Pythonic HEP ecosystem update

Jim Pivarski

Princeton University – IRIS-HEP

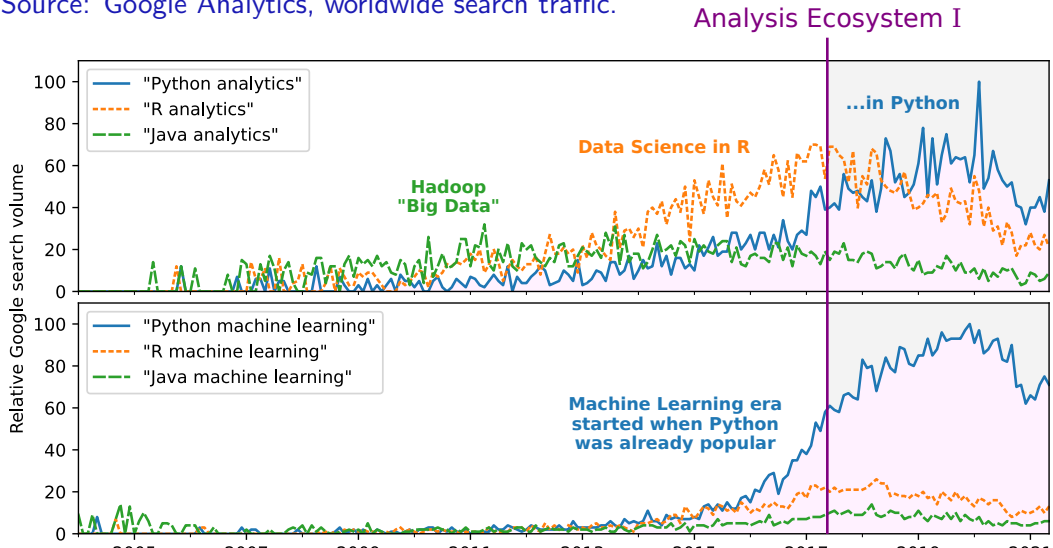
May 23, 2022

From the first Analysis Ecosystem Workshop report (4 Aug 2017)

In terms of languages, amid the many languages in use there is consensus on the special roles of some. Performance is key in HEP, and certainly when we talk about reconstruction and performance-critical code, we will continue to look to C++, because the community has invested a lot in this language. It is not the only high performance language, but it is and (in the estimation of the workshop) will remain the dominant one. However, there was a very strong message from the community that for analysis code, python is highly desirable especially because of the fast development cycle and in general people can achieve the same results with fewer lines of code. Python should be treated as a first class language. Performance-critical code should be offloaded to libraries, probably written in C++, but the analysis code can be python. Python is also the language of choice in the wider scientific data analytics community. Sustaining PyROOT in particular is a key concern of the community. There is an argument that we lose C++ experience in the community by encouraging python but the overall view was that freeing up analysts time by enabling faster analysis turnaround is the best way to free up time for important work outside of analysis. An important point for language choices is that we need to ensure that senior people also need to be included, requiring adequate education and documentation. End users should be exposed to simple interfaces, which should reduce the complexity of analysis code.

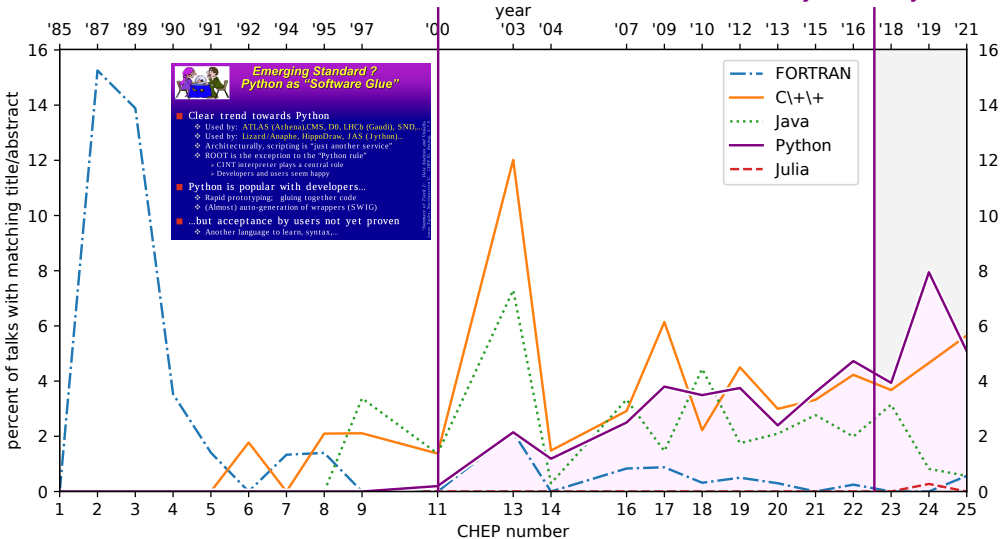
Back then, Python was just beginning to take over data analytics

Source: Google Analytics, worldwide search traffic.



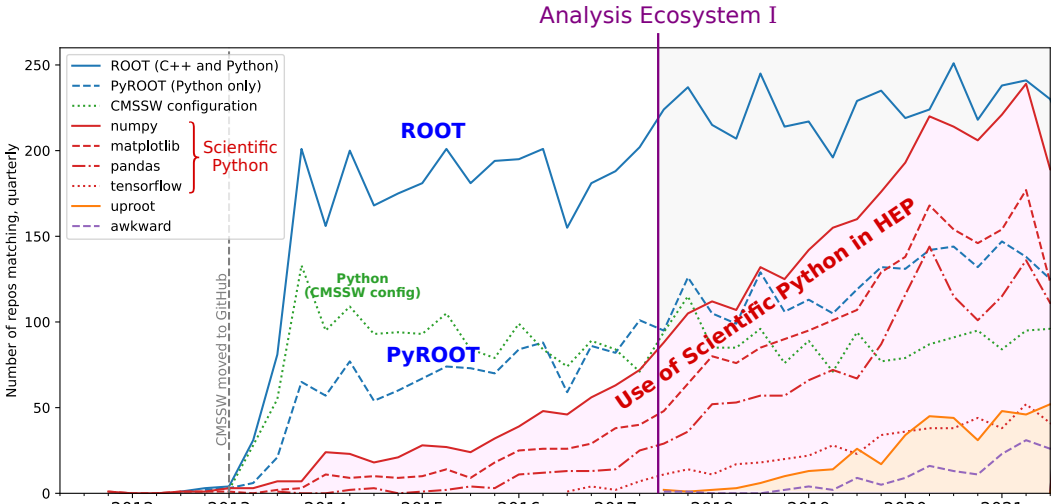
Though it had been slowly growing in HEP for years

Source: Title and abstract matches in CHEP proceedings.



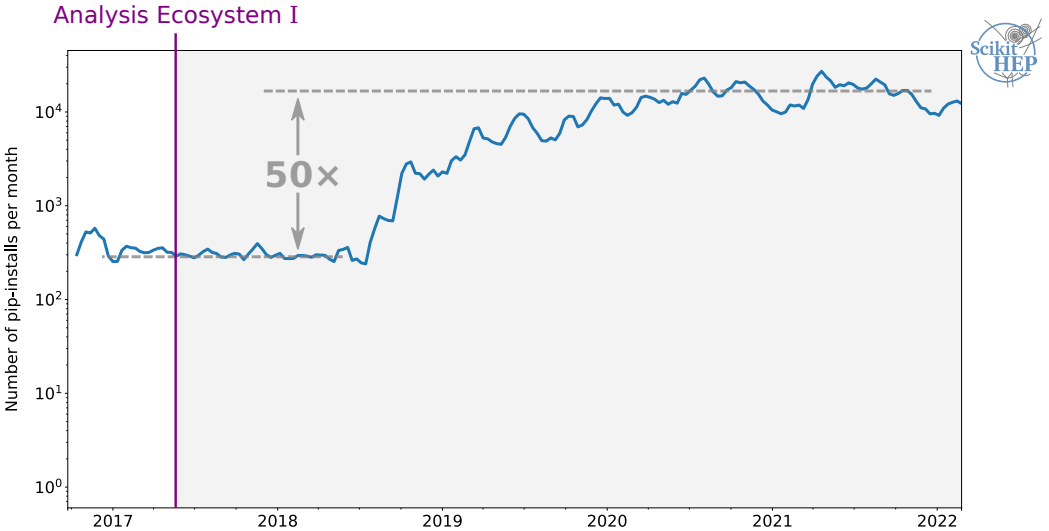
But the use of Scientific Python (NumPy, etc.) was new to HEP

Source: "`import XYZ`" matches in GitHub repos for users who fork CMSSW.



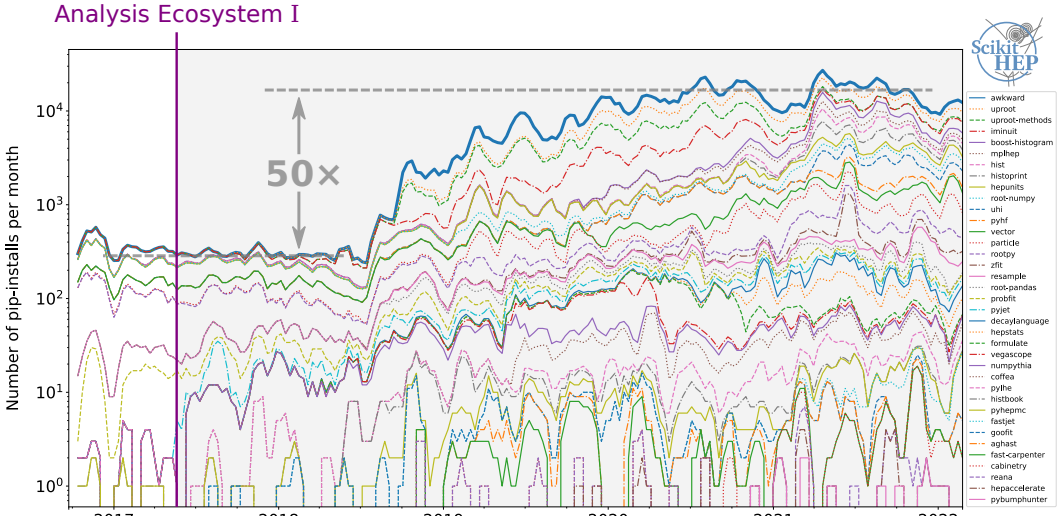
Scikit-HEP had not yet taken off

Source: "`pip install XYZ`" download rate for MacOS/Windows (no batch jobs).



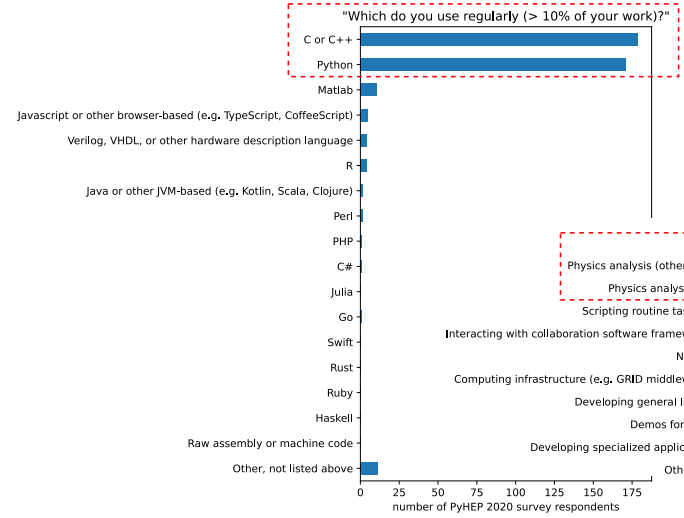
Scikit-HEP had not yet taken off

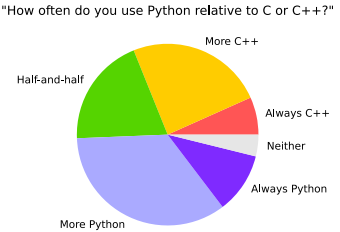
Source: "`pip install XYZ`" download rate for MacOS/Windows (no batch jobs).

