Histogram Serialization

PyHEP

Henry Schreiner • 2025-10-27

Introduction

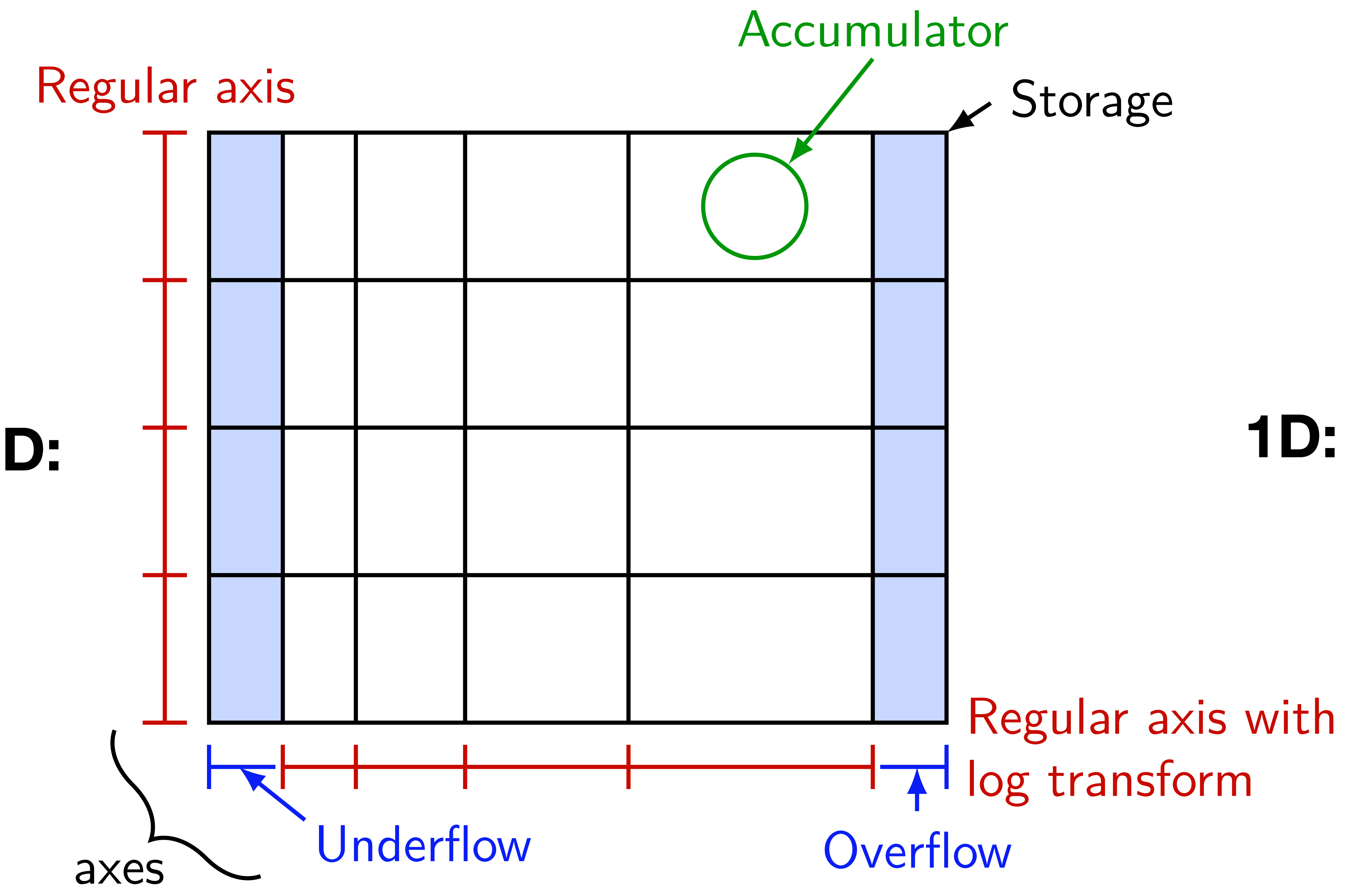
Boost-histogram, hist, uhi, and friends

- **boost-histogram**: core library based on the C++ Boost library
- **Hist**: Extra features like named axes, plotting
- **uhi**: Specification for histogram indexing (new test helpers!), plotting, and serialization (new!)
- Other libraries share via uhi, such as **PyROOT**, **uproot**, **babyyoda**, **histoprint**, and **mplhep**

