A Short Course on ROOT
Day 2

Subir Sarkar
SINP, Kolkata

VIIIth SERC School on EHEP, VECC, Kolkata
June 20 – July 10, 2011

Course Schedule – Day 2

- Physics Classes
  - LorentzVector, math classes
- Histograms and Functions
  - dimension, data type, profile, surface
  - fitting
- Input/Output
  - basic concept
  - TDirectory, TDirectoryFile, Tfile
- Collection Classes
- Plugin manager
  - RFIO, DCAP protocol handler

Math libraries

Histogram library

- TH1
- TF1

MathMore

- Random Numbers
- Extra algorithms
- Extra Math functions
- **GSL and more**

MathCore

- Function interfaces
- Physics Vectors
- Basic algorithms
- Basic Math functions

Statistical Libraries

- Statistical Utilities
- TMVA
- MLP

Fitting and Minimization

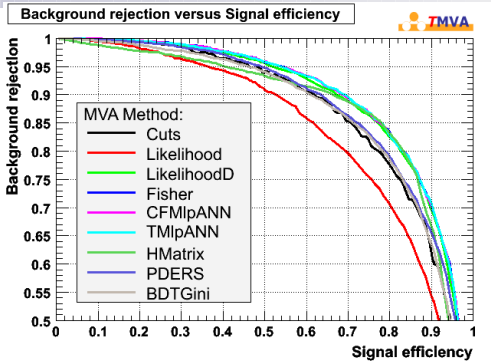
- New Fitter
- RooFit
- TMinuit
- TFumili
- Linear Fitter
- Minuit2 (new C++ Minuit)

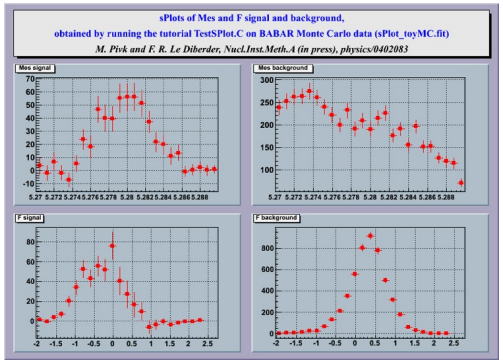
Linear Algebra

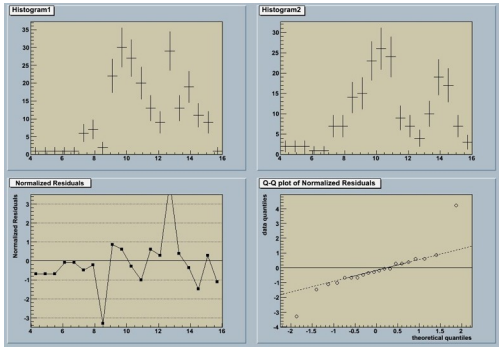
- TMatrix
- SMatrix

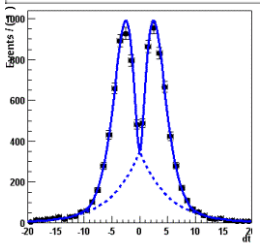
libCore

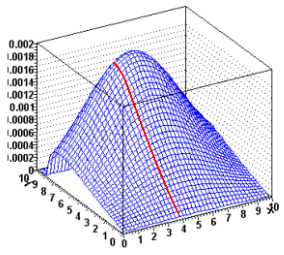
- TMath
- TRandom

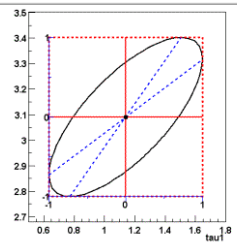












TLorentzVector

Constructor

```
TLorentzVector v1; // (0,0,0,0)
TLorentzVector v2(1,1,1,1);
TLorentzVector v3(v1);
TLorentzVector v4(TVector3(1,2,3),4);
```

