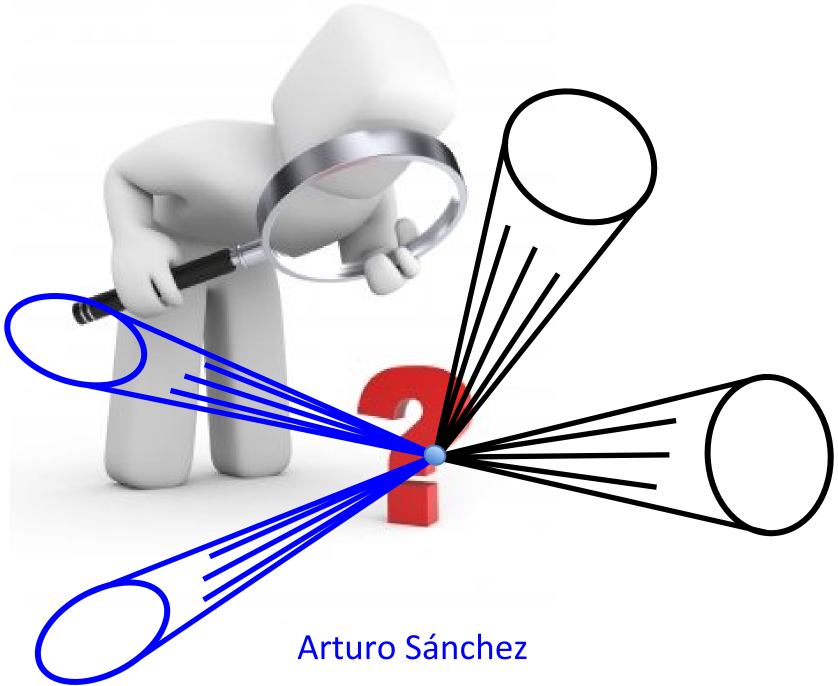
ROOT Analysis Framework



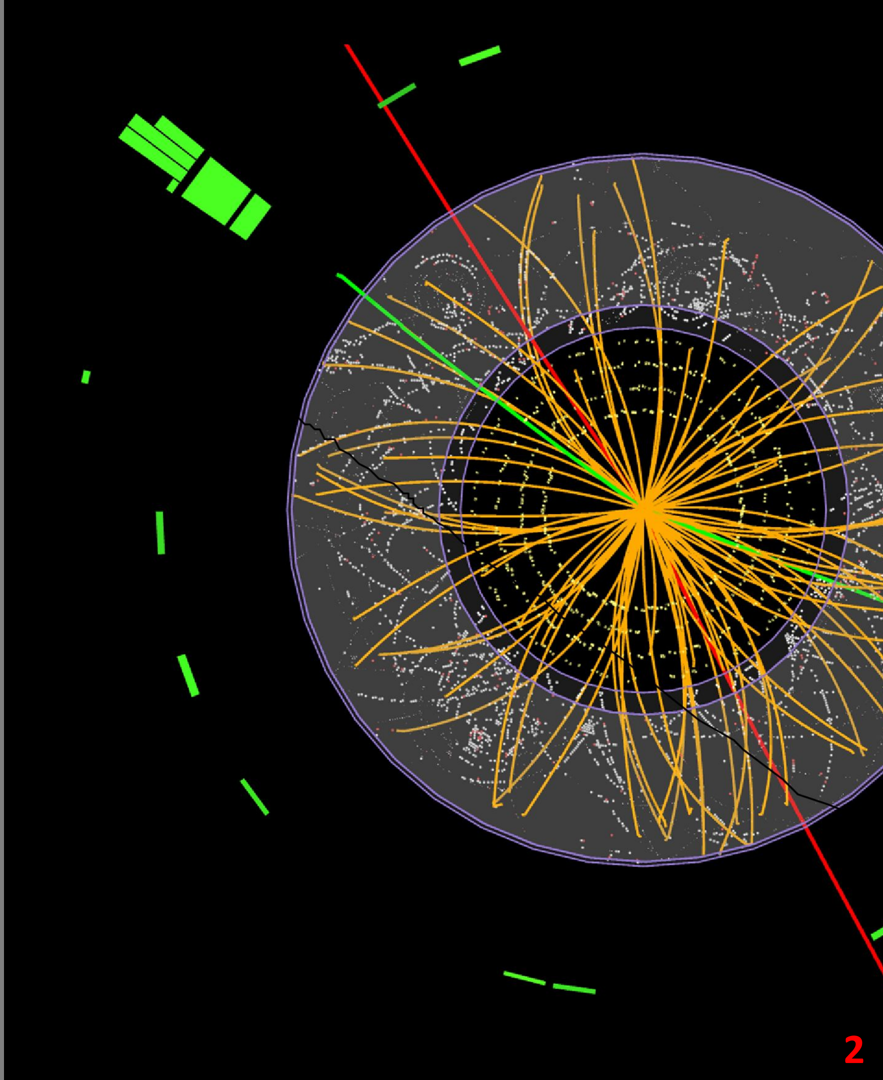
Arturo Sánchez

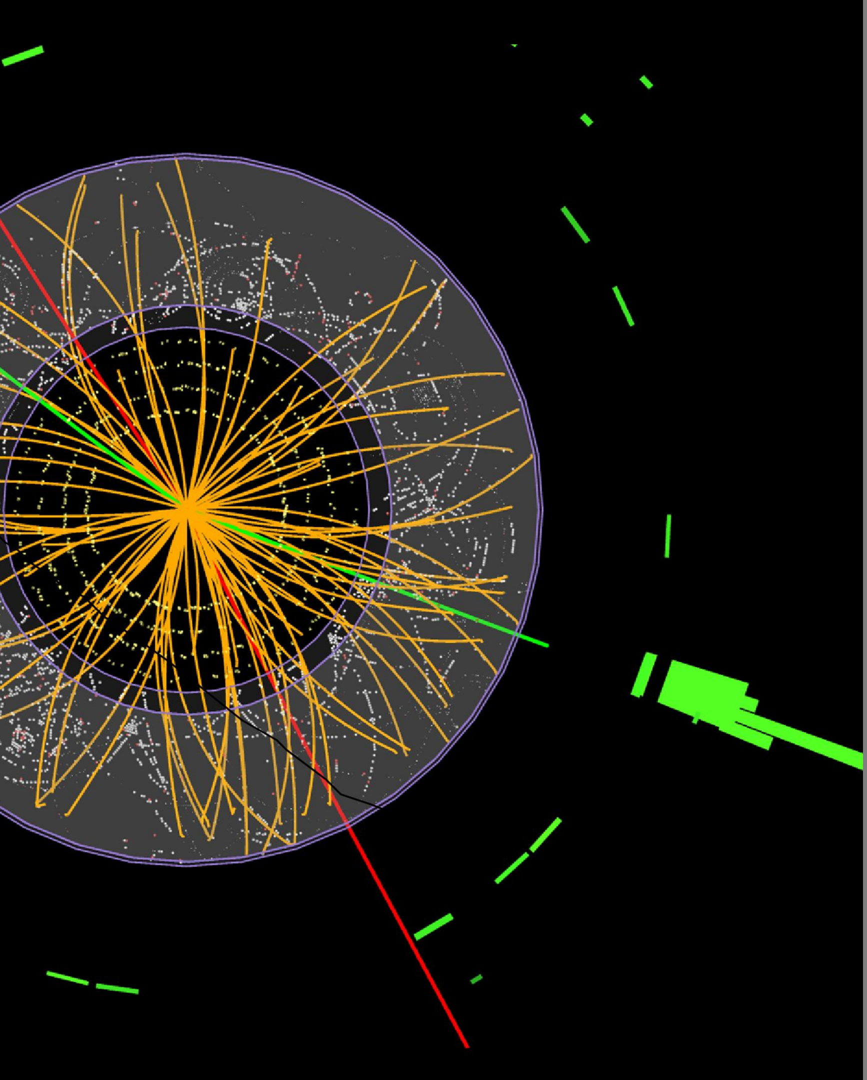
University of Naples and CEVALE2VE

Material from Danilo Piparo, Olivier Couet **CERN**

Module III - class I. April 14th, 2016

Today we will talk about a set of software tools or "framework" called: ROOT





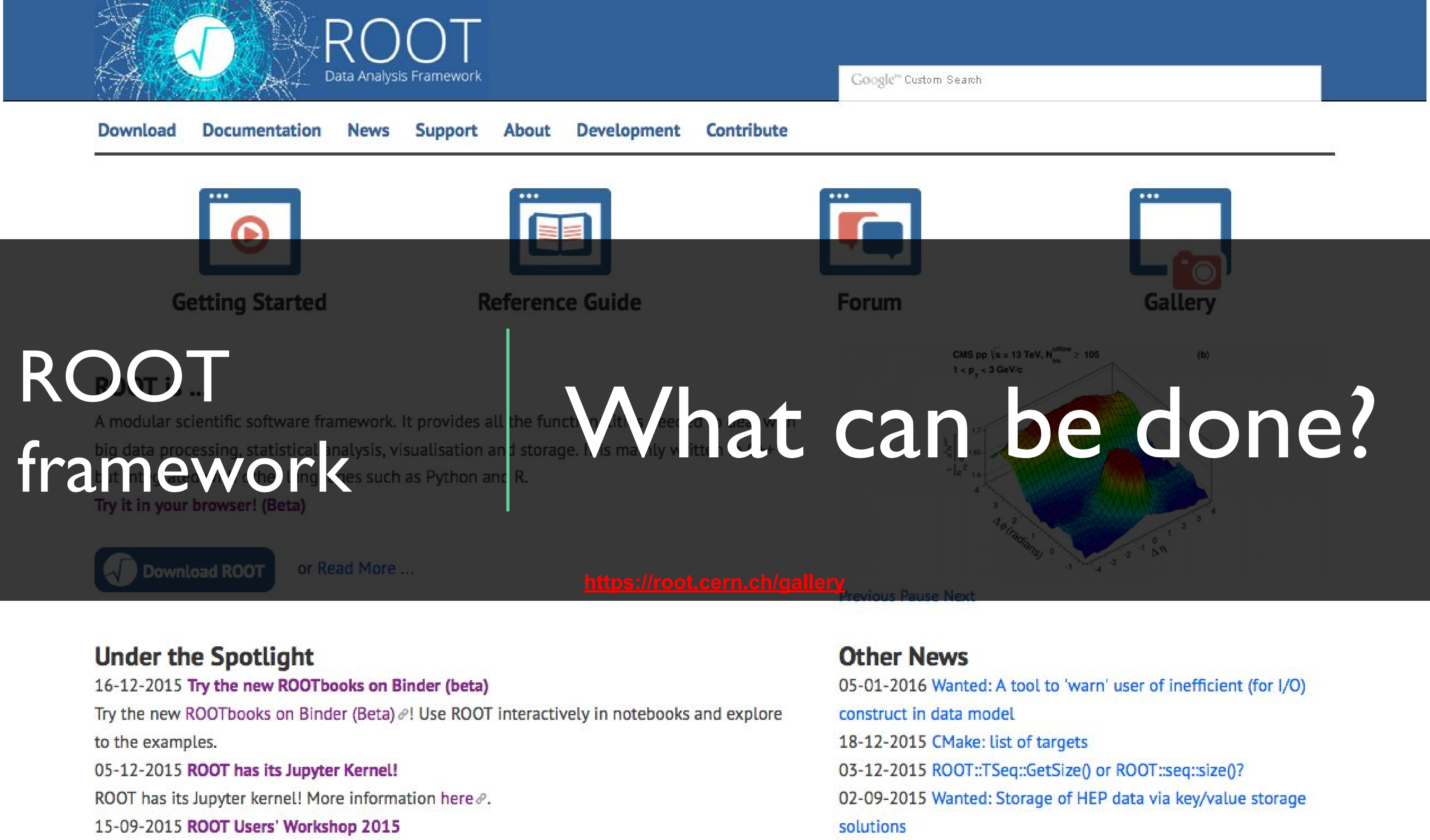
ROOT is an Open Source toolkit use in HEP and other fields

https://root.cern.ch/

ROOT framework

What can be done?

https://root.cern.ch/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)

or Read More ...

Under the Spotlight

16-12-2015 **Try the new ROOTbooks on Binder (beta)**

Try the new ROOTbooks on Binder (Beta)! Use ROOT interactively in notebooks and explore to the examples.

05-12-2015 **ROOT has its Jupyter Kernel!**

ROOT has its Jupyter kernel! More information here.

15-09-2015 **ROOT Users' Workshop 2015**

Other News

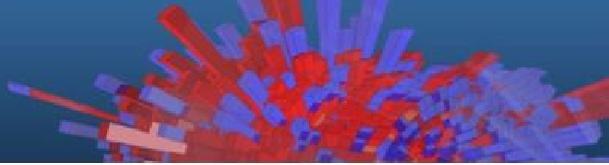
05-01-2016 Wanted: A tool to 'warn' user of inefficient (for I/O) construct in data model

18-12-2015 CMake: list of targets

03-12-2015 ROOT::TSeq::GetSize() or ROOT::seq::size()?

02-09-2015 Wanted: Storage of HEP data via key/value storage solutions

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

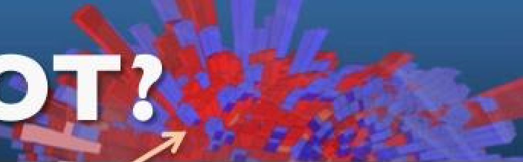
This Tutorial

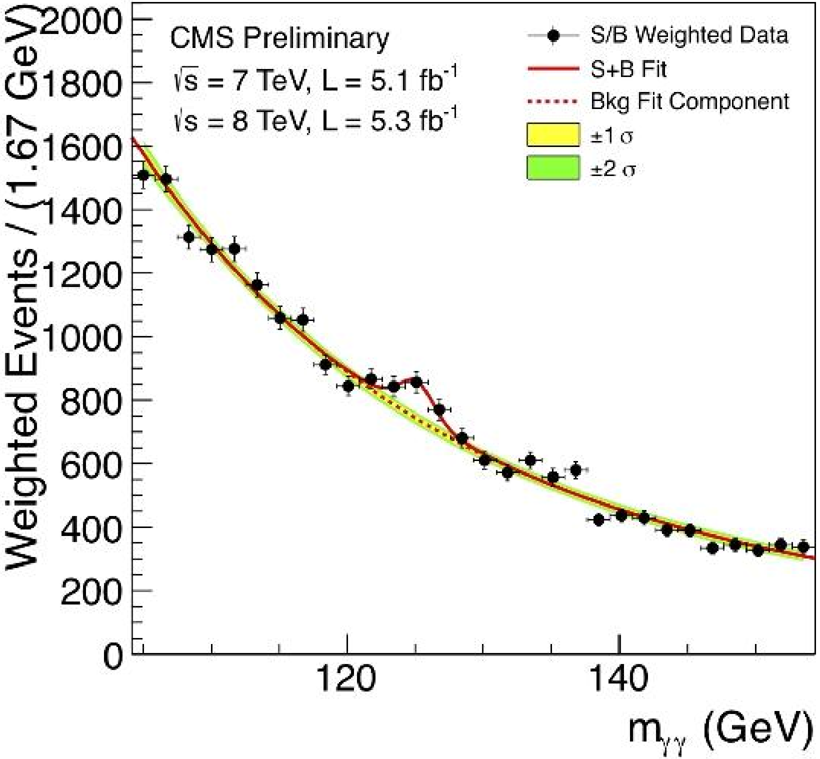
These slides are supported by the **“ROOT Primer”**

- Introductory booklet (~60 pages)
- Available on the ROOT website (html, epub, pdf)
- Code examples will be visualised
  - Signaled with name and the sign: 📘

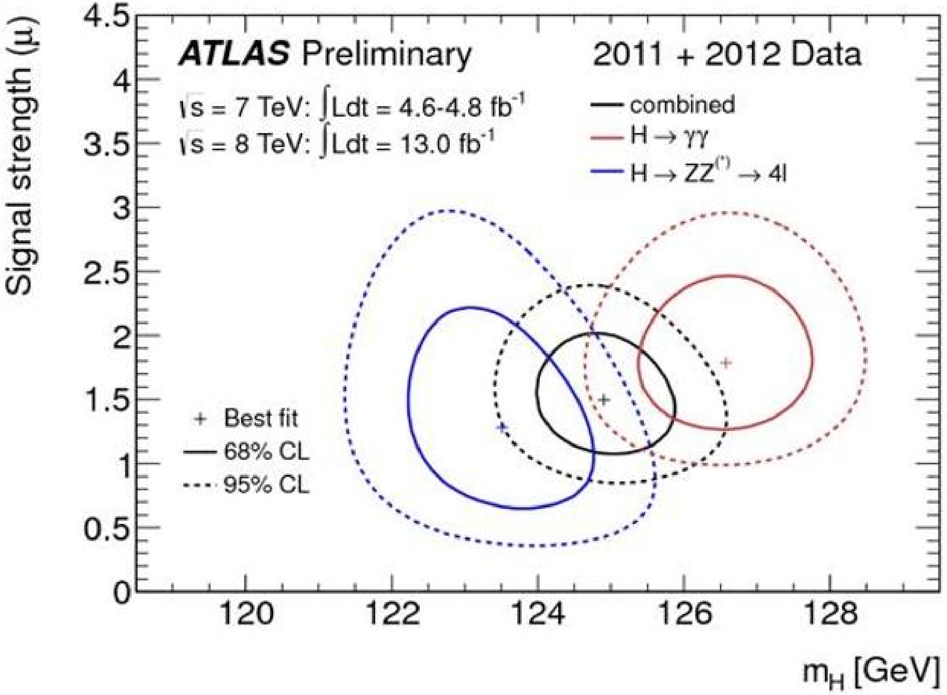
Two release series of ROOT are available: ROOT5 and ROOT6
This lecture refers to ROOT6, version 6.04

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!

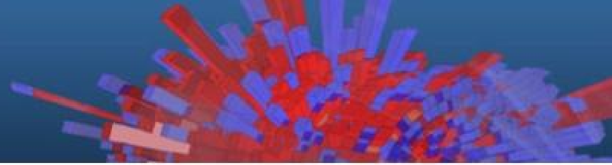
ROOT is written mainly in C++ (C++11 standard)

- 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

ROOT in a Nutshell

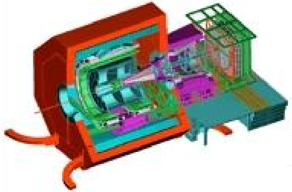


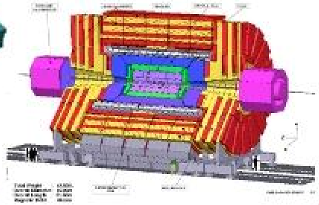
ROOT can be imagined as a family of building blocks for a variety of activities, for example:

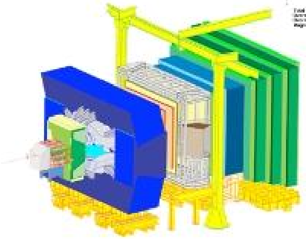
- Data analysis: histograms, graphs, trees
- I/O: row-wise, column-wise storage of **any** C++ object
- Statistical tools (RooFit/RooStats): rich modeling and statistical inference
- Math: non trivial functions (e.g. Erf, Bessel), optimised math functions (VDT)
- C++ interpretation: fully C++11 compliant
- 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

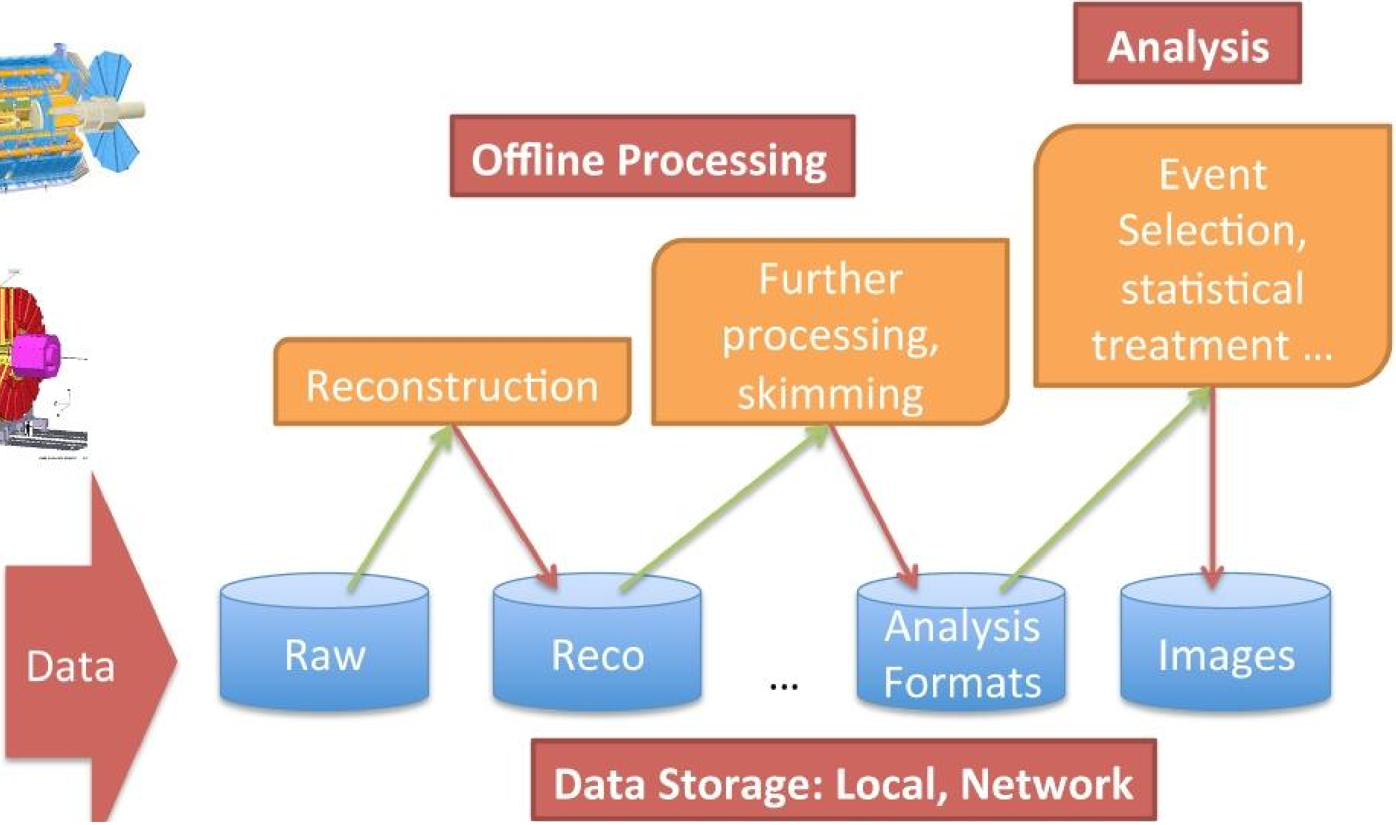
ROOT Application Domains

A selection of the experiments adopting ROOT









Analysis

Offline Processing

Event Selection, statistical treatment ...

