pyhf introduction

Belle-II pyhf workshop (🤯)

Lukas Heinrich March 3rd 2023

Technische Universität München

ODSL ORIGINS Data Science Lab

Introduction

Thanks a lot for the invitation & organization of this workshop!

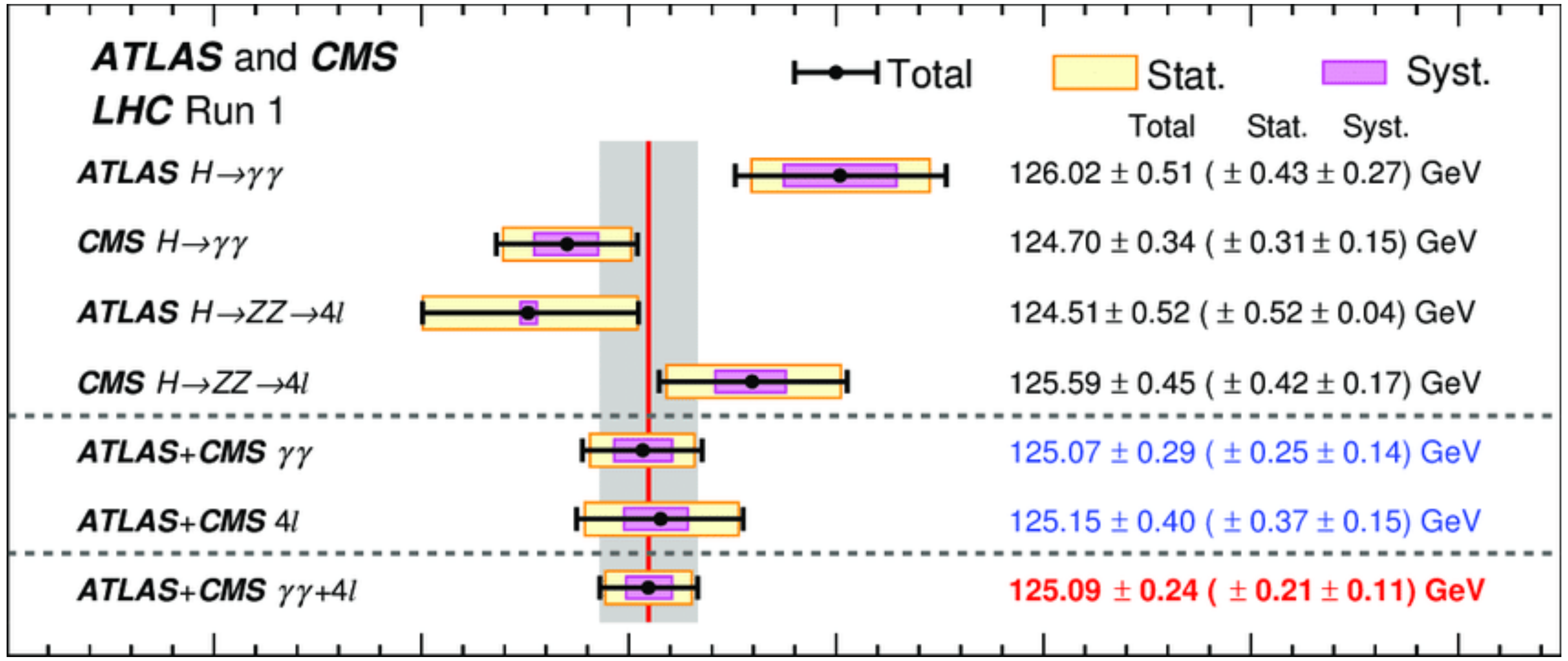
Particularly thanks to Sally & Florian!


CERN
LHC
[Large Hadron Collider]
ALICE
LHCb
ATLAS

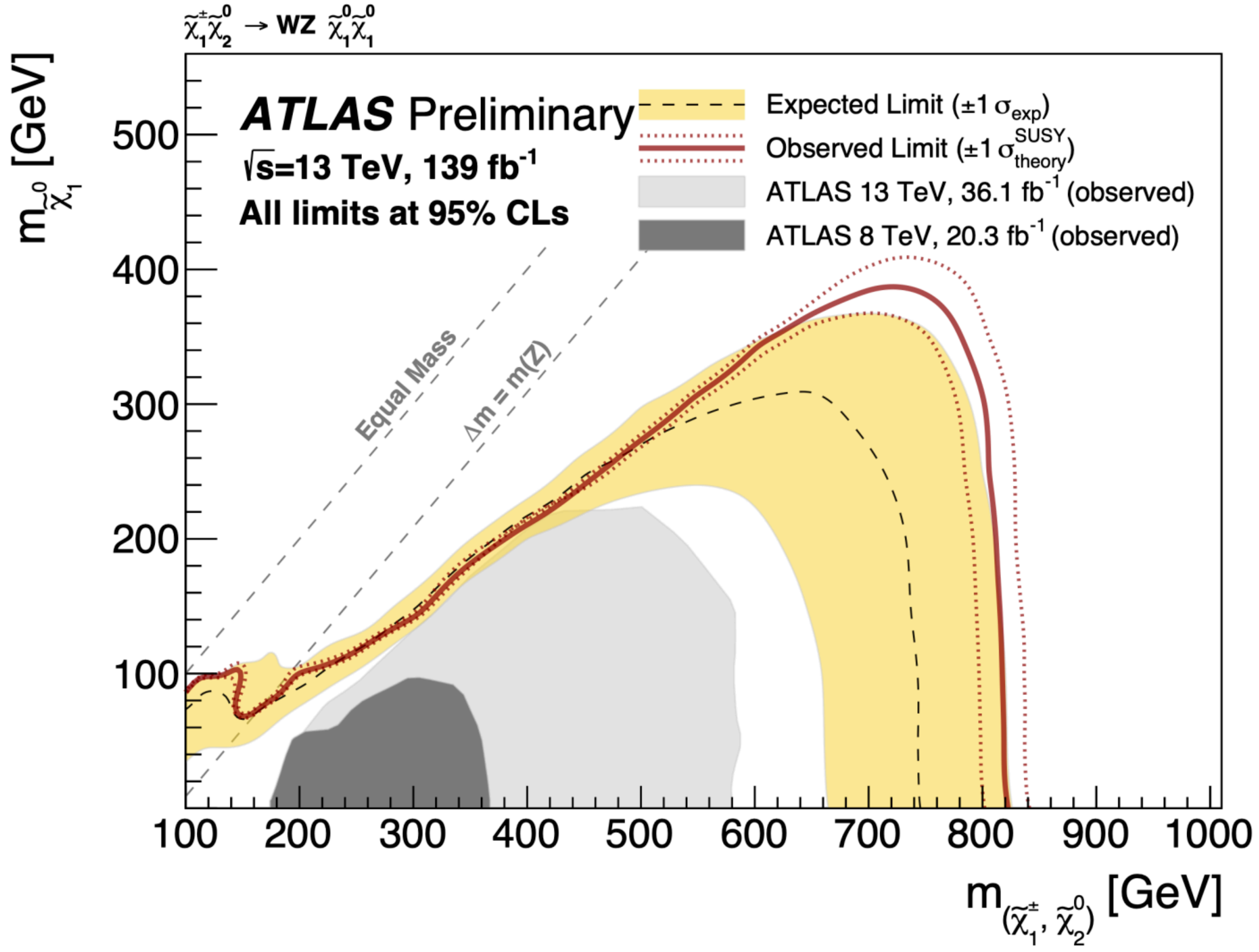

Mt. Tsukuba
BELLE II
~ 1 km
KEKB Ring
~ 1.5 km

Goals

Measure the Standard Model

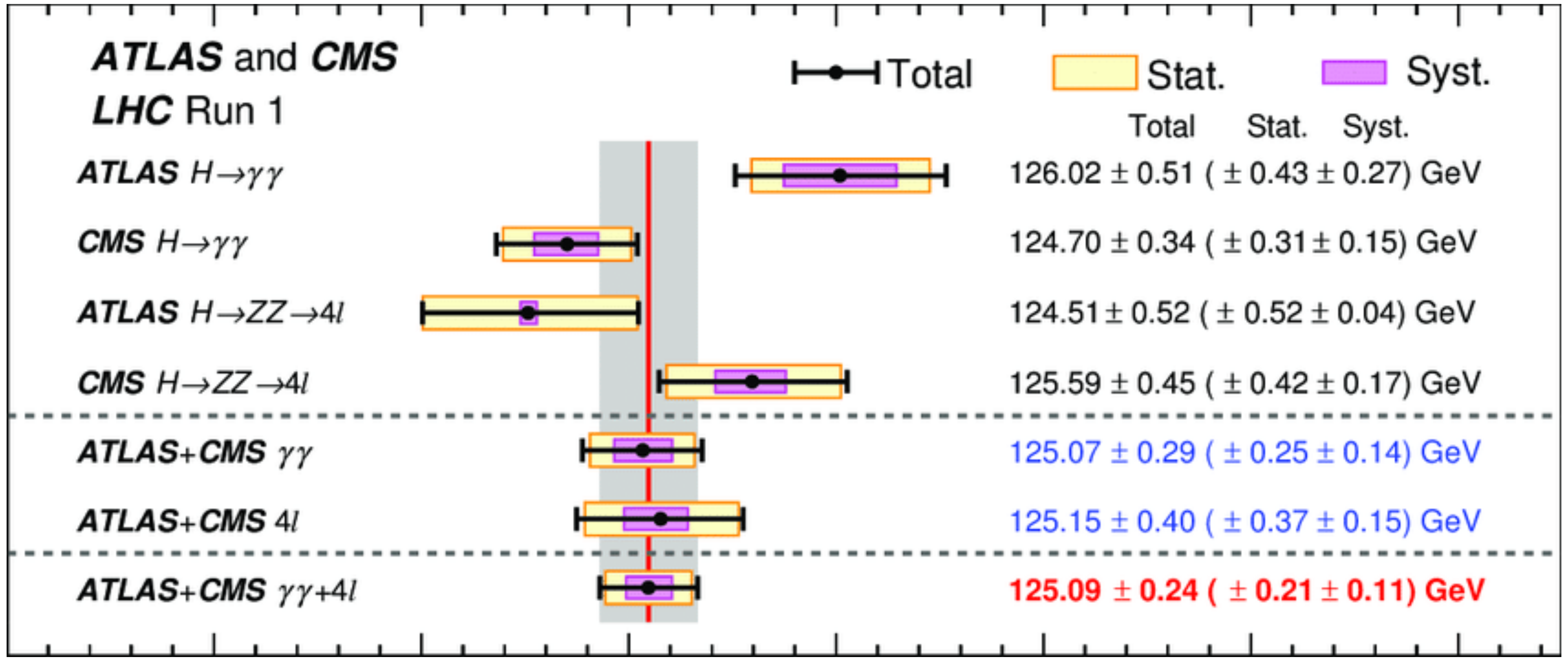


Look for the Unknown

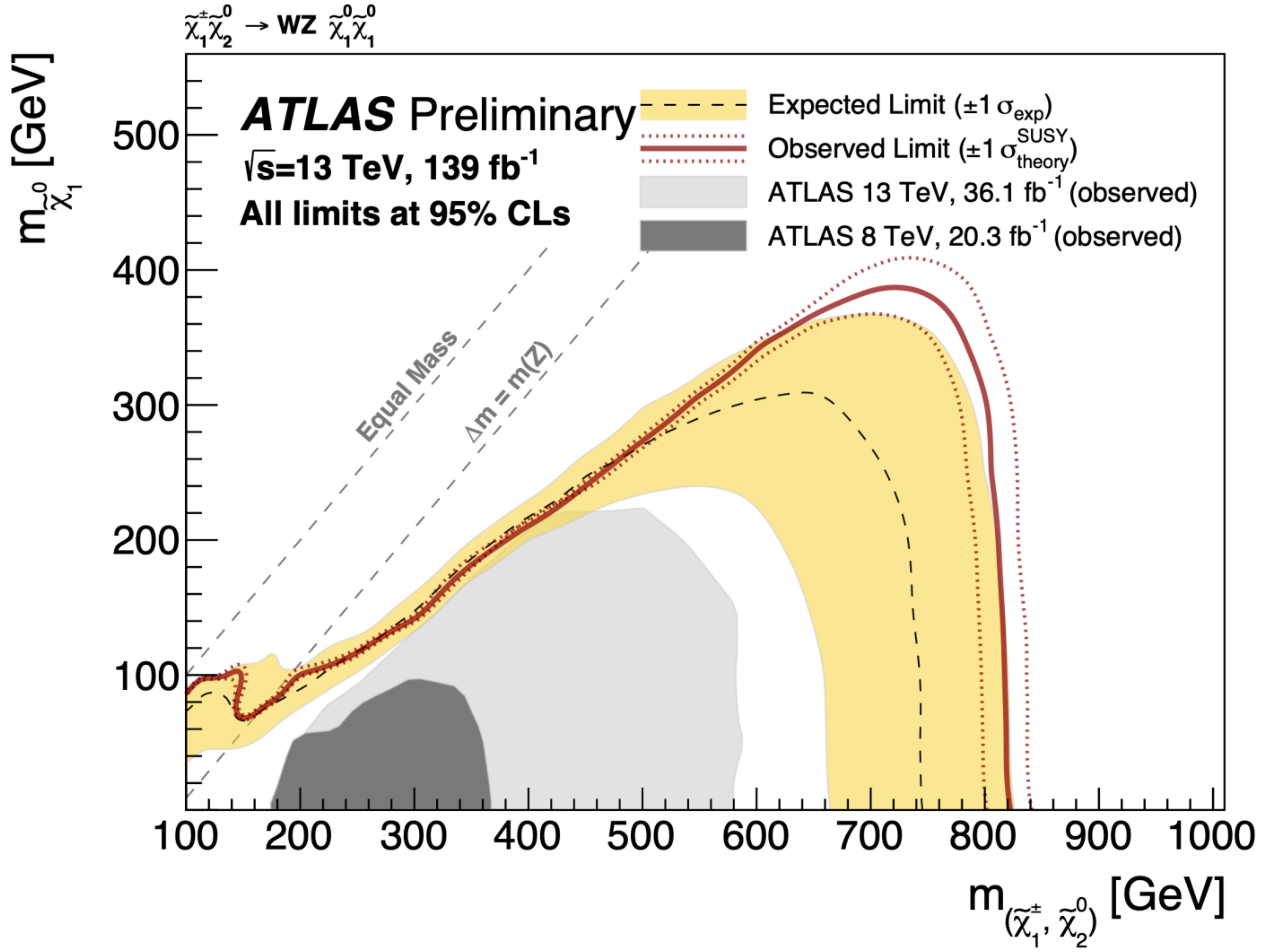


Goals

Measure the Standard Model



Look for the Unknown



The language in which we communicate our science is statistics

The Key Ingredient

The most important object in statistics is the description of the measurement as a data generating process

parameters

$$p(\mathscr{D}\,|\,\theta)$$

experimental data

The Key Ingredient

The **statistical model** $p(x\,|\,\theta)$ **is where the physics lives.**
Once we have it, we can do all kinds of statistics

$$p(\mathscr{D}\,|\,\theta) \to \hat{\theta},\ \ [\theta_-, \theta_+]$$

Frequentist Inference

$$p(\theta\,|\,\mathscr{D}) = \frac{p(\mathscr{D}\,|\,\theta)p(\theta)}{p(\mathscr{D})}$$

Bayesian Inference

interesting discussion, which to use, but remember physics content is the same (!)
and defined by the ***common*** $p(x\,|\,\theta)$

Repeated Experiments

The core of HEP experiments is the collection of an

i.i.d. sample

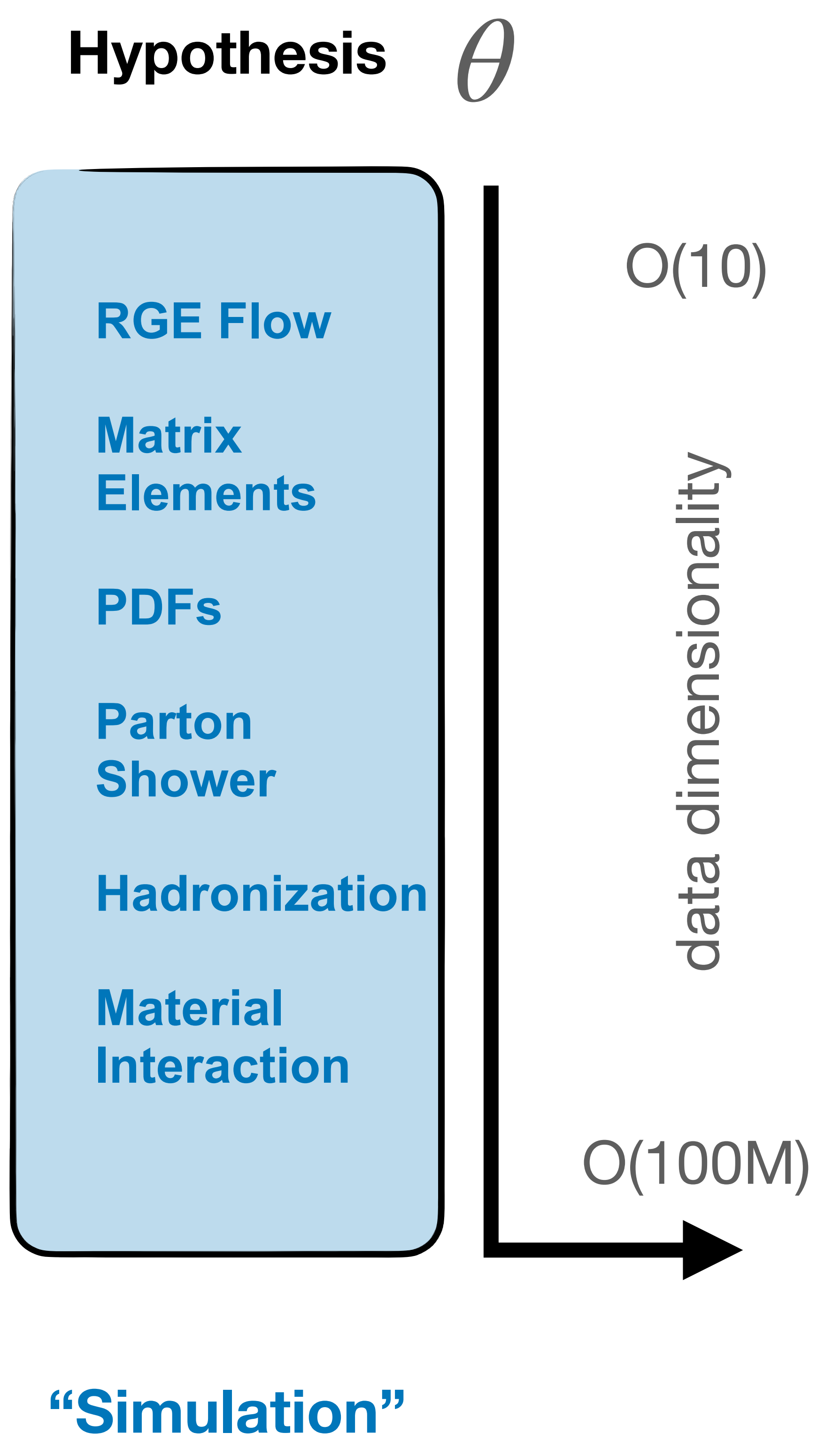
independent ***identically distributed***

$$p(\mathscr{D} \,|\, \theta) = \prod_{x_i \in \mathscr{D}} p(x_i \,|\, \theta)$$

We can build the dataset-wide model from the per-event model

The bad news

The problem in HEP: we do not know the detector-level per-event model in closed form, such that we could evaluate $p(x \mid \theta)$



sum over all possible histories

$$p(x \mid \theta) = \int dz\ p(x \mid z_h)\ p(z_h \mid z_p)\ p(z_p \mid \theta)$$

But: we can sample from this model without any problems (MC simulation)

