Introduction to ROOT

Jan Fiete Grosse-Oetringhaus, CERN PH/ALICE

Summer Student Lectures 2012

July 9th

Content

- Introduction to the ROOT framework
  - Library structure
  - CINT
  - Macros
  - Histograms, Graphs, Advanced graphics examples
  - Input/Output: Files, Trees
  - Fitting
- Nomenclature
  - **Blue: you type it**
  - **Red: you get it**
- You can find a ROOT demo (slides + recorded lecture) at https://indico.cern.ch/conferenceDisplay.py?confId=134330

ROOT in a Nutshell

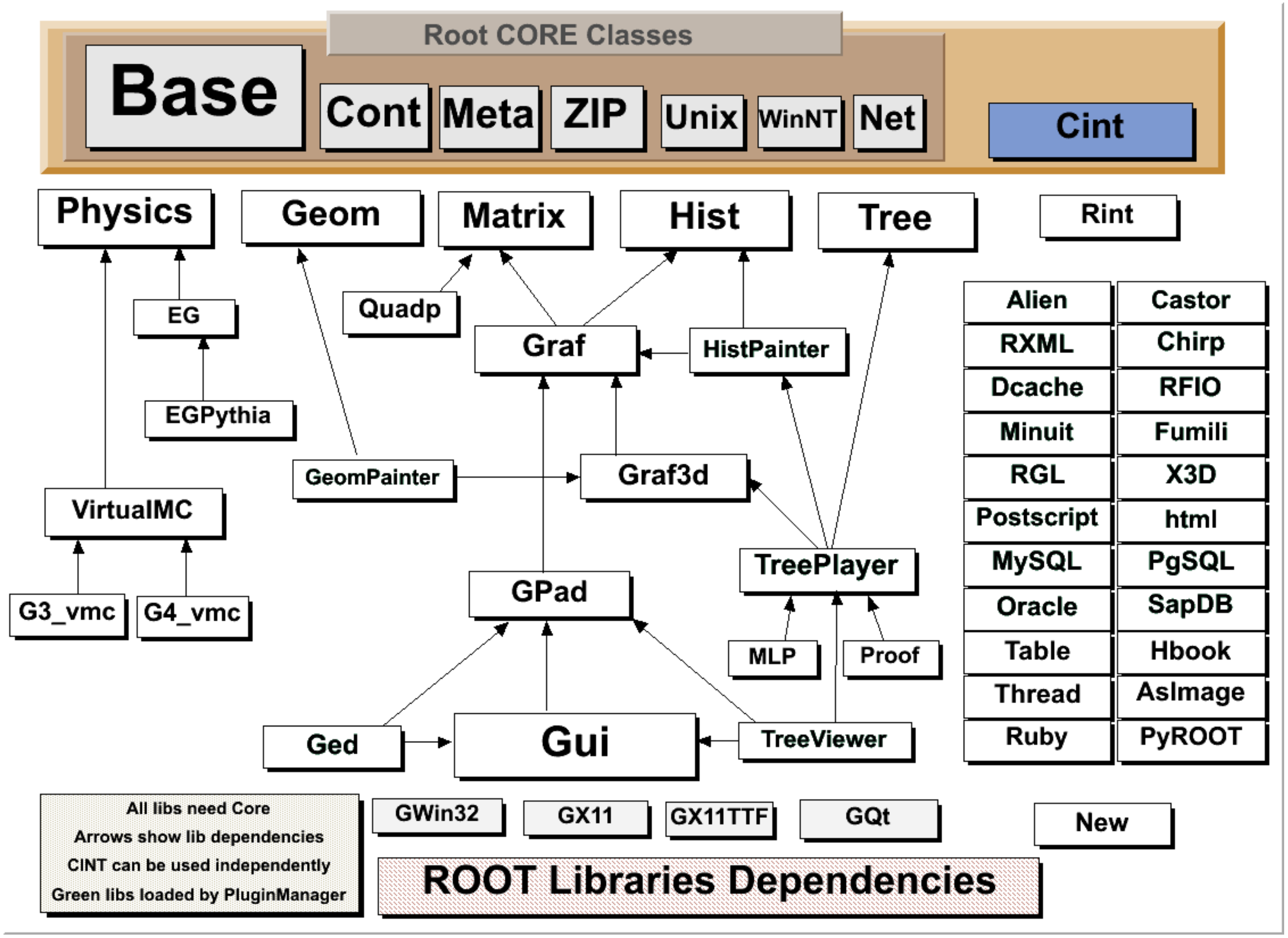
- ROOT is a large Object-Oriented data handling and analysis framework
  - Efficient object store scaling from KB's to PB's
- C++ interpreter
- Extensive 2D+3D scientific data visualization capabilities
- Extensive set of multi-dimensional histograming, data fitting, modeling and analysis methods
- Complete set of GUI widgets
- Classes for threading, shared memory, networking, etc.
- Parallel version of analysis engine runs on clusters and multi-core
- Fully cross platform: Unix/Linux, MacOS X and Windows

ROOT in a Nutshell (2)

- The user interacts with ROOT via a graphical user interface, the command line or scripts
- The command and scripting language is C++
  - Embedded CINT C++ interpreter
  - Large scripts can be compiled and dynamically loaded

And for you?
ROOT is usually the interface (and sometimes the barrier) between you and the data

The ROOT Libraries



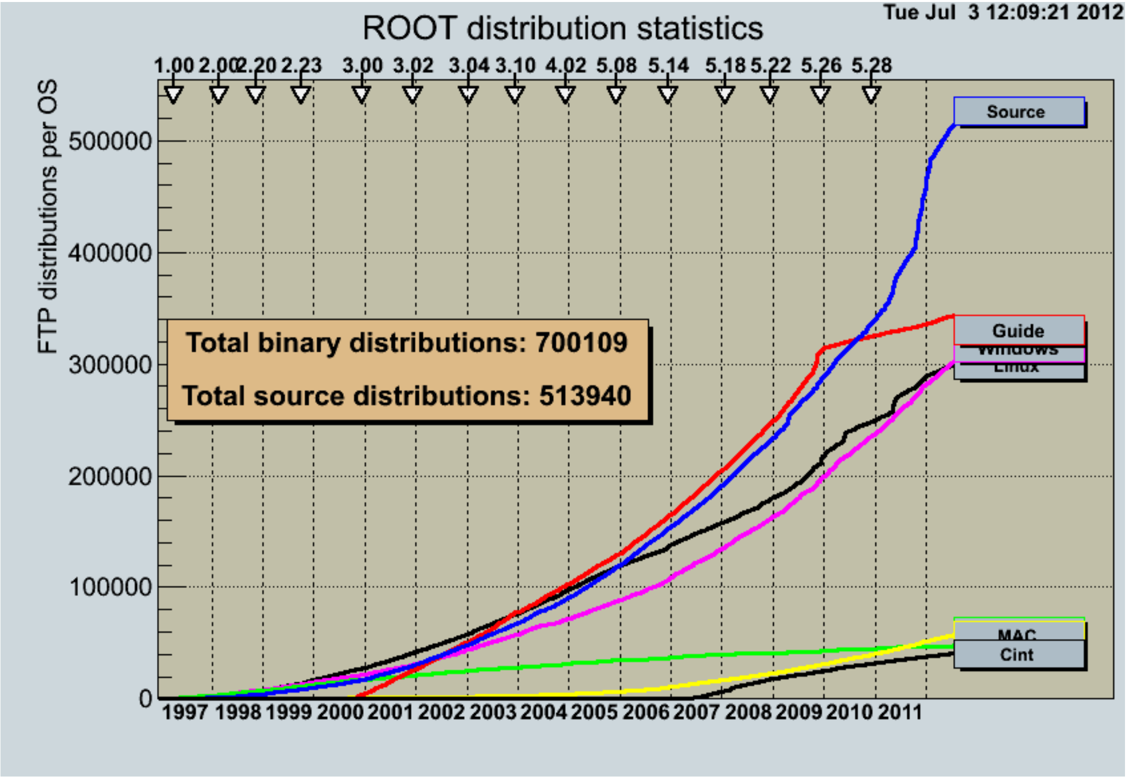
- Over 2,500 classes
- 3,000,000 lines of code
- CORE (8 Mbytes)
- CINT (2 Mbytes)
- Most libraries linked on demand via plug-in manager (only a subset shown)
- 100 shared libs

ROOT: An Open Source Project

- The project was started in Jan 1995
- First release Nov 1995
- The project is developed as a collaboration between:
  - Full time developers:
    - 7 people full time at CERN (PH/SFT)
    - 2 developers at Fermilab/USA
  - Large number of part-time contributors (160 in CREDITS file)
  - A long list of users giving feedback, comments, bug fixes and many small contributions
    - 5,500 users registered to RootTalk forum
    - 10,000 posts per year
- An Open Source Project, source available under the LGPL license
- Used by all HEP experiments in the world
- Used in many other scientific fields and in commercial world

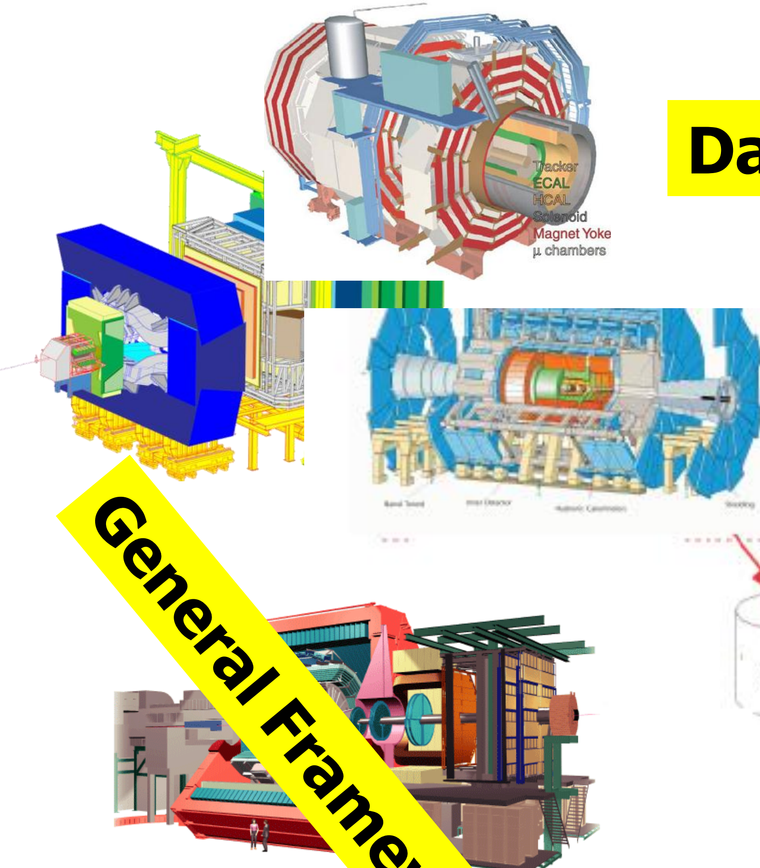
Some ROOT Statistics

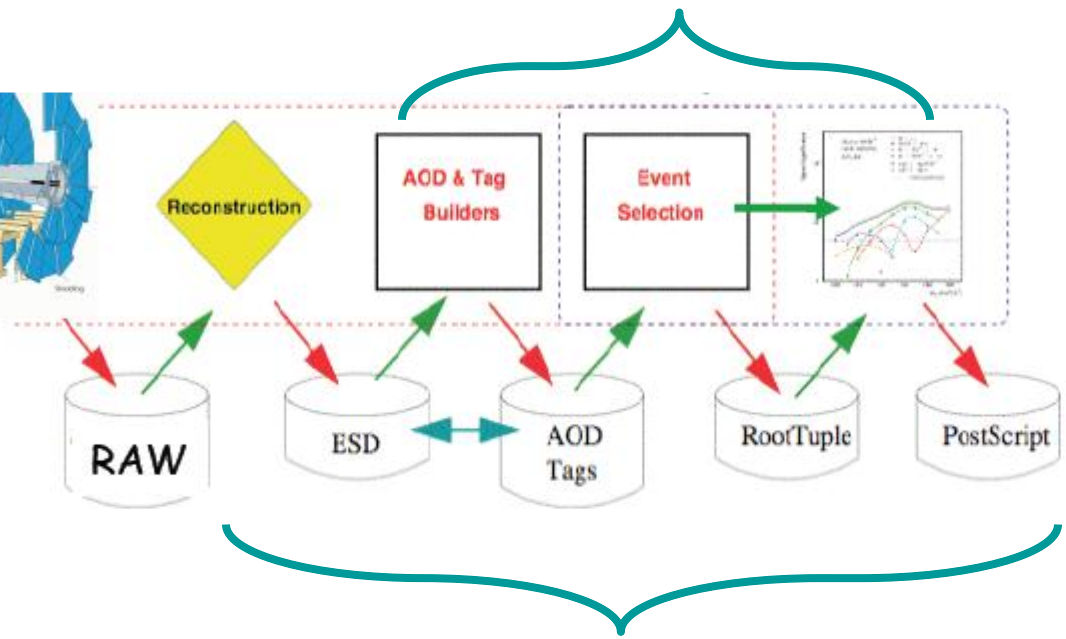
- ROOT binaries have been downloaded about 700,000 times since 1997
- The estimated user base is about 20,000 people



