# Uhepp

## Universal high-energy physics plots

Albert-Ludwigs-Universität Freiburg

Frank Sauerburger

September 15, 2021

# Introduction

- **Why uhepp?**

→ Makes working with plots more efficient

- **Why uhepp.org?**

→ Make collaboration faster

- **This talk: How to use?**
  - Learn key concepts of the format
  - Walk through the creation and modification of plots
  - Typical use cases of uhepp.org
- Used in ATLAS $H \to \tau\tau$ analysis

  [ATLAS-CONF-2021-044]

{...} uhepp.json

uhepp

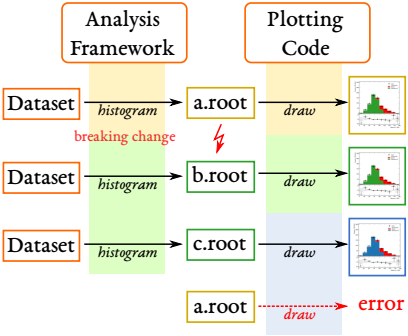
uheppHub

# Why uhepp?

## Workflow before uhepp

- Intermediate results stored as `TH1`
- Breaking change in analysis code (Add/rename sample, region, or systematics uncertainty scheme)
- Change old plot: colors, binning, composition, labels, ...
- Time scale of PhD thesis or analysis: years
- **Code diverges from intermediate results**

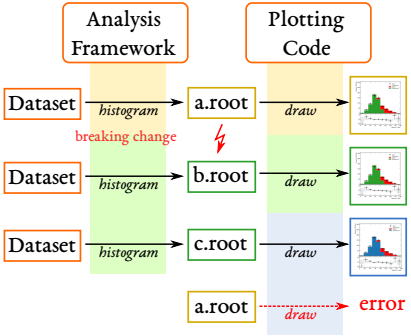
→ **Difficult to reuse, rework, or compare old results**



# Why uhepp?

## How to improve workflow

- Need new intermediate format
- Store initial histogram
- Store variations
- Store *How to draw them*
- Self-contained
- Self-descriptive
- Binning, stacks, colors, labels, ...



# Why uhepp?

## Where to tackle?



→ **Root files**

- Uhepp
- Sequence of MPL calls
  Sequence of ROOT `Draw()` calls
- Vector graphic

### Store root files

- o General purpose data storage
- \+ Raw data available
- \+ Editable
- – Not structured
- – Danger of code divergence
- – No/incomplete style information

# Why uhepp?

## Where to tackle?

- Root files
- Uhepp
- → **Sequence of MPL calls**
  **Sequence of ROOT `Draw()` calls**
- Vector graphic

### Sequence of MPL calls

- o General purpose plotting
- – Raw data not programmatically available
- o Manually editable
- + Complete style information

# Why uhepp?

## Where to tackle?

- Root files
- Uhepp
- Sequence of MPL calls
  Sequence of ROOT `Draw()` calls

→ **Vector graphic**

### Vector graphic

o General purpose drawing
– Raw data not available
– Limited edit capabilities
+ Complete style information

# Why uhepp?

## Where to tackle?

- Root files
- → **Uhepp**
- Sequence of MPL calls
  Sequence of ROOT `Draw()` calls
- Vector graphic

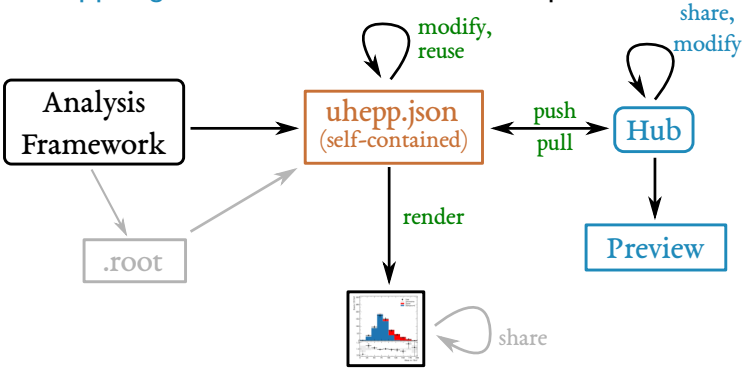
### Uhepp

- o Domain-specific format
- \+ Raw data available
- \+ Editable
- \+ Structured
- \+ Almost complete style information

