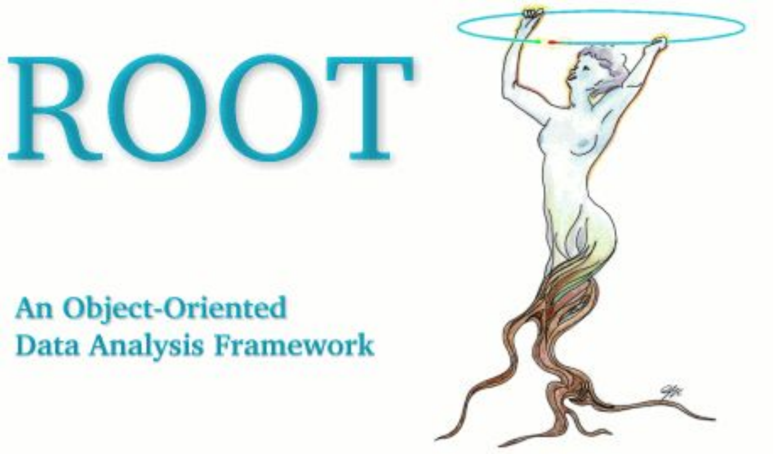
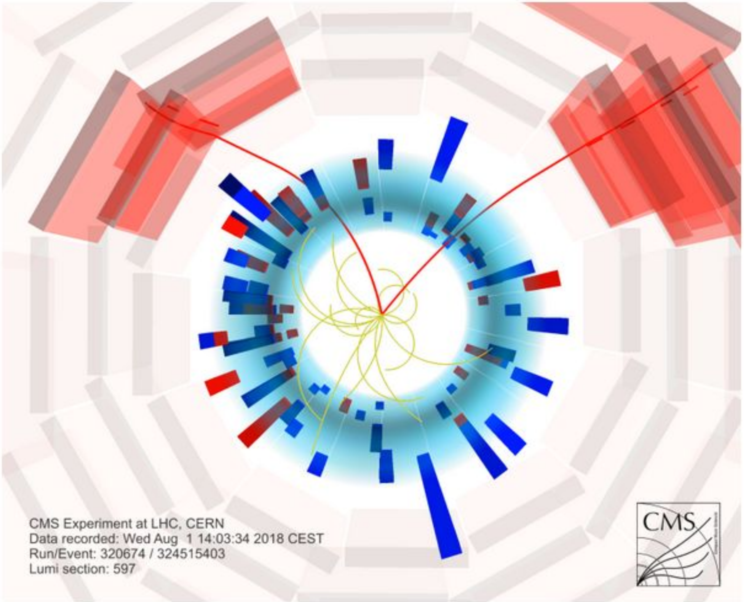
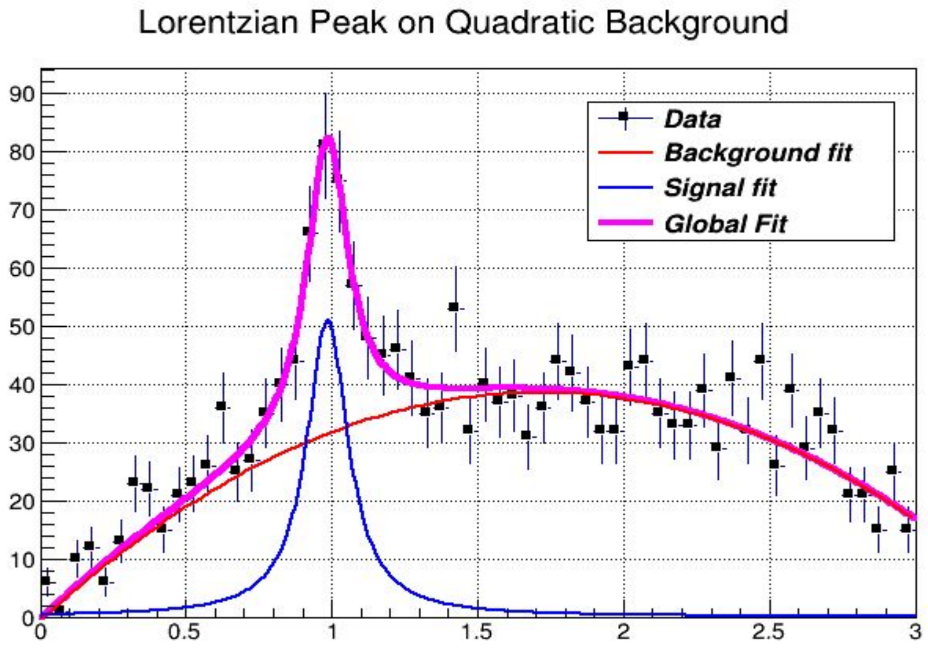






