ROOT

Part 1

Introduction

ROOT

ROOT can be seen as a collection of building blocks for various activities, like:

- ★ **Data analysis: histograms, graphs, functions**
- ★ **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
- ★ **C++ interpretation**: full language compliance
- ★ **Multivariate Analysis** (TMVA): e.g. Boosted decision trees, Neural Nets
- ★ **Advanced graphics** (2D, 3D, event display)
- ★ **Declarative Analysis**: RDataFrame
- ★ And more: HTTP servering, JavaScript visualisation

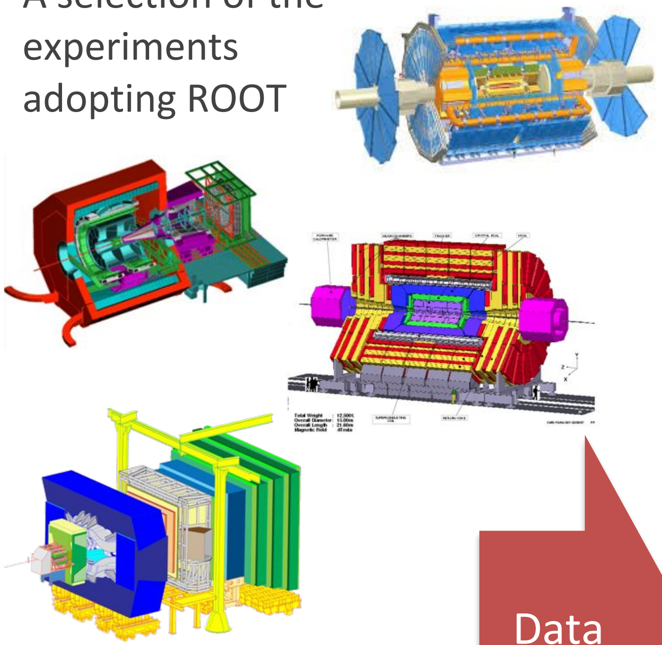
★ Unstar 595 Fork 433

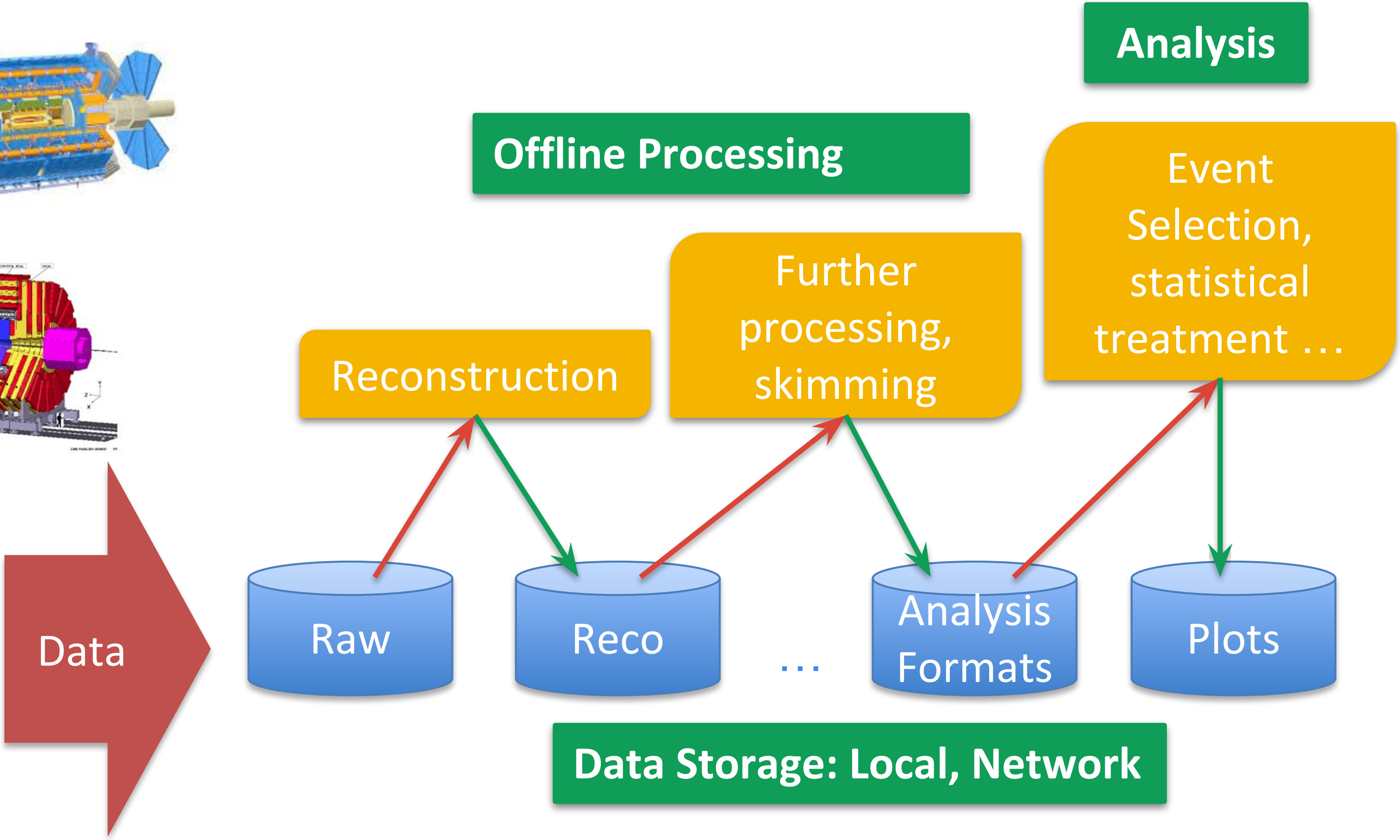
176 contributors

https://github.com/root-project/root

ROOT applications

A selection of the experiments adopting ROOT





Event Filtering

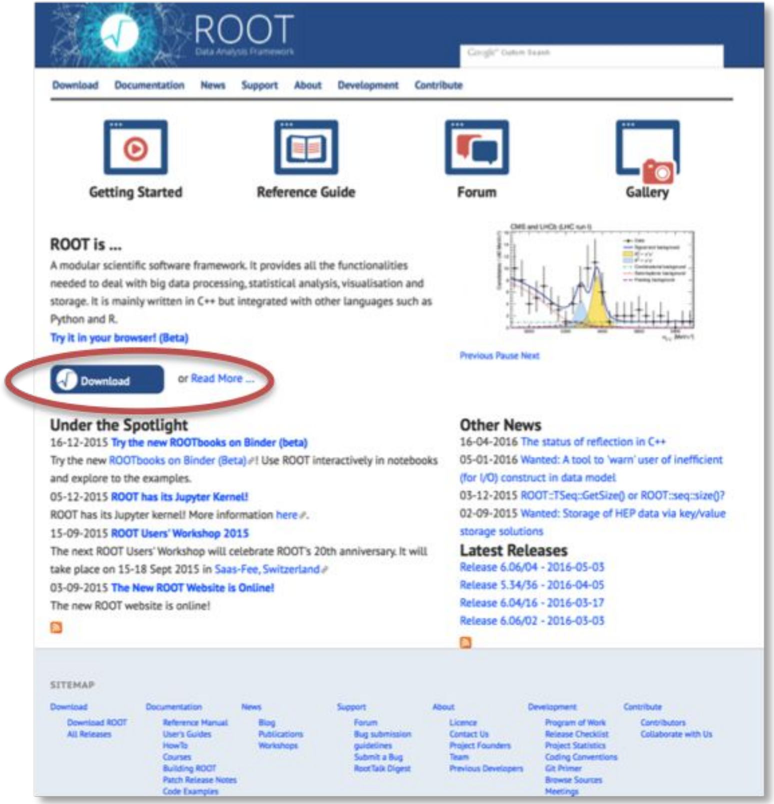
LHC data in ROOT format

~1EB

as of 2019

https://root.cern

★ 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
  - Forum
  - …



Resources

- ROOT Website: https://root.cern
- Training: https://github.com/root-project/training
- More material: https://root.cern/getting-started
  - Includes a booklet for beginners: **the "ROOT Primer"**
- Reference Guide: https://root.cern/doc/master/index.html
- Forum: https://root-forum.cern.ch