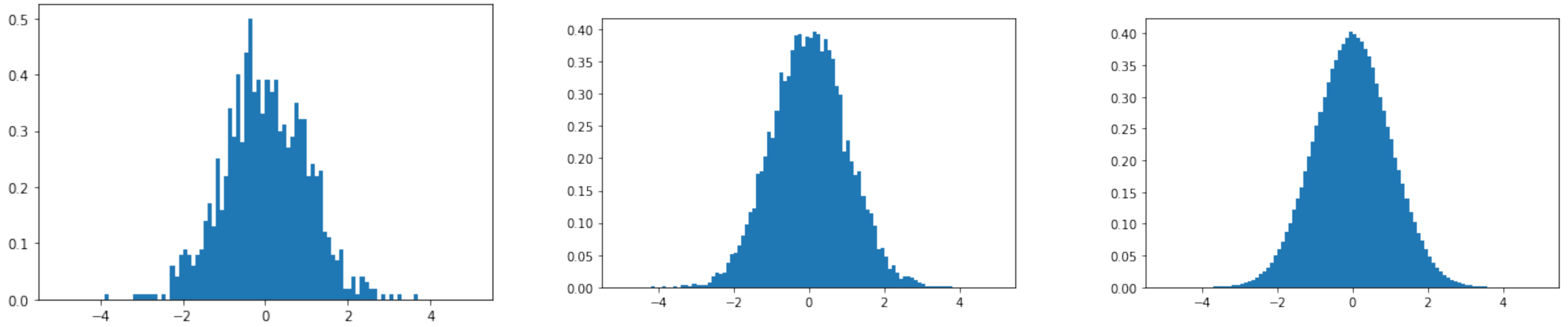
$$x \sim p(x \mid \theta)$$

Empirical Density Estimate

With samples from a density $x \sim p(x \mid \theta)$ we can construct an empirical density estimate

$$\{x_i\} \rightarrow \hat{p}(x \mid \theta)$$

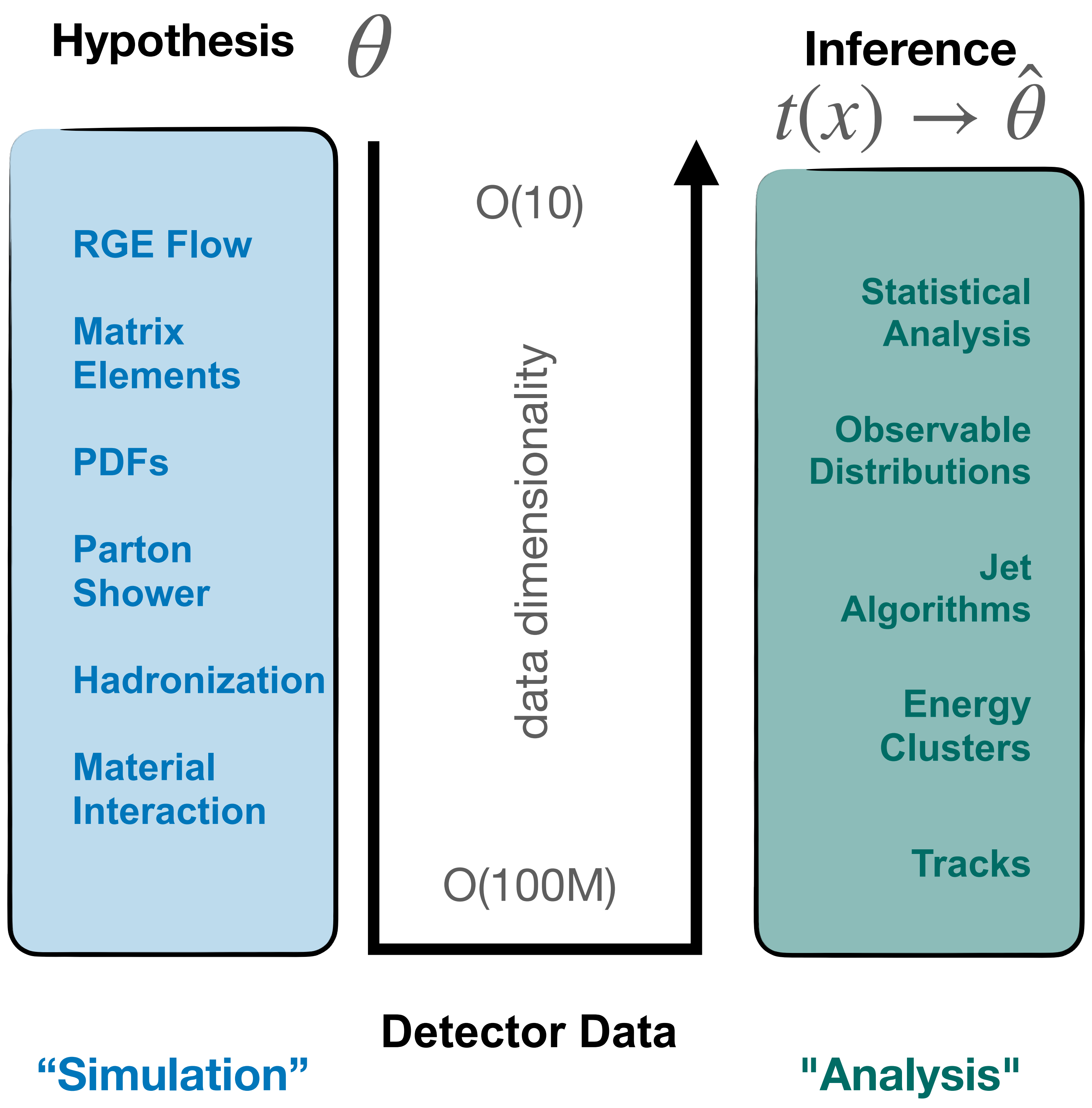
e.g. using a histogram as an approximation



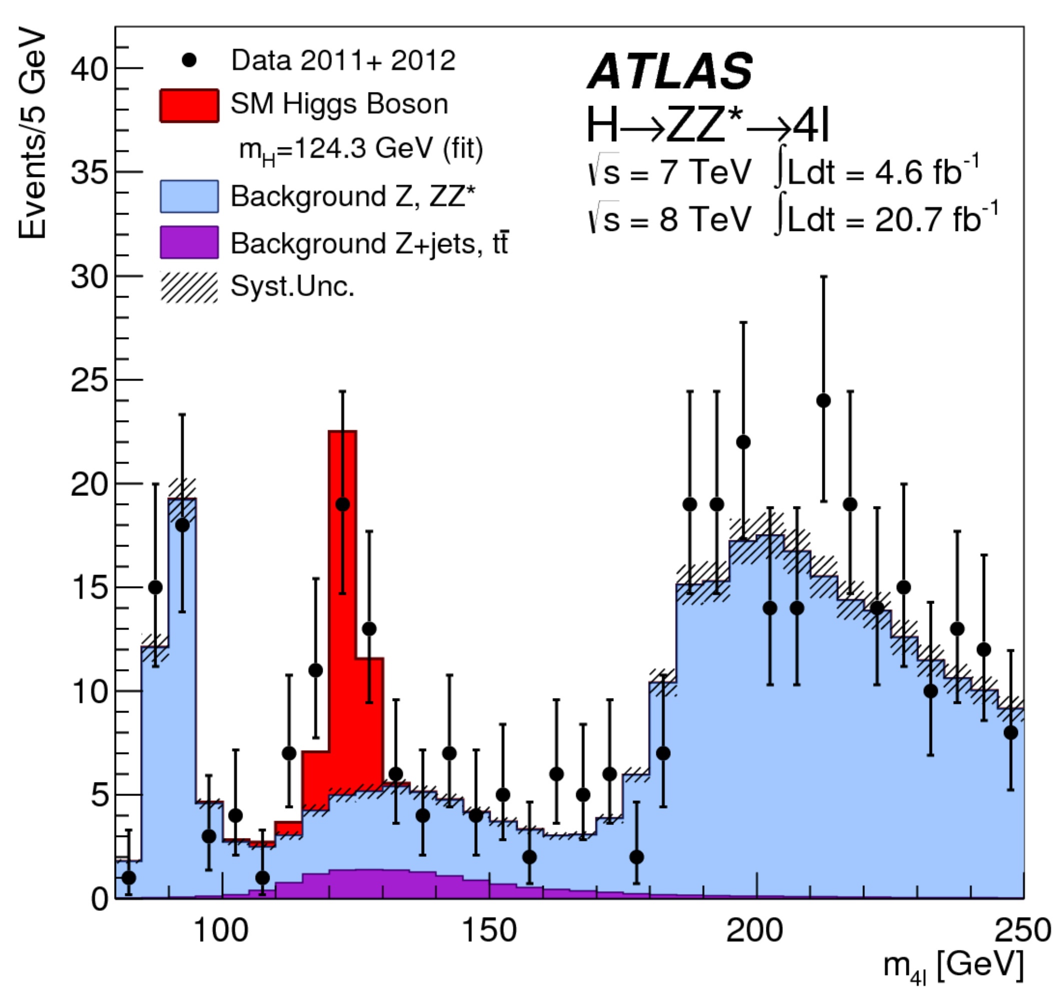
Problem: *We can't fill a histogram in 100M dimensions*

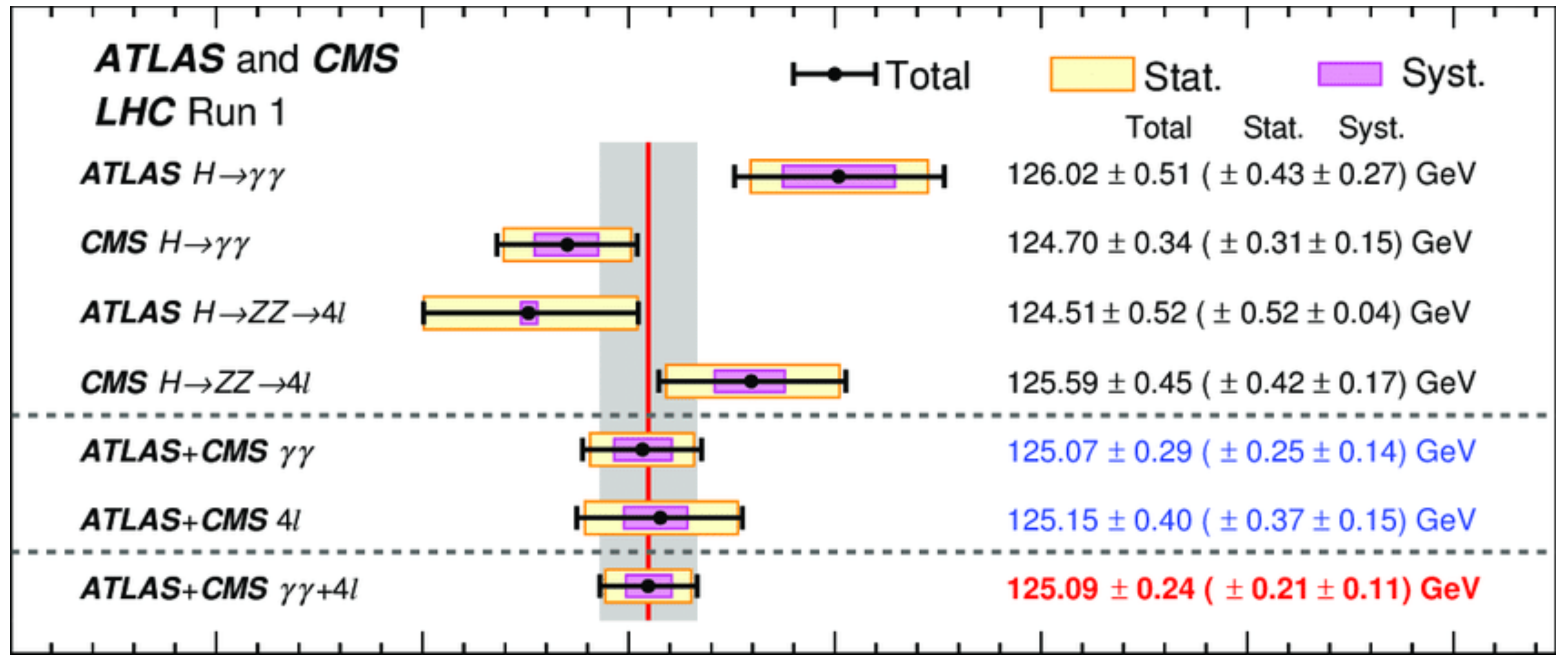
Summary Statistics

Most of what we call Reconstruction & Analysis is about good low-dimensional observables, for which we **can** fill histograms



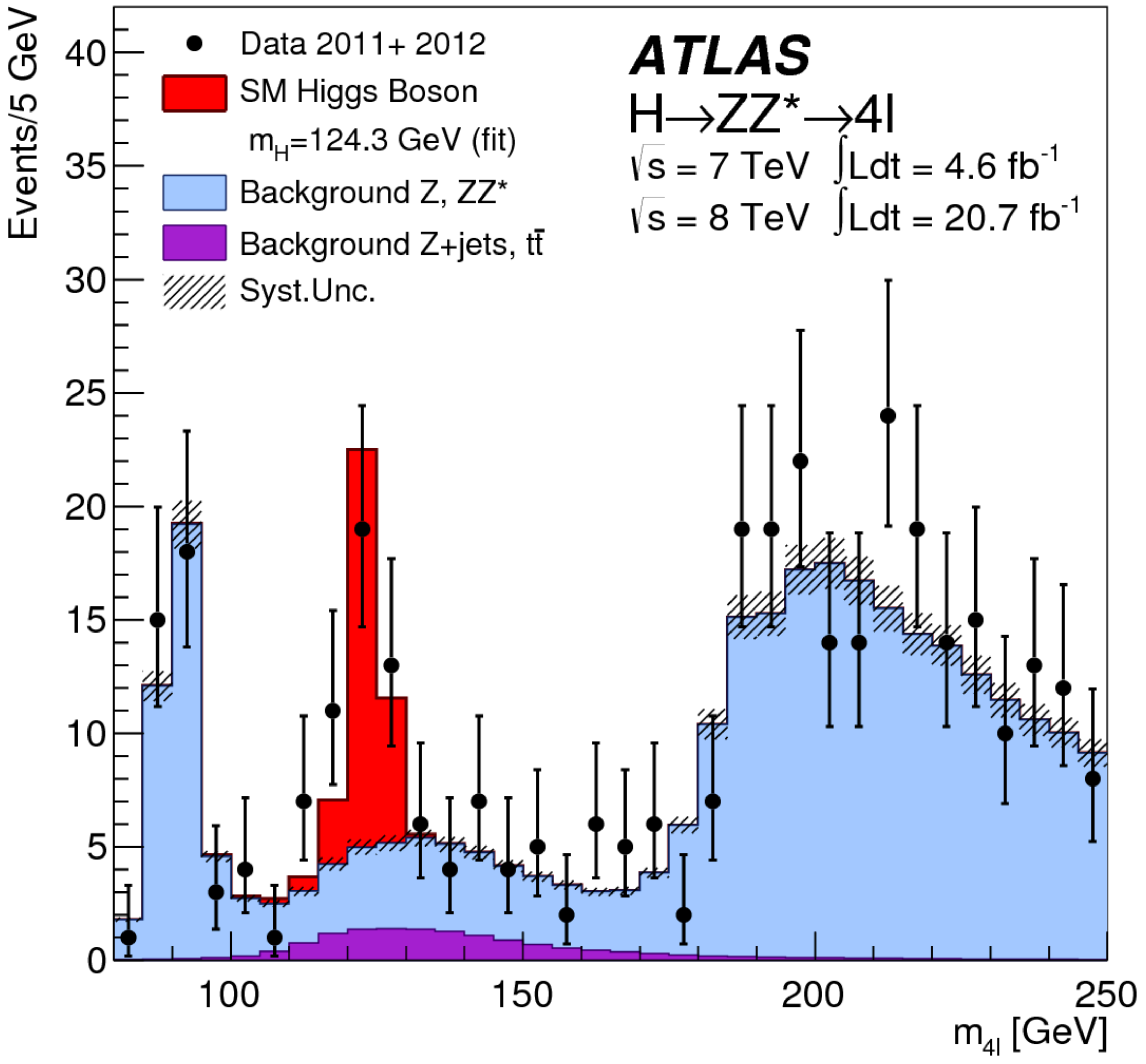
These observable densities are basis the subsequent analysis





Summary Statistics

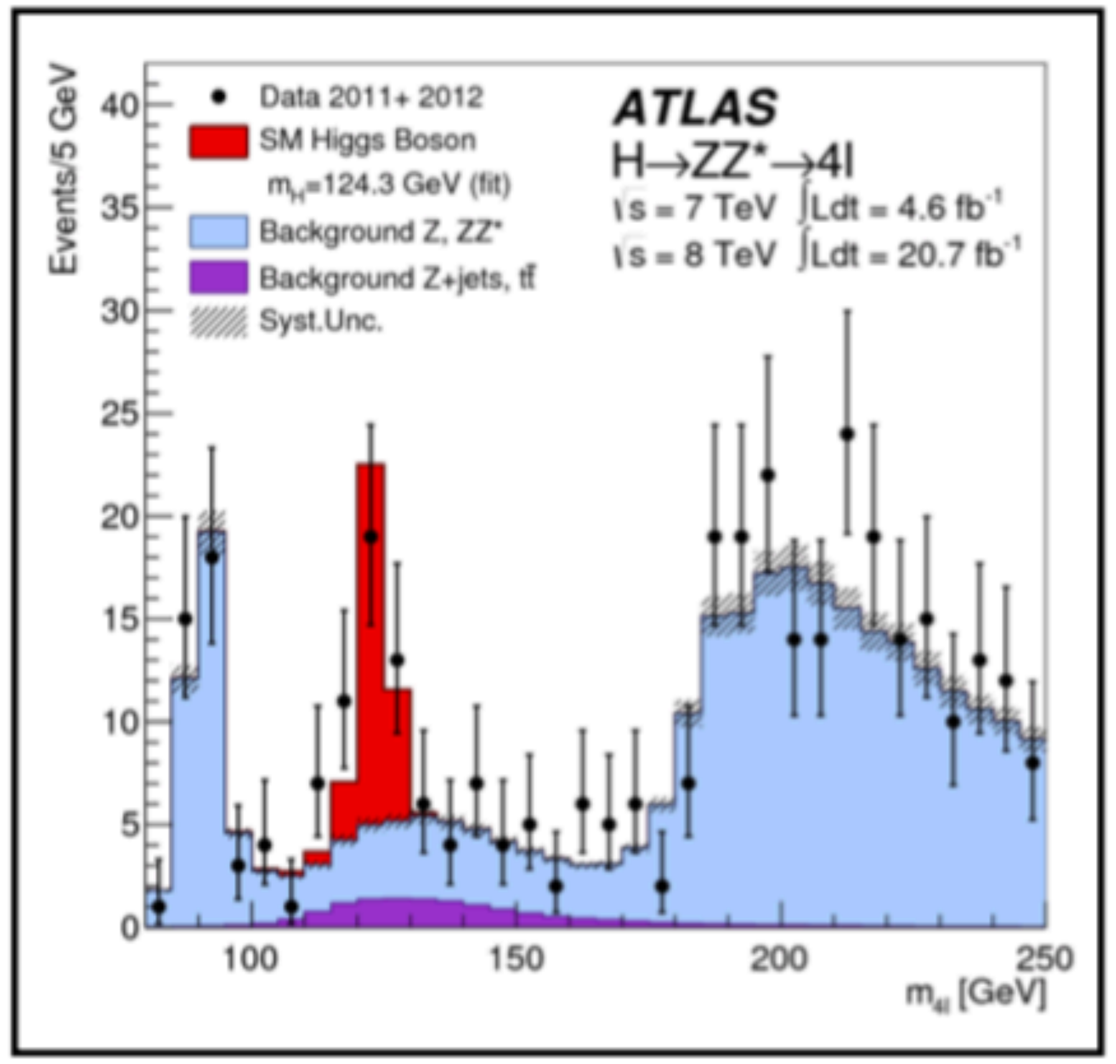
How to build a statistical model using histograms
is what **HistFactory** is all about



