



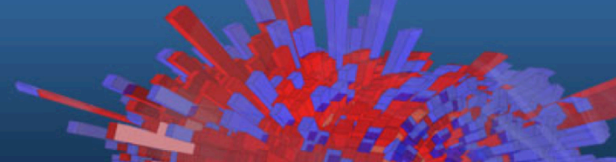
# Summer Students Course 2015

*Danilo Piparo, Olivier Couet*

*CERN PH-SFT*



# This Course



This is an introductory ROOT Workshop, not a lecture about ROOT

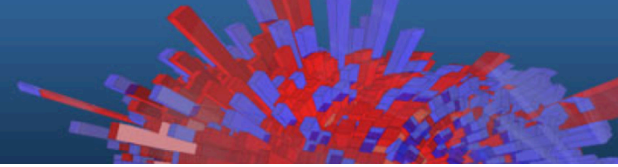
## **Objectives:**

- Become familiar with the ROOT toolkit
- Be able to use the C++ prompt
- Plot data
- Fit data
- Perform basic I/O operations

## **Format:**

- Slides treating the most important concepts
- Hands on exercises proposed during the exposition

# This Tutorial



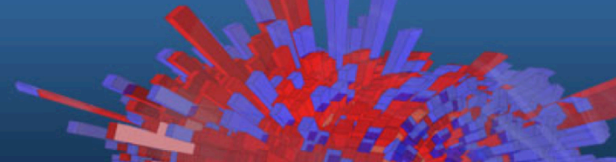
These slides are supported by the “**ROOT Primer**”

- Introductory booklet (~60 pages)
- Available on the ROOT website (html, epub, pdf): <http://cern.ch/go/SV87>
- Code examples will be visualised with the Jupiter Notebooks attached to the indico agenda
  - Signaled with name and the sign:



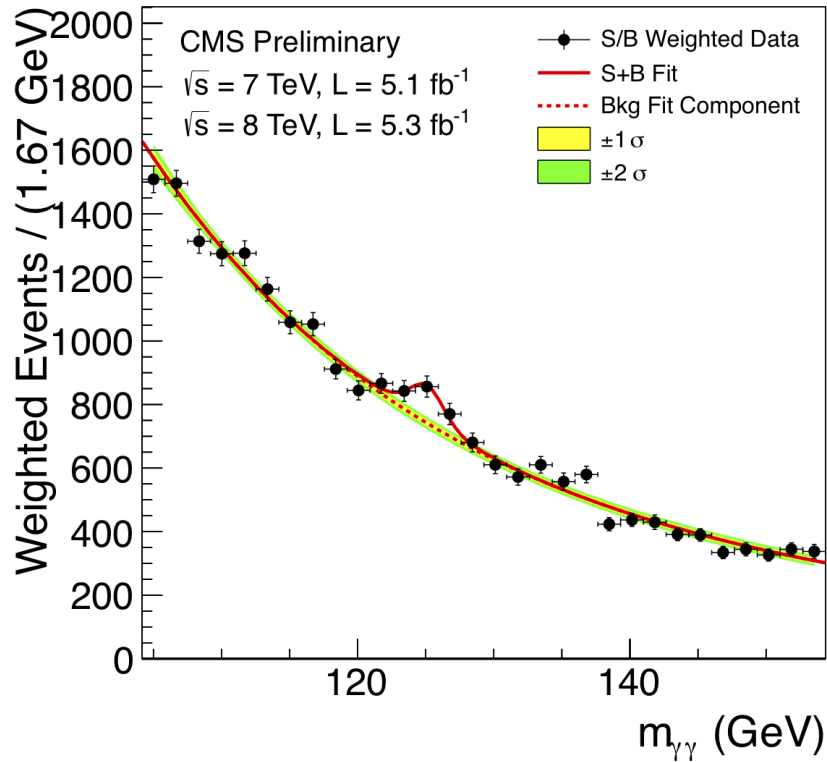
Two release series of ROOT are available: ROOT5 and ROOT6  
**This lecture refers to ROOT6, version 6.04**

# A “Quick Tour” Of ROOT

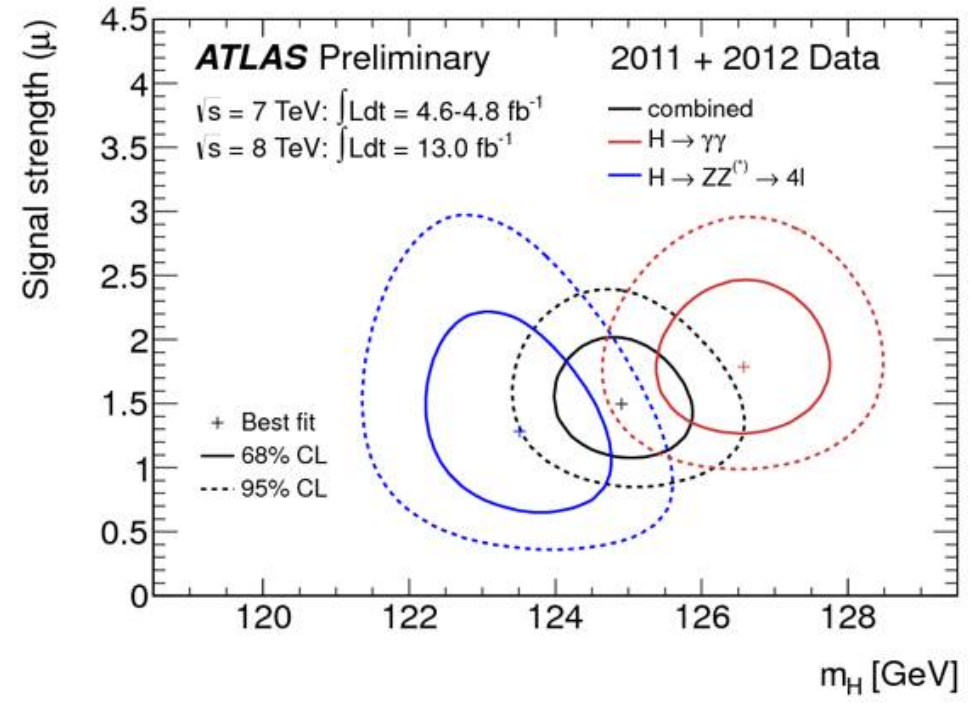




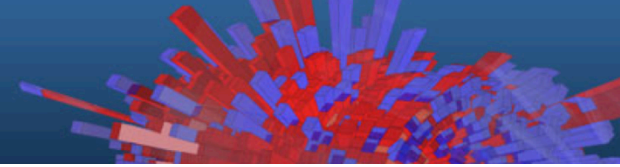
# What can you do with ROOT?



**LHC collision in CMS:  
event display, also done with ROOT!**



# ROOT in a Nutshell



ROOT is a software toolkit which provides building blocks for

- Data processing
- Data analysis
- Data visualisation
- Data storage

**An Open Source Project**

*All contributions are warmly welcome!*



ROOT is written mainly in C++ (C++11 standard)

- Bindings for Python and other languages\* provided

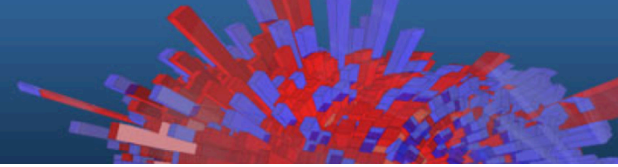


Adopted in High Energy Physics and other sciences (but also industry)

- ~250 PetaBytes of data in ROOT format on the LHC Computing Grid
- Fits and parameters' estimations for discoveries (e.g. the Higgs)
- Thousands of ROOT plots in scientific publications

\* Atm only Python for the 6 series

# ROOT in a Nutshell

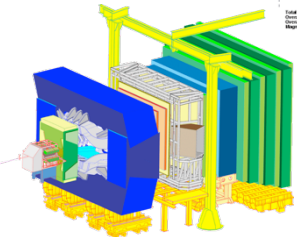
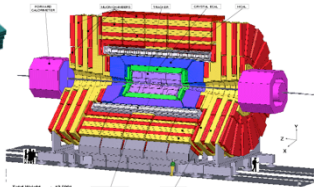
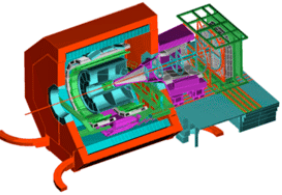
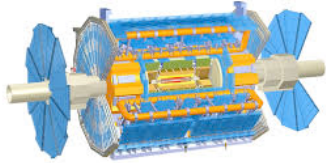


ROOT can be imagined as a family of building blocks for a variety of activities, for example:

- Data analysis: histograms, graphs, trees
- I/O: row-wise, column-wise storage of **any** C++ object
- Statistical tools (RooFit/RooStats): rich modeling and statistical inference
- Math: non trivial functions (e.g. Erf, Bessel), optimised math functions (VDT)
- C++ interpretation: fully C++ II compliant
- Multivariate Analysis (TMVA): e.g. Boosted decision trees, neural networks
- And more: HTTP servering, JavaScript visualisation, advanced graphics (2D, 3D, event display).
- PROOF: parallel analysis facility

# ROOT Application Domains

A selection of the experiments adopting ROOT



**Event Filtering**

Data

**Offline Processing**

**Analysis**

Reconstruction

Further processing, skimming

Event Selection, statistical treatment ...

Raw

Reco

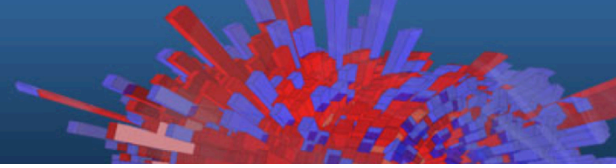
...

Analysis Formats

Images

**Data Storage: Local, Network**

# Interpreter



ROOT is shipped with an interpreter, CLING

- **C++ interpretation:** highly non trivial and not foreseen by the language!
- One of its kind: Just In Time (JIT) compilation
- A C++ interactive shell.

Can interpret “macros” (non compiled programs)

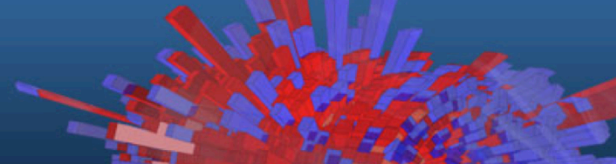
- Rapid prototyping possible

```
$ root -b
root [0] 3 * 3
(const int)9
```

ROOT provides also **Python bindings:**

- Can use Python interpreter directly after a simple *import ROOT*
- Possible to “mix” the two languages (see more in the following slides!)

# Persistency (I/O)



ROOT offers the possibility to write C++ objects into files

- Exceptional: impossible with C++ alone!
- Used for petabytes/year rates of LHC detectors.

Achieved with serialization of the objects using the reflection capabilities, ultimately provided by the interpreter

- Raw and column-wise streaming

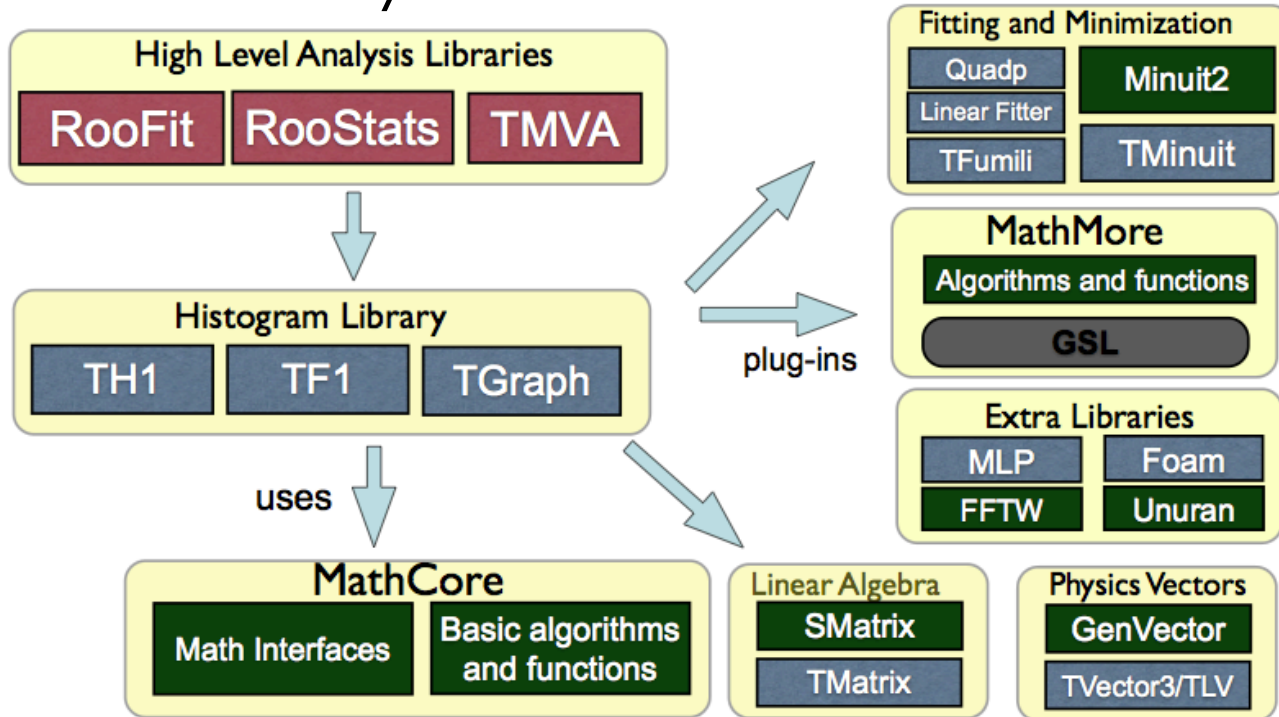
As simple as this for ROOT objects: one method - `TObject::Write`

Cornerstone for storage  
of experimental data

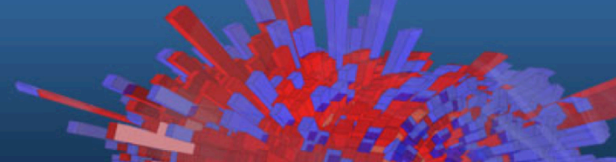


# ROOT Math/Stats Libraries

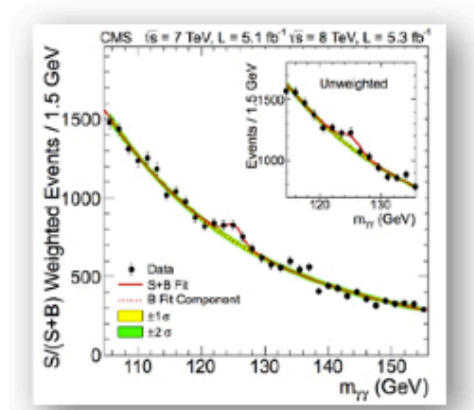
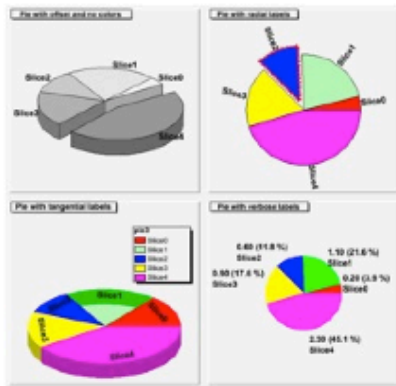
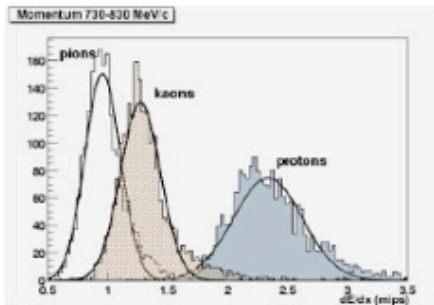
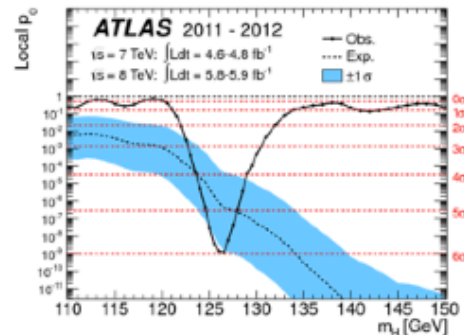
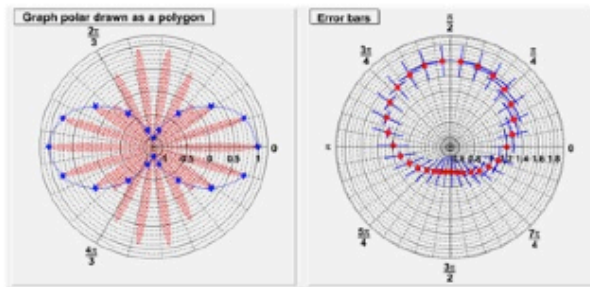
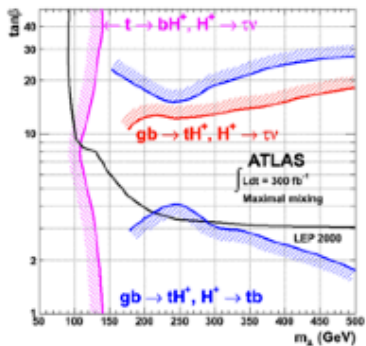
ROOT provides a reach set of mathematical libraries and tools needed for sophisticated statistical data analysis



