ROOT Basic Course

E. Tejedor, D. Piparo for the ROOT Team

ROOT

Data Analysis Framework

https://root.cern

This Course

Divided in 10 “Learning modules” over two days

- Day 1:
  - Introduction
  - C++ Interpreter
  - Histograms, Graphs and Functions
  - Graphics
  - Fitting
- Day 2:
  - Python Interface
  - ROOTBooks
  - Working with Files
  - Working with Columnar Data
  - Developing Packages
- Front lectures and Hands-On exercises

The “Learning Modules”

- Each module treats a well defined topic
- Each module relies on concepts treated in previous modules
- We will declare before each module the “Learning Objectives”

Resources

- ROOT Website: https://root.cern
- Material online: 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

How to Use ROOT at CERN

- Your VM!
- On Lxplus7/Lxbatch7
  - . /cvmfs/sft.cern.ch/lcg/app/releases/ROOT/6.11.02/x86_64-centos7-gcc48-opt/root/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

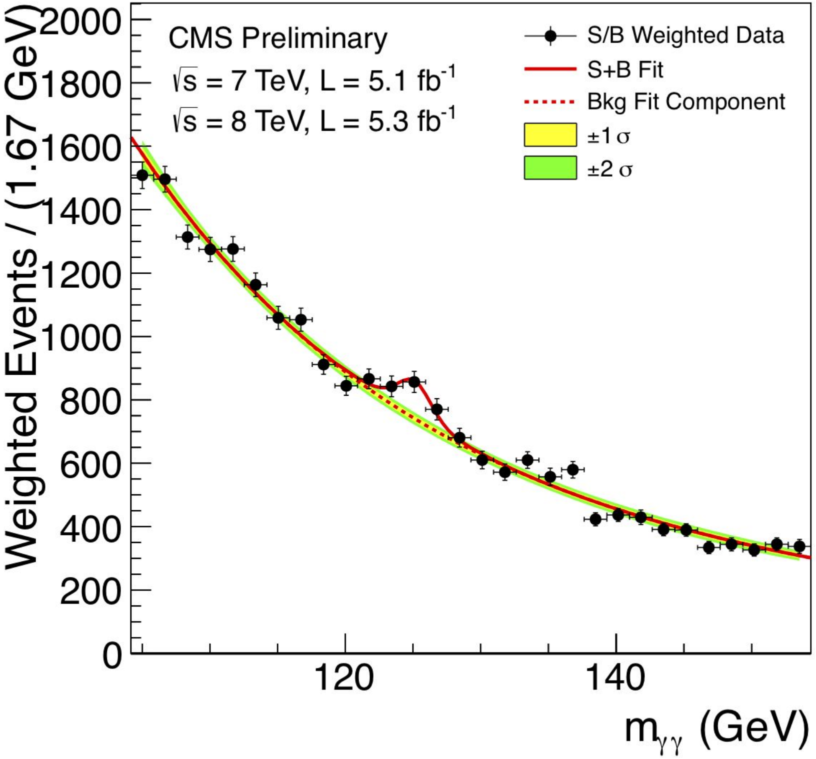
Note: ROOT on Windows is in experimental mode. We'll get there soon.

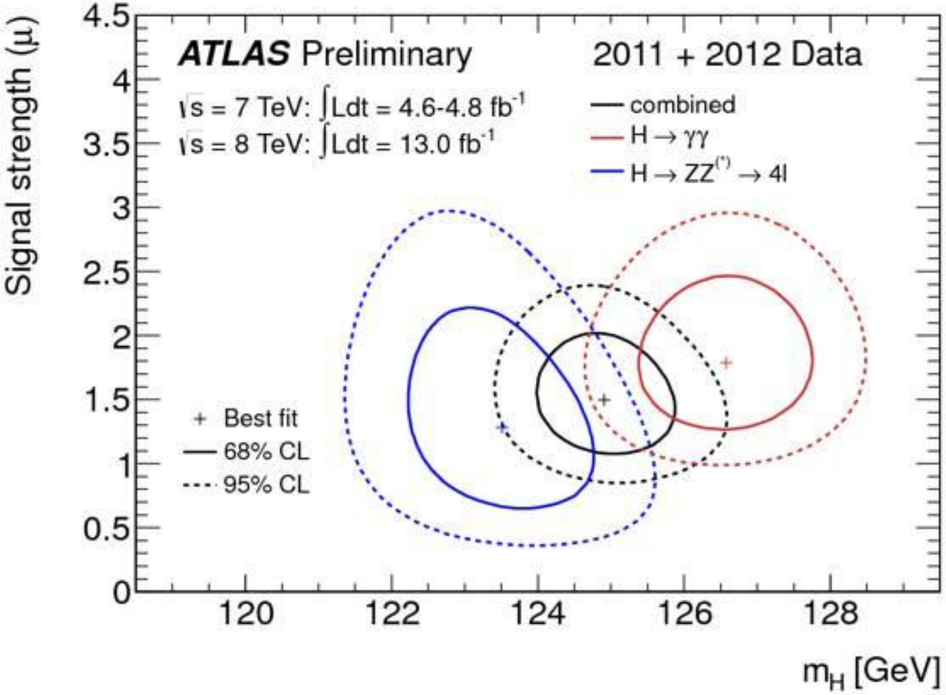
Day 1

Introduction

A Quick Tour of ROOT

What can you do with ROOT?





ROOT in a Nutshell

- ROOT is a software framework with building blocks for:
  - Data processing
  - Data analysis
  - Data visualisation
  - Data storage

An Open Source Project

We are on github

github.com/root-project

All contributions are warmly welcome!

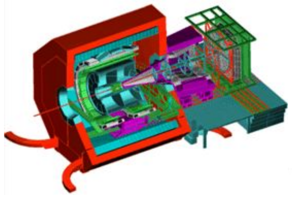
- ROOT is written mainly in C++ (C++11/17 standard)
  - Bindings for Python available as well
- Adopted in High Energy Physics and other sciences (but also industry)
  - 1 EB of data in ROOT format
  - Fits and parameters’ estimations for discoveries (e.g. the Higgs)
  - Thousands of ROOT plots in scientific publications

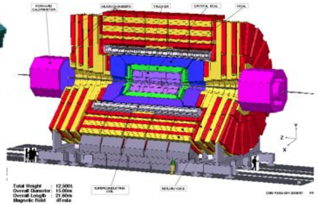
ROOT in a Nutshell

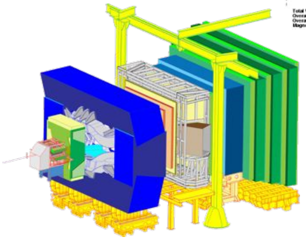
- ROOT can be seen as a collection of building blocks for various activities, like:
  - **Data analysis: histograms, graphs, functions**
  - **I/O: row-wise, column-wise** storage of any C++ object
  - **Statistical tools** (RooFit/RooStats): rich modeling and statistical inference
  - Math: **non trivial functions** (e.g. Erf, Bessel), optimised math functions
  - **C++ interpretation**: full language compliance
  - **Multivariate Analysis** (TMVA): e.g. Boosted decision trees, NN
  - **Advanced graphics** (2D, 3D, event display)
  - **Declarative Analysis**: TDataFrame
  - And more: HTTP servering, JavaScript visualisation

ROOT Application Domains

A selection of the experiments adopting ROOT





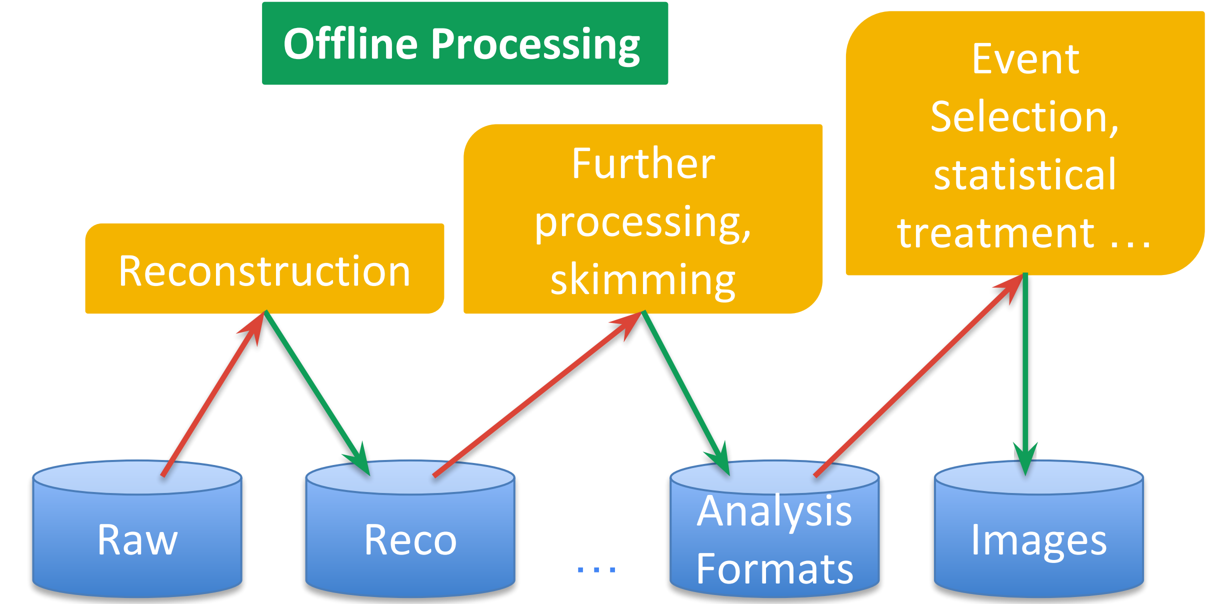


Event Filtering

Data

Analysis

Offline Processing


Event Selection, statistical treatment …
Further processing, skimming
Reconstruction
Raw
Reco
…
Analysis Formats
Images

Data Storage: Local, Network

Interpreter

- ROOT has a built-in 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
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 later)

Persistency or Input/Output (I/O)

- ROOT offers the possibility to write C++ objects into files
  - This is impossible with C++ alone
  - Used the LHC detectors to write several petabytes per year
- 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

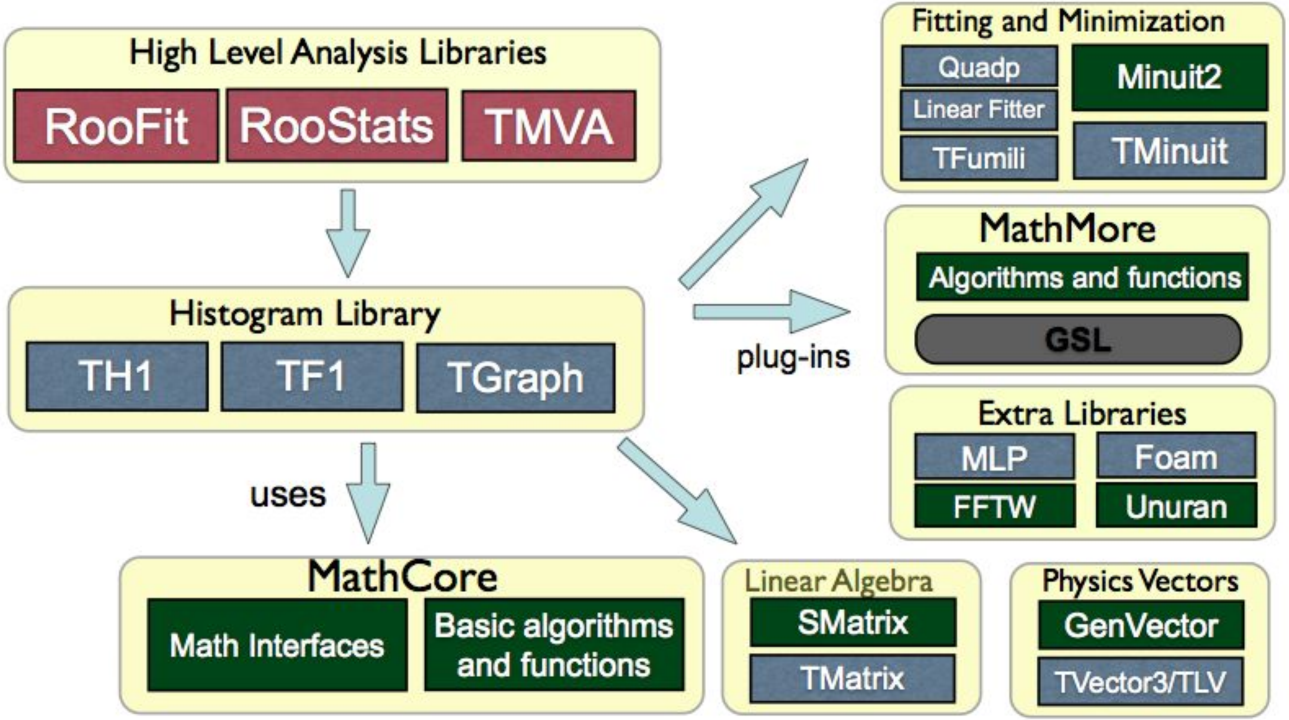
LHC Data in ROOT Format

1 EB

as of 2017

Mathematics and Statistics

- ROOT provides a rich set of mathematical libraries and tools for sophisticated statistical data analysis



High Level Analysis Libraries

RooFit

RooStats

TMVA

Histogram Library

TH1

TF1

TGraph

uses

plug-ins

Fitting and Minimization

Quadp

Linear Fitter

TFumili

Minuit2

TMinuit

MathMore

Algorithms and functions

GSL

Extra Libraries

MLP

Foam

FFTW

Unuran

MathCore

Math Interfaces

Basic algorithms and functions

Linear Algebra

SMatrix

TMatrix

Physics Vectors

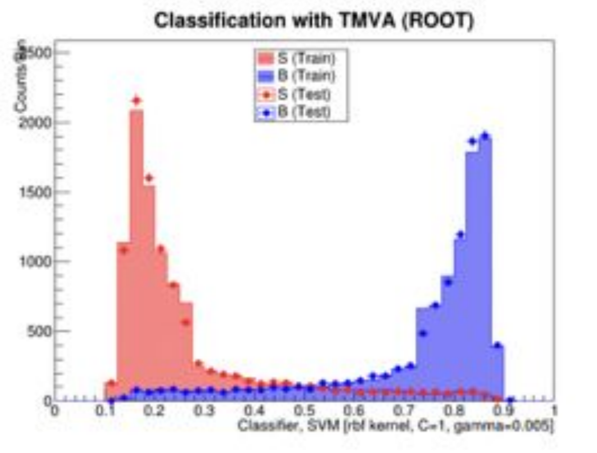
GenVector

TVector3/TLV

Machine Learning: TMVA

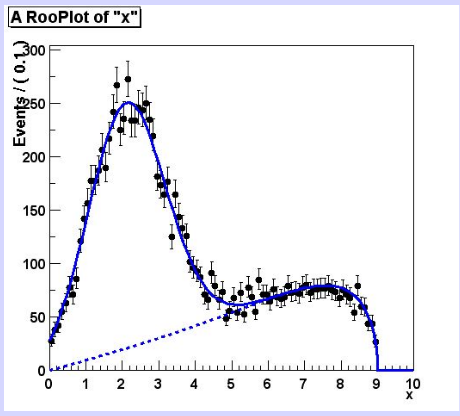
TMVA : Toolkit for Multi-Variate data Analysis in ROOT

- provides several built-in ML methods including:
  - Boosted Decision Trees
  - Deep Neural Networks
  - Support Vector Machines
- and interfaces to external ML tools
  - scikit-learn, Keras (Theano/Tensorflow), R



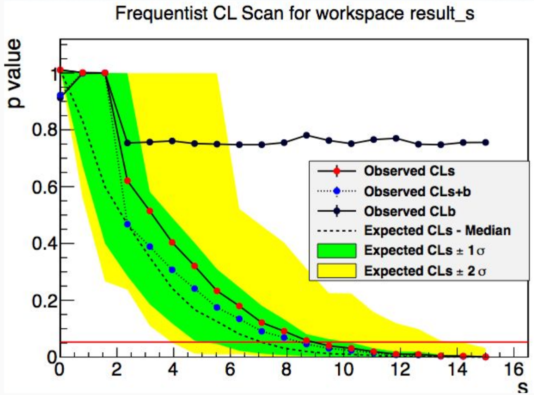
RooFit: Toolkit for Data Modeling and Fitting

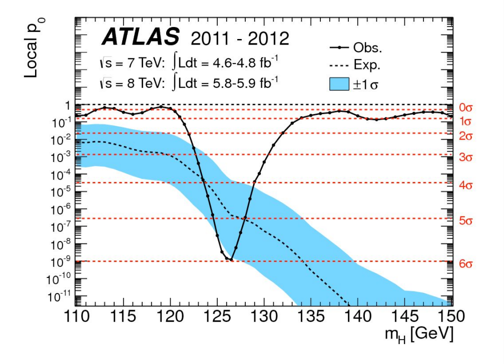
- functionality for building models: probability density functions (p.d.f.)
    - distribution of observables in terms of parameters $P(x;p)$
- complex model building from standard components
    - e.g. composition, addition, convolution,...
- RooFit models have functionality for
    - maximum likelihood fitting for parameter estimation
    - toy MC generation
    - visualization
    - sharing and storing (workspace)



RooStats

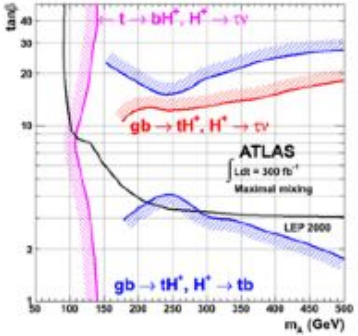
- Advanced Statistical Tools for HEP analysis. Used for :
  - estimation of Confidence/Credible intervals
  - hypotheses Tests
    - e.g. Estimation of Discovery significance
- Provides both Frequentist and Bayesian tools
- Facilitate combination of results

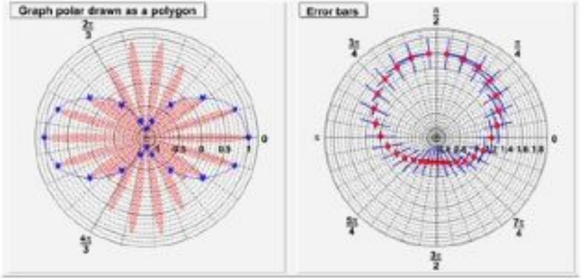


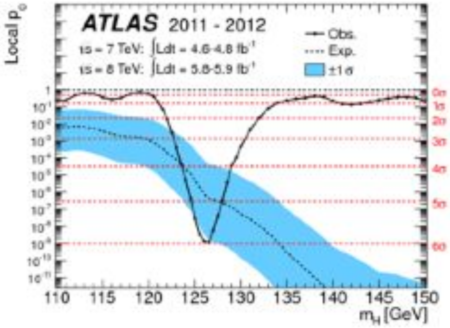


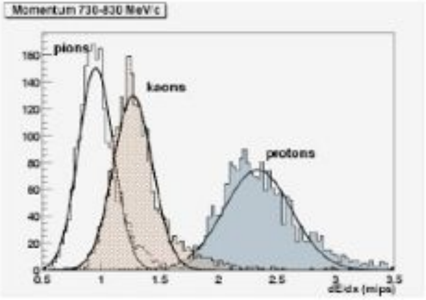
Graphics in ROOT

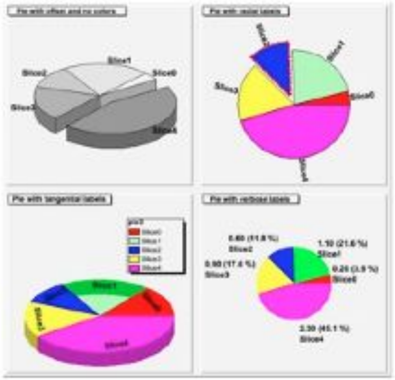
- Many formats for data analysis, and not only, plots

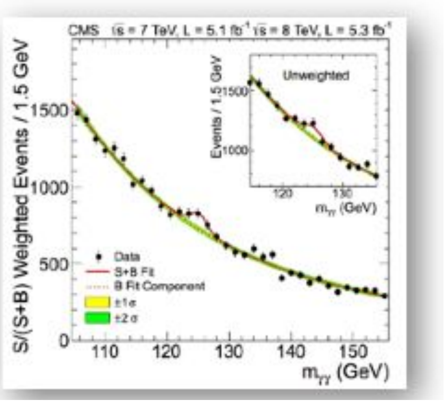




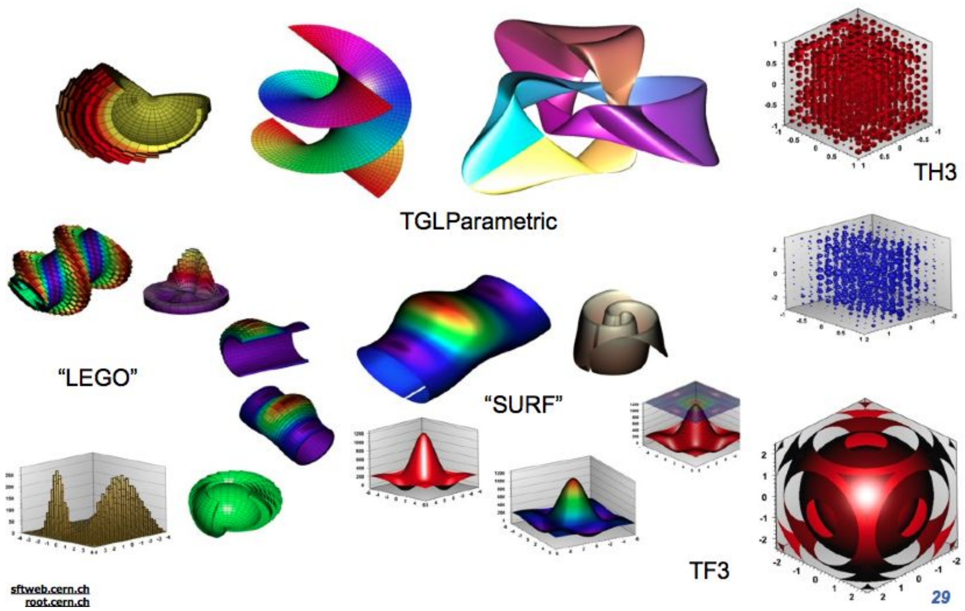


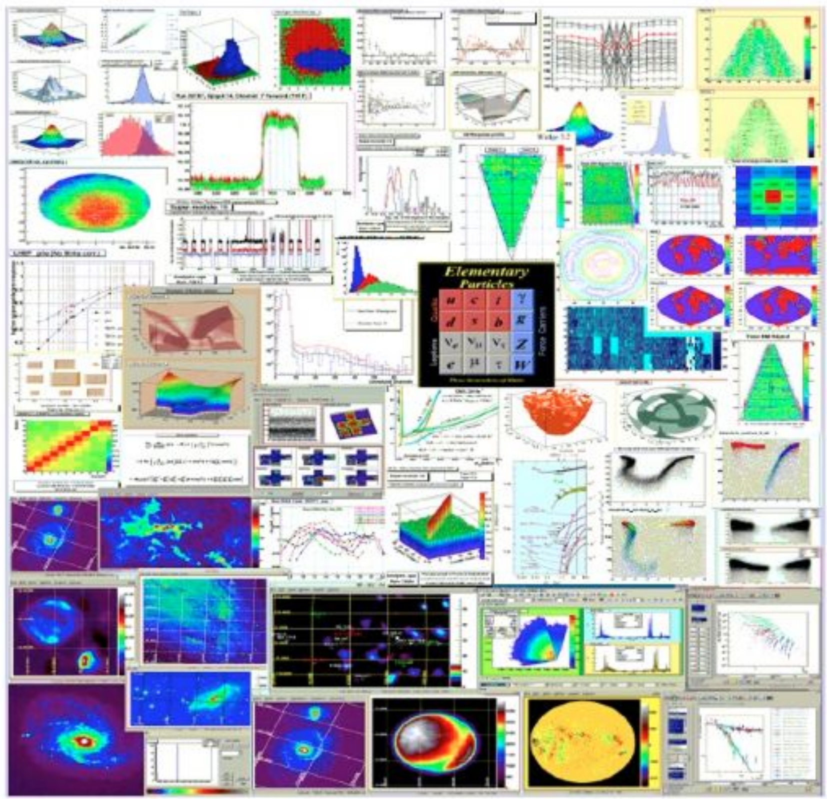






2D and 3D Graphics



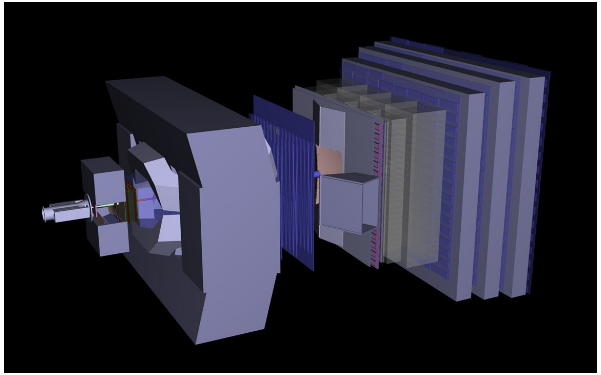


Can save graphics in many formats:
ps, pdf, svg, jpeg, LaTex, png, c, root …

JSROOT

- JSROOT: a JavaScript version of ROOT graphics and I/O
- Complements traditional graphics
- Visualisation on the web or embedded in notebooks
- Basic functionality for exploring data in ROOT format

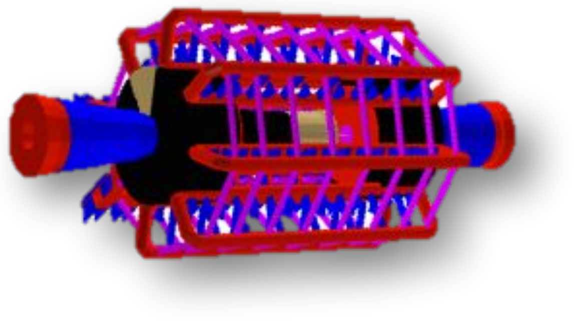
More details to come!

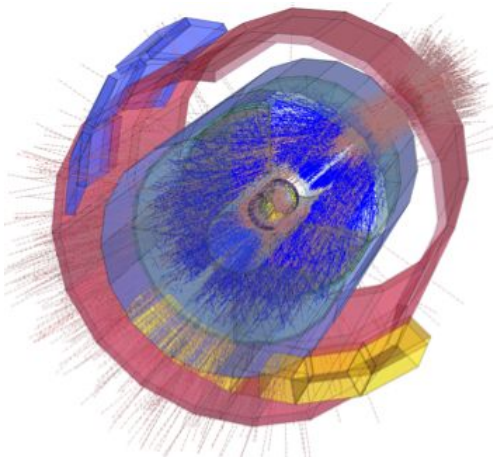


Parallelism

- Many ongoing efforts to provide means for parallelisation in ROOT
- Explicit parallelism
  - **TThreadExecutor** and **TProcessExecutor**
  - Protection of resources
- Implicit parallelism
  - **TDataFrame**: functional chains
  - TTreeProcessor: process tree events in parallel
  - TTree::GetEntry: process of tree branches in parallel
- Parallelism is a fundamental element for tackling data analysis during LHC Run III and HL-LHC

Many More Features!





- Geometry Toolkit
  - Represent geometries as complex as LHC detectors
- Event Display (EVE)
  - Visualise particle collisions within detectors

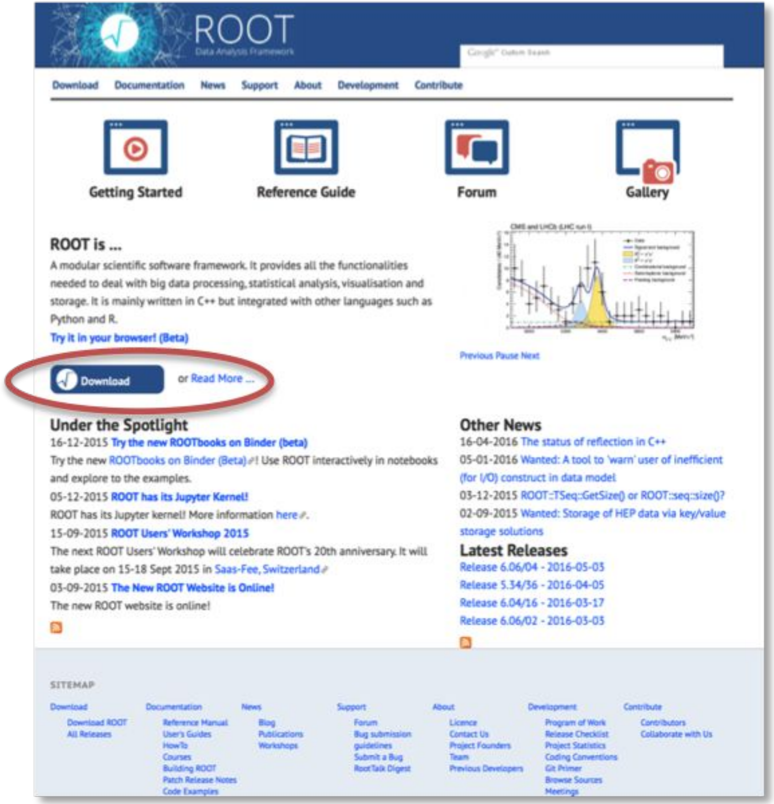
The SWAN Service

- **Data analysis with ROOT “as a service”**
- *Interface:* Jupyter Notebooks
- *Goals:*
  - Use ROOT only with a web browser
    - Platform independent ROOT-based data analysis
    - Calculations, input and results “in the Cloud”
  - Allow easy sharing of scientific results: plots, data, code
    - Through your CERNBox
  - Simplify teaching of data processing and programming

http://swan.web.cern.ch

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
  - …



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)

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 take place on 15-18 Sept 2015 in Saas-Fee, Switzerland

03-09-2015 The New ROOT Website is Online!

The new ROOT website is online!

Other News

16-04-2016 The status of reflection in C++

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

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

Release 6.06/04 - 2016-05-03

Release 5.34/36 - 2016-04-05

Release 6.04/16 - 2016-03-17

Release 6.06/02 - 2016-03-03

Ex Tempore Exercise

- Fire up your Virtual Machine
- Get the ROOT sources:
  - *git clone http://github.com/root-project/root*
  - *Or visit https://root.cern.ch/content/release-61102*
- Create a build directory and configure ROOT:
  - *mkdir rootBuild; cd rootBuild*
  - *cmake ../root*
- Start compilation
  - *make -j 2*
- Prepare environment:
  - *. bin/thisroot.sh*

ROOT in the VM

- ROOT is already installed in the VM
- */opt/root*
- Setup the environment:

```
. /opt/root/bin/thisroot.sh
```

A Little C++

Learning Objectives

- Comfortably handle the basic C++ that ROOT requires
- Understand what a type and a value is
- Be able to handle objects on the heap

A Simple Spec

C++ consists of

- expressions: have a value or result

```cpp
cos(0.1)
```

- declarations: introduce a name

```cpp
int i;
```

Declarations

Declarations introduce

- variables

```
int i; TMyClass obj;
```

- functions

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

- types

```
class TMyClass { int fMember; };
```

auto

Some type names are long:

```
std::vector<int> v {12, 13, 14};
std::vector<int>::iterator iter = v.begin();
```

`auto` takes the type from the right hand side of an assignment:

```
std::vector<int> v {12, 13, 14};
auto iter = v.begin();
```

A typical class

```
class TMyClass {
  int fNum;
public:
  TMyClass(): fNum(42) {}
  int Get() const { return fNum; }
  void Set(int n) { fNum = n; }
};
```

A typical class (2)

Looking at the parts:

```
class TMyClass {
  int fNum;
  TMyClass(): fNum(42) {}
  int Get() const { return fNum; }
  void Set(int n) { fNum = n; }
};
```

Everything inside {} is part of the class. It’s all declarations.

A typical class (3)

```
class TMyClass {
  int fNum;
public:
  TMyClass(): fNum(4
  int Get() const {
  void Set(int n) { fNum = n; }
};
```

By default, classes hide their contents (`private`).

`public:` starts an externally visible section.

A typical class (4)

Looking at the parts:

```
class TMyClass {
   int fNum;
   int Get() const { return fNum; }
   void Set(int n) { fNum = n; }
};
```

The data of the class. Any value of this class type will just be “an int”, internally - its value will be a number.

A typical class (5)

Looking at the parts:

```
class TMyClass {
  int fNum

  TMyClass(): fNum(42) {}
  int Get() const { r
  void Set(int n) { f
};
```

A member function, can be called on values of that class:

```
MyClass obj;
auto i = obj.Get();
```

A typical class (5)

Looking at the parts:

```
class TMyClass {
  int fNum;
public:

  int Get() const { return fNum; }
  void Set(int n) { fNum = n; }
};
```

fNum is the result of the function call.

A typical class (6)

Looking at the parts:

```
class TMyClass {
  int fNum;

  int Get() const { return fNum; }
  void Set(int n) { fNum = n; }
};
```

The function does not change the MyClass object - it’s `const`

A typical class (7)

Looking at the parts:

```
class TMyClass {
  int fNum;
public:

  void Set(int n) { fNum = n; }
};
```

This function takes a parameter - and sets the data member to the new value `n`.

A typical class (8)

Looking at the parts:

```
class TMyClass {

  TMyClass(): fNum(42) {}
  int Get() const {
  void Set(int n) { fNum = n; }
};
```

A “constructor” is called by the compiler when an object is created:

```
MyClass obj;
```

A typical class (9)

Looking at the parts:

```
class TMyClass {

  TMyClass(): fNum(42) {}
  int Get() const { return fNum; }
  void Set(int n) { fNum = n; }
};
```

Constructors can have member initializers and a function body `{}`

Using a class

Put MyClass to action. What does this print?

```
void func() {
  MyClass obj;
  // printf() on screen
  //    “%d” means “an int”
  //    “\n” means “new line"
  printf("%d\n", obj.Get());
}
```

```
class MyClass {
  int fNum;
public:
  MyClass(): fNum(42) {}
  int Get() { return fNum; }
//...
};
```

Using a class (2)

Nothing!!! Why?!

```
void func() {
  MyClass obj;
  // printf() on screen
  //   “%d” means “an int”
  //   “\n” means “new line"
  printf(“%d\n”, obj.Get());
} // but who is calling func()?!
```

Starting something!

Every function needs a caller:

```
void outerFunc() { func(); }
void moreOuter() { outerFunc(); }
// ...?
```

The outermost function

C++ has main()

```cpp
int main(int, char**) {
  func();
  return 0;
}
```

The outermost function (2)

ROOT uses file name:

calls `MyCode()` for code in MyCode.C

root -l MyCode.C

or

root [0] .x MyCode.C

```
// MyCode.C:
int MyCode() {
  func();
  return 0;
}
```

Overload

```
class TNameAndInt {
std::string fName;
int fInt;
public:
  TNameAndInt(?) // ← ?
};
```

Overload (2)

Constructors:

```
TNameAndInt(std::string, int n);
TNameAndInt(std::string);
TNameAndInt(int);
```

can now call

```
TNameAndInt obj0("Anne", 42);
TNameAndInt obj1("Paul");
TNameAndInt obj2(17);
```

Pointers

Values are in memory, at a location (address)

```
int value = 17; // the value
int* addr = &value;
```

`&` takes the address of a value

`addr` now contains the memory address of `value`

Pointers (2)

Can access values through pointers:

```
int value = 17; // the value
int* addr = &value;
*addr = 42;
```

`*addr` goes through the reference:
assigns 42 to value!

Pointers (2): AAA

Can access values through pointers:

```cpp
auto value = 17; // the value
auto addr = &value;
*addr = 42;
```

`*addr` goes through the reference:
assigns 42 to value!

Pointers (3)

Can access members through pointers:

```
// call as print(&obj)
void print(TMyClass* ptr) {
  printf("value is %d\n", ptr->Get());
}
```

`->` accesses member of object pointed to

Scope

Variables have a lifetime

```
void f() {
  TNamed n("a", "b");
} // destroys n
```

Scope (2)

It’s the `` `{}` `` that defines lifetime:

```
void f(bool flag) {
  if (flag) {
    TNamed n("a", "b");
  } // destroys n
  // ERROR: n is not known here!
  printf("%s\n", n.GetName());
}
```

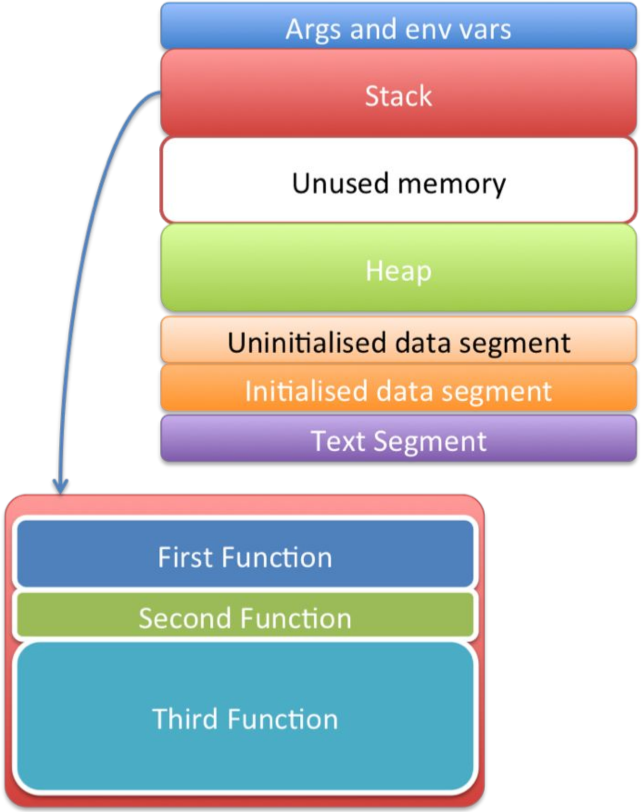
Heap

`new` creates, `delete` destroys

```
auto ptr = new int(17);
delete ptr;
```

A Program in Memory

- **Text Segment**: code to be executed.
- **Initialized Data Segment**: global variables initialized by the programmer.
- **Uninitialized Data Segment**: This segment contains uninitialized global variables.
- **The stack**: The stack is a collection of stack frames. It grows whenever a new function is called. "Thread private".
- **The heap**: Dynamic memory (e.g. requested with "`new`").



Compile C++ programs

- C++ is a compiled language
- Need a tool to transform ASCII files containing C++ code into binaries executable by machines
- **A compiler**
- Several available, free and commercial. We focus on GCC

```
g++ -o myProgram myProgram.cpp
./myProgram
```

Take Home Messages

Your tiny C++ intro:

- Variables, functions (incl overloading) and classes
- Pointers
- Scopes as lifetime of variables, and how to escape them (new/delete)
- Running code

This is sufficient C++ to follow the course.

This is ***NOT*** sufficient C++ for real-life.

Time for Exercises!

https://github.com/root-project/training/tree/master/BasicCourse/Exercises/C%2B%2BInterpreter#compile-and-run-a-simple-program

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 at runtime layout of classes)
  - Is started with the command:

```
root
```

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

ROOT As a Calculator

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \ldots$$

$$= \sum_{n=0}^{\infty} x^n$$

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

Exercise

- ROOT provides mathematical functions, for example the widely known and adopted Gaussian
- 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 input value

```
root [0] double x=0
root [1] exp(-x*x*.5) – TMath::Gaus(x)
[…]
```

Solution

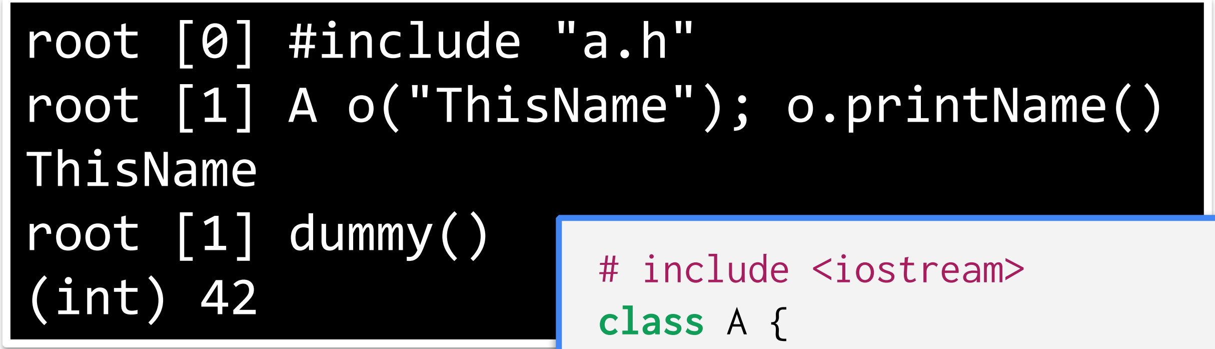
- For x values of 0,1,10 and 20 check the difference of the value of a hand-made non-normalised Gaussian and the TMath::Gaus routine

```
root [0] double x=0
root [1] exp(-x*x*.5) – TMath::Gaus(x)
[…]
```

- Many possible ways of solving this! E.g:

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

Interactivity



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

a.h

```
# 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();
```

Time for Exercises

- Go back to the geometric series example we executed at the prompt
- Make a macro out of it and run it

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:

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

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

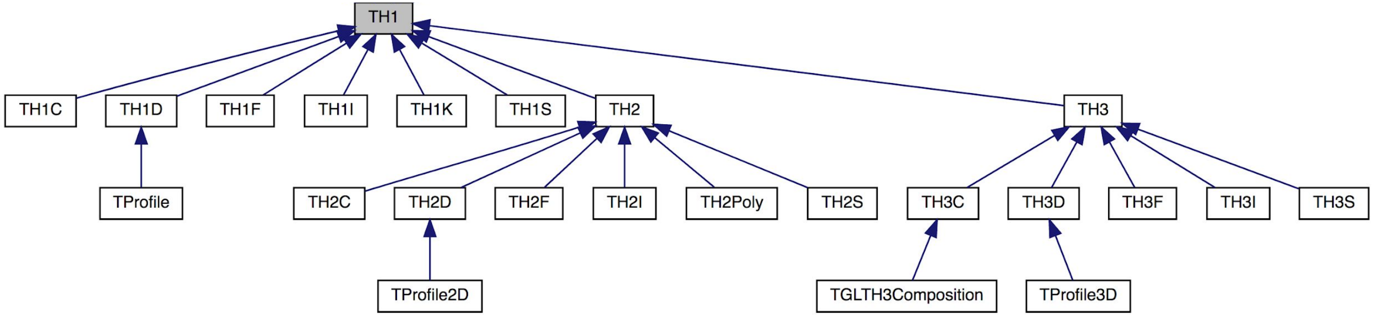
Time For Exercises

https://github.com/root-project/training/blob/master/BasicCourse/Exercises/C++Interpreter/readme.md#complete-a-simple-root-macro

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



TH1

TH1C TH1D TH1F TH1I TH1K TH1S TH2 TH3

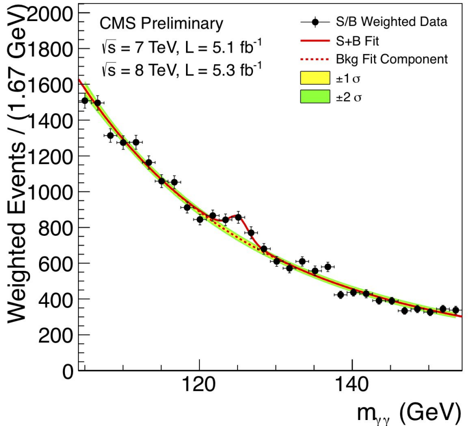
TProfile

TH2C TH2D TH2F TH2I TH2Poly TH2S

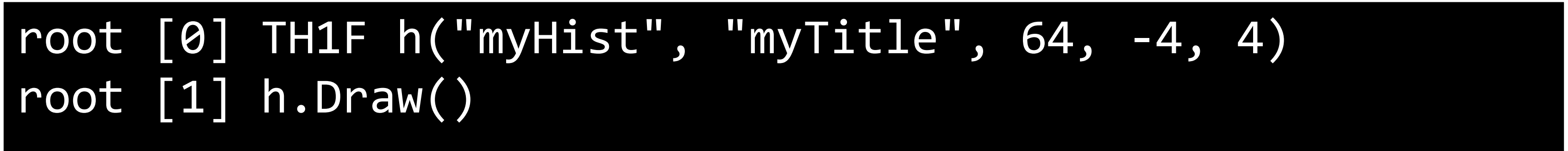
TProfile2D

TH3C TH3D TH3F TH3I TH3S

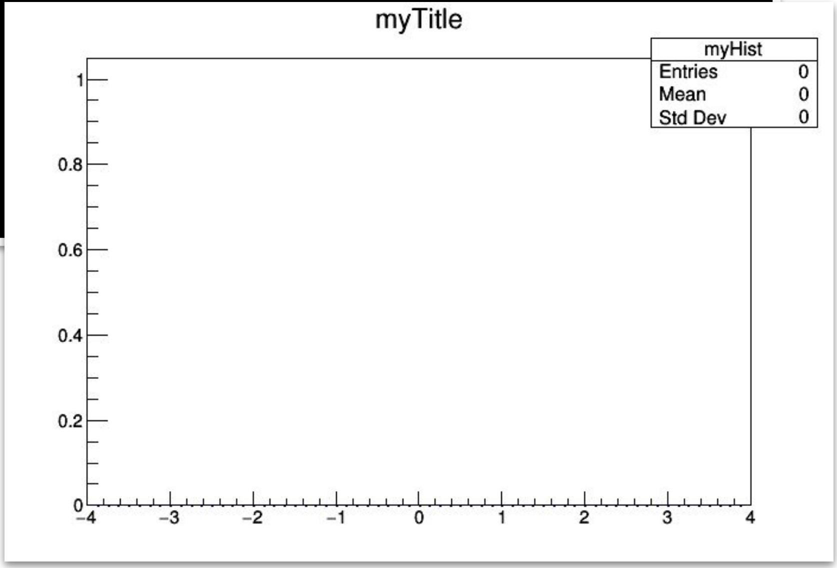
TGLTH3Composition TProfile3D



My First Histogram

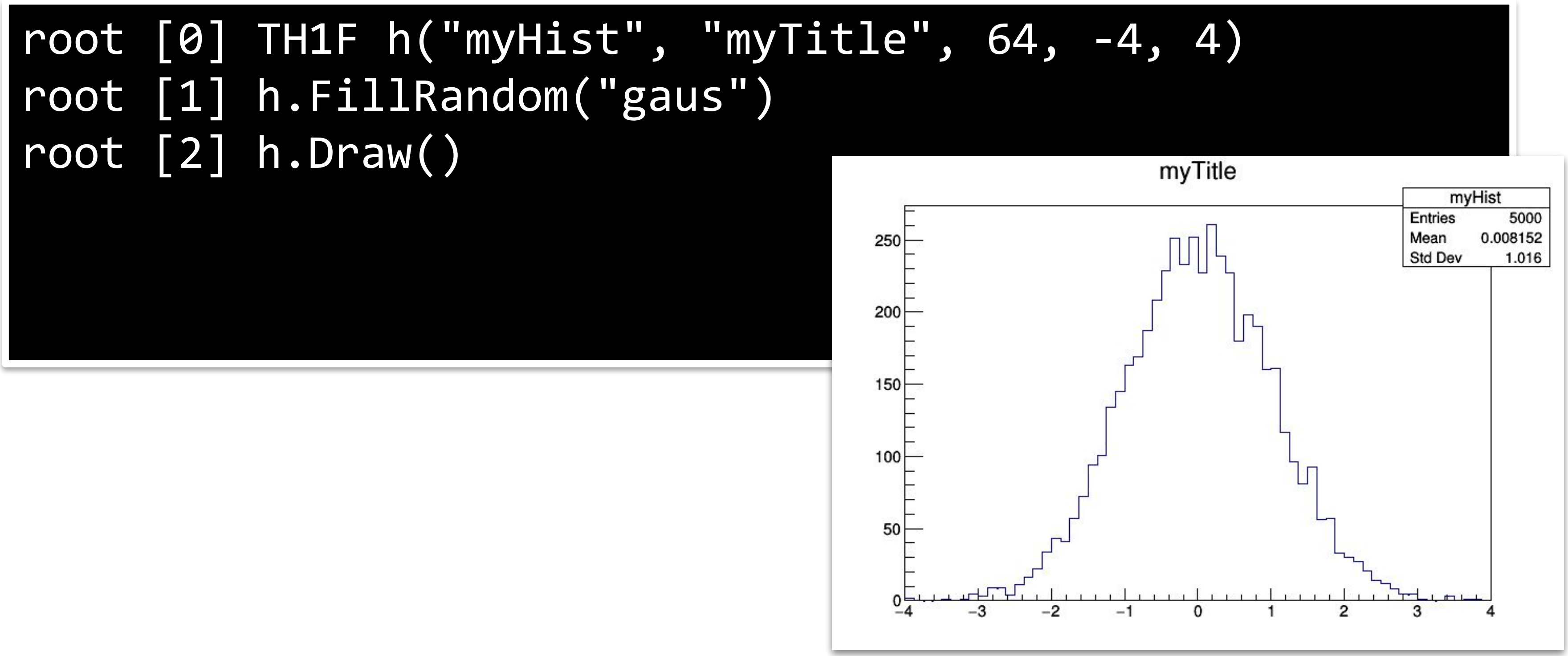


```
root [0] TH1F h("myHist", "myTitle", 64, -4, 4)
root [1] h.Draw()
```



```
myHist
Entries     0
Mean        0
Std Dev     0
```

My First Histogram



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

```
myHist
Entries      5000
Mean     0.008152
Std Dev     1.016
```

Interlude: Scope

Bad for graphics:

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

ROOT doesn’t show my histogram!

Interlude: Scope

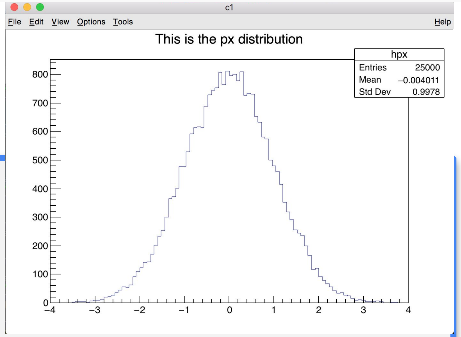
Bad for graphics:

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

ROOT doesn’t show my histogram!

Interlude: Heap

Need a way to control lifetime



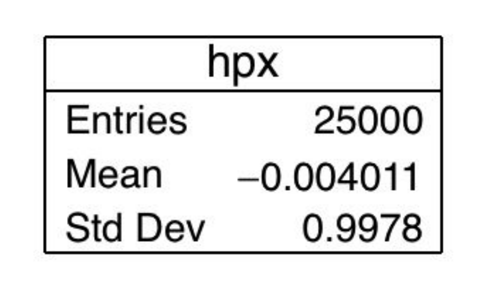
```
// makeHist.C:
void makeHist() {
  TH1F *hist = new TH1F("hist","My Histogram");
  hist->Draw(); // shows histogram
} // does not destroy hist!
```

`new` puts object on “heap”, escapes scope

Statistics and Fit parameters

- ROOT histograms have additional information called "statistics"
- ROOT adds them automatically to the plot when a histogram is drawn
- They can be turned on or off with the histogram method `SetStats()`
- `gStyle->SetOptStat()` defines which statistics parameters must be shown.

