



# Introduction to ROOT Practical Session



**Luca Fiorini**

**IFIC Summer Student 2016**  
**July 11th 2016**

# Content

- **Practical introduction to the ROOT framework**
  - Starting ROOT
  - ROOT prompt
  - Macros– Functions
  - Histograms – Files
  - TTrees – TBrowser
  - Pyroot
- **Nomenclature**
  - **Blue: you type it**
  - **Red: you get it**

Macros and slides are in  
<http://www.ific.uv.es/~fiorini/ROOTTutorial>

# ROOT in a Nutshell

- **ROOT is a large Object-Oriented data handling and analysis framework**
  - **Efficient object store scaling from kB's to PB's**
- **C++ interpreter**
- **Extensive 2D+3D scientific data visualization capabilities**
- **Extensive set of multi-dimensional histogramming, data fitting, modeling and analysis methods**
- **Complete set of GUI widgets**
- **Classes for threading, shared memory, networking, etc.**
- **Parallel version of analysis engine runs on clusters and multi-core**
- **Fully cross platform: Unix/Linux, MacOS X and Windows**

# ROOT in a Nutshell (2)

- The user interacts with ROOT via a graphical user interface, the command line or scripts
- The command and scripting language is C++
  - Embedded C++ interpreter CINT (ROOT5)/ CLING (ROOT6)
  - Large scripts can be compiled and dynamically loaded

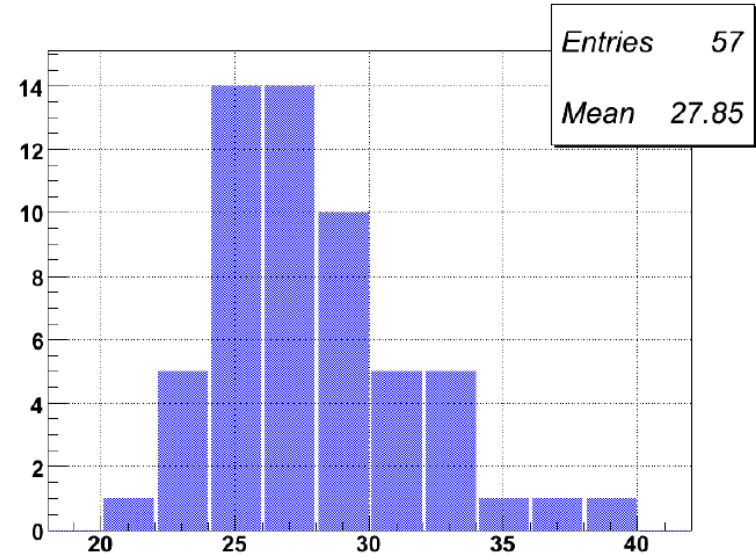
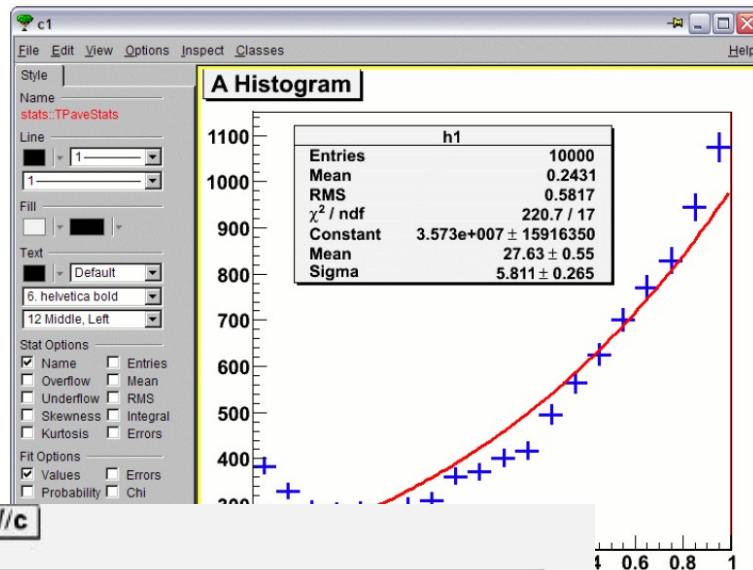
**And for you?**

**ROOT is usually the interface (and sometimes the barrier) between you and the data**

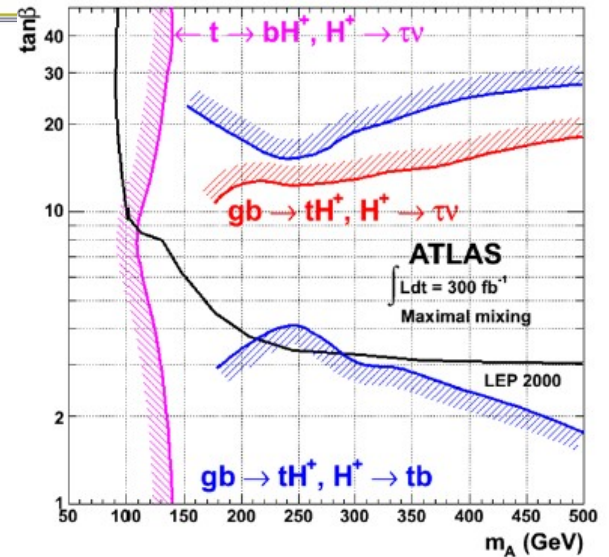
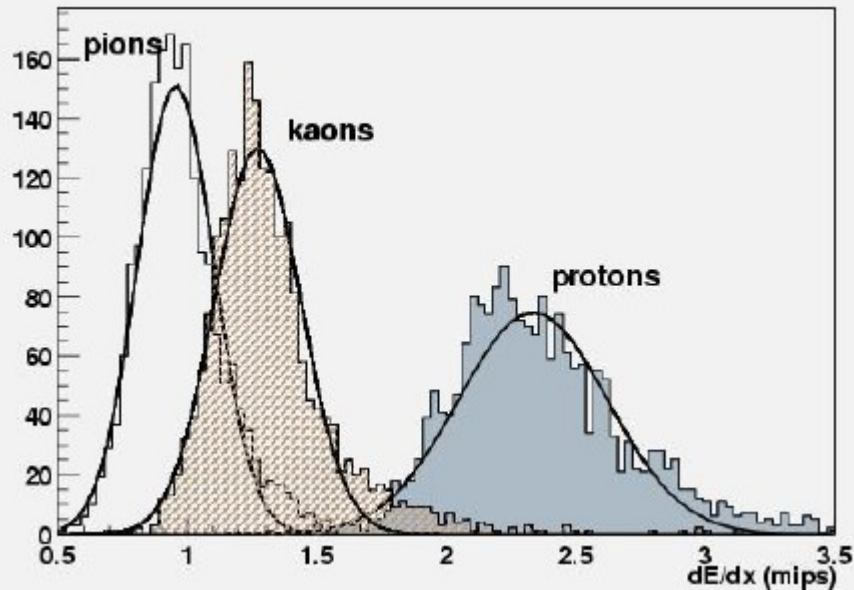
# ROOT: An Open Source Project

- **The project was started in Jan 1995**
- **First release Nov 1995**
- **The project is developed as a collaboration between:**
  - **Full time developers:**
    - 7 people full time at CERN (PH/SFT)
    - 2 developers at Fermilab/USA
  - **Large number of part-time contributors (160 in CREDITS file)**
  - **A long list of users giving feedback, comments, bug fixes and many small contributions**
    - 5,500 users registered to RootTalk forum
    - 10,000 posts per year
- **An Open Source Project, source available under the LGPL license**
- **Used by all major HEP experiments in the world**
- **Used in many other scientific fields and in commercial world**

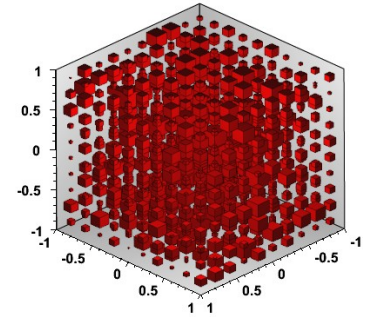
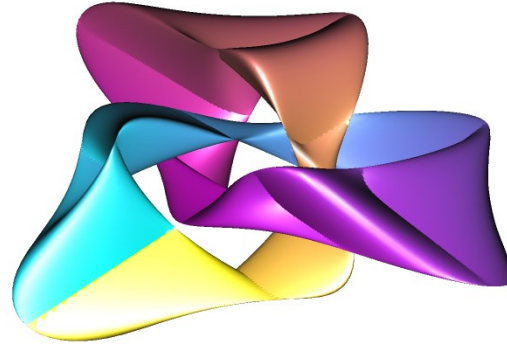
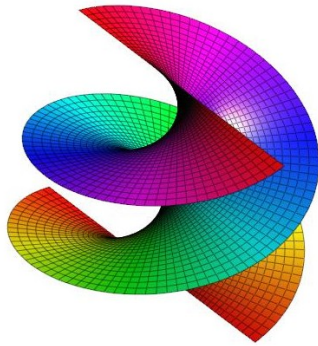
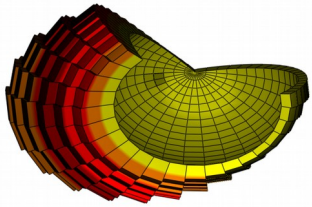
# ROOT: Graphics



Momentum 730-830 MeV/c

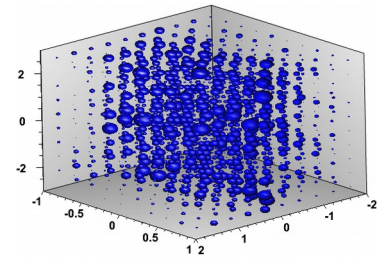
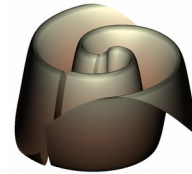
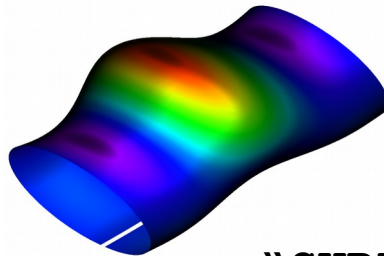
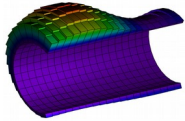
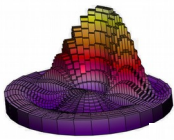
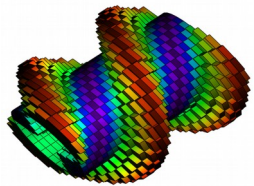


# ROOT: Graphics

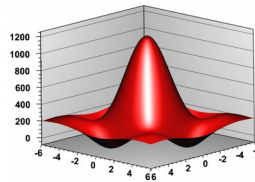
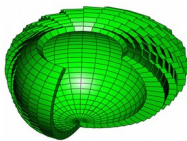
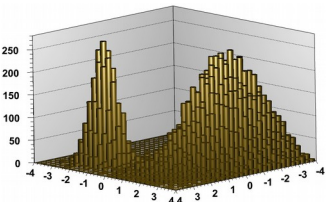