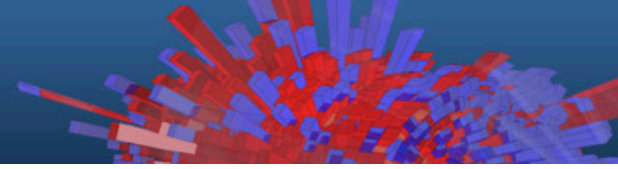
# Graphics In ROOT



Many formats for data analysis, and not only, plots



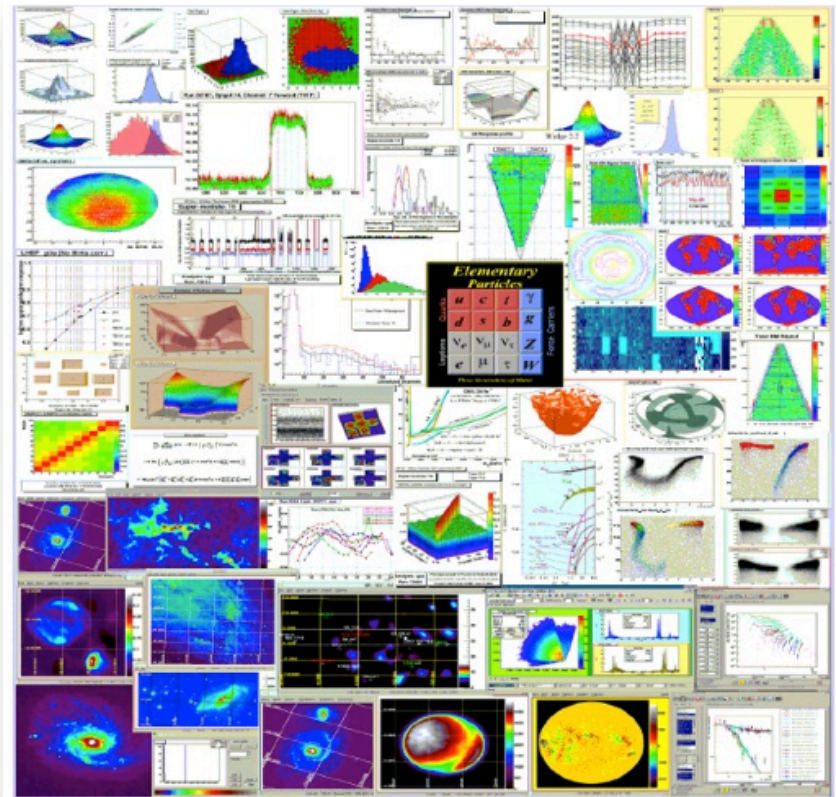
# 2D Graphics



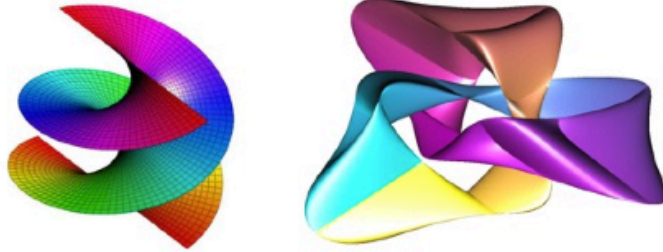
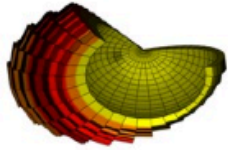
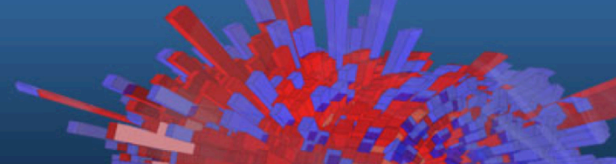
New functionalities added at every new release

Always requests for new style of plots

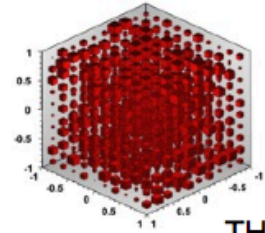
Can save graphics in many formats: *ps*, *pdf*, *svg*, *jpeg*, *LaTeX*, *png*, *c*, *root*



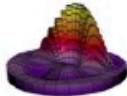
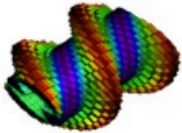
# 3D Graphics



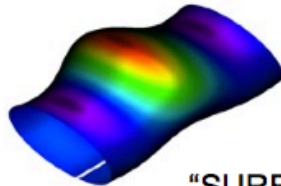
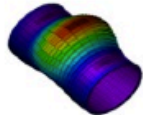
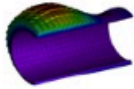
TGLParametric



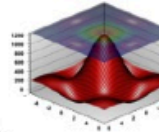
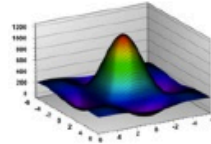
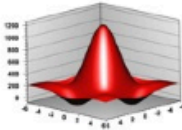
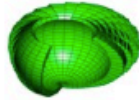
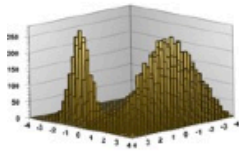
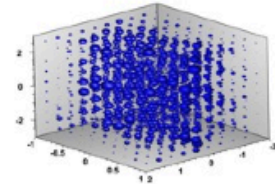
TH3



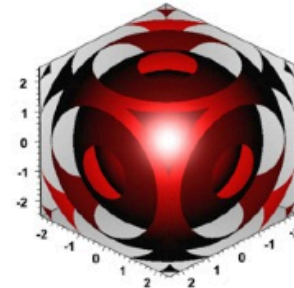
"LEGO"



"SURF"



TF3





# Other ROOT Features

## Geometry Toolkit

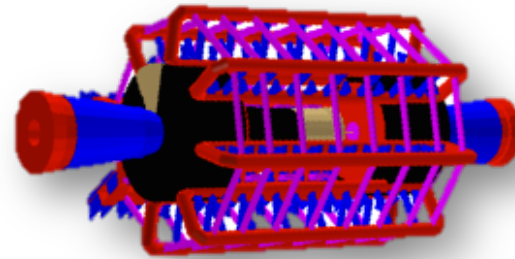
- Represent geometries as complex as LHC detectors

## Event Display (EVE)

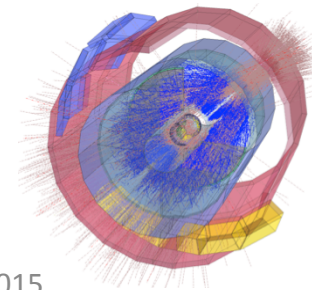
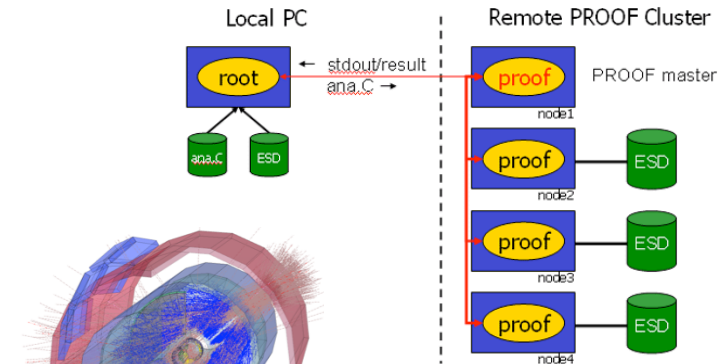
- Visualise particles collisions within detectors

## PROOF: Parallel ROOT Facility

- Multi-process approach to parallelism
- A system to run ROOT queries in parallel on a large number of distributed computers
- Proof-lite: does not need a farm, uses all the cores on a desktop machine



## PROOF Schema



ROOT web site: **the** source of information and help for ROOT users

- For beginners and experts
- Downloads, installation instructions
- Documentation of all ROOT classes
- Manuals, tutorials, presentations and more
- Forum
- ...

We propose to do a quick tour of the web site  
Don't hesitate to use it, even today!



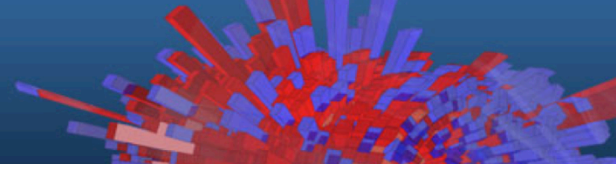
The screenshot shows the ROOT Data Analysis Framework website. The navigation bar includes links for Home, What's New, About, Screenshots, Download, Documentation, Support, Forum, and Developers. The main content area features three large icons: Screenshots, Download (circled in red), and Documentation. Below these are sections for 'What's New' (listing recent updates and patch releases), 'Recent Blog Posts' (listing various technical articles), and 'Follow Us On' (social media links). A 'ROOT Users' Workshop 2015' announcement is also visible.



# Preemptive Trouble Shooting

- ? *What could be the advantage of learning this software technology?*
- ! **1.** Batteries included: you have all the tools to process, store, analyse and visualise data in one single kit.
- ! **2.** You join a huge community,  $O(10^4)$  users + a very supportive team of core developers
  
- ? *Why C++ and not a scripting language?!*
- ! Performance. Support for languages like Python
  
- ? *Why prompt and libraries instead of a GUI?*
- ! ROOT is a programming framework, not an office suite.

# C++ From 10.000 Km



# C++ From 10.000 Km

Compiled, strongly typed language, allows to squeeze all the performance out of the hardware

- Veritable federation of languages, including C

Allows object orientation

Allows generic programming

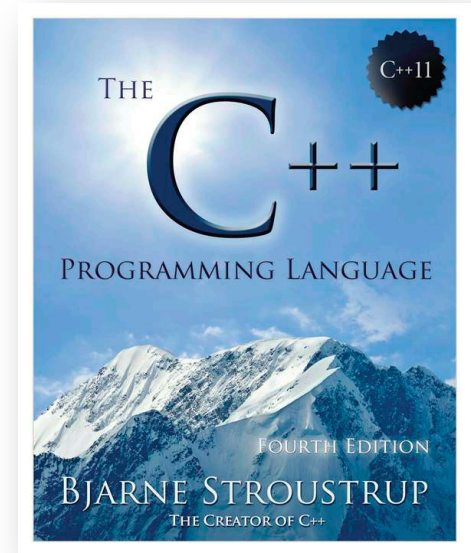
- Templates

Explicit memory management

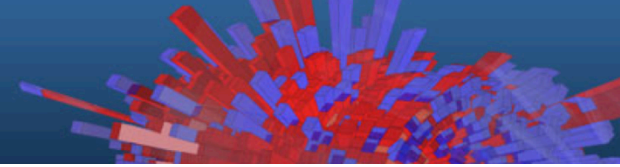
“Everything is a pointer”

Main language, together with Python, of HEP

- 90s: port ~all legacy FORTRAN HEP code to C++
- Reduce costs of management of large codebases (millions of lines of code)
- Allow groups of hundreds of active developers



# Some Useful Terms



- A class is an entity which encapsulate “data” and “actions” on it
- The “data” is represented by the *data members* (“variables of the class”)
- The actions are expressed by the *class methods* (“functions of the class”)
- One *calls, invokes* a method which can have zero or more arguments
- An *object* is an instance of a *class*
- An object is created by a special method, the *constructor*. There can be more than one constructor, e.g.:
  - `TH1F histo = TH1F(); // default constructor`
  - `TH1F histo = TH1F(“histName”, “HistTitle”, 64, 0, 64); // with params`

Note: the language is somehow approximate but certainly ok for this lecture

# -> and .

The *dot* and *arrow operators* are used to access methods and members of objects and pointers to objects

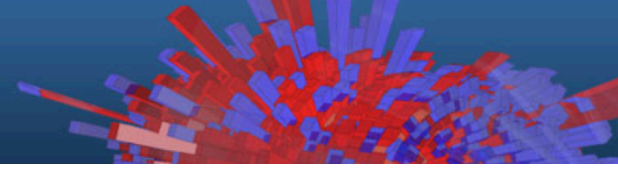
- *Dot*: to access methods and members of objects
- *Arrow*: to access methods and members of pointers to objects

Example:

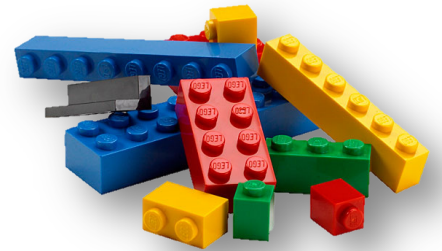
```
MyClass myClassInstance("myName");  
myClassInstance.GetName();  
auto myClassInstancePtr = new MyClass ("myName");  
myClassInstancePtr->GetName();
```

Note: the language is somehow approximate but certainly ok for this lecture

# ROOT Basics



ROOT as a Calculator  
ROOT as Function Plotter  
Plotting Measurements  
Histograms  
Interactive ROOT Section

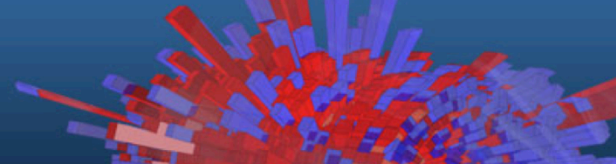




# Let's Fire Up ROOT



# The ROOT Prompt



C++ is a compiled language

- A compiler is used to translate source code into machine instructions

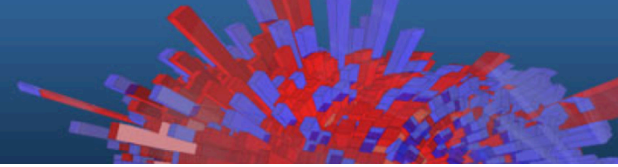
ROOT provides a C++ **interpreter**

- Interactive C++, w/o the need of a compiler, like Python, Ruby, Haskell ...
- Allows reflection (inspect at runtime layout of classes)
- Can be booted with the command:

`root`

- The interactive shell is also called “ROOT prompt” or “ROOT interactive prompt”

# ROOT As a Calculator



ROOT interactive prompt can be used as an advanced calculator!

