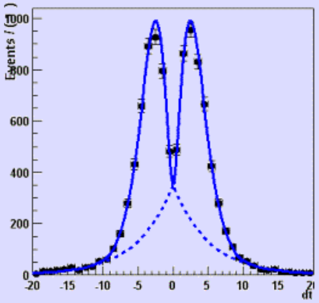
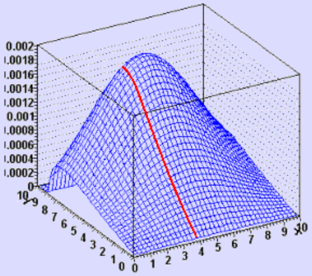
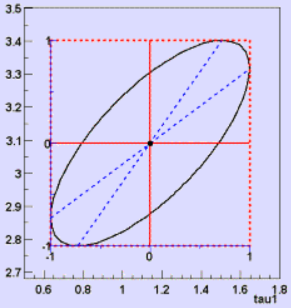






RooFit
A tool kit for data modeling in ROOT

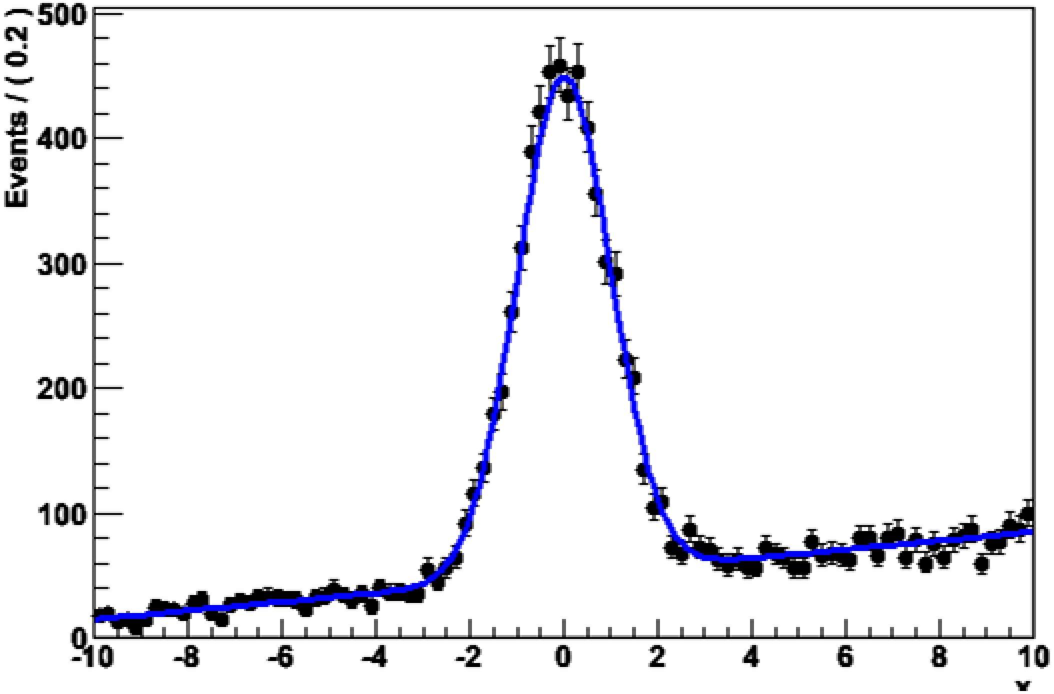
Wouter Verkerke (NIKHEF)

Introduction

1 What is RooFit ?

Focus: coding a probability density function

- Focus on one practical aspect of many data analysis in HEP: How do you formulate your p.d.f. in ROOT
  - For ‘simple’ problems (gauss, polynomial), ROOT built-in models well sufficient



  - But if you want to do unbinned ML fits, use non-trivial functions, or work with multidimensional functions you are quickly running into trouble

Why RooFit was developed

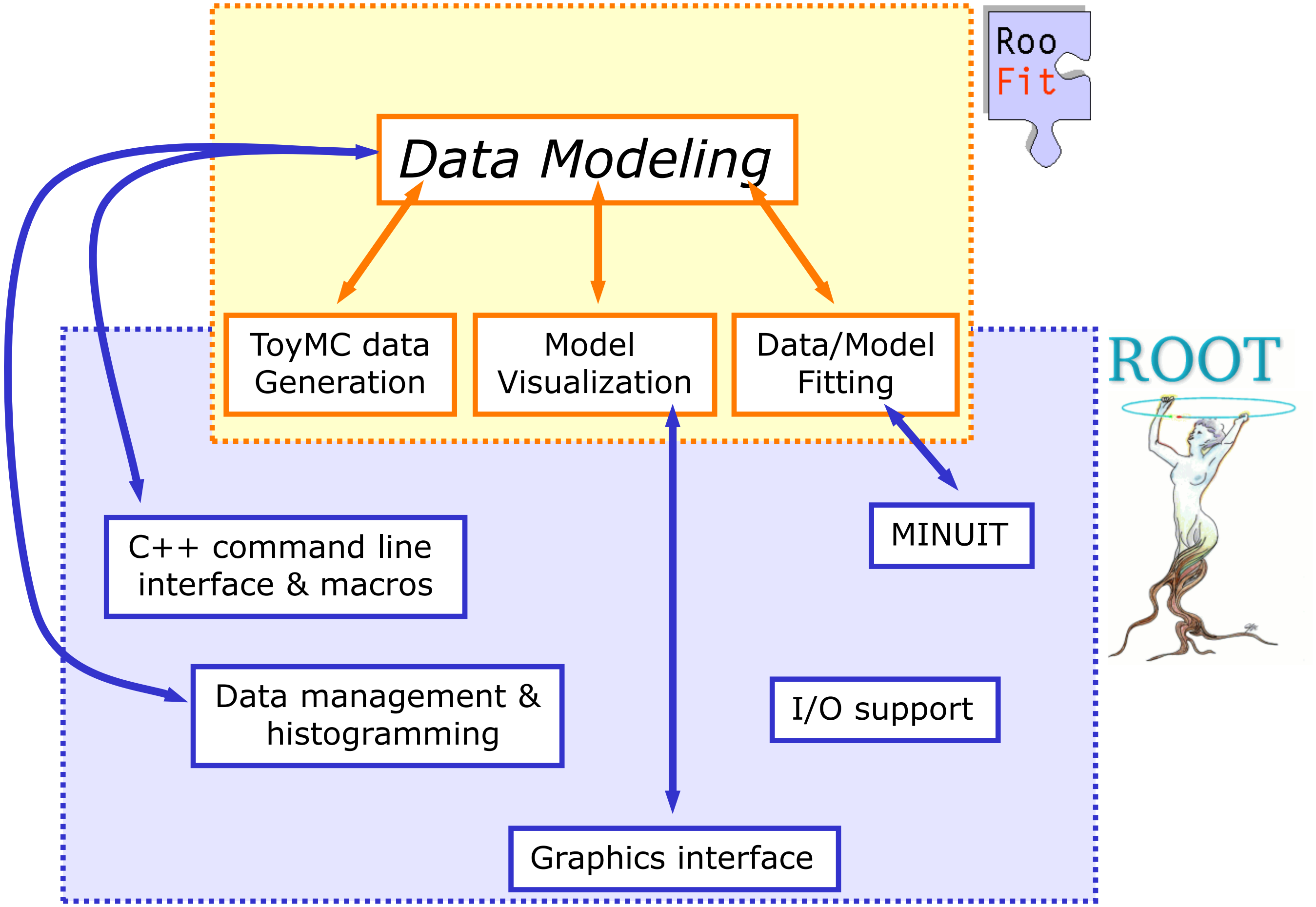
- BaBar experiment at SLAC: Extract sin(2β) from time_ dependent CP violation of B decay: e^+e^- → Y(4s) → $B\bar{B}$
  - Reconstruct both Bs, measure decay time difference
  - Physics of interest is in decay time dependent oscillation

