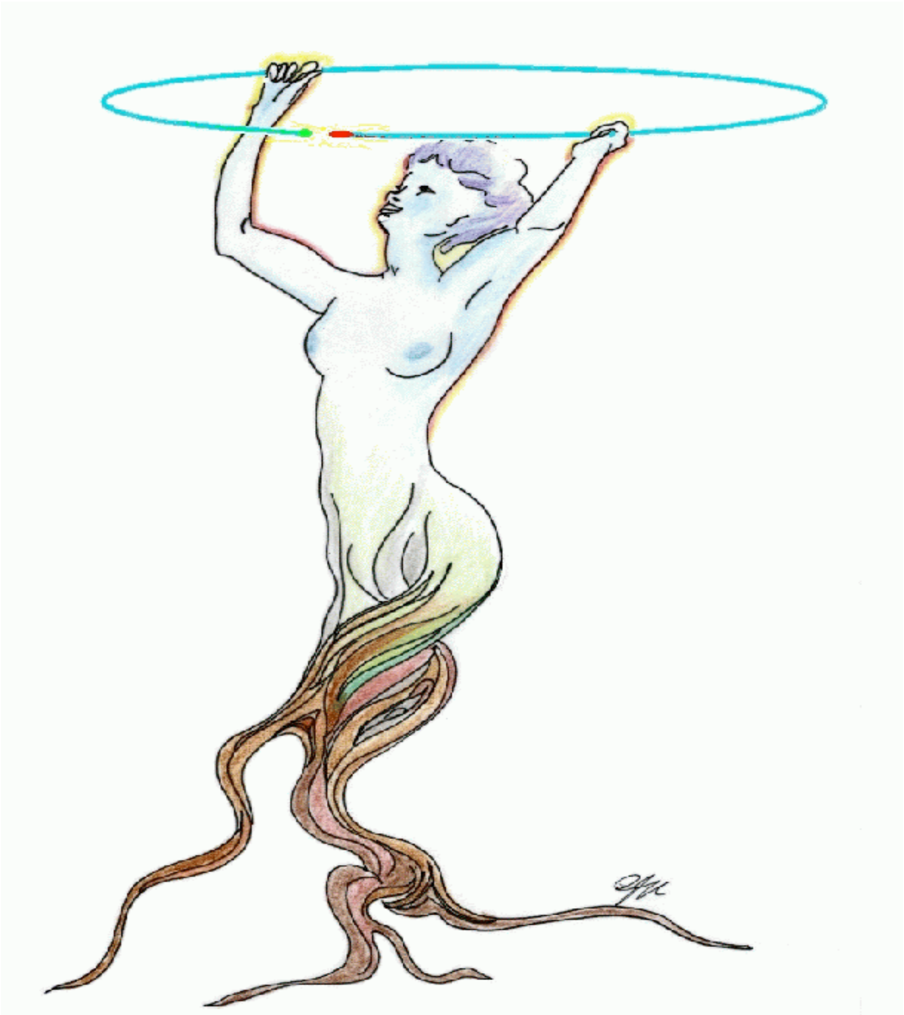
ROOT for beginners

Fifth Day

Advanced use



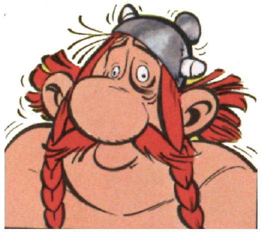
The best for the end

- Use of a TSelector

- Use of a class in a TTree

- Polymorphism

- Adding a class to ROOT



Trees: they are classy!

Let us climb back up the tree

- We have seen yesterday how to open and manipulate a tree with the TreeViewer and the command line (Draw, Scan, SetAlias…)

```
root[0] TFile *f=new TFile("tree_struc.root")
root[1] f->ls()
TFile**         tree_struc.root
 TFile*         tree_struc.root
  KEY: TTree    t;1     TTree avec une structure
root[2] TTree *a=(TTree *)f->Get("t")
root[3] a->StartViewer()
root[4] a->Draw("M_part")
```

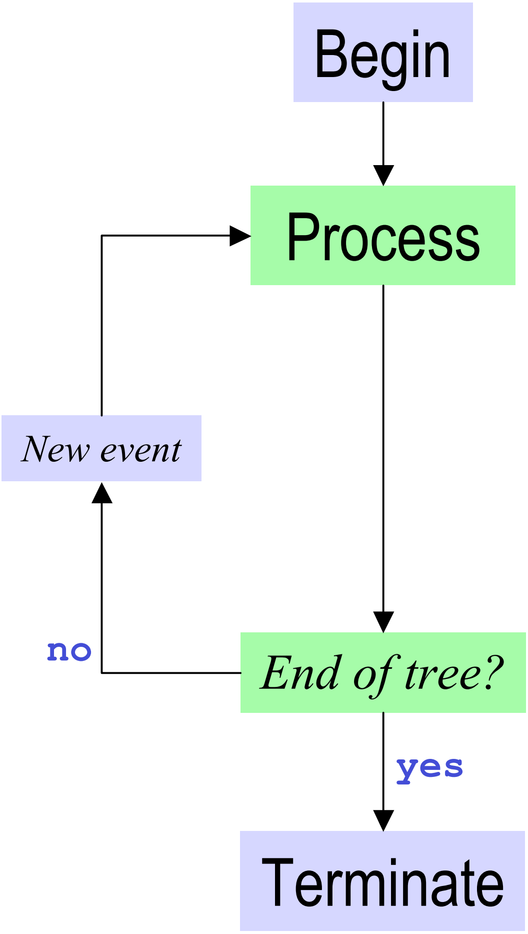
Are we happy with that?

- Advantage : it is very easy.
- Drawback : histograms are built one by one.
- For more complex treatments:
  - Use of an analysis class

```
root[15] a→MakeSelector("MonSelecteur")
Info in <TTreePlayer::MakeClass>: Files: MonSelecteur.h
and MonSelecteur.C generated from Tree: t
```

Use of a TSelector

- Only 3 methods have to be defined
  - Begin : initialisations (histograms, global variables, etc...)
  - Process : event selection and treatment
  - Terminate : end of the analysis (global calculations, write results in a file, etc…)

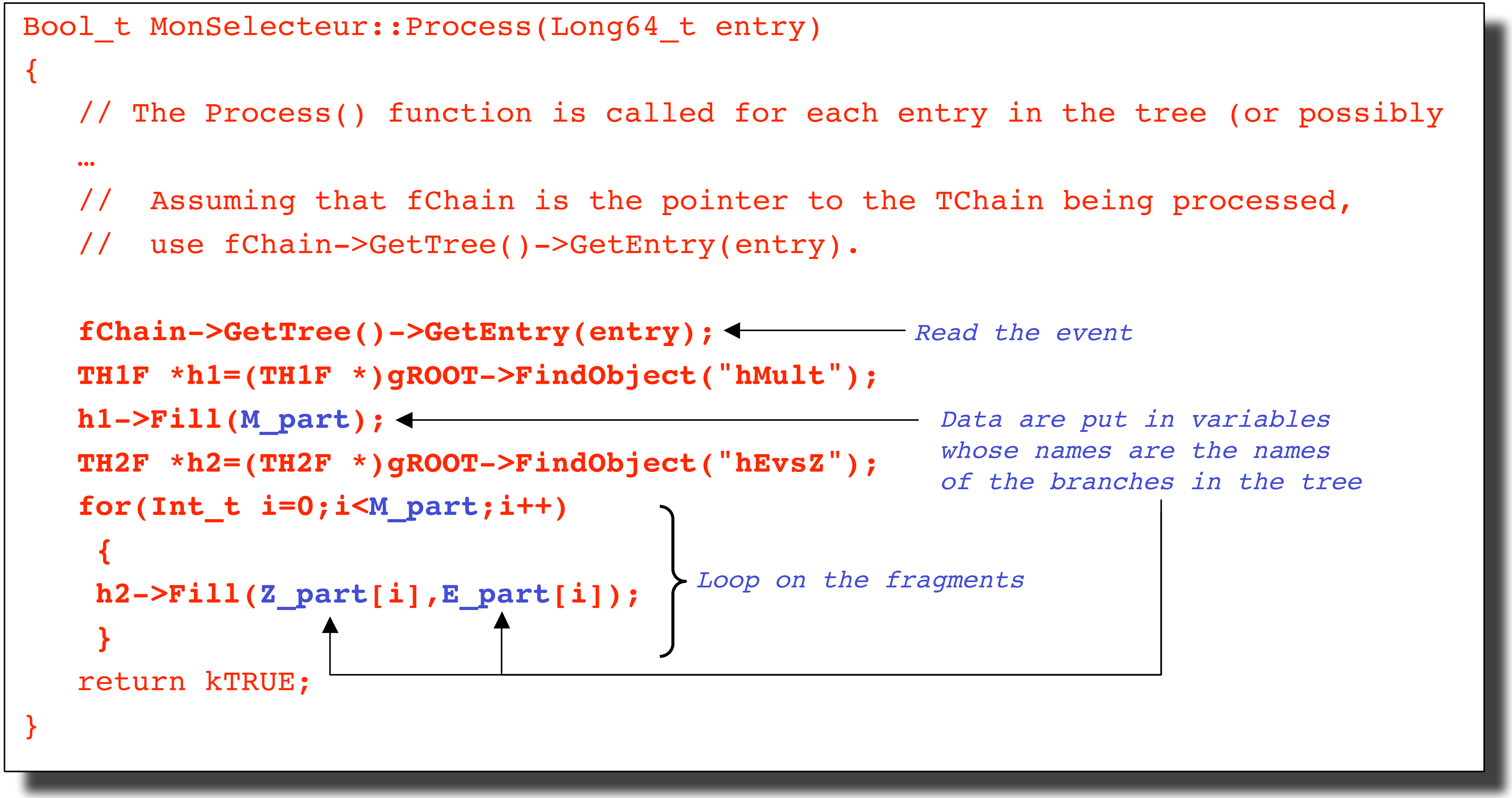


My first Begin

(Edit the file MonSelecteur.C)

```
#include "MonSelecteur.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void MonSelecteur::Begin(TTree *tree)
{
   // The Begin() function is called at the start of the query.
   // When running with PROOF Begin() is only called on the client.
   // The tree argument is deprecated (on PROOF 0 is passed).

   TString option = GetOption();

   TH1F *h1=new TH1F("hMult","Multiplicity",40,-0.5,39.5);
   TH2F *h2=new TH2F("hEvsZ","Energy vs Z",60,-0.5,59.5,40,0,2400);

}
```

My first Process



```
Bool_t MonSelecteur::Process(Long64_t entry)
{
   // The Process() function is called for each entry in the tree (or possibly
   …
   //  Assuming that fChain is the pointer to the TChain being processed,
   //  use fChain->GetTree()->GetEntry(entry).

   fChain->GetTree()->GetEntry(entry);
   TH1F *h1=(TH1F *)gROOT->FindObject("hMult");
   h1->Fill(M_part);
   TH2F *h2=(TH2F *)gROOT->FindObject("hEvsZ");
   for(Int_t i=0;i<M_part;i++)
    {
    h2->Fill(Z_part[i],E_part[i]);
    }
   return kTRUE;
}
```

