Binned and template fits in TensorFlow

zfit

on behalf of zfit

Jonas Eschle

jonas.eschle@cern.ch

FNSNF Swiss National Science Foundation

University of Zurich UZH

Meeting

- Exchange knowledge, learn who does what
- Crucial pieces
  - Parameters
  - Binned data format
  - Template PDF technical
- Find common problems
  - Missing parts in TF

Why zfit

Fitting with TensorFlow is possible

- Binned, unbinned, template
- Several independent approaches

but limited:

- Specific usecases
- TF knowledge required

TF in a nutshell

Use Python to build graphs «Like a compiler»

Execute graph

TF runs computation

zfit: the idea

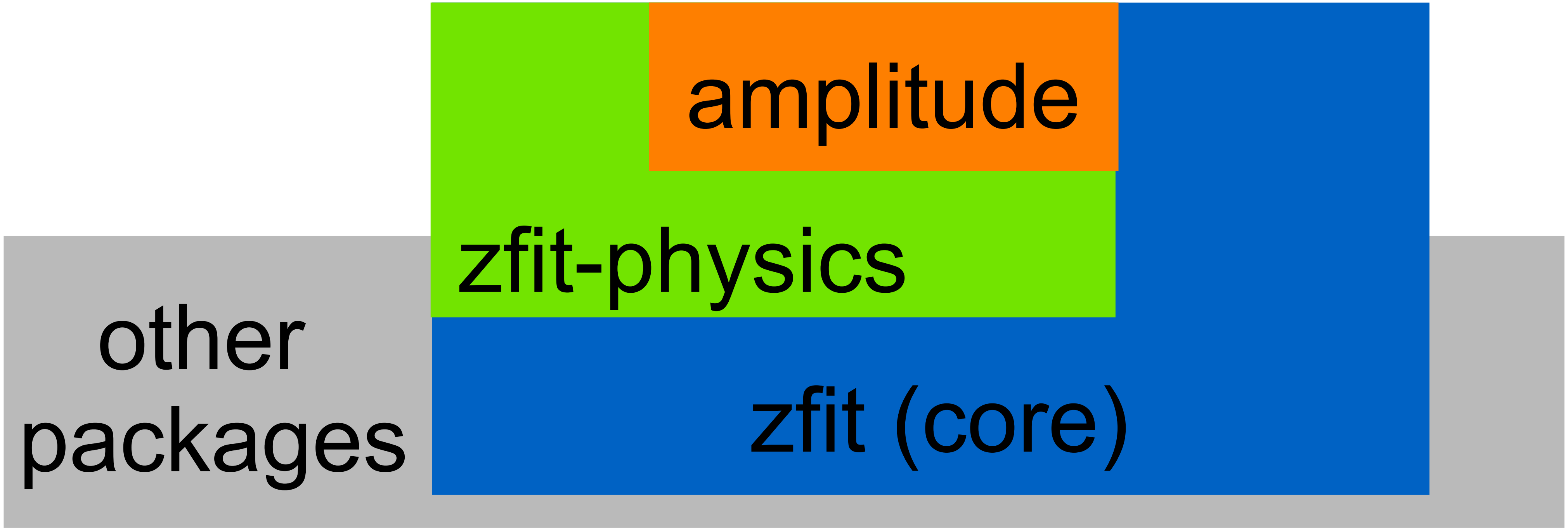
- Combine approaches and efforts, knowledge is there...
- Find stable API: ***Connect*** libraries, build on top
- Implement pieces *once:* Allows to ***optimize parts***