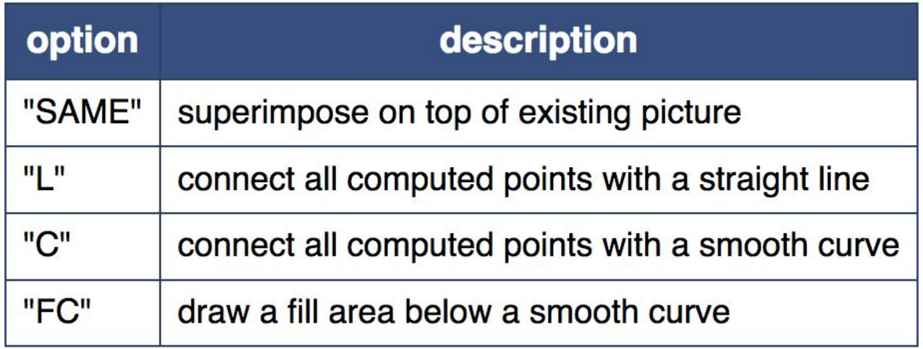
Also, after a fit, the fit result can be drawn on the plot and customized with `gStyle->SetOptFit()`



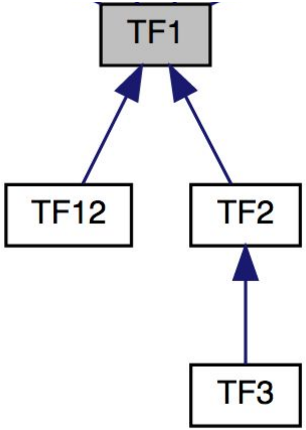
hpx	
Entries	25000
Mean	−0.004011
Std Dev	0.9978

Functions

- 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

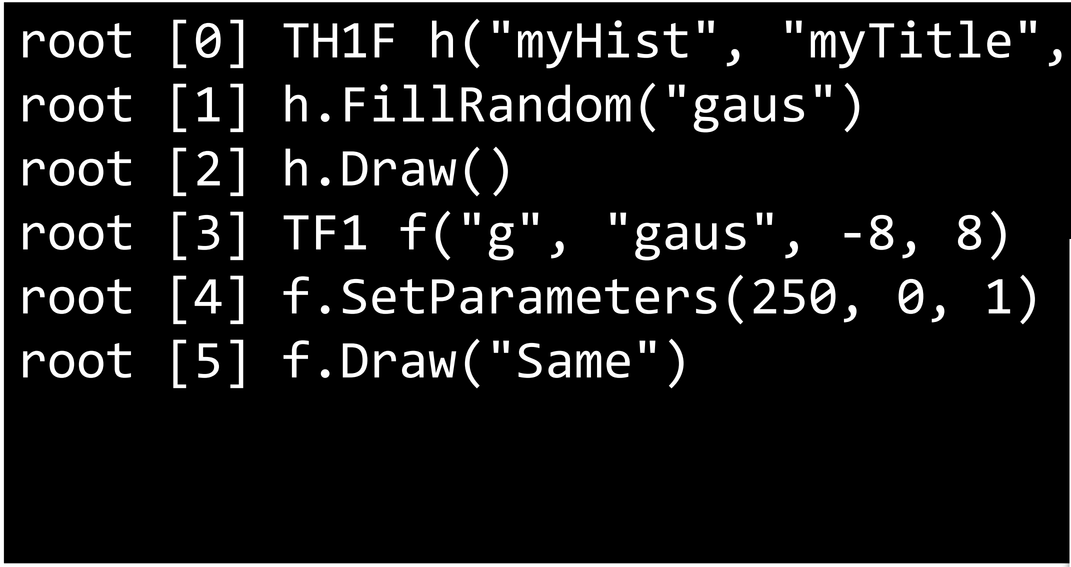


Functions

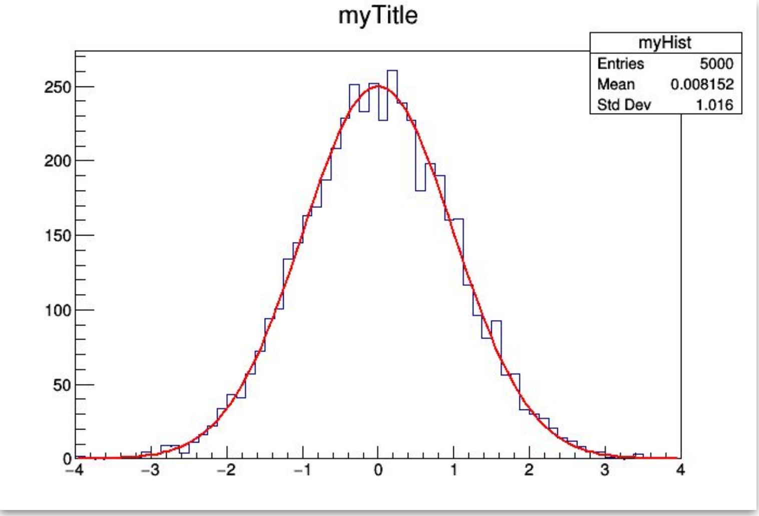
Can describe functions as:

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

My First Function



```
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")
```



ROOT as a Function Plotter

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

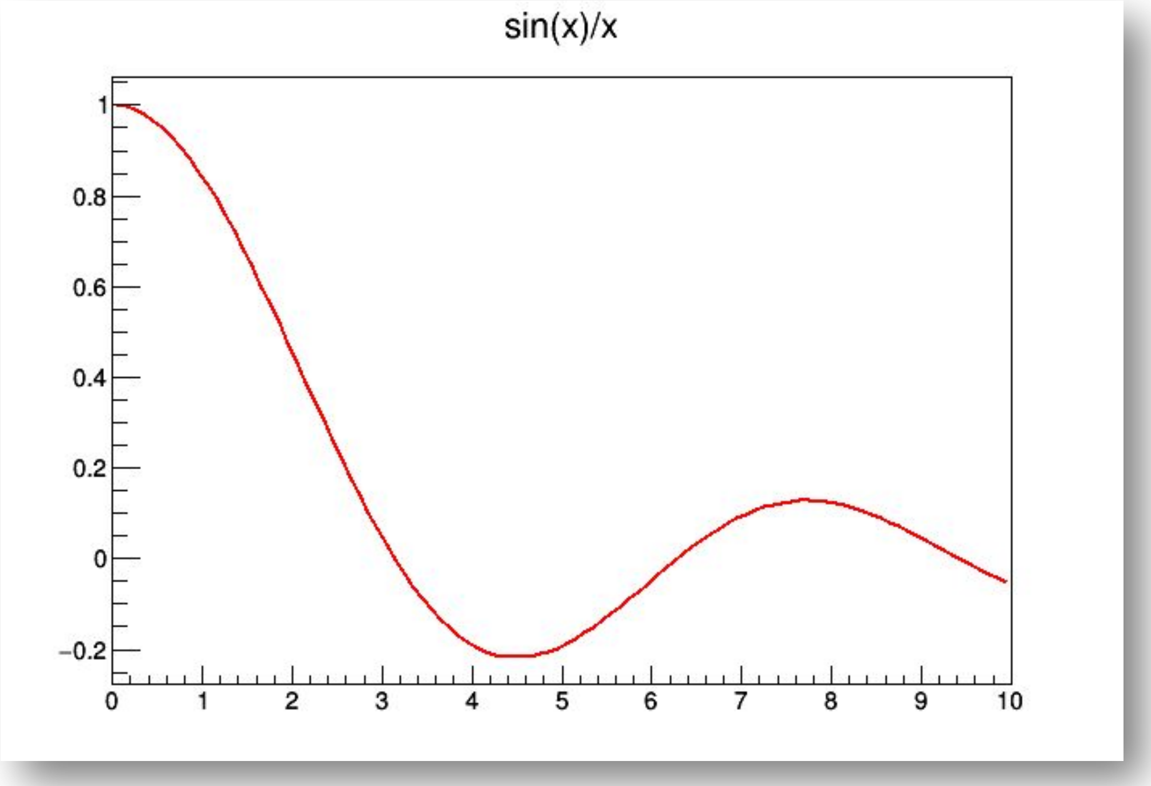
```
root [0] TF1 f1("f1","sin(x)/x",0.,10.);//name,formula, min, max
root [1] f1.Draw();
```

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

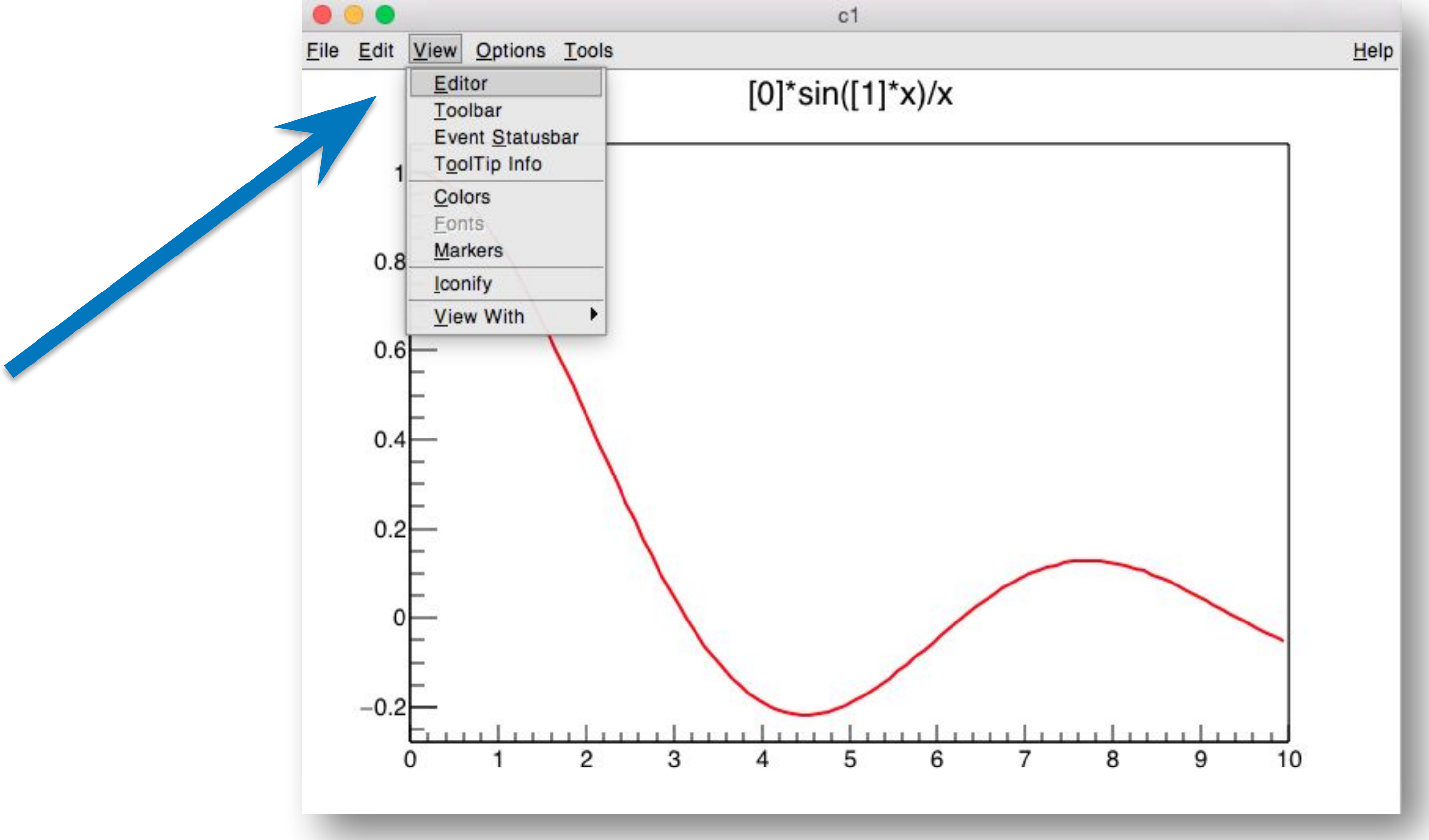
```
root [2] TF1 f2("f2","[0]*sin([1]*x)/x",0.,10.);
root [3] f2.SetParameters(1,1);
root [4] f2.Draw();
```

Try it!

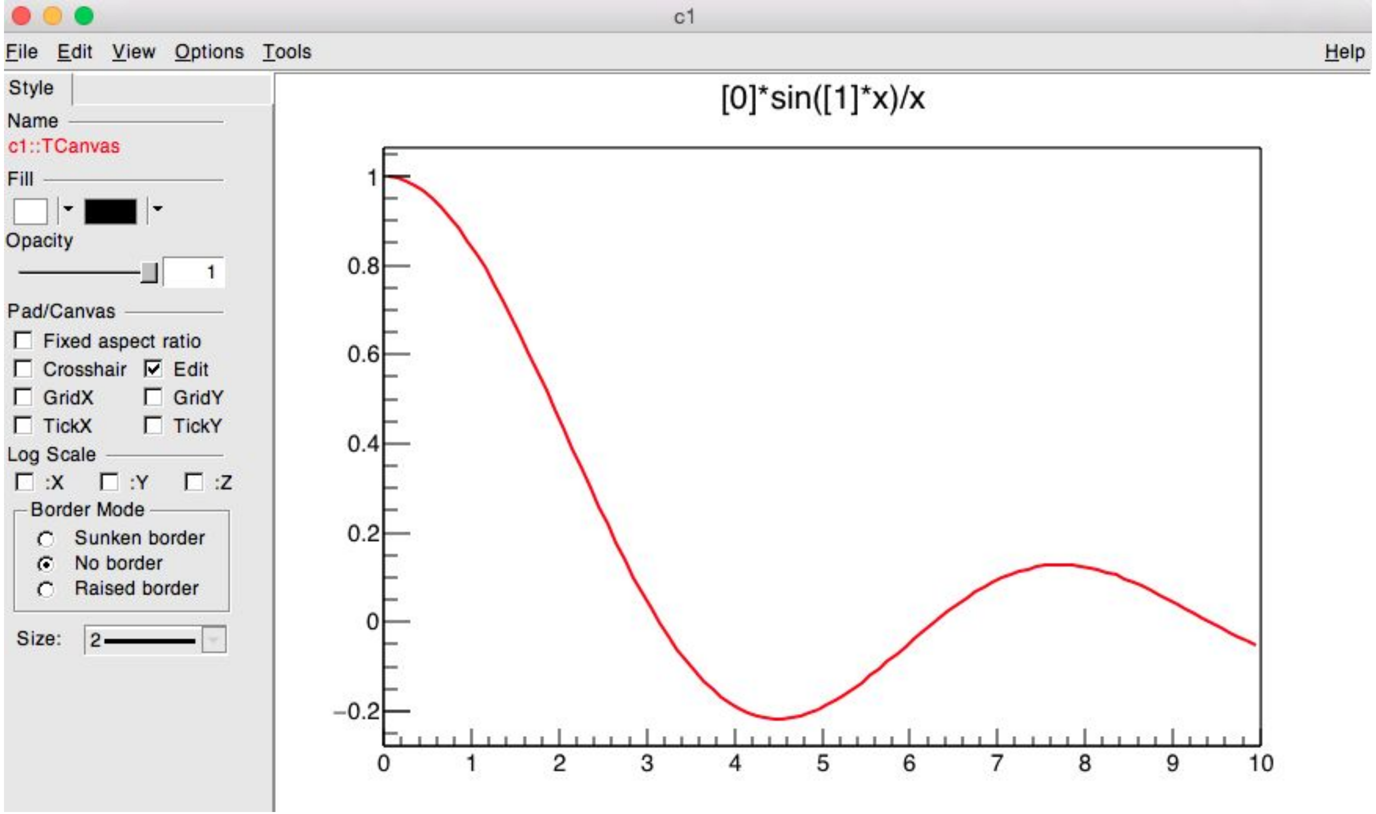
ROOT as a Function Plotter



ROOT as a Function Plotter



ROOT as a Function Plotter



Style

Name

c1::TCanvas

Fill

Opacity

1

Pad/Canvas

☐ Fixed aspect ratio

☐ Crosshair ☑ Edit

☐ GridX ☐ GridY

☐ TickX ☐ TickY

Log Scale

☐ :X ☐ :Y ☐ :Z

Border Mode

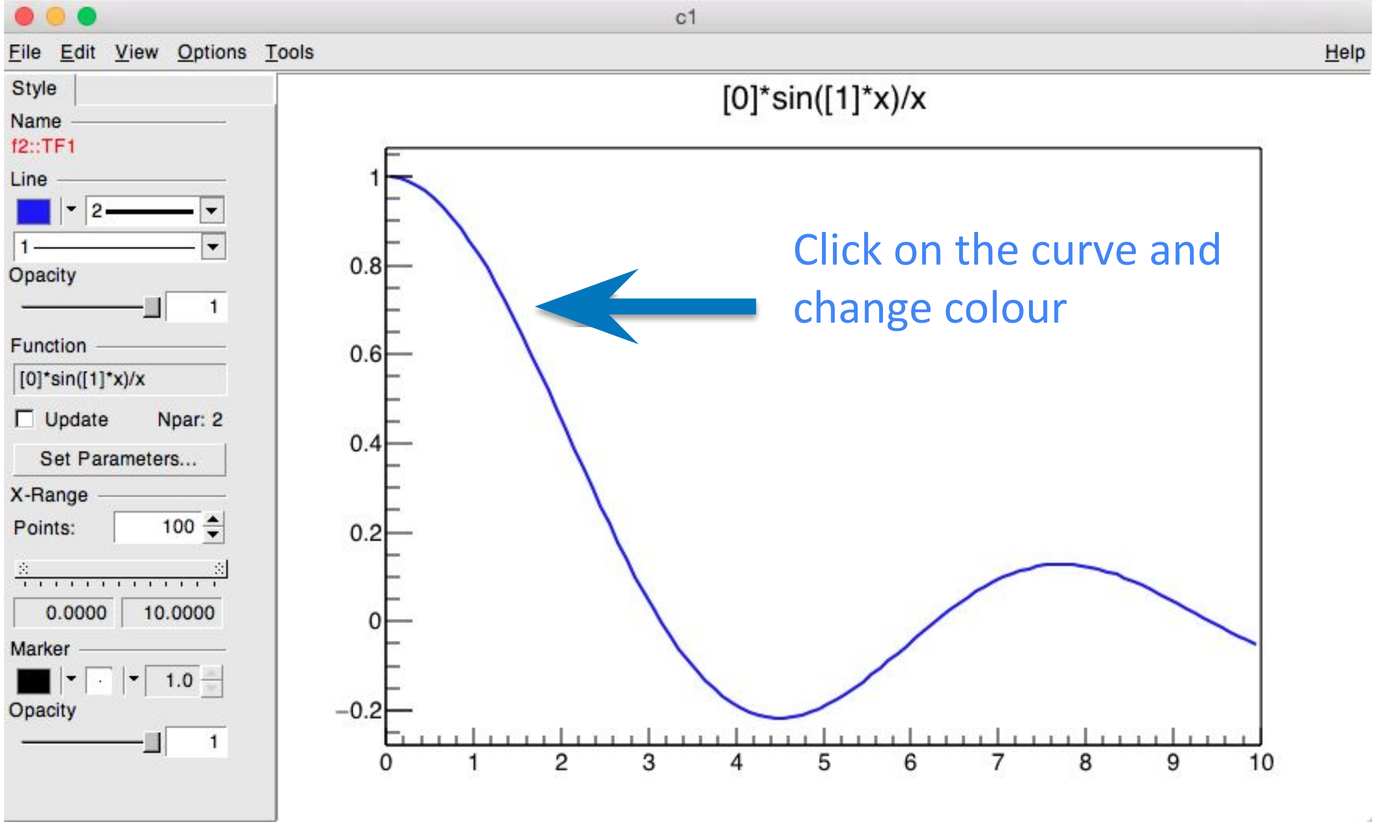
○ Sunken border

● No border

○ Raised border

Size: 2

ROOT as a Function Plotter



Name
f2::TF1

Line
2
1
Opacity 1

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

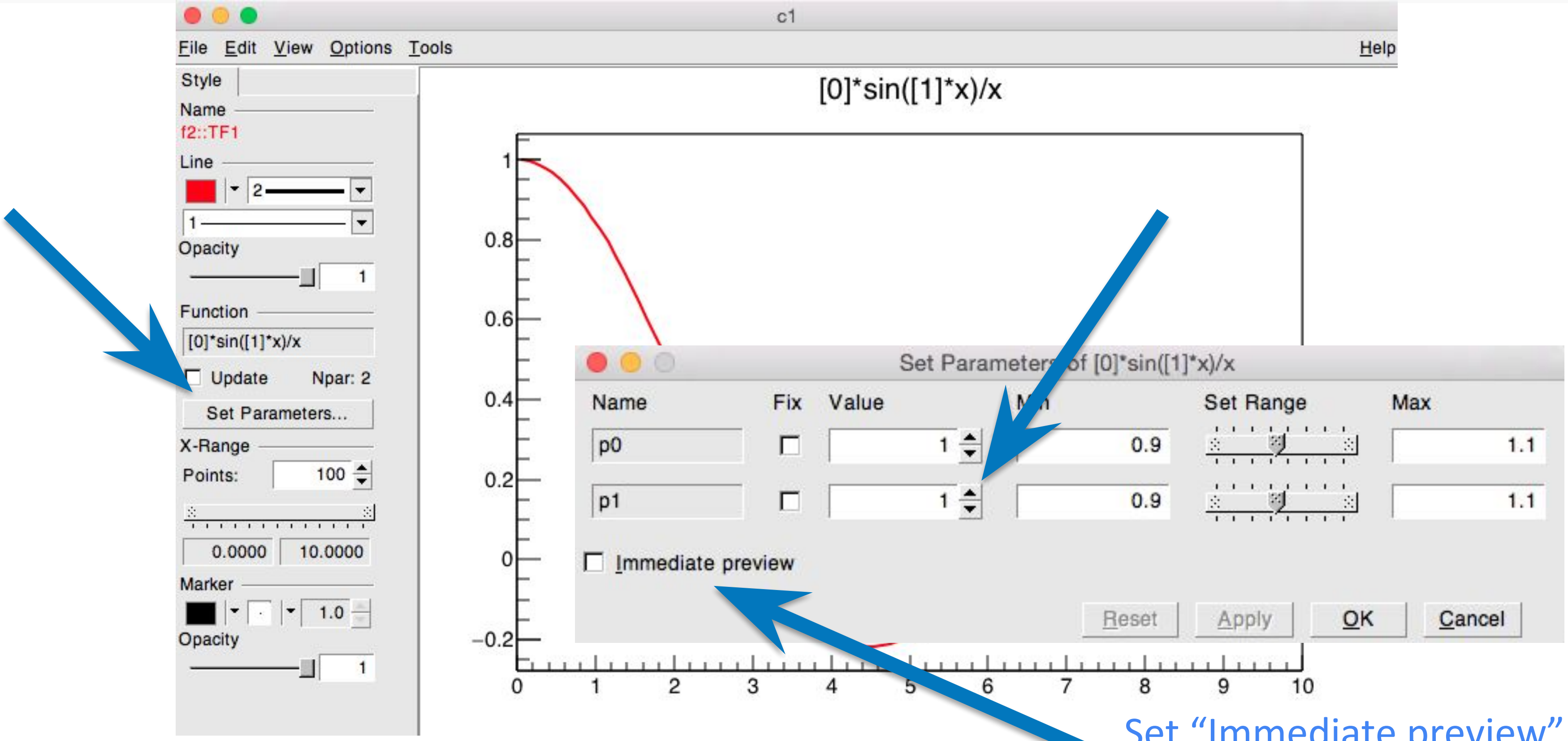
☐ Update Npar: 2

X-Range
Points: 100
0.0000 10.0000

Marker
1.0
Opacity 1

Click on the curve and change colour

ROOT as a Function Plotter



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

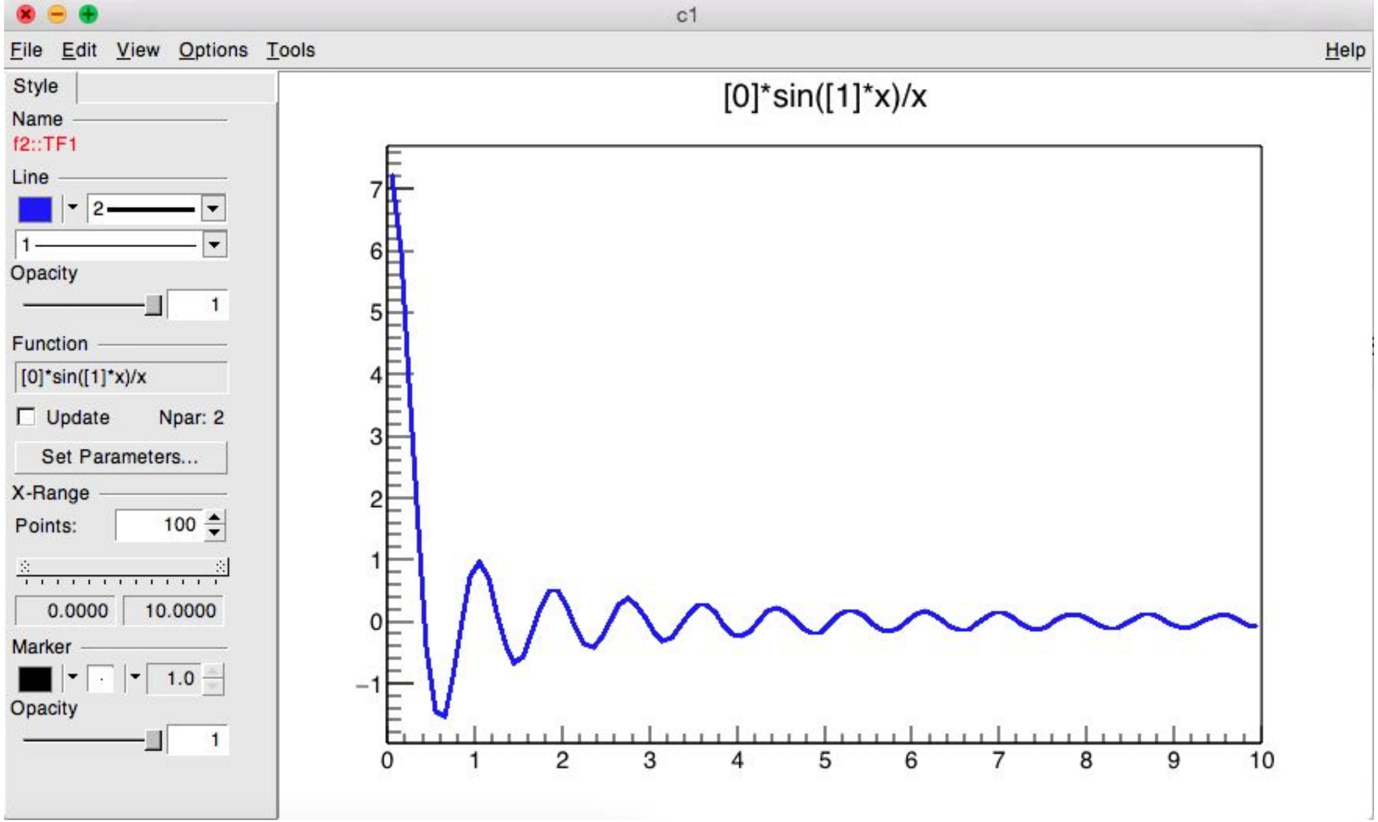
Set Parameters of [0]*sin([1]*x)/x

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

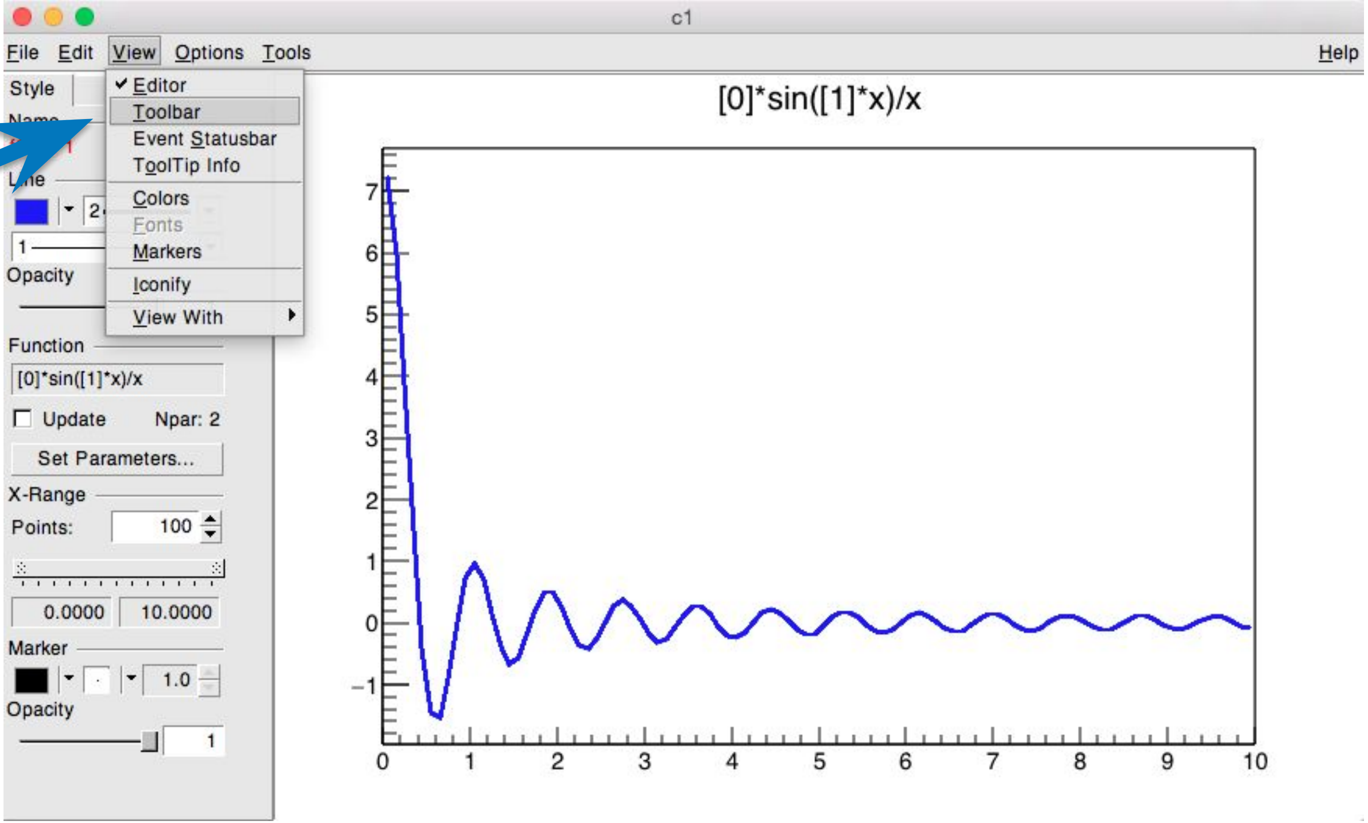
☐ Immediate preview

Set “Immediate preview”

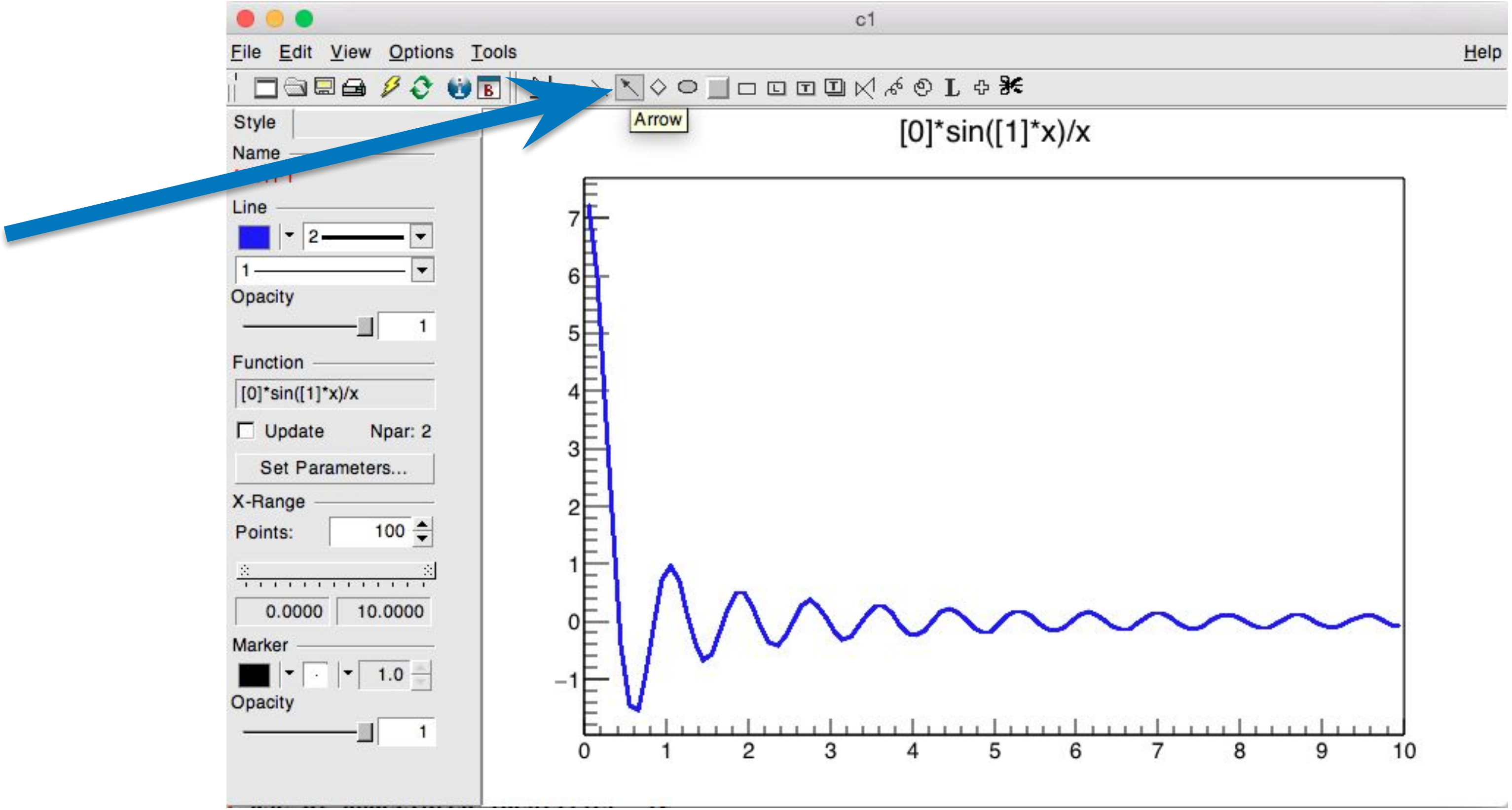
ROOT as a Function Plotter



ROOT as a Function Plotter



ROOT as a Function Plotter



Style

Name

Line

2

1

Opacity 1

Function

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

☐ Update Npar: 2

Set Parameters...

X-Range

Points: 100

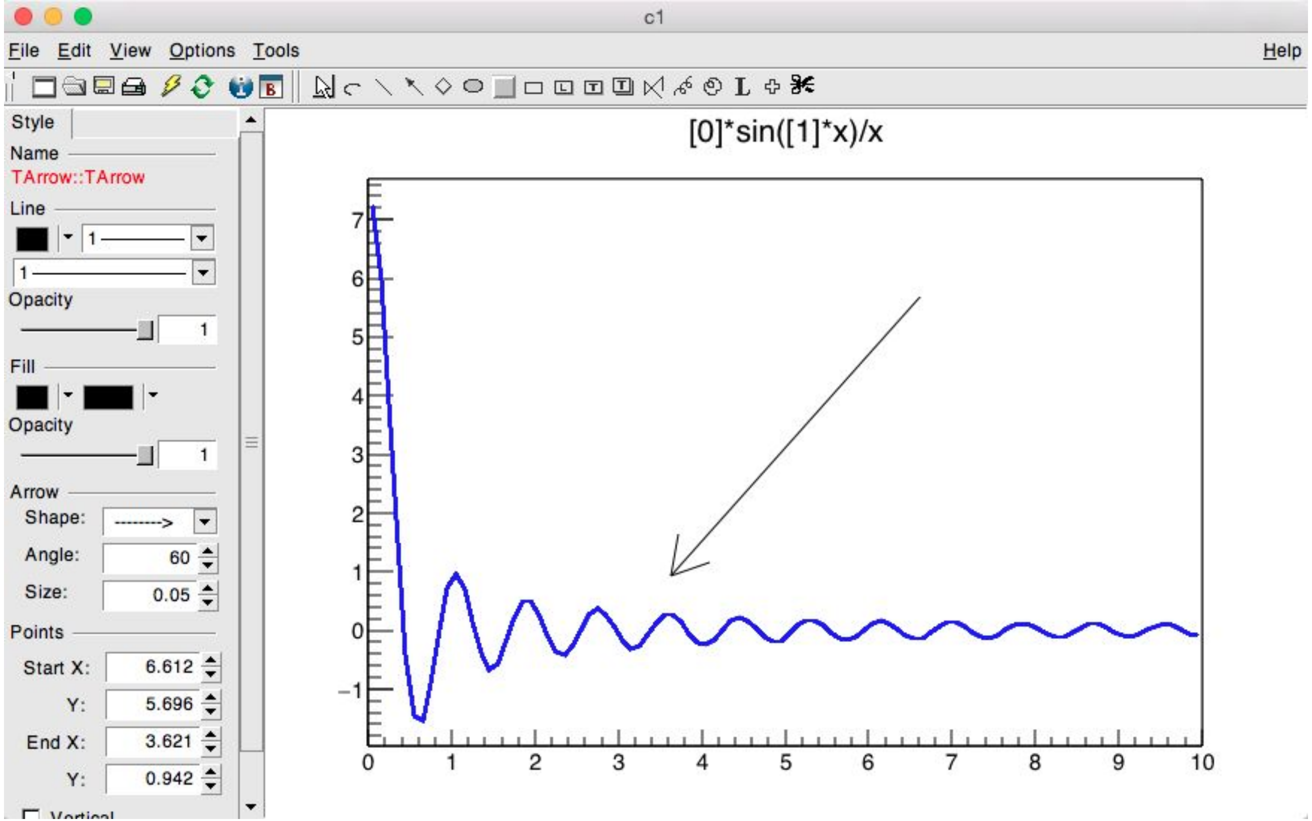
0.0000 10.0000

Marker

1.0

Opacity 1

ROOT as a Function Plotter



Name

TArrow::TArrow

Line

1

1

Opacity 1

Fill

Opacity 1

Arrow

Shape: -------->

Angle: 60

Size: 0.05

Points

Start X: 6.612

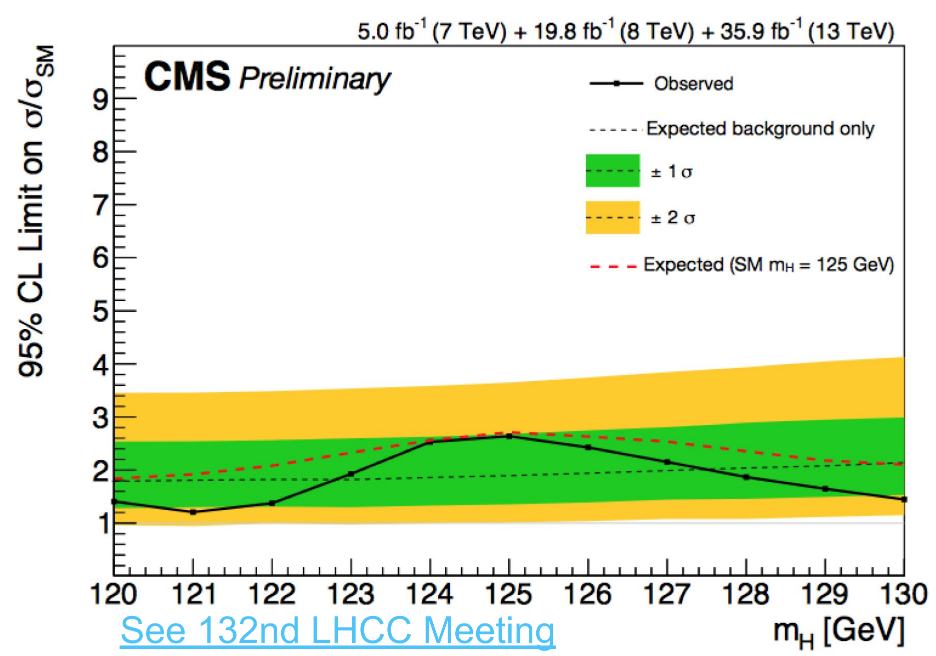
Y: 5.696

End X: 3.621

Y: 0.942

□ Vertical

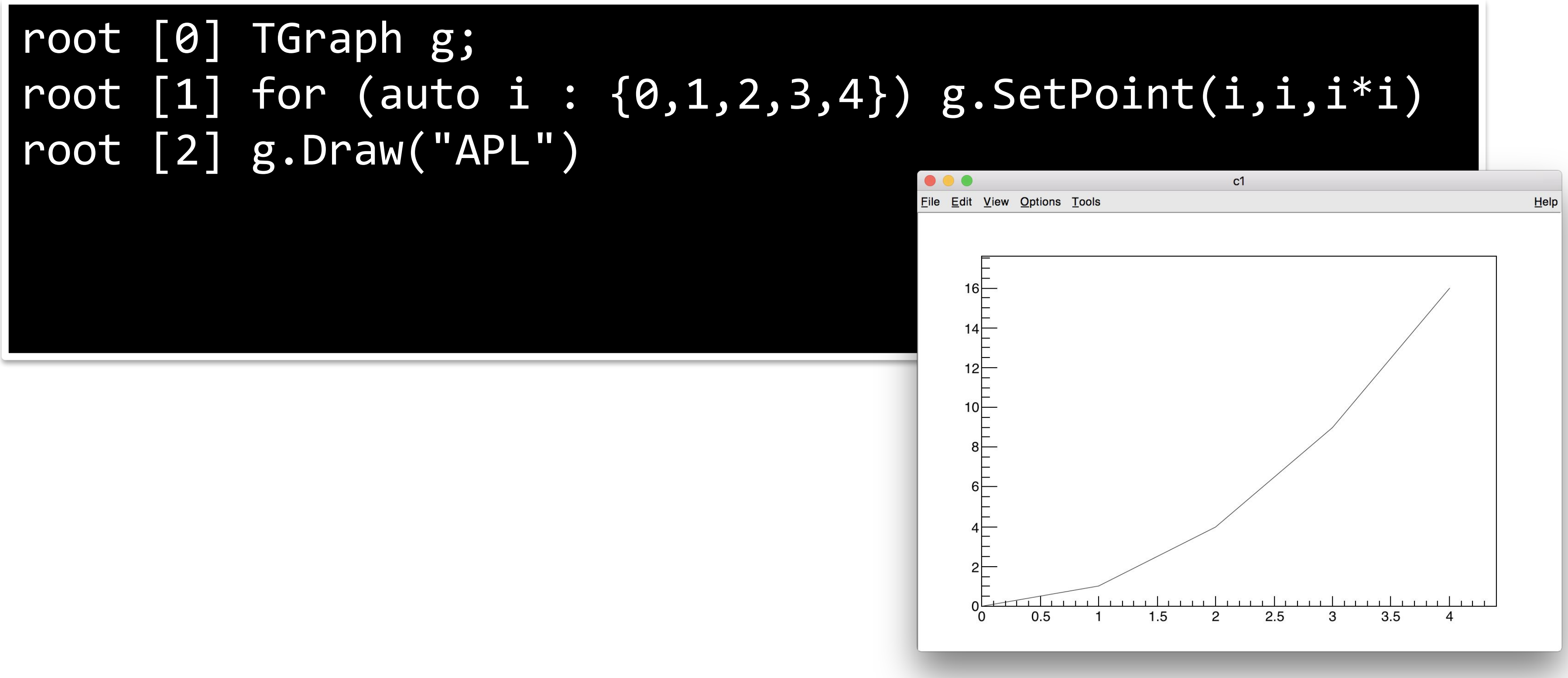
Graphs



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

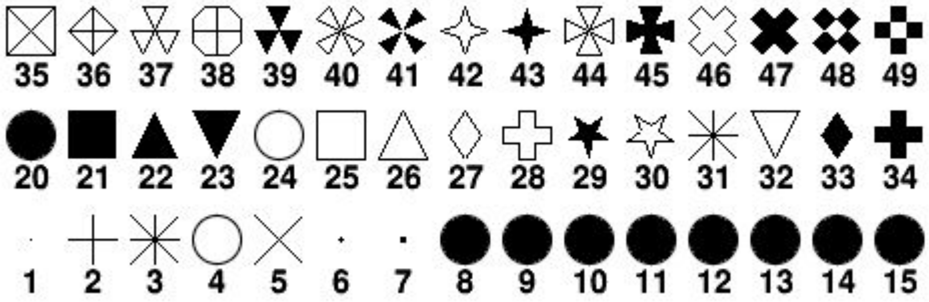


```
root [0] TGraph g;
root [1] for (auto i : {0,1,2,3,4}) g.SetPoint(i,i,i*i)
root [2] g.Draw("APL")
```

Creating a Nice Plot: Survival Kit

Credit: https://www.swissknifeshop.com

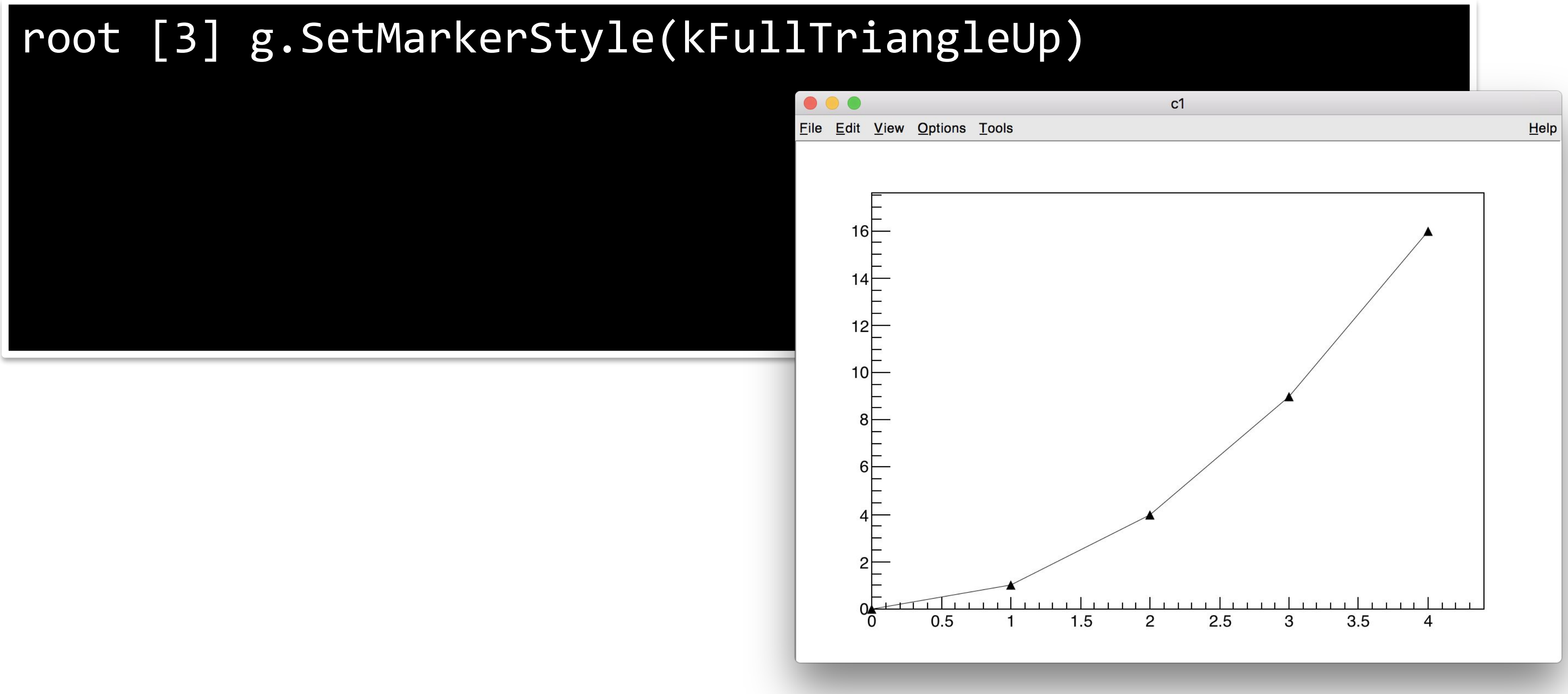
The Markers



```
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 ☺

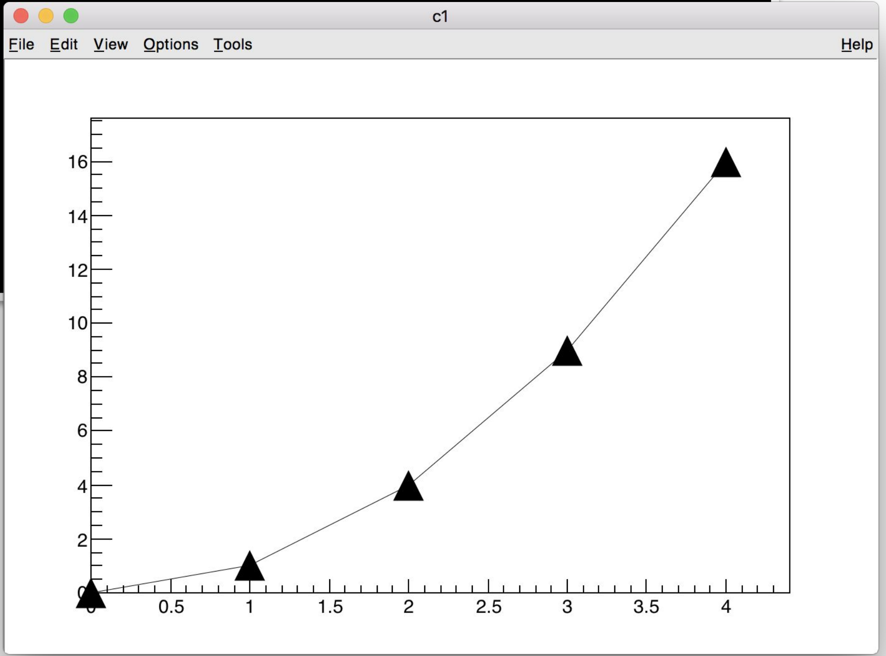
My First Graph



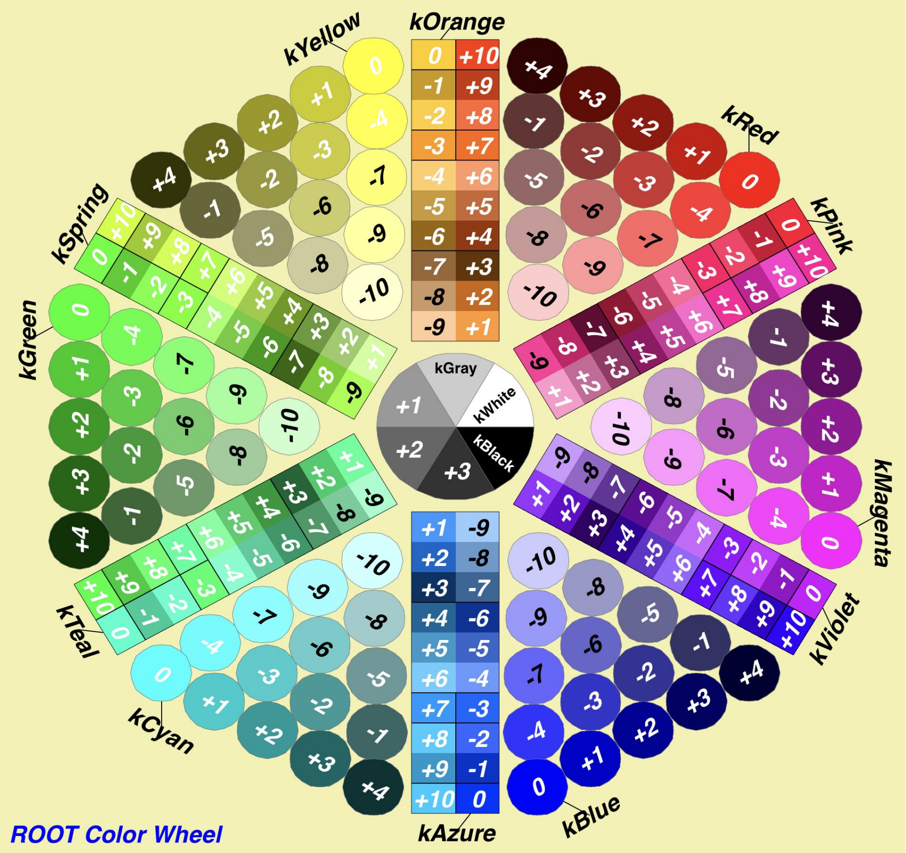
```
root [3] g.SetMarkerStyle(kFullTriangleUp)
```

My First Graph

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

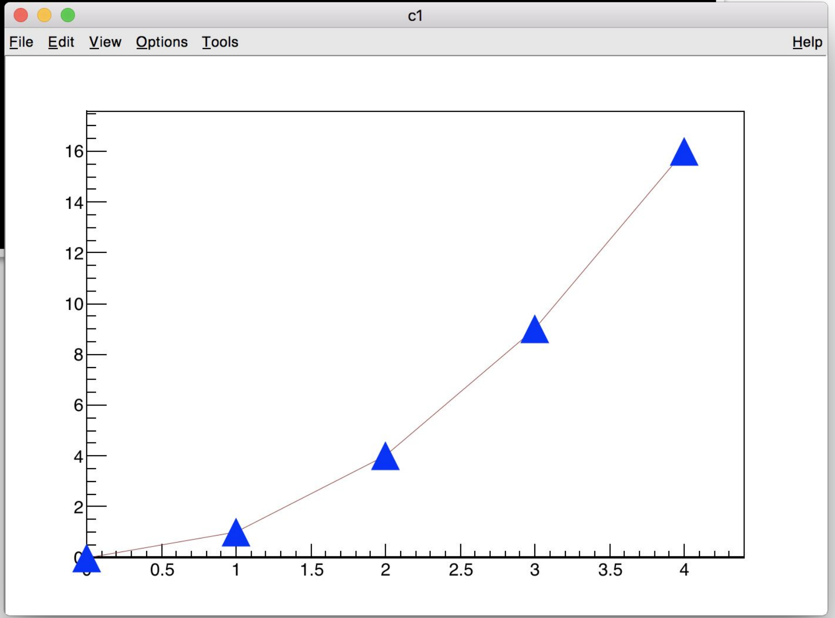


The Colors (TColorWheel)



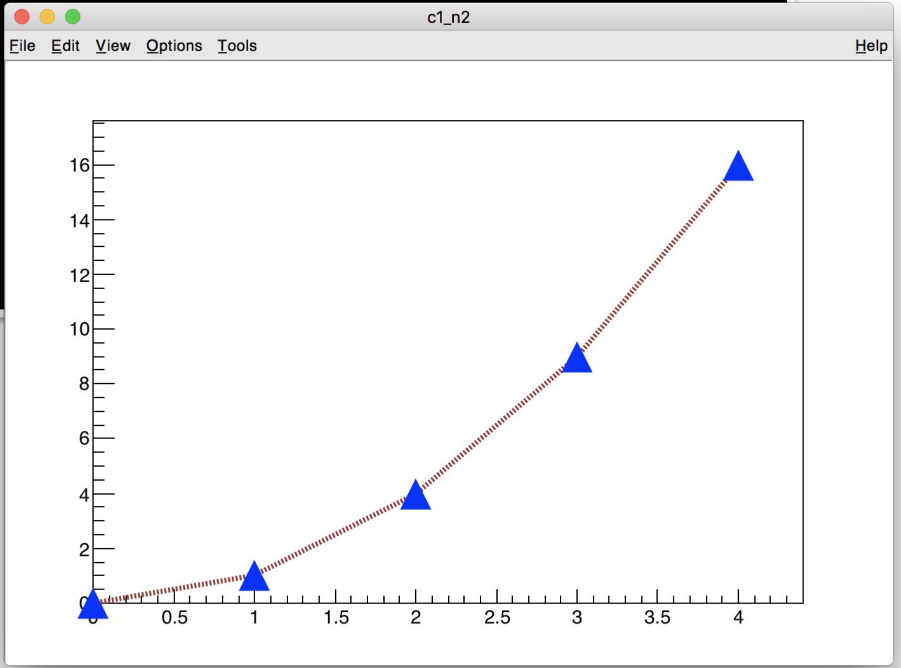
My First Graph

