RooFitTools

A general purpose tool kit for data modeling, developed in BaBar

Wouter Verkerke (UC Santa Barbara)
David Kirkby (Stanford University)

What is RooFitTools?

- A data modeling toolkit for
  - (Un)binned maximum likelihood fits
  - Toy Monte Carlo generation
  - Generating plots/tables

- RooFitTools is a class library built on top of the ROOT interactive C++ environment
  - Key concepts
    - **Datasets**
    - **Variables and generic functions**
    - **Probability density functions**
  - A fit/toyMC is setup in a ROOT C++ macro using the building blocks of the RooFitTools class library
  - TFitter/TMinuit used for actual fitting

Key concepts: a simple fitting example: Gauss+Exp

```
void intro() {
RooRealVar //define the data variables and fit model parameters
        m("m"      ,"Reconstructed Mass", 0.5  ,2.5, "GeV"),
    rmass("rmass"  ,"Resonance Mass"    , 1.5  ,1.4, 1.6, "GeV"),
    width("width"  ,"Resonance Width"   , 0.15 ,0.1, 0.2, "GeV"),
  bgshape("bgshape","Background shape"  ,-1.0 ,-2.0, 0.0),
     frac("frac"   ,"Signal fraction"   , 0.5  ,0.0, 1.0) ;

// Create the fit model components: Gaussian and exponential PDFs
RooGaussian signal("signal","Signal Distribution",m,rmass,width);
RooExponential bg("bg","Background distribution",m,bgshape) ;

// Combine them using addition (with relative fraction parameter)
RooAddPdf model("model","Signal + Background",signal,bg,frac) ;

// Read the values of 'm' from a text file
RooDataSet* data = RooDataSet::read("mvalues.dat",m) ;

// fit the data to the model with an UML fit
model.fitTo(data) ;
```

Variables

Description, unit,
fit and plot ranges,
constant/floating
status stored in object

Probability density functions

Explicitly self-normalized

Dataset

Derived from Ttree

Maps a TTree row onto a set of RFT variable objects

Plotting and Generating ToyMC

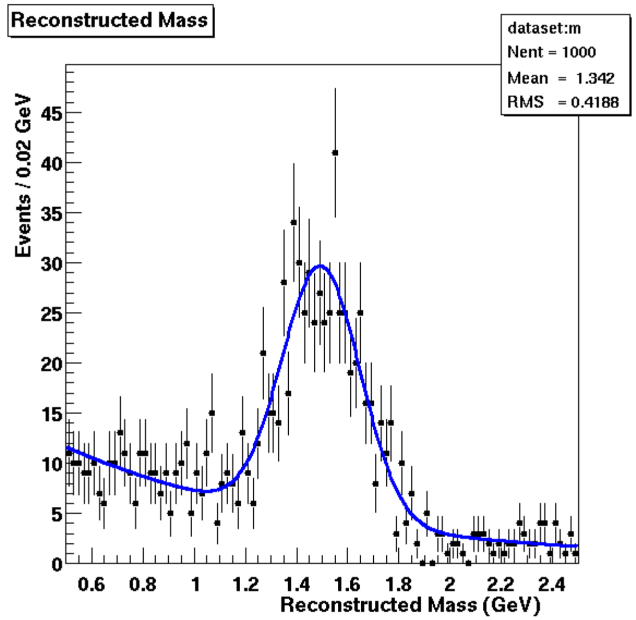
- Plotting

```
// create empty 1-D plot frame for "m"
RooPlot* frame = m.frame() ;

// plot distribution of m in data on frame
data->plotOn(frame) ;

// plot model as function of m
model->plotOn(frame) ;

// Draw the plot on a canvas
frame->Draw() ;
```

