Statistics and Data Analysis (HEP at LHC)

Modeling tools: RooFit, RooStats & HistFactory

Slides extracted from W. Verkerke lectures at SOS School 2018, France

Prof. dr hab. Elżbieta Richter-Wąs

Coding probability models and likelihood functions

- Implementation of systematic uncertainties in likelihood models typically leads to very complex probability models
- All statistical techniques discussed in Section 2,4 require numeric minimization of likelihood functions. See problem in three parts
  1. Construction of probability models and likelihood functions (always needed)
  2. Minimization of likelihood functions (for parameter estimation, variance estimate, likelihood-ratio intervals)
  3. Construction of test statistics and calculation of their distributions, construction of Neyman constructions on test statistics (p-values, confidence intervals) and calculation (MC(MC)) integrals over Likelihood (Bayesian credible intervals, Bayes factors)
- For step 2 (minimization) the HEP industry standard is MINUIT
- For steps 1, 3 good tools have been developed in the past years, will discuss these now.

RooFit, RooStats and HistFactory

Will cover RooFit and HistFactory in
a bit more detail since they relate
to model building – the key topic of this course

RooFit
Language for building
probability models

Comprises datasets,
likelihoods, minimization,
toy data generation,
visualization and persistence

W. Verkerke & D. Kirkby
(exists since 1999)

HistFactory

Language to simplify
construction of RooFit
models of a particular type:
binned likelihood
template (morphing) models

K. Cranmer, A. Shibata, G. Lewis,
L. Moneta, W. Verkerke
(exists since 2010)

Will briefly sketch
workings of RooStats

RooStats

Suite of statistical tests
operating on RooFit
probability models

K. Cranmer, G. Schott,
L. Moneta, W. Verkerke
(exists since 2008)

RooFit core design philosophy

- Mathematical objects are represented as C++ objects

Mathematical concept		RooFit class
variable	x	RooRealVar
function	$f(x)$	RooAbsReal
PDF	$f(x)$	RooAbsPdf
space point	$\vec{x}$	RooArgSet
integral	$\int_{x_{min}}^{x_{max}} f(x)dx$	RooRealIntegral
list of space points		RooAbsData

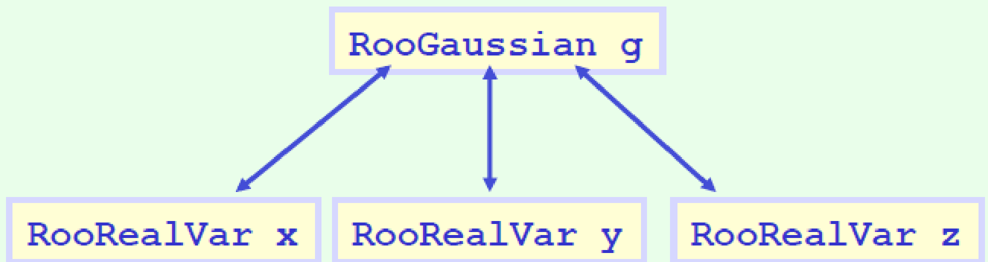
RooFit core design philosophy - Workspace

- The workspace serves a container class for all objects created

Math

$$\mathrm{Gauss}(x,\mu,\sigma)$$

RooFit diagram



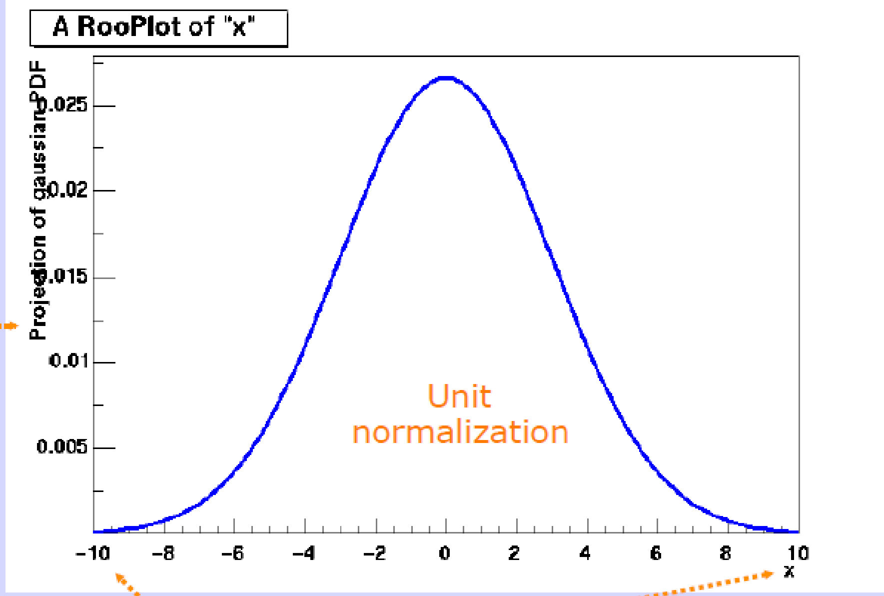
RooFit code

```
RooRealVar x("x","x",-10,10) ;
RooRealVar m("m","y",0,-10,10) ;
RooRealVar s("s","z",3,0.1,10) ;
RooGaussian g("g","g",x,m,s) ;
```

Basics – Creating and plotting a Gaussian p.d.f.

Setup gaussian PDF and plot

```
// Create an empty plot frame
RooPlot* xframe = w::x.frame() ;

// Plot model on frame
model.plotOn(xframe) ;

// Draw frame on canvas
xframe->Draw() ;
```



Axis label from gauss title

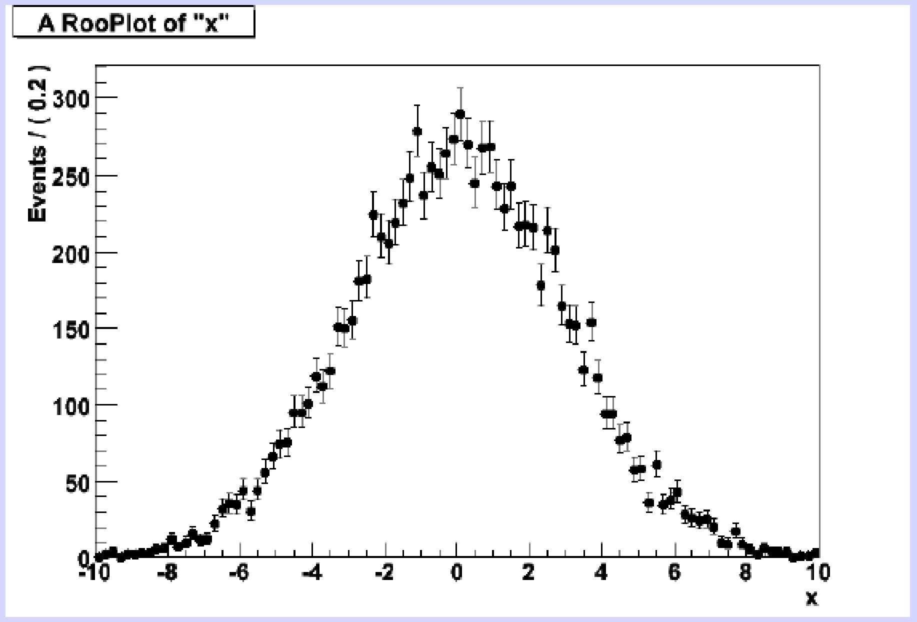
A RooPlot is an empty frame capable of holding anything plotted versus it variable

