# ROOT
# Present and Future

*Lorenzo Moneta*
*(CERN )*
on behalf of the ROOT team

**20-21 May 2019, Trieste**

# Outline

- Current role of ROOT
- Vision for the future
  - what we will have in ROOT 7
  - current progress and planned new developments
- Conclusions

A lot of material presented coming from latest <u>ROOT Users Workshop</u> presentations (2018)

# ROOT in a Nutshell

- ROOT is a software framework with building blocks for:
  - **Data processing**
  - **Data analysis**
  - **Data visualisation**
  - **Data storage**
- ROOT is mainly written in C++ (C++11/17 standard)
  - Bindings for Python available as well
- Adopted in High Energy Physics and other sciences and also industry
  - more than 1 Exabyte of data in ROOT format
  - Data analysis (machine learning), parameters estimations and discovery significances (e.g. the Higgs)
  - Thousands of ROOT plots in scientific publications

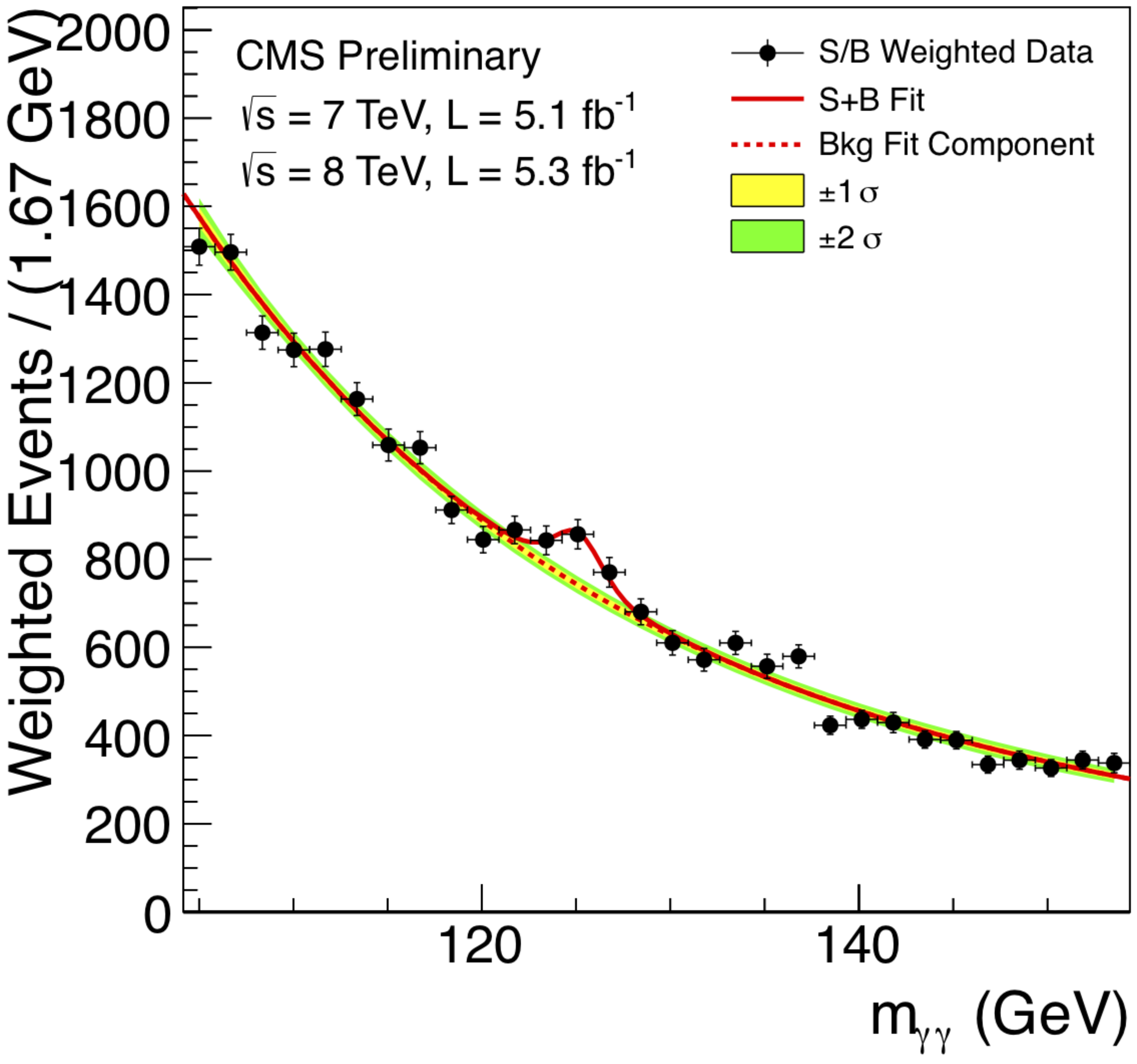
**An Open Source Project**
**We are on github**
**github.com/root-project**
**All contributions are warmly welcome!**

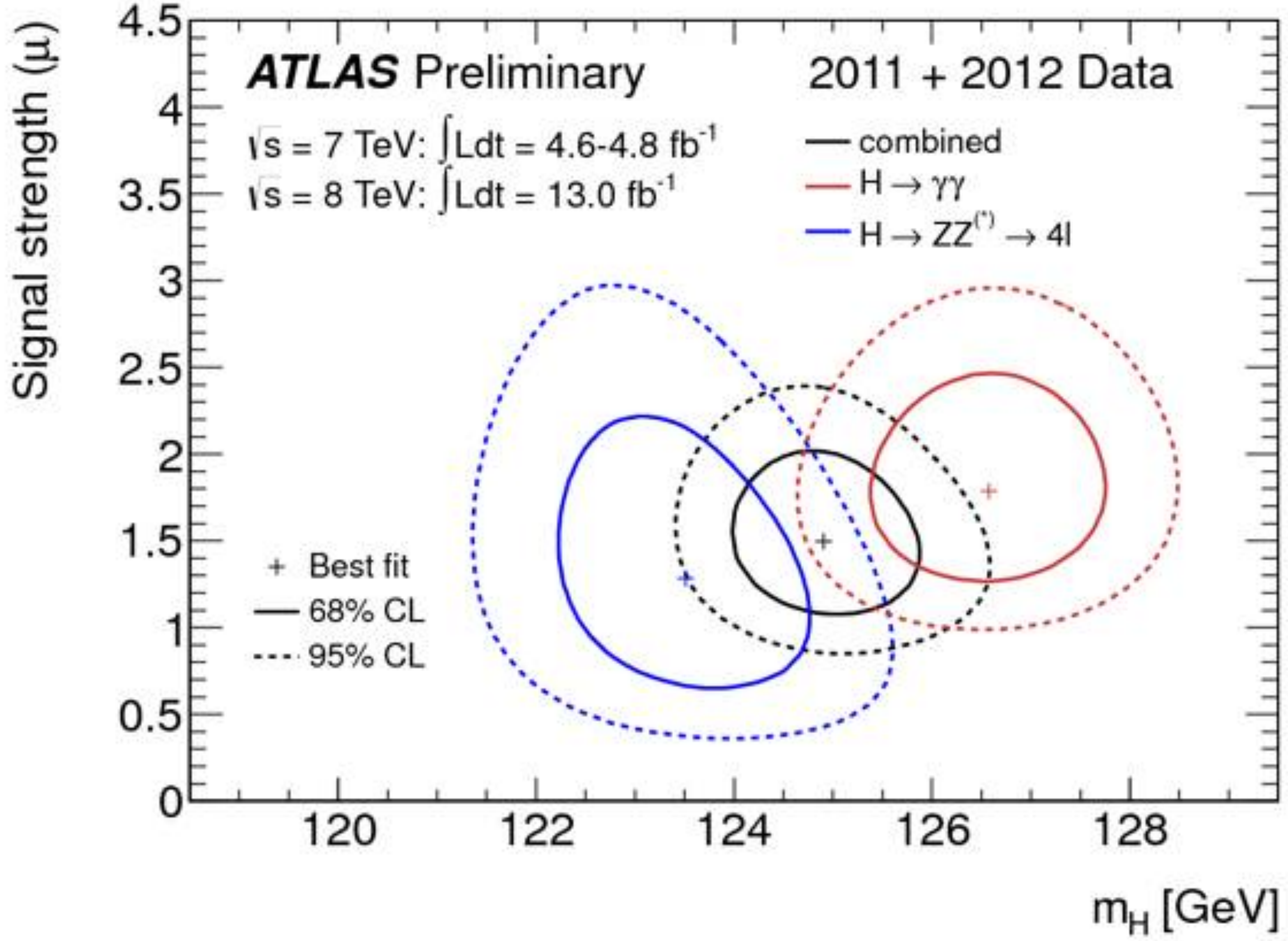
# ROOT Components

ROOT can be seen as a collection of building blocks for various activities, like:

- **Data analysis**: histograms, graphs, functions
- **I/O**: row-wise, column-wise storage of any C++ object
- **Statistical tools**: rich modeling tools and statistical inference
- **Math**: math functions, linear algebra and minimisation algorithms
- **C++ interpretation**: full language compliance
- **Multivariate Analysis (TMVA)**: e.g. Boosted decision trees, Neural networks (including deep learning)
- **Advanced graphics** (2D, 3D, event display)
- **Declarative Analysis**: Data Frame for event filtering and selection
- And more: HTTP serving,  JavaScript visualisation

