The ROOT System

A Data Access & Analysis Framework

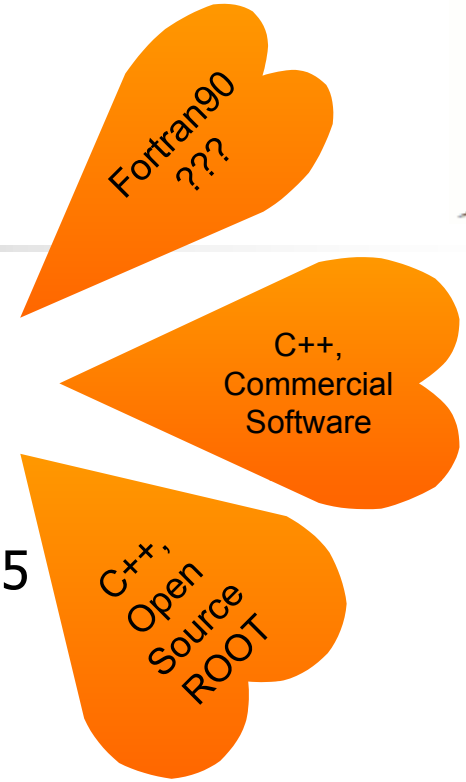
EUSO meeting LAPP

October 4 2001

René Brun

http://root.cern.ch

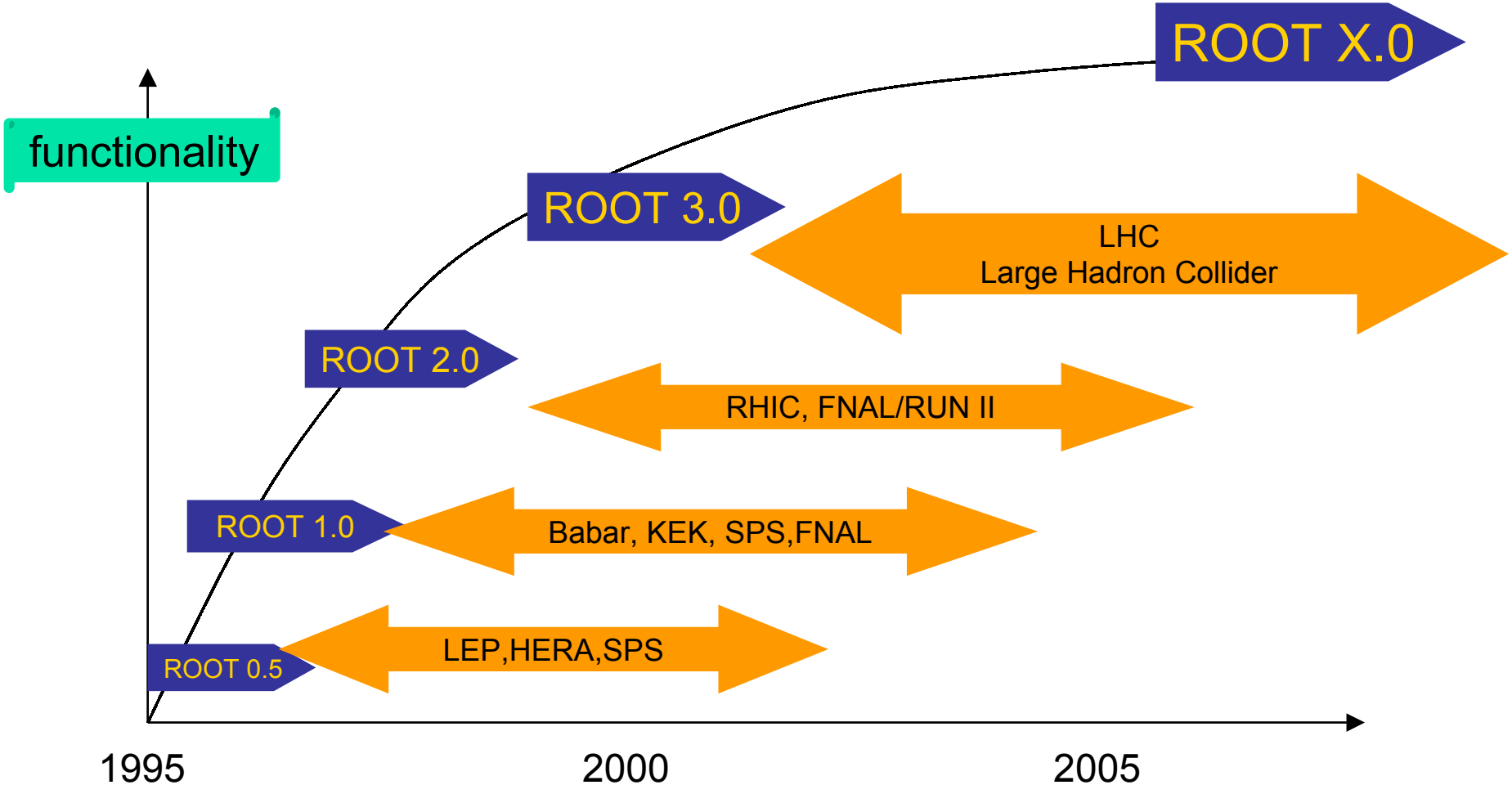
Project History



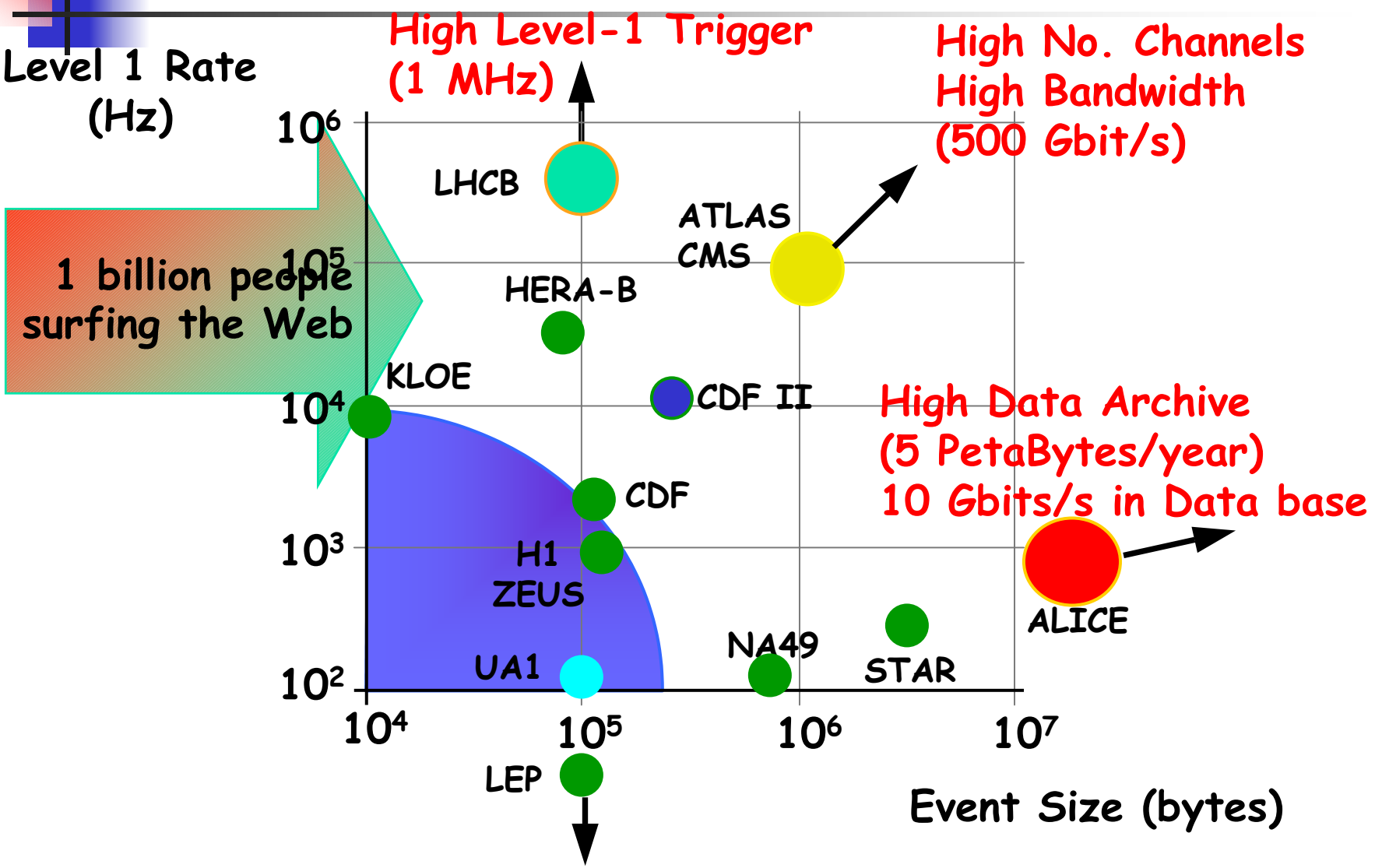
In 1994, fundamental divergence of opinions in Application Software group in IT. The PAW/Geant3 team is dismantled.

- Jan 95: Thinking/writing/rewriting/???
- November 95: Public seminar, show Root 0.5
- Spring 96: decision to use CINT
- Jan 97: Root version 1.0
- Jan 98: Root version 2.0
- Mar 99: Root version 2.21/08 (1st Intl Root workshop FNAL)
- Feb 00: Root version 2.23/12 (2nd Intl Root workshop CERN)
- Sep 00: Root version 2.25/03
- Dec 00: Root version 3.00/01
- Jun 01: 3rd International Root workshop at FNAL

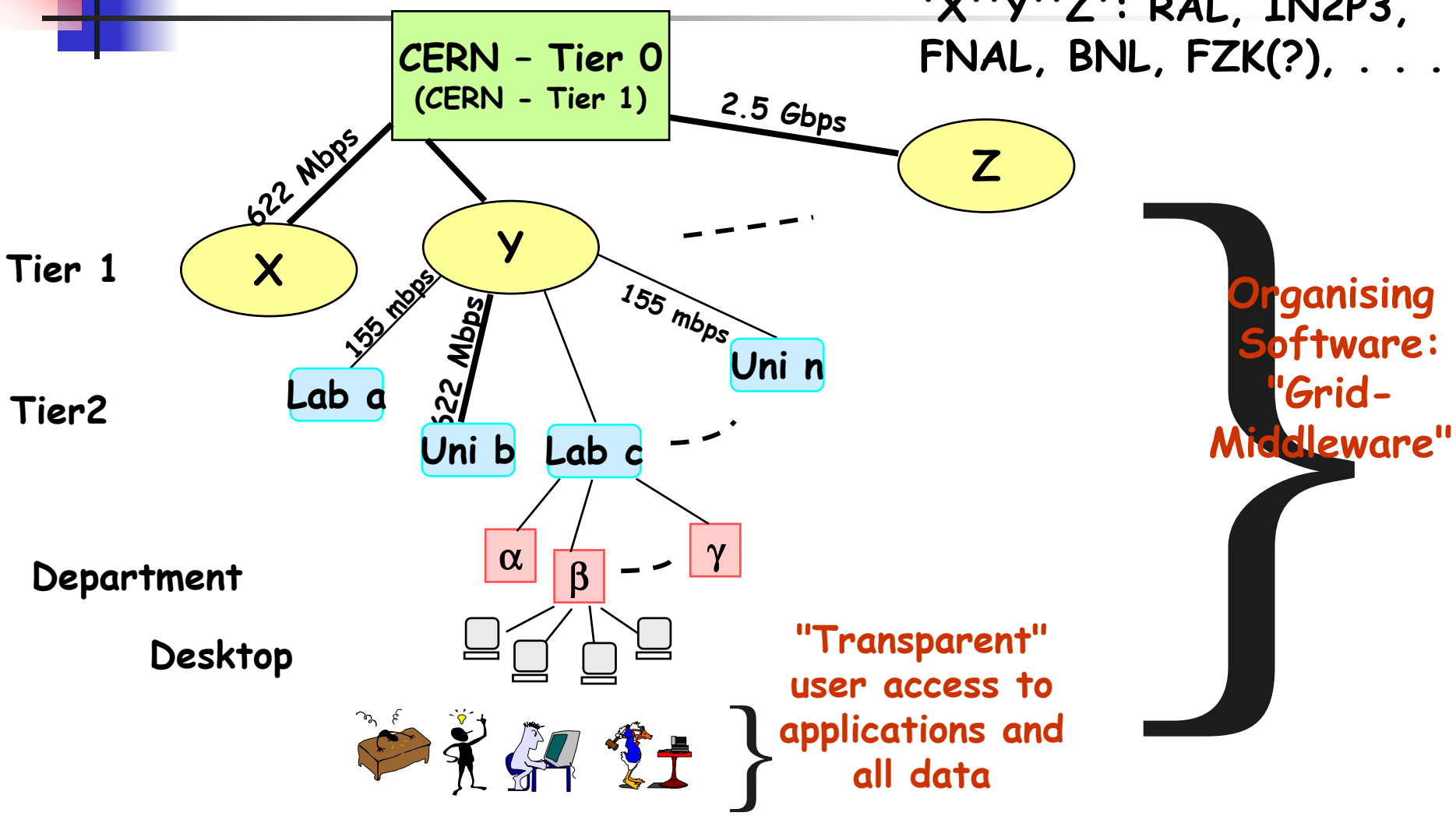
The ROOT Project



How Much Data is Involved?



LHC Computing - a Multi-Tier Model



Languages for Data Analysis

- Data analysis requires an efficient access to objects (both data and functions).
- It requires a powerful programming language:
  - in interpreted AND compiled mode
  - Transition from interpreted mode to compiled mode must be smooth and transparent.
- A scripting language (eg Python) is not the solution
- Java could be a candidate. However, severe performance penalty. I/O is far too slow.
- C++, the only realistic choice.

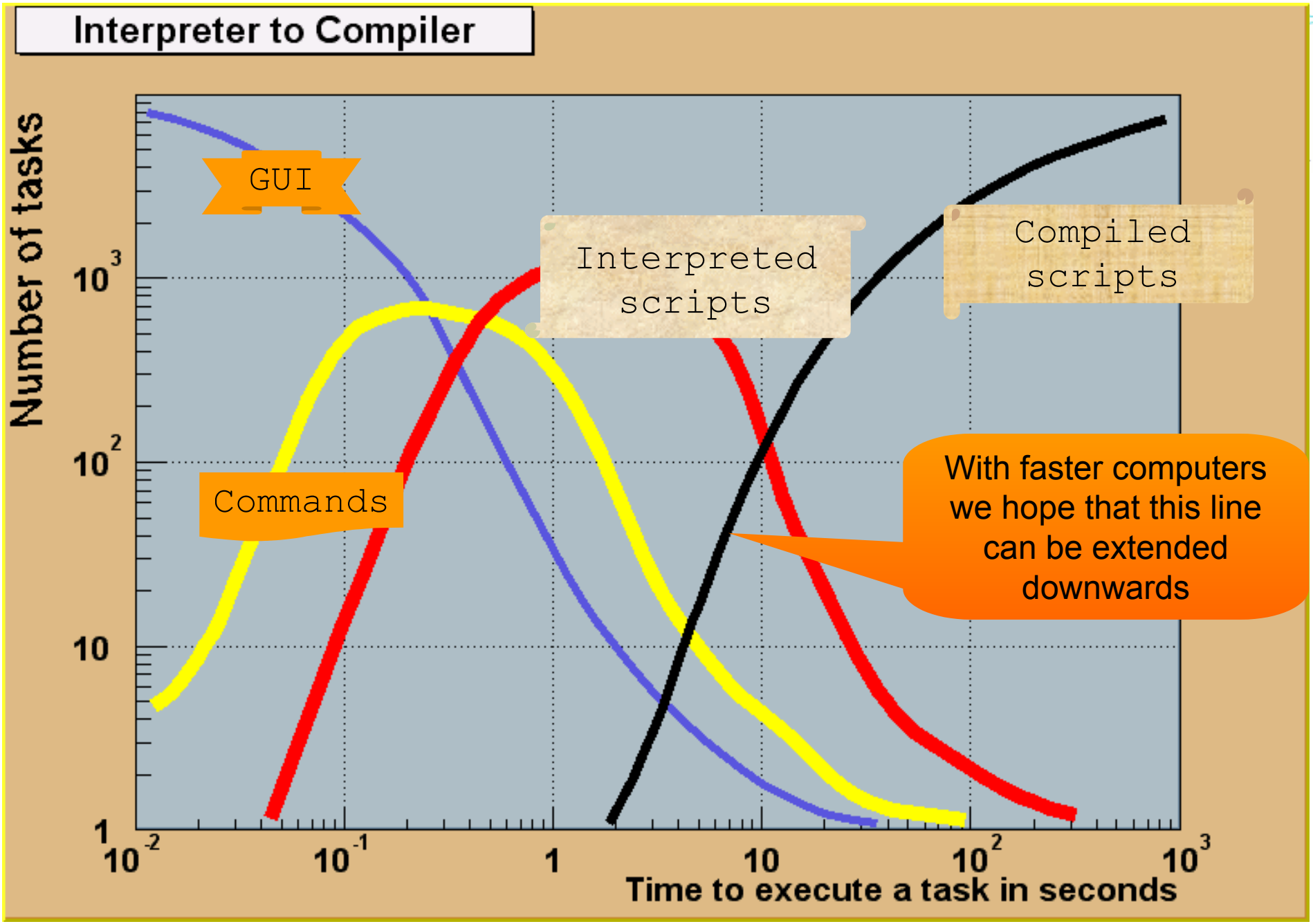
A Scripting Language ?