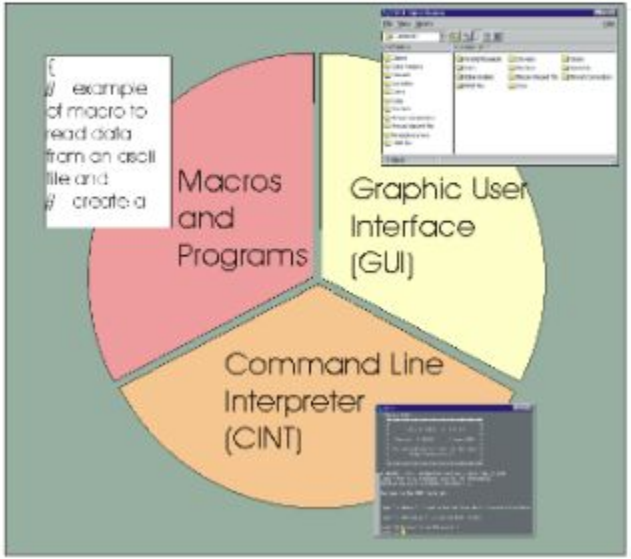
Installation from source

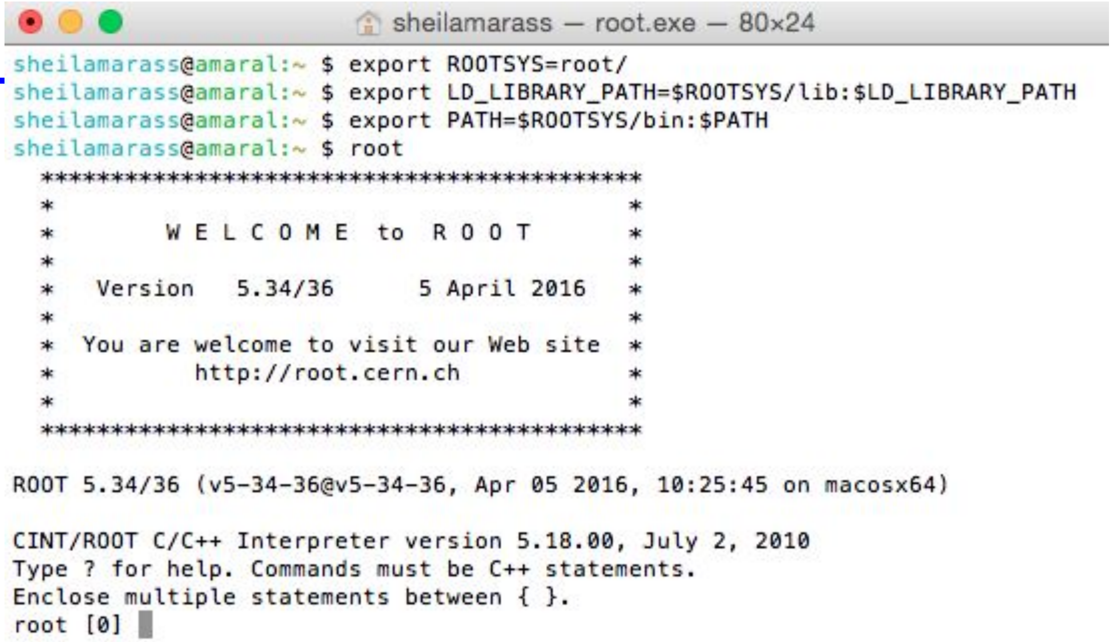
Expert Level

- Get the ROOT sources:
  - *git clone http://github.com/root-project/root*
  - *Or visit https://root.cern.ch/content/release-61600*
- Create a build directory and configure ROOT:
  - *mkdir rootBuild; cd rootBuild*
  - *cmake ../root*
  - [*https://root.cern.ch/building-root*](https://root.cern.ch/building-root) for all the config options
- Start compilation
  - *make -j*
- Prepare environment:
  - *. bin/thisroot.sh*

ROOT prompt and Macros

User Interfaces





.q	Quit
.L macro.C	Load a macro file
.x macro.C	Load and execute macro file
.x macro.C++	Compile and execute

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++, without the need of a compiler, like Python, Ruby, Haskell …
    - Code is **Just-in-Time compiled!**
  - Allows reflection (inspect layout of classes at runtime)
  - Is started with the command:

```
root
```

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

Controlling ROOT

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

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

- For example:
  - To quit root use **.q**
  - To issue a shell command use **.! <OS_command>**
  - To load a macro use **.L <file_name>** (see following slides about macros)
  - **.help** or **.?** gives the full list

ROOT as a calculator

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \dots$$
$$= \sum_{n=0}^{\infty} x^n$$

Here we make a step forward.
We declare **variables** and use a ***for*** control structure.

```
root [0] double x=.5
(double) 0.5
root [1] int N=30
(int) 30
root [2] double gs=0;
```

```
root [3] for (int i=0;i<N;++i) gs += pow(x,i)
root [4] std::abs(gs - (1/(1-x)))
(Double_t) 1.86265e-09
```

Interactivity

```
root [0] #include "a.h"
root [1] A o("ThisName"); o.printName()
ThisName
root [1] dummy()
(int) 42
```

a.h

```cpp
# include <iostream>
class A {
public:
   A(const char* n) : m_name(n) {}
   void printName() { std::cout << m_name << std::endl; }
private:
   const std::string m_name;
};

int dummy() { return 42; }
```

ROOT macros

- 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
                      ...             >
}
```

Unnamed ROOT macros

- ★ Macros can also be defined with no name
- ★ Cannot be called as functions!
  - ○ See next slide :)

```
{
        <           ...
             your lines of C++ code
                    ...              >
}
```

Running a macro

- ★ A 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 run by typing:

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

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

Advanced Users

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

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

Conventions

ROOT uses a set of coding conventions:

- ★ Classes begin with T
- ★ Non-class types end with _t
- ★ Member functions begin with a capital
- ★ Constants bekin with k
- ★ Global variables begin with g
- ★ Getters and setters begin with Get and Set
- ★ Predefined types in ROOT:
  - Int_t, Float_t, Double_t, Bool_t, etc
  - You can, however, use also the C++ types: int, double, etc…
- ★ ROOT has a set of global variables that apply to the session
  - For example the single instance of TROOT is accessible via the global gROOT and hold information relative to the current session:

```
gROOT->Reset();
gROOT->LoadMacro("ftions.cxx");
gSystem->Load("libMyEvent.so")
```

Sintax

Many of the commands we will use will have this general form:

This is called a “constructor”