$$f_{sig} \cdot \left[\mathrm{SigSel}(m; \bar{p}_{sig}) \cdot \left(\mathrm{SigDecay}(t; \vec{q}_{sig}, \sin(2\beta)) \otimes \mathrm{SigResol}(t \mid dt; \vec{r}_{sig}) \right) \right] + \\ (1 - f_{sig}) \left[\mathrm{BkgSel}(m; \bar{p}_{bkg}) \cdot \left(\mathrm{BkgDecay}(t; \vec{q}_{bkg}) \otimes \mathrm{BkgResol}(t \mid dt; \vec{r}_{bkg}) \right) \right]$$

- Many issues arise
  - Standard ROOT function framework clearly insufficient to handle such complicated functions → must develop new framework
  - Normalization of p.d.f. not always trivial to calculate → may need numeric integration techniques
  - Unbinned fit, >2 dimensions, many events → computation performance important → must try optimize code for acceptable performance
  - Simultaneous fit to control samples to account for detector performan

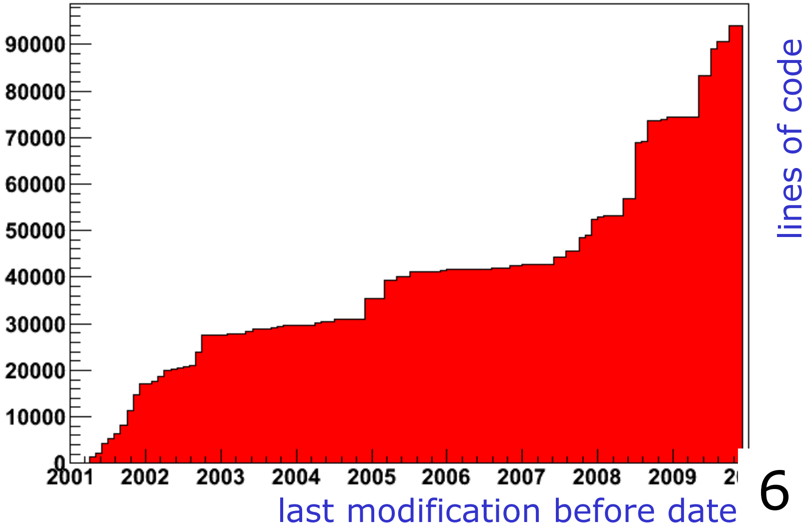
RooFit – a data modeling language for ROOT

Extension to ROOT – (Almost) no overlap with existing functionality

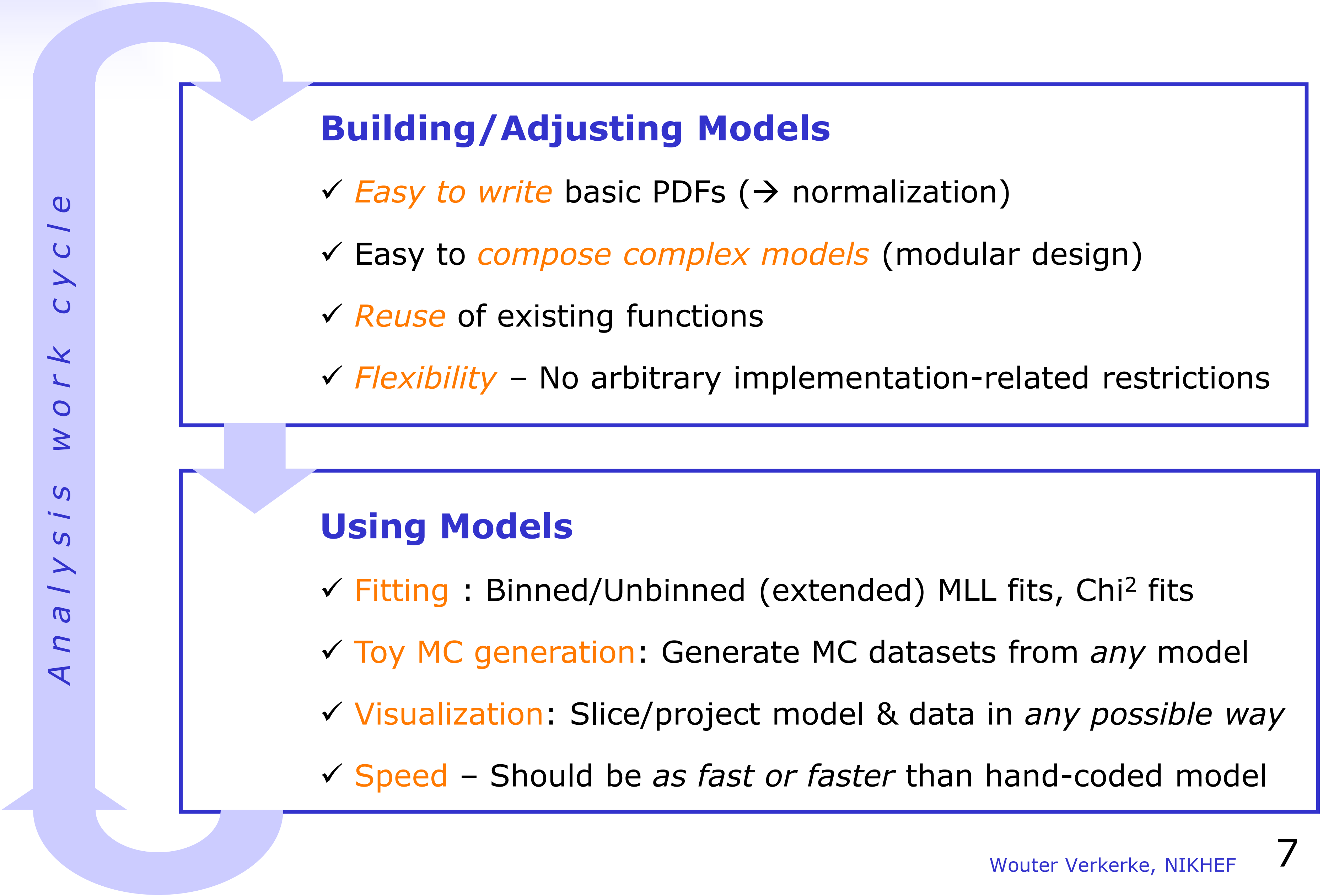


Project timeline

- 1999 : Project started
  - First application: ‘sin2b’ measurement of BaBar (model with 5 observables, 37 floating parameters, simultaneous fit to multiple CP and control channels)
- 2000 : Complete overhaul of design based on experience with sin2b fit
  - Very useful exercise: new design is still current design
- 2003 : Public release of RooFit with ROOT
- 2004 : Over 50 BaBar physics publications using RooFit
- 2007 : Integration of RooFit in ROOT CVS source
- 2008 : Upgrade in functionality as part of RooStats project
  - Improved analytical and numeric integration handling, improved toy MC generation, addition of workspace
- 2009 : Now ~100K lines of code
  - (For comparison RooStats proper is ~5000 lines of code)
- 2012 : Higgs discovery models formulated in RooFit using workspace concept



