# Phys_f_416:
# $A_{FB}$ asymmetry measurement with the CMS detector

## Today's lesson

**Theory:**

- $A_{FB}$ asymmetry in Drell-Yan process
- CMS detector
- Generated vs reconstructed particles

**Practice:**

- Check your account on lxpub
- What is a c++ object
- C++ methods (~ functions in C)

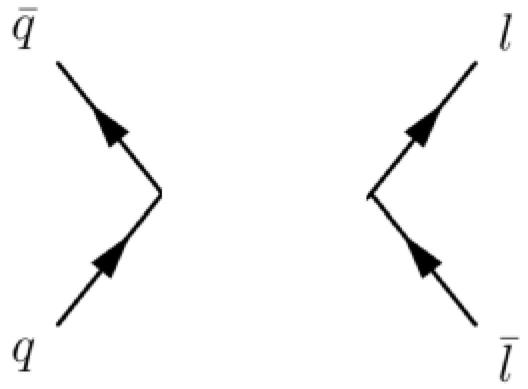
**Target:**

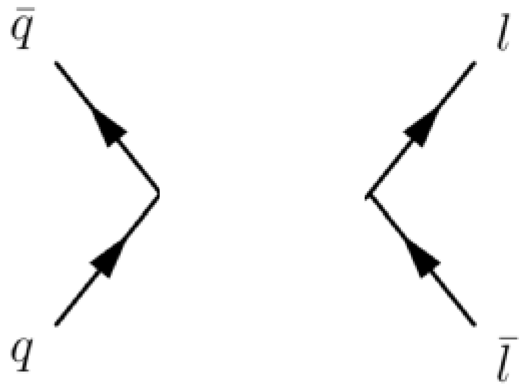
- write a function to compute generated and reconstructed mass spectra

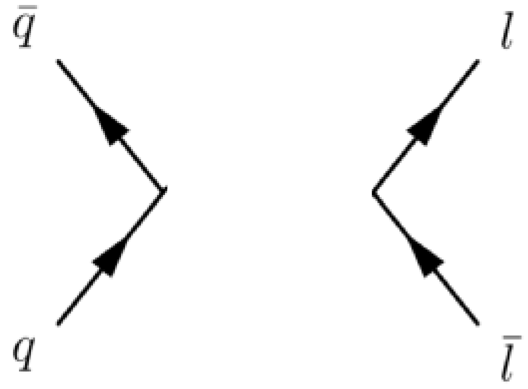
# Drell-Yan process

The Drell-Yan process is the annihilation of qq pairs in ll pairs (ee for us)

What are the SM contributions to this process (at leading order)?



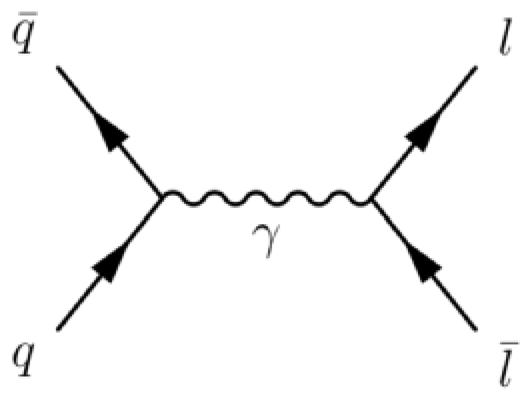


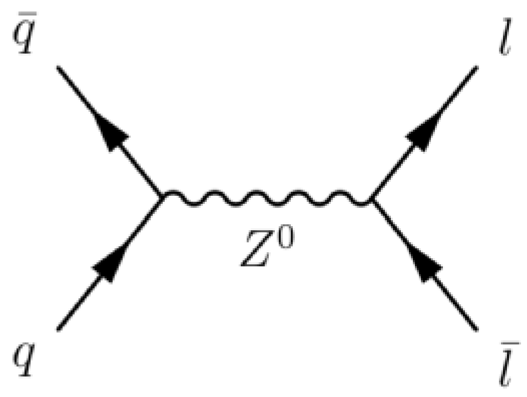


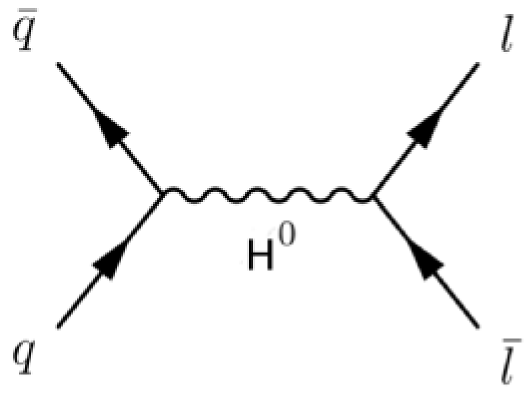
# Drell-Yan process

The Drell-Yan process is the annihilation of qq pairs in ll pairs (ee for us)

What are the SM contributions to this process (at leading order)?





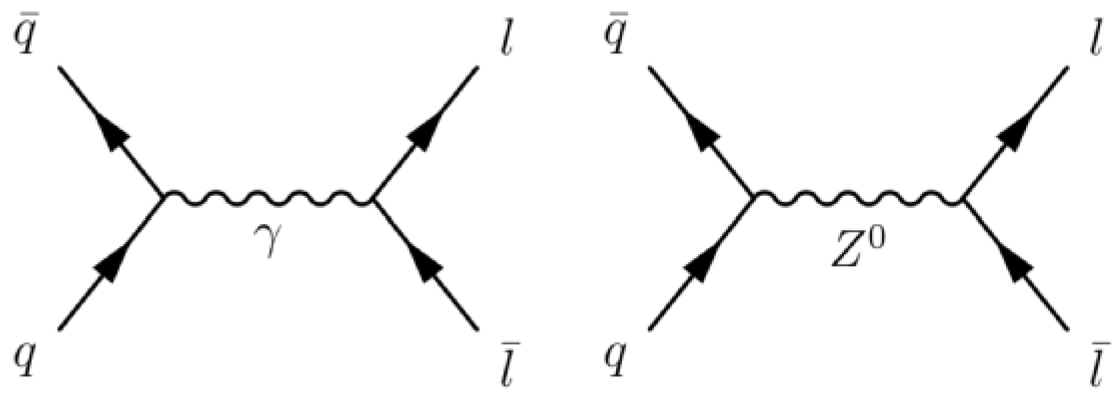


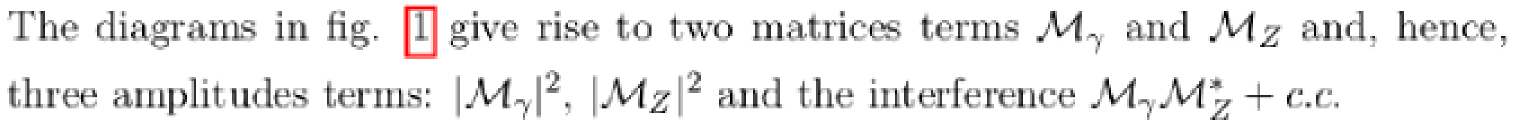
**One of them can be discarded**: which one? Why?

# Drell-Yan process

The Drell-Yan process is the annihilation of qq pairs in ll pairs (ee for us)

What are the SM contributions to this process (at leading order)?




The diagrams in fig. 1 give rise to two matrices terms $\mathcal{M}_\gamma$ and $\mathcal{M}_Z$ and, hence, three amplitudes terms: $|\mathcal{M}_\gamma|^2$, $|\mathcal{M}_Z|^2$ and the interference $\mathcal{M}_\gamma \mathcal{M}_Z^* + c.c.$

**Amplitudes terms:**
- "photon-only"
- "Z-only"
- Interference term

## Drell-Yan cross-section

**Photon-only** cross-section (like you only exchanged photons)

$$\frac{d\sigma_\gamma}{d\Omega} = \frac{e^4}{(4\pi)^2} Q_q^2 Q_l^2 \frac{1}{8s'} \left[(1+\cos\theta)^2 + (1-\cos\theta)^2\right]$$

**Z-only** cross-section (like you only exchanged Z)

$$\frac{d\sigma_Z}{d\Omega} = \frac{e^4}{(4\pi)^2} Q_q^2 Q_l^2 \frac{1}{8s'} |\mathcal{R}|^2 \left[c_{1,Z}(1+\cos\theta)^2 + c_{2,Z}(1-\cos\theta)^2\right]$$

$c_1$ and $c_2$ are **NOT** the same number !!

**Interference** term

$$\frac{d\sigma_{int}}{d\Omega} = \frac{e^4}{(4\pi)^2} Q_q^2 Q_l^2 \frac{1}{8s'} \mathrm{Re}(\mathcal{R}) \left[c_{1,int}(1+\cos\theta)^2 + c_{2,int}(1-\cos\theta)^2\right]$$

$c_1$ and $c_2$ are **NOT** the same number !!

- s' is the center-of-mass energy of the qq process (not the proton's one!)
- theta is the angle between the final state lepton and the initial state quark directions
- What do you notice about the theta dependance of the cross-sections?