Operator Overloading

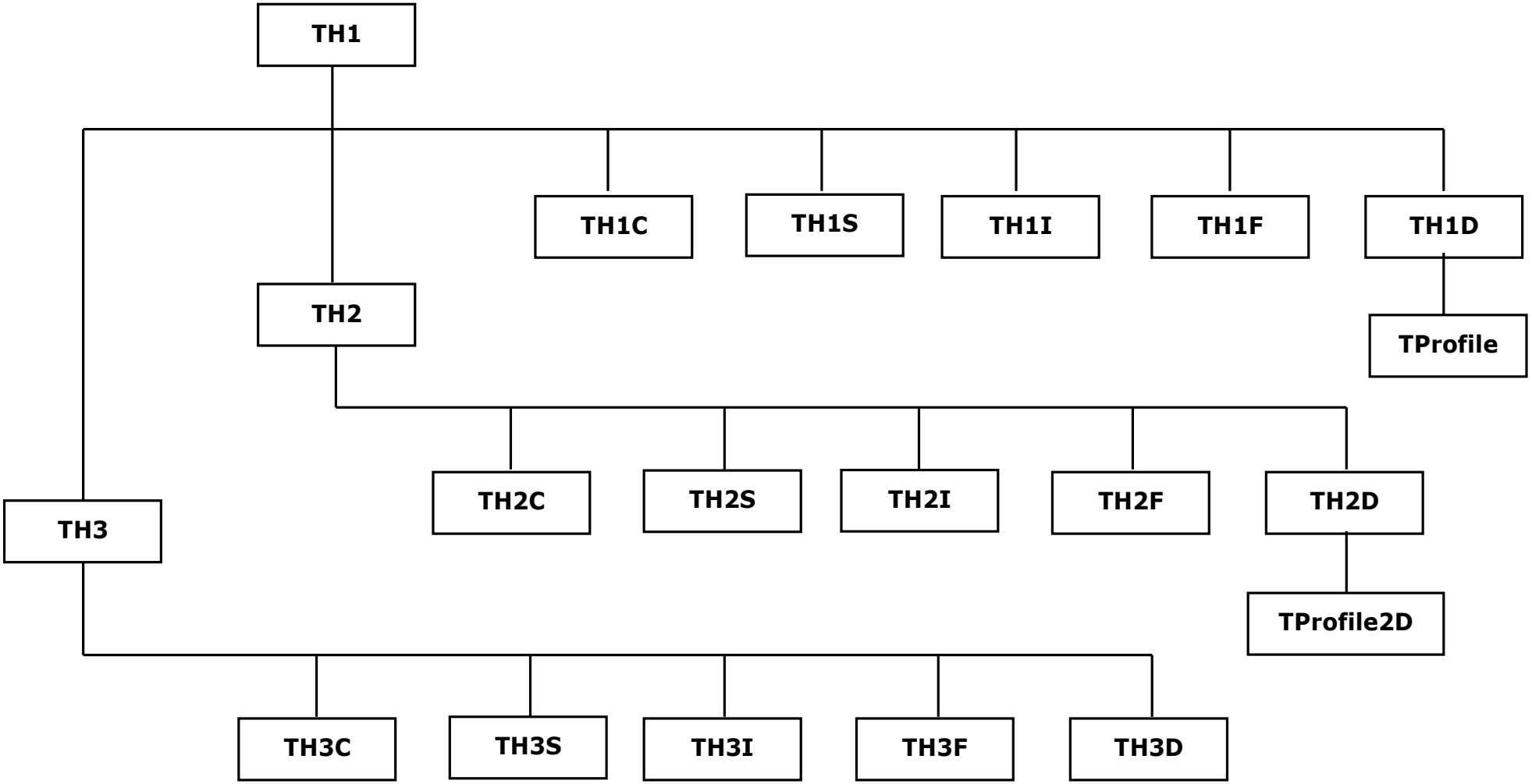
```
vx = v1 + v2;
v1 += v4;
v1 = -v3
```

Operations

```
double px = v1.Px();
TVector3 p = v1.Vect();
v1.SetXYZT(x,y,z,t);
v1.SetPxPyPzE(px,py,pz,e);
v1.SetXYZM(x,y,z,m);
double theta = v1.Theta();
double phi = v1.Phi();
v1.SetPhi(TMath::Pi());
```

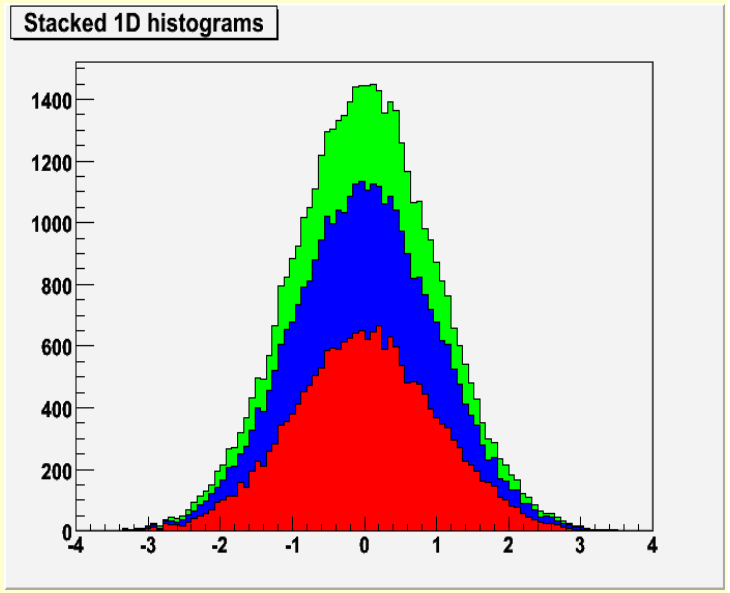
```
double s, s2;
s = v1.Dot(v2); // scalar product
s2 = v1.Mag2();
s = v1.M();
```

