IKTP Dresden Seminar, 2017-12-13

Data Analysis with ROOT, Now and in the Future

Axel Naumann axel@cern.ch

Content

- Short intro to ROOT
- Vision for ROOT in 2020
- Main development areas
- News from v6.12

What is ROOT?

- *The* data analysis tool for High Energy Physics
- Efficient storing and reading of data, analysis, statistical tools, graphics
- About 20,000 users around the globe
- Started 20 years ago, now +/- 3 million lines of code (mostly C++), LGPL'ed
  - C++ interpreter with unique, dynamic Python binding PyROOT
- Used in HEP, astronomy, industry,…

ROOT In Numbers

- 15 team members
- ROOT forum: 11’000 users, >100 new users / month, 1’300 posts / month
- Fixing about 600 bugs a year…
- https://github.com/root-project/
- 👉 Alive and rocking

12 Month Summary

Aug 14 2016 — Aug 14 2017

4541 Commits

Up + 1029 (29%) from previous 12 months

110 Contributors

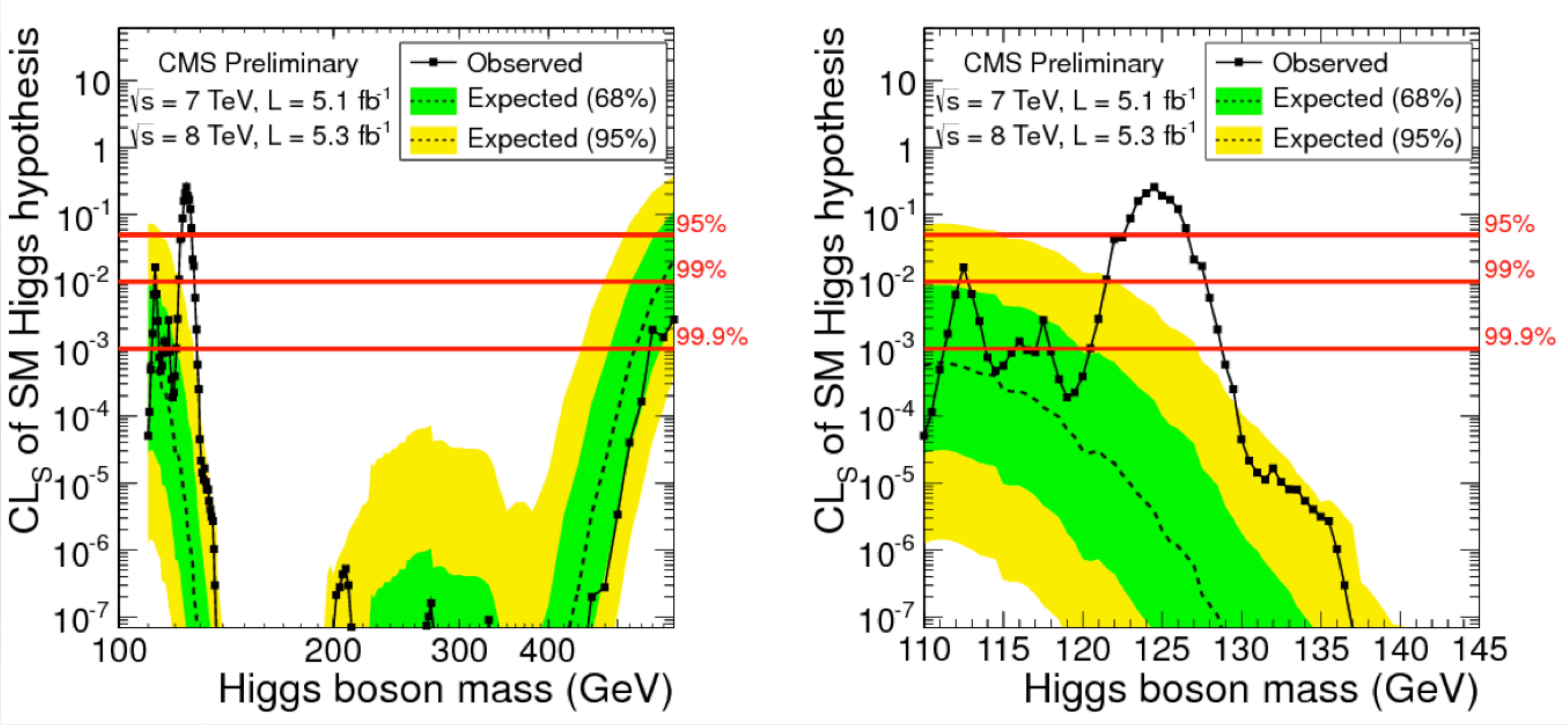
Up + 38 (52%) from previous 12 months

Source: https://www.openhub.net/p/ROOT

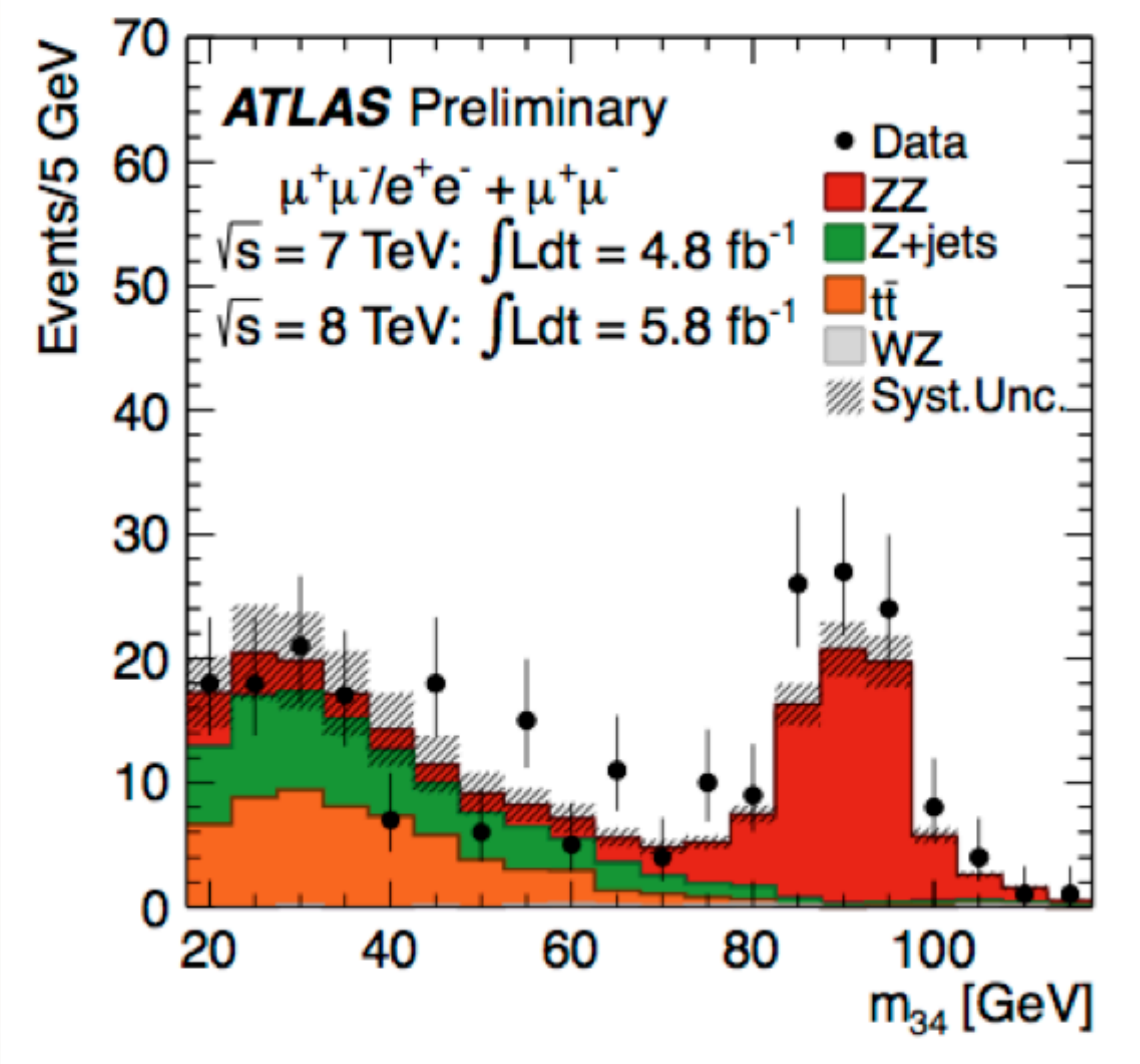
ROOT Features

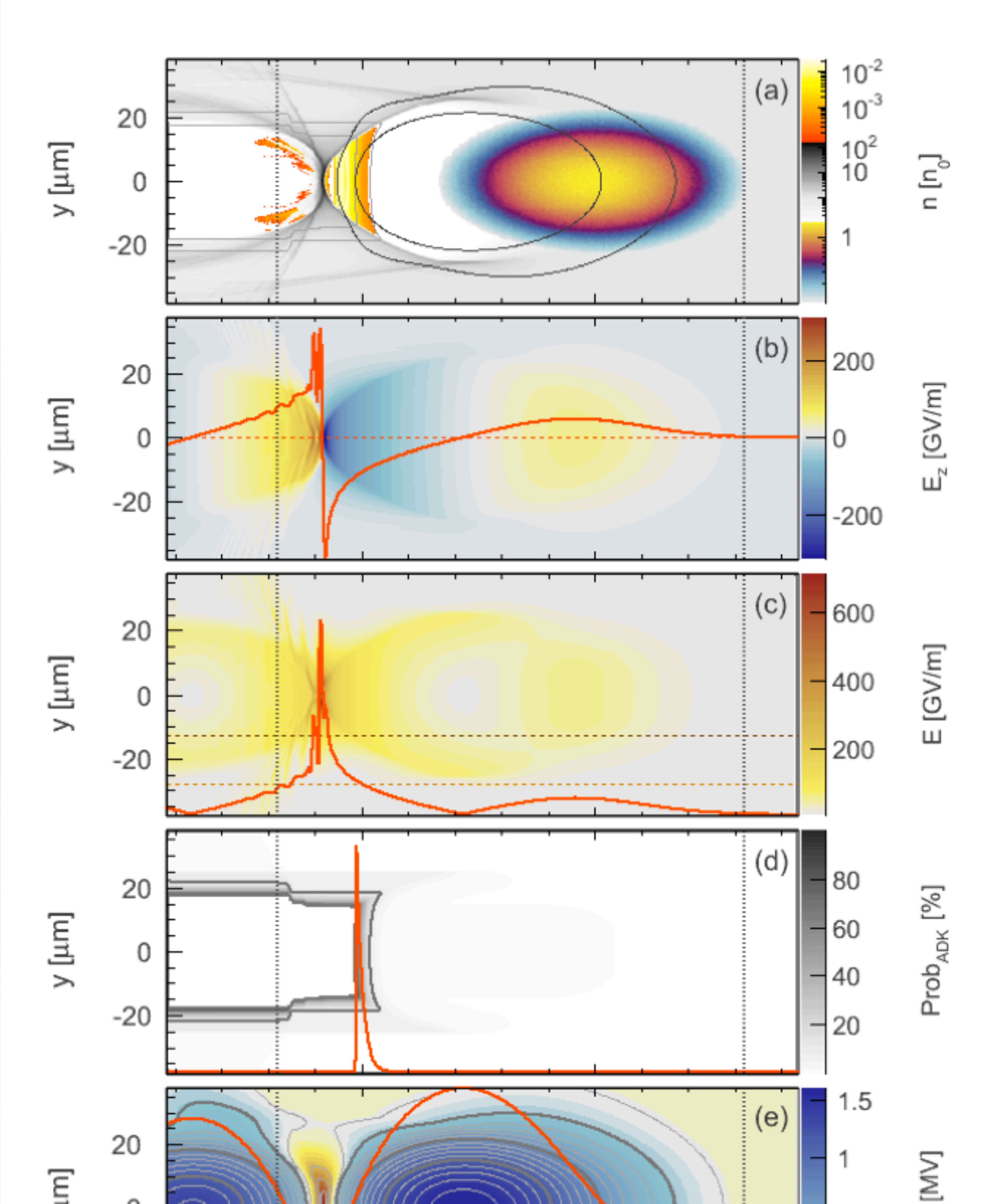
- About 1 Exabyte (i.e. 1′000′000 TB) of data in ROOT files
  - takes your C++ classes and dumps them to disk
  - again proven to be number 1 in performance for HEP data [link]
- “Proper” scientific modeling, statistics, minimization / fitting
- High-quality, highly customizable graphics
- Analysis interfaces for physicists, not computer scientists

