University of Zurich

Introduction to ROOT

Annapaola de Cosa

PHYS451 - Experimental Particle Physics
Exercise class 1
20th September 2016

Practical info

- **Practical lectures**
  - Time: Tuesday 15:00-17:00
  - Place: Y11G34
  - Teaching assistants: Dr. Annapaola de Cosa, Dr. Silvio Donato

- **Dr. Annapaola de Cosa**
  - Email: annapaola.de.cosa@cern.ch
- **Dr. Silvio Donato**
  - Email: silvio.donato@cern.ch

- **Webpage of the course**
  - http://www.physik.uzh.ch/en/teaching/PHY451/HS2016.html

Documentation

- **When can you get informations?**
  - Official ROOT/PyROOT websites:
    - http://root.cern.ch/
    - https://root.cern.ch/root/html/tutorials/
    - https://root.cern.ch/pyroot
  - Some other tutorials
    - BaBar: http://www.slac.stanford.edu/BFROOT/www/doc/workbook/root1/root1.html
    - BaBar 2: http://www.slac.stanford.edu/BFROOT/www/doc/workbook/root2/root2.html

- **Internet is plenty of useful documentation/tutorial/examples**

Google

What is ROOT?

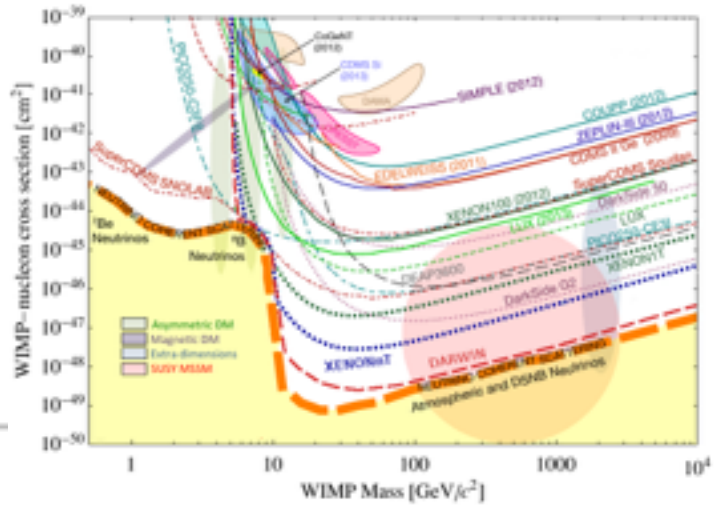
- **ROOT is an object oriented framework for data analysis/storage/ visualization**
- **ROOT is based on C++ language**
  - But it's integrated with other languages as well: Python, etc…
- **Provides many useful tools to:**
  - Access/ select/ save data
  - Perform computations
  - Produce results
- **Contains a lot of useful objects**
  - for histogramming/ fitting/ math computations/ plotting/ etc…
    - TH1 / TF1 / TMath/ TCanvas/ etc…
- **Developed and supported by HEP community:**
  - Documentation and tutorials: https://root.cern.ch/

