Modern tools for simulation and analysis: Geant4, ROOT

P.Hristov

30/09/2011

ISTC-CERN-JINR Summer School

on High Energy Physics and Accelerator Physics

Outlook & References

- Data processing and reconstruction, analysis
  - J. Boyd, "From raw data to physics"

  https://indico.cern.ch/conferenceDisplay.py?confId=134618
- Analysis
  - D. Glenzinski, "Physics and Analysis at a Hadron Collider - An Introduction"

  https://indico.cern.ch/conferenceDisplay.py?confId=77805
- ROOT
  - J.F. Grosse-Oetringhaus, "Introduction to ROOT"

  https://indico.cern.ch/conferenceDisplay.py?confId=134329
- Geant4
  - J. Apostolakis, "The Geant4 Toolkit: Evolution and Status"

  http://bit.ly/g4mc2010
  - U. Penn Geant4 Tutorial (full week)

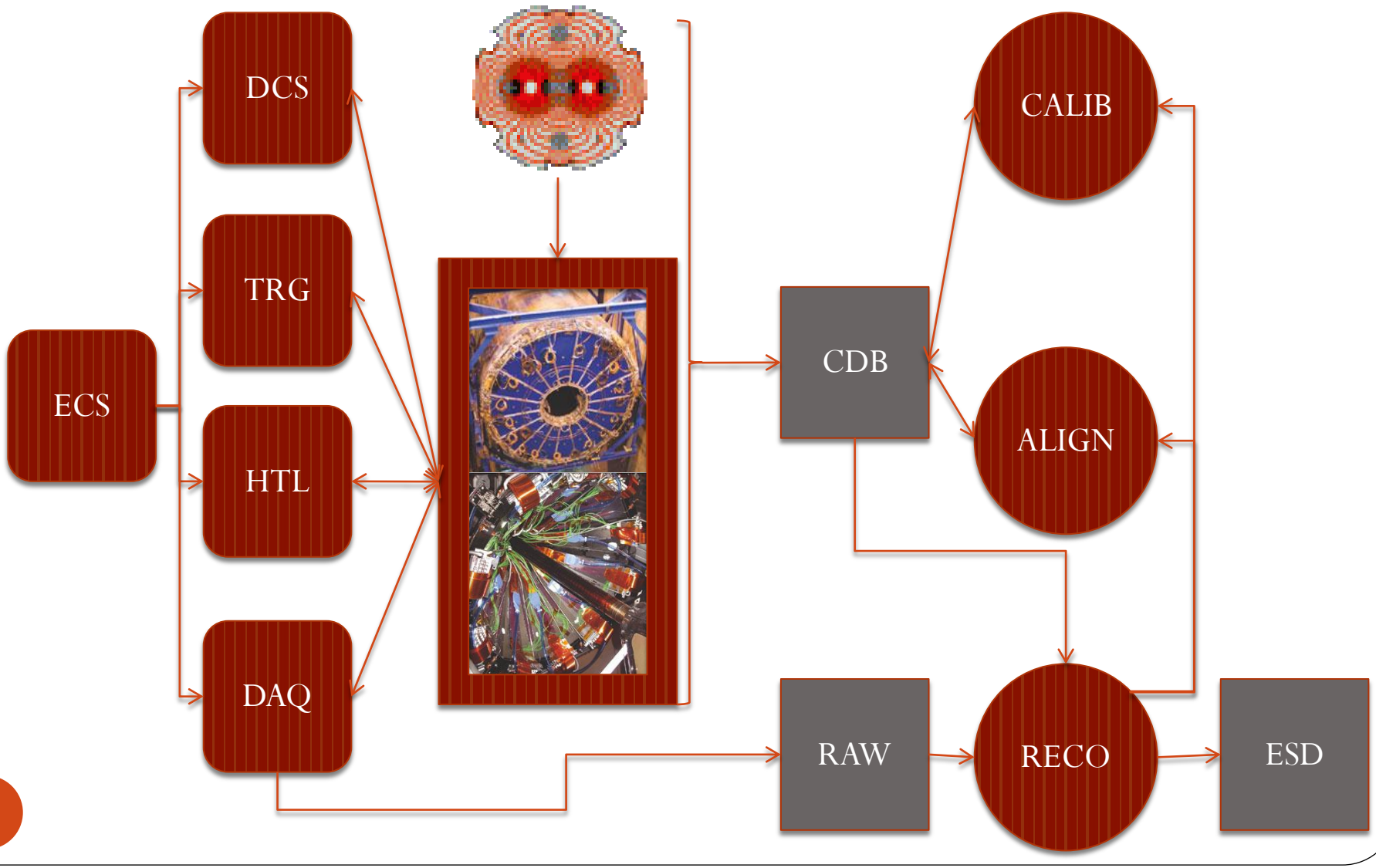
  http://geant4.slac.stanford.edu/UPenn2011/Agenda.html
  - D. Oxley, "An Introduction to Geant4"

  https://ns.ph.liv.ac.uk/~dco/Lectures/An_Introduction_to_Geant4_pdf.pdf

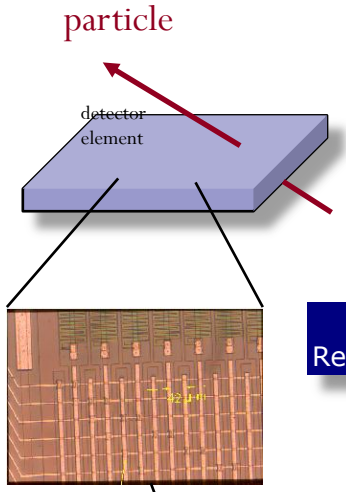
Abbreviations

- ECS – Experiment Control System
- DCS – Detector Control System
- TRG – Trigger
- DAQ – Data Acquisition System
- CDB – Conditions Data Base
- RAW – Raw Data
- CALIB – Calibration Procedures
- ALIGN – Alignment Procedures
- RECO – Reconstruction Procedures
- ESD – Event Summary Data
- AOD – Analysis Object Data
- HITS – Simulated energy deposition at given location and time + MC truth
- DIGI – Digitization Procedures
  - From analog signal to digital representation
- MC truth – the characteristics of the simulated particle and/or "labels" to navigate back to the array of simulated particles

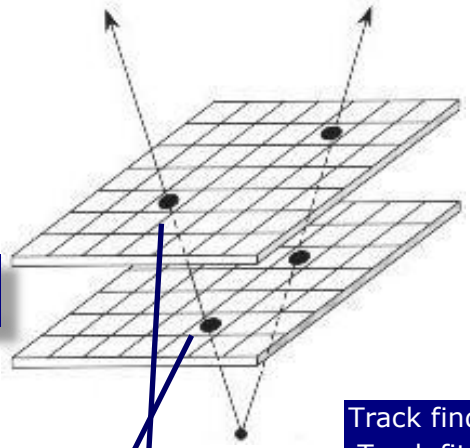
Raw ("Real") Data Processing



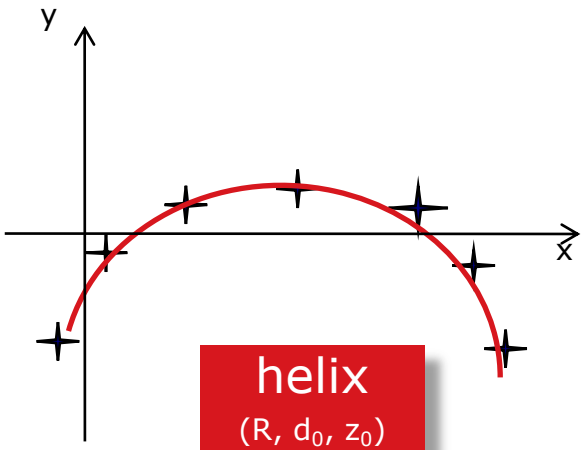
Data reduction/abstraction



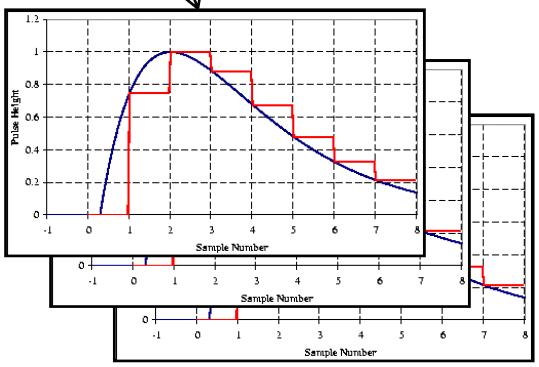
Digitization/
Reconstruction --->



Track finding +
Track fit --->



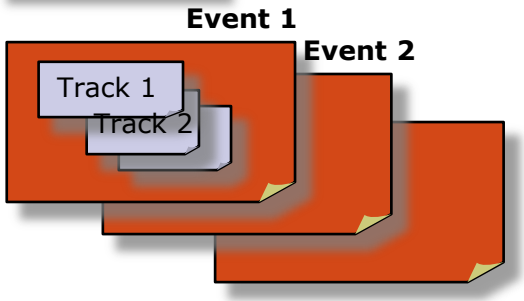
helix
(R, d_0, z_0)



Analog signals

hits
(x_1,y_1,z_1, t_1)
(x_2,y_2,z_2, t_2)
...

Magnetic field B:
reconstruct



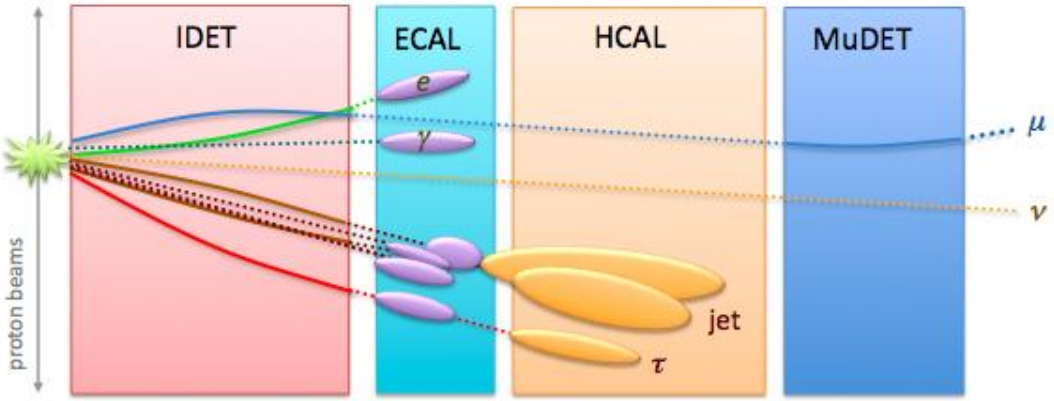
Store the
info for every
event and
every track
<-------

$$\mathbf{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

Track momentum

Reconstruction

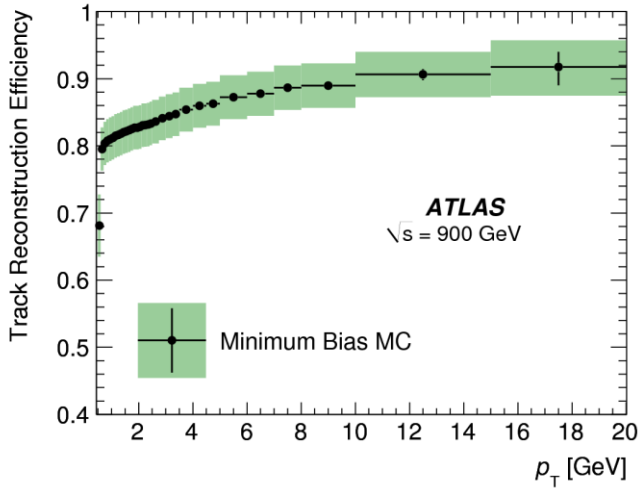
- Detector reconstruction
  - Tracking : global methods (Hough transform, combinatorial, etc), local methods (Kalman filter)
    - finding path of charged particles through the detector
  - Calorimeter reconstruction: clusterization algorithms
    - finding energy deposits in calorimeters from charged and neutral particles
  - Particle identification: dE/dx; transition radiation; E/p; Cerenkov angle, etc.
- Combined reconstruction
  - Track/cluster matching
  - Electron/Photon identification
  - Muon identification
  - Jet finding
- Calibrations and alignments applied at nearly every step

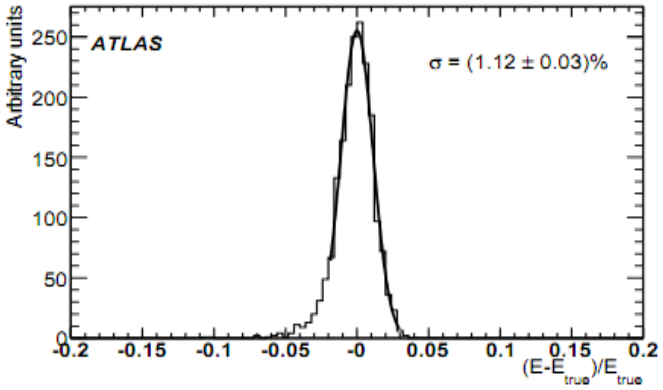


J.Boyd

Important figures of merit for reconstructed objects

- **Efficiency** = (Number of Reconstructed Tracks) / (Number of True Tracks)
  - how often do we reconstruct the object – e.g. tracking efficiency
- **Resolution** = (Measured_Energy – True_Energy)/ True_Energy
  - how accurately do we reconstruct a quantity – e.g. energy resolution
- **Fake rate** = (Number of jets reconstructed as an electron) / (Number of jets)
  - how often we reconstruct a different object as the object we are interested in – e.g. a jet faking a electron





For physics analysis it is important

i) to have high efficiency, good resolution, and low fake rates

ii) to be able to measure the efficiencies, resolutions and fake rates and their uncertainties (not easy)

J.Boyd

Resolution: useful expressions

- Momentum:
- Energy:
- Impact parameter:
- Examples from CDF & D0

	CDF	D0
	12	60 (MeV/$c^{2)}$
	2.5	6.0 (GeV/$c^{2)}$
	3.0	3.0 (GeV/$c^{2)}$
	16	14 (%)
	30	30 (μm)

Analysis Strategy

Q: What constitutes a complete analysis?