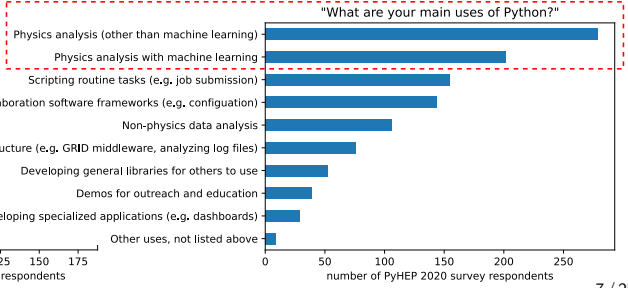


We are now a two-language community

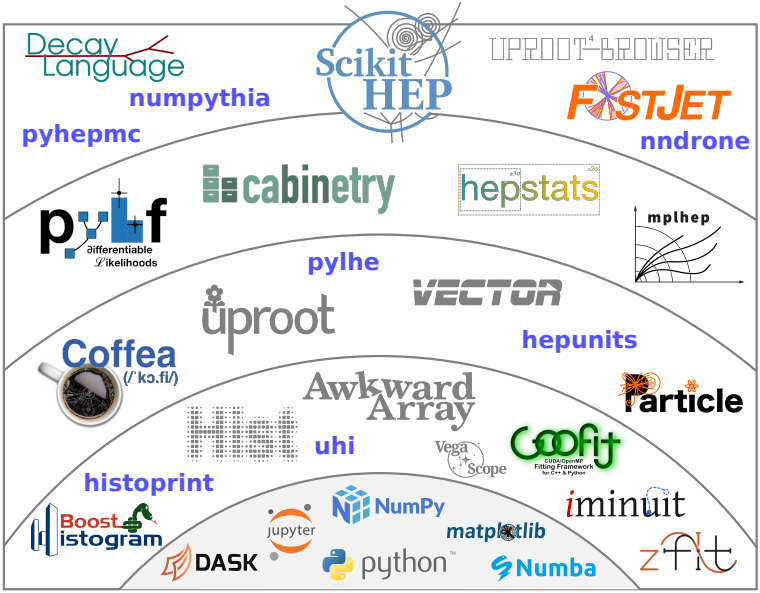
Source: PyHEP 2020 workshop survey.







The Pythonic HEP ecosystem now



Case study: convergence of histogram libraries

The Scientific Python world lacked HEP-style histograms; it's one of the things we have to make ourselves.

Case study: convergence of histogram libraries

The Scientific Python world lacked HEP-style histograms; it's one of the things we have to make ourselves.

Also, it seems easy: just bin and count, right?

Case study: convergence of histogram libraries

The Scientific Python world lacked HEP-style histograms; it's one of the things we have to make ourselves.

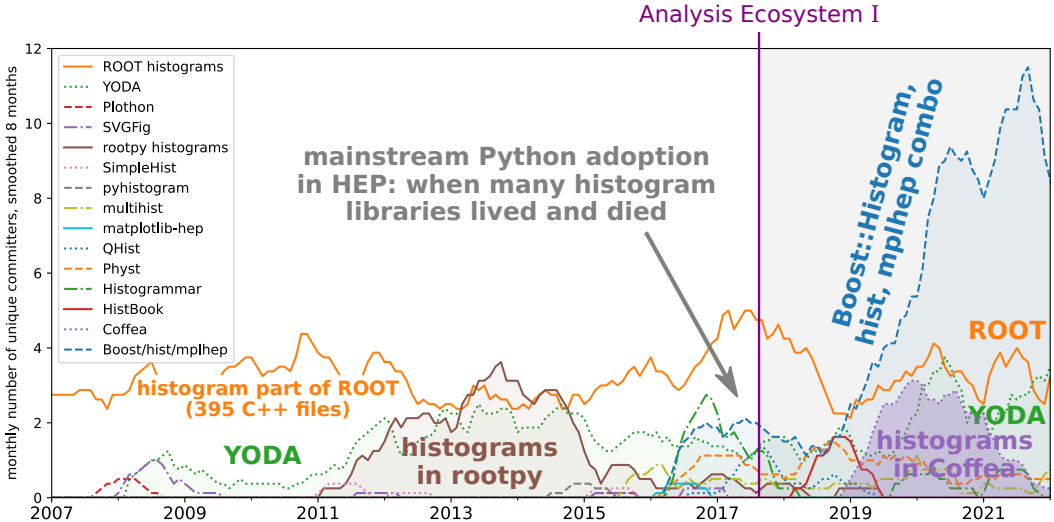
Also, it seems easy: just bin and count, right?

Physicists have created at least 20 histogram libraries in Python, most single-author.

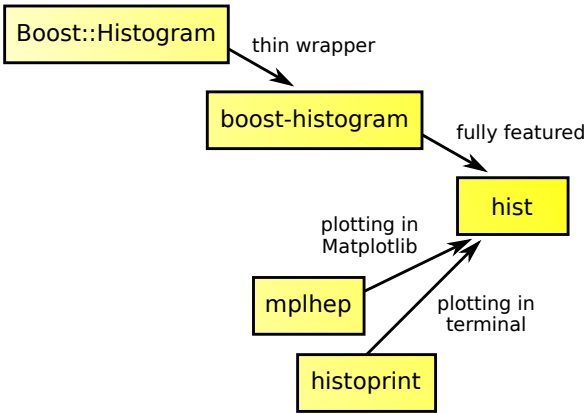
- PyROOT (2004–now)
- PAIDA (2004–2007)
- Plothon (2007–2008)
- SVGFig (2008–2009)
- YODA (2008–now)
- DANSE (2009–2011)
- rootpy (2011–2019)
- SimpleHist (2011–2015)
- pyhistogram (2015)
- multihist (2015–now)
- matplotlib-hep (2016)
- QHist (2017–2019)
- Physt (2016–now)
- Histogrammar (2016–now)
- HistBook (2018–2019)
- Coffea.hist (2019–2022)
- boost-histogram (2019–now)
- mplhep (2019–now)
- histoprint (2020–now)
- hist (2020–now)

Histogram proliferation and convergence

Number of unique developers contributing to each library per month (in git).

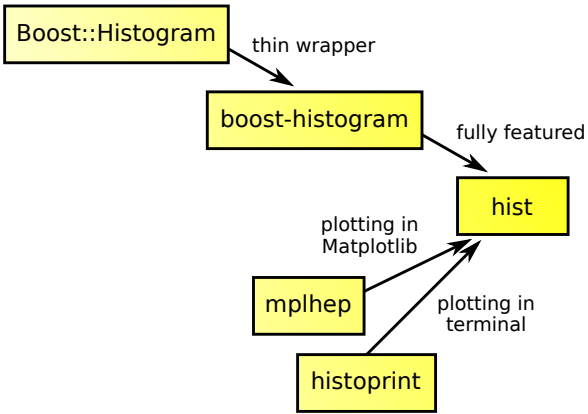


Why combine Boost::Histogram, hist, mplhep?



Originally, each of these was developed independently by a single author.

Why combine Boost::Histogram, hist, mplhep?

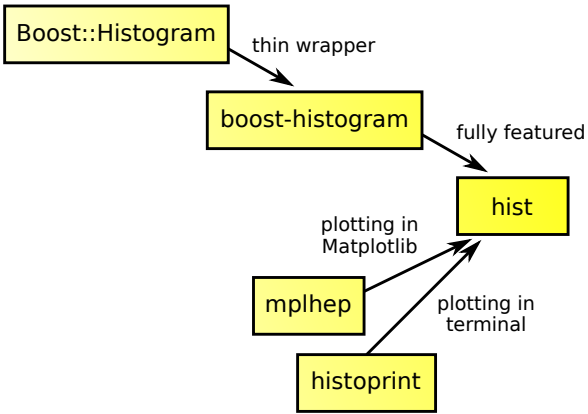


Originally, each of these was developed independently by a single author.

They each provide a piece of functionality users can get through

```
import hist
```

Why combine Boost::Histogram, hist, mplhep?



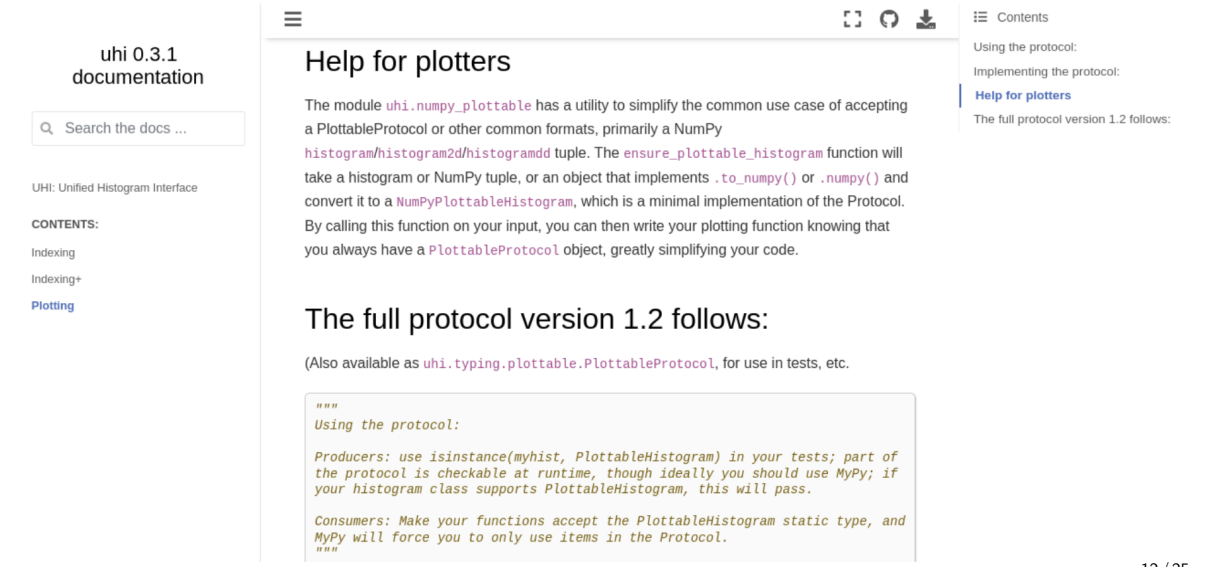
Originally, each of these was developed independently by a single author.

They each provide a piece of functionality users can get through

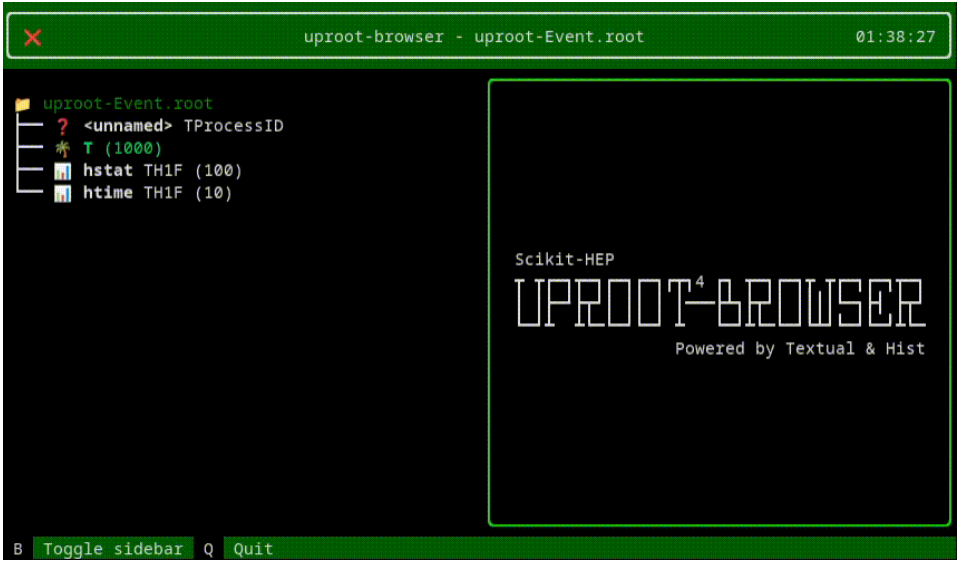
```
import hist
```

Now, 47 developers have contributed to these packages, and 20 contributed to more than one.