Big data With ROOT CERN

Jhovanny Andres Mejia Guisao
Centro Interdisciplinario de Investigación
y Enseñanza de la Ciencia (CIIEC)

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();
```

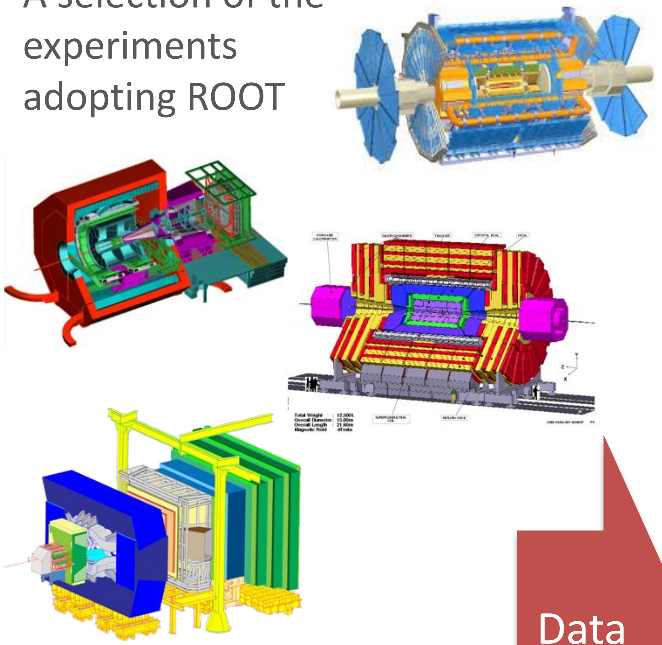
example2_Read_from_file.C

Learning Objectives

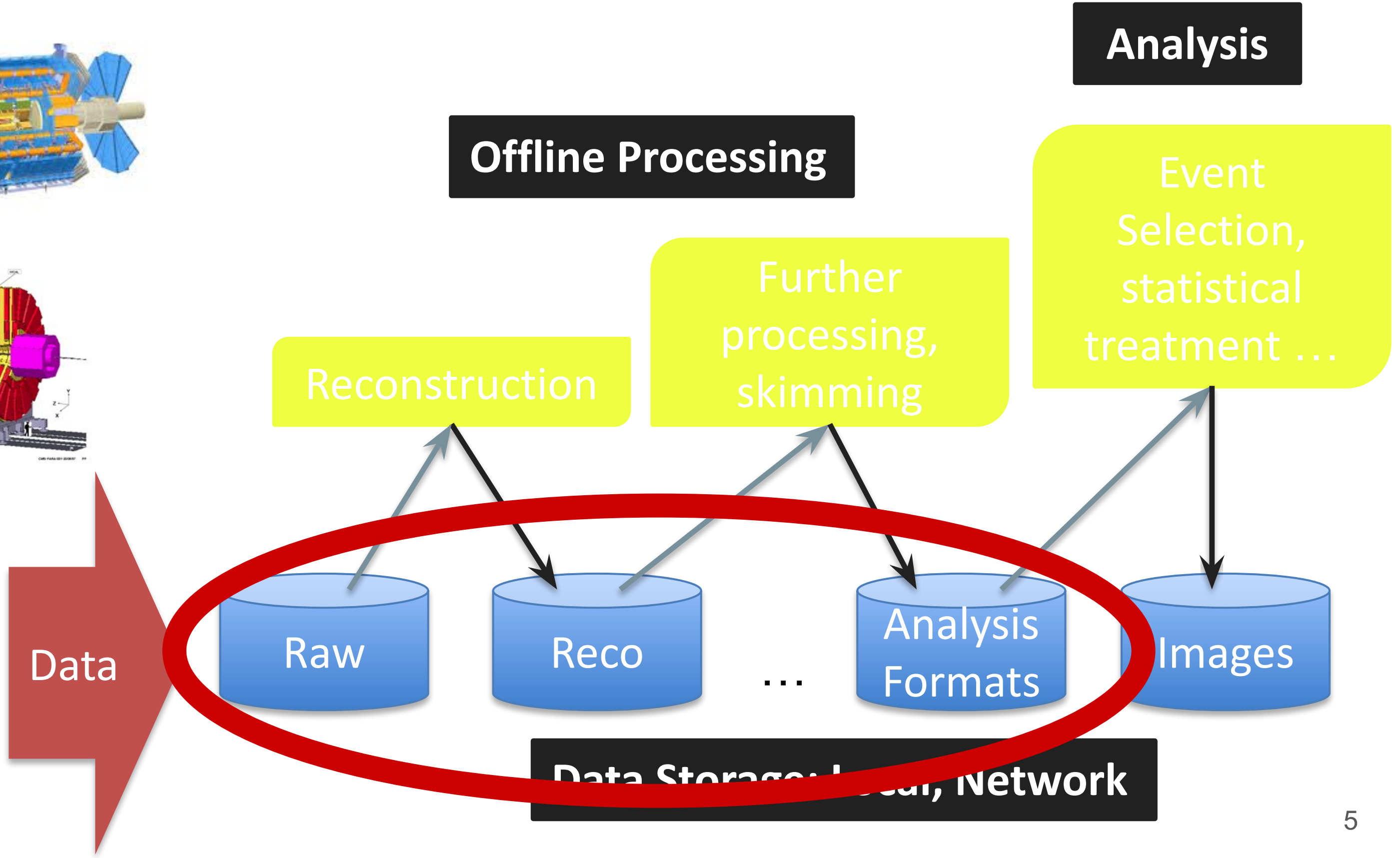
- Understand the relevance of I/O in scientific applications
- Grasp some of the details of the ROOT I/O internals
- Be able to write and read ROOT objects to and from ROOT files

I/O at LHC: an Example

A selection of the experiments adopting ROOT



Event Filtering