A: A suite of studies which together provide a coherent and thorough description of a particular set of data events

- Should cover all aspects necessary to understand and characterize these events
- Should be well documented via internal notes
- Should be subjected to peer review

Rules of Thumb

- Look before you leap
  - Plan your analysis strategy carefully even if the analyses are in general more iterative than linear
  - Identify those aspects which will drive the sensitivity
  - What plots, figures, and tables will be important?
  - What data sets will you need?
  - What triggers do these data sets use?
  - What Monte Carlo (MC) samples will you need?
- Trust but verify
  - Always ask yourself, "Does this make sense?"
  - Know where to find more detailed information if necessary
- A stitch in time saves nine
  - Sweat the (relevant) details, it will save time in the long run
  - When you spot a problem, take the time to understand it

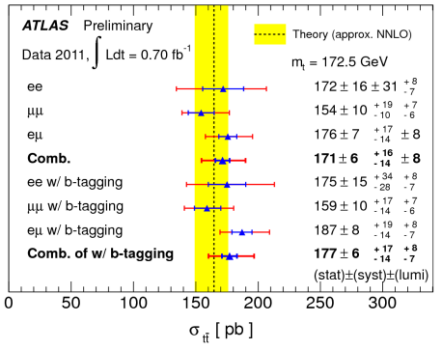
Analysis Basics

- Basic inputs to all analyses essentially the same
  - Estimate of signal acceptance after all requirements
  - Estimate of number of expected background events surviving all selection requirements
  - Statistical and systematic uncertainties for each
- Basic types of analyses
  - Counting experiments (cross sections, BR)
  - Determining properties (mass, lifetime)
  - Search for something new (small SM σ*BR, new physics (NP))
- As you see, we need both "real" data and MC

~TB Physics Analysis Steps

- Start with the output of reconstruction
- Apply an event selection based on the reconstructed object quantities
  - Often calculate new information e.g masses of combinations of particles
  - Event selection designed to improve the 'signal' to 'background' in your event sample
- Estimate
  - Efficiency of selection
  - Background after selection
  - Can use simulation for these – but have to use data-driven techniques to understand the uncertainties
- Make final plot
  - Comparing data to theory
  - Correcting for efficiency and background in data
  - Include the statistical and systematic uncertainties

~kB



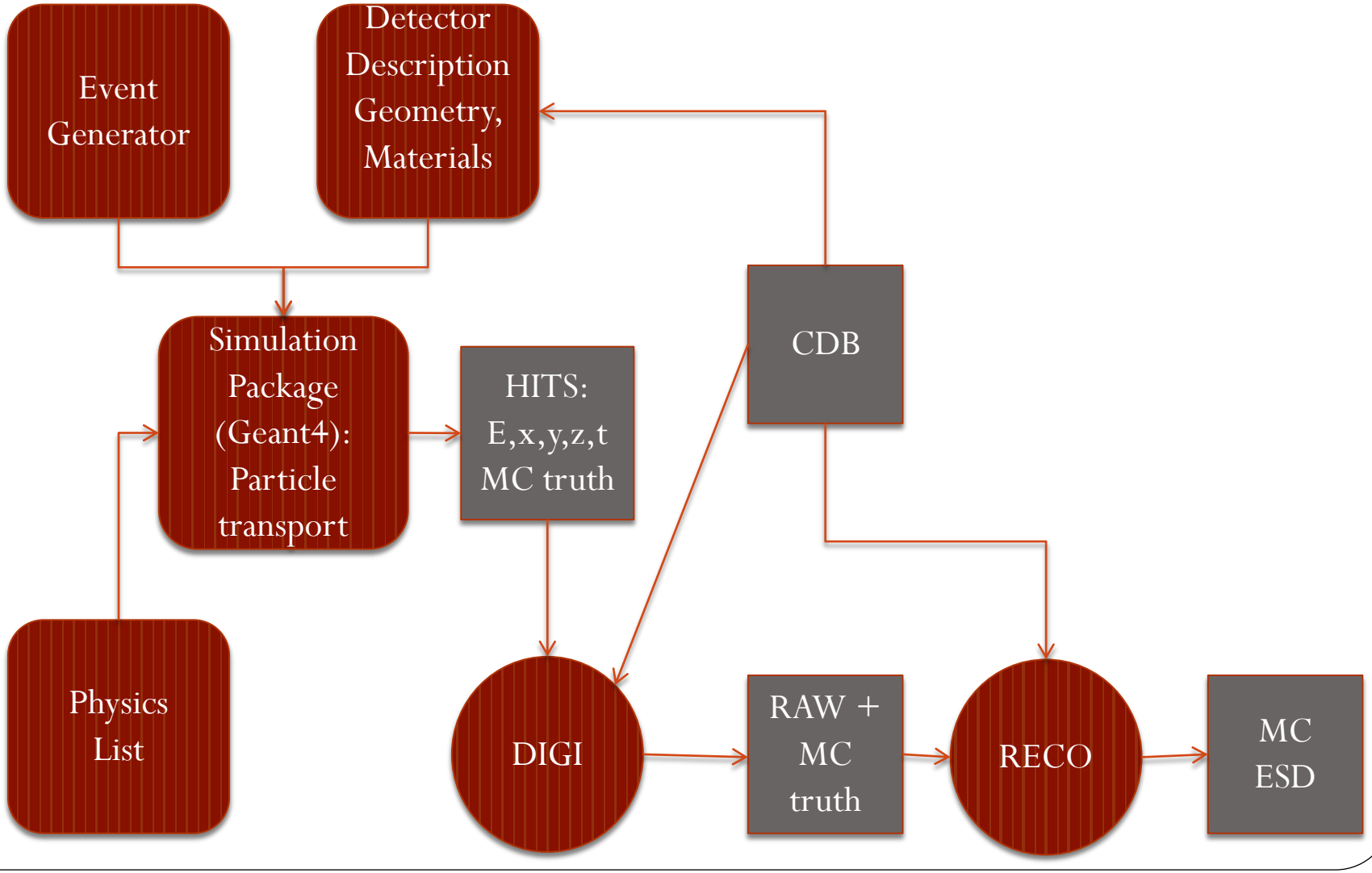
Sensitivity studies (to improve the selection criteria/"cuts")

- You first need to choose a "figure-of-merit" (FOM)
- Obvious for measurements
  - FOM = minimize the expected uncertainty on the quantity being measured
- For searches, a choice needs to be made
- Standard FOM are
  - Maximize $S^2/(S+B)$
  - Minimize expected limit, <Limit>
  - Minimize necessary luminosity to achieve a given level of "discovery", $L_{5\sigma}$

Efficiency

- We factorized the efficiency into several components
  - Used data-driven determinations of efficiency whenever possible (for example the trigger efficiencies)
  - Allows some of the work to benefit other analyses since many of the efficiencies are independent of a specific analysis
  - Requires some forethought to ensure pieces are consistently defined
  - Check the consistency between Data and MC when you estimate efficiencies from MC
- Example
  - $\alpha \cdot \varepsilon_{total} = \alpha \cdot \varepsilon_{Tracking} \cdot \varepsilon_{Trigger} \cdot \varepsilon_{Vertexer} \cdot \varepsilon_{Analysis}$
  - α is the geometric and kinematic acceptance

Simulated ("MC") Data Processing



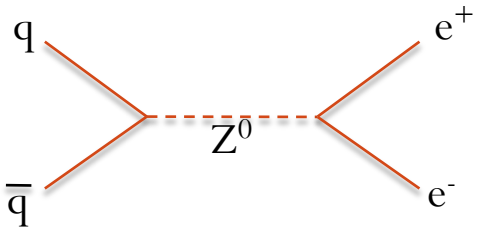
Simulation workflow: Example

Physics simulation

Simulate the physics interaction (set in the simulation configuration)

Output of this part is the 4-vector's of the produced particles.

In this case the 4-vector's of the 2 electrons from the Z decay.



Detector Simulation

Simulate the propagation of the electrons through the detector.

Including:

-bending in the magnetic field

-leaving hits in the tracking detector elements

-interacting with the material in the detector

-interacting in the calorimeter (detailed description of the EM shower)

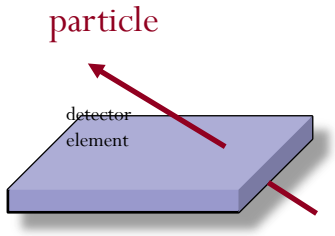
Electronics Simulation

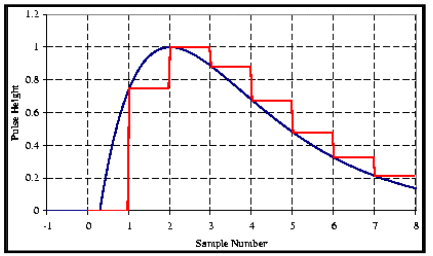
Simulate the response of the detector elements to the 'hits' from the electron.

Simulate the voltage pulse on the detector and how the detector electronics works.

The output of this stage is very similar to the raw data from the detector.

(but we keep the truth information).