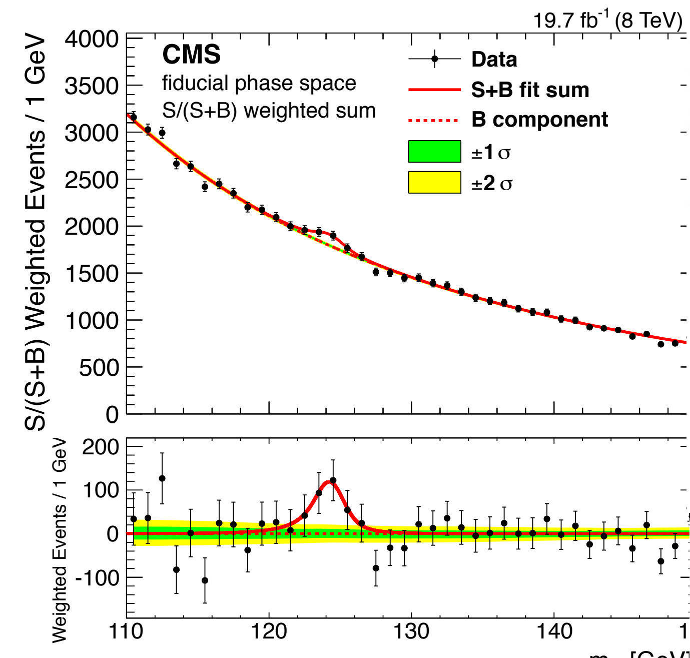
ROOT
Data Analysis Framework

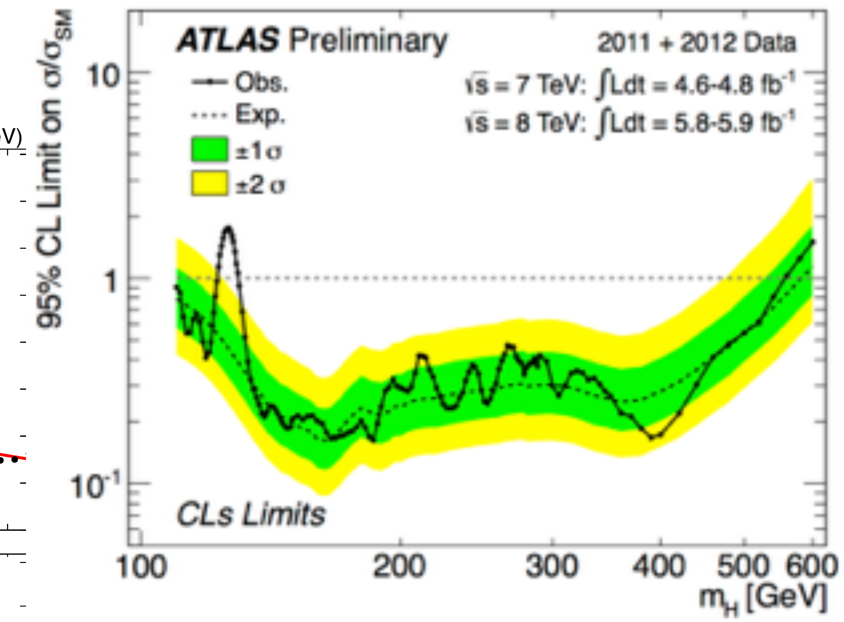
What can we do with ROOT?

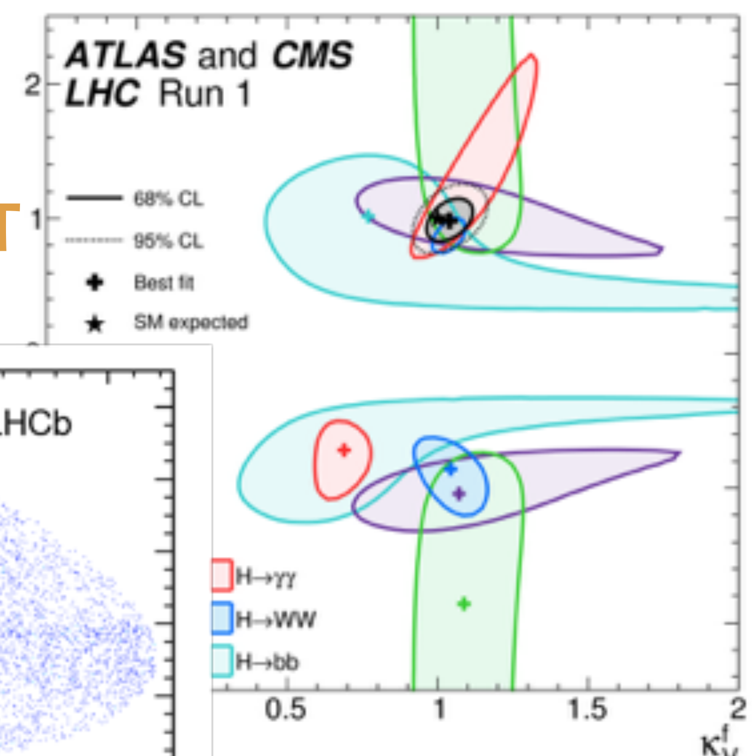
A lot of physics!!!

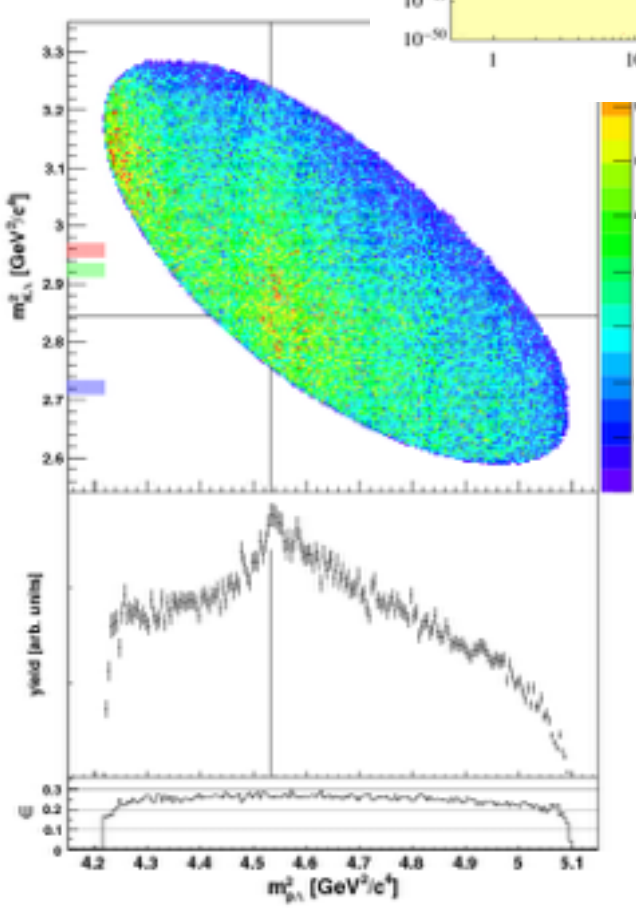
- **Access and analyse data**
- **Produce and visualise results**