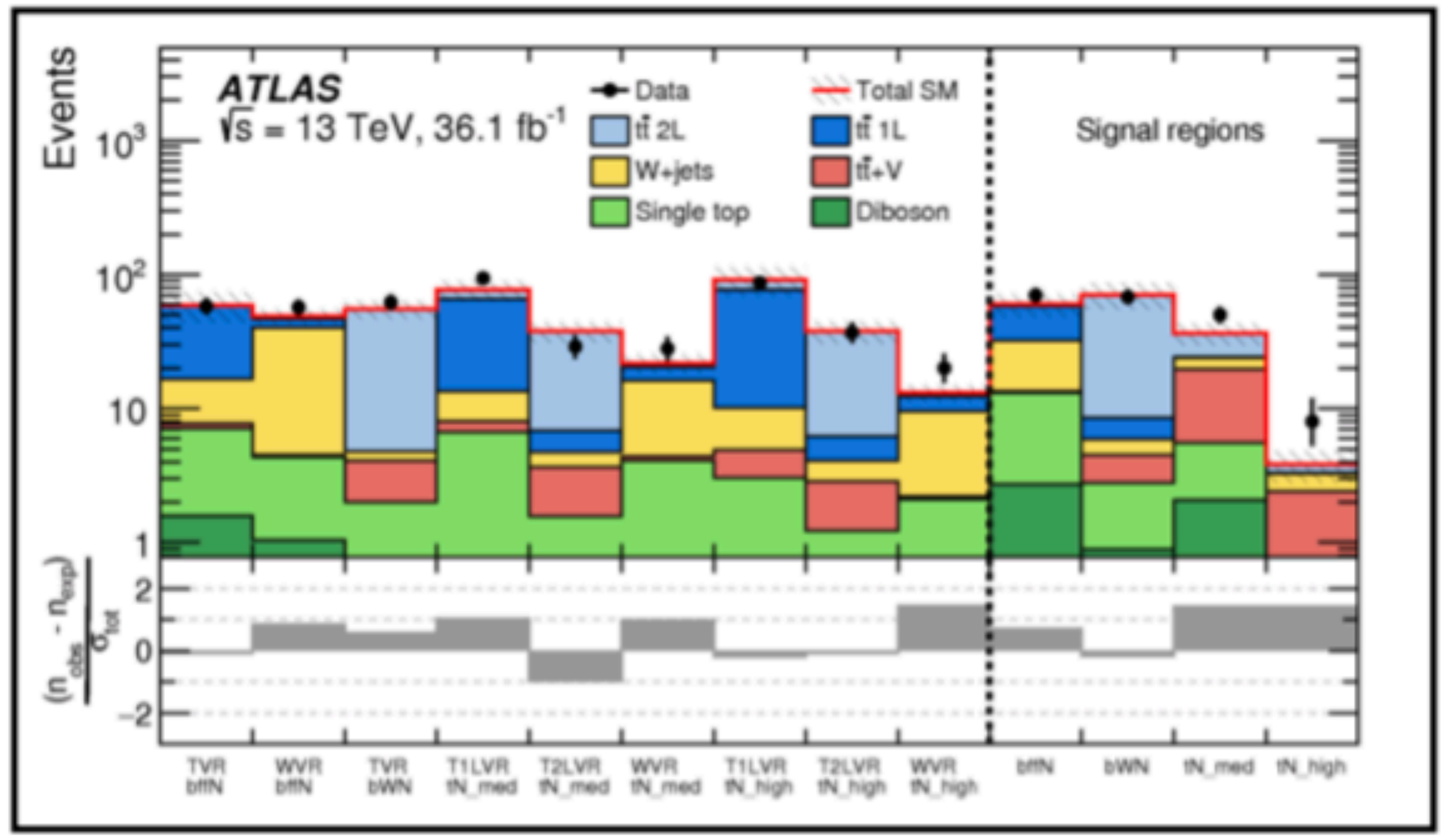
A powerful and flexible statistical modelling approach

A bit of Nomenclature

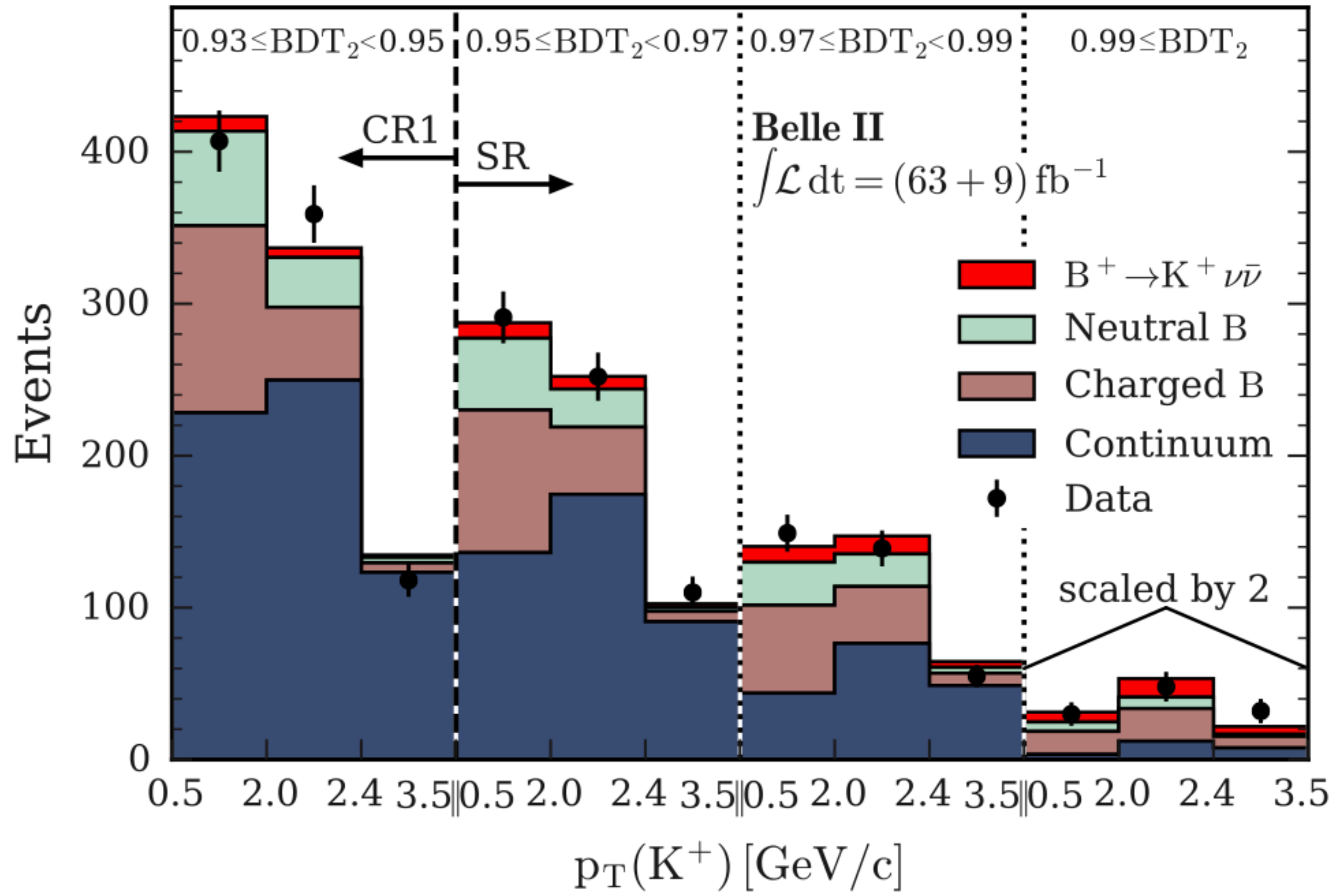
HistFactory denotes a mathematical modelling approach for describing typical HEP situations



Standard Model



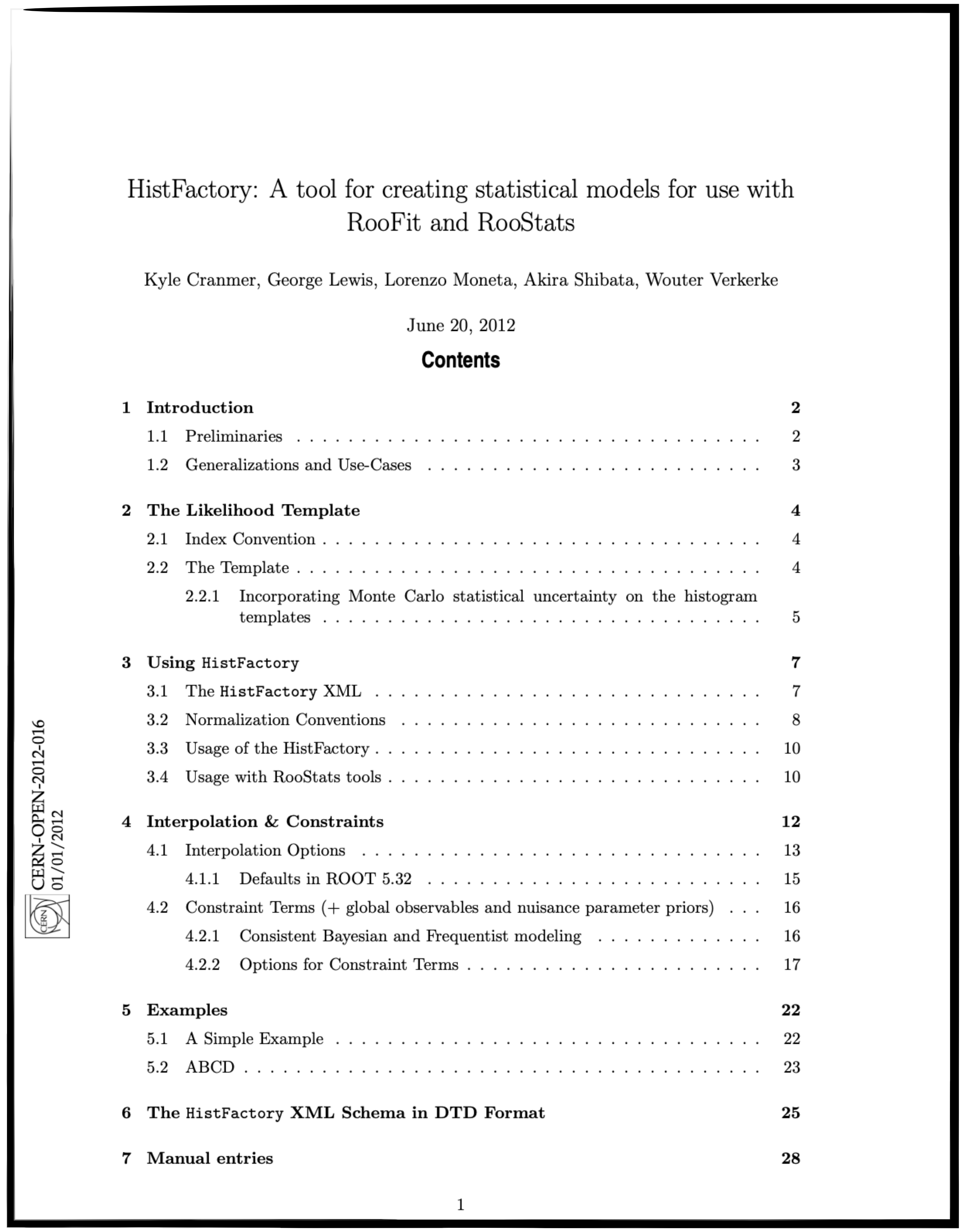
SUSY



Belle-II Analyses

A bit of Nomenclature

HistFactory was first implemented in ROOT / RooFit (2010)



HistFactory: A tool for creating statistical models for use with RooFit and RooStats

Kyle Cranmer, George Lewis, Lorenzo Moneta, Akira Shibata, Wouter Verkerke

June 20, 2012

CERN-OPEN-2012-016
01/01/2012

Contents

for a while the only implementation

A bit of Nomenclature

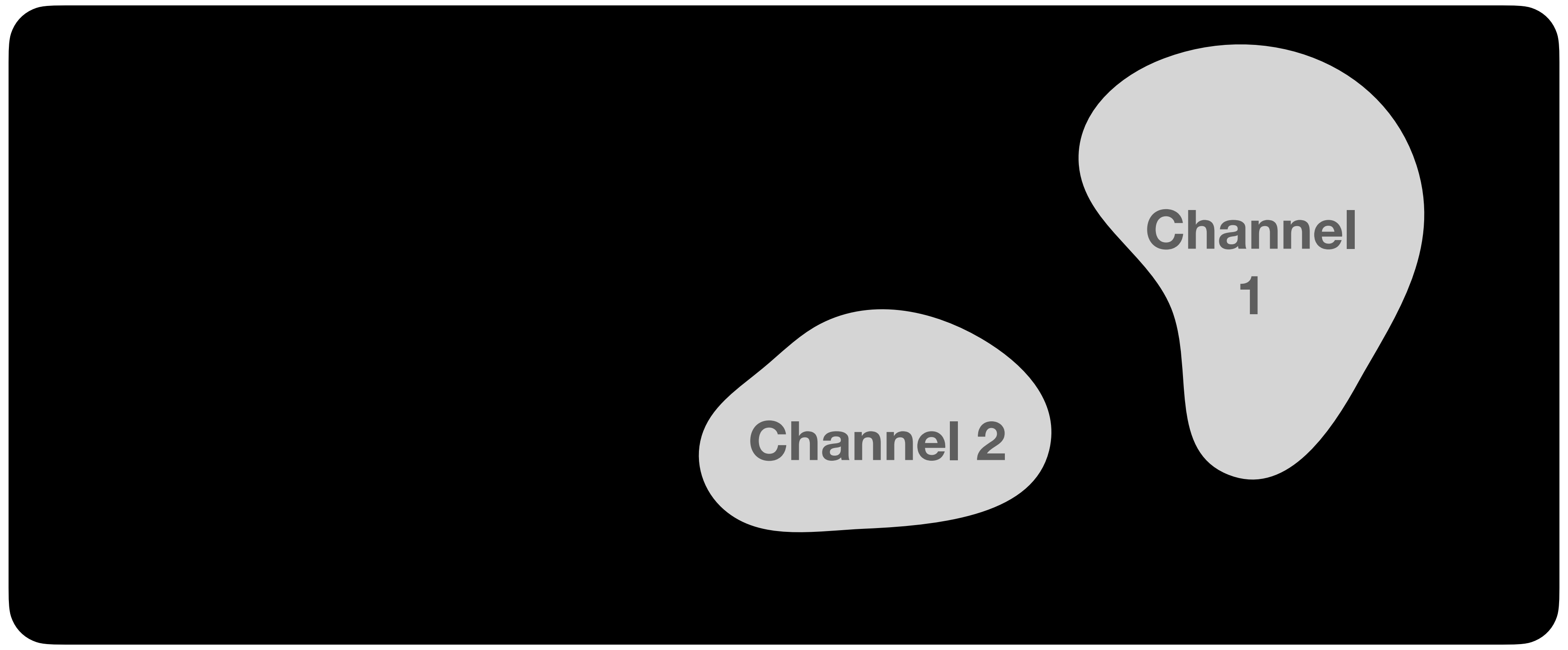
pyhf is a newer implementation within the scientific python ecosystem of the **HistFactory model** (2018)

Based on the Scientific Python and ML ecosystems

So let's Model-Build

The HistFactory model starts from considering space of events.

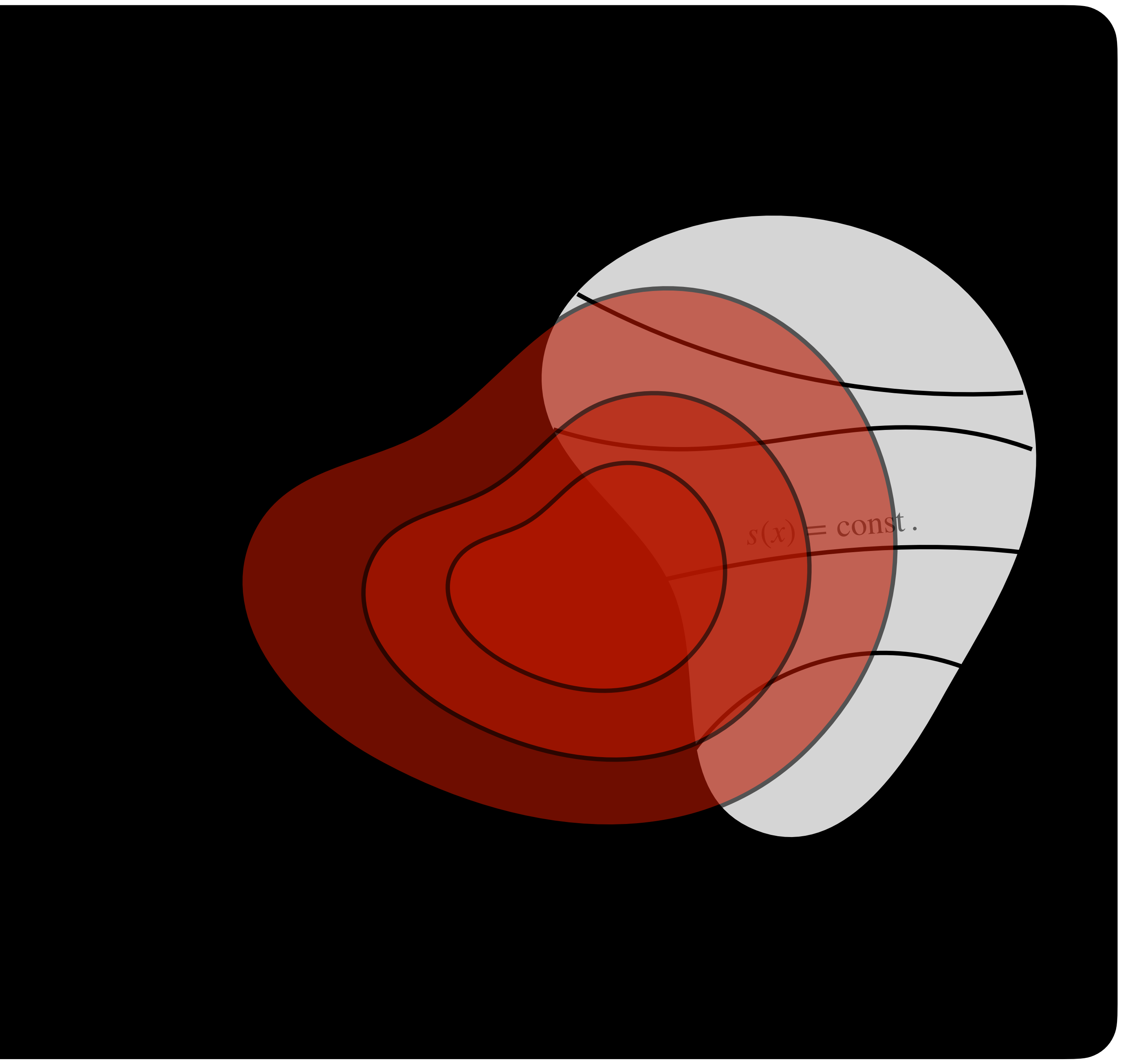
HEP typically cannot look at all events but a subset. Often a handful of orthogonal "regions" or "channels"



$$p(\text{data} \mid \theta) = \prod_c p_c(\text{data} \mid \theta)$$

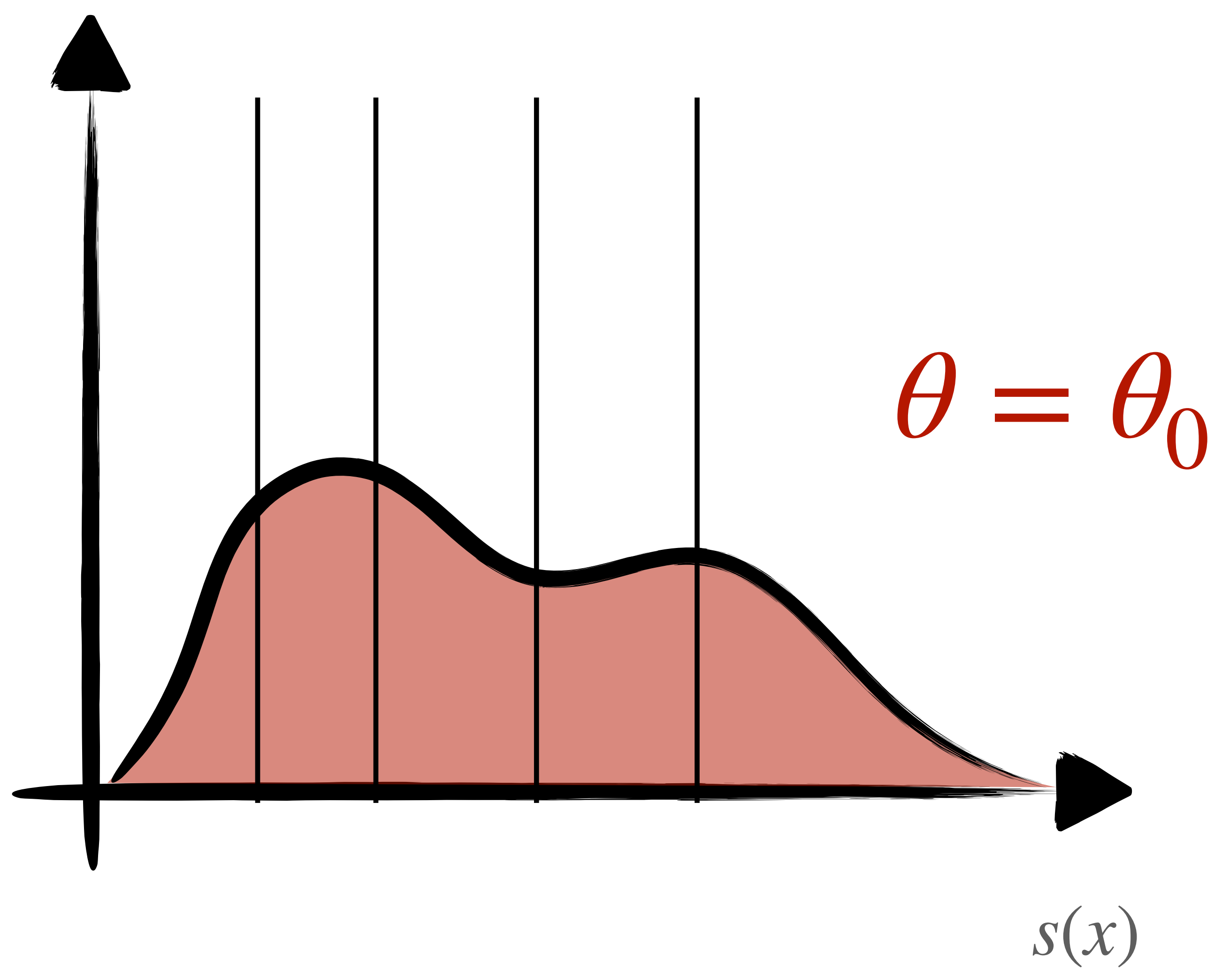
So let's Model-Build

For a single channel, we describe project events onto a single summary statistic $x \rightarrow s(x)$ for which we estimate $\hat{p}(s \mid \theta)$



