ROOT Summer Student Course

S. Hageboeck, E. Tejedor for the ROOT Team

ROOT
Data Analysis Framework
https://root.cern

Agenda

▶ https://indico.cern.ch/e/ROOTSummer2

Make Sure One of These Works for You!

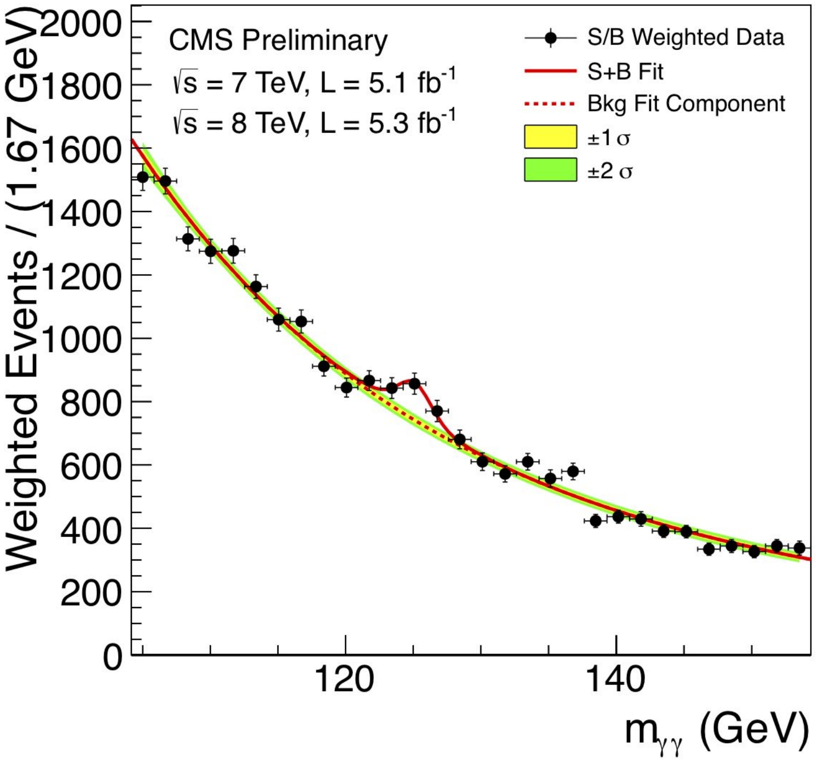
- On Lxplus7/Lxbatch7
  - ssh -XY <username>@lxplus7.cern.ch
  - source /cvmfs/sft.cern.ch/lcg/app/releases/ROOT/6.16.00/x86_64-centos7-gcc48-opt/bin/thisroot.sh
- On SWAN: https://swan.cern.ch
  - The Jupyter Notebook service of CERN
- On your machine (Linux or Mac)
  - Compiled by yourself from sources
  - Using the binaries we distribute
  - See https://root.cern/releases

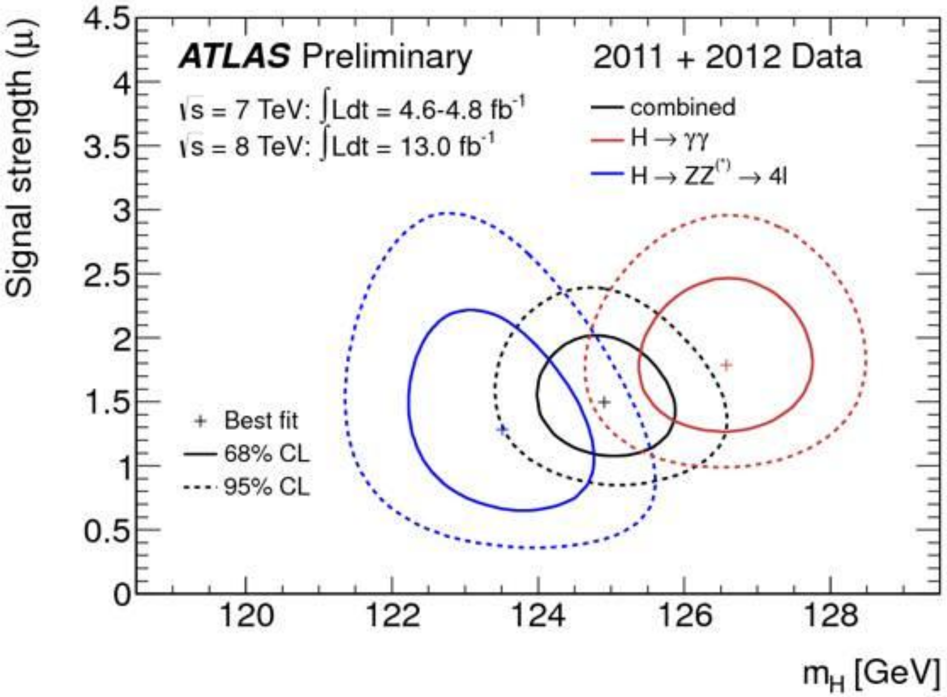
Note: ROOT on Windows is in beta mode.

Introduction

A Quick Tour of ROOT

What can you do with ROOT?