ROOT Application Domains

Data Analysis & Visualization

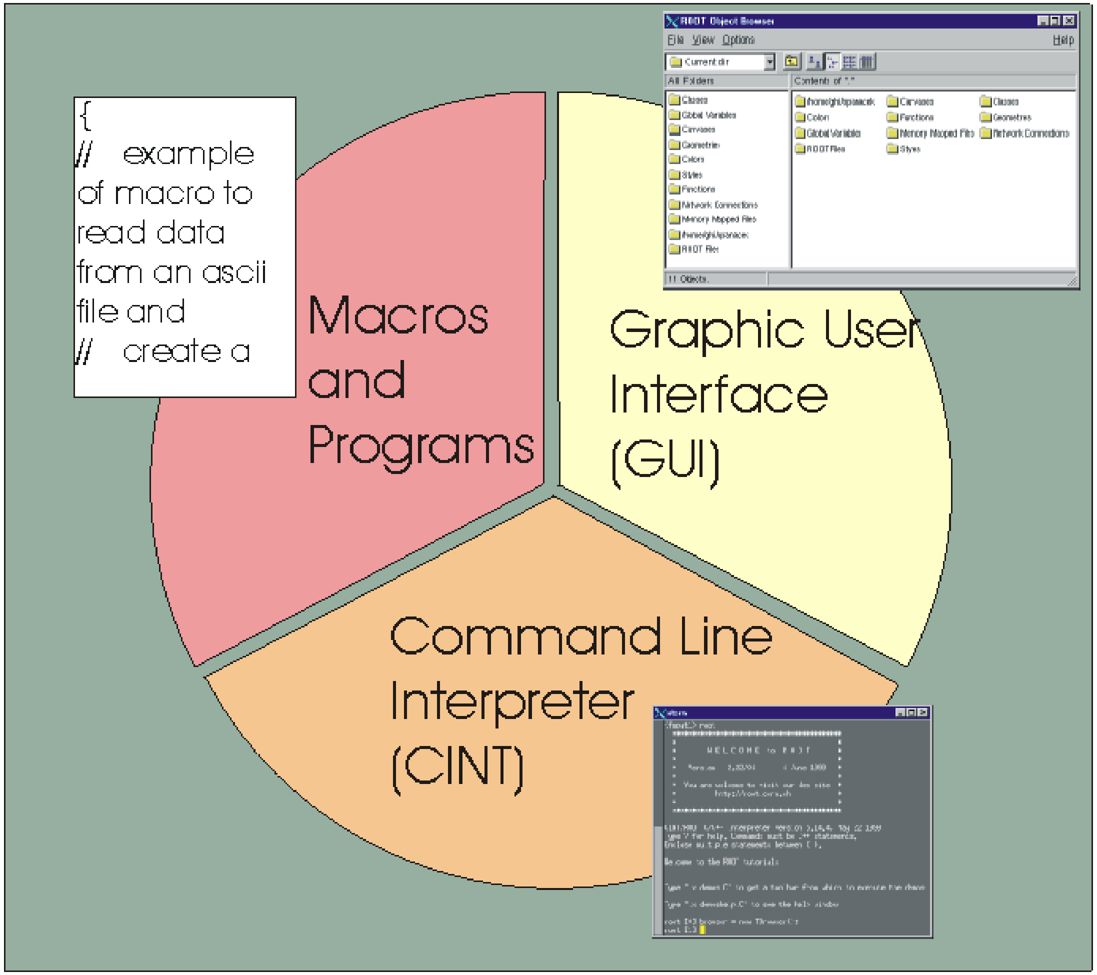




Data Storage: Local, Network

General Framework

Three User Interfaces



- GUI windows, buttons, menus
- Command line CINT (C++ interpreter)
- Macros, applications, libraries (C++ compiler and interpreter)

ROOT Download & Installation

- http://root.cern.ch
  - Binaries for common Linux PC flavors, Mac OS, Windows
  - Preinstalled on AFS (at CERN)
- Source files
  - Installation guide at http://root.cern.ch/drupal/content/installing-root-source
  - Dependencies, discussed here: http://root.cern.ch/drupal/content/build-prerequisites

Basic Blocks of ROOT

- Command line interpreter CINT
- Macros
- Histograms and Graphs
- Files
- Trees

CINT in ROOT

- CINT is used in ROOT:
  - As command line interpreter
  - As script interpreter
  - To generate class dictionaries
  - To generate function/method calling stubs
  - Signals/slots with the GUI
- The command line, script and programming language become the same
- Large scripts can be compiled for optimal performance

First CINT Example

```
$ root
root [0] 344+76.8
(const double)4.20800000000000010e+002
root [1] float x=89.7;
root [2] float y=567.8;
root [3] x+sqrt(y)
(double)1.13528550991510710e+002
root [4] float z = x+2*sqrt(y/6);
root [5] z
(float)1.09155929565429690e+002
root [6] .q
$
```

Display online help with: root [0] .h

ROOT Prompt

- Starting ROOT

  $ root

  $ root -l (without splash screen)
- Command history
  - Scan through with **arrow keys** ↑↓
  - Search with **CTRL-R** (like in bash)
- Online help

  root [ ] new TF1(<TAB>

  TF1 TF1()

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

  …
- Typing multi-line commands

  root [ ] for (i=0; i<3; i++) printf("%d\n", i)

  or

  root [ ] for (i=0; i<3; i++) {

  end with '}', '@':abort > printf("%d\n", i);

  end with '}', '@':abort > }
- Aborting wrong input

  root [ ] printf("%d\n, i)

  end with ';', '@':abort > @

Don't panic!

Don't press CTRL-C!

Just type @

Macros

- It is quite cumbersome to type the same lines again and again
- Create macros for commonly used code
- Macro = file that is interpreted by CINT

```
int mymacro(int value)
{
  int ret = 42;
  ret += value;
  return ret;
}
```

→ **saved in mymacro.C**

- Execute with **root [0] .x mymacro.C(10)**
- Or **root [0] .L mymacro.C**
  root [1] mymacro(10)

Compile Macros – Libraries

- "Library": compiled code, shared library
- CINT can call its functions!
- Building a library from a macro: ACLiC (**A**utomatic **C**ompiler of **Li**braries for **C**INT)
- Execute it with a "+" **root [0] .x mymacro.C(42)+**
- Or **root [0] .L mymacro.C+**
  root [1] mymacro(42)
- No Makefile needed
- CINT knows all functions in the library mymacro_C.so/.dll

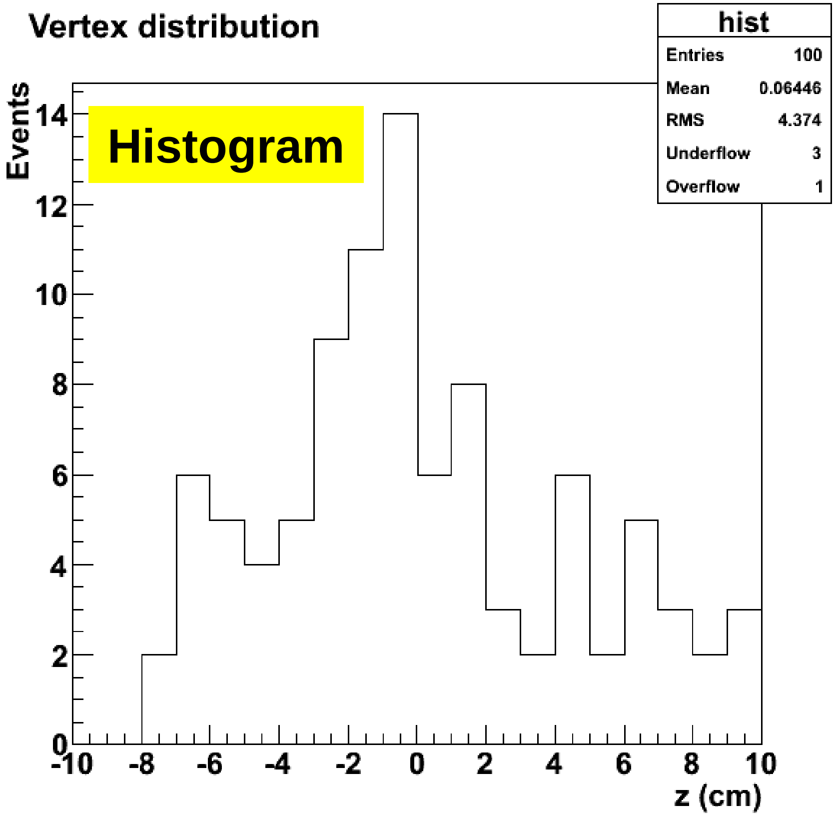
Compiled vs. Interpreted

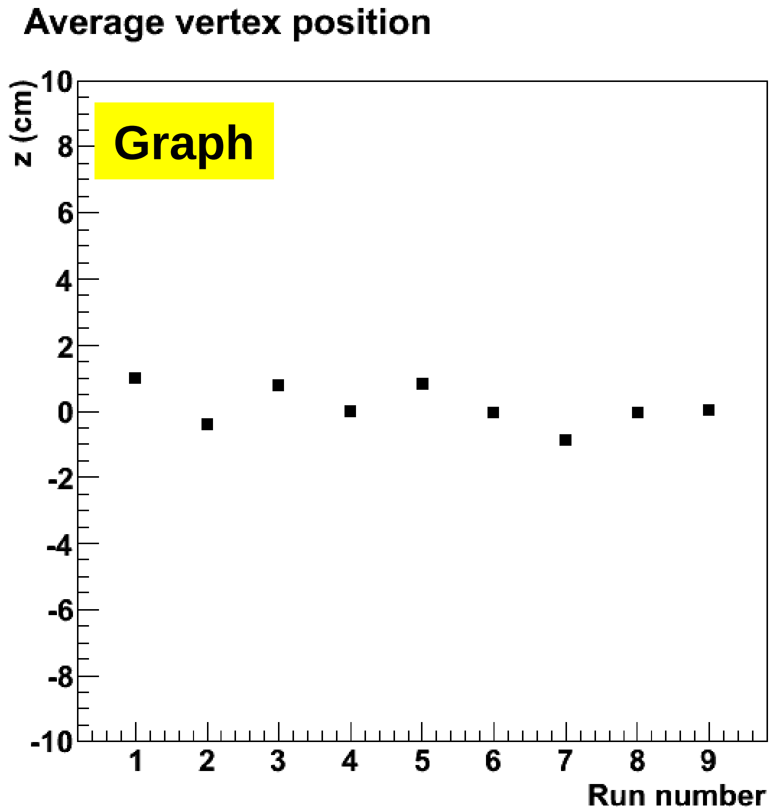
- Why compile?
  - Faster execution, CINT has some limitations…
- Why interpret?
  - Faster Edit → Run → Check result → Edit cycles ("rapid prototyping"). Scripting is sometimes just easier
- So when should I start compiling?
  - For simple things: start with macros
  - Rule of thumb
    - Is it a lot of code or running slow? → Compile it!
    - Does it behave weird? → Compile it!
    - Is there an error that you do not find → Compile it!

ROOT Types

- You can use native C types in your code (as long as you don't make your data persistent, i.e. write to files)
- ROOT redefines all types to achieve platform independency
  - E.g. the type int has a different number of bits on different systems
  - int → Int_t   float → Float_t
    double → Double_t   long → Long64_t (not Long_t)
    etc.
  - See $ROOTSYS/include/Rtypes.h

Histograms & Graphs





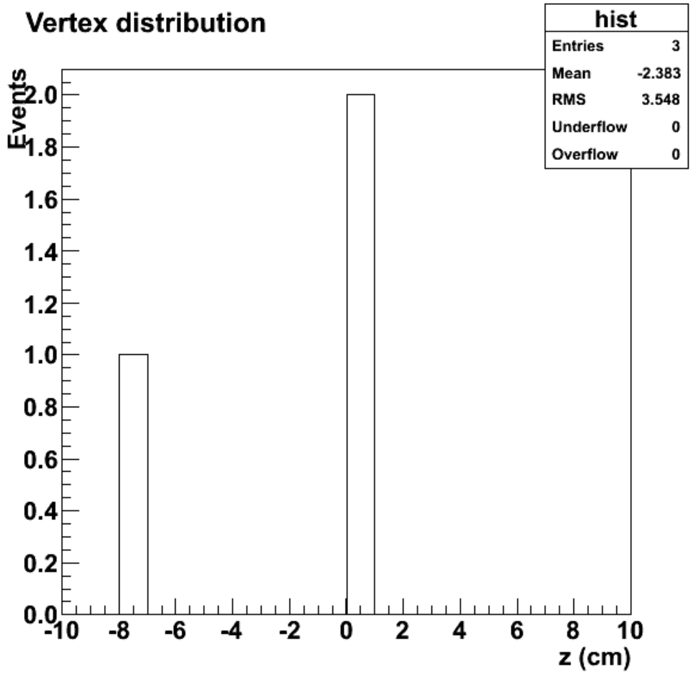
- Container for binned data
  - Most of HEP's distributions
- Container for distinct points
  - Calculation or fit results

Histograms

- Histograms are binned data containers
- There are 1, 2 and 3-dimensional histograms → TH1, TH2, TH3
- The data can be stored with different precision and in different types (byte, short, int, float, double)
  → TH1C, TH1S, TH1I, TH1F, TH1D
  (same for TH2, TH3)

<u>Histogram Example</u>

```
hist = new TH1F("hist", "Vertex
   distribution;z (cm);Events", 20, -10, 10);
hist->Fill(0.05);
hist->Fill(-7.4);
hist->Fill(0.2);
hist->Draw();
```



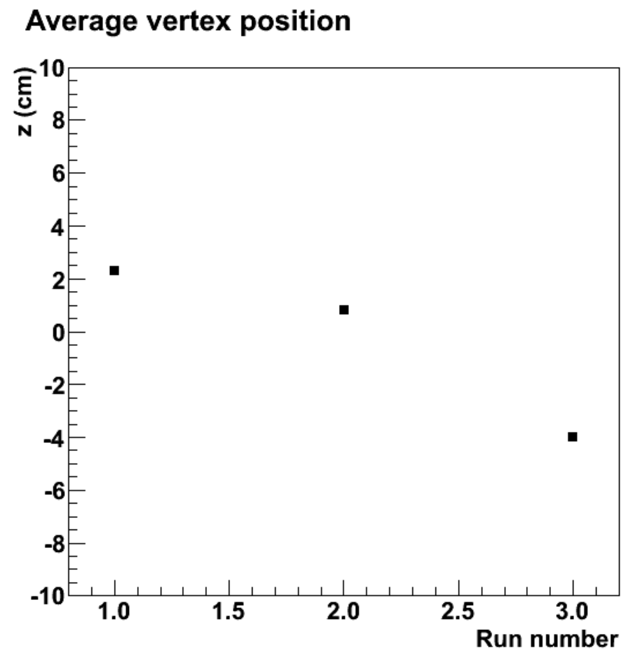
NB: All ROOT classes start with T
Looking for e.g. a string? Try TString

