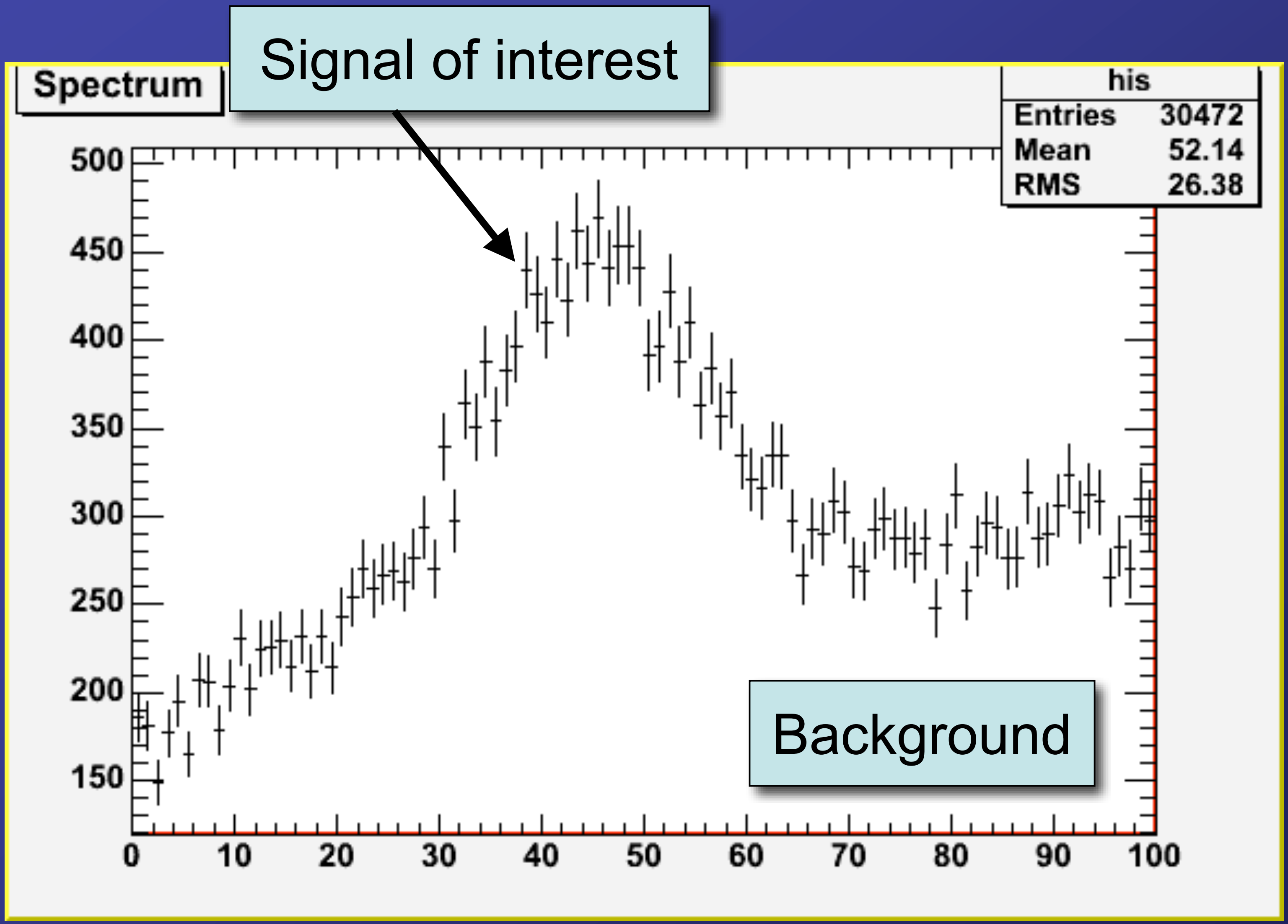
Exercises for Lecture 6

Exercise 1)

Download the root-file on the website:
http://kvir03.kvi.nl/rootcourse/.
Inside you'll find a 1-Dim histogram showing a Gaussian distributed signal on top of a - to-be determined - background signal. Write a macro that fits this spectrum using a user-defined function.

