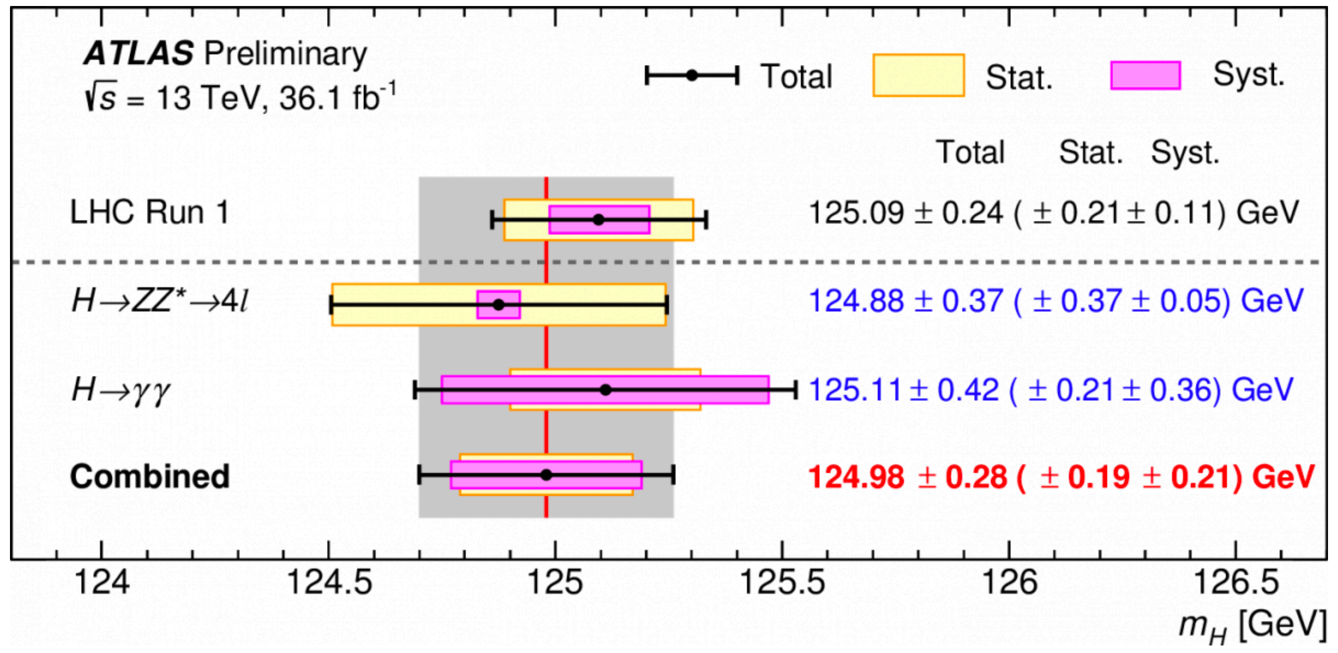
# Publishing Statistical Models

Lukas Heinrich, TUM

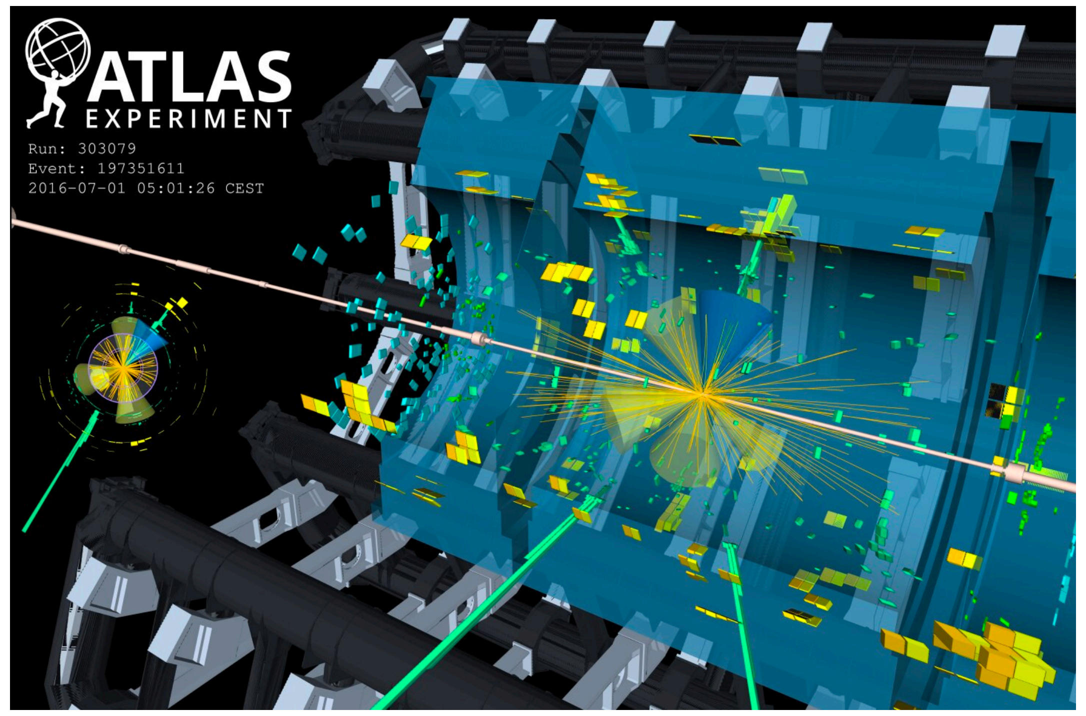
# Big Picture Goals

## Our job: extract as much information from experimental data as possible

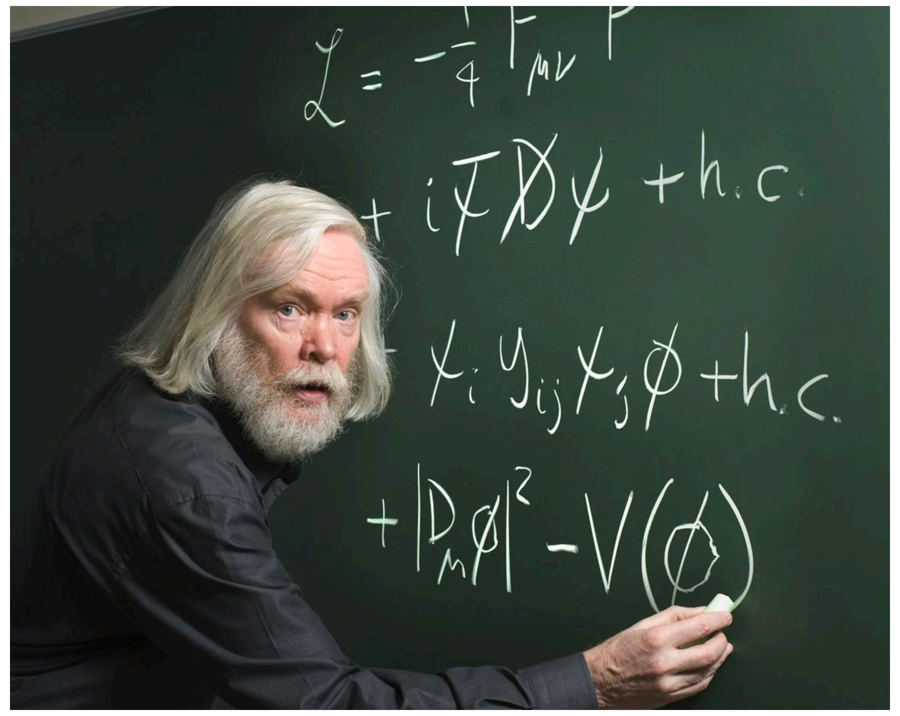
$$p(\text{theory}|\text{data}) = \frac{p(\text{data}|\text{theory})}{p(\text{data})}\,p(\text{theory})$$




