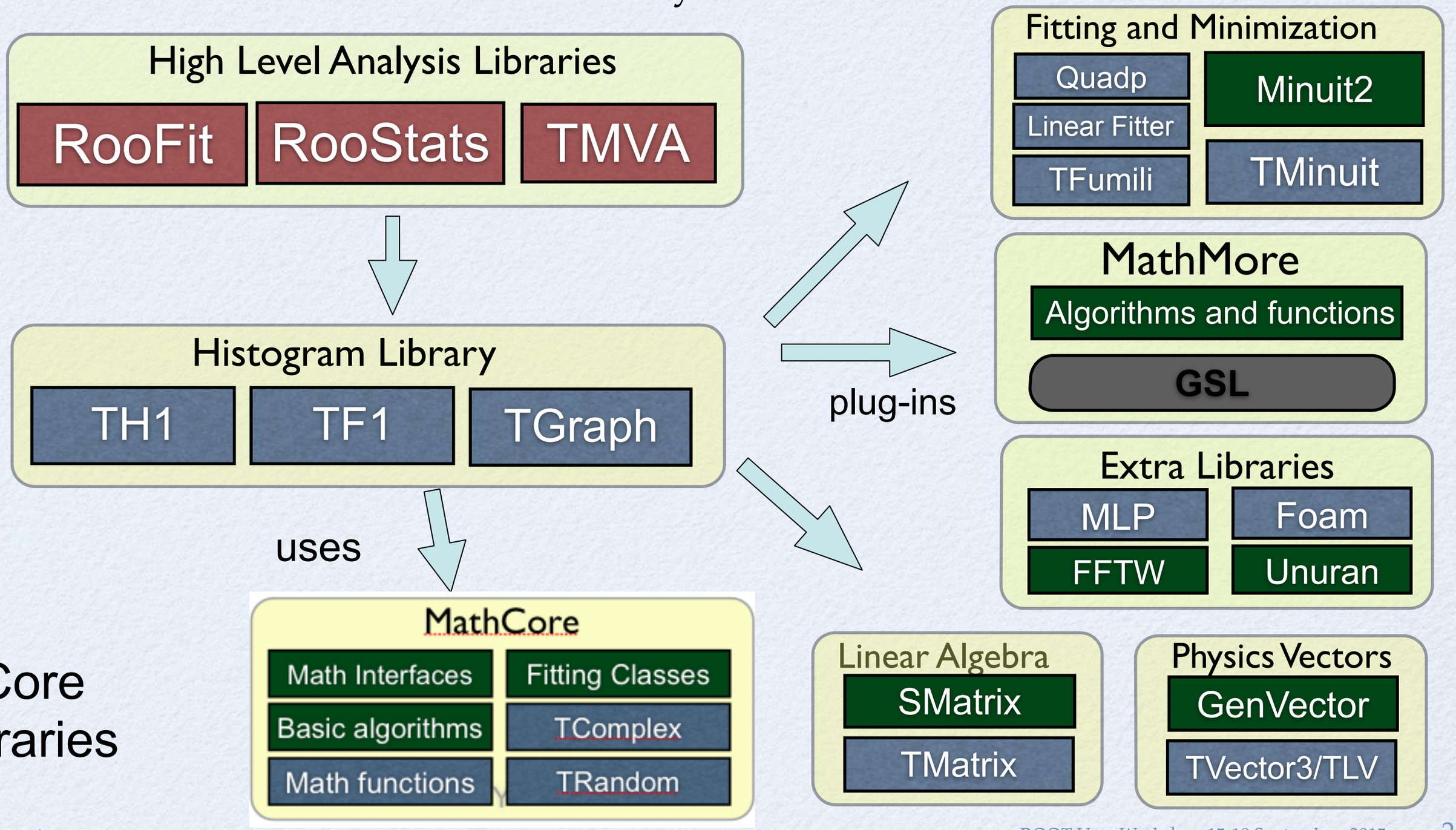
What's new in the ROOT Math/Stat Libraries

L. Moneta
CERN-PH-SFT

ROOT User Workshop
Saas-Fee, 15-18 September 2015

ROOT Math/Stat Libraries

- Large set of mathematical libraries and tools needed for event reconstruction, simulation and statistical data analysis



Core libraries

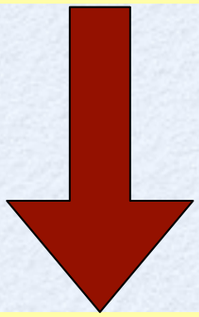
Outline

- New TFormula class
- Improvements in TF1
- New Random number classes
  - new MIXMAX generator
- Recent developments in TMVA (redesign and interfaces to R and Python)
- Outlook and conclusions

New TFormula

- **`TFormula`** creates a C/C++ function which is passed to Cling

```
TFormula f("f", "[0] * sin( x * [1] )" );
```



```
Double_t TFormula____id107089489689037131 75
(Double_t *x,Double_t *p)
{
        return p[0]*TMath::Sin(x[0]*p[1]) ;
}
```

- The created function is now compiled on the fly using the JIT capabilities of Cling
  - evaluation is fast as a library (pre-compiled) function

New TFormula Capabilities

- Expression is (after some pre-processing) parsed by a real compiler
  - correctness of result
  - fast execution
- New functionality:
  - can define directly parameter names in expression

```
TFormula f("f", "[A] * sin( x * [B] )" );
```

