The Scikit-HEP Project

Overview and prospects

Eduardo Rodrigues (University of Cincinnati)
Benjamin Krikler (University of Bristol)
Brian Pollack (Northwestern University)
Chris Burr (CERN)
Dmitri Smirnov (BNL)
Hans Dembinski (Max-Planck-Institute for Nuclear Physics, Heidelberg)
Henry Schreiner (Princeton University)
Jaydeep Nandi (National Institute of Technology, Silchar, India)
Jim Pivarski (Princeton University)
Matthew Feickert (University of Illinois at Urbana-Champaign)
Matthieu Marinangeli (EPFL, Lausanne)
Nick Smith (FNAL)
Pratyush Das (Institute of Engineering and Management, Kolkata, India)

+ many other package contributors

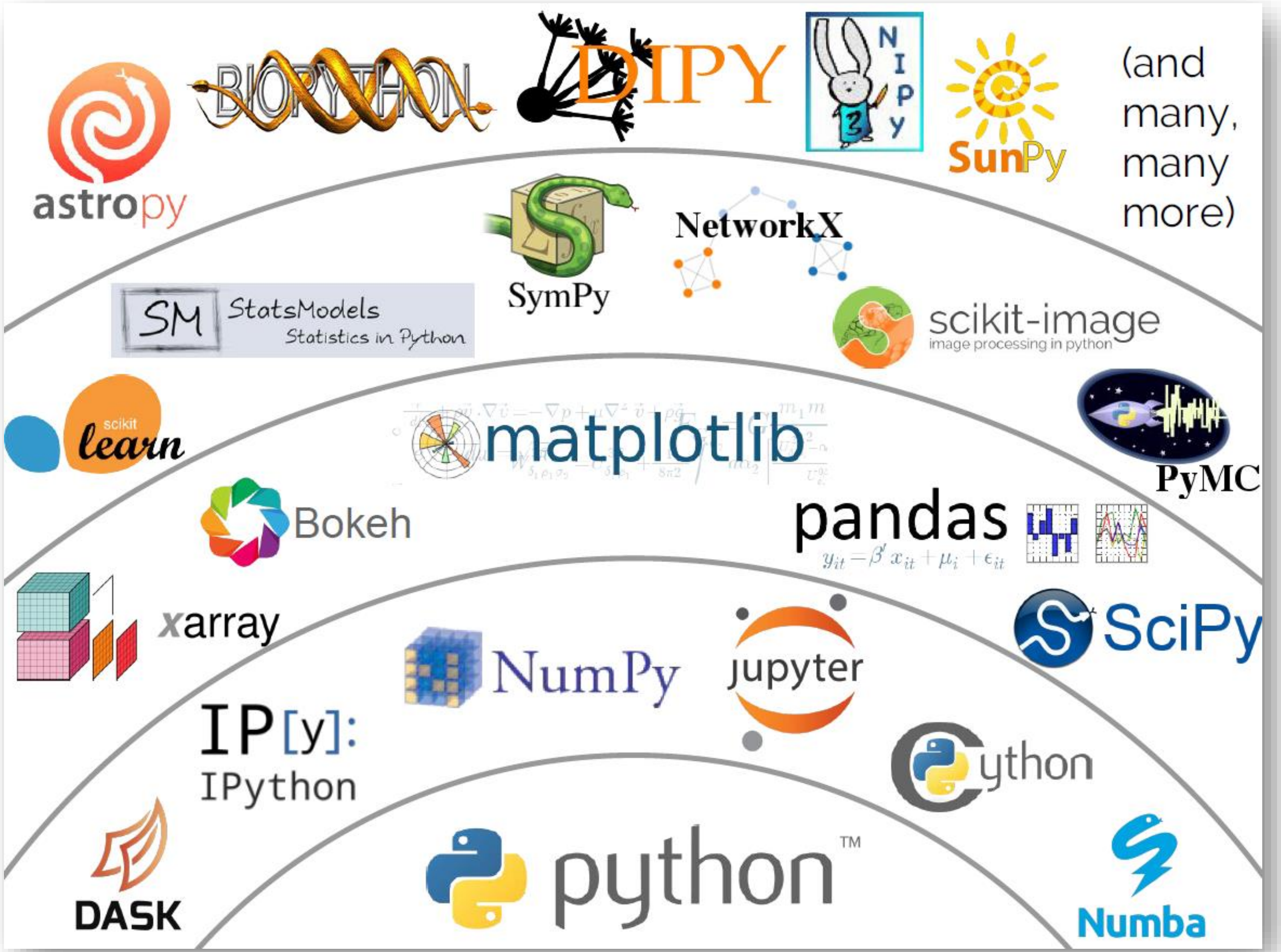
How's the Python scientific ecosystem like, outside HEP?

Domain-specific

Python's

Scientific

stack



What about HEP …?

Jake VanderPlas, *The Unexpected Effectiveness of Python in Science*, PyCon 2017

Scikit-HEP project – the grand picture

- Create an ecosystem for particle physics data analysis in Python
- Initiative to improve the interoperability between HEP tools and the scientific ecosystem in Python
  - Expand the typical ~~toolkit~~ toolset for particle physicists
  - Set common APIs and definitions to ease “cross-talk”
- Initiative to build a community of developers and users
  - Community-driven and community-oriented project
- Effort to improve discoverability of (domain-specific) relevant tools

Collaboration **Reproducibility** **Interoperability** **Sustainability**

Scikit-HEP project – 5 grand "pillars" embracing all major topics

Scikit-HEP

Scikit-HEP on GitHub

Home
Getting in touch
Documentation
Who uses Scikit-HEP?
Affiliated packages
Miscellaneous resources
FAQ
Funding
Supported Python Versions

This site uses Just the Docs, a documentation theme for Jekyll.

Scikit-HEP project - welcome!

