Introduction to ROOT

Jan Fiete Grosse-Oetringhaus, CERN PH/ALICE

Summer Student Lectures 2010

30th July

Content

- First lecture
  - Introduction to the ROOT framework
    - Library structure
    - CINT
    - Macros
    - Histograms, Graphs, Advanced graphics examples
    - Input/Output: Files, Trees
    - Fitting
- Second lecture
  - Practical introduction to the ROOT framework (demo)
- Nomenclature
  - **Blue: you type it**
  - **Red: you get it**

ROOT in a Nutshell

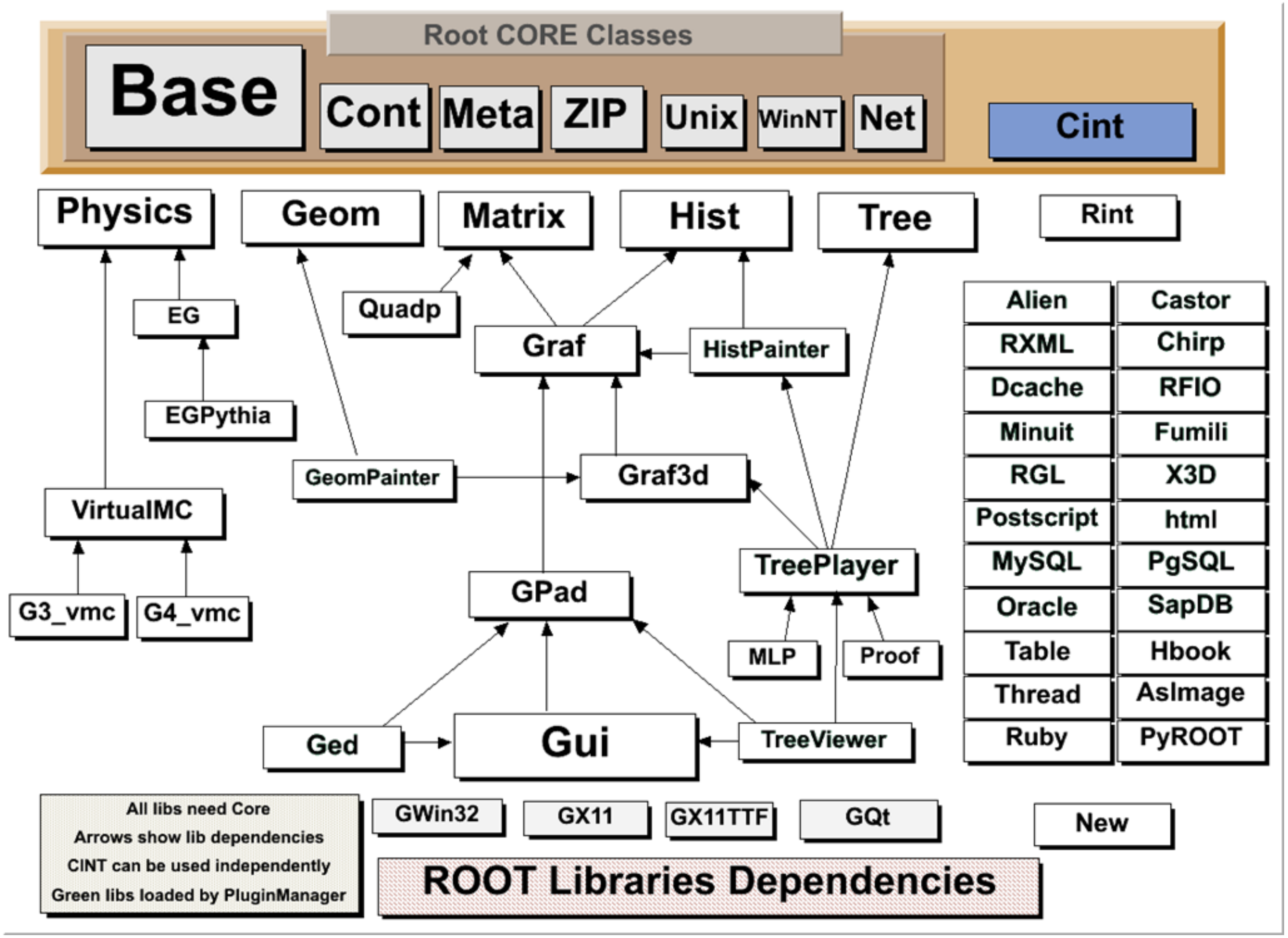
- ROOT is a large Object-Oriented data handling and analysis framework
- Efficient object store scaling from KB's to PB's
- C++ interpreter
- Extensive 2D+3D scientific data visualization capabilities
- Extensive set of multi-dimensional histograming, data fitting, modeling and analysis methods
- Complete 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



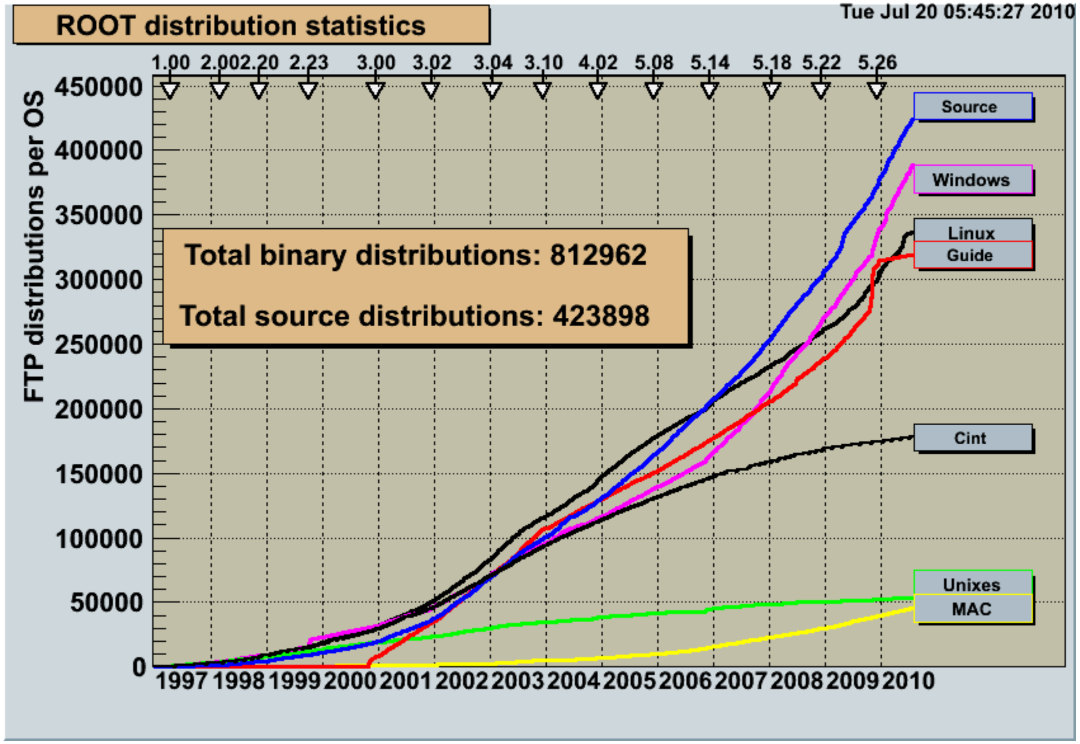
- Over 2500 classes
- 3,000,000 lines of code
- CORE (8 Mbytes)
- CINT (2 Mbytes)
- Most libraries linked on demand via plug-in manager (only a subset shown)
- 100 shared libs