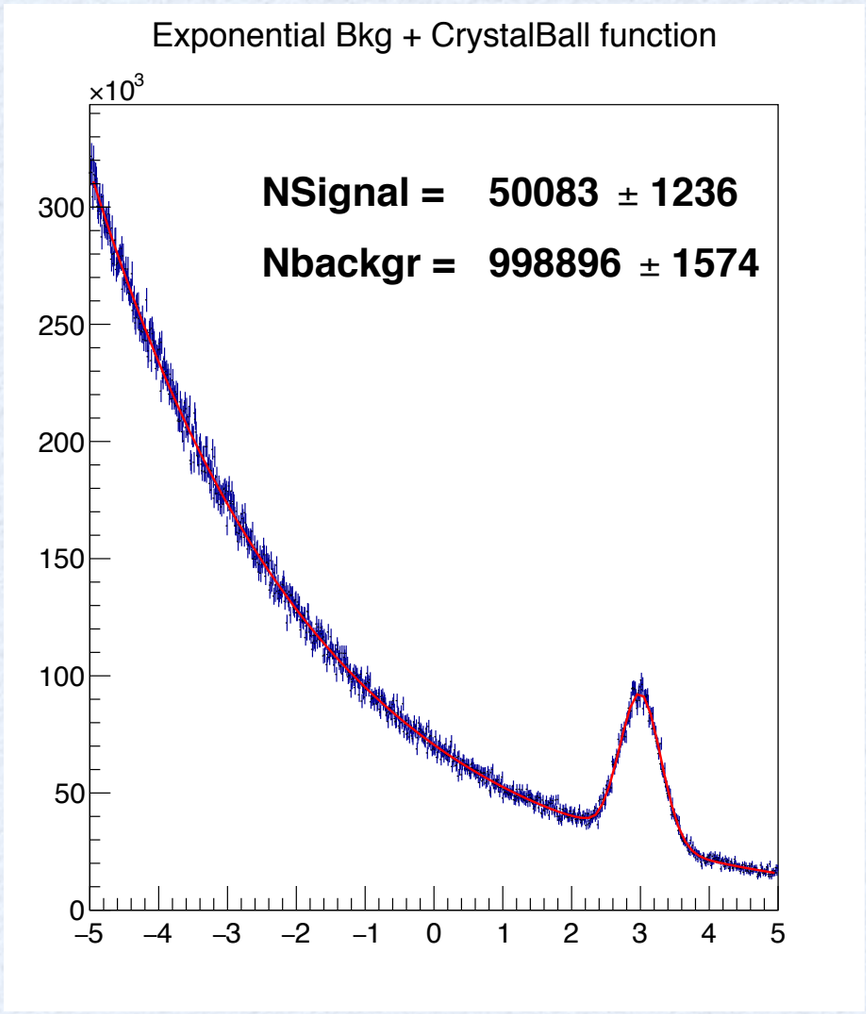
  - easy to extend by adding new pre-defined functions
    - e.g. `crystalball, cheb0,…cheb9`
- can input direct C++ code (no pre-parsing of expression) using a lambda function

```
TFormula f("f", "[]{double *x, double *p){ return p[0]*sin( x[0] * p[1] );}",1,2);
TFormula g("g","[&](double *x, double *p){ return graph->Eval(x[0]);}",1,0}
```

- This works (only for a TF1) using code and not expression, but it cannot be saved in a file `TF1 f1("f1",[&](double *x, double *p){ return graph->Eval(x[0]);},xmin,xmax,0}`
- The TFormula lambda's can be saved in a file and re-use afterwards if all captured objects will be available when reading

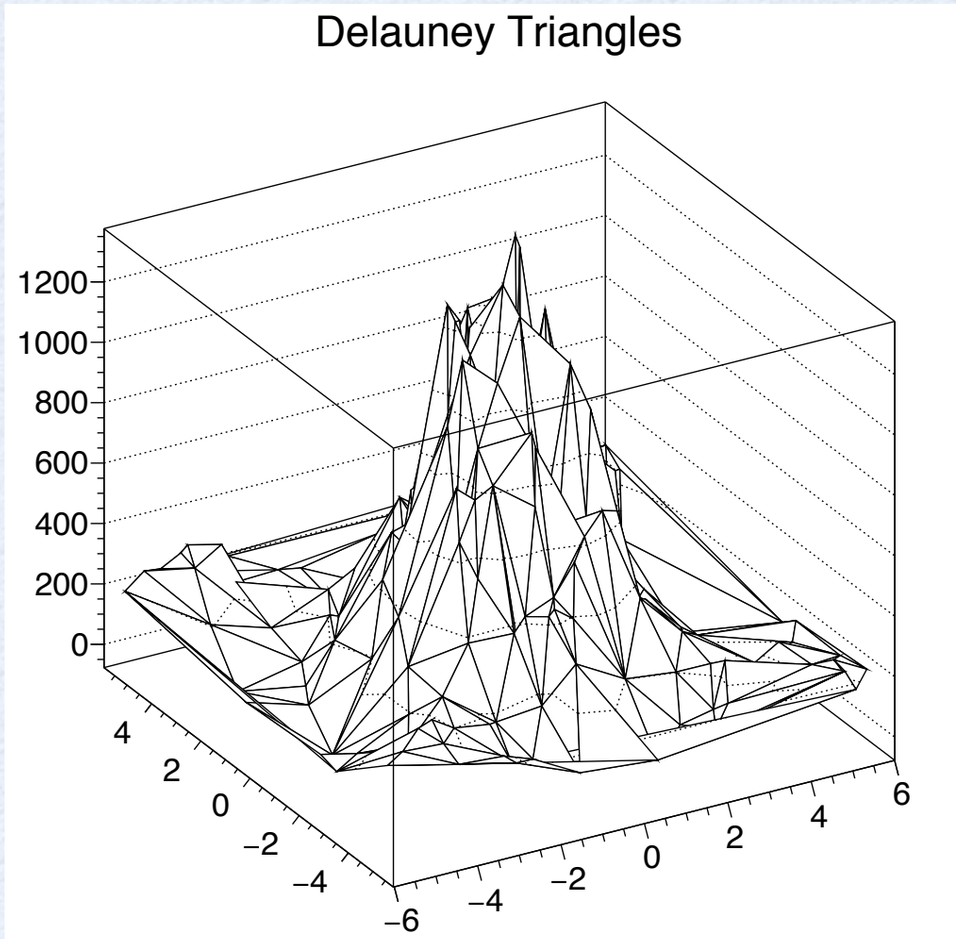
Improvements in TF1

- Support for automatic normalisation
  - normalise functions in their given range (useful for fitting)
- Support normalised additions of TF1 objects
  - n1 * f1/∫f1 + n2 * f2/∫f2
  - can be used to fit directly the number of events
    - no more need to TF1::IntegralError
    - unbiased binned likelihood fit (e.g. using bin function integral)
      - not available in RooFit
- Support for convolutions (analytical and using FFT)
  - create at TF1 from convolution of two function objects
  - can be used for fitting
- TF1 uses new TFormula class (not inheriting anymore)