Plot range taken from limits of x

Basics – Generating toy MC events

Generate 10000 events from Gaussian p.d.f and show distribution

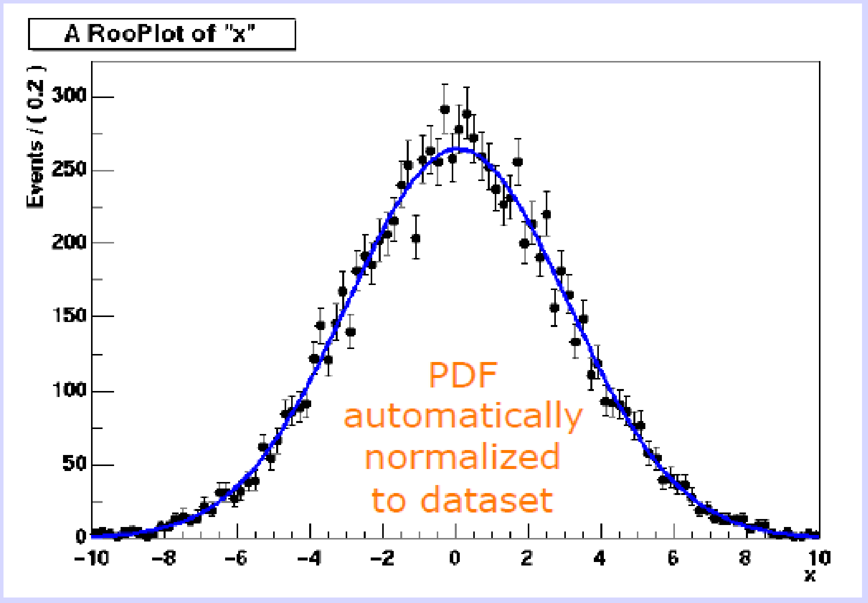
```
// Generate an unbinned toy MC set
RooDataSet* data = w::gauss.generate(w::x,10000) ;

// Generate an binned toy MC set
RooDataHist* data = w::gauss.generateBinned(w::x,10000) ;

// Plot PDF
RooPlot* xframe = w::x.frame()
data->plotOn(xframe) ;
xframe->Draw() ;
```



Can generate both binned and unbinned datasets

Basics – ML fit of p.d.f. to unbinned data

```
// ML fit of gauss to data
w::gauss.fitTo(*data) ;
(MINUIT printout omitted)

// Parameters if gauss now
// reflect fitted values
w::mean.Print()
RooRealVar::mean = 0.0172335 +/- 0.0299542
w::sigma.Print()
RooRealVar::sigma = 2.98094  +/- 0.0217306

// Plot fitted PDF and toy data overlaid
RooPlot* xframe = w::x.frame() ;
data->plotOn(xframe) ;
w::gauss.plotOn(xframe) ;
```



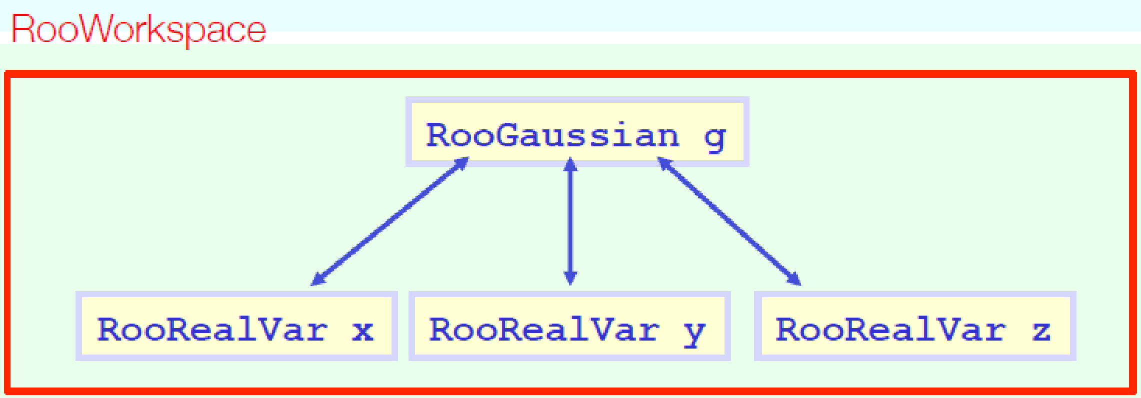
RooFit code design philosophy - Workspace

- The workspace serves a container class for all objects created

Math

$$\mathrm{Gauss}(x,\mu,\sigma)$$

RooFit diagram

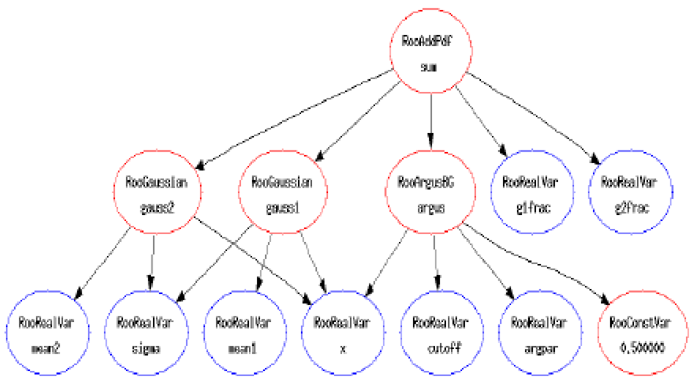


RooFit code

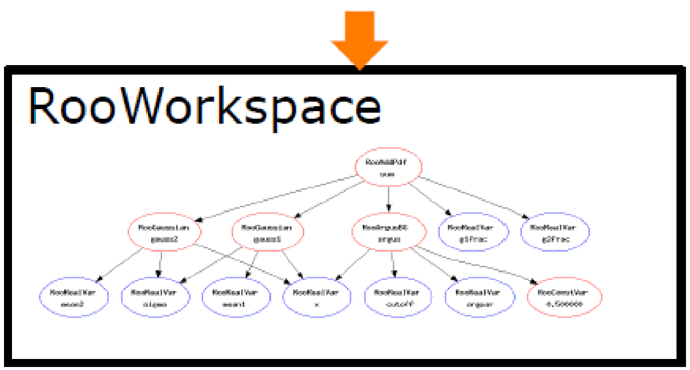
```
RooRealVar x("x","x",-10,10) ;
RooRealVar m("m","y",0,-10,10) ;
RooRealVar s("s","z",3,0.1,10) ;
RooGaussian g("g","g",x,m,s) ;
RooWorkspace w("w") ;
w.import(g) ;
```

The workspace

- The workspace concept has revolutionized the way people share and combine analysis
  - Completely factorizes process of building and using likelihood functions
  - You can give somebody an analytical likelihood of a (potentially very complex) physics analysis in a way to the easy-to-use, provides introspection, and is easy to modify.

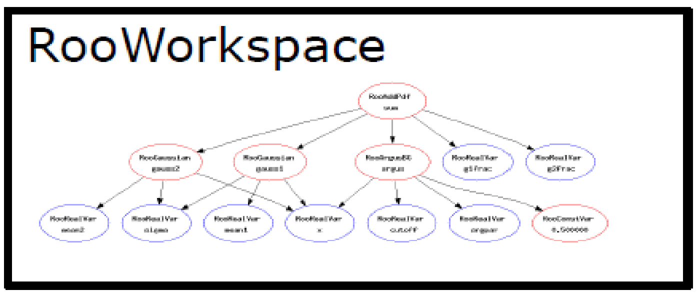


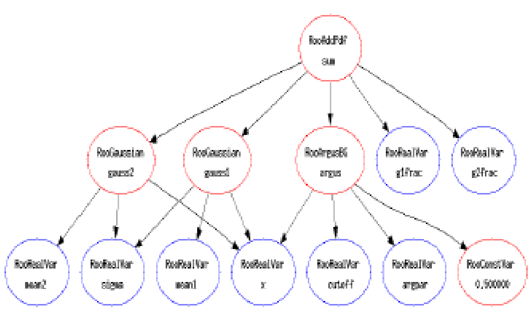

```
RooWorkspace w("w") ;
w.import(sum) ;
w.writeToFile("model.root") ;
```



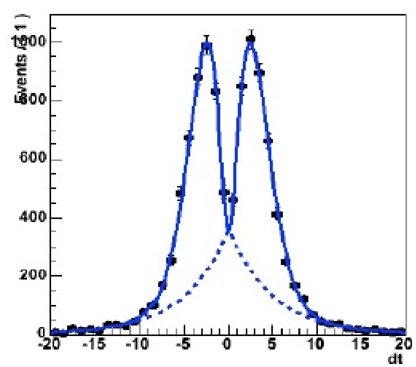
model.root

Using a workspace






```
// Resurrect model and data
TFile f("model.root") ;
RooWorkspace* w = f.Get("w") ;
RooAbsPdf* model = w->pdf("sum") ;
RooAbsData* data = w->data("xxx") ;

// Use model and data
model->fitTo(*data) ;

RooPlot* frame =
         w->var("dt")->frame() ;
data->plotOn(frame) ;
model->plotOn(frame) ;
```



Accessing a workspace contents

- Looking into a workspace

```
w.Print() ;

variables
---------
(mean,sigma,x)

p.d.f.s
-------
RooGaussian::f[ x=x mean=mean sigma=sigma ] = 0.249352
```