Histograms are first-class objects

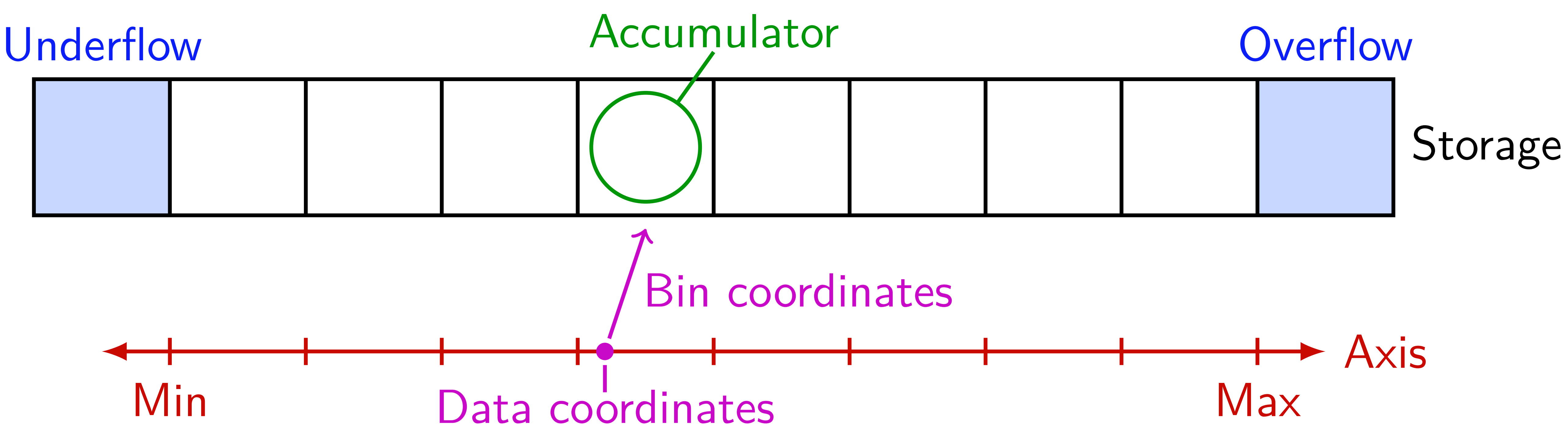
Collection of axes + storage (accumulators)

2D:



Accumulator
Regular axis
Storage
Regular axis with log transform
Underflow
Overflow
axes

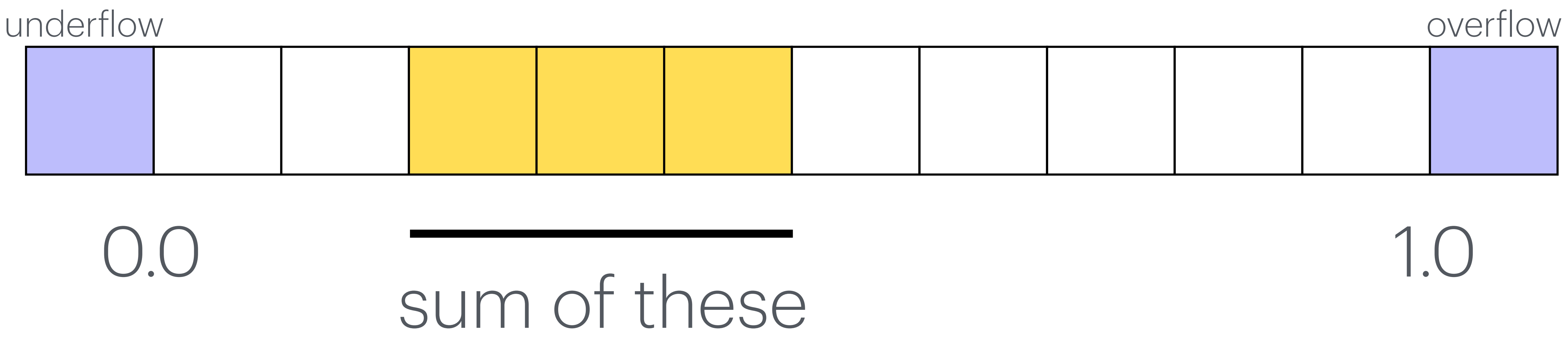
1D:



Underflow
Accumulator
Overflow
Storage
Bin coordinates
Axis
Min
Data coordinates
Max

Introduction to histograms

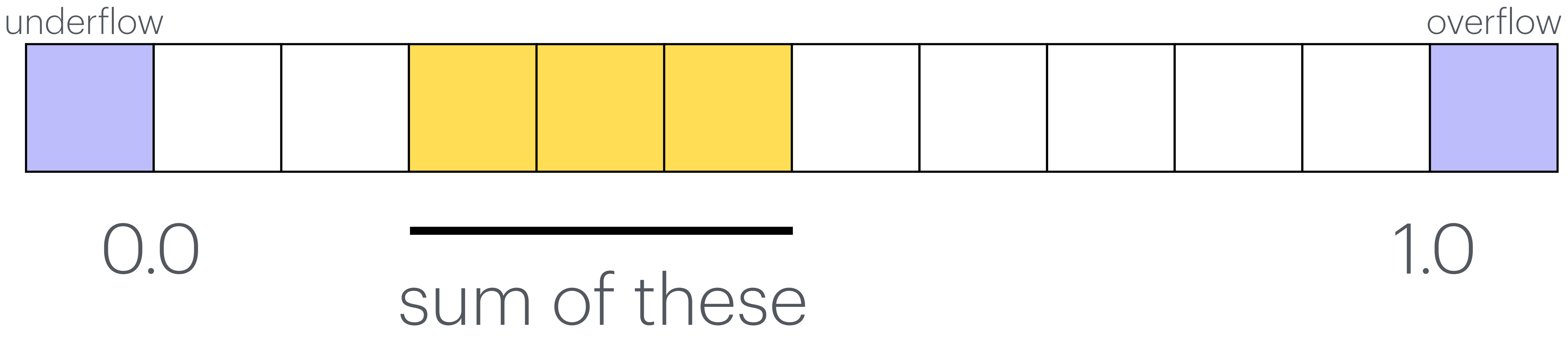
Boost-histogram



```python
import boost_histogram as bh

h = bh.Histogram(
  bh.axis.Regular(bins=10, start=0, stop=1)
)

h[bh.loc(.2):bh.loc(.5):sum]
```

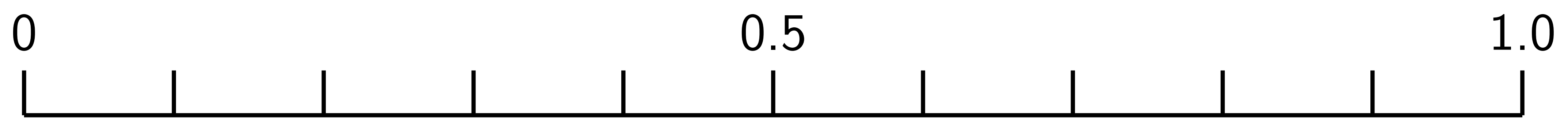
Introduction to histograms

Hist



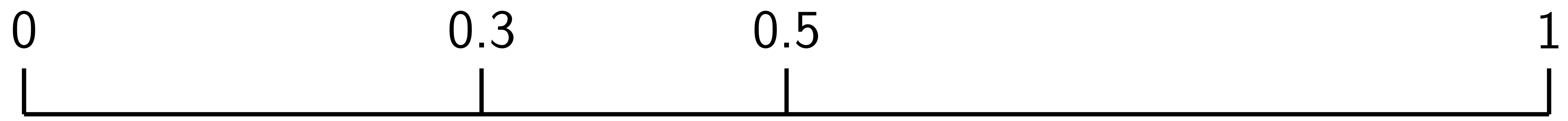
```
import hist

h = hist.Hist.new.Reg(bins=10, start=0, stop=1).Double()

h[.2j:.5j:sum]
```

