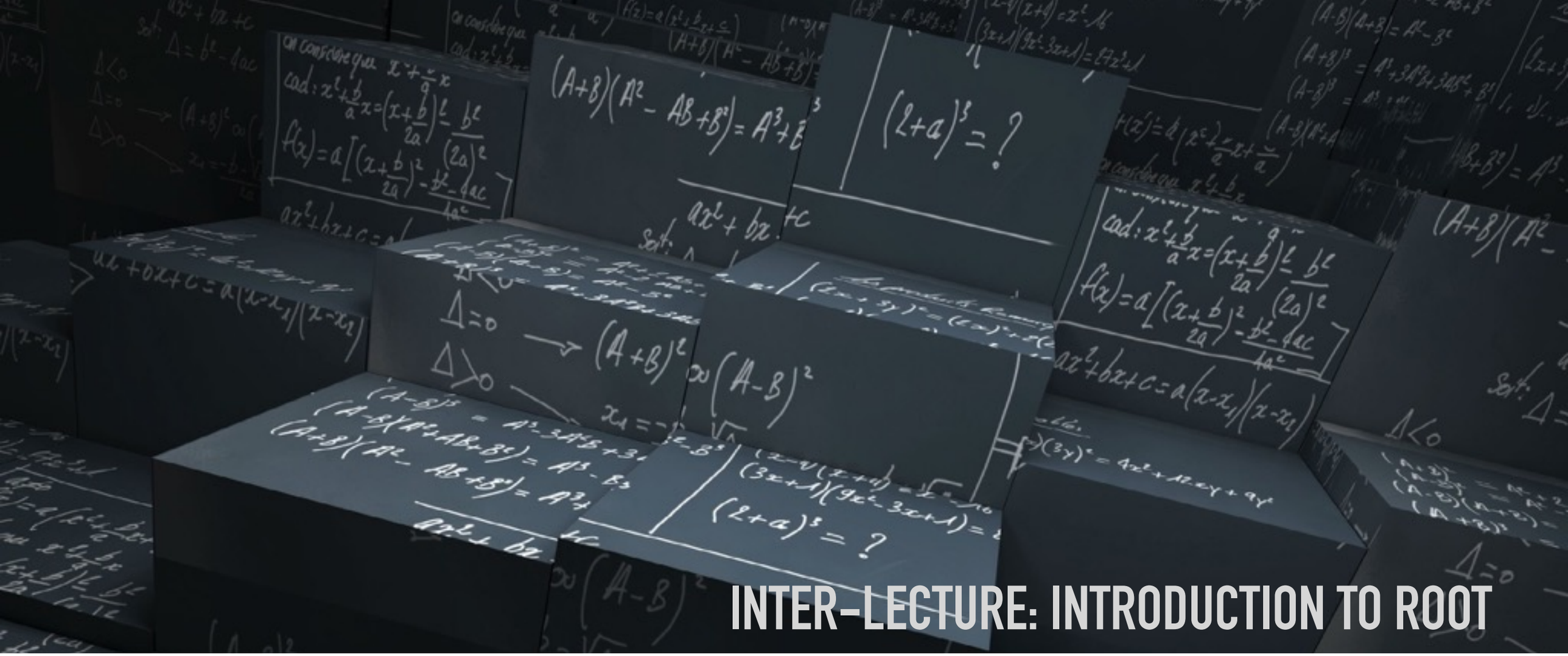


INTER-LECTURE: INTRODUCTION TO ROOT

STATISTICAL ANALYSIS IN EXPERIMENTAL PARTICLE PHYSICS

Kai-Feng Chen
National Taiwan University

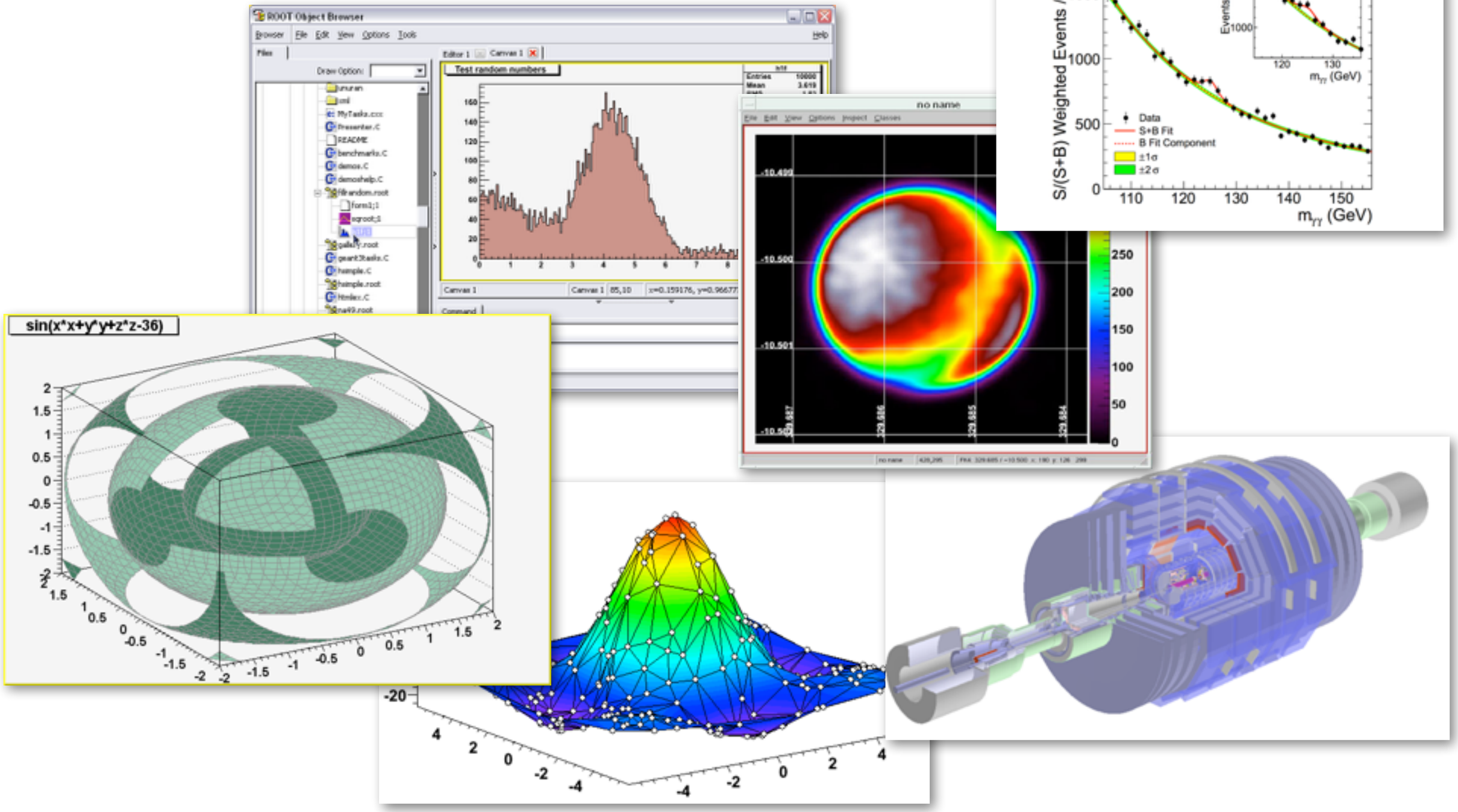
ROOT: AN INTRODUCTION

- An analysis framework for large scale data handling, mostly used by high-energy physics, astrophysics, nuclear physics, including both experiments & theory.
- It's also an open source project.
- Started in 1995, with 8 full time developers at CERN, plus Fermilab, Agilent Tech, Japan, MIT (one each).
- Large number of part-time developers: Users participate its development.
- Available (incl. source) under GNU/LGPL.

ROOT: AN INTRODUCTION (CONT.)

- **The ROOT framework can deal with:**
  - An efficient(?) data storage, access and query system up to PetaBytes level.
  - Advanced statistical analysis tools and algorithms (*multi dimensional histogramming, fitting, minimization and cluster finding, etc.*)
  - Scientific visualization: 2D and 3D graphics, convert the output to Images (jpeg/png/gif), Postscript, PDF, LateX
  - Geometrical modeller (*to show some 3D detector view*)
  - PROOF parallel query engine (MPI...)

SOME FANCY(?) FIGURES



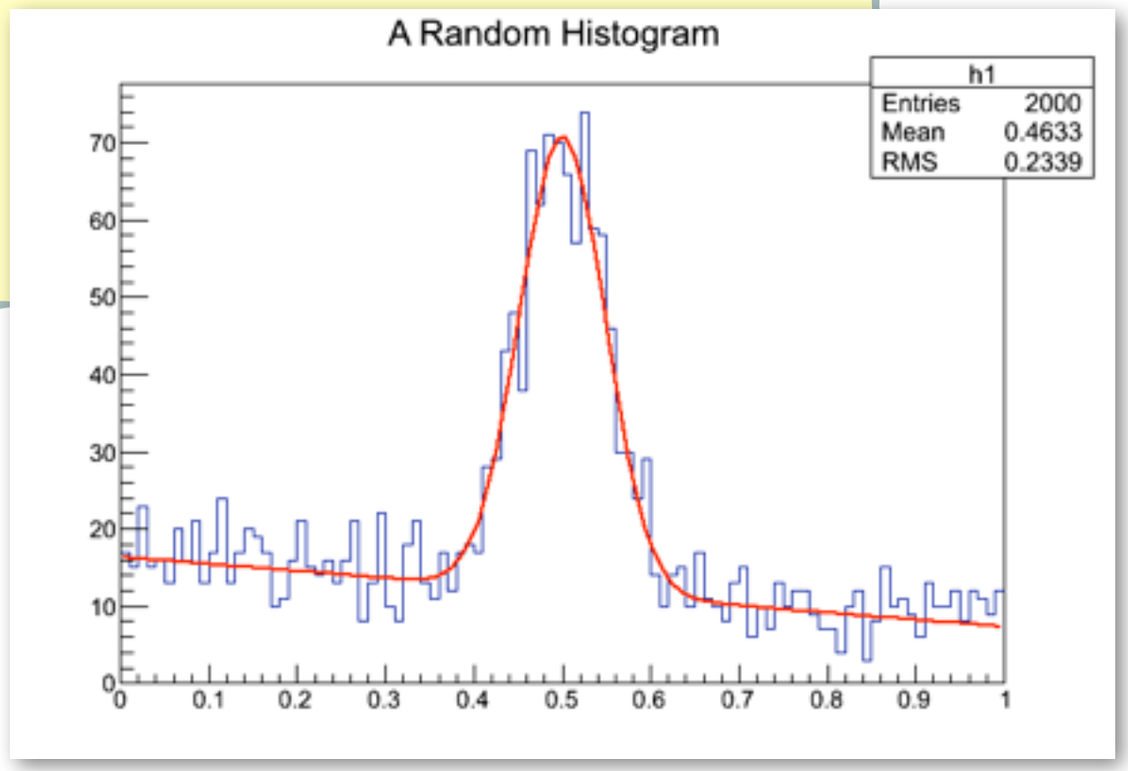
A LITTLE HANDS-ON SHOW

- I'm going to show you a very easy/little thing that can do with the ROOT framework:

example_01.cc

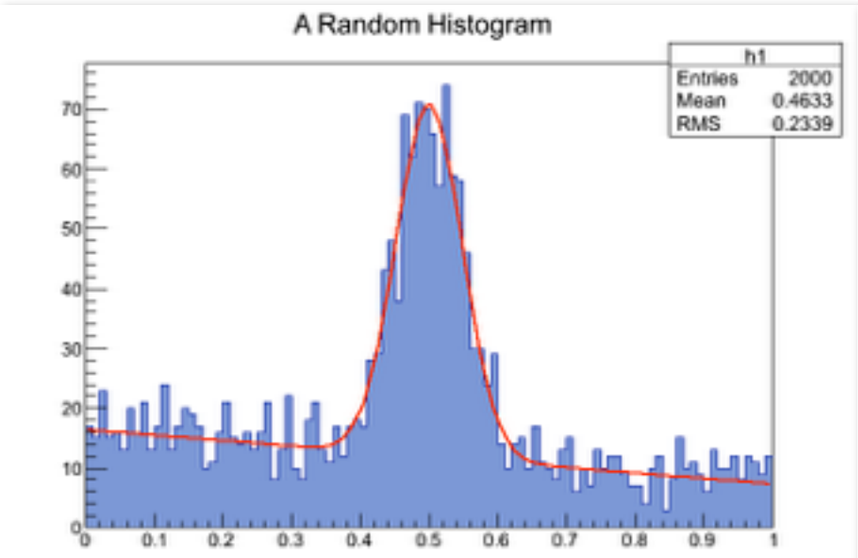
```
{
    TH1F *h1 = new TH1F("h1","A Random Histogram",100,0.,1.);

    TF1 *f1 = new TF1("f1","[0]+[1]*x+[2]*gaus(2)");
    f1->SetParameters(2.,-1.0,2.5,0.5,0.05);

    h1->FillRandom("f1",2000);

    h1->Fit("f1");
}
```

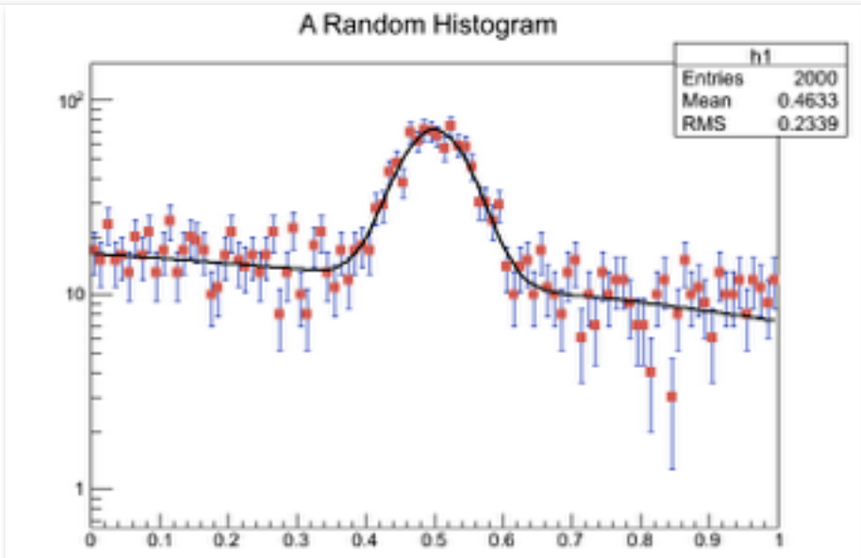
A direct run of this piece of code will generate this plot (with a fit) already:

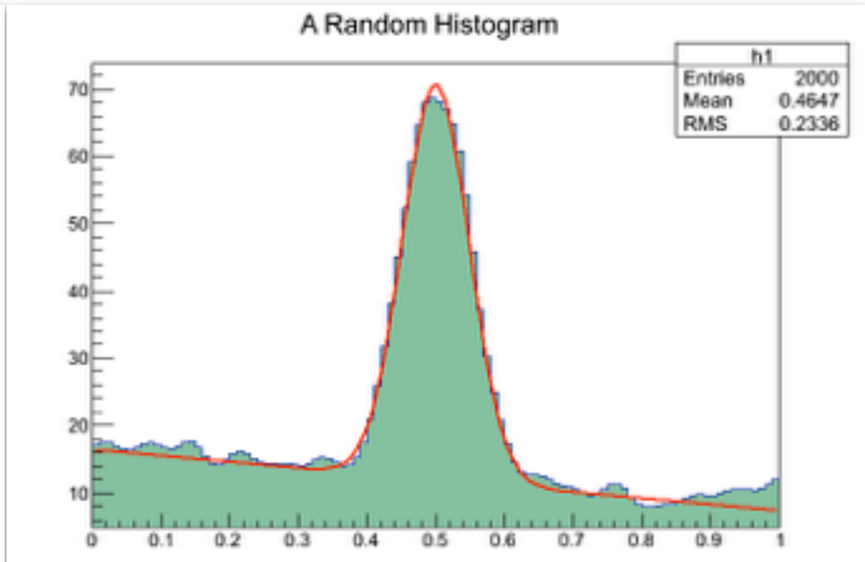


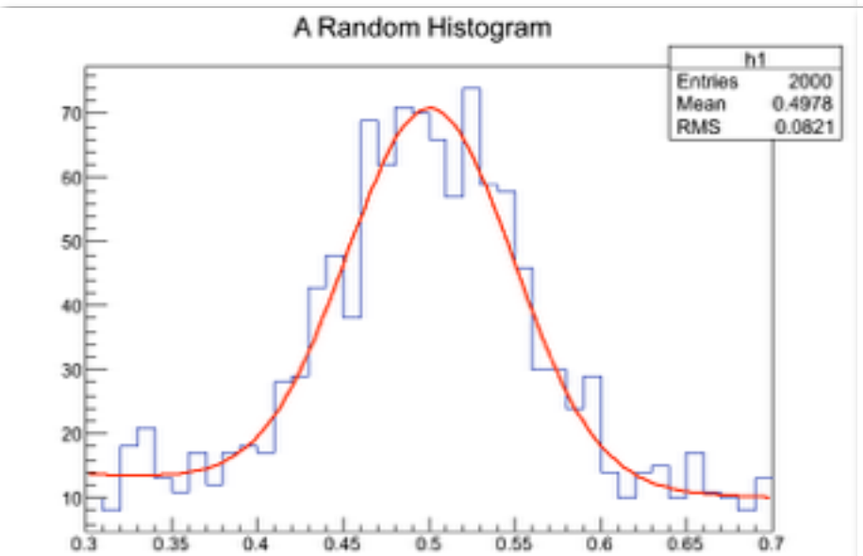
A LITTLE HANDS-ON SHOW (CONT.)

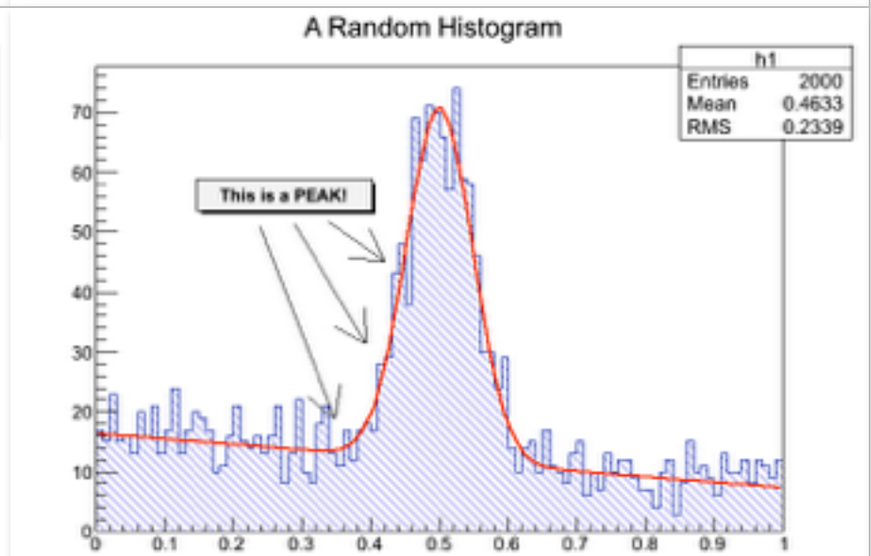
- We can already play around with this figure easily:

