# zfit

## scalable pythonic fitting

**Jonas Eschle** on behalf of zfit
jonas.eschle@cern.ch

FNSNF
Swiss National Science Foundation

University of Zurich[UZH]

# What is `zfit`?

Fitting for HEP

Fresh API, codebase

Pure Python library

# What is zfit?

Fitting for HEP

Fresh API, codebase

Wait…
Reinventing the wheel?

Pure Python library

# What is zfit?

Fitting for HEP

Fresh API, codebase
*(inspired by others)*

TensorProb
mlfit
TensorFlow Probability
scipy
Astropy
carl
probfit
RooFit
TensorFlow Analysis

Let's step back

# Python Model Fitting in HEP

# What is needed?

# 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,…)

# Fitting in Python

A lot of projects are around!

- RooFit
- HEP Python fitting projects
- Non-HEP

# Fitting in Python

A lot of projects are around!

- ~~RooFit~~
- ~~HEP Python fitting projects~~
- ~~Non-HEP~~

No feasible Python model fitting library for HEP

# Fitting in Python

A lot of projects are around!

- ~~RooFit~~
- ~~HEP Python fitting projects~~
- ~~Non-HEP~~

No feasible Python model fitting library for HEP

… but a lot to learn and build from!

zfit

# zfit: the project

## build *the* stable model fitting library for HEP

...the time has come

- Functionality limited to model fitting & sampling
- Use power & knowledge of existing libraries
- Build fresh from scratch
- Community invokation

# zfit: the project

## build *the* stable model fitting library for HEP

...the time has come

- Functionality limited to model fitting & sampling
- Use power & knowledge of existing libraries
- Build fresh from scratch
- Community invokation

API & workflow definition

Computational backend

*(Implementation)*

# API & Workflow

# API & Workflow: why

- High level libraries (statistics packages, amplitude fitters,...)
  - „code against an interface, not an implementation“

# API & Workflow: why

- High level libraries (statistics packages, amplitude fitters,...)
  - „code against an interface, not an implementation“
- Replace each component
  - Allow other libraries to hook in
  - Implement custom parts

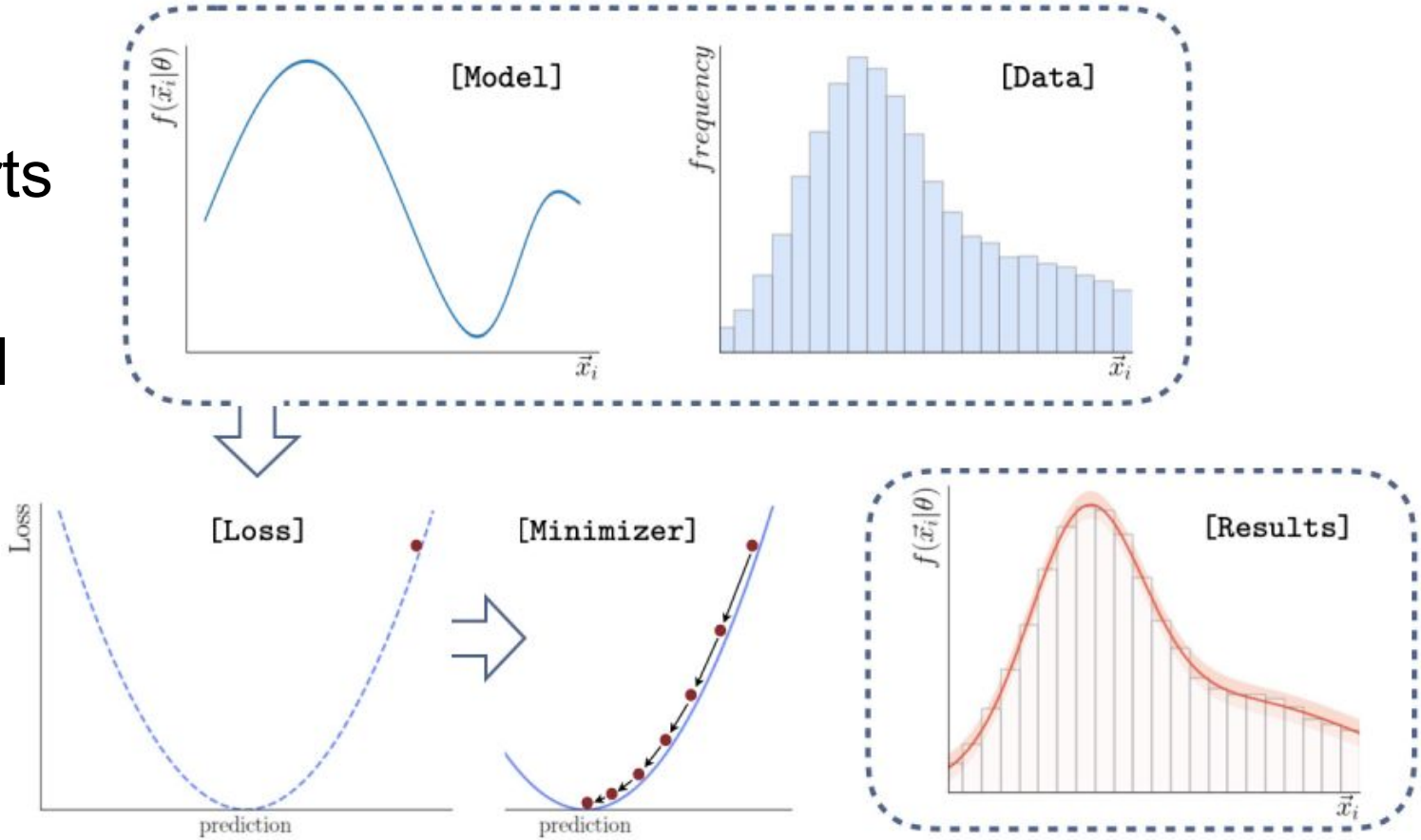
# API & Workflow: why

- High level libraries (statistics packages, amplitude fitters,...)
  - „code against an interface, not an implementation“
- Replace each component
  - Allow other libraries to hook in
  - Implement custom parts
- Consistency along case complexity
  - Simple case is straightforward, hard case is consistent

