High Performance Histograms as *Objects*

Henry Schreiner, Hans Dembinski, Jim Pivarski, Shuo Liu

SciPy2020
Scientific Computing with Python
Virtual Conference • July 6-12

iris hep
Institute for Research & Innovation in Software for High Energy Physics

PRINCETON UNIVERSITY

technische universität dortmund

中山大學
SUN YAT-SEN UNIVERSITY

A histogram is best described as an object.

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Current Status Quo

`np.histogram`
`plt.hist`
⬇
Unbinned data → bins, edge arrays

1D, 2D, ND have different functions/APIs

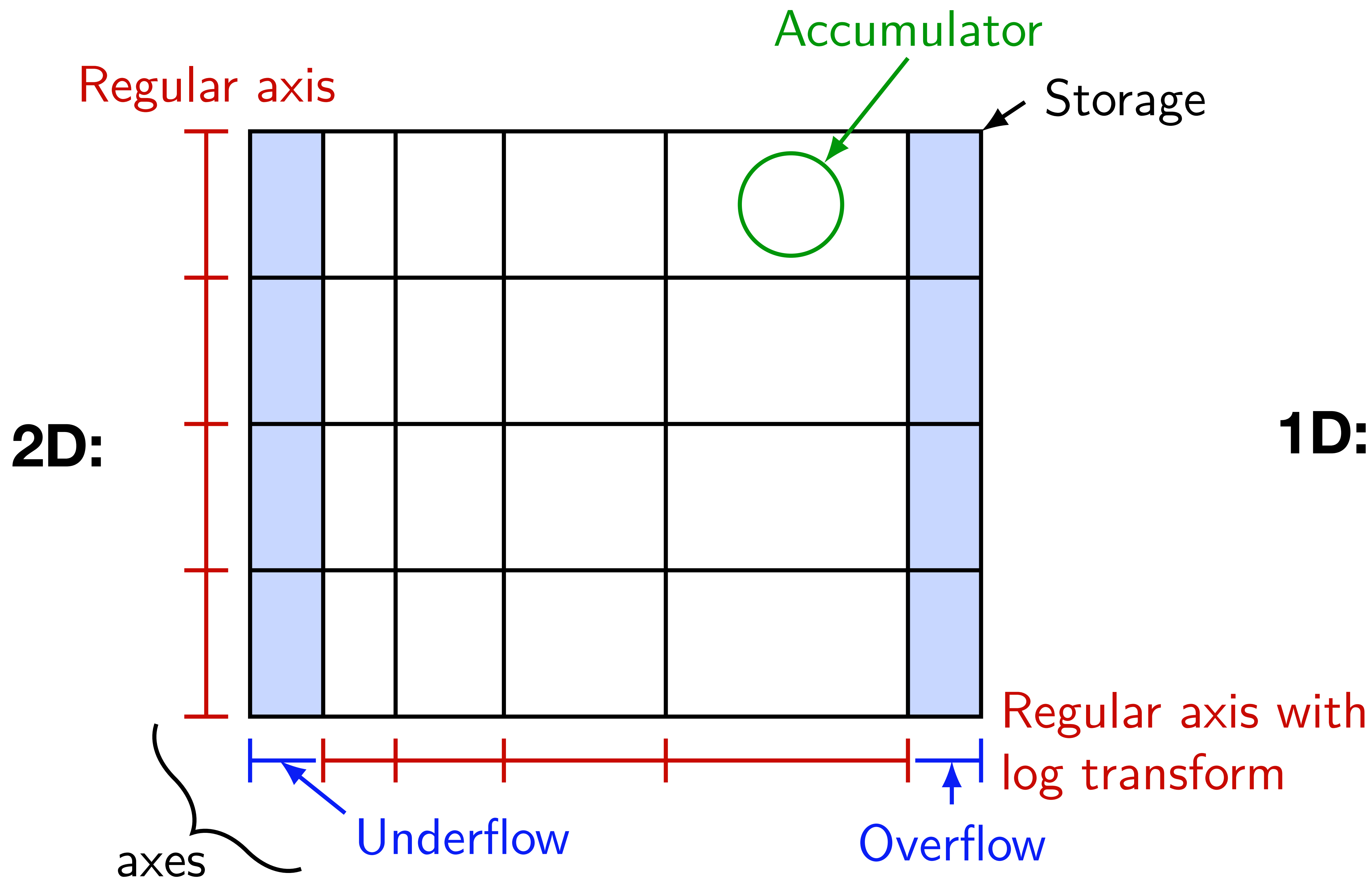
Hard to plot *results* of np.histogram

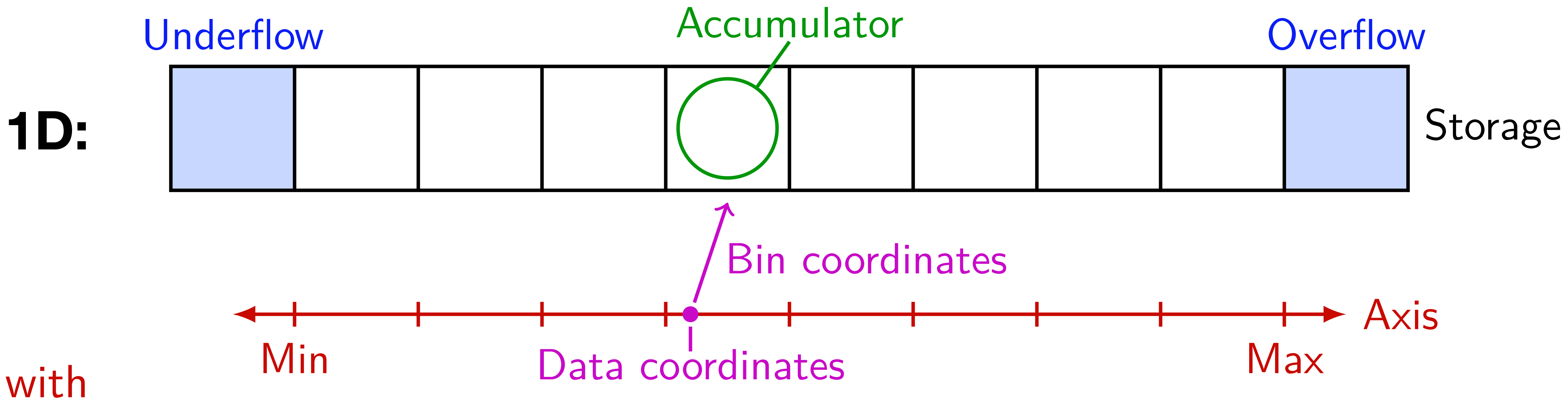
Generalized histograms:
`scipy.stats.binned_statistic`