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

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 ("myfile1.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)

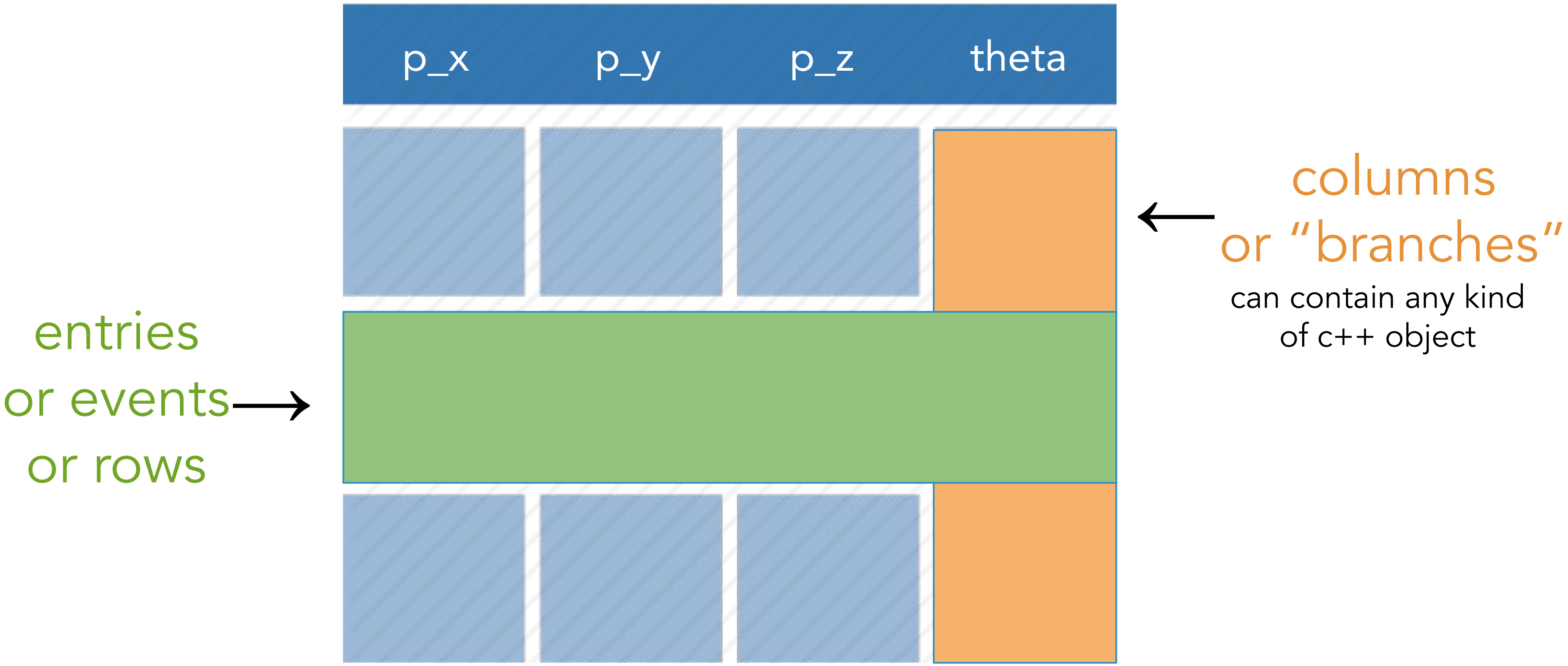
```
root [6] TFile f2 ("newfile.root", "RECREATE")
root [7] .ls
root [8] gDirectory->mkdir ("folder1");
root [9] .ls
root [10] gDirectory->cd ("folder1");
root [11] .pwd
root [12] TH1F h ("myhisto", "", 10, -5., 5);
root [13]  TRandom3 r; r.SetSeed();
root [14] for (int i=0; i<1e5; i++){ h.Fill ( r.Gaus()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

