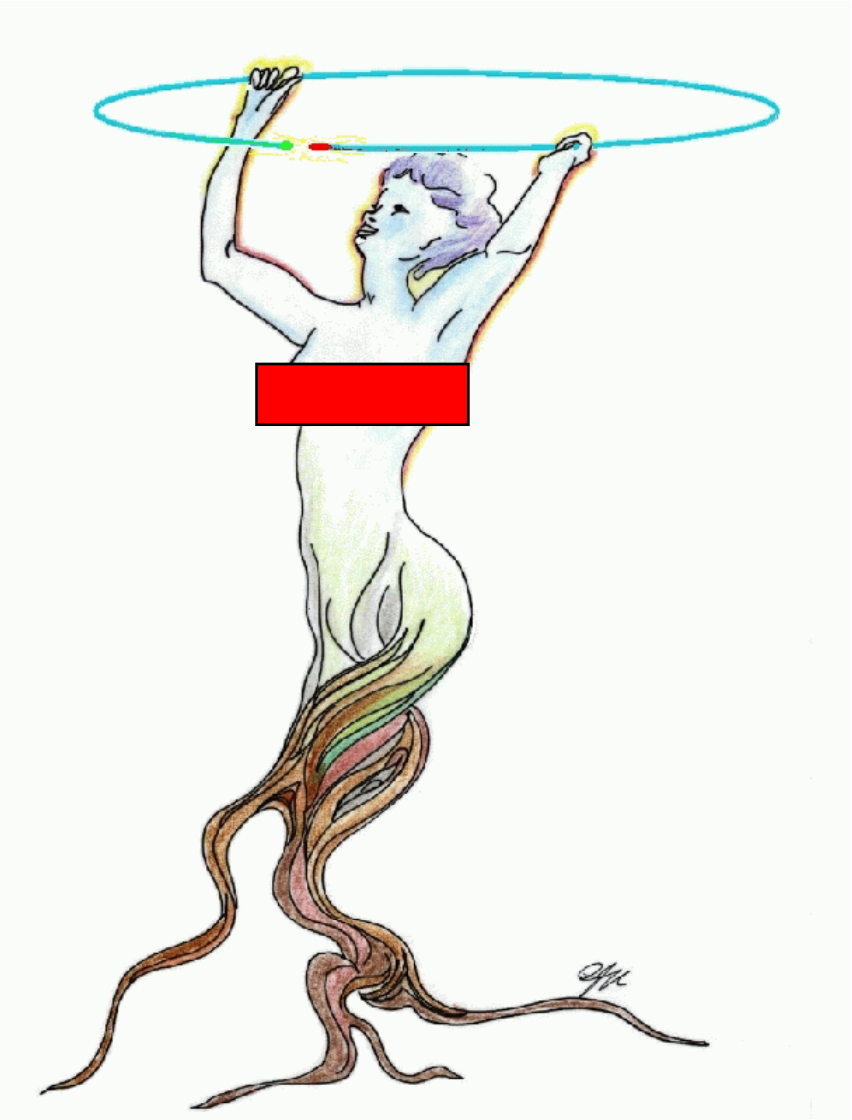


ROOT II ve III: ROOT temelleri (Etkileşimli kullanım, GUI ve Betik Uygulamaları, Analize Giriş)

Müge Karagöz (Ünel)
Oxford Üniversitesi

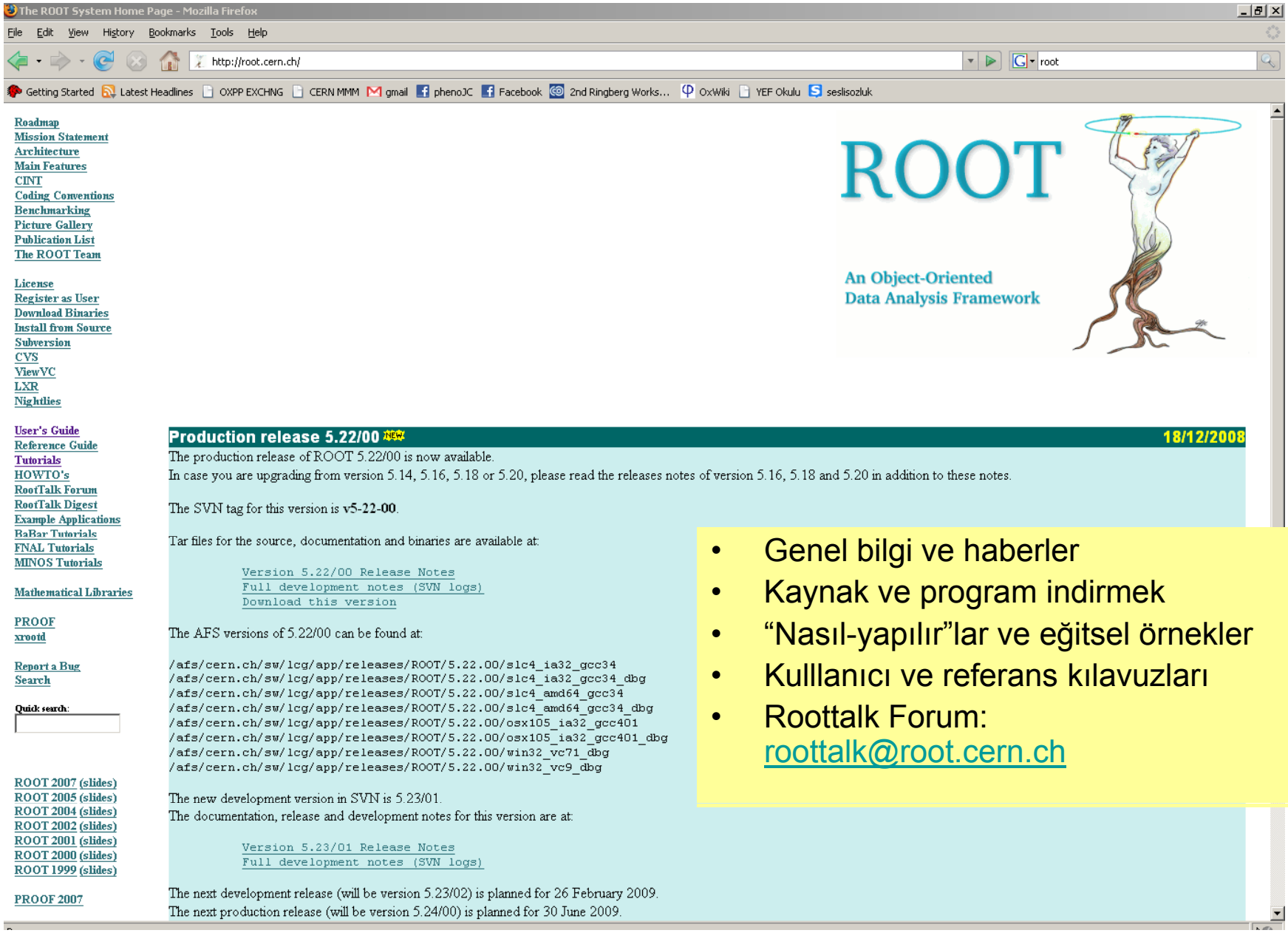
HPF Bilgisayar Uygulamaları Okulu
Çukurova Üniversitesi
26-31 Ocak 2009

Başlamadan…

- ROOT I'de (Özgür Çobanoğlu) ROOT temellerini ve bazı uygulamaları gördünüz:
  - İşletim sisteminde ROOT kurmak ve yönünüzü bulmak
  - Temel GUI uygulamaları
  - Veriyi uyum egrisi ile parametrize etmek
  - Olay görüntülemek (event display)
- ROOT II ve III'te devam edeceğiz:
  - Bazı ROOT I malzemesinin üzerinden (daha detaylı) geçeceğiz
  - Üzerine eklemeler yapacağız: GUI kullanımı, betiklerde işlemler, uyum eğrileri, ntuple veri analizine giriş
  - Cuma günü ROOT'ta analiz yapmayı daha detayli göreceksiniz

- Ozur: Ingilize klavyeden dolayi elimden kacan "grek turkcesi" karakterlerden dolayi…

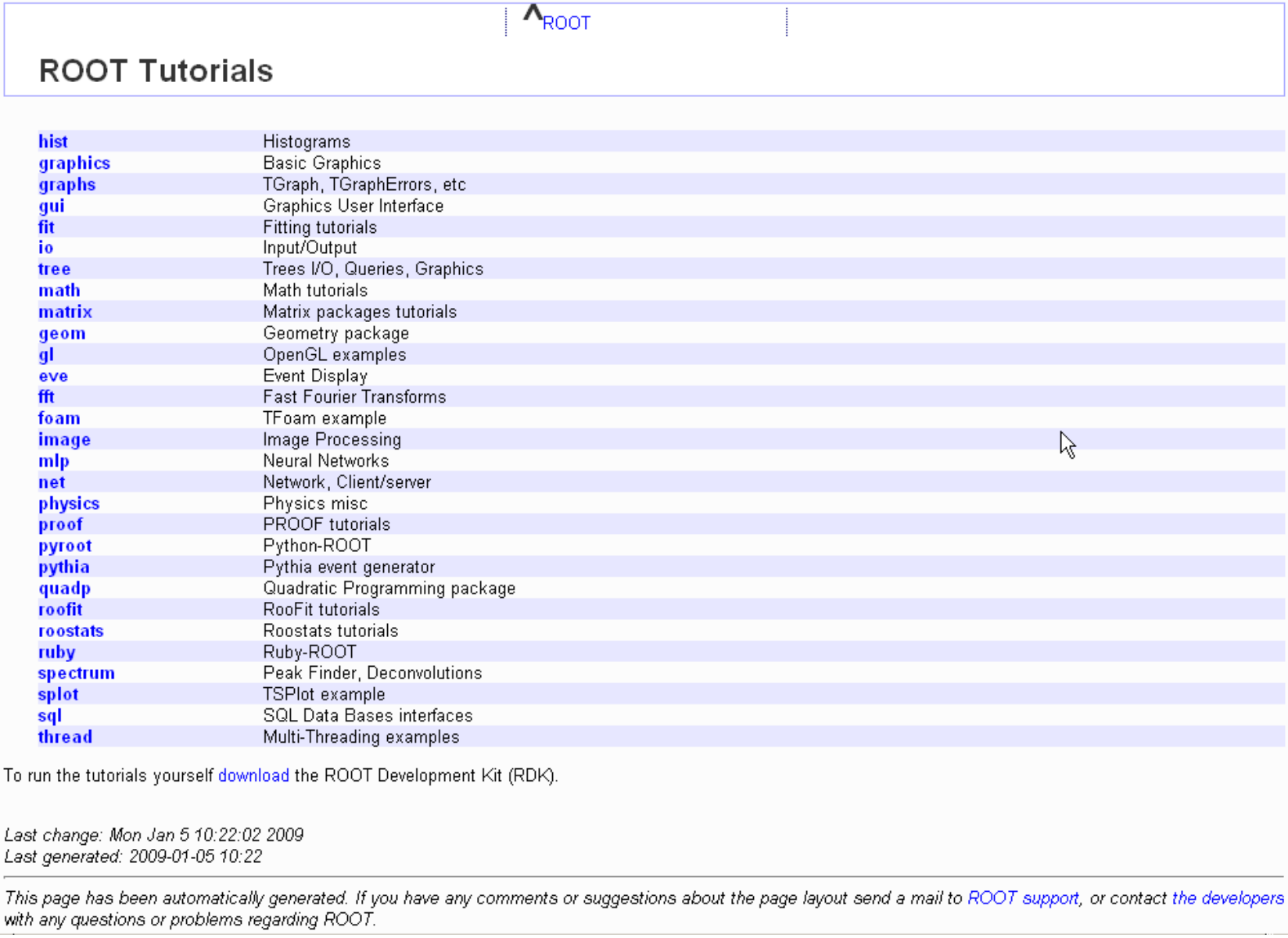
Birincil Kaynağınız: http://root.cern.ch



- Genel bilgi ve haberler
- Kaynak ve program indirmek
- “Nasıl-yapılır”lar ve eğitsel örnekler
- Kulllanıcı ve referans kılavuzları
- Roottalk Forum: roottalk@root.cern.ch

ROOT Eğitselleri: En yakın arkadaşınız

http://root.cern.ch/root/html/tutorials/



ROOT

ROOT Tutorials

hist	Histograms
graphics	Basic Graphics
graphs	TGraph, TGraphErrors, etc
gui	Graphics User Interface
fit	Fitting tutorials
io	Input/Output
tree	Trees I/O, Queries, Graphics
math	Math tutorials
matrix	Matrix packages tutorials
geom	Geometry package
gl	OpenGL examples
eve	Event Display
fft	Fast Fourier Transforms
foam	TFoam example
image	Image Processing
mlp	Neural Networks
net	Network, Client/server
physics	Physics misc
proof	PROOF tutorials
pyroot	Python-ROOT
pythia	Pythia event generator
quadp	Quadratic Programming package
roofit	RooFit tutorials
roostats	Roostats tutorials
ruby	Ruby-ROOT
spectrum	Peak Finder, Deconvolutions
splot	TSPlot example
sql	SQL Data Bases interfaces
thread	Multi-Threading examples

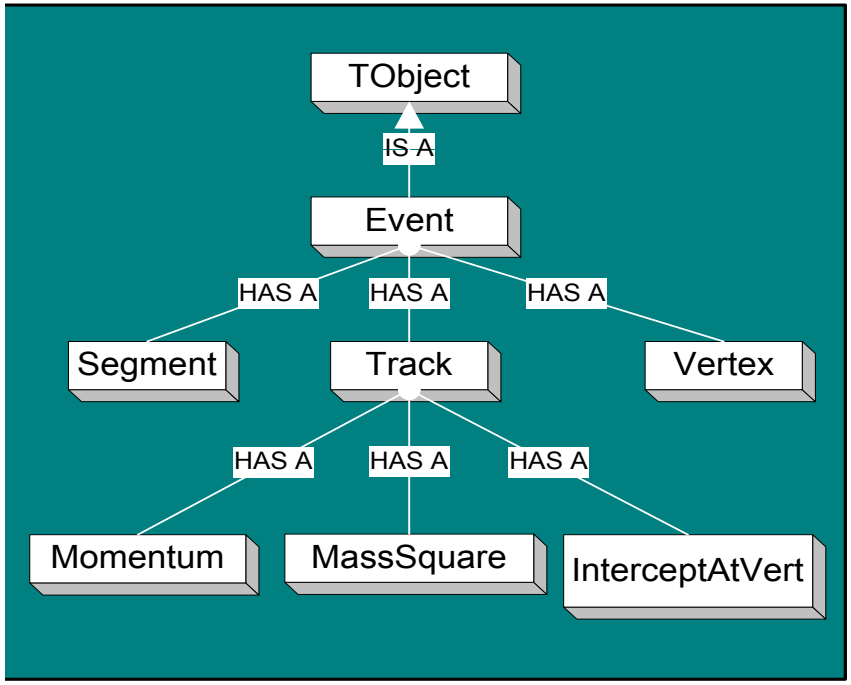
To run the tutorials yourself download the ROOT Development Kit (RDK).

Last change: Mon Jan 5 10:22:02 2009
Last generated: 2009-01-05 10:22

This page has been automatically generated. If you have any comments or suggestions about the page layout send a mail to ROOT support, or contact the developers with any questions or problems regarding ROOT.

ROOT'ta Nesne-Yönelimli (Object-Oriented) Kavramlar

- ROOT c++'ta yazılmış nesne-yönelimli bir programdır; altyapısı sınıflardan oluşur.
  - Sınıf (Class): bir bütünlüğün bir sistemde tanımı
  - Nesne (Object): bir sınıftan yaratılmış belli bir durum
  - Metod (Method): bir sınıfın fonksiyonları
  - Üye (Member): sınıfa "has a" ile bağlı olmak
  - Türeme (Inheritance): sınıfa "is a" ile bağlı olmak
- Yüzlerce tanımlanmış sınıf:
  - Core, CINT
- Kütüphaneler:
  - Açılışta yüklenen: Hist, Tree,…
  - Gerekince yüklenen: HistPainter,…
  - Özel amaçlı: Physics,…



Arasöz: PAW ve ROOT

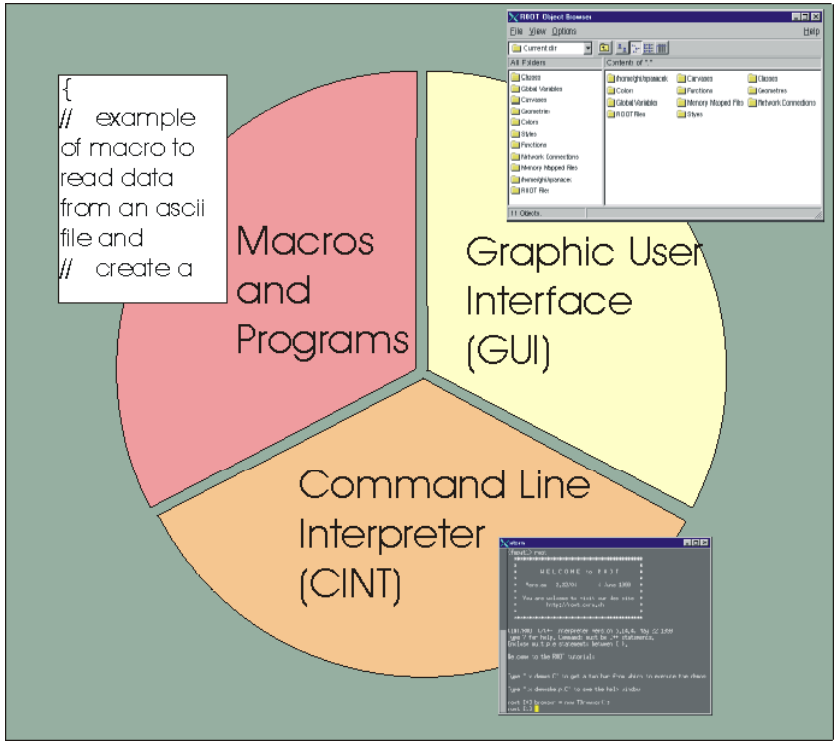
- PAW R. Brun + takımı tarafından OO furyası öncesi Fortan temel alınarak yazılmış bir analiz kodu ama yavaş yavaş geçerliliğini yitiren bir program.
- ROOT'un temel bir sürü kavramı aslında PAW'a dayanıyor.
- Eğer elinizde hala eski bir PAW dosyası varsa,

  % h2root paw.rz root.root

  ile formatını değiştirebilir ve dosyayı ROOT'ta açabilirsiniz.

