Modern Python analysis ecosystem for High Energy Physics

Jim Pivarski, Matthew Feickert, Gordon Watts

Princeton University, University of Wisconsin-Madison, University of Washington

The Python Exchange for DOE Employees
June 29th, 2022

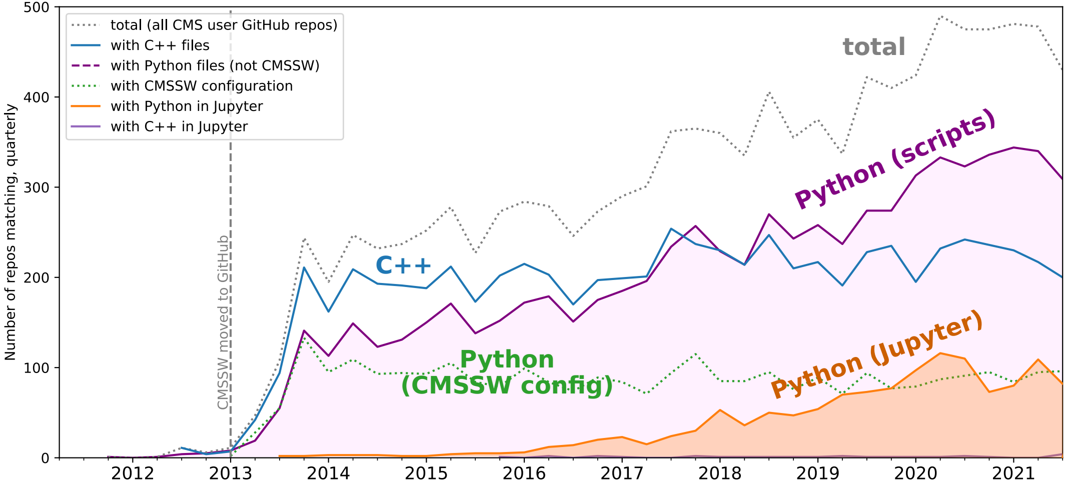
Hello from IRIS-HEP and Scikit-HEP!

- We're members of the Institute for Research and Innovation in Software for High Energy Physics (IRIS-HEP) and the Scikit-HEP community project developing a Pythonic data analysis ecosystem for HEP

- Goals: Empower analysts with modern data science stacks and provide powerful libraries for building expressive workflows

Rapid rise of Python for analysis in HEP

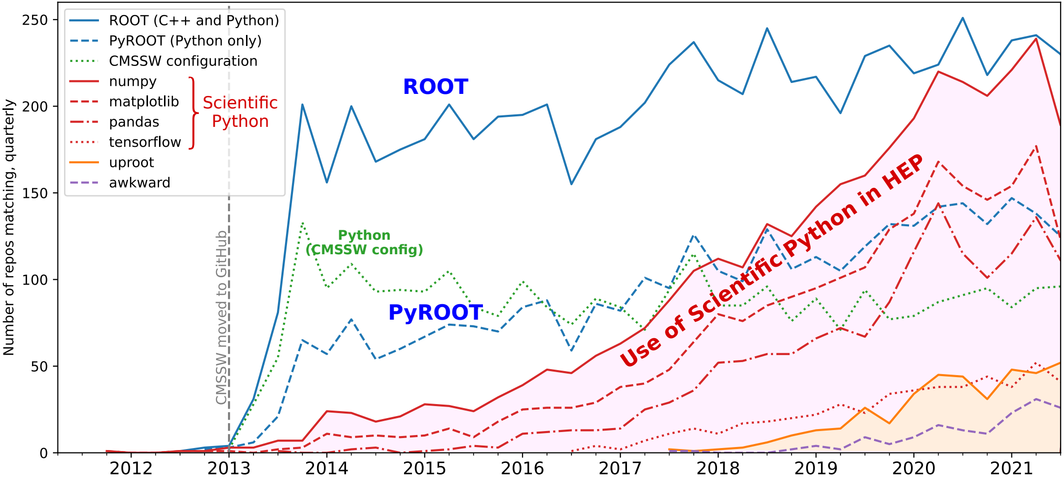
Source: "import XYZ" matches in GitHub repos for users who fork CMSSW.



