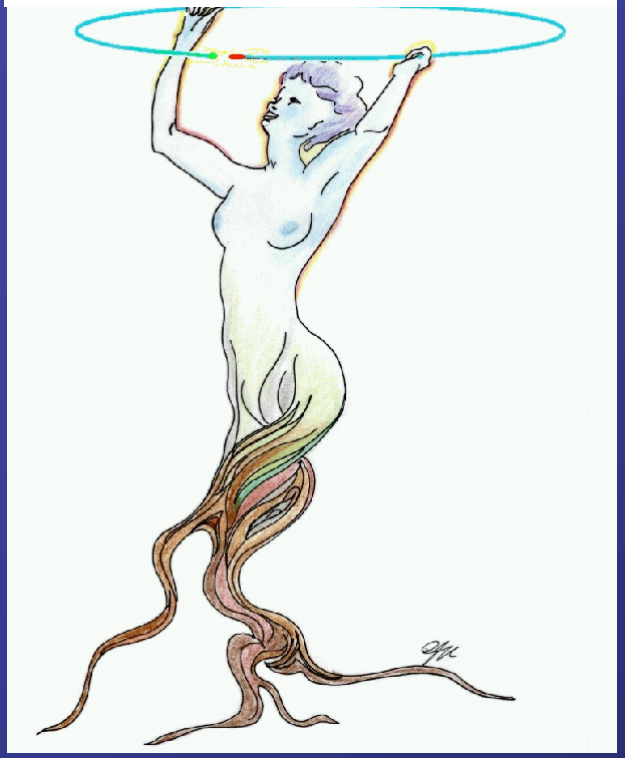
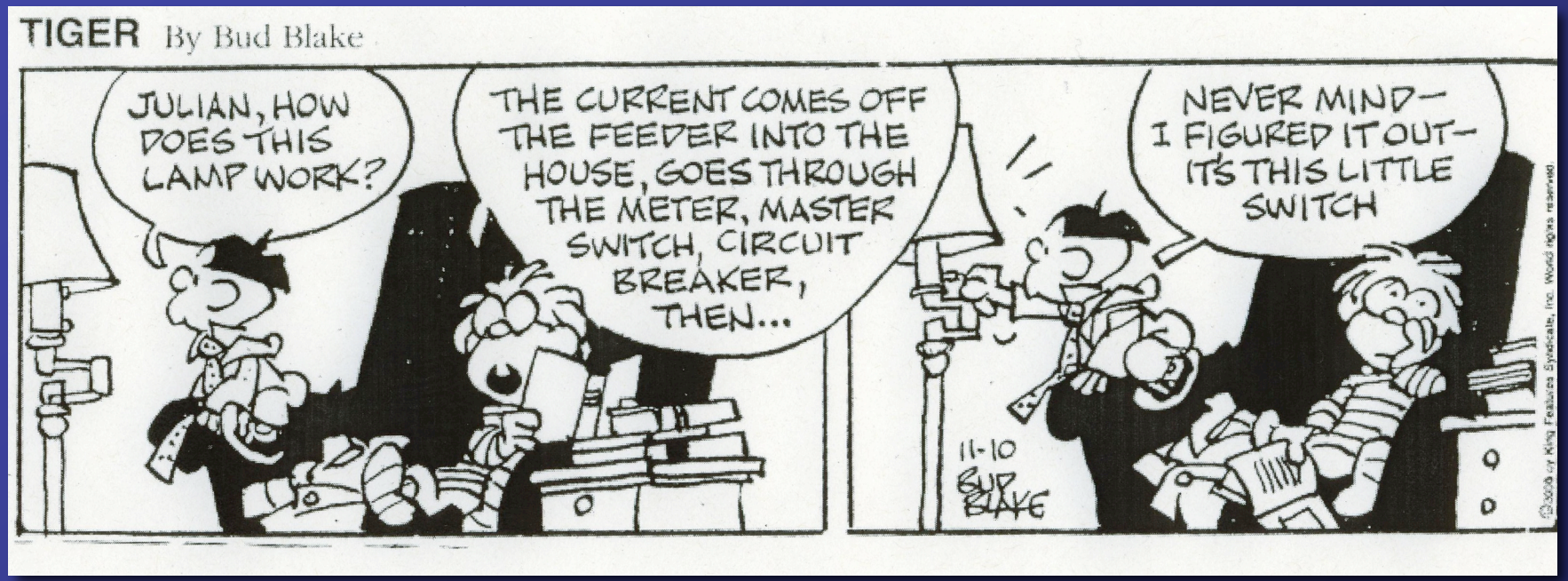
ROOT Course
Part II

ROOT - An Object-Oriented Data Analysis Framework



"The basic concepts behind an Object-Oriented framework"

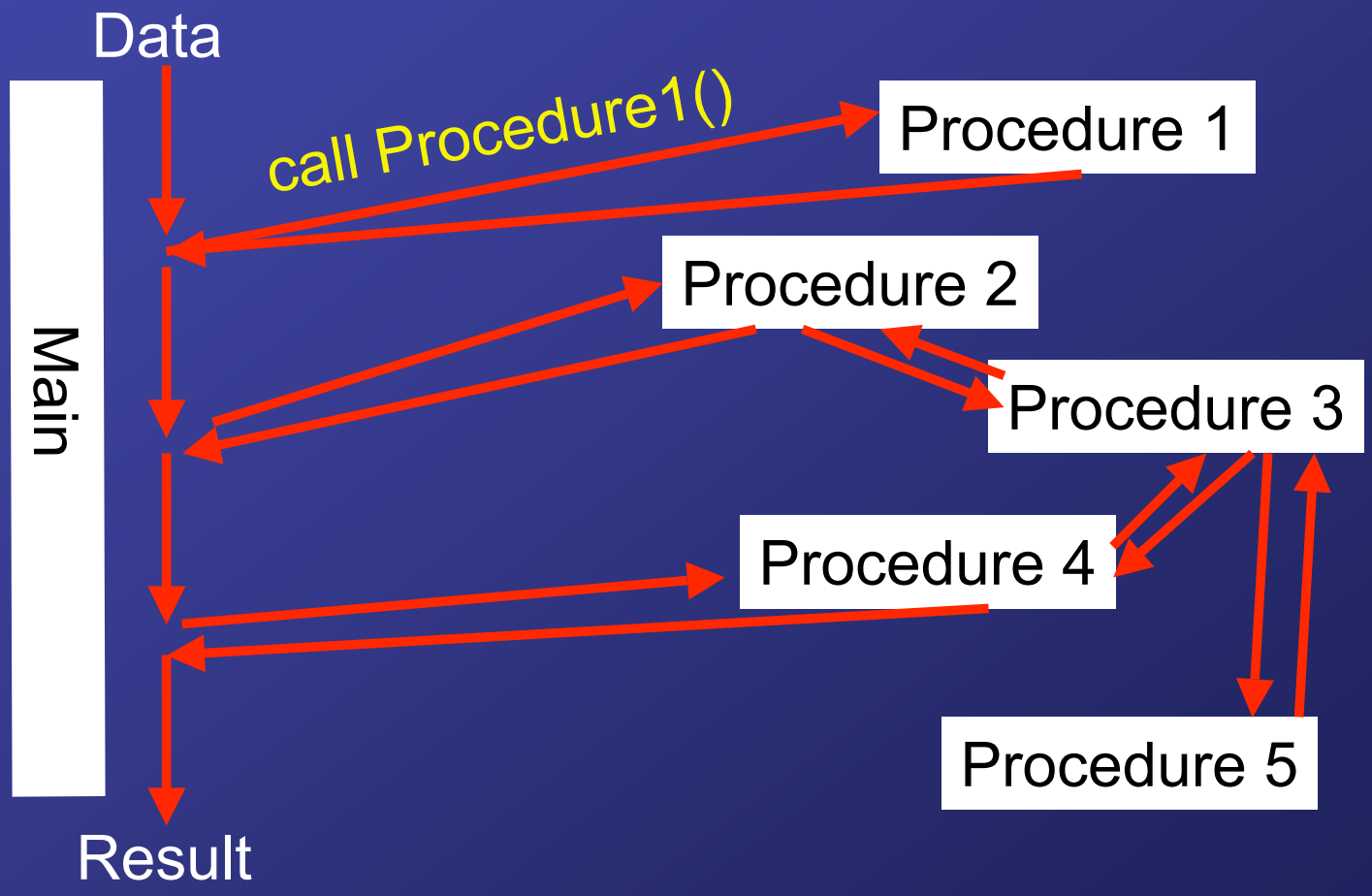
Gerco's view on Object-Oriented Framework