zfit: pythonic fitting with TF

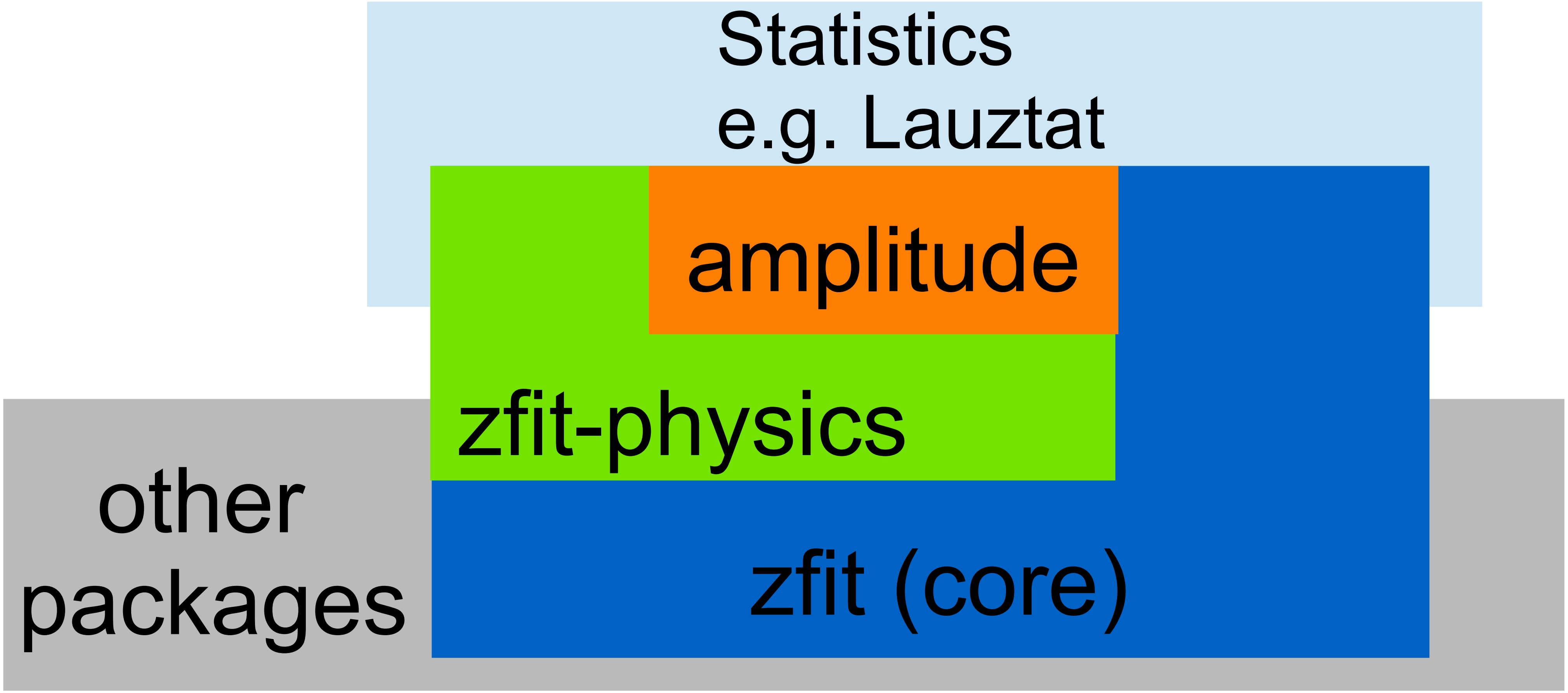
connect specific packages through APIs and workflow

NOT focus on implementing too specific content

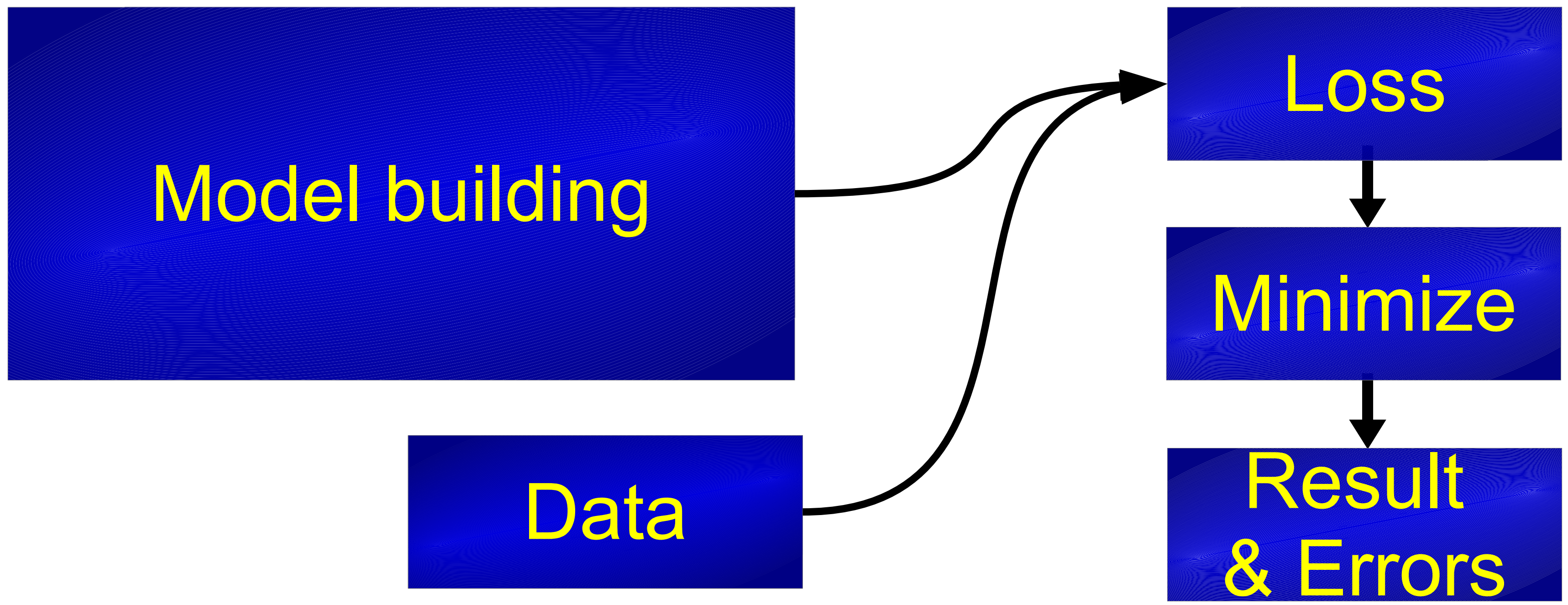
zfit landscape



zfit landscape



zfit workflow



What is `zfit`?