Graphs

- A graph is a data container filled with distinct points
- TGraph: x/y graph without error bars
- TGraphErrors: x/y graph with error bars
- TGraphAsymmErrors: x/y graph with asymmetric error bars

Graph Example

```
graph = new TGraph;
graph->SetPoint(graph->GetN(), 1, 2.3);
graph->SetPoint(graph->GetN(), 2, 0.8);
graph->SetPoint(graph->GetN(), 3, -4);
graph->Draw("AP");
graph->SetMarkerStyle(21);
graph->GetYaxis()->SetRangeUser(-10, 10);
graph->GetXaxis()->SetTitle("Run number");
graph->GetYaxis()->SetTitle("z (cm)");
graph->SetTitle("Average vertex position");
```



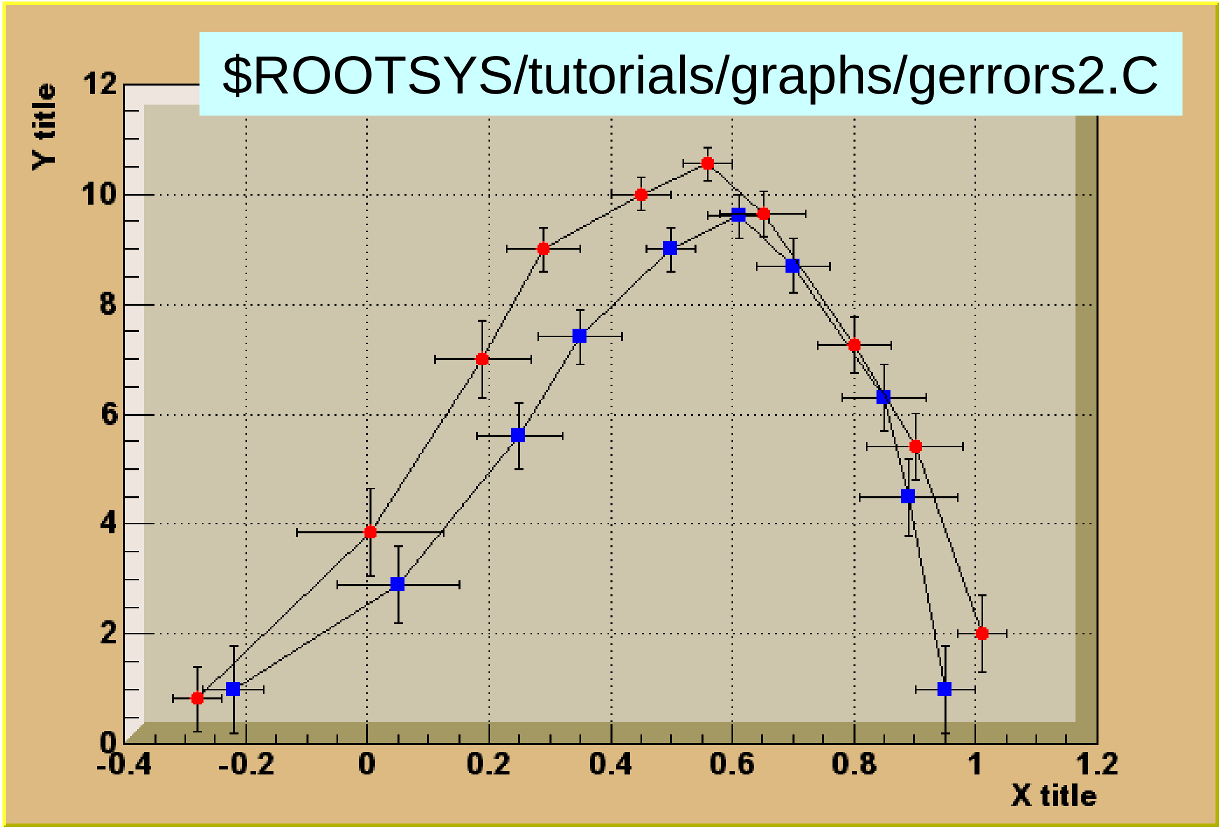
Graphs (2)

TGraphErrors(n,x,y,ex,ey)

TGraph(n,x,y)

TCutG(n,x,y)

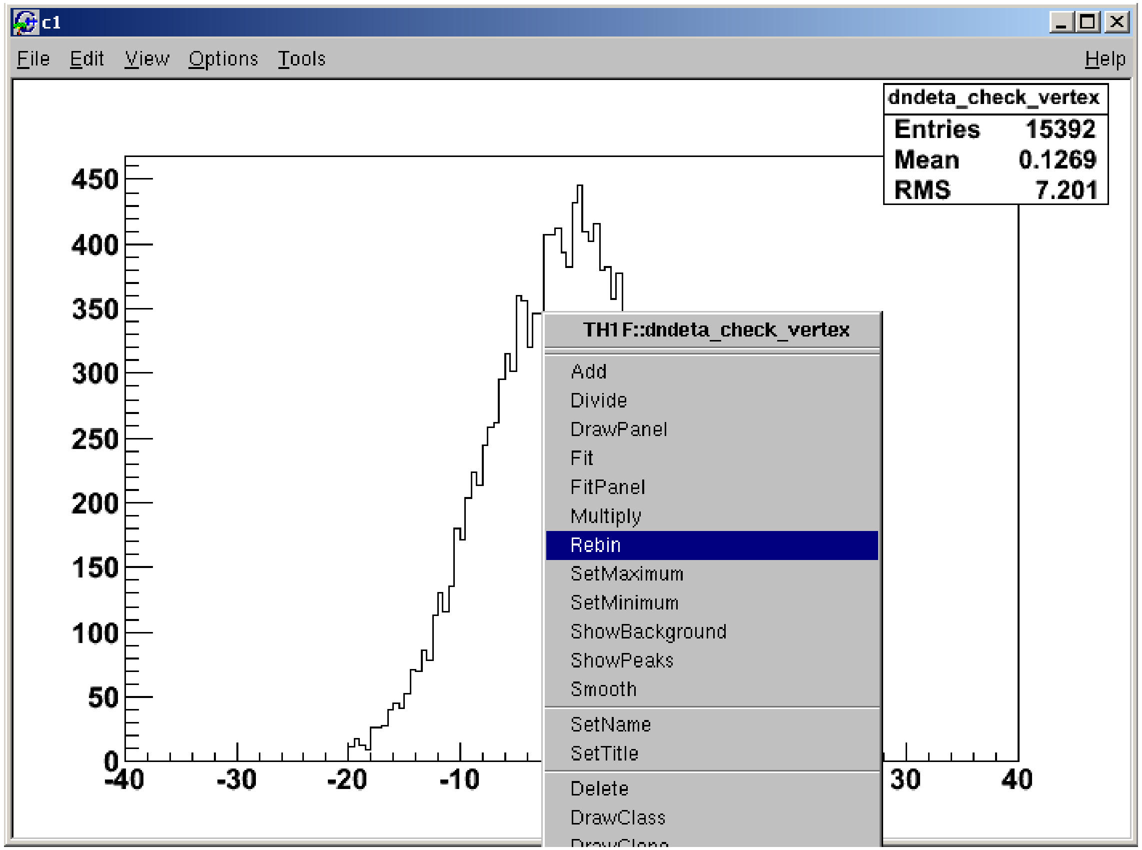
TMultiGraph



TGraphAsymmErrors(n,x,y,exl,exh,eyl,eyh)

Graphical User Interface

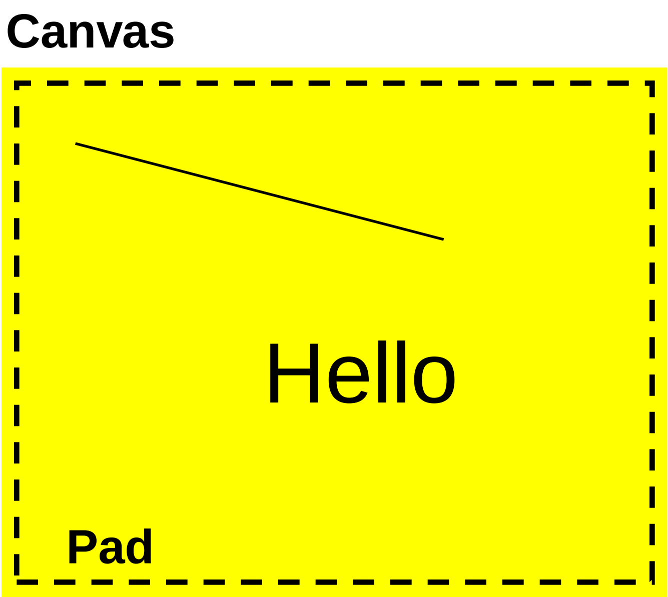
- Manipulate by moving objects or right clicking (→ context menu)



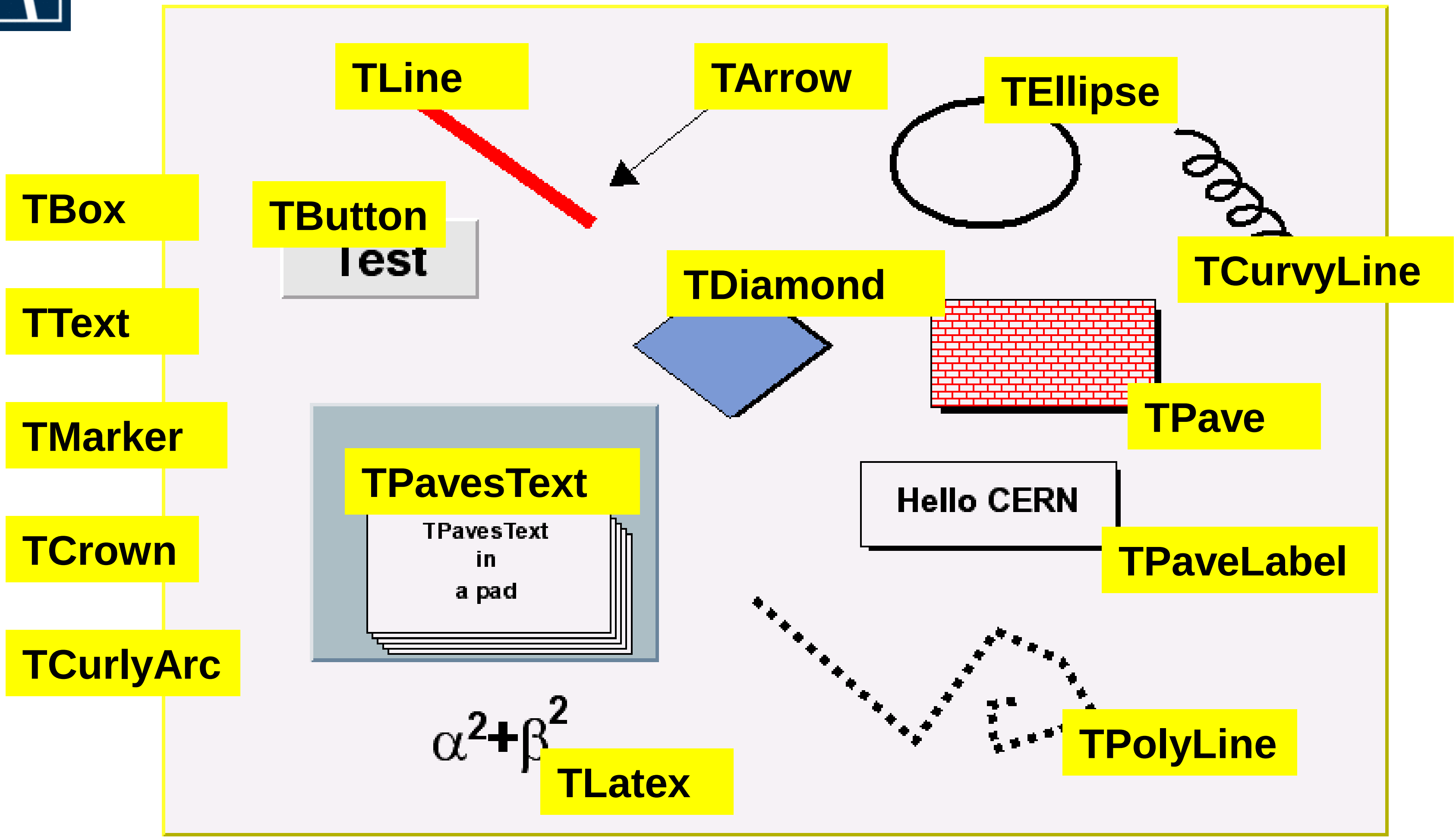
Graphics Objects

- You can draw with the command line
- The Draw function adds the object to the list of *primitives* of the current *pad*
- If no pad exists, a pad is automatically created
- A pad is embedded in a *canvas*
- You create one manually with new TCanvas
  - A canvas has one pad by default
  - You can add more

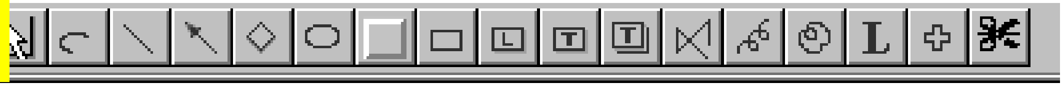
```
root [ ] TLine line(.1,.9,.6,.6)
root [ ] line.Draw()
root [ ] TText text(.5,.2,”Hello”)
root [ ] text.Draw()
```



More Graphics Objects



Can be accessed with the toolbar
View → Toolbar (in any canvas)