Further processing, skimming

Reconstruction

Data

Raw

Reco

...

Analysis Formats

Images

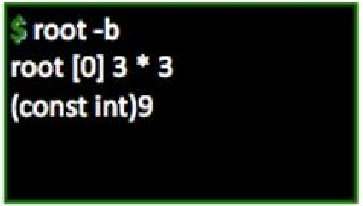
Data Storage: Local, Network

Event Filtering

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.



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

Can interpret "macros" (non compiled programs)

- Rapid prototyping possible

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

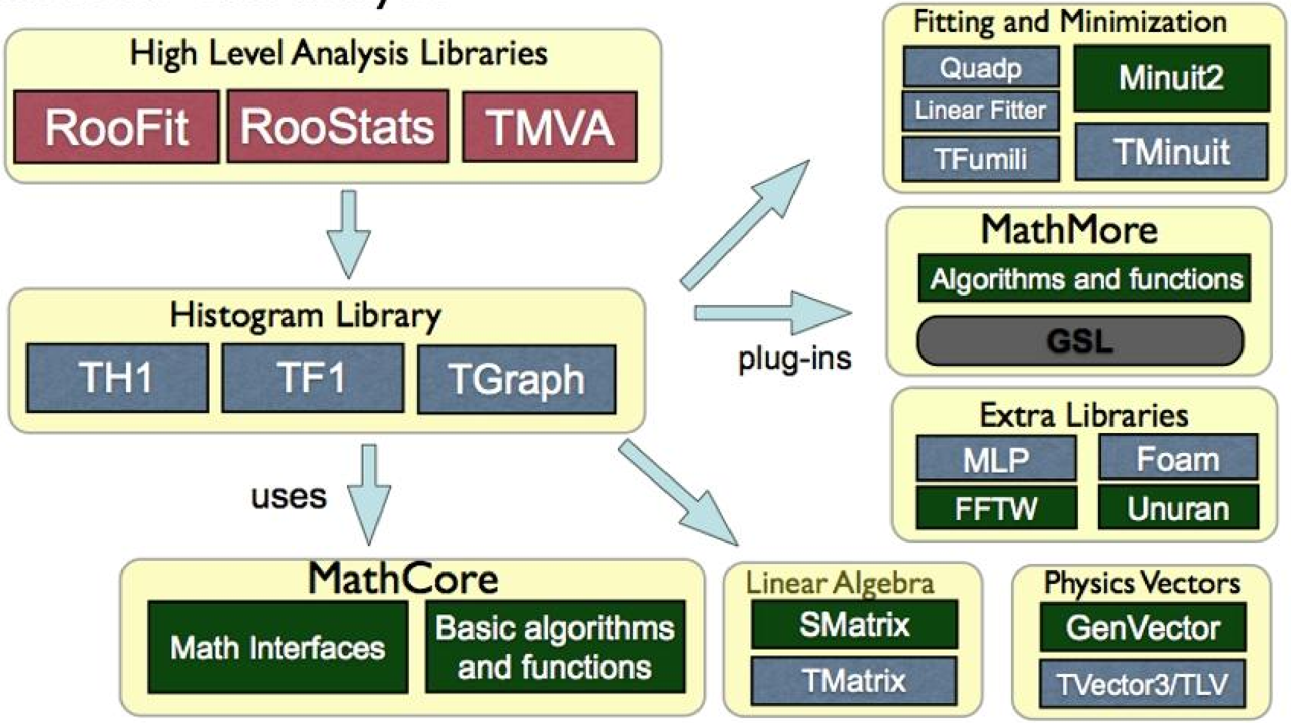
- Raw and column-wise streaming

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

Cornerstone for storage of experimental data

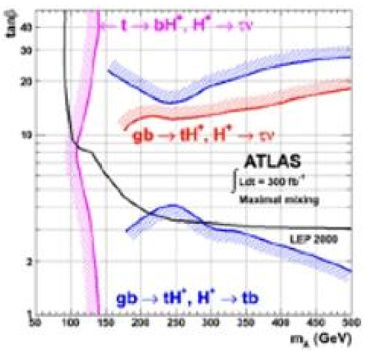
ROOT Math/Stats Libraries

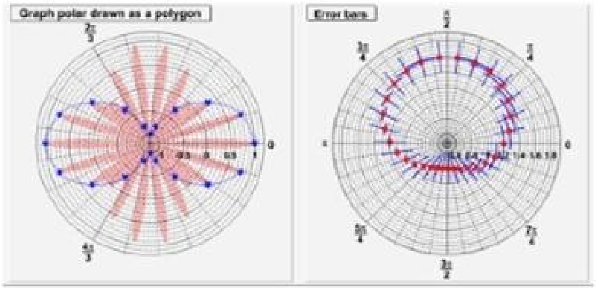
ROOT provides a reach set of mathematical libraries and tools needed for sophisticated statistical data analysis

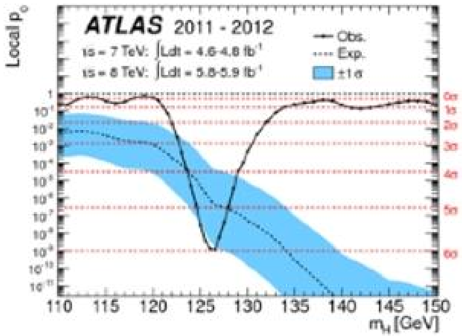


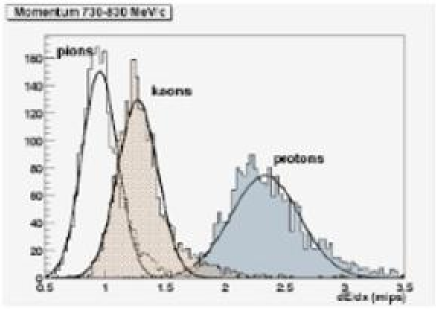
Graphics In ROOT

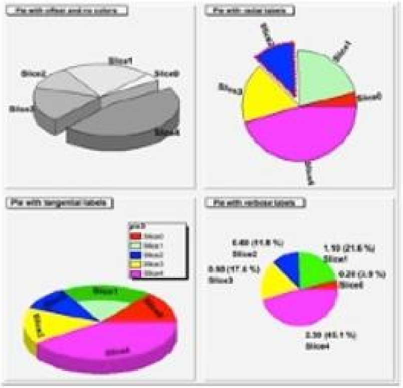
Many formats for data analysis, and not only, plots

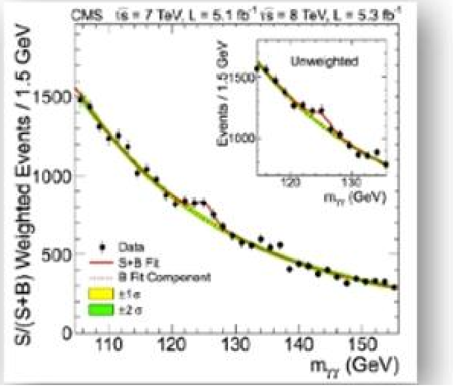










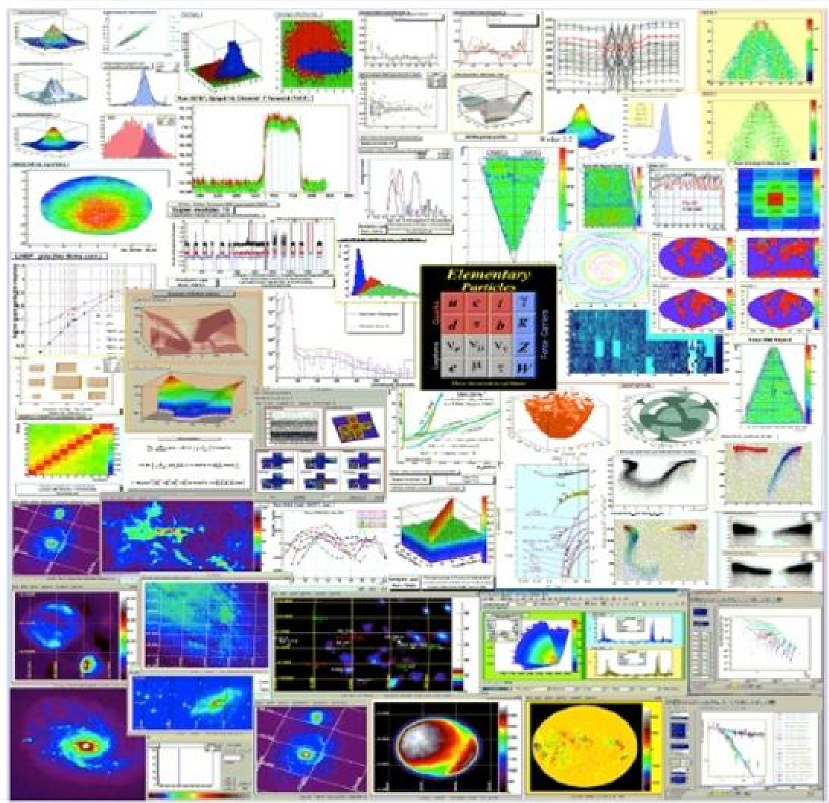


2D Graphics

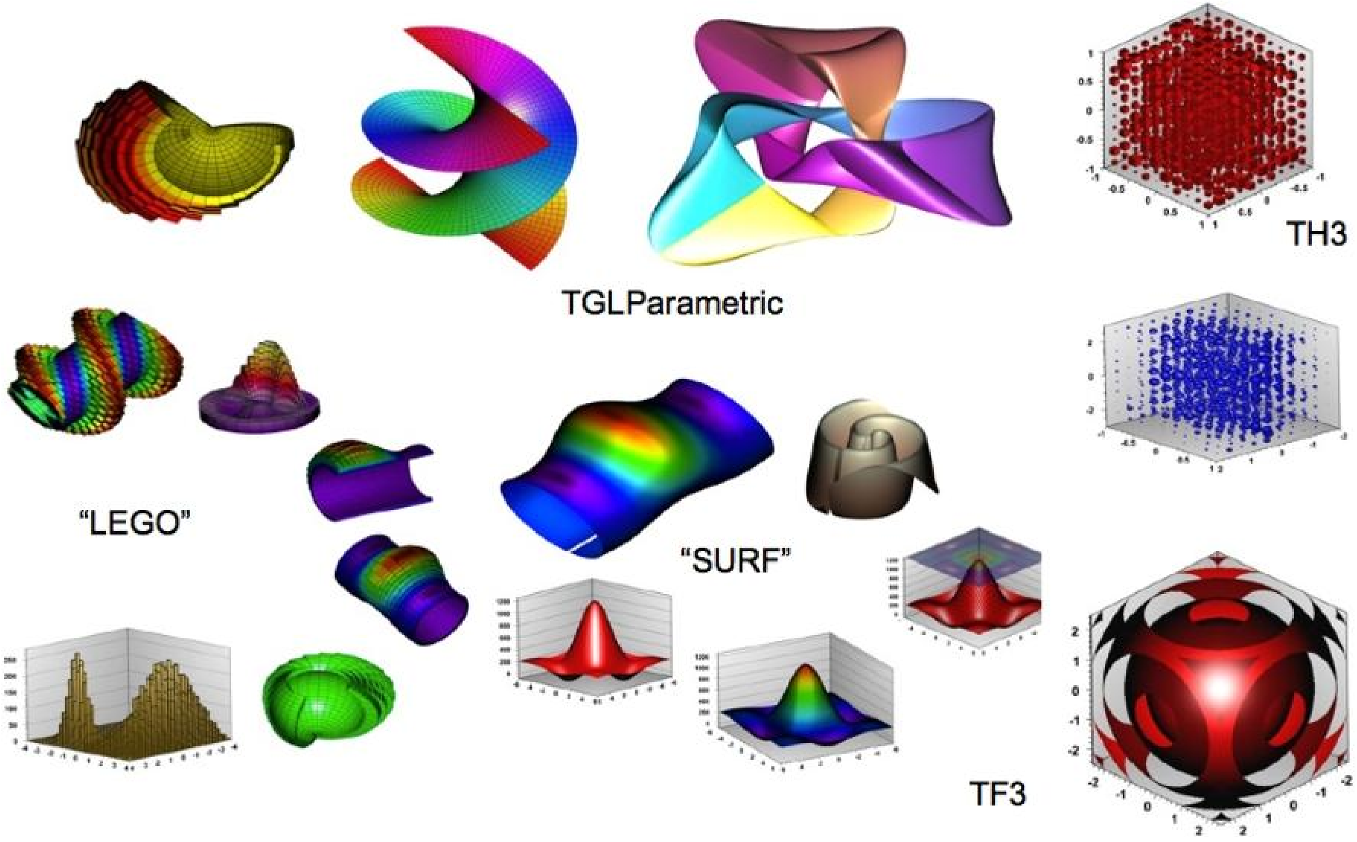
New functionalities added at every new release

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

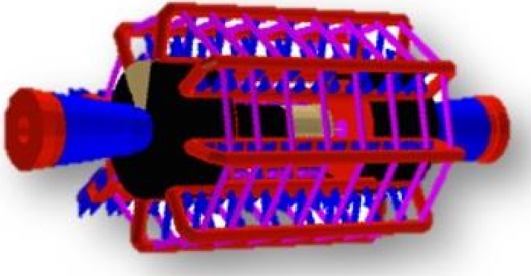
- Represent geometries as complex as LHC detectors

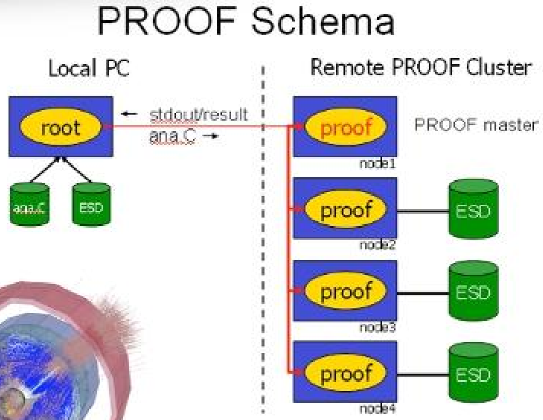
Event Display (EVE)

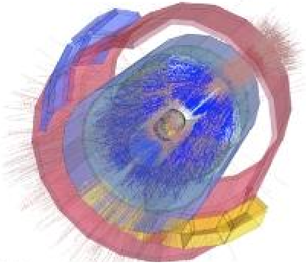
- Visualise particles collisions within detectors

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





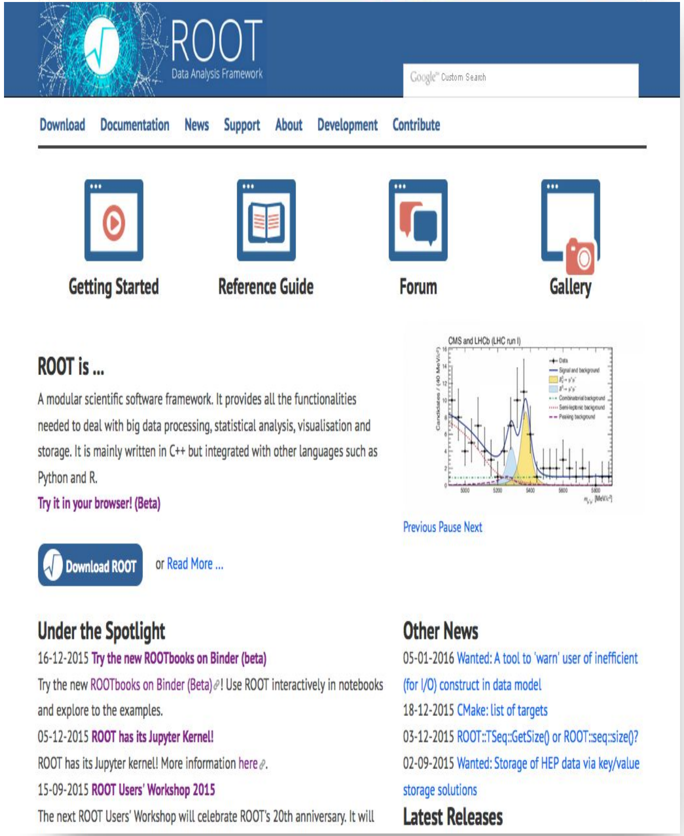


www.root.cern.ch

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!



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)

or Read More ...

Under the Spotlight

16-12-2015 **Try the new ROOTbooks on Binder (beta)**

Try the new ROOTbooks on Binder (Beta)! Use ROOT interactively in notebooks and explore to the examples.

05-12-2015 **ROOT has its Jupyter Kernel!**

ROOT has its Jupyter kernel! More information here.

15-09-2015 **ROOT Users' Workshop 2015**

The next ROOT Users' Workshop will celebrate ROOT's 20th anniversary. It will

Other News

05-01-2016 Wanted: A tool to 'warn' user of inefficient (for I/O) construct in data model

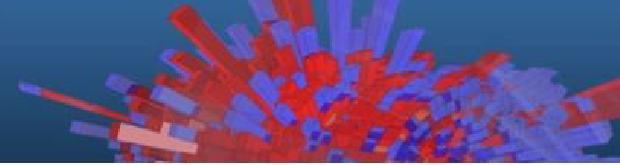
18-12-2015 CMake: list of targets

03-12-2015 ROOT::TSeq::GetSize() or ROOT::seq::size()?

02-09-2015 Wanted: Storage of HEP data via key/value storage solutions

Latest Releases

www.root.cern.ch



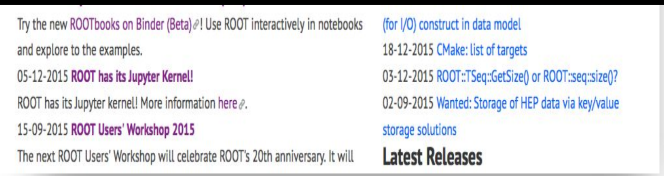
Useful installation video-tutorial at

http://www.cevale2ve.org/es/clases/tutoriales-root/

And macros in:

https://arturos.web.cern.ch/arturos/napoli/OutReach/macros/

Don’t hesitate to use it, even today!



Try the new ROOTbooks on Binder (Beta)! Use ROOT interactively in notebooks and explore to the examples.

05-12-2015 **ROOT has its Jupyter Kernel!**

ROOT has its Jupyter kernel! More information here.

15-09-2015 **ROOT Users' Workshop 2015**

The next ROOT Users' Workshop will celebrate ROOT's 20th anniversary. It will

(for I/O) construct in data model

18-12-2015 CMake: list of targets

03-12-2015 ROOT::TSeq::GetSize() or ROOT::seq::size()?

02-09-2015 Wanted: Storage of HEP data via key/value storage solutions

Latest Releases

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.

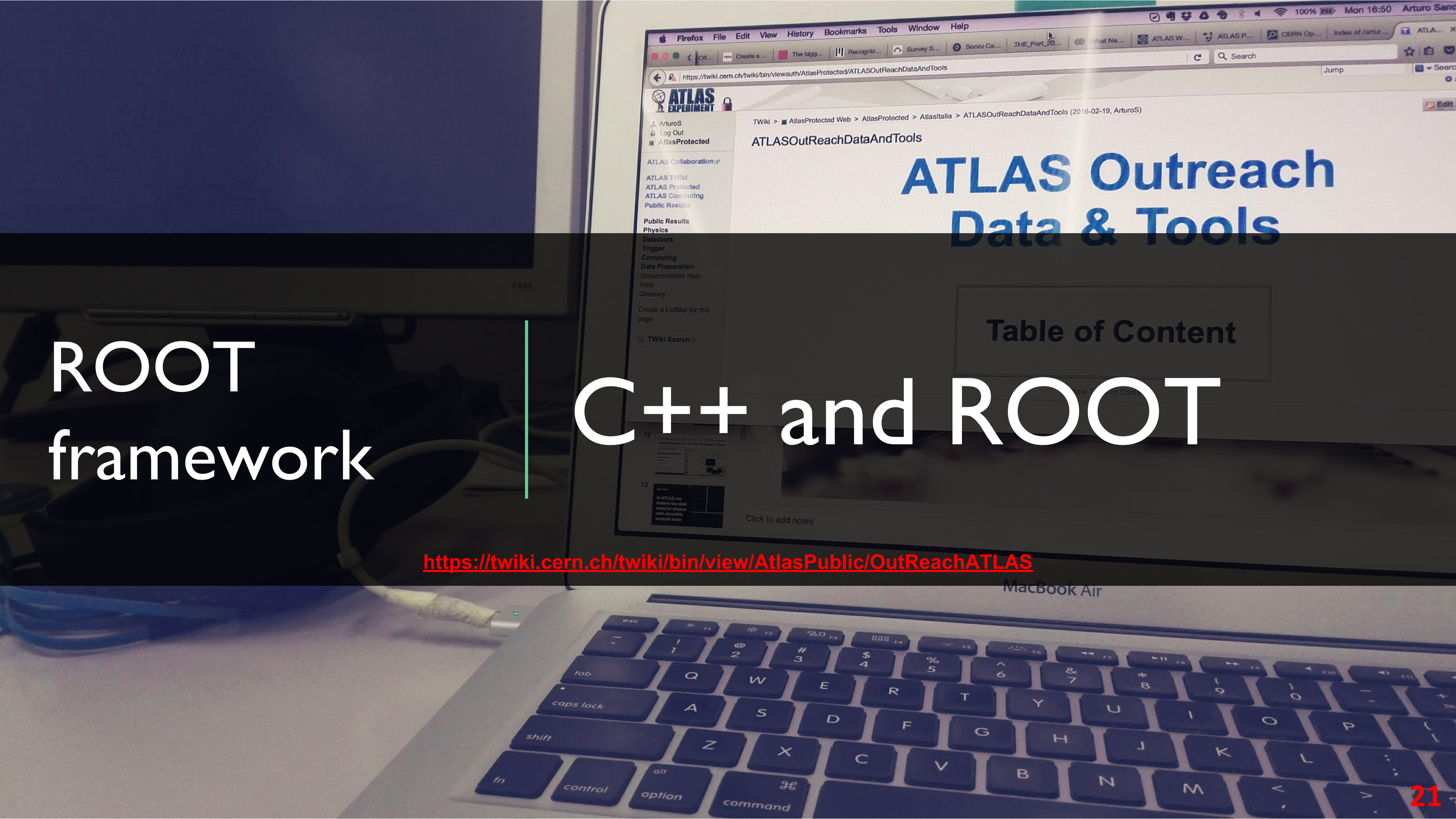
! 2. You join a huge community, $O(10^4)$ users + a very supportive team of core developers

? *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.