# Workflow

Five maximally independent parts

Well defined API

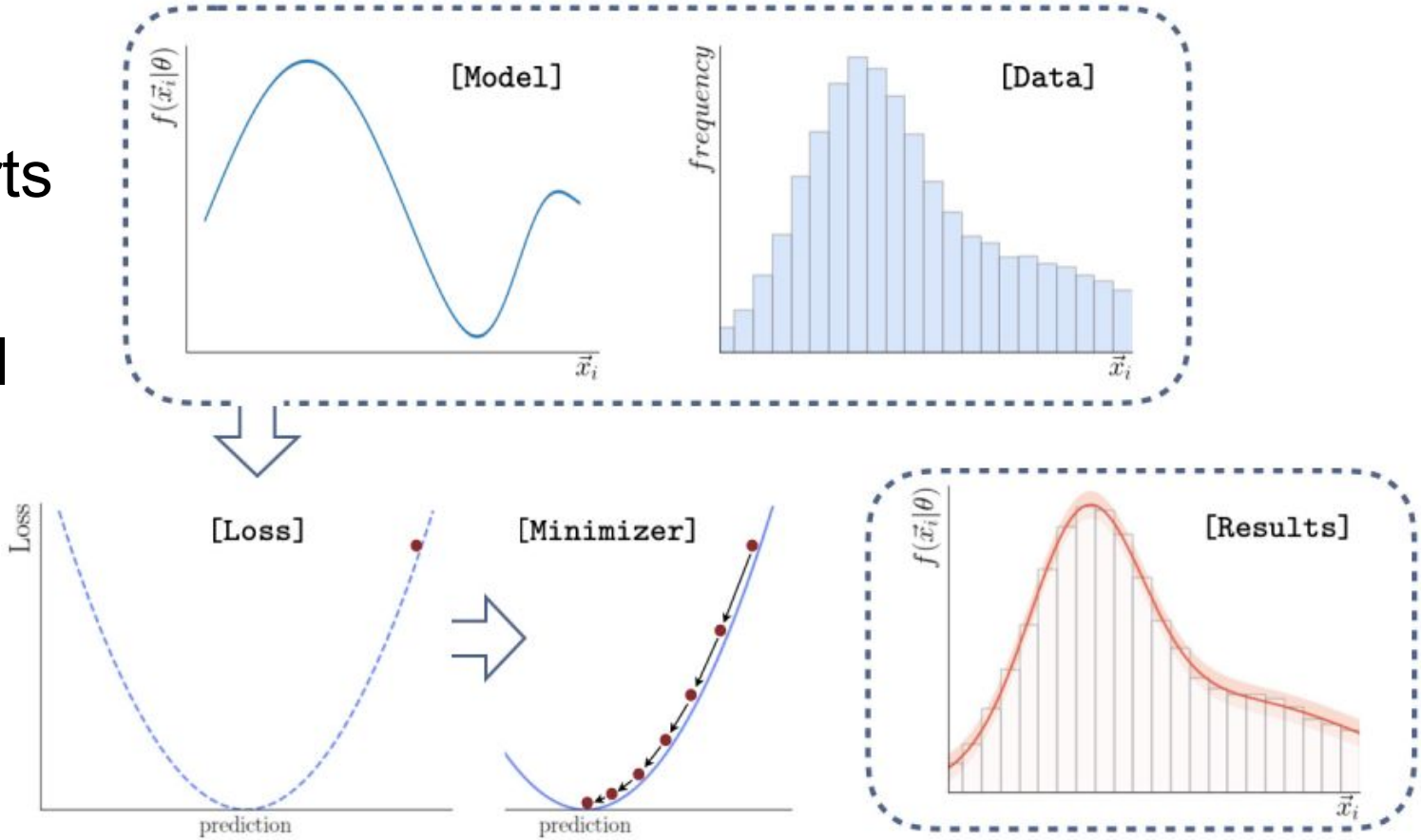


# Workflow

Five maximally independent parts

Well defined API

Example:
Library as
"loss builder"



# Workflow implemented

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

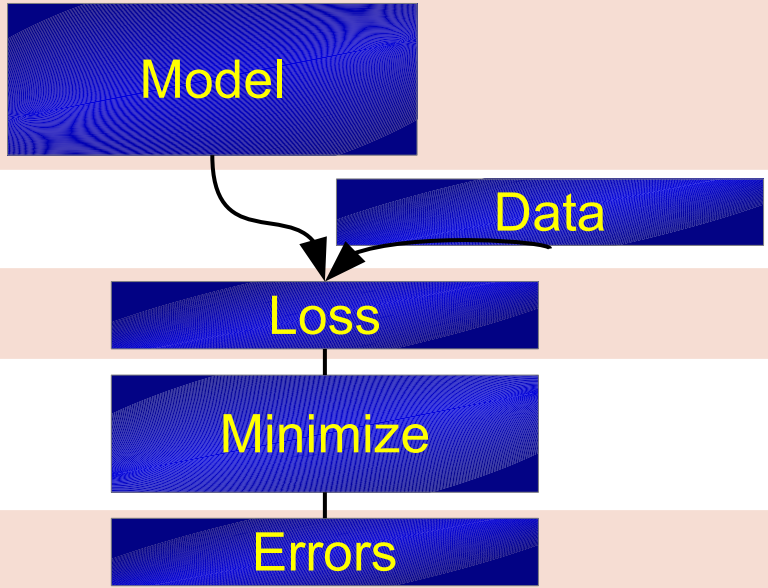
```
mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 10)
gauss = zfit.pdf.Gauss(mu=mu, sigma=sigma, obs=obs)
```

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

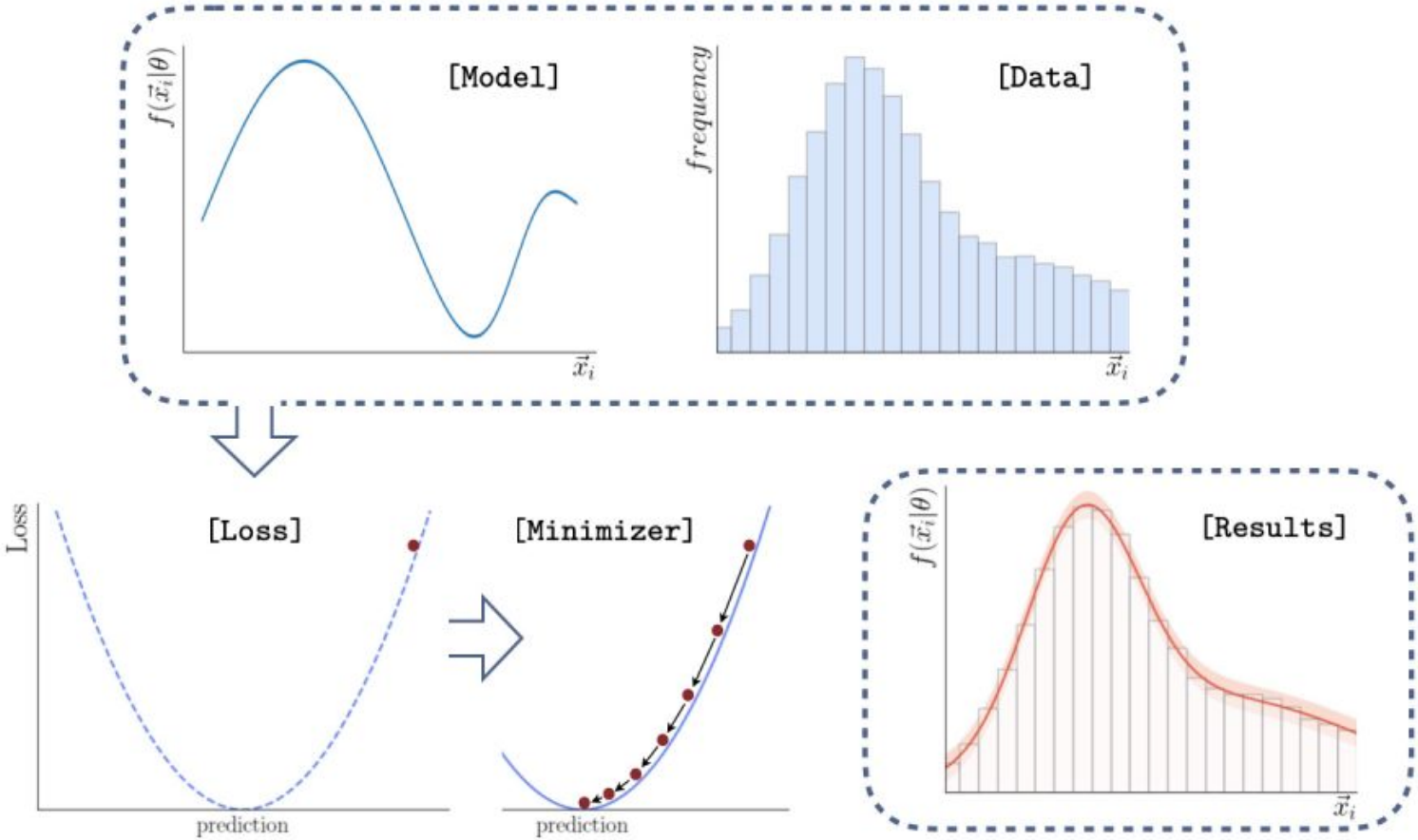
```
nll = zfit.loss.UnbinnedNLL(model=gauss, data=data)
```

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

```
param_errors = result.error()
```



# Workflow


[Model]
$f(\vec{x}_i|\theta)$
$\vec{x}_i$
[Data]
frequency
$\vec{x}_i$
[Loss]
Loss
prediction
[Minimizer]
prediction
[Results]
$f(\vec{x}_i|\theta)$
$\vec{x}_i$