# Why uhepp?

## New workflow

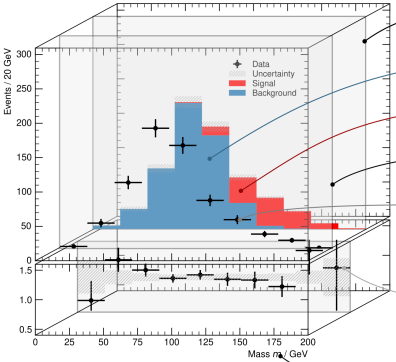
1. Storage format: Combines raw data + visual settings
2. Python package: Modify, render, push, pull
3. uhepp.org: Online hub with interactive preview



# Format and Python package

# Format and Python package

## Concepts




- Stack 0
  - list of items
  - stat error
  - stepfilled
  - StackItem 0
    - content
    - color
    - style
    - label
  - StackItem 1
    - content
    - color
    - style
    - label
- Stack 1
  - list of items
  - stat error
  - points
  - StackItem 0
    - content
    - color
    - style
    - label
- RatioItem
  - numerator
  - denominator
  - style
  - color
  - stat error
  - points
- Variable

Yields

# Format and Python package

## Minimal file structure



- Format different types of plots possible
- Now: only `histogram`
- Required fields: `version` and `type`
- Current version: 0.3
- Backwards compatible, extensible
- JSON schema
- Any format: JSON, YAML, ...

```
{
  "version": "0.3",
  "type": "histogram",
  ...
}
```

# Format and Python package

## Object initialization



- Create histogram object
- $x$-axis label and
- 5 bin edges $\Rightarrow$ 4 bins

```python
import uhepp as u
hist = u.UHeppHist(
    "m",
    [0, 50, 100, 150, 200]
)
```

```
"bin": {
  "edges": [0, 50, 100, 150, 200]
},
"variable": {
  "symbol": "m"
}
```

# Format and Python package

## Adding yields

- Store binned yields
- Total yield, independent of bin width
- Includes statistical uncertainty
- Each process/sample has own yield object
- Yield dict keys, arbitrary identifiers
- Yield identifiers used to reference yields in stacks
- **6 values** per process:
  - underflow
  - 4 bins
  - overflow

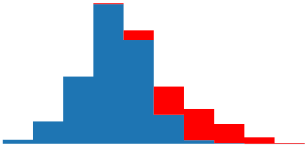
```
hist.yields["Z"] = u.Yield(
  [5, 4, 3, 2, 1, 0],
  [2, 2, 1, 1, 0, 0],
)
hist.yields["other"] =  ...
hist.yields["data"] =  ...
```

```
"yields": {
  "Z": {
    "base": [5, 4, 3, 2, 1, 0],
    "stat": [2, 2, 1, 1, 0, 0]
  },
  "other": {...},
  "data": {...}
}
```

# Format and Python package

## Adding Stack

- Add filled stack
- Two processes
- Stack item could merge multiple yields
- Labels used in legend
- Referenced by internal names from yield storage
  - Z
  - other

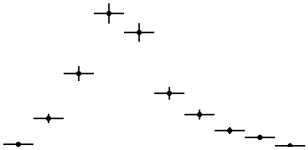


```
hist.stacks = [
  u.Stack([
    u.StackItem(["Z"], label="Z"),
    u.StackItem(["other"],
                label="Other bkg")
  ], bartype="stepfilled"),
  ...  # see json
]
```

# Format and Python package

## Adding Stack

- Add second stack
- Single processes, as points
- Labels used in legend
- Referenced by internal names from yield storage
  - `data`



```
"stacks": [
    ...,  # see python
    {
        "type": "points",
        "content": [{
            "yield": ["data"],
            "label": "Data"
        }]
    }
]
```

# Format and Python package

## Adding Ratio



- Add ratio panel item as points
- Numerator: `data`
- Denominator: sum of `Z` and `other`
- References to yield storage

```
hist.ratio.append(
    u.RatioItem(
        ["data"],
        ["other", "Z"],
        bartype="points"
    )
)
```

```
"ratio": [{
    "numerator": ["data"],
    "denominator": ["other", "Z"],
    "type": "points"
}]
```

