ROOT
modern HEP data analysis

Vincenzo Eduardo Padulano (CERN, EP-SFT)
for the ROOT team

Workshop on "Quasi-Interactive Analysis of Big Data with High Throughput"
Bologna, Italy, 08.01.2025

Outlook

- The ROOT project
- Current activities
- Focus on analysis

Vincenzo Eduardo Padulano

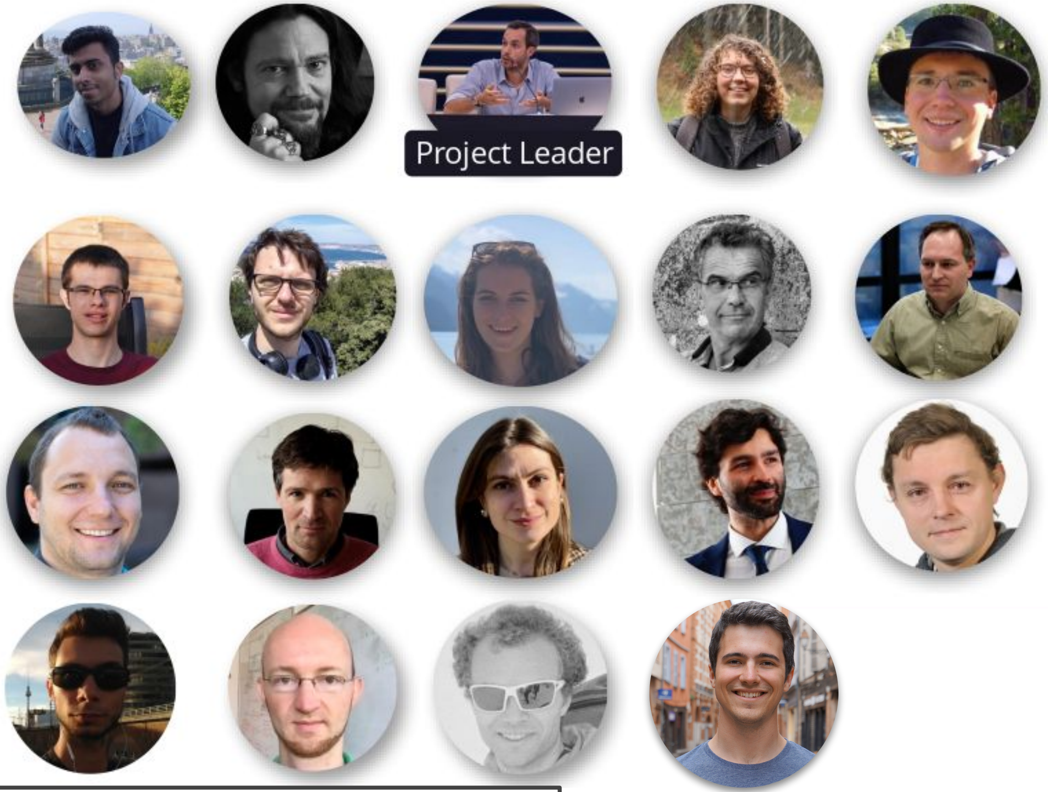
- PhD in Computer Science, Universitat Politècnica de València
- Staff Computing Engineer, CERN, EP-SFT
- Working in the ROOT team since 2019

The ROOT software project

https://root.cern

- Open-source software framework
  - **Storage**, data **analysis**, **processing**, visualization of **big** structured **datasets**
- **Widely adopted** in High Energy Physics and in other scientific and industrial fields
  - **Fits** and parameters estimations for discoveries (e.g. the Higgs)
  - **Thousands** of ROOT **plots** in scientific publications

The ROOT team



- ROOT is an **international** collaboration
- Steady **contributions** coming from the **community**, and institutional responsibilities.

CERN

Fermilab

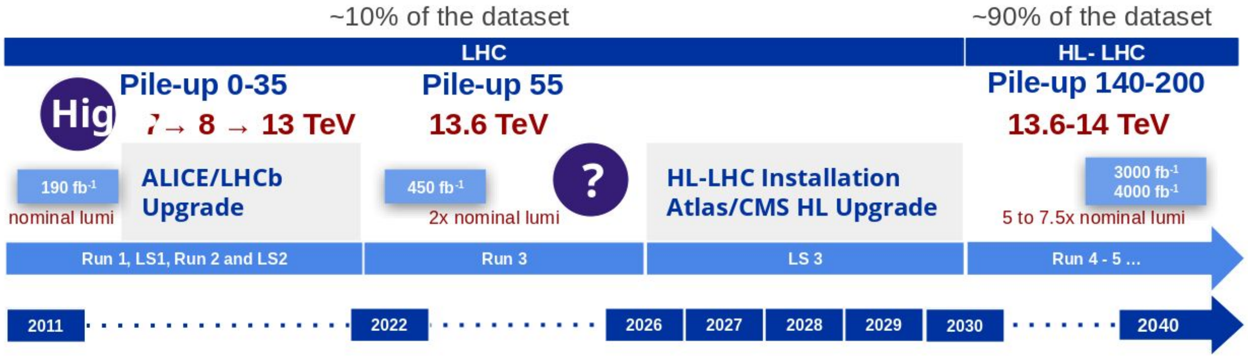
PRINCETON UNIVERSITY

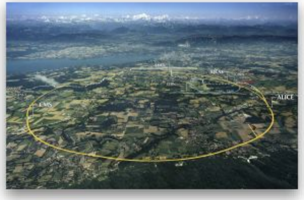
UNIVERSITY OF OSLO

GSI

https://root.cern/about/team/

The need for strategic thinking





HL-LHC timescale: the EIC will also start producing data!

QoS	ALICE	ATLAS	CMS	LHCb	*Total*
Disk [PB]	199	406	304	93	1002
Tape [PB]	283	666	673	250	1875

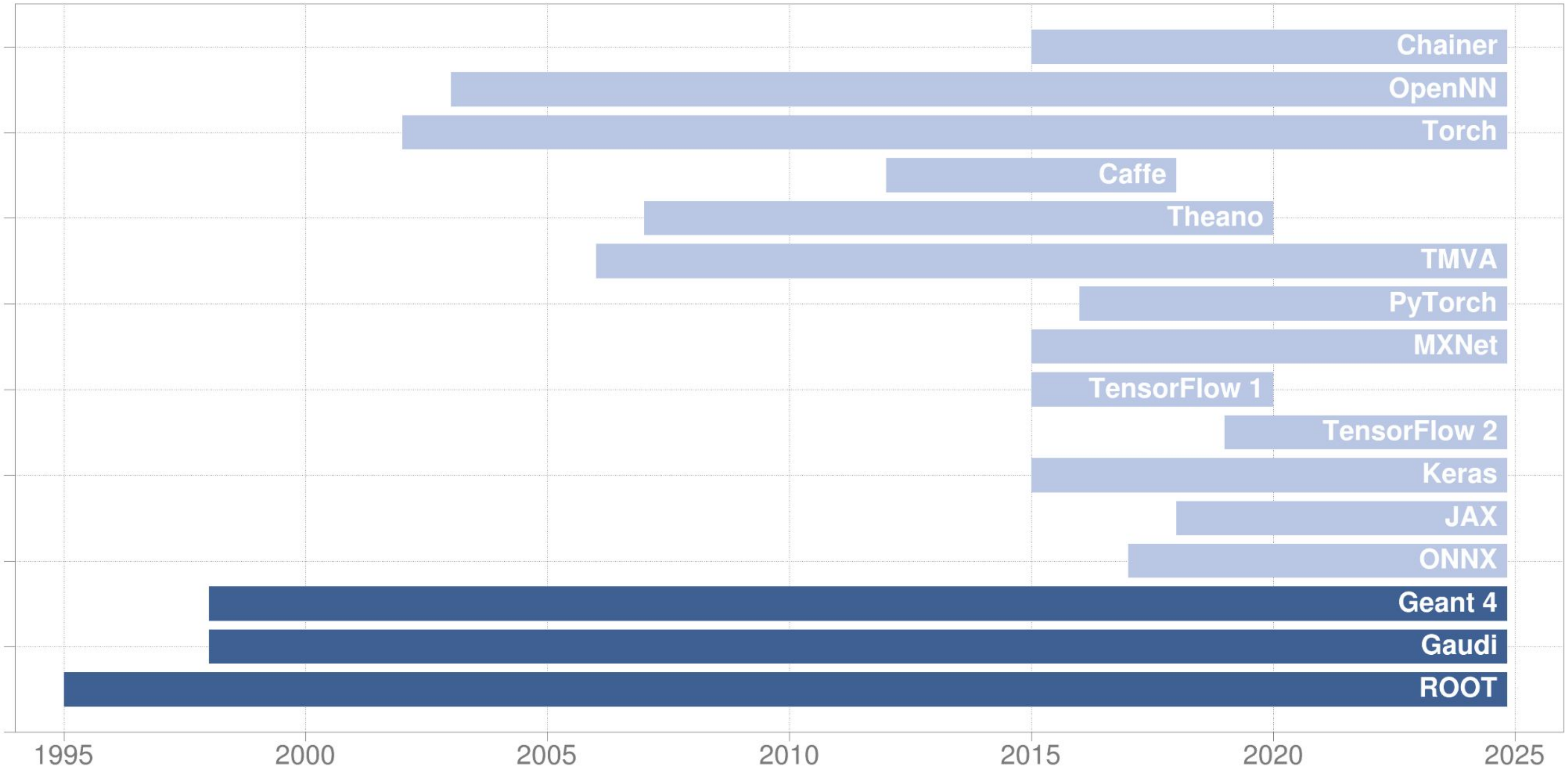
`24 Pledges: source WLCG CRIC

The full exploitation of the physics potential of present and future accelerators also passes through ROOT

LHC Runs 1,2,3 → **Today, >2 EB in ROOT format**

HL-LHC: ~30 EB in ROOT format?