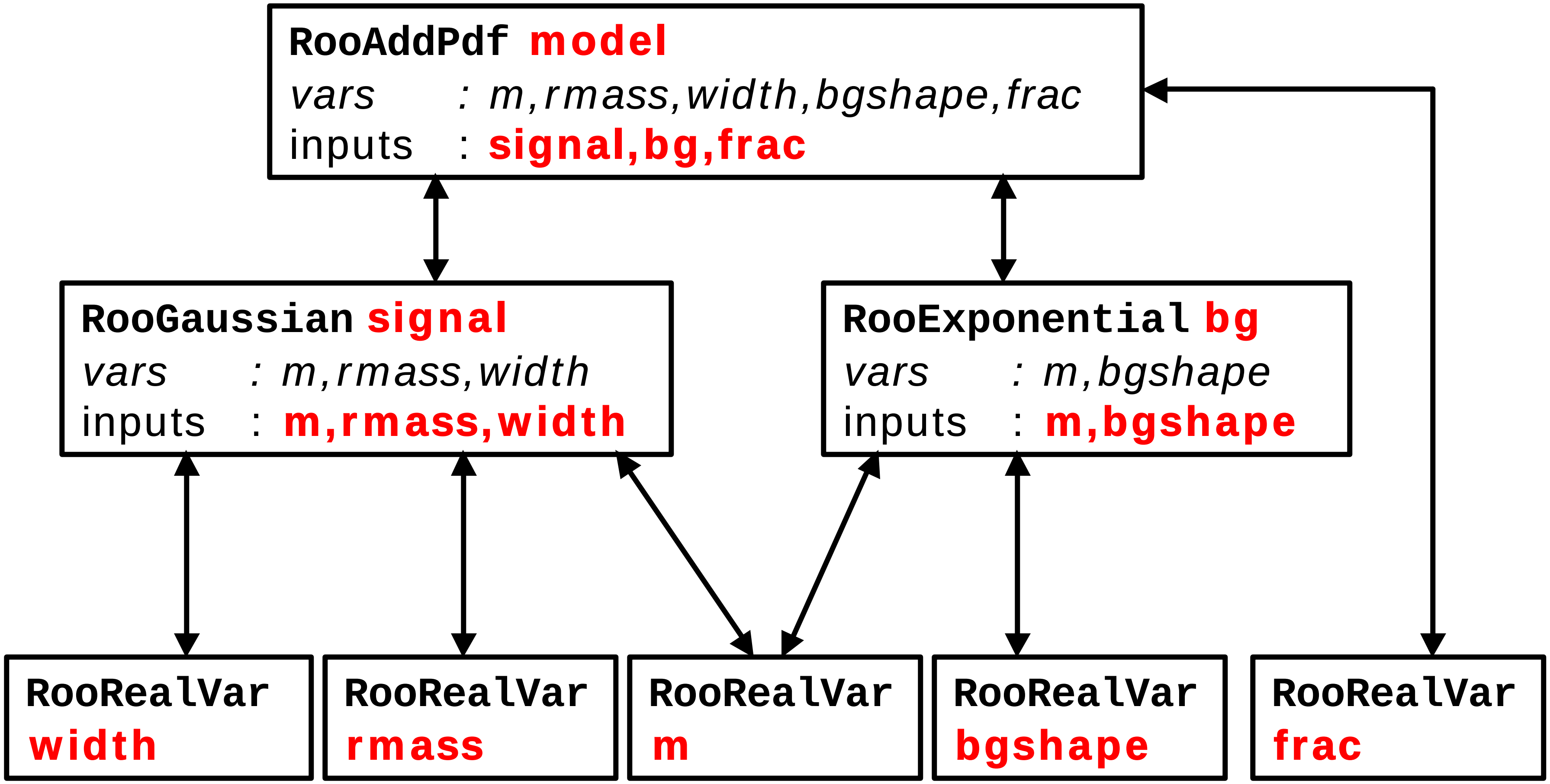


  - A **RooPlot** frame collects multiple histograms, curves, text boxes.
    - Persistable object, I.e. can save complex multi-layer plots in batch fit/generation
  - Automatic adaptive binning for function curves: always smooth functions regardless of data histogram binning
  - Poisson/binomial errors on histograms (automatically selected)