Full LateX support on screen and postscript

$ROOTSYS/tutorials/graphics/latex3.C

Born equation

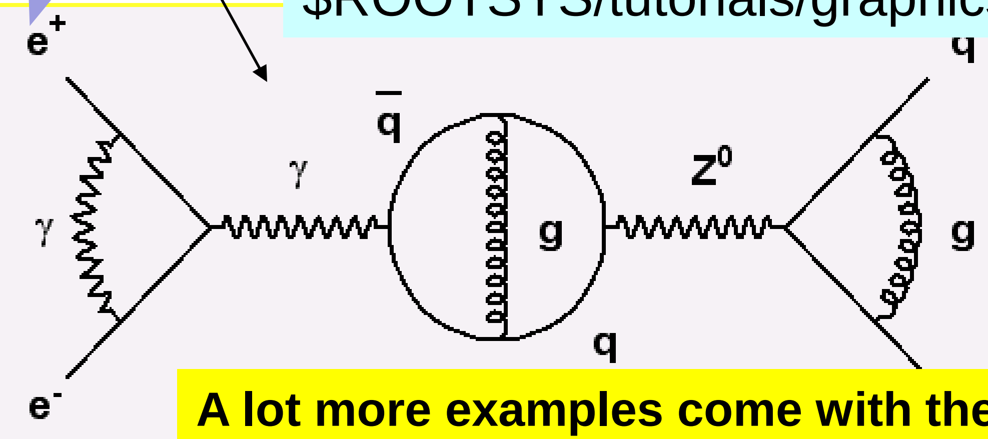
$$\frac{2s}{\pi\alpha^2}\frac{d\sigma}{d\cos\theta}(e^+e^-\to f\bar{f}) = \left|\frac{1}{1-\Delta\alpha}\right|^2(1+\cos^2\theta)$$

$$+4\,\mathrm{Re}\left\{\frac{2}{1-\Delta\alpha}\chi(s)\left[\tilde{g}_v^e\tilde{g}_v^f(1+\cos^2\theta)+2\,\tilde{g}_a^e\tilde{g}_a^f\cos\theta\right]\right\}$$

$$+16|\chi(s)|^2\left[(\tilde{g}_a^{e2}+\tilde{g}_v^{e2})(\tilde{g}_a^{f2}+\tilde{g}_v^{f2})(1+\cos^2\theta)+8\,\tilde{g}_a^e\tilde{g}_a^f\tilde{g}_v^e\tilde{g}_v^f\cos\theta\right]$$

Formula or diagrams can be edited with the mouse

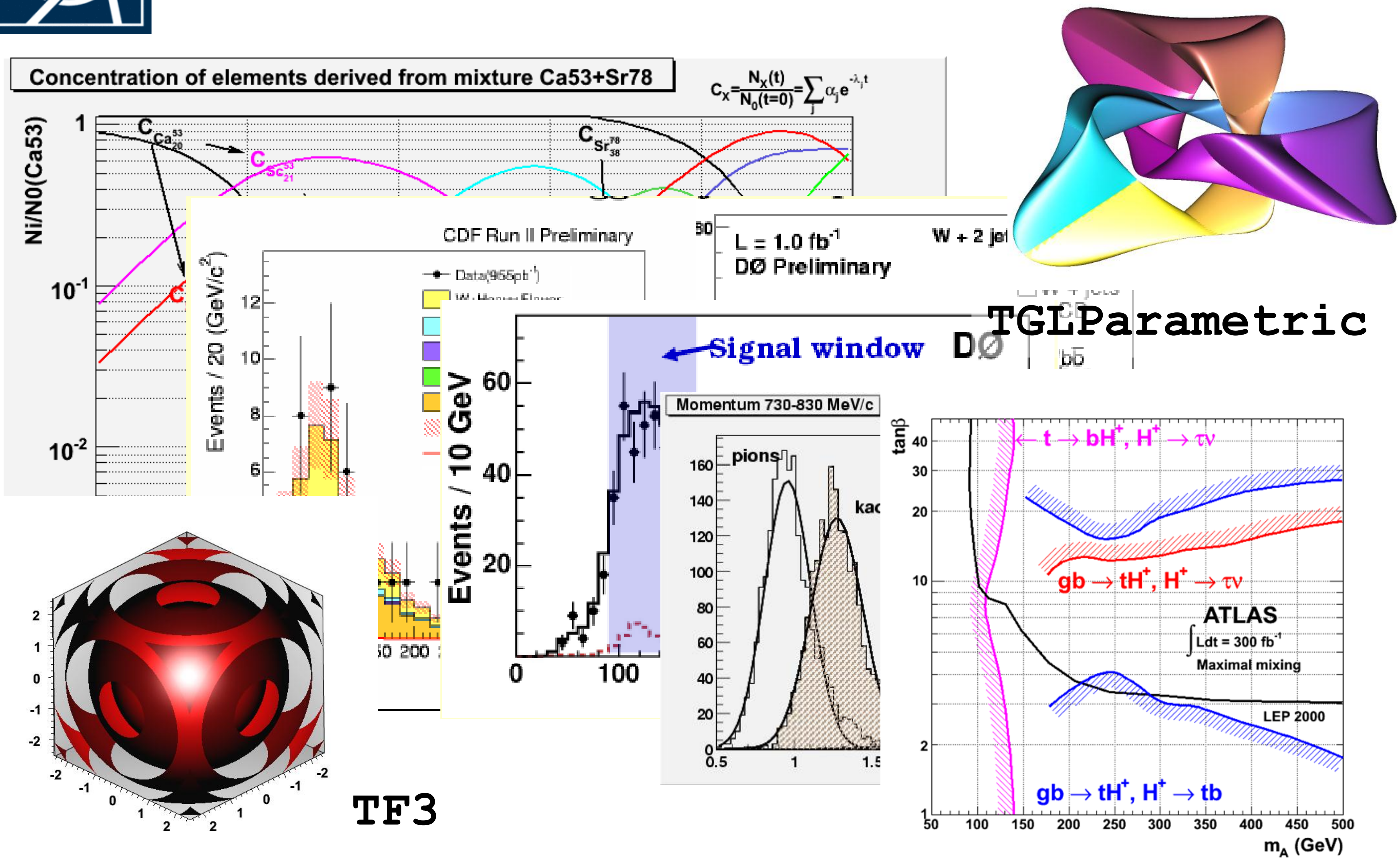
$ROOTSYS/tutorials/graphics/feynman.C



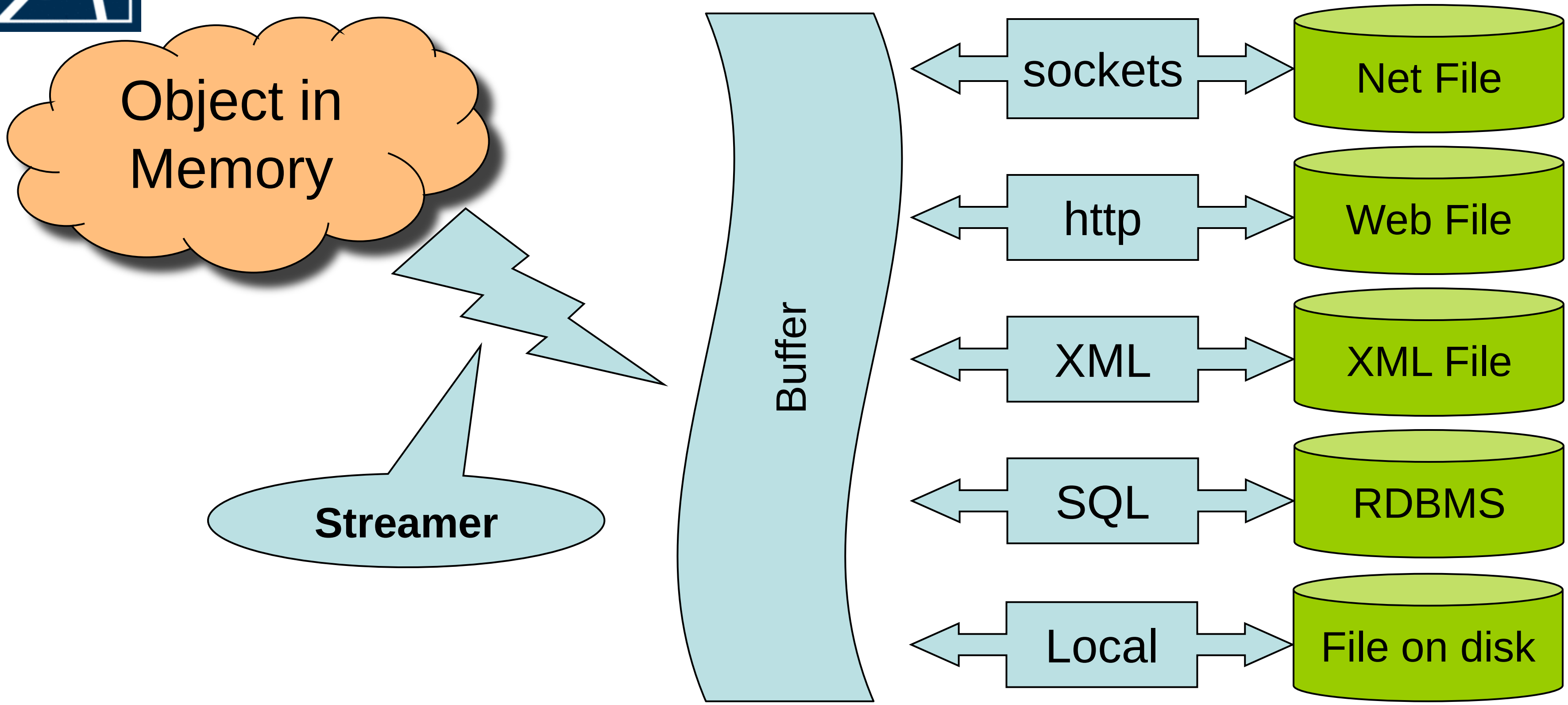
TCurlyArc
TCurlyLine
TWavyLine
and other building blocks for Feynmann diagrams

A lot more examples come with the ROOT installation

Graphics Examples



Input/Output



The automatically generated ROOT streamer for each class streams all class members, resolves circular dependencies and multiply referenced objects
→ No streamer function needs to be written
→ No need for separation of transient and persistent classes

Files

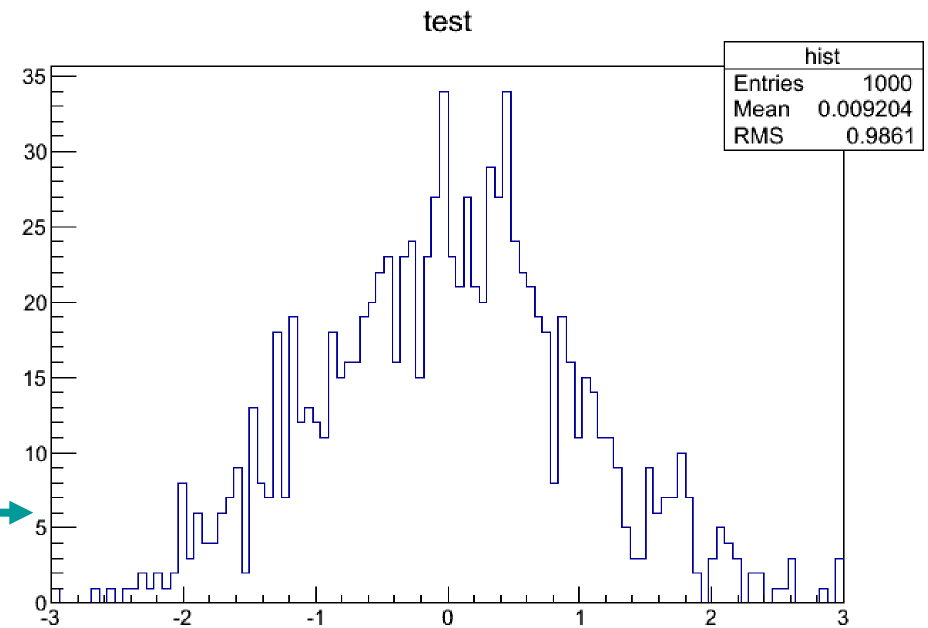
- TFile is the class to access files on your file system (and elsewhere)
- A TFile object may contain directories (TDirectory), like a Unix file system
- ROOT files are self describing
  - Dictionary for persistent classes written to the file
- Support for Backward and Forward compatibility
- Files created in 2006 must be readable in 2020