Read the event

Data are put in variables whose names are the names of the branches in the tree

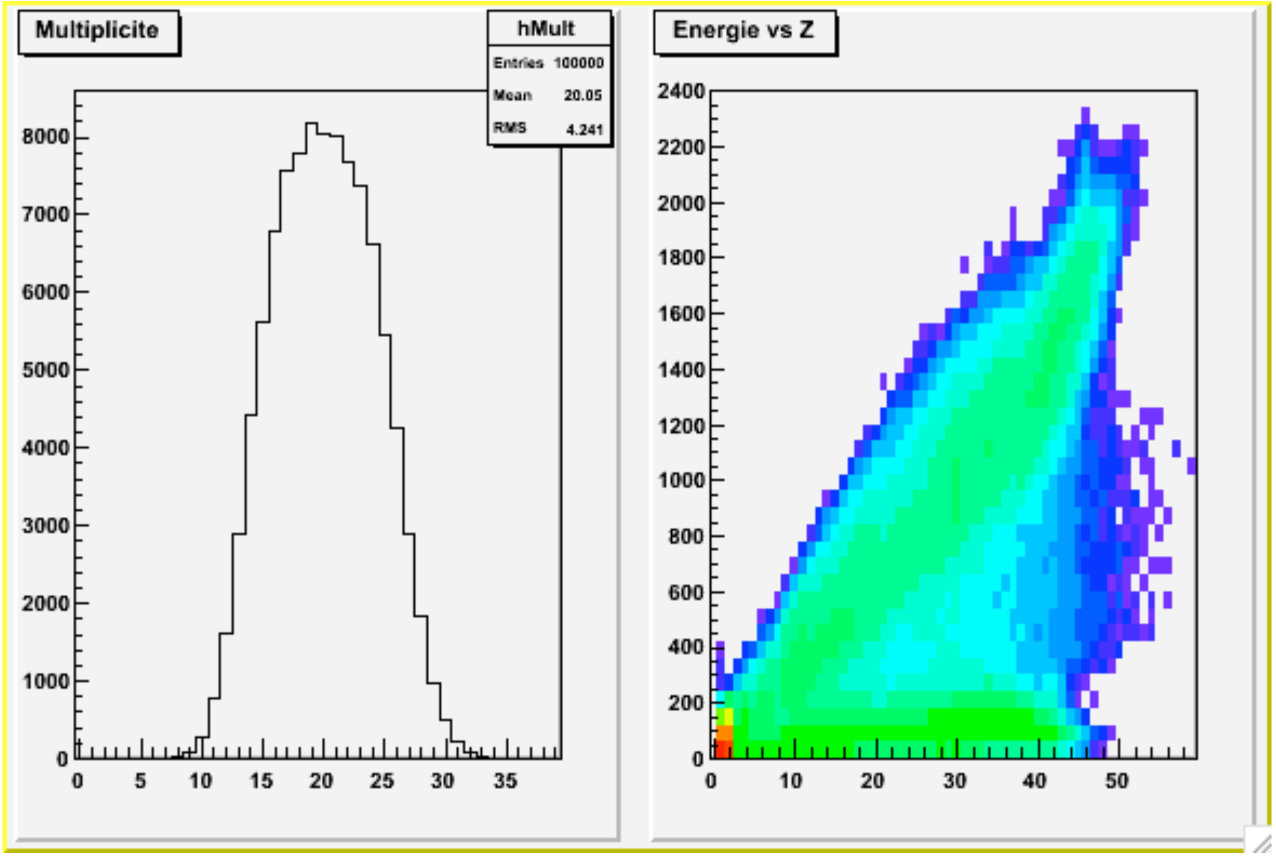
Loop on the fragments

My first Terminate

```
void MonSelecteur::Terminate()
{
   // The Terminate() function is the last function to be called during
   // a query. It always runs on the client, it can be used to present
   // the results graphically or save the results to file.

   TCanvas *c=new TCanvas("CanSelecteur","MonSelecteur");
   c->Divide(2,1);  ←——————  1 line, 2 columns
   c->cd(1);
   gROOT->FindObject("hMult")->Draw();
   c->cd(2);
   TH2F *h2=(TH2F *)gROOT->FindObject("hEvsZ");
   h2->SetStats(kFALSE);  ←——————  No statistics box for the 2D
   h2->Draw("col");
   gPad->SetLogz(kTRUE);  ←——————  Logarithmic scale for the Z axis
   c->Update();

}
```

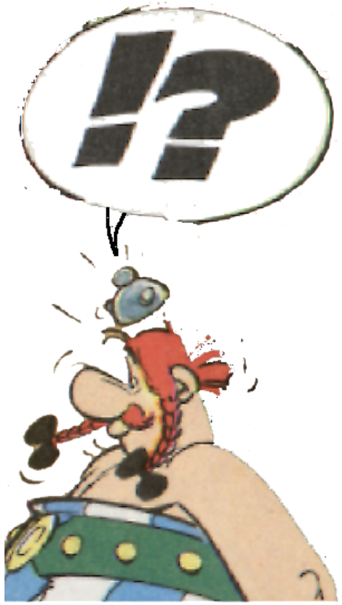
Execution (without guillotine)!

```
root[19] a->Process("MonSelecteur.C+")
Info in <TUnixSystem::ACLiC>: creating shared library ./MonSelecteur_C.so
Class MonSelecteur: Streamer() not declared
Class MonSelecteur: ShowMembers() not declared
```



```
hMult
Entries  100000
Mean      20.05
RMS       4.241
```

Use a class in a tree

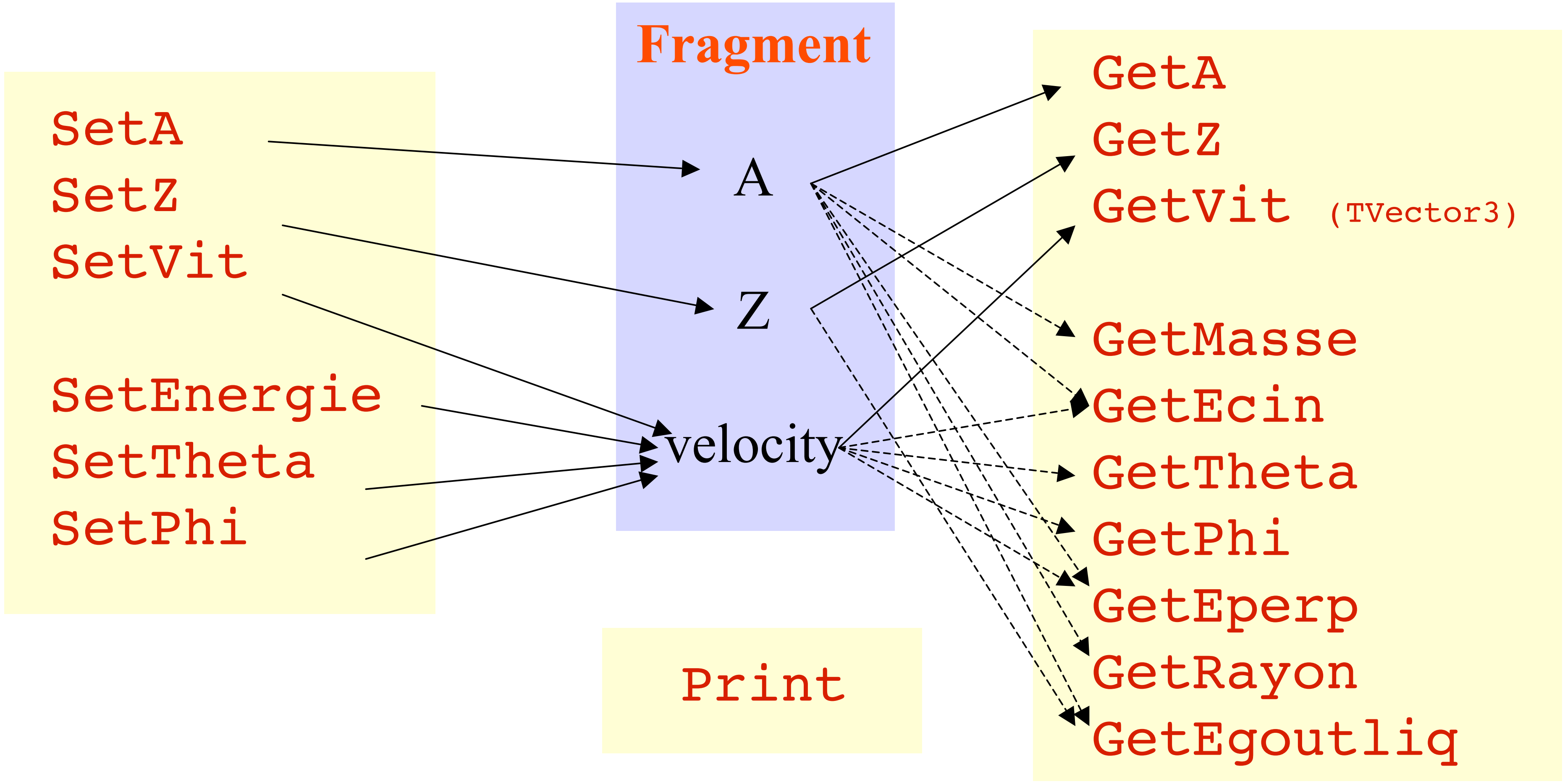


A more complex tree: using classes

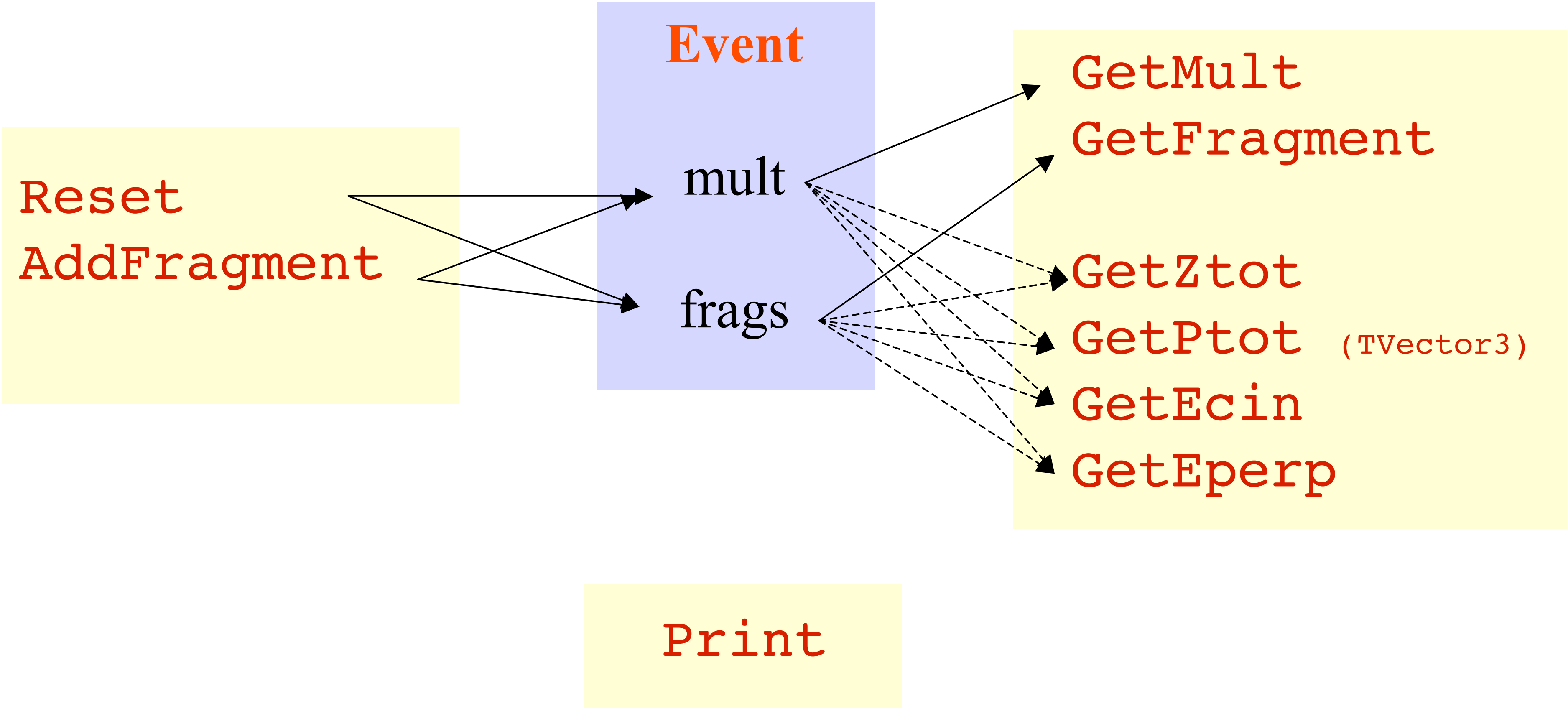
- We will use 2 classes
  - a class named **`Fragment`** which will contain the information corresponding to 1 particle (files **`Fragment.h`** et **`Fragment.C`**)
  - a class named **`Event`** which will contain an array of particles and global information about the event (files **`Event.h`** et **`Event.C`**)