```
root [5] g.SetMarkerColor(kAzure)
root [6] g.SetLineColor(kRed - 2)
```

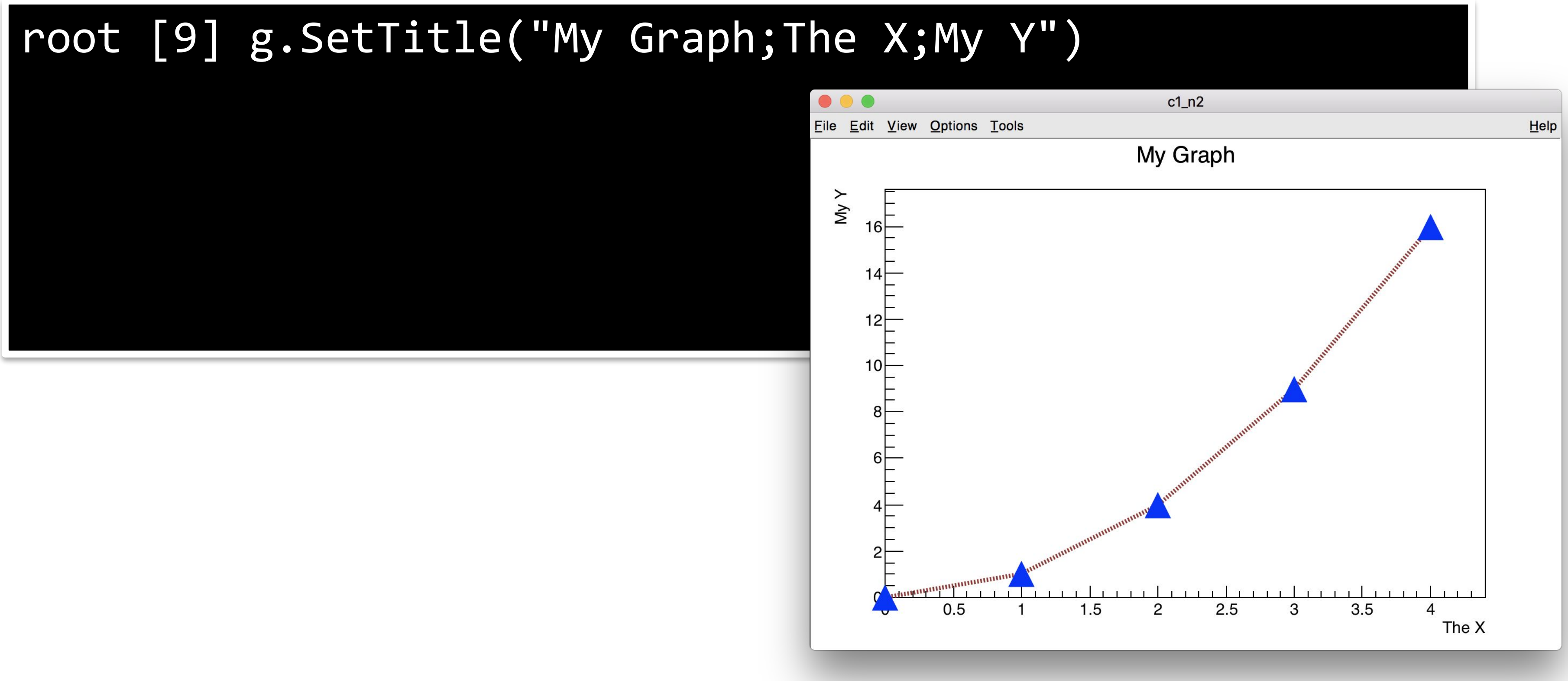


My First Graph

```
root [7] g.SetLineWidth(2)
root [8] g.SetLineStyle(3)
```

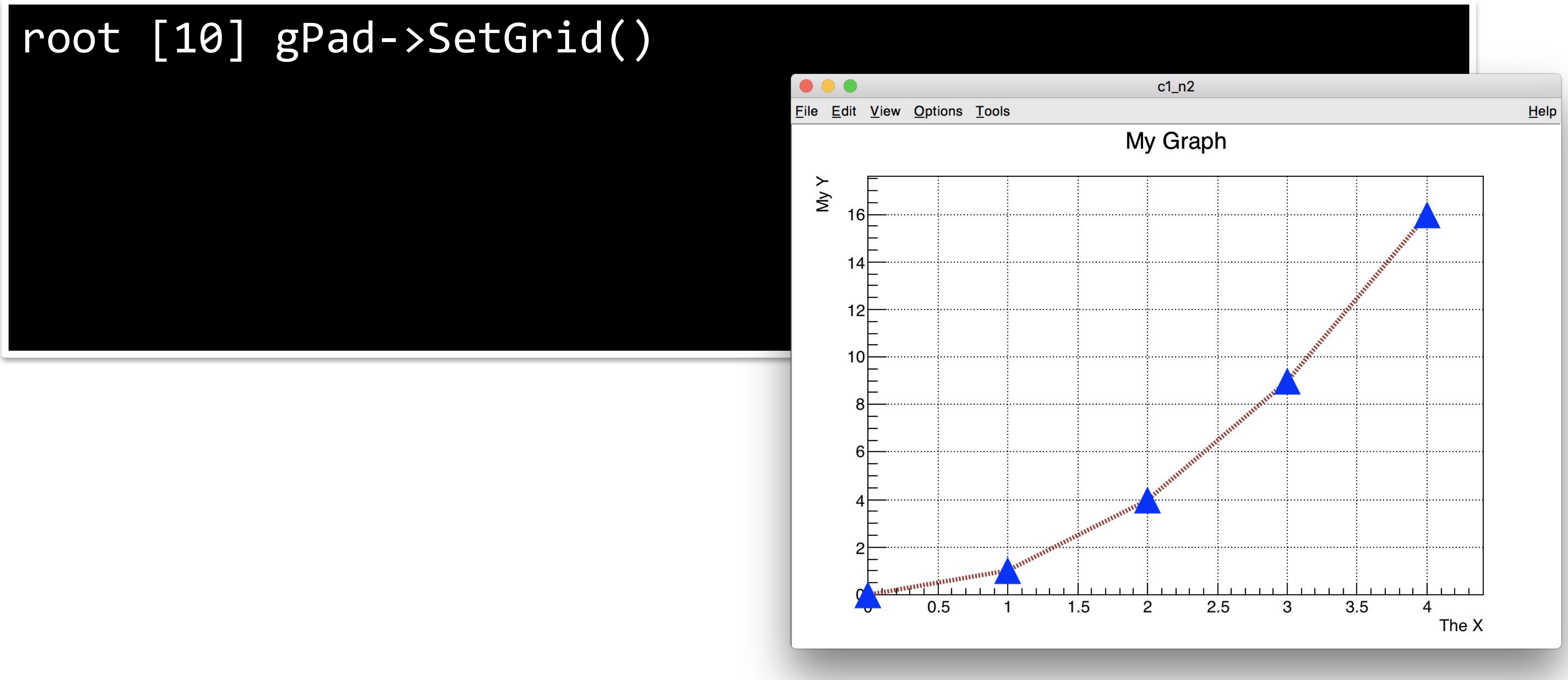


My First Graph



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

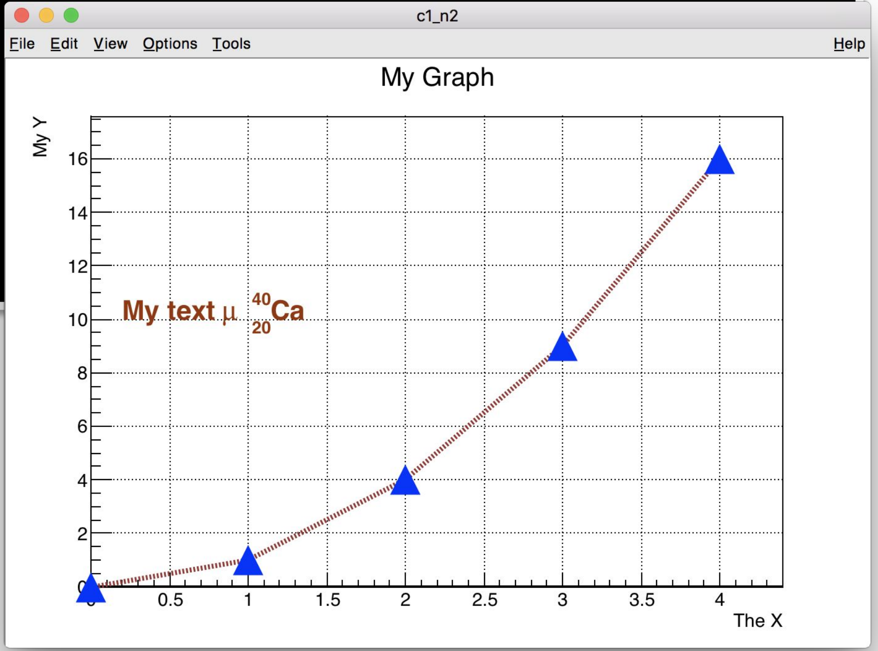
My First Graph



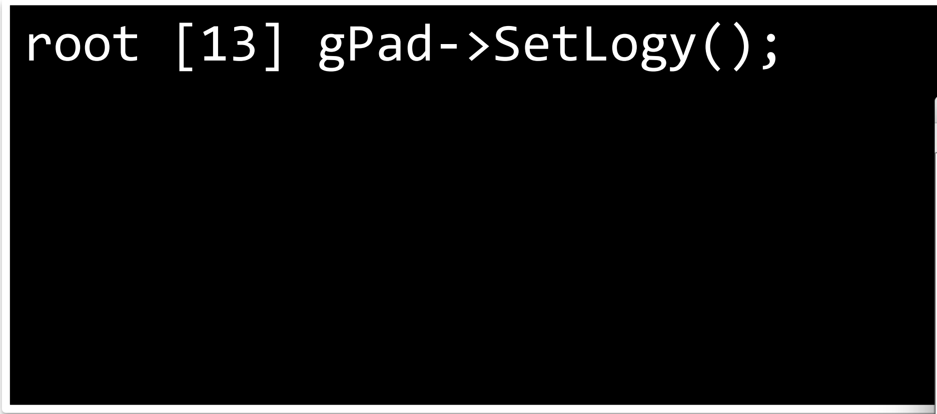
```
root [10] gPad->SetGrid()
```

My First Graph

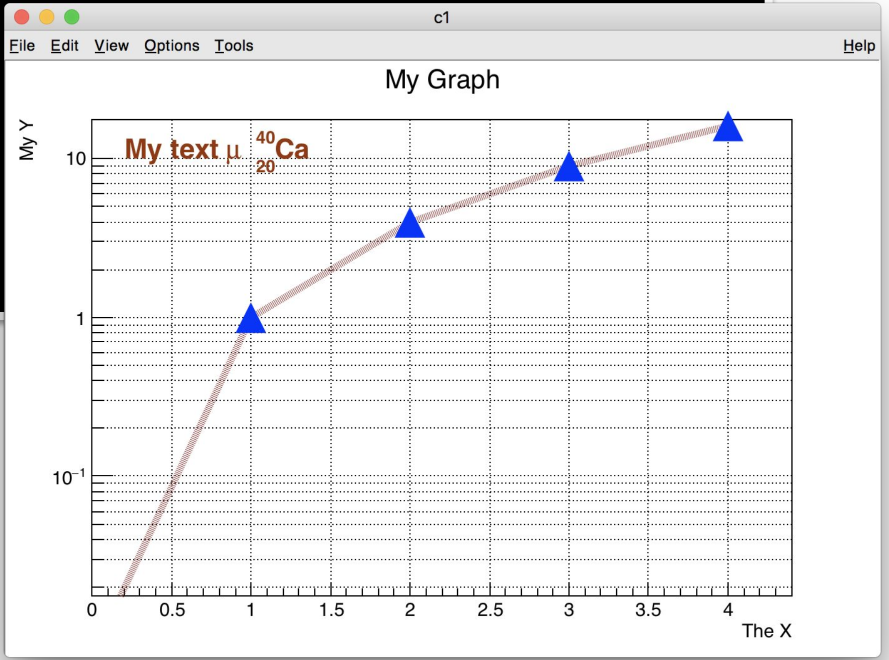
```
root [10] auto txt = "#color[804]{My text #mu {}^{40}_{20}Ca}"
root [11] TLatex l(.2, 10, txt)
root [12] l.Draw()
```



My First Graph



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



myFirstGraph.C

Time for Exercises!

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

Graphics functionalities: a more structured approach

Features of a Good Plot

Every plot should be **self-contained** and deliver a **clear message**, even if extracted from the publication in which it's shown.

To achieve this goal the mandatory pieces of a plot are:

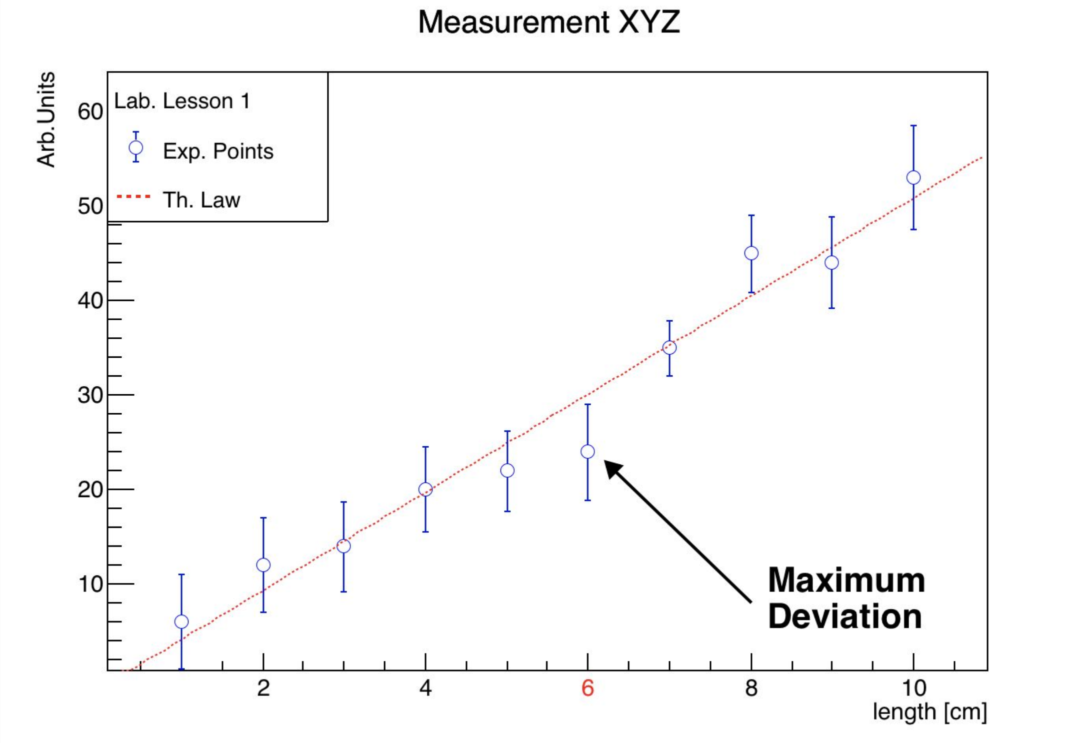
1. **The data**, of course. The best representation should be chosen to avoid any misinterpretation.
2. **The plot title** which should summarize clearly what the data are.
3. **The axis titles**. The X and Y titles should be properly titled with the variable name and its unit.
4. **The axis** themselves. They should be clearly labelled to avoid any ambiguity.
5. **The legend**. It should explain clearly the various curves on the plot.
6. **Annotations** highlighting specific details on the plot.

"Good Plot" example

The next slides will detailed how to build this plot step by step.

Each exercise will be marked as:

⇒ Something to do



The data set

```
void macro1(){
   // The number of points in the data set
   const int n_points = 10;
   // The values along X and Y axis
   double x_vals[n_points] = {1,2,3,4,5,6,7,8,9,10};
   double y_vals[n_points] = {6,12,14,20,22,24,35,45,44,53};
   // The errors on the Y axis
   double y_errs[n_points] = {5,5,4.7,4.5,4.2,5.1,2.9,4.1,4.8,5.43};

   // Instance of the graph
   auto graph = new TGraphErrors(n_points,x_vals,y_vals,nullptr,y_errs);
}
```

This code creates the **data set**.

⇒ Create a macro called "macro1.C" and execute it.

The data drawing

As this graph has error bars along the Y axis an obvious choice will be to draw it as an **error bars plot**. The command to do it is:

```
graph->Draw("APE");
```

The Draw() method is invoked with three options:

- "A" the **axis** coordinates are automatically computed to fit the graph data
- "P" **points** are drawn as marker
- "E" the **error bars** are drawn

⇒ Add this command to the macro and execute it again.

⇒ Try to change the options (e.g. “APEL”, “APEC”, “APE4”).

Customizing the data drawing

Graphical attributes can be set to customize the visual aspect of the plot. Here we change the **marker style** and **color**.

```
// Make the plot esthetically better
graph->SetMarkerStyle(kOpenCircle);
graph->SetMarkerColor(kBlue);
graph->SetLineColor(kBlue);
```

⇒ Add this code to the macro. Notice the visual changes.

⇒ Play a bit with the possible values of the attributes.

Fitting and customizing the fit drawing

The data set we built up looks very linear. It could be interesting to **fit it with a line** to see what the linear law behind this data set could be.

```
// Define a linear function
auto f = new TF1("Linear law","[0]+x*[1]",.5,10.5);
// Let's make the function line nicer
f->SetLineColor(kRed);
f->SetLineStyle(2);
// Fit it to the graph
graph->Fit(f);
```

The function "**`f`**" graphical aspect is customized the same way the graph was before.

⇒ Add this code to the macro. Execute it again.

⇒ Notice the output given by the **`Fit`** method.

⇒ Play with the graphical attributes for the "**`f`**" function.

Add a Function

The data set we built up looks very linear. It could be interesting to **compare it with a line** to see what the linear law behind this data set could be.

```
// Define a linear function
auto f = new TF1("Linear law","[0]+x*[1]",.5,10.5);
// Let's make the function line nicer
f->SetLineColor(kRed);
f->SetLineStyle(2);
// Set parameters
f->SetParameters(-1,5);
f->Draw("Same")
```

The function "`f`" graphical aspect is customized the same way the graph was before.

⇒ Add this code to the macro. Execute it again.

⇒ Play with the graphical attributes for the "`f`" function.

Plot Titles

The graph was created without any titles. It is time to define them. The following command **define the three titles** (separated by a ";") in one go. The format is:

"Main title ; x axis title ; y axis title"

```
graph->SetTitle("Measurement XYZ;length [cm];Arb.Units");
```

The axis titles can be also set individually:

```
graph->GetXaxis()->SetTitle("length [cm]");
graph->GetYaxis()->SetTitle("Arb.Units");
```

⇒ Add this code to the macro. You can choose one or the other way.

⇒ Play with the text. You can even try to add TLatex special characters.

Axis labelling (1)

The axis labels must be very **clear**, **unambiguous**, and **adapted to the data**.

- ROOT by default provides a powerful mechanism of **labelling optimisation.**
- Axis have three levels of divisions: Primary, Secondary, Tertiary.
- The axis labels are on the Primary divisions.
- The method `SetNdivisions` change the number of axis divisions

```
graph->GetXaxis()->SetNdivisions(10, 5, 0);
```

⇒ Add this command. nothing changes because they are the default values :-)

⇒ Change the number of primary divisions to 20. Do you get 20 divisions ? Why ?

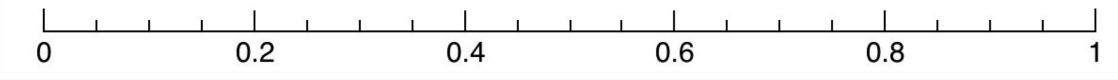
Axis labelling (2)

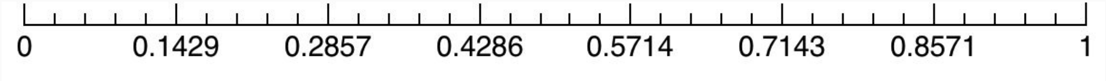
The number of divisions passed to the `SetNdivisions` method are **by default optimized** to get a comprehensive labelling. Which means that the value passed to `SetNDivisions` is a maximum number not the exact value we will get.

To turn off the optimisation the fourth parameter of `SetNDivisions` to `kFALSE`

⇒ try it

As an example, the two following axis both have been made with **seven** primary divisions. The first one is optimized and the second one is not. The second one is not really clear !!





Nevertheless, in some cases, it is useful to have non optimized labelling.

Other kinds of axis

In addition to the normal linear numeric axis axis ROOT provides also:

- **Alphanumeric labelled axis**. They are produced when a histogram with alphanumeric labels is plotted.
- **Logarithmic axis**. Set with `gPad->SetLogx()` ⇒ try it (`gPad->SetLogy()` for the Y axis)
- **Time axis**.

Fine tuning of axis labels

When a specific **fine tuning** of an individual label is required, for instance changing its color, its font, its angle or even changing the label itself the method **`ChangeLabel`** can be used on any axis.

⇒ For instance, in our example, imagine we would like to highlight more the X label with the maximum deviation by putting it in red. It is enough to do:

```
graph->GetXaxis()->ChangeLabel(3,-1,-1,-1,kRed);
```

Legend (1)

ROOT provides a powerful class to build a legend: **`TLegend`**. In our example the legend is build as follow:

```
auto legend = new TLegend(.1,.7,.3,.9,"Lab. Lesson 1");
legend->AddEntry(graph,"Exp. Points","PE");
legend->AddEntry(f,"Th. Law","L");
legend->Draw();
```

⇒ Add this code, try it and play with the options

- The `TLegend` constructor defines the **legend position and its title**.
- Each legend item is added with method **`AddEntry`** which specify
  - **an object to be part of the legend** (by name or by pointer),
  - the **text of the legend**
  - the **graphics attributes to be legended**. "L" for TAttLine, "P" for TAttMarker, "E" for error bars etc ...

Legend (2)

ROOT also provides an **automatic way to produce a legend** from the graphics objects present in the pad. The method **`TPad::BuildLegend()`**.

⇒ In our example try **`gPad->BuildLegend()`**

This is a quick way to proceed but for a plot ready for publication it is recommended to use the method previously described.

⇒ Keep only the legend defined in the previous slide

Annotations (1)

Often the various parts of a plot we already saw are not enough to convey the complete message this plot should. Additional **annotations** elements are needed **to complete and reinforce the message** the plot should give.

ROOT provides a collection of basic graphics primitives allowing to draw such annotations. Here a non exhaustive list:

- `TText` and `TLatex` to draw text.
- `TArrow` to draw all kinds of arrows
- `TBox`, `TEllipse` to draw boxes and ellipses.
- Etc ...

Annotations (2)

In our example we added an annotation pointing the "maximum deviation". It consists of a simple arrow and a text:

```
// Draw an arrow on the canvas
auto arrow = new TArrow(8,8,6.2,23,0.02,"|>");
arrow->SetLineWidth(2);
arrow->Draw();

// Add some text to the plot
auto text = new TLatex(8.2,7.5,"#splitline{Maximum}{Deviation}");
text->Draw();
```

⇒ Try this code adding it in the macro.

⇒ Notice changing the canvas size does not affect the pointed position.

The complete macro

```
void macro1() {
   // The values and the errors on the Y axis
   const int n_points=10;
   double x_vals[n_points] = {1,2,3,4,5,6,7,8,9,10};
   double y_vals[n_points] = {6,12,14,20,22,24,35,45,44,53};
   double y_errs[n_points] = {5,5,4.7,4.5,4.2,5.1,2.9,4.1,4.8,5.43};

   // Instance of the graph
   auto graph = new TGraphErrors(n_points,x_vals,y_vals,nullptr,y_errs);
   graph->SetTitle("Measurement XYZ;length [cm];Arb.Units");

   // Make the plot esthetically better
   graph->SetMarkerStyle(kOpenCircle);
   graph->SetMarkerColor(kBlue);
   graph->SetLineColor(kBlue);

   // The canvas on which we'll draw the graph
   auto  mycanvas = new TCanvas();

   // Draw the graph !
   graph->Draw("APE");
```

```

   // Define a linear function
   auto f = new TF1("Linear law","[0]+x*[1]",.5,10.5);

   // Let's make the function line nicer
   f->SetLineColor(kRed);
   f->SetLineStyle(2);

   // Fit it to the graph and draw it
   graph->Fit(f);

   // Build and Draw a legend
   auto legend = new TLegend(.1,.7,.3,.9,"Lab. Lesson 1");
   legend->AddEntry(graph,"Exp. Points","PE");
   legend->AddEntry(f,"Th. Law", "L");
   legend->Draw();

   // Draw an arrow on the canvas
   auto arrow = new TArrow(8,8,6.2,23,0.02,"|>");
   arrow->SetLineWidth(2);
   arrow->Draw();
   // Add some text to the plot and highlight the 3rd label
    auto text = new TLatex(8.2,7.5,"#splitline{Maximum}{Deviation}");
   text->Draw();
   graph->GetXaxis()->ChangeLabel(3,-1,-1,-1,kRed);
}
```

The complete macro

```
void macro1() {
  // The values and the errors on the Y axis
  const int n_points=10;
  double x_vals[n_points] = {1,2,3,4,5,6,7,8,9,10};
  double y_vals[n_points] = {6,12,14,20,22,24,35,45,44,53};
  double y_errs[n_points] = {5,5,4.7,4.5,4.2,5.1,2.9,4.1,4.8,5.43};

  // Instance of the graph
  auto graph = new TGraphErrors(n_points,x_vals,y_vals,nullptr,y_errs);
  graph->SetTitle("Measurement XYZ;length [cm];Arb.Units");

  // Make the plot esthetically better
  graph->SetMarkerStyle(kOpenCircle);
  graph->SetMarkerColor(kBlue);
  graph->SetLineColor(kBlue);

  // The canvas on which we'll draw the graph
  auto  mycanvas = new TCanvas();

  // Draw the graph !
  graph->Draw("APE");
```

```
  // Define a linear function
  auto f = new TF1("Linear law","[0]+x*[1]",.5,10.5);

  // Let's make the function line nicer
  f->SetLineColor(kRed);
  f->SetLineStyle(2);

  // Set the function parameters
  f->SetParameters(-1,5);
  f->Draw("Same");

  // Build and Draw a legend
  auto legend = new TLegend(.1,.7,.3,.9,"Lab. Lesson 1");
  legend->AddEntry(graph,"Exp. Points","PE");
  legend->AddEntry(f,"Th. Law", "L");
  legend->Draw();

  // Draw an arrow on the canvas
  auto arrow = new TArrow(8,8,6.2,23,0.02,"|>");
  arrow->SetLineWidth(2);
  arrow->Draw();
  // Add some text to the plot and highlight the 3rd label
   auto text = new TLatex(8.2,7.5,"#splitline{Maximum}{Deviation}");
  text->Draw();
  graph->GetXaxis()->ChangeLabel(3,-1,-1,-1,kRed);
}
```

Time for Full Solutions!

https://github.com/root-project/training/tree/master/BasicCourse/Exercises/Graphics

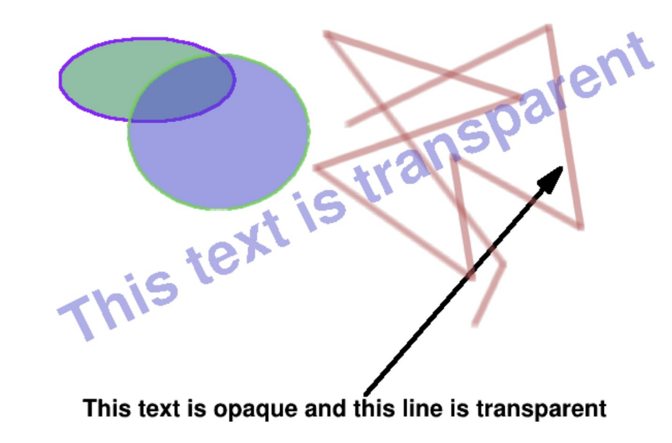
macro1NoFit.C

Graphics Styles

- ROOT provides **"graphics styles"** to define the **general look** of plots.
- The **current style** can be accessed via a global pointer : **`gStyle`**.
- The styles are managed by the class TStyle.
- ROOT users can define their own style. For example an experiment can have its own style to make sure the plots produced by its scientists have the same look.
- There is a series of predefined style "Plain", "Bold", "Pub, "Modern" etc... the default style is "Modern".

Colors

- **The transparency** is set via similar setters but with the keyword "Alpha" at the end. For instance `histo->SetFillColorAlpha(kBlue,0.35)` will set the color of the histogram histo to red with and alpha channel equal to 0.35 (0 = fully transparent and 1 = fully opaque).



Note: The transparency is available on all platforms when the flag `OpenGL.CanvasPreferGL` is set to 1 in `$ROOTSYS/etc/system.rootrc`, or on Mac with the Cocoa backend. On the file output it is visible with PDF, PNG, Gif, JPEG, SVG ...

2D plots with color map (1)

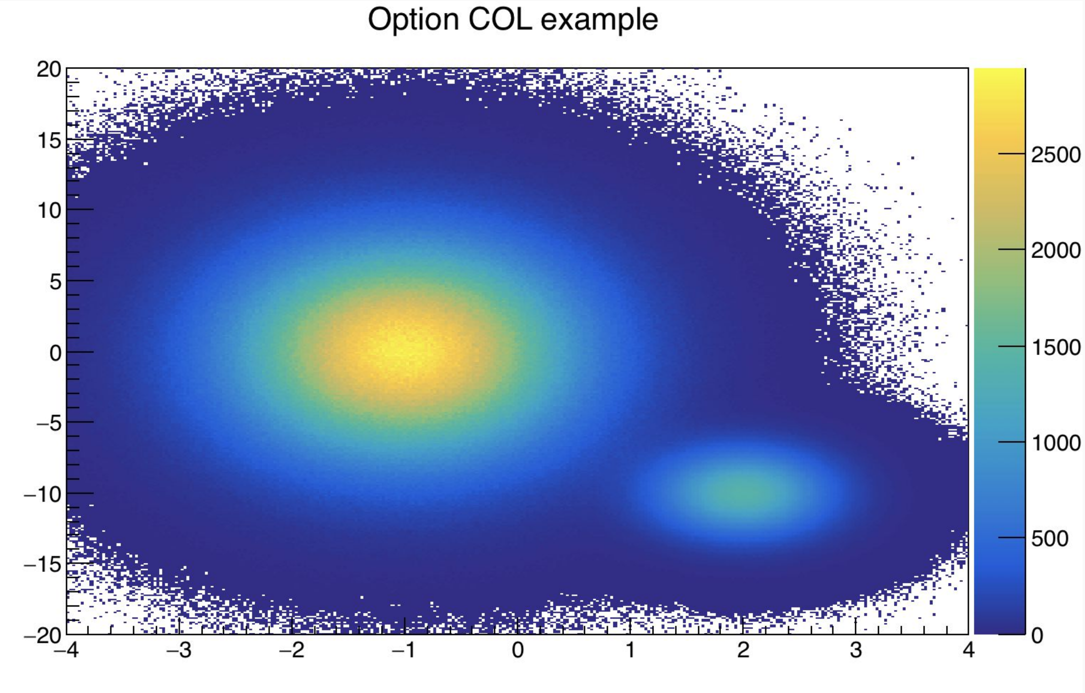
A very common way to represent 2D histograms is the **color plot**. We will use the following macro to illustrate this kind of plot.

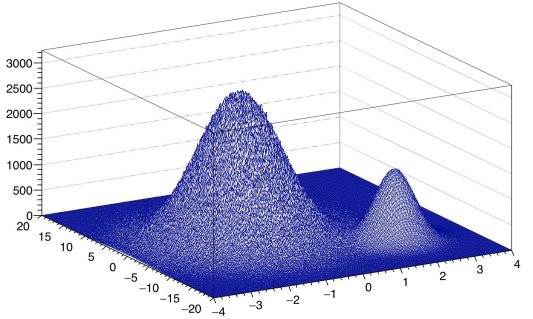
```
void macro2(){
   TH2F *h = new TH2F("h","Option COL example ",300,-4,4,300,-20,20);
   h->SetStats(0);
   h->SetContour(200);
   float px, py;
   for (int i = 0; i < 25000000; i++) {
      gRandom->Rannor(px,py);
      h->Fill(px-1,5*py);
      h->Fill(2+0.5*px,2*py-10.,0.1);
   }
   h->Draw("colz");
}
```

https://github.com/root-project/training/tree/master/BasicCourse/Exercises/Graphics
macro2.C

2D plots with color map (2)

The previous macro generates the following output which is a plot with two smooth 2D gaussian:

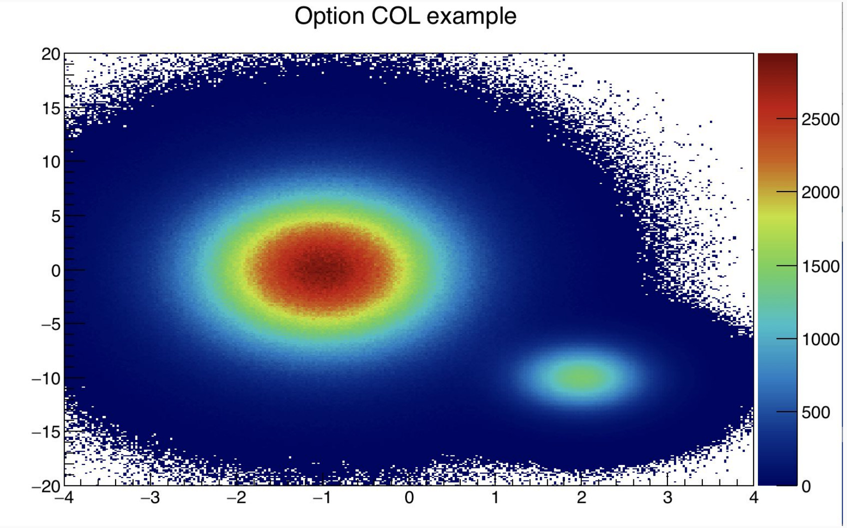




The default ROOT color map (kBird) render perfectly the smoothness of the dataset.

Danger of the Rainbow color map (1)

If instead of the default color maps we use the Rainbow one we get:

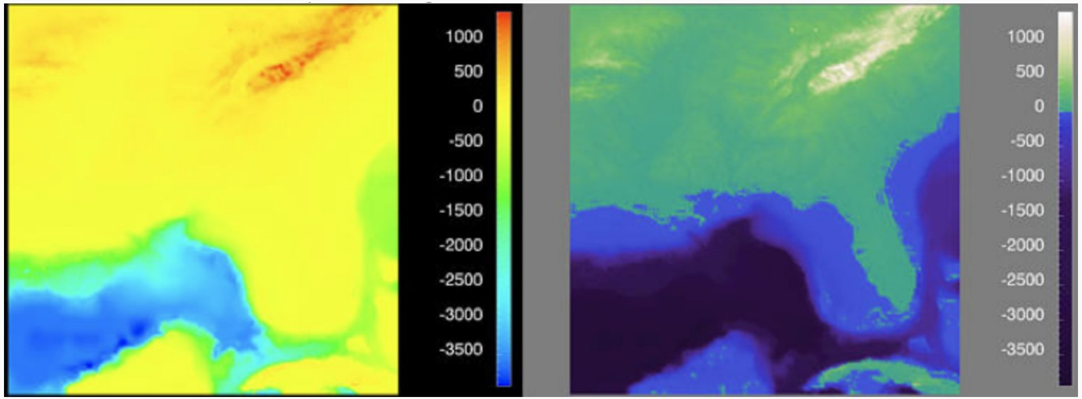


The reasons why this is not a good is explained in detail here.

For instance immediately see some "structure" appearing on the plot which does not exist in the data: just look at this yellow circle.

Danger of the Rainbow color map (2)

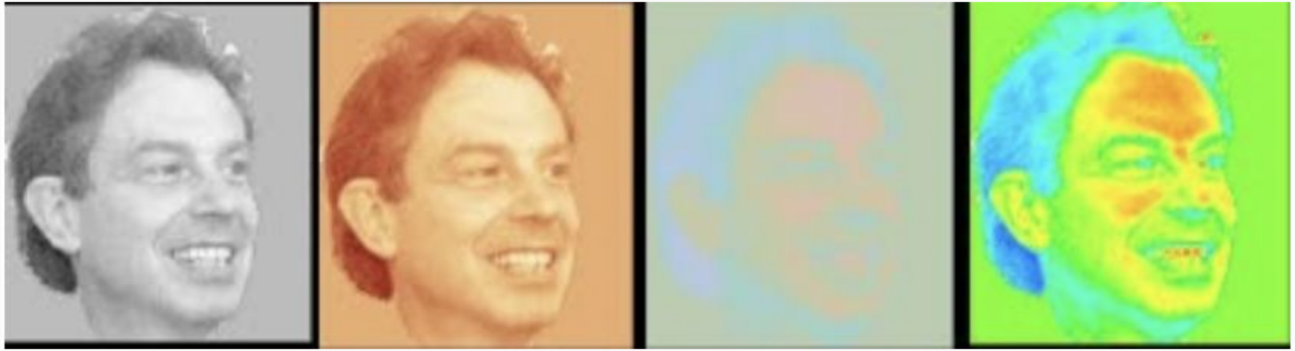
Another example taken from the article mentioned earlier illustrates even better why the Rainbow color map should be avoided.



These two panels show the same data, but with different colormaps. On the left, the "Rainbow" colormap provides a very colorful and vibrant image, however, it masks significant features in the data, and emphasizes less important ones.

Danger of the RainBow color map (3)

A Quick Visual Method for Evaluating color maps has been exposed in the "The 'Which Blair Project:" (Rogowitz, Bernice and Alan Kalvin)



The idea is to use a well know photography of a face presented with different colormaps. The colormaps distorting the image are not good. Clearly the Rainbow one (on the right) distorts the image !

The ROOT standard color maps (1)

ROOT provides 62 color maps (including Rainbow because people like it). Most are monotonic, but several may have quite small luminance variation from max to min.

Displaying the grayscale equivalents of a color map may help people selecting ones having a large and monotonic luminance ranges.

It might be interesting to look at the ROOT color scales in this way. To do that it is enough to turn the grayscale mode on before drawing the histogram:

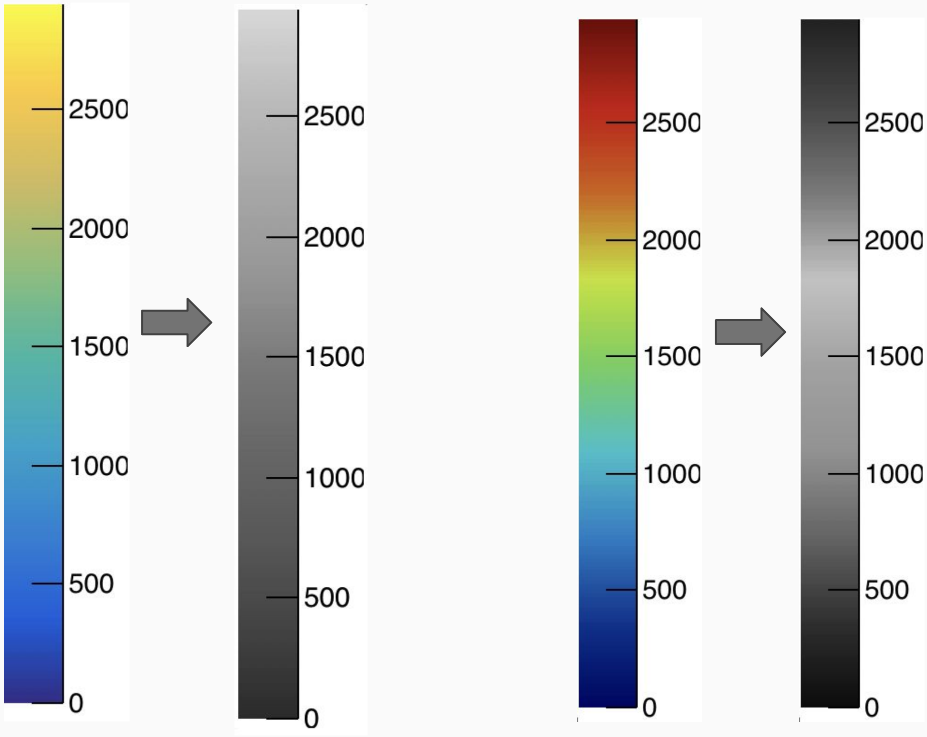
```
canvas->SetGrayscale();
```

https://github.com/root-project/training/tree/master/BasicCourse/Exercises/Graphics

macro2.C   -> Create a TCanvas, draw the histogram and activate the gray scale!

The ROOT standard color maps (2)

Here we show the grayscale equivalent of the two color maps we used before:



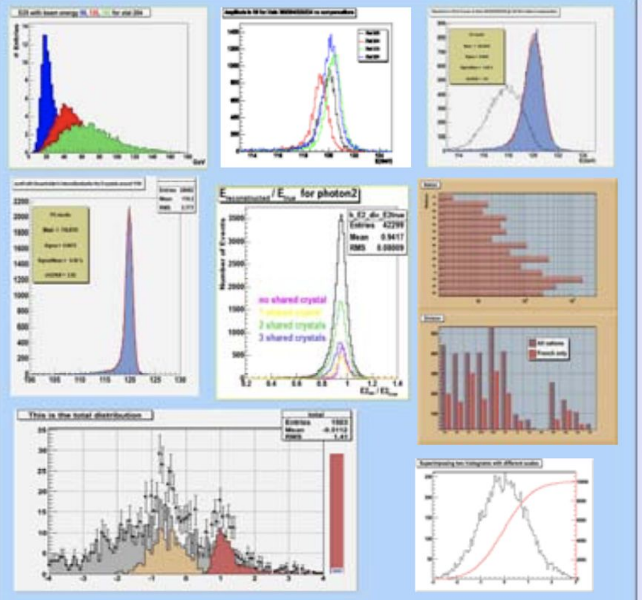
This is another good test showing the validity of a color map. In gray scale the Rainbow color map shows its luminance issue as the minimum and maximum values appear the same.

ROOT visualization techniques

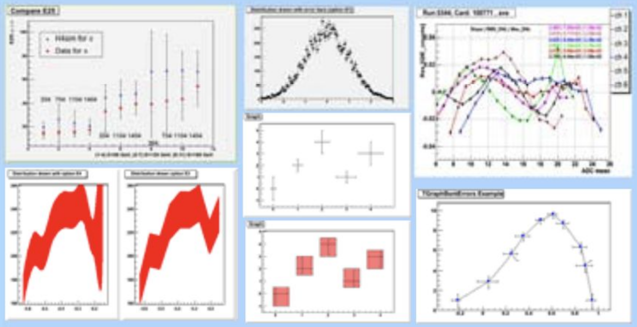
In this "*Introduction to the ROOT graphics capabilities*" we have seen a small fraction of the ROOT graphics capabilities. In the next four slides we will present a quick snapshot of what ROOT provides to visualise 2, 3, 4 and N variables data set.

2 variables visualization techniques

2 variables visualization techniques are used to display Trees, Ntuples, 1D histograms, functions y=f(x), graphs …

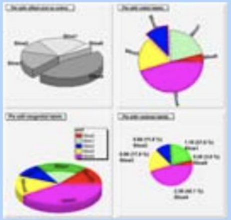


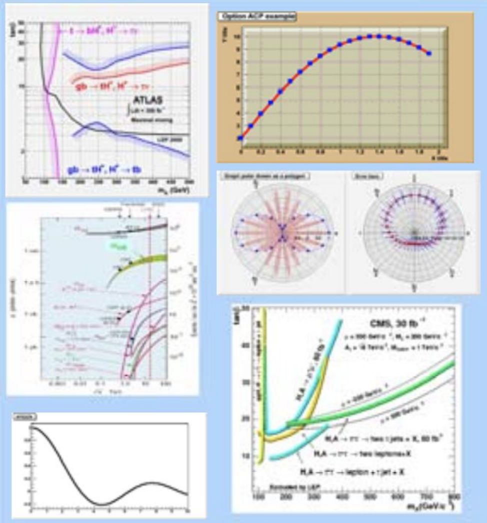
Bar charts and lines are a common way to represent 1D histograms.



Errors can be represented as bars, band, rectangles. They can be symmetric, asymmetric or bent. 1D histograms and graphs can be drawn that way.

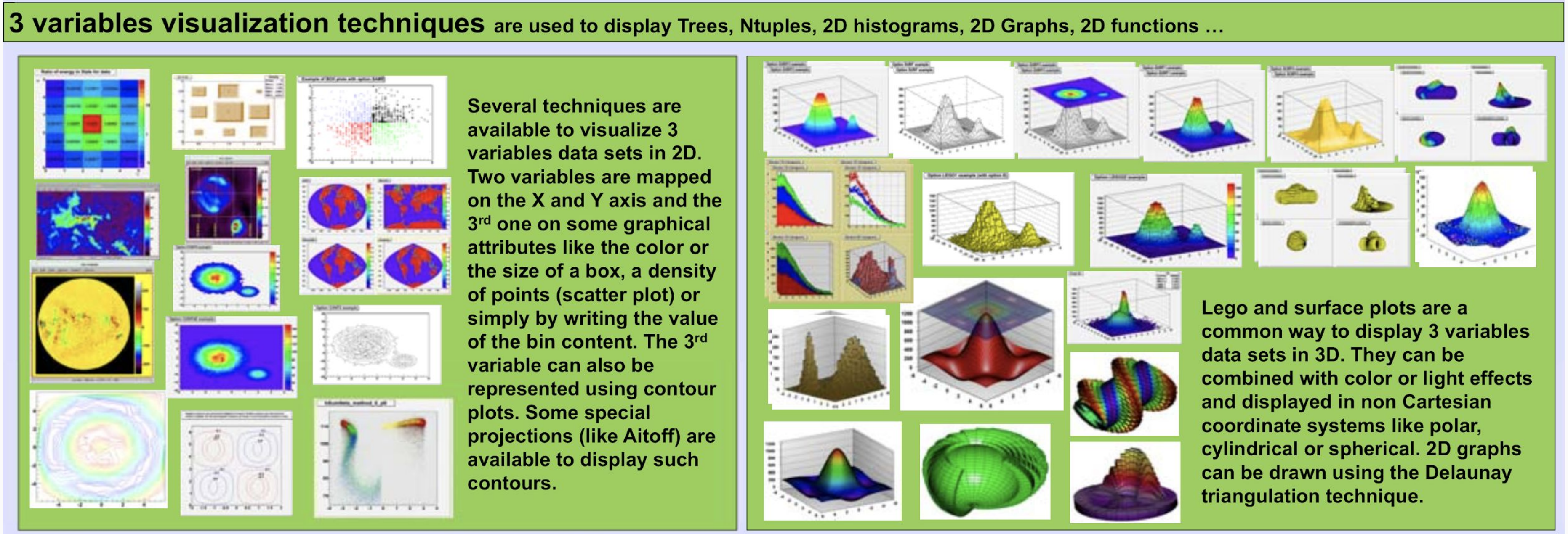
Pie charts can be used to visualize 1D histograms. They also can be created from a simple mono dimensional vector.





Graphs can be drawn as simple lines, like functions. They can also visualize exclusion zones or be plotted in polar coordinates.

3 variables visualization techniques

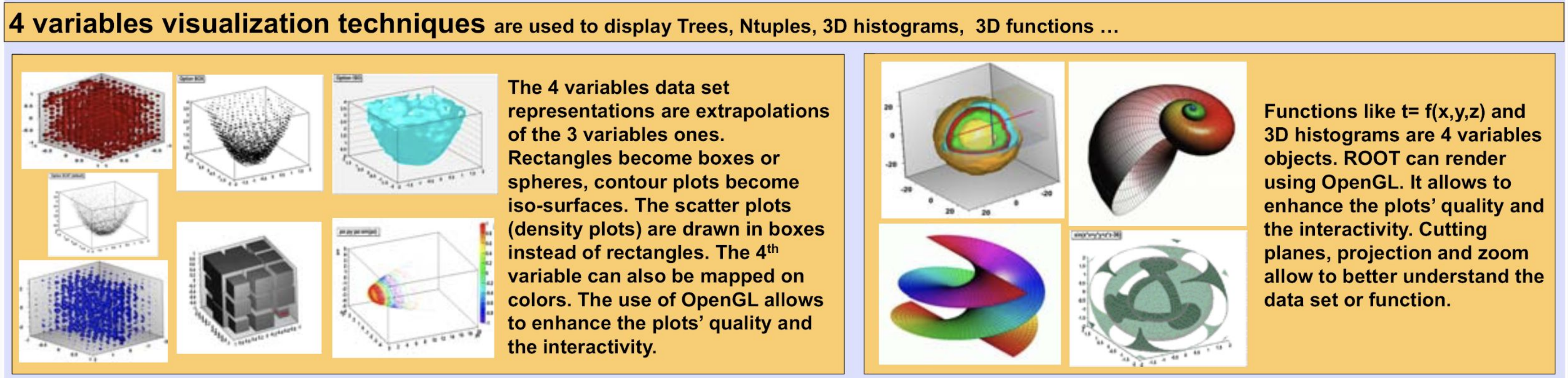


3 variables visualization techniques are used to display Trees, Ntuples, 2D histograms, 2D Graphs, 2D functions …

Several techniques are available to visualize 3 variables data sets in 2D. Two variables are mapped on the X and Y axis and the 3rd one on some graphical attributes like the color or the size of a box, a density of points (scatter plot) or simply by writing the value of the bin content. The 3rd variable can also be represented using contour plots. Some special projections (like Aitoff) are available to display such contours.

Lego and surface plots are a common way to display 3 variables data sets in 3D. They can be combined with color or light effects and displayed in non Cartesian coordinate systems like polar, cylindrical or spherical. 2D graphs can be drawn using the Delaunay triangulation technique.

4 variables visualization techniques

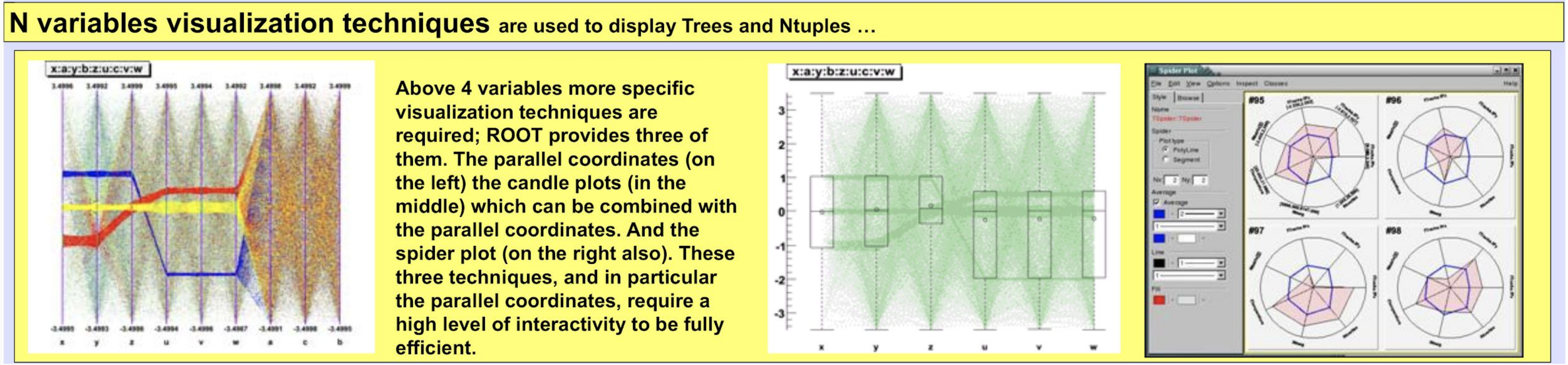


4 variables visualization techniques are used to display Trees, Ntuples, 3D histograms, 3D functions …

The 4 variables data set representations are extrapolations of the 3 variables ones. Rectangles become boxes or spheres, contour plots become iso-surfaces. The scatter plots (density plots) are drawn in boxes instead of rectangles. The 4th variable can also be mapped on colors. The use of OpenGL allows to enhance the plots' quality and the interactivity.

Functions like t= f(x,y,z) and 3D histograms are 4 variables objects. ROOT can render using OpenGL. It allows to enhance the plots' quality and the interactivity. Cutting planes, projection and zoom allow to better understand the data set or function.

N variables visualization techniques



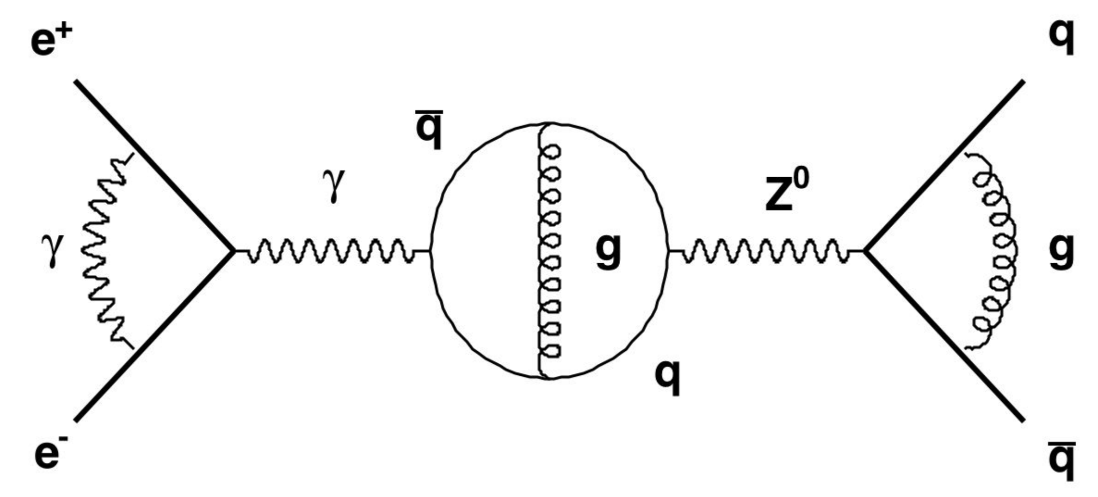
N variables visualization techniques are used to display Trees and Ntuples …

Above 4 variables more specific visualization techniques are required; ROOT provides three of them. The parallel coordinates (on the left) the candle plots (in the middle) which can be combined with the parallel coordinates. And the spider plot (on the right also). These three techniques, and in particular the parallel coordinates, require a high level of interactivity to be fully efficient.

One more thing: Feynman diagrams

In theoretical physics, Feynman diagrams are pictorial representations of the mathematical expressions describing the behavior of subatomic particles.

These diagrams can also be produced by ROOT thanks to a serie of specific graphics primitives as shown in this example.



Take Home Messages

Every plot should be self-contained and deliver a clear message, even if extracted from the publication in which it's shown.

ROOT provides all the tools to make "publication ready" plots but as any tools it can be misused and some basic rules should be kept in mind.

ROOT graphics allow **great flexibility** thanks to the closeness of the data manipulated and their graphical representation.

The JSROOT Graphics Interface

Learning Objectives

- Learn the context into which JSROOT was developed
- Learn the capabilities and limitations of JSROOT
- Understand the technology behind the interface
- Learn more advanced functionalities, like displaying geometries

Introduction

- The JSROOT project intends to read any ROOT file and display its content in Web browsers, using its own implementation of the ROOT graphics
- It only requires a web browser to display and interact with most of the ROOT graphical objects
- Used in conjunction with THttpServer, it can be used (for example) to monitor a running ROOT session

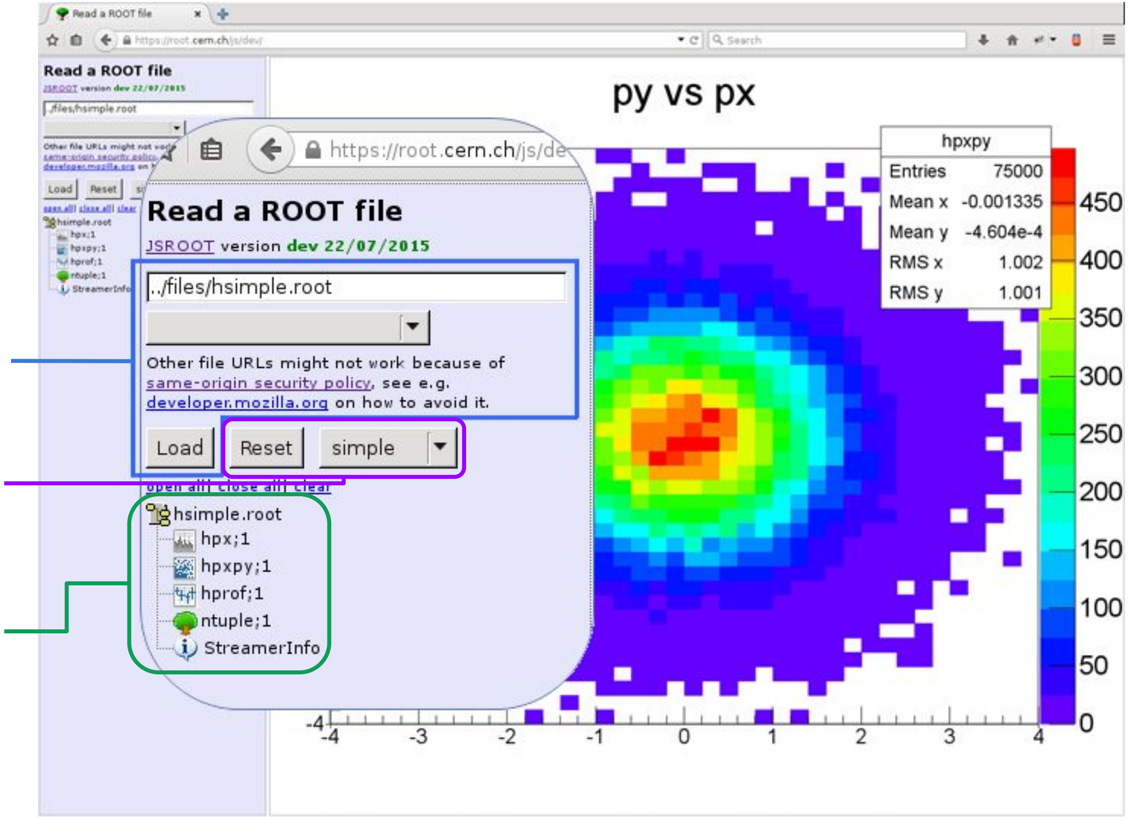
Supported ROOT classes

- Basic primitives
  - TPolyLine, TEllipse, TArrow, TPolyMarker3D, TSpline
- Histograms
  - TH1, TH2, TH2Poly, TH3, TProfile, TProfile2D, THStack
- Graphs and functions
  - TGraph, TGraphErrors, TGraphAsymmErrors, TGraphPolar, TMultiGraph, TGraph2D, TF1, TF2
- Text, Latex, Math
  - TLatex, TMathText, TPaveText, TPaveStats, TPaveLabel
- But also TCanvas, TPad, TLegend, and TTree

JSROOT in Action

https://root.cern.ch/js/latest/

User Interface



Read a ROOT file

JSROOT version dev 22/07/2015

../files/hsimple.root

Other file URLs might not work because of same-origin security policy, see e.g. developer.mozilla.org on how to avoid it.

Load | Reset | simple

open all | close all | clear