(CMSSW is the CMS experiment's "offline" software framework)

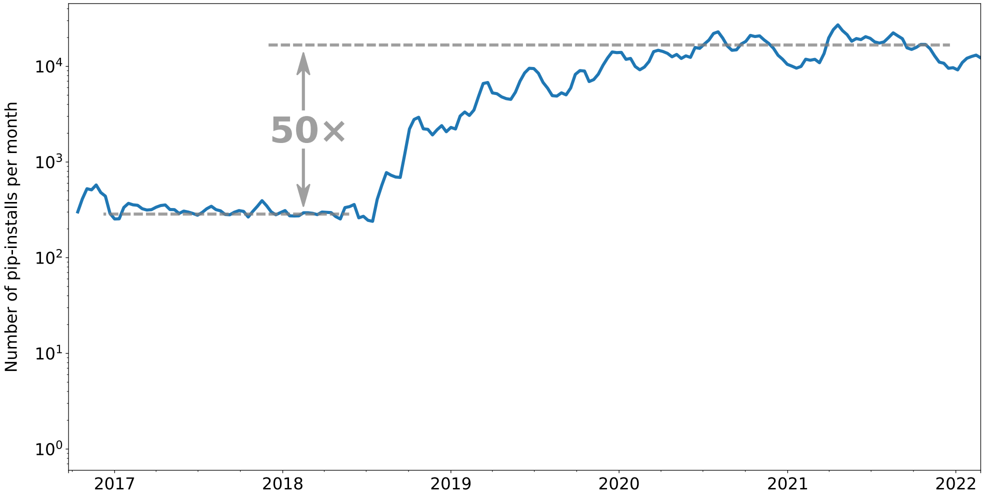
Explosion of Scientific Python (NumPy, etc.) use recent since 2018

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



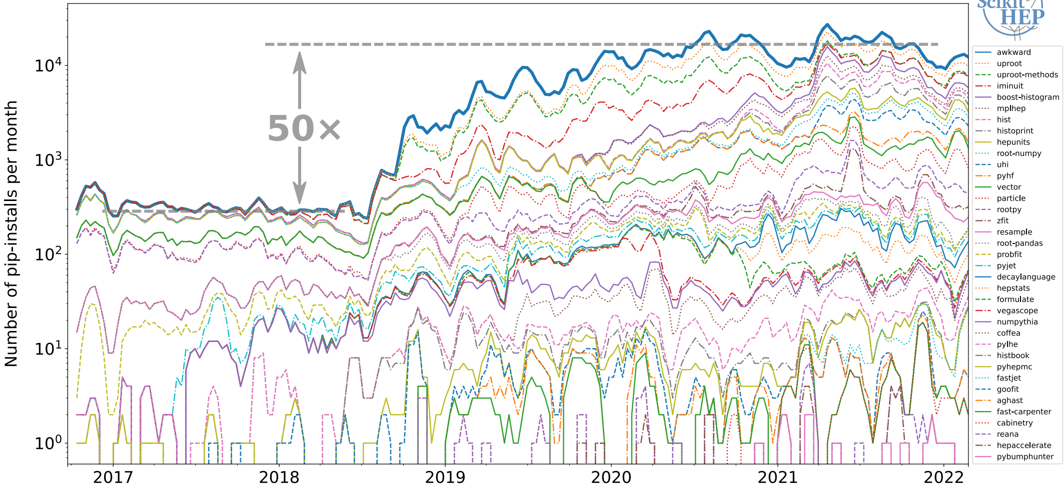
Growth tightly coupled to the rise of Scikit-HEP supported by IRIS-HEP