Well, this might be true...

but what does it really mean?

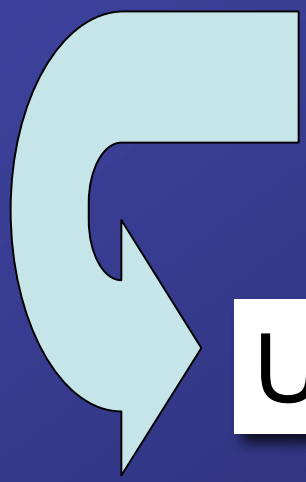
Procedural Oriented Programming



Procedural Oriented Programming

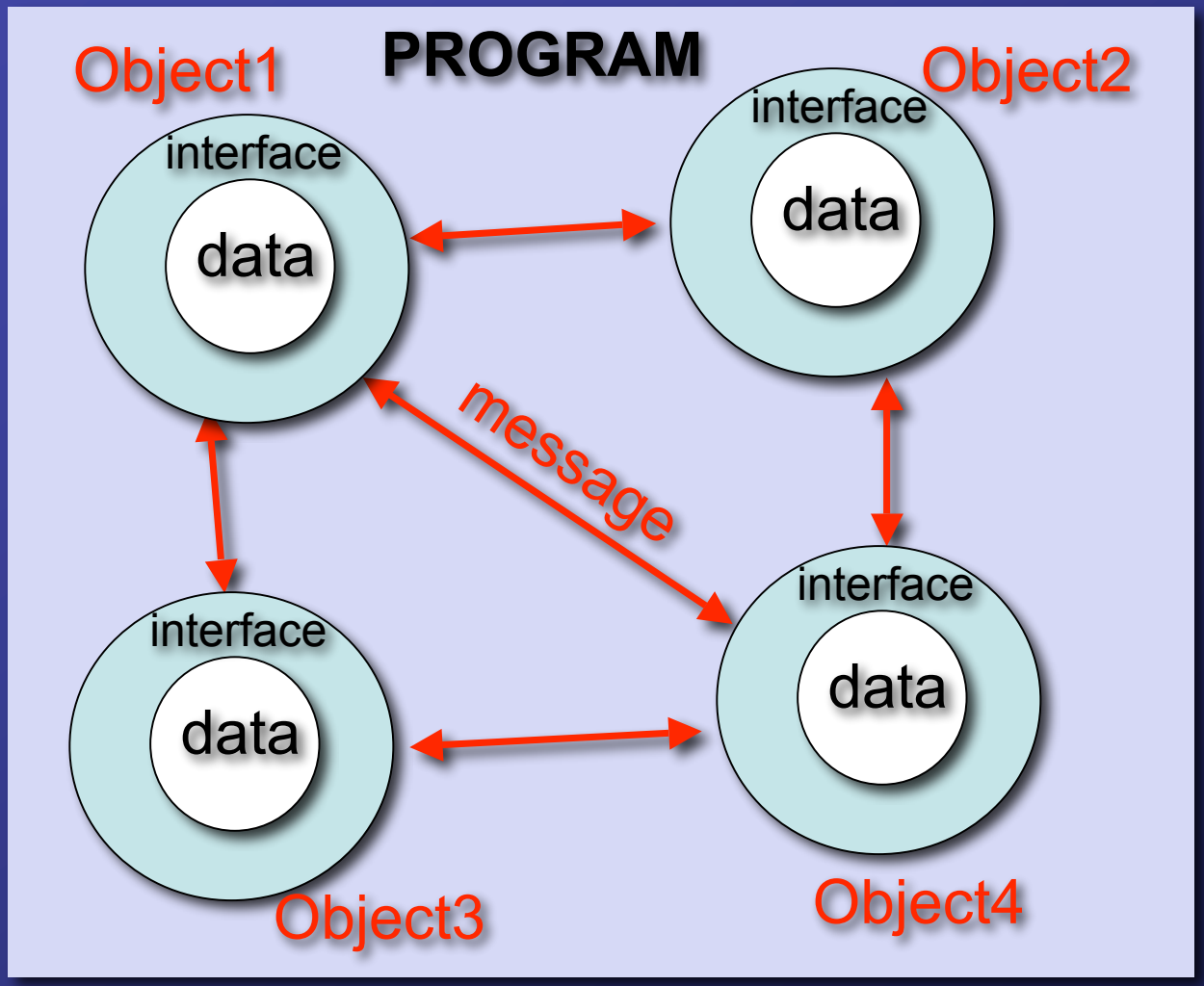
Although widely used and, from a computer perspective, very efficient, it is limited since…

1. …unreadable for large complex codes
2. …difficult to manage
3. …far from the "real world"