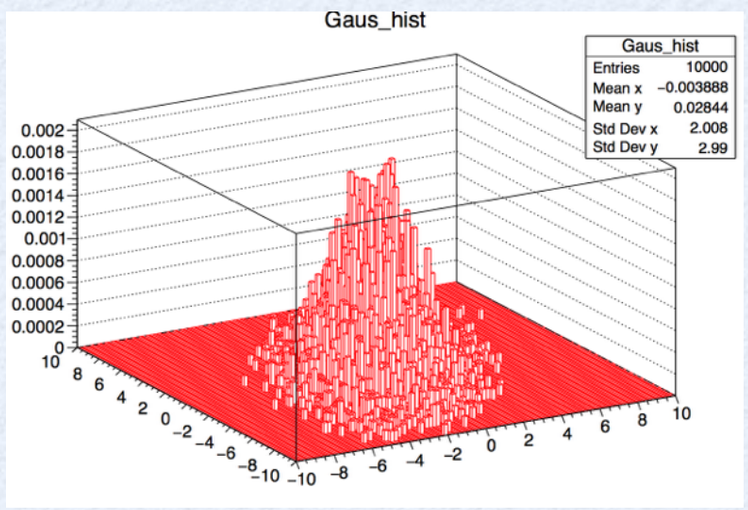
New 2D-Interpolation

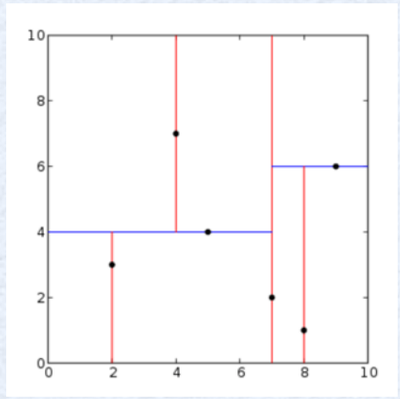
- New algorithm for the 2D Delauney triangulation (work by *D. Funke*)
- Use for 2D interpolation for points not distributed in a grid
  - code from triangle package (from *J. R. Shewchuk*)
    - Very good and efficient code for triangulation
- Use now in TGraph2D::Eval
  - for 1000 points and 1000 evaluation
    - time is 0.1 s, speedup ~ 10
  - for 10000 points
    - time is 0.7 s, speedup is1400 !!

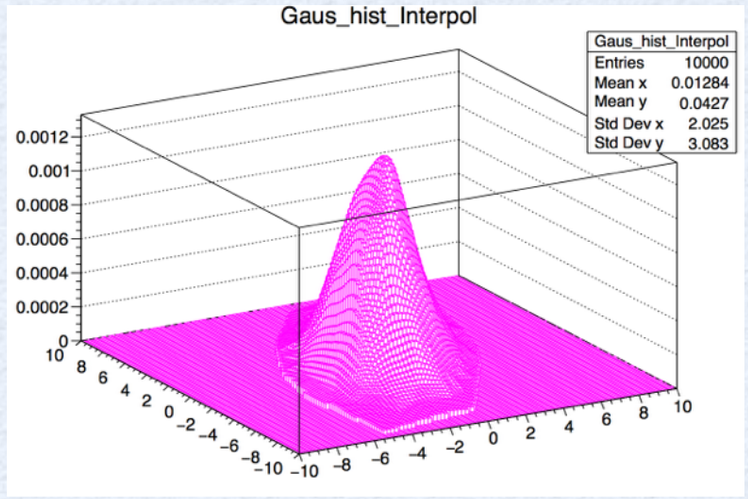


2D-Kernel Estimation

- New class for kernel density estimation in multi-dimension (work by *V. Milosevic*)
- Re-implementation of RooNdKeyPdf using TKDTree and its binning capabilities (TKDTreeBinning)
  - faster for looking at neighbouring points
  - use binned evaluations (with interpolation in 2D) and binned adaptive bandwidth







New Random Number Classes

- New classes for generation of pseudo-random numbers
  - generation of uniform numbers
  - generation according to standard distributions
- Totally independent of other ROOT classes
  - no more TObject inheritance
  - can be shipped as a standalone package (e.g. to be used in simulation programs)
- Avoid as much as possible usage of virtual functions
  - optimal CPU performances
- Maintain I/O capabilities for internal generator state
- Foreseen support for vectorisation
- Easy integration of new random generator engines

New Random Number Design

- Clear separation between
  - engine classes (that generates the numbers)
  - generation of numbers according to distributions
- RandomEngine
  - class implementing algorithm for generation of pseudo-random number
- RandomFunctions
  - class implementing most used random number distributions
- template user class Random with an interface similar to TRandom

Random User class

- User class as interface for generation of random numbers

```
template < class Engine>
class Random {

   double Rndm();

   double Gaus(double mu, double sigma);

   int Poisson(double mu);
```

  - Generation of number in uniform (0,1] interval obtained from the RandomEngine classes
  - Distributions implemented using RandomFunction class
    - same interface as already existing ROOT::Math::Random class in MathMore
  - Define convenient typedefs for common engines (e.g. RandomMT)