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



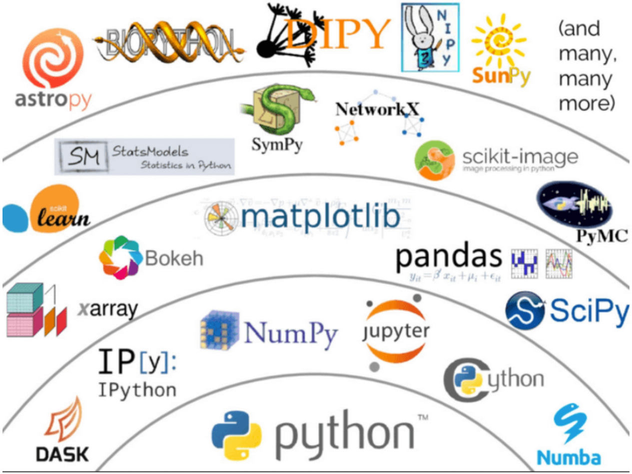
Growth tightly coupled to the rise of Scikit-HEP supported by IRIS-HEP

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



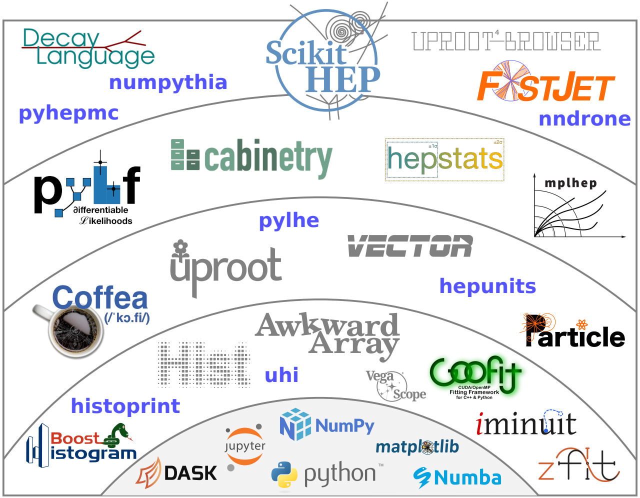
Ecosystems

In his PyCon 2017 keynote, Jake VanderPlas gave us the iconic "PyData ecosystem" image



Pythonic ecosystem for particle physics

Working view of a PyHEP ecosystem (Scikit-HEP and IRIS-HEP supported projects)



Built with intentionality and interoperability

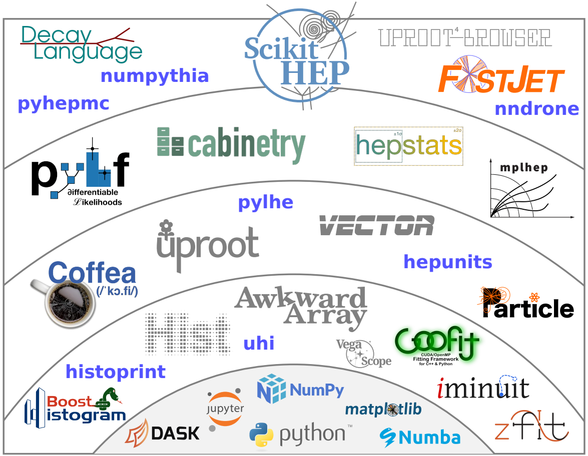
5 HEP-specific UI applications or packaged algorithms

4 HEP-specific for common problems

3 HEP-specific, foundational

2 needed to create, but not really HEP-specific

1 non-HEP software we depend on



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.