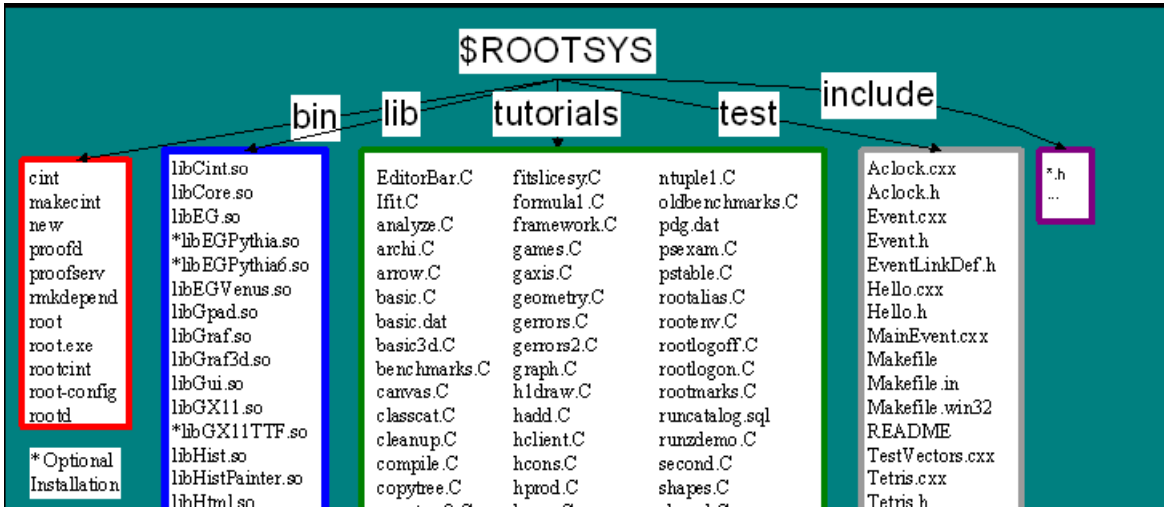
3 ROOT Arayüzü (Interface)

- GUI: Tarayıcılar (browsers), Paneller
- Komut satırı arayüzü (command line interface): C++, yorumlayıcı (interpreter) CINT
  - Buradan ROOTCint'te öntanımlı fonksiyonları çağırarak direk işlem yapabilir, kendi makronuzu çalıştırabilirsiniz. Nesne sözükleri (Object dictionary) ile herşeyi yapabiliriz.
- Betik/program işlemci (Script/Program Processor): (derlenmiş/CINT)
  - Tiyo: genelde CINT'te kod daha yavaş çalışır ve hata oranı daha yüksektir derlenmiş koda göre.



Unix'te ROOT: Bazı Hatırlatmalar

- ROOT oturumları unix'te satırdan komut girmeye benzetilinebilir (ya da bir betik dili olan python'a):
  - $HOME/.root_hist ile en son adımlarınızı takip edebilirsiniz (↑/↓ komut geçmişini verecektir)
  - Sekme tuşu (<TAB>) ile otomatik tamamlama yapabilirsiniz
- Stilinizi $HOME/<dir>/rootlogon.C dosyasından ayarlayabilirsiniz
  - Bu dosyanın adını $HOME/.rootrc dosyasında belirleyebilirsiniz
- Kullanacağınız ROOT sürümünü $HOME/.<your_shell>rc ile ayarlayabilirsiniz.



Etkileşimli (Interactive) Çalışmak

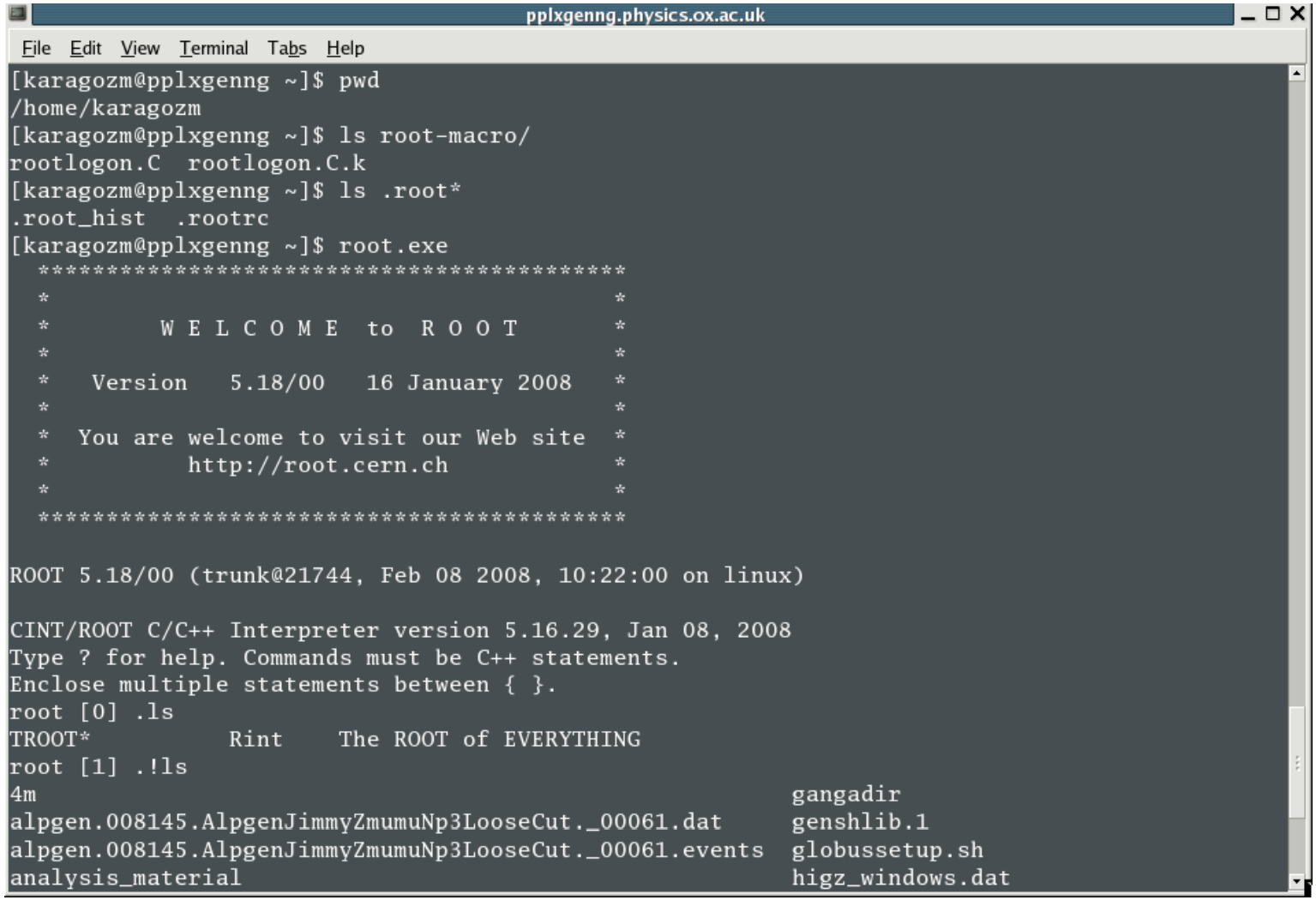
- ROOT'u etkileşimli kullanmak için
  - % root yeterli
  - ROOT logosunu görmekle vakit kaybetmek istemiyorsanız, % root.exe veya % root -l kullanın.
  - Tiyo: eger PC'nizde ROOT kütüphanesini tanımlamazsanız, bu noktada paylaşılan nesne (shared object) hatası görürsünüz.)
  - Kabuk (shell) komutları için ".!" Kullanın: örnek: [ ] .!ls
  - ROOT komut listesi için [ ] ? Veya [ ] .h
  - Oturumdan çıkmak için [ ] .q

Sekme (<Tab>) kullanım örneği:

```
root [9] TRandom *a = new TRandom(
TRandom TRandom(UInt_t seed = 65539)
TRandom TRandom(const TRandom&)
root [9] TRandom *a = new TRandom();
root [10] a
(class TRandom*)0x8da9648
root [11]
```

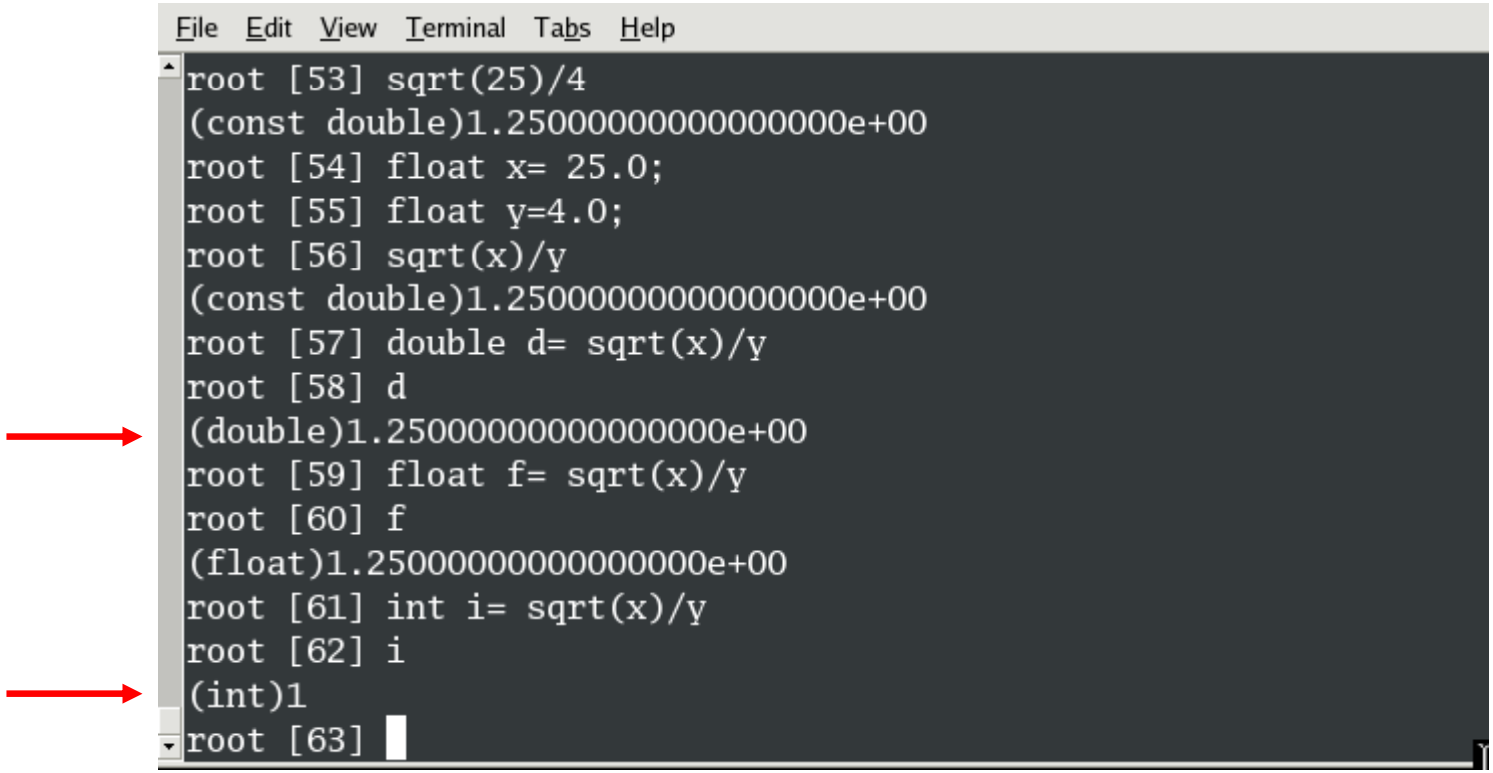
Burada <TAB>'e bastık ve sınıfımızın kuruluş şeklini gördük

ROOT'a Hoşgeldiniz: İlk oturumunuz



Basit Matematik İşlemleri

- ROOT'u önceden yüklenmiş matematik kütüphaleri sayesinde hesap makinası gibi kullanmak mümkündür (Tiyo: basit işlemler için % bc komutunu tercih ederim! ☺).
- Değişken tiplerini nasıl seçtiğiniz hem hesaplarda, hem grafik oluştururken çok önemli.



Bazı Evrensel Değişkenler ve Stiller

- ROOT genelleri: gROOT->
  LoadMacro(); ,Time(); ProcessLine(); FindObject(); Reset();
- Sistem genelleri: gSystem->
  HomeDirectory(); HostName(); ListLibraries();
- Stil seçenekleri: gStyle->
  SetOptStat(); SetTextFont();

```
[karagozm@pplxgennng ~]$ more root-macro/rootlogon.C
{
         //gROOT->SetStyle("Pub");
         gROOT->SetStyle("Plain");
          gStyle->SetPalette(1);
           // gStyle->SetFillColor(0);
            gStyle->SetFillStyle(4000);
            gStyle->SetFrameFillColor(0);
            gStyle->SetFrameFillStyle(4000);
              gStyle->SetOptStat(1111);
                gStyle->SetOptFit(111);
                gStyle->SetOptTitle(0); // do not out histogram title pad on top
```

Ufak Bir Makro Yazalım

```
{//mku
//tanimlanmamis (interaktif kullanim gibi) makro
//bir betik olarak calisir, ("global scope")
//baska bir egitselin uzerine yapilmistir
   int Nolay = 1000;
   TRandom salla;
   double toplam = 0;
   for (int i=0;i<Nolay;i++) {
      //toplam += sin(sall.Rndm());
      //eger sall daha once oturumda tanimlanmissa
      //yukardaki de calisiyor, ama sonuc yanlis!
      toplam += exp(salla.Rndm());
   }
   printf("Nolay=%d iken, toplam= %g\n",Nolay,toplam);
}
```

- bir “isimsiz makro” olan rand1.C adlı betik CINT kullanarak çalışır
- Küresel kapsamda (global scope) tanımlıdır
- [ ] .x ile çalışır

Std::cout kullanmak daha “uygun” olurdu!

```
[karagozm@pplxgenng hpf_kod]$ root -l -q rand1.C
root [0]
Processing rand1.C...
Nolay=1000 iken, toplam= 1720.35
[karagozm@pplxgenng hpf_kod]$ root -l rand1.C
root [0]
Processing rand1.C...
Nolay=1000 iken, toplam= 1720.35
root [1] .x rand1.C
Nolay=1000 iken, toplam= 1720.35
root [2] toplam
(double)1.72034840137045649e+03
root [3] .q
[karagozm@pplxgenng hpf_kod]$
```

- Tiyo: [ ] gROOT->Reset() ile global kapsamda tanımlanmış nesneleri temizleyin. Hafızada ne kaldığını her zaman bilemezsiniz!