The histogram…

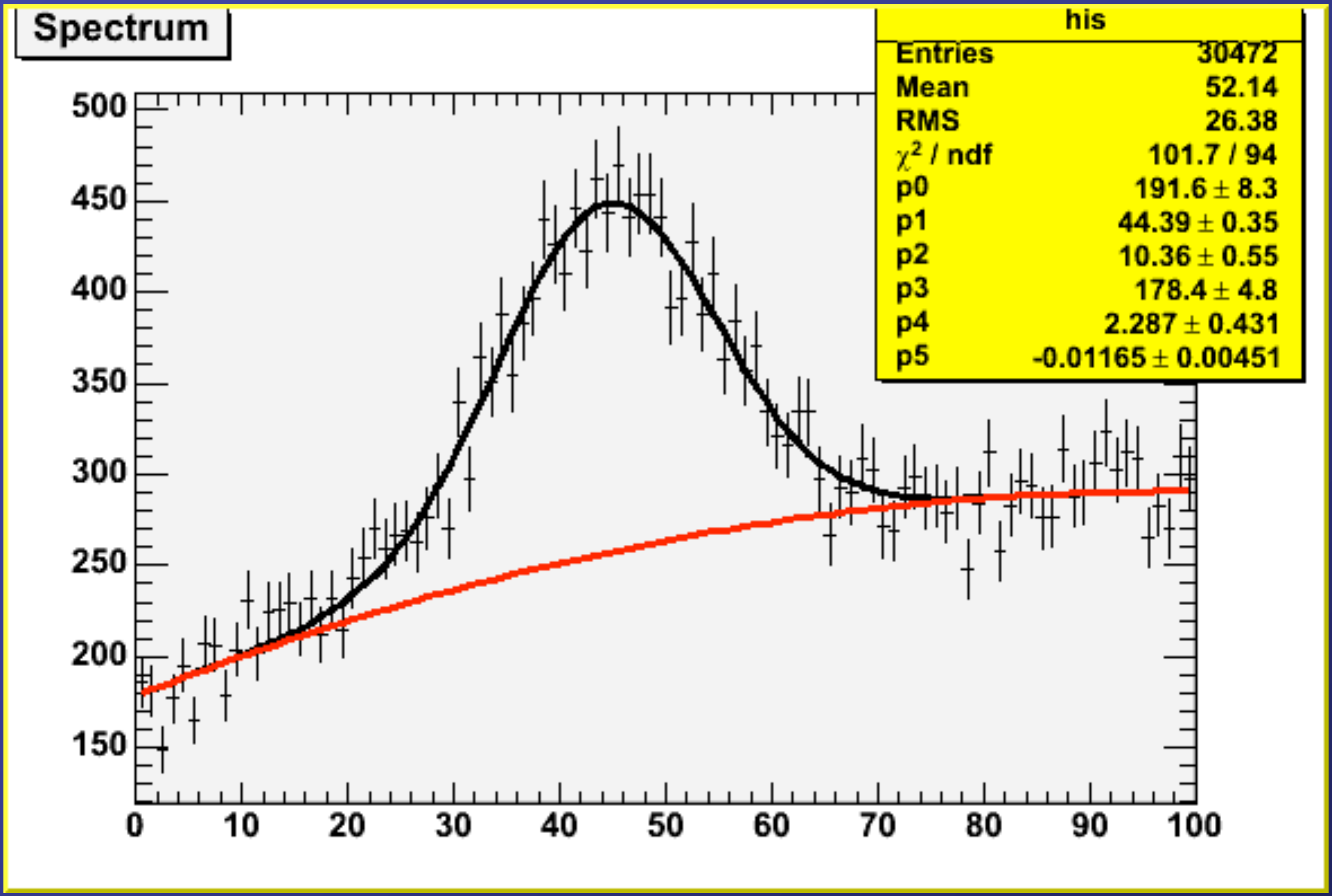


The macro…

- The fit function: Gaussian + 2cnd-order pol.
- The macro "fitExample"
- Obtain histogram
- Define functions
- Initialize fit parameters
- Fit histogram!
- Set background function
- Plot histograms and functions

```
Double_t myFitFunction(Double_t *x, Double_t *par)
{
  Double_t peak           = par[0]*TMath::Gaus(x[0],par[1],par[2]);
  Double_t background = par[3]+par[4]*x[0]+par[5]*x[0]*x[0];

  return (peak+background);
}


void fitExample()
{
TFile *f=new TFile("~/Documents/rootCourse/Lec6FitExample.root");
TH1D *his=f->Get("his");

TF1 *fSpectrum     = new TF1("fSpectrum",myFitFunction,0.,100.,6);
TF1 *fBackground = new TF1("fBackground","pol2",0,100);

fSpectrum->SetParameters(200,45,10,200,0,0);

his->Fit("fSpectrum");

Double_t fitParameters[6];

fSpectrum->GetParameters(fitParameters);
fBackground->SetParameters(&fitParameters[3]);

his->Draw("e");
fBackground->SetLineColor(kRed); fBackground->Draw("same");
}
```

The result…




```
root[0] .L fitExample.C
root[1] fitExample()
```

Exercises for Lecture 6

Exercise 2)

Figure out how to obtain the error matrix (i.e. the co-variance matrix) of the fit performed in Exercise 1. *Hint:* Explore the global **gMinuit** instance after fitting.

A couple of words about Error Matrix

In a nut-shell:

The error or co-variance matrix is a *n*x*n* matrix *M* - with *n* the number of fit parameters - which reflects the errors (=diagonal elements) and associated correlations (=off-diagonal elements) of/between the parameters.

Example function: $f(x, p_0, p_1)$

$$\sigma^2(f) = \left(\frac{\partial f}{\partial p_0}\right)^2 \sigma_{p_0}^2 + \left(\frac{\partial f}{\partial p_1}\right)^2 \sigma_{p_1}^2$$

A couple of words about Error Matrix

In a nut-shell:

The error or co-variance matrix is a *nxn* matrix *M* - with *n* the number of fit parameters - which reflects the errors (=diagonal elements) and associated correlations (=off-diagonal elements) of/between the parameters.

Example function: $f(x, p_0, p_1)$

$$\sigma^2(f) = \left(\frac{\partial f}{\partial p_0}\right)^2 M_{0,0} + \left(\frac{\partial f}{\partial p_1}\right)^2 M_{1,1}$$

A couple of words about Error Matrix

In a nut-shell:

The error or co-variance matrix is a *nxn* matrix *M* - with *n* the number of fit parameters - which reflects the errors (=diagonal elements) and associated correlations (=off-diagonal elements) of/between the parameters.

Example function: $f(x, p_0, p_1)$

$$\begin{aligned}\sigma^2(f) &= \left(\frac{\partial f}{\partial p_0}\right)^2 M_{0,0} + \left(\frac{\partial f}{\partial p_1}\right)^2 M_{1,1} \\ &+ \frac{\partial f}{\partial p_0} M_{0,1} \frac{\partial f}{\partial p_1} + \frac{\partial f}{\partial p_1} M_{1,0} \frac{\partial f}{\partial p_0}\end{aligned}$$

How to obtain the error matrix?

From the root webpage, reference documentation TH1::Fit(…) :