```
TSomething* mything = new TSomething(stuff);
```

All ROOT classes start with T:
TFile
TH1
TTree
TCanvas
...

Means make a “pointer” (in the case, of type TSomething)

Name of pointer

C++ operator that allocates memory

Initializes the allocated memory with whatever “stuff” TSomething requires

For now, don’t worry about what a pointer is. It’s not important for this tutorial.

Note: In C++, if you you allocate memory using the “new” operator, you must later use “delete mything” to release the memory… otherwise your code will have a memory leak.
We will not worry about that today, but keep it in mind for your future code-writing

ROOTBooks

The Jupyter Notebook

A web-based interactive computing platform that combines code, equations, text and visualisations.

Many supported languages: C++, Python, Haskell, Julia…
One generally speaks about a “kernel” for a specific language

In a nutshell: an “interactive shell opened within the browser”

jupyter

http://www.jupyter.org

ROOT interfaces on Jupyter notebook

- ★ ROOT is well integrated with Jupyter Notebook, both for what concerns its Python and C++ interface
- ★ What is Jupyter Notebook? https://jupyter.org/
  - Language of choice, share notebooks, interactive output, big data integration
- ★ **How to integrate Jupyter notebook and ROOT:**
  - Install ROOT6 (> 6.05)
  - Install dependencies: pip install jupyter metakernel
  - Set up the ROOT environment (`. $ROOTSYS/bin/thisroot.[c]sh`) and then type in your shell: `root --notebook`

How It Looks Like

Text

Code

Graphics

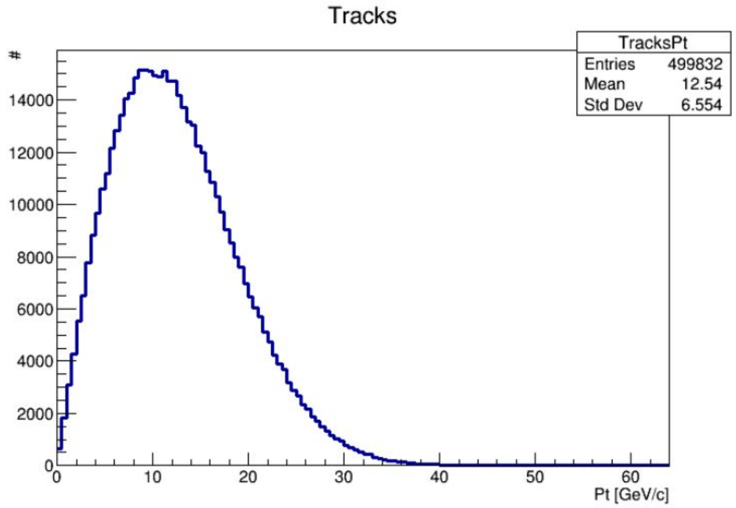
Access TTree in Python using PyROOT and fill a histogram

Loop over the TTree called "events" in a file located on the web. The tree is accessed with the dot operator. Same holds for the access to the branches: no need to set them up - they are just accessed by name, again with the dot operator.

In [1]:
```
import ROOT

f = ROOT.TFile.Open("http://indico.cern.ch/event/395198/material/0/0.root");
h = ROOT.TH1F("TracksPt","Tracks;Pt [GeV/c];#",128,0,64)
for event in f.events:
    for track in event.tracks:
        h.Fill(track.Pt())
c = ROOT.TCanvas()
h.Draw()
c.Draw()
```

To execute the code, click shift+enter



Let’s play with ROOT on Jupyter Notebook

You can fork to your GitHub account from:
https://github.com/ssilvado/ROOT-notebooks

The ROOT notebooks are based on the ROOT Primer
(https://root.cern.ch/guides/primer).

Histograms, Graphs and Functions

TCanvas

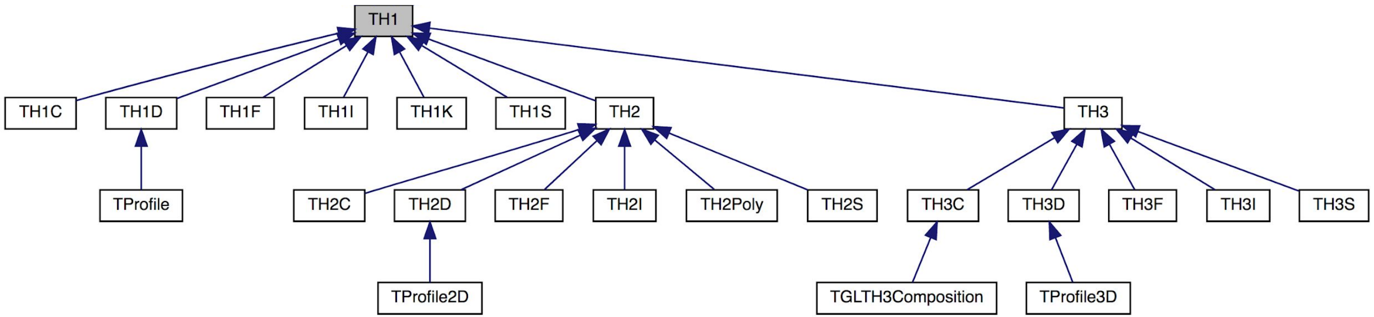
- ★ Canvases may be seen as windows.
- ★ In ROOT a graphical entity that contains graphical objects is called a Pad.

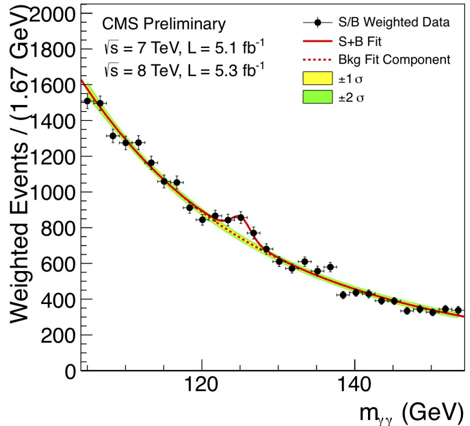
Command	Action
c1 = new TCanvas("c1","Title, w, h)	Creates a new canvas with width equal to w number of pixels and height equal to h number of pixels.
c1->Divide(2,2);	Divides the canvas to 4 pads.
c1->cd(3)	Select the 3rd Pad

c1->SetGridx();
c1->SetGridy();
c1->SetLogy();
	You can set grid along x and y axis.
	
	You can also set log scale plots.

Histograms

- Simplest form of data reduction
  - Can have billions of collisions, the Physics displayed in a few histograms
  - Possible to calculate momenta: mean, rms, skewness, kurtosis ...
- Collect quantities in discrete categories, the bins
- ROOT Provides a rich set of histogram types
  - We'll focus on histogram holding a *float* per bin





1D Histograms: ROOT

★ 1D histogram: TH1F *name = new TH1F(“name”, “title”, bins, lowest bin, highest bin);

h1 is an instance of a TH1F class

Example:

```
root [0] TH1F *h1 = new TH1F("h1", "x distribution", 100, -4, 4);
root [1] h1->FillRandom("gaus")
root [2] h1->Draw()
```

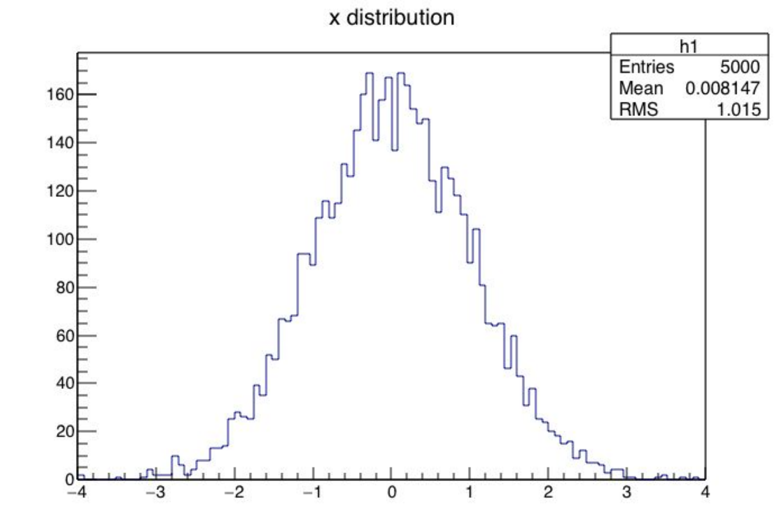
The Draw method display the histogram

1D Histograms: ROOT

★ 1D histogram: TH1F *name = new TH1F(“name”, “title”, bins, lowest bin, highest bin);

Example:

```
root [0] TH1F *h1 = new TH1F("h1", "x distribution", 100, -4, 4);
root [1] h1->FillRandom("gaus")
root [2] h1->Draw()
```



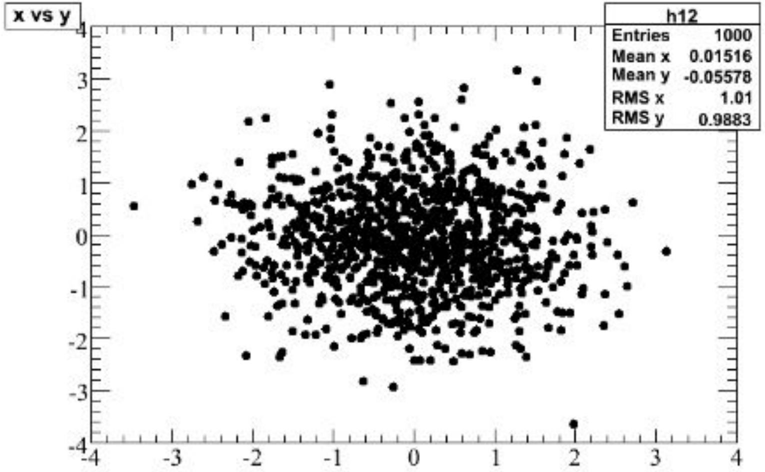
2D Histograms: ROOT

- ★ 2D histogram: TH2F *name = new TH2F(“name”, “title”, xbins, low xbin, up xbin, ybins, low ybin, up2 ybin);
- ★ Example:

TH2F *h12 = new TH2F(“h12”, “x vs y”, 100, -4, 4, 100, -4, 4);

h12->Fill(x,y);

h12->Draw();



3D Histograms: ROOT

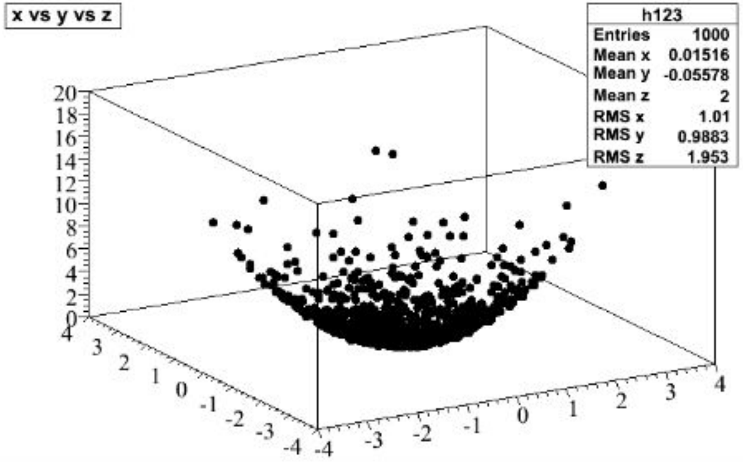
★ 3D histogram: TH3F *name = new TH3F(“name”, “title”, xbins, low xbin, up xbin, ybins, low ybin, up ybin, zbins, low zbin, up zbin);

★ Example:

TH3F *h123 = new TH3F(“h123”, “x vs y vs z”, 100, -4, 4, 100, -4, 4, 100, -4, 4);

h123>Fill(x,y,z);

h123->Draw();

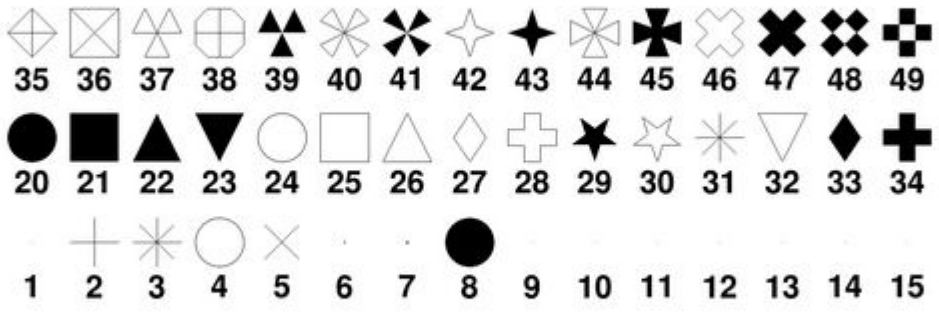


Histograms properties

Command	Parameters
GetMean()	Mean
GetRMS()	Root of Variance
GetMaximum()	Maximum bin content
GetMaximumBin(int bin_number);	Location of maximum
GetBinCenter(int bin_number);	Center of bin
GetBinContent(int bin_number);	Content of bin

Histogram cosmetics

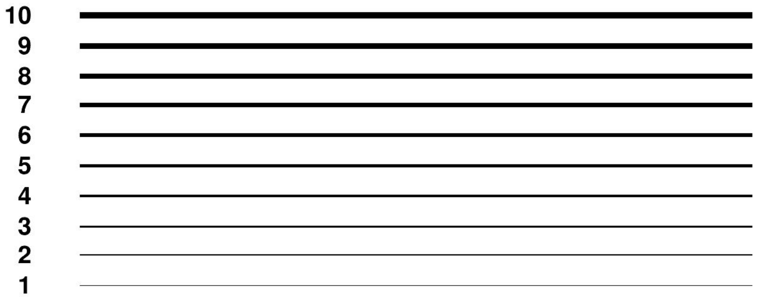
h1.SetMarkerStyle();



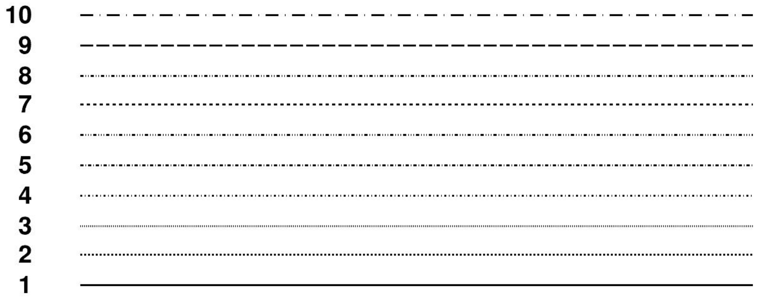
h1.SetFillColor();

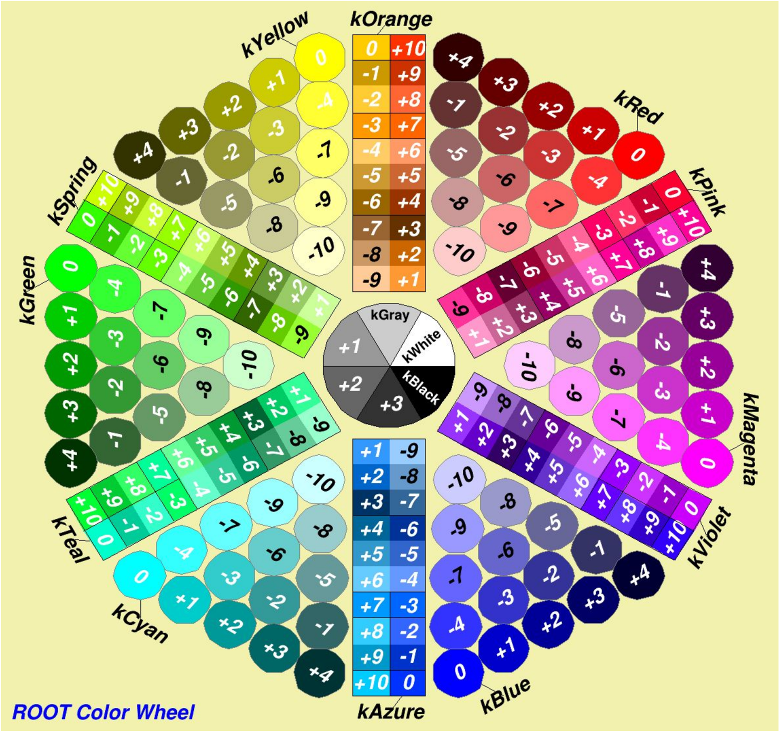
Histogram cosmetics: lines

h1.SetLineWidth();



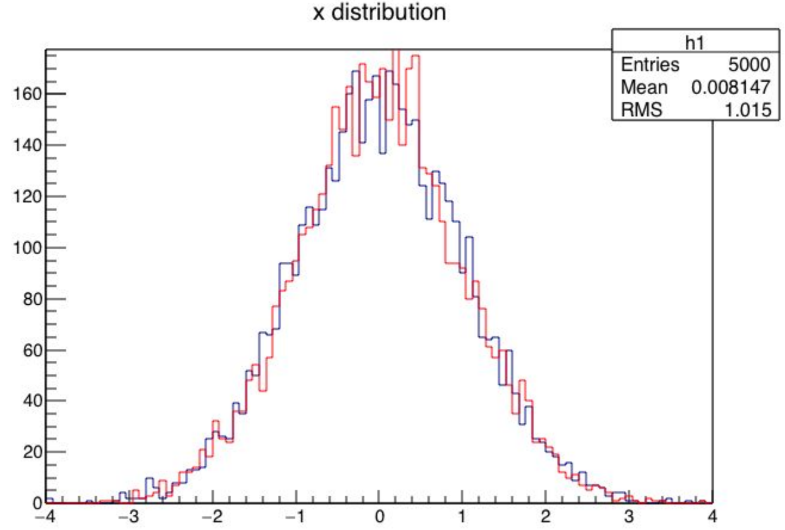
h1.SetLineStyle();





Histogram Drawing Options