Aynı Program İkinci Kere

```
void rand2() {
//tanimlanmis (interaktif kullanim gibi) makro
//bir betik olarak calisir,
//ama normal c++ kurallari gecerlidir.
//baska bir egitselin uzerine yapilmistir
   int Nolay = 1000;
   TRandom salla;
   double toplam = 0;
   for (int i=0;i<Nolay;i++) {
      toplam += exp(salla.Rndm());
   }
   printf("Nolay=%d iken, toplam= %g\n",Nolay,toplam);
}
```

- rand2.C adlı tanımlanmış betik için c++ kapsam kuralları geçerlidir: küresel kapsamda tanımlı değildir

```
[karagozm@pplxgenng hpf_kod]$ root -l rand2.C
root [0]
Processing rand2.C...
Nolay=1000 iken, toplam= 1720.35
root [1] .x rand2.C
Nolay=1000 iken, toplam= 1720.35
root [2] rand2()
Nolay=1000 iken, toplam= 1720.35
root [3] toplam
Error: Symbol toplam is not defined in current scope  (tmpfile):1:
*** Interpreter error recovered ***
root [4]
```

Aynı Program Üçüncü Kere

```
[karagozm@pplxgenng hpf_kod]$ more rand4.C
#include "TRandom.h"
#include <math.h>
int main() {
//onceden derlenmesi beklenen c++ kodu
//include'lara dikkat
//baska bir egitselin uzerine yapilmistir
   int Nolay = 1000;
   TRandom salla;
   double toplam = 0;
   for (int i=0;i<Nolay;i++) {
      toplam += exp(salla.Rndm());
   }
   printf("Nolay=%d iken, toplam= %g\n",Nolay,toplam);
}
```

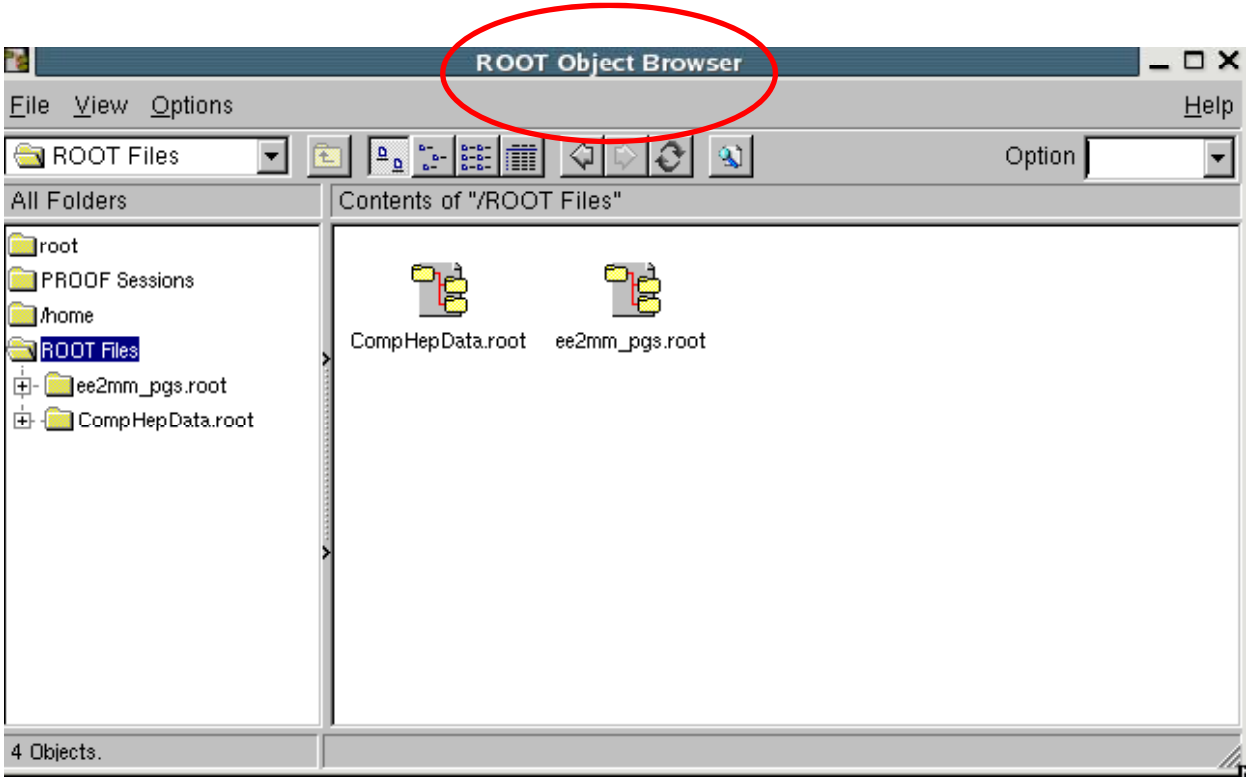
•rand4.C adlı kod ROOT kütüphanelerini bağlayarak derlenmiş bir c++ programını verir.

```
[karagozm@pplxgenng hpf_kod]$ g++ -O `root-config --cflags` -c rand4.C  -o rand4.o
[karagozm@pplxgenng hpf_kod]$ g++ -O rand4.o `root-config --libs`   -o rand4.exe
[karagozm@pplxgenng hpf_kod]$ ./rand4.exe
Nolay=1000 iken, toplam= 1720.35
[karagozm@pplxgenng hpf_kod]$
```

- Tiyo: % time rand4.exe ile programın ne kadar vakit aldığını görebilirsiniz!

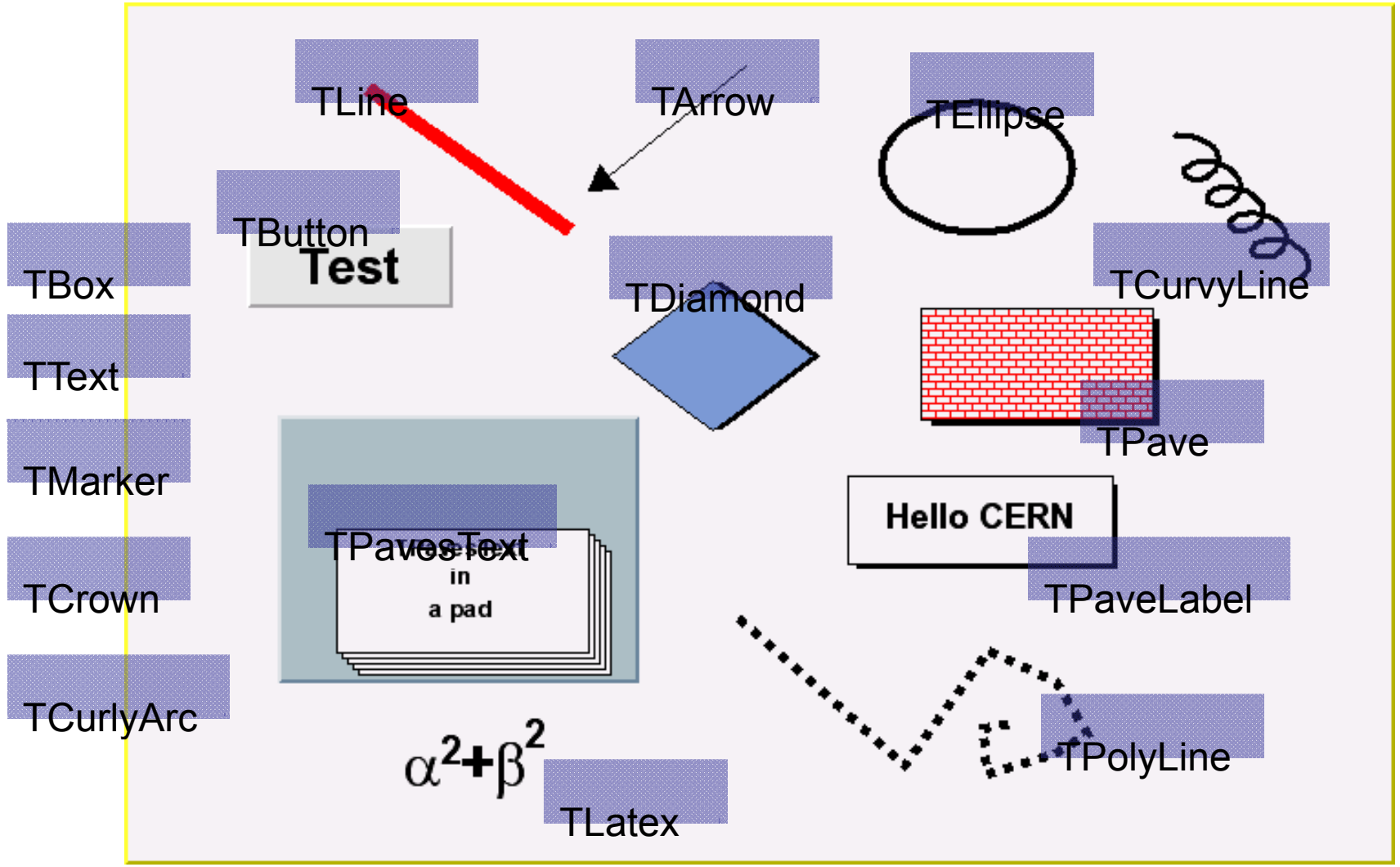
GUI Kullanımı

- ROOT nesne tarayıcı (Object Browser): TBrowser() sınıfı
  - Çağırmak için:: [ ] TBrowser tara;
  - Standard GUI'lerdeki menü hareketlerinin hepsi geçerli:
    - **Sol tıklayıp:** nesne seçmek, açmak, taşımak, boyunu değiştirmek,..
    - **Sağ tıklayıp:** metodlara ulaşmanızı sağlayan içerik (context) menüsü indirmek,..
    - **Orta tıklayıp:** üzerinde olduğunuz çizim bölgesini (kanvas/altlik) seçmek,..



Grafik Uygulamaları

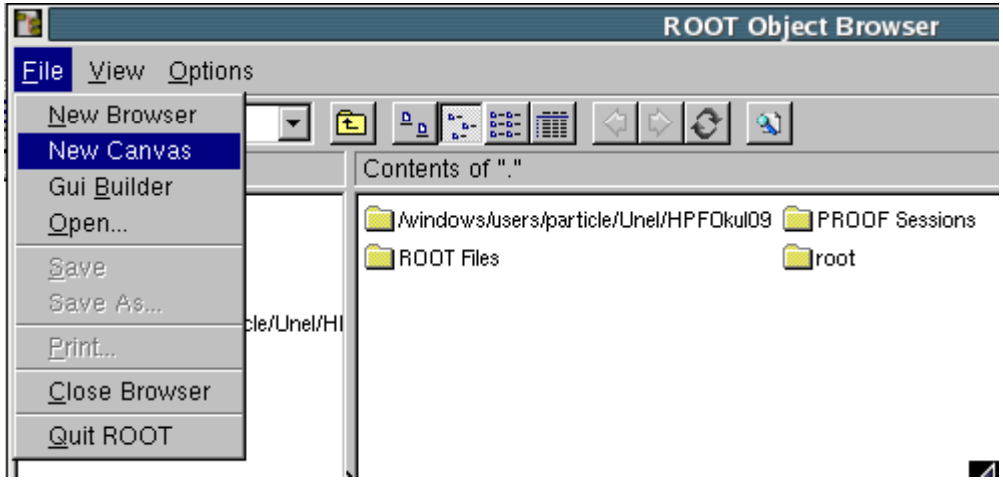
- Etkileşimli menüler ve komutlar ile temel ilkseller (primitives) ile bir sürü grafik uygulama yapılabilinir. (R. Brun'den)

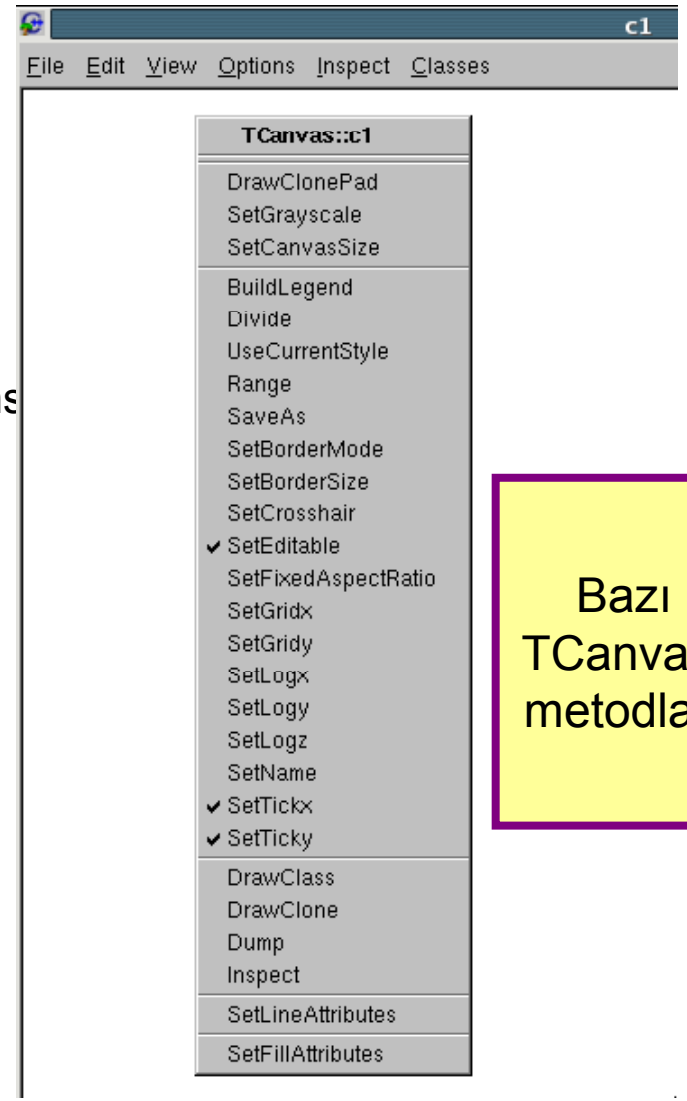


TCanvas Temelleri

- Kanvas üzerinde çizim yapacağınız penceredir.
- Çağırmak için:
  - Nesne tarayıcısı
  - Kurucu (Constructor) ile:

    TCanvas *benimBezim = new TCanvas(cons
    char* name, const char* title, Int_t wtopx,
    Int_t wtopy, I nt_t ww, Int_t wh);
  - Direk çizmek istediğiniz değişkenin üzerine tıklayarak (öntanımlı "c1" isimli isimli kanvası yaratır)

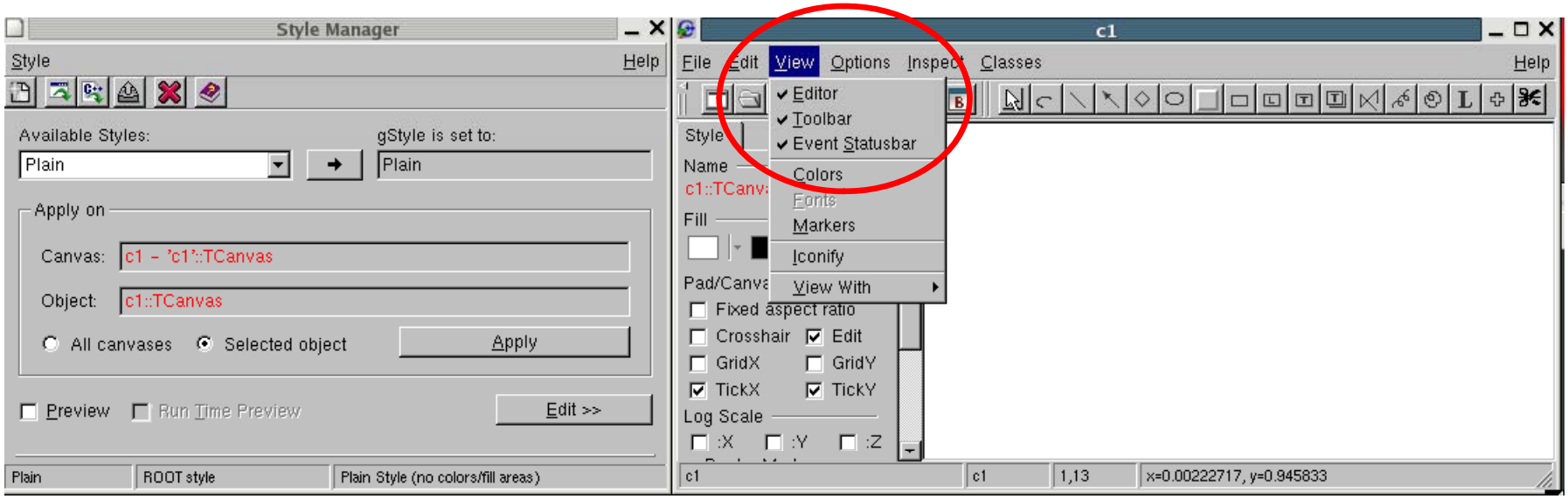




Bazı TCanvas metodları

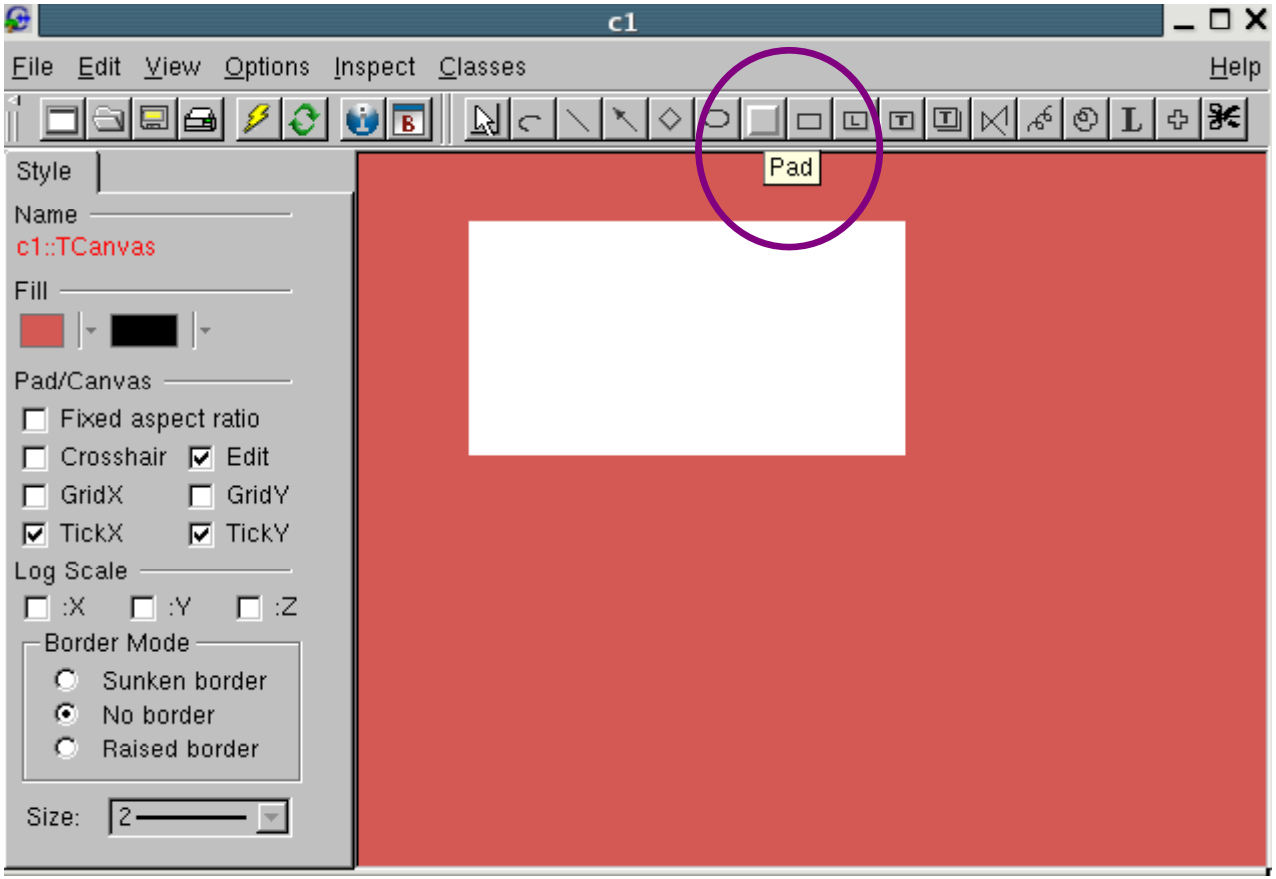
TCanvas Ayarları

- Menuden Style Manager (stiliniz), Editor (stilinizin grafik özellikleri), Toolbar (nesne eklemek, işlem yapmak), Event Status Bar (x,y koordinatlarını görmek) çağırabilirsiniz.
- İçerik menüsü de her zaman kanvas alanında sağ tıklayarak çağrılınabilinir
- Not: Ben ROOT'a başladığımda GUI'de bunların bir kısmını yapmak mümkün değildi. ROOT'un değişkenliğine hep hazırlıklı olun!