results / insight




experimentalists



theorists

# Big Picture Goals

## Our job: extract as much information from experimental data as possible

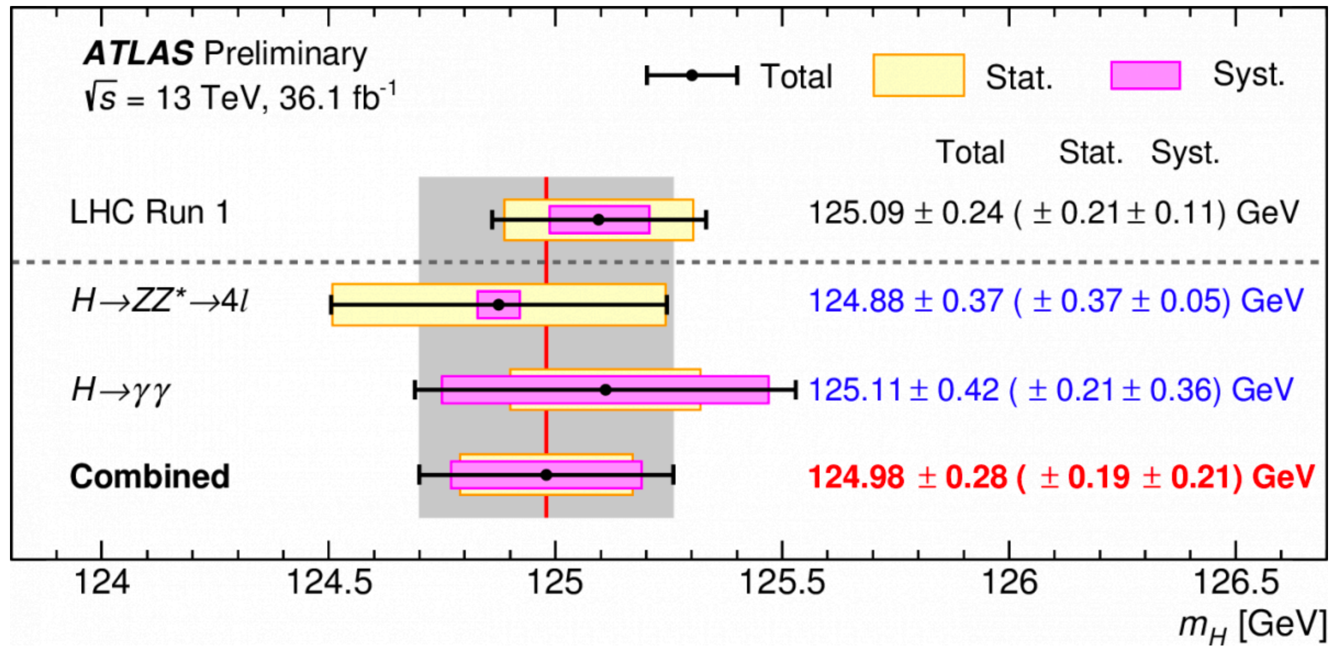
The Likelihood: Focus of this talk

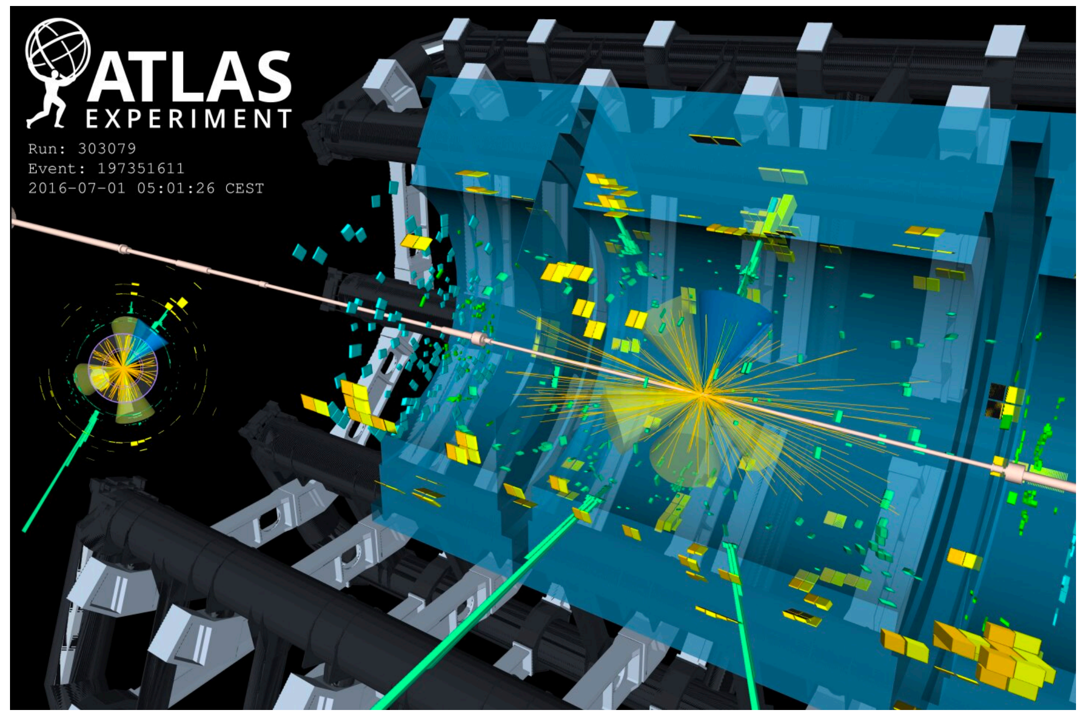
$$p(\text{theory}|\text{data}) = \frac{\boxed{p(\text{data}|\text{theory})}}{p(\text{data})}\, p(\text{theory})$$

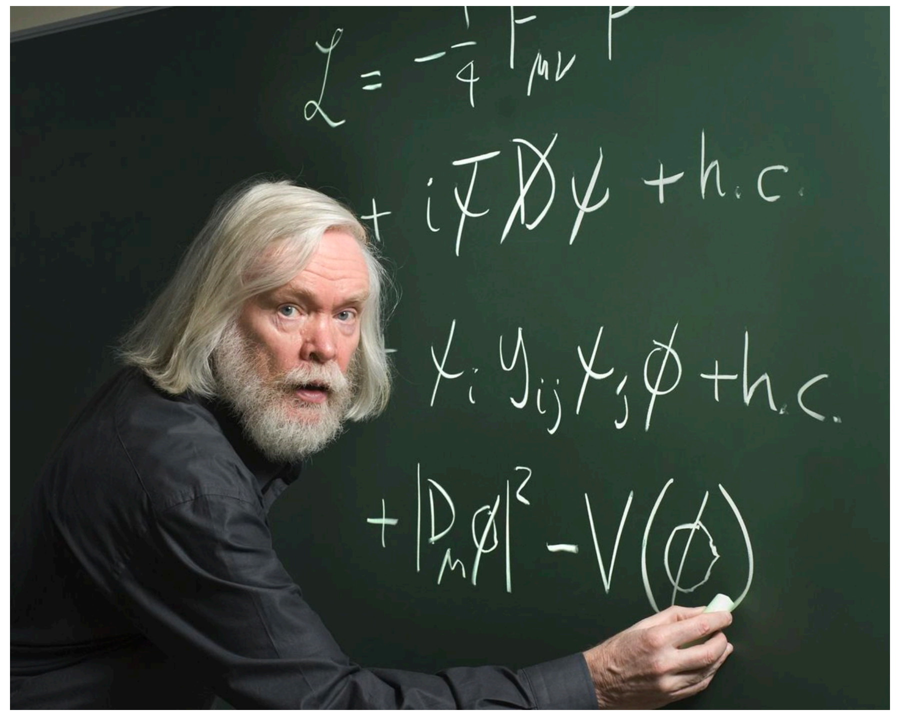
Posterior

Evidence

Prior



results / insight

experimentalists

theorists


**Our data is huge:**

- **we need think hard how we summarize our results to the wider community**

**Our experiments are unique:**

- **These are once-in-a-lifetime machines. Need to preserve data in as much detail as we can, in a format that can be archived for the long-term**

**Search for suitable data products for HEP**

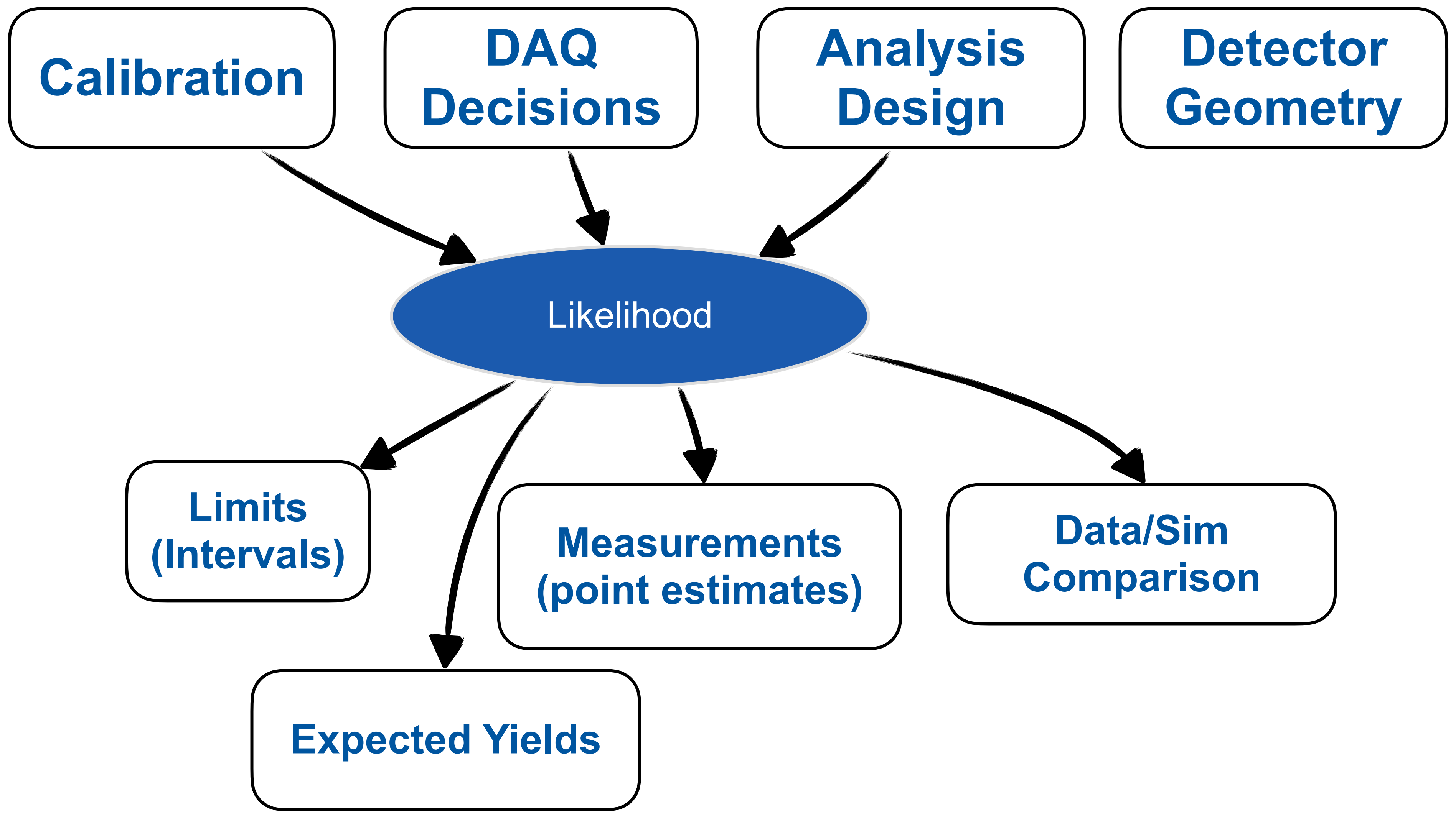
# The Likelihood is unique!