- Access components two ways

```
// 1 - Standard accessor method
RooAbsPdf* pdf = w->pdf("f") ;

// 2 - Import contents into C++ namespace in interpreter
w.exportToCint("w") ;
RooPlot* frame = w::x.frame() ;
w::f.plotOn(frame) ;
// strongly typed: w::f is 'RooGaussian&'
```

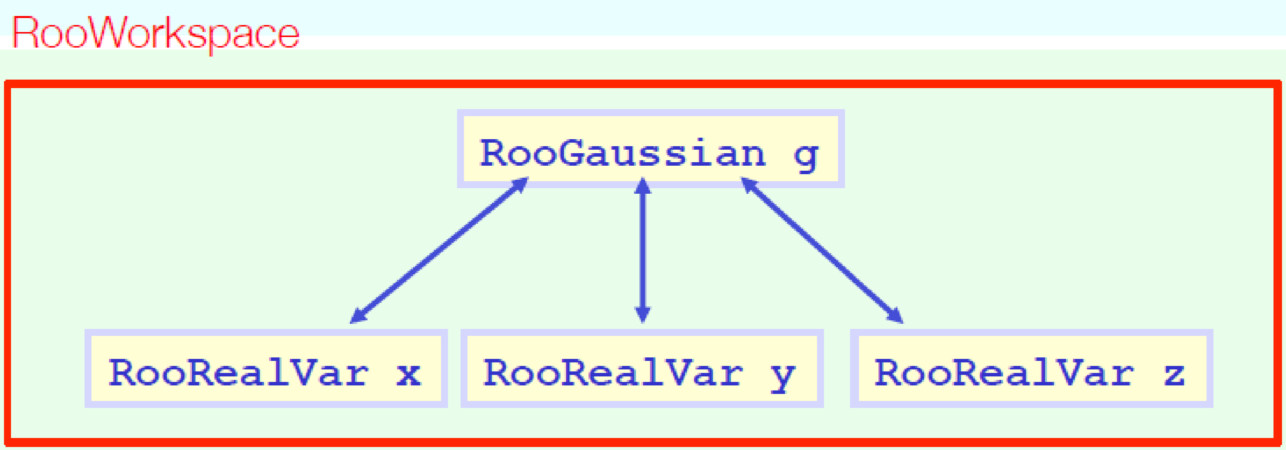
RooFit core design philosophy - Workspace

- The workspace serves a container class for all objects created

Math

$$\mathrm{Gauss}(x,\mu,\sigma)$$

RooFit diagram



RooFit code

```
RooRealVar x(“x”,”x”,-10,10) ;
RooRealVar m(“m”,”y”,0,-10,10) ;
RooRealVar s(“s”,”z”,3,0.1,10) ;
RooGaussian g(“g”,”g”,x,m,s) ;
RooWorkspace w(“w”) ;
w.import(g) ;
```

Factory and Workspace

- *One C++ object per math symbol* provides ultimate level of control over each objects functionality, but results in lengthy user code for even simple macros
- Solution: add factory that auto-generates objects from a math-like language. Accessed through factory() method of workspace
- Example: reduce construction of Gaussian pdf and its parameters from 4 to 1 line of code

```
RooRealVar x(“x”,”x”,-10,10) ;
RooRealVar mean(“mean”,”mean”,5) ;
RooRealVar sigma(“sigma”,”sigma”,3)  ;
RooGaussian f(“f”,”f”,x,mean,sigma) ;
w.import(f) ;
```

```
w.factory(“Gaussian::f(x[-10,10],mean[5],sigma[3])”) ;
```

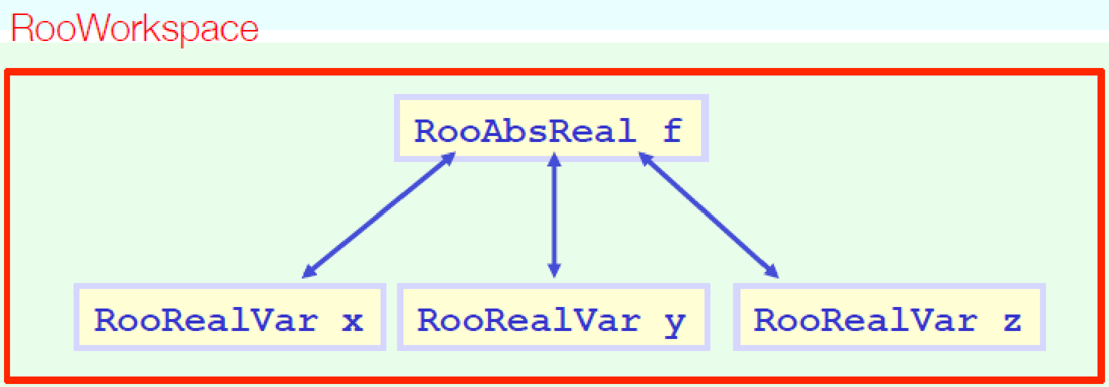
Populating a workspace the easy way – „the factory"

- The factory allows to fill a workspace with pdfs and variables using a simplified scripting language

Math

$$\text{Gauss}(x,\mu,\sigma)$$

RooFit diagram

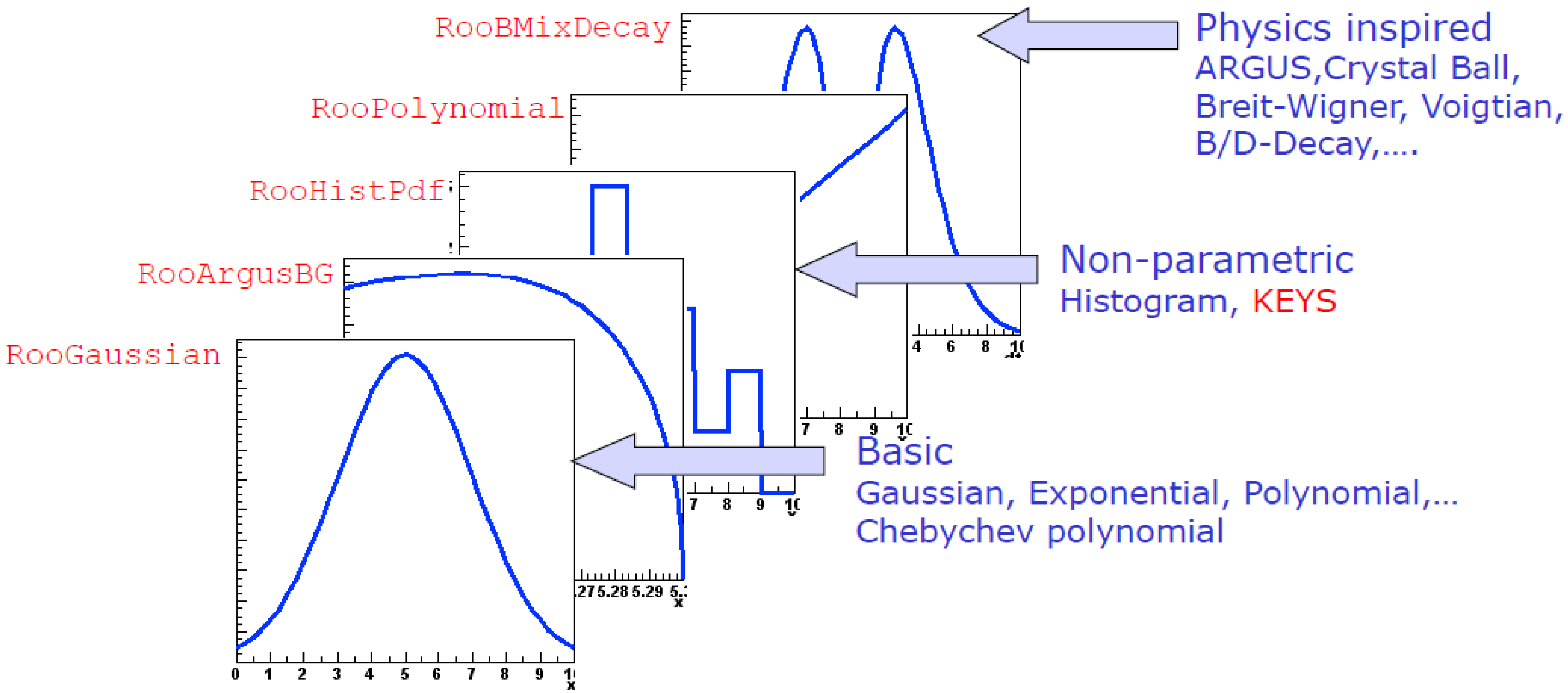


RooFit code

```
RooWorkspace w("w") ;
w.factory("RooGaussian::g(x[-10,10],m[-10,10],z[3,0.1,10])");
```