Histograms


TH1
TH1C
TH1S
TH1I
TH1F
TH1D
TProfile
TH2
TH2C
TH2S
TH2I
TH2F
TH2D
TProfile2D
TH3
TH3C
TH3S
TH3I
TH3F
TH3D

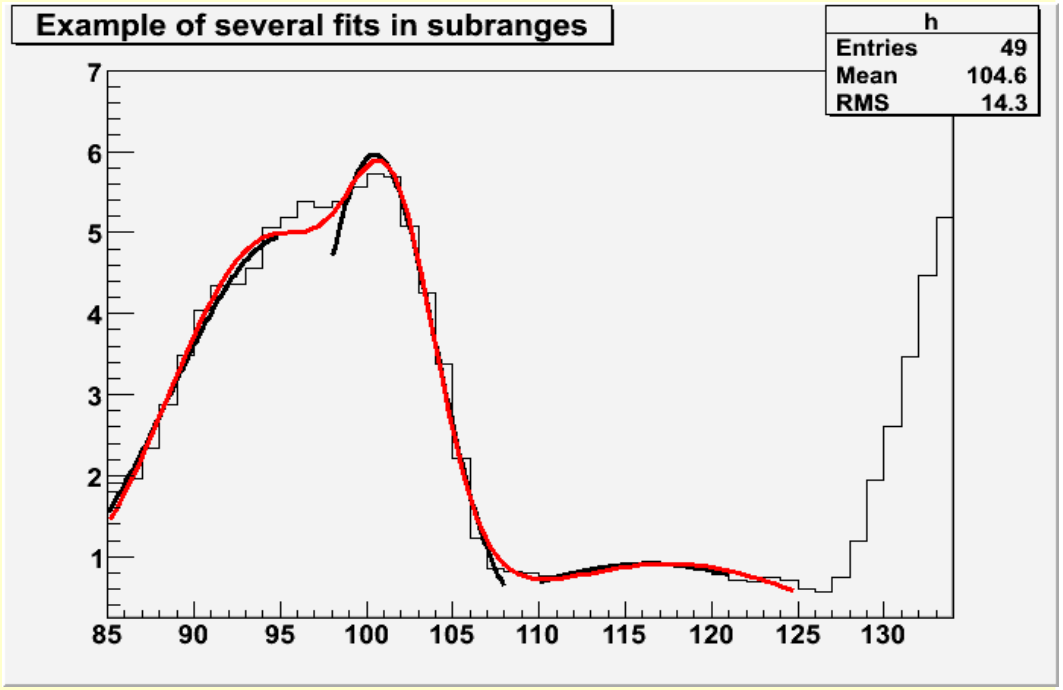
HStack

```
void stack() {
   THStack *hs = new THStack("hs","Stacked 1D histograms");
   //create three 1-d histograms
   TH1F *h1st = new TH1F("h1st","test hstack",100,-4,4);
   h1st->FillRandom("gaus",20000);
   h1st->SetFillColor(kRed);
   h1st->SetMarkerStyle(21);
   h1st->SetMarkerColor(kRed);
   hs->Add(h1st);
   TH1F *h2st = new TH1F("h2st","test hstack",100,-4,4);
   h2st->FillRandom("gaus",15000);
   h2st->SetFillColor(kBlue);
   h2st->SetMarkerStyle(21);
   h2st->SetMarkerColor(kBlue);
   hs->Add(h2st);
   TH1F *h3st
   = new TH1F("h3st","test hstack",100,-4,4
   h3st->FillRandom("gaus",10000);
   h3st->SetFillColor(kGreen);
   h3st->SetMarkerStyle(21);
   h3st->SetMarkerColor(kGreen);
   hs->Add(h3st);
   hs->Draw();
}
```



Fitting

```
{
   gROOT->Reset();
   const Int_t np = 49;
   Float_t x[np] = {1.913521, 1.953769, 2.347435, 2.883654, 3.493567,4.047560, 4.337210,
                    4.364347, 4.563004, 5.054247, 5.194183, 5.380521, 5.303213, 5.384578,
                    5.563983, 5.728500, 5.685752, 5.080029, 4.251809, 3.372246, 2.207432,
                    1.227541,0.8597788,0.8220503,0.8046592,0.7684097,0.7469761,0.8019787,
                    0.8362375,0.8744895, 0.9143721,0.9462768,0.9285364,0.8954604,0.8410891,
                    0.7853871,0.7100883,0.6938808,0.7363682,0.7032954,0.6029015,0.5600163,
                    0.7477068,1.188785, 1.938228,2.602717, 3.472962, 4.465014, 5.177035};
    TH1F* h = new TH1F("h","Example of several fits in subranges",np,85,134);
   h->SetMaximum(7);
   for (int i=0;i<np;i++) {
      h->SetBinContent(i+1,x[i]);
   }
   Double_t par[9];
   TF1* g1    = new TF1("g1","gaus",85,95);
   TF1* g2    = new TF1("g2","gaus",98,108);
   TF1* g3    = new TF1("g3","gaus",110,121);
   TF1* total = new TF1("total","gaus(0)
             +gaus(3)+gaus(6)",85,125);
   total->SetLineColor(2);
   h->Fit(g1,"R"); // use Range
   h->Fit(g2,"R+");
   h->Fit(g3,"R+");
   g1->GetParameters(&par[0]);
   g2->GetParameters(&par[3]);
   g3->GetParameters(&par[6]);
   total->SetParameters(par);
   h->Fit(total,"R+");
}
```



Saving Objects – Saving Types

Let's save a TNamed object

```
TNamed* o
     = new TNamed("name", "title");
std::write("file.bin", "obj1", o);
```

- Store data members of TNamed; need to know
  - type of object
  - data members for the type
  - where data members are in memory
  - read their values from memory, write to disk

Serialization

- Store data members of TNamed: serialization
  - type of object: Run Time Type Information(RTTI)
  - data members for the type: reflection
  - where data members are in memory: introspection
  - read their values from memory, write to disk: raw I/O

Complex task, and C++ is not your friend

TFile / TDirectory

- A TFile object may be divided in a hierarchy of directories, like a Unix file system
- Two I/O modes are supported
  - Key-mode (TKey). An object is identified by a name (key), like files in a Unix directory. OK to support up to a few thousand objects, like histograms, geometries, mag fields, etc.
  - TTree-mode to store event data, when the number of events may be millions, billions.