With today desktop machines, it takes between a few seconds and one minute to compile and dynamically link a realistic analysis script.

With computers becoming faster and faster, one may hope that in a few years from now, dynamic compilation and linking will become affordable for an increasing number of tasks.

Having the same language for the interpreted and the compiled codes will be a tremendous advantage. On the other hand, nobody will trust results produced by a pure interpreted language.

An interpreted language is fundamental for tasks that must be executed rapidly, such as short scripts edited very frequently or all the tasks called via the graphical user interface


Interpreter to Compiler
Number of tasks
10^3
10^2
10
1
GUI
Interpreted scripts
Compiled scripts
Commands
With faster computers we hope that this line can be extended downwards
10^{-2}
10^{-1}
1
10
10^2
10^3
Time to execute a task in seconds

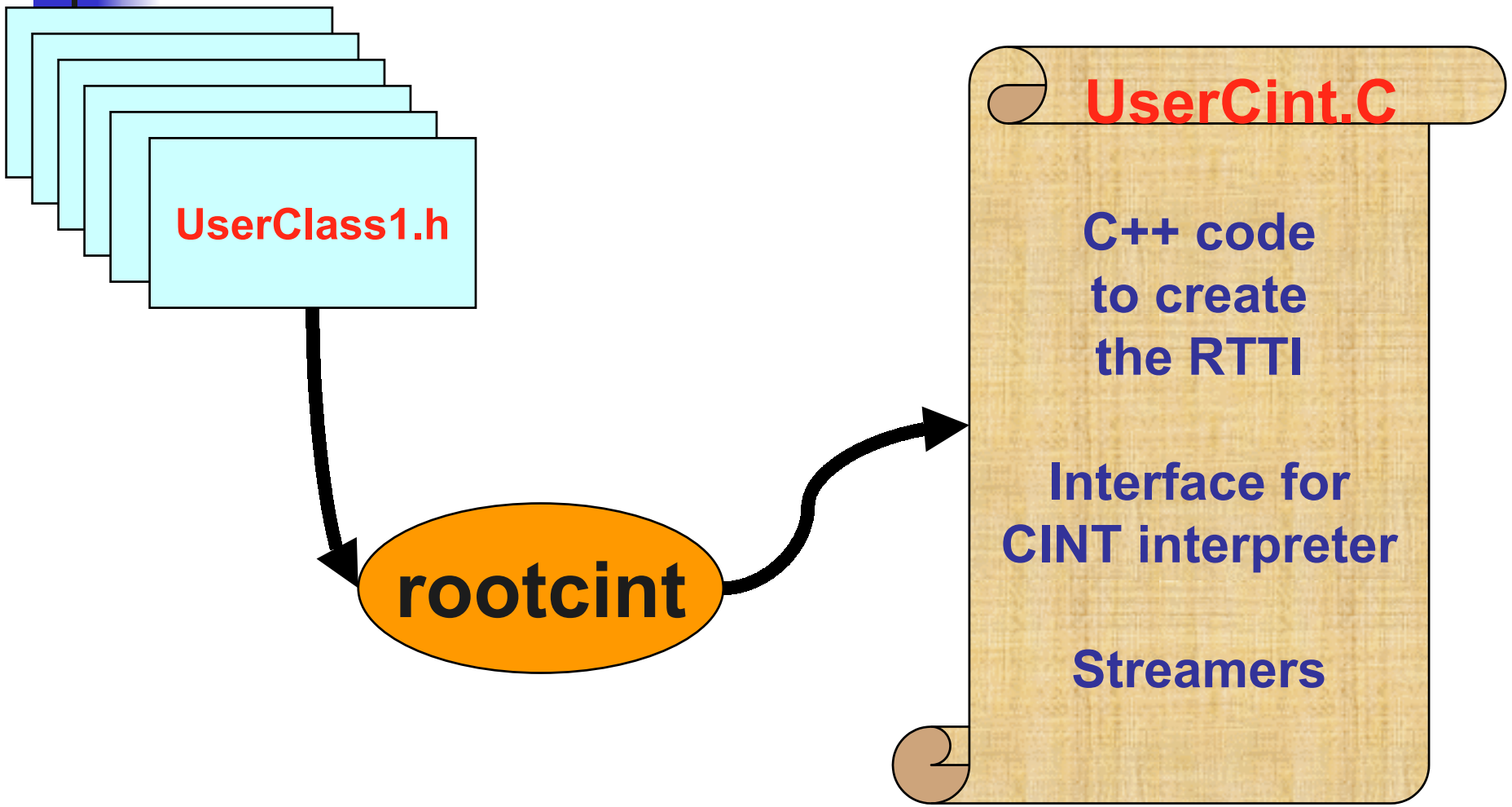
The Choice of CINT

Our goal was to combine the advantages of an interpreted and/or compiled language in one single framework. To achieve this goal, we had to develop a powerful object persistency system with as few limitations as possible to support the main stream proposed OO language C++.

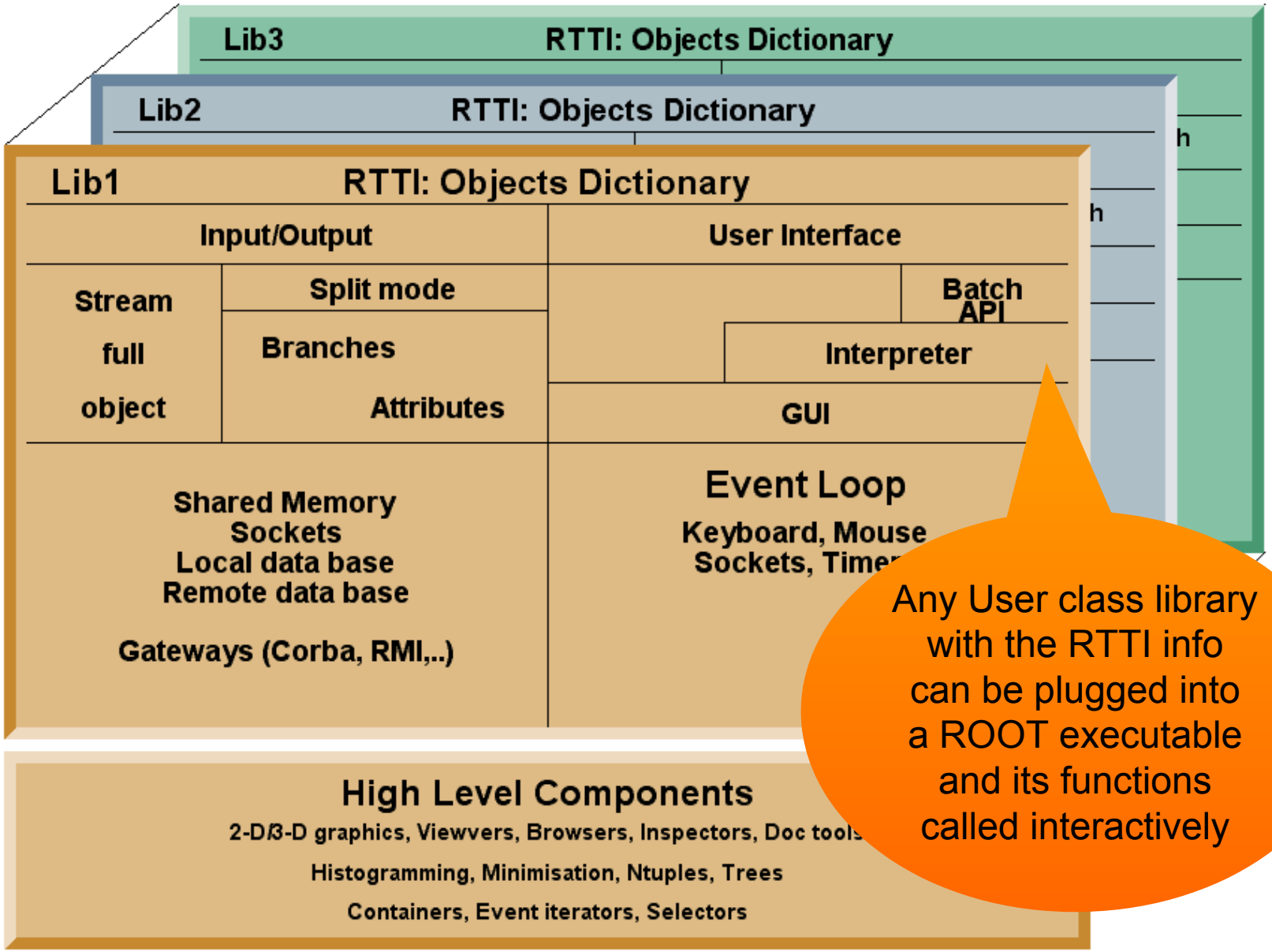
We were lucky to find an existing C++ interpreter CINT capable of parsing the complex C++ header files and to support a very large subset of the language interactively. CINT was developed by Masa Goto from HP/Japan since 1992.

We developed an extended Run Time Type Information (RTTI) used in the I/O system but also in many other places including the Graphical user Interface. This RTTI goes far beyond the C++ RTTI and looks more like the Introspection mechanism in Java.