`http://caeinfo.in2p3.fr/root/Formation/en/Day5/Fragment.*`
`http://caeinfo.in2p3.fr/root/Formation/en/Day5/Event.*`

The Fragment class



The Event class



The tree declaration

- The file will contain 1 TTree:
  - Arbre: physical events

```
TTree *mt=new TTree("Arbre","Evenements")
Event *evt=new Event()
mt→Branch("Data","Event",&evt,64000,0)
```

The events to write will be stored in the object of type **`Event`** pointed by **`evt`**.

Beware!

When using a class, the declaration of a branch is different from the declaration of a branch with a "simple" variable.

To fill it

(See the file FillArbre.C)

http://caeinfo.in2p3.fr/root/Formation/en/Day5/FillArbre.C

```
Fragment *frag=new Fragment();
while(ok) {
evt→Reset();
frag→SetA(4); frag→SetZ(2);
frag→SetEnergie(40);
frag→SetTheta(8);
frag→SetPhi(44);
evt→AddFragment(frag);
frag→SetA(12); frag→SetZ(6);
frag→SetEnergie(22.3);
frag→SetTheta(4);
frag→SetPhi(256);
evt→AddFragment(frag);

…
mt→Fill();
}
…
```

Reset the whole event

Definition of a fragment

Add a fragment to the event

Definition of a fragment

Add a fragment to the event

Fill the TTree

To use it (part 1)

- First load class definition in ROOT

```
root[1] .L $ROOTSYS/lib/libPhysics.so
root[2] .L Fragment.C+
root[3] .L Event.C+
```

Necessary because the TVector3 class is used by the Fragment class

- Generate the HTML documentation

```
root[4] THtml *htm=new THtml()
root[5] htm->MakeClass("Fragment")
root[6] htm->MakeClass("Event")
```

Generates the HTML files in the htmldoc/ directory

The rootlogon.C file

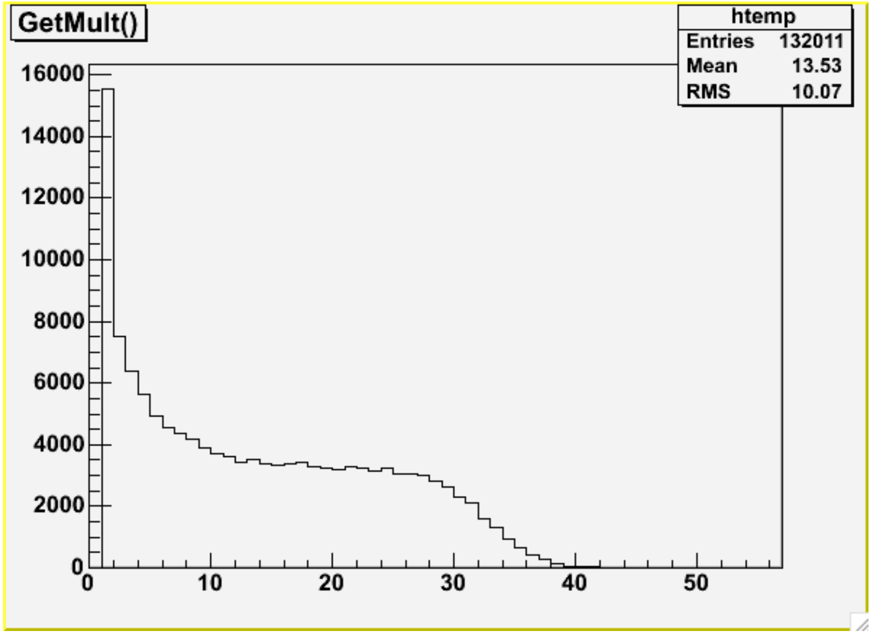
For lazy or clumsy persons only (i.e. almost everybody…)!

```
{
gStyle->SetPalette(1);
gROOT->ProcessLine(".L $ROOTSYS/lib/libPhysics.so");
gROOT->ProcessLine(".L Fragment.C+");
gROOT->ProcessLine(".L Event.C+");
TFile *fi=new TFile("indra_xesn50.root");
TTree *mt=(TTree *)fi->Get("Arbre");
Event *evt=new Event();
mt→SetBranchAddress("Data",&evt);
}
```

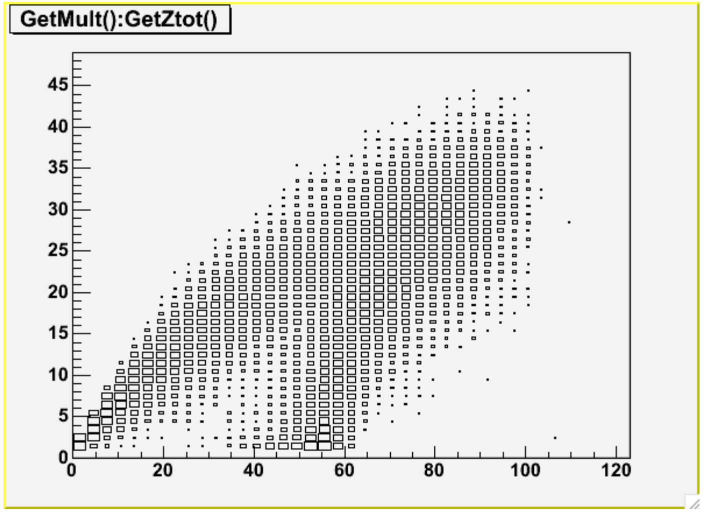
Read an event in the tree

```
root[8] mt→GetEntries() ←——————— Total number of events in the tree
1.3201100000000000e+05
root[9] mt→GetEntry(1567) ←——————— Read the entry 1567 in the tree
                                   (the 1568th event in the tree)
(Int_t)1089
root[10] evt→Print()    ←——————— Listing of the event
=====================================================
Mult : 21
1 ->   8,  4 :    -0.47    -0.22     2.27
2 ->   4,  2 :     0.97     2.43     8.58
3 ->   1,  1 :     0.22     4.84    12.56
4 ->   4,  2 :    -0.25    -2.33     8.96
…

root[11] evt→GetEperp()   ←——————— Transverse energy for this event
(double)2.899…………e+02
```

Building histograms (Step 1)



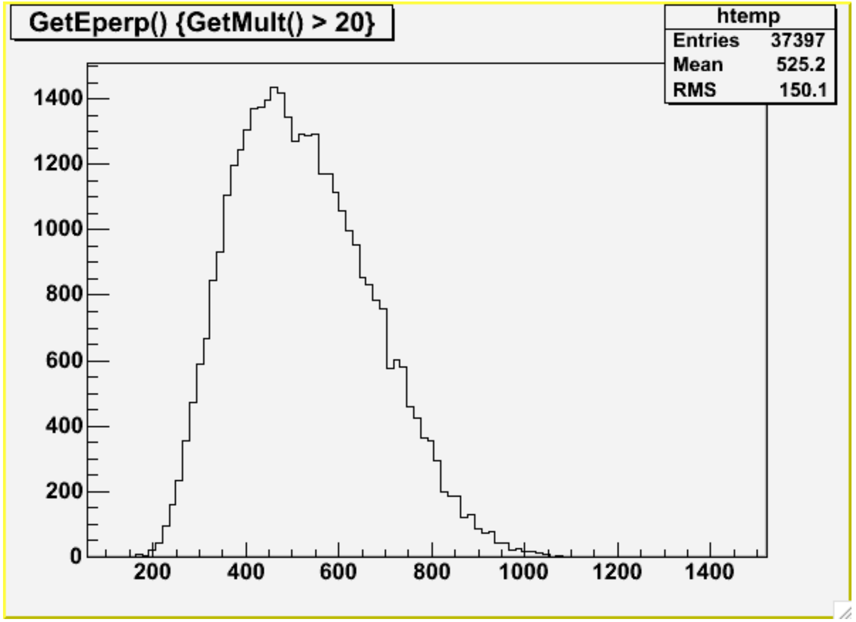
```
root[12] mt→Draw("GetMult()")
```

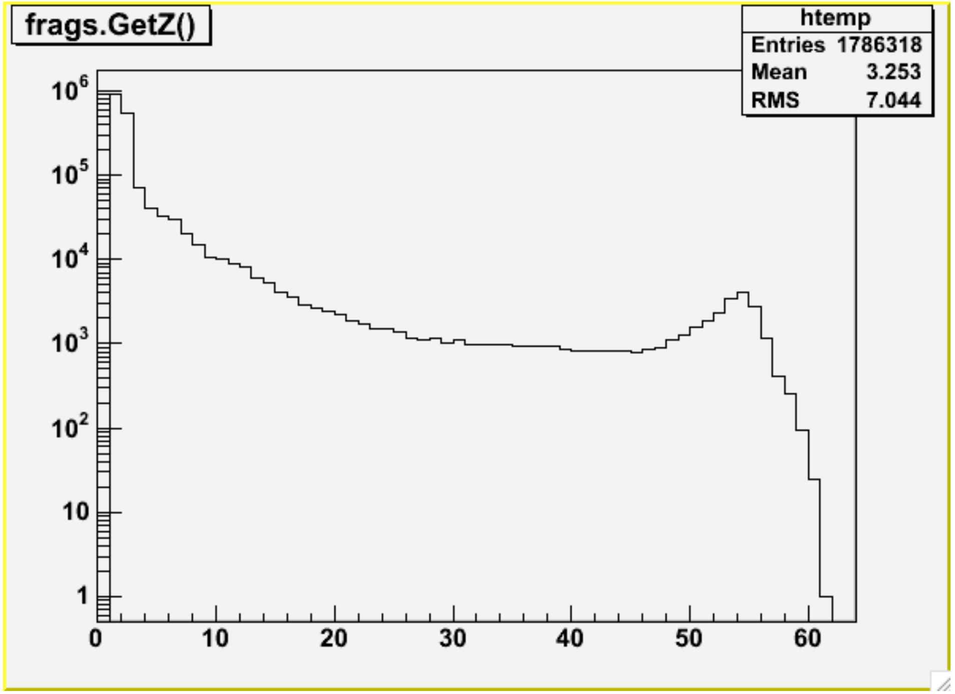


```
root[13] mt→Draw("GetMult():GetZtot()","","box")
```