- Generating
  - Works for real and discrete variables

```
RooDataSet* toyMCData = model.generate(varList,numEvents) ;
RooDataSet* toyMCData = model.generate(varList,protoData) ;
```

Object structure of example PDF



Generic functions and composition

- A more complex PDF:
  - Replace *gaussian(m,mean,width)* ->
    gaussian(m,mean,$w_{off}+w_{slope}$**alpha)*
  - Need object to represent function $w_{off}+w_{slope}*alpha$
- Class **RooFormulaVar** implements expression based functions
  - Based on **Tformula**, very practical for such transformations

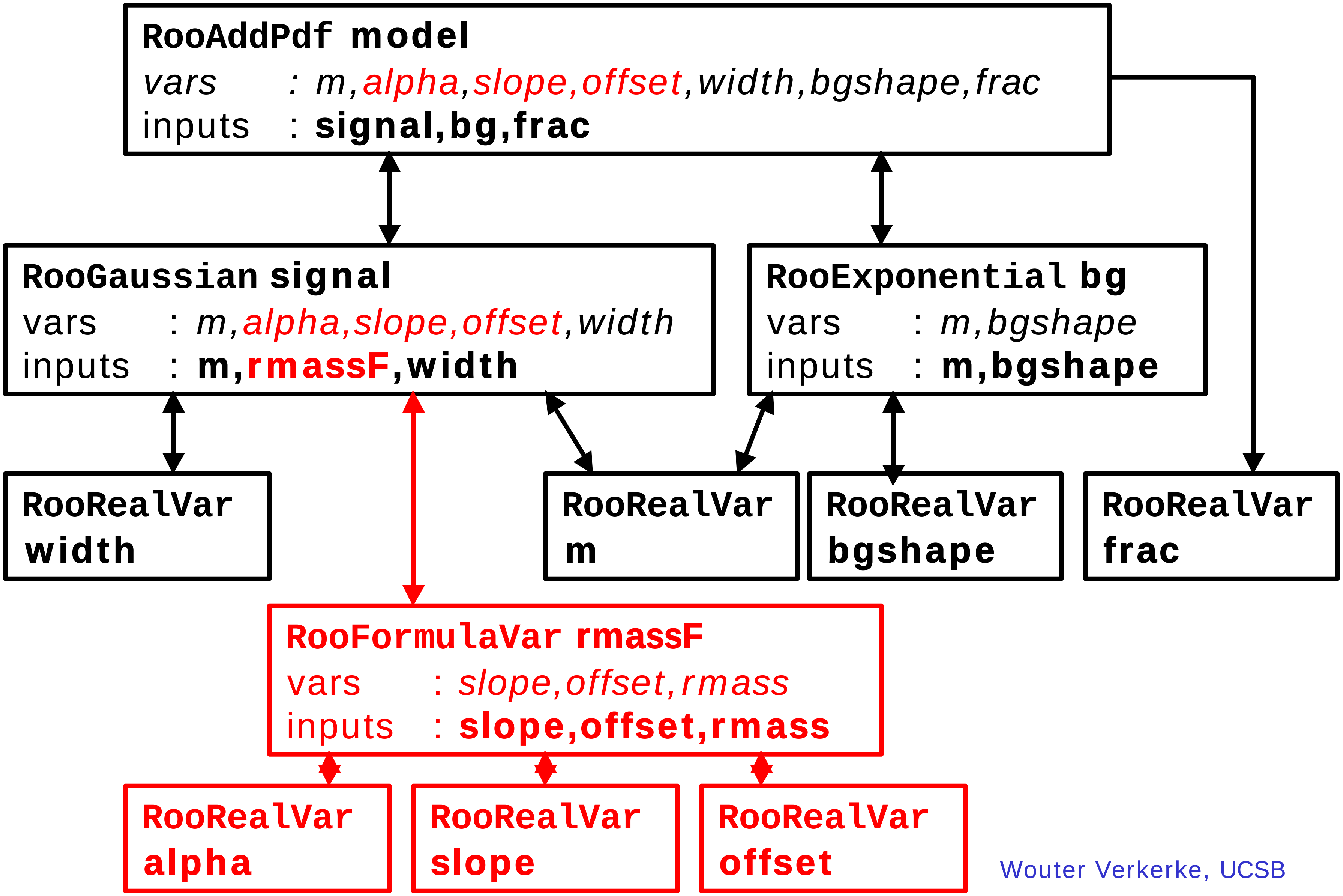
```
RooRealVar     alpha("alpha", "Mystery parameter", 1.5  ,1.4, 1.6, "GeV"),
               slope("slope", "Slope of resonance width" , 0.3  ,0.1 ,0.5),
              offset("offset","Offset of resonance width", 0.0  ,0.0 ,0.5, "GeV") ;

// Construct width object as function of alpha
RooFormulaVar rmassF("rmassF","offset+slope*rmass",
                     RooArgSet(slope,offset,rmass))


// Plug rmassF function in place of rmass variable
RooGaussian   signal("signal","Signal distribution", m,rmassF,width) ;
```

