Access and manipulation of complex data structures: Uproot and Awkward Array

Jim Pivarski

Princeton University – IRIS-HEP

October 26, 2020

Uproot & Awkward Array: part of this complete analysis

uproot

Reading and writing ROOT files (just I/O)

ServiceX

Remote data

DOMA Networks

func-adl

Remote queries

SSL

Awkward Array

Manipulating arrays with nested structure (not HEP-specific)

hep-tables

DataFrame for nested structure

Coffea (/ˈkɔ.fi/)

NanoEvents,
Lorentz vectors,
Histogramming,
Correction functions,
Distributed processing...

iminuit

Raw minimization

zfit

Curve fits

pyhf

∂ifferentiable ℒikelihoods

HistFactory-style fits

hepstats

Statistical tools

vector

2D, 3D, & Lorentz vectors

Boost histogram & hist!

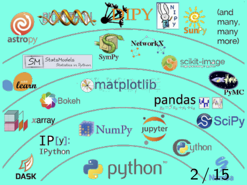
Histogramming

mplhep

Plotting

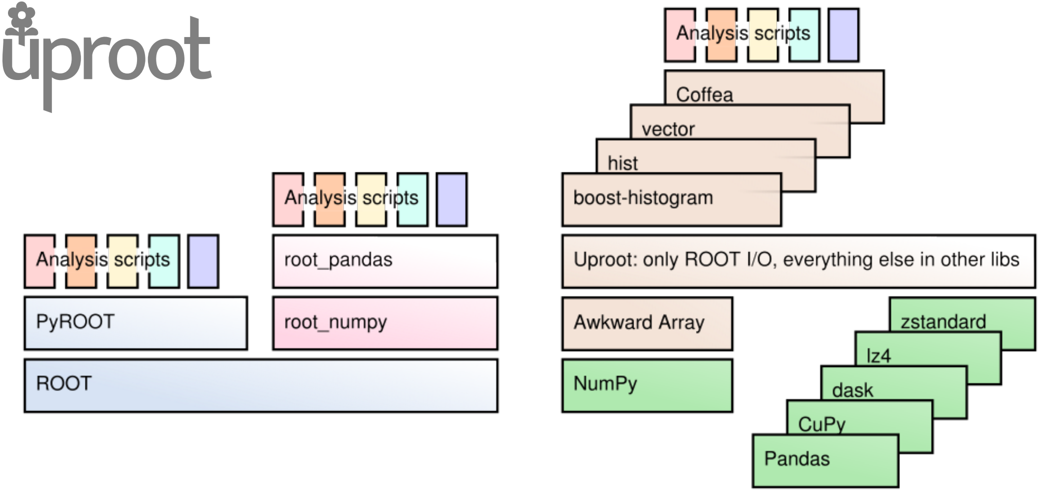
Particle

Pythonic PDG



Uproot is ROOT I/O rewritten in Python

uproot



Analysis scripts

PyROOT

ROOT

Analysis scripts

root_pandas

root_numpy

Analysis scripts

Coffea

vector

hist

boost-histogram

Uproot: only ROOT I/O, everything else in other libs

Awkward Array

NumPy

zstandard

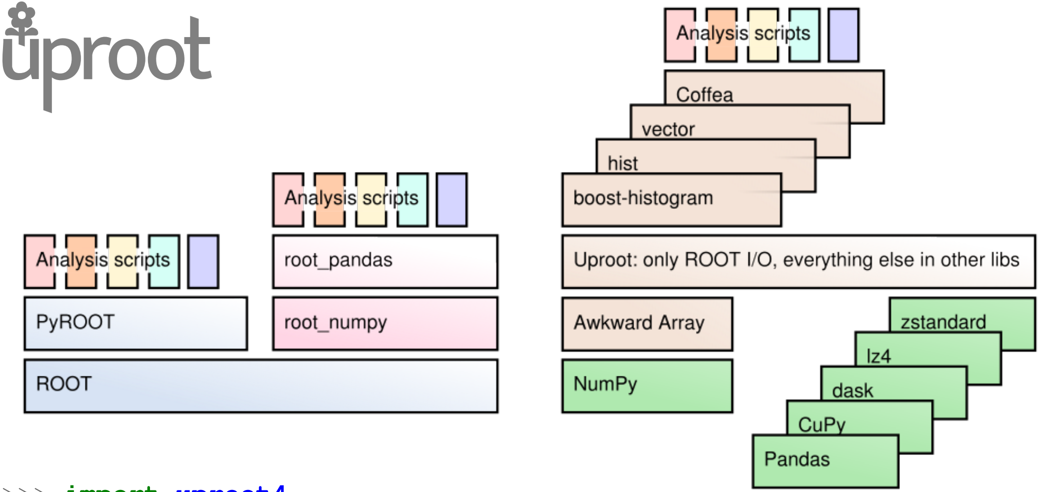
lz4

dask

CuPy

Pandas

Uproot is ROOT I/O rewritten in Python



```
>>> import uproot4
>>> data = uproot4.open("file.root:Events").arrays()
```