```
root [0] TH1F *h1 = new TH1F("h1", "x distribution", 100, -4, 4);
root [1] TH1F *h1same = new TH1F("h1same", "x distribution", 100, -4, 4);
root [2] h1->FillRandom("gaus")
root [3] h1same->FillRandom("gaus")
root [4] h1->Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [5] h1same->Draw("same")
```

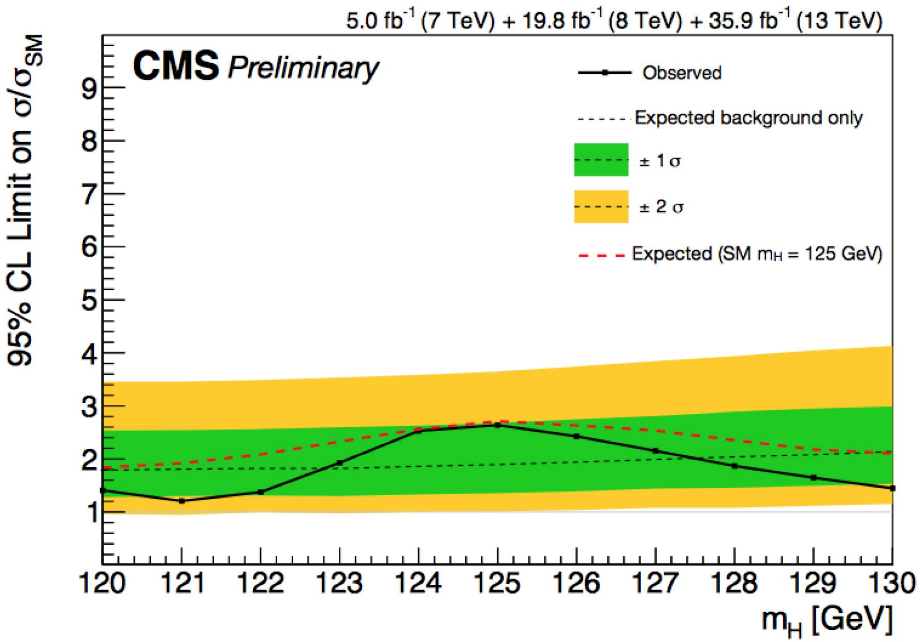


" SAME": Superimpose on previous picture in the same pad.
" CYL": Use cylindrical coordinates.
" POL": Use polar coordinates.
" SPH": Use spherical coordinates.
" PSR": Use pseudo-rapidity/phi coordinates.
" LEGO": Draw a lego plot with hidden line removal.
" LEGO1": Draw a lego plot with hidden surface removal.
" LEGO2": Draw a lego plot using colors to show the cell contents.
" SURF": Draw a surface plot with hidden line removal.
" SURF1": Draw a surface plot with hidden surface removal.
" SURF2": Draw a surface plot using colors to show the cell contents.
" SURF3": Same as SURF with a contour view on the top.
" SURF4": Draw a surface plot using Gouraud shading.
“ SURF5”: Same as SURF3 but only the colored contour is drawn.

Note: Please check chapter 3 in user’s guide to learn more about options.

Graphs

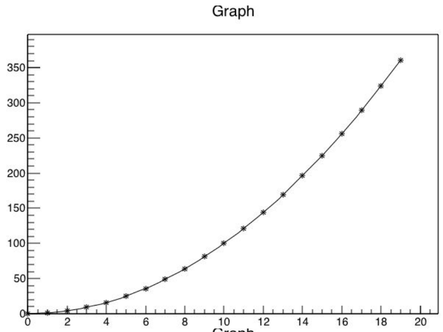
- ★ Graphics object made of two arrays X points
- ★ Display points and errors
- ★ Not possible to calculate momenta
- ★ Not a data reduction mechanism
- ★ **Fundamental to display trends**
- ★ Focus on TGraph and TGraphErrors



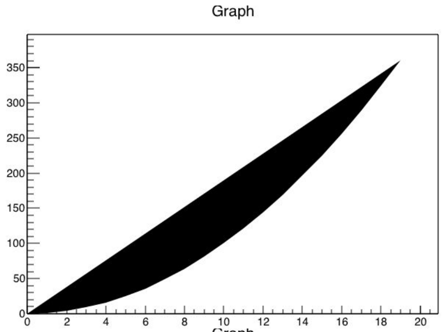
Graph: example

```
root [0] Int_t n=20;
root [1] Double_t x[n], y[n];
root [2] for(Int_t i=0; i<n; i++){ x[i]=i; y[i]=i*i; }
root [3] TGraph *gr1=new TGraph(n,x,y);
root [4] gr1->Draw("AC*");
```

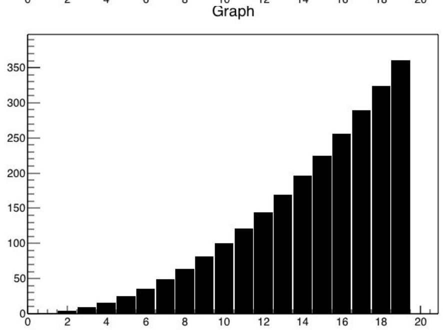
AC*



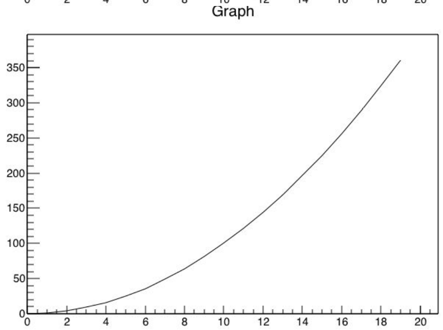
AF



AB

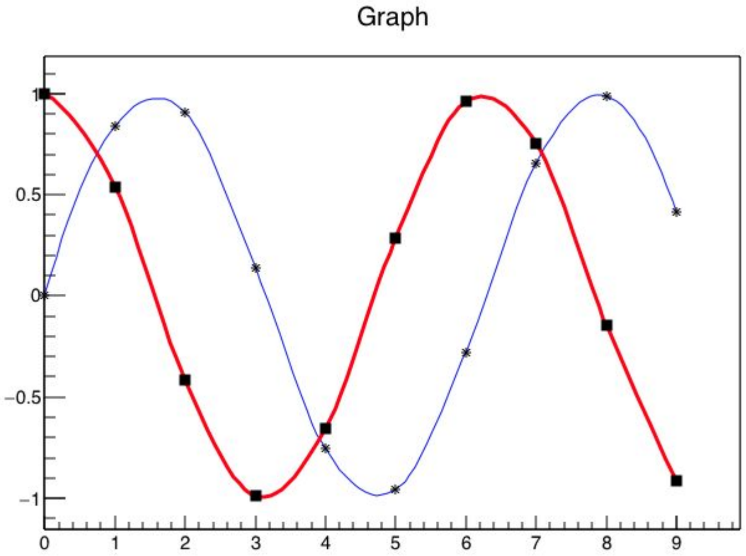


AL



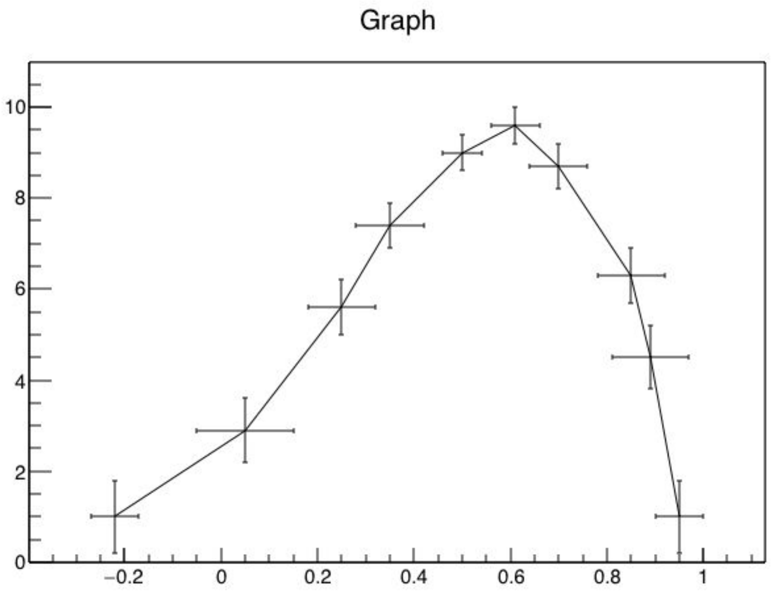
Superimpose two graphs