# $A_{FB}$ asymmetry

The total cross section is:

$$\sigma = \int_\Omega \frac{d\sigma_{\gamma+Z}}{d\Omega} d\Omega = \frac{4\pi}{3}\frac{\alpha^2}{s'} c_1$$

where $\alpha = e^2/(4\pi)$.

$\sigma_F = \sigma_{\theta<\pi/2}$ and $\sigma_B = \sigma_{\theta>\pi/2}$ $\longrightarrow$ $A_{FB} = \frac{\sigma_F - \sigma_B}{\sigma_F + \sigma_B} = \frac{3}{8}\frac{c_2}{c_1}$

$$\boxed{A_{FB} = \frac{3}{8}\frac{c_2}{c_1}}$$

- Does the $A_{FB}$ depend on s' ?

$$c_1 = 1 + 2\mathrm{Re}(R) g_{Vl} g_{Vq} + |R|^2 \left(g_{Vl}^2 + g_{Al}^2\right)\left(g_{Vq}^2 + g_{Aq}^2\right)$$

$$c_2 = 4\mathrm{Re}(R) g_{Al} g_{Aq} + 8|R|^2 g_{Vl} g_{Al} g_{Vq} g_{Aq}$$

$$R = \frac{1}{Q_l Q_q \sin^2 2\theta_W} \frac{s'}{s' - M_Z^2 + is' \Gamma_Z / M_Z}$$

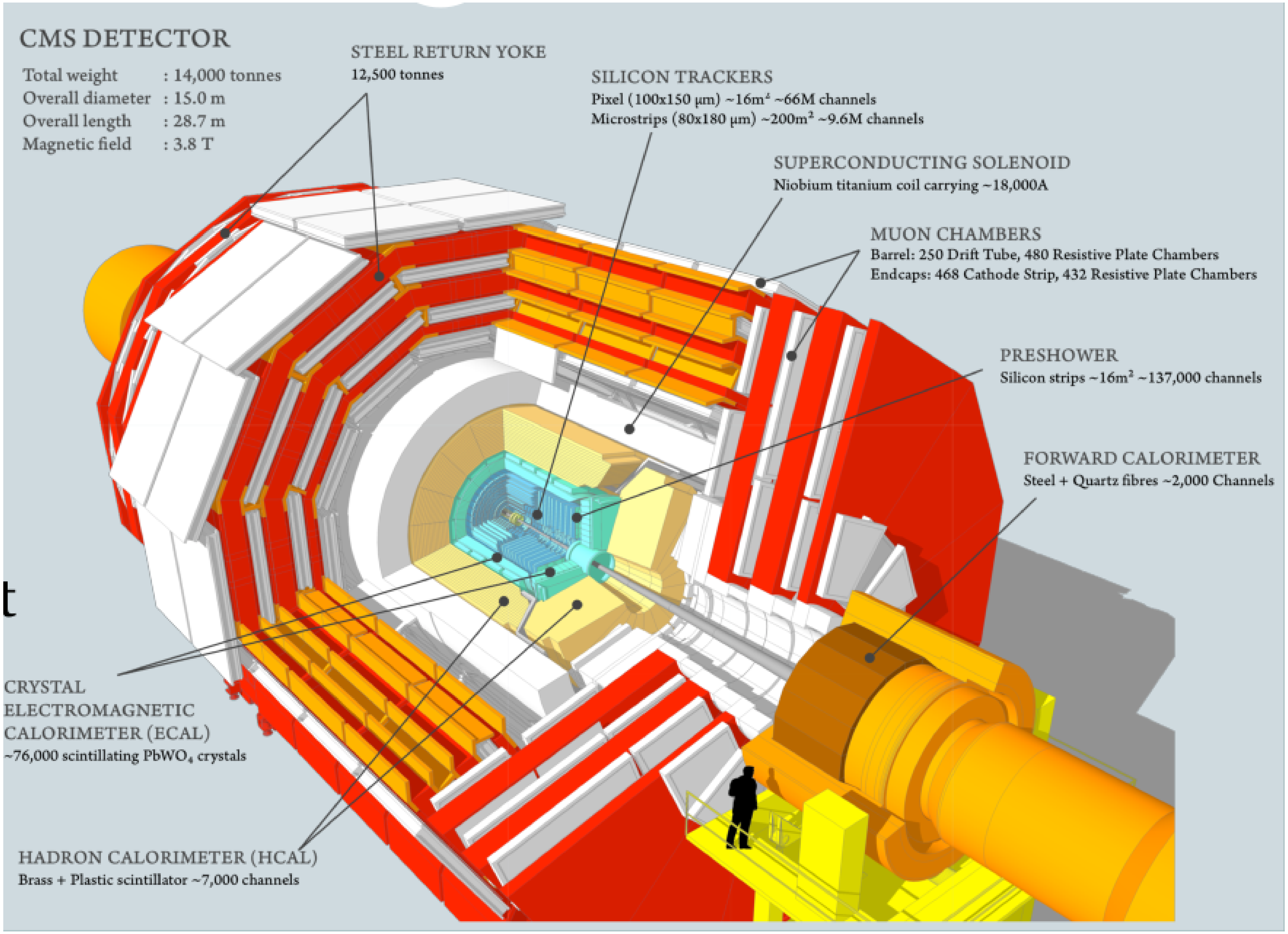
$$g_{Al,q} = -I^3_{Wl,q}$$

$$g_{Vl,q} = I^3_{Wl,q} - 2Q_{l,q} \sin^2\theta_W$$

| | $Q$ | $I^3_W$ |
|---|---|---|
| $e$ | -1 | -1/2 |
| $u$ | 2/3 | 1/2 |
| $d$ | -1/3 | -1/2 |

# The CMS detector

- Cylindrical "onion" made of "detector layers"