# Format and Python package

## Styling



- Common styling options: color, line width or style
- Python interface accepts any MPL color
- JSON must be hex-encoded RGB
- Default color-cycle used if no color set

```
u.StackItem(
    ...,
    linestyle="--",
    linewidth=4,
    color="xkcd:blue"
)
```

```
    ...
    "content": [{
        ...,
        "style": {
            "linestyle": "--",
            "linewidth": 4,
            "color": "#0343df"
        }
    }]
    ...
```

# Format and Python package

## Metadata

- Standardize metadata: author, beam properties, code version, default filename
- Custom key-value pairs: “tags”
- Metadata used as documentation
- Metadata to search database (e.g. mongodb)
- (Soon: tag-based search at uhepp.org)

```
hist.filename = "mass_sr"
hist.Ecm_TeV = 13
hist.lumi_ifb = 139
...  # see json
```

```
"metadata": {
    ...  # see python
    "author": "Frank Sauerburger",
    "tags": {
        "region": "signal",
        "analysis": "Demo"
    }
}
```

# Format and Python package

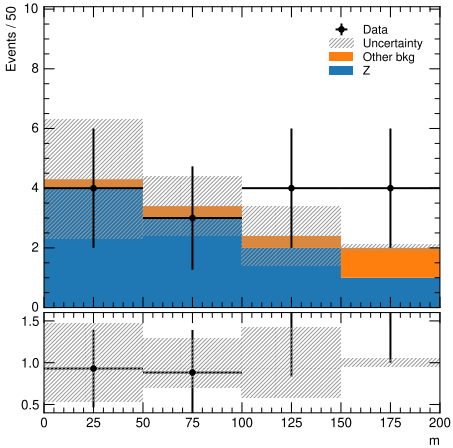
## Save, load, render

- Save python object to JSON (or YAML)
- Load python object from JSON (or YAML)
- (YAML has sensible git-diff)
- Show and render interactively
- Render to graphics file (matplotlib)

```
# Save to JSON or YAML
hist.to_json("demo.json")

# Render/show interactively
hist.render()
hist.show()

# Create graphics file
hist.render("demo.pdf")

# Restore saved histogram
hist2 = u.from_json("demo.json")
```

# Format and Python package

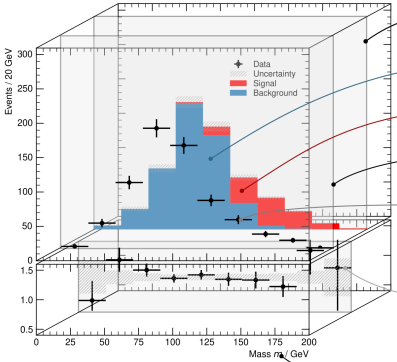
## Where are we?




- Plot with MC stack prediction
- Second stack with data measurement
- Both with statistical uncertainty
- MC / Data panel
- TODO: axis ranges, (colors?) and labels
- Histograms (yields) and presentation stored separately
- JSON
- Preview

# Format and Python package

## Recap: Concepts




- Stack 0
  - list of items
  - stat error
  - stepfilled
- StackItem 0
  - content
  - color
  - style
  - label
- StackItem 1
  - content
  - color
  - style
  - label
- Stack 1
  - list of items
  - stat error
  - points
- StackItem 0
  - content
  - color
  - style
  - label
- RatioItem
  - numerator
  - denominator
  - style
  - color
  - stat error
  - points
- Variable

Yields

# Format and Python package

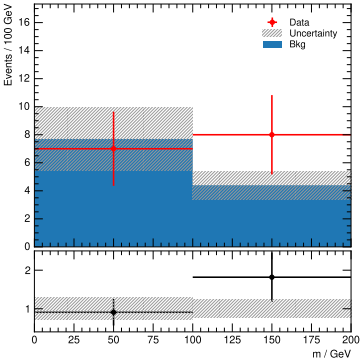
## Modifications

- Make binning coarser
- Change MC composition and label
- Change color
- Change ratio axis range
- Add $x$-axis unit
- All modifications are non-destructive
  Original Yield objects not modified

```
hist = u.from_json("demo.json")

# Draw alternative binning
hist.rebin_edges = [0, 100, 200]

# Merge into single background
hist.stacks[0].content = [
    u.StackItem(["other", "Z"],
                label="Bkg")
]

# Other
hist.unit = "GeV"
hist.ratio_max = 2.5

# Change data color
hist.stacks[1].content[0].color = "red"
```

