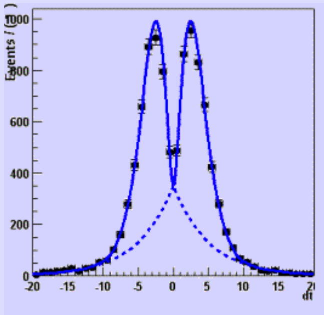
RooFit

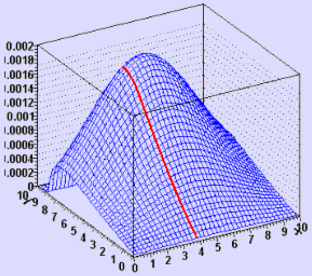
A tool kit for data modeling in ROOT
(W. Verkerke, D. Kirkby)

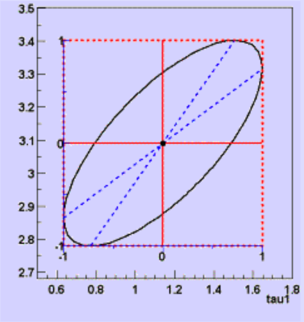
RooStats

A tool kit for statistical analysis
(K. Cranmer, L. Moneta, S. Kreiss, G. Kukartsev, G. Schott,
G. Petrucciani, W. Verkerke)

Wouter Verkerke (NIKHEF)





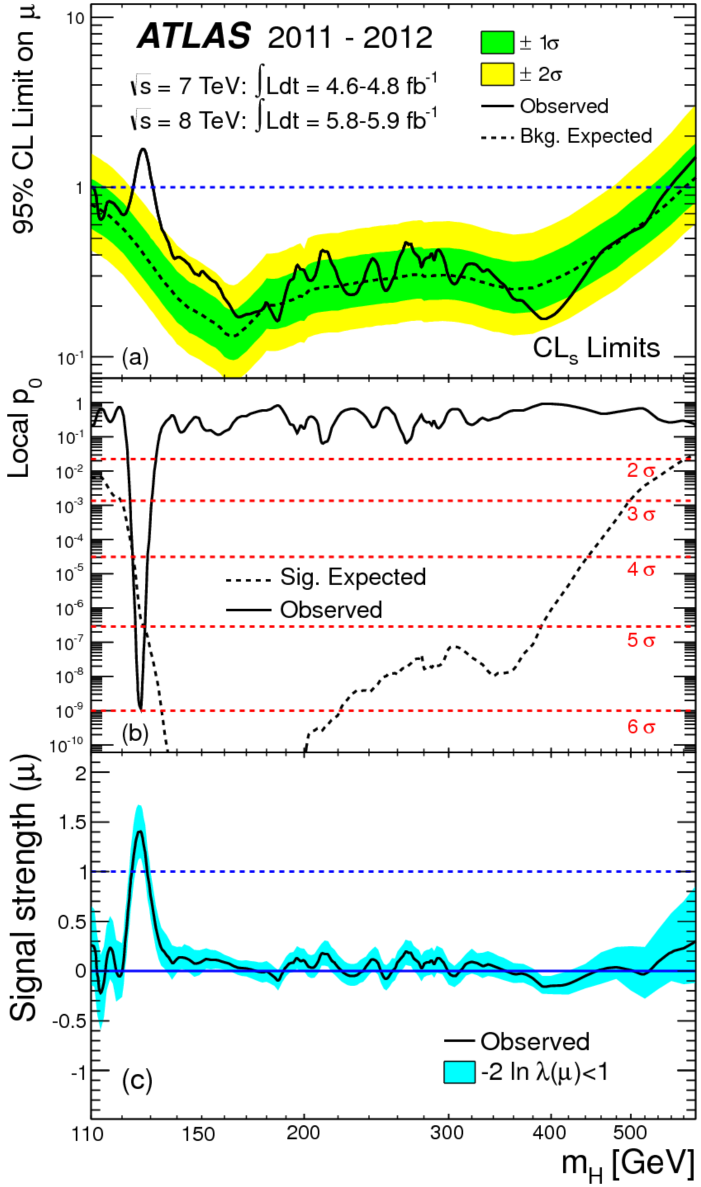


Introduction

- Statistical data analysis is at the heart of all (particle) physics experiments.

- Techniques deployed in HEP get more and more complicated
  → Hunting for ‘difficult signals’ (Higgs)
  → Desire to control systematic uncertainties through simultaneous fits to control measurements

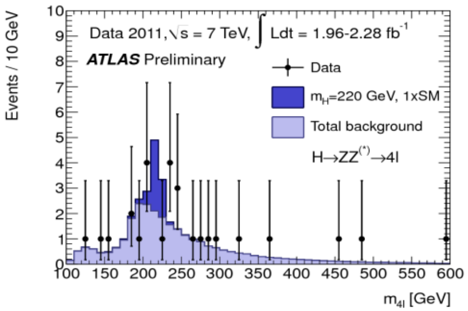
- Nowadays discoveries entail simultaneous modeling of hundreds of distributions with models with over a 1000 parameters → Well beyond ROOTs ‘TF1’ function classes



A structured approach to computational statistical analysis

- A structured approach is needed to be able to describe and use data models needed for modern HEP analyses
- **1 - Data modeling: construct a model f(x|θ)**

‘x_{obs}’ **‘f(x|θ)’** **→ L(θ)=f(x_{obs}|θ)**



$$f_{sig} \times \left[\begin{array}{l} \mathrm{SigSel}(m;\bar{p}_{sig}) \times \\ \left(\mathrm{SigDecay}(t;\vec{q}_{sig},\sin(2b)) \otimes \mathrm{SigResol}(t\,|\,dt;\vec{r}_{sig})\right) \end{array} \right] + (1-f_{sig}) \left[\begin{array}{l} \mathrm{BkgSel}(m;\bar{p}_{bkg}) \times \\ \left(\mathrm{BkgDecay}(t;\vec{q}_{bkg}) \otimes \mathrm{BkgResol}(t\,|\,dt;\vec{r}_{bkg})\right) \end{array} \right]$$

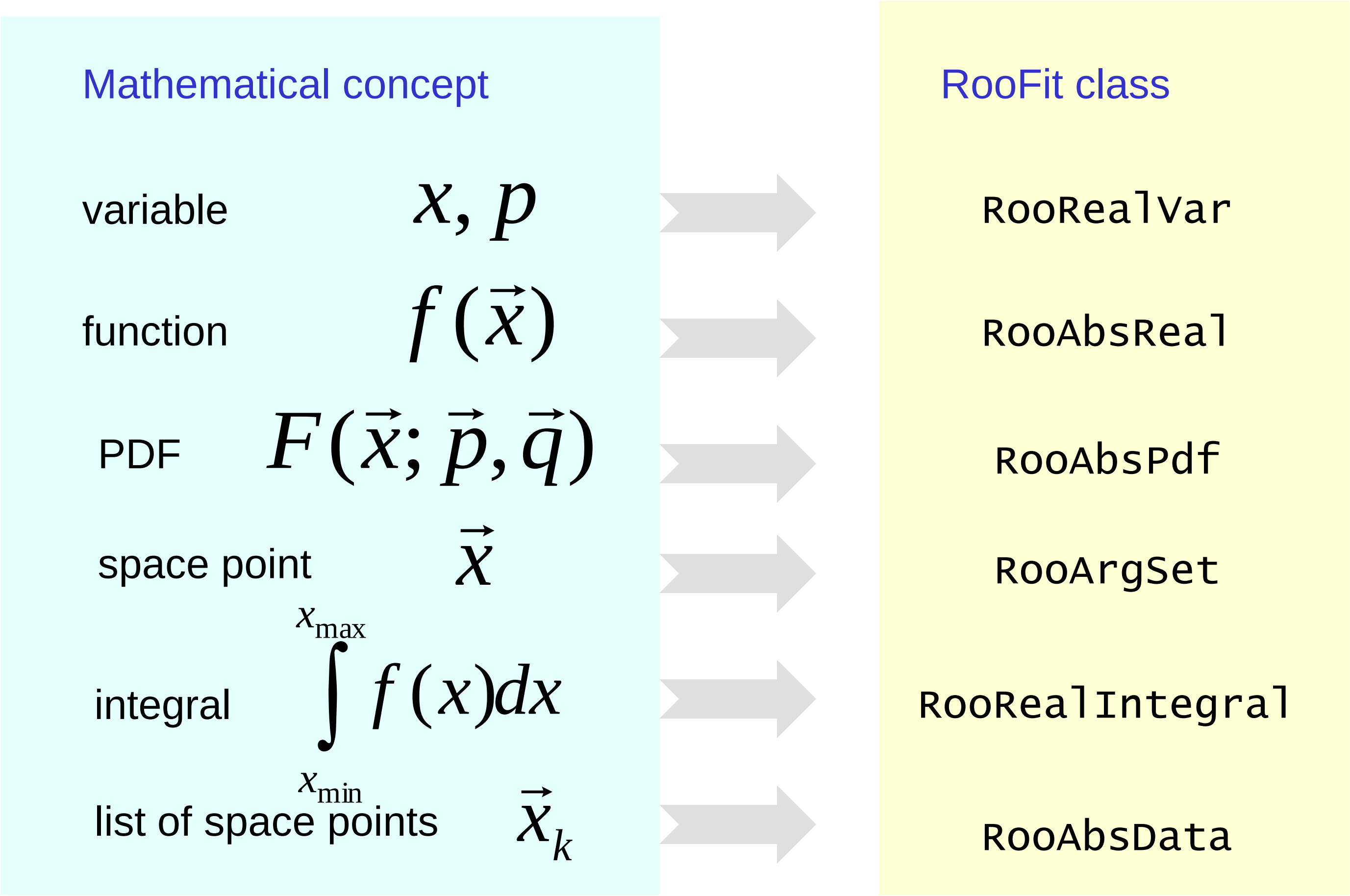
→ **RooFit (since 1999)**

→ RooFit::HistFactory (since 2010)

- **2 - Statistical inference on θ, given x_0 and f(x|θ)**
  - Parameter estimation ‘θ’ & variance estimation (V(θ)) → MINUIT
  - Confidence intervals: [$θ_-$, $θ_+$], θ<X at 95% C.L. hypothesis testing etc: → p(data|θ=0) = 1.10^{-7} → **RooStats (since 2007)**

RooFit – a toolkit to formulate probability models in C++

- Key concept: represent individual elements of a mathematical model by separate C++ objects



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

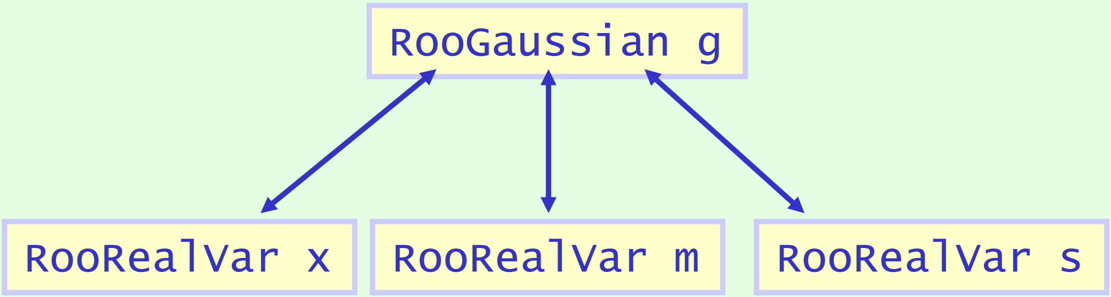
RooFit core design philosophy

- Functions objects are always ‘trees’ of objects, with pointers (managed through proxies) expressing relations

Math

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

RooFit diagram



RooGaussian g

RooRealVar x

RooRealVar m

RooRealVar s

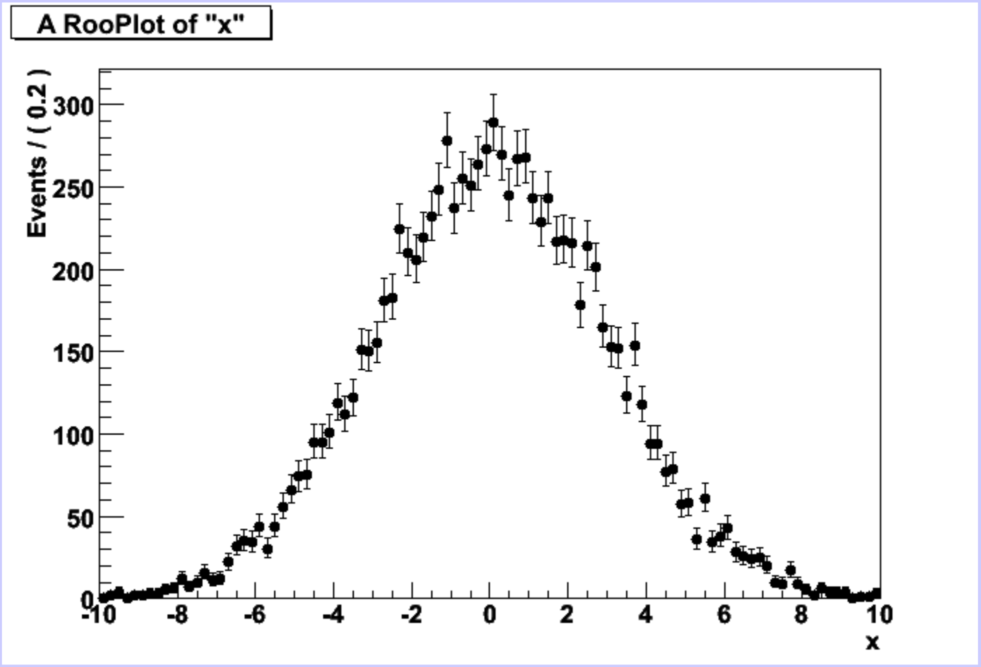
RooFit code

```
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) ;
```