Building histograms (Step 2)

```
root[14] mt→Draw("GetEperp()","GetMult()>20")
```

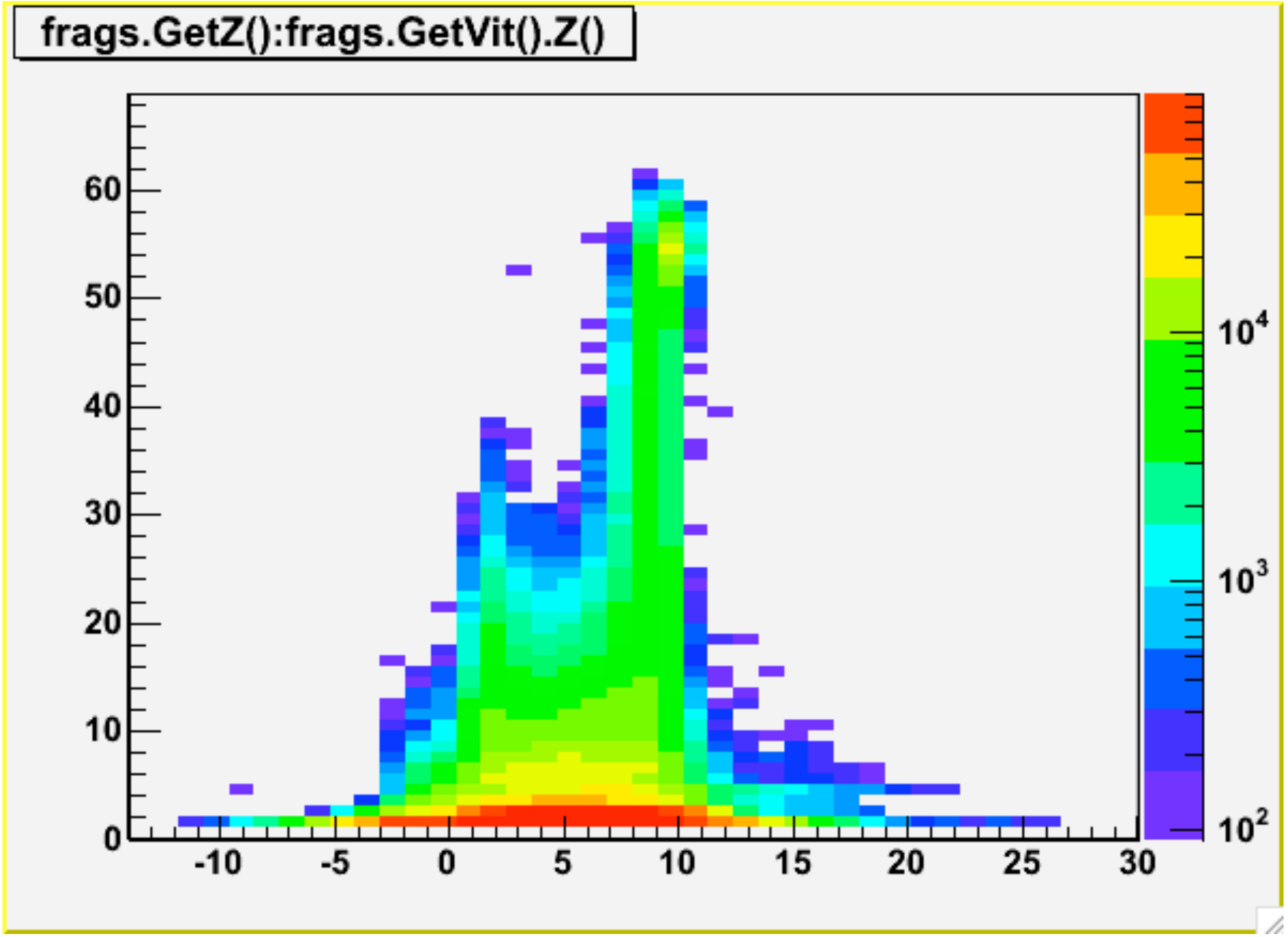




```
root[15] mt→Draw("frags.GetZ()")
```

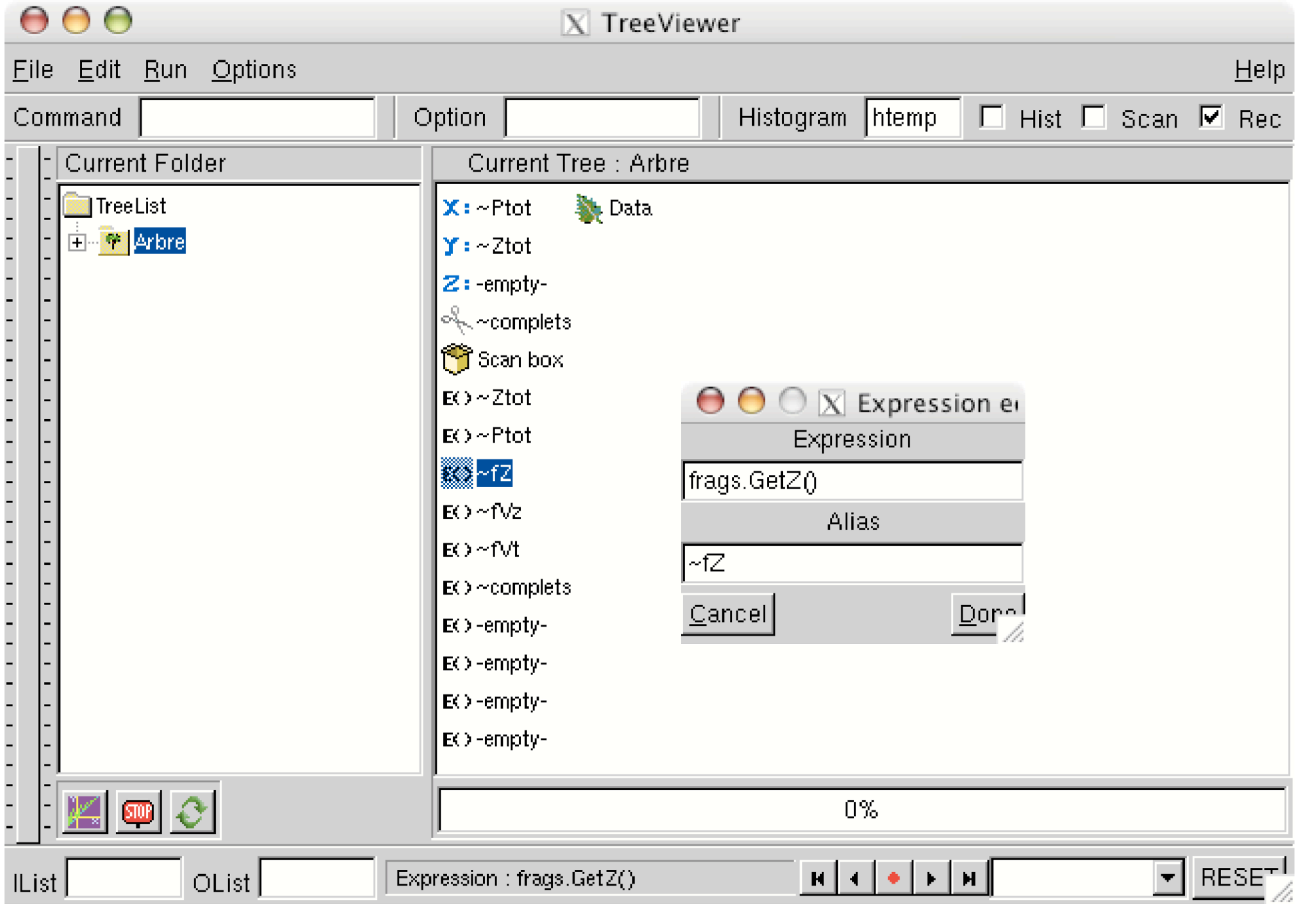
Building histograms (Step 3)

```
root[16] mt→Draw("frags.GetZ():frags.GetVit().Z()","","zcol")
```



Using the TTreeViewer

```
root[17] mt→StartViewer()
```



Command

Option

Histogram htemp

□ Hist □ Scan ☑ Rec

Current Tree : Arbre

X: ~Ptot
Y: ~Ztot
Z: -empty-
~complets
Scan box
E() ~Ztot
E() ~Ptot
E() ~fZ
E() ~fVz
E() ~fVt
E() ~complets
E() -empty-
E() -empty-
E() -empty-
E() -empty-

Expression
frags.GetZ()

Alias
~fZ

0%

Expression : frags.GetZ()

We can use cuts TCut and graphical cuts TCutG

Use of a TSelector

- Because it is a tree, a TSelector can be used:

```
root[18] mt→MakeSelector("MonAnalyse")
Info in <TTreePlayer::MakeClass>: Files:
  MonAnalyse.h
and MonAnanlyse.C generated from Tree: Arbre
```

My second Begin

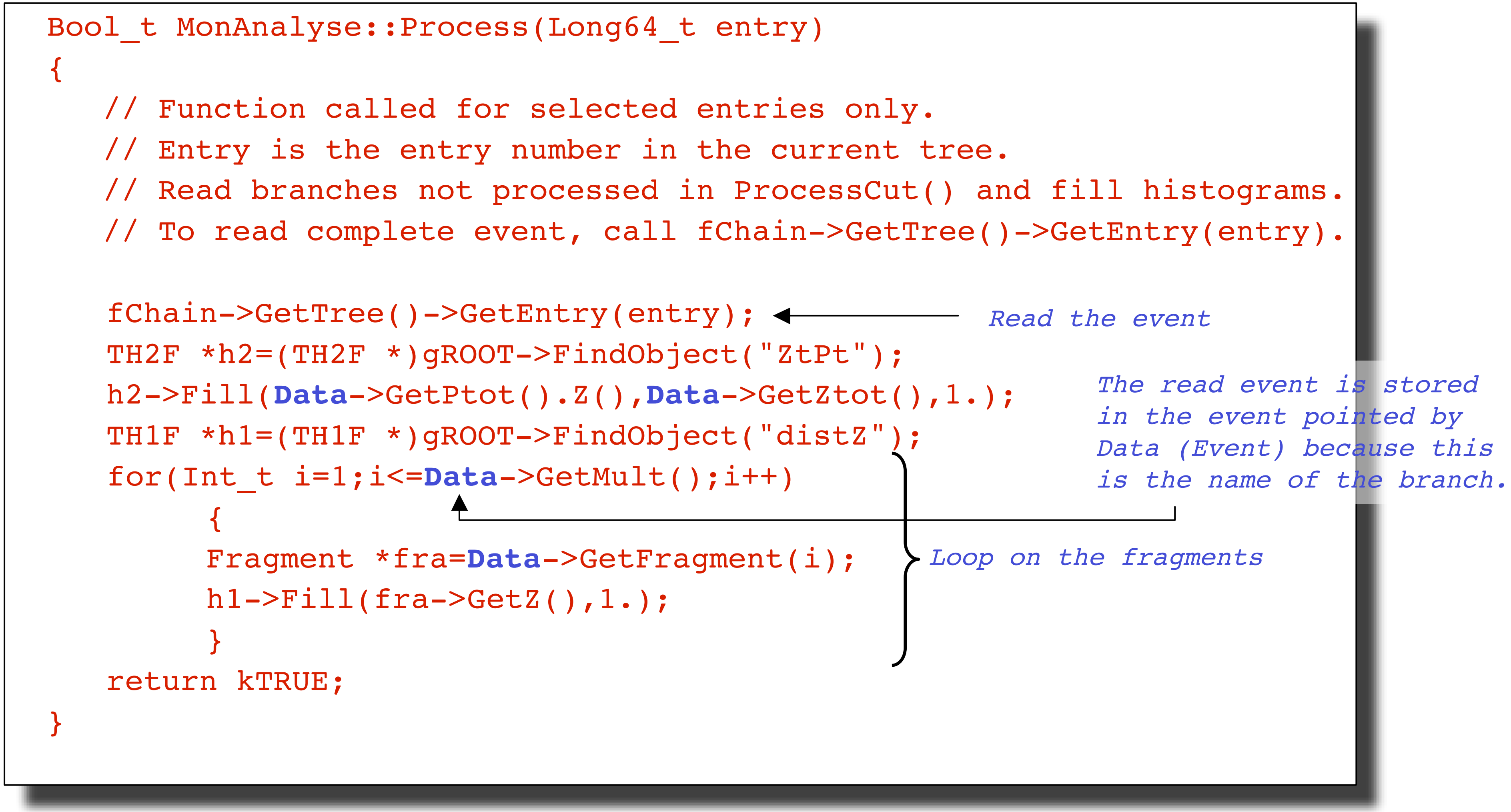
```
#include "Fragment.h"
#include "Event.h"
#include "MonAnalyse.h"
#include "TH2.h"
#include "TStyle.h"
#include "TCanvas.h"

void MonAnalyse::Begin(TTree *tree)
{
   // The Begin() function is called at the start of the query

   TString option = GetOption();
    // Declaration des histogrammes
    TH2F *h2=new TH2F("ZtPt","Ztot vs Ptot",40,0,800,60,0,120);
    TH1F *h1=new TH1F("distZ","Z",120,0,120);
}
```