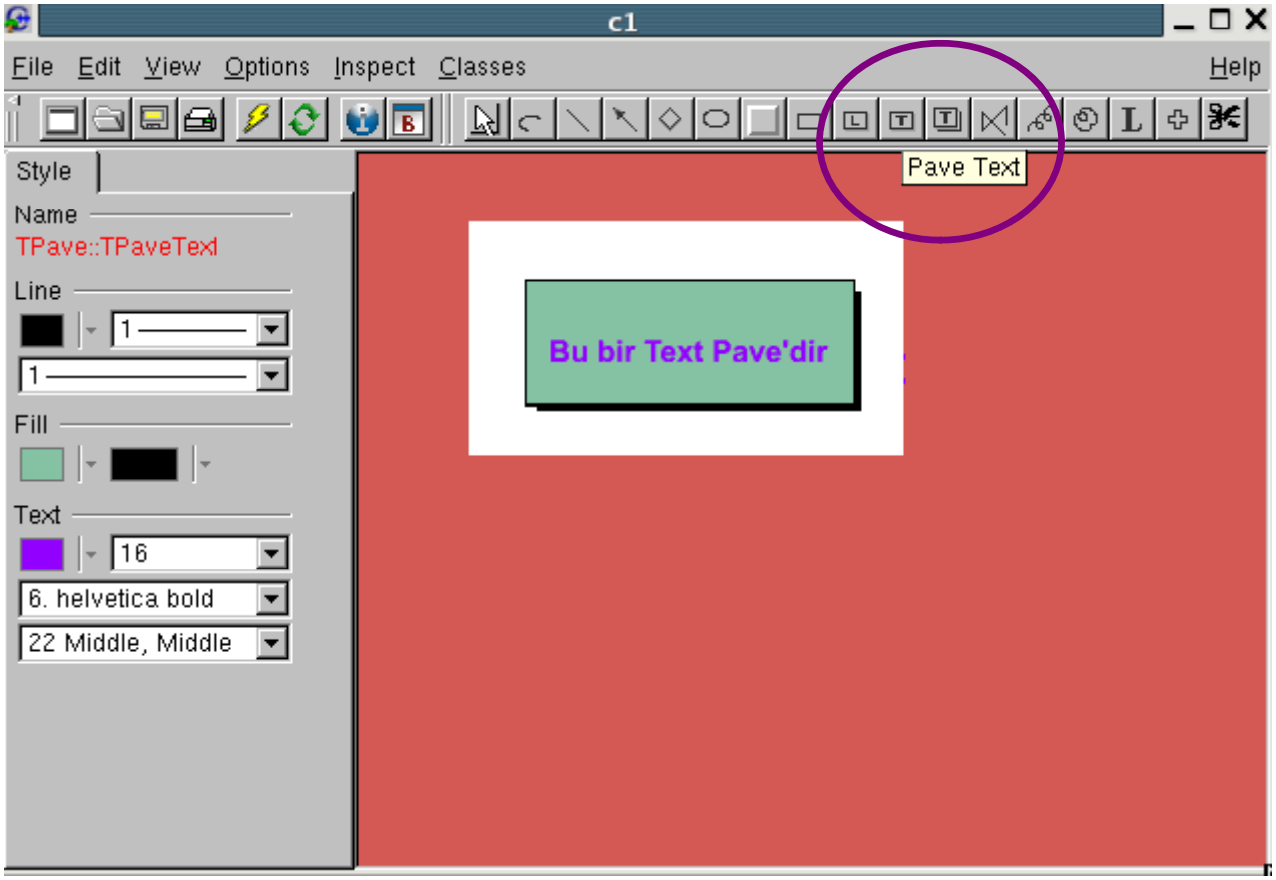
Kanvasta Basit İşlemler

- Grafik ilksellerini kullanarak bir panel yaratalım
- Komut satırından **c1->SetFillColor(kColor);**



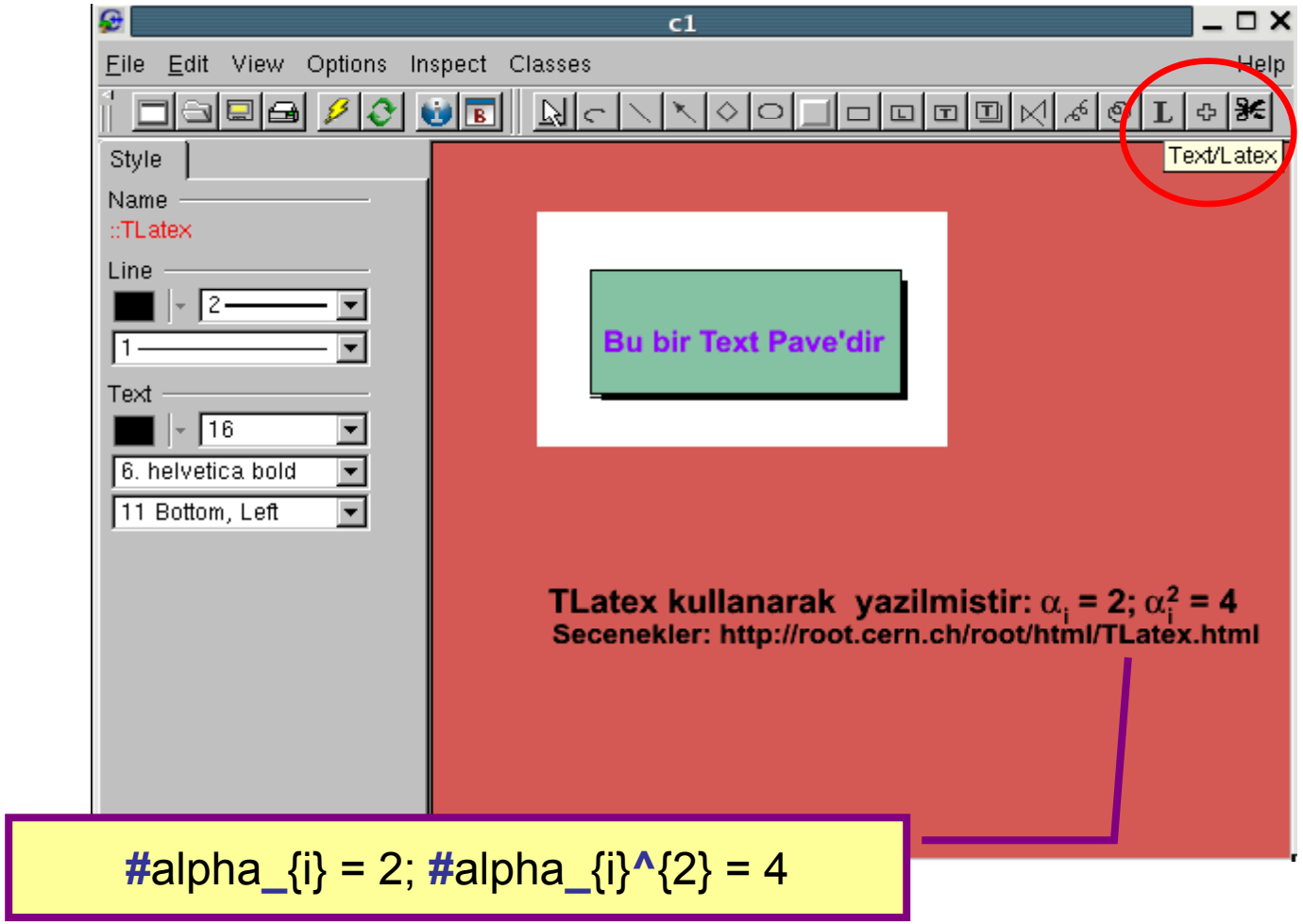
Kanvasta Basit İşlemler

- Grafik ilksellerini kullanarak parke yazısı ekleyelim



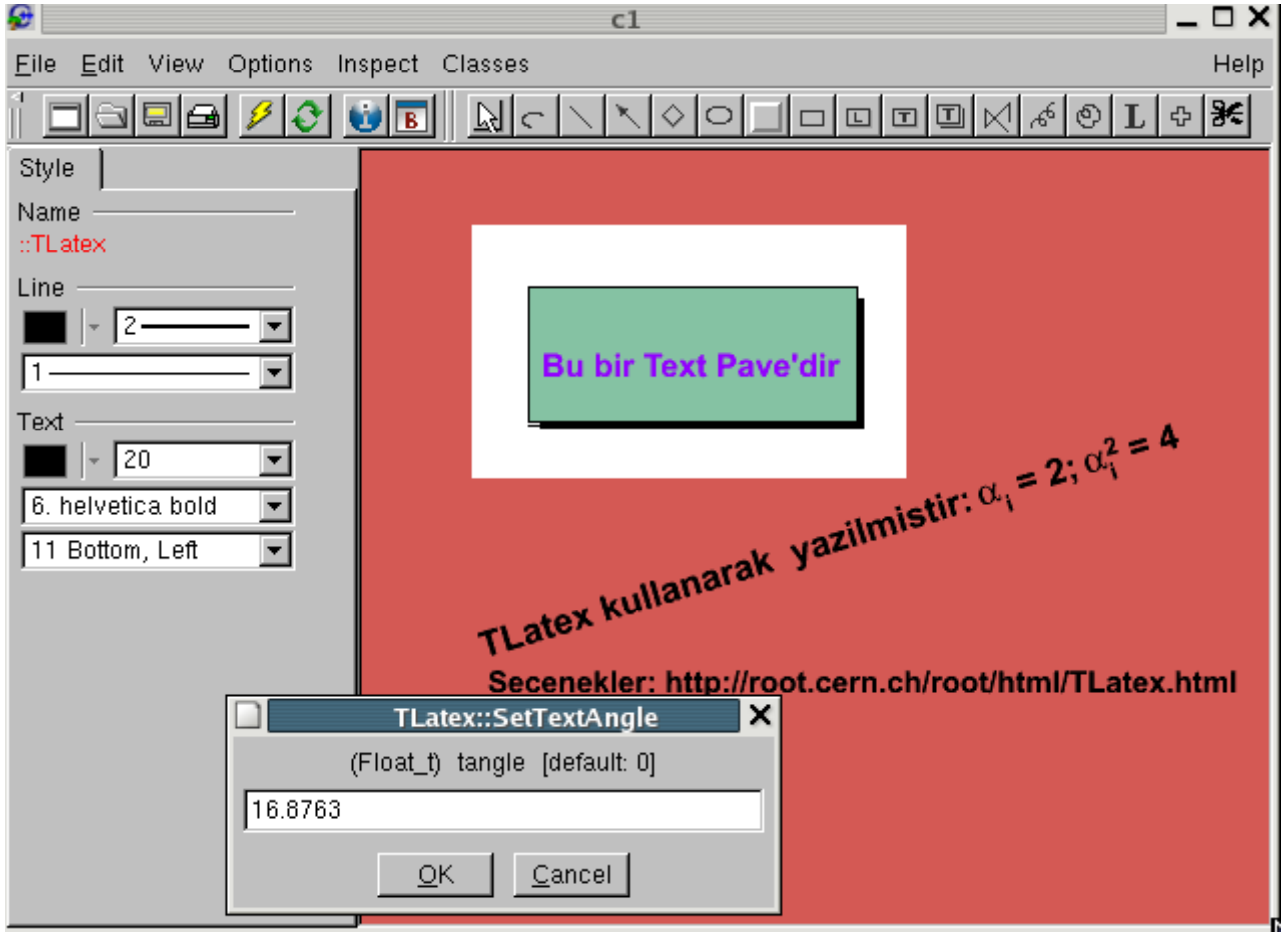
Kanvasta Basit İşlemler

- Grafik ilksellerini kullanarak latex ile karışık bir matematik ifadesi ekleyelim



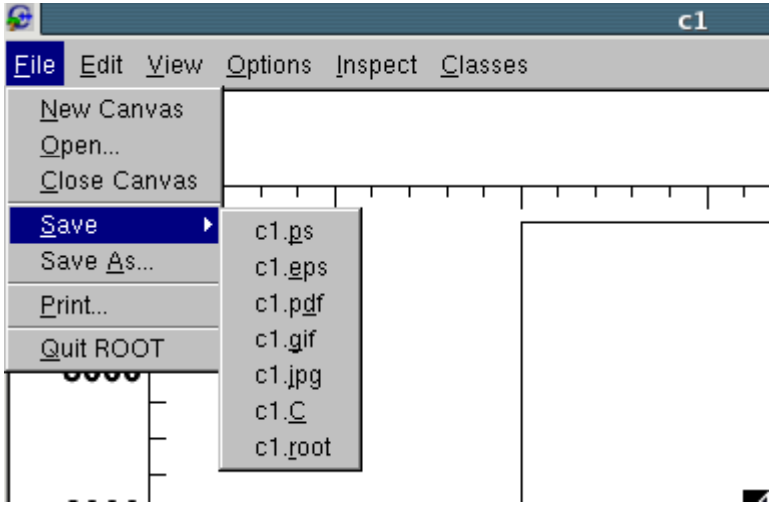
Kanvasta Basit İşlemler

- Nesne üzerine tıklayıp, kıpırdatmak, çevirmek, çok kolay. Özelliklere belirli değerler vermek istiyorsanız, içerik menüsü bunu sağlar.



Kanvası Dosya Olarak Kaydetmek

- ROOT TCanvas'in kaydetme seçeneğini "File" menüsünde açar.
- Yarattığınız grafiği .eps/.jpg .. seçeneklerini kullanarak, bigisayarınıza kaydedebilirsiniz.




```
[karagozm@pplxgenng hpf_data]$ more c1.C
{
//=========Macro generated from canvas: c1/c1
//=========  (Tue Jan 20 13:26:19 2009) by ROOT version5.18/00
   TCanvas *c1 = new TCanvas("c1", "c1",379,97,700,500);
   gStyle->SetOptFit(1);
   c1->Range(0,0,1,1);
   c1->SetFillColor(0);
   c1->SetBorderMode(0);
   c1->SetBorderSize(2);
   c1->SetTickx();
   c1->SetTicky();
   c1->SetFrameFillStyle(4000);
   c1->SetFrameBorderMode(0);
   c1->Modified();
   c1->cd();
   c1->SetSelected(c1);
}
```

Bir grafikle tekrar çalışacaksanız, (renk, yazı değiştirme), ROOT'un bunu çizerken uyguladığı işlemleri makro olarak (.C) kaydedebilirsiniz. İçerik "hardcoded" bir şekilde saklanacağı için, makale yazarken işinize yarayabilir!

Tiyo:Kanvasınızı kapatmadan yarattığınız makronun geçerliliğine bakın, ROOT'a güven olmaz!