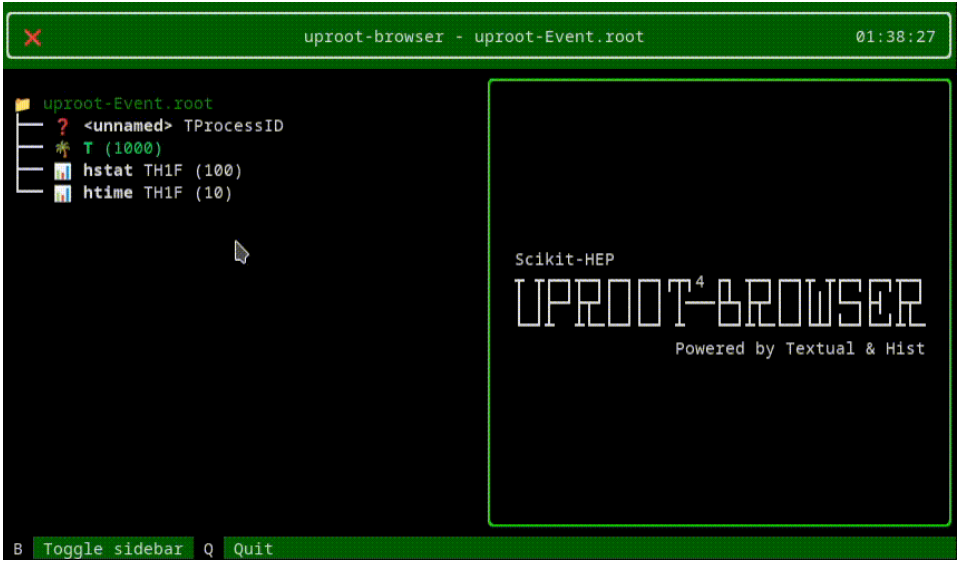
Consistency maintained through agreed-upon protocols



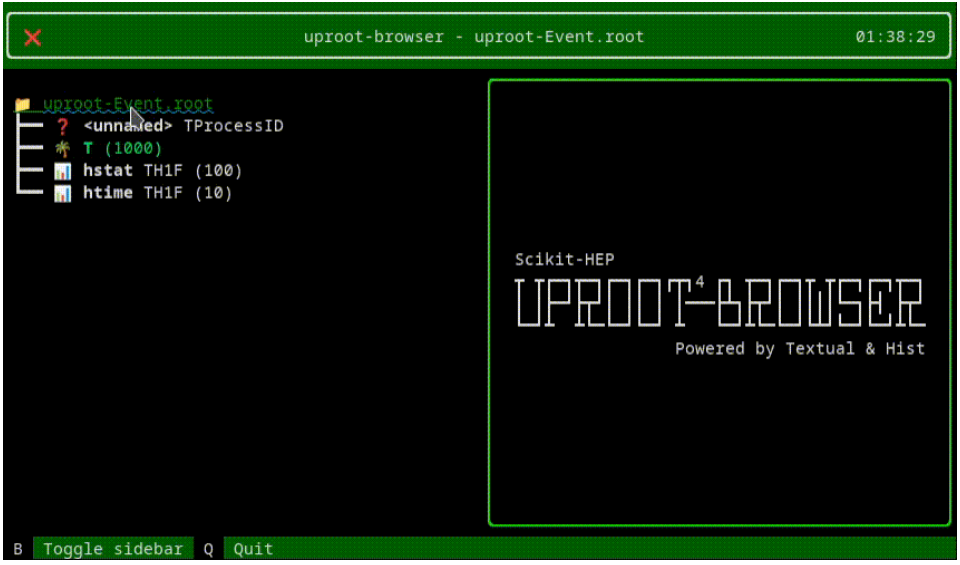
uproot-browser: another interface package that pulls many together



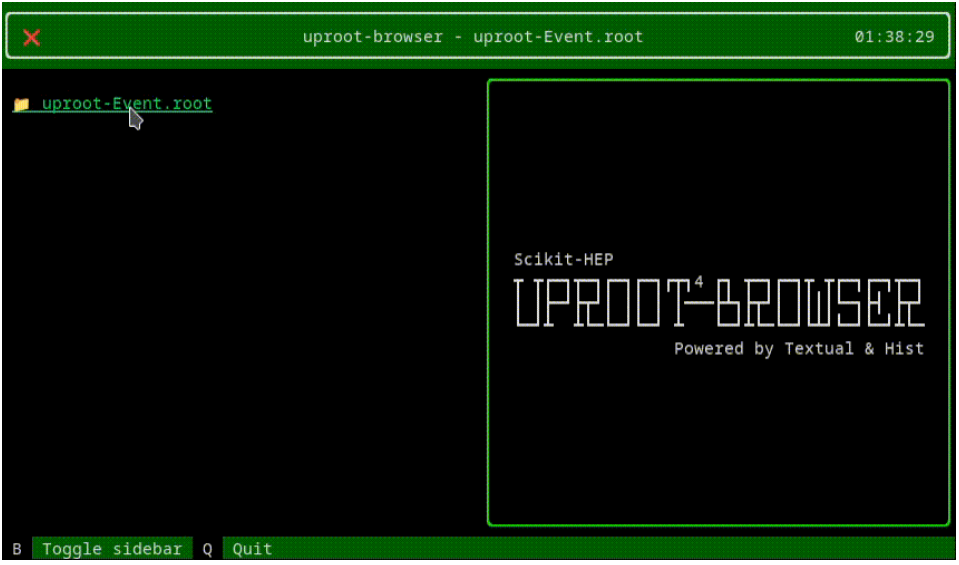
uproot-browser: another interface package that pulls many together



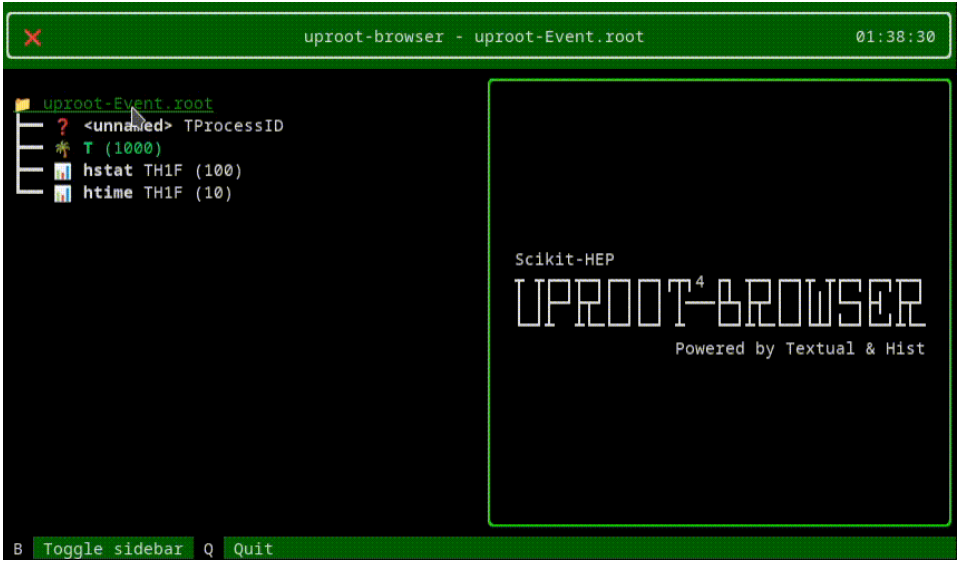
uproot-browser: another interface package that pulls many together



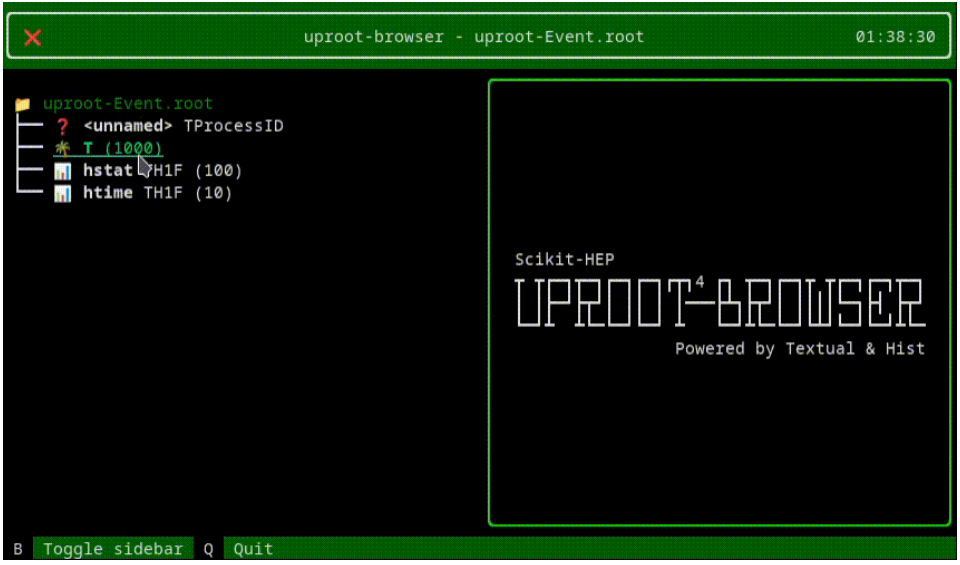
uproot-browser: another interface package that pulls many together



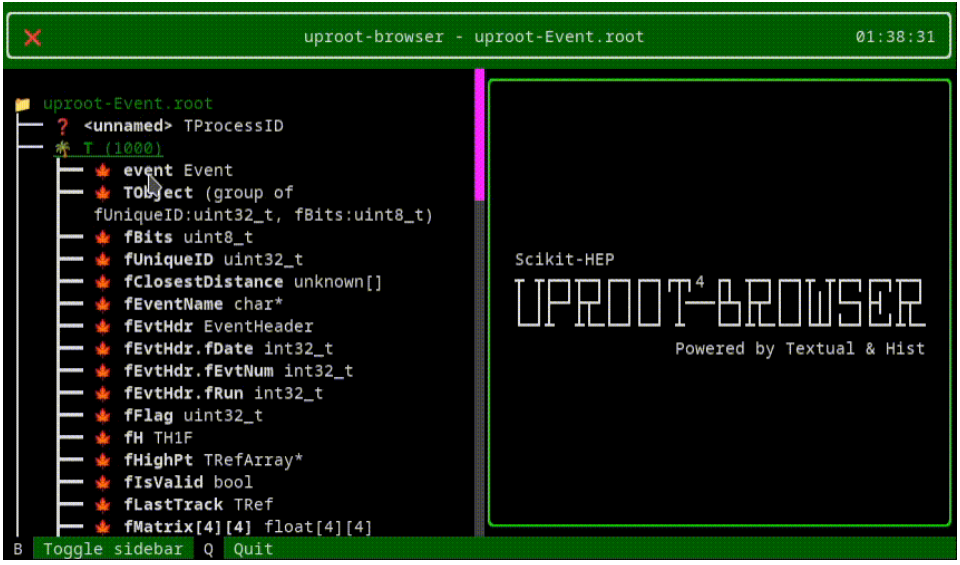
uproot-browser: another interface package that pulls many together



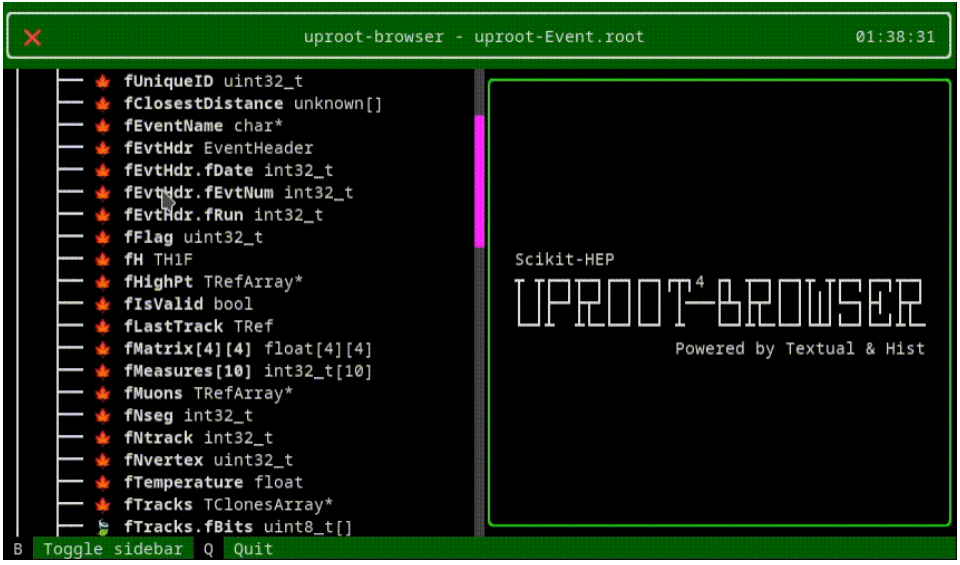
uproot-browser: another interface package that pulls many together