TH3

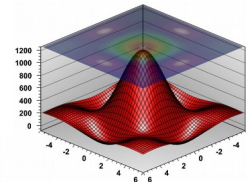
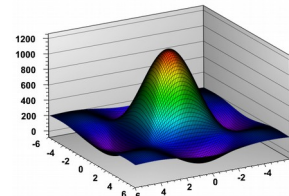
TGLParametric



"LEGO"

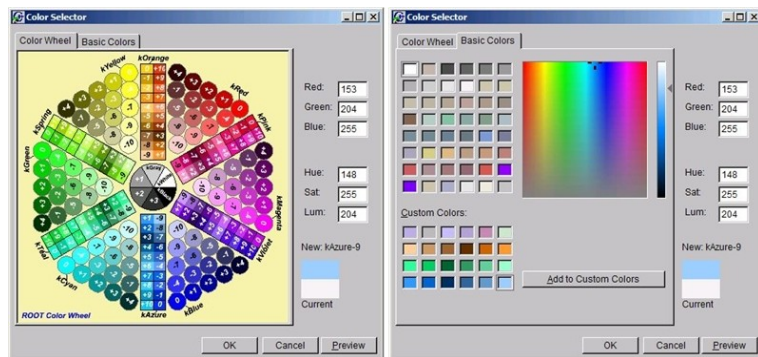
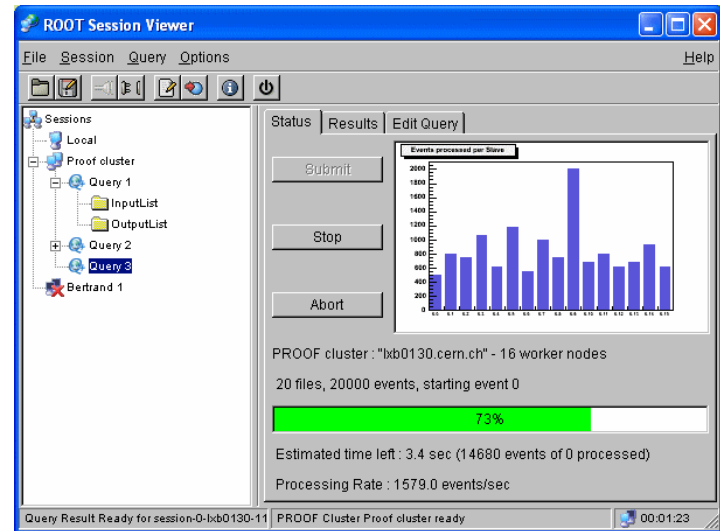
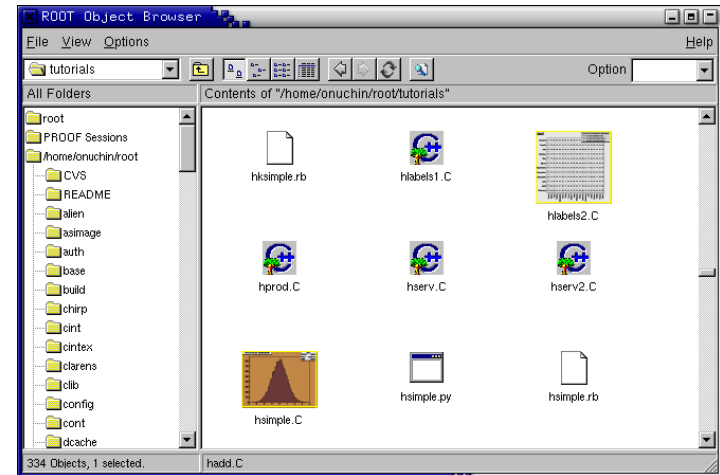
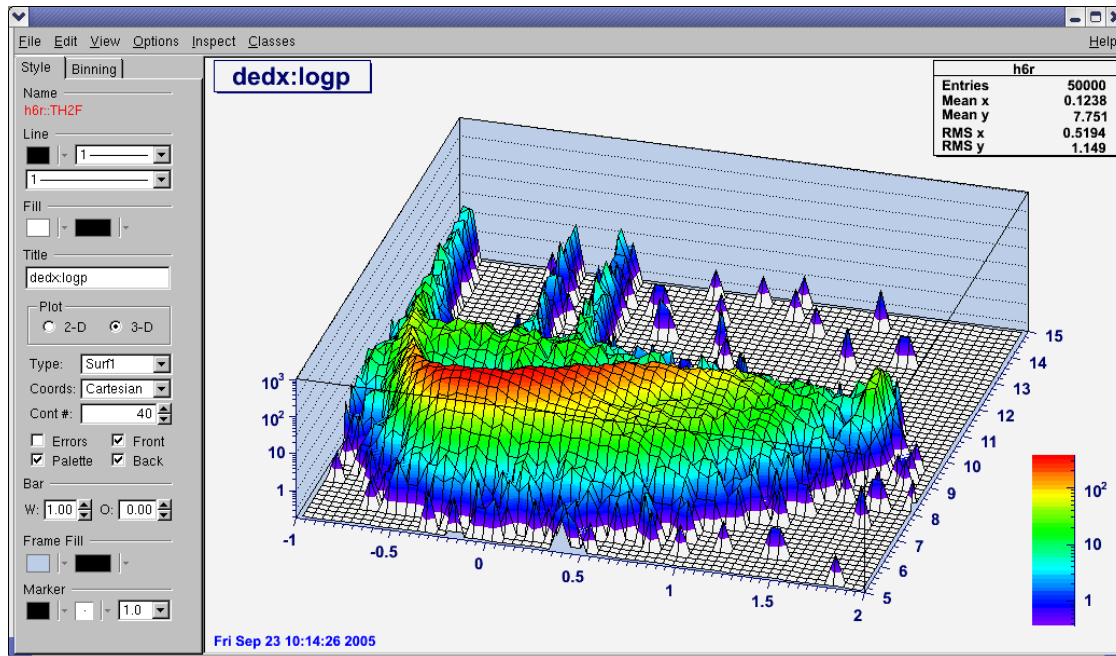


"SURF"



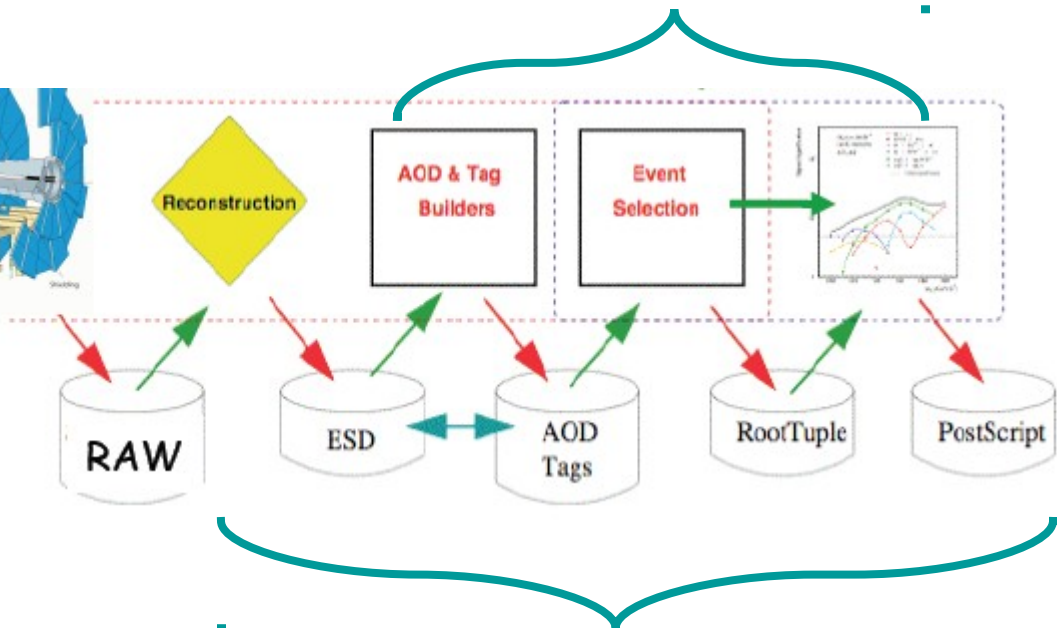
TF3

# ROOT: Graphical Interfaces



# ROOT Application Domains

## Data Analysis & Visualization



## Data Storage: Local, Network

General Framework

# ROOT Download & Installation

[Download](#) [Documentation](#) [News](#) [Support](#) [About](#) [Development](#) [Contribute](#)



Getting Started



Reference Guide



Forum

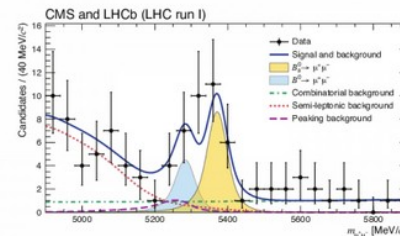


Gallery

## ROOT is ...

A modular scientific software framework. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python and R.

[Try it in your browser! \(Beta\)](#)



[Previous](#) [Pause](#) [Next](#)

- <http://root.cern.ch>
  - Binaries for common Linux PC flavors, Mac OS, Windows (ROOT5)
- Source files

Before Installing ROOT, add dependencies, discussed here:

<https://root.cern.ch/build-prerequisites>

- Linux and MacOS: ROOT6 preferred
- Windows: ROOT5

Installation guide at:

<https://root.cern.ch/installing-root-source>

If nothing works:  
<http://root.cern.ch/notebooks/rootbinder.html>

# ROOT Resources

- **Main ROOT page**
  - <http://root.cern.ch>
- **Class Reference Guide**
  - <http://root.cern.ch/root/html>
- **C++ tutorial**
  - <http://www.cplusplus.com/doc/tutorial/>
  - <http://www-root.fnal.gov/root/CPlusPlus/index.html>
- **Hands-on tutorials:**  
<https://root.cern.ch/courses>

# ROOT Prompt

- Starting ROOT

**\$ root**

**\$ root -l** (without splash screen)

**\$ root -h**

- The ROOT prompt

**root [ ] 2+3**

**root [ ] int i = 42**

**root [ ] log(5)**

**root [ ] cout << i << endl;**

**root [ ] TMath::Pi() // try to type also TMath::Pi**

- Command history

- Scan through with arrow keys ↑↓
- Search with CTRL-R (like in bash)

- Built-in commands:

**root [ ] .? //or .help**

- Online help

**root [ ] new TF1(<TAB>**

**TF1 TF1()**

**TF1 TF1(const char\* name, const char\* formula, Double\_t xmin = 0,  
Double\_t xmax = 1)**

...

# ROOT Prompt (2)

- Typing multi-line commands

```
root [ ] for (i=10; i>0; i--) {cout << i <<  
    endl;}; cout << "BOOM!!" << endl;
```

or

```
root [ ] for (i=0; i<3; i++) {  
end with '}', '@':abort > printf("%d\n",  
    i);  
end with '}', '@':abort > }
```

- Aborting wrong input

```
root [ ] printf("%d\n", i)  
end with ';', '@':abort > @
```

Don't panic!

Don't press CTRL-C!