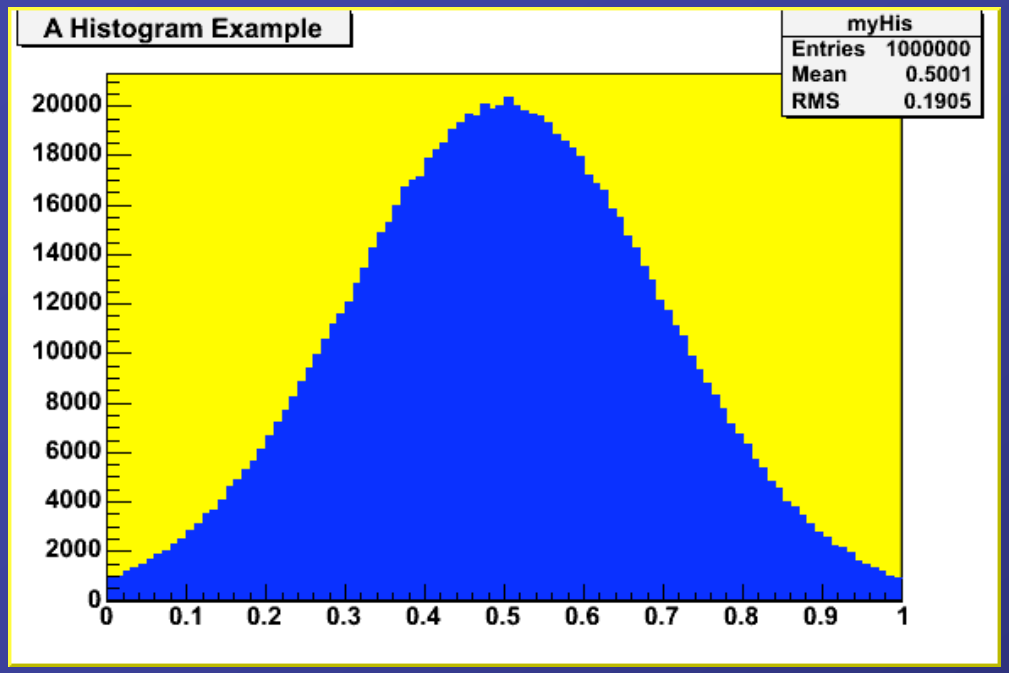
User and Programmer Unfriendly!