The Scikit-HEP project is a community-driven and community-oriented project with the aim of providing Particle Physics at large with an ecosystem for data analysis in Python. The project started in Autumn 2016 and is in full swing.

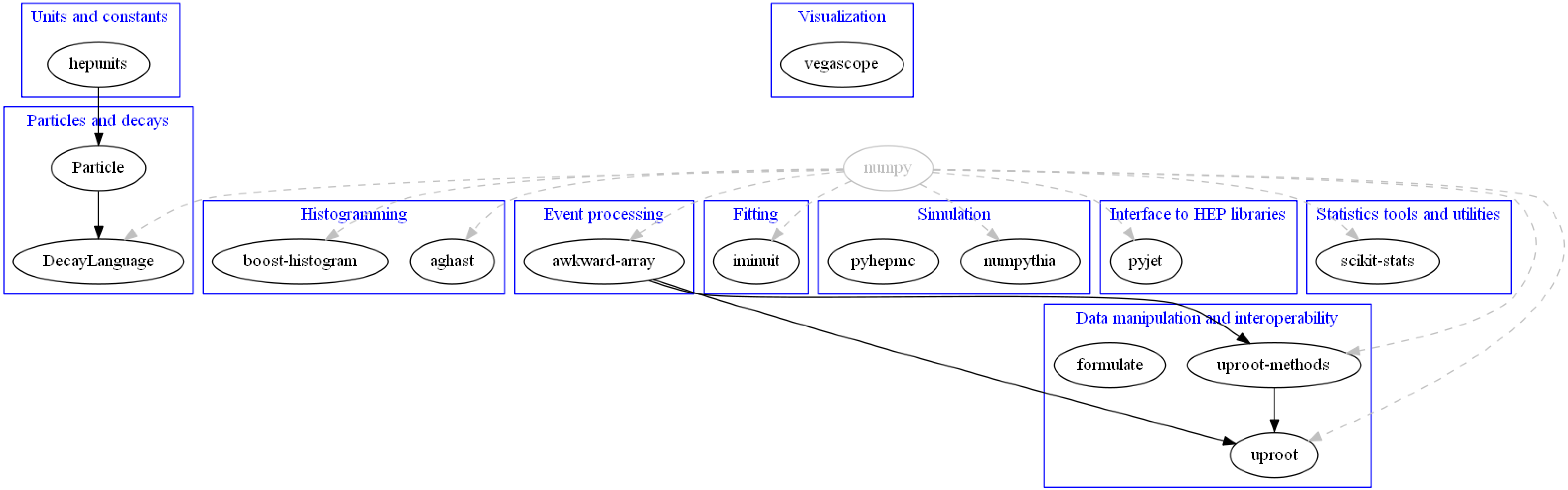
It is not just about providing core and common tools for the community. It is also about improving the interoperability between HEP tools and the scientific ecosystem in Python, and about improving on discoverability of utility packages and projects.

For what concerns the project grand structure, it should be seen as a *toolset* rather than a *toolkit*. The project defines a set of five *pillars*, which are seen to embrace all major topics involved in a physicist's work. These are:

- **Datasets**: data in various sources, such as ROOT, Numpy/Pandas, databases, wrapped in a common interface.
- **Aggregations**: e.g. histograms that summarize or project a dataset.
- **Modeling**: data models and fitting utilities.
- **Simulation**: wrappers for Monte Carlo engines and other generators of simulated data.
- **Visualization**: interface to graphics engines, from ROOT and Matplotlib to even beyond.

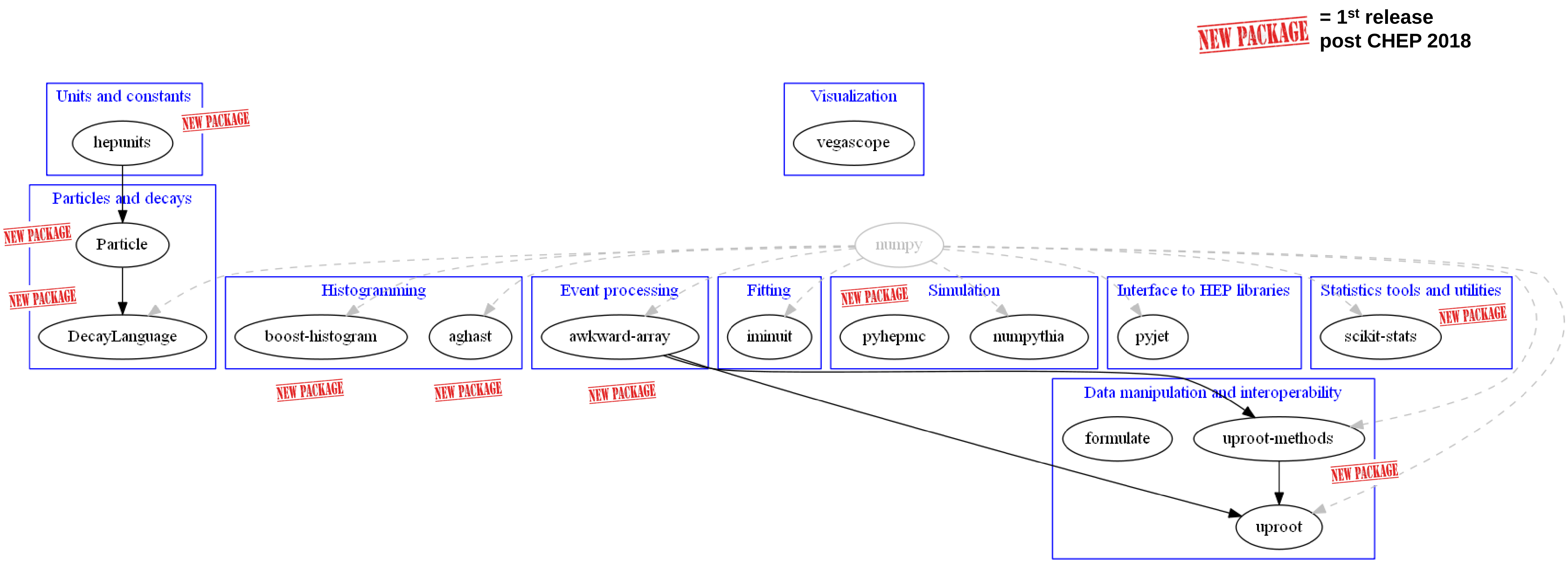
Toolset packages

Scikit-HEP project – overview of (most of the) packages



There are other packages: test data, tutorials, org stats, etc.
(and some which tend to now be superseded, hence deprecated …)

