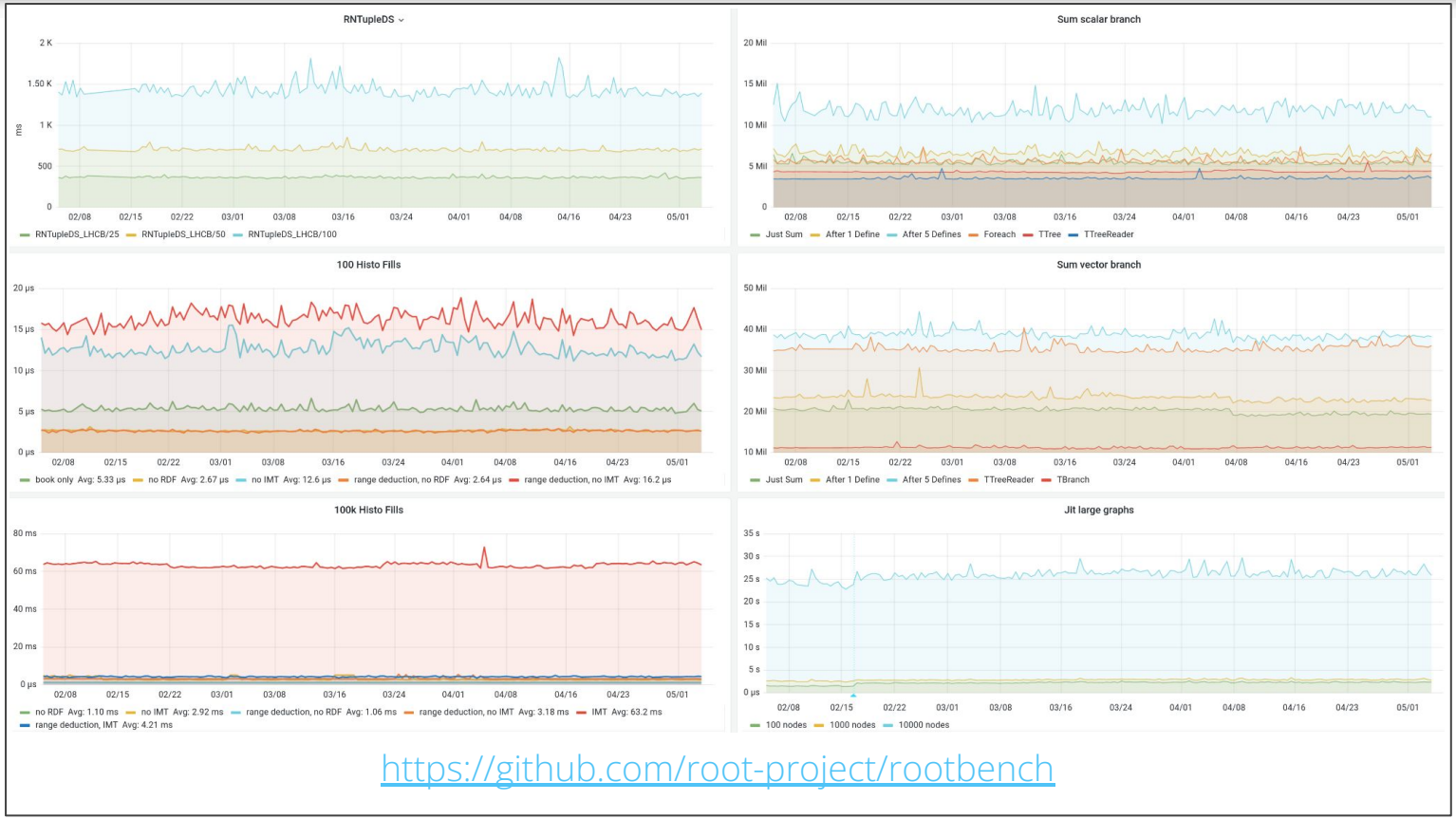
User support
root-forum.cern.ch

Bug reports
github.com/root-project/root/issues

Development discussion