Object Oriented Programming

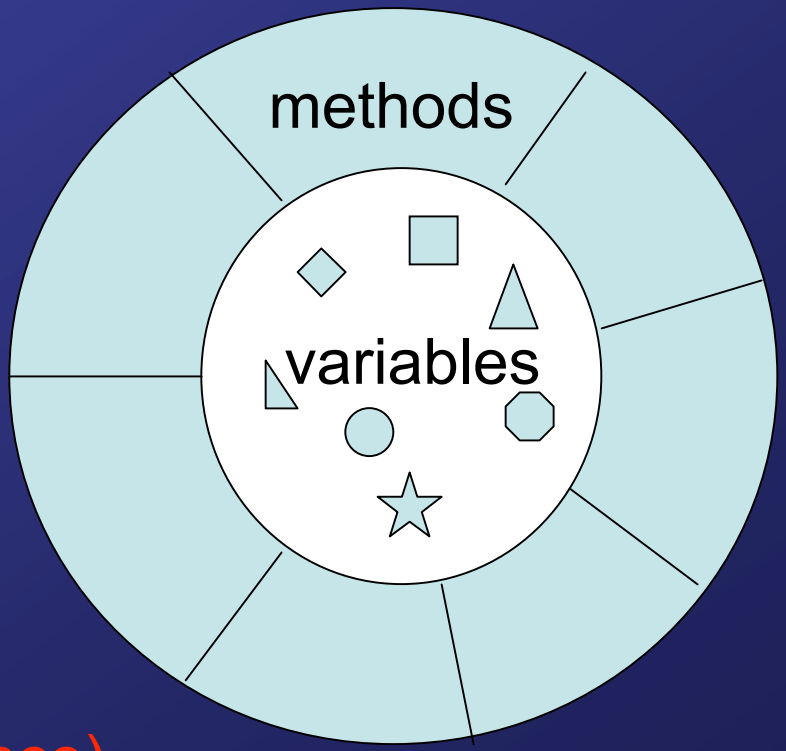


Modularity and Information Hiding

An example Object



… a histogram

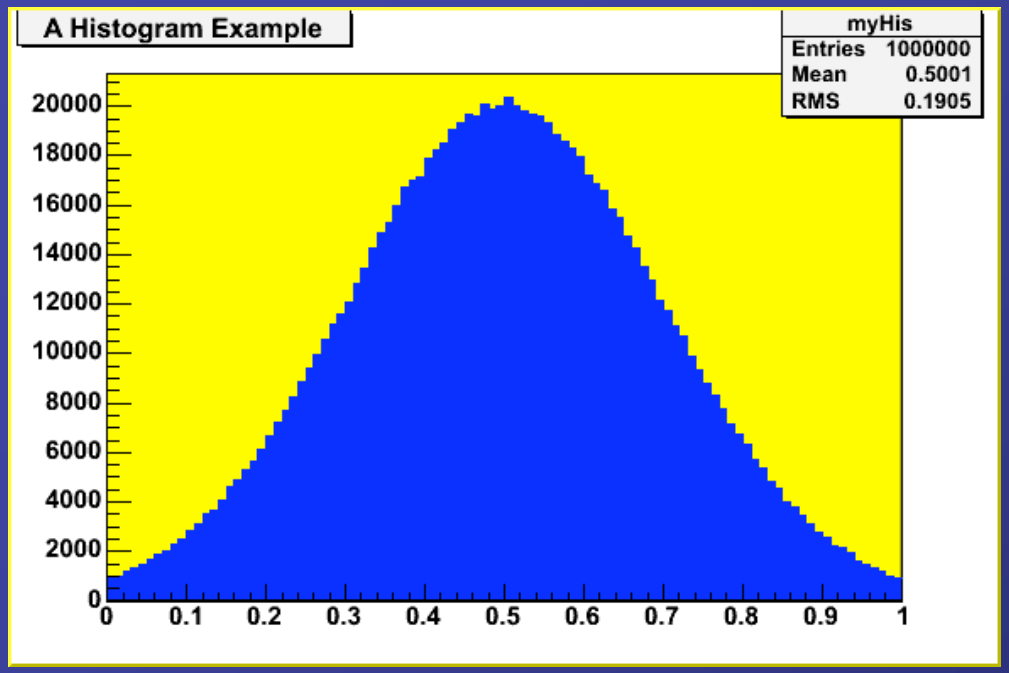


An abstract model of histogram object…

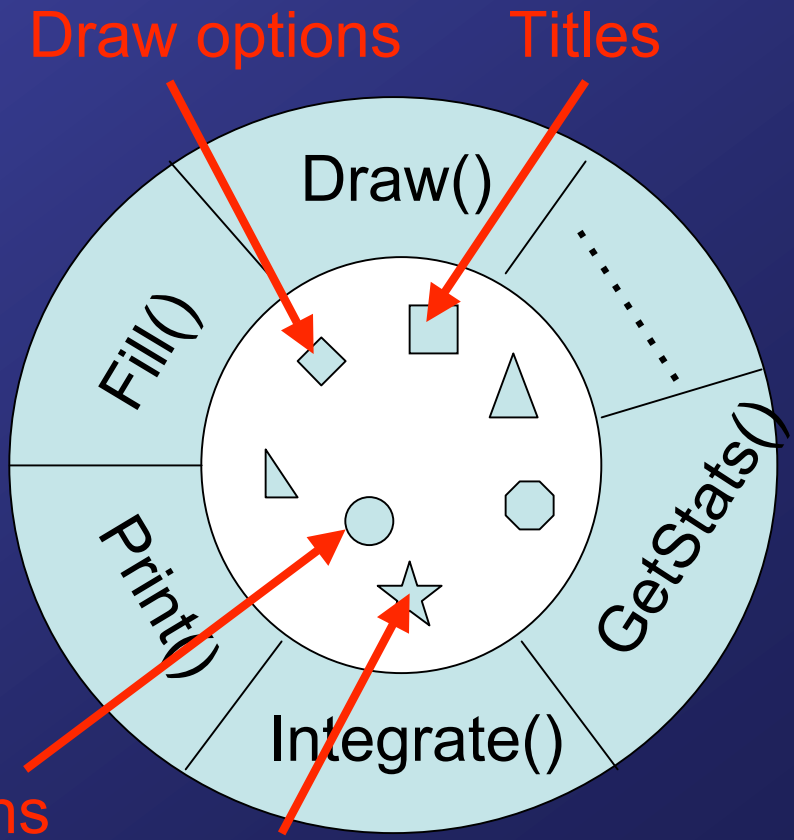
variables = data

methods = operations (interface)

An example Object



… a histogram



An abstract model
of histogram object…

Abstract Data Types (ADTs)

= collection of Methods and Variables as new data type
= prototype for an object!

C++ (=ROOT!) :

CLASS

A CLASS example

A Histogram data type :

class TH1

Public:

TH1()
~TH1()
void Fill(*arguments*)
void Draw(*arguments*)
void Print()
Double_t Integral(*arguments*)
…

Protected:

Int_t fNCells
Float_t fArray[]
TAxis fXaxis
TAxis fYaxis
…

Class name
= new Data Type

Public Methods

Creator of object

Destructor of object

Interface/Methods/
Member functions

Hidden Variables/
Data members

Data Types and Conventions

•Classes begin with **T** :	TH1, TAxis, TBrowser
•Non-class types begin with **_t** :	Int_t, Double_t, ..
•Data members begin with **f** :	fNcells, fArray, …
•Constants begin with **k** :	kInitialSize, kRed, …
•Global variables begin with **g** :	gEnv, gROOT
…for the rest see Manual	

Machine Independent Types:

Char_t	Character 1 byte	CHARACTER*1
Short_t	Short Integer 2 bytes	INTEGER*2
Int_t	Integer 4 bytes	INTEGER*4
Float_t	Float 4 bytes	REAL*4
Double_t	Float 8 bytes	REAL*8
Bool_t	Boolean (0=false, 1=true)	LOGICAL

NOTE: c++/root is case sensitive !!!

Our CLASS example

A Histogram data type:

class TH1

Public:

TH1()
~TH1()
void Fill(*arguments*)
void Draw(*arguments*)
void Print()
Double_t Integral(*arguments*)
…

Protected:

Int_t fNCells
Float_t fArray[]
TAxis fXaxis
TAxis fYaxis
…

Non-Class types

Class type TAxis:

class TAxis

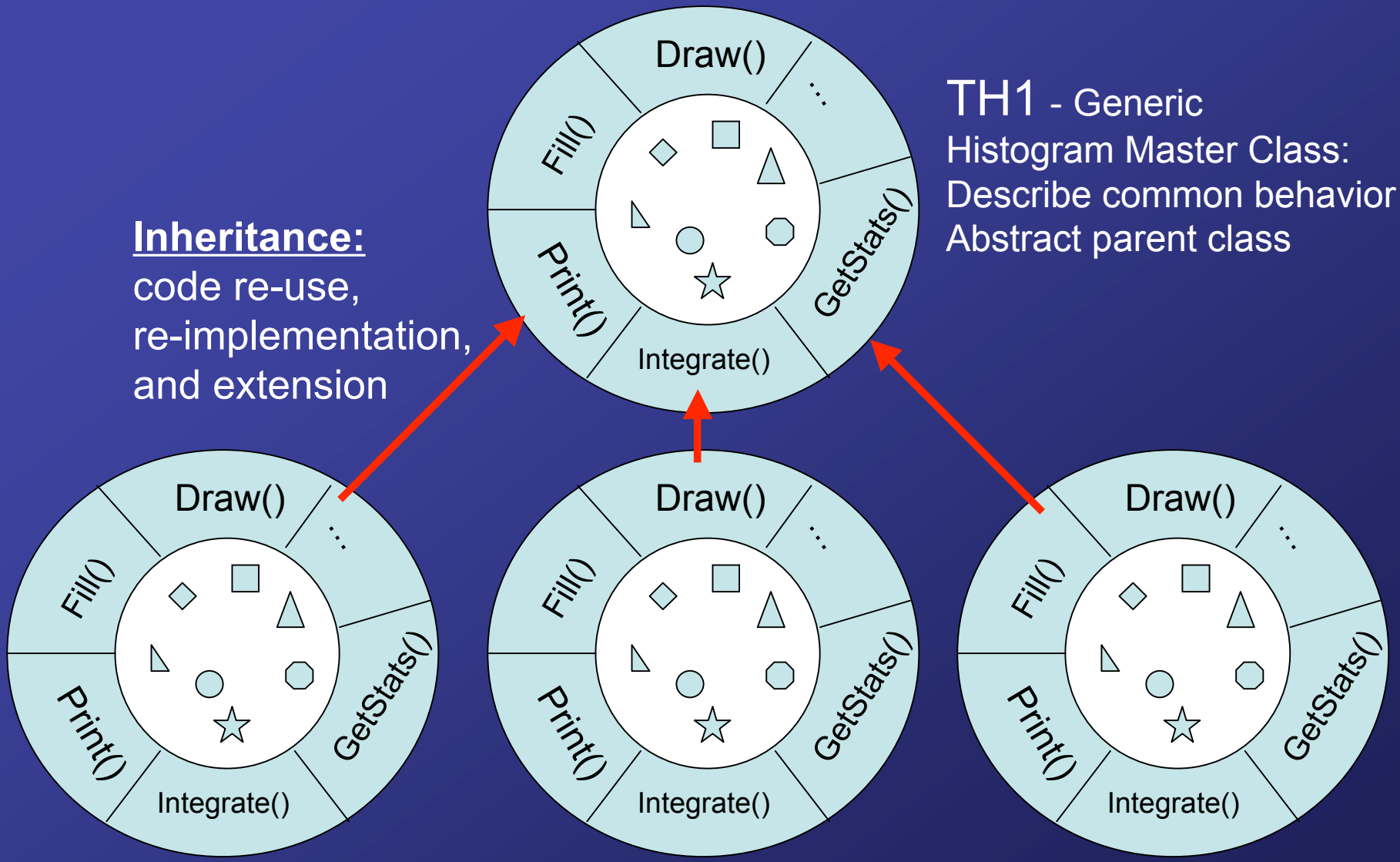
Public:

TAxis()
~TAxis()
…

Protected:

…

Even more CLASS features: Inheritance



Inheritance:
code re-use,
re-implementation,
and extension

TH1 - Generic
Histogram Master Class:
Describe common behavior
Abstract parent class

TH1F/TH1D/… -
1D Histogram

TH2 -
2D Histogram

TH3 -
3D Histogram

Why Inheritance ?