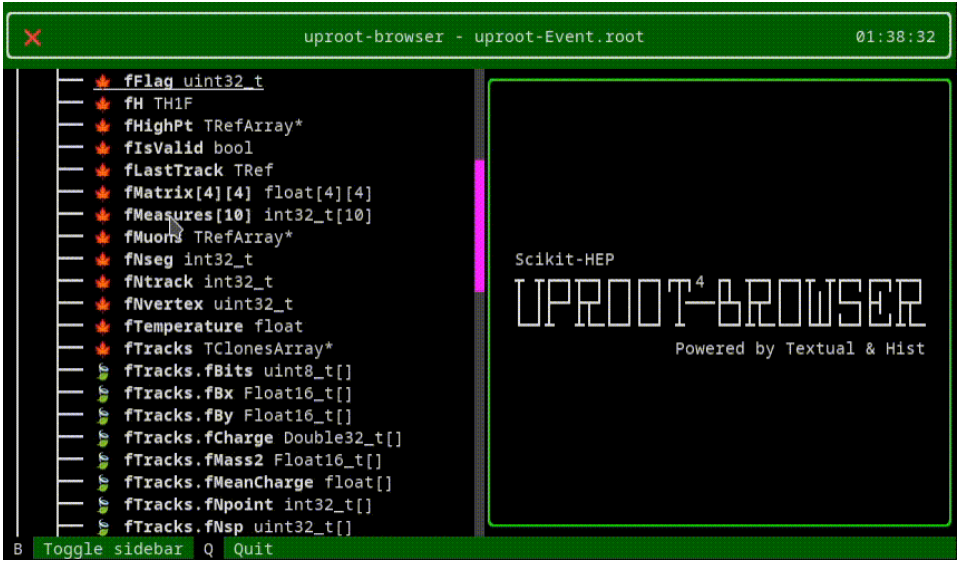
uproot-browser: another interface package that pulls many together



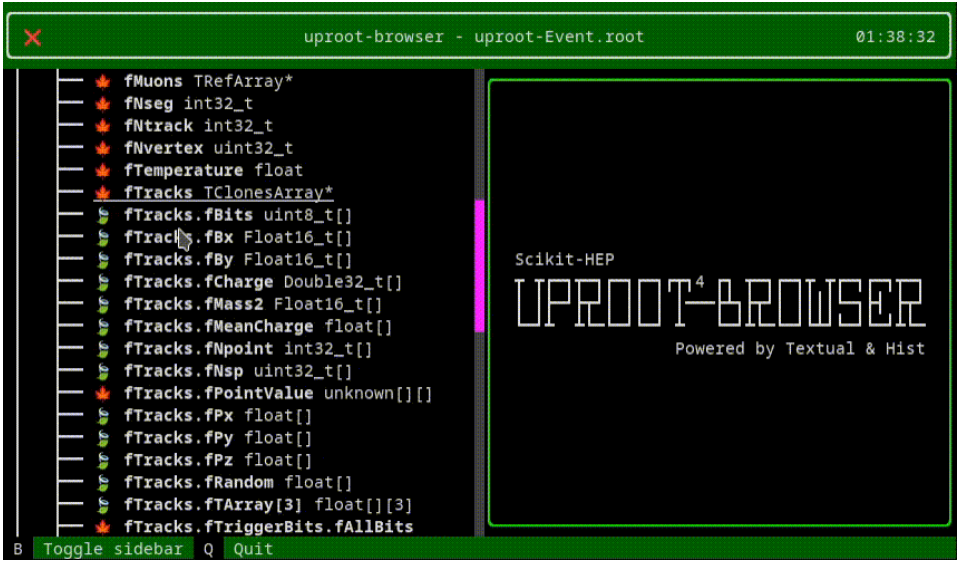
uproot-browser: another interface package that pulls many together



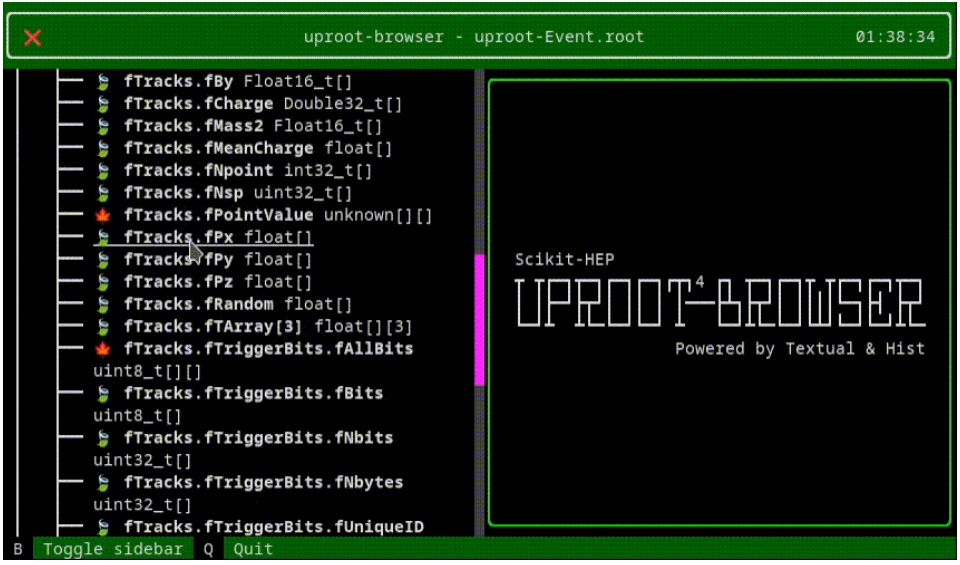
uproot-browser: another interface package that pulls many together



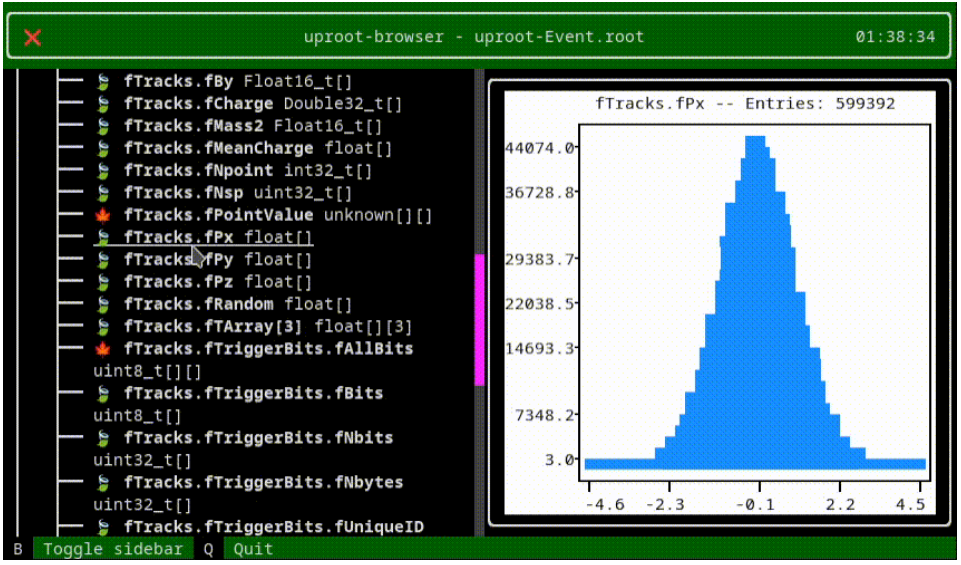
uproot-browser: another interface package that pulls many together



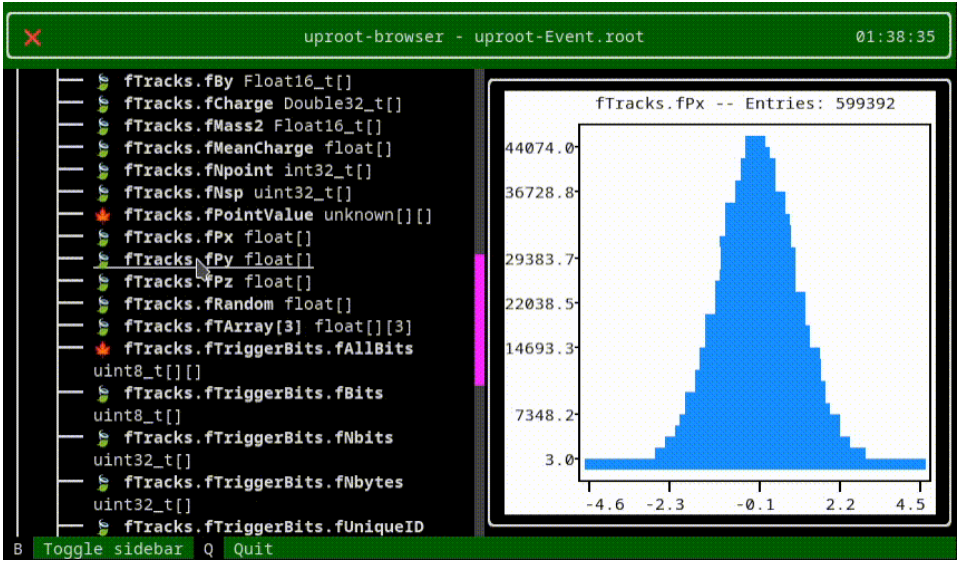
uproot-browser: another interface package that pulls many together



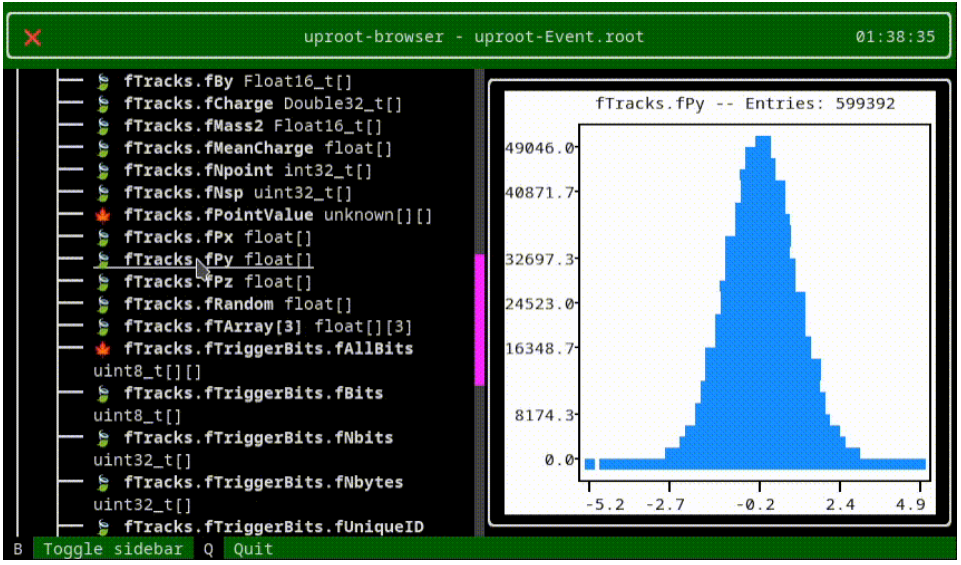
uproot-browser: another interface package that pulls many together