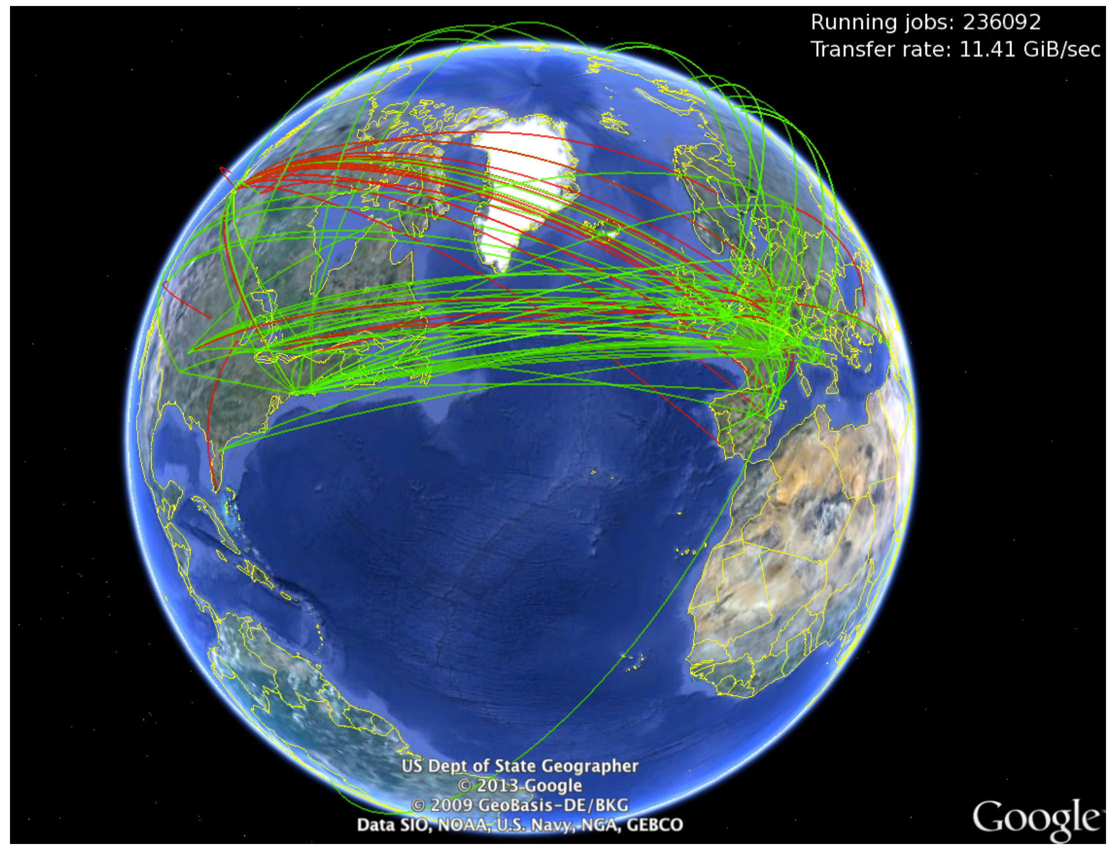
# What can you do with ROOT?



# ROOT Utilization

- ROOT is a centrepiece of HEP software
  - almost every high energy physicist uses ROOT for data analysis
  - central part for software of all experiments
    - running 24/7 on the computing grid
    - more than one EB of data in ROOT format
      - thanks to unique capabilities of ROOT I/O
  - Modular and versatile toolkit:
    - allow to develop specific tools for experiment and across them
    - common ground for physicists developing software

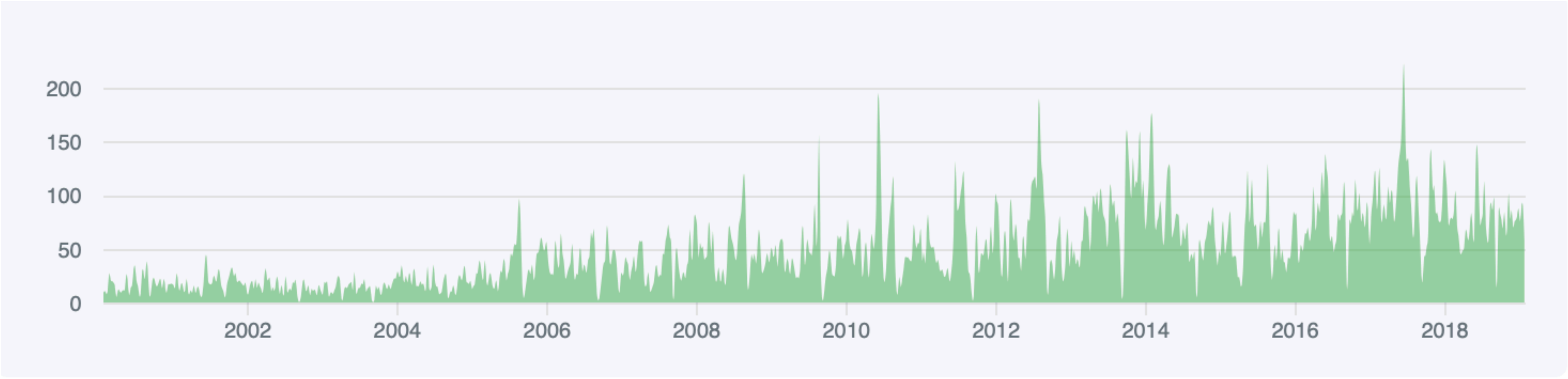


# Current ROOT Team

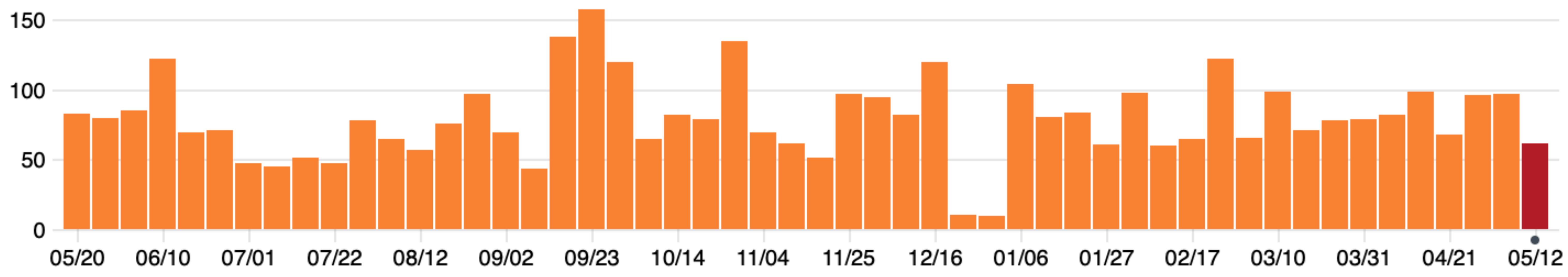
- Kim Albertsson, CERN
- Guilherme Amadio, CERN
- Sitong An, CERN
- Bertrand Bellenot, CERN
- Iliana Betsou, CERN
- Philippe Canal, Fermilab
- Javier Cervantes, CERN
- Olivier Couet, CERN
- Massimiliano Galli, CERN
- Enrico Guiraud, CERN
- Stephan Hageboeck, CERN
- Sergey Linev, GSI
- Lorenzo Moneta, CERN
- Alja Mrak Tadel, UCSD
- Axel Naumann, CERN
- Vincenzo Padulano, CERN
- Danilo Piparo, CERN
- Oksana Shadura, Uni Nebraska
- Matevz Tadel, UCSD
- Enric Tejedor, CERN
- Vassil Vassilev, Princeton Uni
- Stefan Wunsch, CERN

# Contributors and Activity

- ROOT is a very active project with many contributors and several active developments




**commits/week: last year**




**last month:**

Excluding merges, **36 authors** have pushed **469 commits** to master and **481 commits** to all branches. On master, **362 files** have changed and there have been **23,078 additions** and **6,742 deletions**.

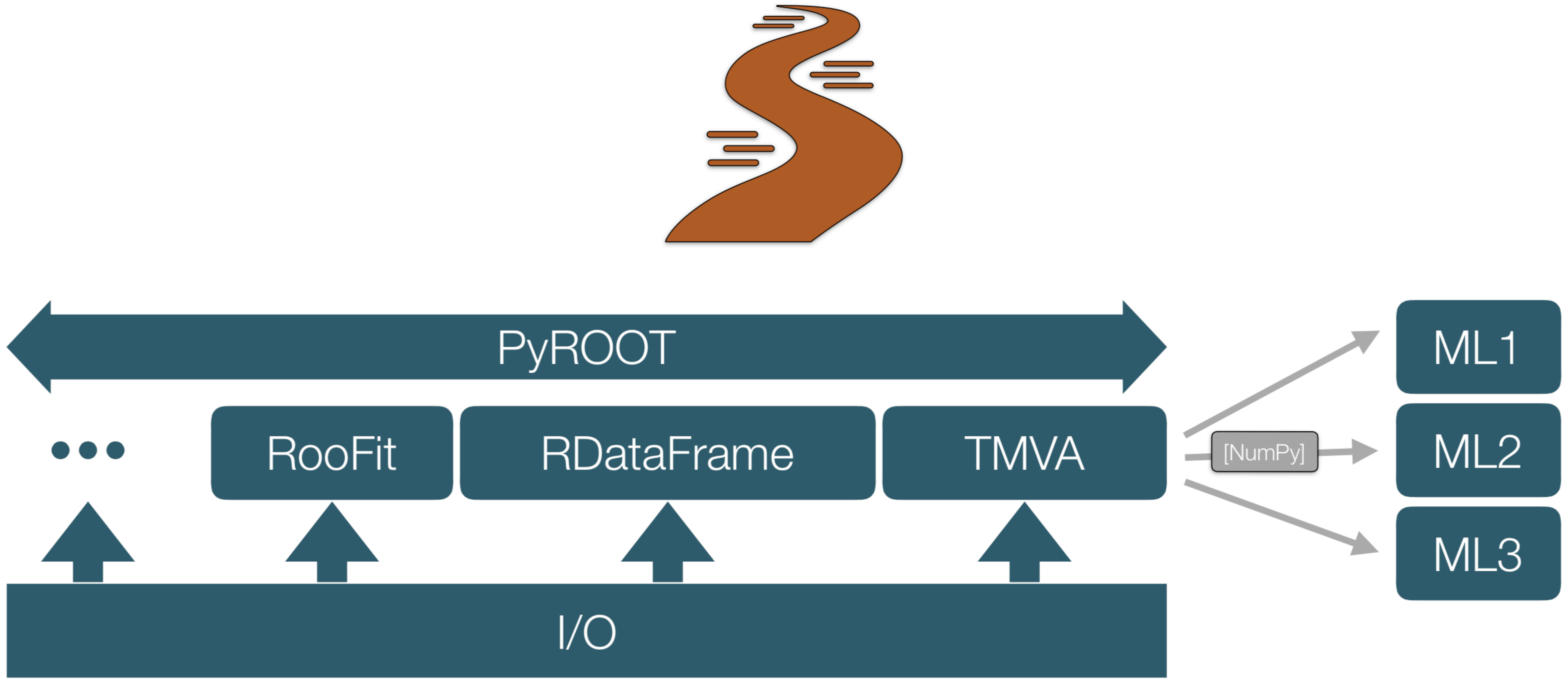
# Evolution

- ROOT is a very lively project
- maintenance of old code but several new important features are being developed
- Focus on key areas important for HEP
  - parts seeing by every physicists
- Focus on performances
  - Efficient code
  - Parallelization and SIMD vectorisation whenever possible
  - Transparent usage of GPU (e.g. in deep learning)
- Improved user interfaces
  - make usage of several new features of new C++
  - maintain a dual C++/Python API (uniqueness of ROOT)