Saving Objects

- Open a File

```
TFile* f = File::Open("file.root","RECREATE")
```

- Write an object deriving from TObject

```
object->Write("optionalName")
```

"optionalName" or TObject::GetName()

- Write any object (with dictionary)

```
f->WriteObject(object, "name")
```

Self-describing files

- Dictionary for persistent classes written to the file
- ROOT files can be read by foreign readers
  - Java Analysis Studio (JAS)
- Support for Backward and Forward compatibility
  - files created today must be readable tomorrow
- Classes (data objects) for all objects in a file can be regenerated via TFile::MakeProject

```
Root > TFile f("demo.root");
Root > f.MakeProject("dir","*","new++");
```

TFile: Object Ownership

TFile owns histograms, graphs, trees
(due to historical reasons)

```
TFile* f = TFile::Open("myfile.root");
TH1F* h = new TH1F("h","h",100,-3.,3.);
h->FillRandom(“gaus”, 10000);
h->Draw();
h->Write();
TCanvas* c = new Tcanvas(“c”, “Canvas”);
c->Write();
delete f;
```

h automatically deleted: owned by file

c still there

TFile acts like a scope for hists, graphs, trees!

TFile: Object Ownership

Separate TFile and histograms

```
TFile* f =
TFile::Open("myfile.root");
TH1F* h = 0;
TH1::AddDirectory(kFALSE);
f->GetObject("h", h);
h->Draw();
delete f;
```

h will stay around even after the file is closed

TFile Example

```
#ifndef __CINT__
#include "TObjArray.h"
#include "TFile.h"
#include "TH1F.h"
#endif
void demo(){
   char name[10], title[20];
   TObjArray hlist(0);       // create an array of Histograms
   TH1F* h;                  // create a pointer to a histogram
   // make and fill histograms and add them to the object array
   for (Int_t i = 0; i < 15; i++) {
      sprintf(name,"h%d",i);
      sprintf(title,"histo nr:%d",i);
      h = new TH1F(name,title,100,-4,4);
      hlist.Add(h);
      h->FillRandom("gaus",1000);
   }
   // open a file and write the array to the file
   TFile f("demo.root","recreate");
   hlist.Write();
   f.Close();
}
```

TFile

```
root [0] TFile f("demo.root") //The first 100 bytes are taken by
the file header.
root [1] f.Map();
20080702/102046  At:100  N=114       TFile
20080702/102046  At:214  N=423       TH1F             CX =  2.29
20080702/102046  At:637  N=414       TH1F             CX =  2.34
20080702/102046  At:1051  N=415       TH1F             CX =  2.33
20080702/102046  At:1466  N=416       TH1F             CX =  2.33
20080702/102046  At:1882  N=423       TH1F             CX =  2.29
20080702/102046  At:2305  N=419       TH1F             CX =  2.31
20080702/102046  At:2724  N=408       TH1F             CX =  2.38
20080702/102046  At:3132  N=417       TH1F             CX =  2.32
20080702/102046  At:3549  N=413       TH1F             CX =  2.35
20080702/102046  At:3962  N=406       TH1F             CX =  2.39
20080702/102046  At:4368  N=417       TH1F             CX =  2.33
20080702/102046  At:4785  N=416       TH1F             CX =  2.34
20080702/102046  At:5201  N=421       TH1F             CX =  2.31
20080702/102046  At:5622  N=420       TH1F             CX =  2.32
20080702/102046  At:6042  N=416       TH1F             CX =  2.34
20080702/102046  At:6458  N=3062      StreamerInfo     CX =  3.15
20080702/102046  At:9520  N=732       KeysList
20080702/102046  At:10252  N=53        FreeSegments
20080702/102046  At:10305  N=1         END
```

TFile

```
root [2] f.ShowStreamerInfo();
```

```
root [3] f.GetListOfKeys()->Print()
TKey Name = h0, Title = histo nr:0, Cycle = 1
TKey Name = h1, Title = histo nr:1, Cycle = 1
TKey Name = h2, Title = histo nr:2, Cycle = 1
TKey Name = h3, Title = histo nr:3, Cycle = 1
TKey Name = h4, Title = histo nr:4, Cycle = 1
TKey Name = h5, Title = histo nr:5, Cycle = 1
TKey Name = h6, Title = histo nr:6, Cycle = 1
TKey Name = h7, Title = histo nr:7, Cycle = 1
TKey Name = h8, Title = histo nr:8, Cycle = 1
TKey Name = h9, Title = histo nr:9, Cycle = 1
TKey Name = h10, Title = histo nr:10, Cycle = 1
TKey Name = h11, Title = histo nr:11, Cycle = 1
TKey Name = h12, Title = histo nr:12, Cycle = 1
TKey Name = h13, Title = histo nr:13, Cycle = 1
TKey Name = h14, Title = histo nr:14, Cycle = 1
```

TFile Traversal

```
#include "TFile.h"
#include "TKey.h"
#include "TH1F.h"
void iterate () {
   TFile f("demo.root");
   TIter next(f.GetListOfKeys());
   TObject *obj;
   while ((obj = next())) {
      TKey *key = dynamic_cast<TKey*>(obj);
      printf ("key: %s points to an object of class: %s at %d\n",
              key->GetName(), key->GetClassName(),
              key->GetSeekKey());
      TH1F *h = dynamic_cast<TH1F*>(key->ReadObj());
      h->DrawCopy();
   }
}
```

```
root [0] .x iterate.C
key: h0 points to an object of class: TH1F at 214
key: h1 points to an object of class: TH1F at 637
key: h2 points to an object of class: TH1F at 1051
key: h3 points to an object of class: TH1F at 1466
key: h4 points to an object of class: TH1F at 1882
key: h5 points to an object of class: TH1F at 2305
key: h6 points to an object of class: TH1F at 2724
...
```

