pyhf: standalone HistFactory

K Cranmer, M Feickert **L Heinrich**, G Stark
4th Reinterpretation Workshop

May 15th 2018

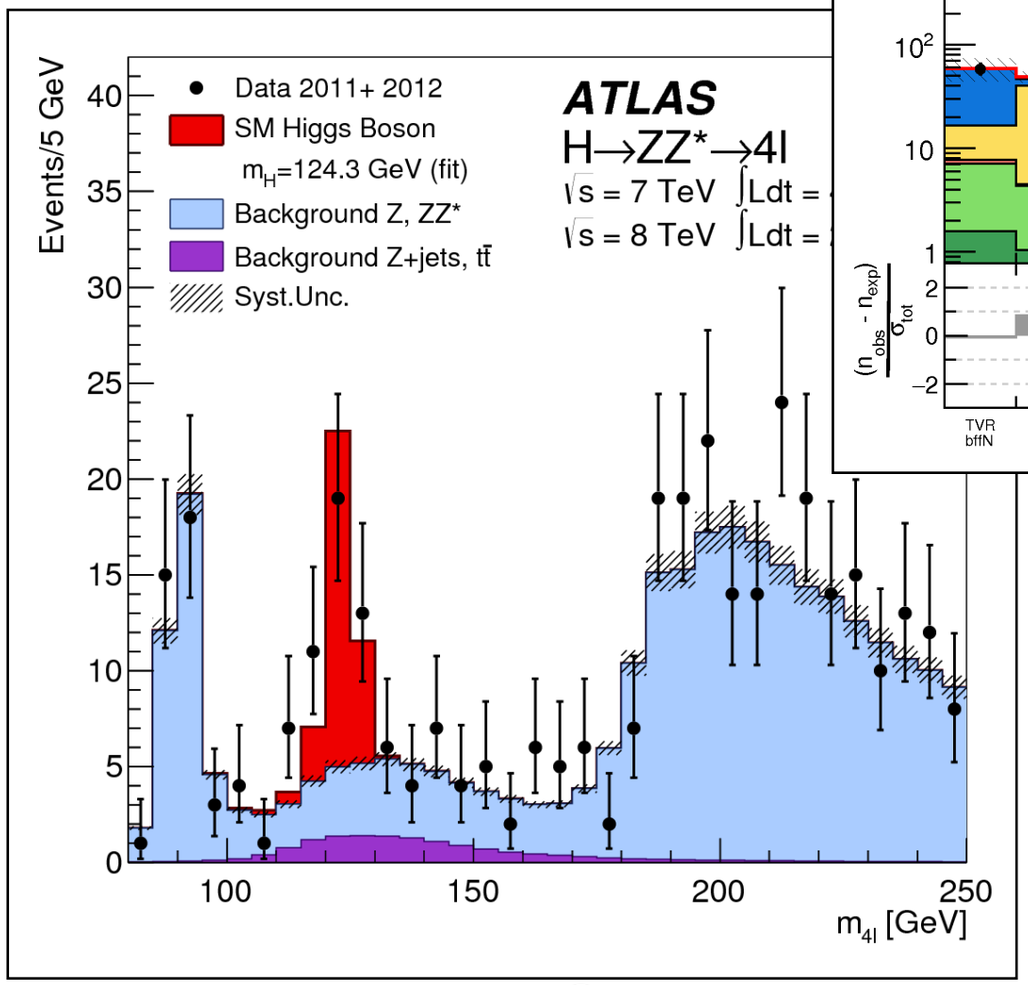
NEW YORK UNIVERSITY

/lukasheinrich

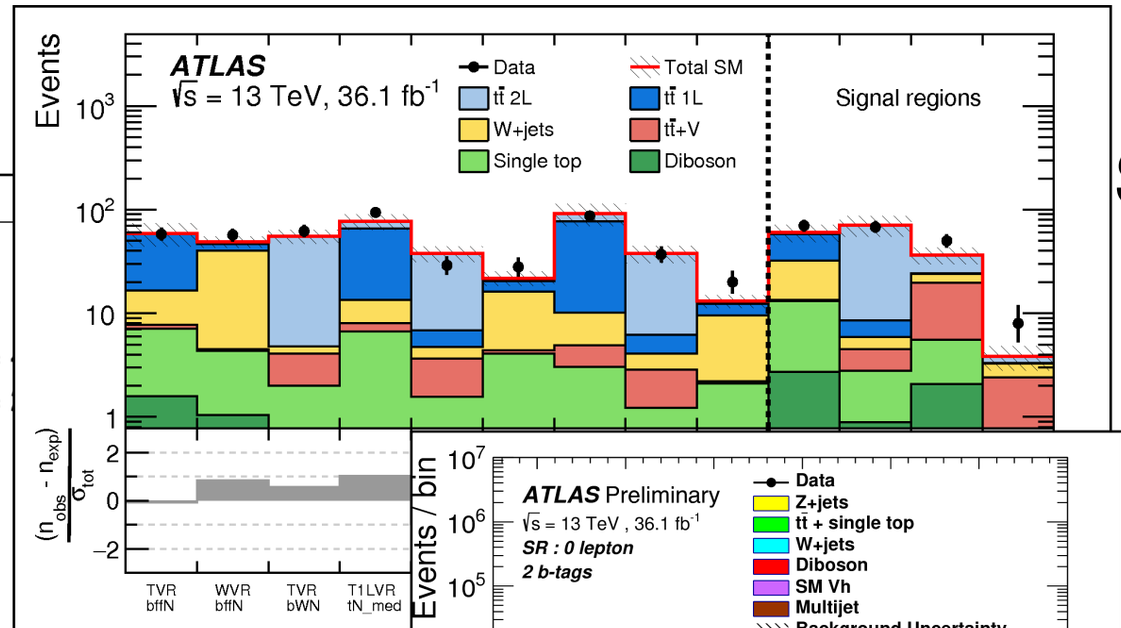
HistFactory

fundamentally a (quite flexible) p.d.f template to build statistical models from binned distributions and data.