Model building – (Re)using standard components

- RooFit provides a collection of compiled standard PDF classes



Easy to extend the library: each p.d.f. is a separate C++ class

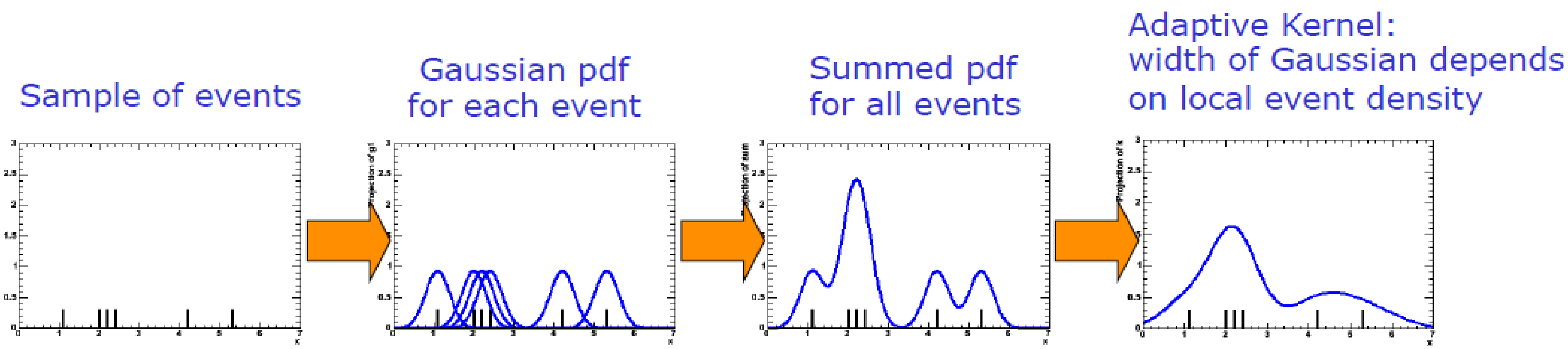
Model building – (Re)using standard components

- List of most frequently used pdfs and their factory spec

Gaussian	`Gaussian::g(x,mean,sigma)`
Breit-Wigner	`BreitWigner::bw(x,mean,gamma)`
Landau	`Landau::l(x,mean,sigma)`
Exponential	`Exponental::e(x,alpha)`
Polynomial	`Polynomial::p(x,{a0,a1,a2})`
Chebychev	`Chebychev::p(x,{a0,a1,a2})`
Kernel Estimation	`KeysPdf::k(x,dataSet)`
Poisson	`Poisson::p(x,mu)`
Voigtian (=BW$\otimes$G)	`Voigtian::v(x,mean,gamma,sigma)`

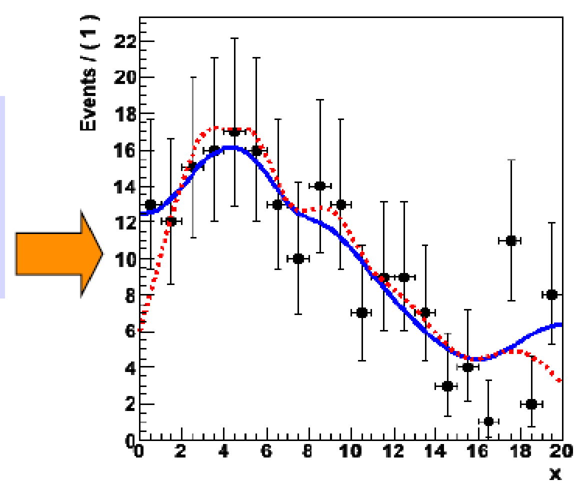
The power of pdf as building blocks – Advanced algorithms

- Example: a ‘kernel estimation probability model’
  - Construct smooth pdf from unbinned data, using kernel estimation technique



- Example

```
w.import(myData,Rename("myData")) ;
w.factory("KeysPdf::k(x,myData)") ;
```



- Also available for n-D data

The power of pdf as building blocks – adaptability

- RooFit pdf classes do not require their parameter arguments to be variables, one can plug in functions as well
- Allows trivial customization, extension of probability models

class RooGaussian

$$Gauss(x \mid \mu, \sigma)$$

also class RooGaussian!

$$Gauss(x \mid \underbrace{\mu \cdot (1+2\alpha)}, \sigma)$$

Introduce a response function for a systematic uncertainty

```
// Original Gaussian
w.factory("Gaussian::g1(x[80,100],m[91,80,100],s[1])")


// Gaussian with response model in mean
w.factory("expr::m_response("m*(1+2alpha)",m,alpha[-5,5])") ;
w.factory("Gaussian::g1(x,m_response,s[1])")
```

NB: “expr” operates builds an intepreted function expression on the fly

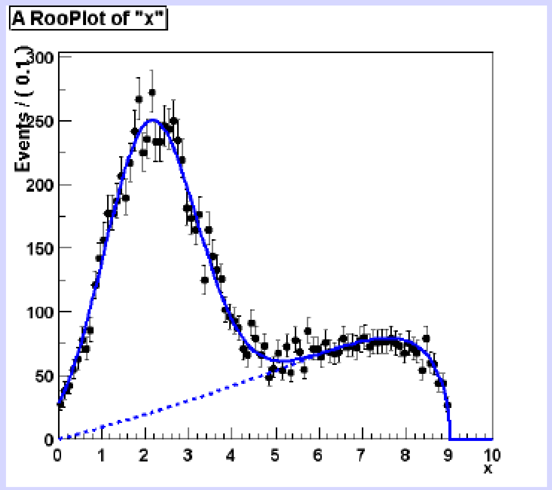
The power of pdf as building blocks – operator expressions

- Create a SUM expression to represent a sum of probability models

```
w.factory("Gaussian::gauss1(x[0,10],mean1[2],sigma[1]") ;
w.factory("Gaussian::gauss2(x,mean2[3],sigma)") ;
w.factory("ArgusBG::argus(x,k[-1],9.0)") ;

w.factory("SUM::sum(g1frac[0.5]*gauss1, g2frac[0.1]*gauss2, argus)")
```

- In composite model visualization components can be accessed by name

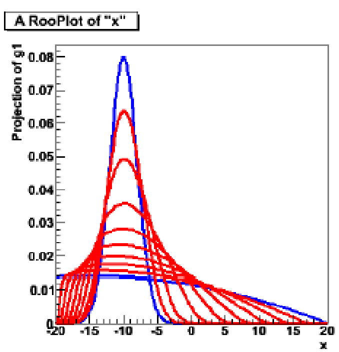
```
// Plot only argus components
w::sum.plotOn(frame,Components("argus"),
              LineStyle(kDashed)) ;
```

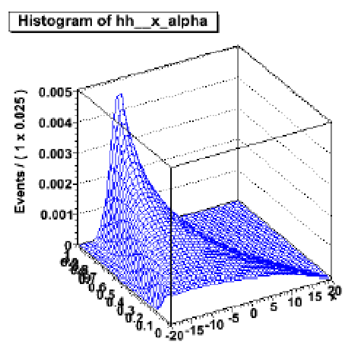


Powerful operators – Morphing interpolation

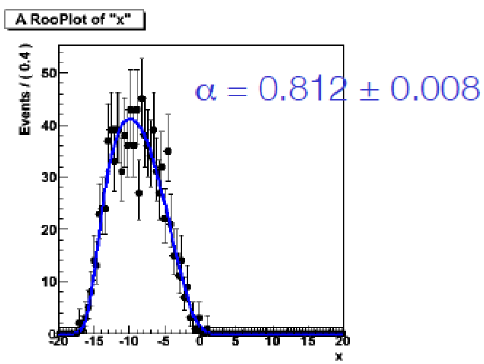
- Special operator pdfs can interpolate existing pdf shapes
  - Ex: interpolation between Gaussian and Polynomial

```
w.factory("Gaussian::g(x[-20,20],-10,2)") ;
w.factory("Polynomial::p(x,{-0.03,-0.001})") ;
w.factory("IntegralMorph::gp(g,p,x,alpha[0,1])") ;
```