Data modeling - Desired functionality



Analysis work cycle

Building/Adjusting Models

- ✓ *Easy to write* basic PDFs (→ normalization)
- ✓ Easy to *compose complex models* (modular design)
- ✓ *Reuse* of existing functions
- ✓ *Flexibility* – No arbitrary implementation-related restrictions

Using Models

- ✓ Fitting : Binned/Unbinned (extended) MLL fits, Chi^2 fits
- ✓ Toy MC generation: Generate MC datasets from *any* model
- ✓ Visualization: Slice/project model & data in *any possible way*
- ✓ Speed – Should be *as fast or faster* than hand-coded model

Data modeling – OO representation

- Idea: represent math symbols as 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`**

  - Result: 1 line of code per symbol in a function (the C++ constructor) rather than 1 line of code per function

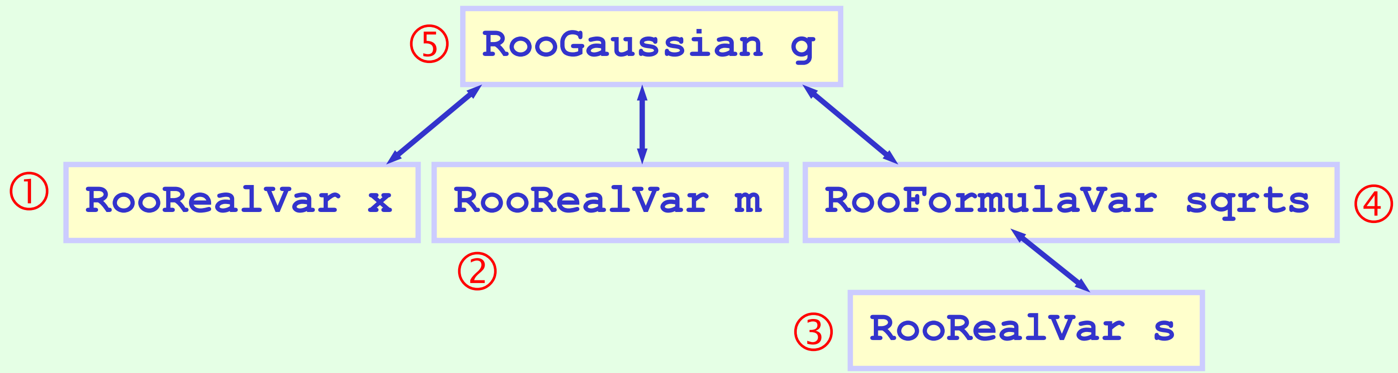
Data modeling – Constructing composite objects

- Straightforward correlation between mathematical representation of formula and RooFit code

Math

$$gauss(x, m, \sqrt{s})$$

RooFit diagram

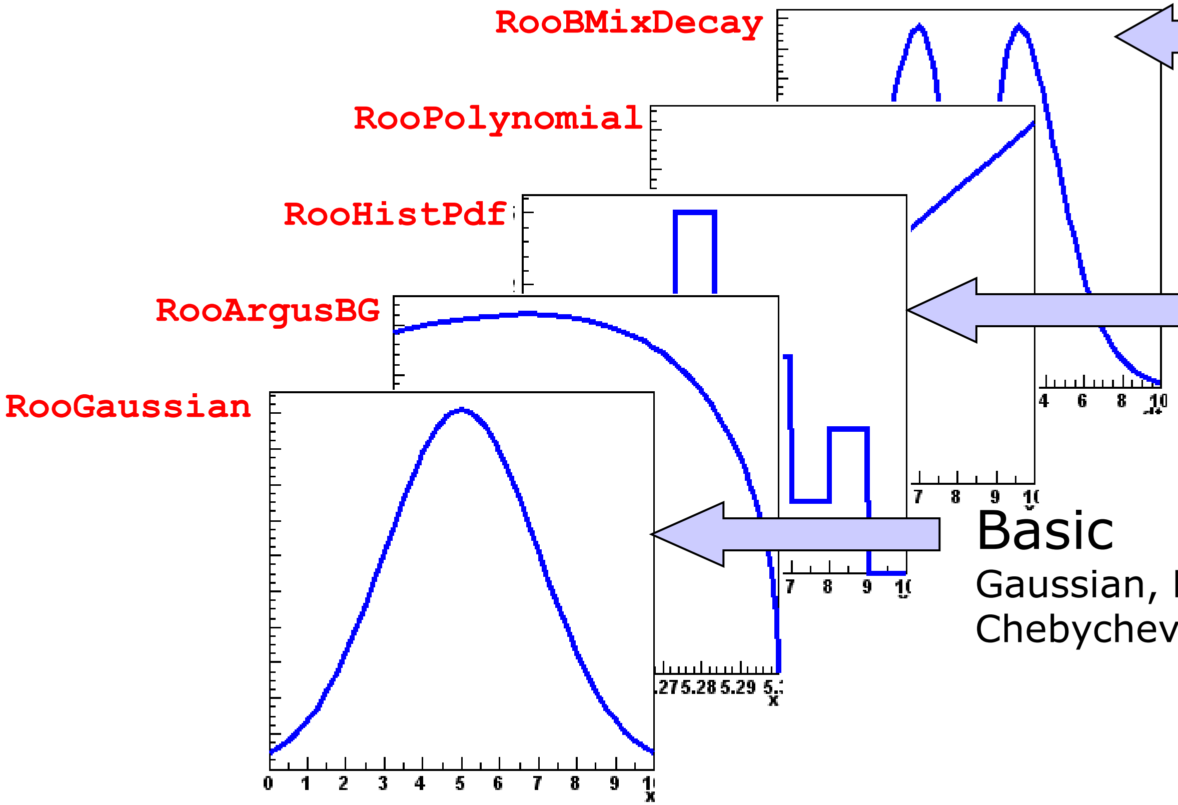


RooFit code

```
① RooRealVar x(“x”,”x”,-10,10) ;
② RooRealVar m(“m”,”mean”,0) ;
③ RooRealVar s(“s”,”sigma”,2,0,10) ;
④ RooFormulaVar sqrts(“sqrts”,”sqrt(s)”,s) ;
⑤ RooGaussian g(“g”,”gauss”,x,m,sqrts) ;
```

Model building – (Re)using standard components

- RooFit provides a collection of compiled standard PDF classes



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

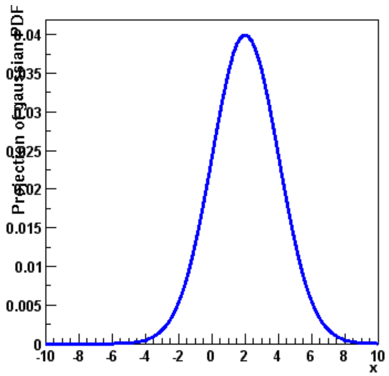
Non-parametric
Histogram, **KEYS**

Basic
Gaussian, Exponential, Polynomial,…
Chebychev polynomial

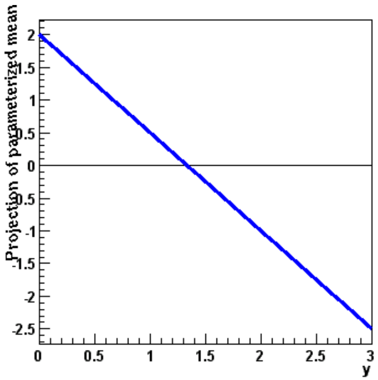
Easy to extend the library: each p.d.f. is a separate C++ class