Long-term support model



Plot inspired by [M. Mazurek]

Highlights of current activities

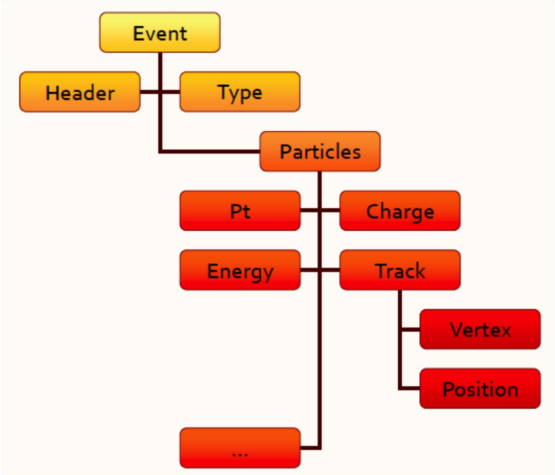
Requirements and challenges of HENP datasets

1. Natural HENP data layout is
   jagged arrays of complex types with columnar access pattern
   - HDF5 does not fit well due to its inherent tensor layout
   - Otherwise only found in Big Data (but with limited type support)

2. HENP data organization: global federation of file sets
   - Requires **XRootD** and **HTTP** remote data access
   - Extra functionality to build data sets from files:
     ***fast* merging, chains, joins**

3. Integration in the HENP software landscape
   - **Rich type system** of experiment central EDMs with **10k+ columns**
   - Multi-threaded reading and writing under tight memory constraints
   - Availability in the Python & C++ analysis ecosystem

4. >10 EB of data to be stored over decades
   - Requires **excellent compression** (lossy and lossless)
   - Data custodianship over time:
     backward & forward compatibility, schema evolution, bit-level checksumming



Dataset schema is the set of user-defined (C++) classes

RNTuple: next-generation data format

- A new data format, based on **25+ yrs** of experience with the established **TTree** data format, with a **modern** and **efficient** implementation:
  - **Smaller** files (typically **10% - 50%**), higher throughput (often by factors)
  - More **robust**: binary format specification, modern API, fully checksummed
  - Efficient support of **modern storage systems**: NVMe, object stores, async & parallel I/O
  - Forward-looking limits: designed for **TB-sized events** and **PB-sized files**
- Feature-rich: works with **complex** experiment **EDMs** and with experiment frameworks
- Supported at HL-LHC timescale **(2030-2040+)**

Rich type system support

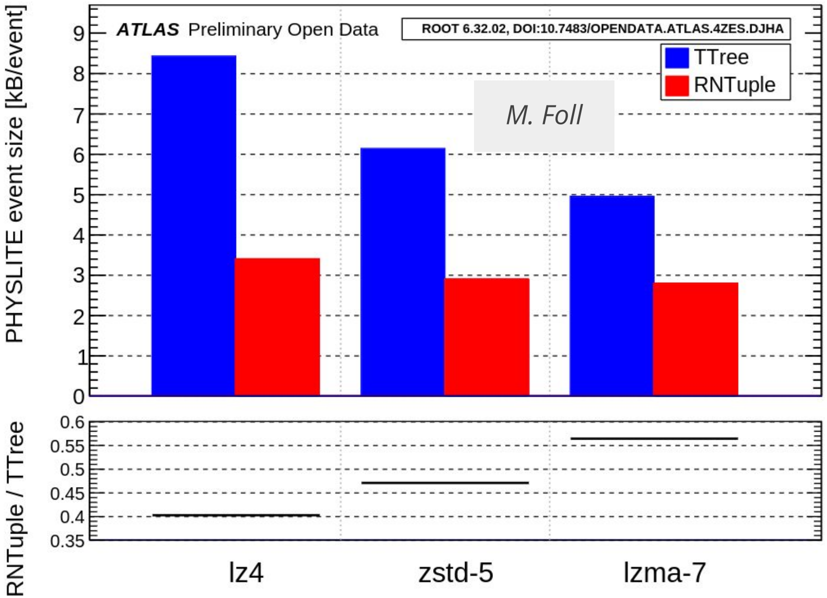
Type Class	Types	EDM Coverage			RNTuple Status
PoD	`bool, char, std::byte, (u)int[8,16,32,64]_t, float, double`				Available
Records	Manually built structs of PoDs	Flat n-tuple			
(Nested) vectors	`std::vector, RVec, std::array,` C-style fixed-size arrays		Reduced AOD		Available
String	`std::string`				Available
User-defined classes	Non-cyclic classes with dictionaries				Available
User-defined enums	Scoped / unscoped enums with dictionaries			Full AOD / ESD / RECO	Available
User-defined collections	Non-associative collection proxy				Available
stdlib types	`std::pair, std::tuple, std::bitset, std::(unordered_)(multi)set, std::(unordered_)(multi)map`				Available
Alternating types	`std::variant, std::unique_ptr, std::optional`				Available
Streamer I/O	All ROOT streamable objects (stored as byte array)				Available
Low-precision floating points	`Double32_t, f16`	*Optimization benefitting all EDMs*			Available
	Custom precision / range (bfloat16, TensorFloat-32, other AI formats)				Available

Rich type system support

Type Class	Types	EDM Coverage			RNTuple Status
PoD	`bool, char, std::byte, (u)int[8,16,32,64]_t, float, double`	Flat n-tuple	Reduced AOD	Full AOD / ESD / RECO	Available
Records	Manually built structs of PoDs				
(Nested) vectors	`std::vector, RVec, std::array,` C-style fixed-size arrays				Available
String	`std::string`				Available
User-defined classes	Non-cyclic classes with dictionaries				Available
User-defined enums	Scoped / unscoped enums with dictionaries				Available
User-defined collections	Non-associative collection proxy				Available
stdlib types	`std::pair, std::tuple, std::bitset, std::(unordered_)(multi)set, std::(unordered_)(multi)map`				Available
Alternating types	`std::variant, std::unique_ptr, std::optional`				Available
Streamer I/O	All ROOT streamable objects (stored as byte array)				Available
Low-precision floating points	`Double32_t, f16`	*Optimization benefitting all EDMs*			Available
	Custom precision / range (bfloat16, TensorFloat-32, other AI formats)				Available

Limit of HDF5 and Big Data formats (e.g., Parquet)

RNTuple space savings



Example: ATLAS DAOD

RNTuple in ATLAS [1] [2] [3]

Note that due to data preconditioning in RNTuple, the relative difference between compression algorithms fades.

Contributors to space savings

- More compact on-disk representation of collections and booleans (trigger bits)
- Same page merging
- Type-based data encoding optimized for better compression ratio

More performance studies

- CMS
- LHCb
- Comparison with HDF5 & Parquet (ACAT 21)

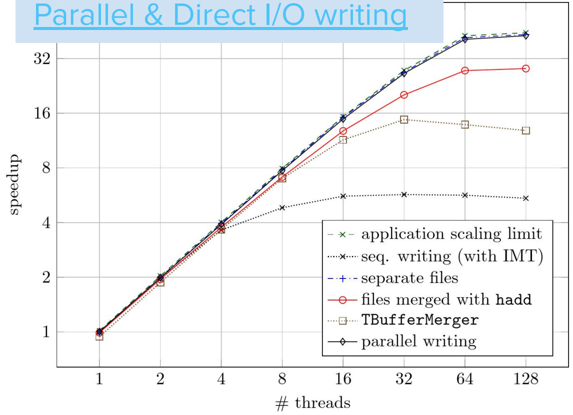
RNTuple throughput studies

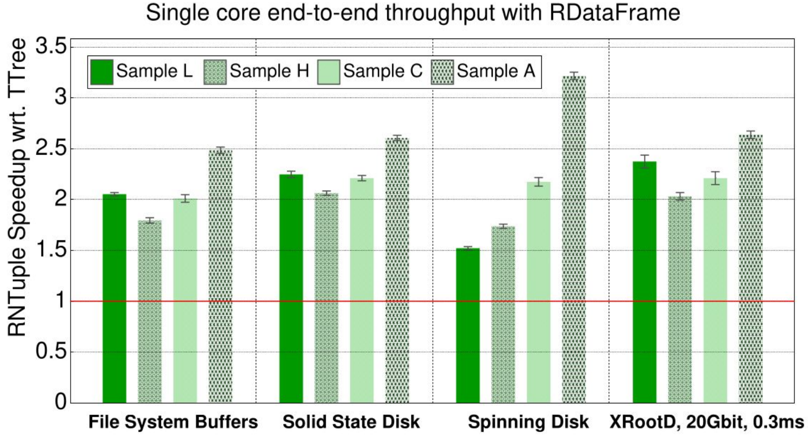
Higher analysis throughput across various final-stage ntuple types and data access modes.