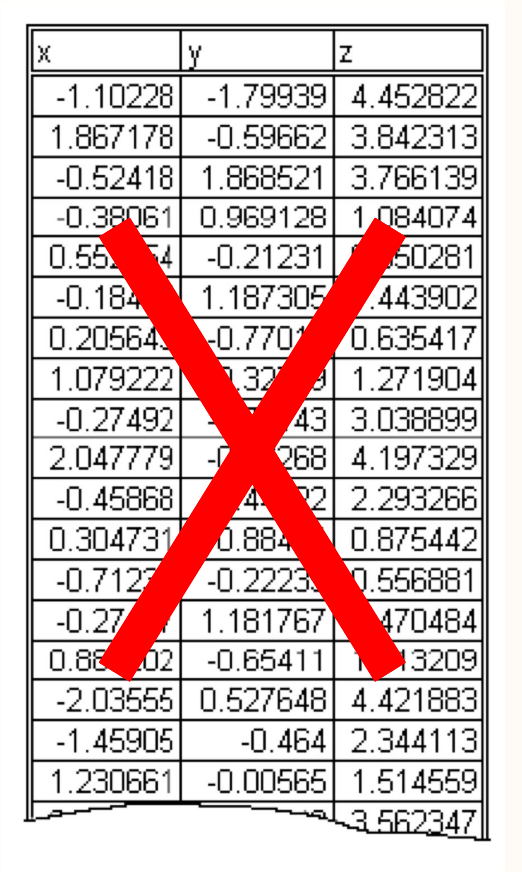
The ROOT Columnar Format

- 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**

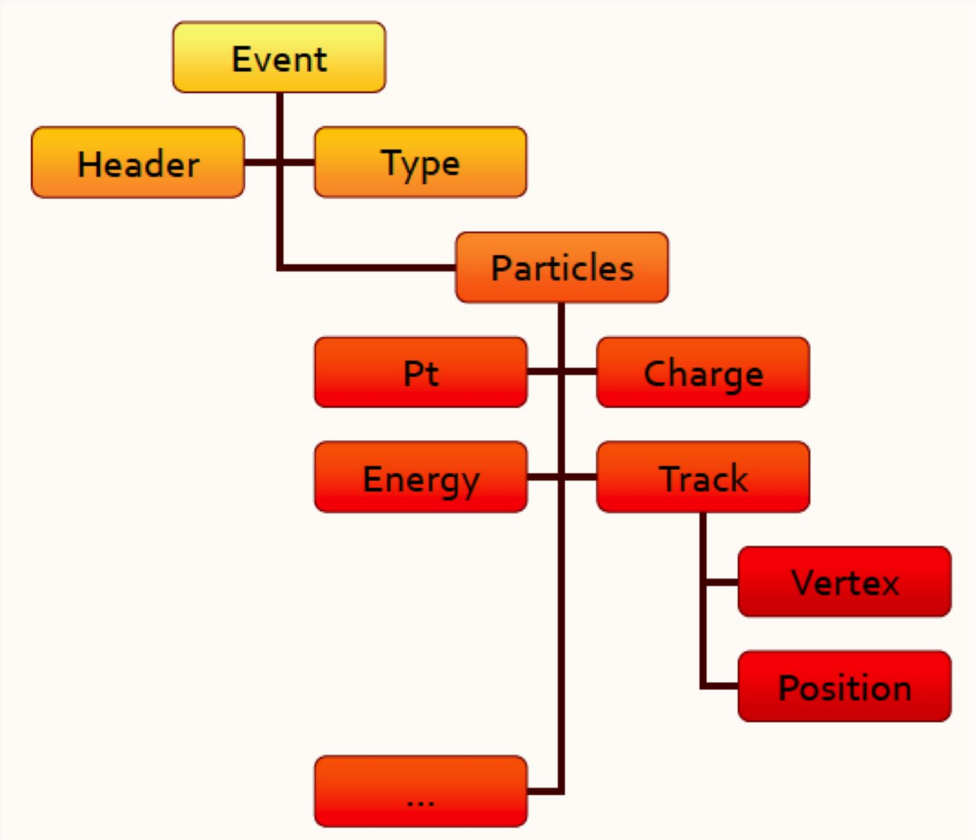
Columnar Representation



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



TTree