RooFit: complete model functionality, e.g. sampling (un)binned data

Example: generate 10000 events from Gaussian p.d.f and show distribution

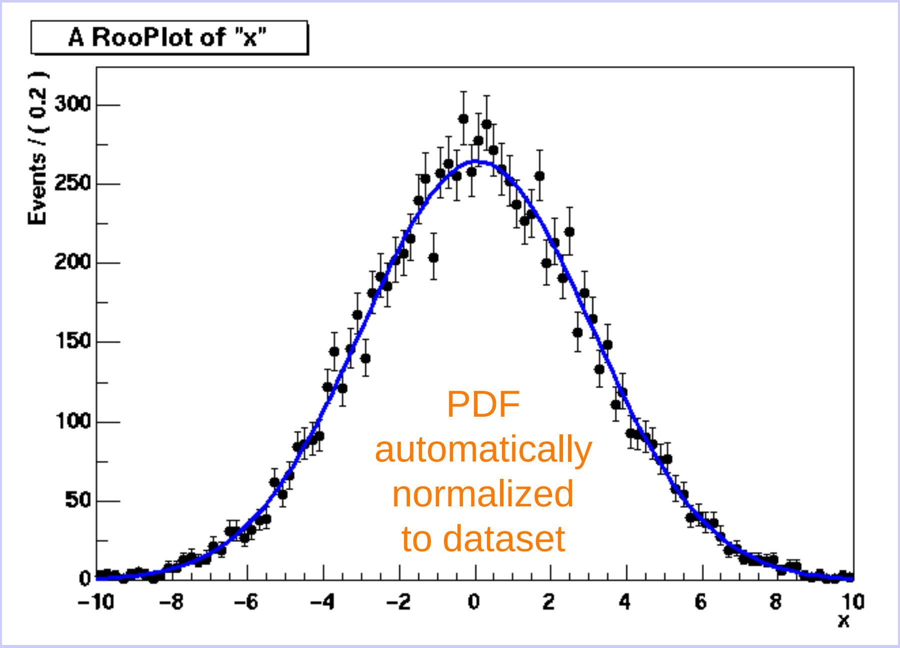
```
// Generate an unbinned toy MC set
RooDataSet* data = gauss.generate(x,10000) ;

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

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



Can generate both binned and unbinned datasets

RooFit model functionality – max.likelihood parameter estimation

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

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

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



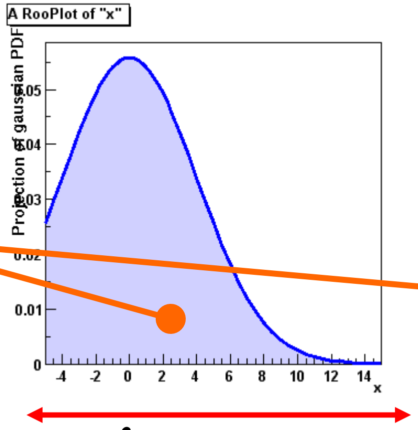
PDF automatically normalized to dataset

RooFit implements *normalized* probability models

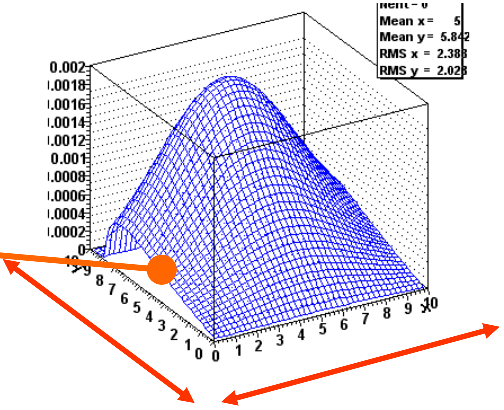
- Normalized probability (density) models are the basis of all fundamental statistical techniques
  - Defining feature:

$$\text{Ò} f(\vec{x},\vec{p})\,d\vec{x}\ \text{º}\ 1,$$

$$f(\vec{x},\vec{p})\ ^{3}\ 0$$



$$\int F(x)dx \equiv 1$$

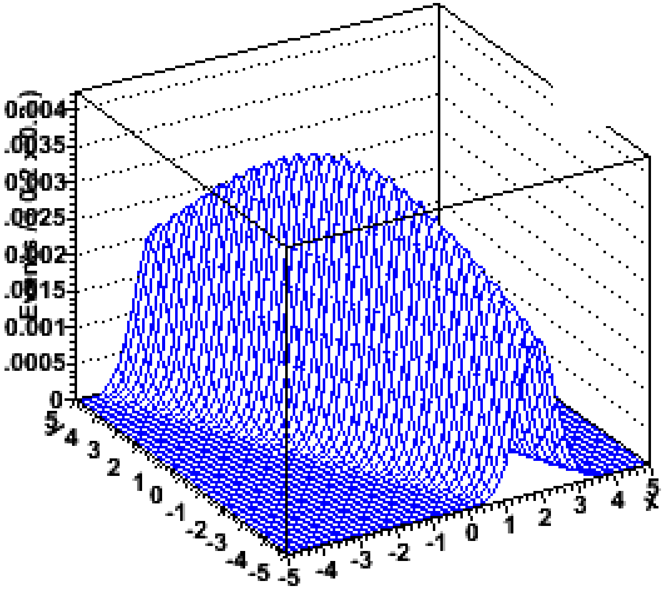


$$\int F(x,y)dxdy \equiv 1$$

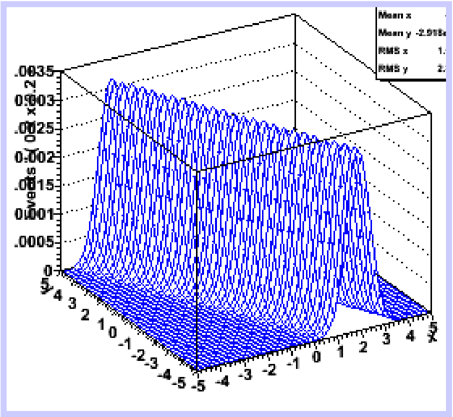
- Normalization guarantee introduces extra complication in calculation, but has important advantages
  - Directly usable in fundamental statistical techniques
  - Easier construction of complex models (will shows this in moment)
- RooFit provides built-in support for normalization, taking away down-side for users, leaving upside
  - Default normalization strategy relies on numeric techniques, but user can specify known (partial) analytical integrals in pdf classes.

The power of *conditional* probability modeling

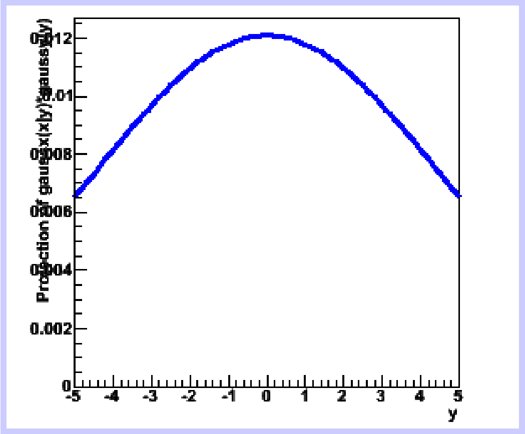
- Take following model f(x,y):
  what is the analytical form?



Gauss f(x|a*y+b,1)



Gauss g(y,0,3)



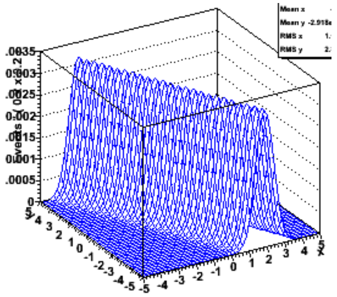
- Trivially constructed with (conditional) probability density functions!

F(x,y) = f(x|y)*g(y)

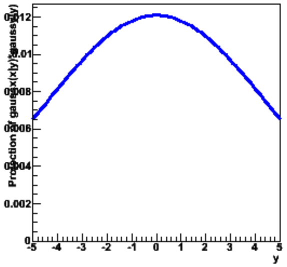
Coding a conditional product model in RooFit

- Construct each ingredient with a single line of code

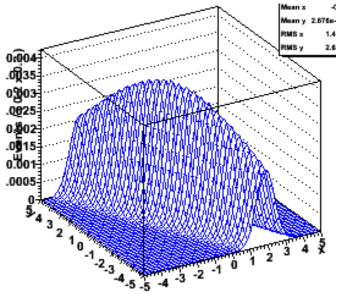
Gauss f(x,a*y+b,1)



Gauss g(y,0,3)



F(x,y) = f(x|y)*g(y)

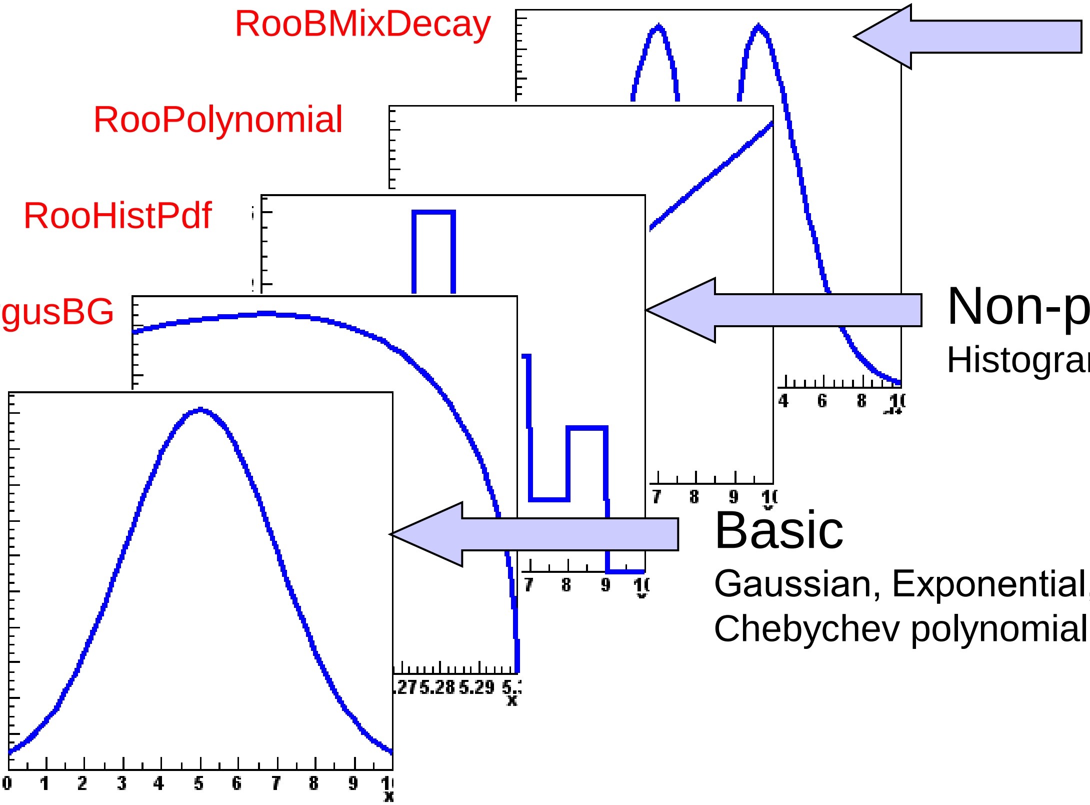


```
RooRealVar x(“x”,”x”,-10,10) ;
RooRealVar y(“y”,”y”,-10,10) ;
RooRealVar a(“a”,”a”,0) ;
RooRealVar b(“b”,”b”,-1.5) ;

RooFormulaVar m(“a*y+b”,a,y,b) ;
RooGaussian f(“f”,”f”,x,m,C(1)) ;

RooGaussian g(“g”,”g”,y,C(0),C(3)) ;