ROOT in a Nutshell

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

LGPL 3 Free Software

An Open Source Project

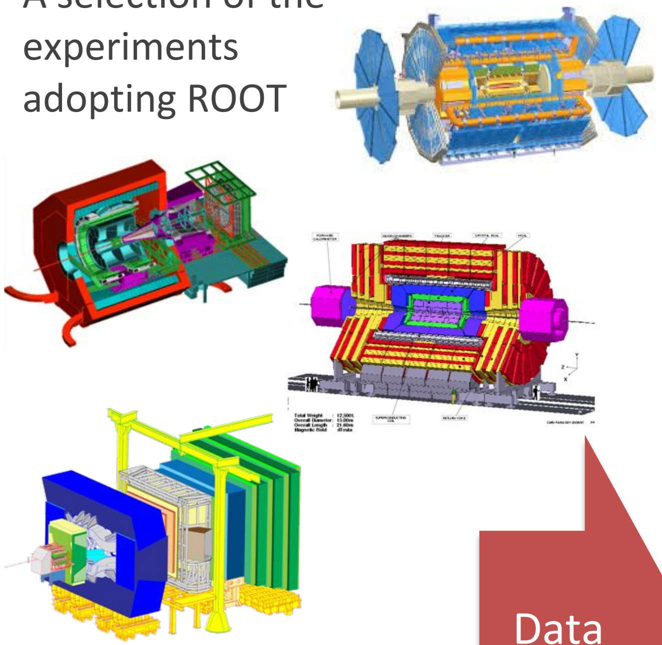
Unstar 595 Fork 433

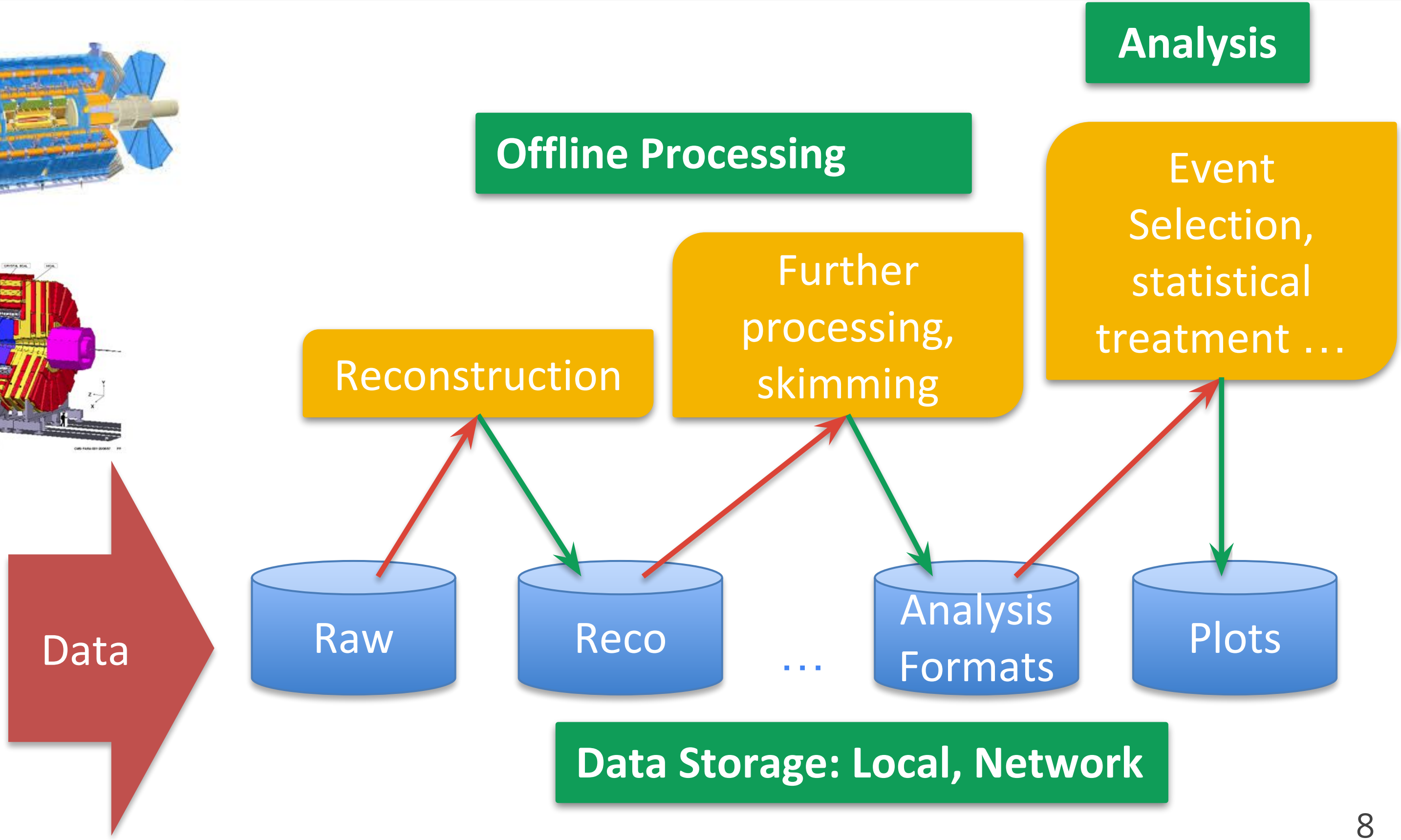
176 contributors

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

ROOT Application Domains

A selection of the experiments adopting ROOT





Event Filtering

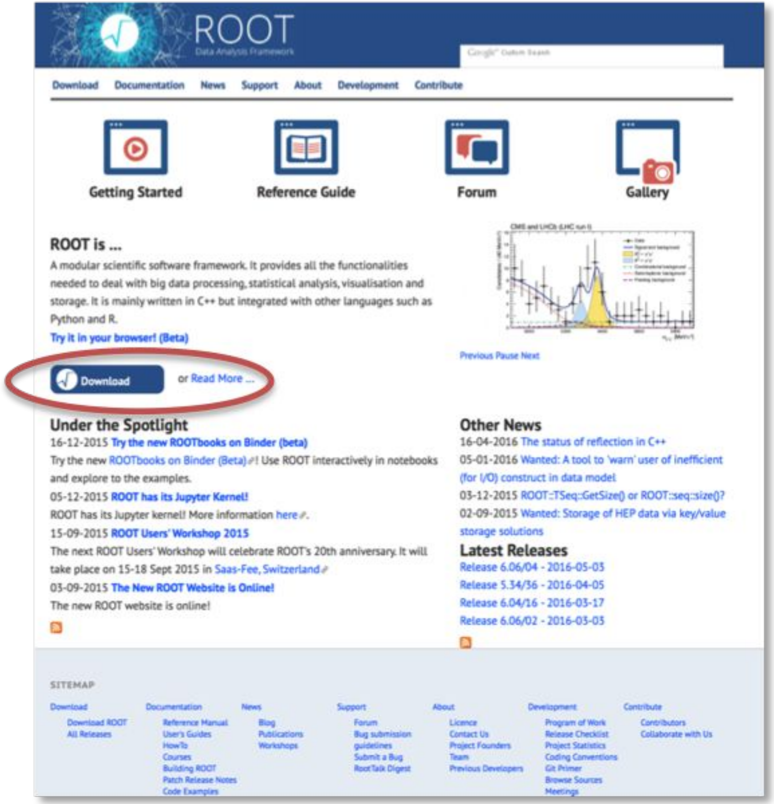
LHC Data in ROOT Format

~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

From Sources

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*

The ROOT Prompt and Macros

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”

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
```

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

Ex Tempore Exercise

- Fire up ROOT
- Verify it works as a calculator
- List the files in /etc from within the ROOT prompt
- Inspect the help
- Quit

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

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

Advanced Users

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

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

Time For Exercises

- Exercises: https://github.com/root-project/training/tree/master/SummerStudentCourse/2019/Exercises/C++Interpreter
- You like don’t have time to complete all

The 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

How It Looks Like

Text

Code

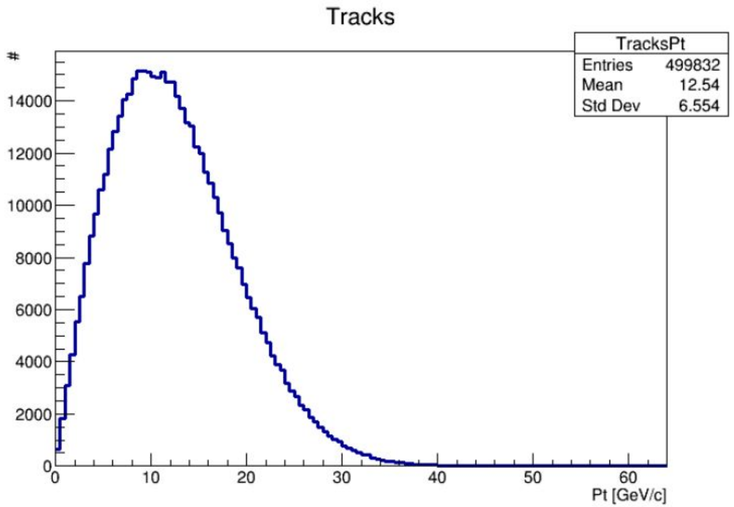
Graphics

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()
```



Use Notebooks at CERN

- **SWAN**: **S**ervice for **W**eb based **AN**alysis

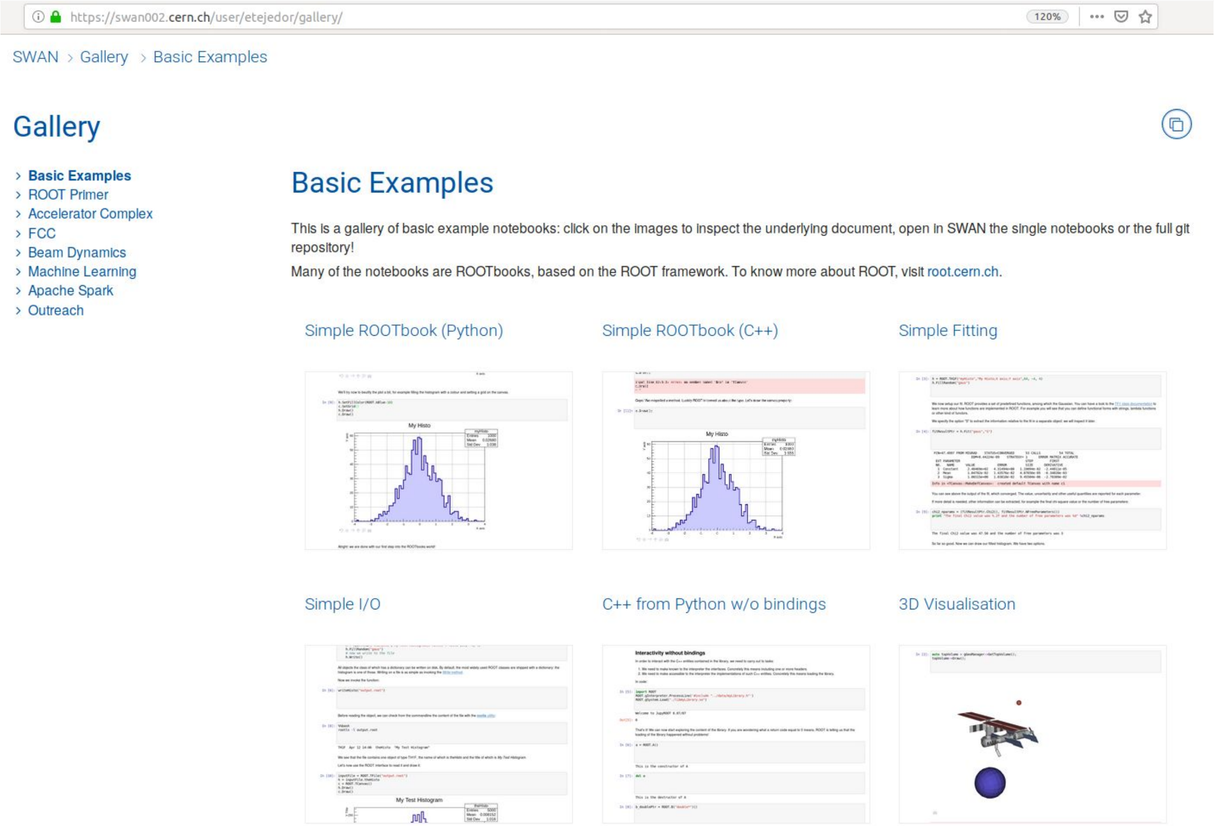
- Get a CERNBox (if you don't have one)
  - Visit https://cernbox.cern.ch
- Log in to https://swan.cern.ch
- Create a project and a C++ notebook
  - Type in some code
  - Run it
  - Create markdown cells

Notebooks On Your Machine

- Possible to install Jupyter as a package
- Fire up with the *`root --notebook`* command

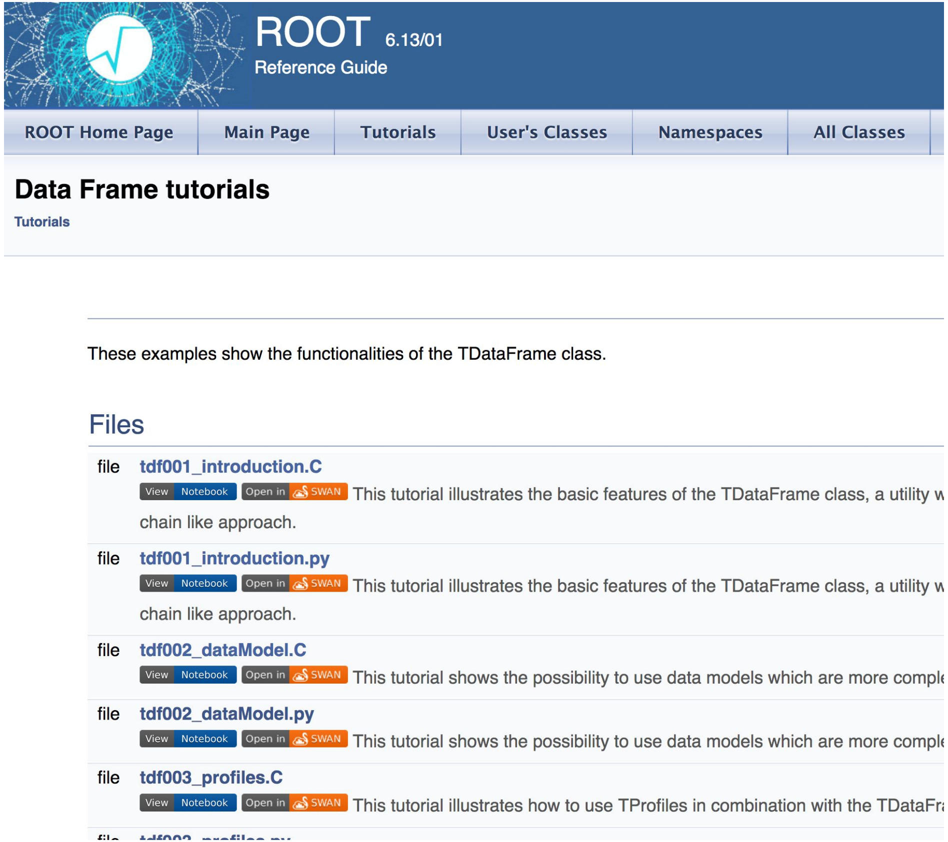
Examples

Open the gallery section in SWAN!