Fit to data



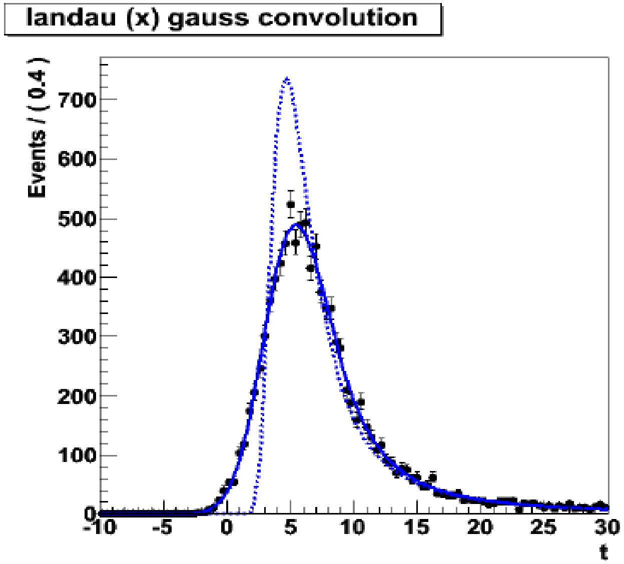
- Three morphing operator classes available
  - IntegralMorph (Alex Read).
  - MomentMorph (Max Baak).
  - PiecewiseInterpolation (via HistFactory)

Powerful operators – Fourier convolution

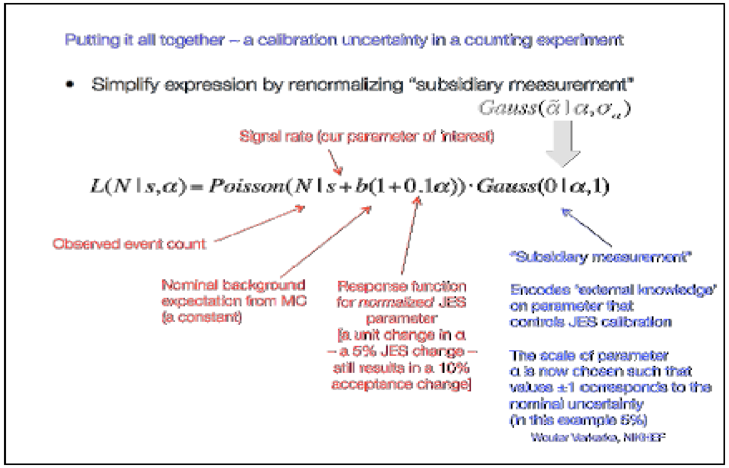
- Convolve any two arbitrary pdfs with a 1-line expression

```
w.factory("Landau::L(x[-10,30],5,1)") :
w.factory("Gaussian::G(x,0,2)") ;

w::x.setBins("cache",10000) ; // FFT sampling density
w.factory("FCONV::LGf(x,L,G)") ; // FFT convolution
```

- Exploits power of FFTW package available via ROOT
  - Hand-tuned assembler code for time-critical parts
  - Amazingly fast: unbinned ML fit to 10.000 events take ~5 seconds!



Example 1: counting expt



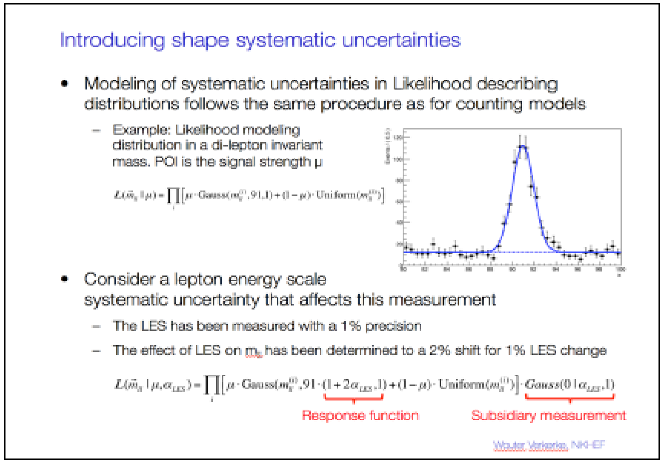
- Will now demonstrate how to construct a model for a counting experiment with a systematic uncertainty

$$L(N \mid s,\alpha) = Poisson(N \mid s + b(1+0.1\alpha)) \cdot Gauss(0 \mid \alpha, 1)$$

```
// Subsidiary measurement of alpha
w.faxtory("Gaussian::subs(0,alpha[-5,5],1)") ;

// Response function mu(alpha)
w.factory("expr::mu('s+b(1+0.1*alpha)',s[20],b[20],alpha)") ;

// Main measurement
w.factory("Poisson::p(N[0,10000],mu)");

// Complete model Physics*Subsidiary
w.factory("PROD::model(p,subs)") ;
```

Example 2: unbinned L with syst.

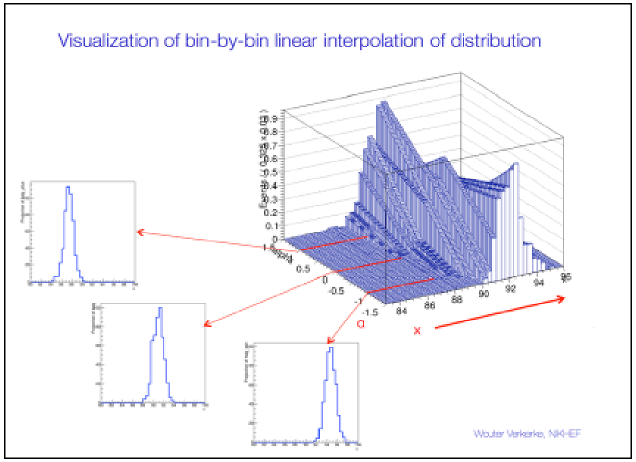
- Will now demonstrate how to code complete example of the unbinned profile likelihood of Section 5:



$$L(\vec{m}_{ll} \mid \mu, \alpha_{LES}) = \prod_i \left[\mu \cdot \text{Gauss}(m_{ll}^{(i)}, 91 \cdot (1 + 2\alpha_{LES}), 1) + (1-\mu) \cdot \text{Uniform}(m_{ll}^{(i)}) \right] \cdot Gauss(0 \mid \alpha_{LES}, 1)$$

```
// Subsidiary measurement of alpha
w.factory("Gaussian::subs(0,alpha[-5,5],1)");

// Response function m(alpha)
w.factory("expr::m_a("m*(1+2alpha)",m[91,80,100],alpha)") ;

// Signal model
w.factory("Gaussian::sig(x[80,100],m_a,s[1])")

// Complete model Physics(signal plus background)*Subsidiary
w.factory("PROD::model(SUM(mu[0,1]*sig,Uniform::bkg(x)),subs)") ;
```

Example 3: binned L with syst.



- Example of template morphing systematic in a binned likelihood

$$s_i(\alpha,\ldots) = \begin{cases} s_i^0 + \alpha\cdot(s_i^+ - s_i^0) & \forall\alpha > 0 \\ s_i^0 + \alpha\cdot(s_i^0 - s_i^-) & \forall\alpha < 0 \end{cases}$$

$$L(\vec{N}\,|\,\alpha,\vec{s}^-,\vec{s}^0,\vec{s}^+) = \prod_{bins} P(N_i\,|\,\underbrace{s_i(\alpha,s_i^-,s_i^0,s_i^+)}_{})\cdot\underbrace{G(0\,|\,\alpha,1)}_{}$$

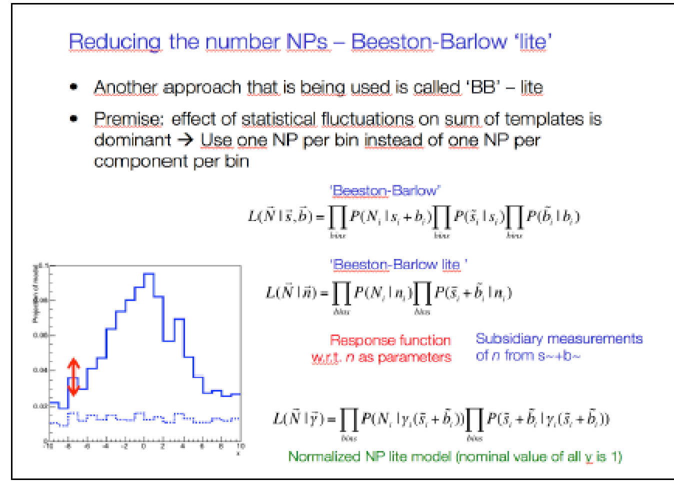
```
// Import template histograms in workspace
 w.import(hs_0,hs_p,hs_m) ;

 // Construct template models from histograms
 w.factory("HistFunc::s_0(x[80,100],hs_0)") ;
 w.factory("HistFunc::s_p(x,hs_p)") ;
 w.factory("HistFunc::s_m(x,hs_m)") ;

 // Construct morphing model
 w.factory("PiecewiseInterpolation::sig(s_0,s_,m,s_p,alpha[-5,5])") ;

 // Construct full model
 w.factory("PROD::model(ASUM(sig,bkg,f[0,1]),Gaussian(0,alpha,1))") ;
```