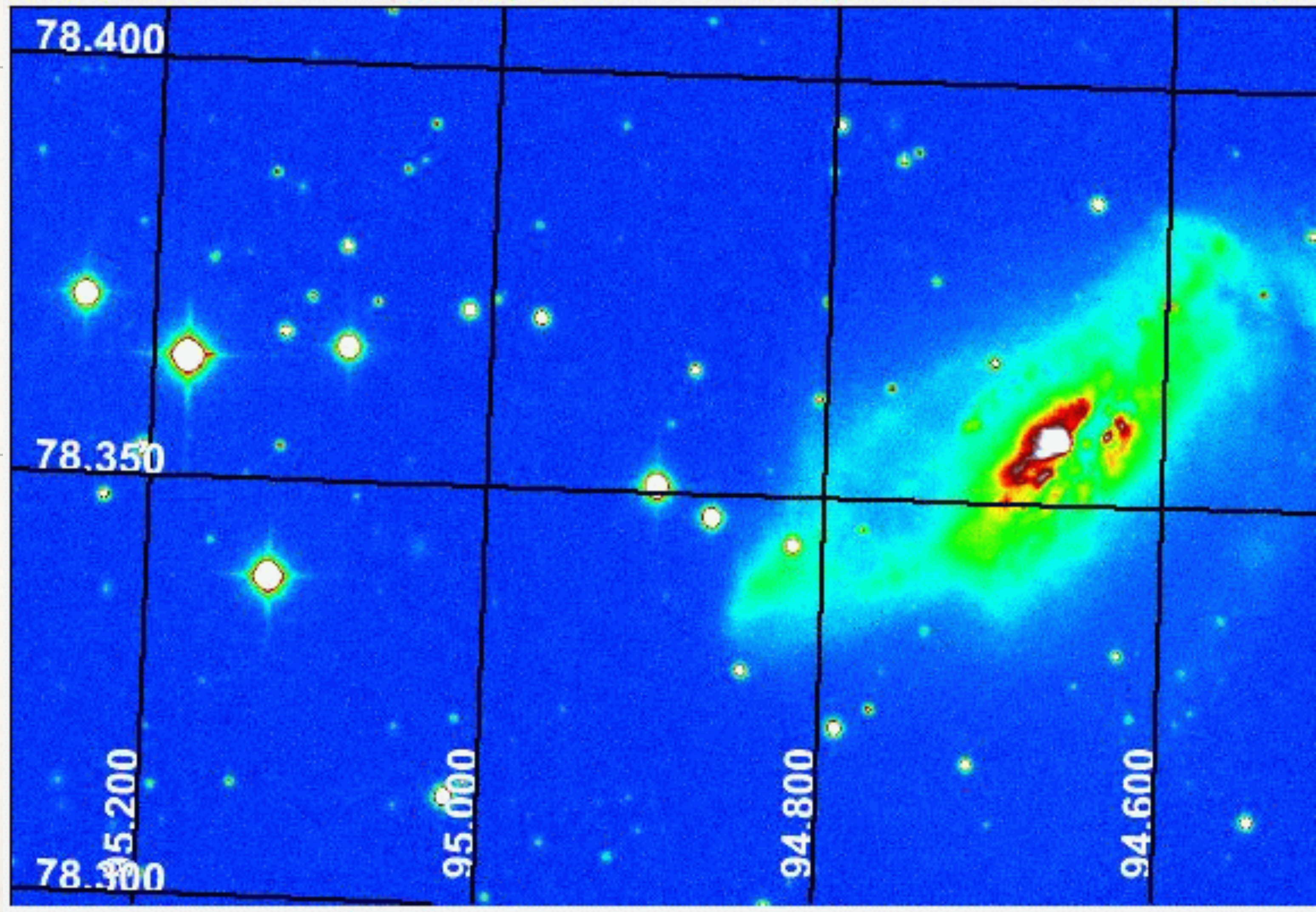
Graphics Examples

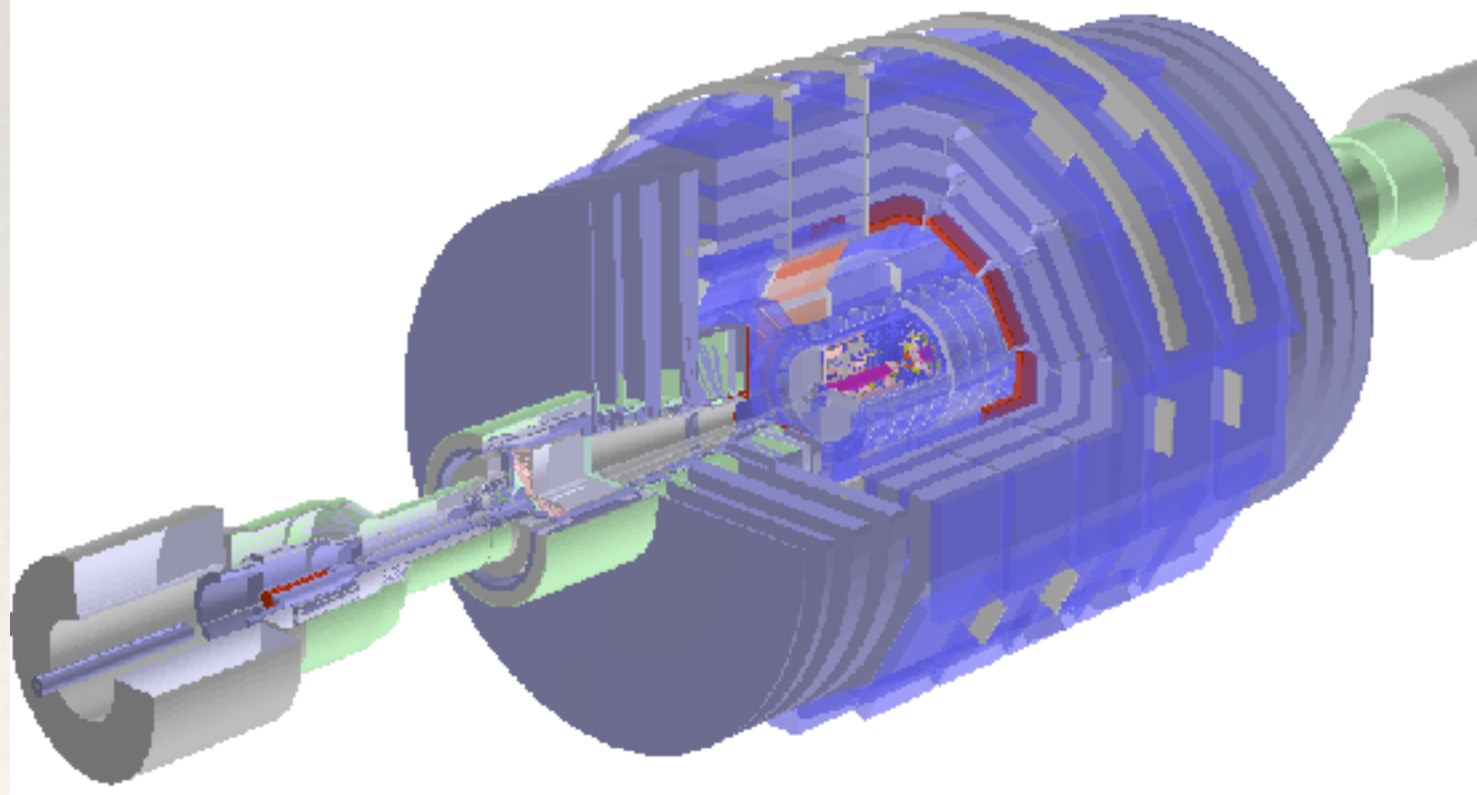


Graphics Examples









ROOT in 2020

Context

- ROOT is at the center of data analysis since +/- 15 years
  - today, the world offers lots of open source tools for big data
- ROOT provides expertise to the community by the community
  - many solutions specific and optimized for HEP
  - alternatives are often not direct solutions
- And yet: 20 years of success is not a guarantee for the future!

The Future

- ROOT's main goals:
  - simplicity
  - robustness
  - speed

Simplicity

- Focus on physics, reduce time spent on coding (and debugging!)
  - clear, consistent C++ interfaces
  - excellent Python support (more pythonic ROOT a la rootpy)
  - do things perfectly by default, but allow for customization
- Modern C++ helps write simple, clear interfaces
- Separation of concerns (e.g. histograms from graphics from I/O)

Robustness

- ROOT’s memory model based on PAW:
  - directories own named objects etc
  - causes crashes due to raw pointers and implicit ownership
- Arrays are pointers, configuration through strings
- Instead: let the compiler check where possible instead of runtime errors

Speed

- C++ from 1995 was all about object oriented code
  - has proven to incur a performance cost
- Instead: modern design, bulk operations where possible, less virtual functions / more vectorization and cache locality
- Thread-safe, context-free objects

"ROOT 7"

- New interfaces, using modern C++ for simplicity, robustness and speed
  - change interfaces after 20 years, and then freeze them again
- Keep interfaces readable for current ROOT users
  - `canv->cd(); hist->Draw();` becomes `canv->Draw(hist);`
- Expose new interfaces early, release gradually
  - see ROOT::Experimental [link], available with -Dcxx14=On

Main Development Areas

Parallelization

- Implicit multi-threading:
  - you ask ROOT to do something, and it does it using all your cores
- Declarative programming for analysis:
  - you tell ROOT *what* to do but not *how*. It knows how, does it in parallel.
- Vectorization:
  - we run hot, numerical loops on multiple data, targeted to your CPU