Awkward Array is NumPy for JSON-like data

```
>>> import awkward1 as ak
>>> import numpy as np
>>>
>>> @ak.mixin_class(ak.behavior)
... class Lorentz:
...     @property
...     def pt(self):
...         return np.sqrt(self.px**2 + self.py**2)
...
>>> array = ak.Array([{"px": 1, "py": 1, "pz": 1, "E": 1},
...                   {"px": 2, "py": 2, "pz": 2, "E": 2},
...                   {"px": 3, "py": 3, "pz": 3, "E": 3},
...                   {"px": 4, "py": 4, "pz": 4, "E": 4},
...                   {"px": 5, "py": 5, "pz": 5, "E": 5}],
...                  with_name="Lorentz")
...
```

Awkward Array is NumPy for JSON-like data

```
>>> import awkward1 as ak
>>> import numpy as np
>>>
>>> @ak.mixin_class(ak.behavior)
... class Lorentz:
...     @property
...     def pt(self):
...         return np.sqrt(self.px**2 + self.py**2)
...
>>> array = ak.Array([{"px": 1, "py": 1, "pz": 1, "E": 1},
...                   {"px": 2, "py": 2, "pz": 2, "E": 2},
...                   {"px": 3, "py": 3, "pz": 3, "E": 3},
...                   {"px": 4, "py": 4, "pz": 4, "E": 4},
...                   {"px": 5, "py": 5, "pz": 5, "E": 5}],
...                  with_name="Lorentz")
...
>>> array[-2]
<LorentzRecord {px: 4, py: 4, pz: 4, E: 4} type='Lorentz["px": int64, "py": int6...'>
```

Awkward Array is NumPy for JSON-like data

```
>>> import awkward1 as ak
>>> import numpy as np
>>>
>>> @ak.mixin_class(ak.behavior)
... class Lorentz:
...     @property
...     def pt(self):
...         return np.sqrt(self.px**2 + self.py**2)
...
>>> array = ak.Array([{"px": 1, "py": 1, "pz": 1, "E": 1},
...                   {"px": 2, "py": 2, "pz": 2, "E": 2},
...                   {"px": 3, "py": 3, "pz": 3, "E": 3},
...                   {"px": 4, "py": 4, "pz": 4, "E": 4},
...                   {"px": 5, "py": 5, "pz": 5, "E": 5}],
...                  with_name="Lorentz")
...
>>> array[-2]
<LorentzRecord {px: 4, py: 4, pz: 4, E: 4} type='Lorentz["px": int64, "py": int6...'>
>>> array * 10
<Array [{px: 10, py: 10, pz: 10, ... E: 50}] type='5 * {"px": int64, "py": int64...'>
```

Awkward Array is NumPy for JSON-like data

Awkward Array

```
>>> import awkward1 as ak
>>> import numpy as np
>>>
>>> @ak.mixin_class(ak.behavior)
... class Lorentz:
...     @property
...     def pt(self):
...         return np.sqrt(self.px**2 + self.py**2)
...
>>> array = ak.Array([{"px": 1, "py": 1, "pz": 1, "E": 1},
...                   {"px": 2, "py": 2, "pz": 2, "E": 2},
...                   {"px": 3, "py": 3, "pz": 3, "E": 3},
...                   {"px": 4, "py": 4, "pz": 4, "E": 4},
...                   {"px": 5, "py": 5, "pz": 5, "E": 5}],
...                  with_name="Lorentz")
...
>>> array[-2]
<LorentzRecord {px: 4, py: 4, pz: 4, E: 4} type='Lorentz["px": int64, "py": int6...'>
>>> array * 10
<Array [{px: 10, py: 10, pz: 10, ... E: 50}] type='5 * {"px": int64, "py": int64...'>
>>> array.pt
<Array [1.41, 2.83, 4.24, 5.66, 7.07] type='5 * float64'>
```