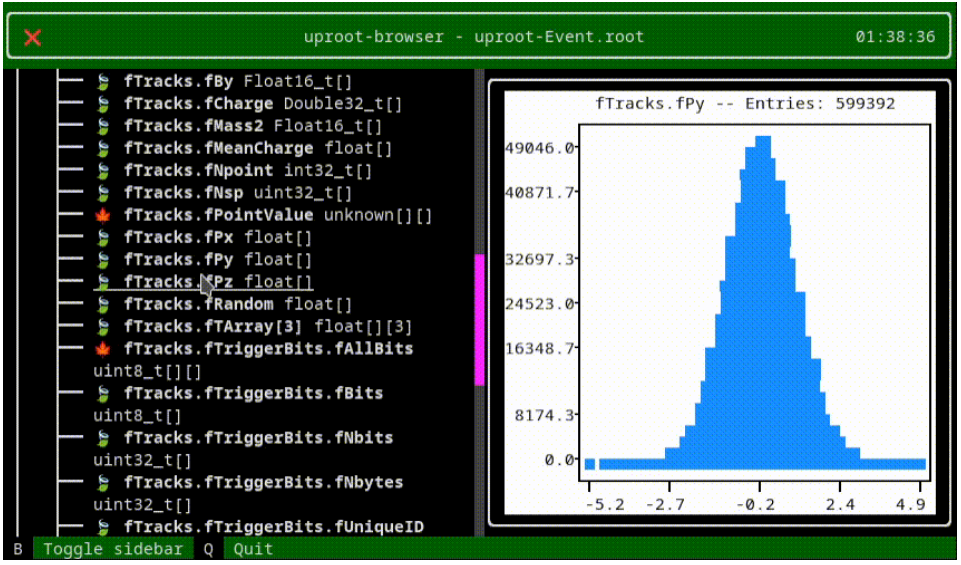
uproot-browser: another interface package that pulls many together



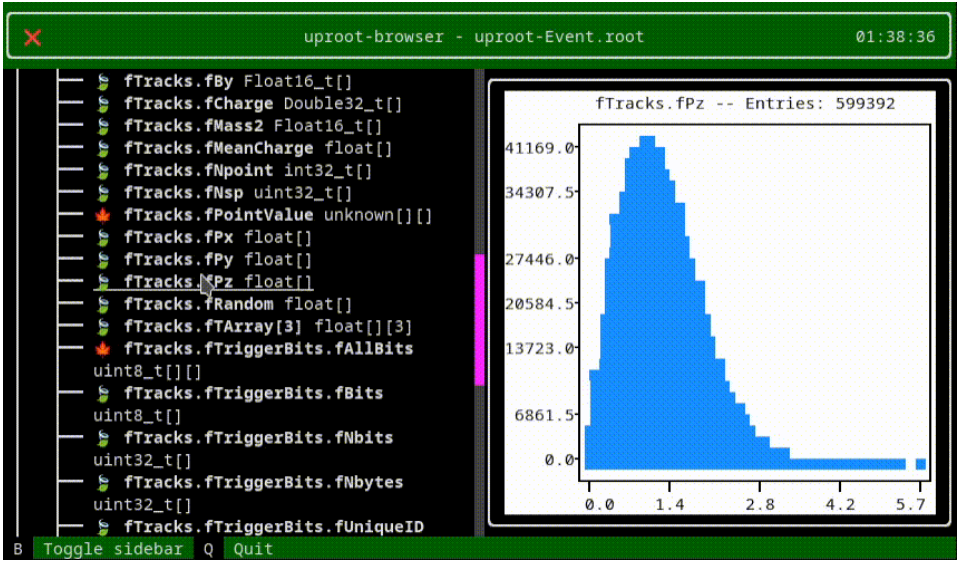
uproot-browser: another interface package that pulls many together



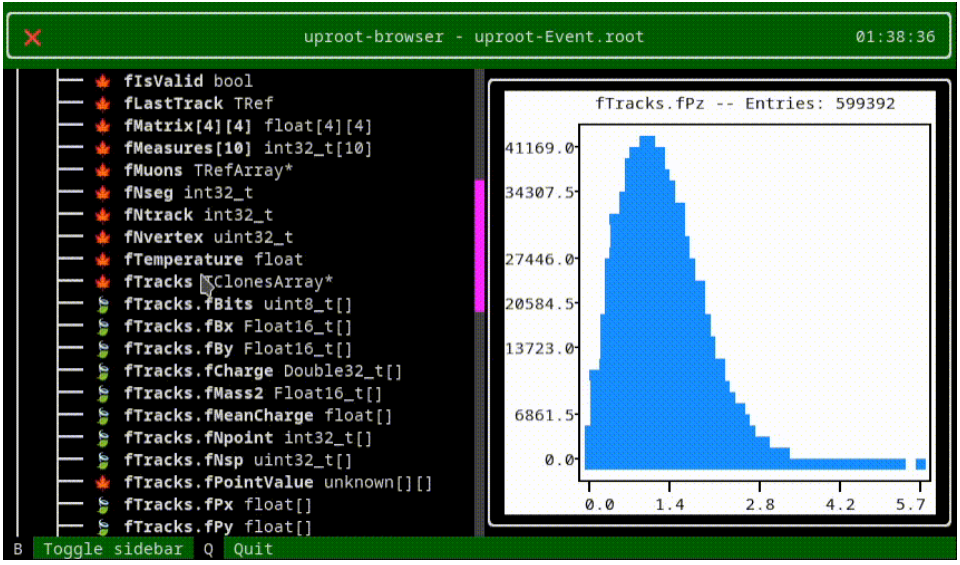
uproot-browser: another interface package that pulls many together



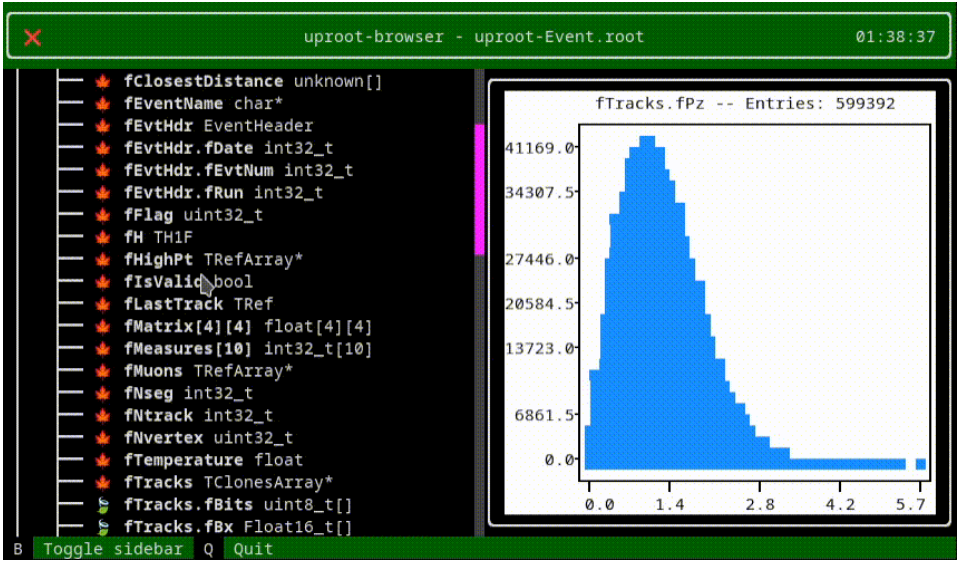
uproot-browser: another interface package that pulls many together



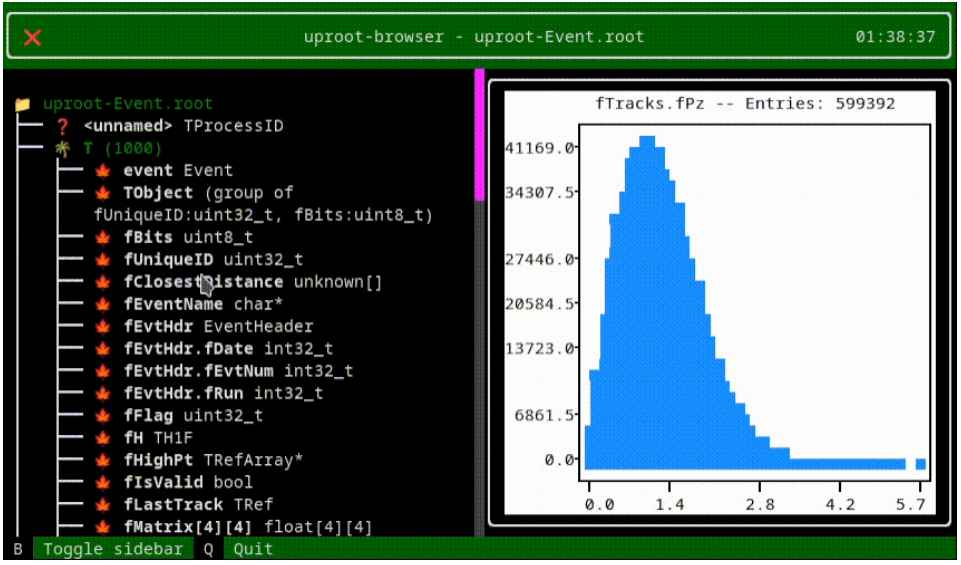
uproot-browser: another interface package that pulls many together



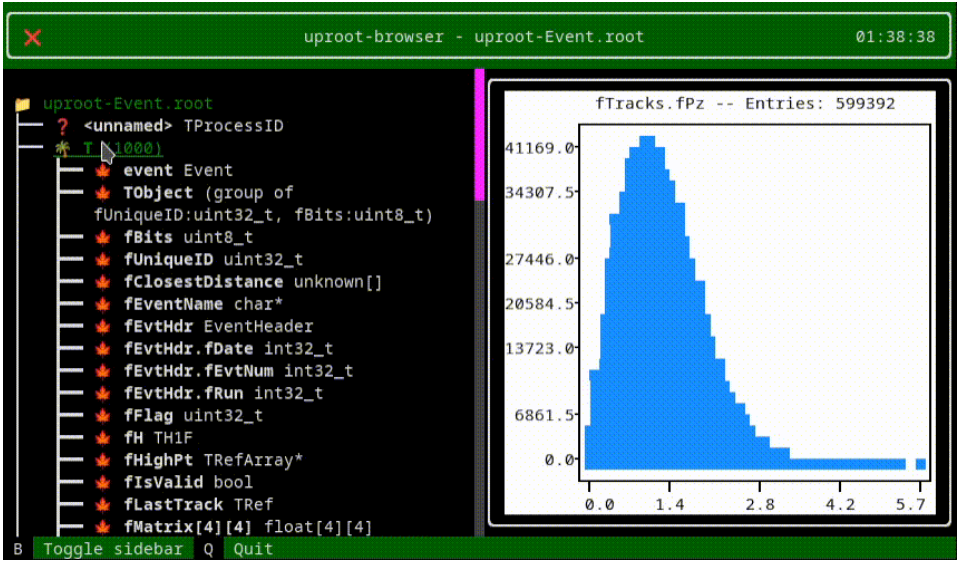
uproot-browser: another interface package that pulls many together



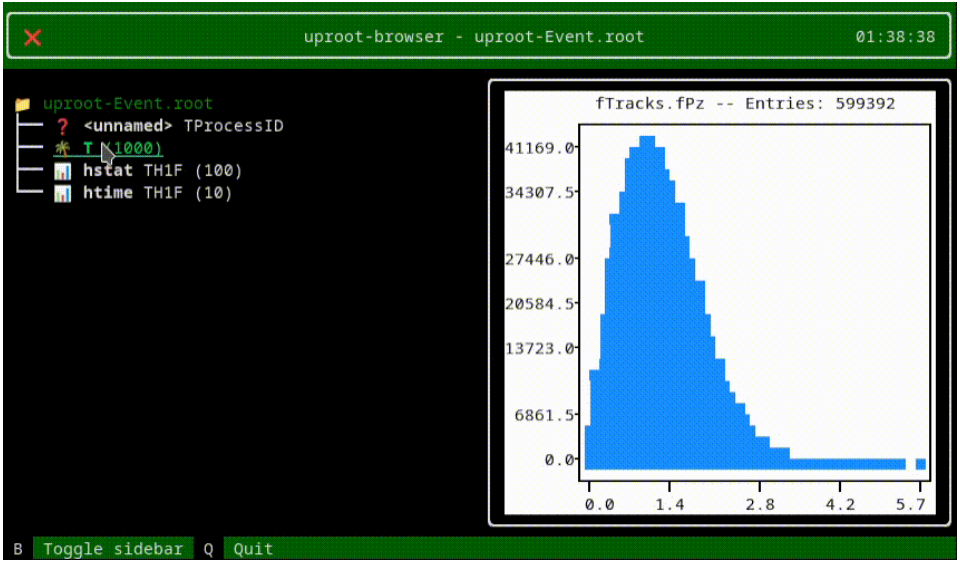
uproot-browser: another interface package that pulls many together