And how about binning/manipulating histograms afterwards?

iris hep
Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

What would a “Histogram” look like?





Axes:

`Regular,`
`Variable,`
`Category,`
…

Accumulators:

`Int,`
`Double,`
`WeightedSum`
`Mean,`
…

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Basics of boost-histogram

2020

Basics:

```
h = bh.Histogram(
    bh.axis.<Type>(),
    bh.axis.<Type>(),
    ...,
    storage=bh.storage.<Type>())

h.fill(data, weights=...
             samples=...)

# Optional classic style
bins, *edges = h.to_numpy()
```

But avoid this, use h directly!

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

iris hep

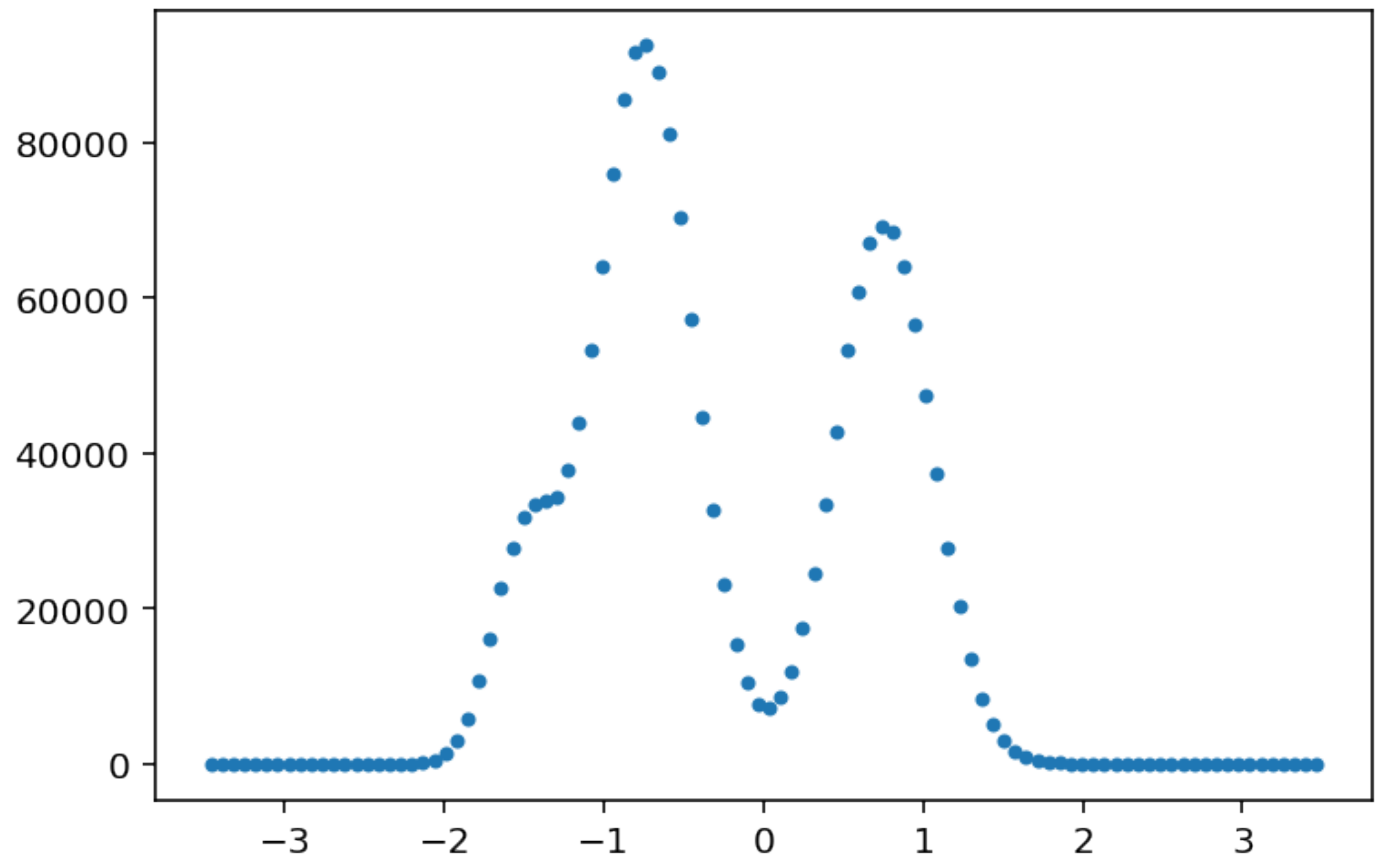
Boost istogram

A taste of manipulation

Assume we have a hist "h":

Plot a histogram: (simple)

```
plt.plot(*h.axes.centers, h, ".")
```



NumPy comparison:

```
bins, edges = h.to_numpy()
centers = (bins[1:] + bins[:-1]) / 2
plt.plot(centers, bins, ".")
```