Random Engine Class

- Implementation of pseudo-random number generation
- Different class (type) for each algorithm
  - **`MersenneTwisterEngine , MixMaxEngine, LCGEngine`**

```
class LCGEngine {
  public:
    LCGEngine() : fSeed(65539) { }

    void SetSeed(unsigned int seed) { fSeed = seed; }

    /// generate a number in (0,1] interval
    inline double operator() () {
       const double kCONS = 4.6566128730774E-10; // (1/pow(2,31)
      fSeed = (1103515245 * fSeed + 12345) & 0x7fffffffUL;
      if (fSeed != 0) return  kCONS*fSeed;
      return (*this)();
    }

   unsigned int fSeed;  /// state of generator
};
```

Advantages of new Random classes

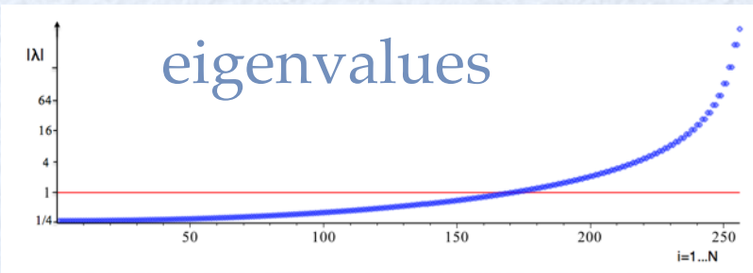
- Easy to include new types of generators
  - plan to add specialised generator exploiting vectorization
  - added MIXMAX generator, maybe we will add Random123
  - can easily include different generator libraries:
    - wrapper to GSL random
    - wrapper using std::random engine
- Possible also to extend random function implementations
  - already have distributions from GSL
    - Gamma, Chisquare, Multinomial, etc..
  - add those from std::random
- will implement TRandom classes using ROOT::Math::Random

MIXMAX generator

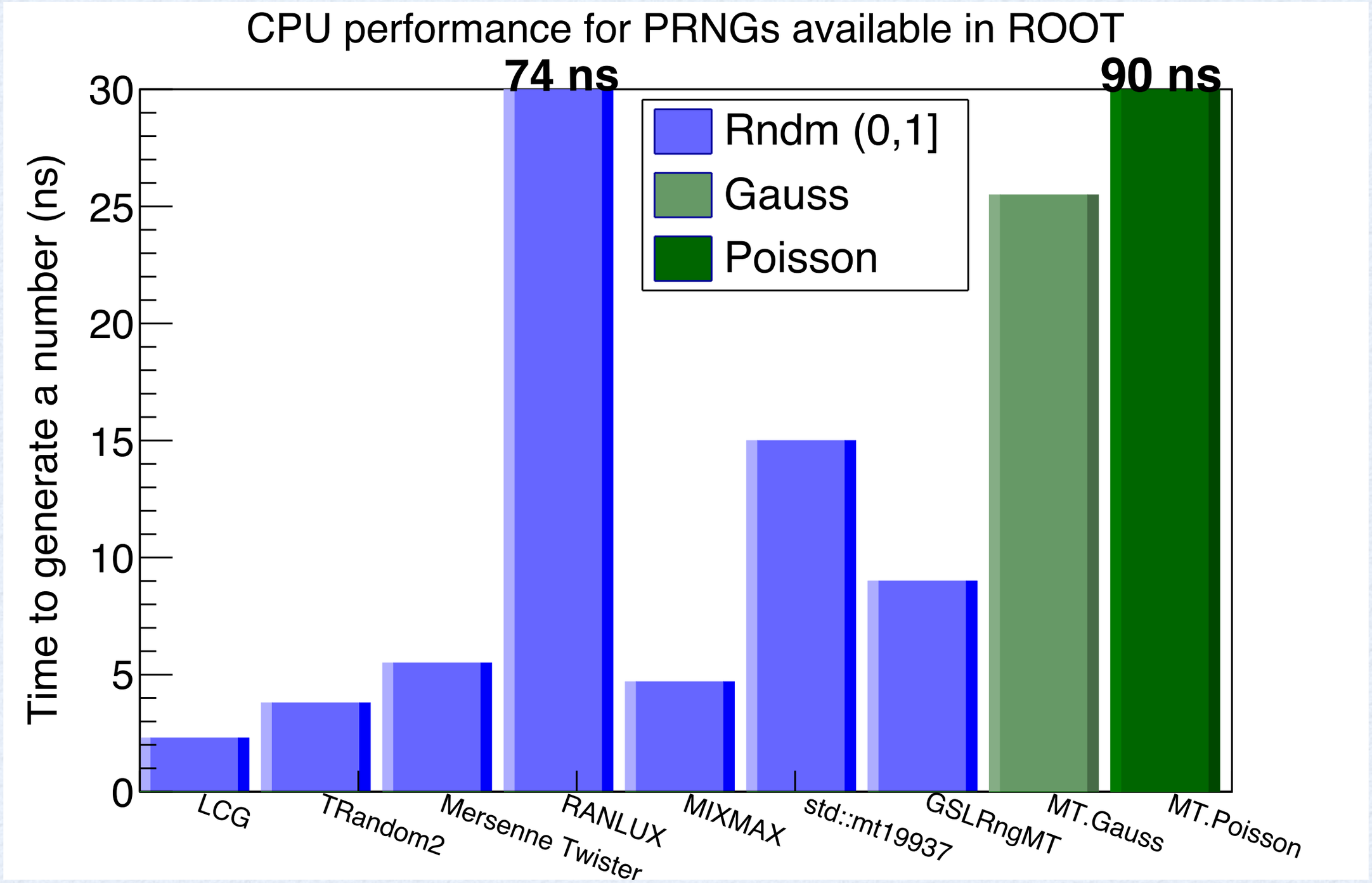
- Matrix recursive random number generator
  - idea from *G. Savvidy* and
    new implementation by *K. Savvidy*
  - based on theory of dynamical system
    (Kolmogorov K-systems)
- New fast implementation
  - CPU time is now comparable to Mersenne Twister
  - passes all Big Crush tests of TestU01
  - default state consists of N=256 (64 bits integers)
- Can generate independent sequences for different given seeds by skipping ahead
    - guaranteed no overlap if n < 10100
- Plan to make it the ROOT default random number generator
- Work done in a network collaboration of SFT with some external institutes (EU project **MIXMAX-H2020-MSCA-RISE02014)**

$$u_i(t+1) = \sum_{j=1}^{N} A_{ij}\, u_j(t) \bmod 1$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & \dots & 1 & 1 \\ 1 & 2 & 1 & 1 & \dots & 1 & 1 \\ 1 & 3+s & 2 & 1 & \dots & 1 & 1 \\ 1 & 4 & 3 & 2 & \dots & 1 & 1 \\ & & & \dots & & & \\ 1 & N & N-1 & N-2 & \dots & 3 & 2 \end{pmatrix}$$