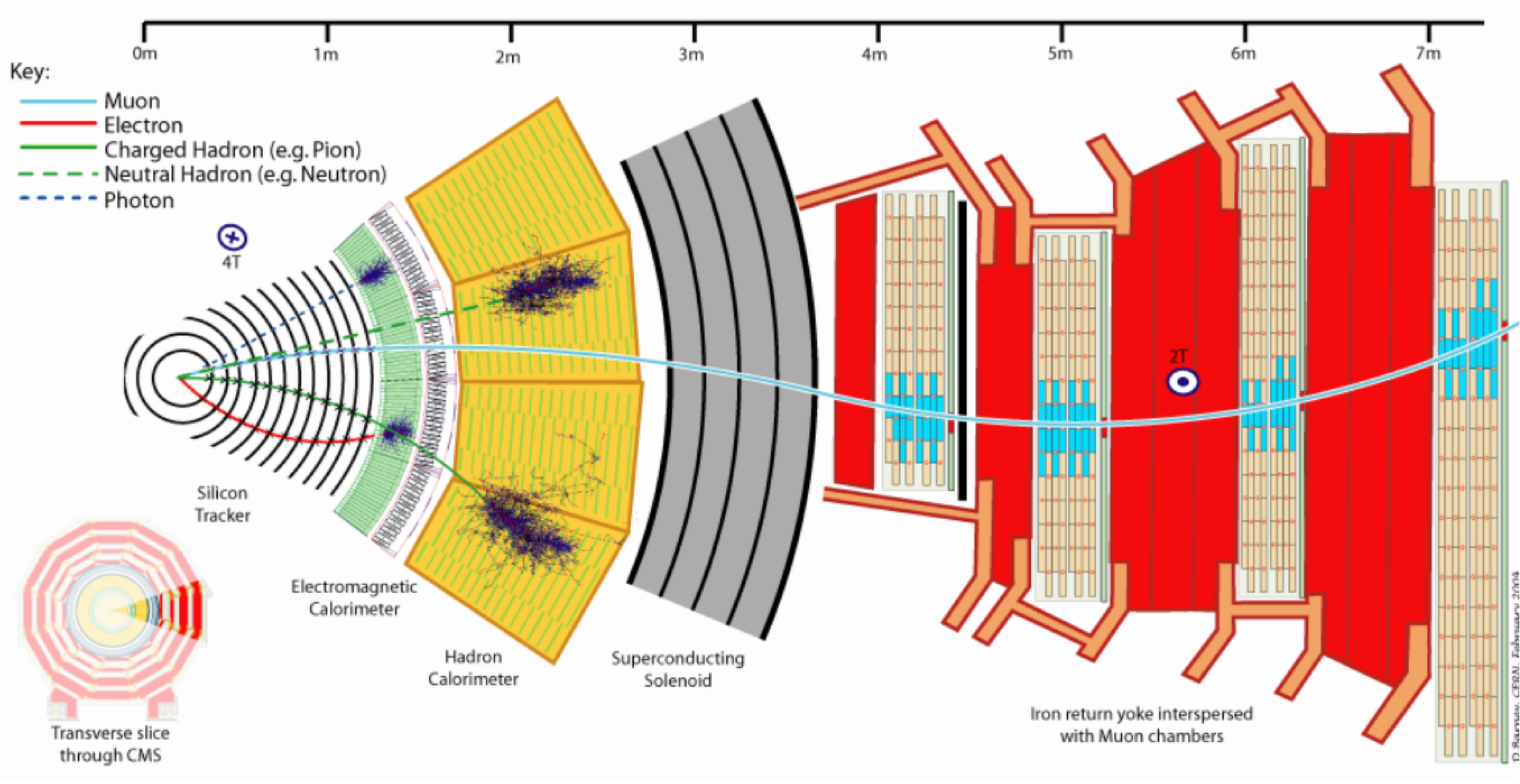


CMS DETECTOR
Total weight : 14,000 tonnes
Overall diameter : 15.0 m
Overall length : 28.7 m
Magnetic field : 3.8 T
STEEL RETURN YOKE
12,500 tonnes
SILICON TRACKERS
Pixel (100x150 μm) ~16m² ~66M channels
Microstrips (80x180 μm) ~200m² ~9.6M channels
SUPERCONDUCTING SOLENOID
Niobium titanium coil carrying ~18,000A
MUON CHAMBERS
Barrel: 250 Drift Tube, 480 Resistive Plate Chambers
Endcaps: 468 Cathode Strip, 432 Resistive Plate Chambers
PRESHOWER
Silicon strips ~16m² ~137,000 channels
FORWARD CALORIMETER
Steel + Quartz fibres ~2,000 Channels
CRYSTAL ELECTROMAGNETIC CALORIMETER (ECAL)
~76,000 scintillating PbWO4 crystals
HADRON CALORIMETER (HCAL)
Brass + Plastic scintillator ~7,000 channels

t

# How does CMS work?

- We look at the “layers of the onion”


0m
1m
2m
3m
4m
5m
6m
7m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon
4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
Transverse slice through CMS

We are going to look at electrons:

- What particle(s) can mimic an electron in the detector?

# Generated vs Reconstructed particles

**Generated** particles are the result of a **simulation** of a proton-proton collision (It's **THE truth**)

**Reconstructed** particles are what we get when generated particles interact with the **detector**:
(imperfect measurements of energy, momentum, position ecc...)

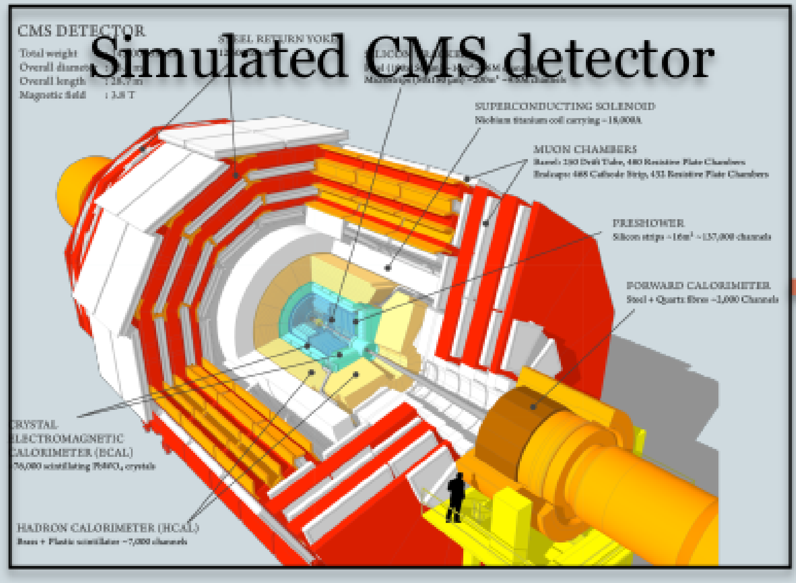
**Simulation:**
We generate particles → we simulate the detector → we simulate the detector reconstruction

**Data**:
Real particles are produced with a certain energy/momentum → they interact in the real detector → they are reconstructed by the real detector

In **Data** we **only** have **reconstructed particles**
Hence, if the simulation is good the simulated reconstructed particles should look like the real reconstructed particles.

# Generated vs Reconstructed particles

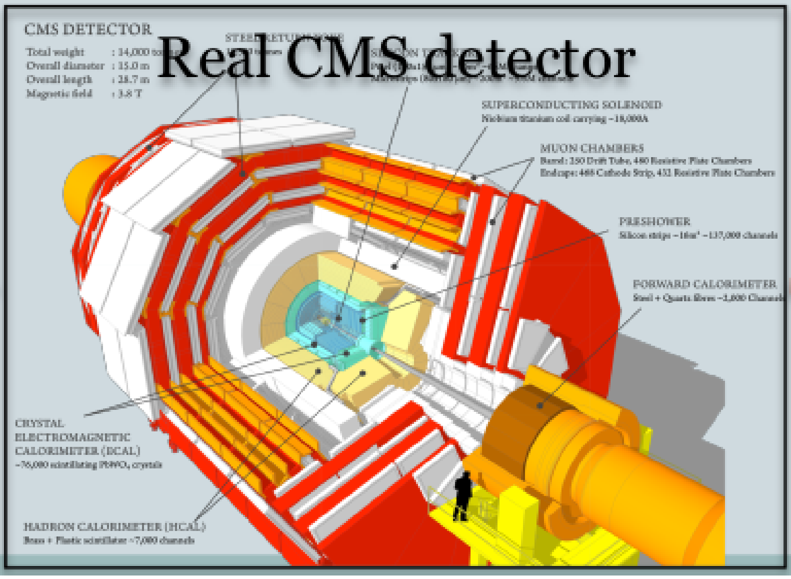
Simulation “Generated” particles



Simulation “Reconstructed” particles

Data

???



Data “Reconstructed” particles

## Hands-on!

## ssh login to your account & emacs basics

**Login:**
ssh -Y yourname@lxpub.iihe.ac.be
**Password** is: xxxxx
Now you pc is just a monitor: you are in fact using another remote pc!

**Open a file using emacs:**
emacs -nw  /ice3/phy_f_416_2016/Message_for_you.txt
**Search for a word**: Ctrl +s then type the word in the minibuffer
**Close the file: Ctrl + x** then (keep pushing Ctrl) **c**

**Now create your first file:**
emacs -nw my_first_file.txt
**Type something**
**Save: Ctrl + x** then (while keep pushing Ctrl) **s**
**Close the file**

This is ~ everything you need to know
about emacs for these lectures

## Linux basics
## (everything you need)

You just created a file my_first_file.txt

**Check if the file is there:** ls

**Create a new directory:** mkdir my_new_directory

**Go inside the directory:** cd my_new_directory

**Move your file inside the directory:** mv ../my_first_file.txt .

**Copy a file somewhere:** cp my_first_file.txt ../

## To do only the first time

source /cvmfs/cms.cern.ch/cmsset_default.sh
**cmsrel CMSSW_8_0_8**
#(the previous command will create a directory called CMSSW_8_0_8 )
cd CMSSW_8_0_8
cmsenv

Now you can use ROOT
Open root with:
root - l

Exit root with:
.q

## All the other times:

You can avoid to create a CMSSW release every time: you already did that!

```
source /cvmfs/cms.cern.ch/cmsset_default.sh
cd CMSSW_8_0_8
cmsenv
```