# New Developments

## What are we focusing on ?

- Data frame for declarative analysts
- new Web based graphics and GUI
- new I/O data interfaces
- new PyROOT
- new C++ Histograms
- modernisation of TMVA
- faster RooFit



# ROOT I/O

- ROOT offers the possibility to write C++ objects into files
  - This is impossible with C++ alone
  - Used by the LHC detectors to write several petabytes per year
  - seamless C++ integration: unique feature of ROOT
- Achieved with serialization of the objects using the reflection capabilities, ultimately provided by the interpreter
  - Raw and column-wise streaming
- As simple as this for ROOT objects: one simple method -

```
file->WriteObject(pObj, "name");
```

# I/O Feature Comparison

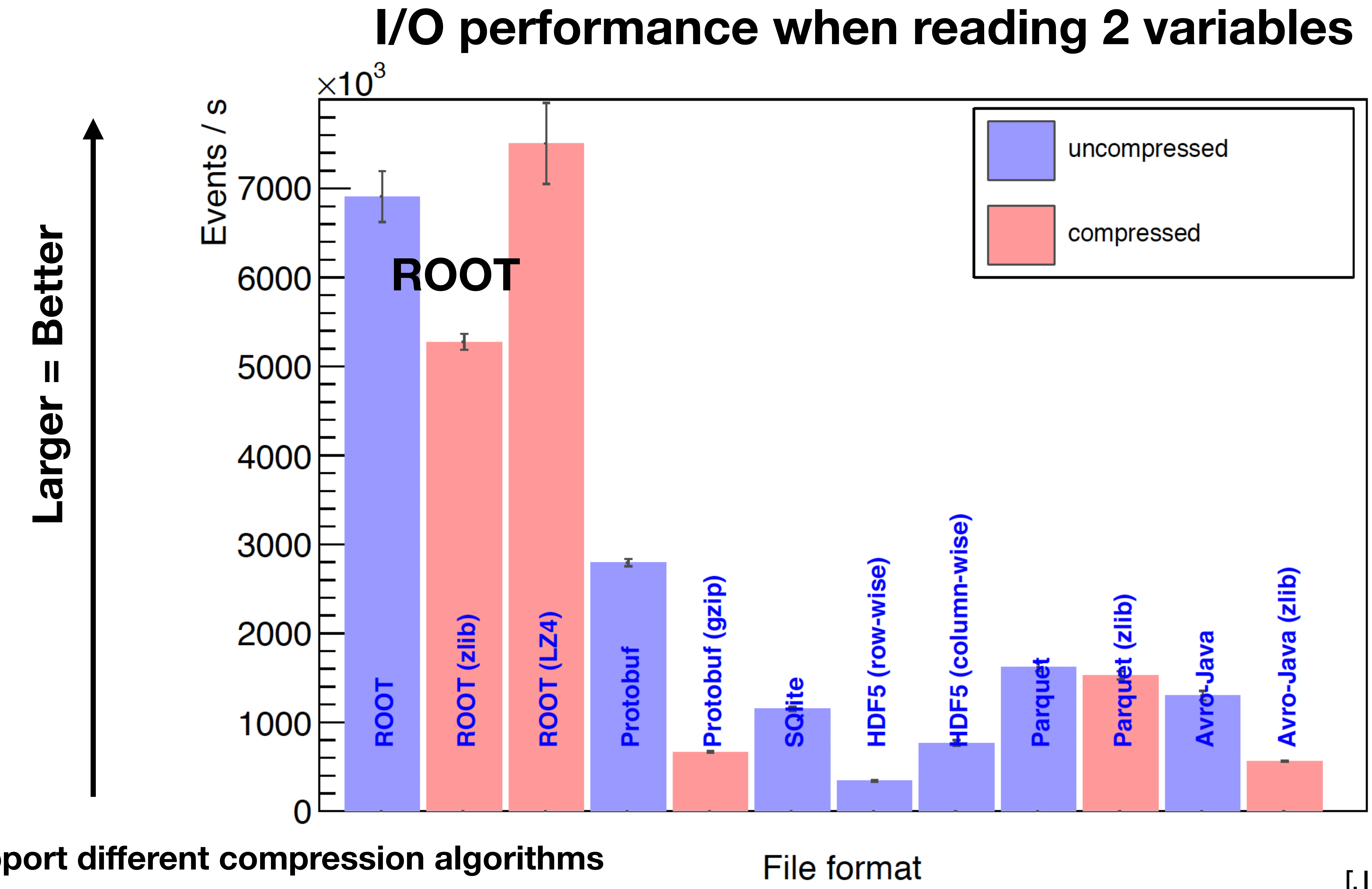
| | ROOT | PB | SQlite | HDF5 | Parquet | Avro |
|---|---|---|---|---|---|---|
| Well-defined encoding | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| C/C++ Library | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Self-describing | ✓ | ⚡ | ✓ | ✓ | ✓ | ✓ |
| Nested types | ✓ | ✓ | ? | ? | ✓ | ✓ |
| Columnar layout | ✓ | ⚡ | ⚡ | ? | ✓ | ⚡ |
| Compression | ✓ | ✓ | ⚡ | ? | ✓ | ✓ |
| Schema evolution | ✓ | ⚡ | ✓ | ⚡ | ? | ? |

[J. Blomer, ACAT 2017]

✓ = supported
⚡ = unsupported
? = difficult / unclear

- Unique capabilities of ROOT required for HEP data

# I/O Performance Comparison




**Support different compression algorithms**

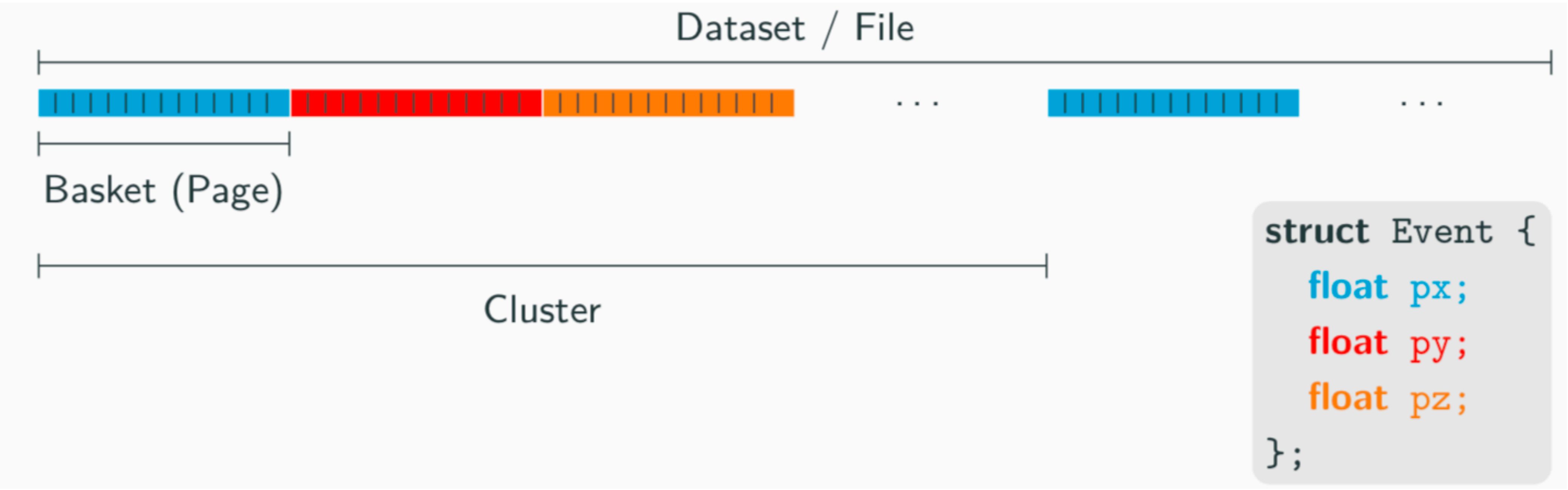
[J. Blomer, ACAT 2017]

# Evolution of ROOT I/O

- New Data Set/NTuple  classes, aiming for efficient HEP data analysis with speed:
  - optimised code for simple types and those known at compile time
  - improve mapping to vectorised and paralleled hardware
  - optimised integration with RDataFrame
- and robust interfaces:
  - type safe interfaces
  - Layer decomposition: logical layer, primitives layer, storage layer
  - Separation of data model with the actual data

# New ROOT DataSet

- Aim to continue to be faster than anything else also in simple cases




```
struct Particle {
  float fEnergy;   // plot only fEnergy...
  float fCharge;
};
struct Jet {
  std::vector<Particle> fParticles;
}
struct Event {
  std::vector<Jet> fJets;
};
```

**Potential gains of the refined layout**

- Natural access to bulk I/O

  First experiments indicate an improvement of the order of factor 5 in de-serialization

- Reading can return a reference to the memory buffer, avoiding value copies

  First experiments indicate an improvement of the order of factor 2 in de-serialization

- Branches of deeply nested collections benefit from columnar access

  Significant speedup but for a small subset of analyses – no additional cost introduced

Note: the reading speed is affected by both deserialization and decompression

# Decomposition

**Logical layer**
cling-assisted mapping of C++ types onto columns
e.g. `std::vector<float>` $\mapsto$ index column and a value column
or BLOB $\mapsto$ size column and `unsigned char` column

**Primitives layer**
"Columns" containing elements of fundamental types (float, int, ...)
grouped into (compressed) pages