Fitting for HEP (*models, composition, unbinned)*

Fresh API, codebase *(inspired by others)*

What is `zfit`?

Fitting for HEP (*models, composition, unl*

Fresh API, codebase *(inspired by others)*

TensorProb

TensorFlow Probability

mlfit

scipy

carl

probfit

RooFit

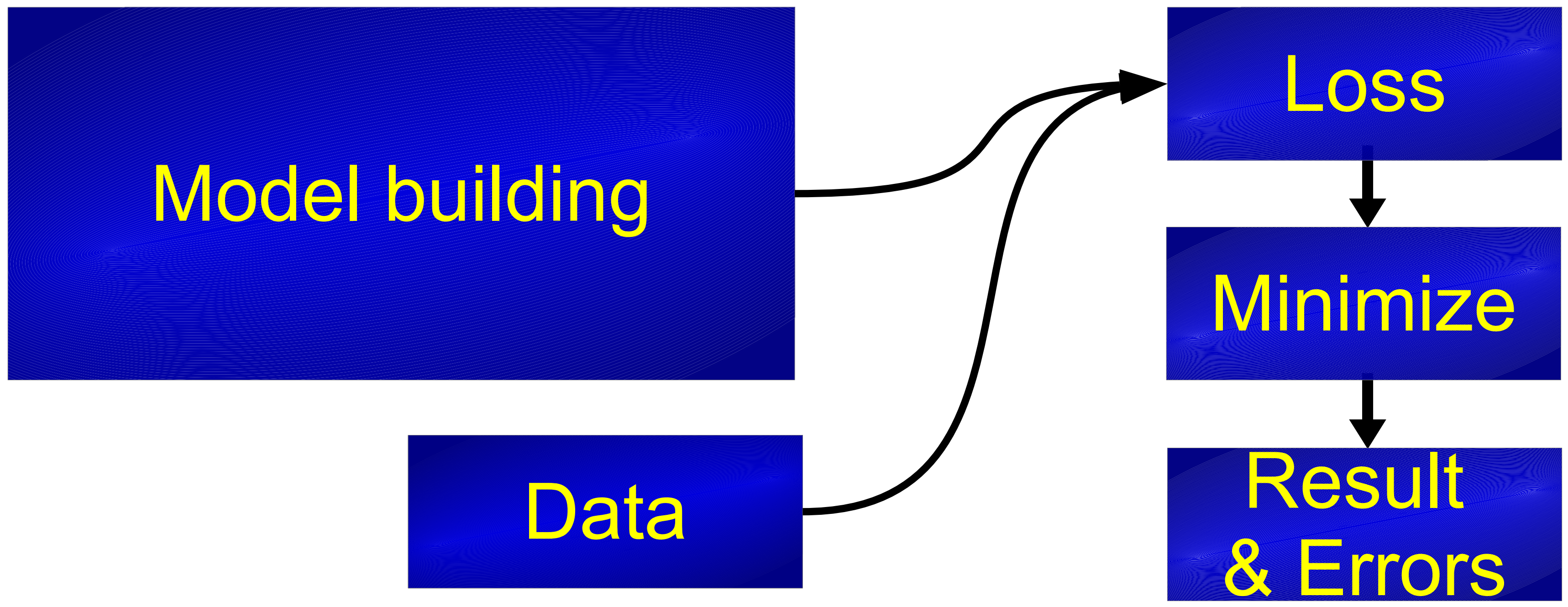
TensorFlow Analysis

zfit status

- Currently, only unbinned fits
  - binned data experimental, no template pdfs (yet)
- Strong n dimensional model building
  - Custom models
  - Model composition
  - Sampling, integration, pdf