$p(x \mid \theta)$

projection
$x \rightarrow s(x)$

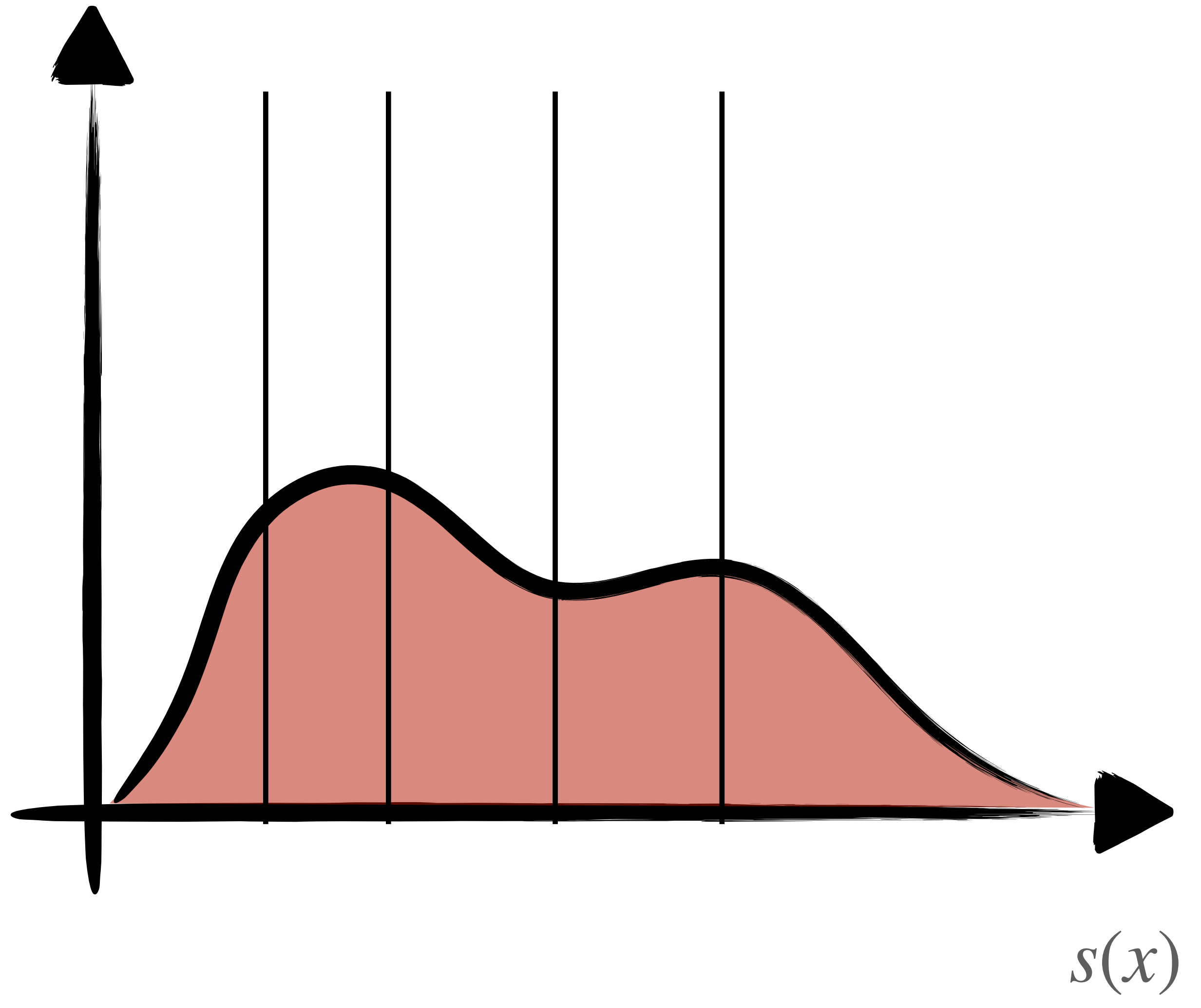


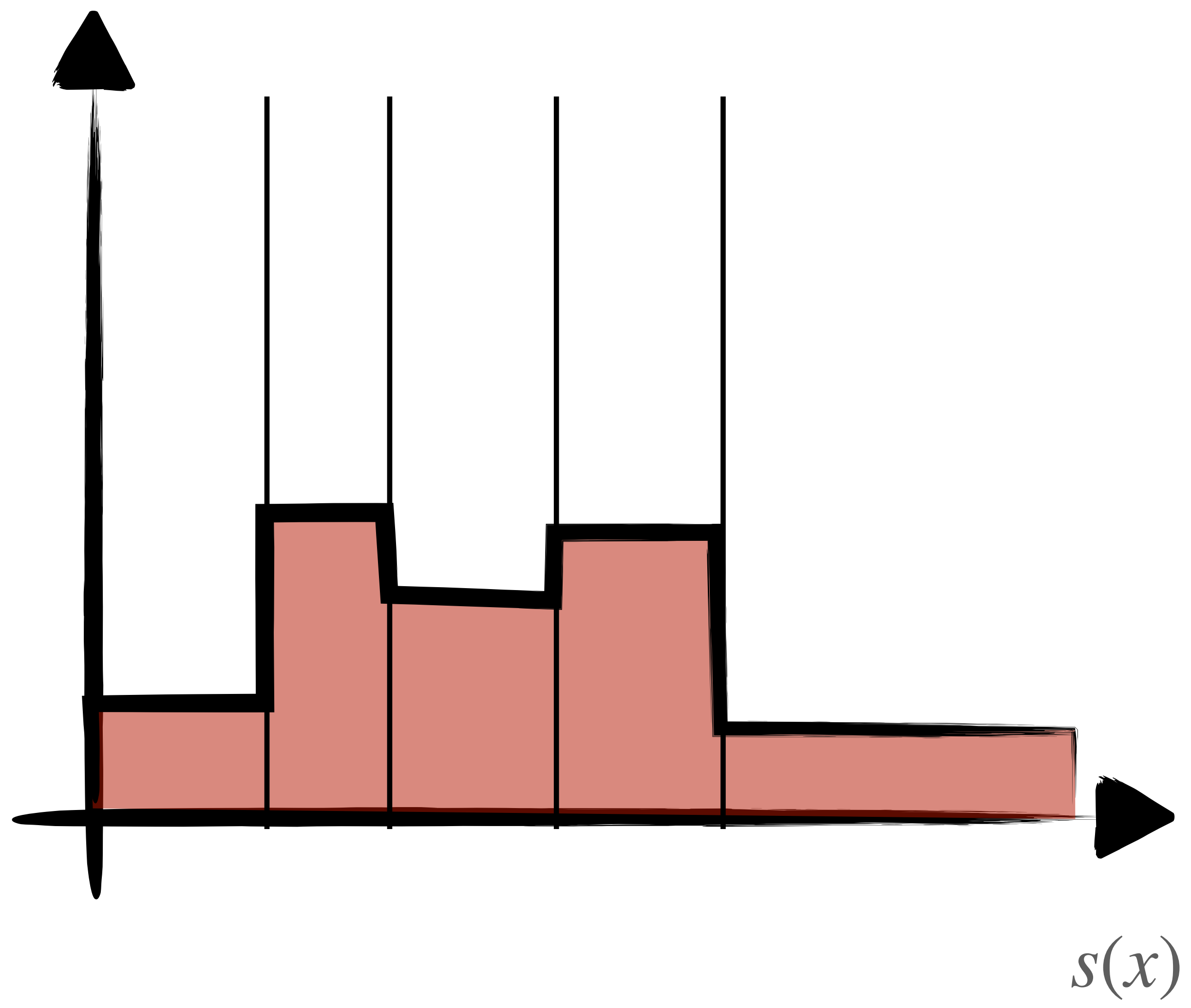
$p(s(x) \mid \theta)$

Changes in expected data distribution
as function of parameters translate accordingly

So let's Model-Build

Describing a 1-D observable in a HEP setting can be done by describing each bin in the density as a Poisson measurement

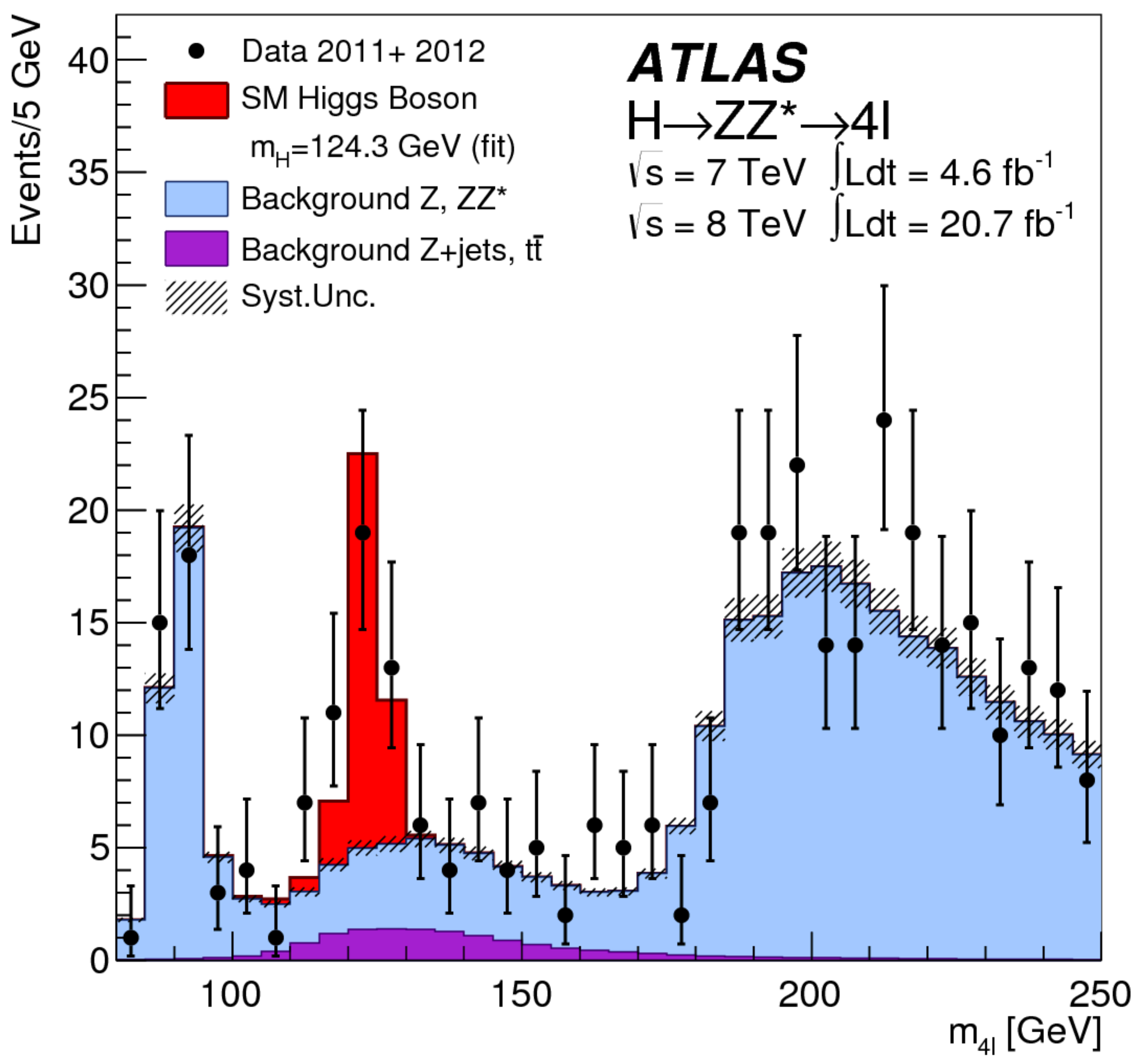




$$\prod_b \mathrm{Pois}(n_b \,|\, \lambda_b(\theta))$$

So let's Model-Build

In a given region of phase-space, typically a number of physics processes ("samples") contribute: we model the total as a sum.



$$\prod_b \mathrm{Pois}(n_b \,|\, \lambda_b = \sum_s \lambda_{sb}(\theta))$$

H signal

ZZ Sample

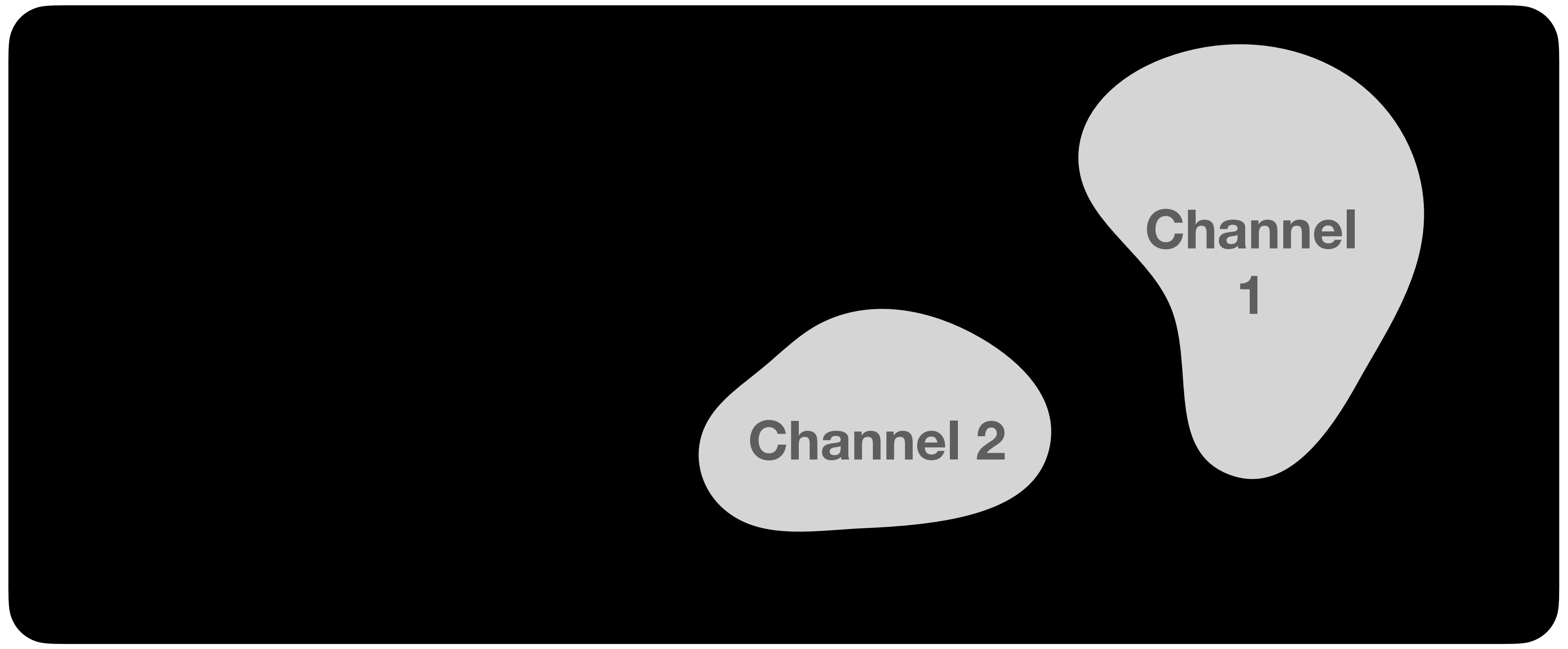
Z+jets

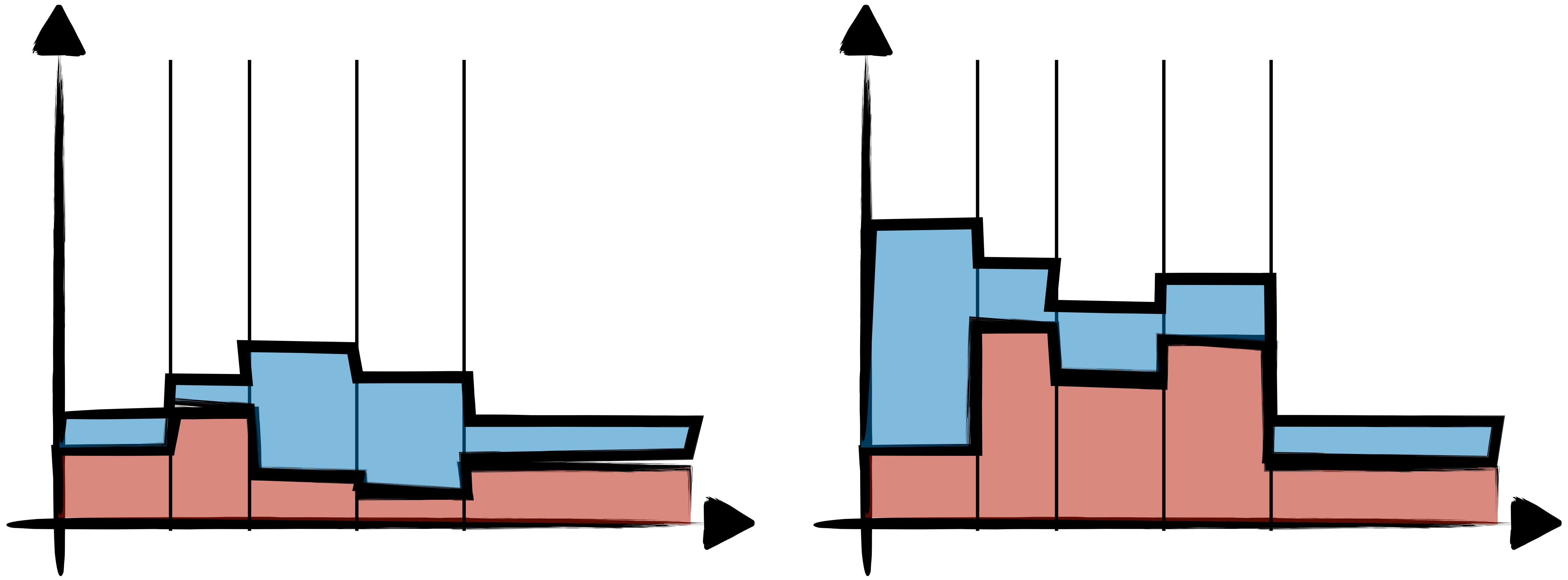
n observed

total expected

So let’s Model-Build

In a given region of phase-space, typically a number of physics processes (“samples”) contribute: we model the total as a sum.





$$\prod_{c\in\text{channels}} \prod_{b\in\text{bins}_\text{c}} \text{Pois}(n_{cb} \,|\, \lambda_b = \sum_s \lambda_{csb}(\theta))$$

Caputing the Information

In order to build a model like this, we need to keep track of a bunch of numbers.