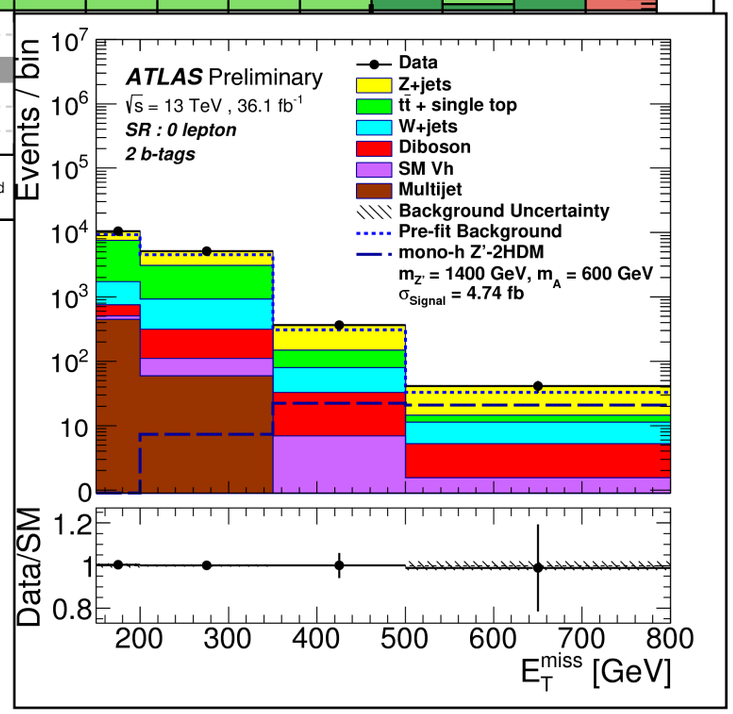
Widely used in Standard Model measurements and BSM searches (in ATLAS it's the lingua franca of binned models.



SM



SUSY



Exotics

HistFactory — The Template

$$\mathcal{P}(n_c, x_e, a_p \mid \phi_p, \alpha_p, \gamma_b) = \prod_{c \in \text{channels}} \left[\text{Pois}(n_c|\nu_c) \prod_{e=1}^{n_c} f_c(x_e|\boldsymbol{\alpha}) \right] \cdot G(L_0|\lambda, \Delta_L) \cdot \prod_{p \in \mathbb{S}+\boldsymbol{\Gamma}} f_p(a_p|\alpha_p)$$

Scenario: multiple disjoint *channels* (or regions) of binned distribution with multiple *samples* contributing to each with additional (possibly shared) systematics between sample estimates. Applies to many scenarios in HEP.

Two main pieces:

- **Poisson pdf for bins observed in all channels**
- **Constraint pdf (+ data) for "auxiliary measurements" — encoding systematic uncertainties (normalization, shape, etc)**

HistFactory — The Template

$$\mathcal{P}(n_c, x_e, a_p \mid \phi_p, \alpha_p, \gamma_b) = \prod_{c \in \text{channels}} \left[\text{Pois}(n_c|\nu_c) \prod_{e=1}^{n_c} f_c(x_e|\boldsymbol{\alpha}) \right] \cdot G(L_0|\lambda, \Delta_L) \cdot \prod_{p \in \mathbb{S}+\boldsymbol{\Gamma}} f_p(a_p|\alpha_p)$$

It's only math, **but** for now HistFactory has been tightly linked to the ROOT ecosystem, since the *only implementation* of the template is available in RooStats + RooFit.

- hard to quickly start using HF pdfs without having to learn ROOT / RooFit / RooStats
- possible scaling issues for large models (both I/O and Memory)
- hard to plug in modern tools for minimization, computation of the pdf
- data to build the likelihood stored in binary ROOT format — not ideal for long-term preservation such as on HepData