```
hsimple.root
├ hpx;1
├ hpxpy;1
├ hprof;1
├ ntuple;1
└ StreamerInfo
```

py vs px

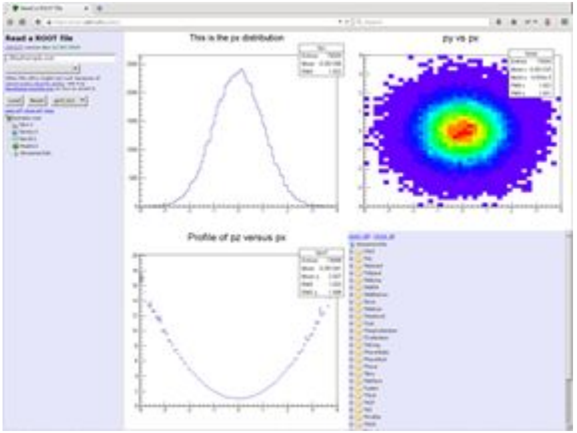
hpxpy	
Entries	75000
Mean x	-0.001335
Mean y	-4.604e-4
RMS x	1.002
RMS y	1.001

File Loading

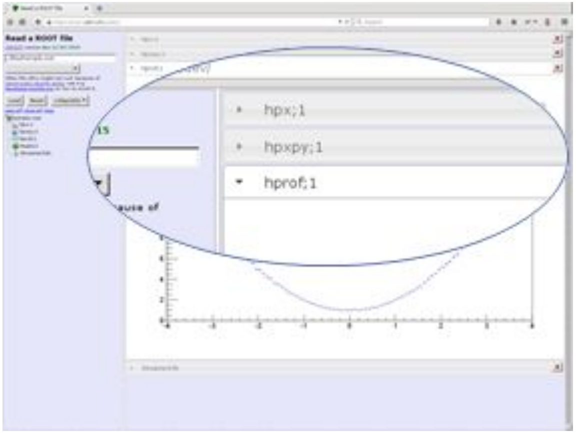
Layout Selection

File Content

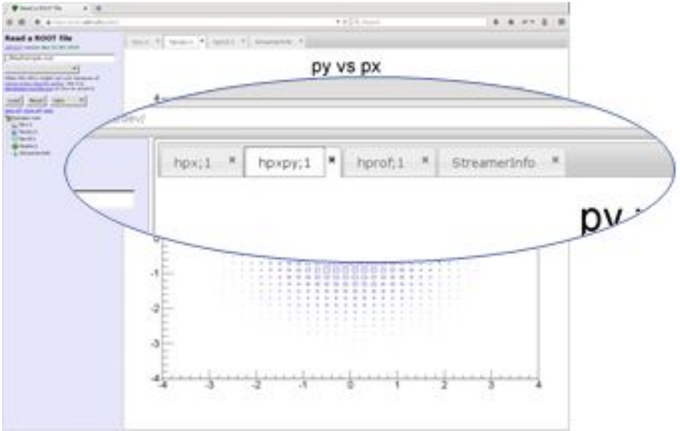
Layout



Grid



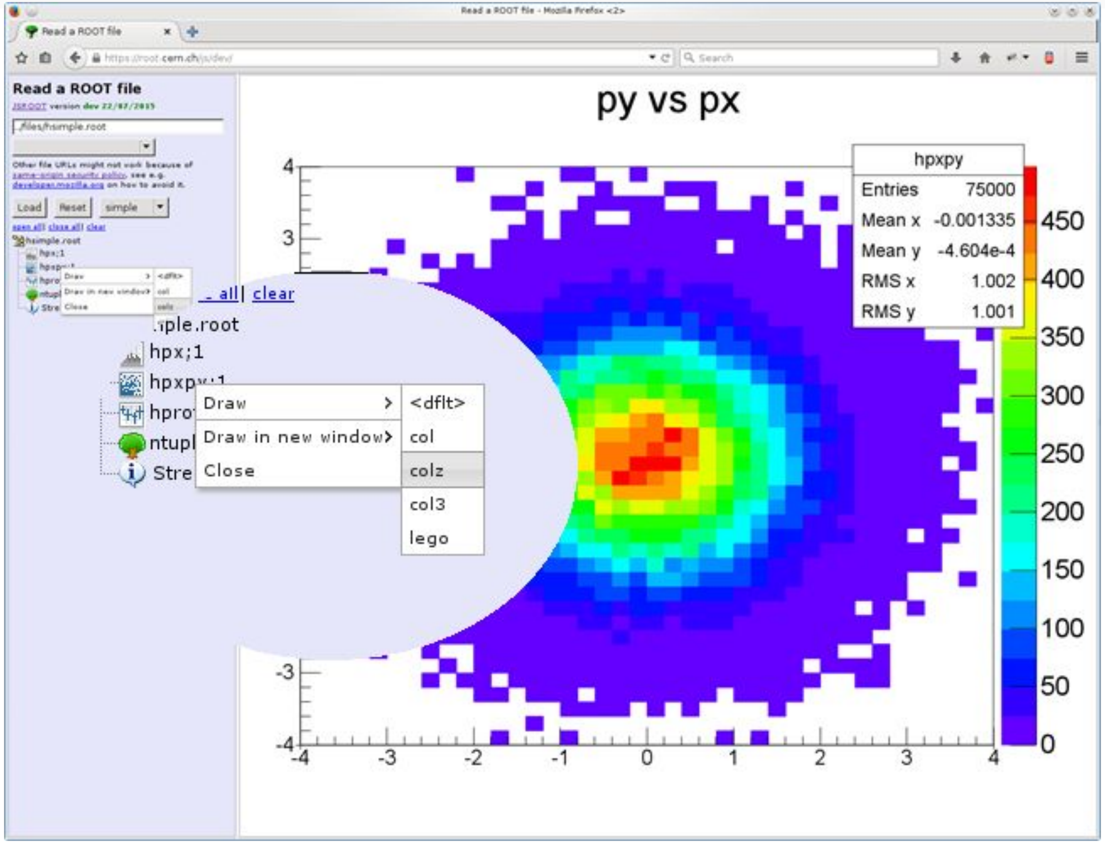
Collapsible



Tabs

Context Menu

Context menu with drawing options



Read a ROOT file

JSROOT version dev 22/07/2015

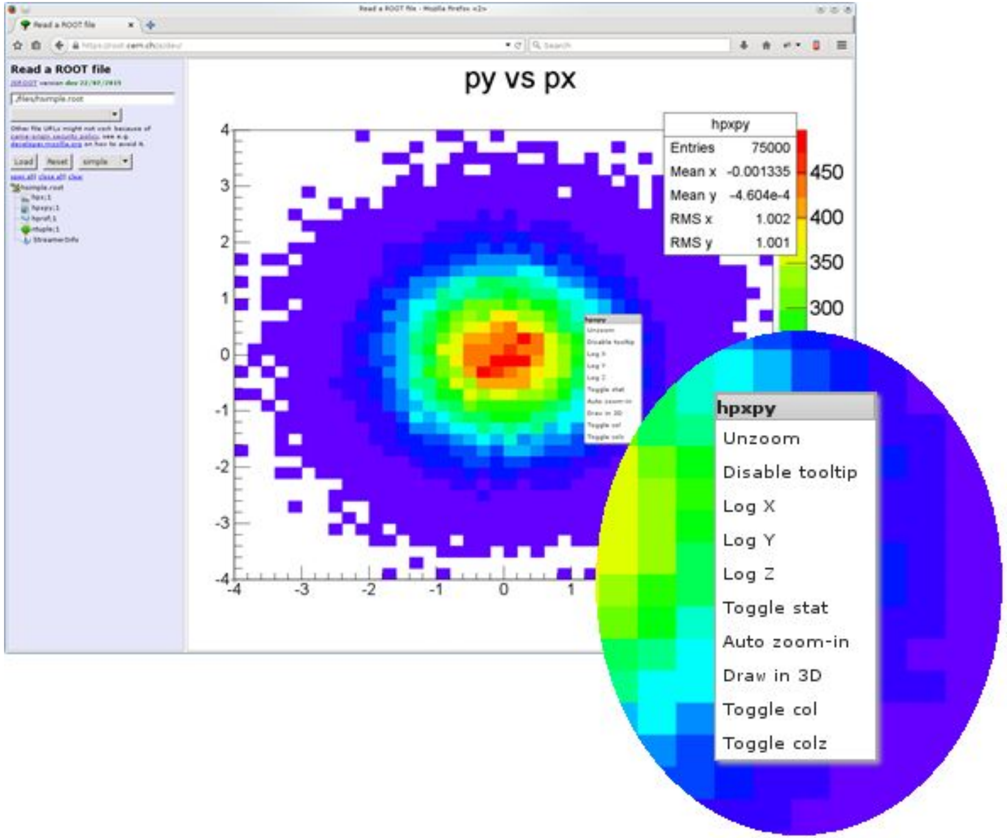
../files/hsimple.root

Other file URLs might not work because of same-origin security policy, see e.g. developer.mozilla.org on how to avoid it.

hpxpy	
Entries	75000
Mean x	-0.001335
Mean y	-4.604e-4
RMS x	1.002
RMS y	1.001

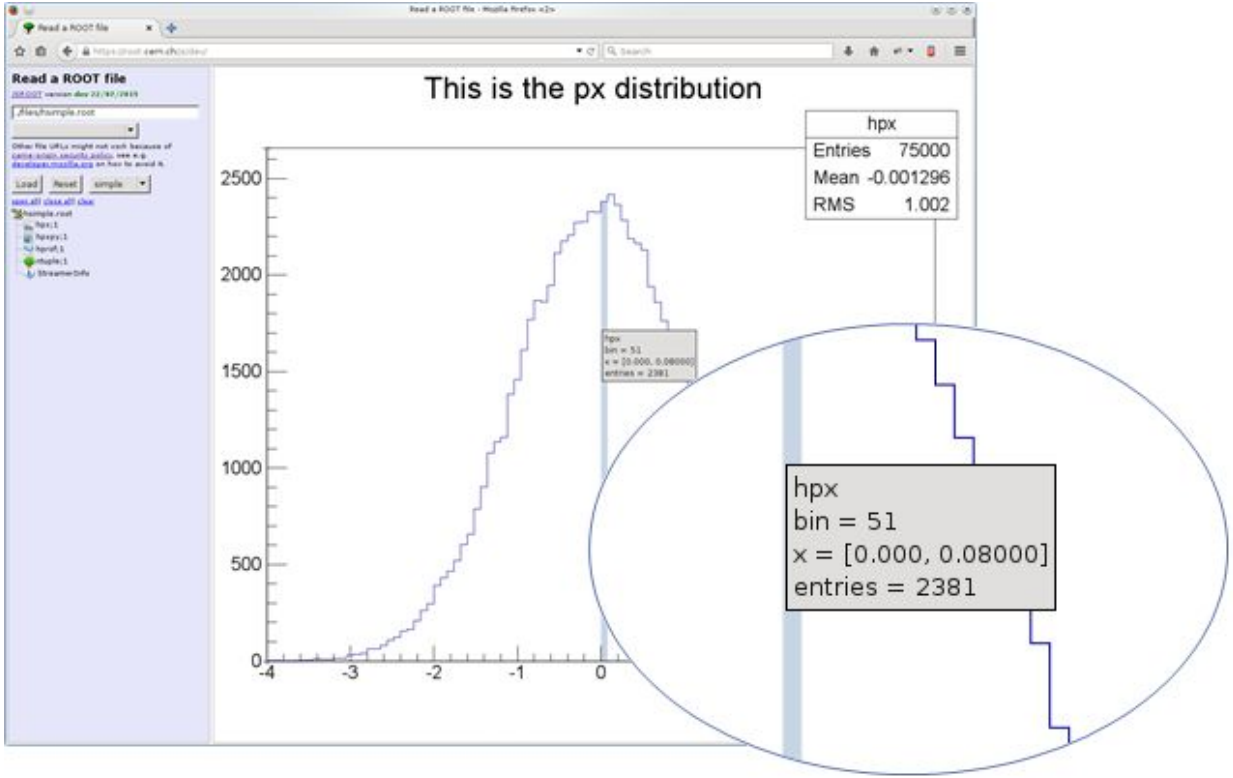
Context Menu

Context menu for the displayed object

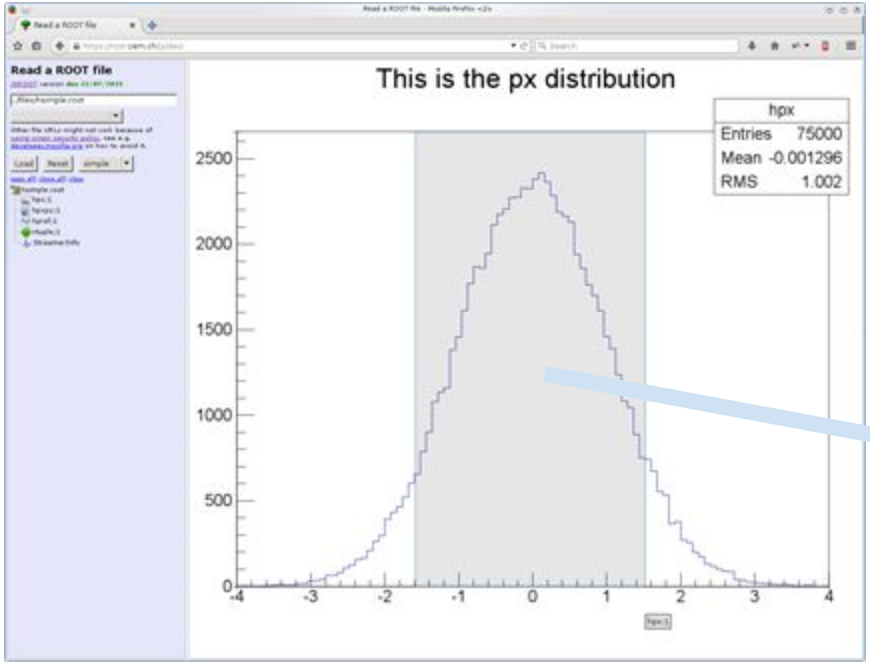


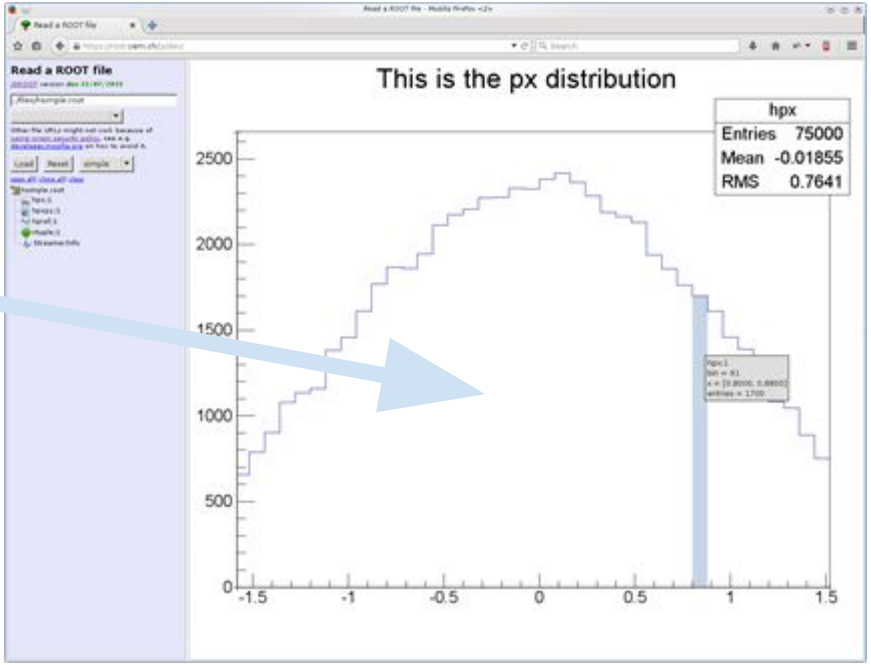
Tooltips

Informative tooltips



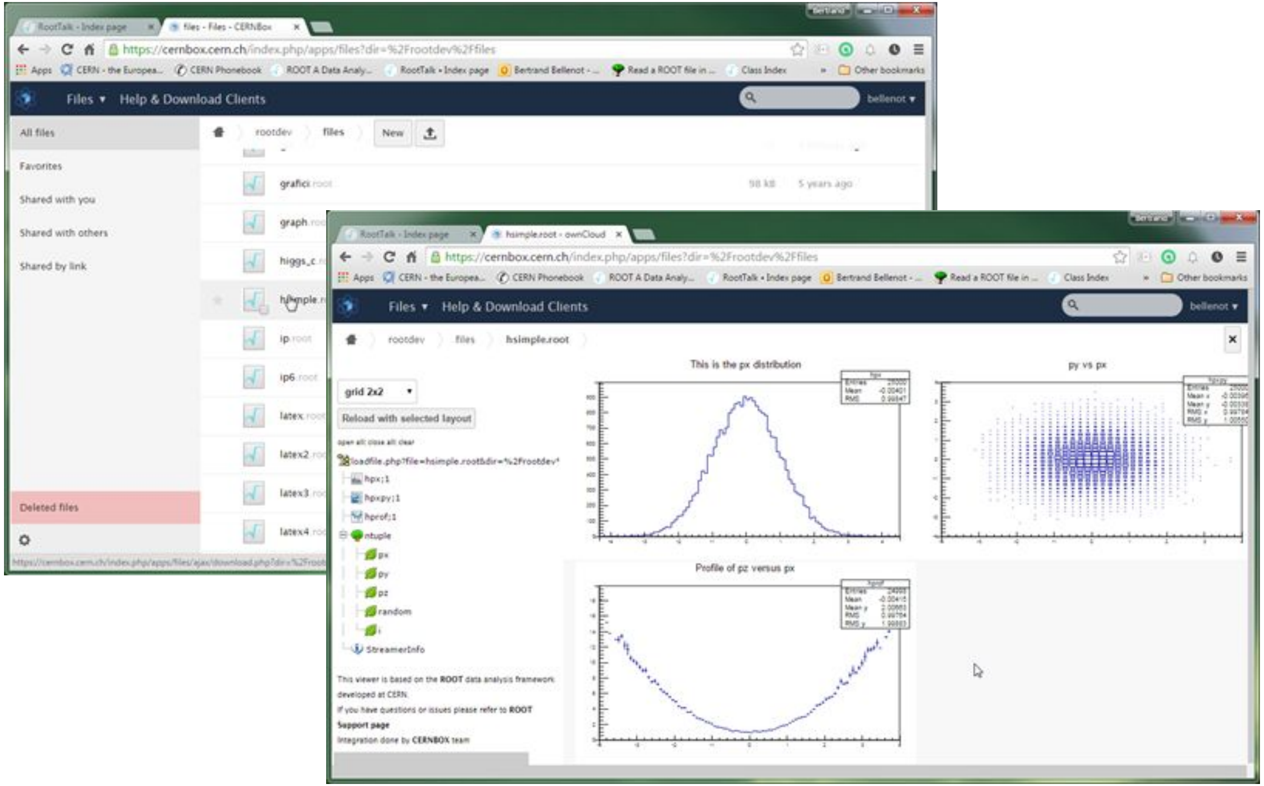
Zooming


Read a ROOT file
Load
Reset
simple
This is the px distribution
hpx
Entries 75000
Mean -0.001296
RMS 1.002
2500
2000
1500
1000
500
0
-4
-3
-2
-1
0
1
2
3
4
hpx;1


Read a ROOT file
Load
Reset
simple
This is the px distribution
hpx
Entries 75000
Mean -0.01855
RMS 0.7641
2500
2000
1500
1000
500
0
-1.5
-1
-0.5
0
0.5
1
1.5
hpx;1
bin = 81
x = [0.8000, 0.8800]
entries = 1700

CERNBox Integration

CERNBox provides a functionality analogous to Dropbox^{TM}, and is managed by CERN IT department (http://cernbox.cern.ch)
It now integrates JSROOT, allowing to display ROOT files content



This viewer is based on the **ROOT** data analysis framework developed at CERN.
If you have questions or issues please refer to **ROOT Support page**
Integration done by **CERNBOX** team

Sharing

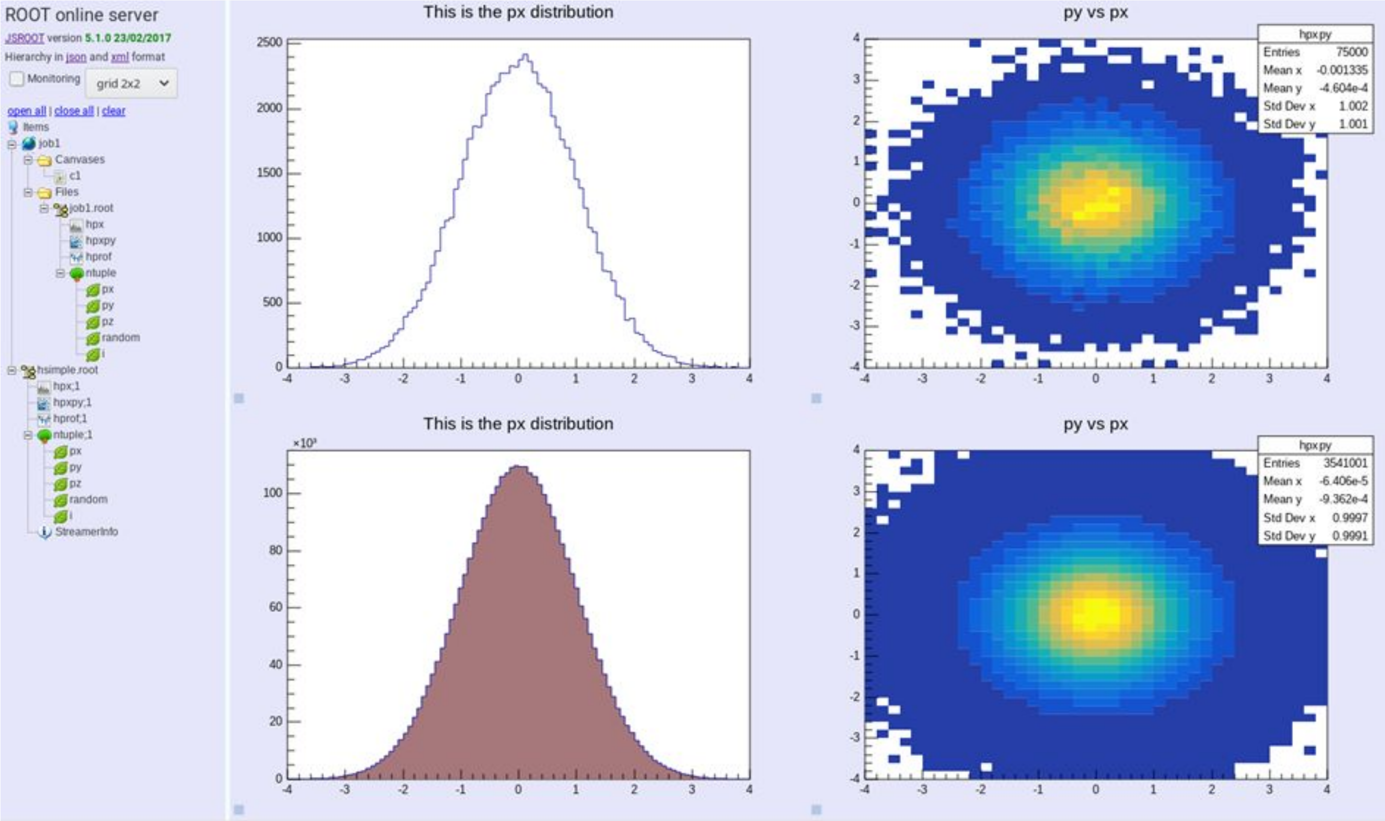
- Without JSROOT
  - create and send PNG image (static), or
  - create and send ROOT file with canvas (interactive)
    - ROOT must be installed everywhere

- With JSROOT
  - copy your ROOT file on the web and send the URL of the file
  - open the JSROOT main page https://root.cern.ch/js/latest/
  - enter the URL of the file (like https://root.cern.ch/js/files/hsimple.root)
  - explore and draw the content of the file

THttpServer

- online access to running ROOT application
- visualization and live update of ROOT objects
- execution of commands and methods
- hierarchy display of objects
- possibility for custom UI

THttpServer



ROOT online server

JSROOT version 5.1.0 23/02/2017

Hierarchy in json and xml format

☐ Monitoring grid 2x2

hpxpy (upper panel)

Entries	75000
Mean x	-0.001335
Mean y	-4.604e-4
Std Dev x	1.002
Std Dev y	1.001

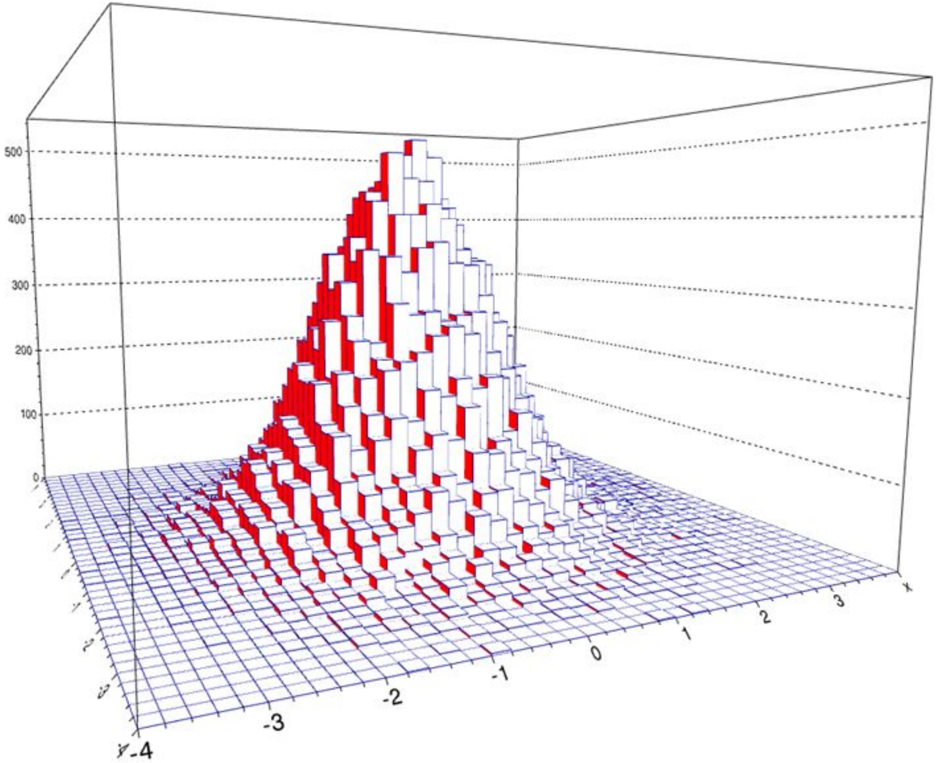
hpxpy (lower panel)

Entries	3541001
Mean x	-6.406e-5
Mean y	-9.362e-4
Std Dev x	0.9997
Std Dev y	0.9991

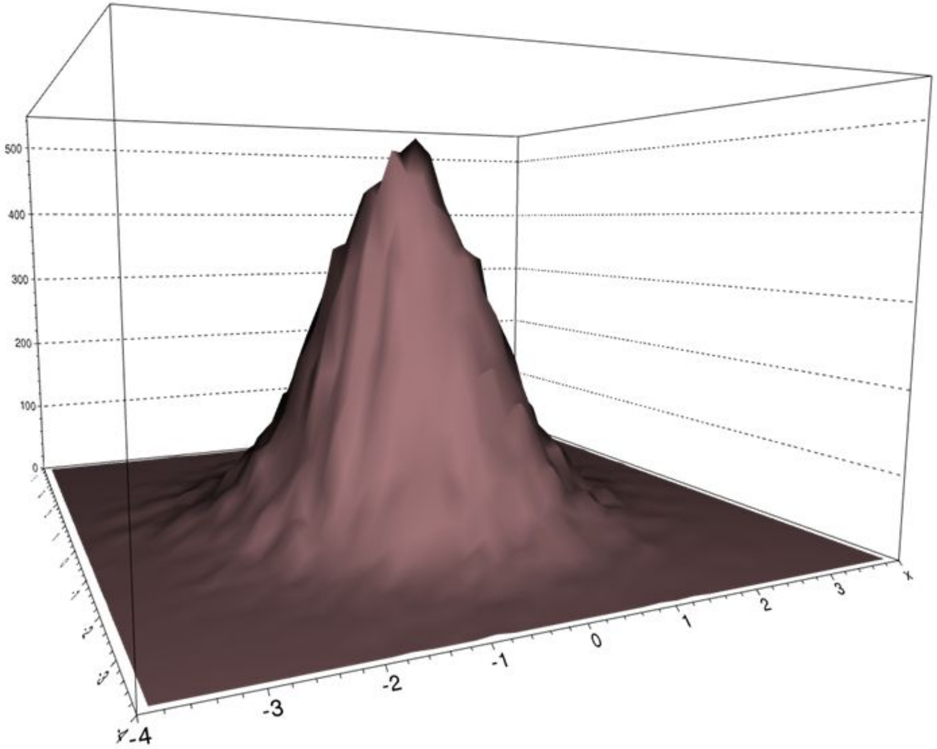
3D histogram display

- Use the three.js package for rendering
- Supports TH1/TH2/TH3
- Interactive zooming
- Tooltips and bins highlight
- Good performance with 200x200 TH2

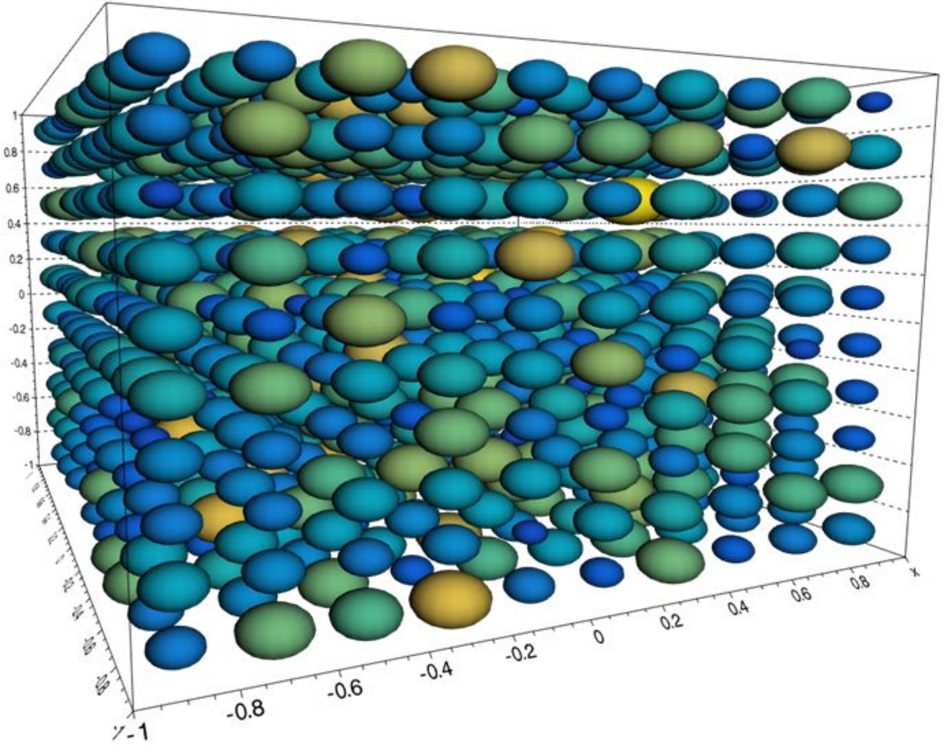
TH2 LEGO1



TH2 SURF4



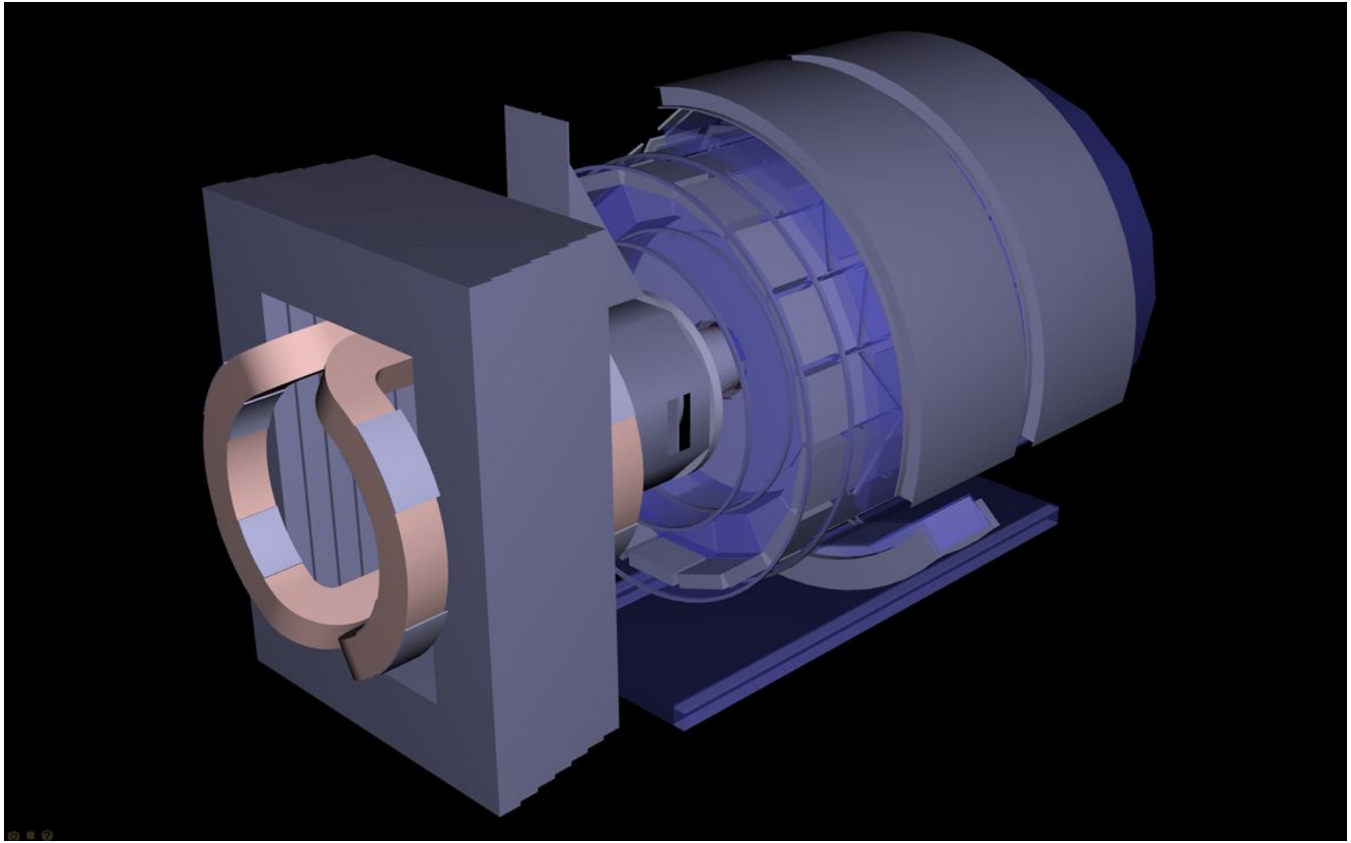
TH3 GLBOX2



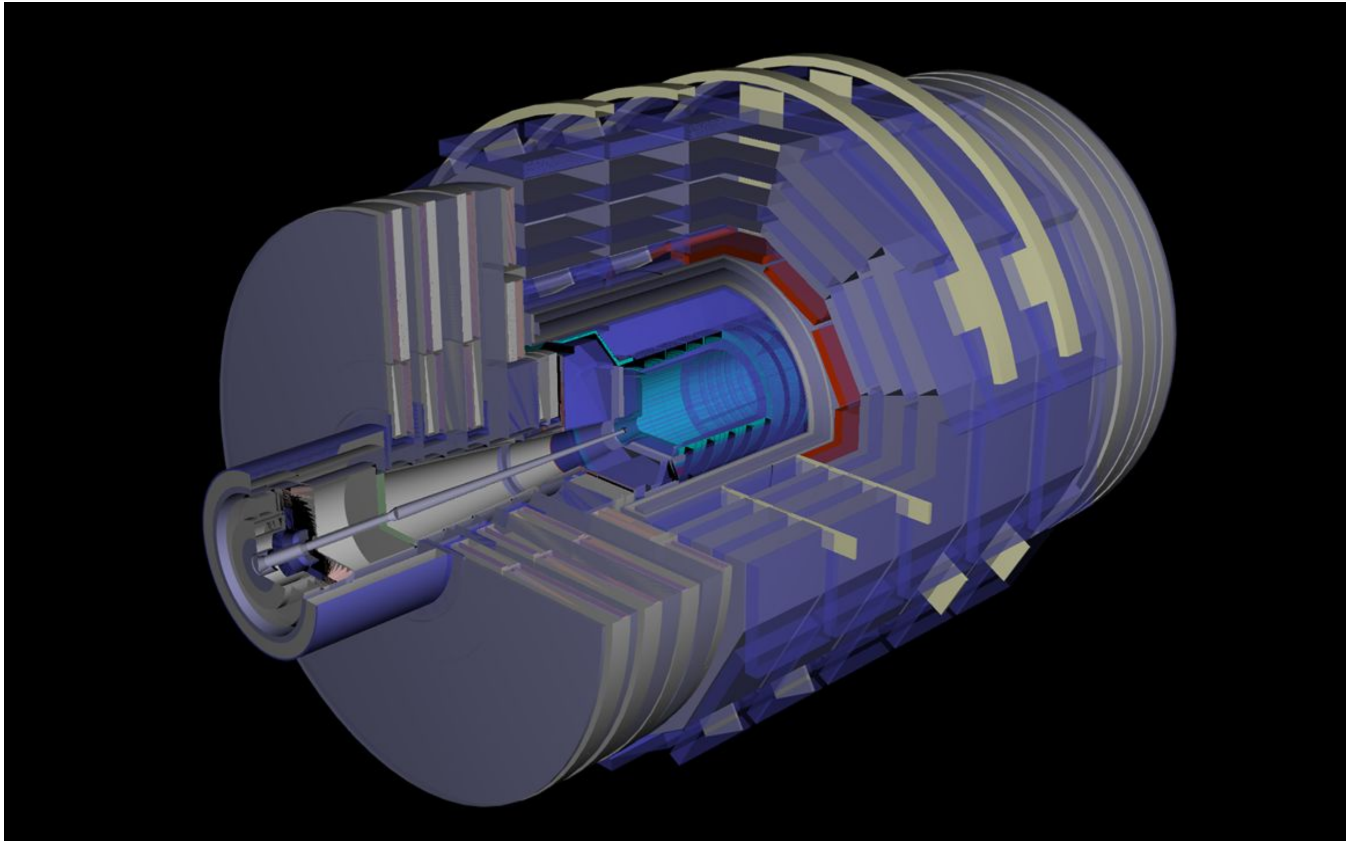
Geometries

- All kind of ROOT TGeo classes
- All kinds of shapes
  - composite with enhanced ThreeCSG.js
- Interactive:
  - rotation and zoom
  - volumes highlight and tooltip
  - context menu
  - clip panels
  - transparency

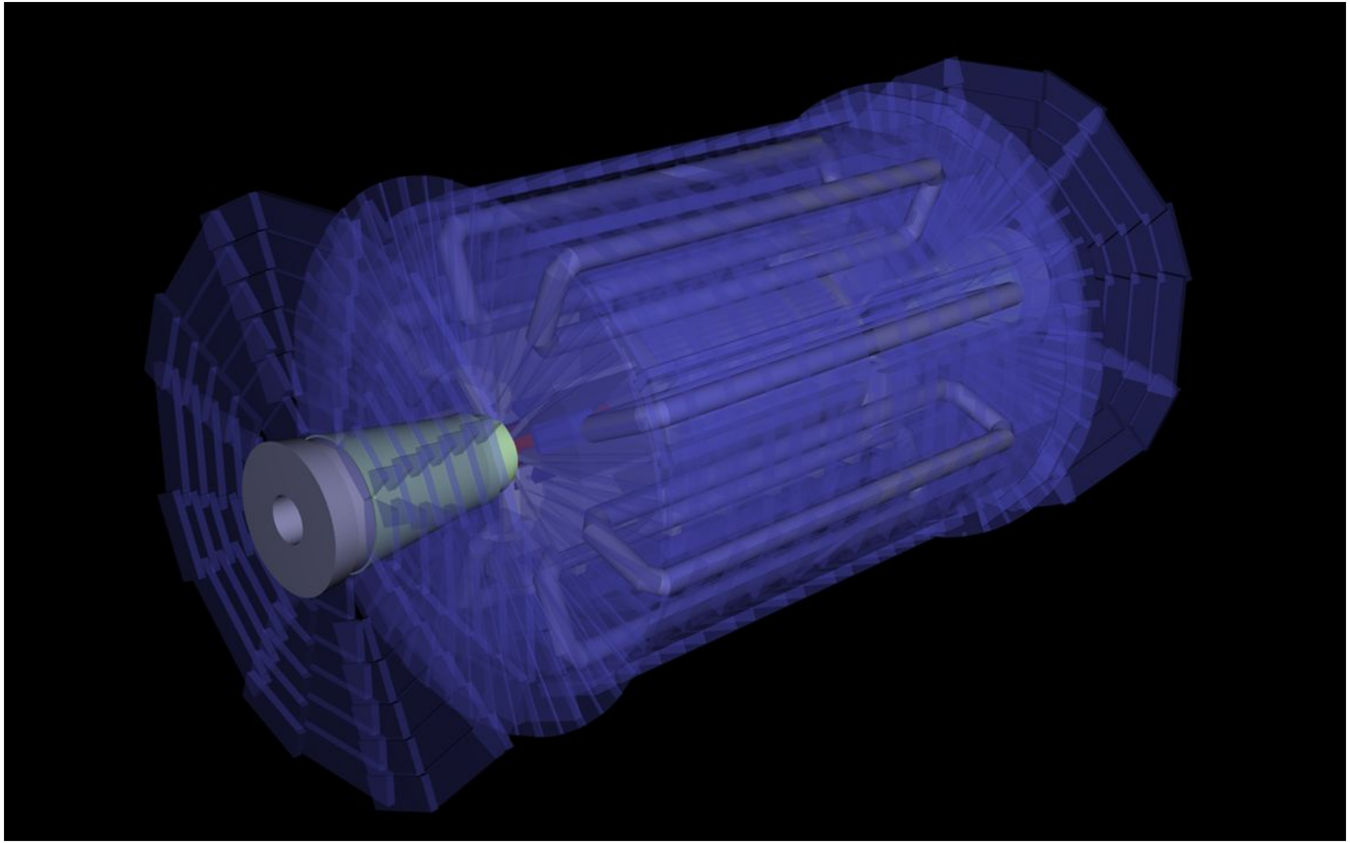
Alice



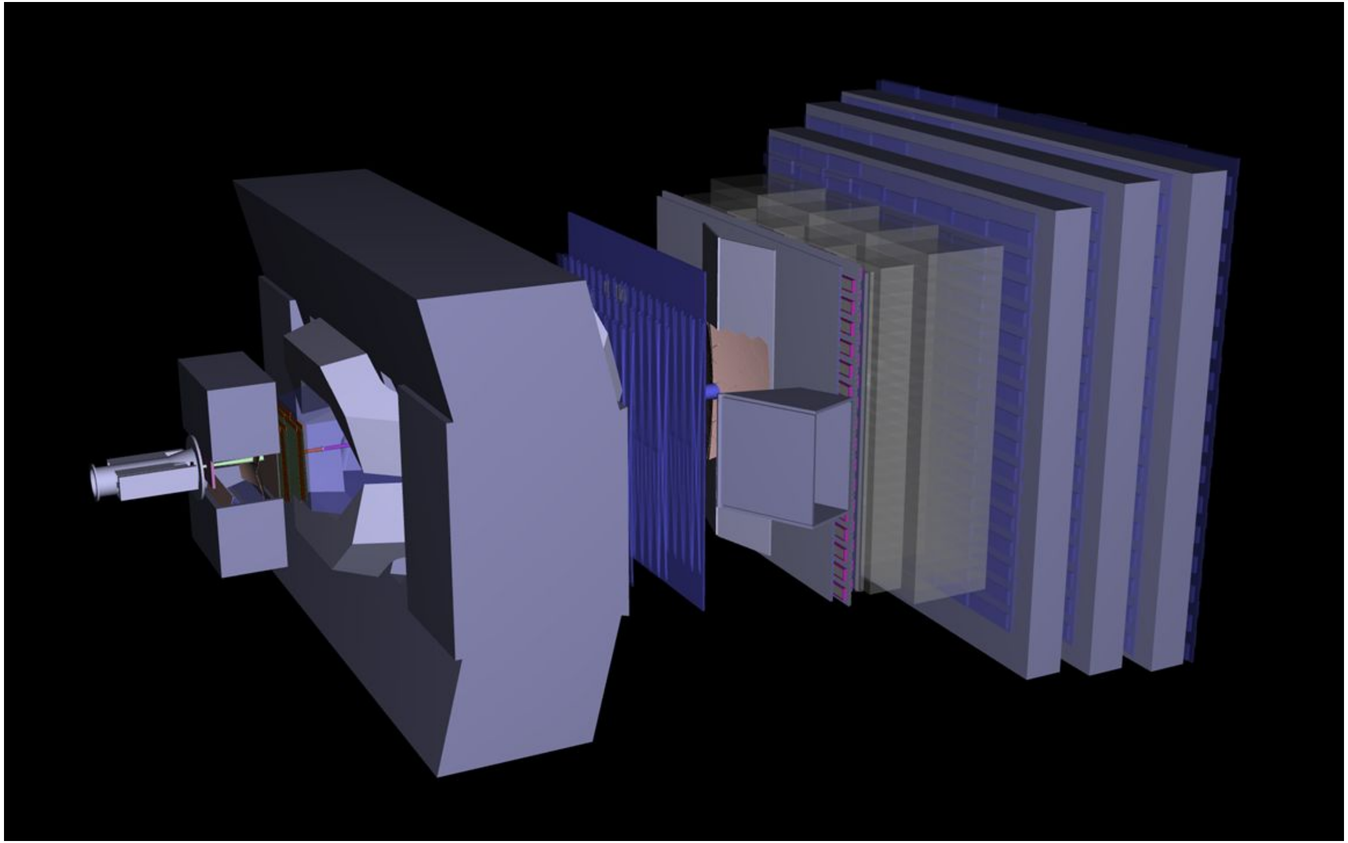
CMS



Atlas



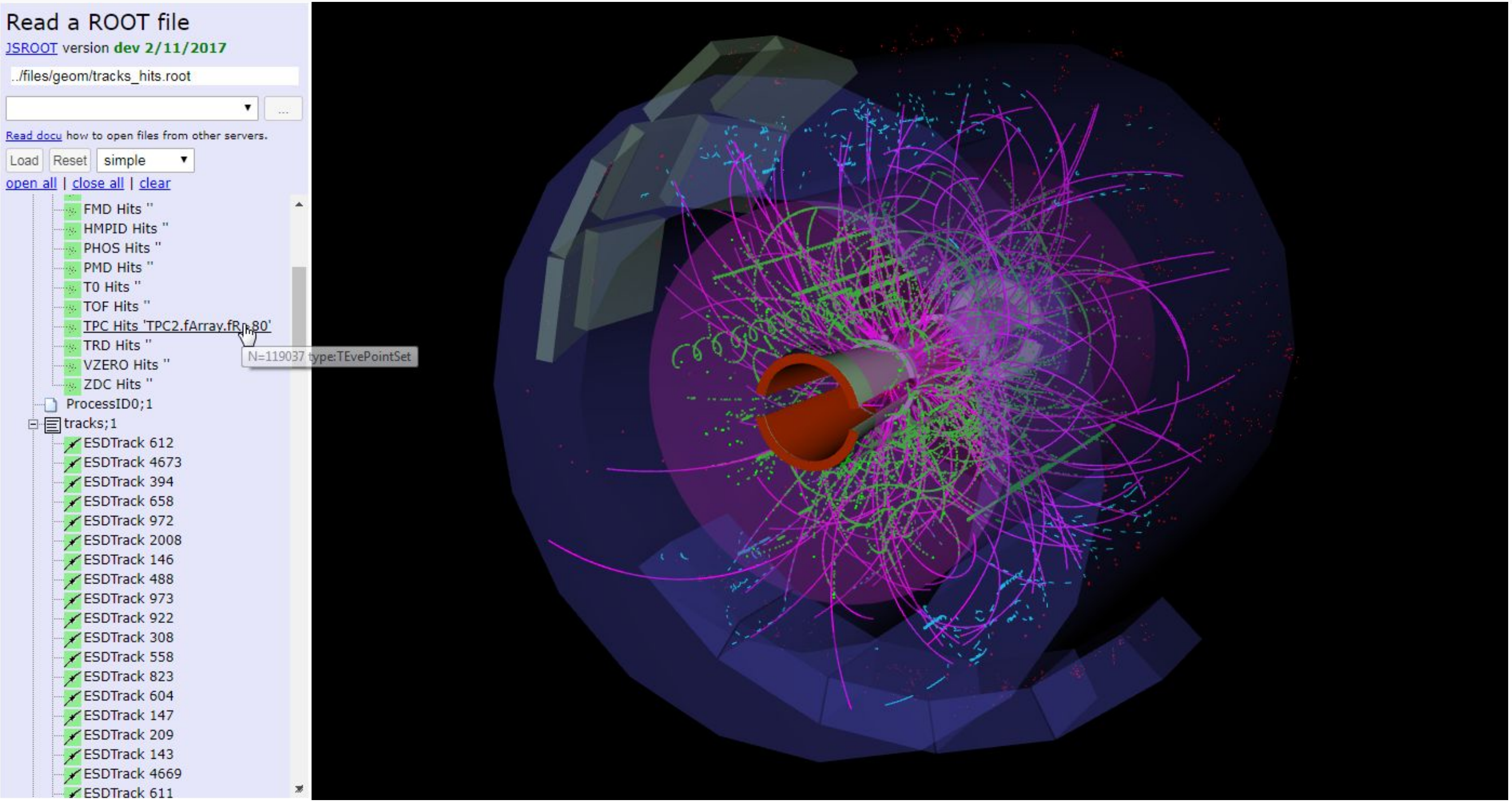
LHCb



Tracks and hits

- Overlay with geometry drawing
  - TGeoTrack, TEveTrack, TEvePointSet
  - extract tracks/hits from the TTree
  - the list can easily be extended
- Hierarchy browser
  - highlight on the 3D scene when selected
  - toggle the visibility flags

Tracks & Hits



Read a ROOT file

JSROOT version dev 2/11/2017

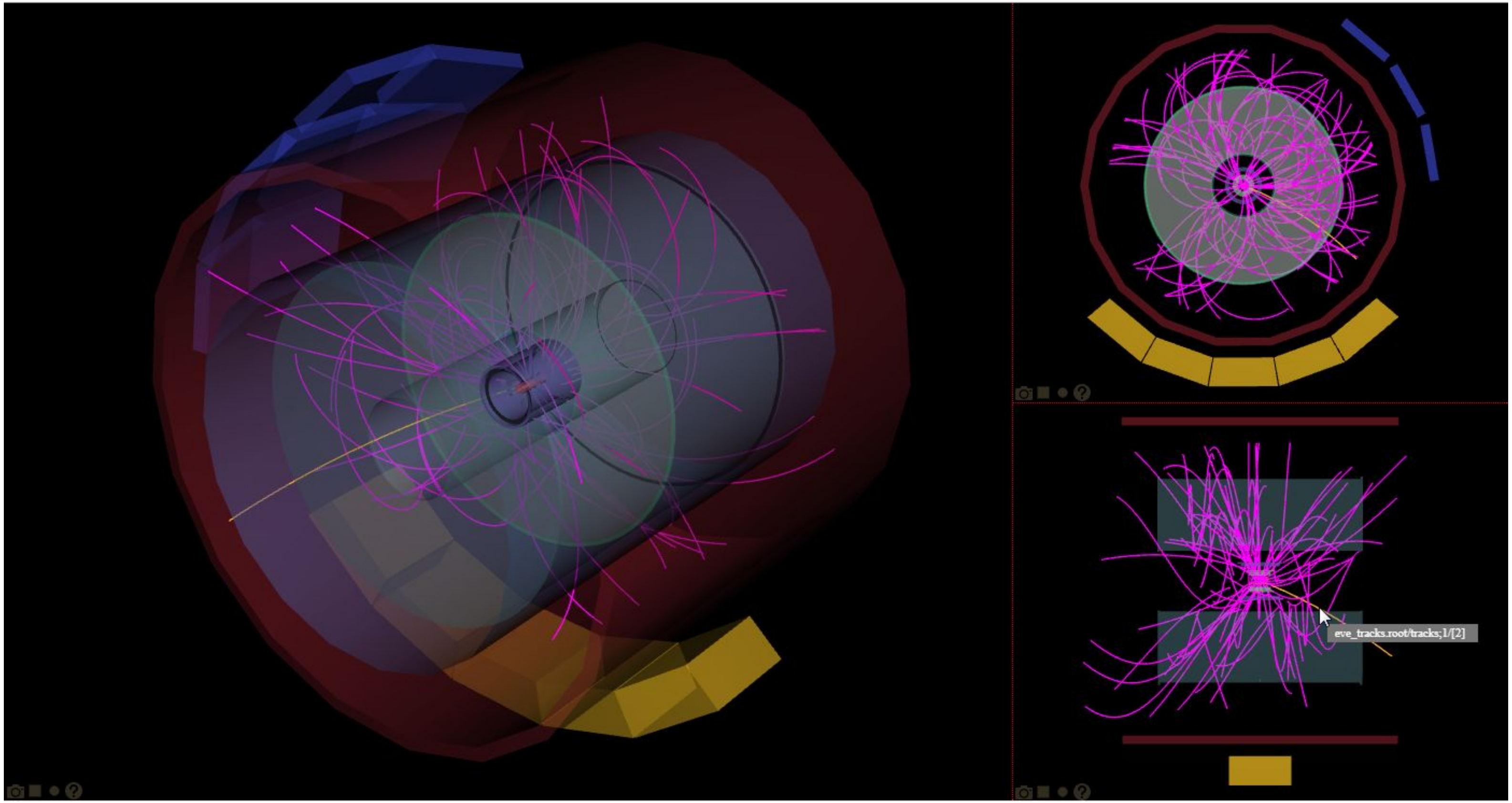
../files/geom/tracks_hits.root

Read docu how to open files from other servers.

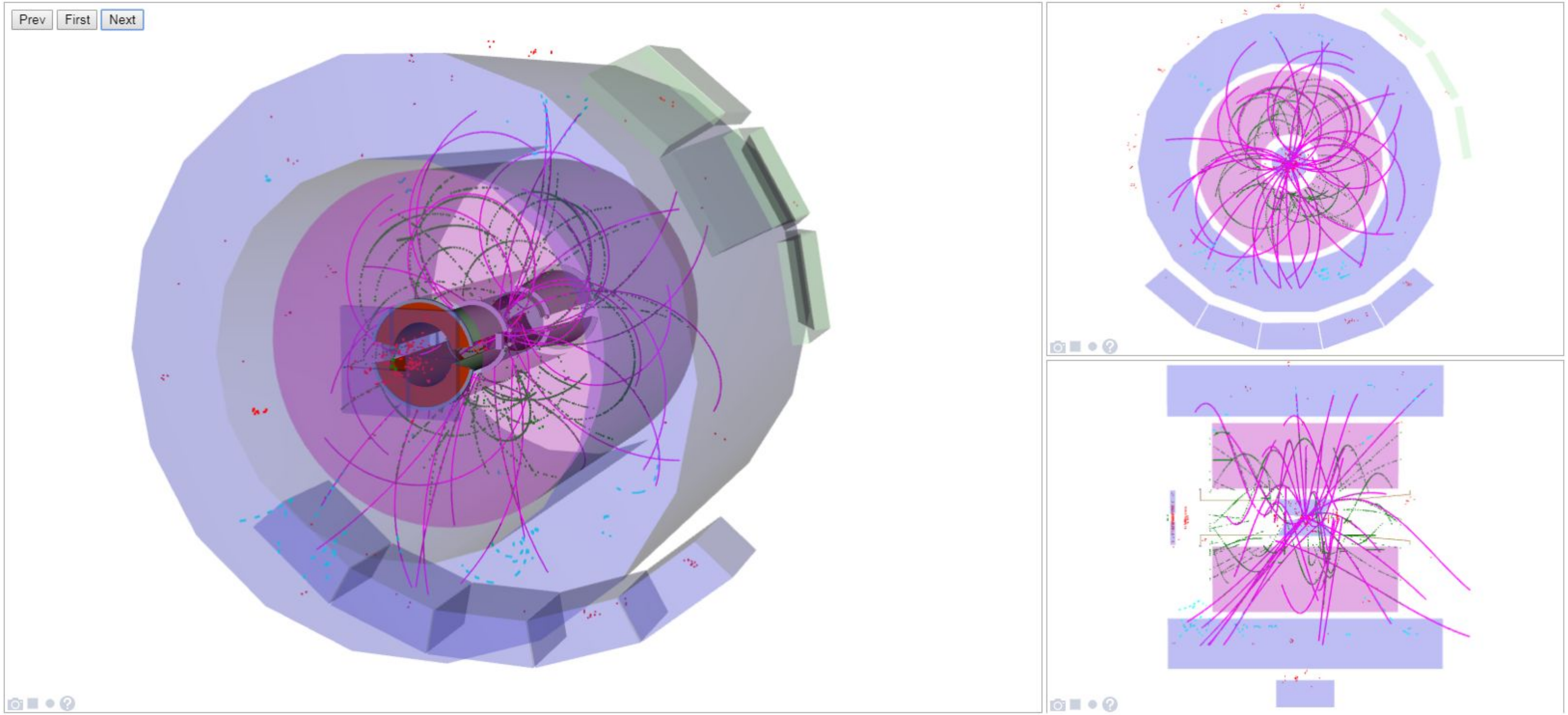
- FMD Hits ''
- HMPID Hits ''
- PHOS Hits ''
- PMD Hits ''
- T0 Hits ''
- TOF Hits ''
- TPC Hits 'TPC2.fArray.fR<80'
- TRD Hits ''
- VZERO Hits ''
- ZDC Hits ''
- ProcessID0;1
- tracks;1
  - ESDTrack 612
  - ESDTrack 4673
  - ESDTrack 394
  - ESDTrack 658
  - ESDTrack 972
  - ESDTrack 2008
  - ESDTrack 146
  - ESDTrack 488
  - ESDTrack 973
  - ESDTrack 922
  - ESDTrack 308
  - ESDTrack 558
  - ESDTrack 823
  - ESDTrack 604
  - ESDTrack 147
  - ESDTrack 209
  - ESDTrack 143
  - ESDTrack 4669
  - ESDTrack 611

N=119037 type:TEvePointSet

Projections



Pseudo Event Display



Your Turn!

Visit:

https://bellenot.web.cern.ch/bellenot/Public/GitHub/?layout=h12_21&json=../latest/files/simple_alice.json&file=../latest/files/pp_900_01.root&items=[[[0],[1]/hits,[1]/tracks],[0],[0]]&opts=[main;black,projz,projx]

Load and visualize the content of the files in the list!

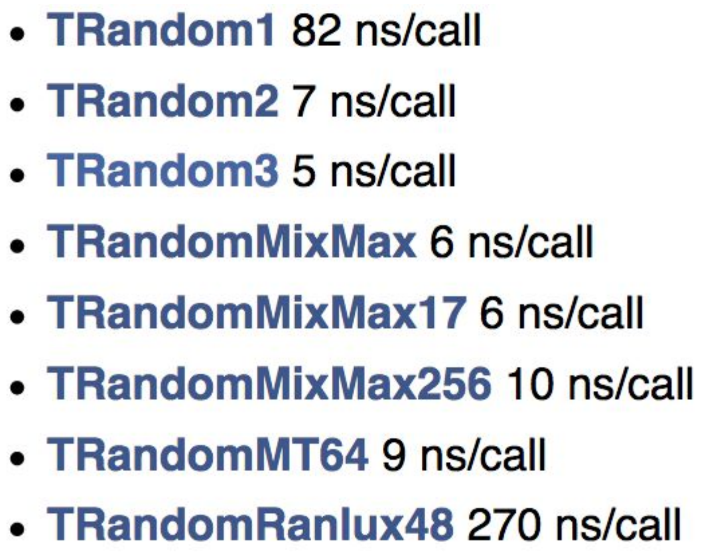
Useful Links

- https://root.cern.ch/js/
- https://github.com/root-project/jsroot
- https://github.com/root-project/jsroot/blob/master/docs/JSROOT.md

Interlude: Random Number Generation

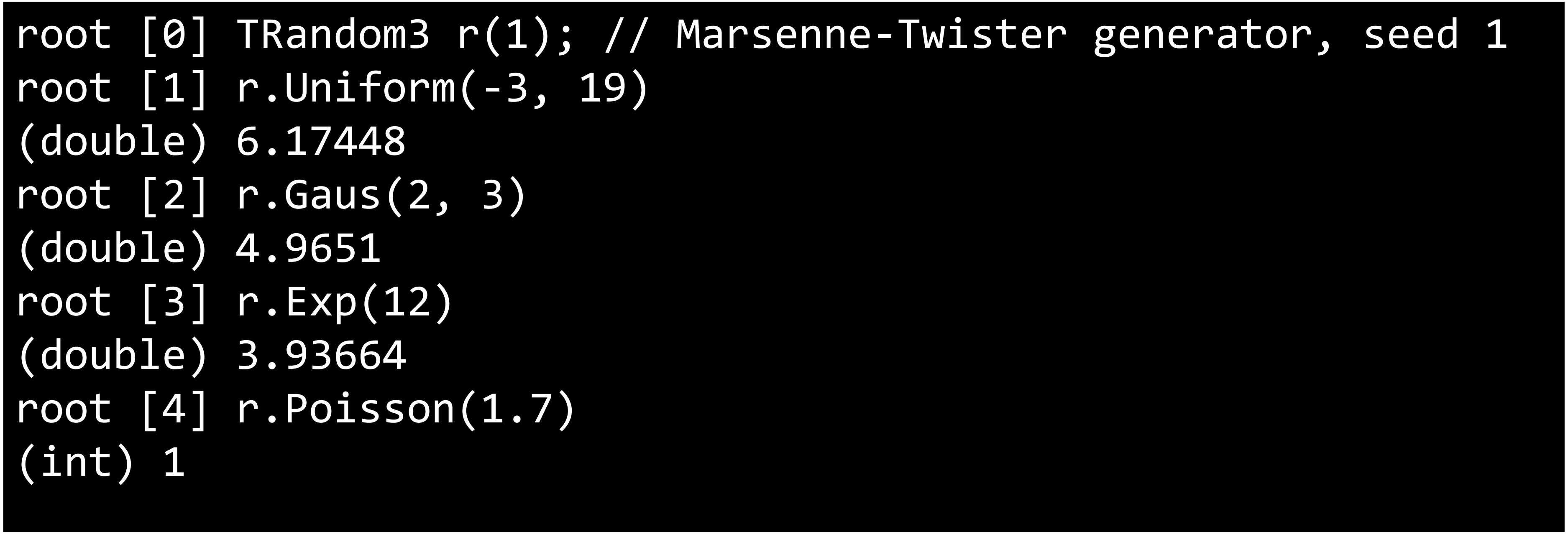
Pseudo Random Number Generation

- Crucial for HEP and science
  - E.g. Simulation, statistics
- Used in our examples
- Achieved with the TRandomX class
  - TRandom1, TRandom2, TRandom3
- Collection of algorithms available



- **TRandom1** 82 ns/call
- **TRandom2** 7 ns/call
- **TRandom3** 5 ns/call
- **TRandomMixMax** 6 ns/call
- **TRandomMixMax17** 6 ns/call
- **TRandomMixMax256** 10 ns/call
- **TRandomMT64** 9 ns/call
- **TRandomRanlux48** 270 ns/call

Generate Numbers in a Nutshell



