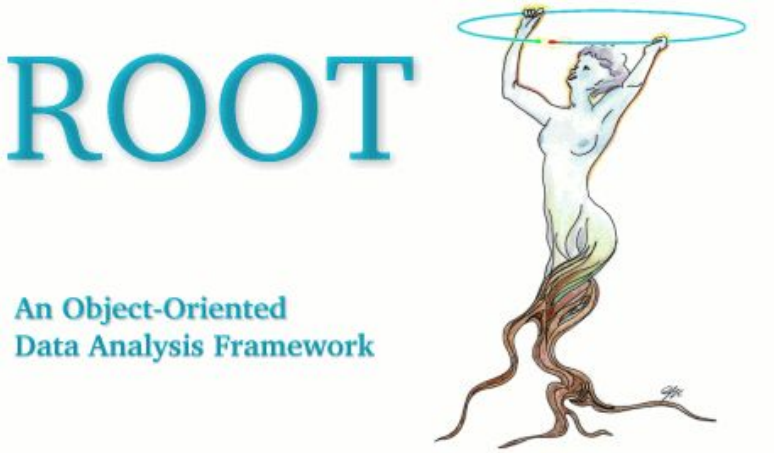
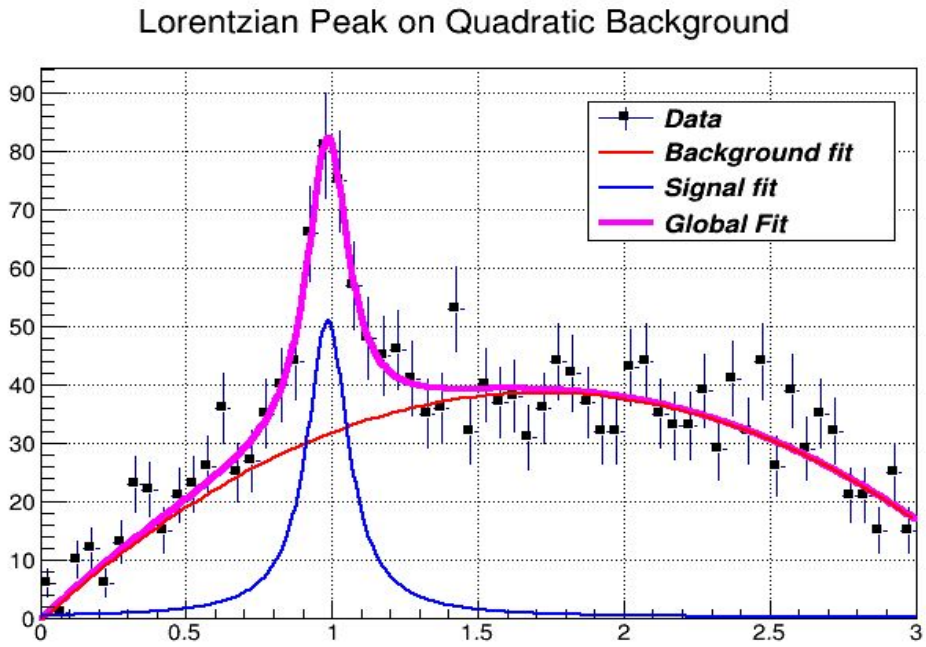
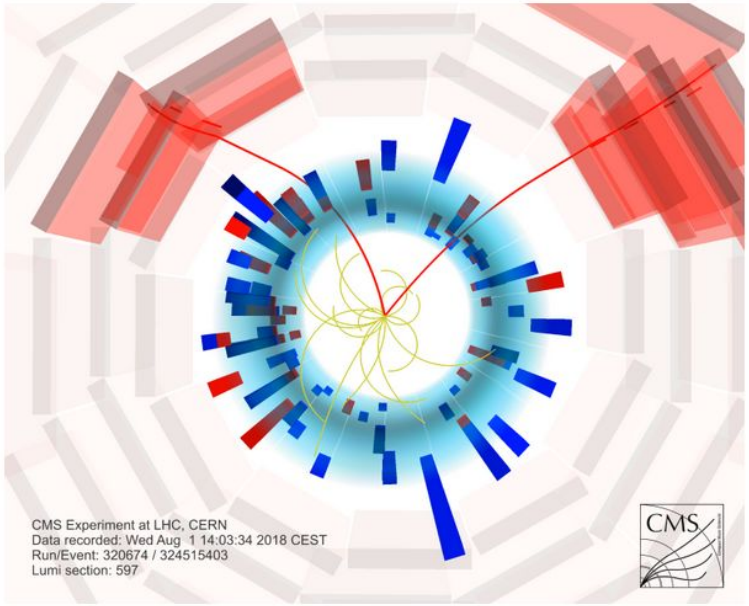






Big data en el CERN y otros contextos

Jhovanny Andres Mejia Guisao
UNIVERSIDAD DE ANTIOQUIA, COLOMBIA

Motivation

What we hope to discuss about scientific data analysis?