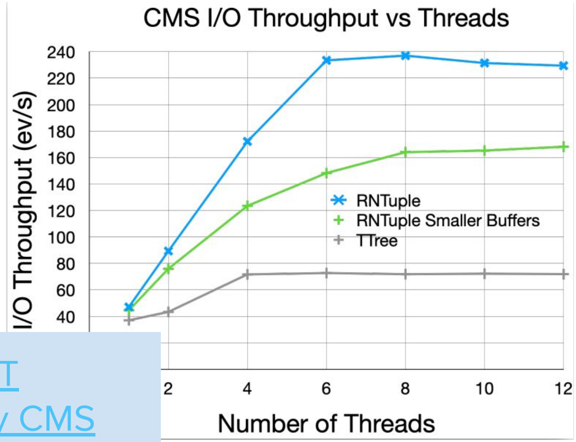
Contributors to higher throughput

- **Asynchronous** prefetching
- **Multi-stream disk access** through `io_uring`
- Code **optimization**
- New on-disk layout allows for higher degree of explicit and implicit **parallelization**
- New analysis I/O scheduler

Parallel & Direct I/O writing



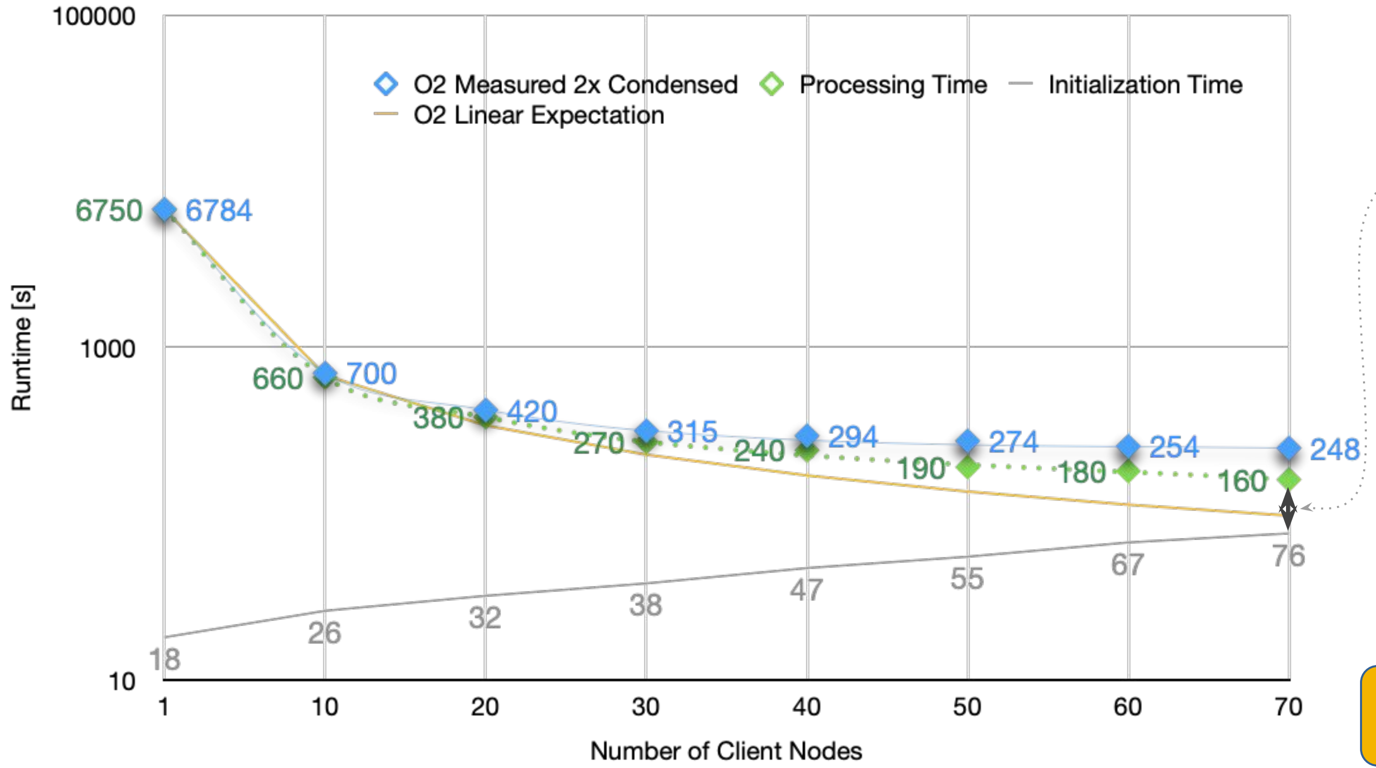




Better IMT scalability CMS

Large scale data analysis with RNTuple

AGC analysis, 100x inflated dataset, 32-core nodes, 2240 cores max



With a 100x inflated **AGC[200]** dataset we observe that as the number of client nodes increases, the initialization time gets close the processing time, resulting in a breakdown of scalability.

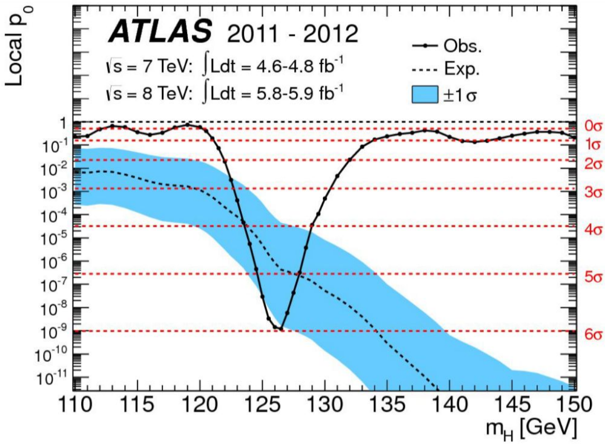
Single Analysis
extremely sparse
reaches avg. INGRES
222 GBit/s
during processing
345 GBit/s

RNTuple CHEP'24 Plenary

Likelihood building and evaluation

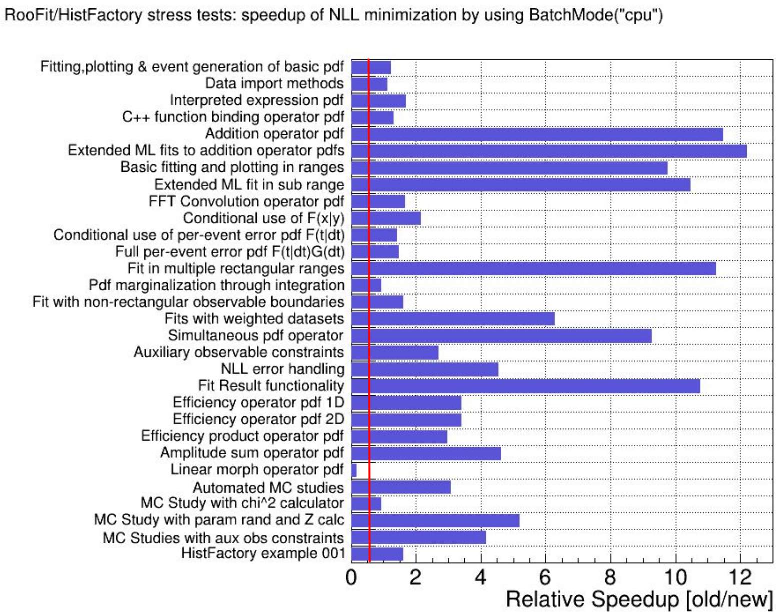
RooFit: C++ library for statistical data analysis in ROOT

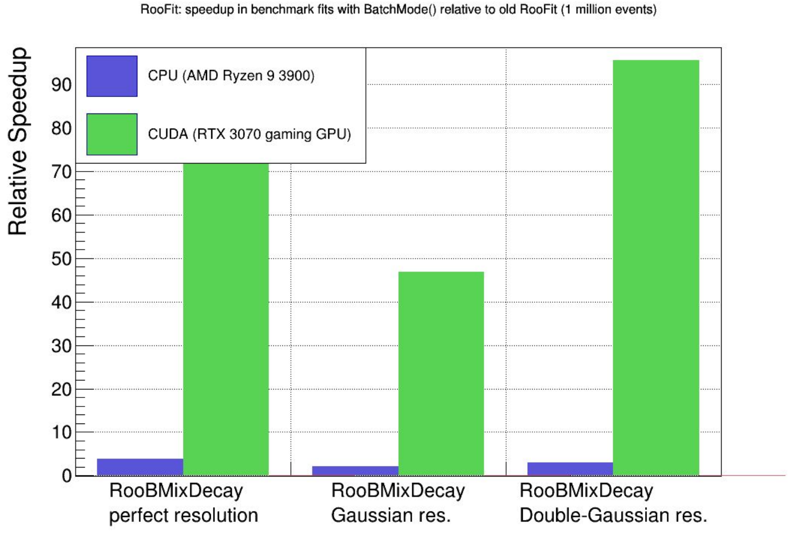
- Provides tools for model building, fitting and statistical tests
  - Sophisticated **binned** models with many nuisance parameters but few data entries
  - **Unbinned** fits of analytic shapes to huge datasets
- Recent development focused on:
  - **Performance** boost (preparing for larger datasets of HL-LHC)
  - More **user friendly interfaces** and high-level tools