Many different axes types



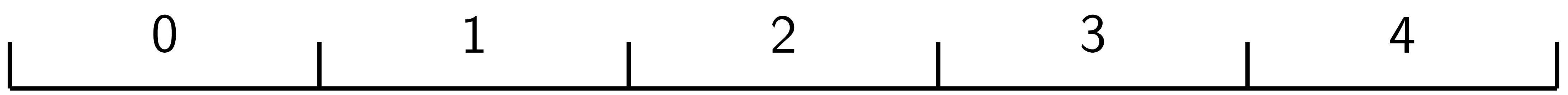
Regular(10, 0, 1)

$\mathcal{O}(1)$

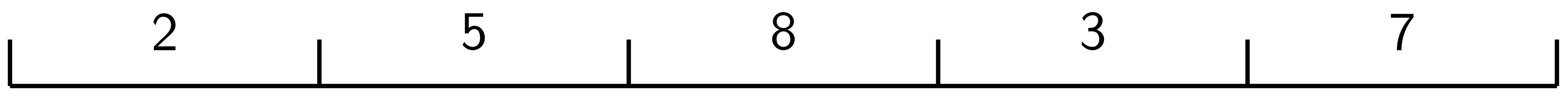


Variable([0, 0.3, 0.5, 1.0])

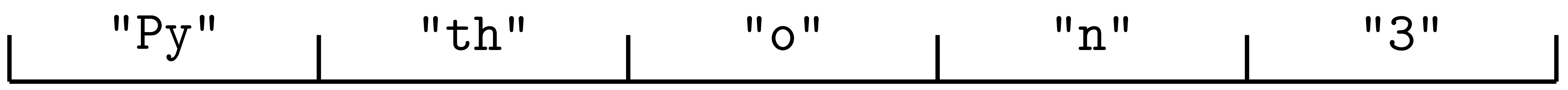
$\mathcal{O}(\ln n)$



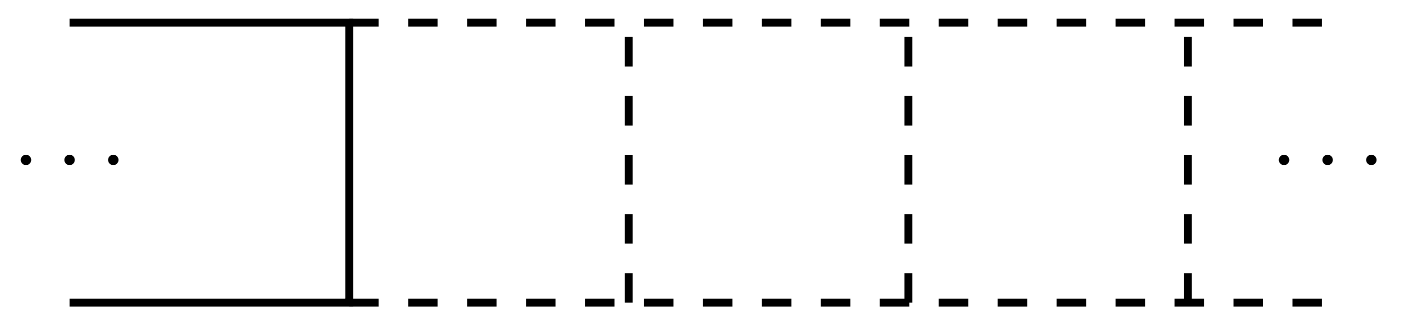
Integer(0, 5)



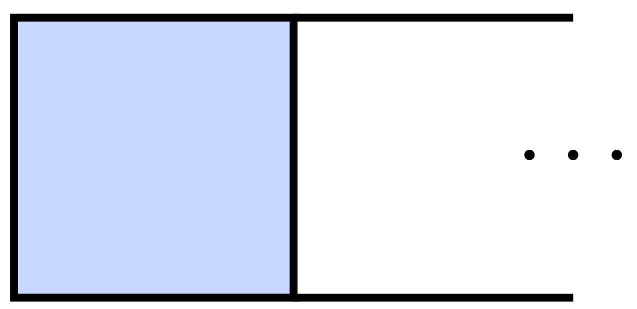
IntCategory([2, 5, 8, 3, 7])