- Advanced graphical user interface
- Interpreter for the C++ programming language
- Persistency mechanism for C++ objects
- Used to write every year petabytes of data recorded by the Large Hadron Collider experiments

Input and plotting of data from measurements and fitting of analytical functions.

Input/Output

```
void write_to_file(){

  // Instance of our histogram
  TH1F myh("myh","myh",100,-5,5);

  // Let's fill it randomly
  myh.FillRandom("gaus");

  // Let's open a TFile
  TFile out_file("my_rootfile.root","RECREATE");

  // Write the histogram in the file
  myh.Write();

  // Close the file
  out_file.Close();
}
```

example1_write_to_file.C

```
root -l my_rootfile.root
root [0]
Attaching file my_rootfile.root as _file0...
root [1] _file0->ls();
TFile**         my_rootfile.root
 TFile*         my_rootfile.root
  KEY: TH1F     myh;1     myh
root [2] myh->Draw();
```

example2_Read_from_file.C

Interlude: I/O on cpp

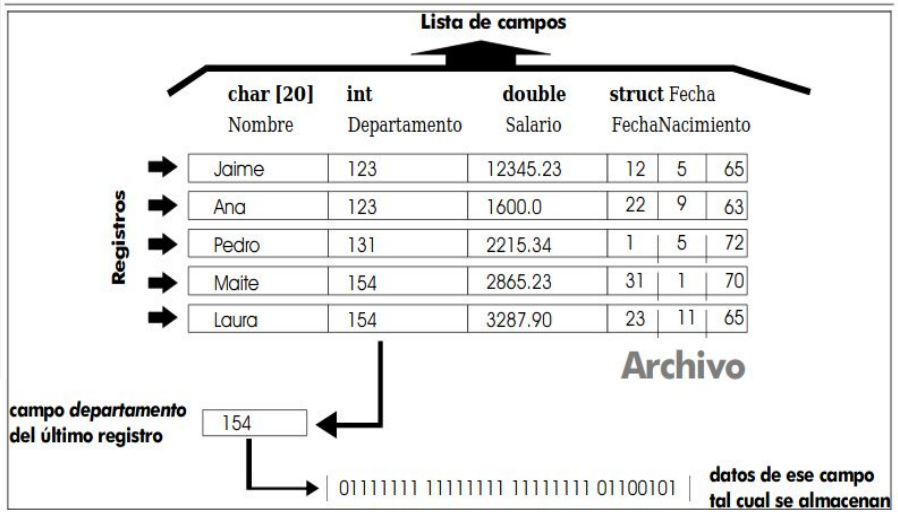
```
#include<iostream>
#include<fstream>
#include<cstdlib>
#include<string>

int main()
{
  cout << "xxxxxxx"<< endl;
    return 0;
}
```

-fstream archivoClientesEntrada("clientes.txt", ios::in);

-ifstream archivoClientesEntrada("clientes.txt");

```
string name_file = “clientes.txt”; // txt, tex, dat, etc
ifstream archivo_entr;
archivo_entr.open(name_file.c_str());
```

Prior to C++11, the filename was specified as a pointer-based string—as of C++11, it also can be specified as a string object.

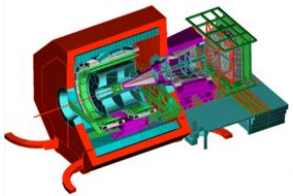


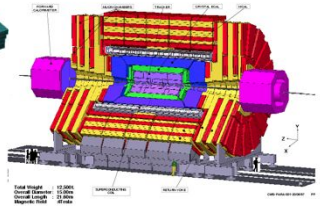
Learning Objectives

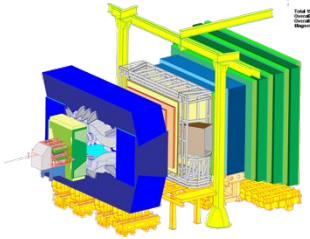
- Understand the relevance of I/O in scientific applications
- Grasp some of the details of the ROOT I/O internals
- Be able to write and read ROOT objects to and from ROOT files

I/O at LHC: an Example

A selection of the experiments adopting ROOT







Event Filtering

Data

Analysis

Offline Processing

Reconstruction

Further processing, skimming

Event Selection, statistical treatment …

Raw

Reco

…

Analysis Formats

Images

Data Storage: Local, Network

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 2022 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!