PRNGs CPU Performance



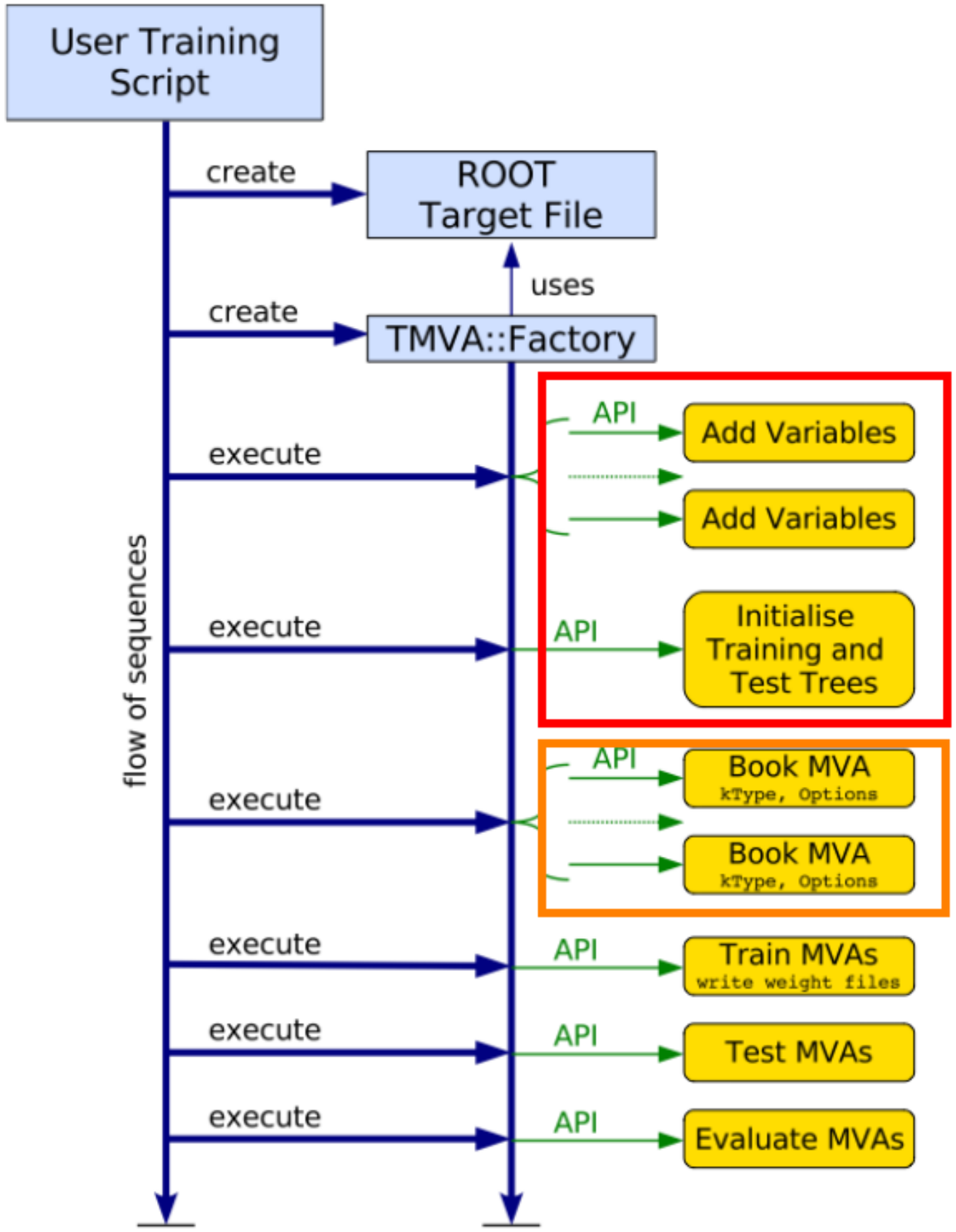
New random classes and MIXMAX available in ROOT 6.05.02

Recent Work in TMVA

(S. Gleyzer, L.M., O. Zapata)

- Prototype redesign of TMVA
- TMVA interface to methods from R (RMVA)
- New interface to Python methods from Scikit-Learn (PyMVA)