ROOT framework

C++ and ROOT

https://twiki.cern.ch/twiki/bin/view/AtlasPublic/OutReachATLAS

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

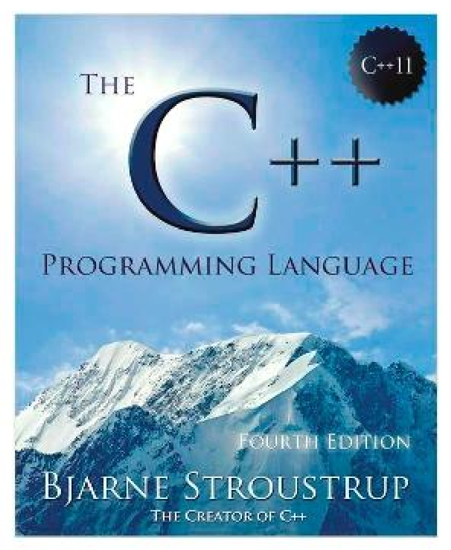
Allows object orientation

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

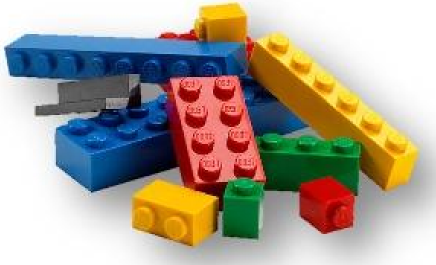
ROOT as a Calculator

ROOT as Function Plotter

Plotting Measurements

Histograms

Interactive ROOT Section



The ROOT Prompt

C++ is a compiled language

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

ROOT provides a C++ **interpreter**

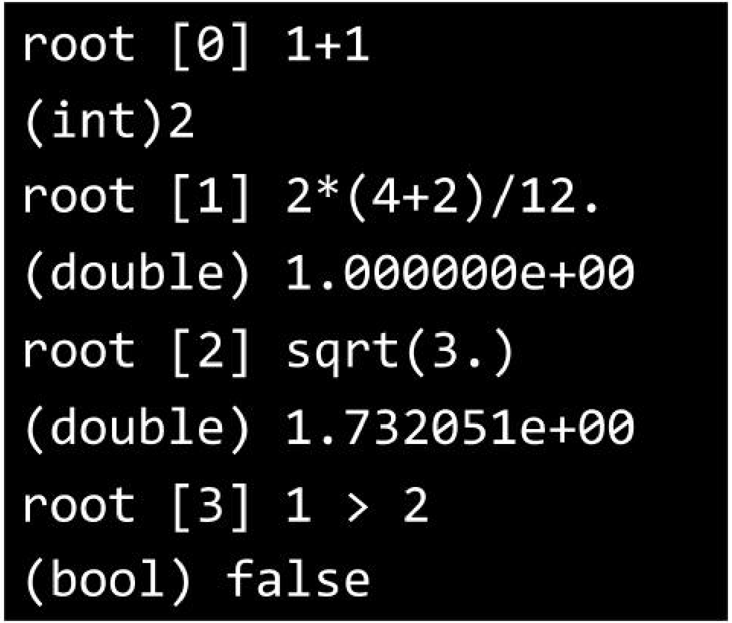
- Interactive C++, w/o the need of a compiler, like Python, Ruby, Haskell …
- Allows reflection (inspect at runtime layout of classes)
- Can be booted with the command:

```
root
```

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

ROOT As a Calculator

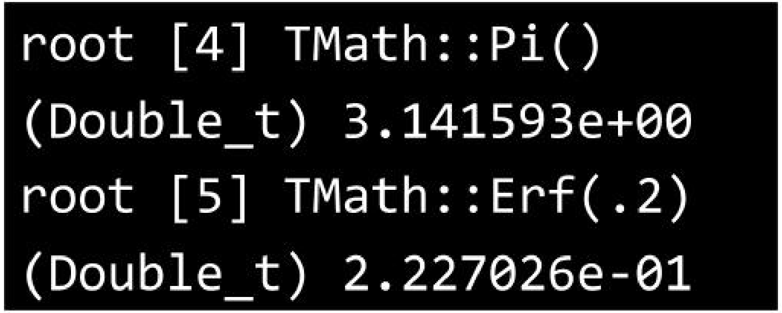
ROOT interactive prompt can be used as an advanced calculator!



```
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
```

Try it!

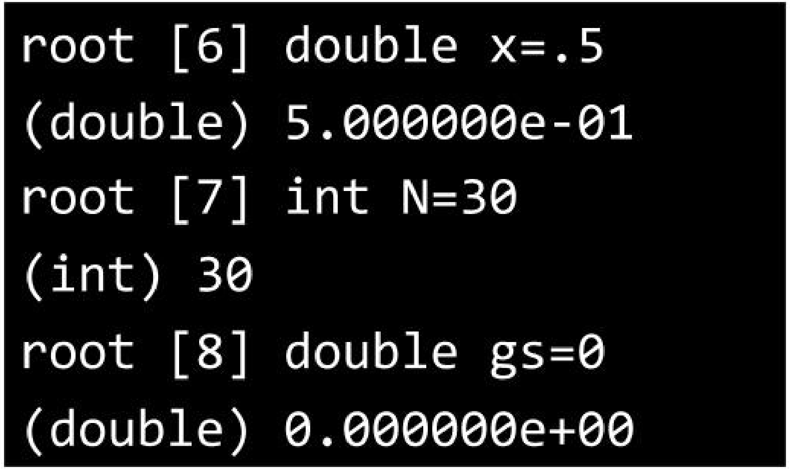
ROOT allows not only to type in **C++ statements**, but also advanced **mathematical functions**, which live in the TMath namespace.



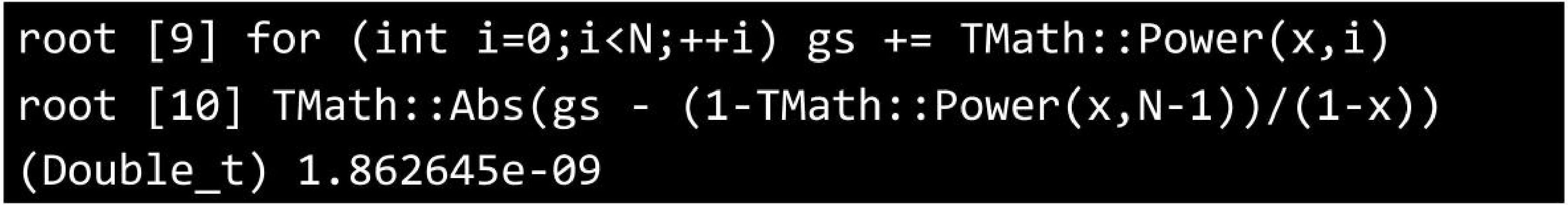
```
root [4] TMath::Pi()
(Double_t) 3.141593e+00
root [5] TMath::Erf(.2)
(Double_t) 2.227026e-01
```

ROOT As a Calculator++

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



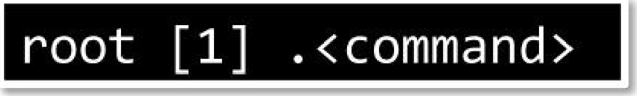
```
root [6] double x=.5
(double) 5.000000e-01
root [7] int N=30
(int) 30
root [8] double gs=0
(double) 0.000000e+00
```



```
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>
```

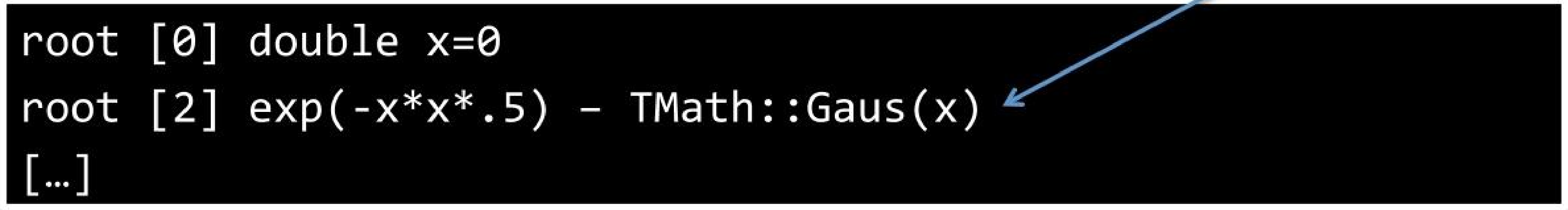
For example:

- 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.

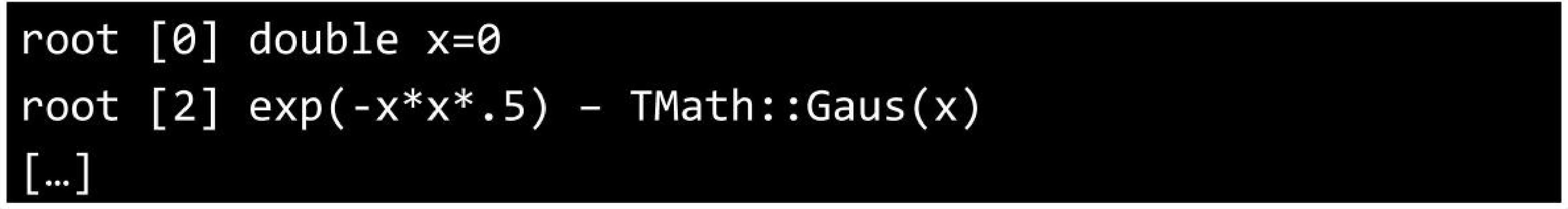
For one number



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

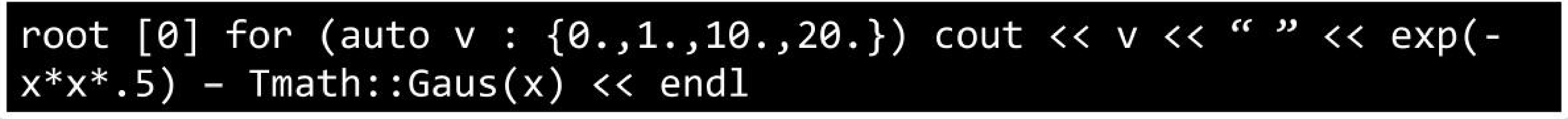
Exercise Solution

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)
[…]
```

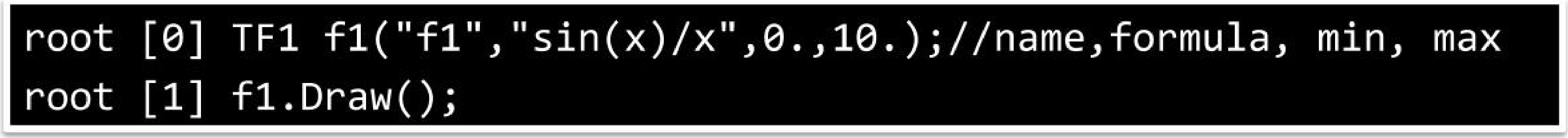
Many possible ways of solving this! E.g:



```
root [0] for (auto v : {0.,1.,10.,20.}) cout << v << “ ” << exp(-
x*x*.5) – Tmath::Gaus(x) << endl
```

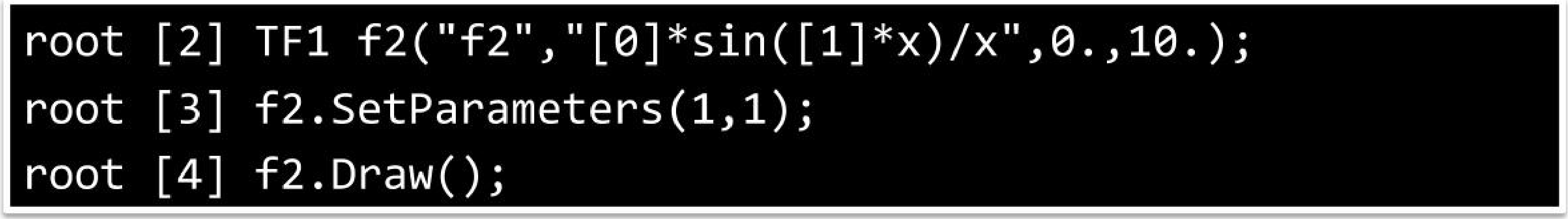
ROOT As a Function Plotter

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();
```

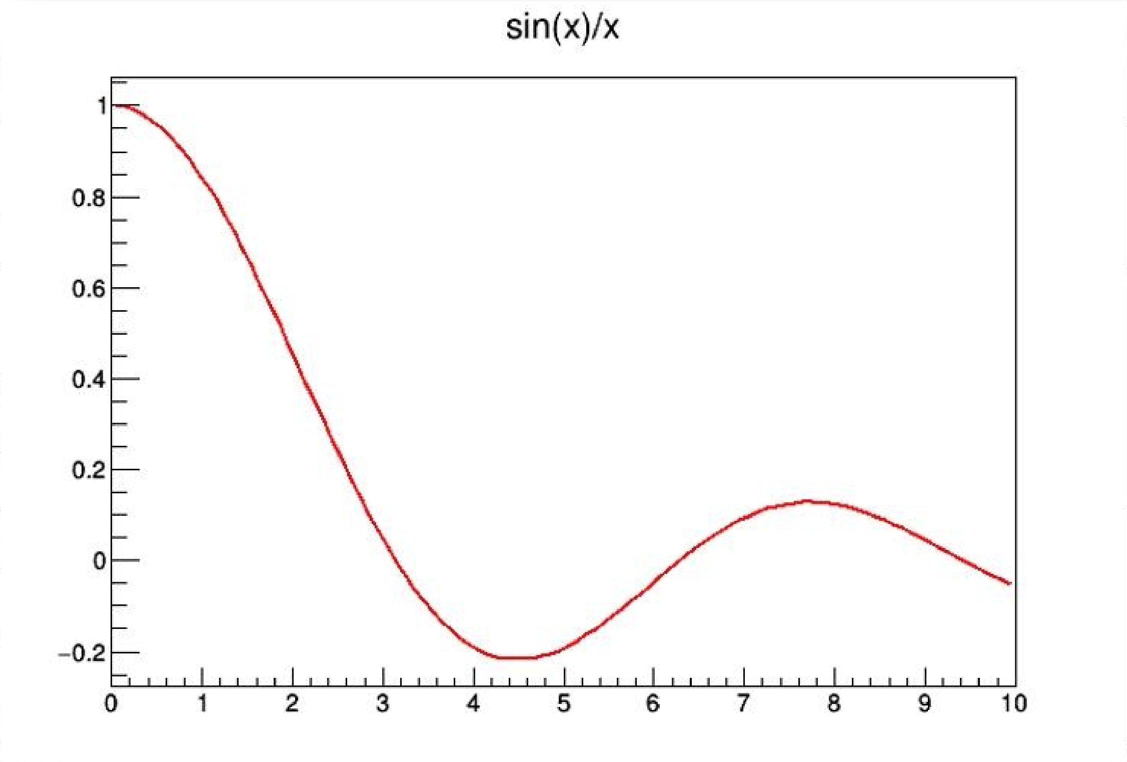
ROOT As a Function Plotter

The class TF1 re

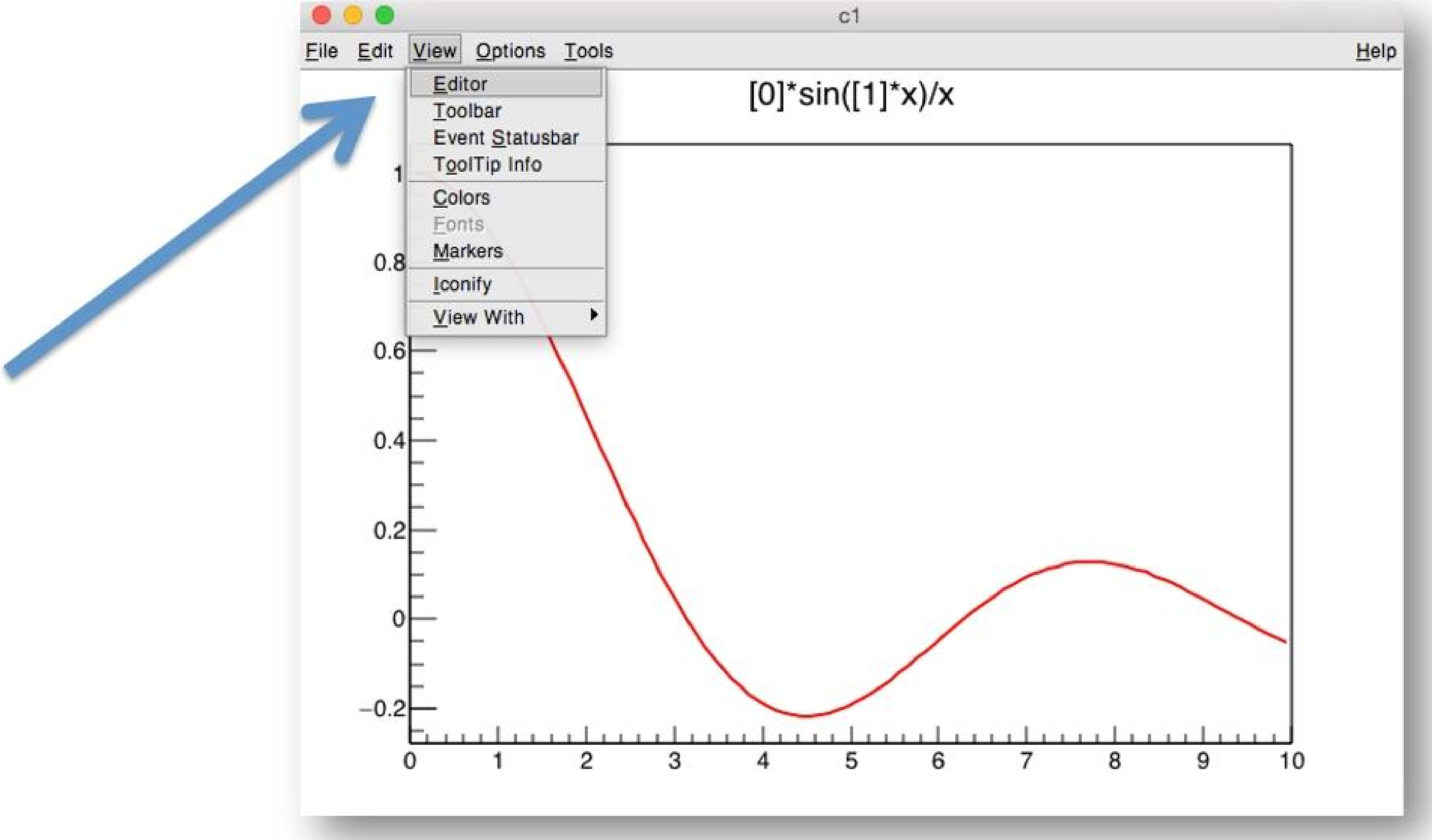
```
root [0] TF1                                    min, max
root [1] f1.D
```

An extended ve
with parameters

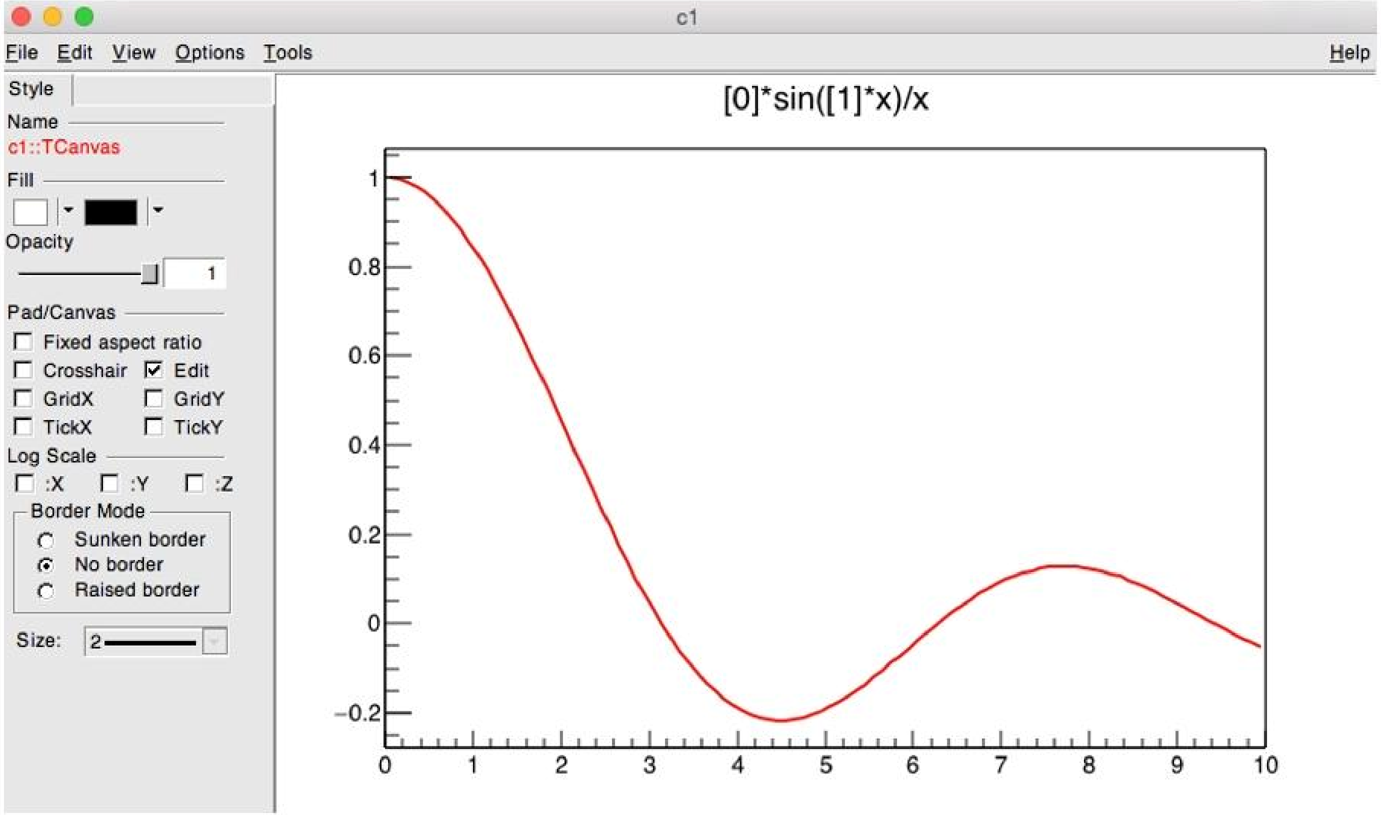
```
root [2] TF1
root [3] f2.
root [4] f2.
```