```
root [0] Int_t n=10;
root [1] Double_t x[n], y[n], x1[n], y1[n];
root [2] for(Int_t i=0; i<n; i++){ x[i]=i; y[i]=sin(i); x1[i]=i; y1[i]=cos(i); }
root [3] TGraph *gr1=new TGraph(n,x,y);
root [4] TGraph *gr2=new TGraph(n,x1,y1);
root [5] gr1->SetLineColor(4);
root [6] gr2->SetLineWidth(3);
root [7] gr2->SetMarkerStyle(21);
root [8] gr2->SetLineColor(2);
root [9] gr1->Draw("AC*");
Info in <TCanvas::MakeDefCanvas>:  created defau
root [10] gr2->Draw("CP");
```



Graph with error bars

```
root [0] Int_t n=10;
root [1] Float_t x[n]={-.22,.05,.25,.35,.5,.61,.7,.85,.89,.95};
root [2] Float_t y[n]={1,2.9,5.6,7.4,9,9.6,8.7,6.3,4.5,1};
root [3] Float_t ex[n]={.05,.1,.07,.07,.04,.05,.06,.07,.08,.05};
root [4] Float_t ey[n]={.8,.7,.6,.5,.4,.4,.5,.6,.7,.8};
root [5] TGraph *gr1=new TGraphErrors(n,x,y,ex,ey);
root [6] gr1->Draw();
```

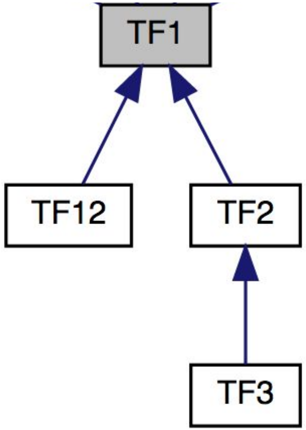


Functions

- Mathematical functions are represented by the **TF1** class
- They have names, formulas, line properties, can be evaluated as well as their integrals and derivatives
  - Numerical techniques for the time being

option	description
"SAME"	superimpose on top of existing picture
"L"	connect all computed points with a straight line
"C"	connect all computed points with a smooth curve
"FC"	draw a fill area below a smooth curve

From the TGraphPainter documentation:
https://root.cern.ch/doc/master/classTGraphPainter.html



Functions

Can describe functions as:

- Formulas (strings)
- C++ functions/functors/lambdas
  - Implement your highly performant custom function
- With and without parameters
  - Crucial for fits and parameter estimation

ROOT as a function plotter

★ The class TF1 represents one-dimensional functions (e.g. *f(x)*):

Declare a pointer to an object of type TF1.
The pointer's name is f1

Name of the function,
Can be nearly anything

Define the function using C-style math
x - is the evaluation variable

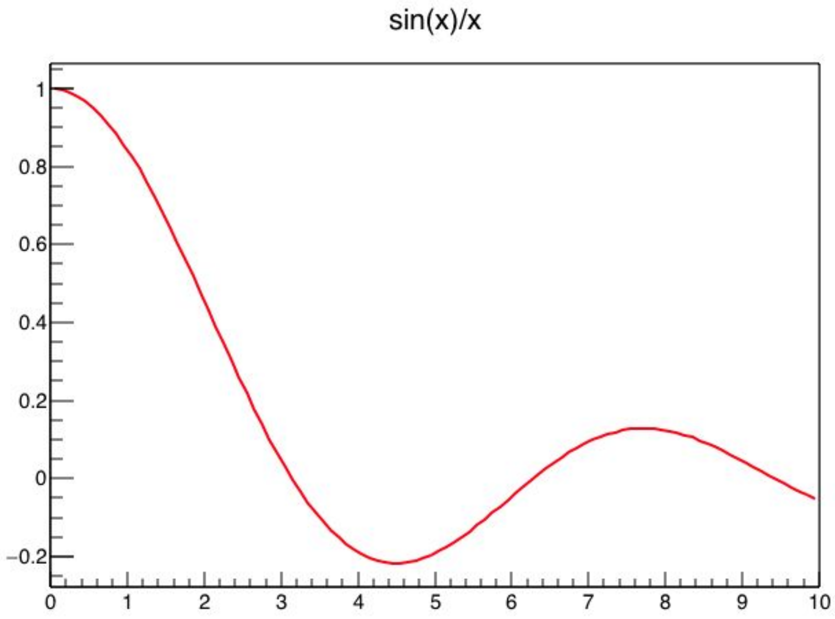
```
root [0] TF1 f1("f1","sin(x)/x",0.,10.); //name, formula, min, max
root [1] f1.Draw();
```

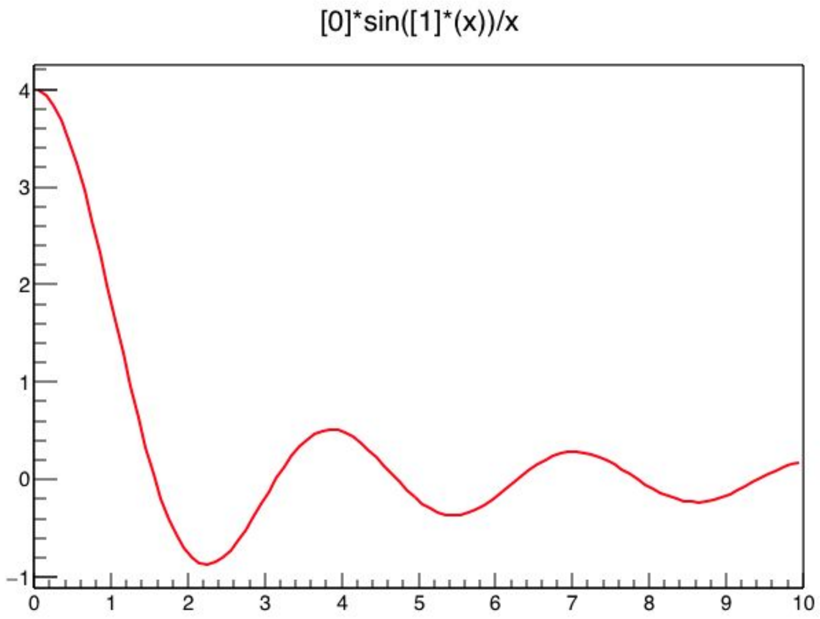
★ An extended version of this example is the definition of a function with parameters:

[0] and [1] - numbers in "[..]" are parameters, and can be set externally.

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

ROOT as a function plotter





ROOT TFile and TTree

ROOT Command Line: Some Objects

Let's open a file ([histograms.root]) and see what is inside

Declare a pointer to a TFile object

Create the object, it will point to (i.e. open the file histograms.root)