Just type @ or .@

# ROOT Macros

- It is quite cumbersome to type the same lines again and again
- Create macros for commonly used code
- Macro = file that is interpreted by CINT/CLING

```
int myfirstmacro(int value)
{
    int ret = 42;
    ret += value;
    return ret;
}
```

—————→ save as myfirstmacro.C

- Execute with **root [0] .x myfirstmacro.C(10)**
- Or **root [0] .L myfirstmacro.C**  
**root [1] myfirstmacro(10)**

# Macros

- Combine lines of codes in macros
- Unnamed macro
  - No parameters

For example: macro1.C

```
{  
  TRandom r;  
  for (Int_t i=0; i<10; i++) {  
    cout << r.Rndm() << endl;  
  }  
  for (Int_t i=0; i<100000; i++) {  
    r.Rndm();  
  }  
}
```

- Executing macros

```
root [ ] .x macro1.C
```

```
$ root -l macro1.C
```

```
$ root -l -b macro1.C (batch mode → no graphics)
```

```
$ root -l -q macro1.C (quit after execution)
```

## Data types in ROOT

Int\_t (4 Bytes)

Long64\_t (8 Bytes)

...

to achieve platform-independency

# Compile Macros – Libraries

- "Library": compiled code, shared library
- CINT/CLING can call its functions!
- Building a library from a macro: ACLiC  
(Automatic Compiler of Libraries for CINT)
- Execute it with a “+”

```
root [0] .x myfirstmacro.C(42)+
```

- Or

```
root [0] .L myfirstmacro.C+
```

```
root [1] myfirstmacro(42)
```

- No Makefile needed
- CINT knows all functions in the library  
mymacro\_C.so/.dll

# Compiled vs. Interpreted

- **Why compile?**
  - Faster execution, CINT/CLING has some limitations...
- **Why interpret?**
  - Faster Edit → Run → Check result → Edit cycles ("rapid prototyping"). Scripting is sometimes just easier
- **So when should I start compiling?**
  - For simple things: start with macros
  - Rule of thumb
    - Is it a lot of code or running slow? → Compile it!
    - Does it behave weird? → Compile it!
    - Is there an error that you do not find → Compile it!

# Functions

- The class TF1 allows to draw functions

```
root [ ] f = new TF1("func", "sin(x)", 0, 10)
```

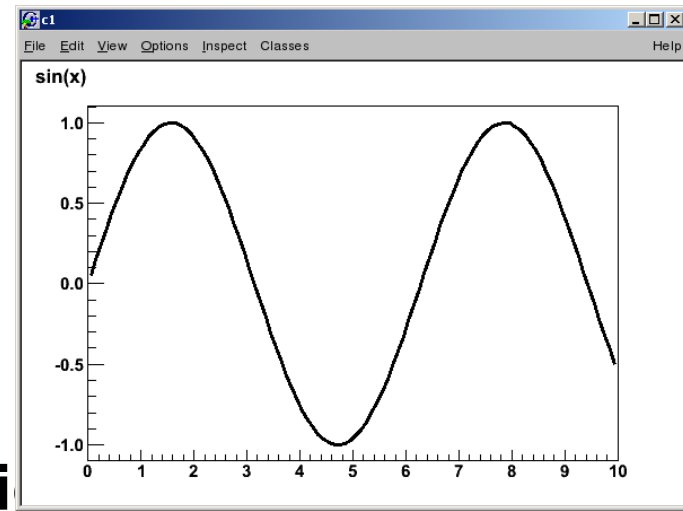
- "func" is a (unique) name
- "sin(x)" is the formula
- 0, 10 is the x-range for the function

```
root [ ] f->Draw()
```

- The style of the function can be changed on the command line or with the context menu (→ right cli

```
root [ ] f->SetLineColor(kRed)
```

- The class TF2(3) is for 2(3)-dimensional functions



↑  
Canvas

# Pointers vs. Value Types

- A value type contains an instance of an object
- A pointer *points* to the instance of an object
- Create a pointer

```
root [ ] TF1* f1 = new TF1("func", "sin(x)", 0, 10)
```

- Create a value type

```
root [ ] TF1 f2("func", "cos(x)", 0, 10)
```

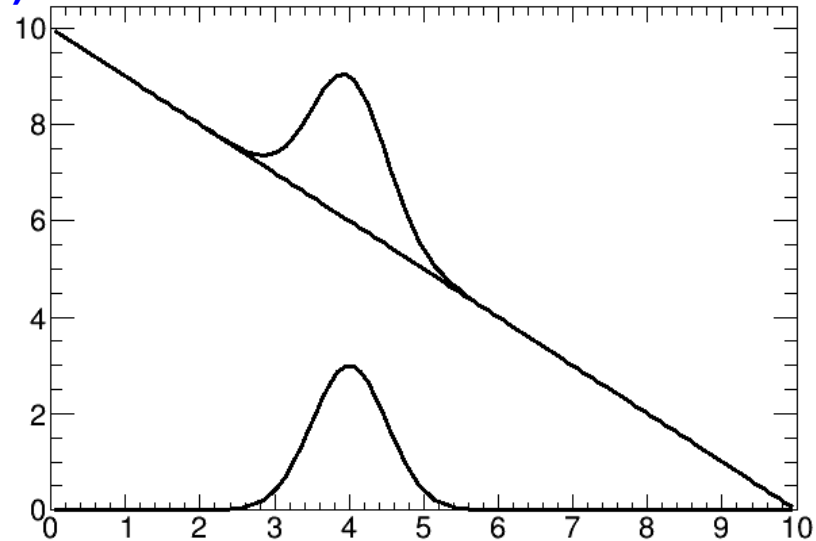
- One can point to the other

```
TF1 f1b(*f1)           // dereference and create a copy
```

```
TF1* f2b = &f2         // point to the same object
```

# Functions

```
root [ ] TF1 *f1 = new TF1("f1","gaus(x)",0,10)
root [ ] TF1 *f2 = new TF1("f2","10.-x",0,10)
root [ ] f2->SetParameter(0,1)
root [ ] f2->Draw()
root [ ] f1->SetParameter(0,2)
root [ ] f1->SetParameter(1,4)
root [ ] f1->SetParameter(2,2.5)
root [ ] f1->Draw()
root [ ] TF1 *f3 = new TF1("f3","f1+f2",0,10)
root [ ] f3->Draw()
root [ ] f3->SetParameter(0,3)
root [ ] f3->SetParameter(2,0.5)
root [ ] f3->Draw()
root [ ] f2->Draw("same")
root [ ] f1->SetParameter(0,3)
root [ ] f1->SetParameter(2,0.5)
root [ ] f1->Draw("same")
```



- Now play a bit with the function class and graphical options.
- Can you change the background shape from a linear function to an exponential function?
- How to save the graphical window (it is called Canvas)?
- code in **function.C**

# Histograms

- Contain binned data – probably the most important class in ROOT for the physicist
- Create a TH1F (= one dimensional, float precision)

```
root [ ] h = new TH1F("hist", "my hist;Bins;Entries", 10, 0, 10)
```

- "hist" is a (unique) name
- "my hist;Bins;Entries" are the title and the x and y labels
- 10 is the number of bins
- 0, 10 are the limits on the x axis.  
Thus the first bin is from 0 to 1, the second from 1 to 2, etc.

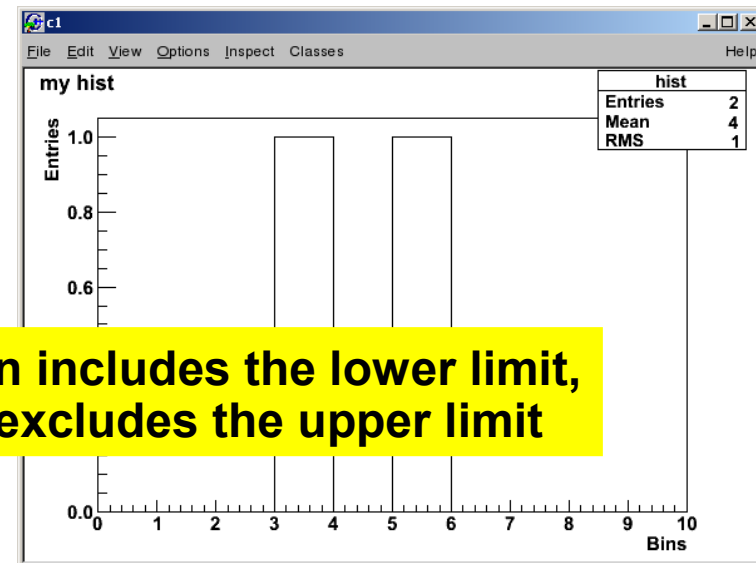
- Fill the histogram

```
root [ ] h->Fill(3.5)
```

```
root [ ] h->Fill(5.5)
```

- Draw the histogram

```
root [ ] h->Draw()
```



# Histograms (2)

```
root [ ] TH1F h("h","h",80,-40,40)
root [ ] TRandom r;
root [ ] for (i=0;i<15000;i++) { h.Fill(r.Gaus(0,7));}
root [ ] h.Draw()
```

- **Rebinning**

```
root [ ] h.Rebin(2)
```

- **Change ranges/canvas**

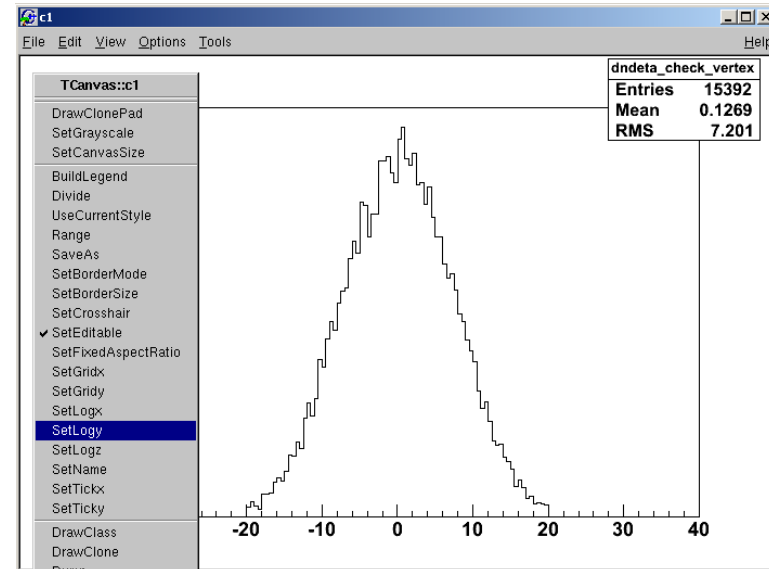
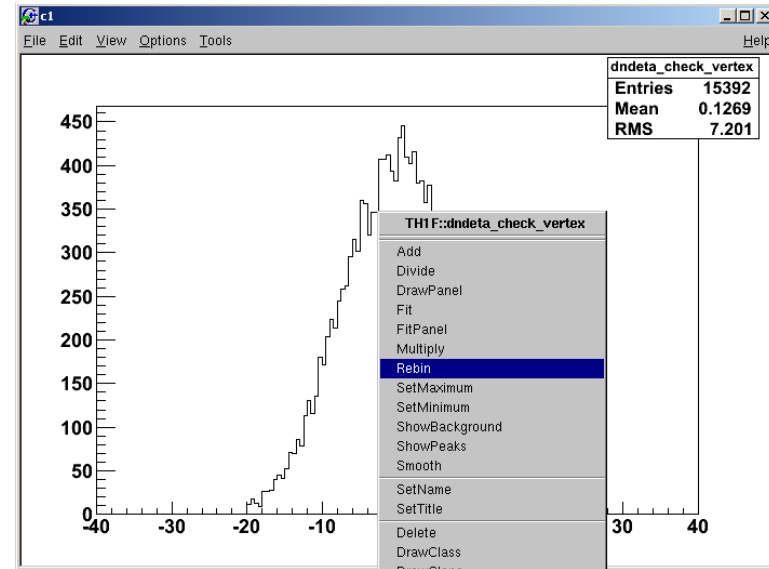
- with the mouse, very easy!
- with the context menu
- command line

```
root [ ] h.GetXaxis()->
SetRangeUser(2, 5)
```