Exercise: Interaction With The Plot



Exercise: Interaction With The Plot



Name

c1::TCanvas

Fill

Opacity

1

Pad/Canvas

☐ Fixed aspect ratio

☐ Crosshair ☑ Edit

☐ GridX ☐ GridY

☐ TickX ☐ TickY

Log Scale

☐ :X ☐ :Y ☐ :Z

Border Mode

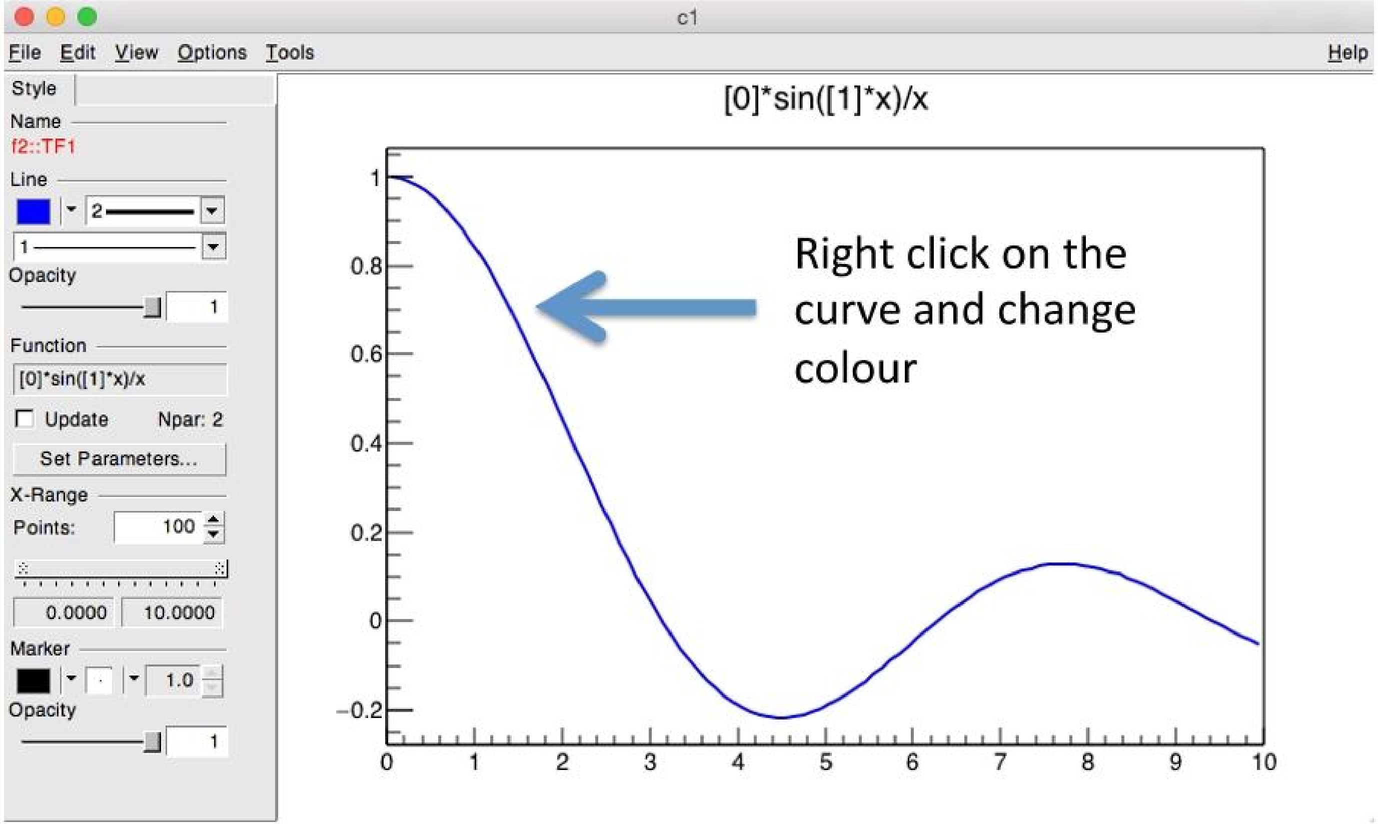
○ Sunken border

● No border

○ Raised border

Size: 2

Exercise: Interaction With The Plot



Name
f2::TF1

Line
2
1

Opacity 1

Function
[0]*sin([1]*x)/x

□ Update Npar: 2

Set Parameters...

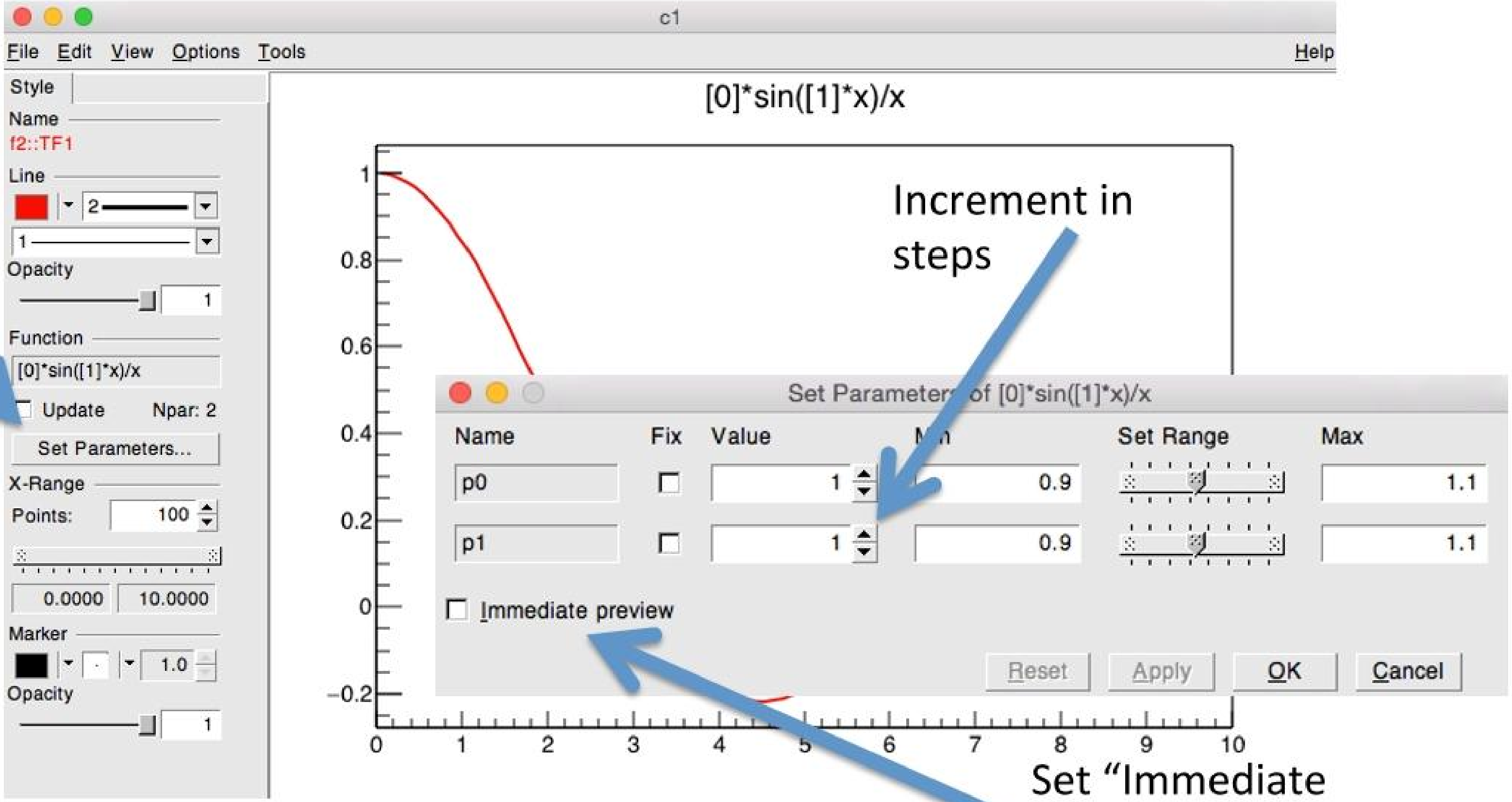
X-Range
Points: 100
0.0000 10.0000

Marker
1.0

Opacity 1

Right click on the curve and change colour

Exercise: Interaction With The Plot



Name: f2::TF1

Line: 2, 1

Opacity: 1

Function: [0]*sin([1]*x)/x

□ Update Npar: 2

X-Range Points: 100

0.0000 10.0000

Marker: 1.0

Opacity: 1

Set Parameters of [0]*sin([1]*x)/x

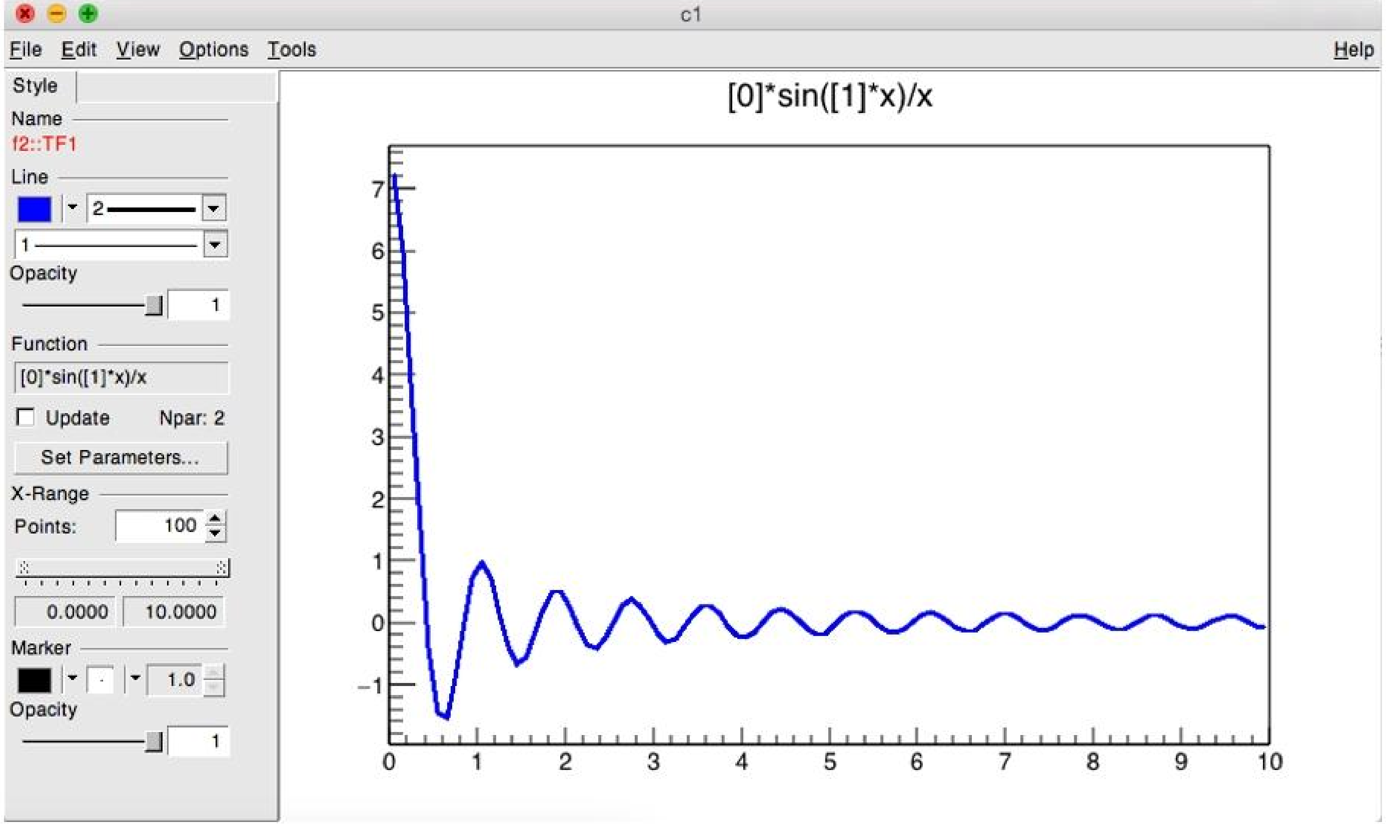
Name	Fix	Value	Min	Set Range	Max
p0	□	1	0.9		1.1
p1	□	1	0.9		1.1

□ Immediate preview

Increment in steps

Set “Immediate preview”

Exercise: Interaction With The Plot



Name

f2::TF1

Line

2

1

Opacity 1

Function

[0]*sin([1]*x)/x

□ Update Npar: 2

Set Parameters...