**Storage layer**
e.g. TFile, raw file, object store

| Static | Live |
|---|---|
| RTreeModel | RTree |
| RColumnModel | RColumn |

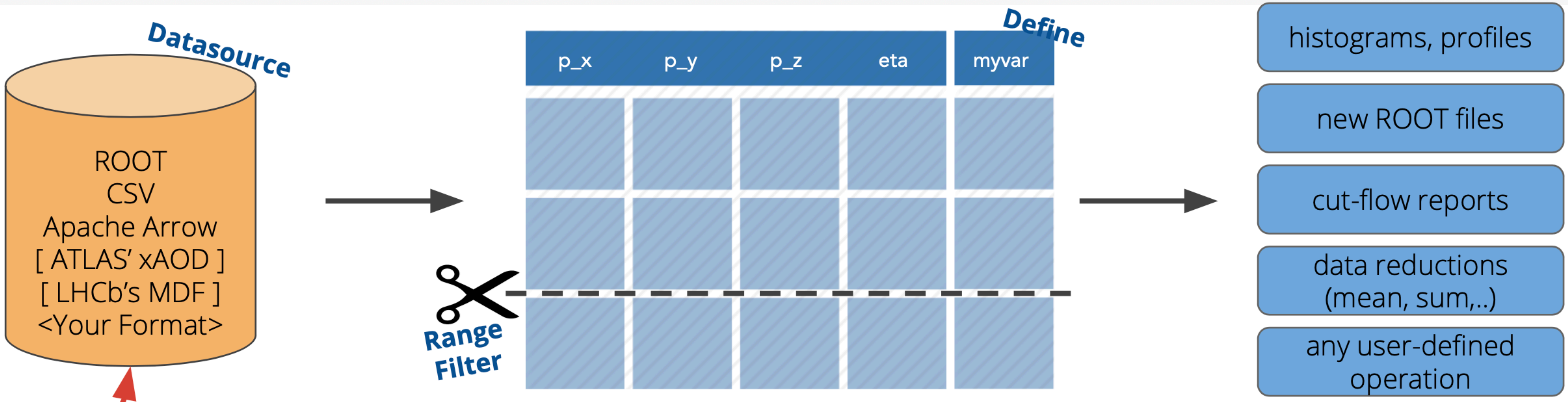
Separates the schema from the data; e.g., signal tree and background tree from same schema

- Allows for measuring performance of individual layers
- Allows us to experiment with different storage backends
- Primitives layer decoupled from C++ type system allows for lightweight 3rd party readers

# ROOT Declarative Analysis

- Highly efficient and simple data analysis
  - from I/O to histograms and statistical and machine learning tools
- Simple programming model
  - interfaces that are easy to use correctly and hard to use incorrectly
    - type safe (complains if code does not match data)
    - same interface in C++ and Python
  - Benefit from parallelism
    - can use all your CPU cores in the machine
    - can be deployed on a cluster with Python and Spark

# RDataFrame




Customisation point,
public interface!

Goals:

- ➔ Be the **fastest** way to manipulate HEP data
- ➔ Be the **go-to ROOT analysis interface** from laptop to cluster
- ➔ Consistent interfaces in **Python and C++**
- ➔ Top notch [documentation and examples]()

**In production since last year (ROOT version 6.14)**

# RDataFrame Example

- Simple example

| | |
|---|---|
| `ROOT::EnableImplicitMT();` | Run a parallel analysis |
| `ROOT::RDataFrame df(dataset);` | on this (ROOT, CSV, …) dataset |
| `auto df2 = df.Filter("x > 0")` | only accept events for which x > 0 |
| `.Define("r2", "x*x + y*y");` | define $r2 = x^2 + y^2$ |
| `auto rHist = df2.Histo1D("r2");` | plot r2 for events that pass the cut |
| `df2.Snapshot("newtree", "out.root");` | write the skimmed data and r2 to a new ROOT file |

**Lazy execution** guarantees that all operations are performed in **one event loop**

- Easy generalise to complex use-case

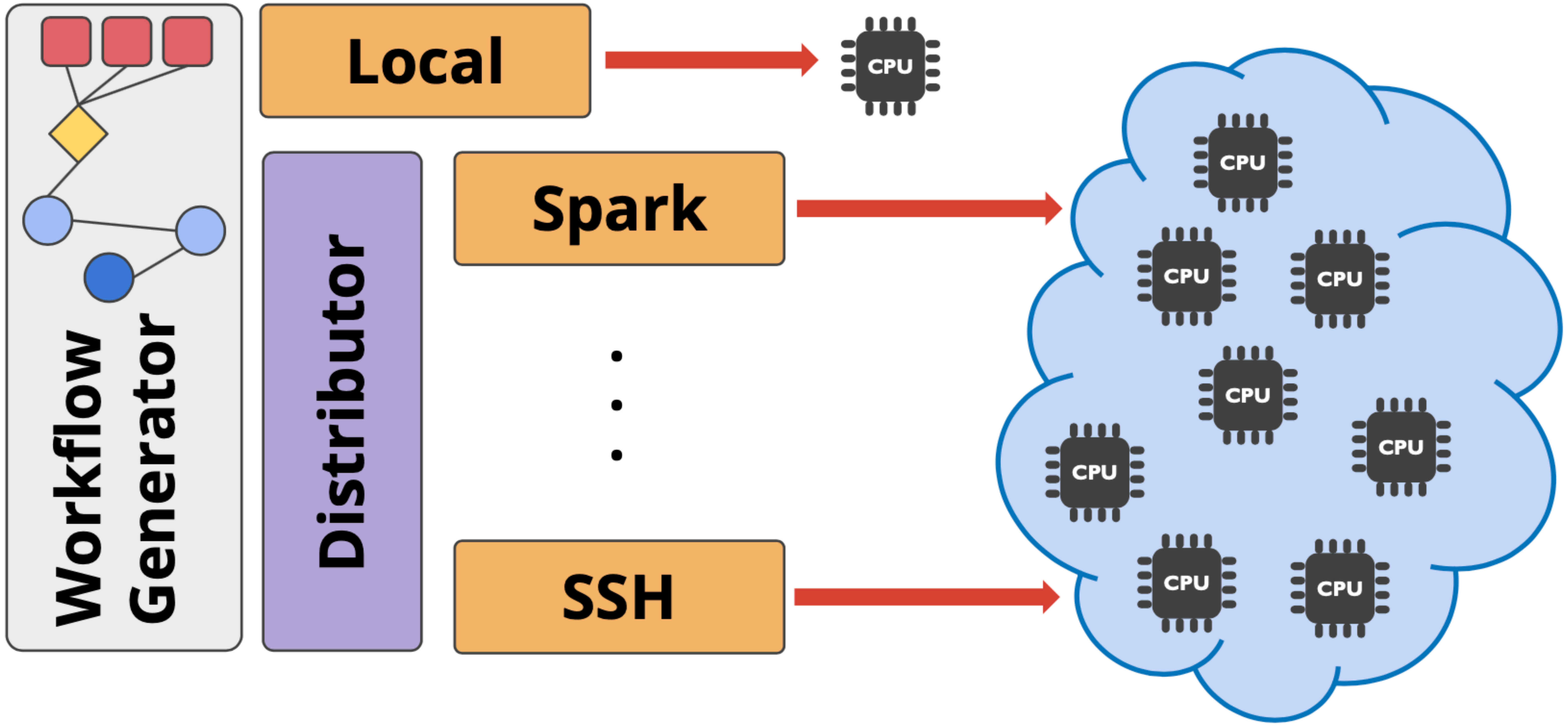
# Distributed Analysis

## Investigate and prototype a complement to PROOF

- Parallelism on many nodes
- Transparent distribution
- Support several different backends

**Re-use RDF interface: Minimal/No change in analysis code**

```
d = RDataFrame ("t", dataset)
f = d.Define(...)
      .Define(...)
      .Filter(...)

h1 = f.Histo1D(...)
h2 = f.Histo1D(...)
h3 = f.Histo1D(...)
```

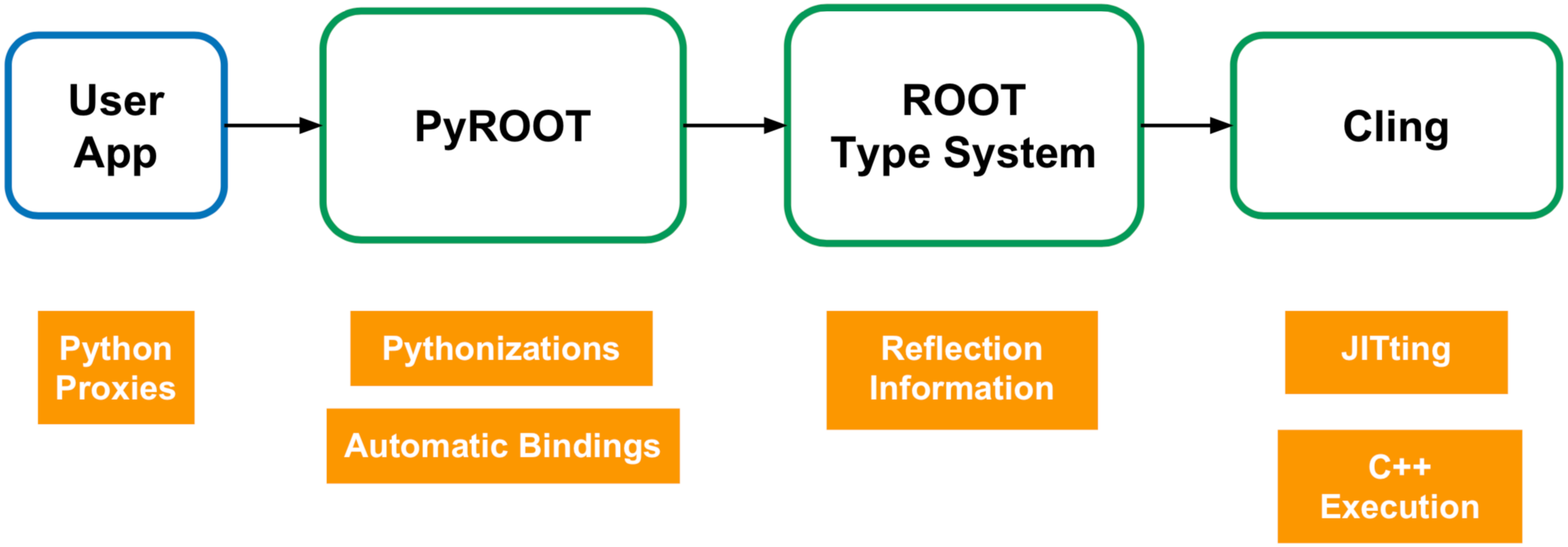



JavierCVilla/PyRDF

Working prototype available!