## Likelihoods are a good bottleneck through which all information flows

It's a high information-density product

- almost every important decision is reflected in the likelihood
  (if it doesn't affect the likelihood, what are you doing?)

- all the usual results are inferences based on the likelihood (downstream)

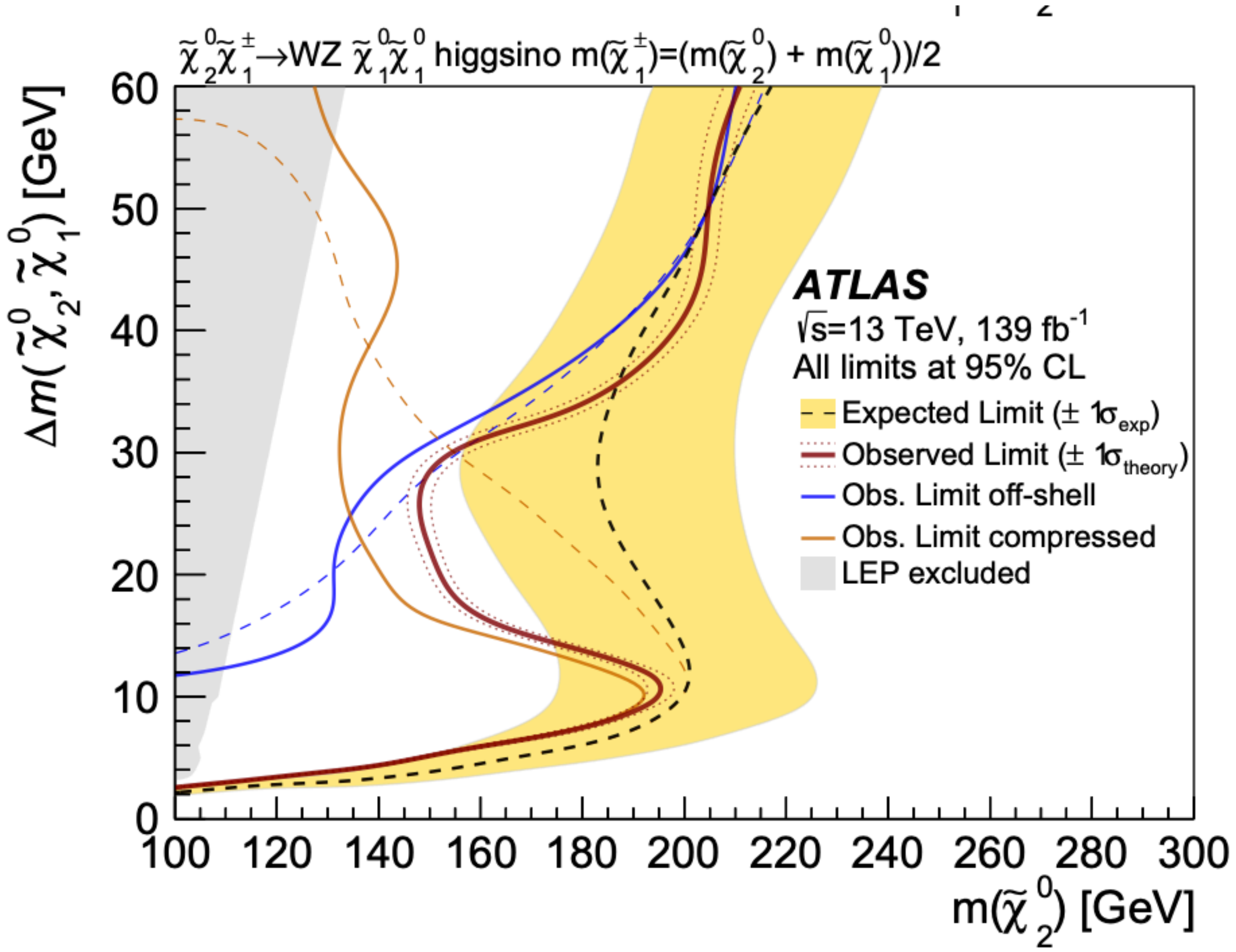


# What you can do with a Likelihood

**Likelihoods are not only useful to <u>recreate</u> results.**
**You can use them to generate new scientific results!**

## 1. Statistical Combination

**e.g. global fits of multiple experiments / analyses**




$$p(x_1 \,|\, s_1(\theta) + b_1)\; p(a \,|\, \theta_{\mathrm{NP}})$$

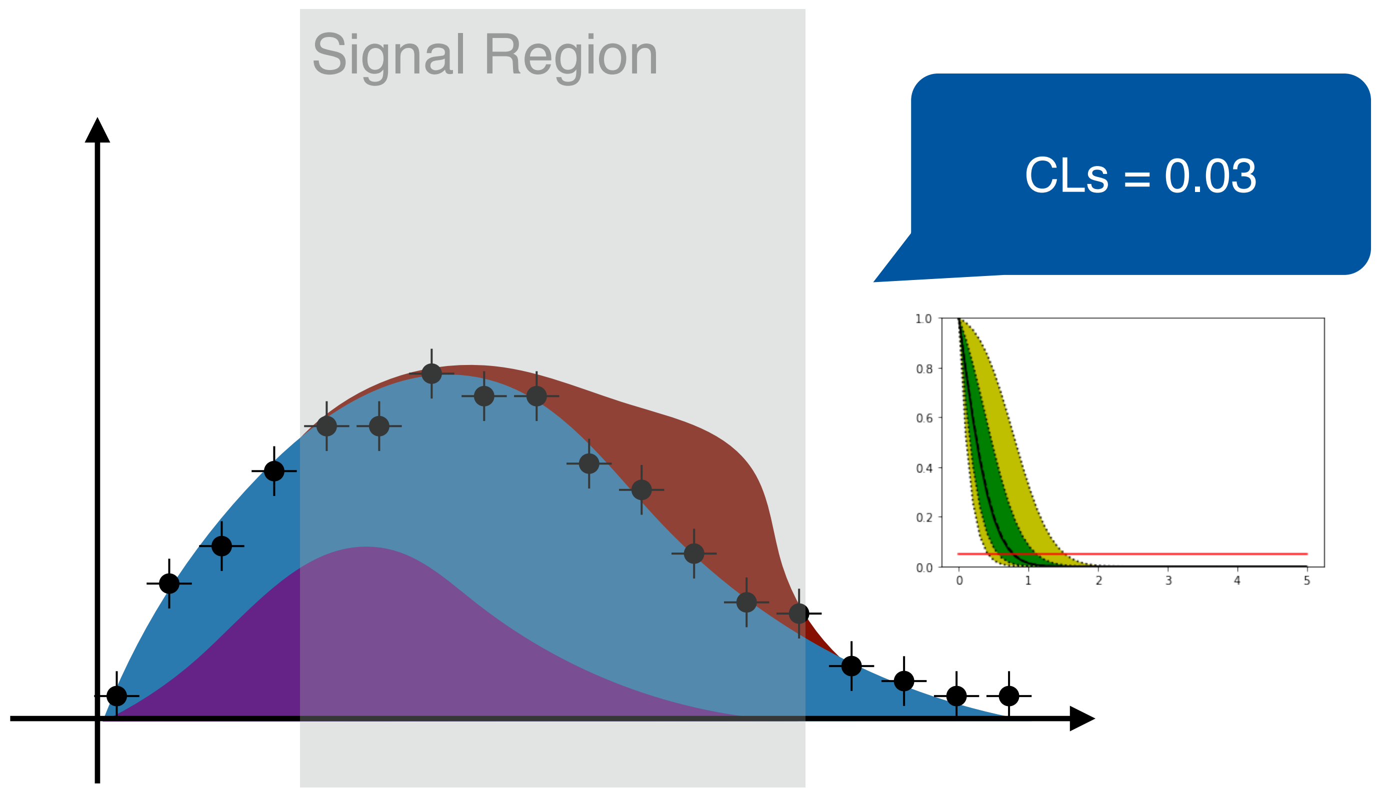
$$p(x_2 \,|\, s_2(\theta) + b2)\; p(a \,|\, \theta_{\mathrm{NP}})$$

$$p(x_1 \,|\, s_1(\theta) + b_1)\, p(x_2 \,|\, s_2(\theta) + b_2)\; p(a \,|\, \theta_{\mathrm{NP}})$$