X-Range

Points: 100

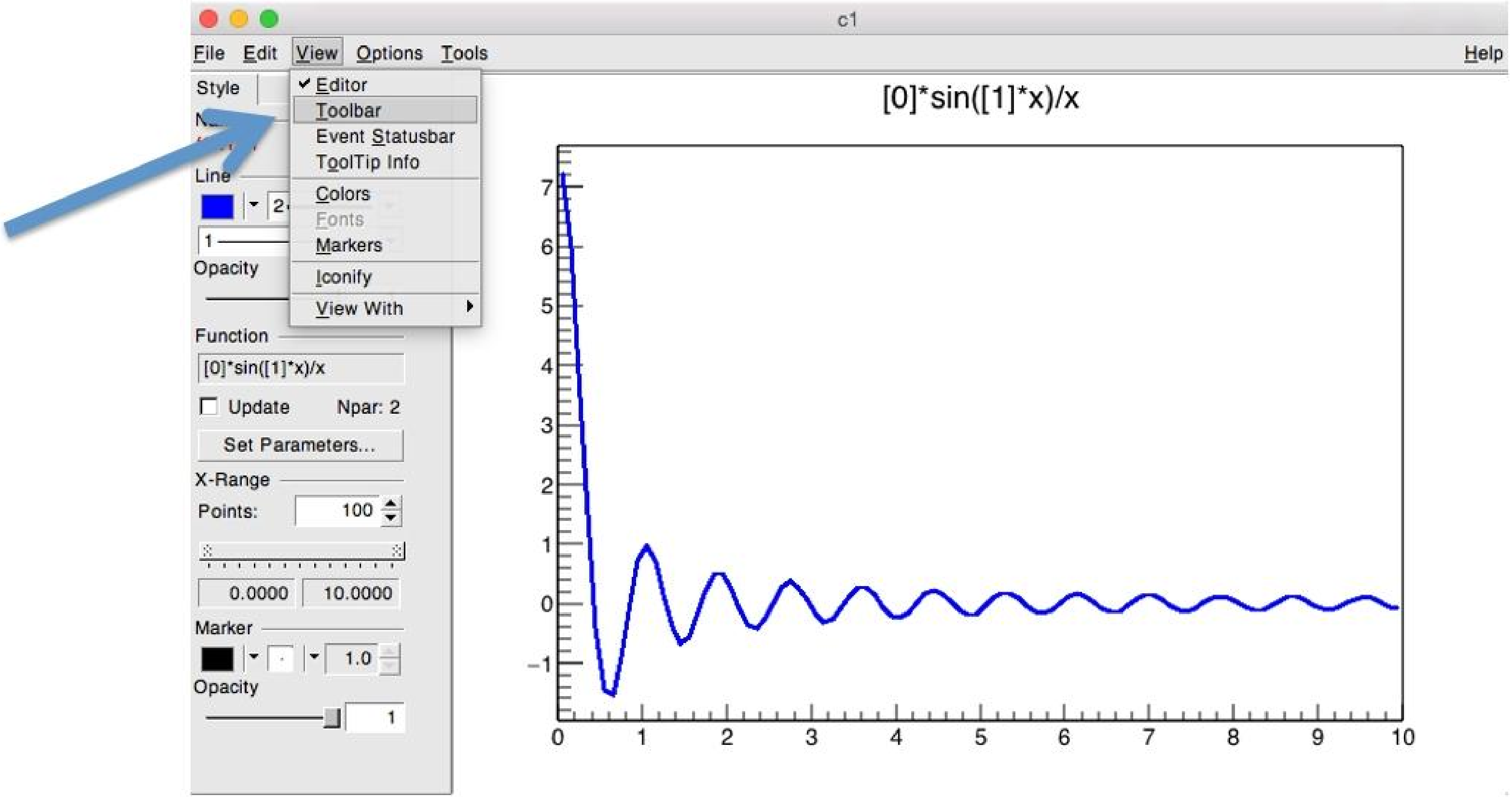
0.0000 10.0000

Marker

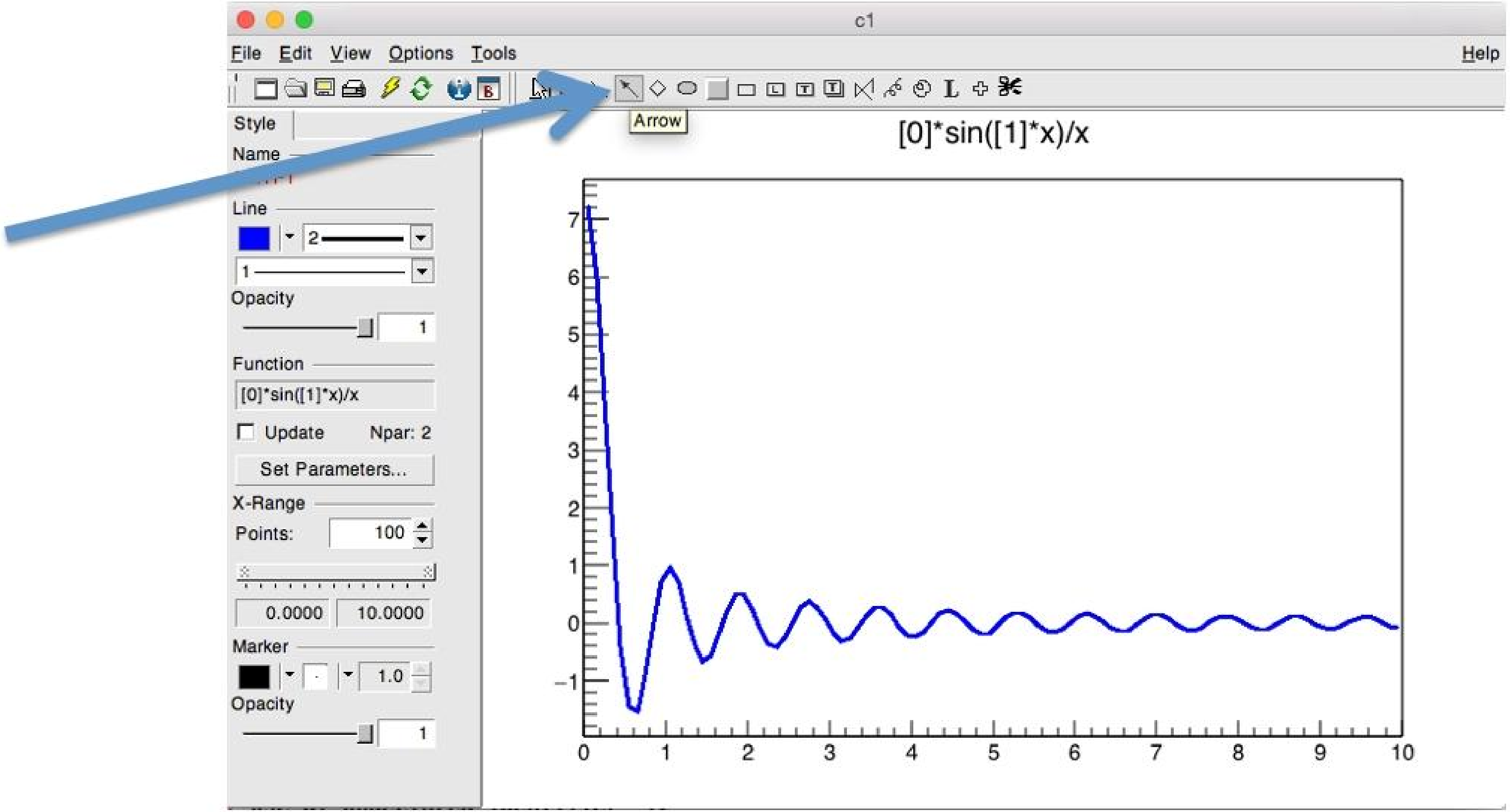
1.0

Opacity 1

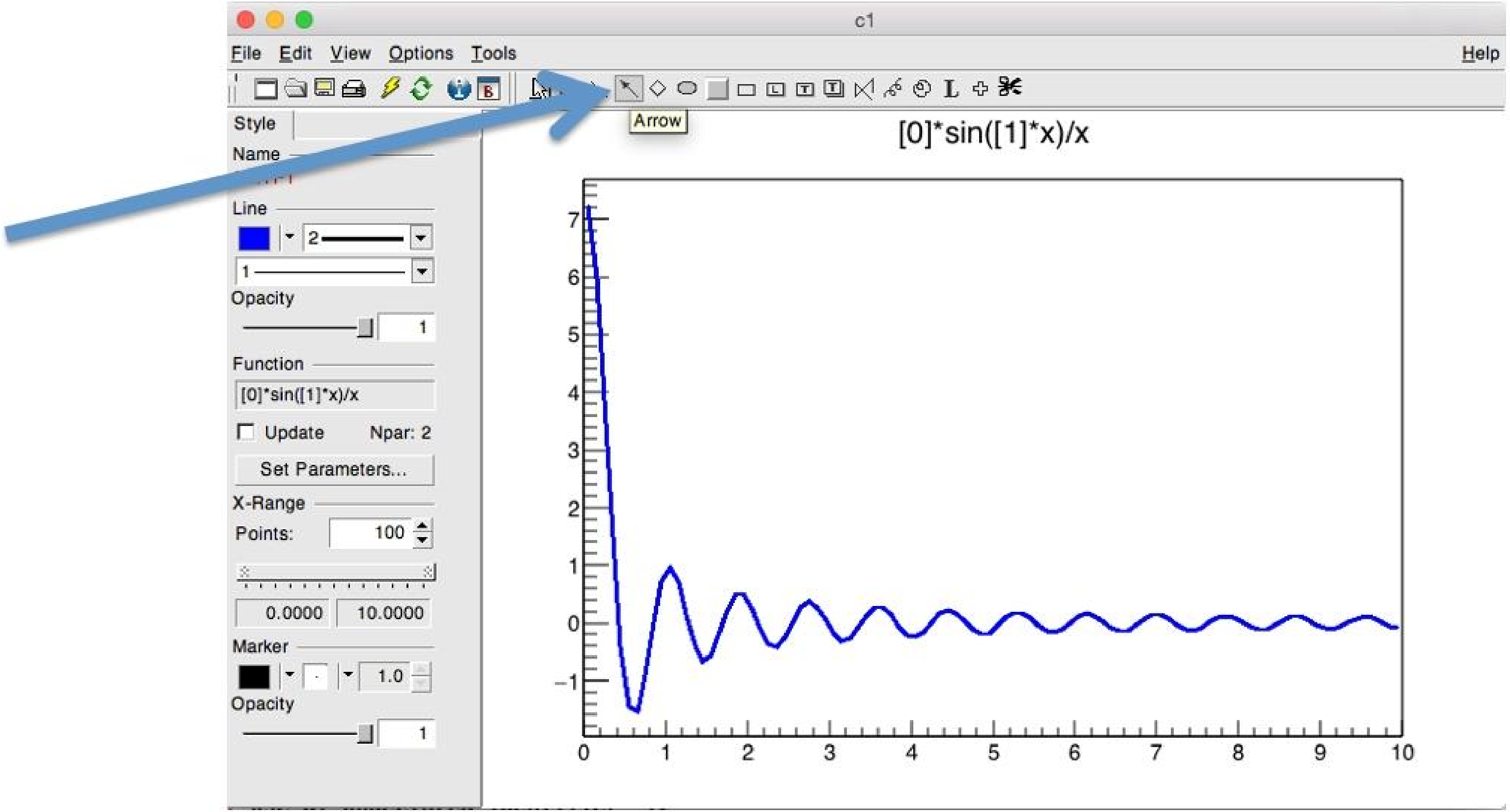
Exercise: Interaction With The Plot



Exercise: Interaction With The Plot



Exercise: Interaction With The Plot



Style

Name

Line

2

1

Opacity 1

Function

[0]*sin([1]*x)/x

□ Update Npar: 2

X-Range

Points: 100

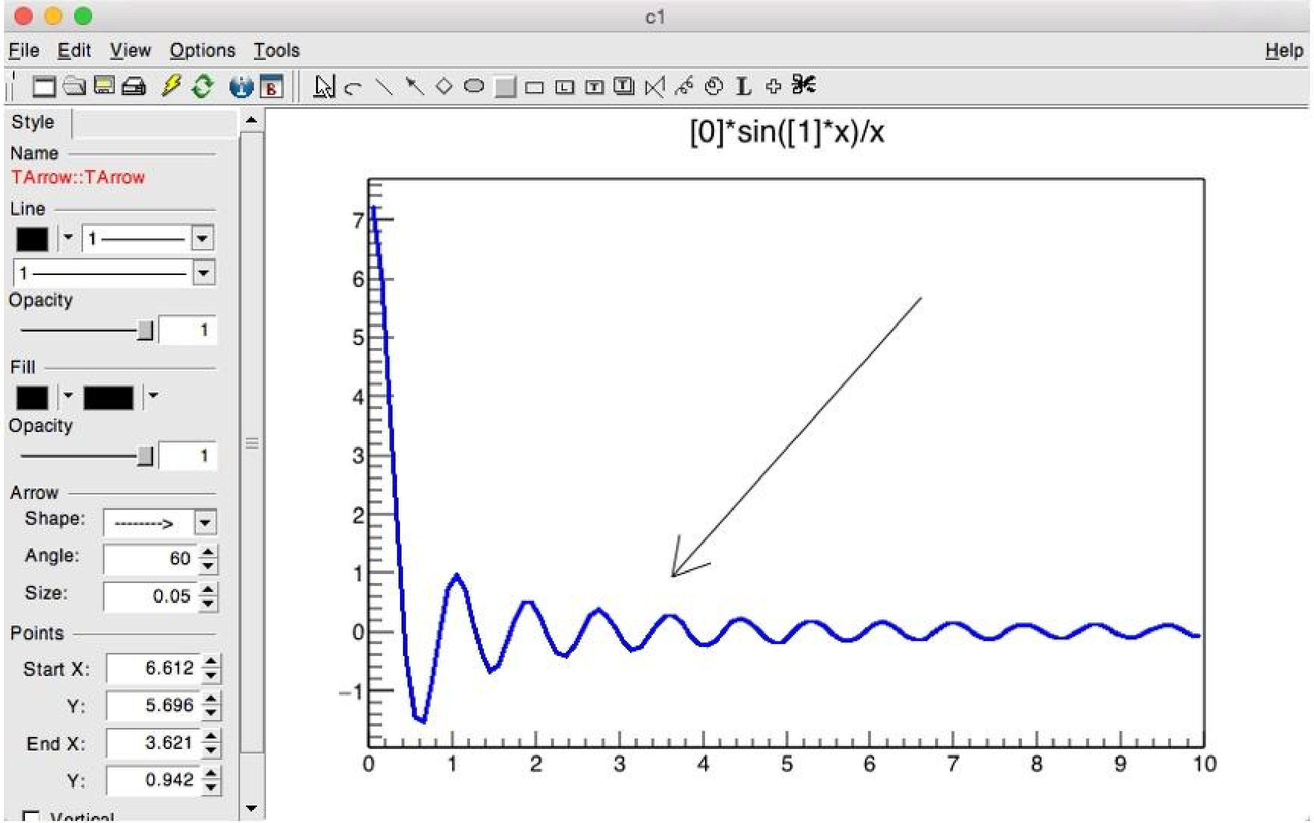
0.0000 10.0000

Marker

1.0

Opacity 1

Exercise: Interaction With The Plot



Style

Name

TArrow::TArrow

Line

1

1

Opacity 1

Fill

Opacity 1

Arrow

Shape: -------->

Angle: 60

Size: 0.05

Points

Start X: 6.612

Y: 5.696

End X: 3.621

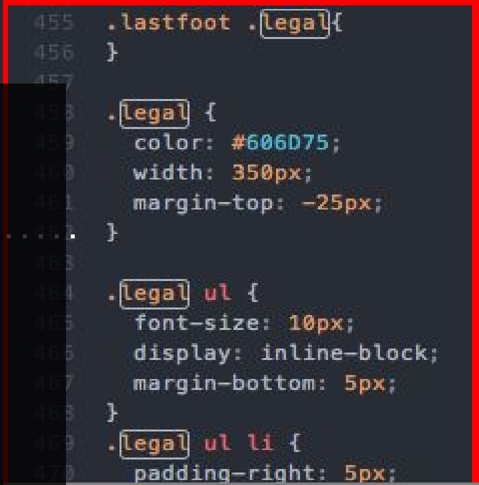
Y: 0.942

□ Vertical

And now?

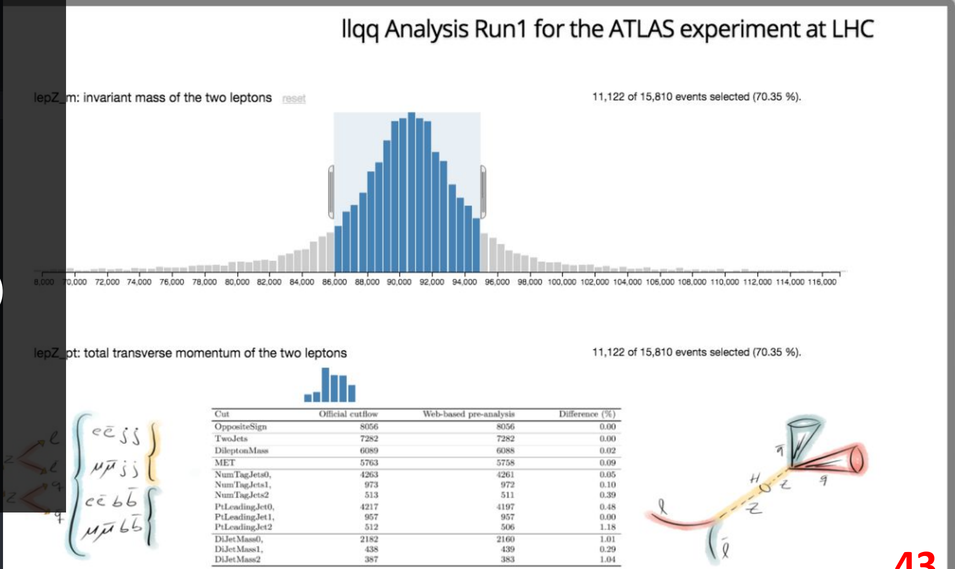
We will need to get a working station with ROOT in order to start to create...

```
   // Declaration of leaf types
   Int_t           Event_;
   UInt_t          Event_fUniqueID[kMaxEvent];   //[Event_]
   UInt_t          Event_fBits[kMaxEvent];   //[Event_]
   Long64_t        Event_Number[kMaxEvent];   //[Event_]
   Int_t           Event_Nparticles[kMaxEvent];   //[Event_]
   Int_t           Event_ProcessID[kMaxEvent];   //[Event_]
   Double_t        Event_Weight[kMaxEvent];   //[Event_]
   Double_t        Event_ScalePDF[kMaxEvent];   //[Event_]
   Double_t        Event_CouplingQED[kMaxEvent];   //[Event_]
   Double_t        Event_CouplingQCD[kMaxEvent];   //[Event_]
   Int_t           Event_size;
   Int_t           Rwgt_;
   UInt_t          Rwgt_fUniqueID[kMaxRwgt];   //[Rwgt_]
   UInt_t          Rwgt_fBits[kMaxRwgt];   //[Rwgt_]
   Double_t        Rwgt_Weight[kMaxRwgt];   //[Rwgt_]
   Int_t           Rwgt_size;
   Int_t           Particle_;
   UInt_t          Particle_fUniqueID[kMaxParticle];   //[Particle_]
   UInt_t          Particle_fBits[kMaxParticle];   //[Particle_]
   Int_t           Particle_PID[kMaxParticle];   //[Particle_]
   Int_t           Particle_Status[kMaxParticle];   //[Particle_]
   Int_t           Particle_Mother1[kMaxParticle];   //[Particle_]
   Int_t           Particle_Mother2[kMaxParticle];   //[Particle_]
   Int_t           Particle_ColorLine1[kMaxParticle];   //[Particle_]
   Int_t           Particle_ColorLine2[kMaxParticle];   //[Particle_]
   Double_t        Particle_Px[kMaxParticle];   //[Particle_]
   Double_t        Particle_Py[kMaxParticle];   //[Particle_]
   Double_t        Particle_Pz[kMaxParticle];   //[Particle_]
   Double_t        Particle_E[kMaxParticle];   //[Particle_]
   Double_t        Particle_M[kMaxParticle];   //[Particle_]
   Double_t        Particle_PT[kMaxParticle];   //[Particle_]
   Double_t        Particle_Eta[kMaxParticle];   //[Particle_]
   Double_t        Particle_Phi[kMaxParticle];   //[Particle_]
   Double_t        Particle_Rapidity[kMaxParticle];   //[Particle_]
   Double_t        Particle_LifeTime[kMaxParticle];   //[Particle_]
   Double_t        Particle_Spin[kMaxParticle];   //[Particle_]
   Int_t           Particle_size;
```



```
.lastfoot .legal{
}

.legal {
  color: #606D75;
  width: 350px;
  margin-top: -25px;
}

.legal ul {
  font-size: 10px;
  display: inline-block;
  margin-bottom: 5px;
}
.legal ul li {
  padding-right: 5px;
```

```
.lastfoot .legal{
}

.legal {
  color: #606D75;
  width: 350px;
  margin-top: -25px;
}

.legal ul {
  font-size: 10px;
  display: inline-block;
  margin-bottom: 5px;
}
.legal ul li {
  padding-right: 5px;
```



llqq Analysis Run1 for the ATLAS experiment at LHC

lepZ_m: invariant mass of the two leptons reset

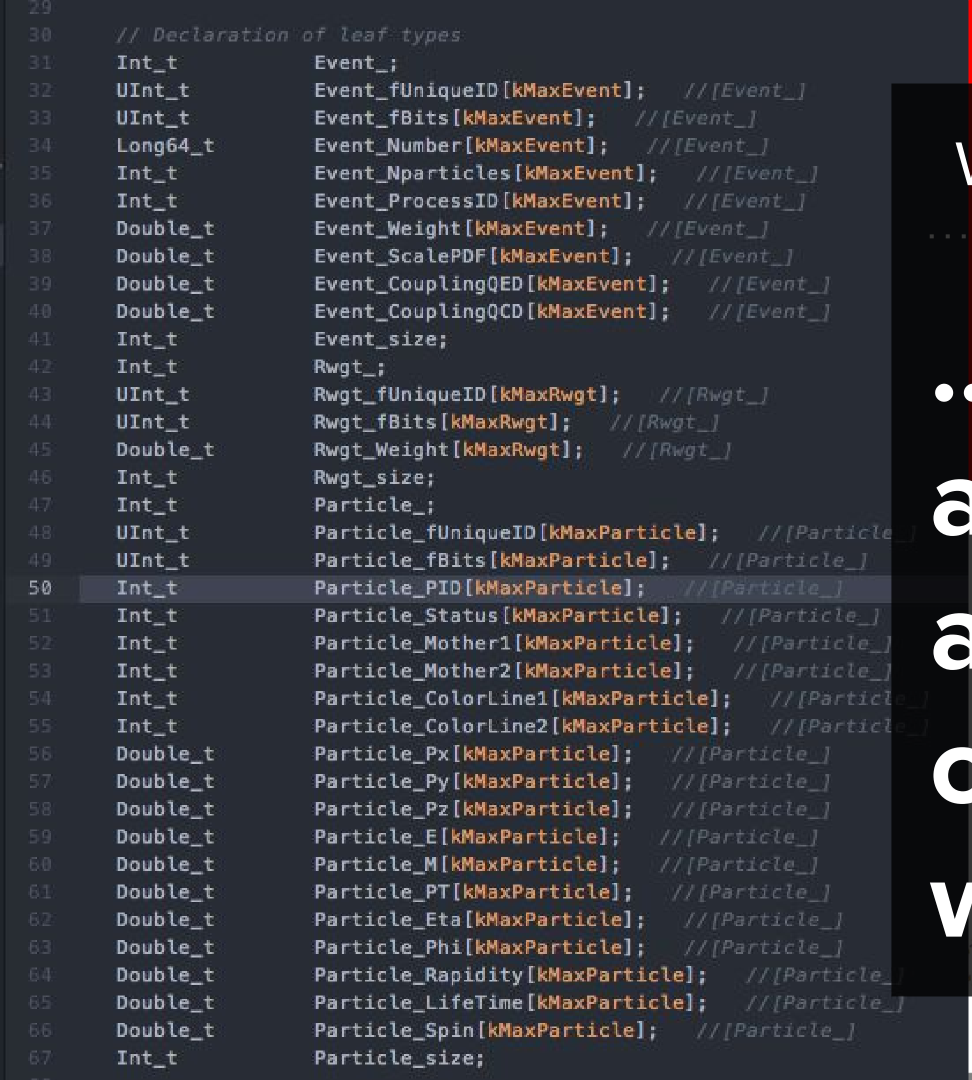
11,122 of 15,810 events selected (70.35 %).

lepZ_pt: total transverse momentum of the two leptons

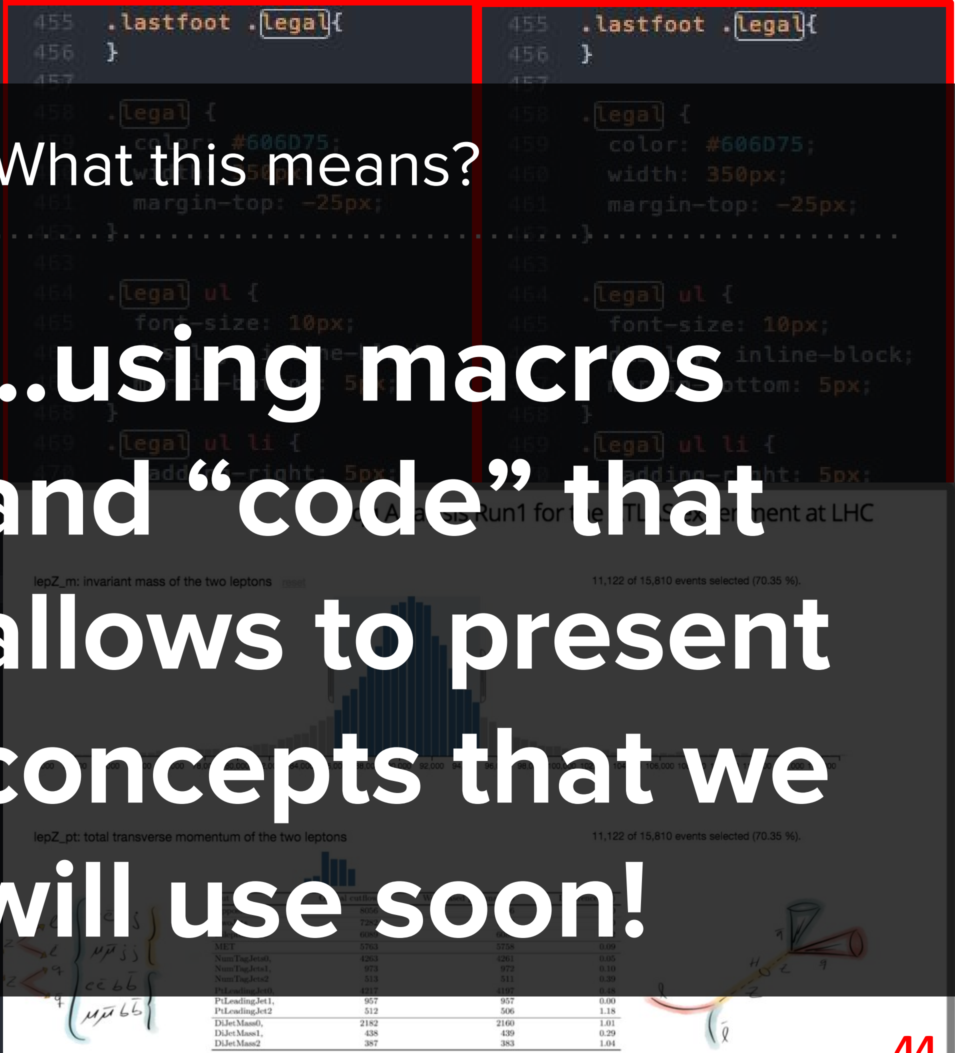
11,122 of 15,810 events selected (70.35 %).

Cut	Official cutflow	Web-based pre-analysis	Difference (%)
OppositeSign	8056	8056	0.00
TwoJets	7282	7282	0.00
DileptonMass	6089	6088	0.02
MET	5763	5758	0.09
NumTagJets0,	4263	4261	0.05
NumTagJets1,	973	972	0.10
NumTagJets2	513	511	0.39
PtLeadingJet0,	4217	4197	0.48
PtLeadingJet1,	957	957	0.00
PtLeadingJet2	512	506	1.18
DiJetMass0,	2182	2160	1.01
DiJetMass1,	438	439	0.29
DiJetMass2	387	383	1.04

http://atlasui03.na.infn.it:8088/~sanchez/ATLAS/outreach/presentations/October_19_2015/OutreachSummary.html



```
   // Declaration of leaf types
   Int_t           Event_;
   UInt_t          Event_fUniqueID[kMaxEvent];   //[Event_]
   UInt_t          Event_fBits[kMaxEvent];   //[Event_]
   Long64_t        Event_Number[kMaxEvent];   //[Event_]
   Int_t           Event_Nparticles[kMaxEvent];   //[Event_]
   Int_t           Event_ProcessID[kMaxEvent];   //[Event_]
   Double_t        Event_Weight[kMaxEvent];   //[Event_]
   Double_t        Event_ScalePDF[kMaxEvent];   //[Event_]
   Double_t        Event_CouplingQED[kMaxEvent];   //[Event_]
   Double_t        Event_CouplingQCD[kMaxEvent];   //[Event_]
   Int_t           Event_size;
   Int_t           Rwgt_;
   UInt_t          Rwgt_fUniqueID[kMaxRwgt];   //[Rwgt_]
   UInt_t          Rwgt_fBits[kMaxRwgt];   //[Rwgt_]
   Double_t        Rwgt_Weight[kMaxRwgt];   //[Rwgt_]
   Int_t           Rwgt_size;
   Int_t           Particle_;
   UInt_t          Particle_fUniqueID[kMaxParticle];   //[Particle_]
   UInt_t          Particle_fBits[kMaxParticle];   //[Particle_]
   Int_t           Particle_PID[kMaxParticle];   //[Particle_]
   Int_t           Particle_Status[kMaxParticle];   //[Particle_]
   Int_t           Particle_Mother1[kMaxParticle];   //[Particle_]
   Int_t           Particle_Mother2[kMaxParticle];   //[Particle_]
   Int_t           Particle_ColorLine1[kMaxParticle];   //[Particle_]
   Int_t           Particle_ColorLine2[kMaxParticle];   //[Particle_]
   Double_t        Particle_Px[kMaxParticle];   //[Particle_]
   Double_t        Particle_Py[kMaxParticle];   //[Particle_]
   Double_t        Particle_Pz[kMaxParticle];   //[Particle_]
   Double_t        Particle_E[kMaxParticle];   //[Particle_]
   Double_t        Particle_M[kMaxParticle];   //[Particle_]
   Double_t        Particle_PT[kMaxParticle];   //[Particle_]
   Double_t        Particle_Eta[kMaxParticle];   //[Particle_]
   Double_t        Particle_Phi[kMaxParticle];   //[Particle_]
   Double_t        Particle_Rapidity[kMaxParticle];   //[Particle_]
   Double_t        Particle_LifeTime[kMaxParticle];   //[Particle_]
   Double_t        Particle_Spin[kMaxParticle];   //[Particle_]
   Int_t           Particle_size;
```


.lastfoot .legal{
}
.legal {
color: #606D75;
width: 350px;
margin-top: -25px;
.legal ul {
font-size: 10px;
.legal ul li {
Analysis Run1 for the ATLAS experiment at LHC
lepZ_m: invariant mass of the two leptons
11,122 of 15,810 events selected (70.35 %).
lepZ_pt: total transverse momentum of the two leptons
11,122 of 15,810 events selected (70.35 %).

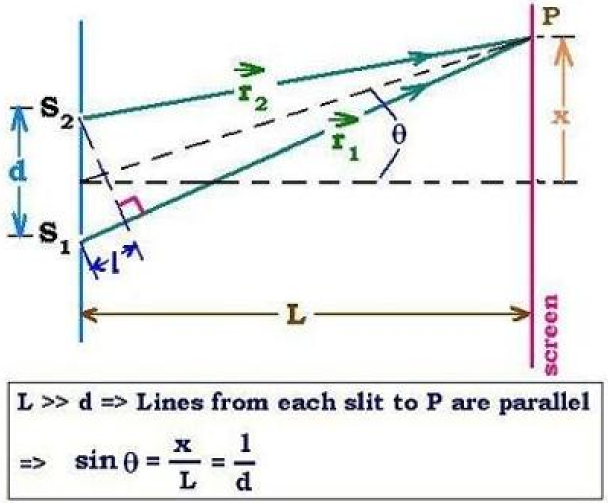
What this means?

