from ROOT to BOOT
from BOOT to ROOT

Rene.Brun@cern.ch

26 March 2007

Starting Point

- Current Developments in "standard ROOT mode" will continue in the coming months/years.
- With more and more features and improvements
- With support for parallelism, multi-threading, etc
- However, there are several important features missing in ROOT. If implemented, they could simplify the life of many users.
  - Speed-up installation, make it transparent.
  - Mixing versions (just in case a bug is fixed in one lib or a nice feature added in another lib).
  - Running from a web browser, from anywhere and with execution anywhere.
  - Embedded & hyperlinked help

Observations

- A considerable amount of time is required to install the experiment software.
- Porting software on a new platform is non trivial.
- Many dependencies between libraries. Many providers.
- Only a small fraction of the software is used.
- The installation costs in time and disk space.
- Users hesitate before installing a new version, and once installed, it is not obvious to go back to an older version.
- This is in total contradiction with the idea of the GRID.
- The GRID software should help this process and not make it worst.

Observations 2

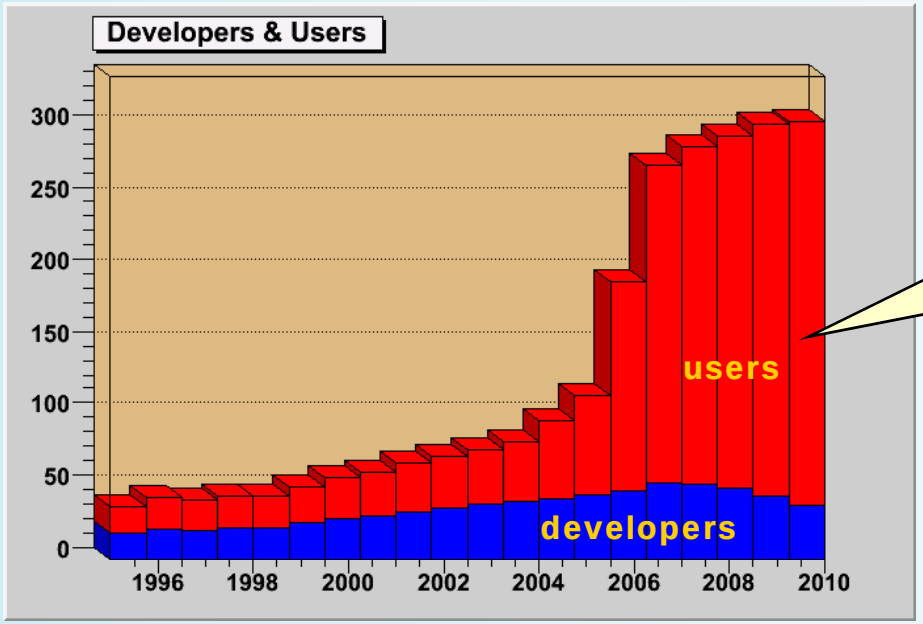
- The users profiles evolve (more physics analysis users than developers when data taking approaches).
- Experiment frameworks evolve too. Data Analysis should be possible without the experiment framework.
- C++ is, by far, the dominant language.
- Object dictionaries are recognized to be important:
  - For I/O
  - For Interpreters
  - For the GUI (signal & slots)
  - For the new generation of event displays.
- Size of dictionaries is becoming problematic
- The size of the executable modules is even more problematic.

Some LHC parameters collected 1 year ago

	Alice	Atlas	CMS	ROOT
number of lines in header files	102282	698208	104923	153775
classes total	1815	8910	???	1500
classes in dict	1669	>4120 2140	835	1422
lines in dict	479849	455705	103057	698000
classes c++ lines	577882	1524866	277923	857390
total lines Classes+dict	1057731	???	380980	1553390
total f77 lines	736751	928574	???	3000
directories	540	19522	<500	958
comp time	25’	750’	90’	30’
lines compiled/s	1196	50 (70)	71	863

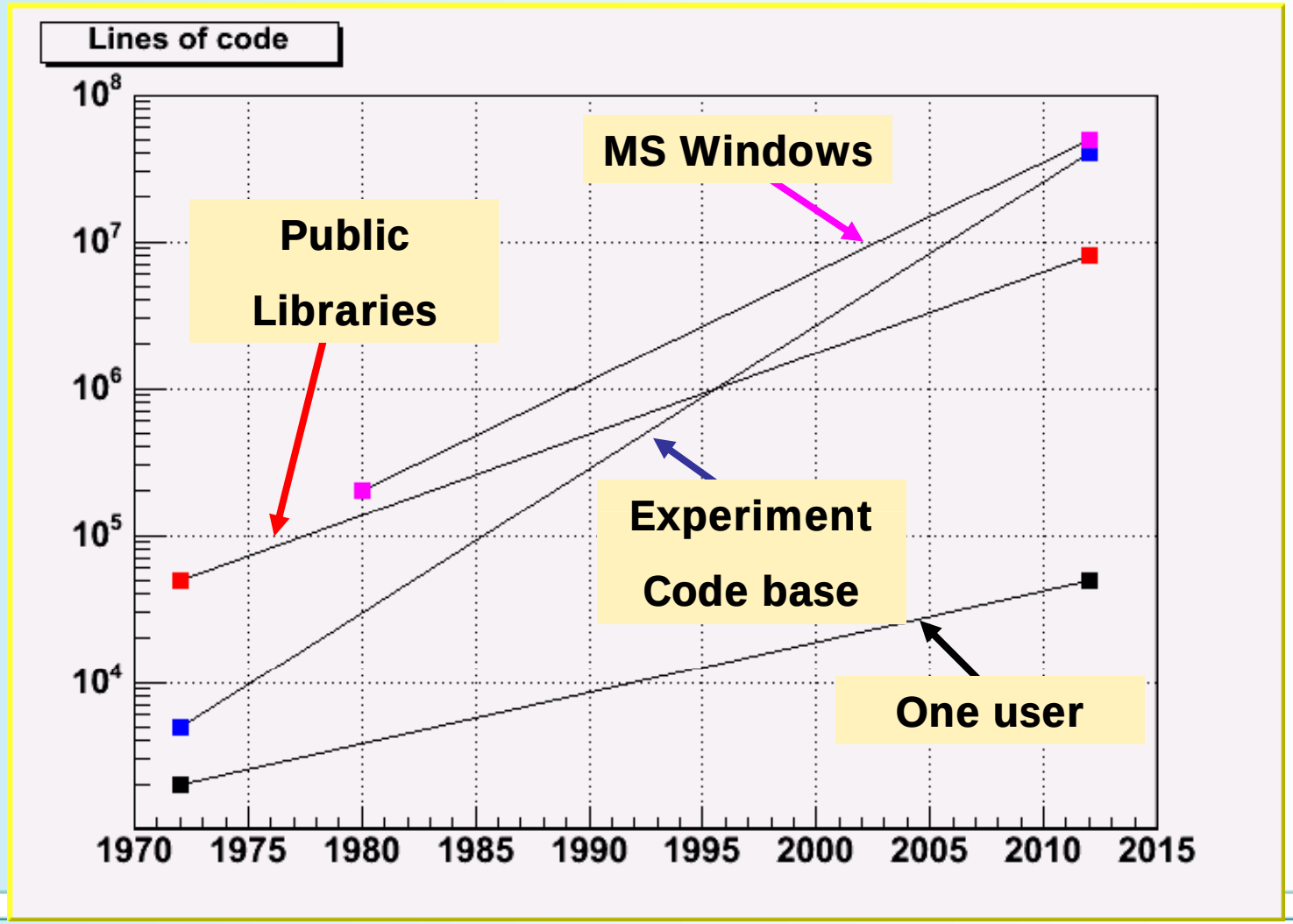
User Profile is evolving

- The ratio developper/user is quickly changing.
- Applications are more and more distributed.
- OS and machines evolve rapidly.