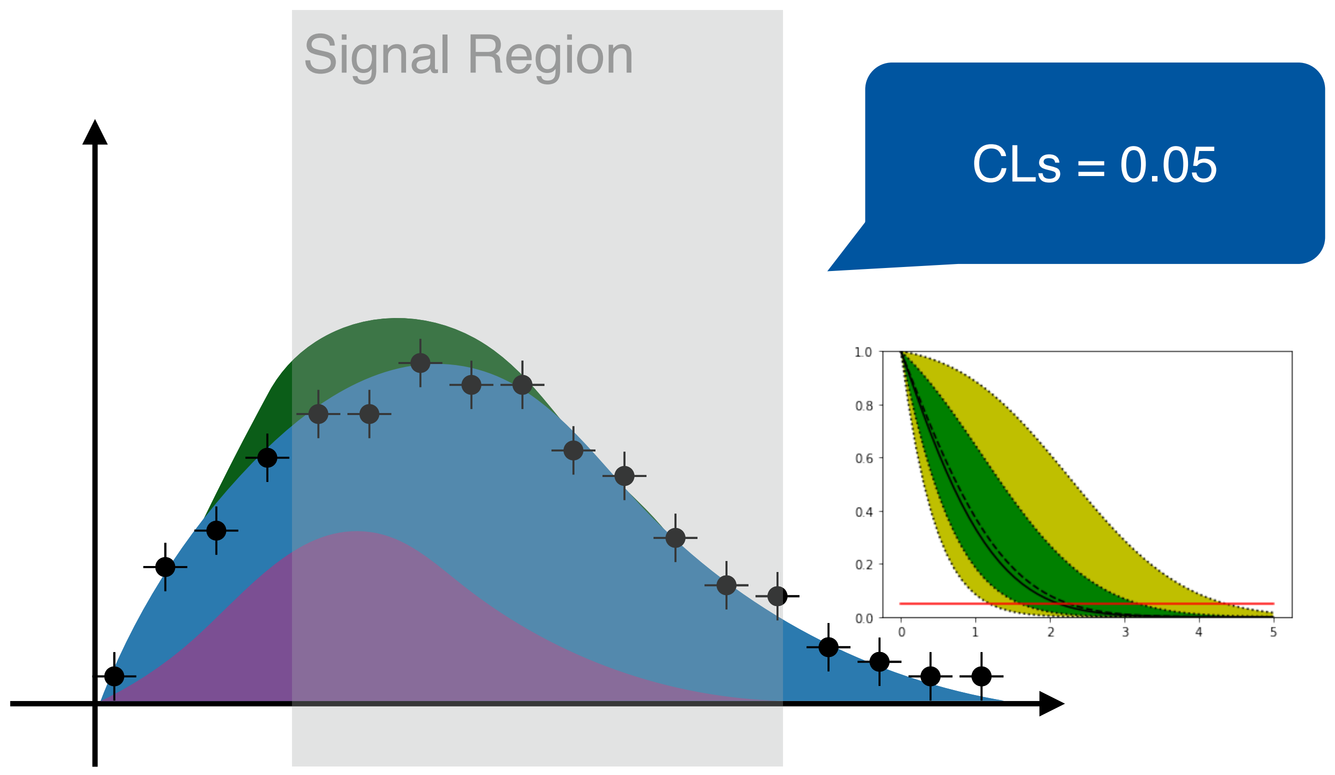
# What you can do with a Likelihood

## 2. Reinterpretation:

**Modify the ingredients of a likelihood (e.g. change signal component) and re-run the statistical analysis. Note: needs sufficiently detailed information to do this!**

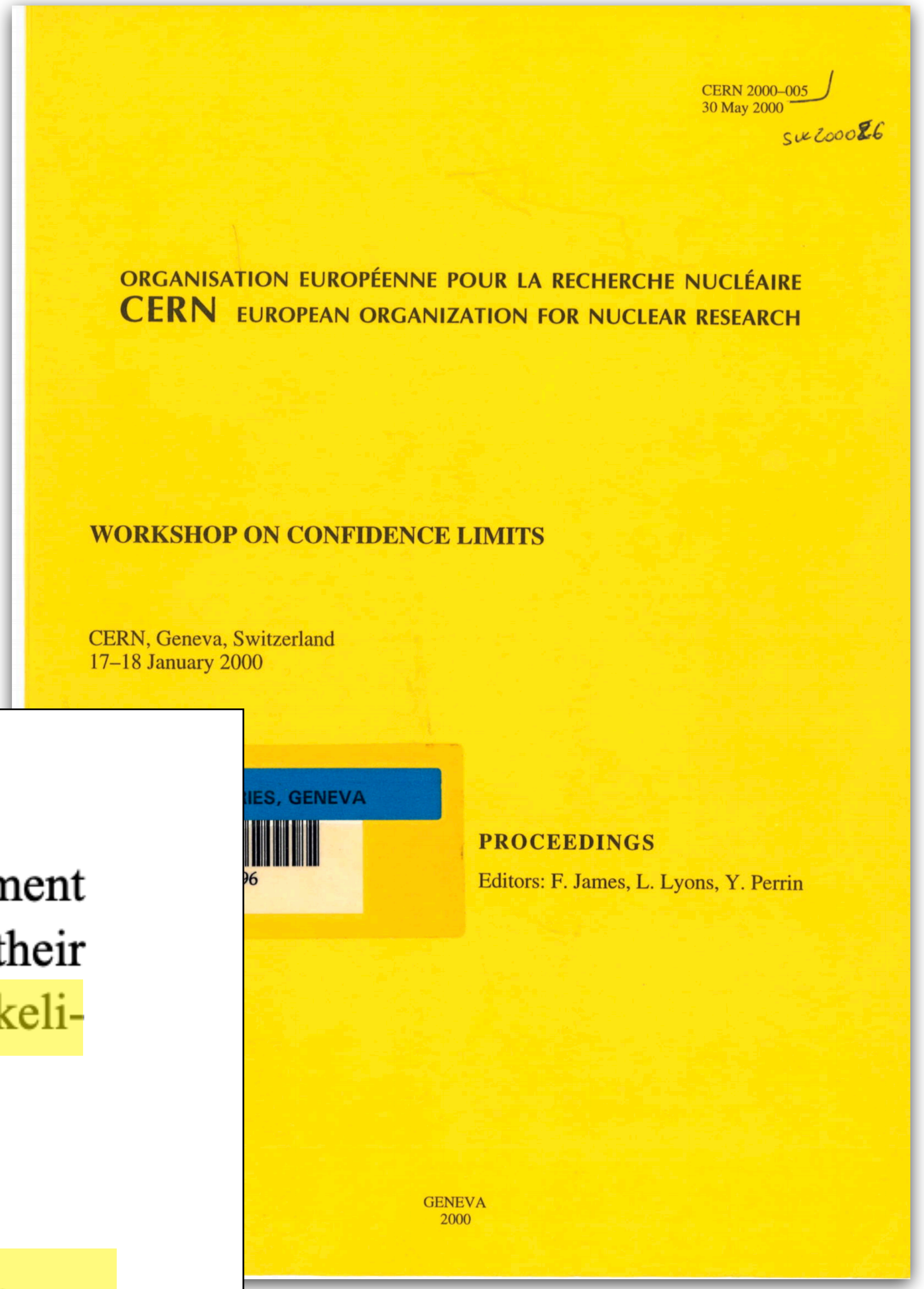



$$\mathrm{Pois}(n\,|\,\mu\, s_A(\theta) + b)\ldots$$




$$\mathrm{Pois}(n\,|\,\mu\, s_B(\phi) + b)\ldots$$

# A (in-)consequential workshop


CERN 2000-005
30 May 2000

ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

WORKSHOP ON CONFIDENCE LIMITS

CERN, Geneva, Switzerland
17–18 January 2000

PROCEEDINGS
Editors: F. James, L. Lyons, Y. Perrin

GENEVA
2000

## Uniqueness of likelihood model as a data product has been recognized 20 years ago

## 1st PHYSTAT: meeting between statisticians + physicists

**Massimo Corradi**

It seems to me that there is a general consensus that what is really meaningful for an experiment is *likelihood*, and almost everybody would agree on the prescription that experiments should give their likelihood function for these kinds of results. Does everybody agree on this statement, to publish likelihoods?

**Louis Lyons**

Any disagreement ? Carried unanimously. That's actually quite an achievement for this Workshop.

## Discussion arrived at conclusion: Experiments should publish likelihoods!

(Spoiler: it didn't happen for a long time)

# 20 Years Later

## Publishing Likelihoods is becoming a thing. Let's see how it works!

### New open release streamlines interactions with theoretical physicists

The ATLAS Collaboration has released the first open likelihoods from an LHC experiment.

12th December 2019 | By Katarina Anthony

# The difficulty in publishing likelihoods

**What exactly should we publish?**

**The likelihood function with data forever fixed ?**

$$L(\theta) = L(\mu, \nu) = p(x|\mu, \mu)$$

**The profile likelihood ratio function (fixed data and fixed profiled NP) ?**

$$t(\mu) = \frac{L(\mu, \hat{\hat{\nu}}(\mu))}{L(\hat{\mu}, \hat{\hat{\nu}}(0))}$$

**Both of these are not great:**

- **just the Likelihood Function cannot be used for reinterpretation**
- **for Frequentist inference we need to be able to sample $x \sim p(x|\theta)$**
- **for combinations we need be able to vary to NP-values**

$$\hat{\hat{\nu}}(x_1) \neq \hat{\hat{\nu}}(x_2) \neq \hat{\hat{\nu}}((x_1, x_2))$$

## The difficulty in publishing likelihoods

**Gold Standard: publish the full model $p(x|\theta)$!**

- **can be use for Frequentist and Bayes Inference**
- **enables combinations, enabled reinterprations through inspection**

**When we say "publish the likelihood" <u>we mean</u> "publish the model $p(x|\theta)$"**

**A typical HEP probability model consists of two parts**

- **you should be preserving both**

$$p(x|\theta) = p_{\mathrm{main}}(x|\mu,\nu) \cdot p_{\mathrm{aux}}(x_a|\nu)$$

**Main Measurements**
**(your analysis)**

**Constraint Terms**
**(simplified summary of e.g. your collab's calibration measurements)**

## How to preserve $p(x \mid \theta)$

Our goals for preserving $p(x \mid \theta)$ should be

- software independent:
  we want to capture the mathematical structure of $p(x \mid \theta)$ not the software that implements it

- long-term archival format

  we want to publish the data on e.g. HepData to be used for decades to come

- optimized for reuse

  reinterpretation / combination should be first-class operations

## Choosing Building Blocks

Almost no constraints to a likelihood $p(x\,|\,\theta)$ except for normalization

If you want to really allow "any" function ("open world of all models") to be preserved, you're back to preserving software

```
double my_lhood(double* theta){
  ....
  double L = ...
  return L
}
```

```
def my_lhood(data, theta):
    L = ...
    return L
```

To have any chance at preserving in a software-independent way, you need to restrict yourself: choose a finite number of building blocks