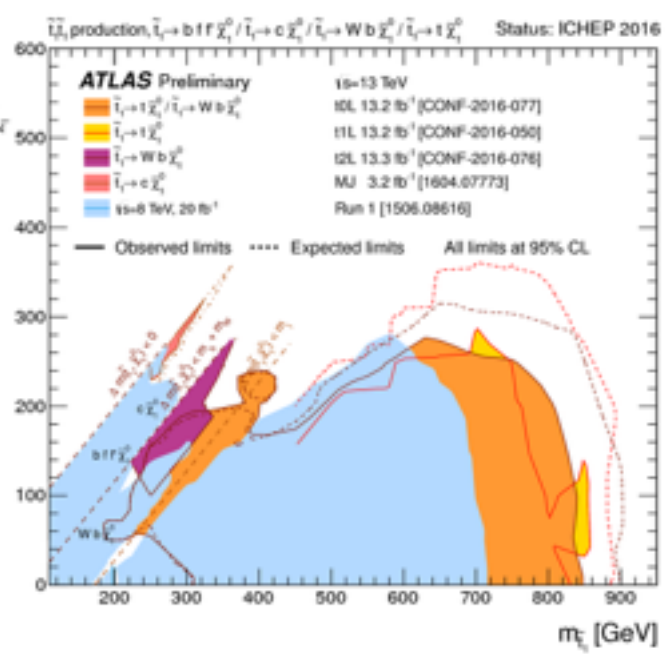


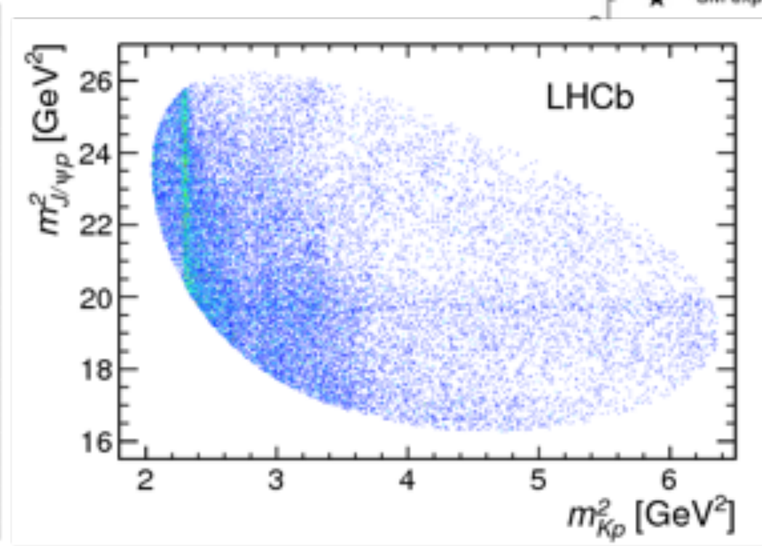






Tons of results produced with ROOT





Using ROOT

- **We can use ROOT in 3 ways:**
  - Interactively
  - With an interpreted macro (C/C++/Python)
  - With a compiled C/C++ program
- **Before starting we need to set up the environment**
  - In the command prompt digit:

```
# Set up ROOT environment
$ cd /app/cern
$ cd /root_v5.34.34/bin
$ source thisroot.sh
```