RooFit performance

- Default **CPU** backend leverages vectorization (**4.4x** speedup on average, see plot on the left)
- The **GPU** backend can drastically speed up fits on large unbinned datasets
- See this [PyHEP 2023] presentation for more benchmarks
  - also compared to **zfit** and **pyhf**

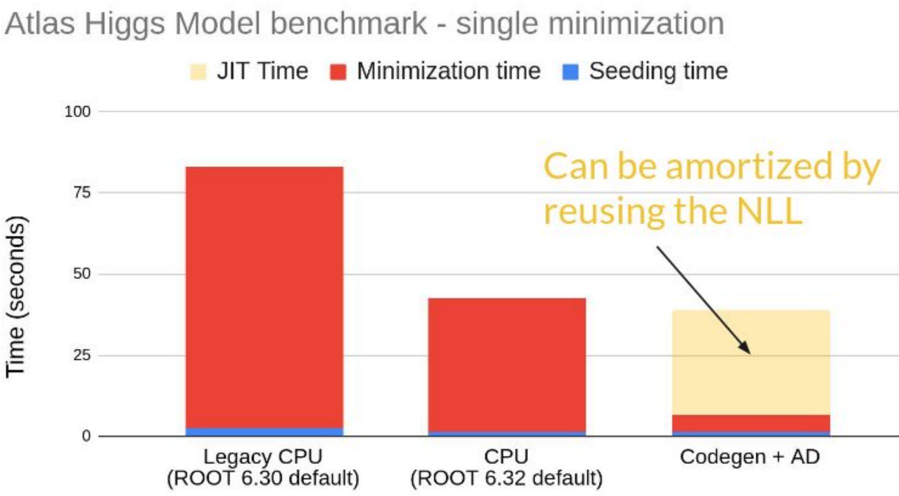




CPU and GPU speedup compared to legacy backend

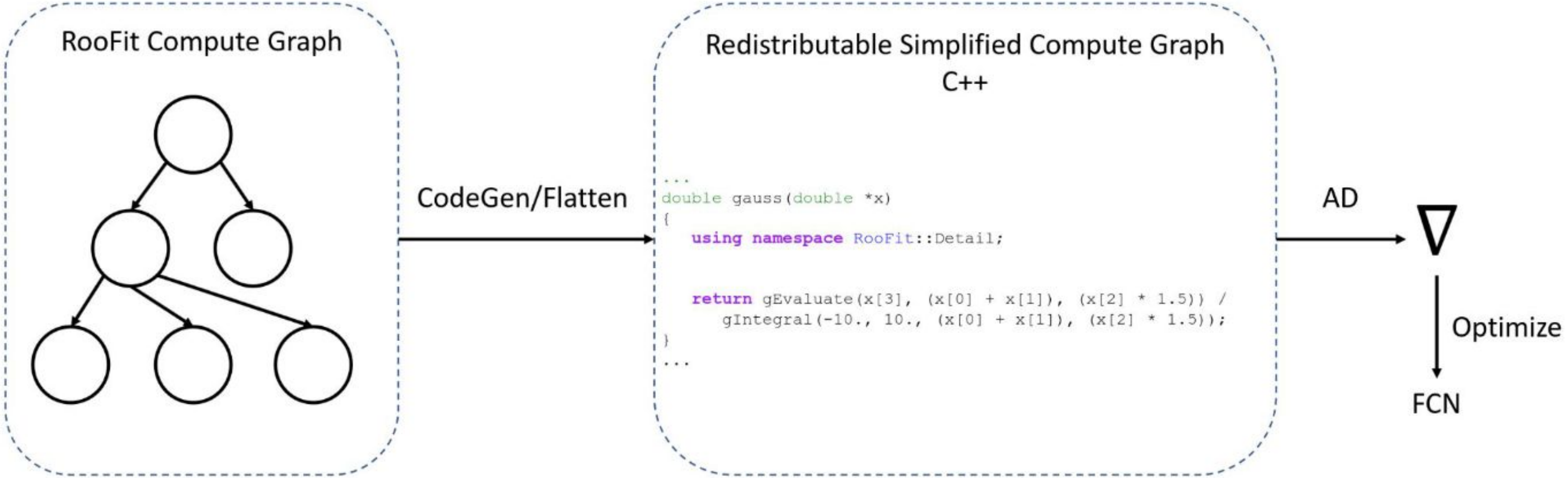
Automatic Differentiation in RooFit

- **RooFit** is a **framework** to build computation graphs for function **minimization**, similar to ML frameworks such as TensorFlow or PyTorch
- Recently, a **new** RooFit **backend** was added which leverages an **automatic differentiation** engine, based on the clad technology
- Result: evaluating **analytic** likelihood **gradients** without compromises



Rembser et al. ICHEP'24

How RooFit uses Clad to get analytic gradients: Code generation (aka. "codegen")



1. **Mathematical** concept
2. **RooFit** user code
3. **Automatic translation** of RooFit model to simple C++ code
4. **Gradient** of C++ code **automatically generated** with **Clad**
5. Gradient code **wrapped** back into RooFit object

Note: for the **nominal NLL** function, we **still use RooFits CPU backend** to benefit from vectorization and caching outside the gradients.

Rembser et al. ICHEP'24

The quality of the ROOT experience for Python users is a priority

- Update to the latest version of cppyy, ROOT's C++-Py 'interoperability engine'
- Provide a demo infrastructure to pip install ROOT
- Improve the usage of several classes from Python through "pythonisations"
- Teach ROOT through its Python interface, especially for beginners courses

More actions are planned for the future, e.g.:

- Revisit Python tutorials and code examples
- Improve and extend the Python interface through pythonisations
- Steadily publish ROOT releases on conda
- Evolve pip install ROOT to Beta mode during 2025 (e.g. automatic publication of wheels, multiple wheels…)

Seeing it in action

pip install ROOT -i https://root-experimental-python-wheels.web.cern.ch

```
$:docker run --rm -it python /bin/bash
root@6f40406ea5f2:/# python -m venv myenv
root@6f40406ea5f2:/# source myenv/bin/activate
(myenv) root@6f40406ea5f2:/# pip install ROOT -i https://root-experimental-python-wheels.w
eb.cern.ch
Looking in indexes: https://root-experimental-python-wheels.web.cern.ch
Collecting ROOT
  Downloading https://root-experimental-python-wheels.web.cern.ch/ROOT-0.1a6-cp313-cp313-m
anylinux_2_28_x86_64.whl.metadata (5.3 kB)
Downloading https://root-experimental-python-wheels.web.cern.ch/ROOT-0.1a6-cp313-cp313-man
ylinux_2_28_x86_64.whl (215.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 215.0/215.0 MB 113.7 MB/s eta 0:00:00
Installing collected packages: ROOT
Successfully installed ROOT-0.1a6
(myenv) root@6f40406ea5f2:/# python
Python 3.13.0 (main, Oct  8 2024, 00:06:32) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import ROOT
>>> df = ROOT.RDataFrame(10)
>>> df.Count().GetValue()
10
```

Bridging Python and C++

ROOT provides a **C++ interpreter**, cling

- Based on LLVM's clang compiler, now available upstream as clang-repl
- But cling still builds on top of clang-repl, the goal is to simplify this infrastructure
- e.g. through CppInterOp, that exposes APIs from Clang and LLVM in a backward compatible way

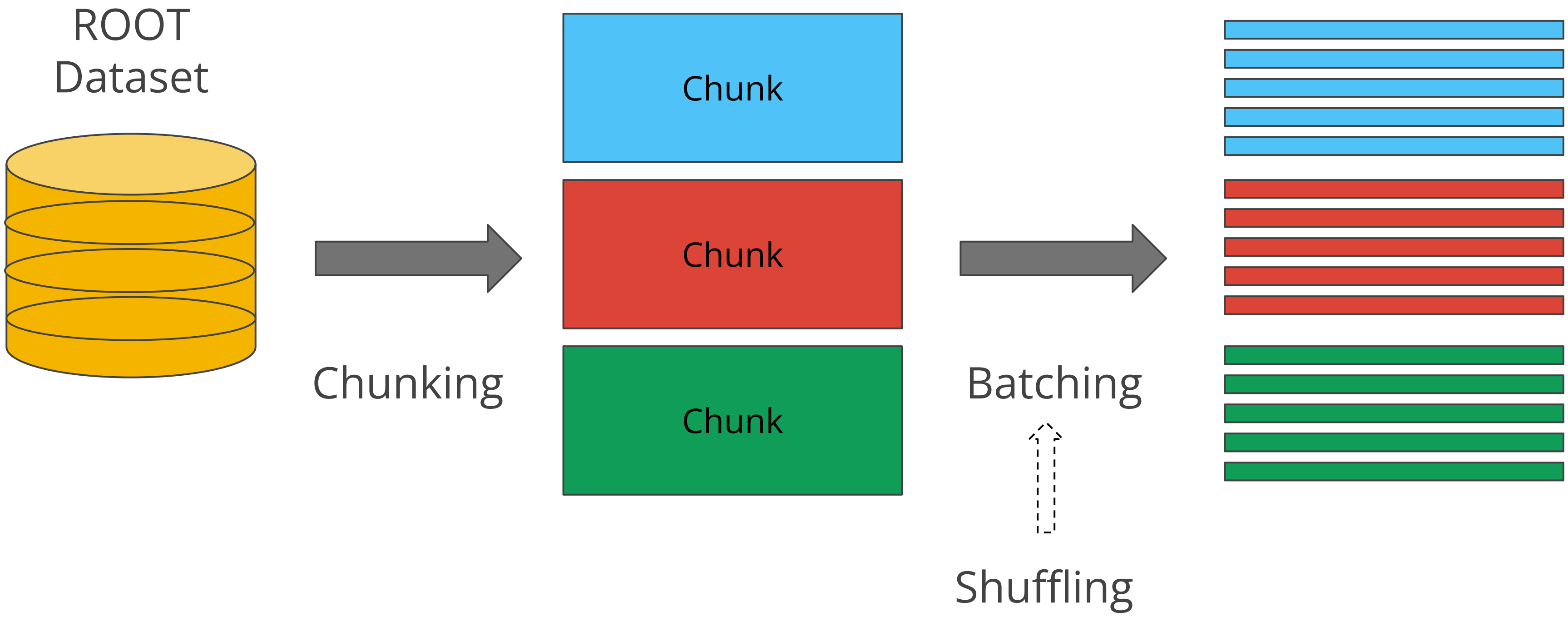
Stay even more **up-to-date** with **LLVM** versions