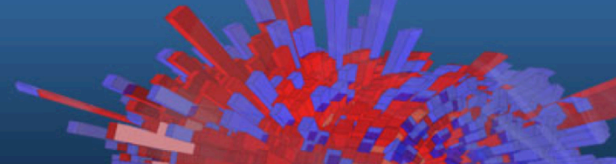
**Try it!**

```
root [0] 1+1
(int)2
root [1] 2*(4+2)/12.
(double) 1.000000e+00
root [2] sqrt(3.)
(double) 1.732051e+00
root [3] 1 > 2
(bool) false
```

ROOT allows not only to type in **C++ statements**, but also advanced **mathematical functions**, which live in the TMath namespace.

```
root [4] TMath::Pi()
(Double_t) 3.141593e+00
root [5] TMath::Erf(.2)
(Double_t) 2.227026e-01
```

# ROOT As a Calculator++



Here we make a step forward.  
We Declare **variables** and used  
a **for** control structure.  
Tab-completion available!

```
root [6] double x=.5  
(double) 5.000000e-01  
root [7] int N=30  
(int) 30  
root [8] double gs=0  
(double) 0.000000e+00
```

```
root [9] for (int i=0;i<N;++i) gs += TMath::Power(x,i)  
root [10] TMath::Abs(gs - (1-TMath::Power(x,N-1))/(1-x))  
(Double_t) 1.862645e-09
```

# Interlude: Controlling ROOT

Special commands which are not C++ can be typed at the prompt, they start with a “.”

```
root [1] .<command>
```

For example:

- Quit root: `.q`
- Issue a shell command: `!.<OS_command>`
- Load a macro: `.L <file_name>` (see following slides about macros)
- `.help` or `.?` gives the full list



# Exercise

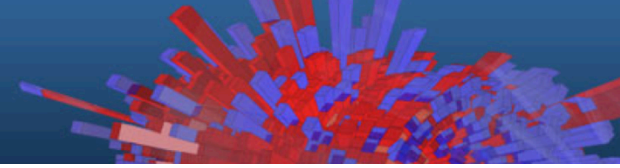
For x values of 0,1,10 and 20 check the difference of the value of a hand-made non-normalised Gaussian and the TMath::Gaus routine.

For one number

```
root [0] double x=0
root [2] exp(-x*x*.5) - TMath::Gaus(x)
[...]
```



# Exercise Solution



For x values of 0,1,10 and 20 check the difference of the value of a hand-made non-normalised Gaussian and the TMath::Gaus routine.

```
root [0] double x=0
root [2] exp(-x*x*.5) - TMath::Gaus(x)
[...]
```

Many possible ways of solving this! E.g:

```
root [0] for (auto v : {0.,1.,10.,20.}) cout << v << " " << exp(-
v*v*.5) - TMath::Gaus(v) << endl
```

# ROOT As a Function Plotter

The class TF1 represents one dimensional functions (e.g.  $f(x)$ ):

```
root [0] TF1 f1("f1","sin(x)/x",0.,10.); //name, formula, min, max
root [1] f1.Draw();
```

An extended version of this example is the definition of a function with parameters:

```
root [2] TF1 f2("f2","[0]*sin([1]*x)/x",0.,10.);
root [3] f2.SetParameters(1,1);
root [4] f2.Draw();
```

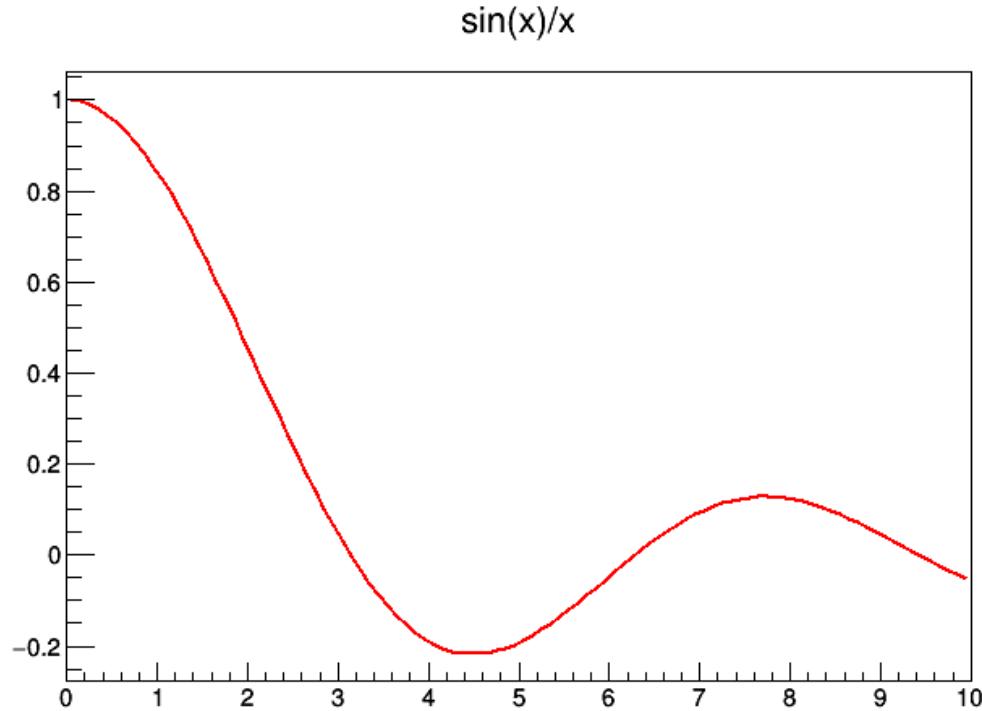
# ROOT As a Function Plotter

The class TFI re

```
root [0] TF1  
root [1] f1.D
```

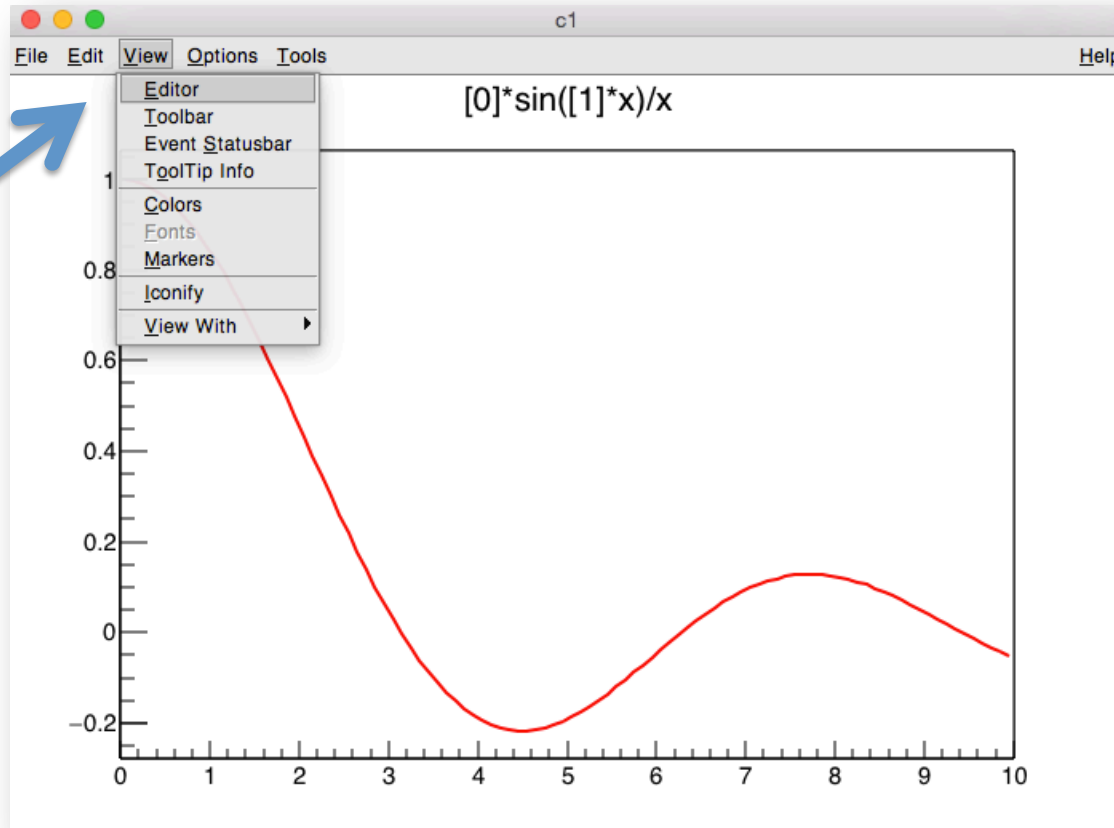
An extended ve  
with parameters:

```
root [2] TF1  
root [3] f2.  
root [4] f2.
```