ROOT: An Open Source Project

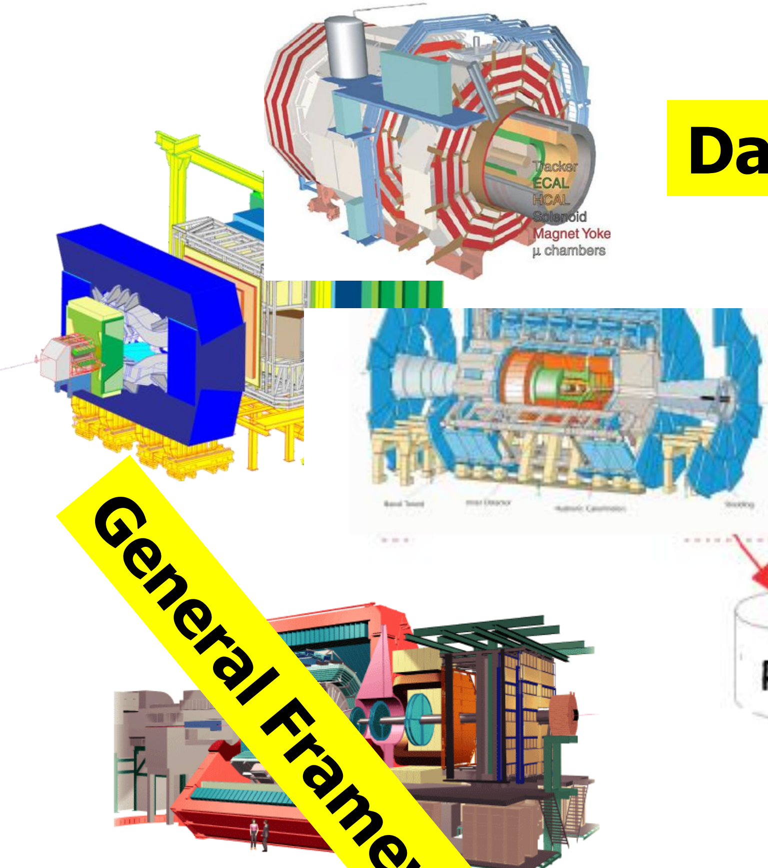
- The project was started in Jan 1995
- First release Nov 1995
- The project is developed as a collaboration between:
  - Full time developers:
    - 7 people full time at CERN (PH/SFT)
    - 2 developers at Fermilab/USA
  - Large number of part-time contributors (160 in CREDITS file)
  - A long list of users giving feedback, comments, bug fixes and many small contributions
    - 4609 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 more than 800,000 times since 1997
- The estimated user base is about 20,000 people



ROOT Application Domains

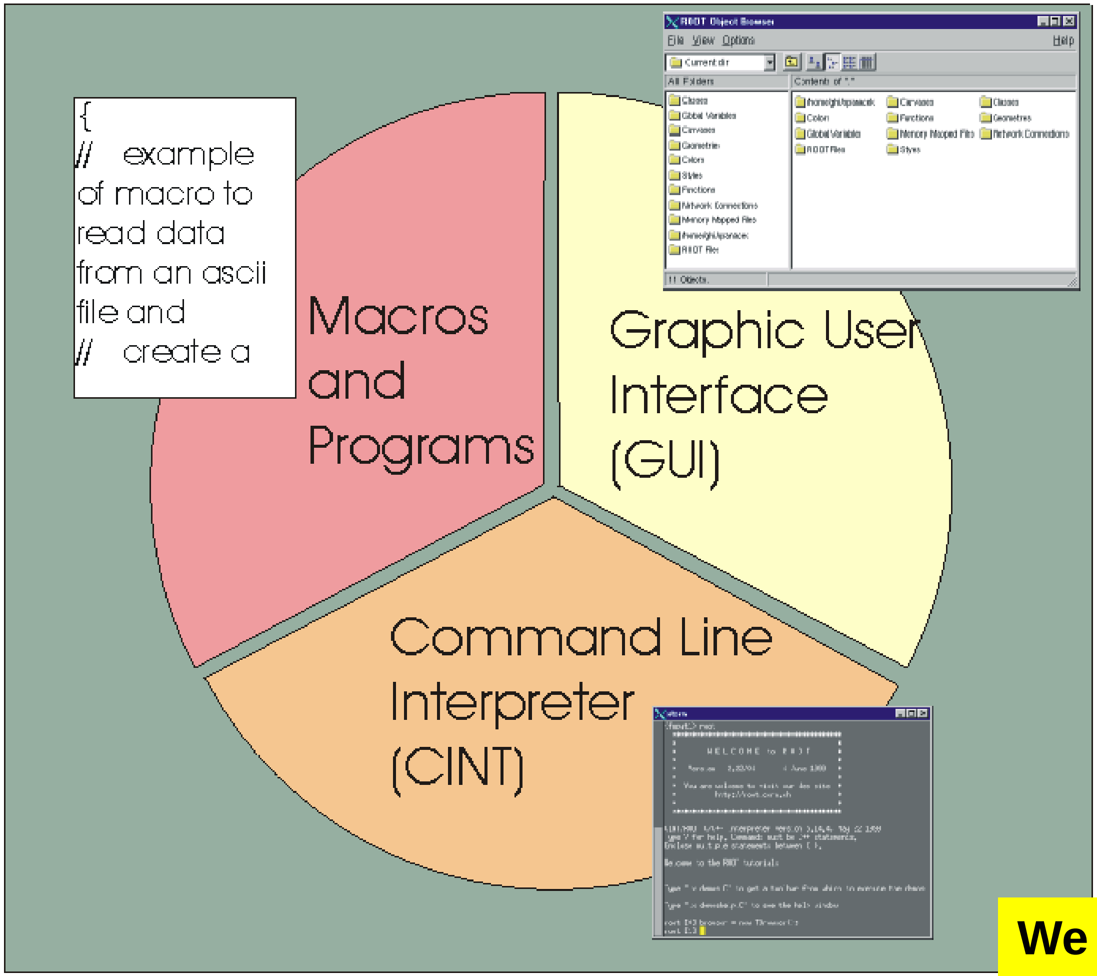


Data Analysis & Visualization

General Framework

Data Storage: Local, Network

Three User Interfaces

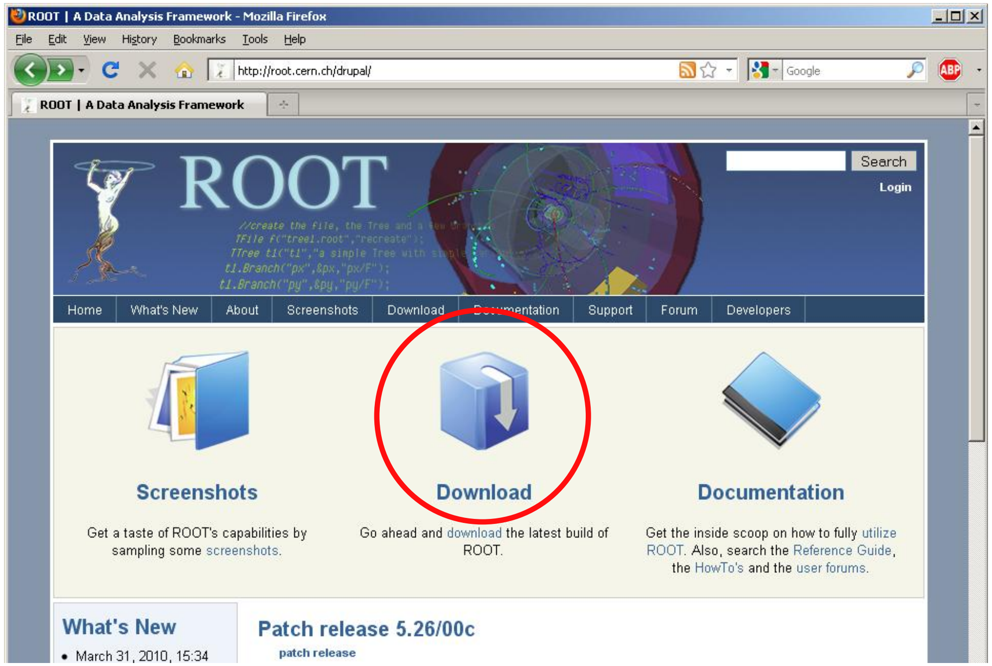


- GUI windows, buttons, menus
- Command line CINT (C++ interpreter)
- Macros, applications, libraries (C++ compiler and interpreter)

We will see that in the demo in the second part of the lecture

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
  - Couple of 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

Named 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 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!

