zfit
scalable pythonic fitting

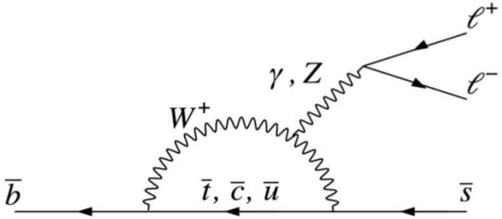
Scikit HEP

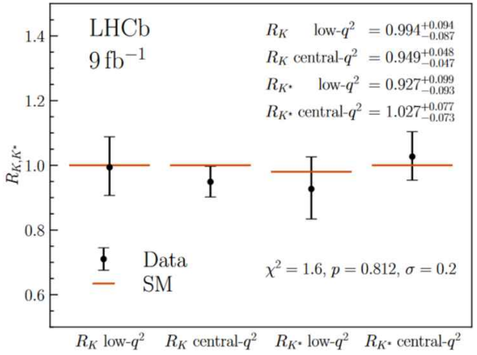
Model fitting in Python with zfit and Scikit-HEP

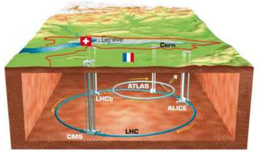
Jonas Eschle
jonas.eschle@cern.ch

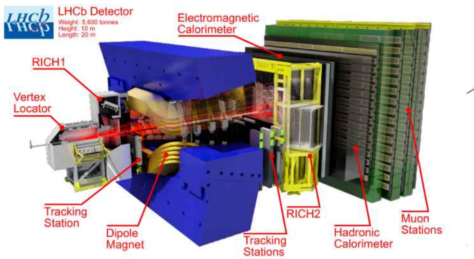
Syracuse University

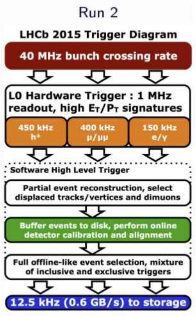
HEP Analysis

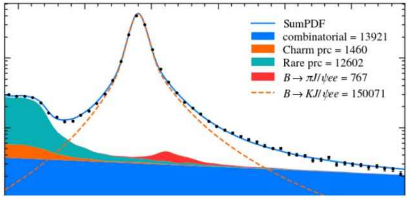




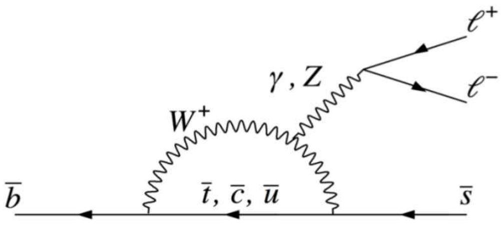




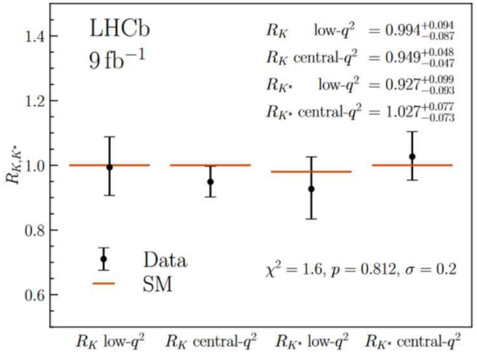


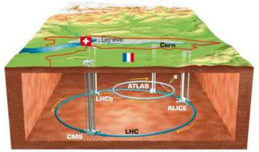


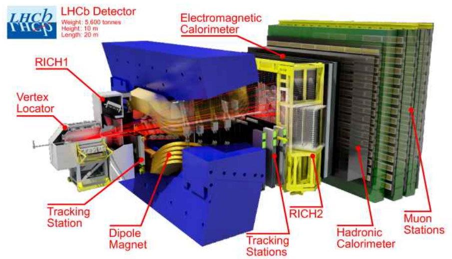
HEP Analysis



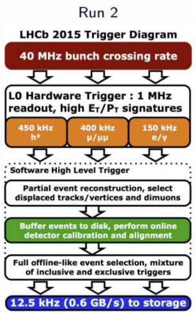
Lots of code

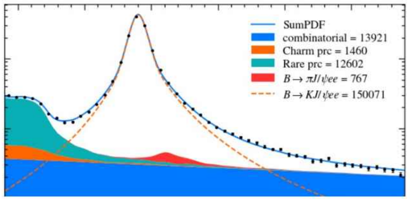






Lots of code

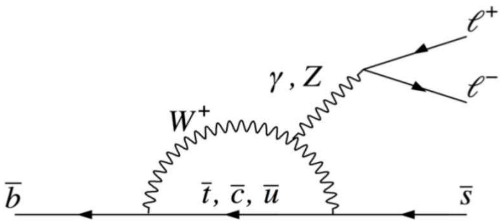




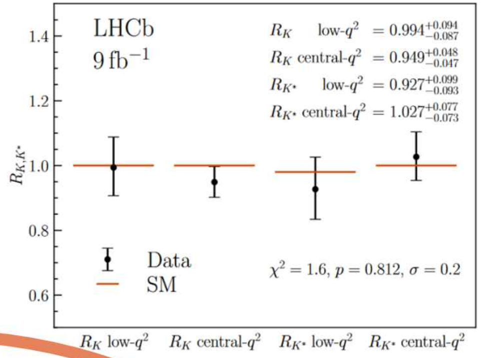
Lots of code

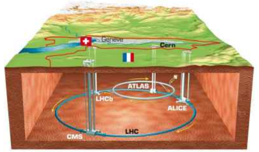
Lots of code

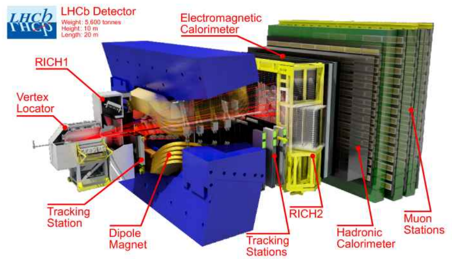
HEP Analysis



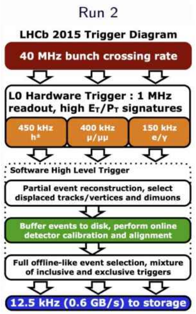
Lots of code



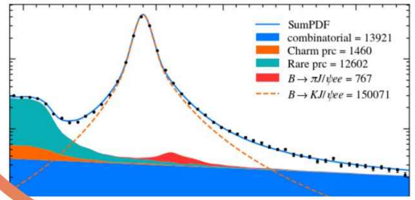




Lots of code



Lots of code



Lots of code

End-user analysis

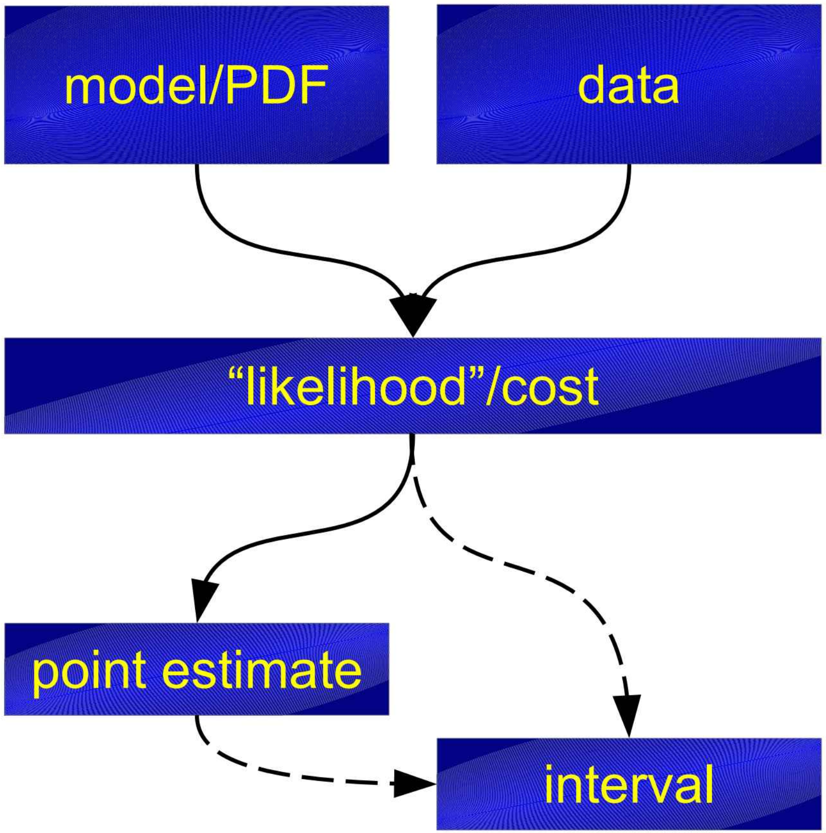
Focus on «fitting»

«Fitting»

«statistical inference»

Variety of possibilities

Focus on likelihood based
(as is very common in HEP)



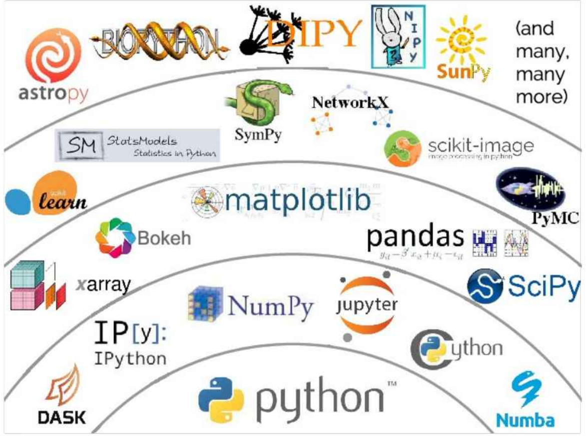
Historic landscape

<u>Analyses transition from C++ to Python</u>

Many, non-monolithic packages

Talk by Eduardo

Philosophy: extend/build on existing ecosystem



2018

Historic landscape

Analyses transition from C++ to Python
Many, non-monolithic packages

Talk by Eduardo

HEP fitting libraries still in C++
Strong libraries, but mediocre bindings, not «pythonic»

2018

Historic landscape

<u>Analyses transition from C++ to Python</u>

Many, non-monolithic packages

Talk by Eduardo

<u>HEP fitting libraries still in C++</u>

Strong libraries, but mediocre bindings, not «pythonic»

Why even move to Python?

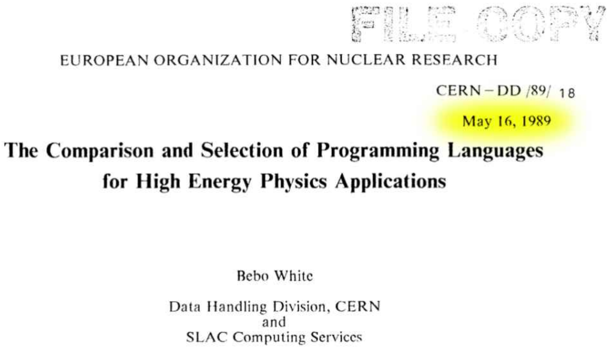
2018

Historic landscape

<u>Analyses transition from C++ to Python</u>
Many, non-monolithic packages

Talk by Eduardo

<u>HEP fitting libraries still in C++</u>
Strong libraries, but mediocre bindings, not «pythonic»


FILE COPY

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN – DD /89/ 18

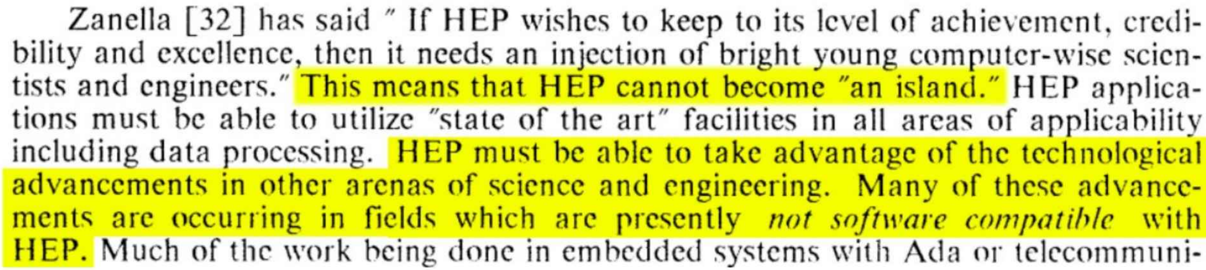
May 16, 1989

The Comparison and Selection of Programming Languages for High Energy Physics Applications

Bebo White

Data Handling Division, CERN
and
SLAC Computing Services

10.1016/0010-4655(89)90283-X


Zanella [32] has said " If HEP wishes to keep to its level of achievement, credibility and excellence, then it needs an injection of bright young computer-wise scientists and engineers." This means that HEP cannot become "an island." HEP applications must be able to utilize "state of the art" facilities in all areas of applicability including data processing. HEP must be able to take advantage of the technological advancements in other arenas of science and engineering. Many of these advancements are occurring in fields which are presently *not software compatible* with HEP. Much of the work being done in embedded systems with Ada or telecommuni-

2018

Historic landscape

Analyses transition from C++ to Python
Many, non-monolithic packages

Talk by Eduardo

HEP fitting libraries still in C++
Strong libraries, but mediocre bindings, not «pythonic»

Python packages
- no advanced features
- «*Python too slow*»

TensorProb
TensorFlow Analysis
probfit

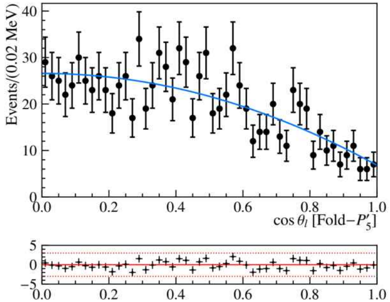
RooFit
mlfit
carl
scipy
TensorFlow Probability

HEP Python

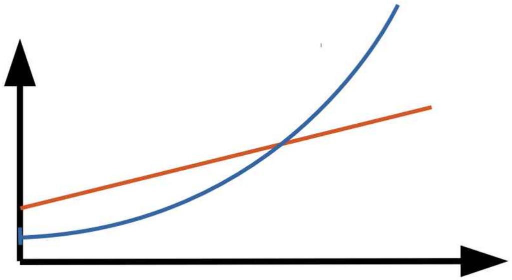
Non-HEP

2018

HEP Model Fitting in Python



HEP
advanced features, simply extendable



Scalable
large data, complex models

Pythonic
integrate into ecosystem, stable API

History

5 years ago

first presentation HSF/WLCG Workshop

History

5 years ago

first presentation HSF/WLCG Workshop

Summary

zfit

scalable pythonic fitting

- Beta stage, usable! (already used in LHCb analyses)
  - Not feature complete, but API stabilizing
- Contributions in form of *feedback and criticism* very welcome
  - API, use-cases, bugs,…
  - Any crazy idea!

