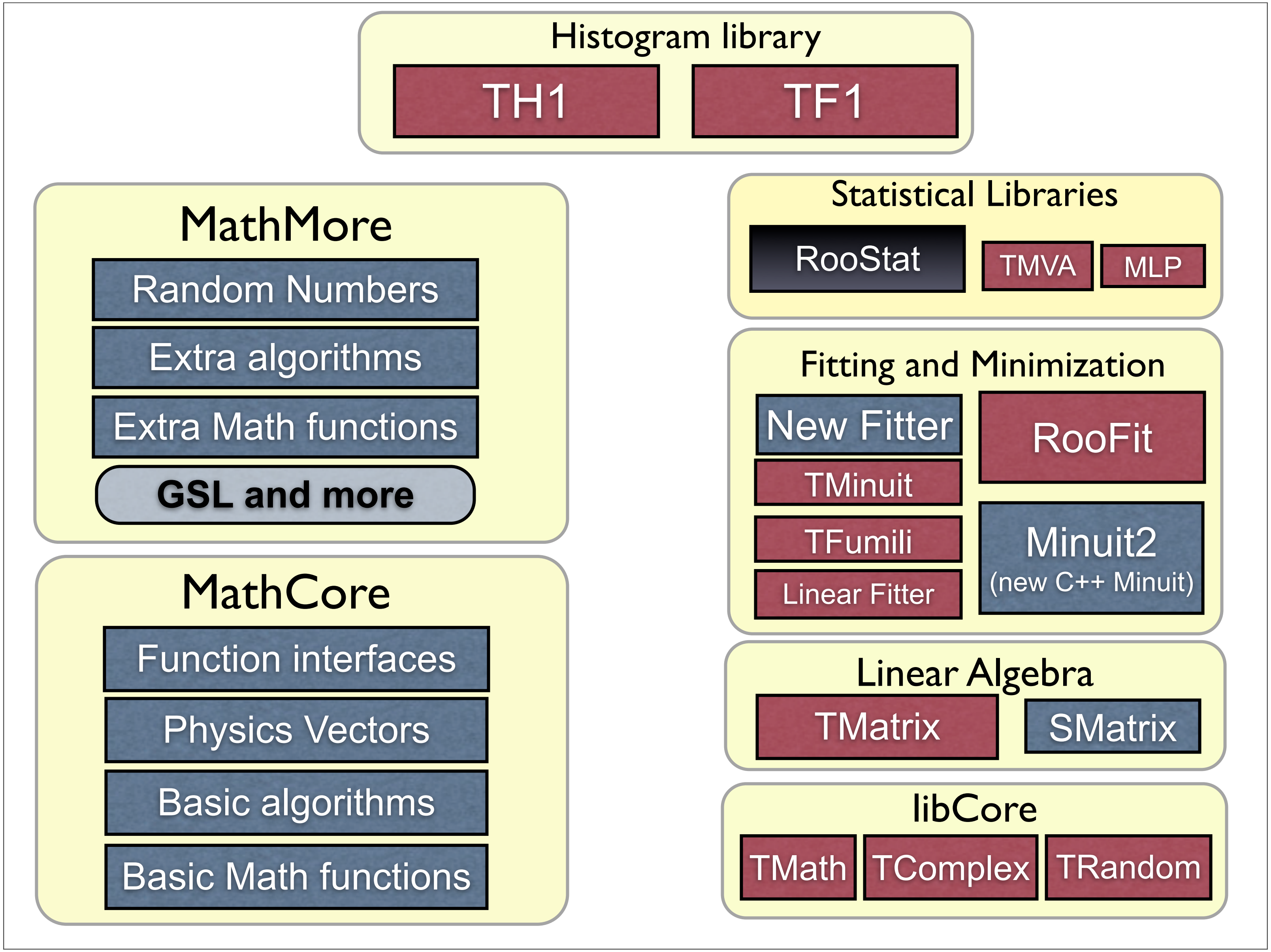
Current ROOT Math

- Current situation in ROOT:
  - libCore :
    - TMath
    - TRandom (1,2,3)
    - TComplex
  - libMathCore:
    - special functions (gamma, erf)
    - probability density functions (pdf)
    - some cumulative distribution functions (cdf)
    - Physics and geometry Vectors
    - Function interfaces and template functor classes
  - libHist:
    - derivation, root finder (1D), integration,
  - libMathMore:
    - numerical algorithms implemented with GSL
    - interface classes for some numerical algorithm (integration)

Current ROOT Math Libraries



Proposal for a new libMath

- Have a new basic Math library with
  - Math classes from base:
    - TRandom classes, TComplex , TMath
      - some functions needed by ROOT core classes are defined in TMathBase and will stay in libCore
  - all classes and interfaces from MathCore
    - basic mathematical and statistical functions
    - physics vector:
      - 3D and LorentzVector
      - Rotation and Boost classes
  - numerical algorithms from TF1
    - numerical derivation (TF1::Derivative, 2,3)
    - numerical integration (TF1::Integral, TF1::IntegralMultiple)
    - 1D minimization and root finder (Brent method) used in TF1::GetMinimum, TF1::GetX
    - use a set of interfaces which can be re-implemented using GSL in MathMore

Library Size

- Current initial estimate size of the library (on Linux slc3 gcc3.2.3)

Classes/Functions	size of Library (KB)	size of Library and Dictionary (KB)
TMath	109	240
TRandom, 1,2,3	55	150
TComplex	4	70
ROOT::Math functions	16	150
Physics Vector	116	~2000
TF1 numerical algo.	15	30
Total for libMath	**315**	**~2600**

