



Introduction to ROOT

Summer Students Lecture
6 July 2009

Fons Rademakers (PH/SFT)
Jan Fiete Grosse-Oetringhaus (PH/AIP)

<http://root.cern.ch>

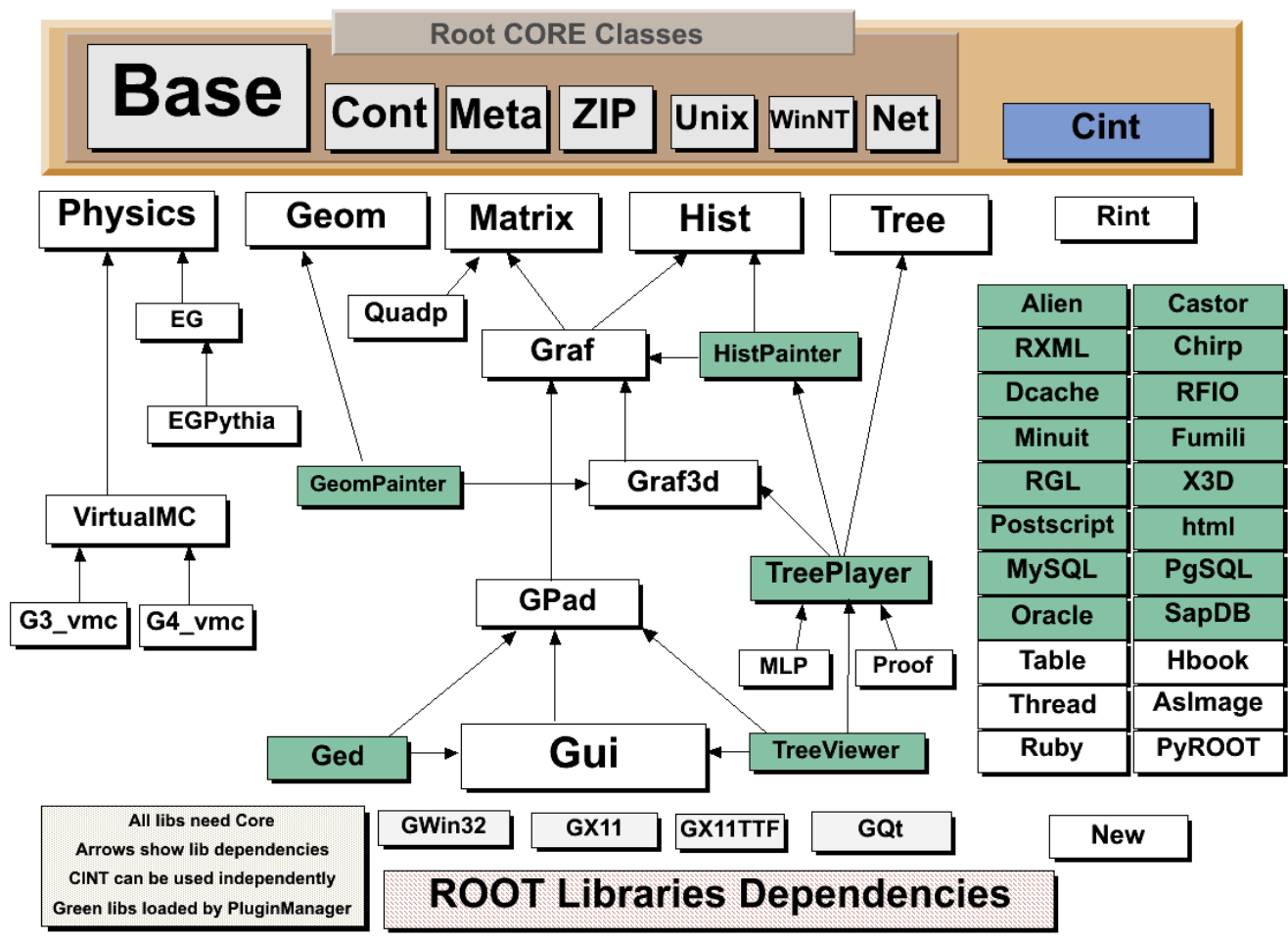


ROOT in a Nutshell

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



The ROOT Libraries



- Over 1500 classes
- 2,000,000 lines of code
- CORE (8 Mbytes)
- CINT (2 Mbytes)
- Green libraries linked on demand via plug-in manager (only a subset shown)
- 100 shared libs

ROOT: An Open Source Project

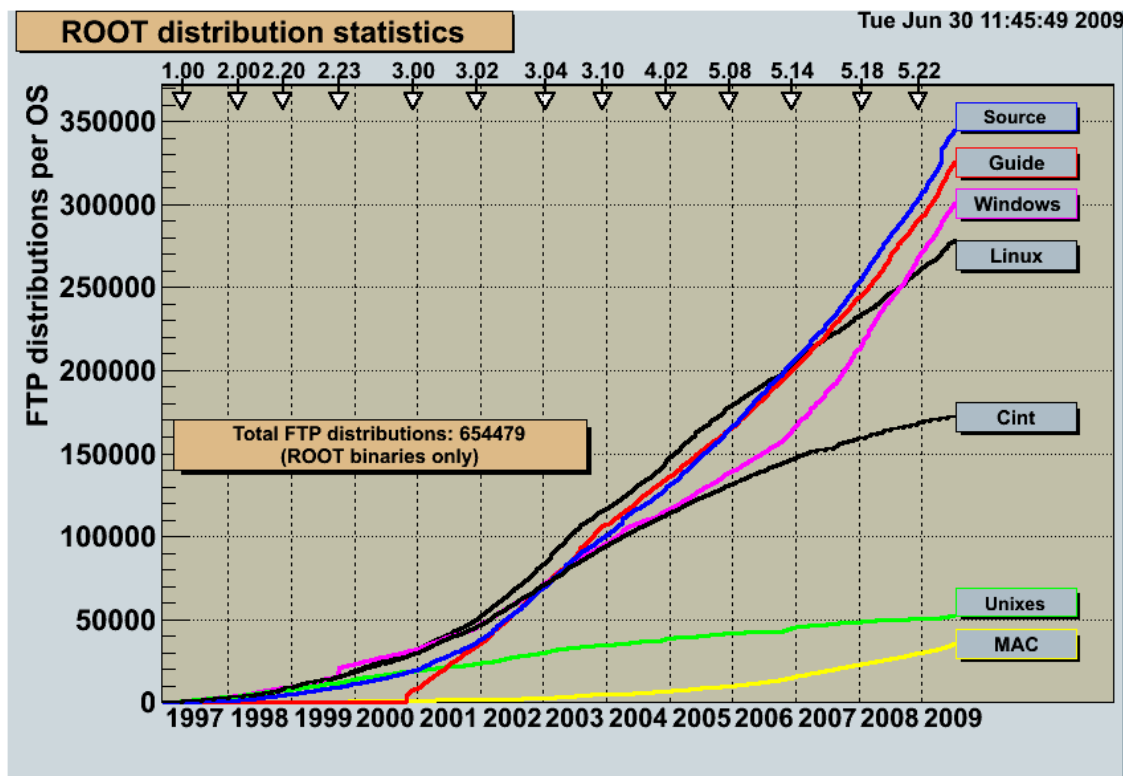


- The project was started in Jan 1995
- First release Nov 1995
- The project is developed as a collaboration between:
 - Full time developers:
 - 8 people full time at CERN (PH/SFT)
 - 2 developers at Fermilab/USA
 - Large number of part-time contributors (160 in CREDITS file)
 - A long list of users giving feedback, comments, bug fixes and many small contributions
 - 3870 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



ROOT Stats

- ROOT binaries have been downloaded more than 650000 times since 1997
- The estimated user base is about 20000 people



ROOT: a Framework and a Library



■ User classes

- User can define new classes interactively
- Either using calling API or sub-classing API
- These classes can inherit from ROOT classes

This is the normal operation mode

■ Dynamic linking

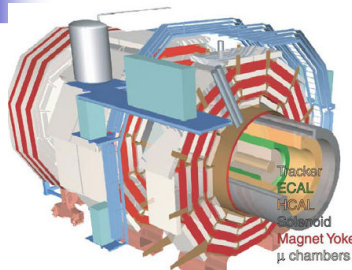
- Interpreted code can call compiled code
- Compiled code can call interpreted code
- Macros can be dynamically compiled & linked

Interesting feature for GUIs & event displays

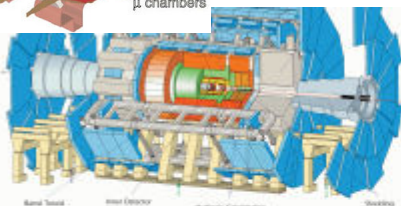
Script Compiler
root > `.x file.C++`



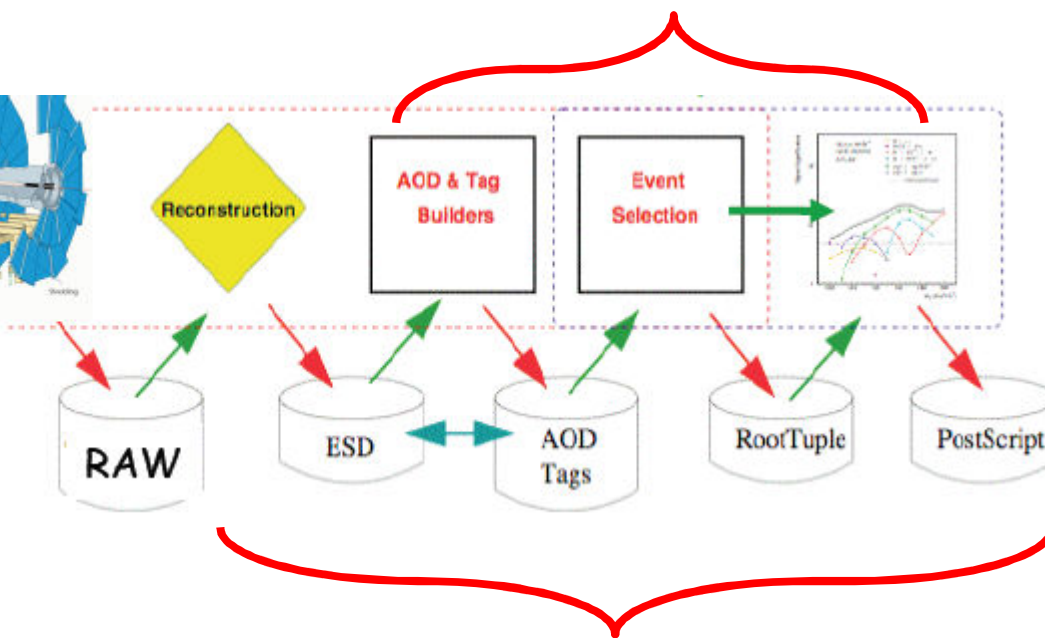
ROOT Application Domains



Data Analysis & Visualization

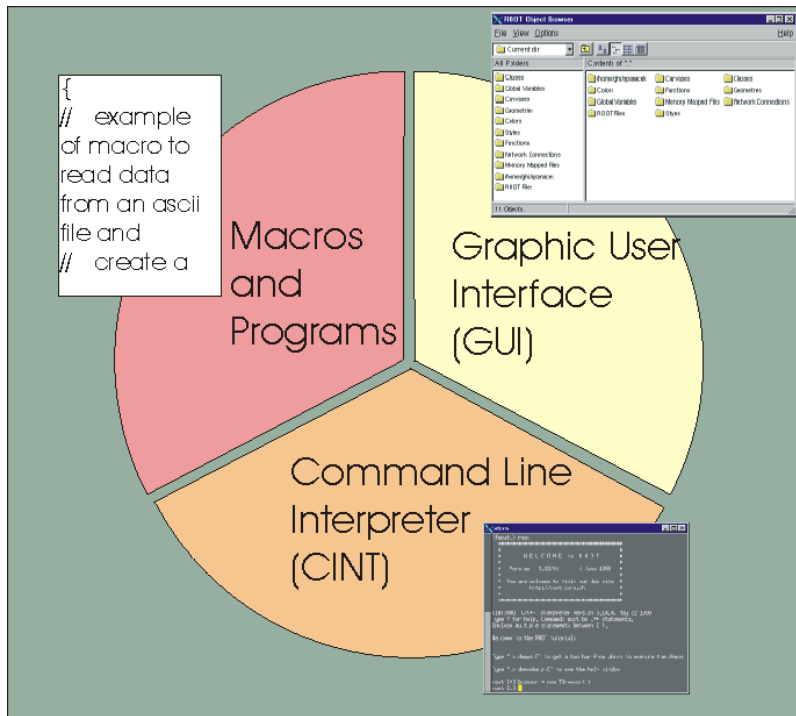


General Framework



Data Storage: Local, Network

Three User Interfaces



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



CINT Interpreter



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



Compiled versus 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.

- ❑ Are Makefiles dead?
 - ! No! if you build/compile a very large application
 - ! Yes! ACLiC is even platform independent!



Running Code

To run function mycode() in file mycode.C:

```
root [0] .x mycode.C
```

Equivalent: load file and run function:

```
root [1] .L mycode.C
```

```
root [2] mycode ()
```

All of CINT's commands (help):

```
root [3] .h
```



Running Code

Macro: file that is interpreted by CINT (**.x**)

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

Execute with **.x mymacro.C(42)**



Unnamed Macros

No functions, just statements

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

Execute with **.x mymacro.C**

No functions thus no arguments

Named macro recommended!

Compiler prefers it, too...



Running Code – Libraries

"Library": compiled code, shared library

CINT can call its functions!

Building a library from a macro: ACLiC

(**A**utomatic **C**ompiler of **L**ibraries for **C**INT)

```
.x mymacro.C(42) +
```

Use "+" instead of writing a Makefile...

CINT knows all functions in mymacro_C.so/.dll

```
mymacro(42)
```

My First Session



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
```

root

See file \$HOME/.root_hist

root [0] try up and down arrows



My Second Session

root

```
root [0] .x session2.C
for N=100000, sum= 45908.6
root [1] sum
(double)4.59085828512453370e+004
Root [2] r.Rndm()
(Double_t)8.29029321670533560e-001
root [3] .q
```

unnamed macro
executes in global scope

session2.C

```
{
    int N = 100000;
    TRandom r;
    double sum = 0;
    for (int i=0;i<N;i++) {
        sum += sin(r.Rndm());
    }
    printf("for N=%d, sum= %g\n",N,sum);
}
```



My Third Session

root

```
root [0] .x session3.C
for N=100000, sum= 45908.6
root [1] sum
Error: Symbol sum is not defined in current scope
*** Interpreter error recovered ***
Root [2] .x session3.C(1000)
for N=1000, sum= 460.311
root [3] .q
```

session3.C

```
void session3 (int N=100000) {
    TRandom r;
    double sum = 0;
    for (int i=0;i<N;i++) {
        sum += sin(r.Rndm());
    }
    printf("for N=%d, sum= %g\n",N,sum);
}
```

Named macro
Normal C++ scope rules

My Third Session with ACLIC



```
root [0] gROOT->Time();
root [1] .x session4.C(10000000)
for N=10000000, sum= 4.59765e+006
Real time 0:00:06, CP time 6.890
root [2] .x session4.C+(10000000)

for N=10000000, sum= 4.59765e+006
Real time 0:00:09, CP time 1.062
root [3] session4(10000000)
for N=10000000, sum= 4.59765e+006
Real time 0:00:01, CP time 1.052
root [4] .q
```

session4.C

File session4.C
Automatically compiled
and linked by the
native compiler.
Must be C++ compliant

```
#include "TRandom.h"
void session4 (int N) {
    TRandom r;
    double sum = 0;
    for (int i=0;i<N;i++) {
        sum += sin(r.Rndm());
    }
    printf("for N=%d, sum= %g\n",N,sum);
}
```



Macros With More Than One Function

```
root [0] .x session5.C >session5.log
root [1] .q
```

```
root [0] .L session5.C
root [1] session5(100); >session5.log
root [2] session5b(3)
sum(0) = 0
sum(1) = 1
sum(2) = 3
root [3] .q
```

`.x session5.C`
executes the function
session5 in session5.C

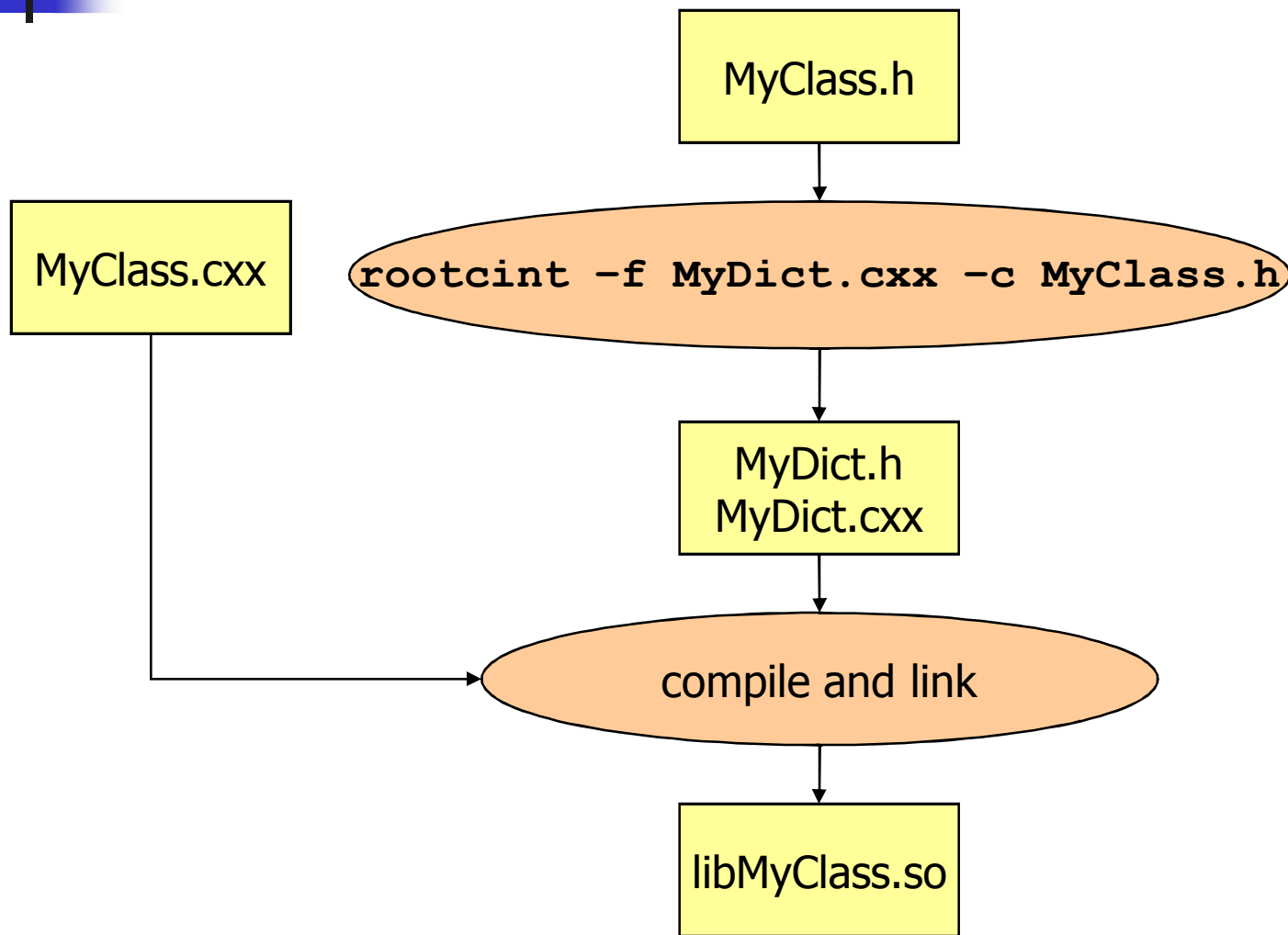
use `gROOT->ProcessLine`
to execute a macro from a
macro or from compiled
code

session5.C