Unnamed Macros

- No function, just statements

```
{
  float ret = 0.42;
  return sin(ret);
}
```

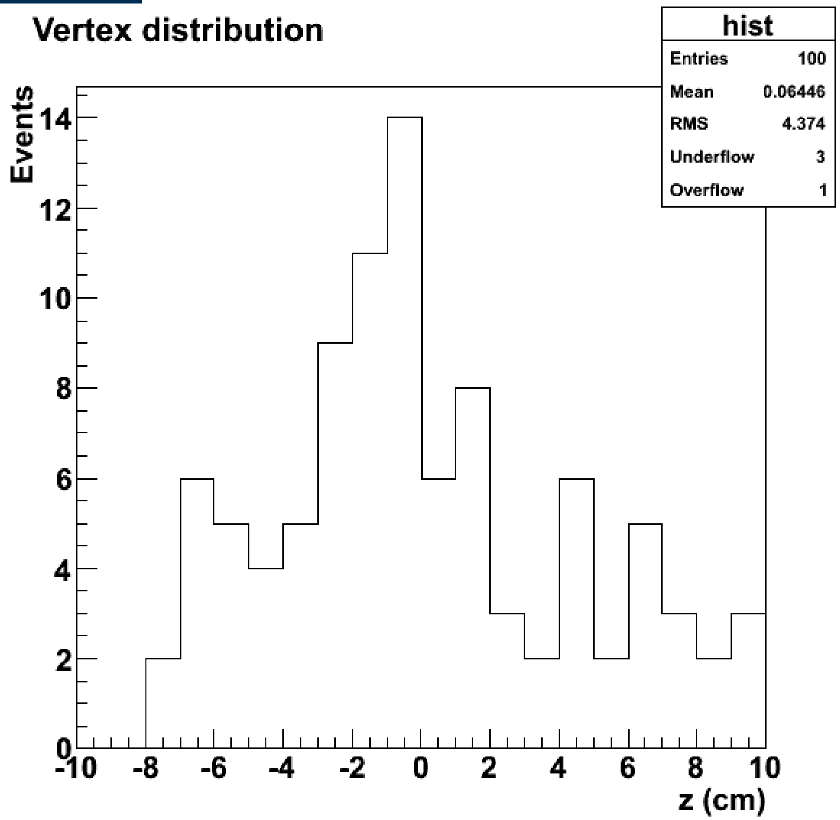
→ **saved in mymacro.C**

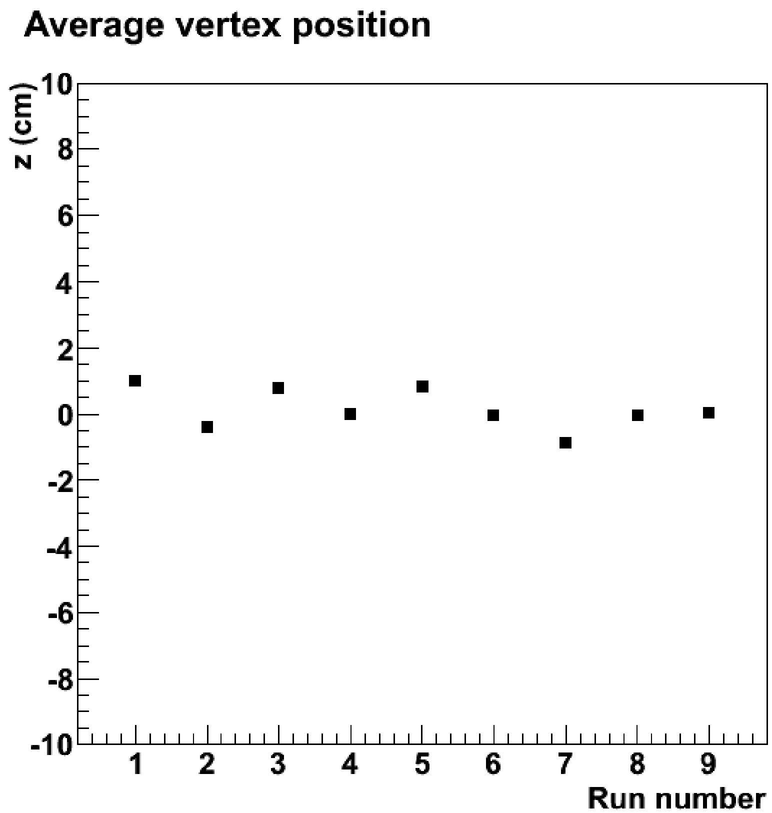
- Execute with **root [0] .x mymacro.C**
  - No functions, thus no arguments
- Named macro recommended!

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





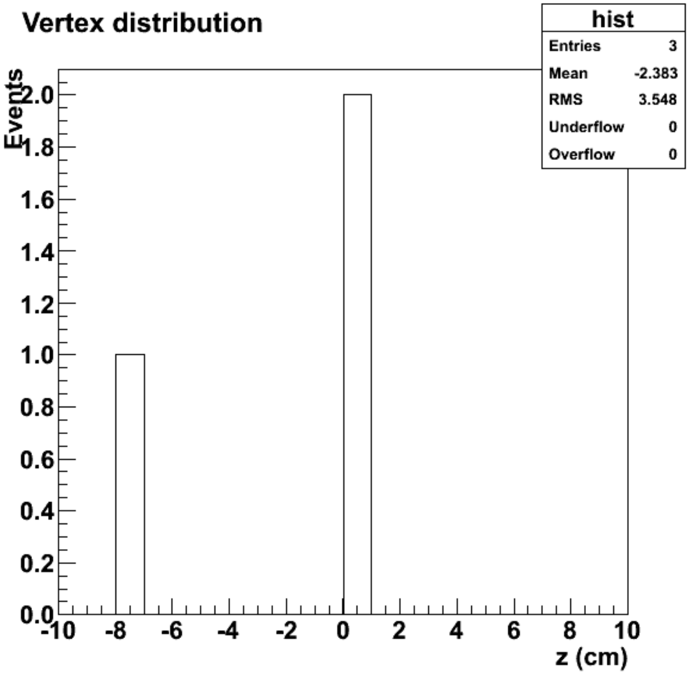
- Container for binned data
  - Most of HEP's distributions
- Container for distinct points
  - Calculation or fit results