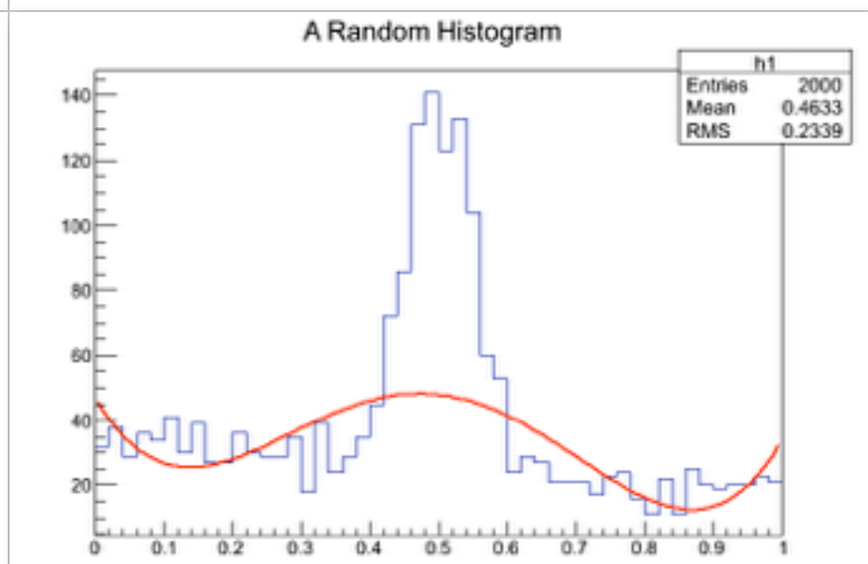








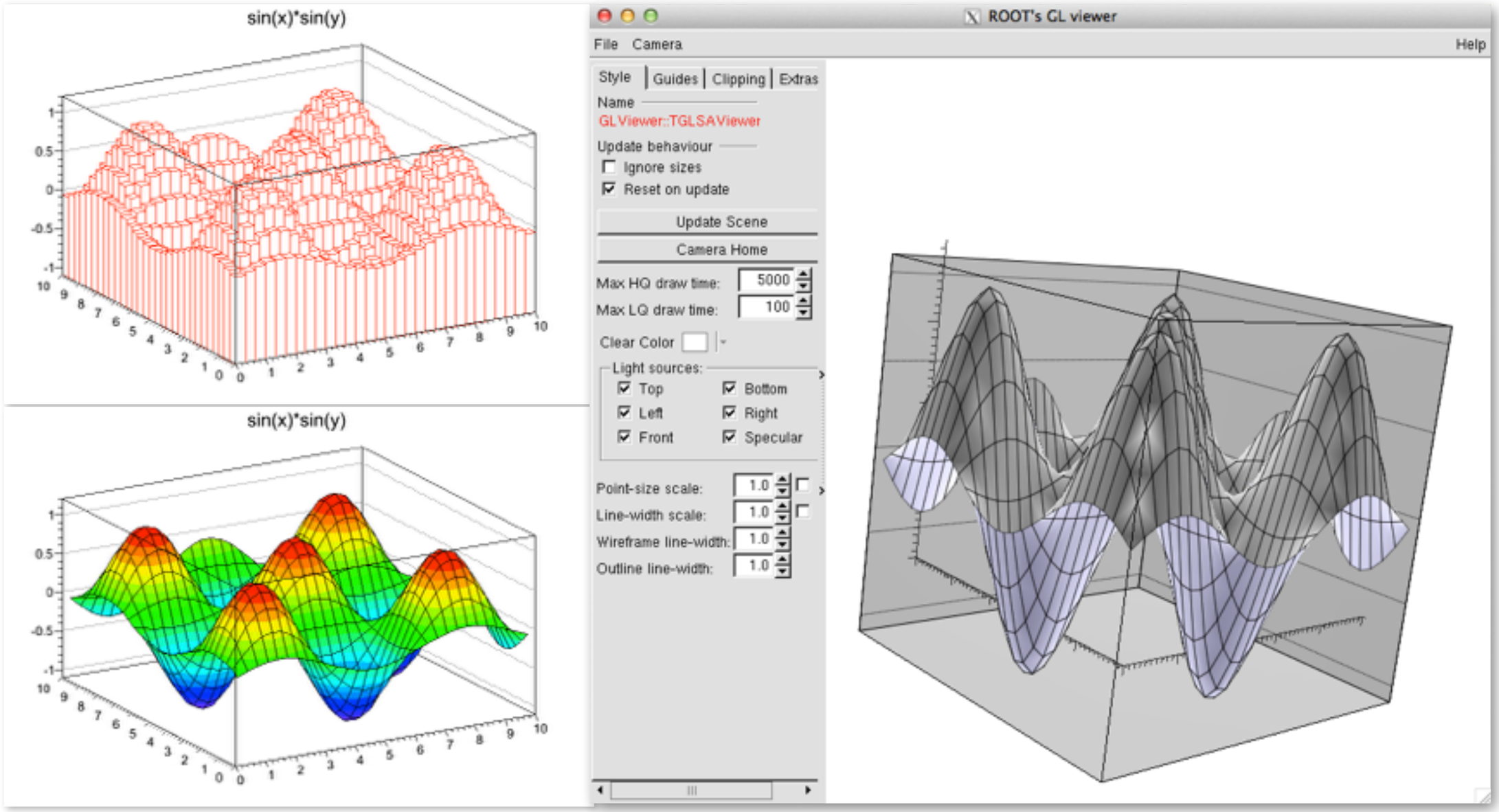




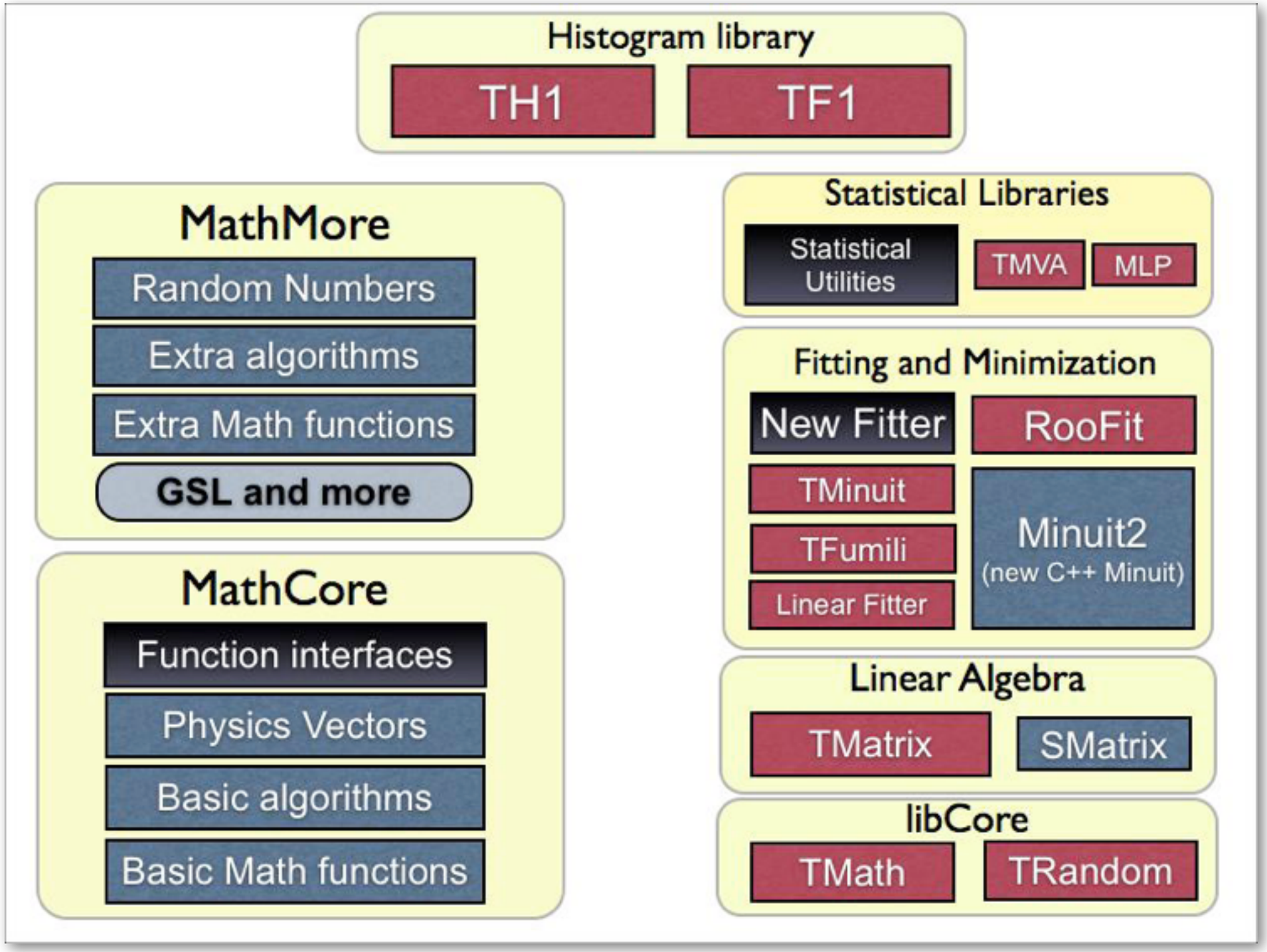
SOME FANCIER(?) STUFF

- 3D views:

```
TF2 *f2 = new TF2("f2","sin(x)*sin(y)",0,10,0,10);
f2->Draw();
```



MATHEMATICS UTILITIES



MATHEMATICS UTILITIES

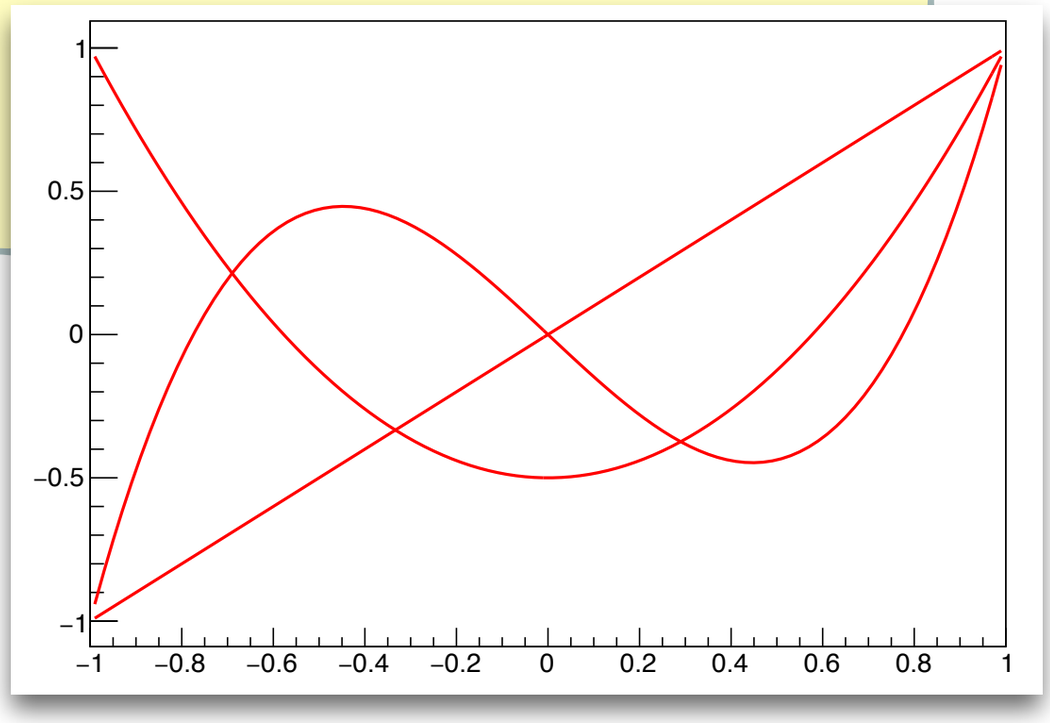
- Most of the computing tasks we have covered in this lecture can be actually made with few root commands.
- e.g. plotting the Legendre polynomials:

example_02.cc

```
{
    gSystem->Load("libMathMore");

    TF1* l1 = new TF1("l1", "ROOT::Math::legendre(1,x)", -1., 1.);
    TF1* l2 = new TF1("l2", "ROOT::Math::legendre(2,x)", -1., 1.);
    TF1* l3 = new TF1("l3", "ROOT::Math::legendre(3,x)", -1., 1.);

    l1->Draw();
    l2->Draw("same");
    l3->Draw("same");
}
```

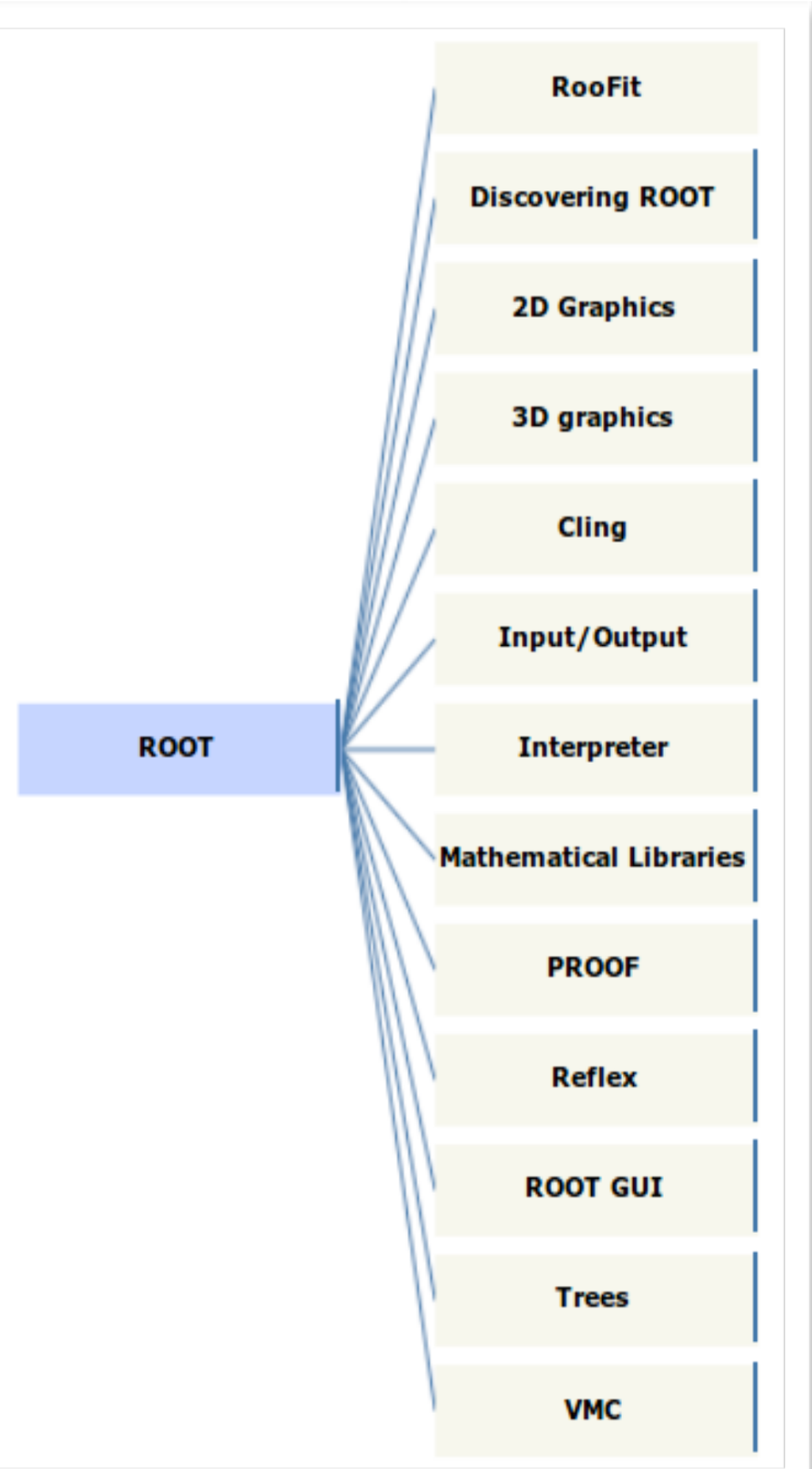
$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} [(x^2 - 1)^n]$$

no needs of implementation by yourself.



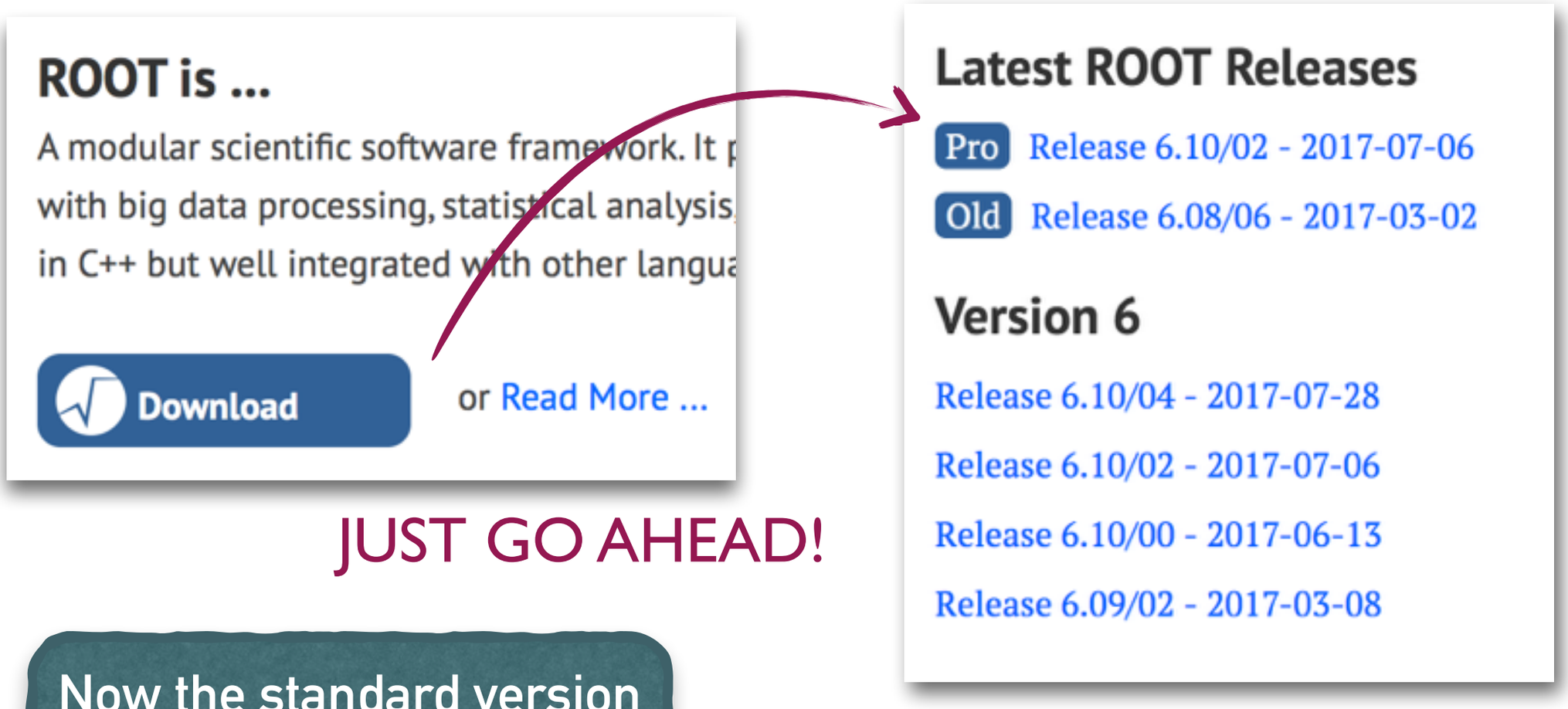
MORE FEATURES TO BE DISCOVERED...

- There are more features to be discovered from the web: http://root.cern.ch
- We are going to show you how to use this tool to carry out your own work during in following lecture time.



GET A WORKING COPY FIRST

- As you connect to the ROOT webpage, you can already find a big icon showing at the left:



JUST GO AHEAD!

Now the standard version is ROOT 6 already.

GET A WORKING COPY FIRST

- For Linux/Mac OSX: Just download the right precompiled binary corresponding to your gcc version.
- For the people who is not using scientific linux/CentOS/Fedora/ Ubuntu, there may be some missing libraries, but in principle they can be solved easily.
- For Windows, not officially supported anymore (it will take a lot of effects to build a windows version…)
- Also a web-based version in beta now!

Source distribution

Platform	Files	Size
source	root_v6.10.02.source.tar.gz	150M

Binary distributions

Platform	Files	Size
CentOS Cern 7 gcc4.8	root_v6.10.02.Linux-centos7-x86_64-gcc4.8.tar.gz	146M
Linux fedora22 gcc5.3	root_v6.10.02.Linux-fedora22-x86_64-gcc5.3.tar.gz	133M
Linux fedora24 gcc6.1	root_v6.10.02.Linux-fedora24-x86_64-gcc6.1.tar.gz	113M
Ubuntu 14 gcc4.8	root_v6.10.02.Linux-ubuntu14-x86_64-gcc4.8.tar.gz	131M
Ubuntu 16 gcc5.4	root_v6.10.02.Linux-ubuntu16-x86_64-gcc5.4.tar.gz	133M
OsX 10.10 clang70	root_v6.10.02.macosx64-10.10-clang70.dmg	115M
OsX 10.10 clang70	root_v6.10.02.macosx64-10.10-clang70.tar.gz	116M
OsX 10.11 clang80	root_v6.10.02.macosx64-10.11-clang80.dmg	120M
OsX 10.11 clang80	root_v6.10.02.macosx64-10.11-clang80.tar.gz	120M
OsX 10.12 clang81	root_v6.10.02.macosx64-10.12-clang81.dmg	128M
OsX 10.12 clang81	root_v6.10.02.macosx64-10.12-clang81.tar.gz	129M

STARTUP FOR LINUX/MAC OS WITH BINARY

- Download and unpack the tar ball, for example:

```
> cd /some/where
> curl -O https://root.cern.ch/download/root_v6.06.08.macosx64-10.10-
clang70.tar.gz
> tar xfvz root_v5.34.32.macosx64-10.10-clang61.tar.gz
> ls root/
LICENSE    bin        emacs      geom       lib        test
README     cmake      etc        icons      macros     tutorials
aclocal    config     fonts      include    man
```

- For linux, you may get the binary tar ball corresponding to your system. But if your system is not listed, you can try to use the binary from a similar system (*note the glibc and gcc version*). You may miss some libraries or version mismatch. The solution will be either make a soft link under your system directory, or build your own binary from the source.