For the programmer …

1) Re-use of existing code
2) Gives a better insight in the code

For the user …

1) Provides a good overview of the software
2) Provides a common standard on how to operate a variety of different objects

-> **USER FRIENDLY!**

Inheritance in C++/ROOT

class TH1F : **public TH1**

Public:
- TH1F()
- ~TH1F()
- void Fill(…)
- Double_t Integral(…)
- …

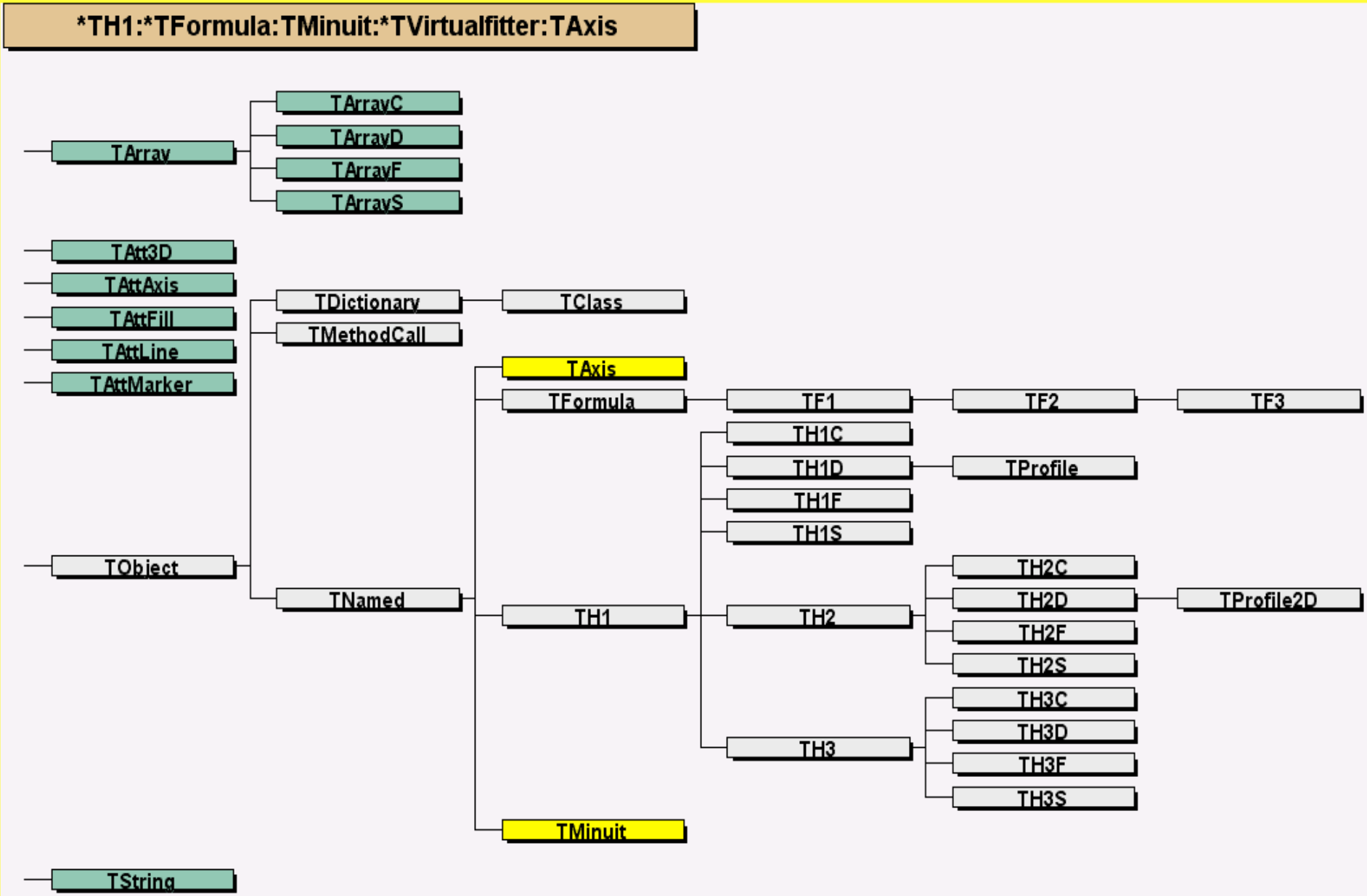
Protected:
- Int_t fNCells
- Float_t fArray[]
- …

class TH1

Public:
- TH1()
- ~TH1()
- virtual void Fill(…)
- virtual void Draw(…)
- virtual void Print()
- …

Protected:
- …

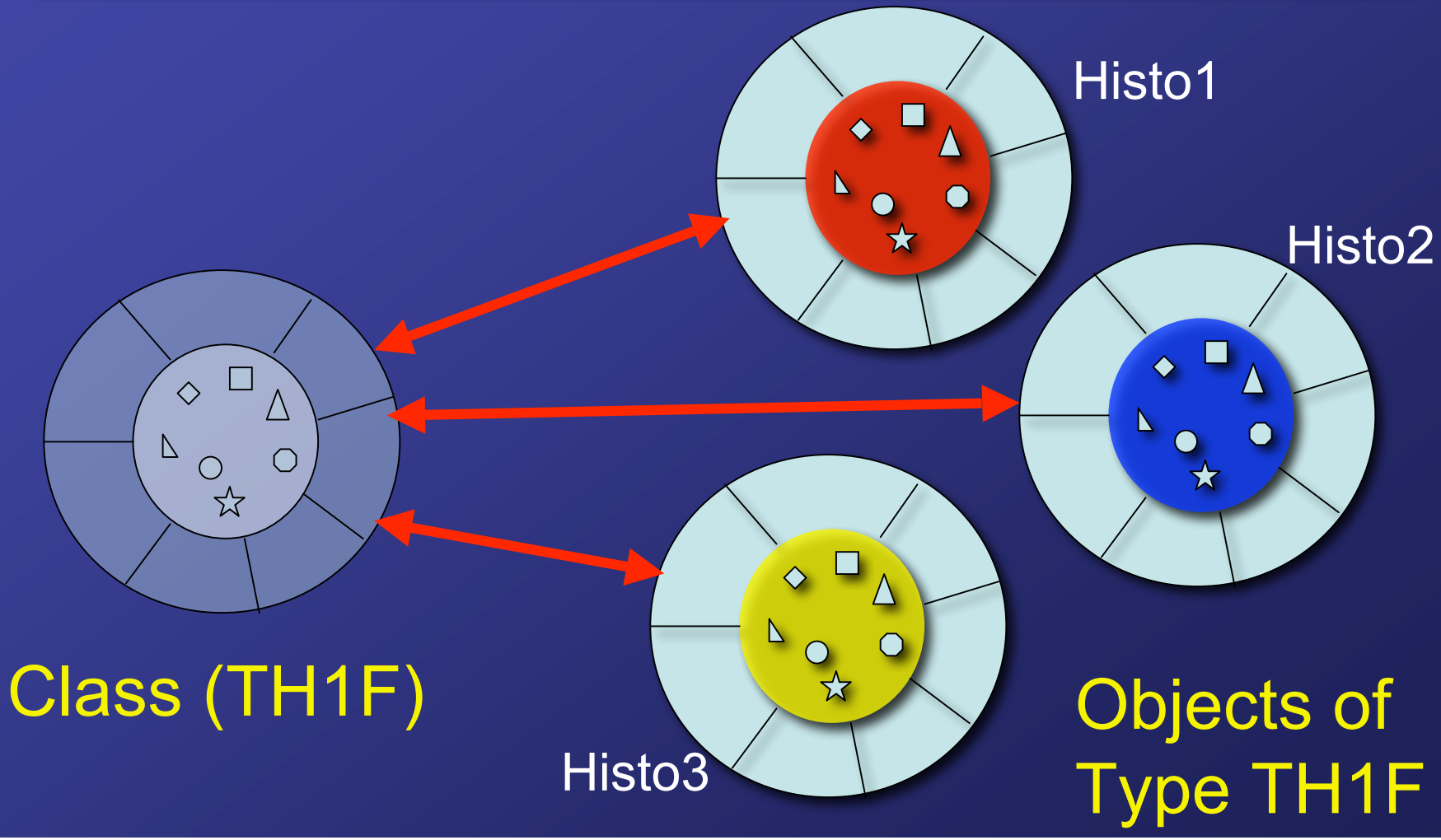
Public and protected member functions and data of TH1 become member functions and data for TH1F unless overwritten