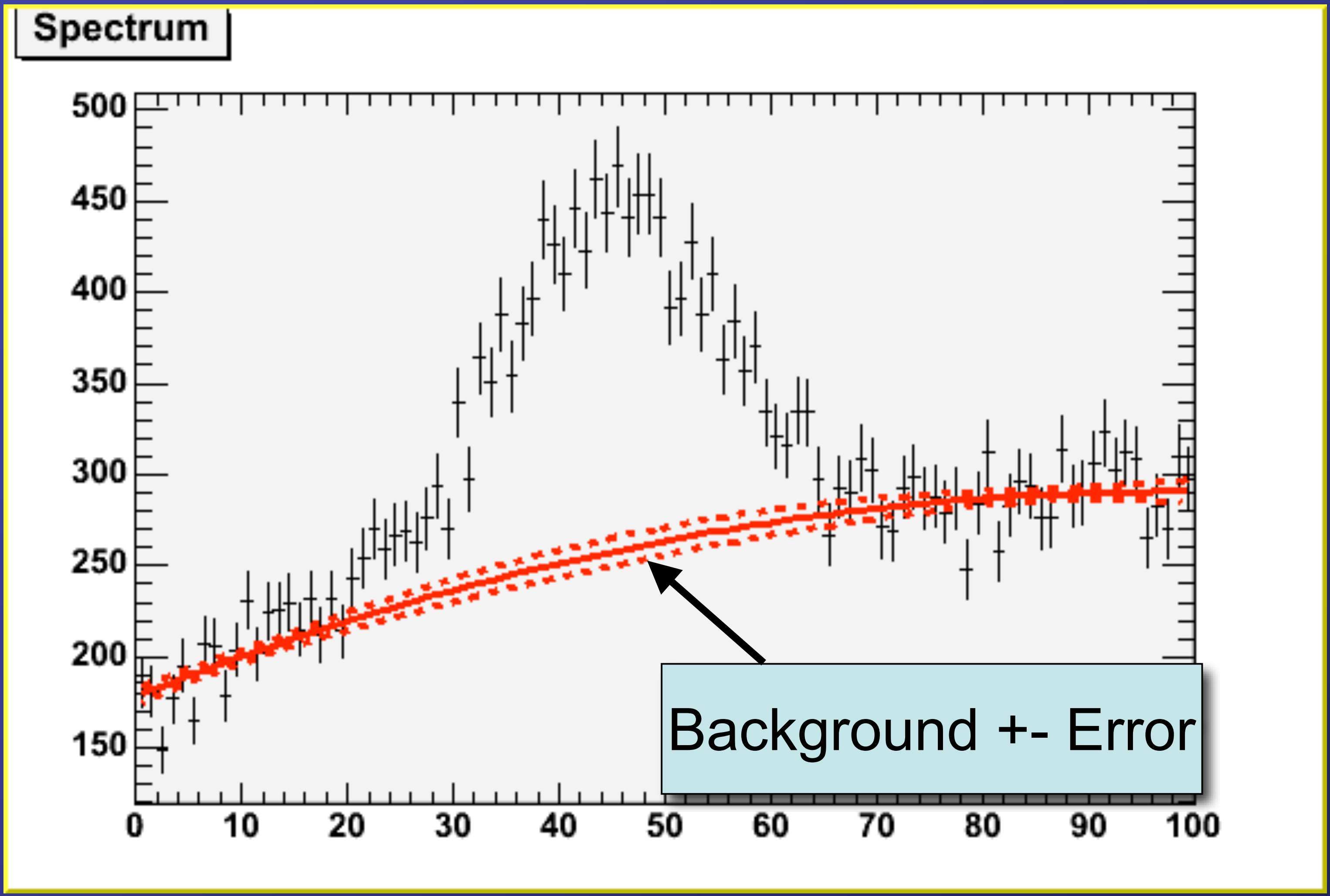
```
Access to the fit covariance matrix
    ====================================
    Example1:
      TH1F h("h","test",100,-2,2);
      h.FillRandom("gaus",1000);
      h.Fit("gaus");
      Double_t matrix[3][3];
      gMinuit->mnemat(&matrix[0][0],3);
    Example2:
      TH1F h("h","test",100,-2,2);
      h.FillRandom("gaus",1000);
      h.Fit("gaus");
      TVirtualFitter *fitter = TVirtualFitter::GetFitter();
      TMatrixD matrix(npar,npar,fitter->GetCovarianceMatrix());
      Double_t errorFirstPar = fitter->GetCovarianceMatrixElement(0,0);
```

Our macro…

```
void fitExample()
{
…
TF1 *fSpectrum     = new TF1("fSpectrum",myFitFunction,0.,100.,6);
his->Fit("fSpectrum");

Double_t errorMatrix[6][6];
gMinuit->mnemat(&errorMatrix[0][0],6);

for (Int_t i=0; i<6; i++) {
  for (Int_t j=0; j<6; j++) {
    cout <<errorMatrix[I][j] << “  “;
   }
  cout << endl;
 }
}
```

68.8701	0.191852	1.36336	13.3192	-2.43024	0.0257033
0.191852	0.123819	0.00464778	0.406327	-0.0181254	0.000128331
1.36336	0.00464778	0.300779	0.546213	-0.163807	0.00175386
13.3192	0.406327	0.546213	22.5861	-1.42522	0.0133135
-2.43024	-0.0181254	-0.163807	-1.42522	0.185814	-0.00192754
0.025703	0.0001283	0.00175386	0.0133135	-0.00192754	2.03559e-05