TFile in Action

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

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).

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();
```

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

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

Let's do some examples

```
root [0] TFile* f = new TFile ("myfile1.root", "READ");
root [1] if (f->IsOpen () == true) cout << "File open.\n";
root [2] f->ls();
root [4] TH1F* h = new TH1F ("h", "myhisto", 10, 0., 10.);
root [5] h->Write ();
```

Example3_files.C

-Setting up the work directory on a disk
-Execution of Linux command

```
root [0] gSystem->pwd ()
root [1] gSystem->cd ("../")
root [2] gSystem->pwd ()
root [3] gSystem->Exec ("date")
root [4] TString datenow = gSystem->GetFromPipe ("date");
root [5] datenow
root [6] TString datenow2(datenow(0,6))
root [7] TString datenow3(datenow(6,11))
```

```
root [0] TFile* f = new TFile ("myfile.root", "UPDATE");
root [1] f->ls();
root [2] TH1F* hr = (TH1F*) f-> Get ("myh1");
root [3] hr->Draw();
root [4] TH1F* myh3 = new TH1F ("myh3", "myh3", 100, 0., 10.);
root [5] TRandom3 r; r.SetSeed();
root [6] for (int i=0; i<1e5; i++) myh3->Fill ( r.Gaus(5,1) );
root [7] myh3->Write ("h_copy");
root [8] f->ls();
root [9] f->Close();
```

Hierarchy of objects in Root files and memory

```
root [1] gDirectory->pwd();
root [2] TFile f1 ("my_rootfile.root");
root [3] cout << gDirectory->GetPath () << endl;
root [4] .ls
```

Creating subdirectories (in memory or inside a .root file)

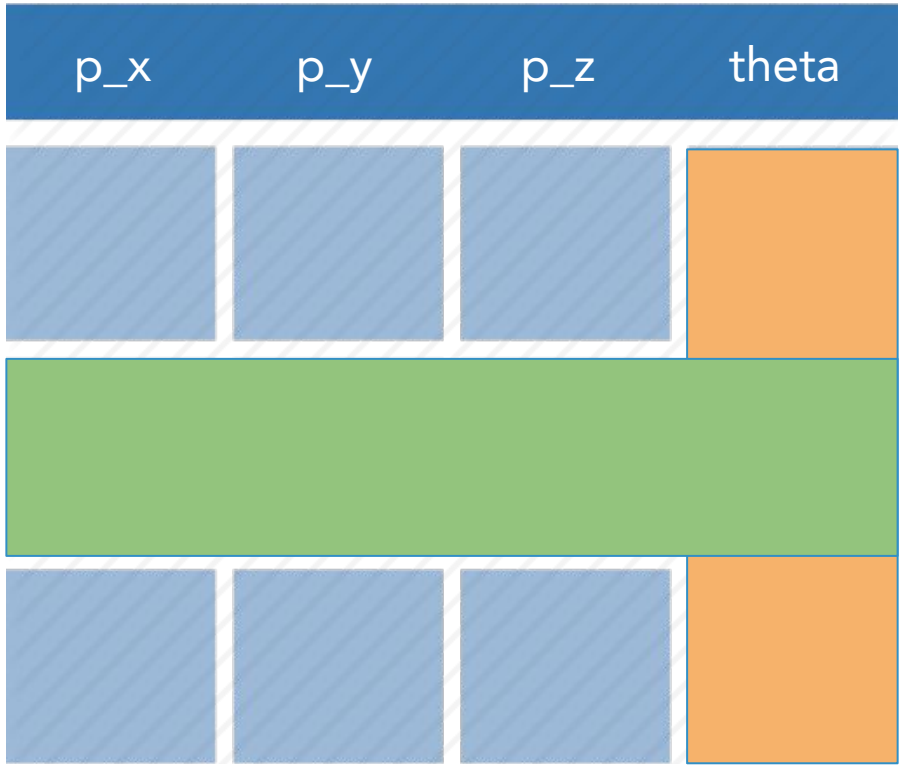
```
root [6] TFile f2 ("newfile.root", "RECREATE")
root [7] .ls
root [8] gDirectory->mkdir ("folder1");
root [9] .ls
root [10] gDirectory->cd ("folder1");
root [11] .pwd
root [12] TH1F h ("myhisto", "", 10, -5., 5);
root [13]  TRandom3 r; r.SetSeed();
root [14] for (int i=0; i<1e5; i++){ h.Fill ( r.Gaus() );}
root [15] h.Write();
root [16] .ls
```