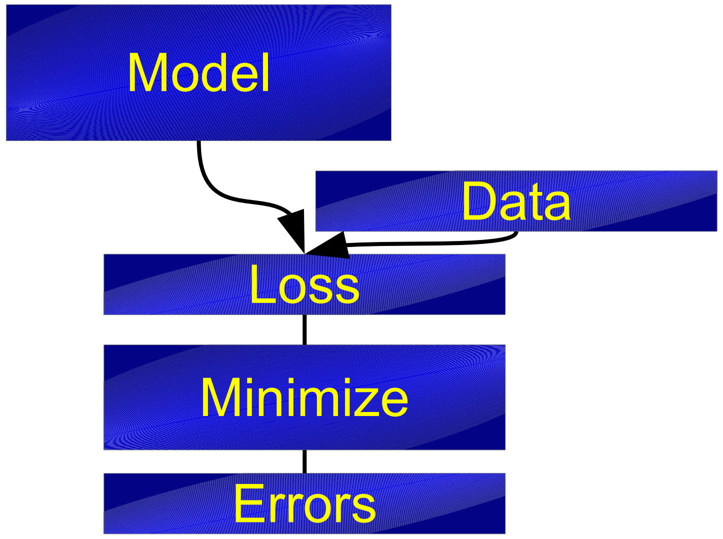
Stablizing API, fixed workflow

Fitting



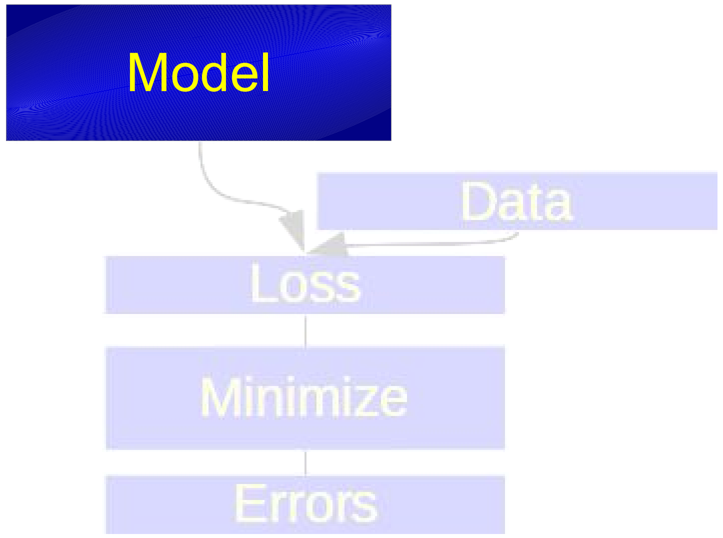
Complete fit

```
import zfit

normal_np = np.random.normal(loc=2., scale=3., size=10000)

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

mu    = zfit.Parameter("mu",    1., -5,  5)
sigma = zfit.Parameter("sigma", 3.,  1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

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

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

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

param_errors = result.error()
```



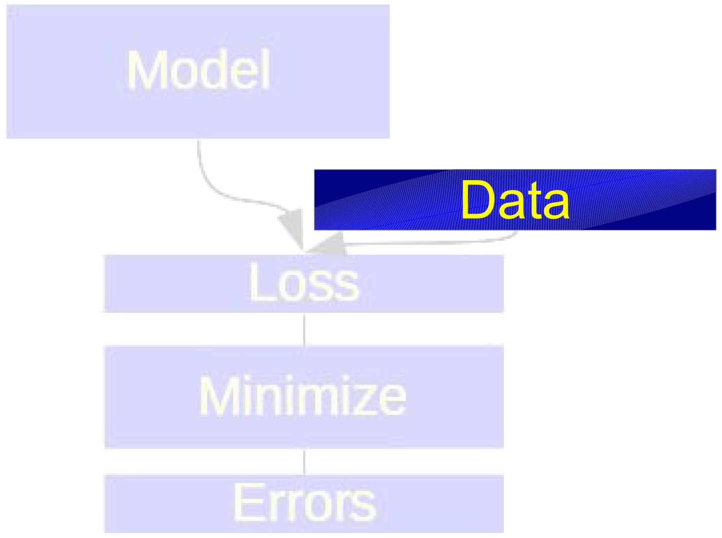
Complete fit

```
import zfit

normal_np = np.random.normal(loc=2., scale=3., size=10000)

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

mu    = zfit.Parameter("mu",    1., -5,  5)
sigma = zfit.Parameter("sigma", 3.,  1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

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

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

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

param_errors = result.error()
```