Panelimize bakalım

- Bir önceki sayfadaki paneli kaydettim! İçine bakip, sınıflar nasıl çağırılmış, nesneler nasıl yaratılmış görelim..

```
  TPaveText *pt = new TPaveText(0.1308749,0.2265843,0.8852389,0.752873,"br");
  pt->SetFillColor(30);
  pt->SetTextColor(51);
  pt->SetTextSize(0.136272);
  TLine *line = pt->AddLine(0,-0.02421222,0,-0.02421222);
  TText *text = pt->AddText("Bu bir Text Pave'dir");
  pt->Draw();
  c1_1->Modified();
  c1->cd();
tuval_stillleri.C                                              22,4
```

```
     tex = new TLatex(0.1346578,0.3419069,"TLatex kullanarak  yazilmistir:
 #alpha_{i} = 2; #alpha_{i}^{2} = 4");
  tex->SetTextAngle(10.87626);
  tex->SetLineWidth(2);
  tex->Draw();
     tex = new TLatex(0.1390728,0.2979275,"Secenekler: http://root.cern.ch
/root/html/TLatex.html");
  tex->SetTextSize(0.04145078);
  tex->SetLineWidth(2);
  tex->Draw();
  c1->Modified();
tuval_stillleri.C                                  46,18          90%
```

Bir Dosyayı Açmak

- Etkileşimli durumda, nesne yaratarak:

```
root [3] TFile benimDosyam("ee2mm_pgs_events.root")
```

  - [ ] benimDosyam.ls();

```
TFile**         ee2mm_pgs_events.root
 TFile*         ee2mm_pgs_events.root
  KEY: TTree    LHCO;1  Analysis tree
root [7]
```

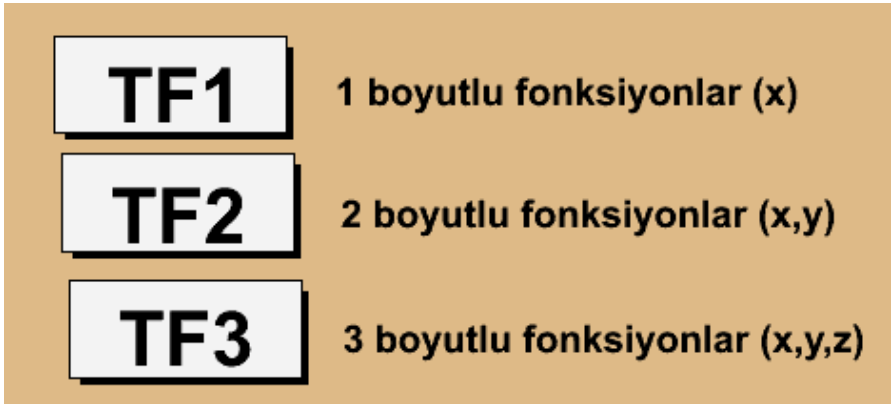
- Eğer dosyayı yüklerken nesne veya işaretçi (pointer) yaratmazsanız, ROOT öntanımlı işaretçiyi yaratır:

```
[karagozm@pplxgenng data]$ root.exe -l ee2mm_pgs_events.root
root [0]
Attaching file ee2mm_pgs_events.root as _file0...
Warning in <TClass::TClass>: no dictionary for class TRootEvent is available
Warning in <TClass::TClass>: no dictionary for class TRootPhoton is available
```

```
root [2] TFile *_file0 = TFile::Open("ee2mm_pgs_events.root")
```

1-2-3 Boyutlu Fonksiyonlar

- ROOT'ta önceden tanımlanmış (TMath::) veya tanımlayacağınız fonksiyonları çizmek, bunlardan histogram doldurmak, uyum eğrisi olarak kullanmak mümkündür.



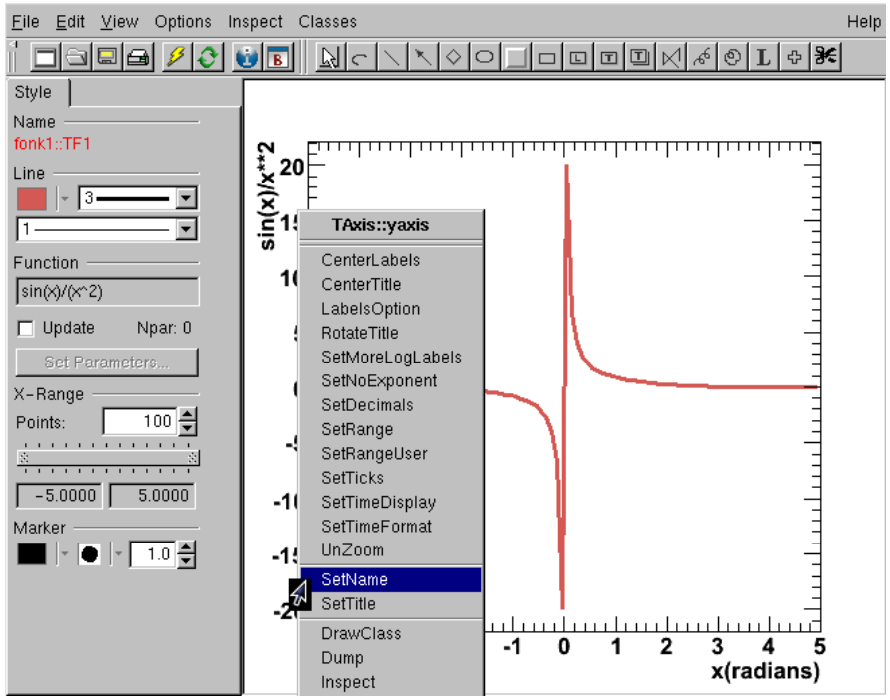
TF1(const char* name, const char* formula, Double_t xmin=0, Double_t xmax=1)

- Aşağıdaki fonksiyon tipleri yaratılınabilinir:
  - değişken kullanan ve parametre gerektirmeyen ifade
  - değişken kullanan ama parametre de isteyen ifade
  - Parametreli genel bir c++ fonksiyonu

Öntanımlı Fonksiyon Çizdirmek

- Tek boyutlu bir fonksiyon: sin(x)/x**2

```
TF1 *fonk1 = new TF1("fonk1","sin(x)/(x^2)",-5,5);
fonk1->SetLineColor(50);
fonk1->SetLineWidth(3);
fonk1->GetXaxis()->SetTitle("x(radians)");
fonk1->GetYaxis()->SetTitle("sin(x)/x**2");
fonk1->Draw("");
```



- Kaydedin:

```
c1->SaveAs("basitFonk.eps");
```

- Tiyo: Daima x ve y eksenlerinizin isimlerini (başlık) koyun!

Kendi Fonksiyonumuzu Yaratalım

- C++ kurallarını takip ederek, bir Gaussian fonksiyon yazalım:

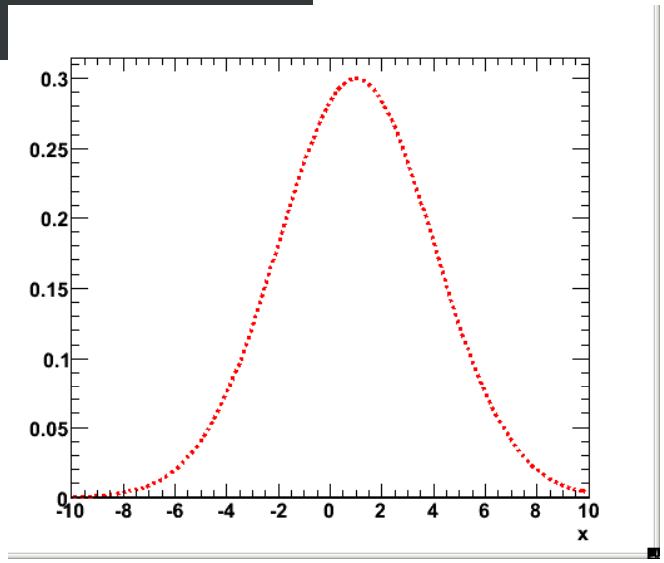
```
Double_t basitGaus(Double_t *x, Double_t *par)
{
  return (par[0]*exp(-0.5*((x[0]-par[1])/par[2])**2));
} //basitGaus.C
```

basitGaus.C [+]

- Sonra ROOT oturumunda, bir örnek yaratalım

```
// eger bunu interaktif kullanacaksaniz .L basitGaus.C de yeterli!
gROOT->LoadMacro("basitGaus.C");
TF1 *bGaus = new TF1("benimGaus",basitGaus,-10,10,3);
bGaus->SetParameters(0.3,1.,3.);
```

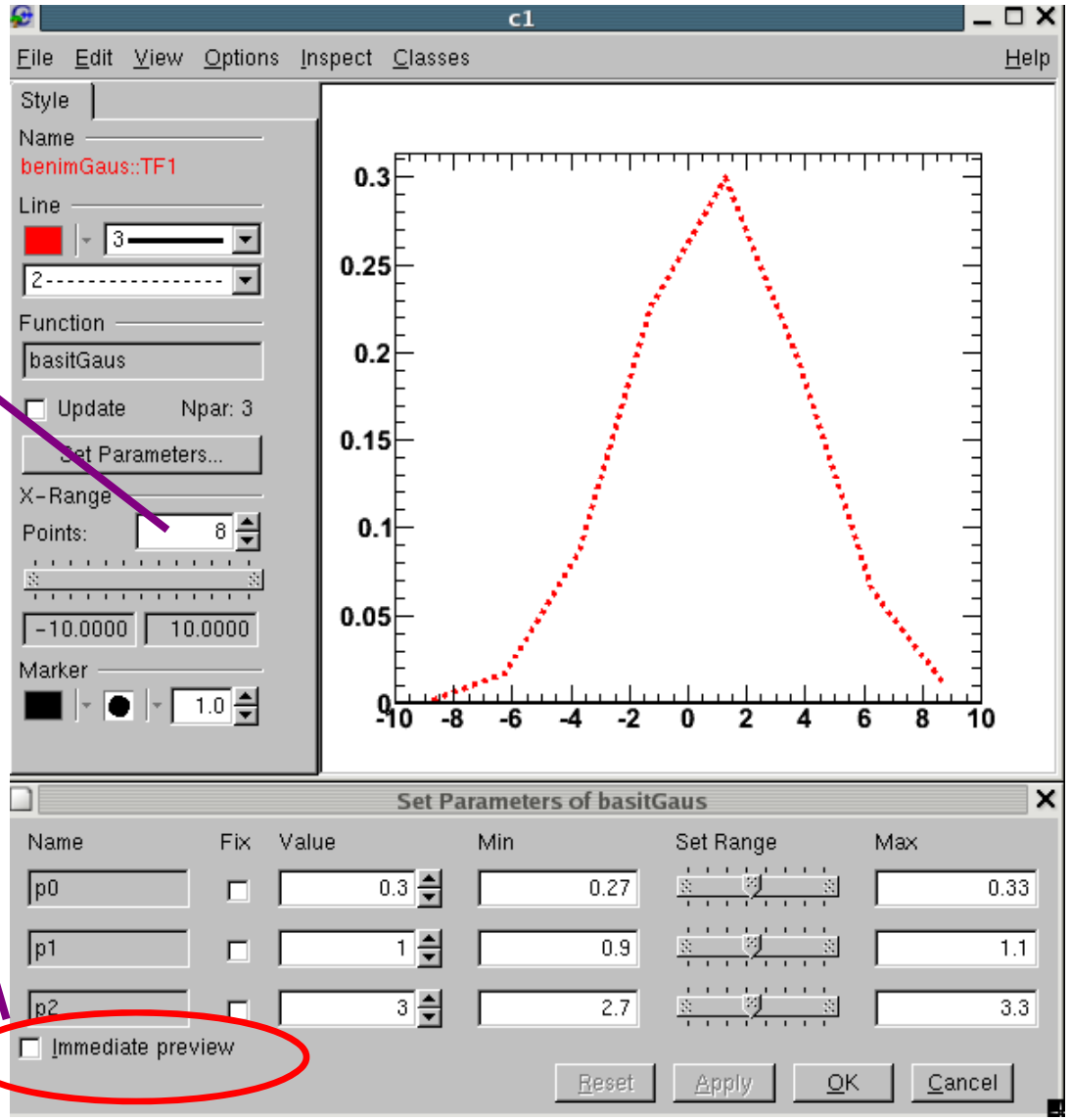
(par)ametre sayısını burada veriyoruz



Kendi Fonksiyonumuzu Değiştirelim

Rasgele nokta sayisini azalttik

Parametreleri etkileşimli değiştirmek mümkün



TGraph(): 2 Boyutlu Çizim

- TGraph() bildiğimiz iki tane *n* tane veri noktası olan iki dizinin (*x,y*) birbirine karşı çizildiği sınıftır. Analiz yapmanıza değil, sonucunu sunmanıza yarar.

TGraph(Int_t n, const Float_t* x, const Float_t* y)

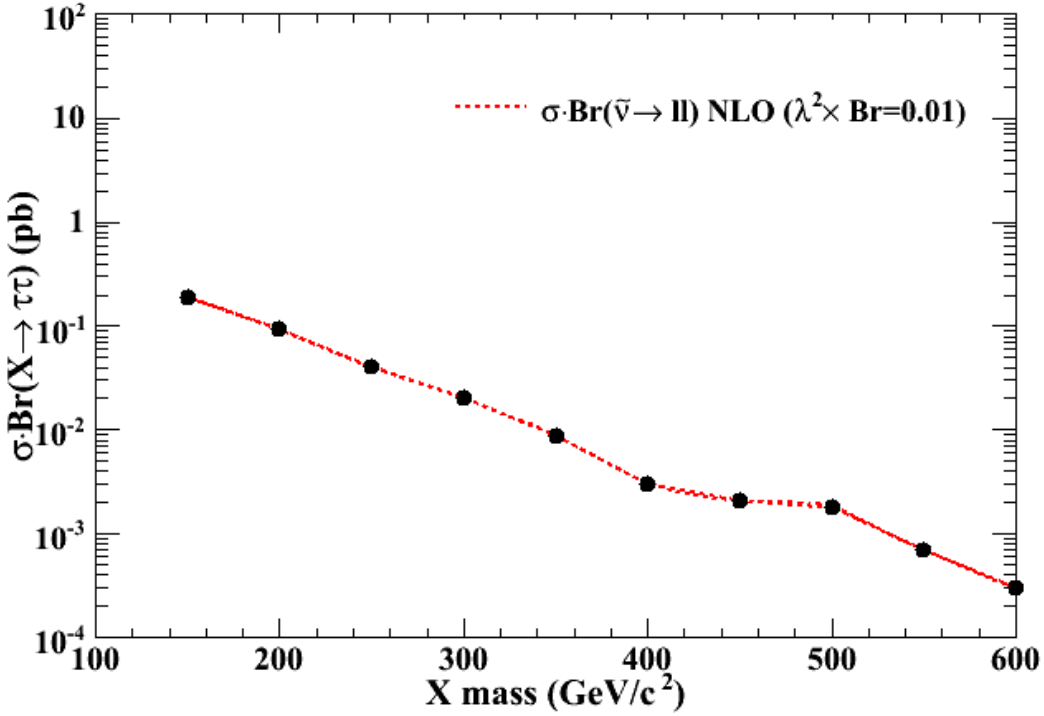
Baglar ROOT'un veri tiplerine goturur

- Eğer verimizin hata paylarını çubuk olarak göstermek istersek, bunları da dizi olarak baska bir grafik sinifina vermeliyiz:

TGraphErrors(Int_t n, const Float_t* x, const Float_t* y, const Float_t* ex = 0, const Float_t* ey = 0)

Bir TGraph Örneği

- Aşağıdaki resim Tevatron'da bir parçacığın tesir kesidini kutlesine göre (y, x) göstermektedir. Bu resmi basitçe nasıl elde ettiğimize bakalım.