# Computational Backend

~~Computational Backend~~


“And now for something completely different.”
Monty Python

~~Computational Backend~~

*(very brief) introduction to*

# *Deep Learning*

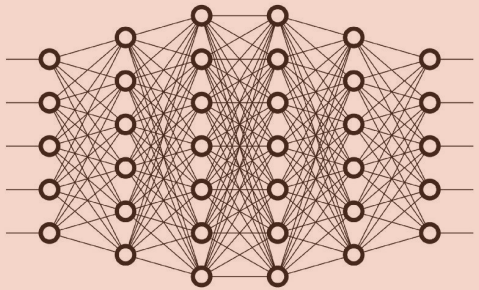
or Neural Networks

or Machine Learning

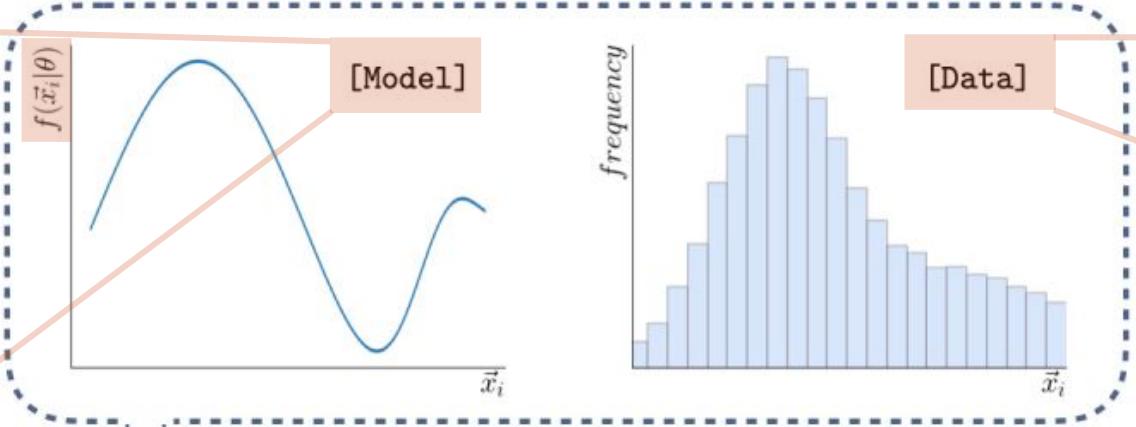
or Big Data…

# Deep Learning

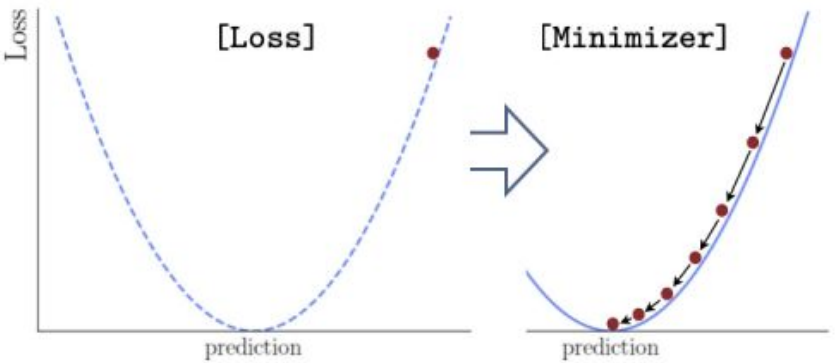
Neural Network



„One huge, complicated function“




„Big Data“




# 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 | | | | | |

# 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 | No real impact | | | | |

# 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 | No real impact | Optimizations for OOC calculations | | | |

# 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 | No real impact | Optimizations for OOC calculations | HEP trivial special case | | |

# 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 | No real impact | Optimizations for OOC calculations | HEP trivial special case | ~~Optimizers~~ „analytic“ derivatives! | |

# 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 | No real impact | Optimizations for OOC calculations | HEP trivial special case | ~~Optimizers~~ „analytic“ derivatives! | No support, but simple |

# 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 | No real impact | Optimizations for OOC calculations | HEP trivial special case | ~~Optimizers~~ „analytic“ derivatives! | No support, but simple |

# 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 | No real impact | Optimizations for OOM calculations | HEP trivial special case | ~~Optimizers~~ Free „analytic“ derivatives! | No support, but simple |

But...

what ***is*** a Deep Learning library?

# 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 | No real impact | Optimizations for OOM calculations | HEP trivial special case | ~~Optimizers~~ „analytic“ derivatives! | No support, but simple |

# 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 | No real impact | Optimizations for OOM calculations | HEP trivial special case | ~~Optimizers~~ „analytic“ derivatives! | No support, but simple |

Modern, high performance computing

# TensorFlow

- By Google, started in 2015 → quite new!
- Highly popular Deep Learning framework (130k ★, 4th on )
- Numpy-style syntax
- Uses static graphs
- Used in multiple physics libraries and analyses

# TensorFlow

- By Google, started in 2015 → quite new!
- Highly popular Deep Learning framework (130k ★, 4th on )
- Numpy-style syntax
- Uses static graphs
- Used in multiple physics libraries and analyses

...but many Deep Learning frameworks are similar

„There is no free lunch“

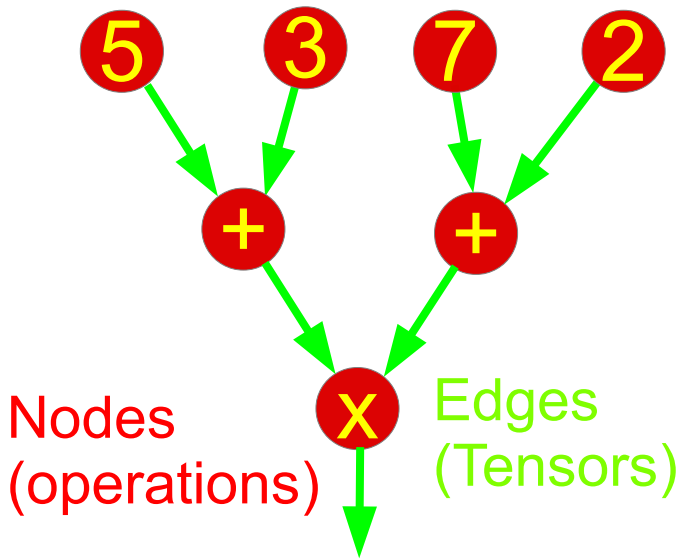
*or*

Graph-based computations

# Static graphs

Declarative: *define* computations, execute when needed

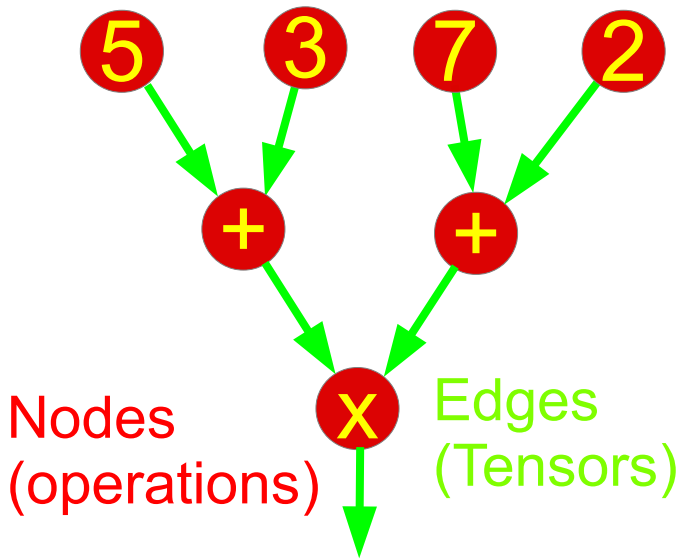
```
sum_1 = tf.constant(5) + tf.constant(3)
sum_2 = tf.constant(7) + tf.constant(2)
product = sum_1 * sum_2
```



# Static graphs

Declarative: *define* computations, execute when needed

```
sum_1 = tf.constant(5) + tf.constant(3)
sum_2 = tf.constant(7) + tf.constant(2)
product = sum_1 * sum_2
```