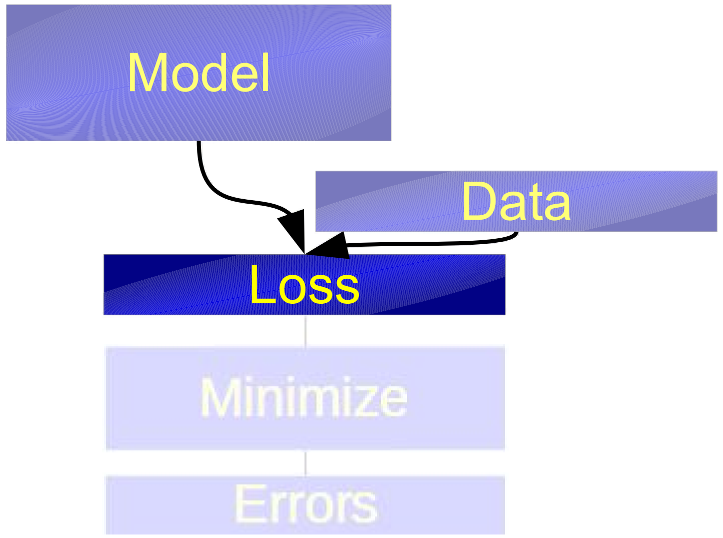
Complete fit

```
import zfit

normal_np = np.random.normal(loc=2., scale=3., size=10000)

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

mu    = zfit.Parameter("mu",    1., -5,  5)
sigma = zfit.Parameter("sigma", 3.,  1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

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

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

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

param_errors = result.error()
```



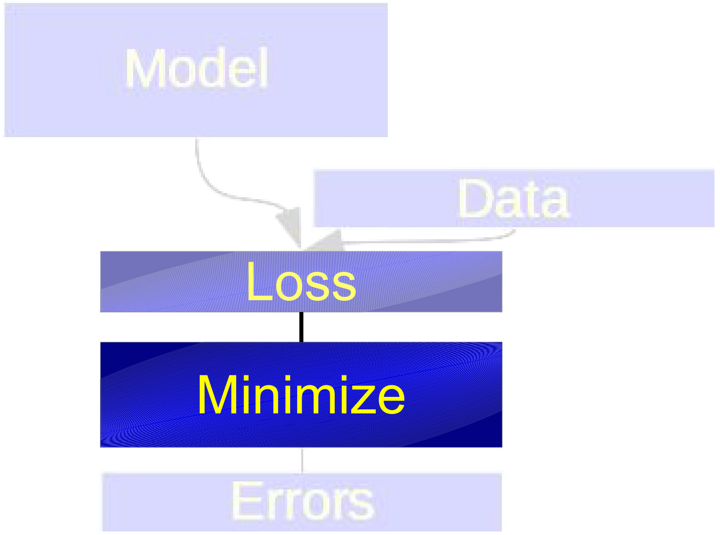
Complete fit

```python
import zfit

normal_np = np.random.normal(loc=2., scale=3., size=10000)

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

mu    = zfit.Parameter("mu",    1., -5,  5)
sigma = zfit.Parameter("sigma", 3.,  1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

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

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

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

param_errors = result.error()
```



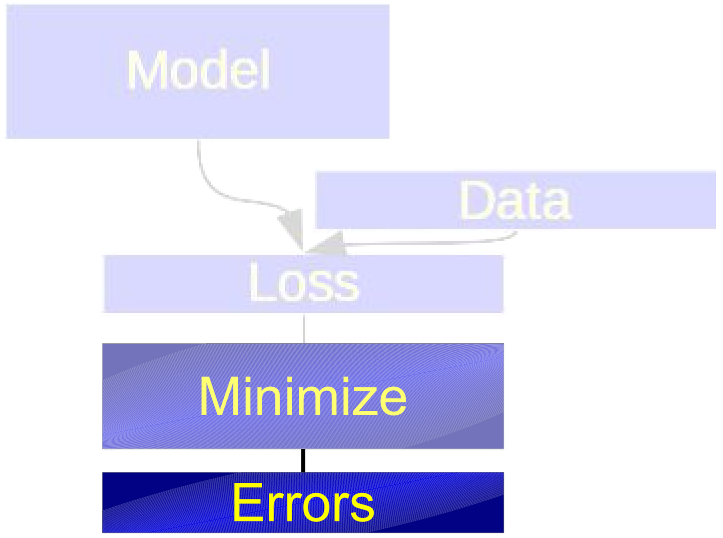
Complete fit

```
import zfit

normal_np = np.random.normal(loc=2., scale=3., size=10000)

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

mu    = zfit.Parameter("mu",    1., -5,  5)
sigma = zfit.Parameter("sigma", 3.,  1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

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

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

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

param_errors = result.error()
```



Complete fit

```
import zfit

normal_np = np.random.normal(loc=2., scale=3., size=10000)

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

mu    = zfit.Parameter("mu",    1., -5,  5)
sigma = zfit.Parameter("sigma", 3.,  1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)

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

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

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

param_errors = result.error()
```



Custom PDF

```python
from zfit import ztf


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

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

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

implement custom function