Compute the density:

```
V = np.prod(h.axes.widths, axis=0)
density = h.view() / h.sum() / V
```


Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

A taste of manipulation

Assume we have a hist "h":

Manual threaded fill:

```python
def fun(d):
    return h.copy().reset().fill(d)

chunks = np.array_split(data, threads)

with ThreadPoolExecutor(threads) as pool:
    results = pool.map(fun, chunks)

for res in results:
    h += res
```

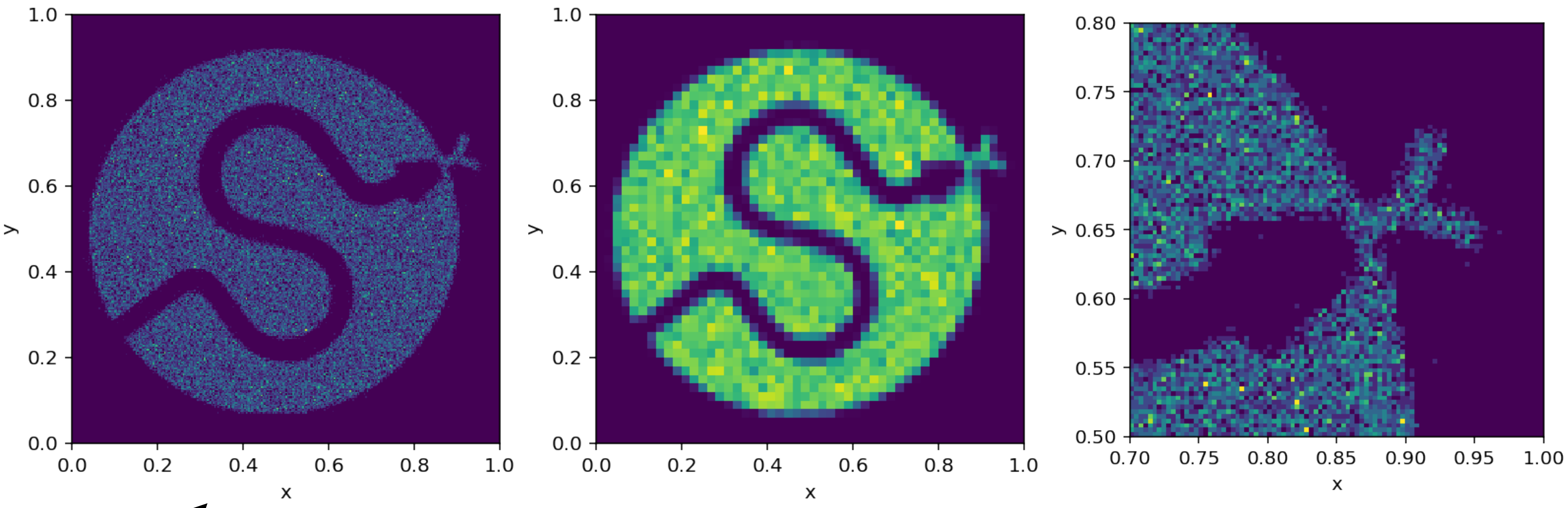
Or just use this:

```python
h.fill(data, threads=threads)
```


Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

A taste of manipulation (2)

```python
h = bh.Histogram(
  bh.axis.Regular(
    300, 0, 1,
    metadata="x"
  ),
  bh.axis.Regular(
    300, 0, 1,
    metadata="y"
  )
)
```



`h`

`h[::bh.rebin(6), ::bh.rebin(6)]`

`h[bh.loc(.7):, bh.loc(.5):bh.loc(.8)]`

iris hep
Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu
Boost histogram

And boost-histogram is *fast*

2020

Tests on 2.4 GHz 8-Core Intel Core i9

1D, 100 bins, 10,000,000 data points

	`bh.numpy.histogram(data, bins, ranges)`	`np.histogram(data, bins, ranges)`
	43.1 ms (41.6 ms in object mode)	74.5 ms
Threaded	13.8 ms (13.3 ms in object mode)	

2D, 100x100 bins, 10,000,000x2 data points

	`bh.numpy.histogram2d(data, bins, ranges)`	`np.histogram2d(data, bins, ranges)`
	84.7 ms (77.6 ms in object mode)	874 ms
Threaded	29.6 ms (28.7 ms in object mode)	

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

A firm foundation

2020

pybind11 — Seamless operability between C++11 and Python, used by SciPy, 1,934 GitHub repos

boost C++ LIBRARIES — ...one of the most highly regarded and expertly designed C++ library projects in the world.
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

Boost Histogram — Designed by Hans Dembinski, accepted into Boost 1.70, improved every version since.

Boost.Histogram features

- Static / Dynamic storage (can avoid allocator!)
- Static fill becomes 57 lines of vectorized assembly
- Adding, scaling, slicing, rebinning, projections, more
- High performance filling and bin iteration