How? A JSON File

Why? A ubiquitous format that is easy to share / validate.

(see later today topic of “likelihood preservation)

Caputing the Information

What we discussed so far would be captured with something very simple like this: **2 Channels - 2 Samples**

```
spec = {
    'channels': [
        {
            'name': 'channel1',
            'samples': [
                {'name': 'sample1', 'data': [50,60,70]},
                {'name': 'sample2', 'data': [10,5,2]}
            ]
        },
        {
            'name': 'channel2',
            'samples': [
                {'name': 'sample1', 'data': [150,160,170]},
                {'name': 'sample2', 'data': [20,10,4]}
            ]
        }
    ]
}
```

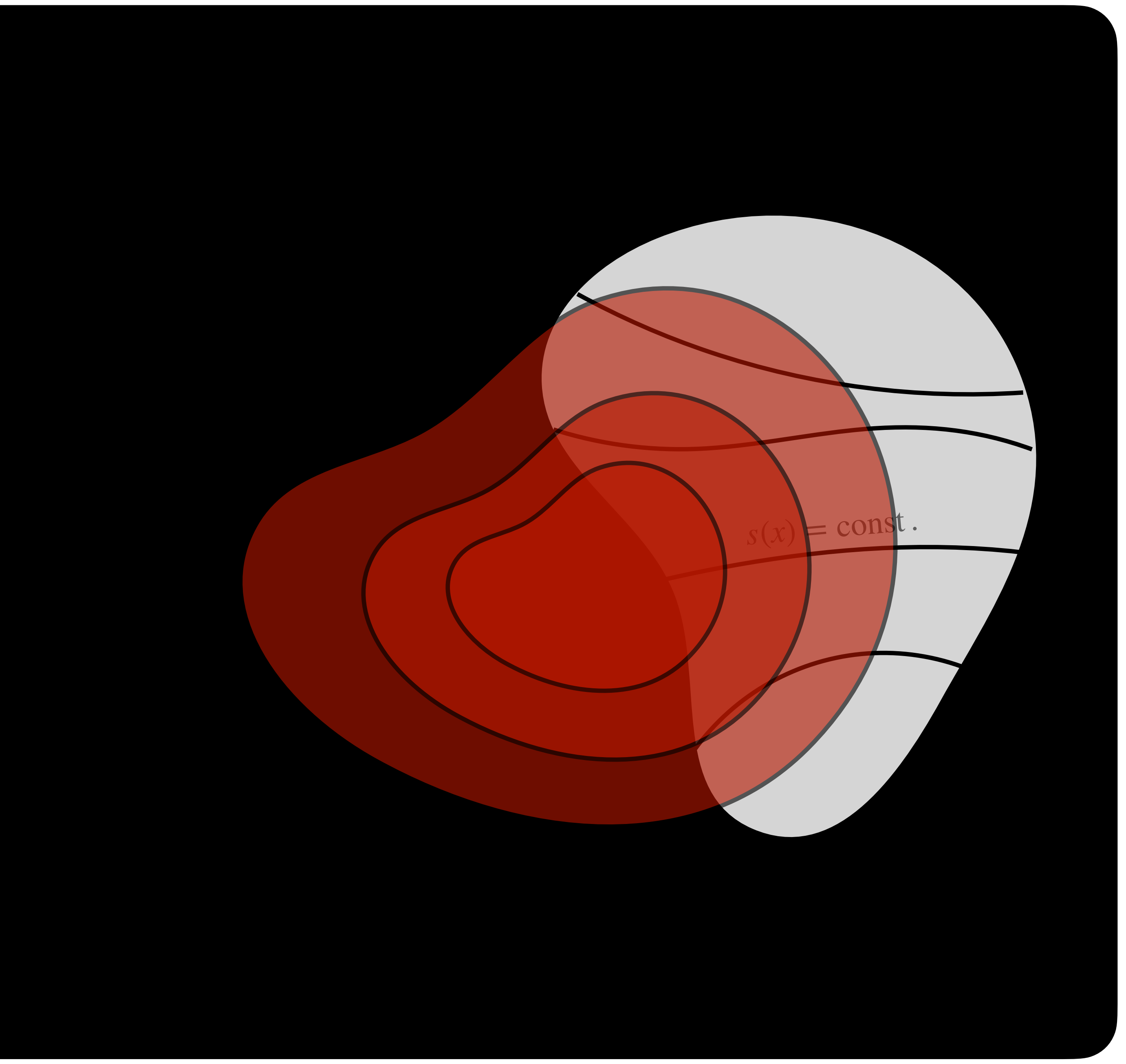
So let's Model-Build

Remaining questions:

- How do we model effect of varying θ
- How do we incorporate prior information on θ

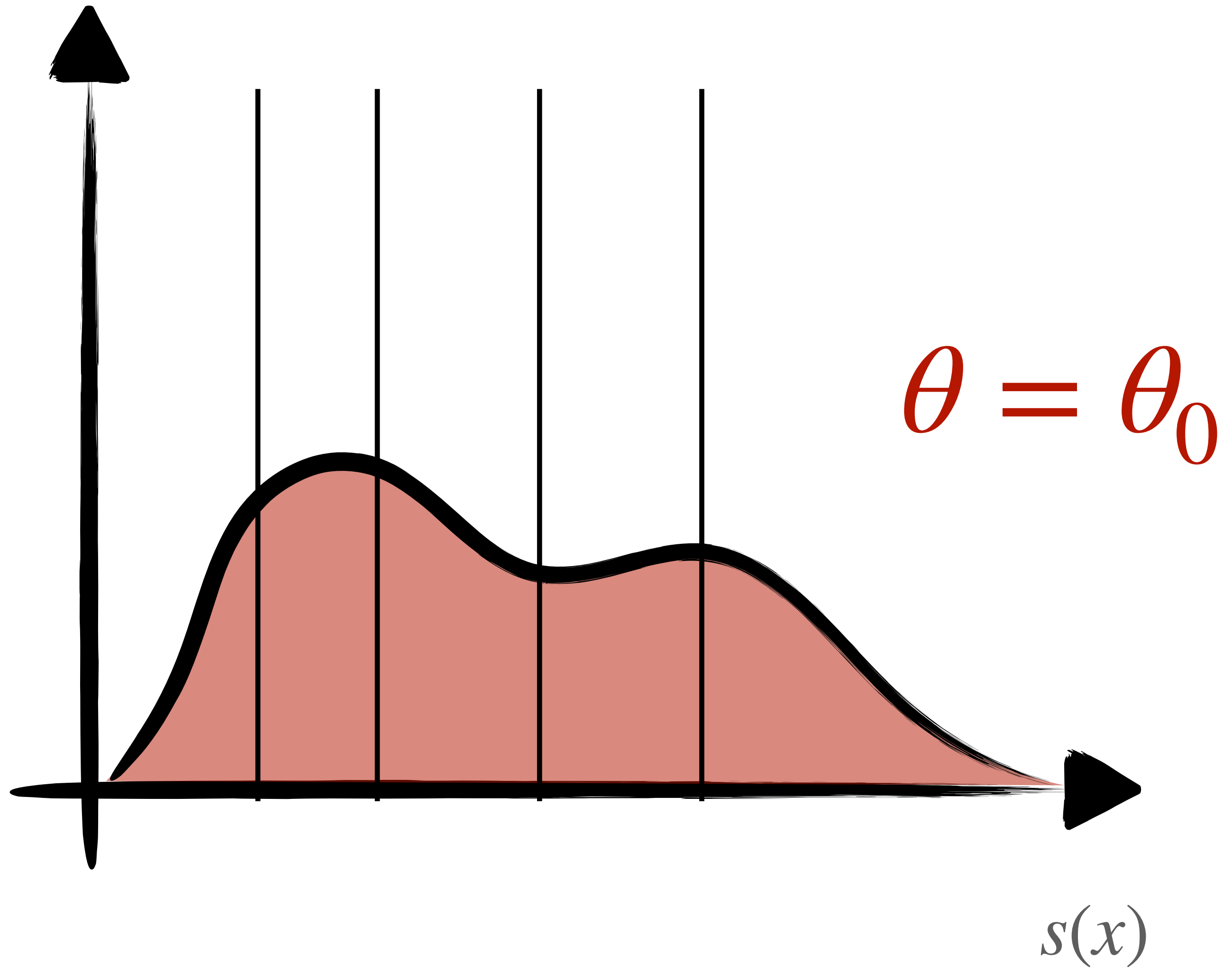
So let's Model-Build

The distribution of $s(x)$ depends on parameters θ - includes core physics parameters and nuisance parameters



$p(x \mid \theta)$

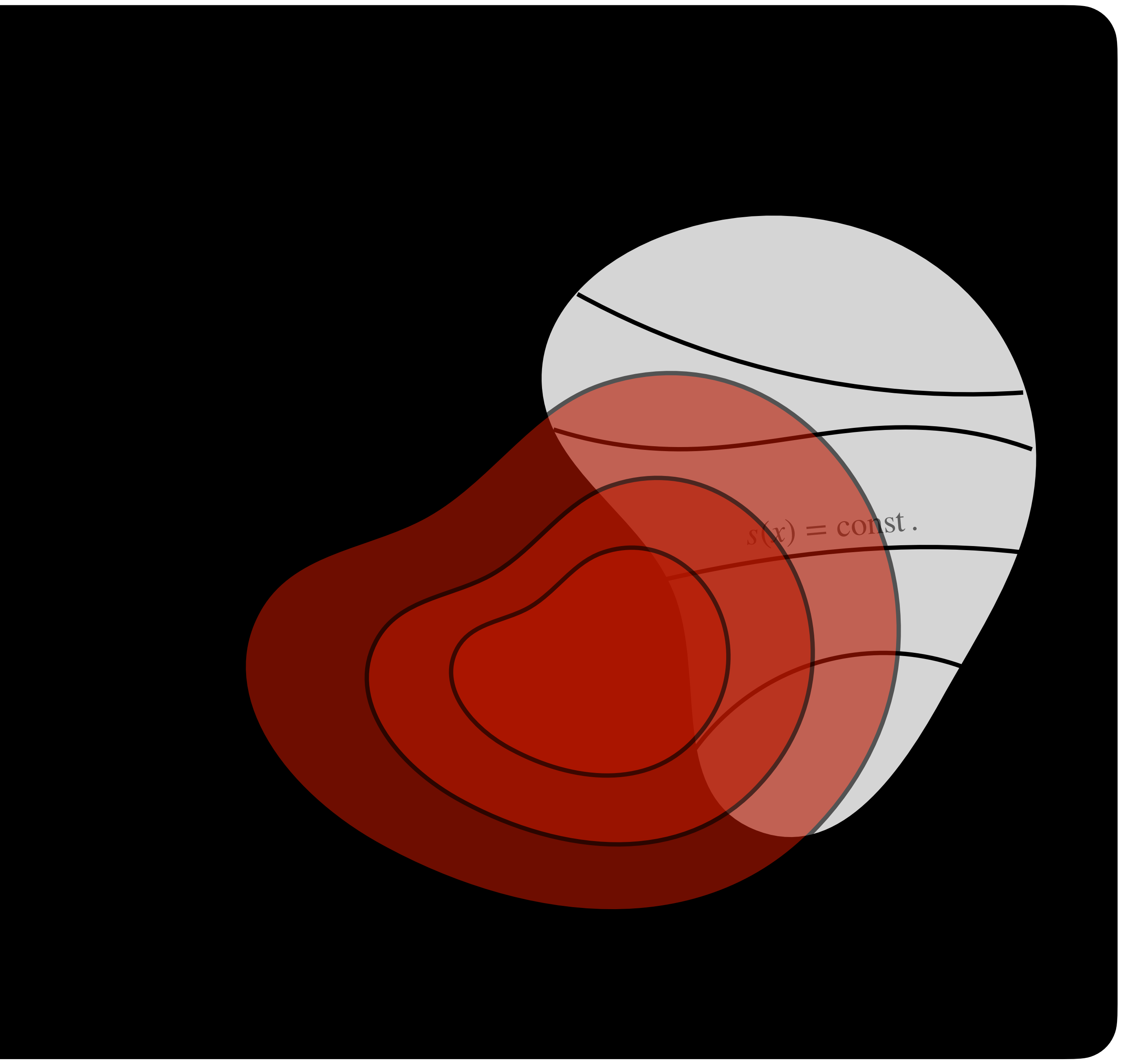
projection
$x \to s(x)$



$p(s(x) \mid \theta)$

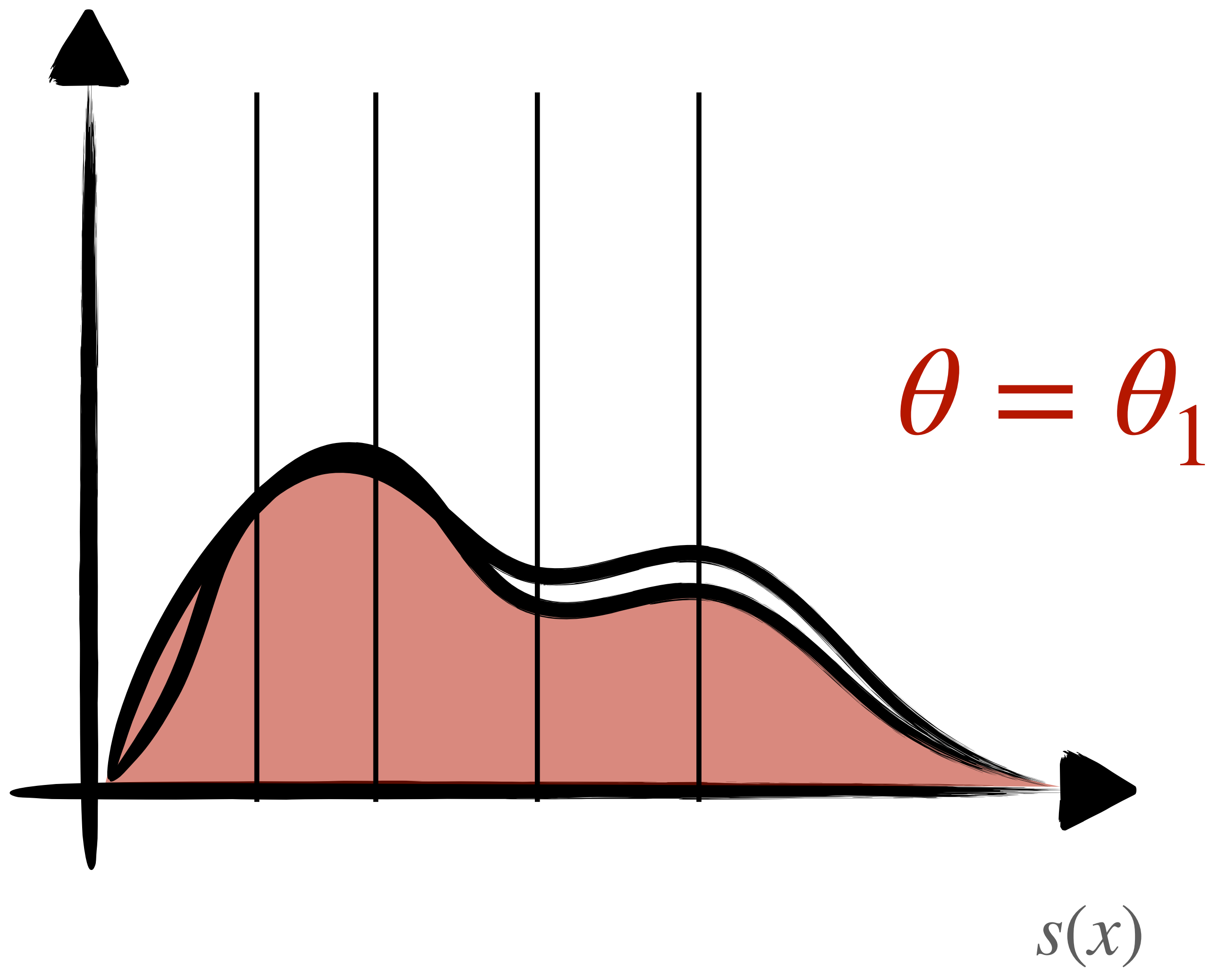
So let’s Model-Build

The distribution of $s(x)$ depends on parameters θ - includes core physics parameters and nuisance parameters