Now you can use ROOT

Open root with:

```
root - l
```

Exit root with:

```
.q
```

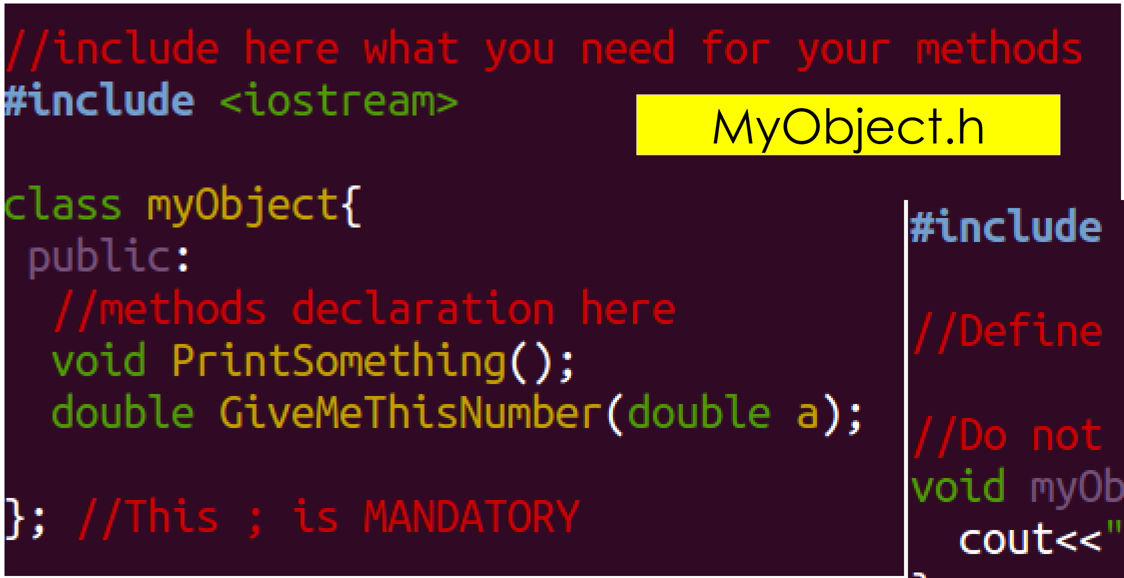
# C++ objects

A C++ object is an entity with specific capabilities

Let's make an example with your first C++ object: **myObject**

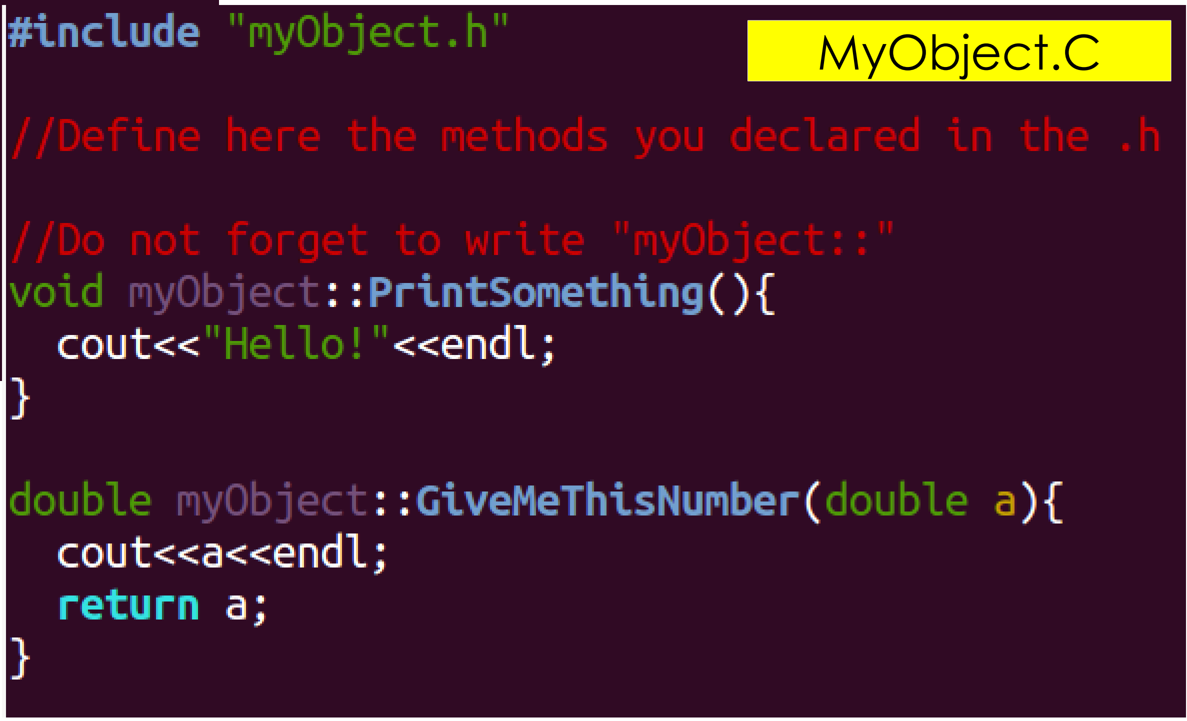
**First of all: clear-up your mind**. What do you need? For example, you want that the object is able to print a message

You need to create two files with emacs: **myObject.h** and **myObject.C**



MyObject.h

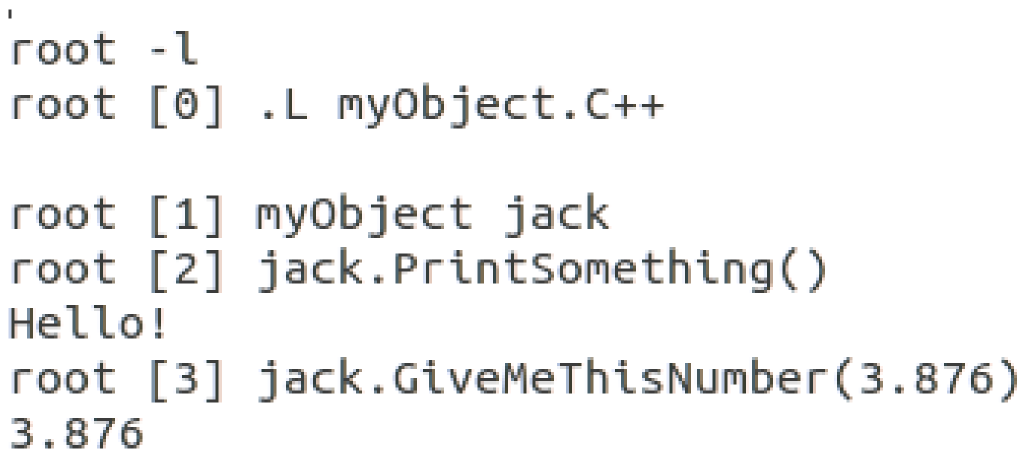
```
//include here what you need for your methods
#include <iostream>

class myObject{
 public:
  //methods declaration here
  void PrintSomething();
  double GiveMeThisNumber(double a);

}; //This ; is MANDATORY
```



MyObject.C

```
#include "myObject.h"

//Define here the methods you declared in the .h

//Do not forget to write "myObject::"
void myObject::PrintSomething(){
  cout<<"Hello!"<<endl;
}

double myObject::GiveMeThisNumber(double a){
  cout<<a<<endl;
  return a;
}
```

# C++ objects (II)



```
root -l
root [0] .L myObject.C++

root [1] myObject jack
root [2] jack.PrintSomething()
Hello!
root [3] jack.GiveMeThisNumber(3.876)
3.876
```

Now give your object another capability:
**write a method to sum two numbers together**

What do you have to do?

## Time for
## a small ROOT tutorial

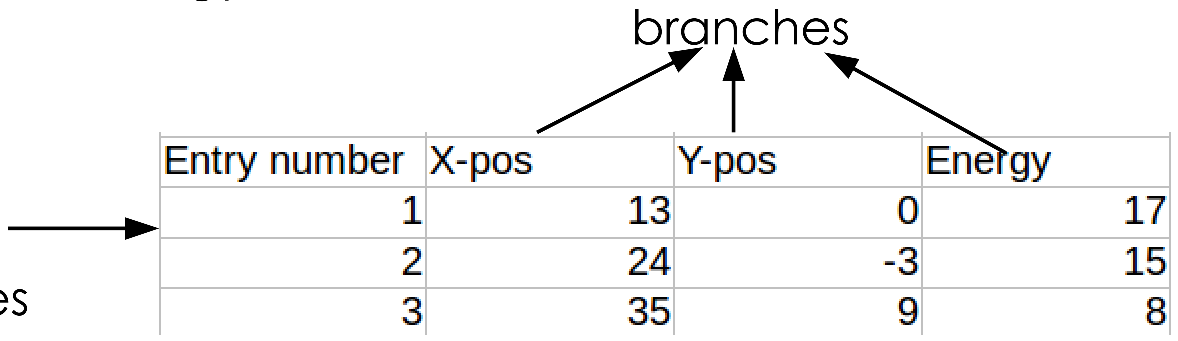
# What is a ROOT tree?

It's the way used by ROOT to organize information

Basically it's a big table

Each proton-proton collision is "labeled" by a number: 1,2,3......
For each collision many info are saved: x-position of the electron (positron), y-position, z-position, energy ecc...