Model building – (Re)using standard components

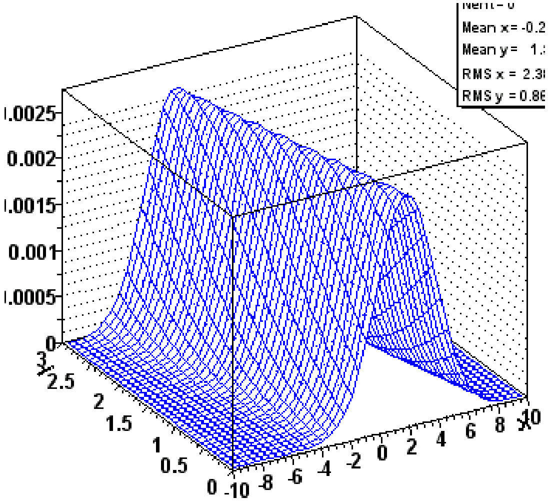
- Library p.d.f.s can be adjusted on the fly.
  - Just plug in *any function expression* you like as input variable
  - Works universally, even for classes you write yourself



$g(x;m,s)$



$m(y;a_0,a_1)$



```
RooPolyVar  m("m",y,RooArgList(a0,a1)) ;
RooGaussian g("g","gauss",x,m,s) ;
```

$g(x,y;a_0,a_1,s)$

- Maximum flexibility of library shapes keeps library small

Special pdfs – Kernel estimation model

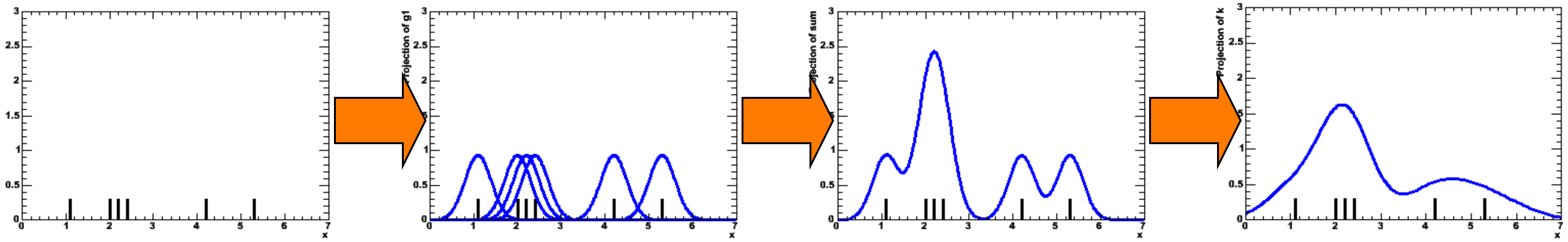
- Kernel estimation model
  - Construct smooth pdf from unbinned data, using kernel estimation technique