File Example

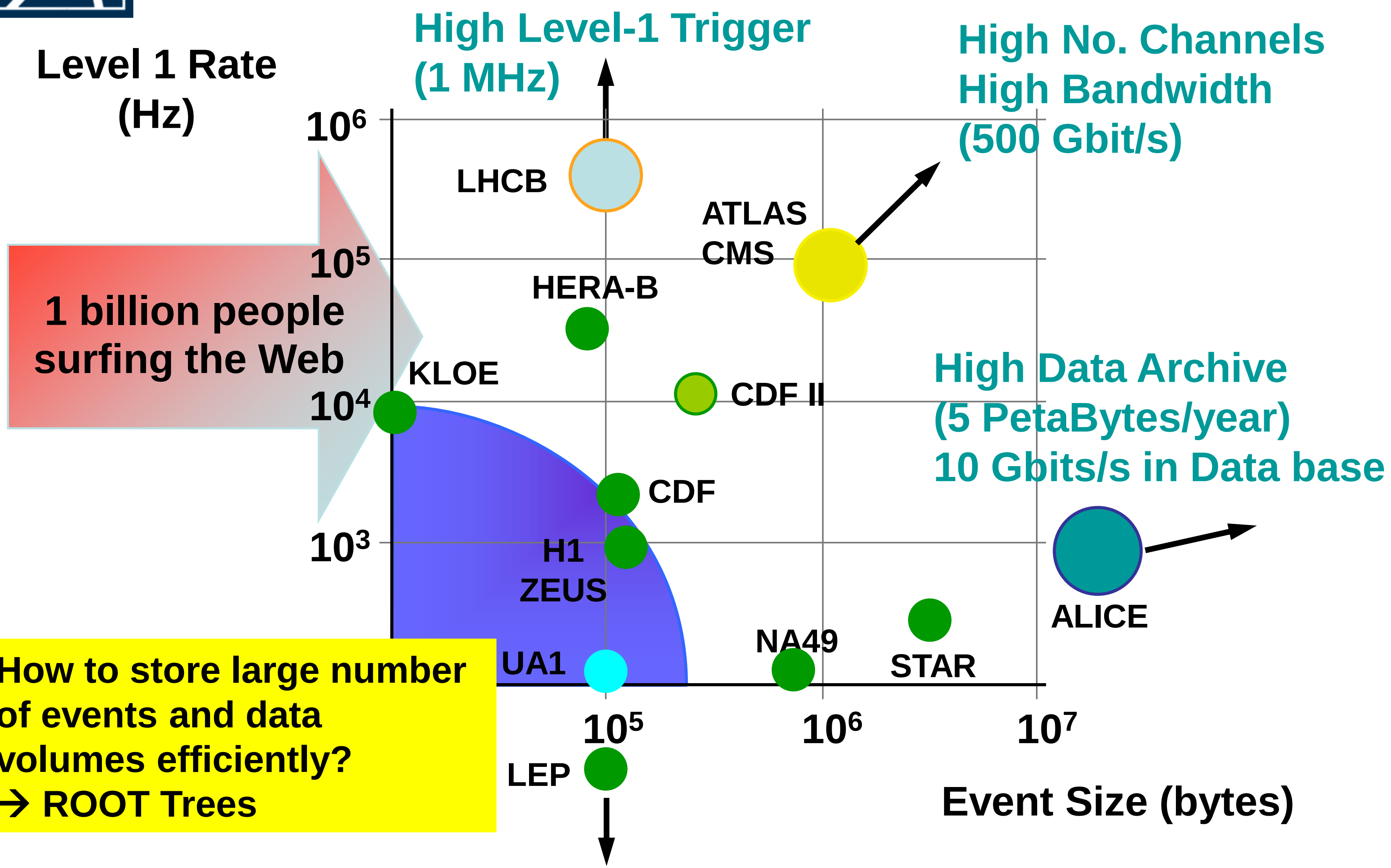
```
void keyWrite() {
   TFile f("file.root", "new");
   TH1F h("hist", "test", 100, -3, 3);
   h.FillRandom("gaus", 1000);
   h.Write()
}
```

```
void keyRead() {
   TFile f("file.root");
   TH1F *h = (TH1F*) f.Get("hist");
   h.Draw();
}
```

This works as well for your own class!



LHC: How Much Data?

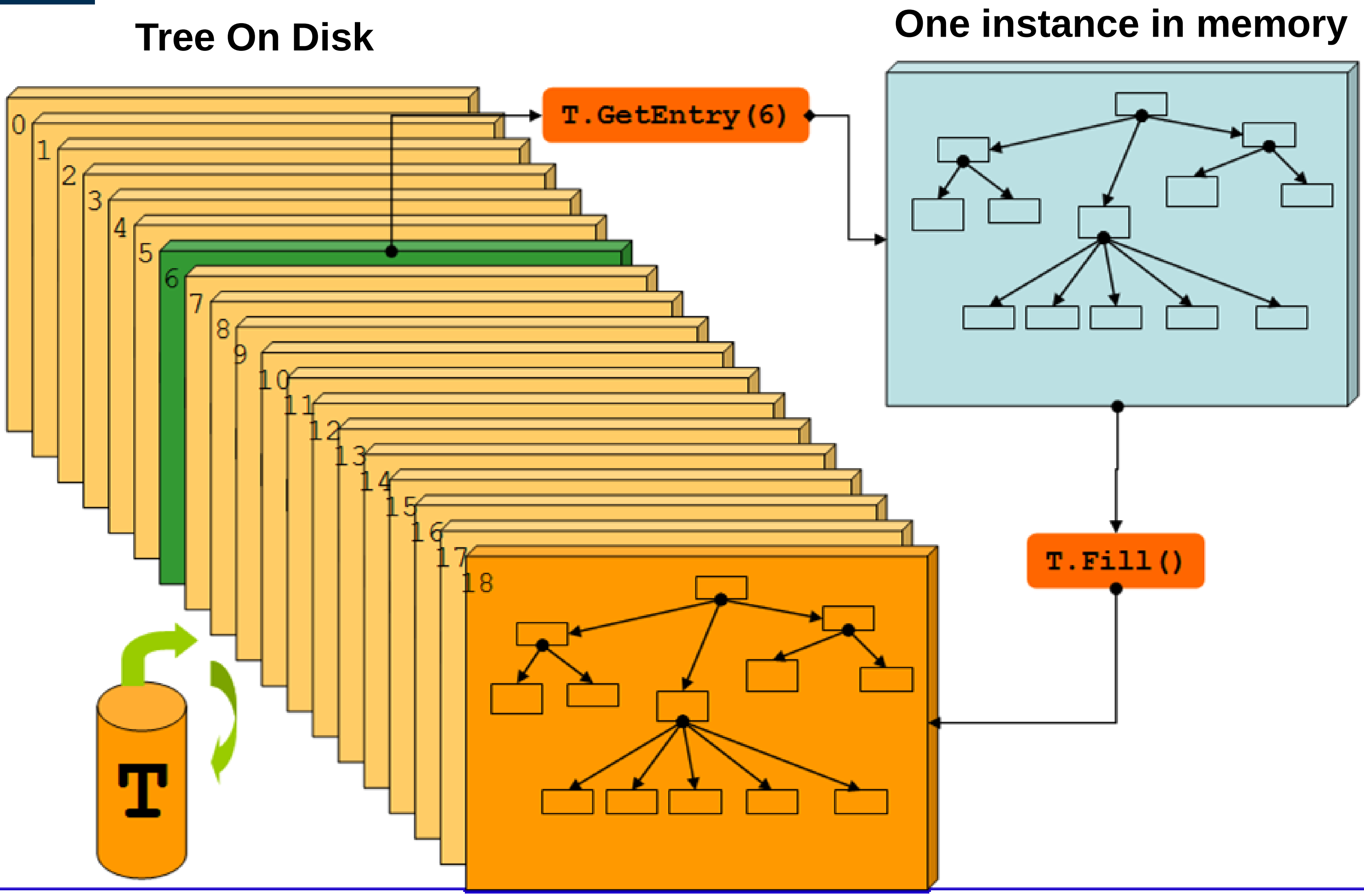


How to store large number of events and data volumes efficiently?
→ ROOT Trees

What is a ROOT Tree?

- Trees have been designed to support very large collections of objects. The overhead in memory is in general less than 4 bytes per entry.
- Trees allow direct and random access to any entry (sequential access is the most efficient)
- Trees are structured into branches and leaves. One can read a subset of all branches
- High level functions like TTree::Draw loop on all entries with selection expressions
- Trees can be browsed via TBrowser
- Trees can be analyzed via TTreeViewer

Stored Trees vs. Memory



TTree - Writing

- You want to store 1 million objects of type TMyEvent in a tree which is written into a file
- Initialization

```
TFile* f = TFile::Open("events.root", "RECREATE");
TTree* tree = new TTree("Events","Event Tree");
TMyEvent* myEvent = new TMyEvent;
TBranch* branch = tree->Branch("myevent",
                               "TMyEvent", &myEvent);
```

- Fill the tree (1 million times)
  - TTree::Fill copies content of member as new entry into the tree

```
myEvent->SetMember(…);
tree->Fill();
```

- Flush the tree to the file, close the file

```
tree->Write();
f->Close();
```

TTree - Reading

- Open the file, retrieve the tree and connect the branch with a pointer to TMyEvent

```
TFile *f = TFile::Open("events.root");
TTree *tree = (TTree*)f->Get("Events");
TMyEvent* myEvent = 0;
tree->SetBranchAddress("myevent", &myEvent);
```

- Read entries from the tree and use the content of the class

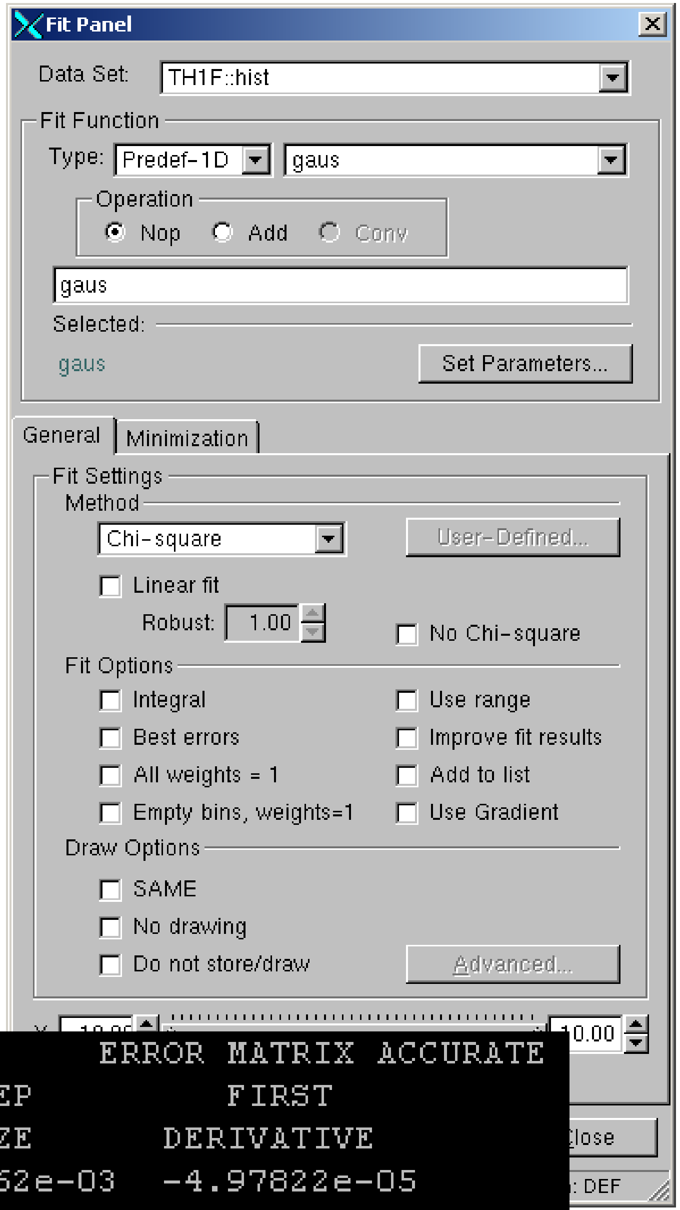
```
Int_t nentries = tree->GetEntries();
for (Int_t i=0;i<nentries;i++) {
  tree->GetEntry(i);
  cout << myEvent->GetMember() << endl;
}
```

A quick way to browse through a tree is to use a TBrowser

Fitting

- Fitting a histogram or graph
- With the GUI
  - If you just try which functions works well or need a single parameter
  - Right click on graph or histogram → Fit panel
- With the command line / macro
  - If you fit many histograms/graphs or several times

```
hist->Fit("gaus")
hist->FindFunction("gaus")->GetParameter(0)
```



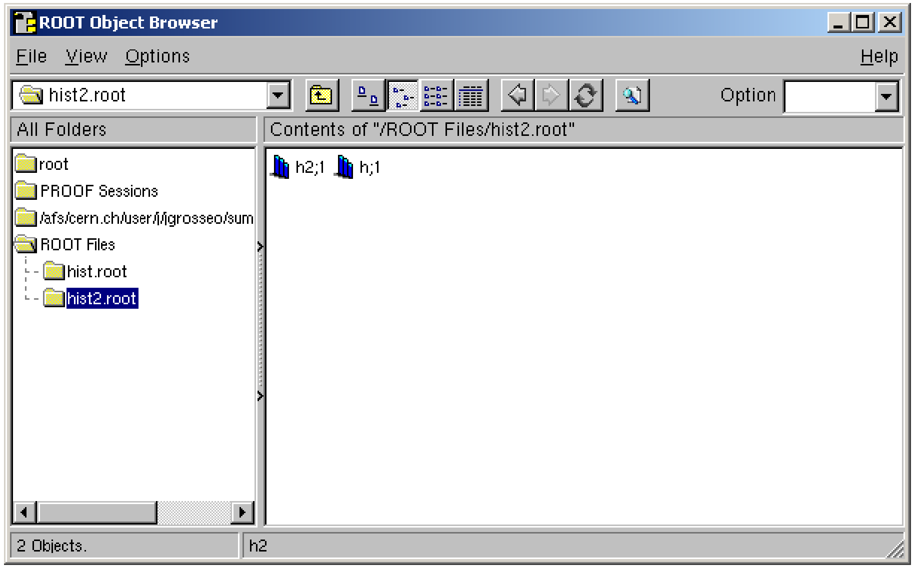
Fit parameters printed to the screen

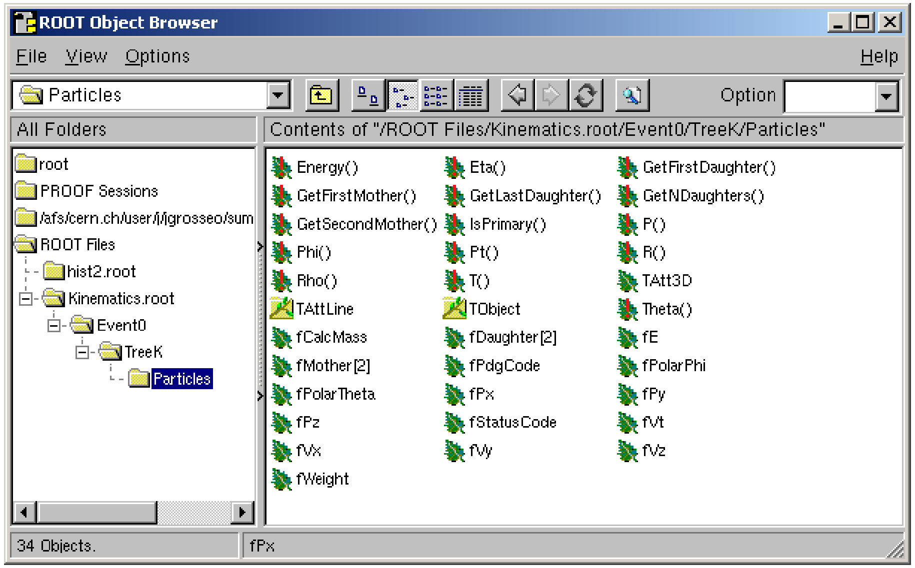
```
                    EDM=4.53716e-09    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     1.02075e+01   1.95215e+00   1.54262e-03  -4.97822e-05
   2  Mean        -1.19280e+00   4.02247e-01   4.69102e-04   1.26482e-04
   3  Sigma        2.55415e+00   5.30233e-01   4.12070e-05  -3.20639e-04
```