HistFactory

HistFactory spec + data

XML

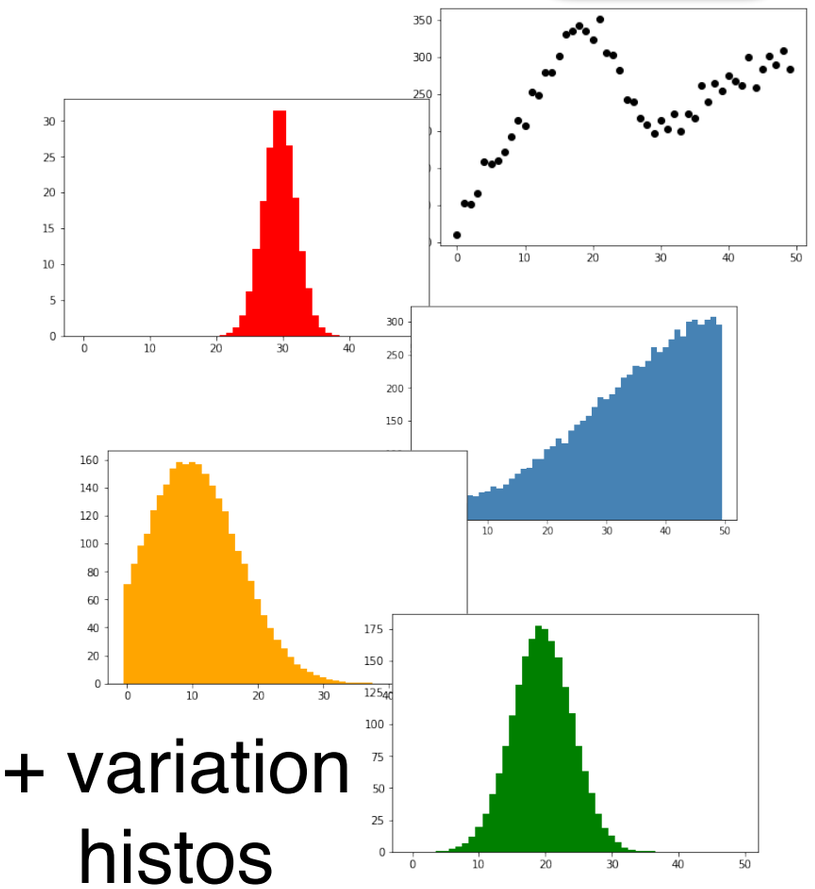
XML

XML

file1.root

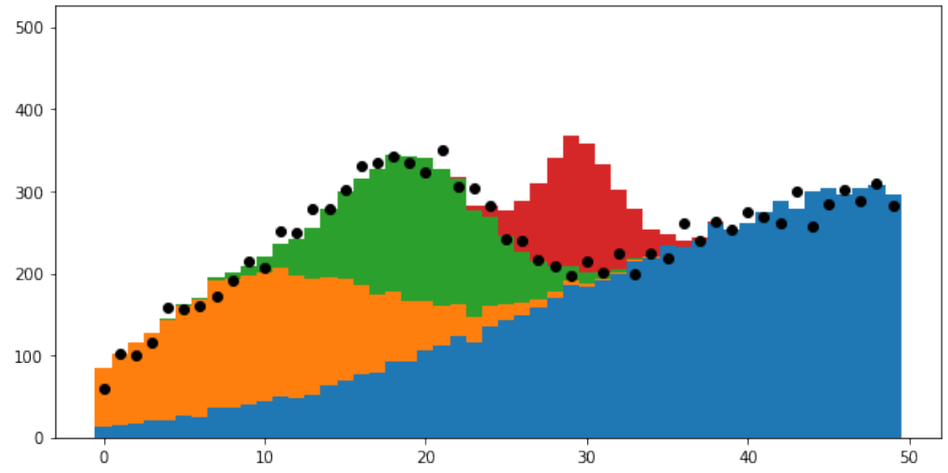
file2.root

file3.root



+ variation histos

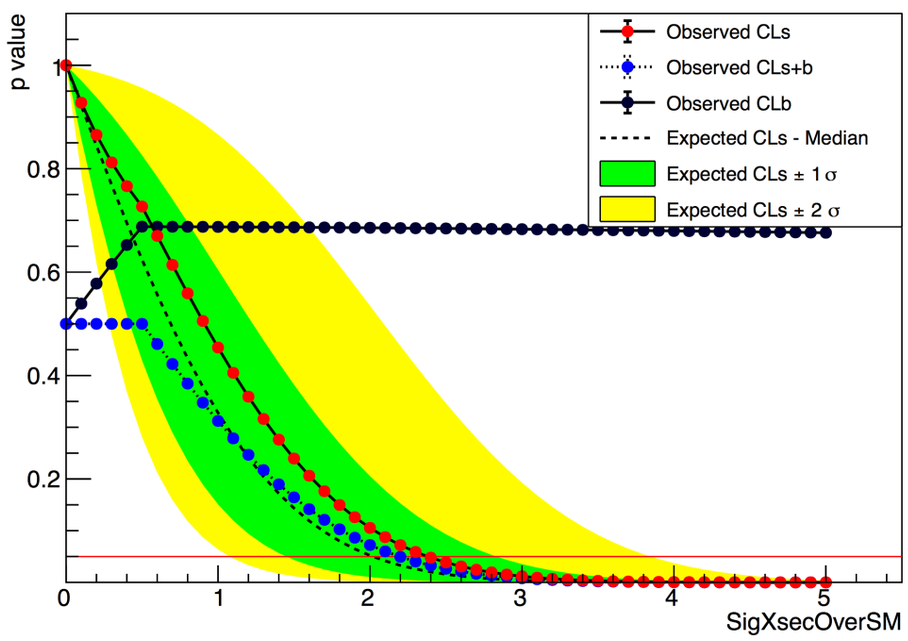
build L'hood/pdf



Likelihood (RooFit Workspace)