```
root [18] gDirectory->cd ("..") ;
root [19] .ls
root [20] gDirectory->rmdir ("folder1")
root [21] .ls
root [22] f2.Close ();
root [23] .ls
root [24] cout << gDirectory->GetPath() << endl;
root [25] gDirectory->cd ("Rint:/");
root [26] .ls
root [27] gDirectory->pwd();
```

The ROOT Columnar Format

- 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



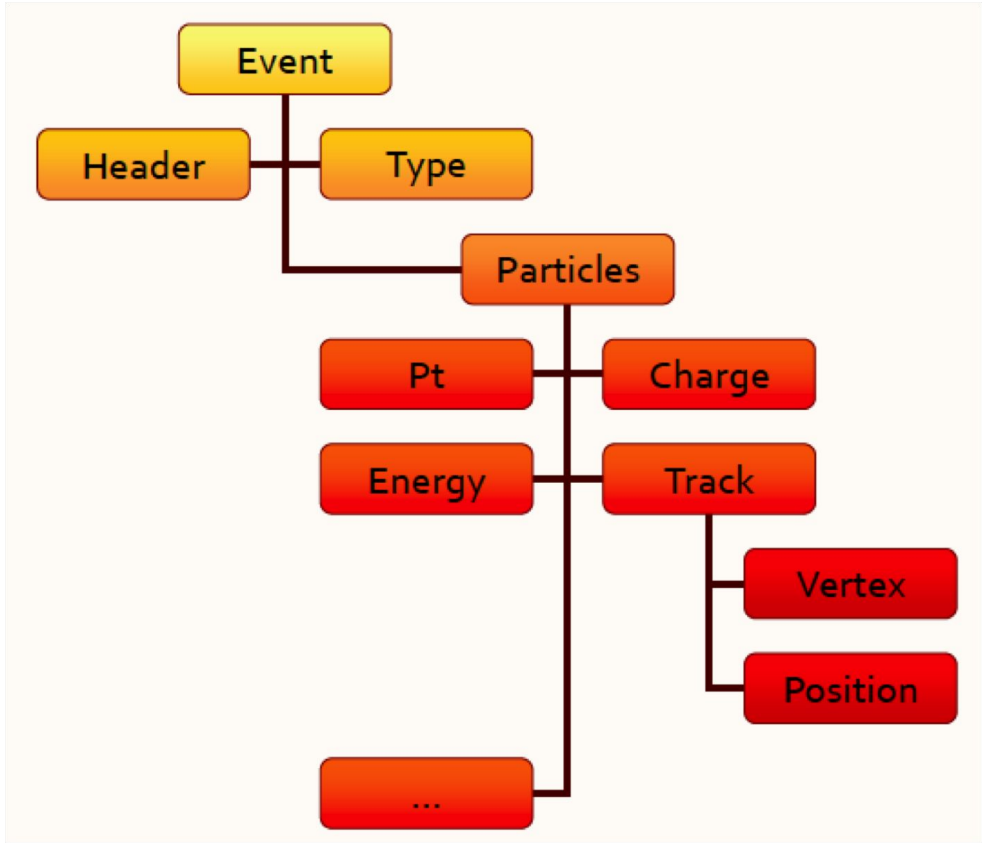
entries
or events
or rows

columns
or "branches"

can contain any kind
of c++ object

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
0.55	-0.21231	50281
-0.184	1.187305	.443902
0.20564	-0.770	0.635417
1.079222	0.32	1.271904
-0.27492	- 43	3.038899
2.047779	- 268	4.197329
-0.45868	4 2	2.293266
0.304731	0.884	0.875442
-0.712	-0.2223	0.556881
-0.27	1.181767	470484
0.88 02	-0.65411	13209
-2.03555	0.527648	4.421883
-1.45905	-0.464	2.344113
1.230661	-0.00565	1.514559
		3.562347



TTree

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

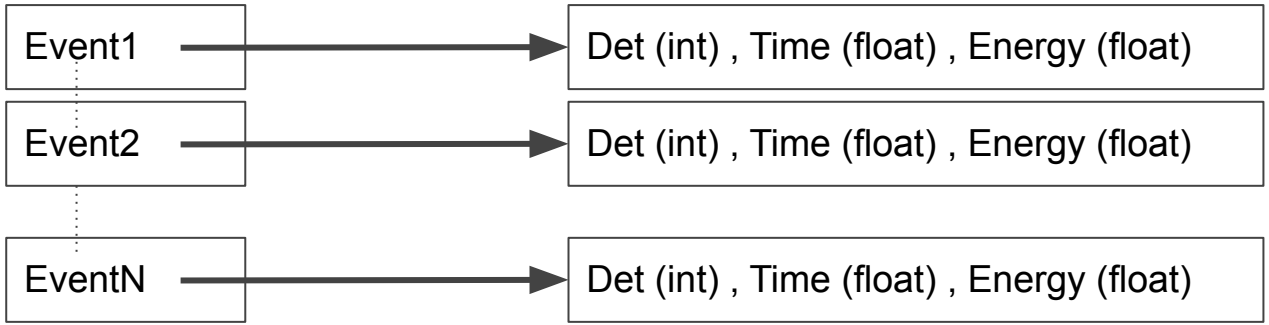
- Also called *tree*, columns also called *branches*
- An object type per column, **any type of object**