It's about a reliable library

not about

«we need to get it working fast»

Different kind of fits

- Binned (*vs histfactory*) vs unbinned
  - Refers to data, cost/loss/likelihood and PDF
  - Unbinned data: product of probabilities
  - Binned data: «counting experiments»

Different kind of fits

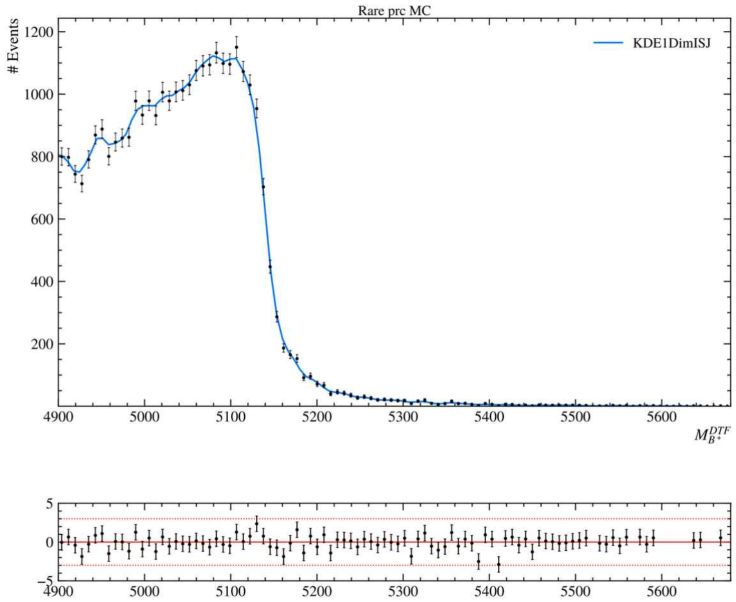
- Binned (*vs histfactory*) vs unbinned
  - Refers to data, cost/loss/likelihood and PDF
  - Unbinned data: product of probabilities
  - Binned data: «counting experiments»
- Template vs analytic
  - Shape from (simulation) sample vs closed-form function

Different kind of fits

- Binned (*vs histfactory*) vs unbinned
  - Refers to data, cost/loss/likelihood and PDF
  - Unbinned data: product of probabilities
  - Binned data: «counting experiments»
- Template vs analytic
  - Shape from (simulation) sample vs closed-form function
- *Analytical vs numerical normalization*
  - *Bin or closed-form integral vs numerical*

Different kind of fits

- Binned (*vs histfactory*) vs unbinned
  - Refers to data, cost/loss/likelihood and PDF
  - Unbinned data: product of probabilities
  - Binned data: «counting experiments»
- Template vs analytic
  - Shape from (simulation) sample vs closed-form function
- *Analytical* vs *numerical normalization*
  - *Bin or closed-form integral vs numerical*



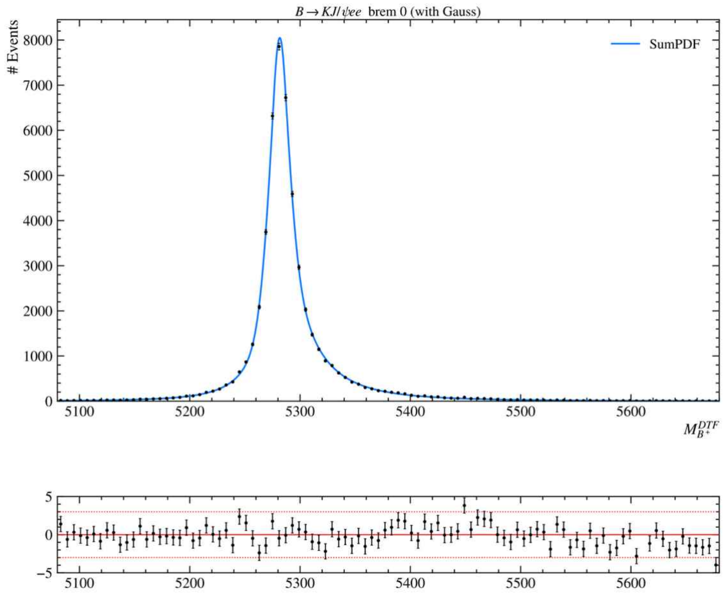
KDE

Gaussian kernel → analytic norm

ISJ → numeric norm

Different kind of fits

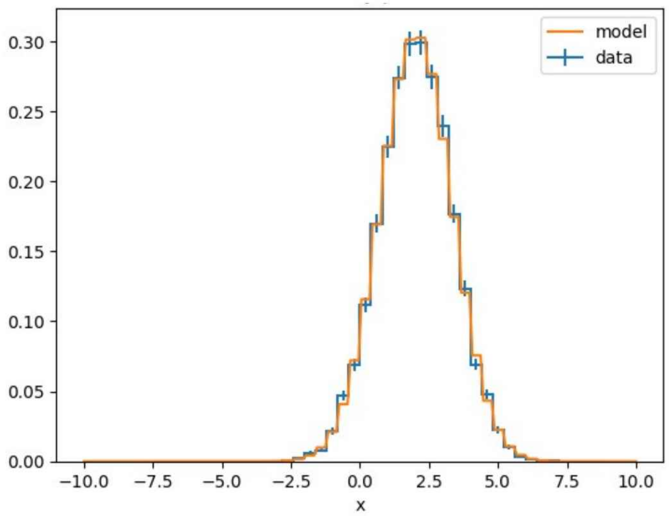
- Binned (*vs histfactory*) vs unbinned
  - Refers to data, cost/loss/likelihood and PDF
  - Unbinned data: product of probabilities
  - Binned data: «counting experiments»
- Template vs analytic
  - Shape from (simulation) sample vs closed-form function
- *Analytical vs numerical normalization*
  - *Bin or closed-form integral vs numerical*



Double Crystalball

Different kind of fits

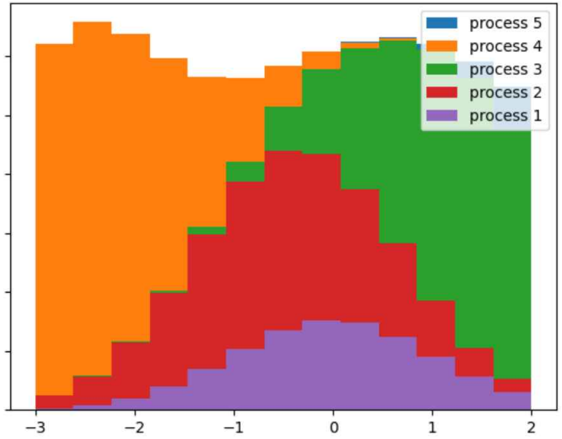
- Binned (*vs histfactory*) vs unbinned
  - Refers to data, cost/loss/likelihood and PDF
  - Unbinned data: product of probabilities
  - Binned data: «counting experiments»
- Template vs analytic
  - Shape from (simulation) sample vs closed-form function
- *Analytical vs numerical normalization*
  - *Bin or closed-form integral vs numerical*



(binned) Gaussian fit to histogram

Different kind of fits

- Binned (*vs histfactory*) vs unbinned
  - Refers to data, cost/loss/likelihood and PDF
  - Unbinned data: product of probabilities
  - Binned data: «counting experiments»
- Template vs analytic
  - Shape from (simulation) sample vs closed-form function
- *Analytical vs numerical normalization*
  - *Bin or closed-form integral vs numerical*



Stacked histograms PDFs

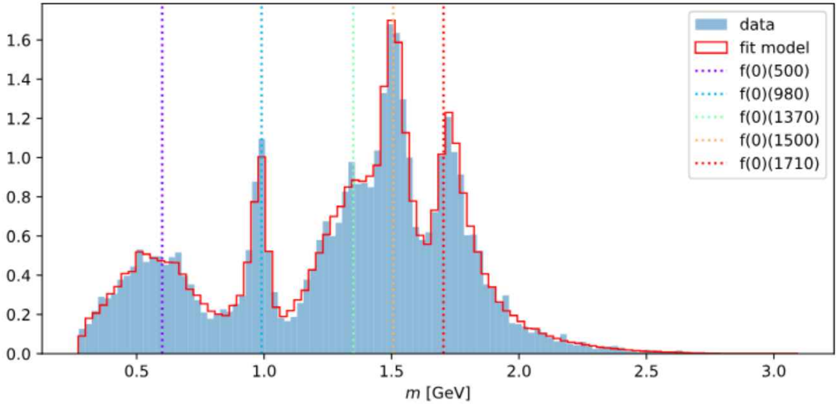
pyhf-like models

- One extreme: HistFactory model (pyhf)
  - Template, binned, analytic normalization
  - Assumption: Bins «free-standing», not next to each other

- «Closed-world» fitter
  - Limited scope, specialized on 80%+ use-case in CMS/ATLAS
  - extremely powerful/tested, serializable

Different kind of fits

- Binned (*vs histfactory*) vs unbinned
  - Refers to data, cost/loss/likelihood and PDF
  - Unbinned data: product of PDFs
  - Binned data: «counting experiments»
- Template vs analytic
  - Shape from (simulation) sample vs closed-form function
- Analytical vs numerical normalization
  - Bin or closed-form integral vs numerical



Amplitude (partial wave) analysis
Angular analysis

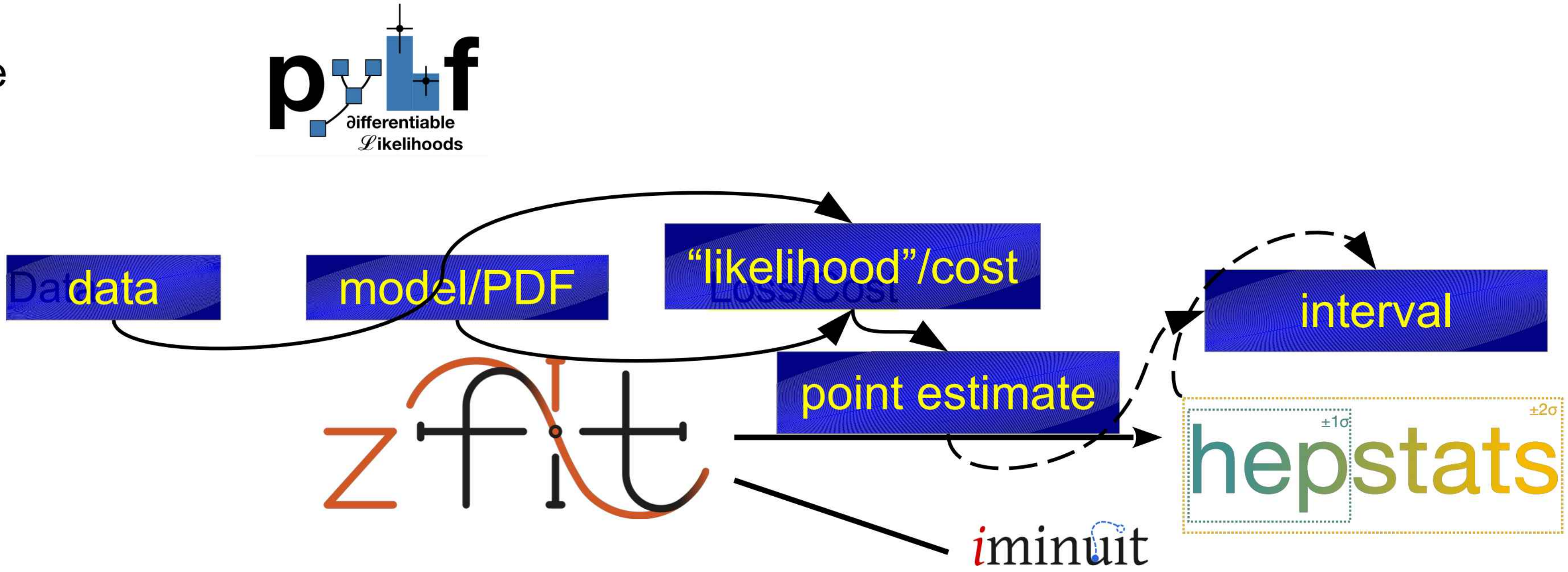
Partial wave analysis

- The other extreme: amplitude analysis (ComPWA, …)
  - Unbinned, analytic, numerical normalisation
  - Description of observable based on amplitude, can be 1k + lines
- Fitting is also hard
  - Fitting time (~100 parameters): hours/days, up to weeks (one fit)
  - Bottleneck: evaluation of PDF