A tree has several branches

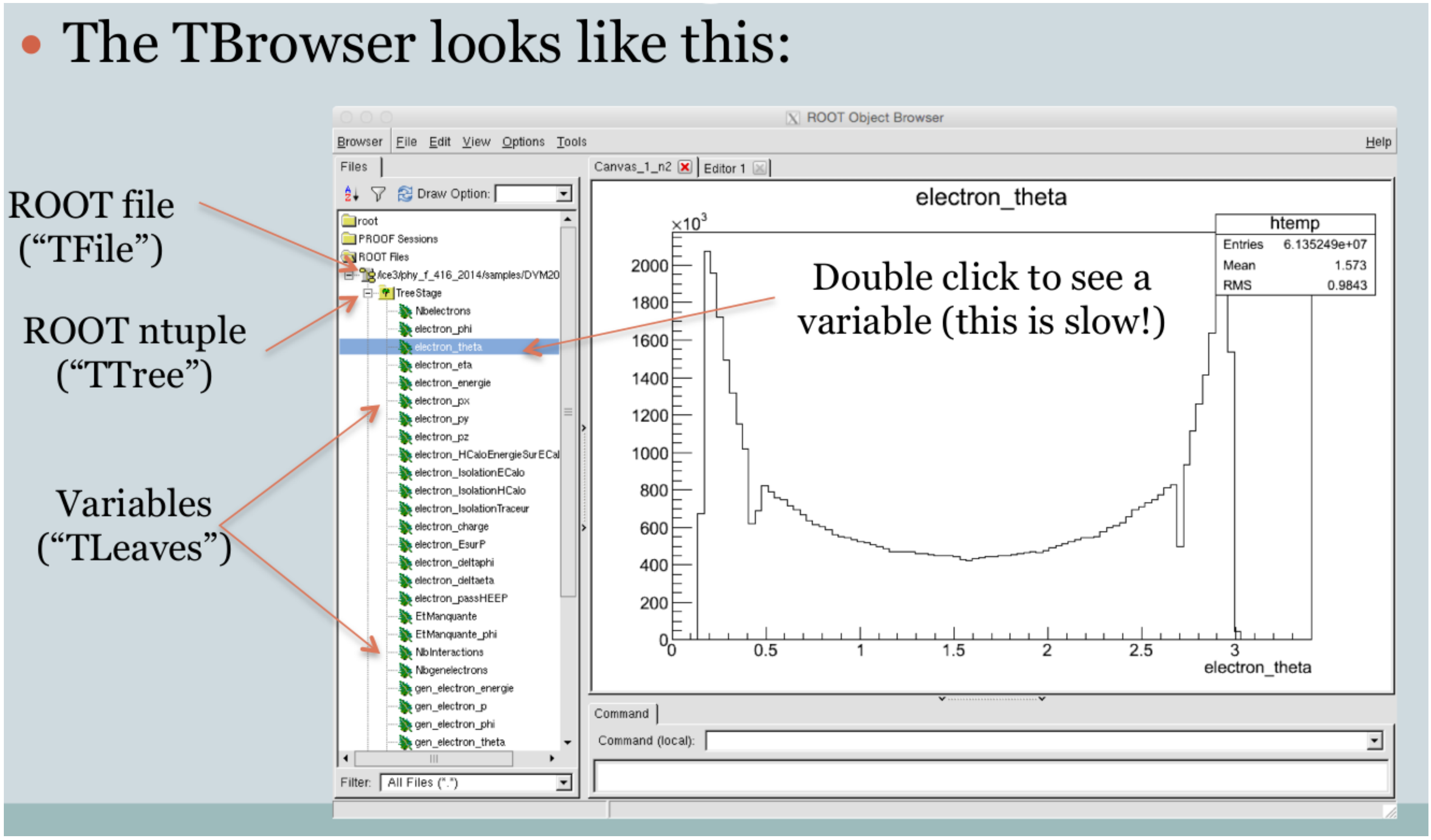


| Entry number | X-pos | Y-pos | Energy |
|---|---|---|---|
| 1 | 13 | 0 | 17 |
| 2 | 24 | -3 | 15 |
| 3 | 35 | 9 | 8 |

Usually the ROOT trees (TTRee) are saved in files:
today we are working with this file: a simulation of the Drell-Yan process
/ice3/phy_f_416_2016/samples/DYM20.root    (This is the complete path)
which hosts a TTree named **TreeStage**

# Visualizing a TTree using TBrowser

root -l
TFile f(“/ice3/phy_f_416_2016/samples/DYM20.root”, “READ”);
TBrowser my_Tbrowser;
The last command opened a window: you can browse the data now
To quit root:  use .q



- The TBrowser looks like this:

ROOT file (“TFile”)

ROOT ntuple (“TTree”)

Variables (“TLeaves”)

Double click to see a variable (this is slow!)

```
root
PROOF Sessions
ROOT Files
  /ice3/phy_f_416_2014/samples/DYM20
    TreeStage
      Nbelectrons
      electron_phi
      electron_theta
      electron_eta
      electron_energie
      electron_px
      electron_py
      electron_pz
      electron_HCaloEnergieSurECal
      electron_IsolationECalo
      electron_IsolationHCalo
      electron_IsolationTraceur
      electron_charge
      electron_EsurP
      electron_deltaphi
      electron_deltaeta
      electron_passHEEP
      EtManquante
      EtManquante_phi
      NbInteractions
      Nbgenelectrons
      gen_electron_energie
      gen_electron_p
      gen_electron_phi
      gen_electron_theta
```

| htemp | |
|---|---|
| Entries | 6.135249e+07 |
| Mean | 1.573 |
| RMS | 0.9843 |

## Work with trees in C++ objects

**Check** where you are under the CMSSW_8_0_8 directory

**Copy** my skeleton of an object called Analizer

```
cp /ice3/phy_f_416_2016/Lesson_1/Analyzer.h  .
cp /ice3/phy_f_416_2016/Lesson_1/Analyzer.C .
```

Open Analyzer.h → you will see that the declaration of the branches are already there: so you can skip the boring parts

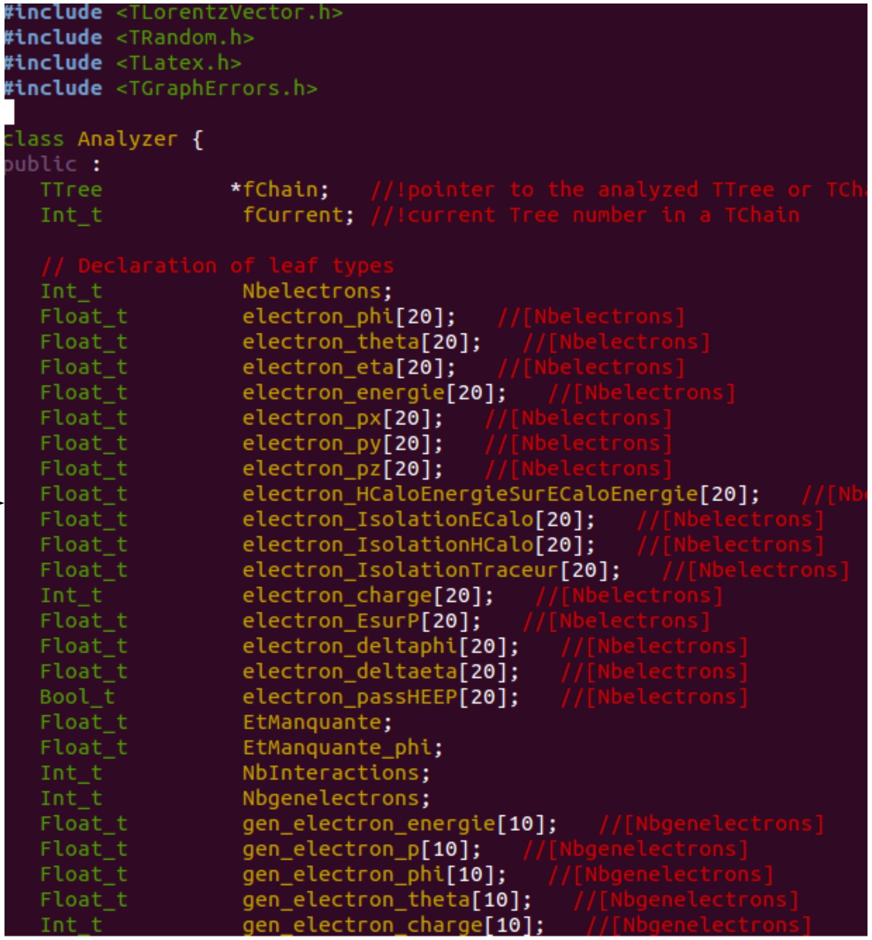
There is already a method declared in Analyzer.h called **Loop**

Have a look at the definition of this method **Loop** in Analyzer.C

Names of the branches you will use:
They should be clear enough.

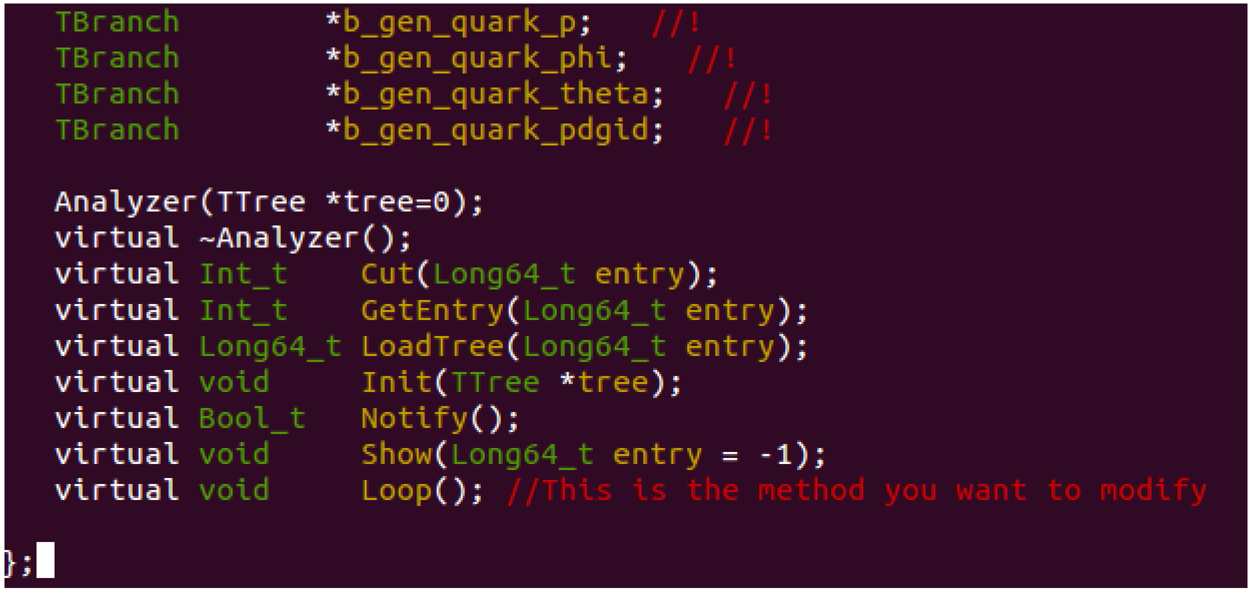
Note that there are both **generated quantities: gen_electron_energie** and the corresponding **reconstructed quantities: electron_energie**