# Format and Python package

## Modifications




- All modifications are non-destructive
  Original Yield objects not modified
- Can restore original binning
- Colors of main and ratio panel independent
- GeV added also to $y$-axis
- Uncertainties for samples and bins added in quadrature
  (i.e. assumes statistical independence)
- JSON
- Preview

# Embedded uncertainties



*class* uhepp.Yield(*base, stat=None, syst=None, var_up=None, var_down=None*)

Collection of yields and uncertainties of a single process

A yield object stores binned yields for a process including underflow and overflow bins. This

[docs]

- **Embed uncertainties** into uhepp files
- Archival: programmatic extraction from JSON file
- Size of uncertainty bands/bars
- Shown as envelops

`base` (list):
The yields per bin

`stat` (list):
Statistical uncertainty

`syst` (list):
Total, pre-computed systematic uncertainty

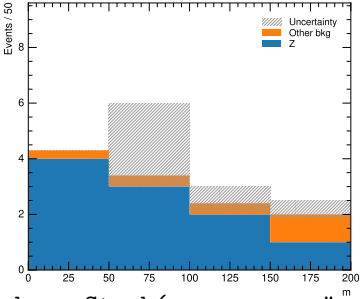
`var_up`, `var_down` (dict):
Systematic variations of base yield (name → list)

# Embedded uncertainties

## Application

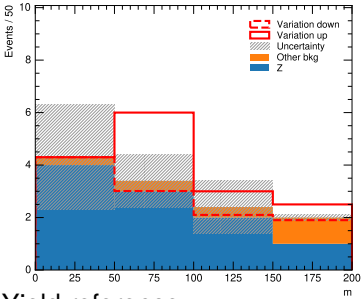


### Uncertainty band/bar




- `uhepp.Stack(..., error="env")`
- Or combiation of `stat`, `syst`, `env`
- E.g. `stat+syst`
- JSON, Preview

### Envelope




- Yield reference: `"process/variation_name/up"` or `"process/variation_name/down"`
- JSON, Preview

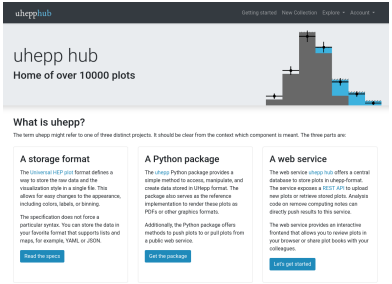
# uhepp hub

# Uhepp Hub



## Central hub to store and preview plots

- Interactive JS preview
- REST API to push and pull JSON
- Uses CERN Single sign-on
- Currently: Access restricted to ATLAS $\rightarrow$ Can be extended to CERN collaborations
- Plots organized into collections
- Owner can push, edit, delete
- Read access: All, collaboration, private
- Each plot has unique ID




uhepp.org

# Uhepp Hub

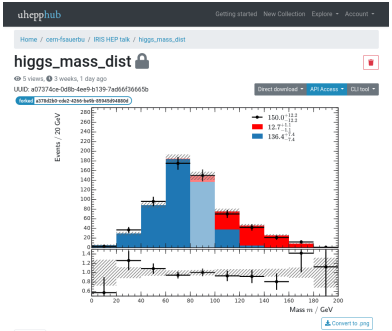
## Technology

### Backend tech stack

- Django app
- REST API with Django REST framework
- Postgresql database (with off-site backup)
- docker-compose deployment
  Soon: Migration to Kubernetes

### Frontend tech stack

- Customized bootstrap style
- React interactivity
- Plotting with VX (= React + D3)
- Latex rendering with MathJax




Preview

# Uhepp Hub

## REST API



### Python

```
In [1]: import uhepp as u

In [2]: hist = u.pull("a07374ce-0d8b-4ee9-b139-7ad66f36665b")

In [3]: hist.push(2)
Out[3]: https://uhepp.org/p/b9bcc6cb-3629-437d-81a7-92f1a4157e8f
```