- Example of composition:
  - PDF **signal** now has 3 extra variables: **offset,slope,alpha**
  - Every variable of a PDF can be can be a function of other variables

Extend PDF: rmass → rmass(alpha) = offset+slope*alpha



Wouter Verkerke, UCSB

Discrete variables

- Often a data model includes discrete variables such as particle ID, decay mode, CP eigenvalue etc.
- Can be represented by **`RooCategory`**:
  - Finite set of labeled states, numeric code optional

```
RooCategory decay("decay","Decay Mode") ; // A category variable
decay.defineType("B0 -> J/psi KS",0) ;  // Type definition with explicit index
decay.defineType("B0 -> J/psi KL") ; // Type definition with automatic index
decay.defineType("B0 -> psi(2s) KS") ;


// Assignment to other RooCategory, string or integer
decay = 0 ;    decay = "B0 -> J/psi KS"      ; decay = otherDecay
```

- Various transformation classes available, e.g.
  - RooMappedCategory: Pattern matching based category-to-category mapping

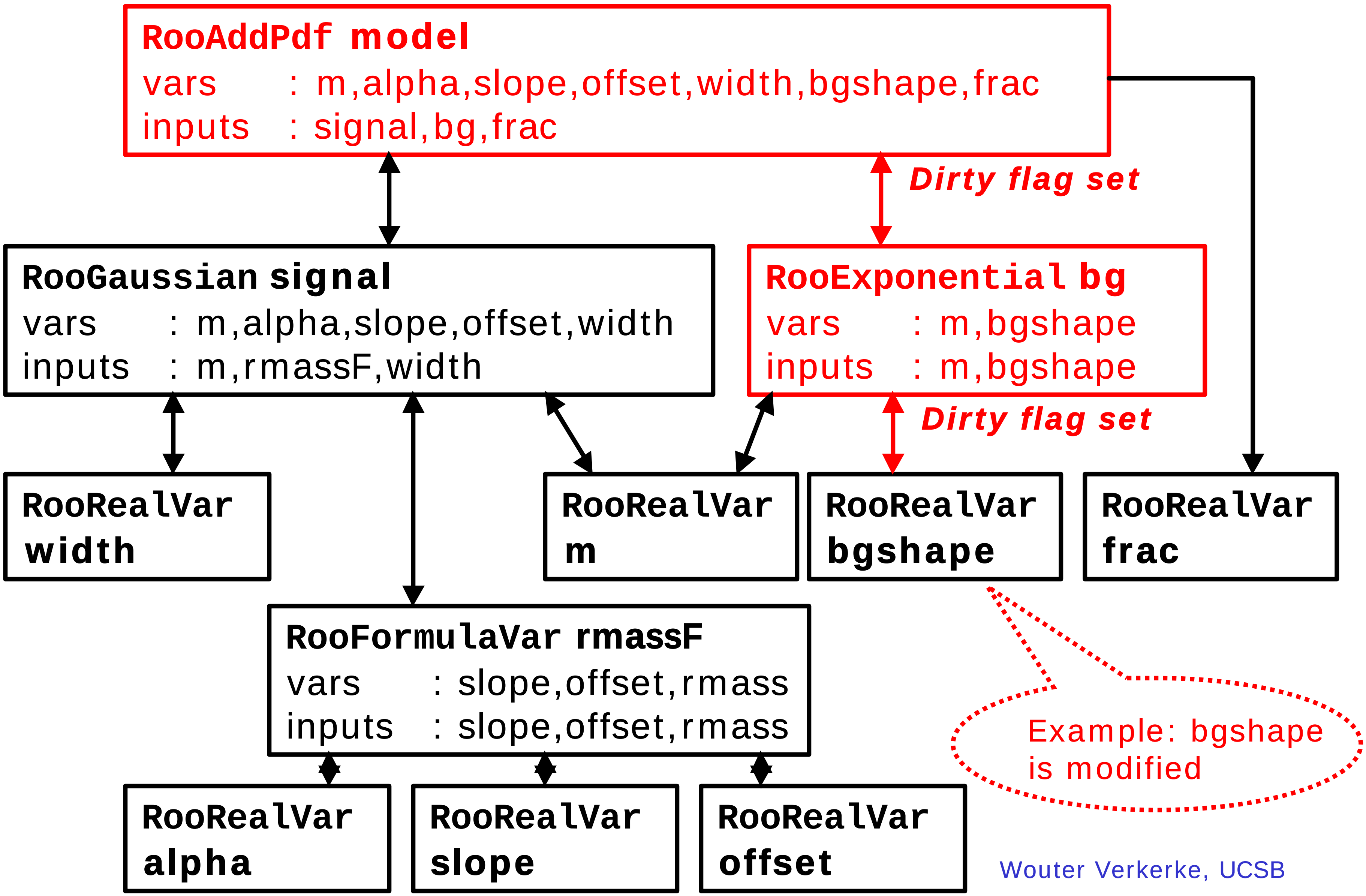
```
RooMappedCategory decayCP("decayCP",decay,"CPunknown") ; // A derived category
decayCP.map("*KS*","CPminus") ;  // Wildcard mapping "*KS*" -> "CPminus"
decayCP.map("*KL*","CPplus") ;
```