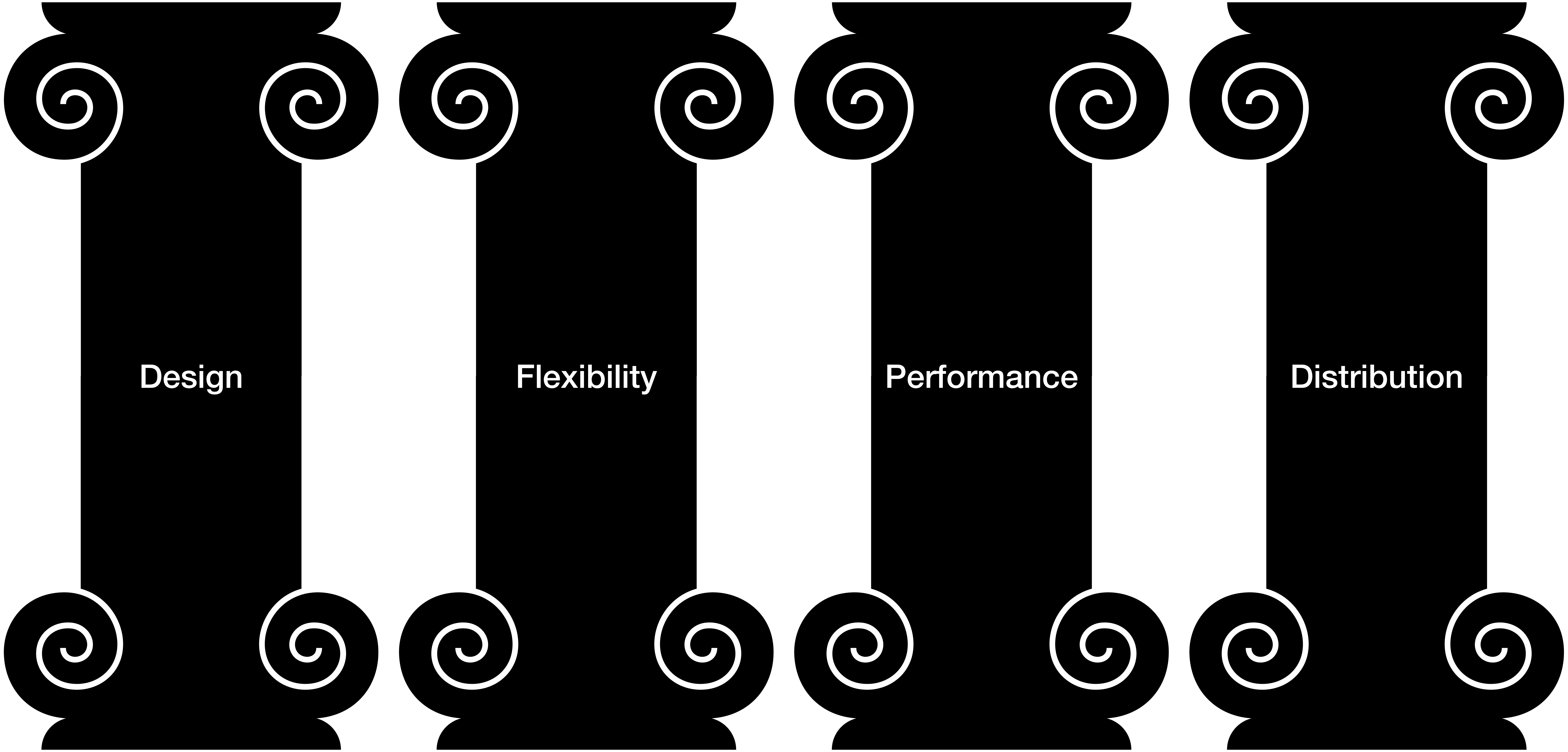
Customizable:

- Storage
- Allocators
- Accumulators
- Axes
- Axis metadata
- Axis transforms

Boost.Histogram author closely involved in boost-histogram!

New bindings: boost-histogram

2020



The bindings were designed around these four key areas

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Design

Pickle
Directly supports pickling (v>0)
Optimized for performance
Cloudpickle supported too

Copy
Supports `h.copy()`, `copy(h)`,
and `deepcopy(h)`

Operators
```
h1 + h2
h * 2.0
```

Python 2 Statement

1.0 LTS w/ Python 2 support
Python 3 should not suffer

UFuncts
NumPy 1.13 overloads being adopted
for accumulator storages

Axes access
Axes return an enhanced tuple
```
h.axes.centers
h.axes.widths
h.axes.edges
h.index(value)
h.value(index)
...
```


Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Design: Unified Histogram Indexing

Bin number
or
Callable (tag)

Reducer
or
Rebinner

Axis number

Anything that
would go in
a normal indexer

```
h[ value , ... ]
```

```
h[ start : stop : action ]
```

```
h[ { 0: slice() } ]
```

```
bin_number
bh.loc(value)
bh.loc(value) + shift
bh.underflow
bh.overflow
len
```

```
bh.rebin(N)
sum
```

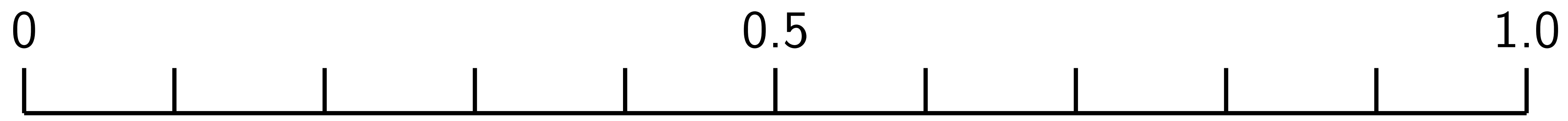
When setting with an array,
you can include flow bins
if `start` or `stop` is `None`

Any callable works, takes the
axes and returns bin number,
from `-1` to `len(ax)+1`

Currently only these implemented,
UHI spec includes arbitrary actions

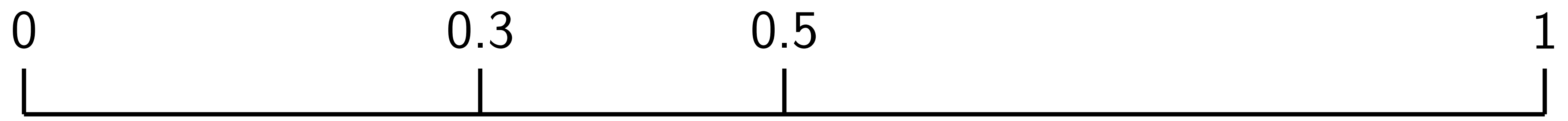
Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Flexibility: Axes Types