$p(x \mid \theta)$

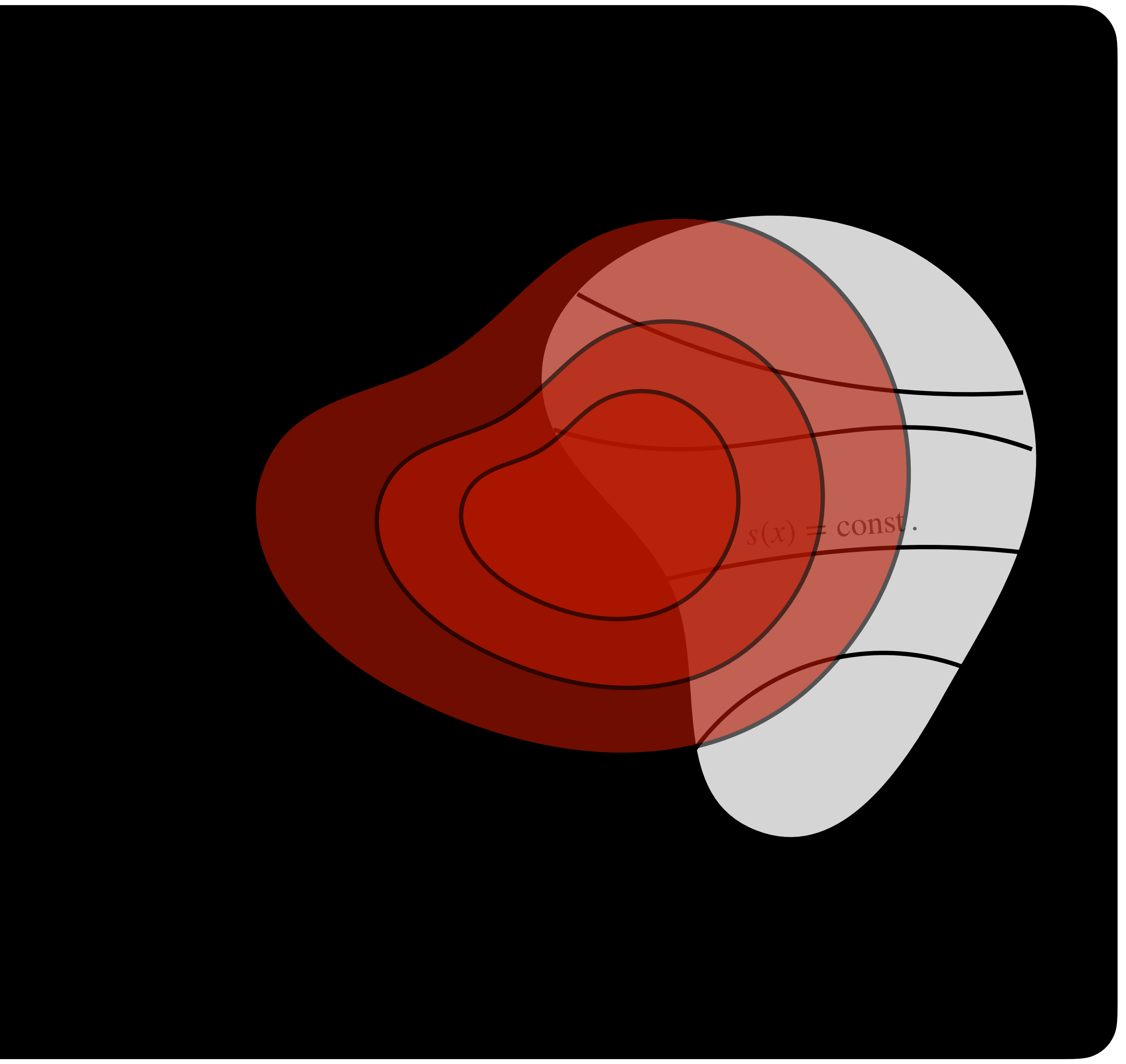
projection
$x \to s(x)$



$p(s(x) \mid \theta)$

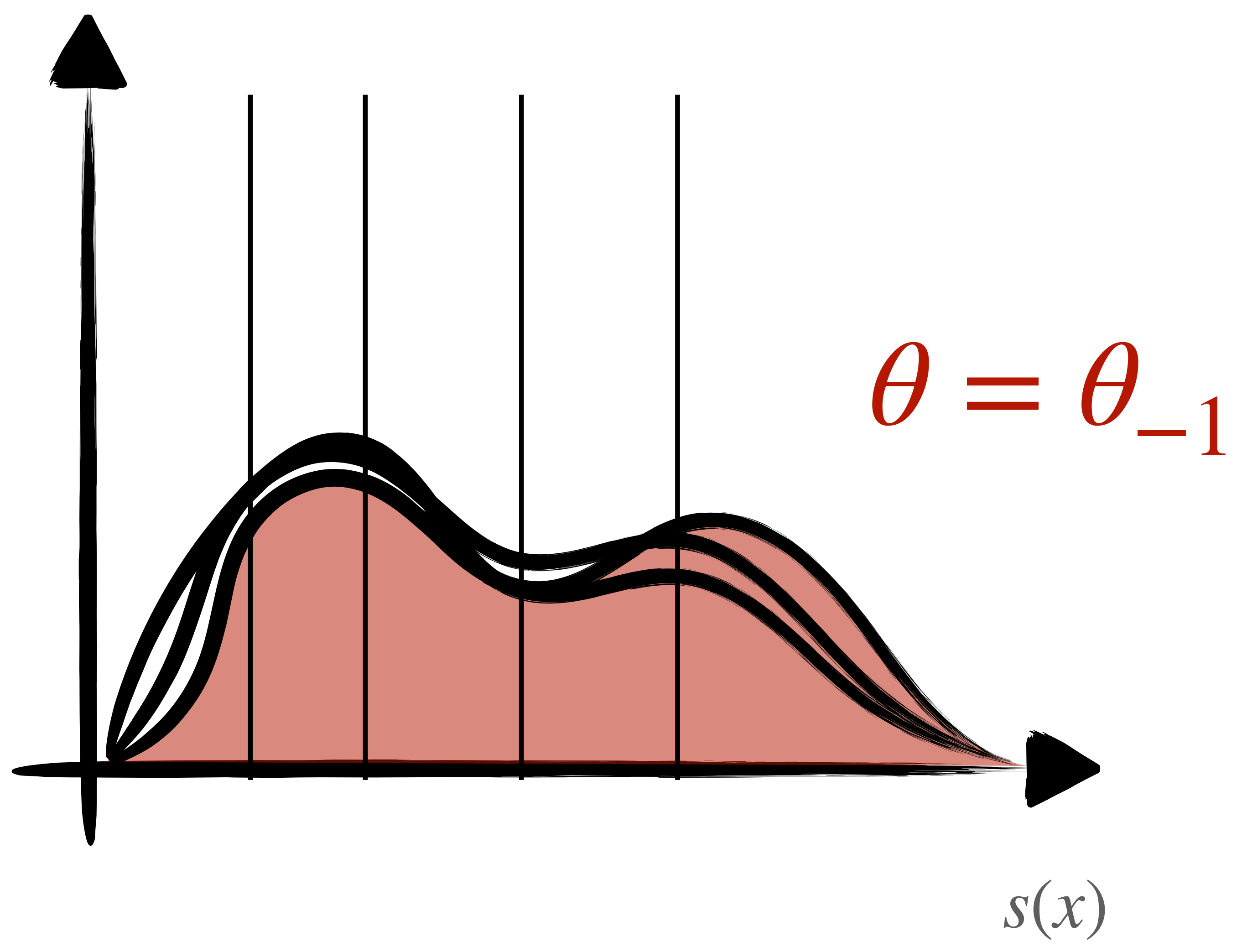
So let's Model-Build

The distribution of $s(x)$ depends on parameters θ - includes core physics parameters and nuisance parameters



$p(x\,|\,\theta)$

projection
$x \to s(x)$

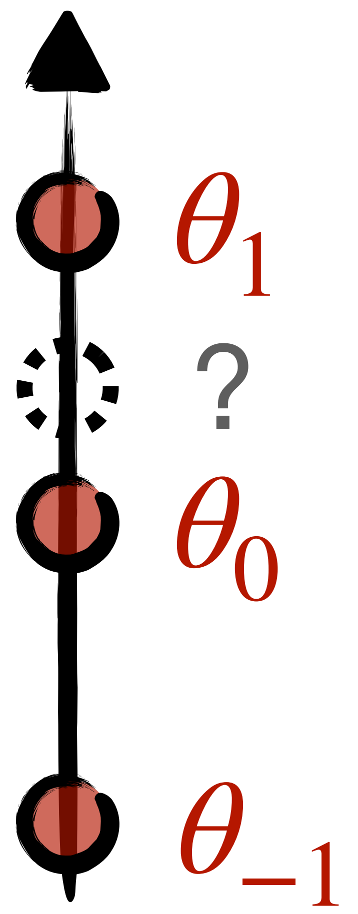


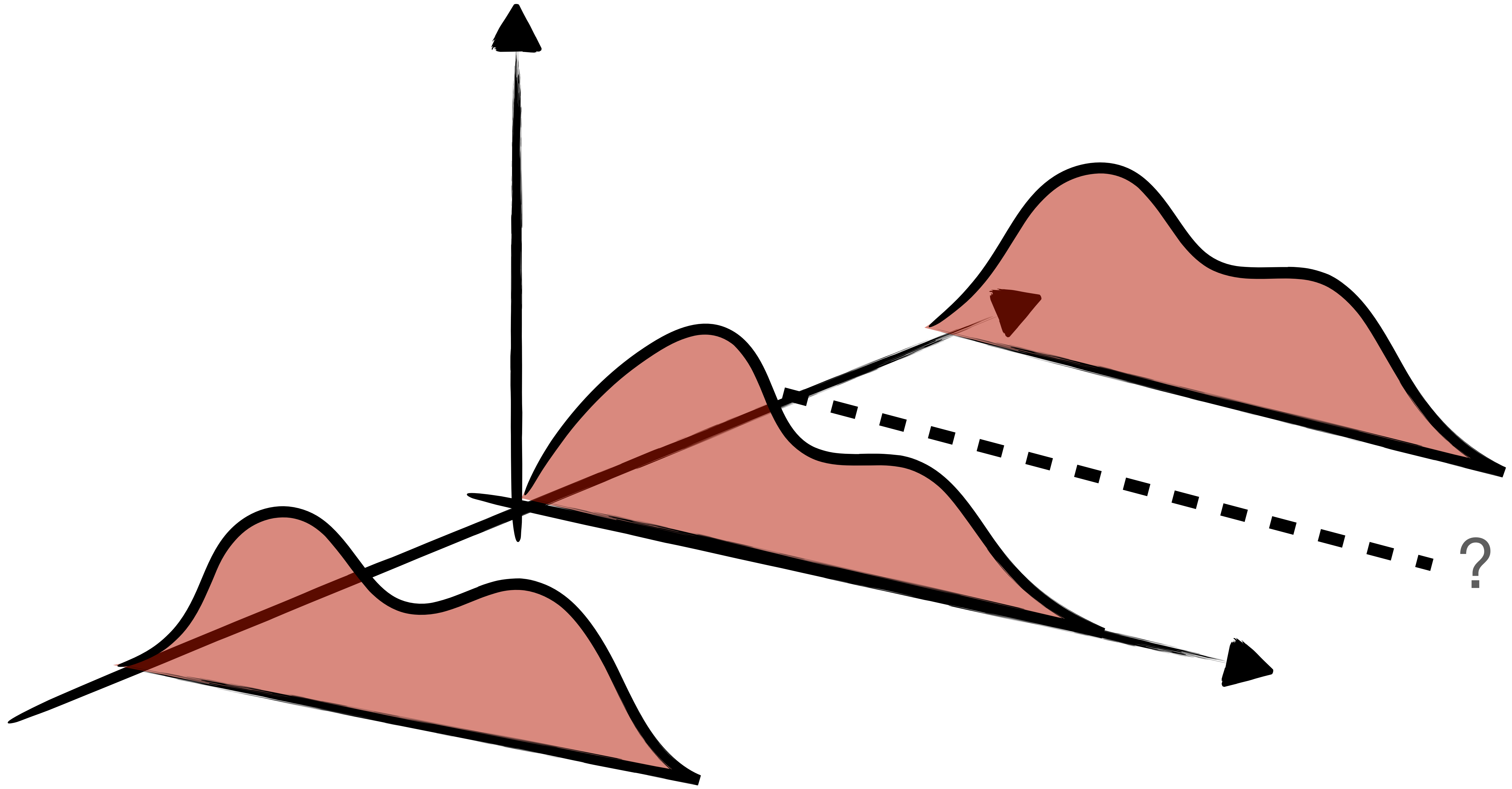
$p(s(x)\,|\,\theta)$

So let's Model-Build

Estimating the low-D distribution with samples from high-D is very expensive. Need a fast way to approximate this.

Parametrization in Histogram space





So let's Model-Build

HistFactory provides a few standard building blocks in order to model the effect of parameters to the distribution

"Modifiers"

Correlated Shape

histosys

Uncorrelated Shape

shapesys

MC Stats Variation

staterror

Normalization

normfactor

Luminosity

lumi

Correlated Scaling

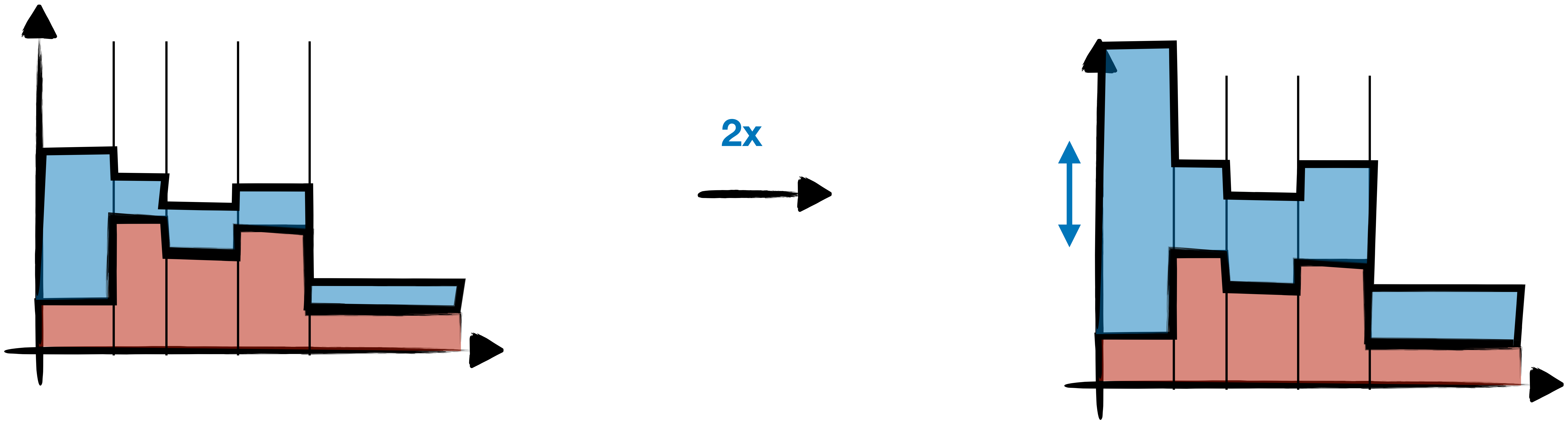
normfactor

Uncorrelated Scaling

shapefactor

Example Normfactor

Simplest modifier is a scaling $\lambda_{csb}(\theta) = \theta\lambda^{0}_{csb}$



Example Normfactor

To capture how each λ_{csb} is a function of various parameters, $\lambda_{csb} = \lambda_{csb}(\theta)$ we can add a number of such modifiers in JSON

```
spec = {
    'channels': [
        {
            'name': 'channel1',
            'samples': [
                {'name': 'sample1', 'data': [50,60,70], 'modifiers': []},
                {'name': 'sample2', 'data': [10,5,2], 'modifiers': [
                    {'type': 'normfactor', 'name': 'mu', 'data': None}
                ]}
            ]
        },
        {
            'name': 'channel2',
            'samples': [
                {'name': 'sample1', 'data': [150,160,170], 'modifiers': []},
                {'name': 'sample2', 'data': [20,10,4], 'modifiers': [
                    {'type': 'normfactor', 'name': 'mu', 'data': None}
                ]}
            ]
        }
    ]
}
```

modifier data.
here: scale only sample 2

```
import pyhf
model = pyhf.Model(spec)
```

```
model.expected_actualdata([1.0])
```

```
array([ 60.,  65.,  72., 170., 170., 174.])
```

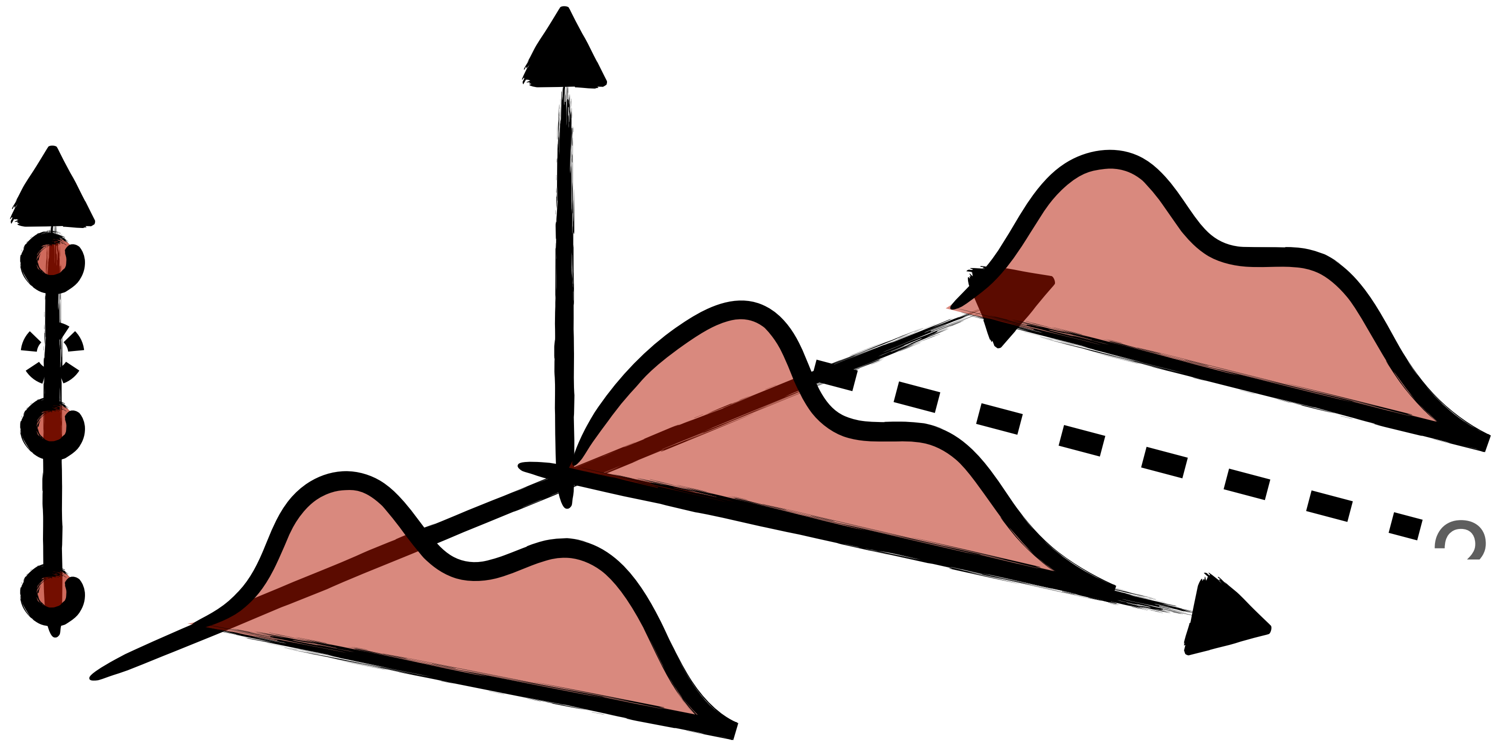
```
model.expected_actualdata([2.0])
```

```
array([ 70.,  70.,  74., 190., 180., 178.])
```

evaluate for different parameter values

Example HistoSys

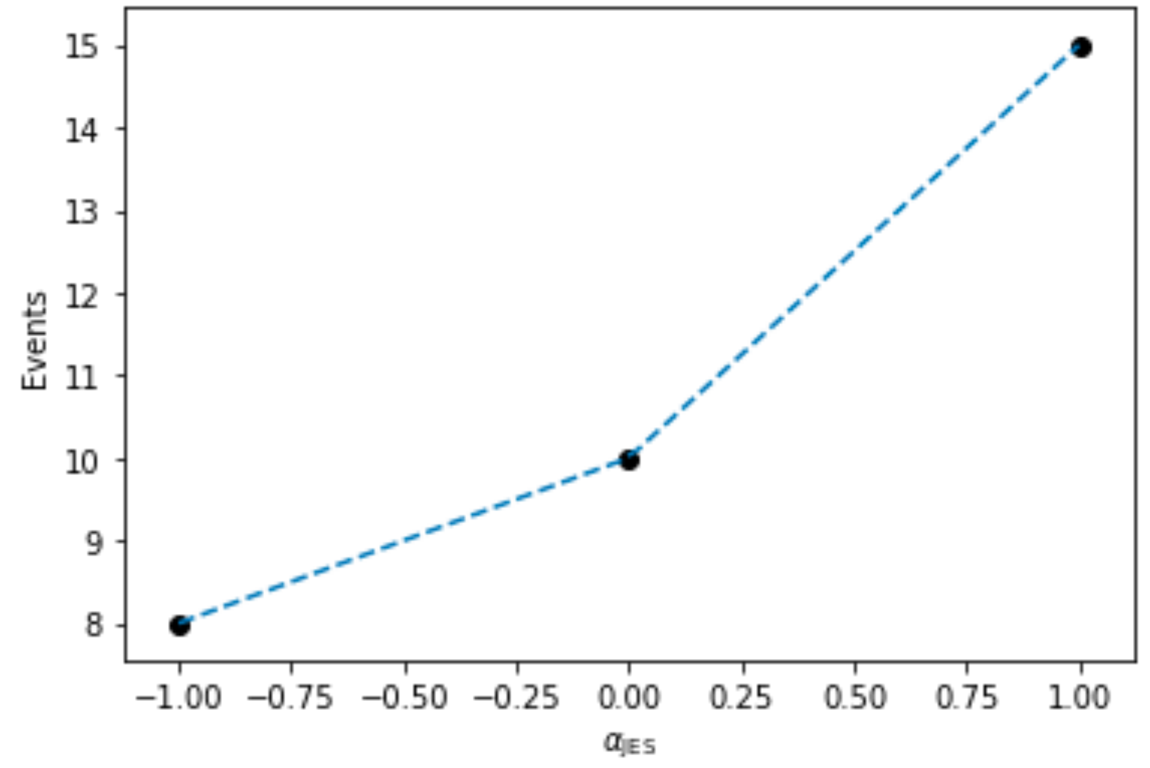
Histosys builds a parametrization of histograms, based on three input histograms (“nominal”, “up”, “down”)