- One row per *entry* (or, in collider physics, *event*)

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

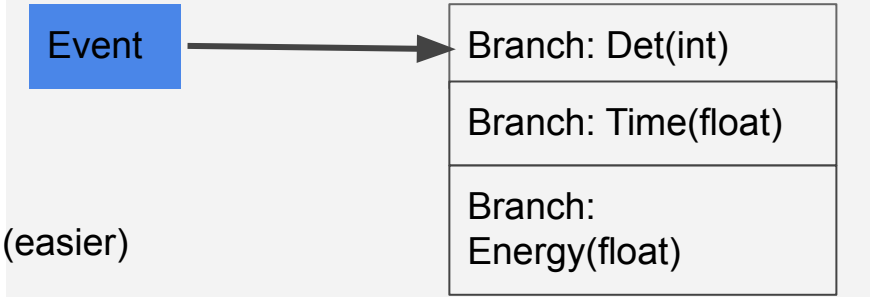
TTree

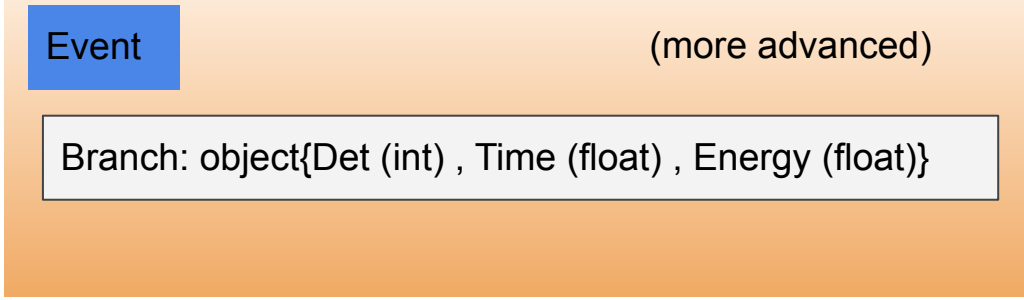
Eg. experiment measuring particles in telescopes: set of N_i, T_i, ΔE_i, E_i from a detector (detectors)
Eg. experiment measuring tracks of particles in drift chambers : set of px , py , pz , ΔE_i from a chamber

A simple data scheme



Possible event structures:





N-tuples in ROOT

TNtuple object can store rows of float entries

example4_write_ntuple_to_file.C **example5_read_ntuple_to_file.C**

Open the ROOT file (conductivity_experiment.root) written by the macro above in an interactive session and use a TBrowser to interactively inspect it:

```
root [1] TBrowser ll
(TBrowser &) Name: Browser Title: ROOT Object Browser
root [2] cond_data->Draw("Current:Potential")
root [3] cond_data->Draw("Current")
root [4] cond_data->Draw("Potential")
root [5] cond_data->Draw("Current:Potential","Temperature<270")
root [6] cond_data->Draw("Current/Potential:Temperature")
```