Scikit-HEP project – overview of (most of the) packages



There are other packages: test data, tutorials, org stats, etc.
(and some which tend to now be superseded, hence deprecated …)

Who uses (some of) Scikit-HEP ?

- ❑ Groups, other projects, HEP experiments
- ❑ Links are important,
  especially if they strengthen the overall ecosystem
- ❑ Community adoption going up ⇔ we're on the right path ;-)
- ❑ Rewarding to collaborate / work with / interact with
  many communities
  - Responsibility and importance of sustainability …

Software projects

Coffea - a prototype Analysis System incorporating Scikit-HEP packages to provide a lightweight, scalable, portable, and user-friendly interface for columnar analysis of HEP data. Some of the sub-packages of Coffea may become Scikit-HEP packages as development continues.

The zfit project - it provides a model fitting library based on TensorFlow and optimised for simple and direct manipulation of probability density functions.

Experiment collaborations

BelleII - the Belle II experiment at KEK, Japan.

CMS - the Compact Muon Solenoid experiment at CERN, Switzerland.

Phenomenology projects

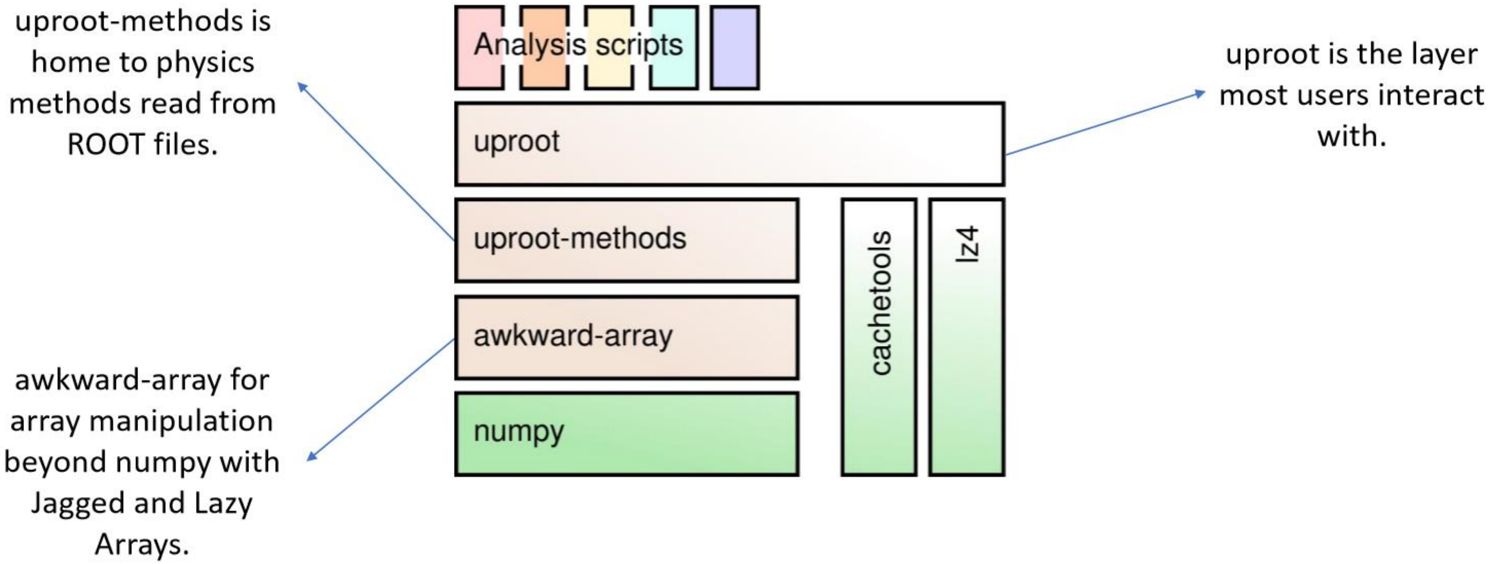
flavio - flavour physics phenomenology in the Standard Model and beyond.

Whirlwind tour of Scikit-HEP packages

➡ *Just exemplifying a sample of recent developments !*

Data manipulation and interoperability – `uproot` "suite of packages"

- (Does it still need an intro ;-)?)
- Trivially and Python-ically read ROOT files
- **Need only Numpy, no ROOT, using this pure I/O library!**
- Design and dependencies:



Minimalist ROOT I/O in pure Python and Numpy

`uproot-methods`

Pythonic mix-ins for non-I/O ROOT classes

- **Write ROOT files:** newest development, limited scope = write Ttree, histograms and a couple more classes only
  - See talk at PyHEP 2019 workshop

Event processing – `awkward-array` package

Awkward Array

- Provide a way to analyse variable-length and tree-like data in Python, by extending Numpy's idioms from flat arrays to arrays of data structures

Manipulate arrays of complex data structures as easily as Numpy

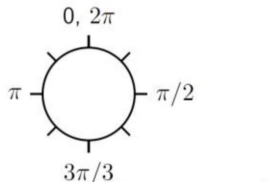
- Pure Python+Numpy library for manipulating complex data structures even if they
  - Contain variable-length lists (jagged/ragged)
  - Are deeply nested (record structure)
  - Have different data types in the same list (heterogeneous)
  - Are not contiguous in memory
  - Etc.
- This is all very relevant and important for HEP applications !