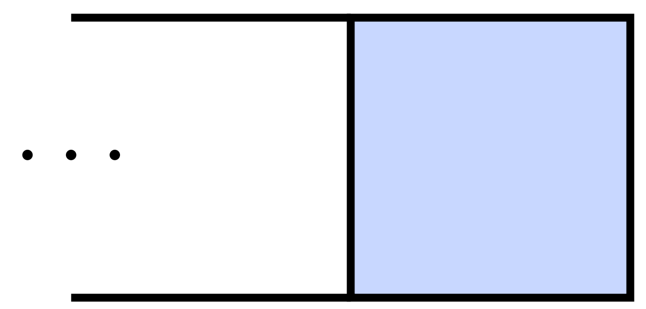
StrCategory(["Py", "th", "o", "n", "3"])



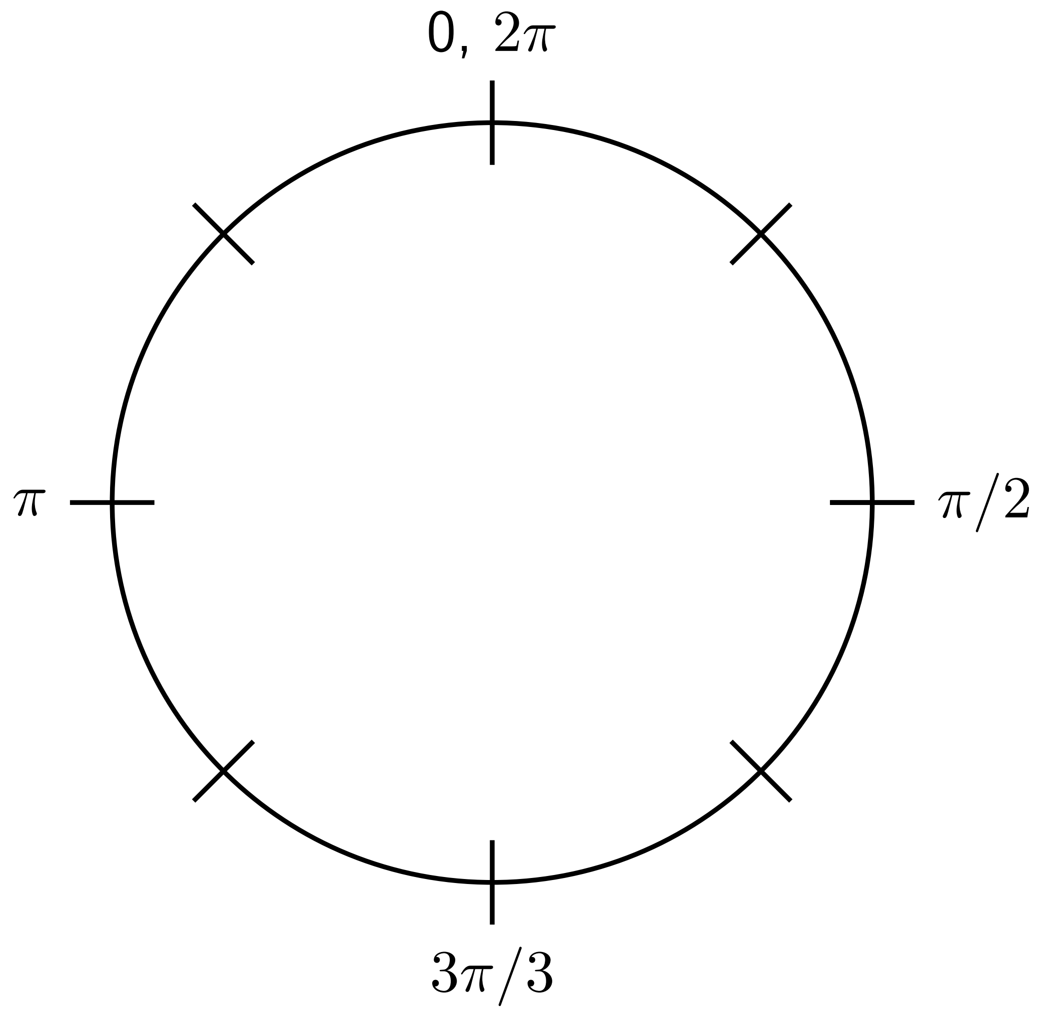
growth=True



underflow=True



overflow=True



circular=True

Boost-histogram and Hist

Hist adds to boost-histogram

Boost histogram

Compiled Boost::Histogram wrapper

ND histograms

Many different axes types to pick from

Several storages (accumulators)