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

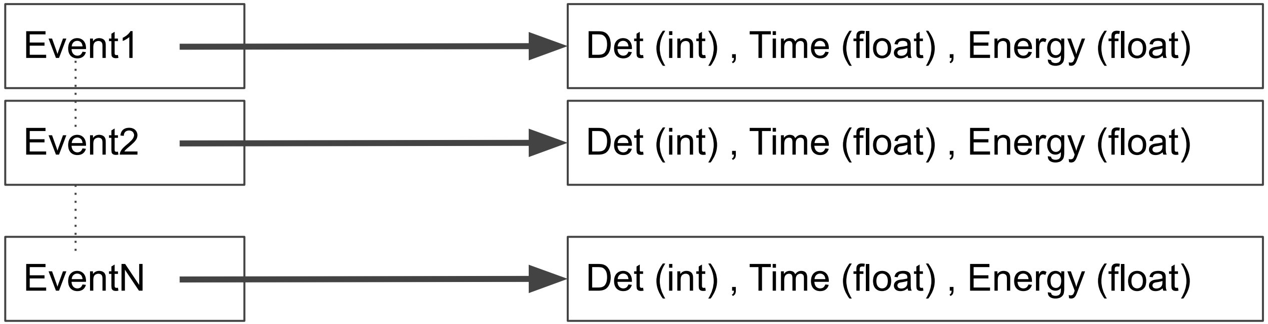
- Also called *tree*, columns also called *branches*
- An object type per column, **any type of object**
- One row per *entry* (or, in collider physics, *event*)