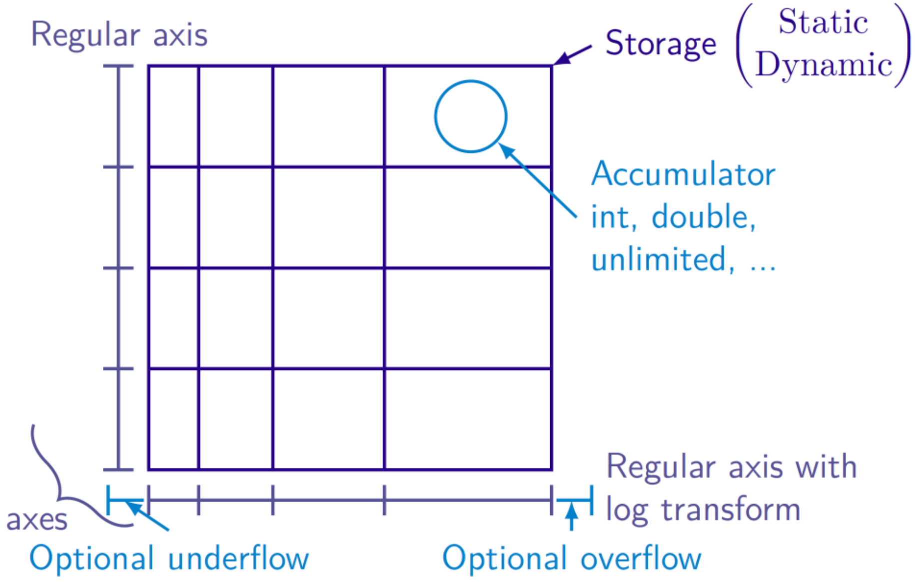
```
pip install awkward                    # maybe with sudo or --user, or in virtualenv
pip install awkward-numba              # optional: integration with and optimization by Numba
```

- Package being re-implemented in C++, with a simpler interface and less limitations
  - Major endeavour
- Work-in-progress, see https://github.com/scikit-hep/awkward-1.0 and dedicated talk ...

Histogramming – `boost-histogram` package

- Provides (pybind11) Python bindings for the C++14 Boost.Histogram library (multi-dimensional templated header-only)
  - Python(-ic) API mimics the C++ library as much as possible, aside changes for Python performance and idioms
- Development via productive exchange of features/ideas between boost-histogram and Boost.Histogram
- Binary wheels for all major platforms, supports for all Python versions; availability via conda-forge
- Alpha release, on the verge of becoming Beta
- A histogram is seen as collection of Axis objects and a storage
  - Several types available, e.g. circular axis



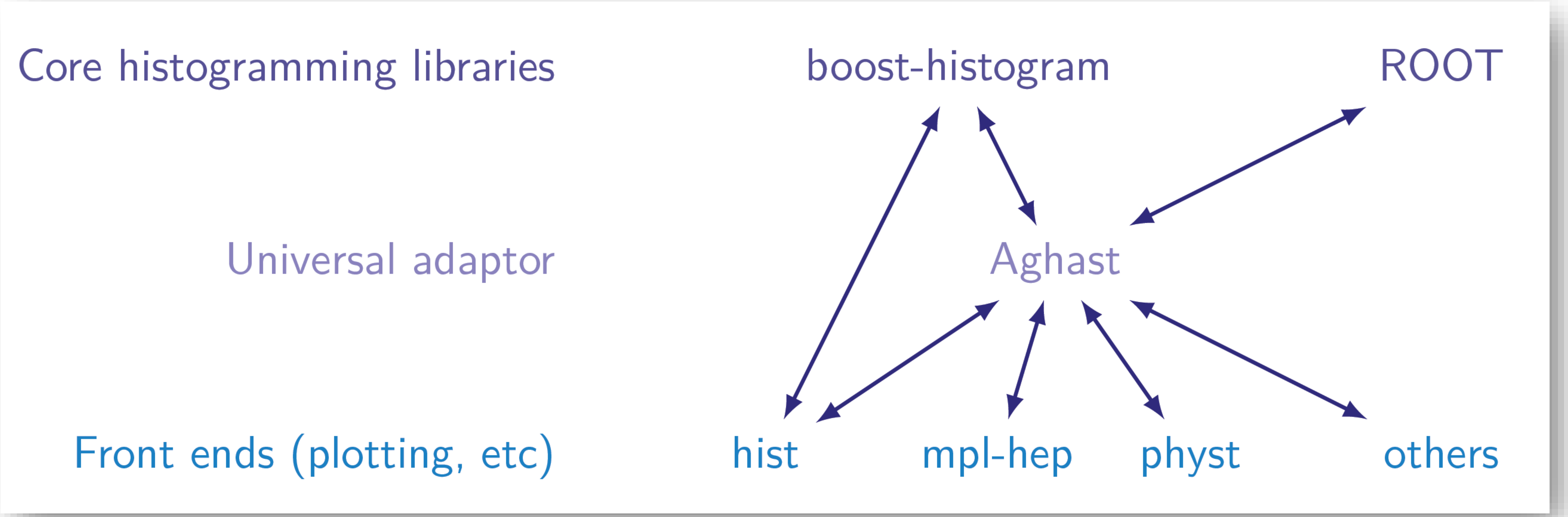


- Example usage:

```python
import boost_histogram as bh

# Compose axis however you like
hist = bh.histogram(bh.axis.regular(2, 0, 1),
                    bh.axis.regular(4, 0.0, 1.0))

# Filling can be done with arrays, one per dimension
hist.fill([.3, .5, .2],
          [.1, .4, .9])

# Numpy array view into histogram counts, no overflow bins
counts = hist.view()
```