- ROOT 6.34 (November 2024) is based on LLVM 18.1 (March 2024)
- Exploit new features, e.g. performance, C++ standards
- Upstream Cling features to the LLVM repo with clang-repl

Continue **research** on C++ **compilers** and **language interoperability**

Native ROOT data loading for ML

Provide a native data loading abstraction to pipe ROOT data (TTree, RNTuple) into ML training workflows (e.g. PyTorch, TF)



Native ROOT data loading for ML

Provide a native data loading abstraction to pipe ROOT data (TTree, RNTuple) into ML training workflows (e.g. PyTorch, TF)

- **Asynchronous** loading (C++ thread)
- Supports **scalar** inputs as well as **collections**
- **Native** ROOT I/O: can read **any** HEP **EDM, local** or **remote** files
- **No** need for **pre-conversion** step to other data formats thus **no duplication**
- Integrated with **RDataFrame** for batch **preprocessing**

See CHEP'24 presentation

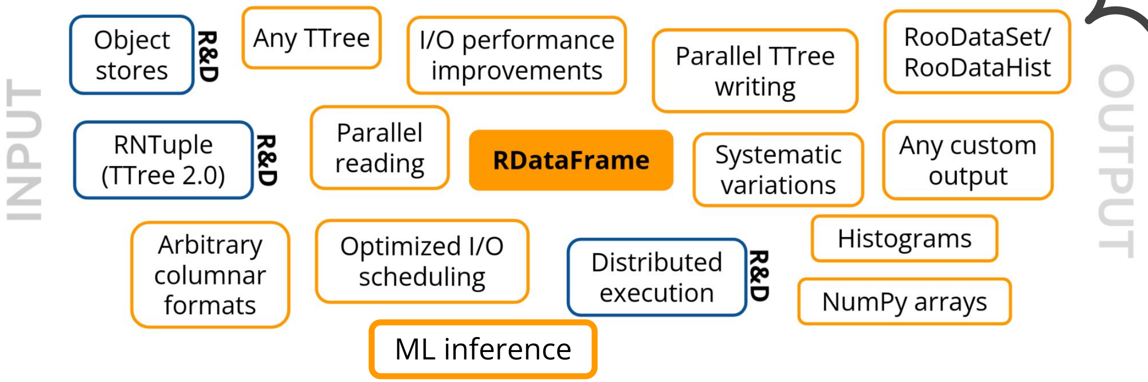
Native ROOT data loading for ML

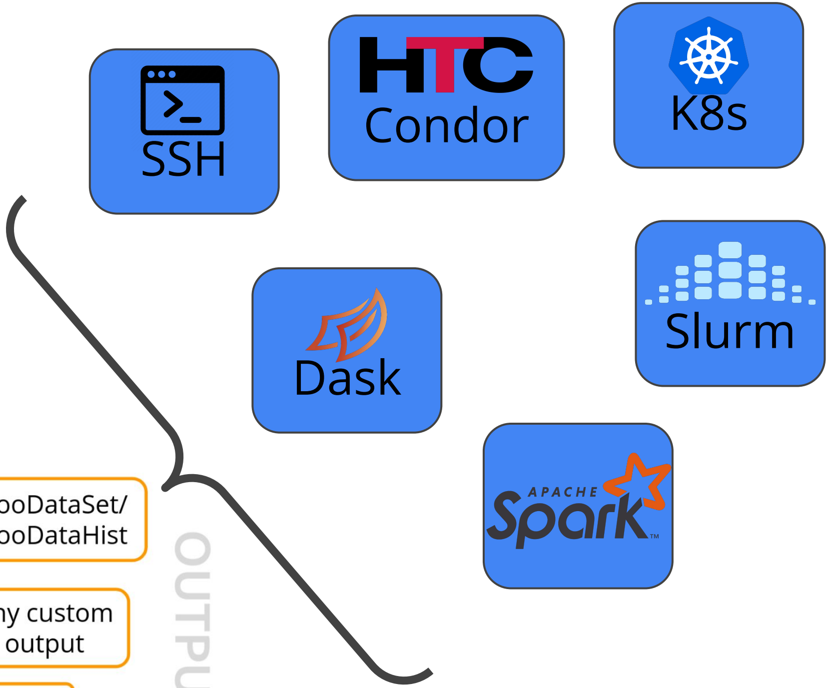
```python
# Returns two generators that return training
# and validation batches as PyTorch tensors.
gen_train, gen_validation =
ROOT.TMVA.Experimental.CreatePyTorchGenerators(
    rdataframe, batch_size, chunk_size,
    columns=features+labels, target=labels,
    max_vec_sizes=100, validation_split=0.3,
)
# [...] Create PyTorch model
for x_train, y_train in gen_train:
    # Make prediction and calculate loss
    pred = model(x_train)
    loss = loss_fn(pred, y_train)
```

Data analysis with ROOT

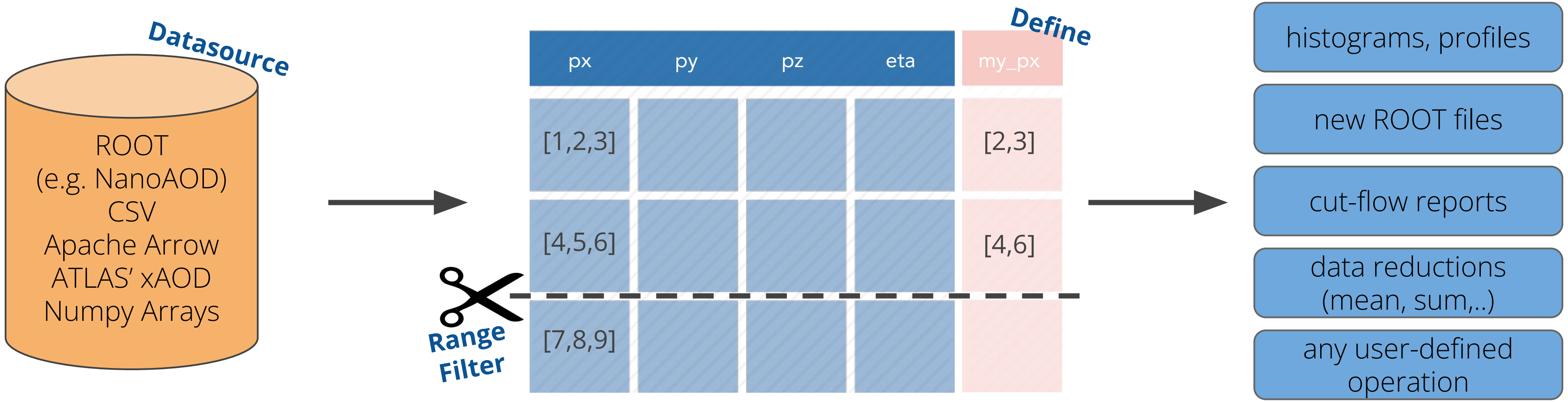
Data analysis with ROOT

- RDataFrame: entry point to **modern** ROOT **data analysis**
- HIgh-level interface
- Native parallel execution
  - Single node (MT), multi node





RDataFrame analysis interface




```
# enable multi-threading
ROOT.EnableImplicitMT()
df = ROOT.RDataFrame(dataset)
```