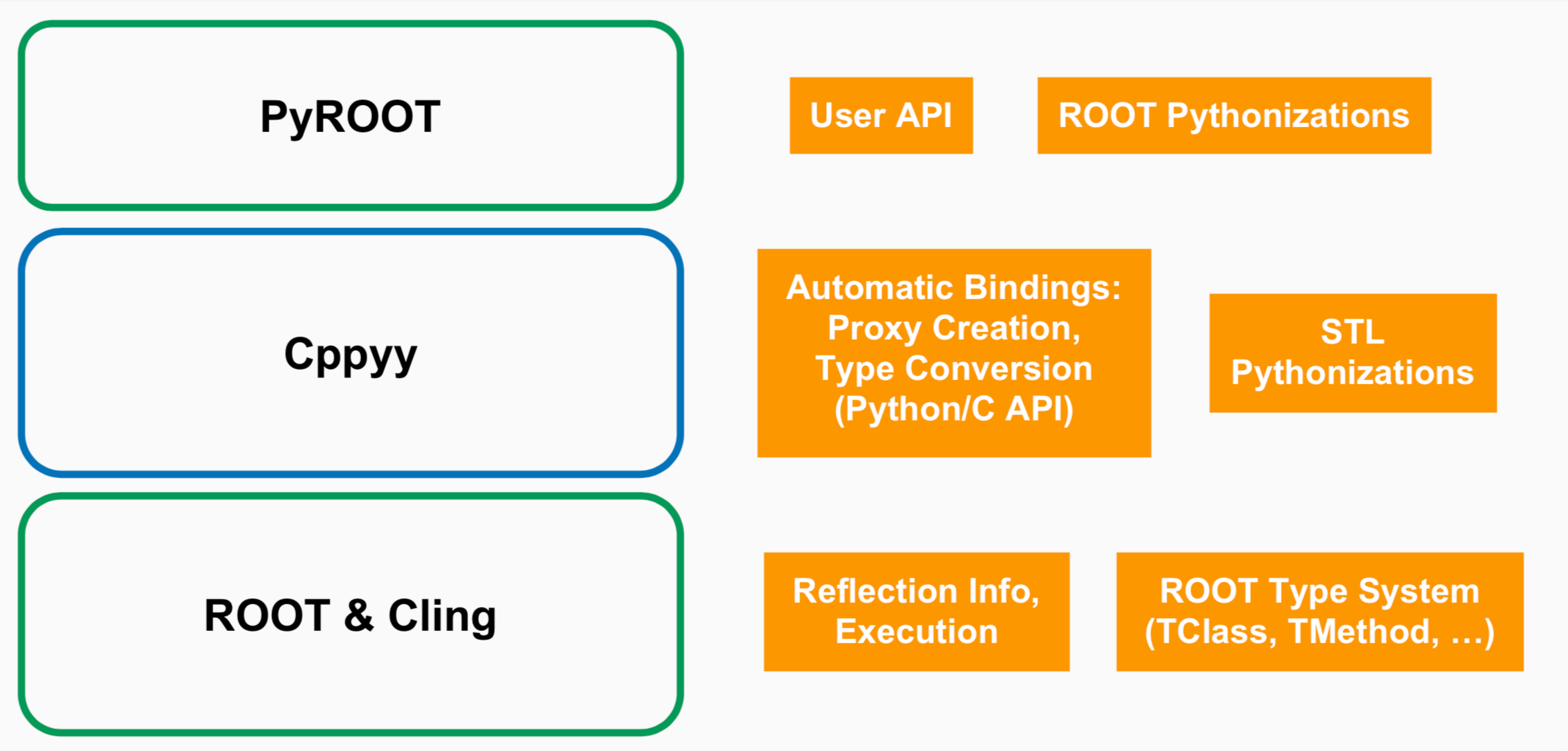
# PyROOT

- Goal make PyROOT more modern and pythonic
  - Previous PyROOT structure



# PyROOT

- New Pythonization based on Cppyy
  - automatic C++-Python bindings using directly Cling/LLVM



# New Features

- Users can define Pythonizations for their own classes
  - lazily executed

```python
@pythonization('MyCppClass')
def my_pythonizor_function(klass):
    # Inject new behaviour in the class
    klass.some_attr = ...
```

Python proxy of the class

- Better support for new C++ features:
  - variadic templates in function arguments
  - defining and using C++ lambda functions from Python

# Interoperability with NumPy

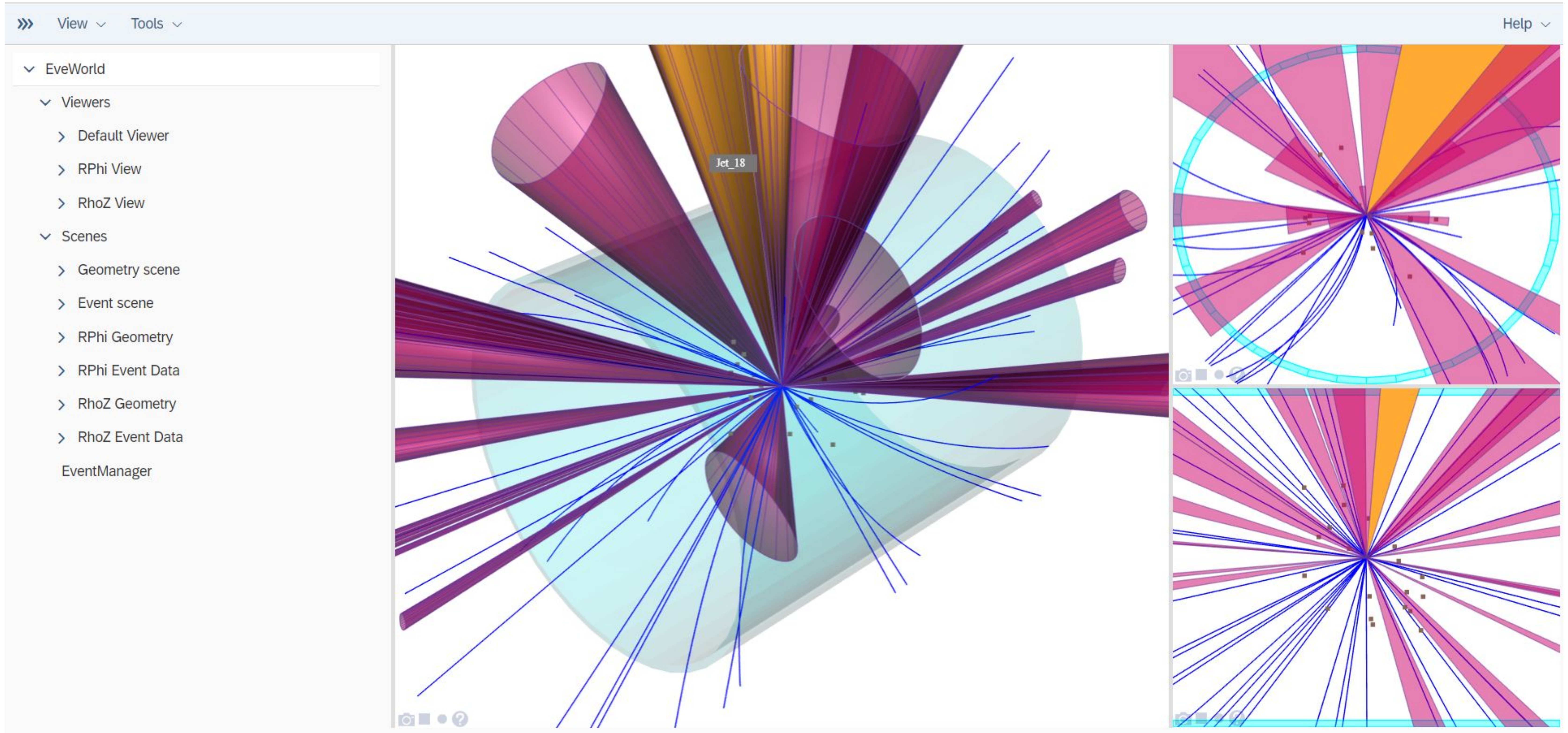
- Better interoperability with Data Science Python tools
  - easier conversion to NumPy and Pandas
  - easier to use powerful machine learning Python tools
- Example: Reading a TTree's directly in NumPy arrays

```
myTree # Contains branches x and y of type float

# Convert to numpy array and calculate mean values of all
# branches
myArray = myTree.AsMatrix()
m = np.mean(myArray, axis = 0)

# Read only specific branches
onlyX = myTree.AsMatrix(columns = ['x'])
```