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

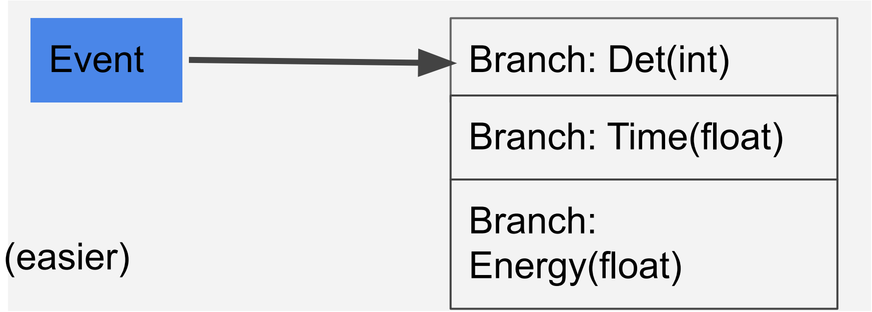
TTree

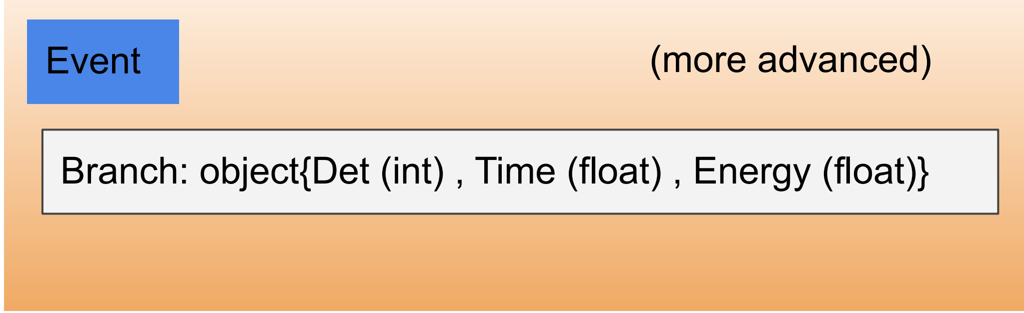
Eg. experiment measuring particles in telescopes: set of N_i, T_i, ΔE_i, E_i from a detector (detectors)
Eg. experiment measuring tracks of particles in drift chambers : set of px , py , pz , ΔE_i from a chamber

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")
```

???

???

???

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 explored how to read TTrees starting from the TNtuple examples

TTree: Simplest way to create a tree from data in text file

MyExpData.txt: They are in 3 columns (int, float, float)

Let's open the TTree object and fill the database with data using the ReadFile method:

ReadFile fills the tree from txt file

Print displays stats

Scan prints out data

Draw : en-time plot
We set up filter on det

Write saves tree in file

Example6_myTtree_fromfile.C

```
int Example6_myTtree_fromfile()
{
  TFile f( "simplest_tree.root", "RECREATE" );
  TTree t( "mytree" , "Tree of data for my analysis" );

  t.ReadFile ("MyExpData.txt", "det/I:energy/F:time/F" );

  t.Print ();

  t.Scan ( "det:energy:time" );

  t.Draw ( "energy:time" , "det >= 7" );

   t.Write ();

  return 0;
}
```

Now we'll design the tree by ourselves.

Scheme "1 branch = 1 variable"

Define the branch:

t.Branch ("Name", &variable,"variable/F");

List of variable types that can be used to define the type of a branch in ROOT:

type	size	C++	identifier
signed integer	32 bit	int	I
	64 bit	long	L
unsigned integer	32 bit	unsigned int	i
	64 bit	unsigned long	l
floating point	32 bit	float	F
	64 bit	double	D
boolean	-	bool	O

Example7_myTtree_fromfile.C
Example7_myTtree_fromfile2.C

```
int TTree_simple () {
Int_t det;
Float_t energy , time;
TFile f ("simple.root", "RECREATE");
TTree t ("tree", "My tree");
t.Branch ("Det" , &det, "det/I");
t.Branch ("En" , &energy, "energy/F");
t.Branch ("Time", &time , "time/F");

TRandom3 r; r.SetSeed ();
for (int i = 0; i < 100; i++) {
det = r.Integer (24);
time = r.Rndm() * 20.;
energy = r.Rndm() * 30.;
t.Fill ();
}
t.Write ();
return 0;
}
```

Rndm??

Making an entry in the tree

Writing the tree in a file

Inspection of the tree in an interactive session

```
root -l --web=off ttree_simple.root
root [2] mytree->Print();
******************************************************************************
*Tree   :tree    : My tree                                                   *
*Entries :       100 : Total =            3213 bytes  File  Size =      1723 *
*        :           : Tree compression factor =   1.20                      *
******************************************************************************
*Br      0 :Det          : det/I                                             *
*Entries :       100 : Total  Size=    947 bytes  File Size  =       231     *
*Baskets :         1 : Basket Size=      32000 bytes  Compression=   2.03    *
*............................................................................*
*Br      1 :En           : energy/F                                          *
*Entries :       100 : Total  Size=    954 bytes  File Size  =       469     *
*Baskets :         1 : Basket Size=      32000 bytes  Compression=   1.00    *
*............................................................................*
*Br      2 :Time         : time/F                                            *
*Entries :       100 : Total  Size=    952 bytes  File Size  =       471     *
*Baskets :         1 : Basket Size=      32000 bytes  Compression=   1.00    *
*............................................................................*
root [3] mytree->Show(10);
======> EVENT:10
 det             = 4
 energy          = 23.3717
 time            = 11.4525
```