TBrowser

- The TBrowser can be used
  - to open files
  - navigate in them
  - to look at TTrees
- Starting a TBrowser

  root [ ] new TBrowser
- Open a file
- Navigate through the file
- Draw a histogram
- Change the standard style
  - Drop down menu in the top right corner
- Access a tree
- Plot a member





Understanding Errors

- Distinguish
  - Compiling error
    - Syntax errors
    - Missing declarations
  - Error while loading the library "dlopen error"
    - Missing implementation of a declared function (much more subtle)
    - Might even be in parent class
- Read error messages from top. Many other (weird) messages follow. Examples:
  - missing }
  - Missing include file
- Problems with macros? → Compile them to find errors

root [ ] .L macro2.C+

Basics of Debugging

- When there is a segmentation violation, you get the stack trace
  - It tells you where the crash happens
  - Find the relevant piece in the stack trace
    - Start from top
    - Few lines after "signal handler called"
    - Most of the times it makes only sense to look at lines that reference to your own code
  - Compile with debug ("g") to see line numbers

Stack Trace

```
 *** Break *** segmentation violation
Using host libthread_db library "/lib/tls/libthread_db.so.1".
Attaching to program: /proc/23893/exe, process 23893
[Thread debugging using libthread_db enabled]
[New Thread -1208858944 (LWP 23893)]
0x0077c7a2 in _dl_sysinfo_int80 () from /lib/ld-linux.so.2
#1  0x002b34b3 in __waitpid_nocancel () from /lib/tls/libc.so.6
#2  0x0025c779 in do_system () from /lib/tls/libc.so.6
#3  0x0022198d in system () from /lib/tls/libpthread.so.0
#5  0x009db83e in TUnixSystem::StackTrace (this=0x9daa440) at core/unix/src/TUnixSystem.cxx:2132
#6  0x009d962d in TUnixSystem::DispatchSignals (this=0x9daa440, sig=kSigSegmentationViolation) at core/unix/src/TUnixSys
#7  0x009d745d in SigHandler (sig=kSigSegmentationViolation) at core/unix/src/TUnixSystem.cxx:350
#8  0x009de7aa in sighandler (sig=11) at core/unix/src/TUnixSystem.cxx:3368
#9  <signal handler called>
#10 0x003effd8 in TSummerStudent::SomeFunction (this=0xa0154b0) at /home/shuttle/Fiete/./TSummerStudent_debug.C:14
#11 0x003ee355 in G__TSummerStudent_debug_C_ACLiC_dict_2564_0_3 (result7=0xbffe0420, funcname=0xa0153f8 "\001", libp=0xb
    at /home/shuttle/Fiete/./TSummerStudent_debug_C_ACLiC_dict.cxx:186
#12 0x00ed8dbf in Cint::G__ExceptionWrapper (funcp=0x3ee32e <G__TSummerStudent_debug_C_ACLiC_dict_2564_0_3>, result7=0xb
    hash=0) at cint/cint/src/Api.cxx:384
#13 0x00f81786 in G__execute_call (result7=0xbffe0420, libp=0xbffda5a0, ifunc=0xa0153f8, ifn=0) at cint/cint/src/newlink
#14 0x00f81ea6 in G__call_cppfunc (result7=0xbffe0420, libp=0xbffda5a0, ifunc=0xa0153f8, ifn=0) at cint/cint/src/newlink
#15 0x00f6295a in G__interpret_func (result7=0xbffe0420, funcname=0xbffe0020 "SomeFunction", libp=0xbffda5a0, hash=1242,
    at cint/cint/src/ifunc.cxx:5277
#16 0x00f4907c in G__getfunction (item=0xbffe3263 "SomeFunction()", known3=0xbffe267c, memfunc_flag=1) at cint/cint/src/
#17 0x0103b145 in G__getstructmem (store_var_type=112, varname=0xbffe0670 "�/5", membername=0xbffe3263 "SomeFunction()",
    varglobal=0x10d9ea0, objptr=2) at cint/cint/src/var.cxx:6691
#18 0x0102f234 in G__getvariable (item=0xbffe3260 "s->SomeFunction()", known=0xbffe267c, varglobal=0x10d9ea0, varlocal=0
#19 0x00f3ccc9 in G__getitem (item=0xbffe3260 "s->SomeFunction()") at cint/cint/src/expr.cxx:1884
#20 0x00f3b338 in G__getexpr (expression=0xbffe4b50 "s->SomeFunction()") at cint/cint/src/expr.cxx:1470
```

Basics of Debugging (2)

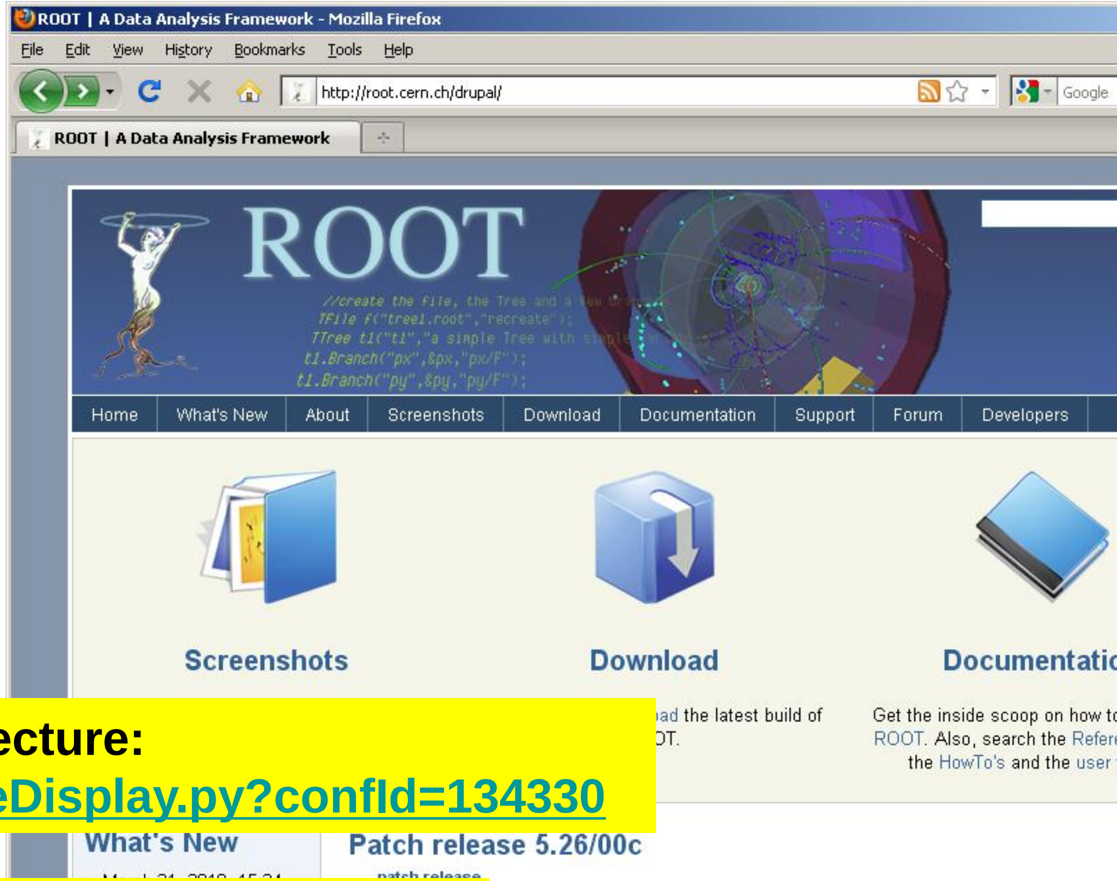
- Reproduce the problem in the debugger
- Most linux systems include gdb (GNU debugger)
- **$ gdb root.exe** (gdb root does not work)
  - Parameter to root have to be passed with
    $ gdb --args root.exe macro.C
  - On the gdb prompt, start the program: **(gdb) run**
- You will see the line where the crash happened
- Basic commands
  - **bt** = backtrace, gives the stack
  - **up**, **down** to navigate in the stack → go to the first frame with your code
  - **p <var>** → prints the variable <var> (of your code, e.g. particle)
  - **quit** to exit

ROOT is MORE….

- In this talk, I presented the most basic classes typically used during physics analyses
- ROOT contains many more libraries, e.g.
  - FFT library
  - Oracle, MySQL, etc interfaces
  - XML drivers
  - TMVA (Multi Variate Analysis)
  - GRID, networking and thread classes
  - Interfaces to Castor, Dcache, GFAL, xrootd
  - Interfaces to Pythia, Geant3, Geant4, gdml
  - Matrix packages, Fitting packages, etc

More Information...

- http://root.cern.ch
  - Download
  - Documentation
  - Tutorials
  - Online Help
  - Mailing list
  - Forum



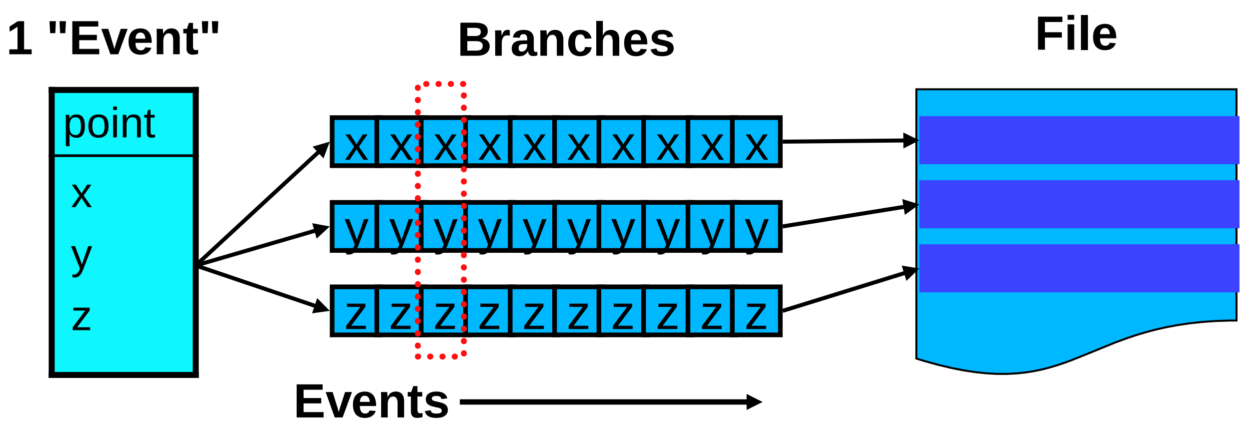
ROOT demo: slides + recorded lecture:
https://indico.cern.ch/conferenceDisplay.py?confId=134330

ROOT hands-on tutorial today and on Wednesday
Details have been announced

Backup

Trees: Split Mode

- The tree is partioned in branches
  - Each class member is a branch (in split mode)
  - When reading a tree, certain branches can be switched off
    → speed up of analysis when not all data is needed

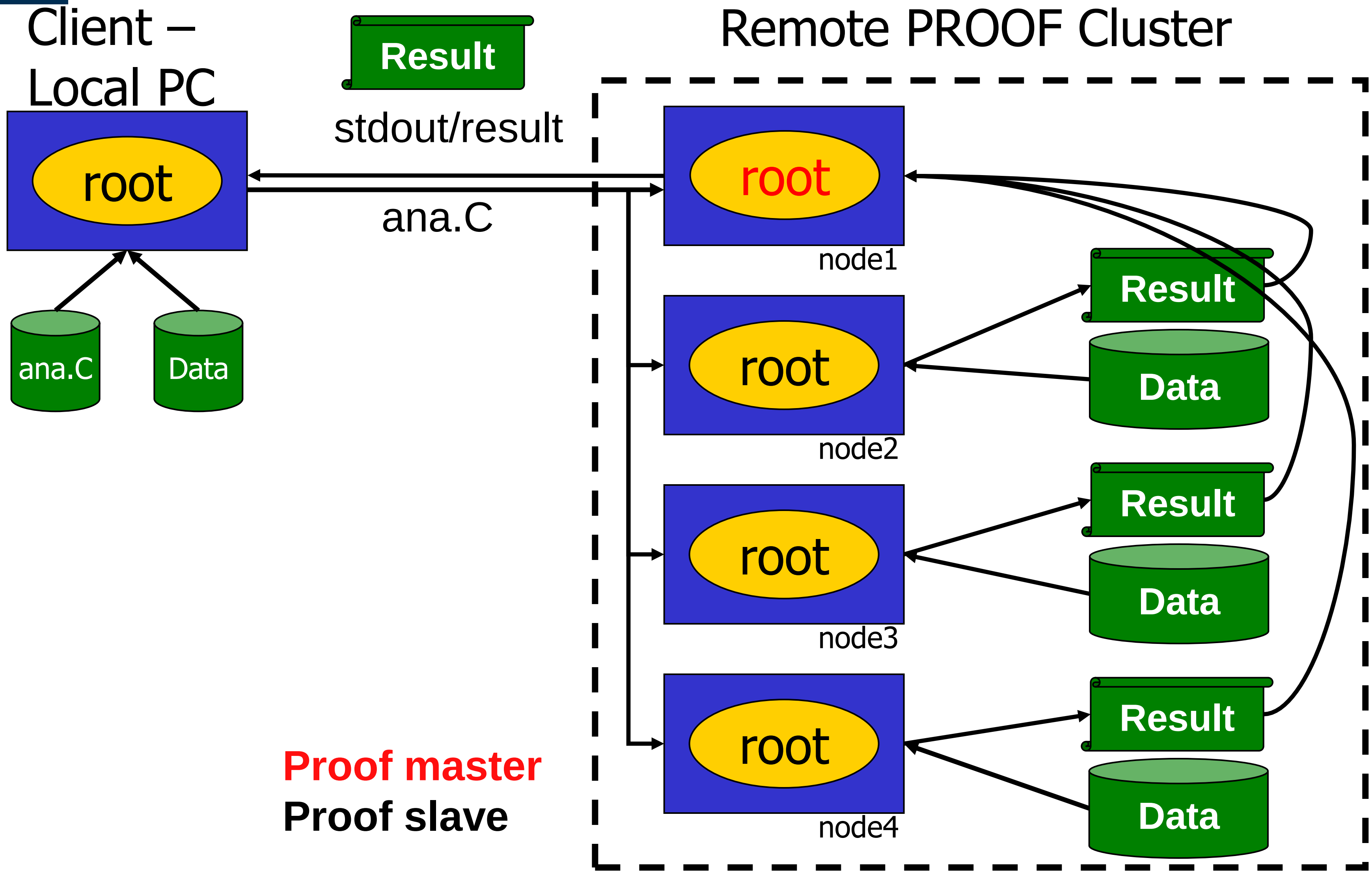


One Example: PROOF

- Parallel ROOT Facility
- Interactive parallel analysis on a local cluster
  - Parallel processing of (local) data (trivial parallelism)
  - Output handling with direct visualization
  - **Not** a batch system
- PROOF itself is not related to Grid
  - Can access Grid files
- The usage of PROOF is transparent
  - The same code can be run locally and in a PROOF system (certain rules have to be followed)
- PROOF is part of ROOT