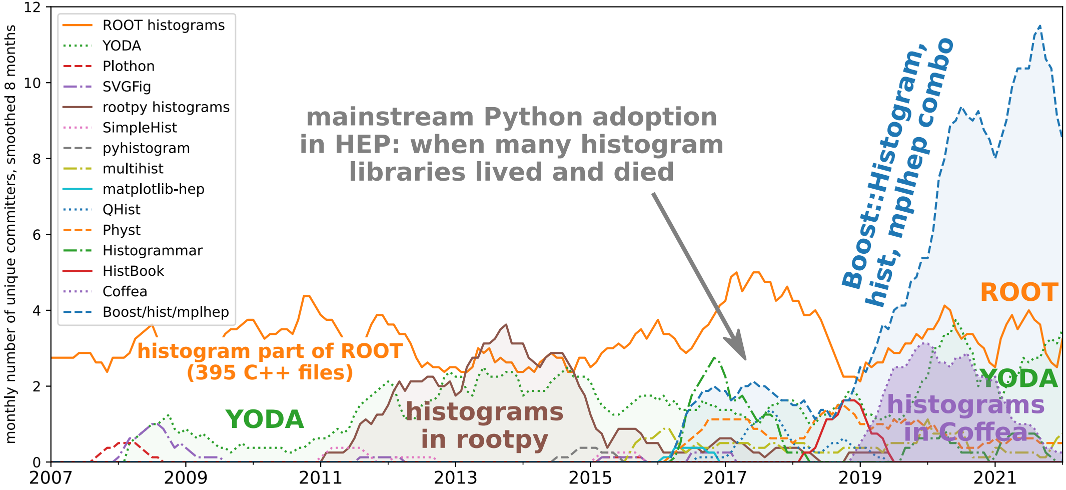
Also, it seems easy: just bin and count, right?

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

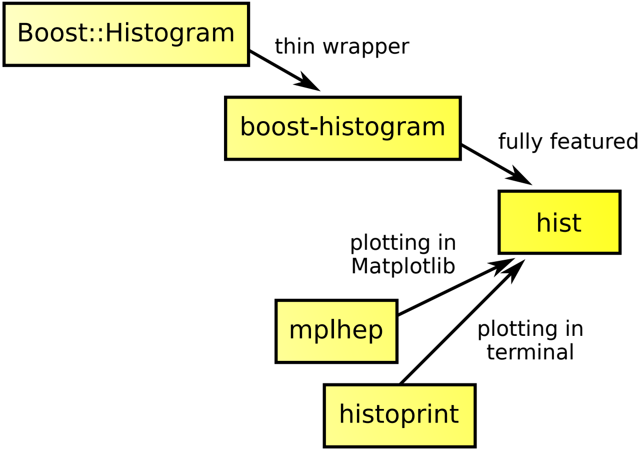
- PyROOT (2004–now)
- PAIDA (2004–2007)
- Plothon (2007–2008)
- SVGFig (2008–2009)
- YODA (2008–now)
- DANSE (2009–2011)
- rootpy (2011–2019)
- SimpleHist (2011–2015)
- pyhistogram (2015)
- multihist (2015–now)
- matplotlib-hep (2016)
- QHist (2017–2019)
- Physt (2016–now)
- Histogrammar (2016–now)
- 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).

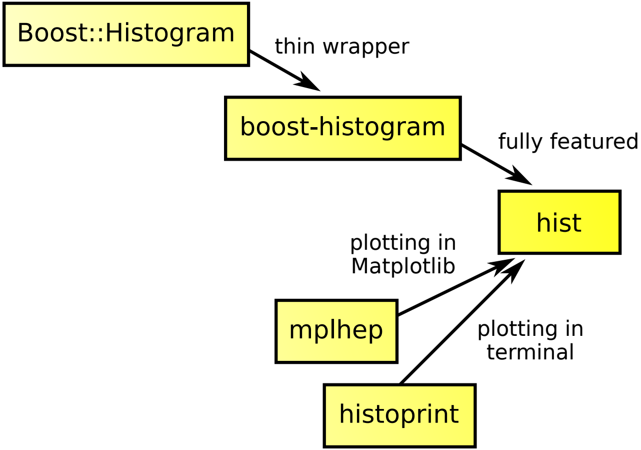


Why combine Boost::Histogram, hist, mplhep?



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

Why combine Boost::Histogram, hist, mplhep?