```
root [4] tree->Scan();
************************************************
*    Row   * Det.Det.d * En.En.ene * Time.Time *
************************************************
*        0 *        16 * 19.158380 * 10.351733 *
*        1 *        15 * 0.9651761 * 2.1733038 *
*        2 *        21 * 17.750726 * 7.3583464 *
*        3 *        22 * 10.996044 * 4.7895526 *
*        4 *         8 * 4.5340204 * 17.619024 *
*        5 *        15 * 10.425326 * 8.0490799 *
*        6 *        11 * 20.414157 * 9.5272665 *
*        7 *        16 * 2.7496554 * 2.2426822 *
*        8 *         5 * 2.5598568 * 8.5067834 *
*        9 *        10 * 7.9277925 * 6.5382695 *
```

Plotting the histogram of a variable (variables, combination of variables, etc)

```
root -l --web=off ttree_simple.root
root [2] tree->Draw("En")
root [3] tree->Draw("sqrt(En)")
root [4] tree->Draw("time:energy","", "colz")
```

Example of function of variable

2-dimensional plot

Plotting the histogram of a variable with some filters (cuts) required

```
root [6] tree->Draw ("Time", "Det>14 && Det<23")
```

Projection of variables from a tree to a histogram

```
root [7] TH1F henergy ("henergy", "henergy", 15, 0., 30. );
root [8] tree->Project("henergy", "En", "Det<=10");
root [9] henergy.Draw();
```

Cuts (TCut)

```
root [21] TCut cut3 ("Det>10") , cut4 = "Det<20" ;
root [22] tree->Project ("henergy", "En", cut3 && cut4 );
root [23] henergy.Draw();
```

One can combine TCut with string

```
root [9] tree->Draw ("energy", cut3 && "Det<20" );
```

Getting the TTree from the ROOT file + readout of data from TTree:

```
int TTree_simple_read () {
  Int_t det;
  Float_t energy , time;
  TFile f ("ttree_simple.root");
  TTree* t = (TTree*) f.Get ("tree");

  t->SetBranchAddress ("Det" , &det);
  t->SetBranchAddress ("En" , &energy);
  t->SetBranchAddress ("Time", &time );

  for(int i=0; i<t->GetEntries(); i++)
        {
        t->GetEntry (i);
        cout << setw( 5) << det
        << setw(12) << time
        << setw(12) << energy << endl;
        }
  return 0;
}
```

Connect to the tree:
TTree* t = (TTree*) f.Get ("tree");

Connect the variables to the branches:
t->SetBranchAddress ("name",&variable);

Get the number of entries:
t->GetEntries();

Read the full event into the variables:
t->GetEntry (i);

Merging data from ROOT files with the same structure

If we need to analyse a series of files with TTree that has the same structure, we can of course make a loop: open i-th file, connect the tree and branches, analyse data, and close that file.
However, if we store the resulting histograms in a common output file, one often has to switch back and forth the gDirectory.

There is an alternative: [merging the input data.]()

[TChain.]() Object being effectively a queue of subsequent TTrees in specified files.
Let's assume that every input file has a TTree called "T".

TTree_simpleN.C
TTree_simpleN_readN.C

1. **Create the TChain: TChain myChain ("T");**
2. **Add subsequent files: myChain.Add ("file1.root");**
   myChain.Add ("file2.root");
   myChain.Add ("file3.root");
1. **Since now we use the myChain object, as if it was the common input tree.**

The hadd executable, runnable from prompt :
> hadd data_merged.root data_1.root data_2.root .... (or: data_*.root)
Caution: Caution the maximum size of resulting file is set to 100 GB. For bigger data there is a TFileMerger class.

TVectorN {N = 2, 3} / TLorentzVector object in an event:

Storage

```
int Example8_MyTreeWithVector() {

 TVector3 v3;
 TVector3 *pv3 = &v3;
 TLorentzVector vL;
 TLorentzVector *pvL = &vL;