Example 4: Beeston-Barlow light

Example 4 – Beeston-Barlow light

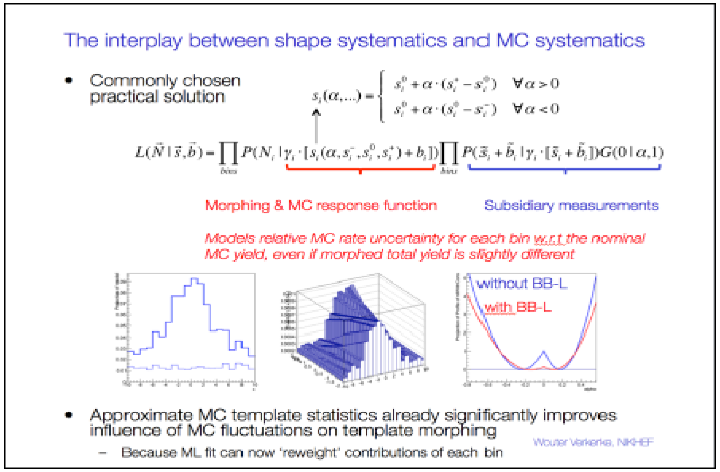
- Beeston-Barlow-(lite) modeling of MC statistical uncertainties

$$L(\vec{N}\,|\,\vec{\gamma}) = \prod_{bins} P(N_i \,|\, \gamma_i(\tilde{s}_i + \tilde{b}_i)) \prod_{bins} P(\tilde{s}_i + \tilde{b}_i \,|\, \gamma_i(\tilde{s}_i + \tilde{b}_i))$$




```
// Import template histogram in workspace
 w.import(hs) ;

// Construct parametric template models from histograms
// implicitly creates vector of gamma parameters
 w.factory("ParamHistFunc::s(hs)") ;

 // Product of subsidiary measurement
 w.factory("HistConstraint::subs(s)") ;

 // Construct full model
 w.factory("PROD::model(s,subs)") ;
```

Example 5: BB-lite and morphing



- Template morphing model with Beeston-Barlow-lite MC statistical uncertainties

$$s_i(\alpha,\ldots) = \begin{cases} s_i^0 + \alpha\cdot(s_i^+ - s_i^0) & \forall\alpha > 0 \\ s_i^0 + \alpha\cdot(s_i^0 - s_i^-) & \forall\alpha < 0 \end{cases}$$

$$L(\vec{N}\,|\,\vec{s},\vec{b}) = \prod_{bins} P(N_i\,|\,\gamma_i\cdot[s_i(\alpha,s_i^-,s_i^0,s_i^+)+b_i])\prod_{bins} P(\tilde{s}_i+\tilde{b}_i\,|\,\gamma_i\cdot[\tilde{s}_i+\tilde{b}_i])G(0\,|\,\alpha,1)$$

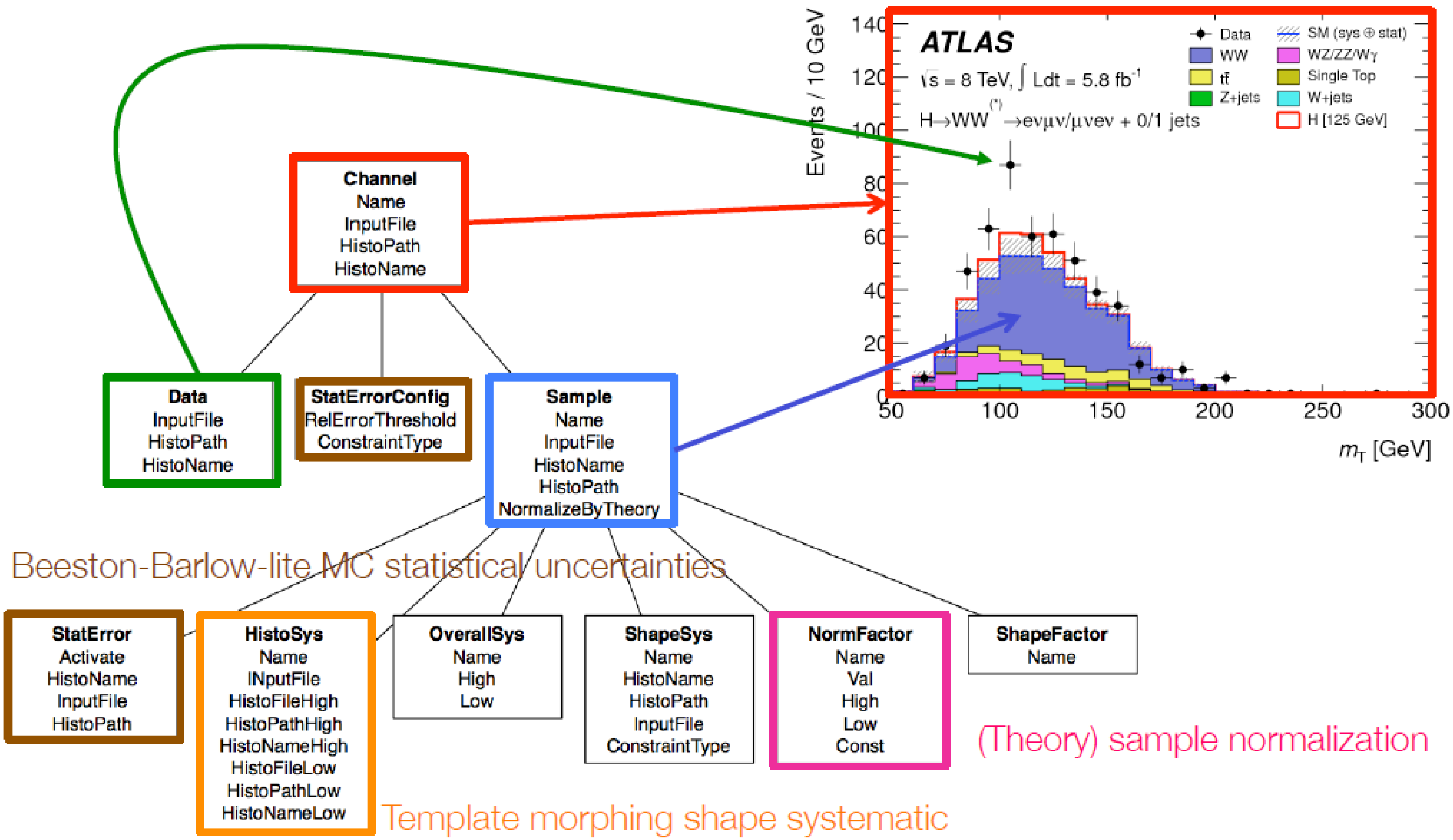
```
// Import template histograms in workspace
w.import(hs_0,hs_p,hs_m,hb) ;

// Construct parametric template morphing signal model
w.factory("ParamHistFunc::s_p(hs_p)") ;
w.factory("HistFunc::s_m(x,hs_m)") ;
w.factory("HistFunc::s_0(x[80,100],hs_0)") ;
w.factory("PiecewiseInterpolation::sig(s_0,s_,m,s_p,alpha[-5,5])") ;

// Construct parametric background model (sharing gamma's with s_p)
w.factory("ParamHistFunc::bkg(hb,s_p)") ;

// Construct full model with BB-lite MC stats modeling
w.factory("PROD::model(ASUM(sig,bkg,f[0,1]),
           HistConstraint({s_0,bkg}),Gaussian(0,alpha,1))") ;
```

HistFactory – structured building of binned template models

- RooFit modeling building blocks allow to easily construct likelihood models that model shape and rate systematics with one or more nuisance parameter
  - Only few lines of code per construction
- Typical LHC analysis required modeling of 10-50 systematic uncertainties in O(10) samples in anywhere between 2 and 100 channels → Need structured formalism to piece together model from specifications. This is the purpose of HistFactory
- HistFactory conceptually similar to workspace factory, but has much higher level semantics
  - Elements represent physics concepts (channels, samples, uncertainties and their relation) rather than mathematical concepts
  - Descriptive elements are represented by C++ objects (like roofit), and can be configured in C++, or alternively from an XML file
  - Builds a RooFit (mathematical) model from a HistFactory physics model.