More Examples

https://root.cern/doc/master/group__Tutorials.html



ROOT 6.13/01
Reference Guide

ROOT Home Page | Main Page | Tutorials | User's Classes | Namespaces | All Classes

Data Frame tutorials

Tutorials

These examples show the functionalities of the TDataFrame class.

Files

file **tdf001_introduction.C**
View Notebook | Open in SWAN This tutorial illustrates the basic features of the TDataFrame class, a utility w
chain like approach.

file **tdf001_introduction.py**
View Notebook | Open in SWAN This tutorial illustrates the basic features of the TDataFrame class, a utility w
chain like approach.

file **tdf002_dataModel.C**
View Notebook | Open in SWAN This tutorial shows the possibility to use data models which are more comple

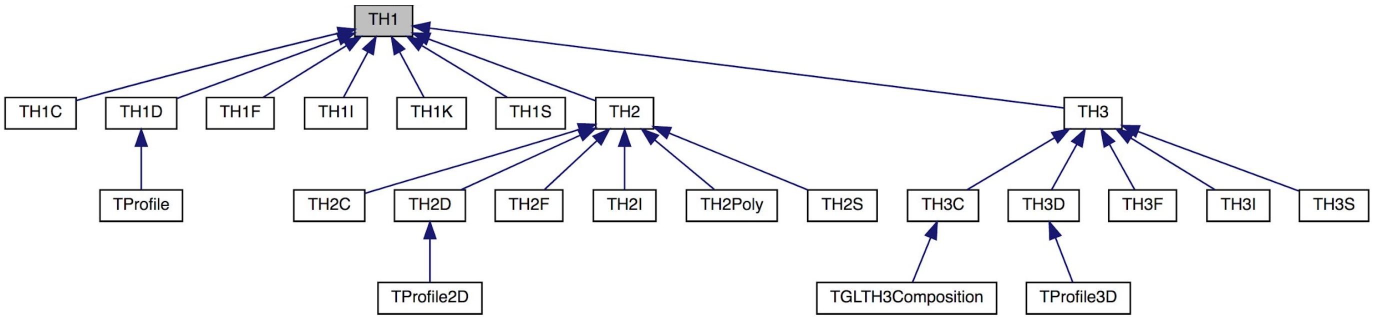
file **tdf002_dataModel.py**
View Notebook | Open in SWAN This tutorial shows the possibility to use data models which are more comple

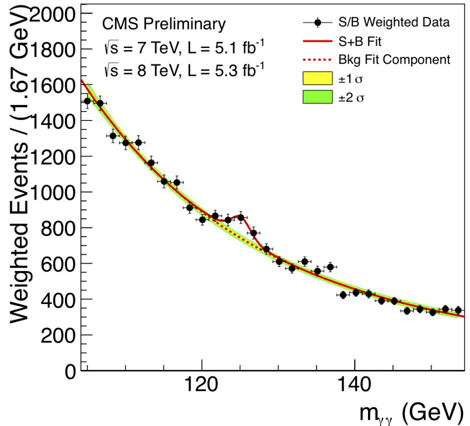
file **tdf003_profiles.C**
View Notebook | Open in SWAN This tutorial illustrates how to use TProfiles in combination with the TDataFr

Histograms, Graphs and Functions

Histograms

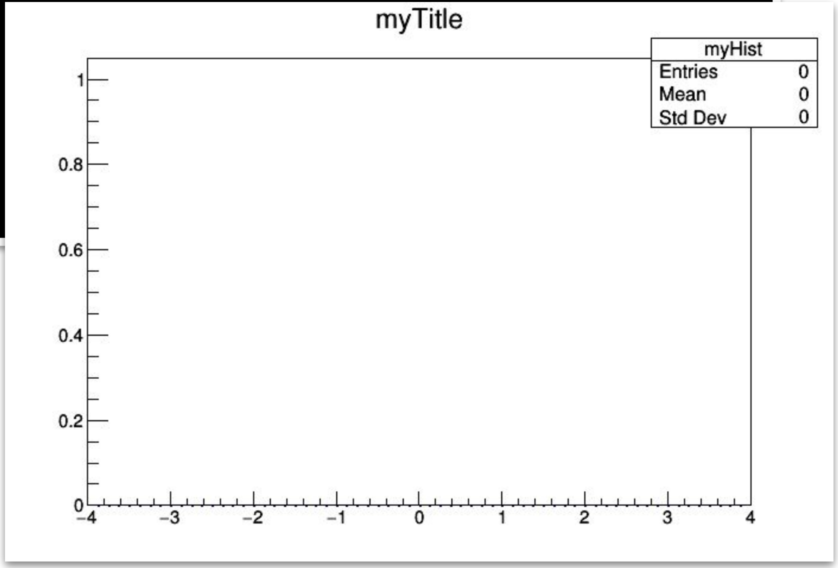
- Simplest form of data reduction
  - Can have billions of collisions, the Physics displayed in a few histograms
  - Possible to calculate momenta: mean, rms, skewness, kurtosis ...
- Collect quantities in discrete categories, the bins
- ROOT Provides a rich set of histogram types
  - We'll focus on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





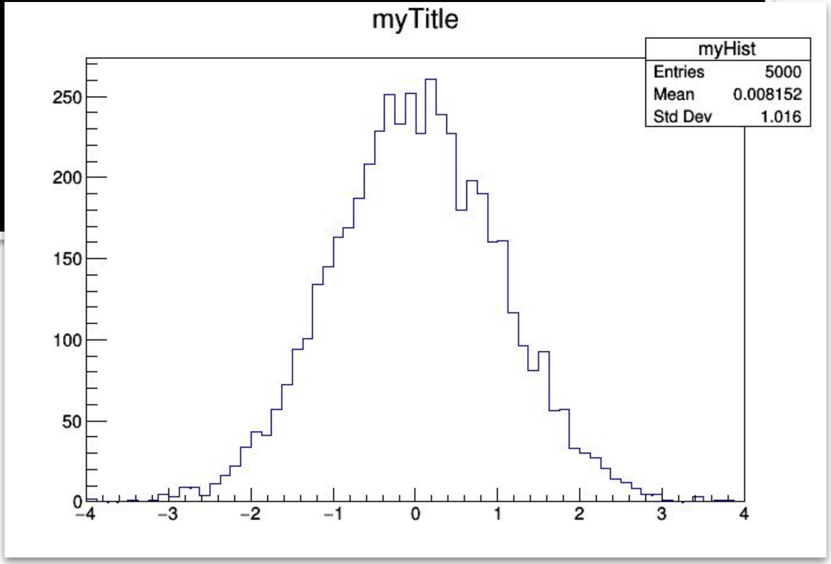
My First Histogram

```
root [0] TH1F h("myHist", "myTitle", 64, -4, 4)
root [1] h.Draw()
```



My First Histogram

```
root [0] TH1F h("myHist", "myTitle", 64, -4, 4)
root [1] h.FillRandom("gaus")
root [2] h.Draw()
```



Interlude: Scope

Bad for graphics:

```
// makeHist.C:
void makeHist() {
  TH1F hist("hist", "My Histogram");
  hist.Draw(); // shows histogram
}
```

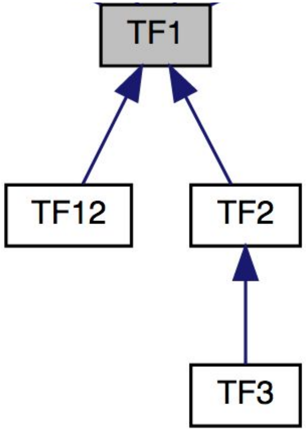
ROOT doesn't show my histogram!

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)*):

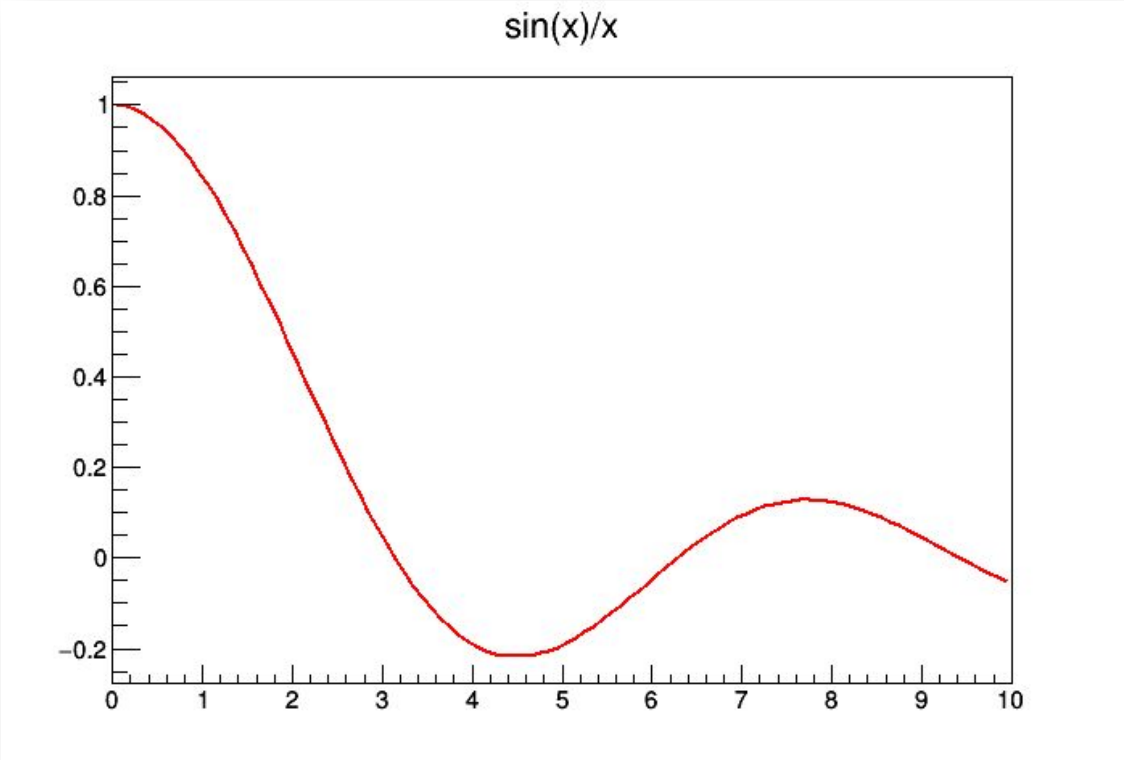
```
root [0] TF1 f1("f1","sin(x)/x",0.,10.); //name,formula,min,max
root [1] f1.Draw();
```