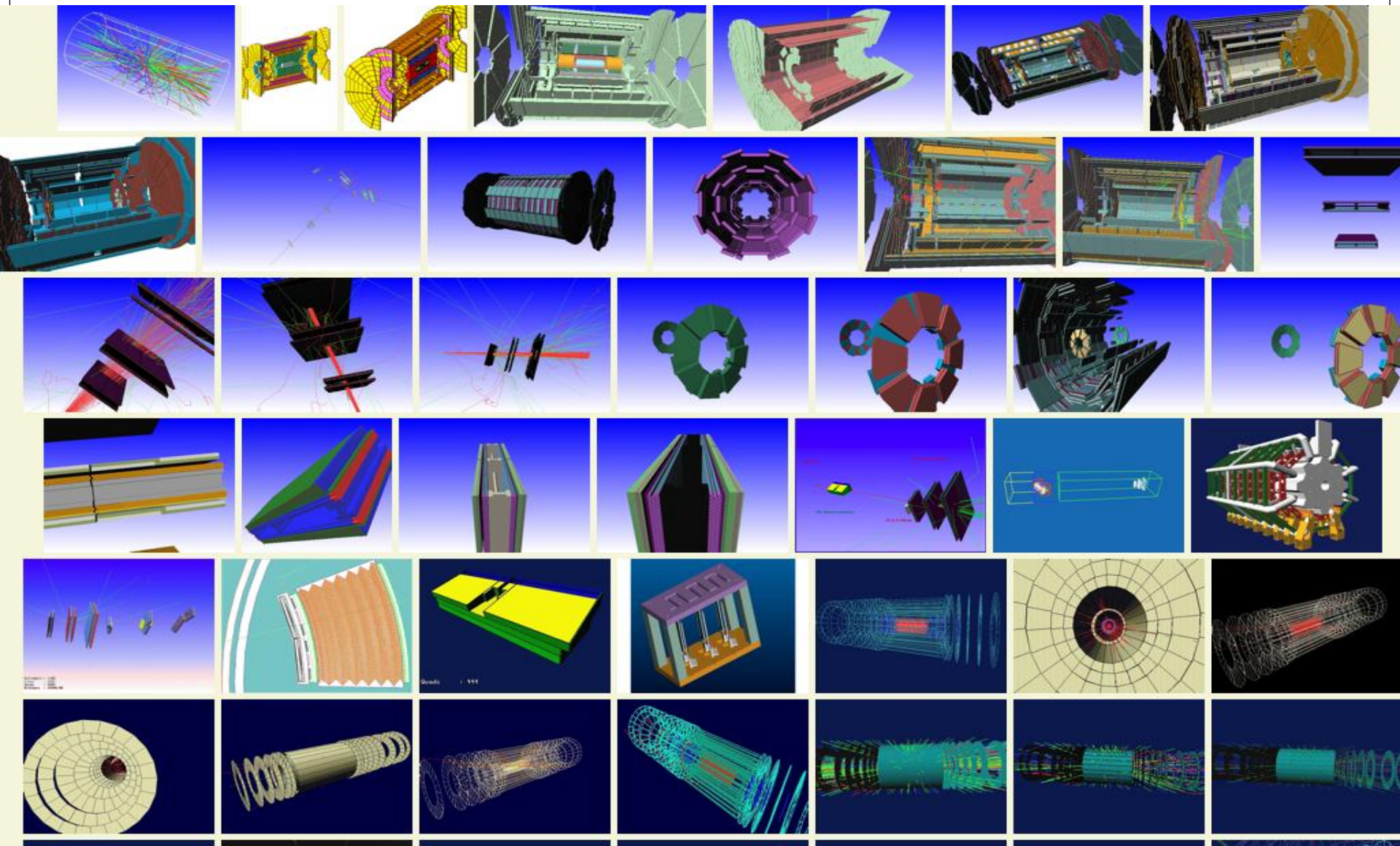
Detector simulation step is very CPU intensive. Requires huge computing resources.

J.Boyd

Geant4 toolkit on one slide

- Mature, extensible kernel
  - Powerful geometry modeler, E/B fields, track stacking
- Diverse set Physics models (mostly 2 alternatives)
  - e-/e+/gamma 10s eV to TeV
  - Hadron-nucleus interactions up to 1 TeV
  - Neutron interactions from thermal to 1 TeV
  - Ion-ion interaction from 100s MeV/n to 10 GeV/n
  - Optical, weak (decay of unstable and radioactivity)
- Tools for input, output, visualization, scripting
- Every increasing use
  - Over 2000 citations for G4 NIMA paper (2003)
- Product of collaboration of 90 contributors
  - Effort: HEP (75%), Biology/medical (15-20%), space (5-10%).
- Open Source: Distributed via web. G4 license since 2006.

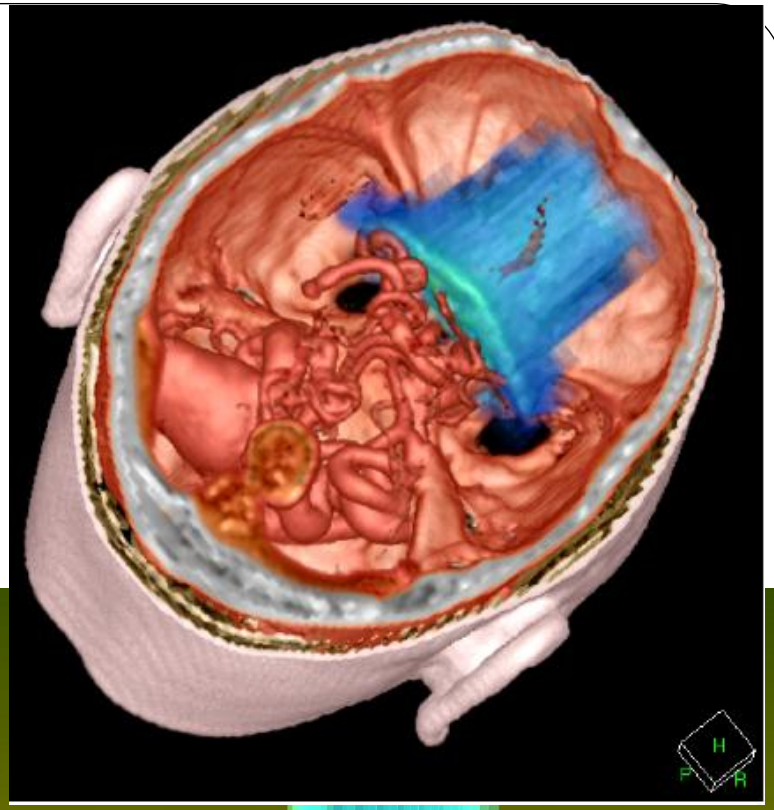
High Energy Physics Experiments

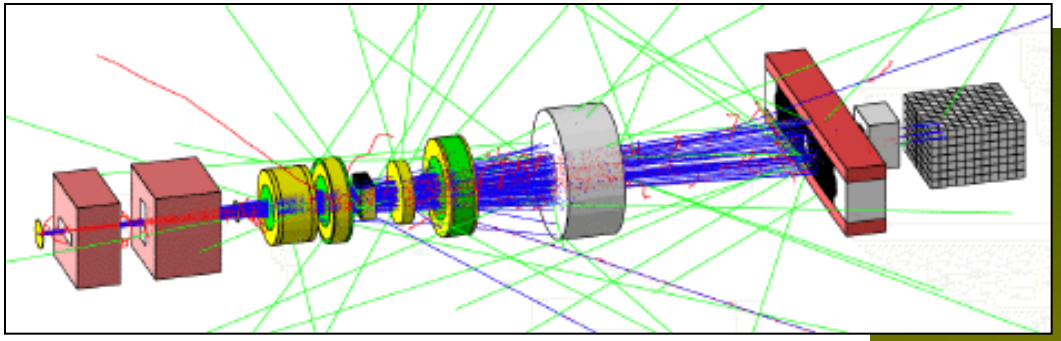


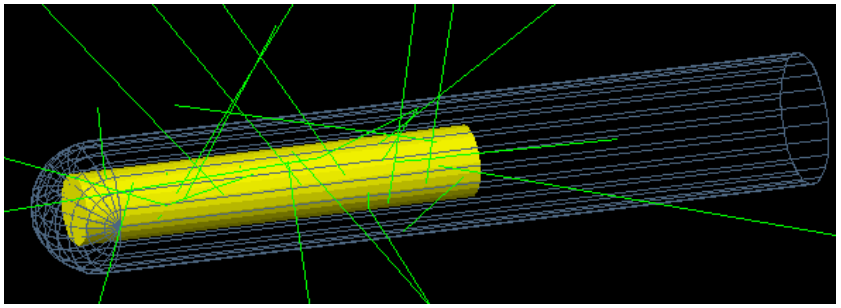
Space: radiation effects, science

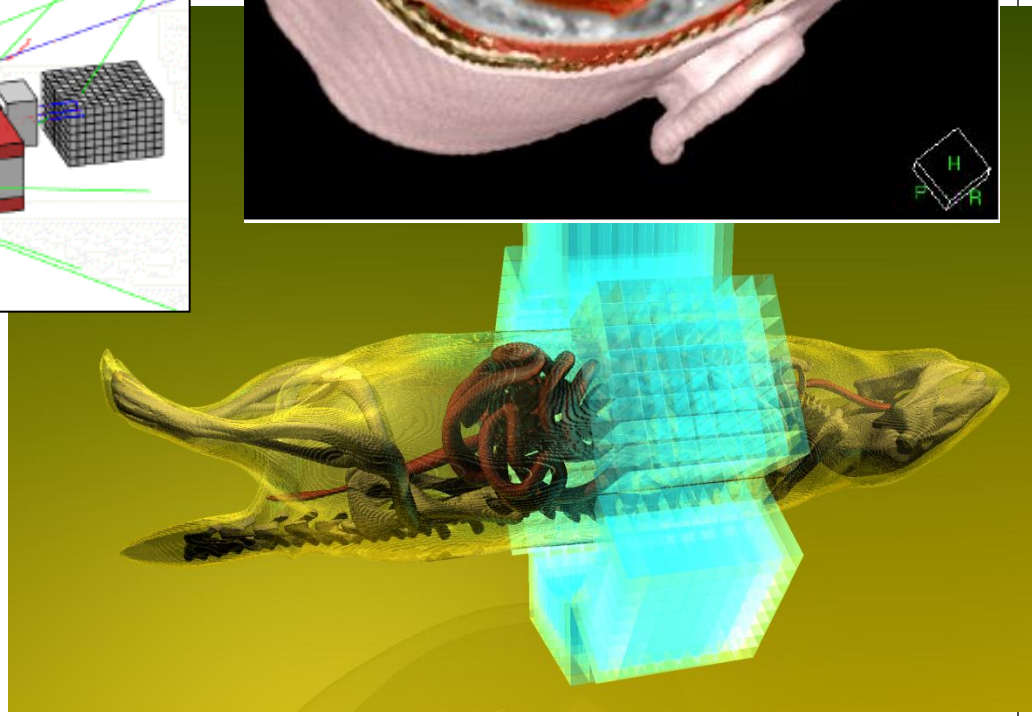
Geant4 @ Medical Science

- Four major use cases
  - Beam therapy
  - Brachytherapy
  - Imaging
  - Cell Irradiation study



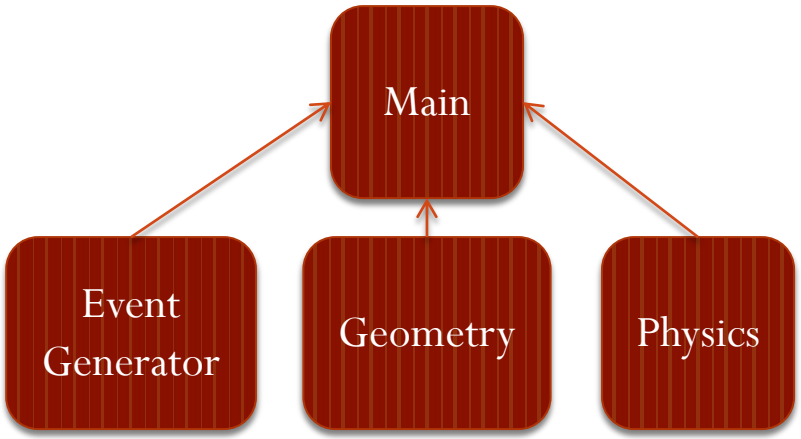






Some Basics

- Simplest Geant simulation needs three classes and a main to run
  - Geometry
  - Primary Generator Action
  - Physics List



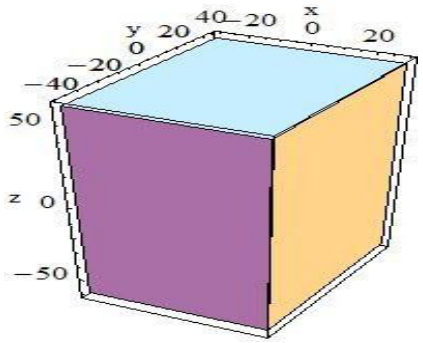
- Running Geant4
  - Hard code your commands into your main
  - Use a macros – a script containing a list of commands executed sequentially
  - Use a GUI

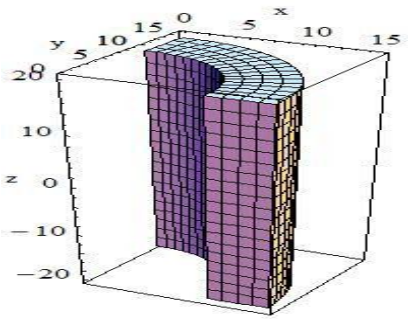
Defining your materials

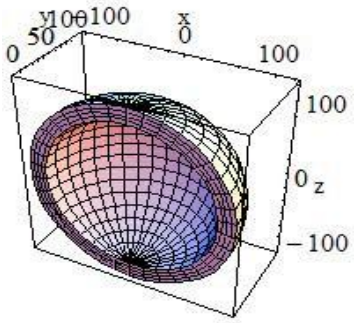
- Materials are defined based upon their chemical structure
- First define your element(s)
- Then define your material
- Assigned weighted quantities of element to material
- Two examples: Argon (element), Water (composite)
- Atomic number, molar mass, density
- Example: Water H 2O
  - Two ingredients : Two hydrogen , one oxygen
  - Define hydrogen
  - Define Oxygen
  - Define Water
  - Assign two hydrogen atoms and one oxygen atom to water
- See also G4NISTManager class, it allows to build materials (and elements) via names