Histogramming and Minimization Hierarchy Tree in ROOT

So what is now an object?

An object is the instance of a class



Creating a 1-D histogram object

1) Static method

TH1F myHisto(*arguments*)

Similar to :

INTEGER*2 I,J,K (Fortran)
float array[10] (C)

- Creates the static object myHisto of type TH1F
- Static: cannot be removed on-the-fly, done by the program

Creating a 1-D histogram object

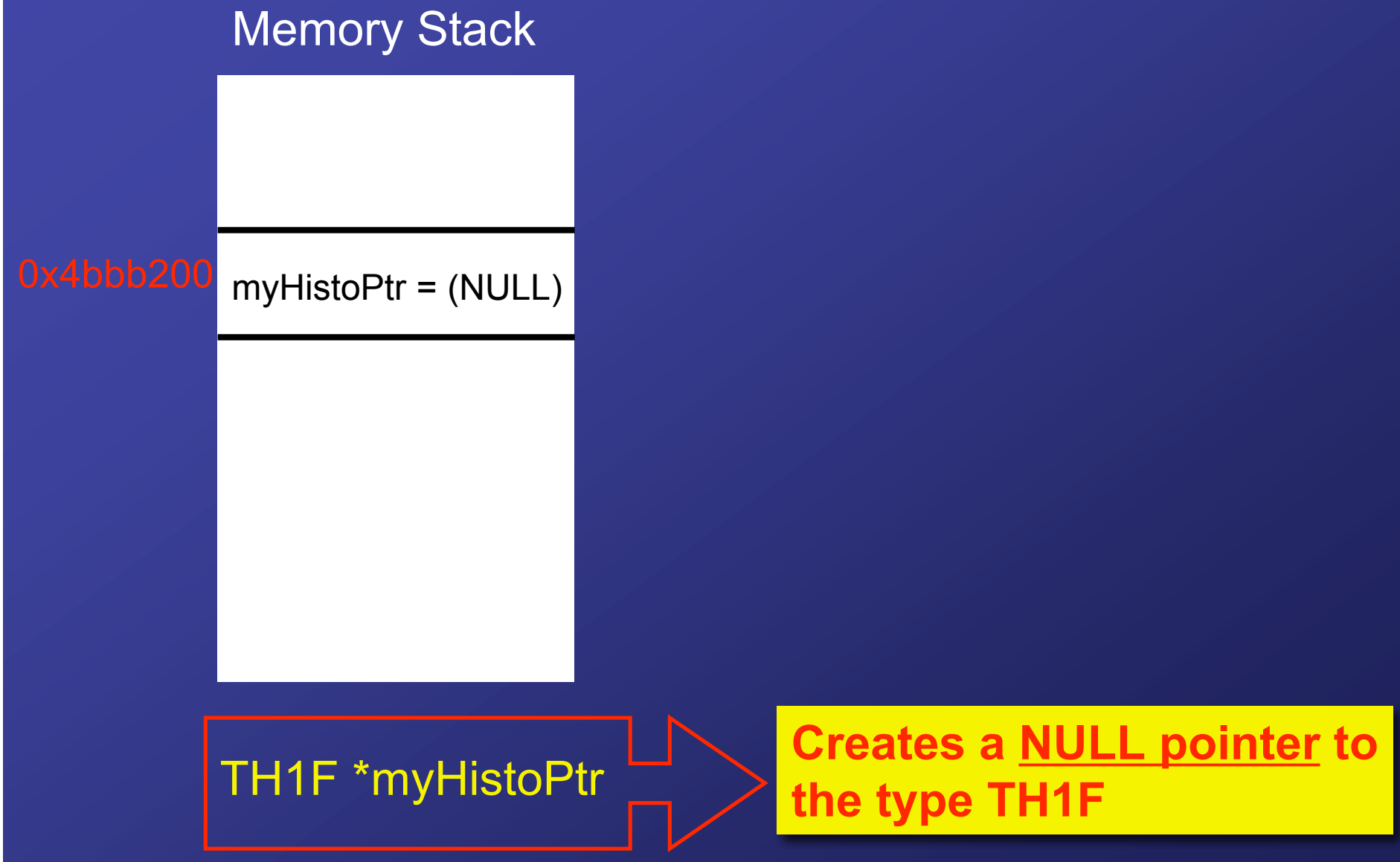
2) Dynamic method

```
TH1F *myHistoPtr ;
myHistoPtr = new TH1F(arguments)
```