- 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();
```

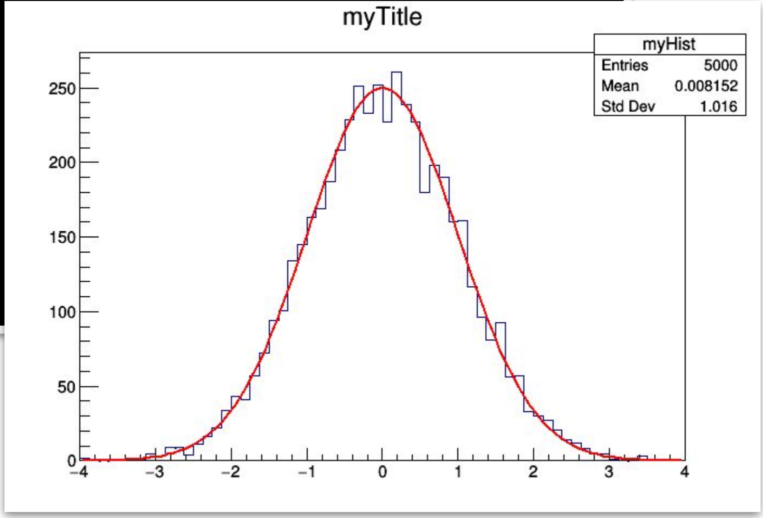
Try it!

ROOT as a Function Plotter

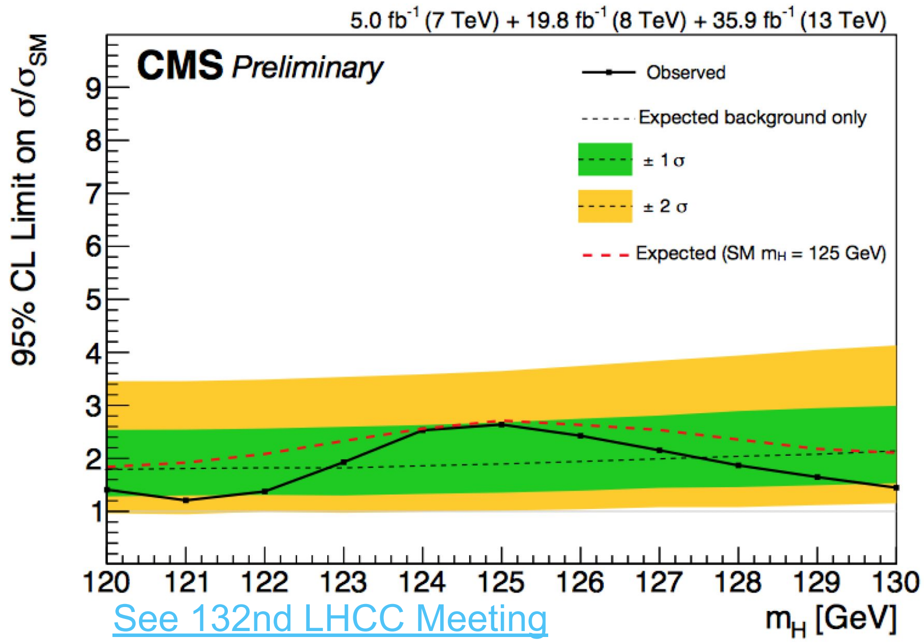


Another Example

```
root [0] TH1F h("myHist", "myTitle", 64, -4, 4)
root [1] h.FillRandom("gaus")
root [2] h.Draw()
root [3] TF1 f("g", "gaus", -8, 8)
root [4] f.SetParameters(250, 0, 1)
root [5] f.Draw("Same")
```



Graphs

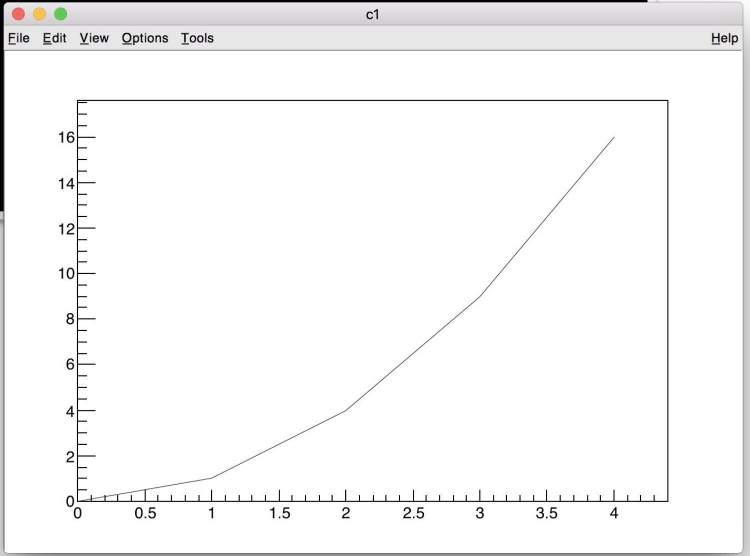


See 132nd LHCC Meeting

- Display points and errors
- Not possible to calculate momenta
- Not a data reduction mechanism
- **Fundamental to display trends**
- Focus on TGraph and TGraphErrors classes in this course

My First Graph

```
root [0] TGraph g;
root [1] for (auto i : {0,1,2,3,4}) g.SetPoint(i,i,i*i)
root [2] g.Draw("APL")
```

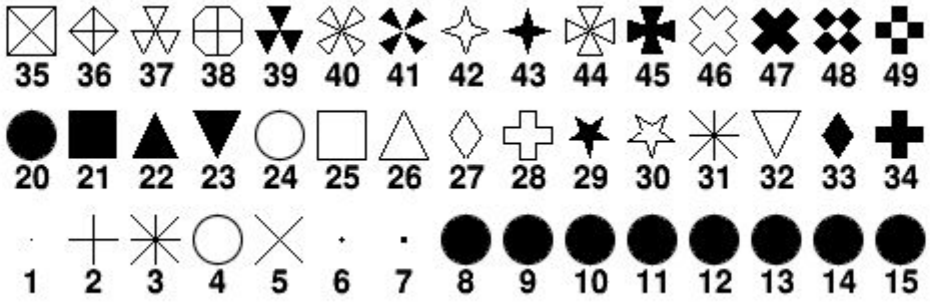


Creating a Nice Plot: Survival Kit

Credit: https://www.swissknifeshop.com

The Markers

From the TAttMarker documentation:
https://root.cern.ch/doc/master/classTAttMarker.html

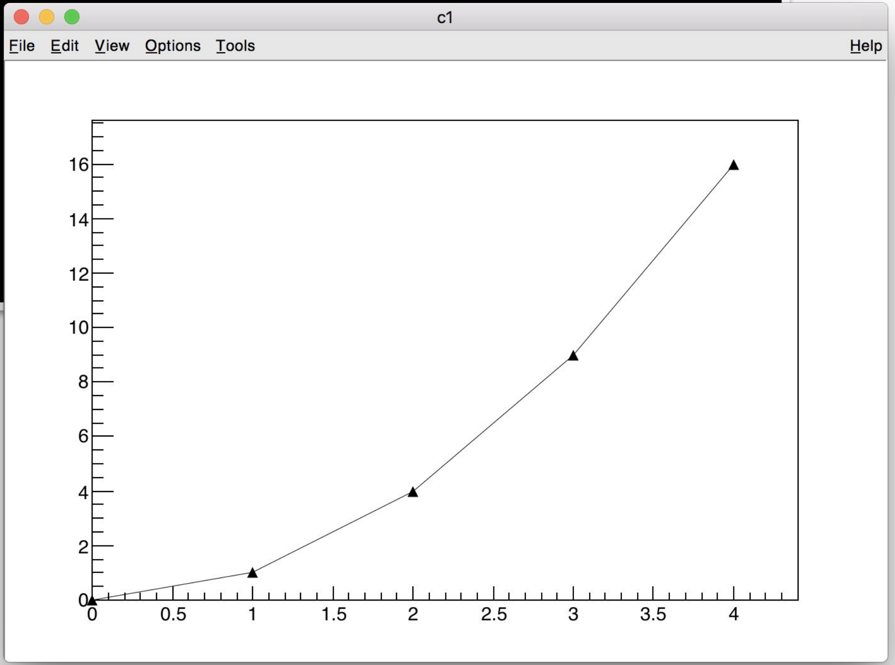



```
kDot=1, kPlus, kStar, kCircle=4, kMultiply=5,
kFullDotSmall=6, kFullDotMedium=7, kFullDotLarge=8,
kFullCircle=20, kFullSquare=21, kFullTriangleUp=22,
kFullTriangleDown=23, kOpenCircle=24, kOpenSquare=25,
kOpenTriangleUp=26, kOpenDiamond=27, kOpenCross=28,
kFullStar=29, kOpenStar=30, kOpenTriangleDown=32,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