interval estimation

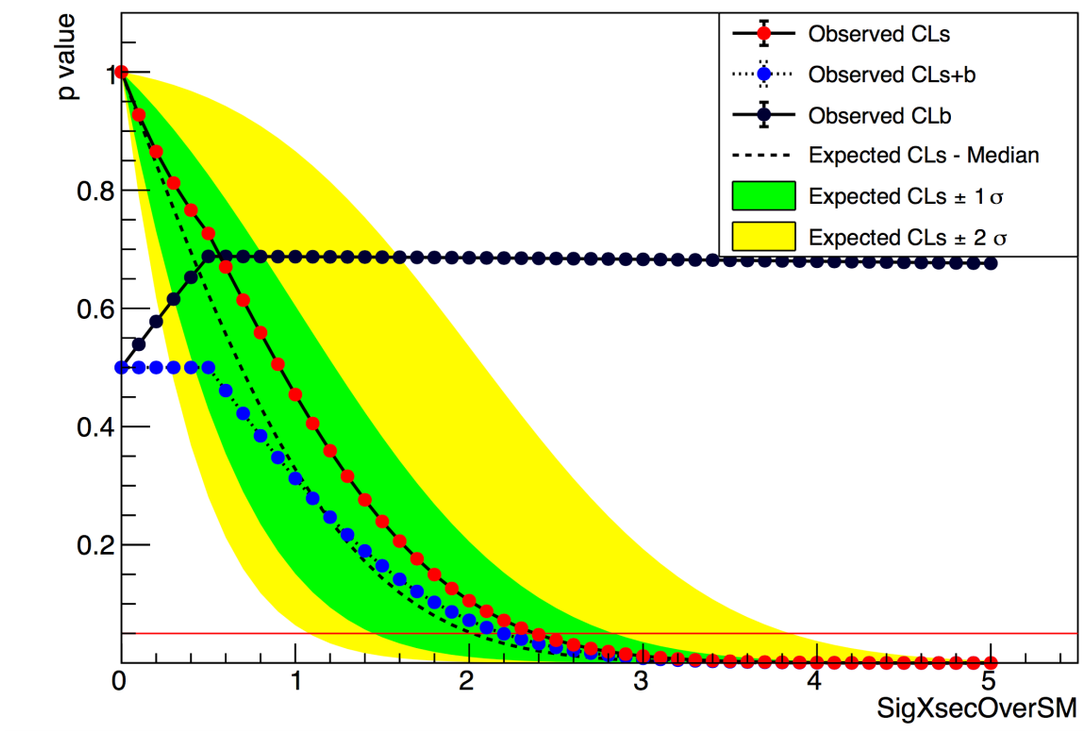
plot



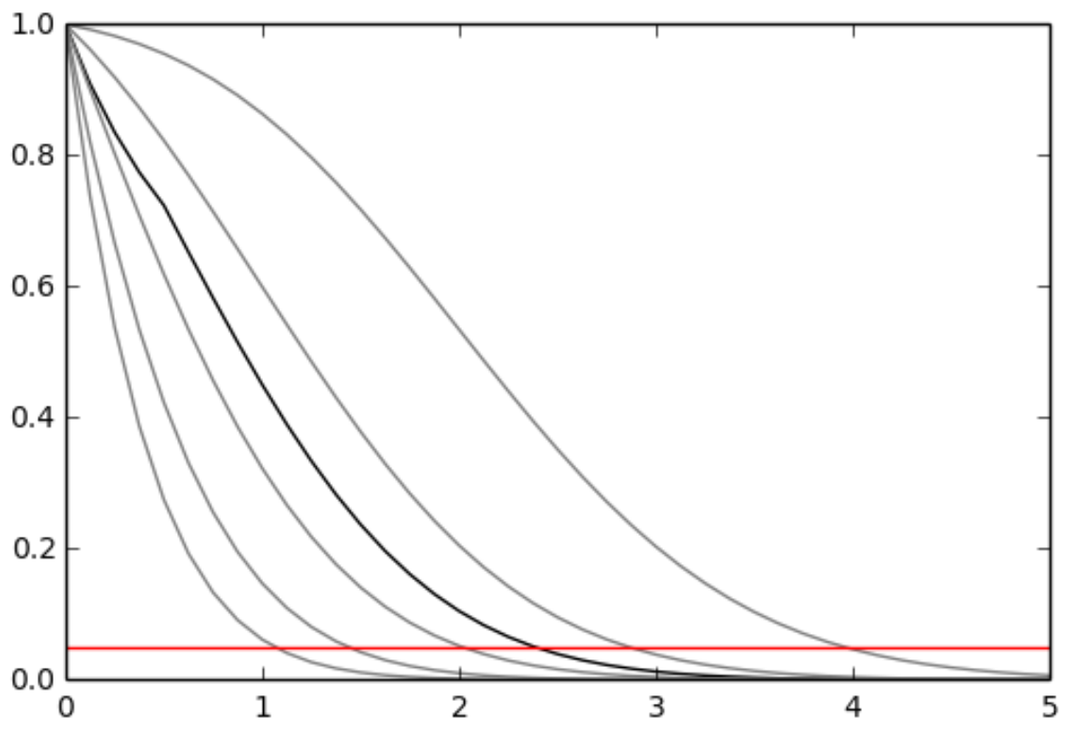
pyhf provides two things

- a standalone pure-python implementation, including hooks into modern deep-learning, autodifferentiable tensor libraries
  - implementation of asymptotic interval estimation algorithm based on profile likelihood test-statistic
- a pure JSON schema to distribute and archive HistFactory models ingredients without any reliance on binary formats