Interactive ROOT

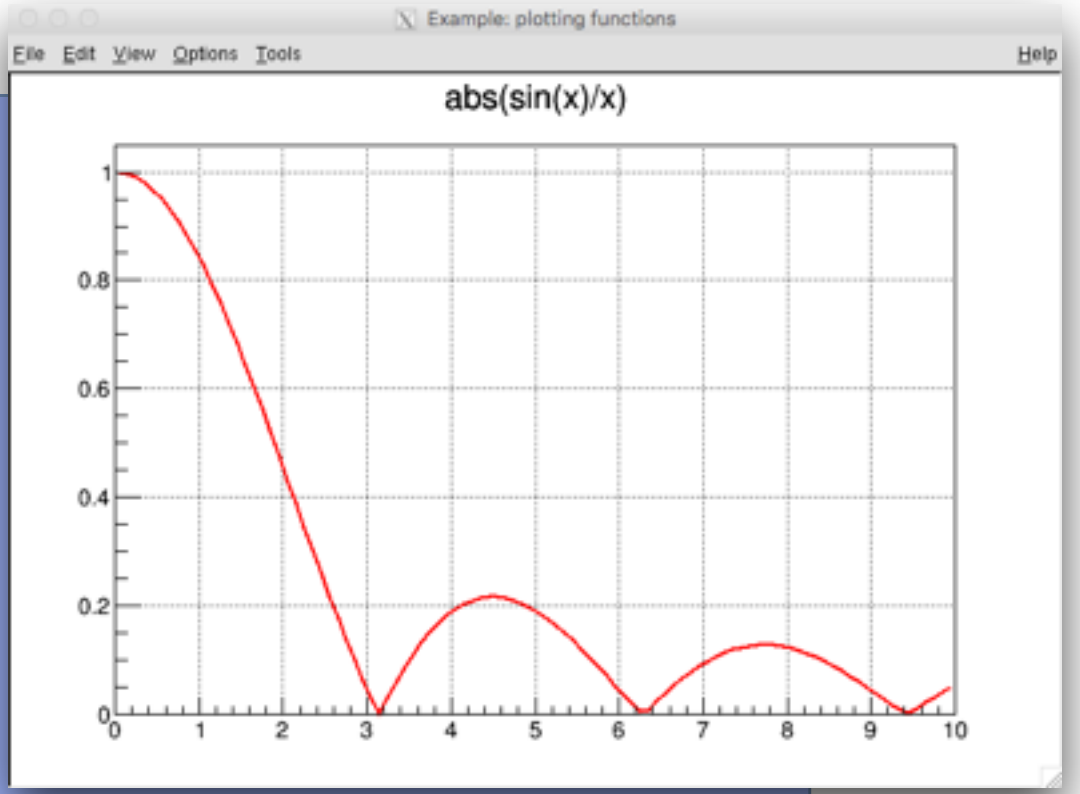
```
# Open interactive ROOT
$ root
[A verbose output will appear here]
# ROOT command line will appear as root [X]
root [0]
# We can use ROOT as a simple calculator
root [1] a = 10
(int) 10
root [2] b = 5
(int) 5
root [3] a + b
(int) 15
# Digit .q to exit ROOT
root [4] .q
```

PyRoot

- **Interactive ROOT is useful for testing and debugging**
  - But very limiting
- **An interpreted macro allows to build up a more complex analysis**
  - Python is very suitable for this purpose
- **PyROOT is a Python extension module conceived to allow interaction with ROOT through Python interpreter**
- **Documentation and tutorials can be found here:**
  - https://root.cern.ch/pyroot
  - https://root.cern.ch/doc/master/group__tutorial__pyroot.html

Let's start: Interactive PyROOT

```
# Open interactive Python ROOT
$ ipython
[A verbose output will appear here]
# iPython command line will appear as In [X]
# Import the objects we need from ROOT
In [1]: from ROOT import gROOT, TCanvas, TF1
# Delete variables created by previous executions
In [2]: gROOT.Reset()
# Instantiate a TCanvas object
# A TCanvas is a graphical objects container
In [3]: c = TCanvas( 'c', 'Example: plotting functions', 200, 10, 700, 500)
# A window with the canvas will pop up
# Create 1 dimensional function (TF1) and draw it
In [4]: fund = TF1('func', 'abs(sin(x)/x)', 0, 10)
In [5]: c.SetGridx()
In [6]: c.SetGridy()
In [7]: func.Draw()
In [8]: c.Update()
```



Click Ctrl + d to exit

ROOT Objects

- **ROOT contains a myriad of useful objects**
- **To get an overview of ROOT classes you can**
  - Have a look at the User Guide
  - Have a look at All Classes list
  - Search in Google

Class Reference page (e.g TH1 for histograms)

TH1F Class Reference

Histogram Library

tomato 1-D histogram with a float per channel (see TH1 documentation)}

Definition at line 575 of file TH1.h.

List of public member functions

Public Member Functions

TH1F ()
Constructor. More...

TH1F (const char *name, const char *title, Int_t nbinsx, Double_t xlow, Double_t xup)
Create a 1-Dim histogram with fix bins of type float (see TH1::TH1 for explanation of parameters) More...

TH1F (const char *name, const char *title, Int_t nbinsx, const Float_t *xbins)
Create a 1-Dim histogram with variable bins of type float (see TH1::TH1 for explanation of parameters) More...

TH1F (const char *name, const char *title, Int_t nbinsx, const Double_t *xbins)
Create a 1-Dim histogram with variable bins of type float (see TH1::TH1 for explanation of parameters) More...

TH1F (const TVectorF &v)
Create a histogram from a TVectorF by default the histogram name is "TVectorF" and title = "". More...