Awkward Array is NumPy for JSON-like data

```
>>> import awkward1 as ak
>>> import numpy as np
>>>
>>> @ak.mixin_class(ak.behavior)
... class Lorentz:
...     @property
...     def pt(self):
...         return np.sqrt(self.px**2 + self.py**2)
...
>>> array = ak.Array([[{"px": 1, "py": 1, "pz": 1, "E": 1},
...                    {"px": 2, "py": 2, "pz": 2, "E": 2},
...                    {"px": 3, "py": 3, "pz": 3, "E": 3}],
...                   [],
...                   [{"px": 4, "py": 4, "pz": 4, "E": 4},
...                    {"px": 5, "py": 5, "pz": 5, "E": 5}]],
...                  with_name="Lorentz")
...
>>> array[0, -1]
<LorentzRecord {px: 3, py: 3, pz: 3, E: 3} type='Lorentz["px": int64, "py": int6...'>
>>> array * 10
<Array [[{px: 10, py: 10, ... E: 50}]] type='3 * var * {"px": int64, "py": int64...'>
>>> array.pt
<Array [[1.41, 2.83, 4.24], ... [5.66, 7.07]] type='3 * var * float64'>
```

Awkward Array is NumPy for JSON-like data

Awkward Array

```
>>> import awkward1 as ak
>>> import numpy as np
>>>
>>> @ak.mixin_class(ak.behavior)
... class Lorentz:
...     @property
...     def pt(self):
...         return np.sqrt(self.px**2 + self.py**2)
...
>>> array = ak.Array([[[{"px": 1, "py": 1, "pz": 1, "E": 1},
...                     {"px": 2, "py": 2, "pz": 2, "E": 2}],
...                    [{"px": 3, "py": 3, "pz": 3, "E": 3}]],
...                   [[]],
...                   [[{"px": 4, "py": 4, "pz": 4, "E": 4}],
...                    [],
...                    [{"px": 5, "py": 5, "pz": 5, "E": 5}]]],
...                  with_name="Lorentz")
>>> array[-1, 2, 0]
<LorentzRecord {px: 5, py: 5, pz: 5, E: 5} type='Lorentz["px": int64, "py": int6...'>
>>> array * 10
<Array [[[{px: 10, py: 10, ... E: 50}]]] type='3 * var * var * {"px": int64, "py...'>
>>> array.pt
<Array [[[1.41, 2.83], [4.24, ... [], [7.07]]] type='3 * var * var * float64'>
```

Awkward Array has NumPy-like conciseness and performance

```
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:])
```

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

4.6 seconds to run (2 GB footprint)

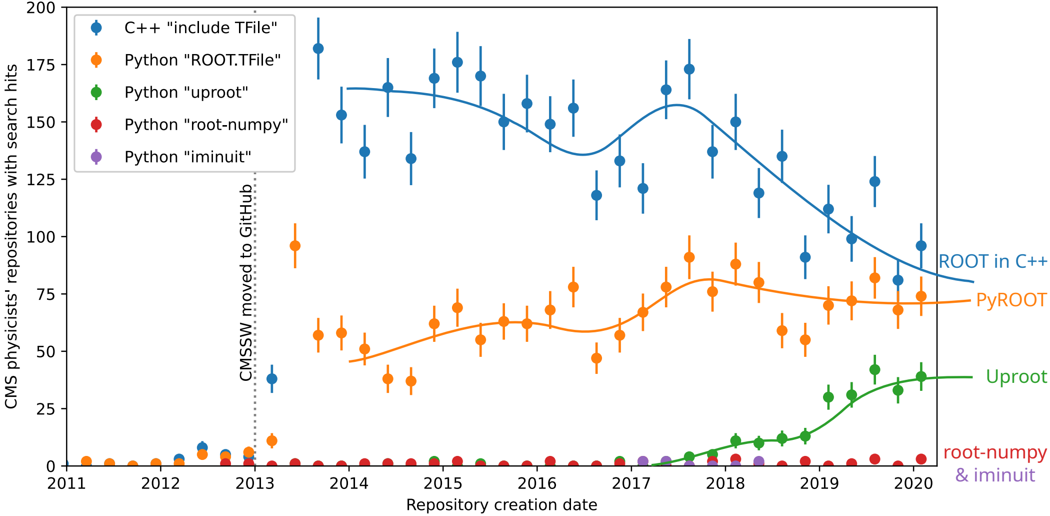
equivalent Python

```
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)
```