plot



ROOT

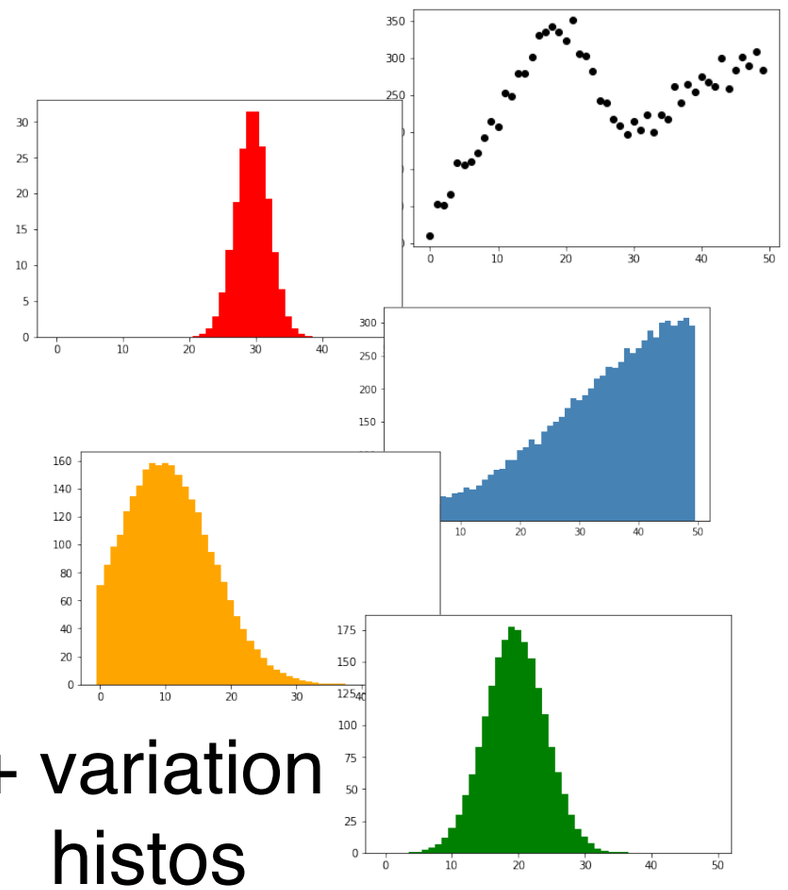


pyhf

HistFactory

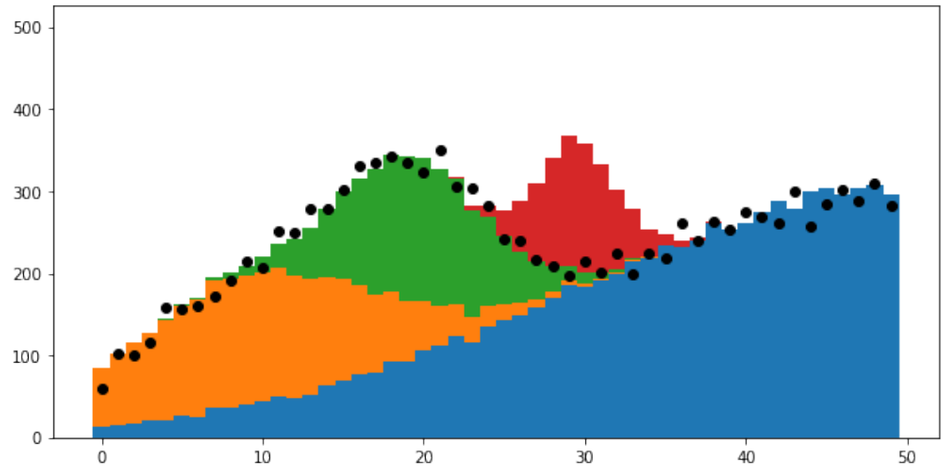
HistFactory spec + data

Single JSON File



+ variation histos

build L'hood/pdf

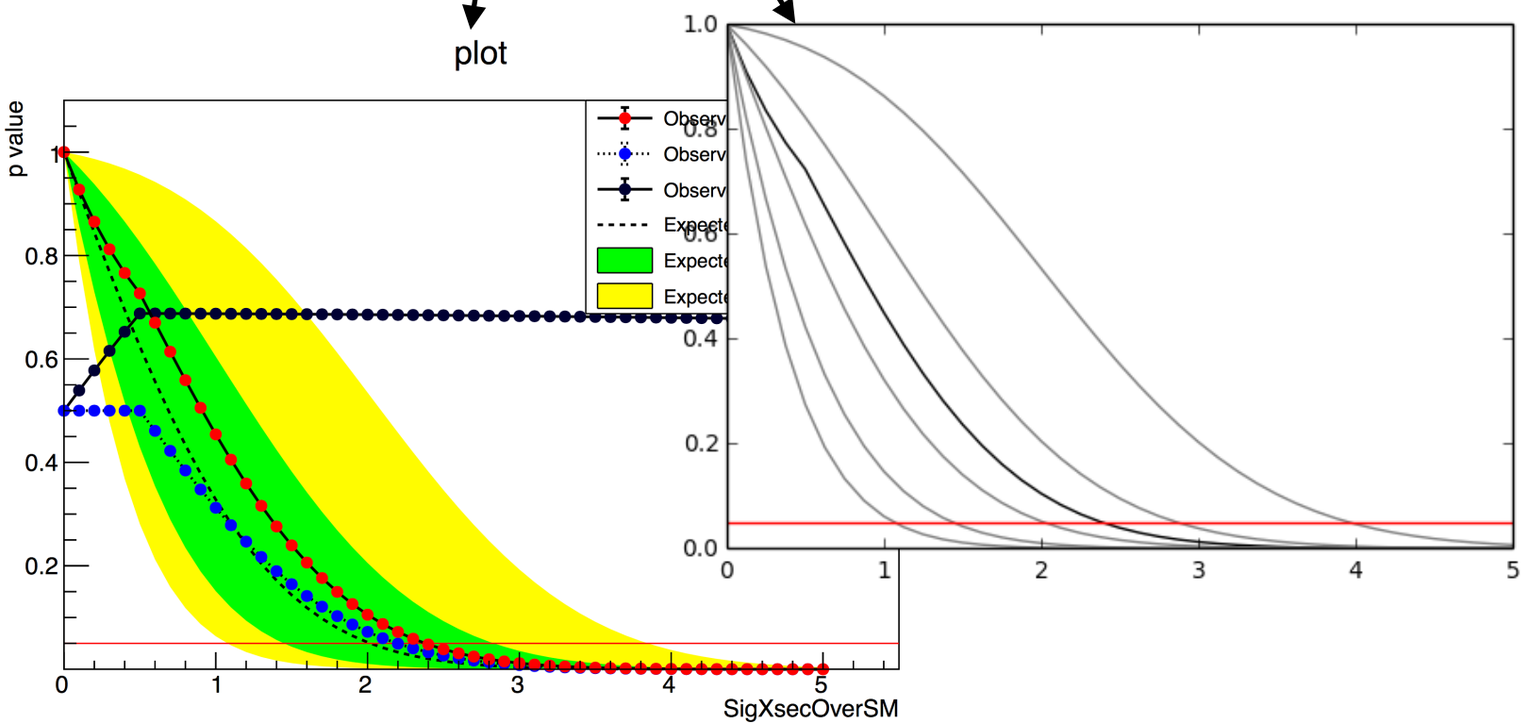


Likelihood (RooFit Workspace or python pdf implementation)

ROOT HistFactory

pyhf

plot



pyhf provides two things

- a standalone pure-python implementation, including hooks into modern deep-learning, autodifferentiable tensor libraries
- implementation of asymptotic interval estimation algorithm based on profile likelihood test-statistic
- a pure JSON schema to distribute and archive HistFactory likelihoods without any reliance on binary formats (think: HepData)

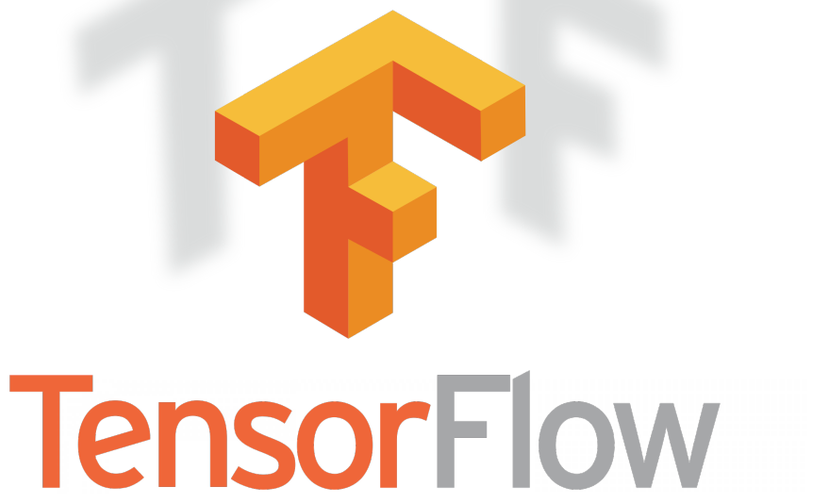
When to use pyhf:

- construct new models: just want to quickly calculate CLs for some background, signal and data? standard python + 2 lines in pyhf
- manipulate full featured hep-ex models outside of ROOT: e.g. for reinterpretation: **take existing model, swap out signal, re-fit.**

pyhf with auto-differentiable tensor backends

pyhf implements all numeric operations through a thin layer of of abstract n-D array operations

allows us to transparently switch out numeric backend of pyhf



numeric backends popular in Deep Learning allow us to compute exact gradients when minimizing likelihood

Easy to perform stat. analysis on GPUs (good for large models)

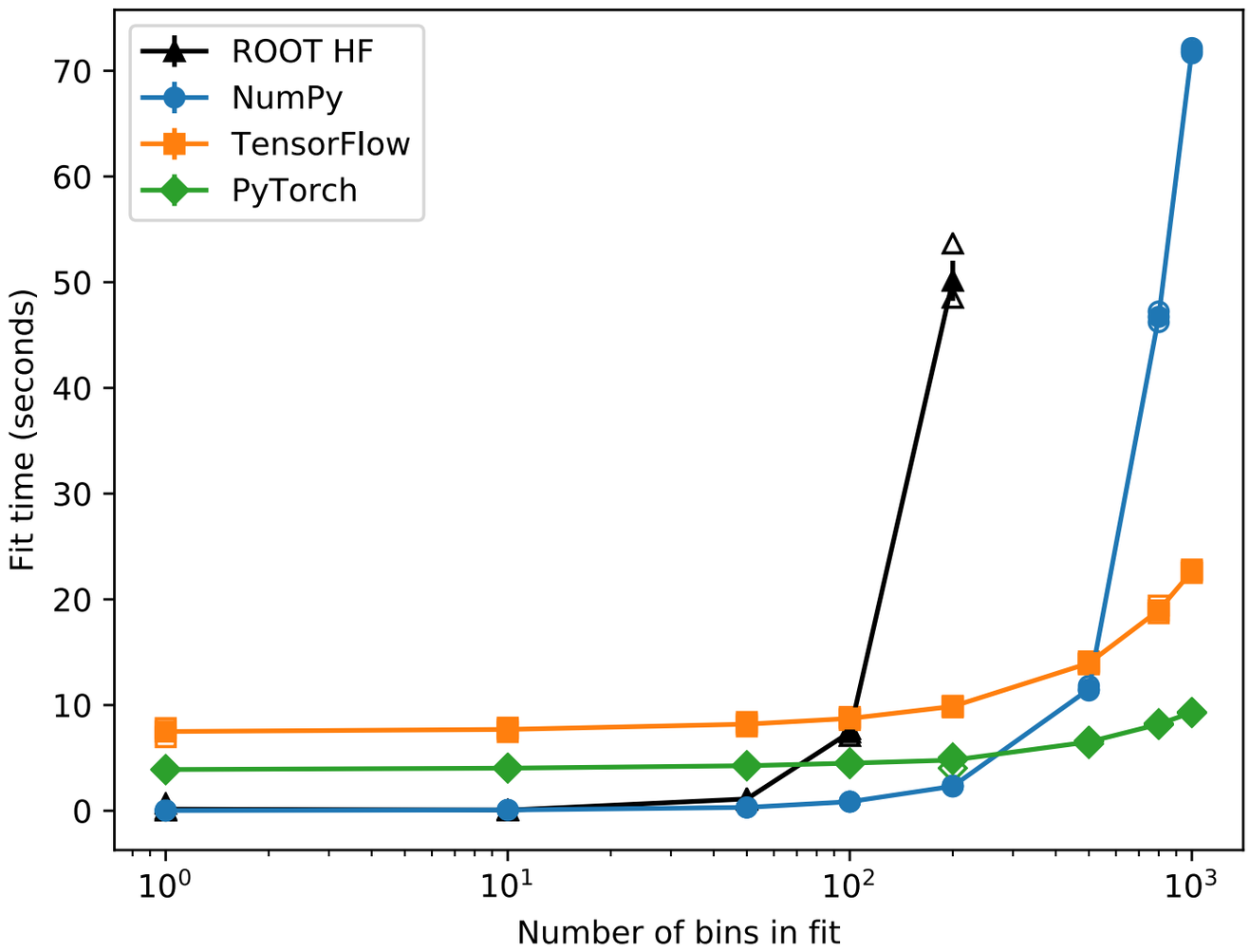
(**Automatic differentiation**: for a given algorithm, convert into basic operations for which exact gradients are known, propagate gradients through entire algorithm)

```
f(a, b):                    f'(a, a', b, b'):
    c = a * b                   (c, c') = (a*b, a'*b + a*b')
    d = sin c          ⟶        (d, d') = (sin c, c' * cos c)
    return d                    return (d, d')
```

pyhf with auto-differentiable tensor backends

Benchmark: single channel with many bins and uncorrelated bin-wise uncertainties

For many channels, ROOT is still faster (→ investigating)



What does it support

Implemented variations:

- [x] HistoSys
- [x] OverallSys
- [x] ShapeSys
- [x] NormFactor
- [x] Multiple Channels
- [x] Import from XML + ROOT via `upr`
- [x] ShapeFactor
- [x] StatError