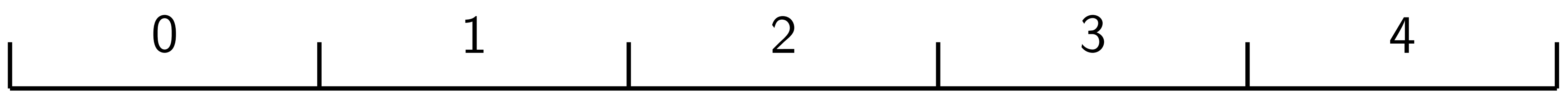
`Regular(10, 0, 1)`

$\mathcal{O}(1)$

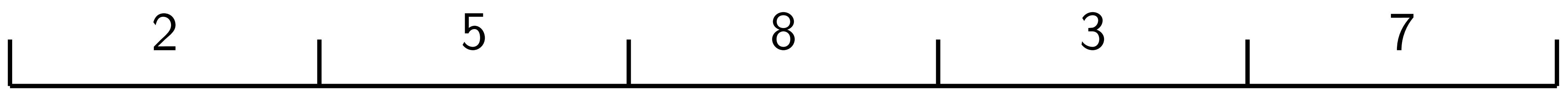


`Variable([0, 0.3, 0.5, 1.0])`

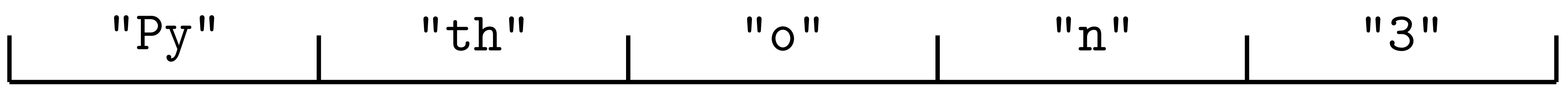
$\mathcal{O}\left(\ln n\right)$



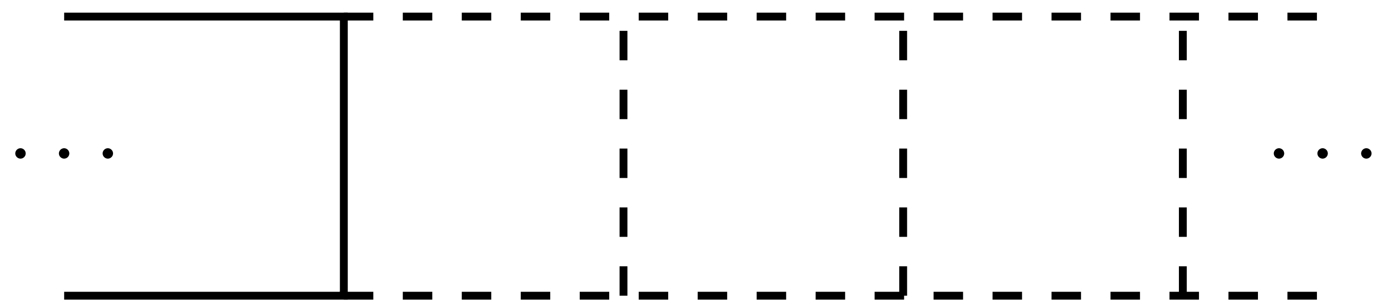
`Integer(0, 5)`



`IntCategory([2, 5, 8, 3, 7])`

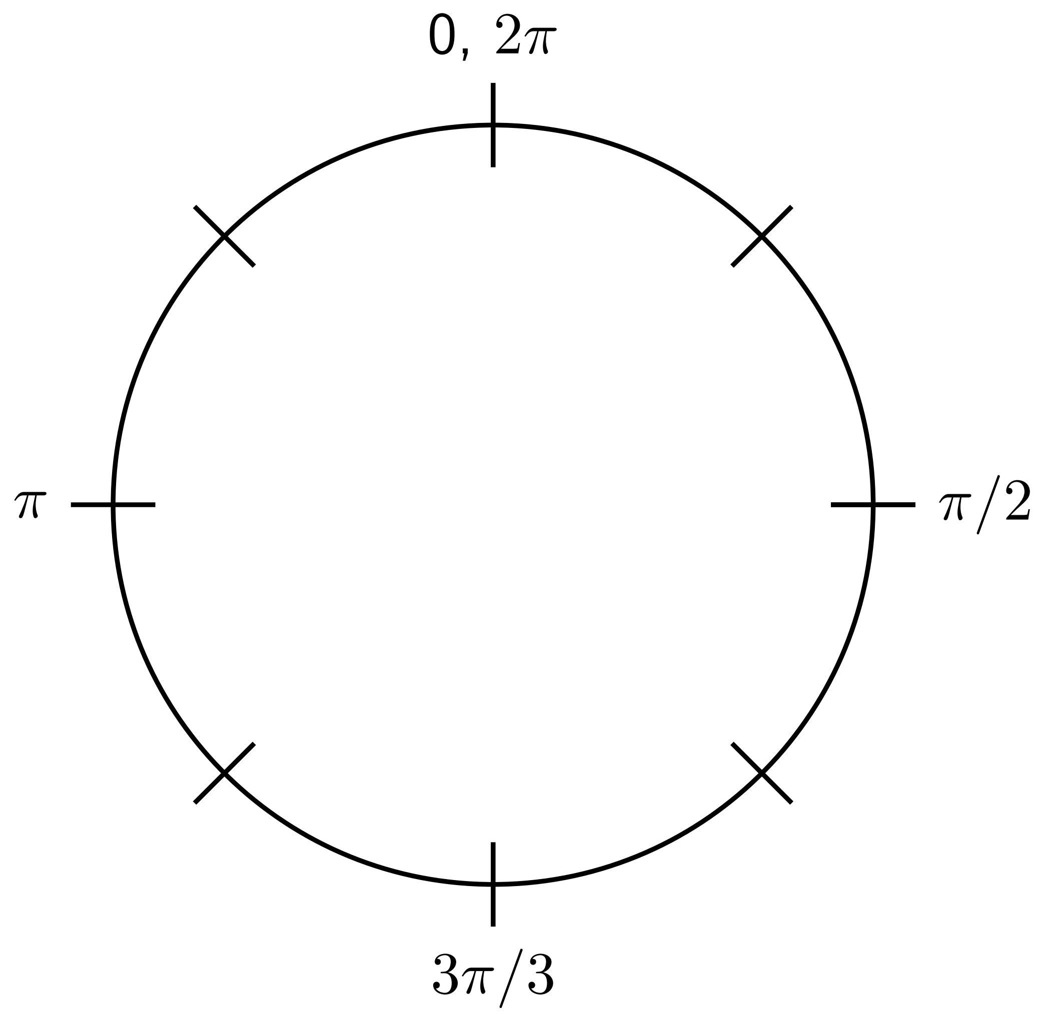


`StrCategory(["Py", "th", "o", "n", "3"])`



`growth=True`

`underflow=True` `overflow=True`



`circular=True`

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Flexibility: Transforms

Regular axes scale better than variable, no sorted lookup.

If the irregularity is functional, you can avoid the lookup!

```
h = bh.Histogram(
    bh.axis.Regular(
        10, 1, 10,
        transform=bh.axis.transform.log