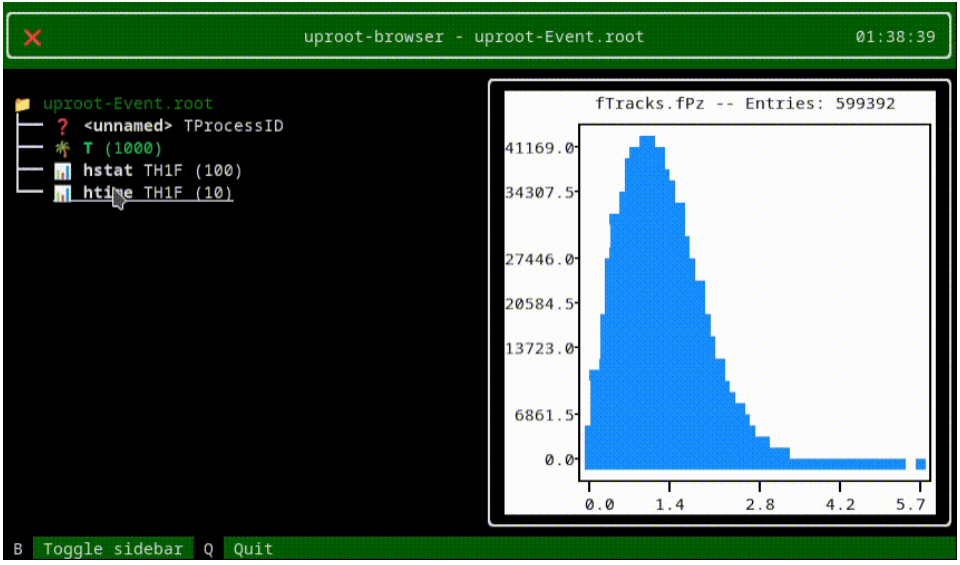
uproot-browser: another interface package that pulls many together



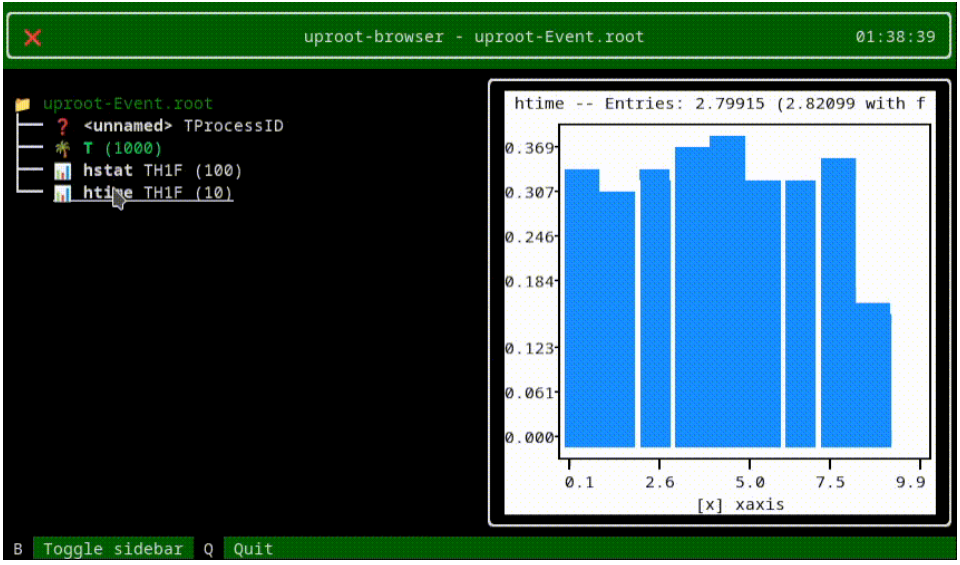
uproot-browser: another interface package that pulls many together



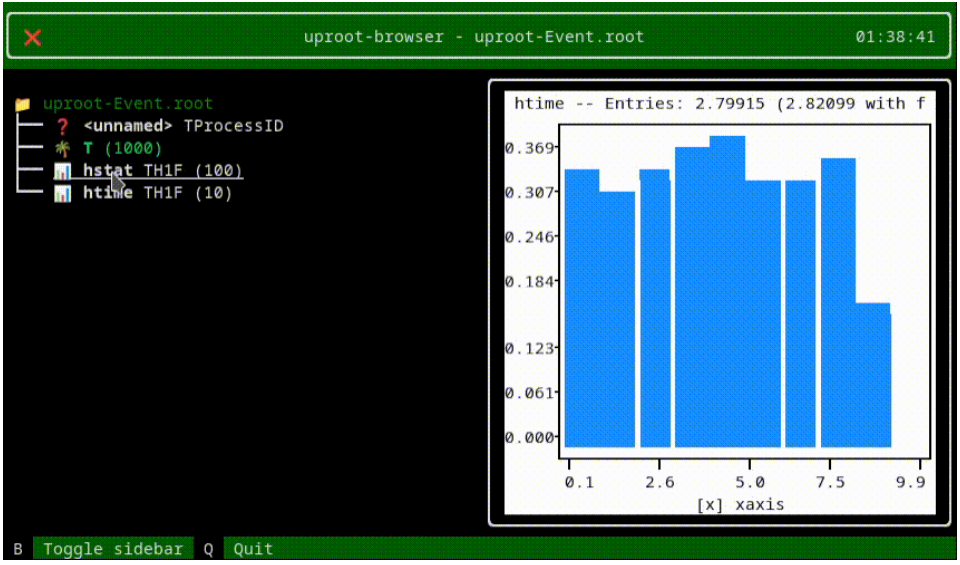
uproot-browser: another interface package that pulls many together



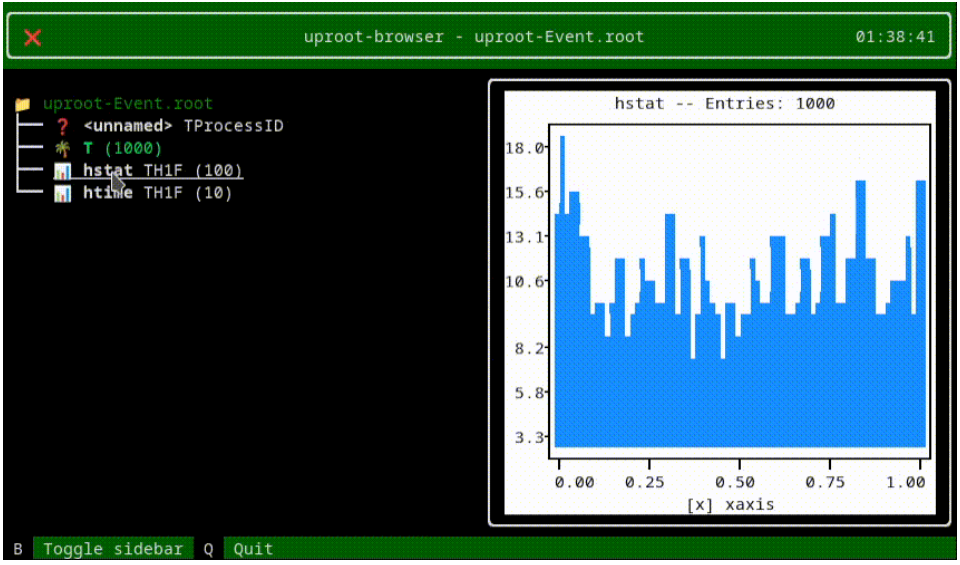
uproot-browser: another interface package that pulls many together



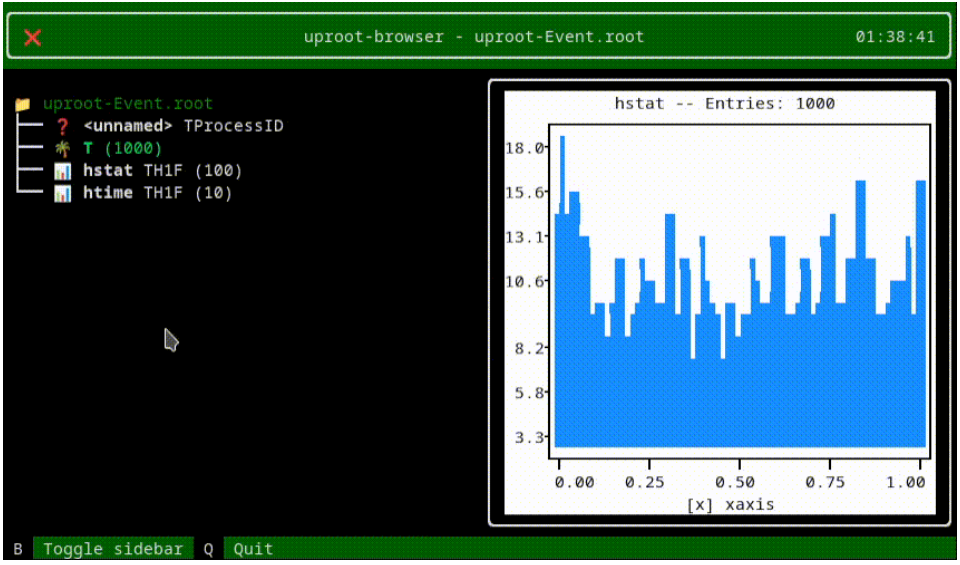
uproot-browser: another interface package that pulls many together



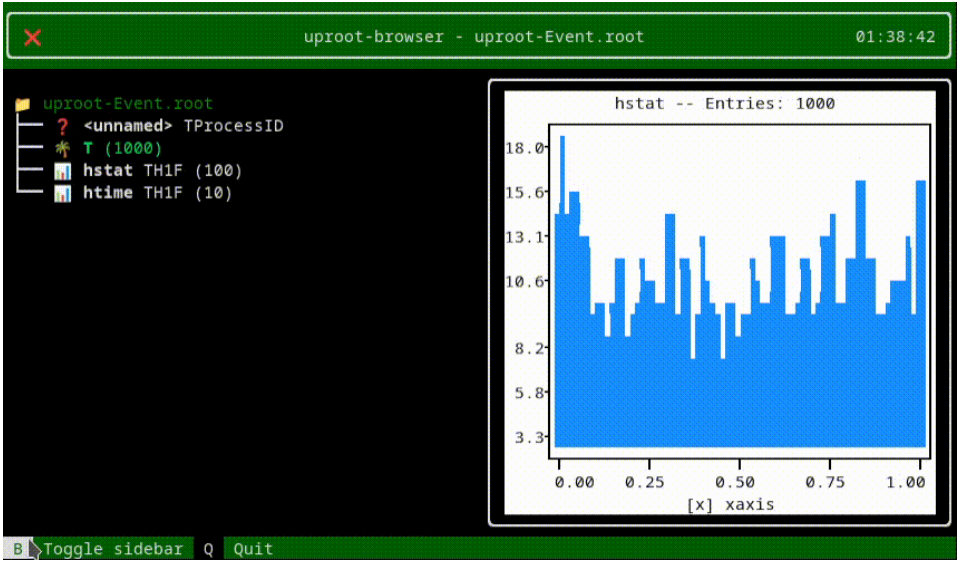
uproot-browser: another interface package that pulls many together