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();
```



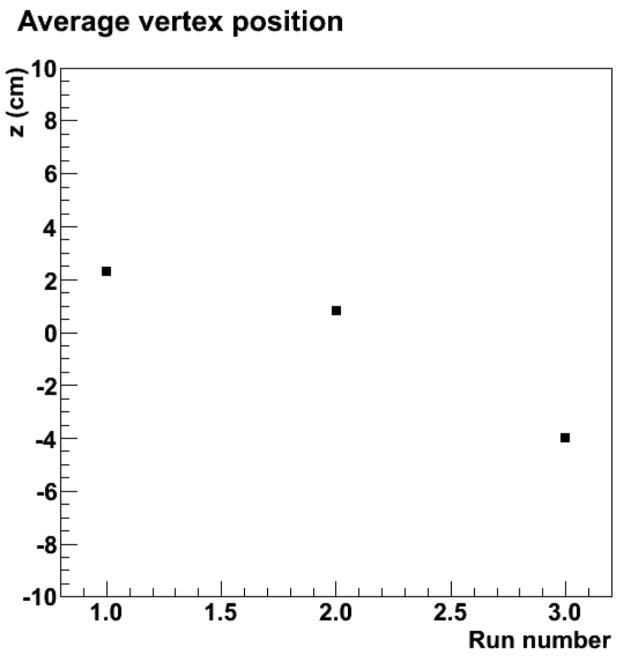
NB: All ROOT classes start with T
Looking for e.g. a string? Try TString

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)

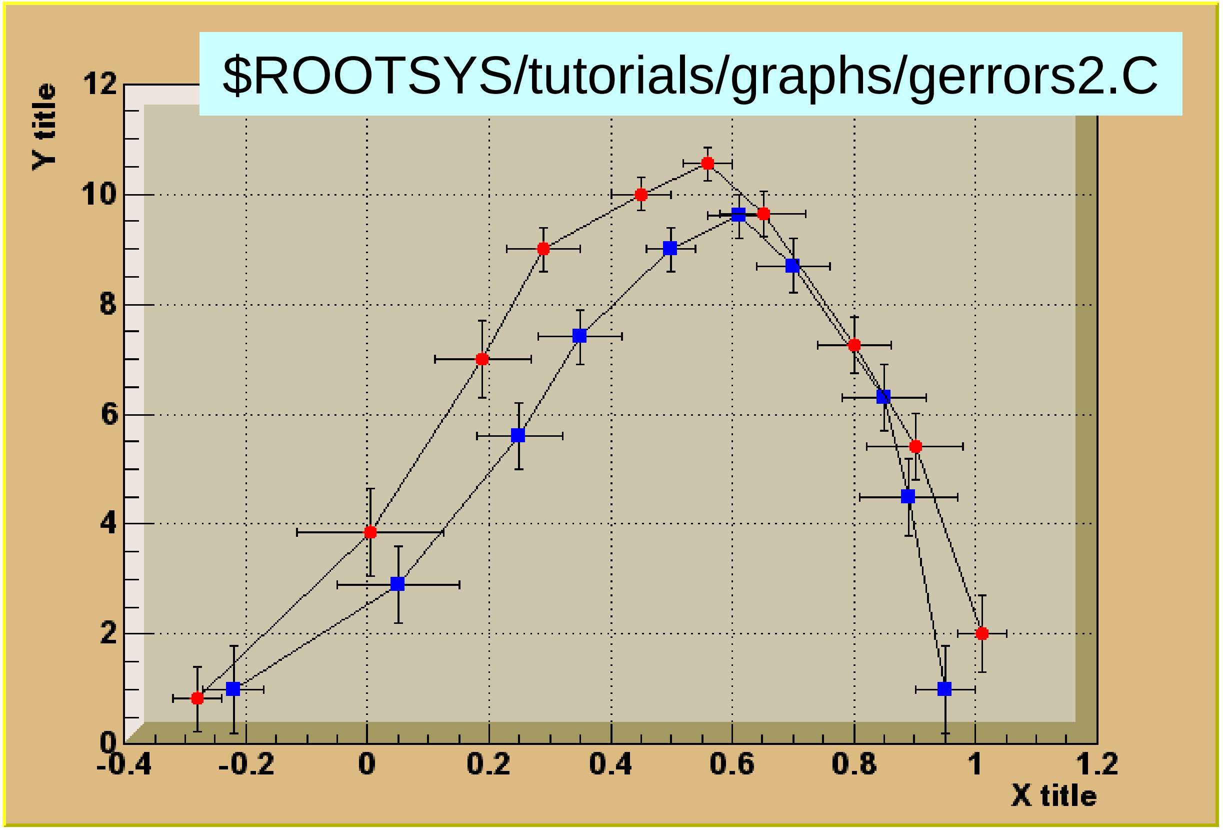
TGraphErrors(n,x,y,ex,ey)

TGraph(n,x,y)

TCutG(n,x,y)

TMultiGraph

$ROOTSYS/tutorials/graphs/gerrors2.C



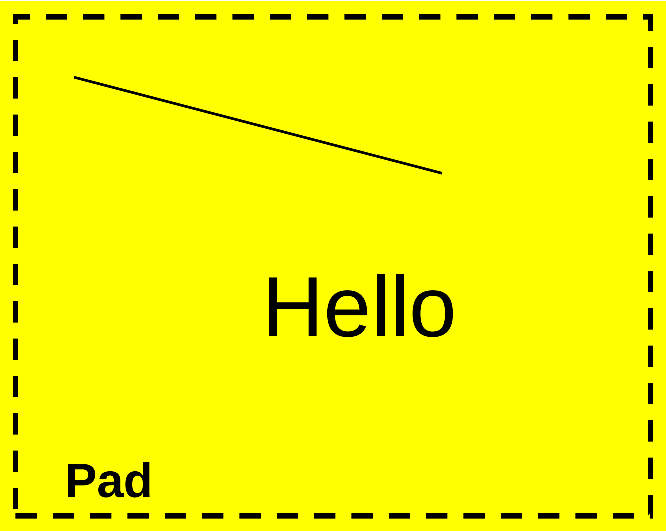
TGraphAsymmErrors(n,x,y,exl,exh,eyl,eyh)

Graphics Objects

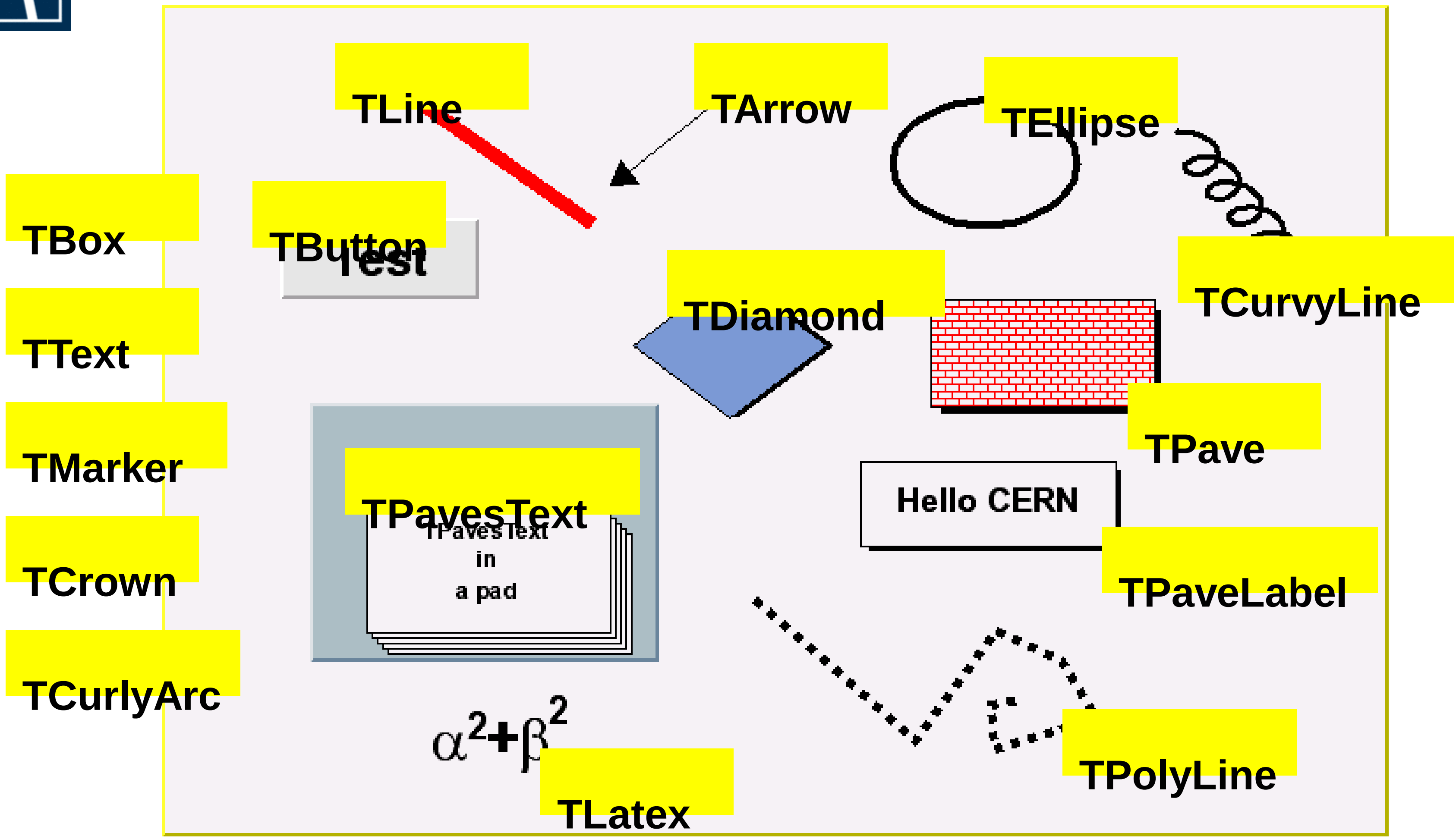
- You can draw with the command line
- The Draw function adds the object to the list of *primitives* of the current *pad*
- If no pad exists, a pad is automatically created
- A pad is embedded in a *canvas*
- You create one manually with new TCanvas
  - A canvas has one pad by default