```
void session5(int N=100) {
    session5a(N);
    session5b(N);
    gROOT->ProcessLine(".x session4.C+(1000)");
}
void session5a(int N) {
    for (int i=0;i<N;i++) {
        printf("sqrt(%d) = %g\n",i,sqrt(i));
    }
}
void session5b(int N) {
    double sum = 0;
    for (int i=0;i<N;i++) {
        sum += i;
        printf("sum(%d) = %g\n",i,sum);
    }
}
```



Generating a Dictionary





Graphics & GUI



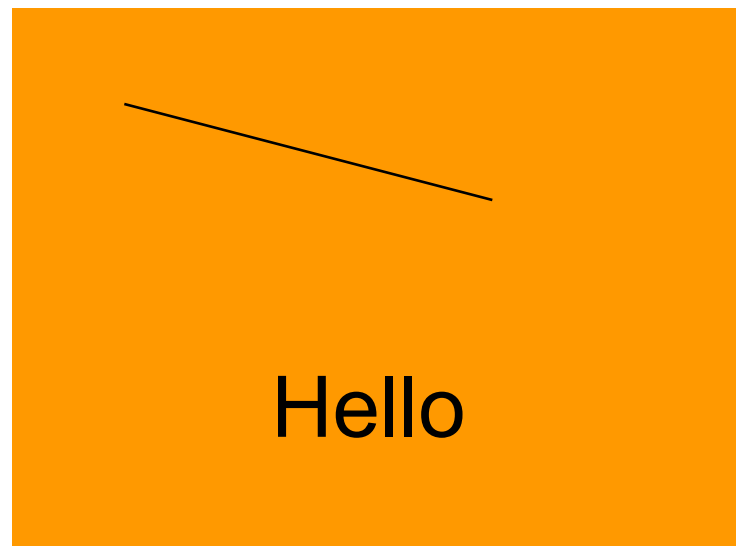
TPad: Main Graphics Container