Sample of events

Gaussian pdf for each event

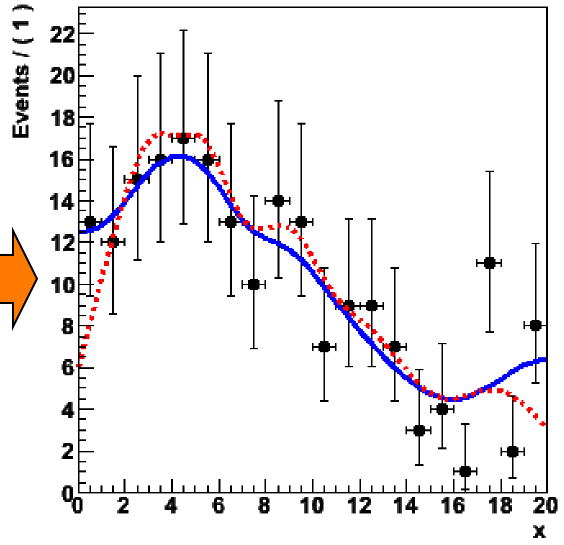
Summed pdf for all events

Adaptive Kernel: width of Gaussian depends on local event density



- Example

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

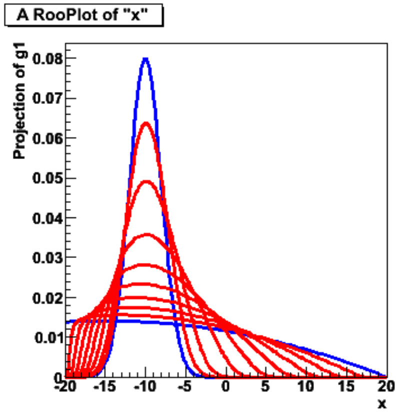


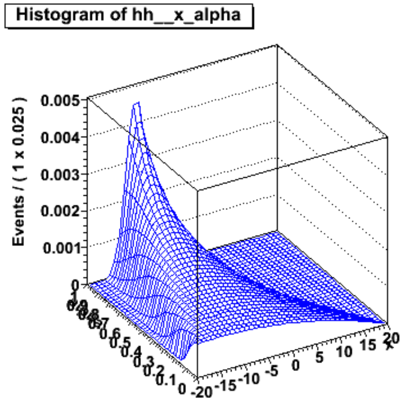
- Also available for n-D data

Special pdfs – Morphing interpolation

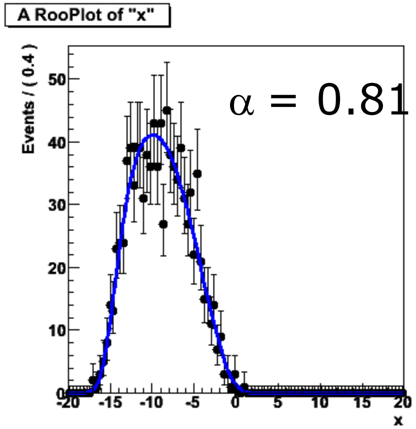
- Special operator pdfs can interpolate existing pdf shapes
  - Ex: interpolation between Gaussian and Polynomial

```
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



α = 0.812 ± 0.008

- Two morphing algorithms available
  - IntegralMorph (Alex Read algorithm).
    CPU intensive, but good with discontinuities
  - MomentMorph (Max Baak).
    Fast, can handling multiple observables (and soon multiple interpolation parameters), but doesn't work well for all pdfs

Handling of p.d.f normalization

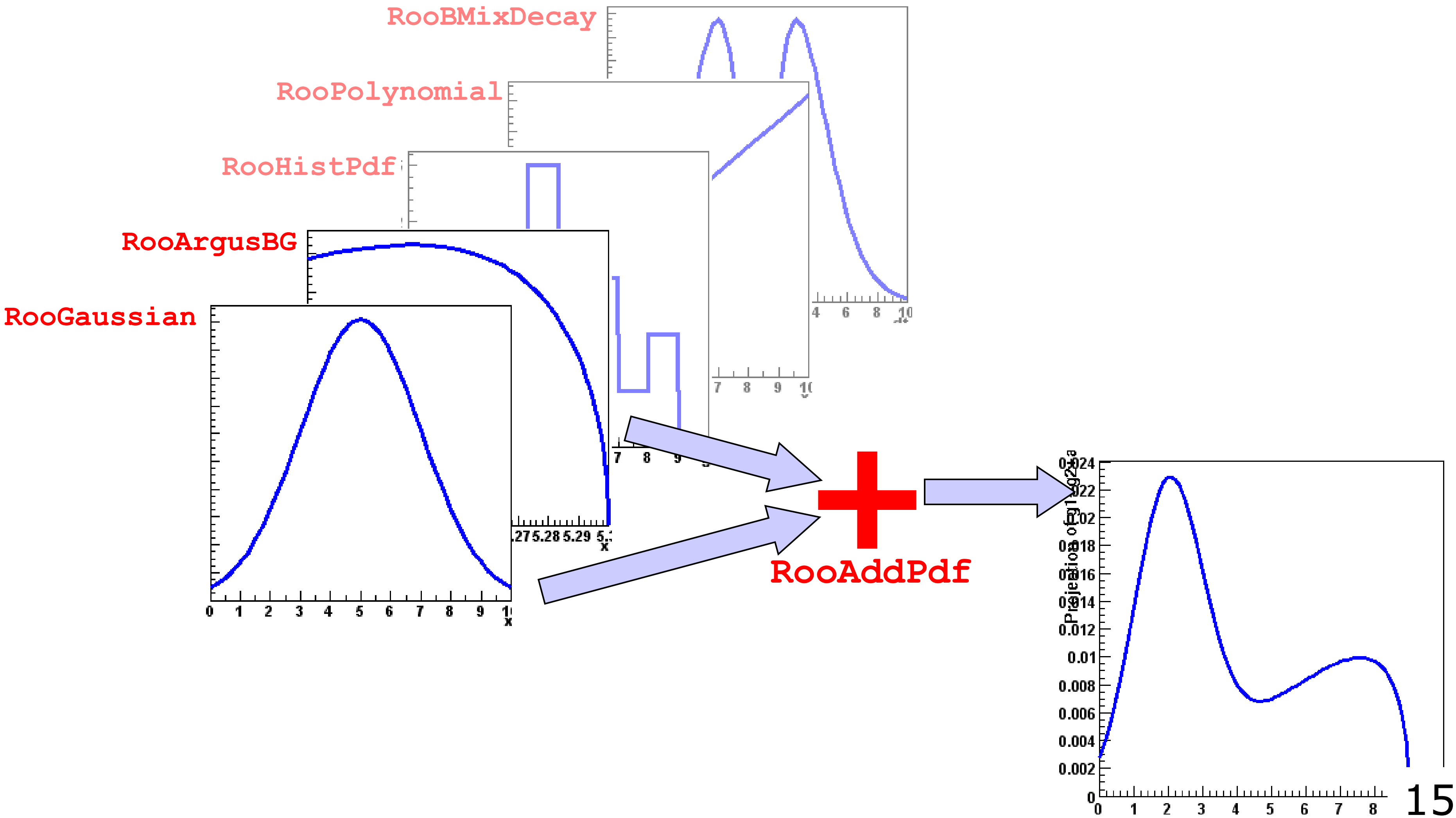
- Normalization of (component) p.d.f.s to unity is often a good part of the work of writing a p.d.f.

- RooFit handles most normalization issues transparently to the user
  - P.d.f can advertise (multiple) analytical expressions for integrals
  - When no analytical expression is provided, RooFit will automatically perform numeric integration to obtain normalization
  - More complicated that it seems: even if *gauss(x,m,s)* can be integrated analytically over x, *gauss(f(x),m,s)* cannot. Such use cases are automatically recognized.
  - Multi-dimensional integrals can be combination of numeric and p.d.f-provided analytical partial integrals

- Variety of numeric integration techniques is interfaced
  - Adaptive trapezoid, Gauss-Kronrod, VEGAS MC…
  - User can override parameters globally or per p.d.f. as necessary

Model building – (Re)using standard components

- Most physics models can be composed from ‘basic’ shapes



RooBMixDecay

RooPolynomial

RooHistPdf

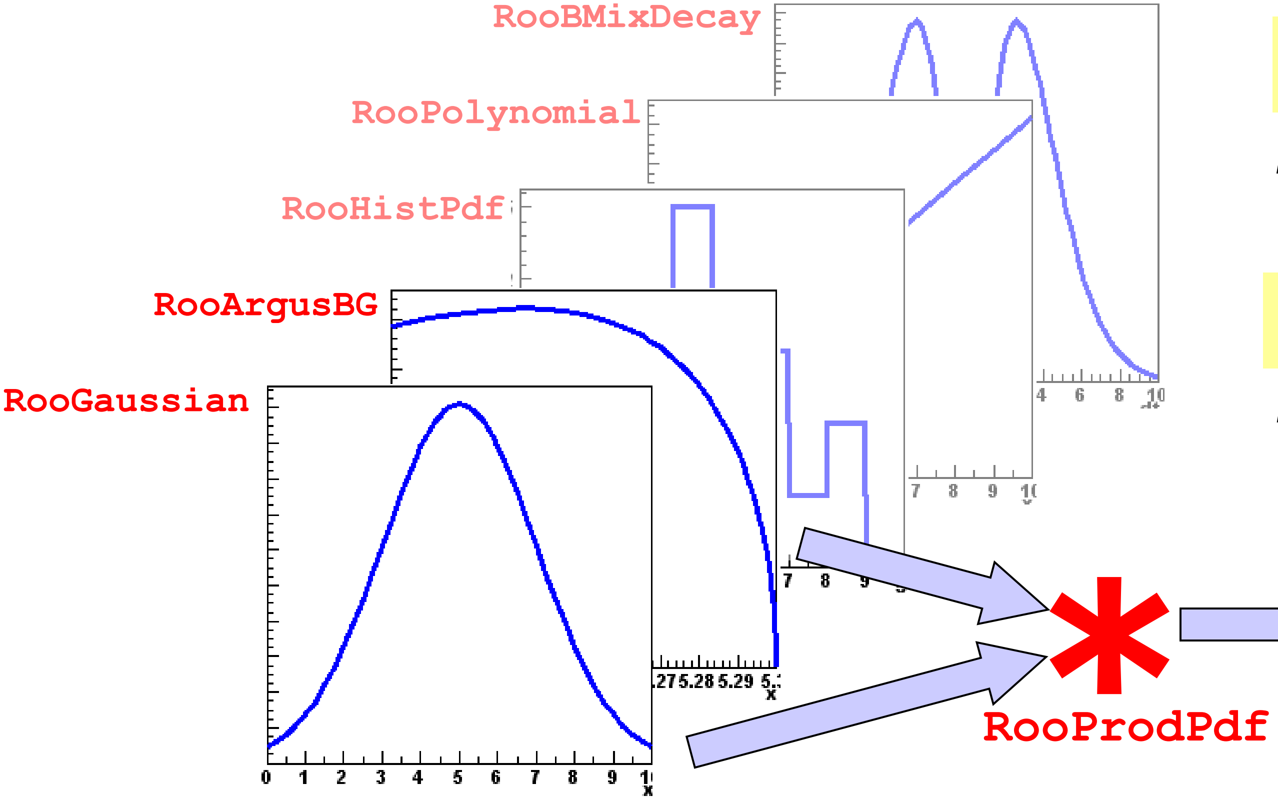
RooArgusBG

RooGaussian

RooAddPdf

Model building – (Re)using standard components

- Most physics models can be composed from ‘basic’ shapes

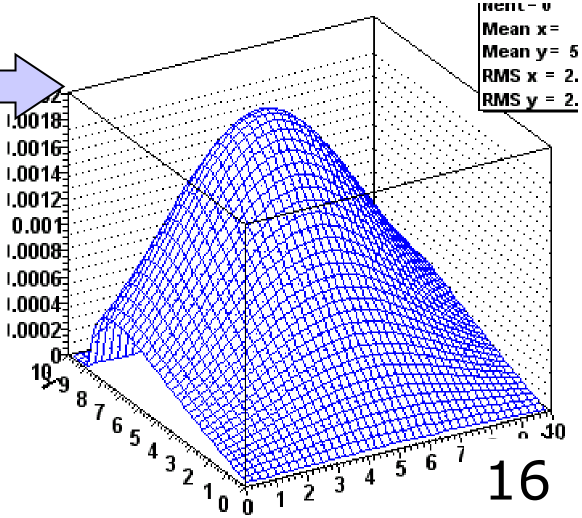