Static graph definition allows magic

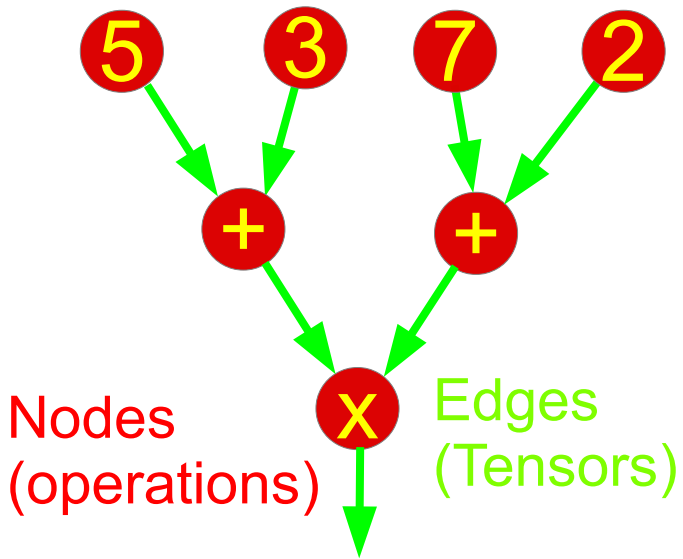
→ Gradient, optimizations, parallelization,…

Inside TF, hidden to end-user

# Static graphs

Declarative: *define* computations, execute when needed

```
sum_1 = tf.constant(5) + tf.constant(3)
sum_2 = tf.constant(7) + tf.constant(2)
product = sum_1 * sum_2
```




Static graph definition allows magic

→ Gradient, optimizations, parallelization,…

Python code (graph definition) is run ***ONCE***

***Can*** we express model fitting as static graphs?

# Graph elements

*… do not have to be constant!*

# Graph elements

*… do not have to be constant!*

**Parameters**

Can change their value

# Graph elements

*… do not have to be constant!*

**Parameters**

Can change their value

**Random numbers**

Generate newly on every graph execution: MC integration,…

# 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.

# 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

*Can* we express model fitting as static graphs?

***Yes!***

# Delegating the workload

| | C++ library (RooFit,...) | Numpy based | zfit |
|---|---|---|---|
| HEP specific content/API | | | |
| Models | | SciPy | TF Probability |
| Gradients | | | TensorFlow |
| Computational optimizations | | | |
| Parallelization/GPU | | Numba NumPy | |
| Low level handling | | python | |

# Delegating the workload

| | C++ library (RooFit,...) | Numpy based | zfit |
|---|---|---|---|
| HEP specific content/API | | | |
| Models | | SciPy | TF Probability |
| Gradients | | | |
| Computational optimizations | | | TensorFlow |
| Parallelization/GPU | | Numba NumPy | |
| Low level handling | | python | |

"Stepping on the shoulders of a giant"

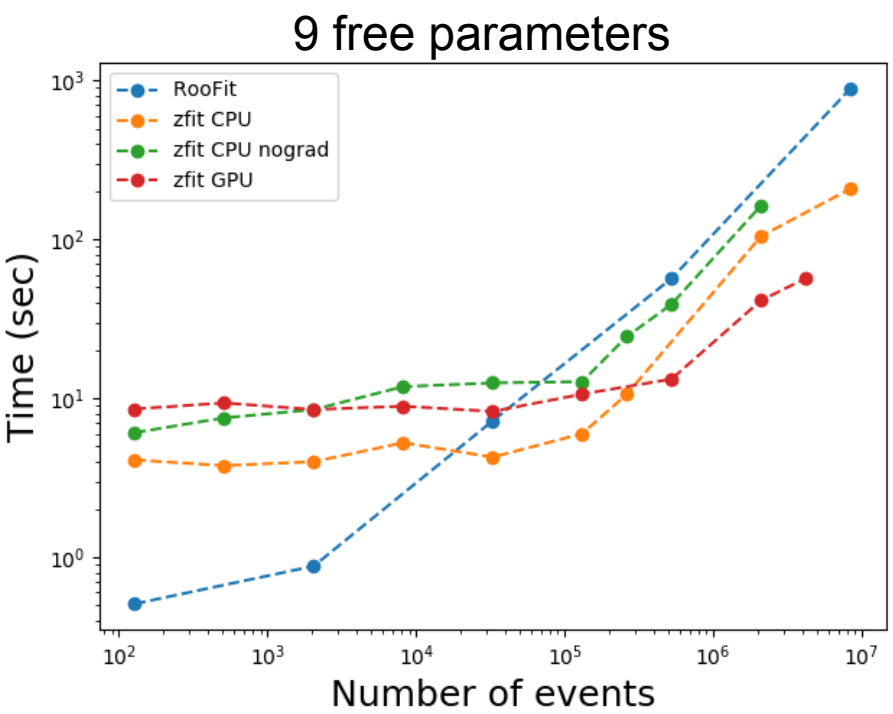
# Delegating the workload

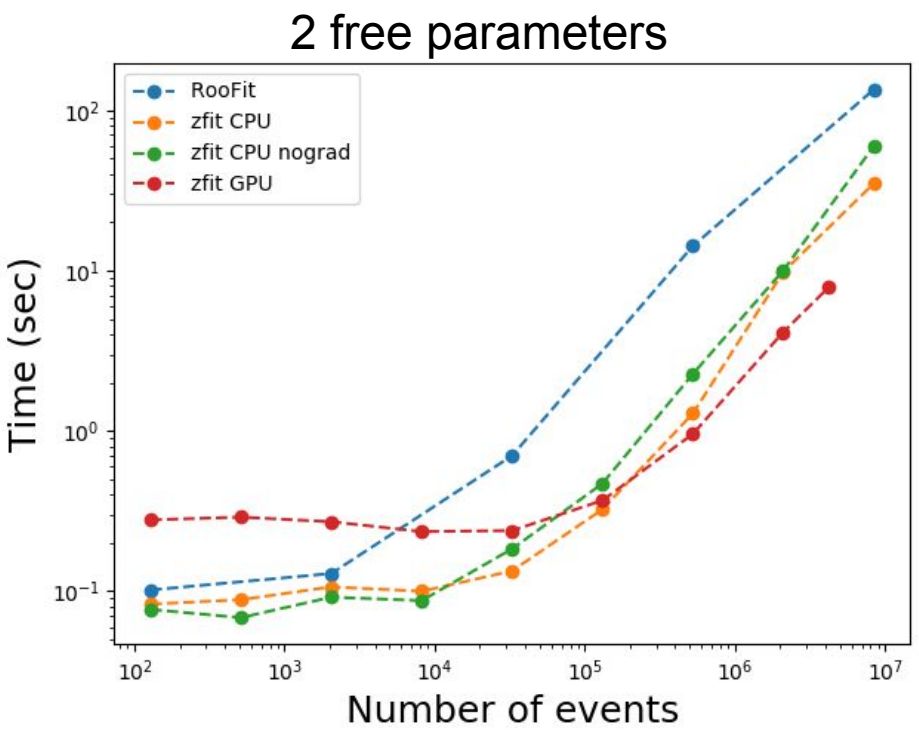
| | C++ library (RooFit,...) | Numpy based | zfit |
|---|---|---|---|
| HEP specific content/API | | | API & Workflow |
| Models | | SciPy | TF Probability |
| Gradients | | | |
| Computational optimizations | | | TensorFlow |
| Parallelization/GPU | | Numba NumPy | |
| Low level handling | | python | |

"Stepping on the shoulders of a giant"