```
Root > TLine line(.1,.9,.6,.6)
```

```
Root > line.Draw()
```

```
Root > TText text(.5,.2,"Hello")
```

```
Root > text.Draw()
```



The **Draw** function adds the object to the list of primitives of the current pad.

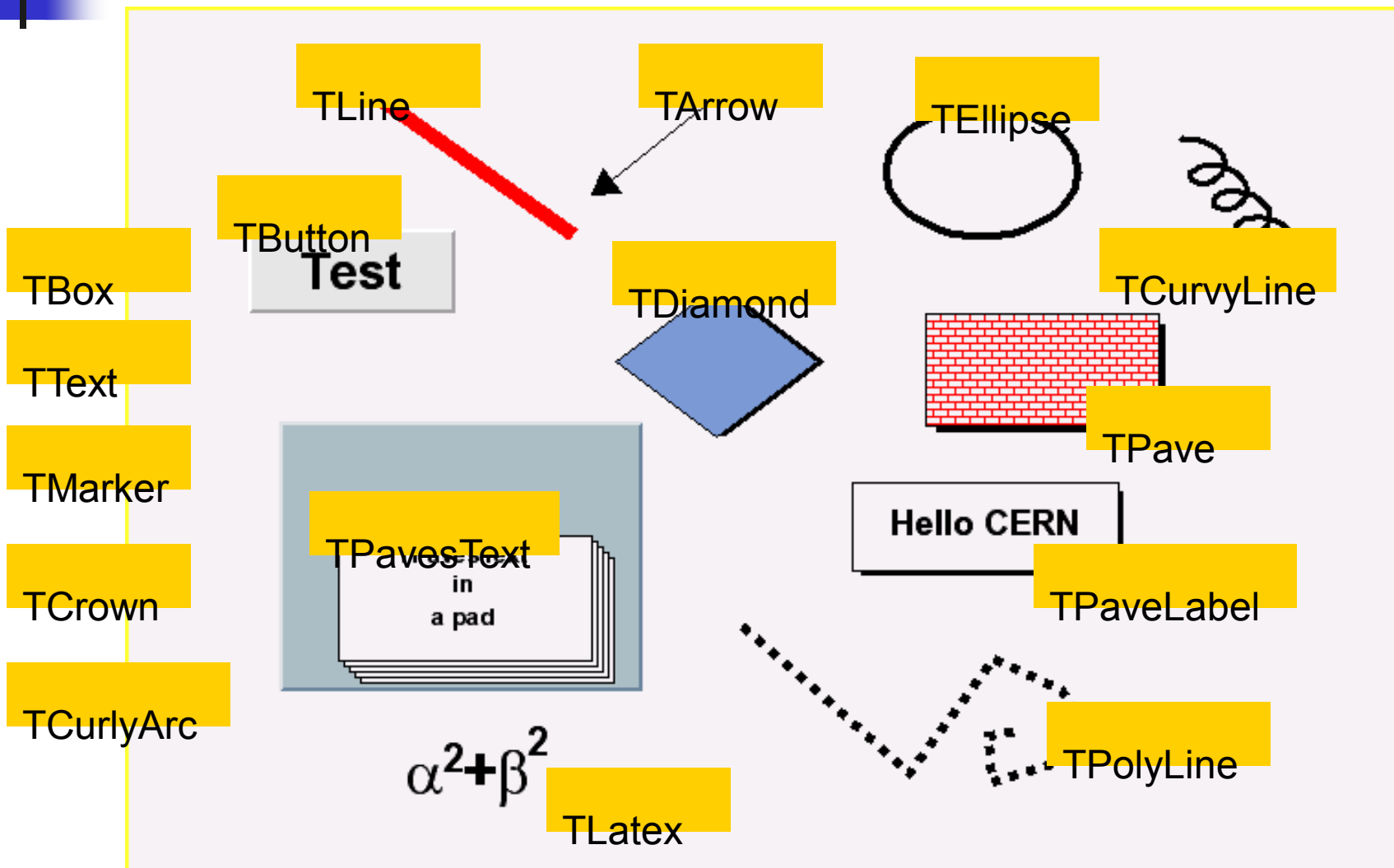
If no pad exists, a pad is automatically created with a default range [0,1].

When the pad needs to be drawn or redrawn, the object **Paint** function is called.

Only objects deriving
from TObject may be drawn
in a pad
ROOT Objects or User objects



Basic Primitives



Full LaTeX
support
on screen
and
postscript

latex3.C

Born equation

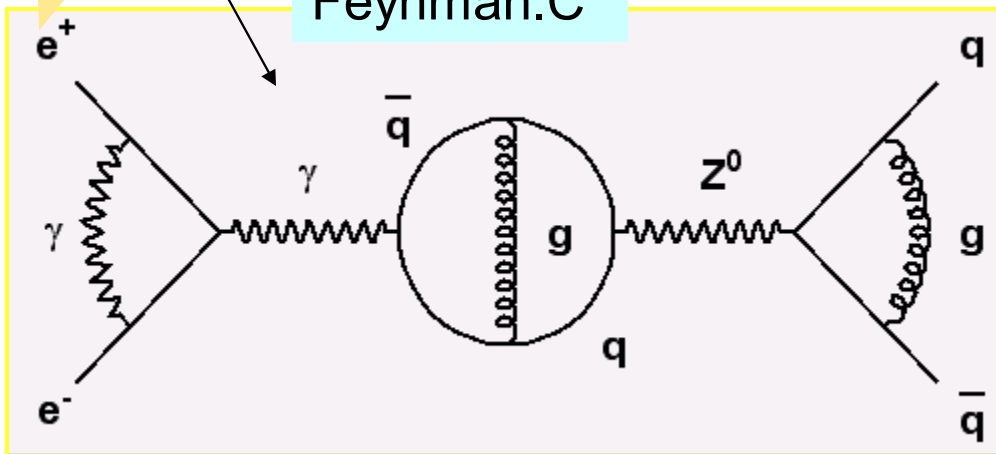
$$\frac{2s}{\pi\alpha^2} \frac{d\sigma}{d\cos\theta} (e^+ e^- \rightarrow f\bar{f}) = \left| \frac{1}{1-\Delta\alpha} \right|^2 (1+\cos^2\theta)$$

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

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

Formula or
diagrams
can be
edited with
the mouse

Feynman.C



TCurlyArc
TCurlyLine
TWavyLine
and other building
blocks for
Feynmann diagrams

Graphs

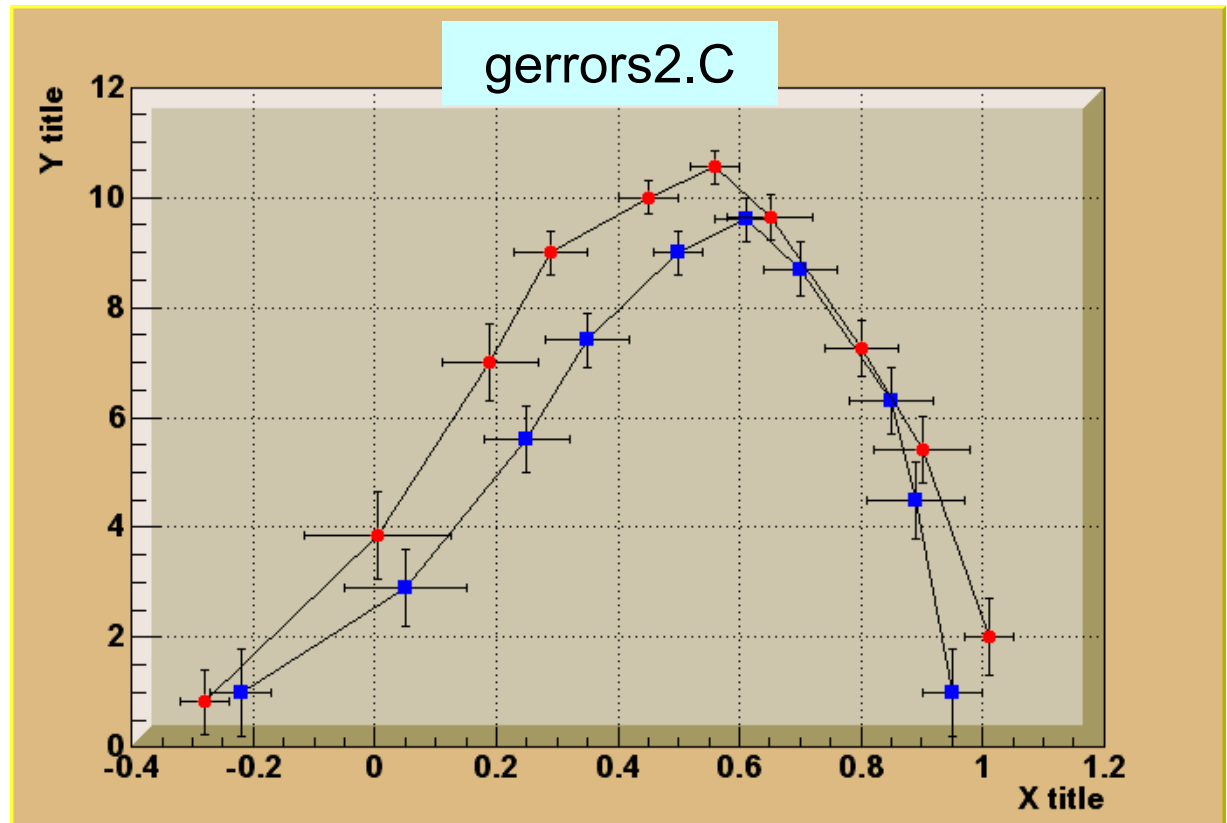


TGraphErrors(n,x,y,ex,ey)

TGraph(n,x,y)

TCutG(n,x,y)

TMultiGraph

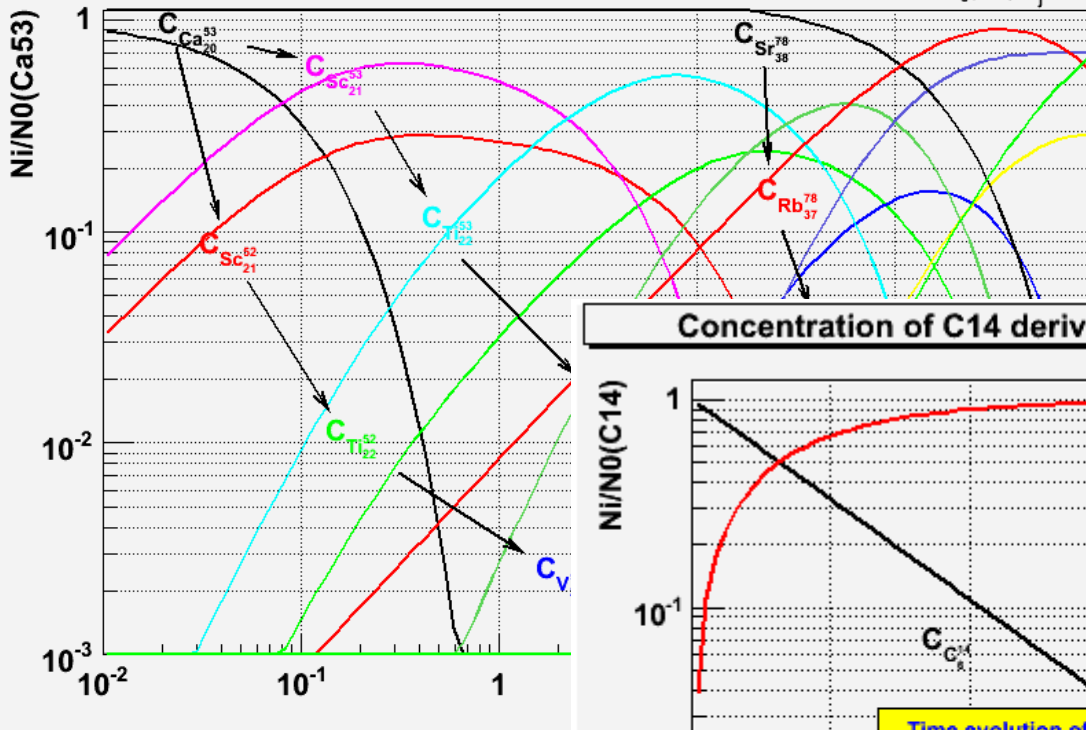


TGraphAsymmErrors(n,x,y,exl,exh,eyl,eyh)

Graphics Examples

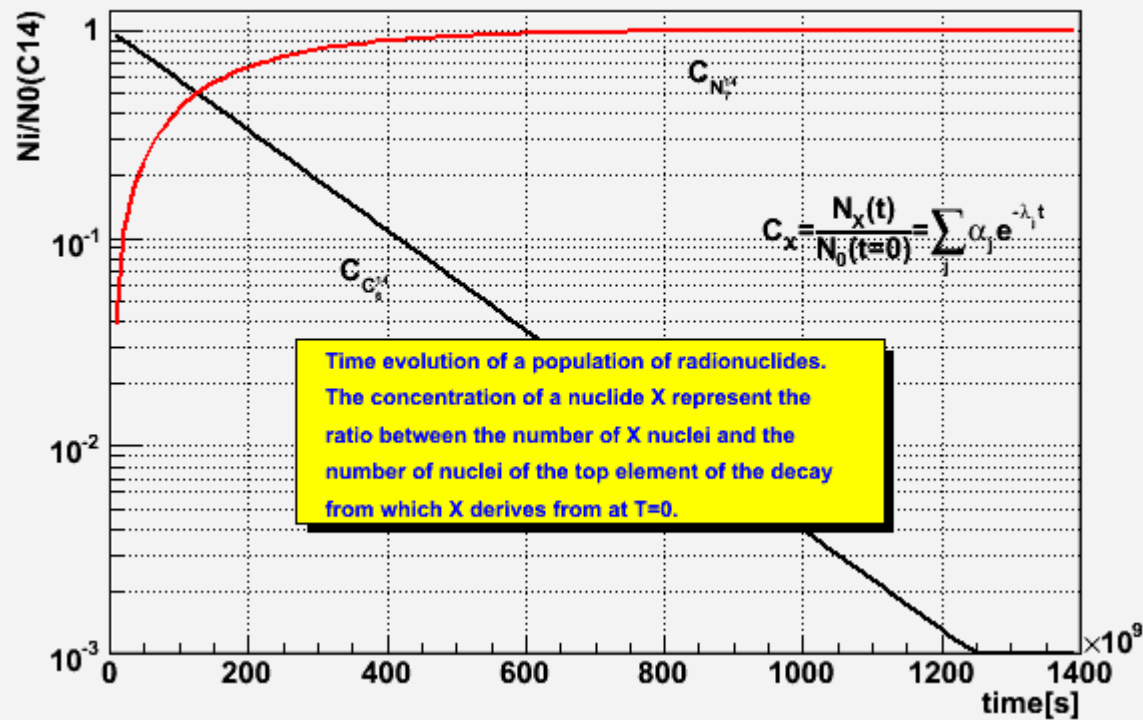


Concentration of elements derived from mixture Ca53+Sr78



$$C_x = \frac{N_x(t)}{N_0(t=0)} = \sum_j \alpha_j e^{-\lambda_j t}$$

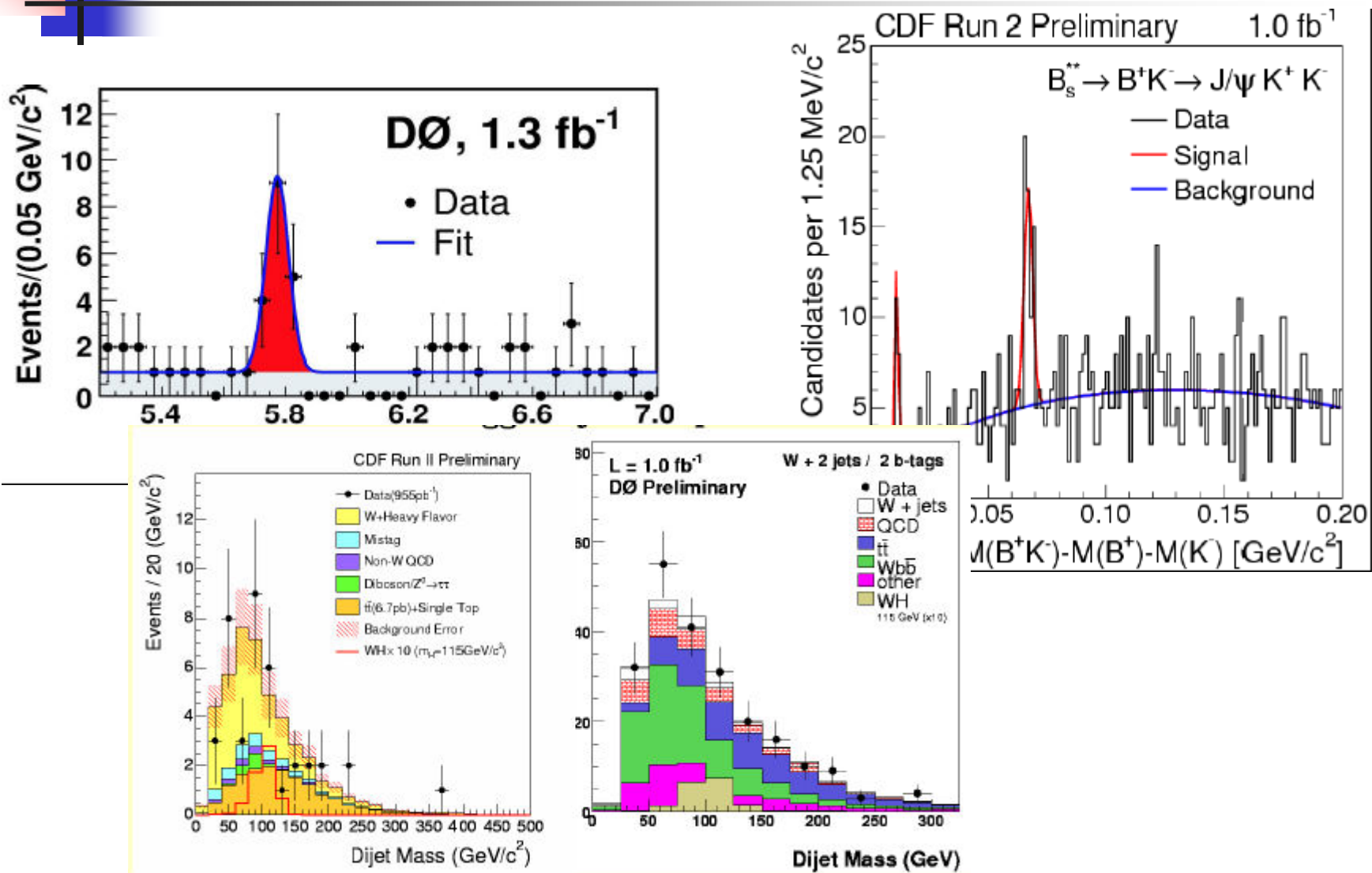
Concentration of C14 derived elements



$$C_x = \frac{N_x(t)}{N_0(t=0)} = \sum_j \alpha_j e^{-\lambda_j t}$$

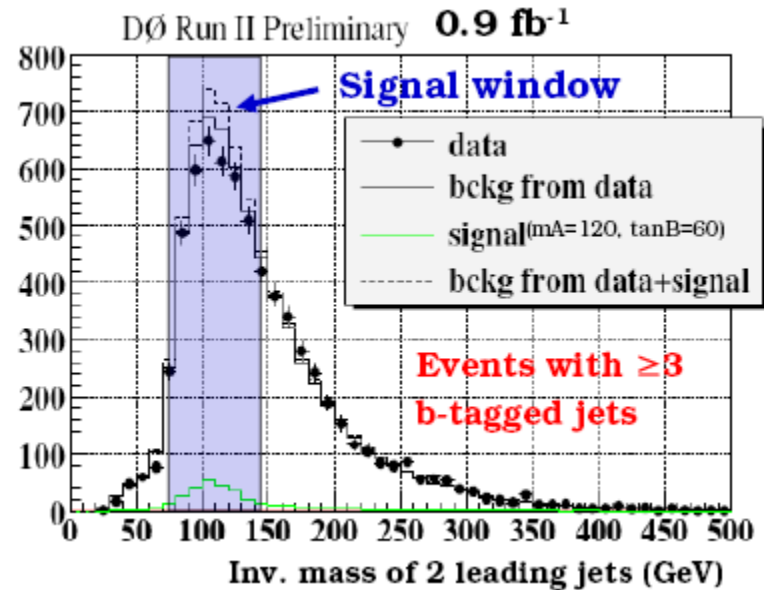
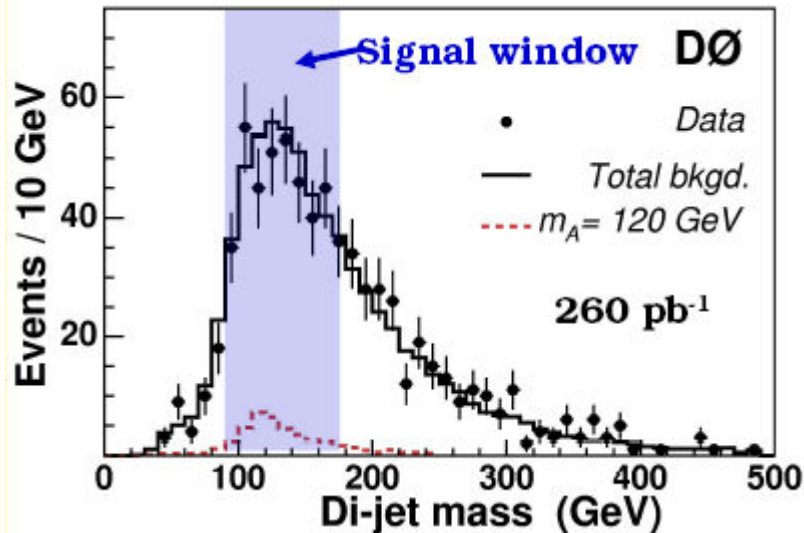
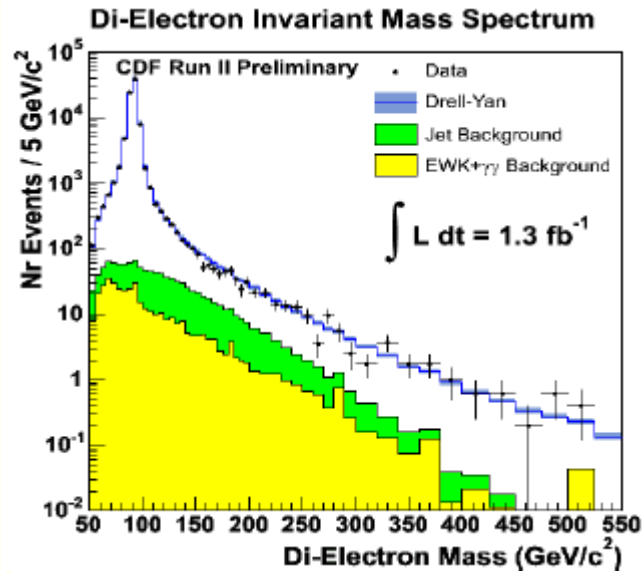
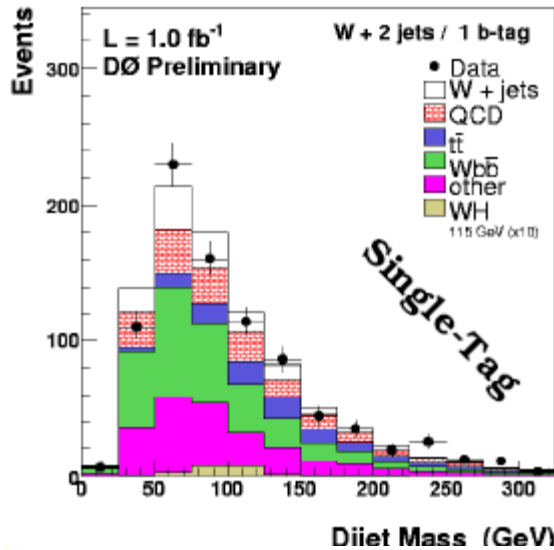


More Graphics Examples

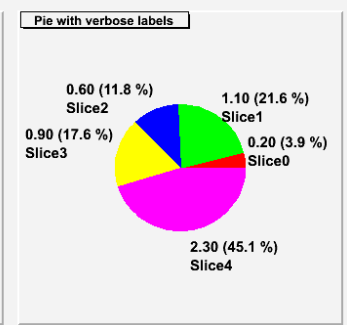
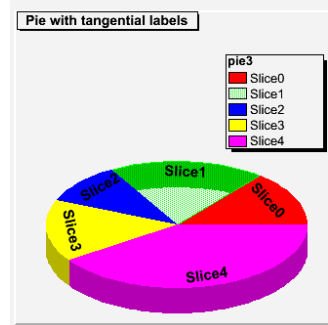
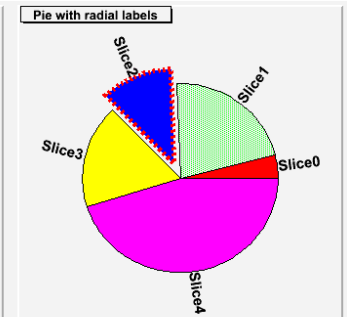
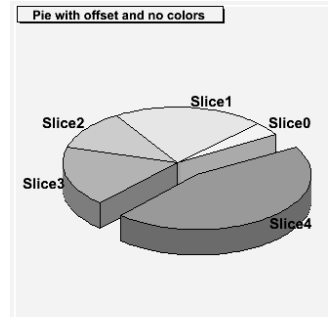
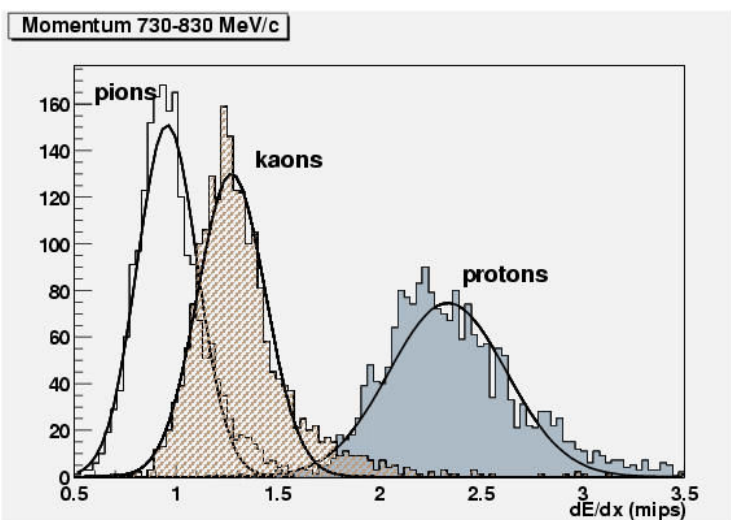
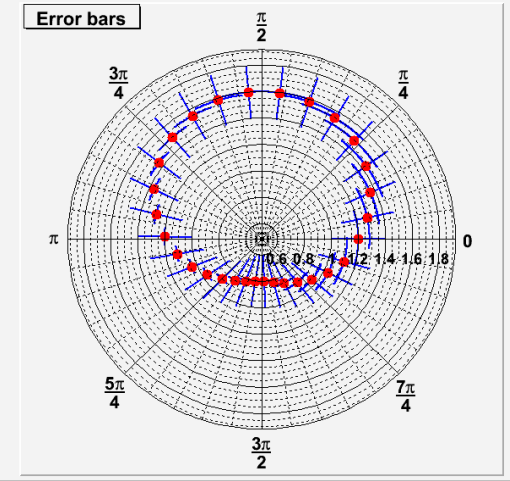
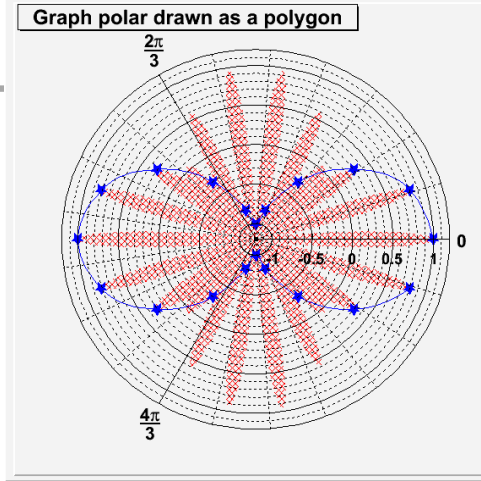
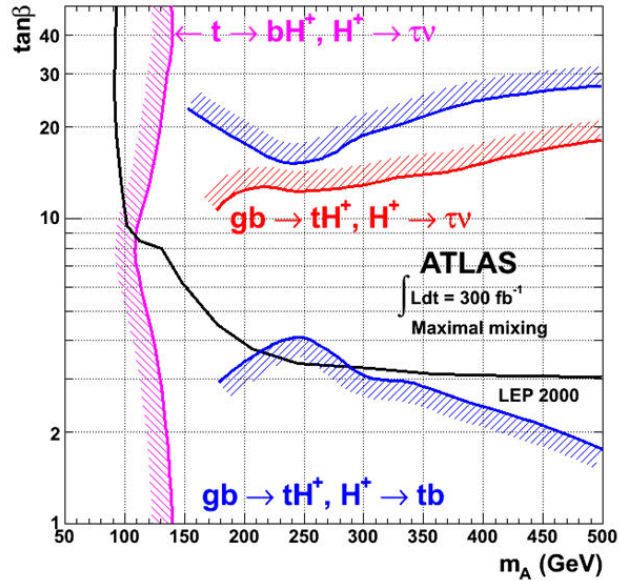


Even M

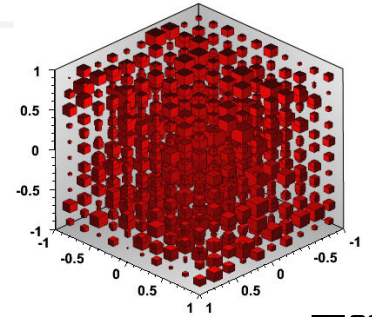
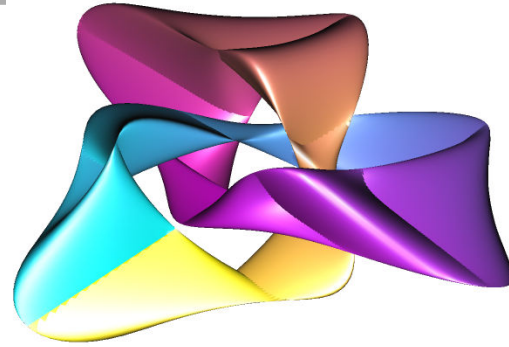
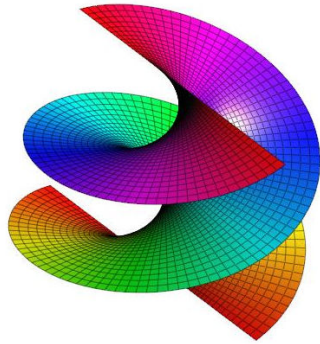
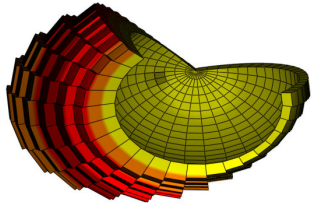
Examples



Special Graphics

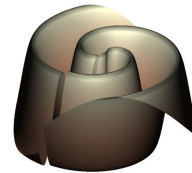
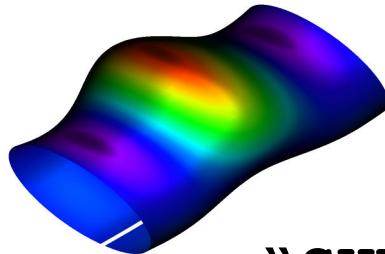
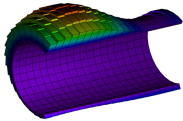
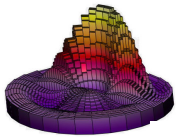
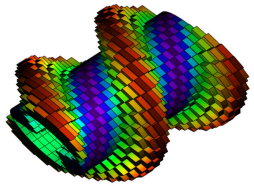
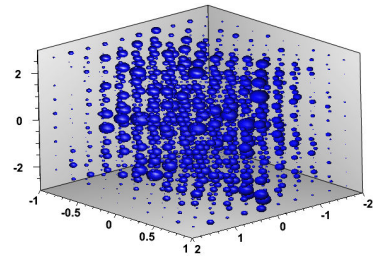


Graphics (2D-3D)

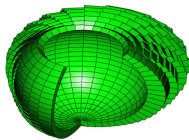
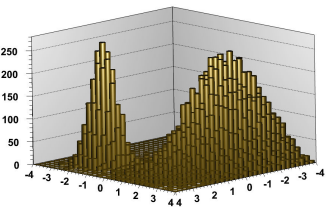


TH3

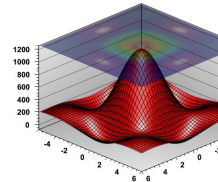
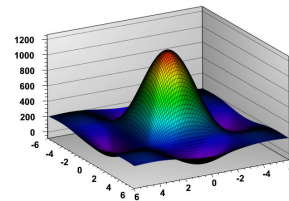
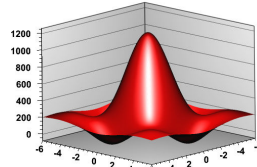
TGLParametric



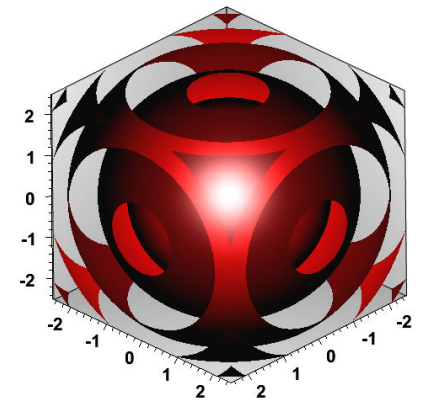
"LEGO"



"SURF"



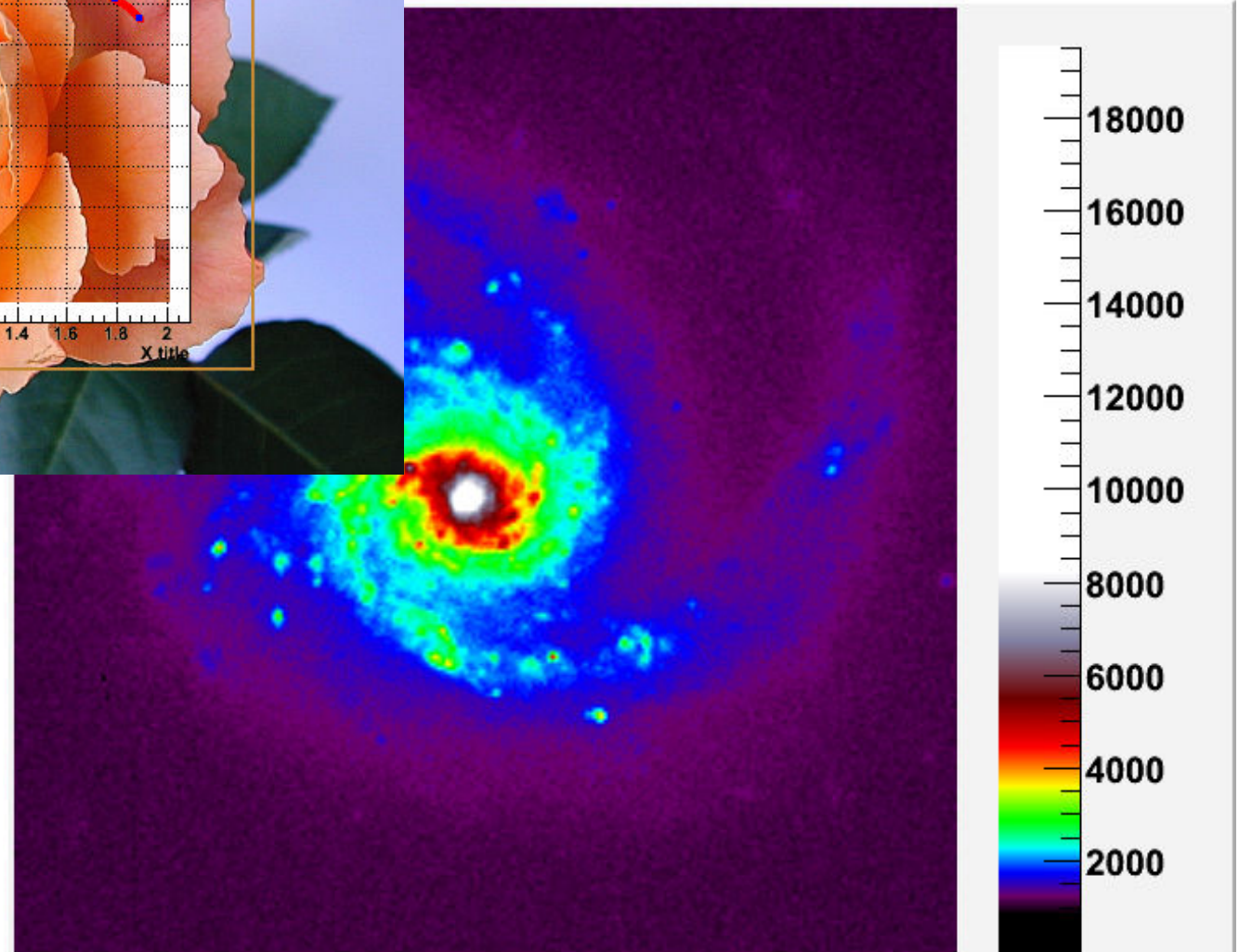
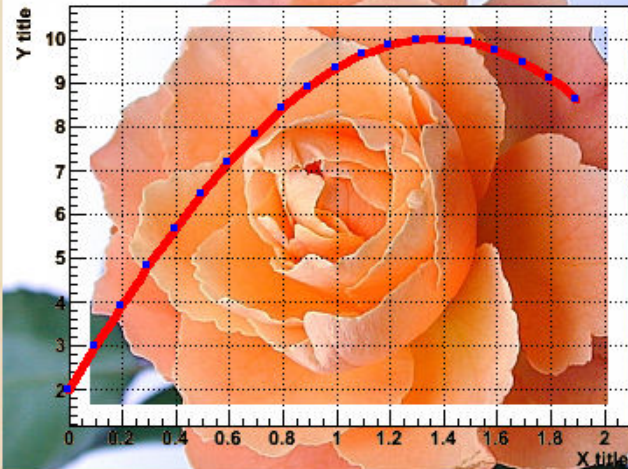
TF3



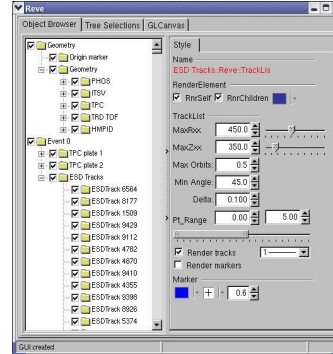
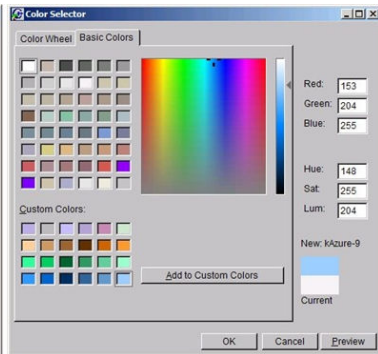
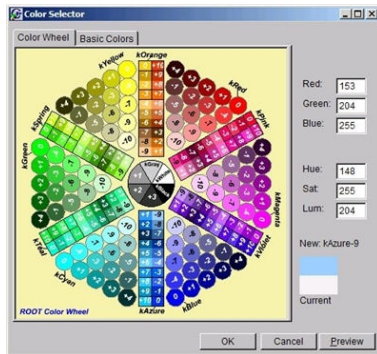
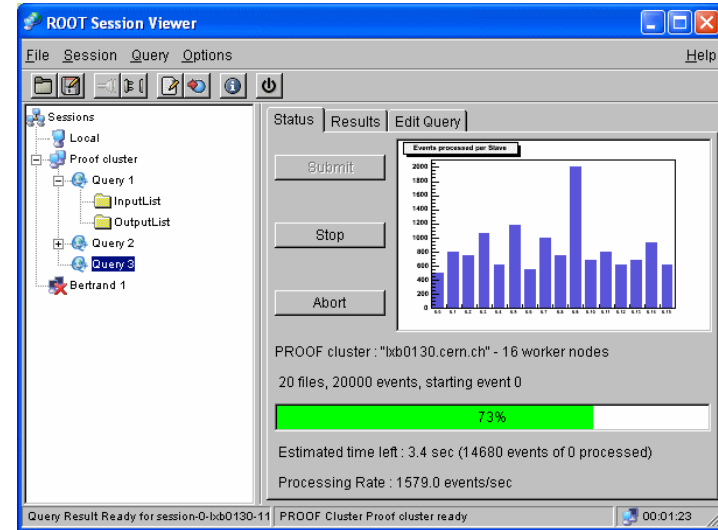
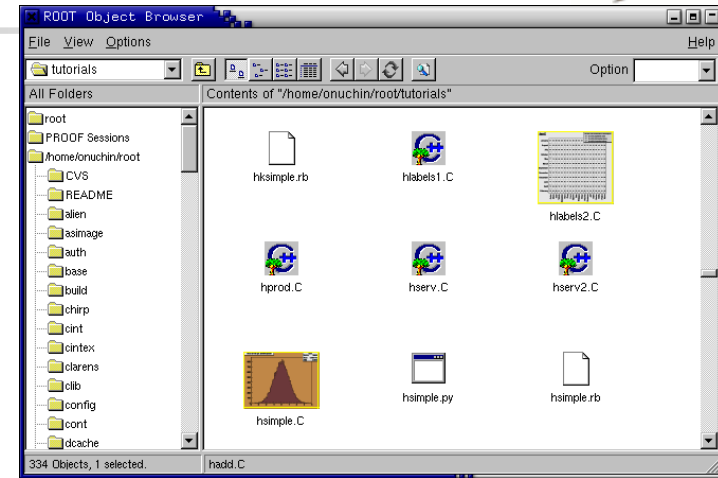
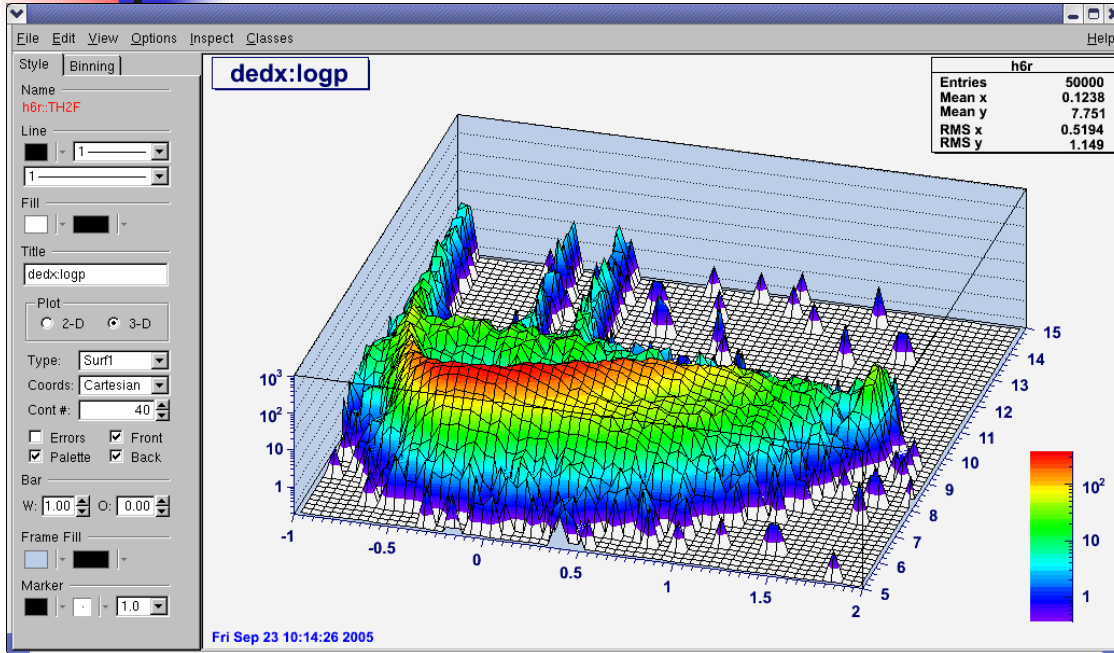
ASImage: Image processor



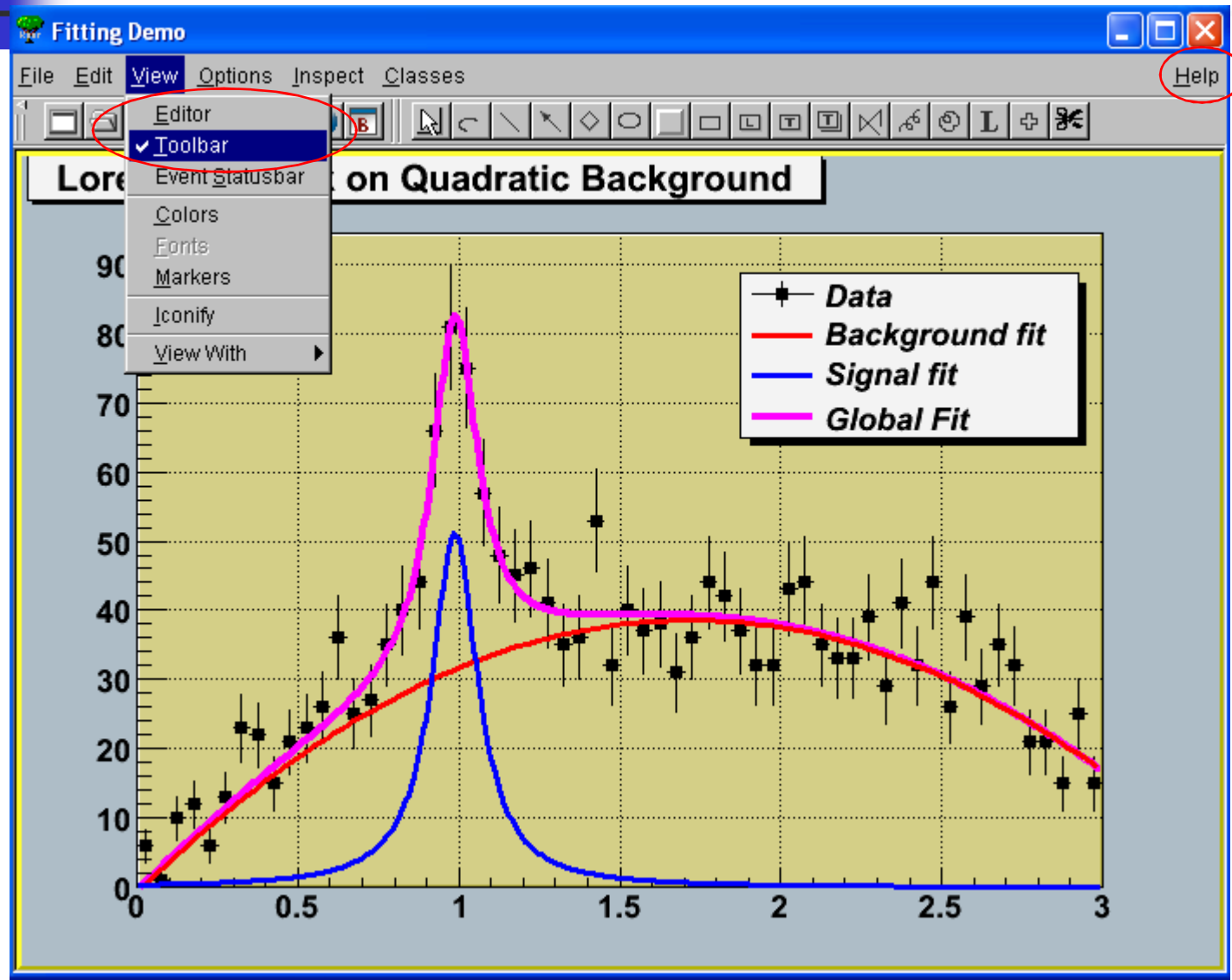
a simple graph



GUI (Graphical User Interface)

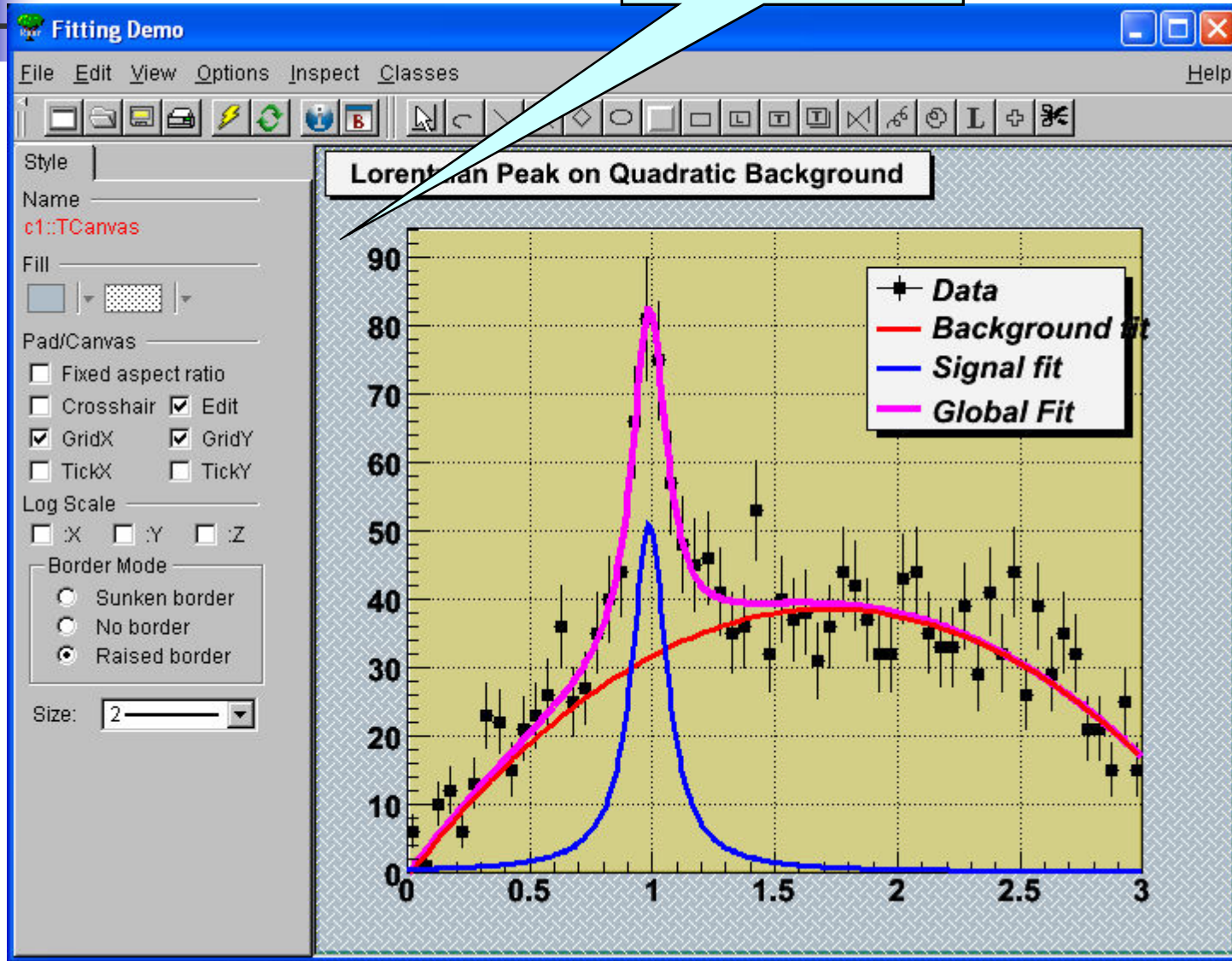


Canvas tool bar/menus/help



Object Editor

Click on any object to show its editor



ROOT Browser



The screenshot shows the TGRootIDE web browser interface. The title bar reads "Untitled - TGRootIDE". The menu bar includes "File", "Edit", "Search", and "Tools". The toolbar contains various icons for file operations and navigation. The address bar shows "http://root.cern.ch". The main content area displays the ROOT website with the following elements:

- Navigation Links:** Roadmap, Mission Statement, Architecture, Main Features, CINT, Coding Conventions, Benchmarking, Picture Gallery, Publication List, The ROOT Team.
- Additional Links:** License, Register as User, Download Binaries, Install from Source, CVS, ViewCVS, LXR, Nightlies.
- User Resources:** User's Guide, Reference Guide, Tutorials, HOWTO's, RootTalk Forum, RootTalk Digest, Example Applications, BaBar Tutorials, FNAL Tutorials, MINOS Tutorials.
- Special Notices:**
 - ROOT 2007 Workshop video archives NEW**
Thanks to **Steven Goldfarb** and **Mitch McLachlan** from ATLAS, you can [view or download the video archives](#).
 - Production Version 5.16/00 NEW 29/06/2007**
The Production release of ROOT 5.16/00 is now available.
 - The CVS tag for this version is **v5-16-00**.

The left sidebar shows a file tree for "bme1brun/root/tutorials/fit", with the "HTML" folder highlighted. The status bar at the bottom indicates "Ln 0, Ch 0".

ROOT Browser



Untitled - TGRootIDE

File Edit Search Tools Help

Command (local):

HTML Untitled FittingDemo.C Execute Macro