mattermost.web.cern.ch/root

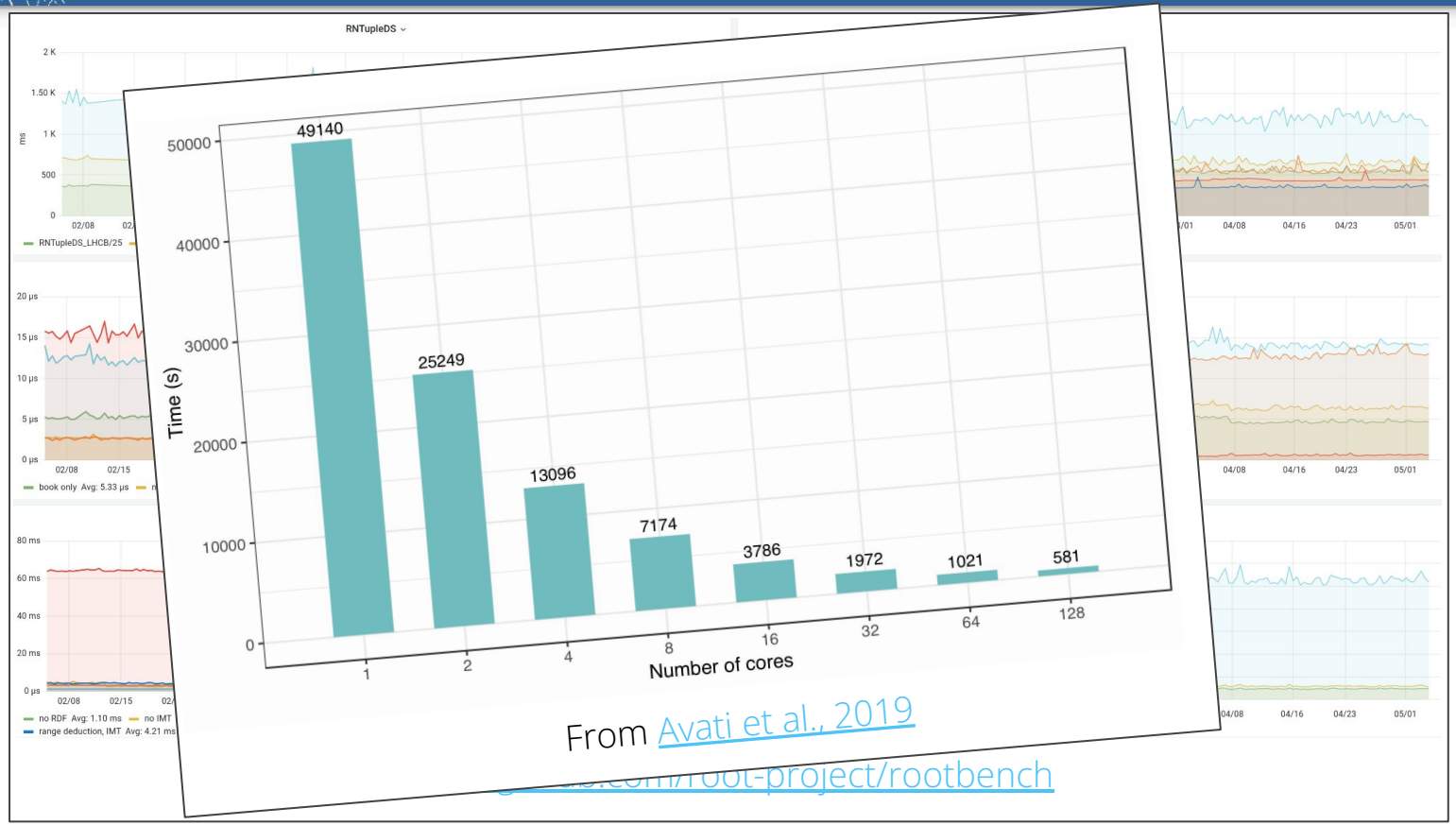
[BACKUP]
A note on performance

We care about performance. A lot.