Defining your world

- Define all objects within the world
  - Shape — SOLID (G4SOLID)
  - Size — LOGIC (G4LOGIC)
  - Material — PHYSICAL (G4PVPLACEMENT)
  - Position
- Use the classes provided by Geant4
  - Example: Box, Cylinder, Sphere
  - Many more existing classes, User's Guide (4.1.2)







Defining a source

- Two types covered here
  - Beams
  - Stationary sources
- Beams are defined in a source file in the Primary Generator Action
- Stationary sources can be defined there using random numbers to generate isotropic distributions
- Or use pre-defined class in a macro G4GeneralParticleSource
- All primary generators are defined in a user Primary Generator Action (a class derived from G4VUserPrimaryGeneratorAction);besides G4GeneralParticleSource there is G4ParticleGun which is more simple generator

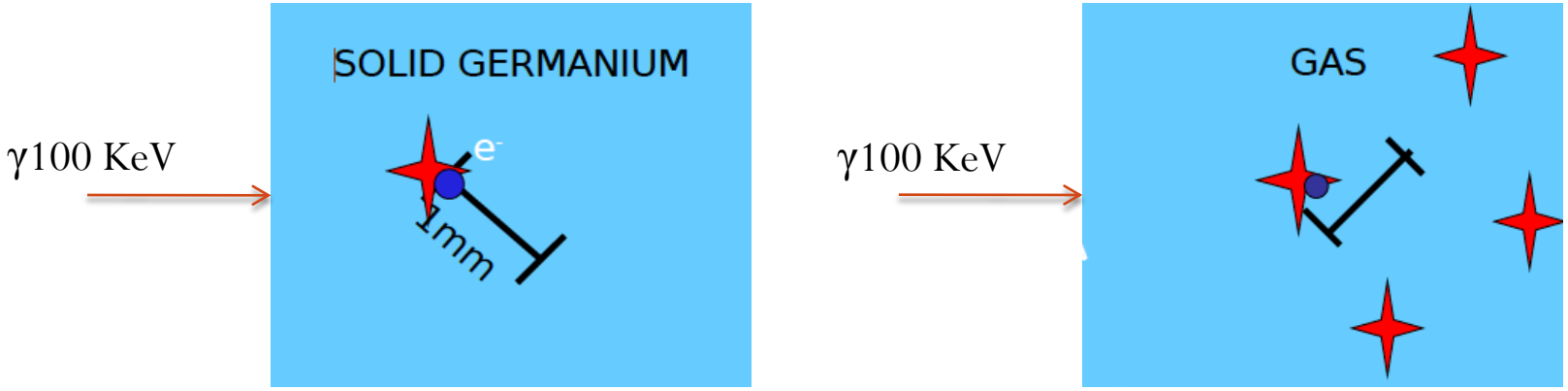
Physics List

- Geant4 doesn't "automatically" include any physics
- You need to tell Geant4 what physical interactions you are interested in
- Originates from particles physics background where "new physics" would need implementing
- Include:
  - Particles you want (bosons, mesons etc.)
  - Interactions you want (photoelectric, Compton, Pair production)
  - The correct energy range: different for the energy range you are interested in
  - Standard, low energy, very low energy
  - Beware of your cut values
- Geant4 provides a set of physics lists and it is recommended to users to start with one of these. See more details at:http://geant4.web.cern.ch/geant4/support/index.shtml --> Physics list

Physics List: Cuts

- Cut value defines the extent to which a particle is tracked
- Cuts are defined in distance (range)
- Converted into energy based on the material
- 100keV

Cut value = 1mm

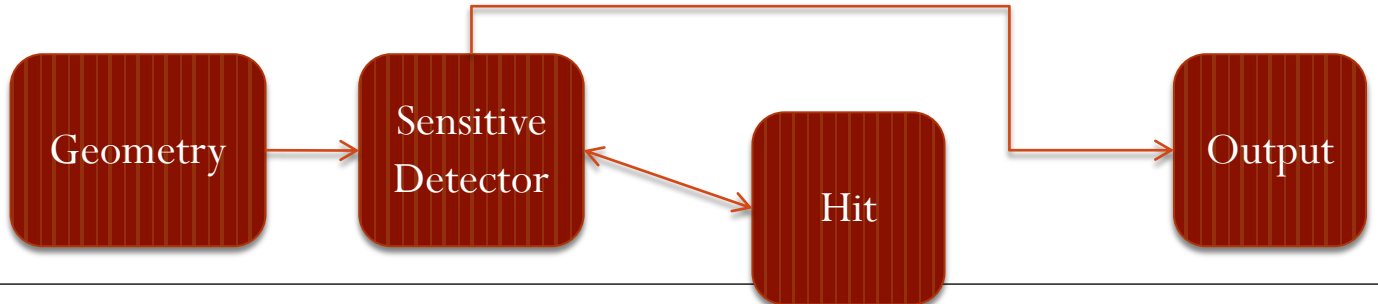


Electron won't be tracked
Distance < Cut

Electron will be tracked
Distance > Cut

Extracting information from Geant4

- Two additional classes (minimum):
  - Sensitive Detector
  - Hit
  - (Event Action) is useful, but not essential
- Which volumes are made sensitive is defined in the geometry class
  - At logical stage
- Hit defines the object of an interaction
  - Energy deposit, Position, Interaction type, Detector segment
- Hits will only be defined in sensitive detectors (not in passive volumes)
- Hits are assigned their attributes (Energy, position) not in the hit class, but in the sensitive detector class
- User can extract info at all stages of event processing: stepping, tracking, event, run action. Besides that he can use scorer classes which are a kind of ready to be used by sensitive detectors for accounting various quantities



Compare C++ codes

- Geometry: assigns a volume to be sensitive
- Sensitive detector:
  - builds a Hit
  - assigns it to HitCollection
- Hit class defines attributes of the hit
- Sensitive Detector either:
  - outputs HitCollection
  - or sends HitCollection to EventAction file
- EventAction:
  - necessary to process several HitCollections
  - examine several HitCollections, establish HitCollection

Geant4 is MORE…

- Kernel: 3 lectures
- Geometry: 4 lectures
- Materials: 1 lecture
- Physics: 3 lectures
- EM Physics: 2 lectures
- Hadron Physics: 3 lectures
- Physics Lists: 1 lecture
- Visualization: 3 lectures
- Primary particles: 1 lecture
- Analysis: 1 lectures
- Upgrading: 1 lecture
- User interface: 2 lectures
- Event biasing: 1 lectures
- User documentation and examples: 2 lectures

Suggestions (not only for Geant4)

DO's

- Consult the user guide with any problems or for more detail
- Consult the HyperNews forum if you get stuck
- Just start playing around with one of the examples
- If you get a bug in it you can't fix, you can always download it again
- Always think about what you need in terms of physics and cut values
- Do discard physics and events you don't need: you are the expert of your work

DON'T's

- Treat this lecture as a replacement of the User Guide:
- Geant4 is very complex and cannot be explained fully in few slides
- Don't wait until you are a C++ expert before you start
- Don't read the whole user guide before you start
- Don't assume Geant4 will "just get it right"
- Don't let too much detail slow you down

Use the Geant4 novice examples to start with
(sec. 9.1 of the Users Guide for Application developers)
http://geant4.fnal.gov/index_web/novice_examples_explained.shtml

ROOT in a Nutshell

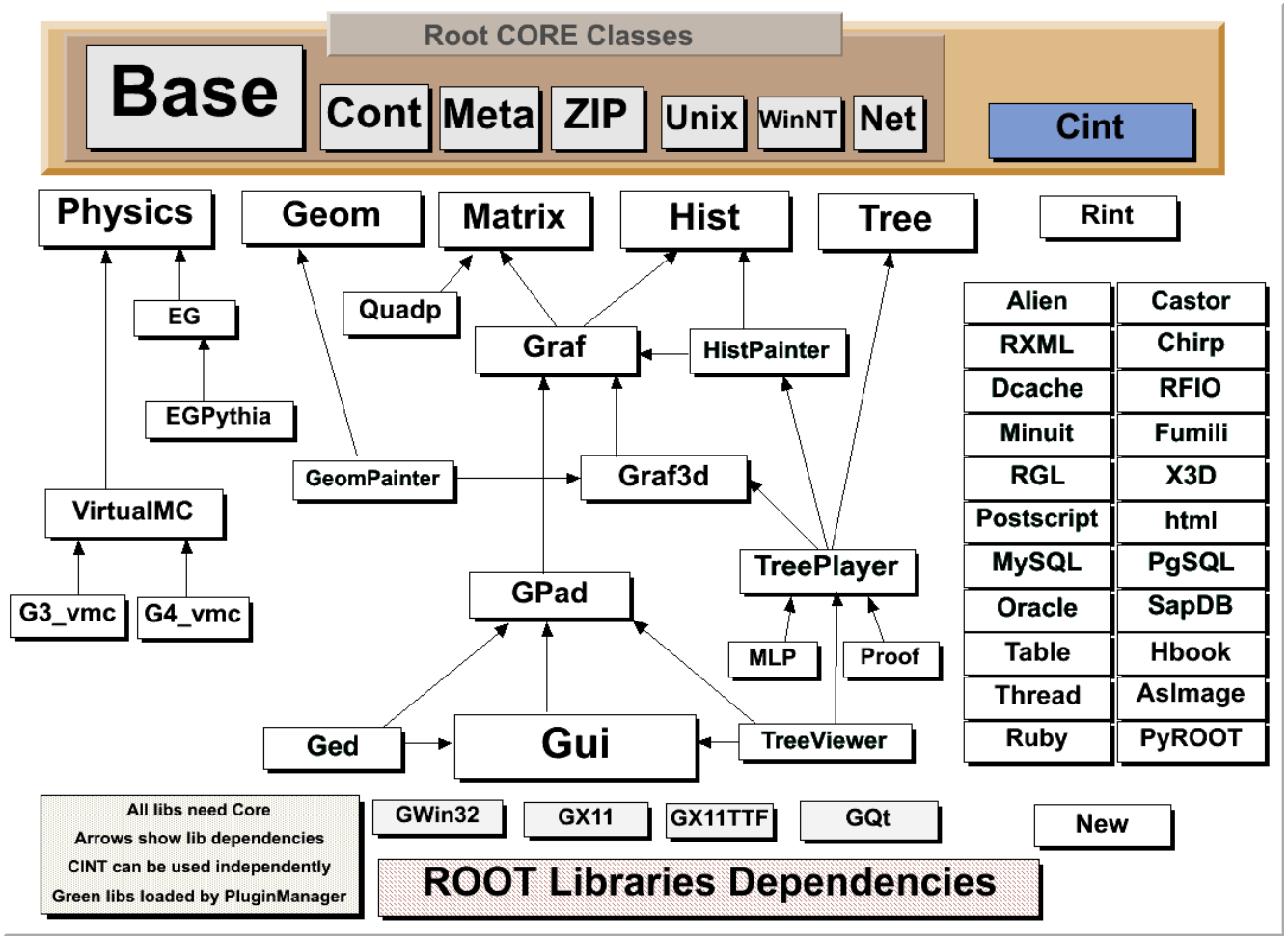
- ROOT is a large Object-Oriented data handling and analysis framework
  - Efficient object store scaling from KB's to PB's
- C++ interpreter
- Extensive 2D+3D scientific data visualization capabilities
- Extensive set of multi-dimensional histograming, data fitting, modeling and analysis methods
- Complete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