    )
)
```


Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Flexibility: Transforms

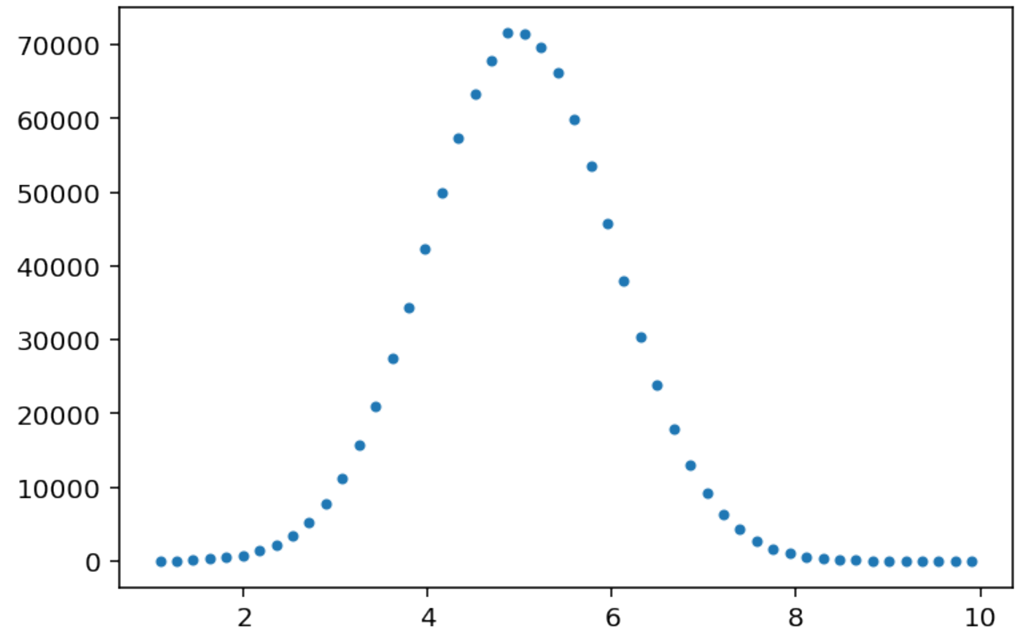
Regular axes scale better than variable, no sorted lookup.

If the irregularity is functional, you can avoid the lookup!

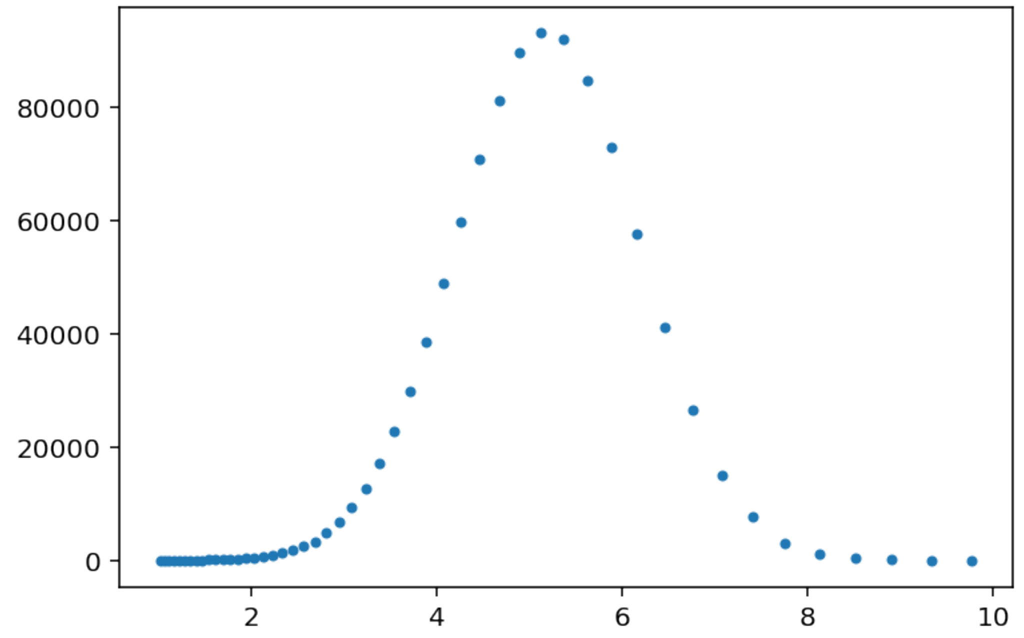
You are not just limited to precompiled transforms (sqrt, log, Pow)!

```python
import numba

@numba.cfunc(numba.float64(numba.float64))
def exp(x):
    return math.exp(x)

@numba.cfunc(numba.float64(numba.float64))
def log(x):
    return math.log(x)

bh.axis.Regular(10, 1, 4,
                transform=bh.axis.transform.Function(log, exp))
```

```python
plt.plot(*ho.axes.centers, ho, '.')
```




```python
plt.plot(*hl.axes.centers, ho, '.')
```