TGraph Örneği Makrosu - I

```
{
  //MKU. basit bir TGraph ornegidir
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  gROOT->ForceStyle();
  //stillerimizi degistirelim
  gStyle->SetLabelFont(22);
  gStyle->SetOptStat(0);
  gStyle->SetOptTitle(0);
  gStyle->SetTextSize(0.080);

  // bir parcacigin kutlesine gore tesir kesidi
  //rakamlar gercekci degildir
  Float_t Mass[10] = {150.0, 200.0, 250.0, 300.0, 350.0, 400.0, 450.0, 500.0, 550.0, 600.0};
  Float_t SigmaStau[10]={0.209 ,0.063 ,4.04e-2 ,2.01e-2 ,8.92e-3 ,3.09e-3 ,2.08e-3 ,1.83e-3 ,
6.95e-4 ,1.01e-4};

  // grafigimizi yaratalim
  TGraph* gSigmaStau = new TGraph(10, Mass,SigmaStau);

  // sablon histogramimizi yaratalim
  TH2F *hr1= new TH2F("hr1", "", 500, 100, 600, 100, 0.0001, 100);

  c1 = new TCanvas("ditau_plot", "ditau", 300, 10, 700, 500);
  c1->SetLogy();
  c1->Draw();
  c1->cd();
```

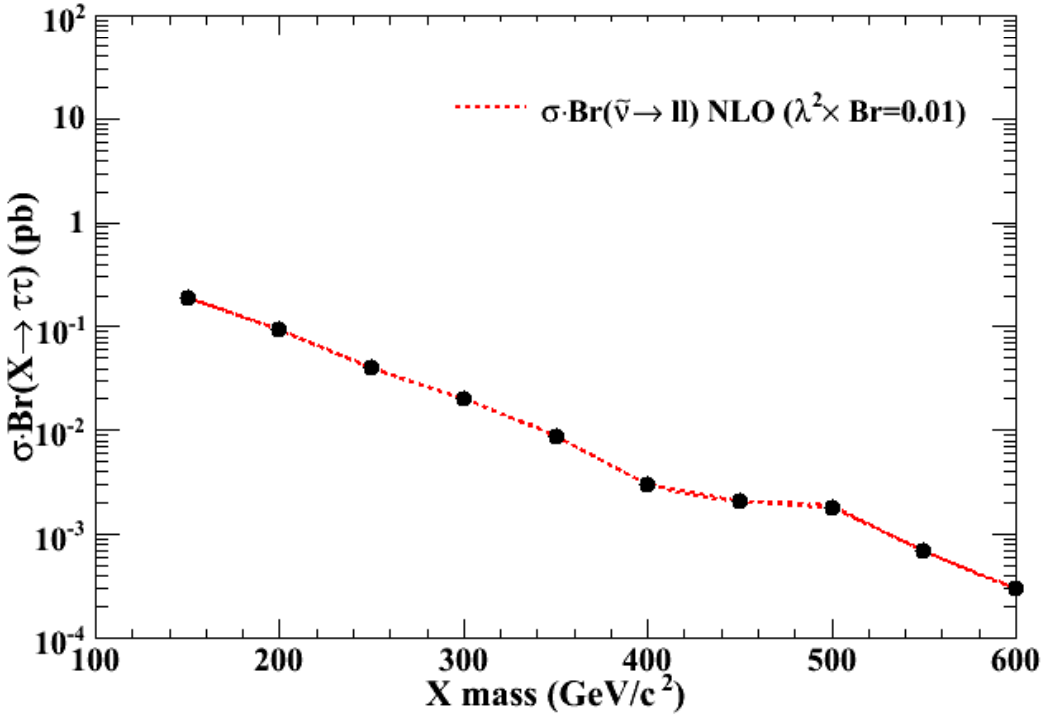
TGraph Örneği Makrosu - Devam

```
  //sablon histogramimizi cizelim
  hr1->Draw();
  hr1->GetXaxis()->SetNdivisions(505);
  hr1->GetYaxis()->SetNdivisions(505);
  hr1->GetXaxis()->SetTitleOffset(0.8);
  hr1->GetYaxis()->SetTitleOffset(0.8);
  hr1->GetXaxis()->SetTitleSize(0.05);
  hr1->GetYaxis()->SetTitleSize(0.05);
  hr1->GetXaxis()->CenterTitle();
  hr1->GetYaxis()->CenterTitle();
  hr1->GetXaxis()->SetTitleFont(22);
  hr1->GetYaxis()->SetTitleFont(22);
  hr1->SetXTitle("X mass (GeV/c^{ 2})");
  hr1->SetYTitle("#sigma#upointBr(X#rightarrow #tau#tau) (pb)");

  //grafigimizi cizelim. CLP'yi referans rehberinden bulabilirsiniz
  gSigmaStau->Draw("CLP");
  gSigmaStau->SetLineWidth(2);
  gSigmaStau->SetLineColor(2);
  gSigmaStau->SetLineStyle(2);
  gSigmaStau->SetMarkerStyle(20);

  //cizdigimiz egrinin neye ait oldugunu belirtelim
  TLegend *legc5 = new TLegend(0.40, 0.67, 0.75, 0.89);
  legc5->SetTextFont(22);
  legc5->SetTextSize(0.044);
  legc5->SetFillColor(0);
  legc5->SetLineColor(0);
  legc5->AddEntry(gSigmaStau,"#sigma#upointBr(#tilde{#nu}#rightarrow ll) NLO (#lambda^
{2}#times Br=0.01)", "L");
  legc5->Draw();

  //kanvasi kaydet
  c1->SaveAs("tgraph_ornegi.png");
}
```

tgraph_ornegi.C 27,18 Bot

Grafigimizi Kaydedelim

- TCanvas::SaveAs() ile grafiğimizi .png olarak kaydettik. Artık makalemizi PRL'e gönderebiliriz (.eps'li halini!) ☺



Histogramla Analize Giriş

- Histogramlar data analizinde önemli yer tutarlar
- Elinizde aynı tipten bir veri olsun. Bunun nasıl bir dağılım gösterdiğini bilmek, karakterlerini anlamak isteyelim (mesela Z bozonundan çıkan muonların momentum dağılımı)
- Bu verinin alt ve üst değerlerini biliyorsak, bu alt ve üst değeri (*xlow, xup*) aralığını n bölmeye bölüp (*n bins*), her bir bölmeye düşen değerleri teker teker doldurarak histogramımızı elde edebiliriz.
- Histogramları birbirleriyle işleme tabi tutabiliriz (Add(), Divide(),..)

```
TH1 h1(const char* name, const char* title, Int_t nbinsx,
Double_t xlow, Double_t xup);
```

kur

```
h1-> Fill(Double_t x, Double_t w);
```

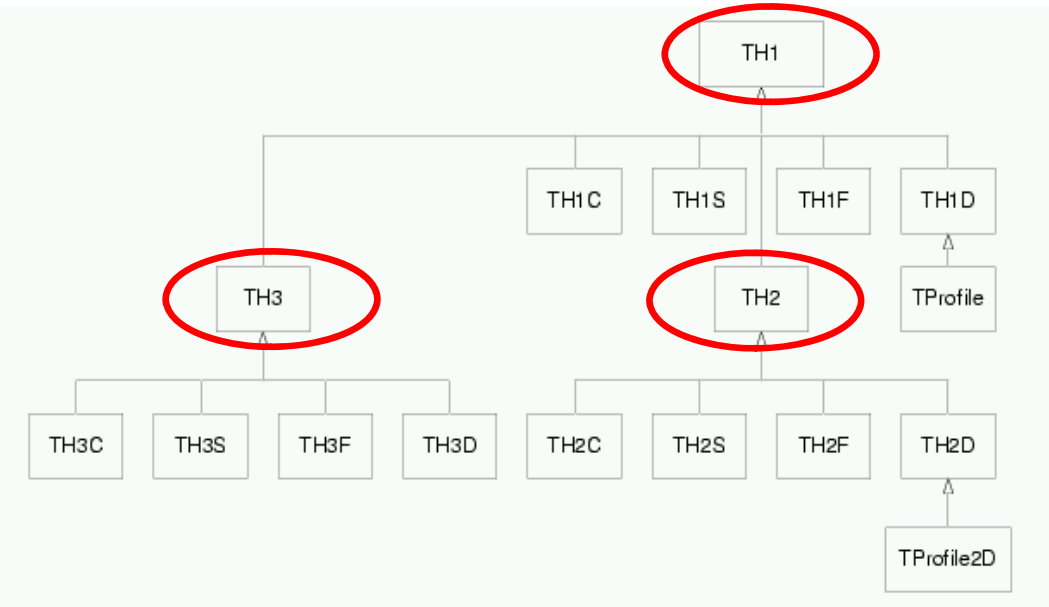
doldur

```
h1-> Write();
```

yaz

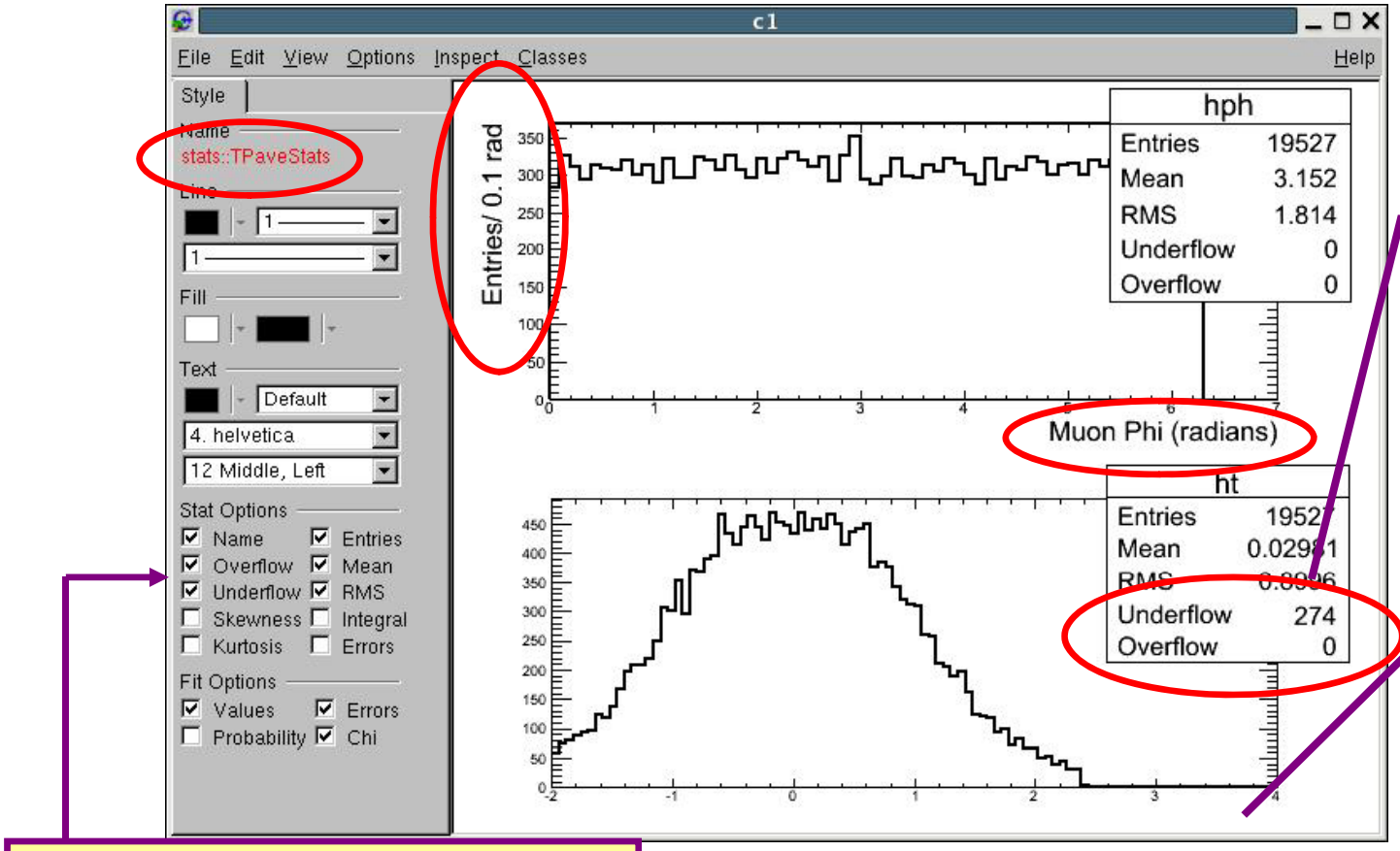
Histogram Tipleri

- Histogram tipinizi iyi seçin. İş yaparken hafıza tasarrufu önemlidir!
  - TH1C : 1 byte/girdi
  - TH1S : 1 kısa inteğer/girdi
  - TH1İ : 1 inteğer/girdi
  - TH1F : 1 float/girdi
  - TH1D : 1 double/girdi
- Profil histogramları (TProfile, TProfile2D): Bir eksenindeki değerlerin ortalamasını, RMS'ini göstermek için kullanırız. Korelasyon/eğilimler görmek için iyidir.



Histogram Ayarları

- Histogramınızın okunulur, anlaşılır olması ve doğru bilgiler içermesi çok önemlidir.
  - Bir fizik sürecinden (mesela Z' ağır rezonansı) çıkan iki muonun phi, eta değerlerinin dağılımını gösteriyoruz.



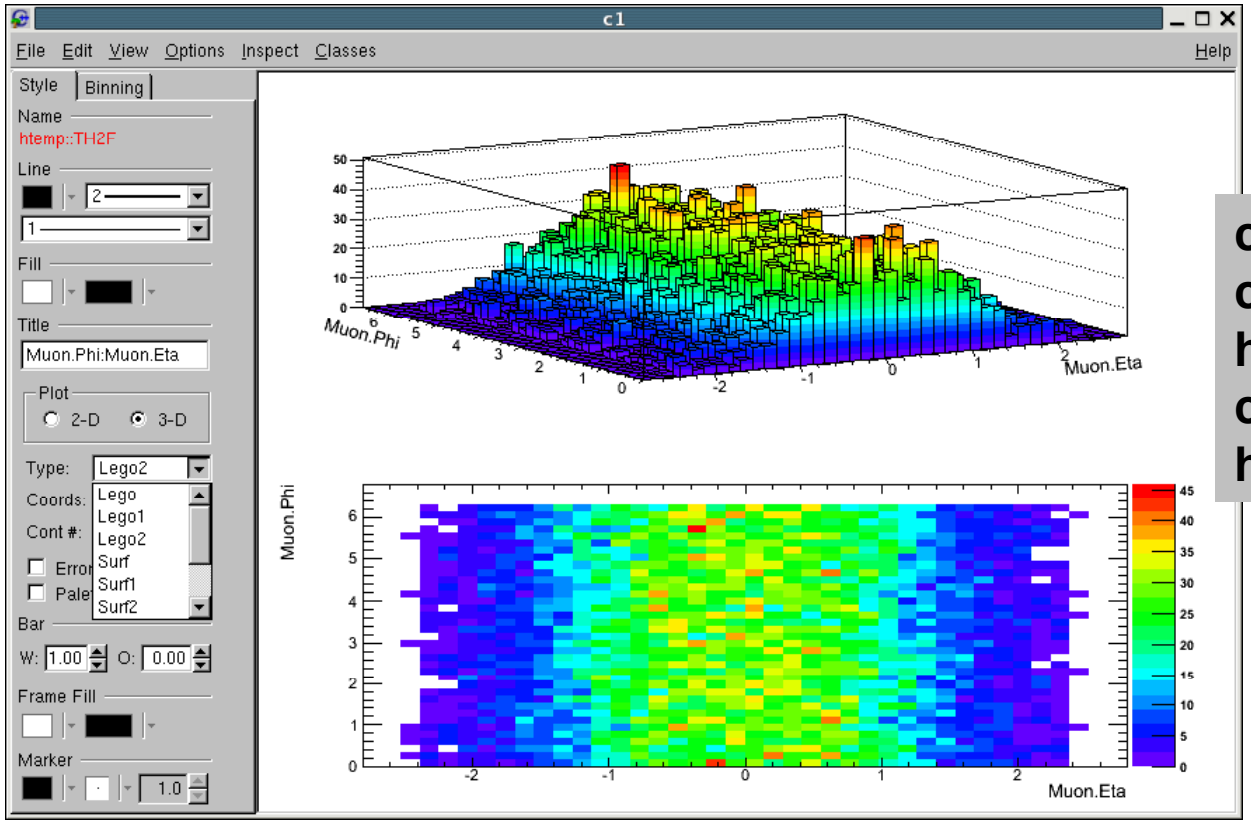
Histogram butun veriyi gostermiyor! (bazen istenilen birseydir)

Eksen basligi yok!

gStyle->SetOptStyle(111111);

Histogram Çizme Seçenekleri

- 2 ve 3 boyutlu histogramlarınızı değişik usüllerde sunmanız mümkündür.
  - Ağır bir parçacıktan çıkan muon ikilileri genelde dedektörden ışın borusuna dik açılarda çıkarlar (eta~0). Bunu eta-phi uzayında gösterebilirsiniz.




```
c1->Divide(1,2);
c1_1->cd();
htemp->Draw("LEGO");
c1_2->cd();
htemp2->Draw("COLZ");
```

Eğri Uydurmak (Fitting a Function)

- THn ve TFn yaratmayı öğrendiniz. Bir sonraki adım histogramlarınıza ve grafiklerinize belirli değerler arasında (*xmin, xmax*) parametrize önceden tanımlanmış veya tanımlanmamış fonksiyonlar uydurmak (*fname*).