Rootcint Preprocessor

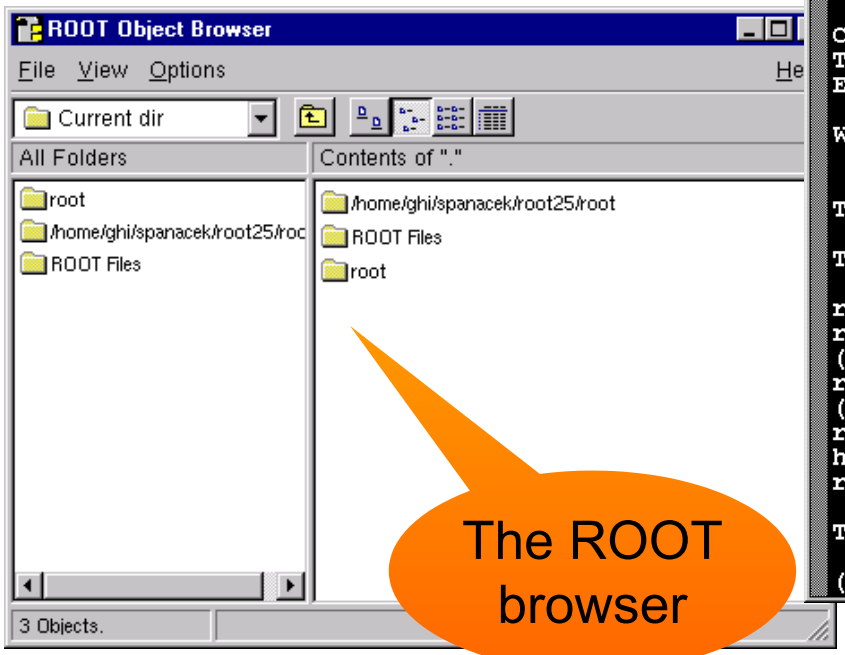


Framework: Basic components

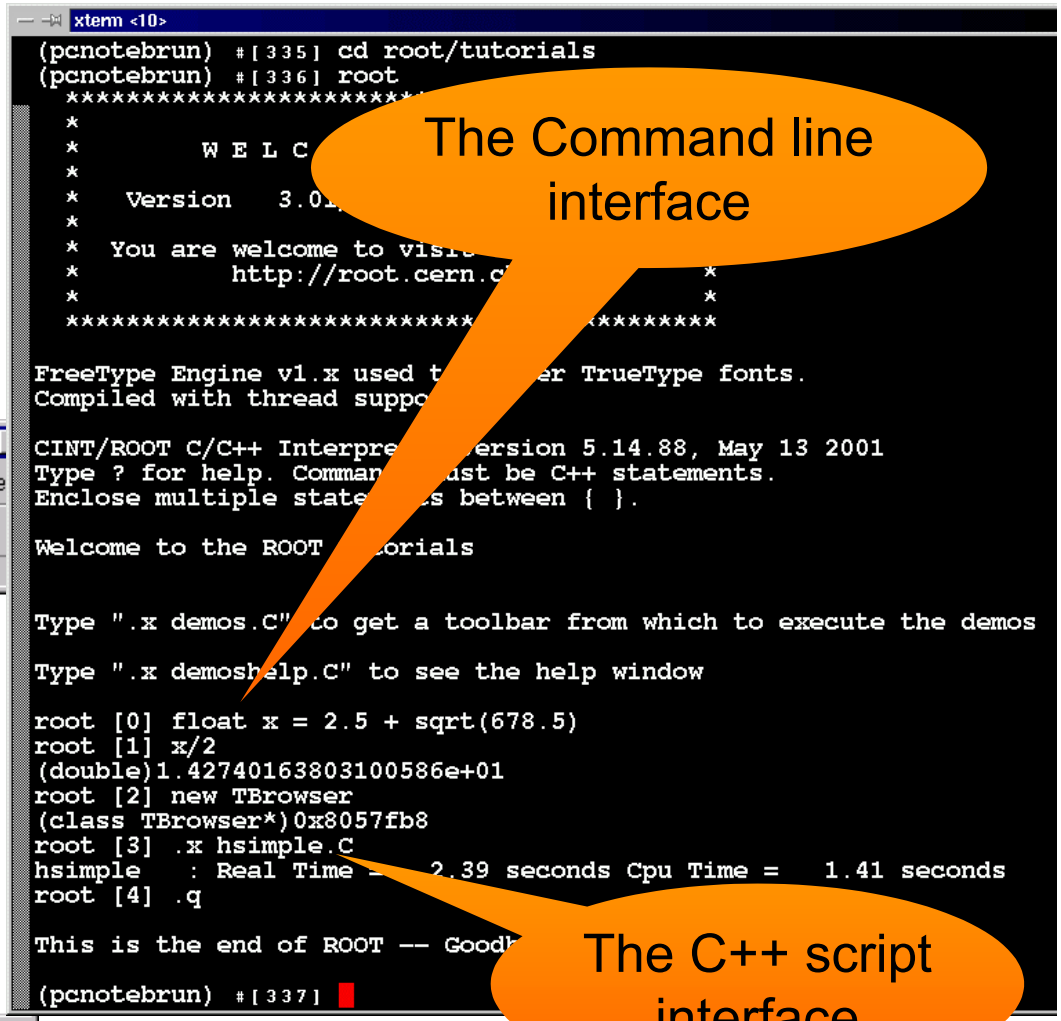


ROOT User Interfaces

Command line
C++ scripts
GUI



The ROOT browser

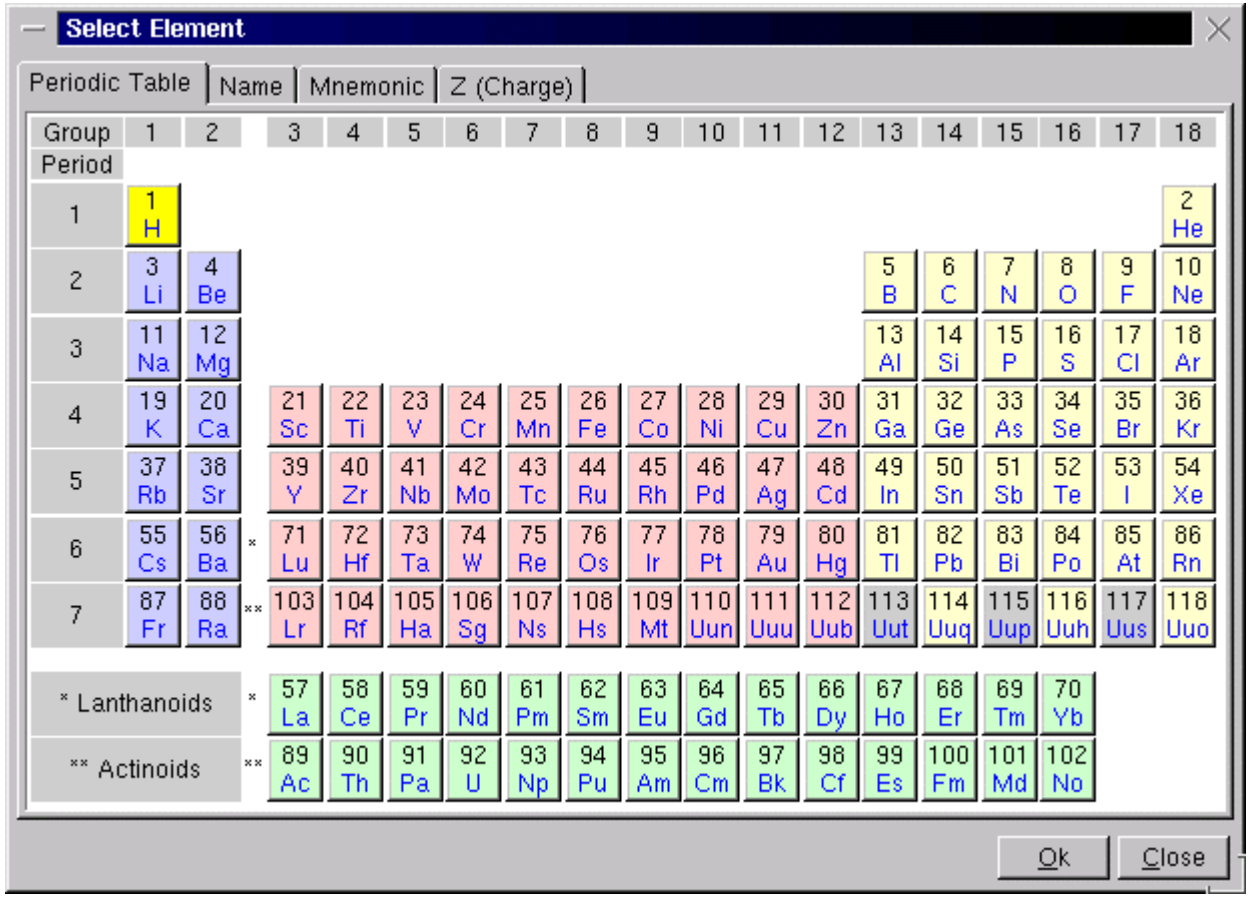


The Command line interface

The C++ script interface

GUI User example

Example of GUI based on ROOT tools
Each element is clickable



The Graphics Event Loop

- The ROOT event handler supports:
  - keyboard interrupts
  - system signals
  - X11, Xt, Xm events
  - Sockets interrupts
  - Special messages (shared memory, threads..)
- Foreign systems (eg Inventor, X3d..) can easily be integrated in the ROOT loop.
  - ROOT will dispatch the foreign events using Timers.
- Signals and Slots like in Qt
  - Qt and ROOT can work together

2-D Graphics

- Basic primitives (lines, text, markers, box, polygones, fills)
- Graphs, annotators (pave, pavetext, pavelabel)
- LateX support (screen and PostScript)
- Graphics Editor
- Pad Graphics via abstract class TVirtualPad
- Basic graphics via abstract class TVirtualX
  - (TGX11, TGWin32)
- TObject::Draw/Paint
- TObject::DistancetoPrimitive/ExecuteEvent

Full LateX support on screen and postscript

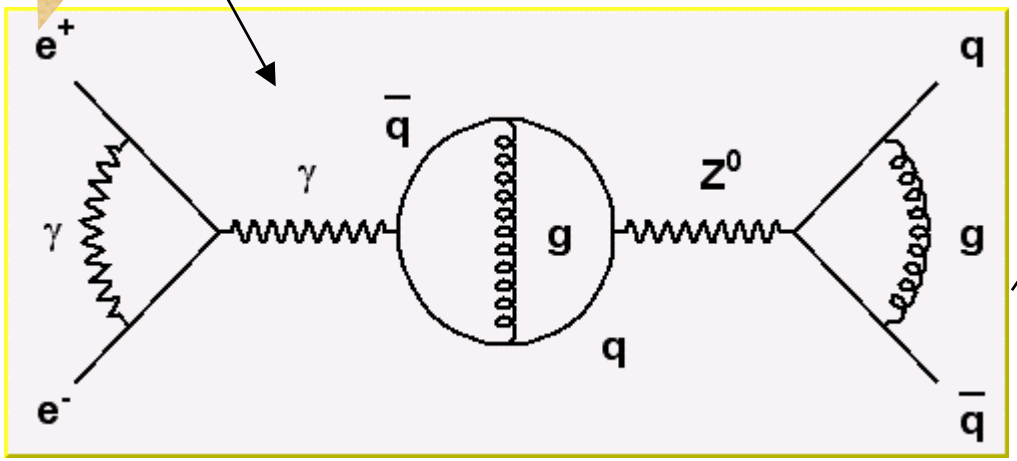
Born equation

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

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

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

Formula or diagrams can be edited with the mouse

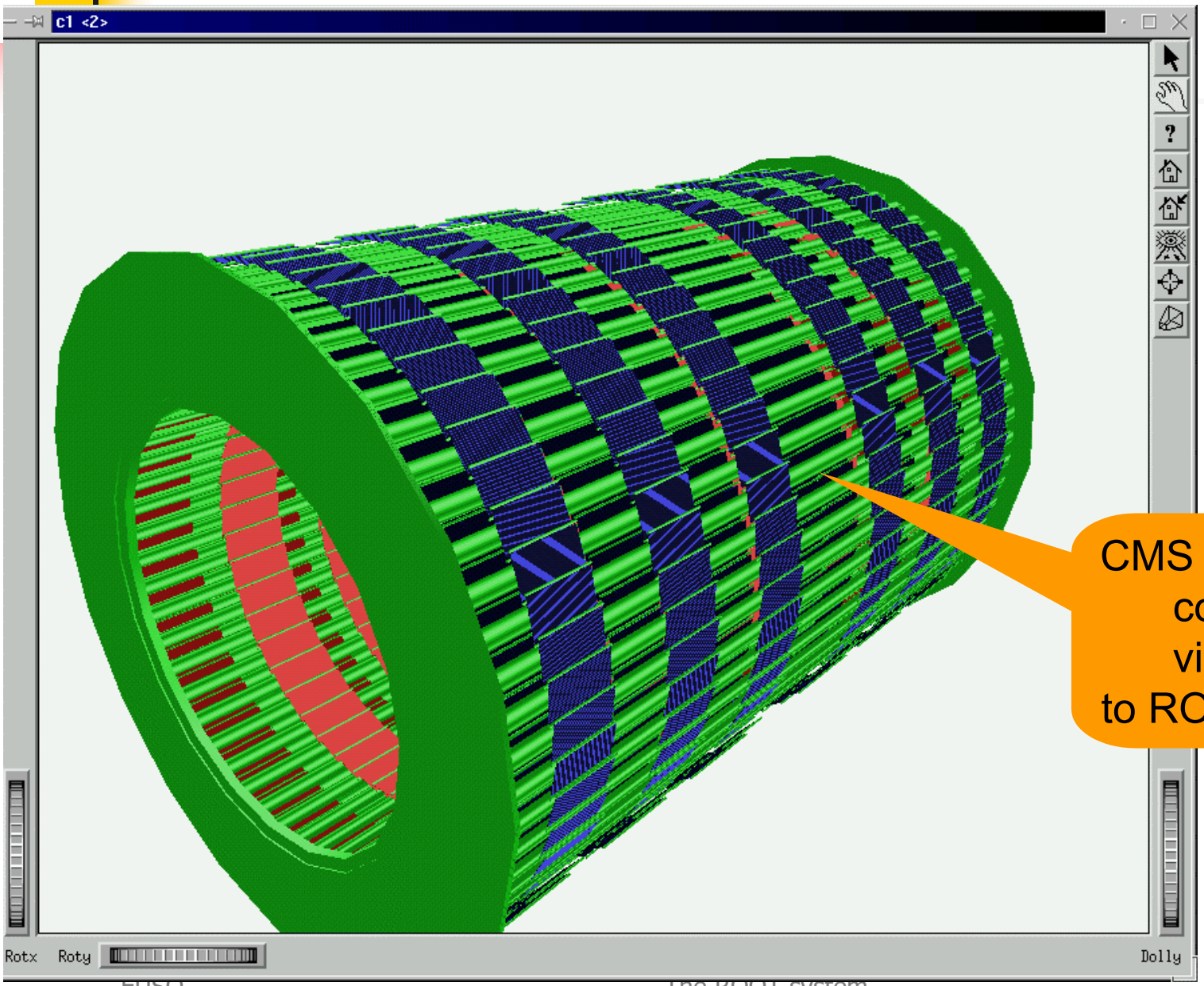


TCurlyArc
TCurlyLine
TWavyLine
and other building blocks for Feynmann diagrams

3-D Graphics

- Basic primitives
  - TPolyLine3D, TPolyMarker3D, THelix, TMarker3DBox,TAxis3D
- Geant primitives
  - Support for all Geant3 volumes + a few new volume types
  - TBRIK,TCONE,TCONS,TCTUB,TELTU,TGTRA,THYPE,TPARA,TPCON, TPGON,TSPHE,TTUBE,TTUBS,TTRAP,TTRD1,TTRD2,TXTRU
- Rendering with:
  - TPad
  - X3D (very fast. Unix only. Good on networks)
  - OpenGL
  - OpenInventor (new addition in 3.01)

ROOT + OpenInventor

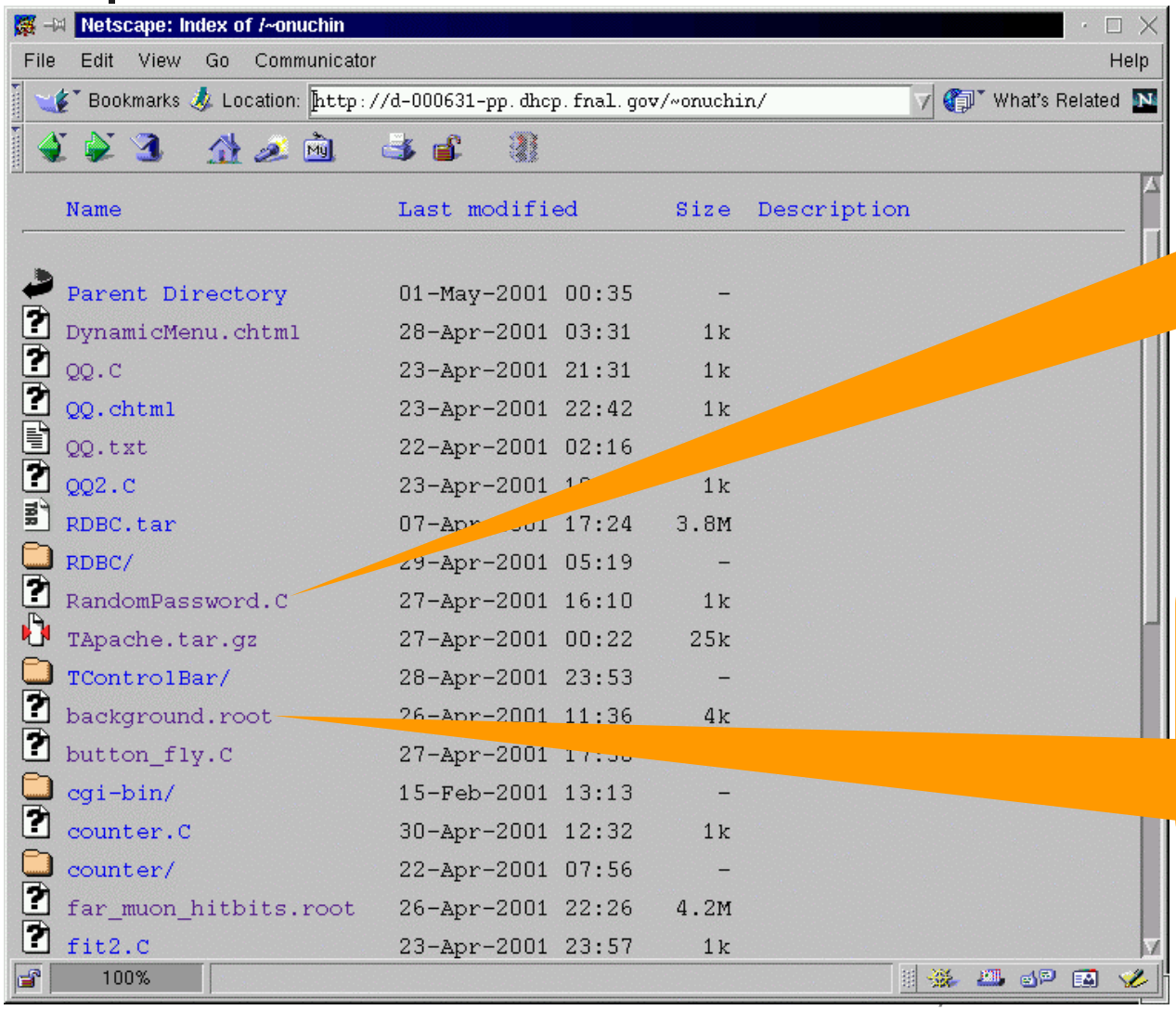


CMS with Geant3
converted
via g2root
to ROOT TNodes

ROOT and the WEB

- An Apache Web-server plug-in module is being developed (presented at FNAL workshop).
- Provides interactive access to ROOT files, CINT macros and all the graphics. Web pages generated on the fly.
- Interesting alternative to PHP using C++ as an embedded scripting system with full access to user classes dynamically

Apache plug-in TApache



Click here to execute CINT script

Click here to browse this ROOT file

Apache plug-in TApache

TCanvas interface

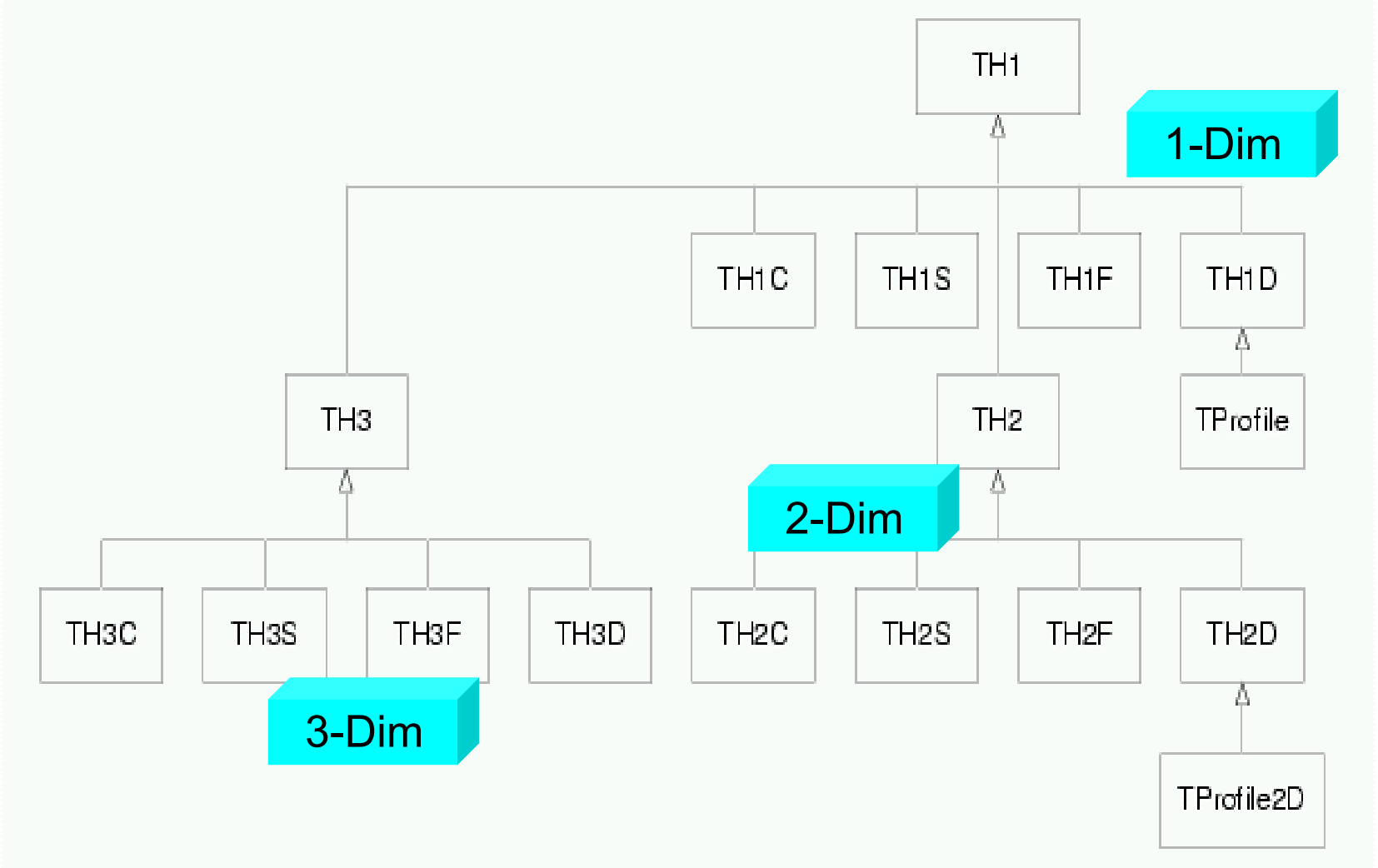
URLs
of ROOT files

Netscape: Source of: http://d-000631-p.dhcp.fnal.gov/~onuchin/far_muon_hitbits.root

```
<HTML>
 <HEAD>
  <TITLE>Index of /~onuchin/far_muon_hitbits.root</TITLE>
 </HEAD>
 <BODY>

<script language="JavaScript">

<!--
function TCanvas() {
canvas= open("", "TCanvas","width=567,height=384,status=no,toolbar=no,menubar=no,resizable=no,screenX=20,screenY=20,alwa
canvas.document.open();
canvas.document.write("<html><head><title>TCanvas");
canvas.document.write("</title></head><body bgcolor=white>");
canvas.document.write("</body></html>");}
<!-- -->
TCanvas();

</script>

<H1>Index of /~onuchin/far_muon_hitbits.root</H1>
<PRE><IMG SRC="/icons/blank.gif" ALT="      "> <A HREF="?QQ ">Name</A>                    <A HREF="?QQ ">ClassName</A>
<HR>
<IMG SRC="/icons/back.gif" ALT="[DIR]"> <A HREF="/~onuchin">Parent Directory</A>
<IMG SRC="/icons/folder.gif" ALT="[REROOT Geometry Tree]"> <A HREF="/~onuchin/far_muon_hitbits.root?/GEOM/">GEOM/</A>
<IMG SRC="/icons/folder.gif" ALT="[REROOT Event Tree]"> <A HREF="/~onuchin/far_muon_hitbits.root?/GEVT/">GEVT/</A>

<HR>
</PRE>
</BODY></HTML>
```

The Histogram Package

The Histogram Classes



Random Numbers and Histograms

- TH1::FillRandom can be used to randomly fill an histogram using
  - the contents of an existing TF1 analytic function
  - another histogram (for all dimensions).