- With RDataFrame.AsNumpy([‘v1’,’v2’,’v3’]) also from TTrees to NumPy arrays

# New ROOT7 Graphics

- ROOT7 uses Web based technologies
  - able to run in the Web browser

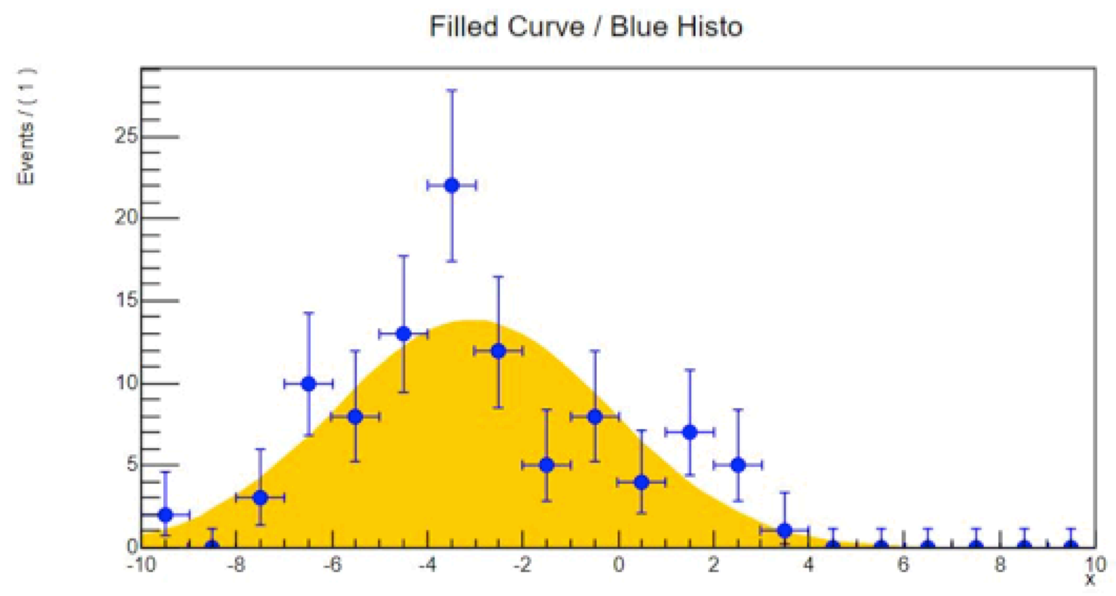



**WebEve**

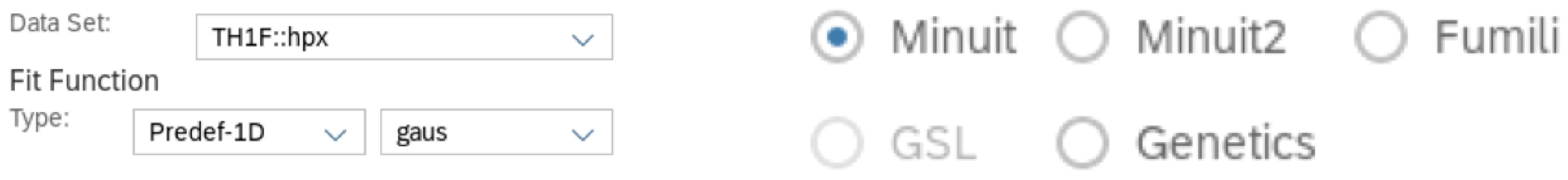
can run as
- a standalone application
- an existing browser
- embedded in other Web based GUI's (e.g. Jupiter notebooks)

# ROOT7 Graphics Technologies

- Runs directly in a browser-backed window
  - client side : Javascript
  - server side: ROOT C++




- Components:
  - Javascript ROOT for displaying ROOT objects in browser (e.g. histograms)
    - using three.js and D3.js
  - THttpServer for the communication
  - TBufferJSON for converting ROOT objects to JSON
  - SAP OpenUI5 for GUI in browser (new fit panel, RBrowser,…)



# ROOT7 Histograms

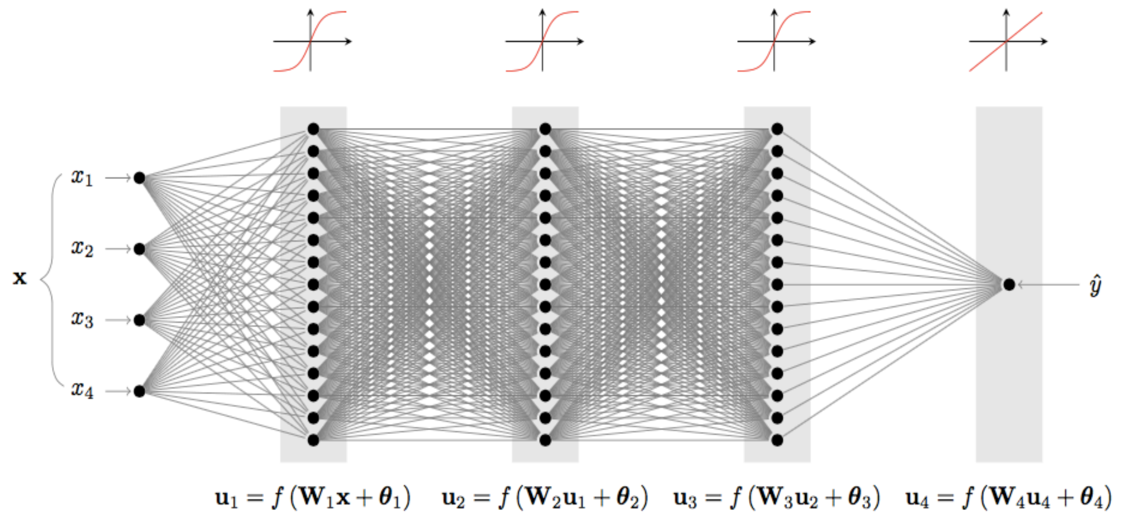
- RHist classes, available already in ROOT::Experimental
- Faster by moving conditional branches to compile time
- New design with separation of concerns:
  - storage, binning, graphics, internal operations
  - make usage of new C++ features
- Extensible:
  - e.g. customisation of uncertainty algorithms

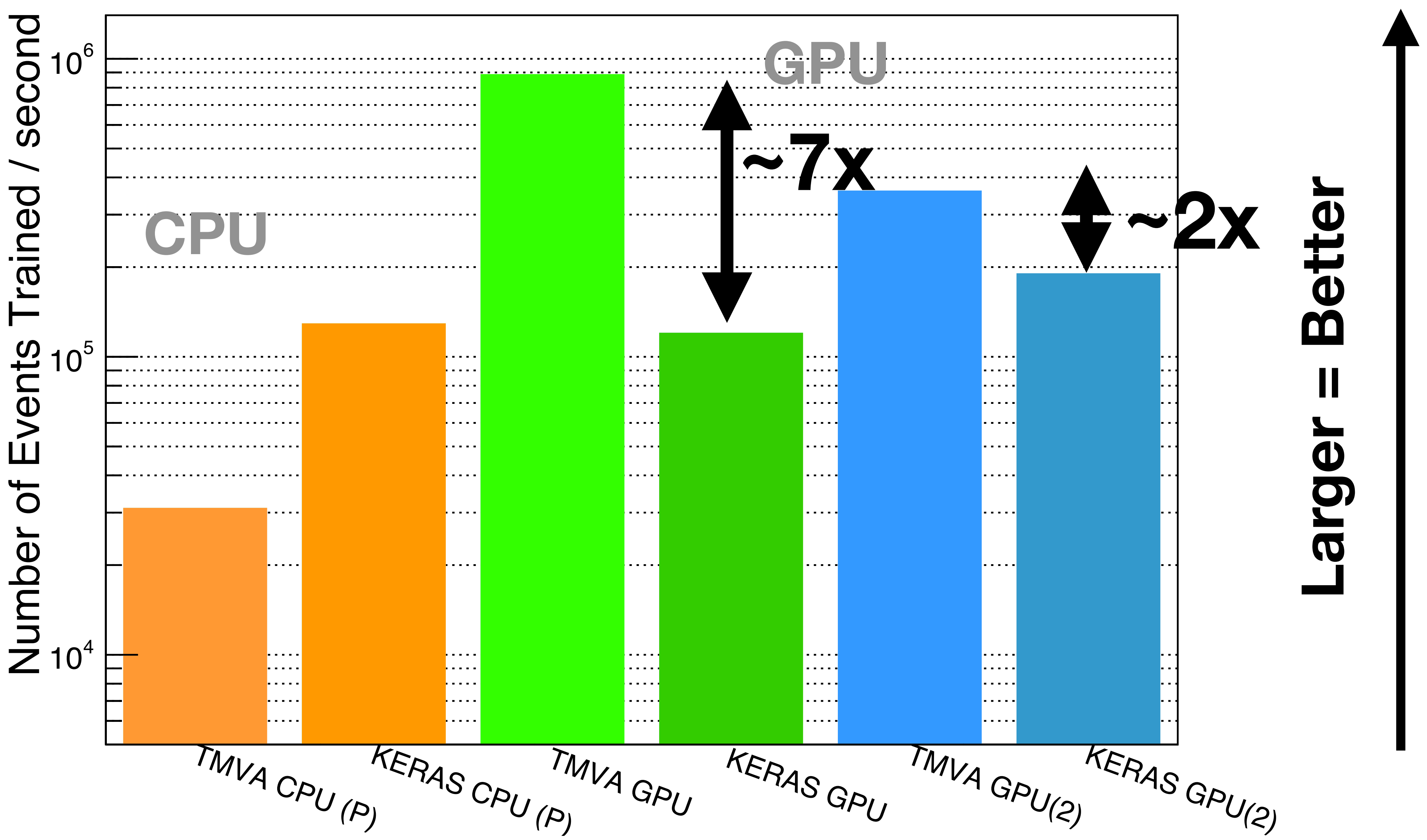
```
// Create a 2D histogram.
// X axis equidistant bins, y axis irregular binning.
Experimental::RH2D hist({100, 0., 1.},
                        {{0., 1., 2., 3., 10.}});

// Fill weight 2.1 at the coordinate 0.01, 1.02.
hist.Fill({0.01, 1.02}, 2.1);
```

- Looking for early adoption to get feedback and guide development
- Integration with RDataframe and new graphics started

# Machine Learning in ROOT

- Going through a modernisation of TMVA
- New deep learning tools added
  - Support also Convolutional and Recurrent layers
  - Efficient running using parallelisation on both CPU and especially GPU
    - faster than Tensorflow/ Keras on GPU for simple models
- Focus on optimal model inference in TMVA