- **Log-view**

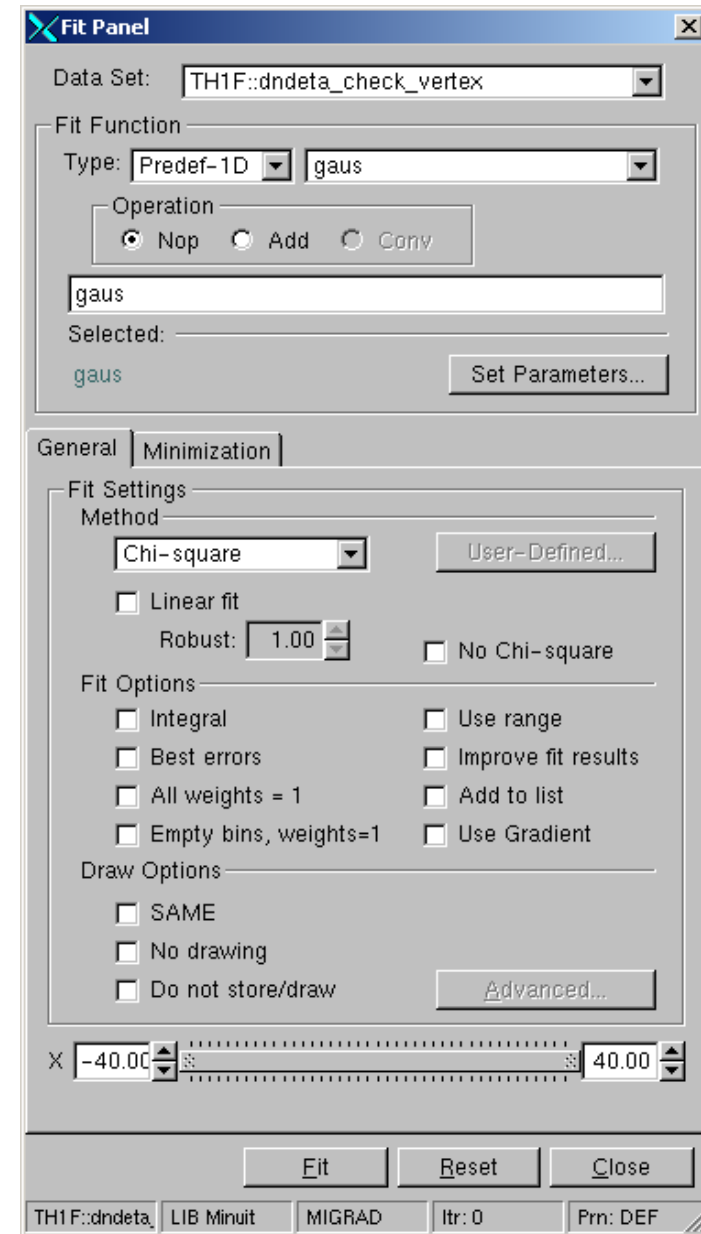
- right-click in the white area at the side of the canvas and select SetLogx (SetLogy)
- command line

```
root [ ] gPad->SetLogy()
```

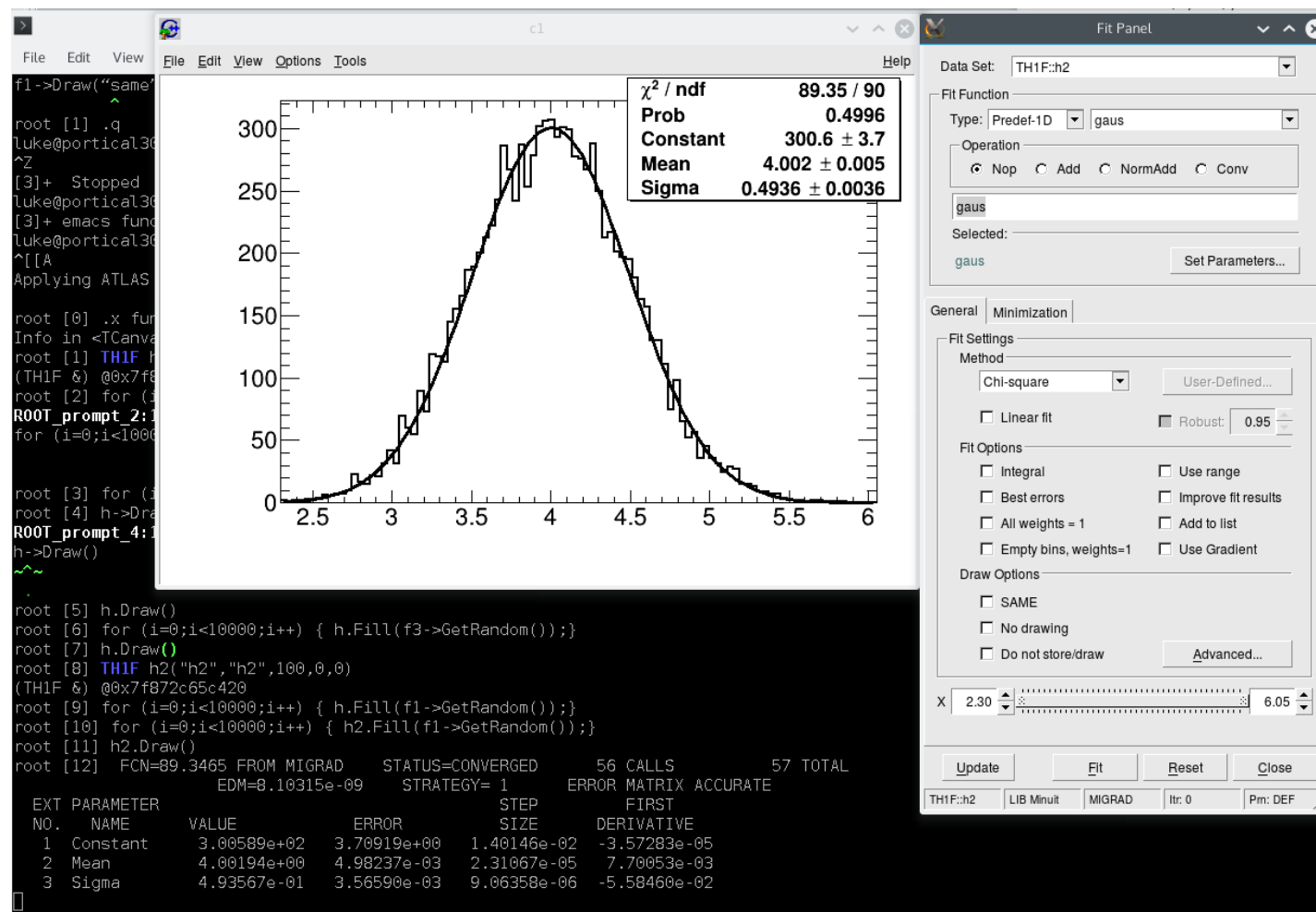


# Fitting Histograms

- **Interactive**
  - Right click on the histogram and choose "fit panel"
  - Select function and click fit
  - Fit parameters
    - are printed in command line
    - in the canvas: options - fit parameters
- **Command line**  
`root [ ] h.Fit("gaus")`
  - Other predefined functions  
polN (N = 0..9), expo, landau
- **Try to fit the histogram with different functions.**



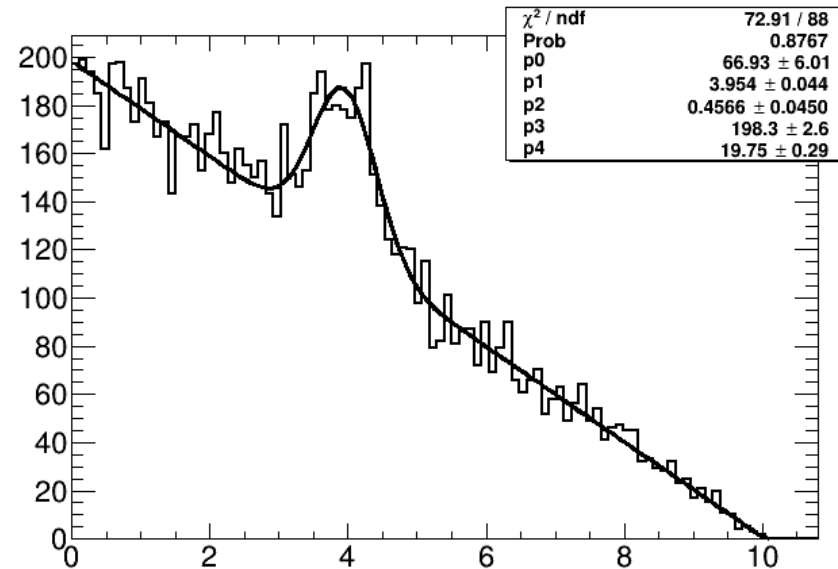
# Fitting Histograms



# Fitting Histograms (2)

- Now edit function.C

```
root [ ] TH1F
h("h","h",100,0,0); //auto range
root [ ] for (i=0;i<10000;i++)
{ h.Fill(f3->GetRandom());}
root [ ] //create random
numbers according to a
function distribution
root [ ] h.Draw()
```



- Try to fit the the histogram:

```
root [ ] TF1* f4 = new TF1("f4",".....",0,10)
```

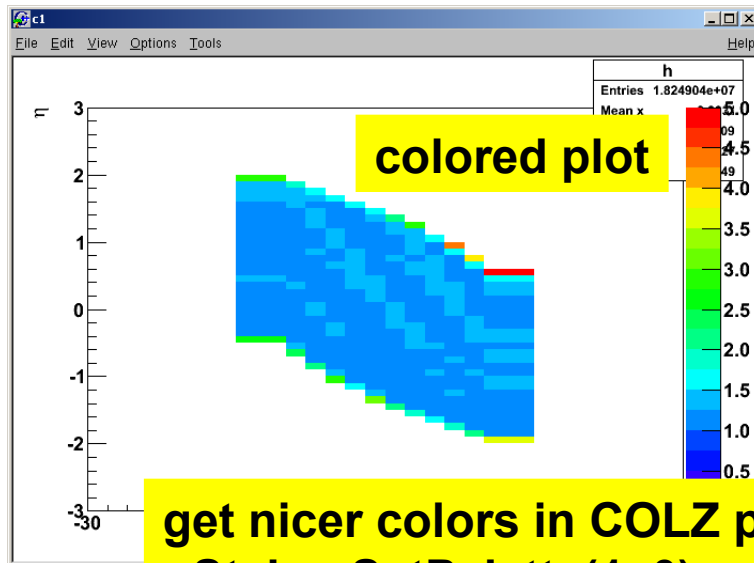
- Tip: A Gaussian function can be written as:  
$$[0] * TMath::Exp( -0.5 * ((x-[1])/[2]) * ((x-[1])/[2]) )$$