  - You can add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

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

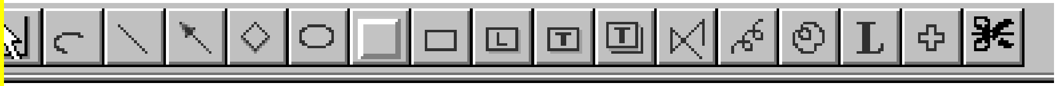
Canvas



More Graphics Objects

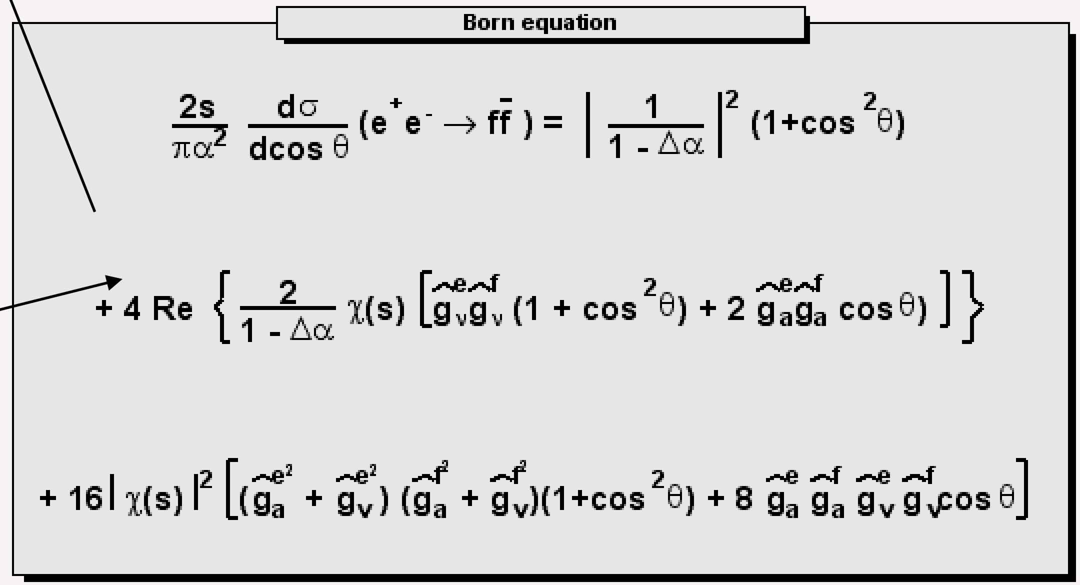


Can be accessed with the toolbar
View → Toolbar (in any canvas)



Full LateX support on screen and postscript

$ROOTSYS/tutorials/graphics/latex3.C



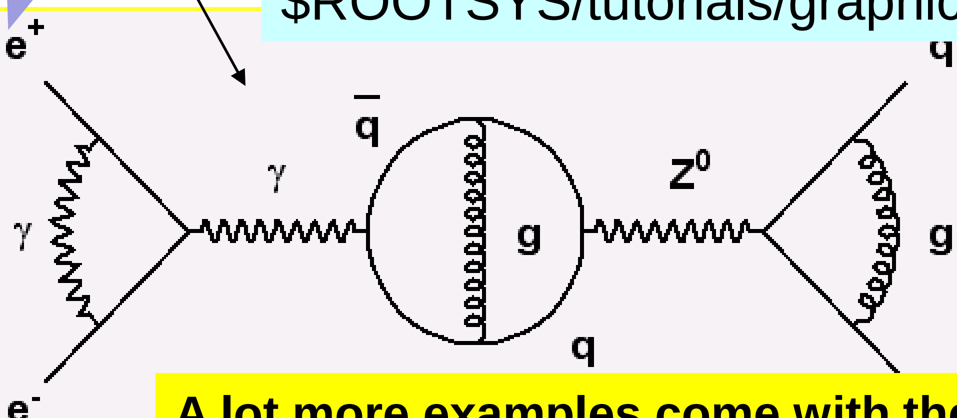
$$\frac{2s}{\pi\alpha^2}\frac{d\sigma}{d\cos\theta}(e^+e^-\rightarrow f\bar{f}) = \left|\frac{1}{1-\Delta\alpha}\right|^2(1+\cos^2\theta)$$

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

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

Formula or diagrams can be edited with the mouse

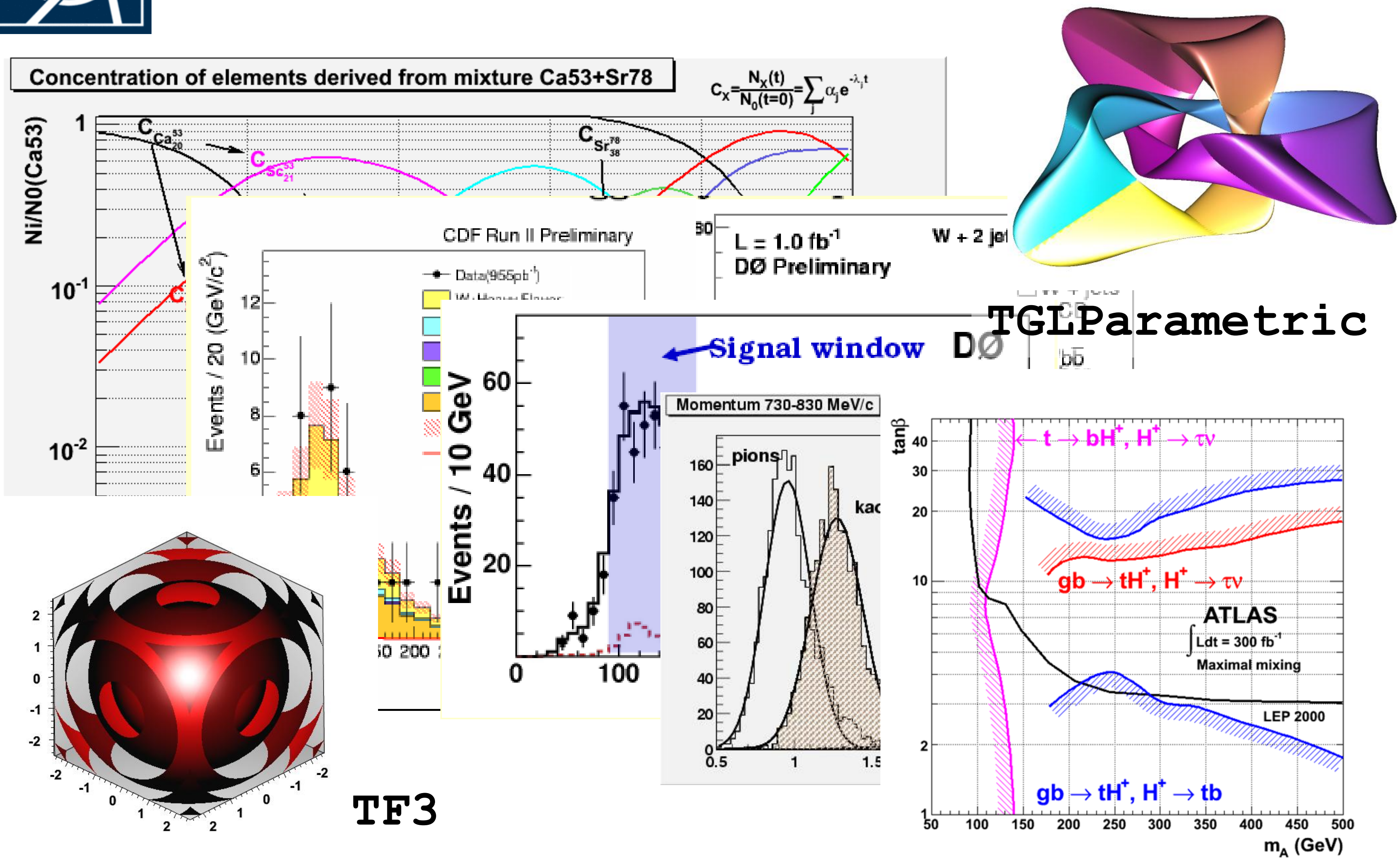