Most common ROOT classes

Histograms: TH1/TH2

Profile Histogram: TProfile

Latex Text: TLatex

Functions: TF1/TF2

Random numbers: TRandom

Legend: TLegend

ROOT files: TFile

Lines: TLine

Graphs: TGraph

Markers: TMarker

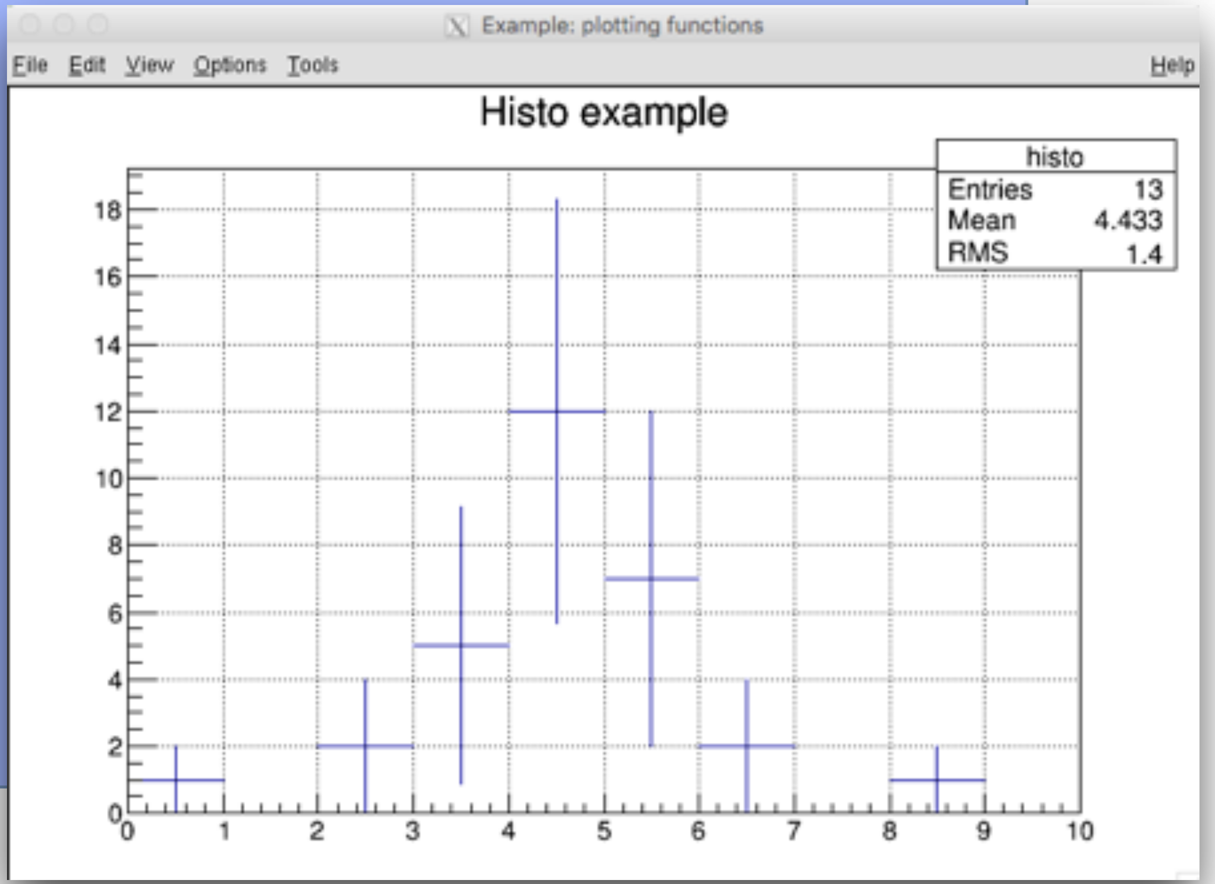
Graphical window: TCanvas

Histograms

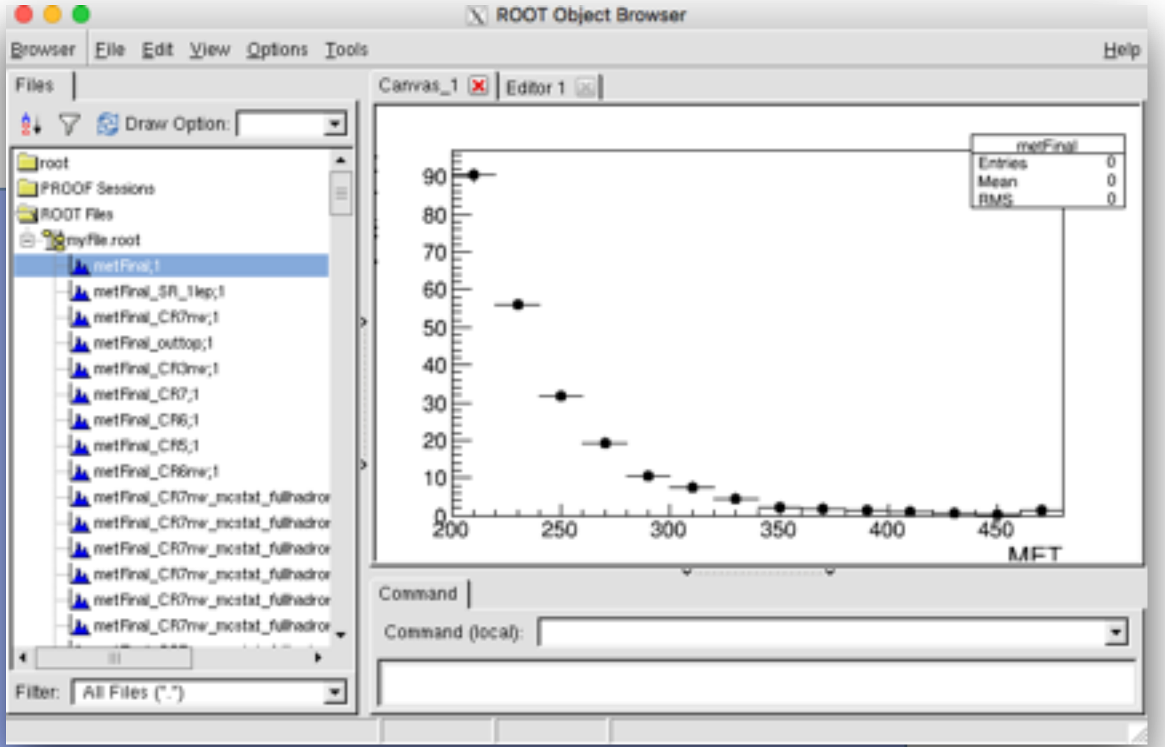
Histograms are widely used in Physics
They are representations of occurrence counting

TH1F('name', 'title', # of bins, x_{min}, x_{max})

```
# Import TH1F object from ROOT
In [9]: from ROOT import TH1F
# Create a 1D histogram of float (F) numbers: TH1F
In [10]: histo = TH1F("histo", "Histo example", 10, 0., 10.)
# Fill histo and draw it on the canvas
In [11]: histo.Fill(3)
In [12]: histo.Fill(4.8,4)
In [13]: histo.Fill(5.6,3)