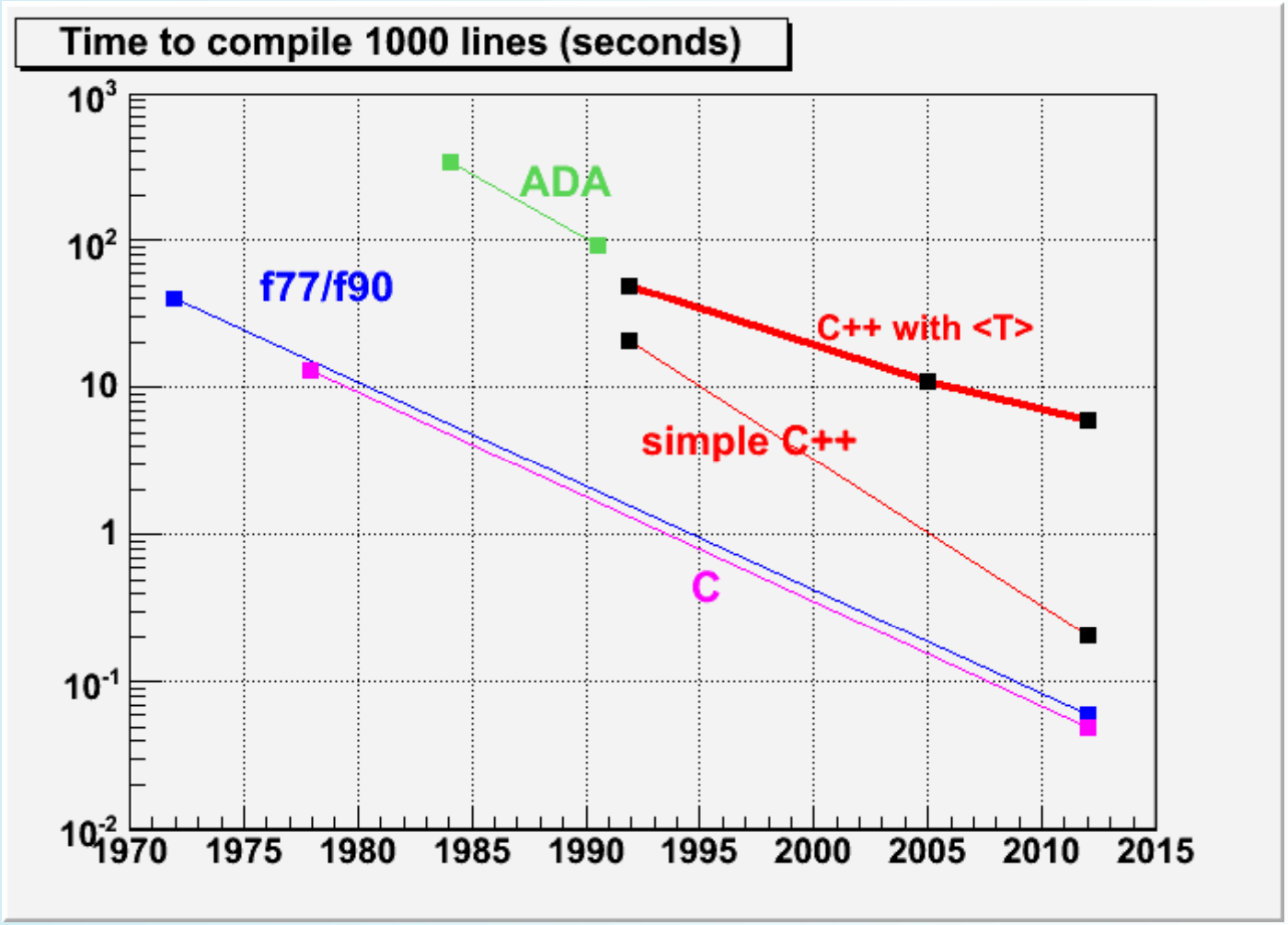


They require
Improved UI
More robustness
or anything
simplifying
their life

Program Size (lines of code)