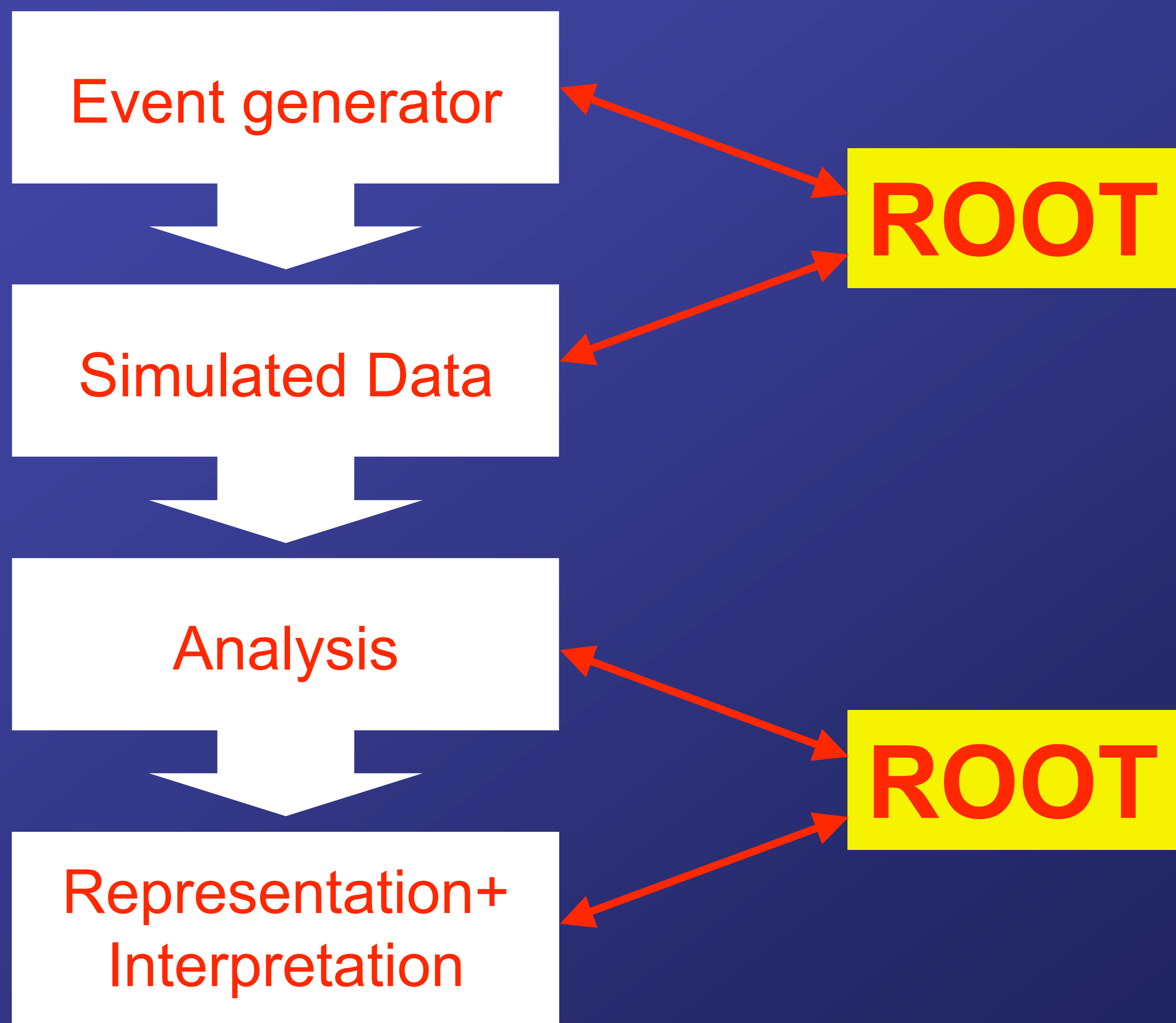
Using the error matrix…



ROOT Lecture 7

Simulations and Event Generators

ROOT as Simulation Package!



Random number generators

A random number generator (RNG) is a computer algorithm that can be used to generate a sequence of numbers which appear to be distributed randomly.

Criteria for "quality" of RNG:

1) "**randomness**": how random is it, does it pass certain statistical tests?
2) "**periodicity**": after how many events does the sequence start over again?
3) "**speed**". How fast can a random number be generated?

Random number generators in ROOT

Class TRandom : public TNamed;

TRandom(***UInt_t*** seed=65539)

- The default RNG in ROOT
- The fastest RNG in ROOT
- Periodicity = 10^8 events

Random number generators in ROOT

Class TRandom2 : public TRandom;

TRandom2(***UInt_t*** seed=65539)

- Slower than TRandom.
- Periodicity > 10^{14} events

Random number generators in ROOT

Class TRandom3 : public TRandom;

TRandom3(***UInt_t*** seed=65539)

- Mersenne Twistor: Primitive Twisted Generalized Feedback Shift Register Sequence (ouch!)
- Nearly as fast as TRandom, faster than TRandom2
- Periodicity = 2^{19937}-1 events

How to use TRandom2/3?

What?	How?
Generate uniformly distributed random number between 0..1	Double_t **Rndm**()
Generate uniformly distributed random number between x1..x2	Double_t **Uniform**(x1, x2)
Generate random numbers according to exponential distribution with slope t	Double_t **Exp**(t)
Generate random numbers according to a Gaussian distribution	Double_t **Gaus**(mean,sigma)
Generate random numbers according to a Poisson distribution	Double_t **Poisson**(mean)

……… (many more)

How to use TRandom2/3? A benchmark…

Create 3 RNGs

Create 3 Histograms

Setup stopwatch

Fill histogram 1 with numbers from RNG 1

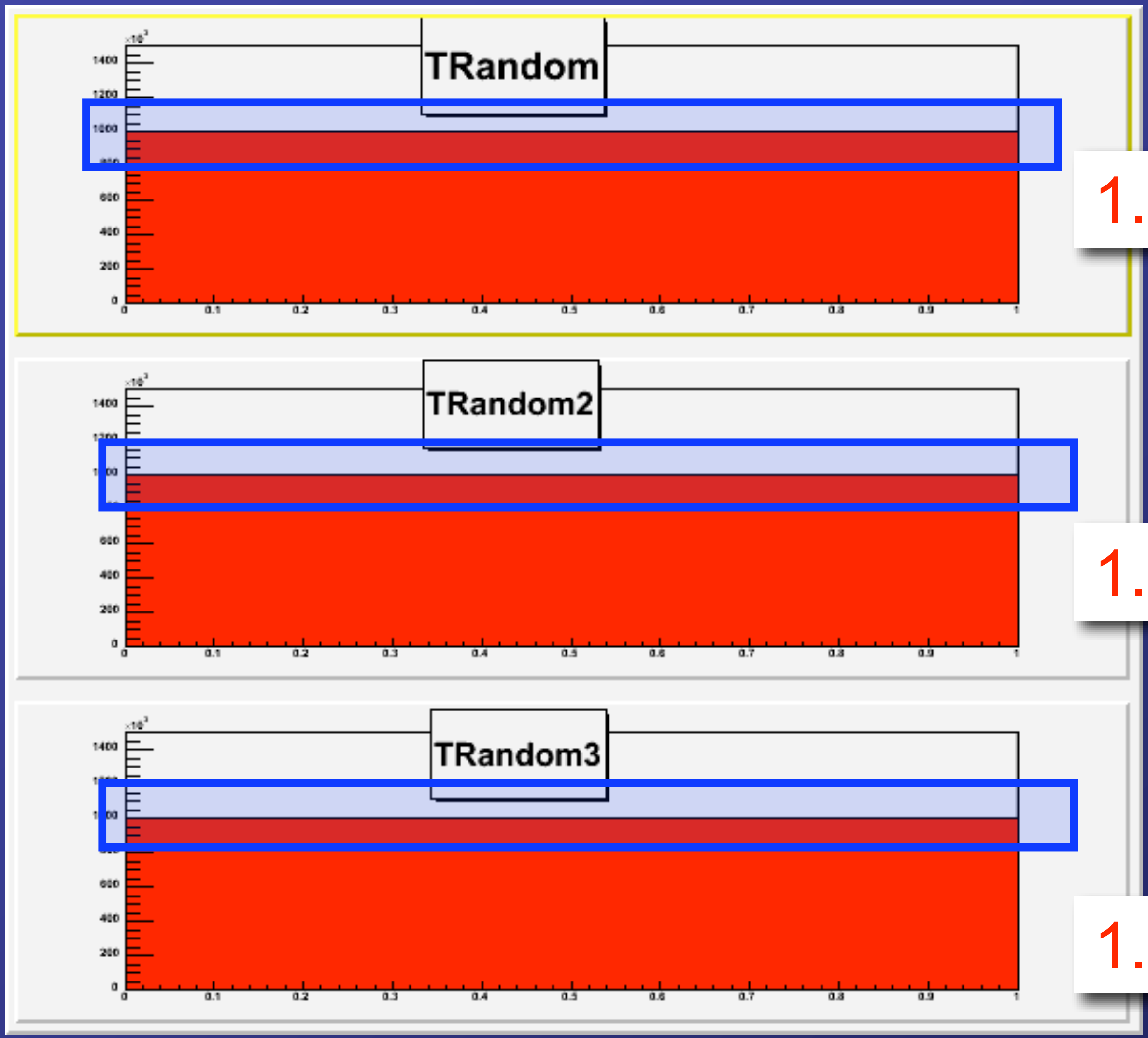
Fill histogram 2 with numbers from RNG 2