Histogramming – looking ahead

- A fair amount of interest in the (HEP) community to develop a histogramming sub-ecosystem that meets our requirements
- Involves packages for core functionality such as filling, plotting, serialisation, and interoperability
- Interaction with popular fitting packages is also paramount



Taken from Henry Schreiner, IRIS-HEP talks

Fitting – `iminuit` package

- Provides Python interface to the MINUIT2 C++ package (built on Cython)
- Most commonly used for likelihood fits of models to data, and to get model parameter error estimates from likelihood profile analysis
- Used in many other HEP (e.g. zfit) and non-HEP (e.g. astroparticle) packages
- Binary wheels for all major platforms, supports for all Python versions; availability via conda-forge
- There is also probfit - cost function builder for fitting distributions

iminuit

Python interface to the MINUIT2 C++ package

- Example usage:

```python
from iminuit import Minuit

def f(x, y, z):
    return (x - 2) ** 2 + (y - 3) ** 2 + (z - 4) ** 2

m = Minuit(f)

m.migrad()  # run optimiser
```

FCN = 1.624E-22			Ncalls = 36 (36 total)	
EDM = 1.62E-22 (Goal: 1E-05)			up = 1.0	
Valid Min.	Valid Param.	Above EDM	Reached call limit	
True	True	False	False	
Hesse failed	Has cov.	Accurate	Pos. def.	Forced
False	True	True	True	False

	Name	Value	Hesse Error	Minos Error-	Minos Error+	Limit-	Limit+	Fixed
0	x	2.0	1.0					
1	y	3.0	1.0					
2	z	4.0	1.0					

Particles and decays – `Particle` package

Particle

- Pythonic interface to the Particle Data Group (PDG) particle data table and MC particle identification codes
- With many extra goodies
- Simple and natural APIs

```
from particle import Particle, PDGID

pid = PDGID(211)
pid
```

```
<PDGID: 211>
```

```
pid.is_meson
```

```
True
```

```
Particle.from_pdgid(415)
```

$D_2^*(2460)^+$

PDG particle data group 1957 2017

- Main classes for queries and look-ups:
  - Particle
  - `PDGID`
  - Command-line queries also available
- Powerful and flexible searches as 1-liners, e.g.

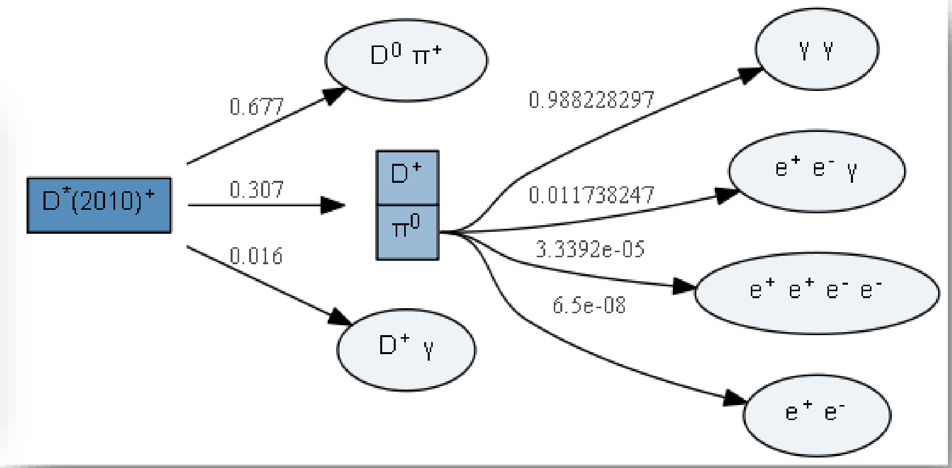
```
In [7]: from particle import Particle, SpinType

        Particle.findall(lambda p: p.pdgid.is_meson and p.pdgid.has_charm and p.spin_type==SpinType.PseudoScalar)

Out[7]: [<Particle: name="D+", pdgid=411, mass=1869.65 ± 0.05 MeV>,
         <Particle: name="D-", pdgid=-411, mass=1869.65 ± 0.05 MeV>,
         <Particle: name="D0", pdgid=421, mass=1864.83 ± 0.05 MeV>,
         <Particle: name="D~0", pdgid=-421, mass=1864.83 ± 0.05 MeV>,
         <Particle: name="D(s)+", pdgid=431, mass=1968.34 ± 0.07 MeV>,
         <Particle: name="D(s)-", pdgid=-431, mass=1968.34 ± 0.07 MeV>,
         <Particle: name="eta(c)(1S)", pdgid=441, mass=2983.9 ± 0.5 MeV>,
         <Particle: name="B(c)+", pdgid=541, mass=6274.9 ± 0.8 MeV>,
         <Particle: name="B(c)-", pdgid=-541, mass=6274.9 ± 0.8 MeV>,
         <Particle: name="eta(c)(2S)", pdgid=100441, mass=3637.6 ± 1.2 MeV>]
```

Particles and decays – DecayLanguage package

- Tools to parse decay files (aka .dec files) and programmatically manipulate them, query, display information
- Universal representation of particle decay chains
- Tools to translate decay amplitude models from AmpGen to GooFit, and manipulate them

- Parse, extract information and visualise a decay chain:

```
from decaylanguage import DecFileParser, DecayChainViewer

dfp = DecFileParser('Dst.dec')
dfp.parse()

chain = dfp.build_decay_chains('D*+', stable_particles=['D+', 'D0'])
DecayChainViewer(chain)
```



- Represent a complex decay chain:

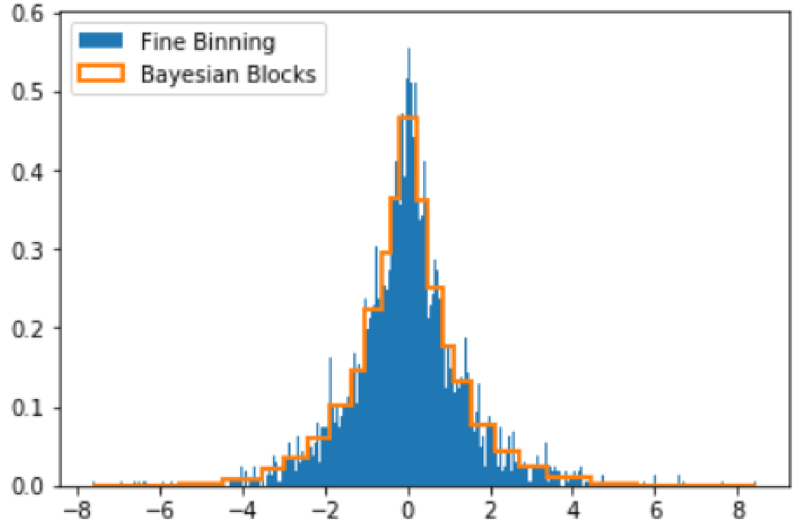
```
dm1 = DecayMode(0.0124, 'K_S0 pi0', model='PHSP')
dm2 = DecayMode(0.692, 'pi+ pi-')
dm3 = DecayMode(0.98823, 'gamma gamma')
dc = DecayChain('D0', {'D0':dm1, 'K_S0':dm2, 'pi0':dm3})
```

```
dc.print_as_tree()

D0
+--> K_S0
|    +--> pi+
|    +--> pi-
+--> pi0
     +--> gamma
     +--> gamma
```

Statistics tools and utilities – `scikit-stats` package

- A (very) recent package
- Being actively developed in collaboration with authors of fitting frameworks, for example, to make sure the needs are covered
  - E.g., zfit (see dedicated talk)
- Plans among IRIS-HEP colleagues to improve/enhance interoperability of statistics tools (e.g. pyhf – see dedicated poster) and fitting frameworks (e.g. RooFit, GooFit, zfit)
  - Common APIs, conversions to enable inter-exchange of models
- Requires community discussion, which is starting at https://gitter.im/HSF/PyHEP-fitting

```python
import numpy as np
import matplotlib.pyplot as plt

from skstats.modeling import bayesian_blocks

data = np.random.laplace(size=10000)
bblocks = bayesian_blocks(data)

plt.hist(data, bins=1000, label='Fine Binning', density=True)
plt.hist(data, bins=bblocks, label='Bayesian Blocks', histtype='step', density=True, linewidth=2)
plt.legend(loc=2);
```



Simulation – `pyhepmc` packages

pyhepmc

Python wrapper for the HepMC3 C++ library

- HepMC3: a new rewrite of the C++ HepMC event record for MC generators

- pyhepmc: Python wrapper for the HepMC3 C++ library
- Bindings built on pybind11
- Supports all Python versions
- On PyPI as source distribution
- Beta release version 0.4.3

- Development done with exchanges with the HepMC3 team
  - Idea is to provide pyhepmc as the official bindings, included in the HepMC3 distribution

Visualisation – VegaScope package

- ❑ **Minimal viewer of Vega & Vega-Lite graphics on the browser from local or remote Python processes**
  - Vega = declarative “visualisation grammar”, see [GitHub org]
  - The Python process generating the graphics does not need to be on the same machine as the web browser viewing them
- ❑ **0 dependencies - can be installed as single file, used as a Python library or as a shell command, watching a file or stdin**
- ❑ **Example:**

```
import vegascope
canvas = vegascope.LocalCanvas()
canvas("https://vega.github.io/vega/examples/stacked-bar-chart.vg.json")
```

- ❑ **`Altair` can use `VegaScope` as a renderer:**

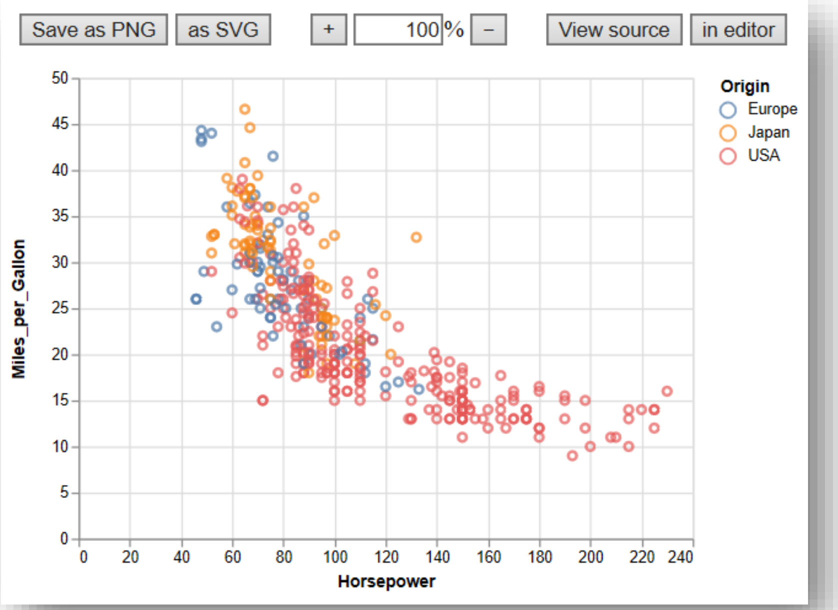
```
import vegascope
canvas = vegascope.LocalCanvas()
canvas("https://vega.github.io/vega/examples/stacked-bar-chart.vg.json")
```

```
import altair as alt
alt.renderers.enable('vegascope')
```

```
RendererRegistry.enable('vegascope')
```

```
from vega_datasets import data
cars = data.cars()
alt.Chart(cars).mark_point().encode(x='Horsepower',
                                    y='Miles_per_Gallon',
                                    color='Origin'
                                   ).interactive()
```

```
Rendered at http://localhost:56574
```