```
root [0] TRandom3 r(1); // Marsenne-Twister generator, seed 1
root [1] r.Uniform(-3, 19)
(double) 6.17448
root [2] r.Gaus(2, 3)
(double) 4.9651
root [3] r.Exp(12)
(double) 3.93664
root [4] r.Poisson(1.7)
(int) 1
```

- Use seed to control random sequence: same seed, same sequence.
- Fundamental for reproducibility

Parameter Estimation and Fitting

Learning Objectives

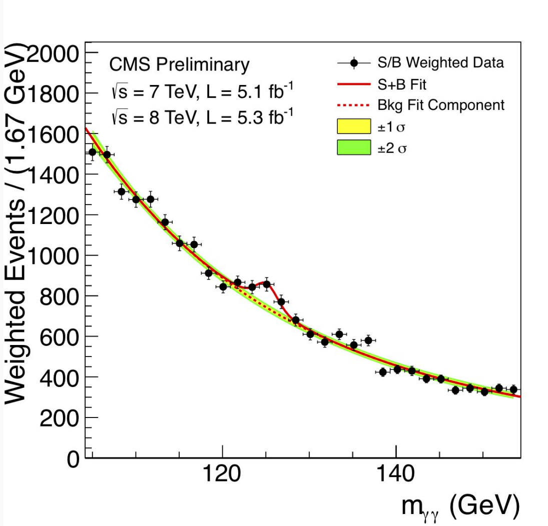
- Understand the problem of parameter estimation
- Know the difference between a likelihood or a Chi2 based approach
- Become familiar with the tools ROOT offers to perform a fit: fit panel and programmatic steering of the procedure

Outline

- Introduction to Fitting:
  - fitting methods in ROOT
  - how to fit a histogram in ROOT,
  - how to retrieve the fit result.
- Building fit functions in ROOT.
- Interface to Minimization.
- Common Fitting problems.
- Using the ROOT Fit GUI (Fit Panel).

What is Fitting ?

- Estimate parameters of an 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)



Example

Higgs ➞ γγ spectrum

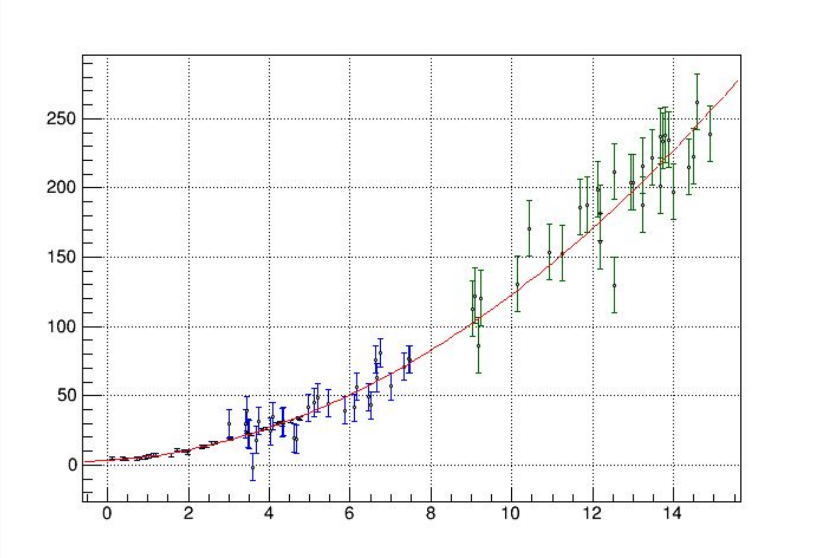
We can fit for:

- the expected number of Higgs events
- the Higgs mass

Least Square (χ^2) Fit

- Minimizes the deviations between the observed y and the predicted function values:
- Least square fit (χ^2) : minimize square deviation weighted by the uncertainties

$$\chi^2 = \sum_i \frac{(Y_i - f(X_i, \theta))^2}{\sigma_i^2}$$

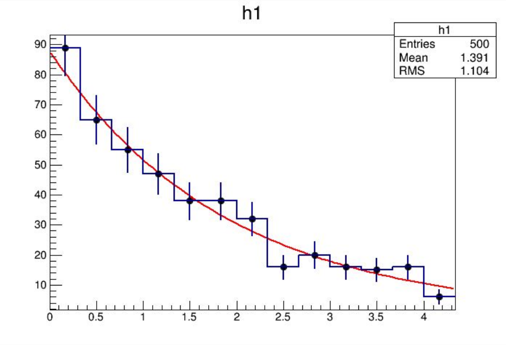


Maximum Likelihood Fit

- The parameters are estimated by finding the maximum of the likelihood function (or minimum of the negative log-likelihood function).
    - Likelihood: $$L(x|\theta) = \prod_i P(x_i|\theta)$$
    - Find best value θ, the maximum of: $$logL = \sum_i \log(f(x_i, \theta))$$

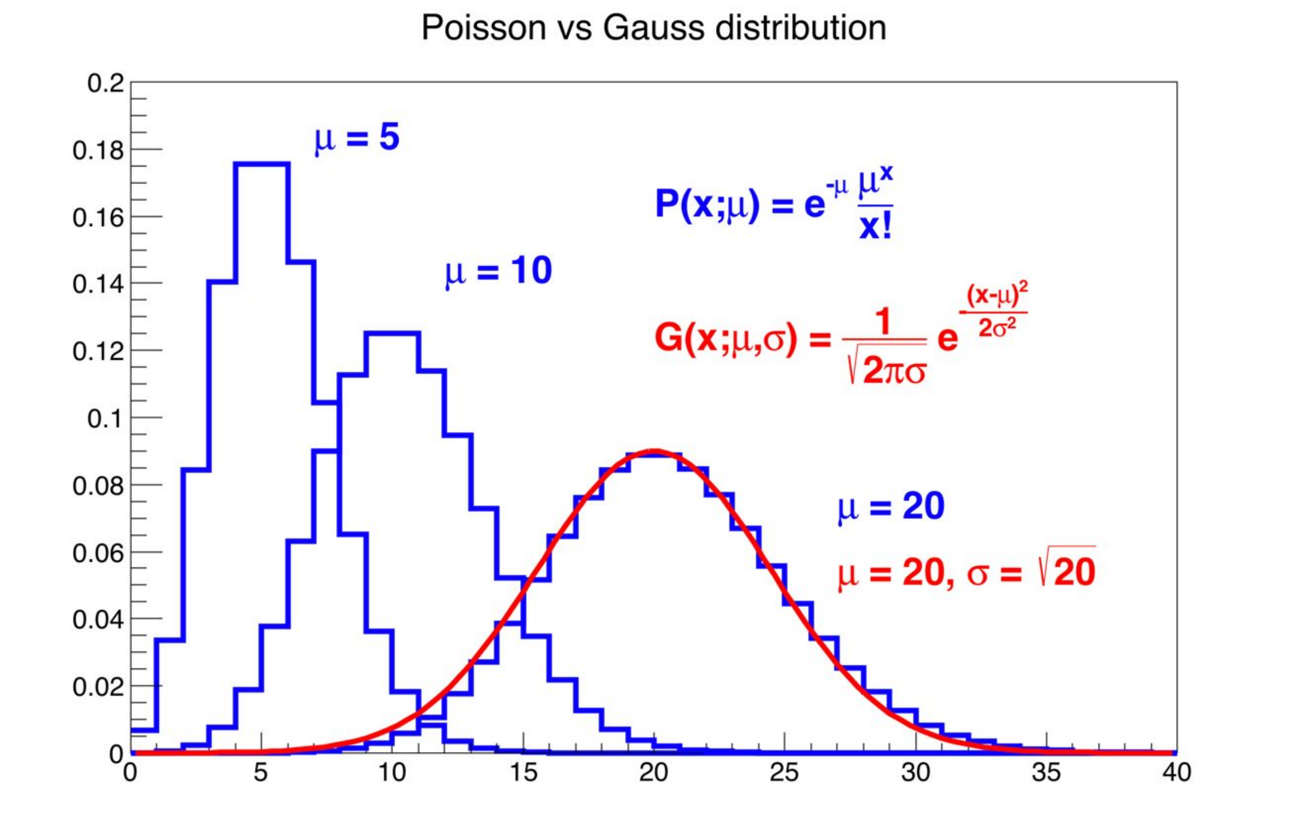
- The least-square fit and the maximum likelihood fit are equivalent when the error model is Gaussian
    - E.g. the observed events in each bin is normal `f(x| θ)` is gaussian

ML Fit of an Histogram



- The Likelihood for a histogram is obtained by assuming a Poisson distribution in every bin:
  - **Poisson(n_i|ν_i)**
    - **n_i** is the observed bin content.
    - **ν_i** is the expected bin content, **$\nu_i = f(x_i|\theta)$** , where x_i is the bin center, assuming a linear function within the bin. Otherwise it is obtained from the integral of the function in the bin.
- For large histogram statistics (large bin contents) bin distribution can be considered normal
  - equivalent to least square fit
- For low histogram statistics the ML method is the correct one !

Comparing Poisson and Gauss



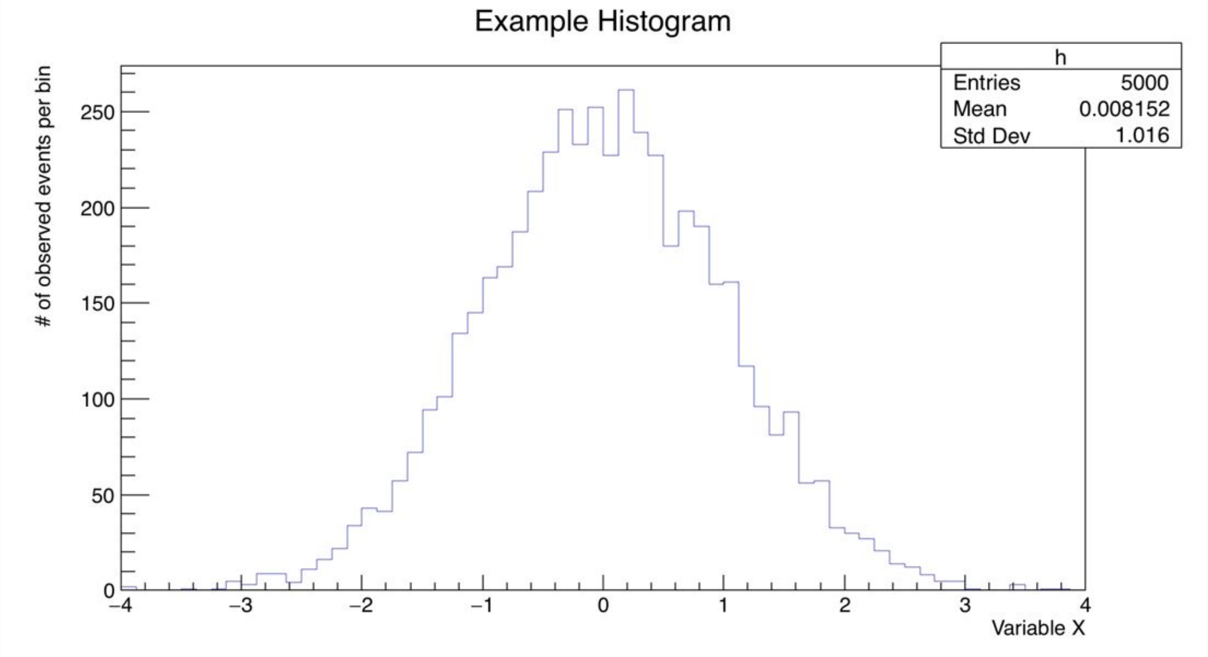
Fitting in ROOT

How do we do fit in ROOT:

- Create first a parametric function object, **`TF1`**, which represents our model, *i.e.* the fit function.
  - 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 possibles (see the **`TH1::Fit`** documentation)
    - e.g select type of fit : least-square (default) or likelihood (option “L”)
- Examine result:
  - get parameter values;
  - get parameter errors (e.g. their confidence level);
  - get parameter correlation;
  - get fit quality.
- The resulting fit function is also draw automatically on top of the Histogram or the Graph when calling **`TH1::Fit`** or **`TGraph::Fit`**

Simple Gaussian Fitting

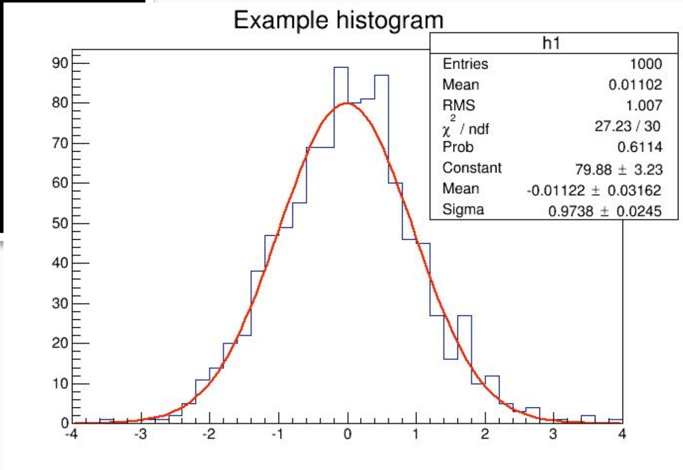
- Suppose we have this histogram
  - we want to estimate the mean and sigma of the underlying gaussian distribution.



Fitting Histograms

- How to fit a histogram:

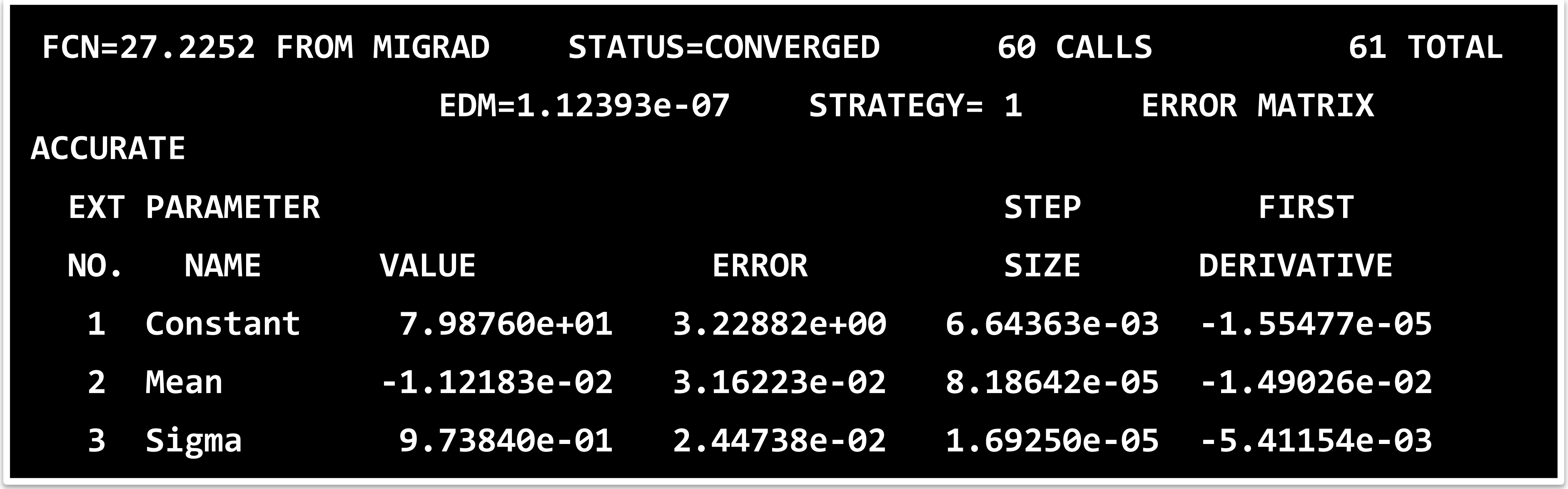
```
root [0] TF1 f1("f1","gaus");
root [1] h1.Fit(&f1);

 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);
```

How to Inspect the Output



```
 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
```

Creating the Fit Function

- How to create the parametric function object (**TF1**) :
  - we can write formula expressions using functions available from the C++ standard library and those available in ROOT (e.g. TMath)

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

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

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

  - with pre-defined functions the parameter names and value are automatically set to meaningful values (whenever possible)
  - pre-defined functions available: *gaus, expo, landau, breitwigner,crystal_ball,pol0,1..,10, cheb0,1,xygaus,xylanday,bigaus*

Building More Complex Functions

- Sometimes better to write directly the functions in C/C++
  - but in this case the function object cannot be fully stored to disk and reused later

- Example: using a general free function with parameters:

```
double function(double *x, double *p){
   return p[0]*TMath::Gaus(x[0],p[0],p[1]);
}
TF1 f1("f1",function,xmin,xmax,npar);
```

Building More Complex Functions

- Any C++ object (functor) implementing

```
double operator() (double *x, double *p)
```

```
struct Function {
      double operator() (double *x, double *p){
        return p[0]*TMath::Gaus(x[0],p[1],p[2]);
      }
};

Function f;
TF1 f1("f1",f,xmin,xmax,npar);
```

- also a lambda function (with Cling and C++-11)

```
TF1 f1("f1",[](double *x, double *p){return p[0]*x[0];},0,10,1);
```

a lambda can be used also as a string expression, which will be JIT'ed by CLING

```
TF1 f1("f1","[](double *x, double *p){return p[0]*x[0];}",0,10,1);
```

Building more complex Functions (2)

- Recently TFormula has been extended to build easily more complicated functions.
  - Support for composite functions, e.g. f2 (f1 (x))

```
TF1 f1("f","Gaus(x,[A],[m0]+[m1]*x,[s])")
```

  - Support for normalized sums

```
TF1 f1("f","NSUM(expo,gaus)",110,160)
```

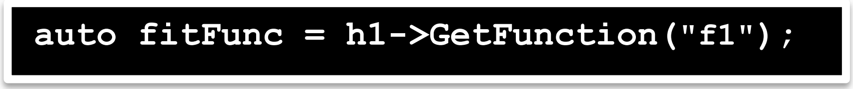
  - Support for convolution

```
TF1 f1("f","CONV(breitwigner,gausn)",-10,10)
```

Retrieving The Fit Result

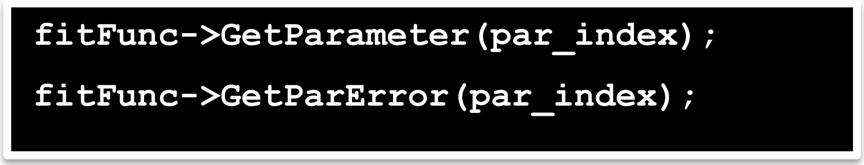
- The main results from the fit are stored in the fit function, which is attached to the histogram; it can be saved in a file (except for C/C++ functions were only points are saved).

- The fit function can be retrieved using its name:



```
auto fitFunc = h1->GetFunction("f1");
```

- The parameter values/error using indices (or their names):



```
fitFunc->GetParameter(par_index);

fitFunc->GetParError(par_index);
```

Retrieving The Fit Result

- It is also possible to access the **TFitResult** class which has all information about the fit, if we use the fit option “S”:

```
TFitResultPtr r = h1->Fit(f1,"S");
TMatrixDSym C = r->GetCorrelationMatrix();
r->Print();
```

C++ Note: the TFitResult class is accessed by using `operator->` of `TFitResultPtr`

Fitting Options

- Likelihood fit for histograms
  - option “L” for count histograms;

```
h1->Fit("gaus","L");
```

  - option “WL” in case of weighted counts.

```
h1->Fit("gaus","LW");
```

- Default is chi-square with observed errors (and skipping empty bins)
  - option “P” for Pearson chi-square

    expected errors, and including empty bins

```
h1->Fit("gaus","P");
```

- Use integral function of the function in bin

```
h1->Fit("gaus","L I");
```

- Compute MINOS errors : option “E”

```
h1->Fit("gaus","L E");
```

Some More Fitting Options

- Fitting in a Range

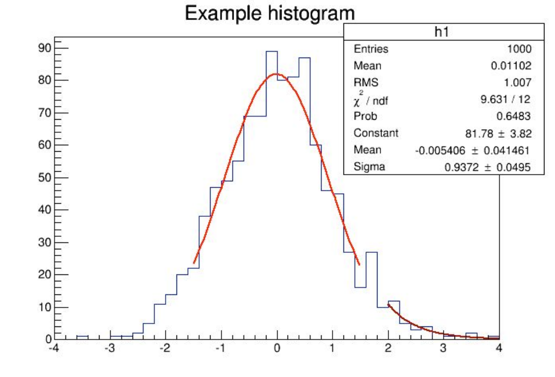
  -
    ```
    h1->Fit("gaus","","",-1.5,1.5);
    ```
- For doing several fits

  -
    ```
    h1->Fit("expo","+","",2.,4);
    ```
- Quiet / Verbose: option “Q”/”V”

  -
    ```
    h1->Fit("gaus","V");
    ```
- Avoid storing and drawing fit function (useful when fitting many times)

  -
    ```
    h1->Fit("gaus","L N 0");
    ```
- Save result of the fit, option “S”

  -
    ```
    auto result = h1->Fit("gaus","L S");
    result->Print("V");
    ```



All fitting options documented in reference guide or User Guide (Fitting Histogram chapter)

Note on Binned Likelihood Fit

- Log-Likelihood is computed using Baker-Cousins procedure (Likelihood χ^2)

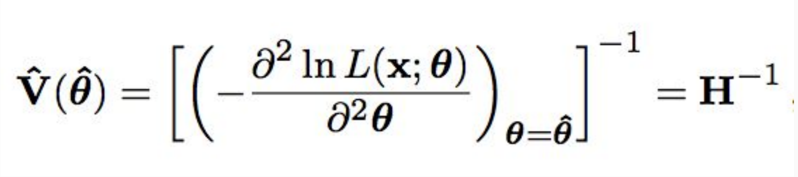
▷

$$\chi^2_\lambda(\theta) = -2\ln\lambda(\theta) = 2\sum_i [\mu_i(\theta) - n_i + n_i \ln(n_i/\mu_i(\theta))]$$

  - -2lnλ(θ) is an equivalent chi-square
  - Its value at the minimum can be used for checking the fit quality
    - mitigate problems with bins with low content
    - toy MC studies show that its distribution is closer to a chi-square distribution than a Neyman or Pearson chi-square
- ROOT computes -lnλ(θ)
  - retrieve it using `TFitResult::MinFcnValue()`

Parameter Errors

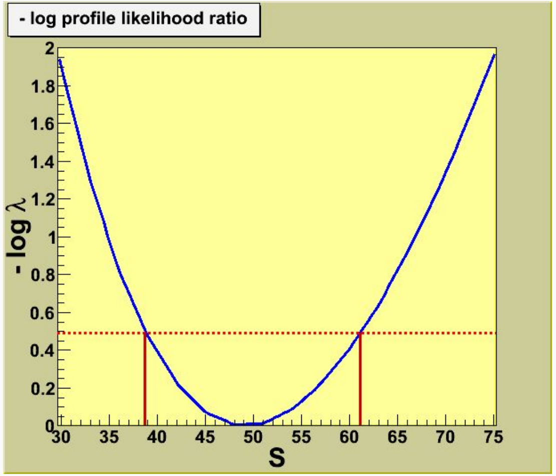
- Errors returned by the fit are computed from the second derivatives of the log-likelihood function
  - Assume the negative log-likelihood function is a parabola around the minimum
  - This is true asymptotically and in this case the parameter estimates are also normally distributed.
  - The estimated correlation matrix is then:

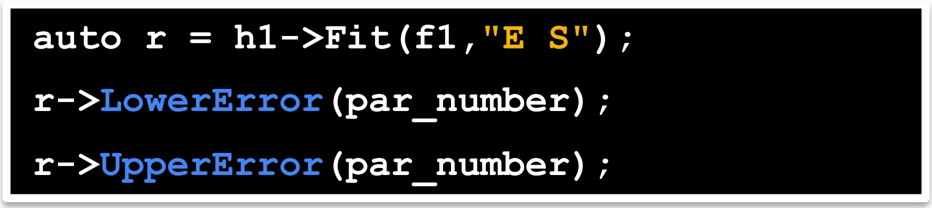


$$\hat{\mathbf{V}}(\hat{\boldsymbol{\theta}}) = \left[\left(- \frac{\partial^2 \ln L(\mathbf{x};\boldsymbol{\theta})}{\partial^2 \boldsymbol{\theta}} \right)_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \right]^{-1} = \mathbf{H}^{-1}$$

Parameter Errors

- A better approximation to estimate the confidence level of the parameter is to use directly the log-likelihood function and look at the difference from the minimum.
  - Method of Minuit/Minos (Fit option “E”)
    - obtain a confidence interval which is in general not symmetric around the best parameter estimate





```
auto r = h1->Fit(f1,"E S");
r->LowerError(par_number);
r->UpperError(par_number);
```

Minimization

- The fit is done by minimizing the least-square or likelihood function.
- A direct solution exists only in case of linear fitting
  - it is done automatically in such cases (e.g fitting polynomials).
- Otherwise an iterative algorithm is used:
  - Minuit is the minimization algorithm used by default
    - ROOT provides two implementations: Minuit and Minuit2
    - other algorithms exists: Fumili, or minimizers based on GSL, genetic and simulated annealing algorithms
  - To change the minimizer:

```
ROOT::Math::MinimizerOptions::SetDefaultMinimizer("Minuit2");
```

  - Other commands are also available to control the minimization:

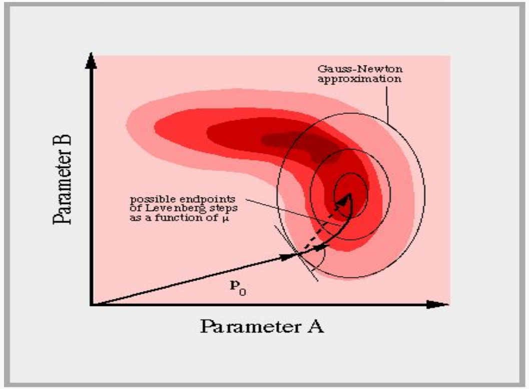
```
ROOT::Math::MinimizerOptions::SetDefaultTolerance(1.E-6);
```

Minimization Techniques

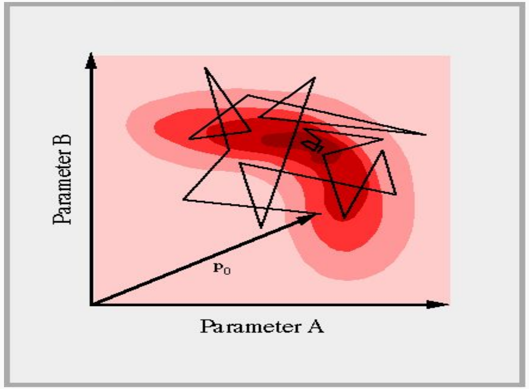
- Methods like Minuit based on gradient can get stuck easily in local minima.
- Stochastic methods like simulated annealing or genetic algorithms can help to find the global minimum but they can be expensive in function calls (and CPU time)

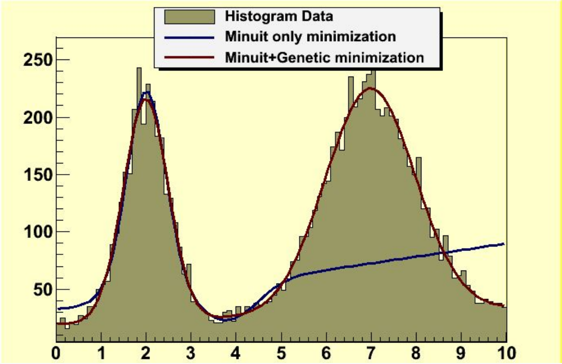
Example: Fitting 2 peaks in a spectrum

Quadratic Newton


Gauss-Newton approximation
possible endpoints of Levenberg steps as a function of μ
P0
Parameter B
Parameter A

Simulated Annealing





Function Minimization

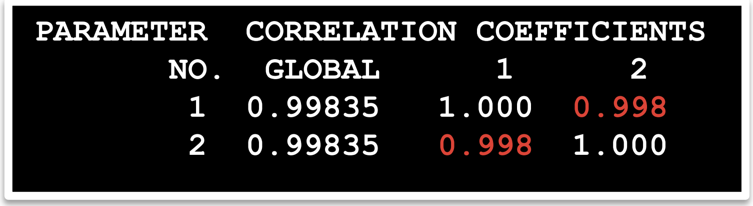
- Common interface class (**ROOT::Math::Minimizer**)
- Existing implementations available as plug-ins:
  - **Minuit** (based on class TMinuit, direct translation from Fortran code)
    - with Migrad, Simplex, Minimize algorithms
  - **Minuit2** (C++ implementation with OO design)
    - with Migrad, Simplex, Minimize and Fumili2
  - **Fumili** (only for least-square or log-likelihood minimizations)
  - **GSLMultiMin**: conjugate gradient minimization algorithm from GSL (Fletcher-Reeves, BFGS)
  - **GSLMultiFit**: Levenberg-Marquardt (for minimizing least square functions) from GSL
  - **Linear** for least square functions (direct solution, non-iterative method)
  - **GSLSimAn**: Simulated Annealing from GSL
  - **Genetic**:   based on a genetic algorithm implemented in TMVA
- All these are available for ROOT fitting and in RooFit/RooStats
- Possible to combine them (e.g. use Minuit and Genetic)

Comments on Minimization

- Sometimes fit does not converge

```
Warning in <Fit>: Abnormal termination of minimization.
```

- can happen because the Hessian matrix is not positive defined
  - e.g. there are no minimum in that region ➞wrong initial parameters;
- numerical precision problems in the function evaluation
  - need to check and re-think on how to implement better the fit model function;
- highly correlated parameters in the fit. In case of 100% correlation the point solution becomes a line (or an hyper-surface) in parameter space. The minimization problem is no longer well defined.

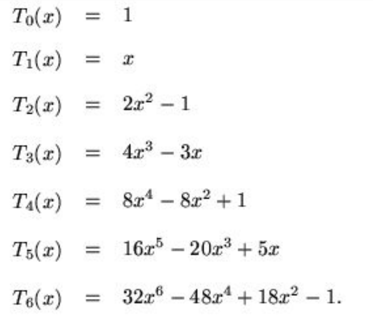


```
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL       1      2
       1  0.99835   1.000  0.998
       2  0.99835   0.998  1.000
```

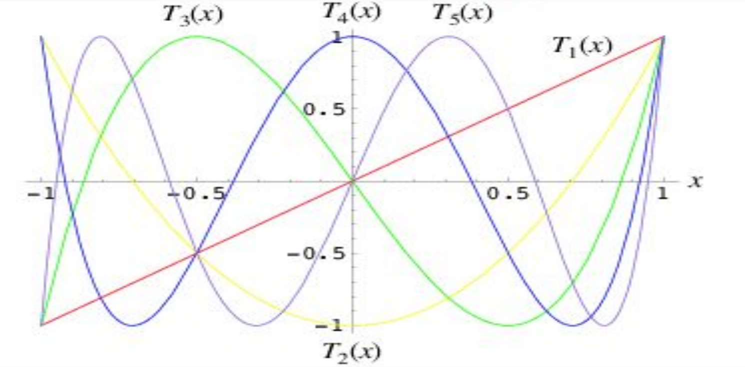
Signs of trouble...

Mitigating fit stability problems

- When using a polynomial parametrization:
  - $a_0+a_1x+a_2x^2+a_3x^3$ nearly always results in strong correlations between the coefficients.
    - problems in fit stability and inability to find the right solution at high order
- This can be solved using a better polynomial parametrization:
  - e.g. Chebychev polynomials

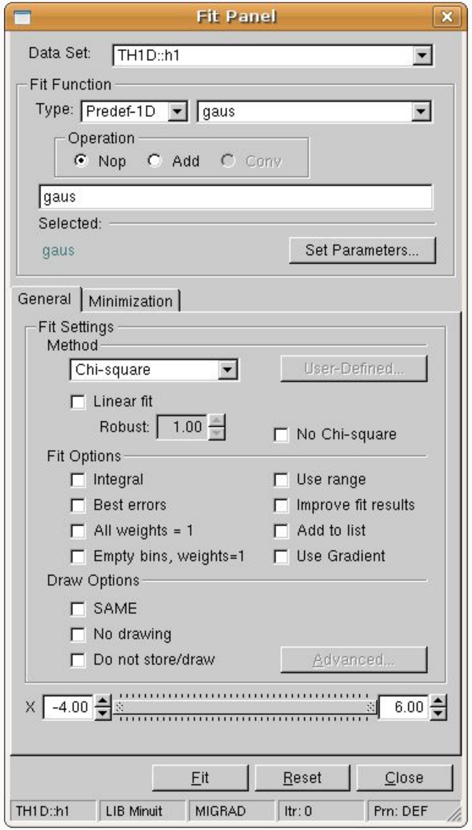


$$\begin{aligned}
T_0(x) &= 1\\
T_1(x) &= x\\
T_2(x) &= 2x^2-1\\
T_3(x) &= 4x^3-3x\\
T_4(x) &= 8x^4-8x^2+1\\
T_5(x) &= 16x^5-20x^3+5x\\
T_6(x) &= 32x^6-48x^4+18x^2-1.
\end{aligned}$$



The Fit Panel

- The fitting in ROOT using the FitPanel GUI
  - GUI for fitting all ROOT data objects
- Using the GUI we can:
  - select data object to fit
  - choose (or create) fit model function
  - set initial parameters
  - choose:
    - fit method (likelihood, chi2)
    - fit options (e.g Minos errors)
    - drawing options
  - change the fit range



Fit Panel

Data Set: TH1D::h1

Fit Function

Type: Predef-1D gaus

Operation

● Nop ○ Add ○ Conv

gaus

Selected:

gaus

Set Parameters...

General | Minimization

Fit Settings

Method

Chi-square

User-Defined...

□ Linear fit

Robust: 1.00

□ No Chi-square

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 -4.00 6.00

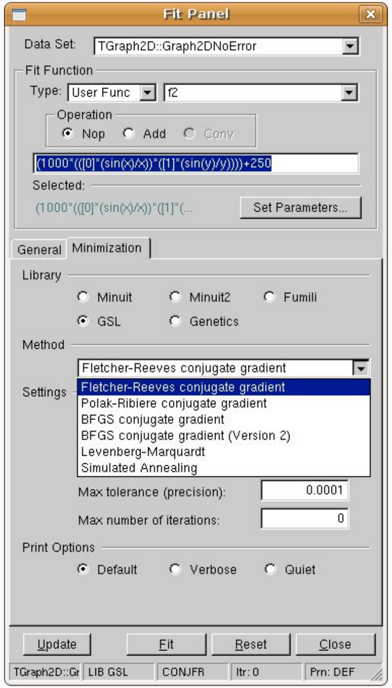
Fit Reset Close

TH1D::h1 | LIB Minuit | MIGRAD | Itr: 0 | Prn: DEF

Fit Panel (2)

- The Fit Panel provides also extra functionality:

Control the minimization



Data Set: TGraph2D::Graph2DNoError

Fit Function

Type: User Func | f2

Operation: ● Nop ○ Add ○ Conv

(1000*(([0]*(sin(x)/x))*([1]*(sin(y)/y))))+250

Selected:

(1000*(([0]*(sin(x)/x))*([1]*(... Set Parameters...

General | Minimization

Library

○ Minuit ○ Minuit2 ○ Fumili

● GSL ○ Genetics

Method

Fletcher-Reeves conjugate gradient

Settings

- Fletcher-Reeves conjugate gradient
- Polak-Ribiere conjugate gradient
- BFGS conjugate gradient
- BFGS conjugate gradient (Version 2)
- Levenberg-Marquardt
- Simulated Annealing

Max tolerance (precision): 0.0001

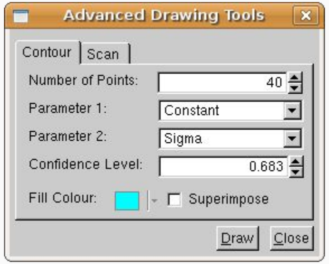
Max number of iterations: 0

Print Options

● Default ○ Verbose ○ Quiet

TGraph2D::Gr | LIB GSL | CONJFR | Itr: 0 | Prn: DEF

Contour plot



Number of Points: 40

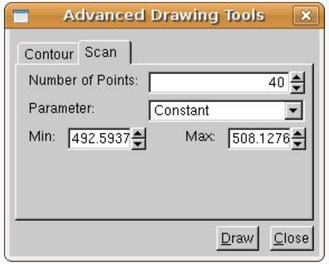
Parameter 1: Constant

Parameter 2: Sigma

Confidence Level: 0.683

Fill Colour: 🟦 □ Superimpose

Scan plot of minimization function

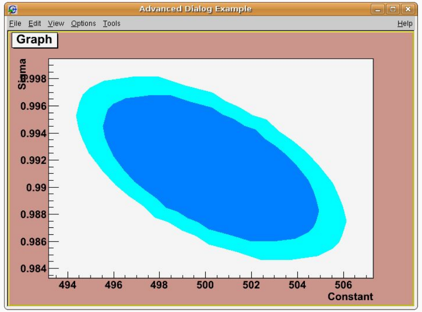


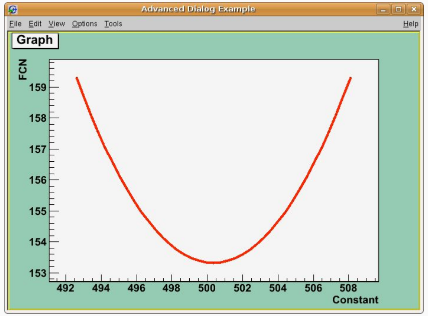
Number of Points: 40

Parameter: Constant

Min: 492.5937 Max: 508.1276

Advanced drawing tools





Time For Exercises

https://github.com/root-project/training/tree/master/BasicCourse/Exercises/Fitting

Day 2

This Course

Divided in 10 “Learning modules” over two days

- Day 1:
  - Introduction
  - C++ Interpreter
  - Histograms, Graphs and Functions
  - Graphics
  - Fitting
- Day 2:
  - Python Interface
  - ROOTBooks
  - Working with Files
  - Working with Columnar Data
  - Developing Packages
- Front lectures and Hands-On exercises

A Little Python

Learning Objectives

- Comfortably handle the basic Python that ROOT requires
- Be able to use the Python shell and write simple scripts
- Know the basic Python types
- Be able to define and call functions

The Python Language

- Focus on readability and productivity
- Interpreted
- Dynamically typed
- Support for object-oriented programming
- Increasing popularity in scientific computing and HEP

Python 2 vs Python 3

- 3.x series started a backwards incompatible line
- 2.x is legacy
    - Large codebase and still used a lot
- 3.x is the future
    - Adoption will take some time
- ROOT supports both Python 2 and Python 3
    - Examples in this tutorial work with both

How to Run Python?

- Python interactive shell

```
> python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> print('Hello Python!')
Hello Python!
>>> exit()
```

How to Run Python (II)?

- Python script

myscript.py

```
print('Hello Python!')
```

```
> python myscript.py
Hello Python!
```

How to Run Python (III)?