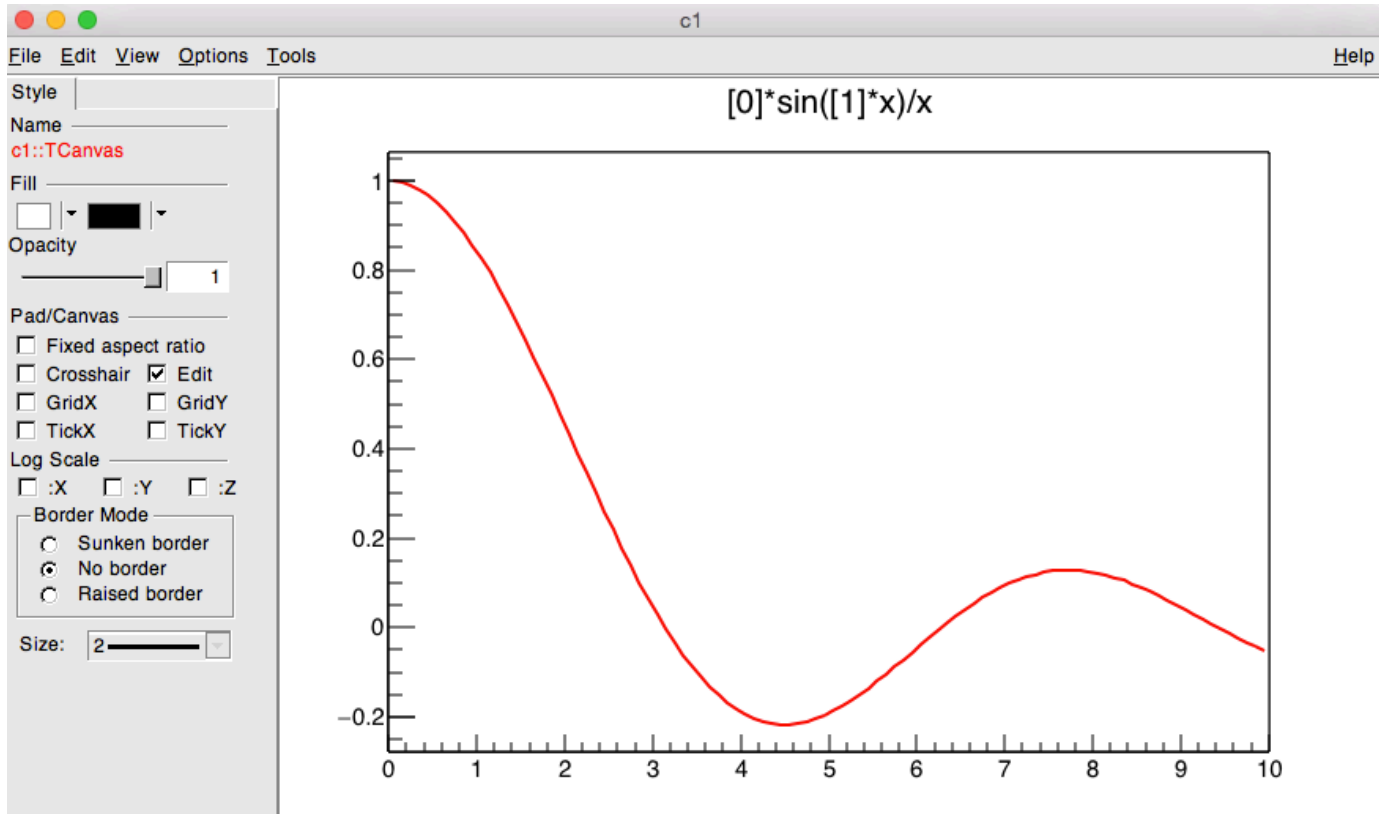


min, max

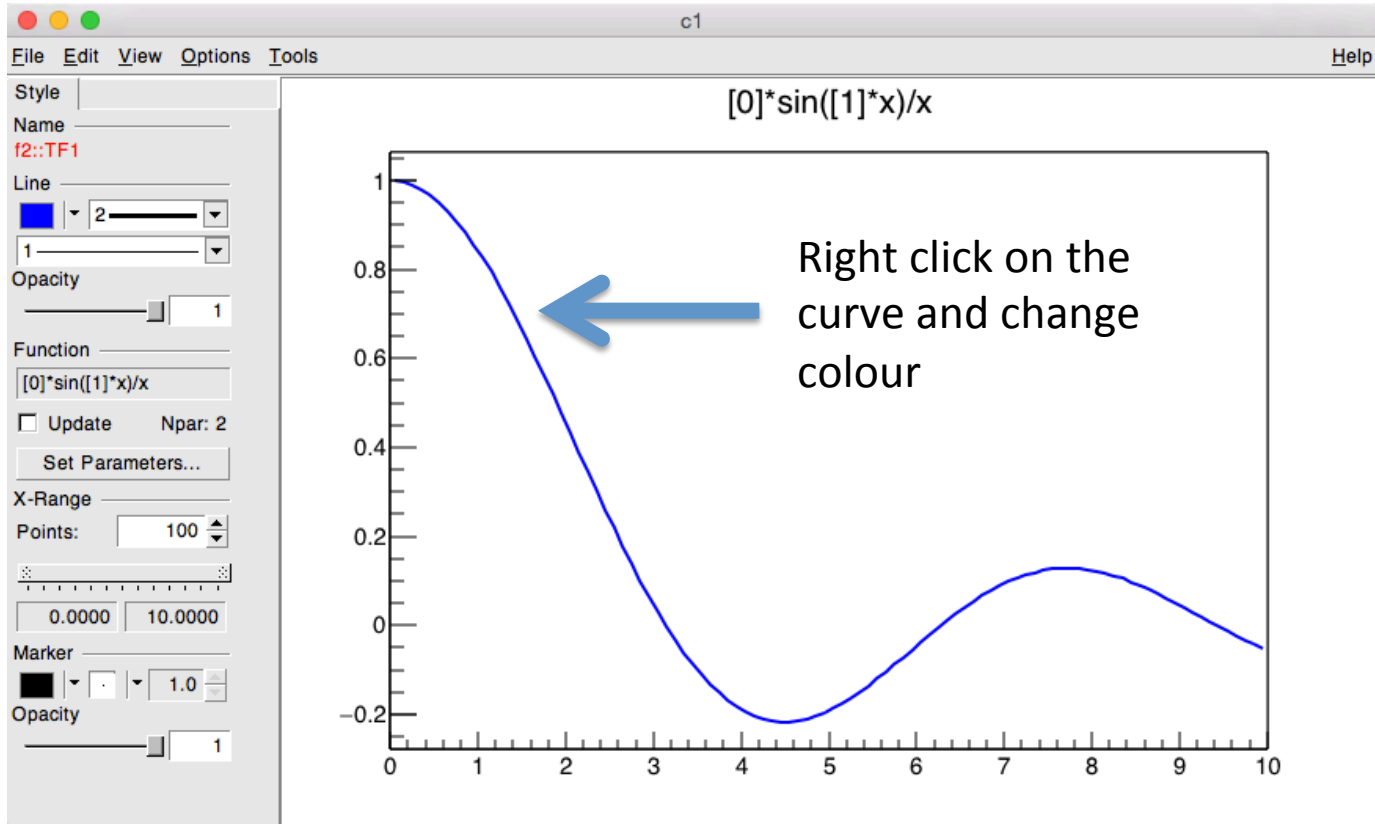
# Exercise: Interaction With The Plot



# Exercise: Interaction With The Plot



# Exercise: Interaction With The Plot



# Exercise: Interaction With The Plot

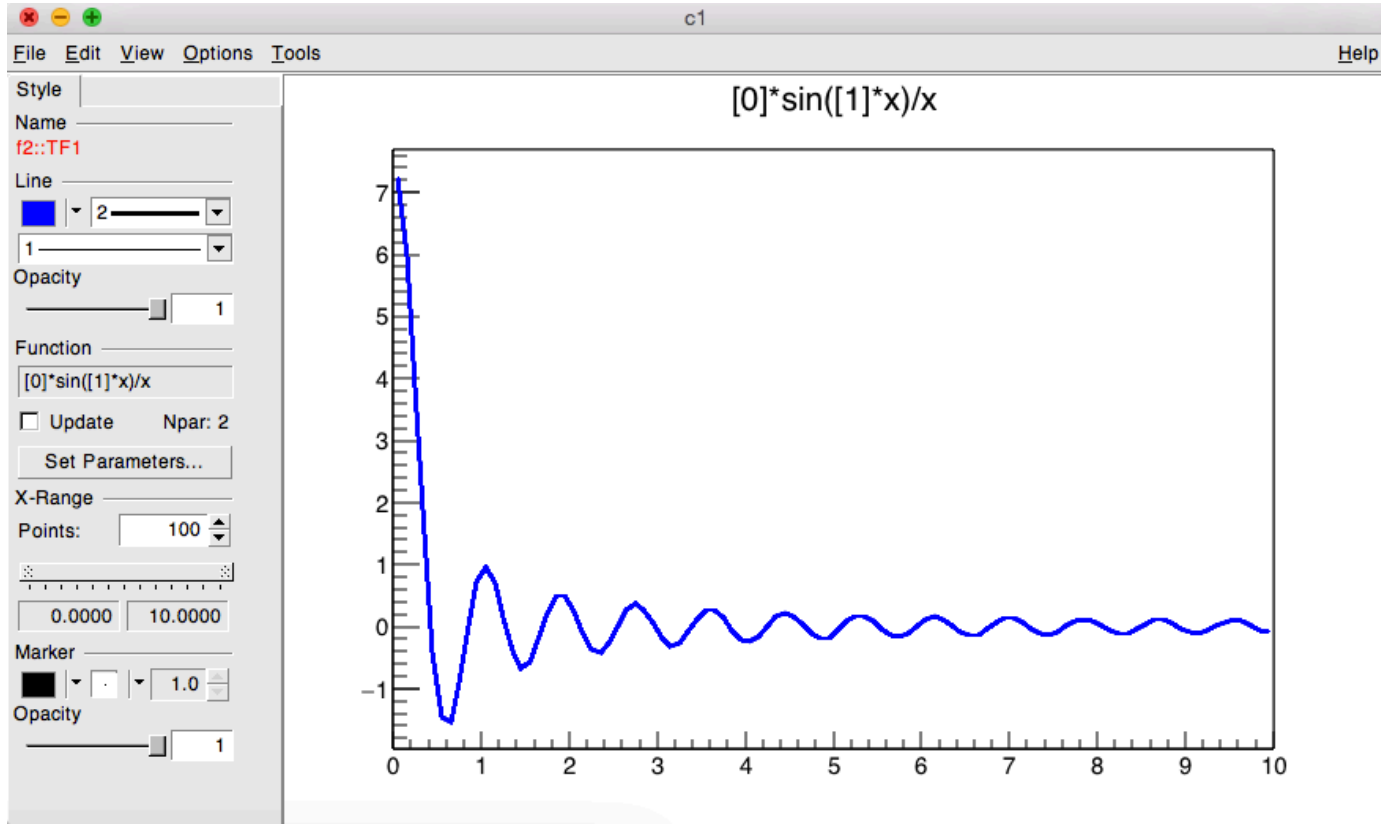
The image shows a ROOT window titled 'c1' with a plot of the function  $[0]*\sin([1]*x)/x$ . The plot shows a red curve starting at (0, 1) and decaying towards zero. A 'Set Parameters of [0]\*sin([1]\*x)/x' dialog box is open, showing the following table:

Name	Fix	Value	Min	Set Range	Max
p0	<input type="checkbox"/>	1	0.9	<input type="text"/>	1.1
p1	<input type="checkbox"/>	1	0.9	<input type="text"/>	1.1

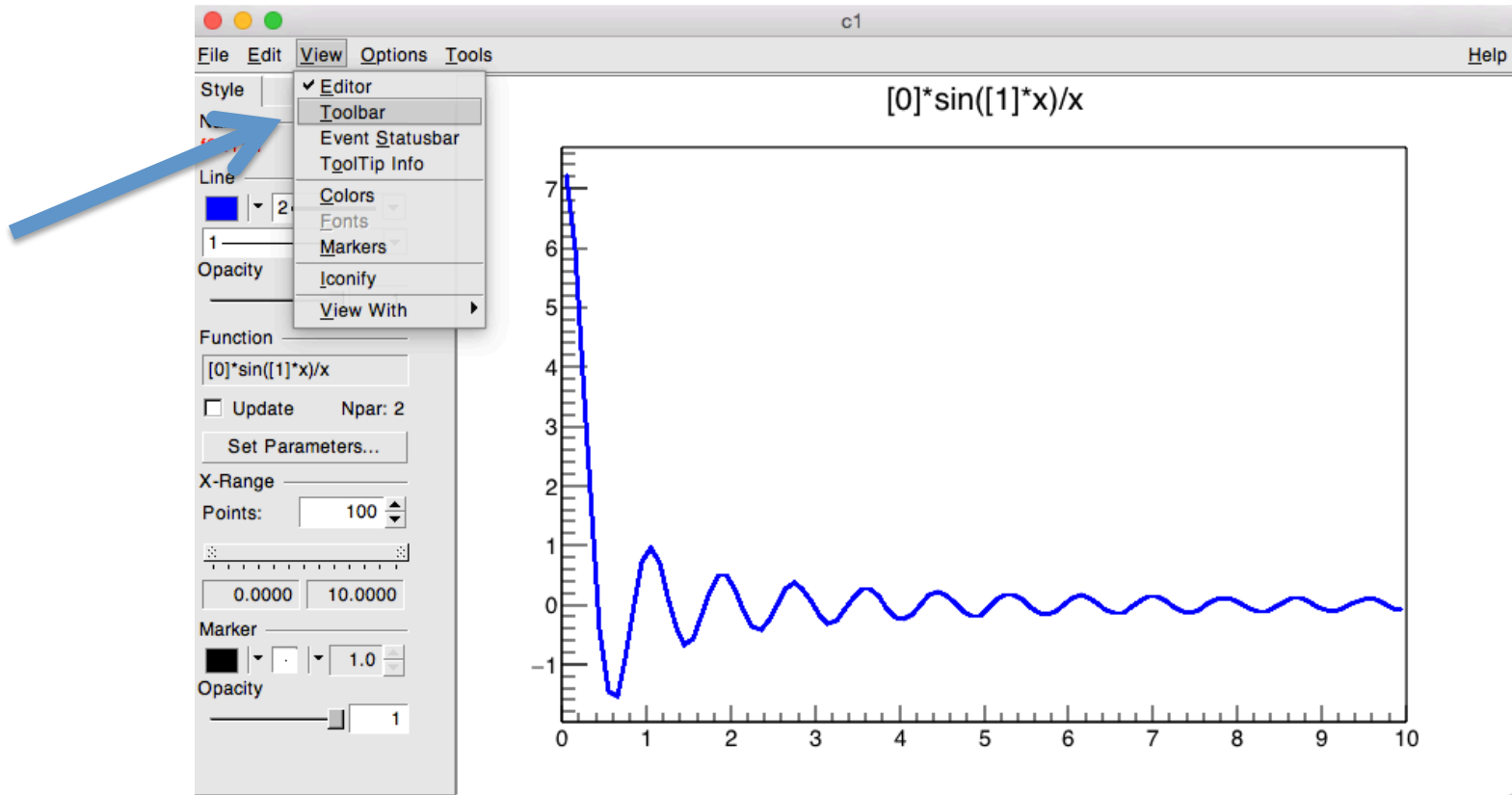
Below the table is the checkbox  Immediate preview. The dialog also has 'Reset', 'Apply', 'OK', and 'Cancel' buttons. Blue arrows point to the 'Update' checkbox in the left sidebar, the 'Value' field for p0 in the dialog, and the 'Immediate preview' checkbox. Text annotations include 'Increment in steps' pointing to the p0 value field and 'Set "Immediate preview"' pointing to the checkbox.



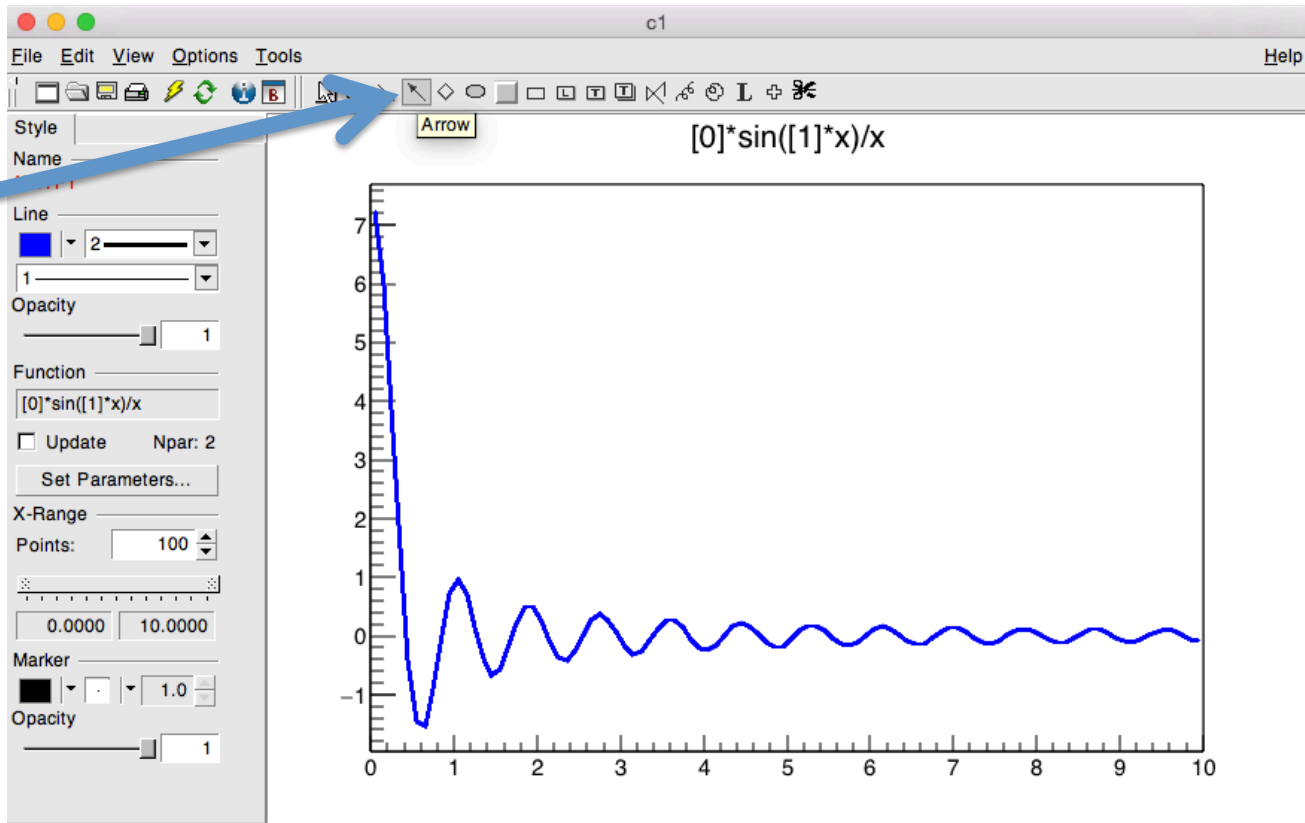
# Exercise: Interaction With The Plot



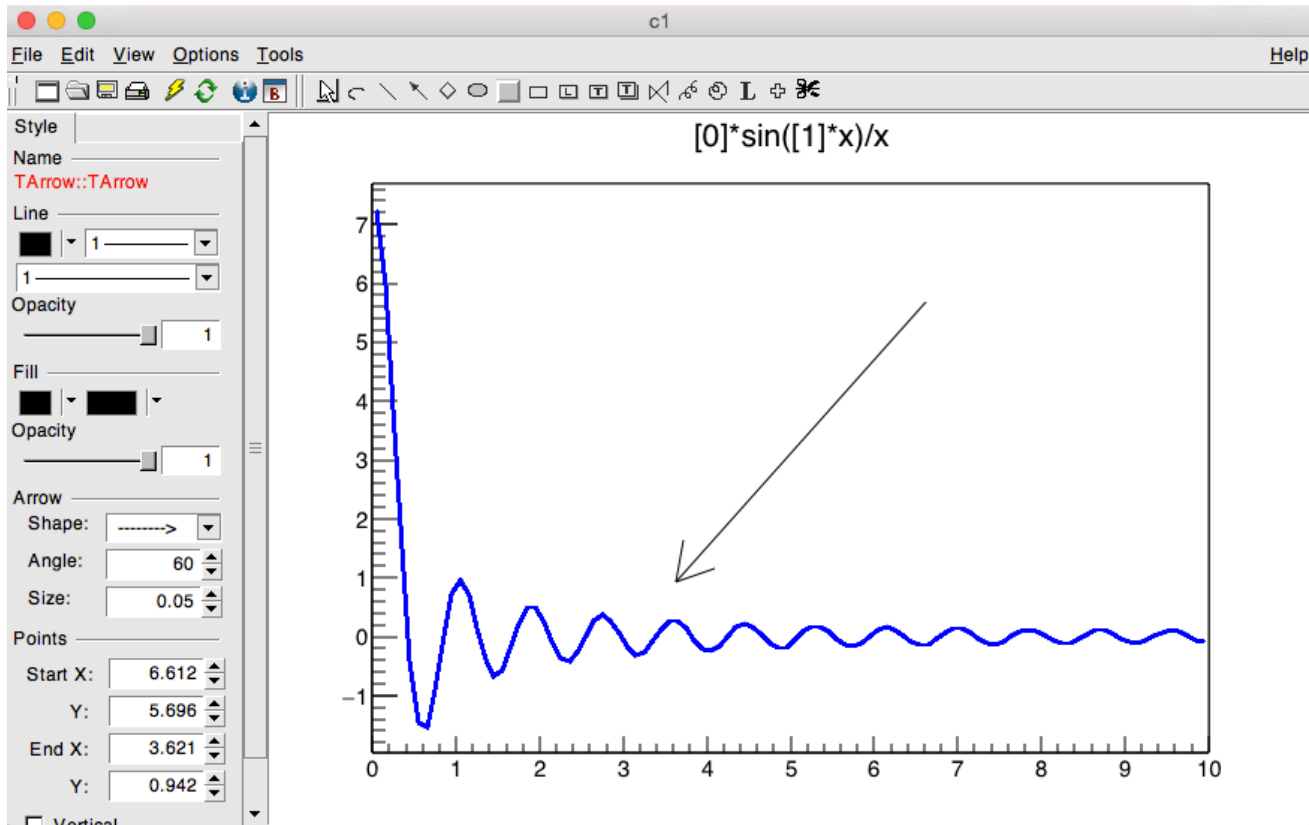
# Exercise: Interaction With The Plot



# Exercise: Interaction With The Plot

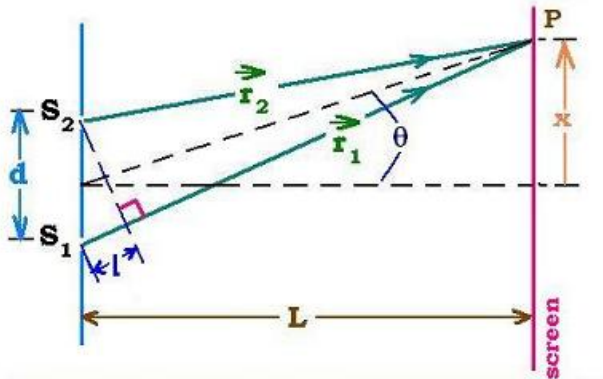


# Exercise: Interaction With The Plot



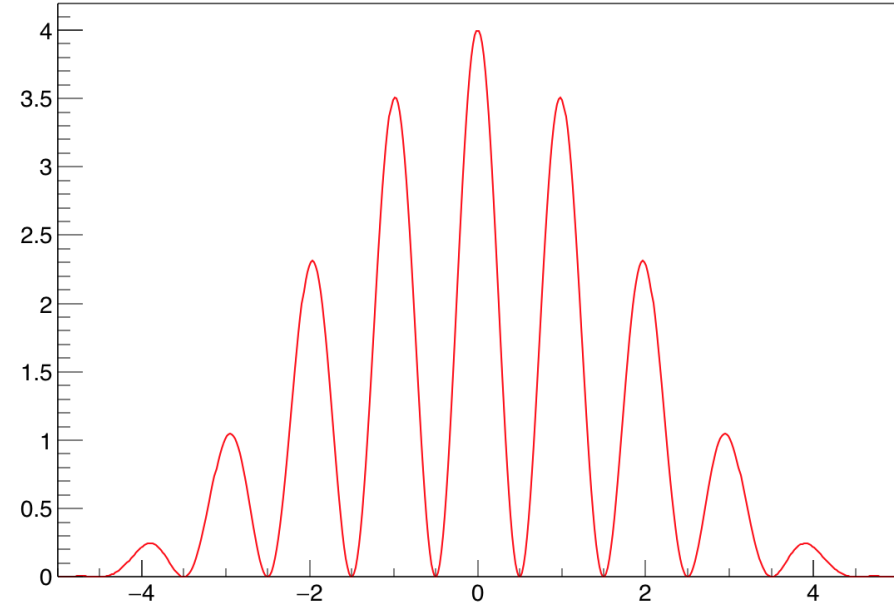
# ROOT As a Function Plotter

The example **slits.C** characterised in the Primer, is a more complex C++ program calculating and displaying the interference pattern produced by light falling on a multiple slit.