```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)
```

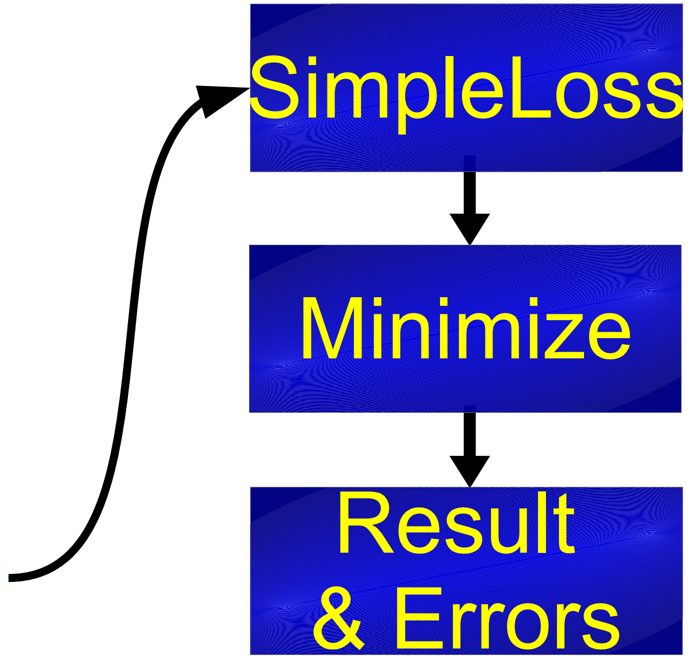
use functionality of model

Common API

Custom Loss

Define function that builds a loss
Returns Tensor

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



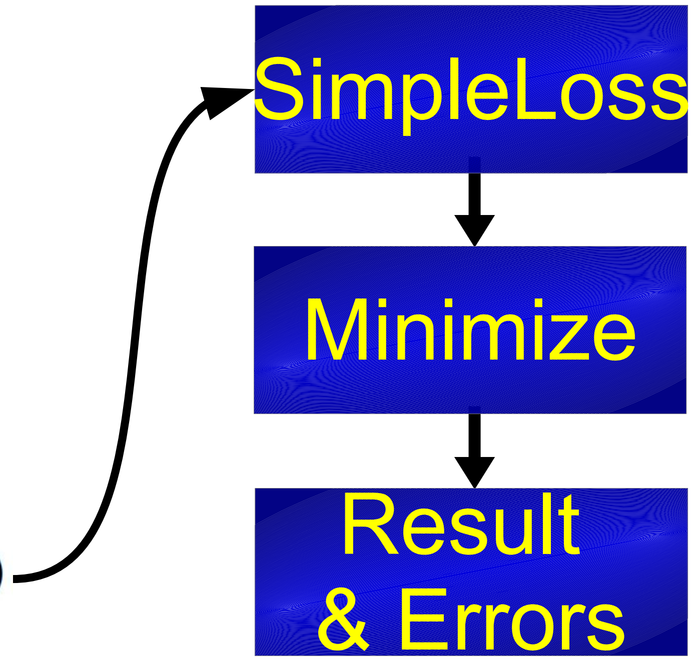
Common API

Custom Loss

Define function that builds a loss
Returns Tensor

Required to use zfit Parameter

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



zfit Parameter

- Has a name
- Has a *single* value
- Can have limits
- Can be a Tensor depending on anything

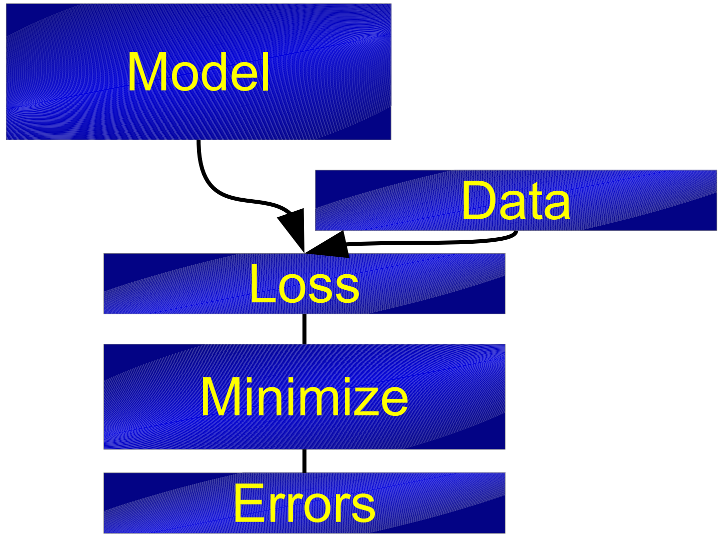
Works well for unbinned fits

But probably different bottlenecks

Binned fits

- Binned and template pdfs with the zfit structure
  - Binned data + continuous pdf?
  - How to use template pdfs?

- Approach so far:
  - Bin data with numpy
  - TF offers basically nothing



Numpy, TF and data

- Any python function can be wrapped (tf.py_function)
  - Allows to have binning as part of the graph
- Gradient not needed for binning
- TensorFlow Analysis
  - functional fitting library for unbinned amplitude fits
  - Also has binned with numpy: works

Numpy, TF and data

- Any python function can be wrapped (tf.py_function)
  - Allows to have binning as part of the graph
- Gradient not needed for binning
- TensorFlow Analysis