Time to compile

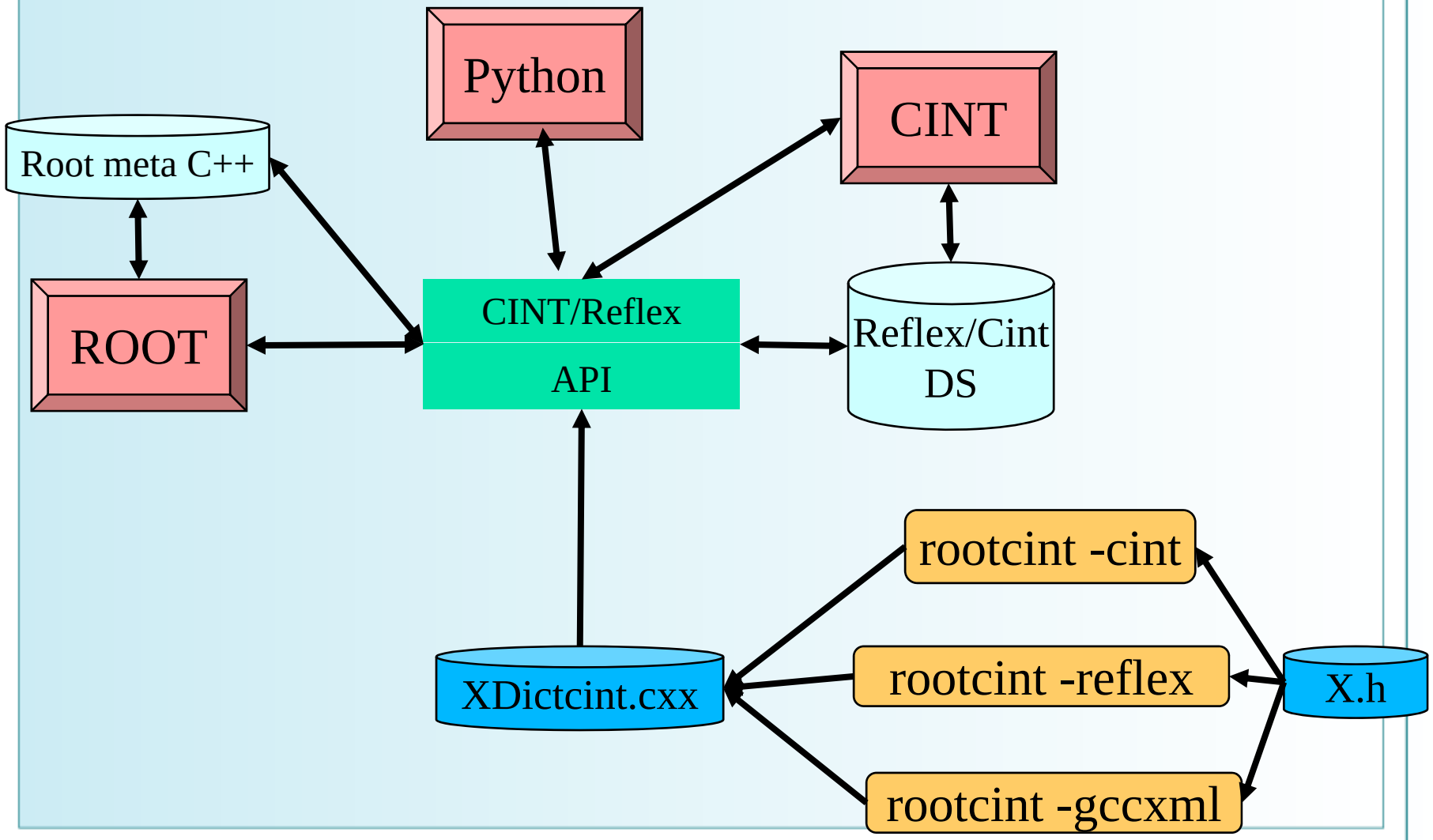


Problem with dictionaries

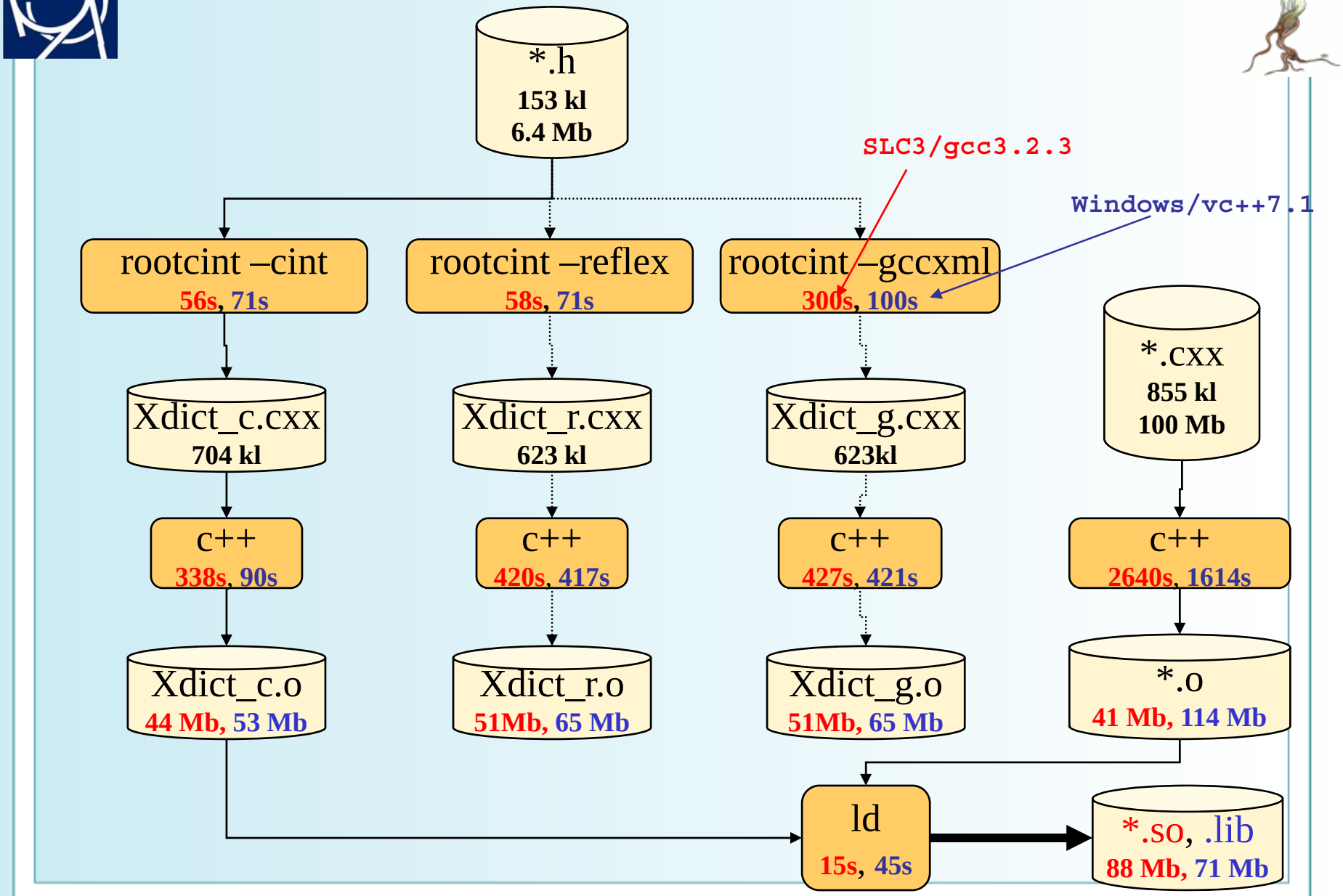
- Today cint/reflex dictionaries are machine dependent.
- They represent a very substantial fraction of the total code.
- We could make a very large fraction machine independent.
- Interface to functions could be reduced with standard ABIs.
- Dict data structures could be saved to a root file instead of generating the code producing these ds.
- In this case, one will import only the ds for the classes really used (I/O or interpreter)

	*.o	G_*.o	Dict %
mathcore	2674520	2509880	93.8%
mathmore	598040	451520	75.5%
base	6920485	4975700	71.8%
physics	786700	558412	71.0%
treeplayer	2142848	1495320	69.8%
geom	4685652	3096172	66.1%
tree	2696032	1592332	59.1%
g3d	1555196	908176	58.4%
geompainter	339612	196588	57.9%
graf	2945432	1610356	54.7%
matrix	3756632	2020388	53.8%
meta	1775888	909036	51.2%
hist	3765540	1914012	50.8%
gl	2313720	1126580	48.7%
gpad	1871020	781792	41.8%
histpainter	538212	204192	37.9%
minuit	581724	196496	33.8%