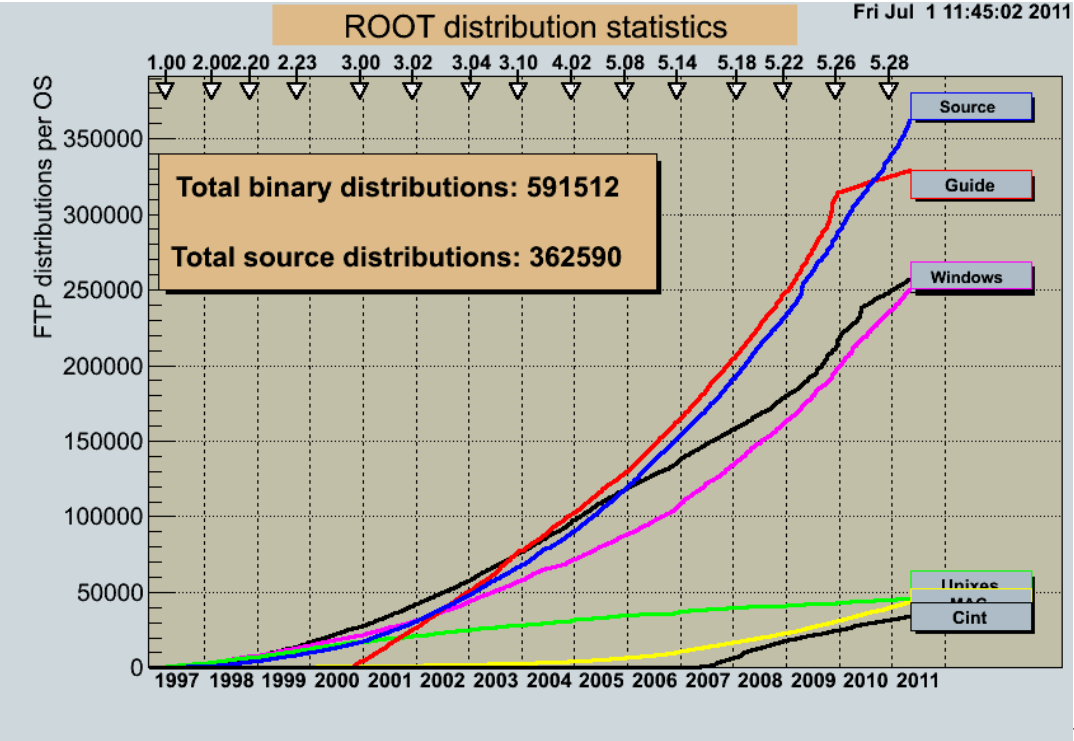
- Over 2,500 classes
- 3,000,000 lines of code
- CORE (8 Mbytes)
- CINT (2 Mbytes)
- Most libraries linked on demand via plug-in manager (only a subset shown)
- 100 shared libs

ROOT: An Open Source Project

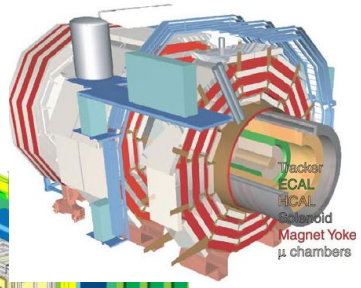
- The project was started in Jan 1995
- First release Nov 1995
- The project is developed as a collaboration between:
  - Full time developers:
    - 7 people full time at CERN (PH/SFT)
    - 2 developers at Fermilab/USA
  - Large number of part-time contributors (160 in CREDITS file)
  - A long list 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

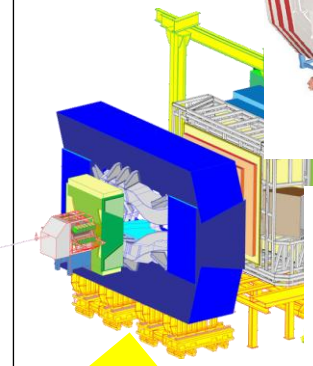
Some ROOT Statistics

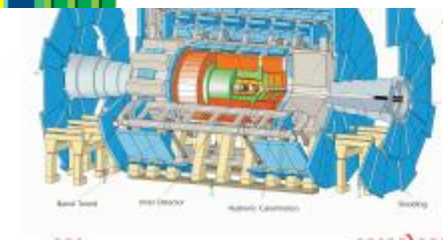
- ROOT binaries have been downloaded about 600,000 times since 1997
- The estimated user base is about 20,000 people



ROOT Application Domains







General Framework

Data Analysis & Visualization

Data Storage: Local, Network

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 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!

Unnamed Macros

- No function, just statements

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

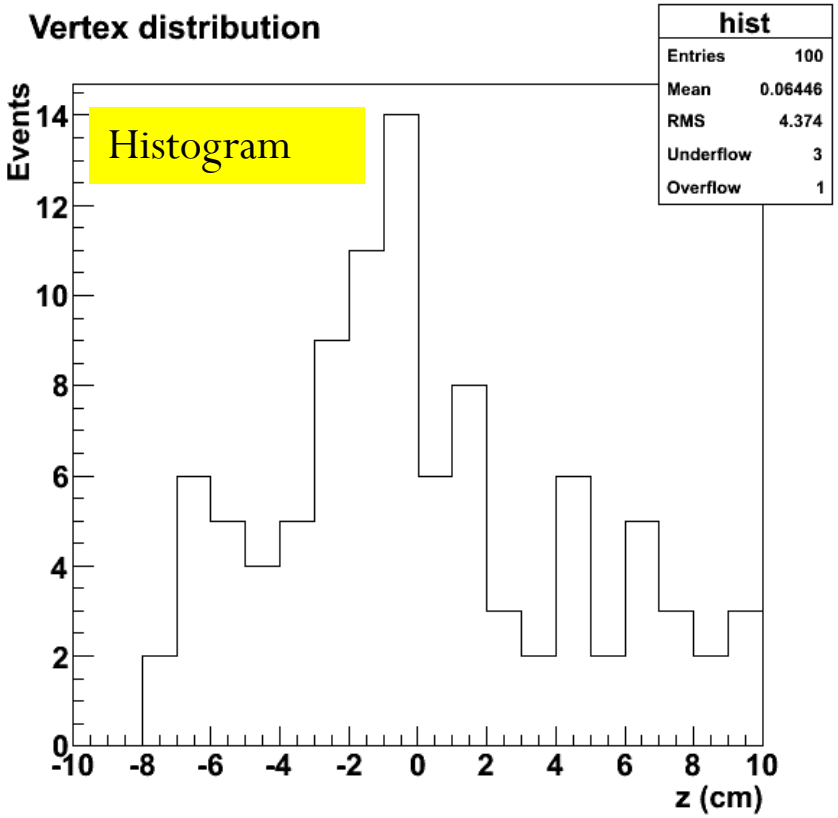

saved in mymacro.C

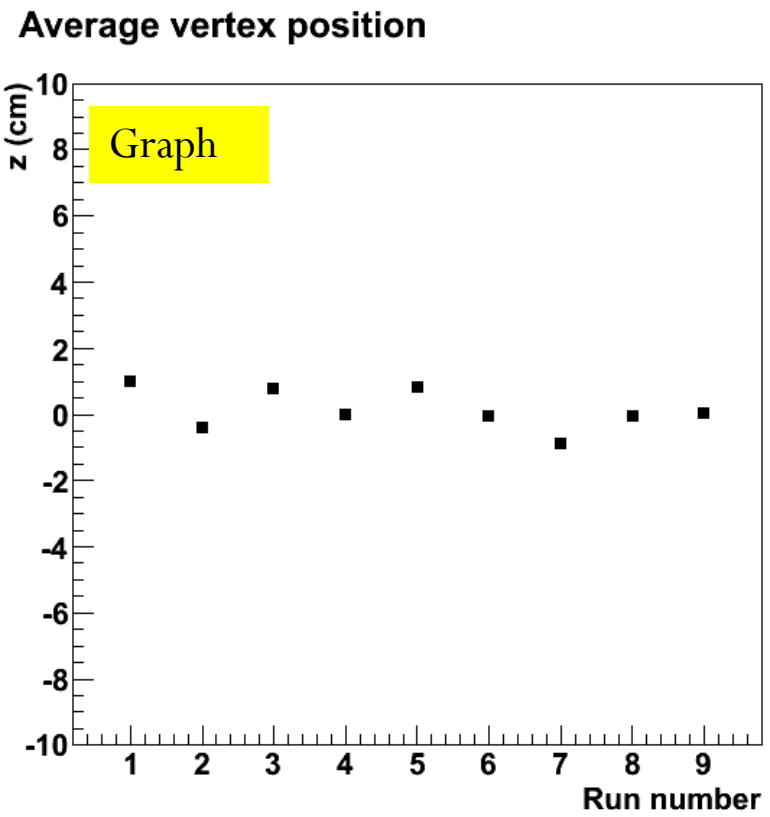
- Execute with **root [0] .x mymacro.C**
  - No functions, thus no arguments
- Named macro recommended!

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

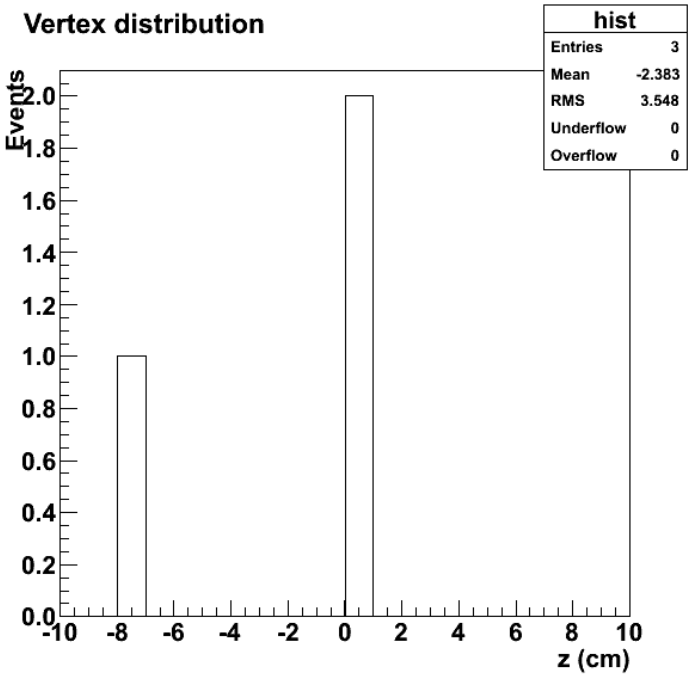
Histograms

- 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)

Histogram Example

```
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

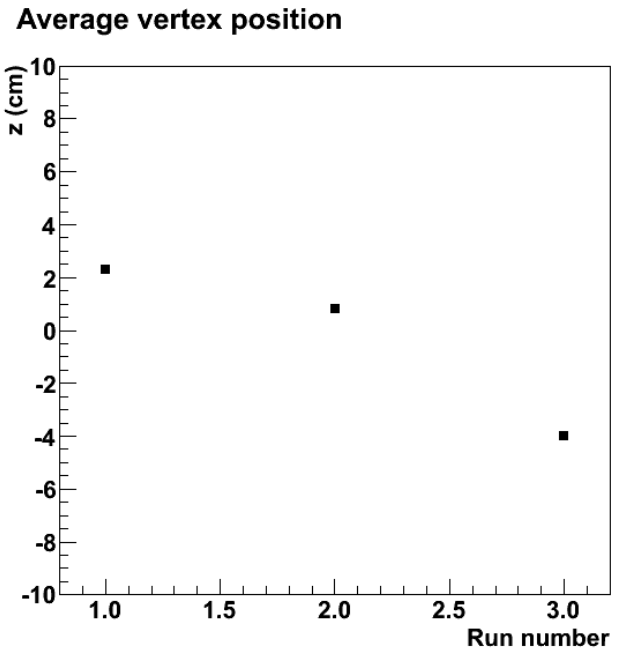


Graphs

- 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");
```