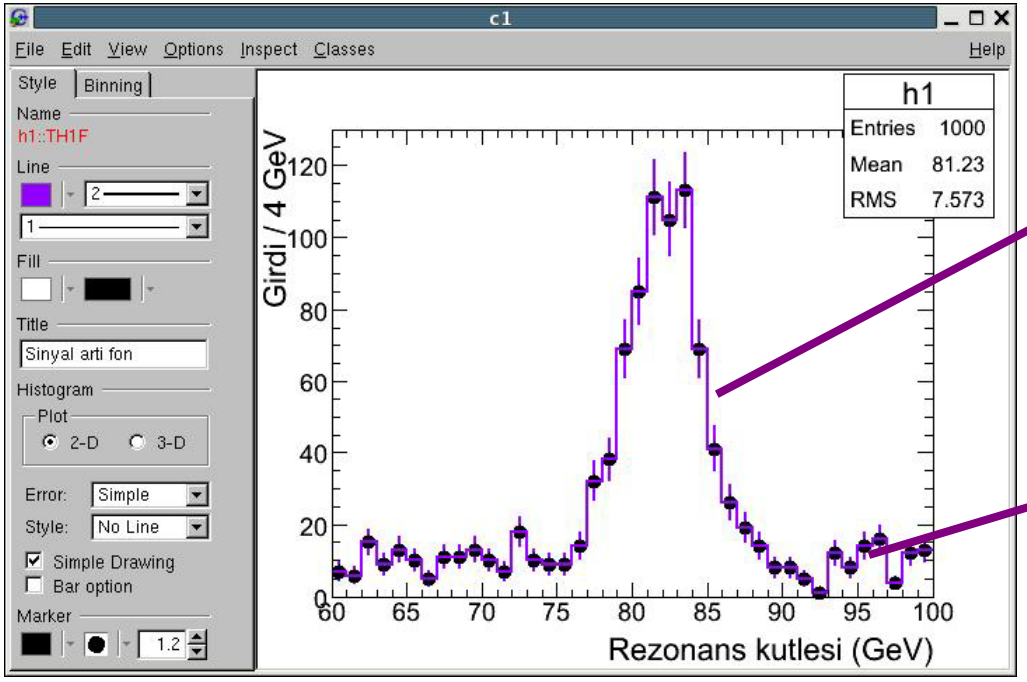
TH1::Fit(***const char**** fname, ***Option_t**** option, ***Option_t**** goption, ***Axis_t*** xmin, ***Axis_t*** xmax)

- En basit uyum eğrisi bulma yöntemi uyum fonskiyonunuzun χ2 değerini enküçültmektir (minimisation).
- Tiyo: Uyumun güzelliğine (goodness of fit) en basit olarak χ2**/NDF ~ 1** ile karar verilir. Bir eğriyi histogramınıza uydurduğunuz zaman mutlaka bu değere bakmalısınız. Makronuzda yapmayı unutmayın:

```
gStyle->SetOptFit(1111);
```

Çoklu Uyum Eğrileri: Bir Fizik Ölçümü Yapalım

- Amacımız elimizdeki veriden bir rezonansın değerlerini (mean, sığma) çekmek.
  - Ne yapacağız? Eğri uyduracağız!
  - Neye? Dağılımımıza
  - Ne şekilde? Sinyal ve gürültüyü en iyi şekilde tanımlamaya çalışarak



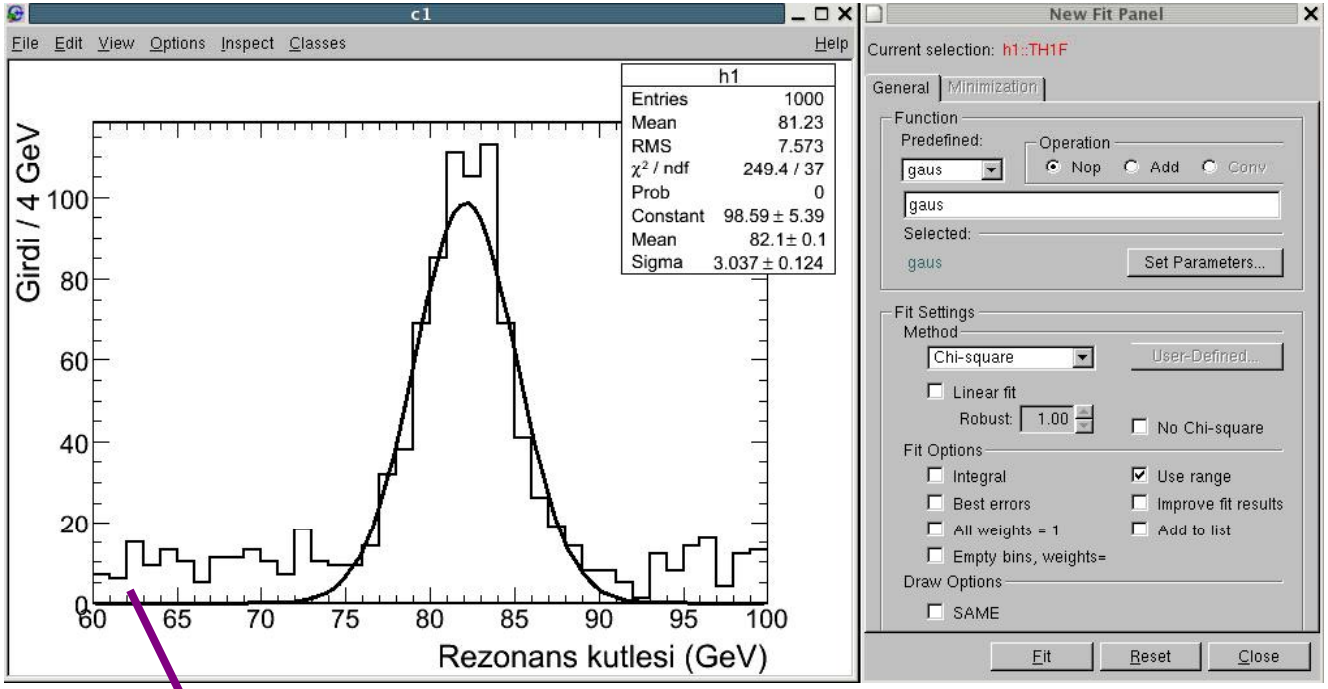
Hata çubukları için: h1->Draw("eSame");

Ilgilendigimiz sinyal

Gurultu (fon)

Sinyal ve Fonu Tahmin Edelim

- GUİ kullanarak Gaus eğri uydurduk.
  - Aslında bu bir rezonans olduğu için ölçüm çözünürlüğü yüzünden Gaussian ile bulanmış bir Breit Wigner fonksiyonu olmalıdır. Ben tembellik edip, basit bir Gaussian dağılıma göre histogram yarattım ☺



Uyumumuz guzel degil (X^2/ndf >> 1). Salt Gaussian kuyruklara guzel uymuyor. ☹

Eğrimizi uyduralım

- Histogramımıza sırf Gaus yetmedi. Sabit gibi görünen bir fonumuz var. O zaman iki fonksiyonun toplamını uydurmaya çalışalım:
  - Gaussian: “Gaus”: [0]*exp(-0.5*((x-[1])/[2])**2)/(sqrt(2*pi)*[2]))
  - Katlıterim: “pol0”

```
TF1 *fSinyal = new TF1("fSinyal","gaus",60.,100.);
TF1 *fFon = new TF1("fFon","pol0",60.,100.);
TF1 *fToplam = new TF1("fToplam","gaus+pol0(3)",0.,30.);
fToplam->SetParNames("Yukseklik","Mean","Sigma","Fon");
fToplam->SetParameters(100, 85, 2.5, 10);

h1->Fit("fToplam", "", "", 60., 100.);
f1->Draw("Same");
```

0,1,2 ci parametreler gaus’a gitti, 3 ten basla

Tahmini başlangıç değerlerini ver

Eğrimizi uyduralım

- Makromuz böyle çalıştı:

```
Processing gaus_konusma.C...
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
Toplam girdi 797
 FCN=86.2931 FROM MIGRAD    STATUS=CONVERGED     128 CALLS         129 TOTAL
                     EDM=7.7163e-09    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Yukseklik    1.03943e+02   5.57879e+00   1.98420e-02   2.71754e-05
   2  Mean         8.20970e+01   1.12706e-01   5.13710e-04  -7.75423e-05
   3  Sigma        2.47173e+00   1.02968e-01   3.49961e-04   1.33766e-03
   4  Fon          6.74270e+00   5.14941e-01   2.19234e-03   6.93752e-05
1. uyum parametresi hatasi 5.578795
root [1]
```

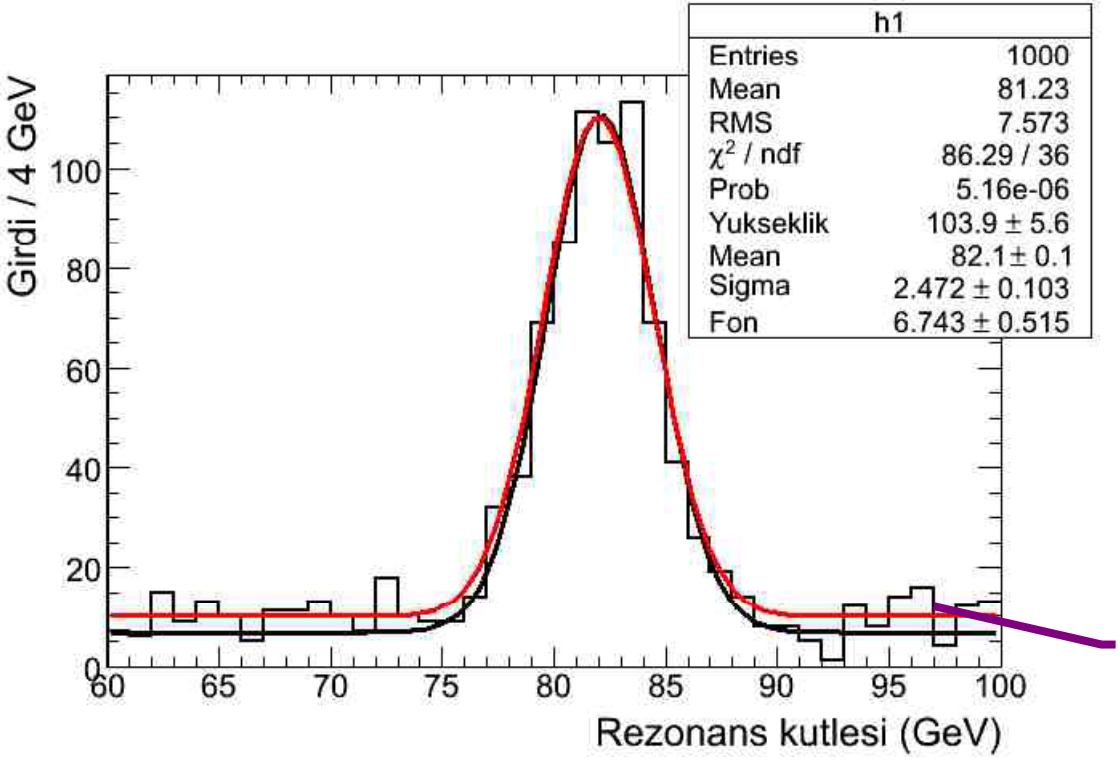
- Arada tepedeki girdileri saymak istedik

```
Float_t toplam=h1->Integral(12,30);
printf("Toplam girdi%d\n", toplam);
```

- Uyum fonksiyonunuz histogram nesnenize ait olur. TH1::GetFunction() ile sonuçlarınıza ulaşmak mümkündür.

```
TF1 *uyumSonucu = h1->GetFunction("fToplam");
Float_t par1hata = uyumSonucu->GetParError(0);
cout << "1. uyum parametresi hatasi " <<  par1hata << endl;
```

Egrimizi uydurduk…



Kuyruklar iyi tanimlanmadi (neden?)

- Bir de bunu ROOT nesnesi olarak kaydedelim

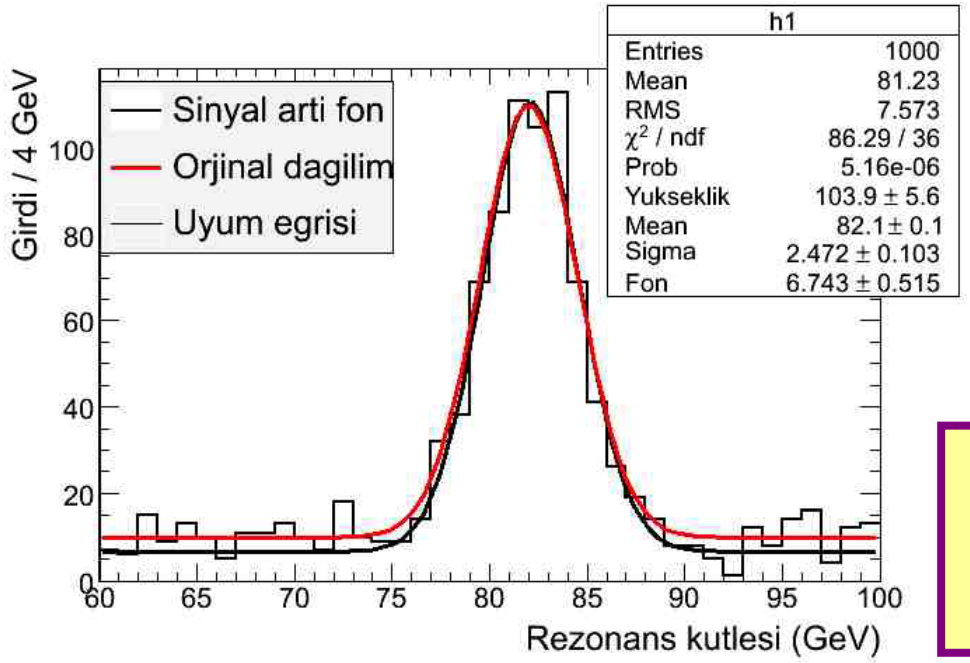
```
TFile dosya("gaus_fit_histo.root","RECREATE");
h1->Write();
dosya->Close();
```

Sonuç

- Eğrimizi uydurduk, ama hangi çizgi nedir?
  - TLegend kullanıp isimlendirelim:

```
TLegend *leg = new TLegend(0.1,0.62,0.40,0.88,NULL,"brNDC");
TLegendEntry *entry=leg->AddEntry("h1","Sinyal arti fon","lf");
entry=leg->AddEntry("rezonans","Orjinal dagilim","lf");
entry=leg->AddEntry("fToplam","Uyum egrisi","lf");
leg->Draw();
```

- Grafiğimiz bitti. Sonucumuz doğruysa, tezimize girebilir!



Asıl (uydurma) parametreler:

100

82

2.5

10

Size bu bir parçacığa uyuyor mu? Niye (değil)?

TTrees (Ağaçlar): Temel Kavramlar

- Ağaçlar benzer sınıf nesnelerini ekonomik bir şekilde (disk yeri ve erişim tasarrufu) bir arada tutmaya yarar
- TTree'yi kullanarak değişik veri tiplerinin derlemesini oluşturabilirsiniz:
  - "has a" ilişkisi ağac-dal-yaprak (TTree-TBranch-TLeaf) ile belirlenir
- Not: TNtuple TTree'den türemiş bir sınıftır.

class TNtuple: public TTree

```
TNtuple

A simple tree restricted to a list of float variables only.

Each variable goes to a separate branch.

A Ntuple is created via
TNtuple(name,title,varlist,bufsize)
It is filled via:
TNtuple::Fill(*x)   or
TNtuple::Fill(v1,v2,v3.....)
```

Kendi ağacımızı dikip, Meyvasını yemek için

- Bir TTree/TNtuple yazmak icin gerekli işlemleri gorelim.

Dosyayı aç

Ağacı yarat

Dalları yarat

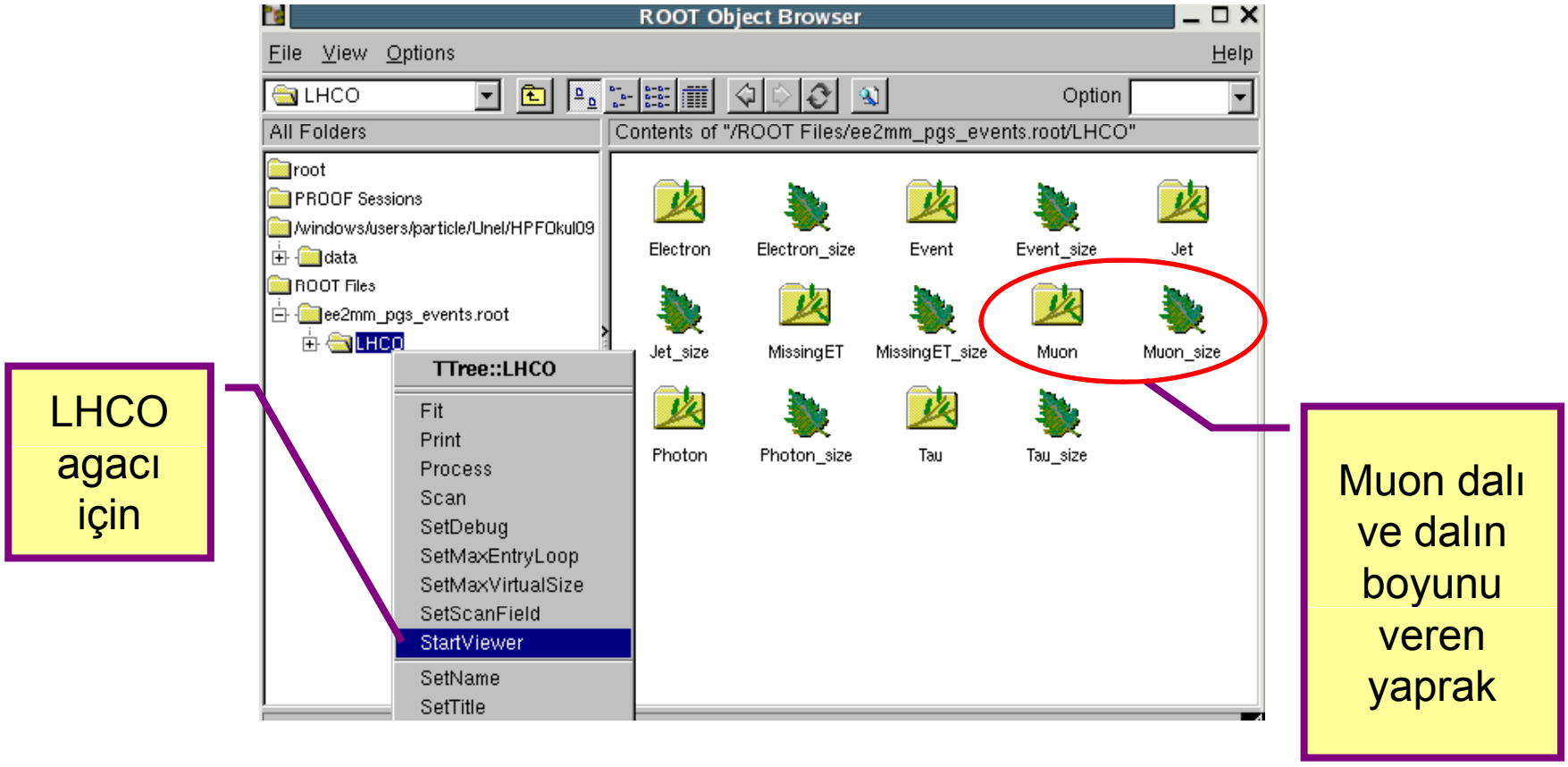
Ağacı doldur

Dosyayı kapat

```
{//agac_yarat.C (yarim bir koddur, TTree ornegi icindir)
//doyayi "RECREATE" secenegi ile yaratalim
TFile *OrnekDosya = new TFile("ornek.root", "RECREATE", "Ornektir");
//agacimizi yaratalim
TTree *OrnekAgac = new TTree("BenimAgacim", "Bir Odundur");
// (onceden kodumuzda tanimlanmis) degiskenlerimizi dal olarak ekliyelim
// hepsi float tipi olsun
OrnekAgac->Branch("x",&,"x/F");
OrnekAgac->Branch("y",&,"y/F");
// eventler uzerinden dongu atalim
for (Int_t olay=0; olay<10;olay++) {
      // gaussiana gore dagilimli iki rasgele rakam dolduralim
      gRandom->Rannor(x,y);
      OrnekAgac->Fill();
   }
OrnekAgac->Write();
OrnekDosya->Close();
delete OrnekDosya;
}
```