Data does not need to be copied
Many CPUs available for analysis
→ much faster processing

PROOF Schema



Macros

- Combine lines of codes in macros
- Unnamed macro
  - No parameters
  - For example: macro1.C

```
{
   for (Int_t i=0; i<3; i++)
     printf("%d\n", i);
}
```

Data types in ROOT
Int_t (4 Bytes)
Long64_t (8 Bytes)
…
to achieve platform-independency

- Executing macros

root [ ] .x macro1.C
$ root –l macro1.C
$ root –l –b macro1.C (batch mode → no graphics)
$ root –l –q macro1.C (quit after execution)

Macros (2)

- Named macro
  - May have parameters
  - For example macro2.C:

```
void macro2(Int_t max = 10)
{
  for (Int_t i=0; i<max; i++)
    printf("%d\n", i);
}
```

- Running named macro

```
root [ ] .x macro2.C(12)
```

- Loading macros

```
root [ ] .L macro2.C
root [ ] macro2(12)
```

- Prompt vs. Macros
  - Use the prompt to test single lines while developing your code
  - Put code that is to be reused in macros

Don't forget to change the function name after renaming a macro

Plots for Papers

It is very useful to have all the code that creates a plot in one macro. Do not create "final" plots using the prompt or the mouse (you'll be doing it again and again).

Functions

- The class TF1 allows to draw functions

  root [ ] f = new TF1("func", "sin(x)", 0, 10)

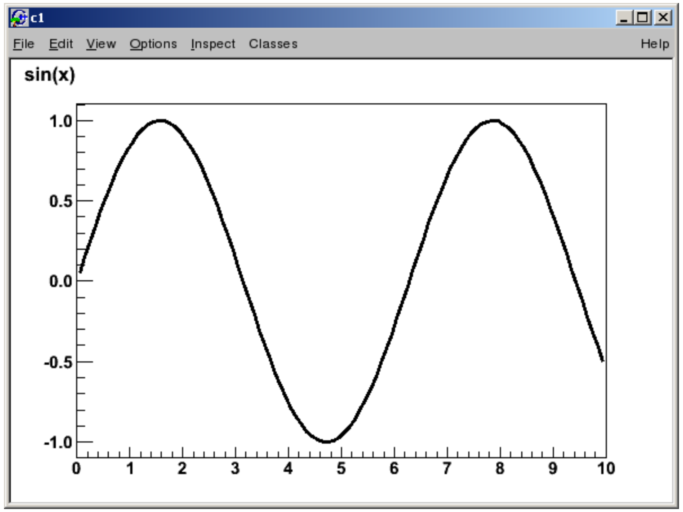
  - **"func"** is a (unique) name
  - **"sin(x)"** is the formula
  - **0, 10** is the x-range for the function

  root [ ] f->Draw()

- The style of the function can be changed on the command line or with the context menu (→ right click)

  root [ ] f->SetLineColor(kRed)

- The class TF2(3) is for 2(3)-dimensional functions



Canvas

Pointers vs. Value Types

- A value type contains an instance of an object
- A pointer *points* to the instance of an object
- Create a pointer

```
root [ ] TF1* f1 = new TF1("func", "sin(x)", 0, 10)
```

- Create a value type

```
root [ ] TF1 f2("func", "cos(x)", 0, 10)
```

- One can point to the other

```
TF1 f1b(*f1)          // dereferences and creates a copy
TF1* f2b = &f2        // points to the same object
```

Histograms

- Contain binned data – probably the most important class in ROOT for the physicist
- Create a TH1F (= one dimensional, float precision)

```
root [ ] h = new TH1F("hist", "my hist;Bins;Entries", 10, 0, 10)
```

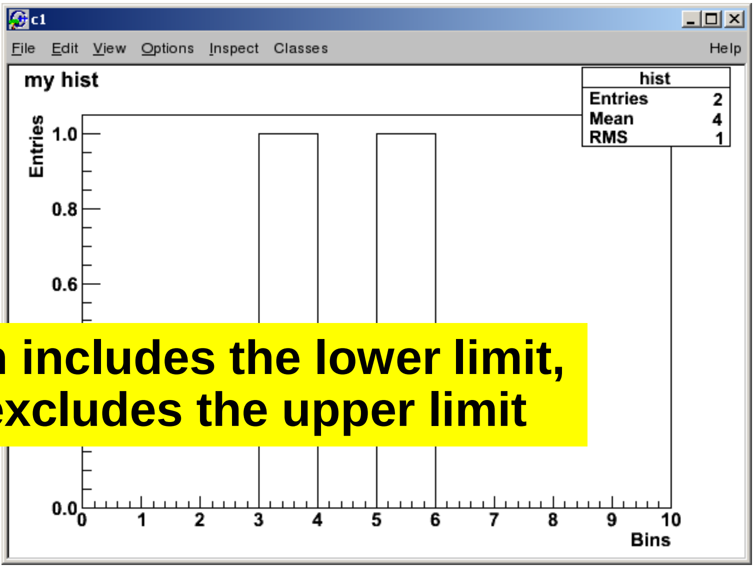
  - **"hist"** is a (unique) name
  - **"my hist;Bins;Entries"** are the title and the x and y labels
  - **10** is the number of bins
  - **0, 10** are the limits on the x axis. Thus the first bin is from 0 to 1, the second from 1 to 2, etc.

- **Fill the histogram**

```
root [ ] h->Fill(3.5)
root [ ] h->Fill(5.5)
```

- **Draw the histogram**

```
root [ ] h->Draw()
```



A bin includes the lower limit, but excludes the upper limit

Histograms (2)

- Rebinning

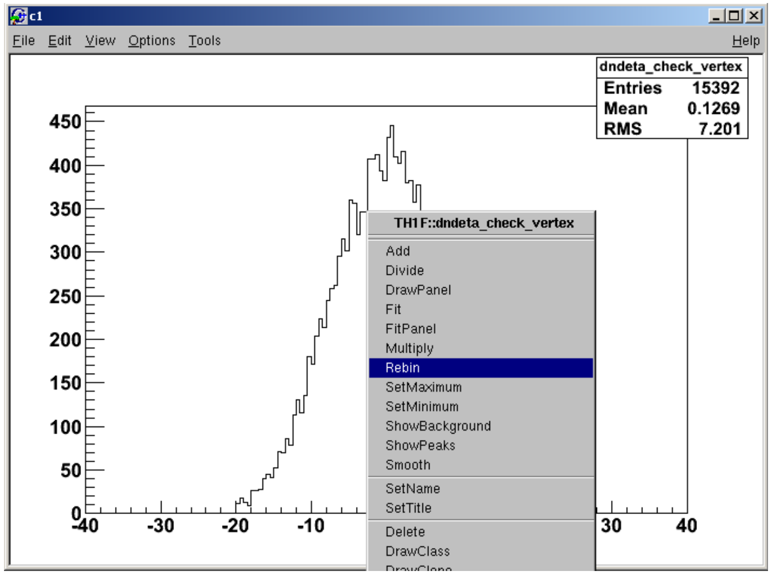
  root [ ] h->Rebin(2)

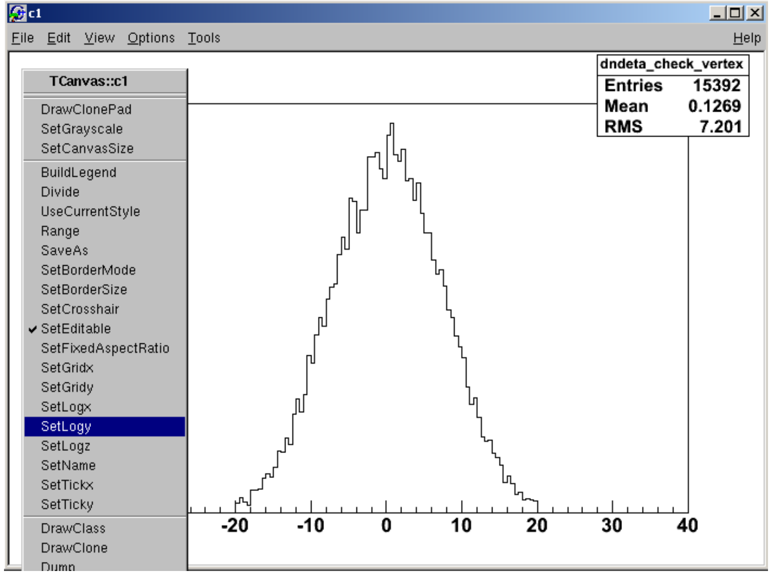
- Change ranges
  - with the mouse
  - with the context menu
  - command line

  root [ ] h->GetXaxis()->SetRangeUser(2, 5)

- Log-view
  - right-click in the white area at the side of the canvas and select SetLogx (SetLogy)
  - command line

  root [ ] gPad->SetLogy()



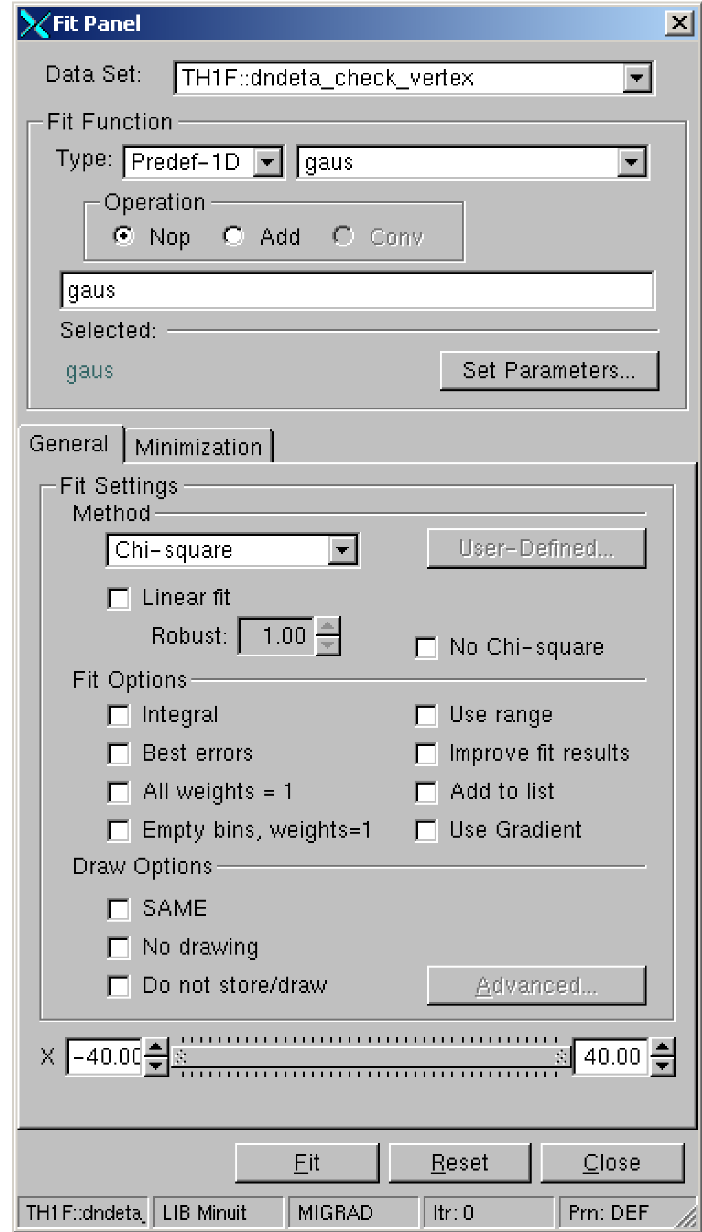


NB: example histogram in file hist.root

Fitting Histograms

- Interactive
  - Right click on the histogram and choose "fit panel"
  - Select function and click fit
  - Fit parameters
    - are printed in command line
    - in the canvas: options - fit parameters
- Command line

  root [ ] h->Fit("gaus")
  - Other predefined functions polN (N = 0..9), expo, landau