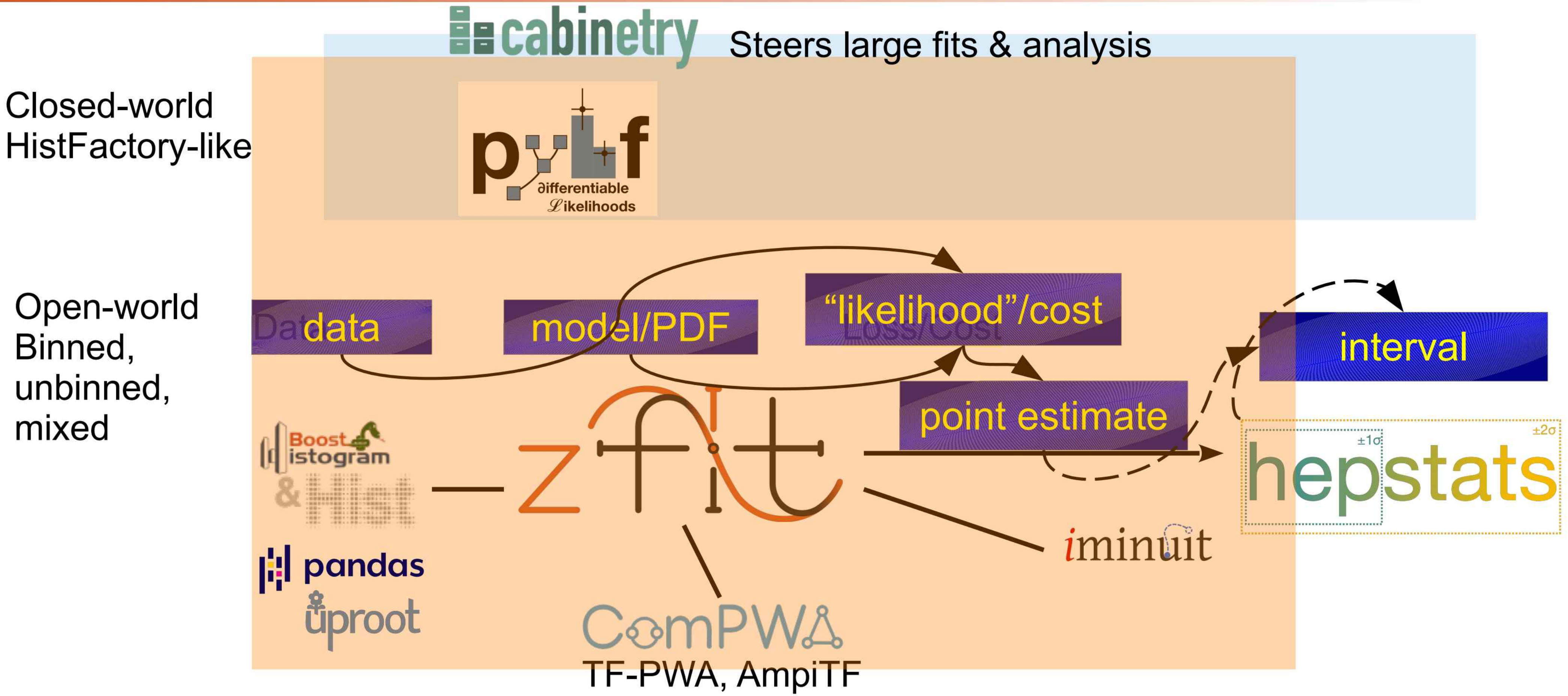
Statistical inference landscape

Closed-world
HistFactory-like

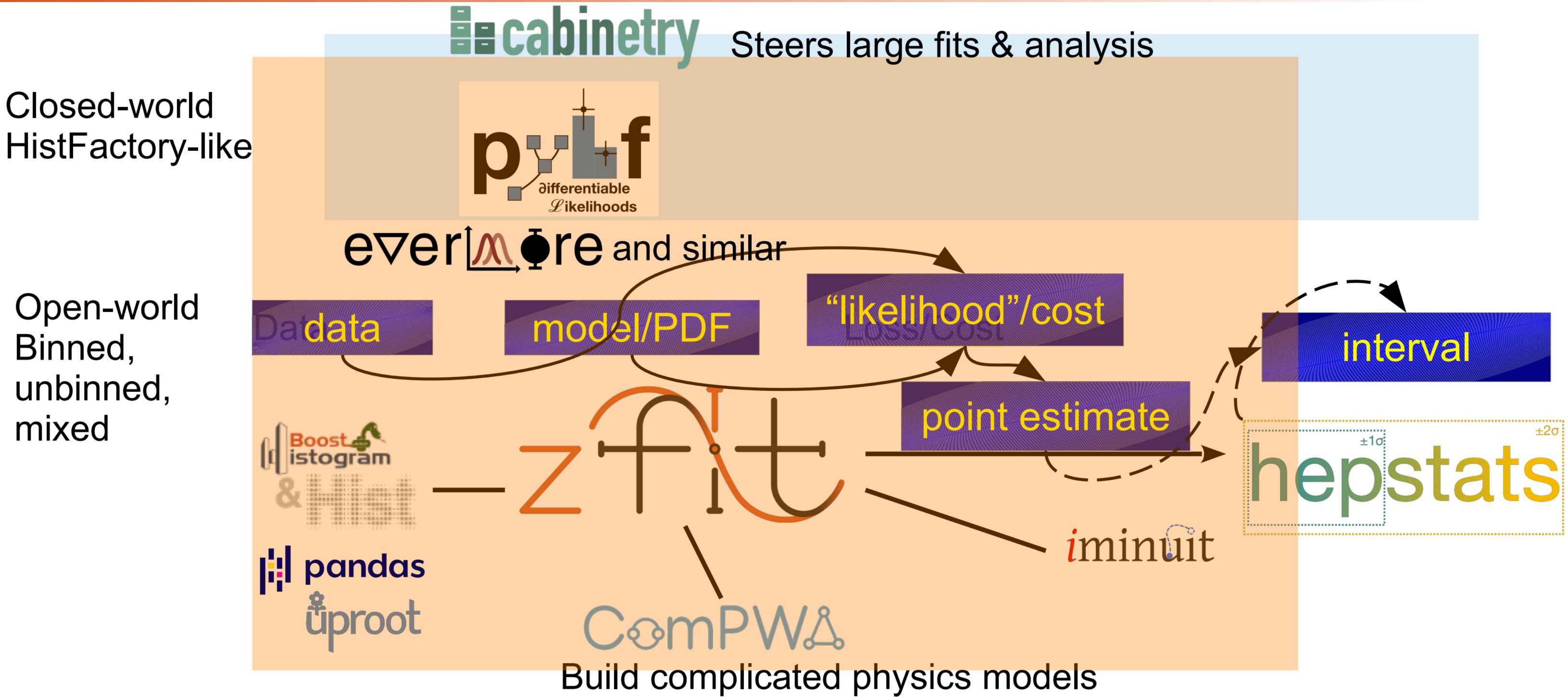
Open-world
Binned,
unbinned,
mixed



Statistical inference landscape



Statistical inference landscape



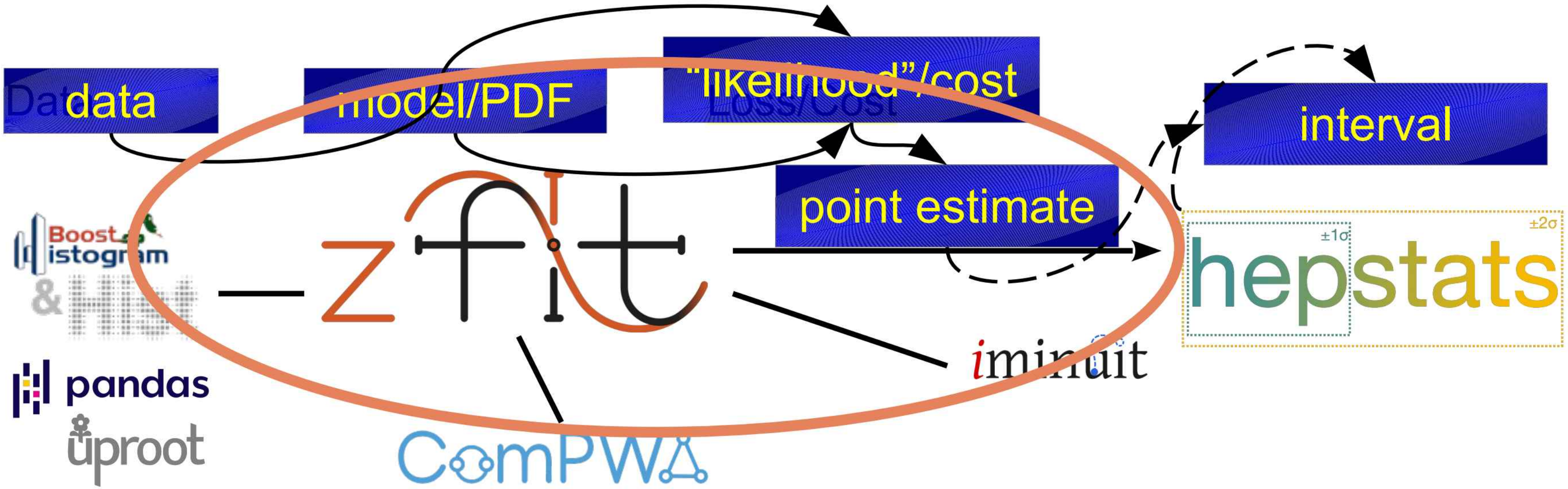
Statistical inference landscape

cabinetry Steers large fits & analysis

Closed-world
HistFactory-like

pyhf ∂ifferentiable ℒikelihoods

Open-world
Binned,
unbinned,
mixed



Build complicated physics models

Historic landscape

Analyses transition from C++ to Python
Many, non-monolithic packages

Talk by Eduardo

HEP fitting libraries still in C++
Strong libraries, but mediocre bindings, not «pythonic»

Python packages
- no advanced features
- «*Python too slow*»

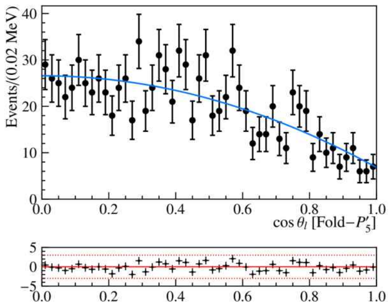
HEP Python:
- TensorProb
- TensorFlow Analysis
- probfit

RooFit

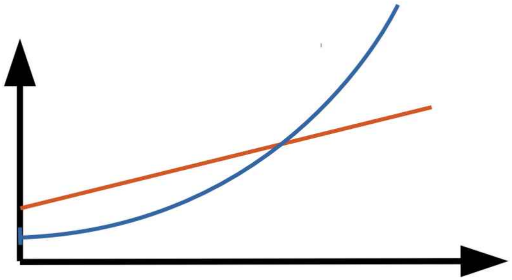
Non-HEP:
- mlfit
- carl
- scipy
- TensorFlow Probability

2018

HEP Model Fitting in Python



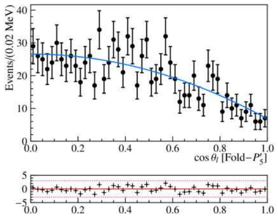
HEP
advanced features,
simply extendable



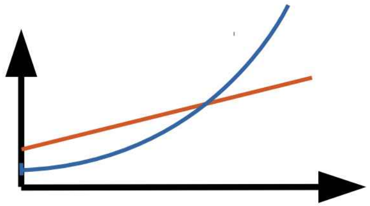
Scalable
large data, complex models

Pythonic
integrate into ecosystem, stable API

HEP Model Fitting in Python



HEP
advanced features,
simply extendable

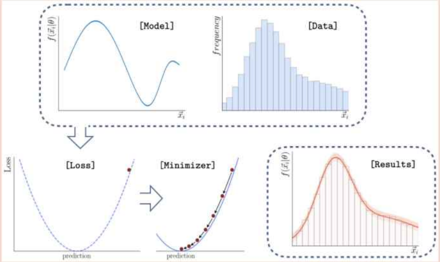


Scalable
large data, complex models

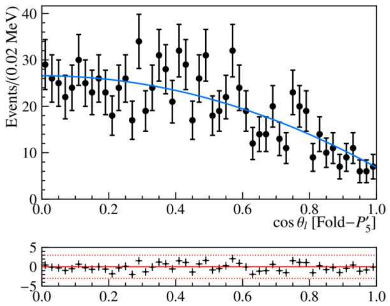
Computing
backend

Pythonic
integrate into ecosystem, stable API

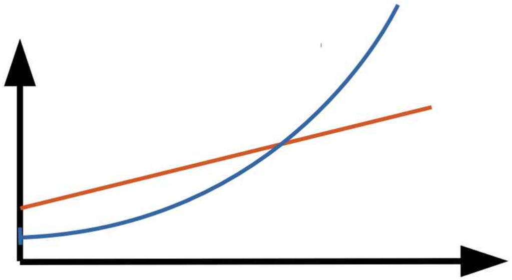
API & Workflow



HEP Model Fitting in Python



HEP
advanced features, simply extendable



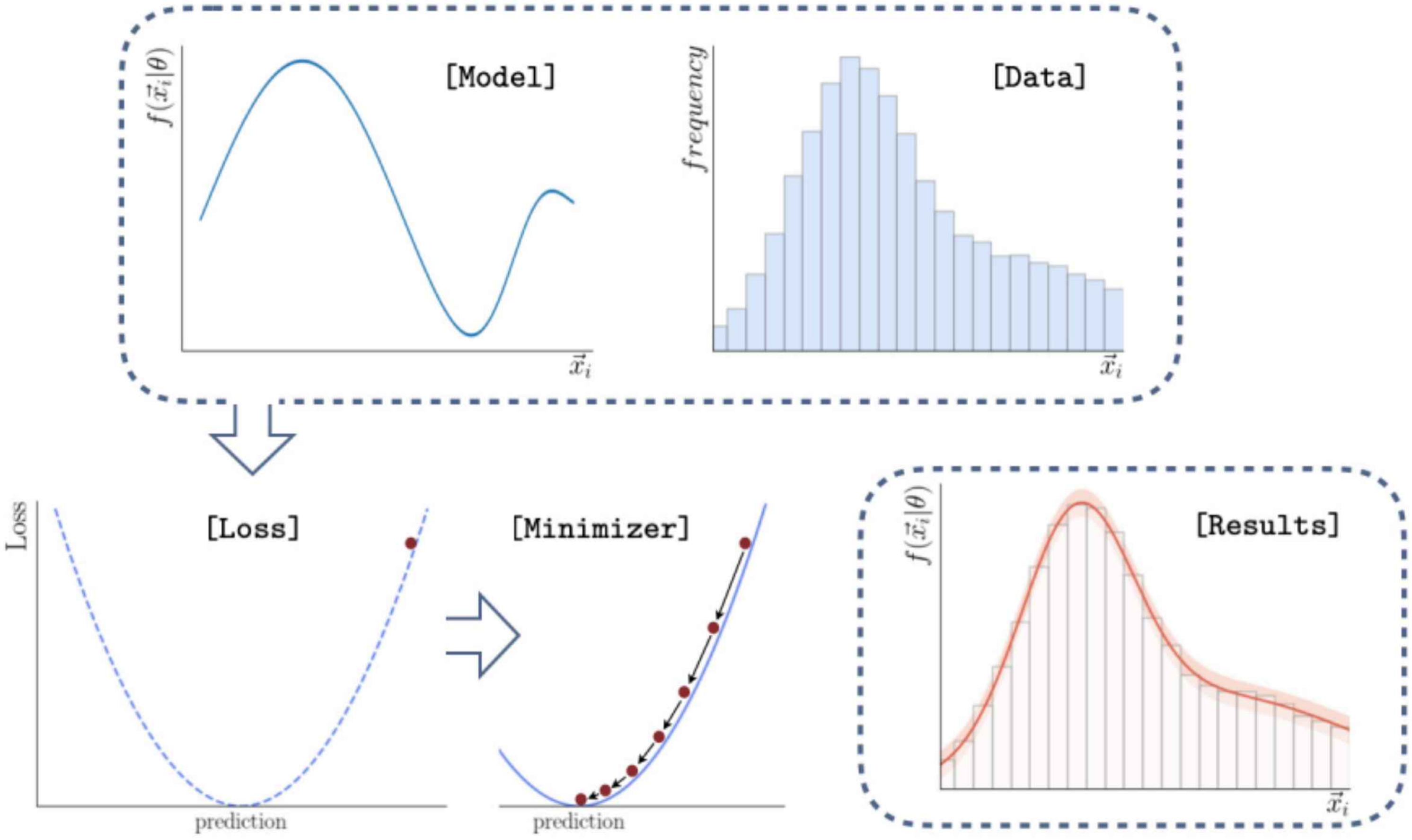
Scalable
large data, complex models

Pythonic
integrate into ecosystem, stable API

API & Workflow

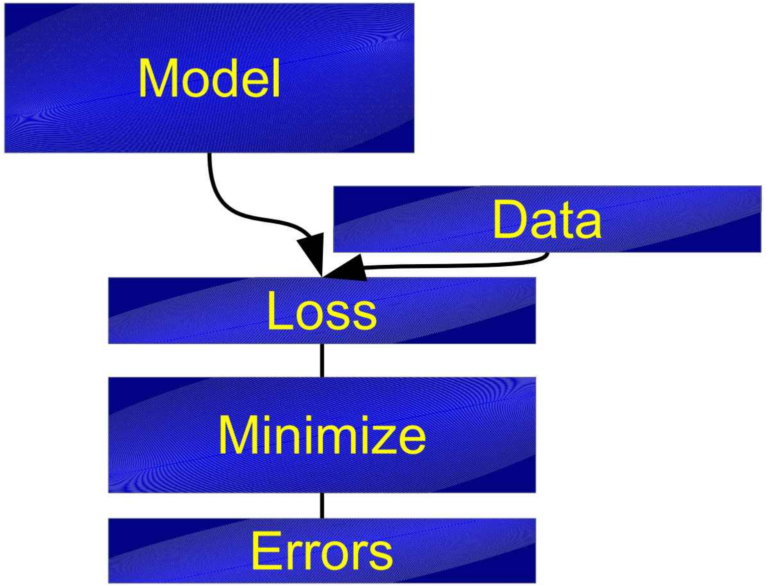
Five maximally independent parts

"Fits look always the same"