You can use any ctypes function pointer, or pure Python (slower)

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Flexibility: Storages

Boost-histogram ships with 7 storages:

`Double()` (default)
Fast, flexible, simple

`Int64()`
Fast, strict, simple

`AtomicInt64()`
Threadsafe fills

`Unlimited()`
No overflow guarantee,
Resizes to optimize small ints

Accumulator storages

`Weight()`
Stores value and variance

`Mean()`
High accuracy

`WeightedMean()`
Mean and weight

Views into accumulators are "smart":

- NumPy record structure
- Property access to real and computed values
- NEP 13 (NumPy 1.13+) UFunc support
- Accessing an element returns an accumulator

```
h.view().value
h.view().variance
np.sum(h.view())
```

The accumulators (including `Sum()`) act like 0D histograms, and can be filled!

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Performance

Tests on 2.4 GHz 8-Core Intel Core i9

1D, 100 bins, 10,000,000 data points

2D, 100x100 bins, 10,000,000x2 data points

Setup	Single threaded	X	Multithreaded	X
NumPy 1D	74.5 ± 2.4 ms	1		
BH 1D	41.6 ± 0.7 ms	1.8	13.3 ± 0.2 ms	5.5
BHNP 1D	43.1 ± 0.8 ms	1.7	13.8 ± 0.2 ms	5.4
NumPy 2D	874 ± 22 ms	1		
BH 2D	77.6 ± 0.6 ms	11	28.7 ± 0.7 ms	30
BHNP 2D	85 ± 3 ms	10	29.6 ± 0.5 ms	29

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Performance: Single Histogram loop

```python
value_ax = bh.axis.Regular(100, -5, 5)
valid_ax = bh.axis.Integer(0, 2,
                           underflow=False,
                           overflow=False)
label_ax = bh.axis.StrCategory([], growth=True)

hist = bh.Histogram(value_ax, valid_ax, label_ax)

hist.fill([-2, 2, 4, 3],
          [True, False, True, True],
          ["a", "b", "a", "b"])

# Just valid data, combine all labels
all_valid = hist[:, bh.loc(True), ::sum]

# 2D histogram of just the “a” label
a_only = hist[..., bh.loc("a")]
```

Loops over the data just once!

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Distribution

2020

Making a distribution:

- Needs to work everywhere

\+ Header-only Boost + PyBind11

\- C++14

Wheels

Python	2.7	3.5	3.6	3.7	3.8
Manylinux1*	✅	✅	✅	✅	✅
Manylinux2010	✅	✅	✅	✅	✅
macOS 10.9+	✅	✅	✅	✅	✅
Windows	✳️	✅	✅	✅	✅

Conda-Forge

Python	2.7	3.5	3.6	3.7	3.8
Linux 64	✴️	✅	✅	✅	✅
Linux ppc64le		✅	✅	✅	✅
Linux aarch64		✅	✅	✅	✅
macOS 10.9+	✴️	✅	✅	✅	✅
Windows		✅	✅	✅	✅

Solutions:

- git submodules for Boost + Pybind11
- Fully setuptools-based build, optional CMake build
- Custom wheel build system
  - Custom image with GCC 9 (manylinux1)
  - Manylinux2014 arch wheels under investigation
  - Replaced by cibuildwheel recently
  - 4-5 other Scikit-HEP packages now provide wheels
- SDist builds on a modern system
  - PEP 518; C++14 support required

https://iscinumpy.gitlab.io/categories/azure-devops/

https://scikit-hep.org/developer

Distribution: checks

Pre-commit & GHA:

- Black
- Pre-commit-hooks
- Flake8 with bugbear
- MyPy (basic)
- Check Manifest
- Clang Format (C++)

Test suite:

- PyTest
- PyTest-Benchmark
- Weekly dependency check

Releases:

- Automated by GHA
- setuptools_scm for versioning

See https://scikit-hep.org/developer
Several other packages in Scikit-HEP are now adopting recommendations and build tools!

pyhf ∂ifferentiable ℒikelihoods

iminuit

Awkward Array

Particle

Decay Language

Boost Histogram

iris hep

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

The Scikit-HEP ecosystem

Focused packages working together.