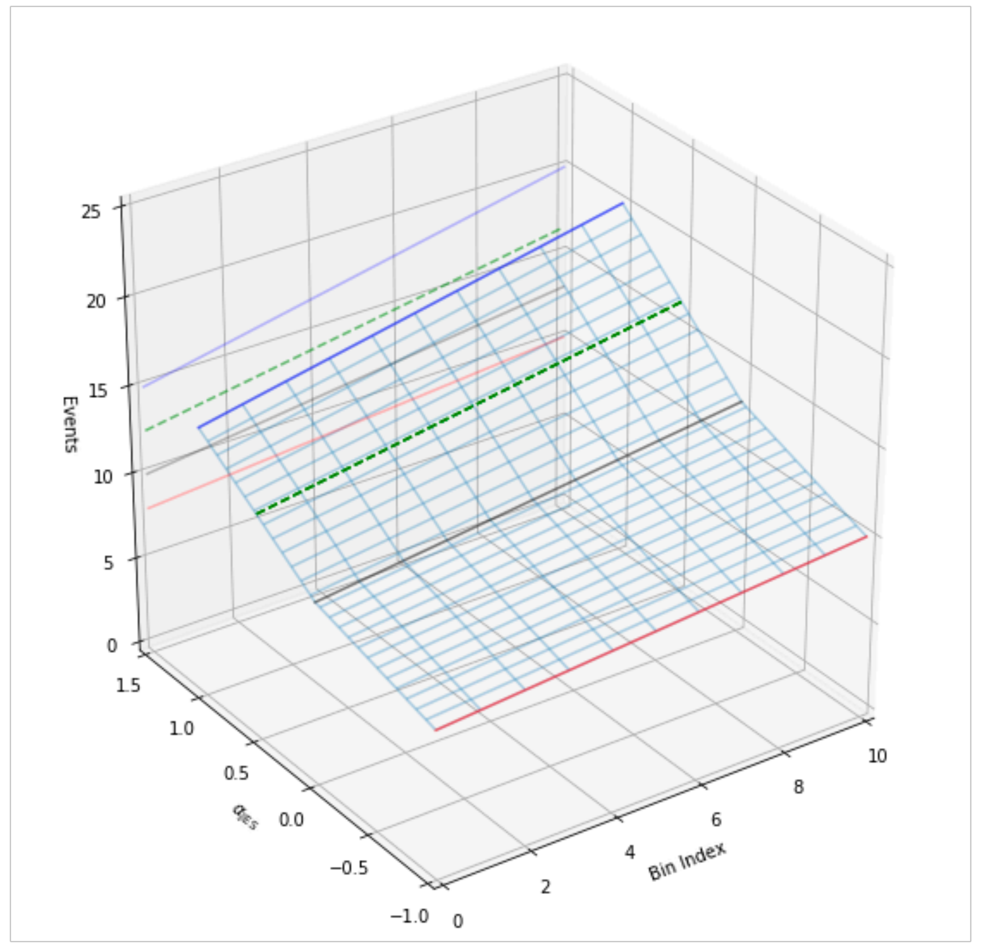


$$\lambda(\alpha = 1) = \lambda^{\mathrm{up}}$$

$$\lambda(\alpha = 0) = \lambda^{\mathrm{nom}}$$

$$\lambda(\alpha = 1) = \lambda^{\mathrm{dn}}$$





Example HistoSys

```python
spec = {
    'channels': [
        {
            'name': 'channel1',
            'samples': [
                {'name': 'sample1', 'data': [50,60,70],
                 'modifiers': [
                    {'type': 'histosys', 'name': 'mu', 'data': {
                        'hi_data': [55, 65,75],
                        'lo_data': [45, 50,30]
                    }}

                ]},
            ]
        }
    ]
}
```

```python
model = pyhf.Model(spec)
model.expected_actualdata([0.0])
```

```
array([50., 60., 70.])
```

```python
model.expected_actualdata([1.0])
```

```
array([55., 65., 75.])
```

```python
model.expected_actualdata([-1.0])
```

```
array([45., 50., 30.])
```

Example NormSys

Normsys parametrizes a histogram to model a normalization uncertainty ("my background is X ± Y")

$$\lambda(\alpha = 1) = k_{\mathrm{up}}\lambda^{\mathrm{nom}}$$

$$\lambda(\alpha = 0) = \lambda^{\mathrm{nom}}$$

$$\lambda(\alpha = 1) = k_{\mathrm{dn}}\lambda^{\mathrm{nom}}$$

```
spec = {
    'channels': [
        {
            'name': 'channel1',
            'samples': [
                {'name': 'sample1', 'data': [50,60,70],
                 'modifiers': [
                    {'type': 'normsys', 'name': 'mu', 'data': { 'hi': 1.1, 'lo': 0.8 }}

                ]},
            ]
        }
    ]
}
```

```
model = pyhf.Model(spec)
model.expected_actualdata([0.0])
```

```
array([50., 60., 70.])
```

```
model.expected_actualdata([1.0])
```

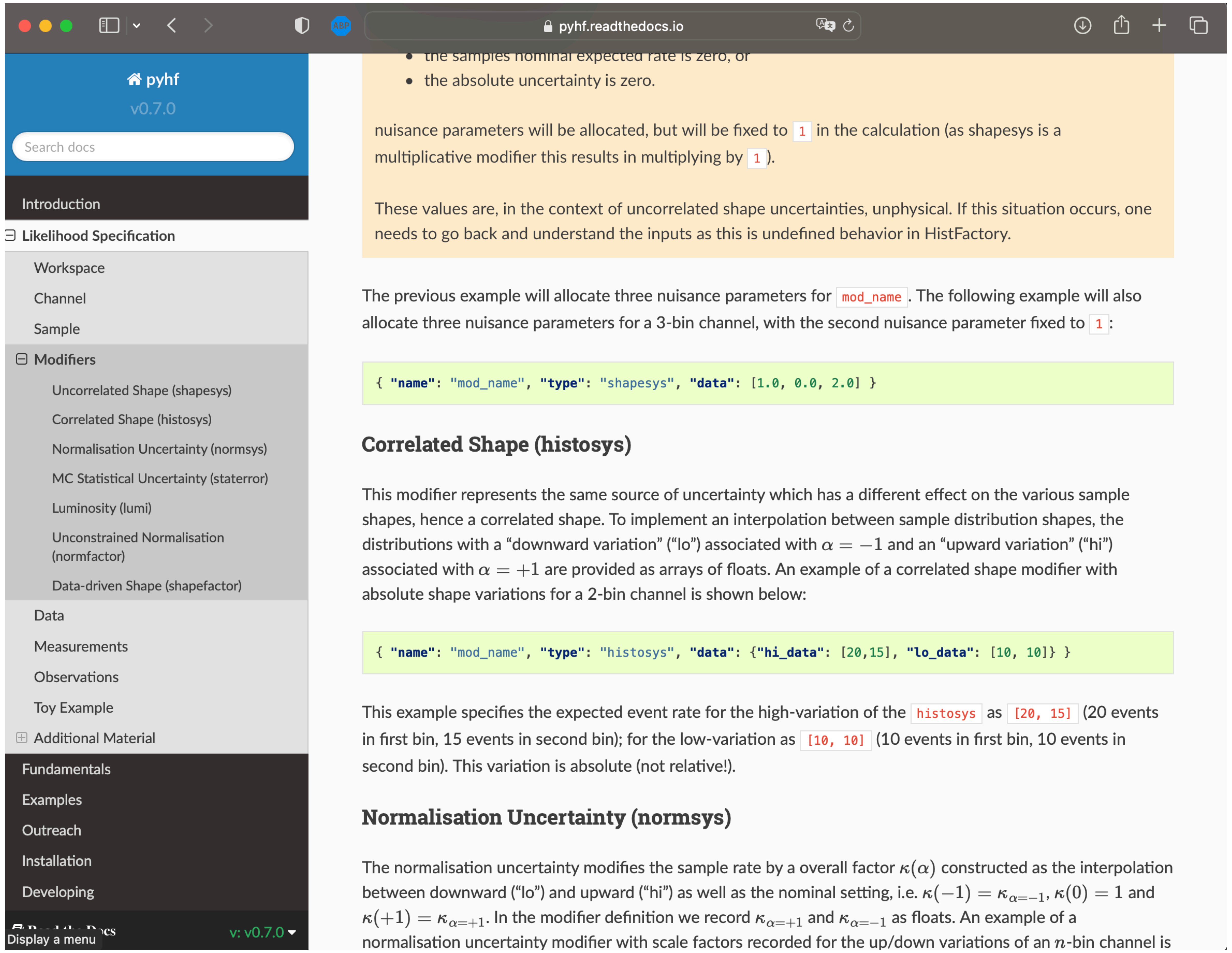
```
array([55., 66., 77.])
```

```
model.expected_actualdata([-1.0])
```

```
array([40., 48., 56.])
```

So let's Model-Build

You can find a detailed description of all available modifiers and what they do in the pyhf documentation



pyhf.readthedocs.io

pyhf

v0.7.0

Search docs

- the samples nominal expected rate is zero, or
- the absolute uncertainty is zero.

nuisance parameters will be allocated, but will be fixed to `1` in the calculation (as shapesys is a multiplicative modifier this results in multiplying by `1`).

These values are, in the context of uncorrelated shape uncertainties, unphysical. If this situation occurs, one needs to go back and understand the inputs as this is undefined behavior in HistFactory.

The previous example will allocate three nuisance parameters for `mod_name`. The following example will also allocate three nuisance parameters for a 3-bin channel, with the second nuisance parameter fixed to `1`:

```
{ "name": "mod_name", "type": "shapesys", "data": [1.0, 0.0, 2.0] }
```

Correlated Shape (histosys)

This modifier represents the same source of uncertainty which has a different effect on the various sample shapes, hence a correlated shape. To implement an interpolation between sample distribution shapes, the distributions with a "downward variation" ("lo") associated with $\alpha = -1$ and an "upward variation" ("hi") associated with $\alpha = +1$ are provided as arrays of floats. An example of a correlated shape modifier with absolute shape variations for a 2-bin channel is shown below:

```
{ "name": "mod_name", "type": "histosys", "data": {"hi_data": [20,15], "lo_data": [10, 10]} }
```

This example specifies the expected event rate for the high-variation of the `histosys` as `[20, 15]` (20 events in first bin, 15 events in second bin); for the low-variation as `[10, 10]` (10 events in first bin, 10 events in second bin). This variation is absolute (not relative!).

Normalisation Uncertainty (normsys)

The normalisation uncertainty modifies the sample rate by a overall factor $\kappa(\alpha)$ constructed as the interpolation between downward ("lo") and upward ("hi") as well as the nominal setting, i.e. $\kappa(-1) = \kappa_{\alpha=-1}$, $\kappa(0) = 1$ and $\kappa(+1) = \kappa_{\alpha=+1}$. In the modifier definition we record $\kappa_{\alpha=+1}$ and $\kappa_{\alpha=-1}$ as floats. An example of a normalisation uncertainty modifier with scale factors recorded for the up/down variations of an n-bin channel is

Display a menu

v: v0.7.0

Adding prior Information

With our toolbox we can quickly compose different modifiers to build up complex

$$p(\text{data}\,|\,\theta) = \prod_{c \in \text{channels}} \prod_{b \in \text{bins}_\text{c}} \text{Pois}(n_{cb}\,|\,\lambda_b = \sum_s \lambda_{csb}(\theta))$$

We could already start fitting those

But we can do more: **add prior information**

Adding prior Information

Just parametrizing is not enough, often we want to give additional information about possible parameter values

Example:

When I say: the background is 50 ± 10 events, I ofen mean it as a type of statistical interval

- $b = 50$ most prefered given what I know
- $b = 40{,}60$ possible
- $b = 10{,}80$ disfavored

How do we express this in our model

Bayesian vs Frequentist

There is a myth that only Bayesian procedures allow you to add such prior information - it's not true, as we will see.

But it's true: in Bayesian it's very simple

Our Experiment

$$p(\text{data} \,|\, \theta) = \prod_{c \in \text{channels}} \prod_{b \in \text{bins}_c} \text{Pois}(n_{cb} \,|\, \lambda_b = \sum_s \lambda_{csb}(\theta))$$

Prior Beliefs over parameter values

$$p(\theta \,|\, \mathcal{D}) = \frac{p(\mathcal{D} \,|\, \theta) p(\theta)}{p(\mathcal{D})}$$

Bayesian vs Frequentist

But where does our prior belief come from? Most likely from a prior measurement!

$$p(\theta) = p(\theta \,|\, \mathrm{data}_{\mathrm{aux}}) = \frac{p(\mathrm{data}_{\mathrm{aux}} \,|\, \theta) p_{\mathrm{ur}}(\theta)}{p(\mathrm{data}_{\mathrm{aux}})}$$

Plugging this in gives us:

$$p(\theta \,|\, \mathrm{data}) = \frac{p(\mathrm{data} \,|\, \theta)}{p(\mathrm{data})} \frac{p(\mathrm{data}_{\mathrm{aux}} \,|\, \theta)}{p(\mathrm{data}_{\mathrm{aux}})} p_{\mathrm{ur}}(\theta)$$

Bayesian vs Frequentist

But where does our prior belief come from? Most likely from a prior measurement!

$$p(\theta) = p(\theta \,|\, \mathrm{data}_{\mathrm{aux}}) = \frac{p(\mathrm{data}_{\mathrm{aux}} \,|\, \theta) p_{\mathrm{ur}}(\theta)}{p(\mathrm{data}_{\mathrm{aux}})}$$

$$p(\theta \,|\, \mathrm{data}) = \boxed{\frac{p(\mathrm{data} \,|\, \theta)}{p(\mathrm{data})} \boxed{\frac{p(\mathrm{data}_{\mathrm{aux}} \,|\, \theta)}{p(\mathrm{data}_{\mathrm{aux}})} \boxed{p_{\mathrm{ur}}(\theta)}}}$$

full information
after all measurements

belief of what we knew
before our measurement

belief before any
measurement was done

Bayesian vs Frequentist

This short Bayesian treatment gives us a hint how to add prior information also in the Frequentist case:

$$p(\theta \,|\, \text{data}) = \boxed{\frac{p(\text{data} \,|\, \theta)}{p(\text{data})} \frac{p(\text{data}_{\text{aux}} \,|\, \theta)}{p(\text{data}_{\text{aux}})}} p_{\text{ur}}(\theta)$$

If our prior information is driven by a prior indepdentnt measurement, we can just use a statistical combination to incorporate it

$$p(\text{data} \,|\, \theta) \to p(\text{data} \,|\, \theta) p(\text{data}_{\text{aux}} \,|\, \theta)$$

Example Aux Measurement

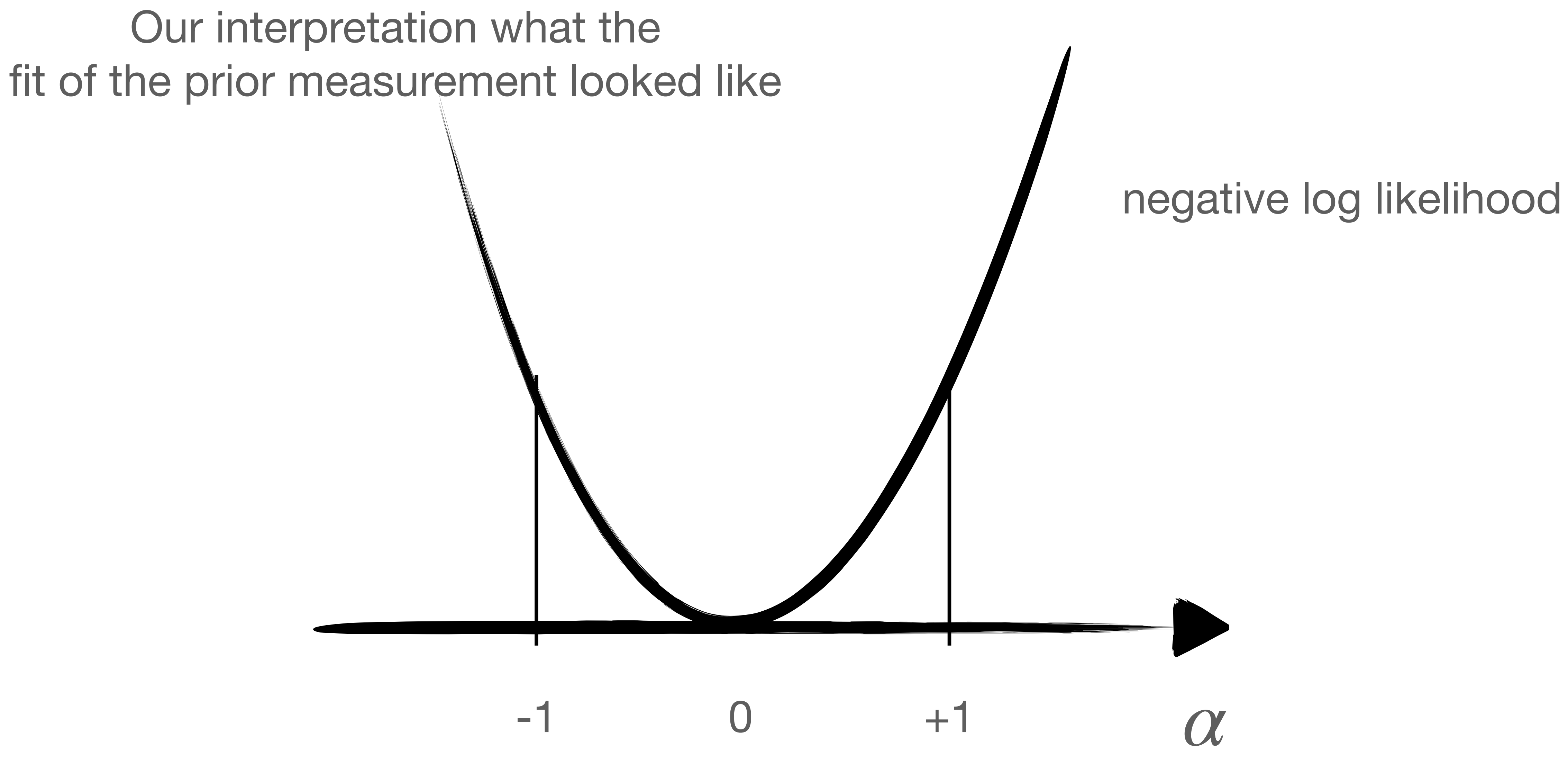
Let's go back to: "my background is X ± Y"

- interpret: there is a prior measurement of data such that $\hat{\alpha} = 0 \pm 1$ would be favored (in some units)

$$\lambda(\alpha = 1) = k_{\mathrm{up}} \lambda^{\mathrm{nom}}$$

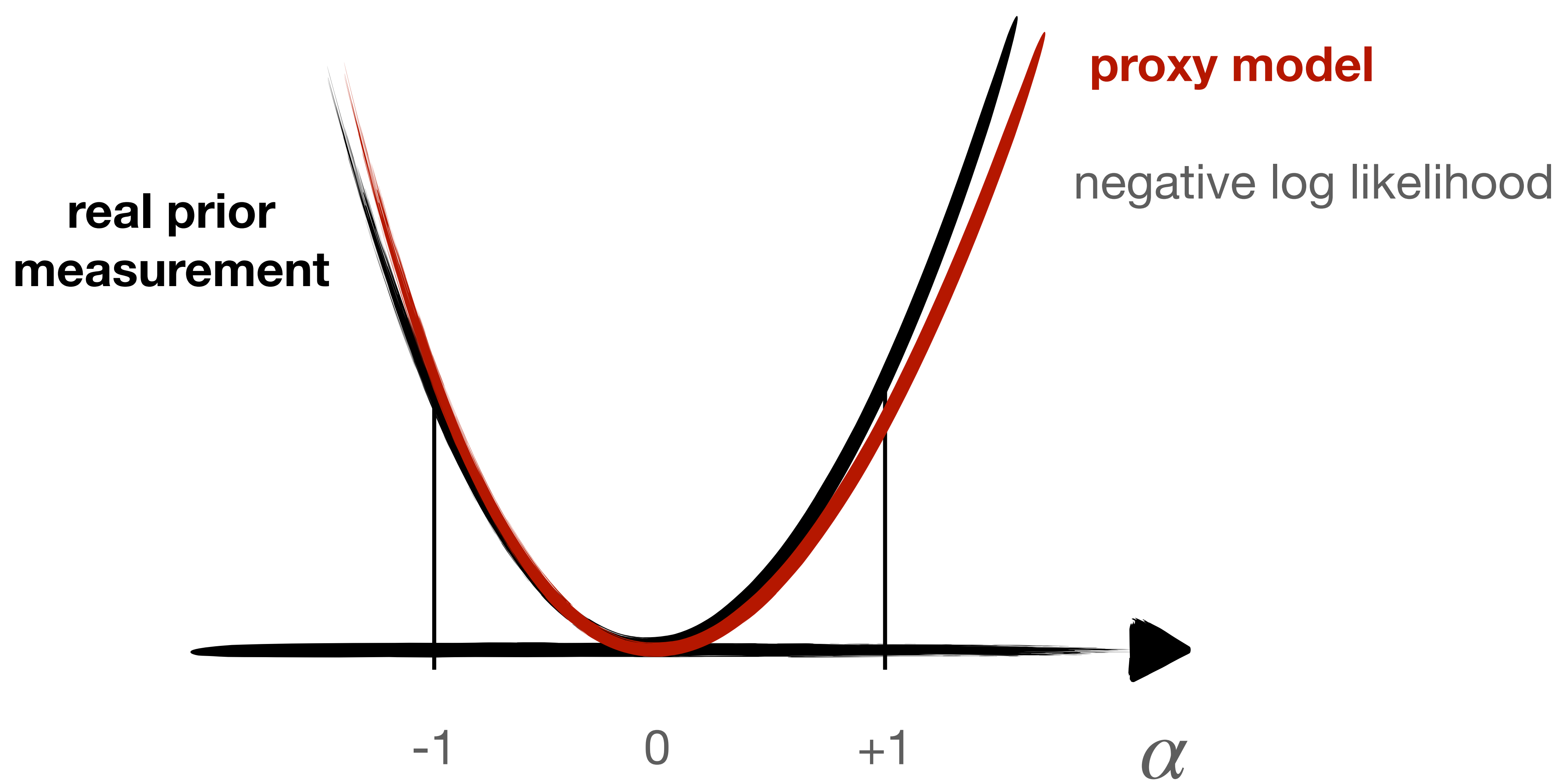
$$\lambda(\alpha = 0) = \lambda^{\mathrm{nom}}$$