Also available through more friendly names 😊

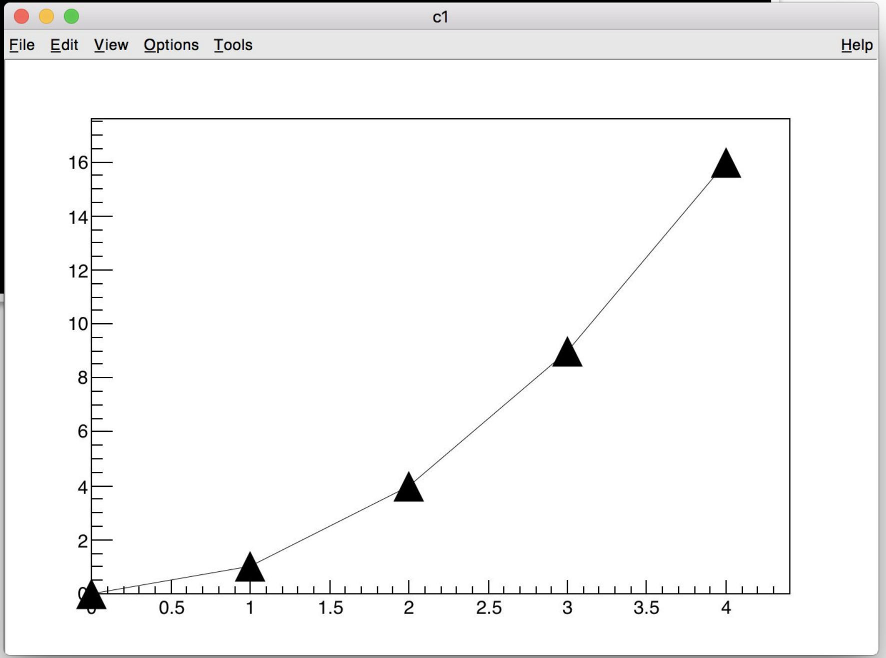
My First Graph

```
root [3] g.SetMarkerStyle(kFullTriangleUp)
```

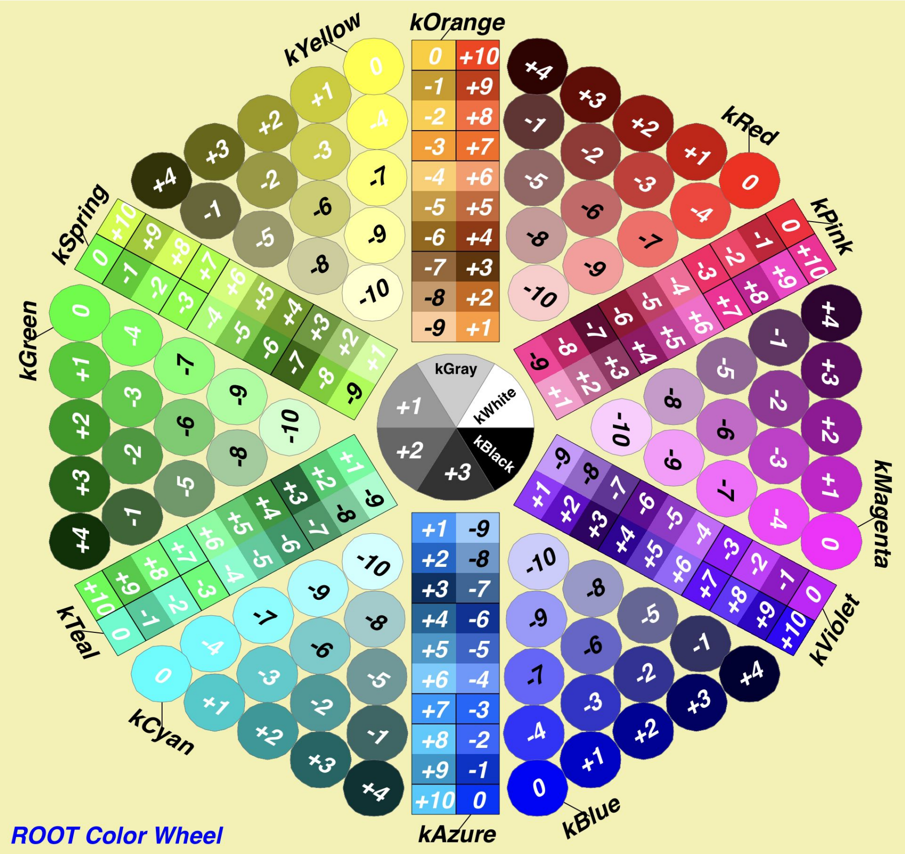


My First Graph

```
root [3] g.SetMarkerStyle(kTriangleUp)
root [4] g.SetMarkerSize(3)
```

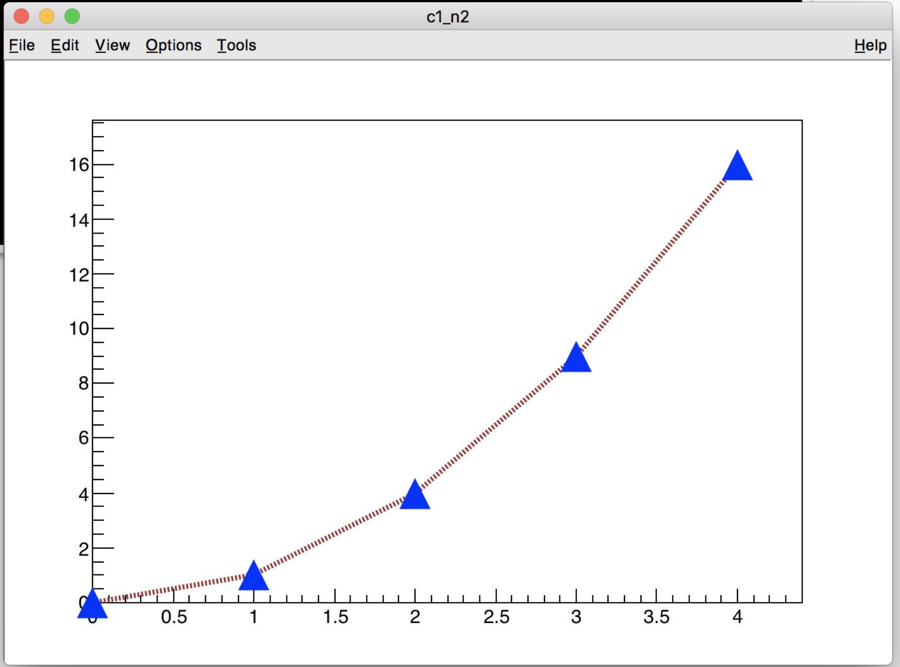


The Colors (TColorWheel)



My First Graph

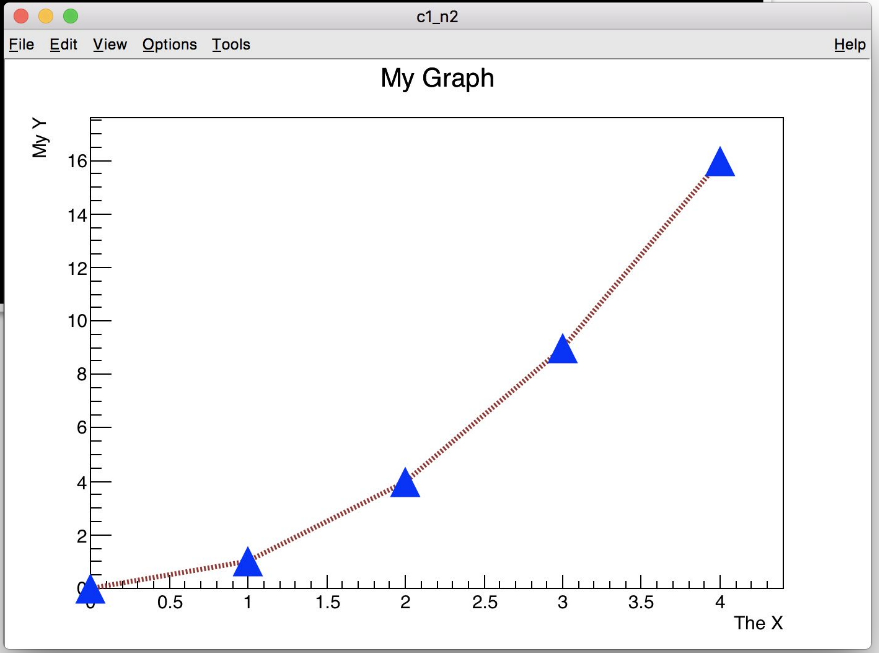
```
root [5] g.SetMarkerColor(kAzure)
root [6] g.SetLineColor(kRed - 2)
root [7] g.SetLineWidth(2)
root [8] g.SetLineStyle(3)
```



Question:
How do you find information on line styles?

My First Graph

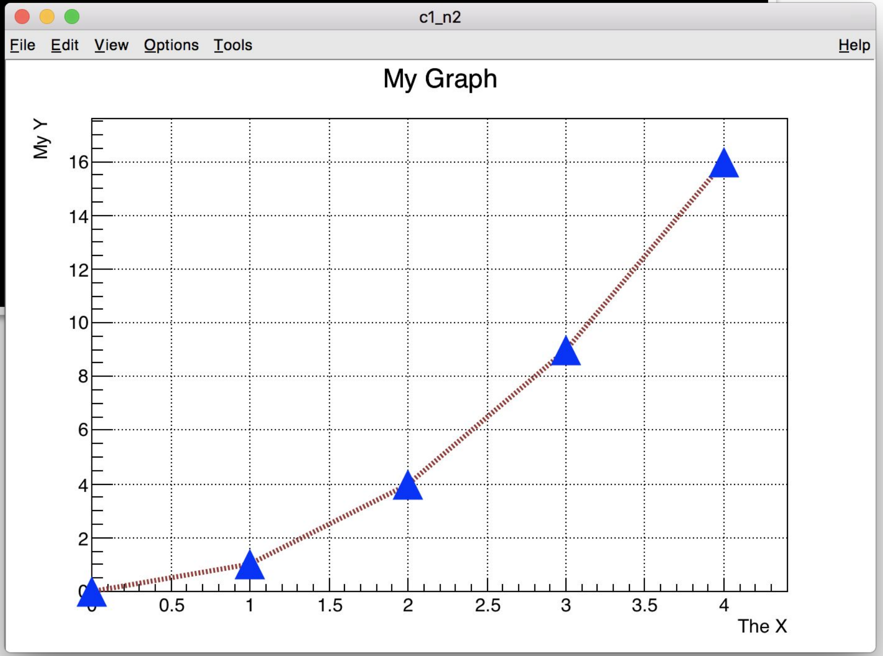
```
root [9] g.SetTitle("My Graph;The X;My Y")
```



Shortcut:
Directly set titles for x and y axis.