Fill histogram 3 with numbers from RNG 3

```
void testRandom(Int_t nrEvents=500000000)
{
 TRandom *r1=new TRandom();
 TRandom2 *r2=new TRandom2();
 TRandom3 *r3=new TRandom3();

 TH1D *h1=new TH1D("h1","TRandom",500,0,1);
 TH1D *h2=new TH1D("h2","TRandom2",500,0,1);
 TH1D *h3=new TH1D("h3","TRandom3",500,0,1);

 TStopwatch *st=new TStopwatch();

 st->Start();
 for (Int_t i=0; i<nrEvents; i++) { h1->Fill(r1->Uniform(0,1)); }
 st->Stop(); cout << "Random:  " << st->CpuTime() << endl;

 st->Start();
 for (Int_t i=0; i<nrEvents; i++) { h2->Fill(r2->Uniform(0,1)); }
 st->Stop(); cout << "Random2: " << st->CpuTime() << endl;

 st->Start();
 for (Int_t i=0; i<nrEvents; i++) { h3->Fill(r3->Uniform(0,1)); }
 st->Stop(); cout << “Random3: “ << st->CpuTime() << endl;
}
```

TRandom::Uniform()

(500 mln events)



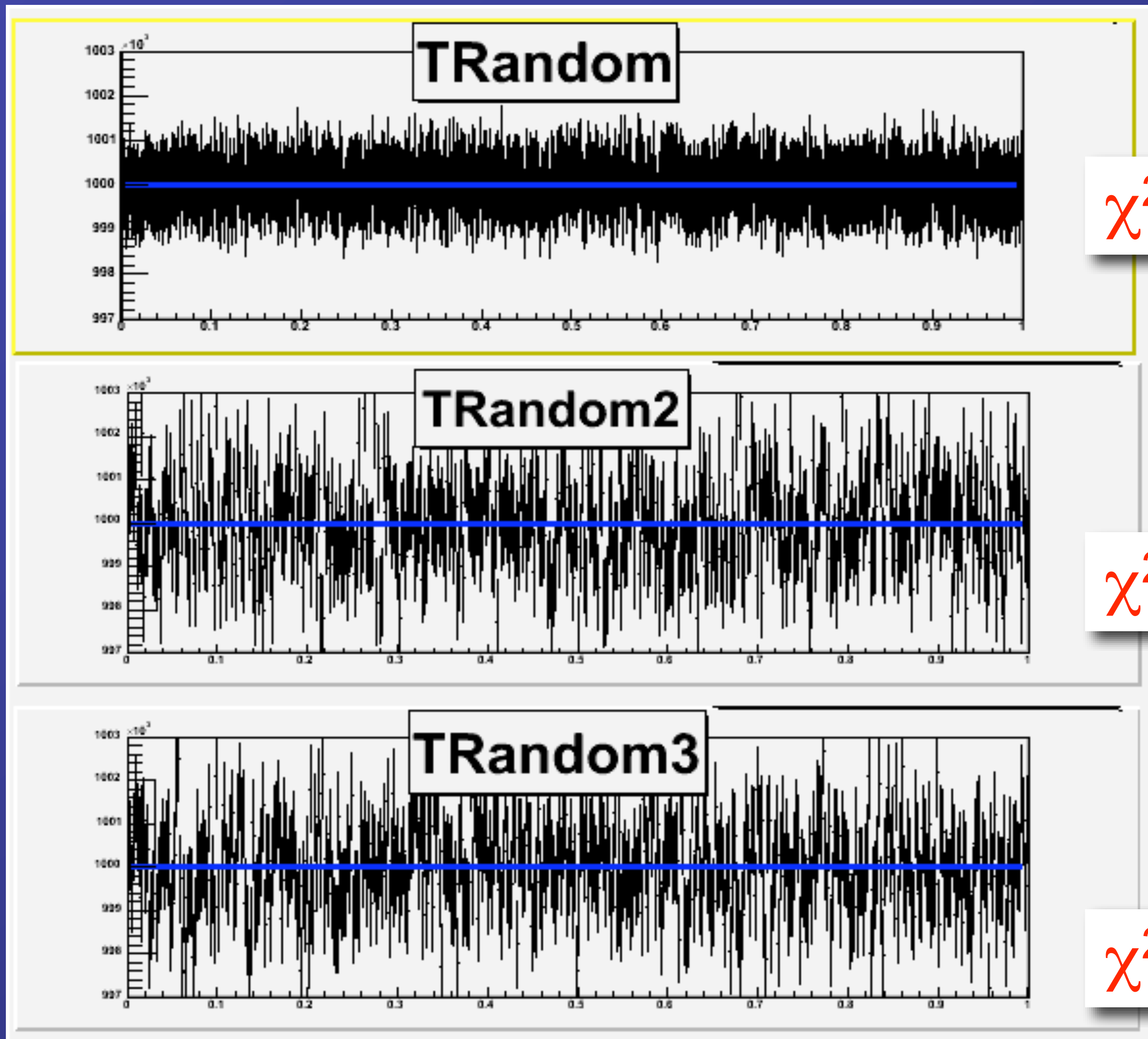
1.315 [ms/event]

1.473 [ms/event]

1.298 [ms/event]

TRandom::Uniform()

(500 mln events)



$\chi^2/ndf = 0.068$

$\chi^2/ndf = 1.023$

$\chi^2/ndf = 0.958$

Reminder: TRandom has periodicity of 100 mln!!!