INSTALLING FROM SOURCE

- **Alternatively**: you may want to compile your own ROOT if your system is not listed in the binary list.
- Just follow the instruction below:
  https://root.cern.ch/building-root

Building ROOT

Introduction

ROOT uses the CMake cross-platform build-generator tool as a primary build system. CMake does not build the project, it generates the files needed by your build tool (GNU make, Ninja, Visual Studio, etc) for building ROOT. The classic build with configure/make is is still available but it will not be evolving with the new features of ROOT. The instructions can be found here.

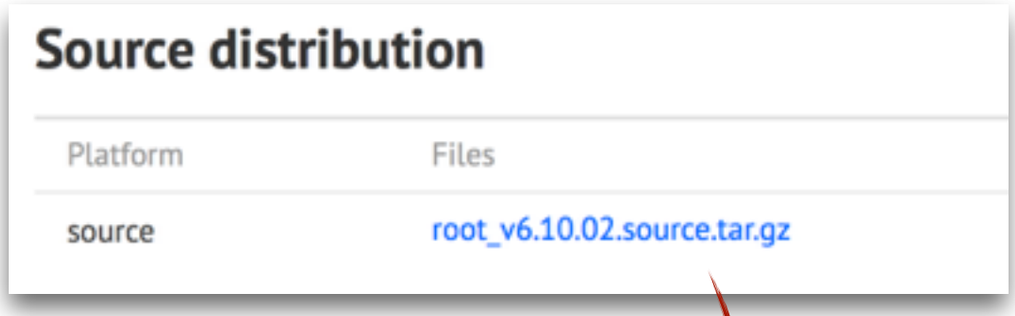
- Note the **prerequisites** are very important:
  https://root.cern.ch/build-prerequisites

Build Prerequisites

The page lists the prerequisite packages that need to be installed on the different platforms to be able to configure and to build basic ROOT. If more advanced ROOT plugins are required look at the `cmake or ./configure` output and add the desired third party packages before running `cmake or ./configure` again.

INSTALLING FROM SOURCE (CONT.)

- Get the source tar ball from the root web:



- Just follow the commands below (classical way, you may find the commands for cmake at the ROOT page):

```
> cd /some/where
> curl -O https://root.cern.ch/download/root_v6.10.02.source.tar.gz
> tar xfvz root_v6.10.02.source.tar.gz
> cd root/
> ./configure –all    (enable any supported libraries, eg. roofit, etc.)
> make                (and just wait...)
```

- It will take a while, please wait (or you can use **`make -j N`**) if you have a N-core CPU.
- If no error messages shown at the end of “make”, you are ready to use it!

ROOT STARTUP

- In order to use the ROOT under the terminal, we have to setup the environment:

```
> source /some/where/root/bin/thisroot.csh        (for tcsh)
or
> source /some/where/root/bin/thisroot.sh         (for bash)
```

- Alternatively you can setup the environment (*for tcsh*) as following:

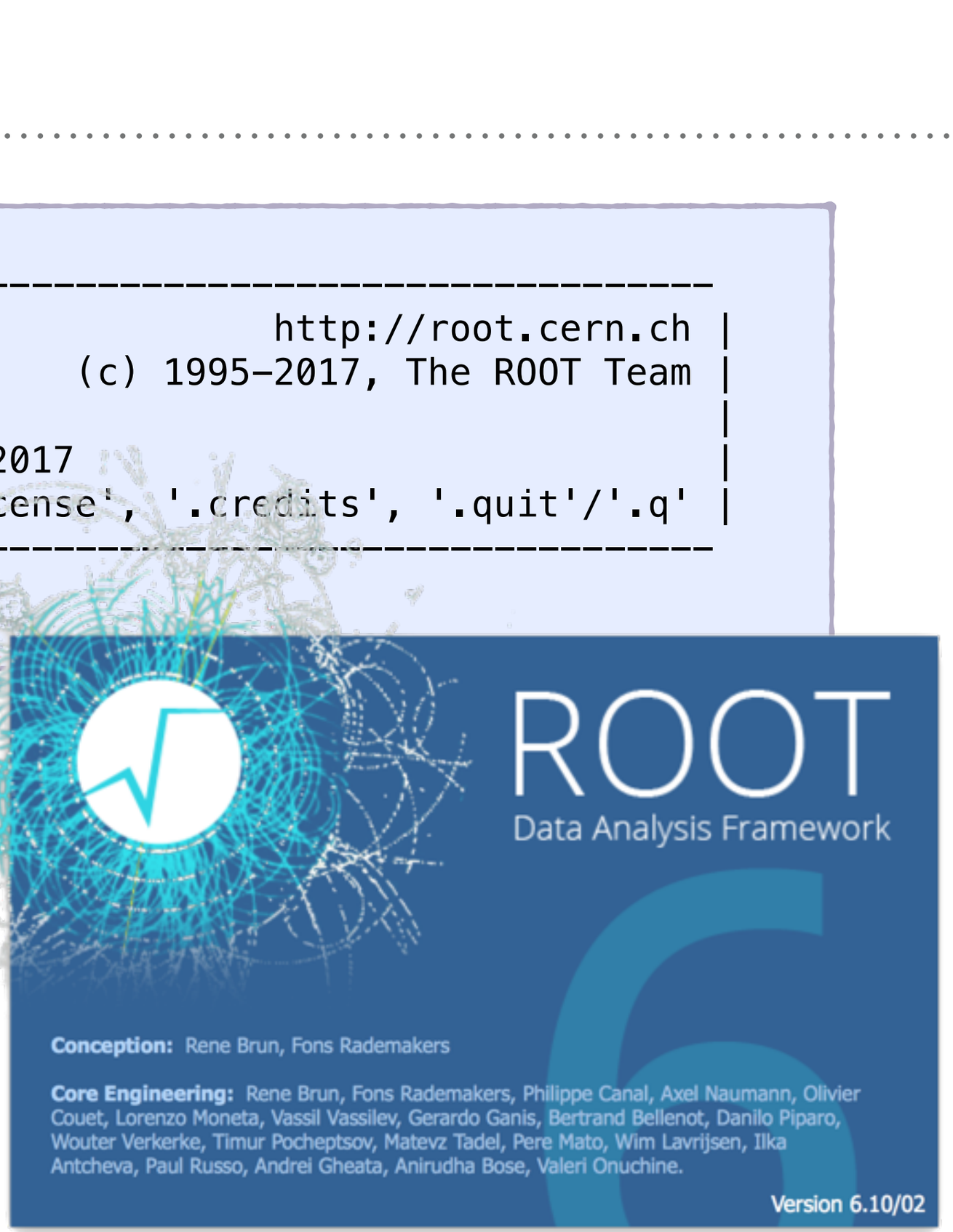
```
> setenv ROOTSYS /some/where/root
> setenv PATH "${ROOTSYS}/bin:$PATH"
> setenv LD_LIBRARY_PATH "${ROOTSYS}/lib"
```

- For bash, you have to replace the **"setenv"** with "**export**", and plus a "**=**" right after the variable name:

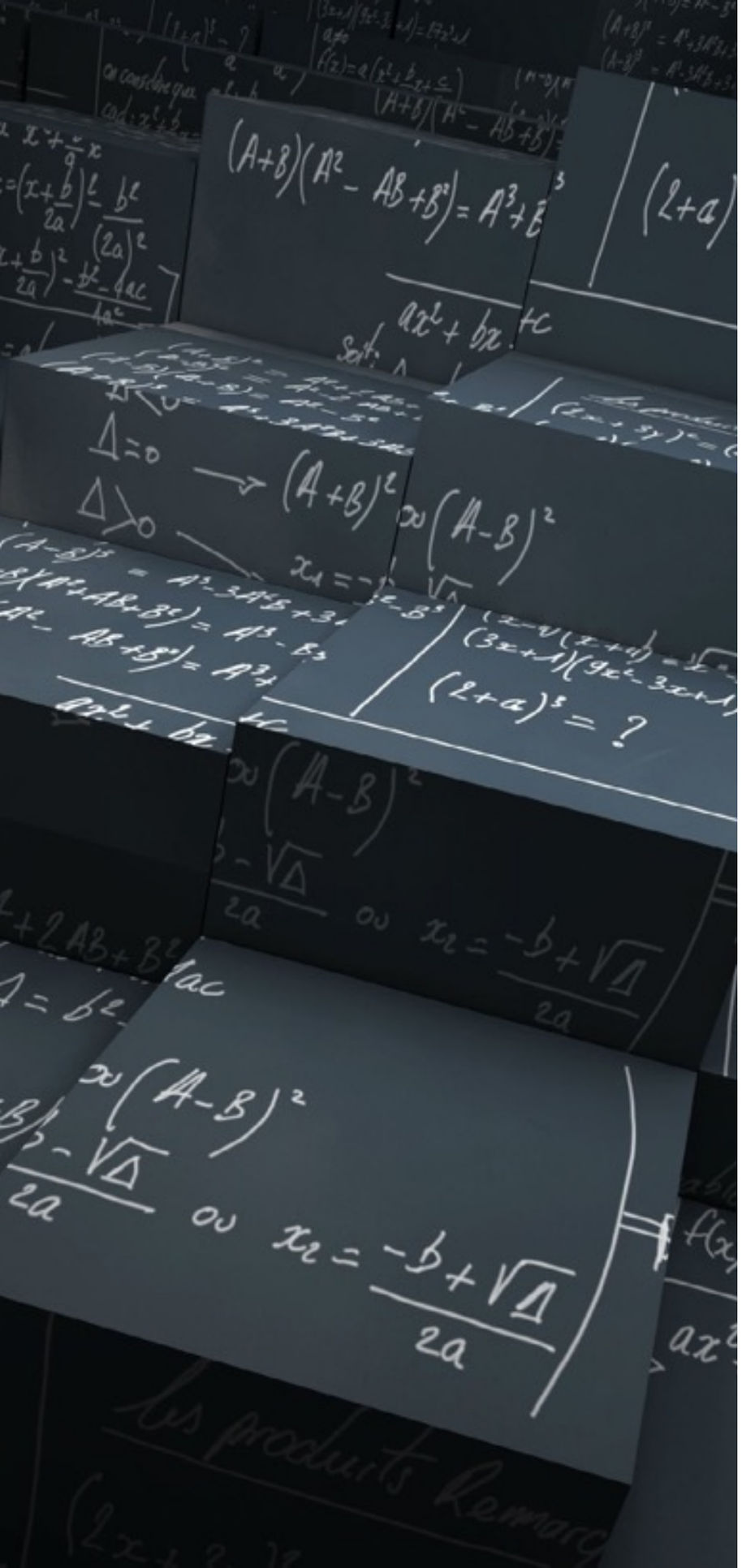
```
> export ROOTSYS=/some/where/root
> export PATH="${ROOTSYS}/bin:$PATH"
> export LD_LIBRARY_PATH="${ROOTSYS}/lib"
```

THEN...JUST LAUNCH IT!

```
root
   ------------------------------------------------------------
  | Welcome to ROOT 6.10/02                        http://root.cern.ch |
  |                                  (c) 1995-2017, The ROOT Team |
  | Built for macosx64                                            |
  | From tag v6-10-02, 6 July 2017                                |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'    |
   ------------------------------------------------------------

root [0]
```



If you have got it setup correctly!



INTERMISSION

- Time to get your ROOT working!
- For experts, please help someone who cannot get his/her system work.
- Remarks for Windows: it is very difficult to have ROOT fully functioning in general. Linux or Mac OSX are strongly recommended. You may want to install a copy of linux on your laptop if your default OS is Windows.

ROOT AS A POCKET CALCULATOR

- You may just use the ROOT prompt as a calculator:

```
root [0] exp(10)
(std::enable_if<true, double>::type) 22026.5
root [1] sqrt(28)
(std::enable_if<true, double>::type) 5.29150
root [2] double x = 0.54;
root [3] x
(double) 0.540000
root [4] x+x*x+x*x*x
(double) 0.989064
root [5] printf("%f\n",sin(x));
0.514136
root [6] for(int i=0;i<5;i++) cout << i << endl;
0
1
2
3
4
root [7] _
```

SIMPLE BUT IMPORTANT OPERATIONS

- The most important command: how to **quit** root:

```
root [0] .q
```

- A command list – help page

```
root [0] .help
```

- Go back to csh/sh temporary

```
root [0] .!csh
```

- Execute a shell command

```
root [0] .! <command>
```

- Reset ROOT

```
root [0] gROOT->Reset();
```

ROOT PROMPT

- Probably you already noticed that, the root prompt is actually based on C/C++, except those special commands starting with a **“.”**.
- This interactive prompt is called CLING (a C++ interpreter in fact), in principle all your codes can be written in C/C++ and execute them directly. *Remark: ROOT version 5 and before is using a CINT interpreter, this is one of the reason why ROOT 5 and 6 are so different.*
- Basically there are some variations (*python, R, etc*) for those who does not like to use C/C++ as a script language.
- Why not GUI interface? There are some fine GUI for tuning your figures, fits, others, but it is still recommended to go the work in scripts/codes.
- A line from ROOT developers: **ROOT is not an office suite, but a programming framework!**

RUNNING UNDER CLING

- A minimum example with a function (hello, world?):

example_03.cc

```
void example_03(double x)
{
   printf("x*x = %g\n",x*x);
}
```

NOTE: the entry point is not **main()** but **example_03()**

- Let's run it with "**.x**":

```
root [0] .x example_03.cc(2.5)
x*x = 6.25
```

- This is equivalent to load it ("**.L**") and run it:

```
root [0] .L example_03.cc
root [1] example_03(2.5)
x*x = 6.25
```

RUNNING UNDER ACLIC

- Direct execute with Cling:

```
root [0] .x example_03.cc(2.5)
```

- For a boost in speed & syntax check, it's recommended to compile it first, and then execute it:

```
root [0] .x example_03.cc+(2.5)
Info in <TMacOSXSystem::ACLiC>: creating shared library /Users/
kfjack/Workspace/course/hep_statistics/slides/inter-1/./
example_03_cc.so
```

- If your code has been compiled already, it just runs immediately!

```
root [0] .x example_03.cc+(2.5)
```

This running mode is not called CLING anymore, but
ACLiC (Automatic Compiler of Libraries for CLING/CINT)

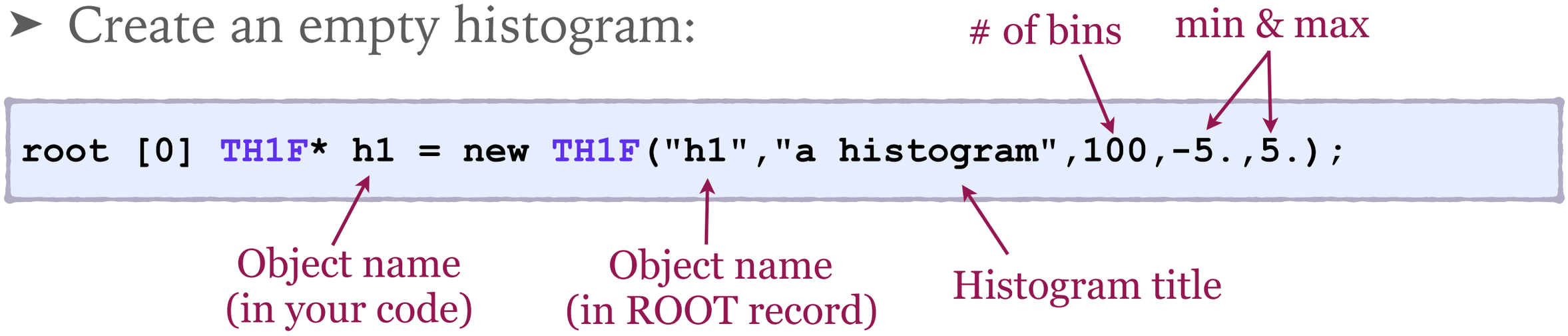
CLING/CINT VERSUS ACLIC

CLING/CINT	ACLIC
Interpreter	(Pre-)compiled
Slower	Faster
Run with partial code	Coding with full headers, C/C++ rules
No grammar checks	A full check before execute
Easy object allocation, freely	A full tight code (prepare for segment violation!)

- General guide lines:
  - If your code needs lots of for/while loops, just compile it.
  - If you just want to write a lazy script, keep it as CLING. CLING already does a good job for most of plotting/graphics works.
  - Restriction of ACLiC: need to include full headers.

SOME BASIC HISTOGRAM OPERATIONS

- Create an empty histogram:



- Make a clone:

```
root [1] TH1F* h2 = (TH1F*)h1->Clone("h2");
```

- List the objects we just created:

```
root [2] .ls
 OBJ: TH1F h1 a histogram : 0 at: 0x7fe60c3e98a0
 OBJ: TH1F h2 a histogram : 0 at: 0x7fe60c445730
```

SOME BASIC HISTOGRAM OPERATIONS (II)

- Let's fill them with random numbers:

Initial a generator with seed = 1234
(Algorithm: Mersenne Twister)

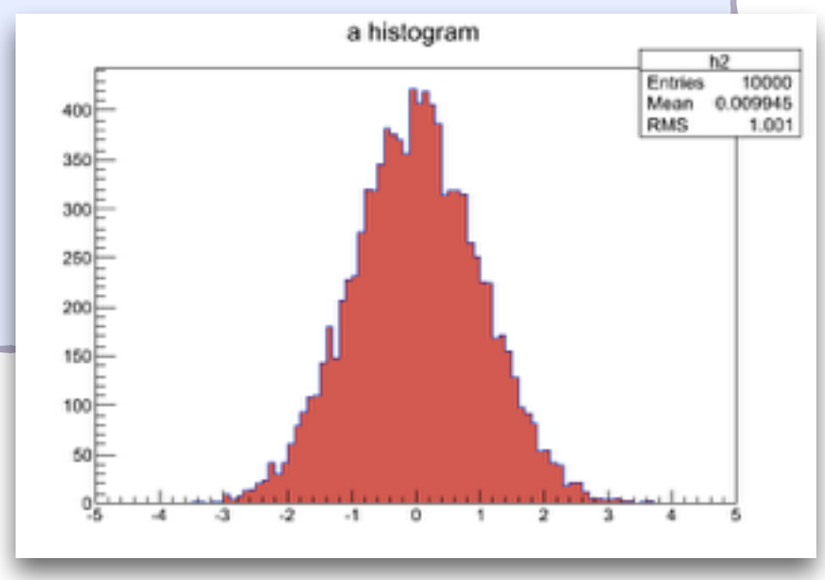
```
root [3] TRandom3 rnd(1234);
root [4] for(int i=0;i<10000;i++) h1->Fill(rnd.Uniform(-5.,5.));
root [5] for(int i=0;i<10000;i++) h2->Fill(rnd.Gaus(0.,1.));
```

Loop & fill for 10,000 times

Uniform generator (min,max) &
Gaussian generator (mean,width)

- Plot them:

```
root [6] h2->Draw();
root [7] h1->Draw();
root [8] h1->GetYaxis()->SetRangeUser(0.,150.);
root [9] h1->Draw()
```



SOME BASIC HISTOGRAM OPERATIONS (III)

- Direct operations: (+/−/×/÷)

```
root [10] TH1F* h3 = (TH1F*)h1->Clone("h3");
root [11] h3->Add(h2);
root [12] TH1F* h4 = (TH1F*)h1->Clone("h4");
root [13] h4->Add(h2,-1.);
root [14] TH1F* h5 = (TH1F*)h1->Clone("h5");
root [15] h5->Multiply(h1);
root [16] TH1F* h6 = (TH1F*)h1->Clone("h6");
root [17] h6->Divide(h1);
```

- Normalize it:

```
root [18] h1->SetNormFactor();
```

- Draw the error bar

```
root [19] h2->Draw("e");
```

SOME BASIC HISTOGRAM OPERATIONS (IV)

- Obtain/set the mean value/error of a specific bin:

```
root [20] h1->GetBinContent(5)
(double) 94.0000
root [21] h1->GetBinError(5)
(double) 9.69536
root [22] h1->SetBinContent(5,128.0)
root [23] h1->SetBinError(5,0.05)
root [24] h1->GetBinContent(5)
(double) 128.000
```