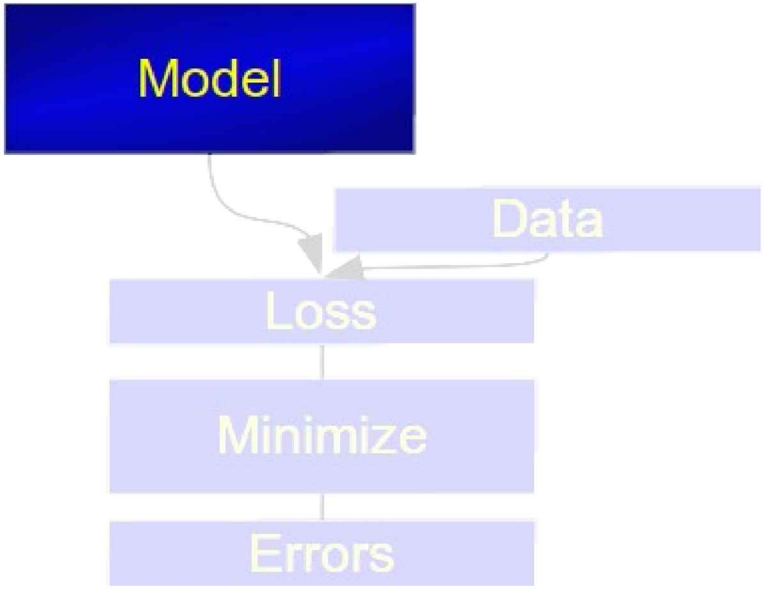
Complete fit

```python
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



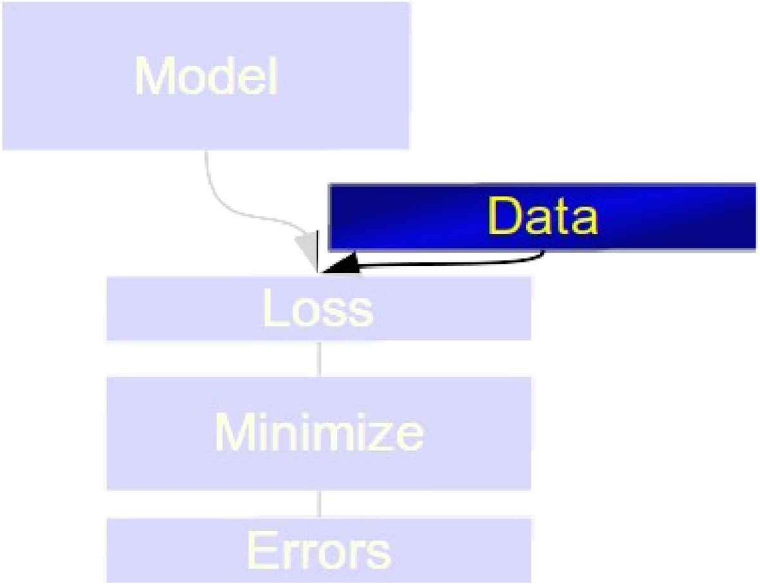
Complete fit: Model

```
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



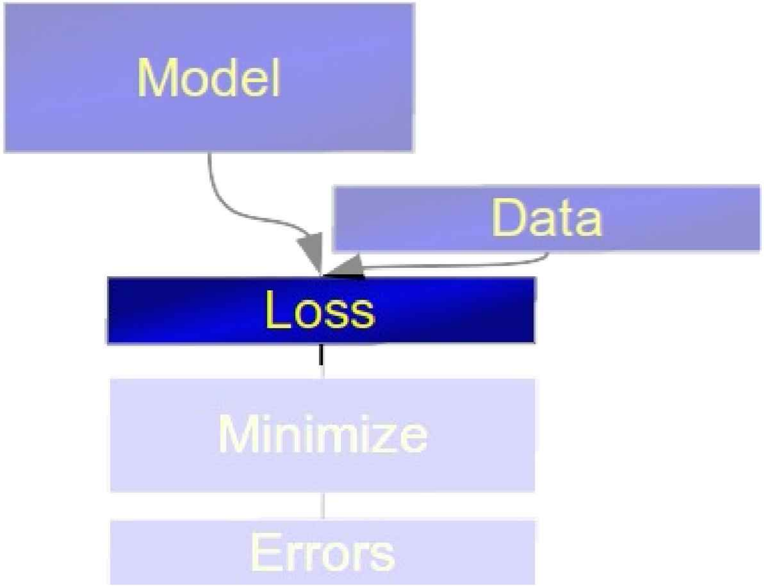
Complete fit: Data

```python
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



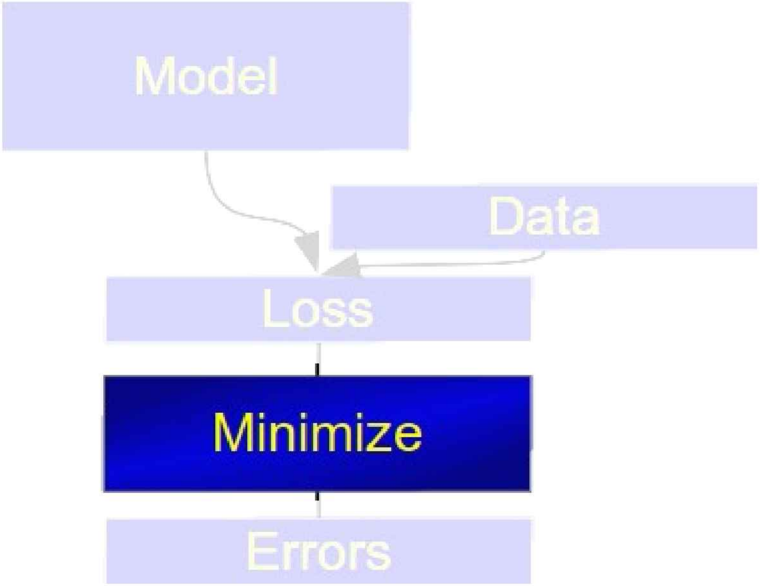
Complete fit: Loss

```
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



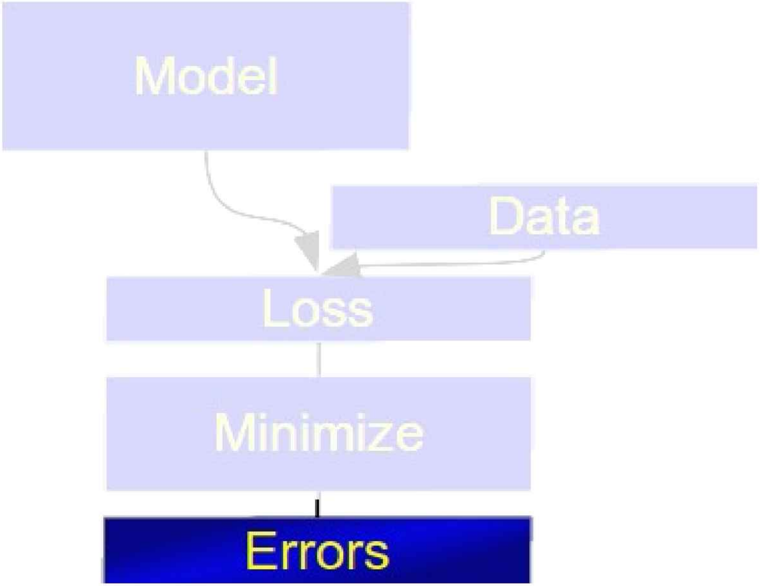
Complete fit: Minimization

```python
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



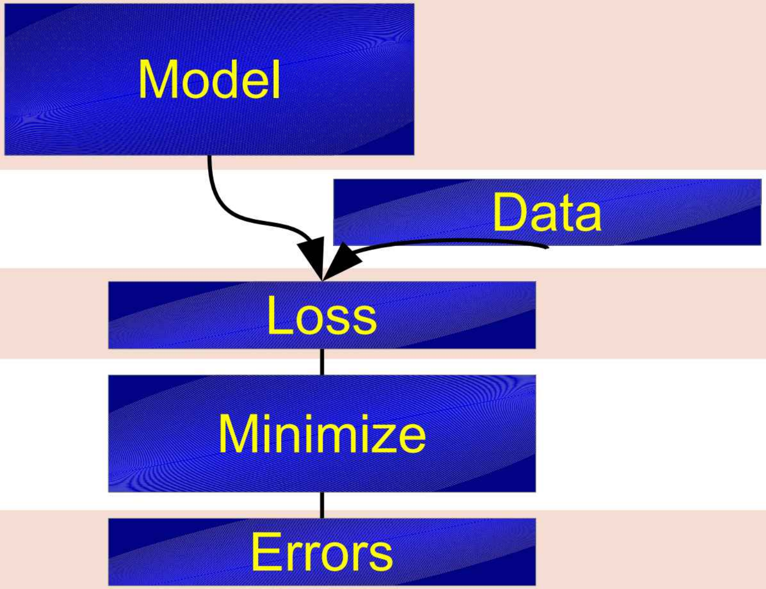
Complete fit: Result

```python
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



Basic API example

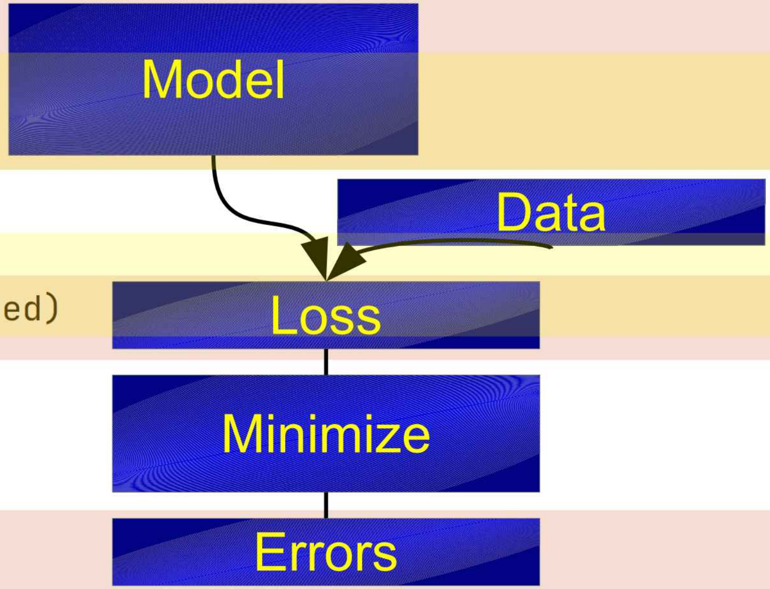
```python
mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



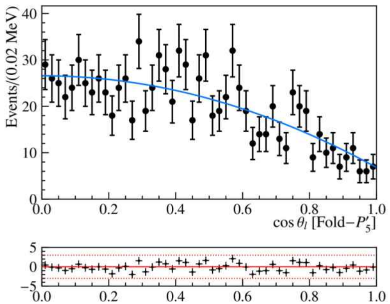
Basic API example

Going binned

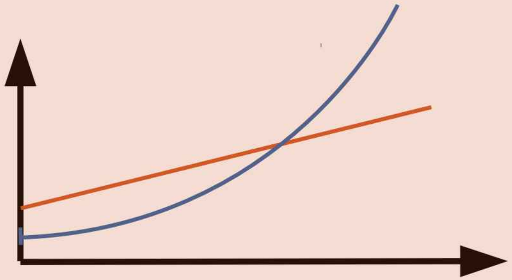
```
mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
obs_binned = obs.with_binning(30)
gauss_binned = gauss.to_binned(obs_binned)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)
data_binned = data.to_binned(obs_binned)
nll = zfit.loss.BinnedNLL(model=gauss_binned, data=data_binned)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