…using macros and “code” that allows to present concepts that we will use soon!

http://atlasui03.na.infn.it:8088/~sanchez/ATLAS/outreach/presentations/October_19_2015/OutreachSummary.html

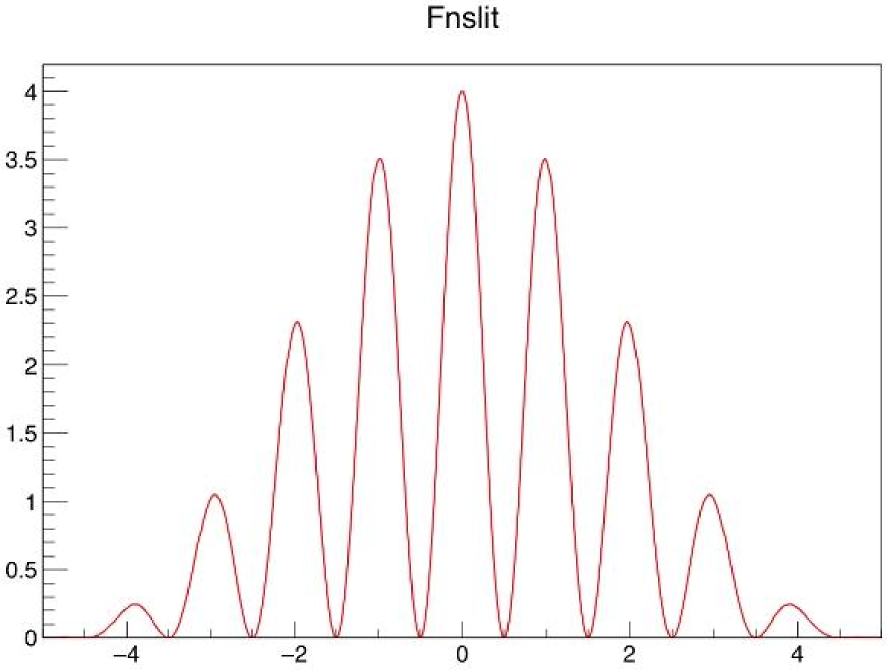
ROOT As a Function Plotter

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 >> d => Lines from each slit to P are parallel

=> $\sin\theta = \frac{x}{L} = \frac{l}{d}$



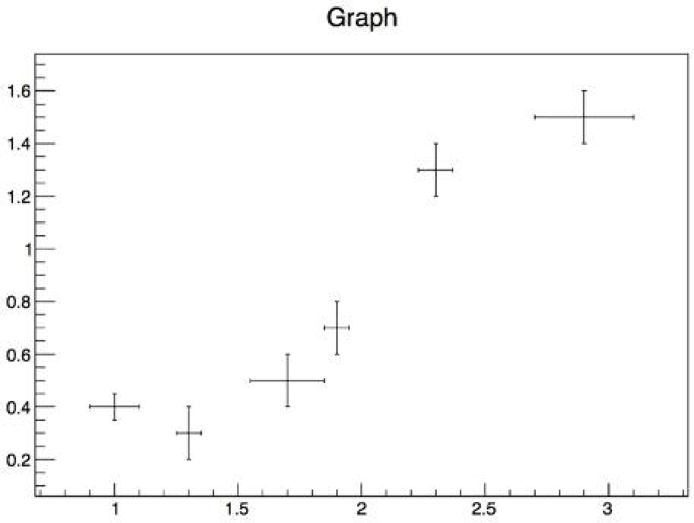
slits.C

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

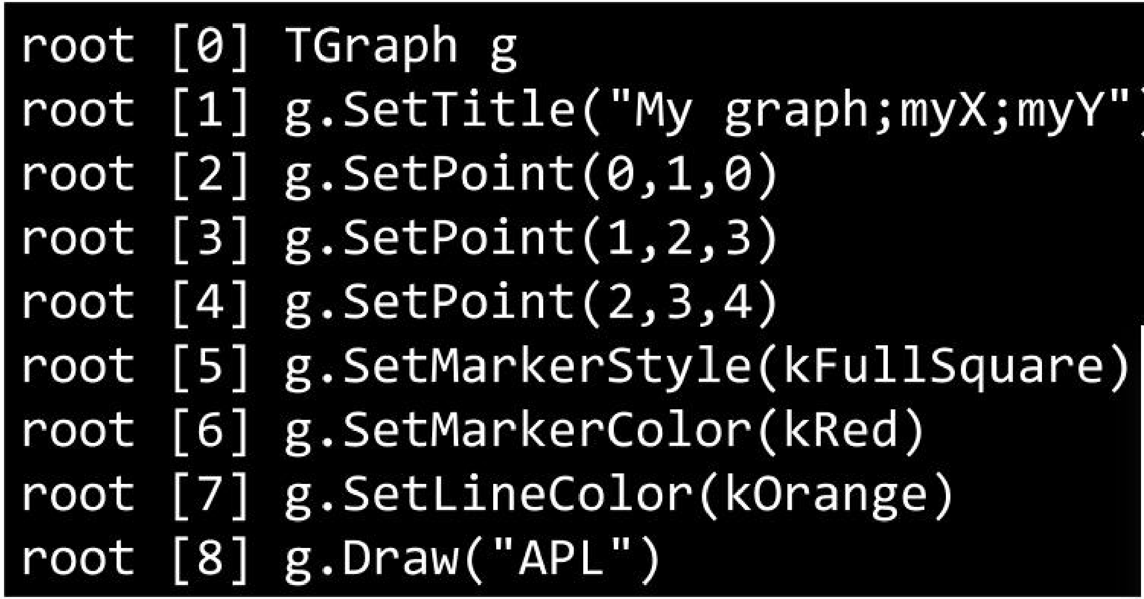


Extempore Exercise

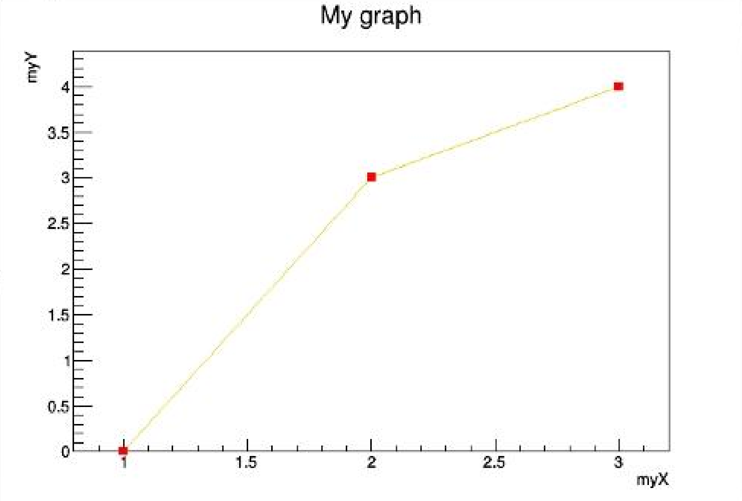
- Create a graph (TGraph)
- Set its title to “My graph”, its X axis title to “myX” and Y axis title to “myY”
- Fill it with three points: (1,0), (2,3), (3,4)
- Set a red full square marker
- Draw a orange line between points

Let’s solve this together at the whiteboard!

Exercise Solution



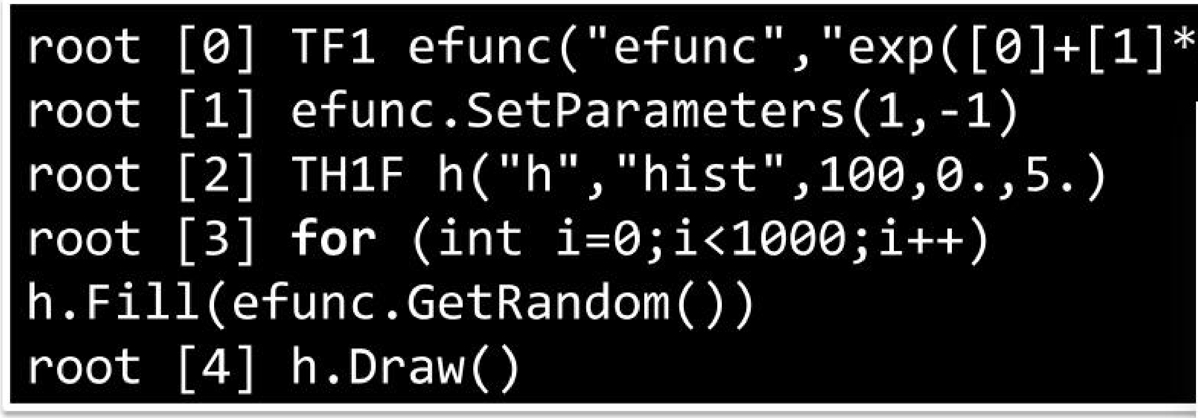
```
root [0] TGraph g
root [1] g.SetTitle("My graph;myX;myY")
root [2] g.SetPoint(0,1,0)
root [3] g.SetPoint(1,2,3)
root [4] g.SetPoint(2,3,4)
root [5] g.SetMarkerStyle(kFullSquare)
root [6] g.SetMarkerColor(kRed)
root [7] g.SetLineColor(kOrange)
root [8] g.Draw("APL")
```



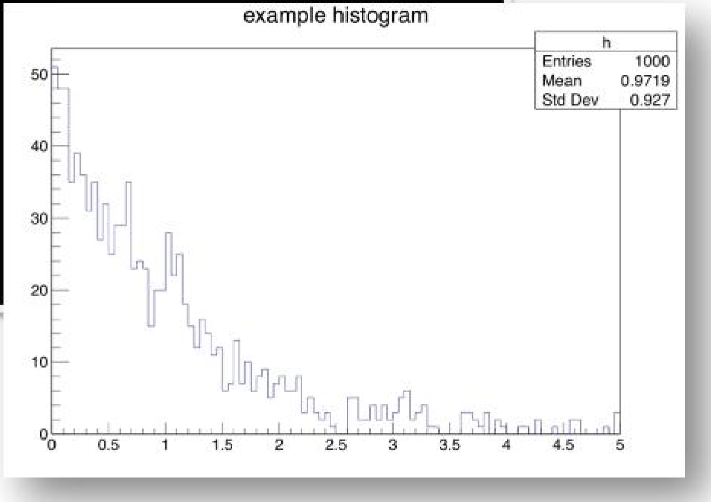
Histograms

- In ROOT, the TH* classes represent histograms
- TH1* are monodimensional, TH2* are bidimensional …
- The final letter describes the type stored in each bin:

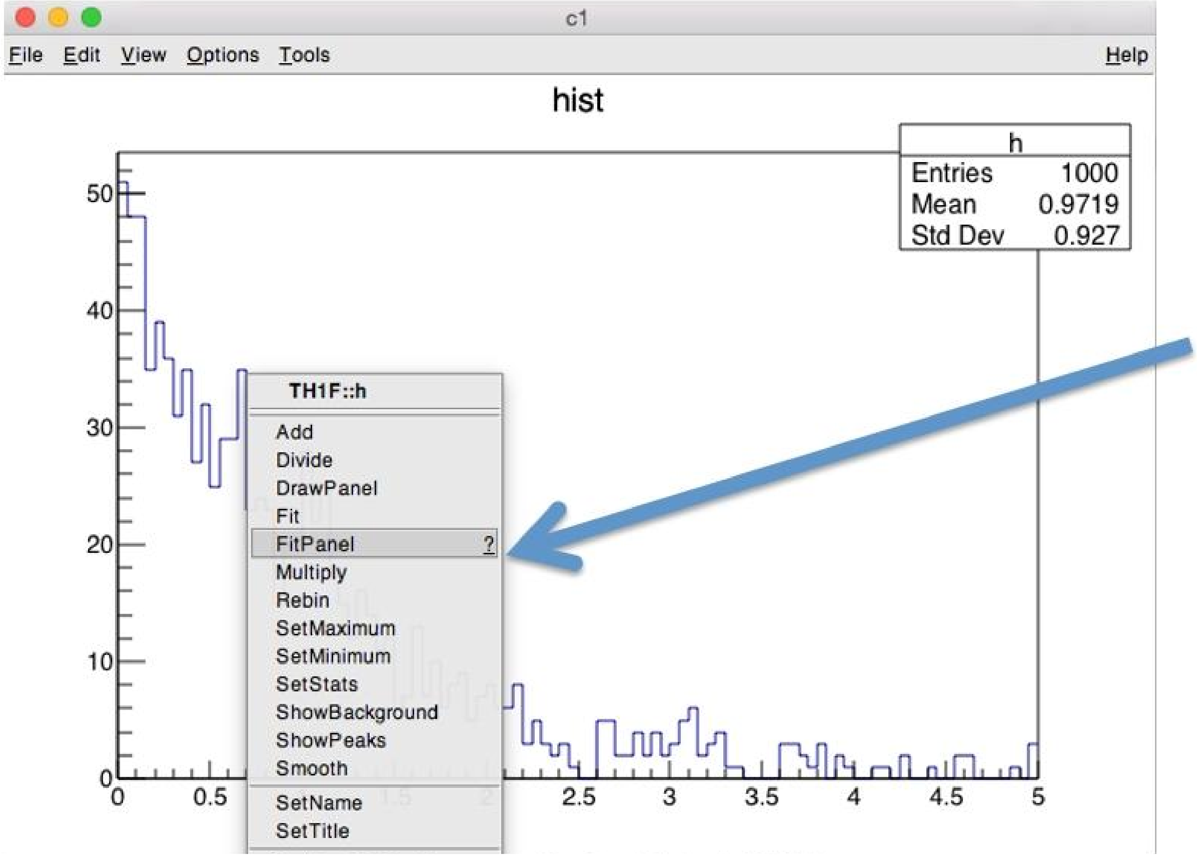
  A double in TH1D, a float in TH1F …



```
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()
```

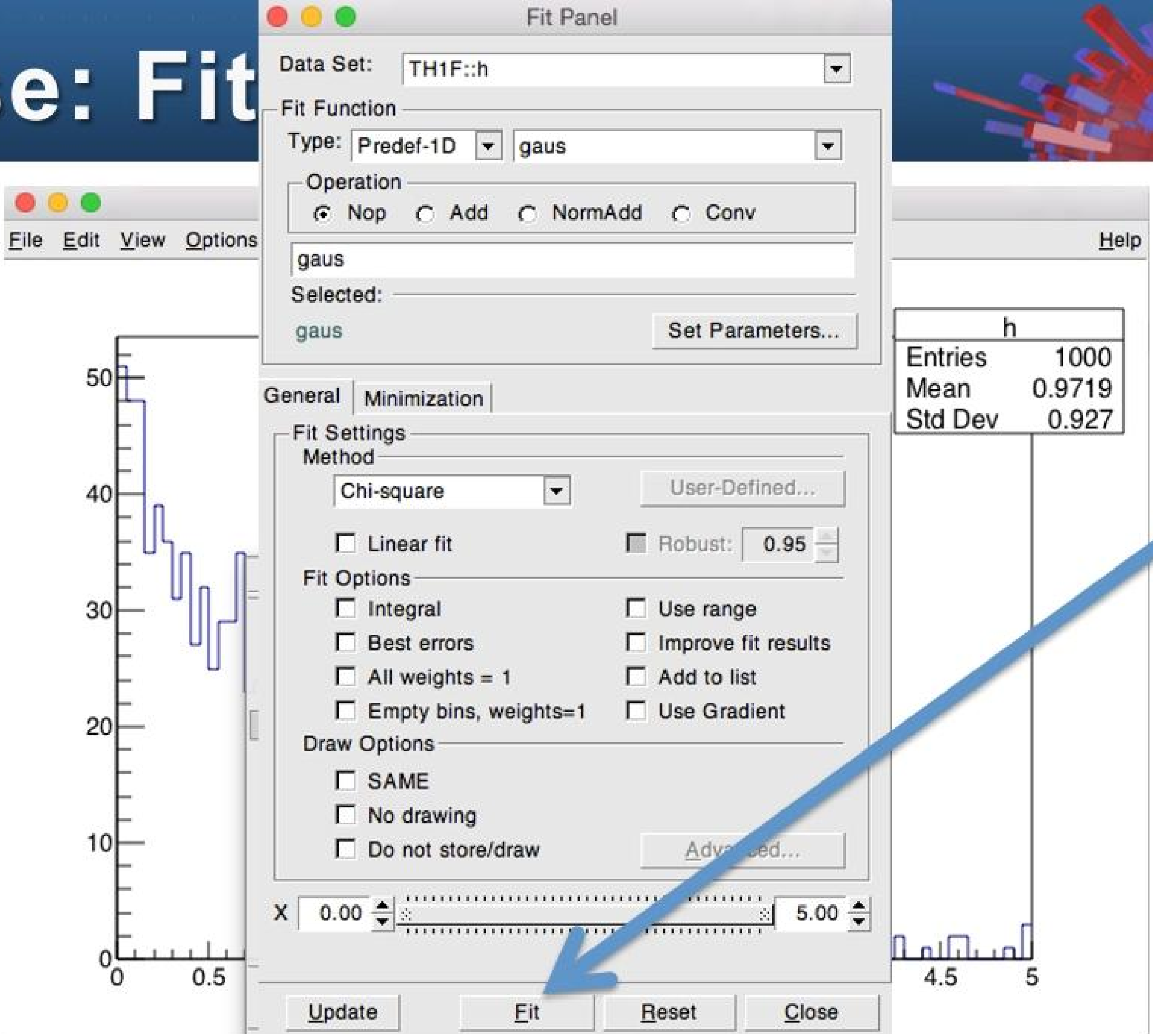


All together: Fitpanel



Click on the histogram “line”

Exercise: Fit



Fit Panel

Data Set: TH1F::h

Fit Function

Type: Predef-1D | gaus

Operation

● Nop ○ Add ○ NormAdd ○ Conv

gaus

Selected:

gaus

Set Parameters...

General | Minimization

Fit Settings

Method

Chi-square

User-Defined...

□ Linear fit

□ Robust: 0.95

Fit Options

□ Integral

□ Use range

□ Best errors

□ Improve fit results

□ All weights = 1

□ Add to list

□ Empty bins, weights=1

□ Use Gradient

Draw Options

□ SAME

□ No drawing

□ Do not store/draw

Advanced...

X 0.00 5.00

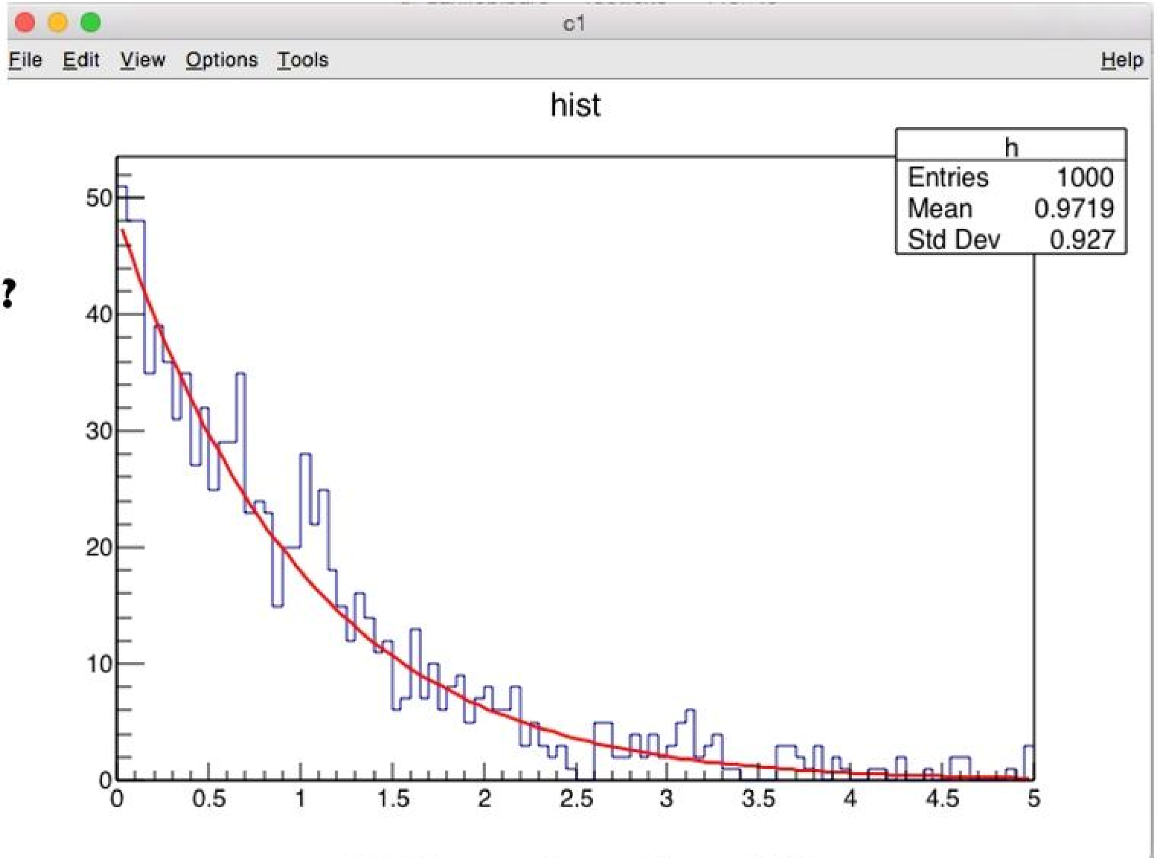
Update | Fit | Reset | Close

h	
Entries	1000
Mean	0.9719
Std Dev	0.927

Fit!

Exercise: Fitpanel

Is this a good fit? Why?