Tips and Tricks

1. Changing the default random number generator **gRandom** (by default of type TRandom) :

```
root[0] TRandom3 *myRNG=new TRandom3();
root[1] gRandom = myRNG;
root[2] …
```

2. Save the information (i.e seeds) of RNG into a file (i.e. avoid to start with the same seed):

```
root[0]  gRandom->ReadRandom(“random.dat”);
root[1]  …. (your analysis using gRandom)…
root[45] gRandom->WriteRandom(“random.dat”);
```

Fun with RNGs of ROOT: GetRandom() and FillRandom()

1. Throwing random numbers according to your own histogram distribution :

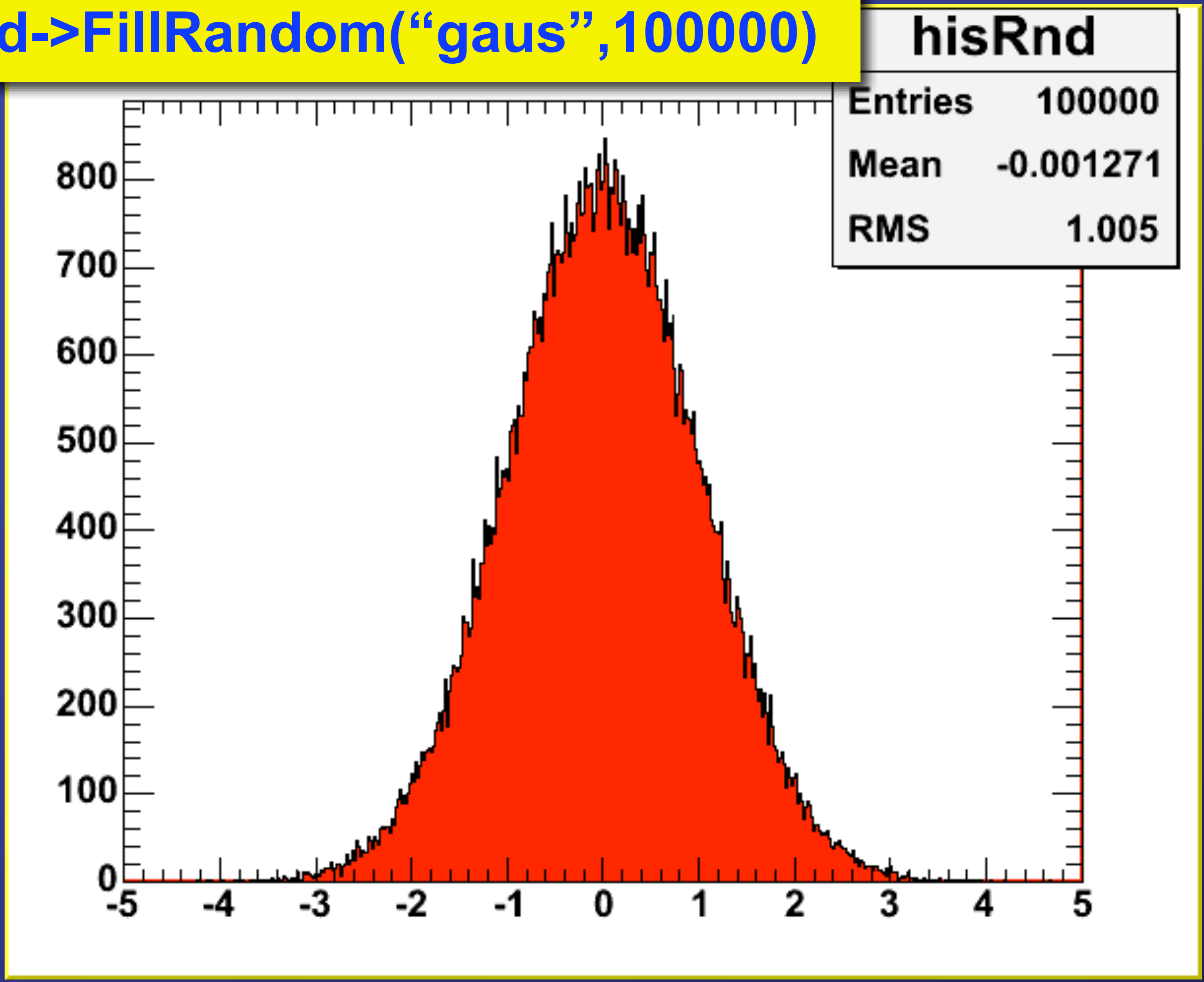
```
root[0] TH1D *his = f->Get("myHis")
root[1] Double_t aRandomNumber = his->GetRandom()
root[2] TH1D *hisRnd = new TH1D(…)
root[3] hisRnd->FillRandom(his,1000)
```

2. Throwing random numbers according to your own function :

```
root[0] TF1 *func = new TF1("myFunc","TMath::Gaus(x,0,1)",-5,5)
root[1] Double_t aRandomNumber = func->GetRandom()
root[2] hisRnd->FillRandom("myFunc",1000)
```

Fun with RNGs of ROOT

hisRnd->FillRandom(“gaus”,100000)