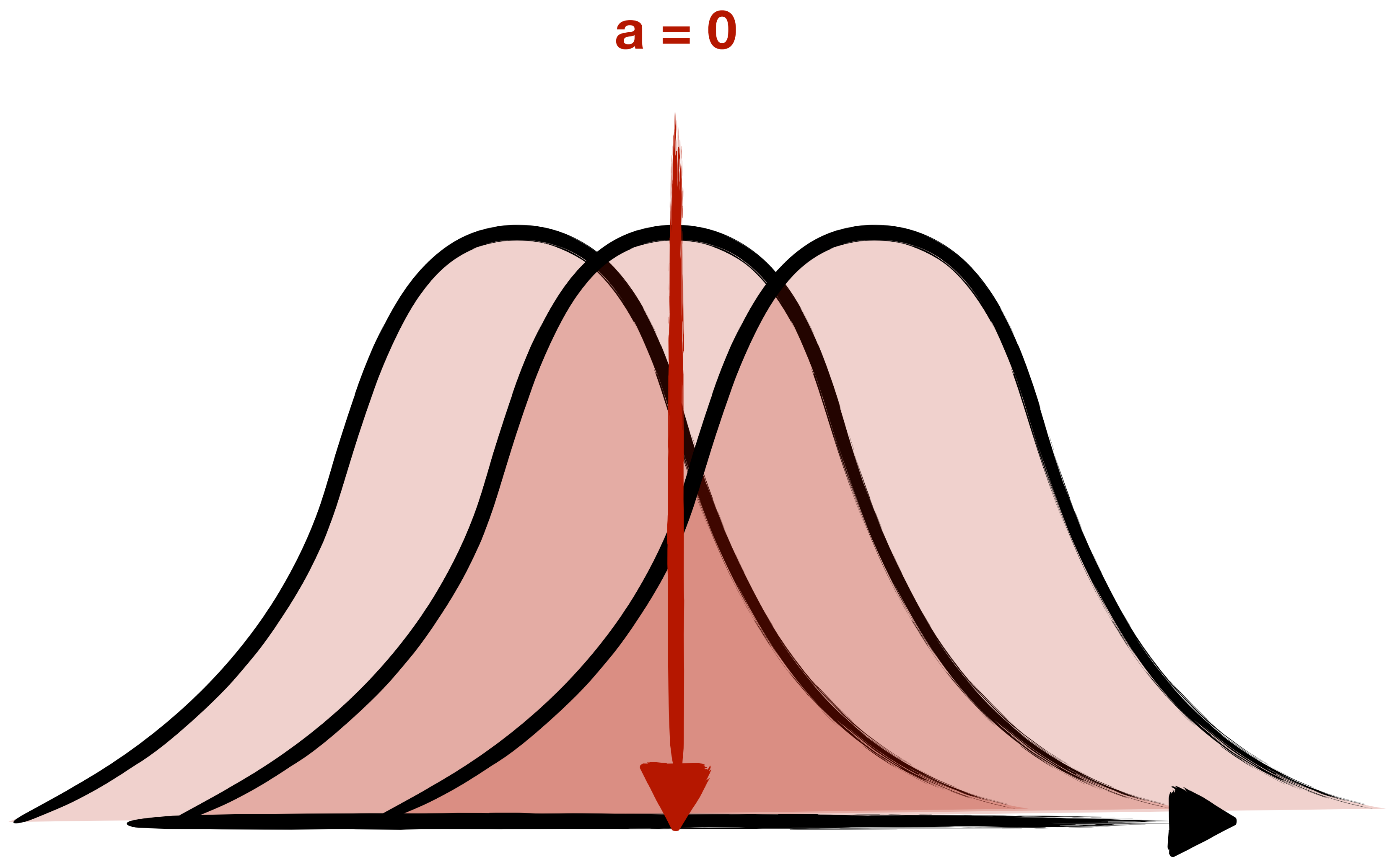
$$\lambda(\alpha = 1) = k_{\mathrm{dn}} \lambda^{\mathrm{nom}}$$



Solution

The solution HistFactory uses is to add “proxy models” that would produce a similar likelihood for the combination





$$p(\mathrm{data}_{\mathrm{aux}} \,|\, \alpha) \approx \boxed{\mathrm{Norm}(a \,|\, \alpha, \sigma = 1)}$$

Modifiers

Some HistFactory modifiers that are associated with "systematic uncertainties" - will automatically add such a term

Correlated Shape

histosys

Uncorrelated Shape

shapesys

MC Stats Variation

staterror

Normalization

normfactor

Luminosity

lumi

Correlated Scaling

normfactor

Uncorrelated Scaling

shapefactor

implies a auxiliary (prior) measurement

no aux. measurement

The full HistFactory Model

Some HistFactory modifiers that are associated with "systematic uncertainties" - will automatically add such a term

Main Measurement — Constraint Terms / Auxiliary Measurent

$$p(\text{data}, \text{data}_{\text{aux}} \,|\, \theta) = \boxed{p(\text{data} \,|\, \theta)}\ \boxed{p(\text{data}_{\text{aux}} \,|\, \theta)}$$

$$\prod_{c \in \text{channels}} \prod_{b \in \text{bins}_c} \text{Pois}(n_{cb} \,|\, \lambda_b = \sum_s \lambda_{csb}(\theta))$$

$$\prod_{a \in A} p(\text{aux}_a \,|\, \alpha)$$

The full HistFactory Model

```
spec = {
    'channels': [
        {
            'name': 'channel1',
            'samples': [
                {'name': 'signal', 'data': [5,10],
                 'modifiers': [
                    {'type': 'normfactor', 'name': 'mu', 'data': None}

                ]},
                {'name': 'background', 'data': [100,50],
                 'modifiers': [
                    {'type': 'normsys', 'name': 'uncrt', 'data': { 'hi': 1.1, 'lo': 0.9 }}

                ]},
            ]
        }
    ]
}
```

```
model = pyhf.Model(spec)
model.expected_actualdata([1.0,0.0])
```

```
array([105.,  60.])
```

```
model.expected_actualdata([1.0,1])
```

```
array([115.,  65.])
```

```
model.expected_actualdata([1.0,-1])
```

```
array([95., 55.])
```

The full HistFactory Model

We can actually see the auxiliary data in action:

```
model = pyhf.Model(spec)
model.expected_actualdata([1.0,0.0])
```

```
array([105.,  60.])
```

```
model.expected_actualdata([1.0,1])
```

```
array([115.,  65.])
```

```
model.expected_actualdata([1.0,-1])
```

```
array([95., 55.])
```

```
model.expected_data([1.0,0.0])
```

```
array([105.,  60.,   0.])
```

```
model.expected_data([1.0,1.0])
```

```
array([115.,  65.,   1.])
```

```
model.expected_data([1.0,-1.0])
```

```
array([95., 55., -1.])
```

```
model = pyhf.Model(spec)
model.config.auxdata
```

```
[0.0]
```

Evaluating the PDF

Take this example: what is the value of the likelihood at $\theta = (\mu = 0, \alpha = 1)$ if we observe N = [102,48] ?

```
spec = {
    'channels': [
        {
            'name': 'channel1',
            'samples': [
                {'name': 'signal', 'data': [5,10],
                 'modifiers': [
                    {'type': 'normfactor', 'name': 'mu', 'data': None}

                ]},
                {'name': 'background', 'data': [100,50],
                 'modifiers': [
                    {'type': 'normsys', 'name': 'uncrt', 'data': { 'hi': 1.1, 'lo': 0.9 }}

                ]},
            ]
        }
    ]
}
```

$$\mathrm{Pois}([102,48]\,|\,\lambda(0,1)) = \mathrm{Pois}([102,48]\,|\,\lambda(0,1))$$

$$= \mathrm{Pois}([102,48]\,|\,[110,55])$$

$$\mathrm{Norm}(0\,|\,\alpha = 1)$$

Evaluating the PDF

Take this example: what is the value of the likelihood at $\theta = (\mu = 0, \alpha = 1)$ if we observe N = [102,48] ?

```
spec = {
    'channels': [
        {
            'name': 'ch
            'samples':
                {'name'
                 'modif
                    {'t

                ]},
                {'name'
                 'modif
                    {'t

                ]},
            ]
        }
    ]
}
```

```
model.logpdf([0.0,1.0], [102,48] + model.config.auxdata)
```

```
array([-8.27134087])
```

```
import scipy.stats
main_term = scipy.stats.poisson([110,55]).logpmf([102,48]).sum()
aux_term = scipy.stats.norm(0).logpdf(1.0)
print(main_term + aux_term)
```

```
-8.271340865142852
```

$$\mathrm{Pois}([102,48]\,|\,\lambda(0,1)) = \mathrm{Pois}([102,48]\,|\,\lambda(0,1))$$
$$= \mathrm{Pois}([102,48]\,|\,[110,55])$$

$$\mathrm{Norm}(0\,|\,\alpha = 1)$$

Evaluating the PDF

Upshot: HistFactory is not magic, it's just a useful toolbox to help you model your measurement

pyhf: implementation of HistFactory in python

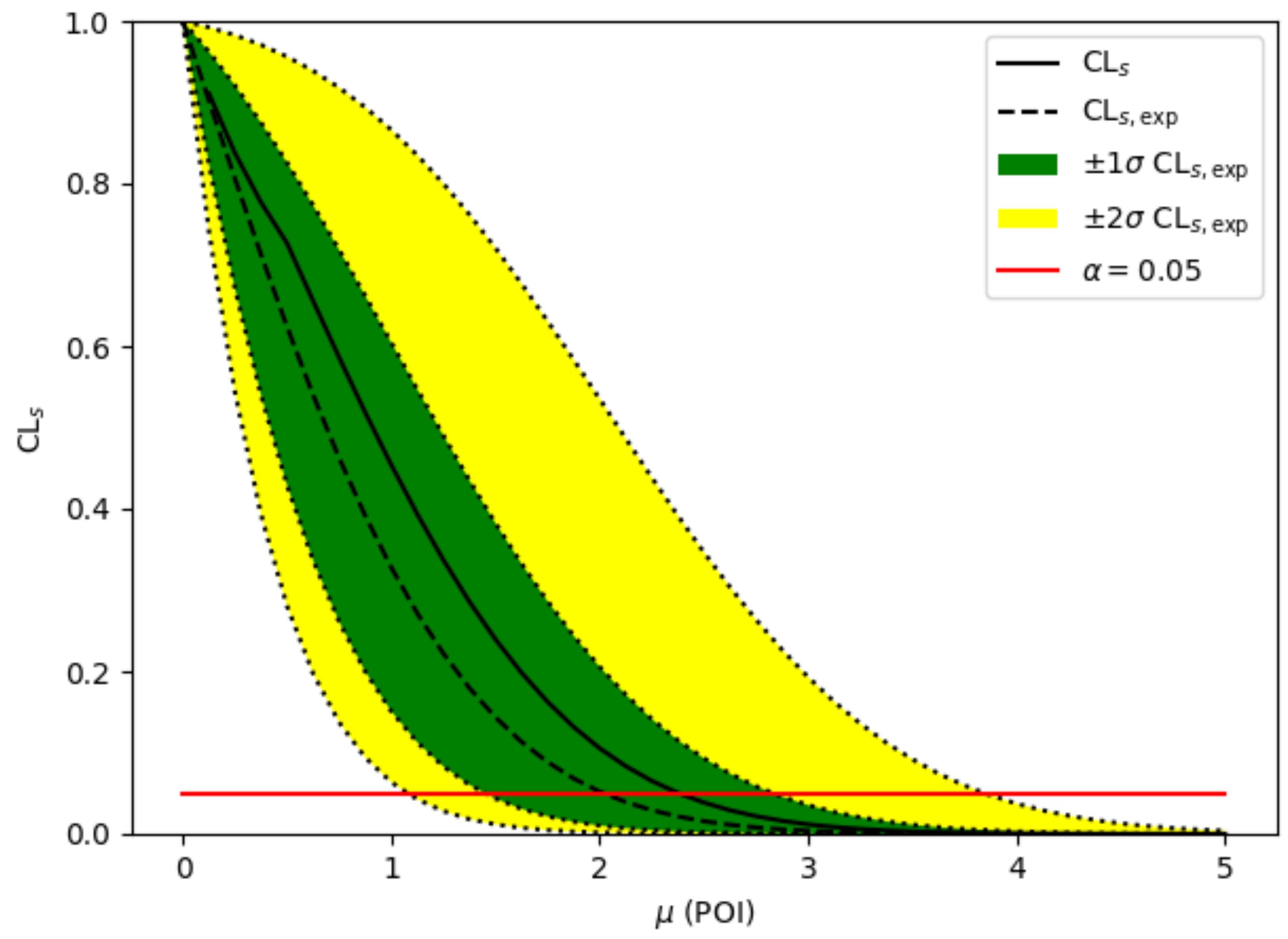
(interpolation algorithms, bookkeeping of constraint terms, …)

Beyond just the Model

Beyond the model building, pyhf also comes with “batteries included” for some basic things

- Fitting / Limit Setting, Basic Plotting

```
>>> import pyhf
>>> pyhf.set_backend("numpy")
>>> model = pyhf.simplemodels.uncorrelated_background(
...     signal=[12.0, 11.0], bkg=[50.0, 52.0], bkg_uncertainty=[3.0, 7.0]
... )
>>> data = [51, 48] + model.config.auxdata
>>> test_mu = 1.0
>>> CLs_obs, CLs_exp = pyhf.infer.hypotest(
...     test_mu, data, model, test_stat="qtilde", return_expected=True
... )
>>> print(f"Observed: {CLs_obs:.8f}, Expected: {CLs_exp:.8f}")
Observed: 0.05251497, Expected: 0.06445321
```



But for more advanced use, best to use a library around pyhf e.g. cabinetry (next talk)

Next Steps in pyhf

The core HistFactory model has been stable for >10 years

- we ***know*** a huge amount of physics can be modelled with it

But some use-cases need to go beyond (see Lorenz' talk)

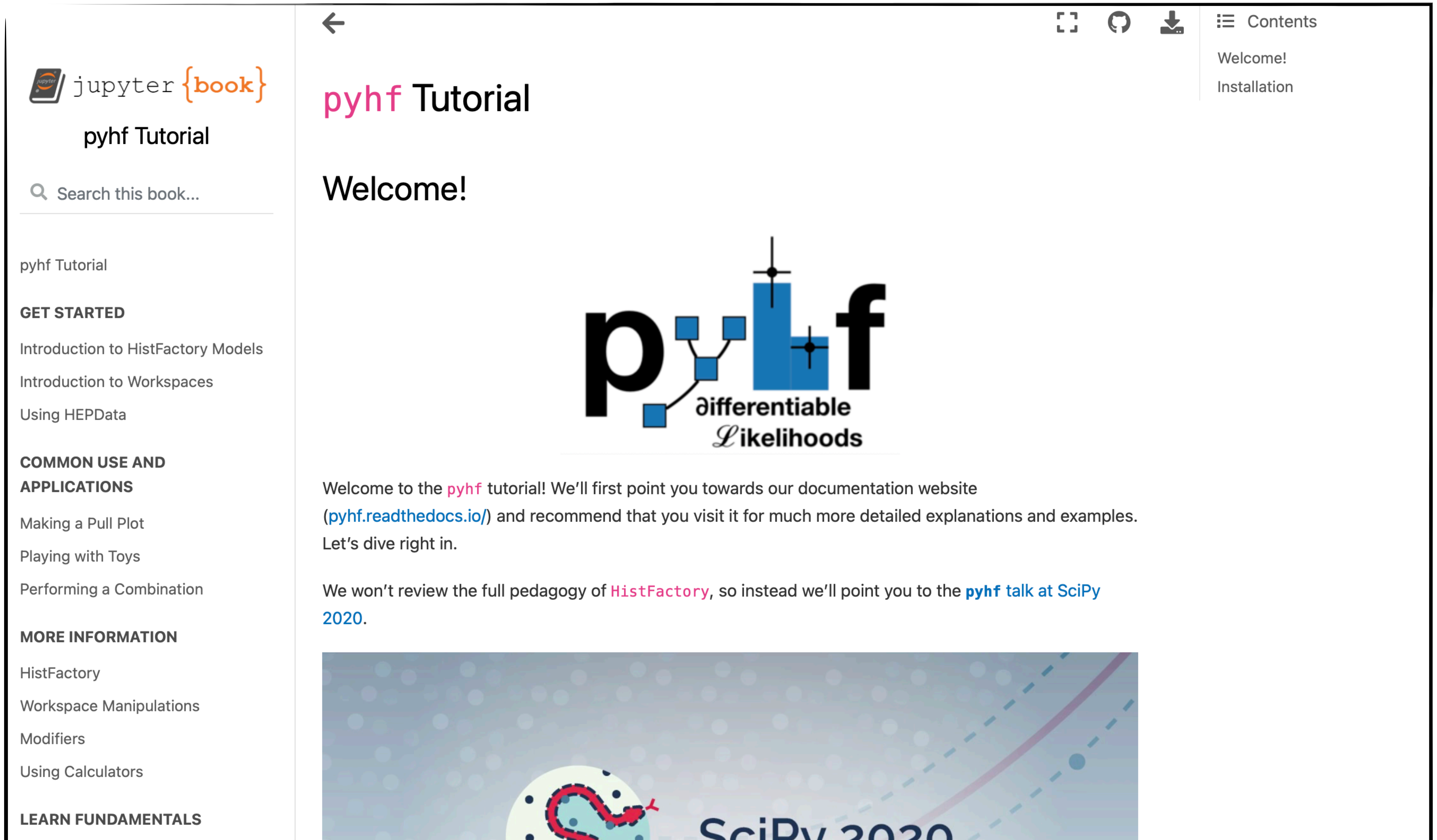
- working on making this easy to to
- custom modifiers (possibly ML-based interpolation)

pyhf is just the model $p(x\,|\,\theta)$ - no reason to be frequentist only

- working on fully consistent Bayesian APIs

Resources

https://pyhf.github.io/pyhf-tutorial/introduction.html



jupyter {book}

pyhf Tutorial

Search this book...

pyhf Tutorial

GET STARTED

Introduction to HistFactory Models

Introduction to Workspaces

Using HEPData

COMMON USE AND APPLICATIONS

Making a Pull Plot

Playing with Toys

Performing a Combination

MORE INFORMATION

HistFactory

Workspace Manipulations

Modifiers

Using Calculators

LEARN FUNDAMENTALS

Contents

Welcome!

Installation

`pyhf` Tutorial

Welcome!

pyhf ∂ifferentiable ℒikelihoods

Welcome to the `pyhf` tutorial! We'll first point you towards our documentation website (pyhf.readthedocs.io/) and recommend that you visit it for much more detailed explanations and examples. Let's dive right in.

We won't review the full pedagogy of `HistFactory`, so instead we'll point you to the `pyhf` talk at SciPy 2020.

SciPy 2020