Math

- TMVA
  - fast data connections to external tools (TensorFlow etc)
  - machine learning implementations targeted to HEP
- RooFit will not be forgotten, either ;-)
- The HEP Common Math Library
  - e.g. random numbers: efficient, also for multi-threaded environments
  - vectorized functions

I/O, TTree

- Want to to be extremely performant:
  - 0-copy where possible
  - I/O using all cores, best compression algorithms
  - multi-thread-friendly: one tree, many entries analyzed by multiple threads
- Robust interfaces: type-safe (no void*), explicit memory ownership
- Optimized for I/O devices of 2020: SSD, 3D XPoint, network

Histograms

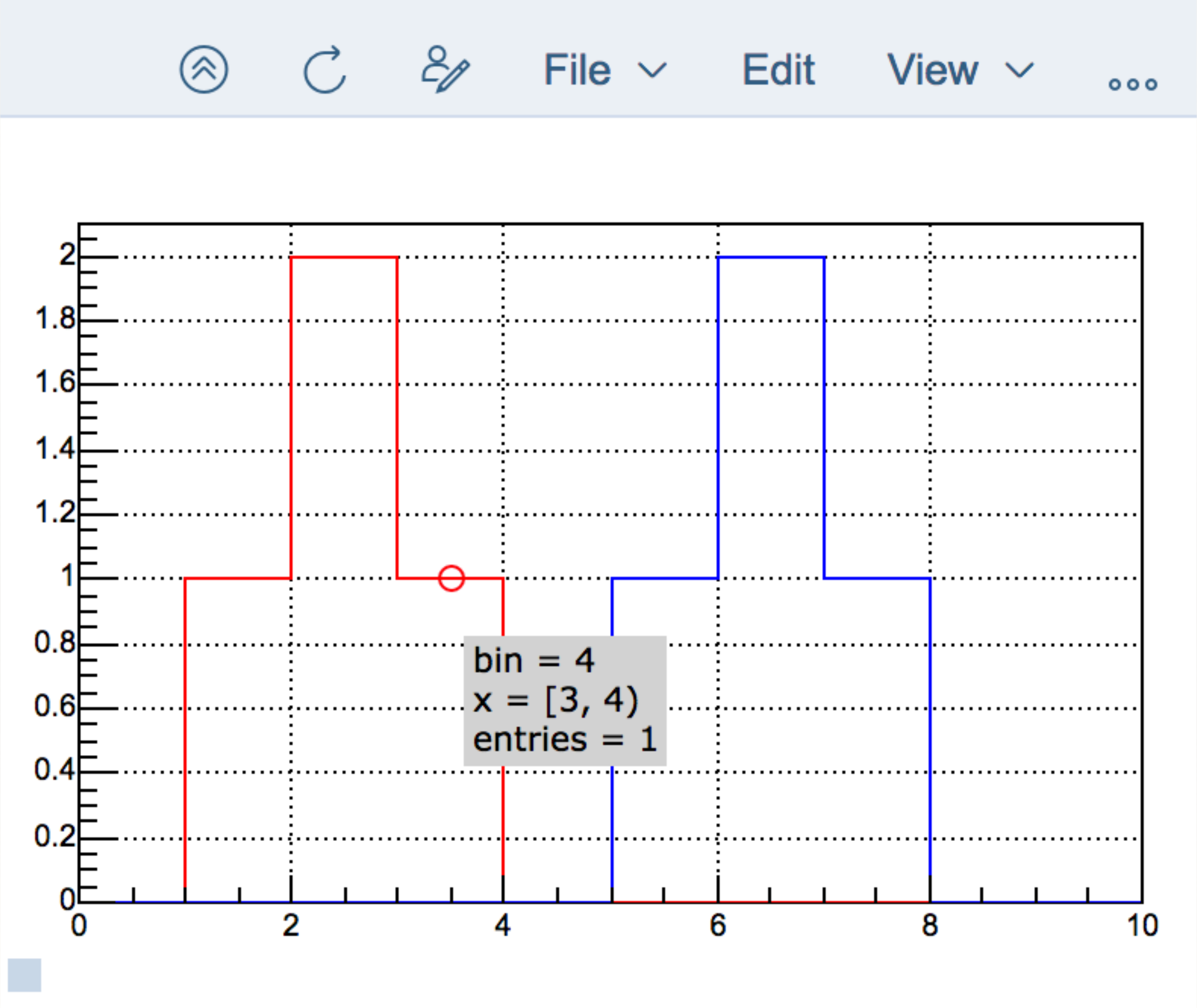
```
// Create a 2D histogram with an X axis with
// equidistant bins, and a y axis with irregular
// binning.
Experimental::TH2D hist({100, 0., 1.},
                        {{0., 1., 2., 3., 10.}});

// Fill weight 1. at the coordinate 0.01, 1.02.
hist.Fill({0.01, 1.02});
```

- Fast and simple
  - shield advanced features
    from basic ones: offer both high-performance interface and usability layer
- Composable and configurable, enabling histogram algorithm library, operating on “any” histogram
- Transform embedded histogram concepts into first-class citizens:
  - axis definition, histogram range, iteration, bin index, bin content storage

Parallel, Simple Analysis

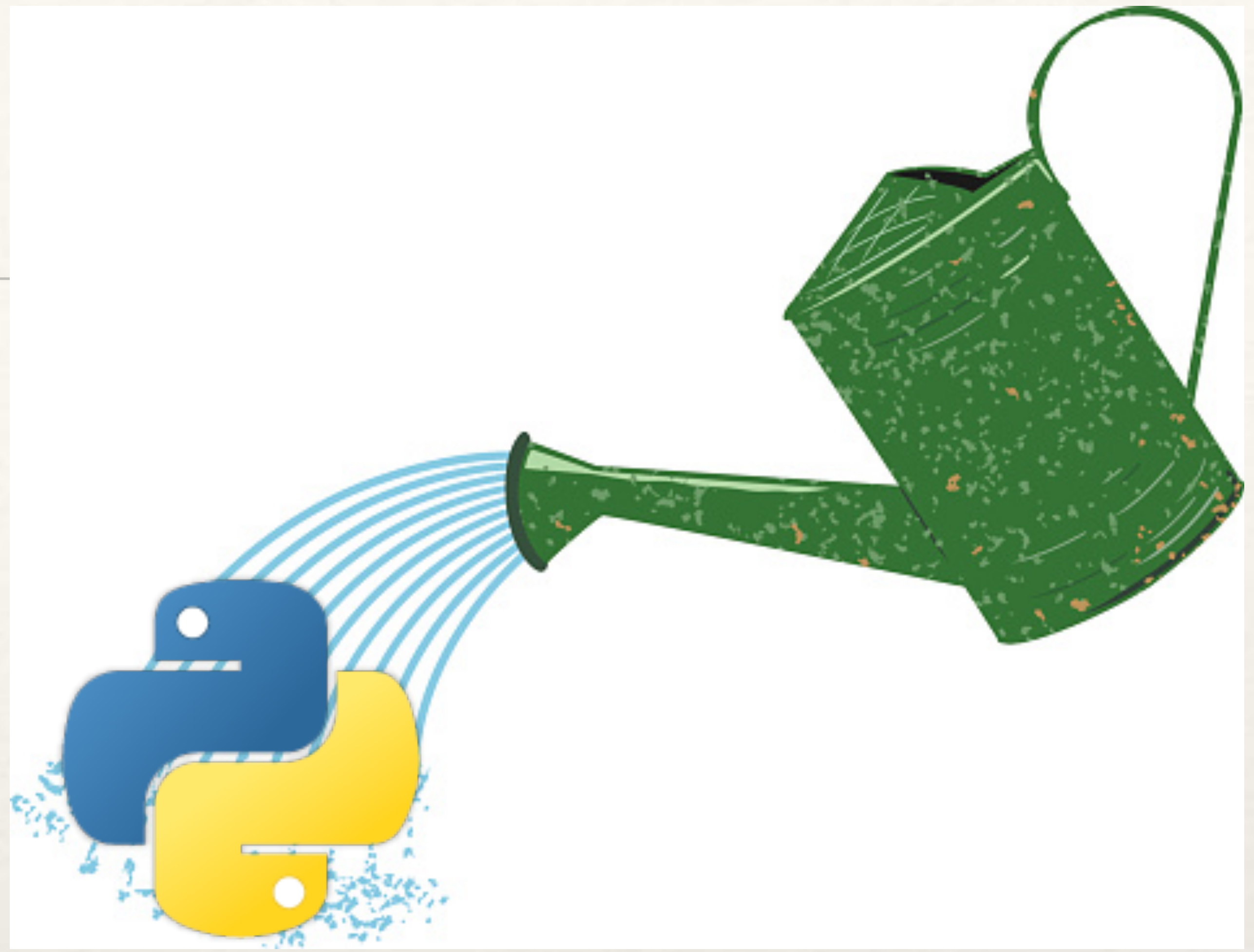
- Currently: you specify reading, looping, selections, output / slimming / skimming (= caching), histogramming; always run everything and on one core - or have smart code and spend time on infrastructure (or TSelector)
- What we want:
  - you focus on the selection, projections, algorithms
  - ROOT takes care of the boring stuff: reading, looping, scheduling, parallelizing - as efficiently as possible
  - with beautiful and efficient Python interfaces