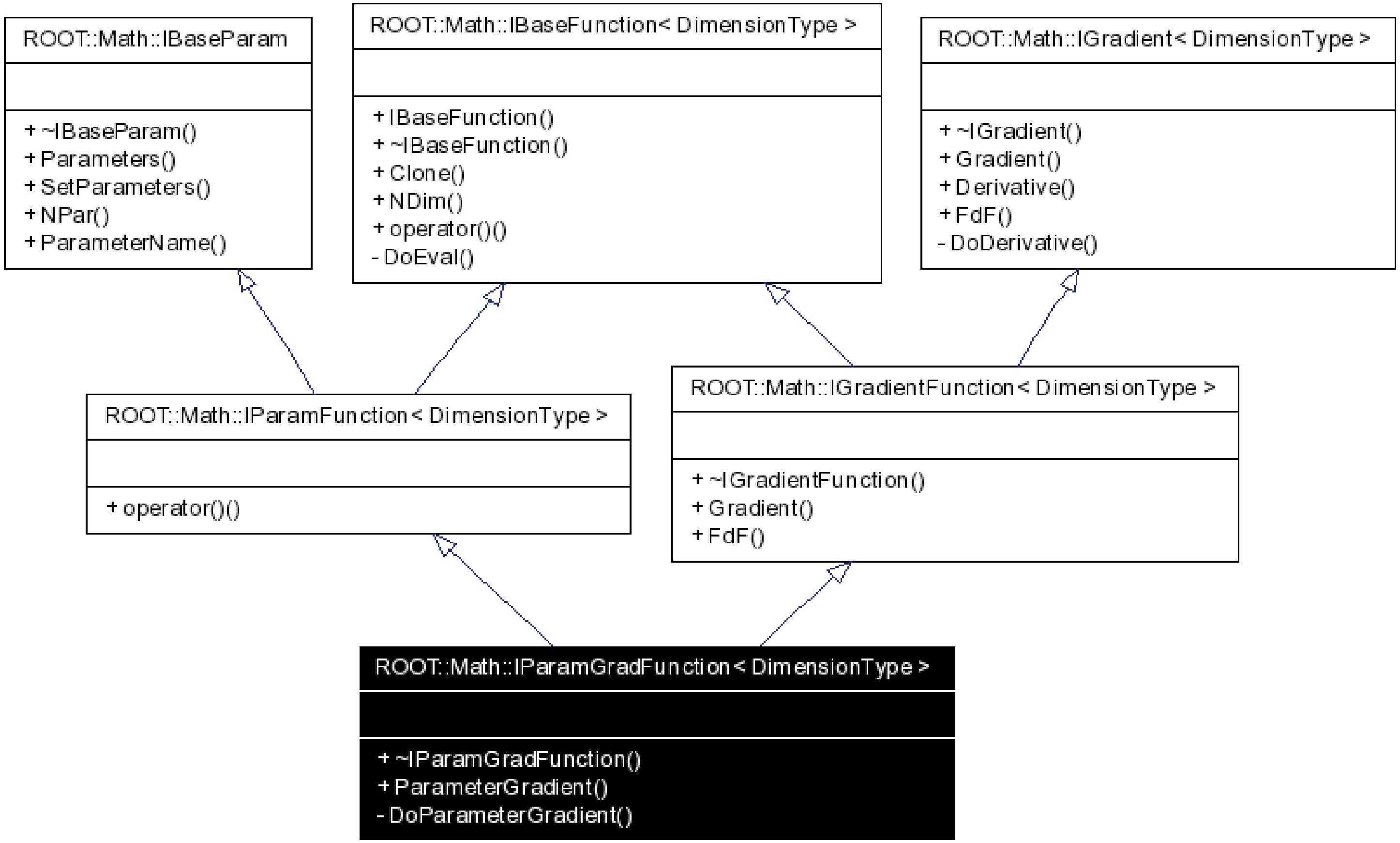
- actual size probably slightly bigger

libMath improvements

- Remove duplications TMath - ROOT::Math functions
  - implement using code from CEPHES some of the mathematical functions (incomplete beta and gamma)
    - better implementation than current one based on Numerical Recipes
  - have a consistent set of mathematical functions and distributions
    - can be extended using MathMore to more sophisticated functions
      - Legendere polynomial, Elliptic integral, etc...
- Improve TRandom classes
  - better naming (remark made also in the internal review)
    - use typedef's for backward compatibility
  - provide more type of random variates and implement some more efficient algorithms
    - additional Gaussian random variates, bi-Gaussian, Poisson, Binomial
  - have Mersenne-Twister as default engine

Function interfaces

- Minimal function interfaces to be commonly used by the numerical algorithms
  - interfaces for functions in one and multi-dimensions
  - distinguish parametric functions from general functions



ROOT::Math::IBaseParam

+ ~IBaseParam()
+ Parameters()
+ SetParameters()
+ NPar()
+ ParameterName()

ROOT::Math::IBaseFunction< DimensionType >

+ IBaseFunction()
+ ~IBaseFunction()
+ Clone()
+ NDim()
+ operator()()
- DoEval()

ROOT::Math::IGradient< DimensionType >

+ ~IGradient()
+ Gradient()
+ Derivative()
+ FdF()
- DoDerivative()

ROOT::Math::IParamFunction< DimensionType >

+ operator()()