Convert between formats
- NumPy, Physt, and HEP-specific formats

Hist:
Friendly histograms for analysis
- GSoC project just started

Fill and manipulate histograms
- Core library

mplhep

Extensions for HEP plotting
- Histograms and more

histoprint:
Display histograms on the command line
- Newest member of Scikit-HEP
- Can plot up to 5 at a time

Scikit-HEP Tutorials:
Packages working together
- Powerered by JupyterBook Beta

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Hist

Python 3.6+ library for users

Testing useful shortcuts (some *may* be upstreamed if popular)

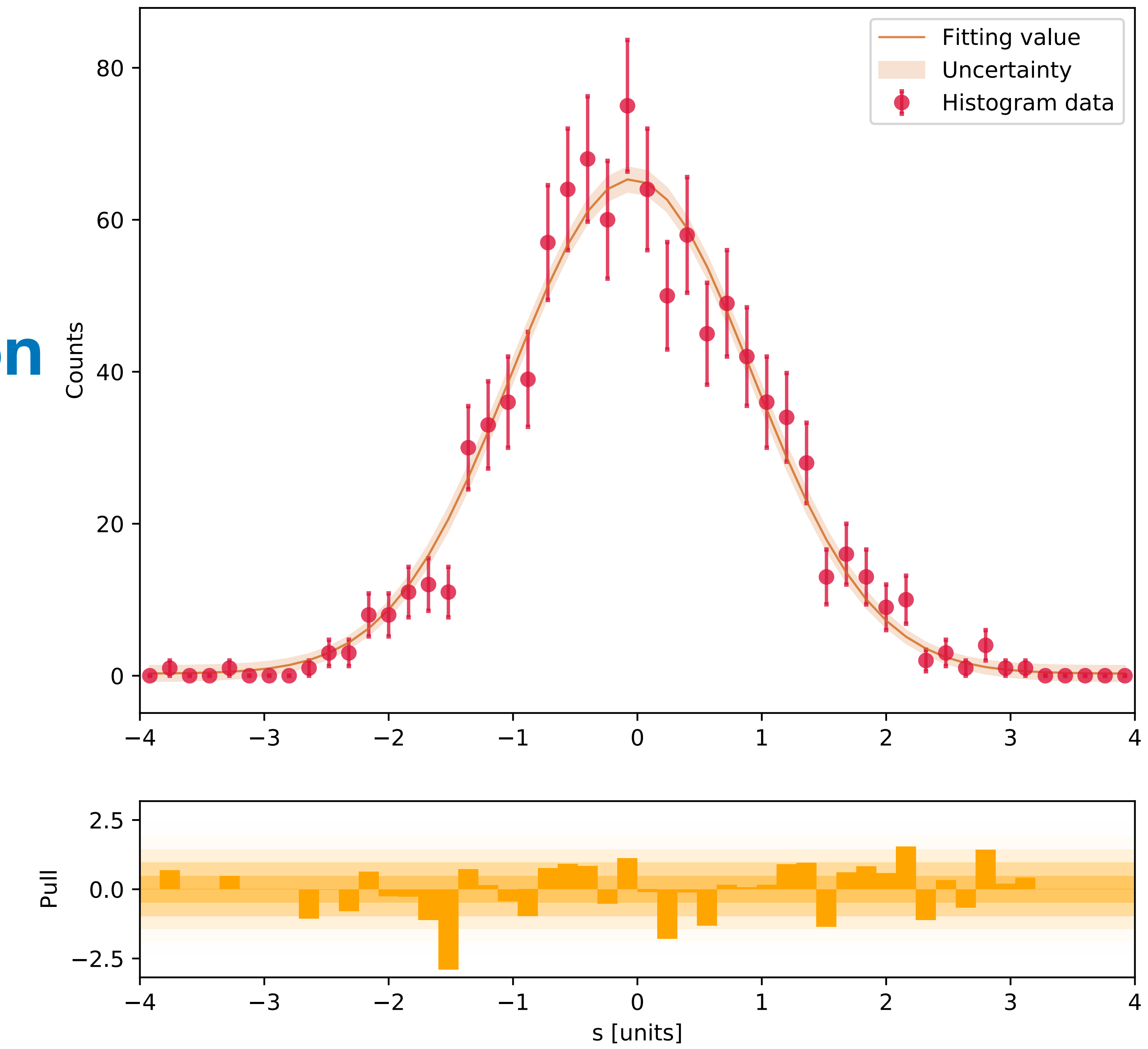
```
h[bh.loc(.7):, bh.loc(.5):bh.loc(.8)+1] → h[.7j, .5j:.8j+1]
h[bh.loc("hi")] → h["hi"]
```

Assign meaning to metadata
name, title, units, etc.

Fill and access by name instead of position
(NamedHist enforces name access)

Direct access to plots
1D, 2D, pull plots, and more

Allowed to have dependencies
such as matplotlib



Early example of pull plot

iris hep
Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Boost histogram

Summary

Design
Built on a firm foundation
Can handle complex situations with ease

Flexibility
Dozens of axes types x 7 storages
Can be extended in the future

Performance
Fast, efficient filling loops, with threads
Can reduce the number of histograms used

Distribution
Easy to build on a modern system
Wheels for every platform
Conda-forge support as well

Easy to install:
`pip install boost-histogram`
Or
`conda install boost-histogram -c conda-forge`

Easy to convert from NumPy:
`np.histogram*` → `bh.numpy.histogram*`
Add `histogram=bh.Histogram` to return object
Use `.to_numpy()` to get NumPy tuple back

And more exciting developments coming this Summer, like Hist!

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

A histogram is best described as an object.

Hopefully you now agree with me.

And boost-histogram is a great example of one!

Acknowledgments

Support for this work was provided by the National Science Foundation cooperative agreement OAC-1836650 (IRIS-HEP) and OAC-1450377 (DIANA/HEP).

https://boost-histogram.readthedocs.io

https://github.com/scikit-hep/boost-histogram

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu

Reference links

https://root.cern

https://scipy.org

https://numpy.org

https://iris-hep.org

https://scikit-hep.org

https://www.boost.org

https://numba.pydata.org

https://pandas.pydata.org

https://github.com/scikit-hep/hist

https://github.com/pybind/pybind11

https://github.com/joerick/cibuildwheel

https://github.com/scikit-hep/histoprint

https://github.com/scikit-hep/awkward-1.0

Henry Schreiner Hans Dembinski Jim Pivarski Shuo Liu