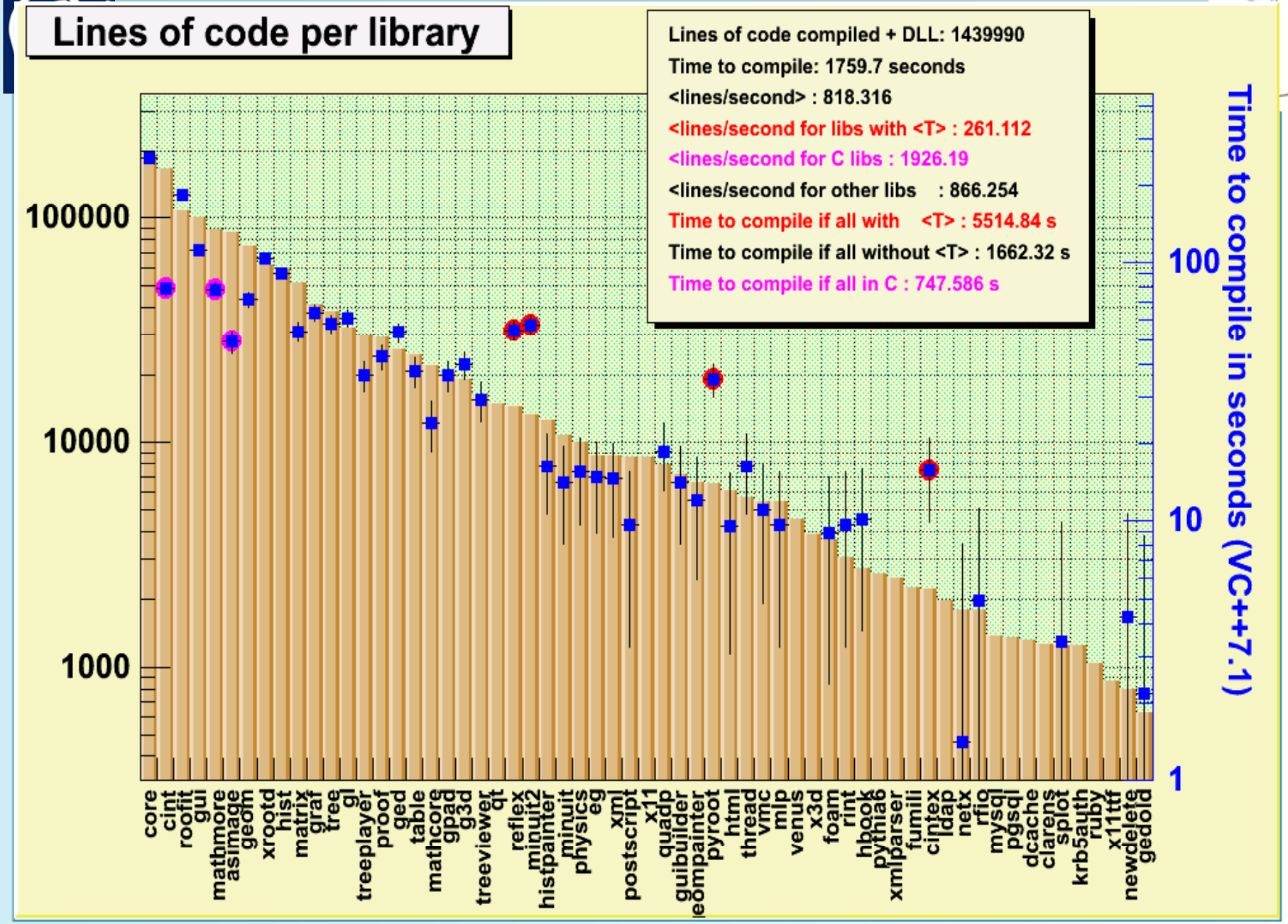
Dictionaries : situation in 2007



ROOT source, bins, dict,libs

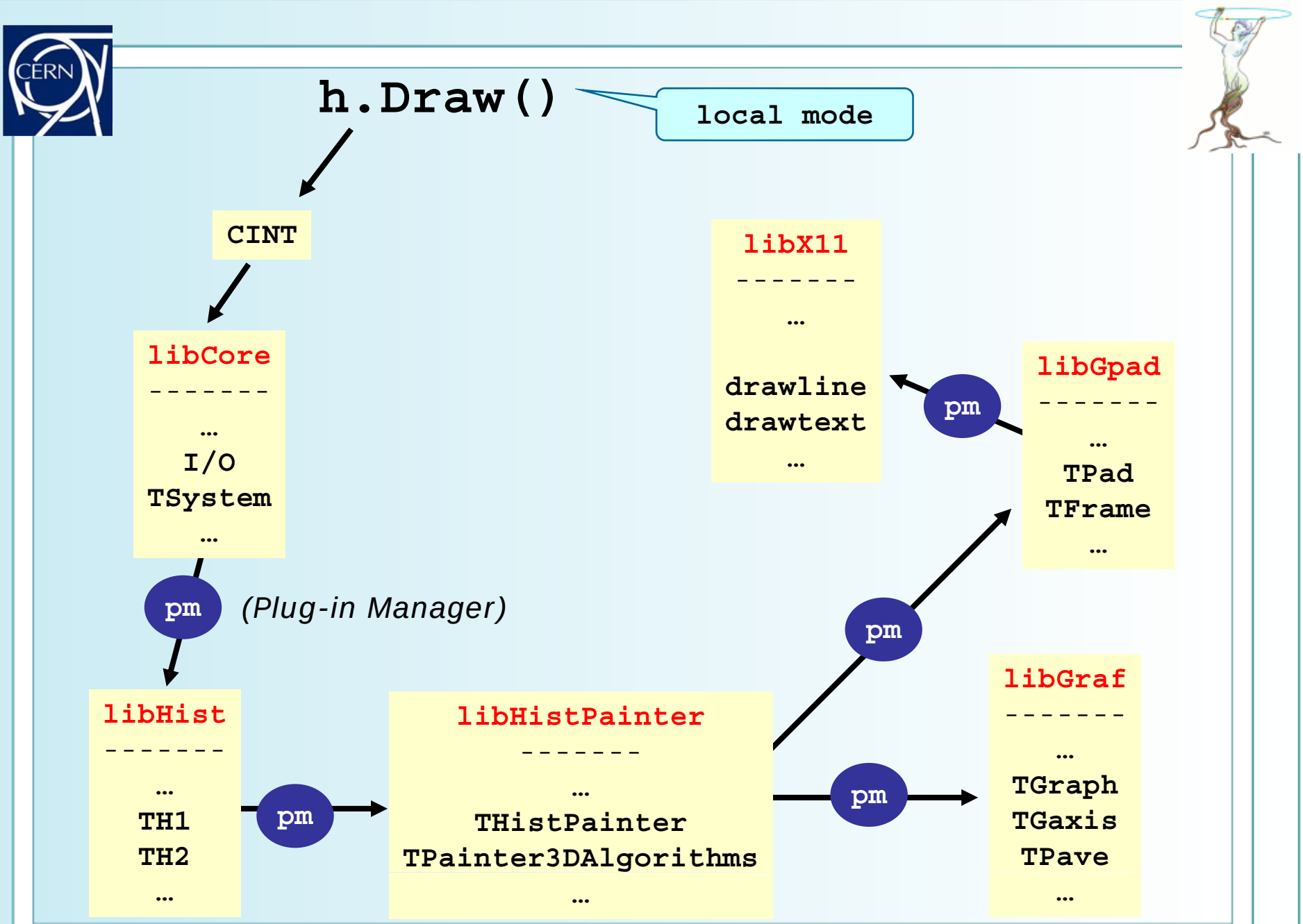


Lines of code per library



From static modules to plug-ins

- app.exe = (main.o, x.o, y.o)
- app.exe = (main.o, x.o, lib1.a, lib2.a)
- app.exe = (main.o, x.o, lib1.a, lib2.so, lib3.so)
- app.exe = (main.o, libs.so) + dyn libs.so
- app.exe = (main.so,libs.so) + plug-in manager


h.Draw()
local mode
CINT
libCore
I/O
TSystem
pm
(Plug-in Manager)
libX11
drawline
drawtext
libGpad
TPad
TFrame
libHist
TH1
TH2
libHistPainter
THistPainter
TPainter3DAlgorithms
libGraf
TGraph
TGaxis
TPave