In [14]: histo.Fill(3.9,2)
[continue filling]
In [15]: histo.Draw()
In [16]: c.Update()
In [17]: histo.GetMean()
Out[17]: 4.433333333333334
In [18]: histo.GetRMS()
Out[18]: 1.400079362829915
```



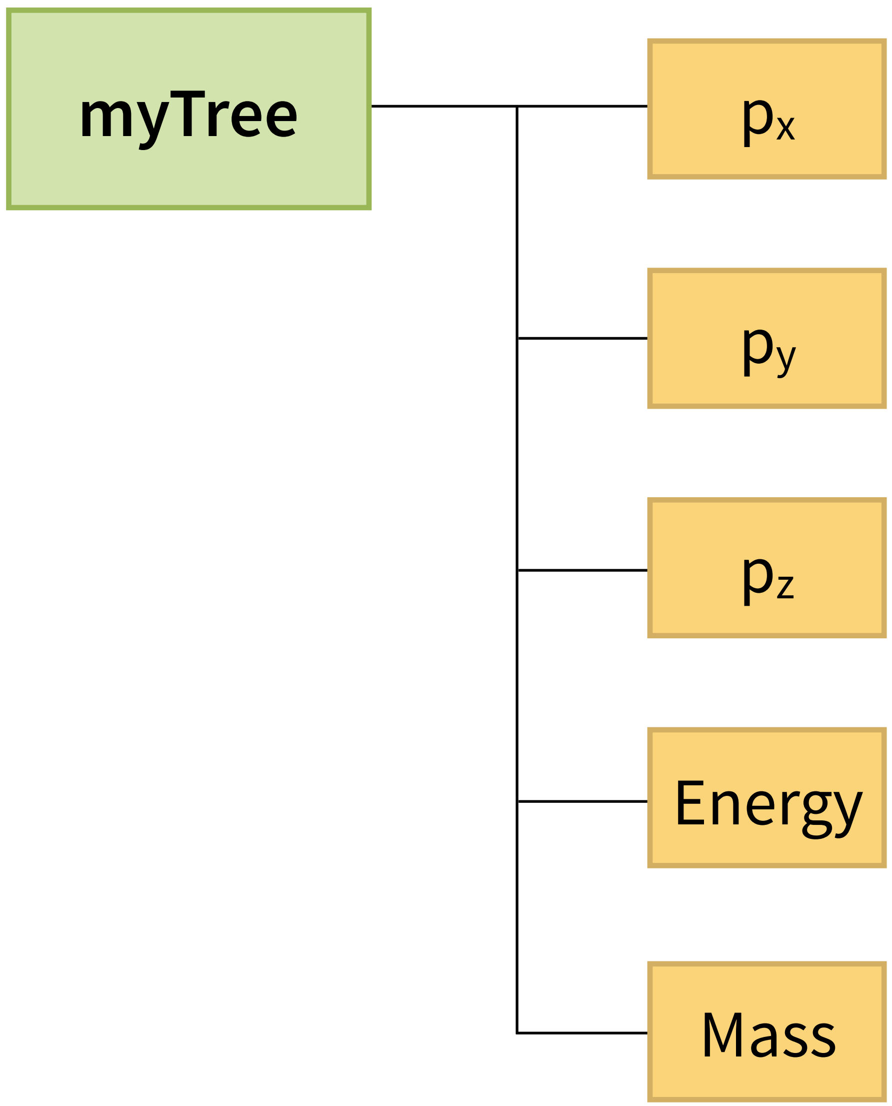
Loading and browsing files




```
In [19]: from ROOT import TFile, TBrowser
# Create a TFile object called f
In [20]: f = TFile("myFile.root")
# List file content and draw one of the histos
In [21]: f.ls()
TFile**       histoFile.root
 TFile*       histoFile.root
 KEY: TH1F        metFinal;1
 [visualization of the whole content]
In [22]: metFinal.Draw()
# Let's open a browser to visual file inspection
In [23]: b = TBrowser()
In [24]: f.Close()
# TBrowser is very useful for file inspection purposes, helps in visualise
file content and data distributions
```