## High- and Low-level Languages

In standard programming, we have high level and and low-level languages

- think: Python vs C++ vs Assembly code. The high-level languages operate usually at higher levels of abstraction
  - high: allow concise description of complex settings
  - low: more freedom, express things that are not possible at high level
- often: high-level languages are implemented in low-level ones

In probabilistic modelling[1] we see the same:

- high-level modelling: few building blocks, lots of assumptions
- low-level modelling: almost "open world", more freedom & more complex

[1](sometimes also called prob. programming)

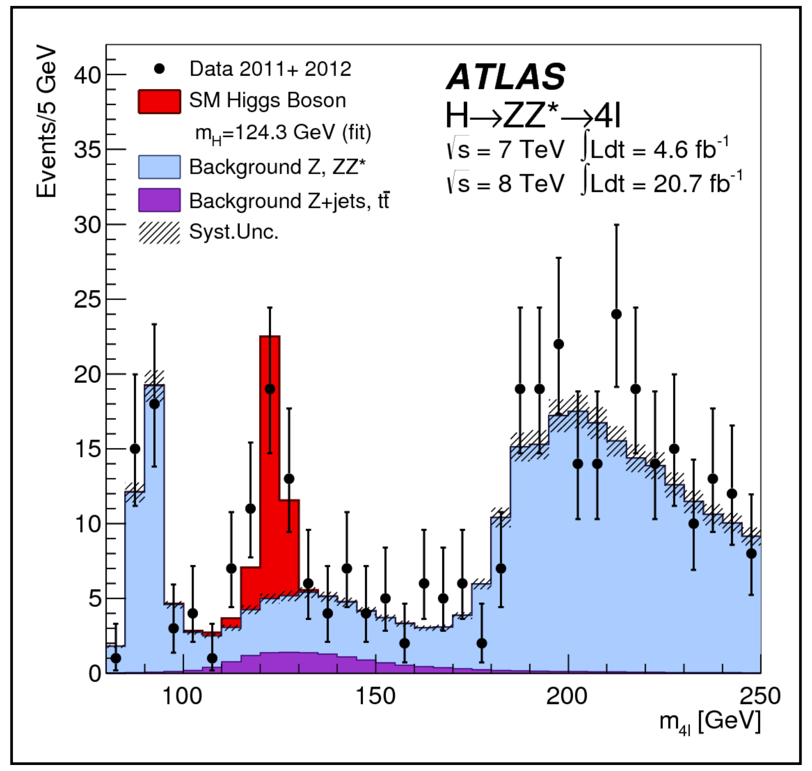
# HistFactory

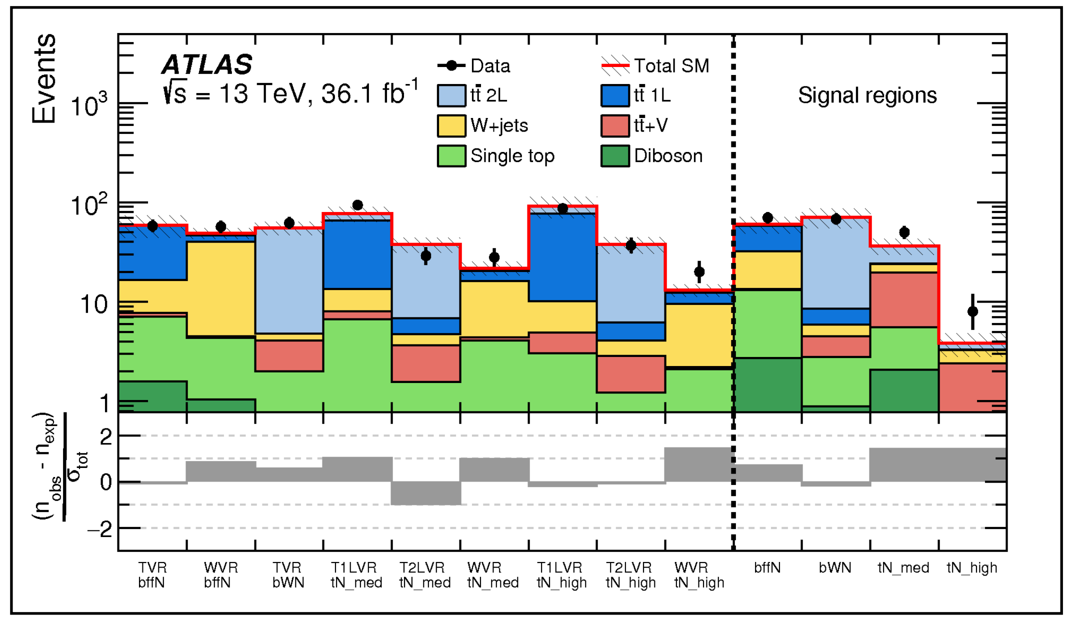
## HistFactory is an example of a high-level language

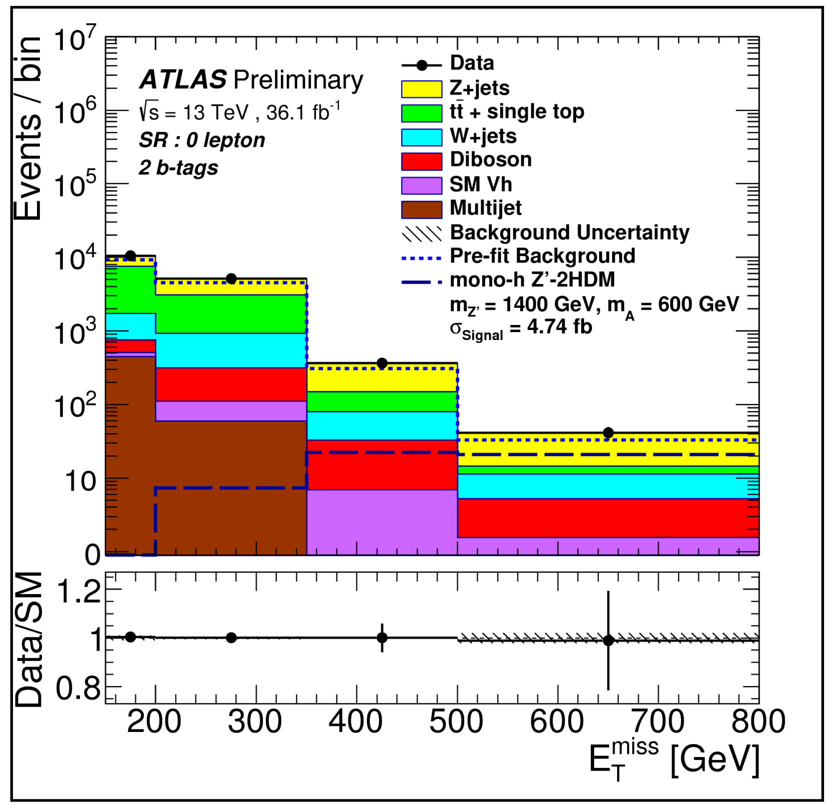
- only supports **binned models**
- systematic modeling only through a fixed (small set) of options

## Despite constraints, it's very versatile (good choice of building blocks!)

- almost all binned analyses in e.g. ATLAS use HistFactory



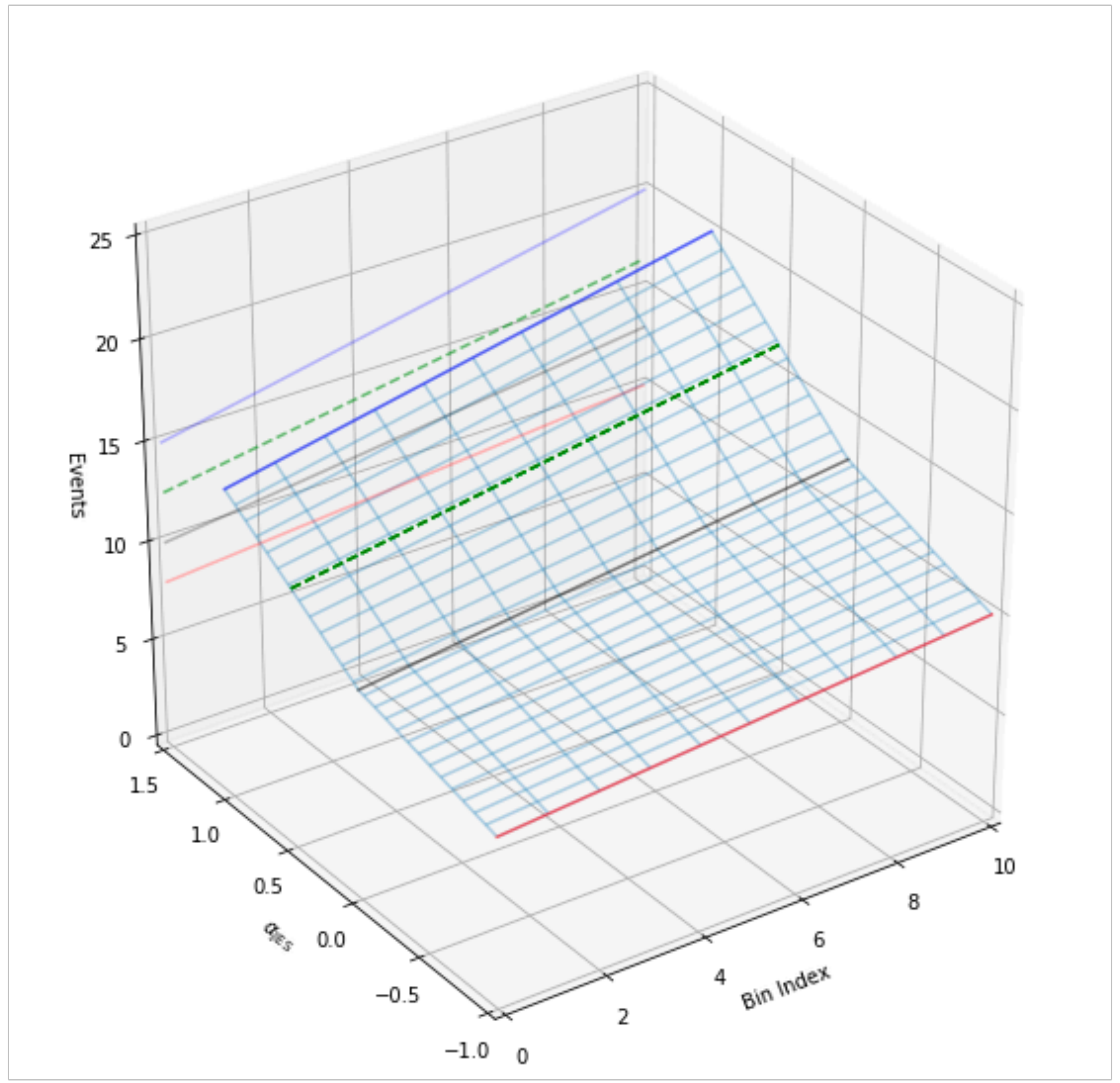




## Implemented in two "low-level" languages:

**pyhf: scipy.stats (Python)** **ROOT HistFactory: RooFit (C++)**

# HistFactory




## Building Blocks of HistFactory:

- nominal histogram shapes
- a fixed set of systematic types
  - chosen to be reusable in many contexts
  - auto-matched with appropriate constraint termss