- Python script + interactive

myscript.py

```python
print('Hello Python!')
```

```
> python -i myscript.py
Hello Python!
>>>
```

Variable Assignment

- Syntax: *variable_name* = *value*
- No type declaration

```
message = 'Hello Python!'
```

Basic Types

```python
# string (also with "")
message = 'Hello Python!'

# integer
year = 2017

# floating point
pi = 3.14159265

# boolean
is_python = True

# None (no value)
nothing = None
```

comment

Operators

```
# arithmetic
a = (50 - 5 * 6) / 4
b = a + 2

# string concatenation
particles = 'leptons' + ' quarks'
particles += ' gluons'
```

Lists

```python
# lists are usually homogeneous
l = ['quarks', 'leptons']

# lists are mutable
l.append('gluons')  # ['quarks', 'leptons', 'gluons']

# list length
len(l)  # 3

# element access
l[0]  # 'quarks'

# list slicing
l[1:3]  # ['leptons', 'gluons']
```

Tuples

```
# tuples are usually heterogeneous
t = ('Python', 3)

# tuples are immutable
t[1] = 2  # ERROR!

# tuple length
len(t)  # 2

# element access
t[0]  # 'Python'

# tuple slicing
t[:1]  # ('Python')
```

Dictionaries

```
# dictionaries store key-value pairs
person = { 'name' : 'John' }

# add a key-value pair
person['age'] = 20

# get value for a key
person['name']  # 'John'

# get keys, values and pairs
person.keys()    # ['name', 'age']
person.values()  # ['John', 20]
person.items()   # [('name','John'), ('age',20)]
```

Conditionals - Indentation

- Python uses indentation to delimit blocks

```python
# Conditional statement, prints 'Positive'
x = 50
if x < 0:
  print('Negative')
elif x == 0:
  print('Zero')
else:
  print('Positive')
```

block

Loops

```python
# while loop
x = 0
while x < 10:
  print(x)
  x += 1

# for loop with range
for y in range(10):  # 0-9
  print(y)

# for loop to iterate on a list
cities = ['Barcelona', 'Milano', 'Geneva']
for c in cities:
  print(c)
```

List Comprehensions

- Construct lists with a more concise syntax
  - Useful for replacing simple for-loops

```python
# for loop that fills a list
odds = []
for x in range(50):
  if x % 2:
    odds.append(x)

# equivalent code with list comprehension syntax
odds = [ x for x in range(50) if x % 2 ]
```

Functions

- Basic function definition

```python
def my_function():
    """Function documentation"""
    print('Hello Python')
```

Function Calls

```
# Function with two arguments, returns sum
def add(x, y):
  return x + y

# Call, positional arguments
result = add(5, 10)  # result = 15

# Call, keyword arguments
result = add(y = 'B', x = 'A')  # result = 'AB'
```

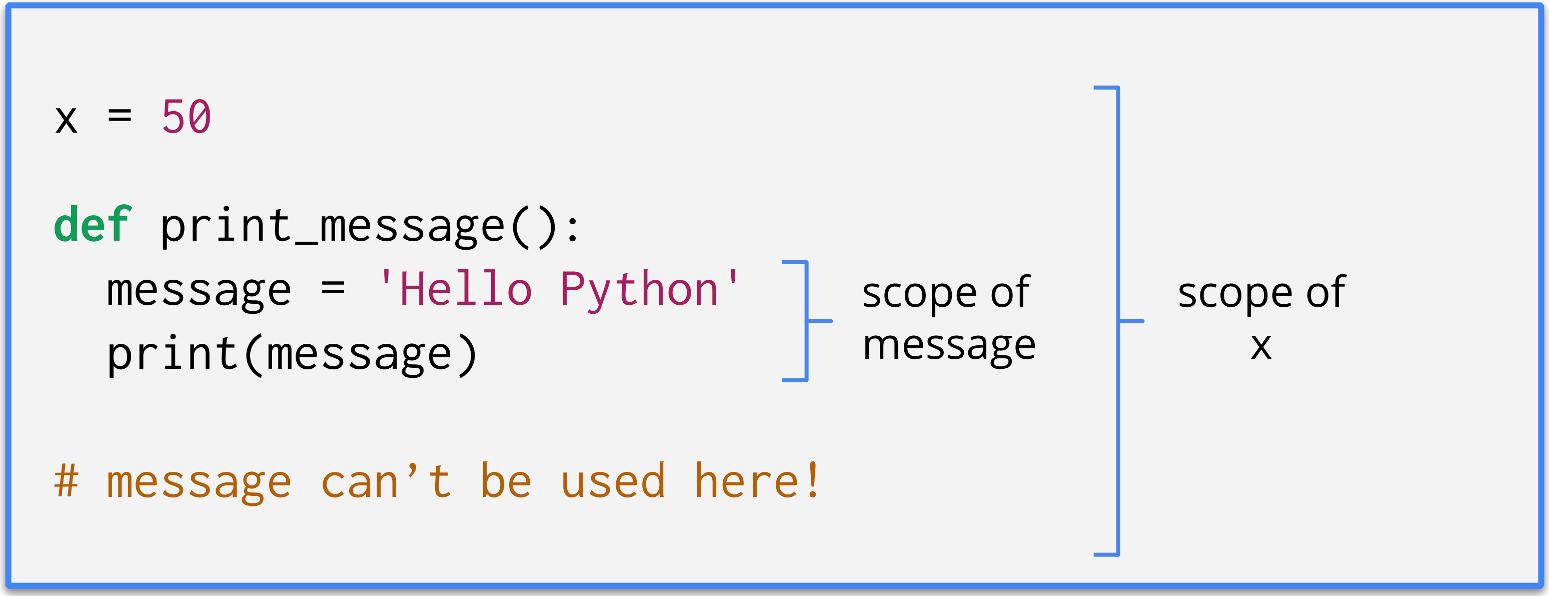
no type for
args or return

Lambdas

```
# Anonymous function
f = lambda x, y: x + y

# Call, positional arguments
result = f(5, 10)  # result = 15

# Call, keyword arguments
result = f(y = 'B', x = 'A')  # result = 'AB'
```

Scope

- Variables have a scope that determines their lifetime



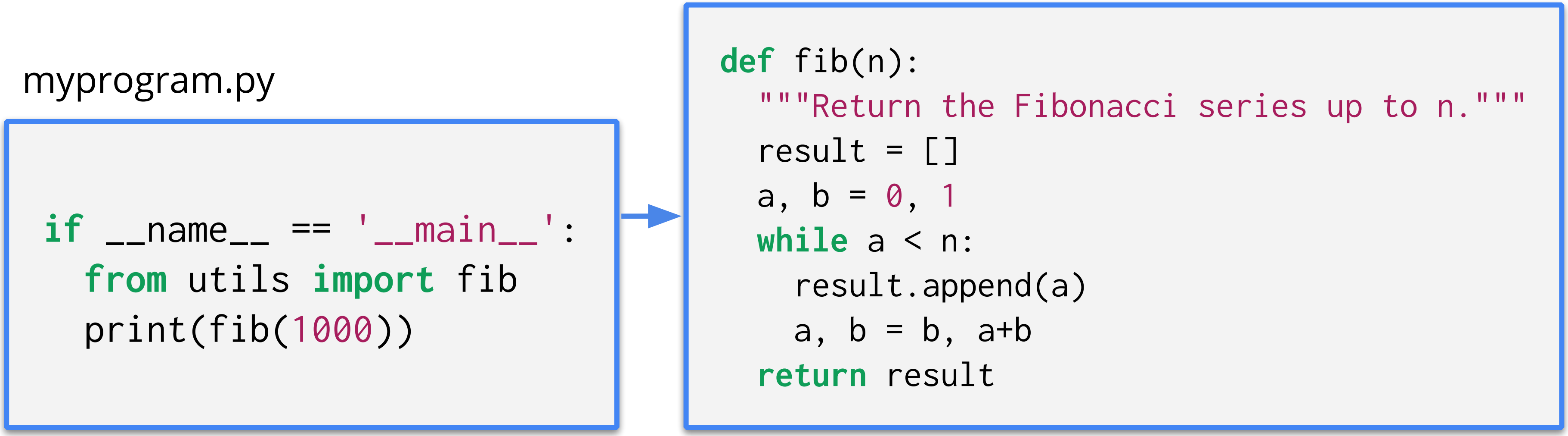
```
x = 50

def print_message():
    message = 'Hello Python'
    print(message)

# message can’t be used here!
```

Modules, Imports

- A module is a file that contains Python code
  - Defines variables, functions, classes
  - Allows code isolation and reuse
- Modules can be imported into current namespace

```python
# Import a module and access one of its variables
import mymodule
mymodule.myvar

# Import a specific function from the module
from mymodule import myfunction
myfunction()
```

Fibonacci Example



myprogram.py

```python
if __name__ == '__main__':
  from utils import fib
  print(fib(1000))
```

utils.py

```python
def fib(n):
  """Return the Fibonacci series up to n."""
  result = []
  a, b = 0, 1
  while a < n:
    result.append(a)
    a, b = b, a+b
  return result
```

```
> python myprogram.py
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Time For Exercises

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

Time For Exercises

- Create a Python module that defines two functions:
  - *count(text)*: given a string parameter that contains some text, it returns the number of words in that text (hint: use *split* function of string)
  - *find(text, word)*: given two string parameters, a text and a word, it returns true if the word is in the text (hint: convert the text into a list with *split* and iterate on it)
- Either from the Python shell or from a script, invoke the defined functions
  - Count the number of words in "*Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.*"
  - Look for the word *magna*

PyROOT: The ROOT Python Bindings

Binding Python and C++

- C and C++ needed for performance-critical code
- Python enables faster development
- Ideally, we should combine them
  - Python as an interface to C++ functionality
- There are ways to invoke C and C++ from Python
  - ctypes, boost, swig
  - Cumbersome, wrappers needed, lots of work
- **In ROOT: PyROOT**

PyROOT

- Python bindings for ROOT
- Access all the ROOT C++ functionality from Python
- Dynamic, automatic
  - Include header, load library -> start interactive usage
- Communication between Python and C++ is powered by the ROOT type system
- "Pythonisations" for specific cases

Interrogating the Type System

```
auto c = TClass::GetClass("myClass")
```

- Fetch the *myClass* representation of ROOT

```
auto ms = c->GetListOfMethods()
```

- Get the methods of the class (appreciate how this is impossible in C++!)

```
auto listOfXXX = gROOT->GetListOfXXX()
```

- Same for collections of many other entities

Using PyROOT

- Entry point to use ROOT from within Python

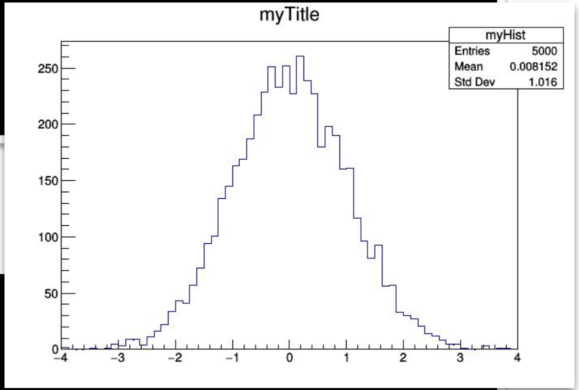
```
import ROOT
```

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

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

Example: C++ to Python

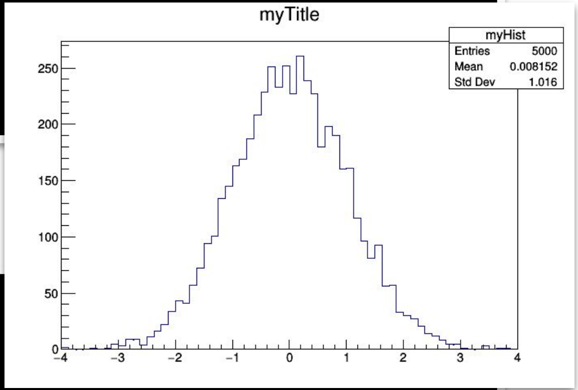
```
> root
root [0] TH1F h("myHist", "myTitle", 64, -4, 4)
root [1] h.FillRandom("gaus")
root [2] h.Draw()
```



```
> 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()
```

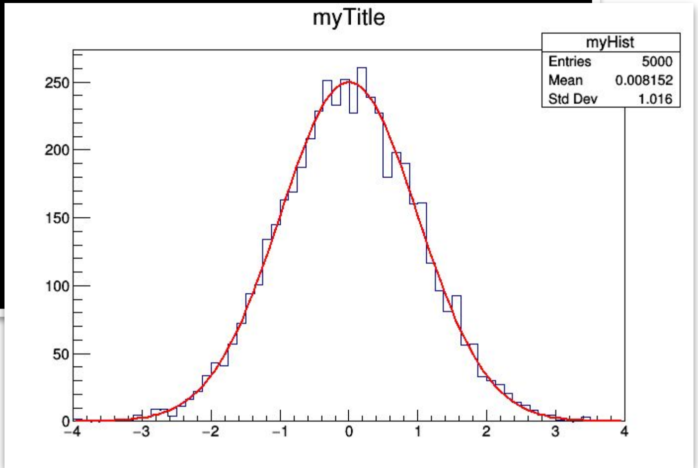
Time For Exercises

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

Exercise Function

- Complete this example in Python:

```
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")
```



Exercise Histogram

- Open the Python interpreter
- Import the ROOT module
- Create a histogram with 64 bins and a axis ranging from 0 to 16
- Fill it with random numbers distributed according to a linear function (*pol1*)
- Change its line width with a thicker one
- Draw it!

Exercise Graph

- Create a new Python module
- In the module, 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 an orange line between points

Dynamic C++ (JITting)

```python
import ROOT
cpp_code = """
int f(int i) { return i*i; }
class A {
public:
  A() { cout << "Hello PyROOT!" << endl; }
};
"""

# Inject the code in the ROOT interpreter
ROOT.gInterpreter.ProcessLine(cpp_code)

# We find all the C++ entities in Python!
a = ROOT.A()   # this prints Hello PyROOT!
x = ROOT.f(3)  # x = 9
```

C++ code we
want to invoke
from Python

Dynamic C++ (JITting)

my_cpp_library.h

```cpp
int f(int i) { return i*i; }

class A {
public:
  A() { cout << "Hello PyROOT!" << endl; }
};
```

my_python_module.py

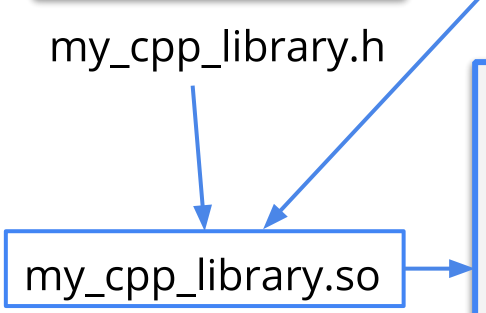
```python
# Make the header known to the interpreter
ROOT.gInterpreter.ProcessLine('#include "my_cpp_library.h"')

# We find all the C++ entities in Python!
a = ROOT.A()   # this prints Hello PyROOT!
x = ROOT.f(3)  # x = 9
```

Dynamic Library Loading

```
int f(int i);

class A {
public:
  A();
};
```

```
#include "my_cpp_library.h"

int f(int i) { return i*i; }

A::A() { cout << "Hello PyROOT!" << endl; }
```

my_cpp_library.cpp



my_cpp_library.h

my_cpp_library.so

my_python_module.py

```
# Load a C++ library
ROOT.gInterpreter.ProcessLine('#include "my_cpp_library.h"')
ROOT.gSystem.Load('./my_cpp_library.so')

# We find all the C++ entities in Python!
a = ROOT.A()   # this prints Hello PyROOT!
x = ROOT.f(3)  # x = 9
```

Exercise Dynamic JITting

- Define a C++ function that counts the characters in an std::string (hint: use std::string::size)
- Make that function known to the ROOT interpreter in a Python module
- Invoke the function via PyROOT

- *Extra*: practise the three scenarios described
  - JITted string
  - JITted header
  - Library loading

Exercise Fitting

- Start by defining the following functions via the C++ interpreter:

```
// Quadratic background function
double background(double *x, double *par) {
  return par[0] + par[1]*x[0] + par[2]*x[0]*x[0];
}

// Lorenzian peak function
double lorentzianPeak(double *x, double *par) {
  return (0.5*par[0]*par[1]/TMath::Pi()) /
    TMath::Max(1.e-10,(x[0]-par[2])*(x[0]-par[2])
    + .25*par[1]*par[1]);
}

// Sum of background and peak function
double fitFunction(double *x, double *par) {
  return background(x, par) + lorentzianPeak(x, &par[3]);
}
```

Exercise Fitting (II)

- Construct and fill a TH1F given the following number of bins, bin content data and x range (hint: use *TH1F::SetBinContent*)

```
nbins = 60
data = [ 6, 1, 10,12,6, 13,23,22,15,21,
         23,26,36,25,27,35,40,44,66,81,
         75,57,48,45,46,41,35,36,53,32,
         40,37,38,31,36,44,42,37,32,32,
         43,44,35,33,33,39,29,41,32,44,
         26,39,29,35,32,21,21,15,25,15 ]
xlow = 0
xup = 3
```

Exercise Fitting (III)

- Create a TF1 with the range from 0 to 3 and 6 parameters. Use the *fitFunction* function that you declared in the first step, which is the addition of a polynomial background and a Lorentzian function
- Try to fit the function with the histogram without setting any parameter. This will result in a good fit for the polynomial function but not for the Lorentzian
- Play with the values of parameters 4 and 5. What happens if you set 4 to 0.2 and 5 to 1, and fit again?
- Can you draw the histogram with error bars?
- Using the fitting results, construct TF1s for the background and Lorentzian functions and draw them in the same canvas
- Create a legend with an entry for each function and the histogram

The ROOTBooks

Learning Objectives

- Become familiar with Jupyter Notebooks
- Use ROOT in Notebooks using SWAN
- Appreciate the differences and advantages of a Notebook-based approach with respect to a traditional command-line prompt

The Jupyter Notebook

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

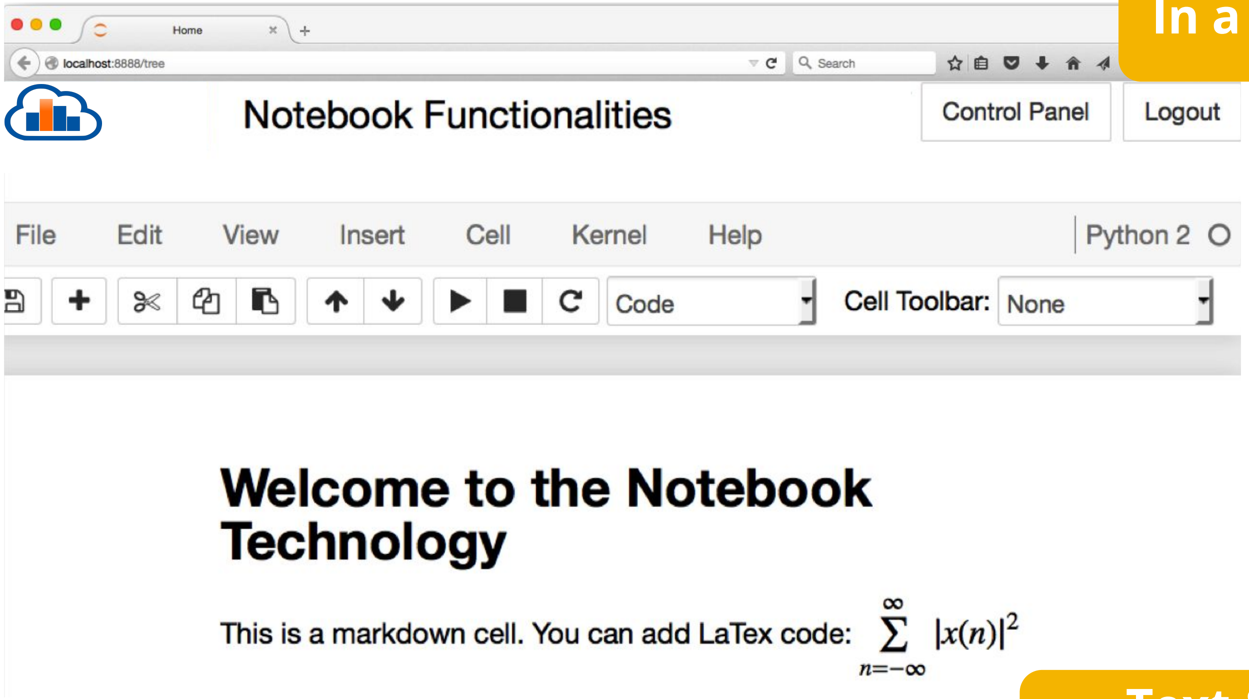
Many supported languages: Python, Haskell, Julia… One generally speaks about a “kernel” for a specific language

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

In the past also called:
iPython Notebooks

http://www.jupyter.org

In a browser

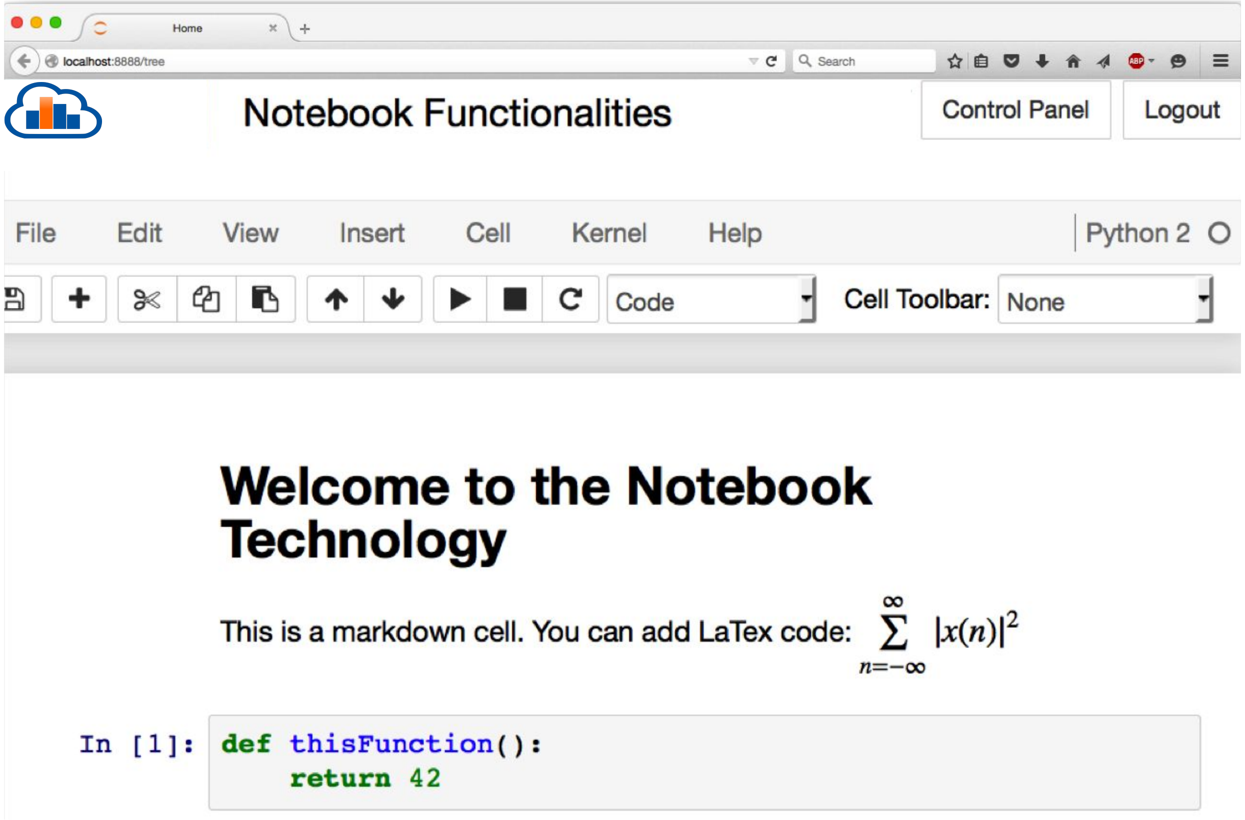


Notebook Functionalities

Welcome to the Notebook Technology

This is a markdown cell. You can add LaTex code: $\sum_{n=-\infty}^{\infty} |x(n)|^2$

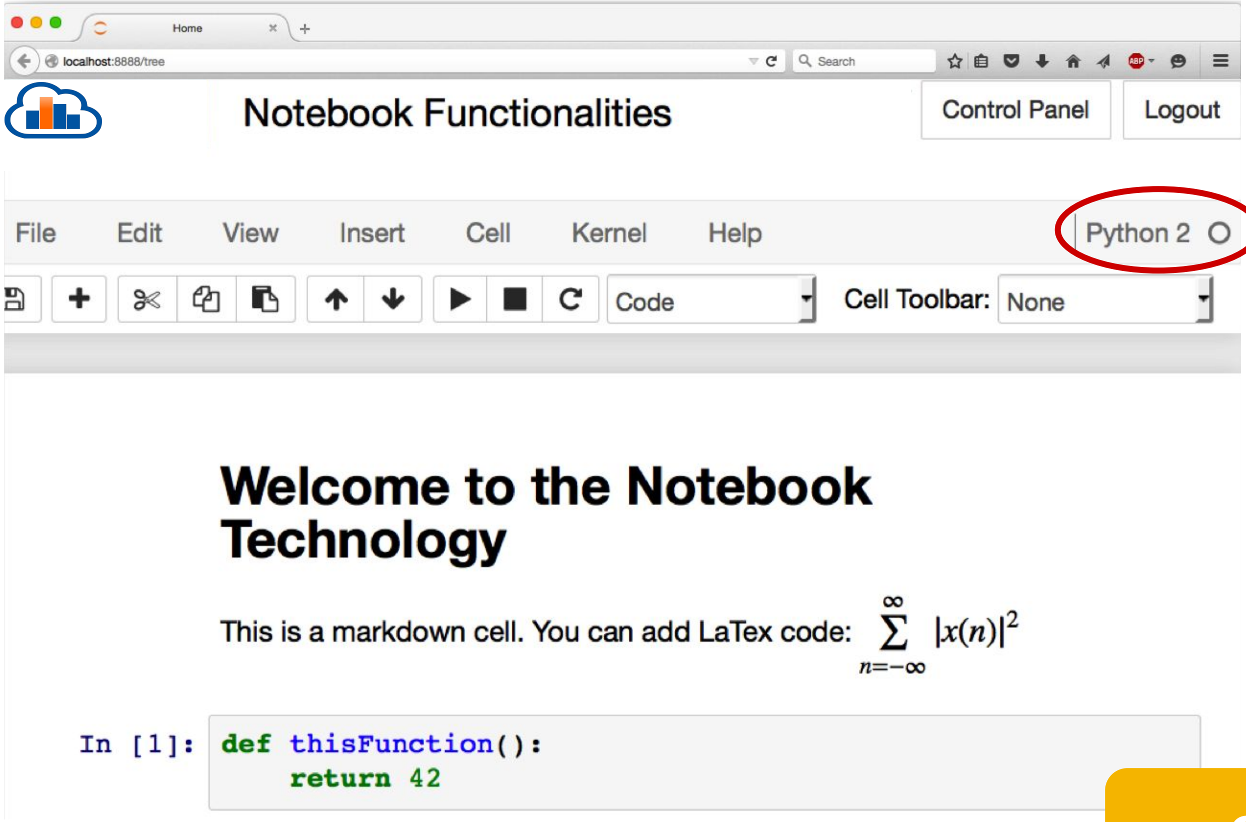
Text & Formulas



Welcome to the Notebook Technology

This is a markdown cell. You can add LaTex code: $\sum_{n=-\infty}^{\infty} |x(n)|^2$

```
In [1]: def thisFunction():
            return 42
```



Notebook Functionalities

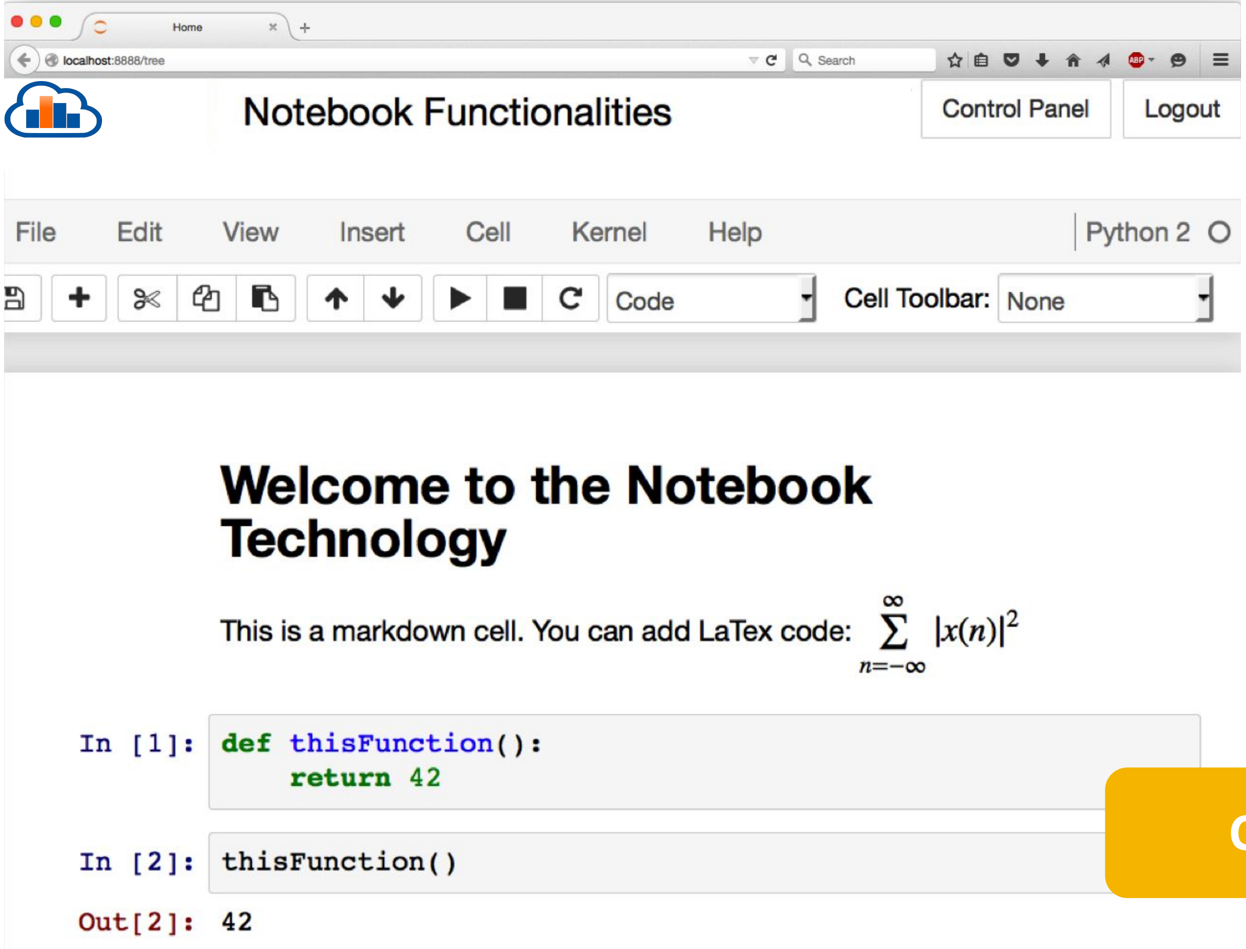
Welcome to the Notebook Technology

This is a markdown cell. You can add LaTex code: $\sum_{n=-\infty}^{\infty} |x(n)|^2$

```
In [1]: def thisFunction():
            return 42
```

Code

A *Python* notebook



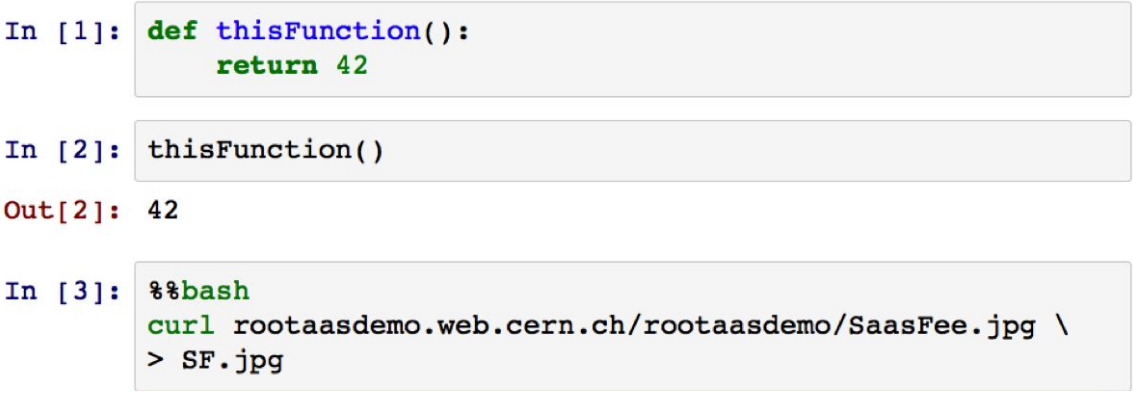
Welcome to the Notebook Technology

This is a markdown cell. You can add LaTex code: $\sum_{n=-\infty}^{\infty} |x(n)|^2$

```
In [1]: def thisFunction():
            return 42
```

```
In [2]: thisFunction()

Out[2]: 42
```

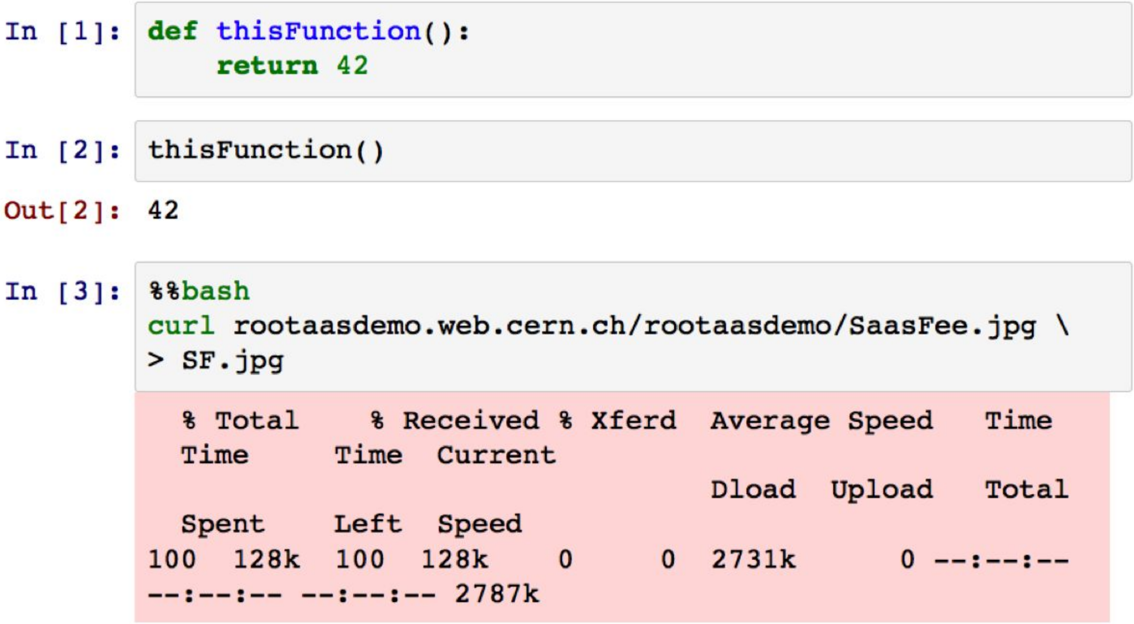
Code



```
In [1]: def thisFunction():
            return 42

In [2]: thisFunction()

Out[2]: 42

In [3]: %%bash
        curl rootaasdemo.web.cern.ch/rootaasdemo/SaasFee.jpg \
        > SF.jpg
```

Invoke shell commands ...

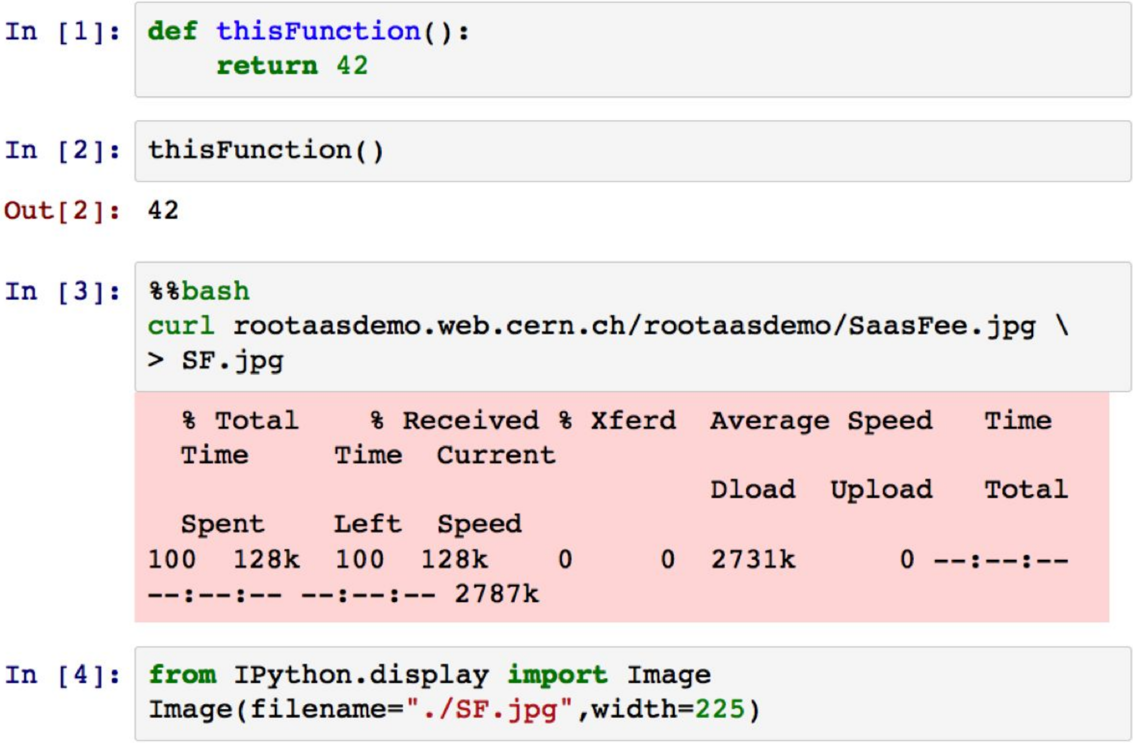
Shell commands



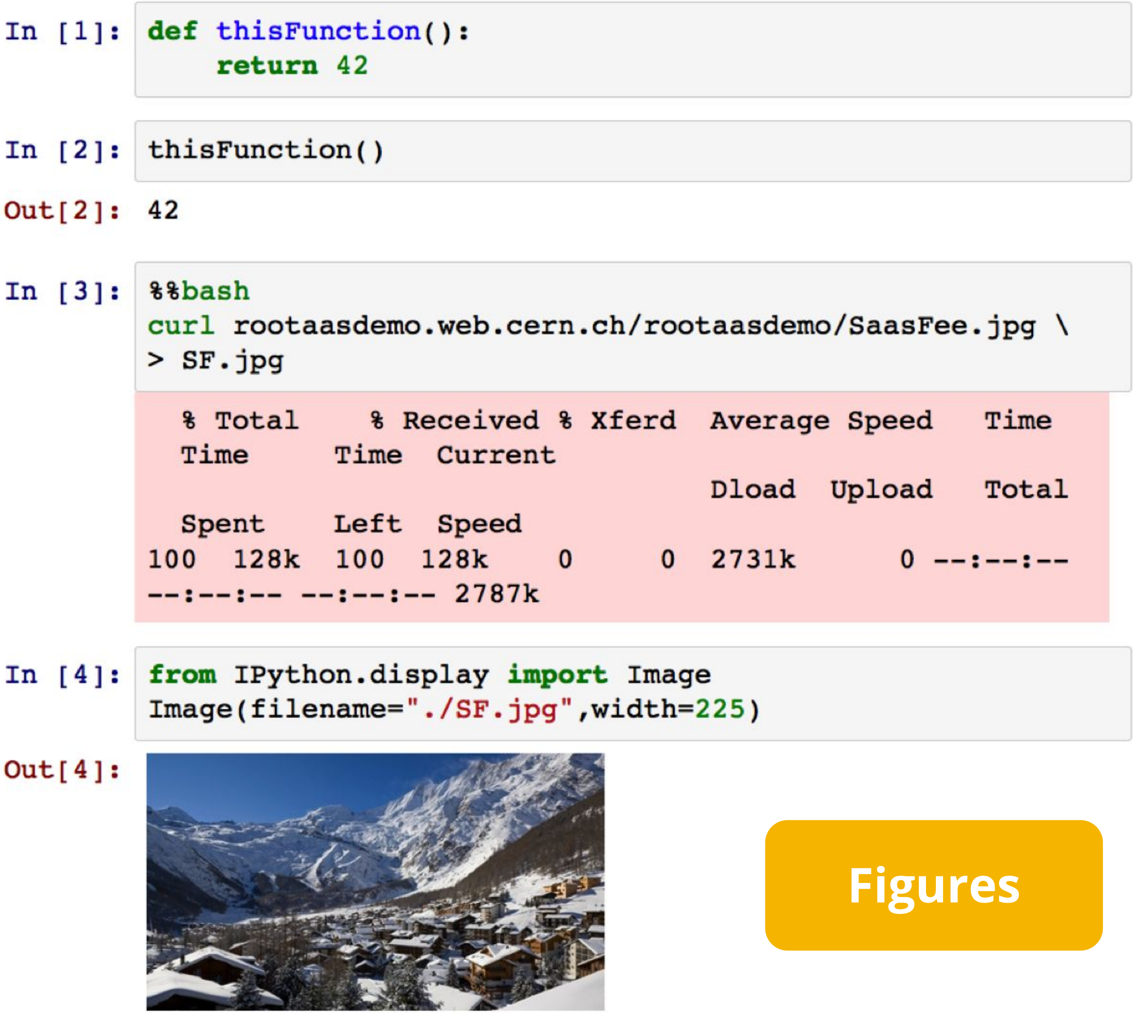
```
In [1]: def thisFunction():
            return 42

In [2]: thisFunction()

Out[2]: 42

In [3]: %%bash
        curl rootaasdemo.web.cern.ch/rootaasdemo/SaasFee.jpg \
        > SF.jpg

          % Total    % Received % Xferd  Average Speed   Time
          Time     Time  Current
                                         Dload  Upload   Total
          Spent    Left  Speed
        100  128k  100  128k    0     0  2731k      0 --:--:--
        --:--:-- --:--:-- 2787k
```

... and get their output

Shell commands



```
In [1]: def thisFunction():
            return 42

In [2]: thisFunction()

Out[2]: 42

In [3]: %%bash
        curl rootaasdemo.web.cern.ch/rootaasdemo/SaasFee.jpg \
        > SF.jpg

          % Total    % Received % Xferd  Average Speed   Time
          Time     Time  Current
                                         Dload  Upload   Total
          Spent    Left  Speed
        100  128k  100  128k    0     0  2731k      0 --:--:--
        --:--:-- --:--:-- 2787k

In [4]: from IPython.display import Image
        Image(filename="./SF.jpg",width=225)
```



```
In [1]: def thisFunction():
            return 42
```

```
In [2]: thisFunction()

Out[2]: 42
```

```
In [3]: %%bash
        curl rootaasdemo.web.cern.ch/rootaasdemo/SaasFee.jpg \
        > SF.jpg
```

```
  % Total    % Received % Xferd  Average Speed   Time
  Time     Time  Current
                                 Dload  Upload   Total
  Spent    Left  Speed
100  128k  100  128k    0     0  2731k      0 --:--:--
--:--:-- --:--:-- 2787k
```

```
In [4]: from IPython.display import Image
        Image(filename="./SF.jpg",width=225)

Out[4]:
```

Figures

No excuse not to document your analysis !!

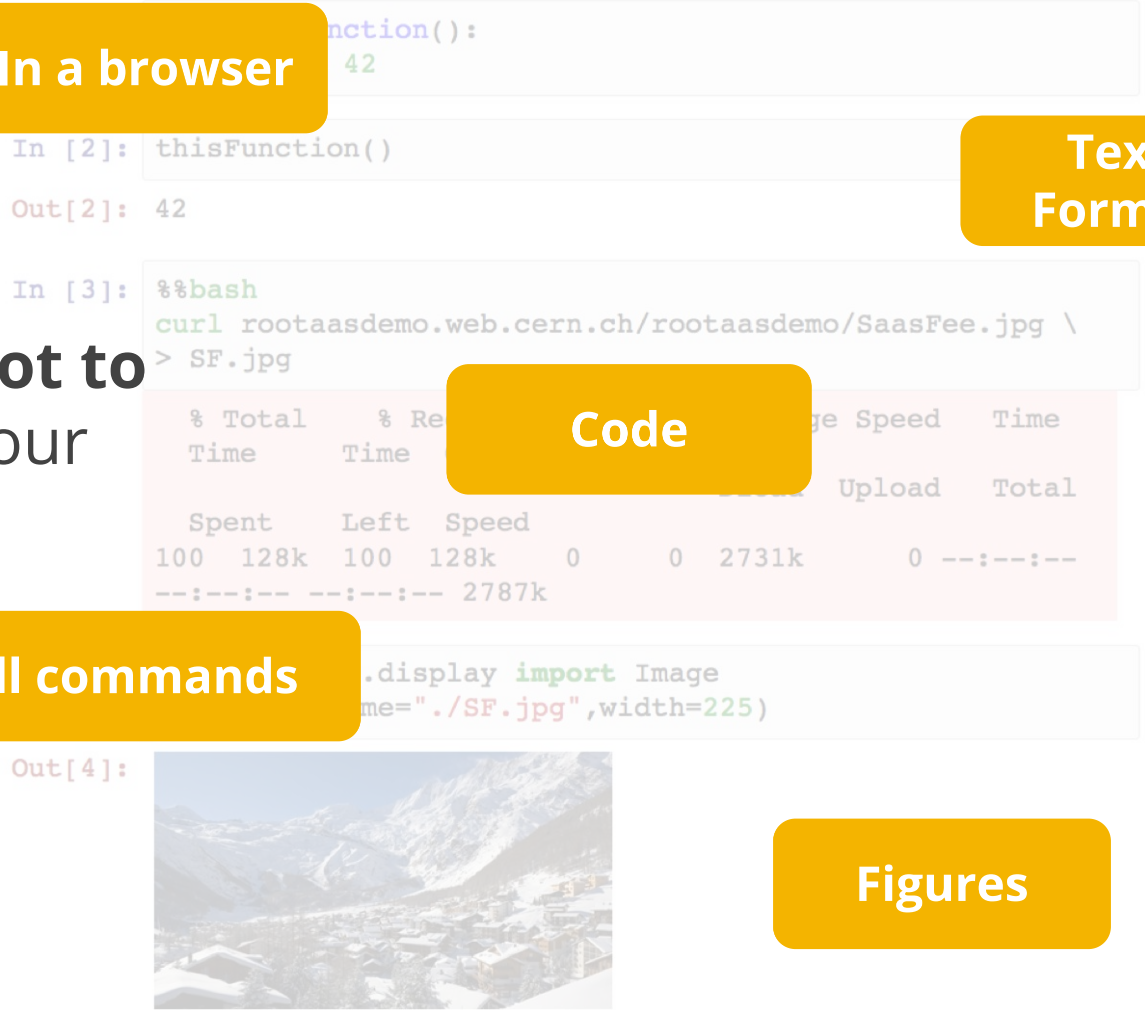
In a browser

Text & Formulas

Code

Shell commands

Figures



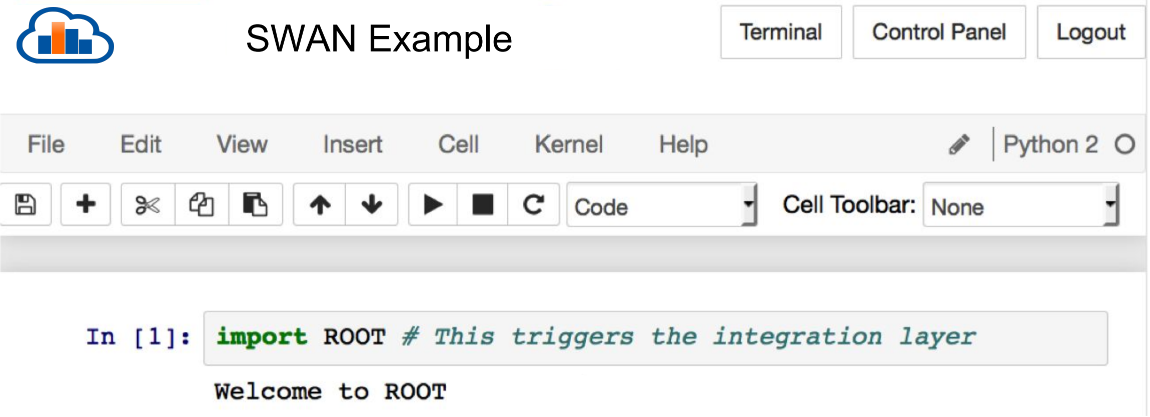
```
                 nction():
                   42

In [2]:  thisFunction()

Out[2]:  42

In [3]:  %%bash
         curl rootaasdemo.web.cern.ch/rootaasdemo/SaasFee.jpg \
         > SF.jpg

           % Total     % Re                     ge Speed    Time
           Time        Time

                                                   Upload    Total

           Spent       Left   Speed
         100   128k  100   128k     0       0   2731k       0 --:--:--
         --:--:-- --:--:-- 2787k

                              .display import Image
                              me="./SF.jpg",width=225)

Out[4]:
```

ROOTBooks

Integrate **ROOT** and note**books** - Goals:

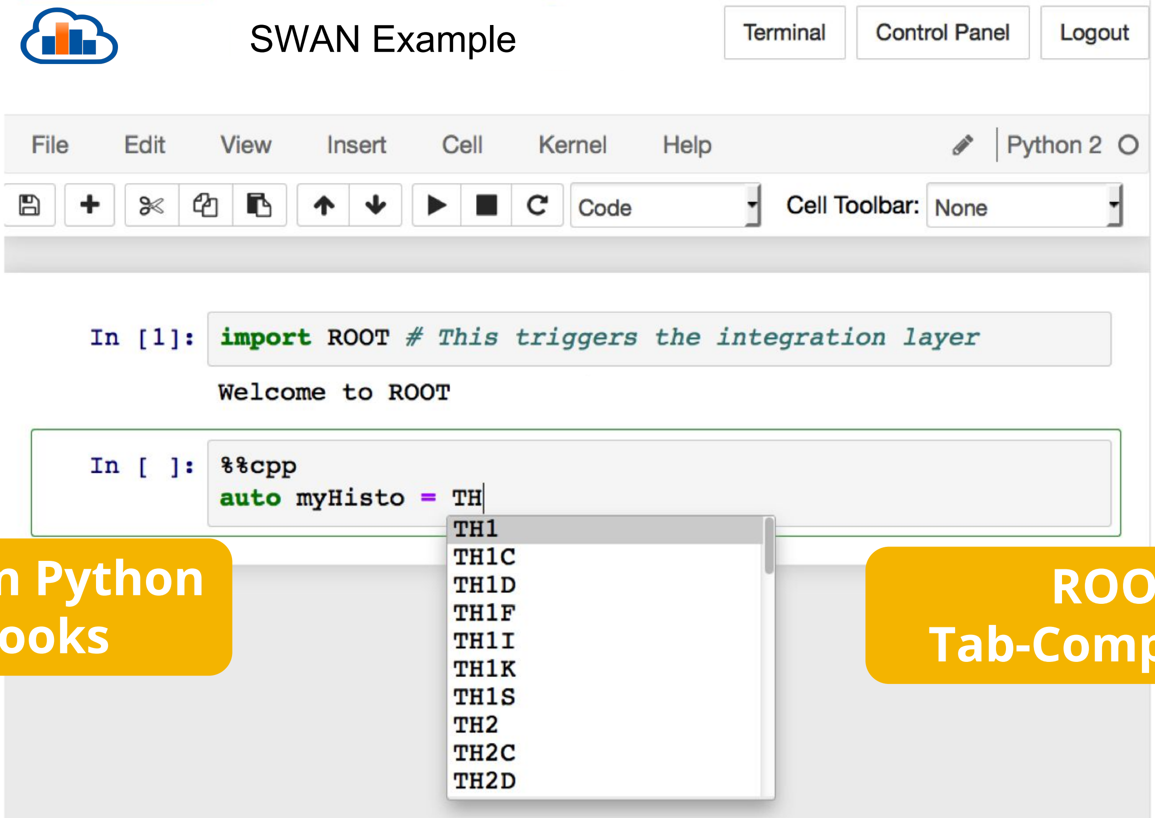
- Complement macros and command line prompts
- Encourage complete documentation
- Interoperability with other tools
    - E.g. from the Python ecosystem

Some Goodies



```
In [1]: import ROOT # This triggers the integration layer

Welcome to ROOT
```

Some Goodies



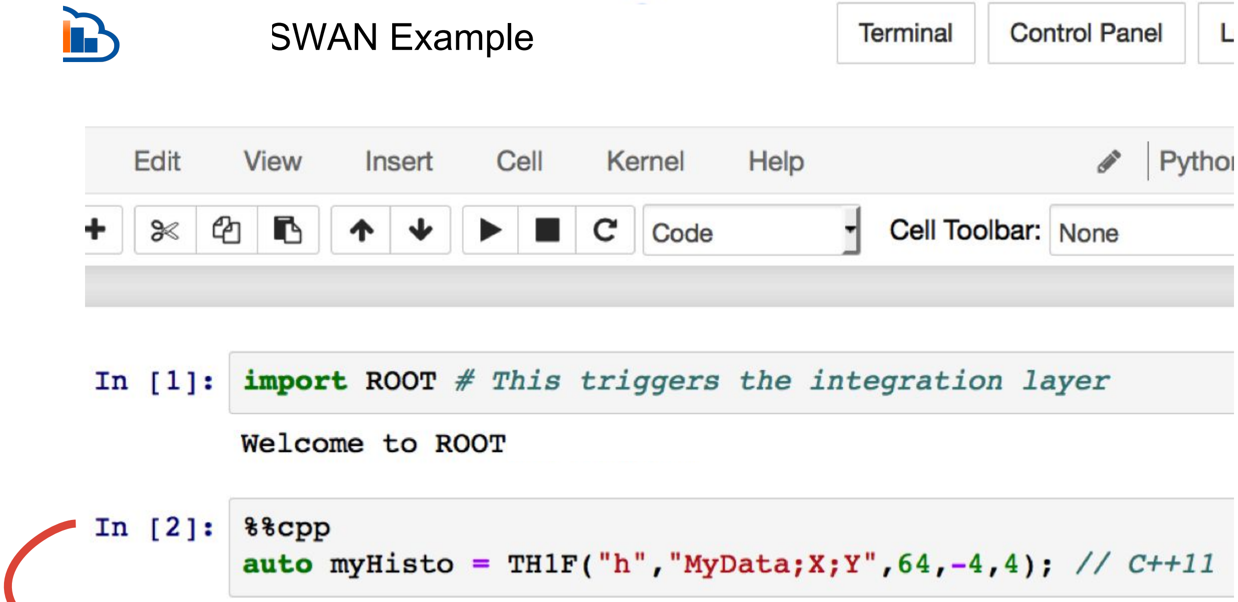
```
In [1]: import ROOT # This triggers the integration layer

Welcome to ROOT

In [ ]: %%cpp
auto myHisto = TH
```

```
TH1
TH1C
TH1D
TH1F
TH1I
TH1K
TH1S
TH2
TH2C
TH2D
```

C++ Cells in Python Notebooks

ROOT Tab-Completion

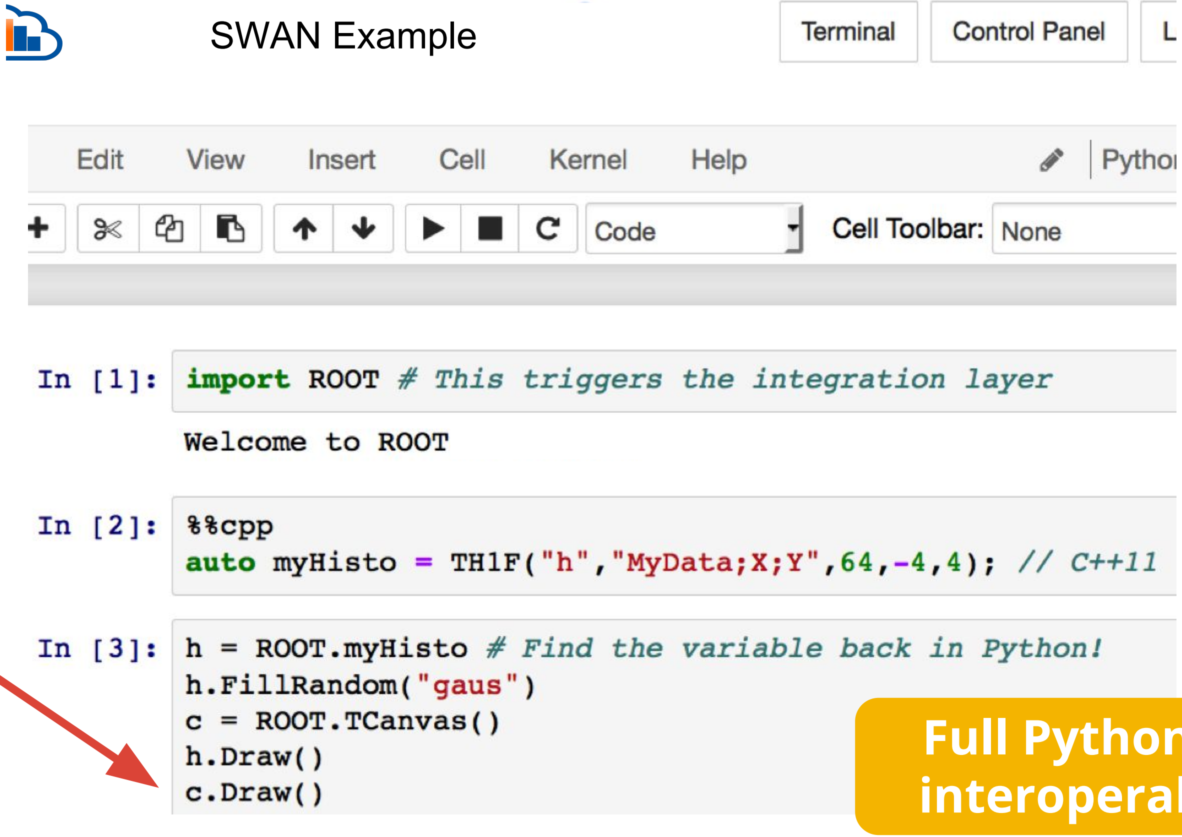
Some Goodies