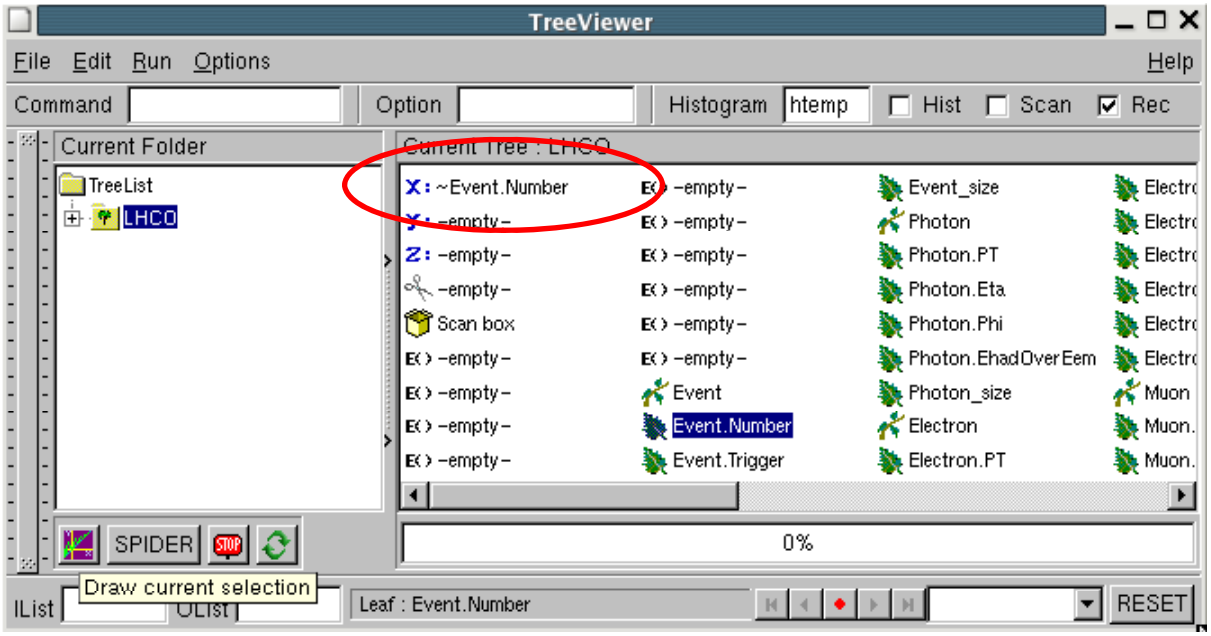
TreeViewer Tarayıcısı

- ROOT'un nesne taraycısından kolaylıkla yaratabileceğiniz bir tarayıcı penceresidir.
  - Komut satırından TTree::StartViewer() ile çağrılabilir .
- TreeViewer çabucak bir ROOT ağacının içine bakmak için iyidir.



TV ile bir Ağaca Erişmek

- Sürükle-bırak yöntemi ile 1-2-3 boyutlu histogramlar çizmek mümkündür.



- ROOT size LHCO ağacı için *tv tree işareticisini yarattı Tiyo: komut satırında yukarı ok yapın!

```
root [3] tv__tree = (TTree *) gROOT->FindObject("LHCO");
```

- Bir değişkeni komut satırından çizdirmek için:

```
root [3] tv__tree->Draw("Event.Number");
```

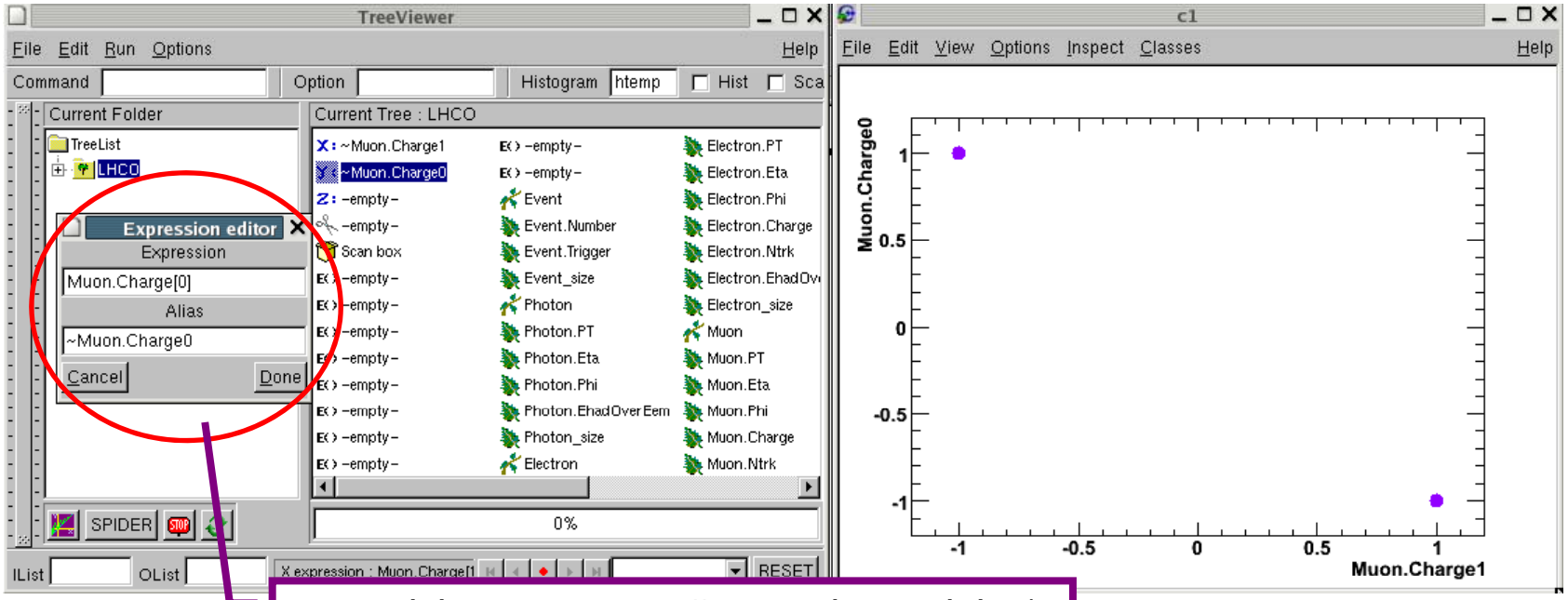
- Tiyo: Kendi kodunuzda TFile ve TTree isaretcinizi kendiniz yaratmalisiniz!

TV ile Analiz Yapmak

- TreeViewer ile verinizdeki dağılımların içeriğine bakmak çok kullanışlıdır.
- Ama verinin alt yapısını direk olarak bilmediğiniz zaman veya aynı anda bir sürü grafik çıkarmanız gerektiği zaman, analiz yapmak kolay değildir.

Kod yazmak şarttır!

- Aşağıda bütün olaylarda daima 2 muon olduğunu kabul edip(!), bu iki muonun yüklerini birbirine karşı çizdirdik. Bu muonlar yuksuz bir parçaçıktan geliyorlar ! ☺



x ve y için muon yaprağının yük üyesinin 1. ve 2. elemanlarını eksen başlıklarına koymak istediğimiz şekilde isimlendirdik

Analiz Kodu Yaratmak

- TTree::MakeClass() metodu:
  - Bir ROOT ntuple'iniz varsa, içine bakmak ve analiz kodunuzu yazmak için en çabuk yöntemdir.
  - Ntuple-bazlı formatlar için güzel çalışır, ama nesne-bazlı formatların metodlarını yaratmayı beceremez.
- TreeViewer'dan LHCO ağacına eriştiğimizi (*tv tree) düşünelim ve ee2mm_pgs adlı analiz kodumuzu yaratalım Tiyo: temelde ee2mm_pgs bir sınıf olduğu için c++ isimlendirme kurallarına uymalı!

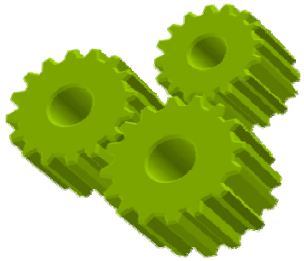
```
root [3] File name : ee2mm_pgs_events.root

root [3] tv__tree->MakeClass("ee2mm_pgs");
Info in <TTreePlayer::MakeClass>: Files: ee2mm_pgs.h and ee2mm_pgs.C generated
 from TTree: LHCO
root [4]
```

- ee2mm_pgs'i .L ile yükler ve metodlarına ulaşırsınız: *yeni artık ee2mm pgs'in bir nesne işaretçisi (instance) olmuştur.

```
root [0] .L ee2mm_pgs.C
root [1] ee2mm_pgs *yeni = new ee2mm_pgs();
```

```
root [2] yeni->Loop();
```

Analiz Koduna Bakış



```
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Tue Jan 20 19:02:25 2009 by ROOT version 5.18/00
// from TTree LHCO/Analysis tree
// found on file: ee2mm_pgs_events.root
//////////////////////////////////////////////////////////

#ifndef ee2mm_pgs_h
#define ee2mm_pgs_h

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
   const Int_t kMaxEvent = 1;
   const Int_t kMaxPhoton = 2;
   const Int_t kMaxElectron = 1;
   const Int_t kMaxMuon = 2;
   const Int_t kMaxTau = 1;
   const Int_t kMaxJet = 2;
   const Int_t kMaxMissingET = 1;

class ee2mm_pgs {
public :
   TTree          *fChain;   //!poin
   Int_t           fCurrent; //!curr
```

ee2mm_pgs.h

```
[karagozm@pplxgenng hpf_data]$ more ee2mm_pgs.C
#define ee2mm_pgs_cxx
#include "ee2mm_pgs.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void ee2mm_pgs::Loop()
{
//   In a ROOT session, you can do:
//      Root > .L ee2mm_pgs.C
//      Root > ee2mm_pgs t
//      Root > t.GetEntry(12); // Fill t data members with entry number 12
//      Root > t.Show();       // Show values of entry 12
//      Root > t.Show(16);     // Read and show values of entry 16
//      Root > t.Loop();       // Loop on all entries
```

- ee2mm_pgs::Loop(): olay başına döngü burada döner; analizinizi burada yaparsınız
- Buradan gerisini Cuma günü dinleyin!

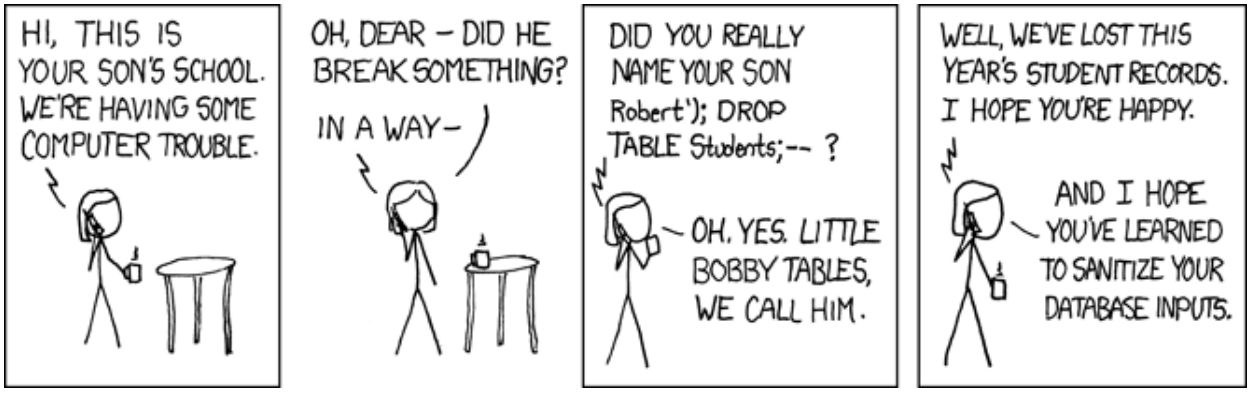
Tree Chains (Zincirleme Ağaçlar)

- Elinizde diyelim ki Grid'den yeni topladığınız 50 tane eşdeğer *ntuple_n.root* dosyası var. Bunları nasıl birbirine bağlayıp, aynı analizde üzerinden nasıl geçeceksiniz?
- TTree::TChain() buna izin verir.
  - Dosyaları ya analiz kodunda direk eklersiniz, ya da önceden bir büyük dosyaya yazarsınız.
  - Tiyo: İşletim sisteminize göre bir ROOT dosyasının boyu sınırlı olabilir (orn. 2 GB)

```
// eger analiz kodunuzda (sinfinizda) birden fazla TTree
//barindiran dosyaya bakmak istiyorsaniz
TChain * chain = new TChain("ch");

chain->Add("../rootfiles/ee2mm_pgs_1.root/LHCO");
chain->Add("../rootfiles/ee2mm_pgs_2.root/LHCO");
chain->Add("../rootfiles/ee2mm_pgs_3.root/LHCO");
// mesela zincirlenmis dosya listesini gormek icin...
chain->GetListOfFiles->Print();
// eger hepsini baska bir dosyaya atacaksaniz
chain->Merge("hepsiburda.root");
```

Bazı Tiyoların Yinelenmesi: analiz yaparken başınız ağrımasın!

- Kodunuzun geçerliliğini bir ROOT sürümünden ötekine geçerken mutlaka onaylayın!
- Histogramlarınızın eksen başlıklarını, birimleri yazmayı unutmayın!
- Kendi stilinizi kendi makronuzda yaratın; deneyinizin stil makrosu yoksa!
  - Öntanımlı gStyle kullanmayın (gROOT->SetStyle("Plain") tercih edin)
- Değişkenlerinize, histogramlarınıza anlaşılır ve düzgün isimler vermeyi adet edinin!
  - Aynı histogram adını birçok histogramda kullanmayın: memory leak ve yanlış hesaba yol açabilirsiniz
- Derlenmiş kodun daha güvenilir ve hızlı olduğunu unutmayın!
- ROOT'un c++ bazlı oduğunu, ve çalışmalarınızın çoğunda kod yazmak gerektiğini unutmayın: c++ temelinizi iyi oturtun!

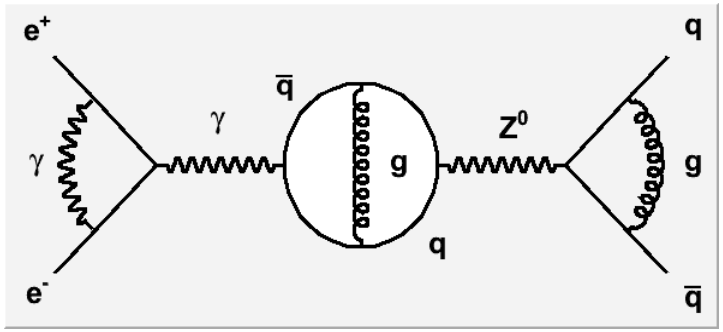


Unutmayın:
Çöp in – Çöp out

Üzerinden Geçmediklerimiz

- İleri seviye YEF kullanımlardan örnekler:
  - TLimit() sınıfı: 95% güvenlik seviyesi hesaplama (LHC'de çok yaygın)
  - RooFit: data modelleme kodu (http://roofit.sourceforge.net)
  - TSelector, MakeProject(): ileri seviye analiz için…
  - TSpectrum(): tepe bulucu

- Grafikler:
  - Profil histogramlar
  - Olay ekranı (event display) yaratmak
  - Feynman Diyagramları çizmek



- Unutmayın bunların çoğu ROOT'un eğitsel sayfasında örneklenmiştir!
- Deneylerin kendi eğitselleri de olduğunu biliyor muydunuz?
  ATLAS: https://twiki.cern.ch/twiki/bin/view/Atlas/RootBasicTutorial

Gelecek Dersler ve Ödevler

- ROOT'ta analiz veri sunumu yapmanın temellerini gördünüz.
- Bir sonraki adım cuma günü ROOT 4 ve Analiz Örneği derslerinde YEF uygulamalarına yönelik örnekler.
- ROOT 2 ve 3 ödevlerini ayrı bir dosya ile program sayfasına eklenmiş bulacaksınız: root-2-3_ödev.doc
- Ödevleri yaparken ROOT eğitsellerine bakabilirsiniz, ama önce kendiniz sonuç bulmaya çalışın.
- Mümkün olduğunca GUI üzerinde el alıştırması yapın. Bunu size bırakıyorum.

ROOT'tan nefret etmeyin, çünkü onunla çalışmak zorundasınız…
İyi Eğlenceler!