# 2D Histograms

```
root [ ] h->Draw()
```

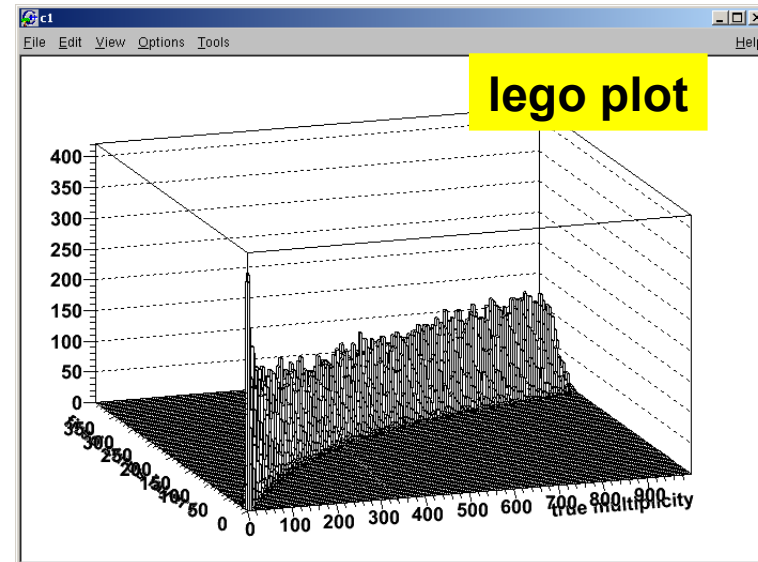
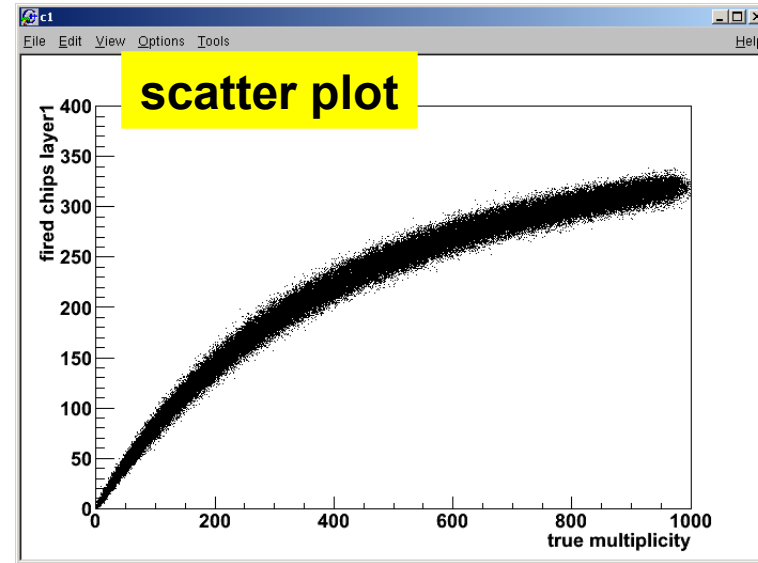
```
root [ ] h->Draw("LEGO")
```

```
root [ ] h2->Draw("COLZ")
```



get nicer colors in COLZ plots by  
`gStyle->SetPalette(1, 0)`

NB: h and h2 are in file **hist2.root**



# Files

- The class TFile allows to store any ROOT object on the disk

- Create a histogram like before with

```
TH1F* h = new TH1F("h", "my hist;...", 10, 0, 10)
```

```
TH1F hist("hist", "test", 100, -3, 3);
```

```
hist.FillRandom("gaus", 1000);
```

"hist" will be the name in the file

etc.

- Open a file for writing

```
root [ ] file = TFile::Open("file.root", "RECREATE")
```

- Write an object into the file

```
root [ ] h->Write()
```

```
root [ ] hist->Write()
```

- Close the file

```
root [ ] file->Close()
```

NEW  
READ  
RECREATE  
UPDATE  
.....

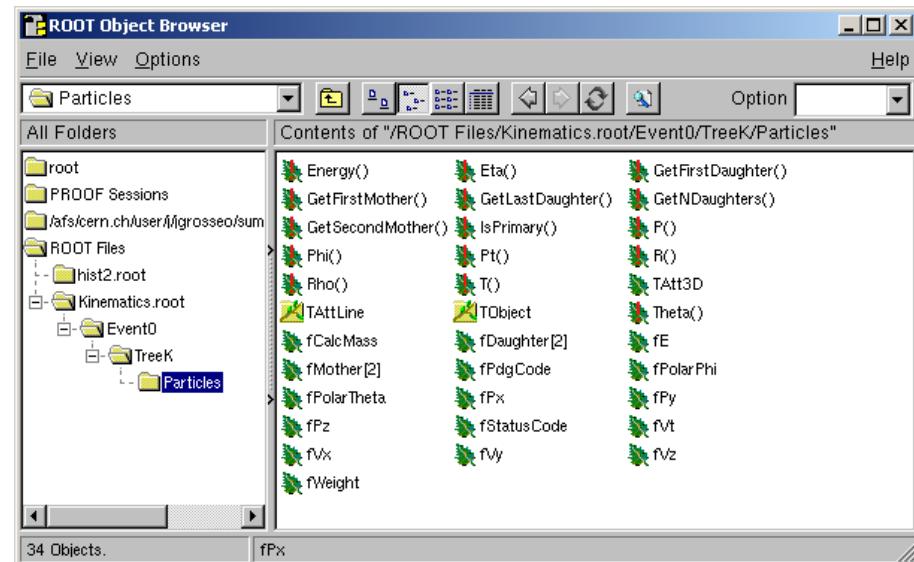
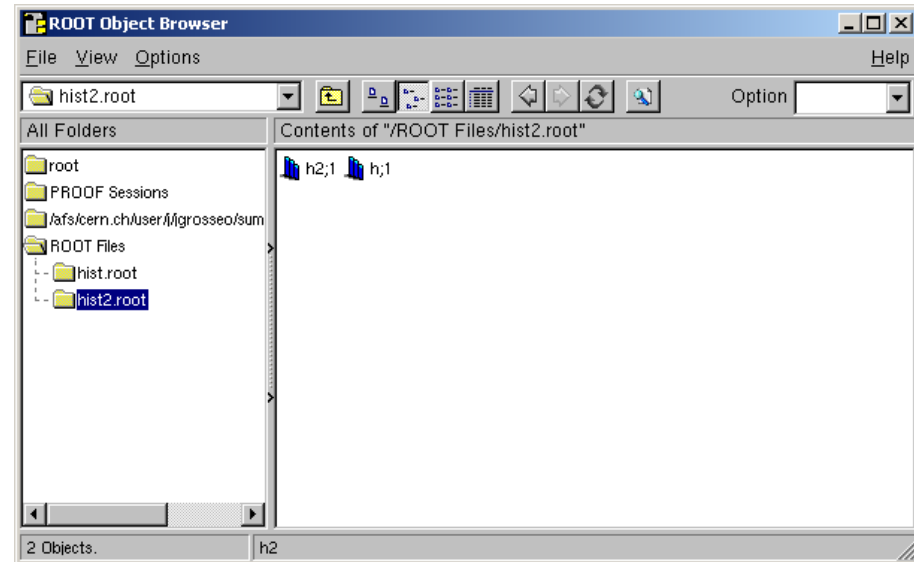
# Files (2)

- Open the file for reading  
`root [ ] file = TFile::Open("file.root")`
- Read the object from the file  
`root [ ] hist->Draw()`  
(only works on the command line!)
- In a macro read the object with  
`TH1F* h = 0;`  
`file->GetObject("hist", h);`
- What else is in the file?  
`root [ ] .ls`
- Open a file when starting root  
`$ root file.root`
  - Access it with the `_file0` or `gFile` pointer

▮ **Object ownership**  
After reading an object from a file don't close it! Otherwise your object is not in memory anymore

# TBrowser

- The TBrowser can be used
  - to open files
  - navigate in them
  - to look at TTrees
- Starting a TBrowser  
**root [ ] new TBrowser**
- Open a file
- Navigate through the file
- Draw a histogram
- Change the standard style
  - Drop down menu in the top right corner
- Access a tree
- Plot a member

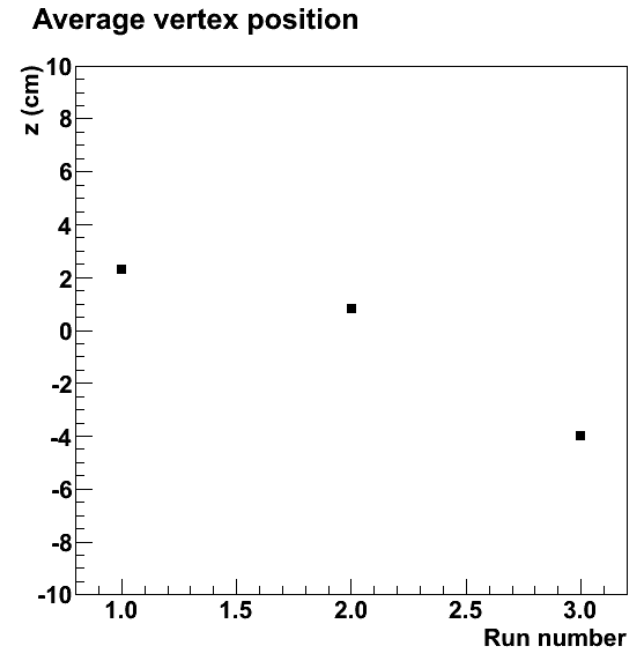


# Graphs

- A graph is a data container filled with distinct points
- TGraph: x/y graph without error bars
- TGraphErrors: x/y graph with error bars
- TGraphAsymmErrors: x/y graph with asymmetric error bars

## Graph Example

```
graph = new TGraph;  
graph->SetPoint(graph->GetN(), 1, 2.3);  
graph->SetPoint(graph->GetN(), 2, 0.8);  
graph->SetPoint(graph->GetN(), 3, -4);  
graph->Draw("AP");  
graph->SetMarkerStyle(21);  
graph->GetYaxis()->SetRangeUser(-10, 10);  
graph->GetXaxis()->SetTitle("Run number");  
graph->GetYaxis()->SetTitle("z (cm)");  
graph->SetTitle("Average vertex position");
```