ROOT::Math::IGradientFunction< DimensionType >

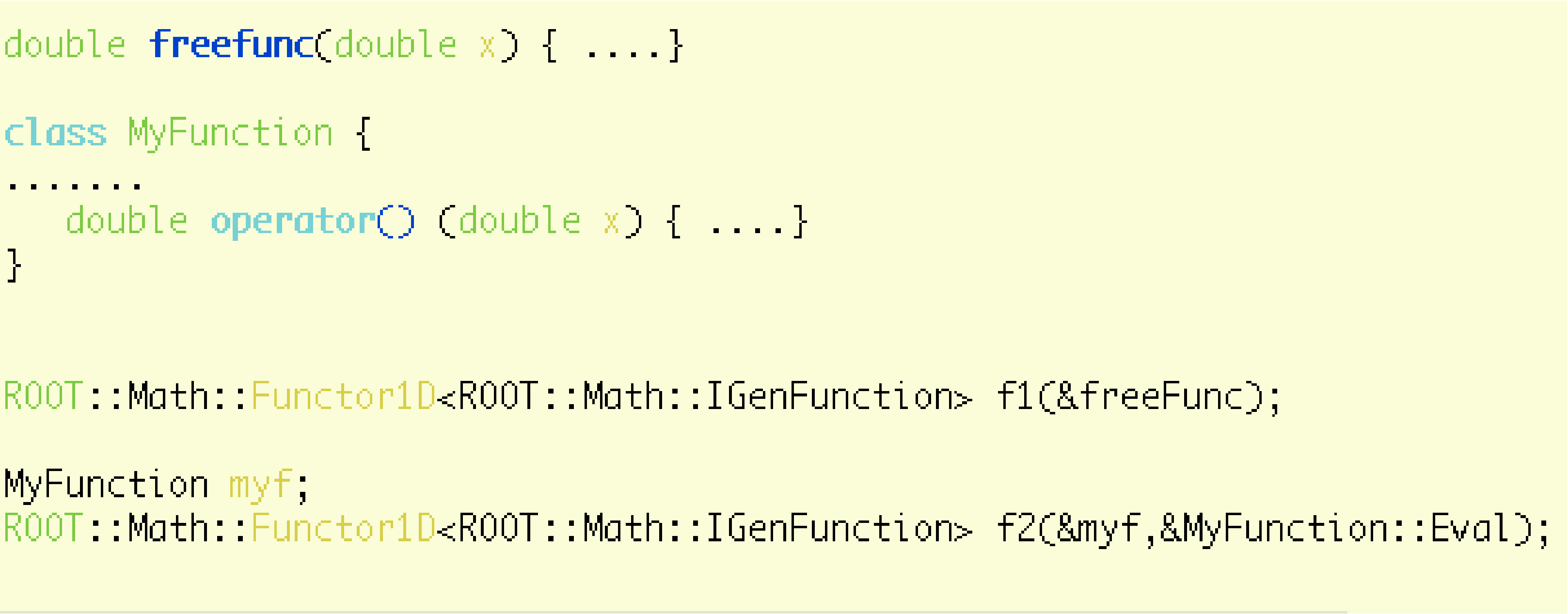
+ ~IGradientFunction()
+ Gradient()
+ FdF()

ROOT::Math::IParamGradFunction< DimensionType >

+ ~IParamGradFunction()
+ ParameterGradient()
- DoParameterGradient()

Functor classes

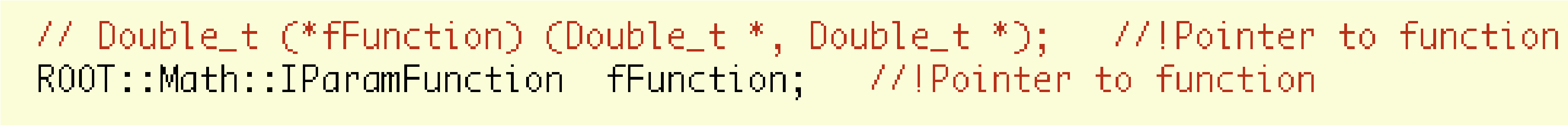
- Functor classes to wrap any C++ callable object in a function with the right interface
  - free function
  - member functions
- User does not need to provide as input a function with the right type of interface.
- Example:



```
double freefunc(double x) { ....}

class MyFunction {
.......
   double operator() (double x) { ....}
}


ROOT::Math::Functor1D<ROOT::Math::IGenFunction> f1(&freeFunc);

MyFunction myf;
ROOT::Math::Functor1D<ROOT::Math::IGenFunction> f2(&myf,&MyFunction::Eval);
```