```
//Example for fitting signal/background. This example can be executed with
// root > .x FittingDemo.C (using the CINT interpreter)
// root > .x FittingDemo.C+ (using the native compiler via ACLIC)
//Author: Rene Brun

#include "TH1.h"
#include "TMath.h"
#include "TF1.h"
#include "TLegend.h"
#include "TCanvas.h"

// Quadratic background function
Double_t background(Double_t *x, Double_t *par) {
    return par[0] + par[1]*x[0] + par[2]*x[0]*x[0];
}

// Lorentzian Peak function
Double_t lorentzianPeak(Double_t *x, Double_t *par) {
    return (0.5*par[0]*par[1]/TMath::Pi() /
           TMath::Max( 1.e-10, (x[0]-par[2])*(x[0]-par[2]) + .25*par[1]*par[1]));
}

// Sum of background and peak function
Double_t fitFunction(Double_t *x, Double_t *par) {
    return background(x,par) + lorentzianPeak(x,&par[3]);
}

void FittingDemo() {
    //Bevington Exercise by Peter Malzacher, modified by Rene Brun

    const int nBins = 60;

    Double_t data[nBins] = { 6, 1,10,12, 6,13,23,22,15,21,
                           23,26,36,25,27,35,40,44,66,81,
                           75,57,48,45,46,41,35,36,53,32
```

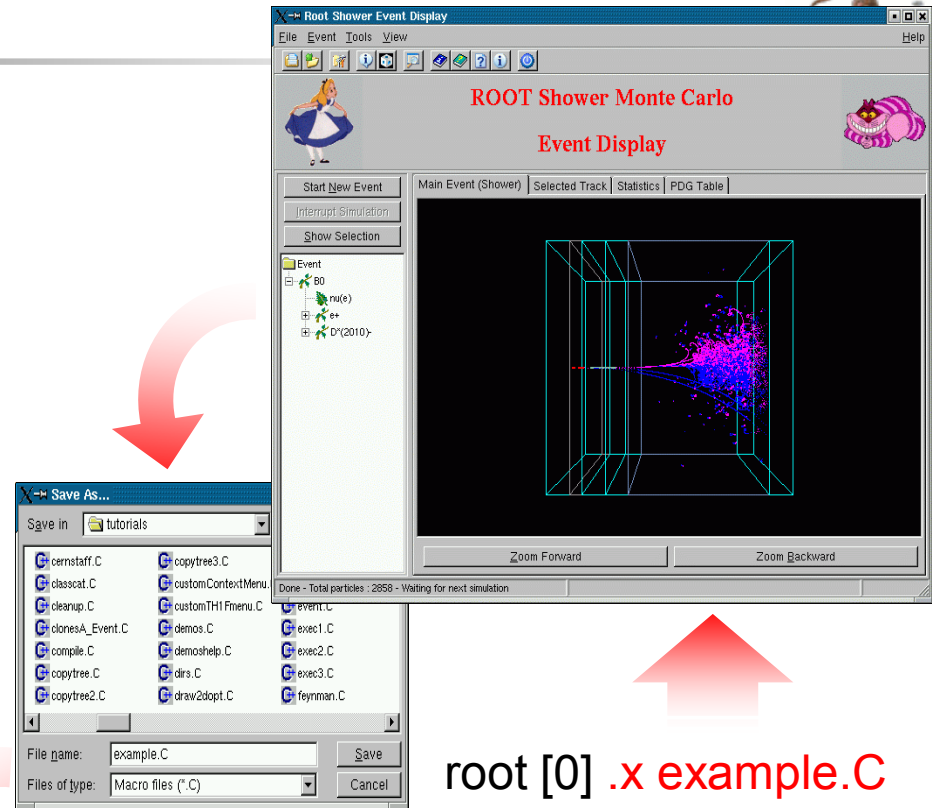
NO.	NAME	VALUE	ERROR	SIZE	DERIVATIVE
1	p0	-8.64649e-001	8.91776e-001	2.08428e-003	3.94624e-004
2	p1	4.58433e+001	2.64183e+000	1.52471e-003	6.75228e-004
3	p2	-1.33214e+001	9.76811e-001	6.23584e-004	1.77931e-003
4	p3	1.38074e+001	2.17651e+000	5.05446e-003	2.11742e-005
5	p4	1.72307e-001	3.58097e-002	9.43560e-005	-3.45333e-003
6	p5	9.87281e-001	1.12681e-002	4.24087e-005	-1.54945e-003

http://root.cern.ch/root/Benchmark.html Ln 0, Ch 0

GUI C++ Code Generator



- When pressing *ctrl+S* on any widget it is **saved as a C++ macro file** thanks to the *SavePrimitive* methods implemented in all GUI classes. The generated macro can be edited and then executed via CINT
- Executing the macro restores the complete original GUI as well as all created signal/slot connections in a global way



root [0] .x example.C

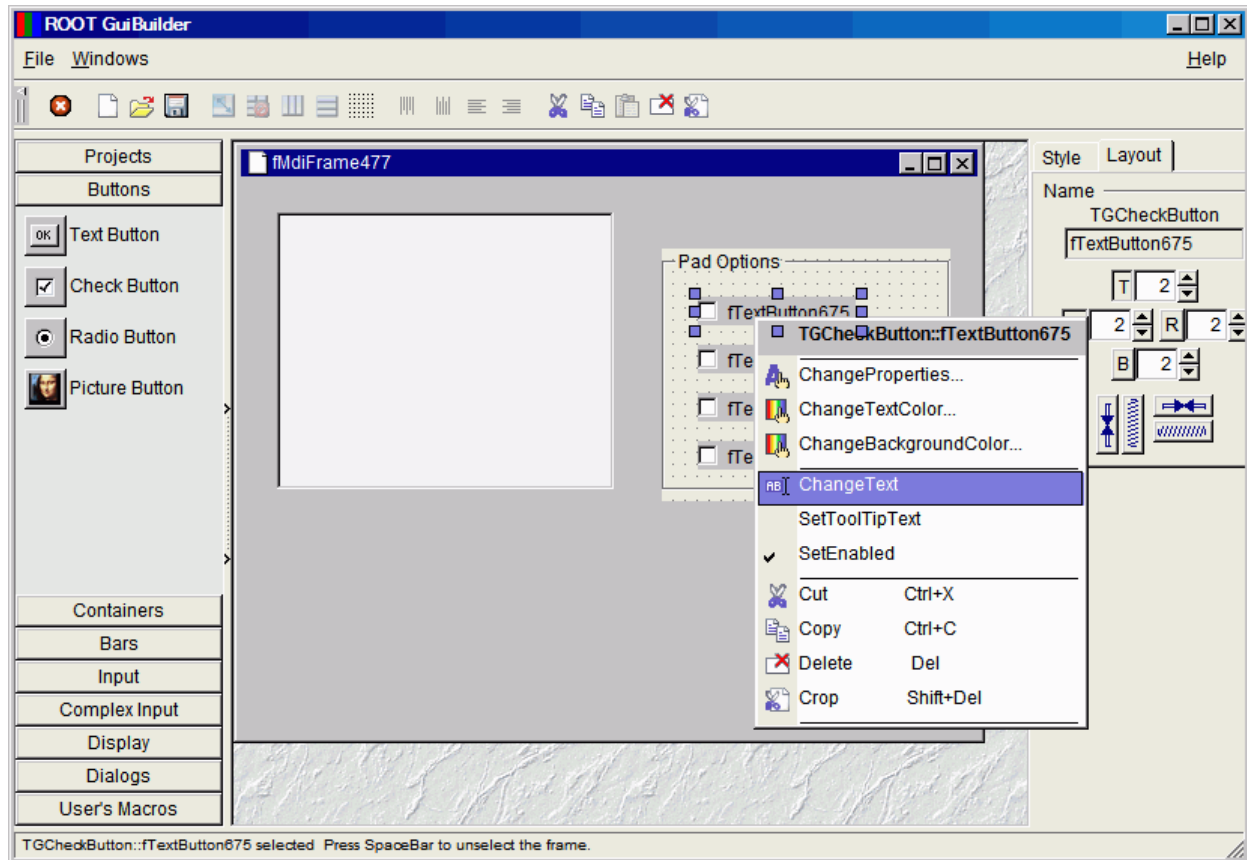
```
// transient frame
TGTransientFrame *frame2 = new TGTransientFrame(gClient-
>GetRoot(), 760, 590);
// group frame
TGGroupFrame *frame3 = new TGGroupFrame(frame2, "curve");