- For example the following two statements create and fill an histogram 10000 times with a default gaussian distribution of mean 0 and sigma 1:
  - ***TH1F h1("h1","histo from a gaussian",100,-3,3);***
  - ***h1.FillRandom("gaus",10000);***

- TH1::GetRandom can be used to return a random number distributed according the contents of an histogram.

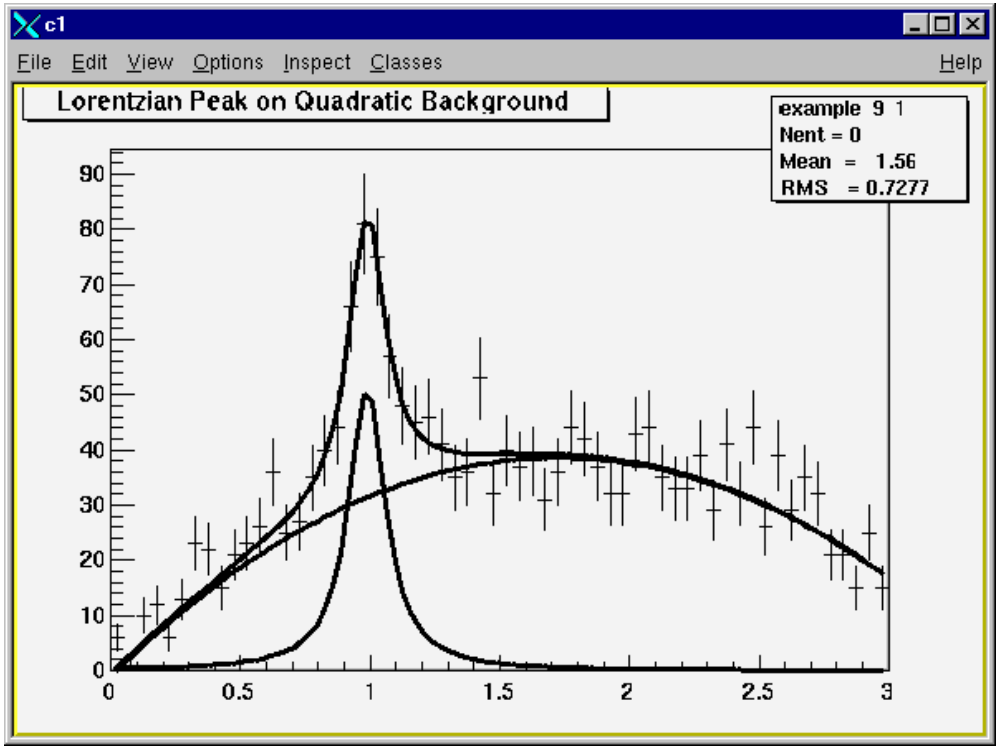
Fitting Histograms with Minuit

- Histograms (1-D,2-D,3-D and Profiles) can be fitted with a user specified function via TH1::Fit. Two Fitting algorithms are supported: Chisquare method and Log Likelihood

- The user functions may be of the following types:
  - standard functions: gaus, landau, expo, poln
  - combination of standard functions; poln + gaus
  - A C++ interpreted function or a C++ precompiled function

- An option is provided to compute the integral of the function bin by bin instead of simply compute the function value at the center of the bin.
- When an histogram is fitted, the resulting function with its parameters is added to the list of functions of this histogram. If the histogram is made persistent, the list of associated functions is also persistent.
- One can retrieve the function/fit parameters with calls such as:
  - ***Double_t chi2 = myfunc->GetChisquare();***
  - ***Double_t par0 = myfunc->GetParameter(0); //value of 1st parameter***
  - ***Double_t err0 = myfunc->GetParError(0); //error on first parameter***

Fitting Demo



Look at

- FittingDemo.C
  - Unnamed Macro
- fitf.C
  - Named Macro

Run FittingDemo.C

More info on fitting:

http://root.cern.ch/root/html/examples/fit1.C.html

http://root.cern.ch/root/html/examples/myfit.C.html

http://root.cern.ch/root/html/examples/backsig.C.html

Combining Functions

$$y(E) = a_1 + a_2 E + a_3 E^2 \quad + \quad A_P \, (\Gamma / 2\,\pi)/((E-\mu)^2 + (\Gamma/2)^2)$$

background

lorenzianPeak

par[0] = a_1
par[1] = a_2
par[2] = a_3

par[0] = A_P
par[1] = Γ
par[2] = μ

fitFunction = background (x, par) + lorenzianPeak (x, &par[3])

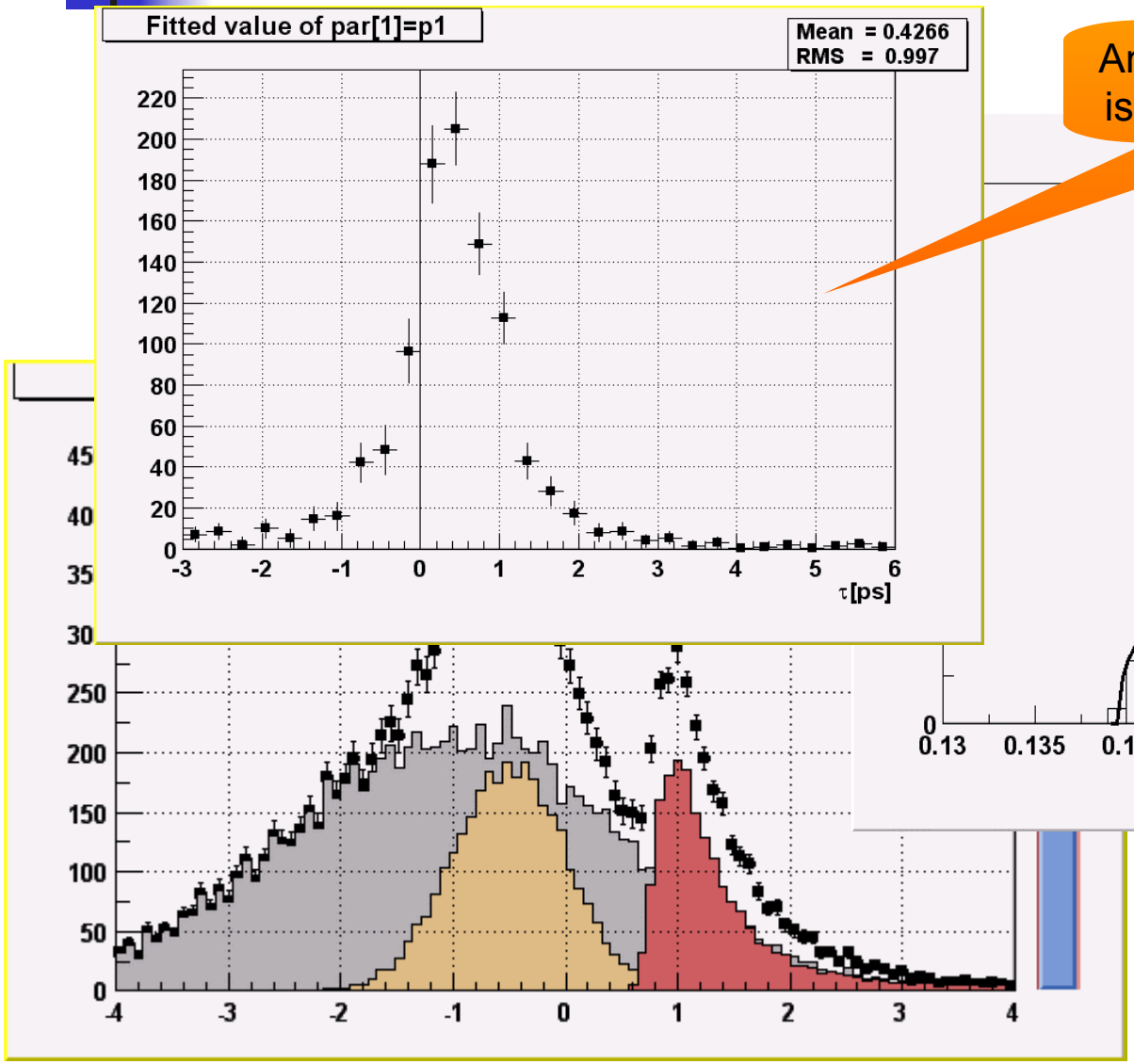
par[0] = a_1
par[1] = a_2
par[2] = a_3
par[3] = A_p
par[4] = Γ
par[5] = μ

Functions with
many parameters (> 200)
can be minimized.
No limitations like in Minuit

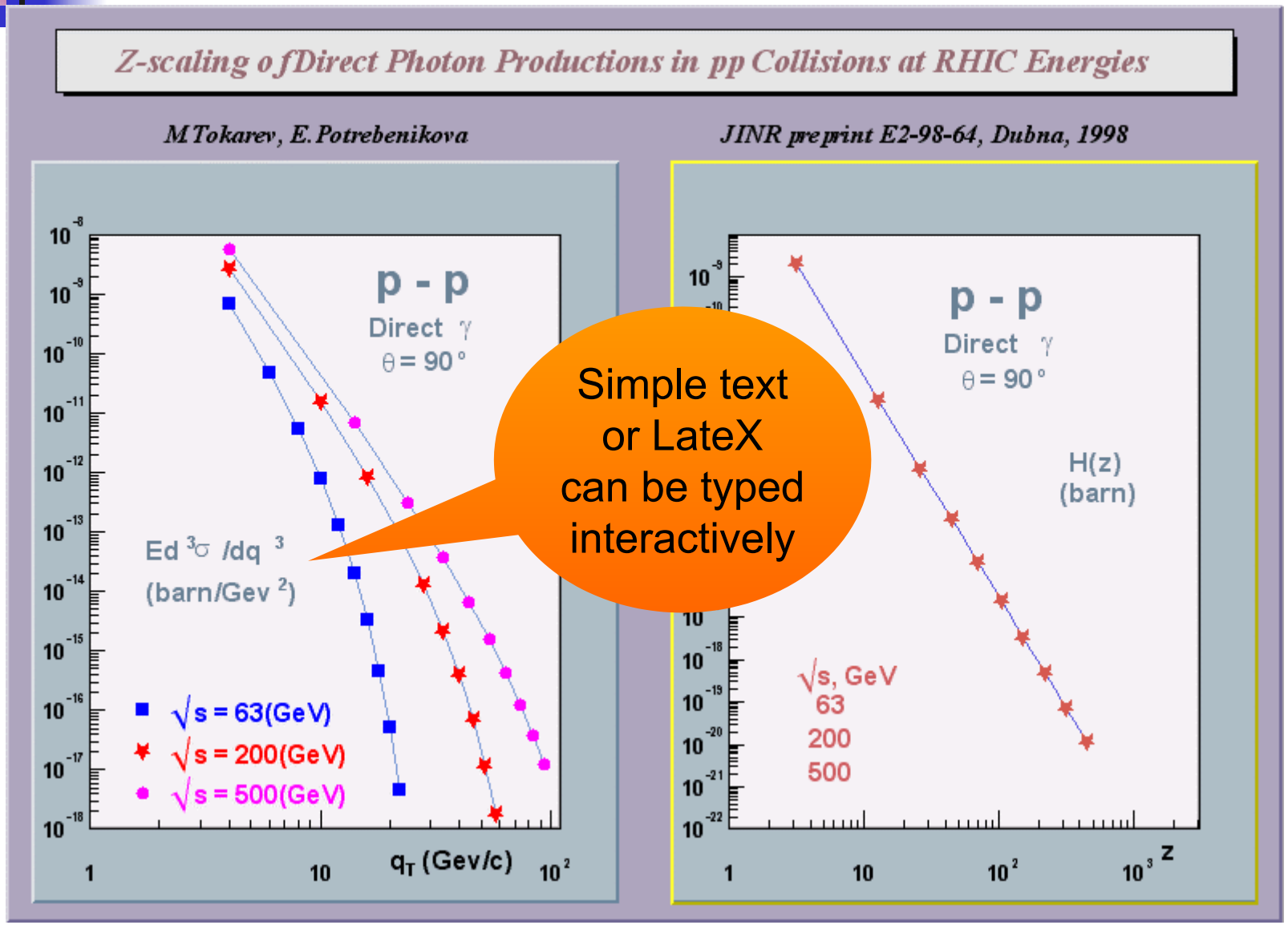
Drawing Histograms

- When you call the Draw method of a histogram for the first time (TH1::Draw), it creates a THistPainter object and saves a pointer to painter as a data member of the histogram.
- The THistPainter class specializes in the drawing of histograms. It is separate from the histogram so that one can have histograms without the graphics overhead, for example in a batch program. The choice to give each histogram have its own painter rather than a central singleton painter, allows two histograms to be drawn in two threads without overwriting the painter's values.
- When a displayed histogram is filled again you do not have to call the Draw method again. The image is refreshed the next time the pad is updated.
- The same histogram can be drawn with different graphics options in different pads.
- When a displayed histogram is deleted, its image is automatically removed from the pad.