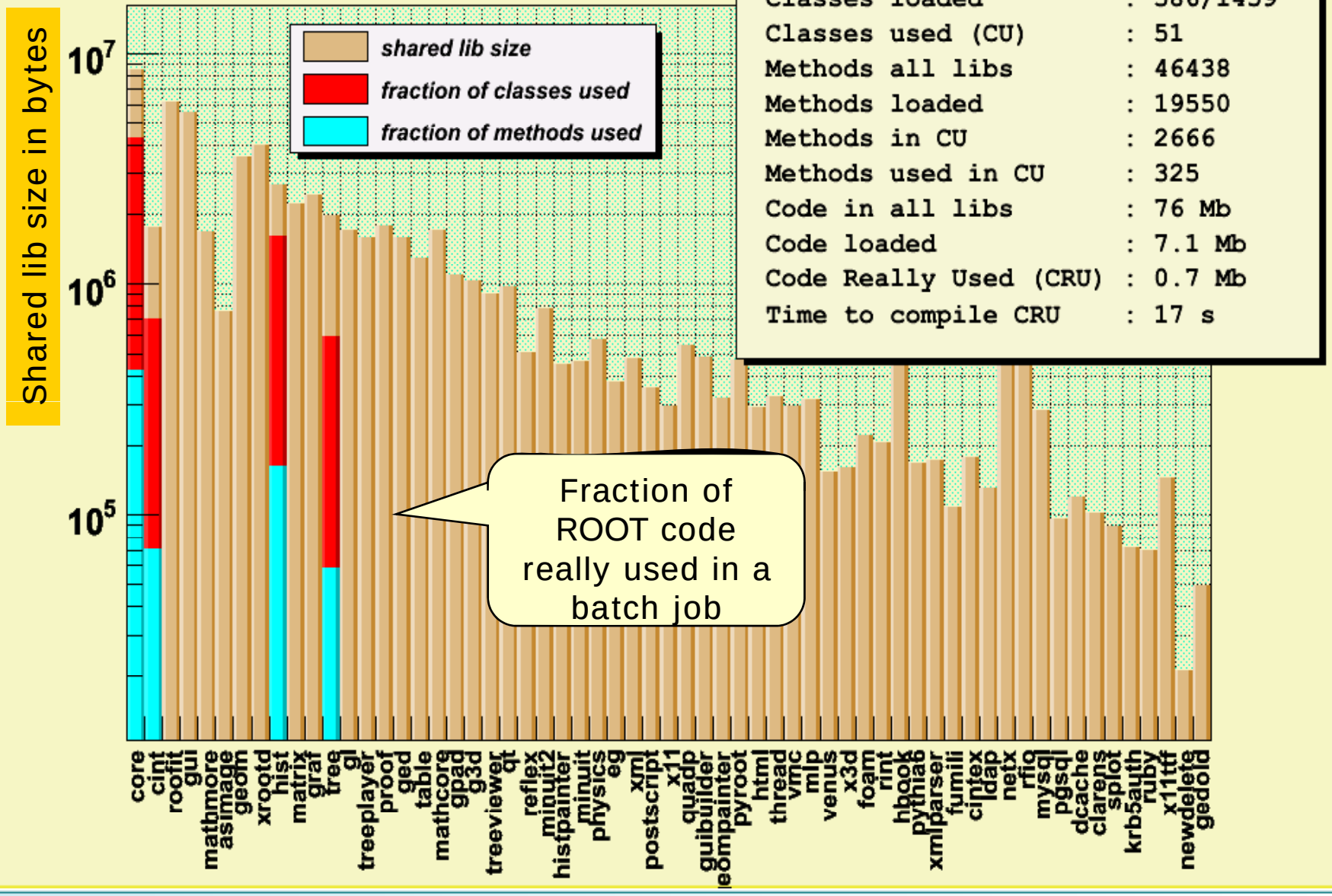
Shared libs

- Shared libs are essential for today large applications.
- They optimize the development time if inter-library dependencies is correctly managed.
- The plug-in manager is an essential component that minimizes the number of libraries linked at the start of an application.
- However, a large number of libs may be a killer, in particular for interactive applications.
- Because of large compilation times, most experiments export pre-compiled shared libs.
- These libs are compiled for maximum portability and do not always use efficiently local processors capabilities.