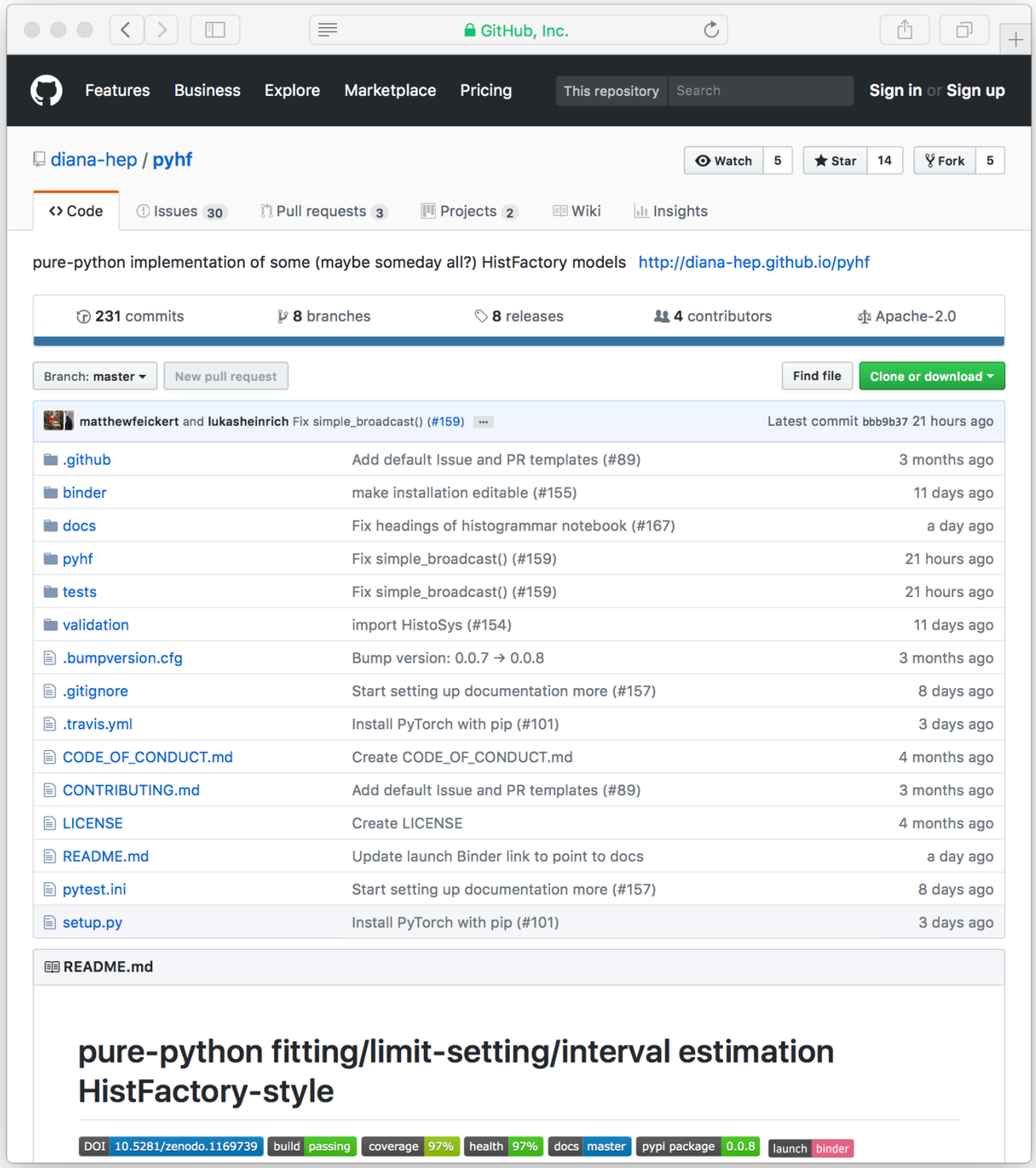
Computational Backends:

- [x] NumPy
- [x] PyTorch
- [x] TensorFlow
- [x] MXNet

Todo

- [ ] Lumi Uncertainty
- [ ] StatConfig
- [ ] Non-asymptotic calculators

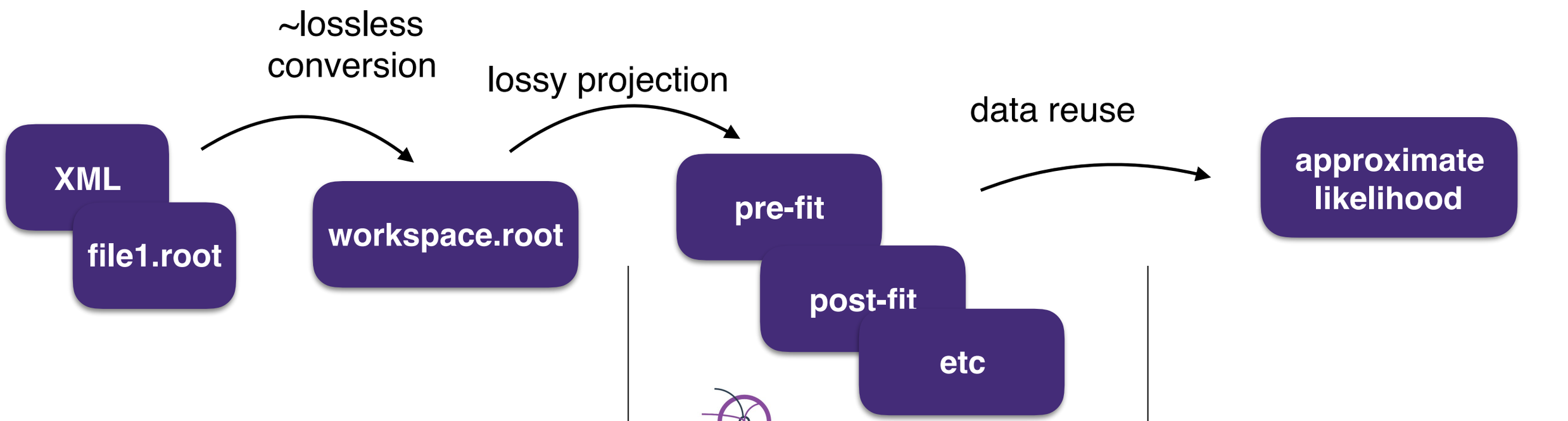
developed on GitHub — everyone is welcome to join. https://github.com/diana-hep/pyhf



HistFactory-JSON as an archive product on HepData

Current path of data to HepData for a BSM search from creation to usage:

- produce XML and ROOT of the analysis and create workspace
- decide what distributions at which values of parameters (pre-fit, post-fit) etc make sense
- use scripts to produce HepData YAML/JSON
- process has many steps and is lossy, we are not archiving the best information we have.