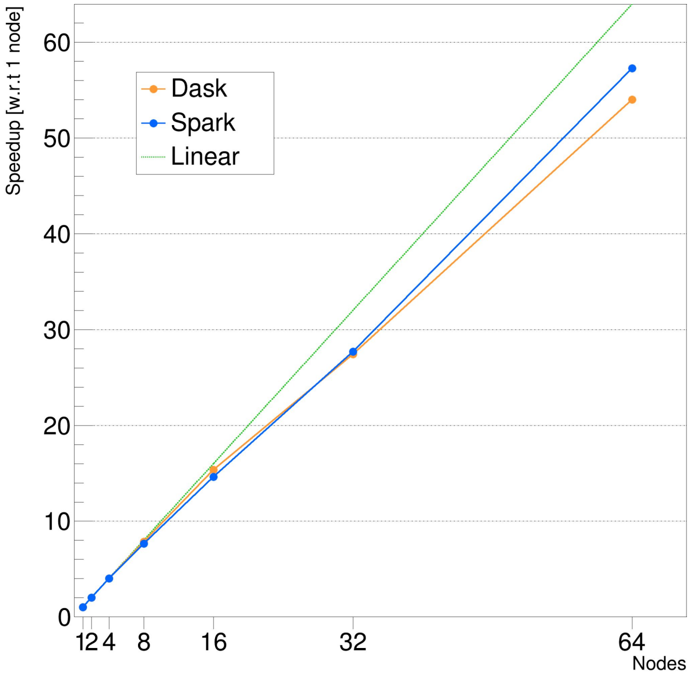
```
df = df.Range(2)
        .Define("my_px", "px[eta > 0]")
```

```
# filled in a single pass
h1 = df.Histo1D("my_px", "w")
h2 = df.Histo1D("px", "w")
```

RDataFrame + HPC centers

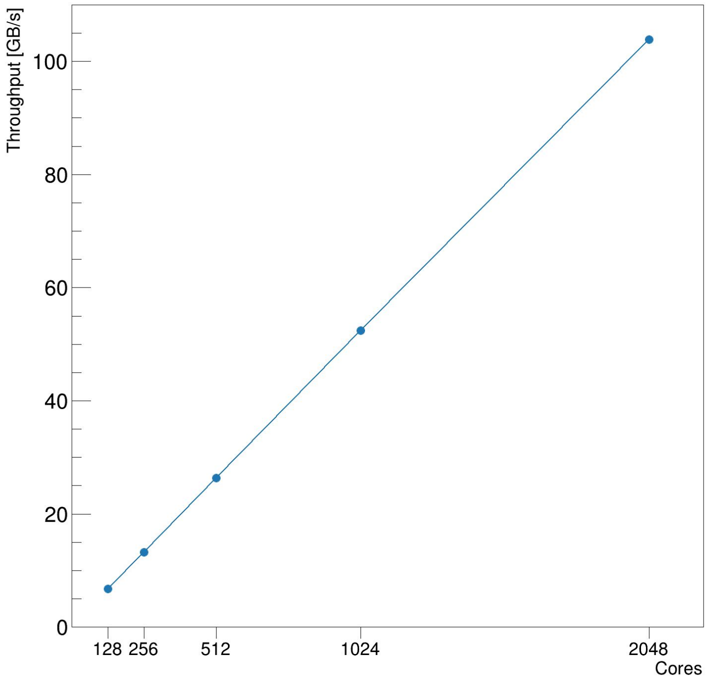
CERN HPC

- Slurm jobs (Spark/Dask)
- ~100 GB/s on 2048 cores
- JGC publication



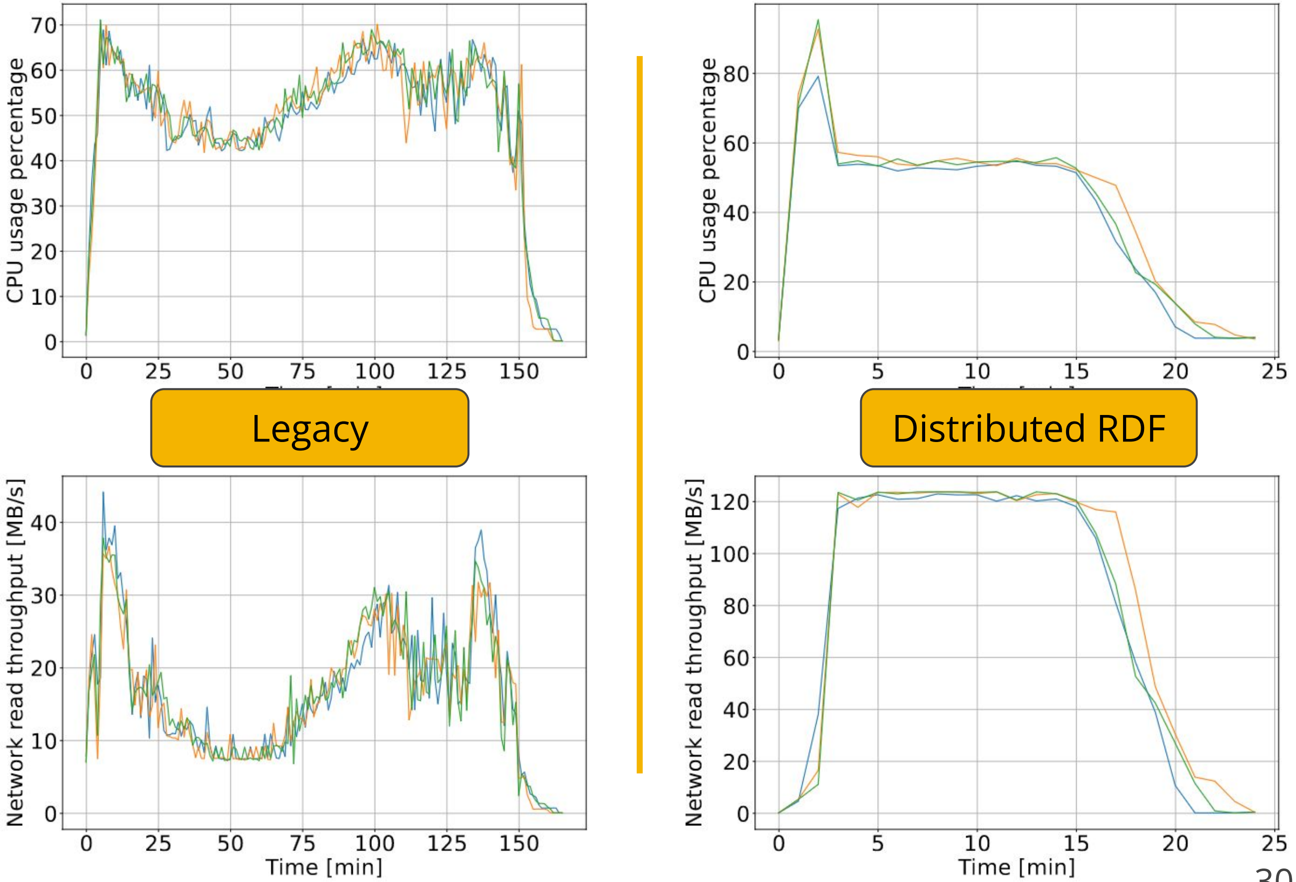
Jülich HPC

- Collaboration with OpenLab
- Slurm jobs (via Dask)
- Presentation

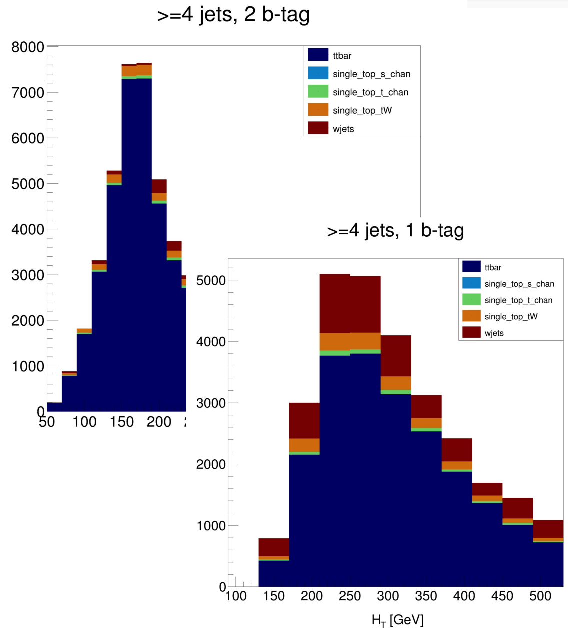


RDataFrame + INFN analysis facility

- CMS production analysis
- Before: Python **for-loop** with [NanoAODtools](), **manual** job submission
- After: **Interactive** distributed RDataFrame
- O(10) speedup
- [T. Tedeschi et al.]()

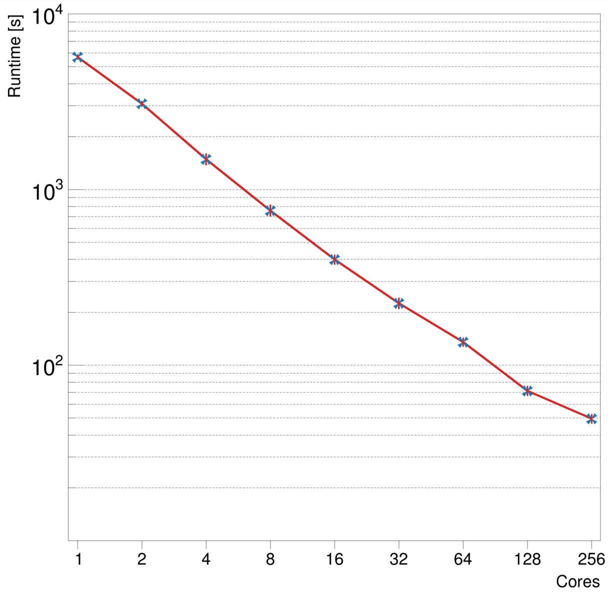


RDataFrame + Analysis Grand Challenge



RDF+AGC on CERN HPC

- Demonstrated scalability
- ~50 seconds for the whole analysis on 256 cores
- CHEP'23 presentation