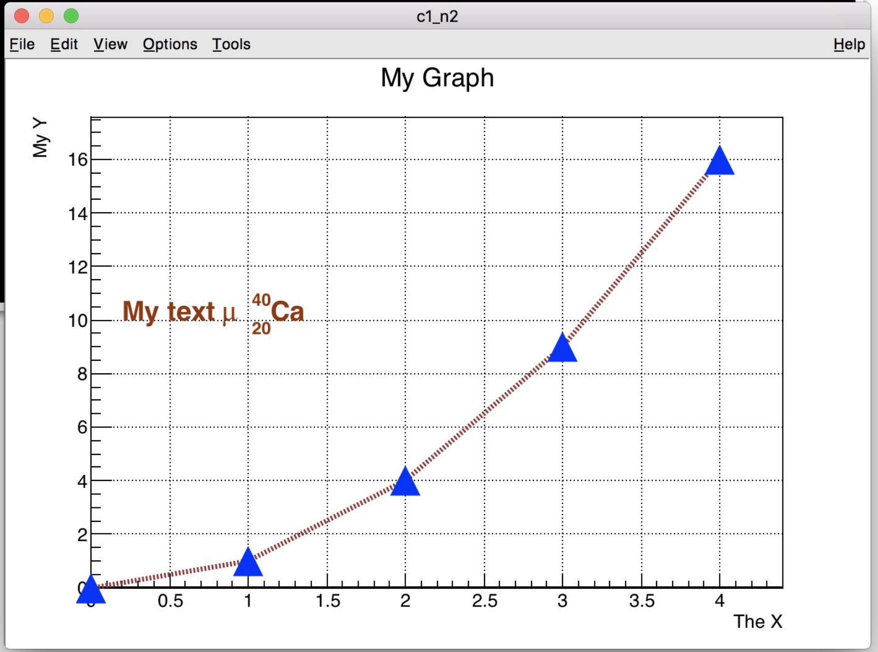
My First Graph

```
root [10] gPad->SetGrid()
```



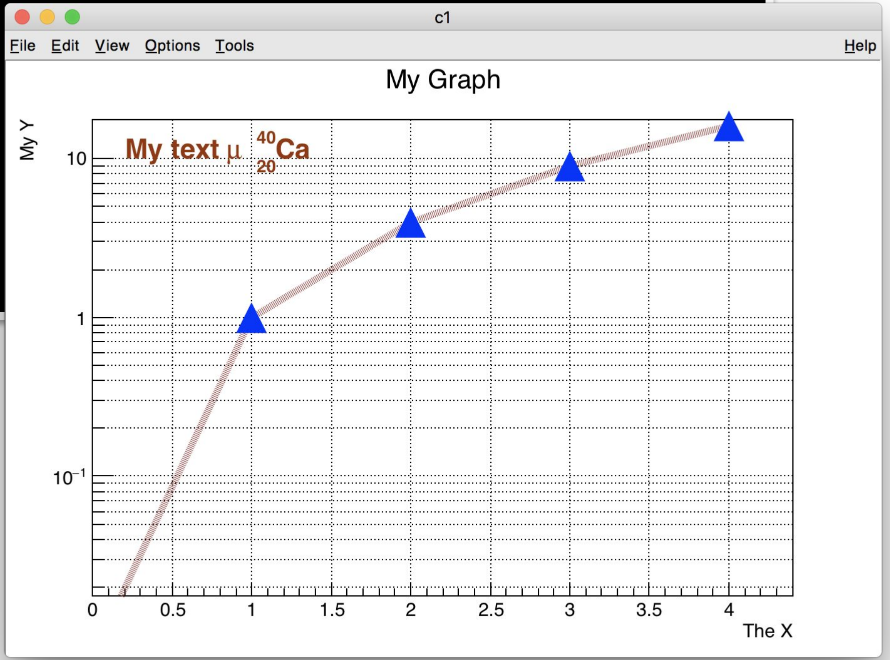
My First Graph

```
root [10] auto txt = "#color[804]{My text #mu {}^{40}_{20}Ca}"
root [11] TLatex l(.2, 10, txt)
root [12] l.Draw()
```



My First Graph

```
root [13] gPad->SetLogy();
```



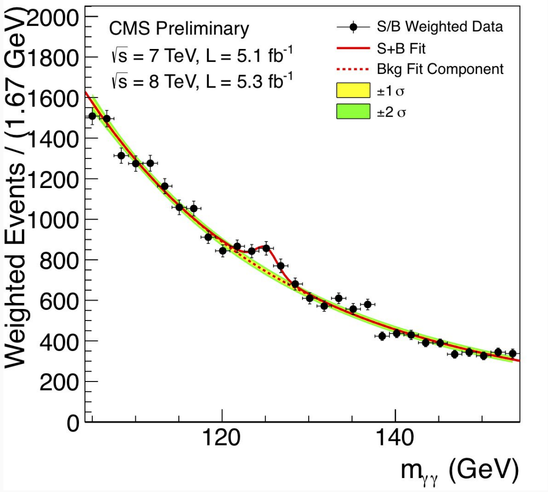
Time for Exercises!

https://github.com/root-project/training/tree/master/SummerStudentCourse/2019/Exercises/HistogramsGraphsFunctions

Parameter Estimation and Fitting

What is Fitting ?

- Estimate parameters of a hypothetical distribution from the observed data distribution
  - y = f (x | θ) is the fit model function
- Find the best estimate of the parameters θ assuming f (x | θ)
- Both Likelihood and Chi2 fitting are supported in ROOT



Example

Higgs ➞ γγ spectrum

We can fit for:

- the expected number of Higgs events
- the Higgs mass

Fitting in ROOT:

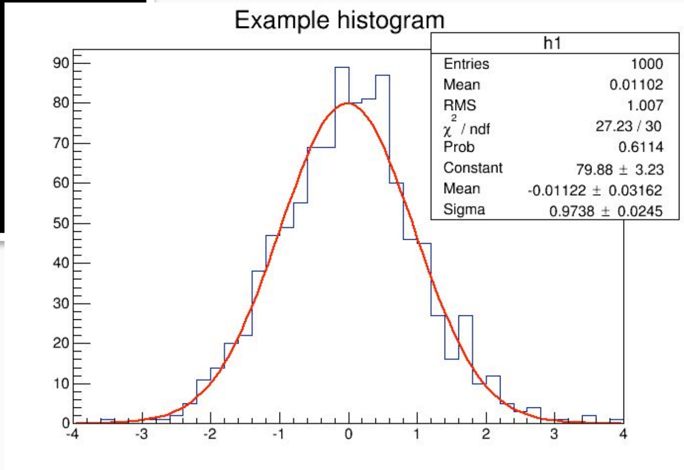
- Create first a parametric function object, **`TF1`**, which represents our model
  - need to set the initial values of the function parameters.
- Fit the data object (Histogram or Graph):
  - Call the **`Fit`** method passing the function object
  - various options are possible (see the **`TH1::Fit`** documentation)
- Examine result:
  - get parameter values, uncertainties, correlation
  - get fit quality estimation
- The resulting fit function is also drawn automatically on top of the Histogram or the Graph when calling **`TH1::Fit`** or **`TGraph::Fit`**

Fitting Histograms

- We have a histogram, h1, and we want to fit it:

```
root [0] TF1 f1("f1","gaus");
root [1] h1.Fit(&f1);

 FCN=27.2252 FROM MIGRAD    STATUS=CONVERGED      60 CALLS          61 TOTAL
                     EDM=1.12393e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     7.98760e+01   3.22882e+00   6.64363e-03  -1.55477e-05
   2  Mean        -1.12183e-02   3.16223e-02   8.18642e-05  -1.49026e-02
   3  Sigma        9.73840e-01   2.44738e-02   1.69250e-05  -5.41154e-03
```



For displaying the fit parameters:

```
gStyle->SetOptFit(1111);
```

Creating the Fit Function

- How to create the parametric function object (**TF1**) :
  - we can write formula expressions using functions:

```
TF1 f1("f1","[0]*TMath::Gaus(x,[1],[2])");
```

    - we can use the available functions in ROOT library and stl
    - **[0],[1],[2] indicate the parameters.**
    - We could also use meaningful names, like [a],[mean],[sigma]
  - There are pre-defined functions

```
TF1("f1","gaus");
```

  - pre-defined functions available: *gaus, expo, landau, breitwigner,crystal_ball,pol{0,1..,10}, cheb{0,1},xygaus,xylanday,bigaus*

PyROOT: The ROOT Python Bindings

PyROOT

- Python bindings for ROOT
- Access all the ROOT C++ functionality from Python
  - Benefit from C++ performance
- Dynamic, automatic
- "Pythonisations" for specific cases

Using PyROOT

- Entry point to use ROOT from within Python

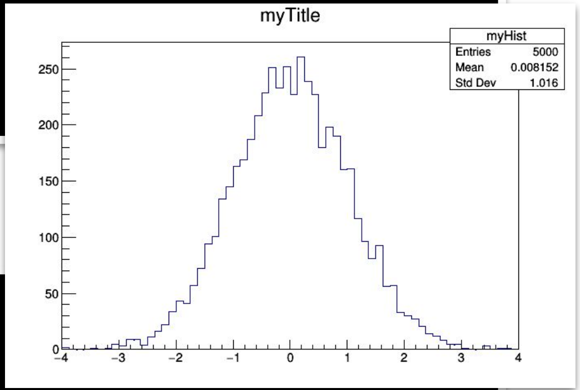
```
import ROOT
```

- All the ROOT classes you have learned so far can be accessed from Python

```
ROOT.TH1F
ROOT.TGraph
...
```