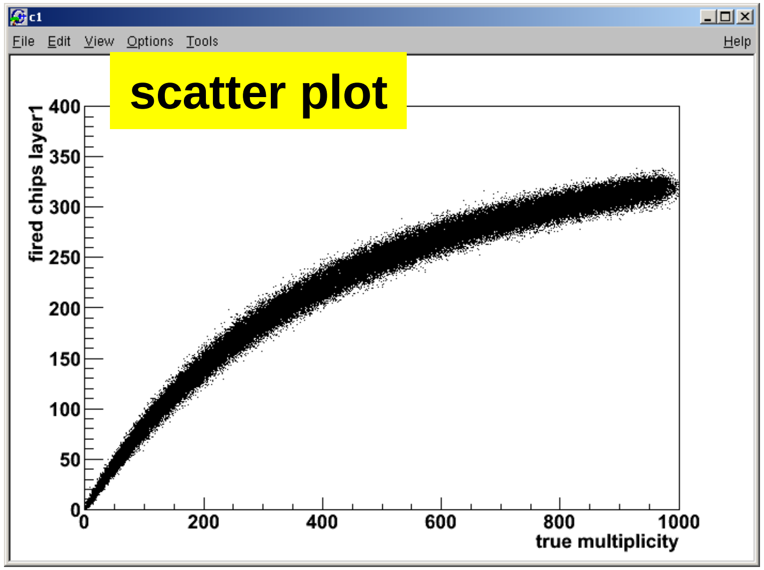


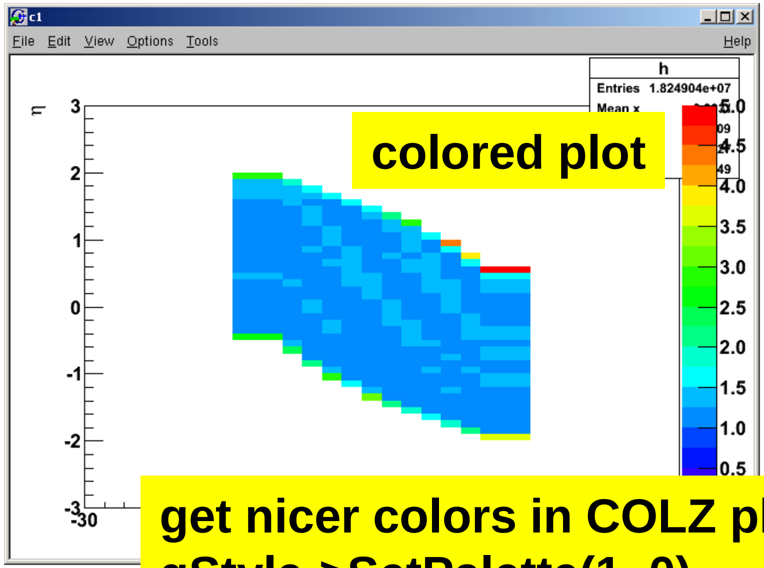
2D Histograms

root [ ] h->Draw()

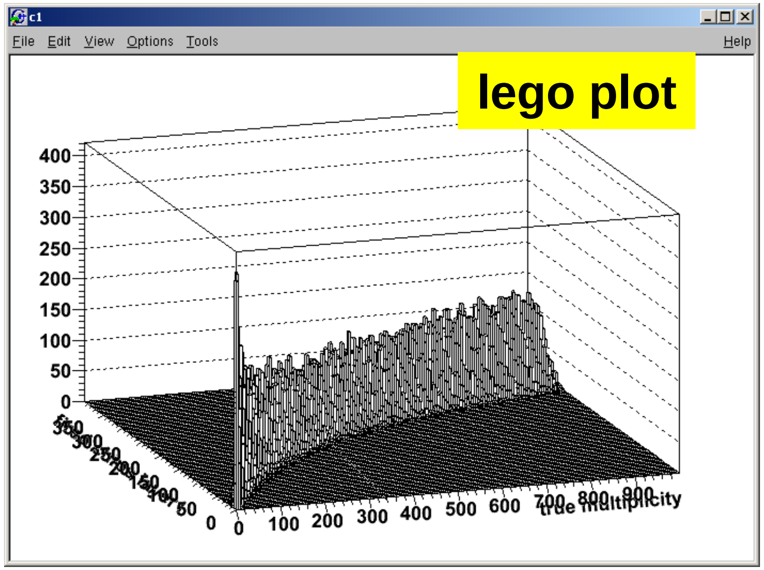
root [ ] h->Draw("LEGO")

root [ ] h2->Draw("COLZ")





get nicer colors in COLZ plots by gStyle->SetPalette(1, 0)



NB: h and h2 are in file hist2.root

Files

- The class TFile allows to store any ROOT object on the disk
- Create a histogram like before with
  h = new TH1F("hist", "my hist;…", 10, 0, 10)
  etc.
- Open a file for writing
  root [ ] file = TFile::Open("file.root", "RECREATE")
- Write an object into the file
  root [ ] h->Write()
- Close the file
  root [ ] file->Close()

"hist" will be the name in the file

Files (2)

- Open the file for reading

  root [ ] file = TFile::Open("file.root")
- Read the object from the file

  root [ ] hist->Draw()

  (only works on the command line!)
- In a macro read the object with

  TH1F* h = 0;

  file->GetObject("hist", h);
- What else is in the file?

  root [ ] .ls
- Open a file when starting root

  $ root file.root

  – Access it with the **_file0** or **gFile** pointer

Object ownership

After reading an object from a file don't close the file!

Otherwise your object is not in memory anymore

TNtuple

- Create a TNtuple

  root [ ] ntuple = new TNtuple("ntuple", "title", "x:y:z")

  - **"ntuple"** and **"title"** are the name and the title of the object
  - **"x:y:z"** reserves three variables named x, y, and z
- Fill it

  root [ ] ntuple->Fill(1, 1, 1)
- Get the contents

  root [ ] ntuple->GetEntries() number of entries

  root [ ] ntuple->GetEntry(0) for the first entry

  root [ ] ntuple->GetArgs()[1] for y (0 for x, and 2 for z)

  - These could be used in a loop to process all entries
- List the content

  root [ ] ntuple->Scan()

NB: The file ntuple.C produces this TNtuple with some random entries

TNtuple (2)

- Draw a histogram of the content
  - to draw only x

root [ ] ntuple->Draw("x")

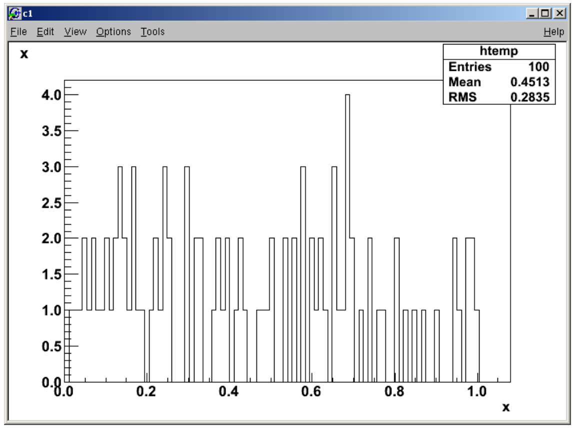
  - draw all x that fulfill x > 0.5

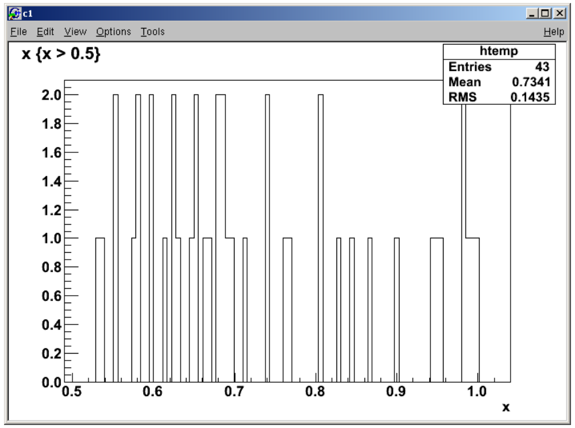
root [ ] ntuple->Draw("x", "x > 0.5")

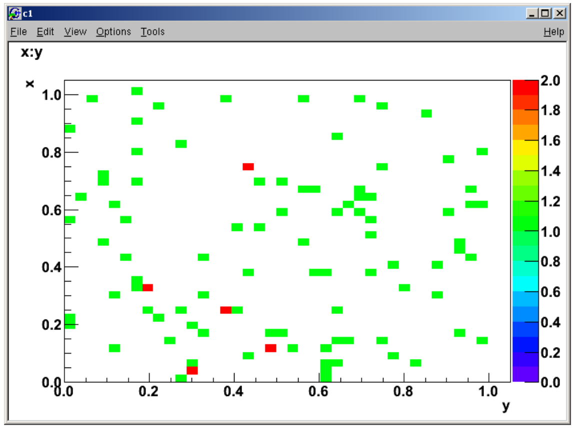
  - to draw x vs. y in a 2d histogram

root [ ] ntuple->Draw("x:y", "", "COLZ")

> **TNtuple (or TTree) with many entries may not fit in memory**
> **→ open a file before creating it**







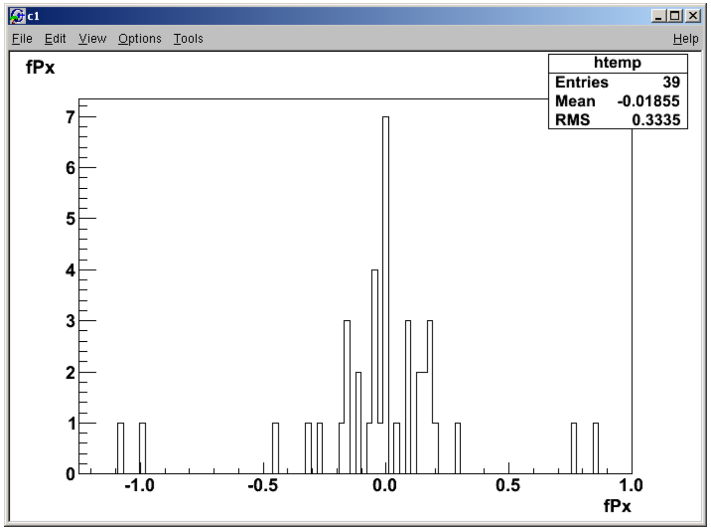
Trees (2)

- Accessing a more complex tree that contains classes
  - Members are accessible even without the proper class library
  - Might not work in all LHC experiments' frameworks
- Example: tree.root (containing kinematics from ALICE)

```
$ root tree.root
root [ ] tree->Draw("fPx")
root [ ] tree->Draw("fPx", "fPx < 0")
root [ ] tree->Draw("fPx",
   "abs(fPdgCode) == 211")
```

PDG code of pions

- From where do you know fPx, fPdgCode?
  - The tree contains TParticles
  - Check ROOT documentation: http://root.cern.ch/root/html/TParticle



Trees (3)

- Connecting a class with the tree

  root [ ] TParticle* particle = 0

  root [ ] tree->SetBranchAddress("Particles", &particle)

- Read an entry

  root [ ] tree->GetEntry(0)

  root [ ] particle->Print()

  root [ ] tree->GetEntry(1)

  root [ ] particle->Print()

  – These commands could be used in a loop to process all particles

The content of the TParticle instance is replaced with the current entry of the tree

```
root [5] particle->Print()
TParticle: pi0                p: -0.036864 -0.0
```

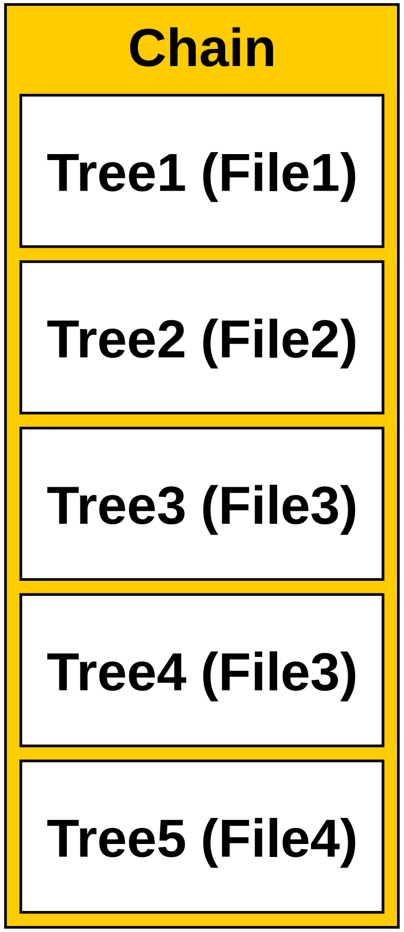
TChain

- A chain is a list of trees (in several files)
- Normal TTree functions can be used

```
root [ ] chain = new TChain("tree")
root [ ] chain->Add("tree.root")
root [ ] chain->Add("tree2.root")
root [ ] chain->Draw("fPx")
```

  - The Draw function iterates over both trees

Name of the tree in the files tree.root and tree2.root



Creating Classes

NB: This code is in TSummerStudent.C

- Any C++ class can be used with ROOT
- Classes derived from TObject can be used directly with many other ROOT classes (e.g. TList, TObjArray)

```
#include <TObject.h>
#include <TString.h>
class TSummerStudent : public TObject {
  private:
    TString fFirstName;
    Int_t fAge;
  public:
    const char* GetFirstName() const { return fFirstName; }
    Int_t GetAge() const { return fAge; }
    TSummerStudent(const char* firstname, Int_t age)
          : fFirstName(firstname), fAge (age) { }
    virtual ~TSummerStudent () {}
    ClassDef(TSummerStudent, 1)
};
```

TString to store strings

version number of class layout
when you add or change a member, increase the version number! 0 = not streamable

This macro adds some ROOT magic by including a dictionary created by CINT

Creating Classes (2)

- Include the class in ROOT

  root [ ] .L TSummerStudent.C+g

  "g" adds debug symbols

- Use it

  root [ ] s = new TSummerStudent(“Lena", 24)

  root [ ] s->GetFirstName()

- The object can be written in a file, send over the network etc.
- You can show the content of any ROOT class

  root [ ] s->Dump()

Resources

- Main ROOT page
  - http://root.cern.ch
- Class Reference Guide
  - http://root.cern.ch/root/html
- C++ tutorial
  - http://www.cplusplus.com/doc/tutorial/
  - http://www-root.fnal.gov/root/CPlusPlus/index.html
- Hands-on tutorials (especially the last one on the page)
  - http://root.cern.ch/drupal/content/tutorials-and-courses

ROOT tutorial on July 12th and 19th
Details have been (will be) announced