  - functional fitting library for unbinned amplitude fits
  - Also has binned with numpy: works

What data layout is feasible?

Template PDFs

- No experience so far
- Big question: efficient lookup table?

Investigation with TF Probability

Maybe implement own kernel

Problems with TF

- Feeding data to TensorFlow
  - feed_dict: convenient, but the *«least efficient way»*
  - Numpy arrays (constant Tensors): works, but not optimal
  - tf.Variable for sometimes changing data
  - Dataset: highly optimized for *small batches*
- *Dtype* problems since float32 is default
- Caching *between* session runs bad

Thinking aloud: wrapping TF as «scientific computing library»?

TensorFlow going to 2.0

- Lazy evaluation default
  - Switch back to graph mode
- Namespace moves to tf.backend
- Estimators handle graphs, sessions
- Lots of cleanup and new functions
  - vectorized_map
  - wrap_function: create graph once, acts as function for Tensors
  - Inheritable Variable
  - Focus on large scale distribution

TensorFlow going to 2.0

- Lazy evaluation default
  - Switch back to graph mode
- Namespace moves to tf.backend
- Estimators handle graphs, sessions
- Lots of cleanup and new functions
  - vectorized_map
  - wrap_function: create graph once, acts as function for Tensors
  - Inheritable Variable
  - Focus on large scale distribution