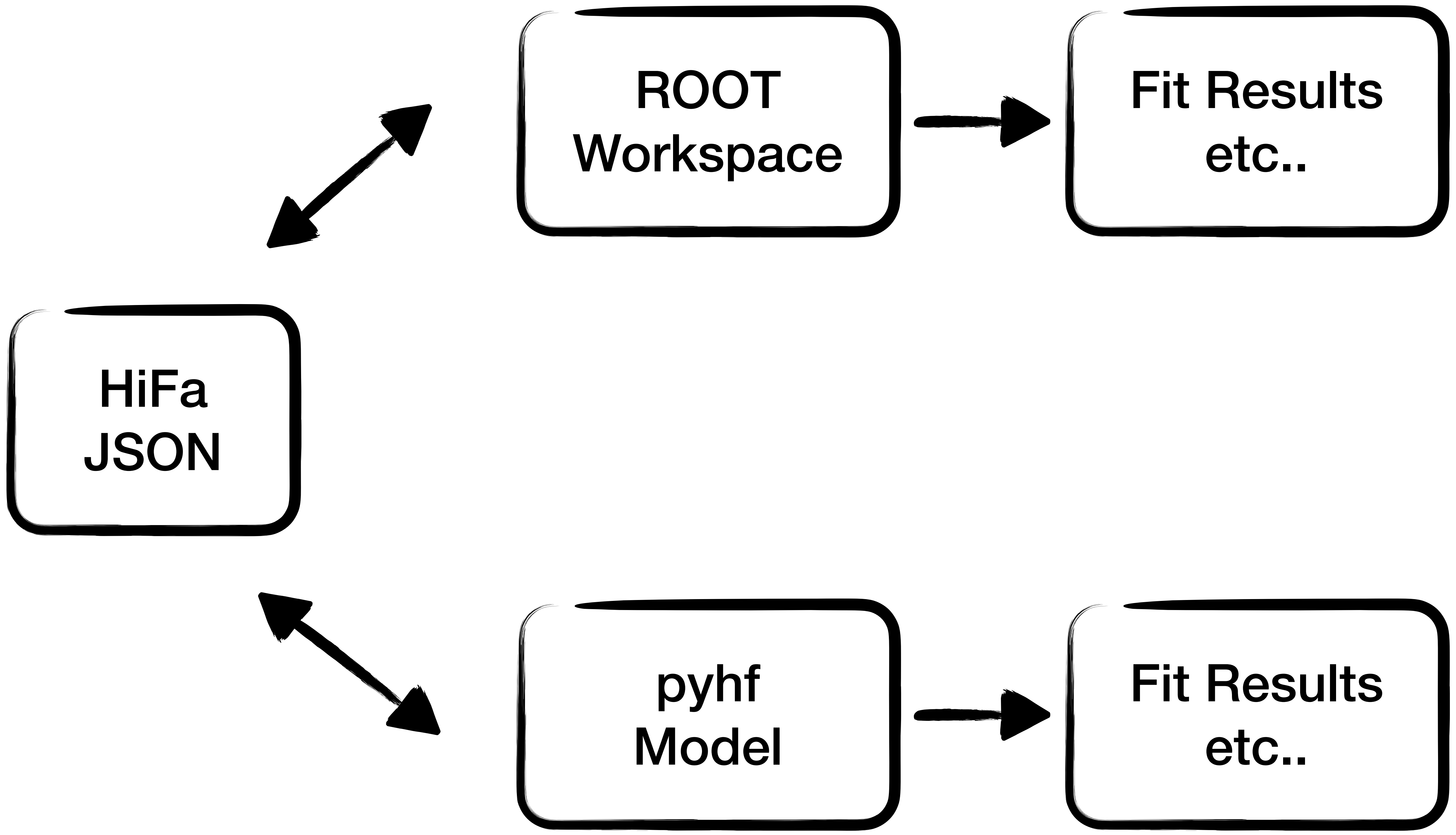
## See tutorial at: [Link]

| Description | Modification | Constraint Term $c_\chi$ | Input |
|---|---|---|---|
| Uncorrelated Shape | $\kappa_{scb}(\gamma_b) = \gamma_b$ | $\prod_b \mathrm{Pois}\left(r_b = \sigma_b^{-2} \mid \rho_b = \sigma_b^{-2}\gamma_b\right)$ | $\sigma_b$ |
| Correlated Shape | $\Delta_{scb}(\alpha) = f_p\left(\alpha \mid \Delta_{scb,\alpha=-1}, \Delta_{scb,\alpha=1}\right)$ | $\mathrm{Gaus}\left(a = 0 \mid \alpha, \sigma = 1\right)$ | $\Delta_{scb,\alpha=\pm 1}$ |
| Normalisation Unc. | $\kappa_{scb}(\alpha) = g_p\left(\alpha \mid \kappa_{scb,\alpha=-1}, \kappa_{scb,\alpha=1}\right)$ | $\mathrm{Gaus}\left(a = 0 \mid \alpha, \sigma = 1\right)$ | $\kappa_{scb,\alpha=\pm 1}$ |
| MC Stat. Uncertainty | $\kappa_{scb}(\gamma_b) = \gamma_b$ | $\prod_b \mathrm{Gaus}\left(a_{\gamma_b} = 1 \mid \gamma_b, \delta_b\right)$ | $\delta_b^2 = \sum_s \delta_{sb}^2$ |
| Luminosity | $\kappa_{scb}(\lambda) = \lambda$ | $\mathrm{Gaus}\left(l = \lambda_0 \mid \lambda, \sigma_\lambda\right)$ | $\lambda_0, \sigma_\lambda$ |
| Normalisation | $\kappa_{scb}(\mu_b) = \mu_b$ | | |
| Data-driven Shape | $\kappa_{scb}(\gamma_b) = \gamma_b$ | | |

# HistFactory JSON

## Part of pyhf a new software-independent JSON format for HistFactory:

```
$ cat << EOF  | tee likelihood.json | pyhf cls
{
    "channels": [
        { "name": "singlechannel",
          "samples": [
            { "name": "signal",
              "data": [12.0, 11.0],
              "modifiers": [ { "name": "mu", "type": "normfactor", "data": null} ]
            },
            { "name": "background",
              "data": [50.0, 52.0],
              "modifiers": [ {"name": "uncorr_bkguncrt", "type": "shapesys", "data": [3.0, 7.0]} ]
            }
          ]
        }
    ],
    "observations": [
        { "name": "singlechannel", "data": [51.0, 48.0] }
    ],
    "measurements": [
        { "name": "Measurement", "config": {"poi": "mu", "parameters": []} }
    ],
    "version": "1.0.0"
}
EOF
```




## Advantages of JSON:

- ubiquitous, human-/machine-readable ASCII, patchable (see Backup)

# HistFactory JSON

## Generating and Reading HistFactory JSON is easy in ROOT & Python

### Writing JSON

pyhf

```
data = [...]
model = pyhf.Model(...)
ws = pyhf.workspace.Workspace.build(model, data)
json.dump(ws,'workspace.json')
```

ROOT

```
$> root workspace.root
root[0] Measurement->PrintXML()
$> pyhf xml2json Measurement.xml
```

### Reading JSON

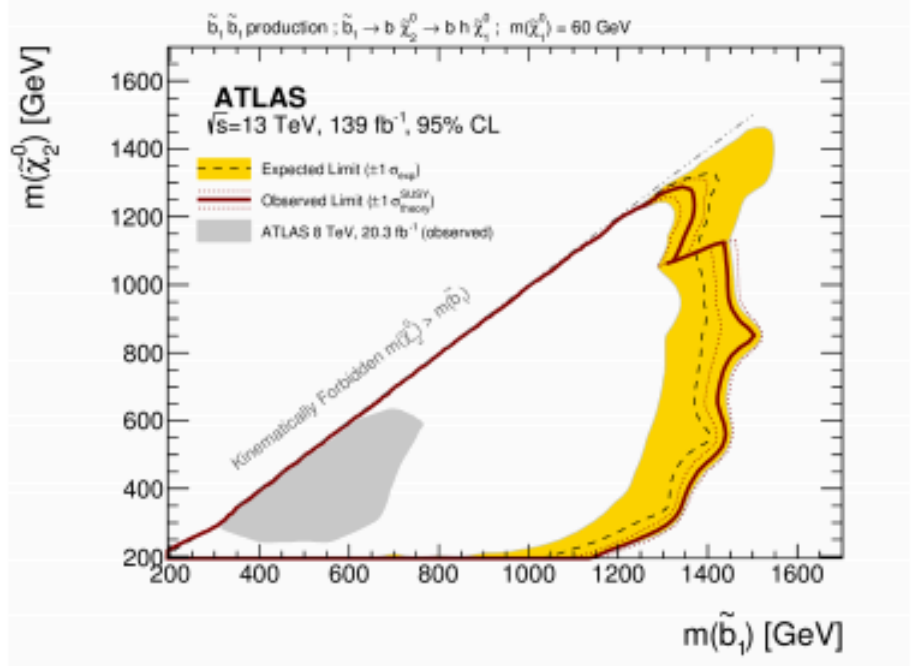
```
import pyhf
pyhf.Workspace(json.load('workspace.json'))
```

```
$> pyhf json2xml workspace.json
$> hist2workspace Measurement.xml
```