TMVA designed for comparison of different Machine Learning (ML) methods



Users operate via the Factory class

- Add variables
- Load datasets
- Book MVA methods
- Train
- Test
- Evaluate

Factory treats equally all booked methods

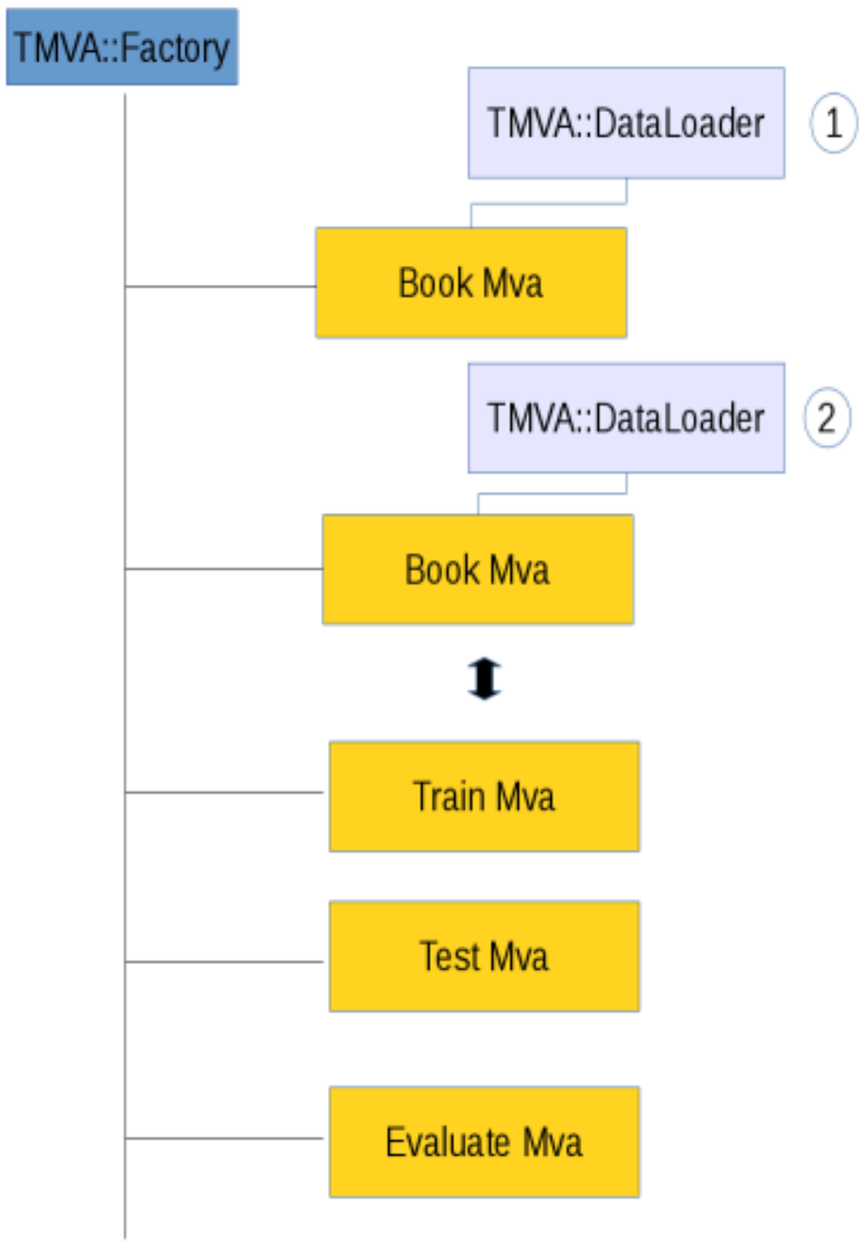
- Facilitate comparison
- Common pre-processing

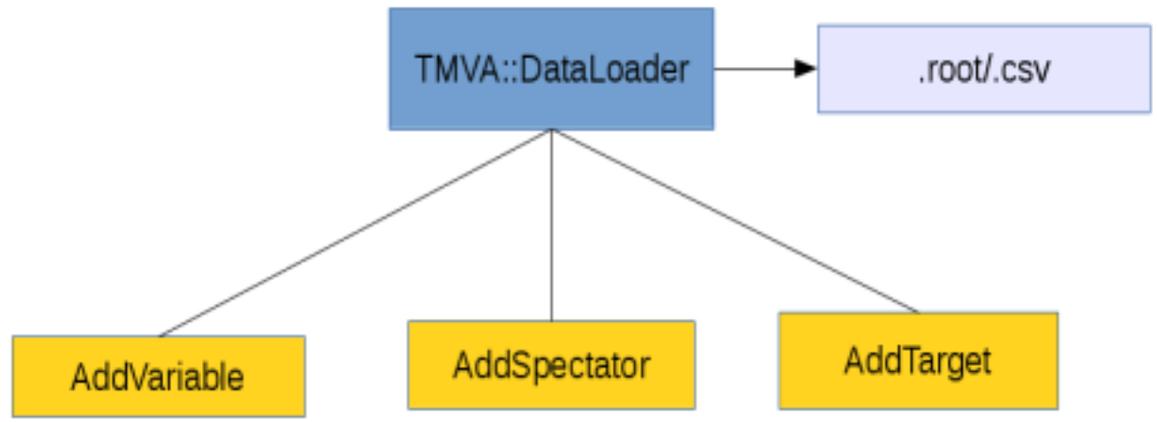
Issues with current TMVA

- All methods see same data for training, test and evaluation
- This is also a limitation, design is not flexible enough
  - cannot use different datasets for different methods
  - cannot change input variables for the methods
- Difficult to integrate new methods for cross validation, variable importance,etc..
- Also large use of static variables
  - cannot run more than one Factory in the same job
  - problem for concurrency
- Several issues raised by users in the ROOT questionnaire
  - performances, missing modern ML methods (e.g. deep neural networks)

Prototype a new TMVA Design

Introduce a TMVA::DataLoader class






```
factory->BookMethod( DataLoader &,Types::EMVA ,  TString methodTitle, TString theOption);
```