- Reset (*clean up*) the histogram:

```
root [25] h1->Reset()
```

- Get total entries:

```
root [26] h2->GetEntries()
```

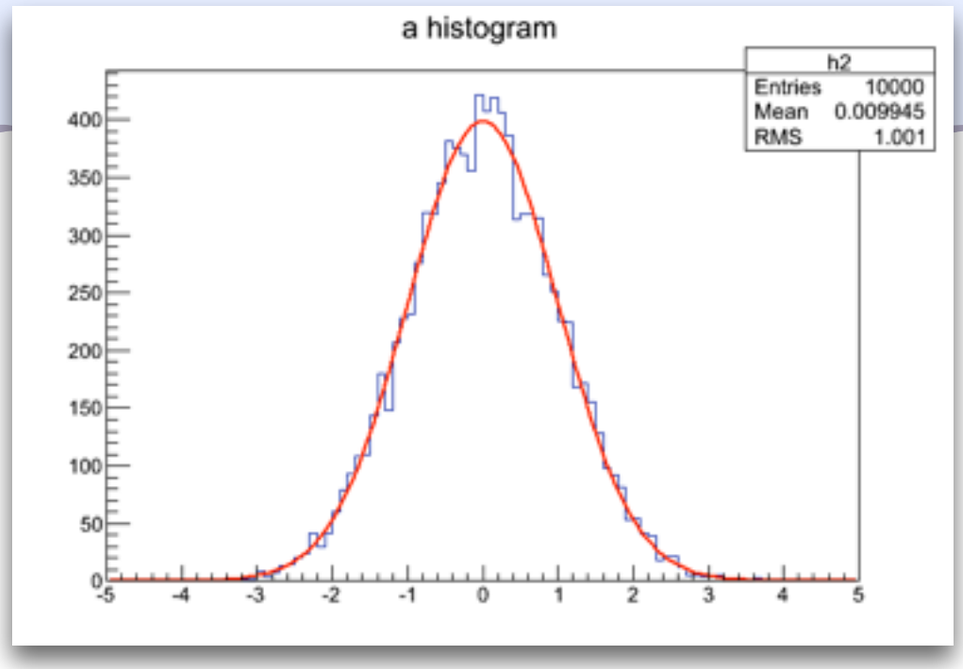
SOME BASIC HISTOGRAM OPERATIONS (V)

- A quick fit (we will come back to the fitting in one of the later lectures)

```
root [26] h2->Fit("gaus");
 FCN=76.7734 FROM MIGRAD    STATUS=CONVERGED      59 CALLS          60
TOTAL
                     EDM=1.60726e-09    STRATEGY= 1      ERROR MATRIX
ACCURATE
  EXT PARAMETER                                          STEP          FIRST
  NO.   NAME      VALUE            ERROR                 SIZE      DERIVATIVE
   1  Constant     3.98520e+02   4.87828e+00   1.72269e-02   1.26437e-05
   2  Mean         1.09934e-02   1.00185e-02   4.31207e-05  -1.81996e-03
   3  Sigma        9.93964e-01   7.01172e-03   8.26736e-06   2.29756e-02
(class TFitResultPtr)140246138123936
root [27]
```

A simple fit with Gaussian distribution:

$$\Gamma \propto \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$



2D HISTOGRAMS

- Working with 2D histograms is also very straightforward:

```
TH2F* h2d = new TH2F("h2d","a 2d histogram",20,-5.,5.,20,-5.,5.);
```

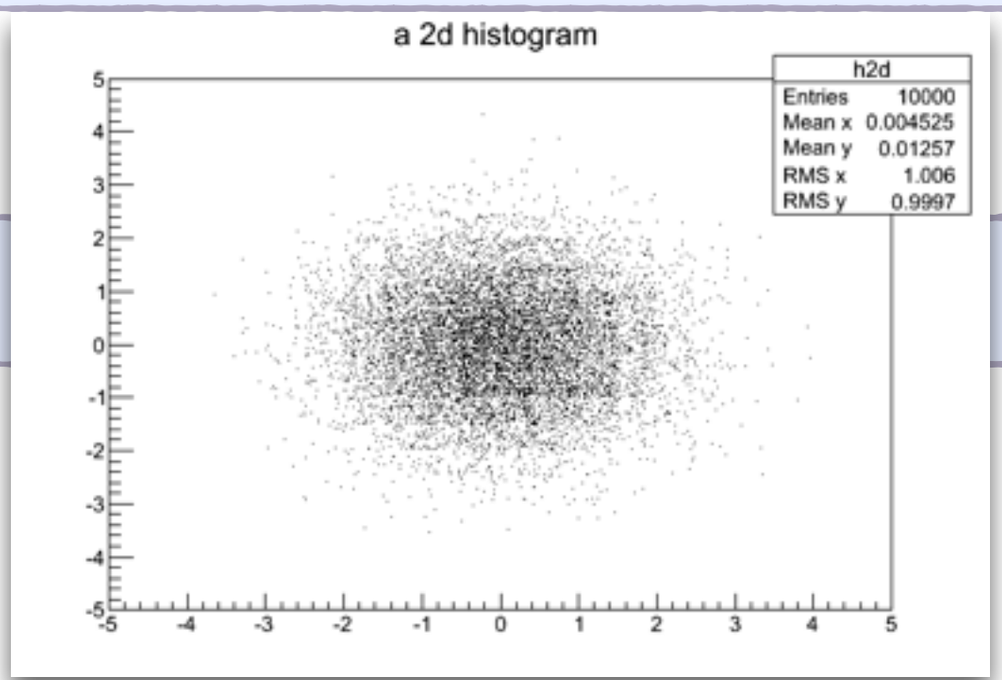
# of bins (x) — min & max (x) — # of bins (y) — min & max (y)

- Let's fill it with some random numbers:

```
for(int i=0;i<10000;i++) h2d->Fill(rnd.Gaus(0,1),rnd.Gaus(0,1));
```

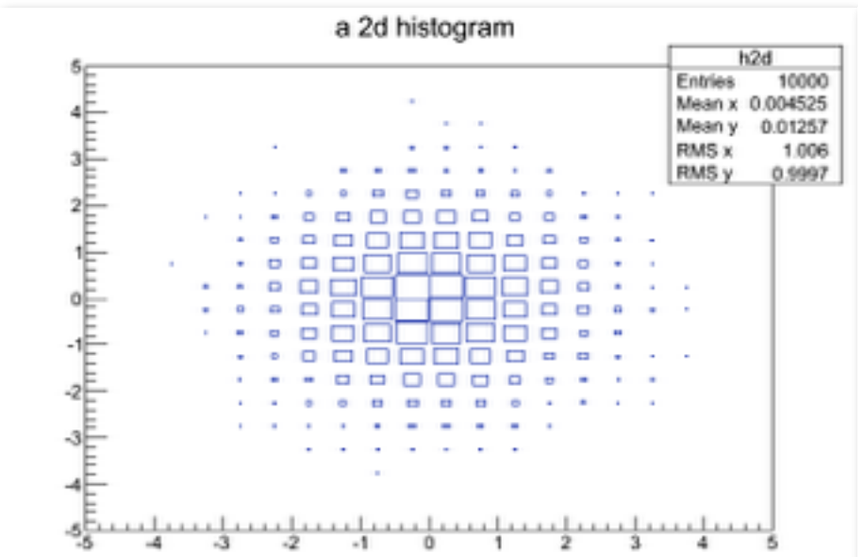
- Just plot it:

```
h2d->Draw();
```

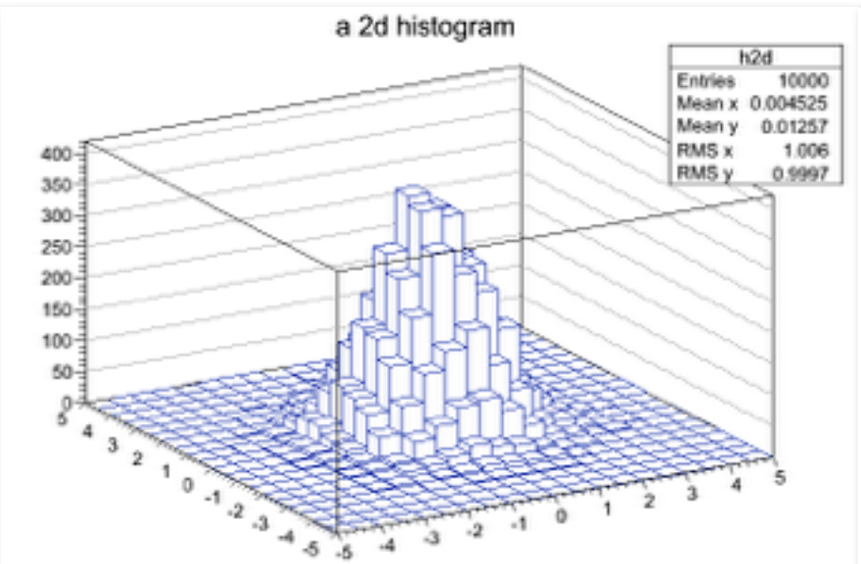


2D HISTOGRAMS: DRAWING OPTIONS

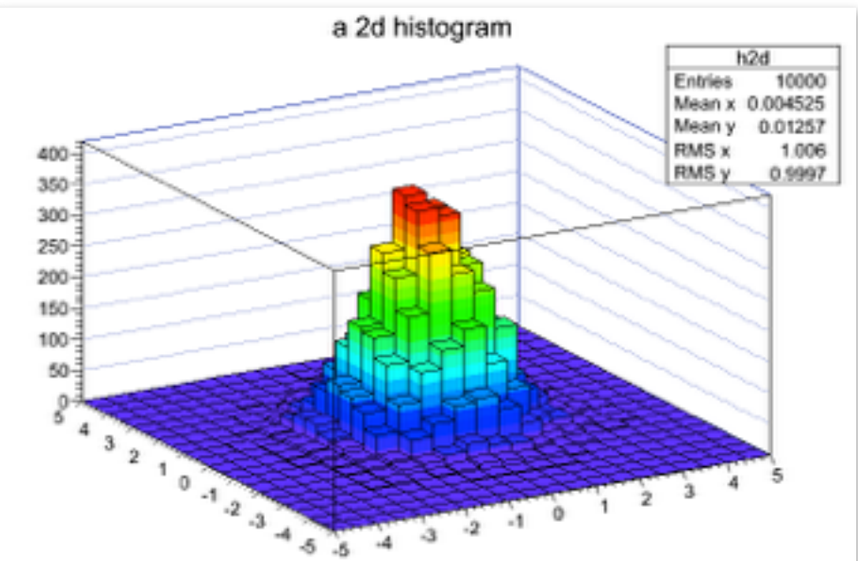
```
h2d->Draw("box");
```



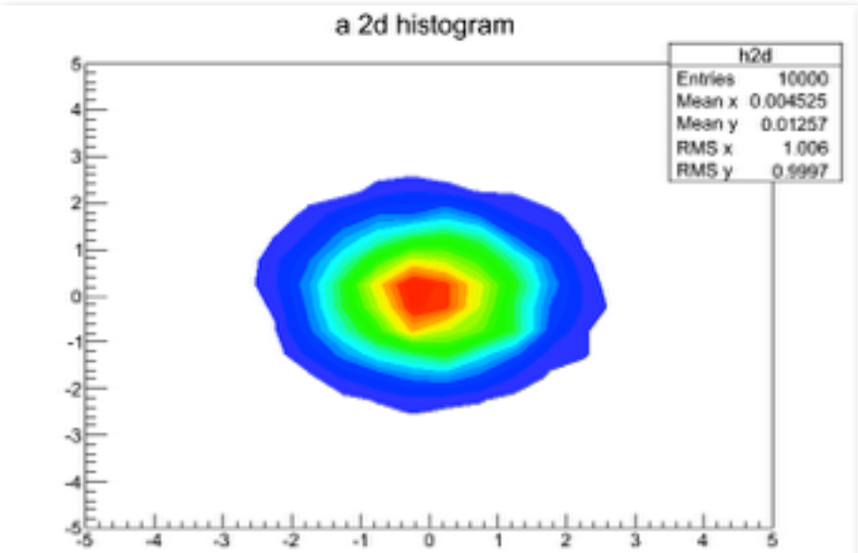

```
h2d->Draw("lego");
```



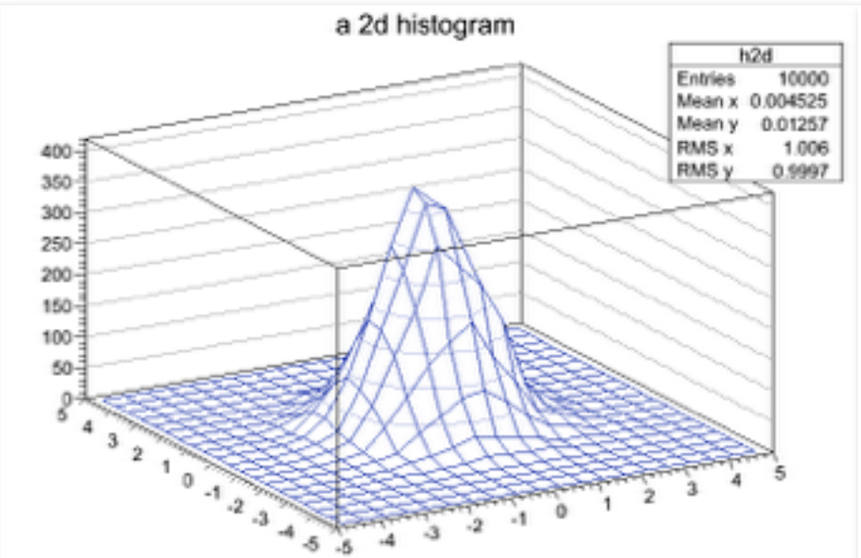

```
h2d->Draw("lego2");
```



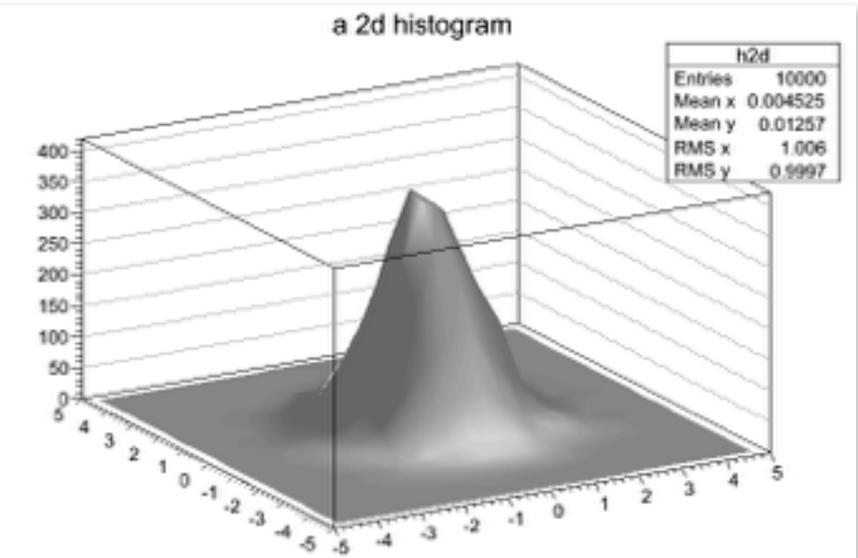

```
h2d->Draw("cont");
```




```
h2d->Draw("surf");
```




```
h2d->Draw("surf4");
```



FUNCTIONS

- What should we do if we just want to plot a function, e.g.:

$$f(x) = x + x^2 + x^3 + x^4 + \sin(5\pi x) + \exp(x)$$

```
TF1* f1 = new TF1("f1","x+x^2+x^3+x^4+sin(5*pi*x)+exp(x)",0.,1.);
```

function object name

free function form

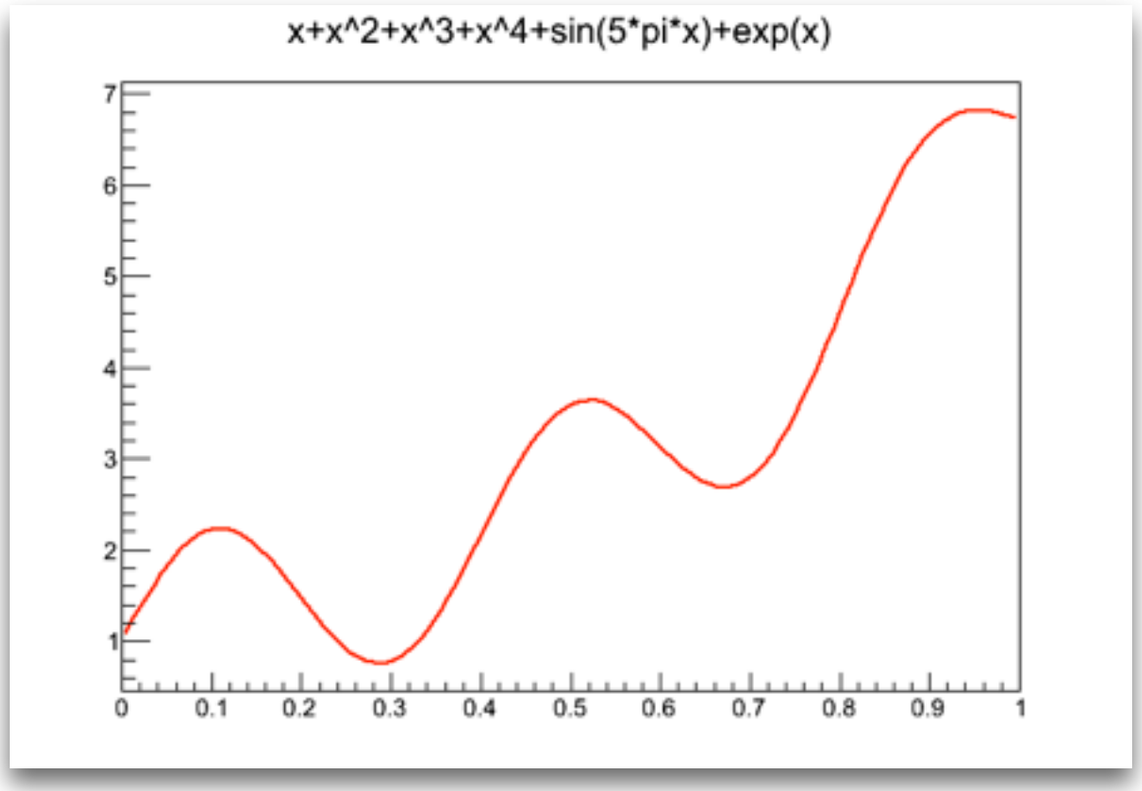
min & max

- Just plot it:

```
f1->Draw()
```

- Get the function value at x=0.5

```
f1->Eval(0.5)
```



FUNCTIONS (CONT.)

- Then we could also practice those "hands-on shows" at the beginning of this lecture:
  - Derivative x = 0.5:

```
root [30] f1->Derivative(0.5)
(double) 4.898708
```

  - Integration in an interval, e.g. [0,1]:

```
root [31] f1->Integral(0.,1.)
(double) 3.128939
```

  - Generate a random number according to this function:

```
root [32] f1->GetRandom()
(double) 0.999880
```

FILL RANDOM HISTOGRAM

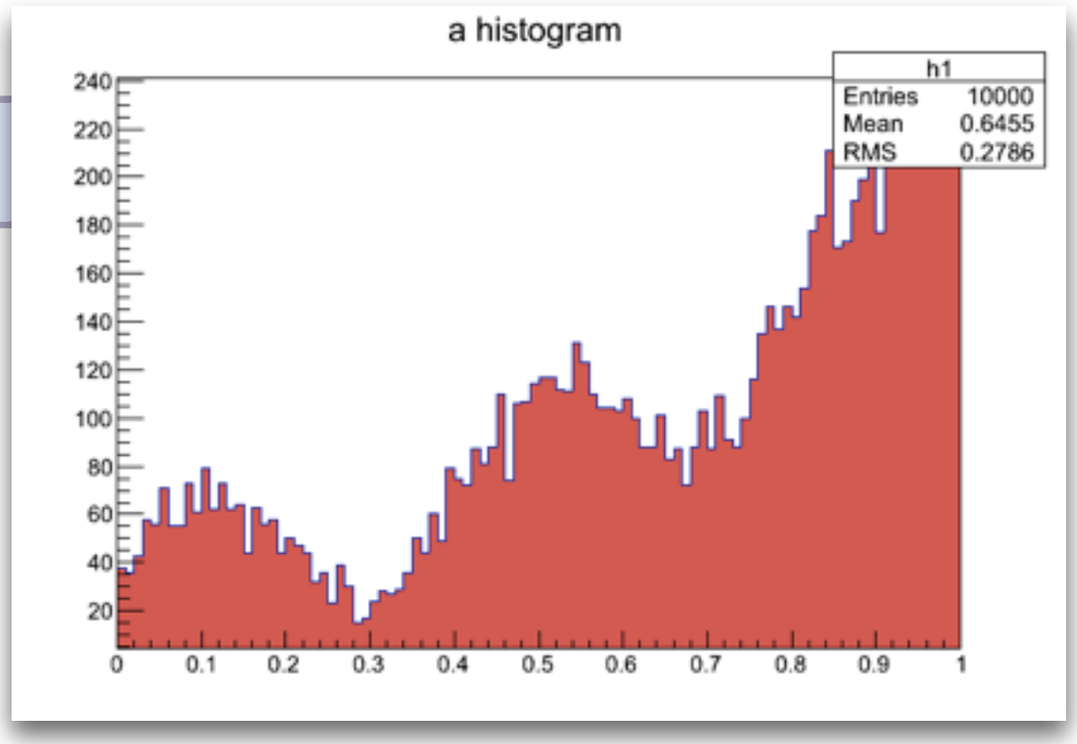
- We could fill a histogram with the random numbers generated according to such a specific function:
  - Create a 1D histogram and fill:

```
root [33] TH1F* h1 = new TH1F("h1","a histogram",100,0.,1.);
root [34] for(int i=0;i<10000;i++) h1->Fill(f1->GetRandom());
```