https://github.com/root-project/rootbench

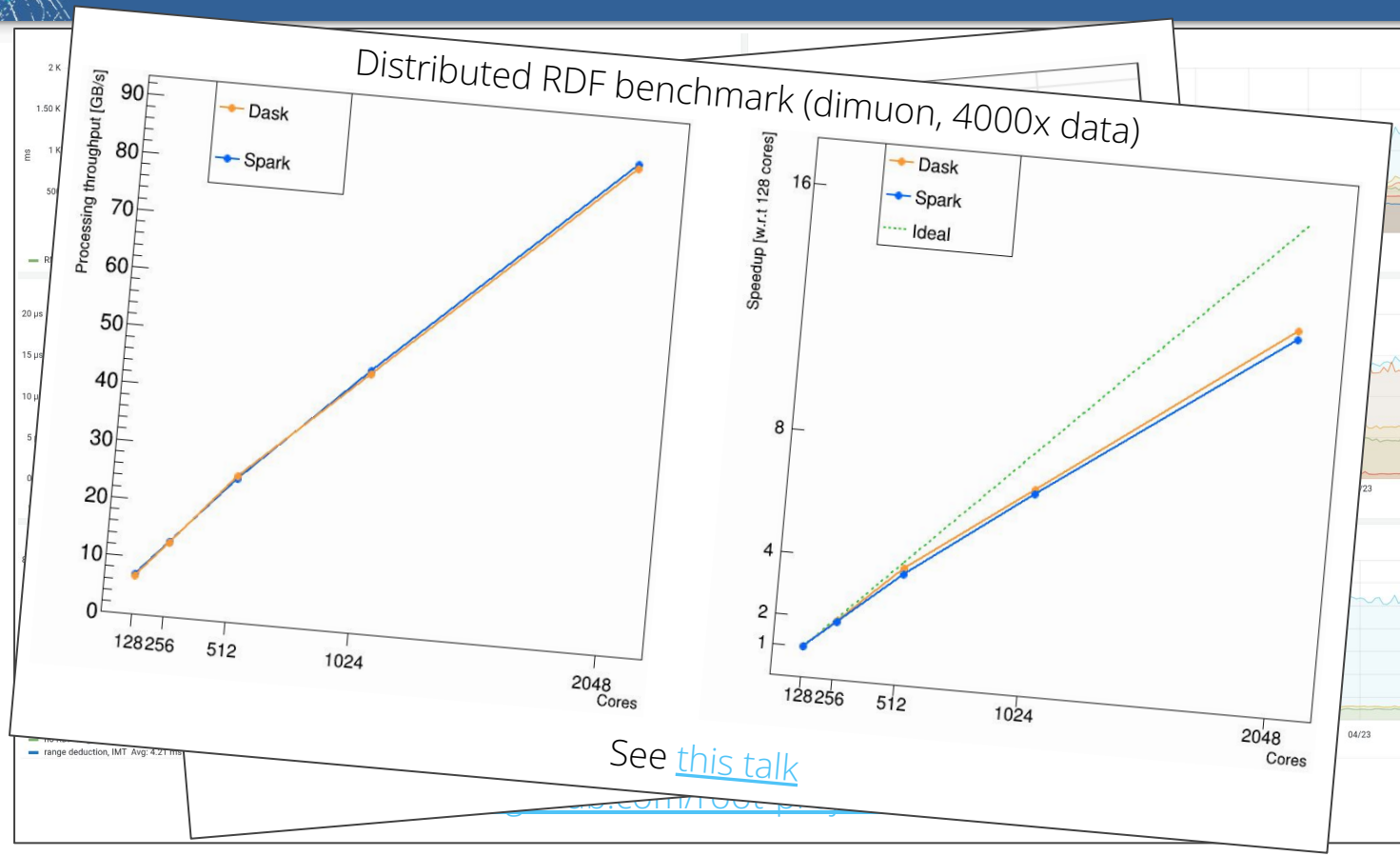
We care about performance. A lot.



From Avati et al., 2019

github.com/root-project/rootbench

We care about performance. A lot.



See [this talk]

We care about performance. A lot.