TGRadioButton *frame4 = new TGRadioButton(frame3, "gaus", 10);
frame3->AddFrame(frame4);
```

The GUI Builder



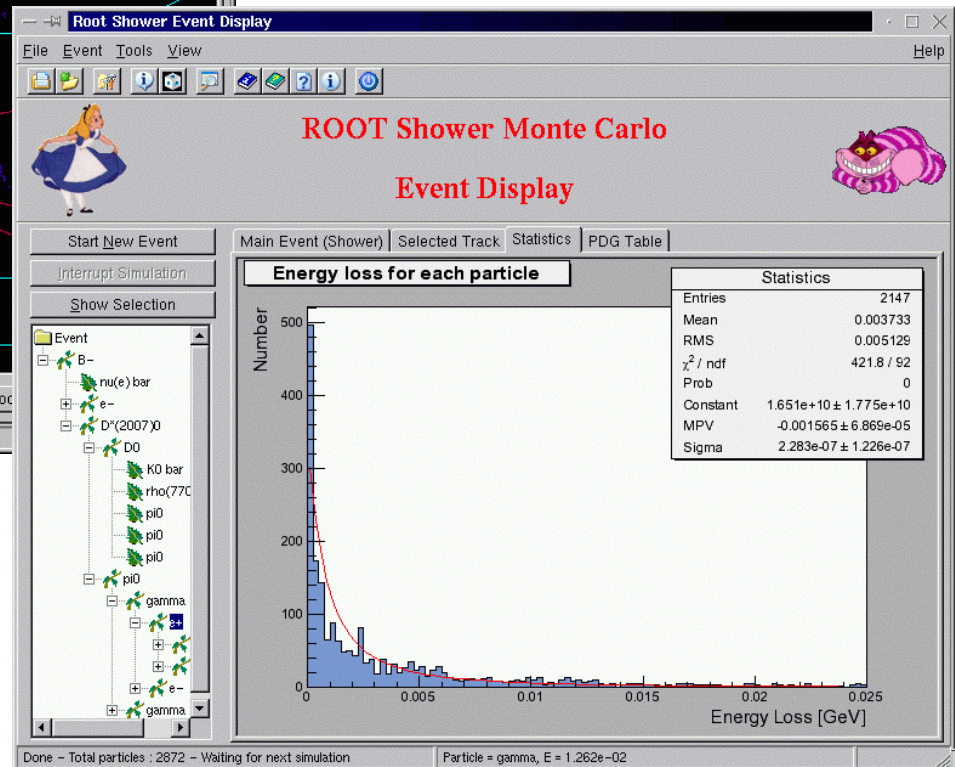
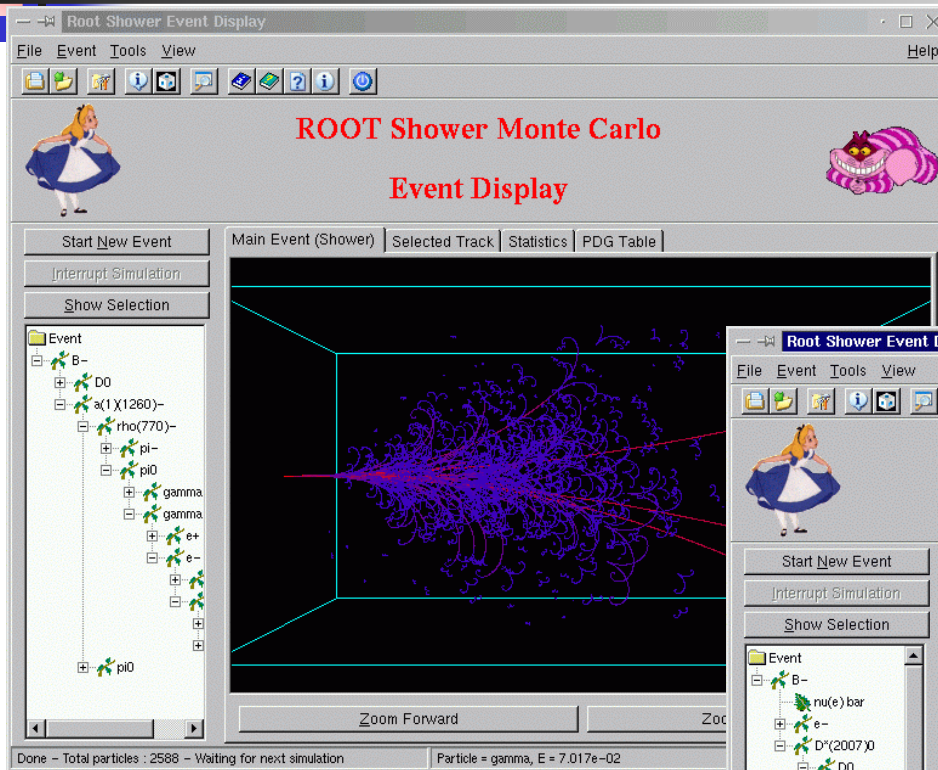
- The GUI builder provides GUI tools for developing user interfaces based on the ROOT GUI classes. It includes over 30 advanced widgets and an automatic C++ code generator.



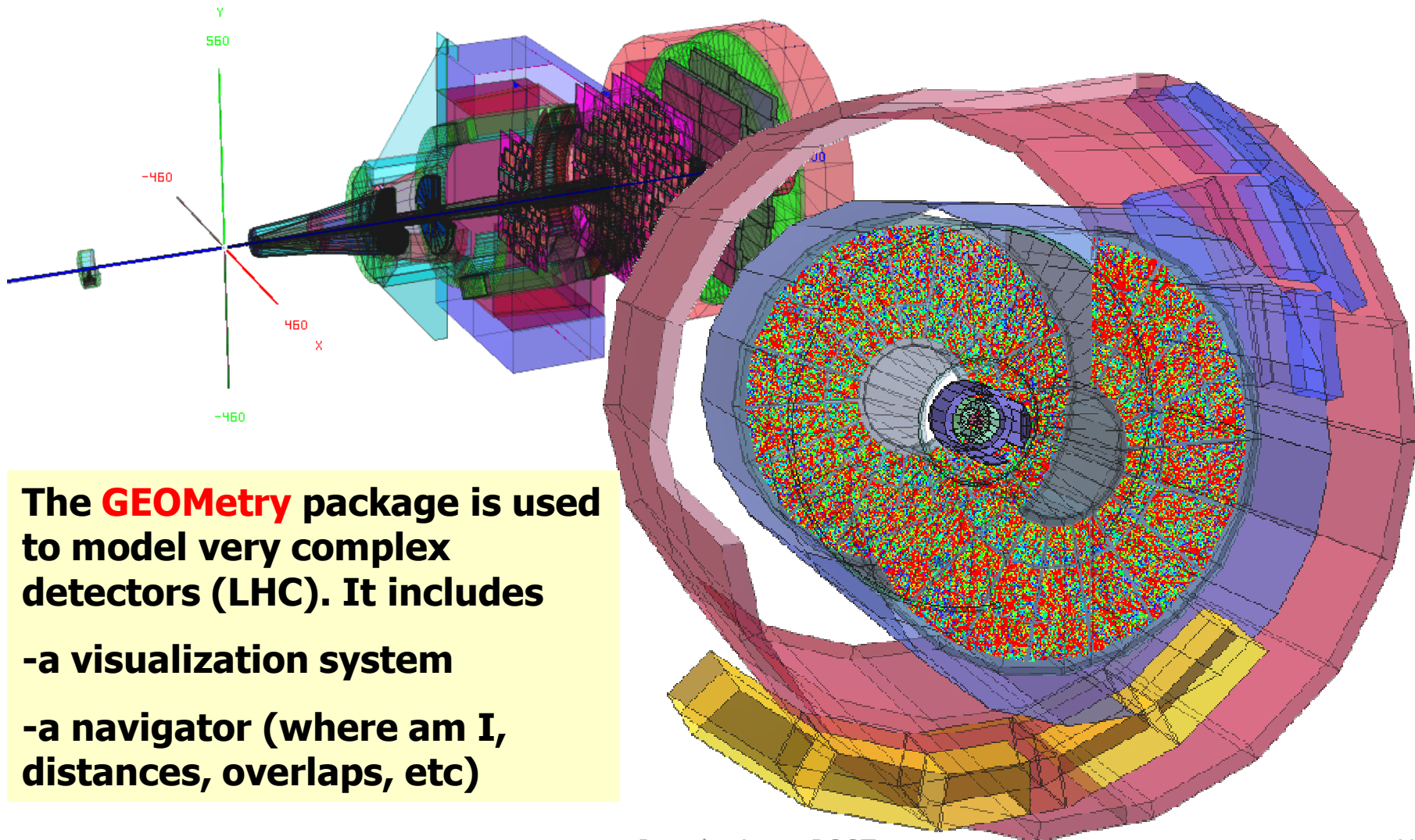
More GUI Examples