### HTTP

```
GET /api/plots/a07374ce-0d8b-4ee9-b139-7ad66f36665b HTTP/1.1
Authorization: Token YOUR_ACCESS_TOKEN
...


POST /api/plots/a07374ce-0d8b-4ee9-b139-7ad66f36665b HTTP/1.1
Authorization: Token YOUR_ACCESS_TOKEN
Content-Type: application/json
...
```

- Pull existing plot with UUID
- Push uhepp object to collection
  here collection 2
- Credentials taken from env vars
  Create token at uhepp.org/tokens

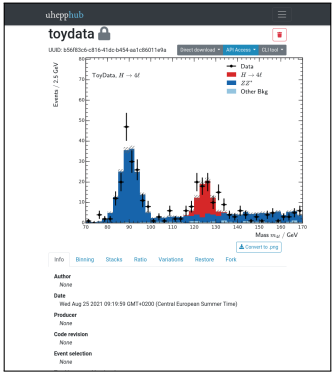
# Demos



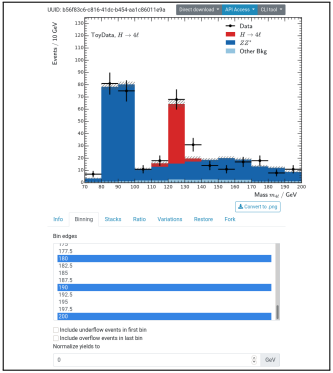
- Demo 1: Change binning and composition
- Demo 2: Investigate variations
- Demo 3: Ratio of samples
- Use toy plot that looks like $H \to 4\ell$
- Toy plot generated by Gaussian