```
In [1]: import ROOT # This triggers the integration layer

        Welcome to ROOT

In [2]: %%cpp
        auto myHisto = TH1F("h","MyData;X;Y",64,-4,4); // C++11
```

I really wanted to show modern C++...

Some Goodies



```
In [1]: import ROOT # This triggers the integration layer
```

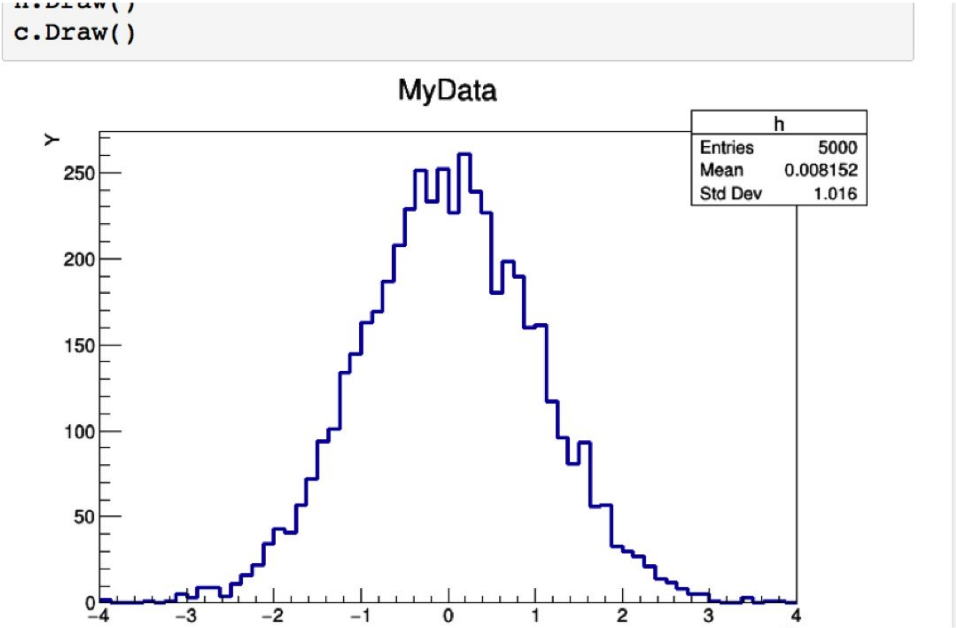
```
Welcome to ROOT
```

```
In [2]: %%cpp
        auto myHisto = TH1F("h","MyData;X;Y",64,-4,4); // C++11
```

```
In [3]: h = ROOT.myHisto # Find the variable back in Python!
        h.FillRandom("gaus")
        c = ROOT.TCanvas()
        h.Draw()
        c.Draw()
```

Full Python-C++ interoperability

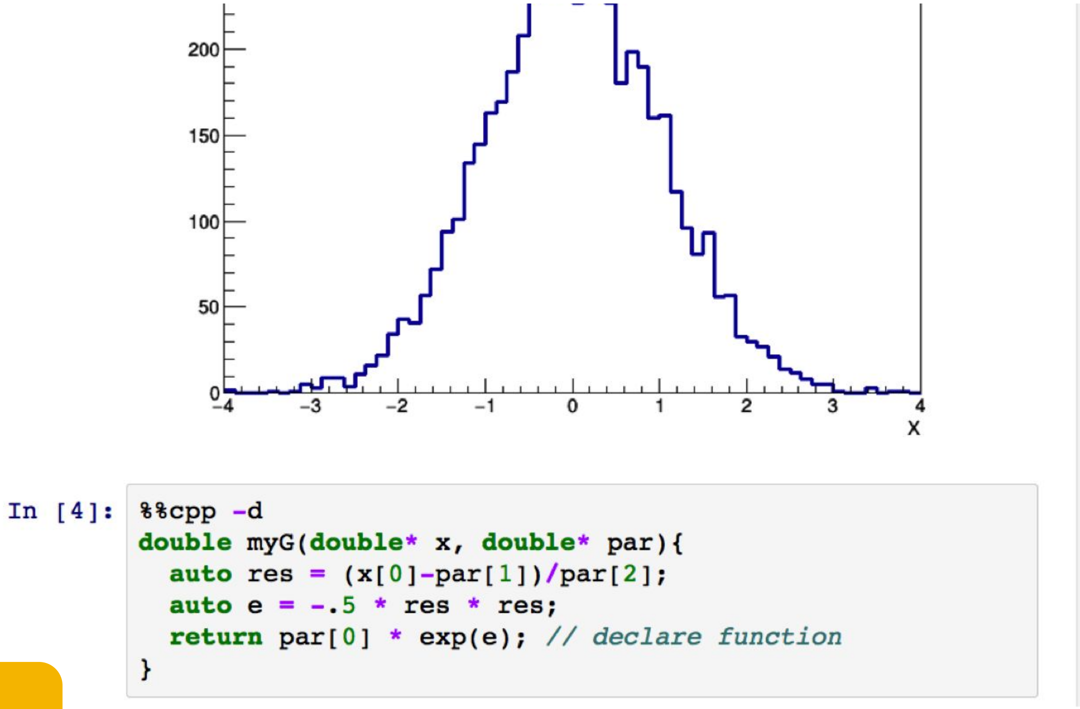
Some Goodies



```
h.Draw()
c.Draw()
```

Seamless display of graphics

Some Goodies

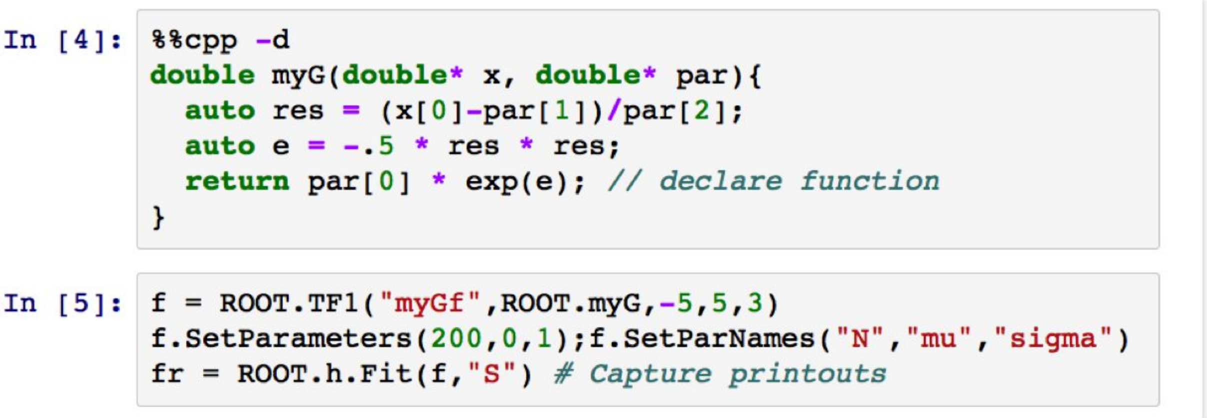


In [4]:

```
%%cpp -d
double myG(double* x, double* par){
  auto res = (x[0]-par[1])/par[2];
  auto e = -.5 * res * res;
  return par[0] * exp(e); // declare function
}
```

Syntax highlighting

Some Goodies



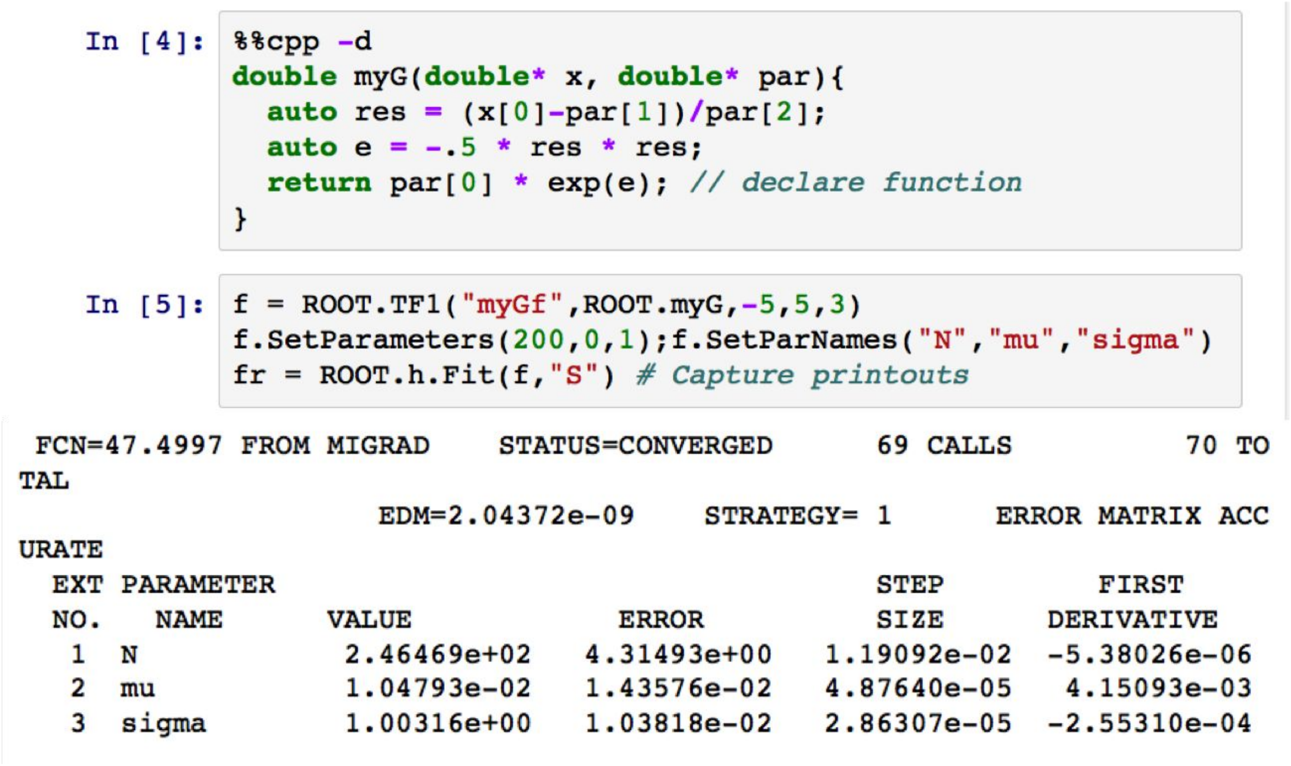
In [4]:

```
%%cpp -d
double myG(double* x, double* par){
  auto res = (x[0]-par[1])/par[2];
  auto e = -.5 * res * res;
  return par[0] * exp(e); // declare function
}
```

In [5]:

```
f = ROOT.TF1("myGf",ROOT.myG,-5,5,3)
f.SetParameters(200,0,1);f.SetParNames("N","mu","sigma")
fr = ROOT.h.Fit(f,"S") # Capture printouts
```

Some Goodies



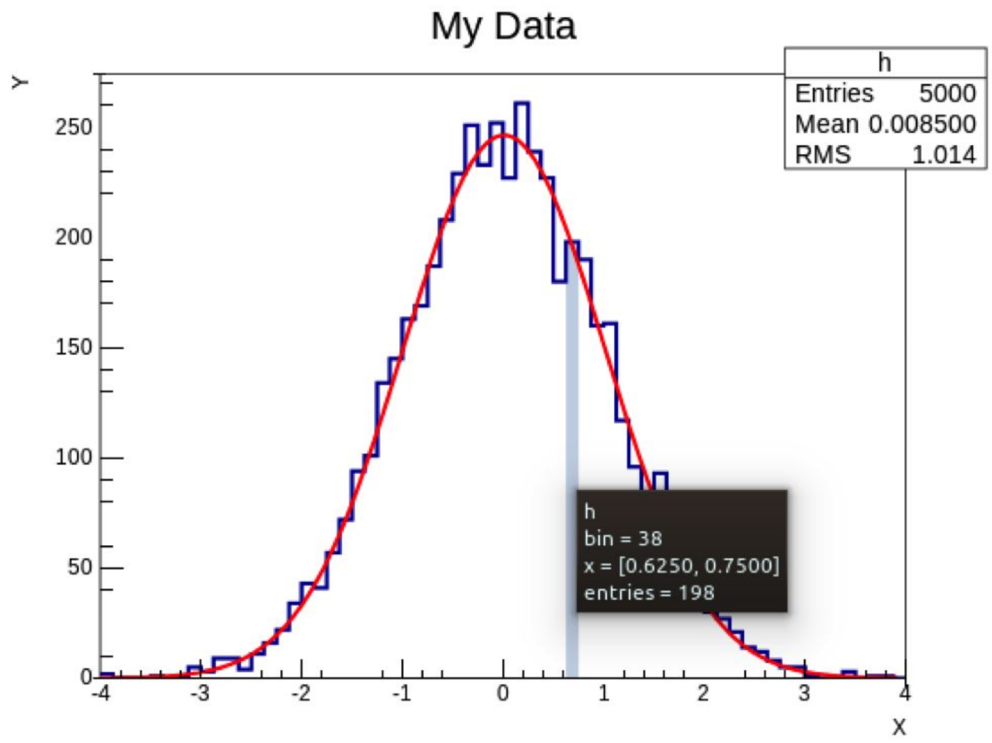
```
In [4]: %%cpp -d
        double myG(double* x, double* par){
          auto res = (x[0]-par[1])/par[2];
          auto e = -.5 * res * res;
          return par[0] * exp(e); // declare function
        }
```

```
In [5]: f = ROOT.TF1("myGf",ROOT.myG,-5,5,3)
        f.SetParameters(200,0,1);f.SetParNames("N","mu","sigma")
        fr = ROOT.h.Fit(f,"S") # Capture printouts
```

```
 FCN=47.4997 FROM MIGRAD    STATUS=CONVERGED      69 CALLS          70 TO
TAL
                     EDM=2.04372e-09    STRATEGY= 1      ERROR MATRIX ACC
URATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  N           2.46469e+02   4.31493e+00   1.19092e-02  -5.38026e-06
   2  mu          1.04793e-02   1.43576e-02   4.87640e-05   4.15093e-03
   3  sigma       1.00316e+00   1.03818e-02   2.86307e-05  -2.55310e-04
```

Some Goodies

```
%jsroot on
```



```
h
Entries      5000
Mean     0.008500
RMS         1.014
```

```
h
bin = 38
x = [0.6250, 0.7500]
entries = 198
```

JSROOT Visualisation

Start On a Laptop

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

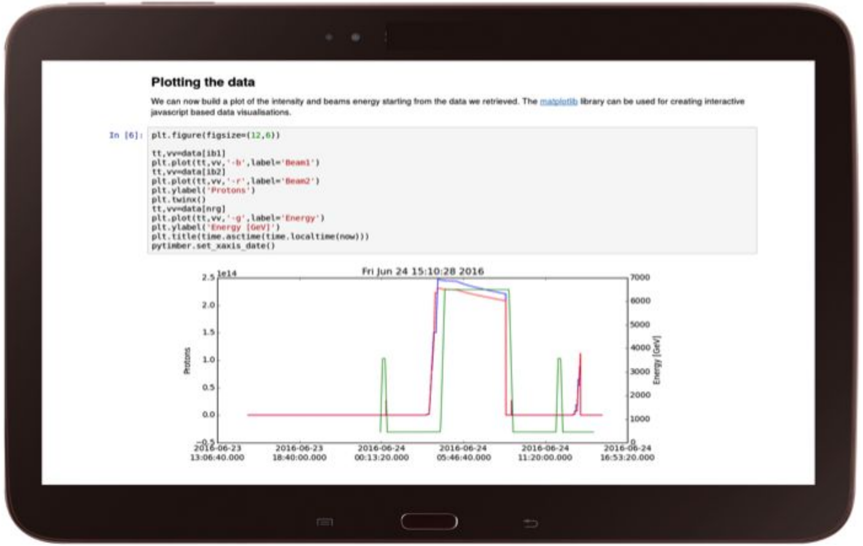
Use Notebooks at CERN

- **SWAN**: **S**ervice for **W**eb based **AN**alysis
- Get a CERNBox (if you don’t have one)
  - Visit https://cernbox.cern.ch

https://swan.cern.ch

The Original Idea: A Turn-Key System

- A ready-to-use system for performing data analysis with **all the software on all the data we need**. **Only requirement: a web browser.**

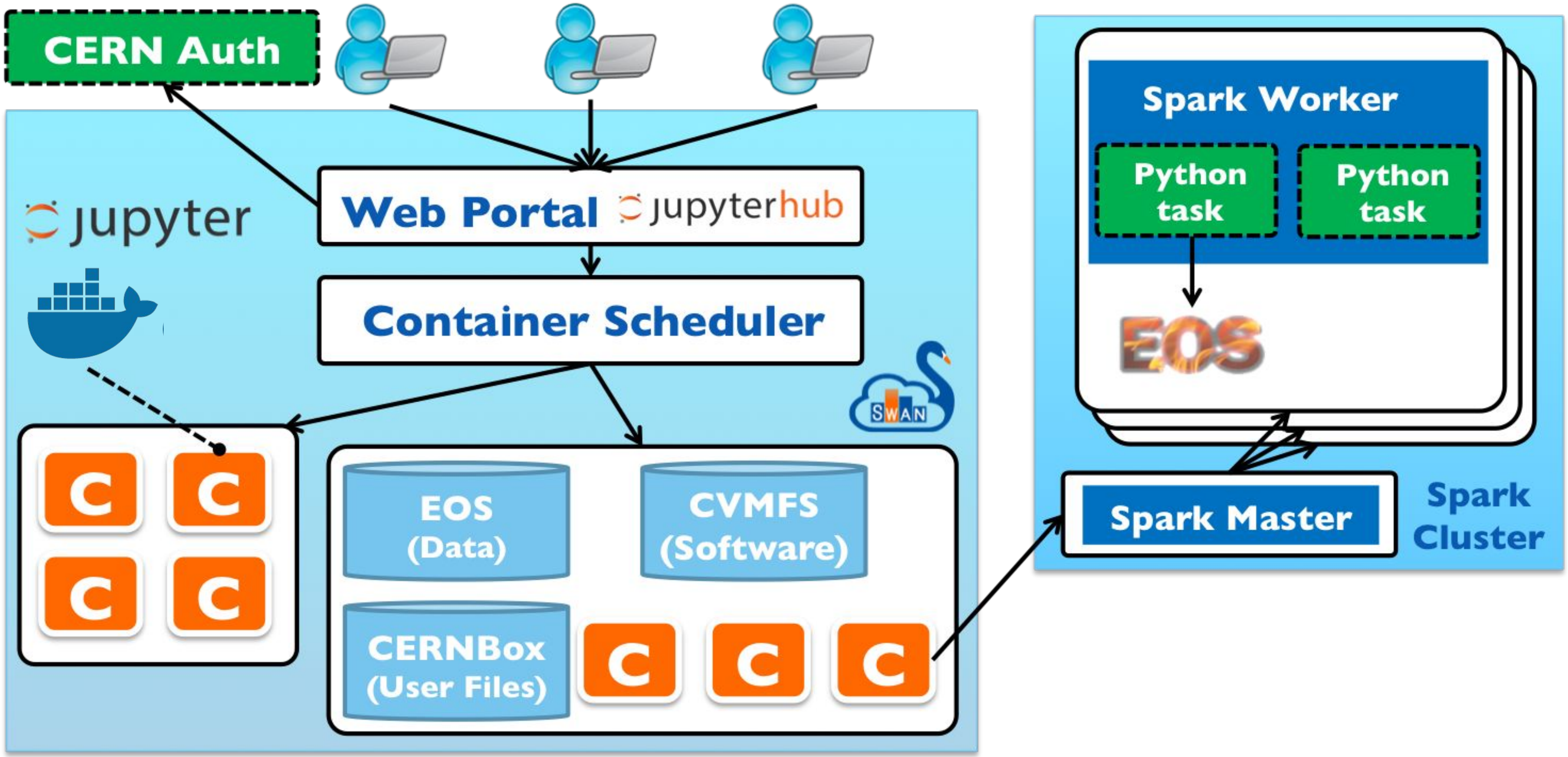


Plotting the data

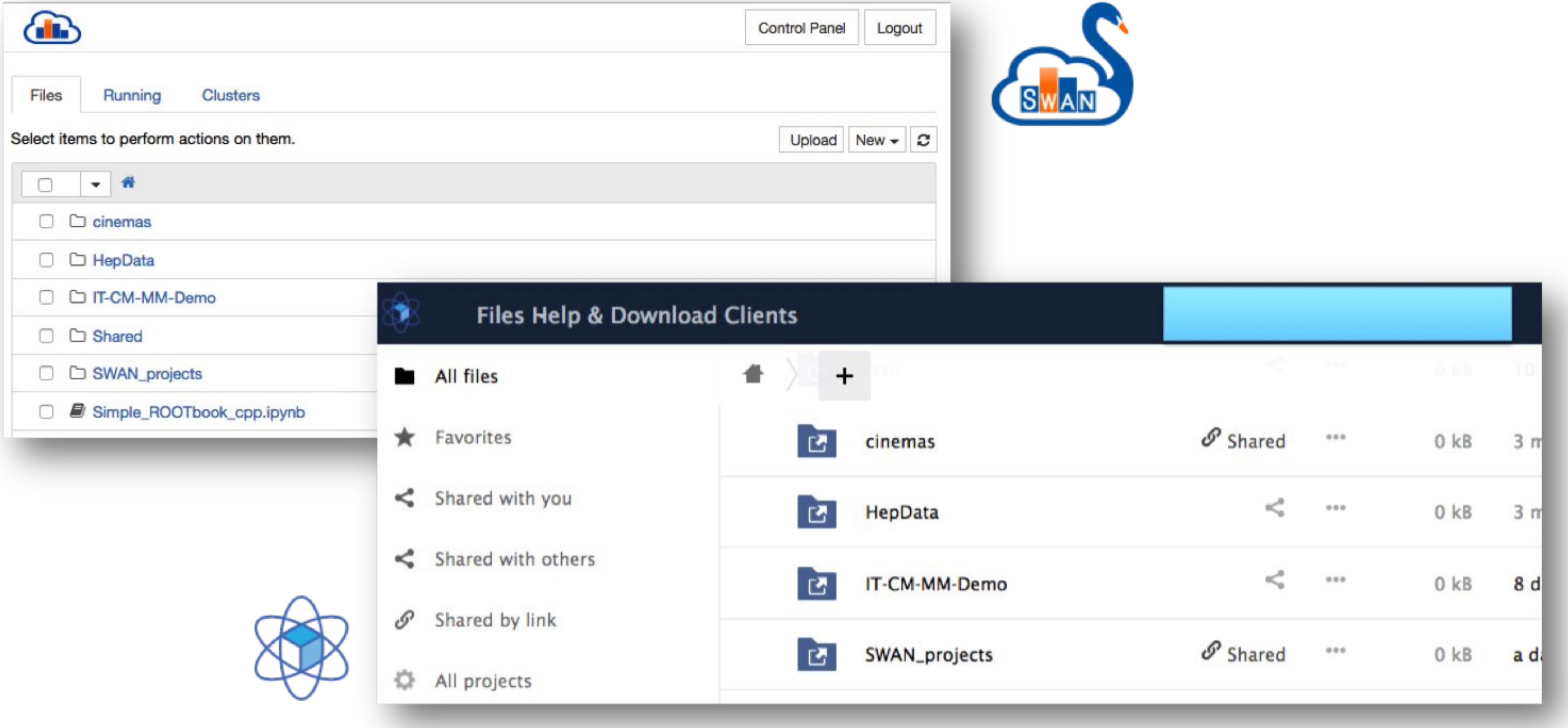
We can now build a plot of the intensity and beams energy starting from the data we retrieved. The matplotlib library can be used for creating interactive javascript based data visualisations.

```
In [6]: plt.figure(figsize=(12,6))

        tt,vv=data[ib1]
        plt.plot(tt,vv,'-b',label='Beam1')
        tt,vv=data[ib2]
        plt.plot(tt,vv,'-r',label='Beam2')
        plt.ylabel('Protons')
        plt.twinx()
        tt,vv=data[nrg]
        plt.plot(tt,vv,'-g',label='Energy')
        plt.ylabel('Energy [GeV]')
        plt.title(time.asctime(time.localtime(now)))
        pytimber.set_xaxis_date()
```

The Service in a Nutshell



CERNBox is Your HOME



Select items to perform actions on them.

- cinemas
- HepData
- IT-CM-MM-Demo
- Shared
- SWAN_projects
- Simple_ROOTbook_cpp.ipynb

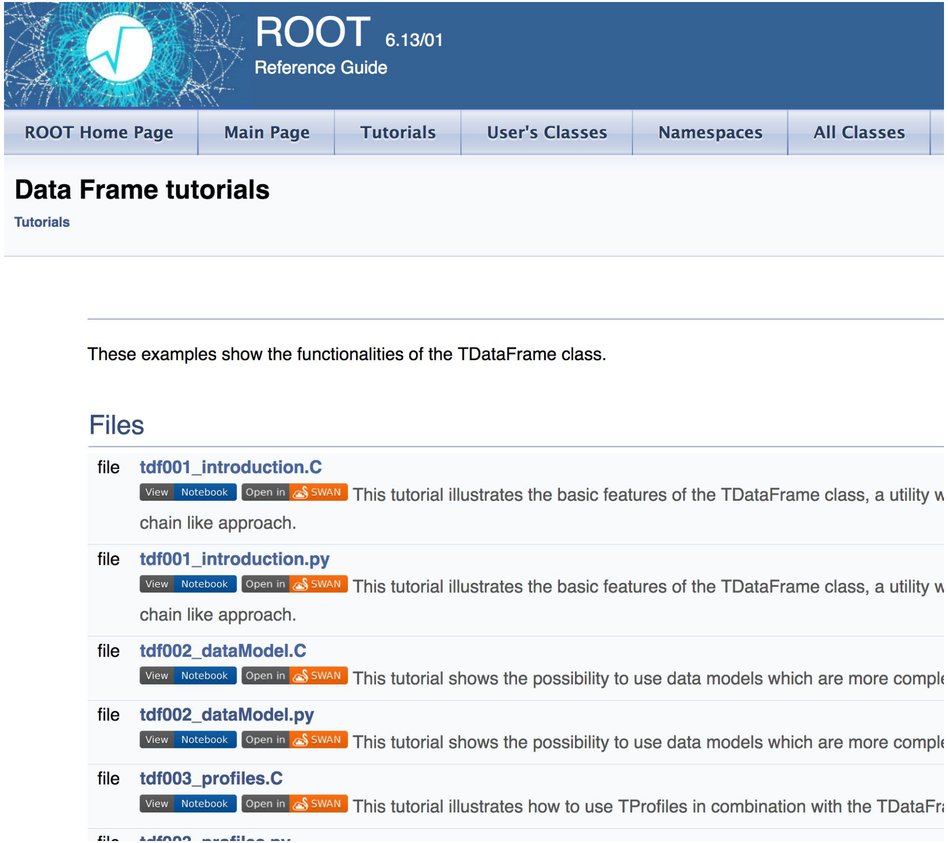
cinemas	Shared	•••	0 kB	3 m
HepData		•••	0 kB	3 m
IT-CM-MM-Demo		•••	0 kB	8 d
SWAN_projects	Shared	•••	0 kB	a d

Interactive Tour of ROOTBooks

Image taken from https://pixabay.com/

More Examples

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



ROOT 6.13/01

Reference Guide

Data Frame tutorials

Tutorials

These examples show the functionalities of the TDataFrame class.

Files

file tdf001_introduction.C

This tutorial illustrates the basic features of the TDataFrame class, a utility w chain like approach.

file tdf001_introduction.py

This tutorial illustrates the basic features of the TDataFrame class, a utility w chain like approach.

file tdf002_dataModel.C

This tutorial shows the possibility to use data models which are more comple

file tdf002_dataModel.py

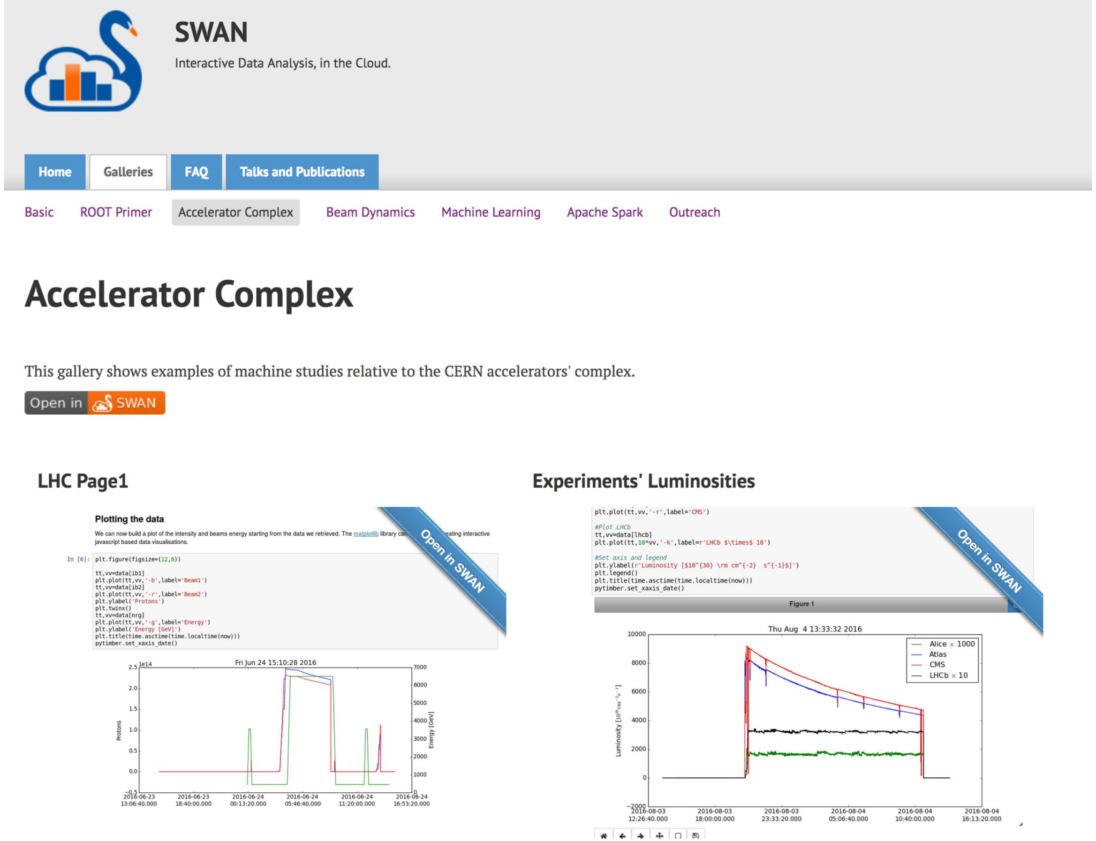
This tutorial shows the possibility to use data models which are more comple

file tdf003_profiles.C

This tutorial illustrates how to use TProfiles in combination with the TDataFr

file tdf003_profiles.py

More Examples



SWAN

Interactive Data Analysis, in the Cloud.

Accelerator Complex

This gallery shows examples of machine studies relative to the CERN accelerators' complex.

LHC Page1

Plotting the data

We can now build a plot of the intensity and beams energy starting from the data we retrieved. The matplotlib library can … eating interactive javascript based data visualisations.

```
In [6]: plt.figure(figsize=(12,6))

        tt,vv=data[ib1]
        plt.plot(tt,vv,'-b',label='Beam1')
        tt,vv=data[ib2]
        plt.plot(tt,vv,'-r',label='Beam2')
        plt.ylabel('Protons')
        plt.twinx()
        tt,vv=data[nrg]
        plt.plot(tt,vv,'-g',label='Energy')
        plt.ylabel('Energy [GeV]')
        plt.title(time.asctime(time.localtime(now)))
        pytimber.set_xaxis_date()
```

Experiments' Luminosities

```
plt.plot(tt,vv,'-r',label='CMS')

#Plot LHCb
tt,vv=data[lhcb]
plt.plot(tt,10*vv,'-k',label=r'LHCb $\times$ 10')

#Set axis and legend
plt.ylabel(r'Luminosity [$10^{30} \rm cm^{-2}  s^{-1}$]')
plt.legend()
plt.title(time.asctime(time.localtime(now)))
pytimber.set_xaxis_date()
```

https://swan.web.cern.ch

Time for Exercises

https://github.com/root-project/training/tree/master/BasicCourse/Exercises/ROOTBooks

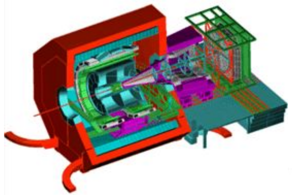
Reading and Writing Data

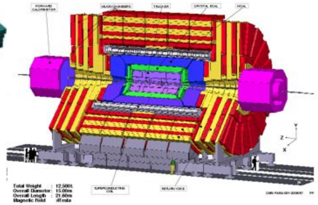
Learning Objectives

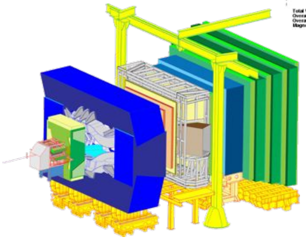
- Understand the relevance of I/O in scientific applications
- Grasp some of the details of the ROOT I/O internals
- Be able to write and read ROOT objects to and from ROOT files

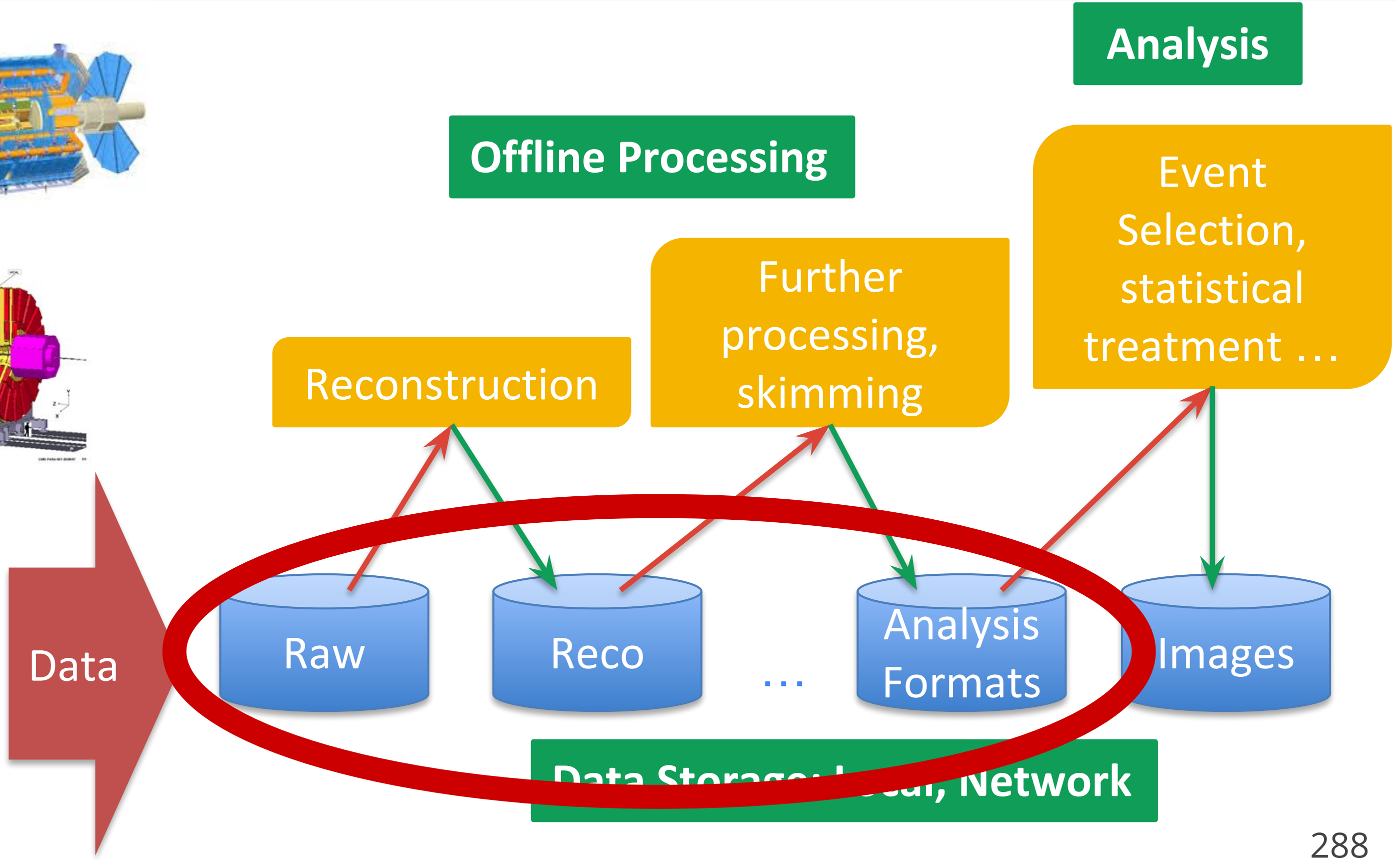
I/O at LHC: an Example

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

More Data in The Future?

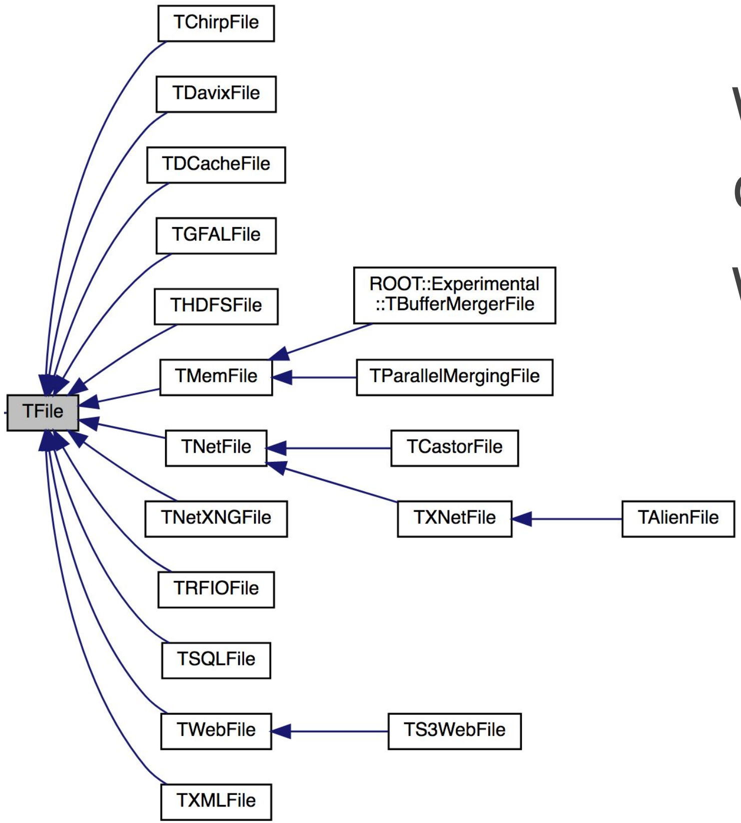
Now	1 EB of data, 0.5 million cores
Run III	LHCb 40x collisions, Alice readout @ 50 KHz (starts in 2021 already!!)
HL-LHC	Atlas/CMS pile-up 60 -> 200, recording 10x evts

The ROOT File

- In ROOT, objects are written in files*
- ROOT provides its file class: the **TFile**
- TFiles are *binary* and have: a *header*, *records* and can be compressed (transparently for the user)
- TFiles have a logical “file system like” structure
  - e.g. directory hierarchy
- **TFiles are self-descriptive**:
  - Can be read without the code of the objects streamed into them
  - E.g. can be read from JavaScript

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

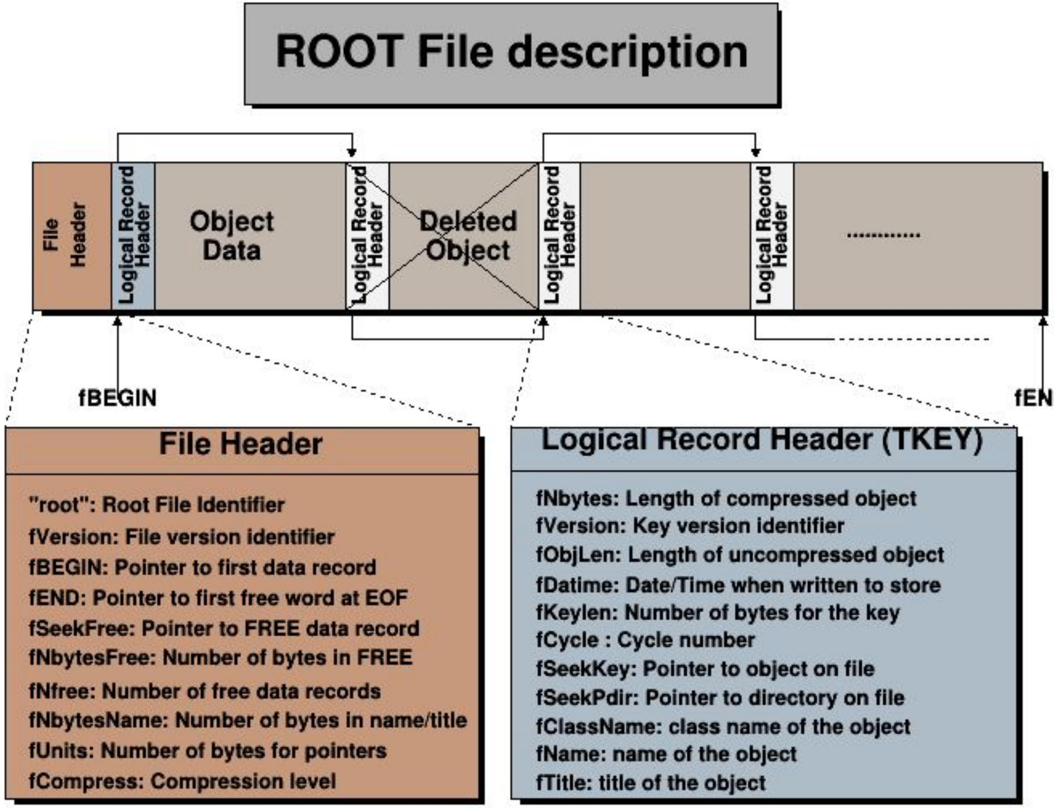
Flavour of TFiles



We'll focus on TFile only in this course.

We'll use TXMLFiles for the exercises.

ROOT File Description



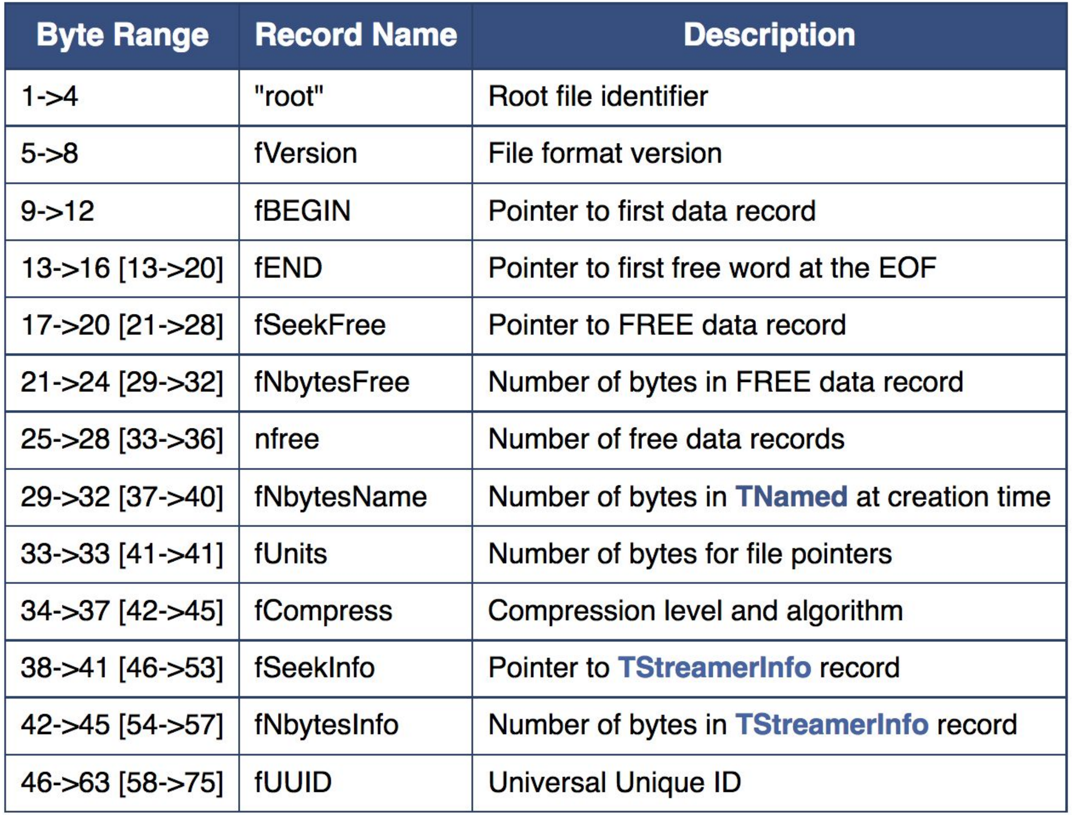
File Header

"root": Root File Identifier
fVersion: File version identifier
fBEGIN: Pointer to first data record
fEND: Pointer to first free word at EOF
fSeekFree: Pointer to FREE data record
fNbytesFree: Number of bytes in FREE
fNfree: Number of free data records
fNbytesName: Number of bytes in name/title
fUnits: Number of bytes for pointers
fCompress: Compression level

Logical Record Header (TKEY)

fNbytes: Length of compressed object
fVersion: Key version identifier
fObjLen: Length of uncompressed object
fDatime: Date/Time when written to store
fKeylen: Number of bytes for the key
fCycle : Cycle number
fSeekKey: Pointer to object on file
fSeekPdir: Pointer to directory on file
fClassName: class name of the object
fName: name of the object
fTitle: title of the object

A Well Documented File Format



Byte Range	Record Name	Description
1->4	"root"	Root file identifier
5->8	fVersion	File format version
9->12	fBEGIN	Pointer to first data record
13->16 [13->20]	fEND	Pointer to first free word at the EOF
17->20 [21->28]	fSeekFree	Pointer to FREE data record
21->24 [29->32]	fNbytesFree	Number of bytes in FREE data record
25->28 [33->36]	nfree	Number of free data records
29->32 [37->40]	fNbytesName	Number of bytes in TNamed at creation time
33->33 [41->41]	fUnits	Number of bytes for file pointers
34->37 [42->45]	fCompress	Compression level and algorithm
38->41 [46->53]	fSeekInfo	Pointer to TStreamerInfo record
42->45 [54->57]	fNbytesInfo	Number of bytes in TStreamerInfo record
46->63 [58->75]	fUUID	Universal Unique ID

How Does it Work in a Nutshell?

- **C++ does not support native I/O** of its objects
- Key ingredient: reflection information - **Provided by ROOT**
  - What are the data members of the class of which this object is instance? I.e. How does the object look in memory?
- The steps, from memory to disk:

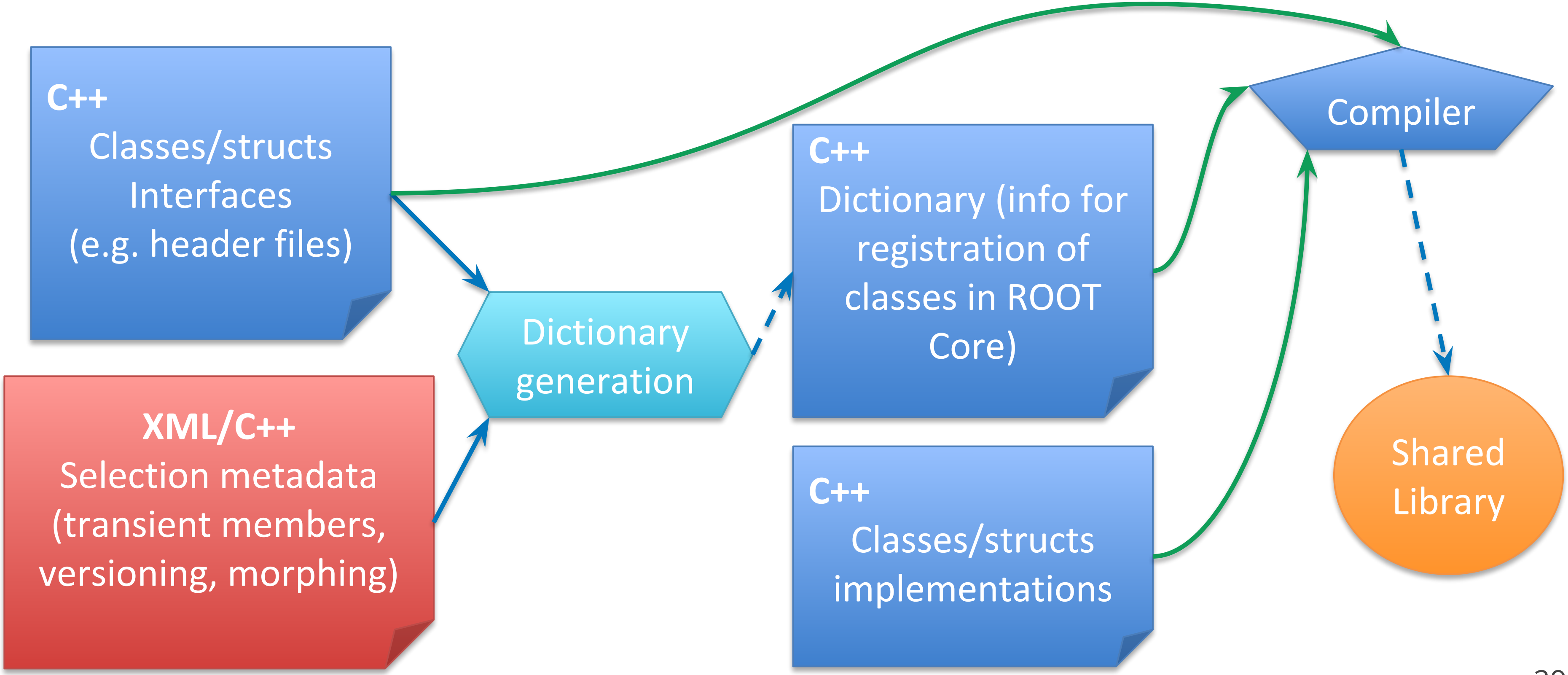
1. Serialisation: from an object in memory to a blob of bytes
2. Compression: use an algorithm to reduce size of the blob (e.g. zip, lzma, lz4)
3. “Real” writing via OS primitives

Serialisation: not a trivial task

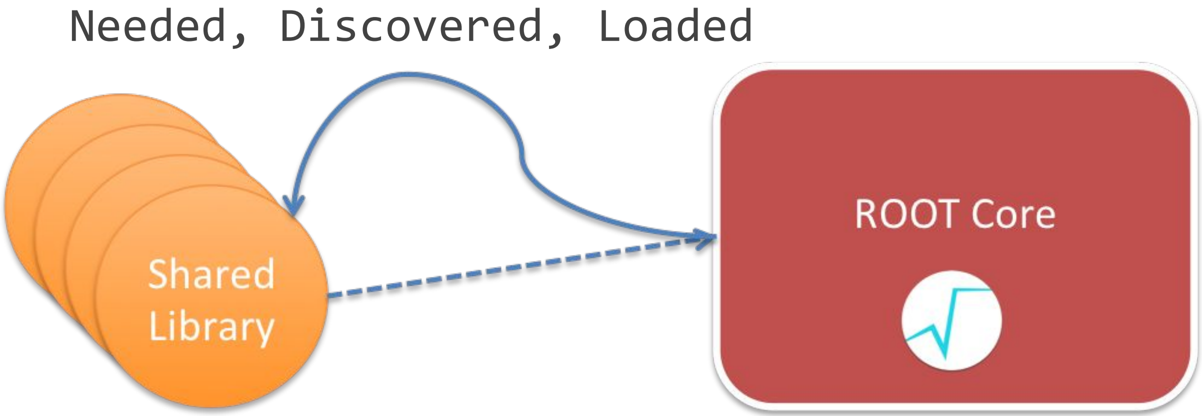
For example:

- Must be platform independent: e.g. 32bits, 64bits
  - Remove padding if present, little endian/big endian
- Must follow pointers correctly
  - And avoid loops ;)
- Must treat stl constructs
- Must take into account customisations by the user
  - E.g. skip “transient data members”

Persistency



Injection of Reflection Information



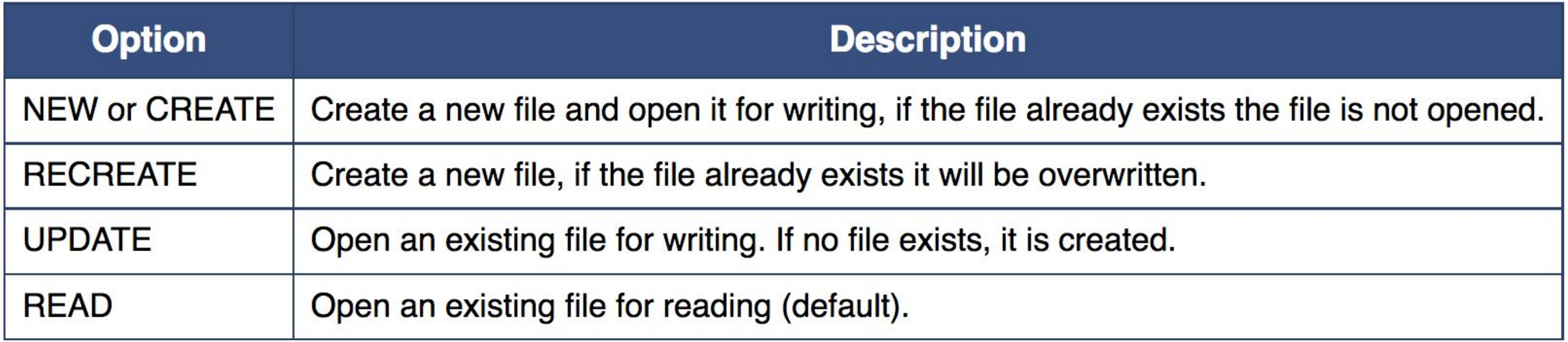
Now ROOT “knows” how to serialise the instances implemented in the library (series of data members, type, transiency) and to write them on disk in row or column format.

TFile in Action

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

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

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

- Write on a file
- Close the file and make sure the operation is finalised

The *gDirectory*

Wait! How does it know where to write?

- ROOT has global variables. Upon creation of a file, the “present directory” is moved to the file.
- Histograms are attached to that directory
- Has up- and down- sides
- Will be more explicit in the future versions of ROOT

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

More than One File

Wait! And then how do I manage more than one file?

- You can “cd” into files anytime.
- The value of the *gDirectory* will change accordingly

```
TFile f1("myfile1.root", "RECREATE");
TFile f2("myfile2.root", "UPDATE");
f1.cd(); TH1F h1("h", "h", 64, 0, 8);
h1.Write();
f2.cd(); TH1F h2("h", "h", 64, 0, 8);
h1.Write();
f1.Close(); f2.Close();
```

TFile in Action

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

TFile in Action: Python

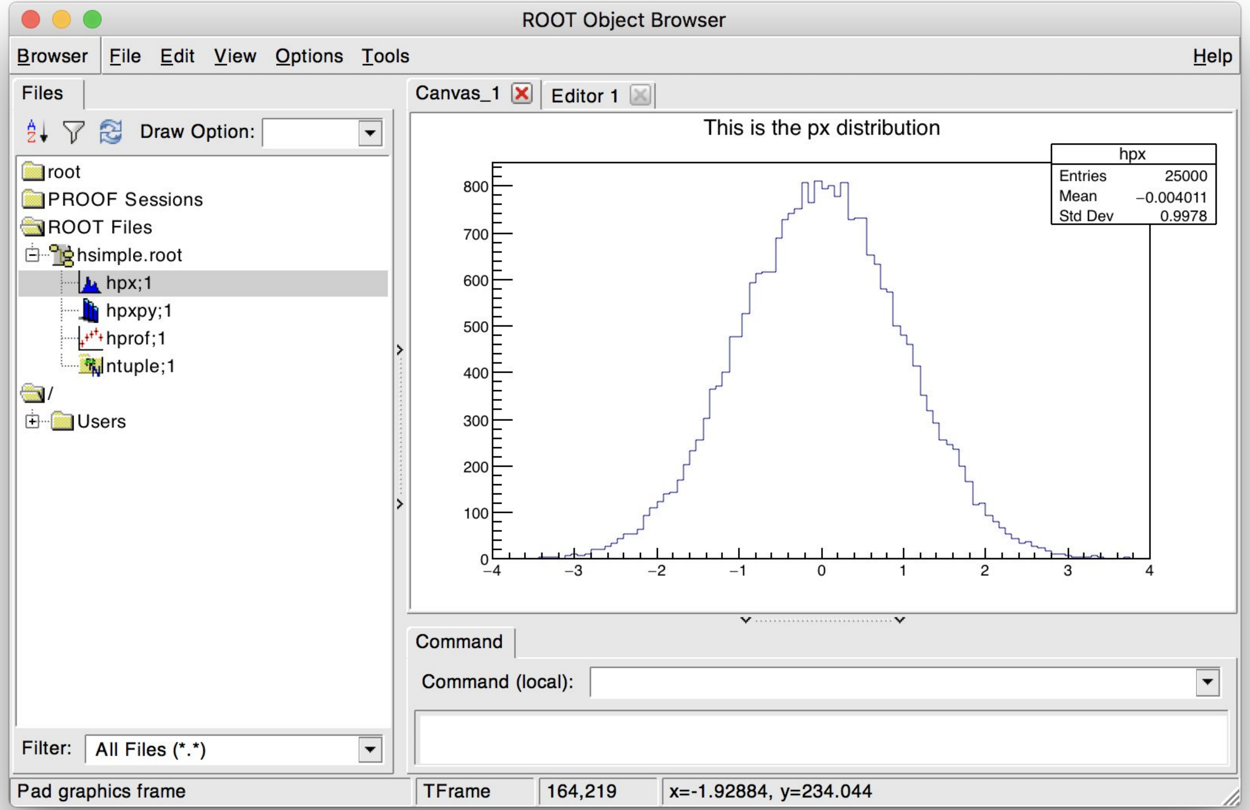
```python
import ROOT
f = ROOT.TFile("myfile.root")
f.h.Draw()
```

Get the histogram by name! Possible because
Python is not a compiled language

Listing TFile Content

- *TFile::ls()*: prints to screen the content of the TFile
  - Great for interactive usage
- *TBrowser* interactive tool
- Loop on the “*TKeys*”, more sophisticated
  - Useful to use “programmatically”
- *rootls* commandline tool: list what is inside

TBrowser


ROOT Object Browser
Browser
File
Edit
View
Options
Tools
Help
Files
Draw Option:
root
PROOF Sessions
ROOT Files
hsimple.root
hpx;1
hpxpy;1
hprof;1
ntuple;1
/
Users
Filter:
All Files (*.*)
Canvas_1
Editor 1
This is the px distribution
hpx
Entries 25000
Mean −0.004011
Std Dev 0.9978
800
700
600
500
400
300
200
100
0
−4
−3
−2
−1
0
1
2
3
4
Command
Command (local):
Pad graphics frame
TFrame
164,219
x=-1.92884, y=234.044

Loop on TKeys

```
TFile f("myfile1.root");
for (auto k : *f.GetListOfKeys()) {
   std::cout << k->GetName() << std::endl;
}
```

Loop on TKeys: Python

List comprehension syntax!