  - This is the same as

```
root [35] h1->FillRandom("f1",10000);
```

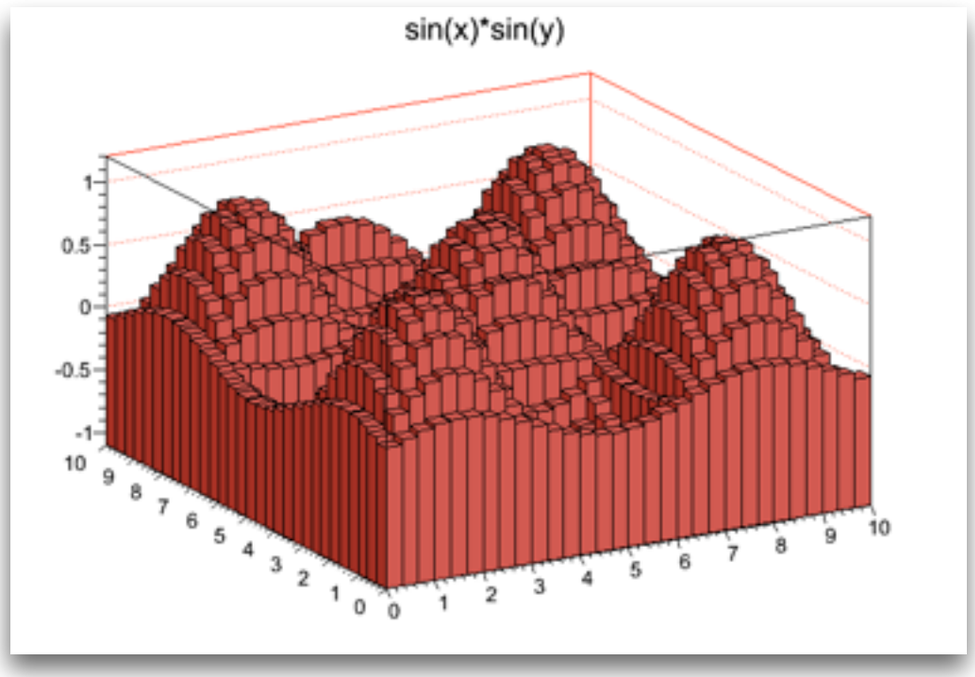
  - If you draw it:

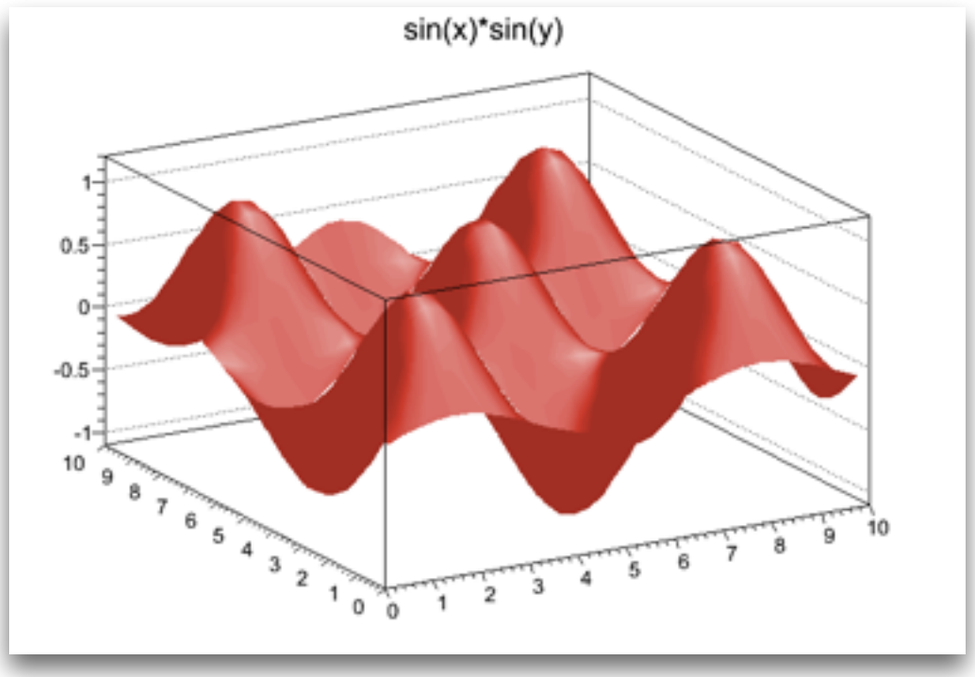


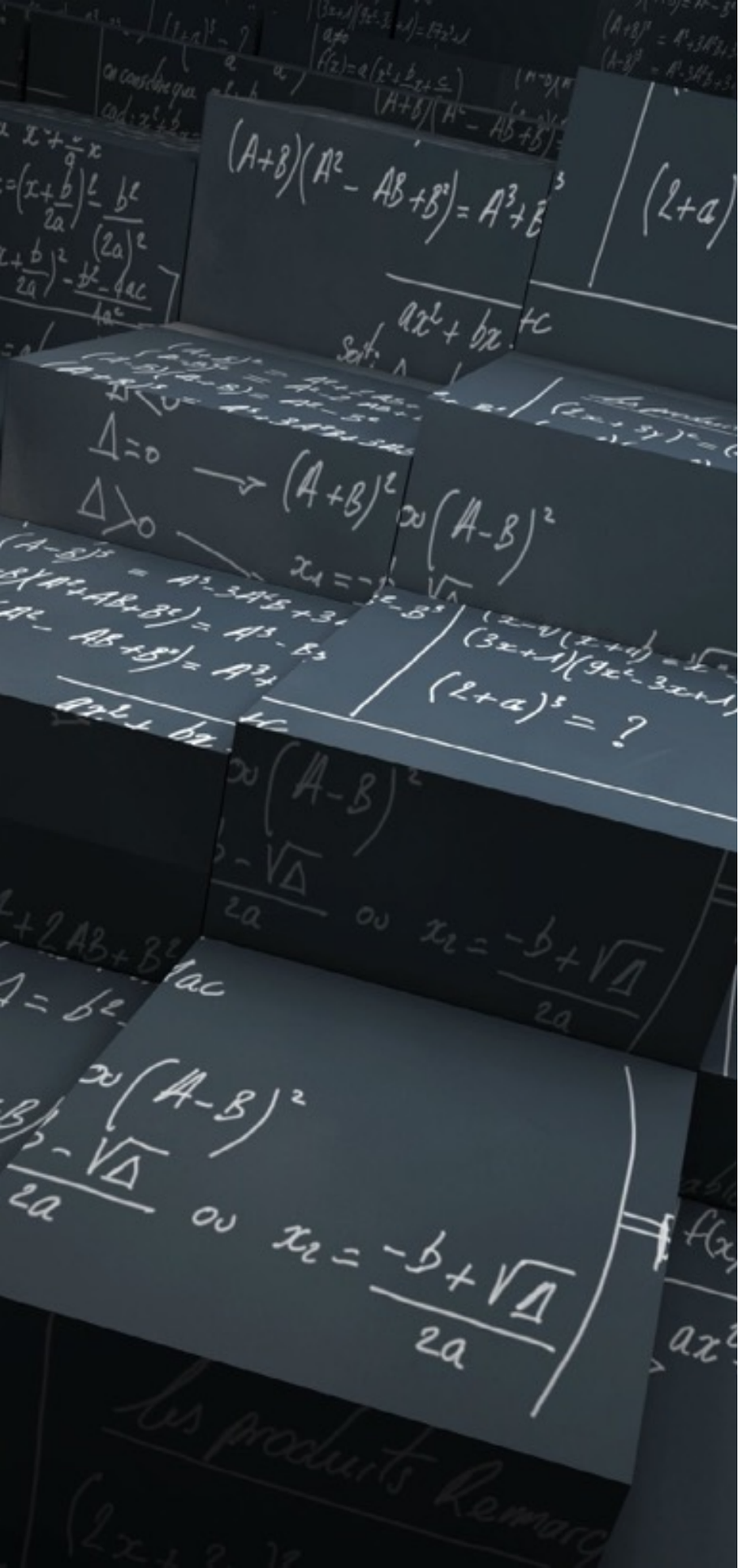
2D FUNCTIONS

- The 2D function is not really different from the 1D functions: (as we already see the example earlier in fact)

```
root [36] TF2* f2 = new TF2("f2","sin(x)*sin(y)",0,10,0,10);
root [37] f2->SetFillColor(50);
root [38] f2->Draw("lego1");
root [39] f2->Draw("surf4");
```





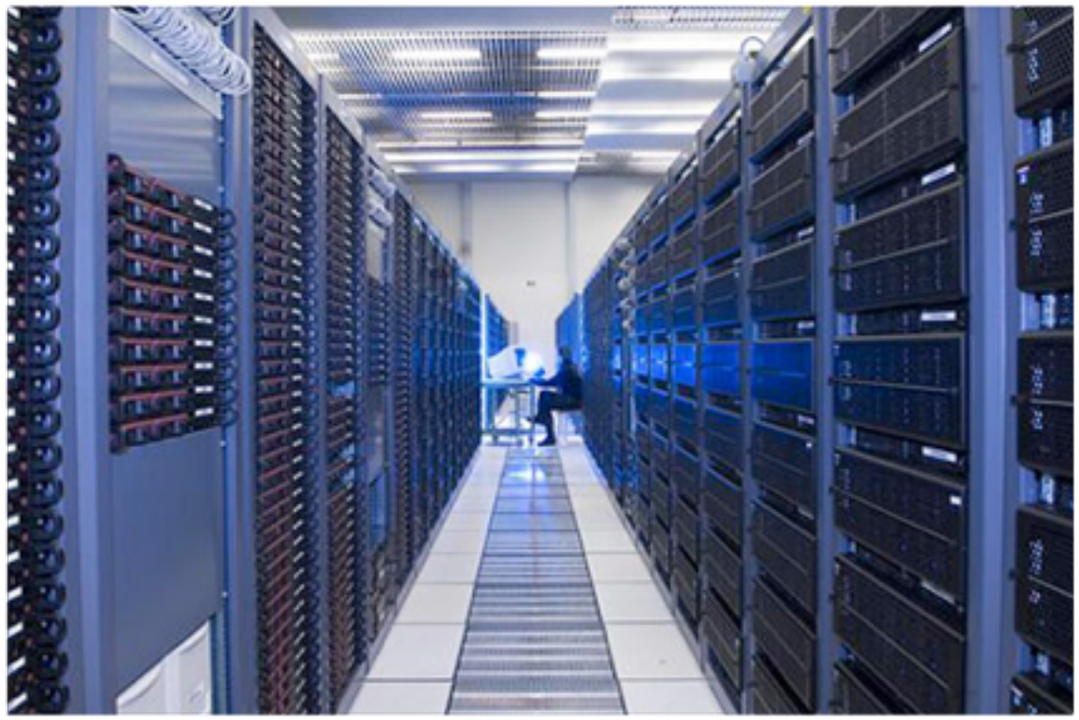


INTERMISSION

- Now we have went though the basic introduction for ROOT.
- It should be above to give the minimal capability to produce the required objects and figures.
- In the later half of this lecture we will discuss the basic operation of “unbinned" data set with ROOT.

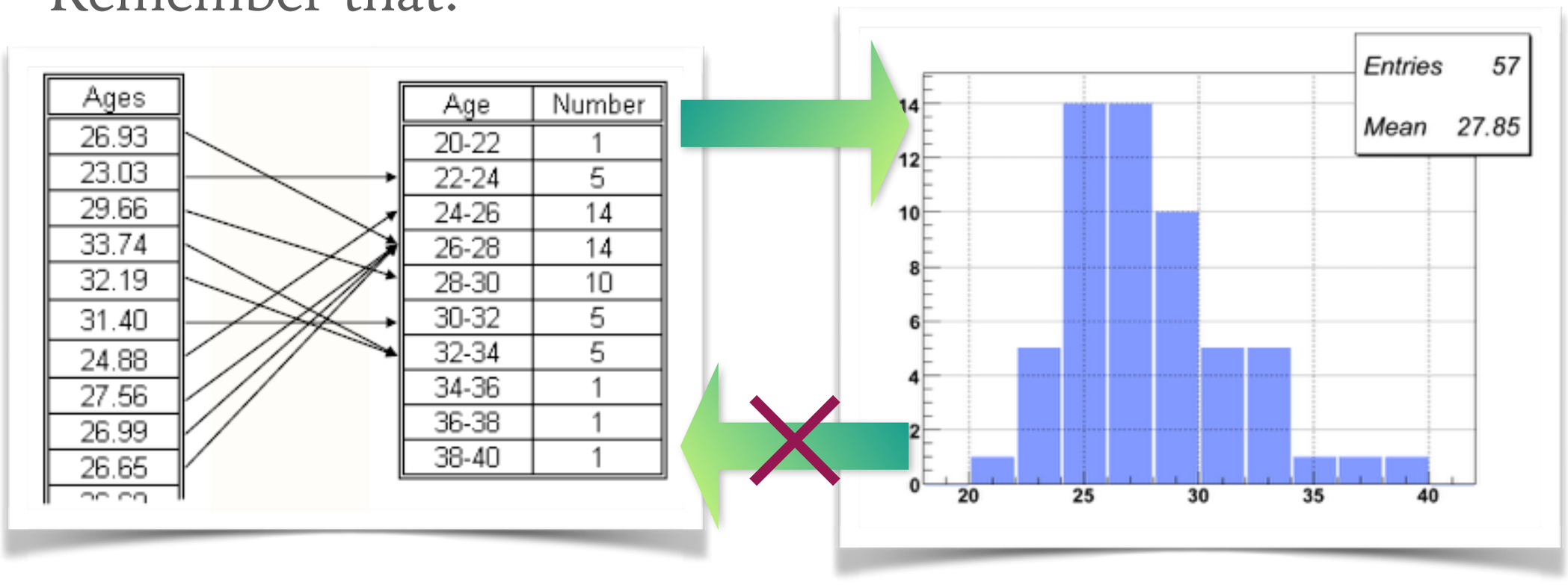
ROOT: FRAMEWORK FOR LARGE SCALE DATA HANDLING

- Last lecture we have only touched the basic commands for ROOT, but nothing really on the data handling.
- ROOT now is the official tool used by the LHC experiments, and the nominal data flow is O(PETABYTE).
- In principle, handling any data sets smaller than LHC should be **a piece of cake**? 🍰
  (*well, not really – we still need a large computing farm…*)



DATA IN HISTOGRAMS

- We already learn how to work with histograms, but:
  - Histograms only store processed data with partial information.
  - Remember that:



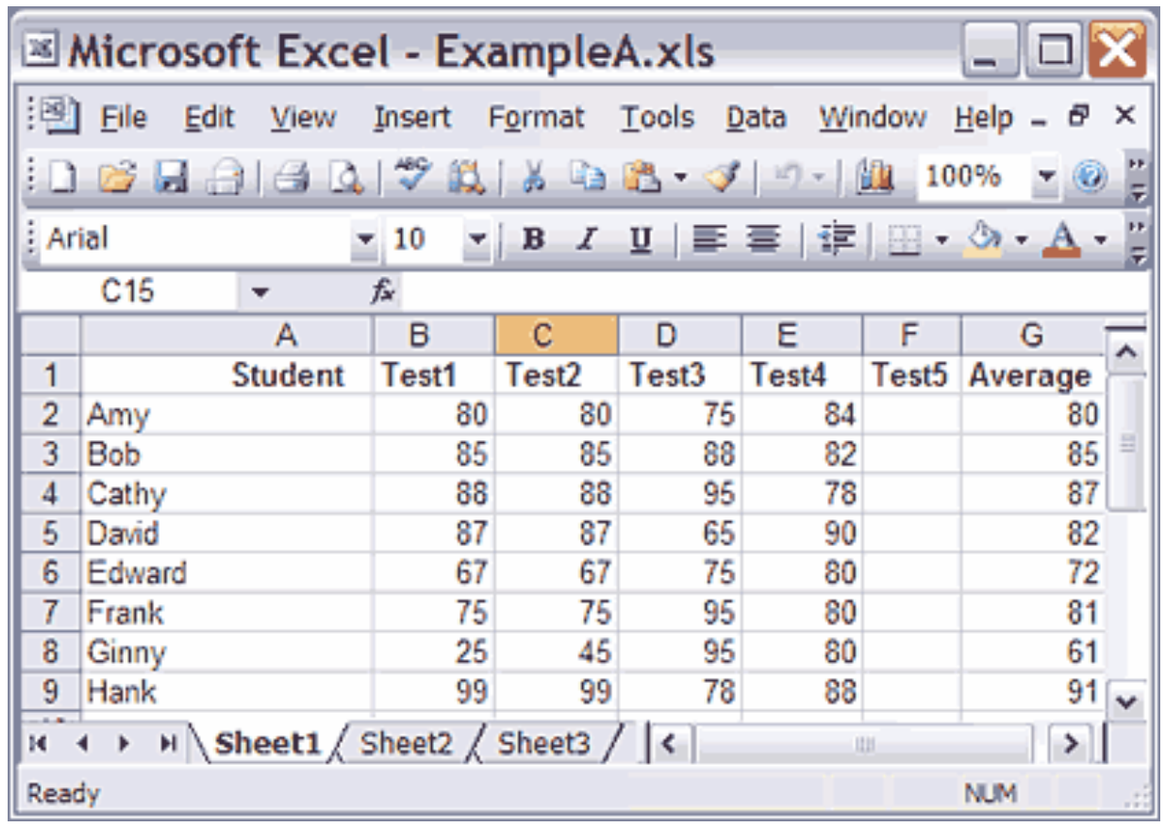
Although we could check the statistics for each age-range, but we already lost how old of each person!

"UNBINNED" DATA

- The better way to preserve information is using the unbinned data format:

  Physics
  Analysis
  Workstation
  ↓

  - **N-tuple**: an old format for unbinned data set since **PAW**.
    [A common analysis tool used in HEP before ROOT (198x~199x)]
  - **Tree**: ROOT default format for unbinned data (*an obvious name?*).



	A	B	C	D	E	F	G
1	Student	Test1	Test2	Test3	Test4	Test5	Average
2	Amy	80	80	75	84		80
3	Bob	85	85	88	82		85
4	Cathy	88	88	95	78		87
5	David	87	87	65	90		82
6	Edward	67	67	75	80		72
7	Frank	75	75	95	80		81
8	Ginny	25	45	95	80		61
9	Hank	99	99	78	88		91

It's just like an excel data-sheet;
We are maintaining the full information for every row or column.

DATA IN HISTOGRAMS (A DEBATE)

- On the other hand, histograms are cheap because **it does not require a large storage space**.
- In a more precise description, the memory occupancy does not increase with data size:
  - As the previous example, if we are binning the ages with an interval of 2 years, then a collection of 100 years data only need 50 integers (200 bytes or 400 bytes).
- And this memory occupancy is the same for 10 people, 100 people, 1000 people, 1 million, 1 billion, or even more.
- It depends on the actual needs. An usual case is that the data starts from unbinned data and converted to histograms in order to see some statistical effects.

USING N-TUPLES UNDER ROOT

- N-tuples are still very useful under ROOT, even this is a legacy idea from PAW.
- N-tuples store event-by-event information, and each event consists with an array of float point numbers:

	X	Y	Z	Alpha	Beta	Gamma
Event 1	0.135	0.472	0.567	1.56	2.73	3.15
Event 2	0.725	0.177	0.736	1.73	2.52	3.67
Event 3	0.114	0.156	0.992	1.94	2.45	3.12
… …						
Event n	0.626	0.362	0.123	1.114	2.23	3.19
… …						

One event has an array of float point numbers.

- It is obvious that more events = more storage space, but we kept the full information.