 TFile file ("TTree_TVector.root", "recreate");
 TTree *ttree = new TTree ("ttree", "ttree");
 ttree->Branch ("v3", "TVector3"             , &pv3);
 ttree->Branch ("vL", "TLorentzVector", &pvL);

 TRandom3 r; r.SetSeed (0);
 for (int evt = 0; evt < 100; evt++)
 {
   v3.SetXYZ  (r.Rndm(), r.Rndm(), r.Rndm());
   vL.SetXYZT (r.Rndm(), r.Rndm(),
            r.Rndm(), r.Rndm() );
   ttree->Fill();
 }
 ttree->Write();
 file.Close();
 return 0;
}
```

Readout

```
int Example8_MyTreeWithVector_read() {
 TVector3 v3;
 TVector3 *pv3 = &v3;
 TLorentzVector vL;
 TLorentzVector *pvL = &vL;

 TFile f ("TTree_TVector.root");
 TTree *ttree = (TTree*) f.Get("ttree");
 ttree->SetBranchAddress ("v3", &pv3);
 ttree->SetBranchAddress ("vL", &pvL);

 for (int evt=0; evt < ttree->GetEntries(); evt++)
 {
   ttree->GetEvent (evt);
   cout << "[" << evt << "]: ["
           << fixed << setprecision (3)
           << v3[0] <<" : "<< v3[1] << " : "
           << v3[2] << "]" << "\t";

   cout << "["   << vL[0] << " : " << vL[1]
           << " : " << vL[2] << " : " << vL[3]
           << "]\n";
 }
 f.Close();
 return 0;
}
```

Nota bene: Methods of TVector3 and TLorentzVector classes work. E.g.: tree->Draw (" v3.Mag() ")

Events with variable number of particles (the simplest way)

Storage

```
int Example9_MyEventmanyparticles() {
  Int_t Npart;
  Int_t det[500];
  Float_t energy[500] , time[500];

  TFile f ("manyparticles.root", "RECREATE");
  TTree t ("tree", "My tree");
  t.Branch ("Npart", &Npart, "Npart/I");
  t.Branch ("Det"  , det   , "det[Npart]/I");
  t.Branch ("Time" , time  , "time[Npart]/F");
  t.Branch ("En"   , energy, "energy[Npart]/F");

  TRandom3 r; r.SetSeed ();
  for (int ievt=0; ievt<100; ievt++) {
        Npart = r.Integer(6);
        cout << "Event " << ievt
        << " has " << Npart << " particles.\n";
        for (int ipart=0; ipart<Npart; ipart++)
        {
        det   [ipart] = r.Integer (24);
        time  [ipart] = r.Rndm() * 20.;
        energy[ipart] = r.Rndm() * 30.;
        cout << setw(10) << det   [ipart]
        << setw(12) << time  [ipart]
        << setw(12) << energy[ipart] << endl;
        }
        t.Fill ();
  }
  t.Write();
  return 0;
}
```

Readout

```
int Example9_MyEventmanyparticles_read() {
  Int_t Npart;
  Int_t det[500];
  Float_t energy[500] , time[500];

  TFile f ("manyparticles.root", "READ");
  TTree *t = (TTree*) f.Get ("tree");
  t->SetBranchAddress ("Npart", &Npart );
  t->SetBranchAddress ("Det"  , det  );
  t->SetBranchAddress ("Time" , time   );
  t->SetBranchAddress ("En"   , energy );

  cout << "* This tree has "
        << t->GetEntries() << " entries.\n\n";

  for (int ievt=0; ievt<t->GetEntries(); ievt++)
  {
        t->GetEntry (ievt);
        cout << "* Event " << ievt
        << " has " << Npart << " particles:\n";

        for (int ipart=0; ipart<Npart; ipart++)
        {
        cout << setw(10) << det   [ipart]
        << setw(12) << time  [ipart]
        << setw(12) << energy[ipart] << endl;
        }
  }
  return 0;
}
```

[myclass]

Using TTree::MakeClass

Tarea Module3

Tarea1:
Por favor, mire nuestros ejemplos:
"Example9_MyEventmanyparticles.C" y
“Example9_MyEventmanyparticles_read.C”

Realice esos mismos ejercicios,
pero esta ves no utilice "arreglos" sino la
clase "vector" de STL.

¿Podria Usar “<TTreePlayer::MakeClass>:”?