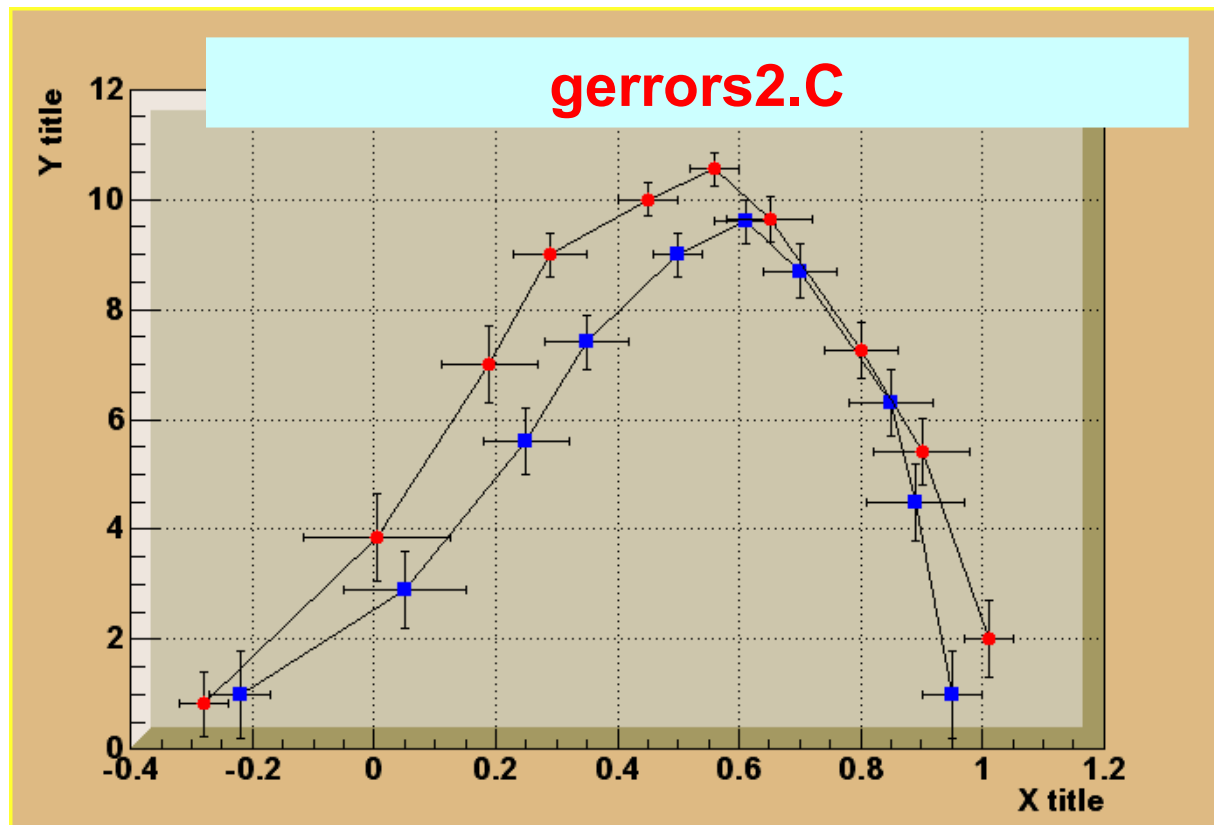
# Graphs (2)

**TGraphErrors(n,x,y,ex,ey)**

**TGraph(n,x,y)**

**TCutG(n,x,y)**

**TMultiGraph**



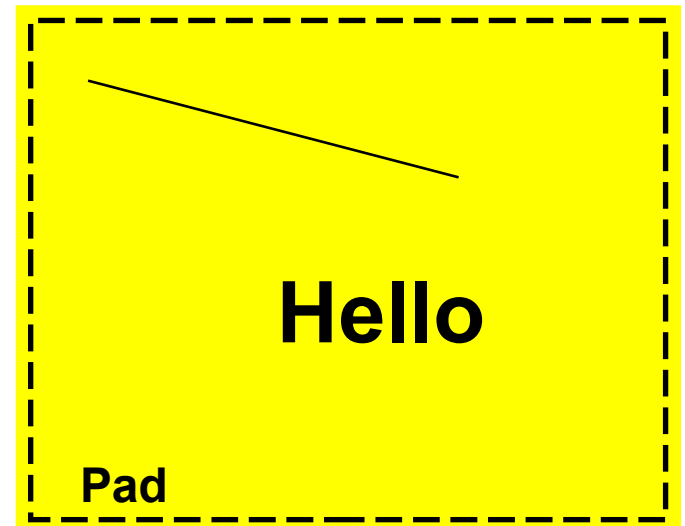
**TGraphAsymmErrors(n,x,y,exl,exh,eyl,eyh)**

# Graphics Objects

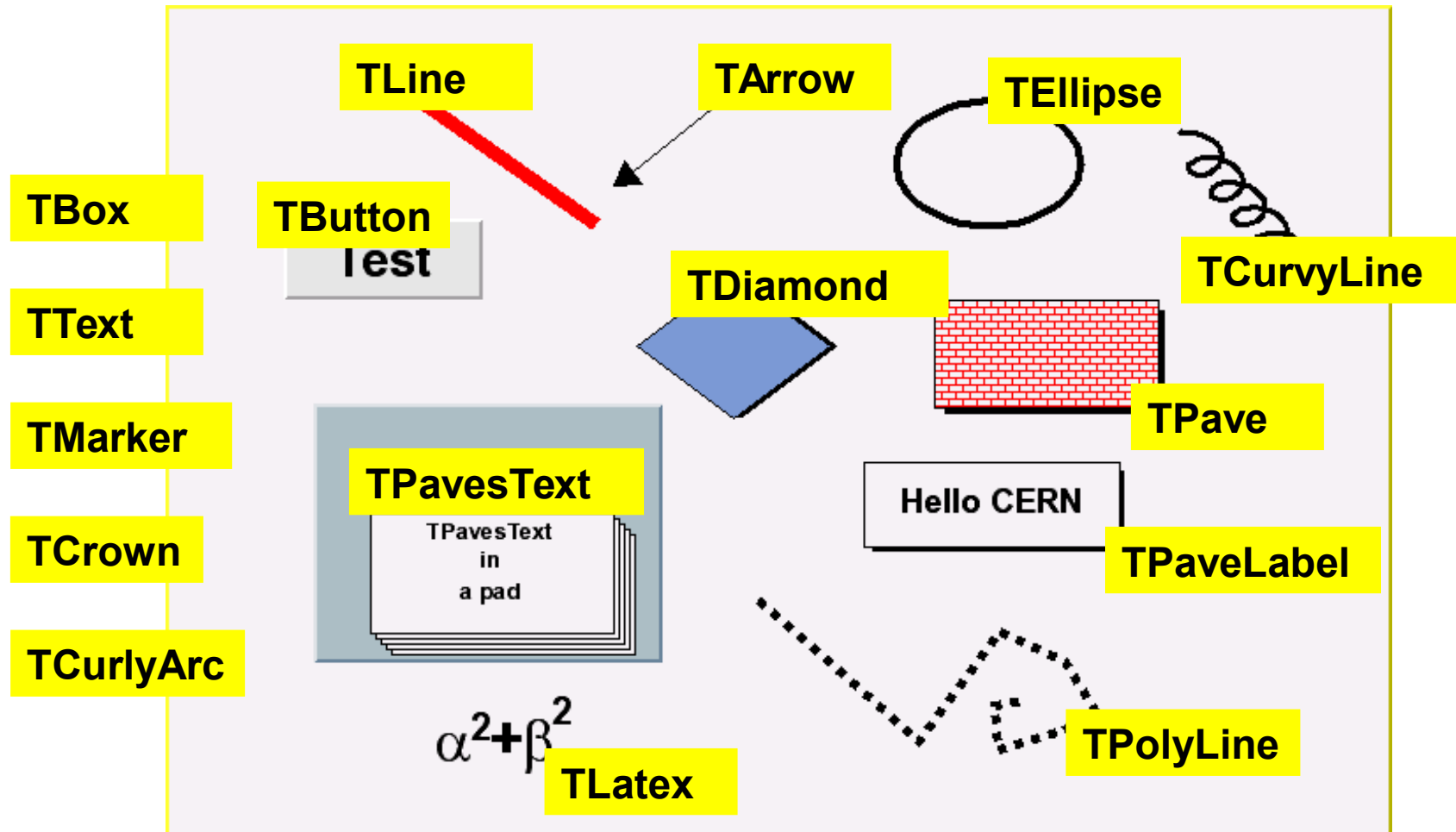
- You can draw with the command line
- The **Draw** function adds the object to the list of *primitives* of the current *pad*
- If no pad exists, a pad is automatically created
- A pad is embedded in a *canvas*
- You create one manually with **new TCanvas**
  - A canvas has one pad by default
  - You can add more

```
root [ ] TLine line(.1,.9,.6,.6)
root [ ] line.Draw()
root [ ] TText text(.5,.2,"Hello")
root [ ] text.Draw()
```

Canvas



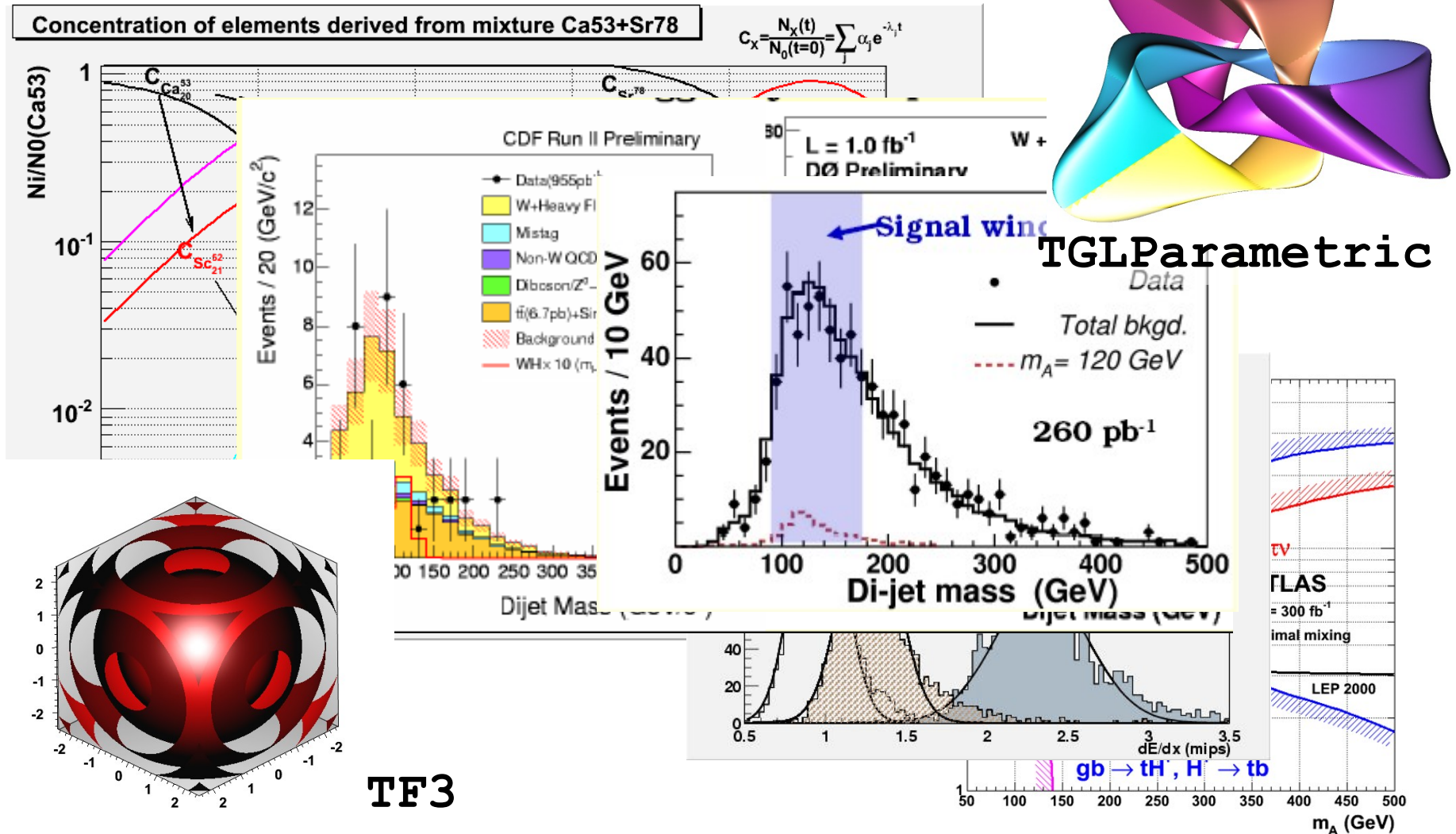
# More Graphics Objects



Can be accessed with the toolbar  
View → Toolbar (in any canvas)