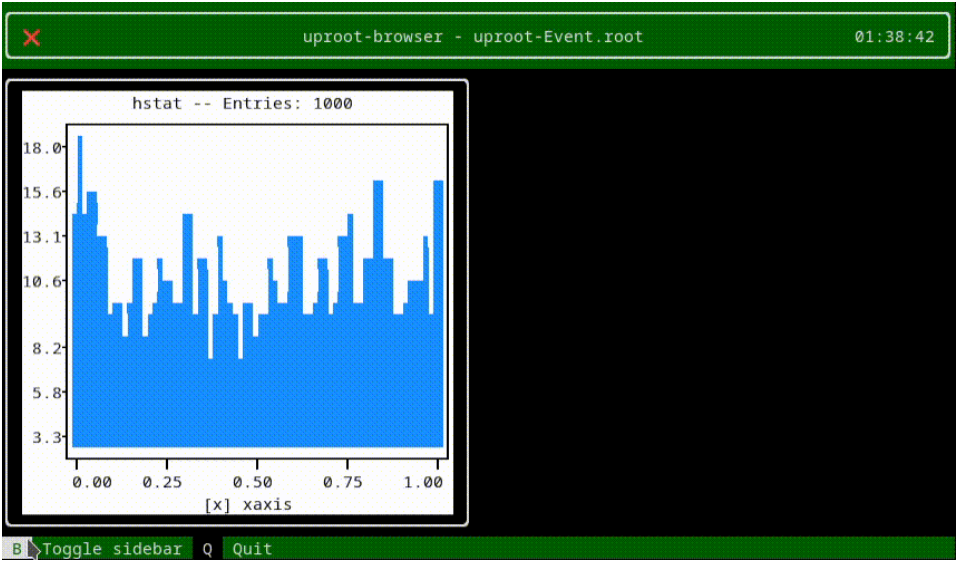
uproot-browser: another interface package that pulls many together



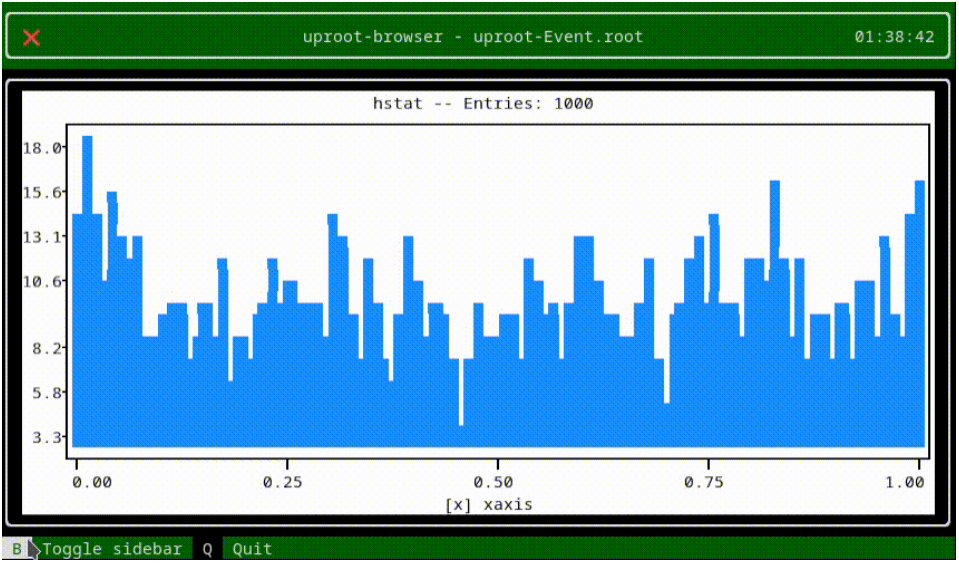
uproot-browser: another interface package that pulls many together



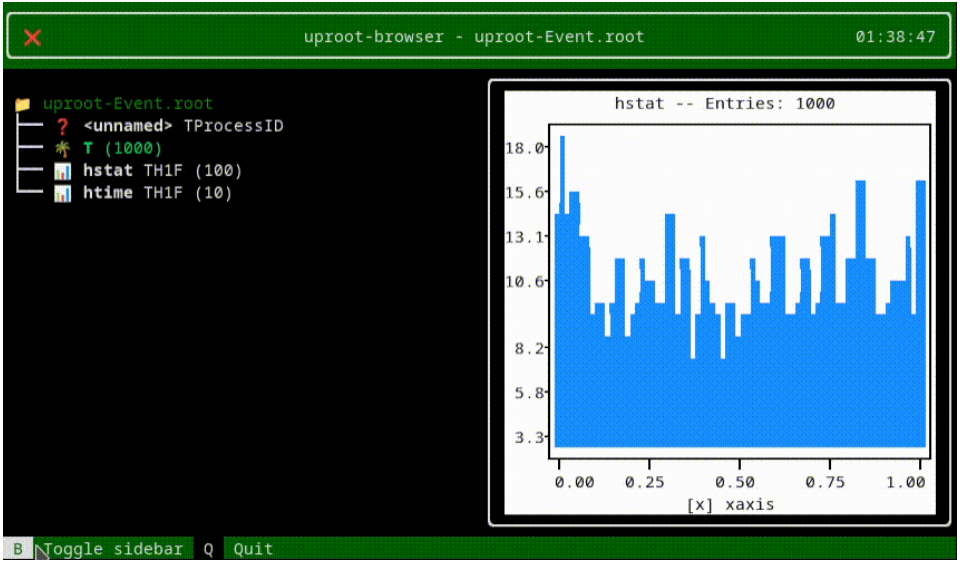
uproot-browser: another interface package that pulls many together



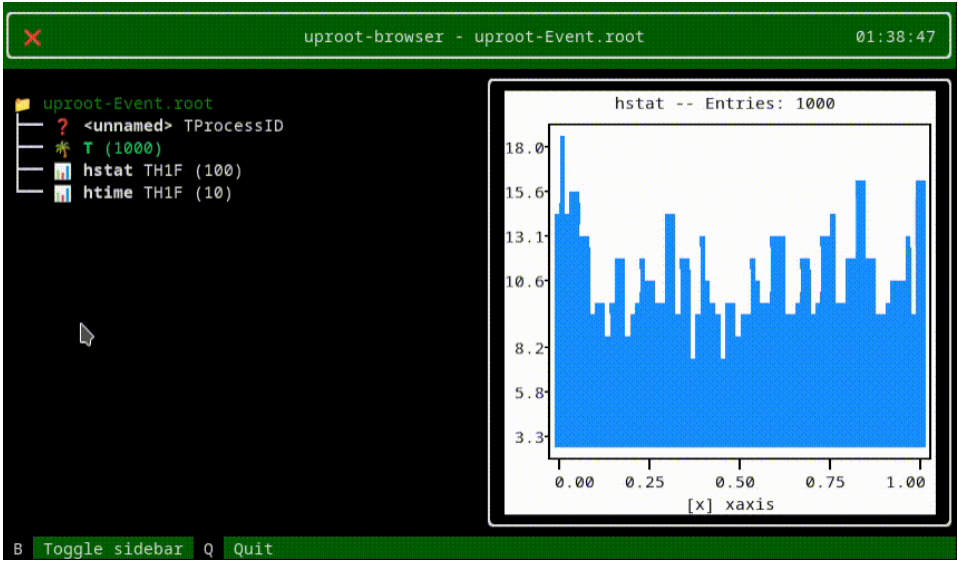
uproot-browser: another interface package that pulls many together



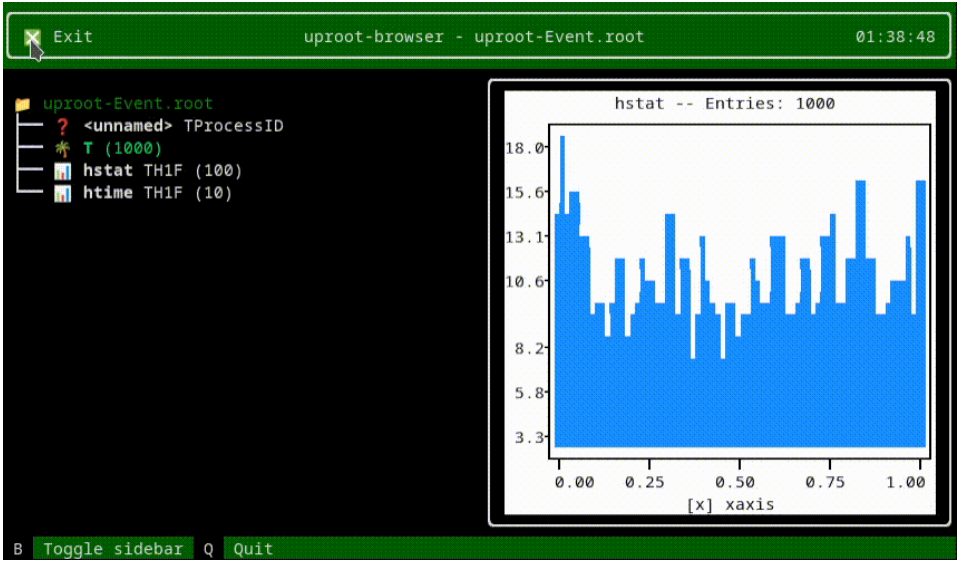
uproot-browser: another interface package that pulls many together



uproot-browser: another interface package that pulls many together



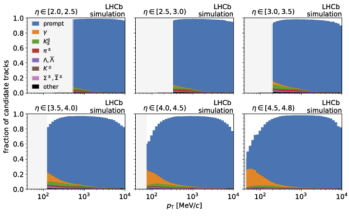
uproot-browser: another interface package that pulls many together

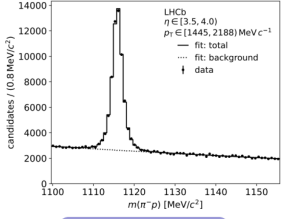


LHCb Publication Using Solely Scikit-HEP Tools

Post data-processing all performed with Python HEP tools!

- Uproot: Interfacing with input ROOT files
- boost-histogram: Replace classic TH* ROOT classes; Bonus: Multi-dimensional histograms!
- iminuit: User-friendly interface to minuit2 to process minimization
- PDF build in Python using SciPy library components





LHCb-PAPER-2021-010

My point is not that you *should*.

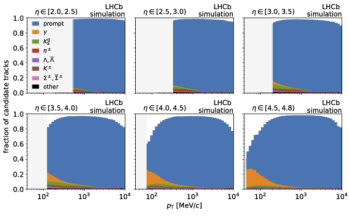
(Use any tool that gets the job done!)

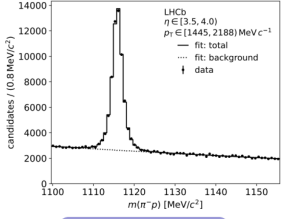
My point is that you *can*. Scikit-HEP packages cover all aspects of analysis.

LHCb Publication Using Solely Scikit-HEP Tools

Post data-processing all performed with Python HEP tools!

- Uproot: Interfacing with input ROOT files
- boost-histogram: Replace classic TH* ROOT classes; Bonus: Multi-dimensional histograms!
- iminuit: User-friendly interface to minuit2 to process minimization
- PDF build in Python using SciPy library components





LHCb-PAPER-2021-010

Analysis Grand Challenge (AGC): full-scale test of all components

The **IRIS-HEP AGC Tools 2022 Workshop** is dedicated to showcasing tools and workflows related to the so-called "Analysis Grand Challenge" (AGC) being organized by IRIS-HEP and partners. This workshop is a part of preparations for HSF Analysis Ecosystems Workshop II.

The AGC focuses on running a physics analysis at scale, including the handling of systematic uncertainties, binned statistical analysis, reinterpretation, and end-to-end optimization. The AGC makes use of new and advanced analysis tools developed by the community in the Python ecosystem and relies on the development of the required cyberinfrastructure to be executed at scale. A specific goal of the AGC is to demonstrate technologies envisioned for use at the HL-LHC.

- *Foundation libraries* ***(uproot, awkward, hist, mplhep)***
- *Queries with* ***func_adl*** *and data delivery with* ***ServiceX***
- *Columnar analysis with* ***coffea***
- *Statistical inference:* ***pyhf and cabinetry***
- *From data delivery to statistical inference:* ***ServiceX, coffea, cabinetry & pyhf***
- *Data management with* ***Skyhook***
- *Scale-out with coffea:* ***coffea-casa analysis facility***
- *Analysis user experience with Python HEP data science tools in different experiments*
- *Experiment related discussions*

The agenda will be composed of hands-on tutorials based on various tools and services developed in the Python ecosystem by and for the particle physics community, and room to discuss the current status of projects and interfaces.