USING N-TUPLES UNDER ROOT (II)

- Let's produce a random n-tuple using a simple code as following:

example_04.cc

```
{
    TNtuple *n1 = new TNtuple("n1","a n-tuple","x:y:z:alpha:beta:gamma");
    TRandom3 rnd;

    float var[6];
    for(int i=0;i<10000;i++) {
        var[0] = rnd.Uniform(0.,1);
        var[1] = rnd.Gaus(0.,1.);
        var[2] = rnd.Exp(-1.);
        var[3] = var[0]*var[1];
        var[4] = var[0]*var[2];
        var[5] = var[0]*var[1]*var[2];

        n1->Fill(var);
    }
}
```

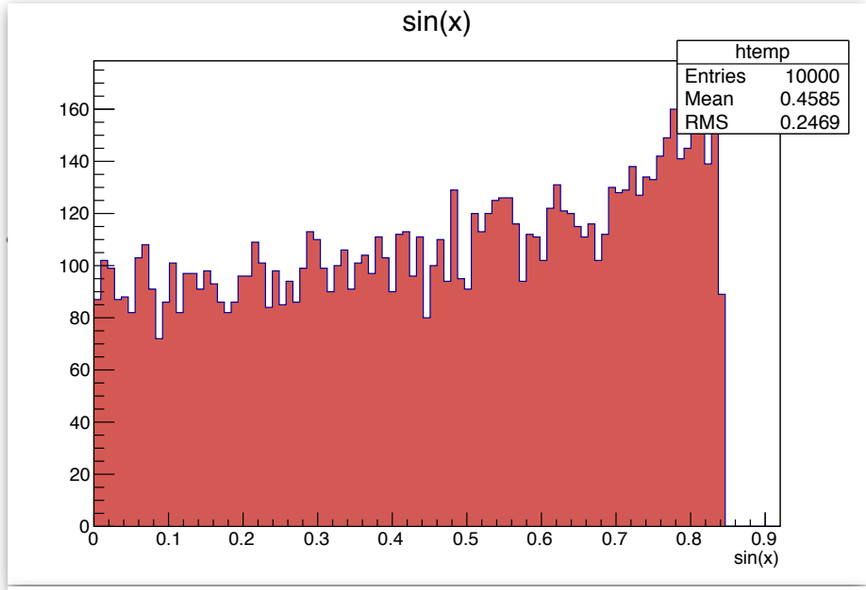
Fill x,y,z with some random distributions

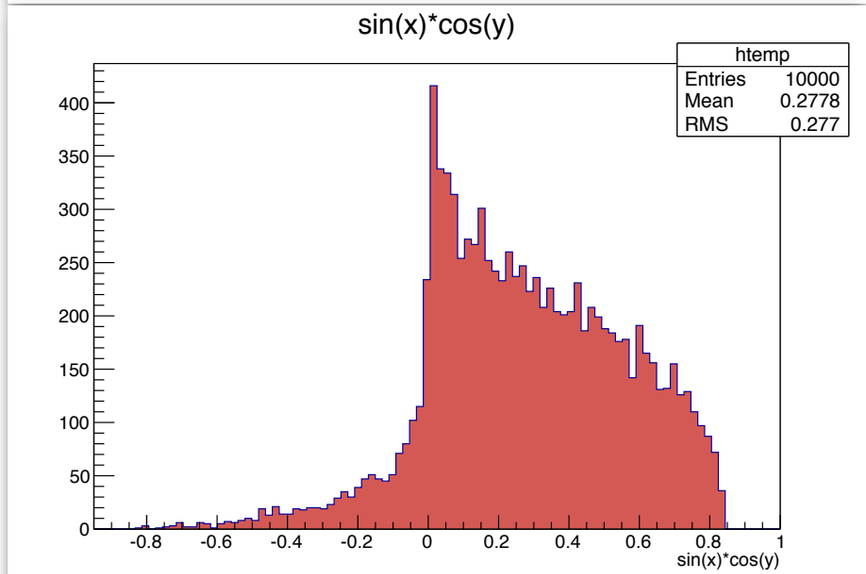
Fill alpha,beta,gamma with some combination of x,y,z

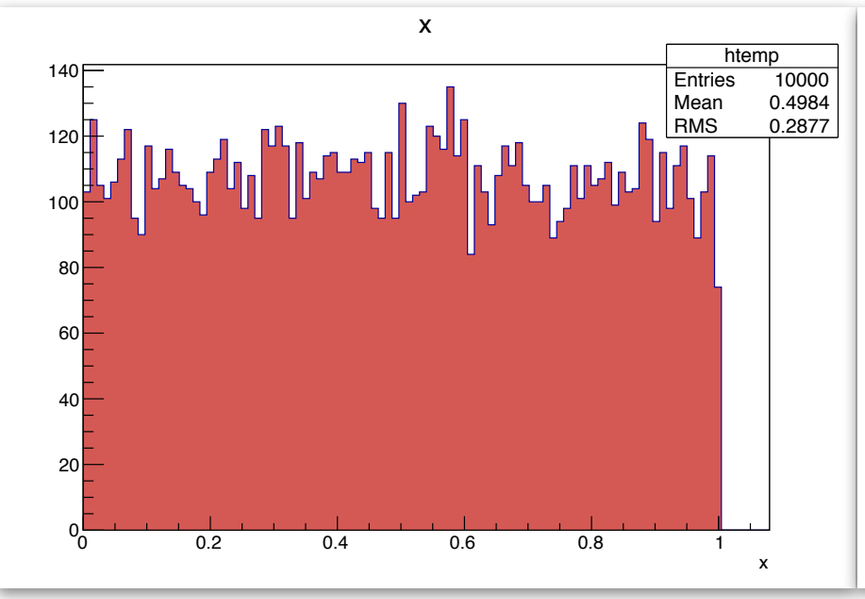
USING N-TUPLES UNDER ROOT (III)

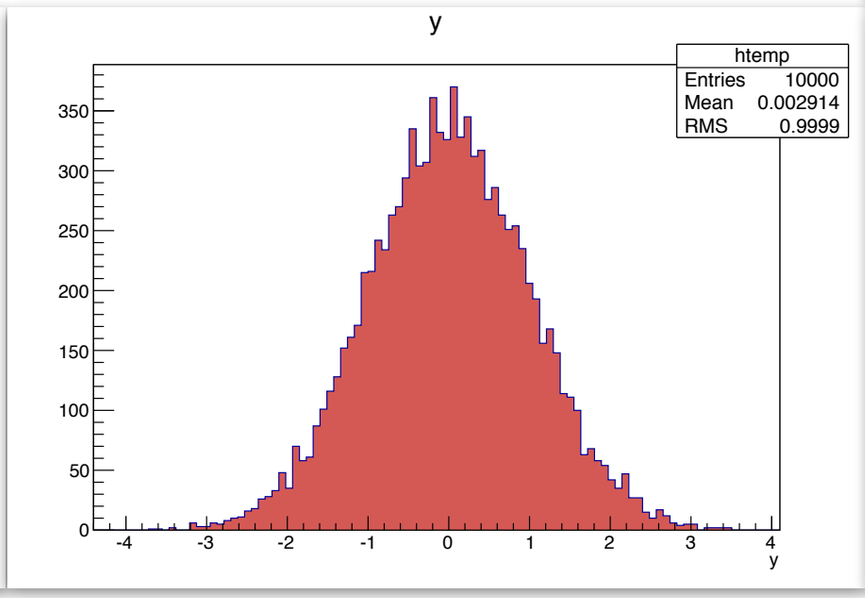
- Let's try it!

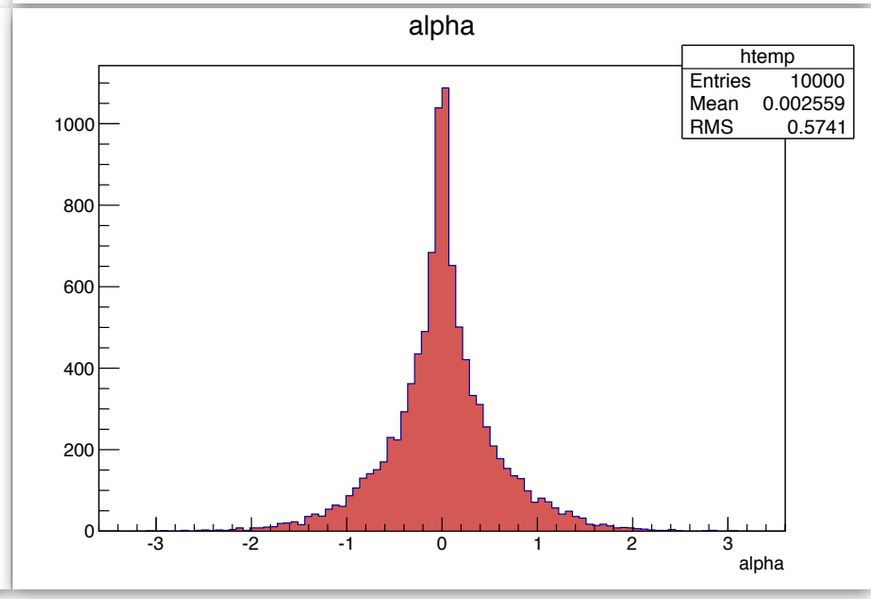
```
root [0] .x example-01.cc
root [1] n1->SetFillColor(50)
root [2] n1->Draw("x")
root [3] n1->Draw("y")
root [4] n1->Draw("alpha")
root [5] n1->Draw("sin(x)")
root [6] n1->Draw("sin(x)*cos(y)")
```







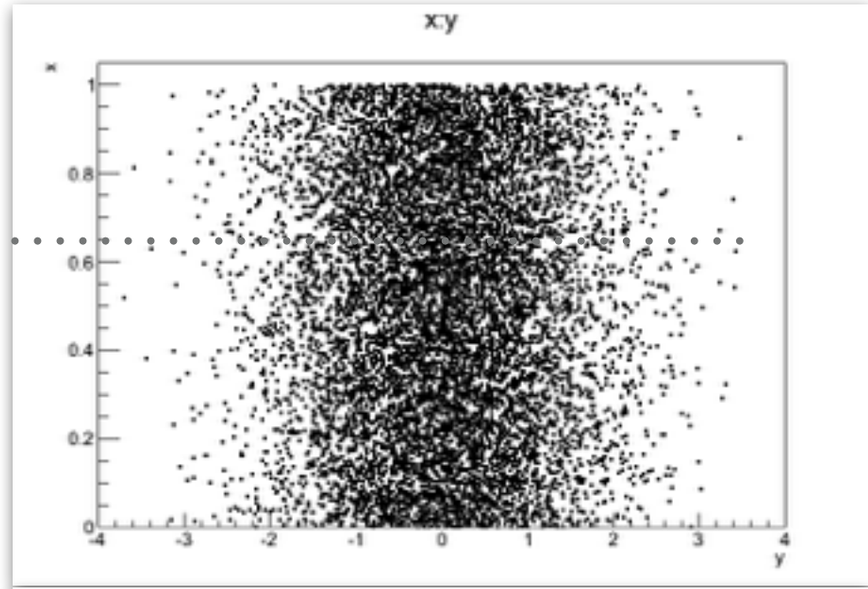


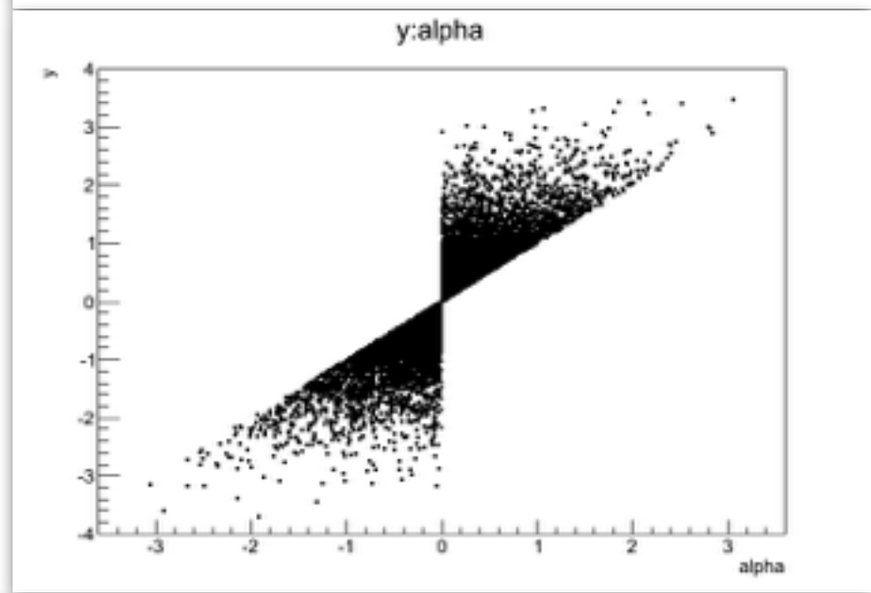


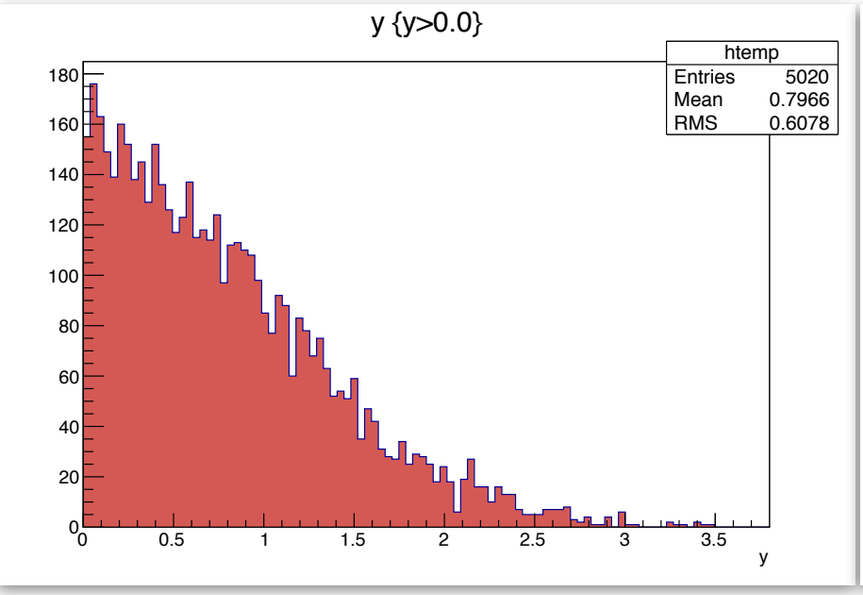
USING N-TUPLES UNDER ROOT (IV)

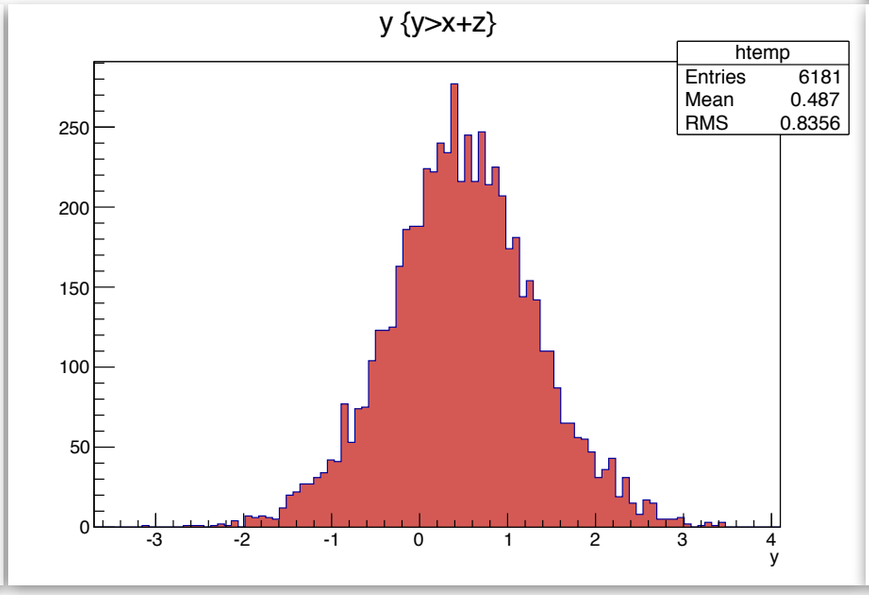
- Let's try it!

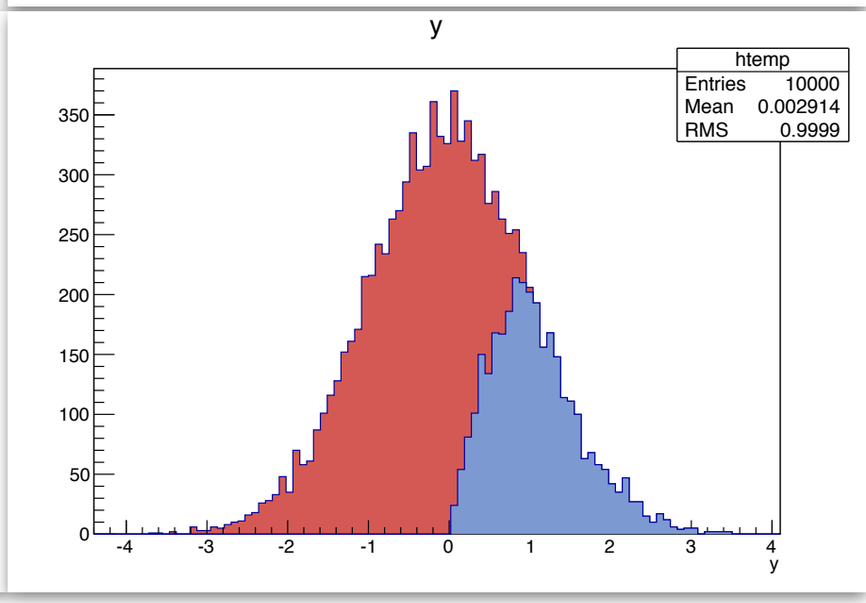
```
root [8] n1->Draw("y","y>0.0")
root [9] n1->Draw("y","y>x+z")
root [10] n1->Draw("y")
root [11] n1->SetFillColor(38);
root [12] n1->Draw("y","y>x","same")
root [13] n1->SetMarkerStyle(7);
root [14] n1->Draw("x:y")
root [15] n1->Draw("y:alpha")
```











SAVE TO A FILE

- Basic ROOT I/O: How could I save the output to a file?

example_04a.cc

```
{
    TFile *f1 = new TFile("n1.root","recreate");
    TNtuple *n1 = new TNtuple("n1","a n-tuple","x:y:z:alpha:beta:gamma");
    TRandom3 rnd;

    float var[6];
    for(int i=0;i<10000;i++) {
        var[0] = rnd.Uniform(0.,1);
        var[1] = rnd.Gaus(0.,1.);
        var[2] = rnd.Exp(-1.);
        var[3] = var[0]*var[1];
        var[4] = var[0]*var[2];
        var[5] = var[0]*var[1]*var[2];

        n1->Fill(var);
    }

    n1->Write();
    f1->Close();
}
```

↶ open a new ROOT file

↶ Save and close the file.

MISSING NTUPLE?

- Where is the n-tuple, exactly?

```
root [0] .x example-04a.cc
root [1] n1->Draw("x")
 *** Break *** segmentation violation
==========================================================
There was a crash.
This is the entire stack trace of all threads:
==========================================================
#0  0x00007f6de7e73b0e in waitpid () from /lib/libc.so.6
#1  0x00007f6de7e111f9 in ?? () from /lib/libc.so.6
```

Due to some historical reason, the TFile "owns" the histograms, n-tuples, tress, and any other things.
(Once the file close, the stored objects are out.)