Under the hood: Integration & Optimization

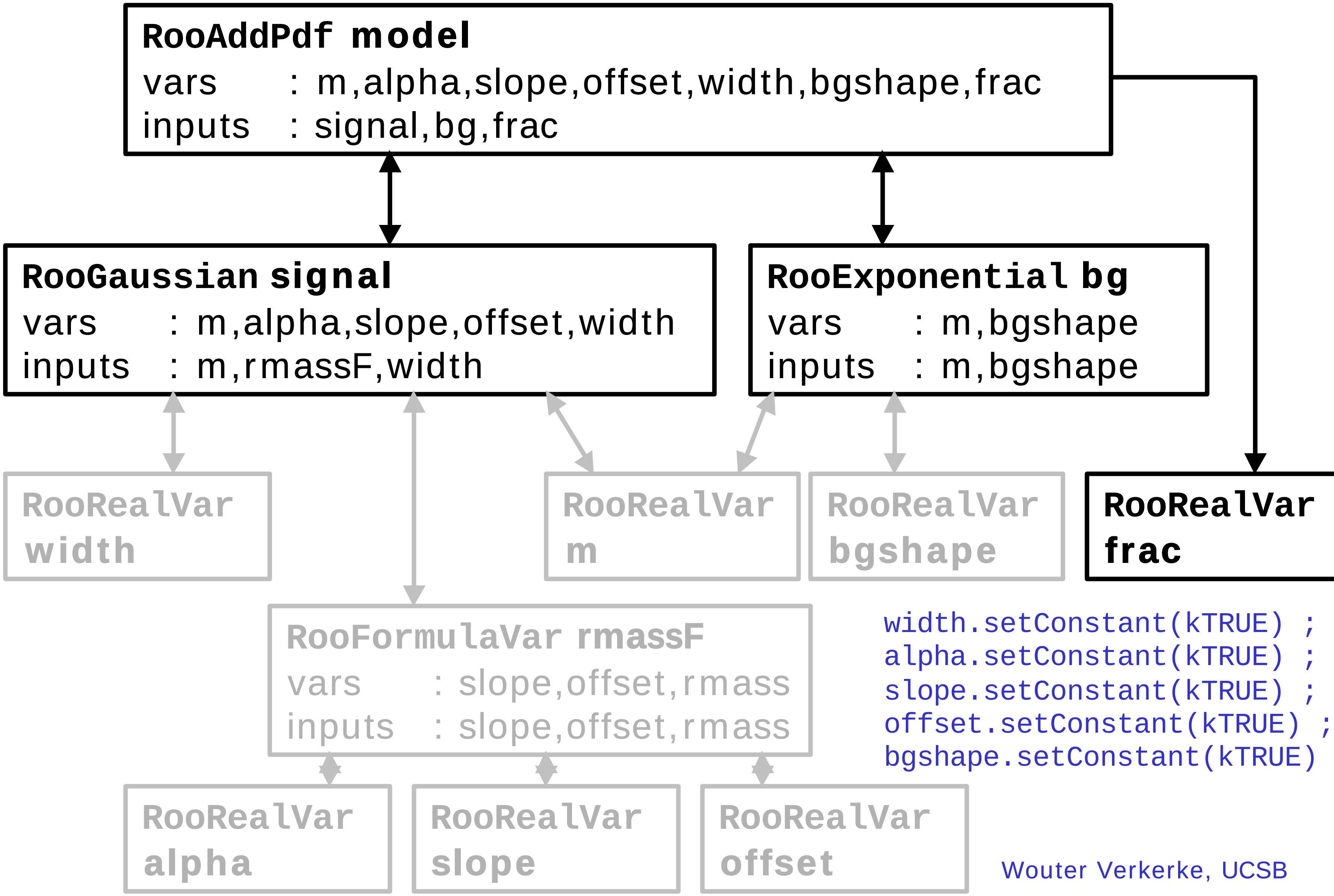
PDF evaluation/normalization speed critical for complex unbinned likelihood fits.
RooFitTools implements several strategies to maximize performance

- Caching & lazy evaluation
  - The output value of all function objects is cached
  - Function value only recalculated if any of the input objects change.
  - Push/pull model:
    - All RFT objects have links to their client and server objects
    - If an object changes value, it pushes a 'dirty' flag to all its registered clients.
    - Clients postpone recalculation to next getVal() call, checking the dirty flag at that point.
- Precalculation of 'constant' functions.
  - If a PDF exclusively depends on
    - variables in the fitted data set
    - constant non-dataset parameters
  - it is precalculated for each dataset row. (Limits calculation to 1 fit iteration)
- Hybrid analytical/numerical integration.
  - PDFs advertises which (partial) analytical integration it can perform
  - Dedicated **`RooRealIntegral`** object, owned by each PDF coordinates maximum possible analytical integration/summation for given configuration.
  - PDF normalization stored/cached separately from PDF value
    - Dependencies of PDF integral and value are different

Lazy evaluation: dirty state propagation



Constant node precalculation example: fit m for signal fraction only




```
width.setConstant(kTRUE) ;
alpha.setConstant(kTRUE) ;
slope.setConstant(kTRUE) ;
offset.setConstant(kTRUE) ;
bgshape.setConstant(kTRUE) ;
```

Wouter Verkerke, UCSB

Writing your own PDF

- Easiest way: use **RooGenericPdf**

```
RooGenericPdfmyPDF("myPDF","exp(abs(x)/sigma)*sqrt(scale)",
                                  RooArgSet(x,sigma,scale)) ;
```

  - No programming required
  - Like **RooFormulaVar**, based on **TFormula**.
  - Automatic normalization via full numerical integration.