```
root [0] .ls()
root [1] TFile * input = new TFile("histograms.root");
root [2] input->ls();
TFile**         histograms.root
 TFile*         histograms.root
  KEY: TH1F     GausHist1d;1    One dimensional Gaussian Distribution
  KEY: TH2F     GausHist2d;1    Two dimensional Gaussian Distribution
```

Use the pointer "input" to list its file's contents with the member operator "->" and function "ls()"

There are two histograms in this file. Both are object with properties and functions we can use to display our data

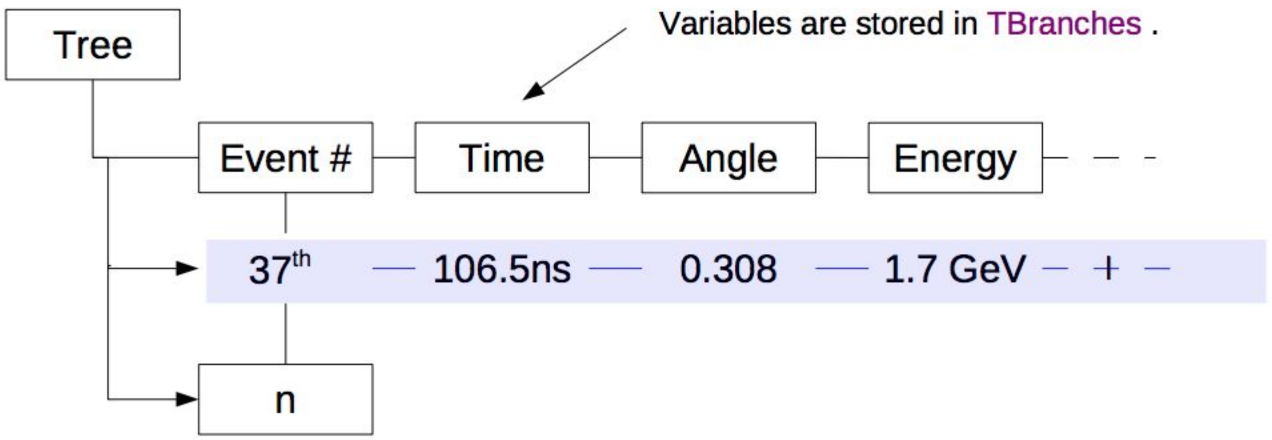
Declare a pointer to a TH1F object

Tell it to point to the object "GausHist1d" stored in our file

```
root [3] TH1F * my1Dhist = (TH1F*)input->Get("GausHist1d");
root [4] my1Dhist->Draw();
```

ROOT TTree

- ★ A TTree is a data structure for organizing and manipulating several data variables at once
- ★ Capable of drawing histograms on the fly including making selection cuts on the data
- ★ Uses ROOT’s internal compression algorithms to reduce the data size
  - Very useful for data storage



ROOT Command Line: TTree Example

```
Troot [0] TFile *f1 = new TFile("tree.root");
root [1] f1->ls();
TFile**         tree.root
 TFile*         tree.root
  KEY: TTree    tree1;1 Reconstruction ntuple
root [2] TTree *mytree = (TTree *)f1->Get("tree1");
root [3] mytree->Print();
******************************************************************************
*Tree    :tree1     : Reconstruction ntuple                                  *
*Entries :   100000 : Total =         2810673 bytes  File  Size =    2171135 *
*        :          : Tree compression factor =   1.30                       *
******************************************************************************
*Br    0 :event     : event/I                                                *
*Entries :   100000 : Total  Size=     421248 bytes  File Size  =     134514 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   2.85     *
*............................................................................*
*Br    1 :ebeam     : ebeam/F                                                *
*Entries :   100000 : Total  Size=     421248 bytes  File Size  =     260330 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.47     *
*............................................................................*
*Br    2 :px        : px/F                                                   *
*Entries :   100000 : Total  Size=     421194 bytes  File Size  =     359238 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.07     *
*............................................................................*
*Br    3 :py        : py/F                                                   *
*Entries :   100000 : Total  Size=     421194 bytes  File Size  =     359138 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.07     *
*............................................................................*
```

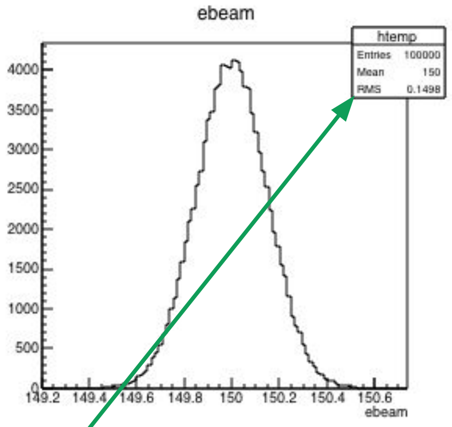
Create pointer to “tree1”

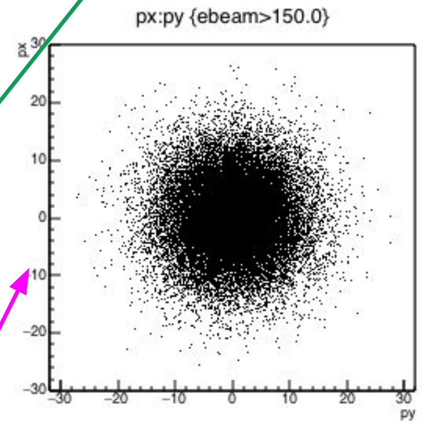
Print structure of tree to screen
This tree contains 7 variables:
`event, ebeam, px, py, pz, zv, chi2`

Turn on statistics box

```
root [4] TCanvas *c2 = new TCanvas("c2", "Tree canvas", 300, 600);
root [5] c2->Divide(1,2);
root [6] c2->cd(1);
root [7] gStyle->SetOptStat(1);
root [8] mytree->Draw("ebeam");
root [9] c2->cd(2);
root [10] mytree->Draw("px:py", "ebeam>150.0");
```

Draw scatter plot (`py` vs `px`) for events with `ebeam` > 150.





ROOT TTree: More about Arguments

- ★ Arguments to many functions in ROOT objects are passed by character strings
- ★ Strings are parsed for both logic and mathematics
- ★ For trees:
  - ○ Any variable in the tree can be manipulated as part of an argument

root [10] mytree->Draw(“px:py”, “ebeam>150.0”, “lego”);

What to draw for each event
- Semicolon “:” indicates adding a new dimension
- Can be functions of variables: e.g. “sqrt(py)”
- Can be combinations of variables: e.g. “ebeam/px : py**2”

Selection cuts: i.e. which events or entries to draw
- Multiple cuts are allowed, combined with C-style logic operators
- Can be functions of variables
- Can be combinations of variables

Drawing options
Options for n-dimensional histograms go here as in previous example

ROOT TTree: Use TTree to fill a histogram

★ Step 1: Define a histogram with a suitable range

root [2] TH1F * h = new TH1F(“hBeamEnergy”, “Beam Energy”, 200, 148.0, 152.0);

★ Step 2: Project the TTree contents into the histogram

root [3] mytree->Project(“hBeamEnergy”, “ebeam”, “px>10.0”);

Project into the NAME of a histogram, not its pointer

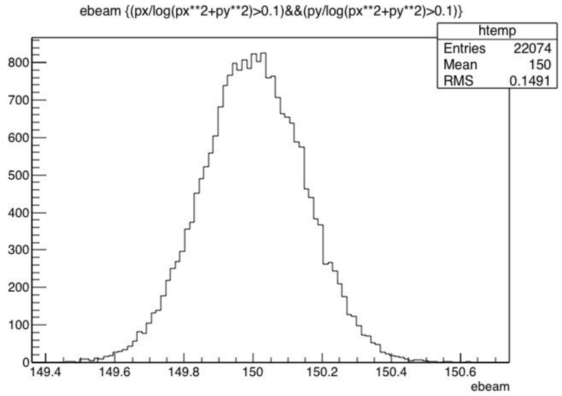
Optional cuts

Variable used to fill the projected histogram.
Make sure the dimensions of your histogram and your projection are the same!

ROOT TTree: Complicated cuts

★ Consider encapsulating your cuts as TCut objects
★ TCut objects can be combined using C-style operators as usual
★ They can be combined with other string cuts