Fully compiled C++ RDataFrame (ROOT@db6a9d62f)

query,	1x data (s),	10x data (s)
Q1	0.37	1.50
Q2	0.46	3.70
Q3	0.73	6.23
Q4	0.65	5.92
Q5	0.84	7.45
Q6	3.08	27.99
Q7	2.56	22.27
Q8	1.17	10.22

Coffea 0.7.12 (using chunksize=2**19)

query,	1x data (s),	10x data (s)
Q1	1.40	4.24
Q2	1.51	5.76
Q3	1.81	7.96
Q4	1.65	6.58
Q5	2.41	12.43
Q6	13.89	124.59
Q7	4.19	29.12
Q8	3.27	17.70

- note that these benchmarks are not representative of large analysis workloads
- see also this ACAT talk by Nick Smith

Benchmark from github.com/nsmith-/coffea-benchmarks
Setup: AMD EPYC 7702P, using 48 physical cores, data read from filesystem cache

https://github.com/root-project/rootbench

We care about performance. A lot.

NanoAOD events processed at 400 kHz
when producing ~6k histograms.

zlib-compressed data read from local SSD
128 threads on 2x AMD EPYC 7742
~CMS Wmass analysis framework

"turnaround of a few hours for O(100) plots (thousands of histograms) of the CMS Run2 data on a batch system"
~bamboo

Hist Type	Hist Config	Evt. Loop	Total	CPUEff	RSS
ROOT THnD	10 x 103 x 5D	59m39s	74m05s	0.74	400GB
ROOT THnD	10 x 6D back	7m54s	25m09s	0.27	405GB
ROOT THnD	10 x 6D front	13m52s	30m27s	0.42	406GB
Boost ("sta")	10 x 6D back	7m07s	7m17s	0.90	9GB
Boost ("sta")	10 x 6D front	3m22	3m33s	0.86	9GB
Boost ("sta")	10 x (5D + 1-tensor)	1m54s	2m04s	0.81	9GB
Boost ("sta")	1 x (5D + 2-tensor)	1m32s	1m42s	0.77	9GB

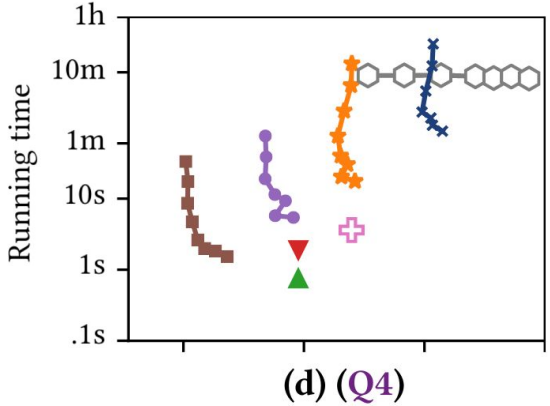
Processing lz4-compressed
ROOT data at 2 GB/sec

32 threads running on AMD Ryzen
Reading from a local NVME SSD disk
~CMS momentum correction

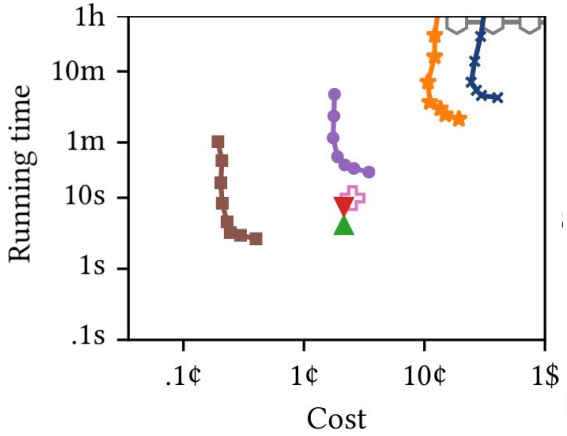
From this talk by Josh Bendavid

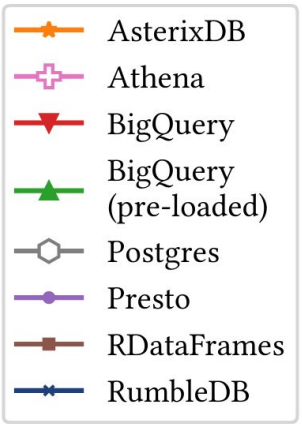
https://github.com/root-project/rootbench

We care about performance. A lot.