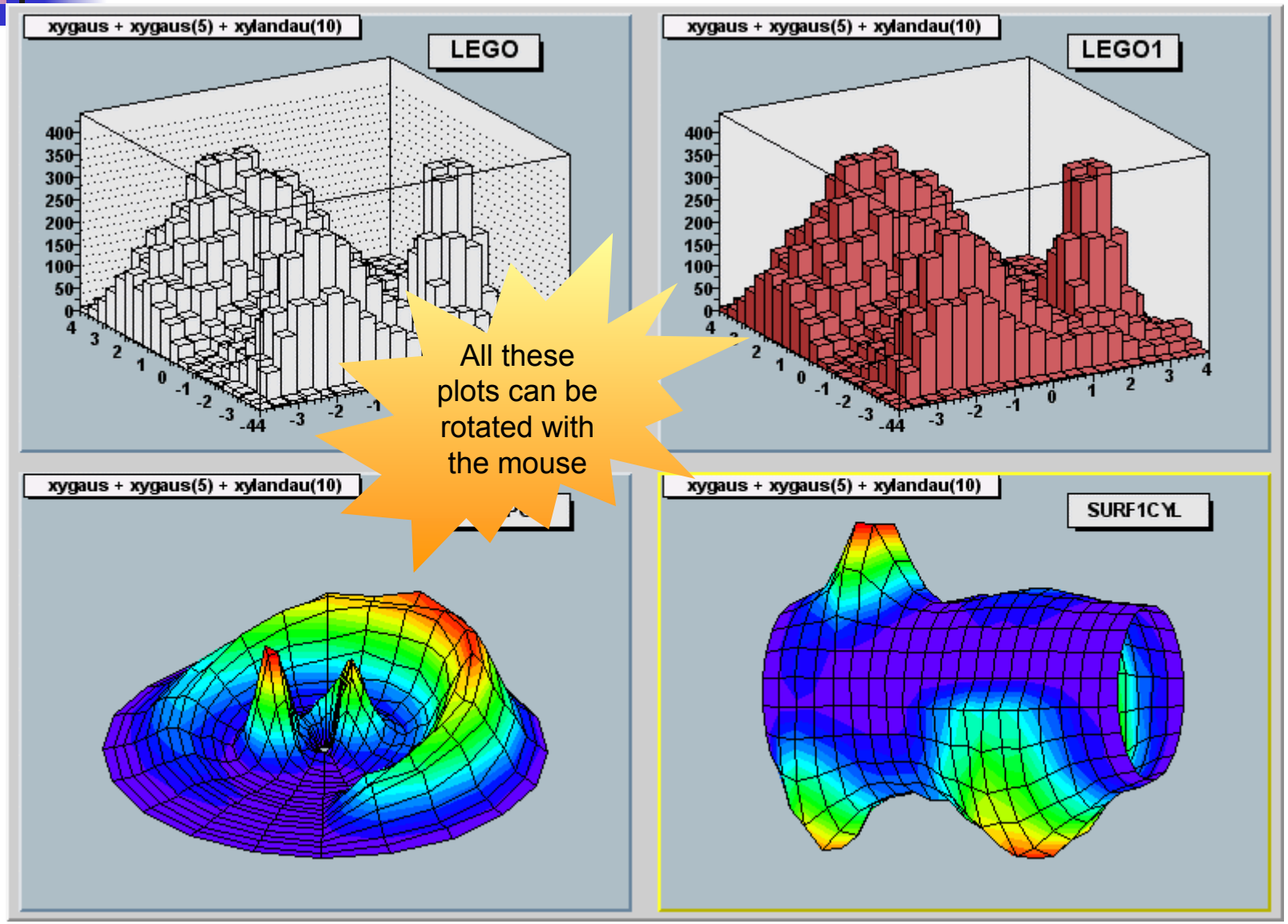
1-D drawing Options



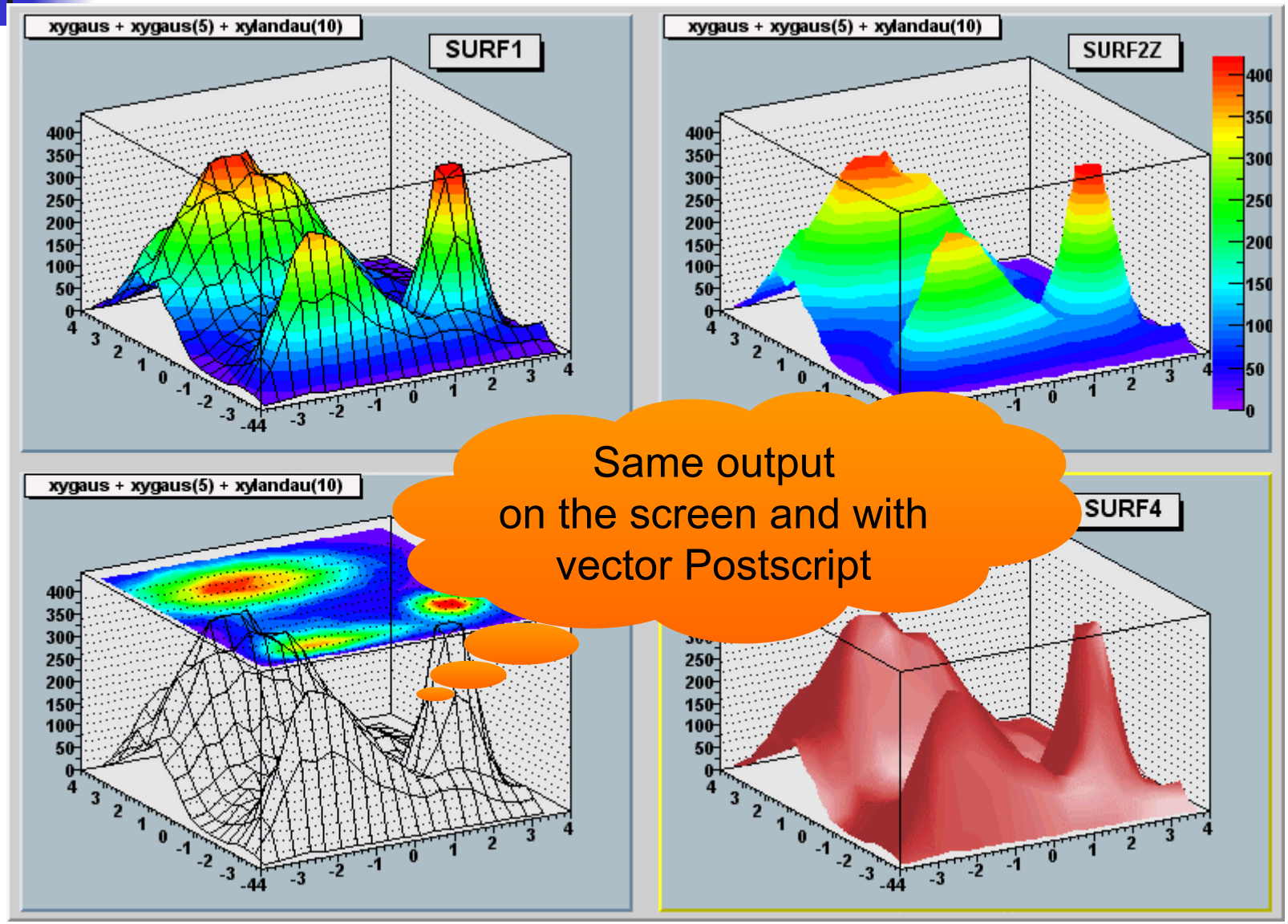
1-D drawing Options



2-D drawing Options



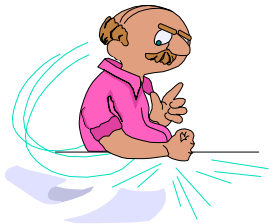
2-D drawing Options



ROOT Data Base Approach

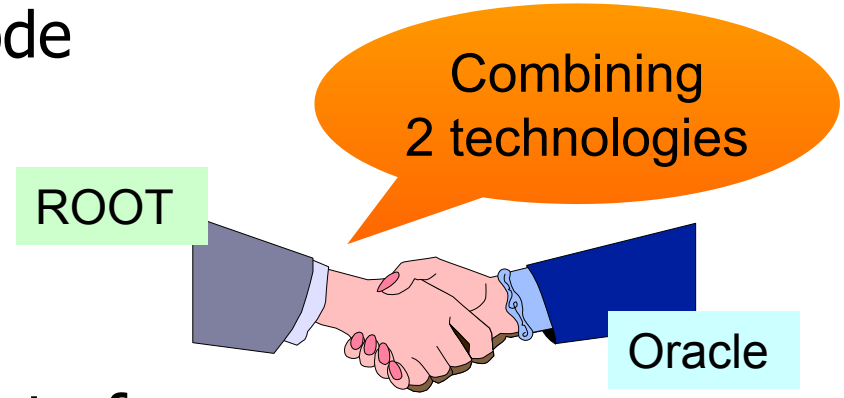
Data Bases: model-1

- Put everything in an Object Data base
  - like Objectivity
  - or Oracle 9i
- Choice of RD45 project
- Many experiments initially following this line
- Abandonned by most experiments recently
- Solution not suited for interactive analysis

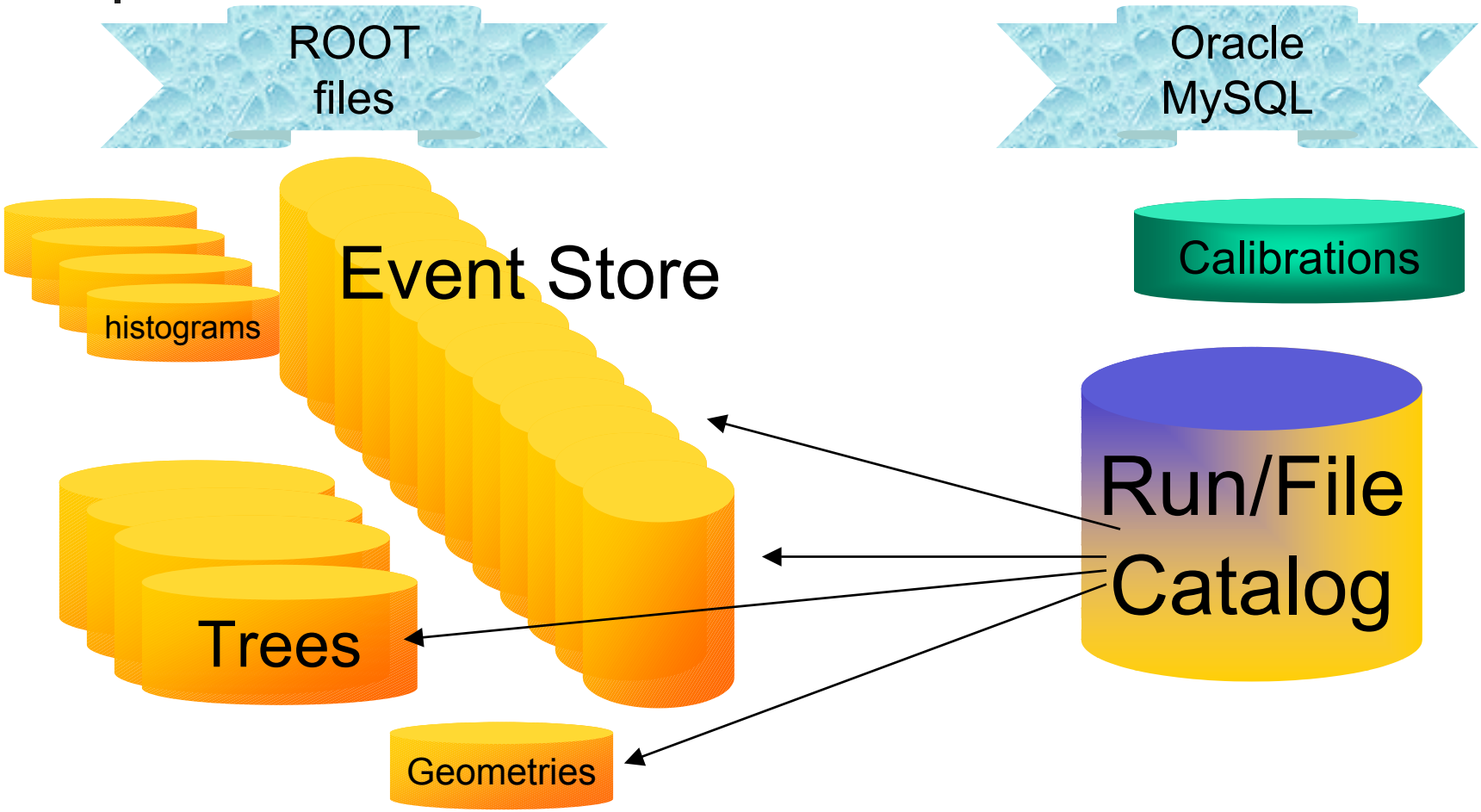


Data Bases: model-2

- Put write-once data in an object store
  - like ROOT in Streamer mode
- Use a RDBMS for :
  - Run/Event catalogs
  - Geometry, calibrations
  - eg with ROOT<->Oracle interface
    - http://www.phenix.bnl.gov/WWW/publish/onuchin/rooObjy/
  - or with ROOT <-> Objectivity interface
    - http://www.phenix.bnl.gov/WWW/publish/onuchin/RDBC/
- Use ROOT split/no-split mode for data analysis



ROOT + RDBMS Model



ROOT working with Objectivity

ROOT
Objectivity

- Introduction
- Naming Rules
- Examples
- FAQ
- Class Reference
- Availability
- TODO
- PHENIX DBrowser

Introduction

This is library of ROOT wrappers around Objectivity base classes and functions.

- **The Goals:**
  - Create a tool library which will allow the retrieval of data from Objy DB from a ROOT prompt.
  - Provide the basis library for ROOT GUI PHENIX Database Browser
  - The library was developed with hope it can be used inside PHENIX software framework
- **The Status:**
  - More than 40 Objectivity classes were wrapped with 10k lines of code and filled with Objy docs. Including:
    - Objectivity object handler and iterator classes, i.e. ooHandle(XXX), ooItr(XXX)
    - Objectivity Active Schema classes
    - Objectivity global functions, constants etc.
  - Compiled on Linux. Tested ++ more testing required. Not ported to S
- **The library allows users to:**
  - connect to Federated database
  - open up database
  - iterate through databases/containers
  - obtain descriptions of the classes in database
  - retrieve persistent data for any object in the database
  - ++ much more, interactively from ROOT

If you have any comments about this page please send them to Valeriy Onuchin

Objy and ROOT
can work together
An interactive interface
developed by Phenix

ROOT working with Oracle (2)

ROOT

ORACLE

SQL

PostgreSQL

- Introduction
- Demos & Tests
- Class Reference
- Download & Install

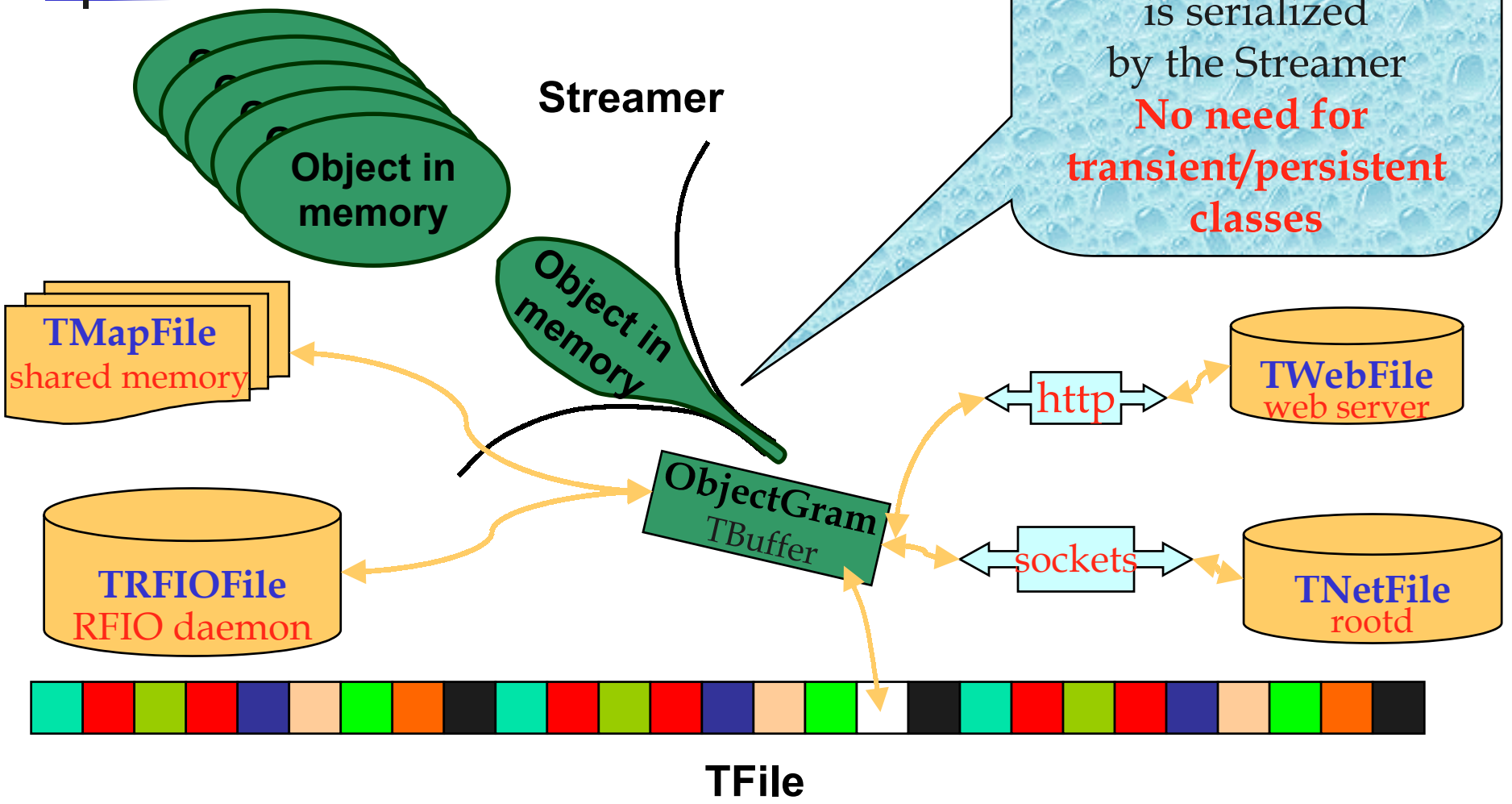
Introduction

- This library is a set of ROOT wrappers of libodbc++ classes that provides an interface based on the JDBC API.
- It corresponds to JDBC–ODBC bridge in Java terminology and makes it possible to do:
  1. establish a connection from ROOT sesssion to any database for which ODBC driver available.
  2. send SQL statements.
  3. process the results.
- Major differences between libodbc++ and RDBC:
  - RDBC written according to ROOT coding conventions. All method names begin with upper case letter.
  - Since ROOT C++ interpreter (CINT) does not provide full support of namespaces, all RDBC classes have TSQL prefix, e.g. TSQLStatement corresponds to odbc::Statement etc..
  - Since ROOT C++ interpreter (CINT) does not provide full support of exceptions, **Rt** object communication mechanism is used for *exception handling* (see TSQL::SetHandler method).
  - RDBC supprots
    - TSQLResultSet:GetObject
    - TSQLResultSet:UpdateObject
    - TSQLPreparedStatement:SetObject

    methods which allow to write/read ROOT objects to/from database (see "Save Your Histos in Oracle Database!" example).
- We consider RDBC as a low–level API and a base for higher–level interface between ROOT based offline enviroment of MINOS experiment and Oracle database. New ROOT–RDBC based GUI tools for Oracle are coming. Stay tuned!