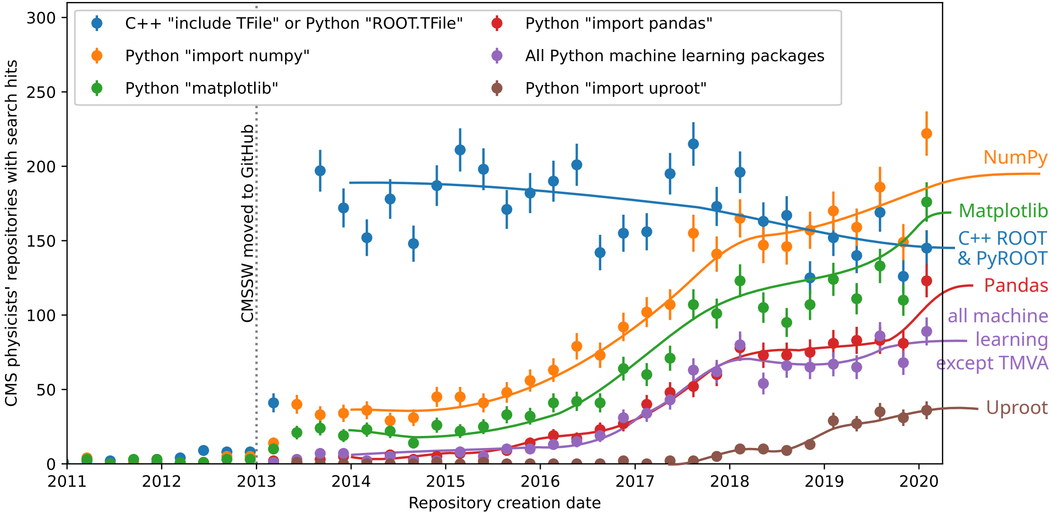
138 seconds to run (22 GB footprint)

(single-threaded on a 2.2 GHz processor with a dataset 10 million times larger than the one shown)

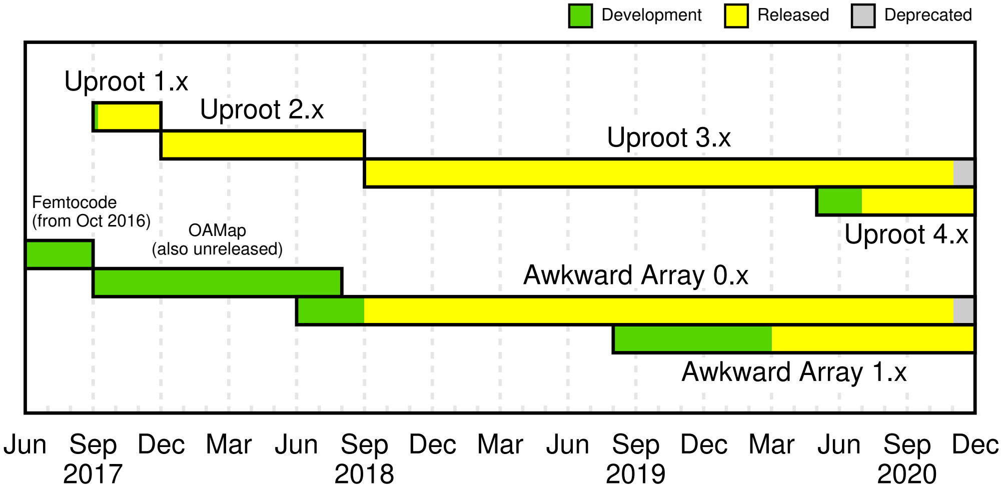
Uproot has become mainstream. . .