```
$ROOTSYS/tutorials/gui  
$ROOTSYS/test/RootShower  
$ROOTSYS/test/RootIDE
```



Geometry

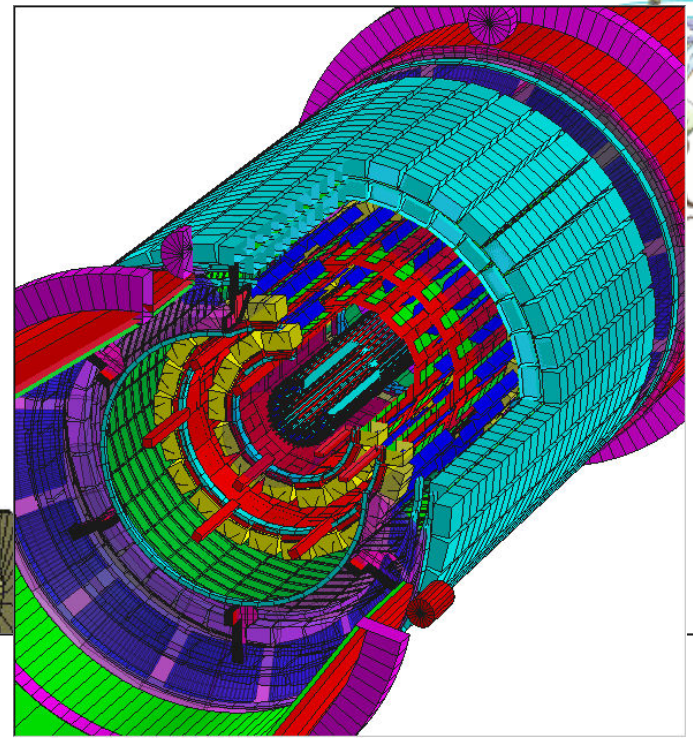
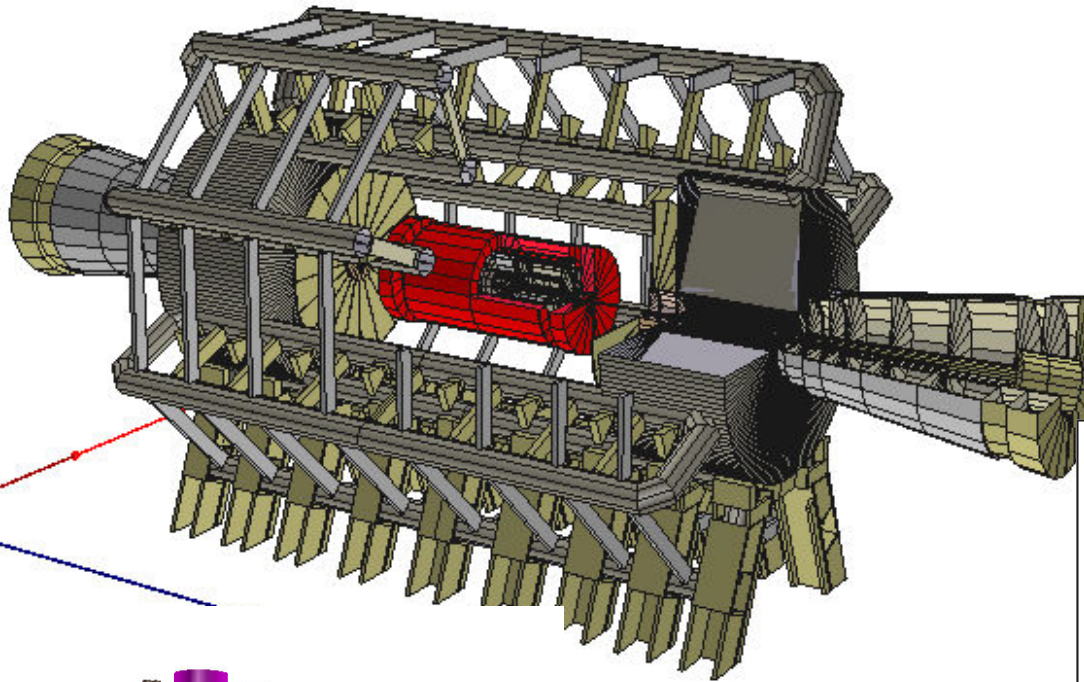


The **GEOMetry** package is used to model very complex detectors (LHC). It includes

- a visualization system

- a navigator (where am I, distances, overlaps, etc)

OpenGL



2315 z

see [\\$ROOTSYS/tutorials/geom](http://$ROOTSYS/tutorials/geom)

Math Libraries



Histogram library

TH1

TF1

MathMore

Random Numbers

Extra algorithms

Extra Math functions

GSL and more

MathCore

Function interfaces

Physics Vectors

Basic algorithms

Basic Math functions

Statistical Libraries

Statistical Utilities

TMVA

MLP

Fitting and Minimization

New Fitter

RooFit

TMinuit

TFumili
Linear Fitter

Minuit2
(new C++ Minuit)

Linear Algebra

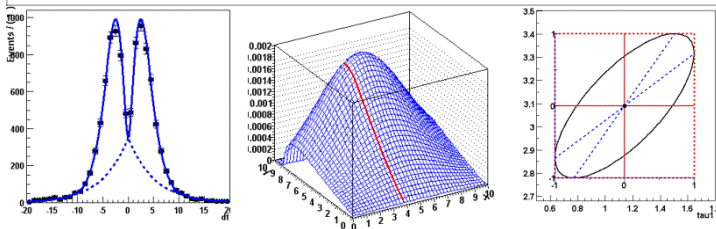
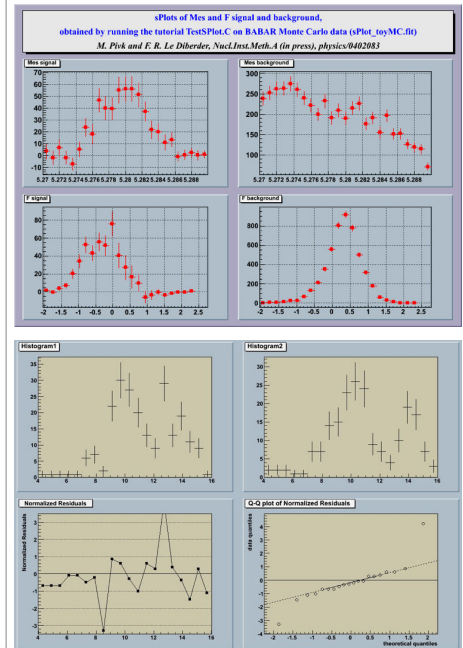
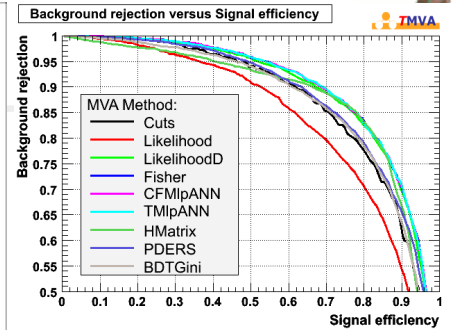
TMatrix

SMatrix

libCore

TMath

TRandom

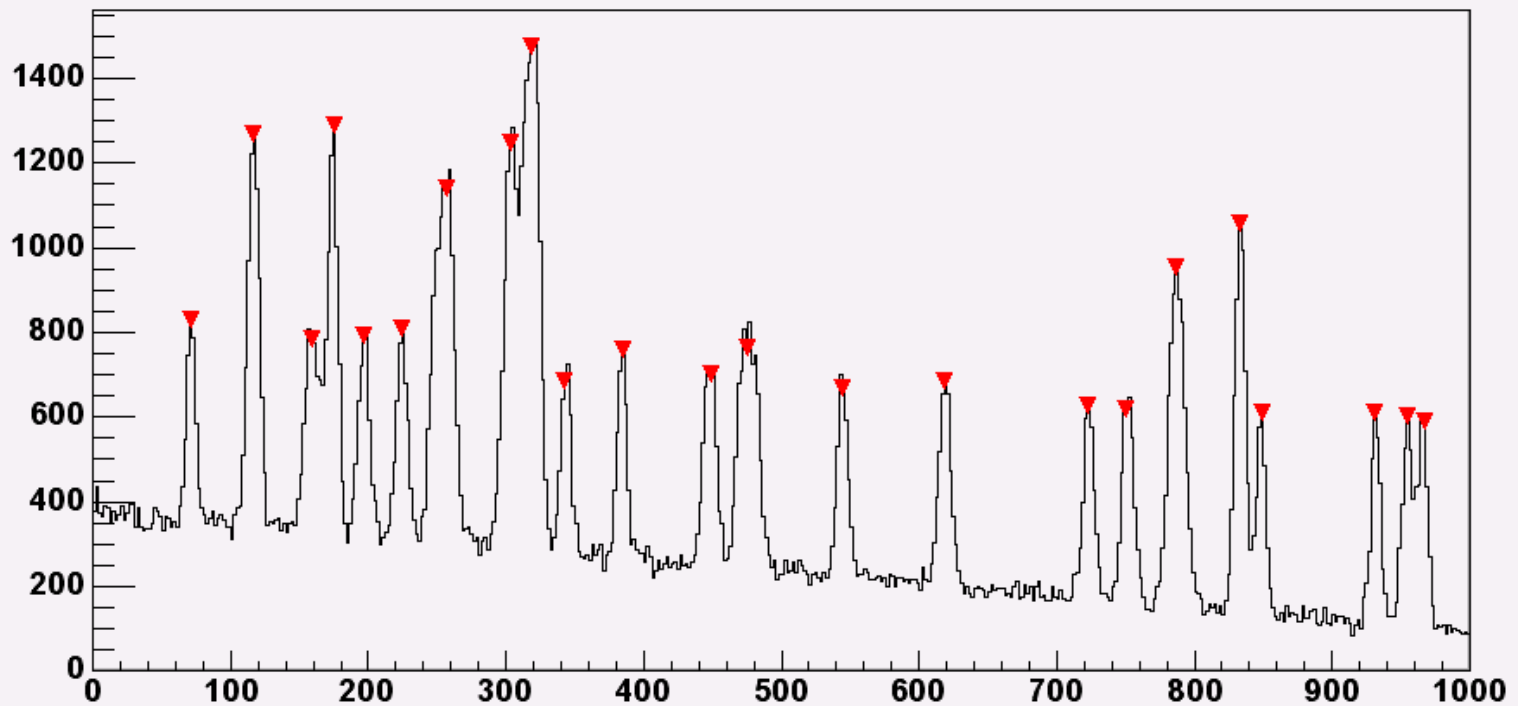


Peak Finder + Deconvolutions



TSpectrum

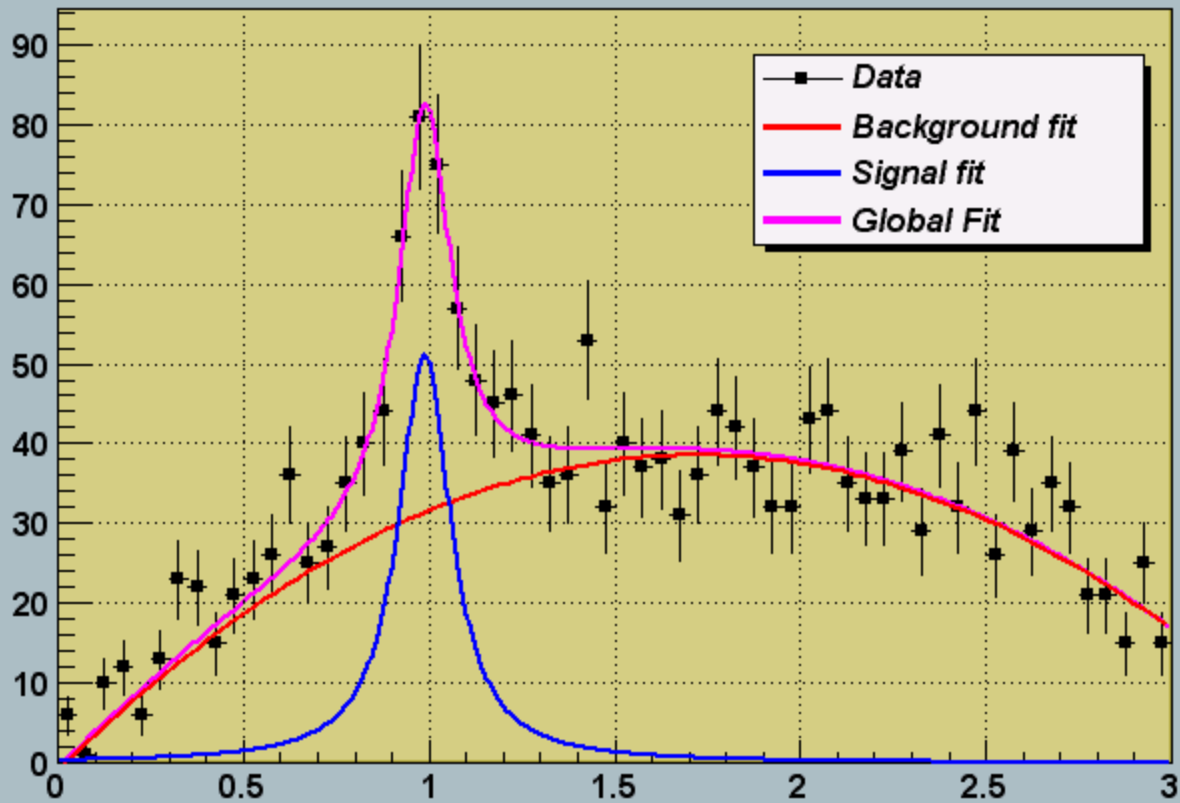
test



Fitters

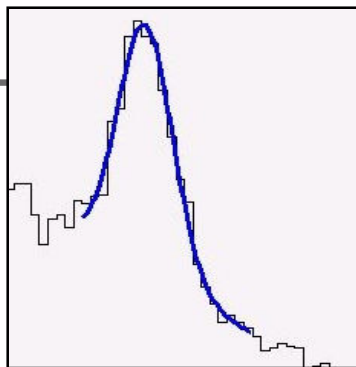


Lorentzian Peak on Quadratic Background



- Minuit
- Fumili
- LinearFitter
- RobustFitter
- RooFit

Fit Panel



Set Parameters of f1

Name	Fix	Bound	Value	Min	Set Range	Max	Step	Errors
p0	<input type="checkbox"/>	<input type="checkbox"/>	285.158	-855.474		855.474	85.5474	11.8594
p1	<input type="checkbox"/>	<input type="checkbox"/>	3392.47	-10177.4		10177.4	1017.74	4.24036
p2	<input type="checkbox"/>	<input type="checkbox"/>	-98.2206	-294.662		294.662	29.4662	4.22357
p3	<input type="checkbox"/>	<input type="checkbox"/>	-0.184565	-0.553694		0.553694	0.0553694	0.00145611
p4	<input type="checkbox"/>	<input type="checkbox"/>	921.91	-2765.73		2765.73	276.573	6.74159

Immediate preview

Reset Apply OK Cancel

New Fit Panel

Current selection: h:TH1F

General | Minimization

Function

Predefined: fitFunc Operation: Nop Add Conv

f1

Selected: fitFunc Set Parameters...

Fit Settings

Method: Chi-square User-Defined...

Linear fit Robust: 1.00 No Chi-square

Fit Options

Integral Use range

Best errors Improve fit results

All weights = 1 Add to list

Empty bins, weights=1

Draw Options

SAME

No drawing

Do not store/draw Advanced...

X:

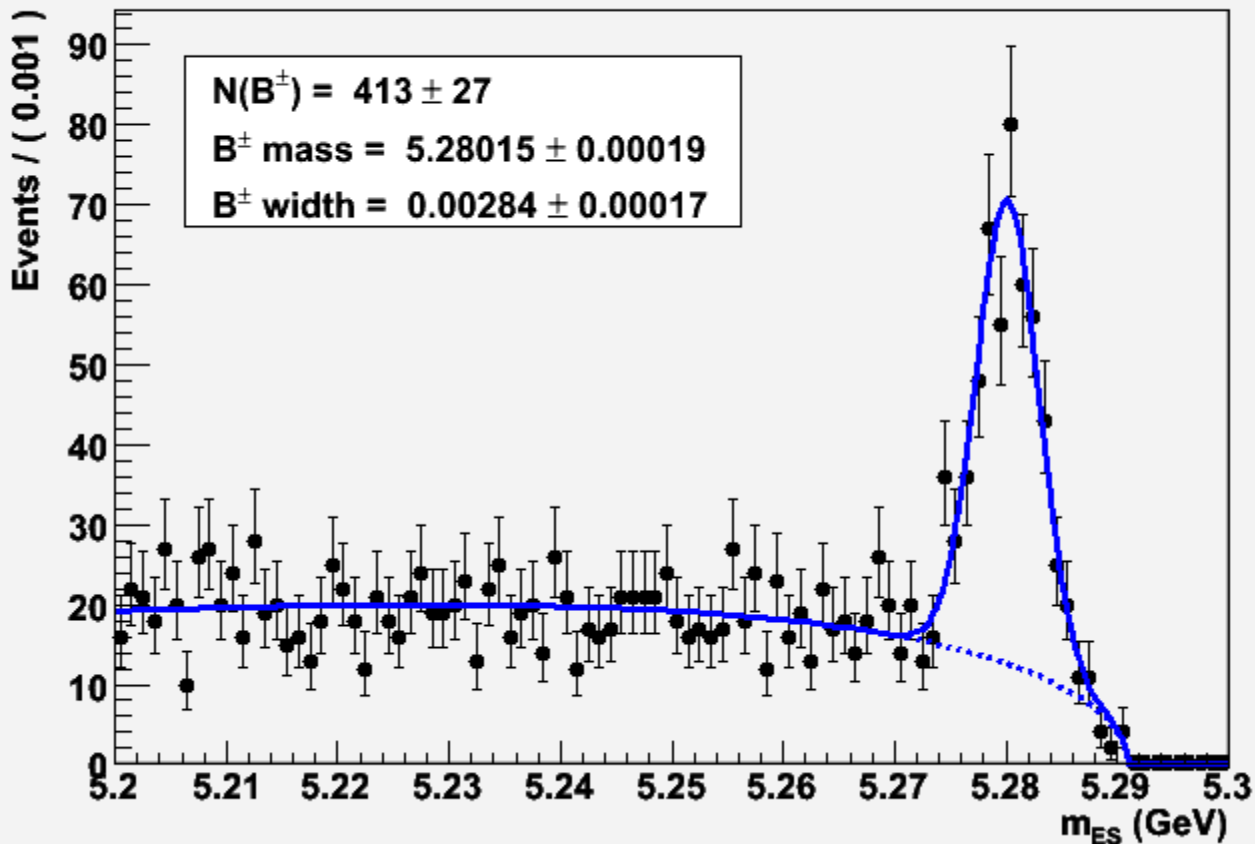
Fit Reset Close

LIB Minuit MIGRAD Itr: 5000 Prn: DEF

RooFit: a Powerful Fitting Framework



$B^{\pm} \rightarrow D^0 K^{\pm}$

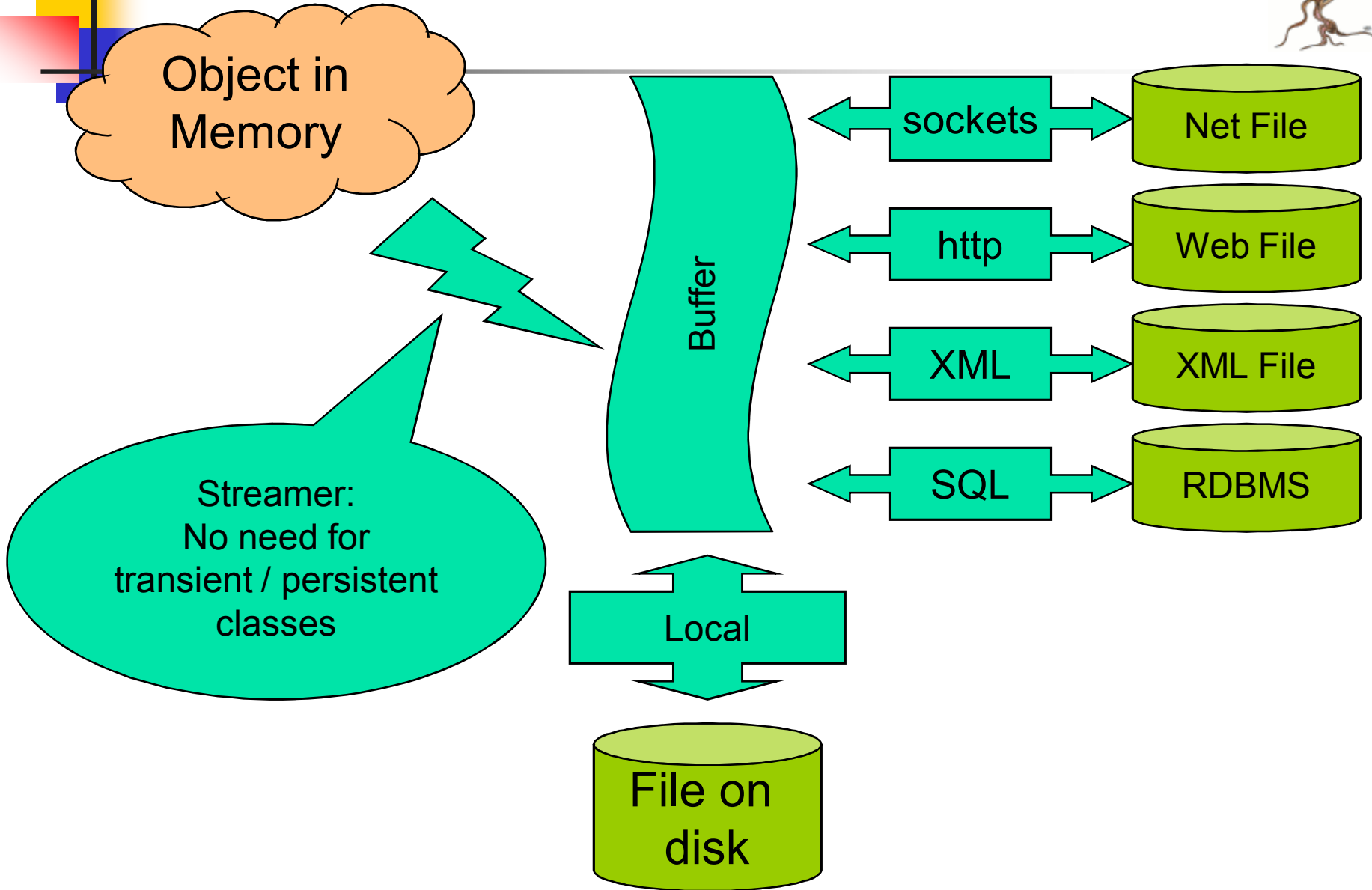


see `$ROOTSYS/tutorials/roofit/RooFitDemo.C`



Input/Output

I/O

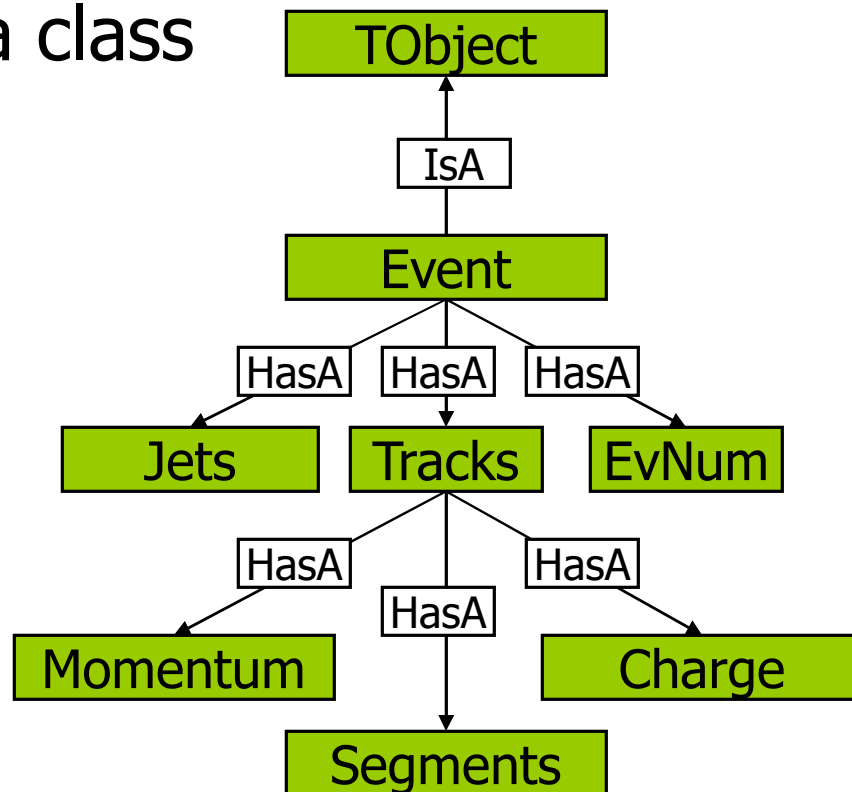




Object Oriented Concepts

- Class: the description of a “thing” in the system
- Object: instance of a class
- Methods: functions for a class

- Members: a “has a” relationship to the class.
- Inheritance: an “is a” relationship to the class.





TFile / TDirectory

- A TFile object may be divided in a hierarchy of directories, like a Unix file system.
- Two I/O modes are supported
 - **Key-mode (TKey)**. An object is identified by a name (key), like files in a Unix directory. OK to support up to a few thousand objects, like histograms, geometries, mag fields, etc.
 - **TTree-mode** to store event data, when the number of events may be millions, billions.



Self-describing Files

- Dictionary for persistent classes written to the file
- ROOT files are self describing
- Support for **Backward** and **Forward** compatibility
- Files created in 2001 must be readable in 2015
- Classes (data objects) for all objects in a file can be regenerated via `TFile::MakeProject`