RooProdPdf F(“F”,”F”,g,Conditional(f,y)) ;
```

Note that code doesn’t care if input expression is variable or function!

Building power – most needed shapes already provided

- RooFit provides a collection of compiled standard PDF classes



RooBMixDecay

RooPolynomial

RooHistPdf

RooArgusBG

RooGaussian

Physics inspired

ARGUS,Crystal Ball,
Breit-Wigner, Voigtian,
B/D-Decay,….

Non-parametric

Histogram, Kernel estimation

Basic

Gaussian, Exponential, Polynomial,…
Chebychev polynomial

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

Individual classes can encapsulate powerful algorithms

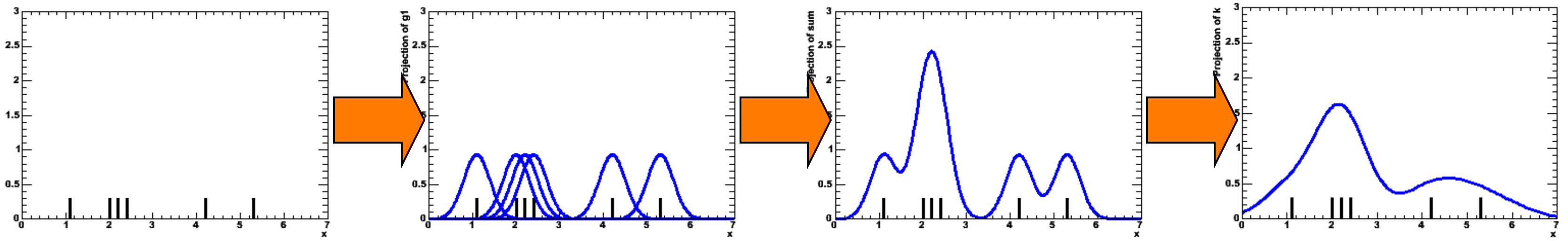
- Example: a ‘kernel estimation probability model’
  - Construct smooth pdf from unbinned data, using kernel estimation technique

Sample of events

Gaussian pdf for each event

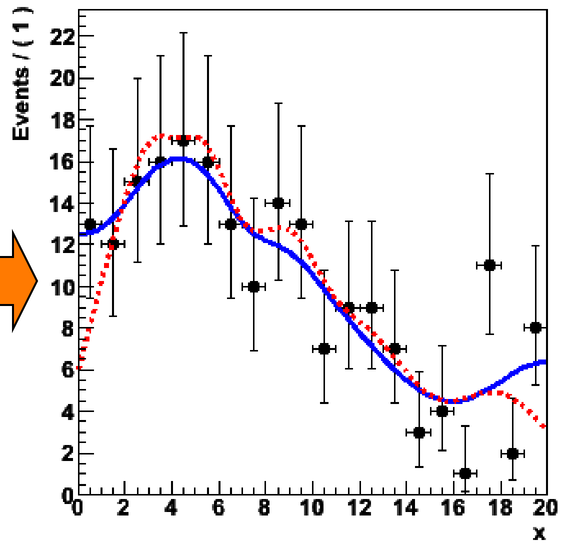
Summed pdf for all events

Adaptive Kernel: width of Gaussian depends on local event density



- Example

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

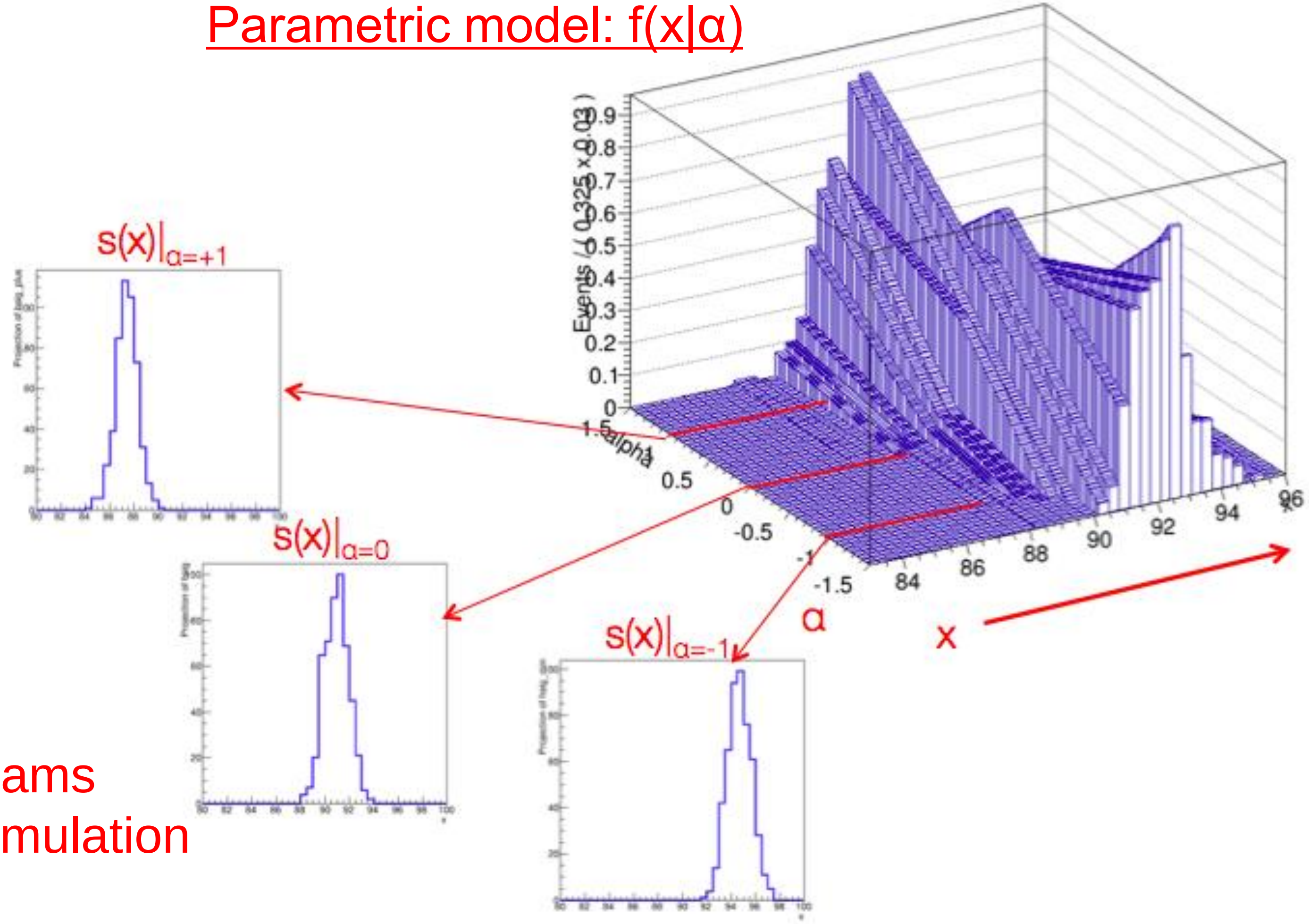


- Also available for n-D data

Advanced modeling building – template morphing

- At LHC shapes are often derived from histograms, instead of relying on analytical shapes . Construct parametric from histograms using ‘template morphing’ techniques

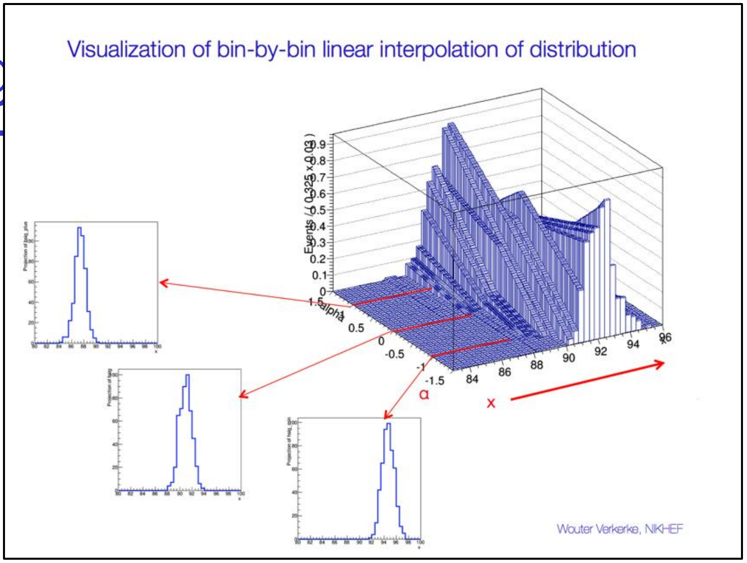
Parametric model: f(x|α)



Input histograms from simulation

Code example – template morphing

- Example of template morphing systematic in a binned likelihood



Visualization of bin-by-bin linear interpolation of distribution

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

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

Class from the HistFactory project (K. Cranmer, A. Shibata, G. Lewis, L. Moneta, W. Verkerke)

```
// 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))") ;
```

Advanced model building – describe MC statistical uncertainty

- Histogram-based models have intrinsic uncertainty to MC statistics…
- How to express corresponding shape uncertainty with model params?
  - Assign parameter to each histogram bin, introduce Poisson ‘constraint’ on each bin
  - ‘Beeston-Barlow’ technique. Mathematically accurate, but introduce results in complex models with many parameters.

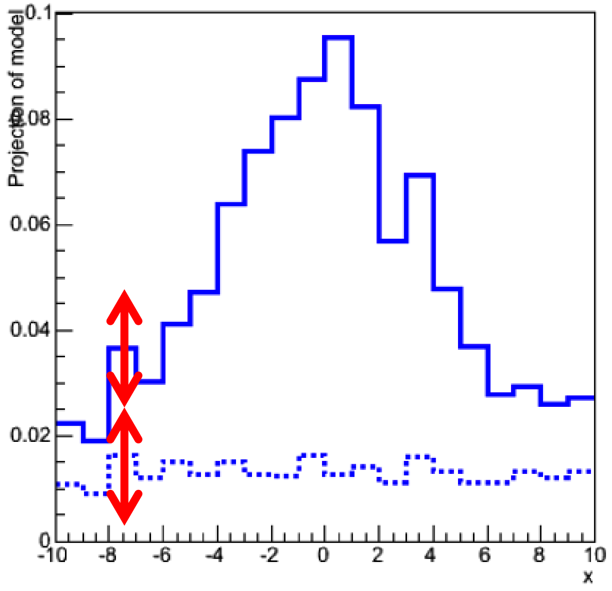
$$L(\vec{N}) = \tilde{O}_{bins} P(N_i \,|\, \tilde{s}_i + \tilde{b}_i)$$

Binned likelihood with rigid template

$$L(\vec{N}\,|\,\vec{s},\vec{b}) = \tilde{O}_{bins} P(N_i \,|\, \underbrace{s_i + b_i}) \underbrace{\tilde{O}_{bins} P(\tilde{s}_i \,|\, s_i) \tilde{O}_{bins} P(\tilde{b}_i \,|\, b_i)}$$

Response function w.r.t. s, b as parameters

Subsidiary measurements of s ,b from s~,b~



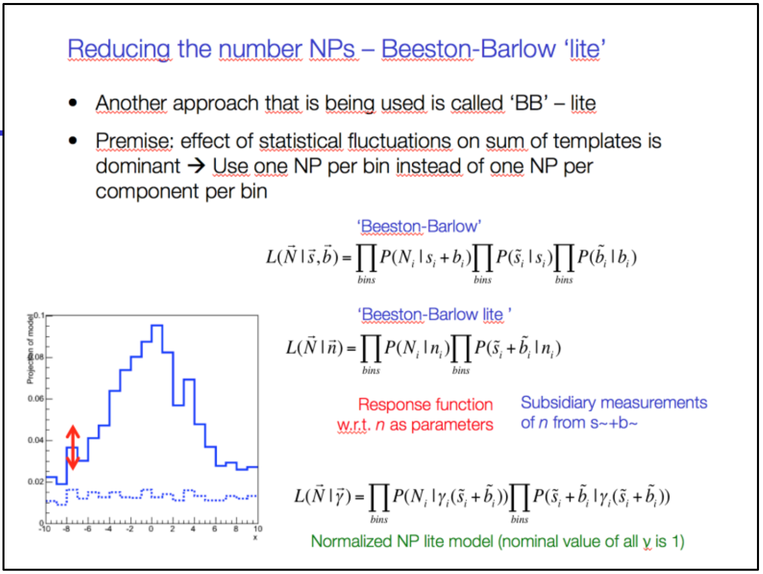
$$L(\vec{N}\,|\,\vec{g}_s,\vec{g}_b) = \tilde{O}_{bins} P(N_i \,|\, g_{s,i}\tilde{s}_i + g_{b,i}\tilde{b}_i) \tilde{O}_{bins} P(\tilde{s}_i \,|\, g_{s,i}\tilde{s}_i) \tilde{O}_{bins} P(\tilde{b}_i \,|\, g_{b,i}\tilde{b}_i)$$

Normalized NP model (nominal value of all γ is 1)

Code example – Beeston-Barlow

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

$$L(\vec{N}|\vec{g}) = \underbrace{\underset{bins}{\tilde{O}} P(N_i|g_i(\tilde{s}_i+\tilde{b}_i))}\underbrace{\underset{bins}{\tilde{O}} P(\tilde{s}_i+\tilde{b}_i|g_i(\tilde{s}_i+\tilde{b}_i))}$$



Reducing the number NPs – Beeston-Barlow ‘lite’

- Another approach that is being used is called ‘BB’ – lite
- Premise: effect of statistical fluctuations on sum of templates is dominant → Use one NP per bin instead of one NP per component per bin

‘Beeston-Barlow’

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

‘Beeston-Barlow lite ’

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

Response function w.r.t. *n* as parameters

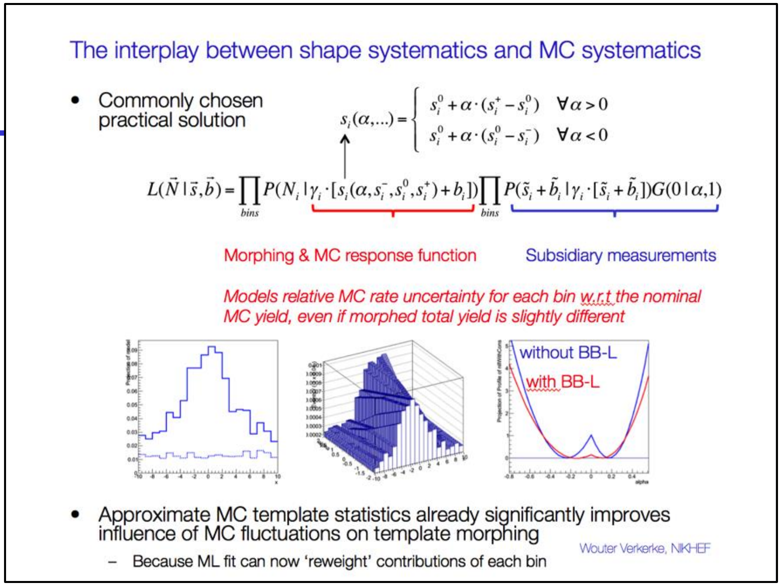
Subsidiary measurements of *n* from s~+b~

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

Normalized NP lite model (nominal value of all γ is 1)

```
// 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)”) ;
```

Code example: BB + morphing



The interplay between shape systematics and MC systematics

- Commonly chosen practical solution

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

$$L(\vec{N}|\vec{s},\vec{b}) = \prod_{bins} P(N_i|\gamma_i \cdot [\underbrace{s_i(\alpha,s_i^-,s_i^0,s_i^+) + b_i]}_{\text{Morphing \& MC response function}}) \prod_{bins} \underbrace{P(\tilde{s}_i + \tilde{b}_i | \gamma_i \cdot [\tilde{s}_i + \tilde{b}_i]) G(0|\alpha,1)}_{\text{Subsidiary measurements}}$$

Models relative MC rate uncertainty for each bin w.r.t the nominal MC yield, even if morphed total yield is slightly different

- Approximate MC template statistics already significantly improves influence of MC fluctuations on template morphing
  - Because ML fit can now 'reweight' contributions of each bin

Wouter Verkerke, NIKHEF

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

$$s_i(a,...) = \begin{cases} s_i^0 + a \times (s_i^+ - s_i^0) & " \; a > 0 \\ s_i^0 + a \times (s_i^0 - s_i^-) & " \; a < 0 \end{cases}$$

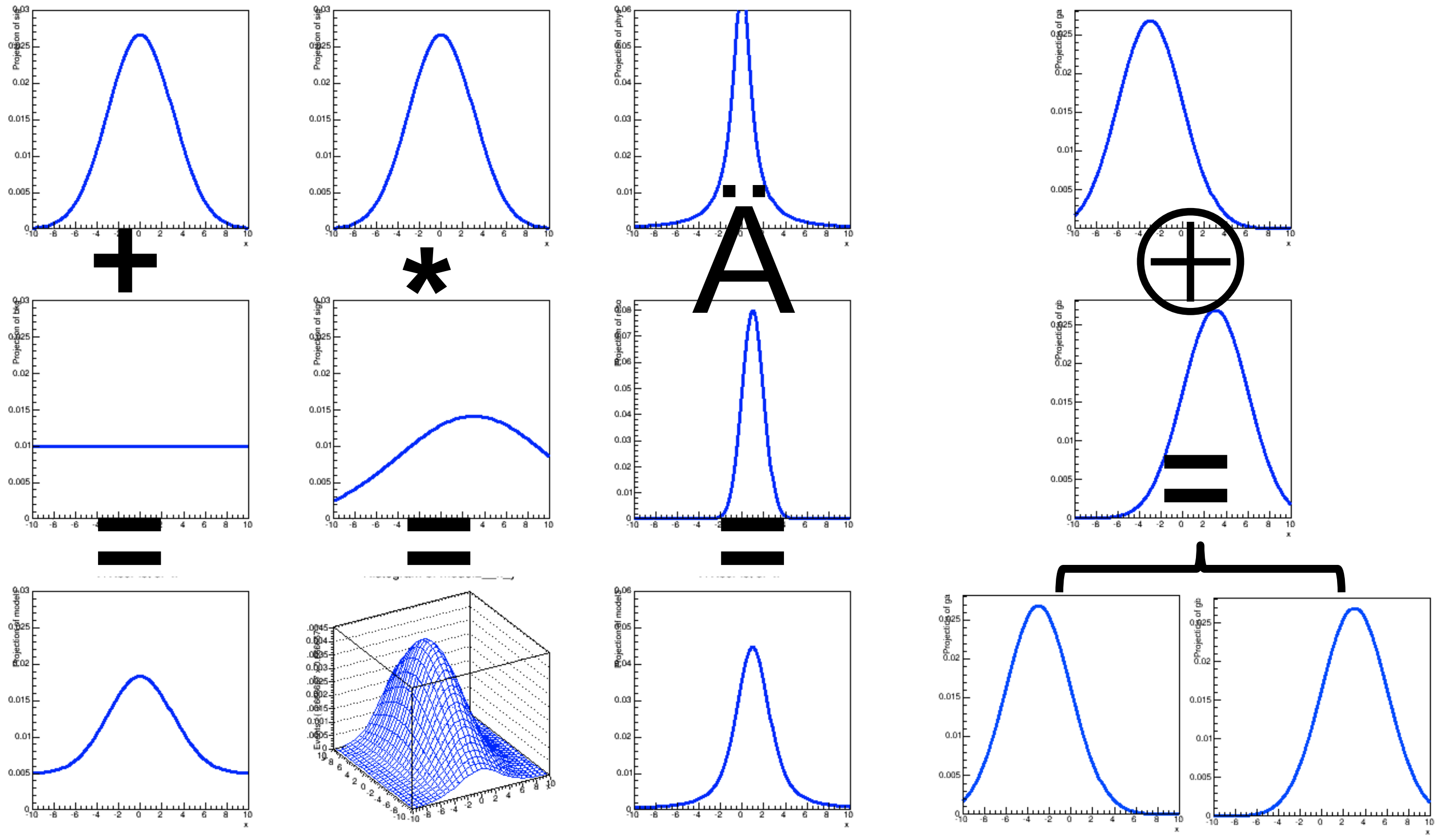
$$L(\vec{N}|\vec{s},\vec{b}) = \tilde{O}_{bins} P(N_i | g_i \times [\underbrace{s_i(a,s_i^-,s_i^0,s_i^+)}_{} + b_i]) \underbrace{\tilde{O}_{bins} P(\tilde{s}_i + \tilde{b}_i | g_i \times [\tilde{s}_i + \tilde{b}_i]) G(0|a,1)}_{}$$