WebGUI Graphics



- WebGUI Graphics = HTML + JavaScript + CSS + OpenUI5 + three.js plus D3.js
- Replace custom GUI with Win32 GDK, X11, Cocoa and GL back-ends (and what about Wayland?!)
- Local and remote interaction, extensible painters, future-proof, beautiful graphics

PyROOT



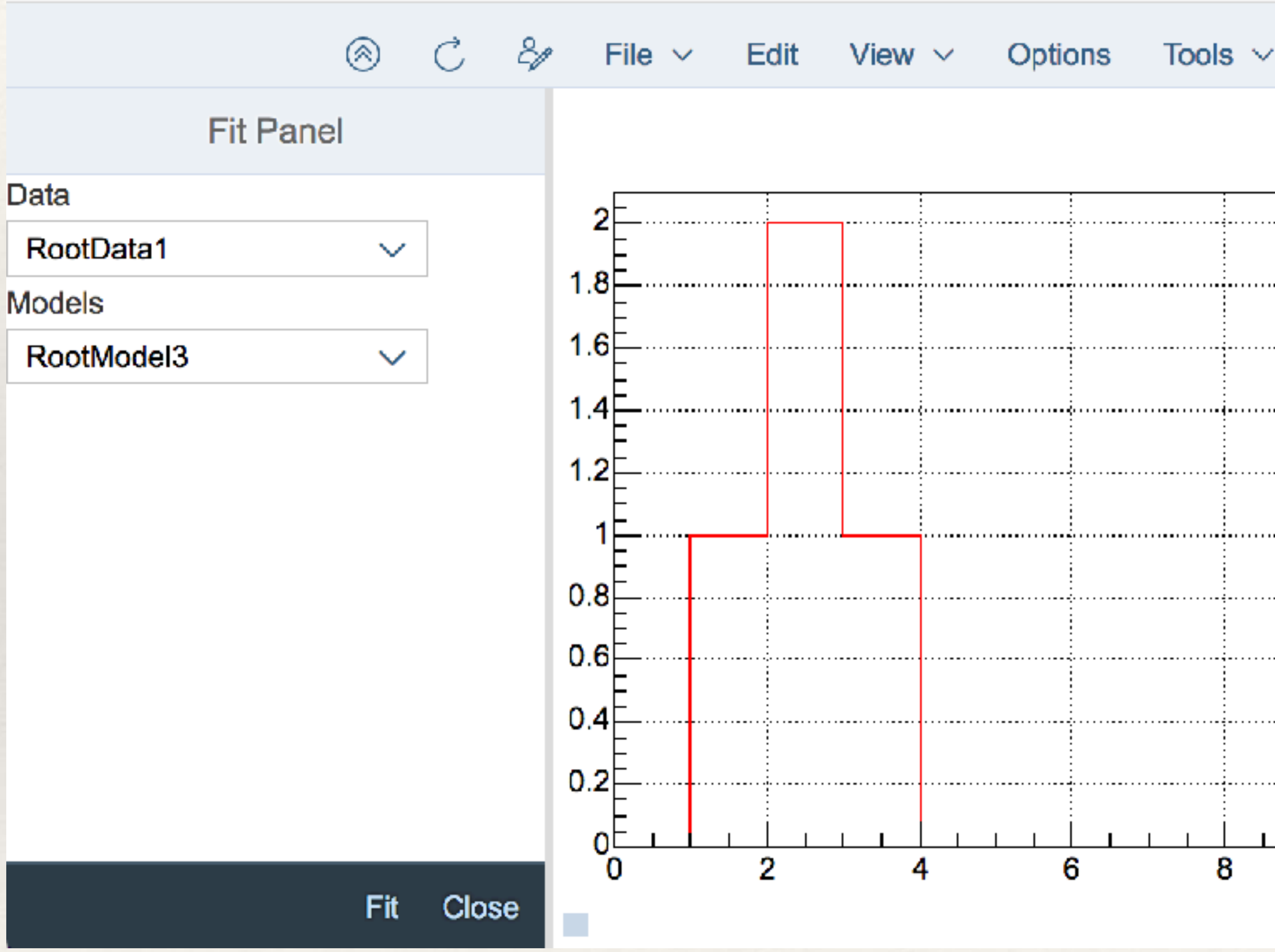
- It's unique - the world is jealous.
  Move from maintaining it to growing it!
- Expand it beyond "C++ to Python":
  - add "pythonic" interfaces a la rootpy
- Enable fast-path interfaces, e.g. to numpy arrays
- Design C++ interfaces such that they play nicely with Python
  - ownership, type-safety, compact code

Build and Install

- Make binary installs simple for users
  - easy install means happy physicists
  - install core parts, build extensions as needed, on demand
- Make it simple to build
  - allow for e.g. experiment's of physics group's extensions
  - ROOT "package manager"