```
Root > TFile f("demo.root");
```

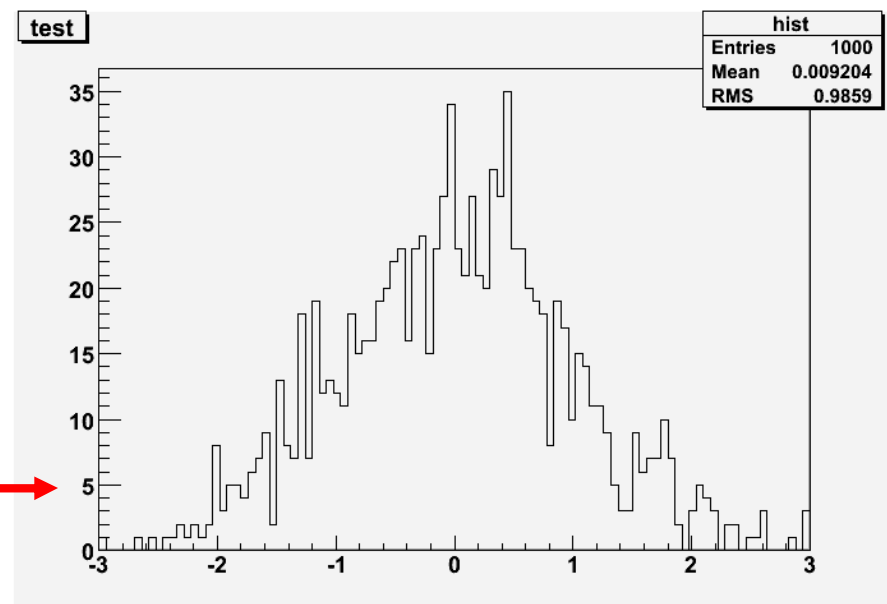
```
Root > f.MakeProject("dir", "*", "new++");
```

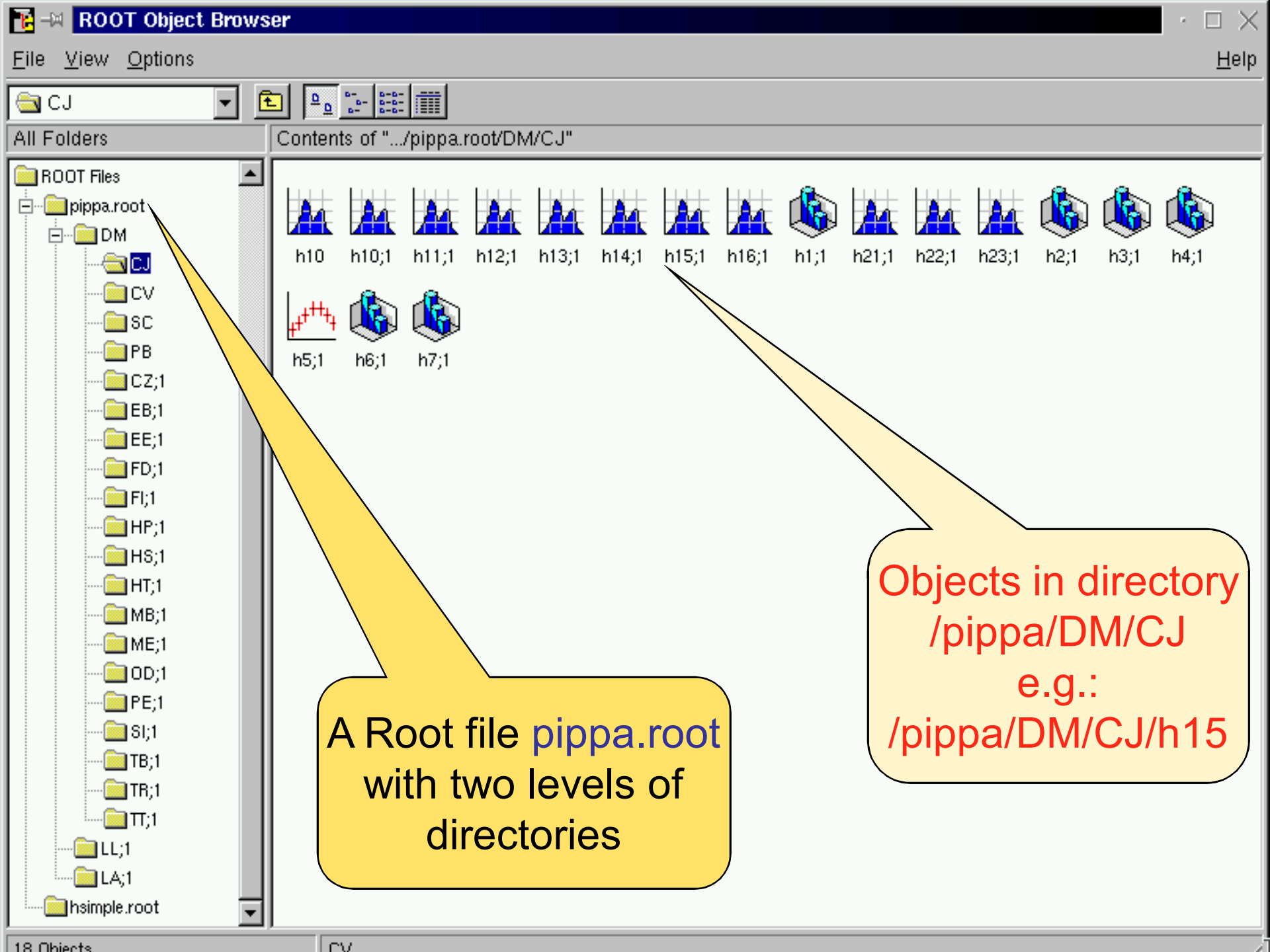


Example of Key Mode

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

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

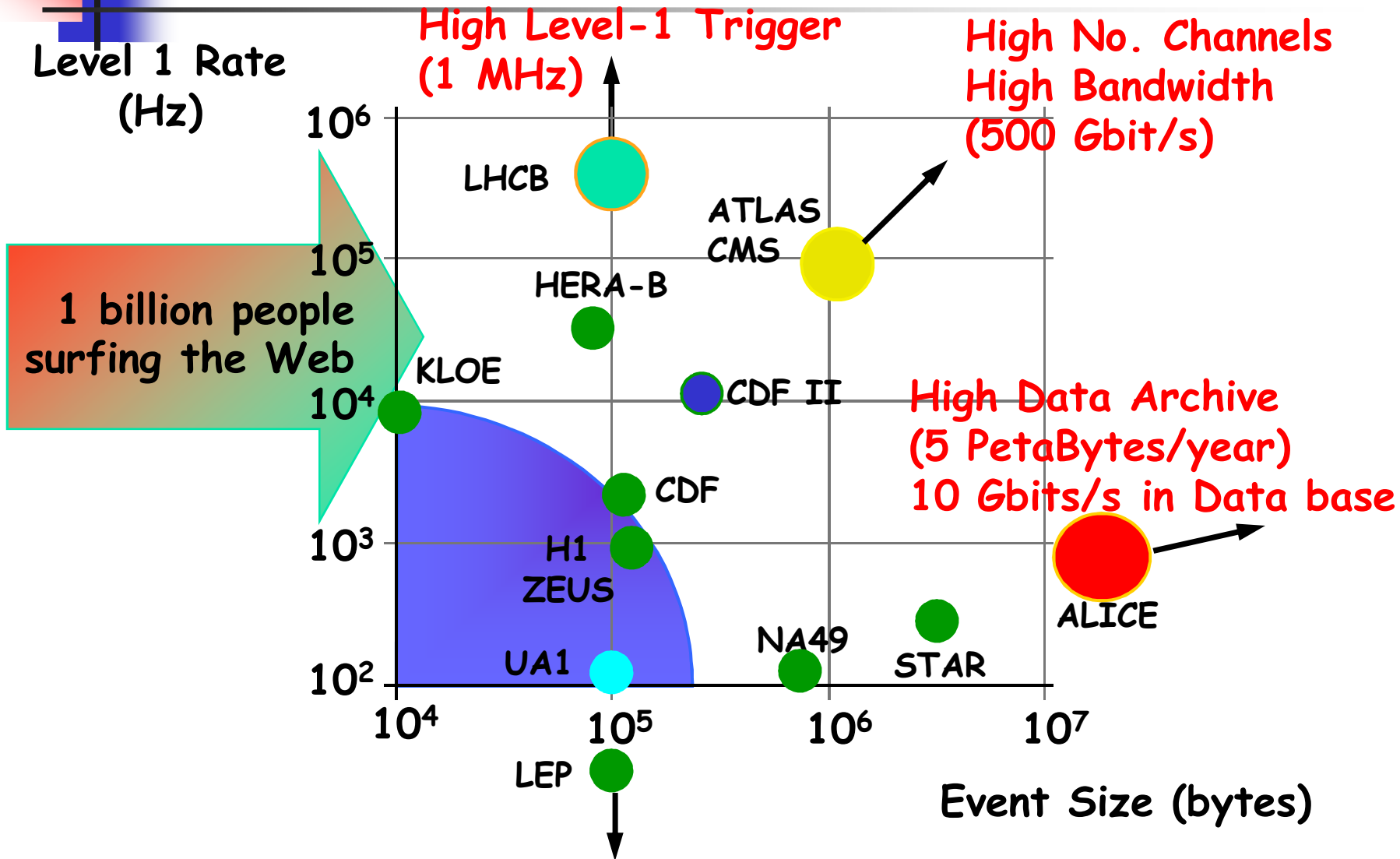




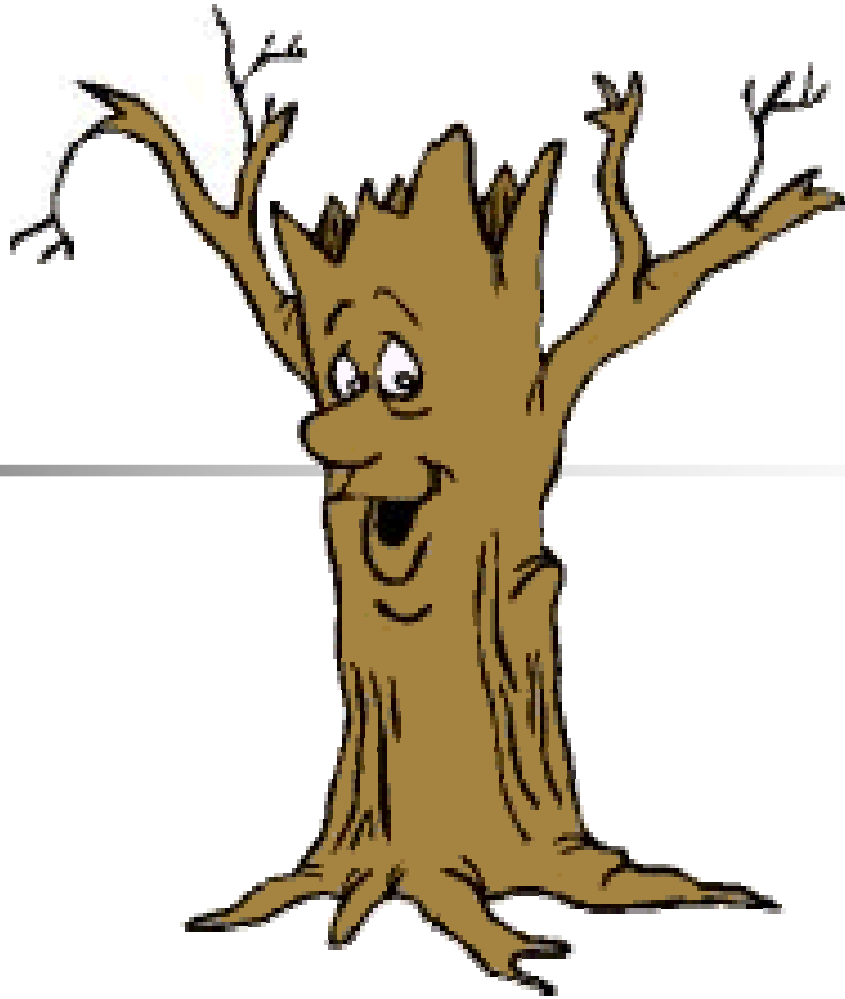
A Root file `pippa.root` with two levels of directories

Objects in directory `/pippa/DM/CJ`
e.g.: `/pippa/DM/CJ/h15`

LHC: How Much Data?



ROOT Trees



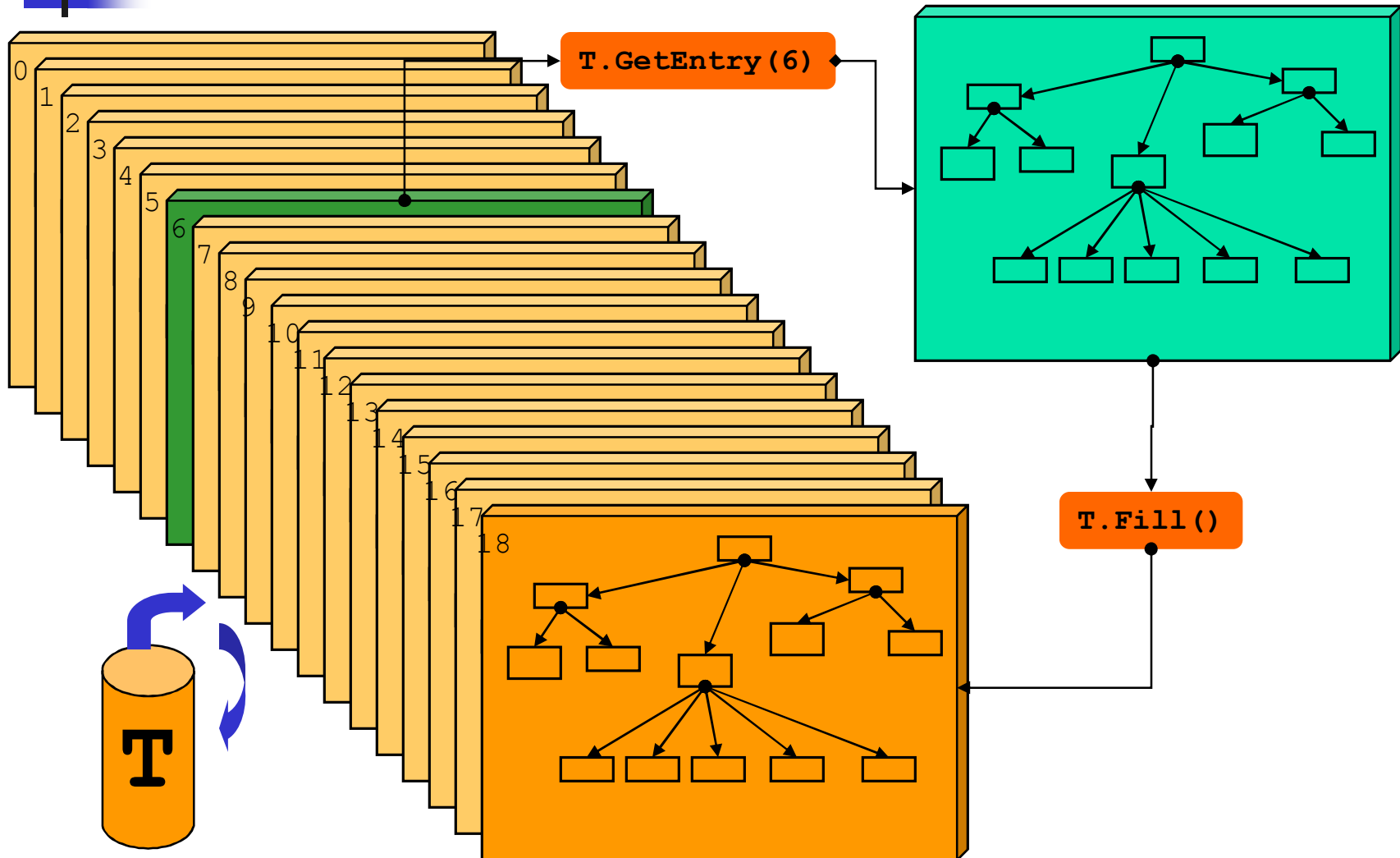


Why Trees ?

- ❑ 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 best)
- ❑ Trees have 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 `TTreeView`

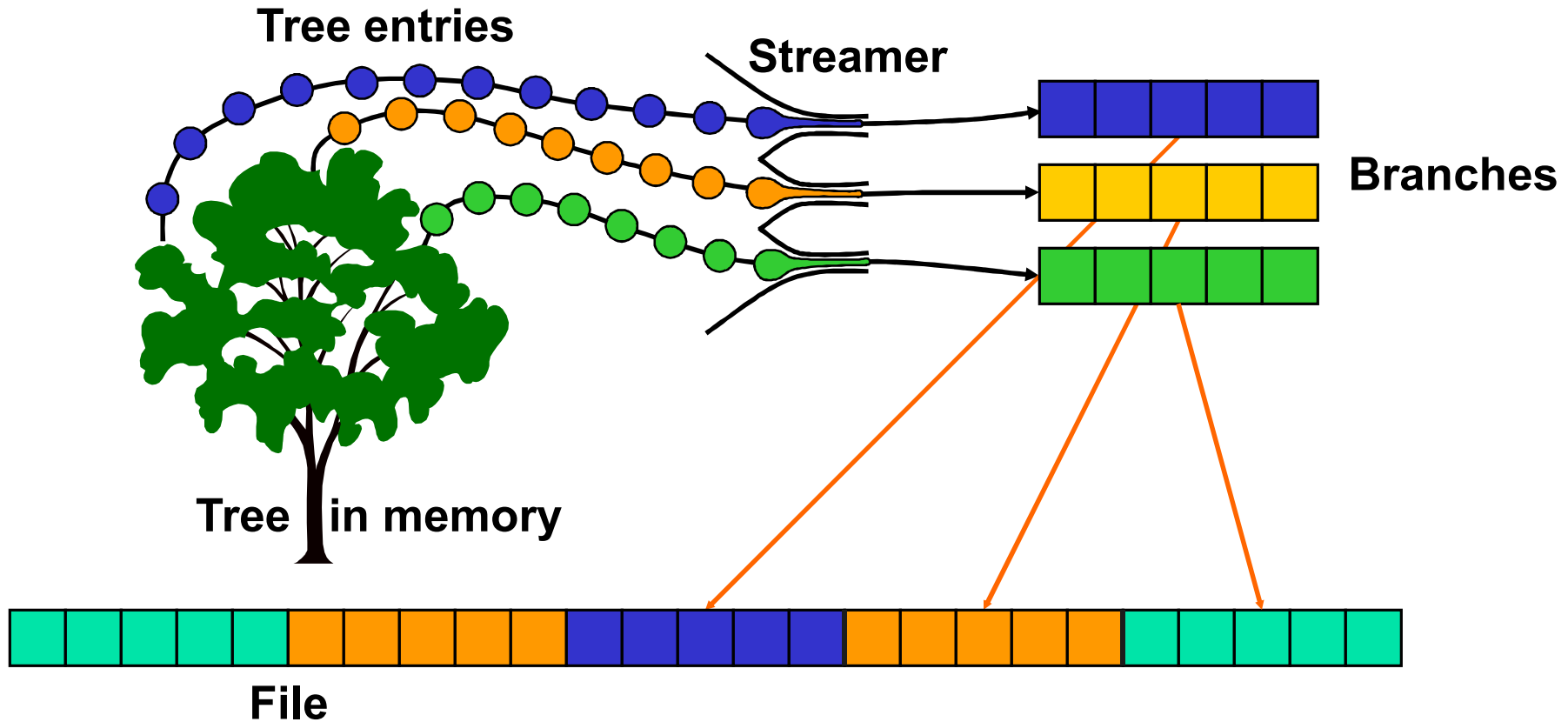
Memory <--> Tree

Each Node is a Branch in the Tree



ROOT I/O -- *Split/Cluster*

Tree version



Writing/Reading a Tree



```
class Event : public Something {
  Header          fHeader;
  std::list<Vertex*> fVertices;
  std::vector<Track> fTracks;
  TOF              fTOF;
  Calor            *fCalor;
}
```

Event.h

Write.C

```
main() {
  Event *event = 0;
  TFile f("demo.root", "recreate");
  int split = 99; //maximum split
  TTree *T = new TTree("T", "demo Tree");
  T->Branch("event", "Event", &event, split);
  for (int ev = 0; ev < 1000; ev++) {
    event = new Event(...);
    T->Fill();
    delete event;
  }
  t->AutoSave();
}
```

Read.C

```
main() {
  Event *event = 0;
  TFile f("demo.root");
  TTree *T = (TTree*)f.Get("T");
  T->SetBranchAddress("event", &event);
  Long64_t N = T->GetEntries();
  for (Long64_t ev = 0; ev < N; ev++) {
    T->GetEntry(ev);
    // do something with event
  }
}
```

Browsing a Tree



The screenshot shows the ROOT Object Browser window. The title bar reads "ROOT Object Browser". The menu bar includes "File", "View", "Options", and "Help". The address bar shows "Electrons". The left pane, titled "All Folders", displays a tree structure with "Electrons" selected. The right pane, titled "Contents of \"../atlfast.root/T/Electrons\"", lists eight files: "Electrons.fBits", "Electrons.fUniqueID", "Electrons.m_Eta", "Electrons.m_KFcode", "Electrons.m_KFmother", "Electrons.m_MCParticle", "Electrons.m_PT", and "Electrons.m_Phi".

**8 leaves of branch
Electrons**

8 Branches of T

A double-click
to histogram
the leaf

8 Objects.

The TTreeView



TreeViewer

File Edit Run Options Help

Command Option Histogram Hist Scan

Current Folder

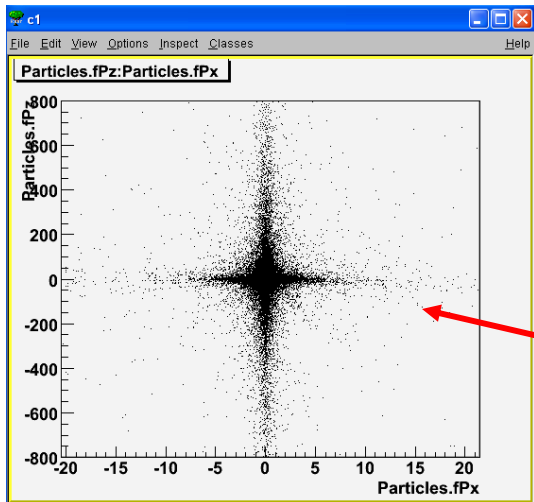
- TreeList

Current Tree : T

X: ~Particles.fPx	E() -empty-	Particles.fMass
~Particles.fPz	Particles	Particles.fVx
Z: -empty-	Particles.fLineColor	Particles.fVy
-empty-	Particles.fLineStyle	Particles.fVz
Scan box	Particles.fLineWidth	Particles.fTime
E() -empty-	Particles.fKS	Particles.fLifetime
E() -empty-	Particles.fKF	Muons
E() -empty-	User-defined expression/cut. Double-click to edit	
E() -empty-	Particles.fFirstChild	Muons.m_MCPParticle
E() -empty-	Particles.fLastChild	Muons.m_KFmother
E() -empty-	Particles.fPx	Muons.m_UseFlag
E() -empty-	Particles.fPy	Muons.m_Isolated
E() -empty-	Particles.fPz	Muons.m_Eta
E() -empty-	Particles.fEnergy	Muons.m_Phi

0%

IList OList Content : Particles.fPz





TTree Selection Syntax

Print the first 8 variables of the tree.