Object in Memory, on Disk

```
root [0] TFile f("demo.root");
root [1] f.ls();
TFile**         demo.root
 TFile*         demo.root
  KEY: TH1F     h0;1    histo nr:0
  KEY: TH1F     h1;1    histo nr:1
  ...
```

```
root [3] f.ls("-m");
TFile**         demo.root
 TFile*         demo.root
root [4] f.ls("-d");
TFile**         demo.root
 TFile*         demo.root
  KEY: TH1F     h0;1    histo nr:0
  KEY: TH1F     h1;1    histo nr:1
  ...
```

```
root [5] TH1F *h = dynamic_cast<TH1F*>(f1.Get("h0"));
root [6] f.ls("-m");
TFile**         demo.root
 TFile*         demo.root
  OBJ: TH1F     h0      histo nr:0 : 0 at: 0xa105ac8
```

Object List

```
root [7] h->Draw();
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [8] gROOT->GetListOfCanvases()->ls()
Canvas Name=c1 Title=c1 Option=
 TCanvas fXlowNDC=0 fYlowNDC=0 fWNDC=1 fHNDC=1 Name= c1 Title= c1 Option=
TFrame  X1= -4.000000 Y1=0.000000 X2=4.000000 Y2=43.050000
  OBJ: TH1F     h0      histo nr:0 : 0 at: 0xa105ac8
  OBJ: TPaveText title X1= -4.900000 Y1=45.471563 X2=-3.263218 Y2=48.162188
```

```
TH1* getHistogram(TFile* file, const string& name) {
  file->cd();
  TCanvas* can = dynamic_cast<TCanvas*>(gDirectory->Get(name.c_str()));
  if (!can) return 0;
  TObject *obj; TIter next(can->GetListOfPrimitives());
  while ((obj = next())) {
    if (obj->InheritsFrom(TPad::Class())) {
      TPad* pad = dynamic_cast<TPad*>(obj);
      // Now look for the histogram inside the pad
      TIter next1(pad->GetListOfPrimitives());
      while ((obj = next1())) {
        if (obj->InheritsFrom(TH1::Class()) &&
           !obj->InheritsFrom(TH2::Class()))
          return dynamic_cast<TH1*>(obj);
      }
    }
  }
  Return 0;
}
```

TDirectory

```
root [0] gDirectory->pwd()
Rint:/
root [1] TFile f1("demo.root")
root [2] gDirectory->pwd()
demo.root:/
root [3] TFile f2("demo2.root")
root [4] gDirectory->pwd()
demo2.root:/
root [5] f1.cd();
root [6] gDirectory->pwd()
demo1.root:/
root [7] gROOT->cd();
root [8] gDirectory->pwd()
Rint:/
```

Subdirectories and navigation

```
root [] TFile *f = Tfile::Open("AFile.root","RECREATE");
root [] f->mkdir("ADir");
(class TDirectory*)0x89e51d8
root [] f->ls()
TFile**         AFile.root
 TFile*         AFile.root
  TDirectoryFile*                ADir    ADir
  KEY: TDirectory       ADir;1   ADir
root [] f->cd("ADir");
root [] gDirectory->pwd();
AFile.root:/ADir
root [] gFile->pwd();
AFile.root:/
root [] f->pwd();
AFile.root:/

root [] TH1F *histo = new TH1F("histo", "histo", 10, 0, 10); // Histogram
root [] gDirectory->ls();
TDirectoryFile*         ADir    ADir
 OBJ: TH1F      histo   histo : 0 at: 0x8a62ae0
root [] f->Write();
root [] gDirectory->ls();
TDirectoryFile*         ADir    ADir
 OBJ: TH1F      histo   histo : 0 at: 0x8a62ae0
 KEY: TH1F      histo;1 histo
root [] TH1 *h = dynamic_cast<TH1*>(f->Get("ADir/histo;1"));
root [] h->Print();
TH1.Print Name  = histo, Entries= 0, Total sum= 0
```

Object Ownership

- Histograms, trees, and event-lists created by the user are owned by the current directory (`gDirectory`). When the current directory is closed or deleted the objects it owns are removed from memory too

```
root [0] h->SetDirectory("new_dir");
root [0] h->SetDirectory(0); //no associated directory
root [0] TH1::AddDirectory(kFALSE);
```