```
RooProdPdf h("h","h",
             RooArgSet(f,g))
```

$$h(x,y) = f(x)\cdot g(y)$$

```
RooProdPdf k("k","k",g,
           Conditional(f,x))
```

$$k(x,y) = f(x\,|\,y)\cdot g(y)$$



(FFT) Convolution – works for all models

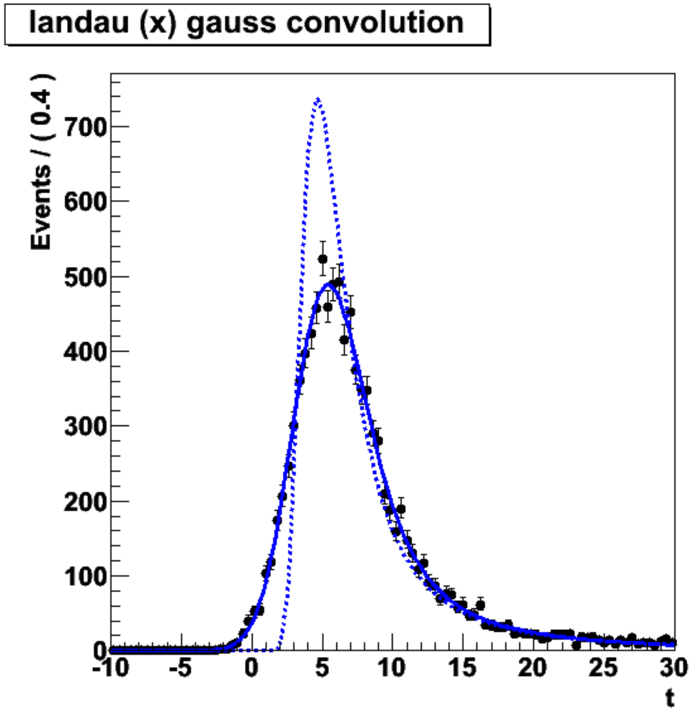
- Example

```
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

w.factory(“NCONV::LGb(x,L,G)”) ; // Numeric convolution
```

- FFT usually best
  - Fast: unbinned ML fit to 10K events take ~5 seconds

(x)

RooFFTConvPdf



Automated vs. hand-coded optimization of p.d.f.

- Automatic pre-fit PDF optimization
  - Prior to each fit, the PDF is analyzed for possible optimizations
  - Optimization algorithms:
    - Detection and *precalculation of constant terms* in any PDF expression
    - Function *caching* and lazy evaluation
    - *Factorization* of multi-dimensional problems where ever possible
  - Optimizations are always tailored to the specific use in each fit.
  - Possible because OO structure of p.d.f. allows automated analysis of structure

- **No need for users to hard-code optimizations**
  - Keeps your code understandable, maintainable and flexible without sacrificing performance
  - Optimization concepts implemented by RooFit are applied consistently and completely to all PDFs
  - Speedup of factor 3-10 reported in realistic complex fits

- Fit parallelization on multi-CPU hosts
  - Option for automatic parallelization of fit function on multi-CPU hosts (no explicit or implicit support from user PDFs needed)

2 Sharing models

Sharing data and functions

- Sharing data is in HEP is relatively easy – ROOT `TTrees`, `THx` histograms almost universal standard

- Sharing functions (likelihood / probability density) much more difficult
  - No standard protocol for defining p.d.f.s and likelihood functions
  - Structurally functions are much more complicated than data

- Essentially all methods start with the basic probability density function or likelihood function $L(x|\theta_r,\theta_s)$
  - Building a good model is the hard part!
  - want to re-use it for multiple methods
  - Language to common tools

- Common language for probability density functions and likelihood functions very desirable for easy exchange of information → RooFit

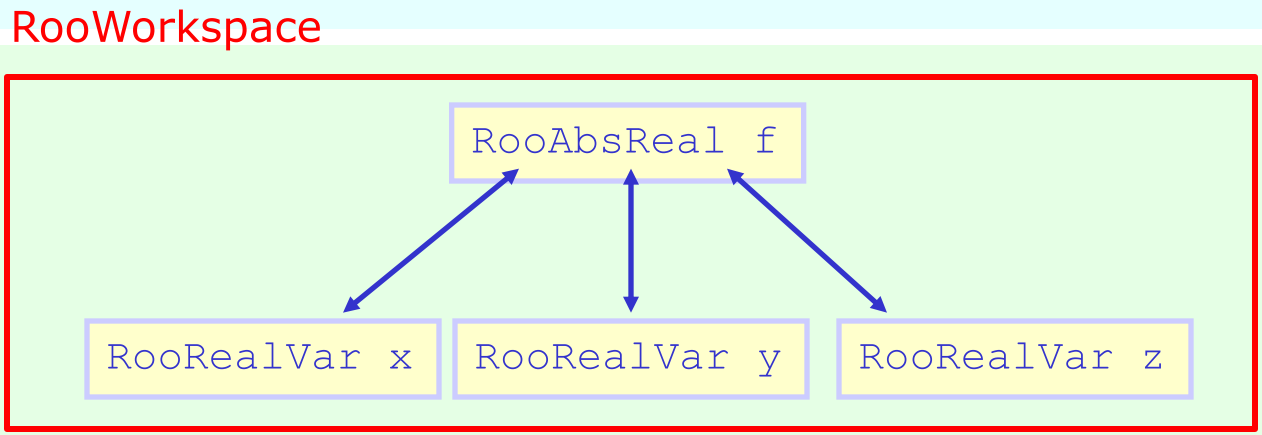
RooFit core design philosophy - Workspace

- The workspace serves a container class for all objects created

Math

$f(x,y,z)$

RooFit diagram



RooFit code

```
RooRealVar x(“x”,”x”,5) ;
RooRealVar y(“y”,”y”,5) ;
RooRealVar z(“z”,”z”,5) ;
RooBogusFunction f(“f”,”f”,x,y,z) ;
RooWorkspace w(“w”) ;
w.import(f) ;
```

Using the workspace

- Workspace
  - A generic container class for all RooFit objects of your project
  - Helps to organize analysis projects
- Creating a workspace

```
RooWorkspace w(“w”) ;
```

- Putting variables and function into a workspace
  - When importing a function or pdf, all its components (variables) are automatically imported too

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

  // imports f,x,mean and sigma
  w.import(myFunction) ;
```

Using the workspace

- Looking into a workspace

```
w.Print() ;

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

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

- Getting variables and functions out of a workspace

```
// Variety of accessors available
RooPlot* frame = w.var(“x”)->frame() ;
w.pdf(“f”)->plotOn(frame) ;
```

Persisting workspaces

- Workspaces can be trivially written to file