**What kind of objects are they?**



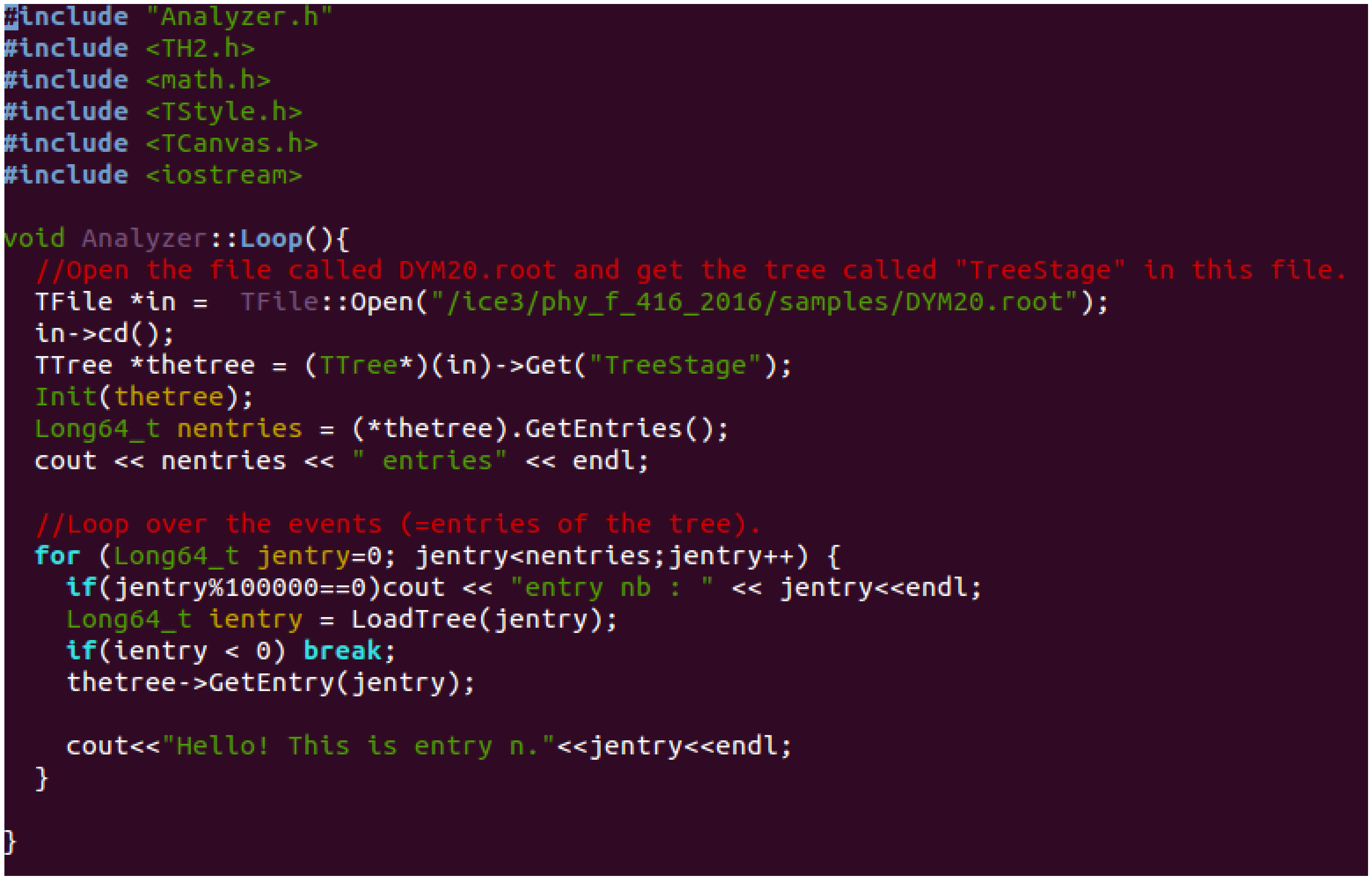
```
#include <TLorentzVector.h>
#include <TRandom.h>
#include <TLatex.h>
#include <TGraphErrors.h>

class Analyzer {
public :
   TTree          *fChain;   //!pointer to the analyzed TTree or TCh
   Int_t           fCurrent; //!current Tree number in a TChain

   // Declaration of leaf types
   Int_t           Nbelectrons;
   Float_t         electron_phi[20];   //[Nbelectrons]
   Float_t         electron_theta[20];   //[Nbelectrons]
   Float_t         electron_eta[20];   //[Nbelectrons]
   Float_t         electron_energie[20];   //[Nbelectrons]
   Float_t         electron_px[20];   //[Nbelectrons]
   Float_t         electron_py[20];   //[Nbelectrons]
   Float_t         electron_pz[20];   //[Nbelectrons]
   Float_t         electron_HCaloEnergieSurECaloEnergie[20];   //[Nb
   Float_t         electron_IsolationECalo[20];   //[Nbelectrons]
   Float_t         electron_IsolationHCalo[20];   //[Nbelectrons]
   Float_t         electron_IsolationTraceur[20];   //[Nbelectrons]
   Int_t           electron_charge[20];   //[Nbelectrons]
   Float_t         electron_EsurP[20];   //[Nbelectrons]
   Float_t         electron_deltaphi[20];   //[Nbelectrons]
   Float_t         electron_deltaeta[20];   //[Nbelectrons]
   Bool_t          electron_passHEEP[20];   //[Nbelectrons]
   Float_t         EtManquante;
   Float_t         EtManquante_phi;
   Int_t           NbInteractions;
   Int_t           Nbgenelectrons;
   Float_t         gen_electron_energie[10];   //[Nbgenelectrons]
   Float_t         gen_electron_p[10];   //[Nbgenelectrons]
   Float_t         gen_electron_phi[10];   //[Nbgenelectrons]
   Float_t         gen_electron_theta[10];   //[Nbgenelectrons]
   Int_t           gen_electron_charge[10];   //[Nbgenelectrons]
```

## Analyzer.h



```
   TBranch        *b_gen_quark_p;   //!
   TBranch        *b_gen_quark_phi;   //!
   TBranch        *b_gen_quark_theta;   //!
   TBranch        *b_gen_quark_pdgid;   //!

   Analyzer(TTree *tree=0);
   virtual ~Analyzer();
   virtual Int_t    Cut(Long64_t entry);
   virtual Int_t    GetEntry(Long64_t entry);
   virtual Long64_t LoadTree(Long64_t entry);
   virtual void     Init(TTree *tree);
   virtual Bool_t   Notify();
   virtual void     Show(Long64_t entry = -1);
   virtual void     Loop(); //This is the method you want to modify

};
```

## Analyzer.C



```
#include "Analyzer.h"
#include <TH2.h>
#include <math.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <iostream>

void Analyzer::Loop(){
  //Open the file called DYM20.root and get the tree called "TreeStage" in this file.
  TFile *in =  TFile::Open("/ice3/phy_f_416_2016/samples/DYM20.root");
  in->cd();
  TTree *thetree = (TTree*)(in)->Get("TreeStage");
  Init(thetree);
  Long64_t nentries = (*thetree).GetEntries();
  cout << nentries << " entries" << endl;

  //Loop over the events (=entries of the tree).
  for (Long64_t jentry=0; jentry<nentries;jentry++) {
    if(jentry%100000==0)cout << "entry nb : " << jentry<<endl;
    Long64_t ientry = LoadTree(jentry);
    if(ientry < 0) break;
    thetree->GetEntry(jentry);

    cout<<"Hello! This is entry n."<<jentry<<endl;
  }

}
```

So: what is the Loop method doing?
Modify it to print the generated energy of the first electron