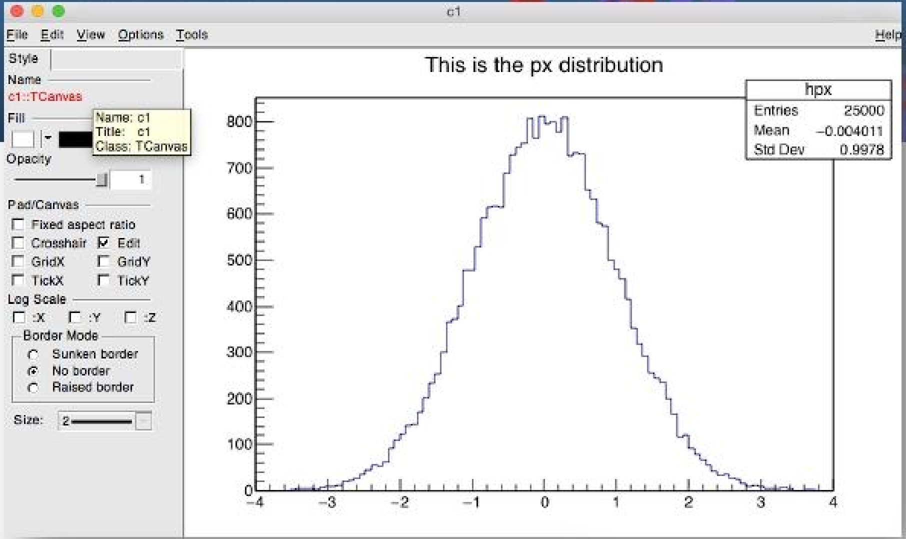


Interactive ROOT

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 !!**



c1
File Edit View Options Tools Help
Style
This is the px distribution
800
700
600
500
400
300
200
100
0
−4 −3 −2 −1 0 1 2 3 4

Name
c1::TCanvas

Name: c1
Title: c1
Class: TCanvas

Fill

Opacity 1

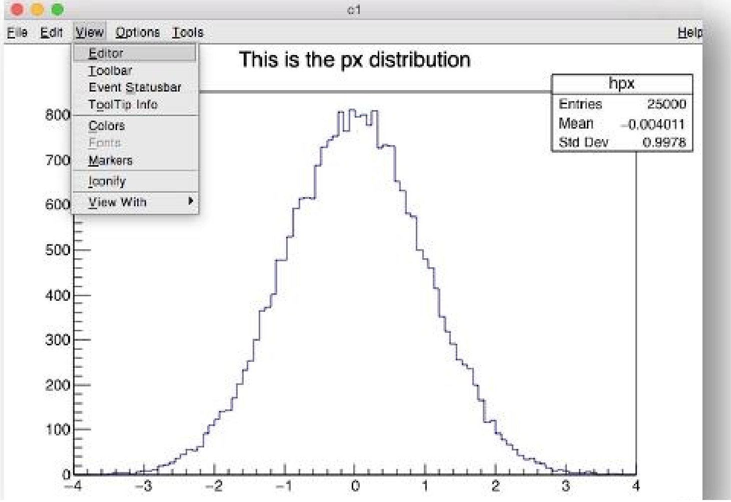
Pad/Canvas
☐ Fixed aspect ratio
☐ Crosshair ☑ Edit
☐ GridX ☐ GridY
☐ TickX ☐ TickY

Log Scale
☐ :X ☐ :Y ☐ :Z

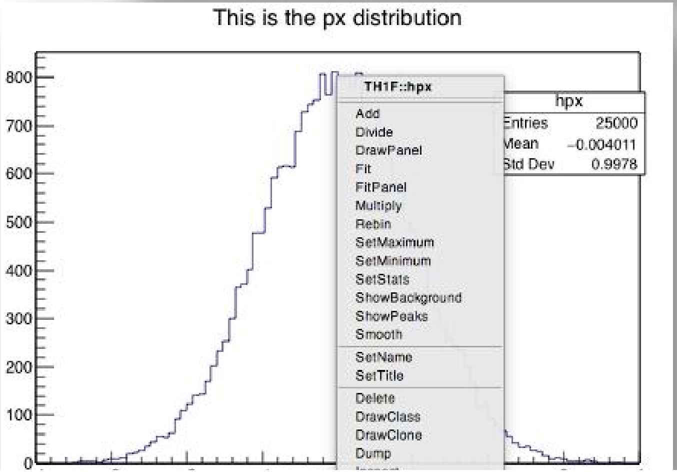
Border Mode
○ Sunken border
● No border
○ Raised border

Size: 2

```
hpx
Entries     25000
Mean    -0.004011
Std Dev    0.9978
```



```
hpx
Entries     25000
Mean    -0.004011
Std Dev    0.9978
```



```
hpx
Entries     25000
Mean    -0.004011
Std Dev    0.9978
```

ROOT Macros

General Remarks

A more complete example

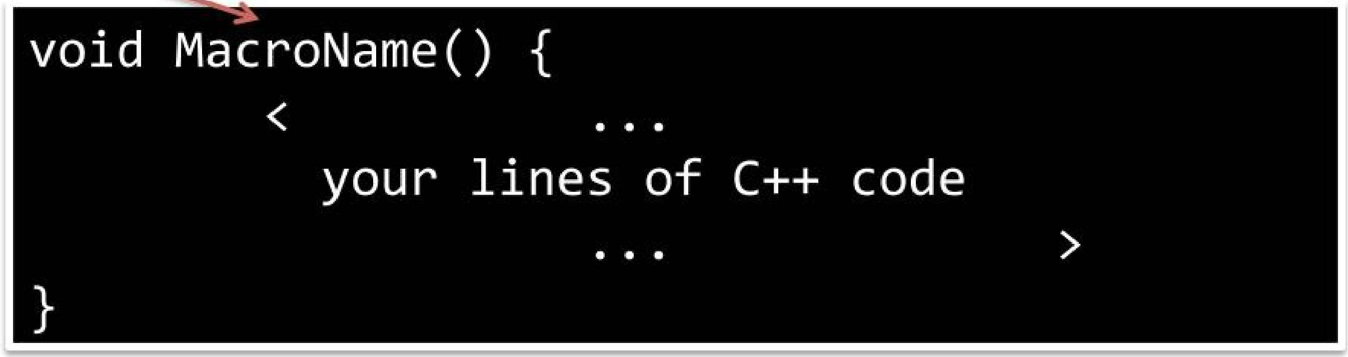
Summary of Visual effects

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:

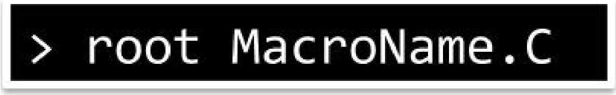
Function, no main,
same name as the file



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

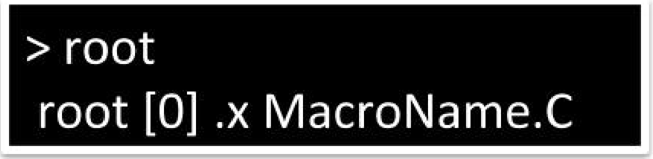
Running a Macro

The macro is executed at the system prompt by typing:



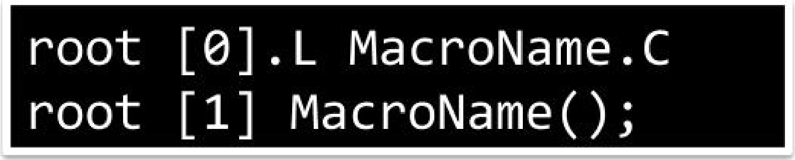
```
> root MacroName.C
```

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

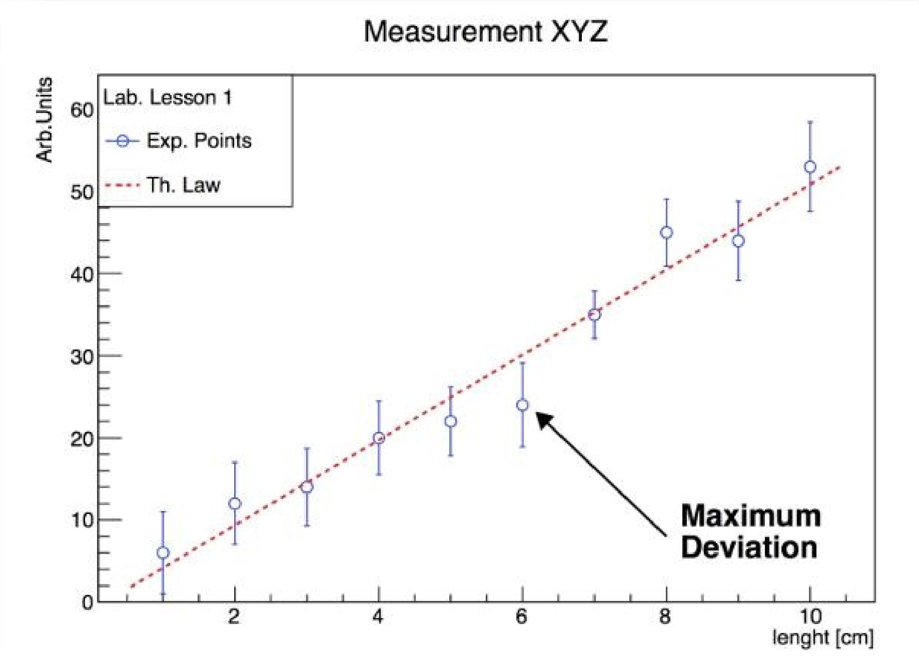
The example in section 3.2 of the ROOT primer, is a typical task in data analysis, a macro that constructs a graph with errors, fits a (linear) model to it and saves it as an image.

Let's inspect it together.

A More Complex Example

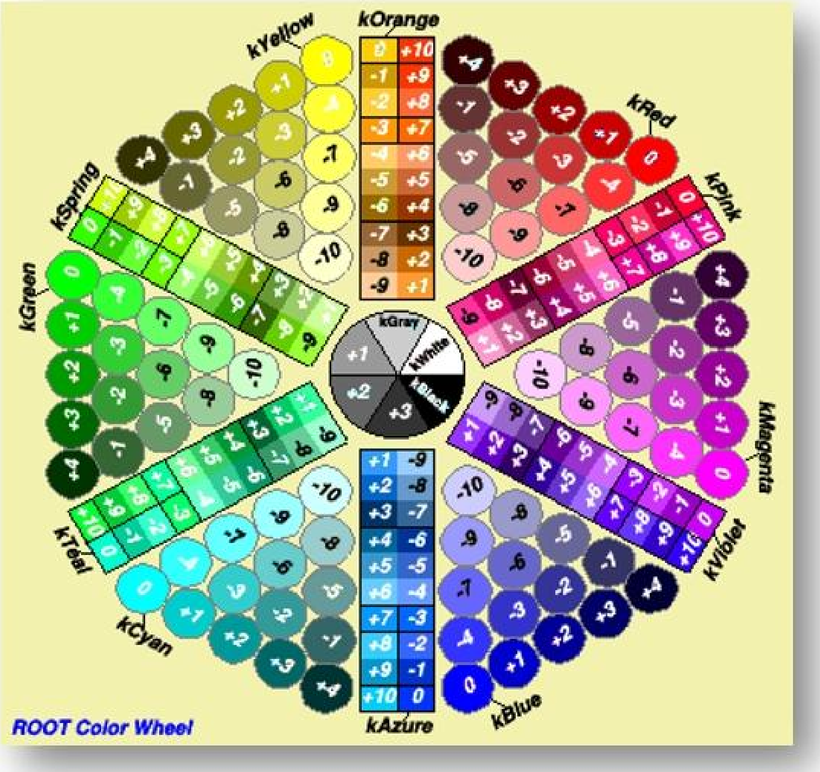
And Run it!

```
> root macro1.C
```



Macro1.C

TColorWheel



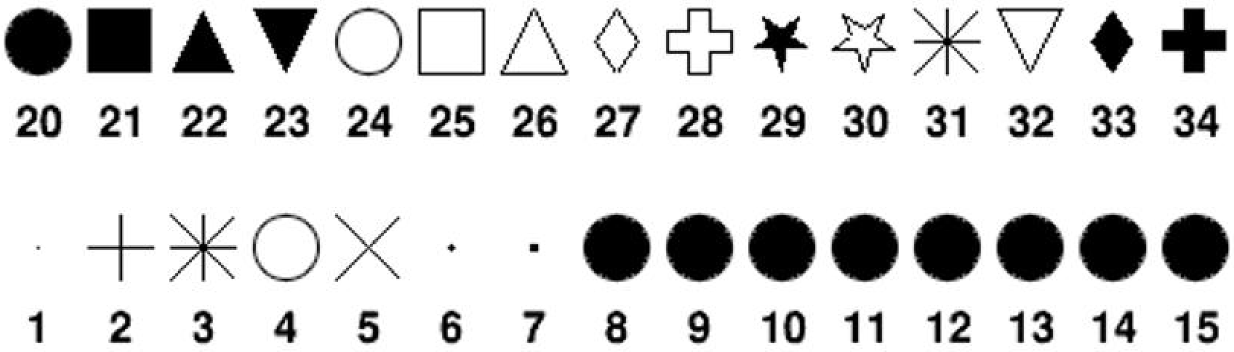
Summary of Visual Effects

- **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 "#".

The Family of Markers

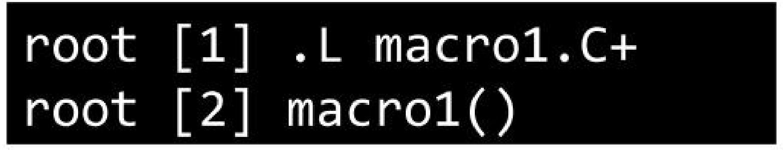


```
kDot=1, kPlus, kStar, kCircle=4, kMultiply=5,
kFullDotSmall=6, kFullDotMedium=7, kFullDotLarge=8,
kFullCircle=20, kFullSquare=21, kFullTriangleUp=22,
kFullTriangleDown=23, kOpenCircle=24, kOpenSquare=25,
kOpenTriangleUp=26, kOpenDiamond=27, kOpenCross=28,
kFullStar=29, kOpenStar=30, kOpenTriangleDown=32,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

Also available through more friendly names ☺

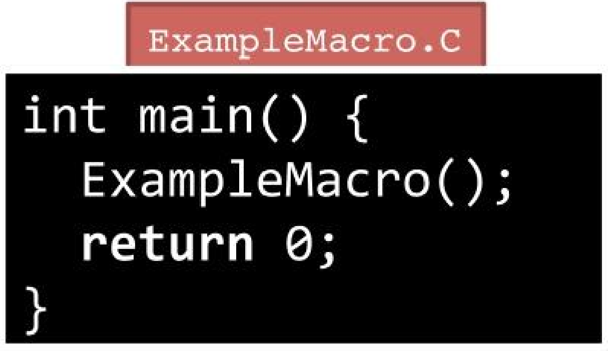
Interpretation and Compilation

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()
```

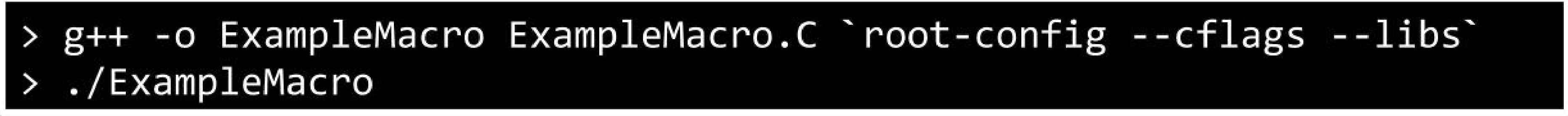
Generate shared library and execute function



ExampleMacro.C

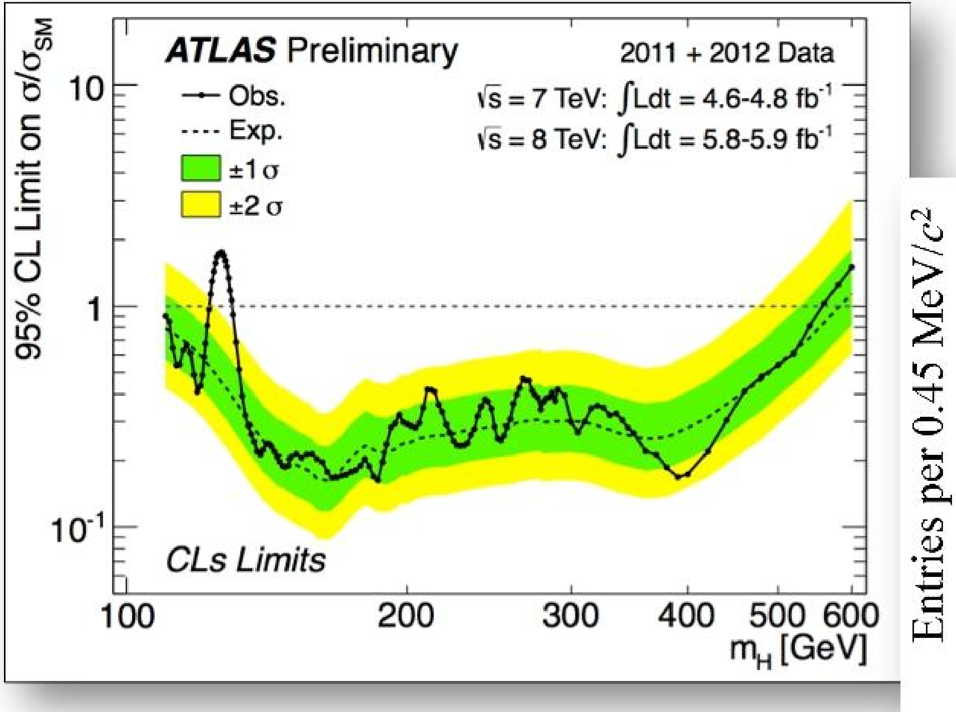
```
int main() {
  ExampleMacro();
  return 0;
}
```

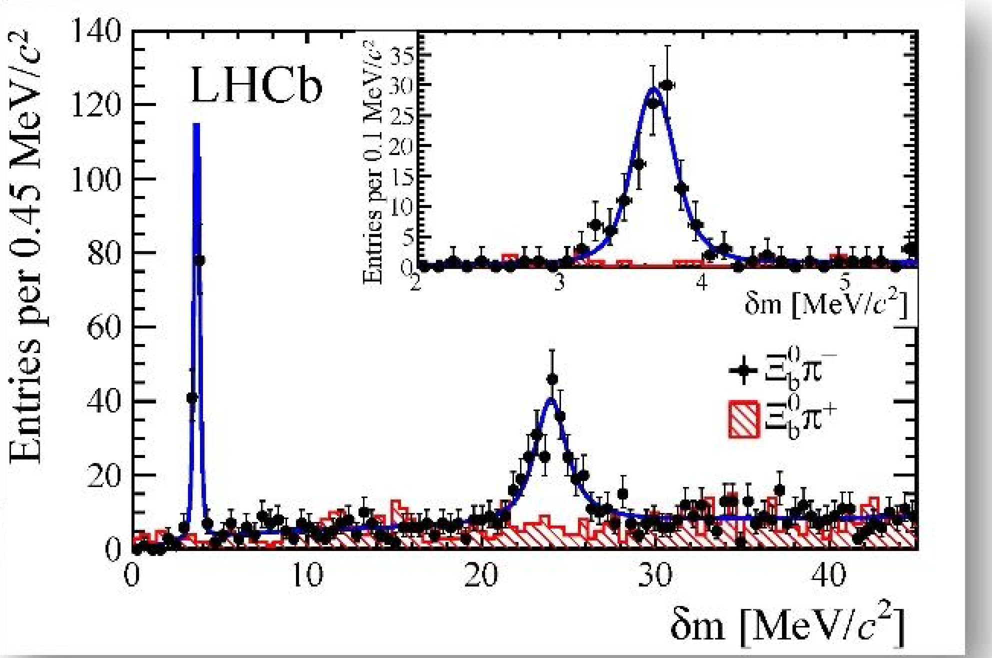
ROOT libraries can be also used to produce standalone, compiled applications:



```
> 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:

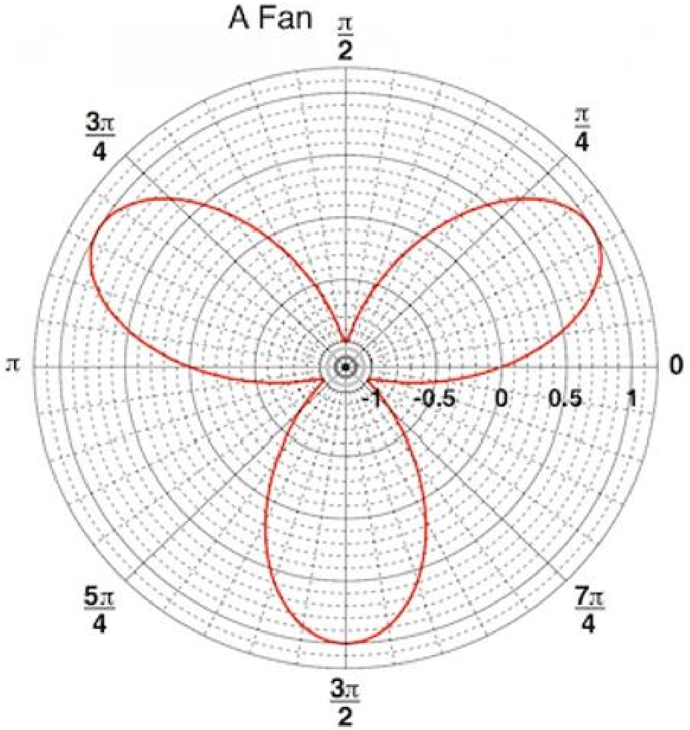
```
TGraphErrors(const char *filename,
             const char *format="%lg %lg %lg %lg",
             Option_t *option="");
```

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

Macro2.C

Polar Graphs

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

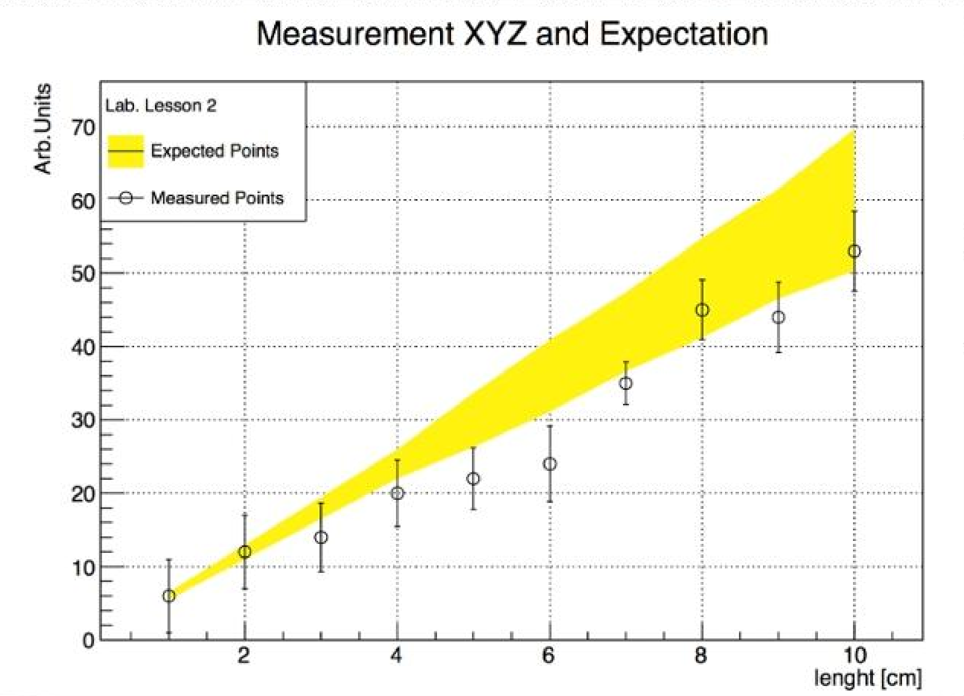


Macro3.C

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(
             (                                          lg",
             (
```