```
import ROOT
f = ROOT.TFile("myfile.root")
keyNames = [k.GetName() for k in f.GetListOfKeys()]
print keyNames
```

Time For Exercises

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

The ROOT Columnar Format

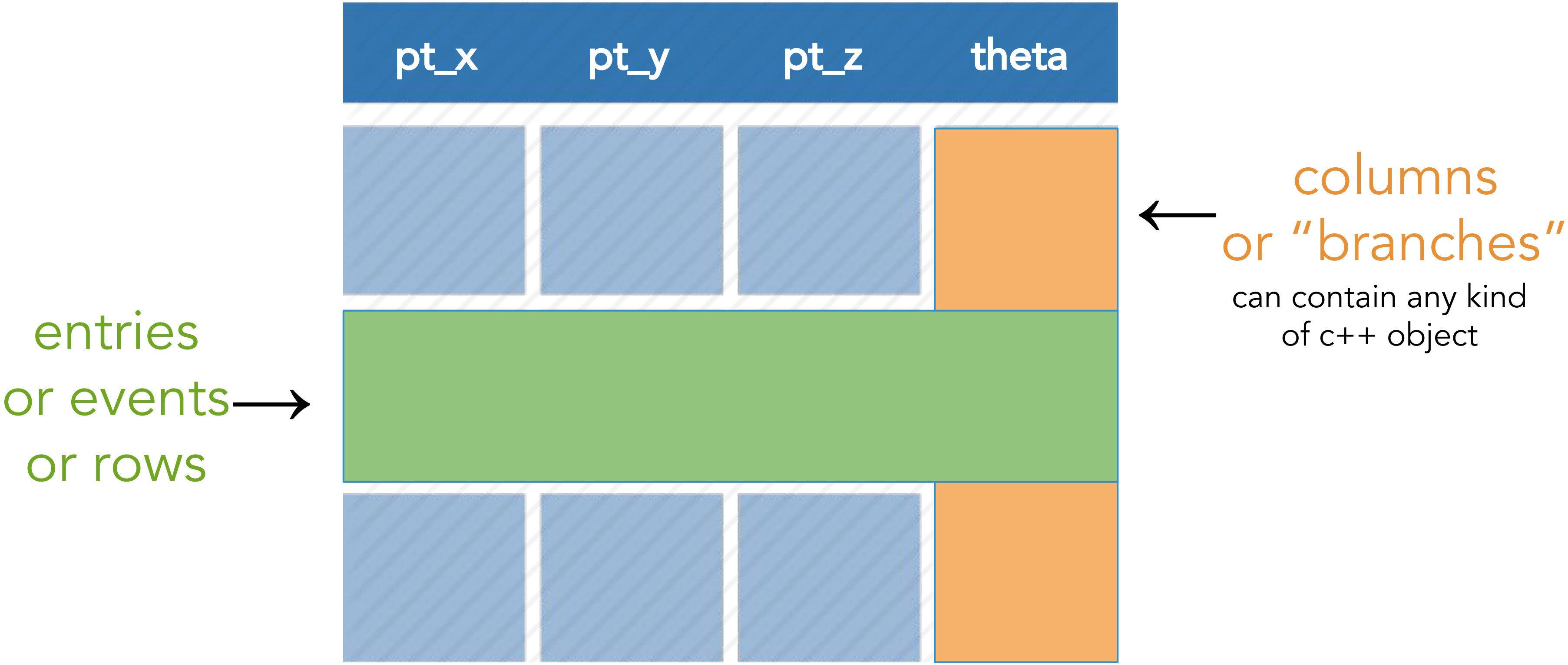
Learning Objectives

- Understand the difference between row-wise and column-wise storage
- Appreciate the difference between low and high-level interfaces for data analysis in ROOT
- Be able to write and read data in 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 on a file? No. Very inefficient!
- Organise the dataset in **columns**

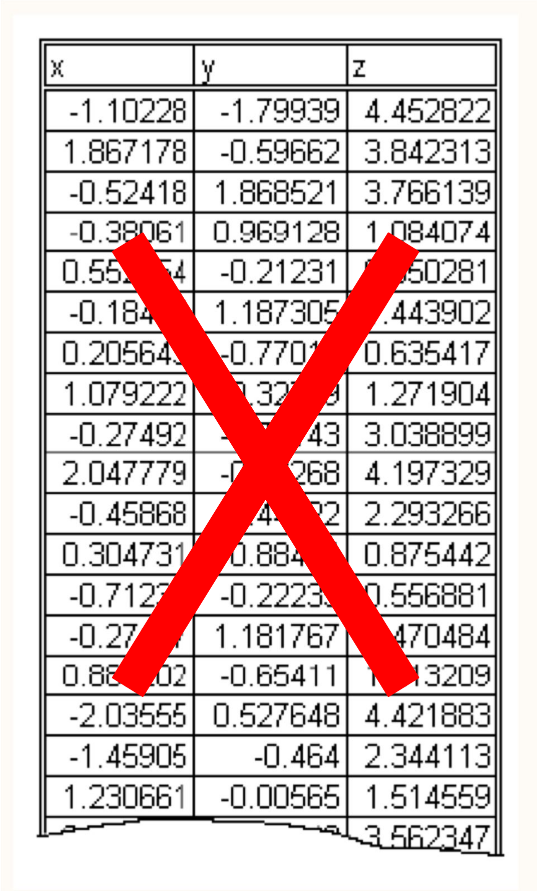
Columnar Representation



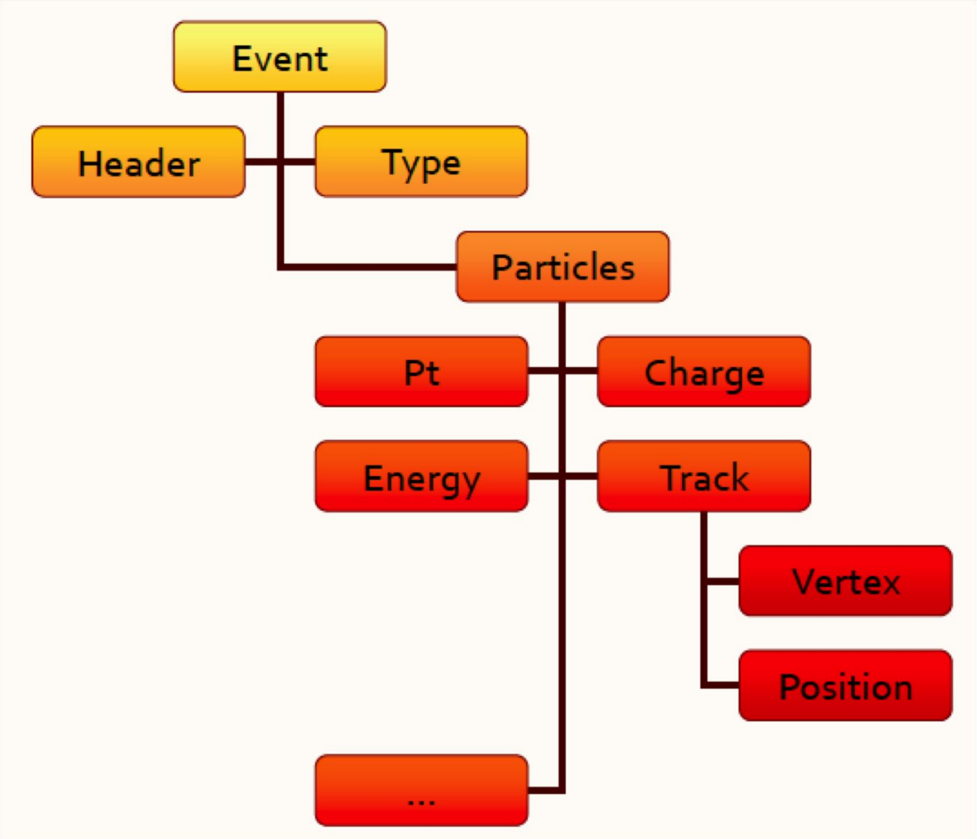
columns
or "branches"
can contain any kind
of c++ object

entries
or events
or rows

Relations Among Columns



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
[illegible]	-0.21231	[illegible]
[illegible]	1.187305	[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]	0.556881
[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



Optimal Runtime and Storage Usage

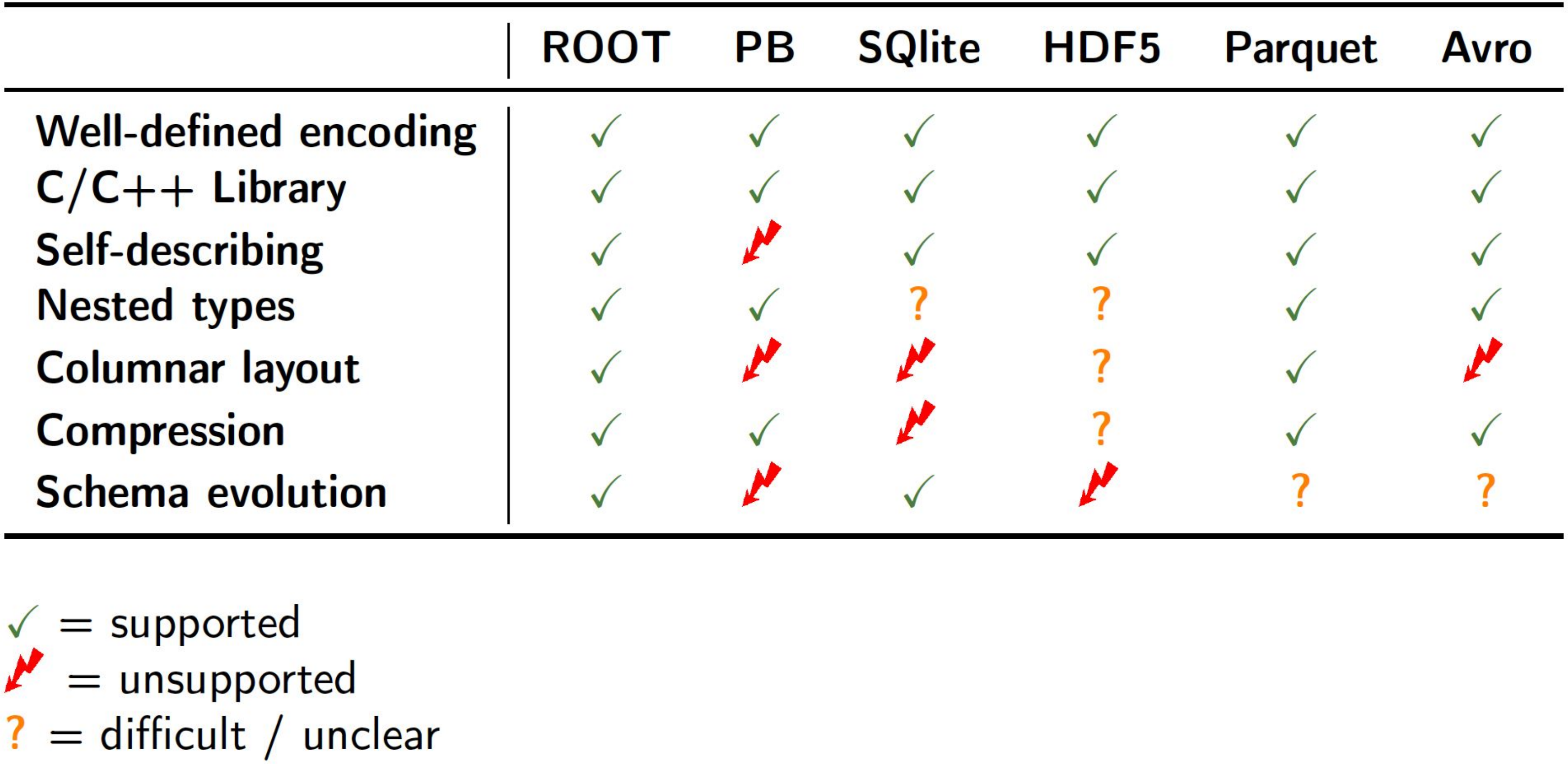
Runtime:

- Can decide what columns to read
- Prefetching, read-ahead optimisations possible

Storage Usage:

- Run-length Encoding (RLE). Compression of individual columns values is very efficient
  - Physics values: potentially all “similar”, e.g. within a few orders of magnitude - position, momentum, charge, index

Comparison With Other I/O Systems



	ROOT	PB	SQlite	HDF5	Parquet	Avro
Well-defined encoding	✓	✓	✓	✓	✓	✓
C/C++ Library	✓	✓	✓	✓	✓	✓
Self-describing	✓	⚡	✓	✓	✓	✓
Nested types	✓	✓	?	?	✓	✓
Columnar layout	✓	⚡	⚡	?	✓	⚡
Compression	✓	✓	⚡	?	✓	✓
Schema evolution	✓	⚡	✓	⚡	?	?

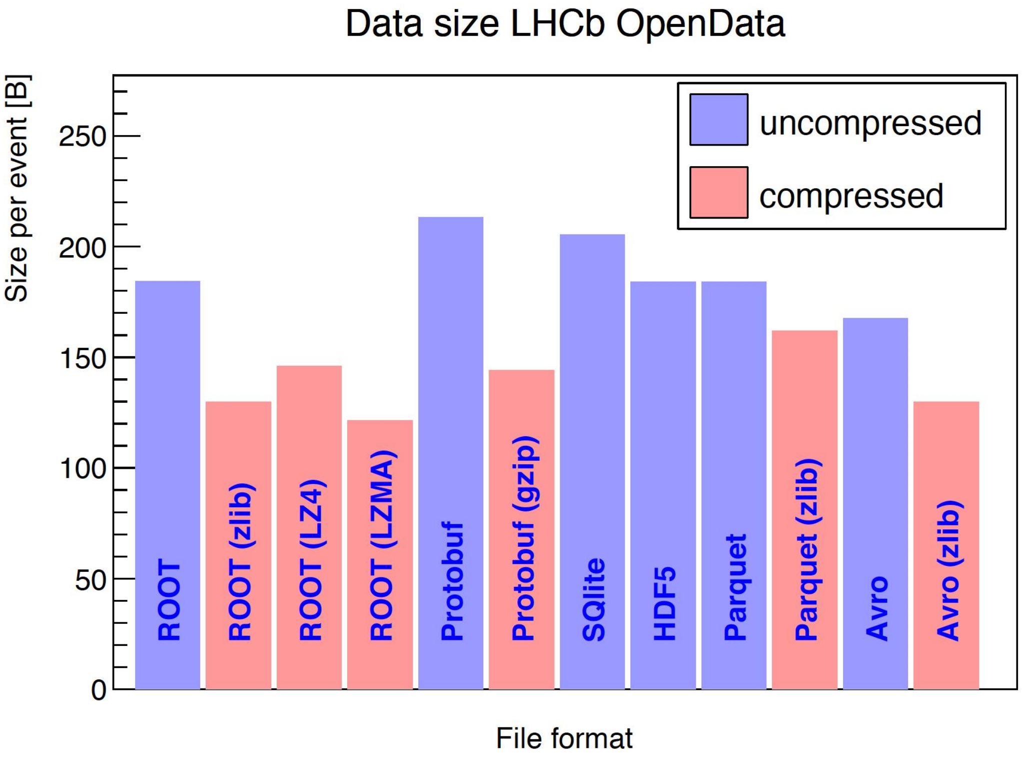
✓ = supported

⚡ = unsupported

? = difficult / unclear

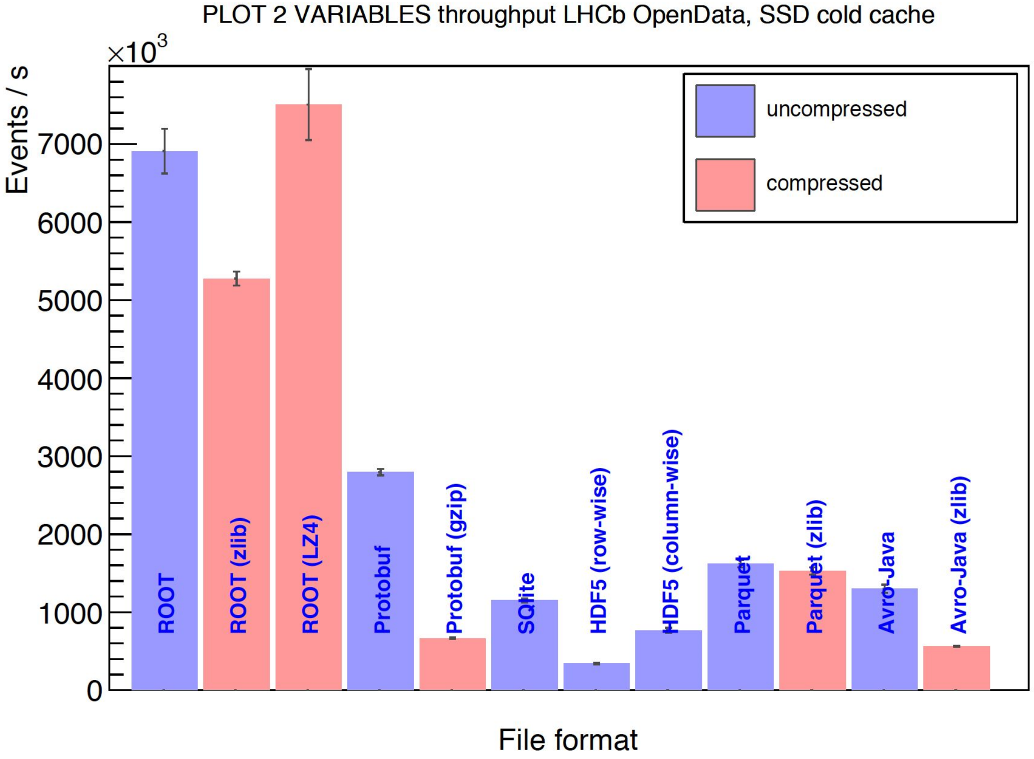
J. Blomer, A quantitative review of data formats for HEP analyses ACAT 2017

Comparison With Other I/O Systems



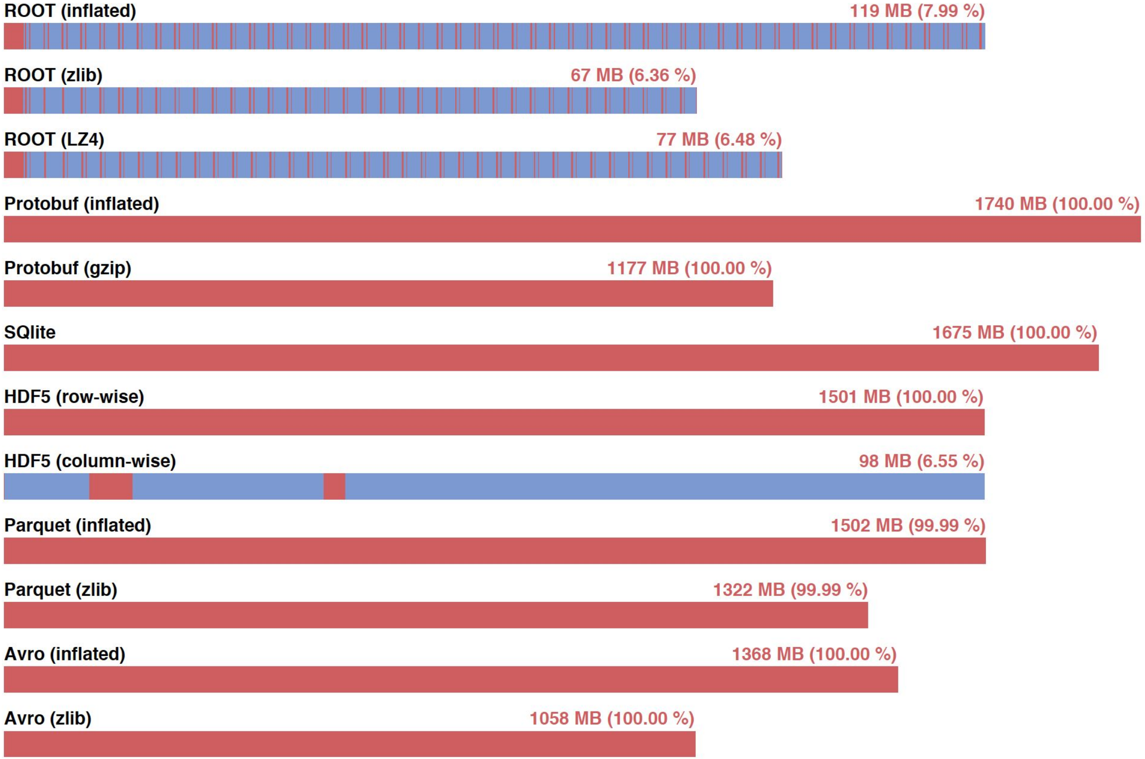
J. Blomer, A quantitative review of data formats for HEP analyses ACAT 2017

Comparison With Other I/O Systems



J. Blomer, A quantitative review of data formats for HEP analyses ACAT 2017

I/O Patterns



The less you read (red sections), the faster

J. Blomer, A quantitative review of data formats for HEP analyses ACAT 2017

The TTree

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

- Also called *tree*, columns also called *branches*
- An object type per column, **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.

Filling a TNtuple

```
TFile f("myfile.root", "RECREATE");
TNtuple myntuple("n", "n", "x:y:z:t");

// We assume to have 4 arrays of values:
// x_val, y_val, z_val, t_val

for (auto ievt: ROOT::TSeqI(128)) {
   myntuple.Fill(x_val[ievt], y_val[ievt],
                 z_val[ievt], t_val[ievt]);
}
myntuple.Write();
f.Close();
```

Names of the columns

Works only if columns are simple numbers

Reading a TNtuple

```
TFile f("hsimple.root");
TNtuple *myntuple;
f.GetObject("hsimple", myntuple);
TH1F h("h", "h", 64, -10, 10);
for (auto ievt: ROOT::TSeqI(myntuple->GetEntries()) {
   myntuple->GetEntry(ievt);
   auto xyzt = myntuple->GetArgs(); // Get a row
   if (xyzt[2] > 0) h.Fill(xyzt[0] * xyzt[1]);
}
h.Draw();
```

Works only if columns are simple numbers

Reading a TNtuple: PyROOT

```python
import ROOT
f = ROOT.TFile("hsimple.root")
h = ROOT.TH1F("h", "h", 64, -10, 10)
for evt in f.hsimple:
   if evt.pz > 0: h.Fill(evt.py * evt.pz)
h.Draw()
```

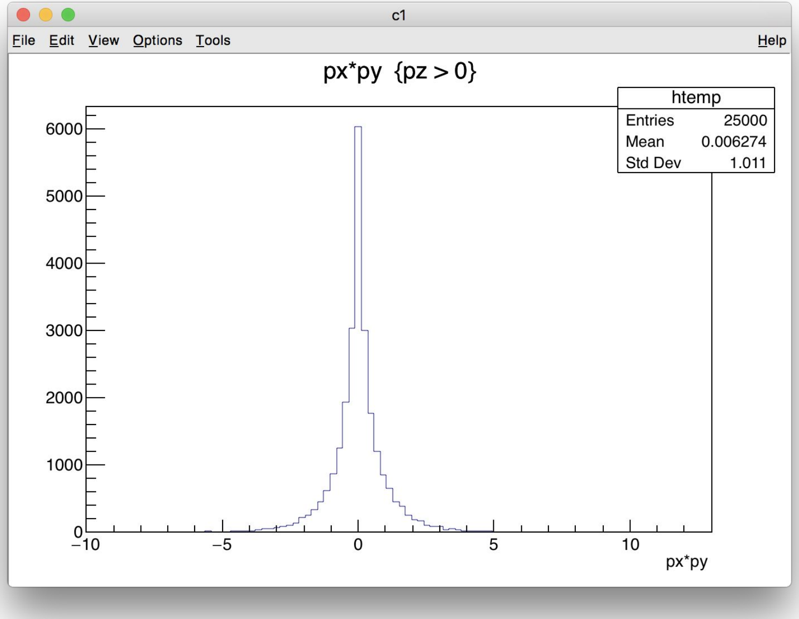
PyROOT: Works for all types of columns, not only numbers!

One Line Analysis

- It is possible to produce simple plots described as strings
- **TNtuple::Draw()** method
- Jargon: known also under the name of *ttreedraw*
- Good for quick looks, does not scale
  - E.g. one loop on all events per plot

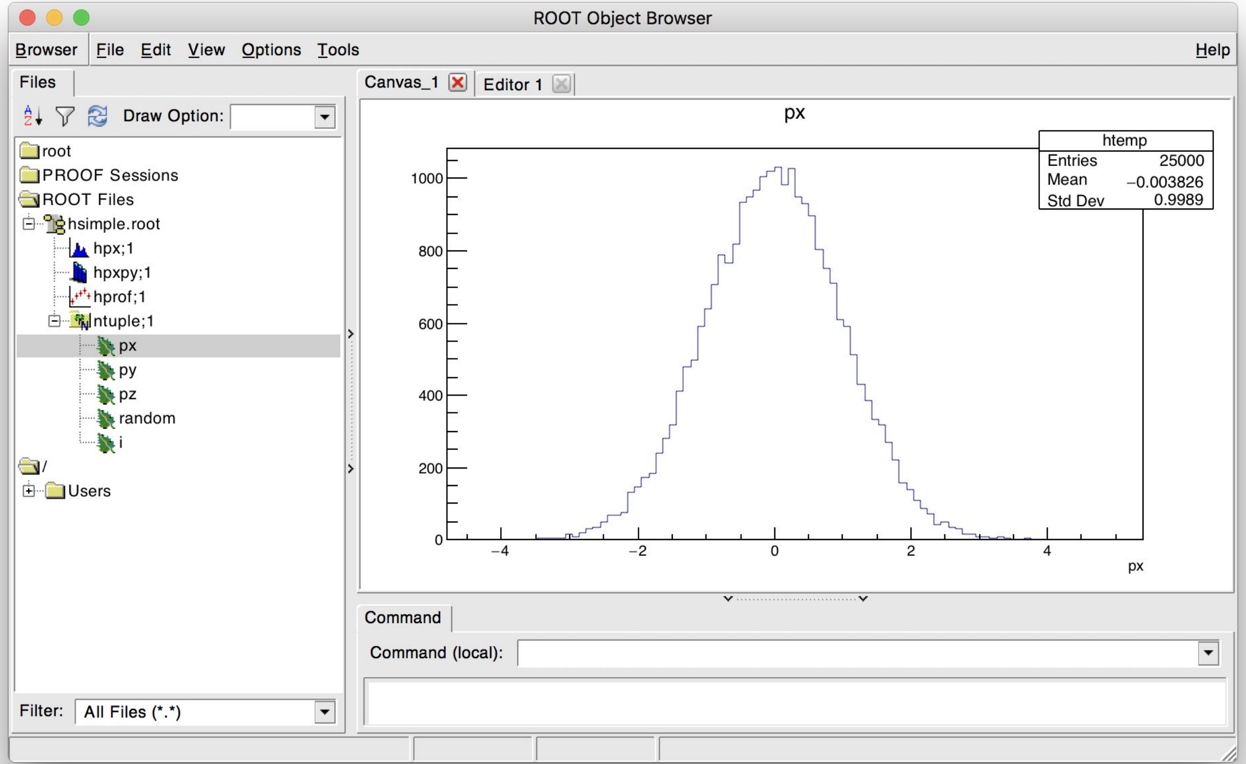
One Line Analysis

Works for all types of columns, not only numbers!



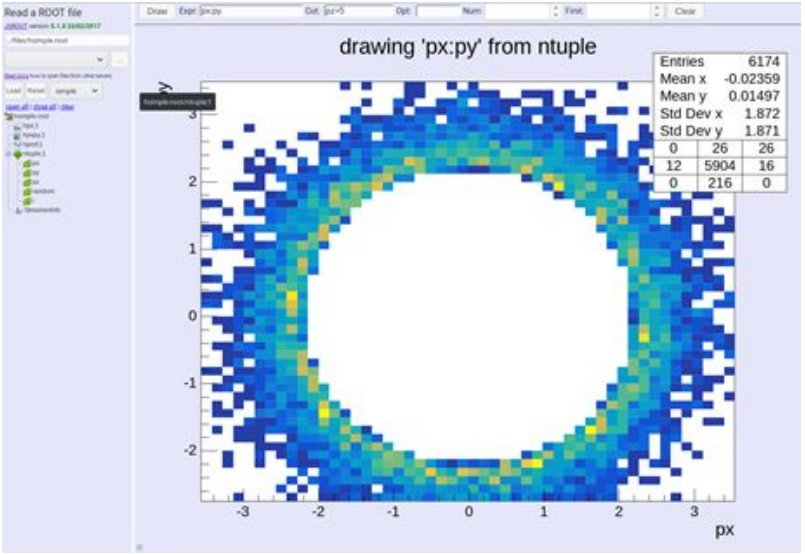
```
TFile f("hsimple.root");
TNtuple *myntuple;
f.GetObject("ntuple", myntuple);
myntuple.Draw("px * py", "pz > 0");
```

Reading a TNtuple with TBrowser



In Javascript

- Direct access to the data from the browser
- All branches types are supported
  - including split STL containers
  - and old TBranchObject
- Complex TTree::Draw syntax
  - Expressions
  - Cut conditions
  - Arrays indexes
  - Math functions
  - Class functions
  - Histogram parameters



Entries	6174	
Mean x	-0.02359	
Mean y	0.01497	
Std Dev x	1.872	
Std Dev y	1.871	
0	26	26
12	5904	16
0	216	0

https://root.cern.ch/js/latest/

Time For Exercises

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

TTrees and TNtuples

- **TNtuple** is great, but only works if columns hold simple numbers
- If something else needs to be in the columns, **TTree** must be used
- **TNtuple** is a specialisation of **TTree**

We’ll explore how to read TTrees starting from the TNtuple examples

Filling a Tree with Numbers

```
TFile f("SimpleTree.root","RECREATE"); // Create file first. The TTree will be associated to it
TTree data("tree","Example TTree");    // No need to specify column names

double x, y, z, t;
data.Branch("x",&x,"x/D");      // Associate variable *pointer* to column and specify its type, double
data.Branch("y",&y,"y/D");
data.Branch("z",&z,"z/D");
data.Branch("t",&t,"t/D");

for (int i = 0; i<128; ++i) {
   x = gRandom->Uniform(-10,10);
   y = gRandom->Gaus(0,5);
   z = gRandom->Exp(10);
   t = gRandom->Landau(0,2);
   data.Fill();                 // Make sure the values of the variables are recorded
}
data.Write();                   // Dump on the file
f.Close();
```

Filling a Tree with Objects

```
TRandom3 R;
using trivial4Vectors =
std::vector<std::vector<double>>;

TFile f("vectorCollection.root",
        "RECREATE");
TTree t("t","Tree with pseudo particles");

trivial4Vectors  parts;
auto partsPtr = &parts;

t1.Branch("tracks", &partsPtr);
// pi+/pi- mass
constexpr double M = 0.13957;
```

```
for (int i = 0; i < 128; ++i) {
   auto nPart = R.Poisson(20);
   particles.clear(); parts.reserve(nPart);
   for (int j = 0; j < nPart; ++j) {
      auto pt = R.Exp(10);
      auto eta = R.Uniform(-3,3);
      auto phi = R.Uniform(0, 2*TMath::Pi() );
      parts.emplace_back({pt, eta, phi, M});
    }
    t.Fill();
  }
  t.Write();
}
```

Reading Objects from a TTree

```
{

using trivial4Vector =
std::vector<double>;
using trivial4Vectors =
std::vector<trivial4Vector>;

TFile f("parts.root");
TTreeReader myReader("t", &f);
TTreeReaderValue<trivial4Vectors>
partsRV(myReader, "parts");

TH1F h("pt","Particles Transverse
Momentum;P_{T} [GeV];#", 64, 0, 10);
```

```
while (myReader.Next()) {
   for (auto &p : *partsRV ) {
     auto pt = p[0];
     h.Fill(pt);
   }
}
h.Draw();

}
```

TDataFrame Basics

Can we do Better?

simple yet powerful way to analyse data with modern C++

provide high-level features, e.g.
less typing, better expressivity, abstraction of complex operations

allow transparent optimisations, e.g.
multi-thread parallelisation and caching

Improved Interfaces

what we write

```
TTreeReader reader(data);
TTreeReaderValue<A> x(reader,"x");
TTreeReaderValue<B> y(reader,"y");
TTreeReaderValue<C> z(reader,"z");
while (reader.Next()) {
   if (IsGoodEntry(*x, *y, *z))
      h->Fill(*x);
}
```

what we *mean*

- full control over the event loop
- requires some boilerplate
- users implement common tasks again and again
- parallelisation is not trivial

TDataFrame: declarative analyses

```
TDataFrame d(data);
auto h = d.Filter(IsGoodEntry, {"x","y","z"})
          .Histo1D("x");
```

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

? parallelization is not trivial?

TDataFrame: declarative analyses

```
ROOT::EnableImplicitMT();
TDataFrame d(data);
auto h = d.Filter(IsGoodEntry, {"x","y","z"})
          .Histo1D("x");
```

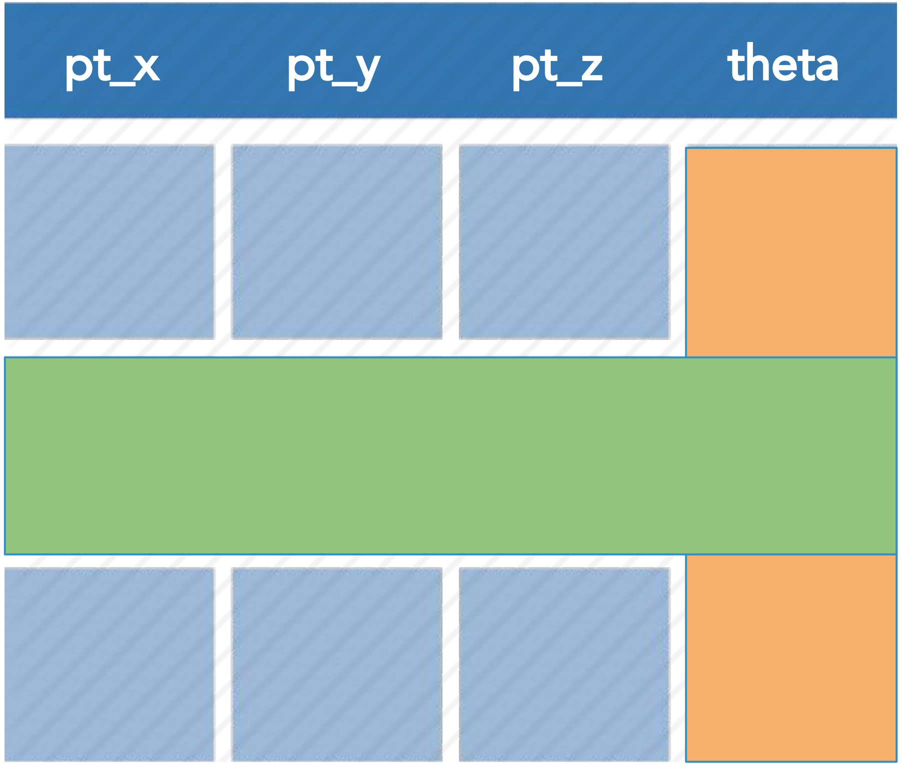
● full control over *the analysis*

● no boilerplate

● common tasks are already implemented

? parallelization is not trivial?

Columnar Representation



pt_x	pt_y	pt_z	theta

columns
or “branches”

can contain any kind
of c++ object

entries
or events
or rows

TDataFrame: 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)

Creating a TDataFrame - 1 file

```
TDataFrame d1("treename", "file.root");

auto filePtr = TFile::Open("file.root");
TDataFrame d2("treename", filePtr);

TTree *treePtr = nullptr;
filePtr->GetObject("treename", treePtr);
TDataFrame d3(*treePtr); // by reference!
```

Three ways to create a TDataFrame that reads tree “treename” from file “file.root”

Creating a TDataFrame - more files

```
TDataFrame d1("treename", "file*.root");
TDataFrame d2("treename", {"file1.root","file2.root"});

std::vector<std::string> files = {"file1.root","file2.root"};
TDataFrame d3("treename", files);

TChain chain("treename");
chain.Add("file1.root); chain.Add("file2.root);
TDataFrame d4(chain); // passed by reference, not pointer!
```

Here TDataFrame reads tree “treename” from files “file1.root” and “file2.root”

Cut on theta, fill histogram with pt

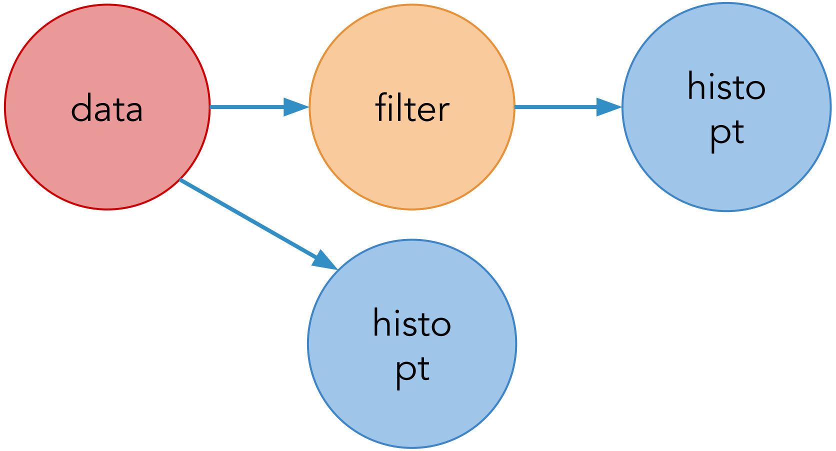
```
TDataFrame d("t", "f.root");
auto h = d.Filter("theta > 0").Histo1D("pt");
h->Draw(); // event loop is run here, when you access a result
           // for the first time
```

event-loop is run *lazily*, upon first access to the results

Time For Exercises

- Draw a plot of px + py for every pz between -2 and 2 using the hsimple.root file
  - Use TDataFrame
  - Compare with the other approaches: number of lines, readability

Think of your analysis as data-flow



```
auto h2 = d.Filter("theta > 0").Histo1D("pt");
auto h1 = d.Histo1D("pt");
```

Using callables instead of strings

```cpp
// 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

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 once here
h2->Draw("SAME");    // no need to run loop again here
```

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

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.

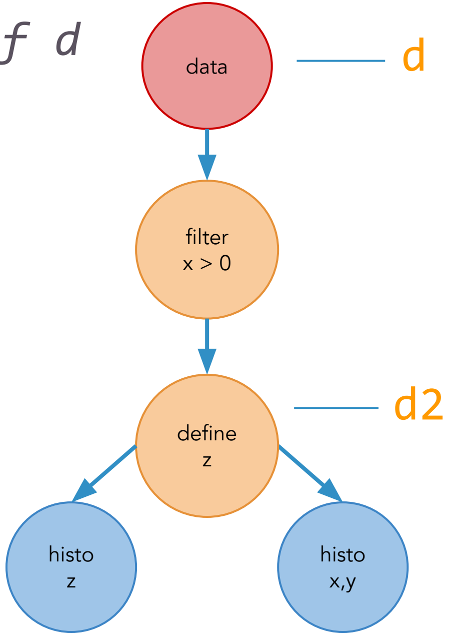
Define a new column

```
double SqrtSumSq(double, double) { return … ; }
double m = d.Filter("x > y")
            .Define("z", SqrtSumSq, {"x","y"})
            .Mean("z");
```

Just like `Filter`, `Define` accepts any callable object (function, lambda, functor class…)

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 hxy = d2.Histo2D("x","y");
```



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

Cutflow reports

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

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

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

Running on a range of entries #1

```
// stop after 100 entries have been processed
auto hz = d.Range(100).Histo1D("x");

// skip the first 10 entries, then process one every two until the end
auto hz = d.Range(10, 0, 2).Histo1D("x");
```

Ranges are only available in single-thread executions.
They are useful for quick initial data explorations.

Running on a range of entries #2

```
// ranges can be concatenated with other transformations
auto c = d.Filter("x > 0")
          .Range(100)
          .Count();
```

This `Range` will process the first 100 entries
that pass the filter

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.

Creating a new data-set

```
TDataFrame d(100);
auto new_d = d.Define("x", []() { return double(rand()) / RAND_MAX; })
              .Define("y", []() { return rand() % 10; })
              .Snapshot("tree", "newfile.root");
```

We create a special TDF with 100 (empty) entries, define some columns, save it to file

N.B. `rand()` is generally not a good way to produce uniformly distributed random numbers

Not Only ROOT Datasets

- TDataSource: Plug *any columnar* format in TDataFrame
- Keep the programming model identical!
- ROOT provides CSV data source
- More to come
  - TDataSource is a programmable interface!
  - E.g. https://github.com/bluehood/mdfds LHCb raw format - not in the ROOT repo

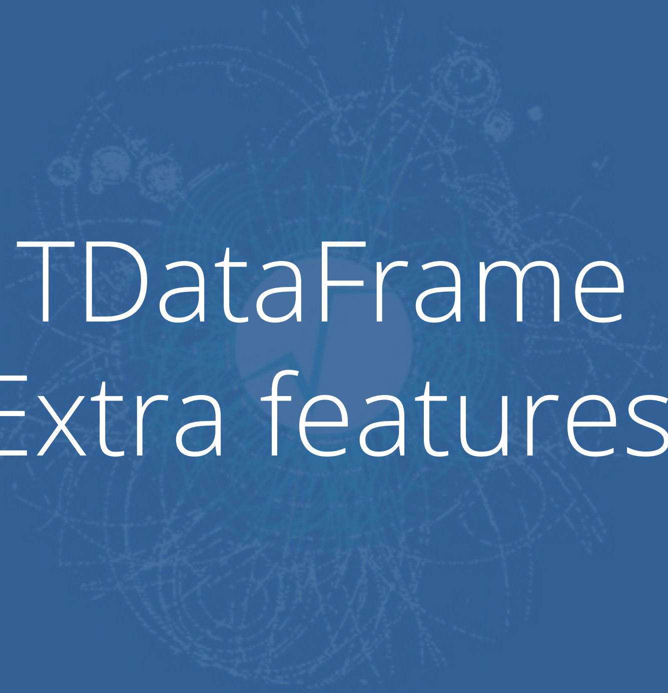
Not Only ROOT Datasets

```
auto fileName = "tdf014_CsvDataSource_MuRun2010B.csv";
auto tdf = ROOT::Experimental::TDF::MakeCsvDataFrame(fileName);

auto filteredEvents =
tdf.Filter("Q1 * Q2 == -1")
.Define("m", "sqrt(pow(E1 + E2, 2) - (pow(px1 + px2, 2) + pow(py1 + py2, 2) + pow(pz1 + pz2, 2)))");

auto invMass =
filteredEvents.Histo1D({"invMass", "CMS Opendata: #mu#mu mass;mass [GeV];Events", 512, 2, 110}, "m");
```

tdf014_CsvDataSource_MuRun2010B.csv:

```
Run,Event,Type1,E1,px1,py1,pz1,pt1,eta1,phi1,Q1,Type2,E2,px2,py2,pz2,pt2,eta2,phi2,Q2,M
146436,90830792,G,19.1712,3.81713,9.04323,-16.4673,9.81583,-1.28942,1.17139,1,T,5.43984,-0.362592,2.62699,-
4.74849,2.65189,-1.34587,1.70796,1,2.73205
146436,90862225,G,12.9435,5.12579,-3.98369,-11.1973,6.4918,-1.31335,-0.660674,-1,G,11.8636,4.78984,-6.26222,
-8.86434,7.88403,-0.966622,-0.917841,1,3.10256
```

TDataFrame
Extra features



Caching

```
TDataFrame d("mytree", "myFile.root");
auto cached_d = d.Cache();
```

All the content of the TDF is now in (contiguous) memory.
Analysis as fast as it can be (vectorisation possible too).

N.B. It is always possible to selectively cache columns to save some memory!

Creating a new data-set - parallel

```
ROOT::EnableImplicitMT();
TDataFrame d(100);
auto new_d = d.Define("x", []() { return double(rand()) / RAND_MAX; })
              .Define("y", []() { return rand() % 10; })
              .Snapshot("tree", "newfile.root");
```

We create a special TDF with 100 (empty) entries, define some columns, save it to file -- in parallel

N.B. `rand()` is generally not a good way to produce uniformly distributed random numbers

More on histograms #1

```
auto h = d.Histo1D("x","w");
```

TDF can produce *weighted* TH1D, TH2D and TH3D.
Just pass the extra column name.

More on histograms #2

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

You can specify a model histogram with a set axis range, a name and a title (optional for TH1D, mandatory for TH2D and TH3D)

Filling histograms with arrays

```
auto h = d.Histo1D("pt_array", "x_array");
```

If `pt_array` and `x_array` are an array or an STL container (e.g. std::vector), TDF fills histograms with all of their elements. `pt_array` and `x_array` are required to have equal size for each event.

C++ / JIT / PyROOT

Pure C++

```
d.Filter([](double t) { return t > 0.; }, {"th”})
 .Snapshot<vector<float>>("t","f.root",{"pt_x"});
```

C++ and JIT-ing with CLING

```
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/BasicCourse/Exercises/WorkingWithColumnarData

Developing Packages against ROOT

Learning Objectives

- What is provided to help you to build your package against ROOT
- Some basic CMake commands
- Example of a user package

Building ROOT: Reminder

- ROOT uses the CMake tool as the (meta) build system
  - CMake has become a very popular build process many large projects have adopted it
  - Also in the HEP community is becoming very popular
- CMake `generates` native build environments
  - UNIX/Linux->Makefiles, Ninja
  - Windows->VisualStudio, nmake
  - Apple->Xcode
- Many nice features
  - Cross-Platform, pretty well documented, etc.
  - Can cope with complex, large build environments
  - …

What is provided by ROOT?

- **`root-config`** command
  - executable command that can be used in shell scripts and makefiles to provide information about compiler and linker flags needed by client projects
- **`ROOTConfig.cmake`** file
  - file used by CMake to obtain some ROOT variables and macros to be used when building the client project

Using root-config

- The user can embed the execution `root-config` to his/her shell or makefile commands
  - See `root-config --help` for the list of available options

- Example to build an executable

```
c++ myprog.cxx -o myprog `root-config --cflags` \
                         `root-config --libs`
```

Basic CMake

- Place a CMakeLists.txt file at the top directory of your project
- Create a build area (`make build; cd build`)
- Configure (`cmake <source>`)
- Build (`make`)
- Install (`make install`)

Most basic project is to build an executable

```
cmake_minimum_required(VERSION 3.4)
project(myproject)

add_executable(myprog myprog.cxx)
```

Making a Library with CMake

- Adding a library is also very simple
  - Let assume that header files are in /includes directory

```
cmake_minimum_required(VERSION 3.4)
project(myproject)

include_directories(${CMAKE_SOURCE_DIR}/includes)
add_library(mylib mylibrary.cxx)

add_executable(myprog myprog.cxx)
target_link_libraries(myprog mylib)
```

Using ROOTConfig.cmake

The CMake command **`find_package(ROOT ...)`** tries to locate an installation of ROOT following a *search procedure* and loads the ROOT settings (i.e. CMake variables)

```
cmake_minimum_required(VERSION 3.4)
project(myproject)

# Add the installation prefix for ROOT
list(APPEND CMAKE_PREFIX_PATH $ENV{ROOTSYS})

# Locate the ROOT package
find_package(ROOT REQUIRED)

# Define useful ROOT functions and macros
include(${ROOT_USE_FILE})

include_directories(${ROOT_INCLUDE_DIRS})
add_definitions(${ROOT_CXX_FLAGS})

add_executable(myprog myprog.cxx)
target_link_libraries(myprog ROOT::Hist)
```

In case you need a Dictionary

The macro ROOT_GENERATE_DICTIONARY wraps the `rootcling` tool for generating ROOT dictionaries (needed for I/O) using header files and LinkDef.h file

The generated file is called `<dictname>.cxx` that can be used to build libraries and executables

```
...
# Use provided macro to generate dictionary
ROOT_GENERATE_DICTIONARY(EventDict Event.h
                         MODULE Event
                         LINKDEF EventLinkDef.h)

# Create a shared library with generated dictionary
add_library(Event SHARED Event.cxx EventDict.cxx)
add_dependencies(Event EventDict)
target_link_libraries(Event ROOT::Hist ROOT::Tree)
```

Wrap up