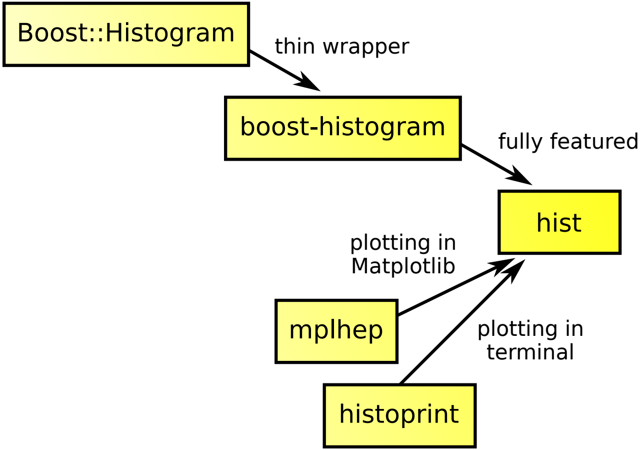


Originally, each of these was developed independently by a single author.

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

uhi 0.3.1 documentation

Search the docs ...

UHI: Unified Histogram Interface

CONTENTS:

Indexing

Indexing+

Plotting

Contents

Using the protocol:

Implementing the protocol:

Help for plotters

The full protocol version 1.2 follows:

Help for plotters

The module `uhi.numpy_plottable` has a utility to simplify the common use case of accepting a PlottableProtocol or other common formats, primarily a NumPy `histogram`/`histogram2d`/`histogramdd` tuple. The `ensure_plottable_histogram` function will take a histogram or NumPy tuple, or an object that implements `.to_numpy()` or `.numpy()` and convert it to a `NumPyPlottableHistogram`, which is a minimal implementation of the Protocol. By calling this function on your input, you can then write your plotting function knowing that you always have a `PlottableProtocol` object, greatly simplifying your code.

The full protocol version 1.2 follows:

(Also available as `uhi.typing.plottable.PlottableProtocol`, for use in tests, etc.

```
"""
Using the protocol:

Producers: use isinstance(myhist, PlottableHistogram) in your tests; part of
the protocol is checkable at runtime, though ideally you should use MyPy; if
your histogram class supports PlottableHistogram, this will pass.

Consumers: Make your functions accept the PlottableHistogram static type, and
MyPy will force you to only use items in the Protocol.
"""
```

Another need: arrays of non-tabular data

Almost all HEP data consists of variable-length lists and nested objects, which would be easiest to describe as JSON (though inefficient). To make use of NumPy-centric tools, we need a way of accessing such data in a NumPy-like way.

Another need: arrays of non-tabular data

Almost all HEP data consists of variable-length lists and nested objects, which would be easiest to describe as JSON (though inefficient). To make use of NumPy-centric tools, we need a way of accessing such data in a NumPy-like way.

```
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

Another need: arrays of non-tabular data

Almost all HEP data consists of variable-length lists and nested objects, which would be easiest to describe as JSON (though inefficient). To make use of NumPy-centric tools, we need a way of accessing such data in a NumPy-like way.

```
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

NumPy-like expression

```
output = np.square(array["y", ..., 1:])
```

```
output.to_list()
```

```
[
    [[], [4], [4, 9]],
    [],
    [[4, 9, 16], [4, 9, 16, 25]]
]
```

Another need: arrays of non-tabular data

Almost all HEP data consists of variable-length lists and nested objects, which would be easiest to describe as JSON (though inefficient). To make use of NumPy-centric tools, we need a way of accessing such data in a NumPy-like way.

```
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

NumPy-like expression

```
output = np.square(array["y", ..., 1:])