see http://oproject.org/tiki-index.php?page=TMVA

TMVA::DataLoader

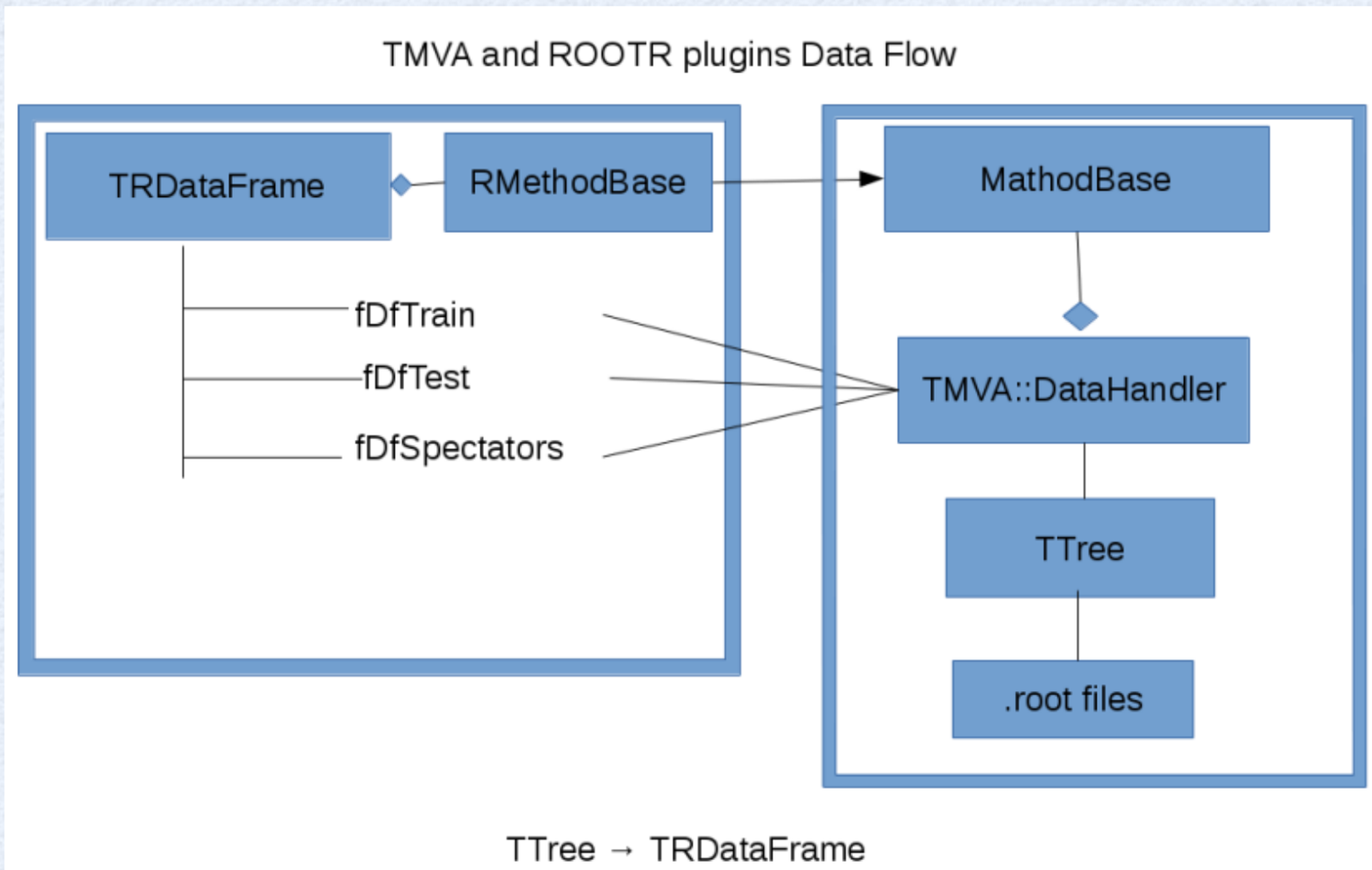
- DataLoader is a new interface to
  - load the datasets (root files, .csv files, ….)
  - add variables
- Factory links DataLoader with a specific MVA method when booking

```
factory->BookMethod( DataLoader *loader, Types::EMVA theMethod,
                     const char * methodTitle, const char *option = "" );
```

- Obtained desired flexibility
  - possible to customise different methods
  - new code available in github and soon will be integrated in ROOT master
    - https://github.com/omazapa/root/tree/master-root-R-tmva

R-TMVA

- Interface R methods in TMVA
  - use new ROOT-R package (allows to use R in ROOT)
  - use current TMVA design (not the new prototype)