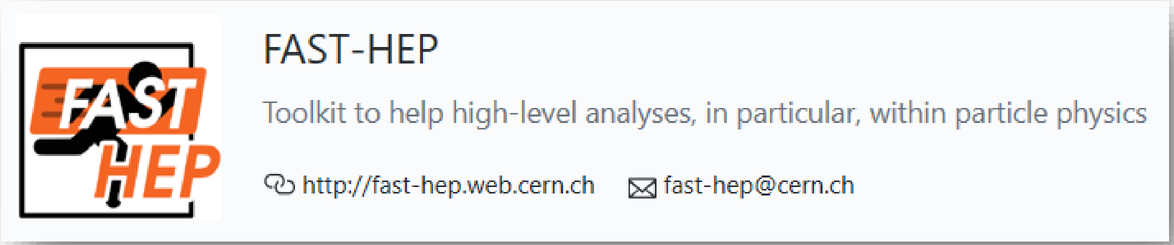


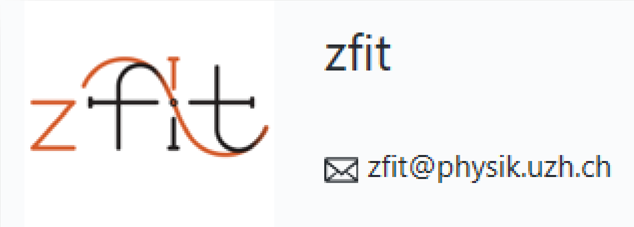
Affiliated projects / packages

Affiliated projects and packages

- As said, key project goal is the creation of an ecosystem for data analysis in Python, which is community-driven and community-oriented
- We are not alone in this endeavour - great !
- Useful concept of affiliated projects/packages:
  - They extend the ecosystem and remain, due to their size and scope, generally independent of Scikit-HEP
  - They work closely together / collaborate with Scikit-HEP
- Overall benefit is obvious

- Projects affiliated:



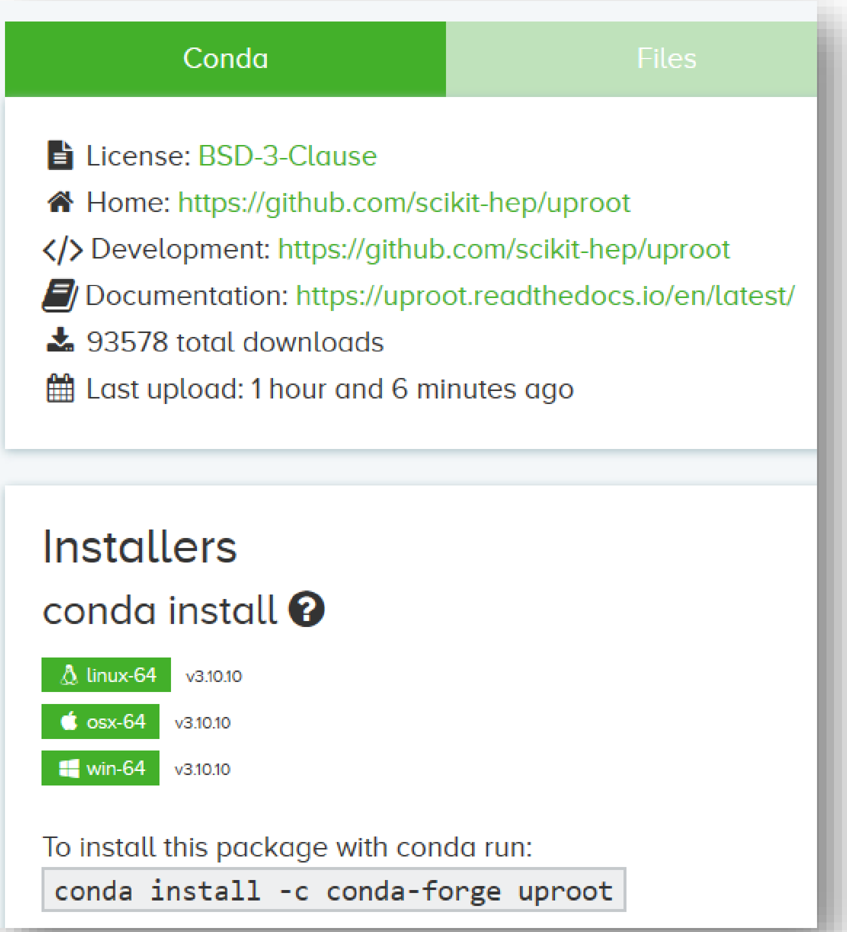


- Just about to join the org- fresh news:

Outlook

Making it easy for users

- ❑ Easy / trivial installation in many environments is a must !
- ❑ Much work has been done this last year to provide binary “wheels” on PyPI, and conda-forge packages
  - See next slide …



- ❑ Python 2 support still a need for many HEP users
- ❑ We provide support as much as feasible / realistically possible
- ❑ But keep in mind Python 2 end of life January 1st, 2020 ;-)
  - Python 2 releases will become locked after a final major release
- ❑ See here for details on our Python support statement

- ❑ Work in progress on a project metapackage …

Making it easy for users – packages on conda

- Org people (Chris, Henry) have been drivers in getting many packages on conda … including ROOT !