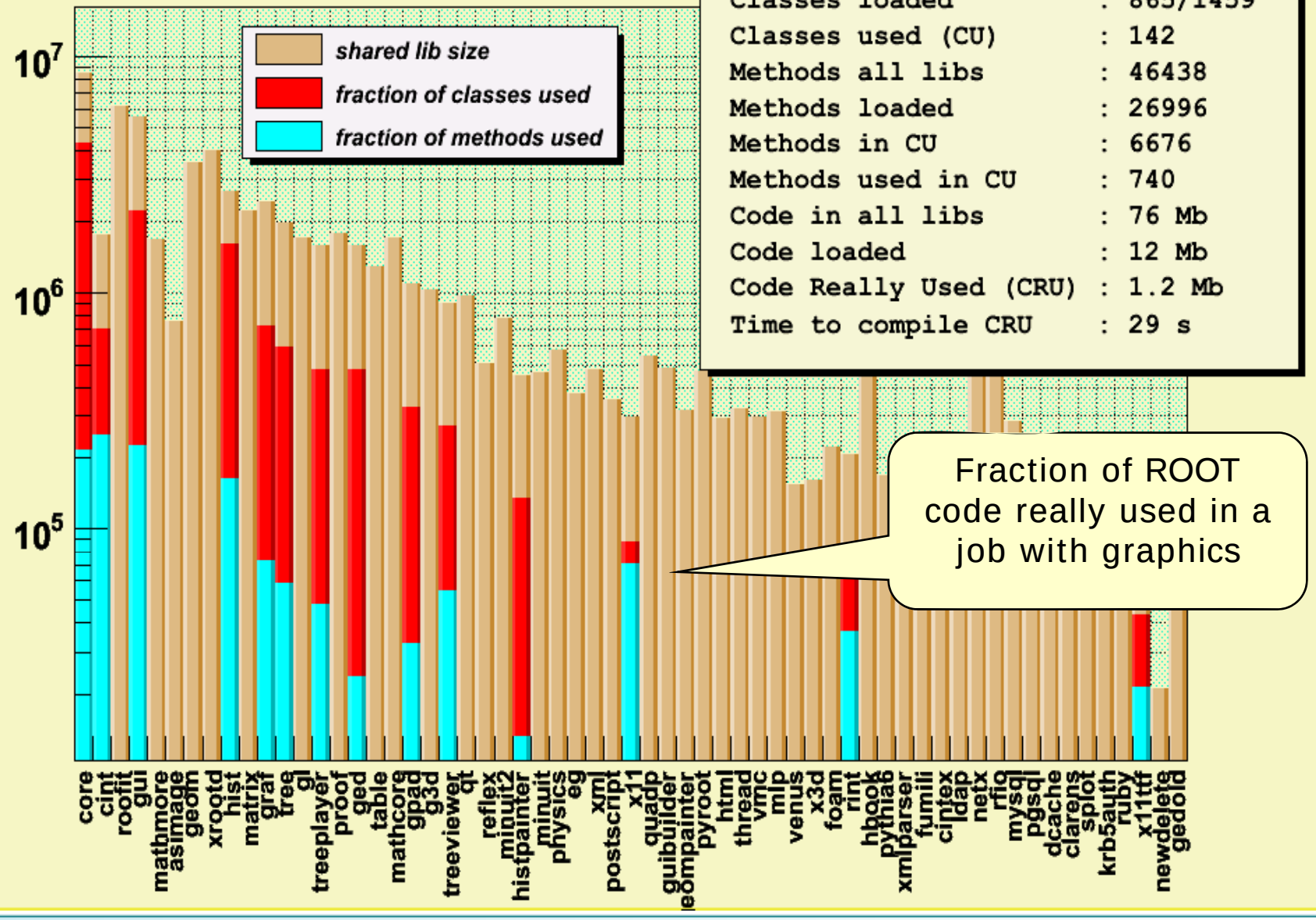
Exported Symbols

- Time to load a shared lib is grosso modo
- time = size * n * log(N)
  - size = shared lib size in bytes (mapped I/O)
  - n = number of exported symbols in lib
  - N = number of existing exported symbols in previously loaded shared libs
- A good compromise must be found between the number of libraries and their size (modularity vs performance)
- GCC4 & Windows allow selection of symbols accessible from outside shared lib (“exported”) .
- Currently most applications export all C++ symbols !

code used in a batch use case



code used in a graphics use case



Libs used	: 14/86
Classes loaded	: 865/1459
Classes used (CU)	: 142
Methods all libs	: 46438
Methods loaded	: 26996
Methods in CU	: 6676
Methods used in CU	: 740
Code in all libs	: 76 Mb
Code loaded	: 12 Mb
Code Really Used (CRU)	: 1.2 Mb
Time to compile CRU	: 29 s

Fraction of ROOT code really used in a job with graphics

Downloading only a subset of libraries?

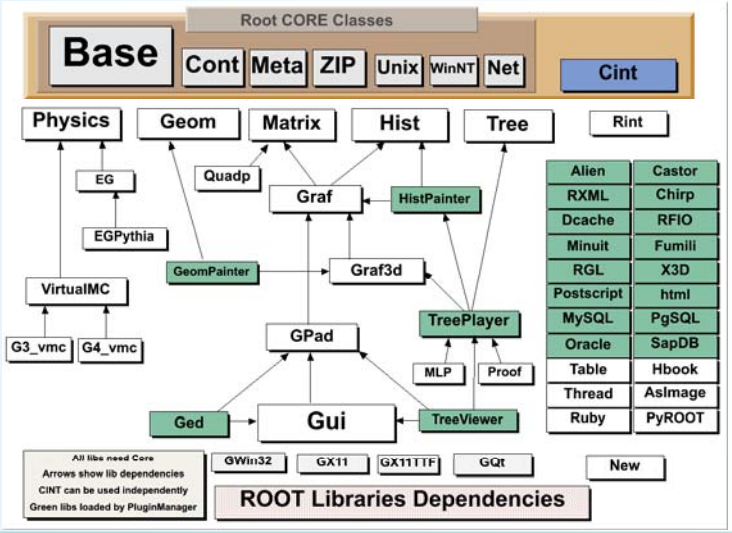
```
  145742  libASImage.lib
  112206  libASImageGui.lib
   54076  libASPluginGS.lib
 1462634  libAfterImage.lib
  162082  libCint.lib
  568696  libCintex.lib
 3985542  libCore.lib
  188566  libEG.lib
  134170  libEGPythia6.lib
  101782  libFFTW.lib
  164120  libFitPanel.lib
  105732  libFoam.lib
   67246  libFumili.lib
  687018  libGdml.lib
  691048  libGed.lib
 1788574  libGeom.lib
  610488  libGeomBuilder.lib
  200660  libGeomPainter.lib
  454560  libGpad.lib
  909146  libGraf.lib
  544992  libGraf3d.lib
 2665060  libGui.lib
  336516  libGuiBld.lib
  129332  libHbook.lib
  999044  libHist.lib
  195110  libHistPainter.lib
 2918132  libHtml.lib
  135346  libMLP.lib
```

```
  135346  libMLP.lib
 1649982  libMathCore.lib
  485136  libMathMore.lib
 1104336  libMatrix.lib
  163642  libMinuit.lib
 1185602  libMinuit2.lib
  465046  libNet.lib
  147158  libNetx.lib
   12362  libNew.lib
  148170  libOracle.lib
  300436  libPhysics.lib
  157772  libPostscript.lib
  791738  libProof.lib
 2224048  libProofPlayer.lib
  275216  libProofx.lib
 2244766  libPyROOT.lib
  246878  libQuadp.lib
   82694  libRCastor.lib
   80434  libRFIO.lib
 2982494  libRGL.lib
  630874  libRIO.lib
  131936  libRODBC.lib
 3709214  libReflex.lib
  124508  libRint.lib
 3150774  libRooFit.lib
  188664  libRootAuth.lib
   75334  libSPlot.lib
  380128  libSQL.lib
```