. . . as part of a general trend toward Python that started earlier

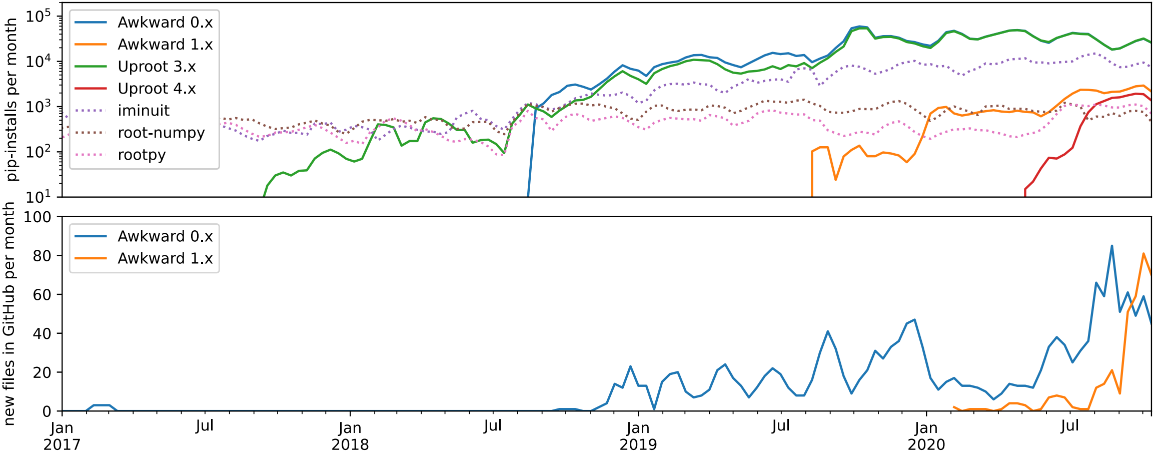


Both libraries rewritten in 2020: new interface, better architecture



Adoption of the new libraries

Plateaued at a small fraction of pip-installs, but significant cross-over for “`import awkward`” vs. “`import awkward1`” in Python files in GitHub.



“Opt-in for new” ⟶ “opt-in for old”

“Opt-in for new” ⟶ “opt-in for old”

	now	**soon**

GitHub	old:
	`scikit-hep/awkward-array`
	`scikit-hep/uproot`
	new:
	`scikit-hep/awkward-1.0`
	`scikit-hep/uproot4`
		old:
		`scikit-hep/awkward-0.x`
		`scikit-hep/uproot-3.x`
		new:
		`scikit-hep/awkward-array`
		`scikit-hep/uproot`

pip & conda	old: `install awkward uproot` new: `install awkward1 uproot4`	old: `install awkward0 uproot3` new: `install awkward uproot`

Python	old:
	`import awkward, uproot`
	new:
	`import awkward1, uproot4`
		old:
		`import awkward0, uproot3`
		new:
		`import awkward, uproot`

When is “soon”? What needs to be done first?

Awkward Array:

- Support Python 3.9.
- Move `libawkward.so` into its own package.
- Awkward 0.x → Awkward 1.x cheat sheet.

Uproot:

- ~~Implement file-writing in Uproot 4.~~
  For now, you’ll have to use Uproot 3 *for writing only*.
- Uproot 3.x → Uproot 4.x cheat sheet.

When is "soon"? What needs to be done first?

Awkward Array:

- Support Python 3.9.
- Move `libawkward.so` into its own package.
- Awkward 0.x → Awkward 1.x cheat sheet.

Uproot:

- ~~Implement file-writing in Uproot 4.~~
  For now, you'll have to use Uproot 3 *for writing only*.
- Uproot 3.x → Uproot 4.x cheat sheet.

My deadline: December 1, 2020.

Beyond that...

Major directions:

- Python ecosystem integration: Dask, Zarr, Xarray?
- Auto-differentiation: JAX, elementwise, reducers
- GPU support: direct and through Numba (`@nb.cuda.jit`)

Working demo of Awkward Array on a GPU ("direct")

This is *the visible part* of Pratyush Das & Anish Biswas's hard work this summer!

```
>>> import awkward1 as ak, numpy as np, cupy as cp

>>> on_cpu = ak.Array([[1.1, 2.2, 3.3], [], [4.4, 5.5]])
>>> on_cpu                                # this jagged array is on the CPU
<Array [[1.1, 2.2, 3.3], [], [4.4, 5.5]] type='3 * var * float64'>
```

Working demo of Awkward Array on a GPU ("direct")

This is *the visible part* of Pratyush Das & Anish Biswas's hard work this summer!

```
>>> import awkward1 as ak, numpy as np, cupy as cp

>>> on_cpu = ak.Array([[1.1, 2.2, 3.3], [], [4.4, 5.5]])
>>> on_cpu                           # this jagged array is on the CPU
<Array [[1.1, 2.2, 3.3], [], [4.4, 5.5]] type='3 * var * float64'>

>>> on_gpu = ak.to_kernels(on_cpu, "cuda")
>>> on_gpu                           # this jagged array is on the GPU
<Array:cuda [[1.1, 2.2, 3.3], ... [4.4, 5.5]] type='3 * var * float64'>
```

Working demo of Awkward Array on a GPU (“direct”)