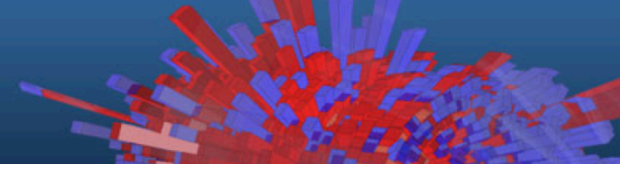
$L \gg d \Rightarrow$  Lines from each slit to P are parallel  
 $\Rightarrow \sin \theta = \frac{x}{L} = \frac{l}{d}$

Fnslit



Slits\_cpp

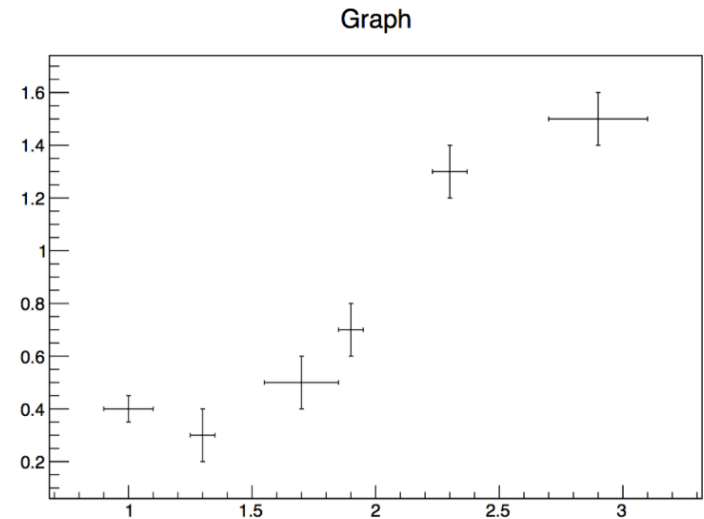
# Plotting Measurements



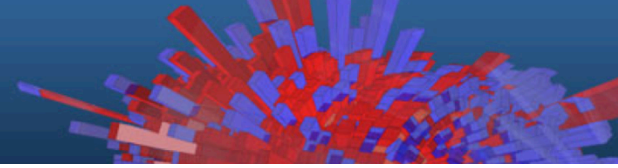
The class `TGraphErrors` allows to display measurements in ROOT, including errors, with different types of constructors. In the following example, data are taken from the file `ExampleData.txt`:

```
root [0] TGraphErrors gr("ExampleData.txt");  
root [1] gr.Draw("AP");
```

Tells ROOT to draw the **A**xis and the **P**oints



# Extempore Exercise

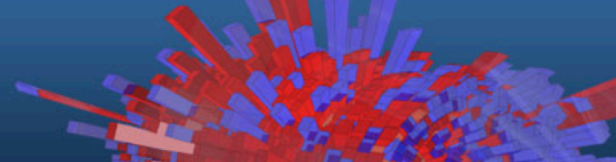


- Create a graph (TGraph)
- Set its title to “My graph”, its X axis title to “myX” and Y axis title to “myY”
- Fill it with three points: (1,0), (2,3), (3,4)
- Set a red full square marker
- Draw a orange line between points

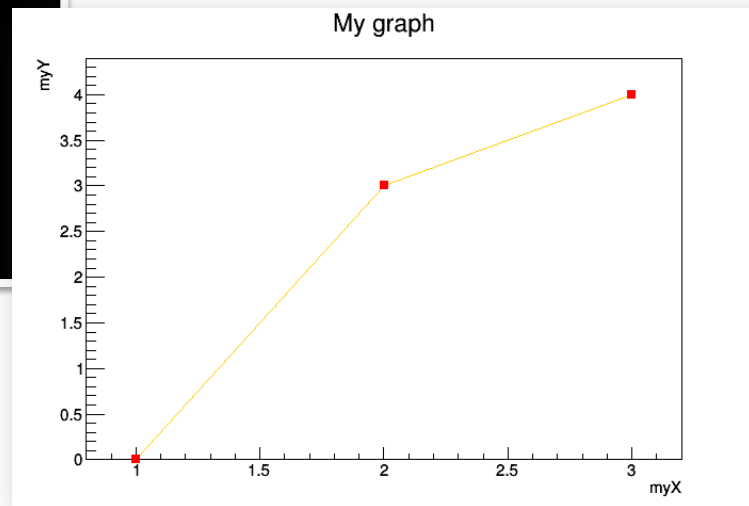
Let's solve this together at the whiteboard!



# Exercise Solution



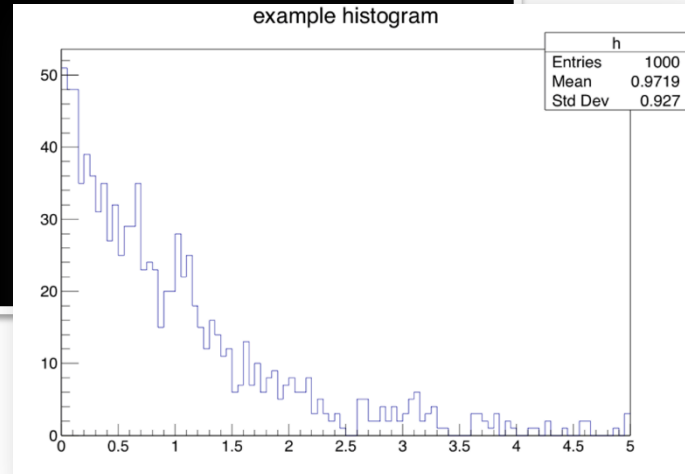
```
root [0] TGraph g
root [1] g.SetTitle("My graph;myX;myY")
root [2] g.SetPoint(0,1,0)
root [3] g.SetPoint(1,2,3)
root [4] g.SetPoint(2,3,4)
root [5] g.SetMarkerStyle(kFullSquare)
root [6] g.SetMarkerColor(kRed)
root [7] g.SetLineColor(kOrange)
root [8] g.Draw("APL")
```



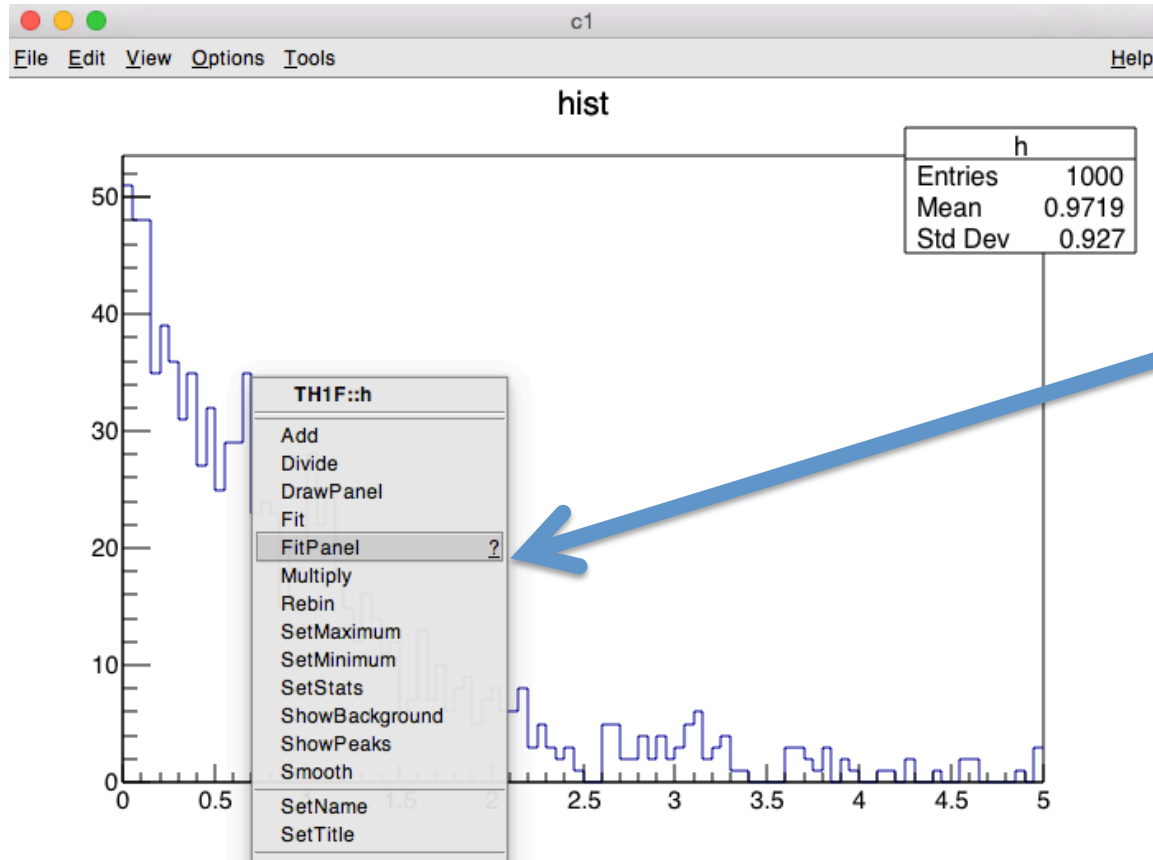
# Histograms

- In ROOT, the TH\* classes represent histograms
- TH1\* are monodimensional, TH2\* are bidimensional ...
- The final letter describes the type stored in each bin:  
A double in TH1D, a float in TH1F ...

```
root [0] TF1 efunc("efunc","exp([0]+[1]*x)",0.,5.)
root [1] efunc.SetParameters(1,-1)
root [2] TH1F h("h","hist",100,0.,5.)
root [3] for (int i=0;i<1000;i++)
h.Fill(efunc.GetRandom())
root [4] h.Draw()
```



# All together: Fitpanel



Click on the histogram "line"

# Exercise: Fit

The screenshot shows a software interface for fitting a Gaussian function to a histogram. The main window is titled "Fit Panel" and contains the following elements:

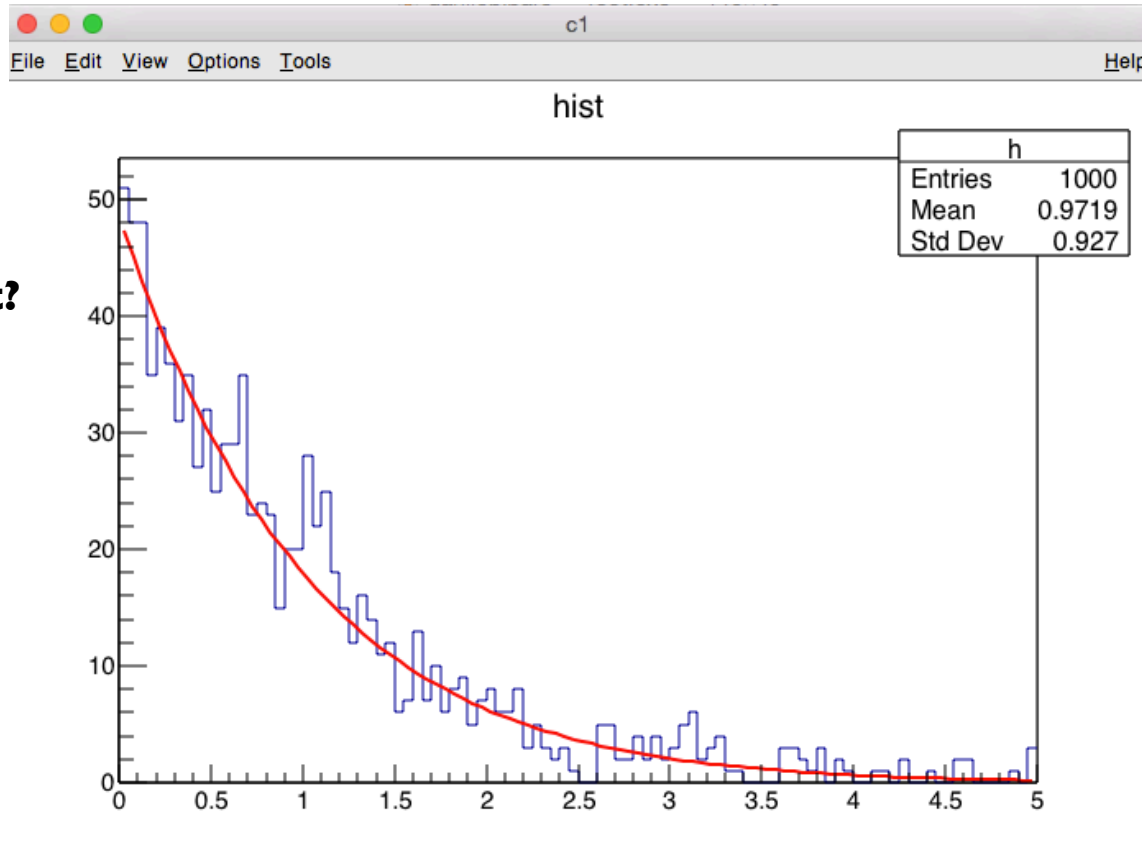
- Data Set:** TH1F::h
- Fit Function:** Type: Predef-1D, Function: gaus
- Operation:** Radio buttons for Nop (selected), Add, NormAdd, and Conv.
- Selected:** gaus
- Buttons:** Set Parameters...
- General / Minimization:** Method: Chi-square, Robust: 0.95
- Fit Options:** Integral, Best errors, All weights = 1, Empty bins, weights=1, Use range, Improve fit results, Add to list, Use Gradient.
- Draw Options:** SAME, No drawing, Do not store/draw.
- X-axis:** Range from 0.00 to 5.00.
- Buttons:** Update, Fit (highlighted with a blue arrow), Reset, Close.

On the right side, a zoomed-in view of the histogram and fit is shown. The histogram is labeled "h" and has the following statistics:

h	
Entries	1000
Mean	0.9719
Std Dev	0.927

The word "Fit!" is written next to the zoomed-in view, and a blue arrow points from the "Fit" button in the main panel to the zoomed-in view.