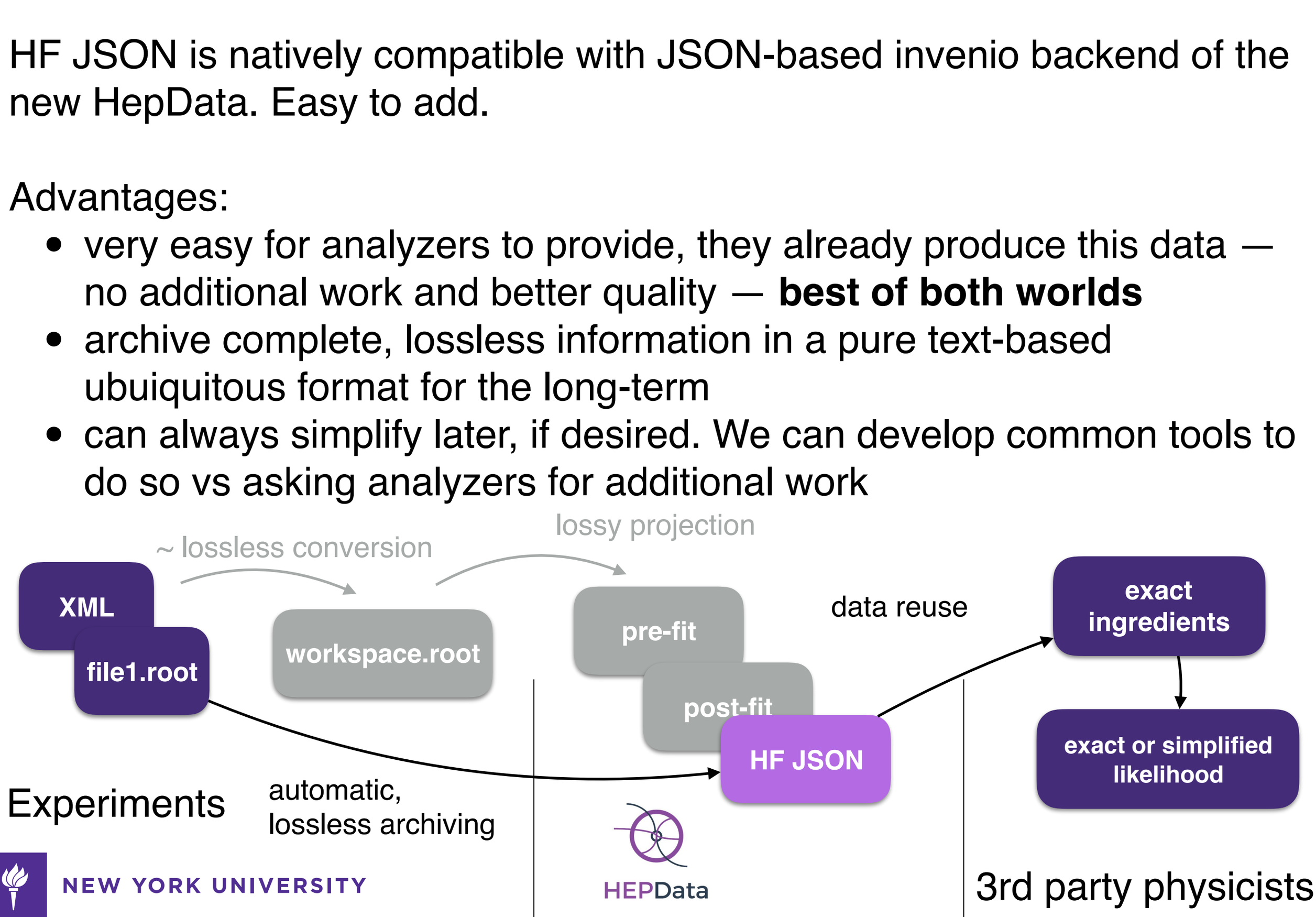


HistFactory-JSON as an archive product on HepData

HF JSON is natively compatible with JSON-based invenio backend of the new HepData. Easy to add.

Advantages:

- very easy for analyzers to provide, they already produce this data — no additional work and better quality — **best of both worlds**
- archive complete, lossless information in a pure text-based ubuiquitous format for the long-term
- can always simplify later, if desired. We can develop common tools to do so vs asking analyzers for additional work



Expressing the full stat. model in industry standard formats allows us to tap into a wide range of industry tools to handle these objects.

Example: **a standard interchange format of likelihood patches for reinterpretations of HistFactory-based analyses.**

JSONPatch is an industry standard (RFC 6902) to patch JSON documents.

Analysis implementations (Rivet, CheckMate, etc) only need to write simple JSON patch for the new signal distribution and then use common fitting tools

```
import requests
import jsonpatch
import pyhf

data = requests.get('http://hepdata.net/ins1234/json')

#replace the 2-bin signal
patch = jsonpatch.JsonPatch([
    {'op': 'replace', 'path': '/channels/0/samples/0/data', 'value': [20.,10.]},
])

newpdf = pyhf.hfpdf(patch.apply(data['histfactory']))
```

NEW YORK UNIVERSITY

Conclusion

- first non-ROOT implementation of HistFactory p.d.f. template
  - very standard python + numpy + scipy stack
  - can run on GPUs / deep learning frameworks w/ autograd
  - easy to do basic things (single-bin countring expt), possible to do full-fledged ATLAS likelihoods

- JSON spec to describe statistical model in a single, text-based file. (incl. tools to convert XML + ROOT into JSON)
  - ideal format to store in HepData
  - very robust, will be supported for a long time