```
// 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))") ;
```

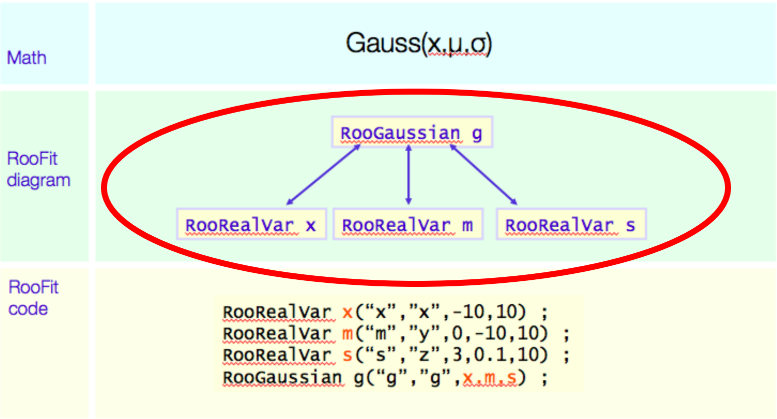
From simple to realistic models: composition techniques

- Realistic models with signal and bkg, and with control regions built from basic shapes using *addition*, *product, convolution, simultaneous* operator classes

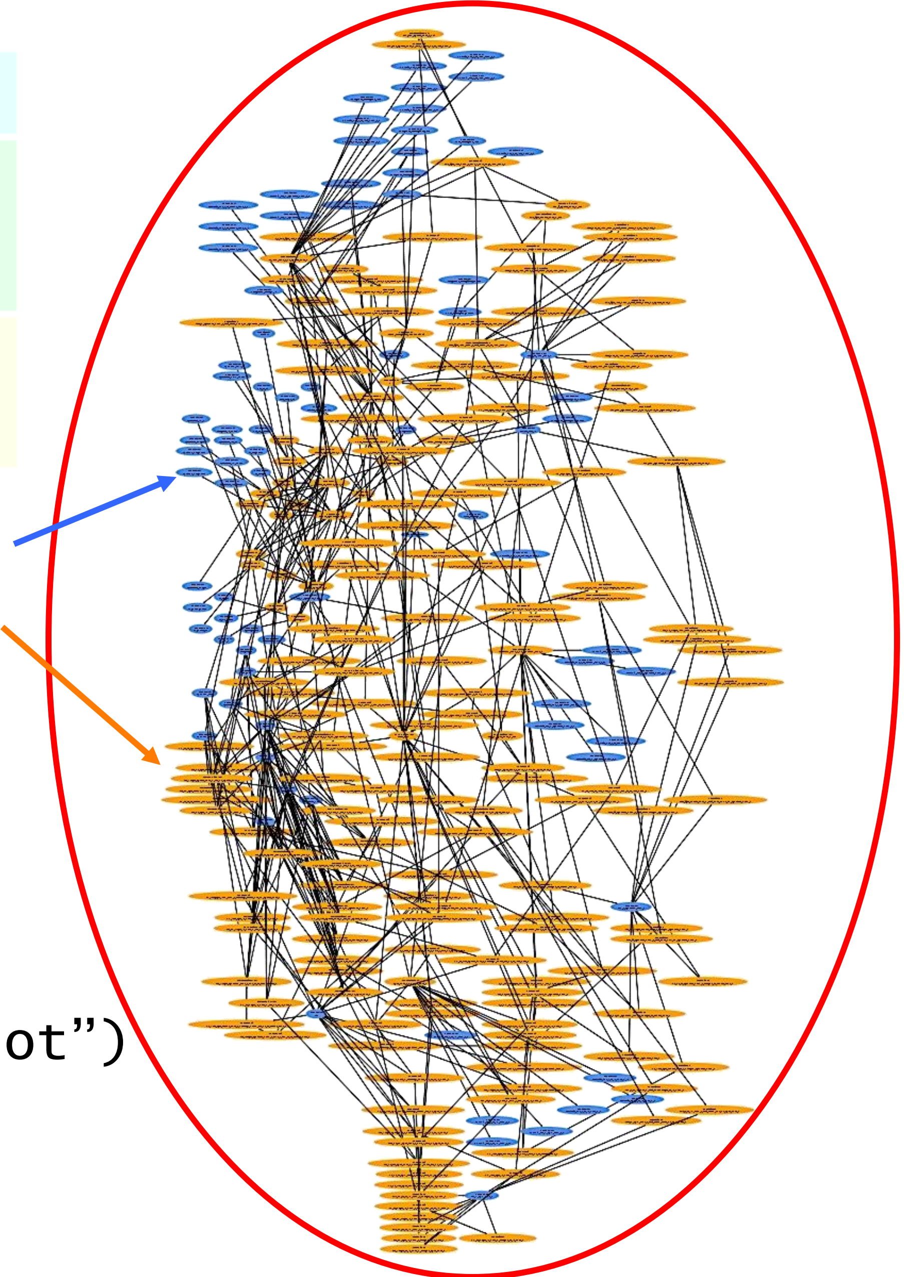
SUM PROD CONV SIMUL



Graphical example of realistic complex models



```
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) ;
```



variables

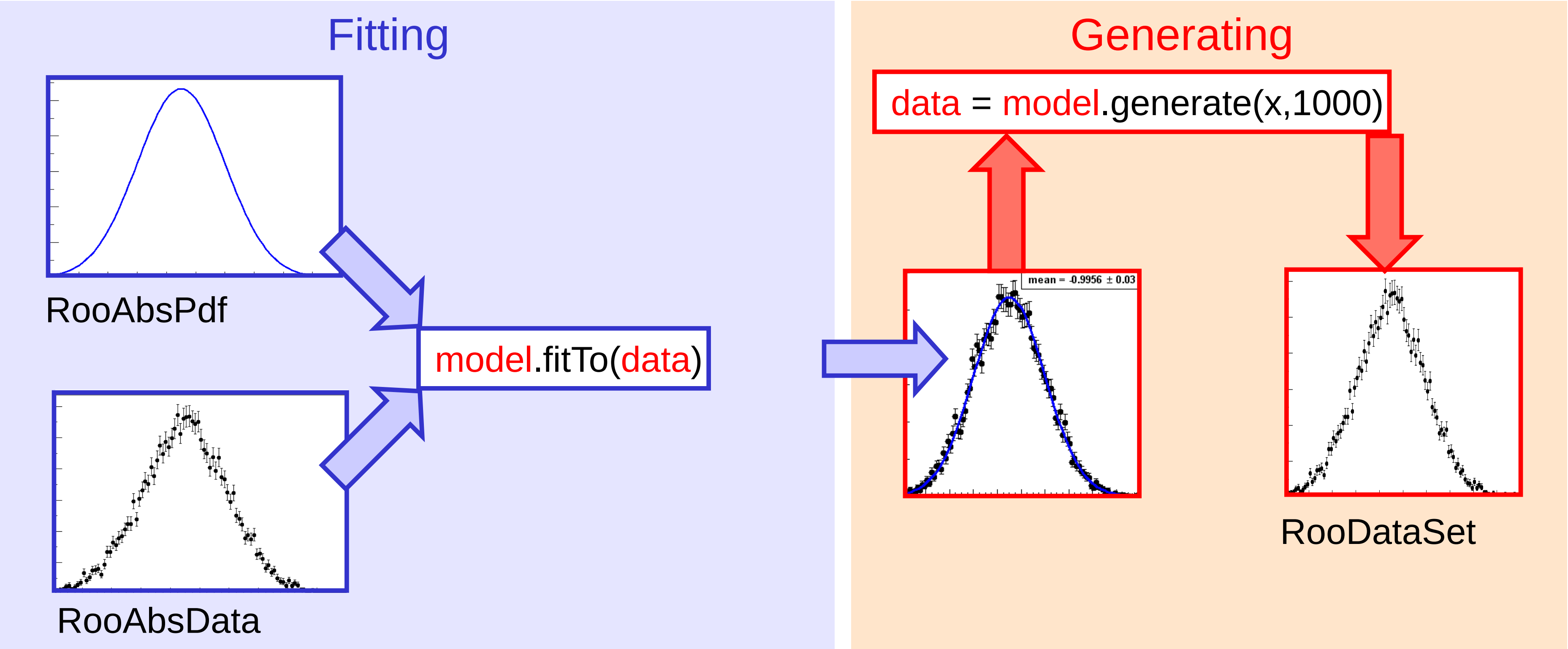
function objects

Expression graphs are autogenerated using