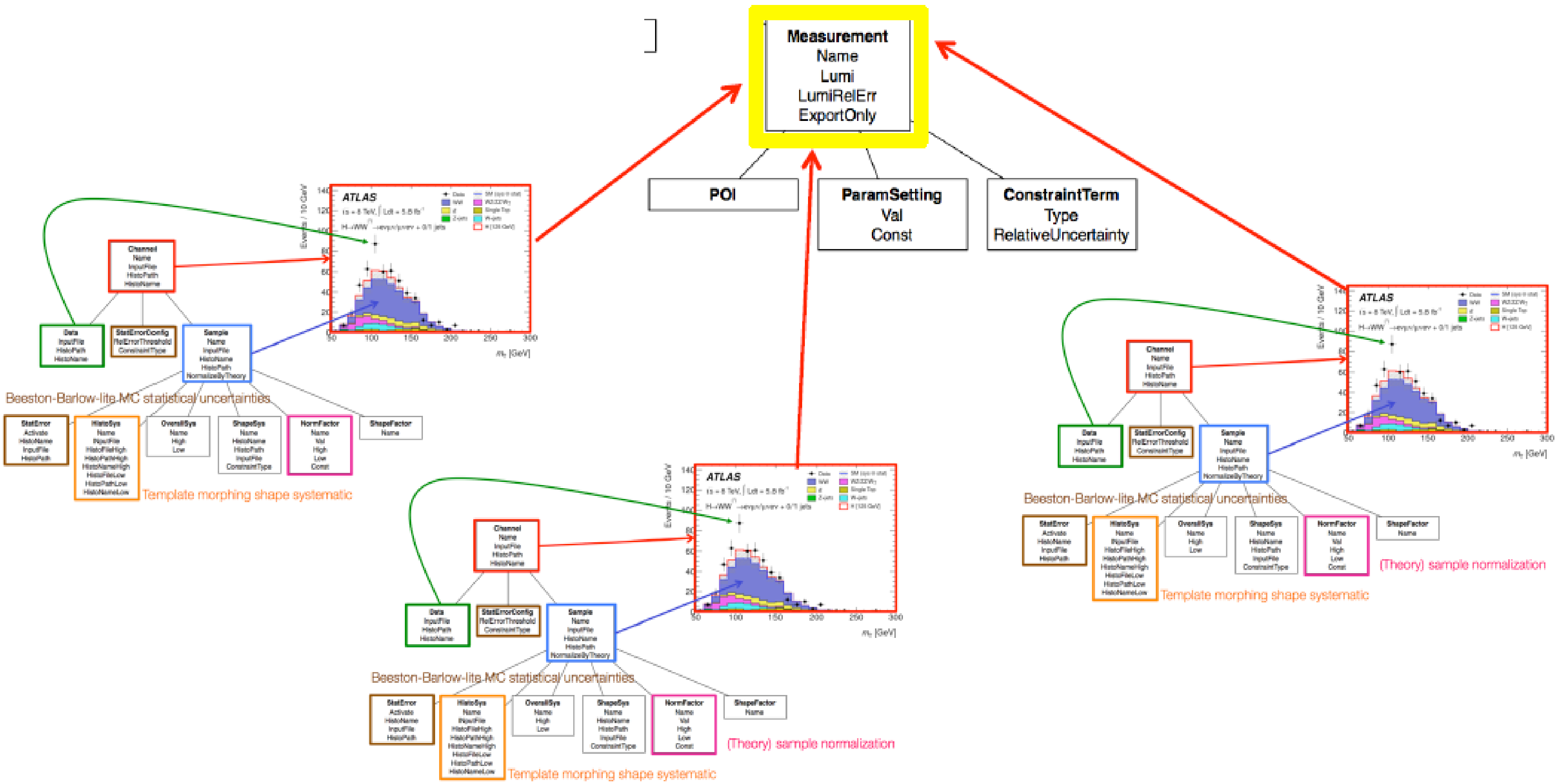
HistFactory elements of a channel

- Hierarchy of concepts for description of one measurement channel



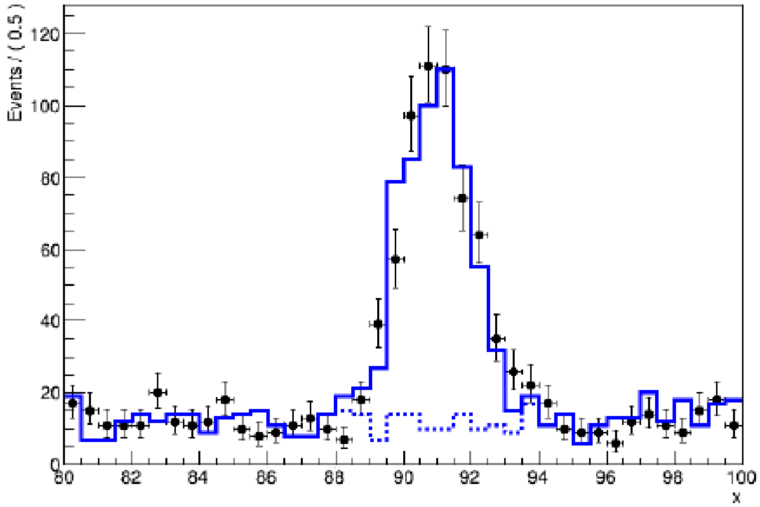
HistFactory elements of measurement

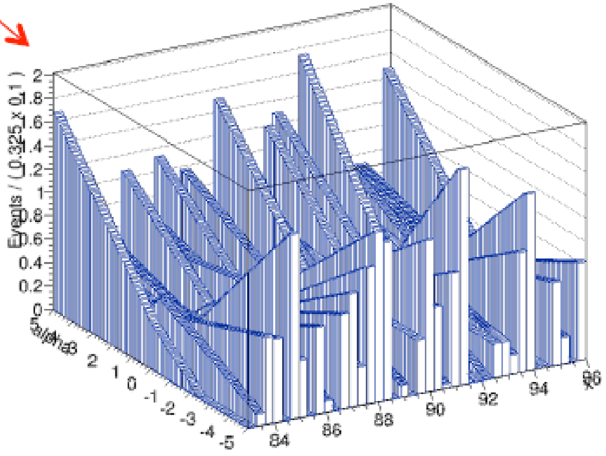
- One or more channels are combined to form a measurement
  - Along with some extra information (declaration of the POI, the luminosity of the data sample and its uncertainty)



Example of model building with HistFactory

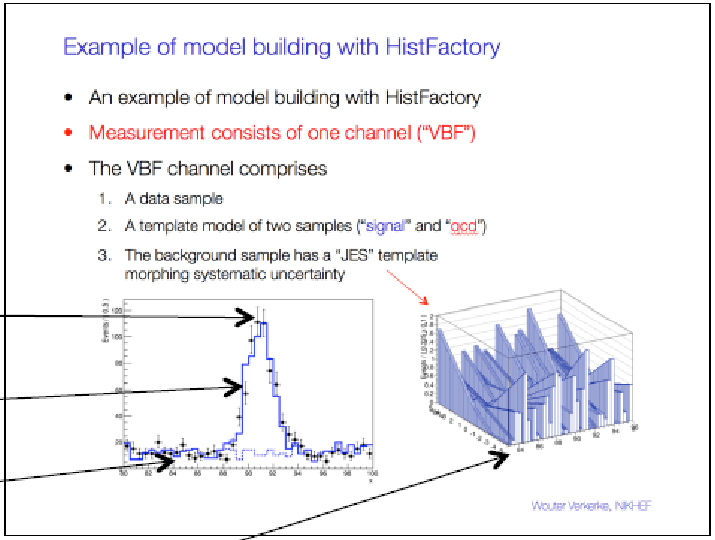
- An example of model building with HistFactory
- Measurement consists of one channel ("VBF")
- The VBF channel comprises
  1. A data sample
  2. A template model of two samples ("signal" and "qcd")
  3. The background sample has a "JES" template morphing systematic uncertainty