ODBC compliant interface to Oracle

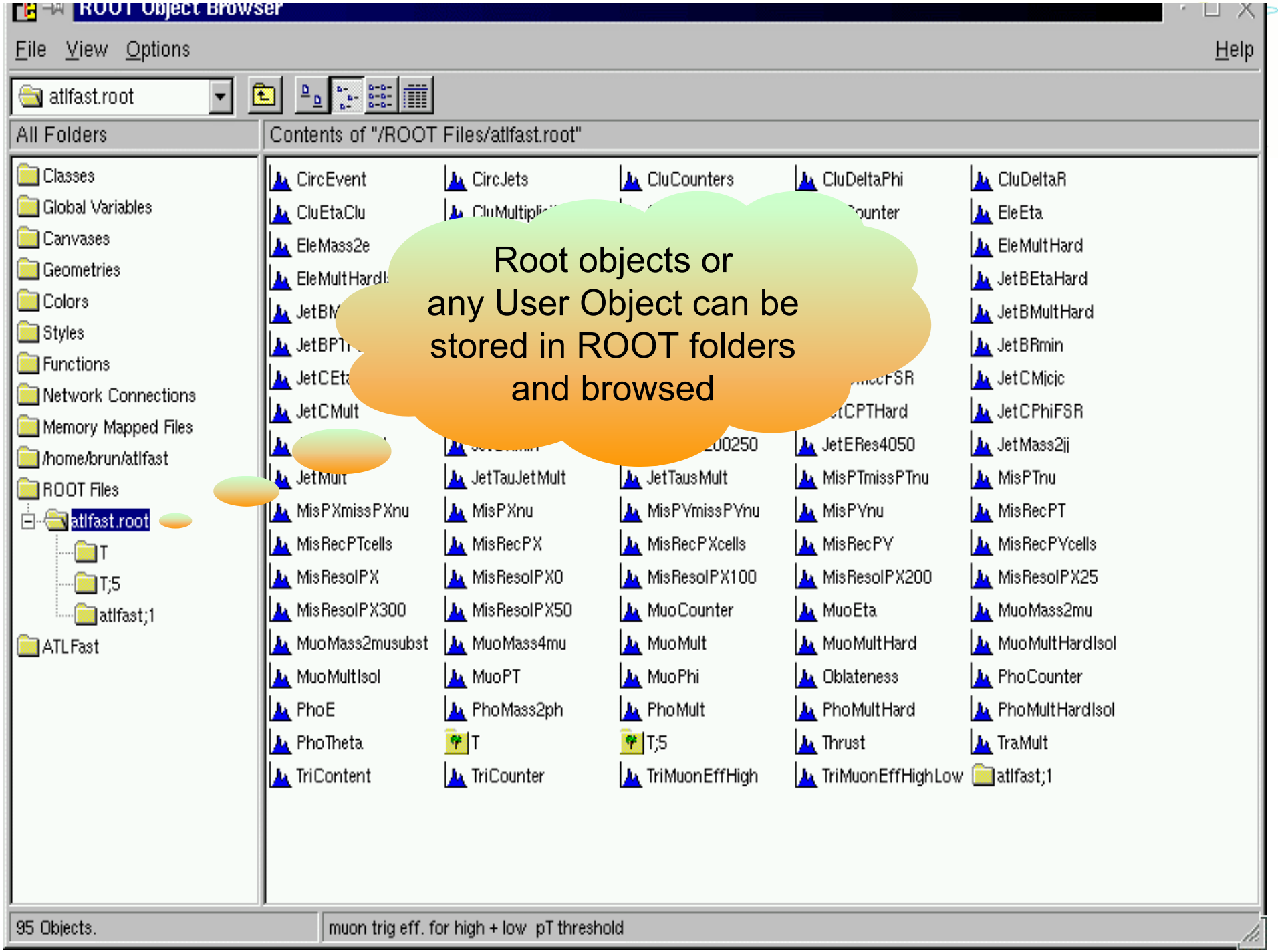
ROOT I/O -- *Sequential/Flat*

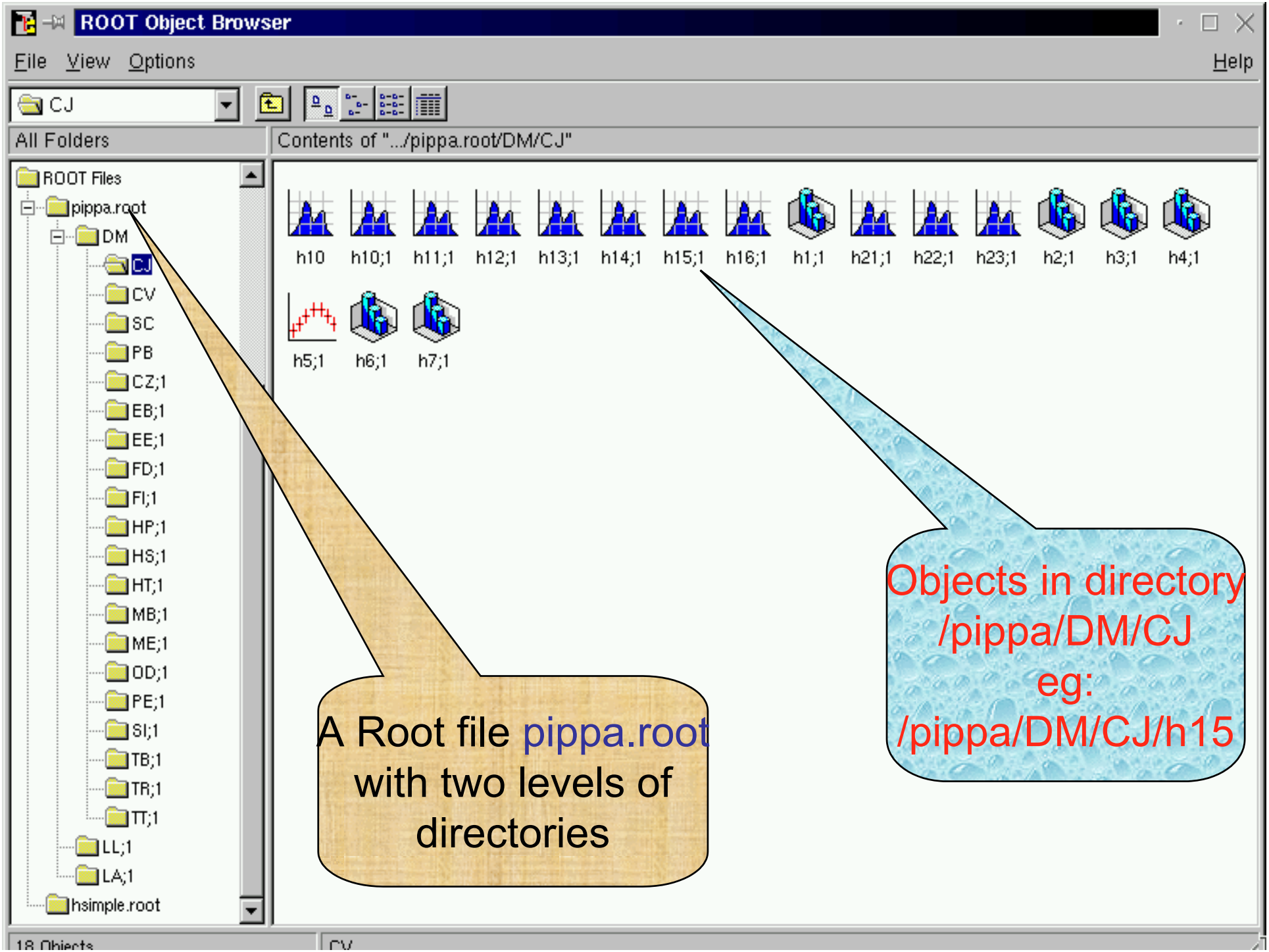


Streaming Objects

```
root [0]  TFile micro(”demo.root","new")
root [1]  TH1F hg("hg","filled with a gaussian",100,-4,4)
root [2]  hg.FillRandom("gaus",5000)
root [3]  hg.Write()
root [4]  micro.Map()
          20000511/092959  At:64          02     TFile
          20000511/093055  At:156   N             H1F      CX =  2.10
root [5] .q
```

The Write function calls TH1F::Streamer The Streamer function generated by rootcint fills a buffer with all the constituents of the object


ROOT Object Browser
File View Options Help
atlfast.root
All Folders
Contents of "/ROOT Files/atlfast.root"
Classes
Global Variables
Canvases
Geometries
Colors
Styles
Functions
Network Connections
Memory Mapped Files
/home/brun/atlfast
ROOT Files
atlfast.root
T
T;5
atlfast;1
ATLFast
CircEvent
CircJets
CluCounters
CluDeltaPhi
CluDeltaR
CluEtaClu
EleEta
EleMass2e
EleMultHard
JetBEtaHard
JetBMultHard
JetBRmin
JetCMicjc
JetCPhiFSR
JetCMult
JetERes4050
JetMass2jj
JetMult
JetTauJetMult
JetTausMult
MisPTmissPTnu
MisPTnu
MisPXmissPXnu
MisPXnu
MisPYmissPYnu
MisPYnu
MisRecPT
MisRecPTcells
MisRecPX
MisRecPXcells
MisRecPY
MisRecPYcells
MisResolPX
MisResolPX0
MisResolPX100
MisResolPX200
MisResolPX25
MisResolPX300
MisResolPX50
MuoCounter
MuoEta
MuoMass2mu
MuoMass2musubst
MuoMass4mu
MuoMult
MuoMultHard
MuoMultHardIsol
MuoMultIsol
MuoPT
MuoPhi
Oblateness
PhoCounter
PhoE
PhoMass2ph
PhoMult
PhoMultHard
PhoMultHardIsol
PhoTheta
T
T;5
Thrust
TraMult
TriContent
TriCounter
TriMuonEffHigh
TriMuonEffHighLow
atlfast;1
Root objects or
any User Object can be
stored in ROOT folders
and browsed
95 Objects.
muon trig eff. for high + low pT threshold


ROOT Object Browser
File View Options Help
CJ
All Folders
Contents of ".../pippa.root/DM/CJ"
ROOT Files
pippa.root
DM
CJ
CV
SC
PB
CZ;1
EB;1
EE;1
FD;1
FI;1
HP;1
HS;1
HT;1
MB;1
ME;1
OD;1
PE;1
SI;1
TB;1
TR;1
TT;1
LL;1
LA;1
hsimple.root
h10
h10;1
h11;1
h12;1
h13;1
h14;1
h15;1
h16;1
h1;1
h21;1
h22;1
h23;1
h2;1
h3;1
h4;1
h5;1
h6;1
h7;1
A Root file pippa.root with two levels of directories
Objects in directory /pippa/DM/CJ eg: /pippa/DM/CJ/h15
18 Objects
CV

LAN/WAN files

- Files and Directories
  - a directory holds a list of named objects
  - a file may have a hierarchy of directories (a la Unix)
  - ROOT files are machine independent
  - built-in compression
- Support for local, LAN and WAN files
  - TFile f1("myfile.root")
  - TFile f2("http://pcbrun.cern.ch/Renefile.root")
  - TFile f3("root://cdfsga.fnal.gov/bigfile.root")
  - TFile f4("rfio://alice/run678.root")

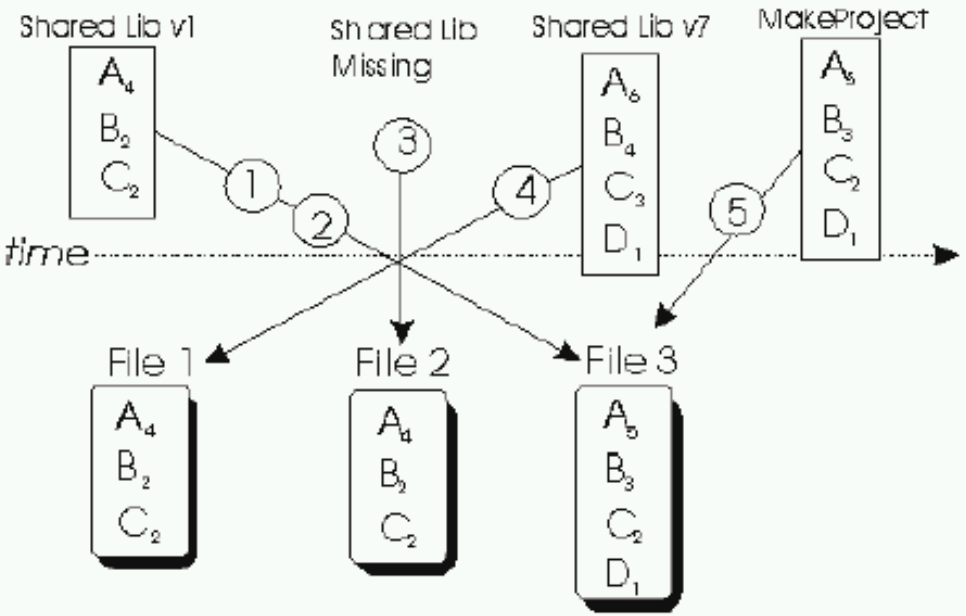
Local file

Remote file access via a Web server

Remote file access via the ROOT daemon

Access to a file on a mass store hpps, castor, via RFIO

Automatic Schema Evolution

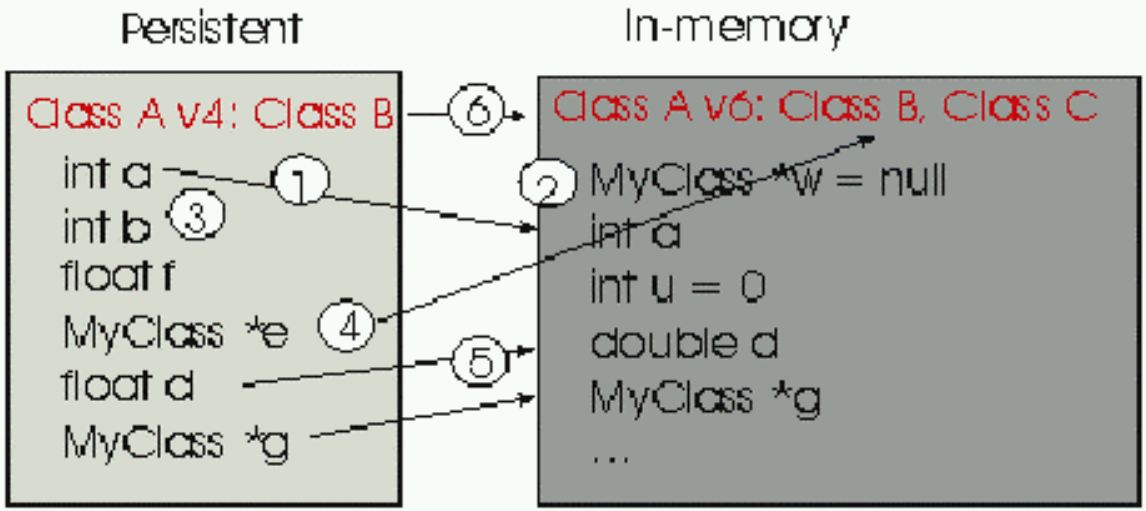


1) An old version of a shared library and a file with new class definitions. This can be the case when someone has not updated the library and is reading a new file.
2) Reading a file with a shared library that is missing a class definition (i.e. missing class D).
3) Reading a file without any class definitions. This can be the case where the class definition is lost, or unavailable.
4) The current version of a shared library and an old file with old class versions (backward compatibility). This is often the case when reading old data.
5) Reading a file with a shared library built with `MakeProject`. This is the case when someone has already read the data without a shared library and has used ROOT's `MakeProject` feature to reconstruct the class definitions and shared library (`MakeProject` is explained in detail later on).