Fast fills, multithreaded support

Manipulation of filled histograms

Hist

Everything in boost-histogram (builds on it)

New construction options (like existing data)

QuickConstruct

Names and labels (usable everywhere)

New methods like density and fill_flattened (Awkward)

Advanced Indexing+ shortcuts

Built-in plotting helpers (pull, pie, etc)

Stacks of histograms

Some interval functions

Part of a larger family

Histogram tooling

What about **histoprint**, **mplhelp**, **uproot**, **yoda**, **PyROOT**, etc?

We introduced uhi: a set of standards that libraries can implement

(It's also a package itself that has optional helpers)

uhi

Unified Histogram Interface

Indexing

Access
Setting
Slices
Projections
Rebinning

New: test framework!

Indexing+

Loc shortcuts
name-based syntax
Rebinning shortcut

Hist only

Plotting

Protocol for basics

Serialization

New!

Some now implemented by PyROOT and babyyoda!

uhi: Indexing

Examples

```
histogram[start:stop:action]
```

```
v = h[b]
v = h[loc(b)]
v = h[loc(b) + 1]
v = h[underflow]
```

```
h[b] = v
h[loc(b)] = v
h[underflow] = v
h[...] = array(...)
```

```
h[{0: np.s_[::rebin(2)]}]
h[{1: np.s_[0:loc(3.5)]}]
h[{7: np.s_[0:2:rebin(4)]}]
```

```
h == h[:]
h2 = h[a:b]
h2 = h[:b]
h2 = h[loc(v):]
h2 = h[::rebin(2)]
h2 = h[a:b:rebin(2)]
v2 = h[::sum]
v2 = h[a:b:sum]
v2 = h[0:len:sum]
h2 = h[v, a:b]
h2 = h[a:b, ...]
```

uhi: test framework

Useful to test implementations!

We provide this

Your library here

```python
class TestAccess1D(uhi.testing.indexing.Indexing1D[bh.Histogram]):
    @classmethod
    def make_histogram(cls) -> bh.Histogram:
        return bh.Histogram(cls.get_uhi())
```

Best with serialization support!

Then run it with pytest!

1D, 2D, and 3D provided

Used by PyROOT and babyyoda

uhi: test framework

Useful to test implementations!

We provide this

```
class TestAccess1D(uhi.testing.indexing.Indexing1D):
    @classmethod
    def make_histogram(cls):
        return bh.Histogram(cls.get_uhi())
```

Best with serialization support!

Then run it with pytest!

1D, 2D, and 3D provided

Used by PyROOT and babyyoda

uhi: Serialization

Two part design

- Started by designing a schema independent of storage format
- This can be implemented for different backends (hdf5, zip, etc)
- Now two parts:
    - A Python dictionary that can be produced by tools (like boost-histogram)
    - A set of helpers to take that dictionary and write it to the supported formats
        - (Also the other direction)

Currently implemented formats

Hdf5, zip, JSON

- The uhi library contains 3 reader/writers
    - HDF5: structured, efficient, and can be stored with other data
    - Zip: metadata in JSON and values in binary files
    - JSON: handy for small histograms and testing
- ROOT may be added in the future
- New Zarr backend being worked on by Peter!
- A Parquet format has been discussed a bit, too

Example

JSON

```json
{
  "somehist": {
    "uhi_schema": 1,
    "writer_info": {
      "boost-histogram": {
        "version": "1.6.1",
      }
    },
    "axes": [
      {
        "type": "regular",
        "lower": 0,
        "upper": 5,
        "bins": 5,
        "underflow": true,
        "overflow": true,
        "circular": false
      }
    ],
    "storage": { "type": "double", "values": [1, 2, 3, 4, 5, 6, 7] }
  }
}
```