- Write your own **RooAbsPdf** derived class
  - Faster execution, allows (partial) analytical normalization.
  - Optimization technology requires PDFs to be 'good objects', i.e. well defined copy/clone behaviour.
    - Gory details of link management well hidden in RooAbsPdf and proxy classes.
  - Minimum implementation consists of 3 functions
    - Constructor/Copy constructor
    - evaluate() function – *Returns function value*
  - Extended implementation with analytical integration needs also
    - getAnalyticalIntegral() – *Indicates which (partial) integrals can be performed*
    - analyticalIntegral() - *Implements advertised (partial) integrals*

RooGaussian: minimum impl.

```
// Constructor
RooGaussian::RooGaussian(const char *name,
   const char *title, RooAbsReal& _x,
   RooAbsReal& _mean, RooAbsReal& _sigma) :

   RooAbsPdf(name,title),
   x("x","Dependent",this,_x),
   mean("mean","Mean",this,_mean),
   sigma("sigma","Width",this,_sigma)
{
}
```

Special proxy class holds object references, implement client/server link management Behaves like 'Double_t' to user

```
// Copy constructor
RooGaussian::RooGaussian(const
   RooGaussian& other, const char* name) :

   RooAbsPdf(other,name),
   x("x",this,other.x),
   mean("mean",this,other.mean),
   sigma("sigma",this,other.sigma)
{
}

// Implementation of value calculation
Double_t RooGaussian::evaluate(
             const RooDataSet* dset) const
{
  Double_t arg= x - mean;
  return exp(-0.5*arg*arg/(sigma*sigma)) ;
}
```

Optional integration support

```
// Advertise which partial analytical
   integrals are supported
Int_t
RooGaussian::getAnalyticalIntegral(
       RooArgSet& allV,
       RooArgSet& anaV) const
{
  if (matchArgs(allV,anaV,x)) return 1 ;
  return 0 ;
}

// Implement advertised analytical integrals
Double_t
RooGaussian::analyticalIntegral(Int_t code)
{
  switch(code) {
  case 0: return getVal() ;
  case 1: // integral over x
    {
      static Double_t root2 = sqrt(2) ;
      static Double_t rootPiBy2 =
                 sqrt(atan2(0.0,-1.0)/2.0);

      Double_t xscale = root2*sigma;
      return rootPiBy2*sigma*
             (erf((x.max()-mean)/xscale)-
              erf((x.min()-mean)/xscale));
    }
  default: assert(0) ;
  }
}
```

Present use of RooFitTools in BaBar

- Most analyses are using RooFitTools for their unbinned maximum likelihood fits, including complex fits like
  - CP analysis ($\sin 2\beta$)
  - Hadronic, semileptonic and dilepton lifetime & mixing analysis (τ,Δm_d)
  - Charmless 2-body decay ($\rightarrow \sin 2\alpha$)
- Example of fit complexity:
  - the composite PDF of the CP fit has 280 PDF components and 35 free parameters
- A major redesign of RooFitTools has just been completed, based on experiences of 1 year of intensive use.
- RooFitTools is a 'package' in the BaBar software structure, but has *no dependency on any other BaBar code*.
  - It should be straightforward to decouple it completely from BaBar for outside use, or to package it as a ROOT add-on.
- Documentation
  - THtml format from source code.
  - Users guide
  - Technical design note (in preparation)

ROOT problems/limitations

- ROOT is a great enabling technology, good value.
  - We are only exercising a small subset of the functionality
- Const correctness in ROOT version 3 real improvement
- CINT problems/limitations:
  - Empirical observation: Function with >10 arguments of the same time fails without proper error message
  - Zero pointer casting results in non-zero pointer
  - #include doesn't execute all code in global file scope
    - Inconvenient, because different behaviour if same code is compiled (ACLIC)
- ROOT collection classes:
  - Container classes cannot hold non-TObjects
    - Inconvenient, can e.g. not collect TIterators, Int_t, Double_t etc...
  - Will STL containers at some point replace TCollection classes?