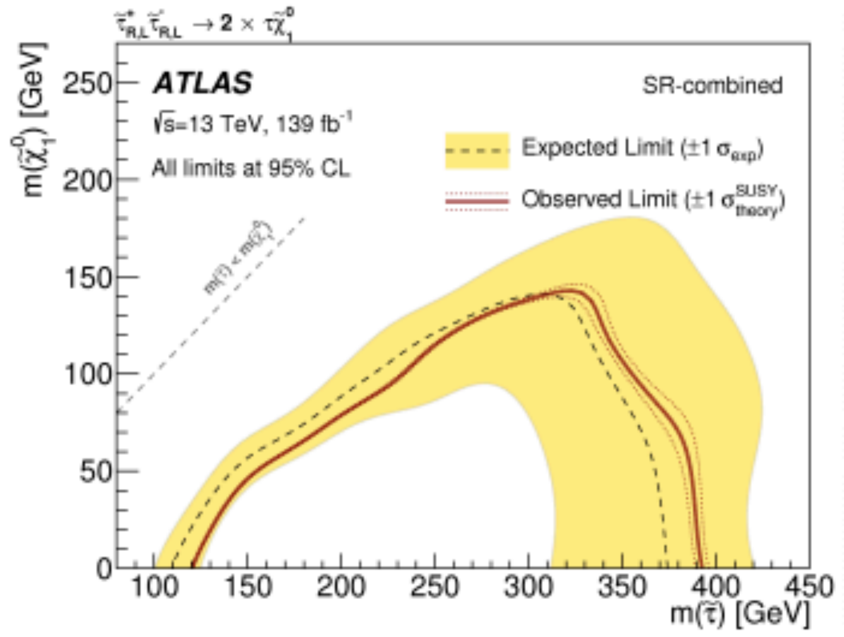
# Publishing Models

## ATLAS is already publishing HistFactory JSON on HepData

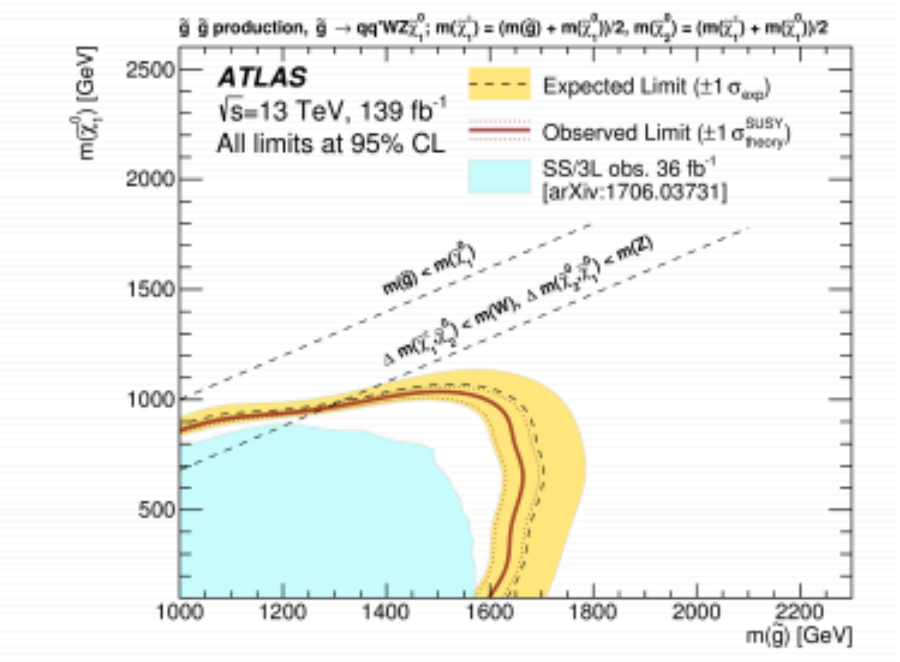
DOI 10.17182/hepdata.89408.v2



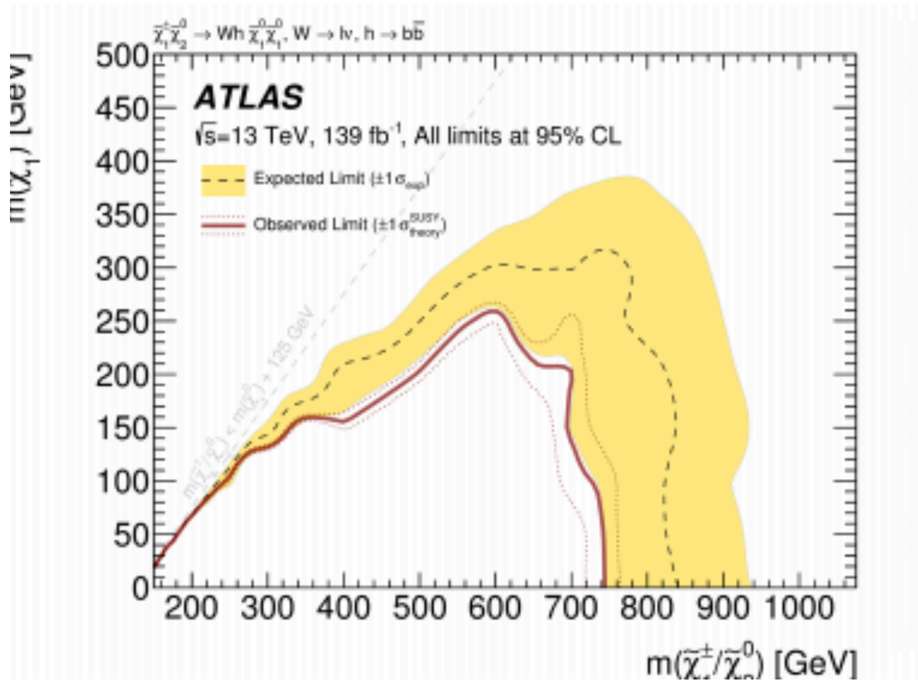

DOI 10.17182/hepdata.92006.v2



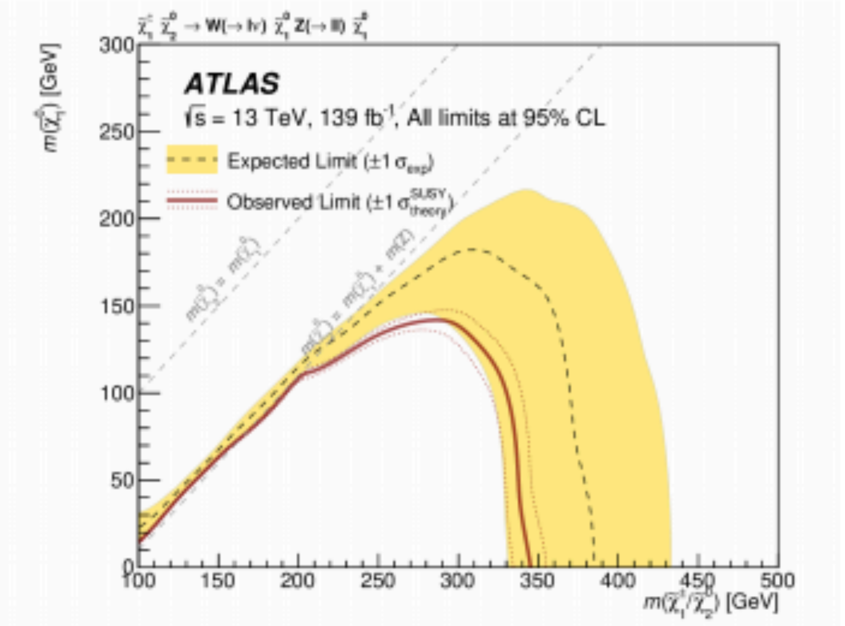

DOI 10.17182/hepdata.91214.v3



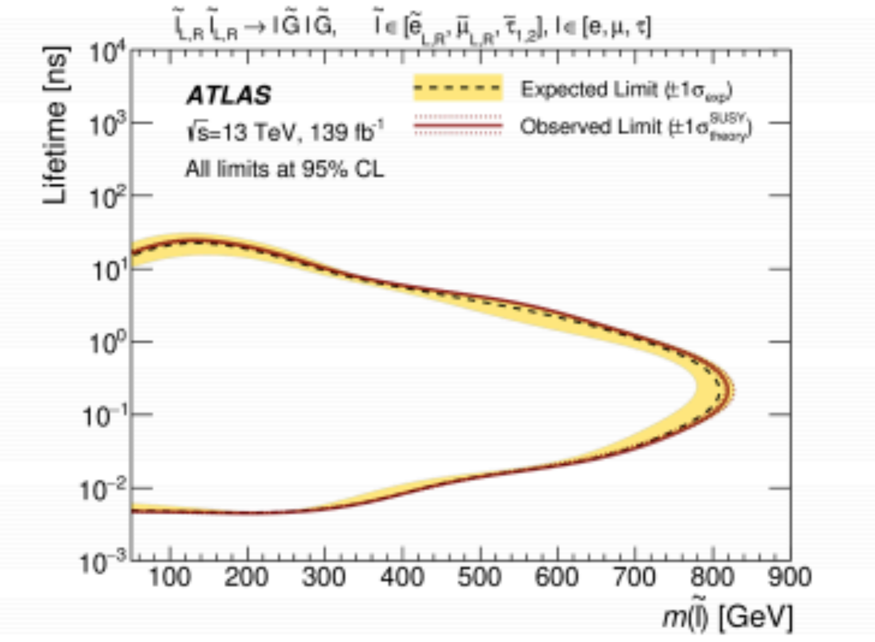

DOI 10.17182/hepdata.90607.v3




DOI 10.17182/hepdata.91127.v2




DOI 10.17182/hepdata.98796.v2




A SModelS interface for pyhf likelihoods

Gaël Alguero[a], Sabine Kraml[a], Wolfgang Waltenberger[b,c]