Summary: ROOT @ 2020

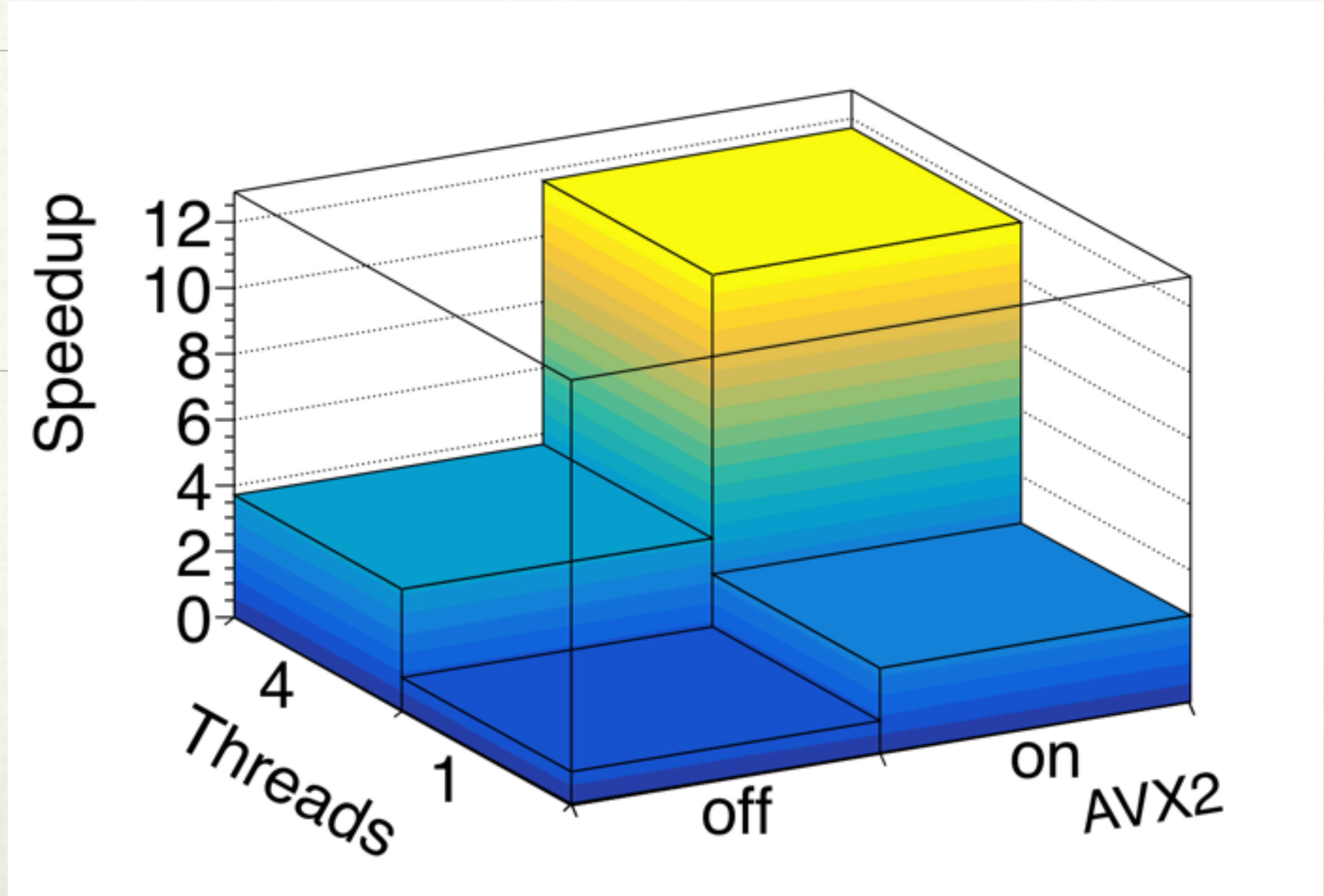
- New histograms, new TTree: simpler and more robust
- Web-based graphics, with new TCanvas etc
- Simple, efficient and composable analysis using all your cores
- Passing data efficiently into machine learning tools, be it TMVA or external



ROOT v6.12

(and a bit of v6.10)

Parallelization



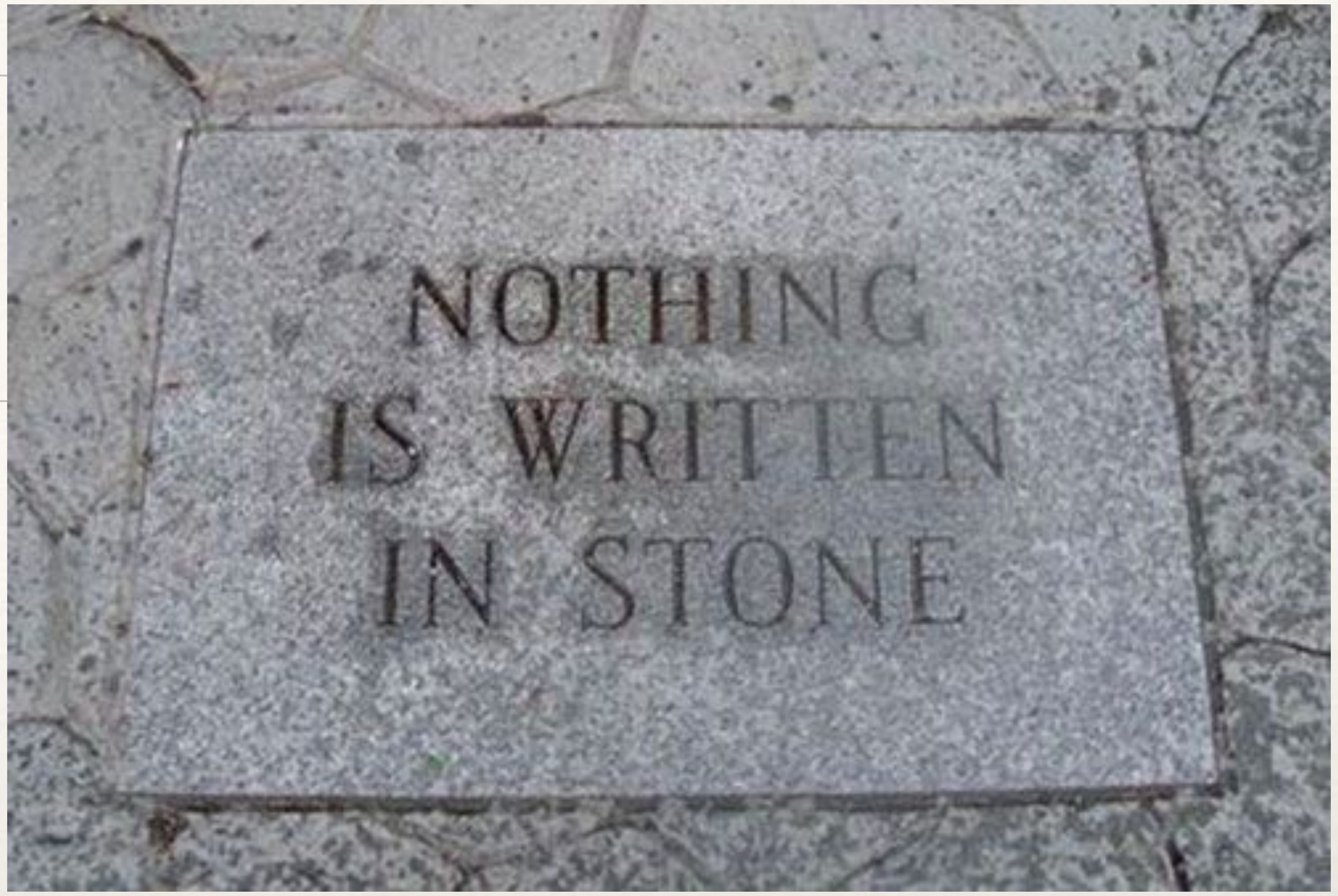
- Implemented parallel reading, writing, and fitting
  - `ROOT::EnableImplicitMT()` switches ROOT to parallel mode
  - `root -t` is a shortcut
- If ROOT is configured for SSE4 or AVX2, fitting is vectorized!
  - next, we'll fix the "if configured" part!