Core Design Choices

Across all backends

- Collection of named histograms
- **"storage"** contains data
- **"axes"** contains Axes objects
- **"metadata"** and **"writer_info"** on histogram and each axes (limited dicts)
- Current storages: **int**, **double**, **weight**, **mean**, **weighted_mean**
- **Dense** and **sparse** support (uhi provides helpers to convert)

Across backends

HDF5, Zip, JSON, (Zarr WIP)

- A new backend just maps this structure to native format
- Allowed to redirect data (like ZIP) or use native structure
- Example implementation inside UHI library, usable if tool produces the required structure
- Limits on what metadata is allowed (simple set of options in dict currently)
- Arbitrary tool info allowed, but not required to open file
- Uses the native tool interface to serialize / unserialize

Interface

Low level only so far

- Only low level interface implemented
- Designed to be natural and completely flexible for the backend
- Feel free to suggest high level ideas for Hist!

Example

JSON

```python
data = filename.read_text(encoding="utf-8")

# Read a JSON file
hist = json.loads(data, object_hook=uhi.io.json.object_hook)

# Save a JSON file
redata = json.dumps(hist, default=uhi.io.json.default)
```

Example

HDF5

```python
# Write to file
with h5py.File(tmp_file, "w") as h5_file:
    for name, hist in hists.items():
        uhi.io.hdf5.write(h5_file.create_group(name), hist)

# Read from file
with h5py.File(tmp_file, "r") as h5_file:
    rehists = {name: uhi.io.hdf5.read(h5_file[name]) for name in hists}
```

Example

ZIP

```
# Save to file
with zipfile.ZipFile(tmp_file, "w") as zip_file:
    for name, hist in hists.items():
        uhi.io.zip.write(zip_file, name, copy.deepcopy(hist))

# Read from file
with zipfile.ZipFile(tmp_file, "r") as zip_file:
    rehists = {name: uhi.io.zip.read(zip_file, name) for name in hists}
```

Versions

Supported in latest versions of libraries

boost-histogram 1.6.1

hist 2.9.0

uhi 1.0.0

uhi 1.0.0
documentation

Search

CONTENTS:

Indexing

Indexing+

Plotting

Serialization

Testing

Changelog

Serialization

Introduction

Histogram serialization has to cover a wide range of formats. As such, we describe a form for s
that covers the metadata structure as JSON-like, with a provided JSON-schema. The data (bi
variable edges) is stored out-of-band in a binary format based on what type of data file you ar
very small (primarily 1D) histograms, data is allowed inline as well.

The following formats are being targeted:

```
┌────────────┐ ┌───────┐ ┌──────────┐
│ ROOT (todo)│ │ HDF5  │ │ ZIP/JSON │
└────────────┘ └───────┘ └──────────┘
```

Other formats can be used as well, assuming they support out-of-band data and text attribute
the metadata. We are working on a Zarr backend in the near future.

Caveats

This structure was based heavily on boost-histogram, but it is intended to be general, and can
expanded in the future as needed. As such, the following limitations are required:

Implementing a new format

Like parquet?

- Try to make a 1:1 mapping of the intermediate format to the native format you target
- Customize as needed to be natural in the native format
- Make helpers in uhi for reading/writing
- Write up the docs and submit!
    - A new format should have some advantage

What about C++?

Boost::Histogram

- Should be implementable, even by an end-user - just follow the spec!
- Not clear where to put helpers though
  - Each backend format needs dependencies - HDF5, Zip, ROOT, etc.

Adding support in other libraries

Everyone using Python can use uhi's helpers!

- Simple input/output the intermediate helper format
- Then use UHI's implementation to support multiple storage formats!
- You can also implement yourself

https://uhi.readthedocs.io/en/latest/serialization.html

How to get involved

Try it, contribute

- Try it out
- Propose / work on a format
- C++ and ROOT are still to-do’s if you are interested
- We should get some examples into test data
- Maybe uproot-browser could gain support for some of these formats?