Auto Schema Evolution (2)

In case of a mismatch between the in-memory version and the persistent version of a class, ROOT maps the persistent one to the one in memory. This allows you to change the class definition at will, for example:

1) Change the order of data members in the class.
2) Add new data members. By default the value of the missing member will be 0 or in case of an object it will be set to null.
3) Remove data members.
4) Move a data member to a base class or vice –versa.
5) Change the type of a member if it is a simple type or a pointer to a simple type. If a loss of precision occurs, a warning is given.
6) Add or remove a base class



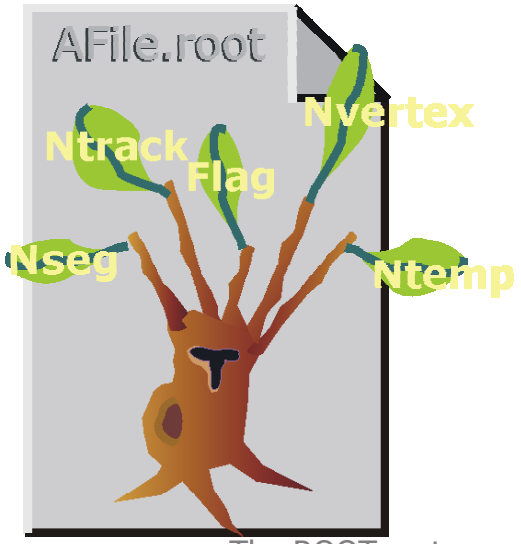
Self-describing files

- Dictionary for persistent classes written to the file.
- ROOT files can be read by foreign readers (JAS)
- 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++");
```

Why Trees ?

- Any object deriving from TObject can be written to a file with an associated key with object.Write()
- However each key has an overhead in the directory structure in memory (about 60 bytes). Object.Write is very convenient for objects like histograms, detector objects, calibrations, but not for event objects.



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. This can speed-up considerably the data analysis processes.

Tree Creation Example

```
class TEvent: public TObject
THeader        *fHeader;     //Event Header object
TObjArray      *fVertex;     //List of vertice
TClonesArray   *fTracks;     //List of tracks
TTOF           *fTOF;        //Time of Fl
TCalor         *fCalor;      //Calorime
```

A few lines of code to create a Tree for structures that may be very complex

```
main()
TEvent *event;
TFile dst("demo.root","NEW");
TTree tree("T","Example of Tree");
Int_t split = 1;  // or split=0
tree.Branch("event", "TEvent", &event, split);

for (int ev =0; ev<10000;ev++) {
   event = new TEvent(ev);
   tree.Fill();
   delete event;
}

dst.Close();
```

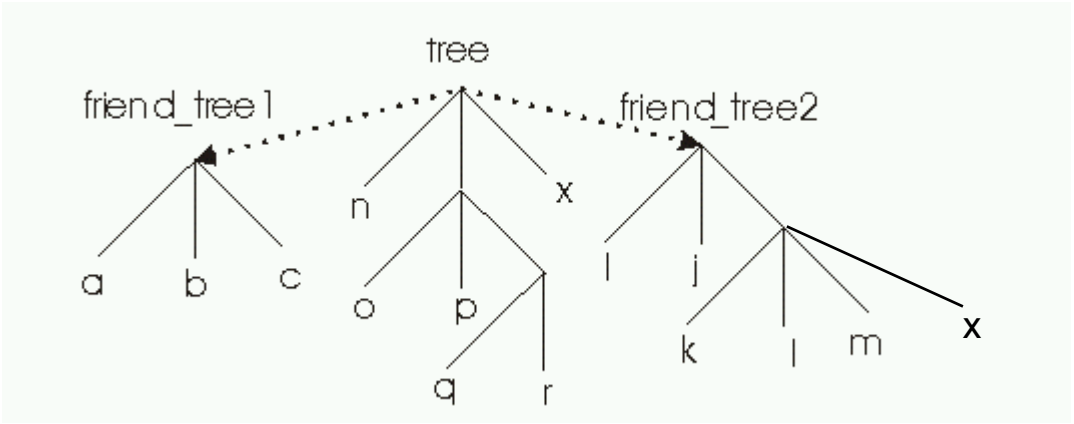
Chains of Trees

- 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");
}
```

Chain.Add("$H1/dstar*.root");

Tree Friends

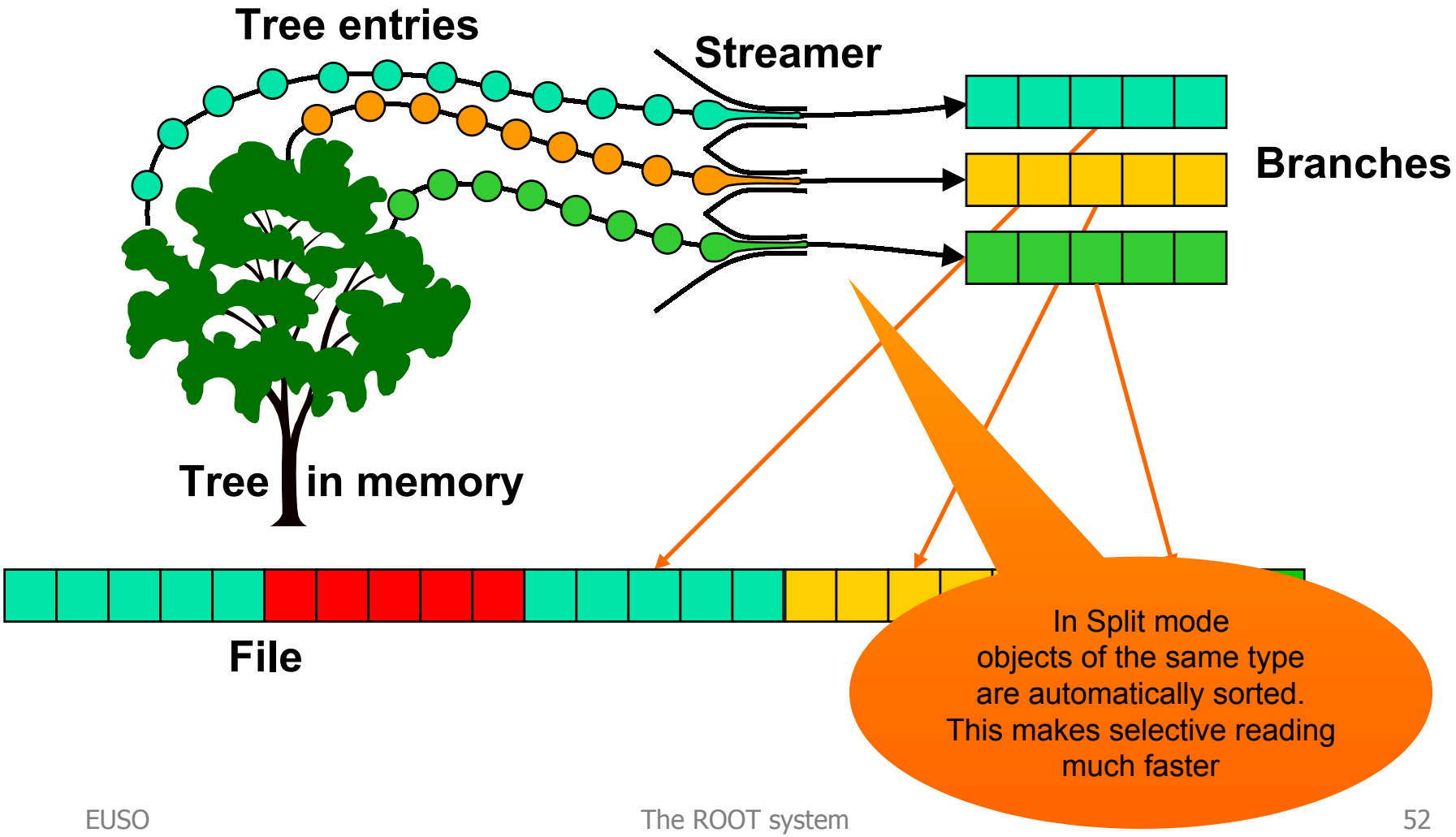


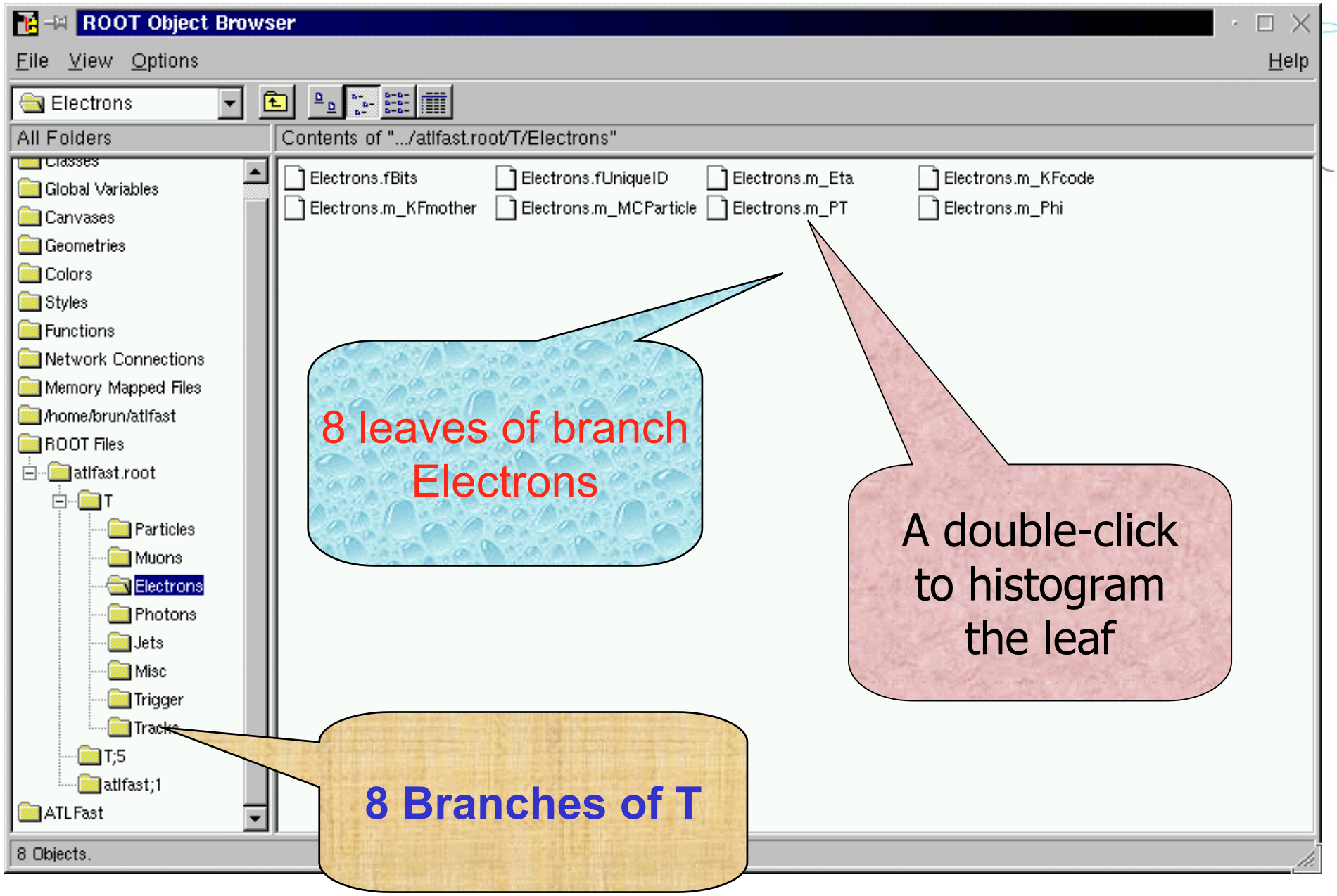
Processing time independent of the number of friends unlike table joins in RDBMS

```
Root > TFile f1("tree1.root");
Root > tree.AddFriend("tree2","tree2.root")
Root > tree.AddFriend("tree3","tree3.root");
Root > tree.Draw("x:a","k<c");
Root > tree.Draw("x:tree2.x","sqrt(p)<b");
```

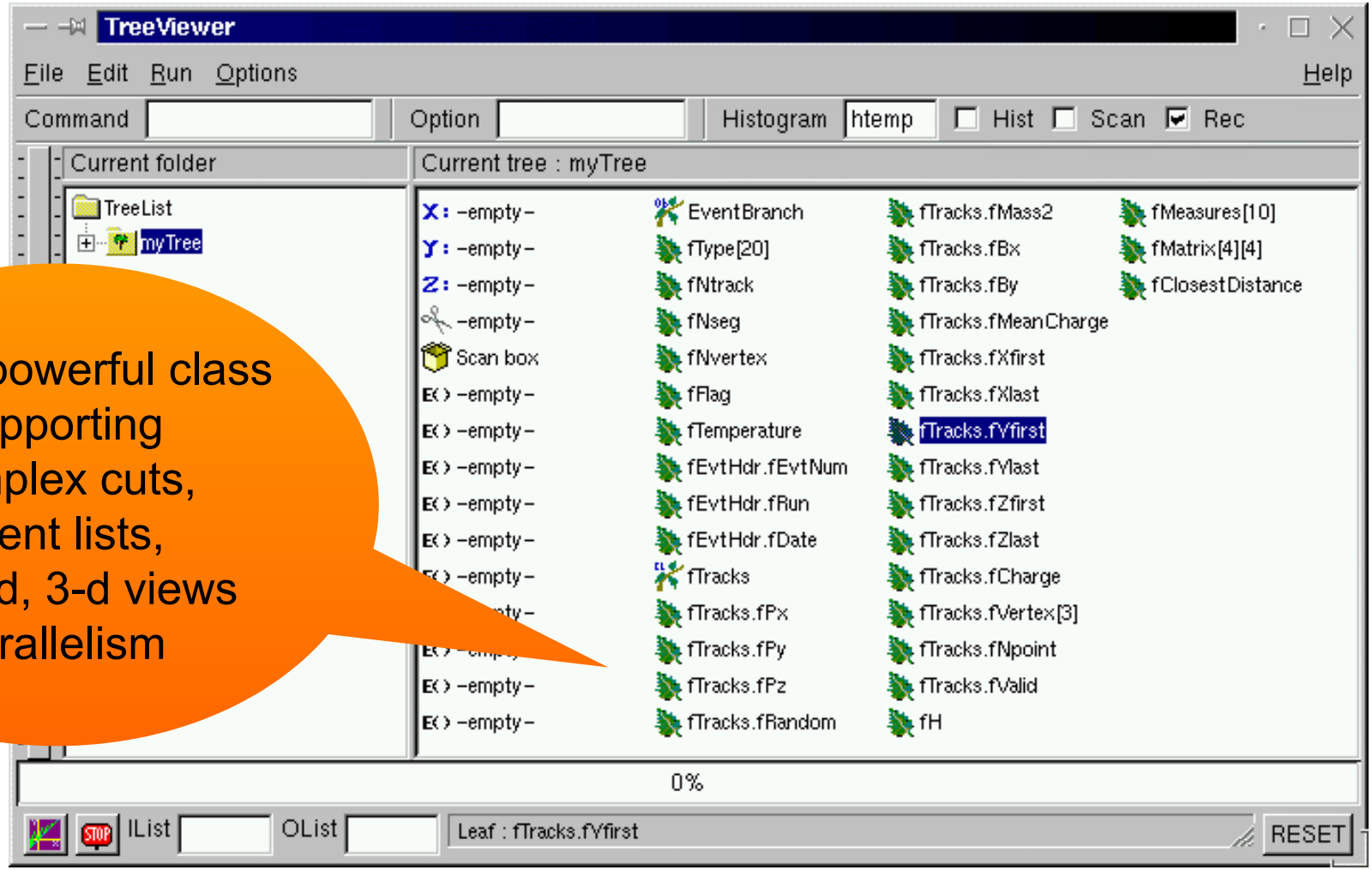
ROOT I/O -- *Split/Cluster*

Tree version




ROOT Object Browser
File View Options
Help
Electrons
All Folders
Contents of ".../atlfast.root/T/Electrons"
Global Variables
Canvases
Geometries
Colors
Styles
Functions
Network Connections
Memory Mapped Files
/home/brun/atlfast
ROOT Files
atlfast.root
T
Particles
Muons
Electrons
Photons
Jets
Misc
Trigger
Tracks
T;5
atlfast;1
ATLFast
Electrons.fBits
Electrons.fUniqueID
Electrons.m_Eta
Electrons.m_KFcode
Electrons.m_KFmother
Electrons.m_MCParticle
Electrons.m_PT
Electrons.m_Phi
8 leaves of branch Electrons
A double-click to histogram the leaf
8 Branches of T
8 Objects.