## histograms

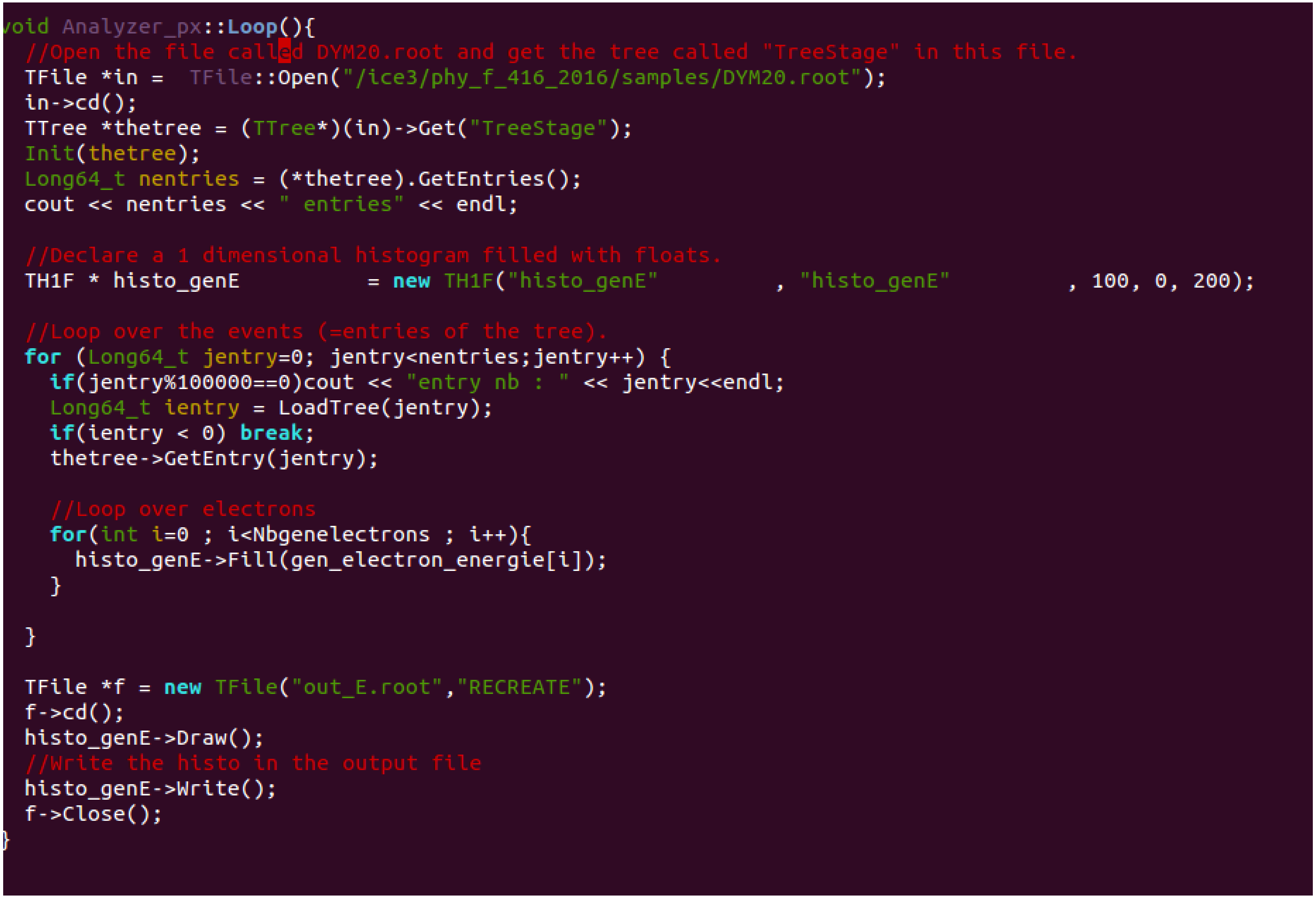
Just printing the value of a variable is a bad idea! You usually want to make histograms of the variables' distribution. Root provides you a solution for this: the so called TH1F (histogram in ROOT)

```
cp /ice3/phy_f_416_2016/Lesson_1/Analyzer_px.h  .
cp /ice3/phy_f_416_2016/Lesson_1/Analyzer_px.C .
```

Analyzer_px is identycal to Analyzer, I just modified the definition of the Loop method to fill a histogram.

Open Analyzer_px.C and have a look at it

# histograms

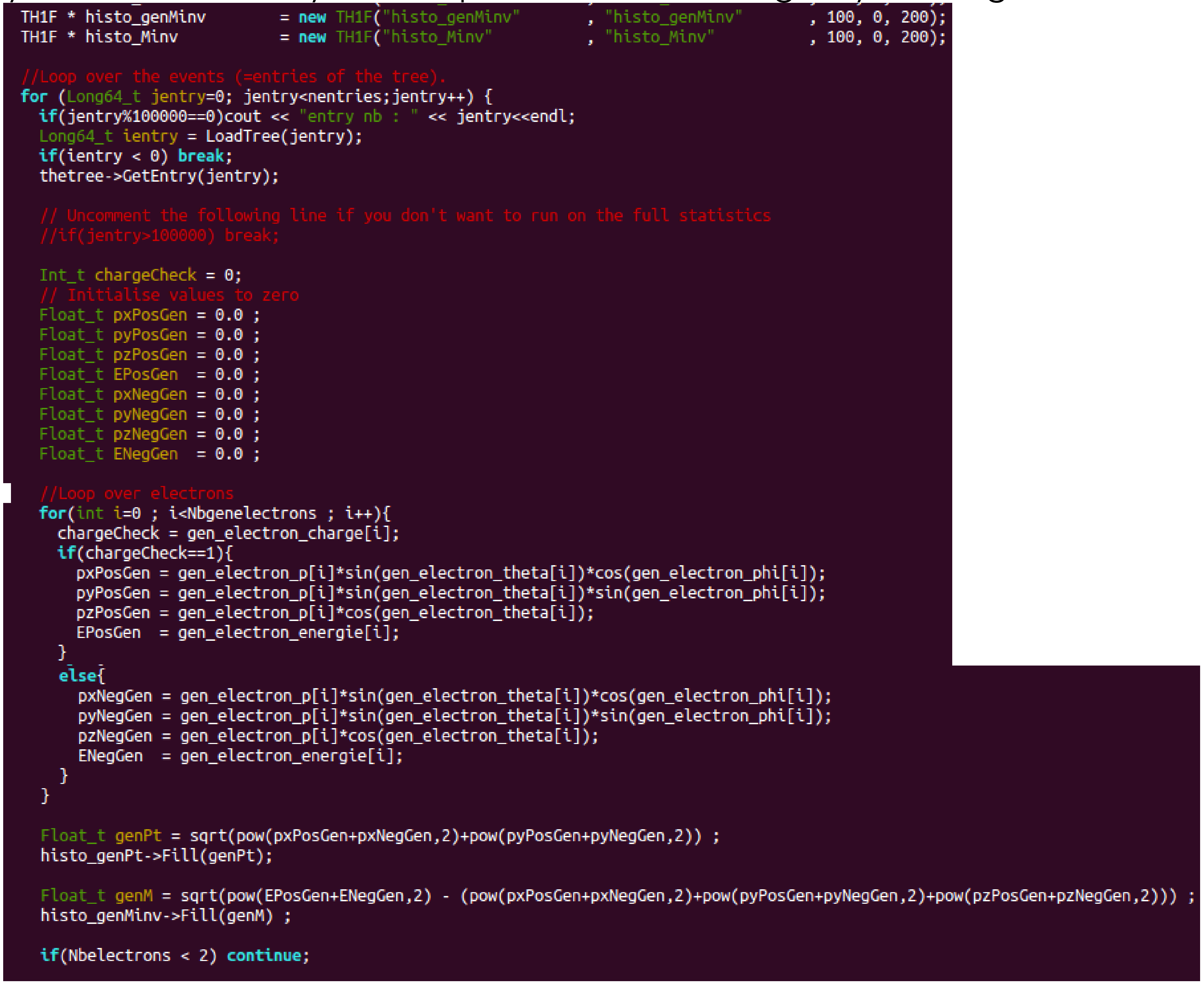


```
void Analyzer_px::Loop(){
  //Open the file called DYM20.root and get the tree called "TreeStage" in this file.
  TFile *in =  TFile::Open("/ice3/phy_f_416_2016/samples/DYM20.root");
  in->cd();
  TTree *thetree = (TTree*)(in)->Get("TreeStage");
  Init(thetree);
  Long64_t nentries = (*thetree).GetEntries();
  cout << nentries << " entries" << endl;

  //Declare a 1 dimensional histogram filled with floats.
  TH1F * histo_genE          = new TH1F("histo_genE"          , "histo_genE"          , 100, 0, 200);

  //Loop over the events (=entries of the tree).
  for (Long64_t jentry=0; jentry<nentries;jentry++) {
    if(jentry%100000==0)cout << "entry nb : " << jentry<<endl;
    Long64_t ientry = LoadTree(jentry);
    if(ientry < 0) break;
    thetree->GetEntry(jentry);

    //Loop over electrons
    for(int i=0 ; i<Nbgenelectrons ; i++){
      histo_genE->Fill(gen_electron_energie[i]);
    }

  }

  TFile *f = new TFile("out_E.root","RECREATE");
  f->cd();
  histo_genE->Draw();
  //Write the histo in the output file
  histo_genE->Write();
  f->Close();
}
```