A metapackage for Scikit-HEP – `scikit-hep` package

- ❑ The `scikit-hep` package has historically contained a variety of things:
  - Kinematics and geometry classes for HEP
  - Modelling module
  - Visualisation utilities
  - Etc.
- ❑ The project has evolved and a different route has emerged as more adequate …

scikit-hep

A metapackage
(WIP, future)

- ❑ Vision for the future: have the scikit-hep package become a metapackage for the Scikit-HEP project
- ❑ Benefit especially for stacks for experiments: `scikit-hep` tags defining compatible releases of the whole toolset
  - Clear what "scikit-hep version 1.0.0" is
  - Stable stacks installable in a simple way
  - Having a well-defined stack also helps in analysis preservation matters, widely discussed at present
- ❑ This is (still) work-in-progress …
- ❑ "vector": example of future package taken out, which will provide awkward-/numpy-array based vector classes, and more

More information in dedicated / related talks

- **Chris Burr –** ***Sustainable software packaging for end users with conda*** **(Tue, 14h45, track 5)**
- **Henry Schreiner –** ***Recent developments in histogram libraries*** **(Thu, 11h15, track 5)**
- **Jim Pivarski –** ***Vectorized, imperative, and declarative processing of Awkward Arrays*** **(Thu, 15h00, track 5)**

On affiliated projects

- **Ben Krikler –** ***The F.A.S.T. toolset: Using YAML to make tables out of trees*** **(Mon, 11h45, track 6)**
- **Jonas Eschle –** ***zfit: scalable pythonic fitting*** **(Mon, 11h00, track 6)**
- **Matthew Feickert –**
  pyhf: a pure Python implementation of HistFactory with tensors and autograd **(poster, Tue, 15h30, track 6)**
  Likelihood preservation and statistical reproduction of searches for new physics **(Thu, 12h00, track 6)**

Interested ? Want to try it ? And contribute ?

- **We are a *growing* community ⇒ everybody welcome !**
  - **Particularly interesting to have a good sampling from the various experiments**
- **A lot to be done, still … and we need feedback too !**

Links

- **GitHub: https://github.com/scikit-hep/**
- **Website: http://scikit-hep.org/**

Get in touch

- **Gitter channel: https://gitter.im/Scikit-HEP/community**
- **Forum for anyone: scikit-hep-forum@googlegroups.com**
- **Get in touch with the team "privately": scikit-hep-admins@googlegroups.com**

Thank you

Simulation & jet clustering – `numpythia` and `pyjet` packages

- Generate events with Pythia and pipe them into NumPy arrays

```python
from numpythia import Pythia, hepmc_write, hepmc_read
from numpythia import STATUS, HAS_END_VERTEX, ABS_PDG_ID

params = {"Beams:eCM": 13000, "WeakSingleBoson:ffbar2gmZ": "on",
          "23:onMode": "off" ,"23:onIfAny": "13", "WeakZ0:gmZmode": 2}

pythia = Pythia(params=params)
selection = ((STATUS == 1) & ~HAS_END_VERTEX)

for event in pythia(events=100):
    array = event.all(selection)
    muplus  =  array[array["pdgid"] == 13]
```

numpythia

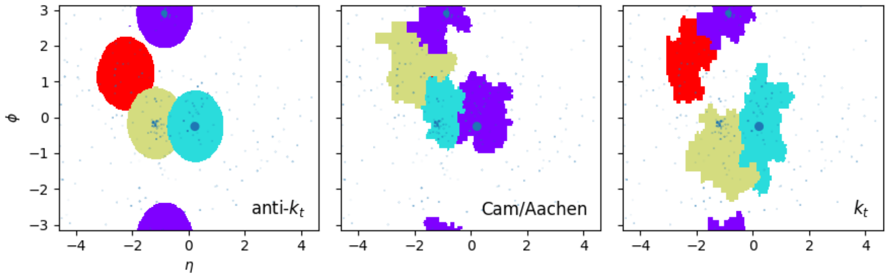
Interface between PYTHIA and NumPy

- Possible to feed those events into FastJet using `pyjet`

```python
from pyjet import cluster
from pyjet.testdata import get_event

vectors = get_event()
sequence = cluster(vectors, R=1.0, p=-1)
jets = sequence.inclusive_jets()  # list of PseudoJets
```

pyjet

Interface between FastJet and NumPy



Units and constants in the HEP system of units – `hepunits` package

- **Units and constants in the HEP system of units**
  - Not the same as the SI system of units
- **Trivial package, but handy**
- **Typical usage:**

```
from hepunits.constants import c_light
from hepunits.units     import picosecond, micrometer

tau_Bs = 1.5 * picosecond    # a particle lifetime, say the Bs meson's
ctau_Bs = c_light * tau_Bs   # ctau of the particle, ~450 microns
print(ctau_Bs)               # result in HEP units, so mm
```

```
0.44968868700000003
```

```
print(ctau_Bs / micrometer)  # result in micrometers
```

```
449.688687
```

Quantity	Name	Unit
Length	millimeter	mm
Time	nanosecond	ns
Energy	Mega electron Volt	MeV
Positron charge	eplus	
Temperature	kelvin	K
Amount of substance	mole	mol
Luminous intensity	candela	cd
Plane angle	radian	rad
Solid angle	steradian	sr

- **More "advanced":**

```
from hepunits import c_light, GeV, meter, ps
from math import sqrt

def ToF(m, p, l):
    """Time-of-Flight = particle path length l / (c * beta)"""
    one_over_beta = sqrt(1 + m*m/(p*p))
    return (l * one_over_beta /c_light)
```

```
from particle.particle.literals import pi_plus, K_plus  # particle name literals
```

```
delta = ( ToF(K_plus.mass, 10*GeV, 10*meter) - ToF(pi_plus.mass, 10*GeV, 10*meter) ) / ps
print("At 10 GeV, Delta-TOF(K-pi) over 10 meters = {:.5} ps".format(delta))
```

```
At 10 GeV, Delta-TOF(K-pi) over 10 meters = 37.374 ps
```