[a] *Laboratoire de Physique Subatomique et de Cosmologie, Université Grenoble-Alpes, CNRS/IN2P3, 53 Avenue des Martyrs, F-38026 Grenoble, France*
[b] *Institut für Hochenergiephysik, Österreichische Akademie der Wissenschaften, Nikolsdorfer Gasse 18, 1050 Wien, Austria*
[c] *University of Vienna, Faculty of Physics, Boltzmanngasse 5, A-1090 Wien, Austria*

The new version, SModelS v1.2.4, is publicly available from `https://smodels.github.io/` and can readily be employed for physics studies. We congratulate ATLAS to the important move of making full likelihood information available in digital format and are looking forward to including more such data in future updates of SModelS.

This completes the work started in contribution 15 of [9] for SModelS; the MadAnalysis 5 interface to pyhf should become available in the upcoming MadAnalysis 5 v1.9 release.

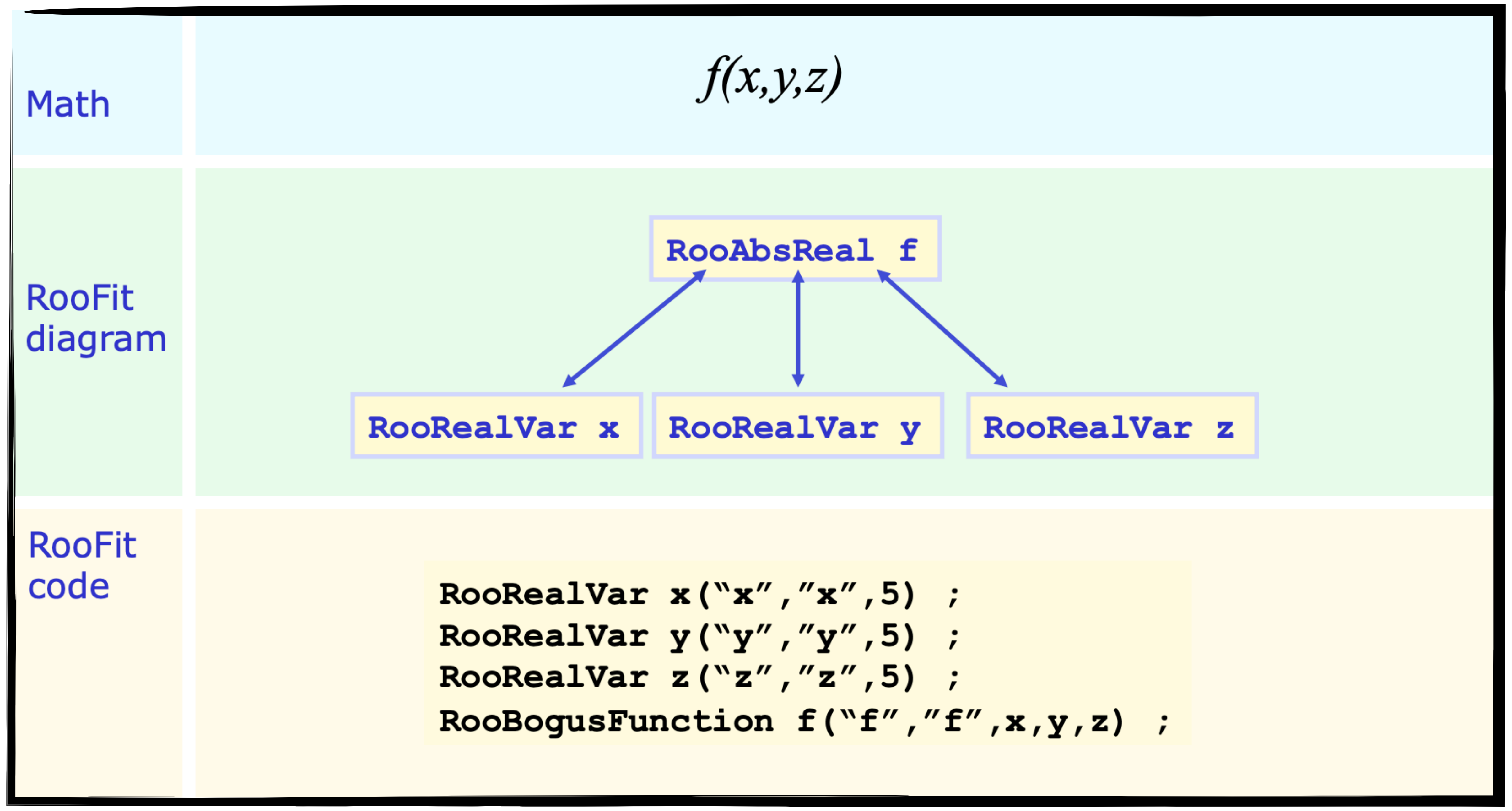
Last but not least we note that the technical discussions with the pyhf team are handled via github's issue tracking system, see e.g. `https://github.com/scikit-hep/pyhf/issues/620`, and are thus transparent and open to all.

## ... and it's being reused immediatly by theorists

# RooFit JSON

## Generalizing from pyhf: Can try to do something similar to RooFit?

- much more "open-world": more freedom, but  lower-level description of the intended model. More difficult to keep implementation-agnostic
- works for binned & unbinned models
- building blocks are: PDF types, connectors



# An early look at RooFitJSON

## New Feature in ROOT:

```python
ws = ROOT.RooWorkspace("workspace")
tool = ROOT.RooJSONFactoryWSTool(ws)
tool.importJSON('workspace.json')
```

```python
ws = ROOT.RooWorkspace("workspace")
tool = ROOT.RooJSONFactoryWSTool(ws)
tool.exportJSON('workspace.json')
```


```
"pdfs": {
    "background": {
        "mass": "mes",
        "power": "0.5",
        "resonance": "5.291",
        "slope": "argpar",
        "type": "ARGUS"
    },
    "model": {
        "coefficients": [
            "nsig",
            "nbkg"
        ],
        "dict": {
            "ModelConfig": "ModelConfig"
        },
        "summands": [
            "signal",
            "background"
        ],
        "tags": [
            "toplevel"
        ],
        "type": "pdfsum"
    },
    "signal": {
        "mean": "sigmean",
        "sigma": "sigwidth",
        "type": "Gaussian",
        "x": "mes"
    }
},
```

```
"variables": {
    "argpar": {
        "max": -1.0,
        "min": -100.0,
        "value": -20.0
    },
    "mes": {
        "max": 5.3,
        "min": 5.2,
        "value": 5.25
    },
    "nbkg": {
        "max": 10000.0,
        "min": 0.0,
        "value": 800.0
    },
    "nsig": {
        "max": 10000.0,
        "min": 0.0,
        "value": 200.0
    },
    "sigmean": {
        "max": 5.3,
        "min": 5.2,
        "value": 5.28
    },
    "sigwidth": {
        "max": 1.0,
        "min": 0.001,
        "value": 0.0027
    }
}
```


# Implemented Building Blocks

## As in pyhf: a subset of the "open world" is supported:

**Support so far for some of the most common PDFs. Available in ROOT 6.26**

**Roundtrip Workspace ROOT <> JSON is a design goal!**

**So far only a ROOT implementation, (maybe a independent one is possible?)**

**This is a new development for ROOT Testers are very welcome! More Info: [Link]**

- Importers exist for
  - RooBinSamplingPdf
  - RooBinWidthFunction
  - RooFormulaVar
  - RooGenericPdf
  - RooRealSumPdf
  - RooHistFunc
  - PiecewiseInterpolation
  - RooProdPdf
  - RooAddPdf
  - RooSimultaneous
  - RooRealSumPdf
- ImportExpressions exist for
  - RooGaussian
  - RooExponential
  - RooPoisson
  - RooProduct
  - FlexibleInterpVar
  - RooAddition
  - ParamHistFunc
  - RooArgusBG
- Exporters exist for
  - RooBinWidthFunction
  - RooProdPdf
  - RooProdPdf
  - RooSimultaneous
  - RooBinSamplingPdf
  - RooHistFunc
  - RooGenericPdf
  - RooFormulaVar
  - RooRealSumPdf
  - FlexibleInterpVar
  - PiecewiseInterpolation
- ExportExpressions exist for
  - RooGaussian
  - RooPoisson
  - RooExponential
  - RooProduct
  - RooProdPdf
  - ParamHistFunc
  - RooAddPdf
  - RooAddition
  - RooArgusBG

# Summary

$$p(\text{theory}|\text{data}) = \frac{\boxed{p(\text{data}|\text{theory})}}{p(\text{data})} p(\text{theory})$$

**Publishing stat. models is an obviously good thing to do.**

- enables a lot of new physics, key especially for "big science"

**But it didn't happen for 20 years:**

- Now we have a few good options:
  pyhf (for HistFactory), HS3 for RooFit models

**Now we have new momentum across the field**

- If you need help get in touch!