HEP Model Fitting in Python



HEP
advanced features, simply extendable

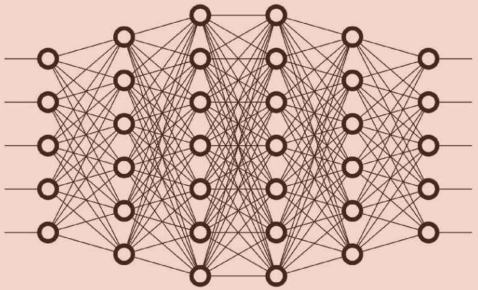


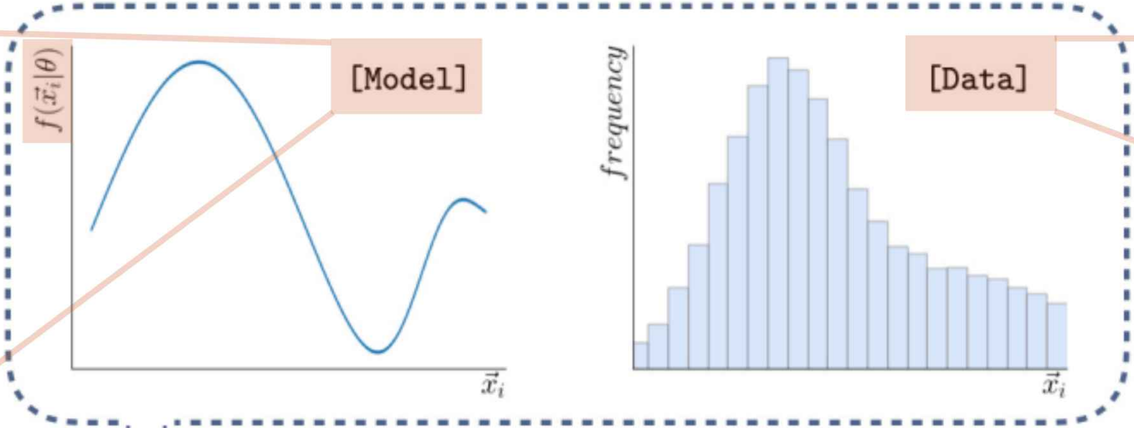
Scalable
large data, complex models

Pythonic
integrate into ecosystem, stable API

Deep Learning

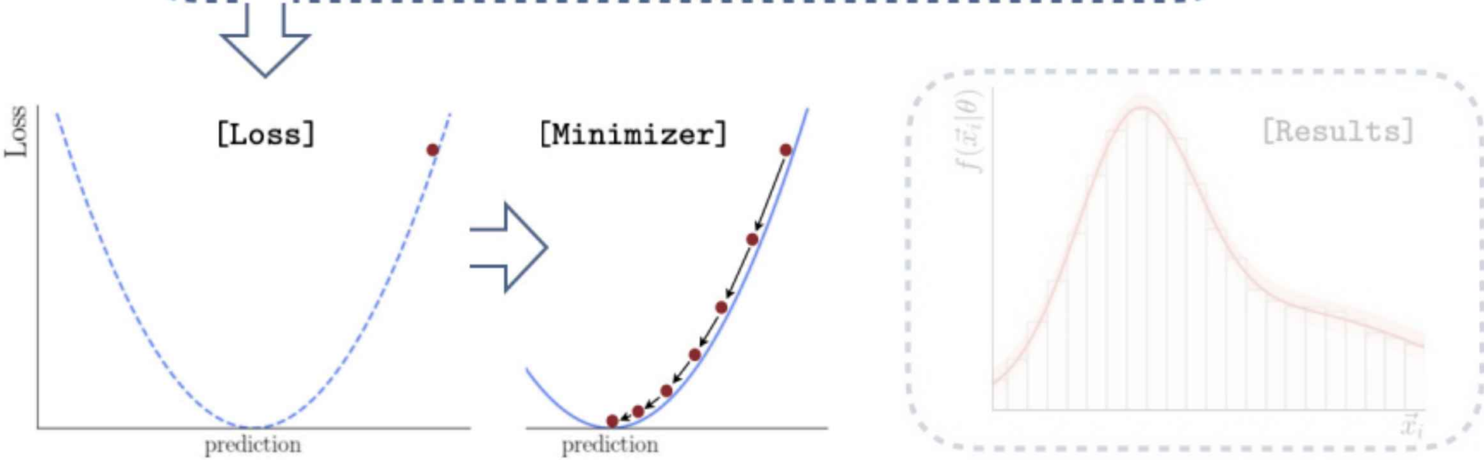
Neural Network





„Big Data“

„One huge, complicated function“



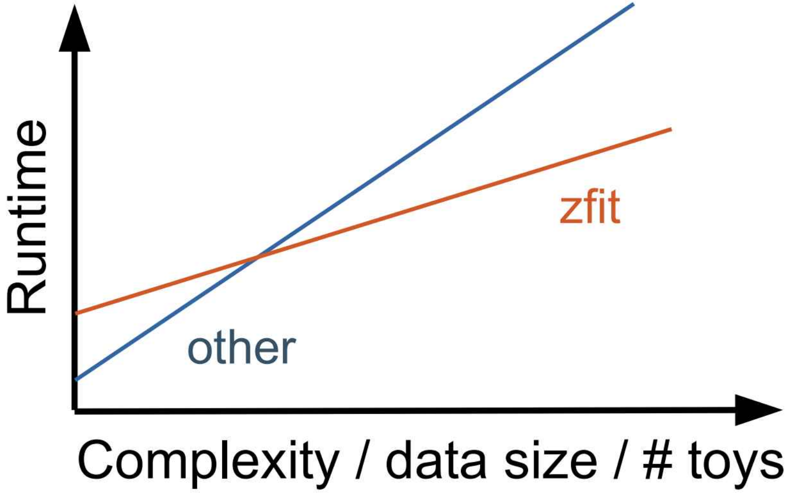
Scalable: Performance

Use same backend as ML uses

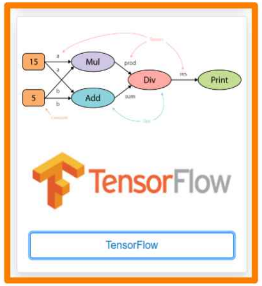
- Numpy-like backend TensorFlow *(JAX)*
  - JIT compiled, CPU or GPU
  - Automatic gradient

Single, simple fit "slow"

- 0.01 or 1 sec not relevant
- 1 or 10 hours relevant



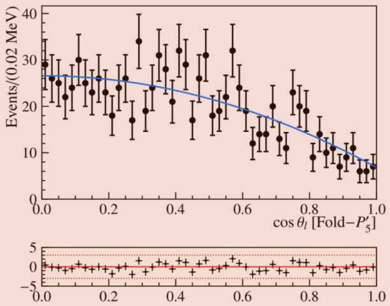

```python
import zfit.z.numpy as znp

ar = znp.linspace(0, 1, 10)
sin = znp.sin(ar)
sum_sin = znp.sum(sin)
```



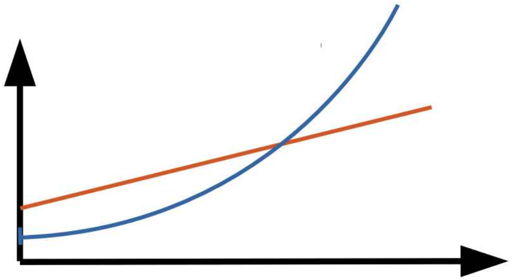
Delegating the workload

	C++ library	Numpy based	zfit
HEP specific content/API			
Models		SciPy	TF Probability
Gradients	CLAD		TensorFlow
Computational optimizations			TensorFlow
Parallelization/GPU		Numba NumPy	TensorFlow intel NVIDIA
Low level handling		python	python

HEP Model Fitting in Python



HEP
advanced features, simply extendable

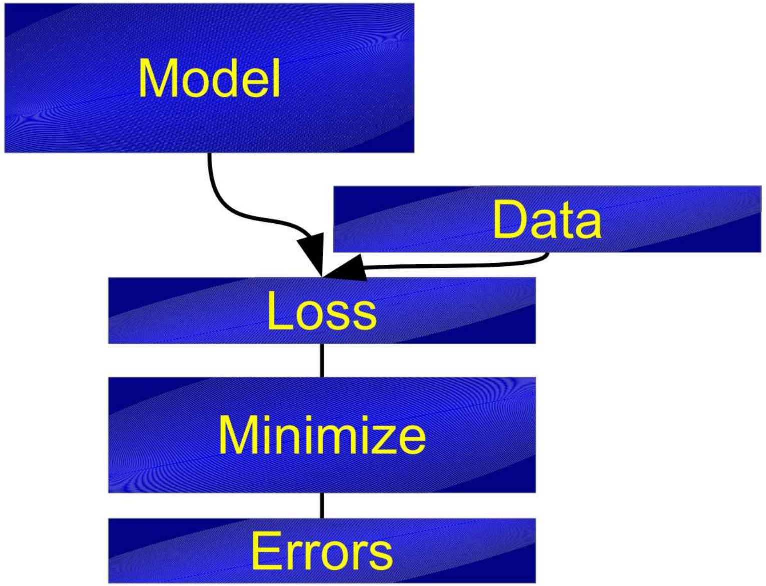


Scalable
large data, complex models

Pythonic
integrate into ecosystem, stable API

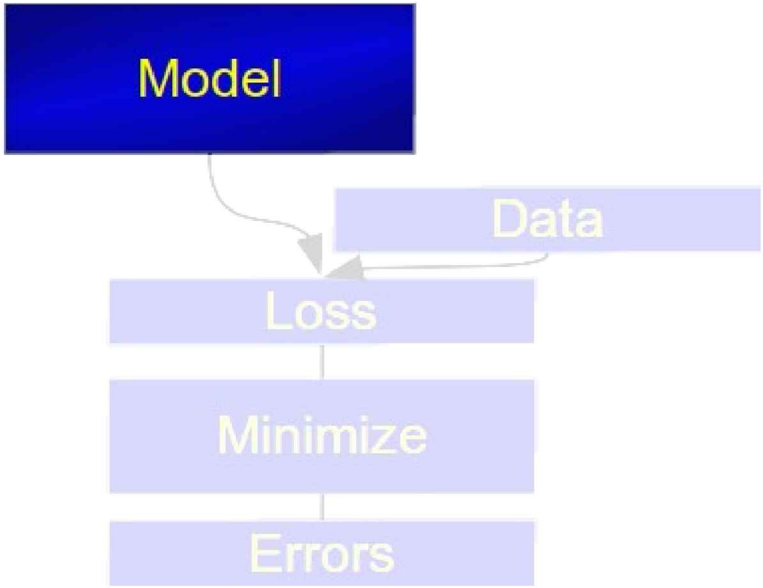
Complete fit

```python
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



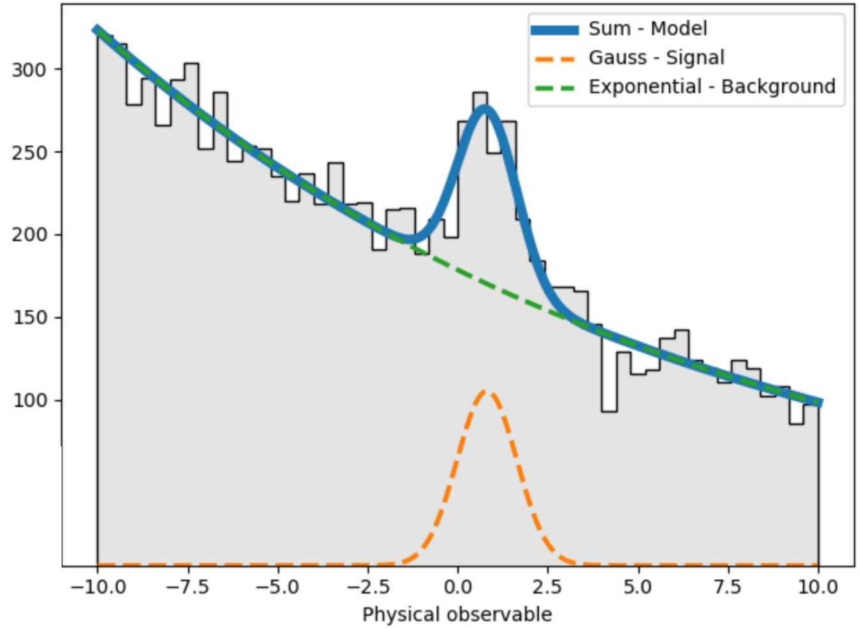
Complete fit: Model

```python
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



Example: Mass fit

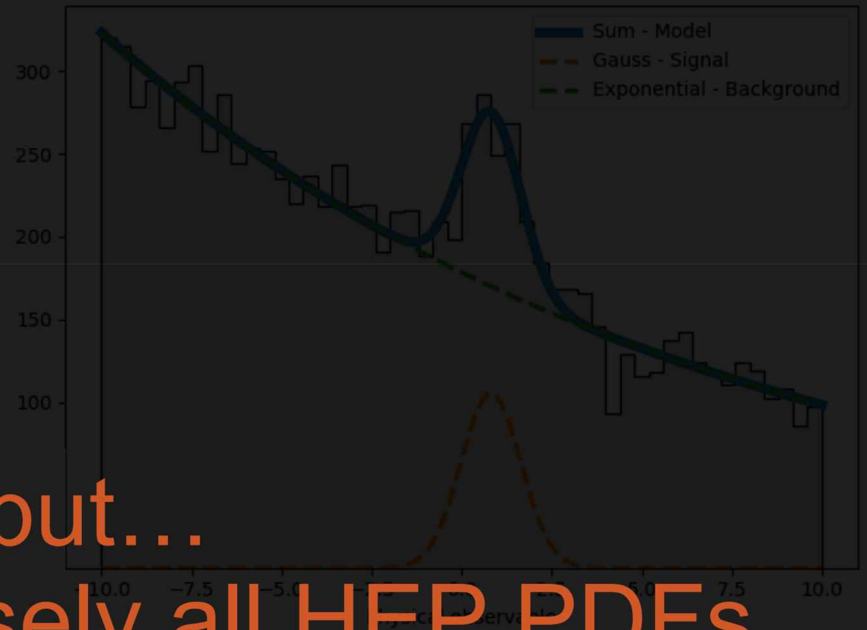
- Sum, Product, *(Convolution)*
- Gauss, (double) Crystalball,...
- Exponential, Polynomials,…
- Histograms, SplineInterpolation,...




```
lambd = zfit.Parameter("lambda", -0.06, -1, -0.01)
frac = zfit.Parameter("fraction", 0.3, 0, 1)

gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
exponential = zfit.pdf.Exponential(lambd, obs=obs)
model = zfit.pdf.SumPDF([gauss, exponential], fracs=frac)
```

Example: Mass fit

- Sum, Product, *(Convolution)*
- Gauss, (double) Crystalball,...
- Exponential, Polynomials,…
- Histograms, SplineInterpolation,...




```
lambd = zfit.Parameter("lambda", -0.06, -1, -0.01)
frac = zfit.Parameter("fraction", 0.5, 0, 1)

gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
exponential = zfit.pdf.Exponential(lambd, obs=obs)
model = zfit.pdf.SumPDF([gauss, exponential], fracs=frac)
```

Good for out-of-the-box but…
does not cover even closely all HEP PDFs

Custom PDF

```python
from zfit import z
from zfit.z import numpy as znp

class CustomPDF(zfit.pdf.ZPDF):
    _PARAMS = ['alpha']

    def _unnormalized_pdf(self, x):
        data = z.unstack_x(x)
        alpha = self.params['alpha']

        return znp.exp(alpha * data)
```

implement custom function

Custom PDF

```python
from zfit import z
from zfit.z import numpy as znp

class CustomPDF(zfit.pdf.ZPDF):
    _PARAMS = ['alpha']

    def _unnormalized_pdf(self, x):
        data = z.unstack_x(x)
        alpha = self.params['alpha']

        return znp.exp(alpha * data)
```

```python
custom_pdf = CustomPDF(obs=obs, alpha=0.2)

integral = custom_pdf.integrate(limits=(-1, 2))
sample   = custom_pdf.sample(n=1000)
prob     = custom_pdf.pdf(sample)
```

} use functionality of model

Custom PDF

```
from zfit import z
from zfit.z import numpy as znp

class CustomPDF(zfit.pdf.ZPDF):
    _PARAMS = ['alpha']

    def _unnormalized_pdf(self, x):
        data = z.unstack_x(x)
        alpha = self.params['alpha']

        return znp.exp(alpha * data)
```

Example of zfit Base Classes

Can also override:

- integrate → _integrate
- pdf → _pdf
- sample → _sample

Or register integral

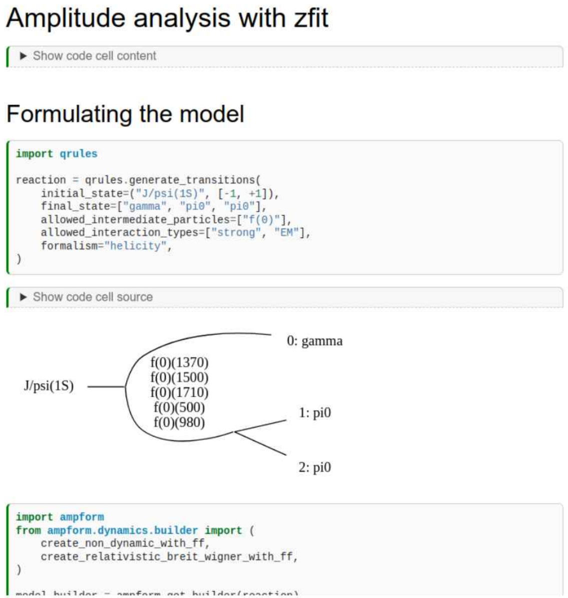
```
custom_pdf = CustomPDF(obs=obs, alpha=0.2)

integral = custom_pdf.integrate(limits=(-1, 2))
sample   = custom_pdf.sample(n=1000)
prob     = custom_pdf.pdf(sample)
```

} use functionality of model

Arbitrary analytic shapes

```python
class P5pPDF(zfit.pdf.ZPDF):
    _PARAMS = ['FL', 'AT2', 'P5p']
    _N_OBS = 3

    def _unnormalized_pdf(self, x):
        FL = self.params['FL']
        AT2 = self.params['AT2']
        P5p = self.params['P5p']
        costheta_l, costheta_k, phi = ztf.unstack_x(x)

        sintheta_k = tf.sqrt(1.0 - costheta_k * costheta_k)
        sintheta_l = tf.sqrt(1.0 - costheta_l * costheta_l)

        sintheta_2k = (1.0 - costheta_k * costheta_k)
        sintheta_2l = (1.0 - costheta_l * costheta_l)

        sin2theta_k = (2.0 * sintheta_k * costheta_k)
        cos2theta_l = (2.0 * costheta_l * costheta_l - 1.0)

        pdf = ((3.0 / 4.0) * (1.0 - FL) * sintheta_2k +
                  FL * costheta_k * costheta_k +
                  (1.0 / 4.0) * (1.0 - FL) * sintheta_2k * cos2theta_l +
                  -1.0 * FL * costheta_k * costheta_k * cos2theta_l +
                  (1.0 / 2.0) * (1.0 - FL) * AT2 * sintheta_2k *
                  sintheta_2l * tf.cos(2.0 * phi) + tf.sqrt(FL * (1 - FL))
                  * P5p * sin2theta_k * sintheta_l * tf.cos(phi))

        return pdf
```

For example, create amplitude with ComPWA and zfit



Binned models

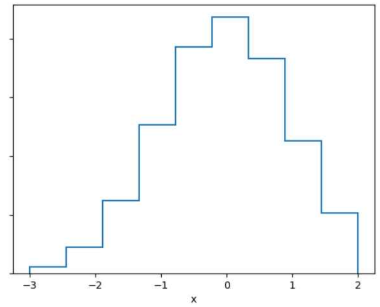
- Success story of Universal Histogram Interface (UHI)
- Modelled after/ compatible with **boost-histogram/hist/UHI**
  - Axes, names, ….

```
h = hist.Hist(hist.axis.Regular(3, -3, 3, name="x", flow=False),
              hist.axis.Regular(2, -5, 5, name="y", flow=False))
x = np.random.randn(1_000_000)
y = 0.5 * np.random.randn(1_000_000)
h.fill(x=x, y=y)

pdf = zfit.pdf.HistogramPDF(data=h)
```

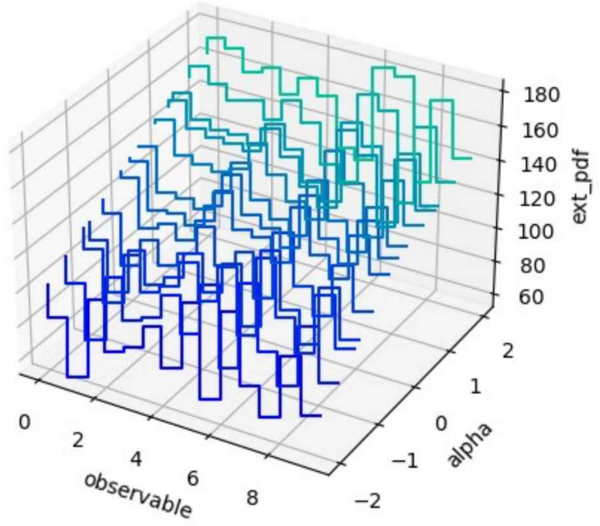
...and back

```
h_back = pdf.to_hist()
```

```
mplhep.histplot(h_back)
```



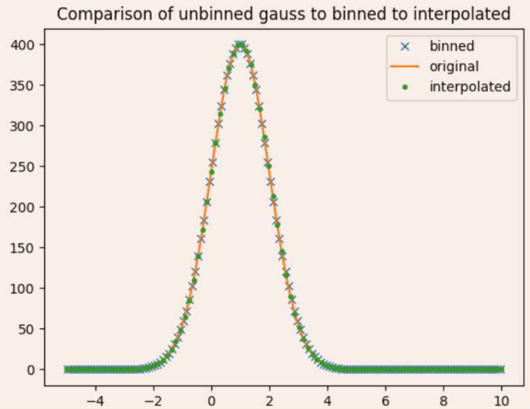
More histograms



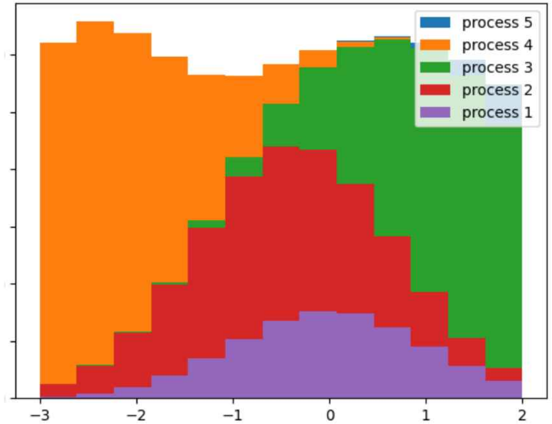

```
pdfs = [zfit.pdf.HistogramPDF(h) for h in histos]
alpha = zfit.Parameter('alpha', 0, -5, 5)
morph = SplineMorphingPDF(alpha=alpha, hists=pdfs)
```

Shape modifier

```
pdf_syst = zfit.pdf.BinwiseScaleModifier(pdf, modifiers=True)
```



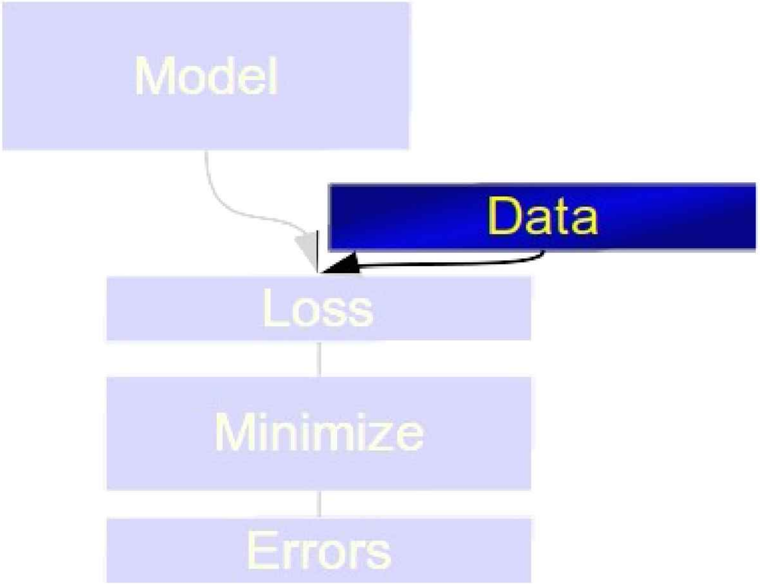
Unbinned → binned → interpolated




```
pdfs = [zfit.pdf.HistogramPDF(h) for h in histos]
sumpdf = zfit.pdf.BinnedSumPDF(pdfs)
```

Complete fit: Data

```
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



Complete fit: Data

- From different sources
  - Hist, numpy, Pandas, ROOT, …
  - Can directly be given

Use the HEP/Python ecosystem for preprocessing

- Sampled from a model (toy studies)

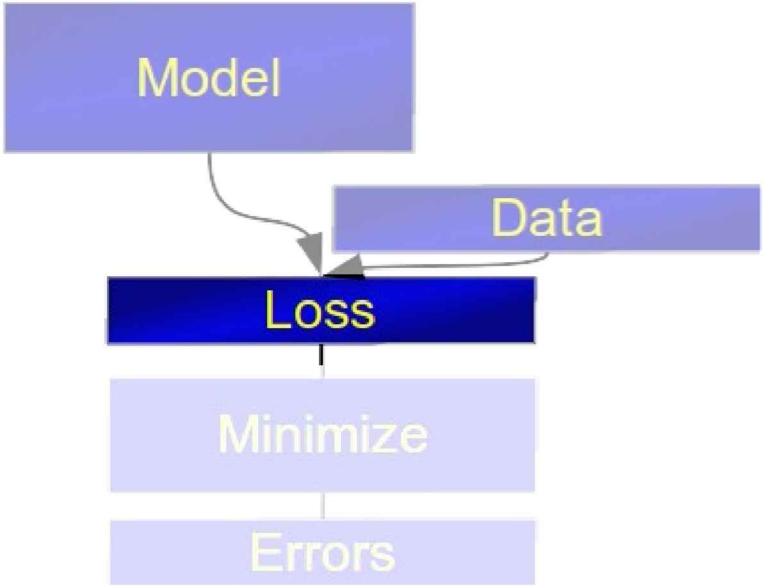
```
data = model.create_sampler(n_sample, limits=obs)
```

- UHI compatible!

```
binneddata = binnedpdf.sample()
mplhep.histplot(binneddata)
```

Complete fit: Loss

```python
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



Loss

```
nll_simultaneous = zfit.loss.UnbinnedNLL(model=[gauss1, gauss2],
                                         data=[data1, data2])

nll1 = zfit.loss.UnbinnedNLL(model=gauss1, data=data1)
nll2 = zfit.loss.UnbinnedNLL(model=gauss2, data=data2)
nll_simultaneous2 = nll1 + nll2
```

Equivalent

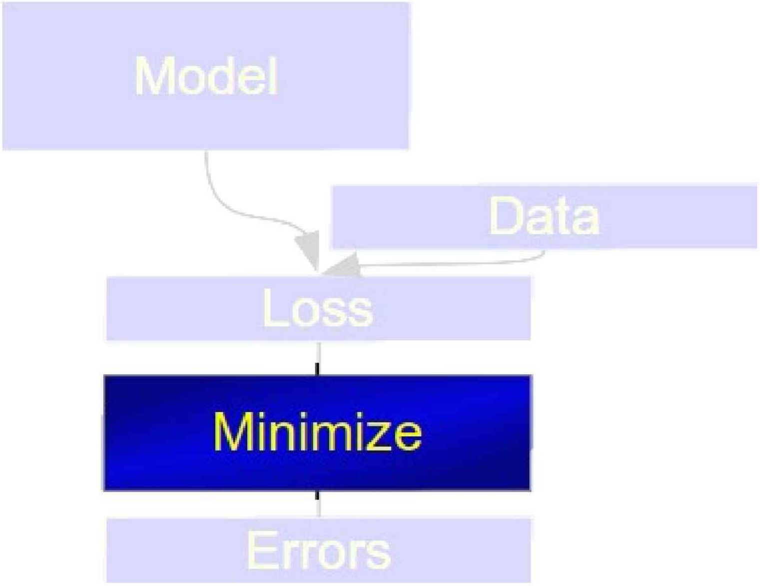
(arbitrary) constraints supported, added to loss

```
constr = GaussianConstraint(params=params, observation=observed, uncertainty=sigma)
nll = zfit.loss.BinnedNLL(model=model, data=data, constraint=constr)
```

Directly compatible with iminuit

Complete fit: Minimization

```python
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



Minimize

- Problem: many, non-unified minimizer APIs
  - SciPy inferface "a bit messy", different convergence criterion, etc...
- Unified API: zfit minimizers, simply switch

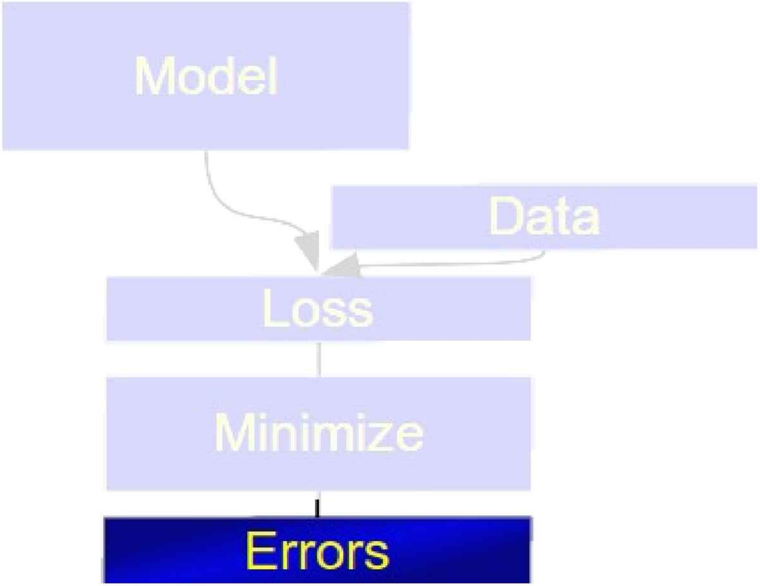
```
minimizer = zfit.minimize.IpyoptV1()
minimizer = zfit.minimize.Minuit()
minimizer = zfit.minimize.ScipyTrustConstrV1()
minimizer = zfit.minimize.NLoptLBFGSV1()
```

- Can use zfit loss, but also ***pure Python function***