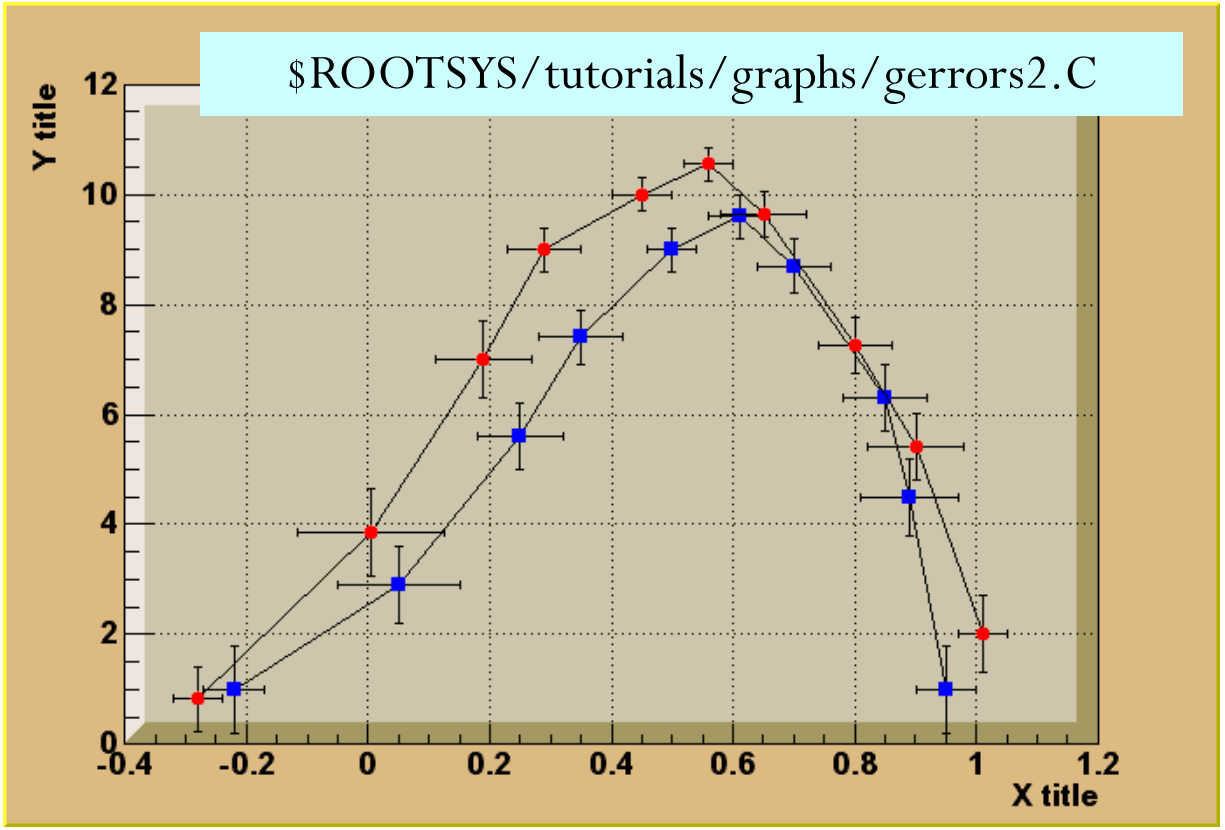
Graphs (2)

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

TGraph(n,x,y)

TCutG(n,x,y)

TMultiGraph

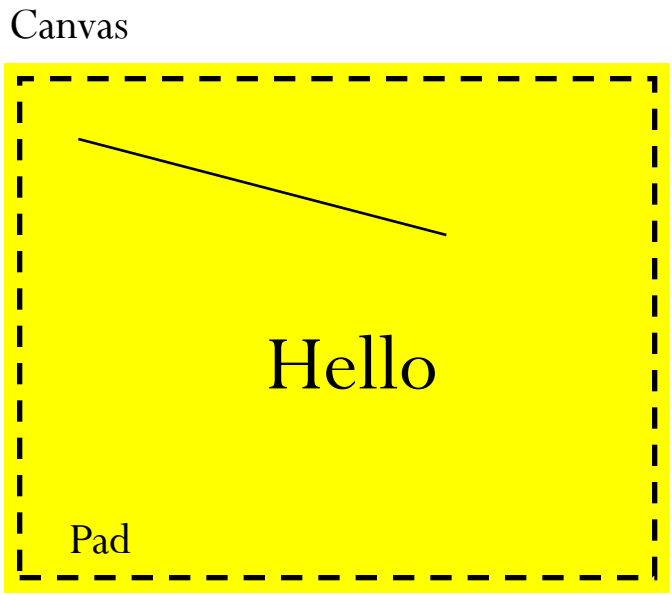


TGraphAsymmErrors(n,x,y,exl,exh,eyl,eyh)