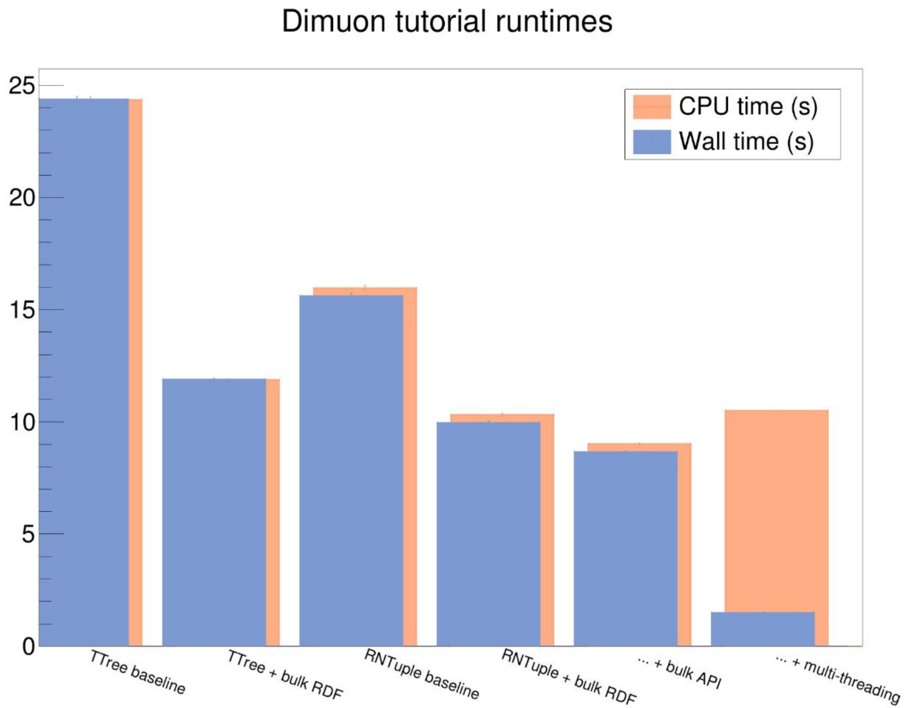


New! AGC on **SWAN**, scheduling with **Dask** on **CERN Condor** pools!
Rediscovering **existing** infrastructures and services in a modern way

cvmfs + EOS + CERN batch + ROOT $\stackrel{?}{=}$ CERN AF

RDataFrame + RNTuple

- Bulk, asynchronous I/O **and** bulk processing
  - Hide network latency
  - Enable SIMD on CPU, GPU offloading

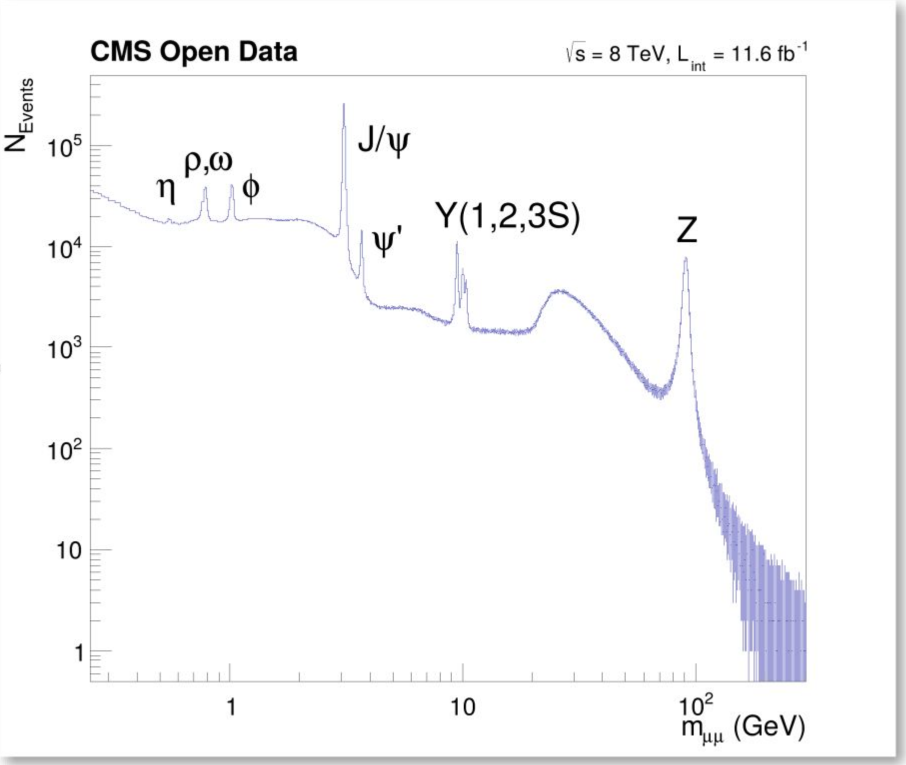


Moving to RNTuple with RDF: zero code changes

```python
import ROOT as rt

rt.EnableImplicitMT()

# Create dataframe from NanoAOD files
dataset = 'root://eospublic.cern.ch//eos/opendata/cms/derived-data/'\
          'AOD2NanoAODOutreachTool/Run2012BC_DoubleMuParked_Muons.root'
df = rt.RDataFrame ('Events', dataset)

# Select only events with exactly two muons and require opposite charge
df_2mu = df.Filter('nMuon == 2',
                   'Events with exactly two muons')
df_os = df_2mu.Filter('Muon_charge[0] != Muon_charge[1]',
                      'Muons with opposite charge')

# Compute invariant mass of the dimuon system
df_mass = df_os.Define('Dimuon_mass',
                       'InvariantMass(Muon_pt, Muon_eta, Muon_phi, Muon_mass)')

# Make histogram of dimuon mass spectrum. Note how we can set titles and axis labels in o
h = df_mass.Histo1D(('', 'Dimuon mass;m_{#mu#mu} (GeV);N_{Events}',
                     30000, 0.25, 300), 'Dimuon_mass')
# Request cut-flow report
report = df_mass.Report()

# Produce plot
rt.gStyle.SetOptStat(0); rt.gStyle.SetTextFont(42)
c.SetLogx(); c.SetLogy()
h.GetXaxis().SetTitleSize(0.04)
h.GetYaxis().SetTitleSize(0.04)
h.Draw()

label = rt.TLatex(); label.SetNDC(True)
label.DrawLatex(0.175, 0.740, '#eta'); label.DrawLatex(0.205, 0.775, '#rho,#omega')
label.DrawLatex(0.270, 0.740, '#phi'); label.DrawLatex(0.400, 0.800, 'J/#psi')
label.DrawLatex(0.415, 0.670, '#psi'); label.DrawLatex(0.485, 0.700, 'Y(1,2,3S)')
label.DrawLatex(0.755, 0.680, 'Z')
label.SetTextSize(0.04); label.DrawLatex(0.10, 0.92, '#bf{CMS Open Data}')
label.SetTextSize(0.03); label.DrawLatex(0.63, 0.92, '#sqrt{s} = 8 TeV, L_{int} = 11.6 fb^{-1}')
```

TTree Version

[Dimuon analysis tutorial]