This is *the visible part* of Pratyush Das & Anish Biswas’s hard work this summer!

```
>>> import awkward1 as ak, numpy as np, cupy as cp

>>> on_cpu = ak.Array([[1.1, 2.2, 3.3], [], [4.4, 5.5]])
>>> on_cpu                                # this jagged array is on the CPU
<Array [[1.1, 2.2, 3.3], [], [4.4, 5.5]] type='3 * var * float64'>

>>> on_gpu = ak.to_kernels(on_cpu, "cuda")
>>> on_gpu                                # this jagged array is on the GPU
<Array:cuda [[1.1, 2.2, 3.3], ... [4.4, 5.5]] type='3 * var * float64'>

>>> ak.num(on_gpu)                        # some operations work on the GPU
<Array:cuda [3, 0, 2] type='3 * int64'>
```

Working demo of Awkward Array on a GPU (“direct”)

This is *the visible part* of Pratyush Das & Anish Biswas’s hard work this summer!

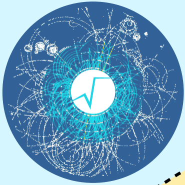
```
>>> import awkward1 as ak, numpy as np, cupy as cp

>>> on_cpu = ak.Array([[1.1, 2.2, 3.3], [], [4.4, 5.5]])
>>> on_cpu                           # this jagged array is on the CPU
<Array [[1.1, 2.2, 3.3], [], [4.4, 5.5]] type='3 * var * float64'>

>>> on_gpu = ak.to_kernels(on_cpu, "cuda")
>>> on_gpu                           # this jagged array is on the GPU
<Array:cuda [[1.1, 2.2, 3.3], ... [4.4, 5.5]] type='3 * var * float64'>

>>> ak.num(on_gpu)                   # some operations work on the GPU
<Array:cuda [3, 0, 2] type='3 * int64'>

>>> np.sqrt(on_gpu)                  # performs math on GPU (using CuPy)
<Array:cuda [[1.05, 1.48, 1.82, ... 2.1, 2.35]] type='3 * var * float64'>
```

Working demo of Awkward Array on a GPU ("direct")

This is *the visible part* of Pratyush Das & Anish Biswas's hard work this summer!

```
>>> import awkward1 as ak, numpy as np, cupy as cp

>>> on_cpu = ak.Array([[1.1, 2.2, 3.3], [], [4.4, 5.5]])
>>> on_cpu                           # this jagged array is on the CPU
<Array [[1.1, 2.2, 3.3], [], [4.4, 5.5]] type='3 * var * float64'>

>>> on_gpu = ak.to_kernels(on_cpu, "cuda")
>>> on_gpu                           # this jagged array is on the GPU
<Array:cuda [[1.1, 2.2, 3.3], ... [4.4, 5.5]] type='3 * var * float64'>

>>> ak.num(on_gpu)                   # some operations work on the GPU
<Array:cuda [3, 0, 2] type='3 * int64'>

>>> np.sqrt(on_gpu)                  # performs math on GPU (using CuPy)
<Array:cuda [[1.05, 1.48, 1.82, ... 2.1, 2.35]] type='3 * var * float64'>

>>> cp.asarray(ak.flatten(on_gpu))   # stays on the GPU as a CuPy array
array([1.1, 2.2, 3.3, 4.4, 5.5])
```

Uproot & Awkward Array: part of this complete analysis



uproot

Reading and writing ROOT files (just I/O)

ServiceX

Remote data

func-adl

Remote queries

DOMA Networks

SSL

Awkward Array

Manipulating arrays with nested structure (not HEP-specific)

hep-tables

DataFrame for nested structure

Coffea (/ˈkɔ.fi/)

NanoEvents,
Lorentz vectors,
Histogramming,
Correction functions,
Distributed processing...

iminuit

Raw minimization

zfit

Curve fits

pyhf

∂ifferentiable ℒikelihoods

HistFactory-style fits

hepstats

Statistical tools

vector

2D, 3D, & Lorentz vectors

Boost histogram & hist!

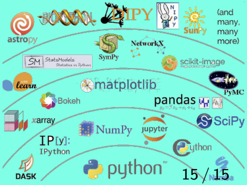
Histogramming

mplhep

Plotting

Particle

Pythonic PDG


astropy
BIOPYTHON
DIPY
NIPY
SunPy
(and many, many more)
SymPy
NetworkX
StatsModels Statistics in Python
scikit-image
scikit learn
matplotlib
PyMC
Bokeh
pandas
xarray
SciPy
NumPy
jupyter
IP[y]: IPython
Cython
DASK
python
Numba