Absolutely necessary!

(Edit the file `MonAnalyse.C`)

http://caeinfo.in2p3.fr/root/Formation/en/Day5/MonAnalyse.C

My second Process



```
Bool_t MonAnalyse::Process(Long64_t entry)
{
   // Function called for selected entries only.
   // Entry is the entry number in the current tree.
   // Read branches not processed in ProcessCut() and fill histograms.
   // To read complete event, call fChain->GetTree()->GetEntry(entry).

   fChain->GetTree()->GetEntry(entry);
   TH2F *h2=(TH2F *)gROOT->FindObject("ZtPt");
   h2->Fill(Data->GetPtot().Z(),Data->GetZtot(),1.);
   TH1F *h1=(TH1F *)gROOT->FindObject("distZ");
   for(Int_t i=1;i<=Data->GetMult();i++)
        {
        Fragment *fra=Data->GetFragment(i);
        h1->Fill(fra->GetZ(),1.);
        }
   return kTRUE;
}
```

Read the event

The read event is stored in the event pointed by Data (Event) because this is the name of the branch.

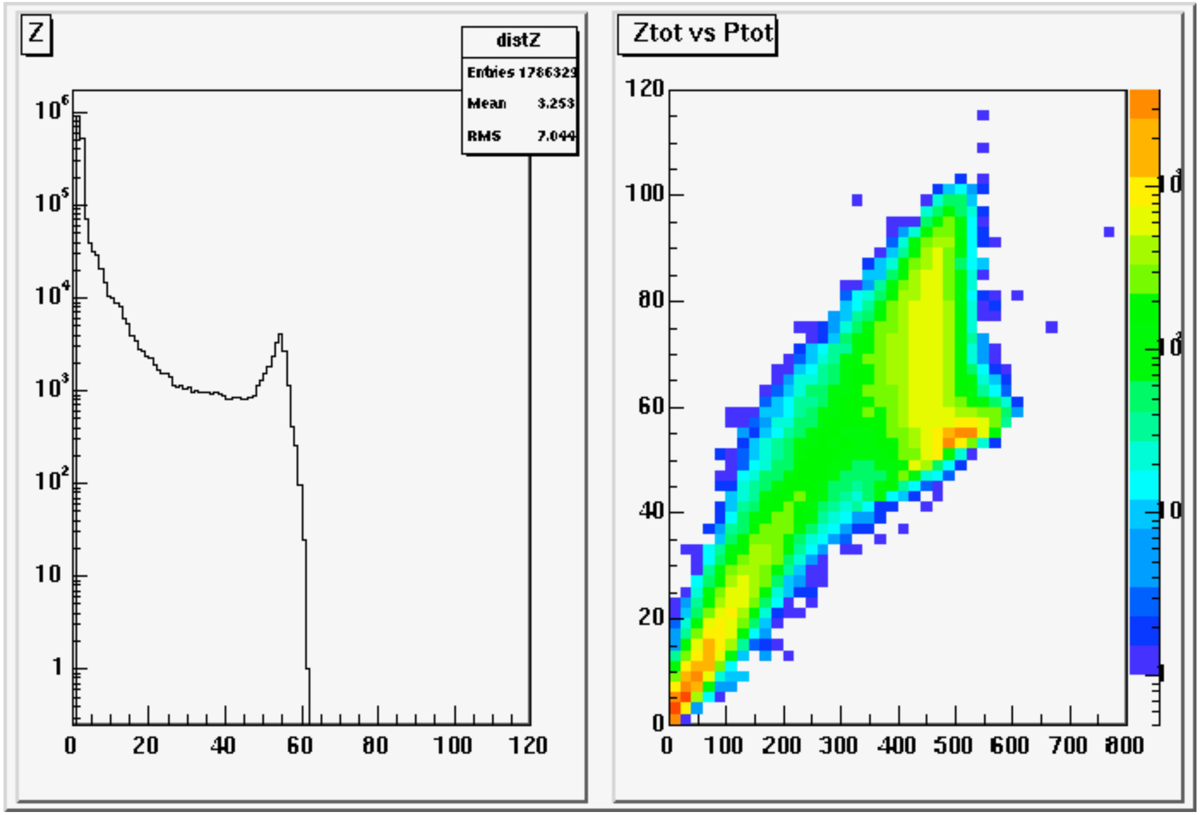
Loop on the fragments

My second Terminate

```
void MonAnalyse::Terminate()
{
   // Function called at the end of the event loop.
   // On affiche les spectres
   TH2F *h2=(TH2F *)gROOT->FindObject("ZtPt");
   TH1F *h1=(TH1F *)gROOT->FindObject("distZ");
   TCanvas *c2=(TCanvas *)gROOT->FindObject("c2");
   if(!c2)
    {
    c2=new TCanvas("c2","Resultat");
    }
   c2->Clear();
   c2->Divide(2,1);
   c2->cd(1);h1->SetStats(kTRUE);h1->Draw(); gPad->SetLogy(kTRUE);
   c2->cd(2);h2->SetStats(kFALSE);h2->Draw("zcol");gPad->SetLogz(kTRUE);
   c2->Update();
}
```

Execution!

```
root[19] mt->Process("MonAnalyse.C+")
Info in <TUnixSystem::ACLiC>: creating shared library ./MonAnalyse_C.so
Class MonAnalyse: Streamer() not declared
Class MonAnalyse: ShowMembers() not declared
```



We have lost the source files of Event and Fragment!

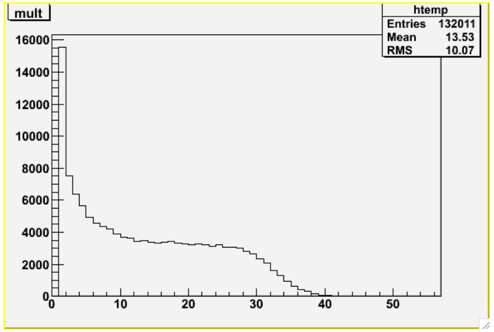
We will rebuild the essential part: the access to the fields (variables) of the class.

```
root[0] TFile *f=new TFile("indra_xesn50.root")
Warning in <TClass::TClass>: no dictionary for class Event is available
Warning in <TClass::TClass>: no dictionary for class Fragment is available
Warning in <TClass::TClass>: no dictionary for class TVector3 is available
root[1] f->MakeProject("indra","*","recreate++")
MakeProject has generated 3 classes in indra
indra/MAKE file has been generated
Shared lib indra/indra.so has been generated
Shared lib indra/indra.so has been dynamically linked
root[2] .class Event
List of member variable---------------------------------------------------
Defined in Event
indra/indra.so    1 0x1b       int mult //nombre de fragments
indra/indra.so    1 0x1f       TClonesArray* frags //-> tableau des fragments
root[3] .class Fragment
List of member variable---------------------------------------------------
Defined in Fragment
indra/indra.so    1 0xb        int A //nombre de nucleons
indra/indra.so    1 0xf        int Z //nombre de charges
indra/indra.so    1 0x13       TVector3 v , size=40 //vitesse
indra/indra.so    1   0xf        Double_t fX
indra/indra.so    1   0x17       Double_t fY
indra/indra.so    1   0x1f       Double_t fZ
```

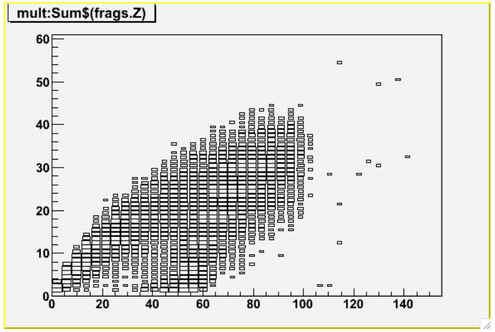
The raiders of the lost classes (part 2)

```
root[4] TTree *mt=(TTree *)f->Get("Arbre")
```

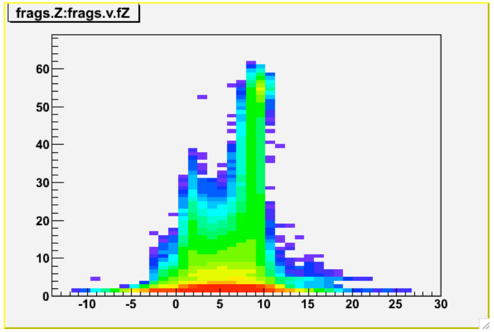
```
root[5] mt→Draw("mult")
```



```
root[6] mt→Draw("mult:Sum$(frags.Z)","","box")
```



```
root[7] mt→Draw("frags.Z:frags.v.fZ","","col")
```



Polymorphism

What is polymorphism?

- Here is a very simple example:

```
root[20] TList *tl=new TList()      ←—————————————— List of TObject
root[21] TF1 *f1=new TF1("gs1","gaus",0,100)
root[22] f1->SetParameters(70,15,2)
root[23] TF1 *f2=new TF1("gs2","gaus",0,200)
root[24] f2->SetParameters(50,65,8)
root[25] TArc *arc=new TArc(10,20,10)
root[26] tl->Add(f1)      ←—————————————————————— Is it correct ?
root[27] tl->Add(f2)
root[28] tl->Add(arc)
root[29] tl->ls()
```

- What is going on ?

How does this work?