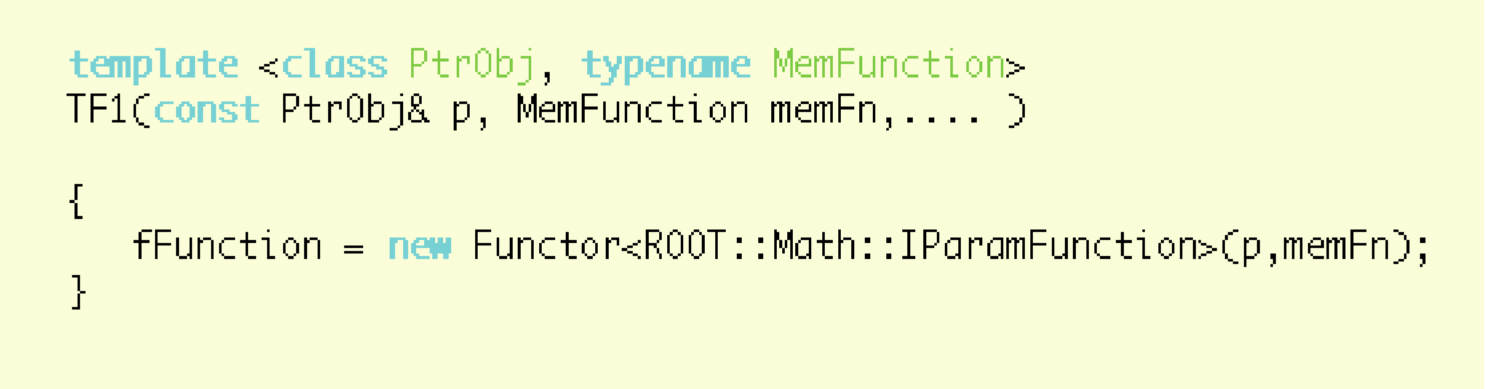
Modifications to TF1

- Ideal would be that TF1 contains inside a pointer to a parametric function interface



```
// Double_t (*fFunction) (Double_t *, Double_t *);   //!Pointer to function
ROOT::Math::IParamFunction  fFunction;   //!Pointer to function
```

- Have template constructor to create a TF1 from a :
  - free C function like now
  - an object pointer and a member function name
- use internally the Functor classes to create the fFunction pointer.



```
template <class PtrObj, typename MemFunction>
TF1(const PtrObj& p, MemFunction memFn,.... )

{
    fFunction = new Functor<ROOT::Math::IParamFunction>(p,memFn);
}
```

Numerical Algorithm

- Collect in the new libMath all the numerical algorithms (Derivation, integration, root finders, etc..) from TF1.
  - maintain the current methods for user convenience and backward compatibility
- use the classes already developed in MathMore:
  - Derivator, Integrator, RootFinder
  - Have a direct implementation extracting the code from TF1
  - same interface can be used for algorithms implemented using GSL
    - the code will be in the MathMore library and plug-in manager could be used in this case to load the plug-in's in MathMore
- Algorithms could be used directly by the users (with-out the need of having a TF1) or from other ROOT classes
  - user just needs to provide any callable object

Summary

- Proposing a new Math library merging MathCore with some existing ROOT Math functionality present in libCore and libHist.
  - it would be nice to maintain independence of the library
  - small library size : ~ 500 KB
  - we should temporarily have current MathCore dictionaries (for the template physics vector) in a separate library
- Proposed restructure of TF1 :
  - use new function interfaces
    - extend capability of the class
  - use numerical algorithms from libMath
- Possible future extensions:
  - Add the interfaces and base classes for fitting and minimization
    - Fitter and Minimizer interfaces, FitData, FitResult
      - will use plug-in manager to load minimization library