Math

- TMVA
  - new Deep Neural Network (working in parallel in CPU or GPU)
  - interfaces to Keras (PyKeras) which can use Theano or Tensorflow
  - improved support for multi-class classification
- Nicer TF1 construction

```
// Composition:
TF1 comp("sin( f1(x) )");
// Sum of normalized functions:
TF1 nsum("NSUM([A]*gaus, [B]*expo)",
  xmin, xmax);
// Convolution:
TF1 conv("CONV(expo, gaus)”, xmin, xmax);
```

I/O



- LZ4 compression: super-fast reading, approx 15% larger files
  - default for v6.14?
- TTreeReader has support for TEntryLists
  - TTreeReader became *the* way to read trees (if not TDataFrame!)

TDataFrame

- From tutorials/dataframe/tdf001_introduction.C:

```
ROOT::Experimental::TDataFrame d(treeName, fileName, {“b1"});
auto cutb1 = [](double b1) { return b1 < 5.; };

                d.Filter(cutb1) // <- no column name specified here!
```

TDataFrame

- From tutorials/dataframe/tdf001_introduction.C:

```
ROOT::Experimental::TDataFrame d(treeName, fileName, {“b1"});
auto cutb1 = [](double b1) { return b1 < 5.; };
auto cutb1b2 = [](int b2, double b1) { return b2 % 2 && b1 < 4.; };
                d.Filter(cutb1) // <- no column name specified here!
                 .Filter(cutb1b2, {"b2", "b1"})
```

TDataFrame

- From tutorials/dataframe/tdf001_introduction.C:

```
ROOT::Experimental::TDataFrame d(treeName, fileName, {“b1"});
auto cutb1 = [](double b1) { return b1 < 5.; };
auto cutb1b2 = [](int b2, double b1) { return b2 % 2 && b1 < 4.; };
auto entries1 = d.Filter(cutb1) // <- no column name specified here!
                 .Filter(cutb1b2, {"b2", "b1"})
                 .Count();
```

TDataFrame

- From tutorials/dataframe/tdf001_introduction.C:

```
ROOT::Experimental::TDataFrame d(treeName, fileName, {“b1"});

auto cutb1b2 = [](int b2, double b1) { return b2 % 2 && b1 < 4.; };



auto b1b2_cut = d.Filter(cutb1b2, {"b2", "b1"});
auto minVal = b1b2_cut.Min();
auto maxVal = b1b2_cut.Max();
auto meanVal = b1b2_cut.Mean();
auto nonDefmeanVal = b1b2_cut.Mean("b2"); // <- Column is not the default
```

TDataFrame

- From tutorials/dataframe/tdf001_introduction.C:

```
ROOT::Experimental::TDataFrame d(treeName, fileName, {“b1"});
auto cutb1 = [](double b1) { return b1 < 5.; };




auto hist = d.Filter(cutb1).Histo1D();
```

TDataFrame

- From tutorials/dataframe/tdf007_snapshot.C:

```
ROOT::Experimental::TDataFrame d(treeName, fileName);
auto d_cut = d.Filter("b1 % 2 == 0");
```

TDataFrame

- From tutorials/dataframe/tdf007_snapshot.C:

```
ROOT::Experimental::TDataFrame d(treeName, fileName);
auto d_cut = d.Filter("b1 % 2 == 0");
auto d2 = d_cut.Define("b1_square", "b1 * b1")
```

TDataFrame

- From tutorials/dataframe/tdf007_snapshot.C:

```
ROOT::Experimental::TDataFrame d(treeName, fileName);
auto d_cut = d.Filter("b1 % 2 == 0");
auto d2 = d_cut.Define("b1_square", "b1 * b1")
               .Define("b2_vector",
                       [](float b2) {
                          std::vector<float> v;
                          for (int i = 0; i < 3; i++)
                             v.push_back(b2 * i);
                          return v;
                       },
                       {"b2"});
```

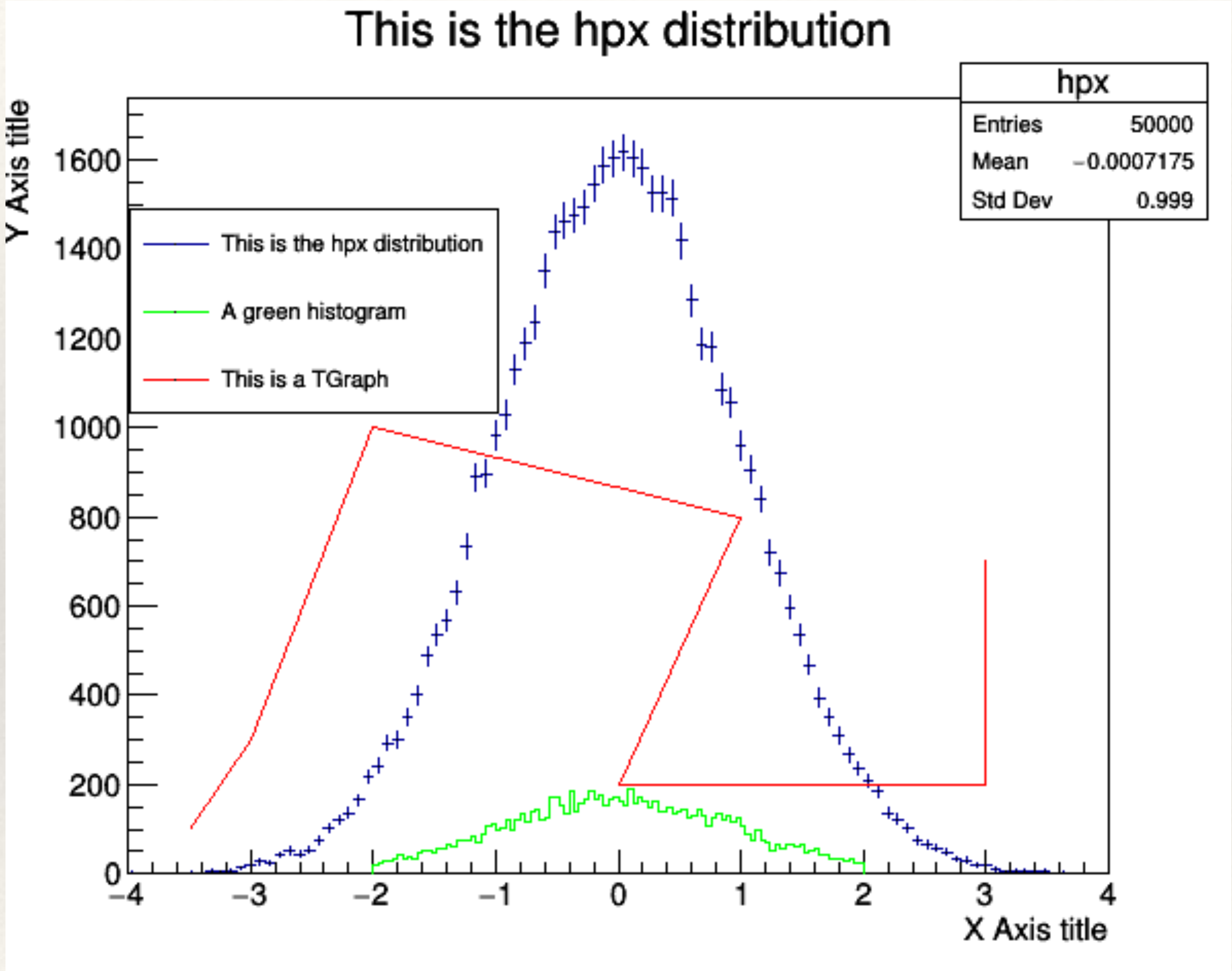
TDataFrame

- From tutorials/dataframe/tdf007_snapshot.C:

```
ROOT::Experimental::TDataFrame d(treeName, fileName);
auto d_cut = d.Filter("b1 % 2 == 0");
auto d2 = d_cut.Define("b1_square", "b1 * b1")
               .Define("b2_vector",
                       [](float b2) {
                          std::vector<float> v;
                          for (int i = 0; i < 3; i++)
                             v.push_back(b2 * i);
                          return v;
                       },
                       {"b2"});
d2.Snapshot(treeName, outFileName,
            {"b1", "b1_square", "b2_vector"});
```