# Performance

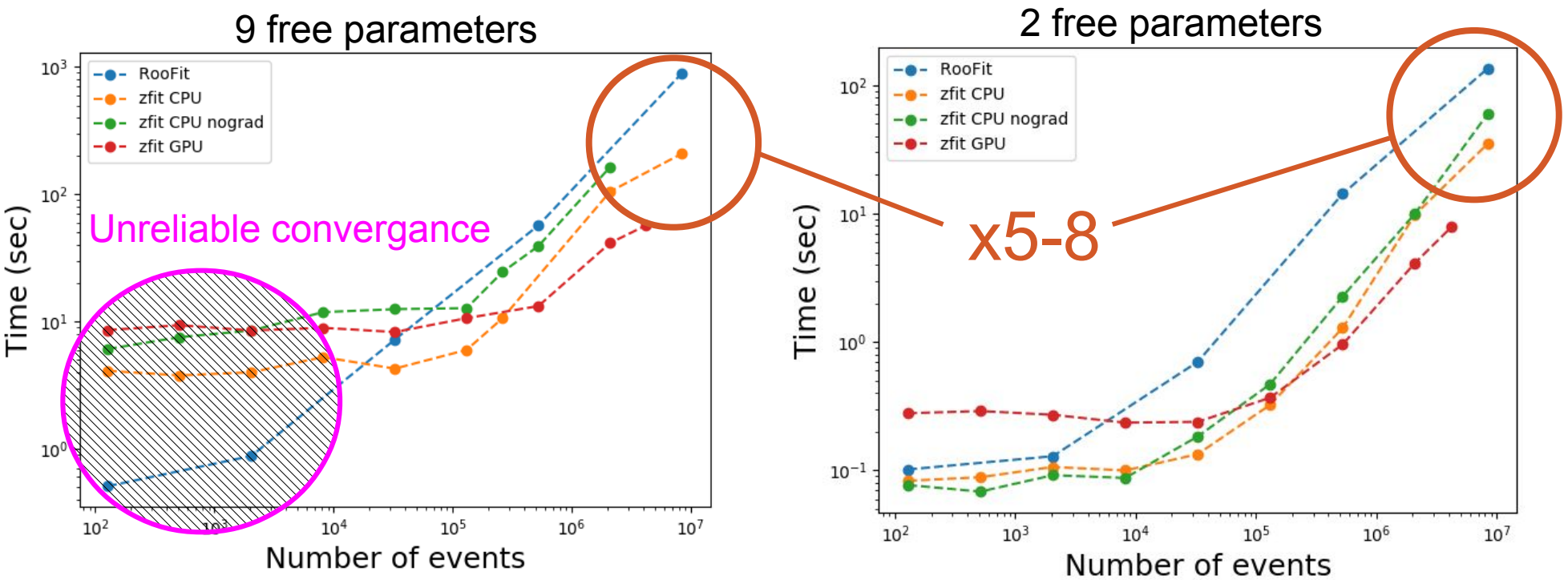
Sum of 9 Gaussians, fitting time of toy



# Performance

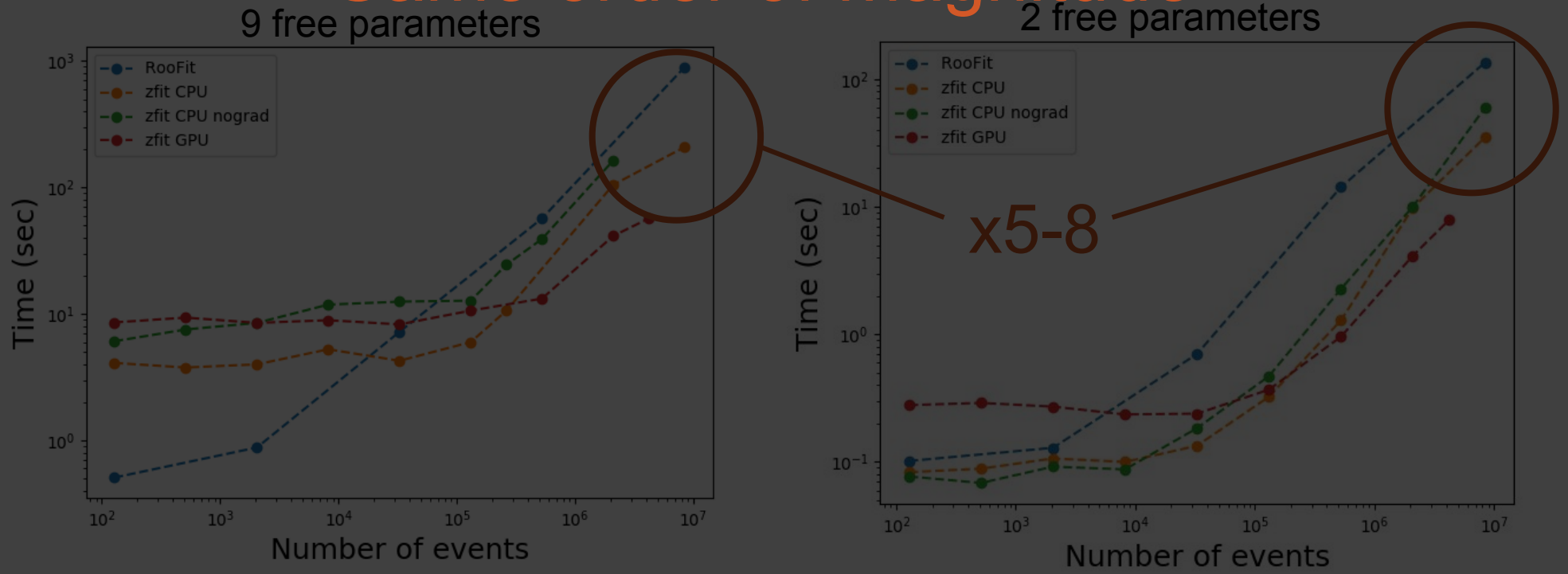
Sum of 9 Gaussians, fitting time of toy



# Performance

Sum of 9 Gaussians, fitting time of toy

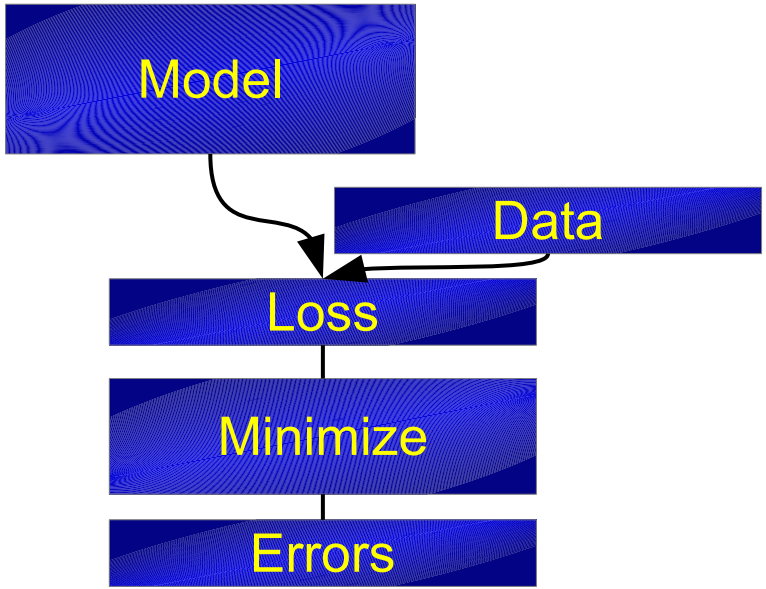
Same order of magnitude



# Implementation

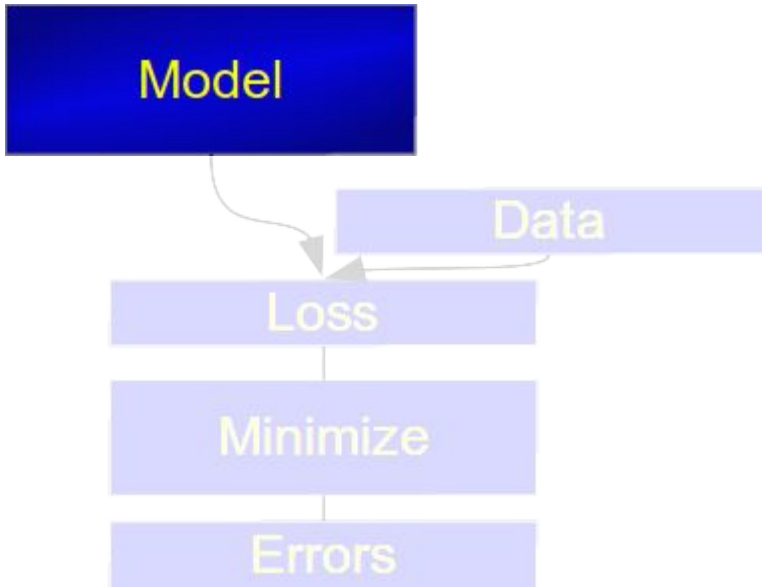
# Complete fit

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

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 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.error()
```



# Complete fit: Model

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

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 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.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

# Custom PDF