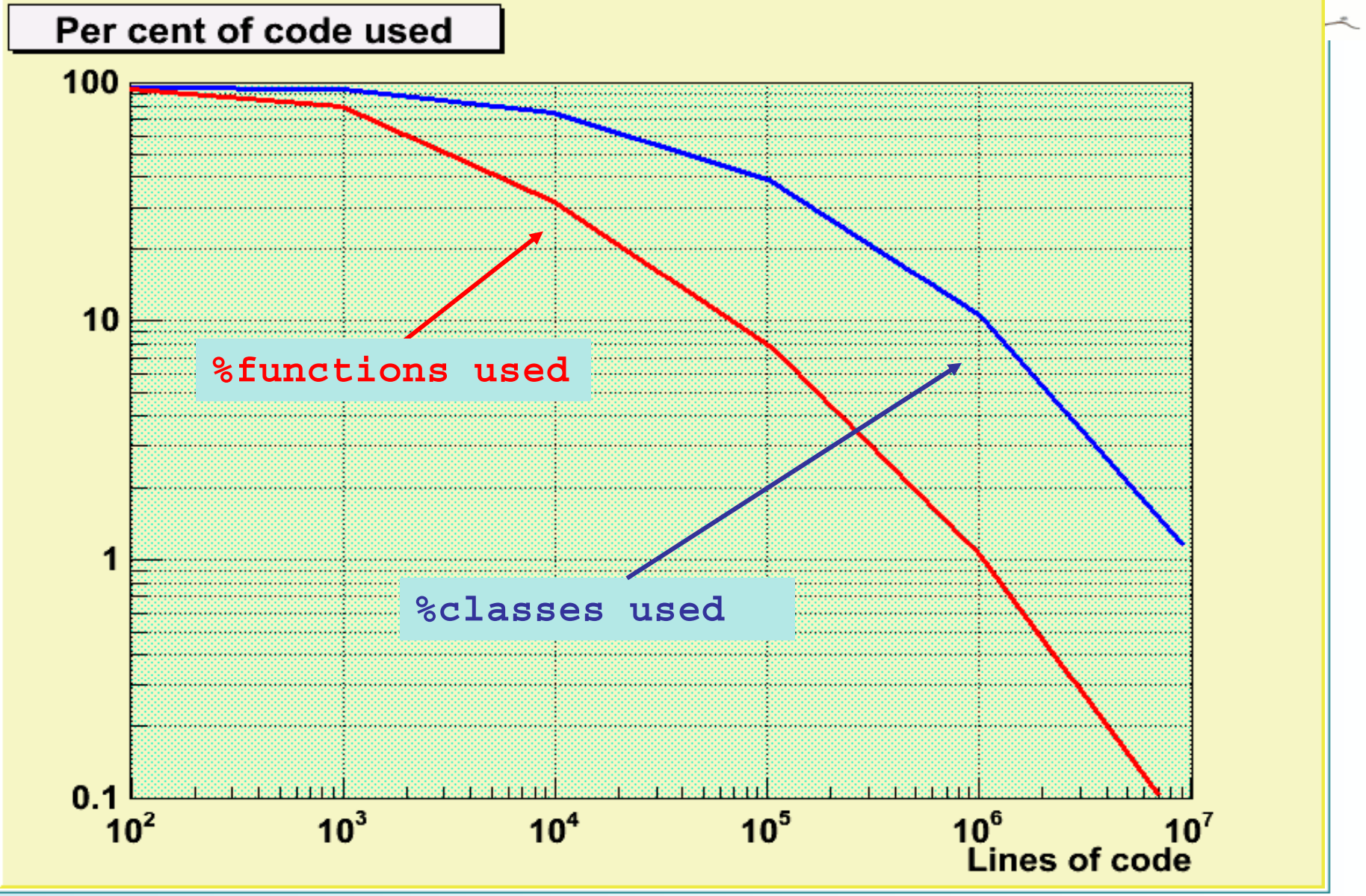
```
  388176  libSessionViewer.lib
 1390928  libSmatrix.lib
  160874  libSpectrum.lib
   69134  libSpectrumPainter.lib
 3308768  libTMVA.lib
  698472  libTable.lib
  194856  libThread.lib
 1012952  libTree.lib
 1053616  libTreePlayer.lib
  244666  libTreeViewer.lib
  168870  libUnuran.lib
  135486  libVMC.lib
  653740  libWin32gdk.lib
  345720  libXMLIO.lib
  130272  libXMLParser.lib
    4734  libXrdProofd.lib
   29958  libdequeDict.lib
 2494354  libfreetype.lib
   29778  liblistDict.lib
   29778  libmap2Dict.lib
   29626  libmapDict.lib
   30550  libmultimap2Dict.lib
   30404  libmultimapDict.lib
   30404  libmultisetDict.lib
  234532  libpcre.lib
   29626  libsetDict.lib
   30108  libvectorDict.lib
```

Can we gain with a better packaging?

- Yes and no. Going beyond the recent ROOT restructuring is difficult.
- One shared lib per class implies more administration, more dictionaries, more dependencies.
- 96 shared libs for ROOT is already a lot. 1500 would be non sense
- A small CORE library is essential.
- Plug-in Manager helps



Fraction of code really used in one program



Interpreter & Compiler integration

```
root > .x script.C
root >  DoSomething(...);
root > .x script.C++
root > .x script.C+
```

execute file script.C

execute function DoSomething

compile file script.C
and execute it

compile file script.C
if file has been modified.
execute it

```
gROOT->ProcessLine(".L script.C+");
gROOT->ProcessLine("DoSomething(...)");
```

same from
compiled
or interpreted
code

Possible Progress with Interpreters

- Eliminate the stub interface to call C/C++ functions.
  - This is already possible in CINT with C libraries.
  - It will be possible with C++ when a standard ABI will be available, otherwise compiler&linker dependent.
- If compiler is fast enough (eg C), use the interpreter only for organizing the top level.
- If next C++ provides introspection, one could eliminate
  - the header files parser
  - 95 per cent of the dictionary structure in memory
- A good argument to have the interpreted and compiled code being in the same language!
- But WHEN ???????

Consequences

- The fact that only a very small fraction of the total code base is used has important consequences.
- We must turn this apparent problem into a great feature.
- BOOT: a proposal to solve this problem.

Proposal for a new scenario

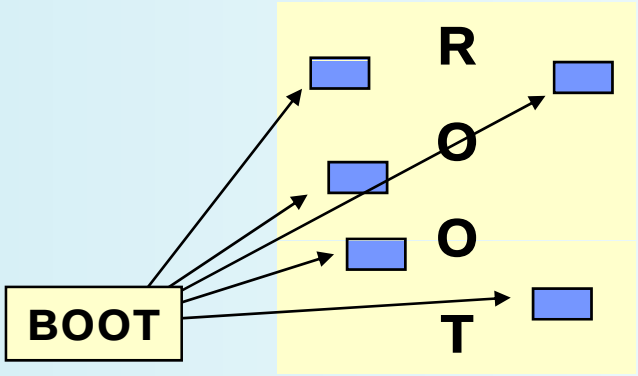
Introducing

BOOT

A Software Bootstrap system

What is BOOT?

- A small system to facilitate the life of many users doing mainly data analysis with ROOT and their own classes (users + experiment).
- It is a very small subset of ROOT (5 to 10 per cent)
- The same idea could be extended to other domains, like simulation and reconstruction.



What is BOOT (2)?

- A small, easy to install, standalone executable module (< 5 Mbytes)
  - One click in the web browser
- It must be a stable system that can cope with old and new versions of other packages including ROOT itself.
- It will include:
  - A subset of ROOT I/O, network and Core classes
  - A subset of Reflex
  - A subset of CINT (could also have a python flavor)
  - Possibly a GUI object browser (**BROOTER**)
- From the BOOT GUI or command line, the referenced software (URL) will be automatically downloaded and locally compiled/cached in a transparent way.

What is BOOT (3)?

- No binary files or shared libs
- Start from the source URL whenever possible
- But an option to download only a subset of the libs could be implemented too.
- Compile into local cache and reuse at next session.
- A tool is provided to convert a CVS source tree into a compact file that also includes the dictionary data structures and the classes/functions documentation.
- Compile with the best options for the local hardware.
- A front-end to C++ (with the corresponding C++ code generation) for very frequent use cases.

Downloading source, binaries, doc, etc

Availability

ROOT is available in binary and source form. The binaries are available for most supported platforms. T
CVS and can easily be compiled on any supported platform/compiler combo.

For what is new in this version see the development notes.

Via Anonymous FTP

Source

- ROOT 5.15/04 complete source tree for all systems (21.6 MB). NEW
  After unpacking read the file root/README/INSTALL.

source.tar.gz = 22 MBytes

Documentation

- ROOT 5.15/04 classes html documentation compressed tar file(199 MB). NEW

html.tar.gz = 200 MBytes

Binaries

Note 1: Before downloading a binary version make sure your machine contains the right run-time enviro
compiled with, e.g., gcc4.0 on a platform where only gcc 3.2 is installed. In such cases you'll have to ins
build the binaries below.