ROOT Trees

- **ROOT trees are data containers**
- **Used to store/process data “per event”:**
  - i.e. the same structure is repeated for each event
  - Variables are saved as branches in the tree
    - Every branch can be read independently from others



- **Example: In a proton-proton collision several particles are generated**
  - Variables characterising particles trajectory, mass, momentum etc. can be stored as branches:
    - px, py, pz, Energy, Mass, etc…
  - One event per each collision

Writing a TTre

- **A ROOT tree, as any other ROOT object, can be used in a macro.**
- **This is an example of python macro to write a tree storing the variable "mass"**

```
from ROOT import TFile, TTree, TBranch, TRandom3

import numpy as n

f = TFile("myFile.root", "RECREATE")
f.cd()
t = TTree("myTree", "Example Tree")
m = n.zeros(1, dtype=float)
t.Branch('mass', m, 'mass/D')

rnd = TRandom3()
for i in xrange(10000):
    m[0]= rnd.Gaus(91,2.5)
    t.Fill()

f.Write()
f.Close()
```

Reading a TTree from a file

- **We can visualise the content of a TTree using TBrowser, but we can also check the values of a certain variable per each entry and draw its distribution**

```
# We can check the value of a certain variable, e.g Pt,
# for each entry using Scan method, or we can draw
# the distribution of this variable using Draw method
In [1]: from ROOT import TFile, TBrowser, TTree
In [2]: f = TFile("myFile.root")
In [3]: myTree = f.Get("myTree")
In [4]: myTree.Scan("mass")
In [5]: myTree.Draw("mass", "mass>50")
In [6]: f.Close()
```

Reading a TTree from a file

- **This is an example of python macro to read the branch (“mass”) from the tree we just created**

```
from ROOT import TFile, TTree

f = TFile("myFile.root")
myTree = f.Get("myTree")
entries = myTree.GetEntries()
print "Number of entries: ", entries

for i in xrange(entries):
    myTree.GetEntry(i)
    print myTree.mass
```

Fitting

- **ROOT has many predefined mathematical functions**
  - Exp(tau), Gauss(mean, sigma), Poisson(mean)…
- **Other functions can be defined by the user**
  - TF1 is the ROOT object for 1D functions
  - TF2 is the ROOT object for 2D functions
- **We can fit the distribution of a variable with a gaussian fit for instance:**