```
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)



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 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)
```

## Example of Base Classes *in general* inside zfit

```
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)
```

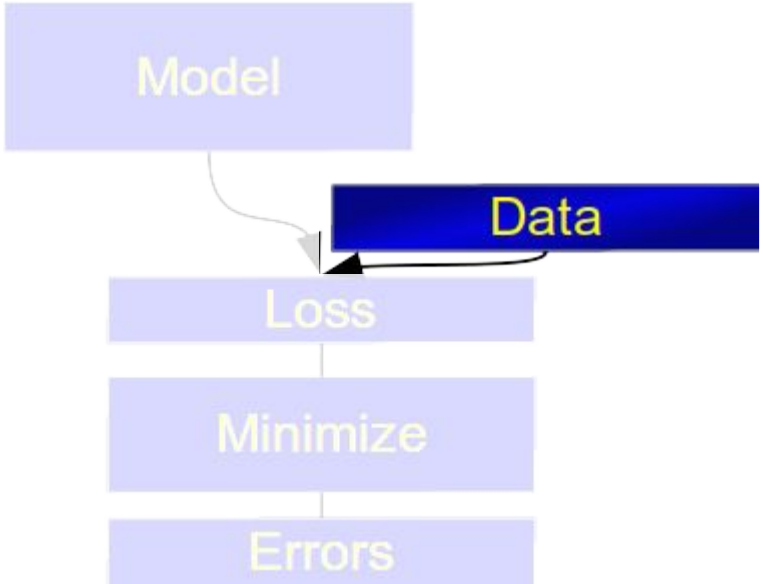
} use functionality of model

# LHCb Angular Analysis

```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_k, costheta_l, 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
```

# Complete fit: Data

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

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 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.error()
```



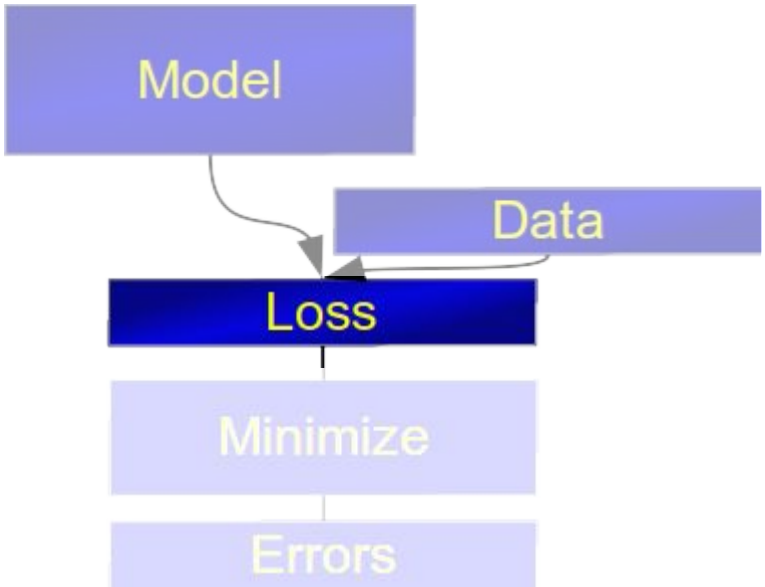
# Data

- From different sources
  - Numpy, Pandas, ROOT, …
- No pre-processing (use e.g. Pandas)
- Sampled from a model (toy studies)

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

# Complete fit: Loss

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

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 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.error()
```



# Simultaneous fit

```
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)
```

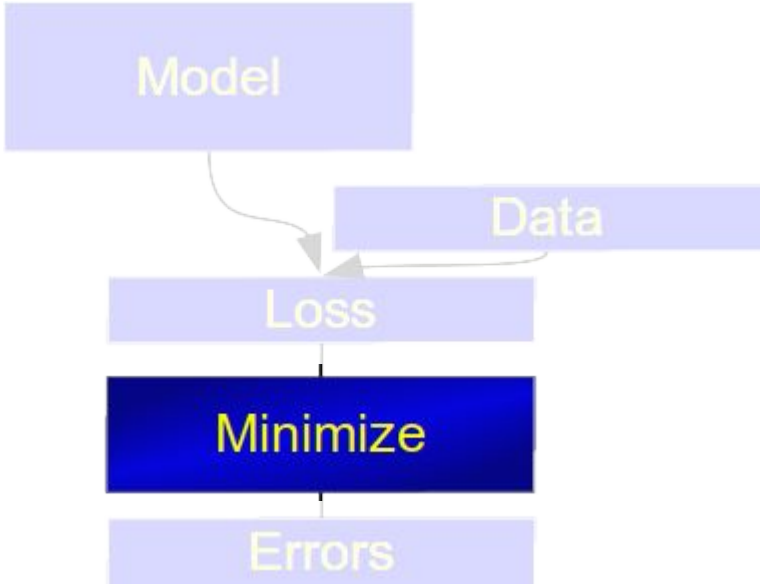
shared parameters

```
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
```

Both work

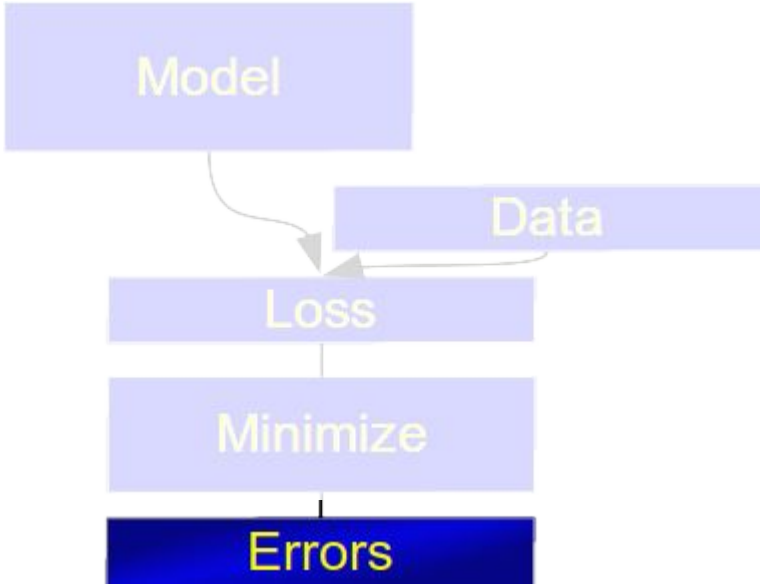
# Complete fit: Minimization

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

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 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.error()
```