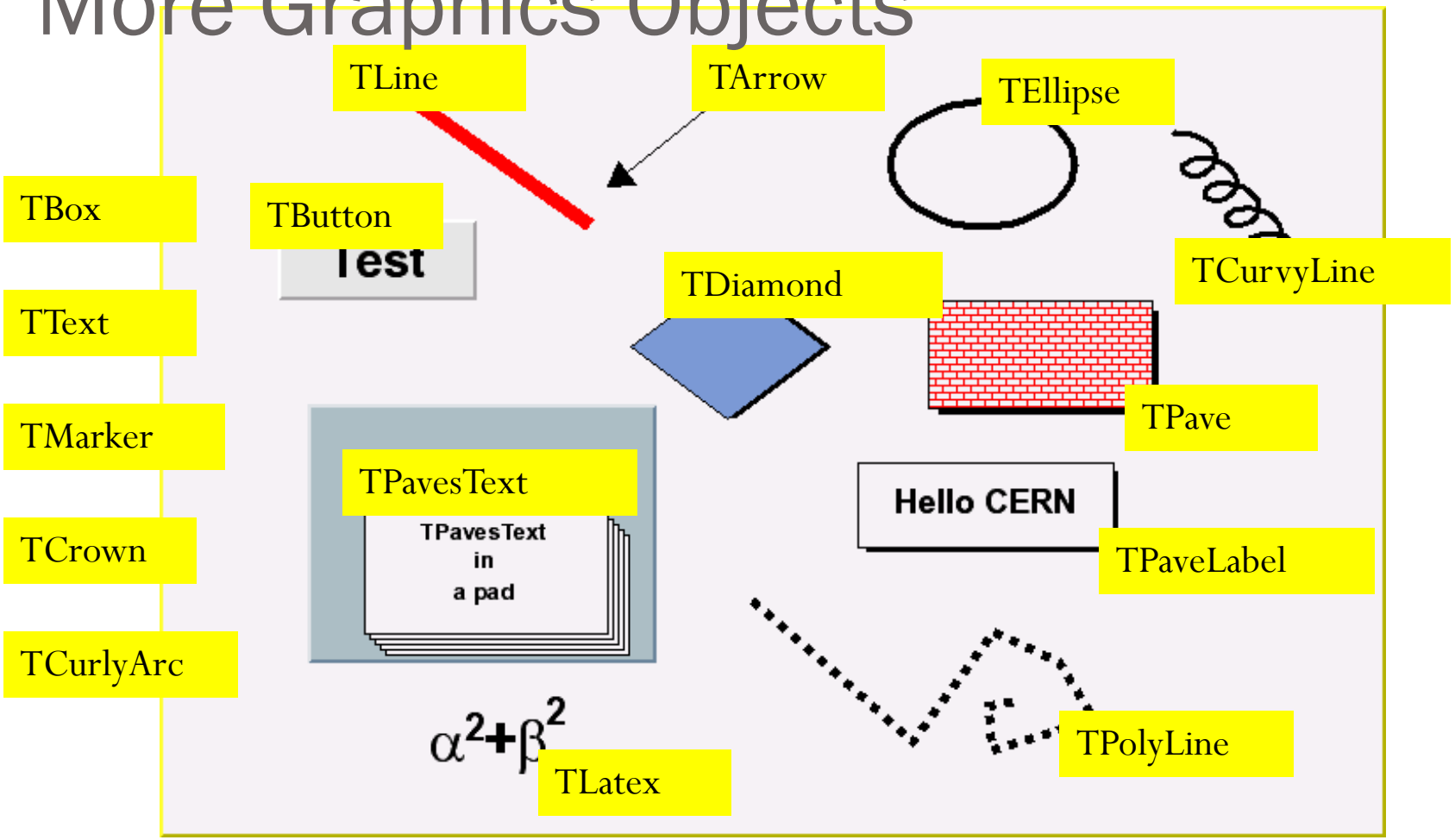
Graphics Objects

- 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()
```



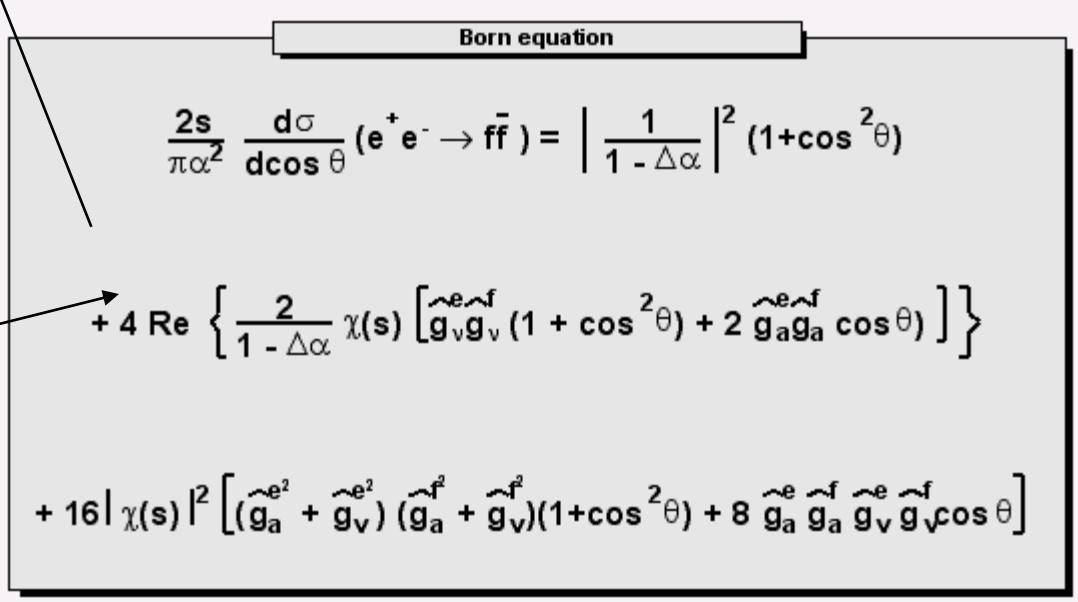
More Graphics Objects



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

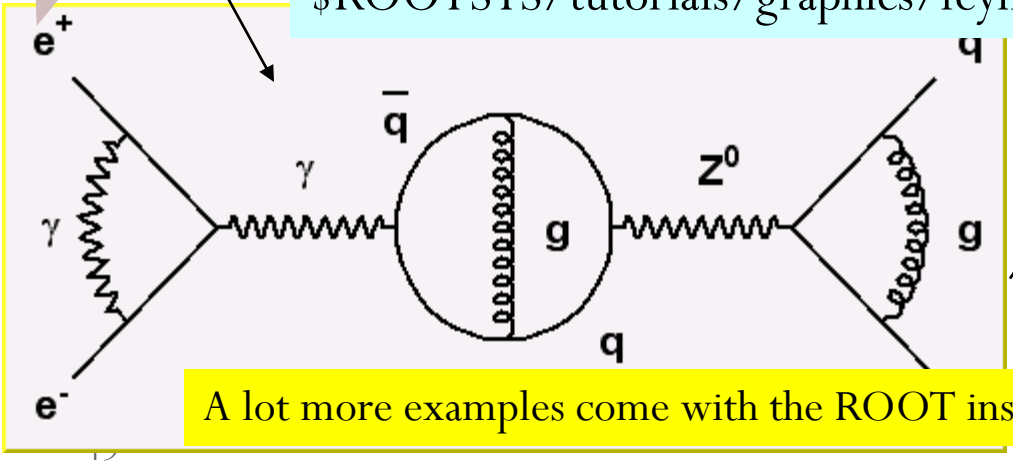
Full LateX support on screen and postscript

$ROOTSYS/tutorials/graphics/latex3.C



Formula or diagrams can be edited with the mouse

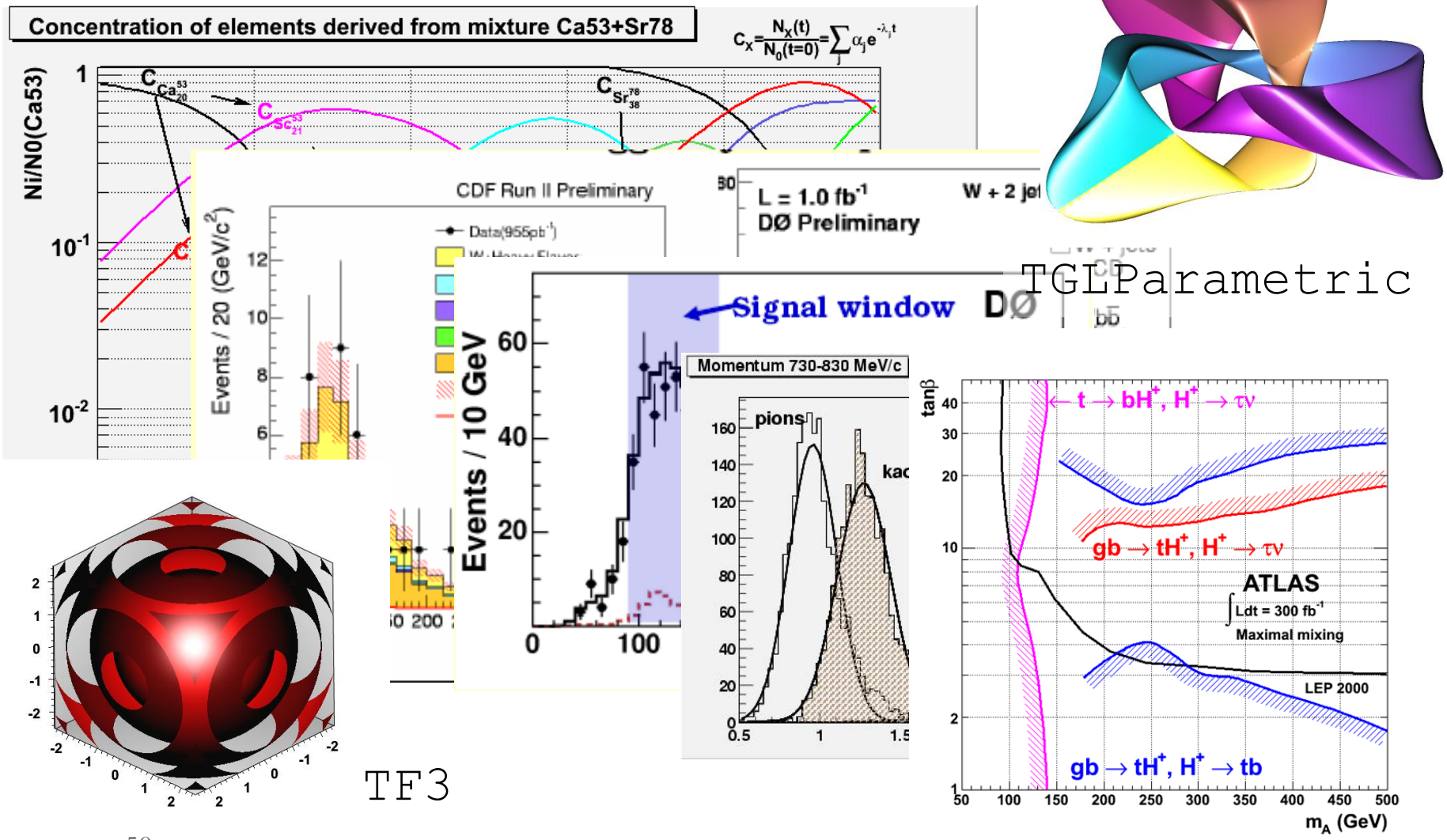
$ROOTSYS/tutorials/graphics/feynman.C



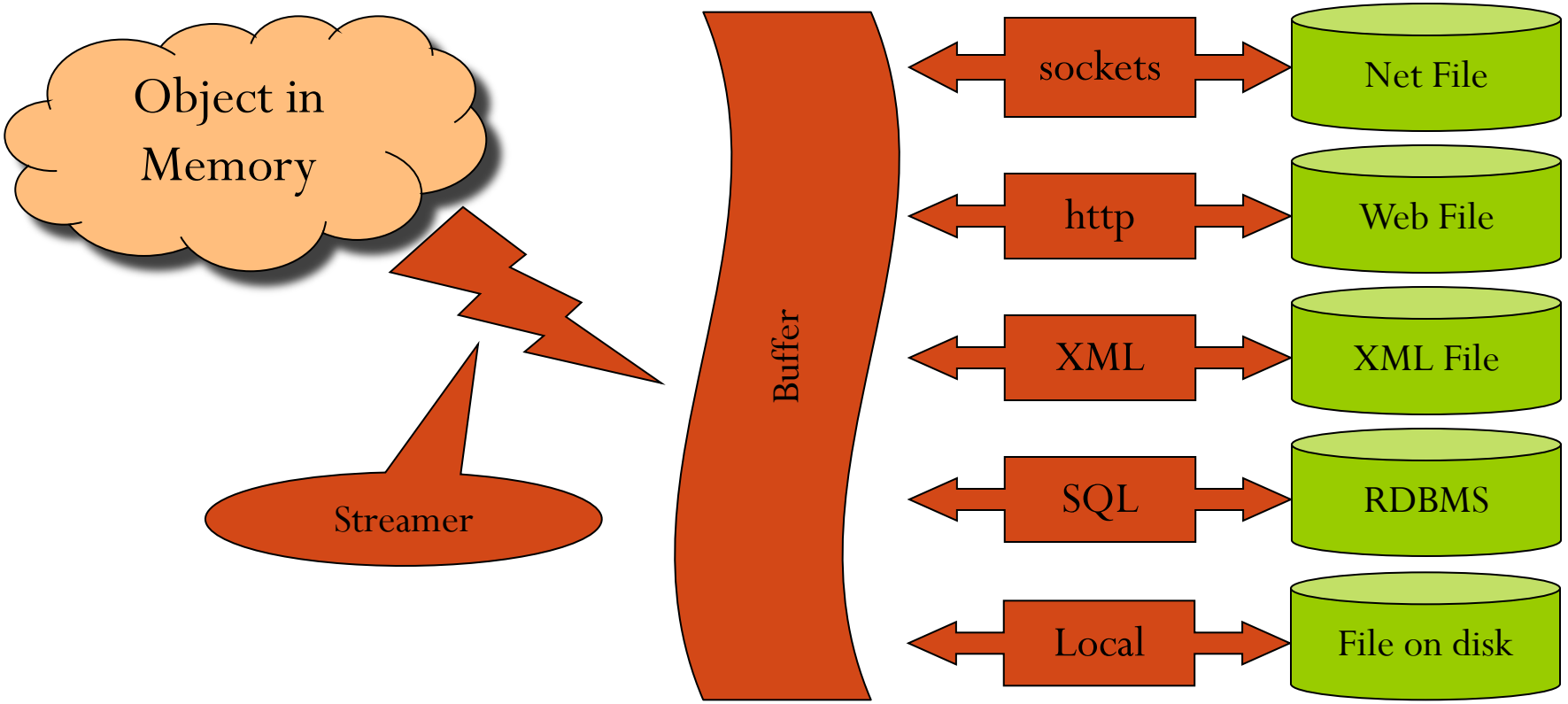
TCurlyArc
TCurlyLine
TWavyLine
and other building blocks for Feynmann diagrams

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

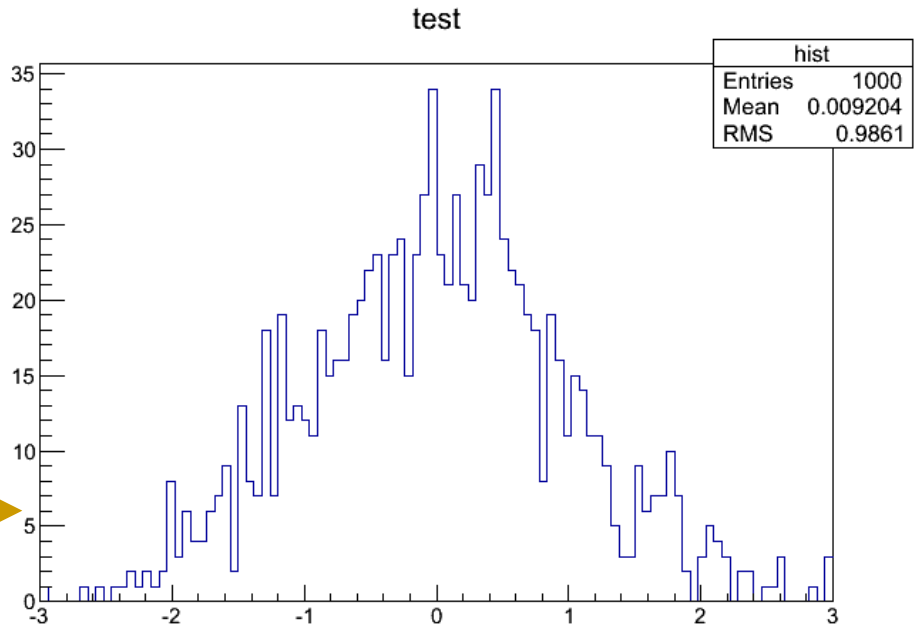
- TFile is the class to access files on your file system (and elsewhere)
- A TFile object may contain directories (TDirectory), like a Unix file system
- ROOT files are self describing
  - Dictionary for persistent classes written to the file
- Support for Backward and Forward compatibility
- Files created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

File Example

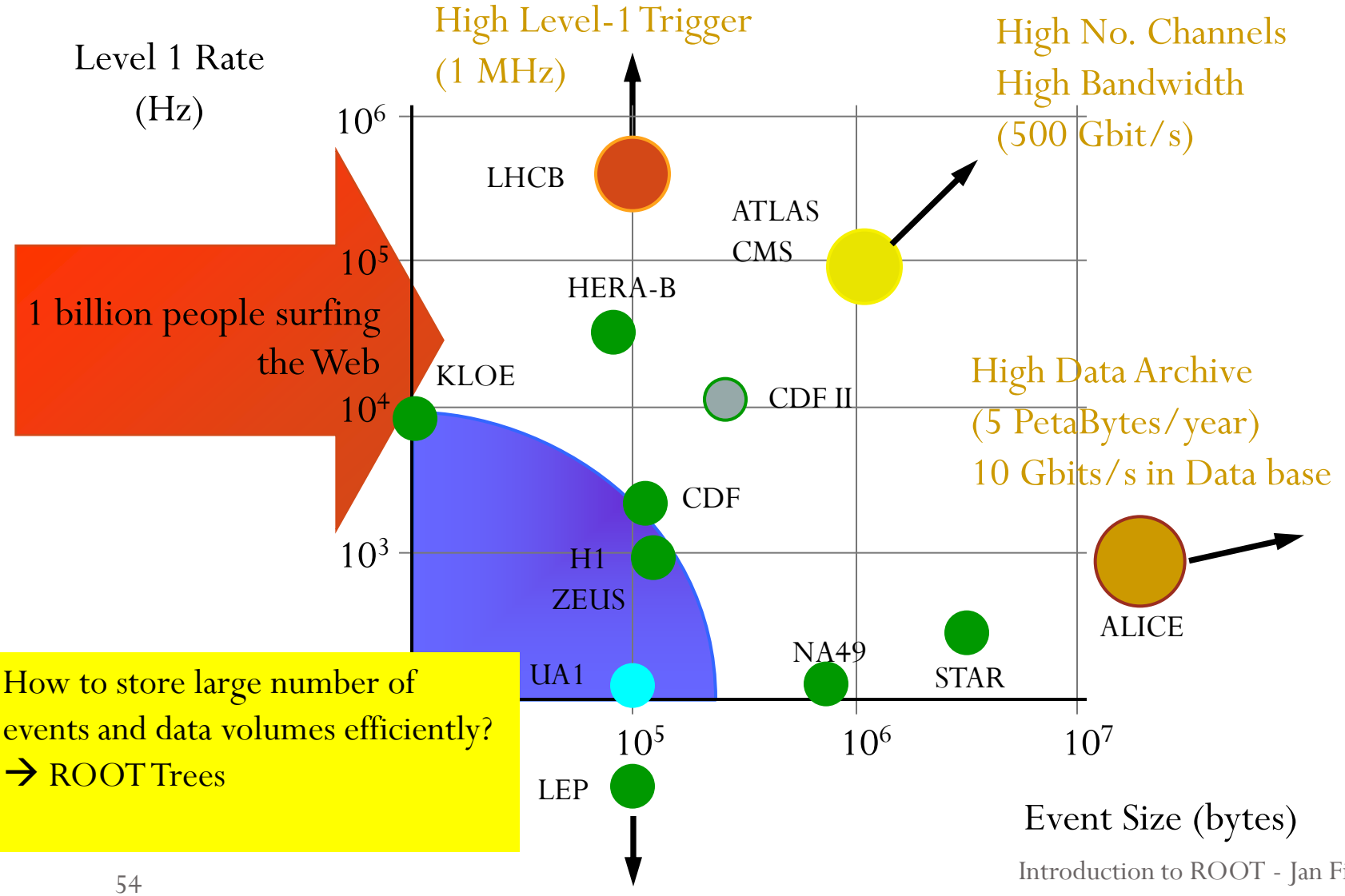
```
void keyWrite() {
  TFile f("file.root", "new");
  TH1F h("hist", "test", 100, -3, 3);
  h.FillRandom("gaus", 1000);
  h.Write()
}
```

This works as well for your own class!