# Graphics Examples



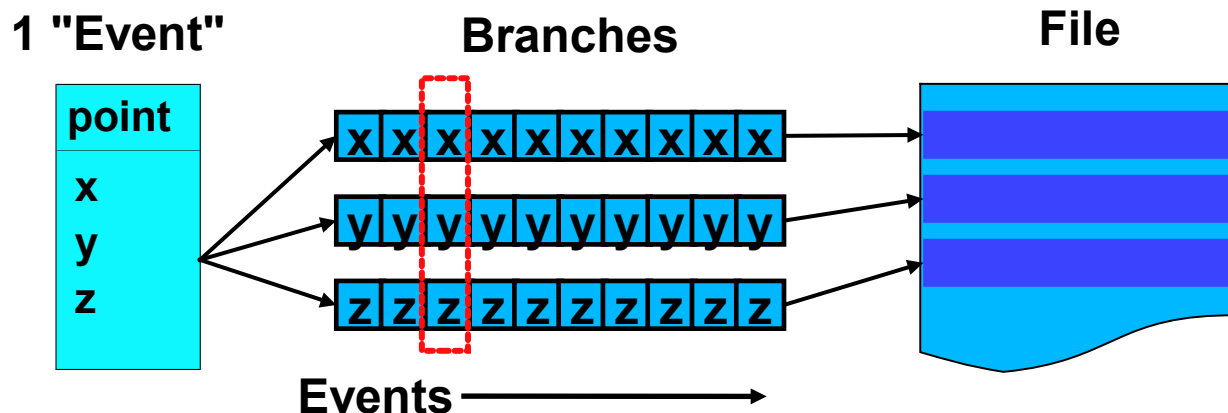
# What is a ROOT Tree?

- Trees have been designed to support very large collections of objects. The overhead in memory is in general less than 4 bytes per entry.
- Trees allow direct and random access to any entry (sequential access is the most efficient)

The class TTree is the main container for data storage

It can store any class and basic types (e.g. Float\_t)

When reading a tree, certain branches can be switched off  
→ speed up of analysis when not all data is needed



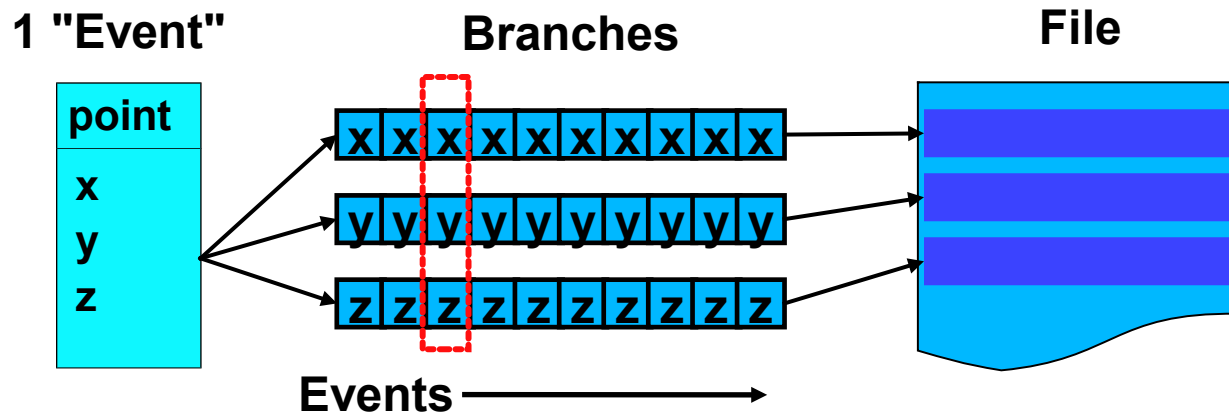
# Trees

Trees are structured into branches and leaves. One can read a subset of all branches

High level functions like **TTree::Draw** loop on all entries with selection expressions

Trees can be browsed via **TBrowser**

Trees can be analyzed via **TTreeViewer**



# TTree - Writing

- You want to store objects in a tree which is written into a file
- Initialization

```
root [ ] TFile* f = TFile::Open("events.root",  
"RECREATE");  
root [ ] TTree* t = new TTree("Events","Event Tree");  
root [ ] Int_t      var1;  
root [ ] Float_t    var2;  
root [ ] Float_t    var3;  
root [ ] t->Branch("var1", &var1, "var1/I");  
root [ ] t->Branch("var2", &var2, "var2/F");  
root [ ] t->Branch("var3", &var3, "var3/F");
```

# TTree - Writing

## Fill the TTree

TTree::Fill copies content of member as new entry into the tree

```
root [ ] var1=5; var2=3.1; var3=10.;  
root [ ] t->Fill();  
root [ ] var1=1; var2=7; var3=4.5;  
root [ ] t->Fill();
```

## Inspect the tree

Flush the tree to the file  
close the file

```
root [ ] t->Print();  
root [ ] t->Show(1);  
  
root [ ] t->Write();  
root [ ] f->Close();
```

Code is in:

**simpletree.C**

# TTree - Reading

- Open the file, retrieve the tree and connect the branch with a pointer to TMyEvent

```
TFile *f = TFile::Open("events.root");  
TTree *tree = (TTree*)f->Get("Events");  
Float_t var2;  
tree->SetBranchAddr("var2", &var2);
```

- Read entries from the tree and use the content of the class

```
Int_t nentries = tree->GetEntries();  
for (Int_t i=0;i<nentries;i++) {  
    tree->GetEntry(i);  
    cout << var2 << endl;  
}
```

Code is in: **readtree.C**

A quick way to browse through a tree is to use a **TBrowser** or **TTreeView**

# Trees (2)

- Accessing a more complex objects from non-standard classes
  - Members are accessible even without the proper class library
  - Might not work in all frameworks
- Example: eventdata.root (containing kinematics from ALICE)

```
$ root eventdata.root
```

```
root [ ] tree->Scan();
```

```
root [ ] tree->Scan("*");
```

```
root [ ] tree->Scan("fParticles.fPosX:fParticles.fPosY:fParticles.fPosZ");
```

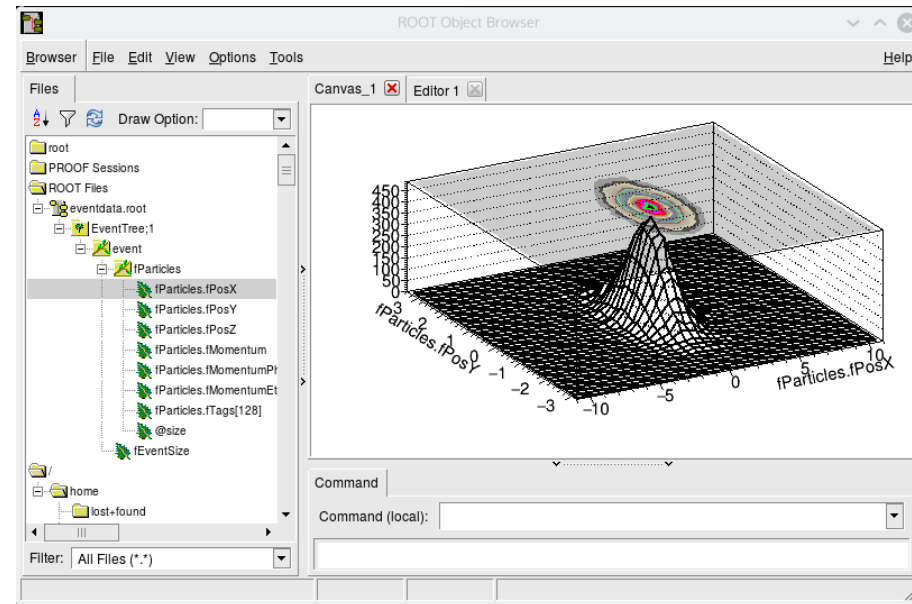
```
root [ ] tree->Scan("fParticles.fPosX:fParticles.fPosY:fParticles.fPosZ",  
"fParticles.fPosX<0")
```

# Trees (2)

- Accessing a more complex objects from non-standard classes
  - Members are accessible even without the proper class library
  - Might not work in all frameworks
- Example: eventdata.root (containing kinematics from ALICE)

```
$ root eventdata.root  
root [ ] tree->Draw("fParticles.fPosX")  
root [ ] tree->Draw("fParticles.fPosY:fParticles.fPosX")  
root [ ] tree->Draw("fParticles.fPoxY", "fParticles.fPoxX< 0")
```

- Perform more complex selections
- Plot 1D, 2D histograms with different styles
- Perform fits of some of these distributions
- distributions