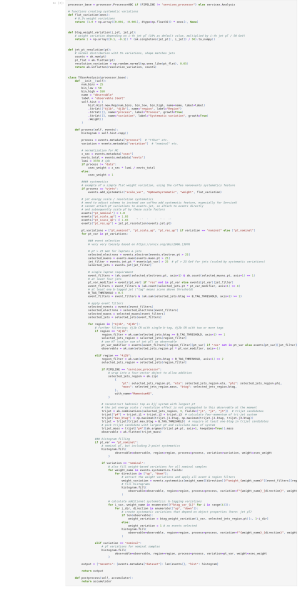
Analysis Grand Challenge (AGC): full-scale test of all components

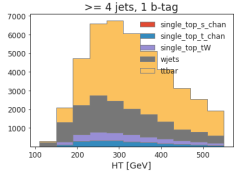
- Sample analysis on Run-2 CMS Open Data
  - open data is crucial!
  - currently, MiniAOD → NanoAOD-like
  - will switch to NanoAOD when available
  - mirrors PHYSLITE
  - thanks to the CMS DPOA team
- Datasets listed here
- Everything openly developed
  - github:iris-hep/analysis-grand-challenge
- Using coffea-casa analysis facilities at UNL, UChicago, Fermilab

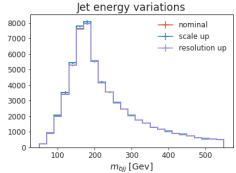
Nov 2021:	demo at first AGC workshop
Apr 2022:	second second AGC workshop
mid 2022:	benchmarking of components
spring 2023:	full scale challenge

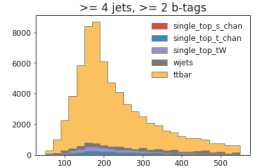
analysis-grand-challenge@iris-hep.org (sign up)

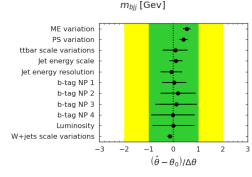
Sample $t\bar{t}$ analysis: open data delivery → inference in a notebook

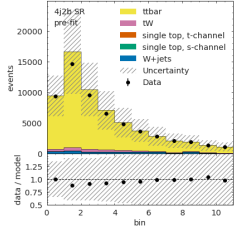


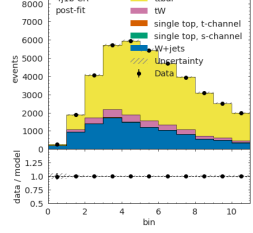


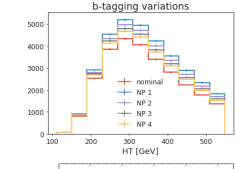


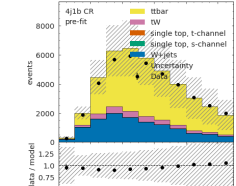


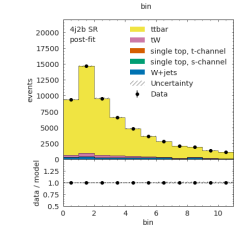






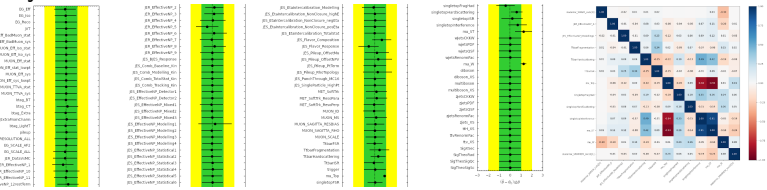




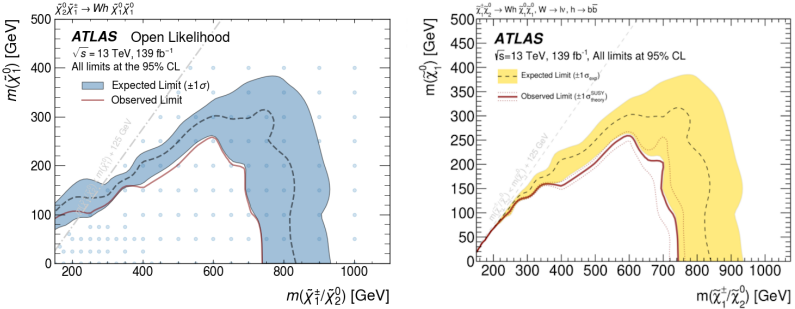


More AGC tutorials: electroweakinos from funcX to pyhf/cabinetry

Laundry list of systematics

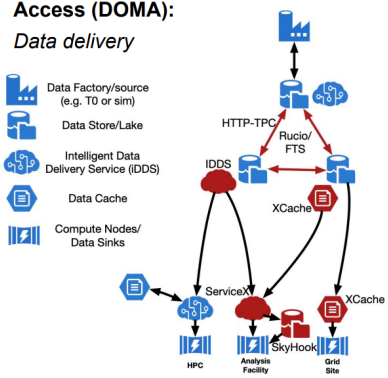


Reproduce the published plot (in a *tutorial*)

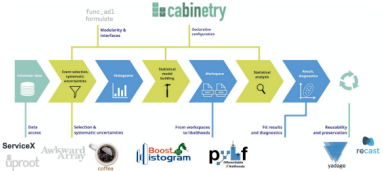


AGC tests connections between services and tools

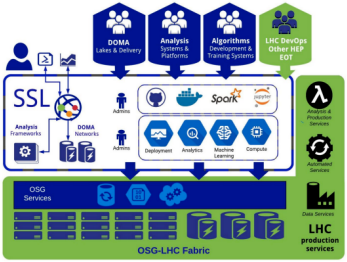
Data Organization, Management and Access (DOMA):
Data delivery



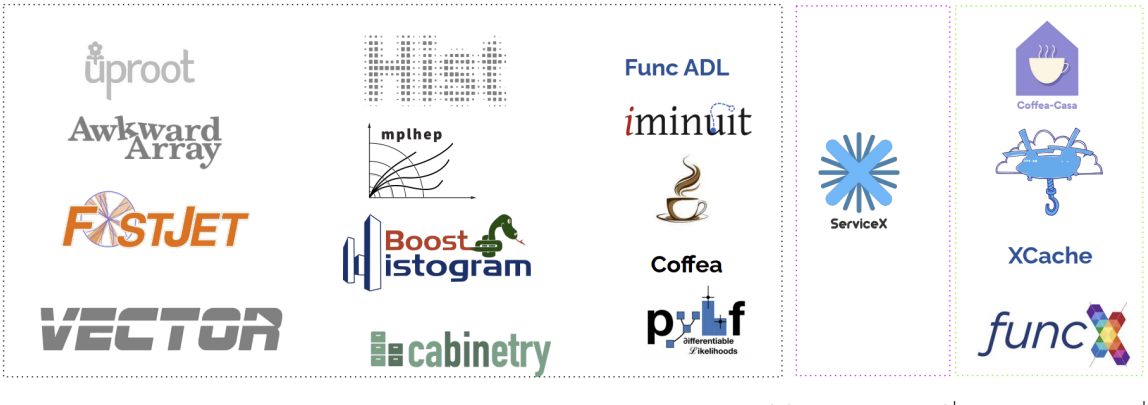
Analysis Systems (AS):
tools



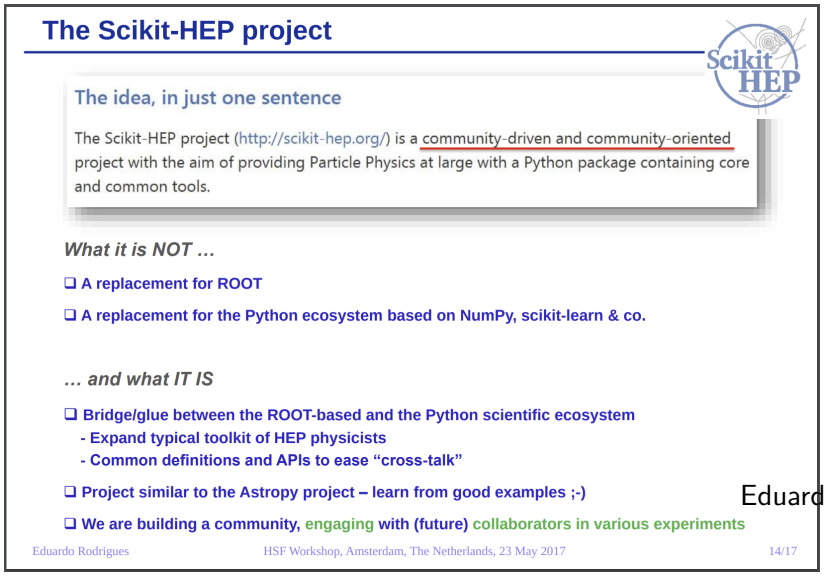
Scalable Systems Laboratory (SSL): *deployment techniques and resources*



AGC tests connections between services and tools



Conclusions: what we talked about 5 years ago is happening now



The Scikit-HEP project

The idea, in just one sentence

The Scikit-HEP project (http://scikit-hep.org/) is a community-driven and community-oriented project with the aim of providing Particle Physics at large with a Python package containing core and common tools.

What it is NOT …

- ❑ **A replacement for ROOT**
- ❑ **A replacement for the Python ecosystem based on NumPy, scikit-learn & co.**

… and what IT IS

- ❑ **Bridge/glue between the ROOT-based and the Python scientific ecosystem**
  - **Expand typical toolkit of HEP physicists**
  - **Common definitions and APIs to ease "cross-talk"**
- ❑ **Project similar to the Astropy project – learn from good examples ;-)**
- ❑ **We are building a community, engaging with (future) collaborators in various experiments**

Eduardo Rodrigues HSF Workshop, Amsterdam, The Netherlands, 23 May 2017 14/17

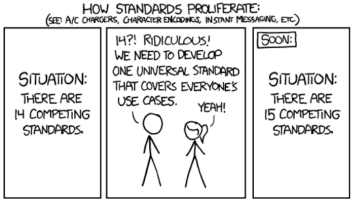
Eduardo Rodrigues

Conclusions: what we talked about 5 years ago is happening now

VISION 25

HOW TO BENEFIT FROM THE WORLD OUT THERE

- Industry has grown a LOT
- Google/Facebook/Amazon/Microsoft/Apple employ a lot of very clever people
- Doing your own bare-bones 'GPU' framework will not keep up
- Better to re-use/interop with eg. TensorFlow / SPARK (or lower level like Thrust) and focus on how to leverage those
- But what if you pick the "wrong" standard, and it dead-ends?
  - Major reason why in the past we 'did it ourselves'....
  - Contribute back (eg. to standards)



Gerhard Raven

Conclusions: what we talked about 5 years ago is happening now

2017

What we want to plot

1D Histogram Specification

```
events
    .Select(e => e.Data.eventWeight)
    .FuturePlot("event_weights", "Sample EventWeights",
        100, 0.0, 1000.0)
    .Save(hdir);
```

Save the plot in a file

today

In [4]:

```
filtered = (edm
    .Select(lambda jets: jets.Where(lambda j: j.pT > 30 and abs(j.eta) < 2.5))
)
```

Behind your back, the `uproot` transformer in ServiceX translates this into a column operation!

Gordon Watts

Representing objects as flat arrays

diana hep

For simple collections of records (e.g. particles), these arrays have the same interpretation as ROOT TLeaves:

- data arrays contain all values, ignoring event boundaries,
- size array contains the size of each event's collection.

Except for runtime calculations, rather than just on disk.

For collections of collections (with fixed, known depth), we can extend this definition recursively:

Given:	[	[	a	b	c	]	[	d	e	f	g	]	]	[	[	h	]	[	i	j	]	]
Data array:			a	b	c			d	e	f	g					h			i	j			
Recursive counter:	2	3					4							2	1			2					

embryonic Awkward Array

Conclusions: what we talked about 5 years ago is happening now