```
// Write workspace contents to file
w.writeToFile(“myworkspace.root”) ;
```

- And be read back into another ROOT session

```
// Open ROOT file
TFile* f = TFile::Open(“myworkspace.root”) ;

// Retrieve workspace
RooWorkspace* w = f->Get(“w”) ;
```

Sharing models using RooFit workspaces

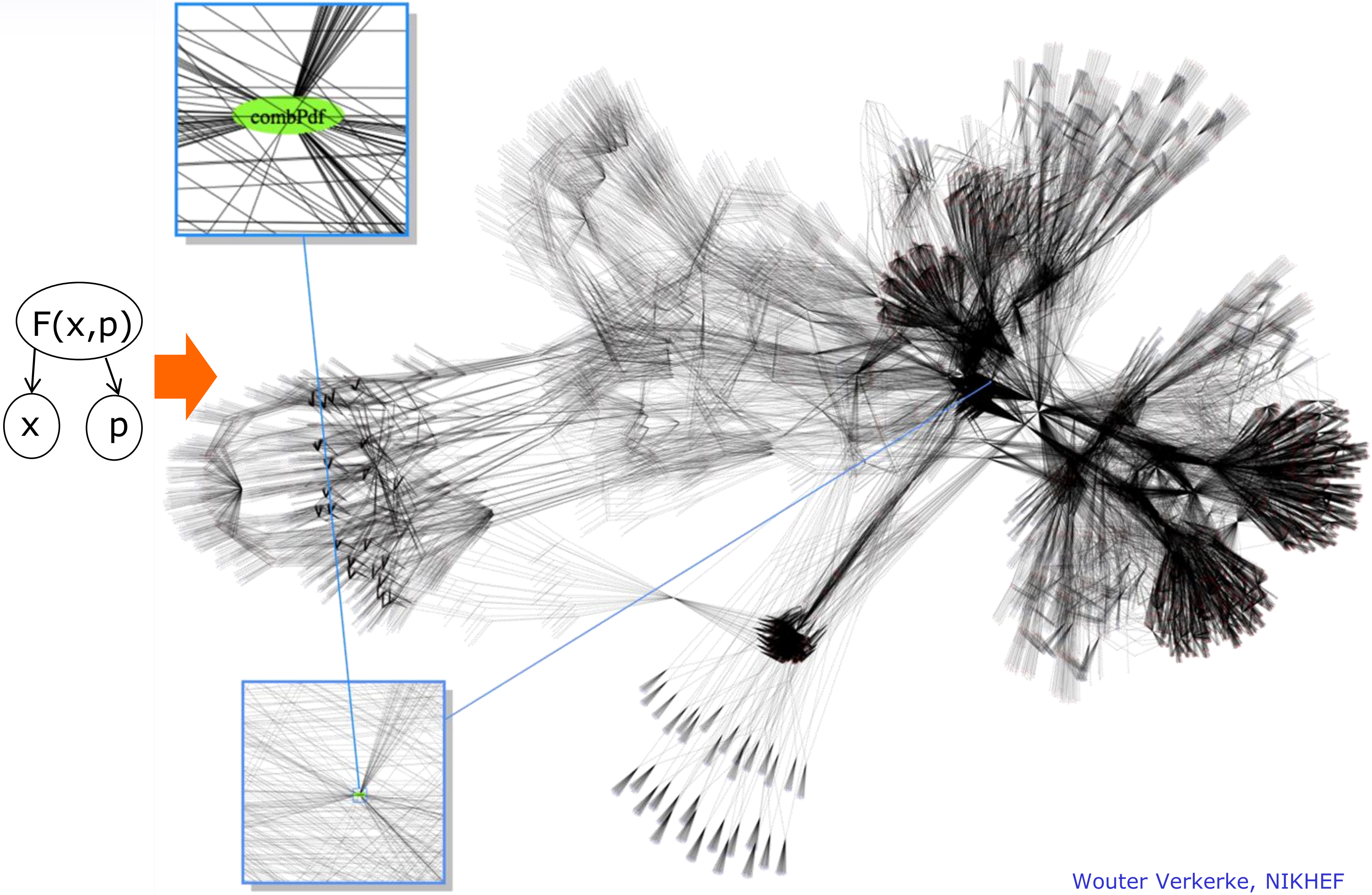
- Internal sharing of likelihood models between analysis groups has been common within Higgs effort
- Aided by RooFit workspace concept
- What you share is not only a description of model, but an actual implementation (a callable C++ function)
  - Virtually zero overhead in getting things to work

```
// Setup [4 lines ]
TFile* f = TFile::Open("myfile.root") ;
RooWorkspace* w = f->Get("mywspace") ;
RooAbsPdf* model = w->pdf("mymodel") ;
RooAbsData* data = w->data("obsData") ;

// Core business
model->fitTo(*data) ;
RooAbsReal* nll = model->createNLL(*data) ;
```

- Works for **any** model

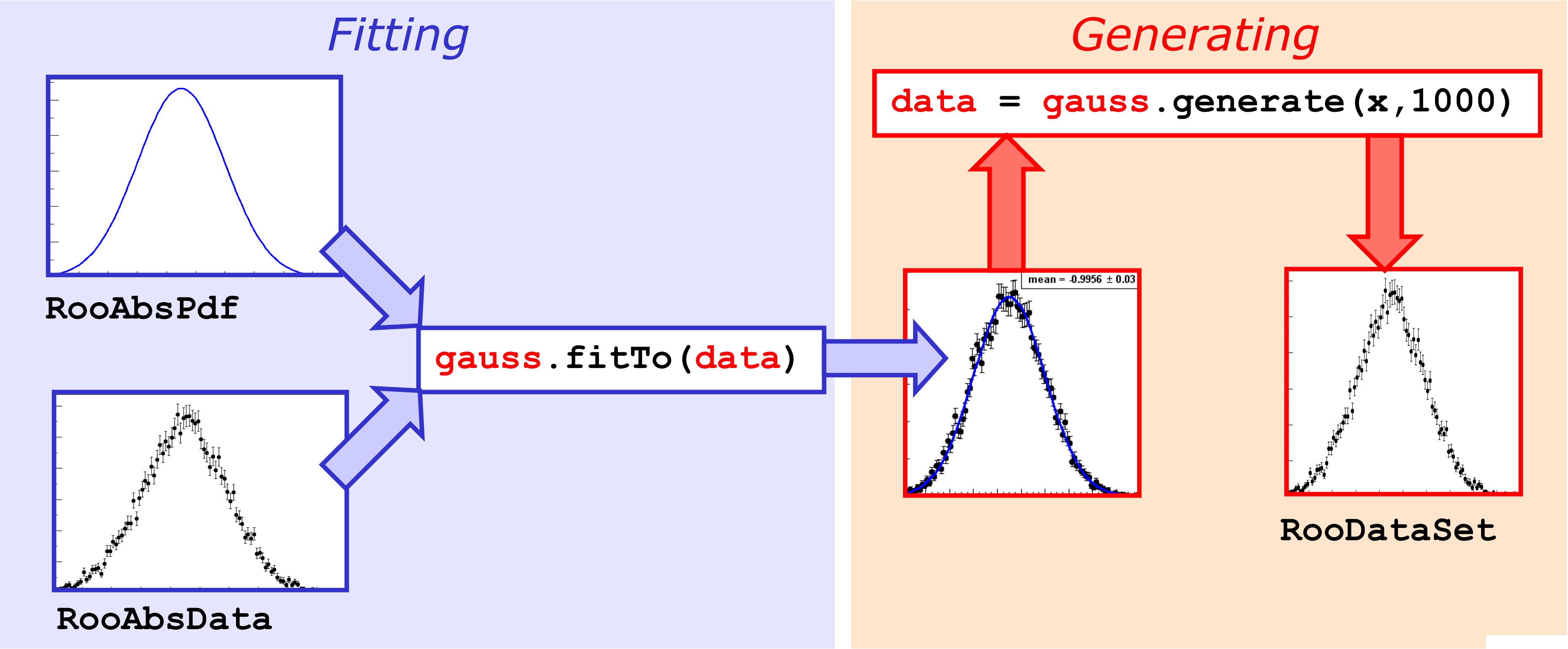
Scalability – an extreme example

A Higgs model (23.000 functions, 1600 parameters)



A workspace provides you with a model **implementation**

- *All* RooFit models provide *universal and complete fitting* and Toy Monte Carlo *generating* functionality
  - Model complexity only limited by available memory and CPU power
  - Fitting/plotting a 5-D model as easy as using a 1-D model
  - Most operations are one-liners



Fitting

RooAbsPdf

RooAbsData