$ROOTSYS/tutorials/graphics/feynman.C



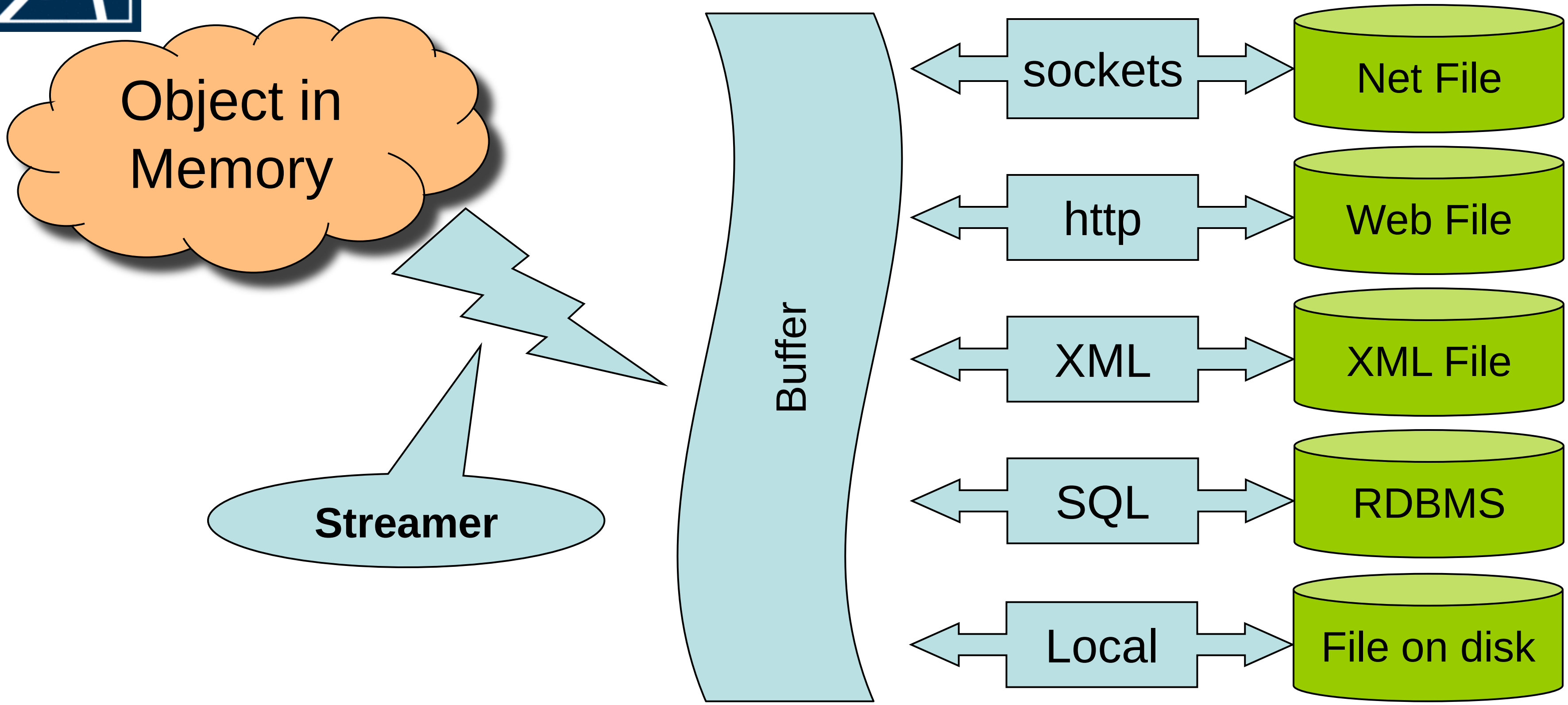
TCurlyArc
TCurlyLine
TWavyLine
and other building blocks for Feynmann diagrams

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

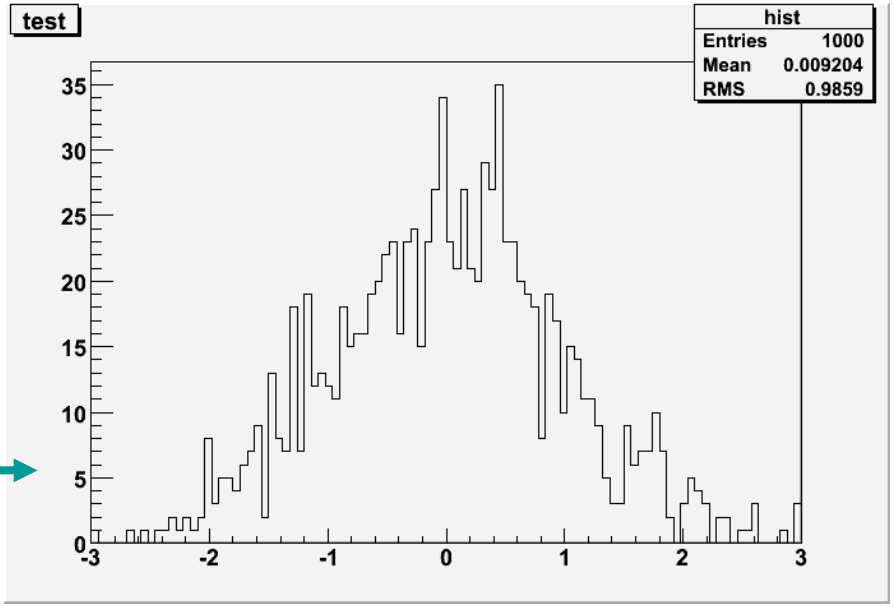
- TFile is the class to access files on your file system (and elsewhere)
- A TFile object may contain directories (TDirectory), like a Unix file system
- ROOT files are self describing
  - Dictionary for persistent classes written to the file
- Support for Backward and Forward compatibility
- Files created in 2001 must be readable in 2015

File Example

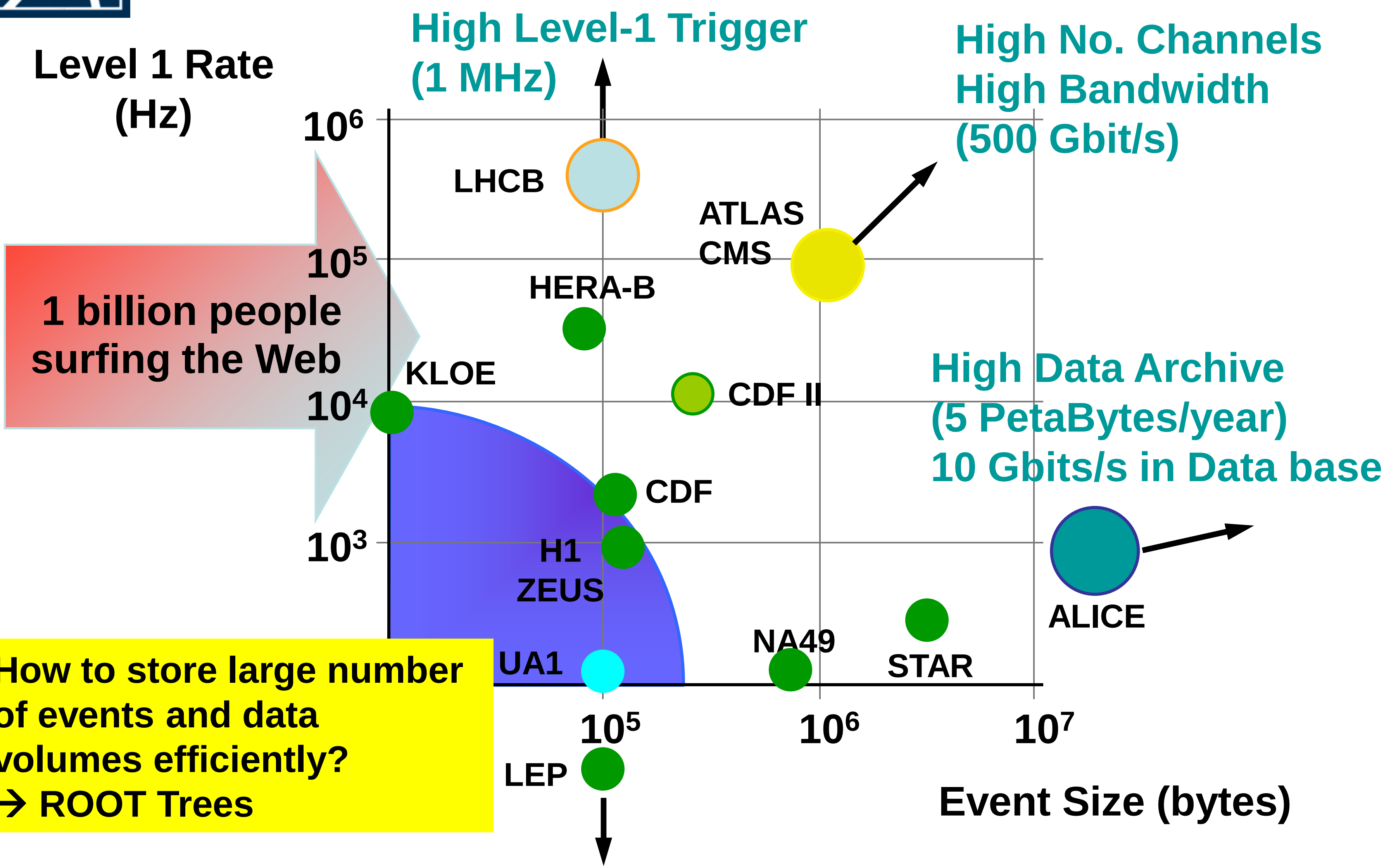
```
void keywrite() {
   TFile f("file.root", "new");
   TH1F h("hist", "test", 100, -3, 3);
   h.FillRandom("gaus", 1000);
   h.Write()
}
```