```
pdf->graphVizTree("file.dot")
```

Abstracting model building from model use - 1

- For *universal statistical analysis tools* (RooStats), must be have universal functionality of models (independent of structure and complexity)
- Was already possible in RooFit since 1999



Fitting

RooAbsPdf

RooAbsData

model.fitTo(data)

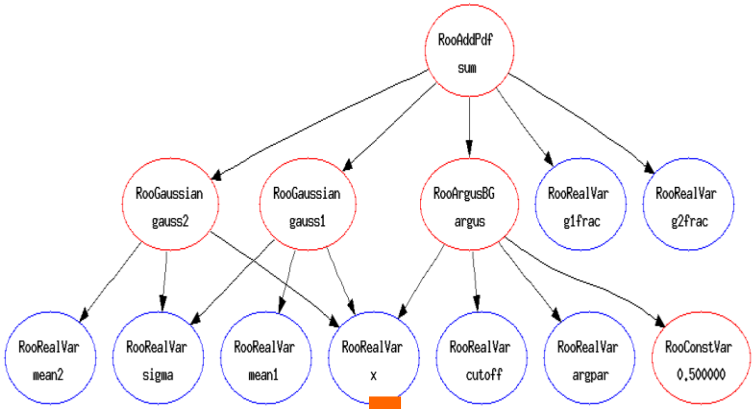
Generating

data = model.generate(x,1000)

RooDataSet

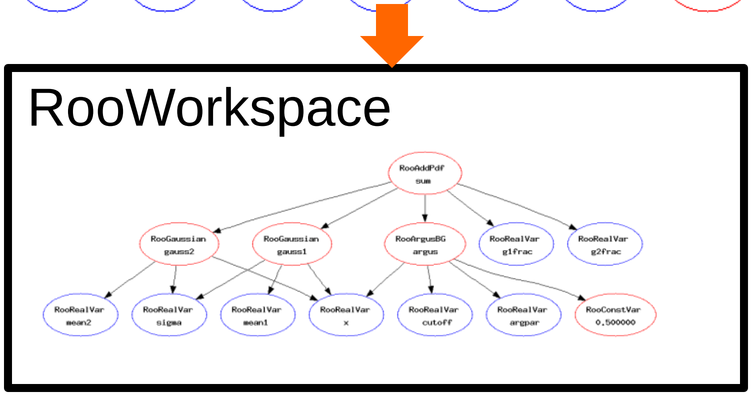
Abstracting model building from model use - 2

- Must be able to *practically* separate model building code from statistical analysis code.
- Solution: you can *persist* RooFit models of arbitrary complexity in ‘workspace’ containers
- The workspace concept has revolutionized the way people share and combine analyses!



Realizes complete and practical factorization of process of building and using likelihood functions!

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

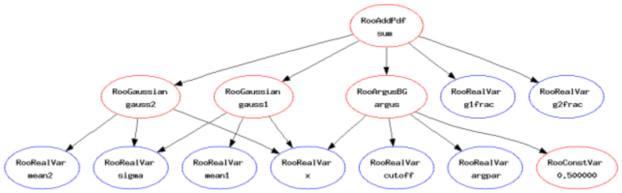


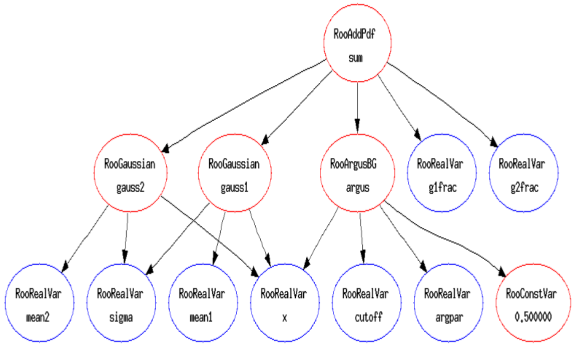
RooWorkspace

model.root

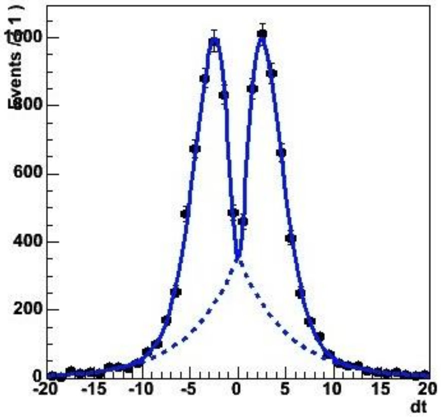
Using a workspace file given to you…

RooWorkspace



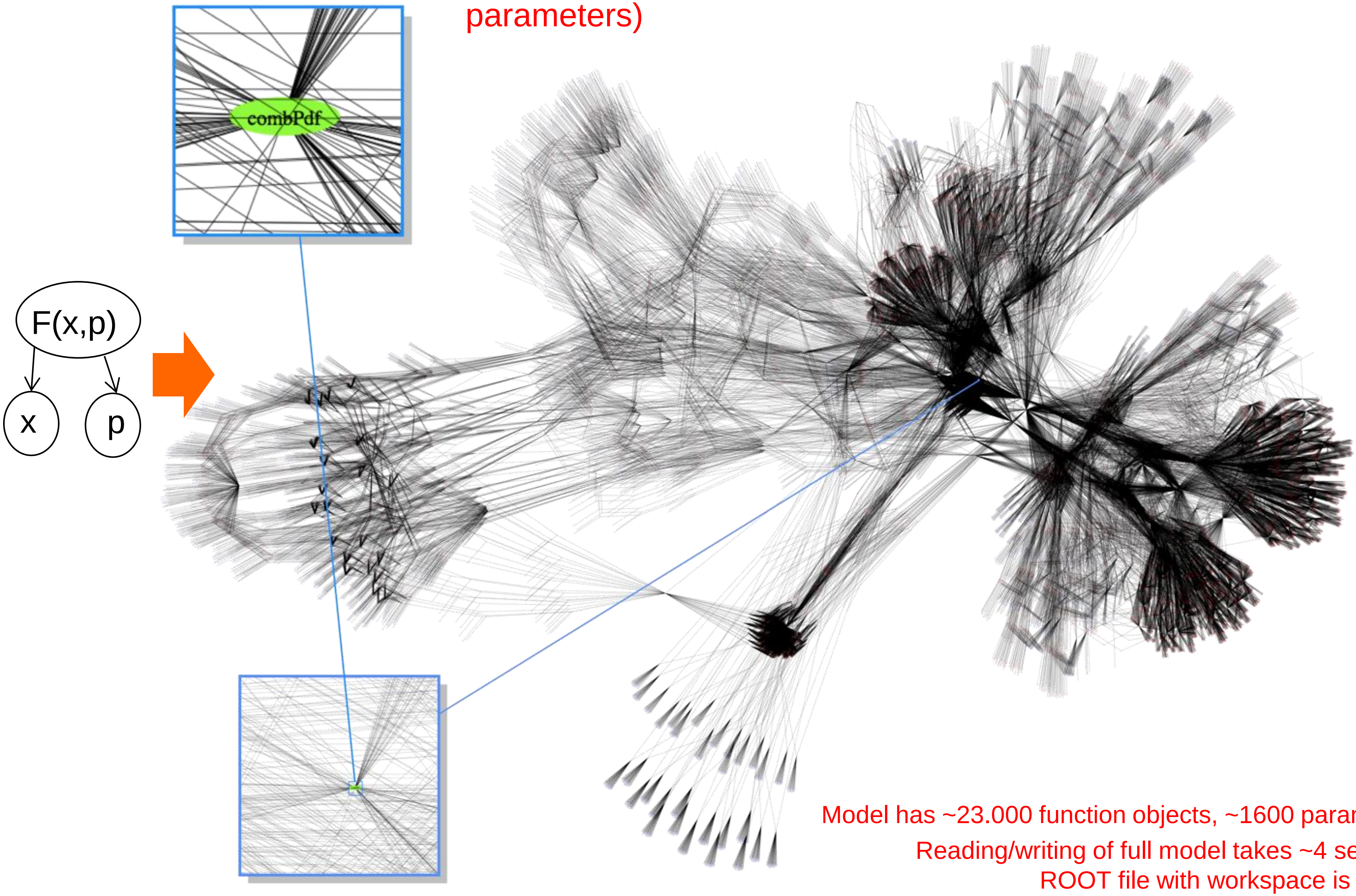


```
// Resurrect model and data
TFile f(“model.root”) ;
RooWorkspace* w = f.Get(“w”) ;
RooAbsPdf* model = w->pdf(“sum”) ;
RooAbsData* data = w->data(“xxx”) ;

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

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



Persistence of *really* complex models works too!

Atlas Higgs combination model (23.000 functions, 1600 parameters)



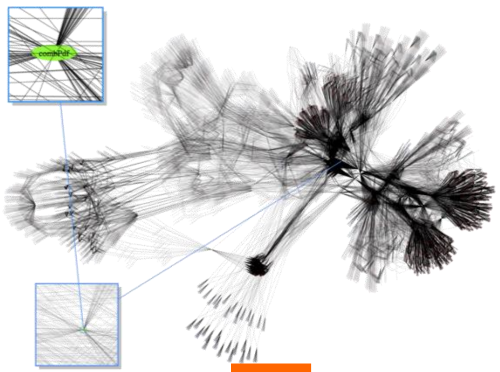
Model has ~23.000 function objects, ~1600 parameters
Reading/writing of full model takes ~4 seconds
ROOT file with workspace is ~6 Mb

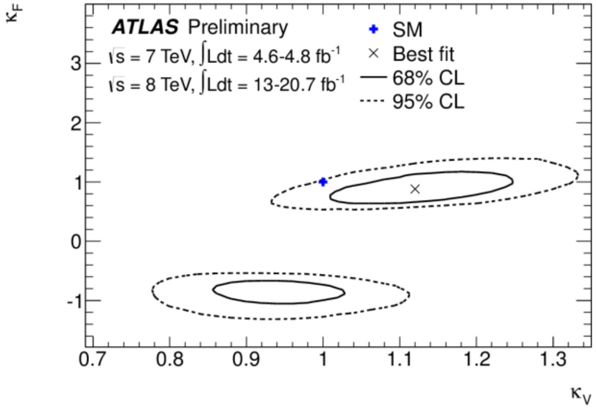
An excursion – Collaborative analyses with workspaces

- Workspaces allow to share and modify very complex analyses with very little technical knowledge required

- Example: Higgs coupling fits

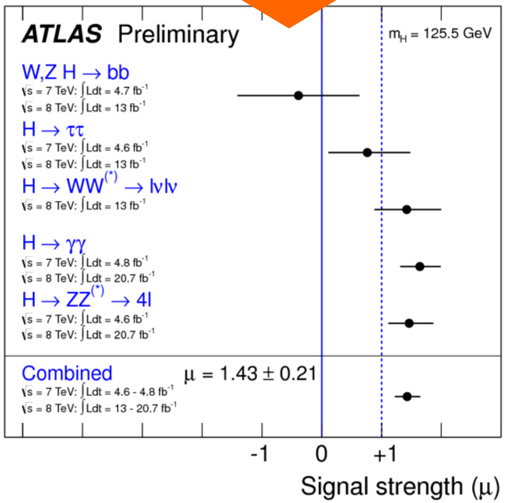
Full Higgs model





Confidence intervals on Higgs fermion, boson couplings

Signal strength in 5 channels



$$
\begin{aligned}
\sigma(gg \to H) * \mathrm{BR}(H \to \gamma\gamma) &\sim \frac{\kappa_F^2 \cdot \kappa_\gamma^2(\kappa_F, \kappa_V)}{0.75 \cdot \kappa_F^2 + 0.25 \cdot \kappa_V^2} \\
\sigma(qq' \to qq'H) * \mathrm{BR}(H \to \gamma\gamma) &\sim \frac{\kappa_V^2 \cdot \kappa_\gamma^2(\kappa_F, \kappa_V)}{0.75 \cdot \kappa_F^2 + 0.25 \cdot \kappa_V^2} \\
\sigma(gg \to H) * \mathrm{BR}(H \to ZZ^{(*)}, H \to WW^{(*)}) &\sim \frac{\kappa_F^2 \cdot \kappa_V^2}{0.75 \cdot \kappa_F^2 + 0.25 \cdot \kappa_V^2} \\
\sigma(qq' \to qq'H) * \mathrm{BR}(H \to ZZ^{(*)}, H \to WW^{(*)}) &\sim \frac{\kappa_V^2 \cdot \kappa_V^2}{0.75 \cdot \kappa_F^2 + 0.25 \cdot \kappa_V^2} \\
\sigma(qq' \to qq'H, VH) * \mathrm{BR}(H \to \tau\tau, H \to b\bar{b}) &\sim \frac{\kappa_V^2 \cdot \kappa_F^2}{0.75 \cdot \kappa_F^2 + 0.25 \cdot \kappa_V^2} \quad ,
\end{aligned}
$$

Reparametrize in terms of fermion, boson scale factors

An excursion – Collaborative analyses with workspaces

- How can you reparametrize existing Higgs likelihoods *in practice*?
- Write functions expressions corresponding to new

$$\sigma(gg \rightarrow H) * \mathrm{BR}(H \rightarrow \gamma\gamma) \sim \frac{\kappa_F^2 \cdot \kappa_\gamma^2(\kappa_F, \kappa_V)}{0.75 \cdot \kappa_F^2 + 0.25 \cdot \kappa_V^2}$$

```
RooFormulaVar mu_gg_func(“mu_gg_func”,
                         “(KF2*Kg2)/(0.75*KF2+0.25*KV2)”,
                         KF2,Kg2,KV2) ;
```

- Edit existing model

```
w.import(mu_gg_func) ;
w.factory(“EDIT::newmodel(model,mu_gg=mu_gg_gunc)”) ;
```

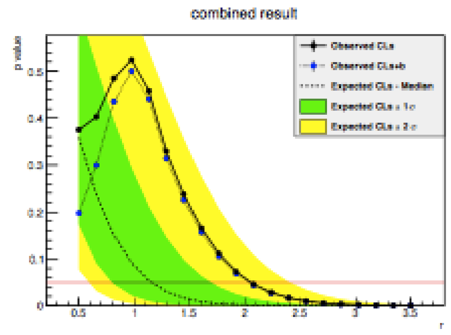
Top node of *modified* Higgs combination pdf

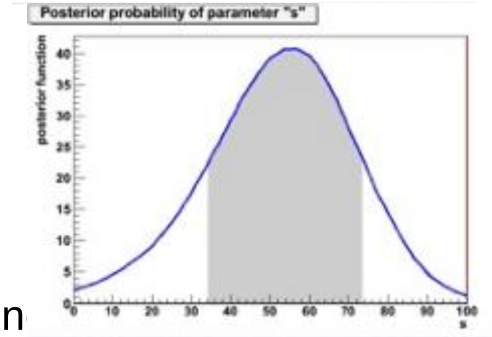
Top node of *original* Higgs combination pdf

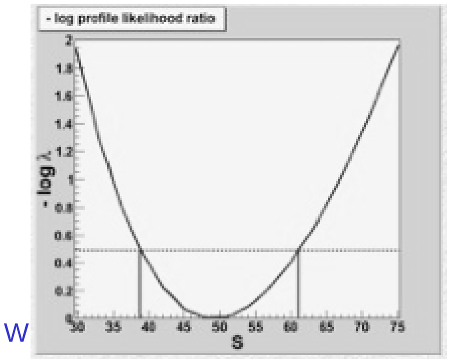
Modification prescription: replace parameter mu_gg with function mu_gg_func everywhere

RooStats – Statistical analysis of RooFit models

- With RooFits one has (almost) limitless possibility to construct probability density models
  - With the workspaces one also has the ability to deliver such models to statistical tools that are completely decoupled from the model construction code.
    Will now focus on the design of those statistical tools