- Simply read it back:

```
TFile* f2 = new TFile("n1.root");
TNutple *n1 = (TNtuple*)f2->Get("n1");
```

MORE ON TFILE, THE ROOT I/O

- Get (*recreate*) an empty new file:

```
TFile* f1 = new TFile("file.root", "recreate");
```

- Store a ROOT object:

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

- Create a directory:

```
f1->mkdir("dir");
```

- Enter the directory:

```
f1->cd("dir");
```

- See how good the compression works:

```
root [] f1->GetCompressionFactor()
(Float_t)1.82390785217285156e+00
```

MORE ON TFILE, THE ROOT I/O (II)

- Read the file back:

```
TFile* f1 = new TFile("n1.root");
```

- Get the stored object back:

```
TNtuple *n1;
n1 = (TNtuple*)f1->Get(“n1”);
n1->Draw("x");
```

- What are the stuffs in the file?

```
root [] f1->ls()
TFile**     n1.root
 TFile*     n1.root
  OBJ: TNtuple   n1 a n-tuple : 0 at: 0x1814370
  KEY: TNtuple   n1;1  a n-tuple
```

MORE ON TFILE, THE ROOT I/O (III)

- Actually ROOT can even read a root file from the web:

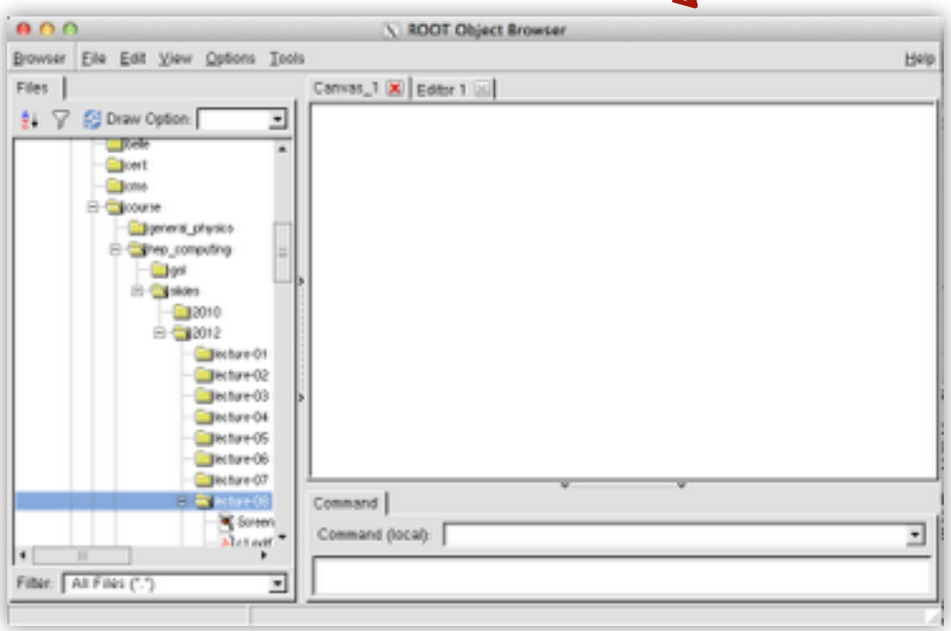
```
TFile::Open("http://cern.ch/file.root")
```

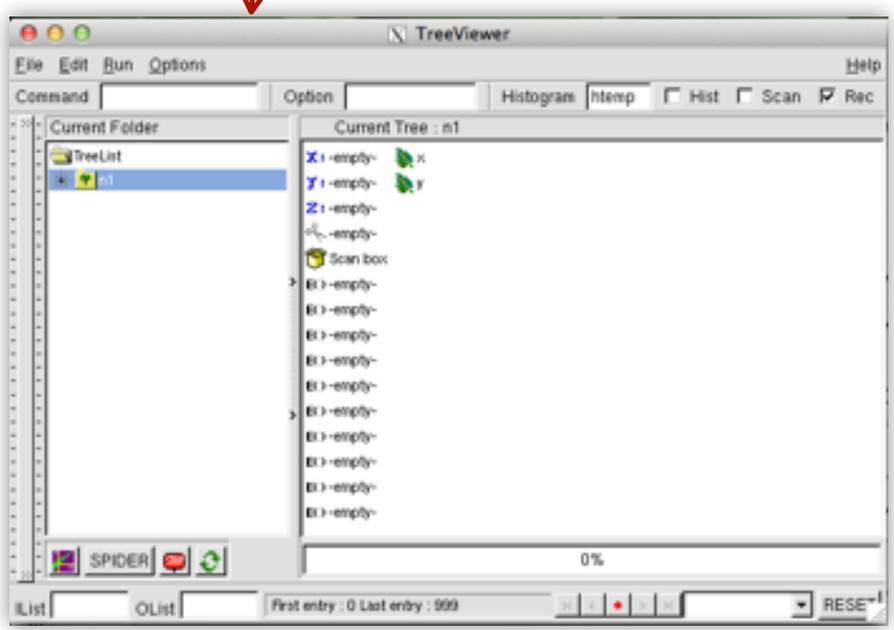
- Viewers (*if you already have a n-tuple/tree, what are the contents?*)

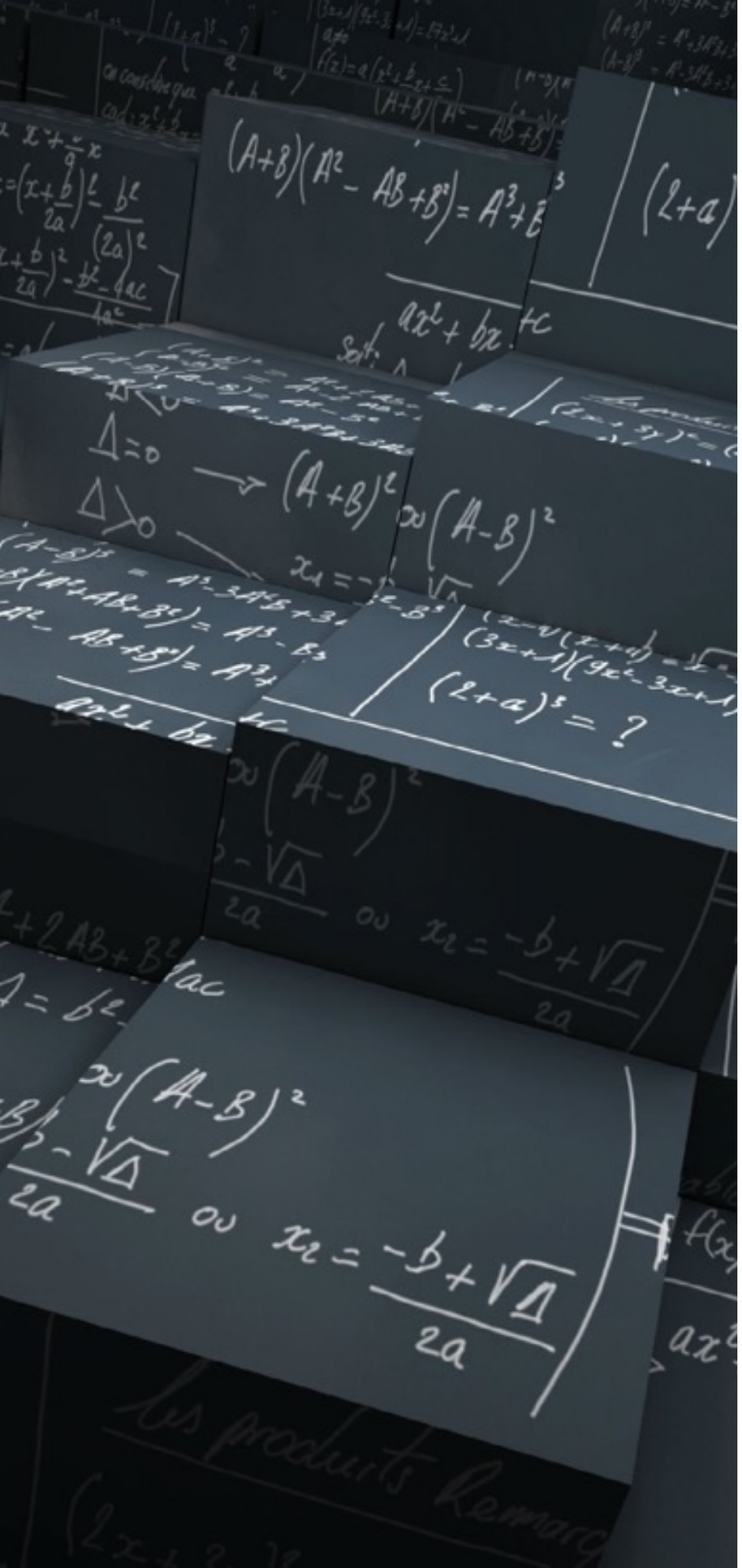
```
n1->StartViewer();
```

- Browser –– a full GUI file browser:

```
TBrowser b;
```







INTERMISSION

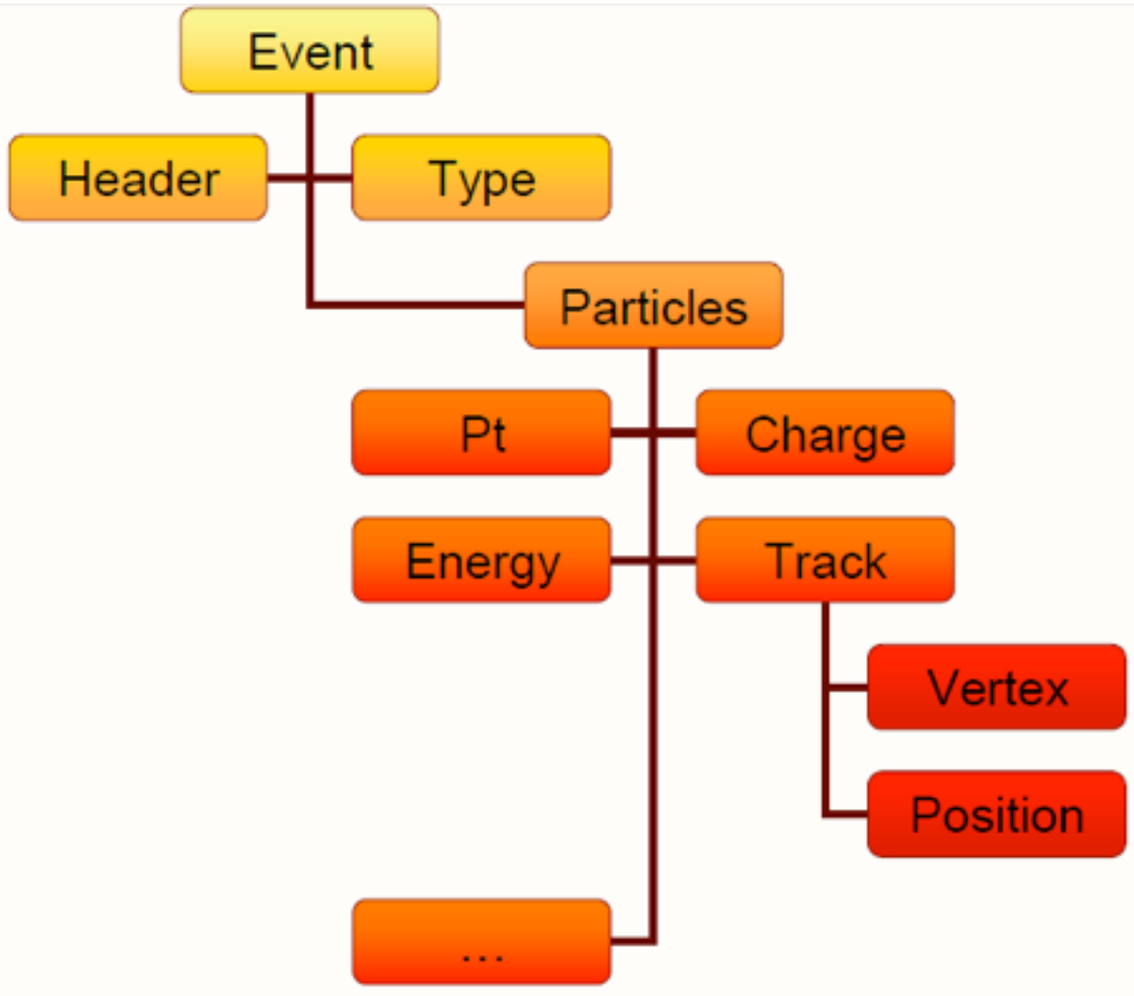
- Please try to play with this example ntuple, and see what you can actually do!
- You can also try few more drawing options!
- TFile can store any ROOT object (not just ntuples!), so you can try to store a histogram or any other things you have already practiced before.
- Next we will learn how to **plant a tree**!

NTUPLE VERSUS TREE

N-tuple:
Simple data types
(e.g. Excel tables)

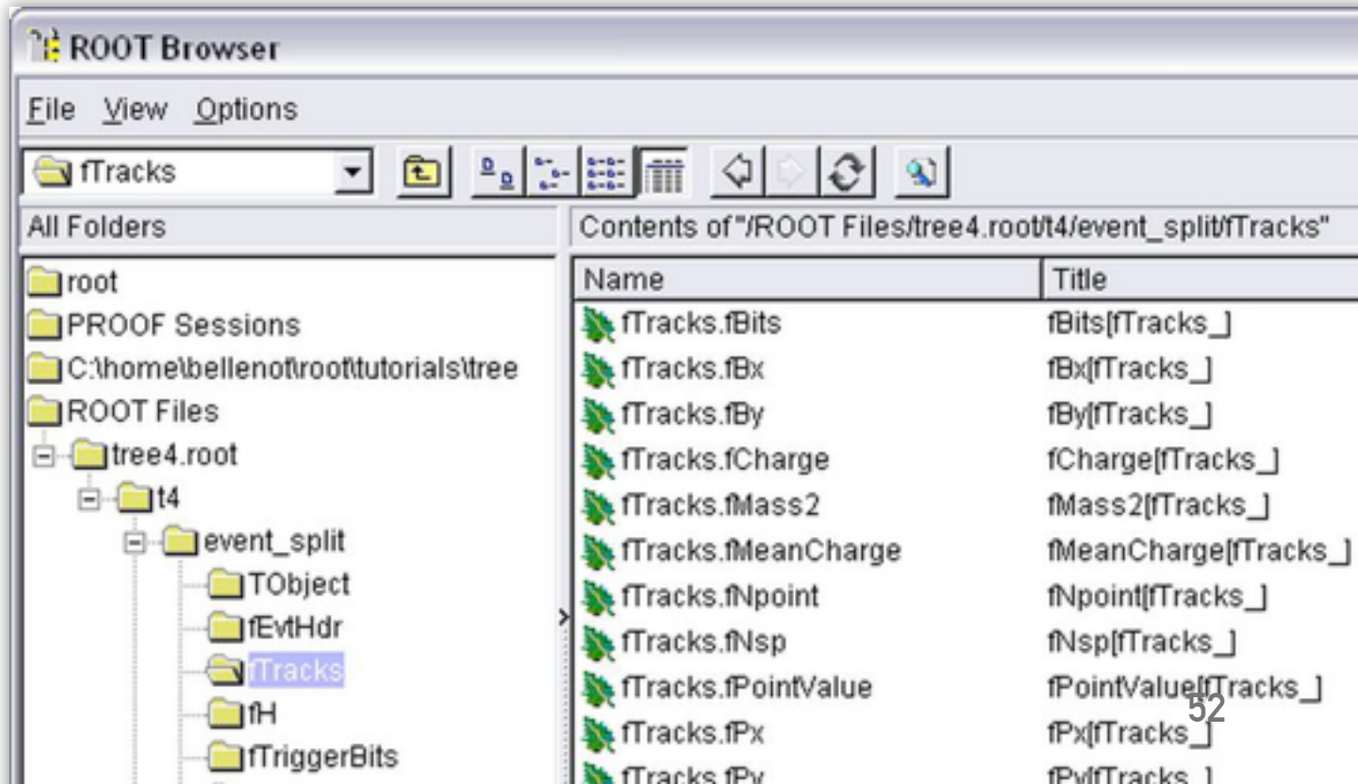
x	y	z
-1.10228	-1.79939	4.452822
1.867178	-0.59662	3.842313
-0.52418	1.868521	3.766139
-0.38061	0.969128	1.084074
0.552454	-0.21231	0.350281
-0.18495	1.187305	1.443902
0.205643	-0.77015	0.635417
1.079222	-0.32739	1.271904
-0.27492	-1.72143	3.038899
2.047779	-0.06268	4.197329
-0.45868	-1.44322	2.293266
0.304731	-0.88464	0.875442
-0.71234	-0.22239	0.556881
-0.27187	1.181767	1.470484
0.886202	-0.65411	1.213209
-2.03555	0.527648	4.421883
-1.45905	-0.464	2.344113
1.230661	-0.00565	1.514559
		3.562347

Tree:
Complex data types
(e.g. Database tables)



ROOT TREES

- Databases have row wise access
  - Can only access the full object (*e.g. a full event class!*).
- ROOT trees have column wise access
  - Direct access to any event, any branch or any leaf even in the case of variable length structures.
  - Designed to access only a subset of the object attributes (e.g. only particles' energy).
- A tree can be very complicated:

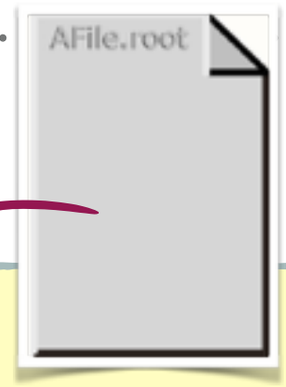


STEPS TO PLANT A TREE

- Prepare a storage area: a **TFile**
- Create a **TTree** class as the starting point
- Add **TBranch** to the Tree, build up the structure.
- Fill it!
- Write to the file. Done.

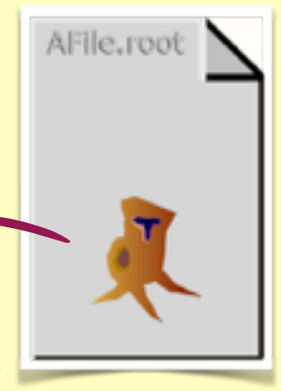
STEPS TO PLANT A TREE (II)

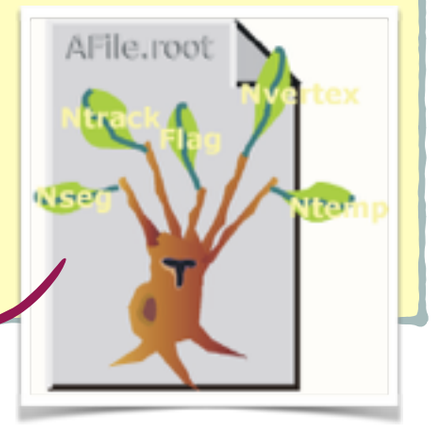
- Let's create a ***plain tree***:



example_05.cc

```
{
    TFile *myfile = new TFile("myfile.root","recreate");

    int eventno;
    int version;
    int npoints;
    double px[100],py[100],pz[100];
    bool isgood[100];

    TTree *mytree = new TTree("mytree","this is a plain tree");
    mytree->Branch("eventno",&eventno,"eventno/I");
    mytree->Branch("version",&version,"version/I");
    mytree->Branch("npoints",&npoints,"npoints/I");
    mytree->Branch("px",px,"px[npoints]/D");
    mytree->Branch("py",py,"py[npoints]/D");
    mytree->Branch("pz",pz,"pz[npoints]/D");
    mytree->Branch("isgood",isgood,"isgood[npoints]/O");
```

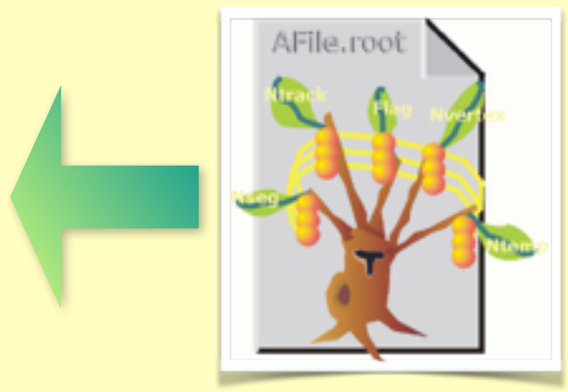


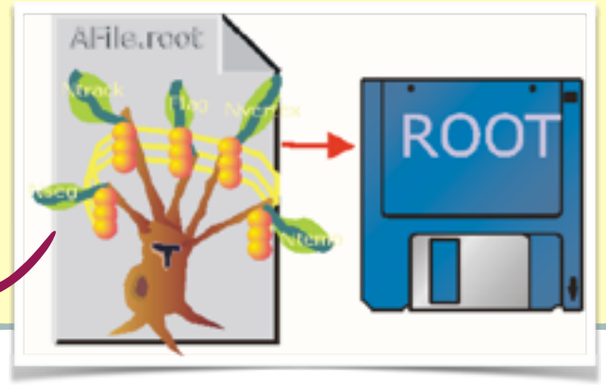


STEPS TO PLANT A TREE (III)

example_05.cc

```
TRandom3 rnd;

for(int evt=0;evt<1000;evt++) {
    eventno = evt;
    version = 1234;
    npoints = (int)rnd.Uniform(1.,100.);
    for(int i=0;i<npoints;i++) {
        px[i] = rnd.Gaus(0.,1.);
        py[i] = rnd.Gaus(0.,1.);
        pz[i] = rnd.Gaus(0.,1.);

        double R = sqrt(px[i]*px[i] + py[i]*py[i] + pz[i]*pz[i]);
        if (R<1.) isgood[i] = true;
        else      isgood[i] = false;
    }

    mytree->Fill();
}

mytree->Write();
myfile->Close();
}
```





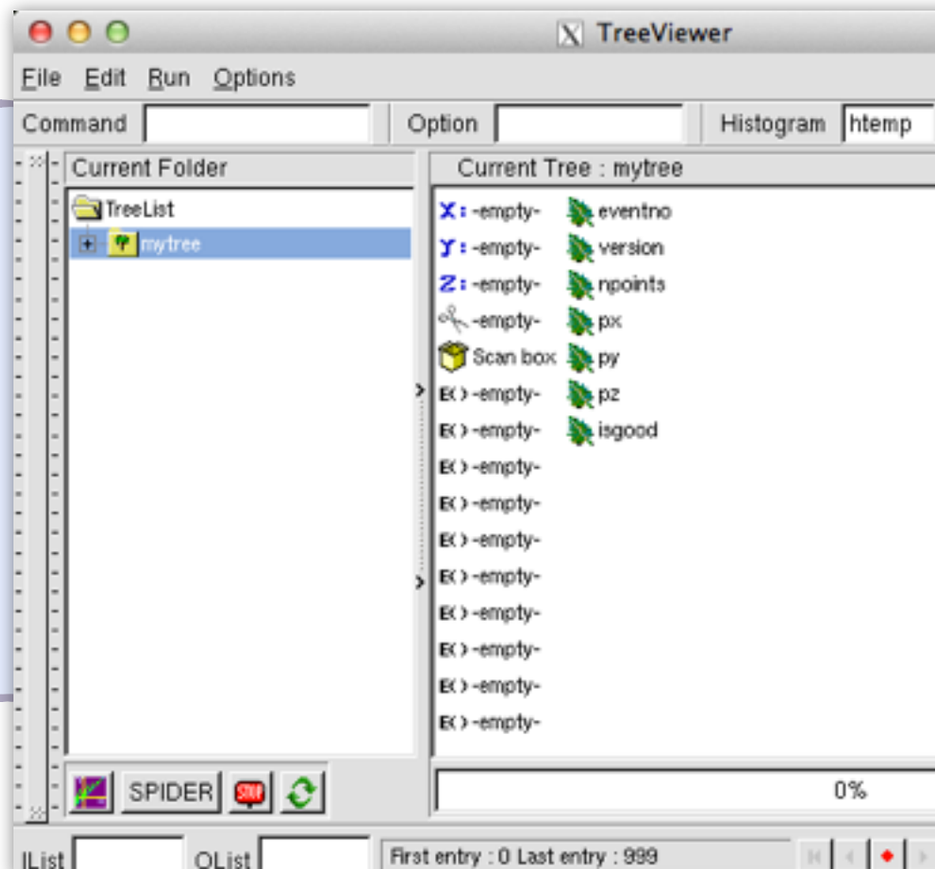
AS A TREE PLANTED!