```
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?

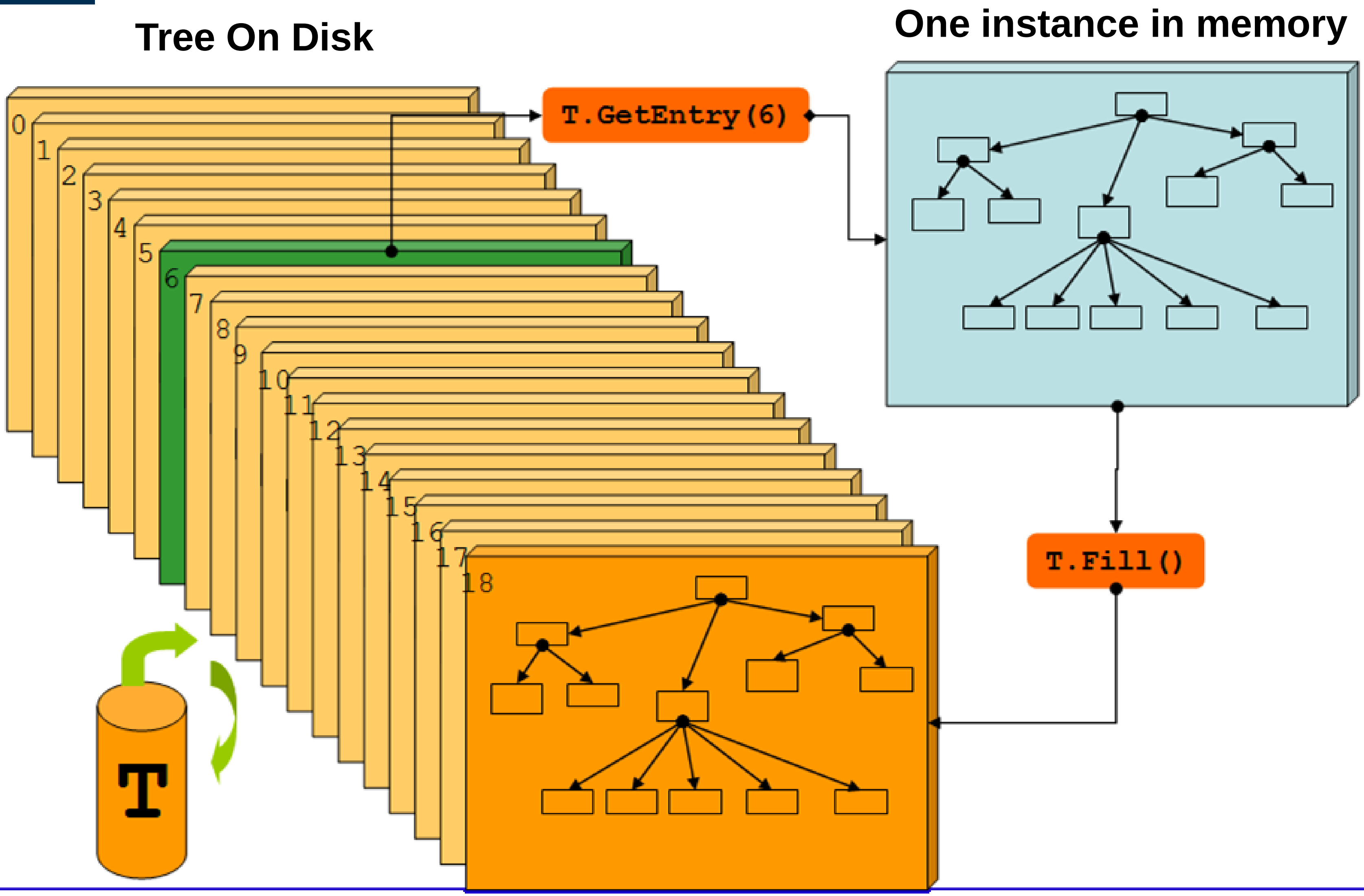


How to store large number of events and data volumes efficiently?
→ ROOT Trees

What is a ROOT Tree?