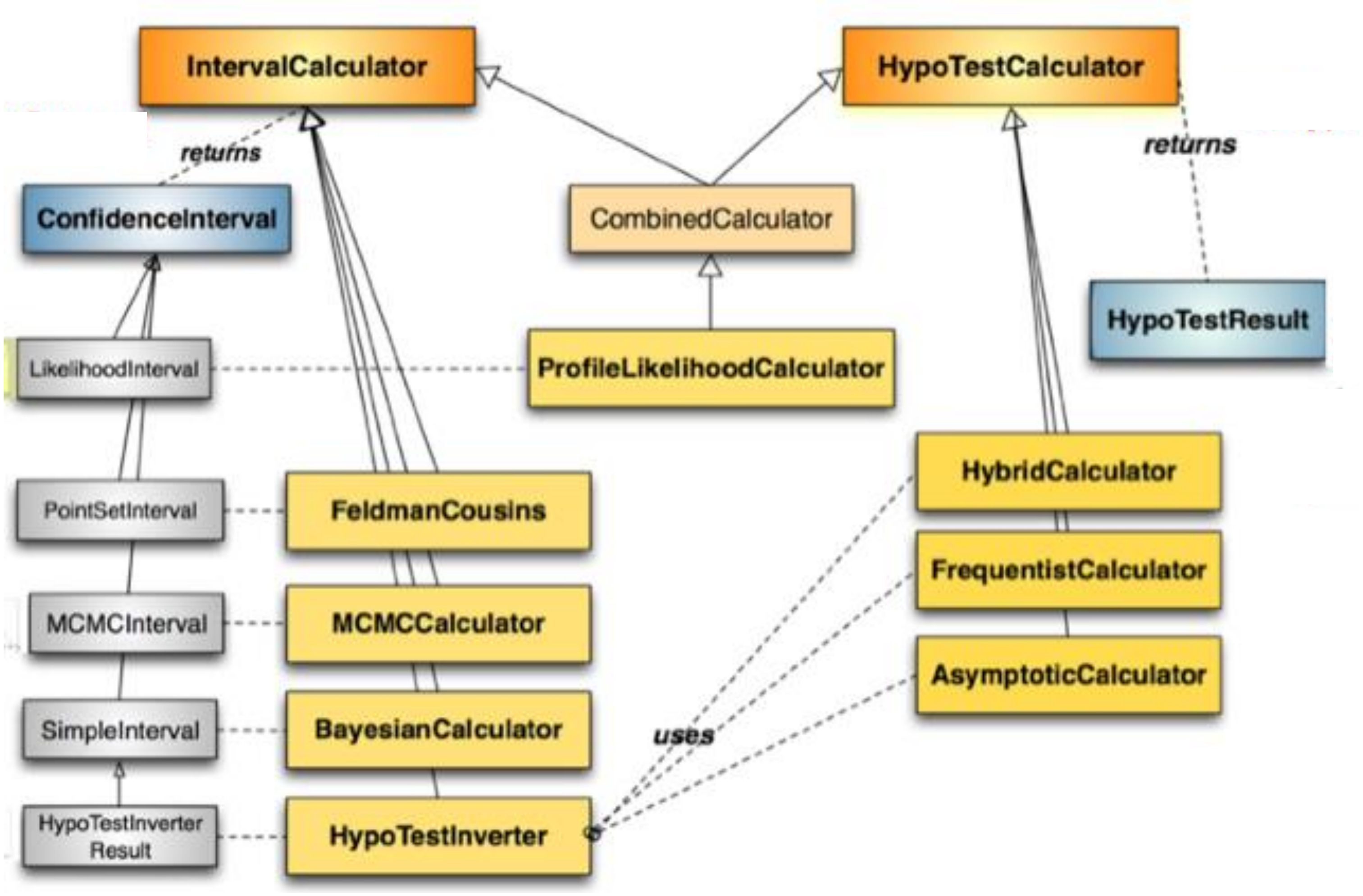




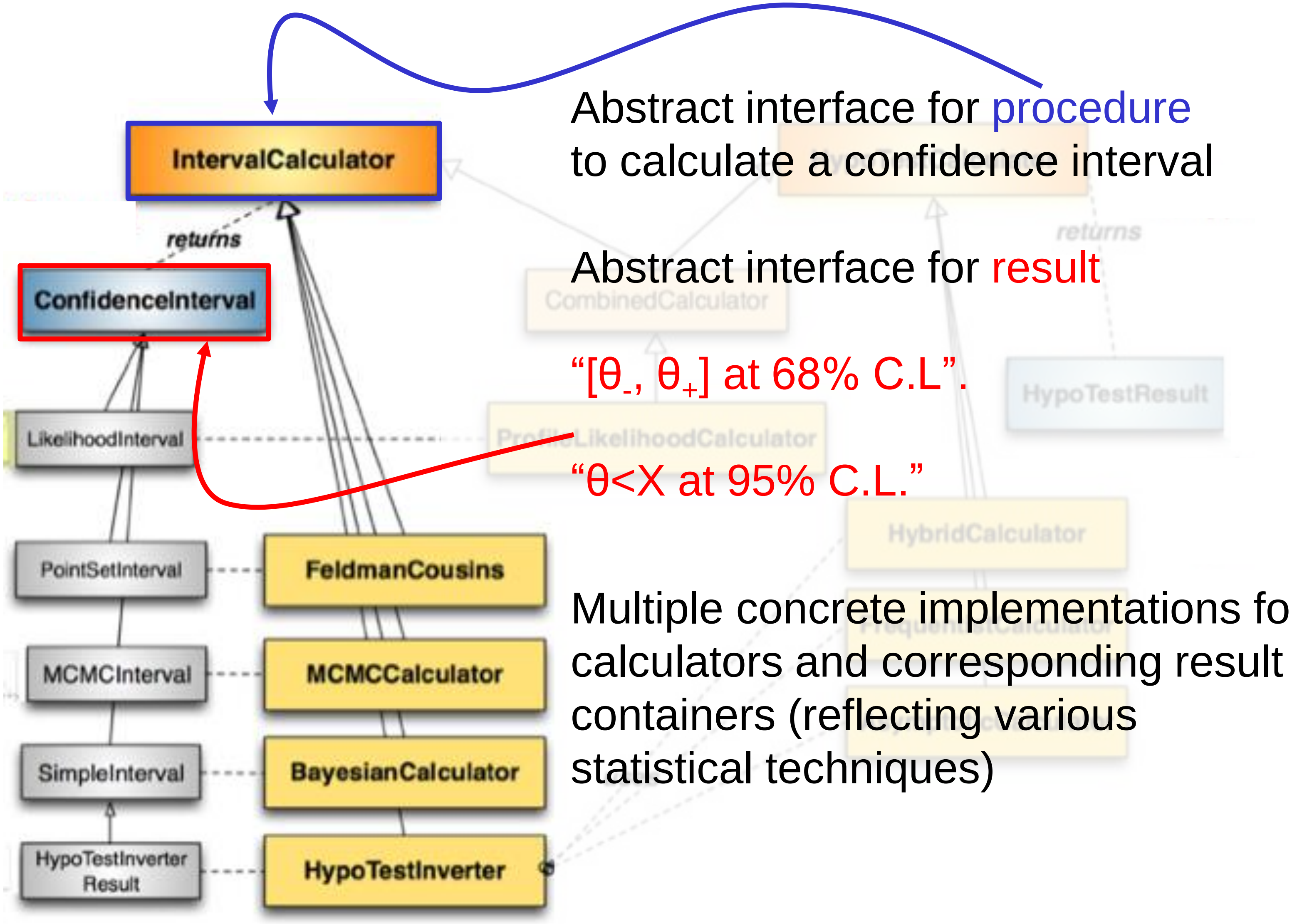
- The RooStats projected was started in 2007 as a joint venture between ATLAS, CMS, the ROOT team and myself.
  Goal: to deliver a series of tools that can calculate intervals and perform hypothesis tests using a variety of statistical techniques
  - Frequentist methods (confidence intervals, hypothesis testin
  - Bayesian methods (credible intervals, odd-ratios)
  - Likelihood-based methods

Confidence intervals: $[\theta_-, \theta_+]$,or $\theta<X$ at 95% C.L.
Hypothesis testing: → $p(data|\theta=0) = 1.10^{-7}$

RooStats class structure



RooStats class structure



Abstract interface for procedure to calculate a confidence interval

Abstract interface for result

"[θ_-, θ_+] at 68% C.L".

"θ<X at 95% C.L."

Multiple concrete implementations for calculators and corresponding result containers (reflecting various statistical techniques)

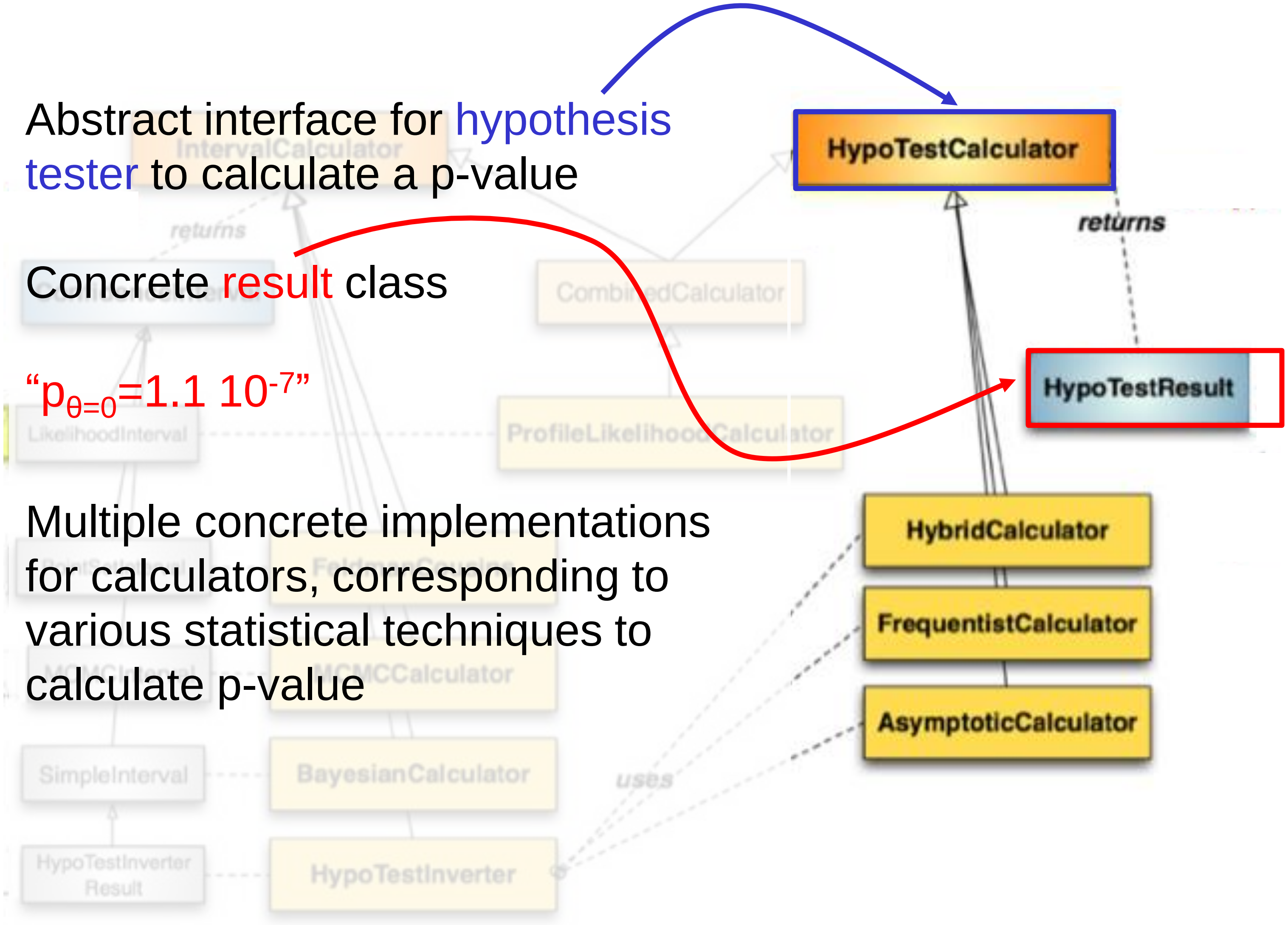
RooStats class structure

Abstract interface for hypothesis tester to calculate a p-value

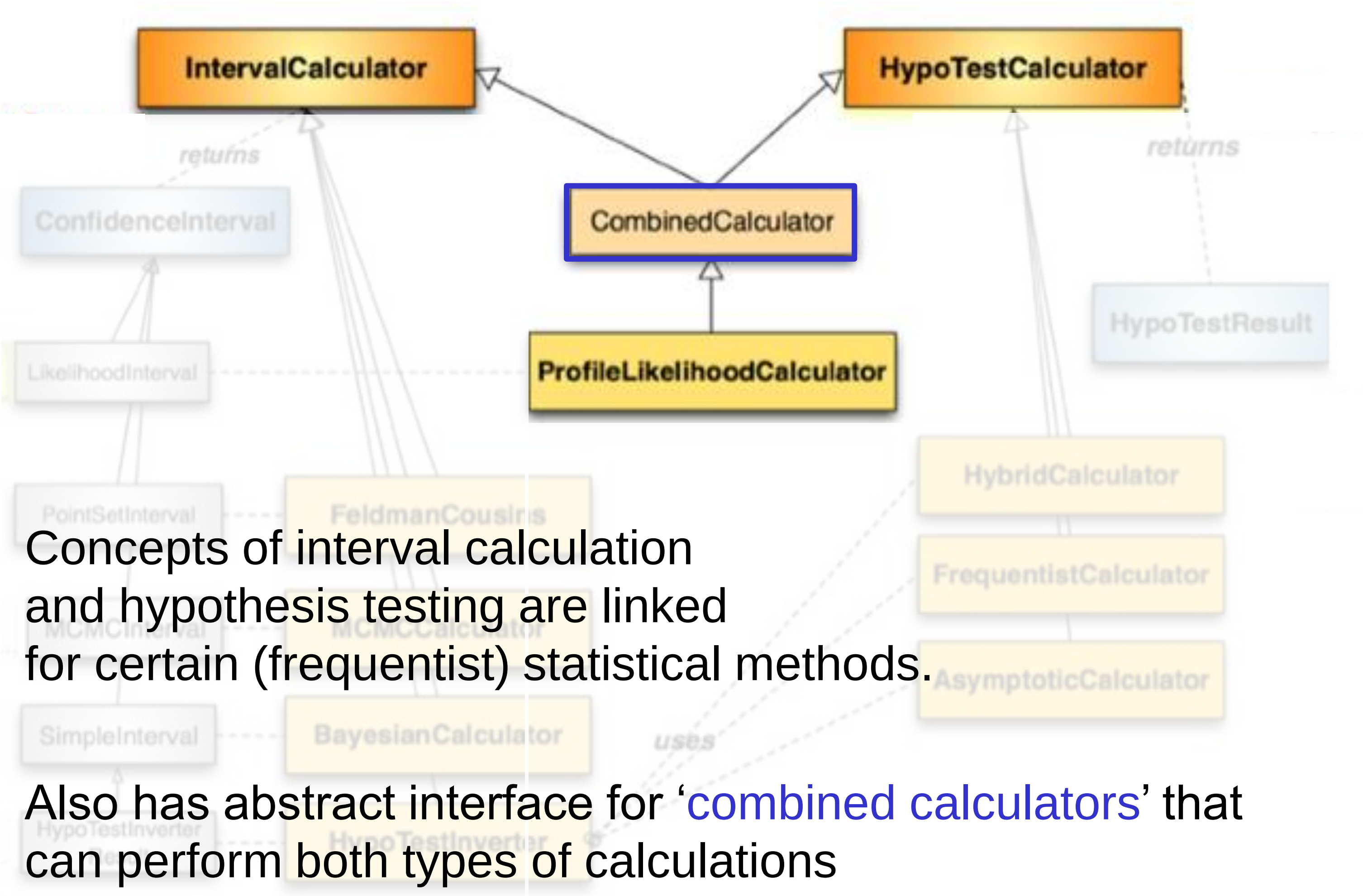
Concrete result class

“$p_{\theta=0}=1.1\ 10^{-7}$”

Multiple concrete implementations for calculators, corresponding to various statistical techniques to calculate p-value



RooStats class structure

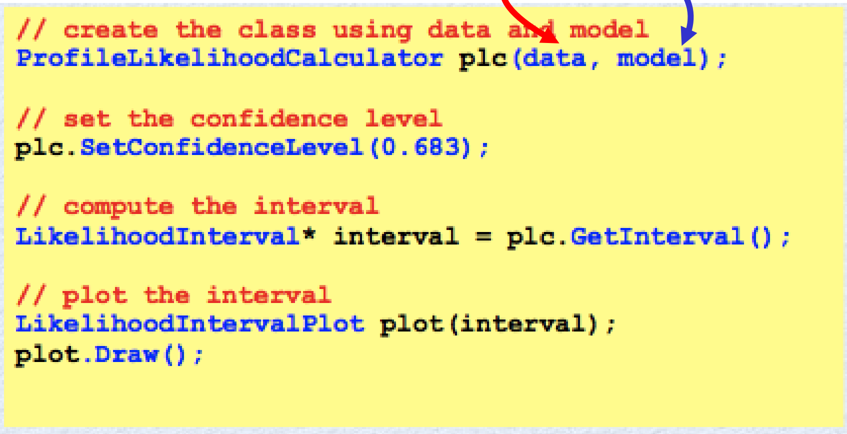


Concepts of interval calculation
and hypothesis testing are linked
for certain (frequentist) statistical methods.