# To do: InvMass histograms (Gen and Reco)

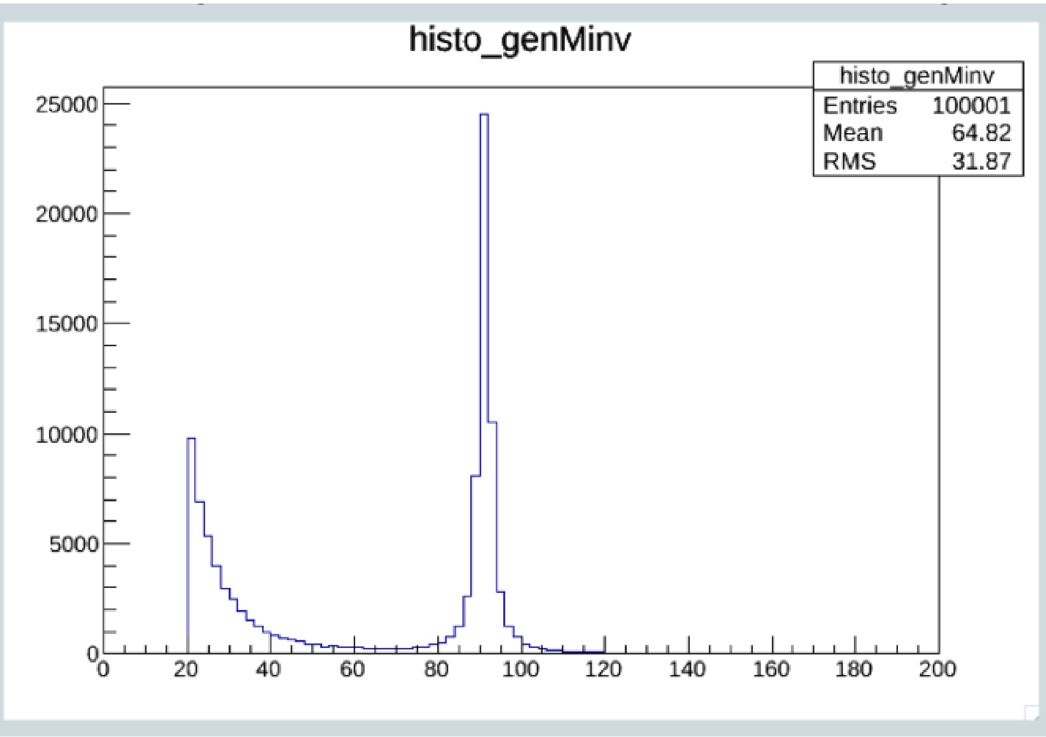
Again: you have to modify the Loop method.  Let me give you a big hint:

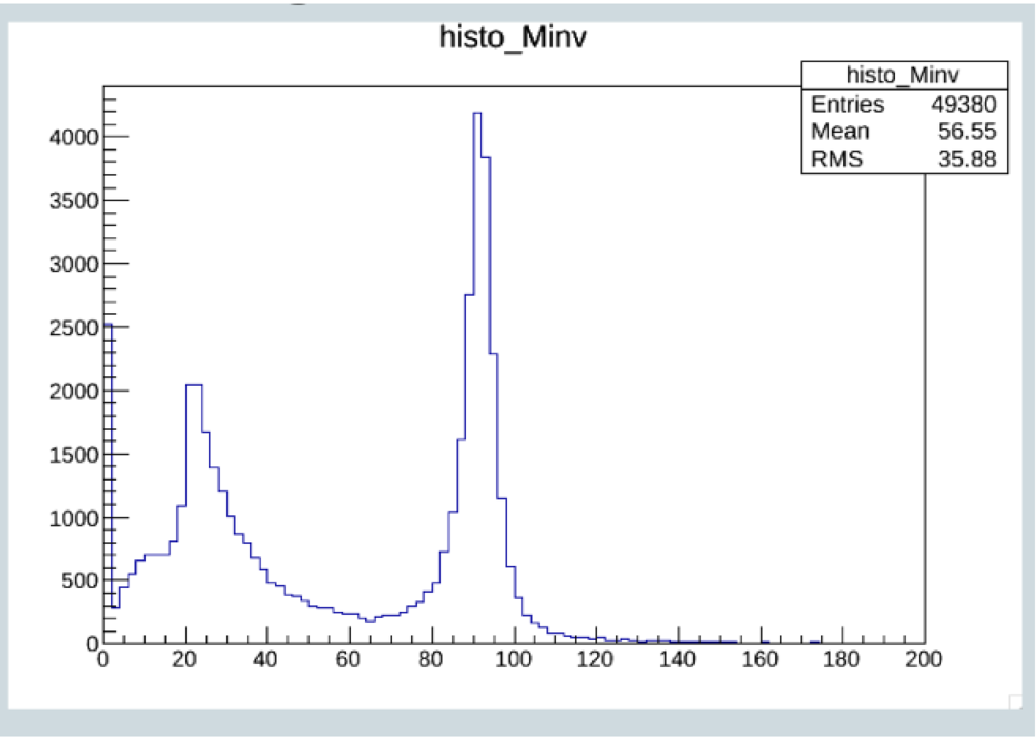
You have
20 minutes
to complete
 it with Reco



```
TH1F * histo_genMinv        = new TH1F("histo_genMinv"       , "histo_genMinv"       , 100, 0, 200);
TH1F * histo_Minv           = new TH1F("histo_Minv"          , "histo_Minv"          , 100, 0, 200);

//Loop over the events (=entries of the tree).
for (Long64_t jentry=0; jentry<nentries;jentry++) {
  if(jentry%100000==0)cout << "entry nb : " << jentry<<endl;
  Long64_t ientry = LoadTree(jentry);
  if(ientry < 0) break;
  thetree->GetEntry(jentry);

  // Uncomment the following line if you don't want to run on the full statistics
  //if(jentry>100000) break;

  Int_t chargeCheck = 0;
  // Initialise values to zero
  Float_t pxPosGen = 0.0 ;
  Float_t pyPosGen = 0.0 ;
  Float_t pzPosGen = 0.0 ;
  Float_t EPosGen  = 0.0 ;
  Float_t pxNegGen = 0.0 ;
  Float_t pyNegGen = 0.0 ;
  Float_t pzNegGen = 0.0 ;
  Float_t ENegGen  = 0.0 ;

  //Loop over electrons
  for(int i=0 ; i<Nbgenelectrons ; i++){
    chargeCheck = gen_electron_charge[i];
    if(chargeCheck==1){
      pxPosGen = gen_electron_p[i]*sin(gen_electron_theta[i])*cos(gen_electron_phi[i]);
      pyPosGen = gen_electron_p[i]*sin(gen_electron_theta[i])*sin(gen_electron_phi[i]);
      pzPosGen = gen_electron_p[i]*cos(gen_electron_theta[i]);
      EPosGen  = gen_electron_energie[i];
    }
    else{
      pxNegGen = gen_electron_p[i]*sin(gen_electron_theta[i])*cos(gen_electron_phi[i]);
      pyNegGen = gen_electron_p[i]*sin(gen_electron_theta[i])*sin(gen_electron_phi[i]);
      pzNegGen = gen_electron_p[i]*cos(gen_electron_theta[i]);
      ENegGen  = gen_electron_energie[i];
    }
  }

  Float_t genPt = sqrt(pow(pxPosGen+pxNegGen,2)+pow(pyPosGen+pyNegGen,2)) ;
  histo_genPt->Fill(genPt);

  Float_t genM = sqrt(pow(EPosGen+ENegGen,2) - (pow(pxPosGen+pxNegGen,2)+pow(pyPosGen+pyNegGen,2)+pow(pzPosGen+pzNegGen,2))) ;
  histo_genMinv->Fill(genM) ;

  if(Nbelectrons < 2) continue;
```

# Invartiant mass histogram





If you wrote your code correctly you should see something like these two plots

**These plots will be part of your final report:**
Why gen and reco are different?
Why the gen histogram does not start at zero?

## Check with my implementation:
## InvMass histograms
## Generated and reconstructed

This is how I did it:

```
cp /ice3/phy_f_416_2016/Lesson_1/Analyzer_Mass.h  .
cp /ice3/phy_f_416_2016/Lesson_1/Analyzer_Mass.C .
```

Compile Analyzer_Mass  and run the Loop method, so:

```
root -l
.L Analyzer_Mass.C++
Analyzer_Mass John
John.Loop()
```

$A_{FB}$ depends on s' (the center-of-mass energy of the initial state quarks)

- How is s' related to the invariant mass of the electrons in the final state?

## Homework

# R calculation

Have your mass plots ready (generated vs reconstructed)

Use the R definition in slide 7

R is a complex number: try to write down explicitely Re(R) and |R|

You'll see that they both depend on s'=center-of-mass energy of initial state involving the quarks

How is s' related to the invariant mass of the electrons in the final state?

Think about how to write a method in your Analyzer.C to compute Re(R) and |R| (We'll start from this next time)