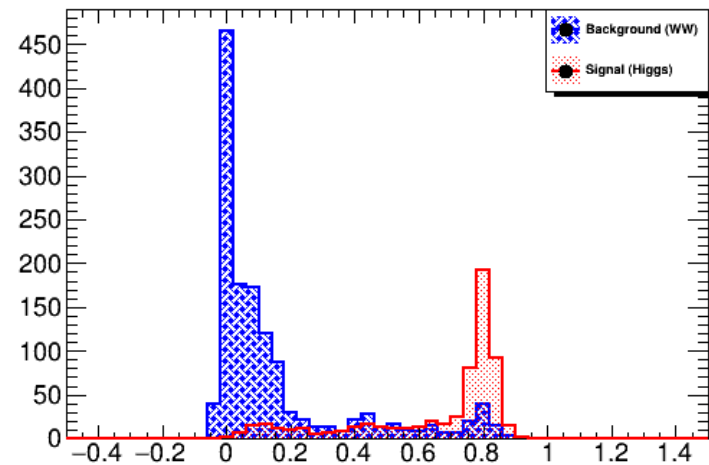
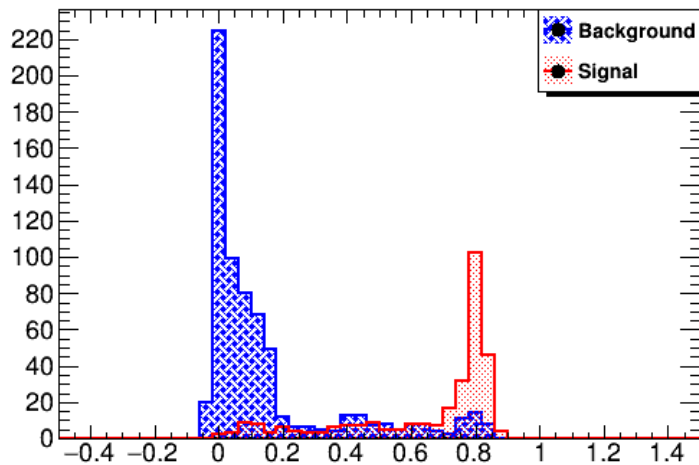
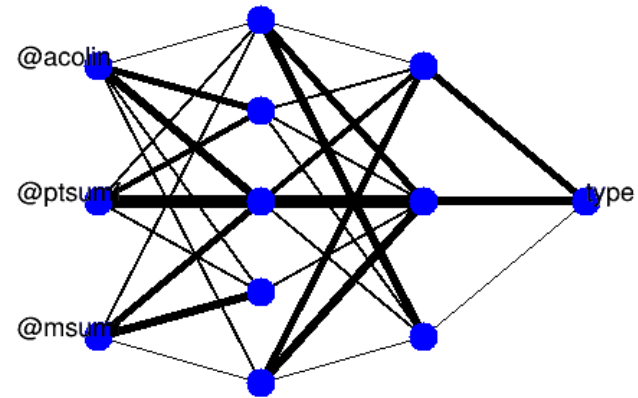
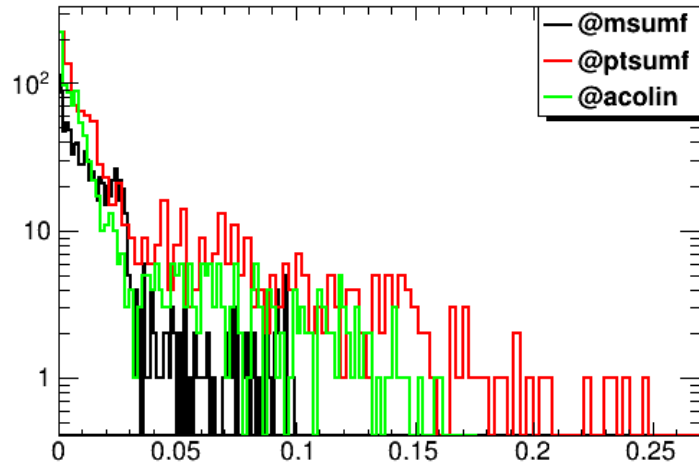


# Machine Learning

- **Example of advanced statistical analysis:**
  - **Read from a tree the event variables for:**
    - “signal” process, e.g. a simulation of a new phenomena you are looking for.
    - simulation of a “background process you want to separate the signal from.
  - **Build a Neural Network with these variables, whose separation of the signal to background is much better than the each of the input variables.**
  - **Launch the macro: mlpHiggs.C**
  - **Check the contents of the macro and of the mlpHiggs.root file:**

**[TFile::Open\("http://root.cern.ch/files/mlpHiggs.root"\)](http://root.cern.ch/files/mlpHiggs.root)**

# Machine Learning



# PyRoot

**ROOT is developed in C++ and has a native C++ interpreter, but it is interfaced also to other languages, such as python.**

**Open (i)python:**

**In [1]: import ROOT**

**In [2]: h = ROOT.TH1F("h", "h", 100, 0, 0)**

**In [3]: h.GetName()**

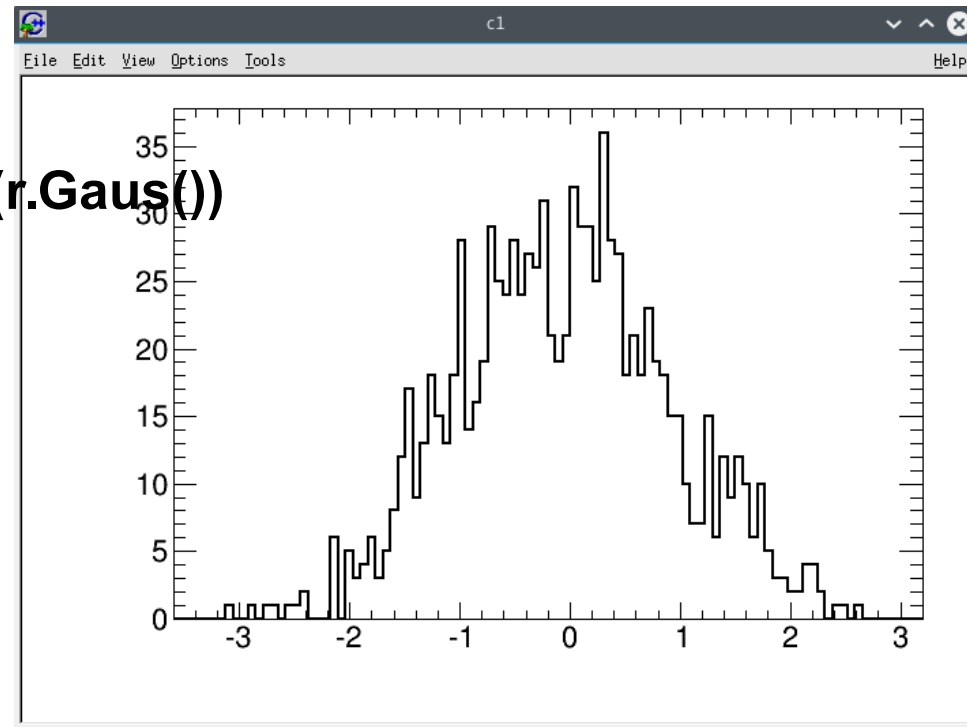
**Out[3]: 'h'**

**In [4]: r= ROOT.TRandom()**

**In [5]: for i in xrange(0,1000): h.Fill(r.Gaus())**

**In [6]: h.Draw()**

**Now you can redo all the tutorial in python if you wish!**



# RootBook

## Interactive ROOT in your Browser!

<https://app.mybinder.org:80/3000949792/notebooks/index.ipynb>

ROOT\_Example (unsaved changes) Terminal

File Edit View Insert Cell Kernel Help | ROOT C++

Code CellToolbar

### How to write a C++ ROOTBook

In order to use a C++ [ROOTBook](#), just start the [ROOT kernel](#). That's it. We will see also how the user can mark cells containing C++ statements with the `%%cpp` magic.

```
In [1]: cout << "From this point on...\n"
        << "... it's only C++ ..." << endl;
```

From this point on...  
... it's only C++ ...

We now create a [ROOT histogram](#) and fill it with [random numbers](#) distributed according to a Gaussian.

```
In [2]: TH1F h("gauss", "Example histogram", 64, -4, 4);
        h.FillRandom("gaus");
```

Now, we create a [canvas](#), the entity which holds graphics primitives.

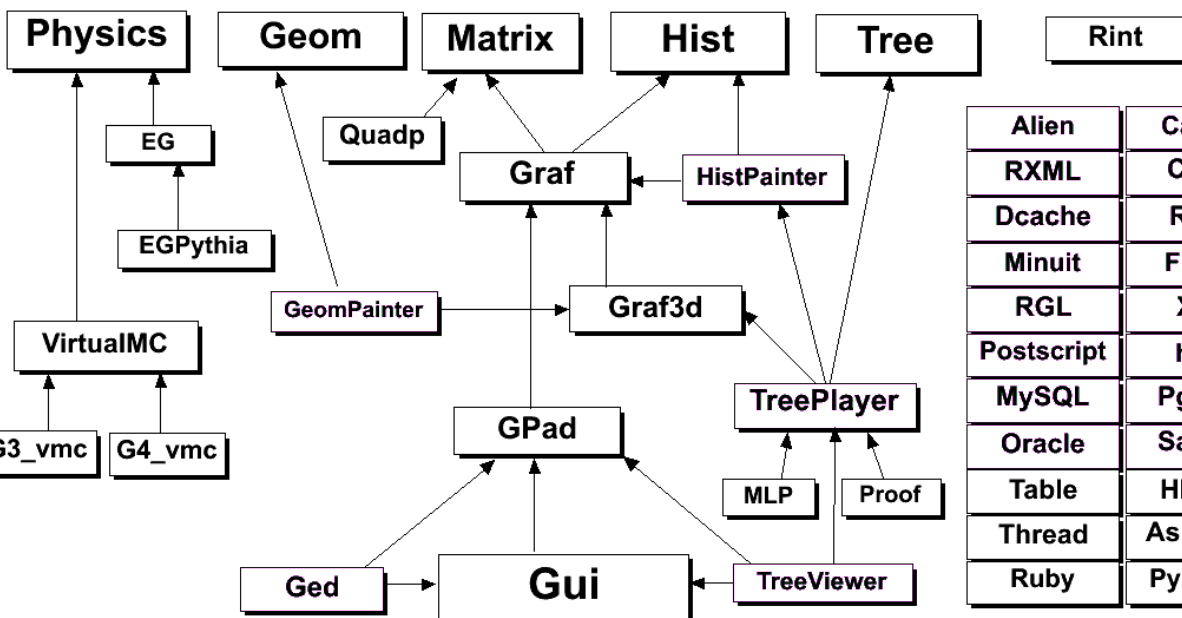
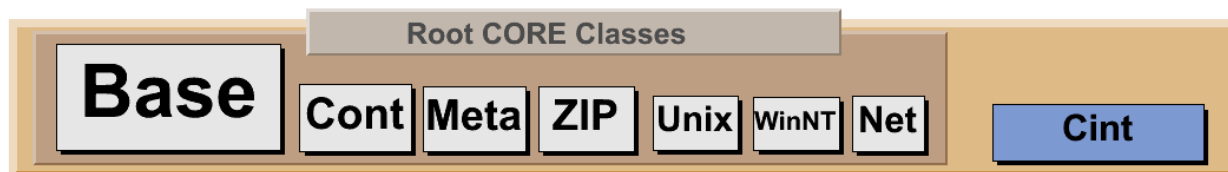
```
In [3]: TCanvas c("myCanvasName", "The Canvas Title", 1600, 1200);
        c.Draw();
```

For the histogram to be displayed in the notebook, we need to draw the canvas.

```
In [4]: c.Draw();
```

# ROOT is MUCH more

In this talk, I presented the most basic classes typically used during physics analyses



All libs need Core  
Arrows show lib dependencies  
CINT can be used independently  
Green libs loaded by PluginManager

ROOT Libraries Dependencies

ROOT contains many more libraries, and has several more applications

Alien	Castor
RXML	Chirp
Dcache	RFIO
Minuit	Fumili
RGL	X3D
Postscript	html
MySQL	PgSQL
Oracle	SapDB
Table	Hbook
Thread	AsImage
Ruby	PyROOT