The Tree Viewer & Analyzer



A very powerful class supporting complex cuts, event lists, 1-d,2-d, 3-d views parallelism

Automatic Code Generators

- Data sets can be analyzed by the same classes used to store the data.
- However, one must be able to read the data without these original classes. The classes may not be available after some time.
- Root provides two utilities to gene te a class skeleton to read the data, still preserving the ribute names, types and the structure.
  - TTree::MakeClass
  - TTree::MakeSelector

This point is important.
You can always analyze
a data set even if you have lost
the class(es) that generated
this data set

TTree::MakeClass

- tree.MakeClass("myClass"); generates two files: myClass.h and myClass.C
- myClass.h contains the class declaration and member functions code that is selection invariant.
- myClass.C contains an example of empty loop where one can insert the analysis code
- Usage:
  - root > .L myClass.C or .L myClass.C++
  - root > myClass xx;
  - root > xx.Loop();

Use the interpreter

Use the native compiler
The file myClass.C
is automatically compiled
and linked !!

TTree::MakeSelector

- tree.MakeSelector("myClass"); generates two files: myClass.h and myClass.C that can work in a parallel system like PROOF. The event loop is not under user control.
- myClass.h contains the class declaration and member functions code that is selection invariant.
- myClass.C contains the skeleton of 4 functions: Begin, ProcessCut, ProcessFill, Terminate.
- Usage:
  - root > tree.Process("myClass.C");
  - root > chain.Process("myClass.C++");

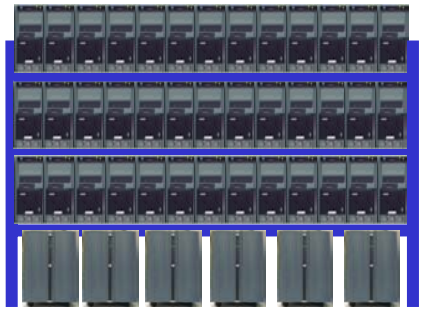
Macro is automatically compiled and linked

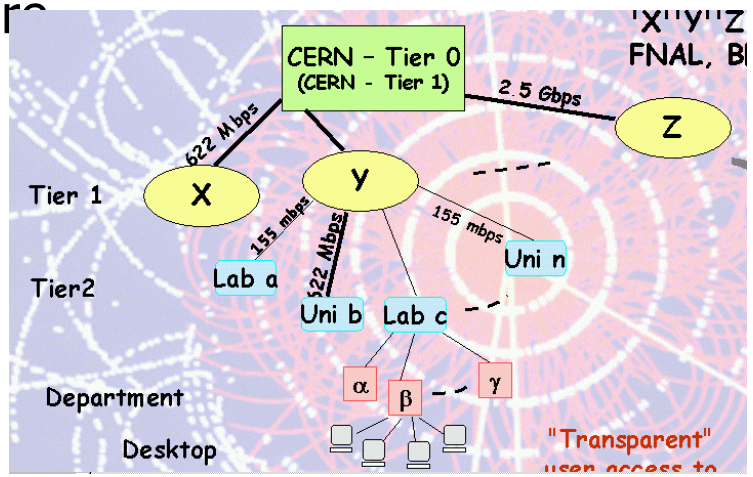
GRIDs and PROOF

PROOF and GRIDs

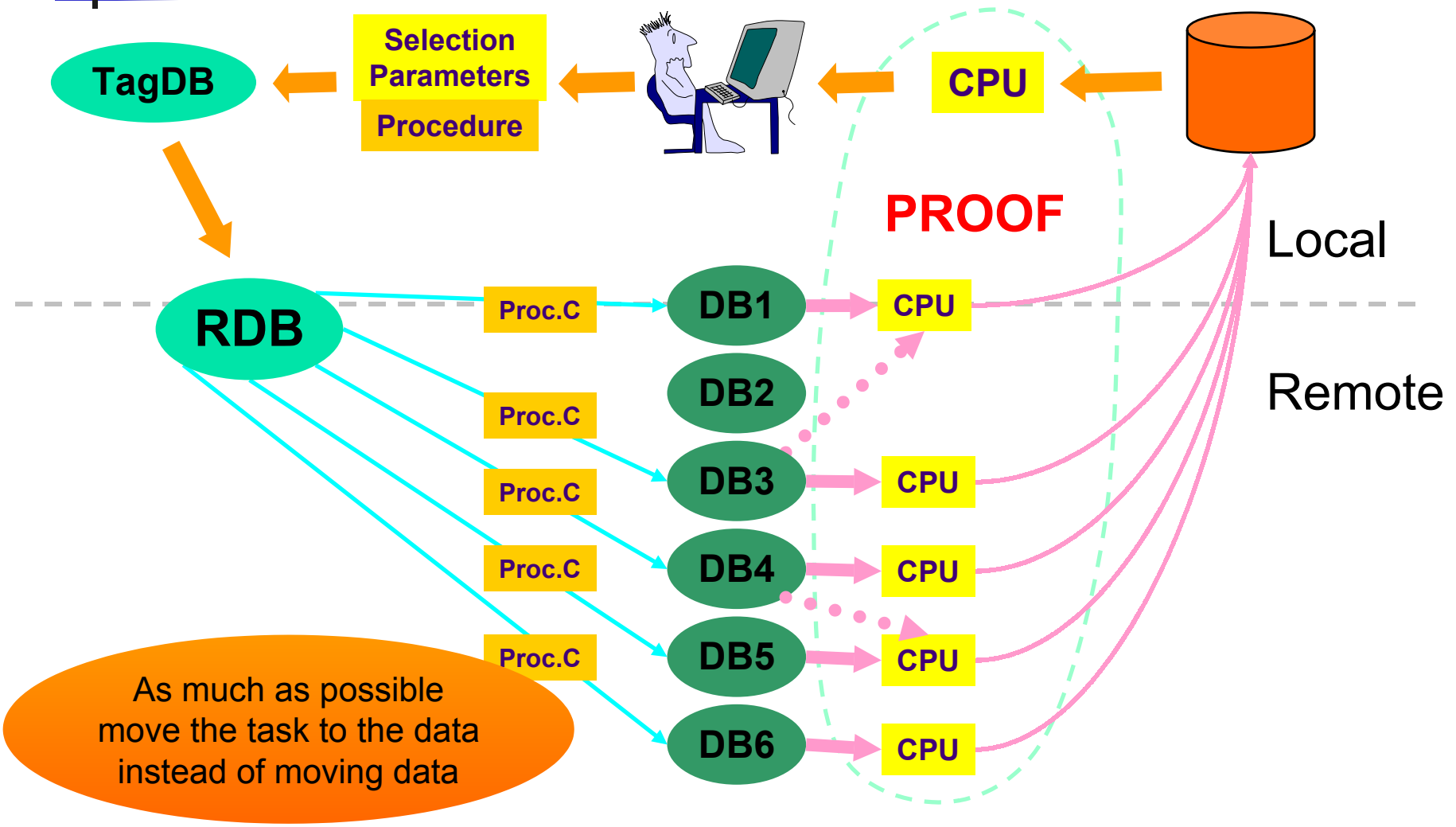
- The PROOF system allows parallel processing of chains of trees on clusters of heterogeneous machines. Its main features are:
  - Transparency, scalability, adaptivity
- A first prototype developed in 1997 as proof of concept (only for simple queries resulting in 1D histograms)

- We are now implementing the system taking into the most recent developments in the GRID middlewar

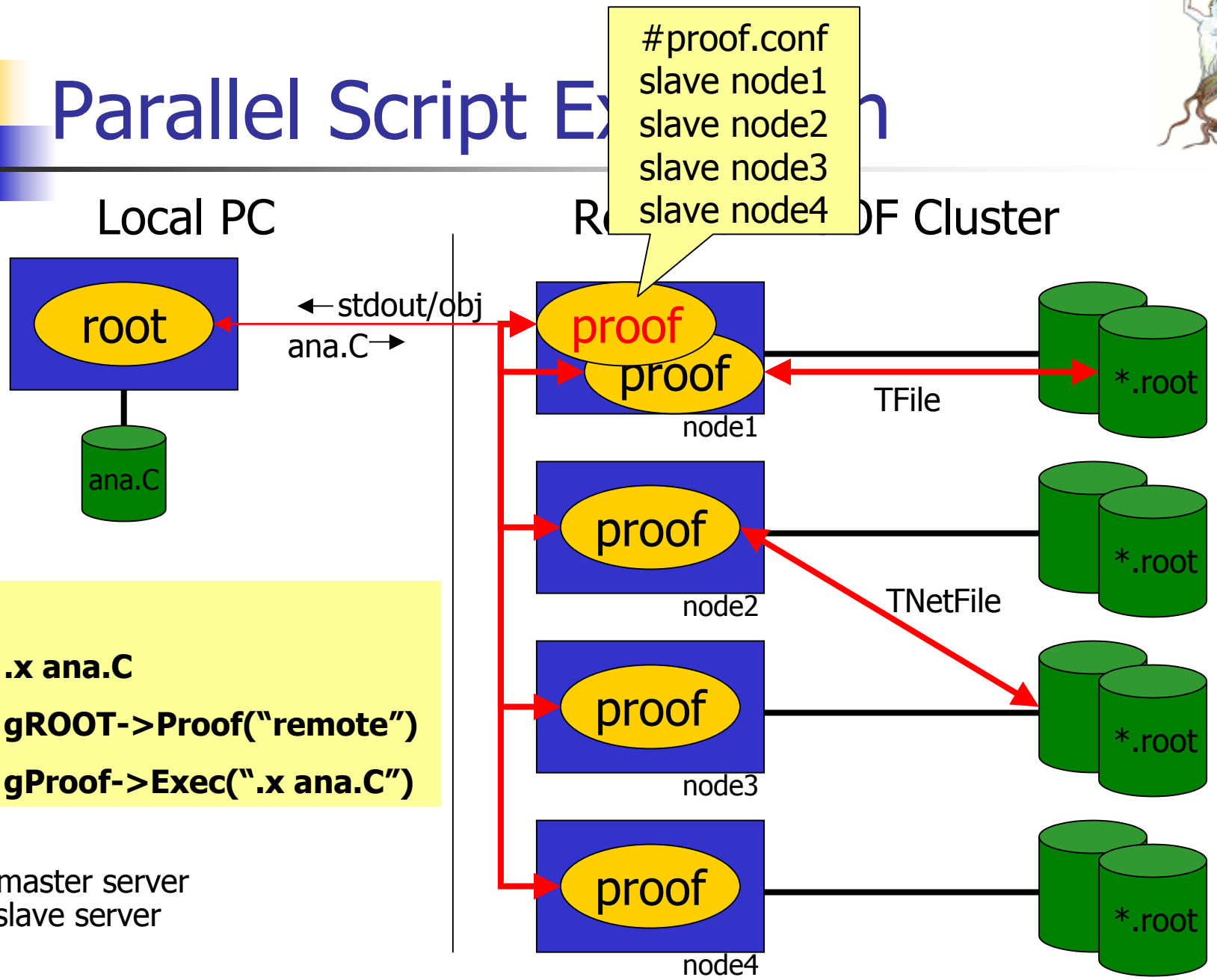




ROOT/PROOF and GRIDs



Parallel Script Ex n



$ root

root [0] .x ana.C

root [1] gROOT->Proof("remote")

root [2] gProof->Exec(".x ana.C")

proof = master server
proof = slave server

Summary

- We are implementing a powerful system designed for large scale data analysis with parallel architectures in a GRID context.
- The ROOT system is a framework providing a coherent object bus in DAQs, simulation, reconstruction and analysis phases.
- We have learnt a lot in the past 6 years, also following our 10 years of experience with PAW.
- Developing the system and at the same time supporting a rapidly growing users base is a demanding but also rewarding job.

ROOT: an Evolving System

- The ROOT system has been in continuous development since 1995 surviving major changes, major enhancements and an ever increasing number of users.
- Major developments must still be done to be ready for the fantastic LHC challenge.
- In the same way that Root2001 is far from the original Root1995, we expect that Root2006 will include many contributions reflecting the continuous changes and new ideas in the field of computing.
- This implies a strong cooperation between software developers in the major experiments.
- Root is being developed in very close cooperation with a cloud of software developers in small, medium and large experiments. Computer scientists from non-HEP fields are also contributing.

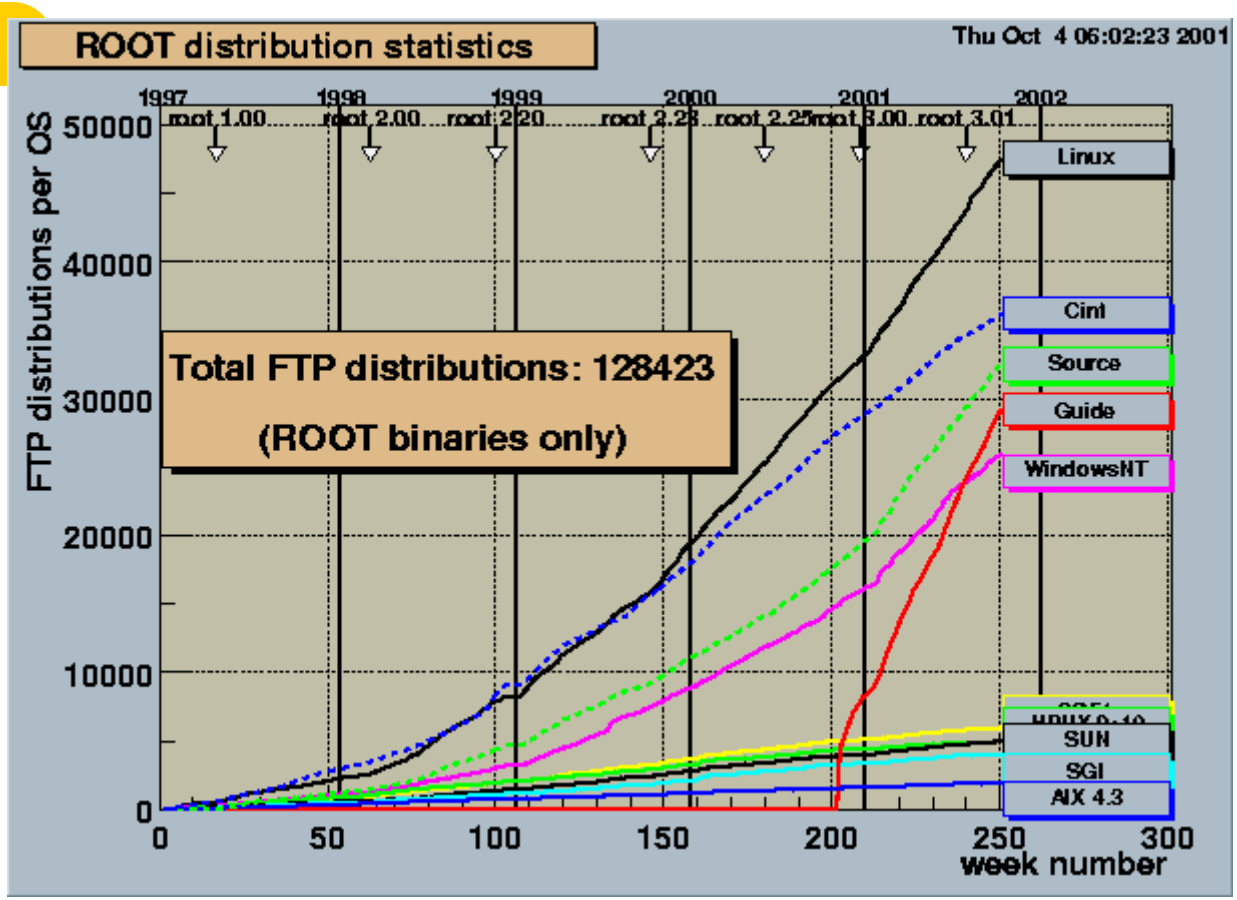
ROOT Downloads

128,000 binaries download

650,000 clicks per month

30,000 docs in 12 months

2200 reg users in roottalk



ROOT Users in the large experiments

ATLAS	133	WA98	25	LIGO	7
ALICE	120	BELLE	22	INTEGRAL	6
CDF	88	COMPASS	22	SNO	6
PHENIX	86	KLOE	19	CELESTE	5
CMS	85	ALEPH	18	HESS	5
STAR	82	OPAL	17	VIRGO	5
JLAB	77	AUGER	16		
D0	70	MINOS	16		
BABAR	69	NOMAD	16		
H1	48	BRAHMS	15		
L3	43	GLAST	14		
HERAB	37	AMS	12		
NA49	37	NA45	12		
LHCB	35	NA48	11		
DELPHI	34	AMANDA	10		
ZEUS	32	....			
HADES	27	....			
PHOBOS	27				

Registered users in the ROOT system