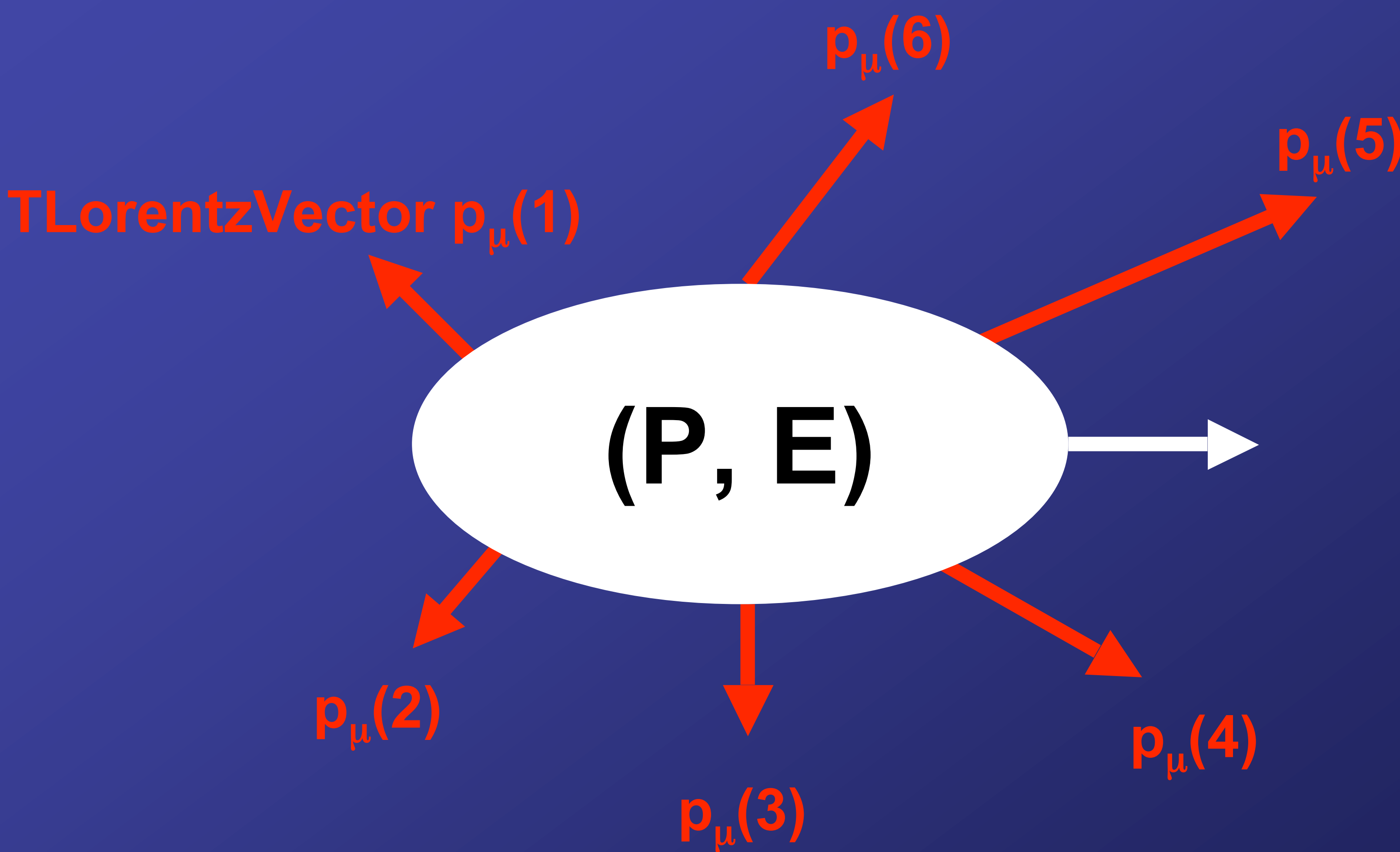


Kinematics and RNGs

class TGenPhaseSpace : public TObject ;

- N-body relativistic phase space event generator
- Produces kinematically allowed events according to phase space distribution
- Original code : GENBOD (F. James, CERNLIB)
- Fastest multi-purpose phase space generator available!!

TGenPhaseSpace



Produces randomly distributed particles with 4-momentum p_μ

TLorentzVector

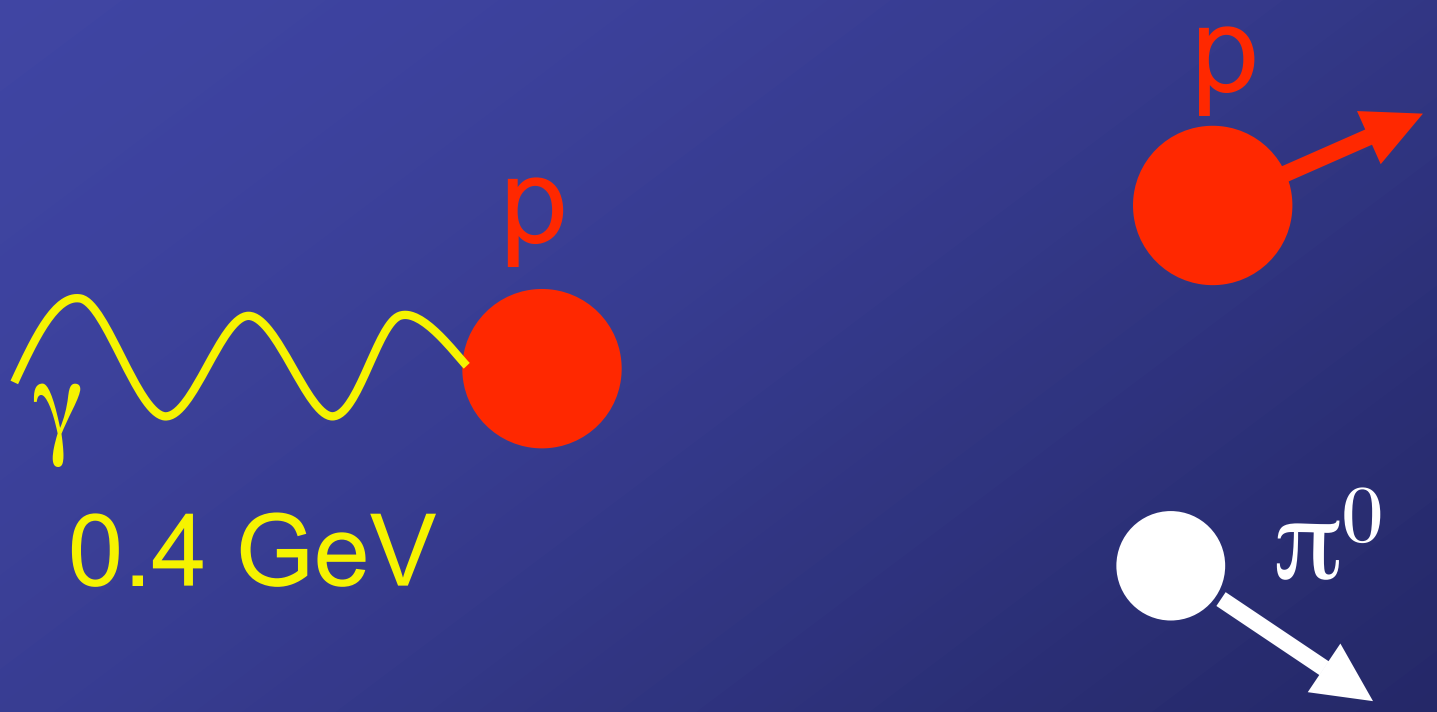
```
class TLorentzVector : public TObject ;
```

TLorentzVector fourMom(p_x, p_y, p_z, E)
TLorentzVector fourVec(x, y, z, t)

fourMom.M(); /* Returns the mass */
fourMom.Beta(); /* Returns β */
fourMom.Gamma(); /* Returns γ */
fourMom.Theta(); /* Return polar angle θ */
fourMom.Phi(); /* Returns azimuthal angle ϕ */
fourMom.Boost(P_x,P_y,P_z); /* Boost vector */

TLorentzVector totalMom = fourMom1+fourMom2;
/* Add LorentzVectors! */

TGenPhaseSpace
An application



How to simulate this reaction using TGenPhaseSpace? What is for instance the angular distribution of pion?