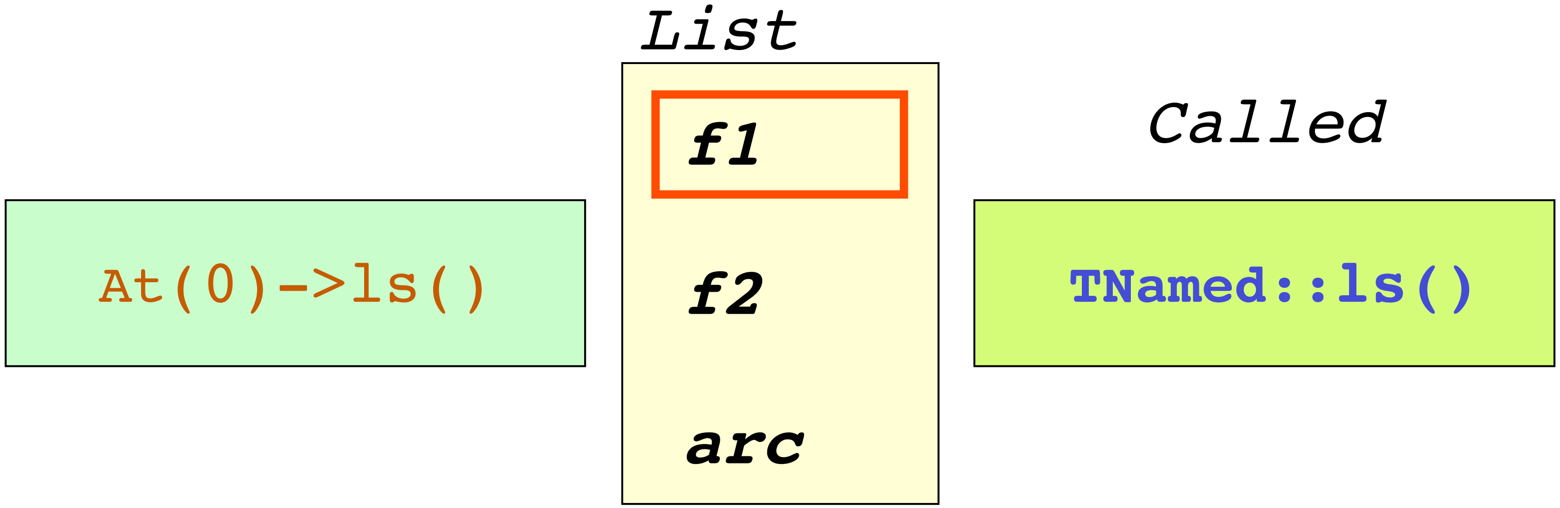
- The objects added to the list are all **`TObject`** each having its own **`ls`** method
- **`tl->ls()`** calls the **`ls`** method for each element. The "right" method is selected automatically

```
TList::ls(void)
{
 for(int i=0;i<GetSize();i++)
 {
 At(i)->ls();
 }
}
```

Number of elements in the list

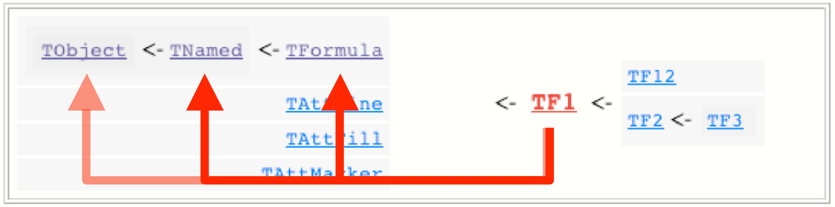
Fetch the i^{th} element (TObject) in the list

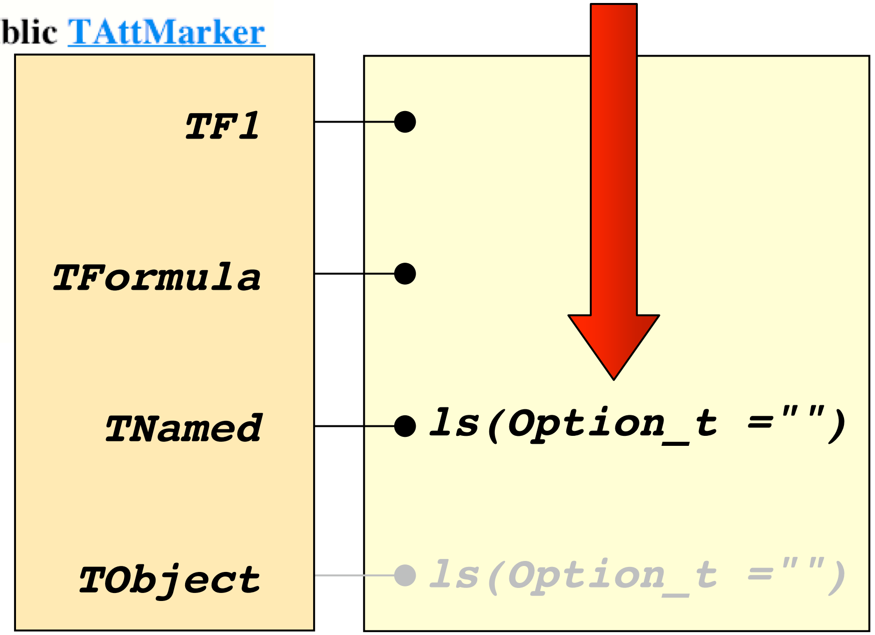
What happens in memory



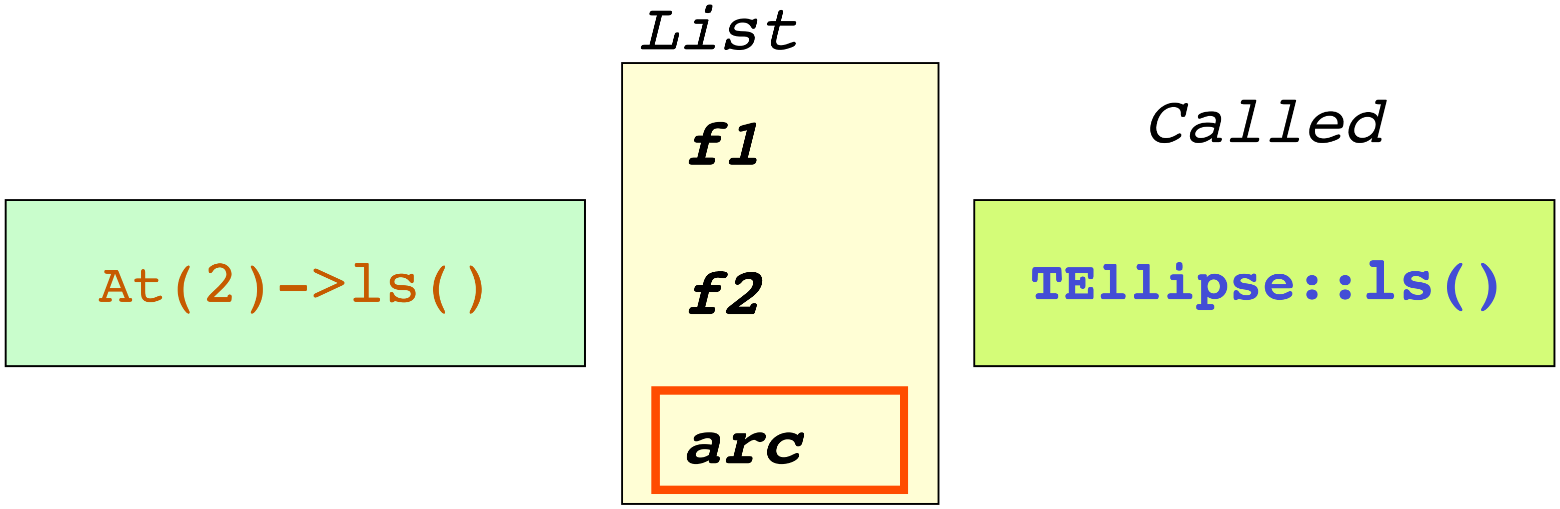
class TF1 : public TFormula, public TAttLine, public TAttFill, public TAttMarker

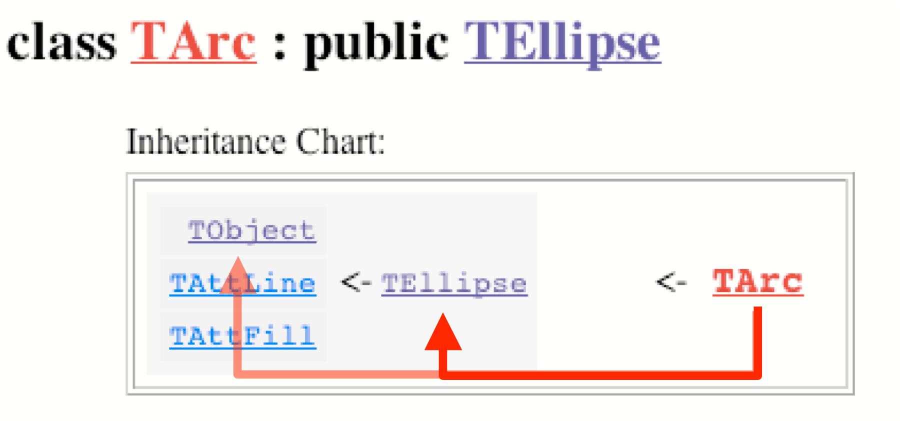
Inheritance Chart:





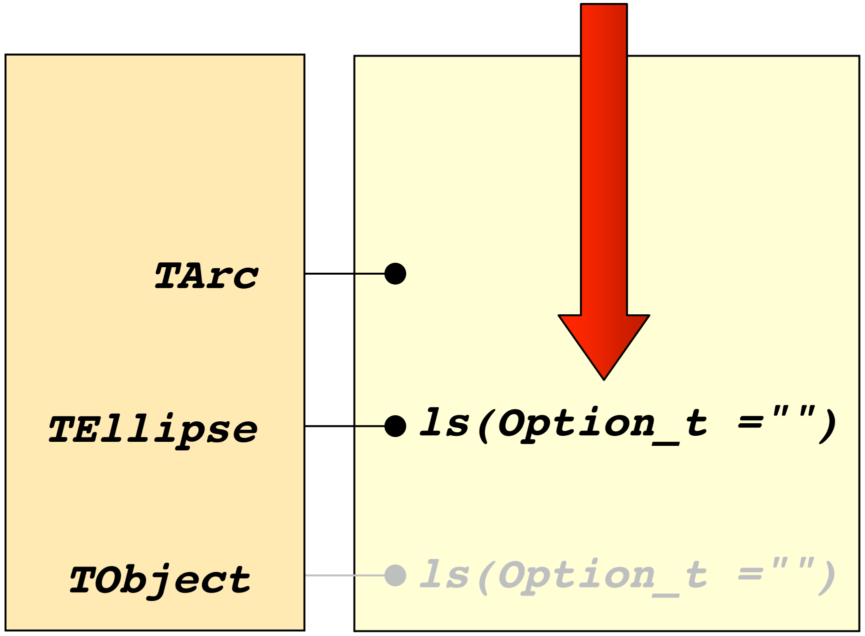
What happens in memory





class TArc : public TEllipse

Inheritance Chart:



We can do this in Fortran!

```
Subroutine ListeLs(nb_element,elementId,element)
    do i=1,nb_element
     if(elementId(i).eq.idTF1) then
       call TF1Ls(element(i))
     else if(elementId(i).eq.idArc) then
       call ArcLs(element(i))
     else if(elementId(i).eq.idLatex) then
       call LatexLs(element(i))
     endif
    enddo
    return
    end
```

- This becomes very complex when we want to add other objects, whereas **`TList::ls()`** is written only once for all!