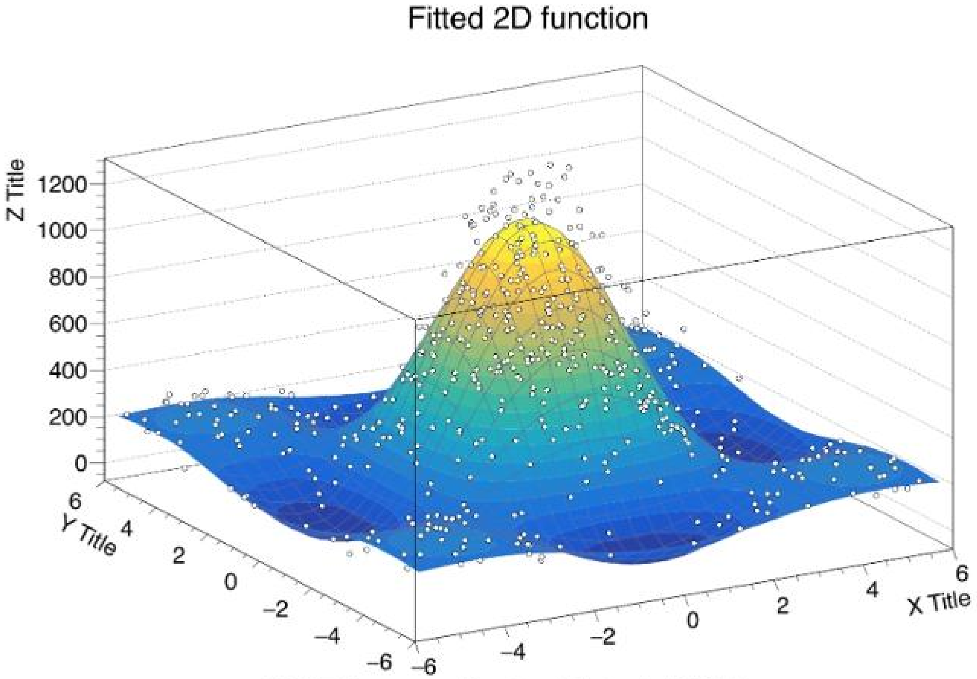


Let's have a look tc

Macro2.C

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.C

Histograms

Your First (well second!) Histogram

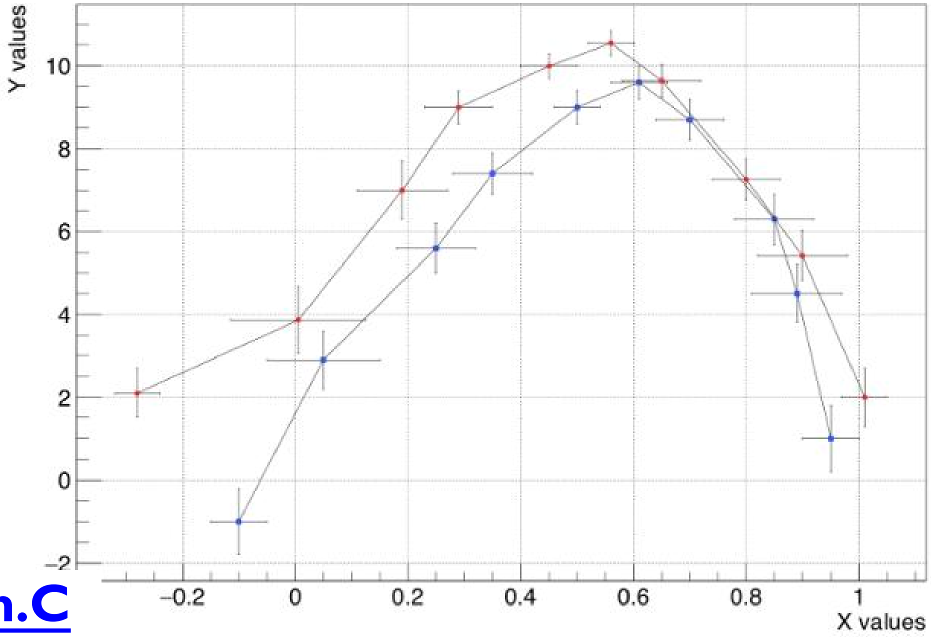
Add and Divide Histograms

Two-dimensional Histograms

Multiple Histograms

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.C

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)

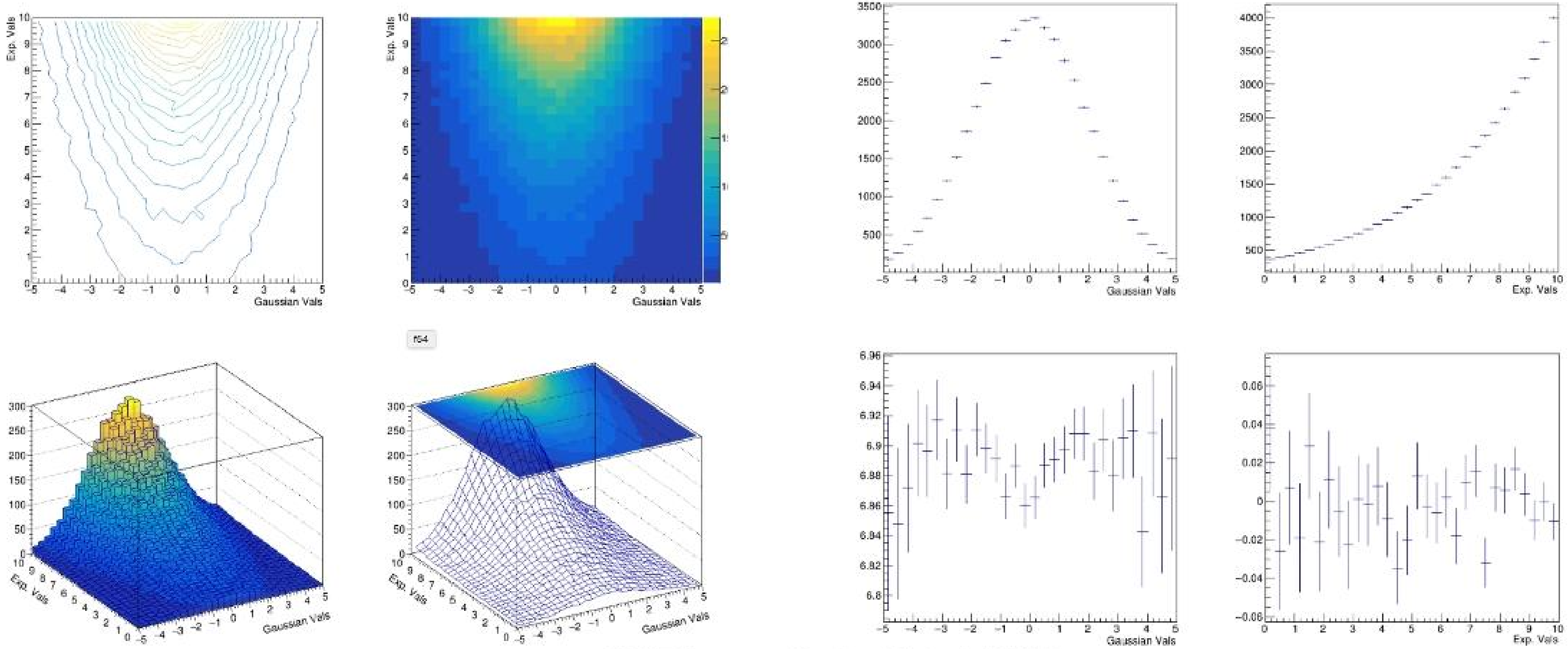
Exercise - Optional

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:



Input and Output

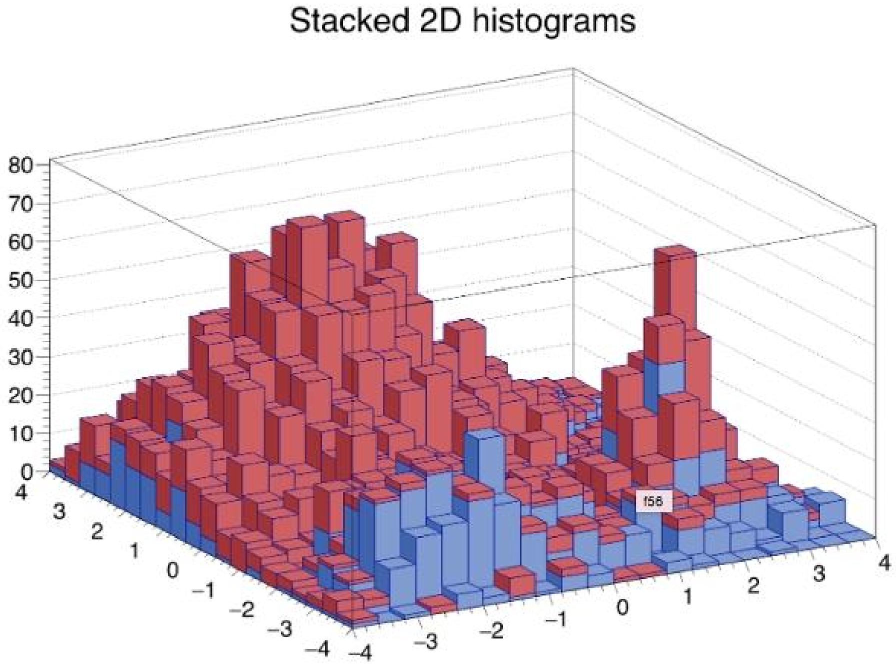
Storing Objects

N-tuples

Multiple Histograms

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

Class THStack



hstack.C

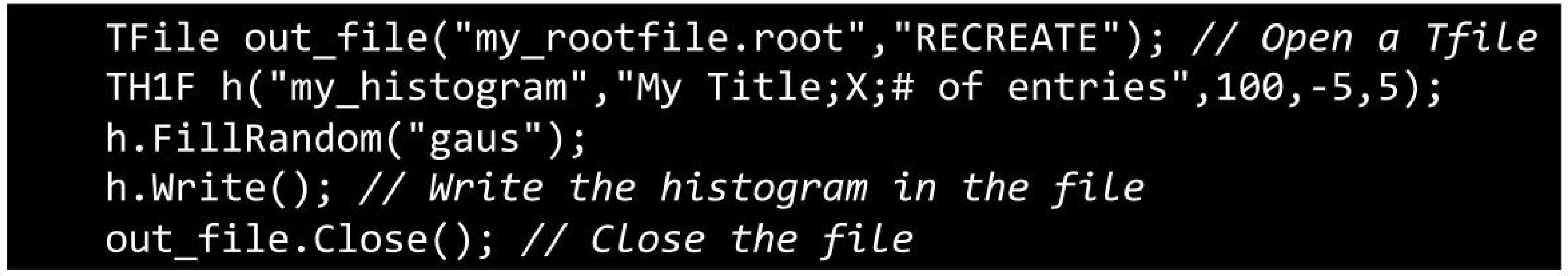
Storing Objects in a File

- 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
```

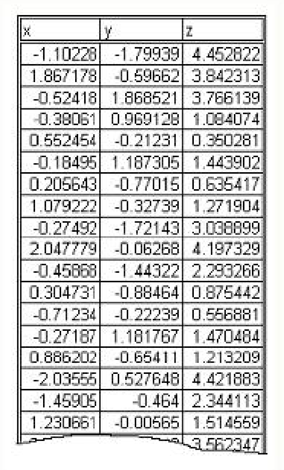
Exercise

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

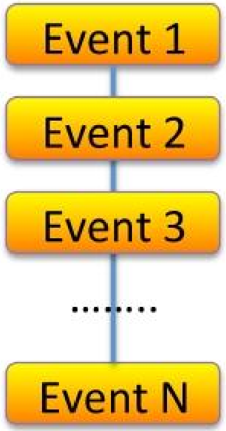
Trees

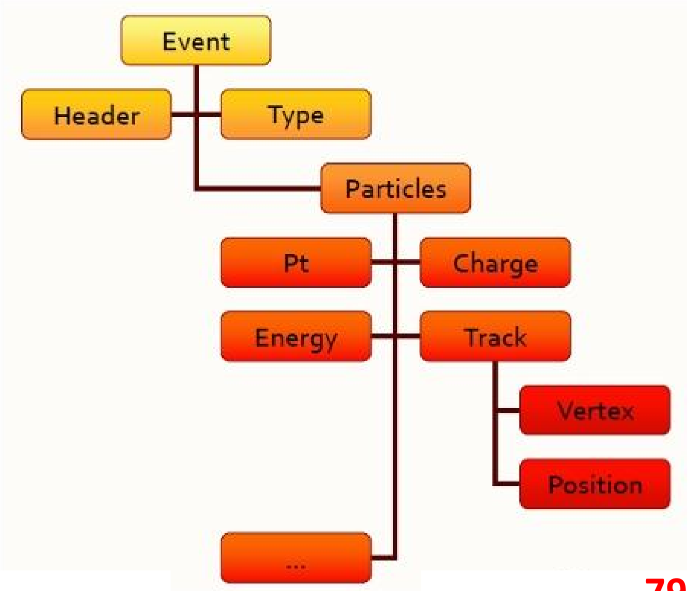
- The TTree is the data structure ROOT provides to store large quantities of same types objects
- Organised in branches, each one holding objects
- Organised in independent events, e.g. collision events
- Efficient disk space usage, optimised I/O runtime



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)



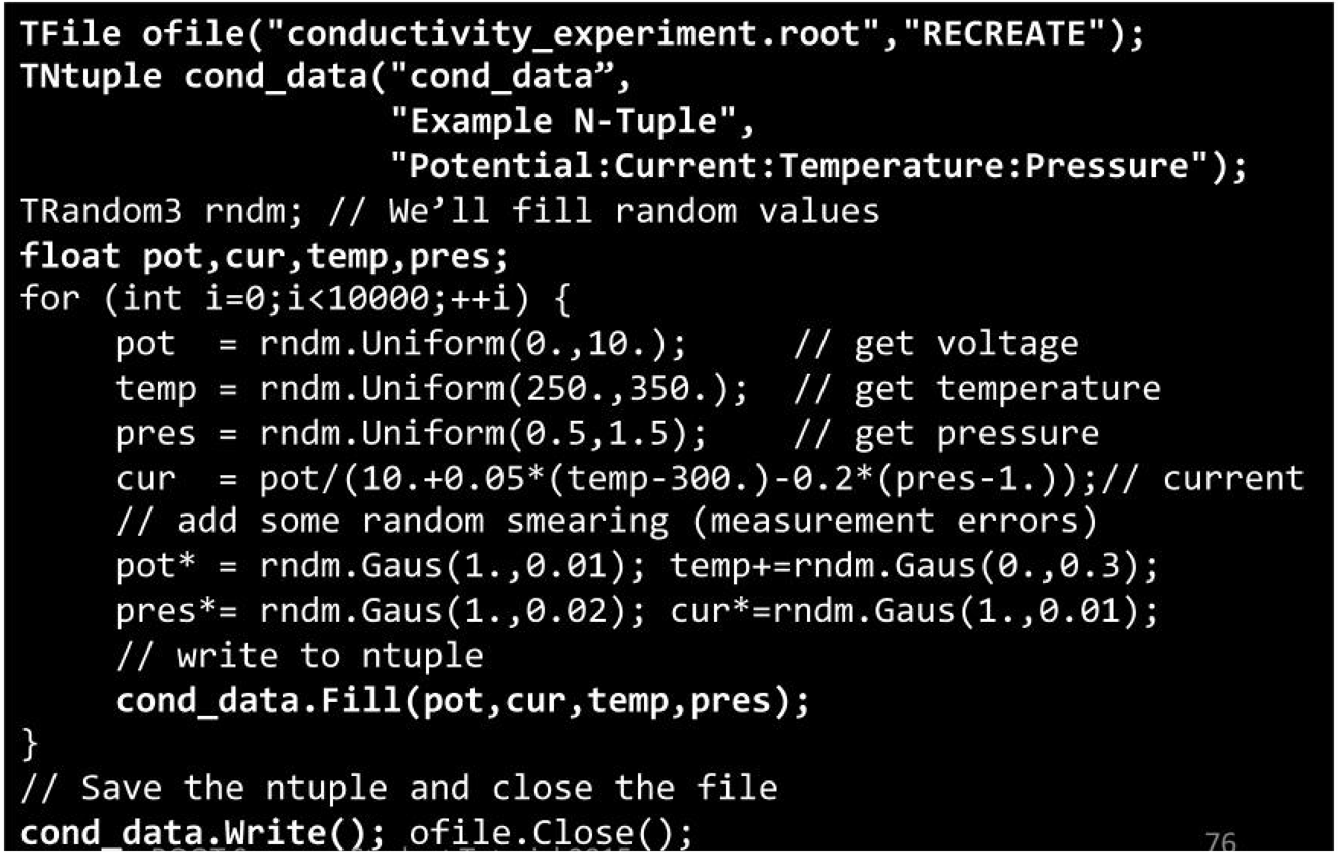


Ntuples

- 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



```
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();
```

write_ntuple_to_file.C

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 the Data

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

Check “read” examples

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");
```

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:

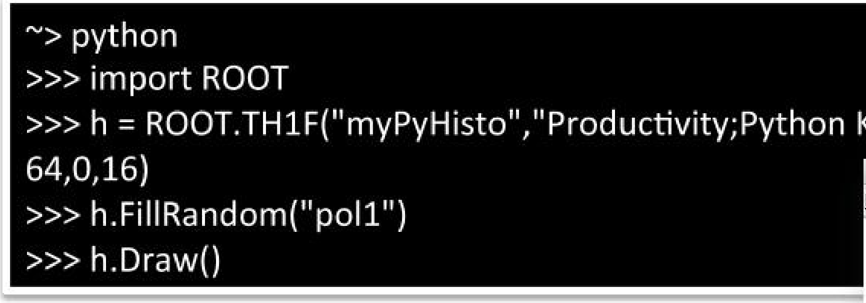
```
import ROOT
```

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

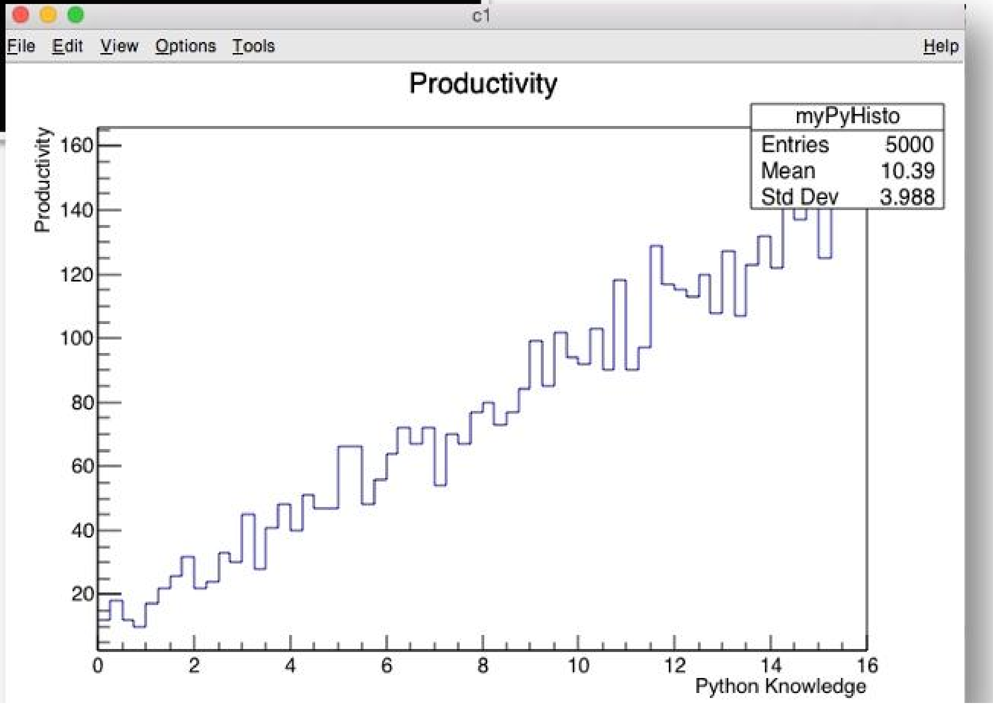
Extempore Exercise

- 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()
```



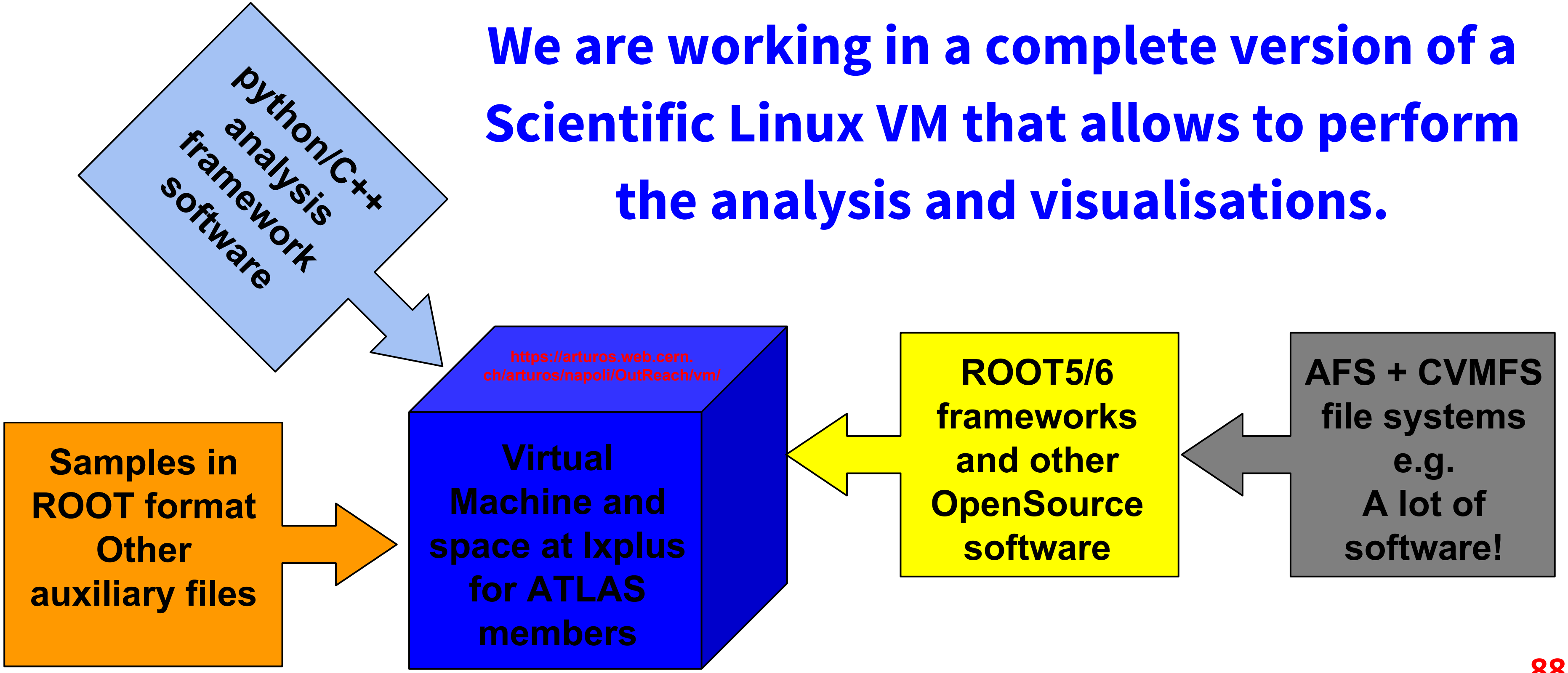
```
myPyHisto
Entries     5000
Mean       10.39
Std Dev    3.988
```

FillHistogram_Example_py

TTreeAccess_Example_py

The Virtual Machine

We are working in a complete version of a Scientific Linux VM that allows to perform the analysis and visualisations.



python/C++ analysis framework software

https://arturos.web.cern.ch/arturos/napoli/OutReach/vm/

Virtual Machine and space at lxplus for ATLAS members

Samples in ROOT format
Other auxiliary files

ROOT5/6 frameworks and other OpenSource software

AFS + CVMFS file systems e.g.
A lot of software!

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