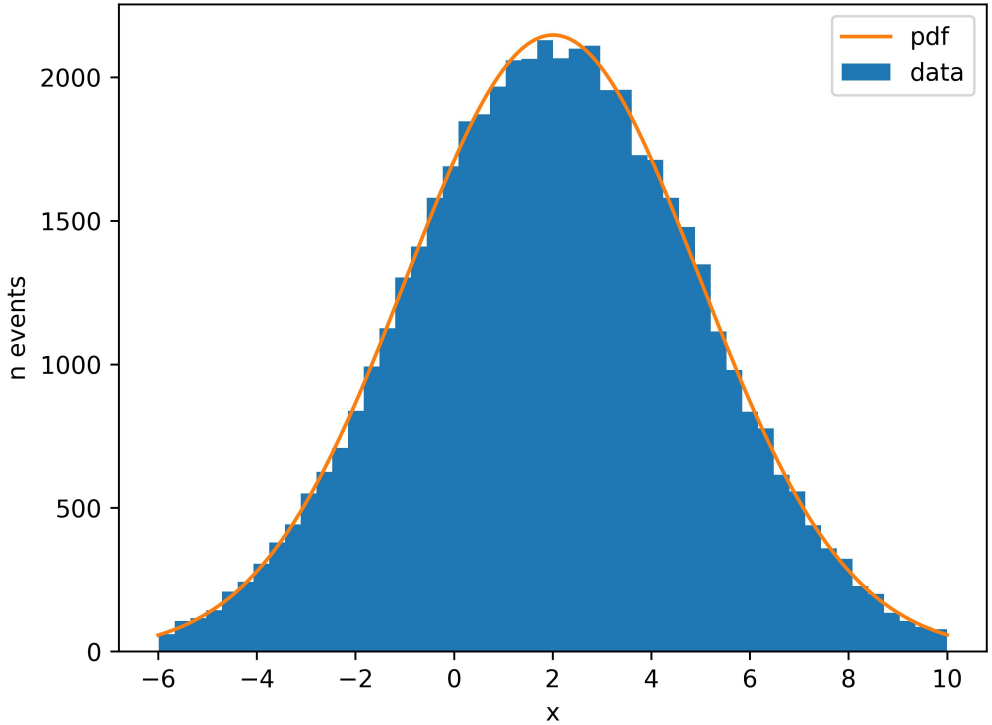
# Complete fit: Result

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

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

mu = zfit.Parameter("mu", 1.2, -4, 6)
sigma = zfit.Parameter("sigma", 1.3, 0.1, 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.error()
```

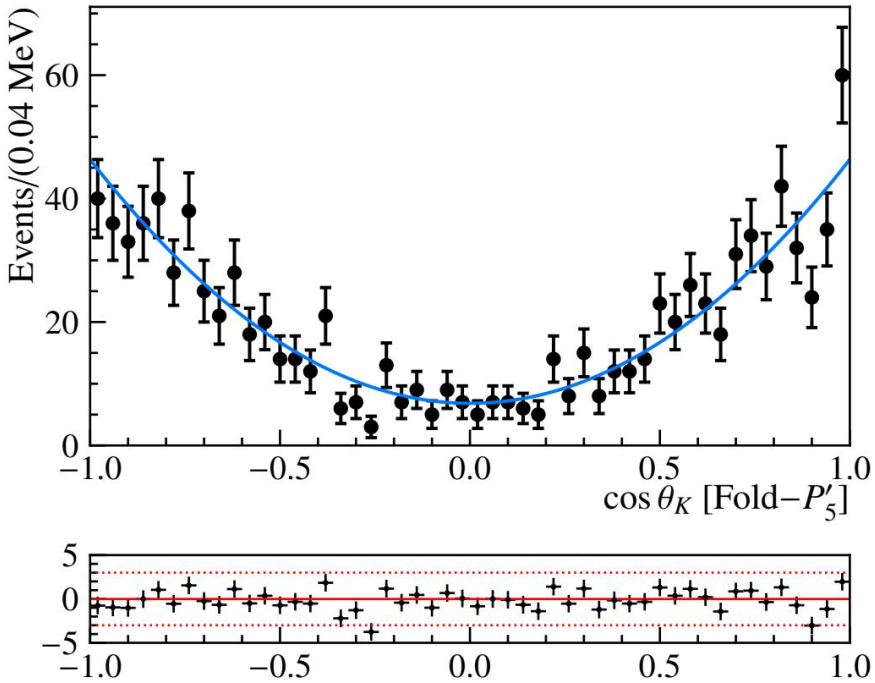


# Complete fit: plots

## Gaussian example




## Angular Analysis



# universe

**zfit-physics**

HEP specific content, for community contribution

**phasespace**

generate particles in phasespace

**zfit-amplitude**

WIP, high level library for Amplitude fits

affiliated

**TensorFlowAnalysis**

Amplitudes in TF; use model in zfit

**flamedisx**

Independent (bindings WIP)

# universe

**zfit-physics**
HEP specific content, for community contribution

**phasespace**
generate particles in phasespace

**zfit-amplitude**
WIP, high level library for Amplitude fits

affiliated

**TensorFlowAnalysis**
Amplitudes in TF; use model in zfit

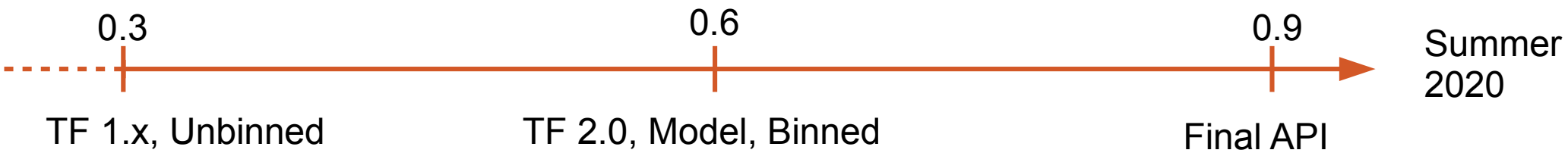
**flamedisx**

Next talks

Independent
(bindings WIP)

# zfit: status

Public beta stage *(pip/conda install)*




Focus on
- stable API & workflow
- core implementations
- interface & base classes

# Getting involved

Who

TensorFlow/Python/Fitting background welcome

What

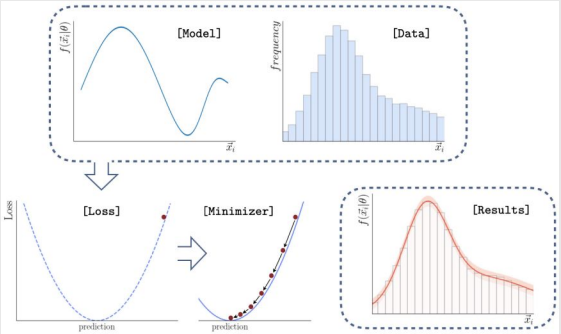
- Discussions (API, features, …): zfit-development
- Implement a part (dumping, integration,pdfs,…)
- Use it; ask; wish; criticize

*Hot topic: binned fits*

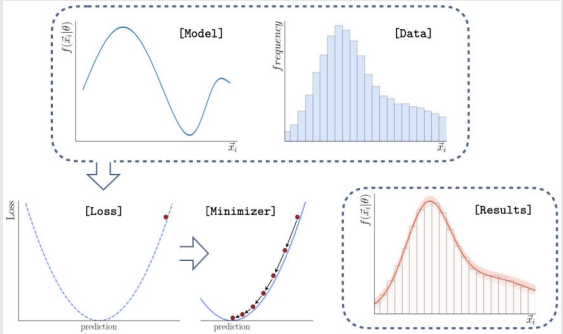
scalable

pythonic

fitting



scalable pythonic fitting




Try it out: https://github.com/zfit/zfit-tutorials

# Backup Slides

# https://zfit.github.io/zfit/

zfit@GitHub

Gitter channel

zfit@physik.uzh.ch

**Join the discussion!**

# RooFit

- *Limited customization and extendibility*
- *Sub-optimal scalability for ever larger datasets and modern computing infrastructure*

- Isolated, aging ecosystem, no cutting-edge software
- Not Python native
  - *Memory allocation errors*
  - *Arbitrary C++ limitations*
  - *No real integration into the Python ecosystem*

# 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,...

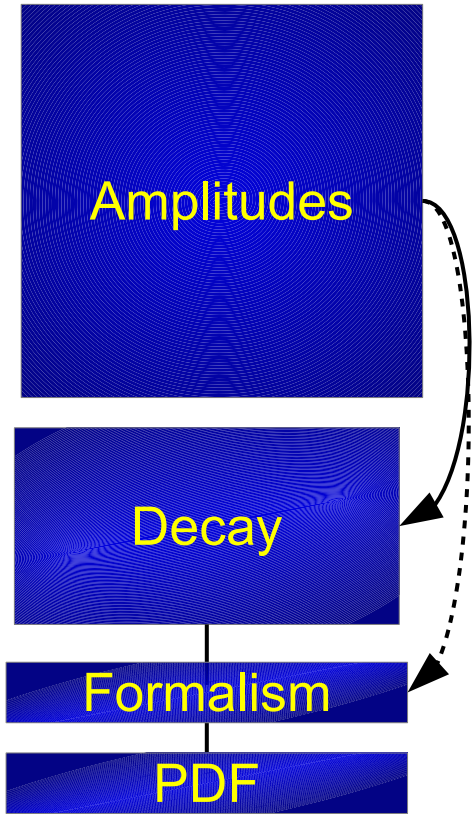
- Lack of specific HEP features
  - *Normalization: specific range, numerical integration,...*
  - *Composition of models*
  - *Multiple dimensions*
  - *Custom models*
- Irrelevant functionality supported in API
  - Survival function, …

# TFA: approach & differences

- Build «optimized» TensorFlow
  - accept-reject as tf.while_loop, Dataset input,...
- ...and hide the tedious, unambiguous parts
  - automatic normalization, Tensor cache, ...
- Well defined structures, e.g.
  - String name order (like columns) in PDFs, data, limits,...
    - pdf(„x“) * pdf(„y“) => pdf(„x“, „y“)
      1-dim 1-dim 2-dim
  - Local/recursive dependency resolution of Parameters

# Example amplitude