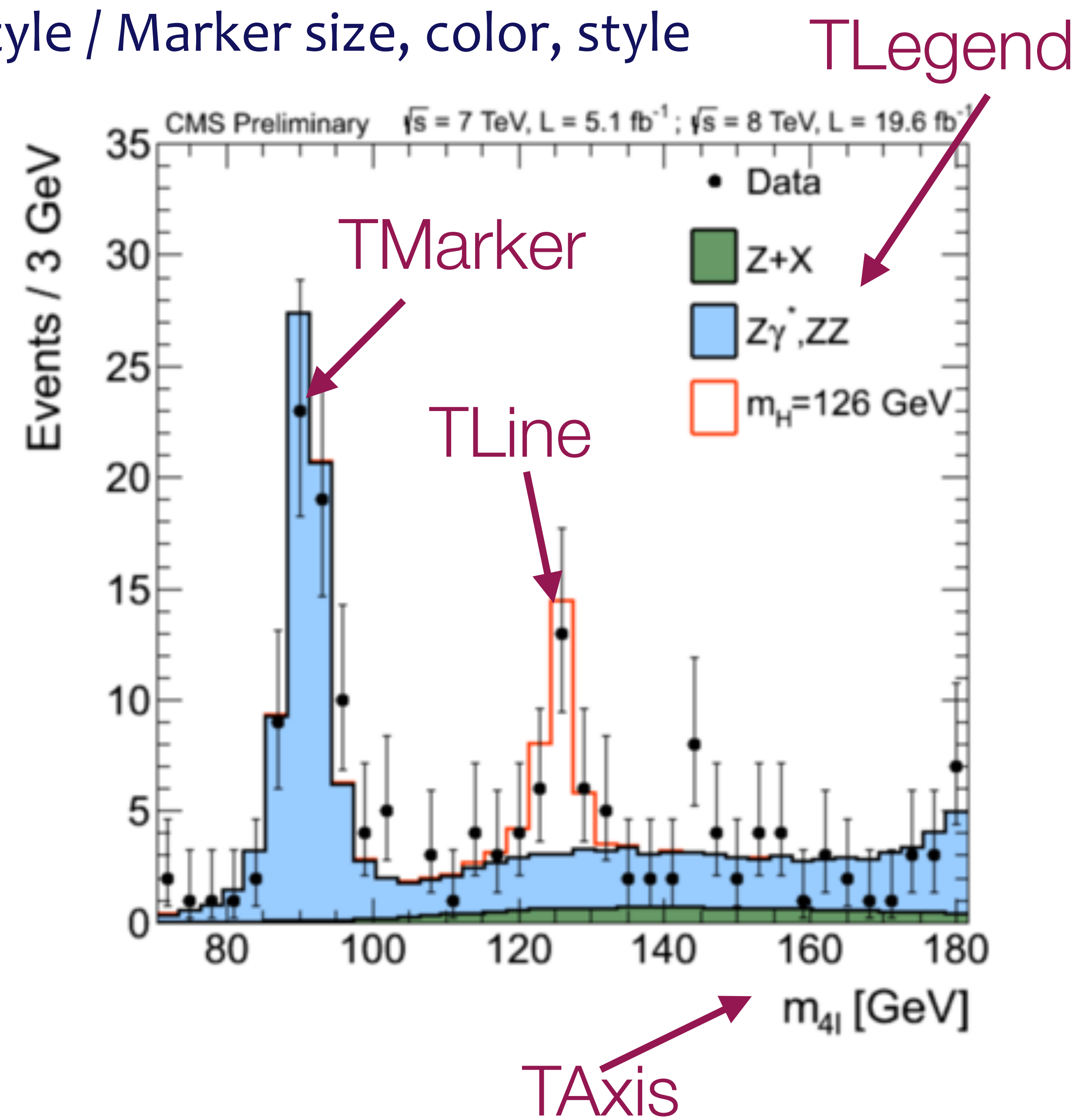
```
## Predefined function
h.Fit("gaus")

print h.GetFunction("gaus").GetParameter(1)
print h.GetFunction("gaus").GetParameter(2)
```

```
## User-defined function
fnc_gaus = TF1("MyGaus", "[0]*exp(-0.5*((x-[1])/[2])**2)", 50., 150.)
fnc_gaus.SetParameter(1, 91.5)
fnc_gaus.SetParameter(2, 2.45)
h.Fit("MyGaus")
```

Style

- **Plotting style can be set according to user preferences**
  - Filled histogram color / Line width, color, style / Marker size, color, style
  - Legend, text box
  - Axis title and table font size, etc.

```
h.SetFillColor(kAzure)
h.SetMarkerStyle(21)
…
h.GeXaxis().SetTitle(“m_{4l} [GeV]”)
h.GeYaxis().SetTitle(“Events/3 GeV”)
…
## Legends
leg = TLegend(0.1, 0.5, 0.8, 0.9)
leg.AddEntry(h, ‘description’)
leg.Draw(‘FLP’)
## F= show Fill color
## L = show Line color
## P = show Marker color/style
## To save an histogram to a pdf file
h.Draw()
c.Print(“nomePdf.pdf”)
```



Exercise 0

- **Reproduce examples on slides 7, 9, 12, 13**

Exercise 1

- **Write a root macro that randomly generates data as a signal peak (gaussian) plus a falling exponential background**
  - Generate 20 k events out of which 3 k signal events
  - Consider
    - a gaussian with mean 91.19 and width 2.49 to generate signal events
    - an exponential with parameter ~ 0.010/0.015)
  - Limit generation within the range [50., 150.]
- **Save data in a ROOT tree, e.g under branch name “mass”.**

Exercise 2

- **Write a macro that reads the root file created in the previous exercise**
- **Create an histogram of “mass” variable**
- **Fit the data with a user-defined function**
  - Inspect the parameters: Which are the fitted values for the mean and sigma of the gaussian?
  - Are they close to what you used for generation?
- **Produce an histogram with overlaid fitted function**
  - Set the X axis title to “M [GeV]”
  - Add a legend
  - Save the histogram in a pdf file