output.to_list()
 [
     [[], [4], [4, 9]],
     [],
     [[4, 9, 16], [4, 9, 16, 25]]
 ]
```

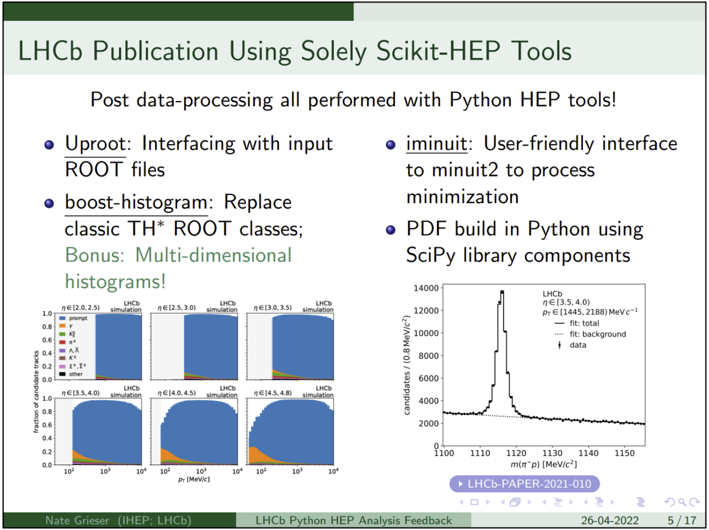
equivalent Python