Graphics

- Two major feature requests implemented
  - automatic palette colors, e.g. line: `hist->Draw(“PLC”)`
  - auto-placement, e.g. `canvas->BuildLegend()`
  - “do the right thing” options!
- Plus constant flow of smaller improvements, e.g. “BOX1” TH3 option



ROOT std:: backports

- We loved std::string_view even before C++17. Same with std::make_unique, std::span, etc (and soon likely std::variant)
- ROOT injects implementations of these into std::
    - only if your stdlib does not have it
    - once it does, uses std::experimental::XYZ or std::XYZ
- Allows us all to use current and near-future features with older compilers!

Current "v7" Features

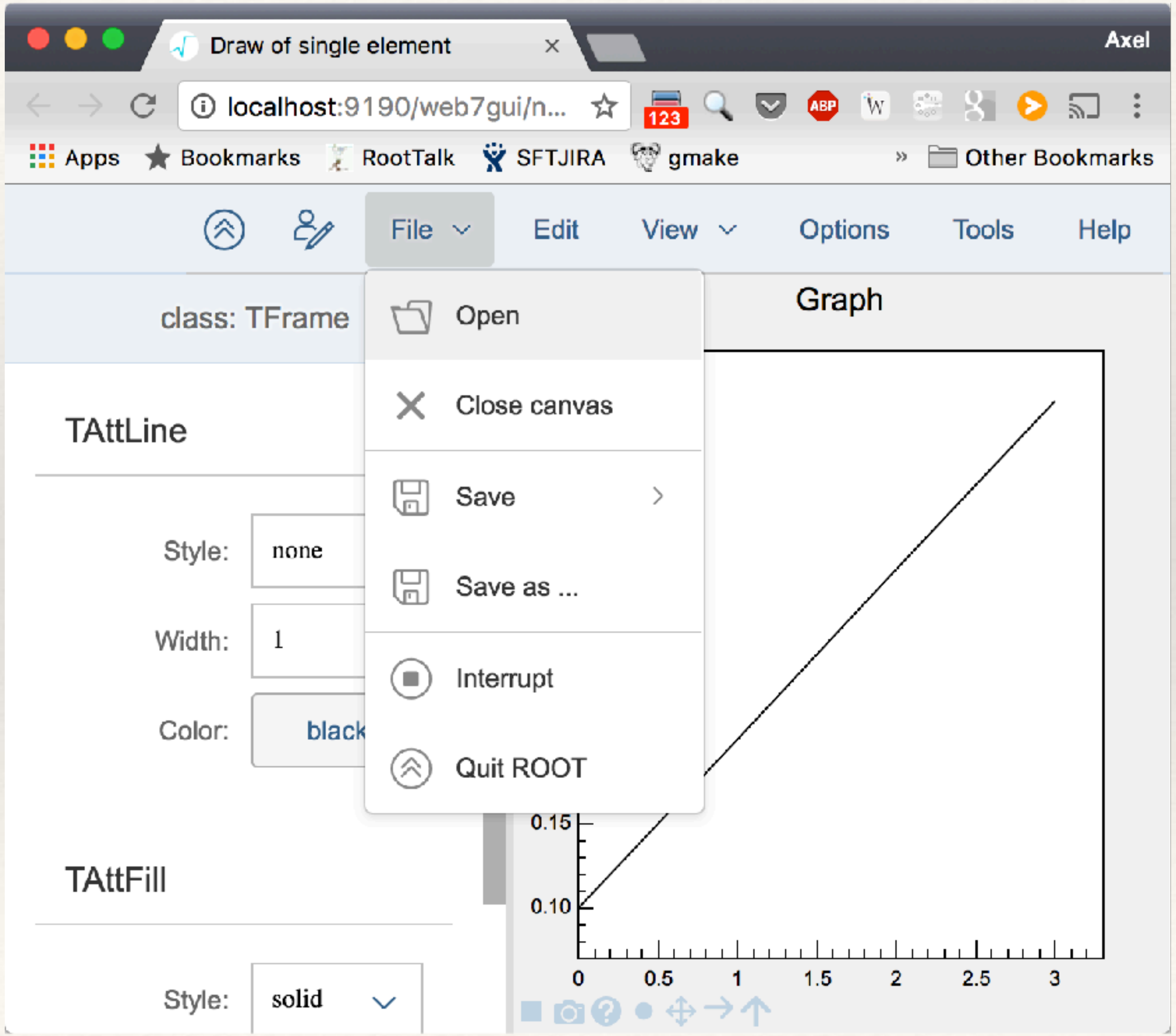
- **THist**, **TFile**, **TPad** / **TCanvas** (new!)
  - with explicit "pixel" / "normal" / "user" coordinates
- Decided on new interface personality: ownership, separation of simple / advanced interfaces, safer code through array spans + unique_ptr +…
- Features added continuously
- Release from Experimental:: as use suggests and stabilization allow

Example: THist

- Fast: less virtual interfaces, more inlined, more bulk data operations
- Safe: 1D histogram has no `hist->GetBinError(x, y)`
- Simple: keeps most interface names `TH1F::Fill()`, `TH2D::GetEntries()`
- Thread-safe: no directory registration, no raw pointers, explicit ownership
- Focused: no ~~THist::SetLineColor()~~
- Yet composable and configurable for experts: statistics, storage

And It Works!

```
$ root -l tutorials/v7/draw_v6.cxx
root [0]
Processing tutorials/v7/draw_v6.cxx...
Info in <TCivetweb::Create>: Starting HTTP
 server on port 9504
```



Conclusion

Bottom Line

- ROOT's main goals:
  - simplicity
  - robustness
  - speed
- Keep ROOT at the heart of physicists' data analysis, and make it nice!
- Focus on physicists! Efficiency: brain / second, more than CPU / second

Conclusion

- New interfaces == new momentum
  - plus several new team members
- TDataFrame!

Your Core ROOT Team

Xavi [1], Vassil [2], Sergei[3], Raphael [4], Philippe [5], Olivier [1], Oksana [6], Lorenzo [1], Kim [1], Guilherme [1], Enrico [1], Enric [1], Danilo [1], Bertrand [1], Axel [1]

1: CERN
2: Princeton University
3: GSI
4: Chalmers University
5: Fermilab
6: University of Nebraska

Plus several regular +
essential contributors!

@ROOT

- https://root.cern
- https://root-forum.cern.ch
- https://root.cern/bugs