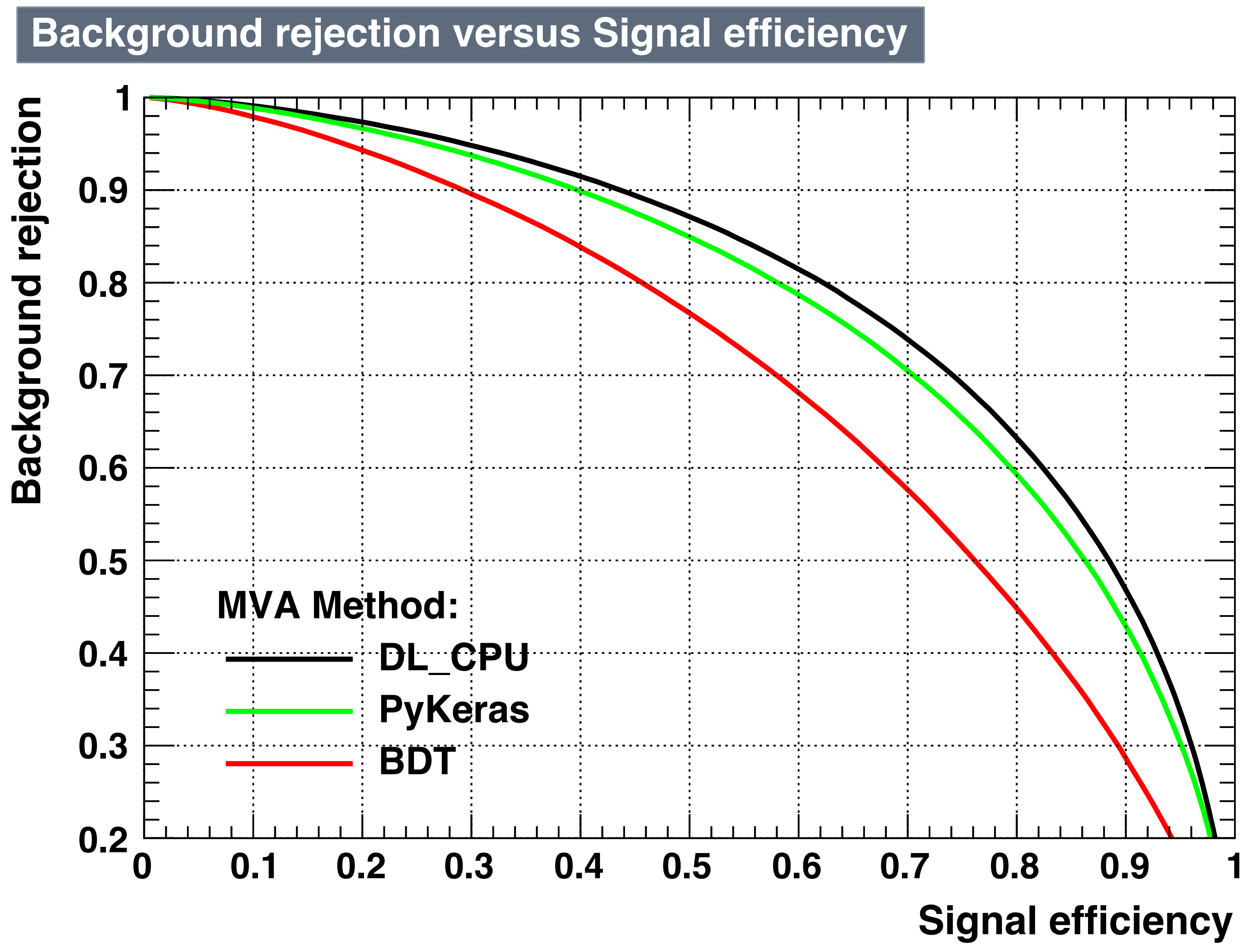




# TMVA Interfaces

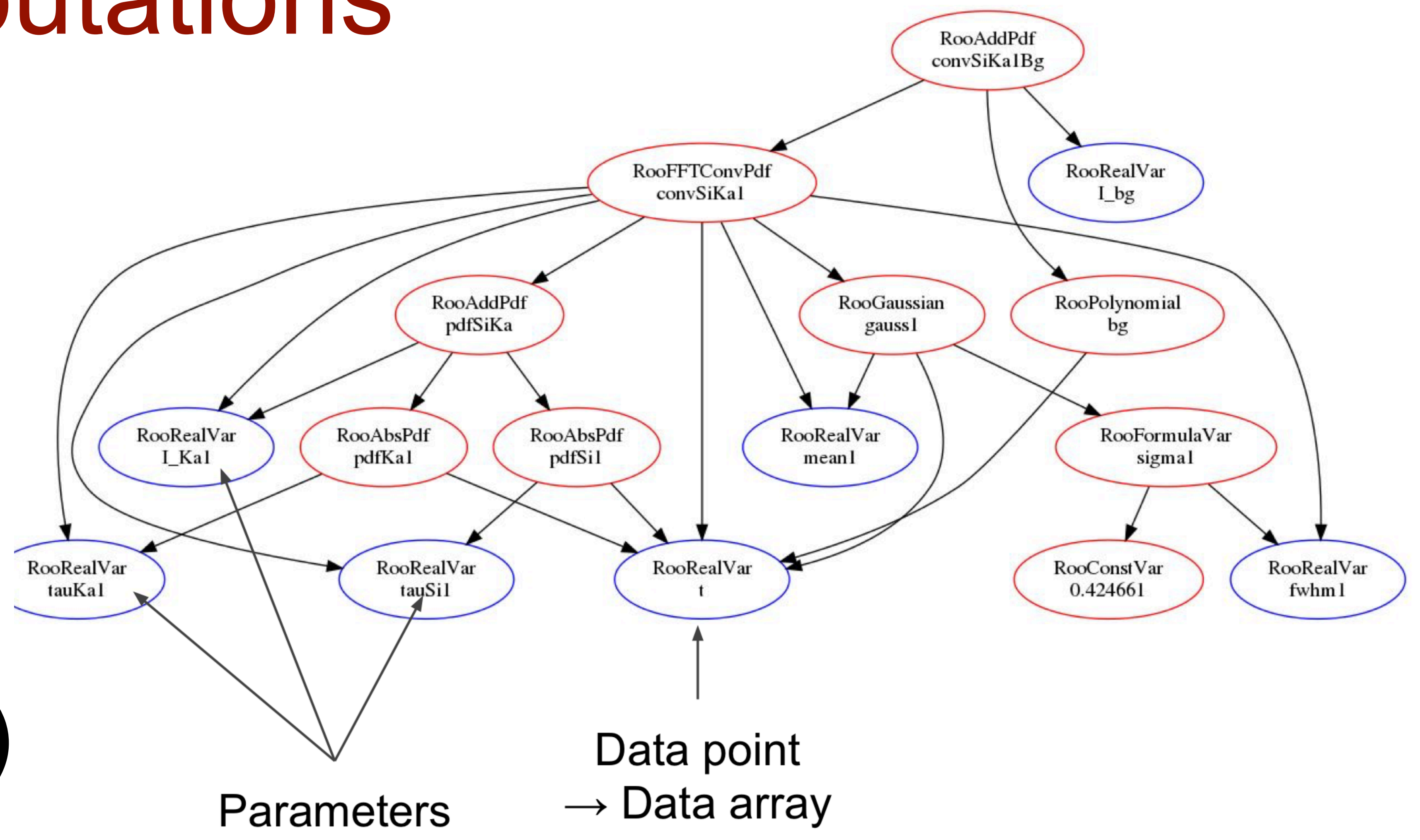
External tools are available as additional methods in TMVA and they can be trained and evaluated as any other internal ones.

- **RMVA**: **Interface to Machine Learning methods in R**
  - c50, xgboost, RSNNS, e1071
- **PYMVA**: **Interface to Python ML packages**
  - **scikit-learn**
    - RandomForest, Gradient Tree Boost, Ada Boost
  - **Keras** (Theano & Tensorflow)
    - support model definition in Python
    - can perform training and evaluation inside ROOT/TMVA in C++ with direct connection to ROOT data



# Faster Fitting

- Implemented already parallelisation in ROOT Fitting
- RooFit : unique tools for data modelling and fitting in HEP
  - focus on modernisation
    - better interfaces (more Pythonic), usage of standard collections, etc..
  - improving performances
    - refactor code for vectorisation in function computations
    - reducing virtual function calls
    - parallelisation whenever possible
      - likelihoods computation (loop on events)
        - first trying using multi-processes
          (need optimal scheduling for complex models)
      - toy MC generation



# Auto-Differentiation

- ROOT will have also the capability for Auto-differentiation
  - working in C++ and using power of Cling (Clad) [vgvassilev/clad]
- Essential component in many machine learning frameworks
- Allows to compute gradients directly by computer analysing the code path
  - Back-propagation model allows in a single pass to compute gradient for all input variables
  - Much faster than applying numerical differentiation where each parameter needs to be varied followed by a forward model pass
- Extremely useful for machine learning tools and fitting

# Other new developments

- C++ modules for ROOT dictionary
  - improved compilation time and easier deployment
- Building components on-demand
- CUDA back-end for Cling
- More vectorisation:
  - development of new vectorised random number generators for parallel application (e.g MixMax generator)
- New interfaces
  - Integration in Jupyter notebooks
  - Jupyter C++ kernel based on ROOT/Cling available world-wide

# Python vs C++ Interfaces

- ROOT allows to have exactly same code in both C++ and Python
  - Example DataFrame:

**C++ with cling's just-in-time compilation**

```
d.Filter("theta > 0").Snapshot("mytree","f.root","pt_x");
```

---

**PyROOT, automatically generated Python bindings**

```
d.Filter("theta > 0").Snapshot("mytree","f.root","pt_x")
```

- In ROOT 7 we have value based C++ (same semantics)

`rcanvas.Draw(hist,options);` **C++**

`rcanvas.Draw(hist,options)` **PyROOT**

# Non ROOT Analysis Software

- Many alternative open source software exist now for data analysis
  - Some of the Python data science tools are very appealing
- But big data processing different than physics analysis
  - coding analysis; usability; CPU efficiency; data delivery; setup- cost / scalability; event-based; must not skip data points; role of uncertainties, etc…
- Major developments of these tools driven by big players (Google, Facebook, Amazon,…)
  - adapting to them my require large developments
- Lifecycle of these tools much shorter than timescale of HEP experiments
  - e.g Theano once major deep learning tool now already deprecated