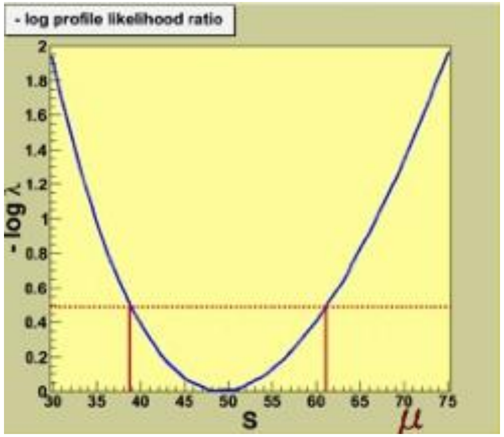
Also has abstract interface for ‘combined calculators’ that
can perform both types of calculations

Working with RooStats calculators

- Calculators interface to RooFit via a ‘ModelConfig’ object
- ModelConfig completes $f(x|\theta)$ from workspace with additional information to become an *unambiguous statistical problem specification* (together with x_{obs})
  - E.g. which of parameters θ are ‘of interest’ which are ‘nuisance parameters’.
  - For certain types of complex models, additional info is needed

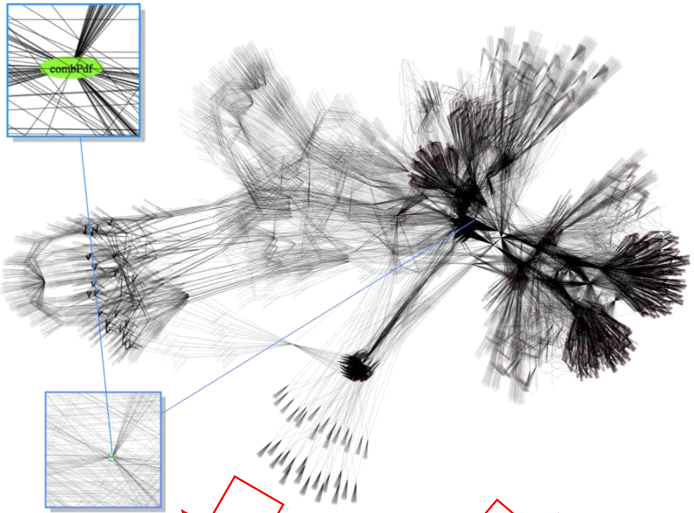


```
// create the class using data and model
ProfileLikelihoodCalculator plc(data, model);

// set the confidence level
plc.SetConfidenceLevel(0.683);

// compute the interval
LikelihoodInterval* interval = plc.GetInterval();

// plot the interval
LikelihoodIntervalPlot plot(interval);
plot.Draw();
```



- Calculator works for any model, no matter how complex

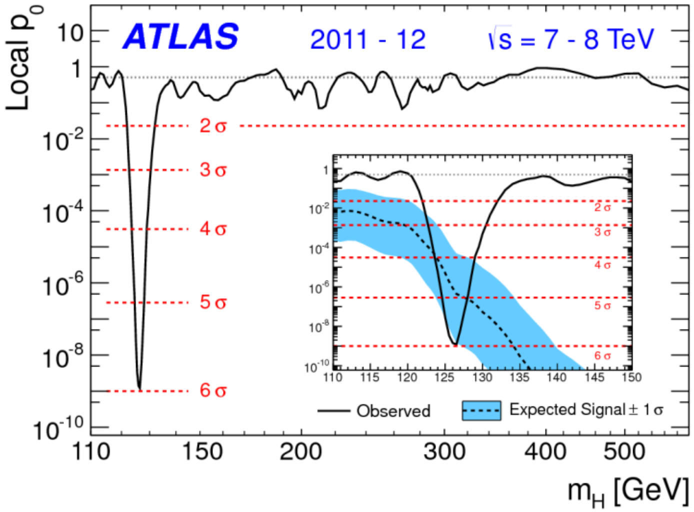
Some famous RooFit/RooStats results



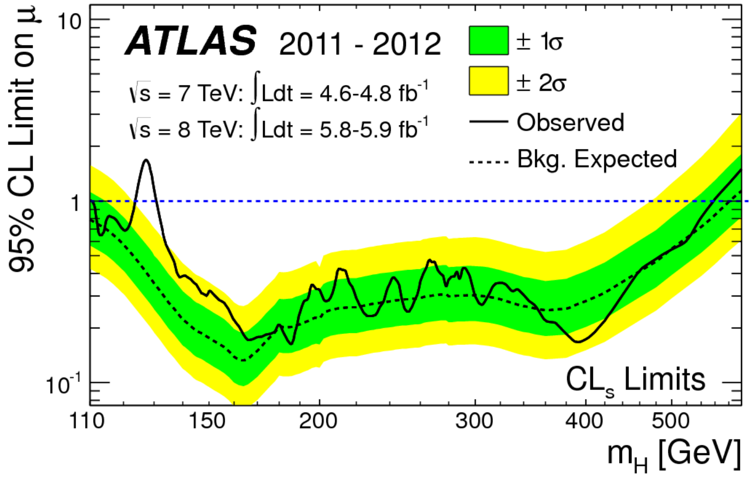
RooFit workspace with

Atlas Higgs combination model (23.000 functions, 1600 parameters)

RooStats hypothesis testing (p-value of bkg hypothesis)



RooStats interval calculation (upper limit intervals at 95%)



Performance considerations

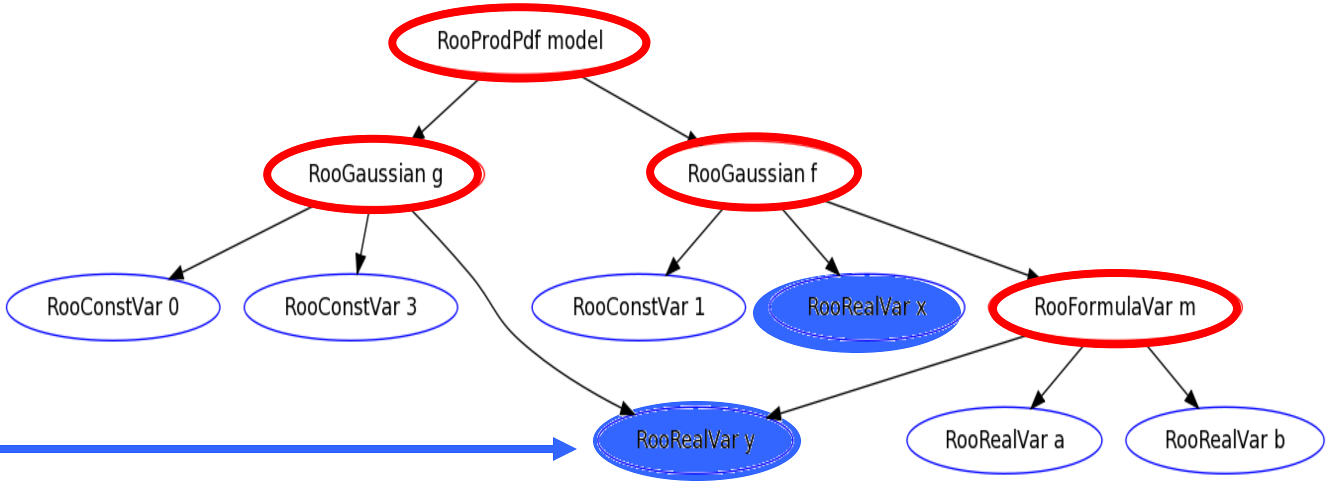
- While functionality is (nearly) universal, good computational performance for all models requires substantial work behind the scenes.
  - Will highlight three techniques that are used to boost performance
- Heuristic constant-expression detection
  - Identify (highest)-level constant expression in user expression in a given use context and prevent unnecessary recalculation of these
- (Pseudo)-vectorization
  - Reorder calculations to approach concept of vectorization
- Parallelization
  - Exploit pervasive ability of CPU farms and multi-core host to parallelize calculations that intrinsically of a repetitive nature
- The boundary condition of all optimizations is that user code should not need to accommodate these.
  - User probability models are often already complex, must be kept in ‘most readable’ representation
  - Use RooFit model introspection to reorganize user functions ‘on the fly’ in vectorization-friendly order

Optimization of likelihood calculations

- Likelihood evaluates pdf at all data points, essentially a ‘loop’ call

$$-\log L(\vec{p}) = -\sum_{i=0\ldots n} \log f(\vec{x}_i,\vec{p})$$

X	Y
1	2
2	3
0	1
-1	5
3	6
7	6
-3	-2

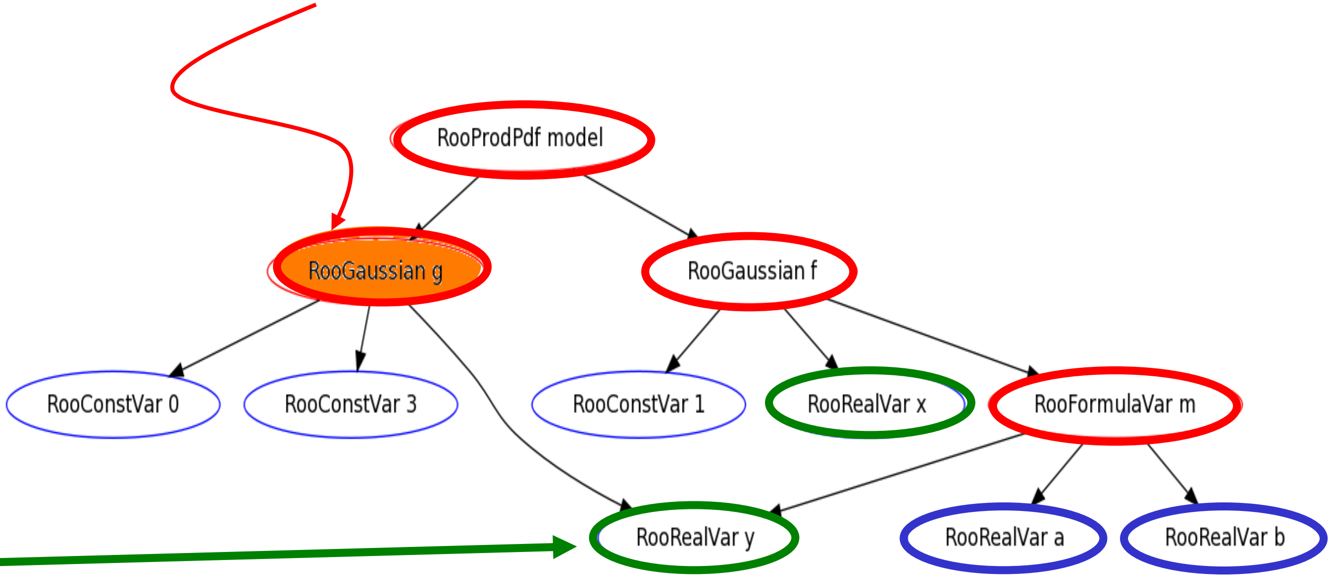


As written by user, the p.d.f is a scalar expression that is *unaware* of underlying repeated calculation of likelihood

Level-1 optimization of likelihood calculation

- RooFit can heuristically detect constant terms (depends only on observables, not on parameters) are pre-calculated, cached with likelihood dataset. *Calculation tree modified to omit recalculation of*

X	Y	g
1	2	1.5
2	3	2.7
0	1	1.2
-1	5	0.6
3	6	9.8
7	6	3.4
-3	-2	5.7



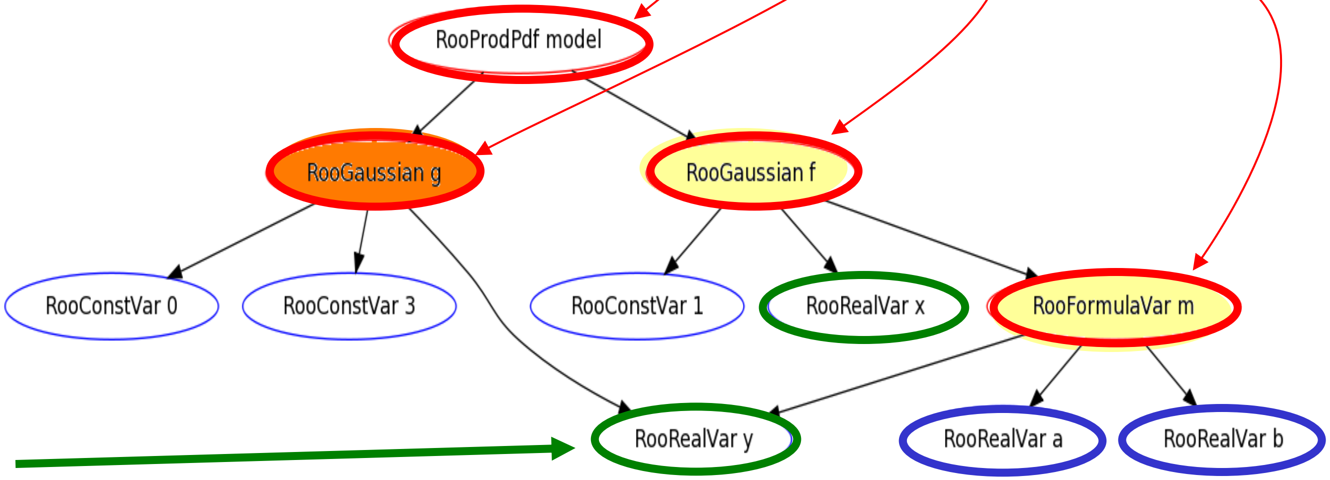
Level-2 optimization of likelihood calculation

- Can also apply caching strategy to *all functions nodes*, instead of just constant nodes

Depends on m

Depends on a,b

X	Y	g	f	m
1	2	1.5	..	..
2	3	2.7	..	..
0	1	1.2	..	..
-1	5	0.6	..	..
3	6	9.8	..	..
7	6	3.4	..	..
-3	-2	5.7	..	..