# Exercise: Fitpanel



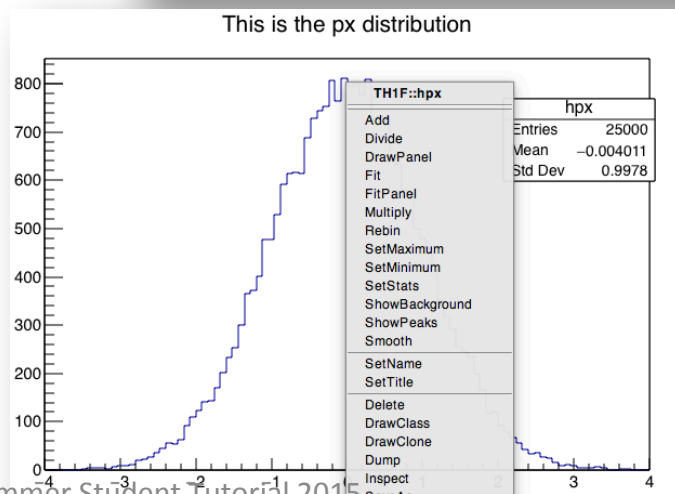
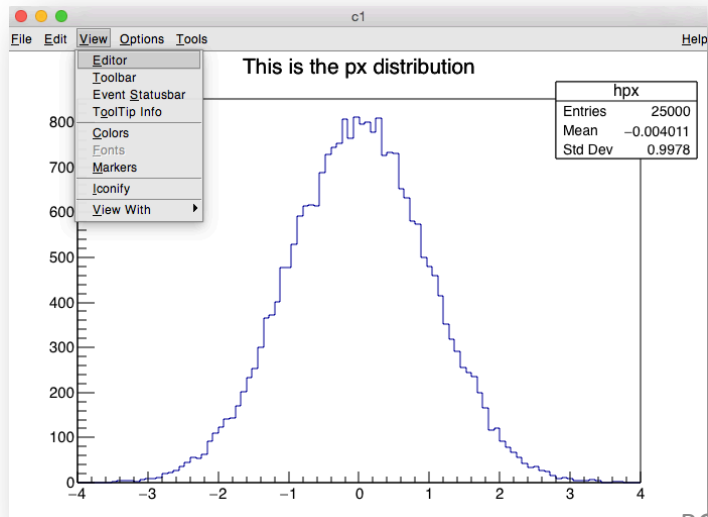
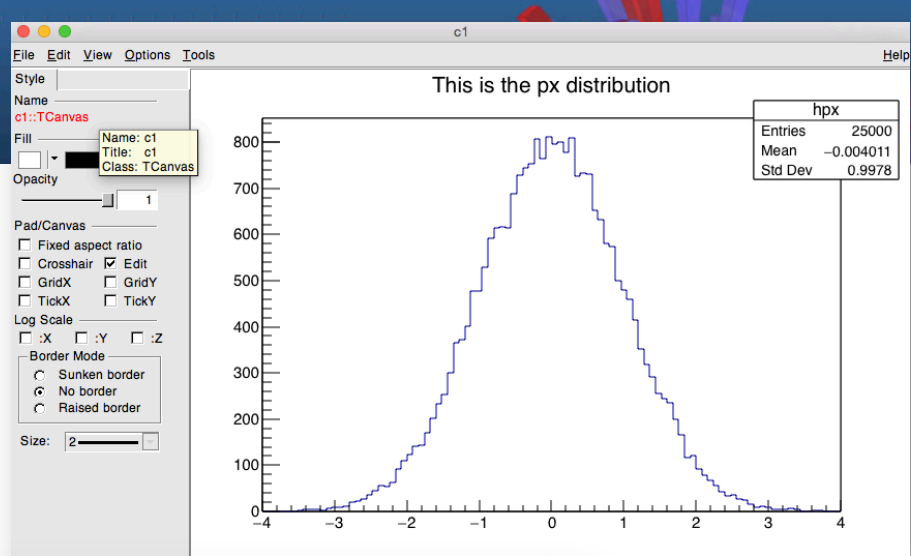
**Is this a good fit?  
Why?**

# Interactive ROOT

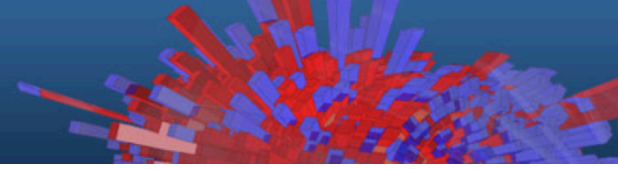
Look at one of your plots again and move the mouse across.

You will notice that this is much more than a static picture !

You can interact with objects and manipulate them. **Try it !!**

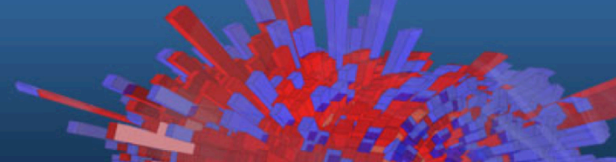






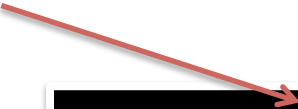
General Remarks  
A more complete example  
Summary of Visual effects  
Interpretation and Compilation

# General Remarks



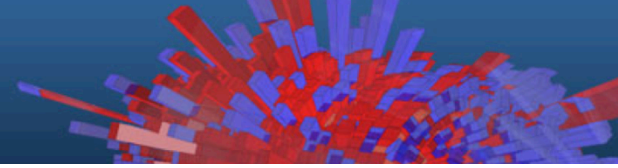
We have seen how to interactively type lines at the prompt.  
The next step is to write “ROOT Macros” – lightweight programs  
The general structure for a macro stored in file *MacroName.C* is:

**Function, no main,  
same name as the file**



```
void MacroName() {  
    <      ...  
    your lines of C++ code  
    >  
    ...  
}
```

# Running a Macro



The macro is executed at the system prompt by typing:

```
> root MacroName.C
```

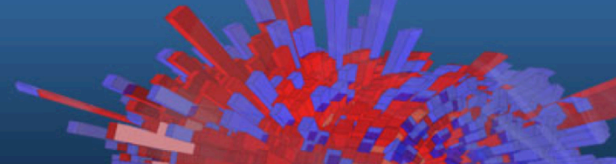
or executed at the ROOT prompt using .x:

```
> root  
root [0] .x MacroName.C
```

or it can be loaded into a ROOT session and then be executed by typing:

```
root [0].L MacroName.C  
root [1] MacroName();
```

# A More Complex Example



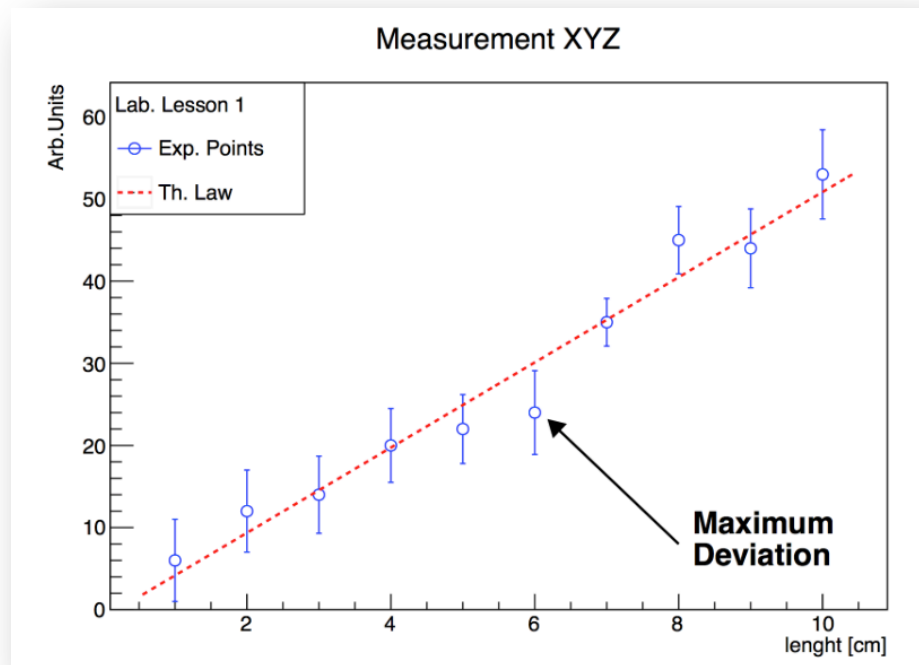
The example in section 3.2 of the ROOT primer, is a typical task in data analysis, a macro that constructs a graph with errors, fits a (linear) model to it and saves it as an image.

Let's inspect it together.

# A More Complex Example

And Run it!

```
> root macro1.C
```



Macro1\_cpp

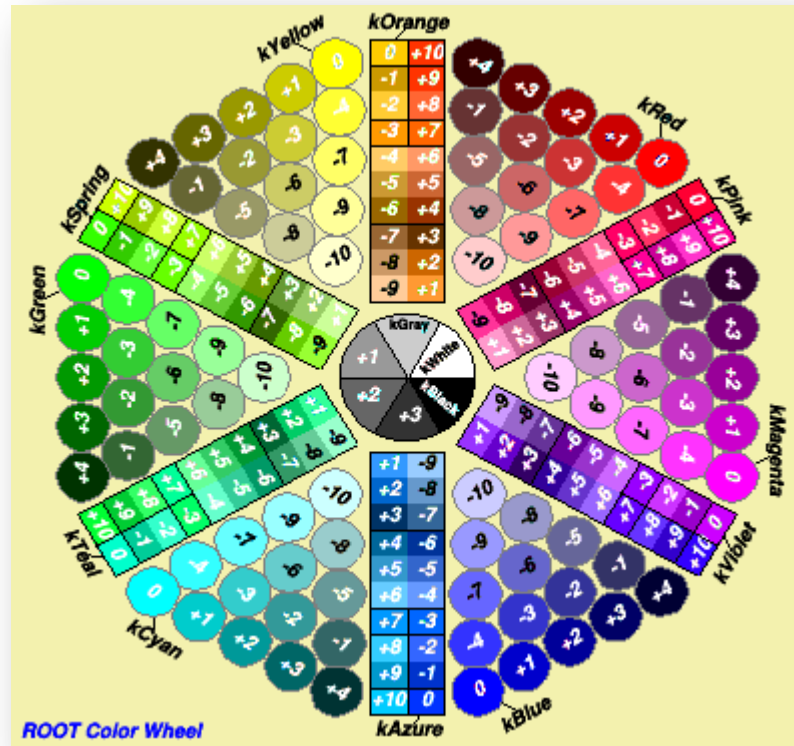
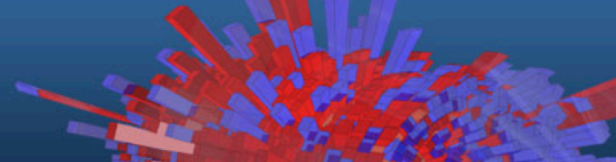
# Summary of Visual Effects



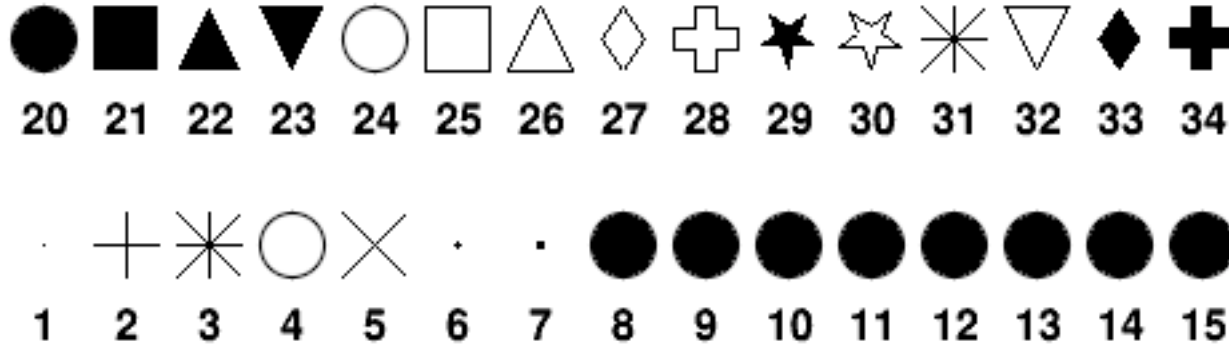
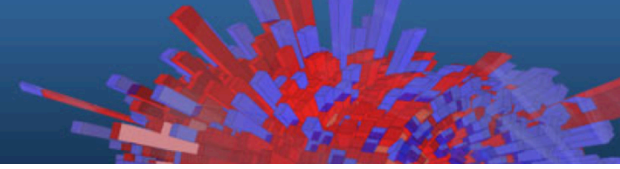
- **Colours and Graph Markers:** To specify a colour, some identifiers like `kWhite`, `kRed` or `kBlue` can be used for markers, lines, arrows etc. The complete summary of colours is represented by the ROOT “colour wheel”. ROOT provides several graphics markers like triangles, crosses or stars.
- **Arrows and Lines:** The class representing arrows is `TArrow`, which inherits from `TLine`. The constructors of lines and arrows always contain the coordinates of the endpoints.
- **Text:** A possibility to add text in plots is provided by the `TLatex` class. Latex mathematical symbols are automatically interpreted, you just need to replace the “\” by a “#”.



# TColorWheel



# The Family of Markers



kDot=1, kPlus, kStar, kCircle=4, kMultiply=5,  
kFullDotSmall=6, kFullDotMedium=7, kFullDotLarge=8,  
kFullCircle=20, kFullSquare=21, kFullTriangleUp=22,  
kFullTriangleDown=23, kOpenCircle=24, kOpenSquare=25,  
kOpenTriangleUp=26, kOpenDiamond=27, kOpenCross=28,  
kFullStar=29, kOpenStar=30, kOpenTriangleDown=32,  
kFullDiamond=33, kFullCross=34

Also available  
through more  
friendly names 😊

# Interpretation and Compilation

We have seen how ROOT interprets and “just in time compiles” code. ROOT also allows to compile code “traditionally”. At the ROOT prompt:

```
root [1] .L macro1.C+  
root [2] macro1()
```

**Generate shared library and  
execute function**



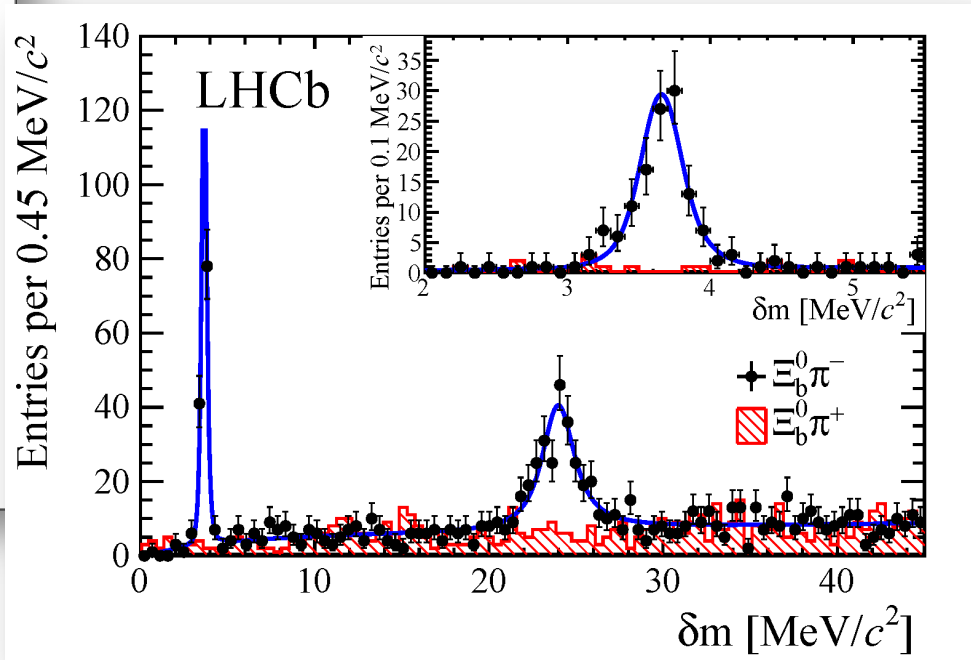
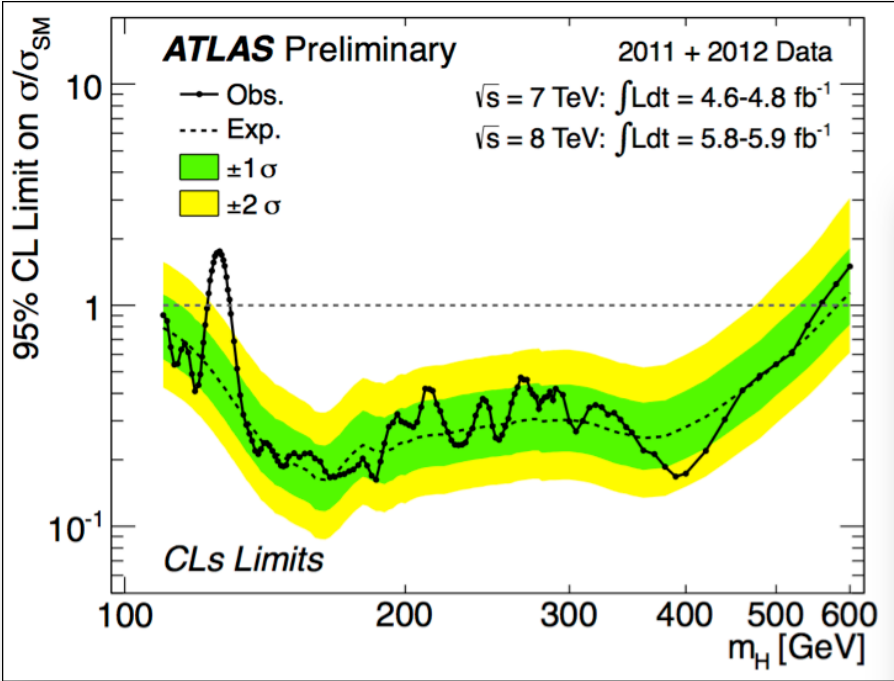
ExampleMacro.C

```
int main() {  
    ExampleMacro();  
    return 0;  
}
```