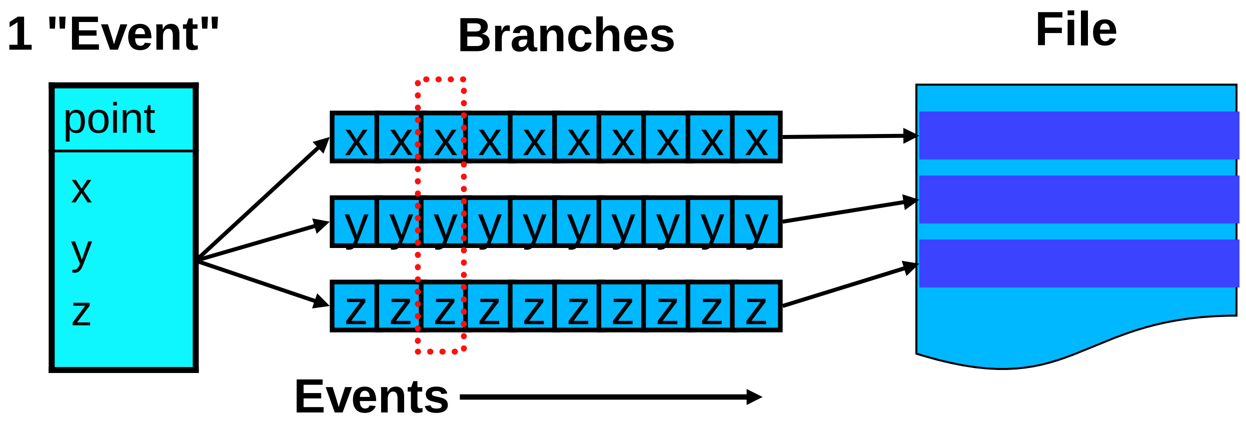
- Trees have been designed to support very large collections of objects. The overhead in memory is in general less than 4 bytes per entry.
- Trees allow direct and random access to any entry (sequential access is the most efficient)
- Trees are structured into branches and leaves. One can read a subset 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



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



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);
```

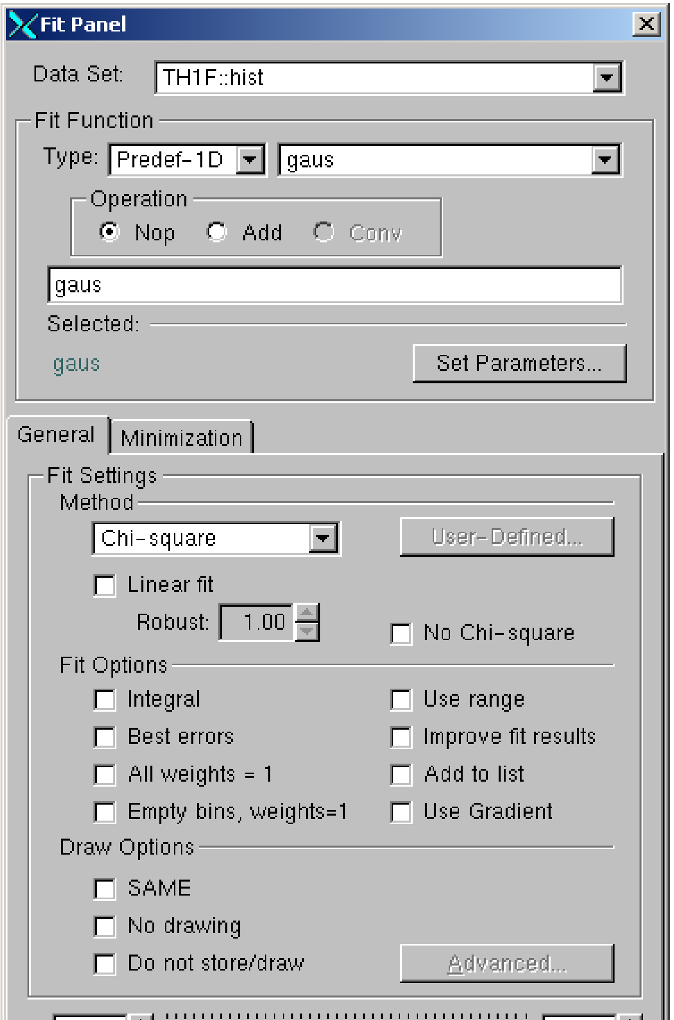
- Read entries from the tree and use the content of the class

```
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
```

ROOT is MORE….

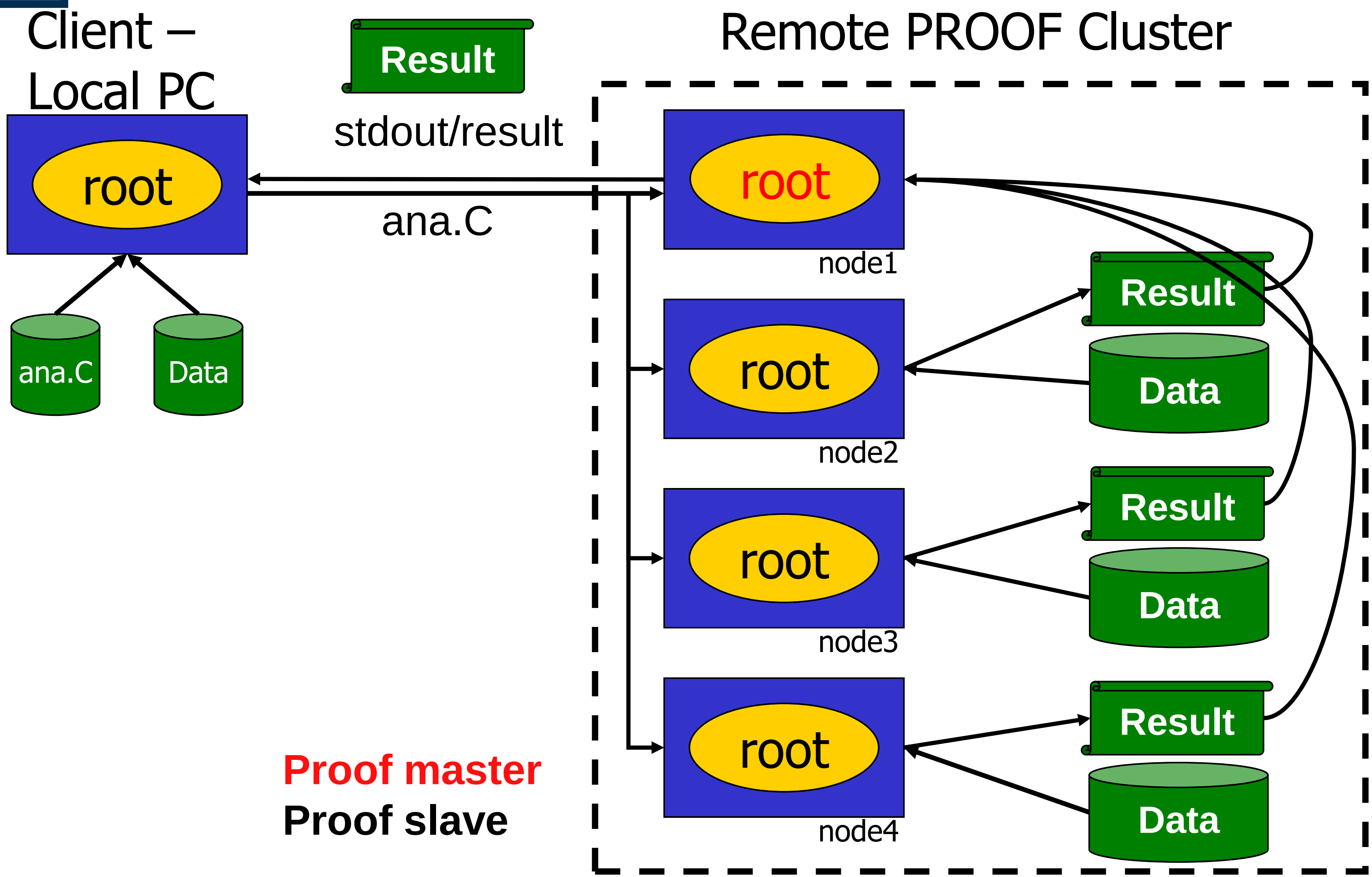
- In this talk, I presented the most basic classes typically used during physics analyses
- ROOT contains many more libraries, e.g.
  - FFT library
  - Oracle, MySQL, etc interfaces
  - XML drivers
  - TMVA (Multi Variate Analysis)
  - GRID, networking and thread classes
  - Interfaces to Castor, Dcache,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

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



More Information...

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

The second lecture will show many of the presented features in a demo

Credits to Fons Rademakers