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



LHC: How Much Data?

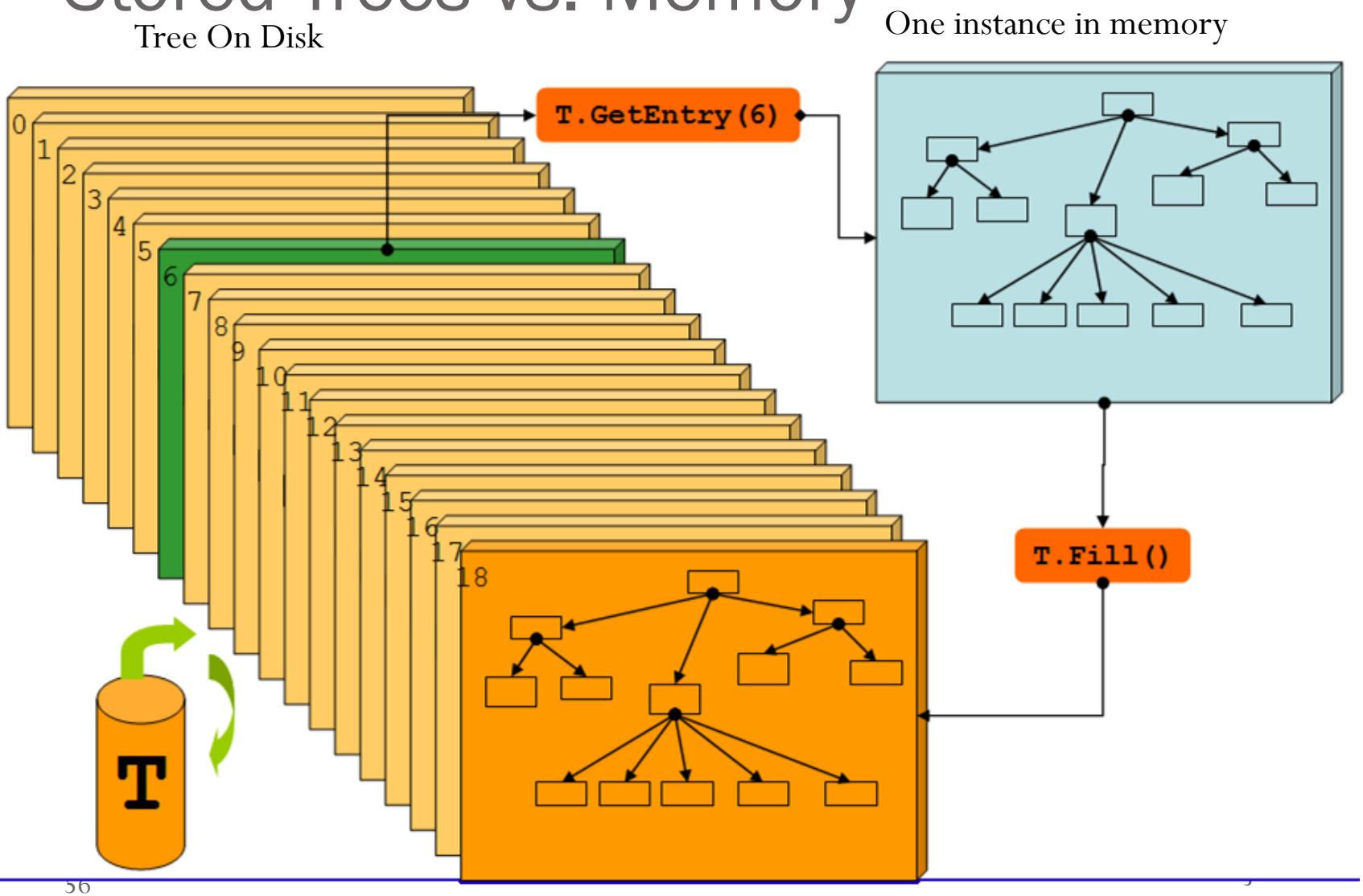


How to store large number of events and data volumes efficiently?
→ ROOT Trees

What is a ROOT Tree?

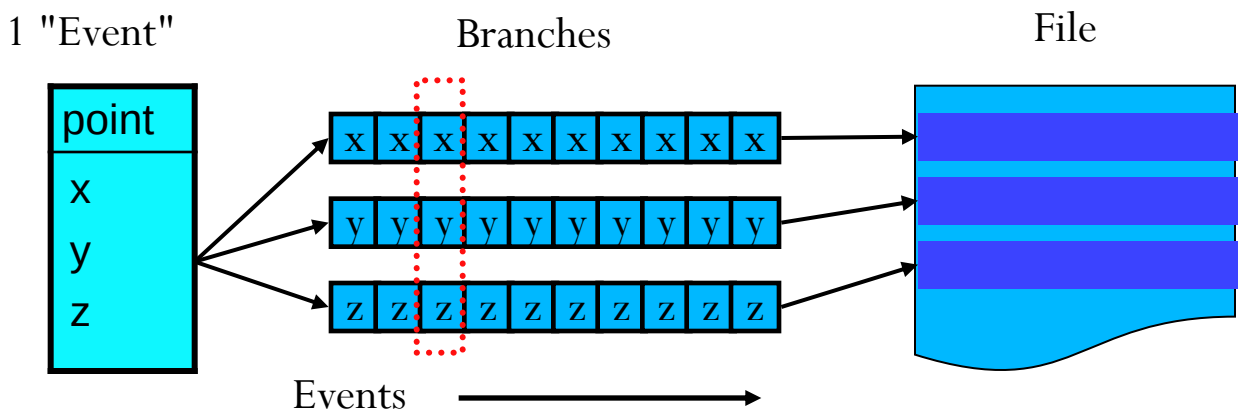
- Trees have been designed to support very large collections of objects. The overhead in memory is in general less than 4 bytes per entry.
- Trees allow direct and random access to any entry (sequential access is the most efficient)
- Trees are structured into branches and leaves. One can read a subset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

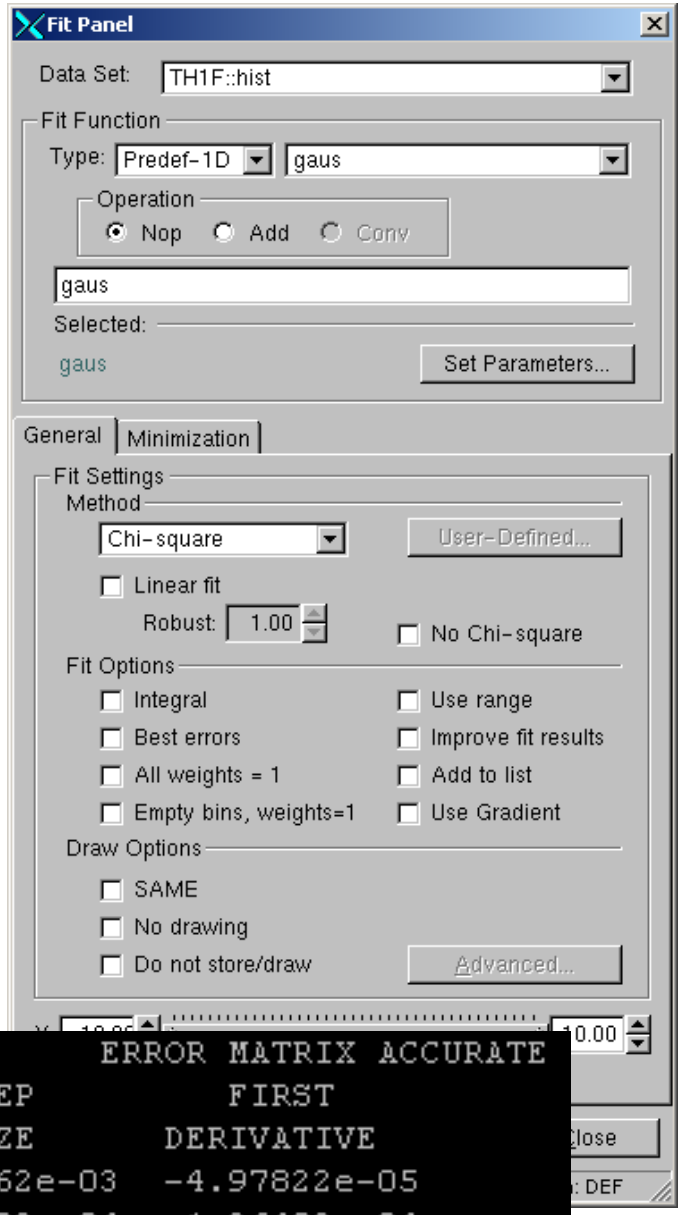
```
Int_t nentries = tree->GetEntries();
for (Int_t i=0;i<nentries;i++) {
  tree->GetEntry(i);
  cout << myEvent->GetMember() << endl;
}
```

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....

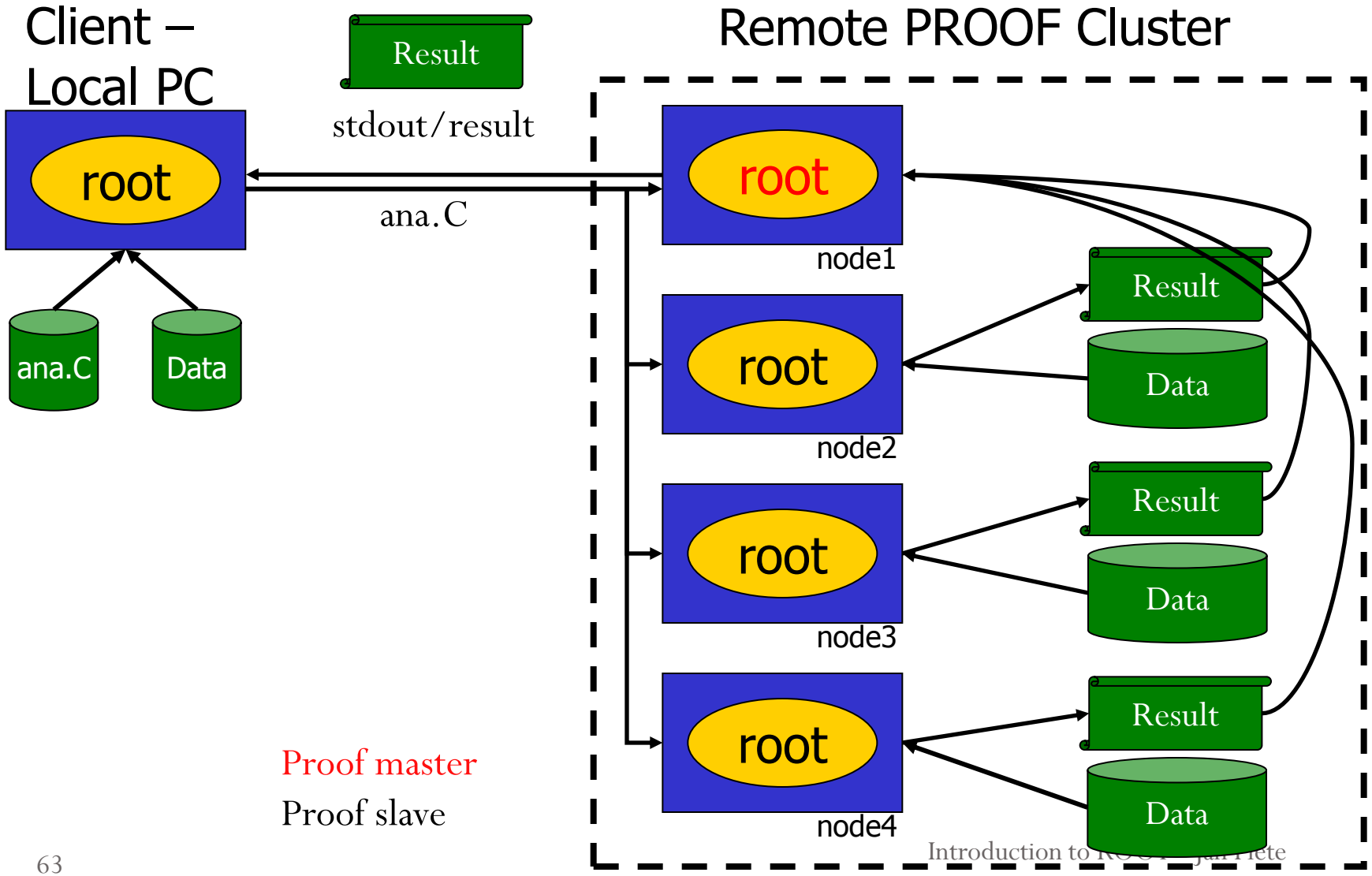
- In this talk, I presented the most basic classes typically used during physics analyses
- ROOT contains many more libraries, e.g.
  - FFT library
  - Oracle, MySQL, etc interfaces
  - XML drivers
  - TMVA (Multi Variate Analysis)
  - GRID, networking and thread classes
  - Interfaces to Castor, Dcache,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

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



Questions