```
output = []
for sublist in python_objects:
    tmp1 = []
    for record in sublist:
        tmp2 = []
        for number in record["y"][1:]:
            tmp2.append(np.square(number))
        tmp1.append(tmp2)
    output.append(tmp1)
```

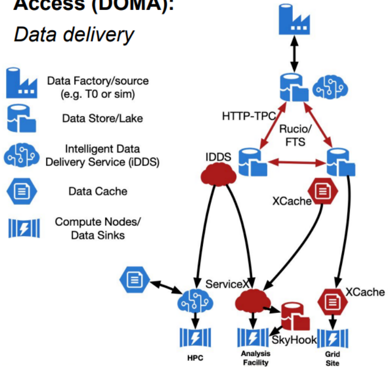
Scikit-HEP is feature-complete for modern analysis

LHCb collaboration has published *JHEP* 01 (2022) 166 using only Scikit-HEP tools for the analysis

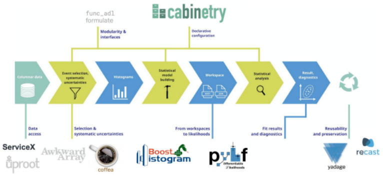
Scikit-HEP packages cover all aspects of analysis and working with IRIS-HEP to spread adoption



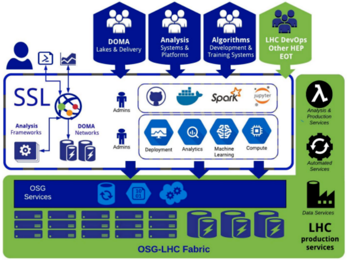
Data Organization, Management and Access (DOMA):
Data delivery



Analysis Systems (AS): *tools*



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



Optimizing Data Delivery

Requires large intermediate files

- Disk space is one of the most constrained resource for the HL-LHC's next run.
- Common formats are less inadequate as Deep Learning becomes more and more prevalent

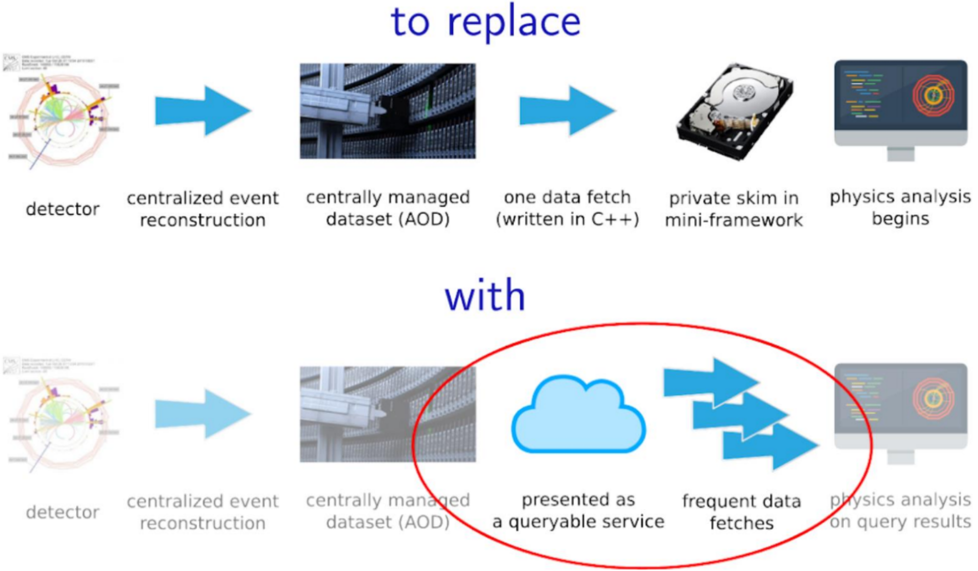
Runs can take days to weeks to complete

- Wasted analyzer time
- Inability to quickly try new ideas (time-to-insight)

Everyone writes their own version of this ("ntuplizer")

- Wasted Collaboration Effort

Optimizing Data Delivery

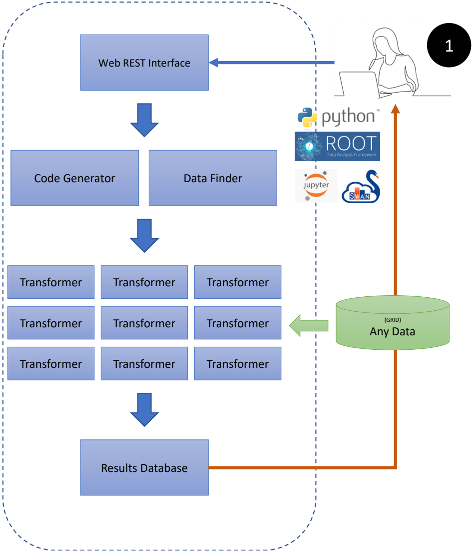


ServiceX

1

User builds a query for the system

- Original Source Data
    - $Z \rightarrow ee$ dataset
- What columns of the data should be extracted
    - Electron p_T, η, ϕ
- Filtering of the data is specified
    - $p_T > 5$ GeV, $|\eta| < 2.4$
- Derived output quantiles specified
    - Electron p_T, η, ϕ
    - Missing E_T

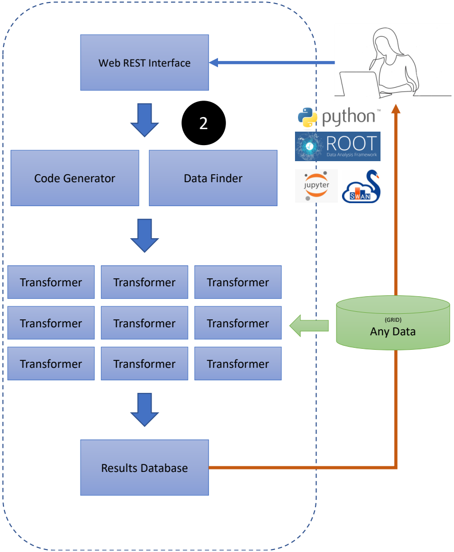


Optimizing Data Delivery

ServiceX

2 System Finds the Data & Builds Code

- Location of Original Source Data is discovered
  - URL’s for files via http
  - Files served via the rootd protocol
  - Source data can translate to many files (1000’s)
- Query language is translated into source code
  - Pyhon
  - C++
  - Anything really – depends on what consumes it in the next step