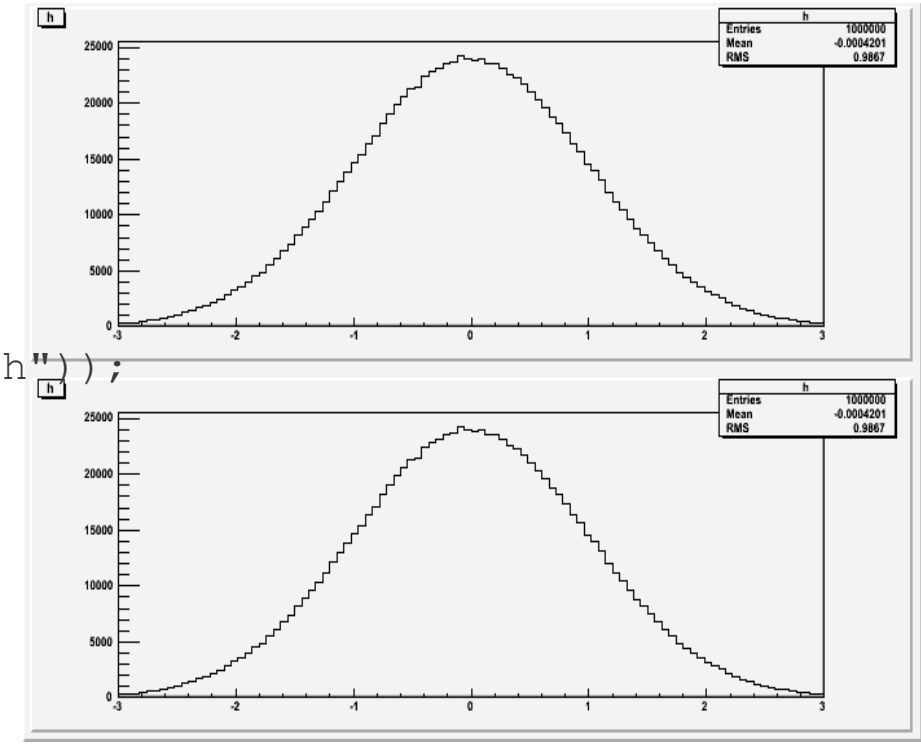
- The TROOT master object (gROOT) has several collections of objects. Objects that are members of these collections are owned by gROOT
- An object created by another object is owned by the later
  - TH1::Fit creates a TF1 and owns it
- An object created by `DrawCopy` is owned by the drawing pad

```
root [0] h->Draw();
root [1] TH1F* hnew = h->DrawCopy();
```

- The collection classes always store pointers to objects, *never copies of objects*
  - it is the user's responsibility to keep track of ownership

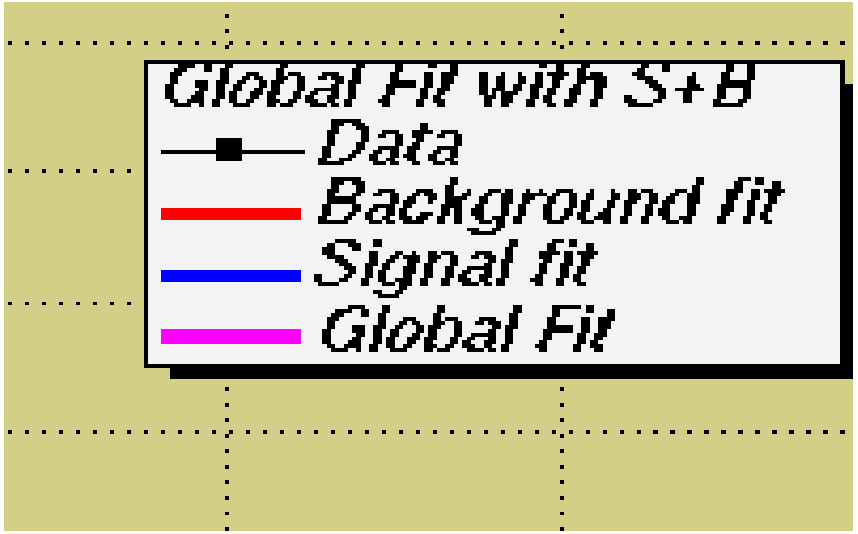
Object Ownership

```
root [] TFile f("test.root")
root [] f.ls()
TFile**         test.root
 TFile*         test.root
  KEY: TH1F     h;1     h
root [] TCanvas c;
root [] c->Divide(1,2);
root [] TH1F* h = dynamic_cast<TH1F*>(f.Get("h"));
root [] c->Divide(1,2);
root [] c->cd(1);
root [] h->Draw();
root [] c->cd(2);
root [] h->DrawCopy()
(const class TH1*)0x9cee5a8
root [] c->ls()
Canvas Name=c1 Title=c1 Option=
 TCanvas fXlowNDC=0 fYlowNDC=0 fWNDC=1 fHNDC=1 Name= c1 Title= c1 Option=
  TPad fXlowNDC=0.01 fYlowNDC=0.51 fWNDC=0.98 fHNDC=0.48 Name= c1_1 Title= c1_1
Option=
   TFrame  X1= -3.000000 Y1=0.000000 X2=3.000000 Y2=25498.200000
   OBJ: TH1F    h       h : 0 at: 0x9bd62d8
   OBJ: TPaveText       title   X1= -3.675000 Y1=26932.474034 X2=-3.459025
Y2=28526.111549
  TPad fXlowNDC=0.01 fYlowNDC=0.01 fWNDC=0.98 fHNDC=0.48 Name= c1_2 Title= c1_2
Option=
   TFrame  X1= -3.000000 Y1=0.000000 X2=3.000000 Y2=25498.200000
   OBJ: TH1F    h       h : 1 at: 0x9cee5a8
   OBJ: TPaveText       title   X1= -3.675000 Y1=26932.474034 X2=-3.459025 Y2=28526.111549
```



Legends

```
TLegend *legend = new TLegend(0.6, 0.65, 0.88, 0.85);
legend->SetTextFont(72);
legend->SetTextSize(0.04);
legend->AddEntry(histo, ''Data'', ''lp'');
legend->AddEntry(backfcn, ''Background'', ''l'');
legend->AddEntry(signalfcn, ''Signal Fit'', ''l'');
legend->AddEntry(fitFcn, ''Global Fit'', ''l'');
legend->SetHeader(''Global Fit with S+B '');
legend->Draw();
```