# Outlook

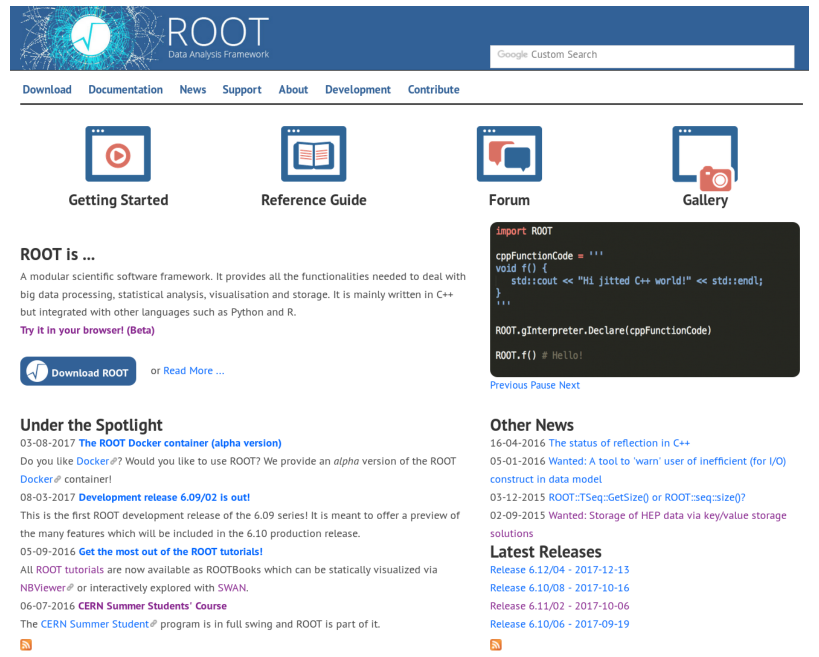
- Essential to maintain expertise of data analysis in HEP community
- Steer ROOT to maintain it competitive to alternative solutions
  - make use of external tools whenever useful
    - at the lower level (Clang) or for GUI (OpenUI)
  - provide interfaces to powerful external components
    - e.g Python tools for Machine Learning (e.g. Numpy, Tensorflow)
- We need to prove itself against these existing alternative
  - accepting challenge to deliver a simpler, friendly and more robust ROOT

# Conclusions

- ROOT modernisation is possible thanks to the contributions from CERN, Fermilab, Google (Summer of code students) and all our users
  - special  thanks to their essential contributions in reporting bugs, providing patches, helping out on the forum, criticism and feedback

- Long Term support and evolution of ROOT is assured from CERN
  - resources are guaranteed (for long lifetime of experiments)
  - experience supporting more than 30k users

# References




- [root.cern](root.cern)
- [https://cern.ch/forum](https://cern.ch/forum)
- https://github.com/root-project


- For more information of current developments see presentations at latest <u>ROOT Users Workshop</u>

- See also ROOT work planning <u>presentation</u> for 2019

Next ROOT Users workshop will be in 2020 at Fermilab
- we will organise before a train the trainers event

# Backup Slides

# New I/O interfaces

### Writing

```
auto eventModel = std::make_unique<RTreeModel>();
auto particleModel = std::make_shared<RTreeModel>();
auto pt = particleModel->Branch<float>("pt");
auto particles = eventModel->BranchCollection(
  "particles", particleModel);

// With cling:
//auto event = eventModel->Branch<Event>();

RColumnOptions opt;
RTree tree(eventModel, RColumnSink::MakeSink(opt));

for (/* events */) {
  for (/* paricles */) {
    *pt = ...;
    partilces->Fill()
  }
  tree.Fill();
}
```

### Reading

```
RColumnOptions opt;
RTree tree(RColumnSource::MakeSource(opt));
auto view_particles =
  tree.GetViewCollection("particle");
auto view_pt = view_particles.GetView<float>("pt");
for (auto e : tree.GetEntryRange()) {
  for (auto p : view_particles.GetRange(e)) {
    cout << view_pt(p) << endl;
  }
}
```

### RDataFrame

```
RColumnOptions opt;
opt.pathName = "";  // ...
auto rdf = ROOT::MakeForestDataFrame(opt);
```

# RDataFrame: Collections

### Jitted C++ or PyROOT

All is a helper to check if all values in an array are true

muon_eta is a collection, not a scalar

```
auto inMemDF = d.Filter("All(muon_eta < 2.5)")
                  .Cache({"muon_eta"});
```

---

### C++

```
auto cutEtas = [](RVec<float> &etas) { return All(etas < 2.5); };
auto inMemDF = d.Filter(cutEtas, {"muon_eta"})
                  .Cache<RVec<float>>({"muon_eta"});
```

[RVec reference guide](#) **(top notch doc, too!)**

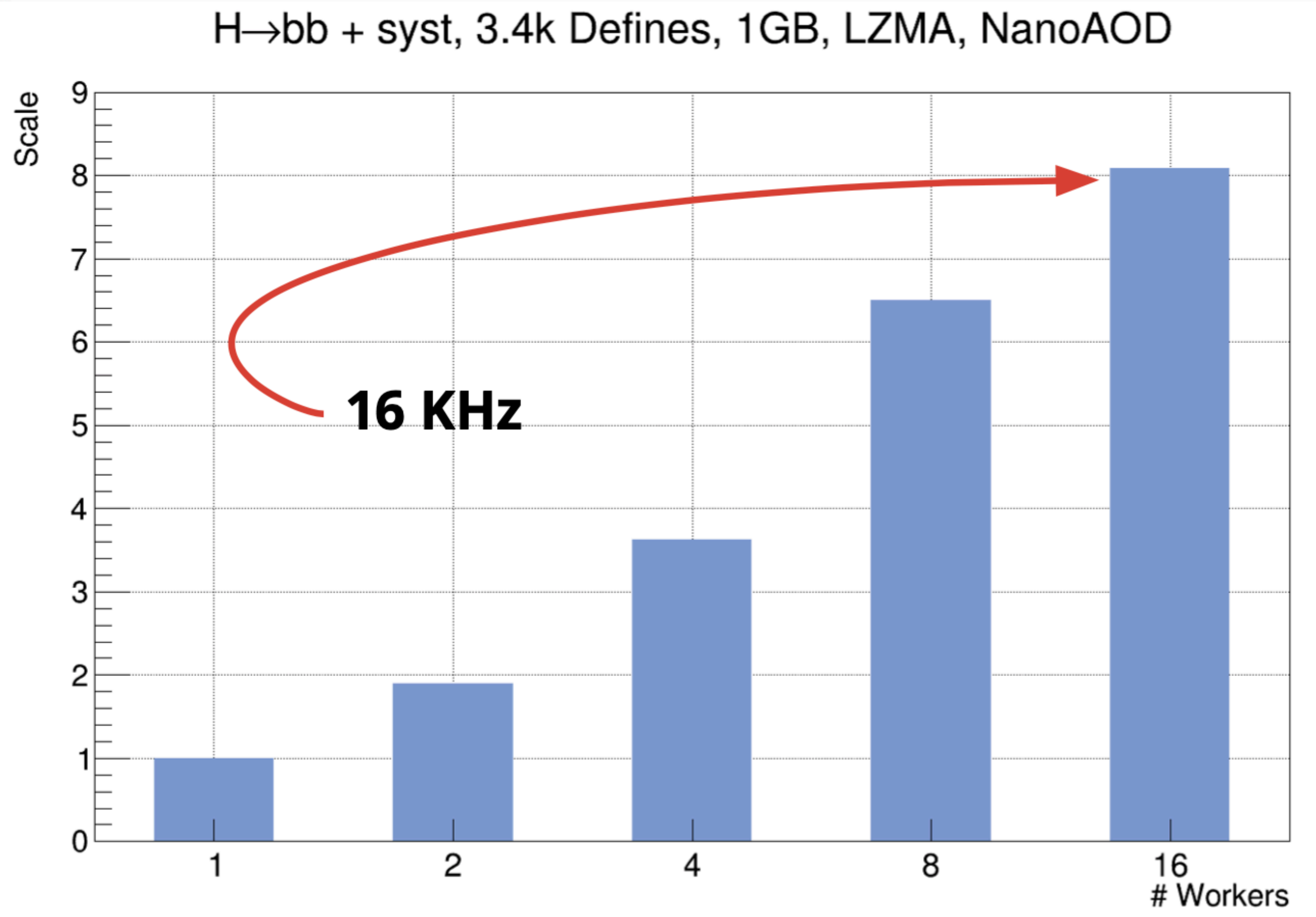
# RDataFrame Scaling

## H → μμ

Realistic analysis, 100 systematics

- **3400 nodes in the computation graph**, heavy usage of RVec
- 1GB input file, NanoAOD format, LZMA compressed
- Reading+Decompressing: ~20% of the sequential runtime

Intel Core i7 7820X (8*2 cores, 3.60GHz)




**Realistic Analysis, Large Computation Graph: Good Performance & Efficient Scaling**

# Variadic Templates in PyROOT

- Example: RooFit. Want to move to variadic template arguments
  - Clean up documentation and function signatures
  - Make it work with any number of arguments
  - Arguments need not be same type

```
# Old signature in C++:
# RooFitResult * RooAbsPdf::fitTo(RooAbsData & data, const RooCmdArg & arg1 = RooCmdArg::none(),
# const RooCmdArg & arg2 = RooCmdArg::none(), const RooCmdArg & arg3 = RooCmdArg::none(), ...)
result = myPdf.fitTo(data, ROOT.RooFit.Save(), ROOT.RooFit.Minos(), … )
```

```
# New signature:
# template <typename ...CmdArg>
# RooFitResult * RooAbsPdf::fitTo(RooAbsData & data, CmdArg… args)
result = myPdf.fitTo(data, ROOT.RooFit.Save(), ROOT.RooFit.Minos(), … )
```

Works in new PyROOT!

The ROOT Team, ACAT 2019

# C++ Lambda in PyROOT

- Define and use a C++ Lambda in Python

```
>>> import ROOT
>>> ROOT.gInterpreter.ProcessLine(
"auto mylambda = [](int i) { std::cout << i << std::endl; };")
140518947094560L
>>> ROOT.mylambda
<cppyy.gbl.function<void(int)>* object at 0x35f9570>
>>> ROOT.mylambda(2)
2
```

# RDF to Numpy and Pandas

```
# Run input pipeline with C++ performance that can process TBs of data, reads from remote, ...
import ROOT
df = ROOT.RDataFrame("tree", "file.root")
          .Filter("All(pt>30)", "Trigger requirement")
          .Filter("All(tight_iso)", "Quality cut")
          .Define("r", "sqrt(eta*eta + phi*phi)")

# Extract selection w/ defined variables as numpy arrays
col_dict = df.AsNumpy(["r", "eta", "phi"])

# Wrap data with pandas
import pandas
p = pandas.DataFrame(col_dict)
print(p)

   r     eta   phi
0  0.26  0.1  -0.5
1  1.0  -1.0   0.0
2  4.45  2.1   0.2
...
```

**All the power of RDF + possibility to convert to NumPy**

# Deep Learning in TMVA

- Recent additions to standard dense layers architectures:
  - Convolutional and recurrent layers
  - New optimisers for faster convergence
- Development ongoing!
  - Long Short Term Memory (LSTM) cells for recurrent layers
  - Generative adversarial networks (GAN) and Variational auto-encoder (VAE) for event generation