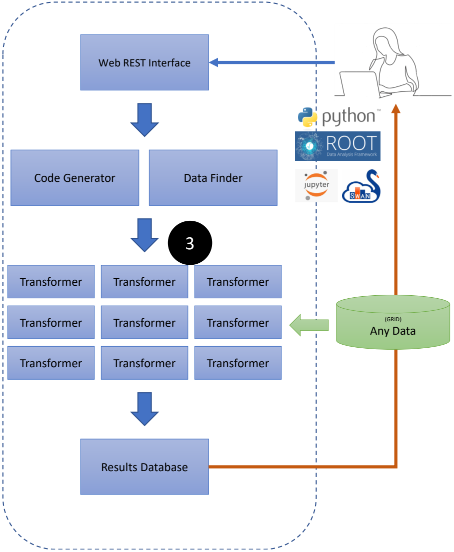
Optimizing Data Delivery

ServiceX

3

Input data is transformed into requested output data

- Up to some preset limit, a processes is spun up for every file
- As long as bandwidth is there, it run 100’s of files in parallel, reading from the list of sources the Data Finder pulled in



Optimizing Data Delivery

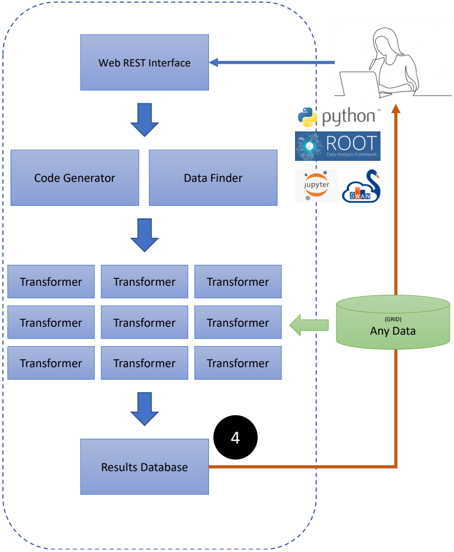
ServiceX

4

Results are cached internally in S3-like database

- S3 is an object store (AWS protocol)
- Cached locally for some amount of time – can be re-queried
- Since request is unique – can always calculate hash of it and find it in the database
  - Same user making same request repeatedly
- Data streamed to DASK or single process

PB Of data → GB of data

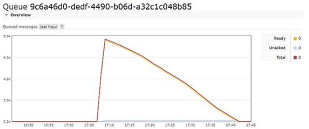


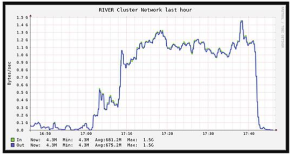
Thoughts Around This Effort

We wrote **Very Little Code** of our own! I learned some object lessons…

1. We should stop thinking about how we solve the problem. First ask – has someone else solved it?
2. Stop thinking about single-computer, single-process solutions – our data is big enough – time to think outside our knowledge base
3. It is not about what I can write – it is about the ecosystem
    - And taking advantage of everything out there

- Most efficient: collaborate with someone that knows the ecosystem and the tools
    - We stole: Kubernets, RabbitMQ, SQL, Postgress, minio, etc.



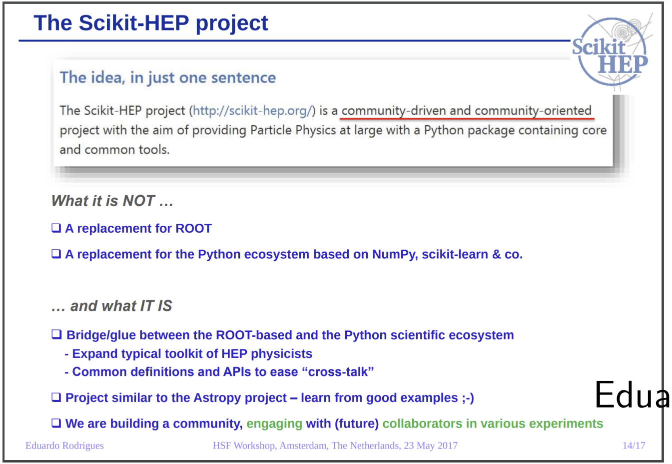


Testing on 10 TB xAOD input sample.

- Requested 100 columns from 7 collections (~30% of file)
- Scale up to 1,000 workers the River SSL Cluster
- Results in less than 30 minutes.
- Output rate was in excess of 300MB/s.

Conclusions: Modern Pythonic analysis in HEP is happening now

A confluence of scientific tools and scientists has lead to a feature-complete **Scikit-HEP** in the last 5 years



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: Modern Pythonic analysis in HEP is happening now

Growing PyHEP **ecosystem** is enabling analysts in HEP to explore and reduce the time to insight