```
root [14] TCut * px_plane = new TCut(“px / log(px*2 + py**2) > 0.10”);
root [15] TCut * py_plane = new TCut(“py / log(px*2 + py**2) > 0.10”);
root [16] mytree->Draw(“ebeam”, *px_plane && *py_plane);
```



Exercises

1. Create a function with parameters, p0 * sin (p1 * x) / x, and also draw it for different parameter values. Set the colour of the parametric function to blue. After having drawn the function, compute for the parameter values (p0 = 1, p1 = 2):
   a. Function value for x=1
   b. Function derivative for x=1
   c. Integral of the function between 0 and 3
2. Suppose you have this set of points defined in the attached file graphdata.txt. Plot these points using the TGraph class. Use as marker point a black box. Looking at the possible options for drawing the TGraph in TGraphPainter, plot a line connecting the points. Make a TGraphError and display it by using the attached data set, graphdata_error.txt, containing error in x and y.
3. Create a one-dimensional histogram with 50 bins between 0 to 10, and fill it with 10000 gaussian distributed random numbers with mean 5 and sigma 2. Plot the histogram and, looking at the documentation in the THistPainter, show in the statistic box the number of entries, the mean, the RMS, the integral of the histogram, the number of underflows, the number of overflows, the skewness and the kurtosis.
4. Using the tree contained in tree.root make a distribution of the total momentum of each whose beam energy was outside of the mean by more than 0.2. Use TCut objects to make your events selections. Project this distribution into a histogram, draw it and save it to a file.

Installing ROOT

- ★ You can install the ROOT's sources from the download area or using directly the Git repository.
- ★ Install using Git repository:

Clone the repo

```
$ git clone https://github.com/root-project/root.git
```

Make a directory for building

```
$ mkdir build
$ cd build
```

Run cmake and make

```
$ cmake ../root
$ make -j8
```

Setup and run ROOT

```
$ source bin/thisroot.sh
$ root
```

Installing ROOT

- ★ You can install the ROOT's sources from the download area or using directly the Git repository.
- ★ Install from the download area: Download the source from http://root.cern.ch/drupal/content/downloading-root

Unpack tar file

```
$ tar zxvf root_6.20.xx.source.tar.gz
```

Create a directory for containing the build

```
$ mkdir root-build
$ cd root-build
```

Execute the cmake command on the shell

```
$ cmake
/Users/sheilamarass/Downloads/root-6.20
.04
```

After CMake has finished running, start the build

```
$ cmake --build .
```

Setup the environment to run

```
$ source
/Users/sheilamarass/root-build/bin/thisr
oot.sh
```

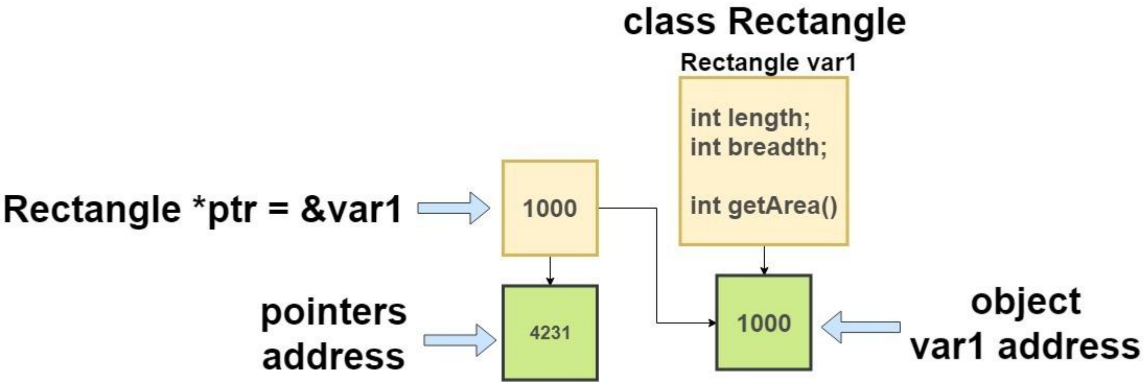
Atart ROOT interactive application

```
$root
```

A Little About C++

Object-Oriented Programming Concepts

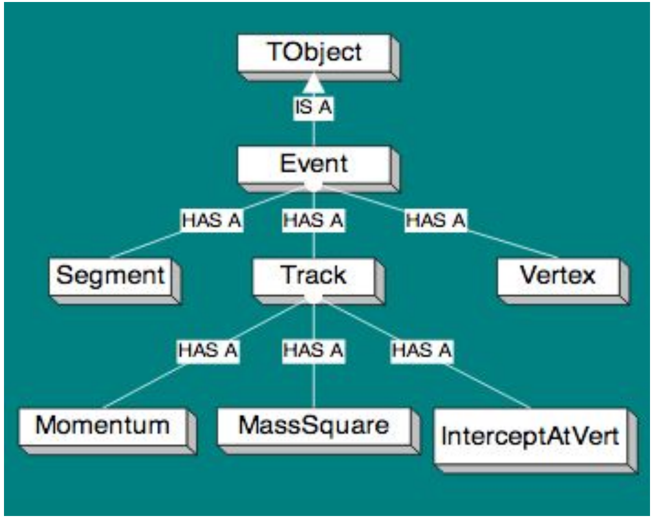
- ★ Classes are an expanded concept of data structures: like data structures, they can contain data members, but they can also contain functions as members
- ★ Object is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable.
- ★ Pointers is a variable that stores the memory address as its value.




```
class class_name{
   access_specifier_1: member1;
   access_specifier_2: member2;
   …
} object_names;
```

Object-Oriented Programming Concepts

- ★ Classes: the description of a “thing” in the system
- ★ Object : instance of a class
- ★ Methods: functions for a class
  - ○ Members: a “has a” relationship to the class
  - ○ Inheritance: an “is a” relationship to the class



The class constructor

- ★ A *constructor* constructs values of the class type. It is a member function whose name is the same as the class name.
- ★ This process involves initializing data members and, frequently, allocating free store using *new*.
- ★ A class constructor will have exact same name as the class and it does not have any return type at all, not even void.

For example: the Graph class (https://root.cern.ch/doc/master/TGraph_8h_source.html)

```
class TGraph : public TNamed, public TAttLine, public TAttFill, public TAttMarker {
...
   public:
         TGraph();
         TGraph(Int_t n);
         TGraph(Int_t n, const Int_t *x, const Int_t *y);
         TGraph(const TGraph &gr);

         virtual Double_t GetErrorX(Int_t bin) const;
         virtual Double_t GetErrorY(Int_t bin) const;
         ….
};
```

Loops: C++

for

```
for (initialization expr; test expr; update expr)
{
    // body of the loop
    // statements we want to execute
}
```

Example:

```
int i=0;
for (i = 1; i <= 10; i++)
{
    printf( "Hello World\n");
}
```

Loops: C++

while

```
initialization expression;
while (test_expression)
{
   // statements
   update_expression;
}
```

Example:

```
int i = 1; // initialization expression

while (i < 6) // test expression
{
    printf( "Hello World\n");

    I++; // update expression
}
```

Loops: C++

do

```
initialization expression;
do
{
   // statements
   update_expression;
} while (test_expression);
```

Example:

```
int i = 2; // Initialization expression

do
{
    printf( "Hello World\n");

     I++; // update expression
}  while (i < 1);   // test expression
```

if … then … else: ROOT

Logical conditions:

A == B (A equal to B)
A != B (A not equal to B)
A && B (condition A and B)
A || B (condition A or B)
A >= B (A greater or equal than B)
A >=B (A greater than B)
A <= B (A less or equal than B)
A < B (A less than B)

```
if (testExpression1)
{
   // statements to be executed if testExpression1 is true
}
else if(testExpression2)
{
   // statements to be executed if testExpression1 is false and testExpression2 is true
}
else if (testExpression 3)
{
   // statements to be executed if testExpression1 and testExpression2 is false and
testExpression3 is true
}
.
.
else
{
   // statements to be executed if all test expressions are false
}
```

Function: C++

A function is a block of code which only runs when it is called

```
type name(parameter1, parameter2, ...)
{
        statements
}
```

- type is the type of the value returned by the function
- name is the identifier by which the function can be called
- parameters (as many as needed): each parameter consists of a type followed by an identifier.
- Statements is the function's body

Void functions are created and used just like value-returning functions except they do not return a value after the functions executes.

& and *
http://www.cplusplus.com/doc/tutorial/functions/

References

- http://webhome.phy.duke.edu/~raw22/public/root.tutorial/basic_root_20100701.pdf
- https://docs.google.com/presentation/d/1nNFRdh483KSYnoaA6q7x0nVeDPbhY7gjWyaGmr0ZdTA/edit#slide=id.g2a0483ea55_3_300