(d) (Q4)



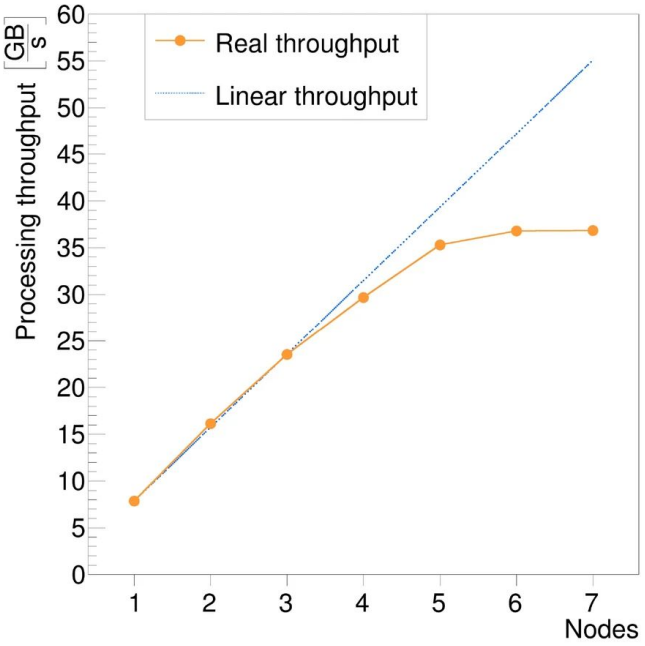


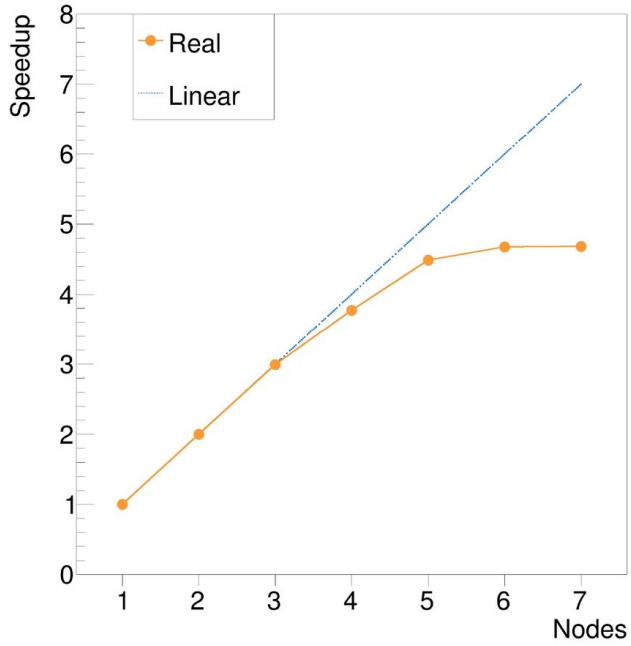
"...the general-purpose data processing systems are significantly less performant than the domain-specific ROOT framework—due to limited scalability and inefficient handling of the data and queries relevant to HEP"

~Graur, Muller, Proffit, Fourny, Watts et al, 2021

We care about performance. A lot.

Distributed RDataFrame w/ DAOS





See this paper

Performance: the bottom line

- Given users' feedback and our own benchmarks,
  RDataFrame enables fast turnaround for complex analysis use cases
- RDataFrame scales well to many cores, many nodes, many histograms
- Performance is always ongoing work: we are constantly looking for feedback/use cases

Wide adoption from analysts

- Dark matter sensitivity study (Pani & Polesello, 2018)
- Distributed analysis with RDataFrame in TOTEM (Avati et al., 2019)
- **ATLAS**: prototype xAOD data source DOI 10.5281/zenodo.1303038
- **ALICE**: Apache Arrow support contributed by G. Eulisse
- **FCC** is developing analysis workflows based on RDF (see also the GitHub project)
- Building block in INFN analysis facility effort
- many users **“in the wild”**: 650+ threads tagged #rdataframe on the ROOT forum, about the same as #tree and #hist

RDF as a framework building block

Some examples of analysis software based on RDataFrame

- bamboo (recent talk)
- KIT's CROWN (recent talk)
- W mass analysis framework
- LoopSUSYFrame ATLAS analysis tool
- ("Latinos" CMS framework planning transition to RDF)
- narf (recent talk)
- ...

Feedback (and code) from users regularly integrated upstream (*thank you*!)