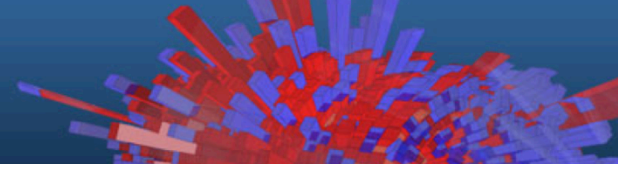
ROOT libraries can be also used to produce standalone, compiled applications:

```
> g++ -o ExampleMacro ExampleMacro.C `root-config --cflags --libs`  
> ./ExampleMacro
```

# More about Graphs and Histograms

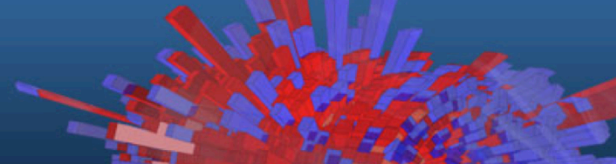


# Graphs



Read Graph Points from File  
Polar Graphs  
2D Graphs  
Multiple graphs

# From an ASCII File



To build a graph, experimental data can be read from an ASCII file (i.e. standard text) using this constructor:

```
TGraphErrors(const char *filename,  
             const char *format="%lg %lg %lg %lg",  
             Option_t *option="");
```

Let's have a look to macro2.C (section 4.1 in the Primer).



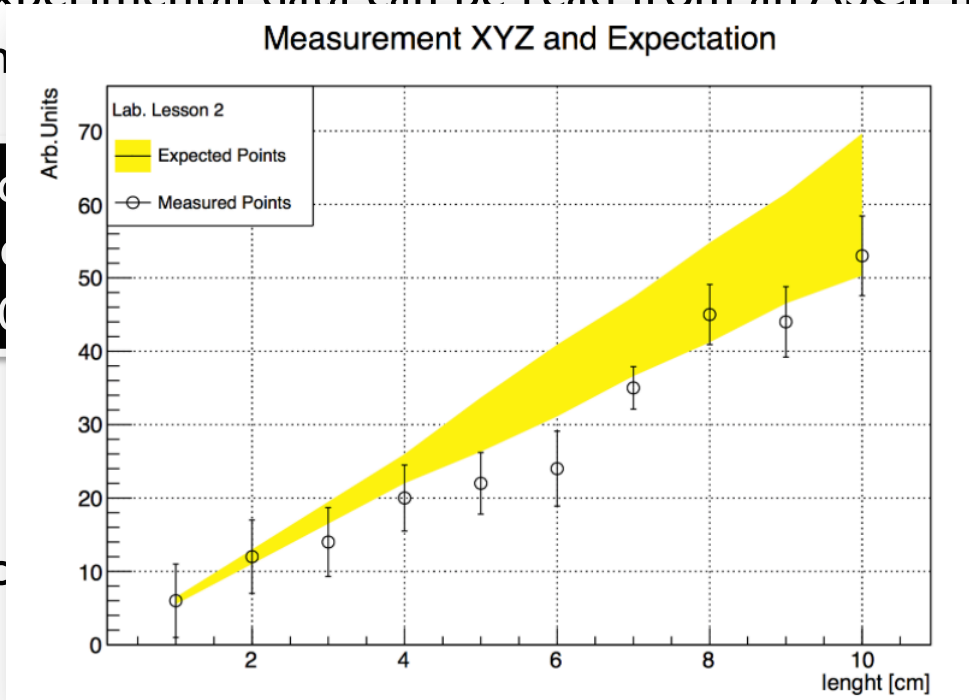
**Macro2\_cpp**

# From an ASCII File

To build a graph, experimental data can be read from an ASCII file (i.e. standard text) using this con

```
TGraphErrors(
```

Let's have a look to



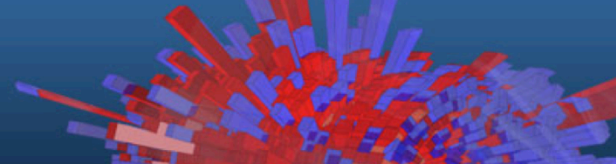
```
lg",
```



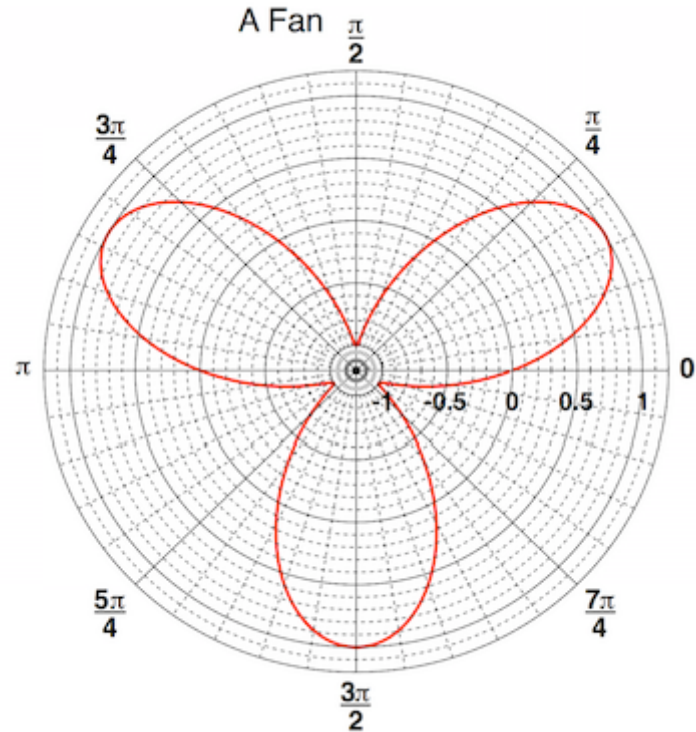
Macro2\_cpp



# Polar Graphs



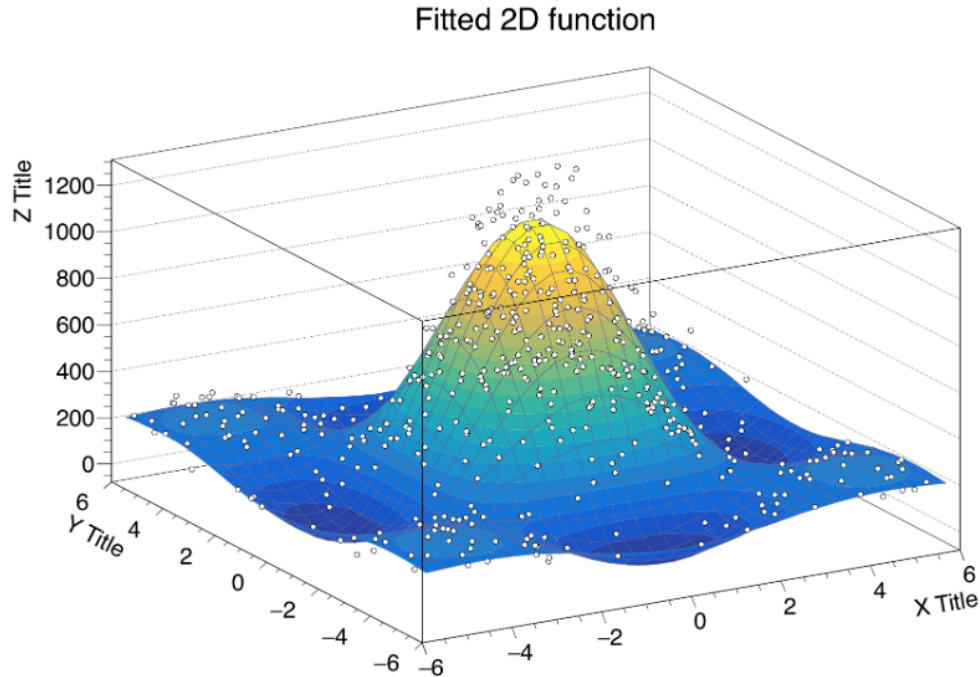
Graphs can also be displayed in polar coordinate like in *macro3.C* (section 4.2 in the Primer):



Macro3\_cpp

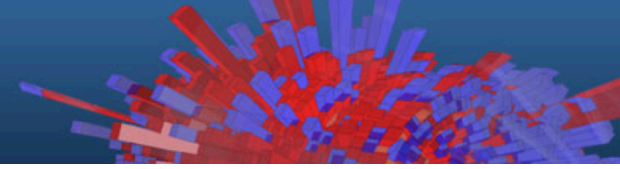
# 2D Graphs

Bi-dimensional graphs can be created in ROOT with the `TGraph2DErrors` class. `macro4.C`, described in Primer's section 4.3, gives a nice example:

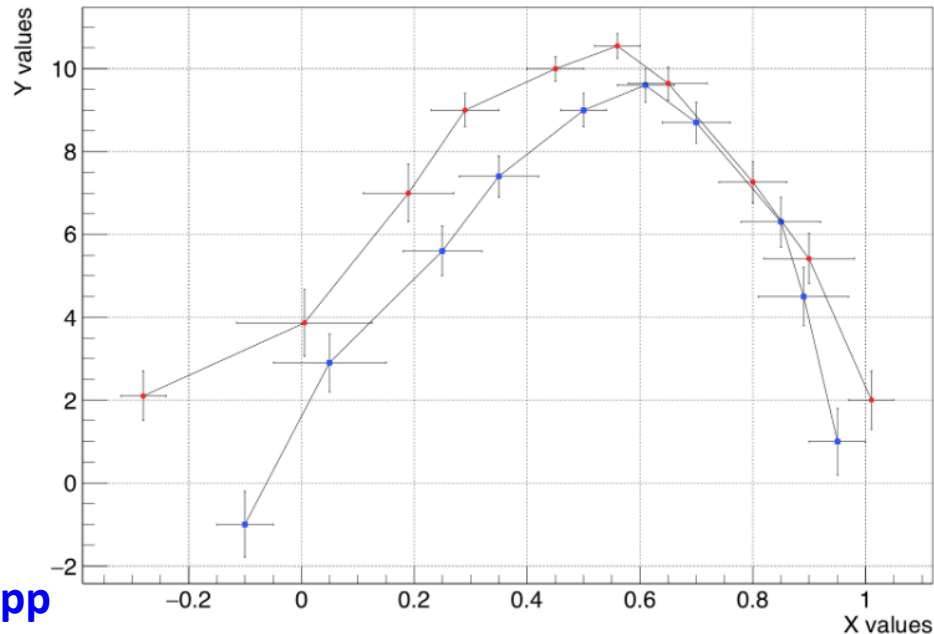


 `Macro4_cpp`

# Multiple Graphs

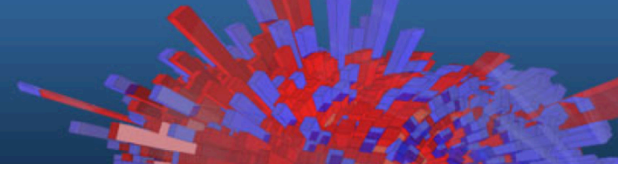


It is sometimes useful to group graphs in a single entity, for instance to compute a common axis system. The class *TMultiGraph* described in section 4.4 of the Primer allows that.



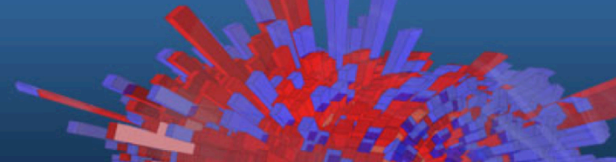
**Multigraph\_cpp**

# Histograms



Your First (well second!) Histogram  
Add and Divide Histograms  
Two-dimensional Histograms  
Multiple Histograms

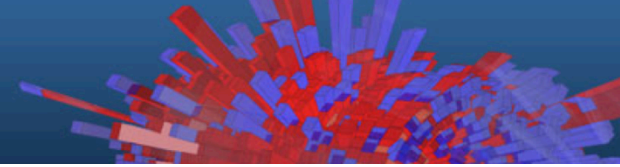
# Exercise



Write a macro to visualise a Poisson distribution in a histogram

- Create a 1D histogram the bins of which are double precision numbers
- The max number of counts collected is 15 (max value on the x axis)
- Use a random generator to generate 1000 Poissonian counts,  $\mu=4$
- Properly set the title and axes names, fill the histogram in blue
- Fit it, programmatically or with the fit panel (right click on the histogram)

# Exercise - Optional

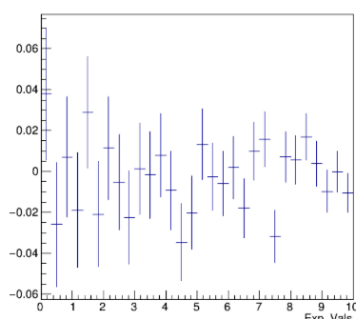
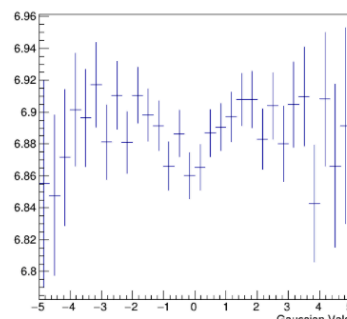
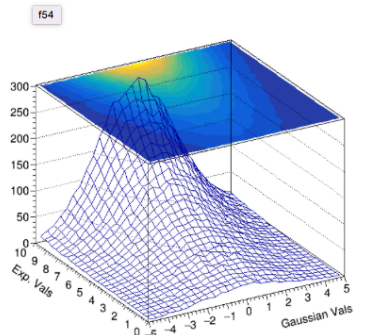
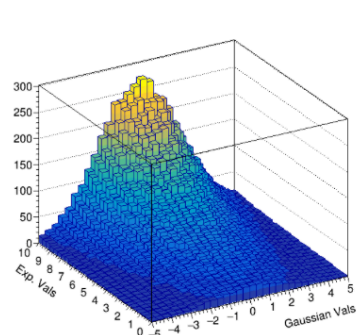
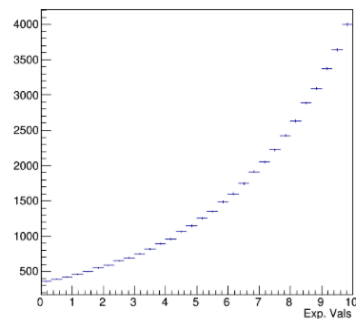
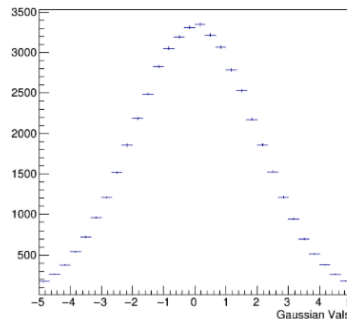
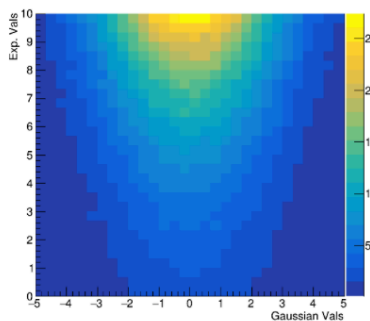
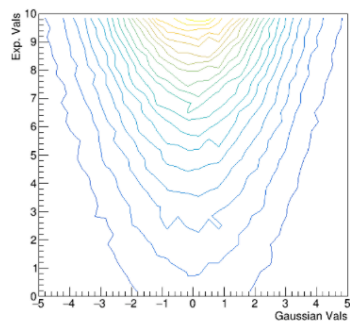