Another example: drawing the objects from a TList

```
root[30] tl->Draw()
```

- What happens ?

```
TList::Draw(Option_t *opt)
{
 for(int i=0;i<GetSize();i++)
 {
 At(i)->Draw(opt);
 }
}
```

- The loop in detail:

```
At(0)->Draw() ≡ f1->Draw()
At(1)->Draw() ≡ f2->Draw()  ←——————  Erases the first drawing!
At(2)->Draw() ≡ arc->Draw()
```

Modify the display of a TList while keeping all the rest…

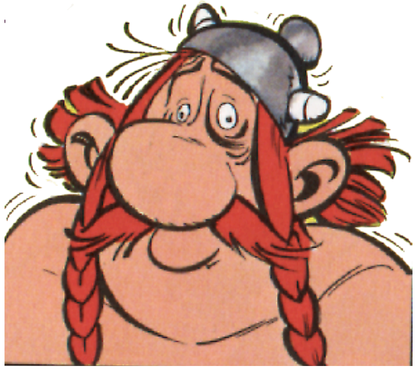
- We want to have a **`TList`** for which **`Draw()`** draws the first element with the required option and the other elements are drawn with the **`"same"`** option.
- It is possible because of the inheritance!

```
Class MaListe: public TList
```

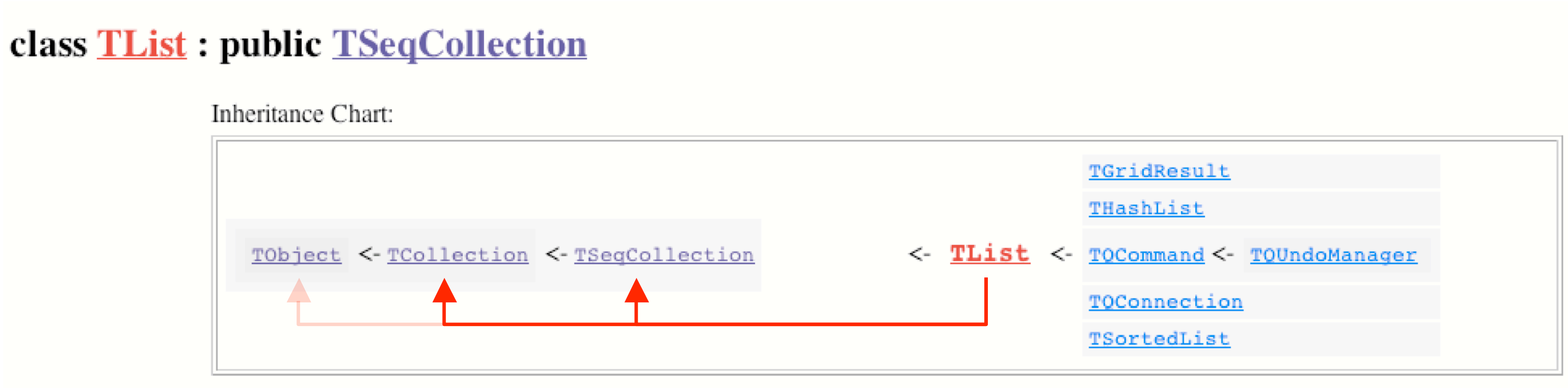
- We will only modify the **`Draw()`** method!

```
void MaListe::Draw(Option_t *opt="")
```

Include a new class MaListe in ROOT



Which Draw() method?



class TList : public TSeqCollection

Inheritance Chart:

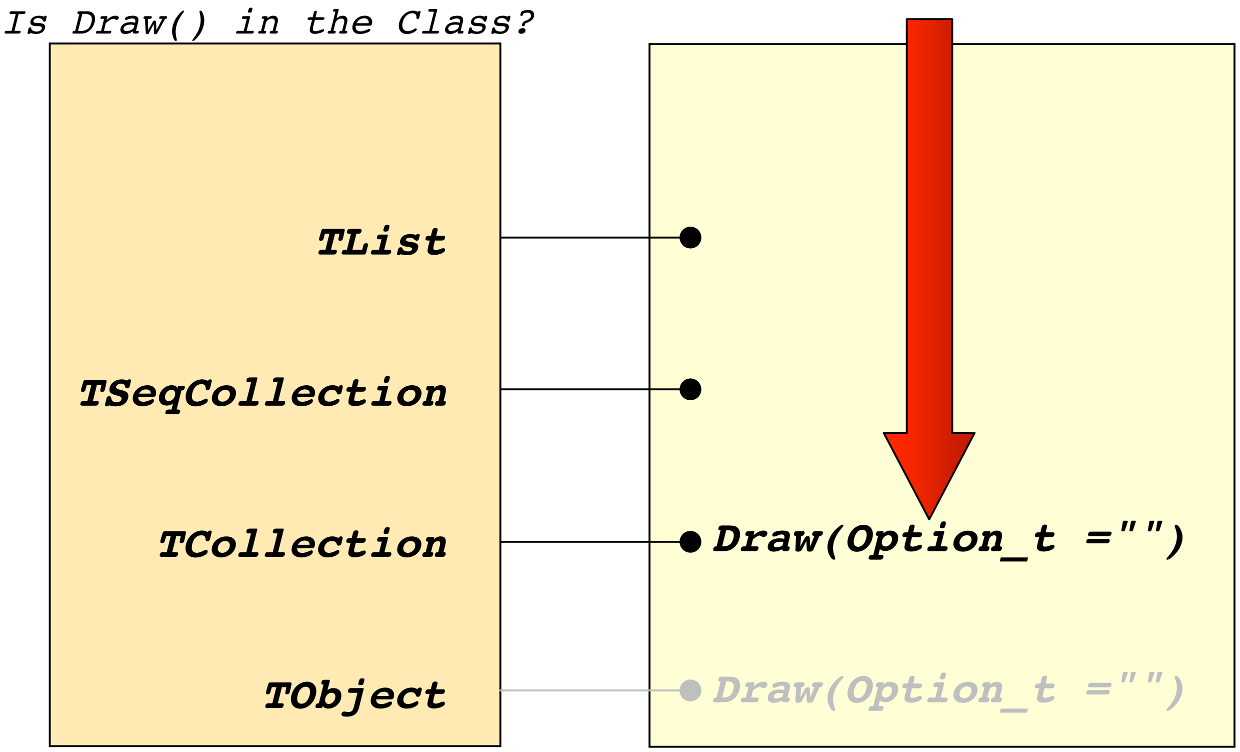
TObject <- TCollection <- TSeqCollection <- TList <- TGridResult

THashList

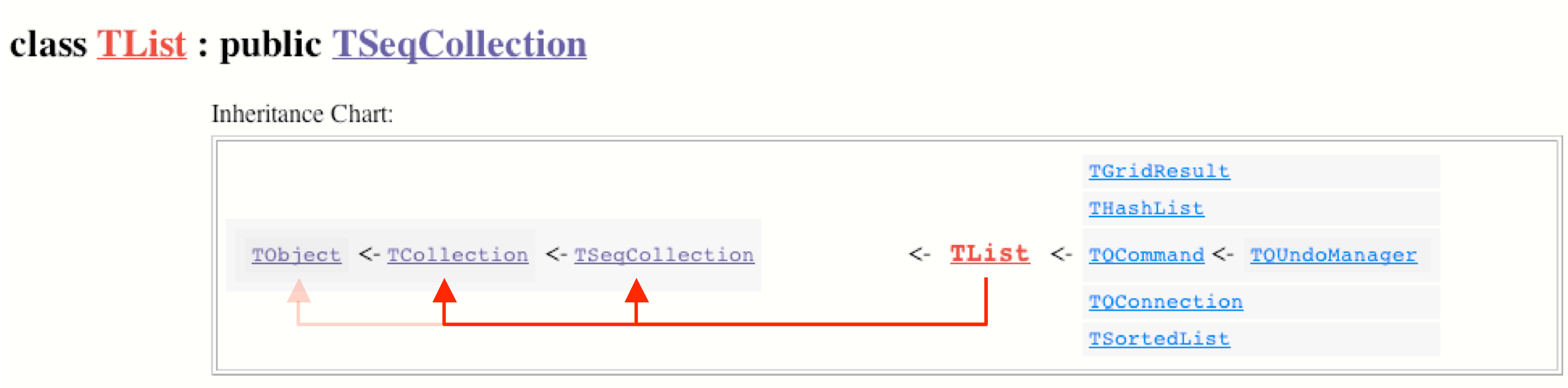
TQCommand <- TQUndoManager

TQConnection

TSortedList

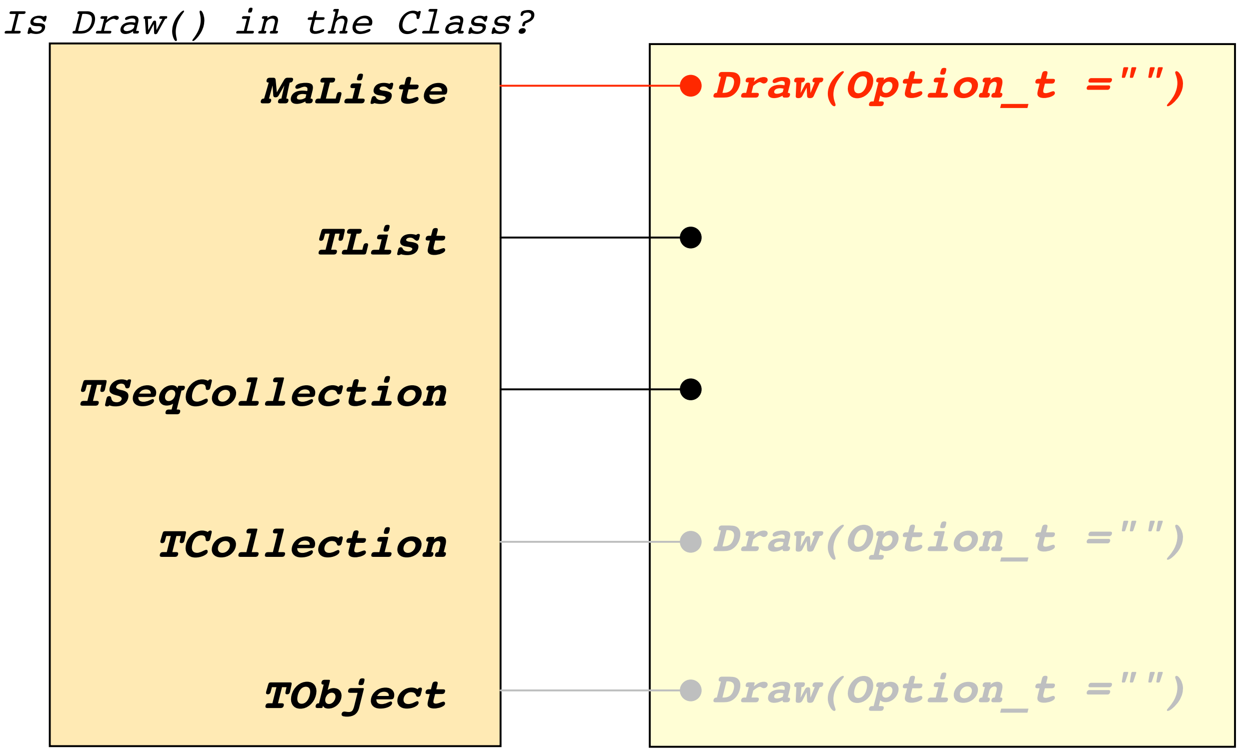


Which Draw() method?



class TList : public TSeqCollection

Inheritance Chart:



Is Draw() in the Class?