# Demo 1

## Binning and style




Source

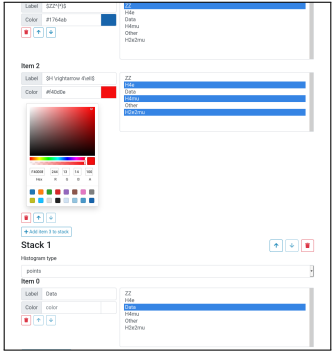



Adjust binning

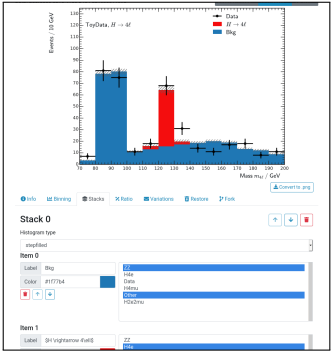
# Demo 1

## Binning and style






Change colors

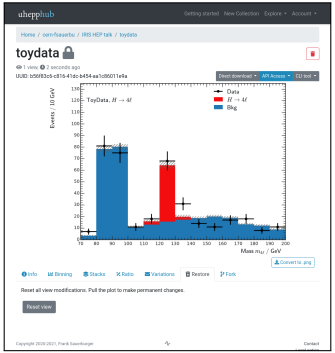



Merge backgrounds

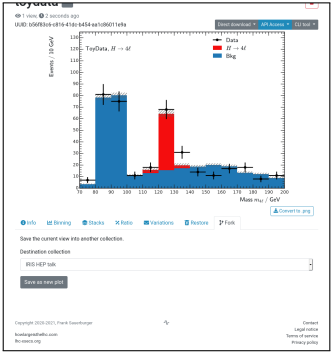
# Demo 1

## Binning and style






Changes are temporary in browser
$\rightarrow$ reset

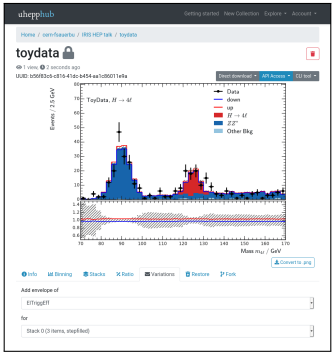



... or fork as new plot

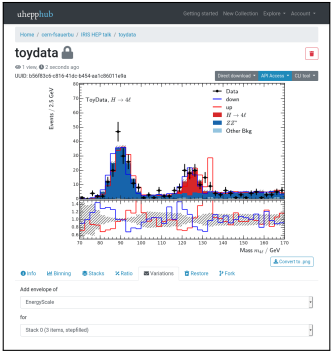
# Demo 2

## Systematic variations





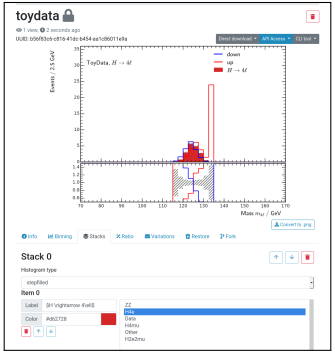

Variation overlay: El. trigger eff




Variation overlay: Energy scale
**Suspicious spike in signal**

# Demo 2

## Systematic variations



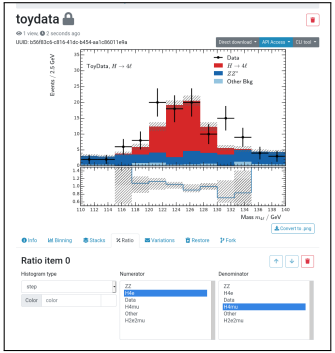

Decompose signal sample to investigate spike

- Deep-link to interactive envelope: Share with collaborators
- Quicker than navigating through classical plot books
- Classical plot books usually don't offer sample decomposition
- Here: Suspicious sample identified within seconds

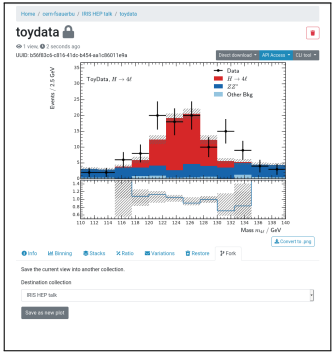
# Demo 3

## Ratio plot





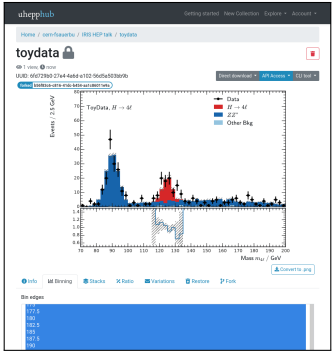

Add ratio item for two signal samples, Zoom in $x$-axis




Fork as new plot

# Demo 3

## Ratio plot




Original data unmodified
$\rightarrow$ Restore original binning

# What's more?

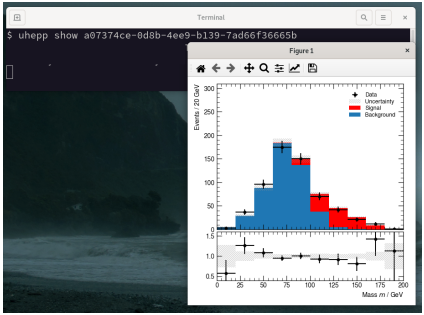
## Integration

- Horizontal and vertical lines
- Difference instead of ratio
- Graph overlay

**Integration**

- Python uhepp package: all features
- CLI: push, pull, render, show
- CAF: push
- Latex: pull
- Any language that handles JSON or HTTPS

**Example: CLI show**




Downloaded from hub, interactive MPL dialog

# What's more?

## Limitations and plans



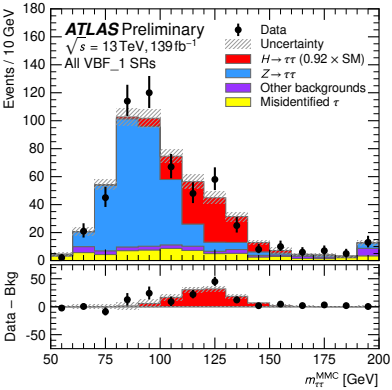
- No statistical uncertainties of variations in data model
- Currently only histogram-type plots
- Future: limit plots, NLL-scan plots, NP pull, NP rankings
- Metadata-based search at uhepp.org
- Thumbnail preview for plots in collections
- Interface for HEPData
- CERN e-group based permission model
- Open-source uhepp hub code

# Summary

# Summary

- uhepp domain-specific format
- Python library to handle
- uhepp.org to share+archive plots
- Format is self-contained:
  raw data + composition/style
- Open to new ideas and extensions
- Resources:
  - PyPi
  - Read the docs
  - Python repo
  - uhepp.org
  - JSON schema

**Real-life example**




[ATLAS-CONF-2021-044]
(ATLAS internal: preview)