To ensure correct calculation:
Value cache of non-constant function objects will be invalidated if dependent parameters changed

Faster than level-1 if non-constant cache miss rate <100%

What is the value cache miss rate for non-constant objects?

- It is quite a bit better than 100% as most MINUIT calls to likelihood vary one parameter at a time (to calculate derivative) → Computed cached values will often stay valid

```
prevFCN = 5170.289989   FCN=5170.53 FROM MIGRAD  STATUS=INITIATE  6 CALLS  7 TOTAL
prevFCN = 4495.931306  a=0.9961, b=0.106, c=0.06274,
prevFCN = 3936.921265  a=0.9967,
prevFCN = 3936.938281  a=0.9954,
prevFCN = 3936.907905  a=0.9965,
prevFCN = 3936.933086  a=0.9956,
prevFCN = 3936.911321  a=0.9961, b=0.108,
prevFCN = 3937.05644  b=0.104,
prevFCN = 3936.790003  b=0.1074,
prevFCN = 3937.014478  b=0.1046,
prevFCN = 3936.829929  b=0.106, c=0.06845,
prevFCN = 3936.934463  c=0.05703,
prevFCN = 3936.911648  c=0.06688,
prevFCN = 3936.930463  c=0.05861,
prevFCN = 3936.913944  a=1, b=-0.02103, c=0.02074,
prevFCN = 3936.613348  a=0.9982, b=0.04018, c=0.04096,
```

Only a changes, caches depending on b,c remain valid

Only b changes, caches depending on a,c remain valid

Only c changes, caches depending on b,c remain valid

From level-2 optimization to vectorization

- Note that resequencing of calculation in full level-2 optimization mode results in ‘natural ordering’ for *complete vectorization*

Level-1 sequence

$m(y_0)$
$f(m_0)$
$g(x_0)$
$Model(f_0,g_0)$

$m(y_1)$
$f(m_1)$
$g(x_1)$
$Model(f_1,g_1)$

$m(y_2)$
$f(m_2)$
$g(x_2)$
$Model(f_2,g_2)$

Level-2-max sequence

$m(y_0)$
$m(y_1)$
$m(y_2)$

$f(m_0)$
$f(m_1)$
$f(m_2)$

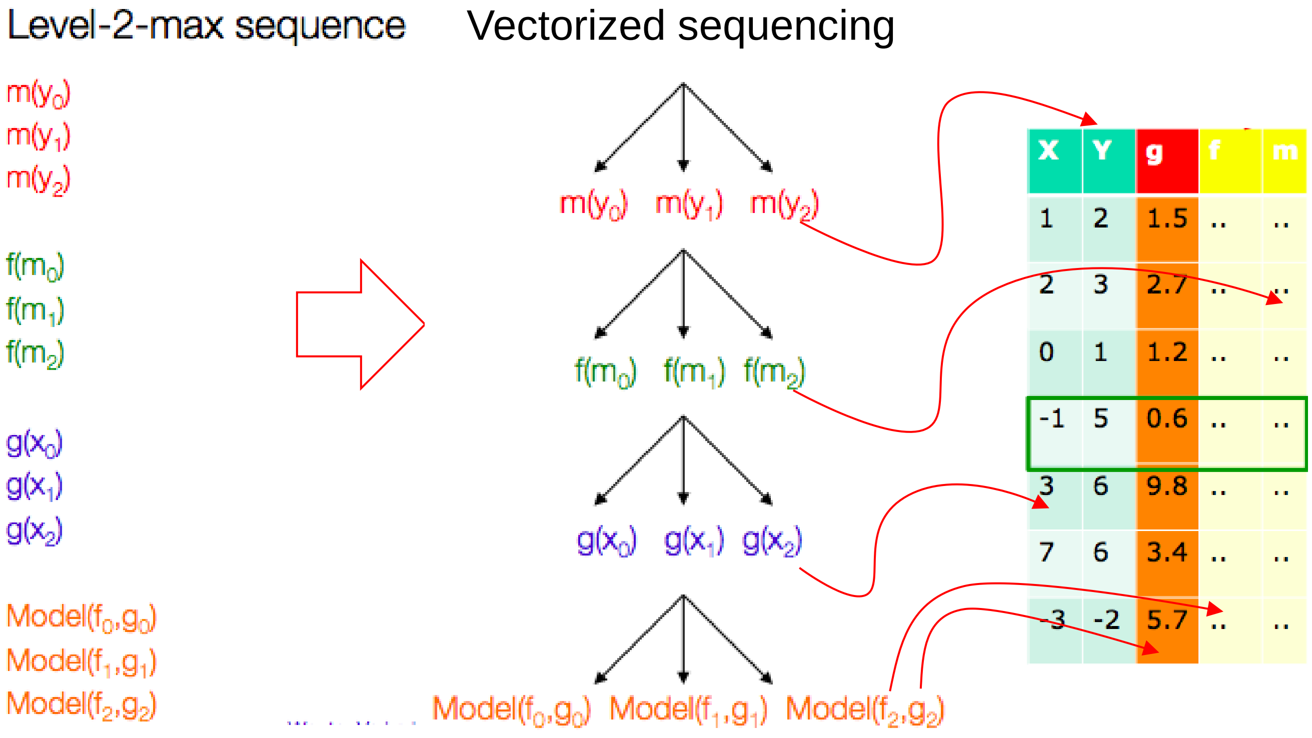
$g(x_0)$
$g(x_1)$
$g(x_2)$

$Model(f_0,g_0)$
$Model(f_1,g_1)$
$Model(f_2,g_2)$

Work in progress – automatic code vectorization

- Axel noted in his plenary presentation that ‘vectorization’ is invasive… True, but modular structure of RooFit function expression allows this invasive reorganization to be performed *automatically. Aim to vectorize code without making the ‘user code’ messy!*

Construct custom sequence driver on the fly with CLING to eliminate virtual function calls



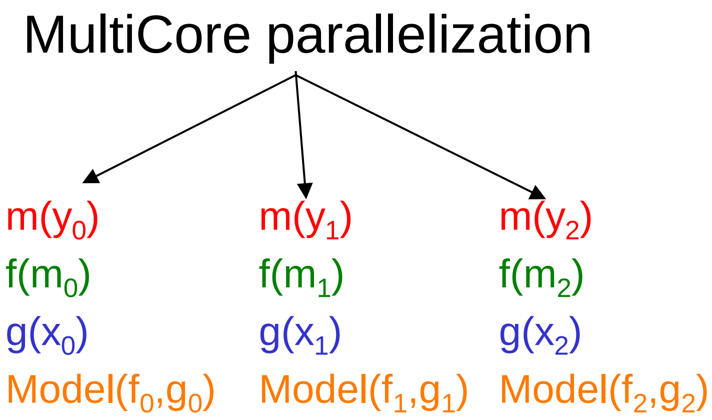
X	Y	g	f	m
1	2	1.5	..	..
2	3	2.7	..	..
0	1	1.2	..	..
-1	5	0.6	..	..
3	6	9.8	..	..
7	6	3.4	..	..
-3	-2	5.7	..	..

Level-2 optimization ensures all inputs are already in vector form

But, as inputs are already always held in proxies in user code, user code is unaware of scalar/vector nature of inputs

Other parallelization techniques – multicore Likelihood calculation

- Parallelization of calculations already introduce at a higher level
- Multi-core calculation of likelihood at the granularity of the event level, rather than the function call level



MultiCore parallelization

$m(y_0)$ $m(y_1)$ $m(y_2)$

$f(m_0)$ $f(m_1)$ $f(m_2)$

$g(x_0)$ $g(x_1)$ $g(x_2)$

$\mathrm{Model}(f_0,g_0)$ $\mathrm{Model}(f_1,g_1)$ $\mathrm{Model}(f_2,g_2)$

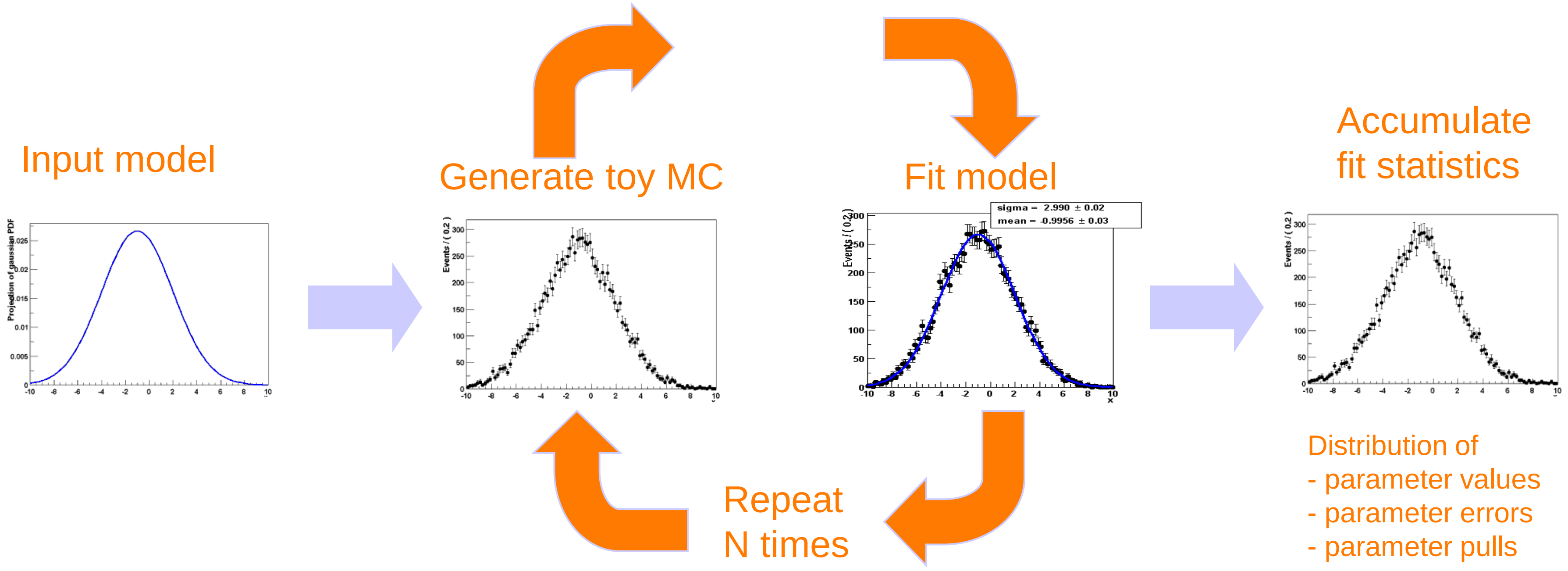
   - Trivial use invocation make this already popular with users

```
model->fitTo(data,NumCPU(8),…)
```

   - But load balancing can become uneven for ‘simultaneous fits’ (not every event has the same probability model in that case)

Parallelization using PROOF

- Simple parallelization of likelihood calculation using NumCPU(n) option of RooAbsPdf::fitTo() very popular, but restricted to likelihood calculations
- Another common CPU-intensive task are toy studies



Input model

Generate toy MC

Fit model

Accumulate fit statistics

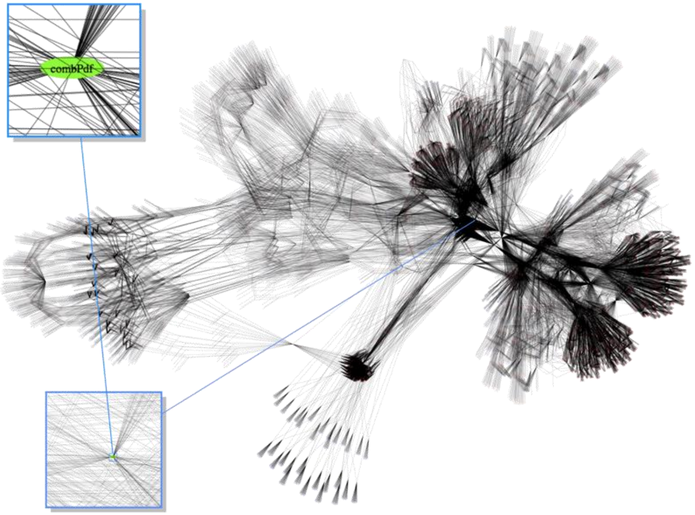
Repeat N times

Distribution of
- parameter values
- parameter errors
- parameter pulls

- Have generic interface to PROOF(-lite) to parallelize loop tasks. *Also used by RooStats for sampling procedures*

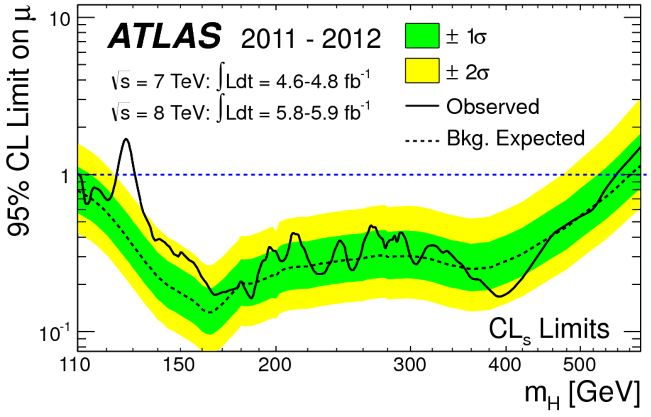
Summary

- RooFit and RooStats allow you to perform advanced statistical data analysis
  - LHC Higgs results a prominent example



- RooFit provides (almost) limitless model building facilities
  - Concept of persistable model workspace allows to separate model building and model interpretation
  - HistFactory package introduces structured model building for binned likelihood template models that are common in LHC analyses

- RooStats provide a wide set of statistica tests that can be performed on RooFit models
  - Bayesian, Frequentist and Likelihood-based test concepts
  - Wide range op options (Frequentist test statistics, Bayesian integration methods, asympotic calculators…)


ATLAS 2011 - 2012
√s = 7 TeV: ∫Ldt = 4.6-4.8 fb⁻¹
√s = 8 TeV: ∫Ldt = 5.8-5.9 fb⁻¹
± 1σ
± 2σ
Observed
Bkg. Expected
CLs Limits
95% CL Limit on μ
10⁻¹
mH [GeV]