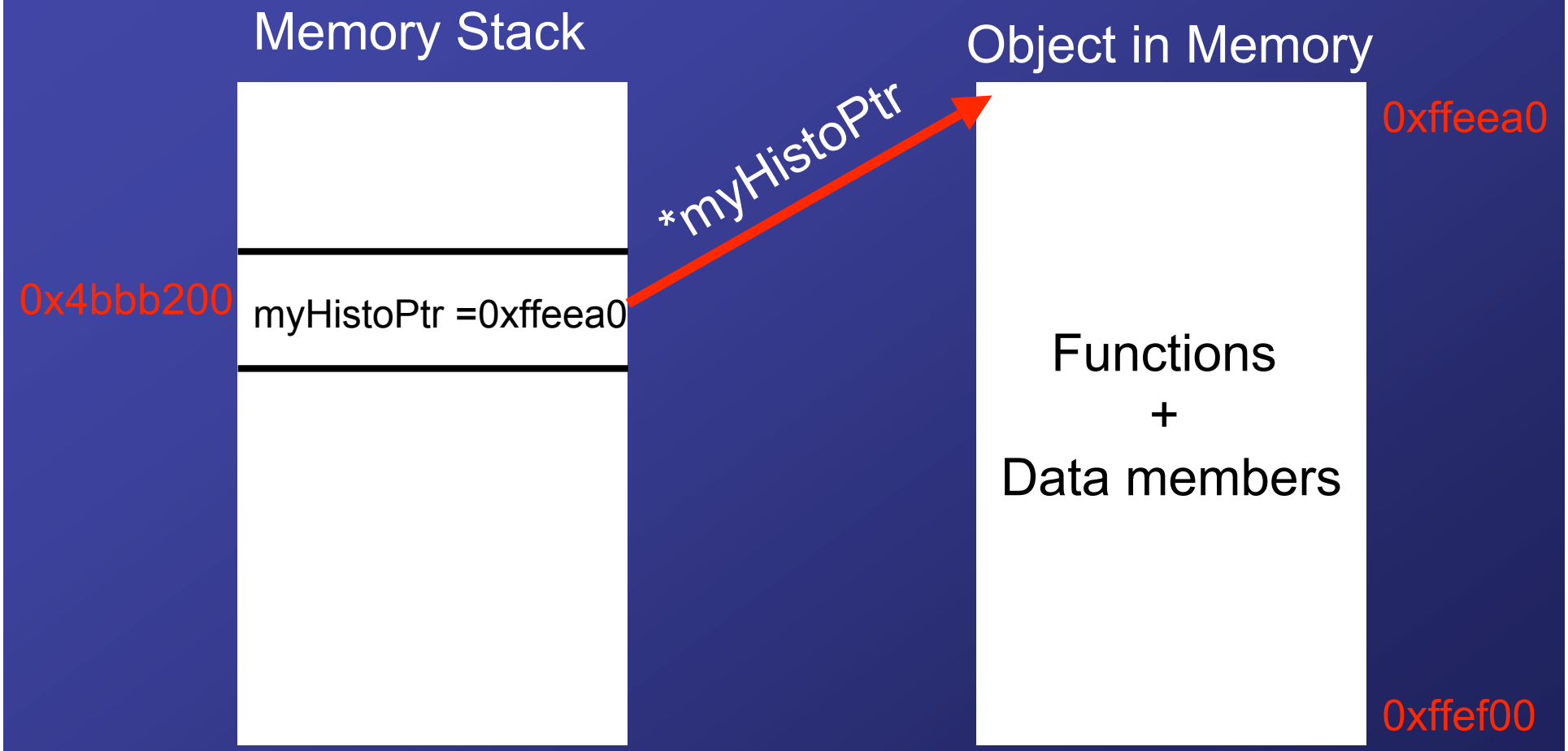
- line 1: declares myHistoPtr as a pointer (=memory address) of type TH1F
- line 2: creates the object and returns its memory address
- Dynamic: can be created/removed on-the-fly, done by YOU:

```
delete myHistoPtr
```

Pointers and all that…



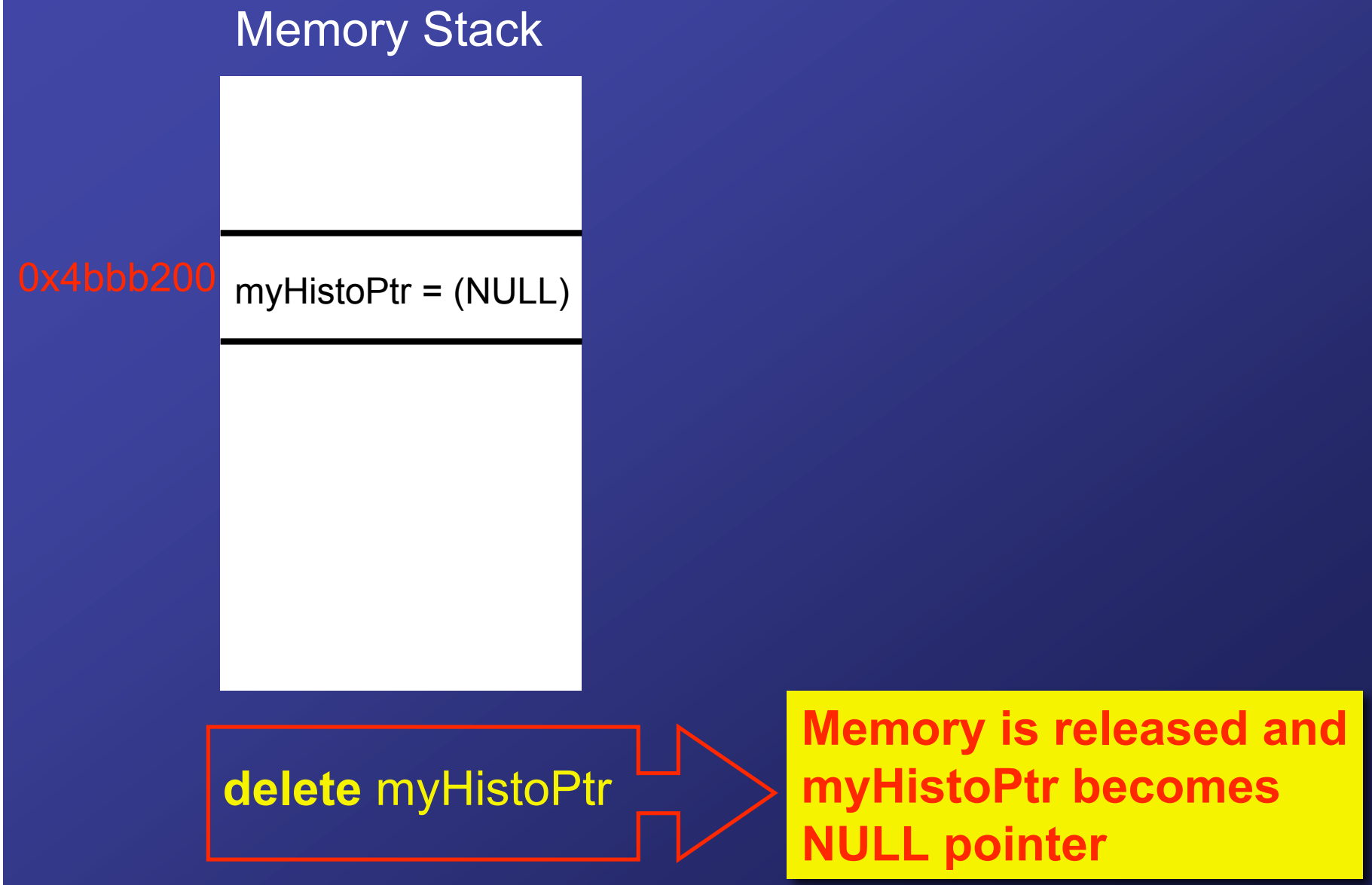
Pointers and all that…



myHistoPtr = **new** TH1F(…)