```
gauss.fitTo(data)
```

Generating

```
data = gauss.generate(x,1000)
```

RooDataSet

Probability density function → Likelihood

- Easy to create a likelihood from a model and a dataset

```
// Create likelihood (calculation parallelized on 8 cores)
RooAbsReal* nll = w::model.createNLL(data,NumCPU(8)) ;
```

- Easy to manipulate with ROOT minimizers

```
RooMinuit m(*nll) ;   // Create MINUIT session
m.migrad() ;          // Call MIGRAD
m.hesse() ;           // Call HESSE
m.minos(w::param) ;   // Call MINOS for ‘param’

RooFitResult* r = m.save() ; // Save status (cov matrix etc)
```

- Can also insert likelihood function in a workspace

```
w.import(*nll) ; // for direct use by others
```

Working with profile likelihood

- A profile likelihood ratio

$$\lambda(p) = \frac{L(p,\hat{\hat{q}})}{L(\hat{p},\hat{q})}$$

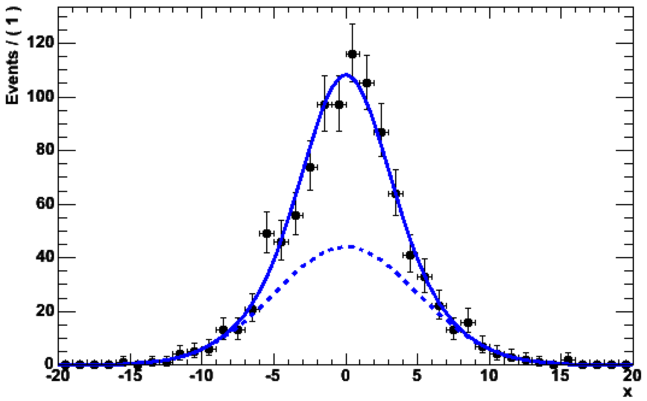
← Best L for given p

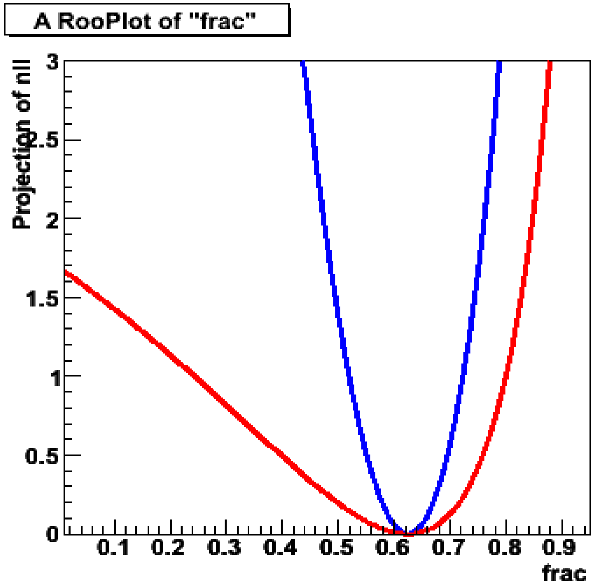
← Best L

can be represent by a regular RooFit function
(albeit an expensive one to evaluate)

```
RooAbsReal* ll = model.createNLL(data,NumCPU(8)) ;
RooAbsReal* pll = ll->createProfile(params) ;
```

```
RooPlot* frame = w::frac.frame() ;
nll->plotOn(frame,ShiftToZero()) ;
pll->plotOn(frame,LineColor(kRed)) ;
```





(Not) sharing the (unbinned) data

- Potential discussion item when sharing workspaces is that you share not only the model, but also the (unbinned) data – which a collaboration for various reason may not want to make public
- No easy iron-clad solutions to this issue – likelihood must have access to the data
- One simple solution currently provided are ‘sealed’ likelihood functions in workspace → These refuse access to internal data.
  - Not iron-clad since a good programmer with a debugger can still extract this
  - But sealed likelihoods also offer opportunity to include ‘copyright’ message – printed whenever workspace with sealed likelihoods is loaded into memory

```
nll->seal(“your copyright message goes here”) ;
w->import(*nll) ;
```

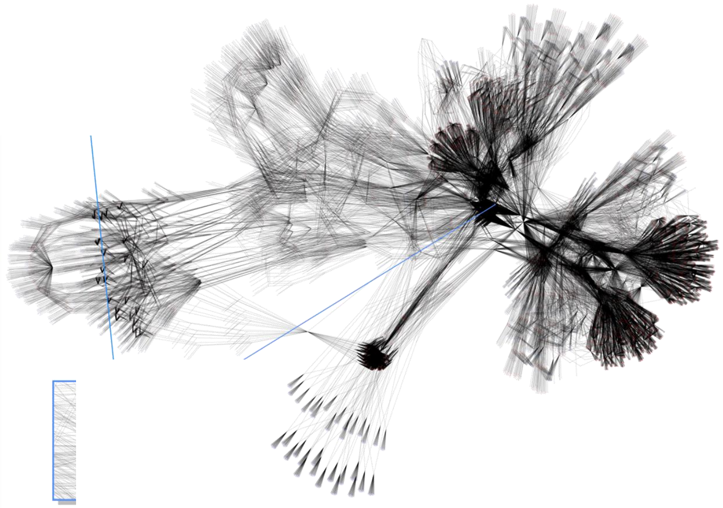
Interfacing RooFit functions to other code

- Binding exist to represent RooFit likelihood and probability density functions as ‘simple’ C++ functions

```
// RooFit pdf object
RooAbsPdf* pdf ;

// Binding object
RooFunctor lfunc(*pdf,observables,parameters) ;

// Evaluate pdf through binding
// takes variables as array of doubles
double obs[n] ;
double par[m] ;
double val = lfunc.eval(obs,par) ;
```

Summary

- RooFit is a object-oriented data modeling language for HEP, part of ROOT distribution
  - Key concept is representing mathematical entities as C++ objects
- Extensively used since nearly 13 years, highly scalable with good performance
- Ability to persist these models into 'workspaces' in ROOT files allows to trivially share **implementations** of models
  - You read and use parametric likelihoods from other scientists with almost zero effort
  - Very effectively used in Higgs discovery effort



- Long-term retention ability of workspaces explicit goal
  - ROOT schema evolution framework provides tools to guarantee backward compatibility for reading existing workspaces