Postscript Interface

```
TPostScript ps("file.ps", 111);    // Open ps file
ps.range(16,18);                   // dimension
```

Single Page

```
hist->Draw();
```

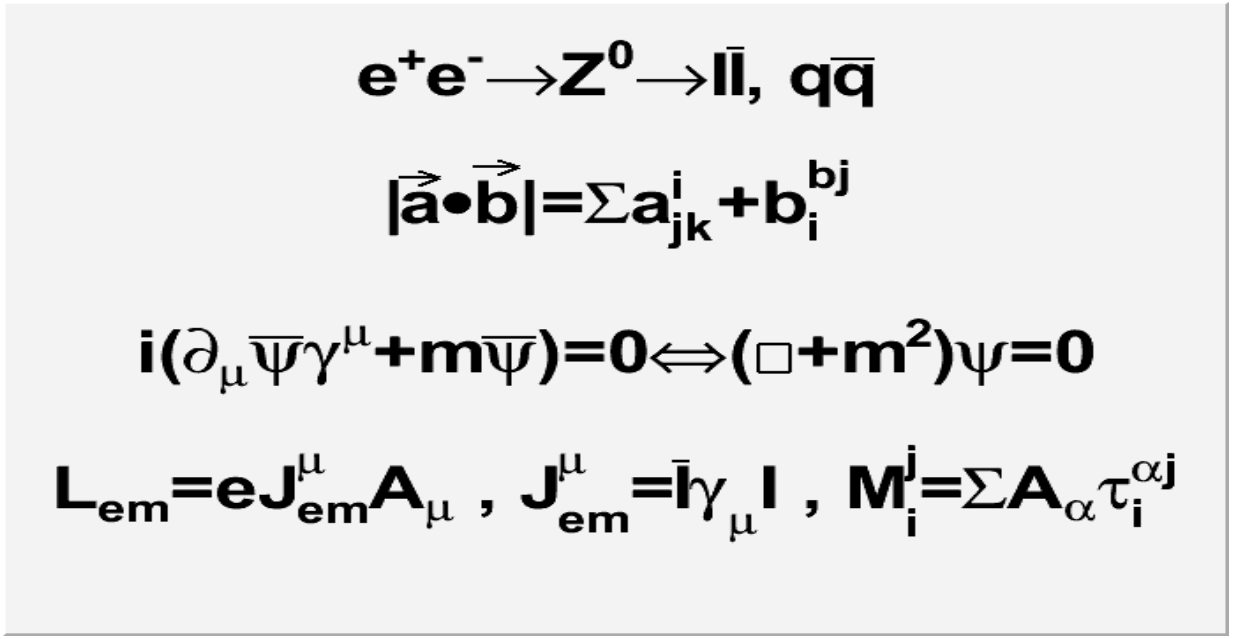
Multipage, Single zone

```
hist_1->Draw(); canvas->Update();
hist_2->Draw(); canvas->Update(); // etc.
```

Multiple Zone

```
canvas->Divide(2,2);
ps.NewPage();canvas->cd(1); hist_1->Draw(); canvas->Update();
ps.NewPage();canvas->cd(2); hist_2->Draw(); canvas->Update();
// etc.
ps.Text(x,y,"Hello World!");     // Add extra text
ps.Close();                      // Close file
gSystem->Exec("gv file.ps");     // Open Ghostview
```

Latex Support




```
#include "TROOT.h"
#include "TCanvas.h"
#include "TLatex.h"
void latex()
{
  gROOT->Reset();
  TCanvas *c1 = new TCanvas("c1");
  TLatex l;
  l.SetTextAlign(23);
  l.SetTextSize(0.1);
  l.DrawLatex(0.5,0.95,"e^{+}e^{-}#rightarrowZ^{0}#rightarrowI#bar{I},
              q#bar{q}");
  l.DrawLatex(0.5,0.75,"|#vec{a}#bullet#vec{b}|=#Sigmaa^{i}_{jk}+b^{bj}_{i}");
  l.DrawLatex(0.5,0.5,"i(#partial_{#mu}#bar{#psi}#gamma^{#mu}+m#bar{#psi})=0\
              #Leftrightarrow(#Box+m^{2})#psi=0");
  l.DrawLatex(0.5,0.3,"L_{em}=eJ^{#mu}_{em}A_{#mu}, J^{#mu}_{em}=#bar{I}\
              #gamma_{#mu}I, M^{j}_{i}=#SigmaA_{#alpha}#tau^{#alphaj}_{i}");
  c1->Print("latex2.ps");
}
```