Allocates memory for object of type TH1F and return the address

Pointers and all that…



Pointers and all that…

Combined on one line :

```
TH1F *myHistoPtr = new TH1F(arguments)
```

But also … :

```
TH1F myHisto(…) ;
TH1F *myHistoPtr = &myHisto ;
```

& = address operator

```
TH1F *myHistoPtr  = new TH1F(…) ;
TH1F myHisto = *myHistoPtr ;
```

*** = dereference operator**

What are the arguments for creating an object?

class TH1F : **public** TH1

Public:

TH1F()
TH1F(char* name, char* title, Int_t nbinsx, Axis_t xlow, Axis_t xup)
TH1F(char* name, char* title, Int_t nbinsx, Float_t* xbins)
TH1F(char* name, char* title, Int_t nbinsx, Double_t* xbins)
~TH1F()
…

Pick your favorite! => creator methods are often defined in several ways => function overloading

TH1F *myHistoPtr = **new** TH1F(*"myHisto","My Title",100,0.,1.*)

How to operate on an object?

(lets say you like to fill the histogram)

```
class TH1F : public TH1

Public:
        …
        Int_t Fill(Axis_t x)
        Int_t Fill(Axis_t x, Stat_t w)
        ….
```

Also member functions are often overloaded

1) Static object:

myHisto.**Fill**(…)

use a dot!

How to operate on an object?

(lets say you like to fill the histogram)

```
class TH1F ; public TH1

Public:
        …
        Int_t Fill(Axis_t x)
        Int_t Fill(Axis_t x, Stat_t w)
        ….
```

Also member functions are often overloaded

2) Dynamic pointer object:

myHistoPtr->Fill(…)

use arrow for address pointers (“pointer to”)!

How to operate on an object?

(lets say you like to fill the histogram)

```
class TH1F : public TH1

Public:
        …
        Int_t Fill(Axis_t x)
        Int_t Fill(Axis_t x, Stat_t w)
        ….
```

Also member functions are often overloaded

3) Alternatives for the real freaks :

(&myHisto)->**Fill**(…)

(*myHistoPtr).**Fill**(…)

Pointer to pointer… an example

```
class TH1 : …

Public:
        …
        TAxis* GetXaxis()
        ….  =return (&fXaxis)
Protected:
        …
        TAxis fXaxis
        …
```

```
class TAxis : …

Public:
        …
        void SetTitle(char *name)
        ….
```

```
TH1F *myHistoPtr = new TH1F(….)
myHistoPtr->GetXaxis()->SetTitle("x-axis")
```

= (TAxis *)

Good news for those who cannot deal with pointers…

In the ROOT C/C++ interpreter you can use a pointer (->) or a dot (.), whatever you like!

… but certainly advised to stick to the rules!!

What kind of classes are there and how to operate?

1) Read the f@&#cking manual

2) Look it up on WWW …

http://root.cern.ch

A simple example code…

```
{
  TH1F       *myHis = new TH1F("myHis","A Histogram Example",100,0.,1.);
  TRandom *ranNumGen = new TRandom();
  TCanvas   *myCanvas = new TCanvas("myCanvas","This is a drawing table",1);

  myCanvas->SetFillColor(kWhite);
  myCanvas->SetFrameFillColor(kYellow);

  myHis->SetFillColor(kBlue);
  myHis->SetLineColor(kBlue);

  for (Int_t i=0; i<100000; i++)
    {
      myHis->Fill(ranNumGen->Gaus(0.5,0.2));
      if ((i%200)==0)
       {
         myHis->Draw();
         myCanvas->Refresh();
      }
    }
}
```

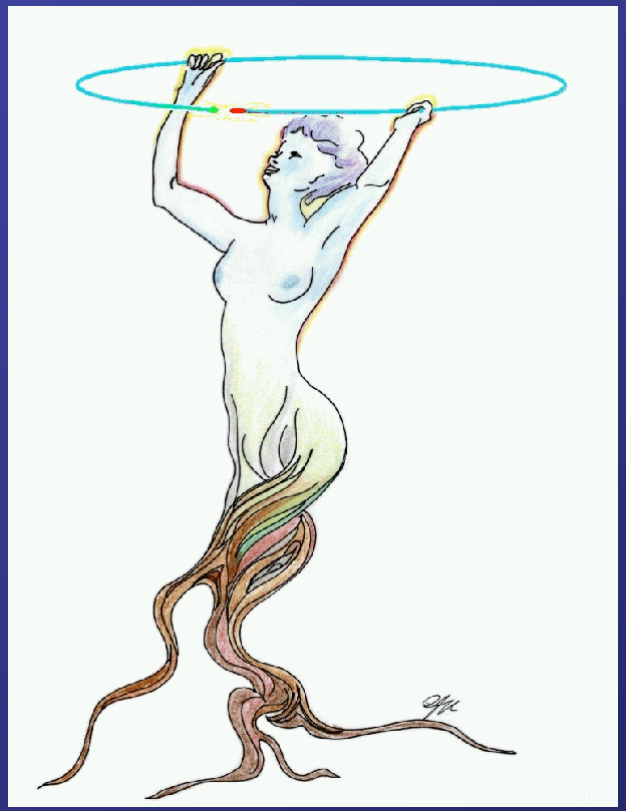
Create a few objects

Set the colors of the drawing table (canvas)

Set some colors of histogram

Fill the histogram with random numbers and update every 200 events

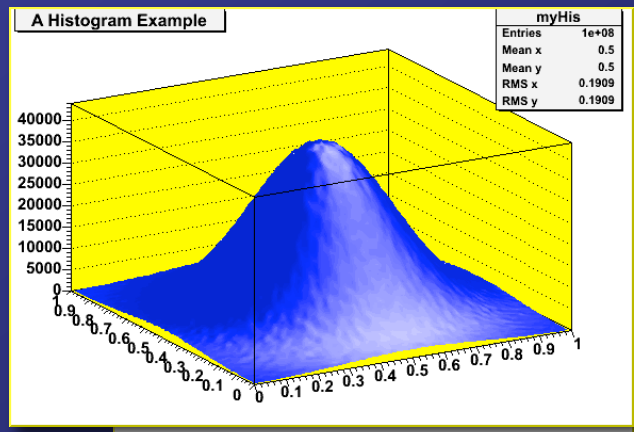
Lets try with the ROOT C/C++ interpreter…



Exercises

1) Find out which method changes the markerstyle of a 1D histogram.

2) Create a 2D histogram and fill it with a gaussian distribution.



3) Create a canvas and put a text with your name in it. Specify which classes are used and which methods and print the results as a postscript file.

Johan Messchendorp

4) Find the longest chain of pointers possible using the TH1 as base class, I.e. object->a()->b()->c()...