Create a macro that draws the sum, difference and ratio of two histograms

- Create three pairs of histograms, fill them randomly with normally distributed numbers (TH1::FillRandom(“gaus”))
- Divide, sum and subtract them
  - Useful methods:
    - TH1::Divide(const TH1\*),
    - TH1::Add(const TH1\*, Double\_t) the second parameter is a weight
- Nota bene: for every plot a different canvas has to be created and before drawing, one has to “cd” into it
  - TCanvas c; c.cd();

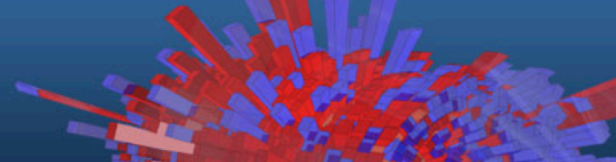
# Two Dimensional Histograms

Two-dimensional histograms are a very useful tool, for example to inspect correlations between variables, as in the example in section 5.3 of the Primer:





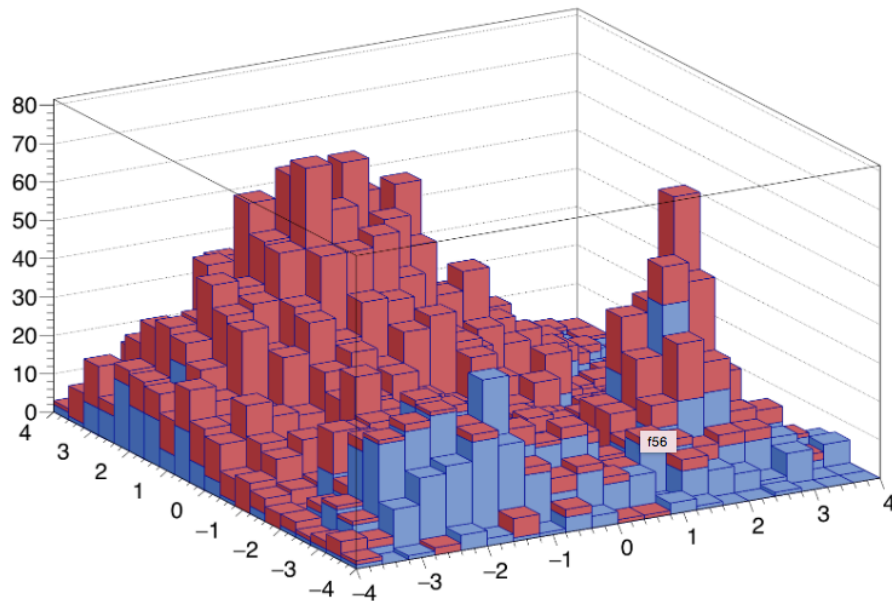
# Multiple Histograms



The example in section 5.4 shows how to group histograms in a single entity call a “stack”.

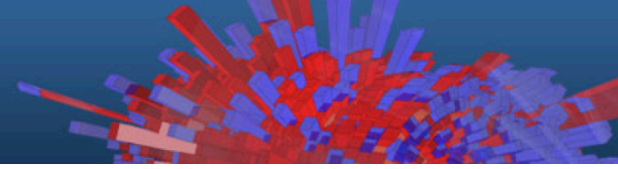
## Class THStack

Stacked 2D histograms



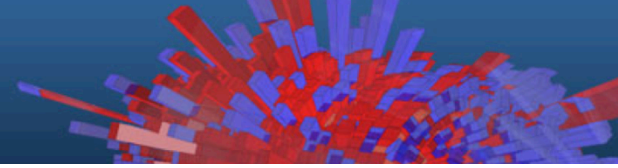
THStack\_cpp

# Input and Output



Storing Objects  
N-tuples

# Storing Objects in a File

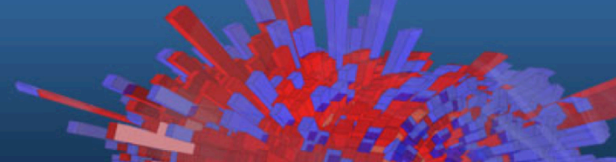


- ROOT allows to store C++ objects on disk (natively the language cannot)
- All ROOT objects\* can be written on disk via the Write method
  - In general, all instances of classes with dictionaries\*\*
- Two ways of storing: row wise and column wise
  - Single object dump and N-tuple like storage respectively
- Feature widely used, e.g. by all LHC experiments

\* All objects which are instances of classes inheriting from TObject

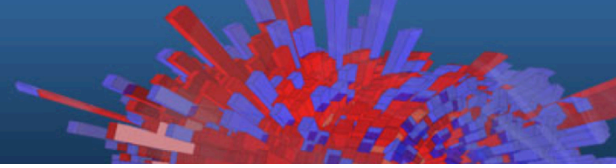
\*\* This discussion is beyond the scope of this lecture

# An Example



```
TFile out_file("my_rootfile.root","RECREATE"); // Open a Tfile
TH1F h("my_histogram","My Title;X;# of entries",100,-5,5);
h.FillRandom("gaus");
h.Write(); // Write the histogram in the file
out_file.Close(); // Close the file
```

# Exercise



## Inspect the content of a file with the TBrowser

- Create a file copying the lines of the previous slide at the prompt
- Quit the command line interpreter
- Boot ROOT opening the file: `root my_rootfile.root`
- Type: `TBrowser myBrowser`
- Inspect the content of the file

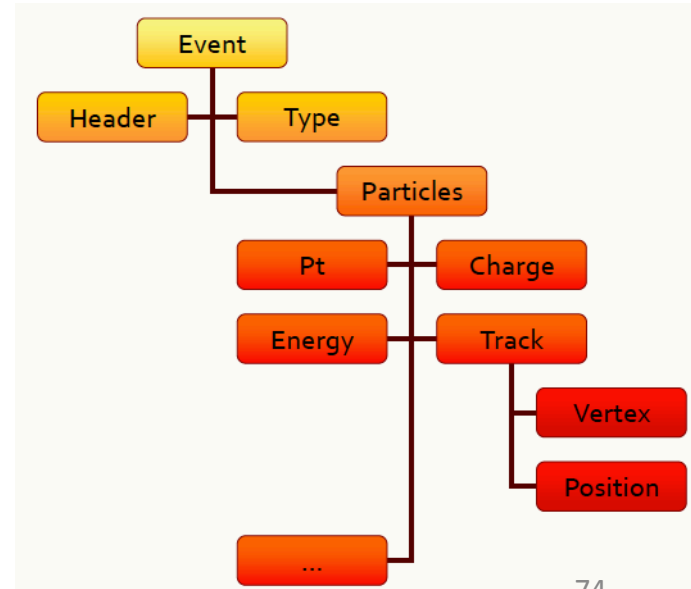
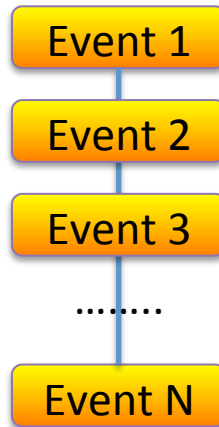
# Trees

- The TTree is the data structure ROOT provides to store large quantities of same types objects
- Organised in branches, each one holding objects
- Organised in independent events, e.g. collision events
- Efficient disk space usage, optimised I/O runtime

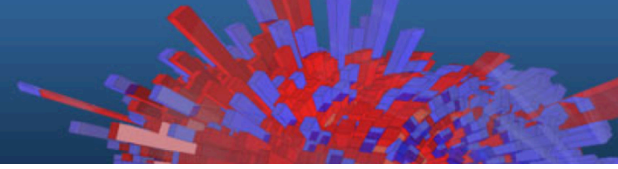
x	y	z
-1.10228	-1.79939	4.452822
1.867178	-0.59662	3.842313
-0.52418	1.868521	3.766139
-0.38061	0.969128	1.084074
0.552454	-0.21231	0.350281
-0.18495	1.187305	1.443902
0.205643	-0.77015	0.635417
1.079222	-0.32739	1.271904
-0.27492	-1.72143	3.038899
2.047779	-0.06268	4.197329
-0.45868	-1.44322	2.293266
0.304731	-0.88464	0.875442
-0.71234	-0.22239	0.556881
-0.27187	1.181767	1.470484
0.886202	-0.65411	1.213209
-2.03555	0.527648	4.421883
-1.45905	-0.464	2.344113
1.230661	-0.00565	1.514559
		3.562347

→

LEP style flat n-tuples evolved in more efficient trees (fast access, read ahead)



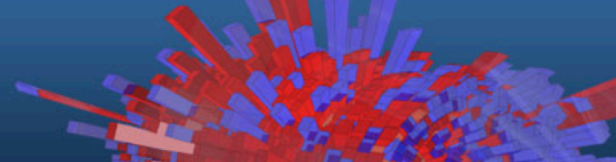
# Ntuples



- The `TNtuple` is a simplified version of the `TTree`: store floating point numbers
- As powerful for analysis



# Example



Primer macro

*write\_ntuple\_to\_file.C*



*write\_ntuple\_to\_file\_cpp*

```
TFile ofile("conductivity_experiment.root","RECREATE");
TNtuple cond_data("cond_data",
                  "Example N-Tuple",
                  "Potential:Current:Temperature:Pressure");
TRandom3 rndm; // We'll fill random values
float pot,cur,temp,pres;
for (int i=0;i<10000;++i) {
    pot = rndm.Uniform(0.,10.); // get voltage
    temp = rndm.Uniform(250.,350.); // get temperature
    pres = rndm.Uniform(0.5,1.5); // get pressure
    cur = pot/(10.+0.05*(temp-300.)-0.2*(pres-1.)); // current
    // add some random smearing (measurement errors)
    pot* = rndm.Gaus(1.,0.01); temp+=rndm.Gaus(0.,0.3);
    pres*= rndm.Gaus(1.,0.02); cur*=rndm.Gaus(1.,0.01);
    // write to ntuple
    cond_data.Fill(pot,cur,temp,pres);
}
// Save the ntuple and close the file
cond_data.Write(); ofile.Close();
```

# Exercise: Potential of the Tree



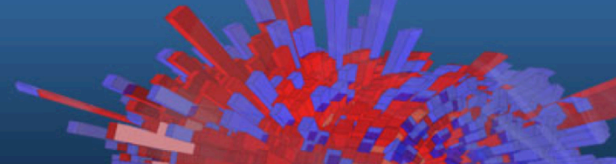
- Run the `write_ntuple_to_file.C` macro
- Open the file in the TBrowser
- Create plots clicking on the leaves

# Accessing Complex Trees

- TTreeReader class: tool to access complex trees in a type-safe manner
  - Not only floating point numbers as in TNtuple, but all objects!

```
// Access a TTree called "MyTree" in the file:  
TTreeReader reader("MyTree", file);  
// Establish links with two of the branches  
TTreeReaderValue<float> rvMissingET(reader, "missingET");  
TTreeReaderValue<std::vector<Muon>> rvMuons(reader, "muons");
```

# Accessing the Data

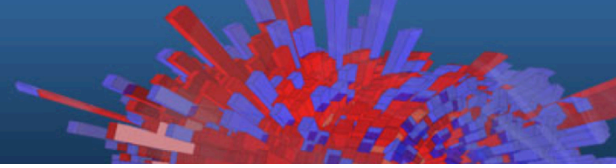


```
// Loop through all the TTree's entries
// It behaves like an iterator...
while (reader.Next()) {
    float missingET = *rvMissingET;
    ...
    for (auto&& mu: rvMuons) { hist->Fill(pT); }
}
```



**TTreeReader\_Example\_cpp**

# PyROOT



- ROOT offers the possibility to interface to Python via a set of bindings called PyROOT
- Mix the power of C++ (compiled libraries) and flexibility of Python
- Killer application: JIT of C++ code from within Python
  - Real mix of the two languages

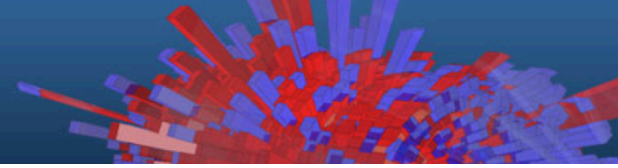
See Primer's section 8 for more details

Entry point to use ROOT from within Python:

```
import ROOT
```

All classes you now know can be accessed like `ROOT.TH1F`, `ROOT.TGraph`, ...

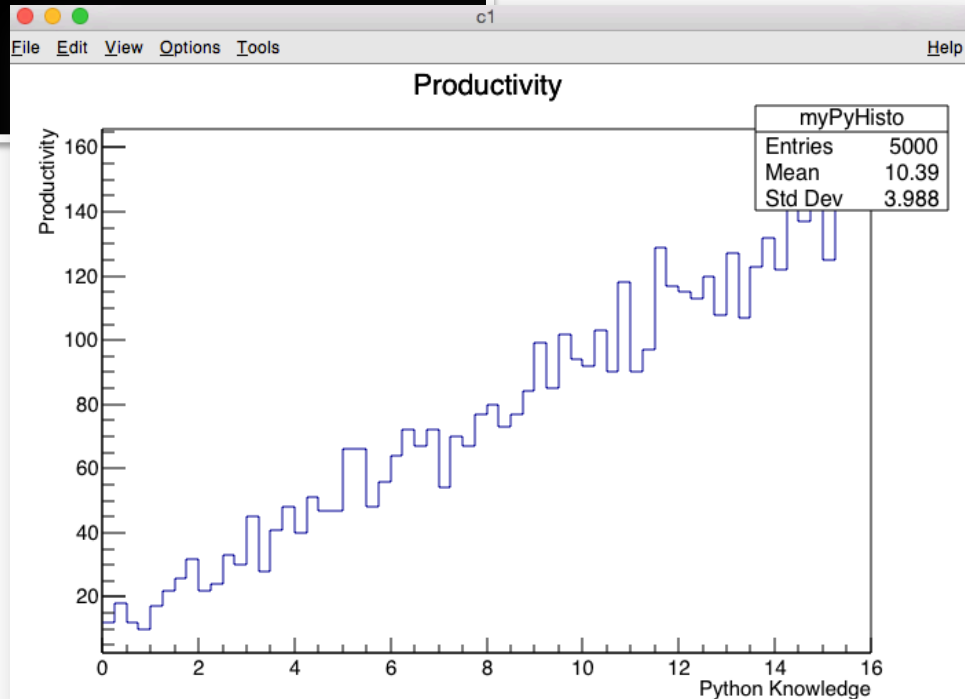
# Extempore Exercise



- Open the Python interpreter (type python)
- Import the ROOT module
- Create an histogram with 64 bins and a x axis ranging from 0 to 16
- Fill it with random numbers distributed according to a linear function (“pol0”)
- Change its line width with a thicker one
- Draw it!

# Extempore Exercise

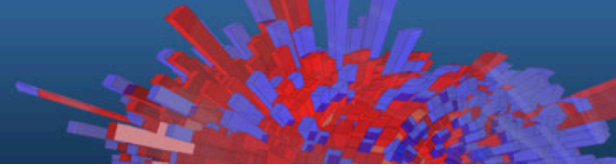
```
~> python
>>> import ROOT
>>> h = ROOT.TH1F("myPyHisto", "Productivity;Python Knowledge;Productivity",
64,0,16)
>>> h.FillRandom("pol1")
>>> h.Draw()
```



 [FillHistogram\\_Example\\_py](#)

 [TTreeAccess\\_Example\\_py](#)

# Review of the objectives



## **Objectives:**

- Become familiar with the ROOT toolkit
- Be able to use the C++ prompt
- Plot data
- Fit data
- Perform basic I/O operations