```
result = minimizer.minimize(func, params)
```

Complete fit: Result

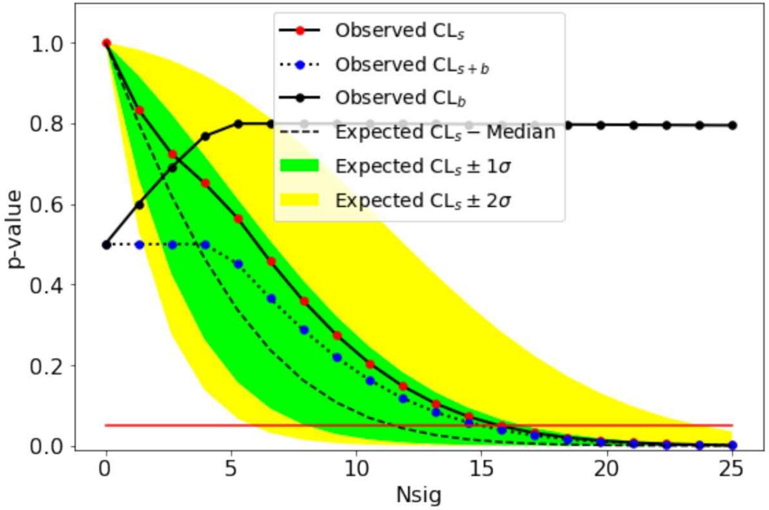
```python
normal_np = np.random.normal(2., 3., size=10_000)

obs = zfit.Space("x", limits=(-2, 3))

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.5, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

data = zfit.Data.from_numpy(obs=obs, array=normal_np)

nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)

minimizer = zfit.minimize.Minuit()
result = minimizer.minimize(nll)

param_errors = result.hesse()
param_errors_asymmetric, new_result = result.errors()
```



hepstats

- Inference library for hypothesis tests
- Takes model, data, loss from zfit
- sWeights, CI, limits, …
- asymptotic or toys calculator

```
calculator = AsymptoticCalculator(loss, minimizer)
poinull = POIarray(Nsig, np.linspace(0.0, 25, 20))
poialt = POI(Nsig, 0)
ul = UpperLimit(calculator, poinull, poialt)
ul.upperlimit(alpha=0.05, CLs=True)
```



HS^3

HEP Statistics Serialization Standard

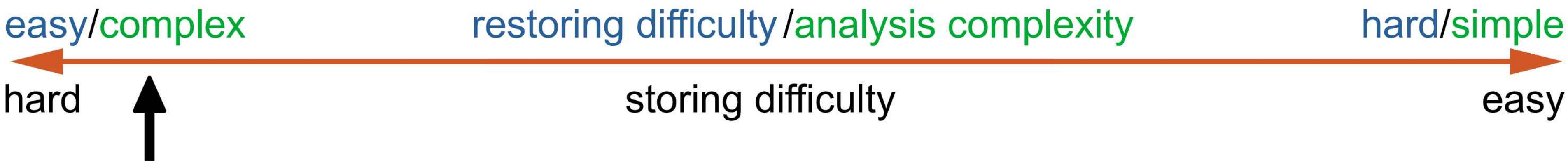
Preservation and interoperability

- Goal: restore and exchange likelihood information
  - Use different frameworks (connect across languages)
  - Easily modify likelihoods (theorists!)
- Question: how? Which format?
  - Human-readable vs binary, scripts vs description,
    virtual machines vs software dependencies, paper vs electronic,…

HS3

HEP Statistics Serialization Standard
Human-readable & preservable format for HEP statistics

- By RooFit, zfit, pyhf and bat.jl; developing stage
- Explore and define common ground
  - What is a Gaussian/Gauss/Normal? Sum? Variable?
  - Defining channels and nuisance parameter (HistFactory-like)
- Best effort base: «What works for all, works»

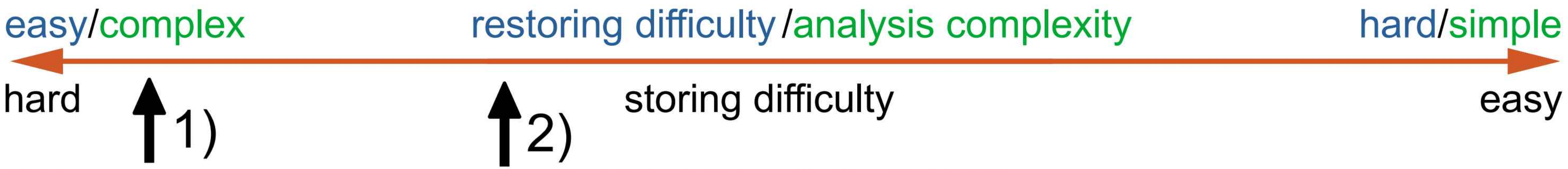


zfit serialization - HS3

1) Can dump/load (some) PDFs HS3-like

2) Custom functions currently pickled, in future as SymPy/Mathematica
   - Project already achieved that
   - ComPWA for amplitudes in Sympy


```
'pdfs': {'SumPDF': {'pdfs': [{'extended': 'n_sig',
                              'mu': 'mu',
                              'sigma': 'sigma',
                              'type': 'Gauss',
                              'x': 'x'},
                             {'extended': 'n_bkg',
                              'lam': 'lambda',
                              'type': 'Exponential',
                              'x': 'x'}],
                    'type': 'SumPDF'}},
'variables': {'lambda': {'max': -0.009999999776482582,
                         'min': -1.0,
                         'name': 'lambda',
                         'step_size': 0.001,
                         'value': -0.06294756382703781},
```

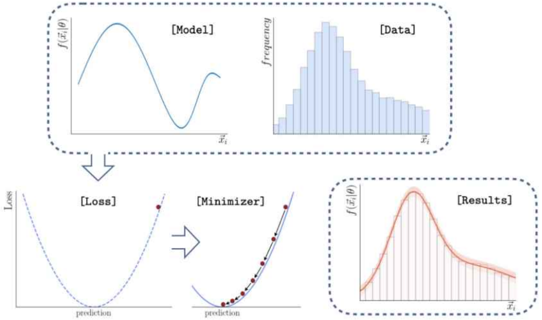
Conclusion

- Python HEP fitting ecosystem
  built from multiple libraries

- zfit «open-world» fitter,
  well integrated with the ecosystem

- Interoperability & building on existing
  Python scientific ecosystem is key

- HS³ likelihood serialization standard



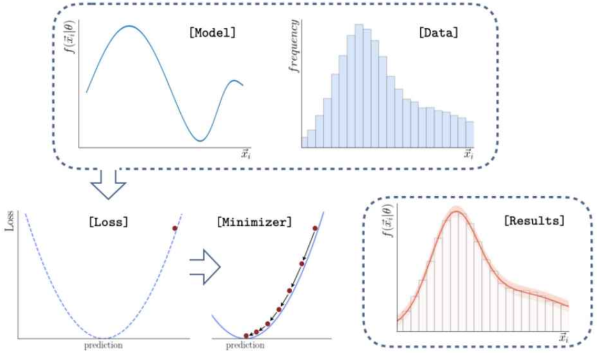
Conclusion

build stable model fitting ecosystem for HEP

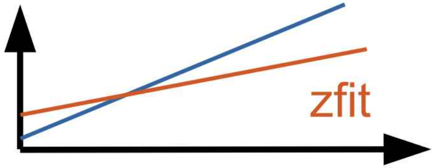
- Recent addition of binned/mixed fits
- Human-readable serialization

    HS3 JSON serialization (WIP)
    Pickling of results
    Custom dumping simple
    Serialization of toys

- Planning for zfit V2




```
'pdfs': {'SumPDF': {'pdfs': [{'extended': 'n_sig',
                              'mu': 'mu',
                              'sigma': 'sigma',
                              'type': 'Gauss',
                              'x': 'x'},
                             {'extended': 'n_bkg',
                              'lam': 'lambda',
                              'type': 'Exponential',
                              'x': 'x'}],
                    'type': 'SumPDF'}},
```

Backup Slides

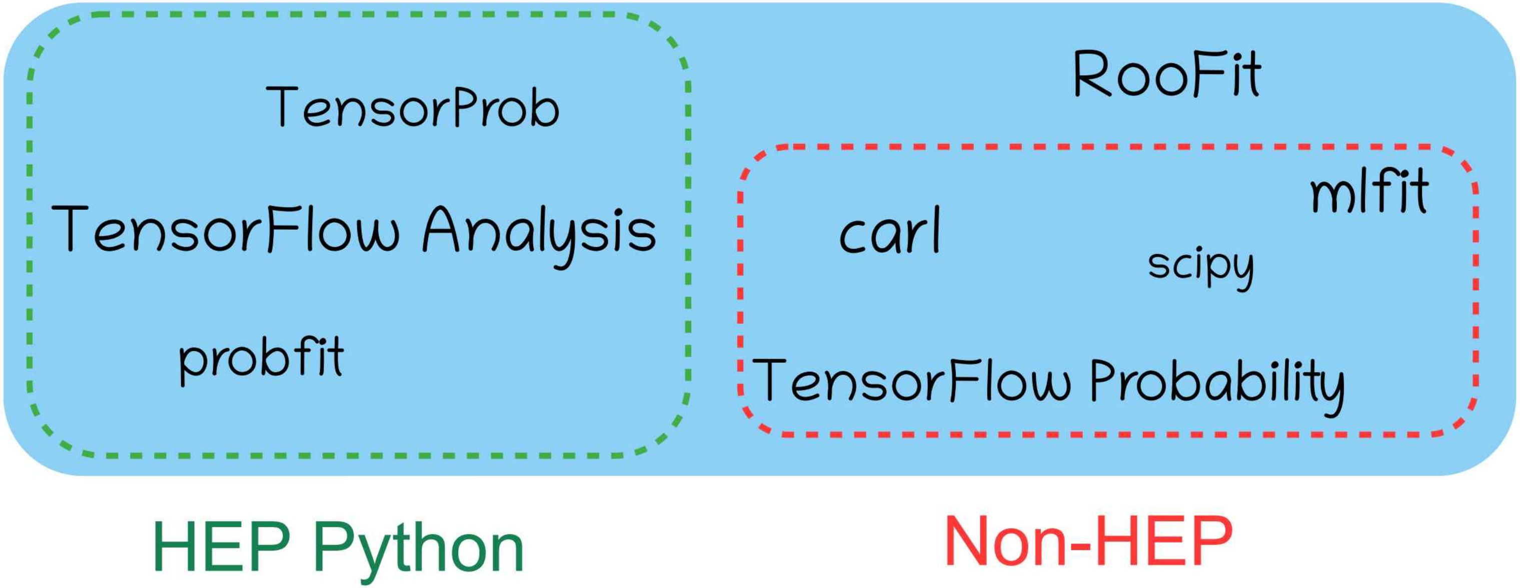
- Backend & TF
- Amplitude
- K*ll toys
- K*mumu Wilson coeffs
- Other fitting packages
- Zfit (associated) packages
- Zfit project
- Zfit elements examples

zfit features

- Extended fits, Chi2, binned, unbinned, mixed
- PDFs convertable binned ↔ unbinned (including to hist), mixed
- Multidimensional
- Any backend supported (numpy-like), optimal with TF currently
- Sample from PDF
- Arbitrary constraints (custom made)
- Custom PDF: define shape → auto normalized, sampling etc.
- Automatic/numerical gradient
- Different minimizers, optimized API
- JIT/eager support

Fitting in Python

A lot of projects are around



Backend & TensorFlow

Backend: a comparison

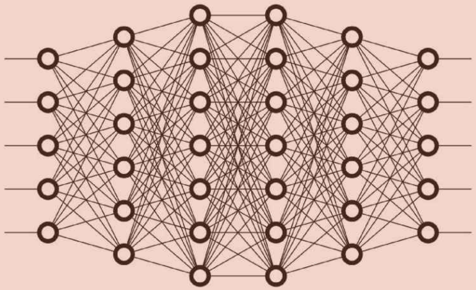
- TensorFlow: supports the most features to this day
- PyTorch: missing advanced math (complex support, …)
- Numpy/SciPy: Too slow, no gradient, no GPU
- JAX: very promising, but *no globals (cache,…)*,
  only static known shapes (adaptive algorithms, accept-reject...), only JAX/Numpy arrays compatible
- SymPy: limited to mathematical expressions (no control-flow,…)
  but can convert to any other backend (used by TensorWaves)

Deep Learning

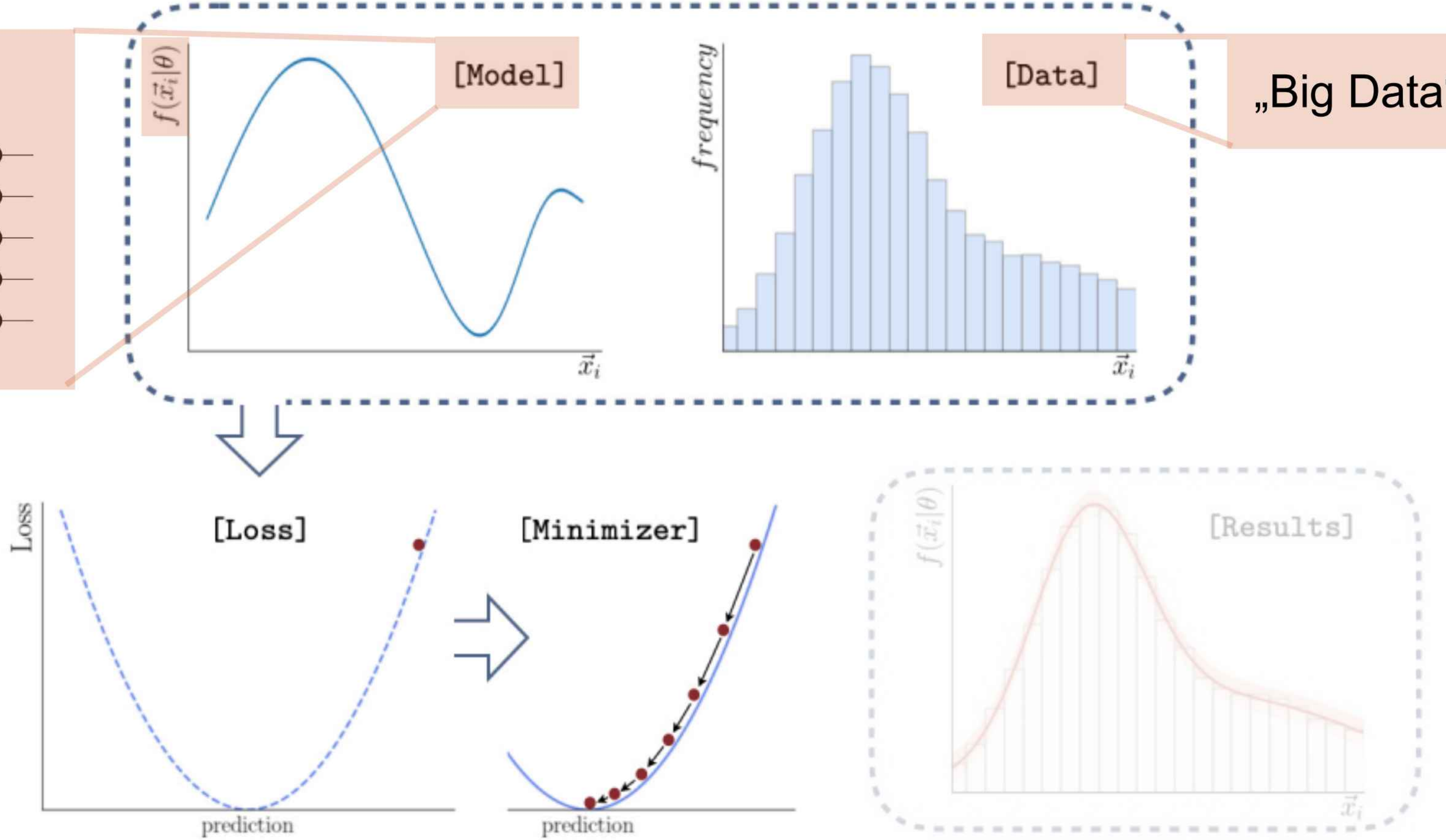
lessons for model fitting

Deep Learning

Neural Network



„One huge, complicated function“



„Big Data“

Can we express model fitting as static graphs?

Yes!

HPC perspective

1) Definition of computation, shape etc. (add static knowledge)
2) Compilation of the graph
3) Execution of computation (re-use optimized graph)

Inside TF, hidden to end-user

HPC: the more is know *before* the execution, the better

TensorFlow takes care of *how* to use this knowledge

Graph elements

… do not have to be constant!

Parameters

Can change their value

Random numbers

Generate newly on every graph execution: MC integration,…

Control flow (if, while)

Steer the execution: Accept-reject sampling (while), etc.

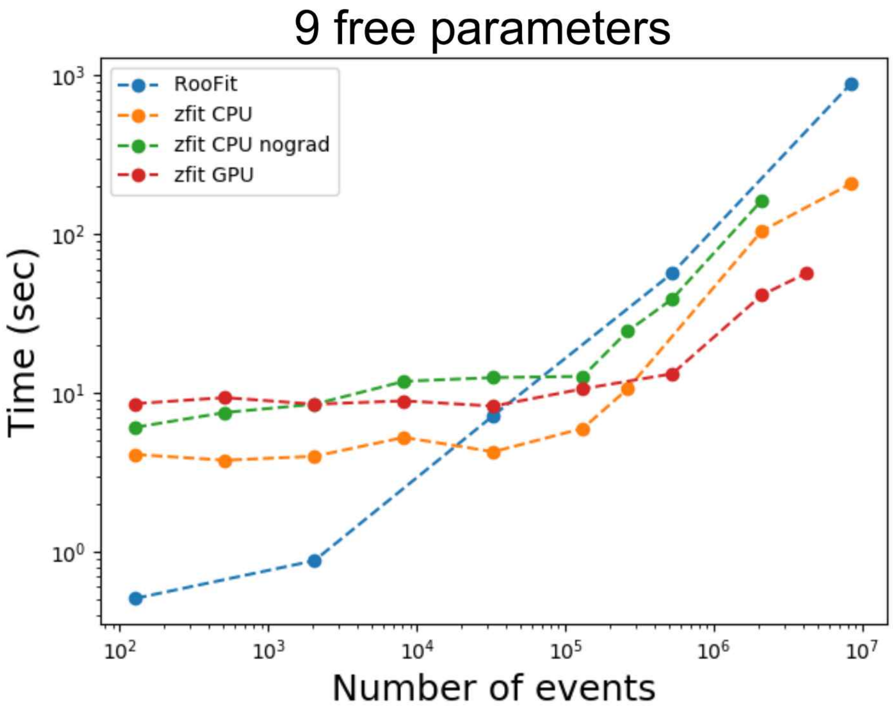
Static, not constant

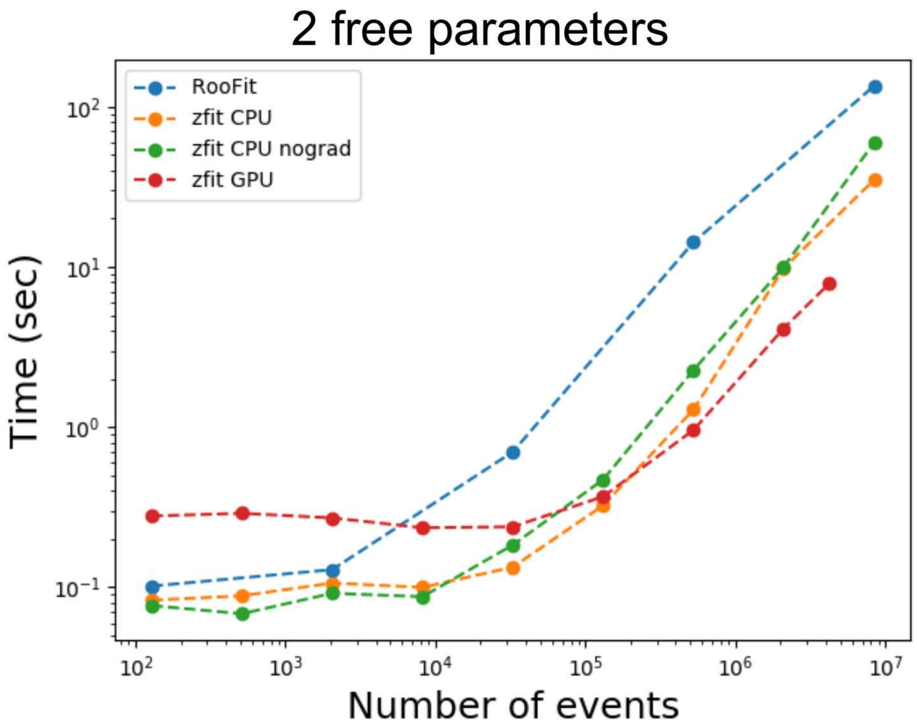
Deep Learning vs. Model Fitting

Similarity	Complicated Models	Large Data	Composed loss	Minimization	Results and uncertainties
HEP	Non-trivial functions	Whole Dataset	simultaneous, constraints	Global min, 2nd derivative algorithm	Hesse, profiling
Deep Learning	Combine many, trivial functions	Many, small Batches	*Anything! (GANs, RL,...)*	Local (!) min, 1th derivative, many steps	None
Conclusion					

Scalability: Performance

Fitting time (lower is better): **RooFit** vs. **zfit**





Amplitude

Angular toys

$B^0 \to K^{*0} l^+ l^-$ angular: toy study

Sensitivity study

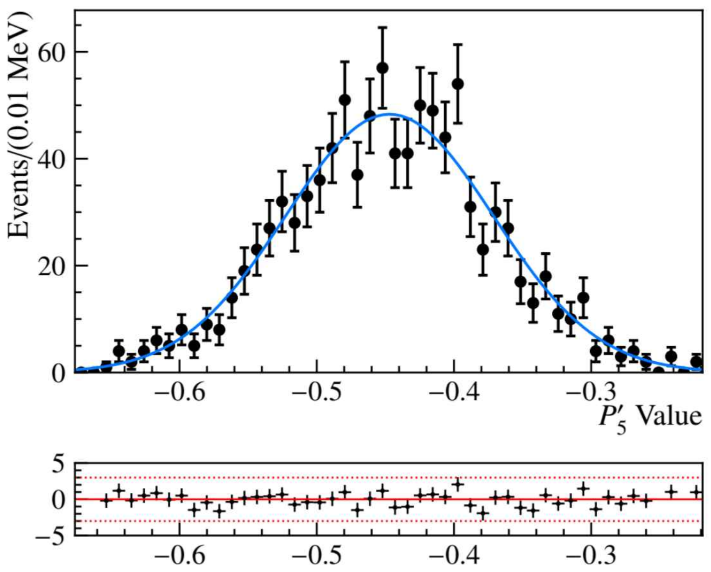
- draw toys (sample) from PDF
- Fit to sample