TGenPhaseSpace

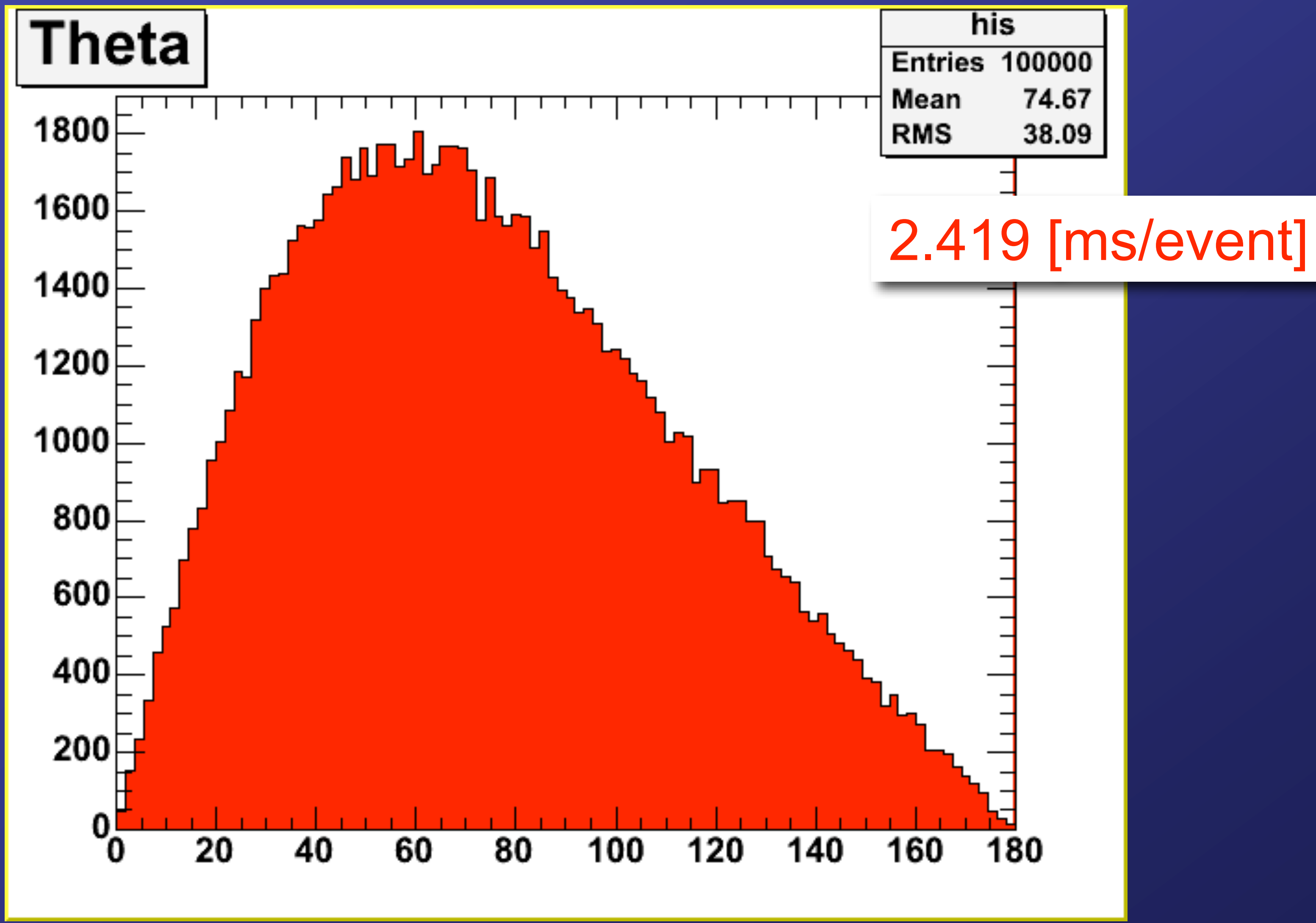
An application

- **Load the physics library to use TGenPhaseSpace/TLorentzVector**
- **Define initial 4-momenta**
- **Masses of final-state particles**
- **Define Decay for TGenPhaseSpace**
- **Book a histogram for results**
- **Generate events!!**
- **Extract 4-momenta of final-state particles**
- **Update histogram**
- **Draw histogram**

```
{
  gSystem.Load("libPhysics");

  TLorentzVector target(0.0, 0.0, 0.0, 0.938);
  TLorentzVector beam(0.0, 0.0, .4, .4);
  TLorentzVector W = beam + target;

  Double_t masses[2] = { 0.938, 0.135} ;

  TGenPhaseSpace event;
  event.SetDecay(W, 2, masses);

  TH1D *h = new TH1D(“his”, “Theta”, 100, 0, 180);

  for (Int_t n=0;n<100000;n++) {
    event.Generate();

    TLorentzVector *pProton  = event.GetDecay(0);
    TLorentzVector *pPi0       = event.GetDecay(1);

    h->Fill(pPi0->Theta()*57.3);
  }
  h->Draw();
}
```

TGenPhaseSpace

An application



More advanced Generators and Simulation Packages

PLUTO++:

Fast Simulation Package for Hadronic Interactions, ROOT-based, no development

www-hades.gsi.de/computing/pluto/html/PlutoIndex.html

Geant3:

Detector Simulation Package Fortran-based, CERN, no development

Geant4:

Detector Simulation Package C++-based, in development

geant4.web.cern.ch/geant4/

Exercises for Lecture 7

Exercise 1)

Write a macro which performs a benchmark comparison between TRandom, TRandom2, TRandom3. Compare the performance of the "Gaus" method of these classes. Also judge the "randomness" for 200 mln events by making a fit through the simulated data.

Exercise 2)

Write a macro which generates kinematically allowed events for the reaction p+d->p+p+n with an incident proton energy of 200 MeV and a deuteron at rest. Make a histogram of the scattering angle of the neutron in the lab. frame and in the center-of-mass frame.

Send your results to messchendorp@kvi.nl