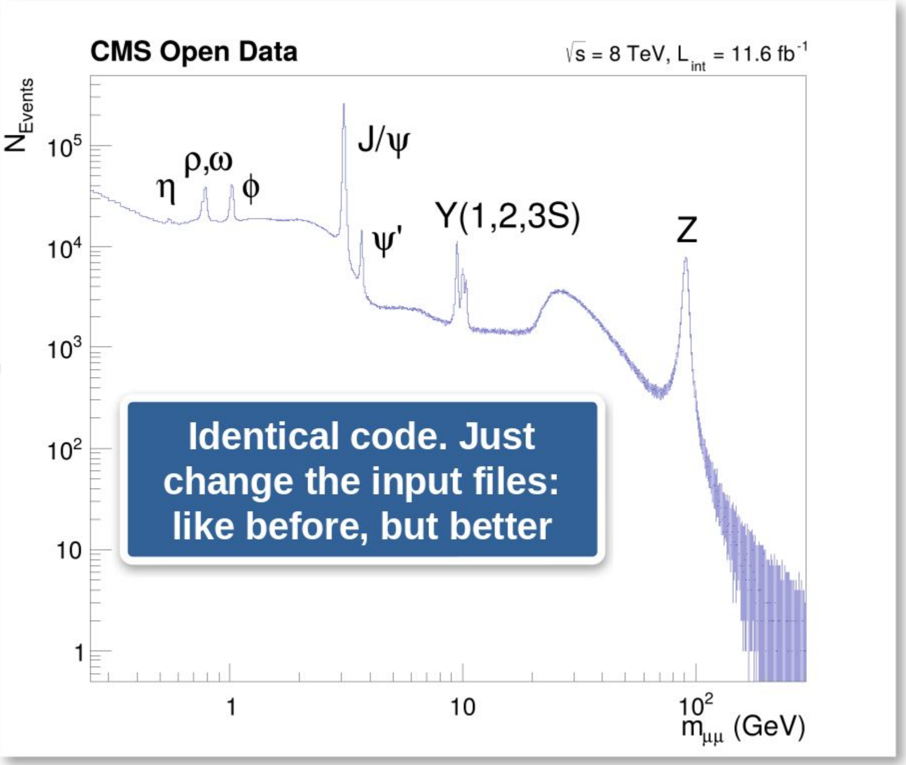


Moving to RNTuple with RDF: zero code changes

```
import ROOT as rt

rt.EnableImplicitMT()

# Create dataframe from NanoAOD files
dataset = 'http://root.cern/files/tutorials/ntpl004_dimuon_v1rc2.root'

df = rt.RDataFrame ('Events', dataset)

# Select only events with exactly two muons and require opposite charge
df_2mu = df.Filter('nMuon == 2',
                   'Events with exactly two muons')
df_os = df_2mu.Filter('Muon_charge[0] != Muon_charge[1]',
                      'Muons with opposite charge')

# Compute invariant mass of the dimuon system
df_mass = df_os.Define('Dimuon_mass',
                       'InvariantMass(Muon_pt, Muon_eta, Muon_phi, Muon_mass)')

# Make histogram of dimuon mass spectrum. Note how we can set titles and axis labels in o
h = df_mass.Histo1D(('', 'Dimuon mass;m_{#mu#mu} (GeV);N_{Events}',
                     30000, 0.25, 300), 'Dimuon_mass')
# Request cut-flow report
report = df_mass.Report()

# Produce plot
rt.gStyle.SetOptStat(0); rt.gStyle.SetTextFont(42)
c.SetLogx(); c.SetLogy()
h.GetXaxis().SetTitleSize(0.04)
h.GetYaxis().SetTitleSize(0.04)
h.Draw()

label = rt.TLatex(); label.SetNDC(True)
label.DrawLatex(0.175, 0.740, '#eta'); label.DrawLatex(0.205, 0.775, '#rho,#omega')
label.DrawLatex(0.270, 0.740, '#phi'); label.DrawLatex(0.400, 0.800, 'J/#psi')
label.DrawLatex(0.415, 0.670, '#psi'); label.DrawLatex(0.485, 0.700, 'Y(1,2,3S)')
label.DrawLatex(0.755, 0.680, 'Z')
label.SetTextSize(0.04); label.DrawLatex(0.10, 0.92, '#bf{CMS Open Data}')
label.SetTextSize(0.03); label.DrawLatex(0.63, 0.92, '#sqrt{s} = 8 TeV, L_{int} = 11.6 fb^{-1}')
```

RNTuple Version

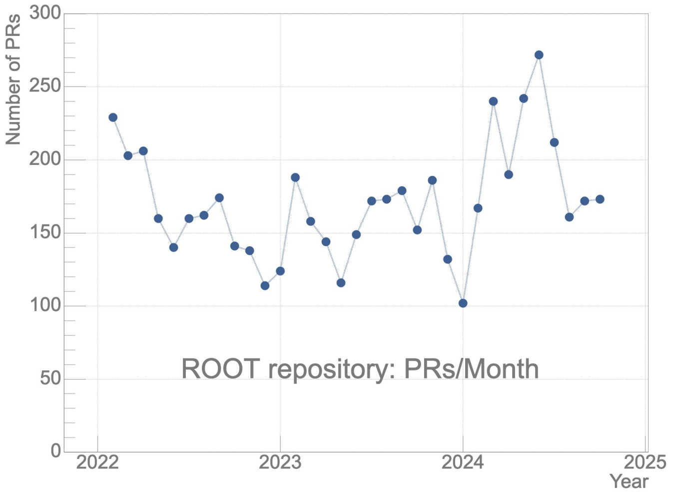
[Dimuon analysis tutorial]



Outreach

An Open approach to boost collaboration

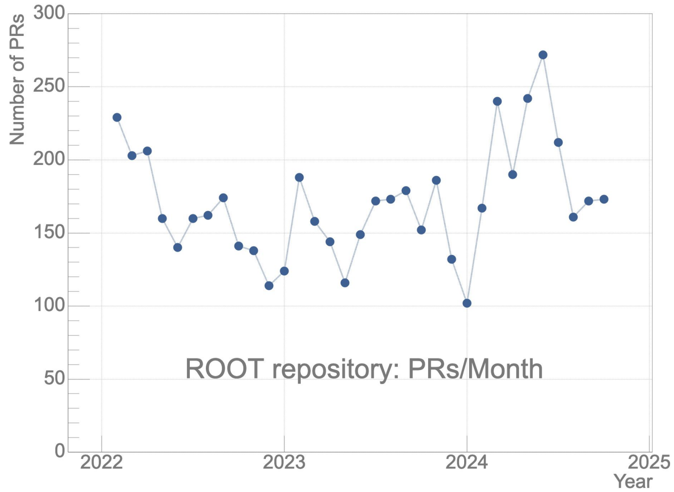
- Open-source and Open-development
  - On GitHub, LGPL 2.1
  - PR based model with public review process
  - Very visible authorship of contributions
- **Open-planning**: https://cern.ch/root-pow
- Yearly plan of work (**PoW**) formed internally, then **discussed** with users
- You can **influence** the PoW, with **your input**, active **engagement** and **contributions**!
- Formal reporting process engaging experiments



An Open approach to boost collaboration

- Open-source and Open-development
  - On GitHub, LGPL 2.1
  - PR based model with public review process
  - Very visible authorship of contributions
- **Open-planning**: https://cern.ch/root-pow
- Yearly plan of work (**PoW**) formed internally, then **discussed** with users
- You can **influence** the PoW, with **your input**, active **engagement** and **contributions**!
- Formal reporting process engaging experiments

SFT Plans of Work Meeting 22.01.2025

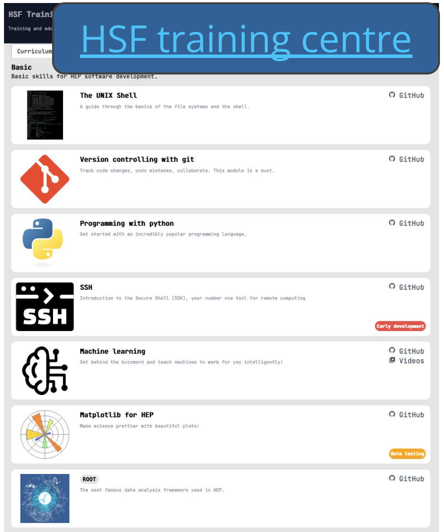


Scaling up the ROOT training

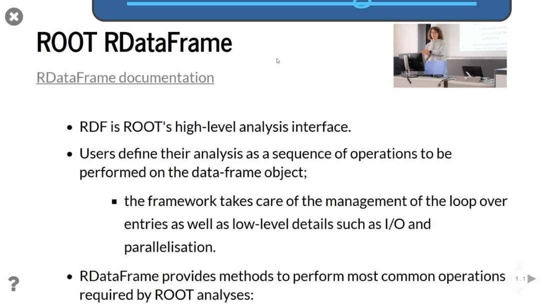
- CERN Summer Student Course ~200 participants, ~5 per year
- With IRIS-HEP and HSF: Python for analysis course ~90 participants, ~3 per year
- Based exclusively on ROOT's Python interface and notebooks
- Besides the value of the trainings themselves:
  - Surveys: feedback received now incorporated in the material
  - Several ROOT devs involved: we trained to train!

The ROOT team is available to give trainings and help train the trainers

[HSF training centre]



[ROOT training video]



ROOT Hackathons

- New in 2024!
- An event born for ROOT core devs, open to everybody
- A welcoming, positive and inclusive atmosphere
- 1st Hackathon in February 2024
  - Nickname “Fixathon”, aim at fixing various github issues
- 2nd Hackathon 25-27 November 2024
  - Topic: Python, Docs, Tutorials
- In presence only, very informal
  - Currently cannot provide sponsorships for attendance

ROOT Users Workshop 2025

- A welcoming, positive and inclusive atmosphere
- An opportunity to shape *together* the future of ROOT!
- A venue for ROOT users, world-class experts of scientific computing and the ROOT core team to *exchange ideas* and learn from each other
- A *rich program* of presentations, tutorials, and most importantly, discussions

In Europe
17-21 November 2025
Save the date!

Conclusions

Conclusions

- The **ROOT** core team is here to **support** you, listen to your needs and make your data processing and analysis a success!
  - All sustained by a **long-term support model**, and a **rich** result-oriented **R&D** program
- **Open approach**: open-source, open-development, open-planning
  - For ROOT, collaborations and contributions are essential and highly valued!
- **Forward-looking**: a core set of **modern** features to support **HL-LHC** and future colliders

Useful links

- ROOT web page: root.cern
- ROOT GitHub: github.com/root-project/root
- Careers at CERN: https://careers.cern/
- Email: vincenzo.eduardo.padulano@cern.ch