```
for i in range(ntoys):

    # set initial sampling values
    for param in params:
        param.set_value(...)

    sampler.resample()

    # set random initial values
    for param in params:
        param.set_value(...)

    result = minimizer.minimize(nll)

    if result.converged:
        ...
```

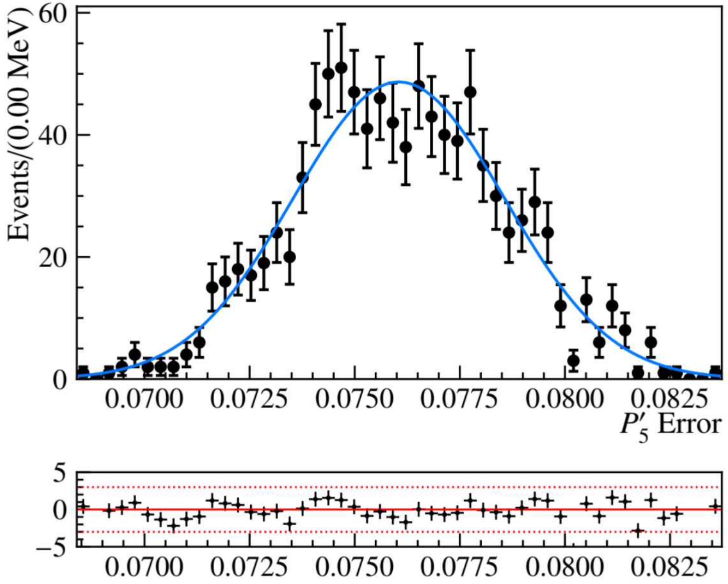
$B^0 \to K^{*0} l^+ l^-$ angular: toy study

Result of toy study

P5' value



P5' error



Extending with a mass shape

```python
# Create mass pdf
mu = zfit.Parameter("mu", 5279, 5200, 5400)
sigma = zfit.Parameter("sigma", 30, 0, 300)
a0 = zfit.Parameter("a0", 1.0, 0, 10)
a1 = zfit.Parameter("a1", 1.0, 0, 10)
n0 = zfit.Parameter("n0", 5, 0, 10)
n1 = zfit.Parameter("n1", 5, 0, 10)

mass = zfit.Space("mass", limits=(4900, 5600))

massPDF = zfit.pdf.DoubleCB(obs=mass, mu=mu, sigma=sigma,
                            alphal=a0, nl=n0, alphar=a1, nr=n1)

pdf = massPDF * angularPDF
```

Build model

Fitting libraries and comparison

Python model fitting in HEP

- Scalable: large data, complex models
- Pythonic: use Python ecosystem/language
- Specific HEP functionality:
  - Normalization: specific range, numerical integration,...
  - Composition of models
  - Multiple dimensions
  - Custom models
  - Non-trivial loss (constraints, simultaneous,…)

HEP Python projects

Probfit, TensorProb,…

- Lack generality and extendibility
- “experimental”, but great proof of concept
  - API and Python in general
  - Computational backends (e.g. Cython, TensorFlow)
  - Building an ecosystem (iminuit,…)

General impression in comparison with other HEP packages

Non-HEP

Scipy, lmfit, TensorFlow Probability,...

- Lack of specific HEP features
  - *Normalization: specific range, numerical integration,...*
  - *Composition of models*
  - *Multiple dimensions*
  - *Custom models*
- Irrelevant functionality supported in API
  - Survival function, …

zfit related packages

phasespace

- Package for phasespace generation of particles
- Covers functionality of TGenPhaseSpace (and more)
- Pure Python (& TensorFlow), integrates seemless with zfit

```
pion = GenParticle('pi+', PION_MASS)
kaon = GenParticle('K+', KAON_MASS)
kstar = GenParticle('K*', KSTARZ_MASS).set_children(pion, kaon)
gamma = GenParticle('gamma', 0)
bz = GenParticle('B0', B0_MASS).set_children(kstar, gamma)

weights, particles = bz.generate(n_events=1000)
```

Zfit: project description

Ecosystem: API & Workflow

Establish a stable API

- High level libraries (statistics, plotting,...)
  - „code against an interface, not an implementation“
- Replace each component
  - Allow other libraries to implement custom parts

Many discussions with community
to avoid splitting/duplication

Pythonic

- Pure Python («pip install zfit»)
- Integrated into python ecosystem
  - Load ROOT files (uproot,no ROOT dependence!)
  - Use Minuit for minimization (iminuit)
  - Data preprocessing with Pandas DataFrame
  - Plotting with matplotlib
  - High level statistics (lauztat, more WIP)
- Extendable classes
  - e.g. custom PDF

Scalable: TensorFlow

- Deep Learning framework by Google
- Modern, declarative graph approach
- Built for highly parallelized, fast communicating CPU, GPU, TPU,… clusters
- Built to use «Big Data»

Zfit library examples

Minimize Python function

```python
def func(x):
    x = np.array(x)  # make sure it's an array
    return np.sum((x - 0.1) ** 2 + x[1] ** 4)

func.errordef = 0.5

params = [1, -3, 2, 1.4, 11]

result = minimizer.minimize(func, params)
```

Model, loss building

sum of two pdfs

```
sum_pdf = zfit.pdf.SumPDF([gauss, exponential], fracs=frac)
```

shared parameters

```
mu_shared = zfit.Parameter("mu_shared", 1., -4, 6)

gauss1 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma1, obs=obs)
gauss2 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma2, obs=obs)
```

simultaneous loss

```
nll1 = zfit.loss.UnbinnedNLL(model=gauss1, data=data1)
nll2 = zfit.loss.UnbinnedNLL(model=gauss2, data=data2)
nll_simultaneous2 = nll1 + nll2
```

From classical

to more TensorFlow

Model, loss building

Simple combinations

```
func_n = zfit.func.ZFunc(...)  # pseudo code
func = func_1 + func_2 * func_3
```

Composite Parameter

```
pdf = zfit.pdf.Gauss(mu=tensor1, sigma=4)
```

Custom Loss

```
loss = zfit.loss.SimpleLoss(lambda: tensor_loss)
```

up to pure TensorFlow

=> use all of zfit functionality like minimizers

Model building

```python
obs = zfit.Space("x", limits=(-10, 10))

mu =    zfit.Parameter("mu",           1,  -4,  6)
sigma = zfit.Parameter("sigma",        1, 0.1, 10)
lambd = zfit.Parameter("lambda",      -1,  -5,  0)
frac =  zfit.Parameter("fraction", 0.5,   0,  1)

gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
exponential = zfit.pdf.Exponential(lambd, obs=obs)
```

parameters

models

Simultaneous fit

```python
mu_shared = zfit.Parameter("mu_shared", 1., -4, 6)
sigma1 = zfit.Parameter("sigma_one", 1., 0.1, 10)
sigma2 = zfit.Parameter("sigma_two", 1., 0.1, 10)

gauss1 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma1, obs=obs)
gauss2 = zfit.pdf.Gauss(mu=mu_shared, sigma=sigma2, obs=obs)
```

shared parameters

```python
nll_simultaneous = zfit.loss.UnbinnedNLL(model=[gauss1, gauss2],
                                         data=[data1, data2])

nll1 = zfit.loss.UnbinnedNLL(model=gauss1, data=data1)
nll2 = zfit.loss.UnbinnedNLL(model=gauss2, data=data2)
nll_simultaneous2 = nll1 + nll2
```

Completely equivalent