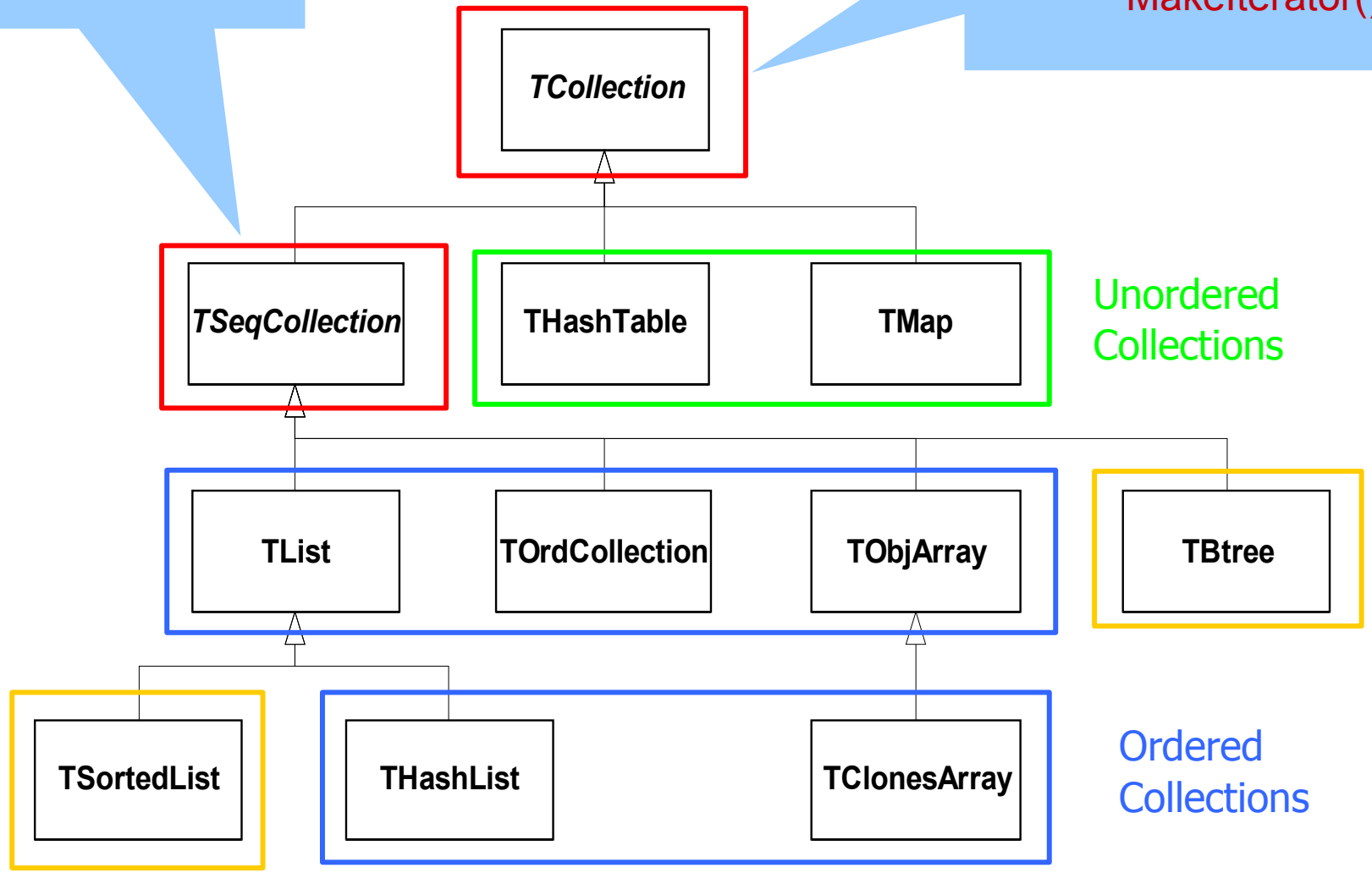
ROOT Collection Classes

- The ROOT collections are so called *polymorphic collections*
- The collections can contain different types of elements (polymorphism)
  - elements must be instances of classes
  - elements must be instances of classes deriving from **TObject**
- Collections themselves derive from TObject
  - So you can have collections of collections and collections can be made persistent

Collection Classes Hierarchy

Abstract Base Class
Defines methods like:
AddFirst(), AddLast(),
AddBefore(), AddAfter(),
RemoveFirst(), etc.

Abstract Base Class
Defines methods like:
Add(), Remove(), Clear(),
Delete(), FindObject(),
MakeIterator()



Iterators

- An iterator is used to traverse (walk through) a collection
- Having the iterator separate from the collection allows you to have several iterators on a single collection at the same time
- Each collection has its own associated iterator class
  - TList TListIter
  - TMap TMapIter
- In general you will use the generic **TIter** wrapper class
  - a TIter object can be used to iterate over any collection

```
TIter next(GetListOfTracks());
while (Track *tr = (Track*) next())
   tr->Fit();
```

TObject Protocol for Collections

- TObject defines basic protocol for collection elements
  - IsEqual() used by FindObject(), by default compares addresses
  - IsSortable() used for sorting, by default false
  - Compare() used for sorting, by default not usable
  - Hash() used for hashing, by default address of object
- The collections will call these TObject methods to find, sort and hash elements
- By overriding these methods a class can customize its behavior in a collection

Plug-in Manager

- Where are plug-ins used?

```
TFile *rf = TFile::Open("rfio://castor.cern.ch/alice/aap.root");
TFile *df = Tfile::Open("dcap://cmsdcdcap/pnfs.pi.infn.it\
/data/cms/store/user/sarkar/test.root");
```

- For example, to extend the base class TFile to be able to read RFIO files one needs to load the plug-in library libRFIO.so which defines the TRFIOFile class
- Protocol part of the file name URI triggers loading of plug-in. In these cases TRFIOFile and TDCacheFile objects are used, which both derive from TFile
- Currently 34 plug-ins are defined for 21 different (abstract) base classes
- Plug-in handlers can also be registered at run time
- A list of currently defined handlers can be printed using:

```
gROOT->GetPluginManager()->Print();
```