```
MyTree->Scan ();
```

Print all the variables of the tree.

```
MyTree->Scan ("*");
```

Print the values of var1, var2 and var3.

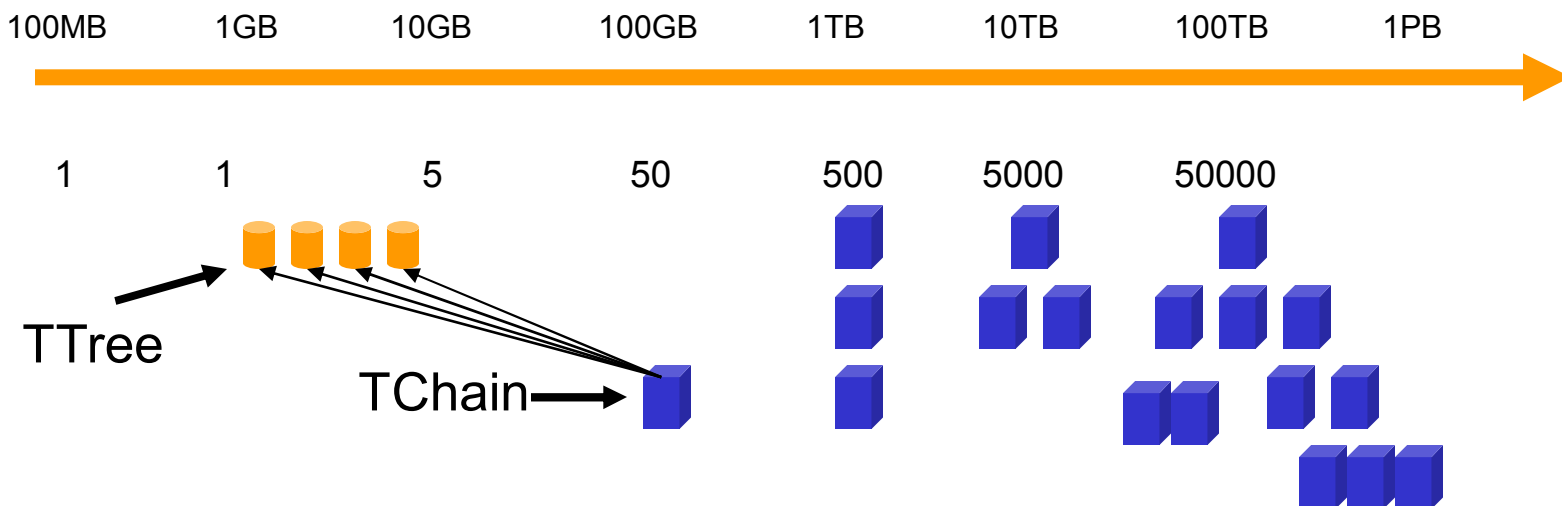
```
MyTree->Scan ("var1:var2:var3");
```

Print the values of var1, var2 and var3 for the entries where var1 is exactly 0.

```
MyTree->Scan ("var1:var2:var3", "var1==0");
```



Data Volume & Organisation



A TFile typically contains 1 TTree

A TChain is a collection of TTrees or/and TChains

A TChain is typically the result of a query to the file catalogue



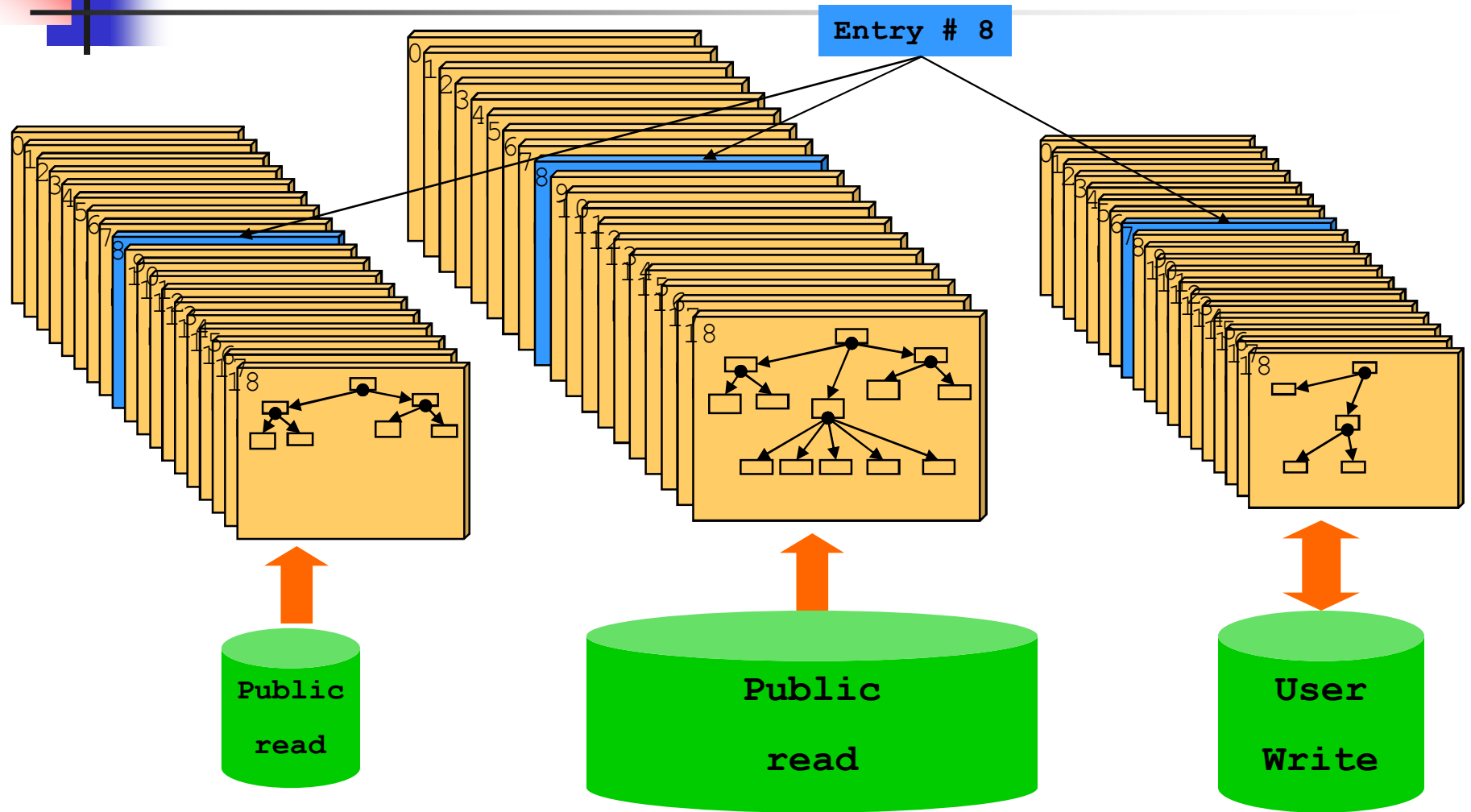
Chains of Trees

- A TChain is a collection of Trees.
- Same semantics for TChains and TTrees
 - `root > .x h1chain.C`
 - `root > chain.Process("h1analysis.C")`

```
{  
//creates a TChain to be used by the h1analysis.C class  
//the symbol H1 must point to a directory where the H1 data sets  
//have been installed  
  
TChain chain("h42");  
  chain.Add("$H1/dstarmb.root");  
  chain.Add("$H1/dstarp1a.root");  
  chain.Add("$H1/dstarp1b.root");  
  chain.Add("$H1/dstarp2.root");  
}
```



Tree Friends





GRIDs & Multi-Cores & PROOF



From the desktop to the GRID

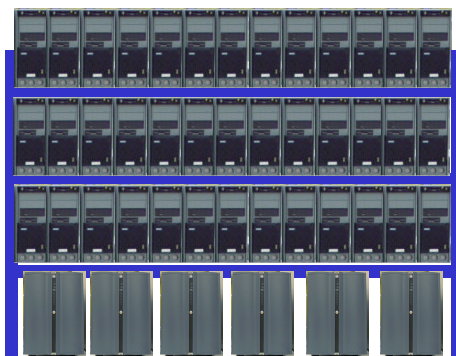
Online/Offline

Local/remote

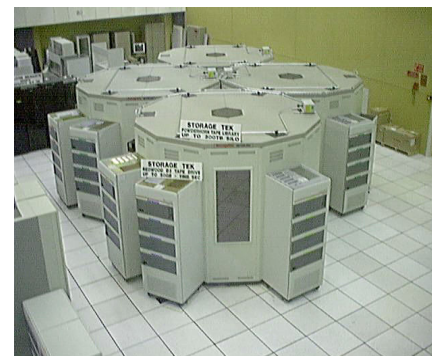
Desktop



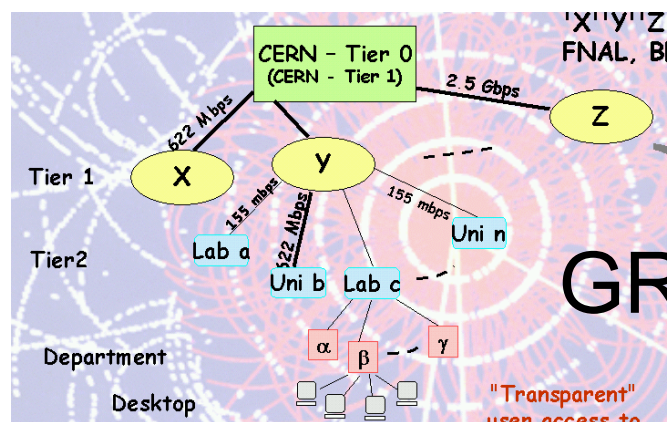
Farms



Storage



New data analysis tools must be able to use in parallel remote CPUs, storage elements and networks in a transparent way for a user at a desktop



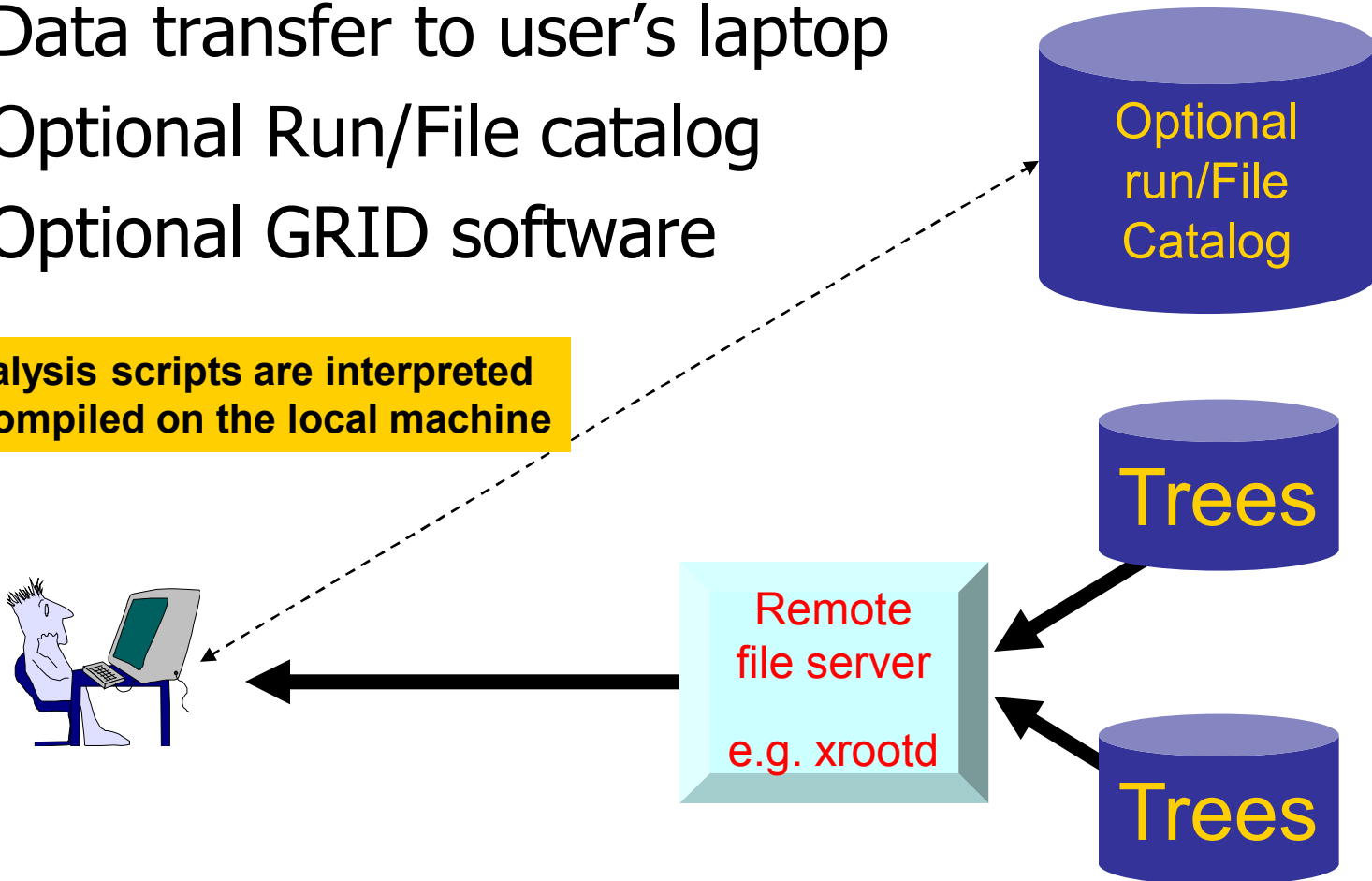
GRID: Interactive Analysis

Case 1



- Data transfer to user's laptop
- Optional Run/File catalog
- Optional GRID software

Analysis scripts are interpreted or compiled on the local machine



GRID: Interactive Analysis

Case 2



- Remote data processing
- Optional Run/File catalog
- Optional GRID software

Analysis scripts are interpreted or compiled on the remote master(s)



Commands, scripts

Histograms, trees

Remote
data analyzer
e.g. PROOF

slave

slave

slave

slave

slave

slave

Run/File
Catalog

Trees

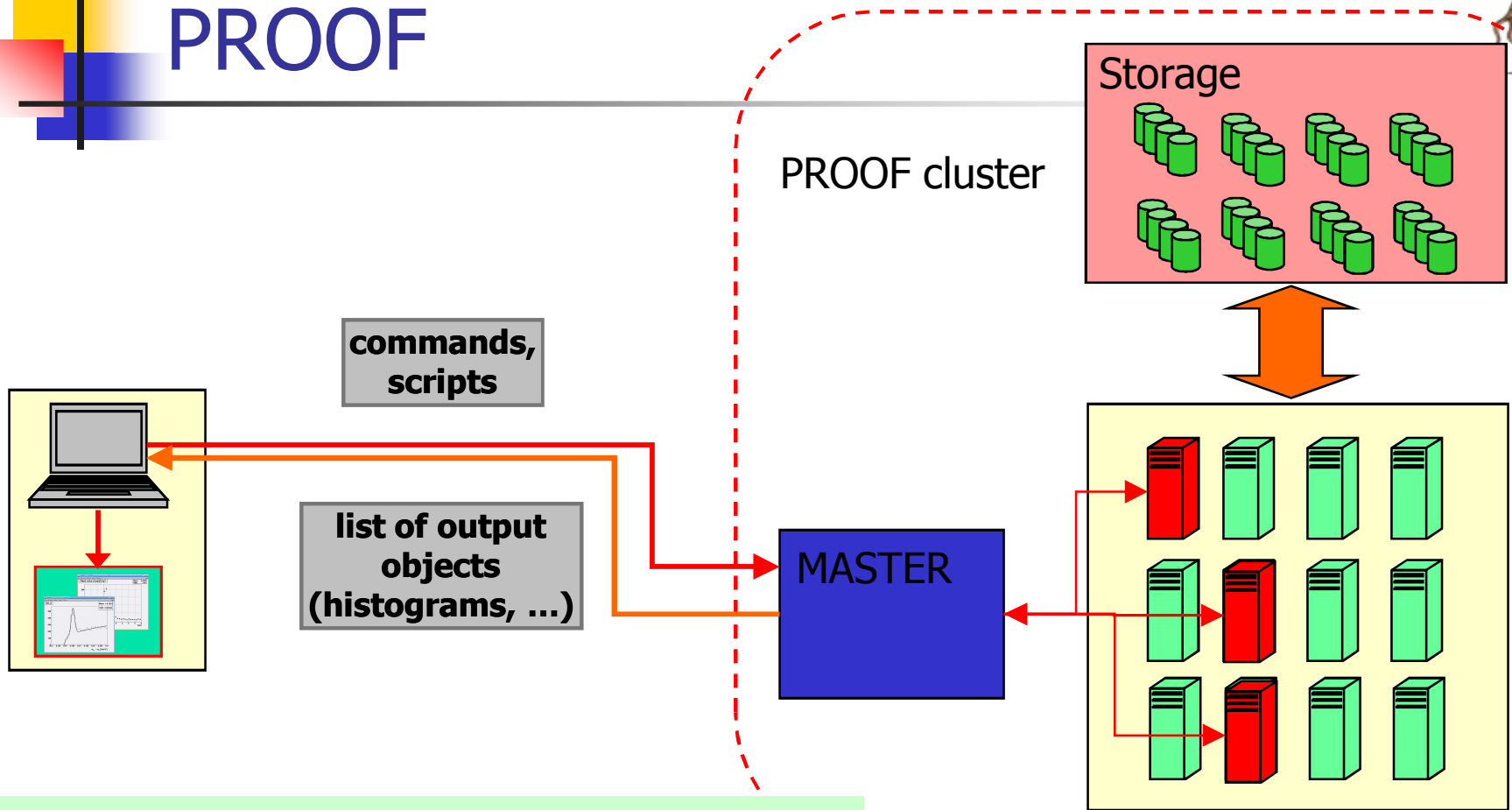
Trees



Parallel ROOT Facility

- A system for running ROOT queries in parallel on a large number of distributed computers or many-core machines
- PROOF is designed to be a transparent, scalable and adaptable extension of the local interactive ROOT analysis session
- Extends the interactive model to long running “interactive batch” queries
- Uses xrootd for data access and communication infrastructure
- For optimal CPU load it needs fast data access (SSD, disk, network) as queries are often I/O bound
- Can also be used for pure CPU bound tasks like toy Monte Carlo’s for systematic studies or complex fits

PROOF



- Cluster perceived as extension of local PC
 - *Same macro and syntax* as in local session
- More *dynamic* use of resources
- Real-time feedback
- Automated *splitting* and *merging*



ROOT is MORE....

- In this talk, I presented the most basic classes typically used during Physics Analysis.
- 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



Documentation

- Users Guide and Reference Manuals are available at <http://root.cern.ch>

Tomorrow Jan Fiete will demo in his session many of the features I've presented today