- We got a file on the disk:

```
> root -l example-05.cc
root [0]
Processing example-05.cc...
root [1] .q
> ls -lh myfile.root
-rw-r--r--  1 kfjack  staff   1.1M 11 15 16:15 myfile.root
>
```

- Simply read it back and check:

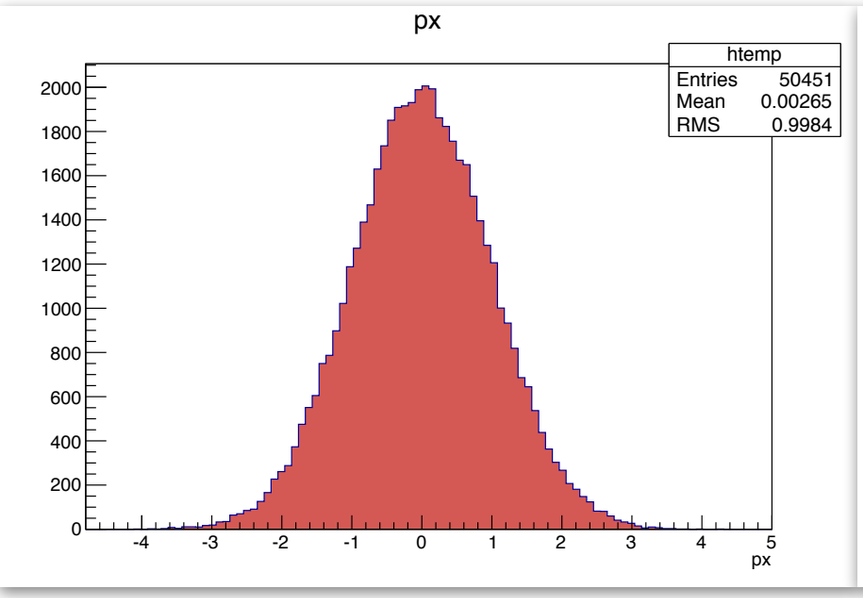
```
root -l myfile.root
root [0]
Attaching file myfile.root as _file0...
root [1] _file0->ls()
TFile**     myfile.root
 TFile*     myfile.root
  KEY: TTree  mytree;1  this is a plain tree
Root [2] mytree->StartViewer()
```

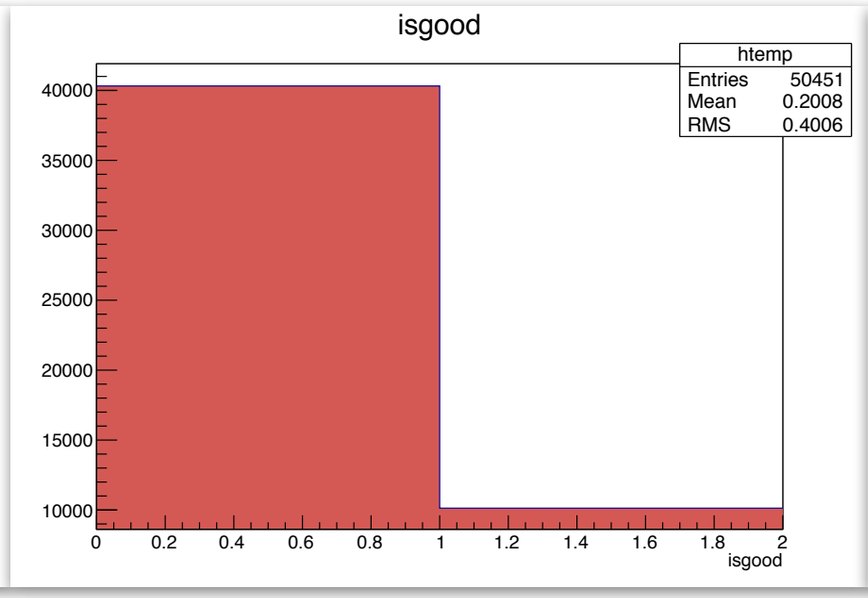


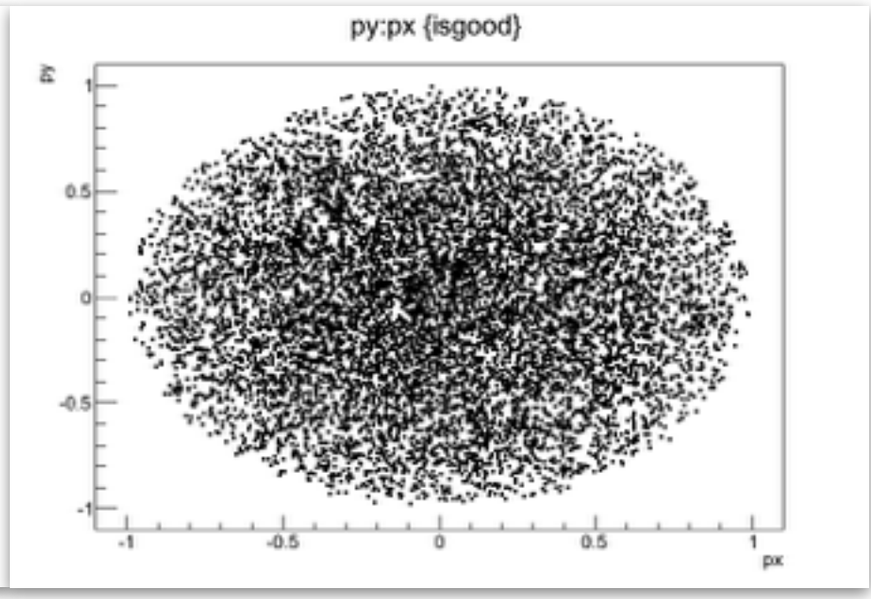
SOME HANDS ON DRAWING

- We can already draw some plots quickly:

```
root [3] mytree->SetFillColor(50);
root [4] mytree->Draw("px")
root [5] mytree->Draw("isgood")
root [6] mytree->SetMarkerStyle(7);
root [6] mytree->Draw("py:px","isgood")
```







SOME INTERACTIVE CHECKS

➤ Just show me an event:

```
root [13] mytree->Show(123)
======> EVENT:123
 eventno          = 123
 version          = 1234
 npoints          = 83
 px               = 0.583106,
                    -0.185977, 0.177258, 0.804539, 0.55251, -1.14824,
                    1.5605, -0.54756, 0.408414, 0.748365, -1.11934,
                    -1.20243, 0.418382, -0.25497, -1.10096, 0.368731,
                    1.19286, 0.0375407, 1.09334, -0.370777
 py               = -1.72733,
                    -0.675075, 1.09385, 1.94544, 1.12607, 0.231102,
                    0.369161, -1.31054, -0.111109, 0.0661841, 0.817135,
                    2.07414, -0.216045, -0.452616, -1.77727, 0.0593586,
                    1.6526, 1.58467, -1.22544, -0.169393
 pz               = 0.251957,
                    -0.274804, 1.04122, 0.374447, -0.964622, 0.268402,
                    -0.420097, 1.51291, 0.712491, 1.02021, 1.07815,
                    -0.61005, 0.215732, 1.53295, 0.102865, 1.30872,
                    -1.09856, -0.708144, 0.956858, 1.93922
 isgood           = 0,
                    1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                    0, 1, 0, 0, 0, 0, 0, 0, 0
```

SOME INTERACTIVE CHECKS (II)

- How many events do we have?

```
root [] mytree->GetEntries()
(const Long64_t)1000
```

- Print out the tree structure:

```
root [] mytree->Print()
*****************************************************************************
*Tree    :mytree    : this is a plain tree                                    *
*Entries :     1000 : Total =         1435603 bytes  File  Size =    1115885 *
*        :          : Tree compression factor =   1.16                       *
*****************************************************************************
*Br    0 :eventno   : eventno/I                                               *
*Entries :     1000 : Total  Size=       8726 bytes  File Size  =       1519 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.68     *
*.............................................................................*
*Br    1 :version   : version/I                                               *
*Entries :     1000 : Total  Size=       8726 bytes  File Size  =        134 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=  30.42     *
*.............................................................................*
```

SOME INTERACTIVE CHECKS (III)

- Look **(scan)** over the data:

```
root [] mytree->Scan("eventno:npoints:px[0]:isgood[0]")
************************************************************
*    Row   *   eventno *   npoints *     px[0] * isgood[0] *
************************************************************
*        0 *         0 *        99 * -0.434764 *         1 *
*        1 *         1 *        45 * -1.246779 *         0 *
*        2 *         2 *        19 * 0.2128659 *         0 *
*        3 *         3 *        95 * -1.464682 *         0 *
*        4 *         4 *         7 * -0.621973 *         0 *
*        5 *         5 *        88 * -0.353350 *         1 *
*        6 *         6 *         3 * -0.699649 *         0 *
```

- Apply some selection cuts:

```
root [] mytree->Scan("eventno:npoints:px[0]:isgood[0]","isgood[0]")
************************************************************
*    Row   *   eventno *   npoints *     px[0] * isgood[0] *
************************************************************
*        0 *         0 *        99 * -0.434764 *         1 *
*        5 *         5 *        88 * -0.353350 *         1 *
*        7 *         7 *        30 * -0.678589 *         1 *
*       11 *        11 *         7 * 0.2192087 *         1 *
```

DATA ACCESS IN THE CODE

- Interactive access is easy, but we cannot do complicated analysis with it.
- You can imagine that you have several millions of events, the selection criteria are also very complicated, it is definitely not possible to process the events with interactive commands only.
- As an example: can you count how many events with **"# of good points > 10"**?
  - Basically we have to load each event.
  - Setup a counter, increase the number if # of good points > 10.
  - Report the counts at the end.
- Well, one can still look over the data, one-event-by-one-event, surely this will kill lots of your time!

See the next page for a quick example!

example_05.cc

```
{
    TFile *myfile = new TFile("myfile.root");

    int eventno;
    int version;
    int npoints;
    double px[100],py[100],pz[100];
    bool isgood[100];
```

Allocate the spaces for the branches

```
    TTree *mytree;
    myfile->GetObject("mytree",mytree);
```

← Get the tree object

```
    mytree->SetBranchAddress("eventno",&eventno);
    mytree->SetBranchAddress("version",&version);
    mytree->SetBranchAddress("npoints",&npoints);
    mytree->SetBranchAddress("px",px);
    mytree->SetBranchAddress("py",py);
    mytree->SetBranchAddress("pz",pz);
    mytree->SetBranchAddress("isgood",isgood);
```

connect the spaces with branches

```
    int count = 0;
    for(int evt=0;evt<mytree->GetEntries();evt++) {
        mytree->GetEntry(evt);

        int count_pts = 0;
        for(int i=0;i<npoints;i++)
            if (isgood[i]) count_pts++;
        if (count_pts>10) count++;
    }
```

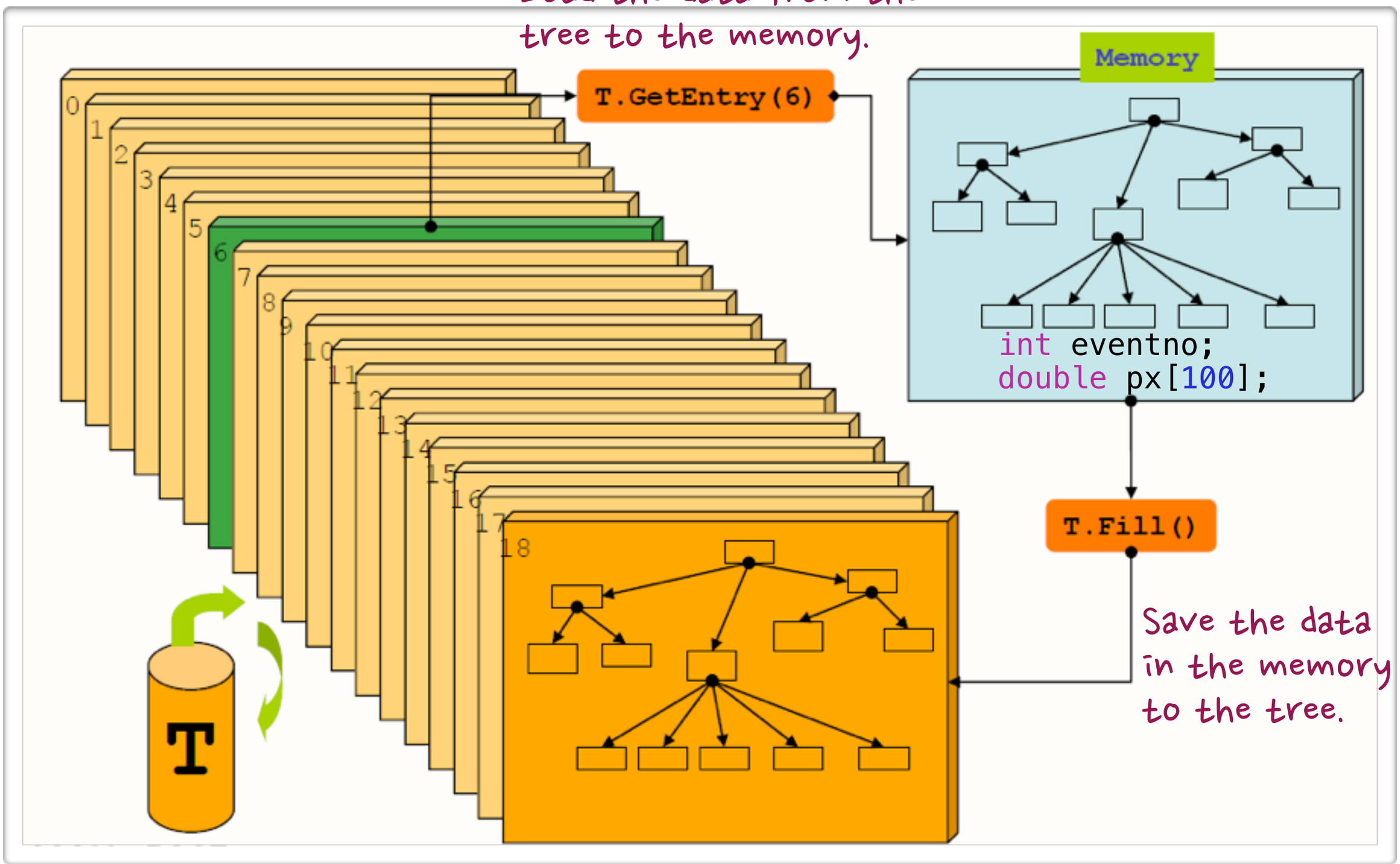
Do our little analysis work!

```
    printf("count = %d\n",count);
}
```

Terminal output

```
root [0] .x example_04.cc
count = 438
```

TREE DATA ACCESS: WRITE AND READ




```
TFile("myfile.root")
```

ACCESS TO MULTIPLE FILES

- What should we do if my data is huge, and I have to split my results into several ROOT files?
- **TChain** is (one of) the straightforward solution(s):
  - List of ROOT files containing the same tree.
  - Same semantics as TTree
  - As an example, assume we have three files called file1.root, file2.root, file3.root. Each contains tree called "mytree". Create a chain:

```
root [0] TChain *chn = new TChain("mytree");
root [1] chn->Add("file1.root");
root [2] chn->Add("file2.root");
root [3] chn->Add("file3.root");
```

OR

```
root [1] chn->Add("file*.root");
```

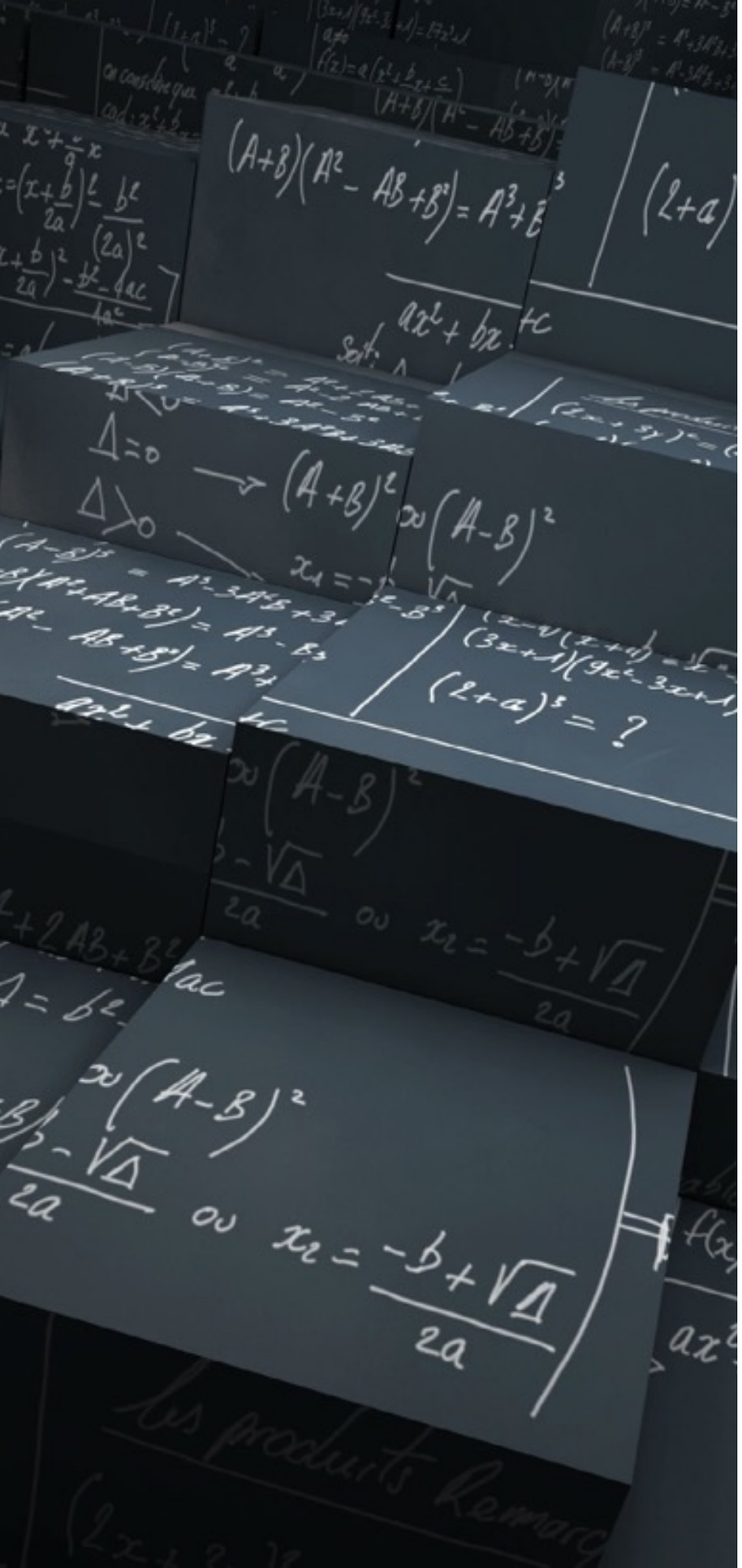
Then this "chn" can be used just like a tree!

COMMENTS: DATA ACCESS IN THE CODE

- In fact this is the most common way to analysis data. With a proper code you can access the information stored in your root trees in an efficient way.
- There are several other way to read and analyze the root data (including automatically code generation, structured processing, using multi CPU cores, etc.) You may need to check few more examples or tutorials.
- Nevertheless the contents discussed in this lecture should give you the minimal hands-on control of the ROOT framework. This should be more-or-less enough for you to handle the materials given in the lectures afterwards.
- Note we will discuss some more advanced usage of ROOT (for example, RooFit) in a later lecture.

OTHER USEFUL ROOT OBJECTS

- Here we list many useful tools that are commonly used in many places, for you to explore a little bit more!
  - **TGraph**: a 2D graphical object based on X-Y arrays.
  - **TCanvas**: the graphical area
  - **TLegend**: if you want to add some legend to your plot!
  - **TLatex**: if you want to add some latex labels!
  - **TMath**: a collection of many useful mathematical tools
  - **TRandom3**: random number generator wrapper class (algorithm: Mersenne Twister)
  - **TMinuit**: the Minuit wrapper, useful to derive your own minuit-based program.
  - **TSelector**: for structured data processing module.
  - **TMatrixD/TMatrixF**: linear algebra tools in double or float
  - … (*and many many other stuff!*)



SUMMARY

- As the “first time with ROOT” –– we have covered the most common ROOT commands, create object, handling histograms and functions, unbinned data format.
- If you are familiar with ROOT already, this is generally too easy for you. If not, you may want to practice the example codes given in this lecture, as well as some other examples you can find on the ROOT webpage.
- Next we will have our first exercise hour.