Example: C++ to Python

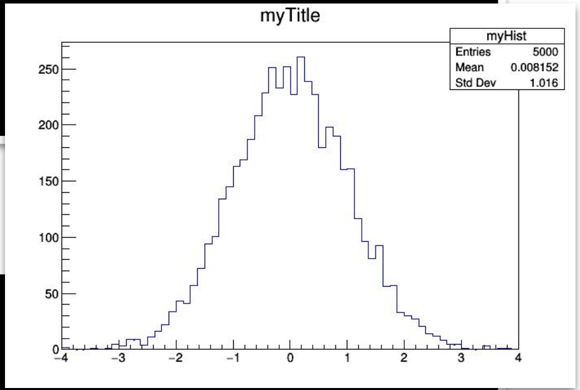
```
> root
root [0] TH1F h("myHist", "myTitle", 64, -4, 4)
root [1] h.FillRandom("gaus")
root [2] h.Draw()
```




```
> python
>>> import ROOT
>>> h = ROOT.TH1F("myHist", "myTitle", 64, -4, 4)
>>> h.FillRandom("gaus")
>>> h.Draw()
```

Example: C++ to Python

```
> root
root [0] TH1F h("myHist", "myTitle", 64, -4, 4)
root [1] h.FillRandom("gaus")
root [2] h.Draw()
```



also with individual import

```
> python
>>> from ROOT import TH1F
>>> h = TH1F("myHist", "myTitle", 64, -4, 4)
>>> h.FillRandom("gaus")
>>> h.Draw()
```

Time For Exercises

https://github.com/root-project/training/tree/master/SummerStudentCourse/2019/Exercises/PythonInterface

- In order to run the exercises:
  - Use the Python prompt

    ```
    > python
    >>>
    ```

  - Run a Python script

    ```
    > python -i myscript.py
    ```

  - Use SWAN
    - Create a canvas before drawing: `c = ROOT.TCanvas()`
    - Run `c.Draw()` at the end to see the plot

Reading and Writing Data

The ROOT File

- In ROOT, objects are written in files*, represented by **TFile** instances
- TFiles are *binary* and can be compressed (transparently for the user)
- **TFiles are self-descriptive**:
  - The information how to retrieve objects from a file is stored with the objects

* this is an understatement - we'll not go into the details in this course!

TFile in Action

```
TFile f("myfile.root", "RECREATE");
```

Option	Description
NEW or CREATE	Create a new file and open it for writing, if the file already exists the file is not opened.
RECREATE	Create a new file, if the file already exists it will be overwritten.
UPDATE	Open an existing file for writing. If no file exists, it is created.
READ	Open an existing file for reading (default).

TFile in Action: Writing

```cpp
TFile f("myfile.root", "RECREATE");
TH1F h("h", "h", 64, 0, 8);
h.Write("h");
f.Close();
```

- Write to a file
- Close the file and make sure the operation succeeded

```
> rootls -l myfile.root
TH1F  Jun 24 15:02 2019 h  "h"
```

TFile in Action: Reading

```cpp
TH1F* myHist;
TFile f("file.root");
f.GetObject("h", myHist);
myHist->Draw();
```

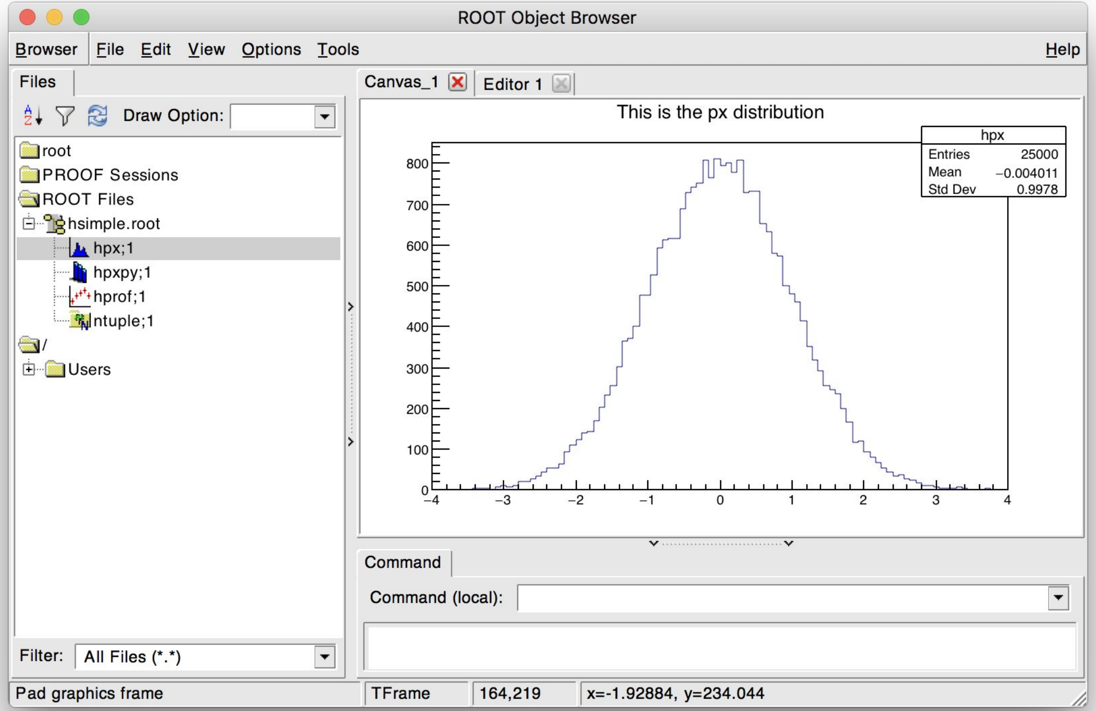
C++

Python

```python
import ROOT
f = ROOT.TFile("file.root")
f.h.Draw()
```

Get the histogram by name!
Possible only in Python

Listing TFile Content



- ***TBrowser*** interactive tool
- ***rootls*** tool: list content
- ***TFile::ls()***: prints content
  - Great for interactive usage

Time For Exercises

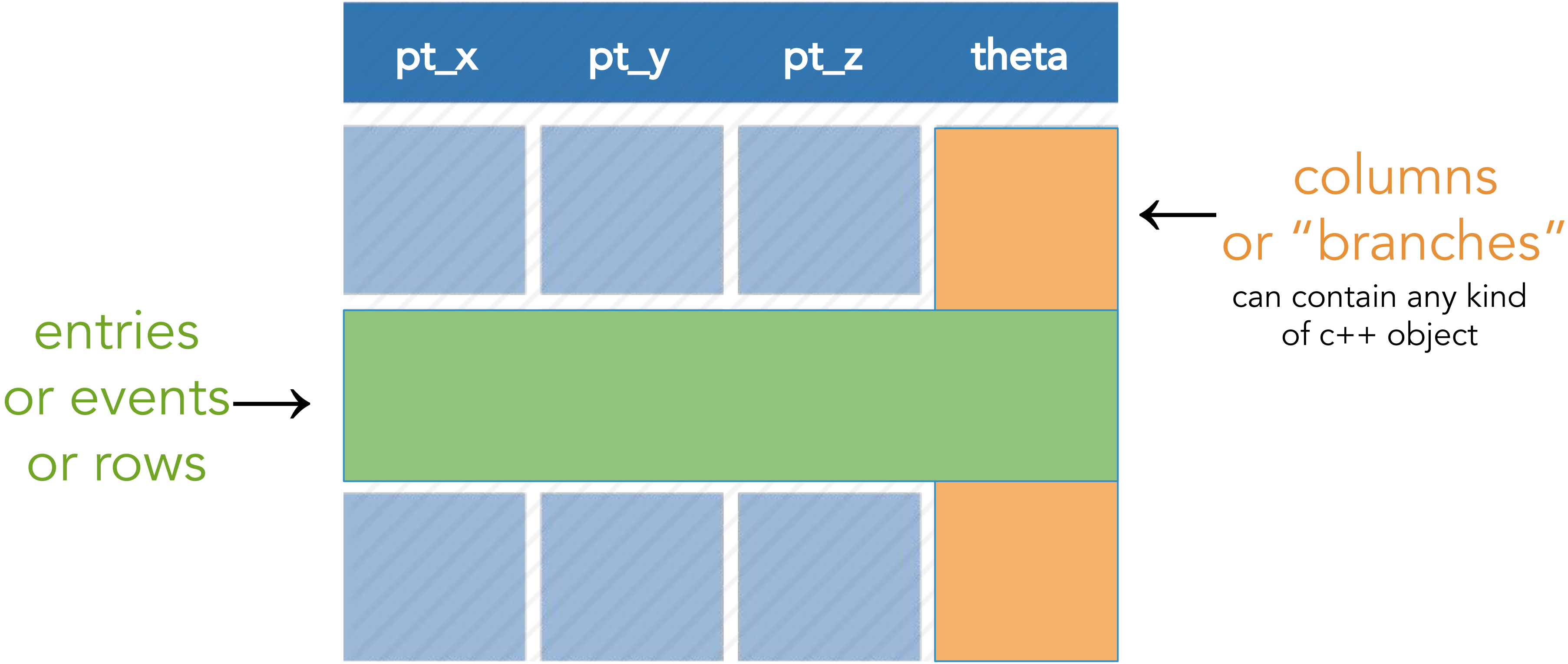
https://github.com/root-project/training/tree/master/SummerStudentCourse/2019/Exercises/WorkingWithFiles