MaListe — Draw(Option_t ="")

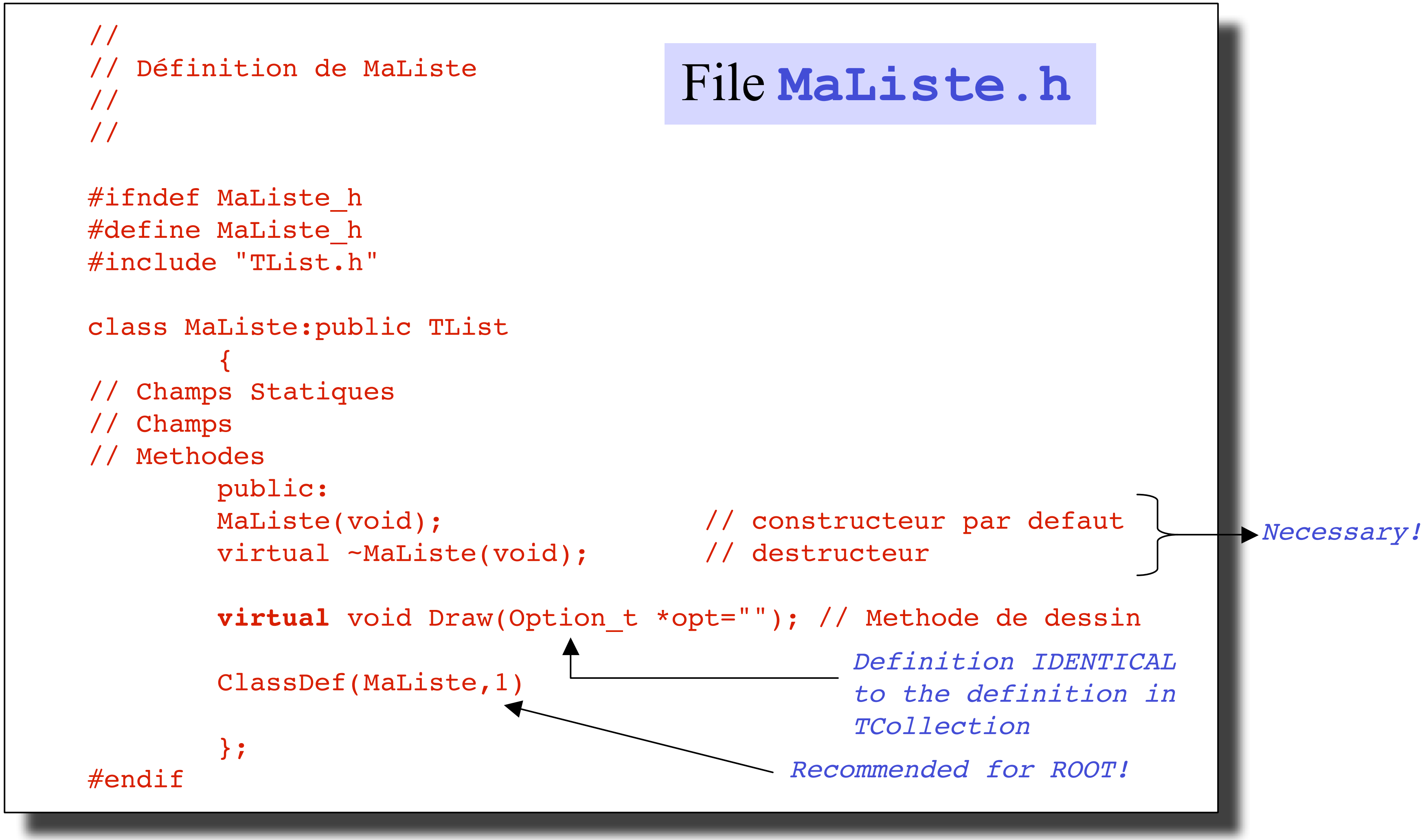
TList

TSeqCollection

TCollection — Draw(Option_t ="")

TObject — Draw(Option_t ="")

My own class MaListe
I: Definition



File **MaListe.h**

```
//
// Définition de MaListe
//
//

#ifndef MaListe_h
#define MaListe_h
#include "TList.h"

class MaListe:public TList
        {
// Champs Statiques
// Champs
// Methodes
        public:
        MaListe(void);                   // constructeur par defaut
        virtual ~MaListe(void);          // destructeur

        virtual void Draw(Option_t *opt=""); // Methode de dessin

        ClassDef(MaListe,1)

        };
#endif
```

Necessary!

Definition IDENTICAL to the definition in TCollection

Recommended for ROOT!

http://caeinfo.in2p3.fr/root/Formation/en/Day5/MaListe.h

My own class MaListe II: Implementation

```
#include "MaListe.h"
#include <stdio.h>
#include <iostream.h>
ClassImp(MaListe)
```

Recommended for ROOT!

```
MaListe::MaListe(void):TList()
{
//
// constructeur par defaut
//
}
MaListe::~MaListe(void)
{
//
// Destructeur
//
}
```

```
void MaListe::Draw(Option_t *opt)
{
//
// Méthode de dessin
//
 for(int i=0;i<GetSize();i++)
   {
    if(i == 0)
    {
    At(i)->Draw(opt);
    }
   else
    {
    At(i)->Draw("same");
    }
   }
 }
```

File **MaListe.C**

http://caeinfo.in2p3.fr/root/Formation/en/Day5/MaListe.C

My own class MaListe III: Use

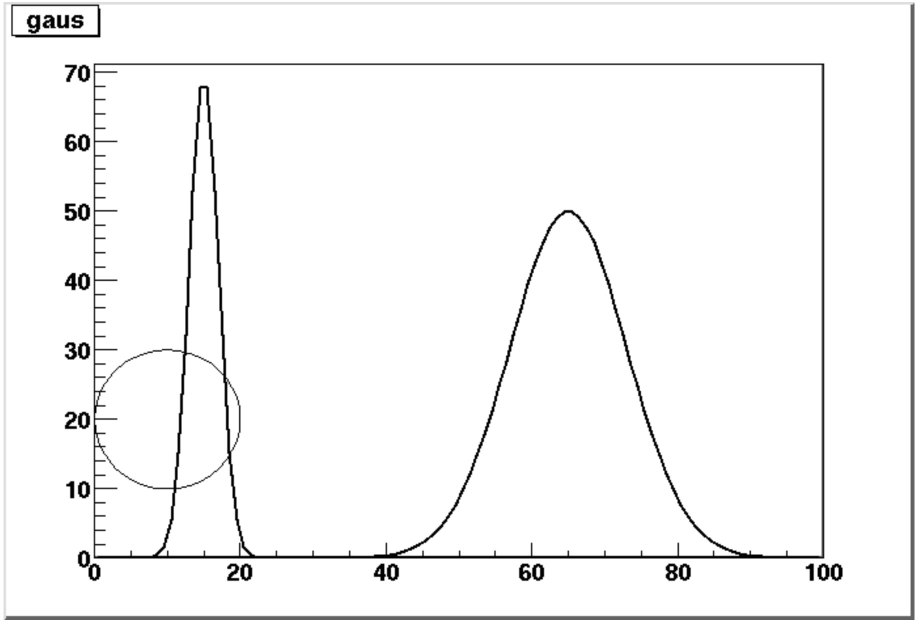
- To include it in ROOT:

```
root[30] .L MaListe.C+
```

- To use it

```
root[31] MaListe *ml=new MaListe()
root[32] ml->AddAll(tl)
root[33] ml->Draw()
```

- Bingo!



My own class MaListe IV: Let's have fun

- Try this:

```
root[40] gPad->DrawFrame(0,0,80,200)
root[41] ml->Draw("same")
root[42] .class MaListe
root[43] ml->DrawClass()
root[44] ml->ls()
root[45] ml->Inspect()
root[46] ml->ClassName()
root[48] htm->MakeClass("MaListe")
```

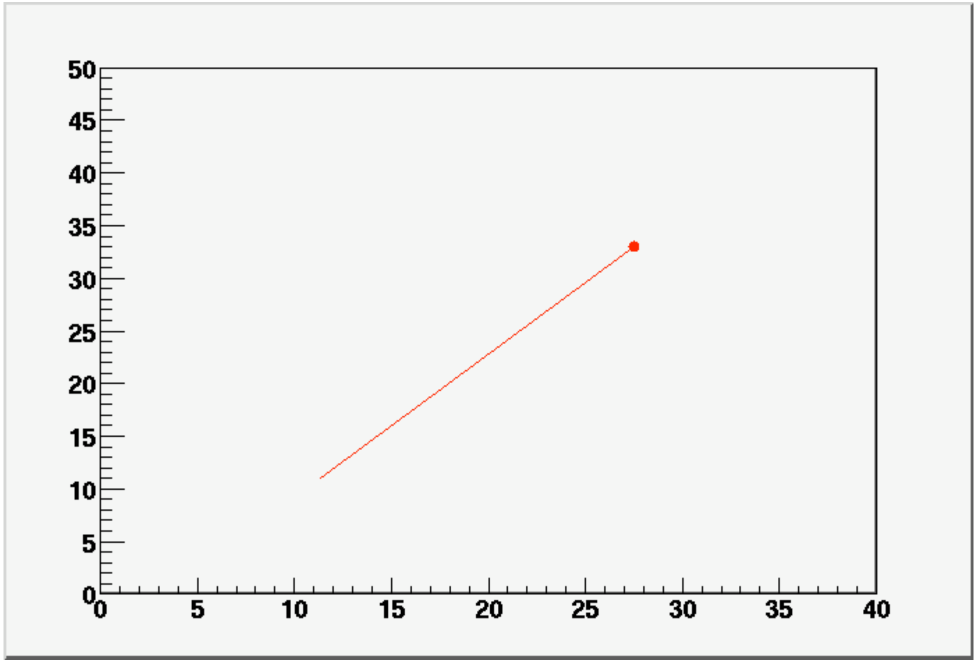
→ *Generates the documentation!*

Exercise 1

- Use a TSelector to build the following histograms for the events in the file **`indra_xesn50.root`**. Fill histograms only for events having a total charge (Z_{tot}) greater than 80:
  - Z vs V_z (have a look at the methods of **`TVector3`**)
  - Z_{max} (the largest Z value of the event) vs E_{perp}
  - $V_{transverse}$ vs V_z for Z=2 and Z=6 (see **`TVector3`**)
  - <Z>
- Save the results in the file **`results.root`**.

Exercise 2

Starting from the **`TLine`** class, build a new **`MPointeur`** class which plots a filled circle at the end of the line, as shown below. The **`Paint()`** method will be overloaded. The filled circle can be drawn by using the **`TMarker`** class with a style set to 20. The line will be drawn by calling **`TLine::Paint()`**. Declare two constructor methods: **`MPointeur()`** and **`MPointeur(x1,y1,x2,y2)`** (see those of **`TLine`**).



A standalone ROOT application

```
#include "TH1F.h"
#include "TApplication.h"
#include "TRint.h"

int main(int argc, char *argv[])
{
#ifdef WITHRINT
TRint *myapp=new TRint("RootSession",&argc,argv,NULL,0);
#else
TApplication *myapp=new TApplication("myapp",0,0);
#endif
TH1F *h=new TH1F("h","Test",100,-10,10);
h->FillRandom("gaus",100000);
h->Draw();
myapp->Run();
return 0;
}
```

http://caeinfo.in2p3.fr/root/Formation/en/Day5/MyApp.C

Under Unix/Linux/MacOsX

```
g++ MyApp.C -I$ROOTSYS/include `root-config --libs` `root-config --glibs`
a.out

g++ MyApp.C -DWITHRINT -I$ROOTSYS/include `root-config --libs` `root-config --glibs`
a.out
```