```
RESONANCES = [('rho(770)', ('pi-', 'pi0'), bw_amplitude),
              ('K(2)*(1430)0', ('K+', 'pi-'), bw_amplitude),
              ('K(0)*(1430)+', ('K+', 'pi0'), bw_amplitude),
              ('K*(892)+', ('K+', 'pi0'), bw_amplitude),
              ('K(0)*(1430)0', ('K+', 'pi-'), bw_amplitude),
              ('K*(892)0', ('K+', 'pi-'), bw_amplitude)]

COEFFS = {...}

D2Kpipi0 = Decay('D0', ['K+', 'pi-', 'pi0'])

for res, children, amp in RESONANCES:
    D2Kpipi0.add_amplitude(res, children, amp, COEFFS[res])

formalism = ThreeBodyDalitzFormalism("Zemach B Frame")

pdf = D2Kpipi0.create_pdf(name="D2Kpipi0", formalism=formalism)
```



# zfit project

- zfit: stable core
  - Unbinned fits, binned WIP
  - n-dim models with integral, pdf, sample
- zfit-physics: HEP specific content
  - BreitWigner, DoubleCB,…
  - Faster development, more content
  - Ideal for contributions
    - Auto testing of new pdfs/func
    - Contribution guidelines

# 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 **hidden** backend, uses graphs
  - numpy-like syntax
  - parallelization on CPU/GPU, analytic gradient,...
- Writing functions simple for users *and* developers
  - No Cython, MPI, CUDA,... for *state-of-the-art performance*
  - No low-level maintenance required!
- Used in multiple physics libraries and analyses

# 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»

# 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

# Model, loss building

## sum of two pdfs

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

From classical

## 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)
```

to more TensorFlow

## simultaneous loss

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

# 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)
```

=> use all of zfit functionality like minimizers

up to pure TensorFlow

# Model building

```
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

```
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)
```

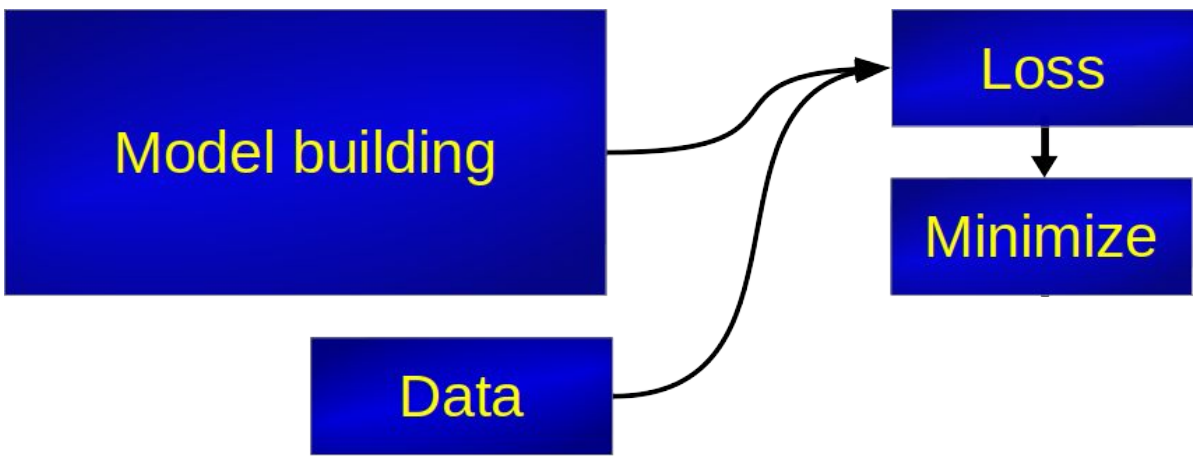
shared parameters

```
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

# 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



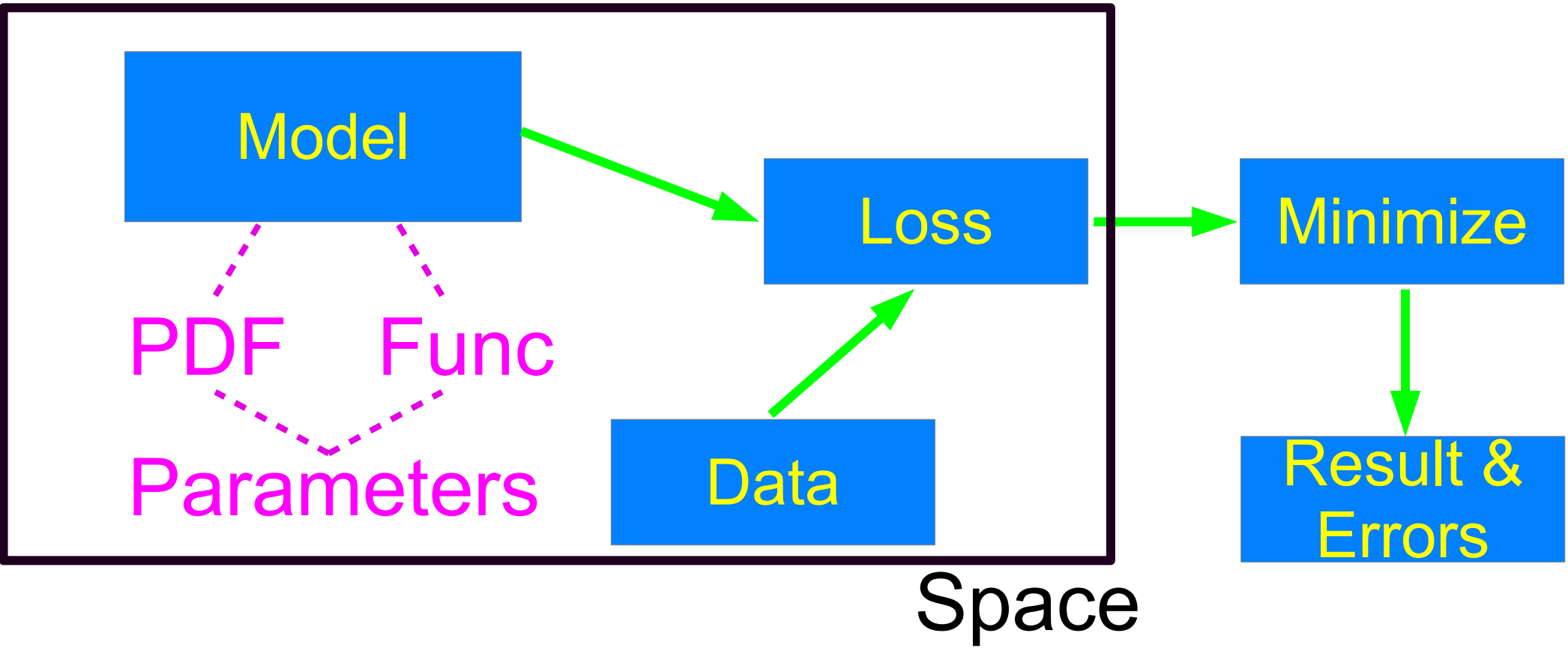
# Pythonic: statistics tool «lauztat»

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

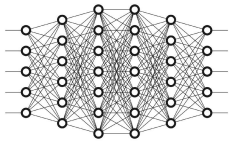
# Pythonic: «phasespace»

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

# Fitting: complete structure



# Sources

- Neural network: 
https://cdn-media-1.freecodecamp.org/images/Q8MdDayhWcsEz9A2fItm7lFEovErXC4a8L7S
-