The ROOT Columnar Format

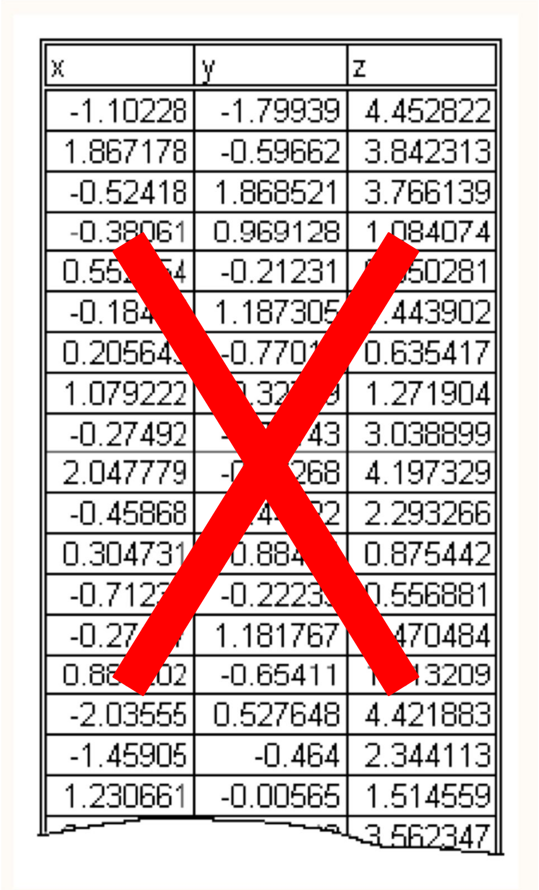
Columns and Rows

- High Energy Physics: many statistically independent *collision events*
- Create an event class, serialise and write out N instances into a file?
  → No. Very inefficient!
- Organise the dataset in **columns**

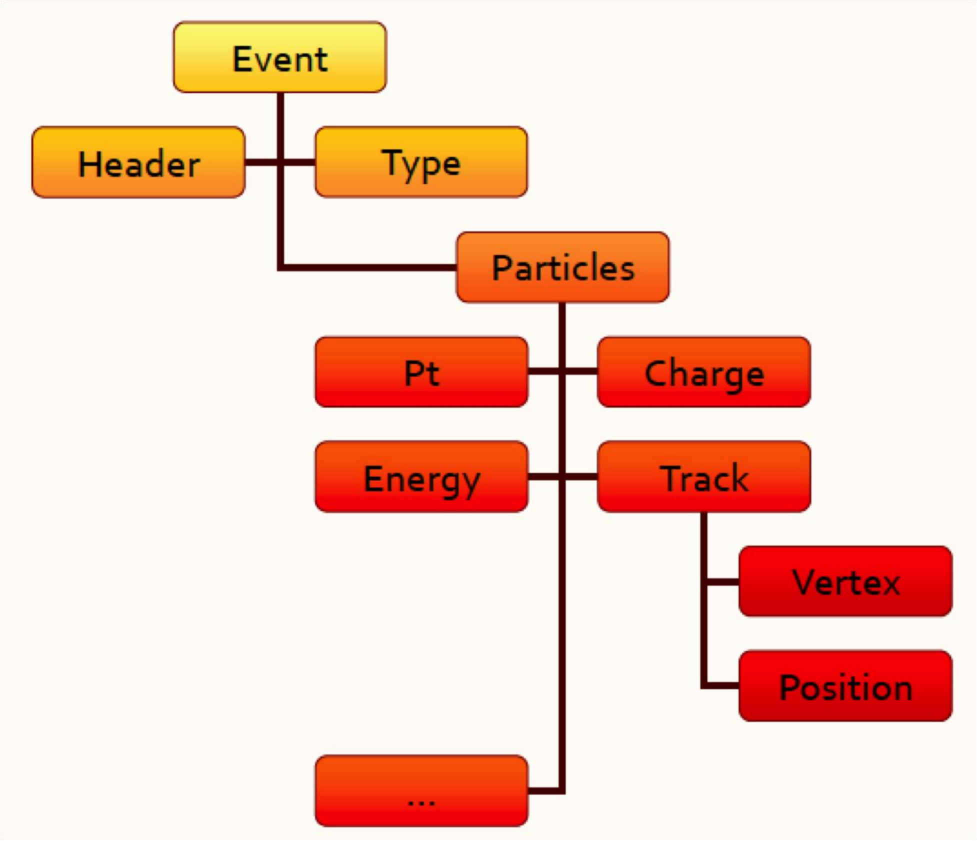
Columnar Representation



Relations Among Columns



x	y	z
-1.10228	-1.79939	4.452822
1.867178	-0.59662	3.842313
-0.52418	1.868521	3.766139
[illegible]	0.969128	[illegible]
[illegible]	-0.21231	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	0.635417
1.079222	[illegible]	1.271904
-0.27492	[illegible]	3.038899
2.047779	[illegible]	4.197329
-0.45868	[illegible]	2.293266
0.304731	[illegible]	0.875442
[illegible]	[illegible]	[illegible]
[illegible]	1.181767	[illegible]
[illegible]	-0.65411	[illegible]
-2.03555	0.527648	4.421883
-1.45905	-0.464	2.344113
1.230661	-0.00565	1.514559
[illegible]	[illegible]	3.562347



The TTree

A columnar dataset in ROOT is represented by the class **TTree**:

- Also called *tree*, columns also called *branches*
- Columns can contain different types.
- **Support any type of object**
- One row per *entry* (or, in collider physics, *event*)

If just a **single number** per column is required, the simpler **TNtuple** *can* be used.

A modern and simple way to interact with ROOT datasets is to use `RDataFrame`

- Low-level interfaces to deal with datasets do exist but are beyond the scope of this course

RDataFrame: quick how-to

1. build a data-frame object by specifying your data-set
2. apply a series of transformations to your data
   - filter (e.g. apply some cuts) or
   - define new columns
3. apply actions to the transformed data to produce results (e.g. fill a histogram)

Simple Code Example

1. Build RDataFrame

```
ROOT::RDataFrame d("t", "f.root");
auto h = d.Filter("theta > 0").Histo1D("pt");
h->Draw();
```

2. Cut on theta

3. Fill histogram with pt

Filling multiple histograms

```
auto h1 = d.Filter("theta > 0").Histo1D("pt");
auto h2 = d.Filter("theta < 0").Histo1D("pt");
h1->Draw();          // event loop is run lazily once here
h2->Draw("SAME"); // no need to run loop again here
```

Book all your actions upfront. The first time a result is accessed, RDataFrame will fill all booked results.

More on histograms

Expert Feature

```
auto h = d.Histo1D({"myName","Title;x",10,0.,1.},
"x");
```

You can specify a model histogram with

- a name and a title
- a predefined axis range

Here, the histogram is created with 10 bins ranging from 0 to 1, and the axis is labelled “x”.

Define a new column

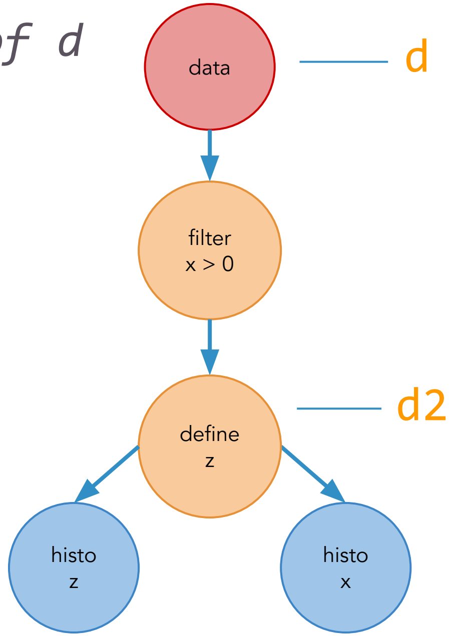
```
double m = d.Filter("x > y")
            .Define("z", "sqrt(x*x + y*y)")
            .Mean("z");
```

`Define` takes the name of the new column and its expression. Later you can use the new column as if it was present in your data.

Think of your analysis as data-flow

```
// d2 is a new data-frame, a transformed version of d
auto d2 = d.Filter("x > 0")
           .Define("z", "x*x + y*y");

// make multiple histograms out of it
auto hz = d2.Histo1D("z");
auto hx = d2.Histo1D("x");
```



You can store transformed data-frames in variables, then use them as you would use a RDataFrame.

Cutflow reports

```
d.Filter("x > 0", "xcut")
 .Filter("y < 2", "ycut");
d.Report()->Print();
```

```
// output
xcut         : pass=49           all=100          --   49.000 %
ycut         : pass=22           all=49           --   44.898 %
```

When called on the main RDF object, `Report` prints statistics for all filters *with a name*

Saving data to file

```
auto new_df = df.Filter("x > 0")
                .Define("z", "sqrt(x*x + y*y)")
                .Snapshot("tree", "newfile.root");
```

We filter the data, add a new column, and then save everything to file. No boilerplate code at all.

Using callables instead of strings

Expert Feature

```
// define a c++11 lambda - an inline function - that checks "x>0"
auto IsPos = [](double x) { return x > 0.; };
// pass it to the filter together with a list of branch names
auto h = d.Filter(IsPos, {"theta"}).Histo1D("pt");
h->Draw();
```

any callable (function, lambda, functor class) can be used as a filter, as long as it returns a boolean

RDataFrame: declarative analyses

```
ROOT::RDataFrame d("treename", "file.root");
auto h = d.Filter(IsGoodEntry, {"x","y"})
          .Histo1D("x");
```

A function taking 2 values in input, returns a boolean

- full control over *the analysis*
- no boilerplate
- common tasks are already implemented

? parallelization is not trivial?

RDataFrame: parallelism

```
ROOT::EnableImplicitMT();
ROOT::RDataFrame d("treename", "file.root");
auto h = d.Filter(IsGoodEntry, {"x","y"})
          .Histo1D("x");
```

- full control over *the analysis*
- no boilerplate
- common tasks are already implemented

? parallelization is not trivial?

C++ / JIT / PyROOT

C++ and just-in-time compiled code

```
d.Filter("th > 0").Snapshot("t","f.root","pt*");
```

PyROOT -- just leave out the `;`

```
d.Filter("th > 0").Snapshot("t","f.root","pt*")
```

Time For Exercises

https://github.com/root-project/training/tree/master/SummerStudentCourse/2019/Exercises/WorkingWithColumnarData

Wrap up