Definitely improving

Summary

Goal: Pythonic fitting with TensorFlow for HEP

- What we known:
  - Binned fits using numpy (also wrapped) work
  - TensorFlow changing, clearly improving
- What is needed:
  - Common parameter API
  - Efficient lookup for template PDFs
  - Binned data format
- What we work on:

  Bring binned data and template PDFs into zfit structure https://zfit.github.io/zfit/

zfit@GitHub

Gitter channel

zfit@physik.uzh.ch

Join the discussion!

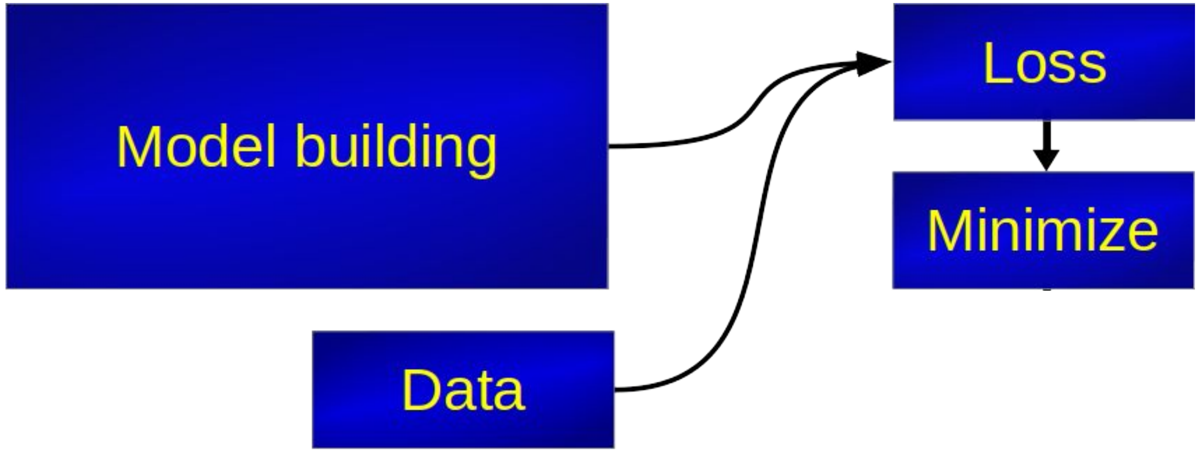
Backup Slides

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»

Scalable: TensorFlow

- Machine learning in a nutshell:
  - Build a model (a lot of matrix multiplications with simple non-linear functions in between) with 100k+ free parameters
  - Create a loss function (see how good/bad the predictions are)
  - Minimize it



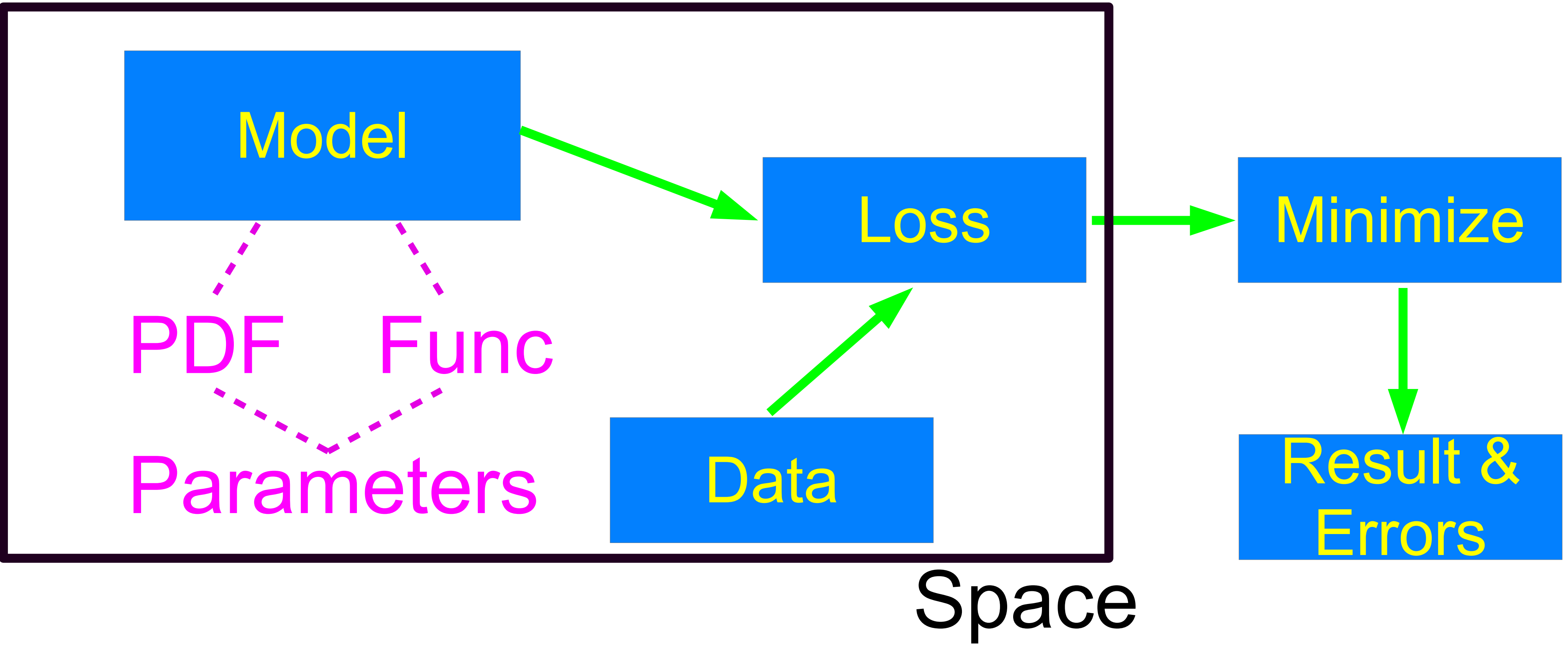
Pythonic: statistics tool «lauztat»

- Author: Matthieu Marinangeli
- WIP, pre-beta
- Python statistics tool for limits, significance etc.
  (~ RooStats)
- lauztat on Github with example notebooks using `zfit`

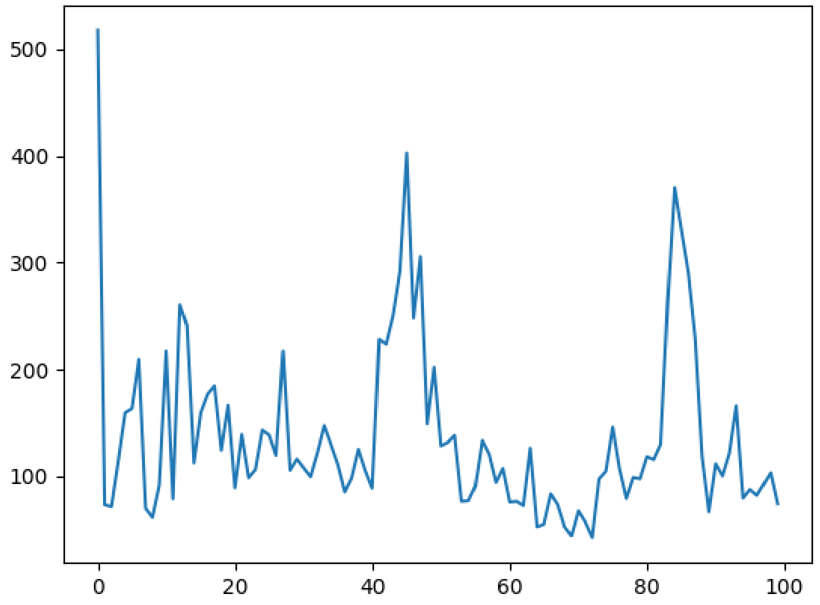
Pythonic: «phasespace»

- Author: Albert Puig
- Python tool for n-body phasespace generation (~ TGenPhaseSpace)

Fitting: complete structure



Performance toy example



Time per toy