Example of model building with HistFactory

```
// external input in form of TH1 shown in green

// Declare ingredients of measurement
HistFactory::Data data ;
data.SetHisto(data_hist) ;

HistFactory::Sample signal("signal") ;
signal.SetHisto(sample_hist) ;

HistFactory::Sample qcd("QCD") ;
qcd.SetHisto(sample_hist) ;

HistFactory::HistoSys hsys("QCD_JetEnergyScale") ;
hsys.SetHistoLow(sample_hist_sysdn) ;
hsys.SetHistoHigh(sample_hist_sysup) ;
qcd.AddHistoSys(hsys) ;

HistFactory::Channel channel("VBF") ;
channel.SetData(data) ;
channel.AddSample(sample) ;

HistFactory::Measurement meas("MyAnalysis") ;
meas.AddChannel(channel) ;

// Now build RooFit model according to specs
HistFactory::HistoToWorkspaceFactoryFast h2w(meas) ;
RooWorkspace* w = h2w.MakeCombinedModel(meas) ;
w->Print("t") ;
w->writeToFile("test.root") ;
```



HistFactory model output

- Contents of RooFit workspace produced by HistFactory

RooFit probability model as specified

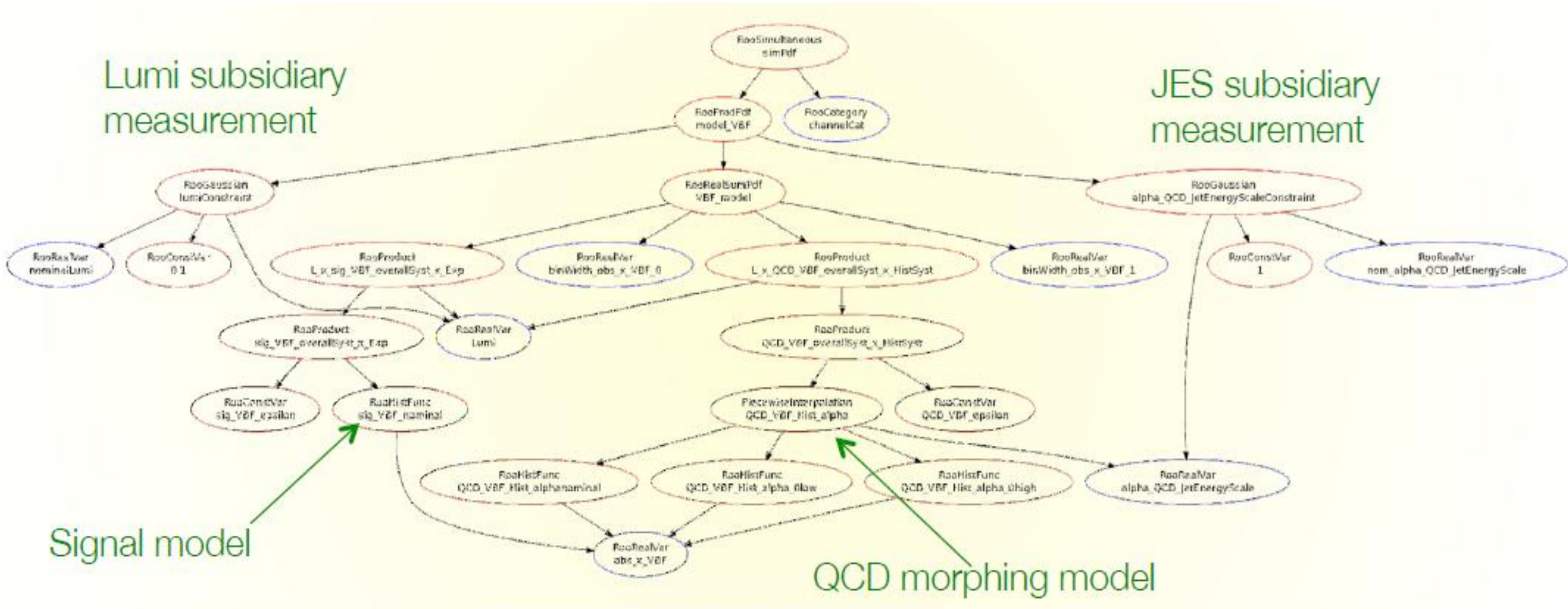
Definition of POI, NPs, Observables Global observables

Universal Model Configuration

```
RooWorkspace(combined) combined contents

variables
---------
(Lumi,alpha_QCD_JetEnergyScale,binWidth_obs_x_VBF_0,binWidth_obs_x_VBF_1,channelCat,
 nom_alpha_QCD_JetEnergyScale,nominalLumi,obs_x_VBF,weightVar)

p.d.f.s
-------
RooSimultaneous::simPdf[ indexCat=channelCat VBF=model_VBF ] = 0
  RooProdPdf::model_VBF[ lumiConstraint * alpha_QCD_JetEnergyScaleConstraint * VBF_model(obs_x_VBF) ] = 0
    RooGaussian::lumiConstraint[ x=Lumi mean=nominalLumi sigma=0.1 ] = 1
    RooGaussian::alpha_QCD_JESConstraint[ x=alpha_QCD_JetEnergyScale mean=nom_alpha_QCD_JetEnergyScale sigma=1 ] = 1
    RooRealSumPdf::VBF_model[ binW_obs_x_VBF_0 * L_x_sig_VBF_overallSyst_x_Exp + binW_obs_x_VBF_1 * L_x_QCD_VBF_overallSyst_x_HistSyst ] = 0
      RooProduct::L_x_sig_VBF_overallSyst_x_Exp[ Lumi * sig_VBF_overallSyst_x_Exp ] = 0
        RooProduct::sig_VBF_overallSyst_x_Exp[ sig_VBF_nominal * sig_VBF_epsilon ] = 0
          RooHistFunc::sig_VBF_nominal[ depList=(obs_x_VBF) ] = 0
      RooProduct::L_x_QCD_VBF_overallSyst_x_HistSyst[ Lumi * QCD_VBF_overallSyst_x_HistSyst ] = 0
        RooProduct::QCD_VBF_overallSyst_x_HistSyst[ QCD_VBF_Hist_alpha * QCD_VBF_epsilon ] = 0
          PiecewiseInterpolation::QCD_VBF_Hist_alpha[ ] = 0
            RooHistFunc::QCD_VBF_Hist_alphanominal[ depList=(obs_x_VBF) ] = 0
            RooHistFunc::QCD_VBF_Hist_alpha_0low[ depList=(obs_x_VBF) ] = 0
            RooHistFunc::QCD_VBF_Hist_alpha_0high[ depList=(obs_x_VBF) ] = 0

datasets
--------
RooDataSet::asimovData(obs_x_VBF,weightVar,channelCat)
RooDataSet::obsData(channelCat,obs_x_VBF)

embedded datasets (in pdfs and functions)
-----------------------------------------
RooDataHist::sig_VBFnominalDHist(obs_x_VBF)
RooDataHist::QCD_VBF_Hist_alphanominalDHist(obs_x_VBF)
RooDataHist::QCD_VBF_Hist_alpha_0lowDHist(obs_x_VBF)
RooDataHist::QCD_VBF_Hist_alpha_0highDHist(obs_x_VBF)

parameter snapshots
-------------------
NominalParamValues = (nominalLumi=1[C],nom_alpha_QCD_JetEnergyScale=0[C],weightVar=0,obs_x_VBF=-4.5,Lumi=1,alpha_QCD_JetEnergyScale=0,
                      binWidth_obs_x_VBF_0=1[C],binWidth_obs_x_VBF_1=1[C])

named sets
----------
ModelConfig_GlobalObservables:(nominalLumi,nom_alpha_QCD_JetEnergyScale)
ModelConfig_Observables:(obs_x_VBF,weightVar,channelCat)
ModelConfig_POI:()
globalObservables:(nominalLumi,nom_alpha_QCD_JetEnergyScale)
observables:(obs_x_VBF,weightVar,channelCat)

generic objects
---------------
RooStats::ModelConfig::ModelConfig
```

HistFactory model structure

- RooFit object structure
  - As visalized with `simPdf::graphVizTree("model.dot")` followed by `dot -Tpng -omodel.png model.dot'`



- This RooFit probability model can be evaluated without knowledge of HistFactory
  - Additional (documentary) information stored in workspace specifies a uniquely specified statistical model (definition of POI, NP etc)