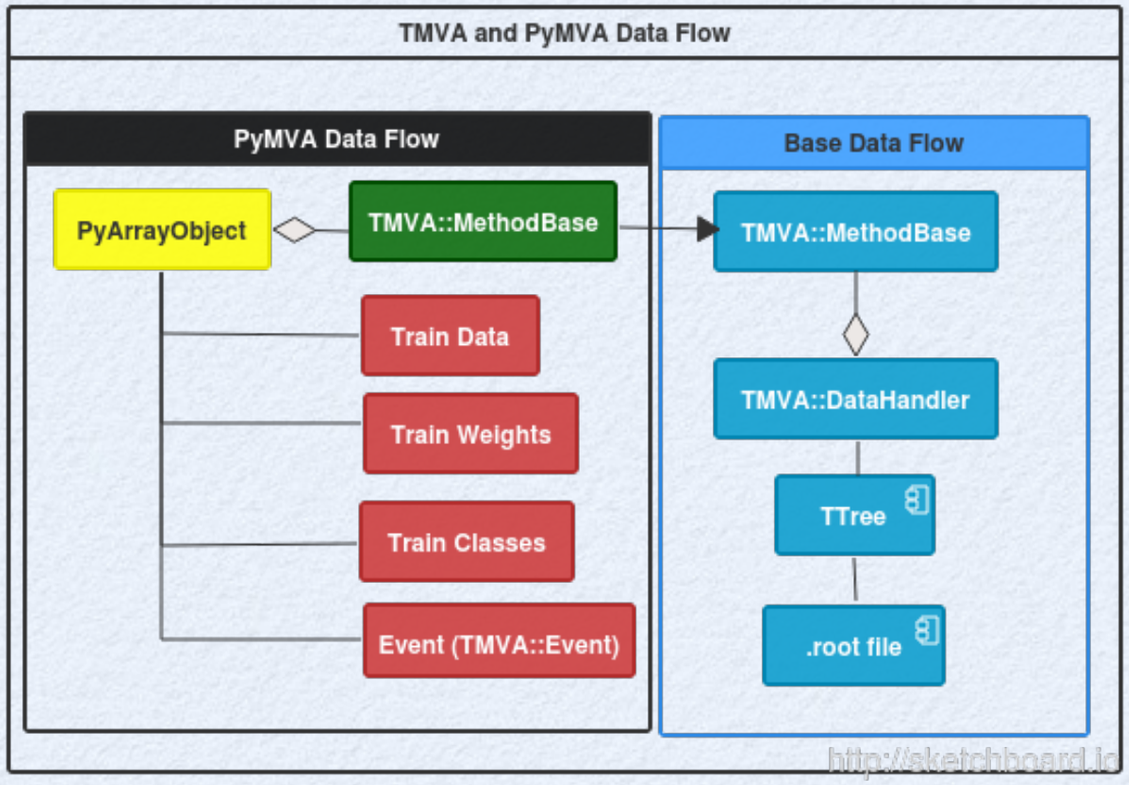


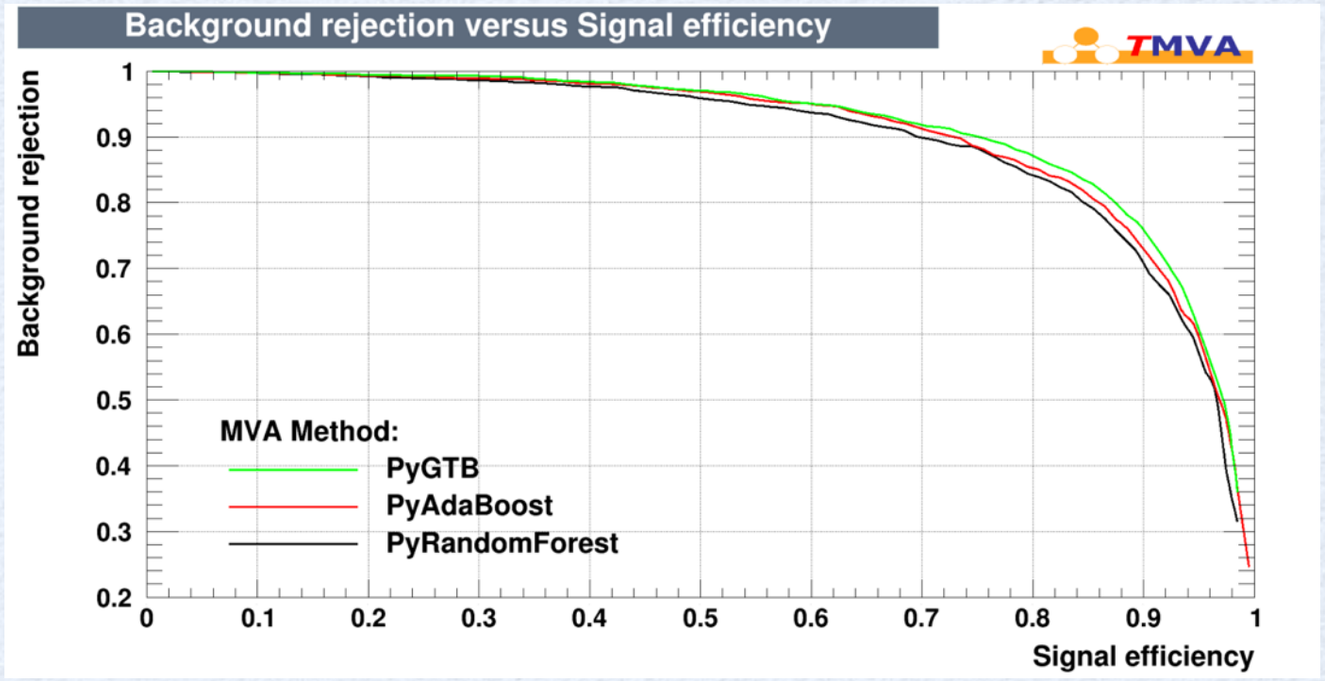
- Map ROOT data in a R data frame (TRDataFrame)
- Implement new R methods as derived class of TMVA::MethodBase

Available now in ROOT 6.05.02. See doc at http://oproject.org/tiki-index.php?page=RMVA

PyMVA

- New interface to use Python ML tools from TMVA
  - Use Scikit-Learn methods
    - Random Forest, Gradient Tree Boost, Ada Boost
  - Convert ROOT data in PyArrayObjects (C interface to numpy)
  - Use directly Python from C++ using its C interface





see http://oproject.org/tiki-index.php?page=PyMVA

code available now in ROOT 6.05.02 !

Future Improvements in TMVA

- Persistency of methods
  - use general ROOT I/O (and not be limited to XML) for output of training
- Parallelisation (needed to exploit better new hardware)
  - limit usage of static variables for multi-threads
  - porting eventually to GPU when possible
- Code improvements and optimisation, exploit vectorisation and new C++ features.
- Remove duplications
  - Some functionality already exists in ROOT
    (e.g. KDE, splines, several statistical functions in TMVA::Tools)

Future Math Developments

- New version of TTreeFormula based on new TFormula
  - a prototype has been developed by a GSOC student (*M. Zminoch*)
- Exploit parallelisation using both multi-process (e.g using new MultiProc module) and multi-threads
  - fitting, TMVA, toy MC studies (for statistics)
- Exploit vectorisation
  - worked already on a prototype vector interface of TFormula
  - vector function interfaces to be used when fitting
    - template interface which can be specialised to Vc vector types

Conclusions

- We are focusing on
  - consolidation of the Math and Statistics libraries
  - adding new algorithms/methods following user requests
  - re-design of old classes when needed
  - interfaces to external software packages
    - R, Python libraries, etc..
  - performance optimisation
    - vectorisation, parallelisation and exploiting new hardware(GPU)
  - improve the documentation and provide more tutorials (RooStats)
- Driven by user needs
  - your feedback is essential !