To install, unzip and untar the file. For example:

```
$ gunzip   root_v5.15.04.Linux.slc4.gcc3.4.tar.gz
$ tar xvf  root_v5.15.04.Linux.slc4.gcc3.4.tar
```

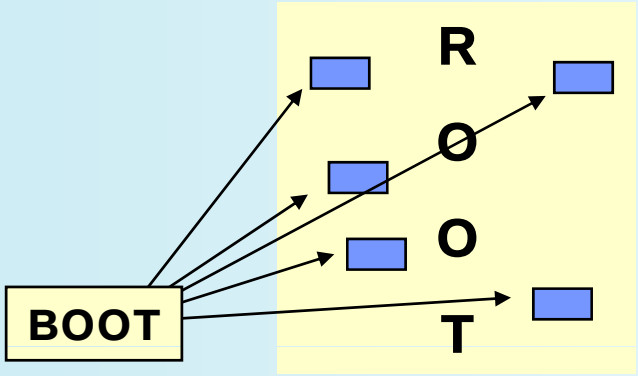
One binary = 40 MBytes

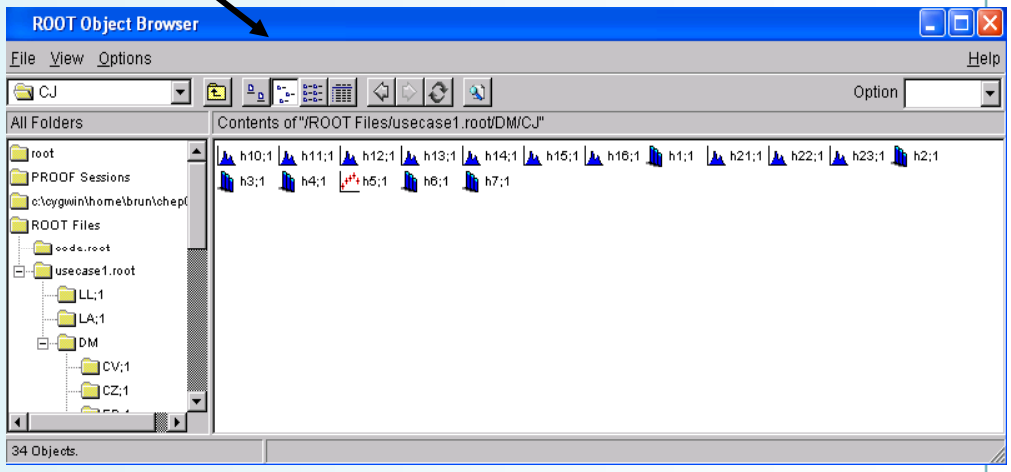
This will create the directory root. Before getting started read the file README/README. *Remember*, yo
this web site at the location Classes and Members. The distribution also contains all tutorials and a set o

Linux

BOOT: Use Case 1

- Assumes BOOT already installed on your machine user@xxx.yyy.zzz
- Nothing else on the machine , except the compiler (no ROOT, etc)
- Import a ROOT file containing histograms, Trees and other classes (usecase1.root)
- Browse contents of file
- Draw an histogram





Use Case 1

Usecase1.root
(2 Mbytes)
Contains references (URL) to classes in namespace ROOT

Local cache with the source of the classes really used
+
binaries for the classes or functions that are automatically generated from the interpreter
(like ACLIC mechanism)

user@xxx.yyy.zzz

http://root.cern.ch/source.root

This is a compressed ROOT file containing the full ROOT source tree automatically built from CVS
(25 Mbytes)
+

ROOT classes dictionary DS generated by Reflex
(5 Mbytes)
+

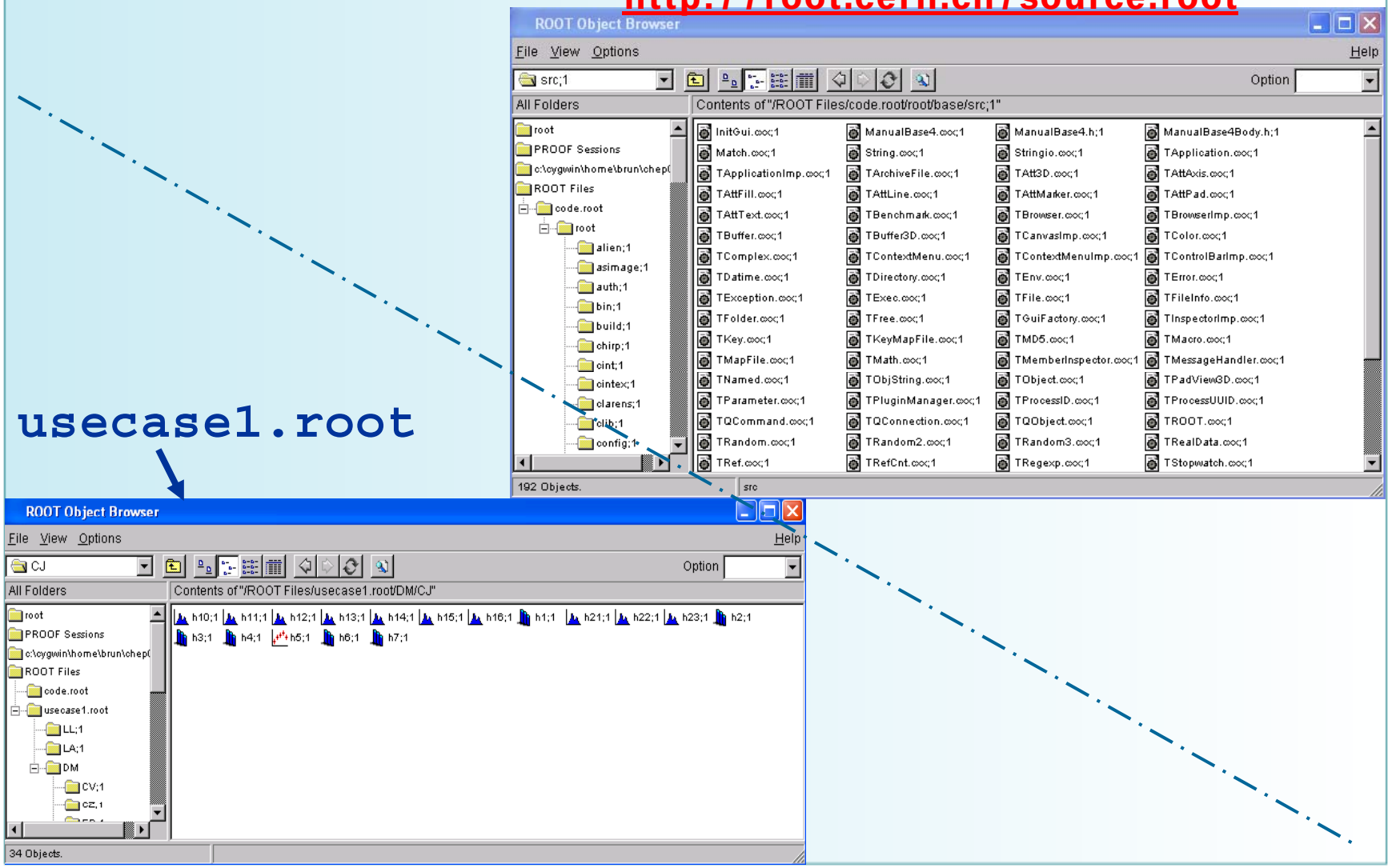
The full classes documentation Objects generated by the source parser
(5 Mbytes)

pcroot@cern.ch

Use Case 1 pictures

http://root.cern.ch/source.root

usecase1.root



Use Case 2

- BOOT already installed
- Want to write the shortest possible program using some classes in namespace ROOT and some classes from another namespace YYYY

```
//This code can be interpreted line by line
//executed as a script or compiled with C/C++
//after corresponding code generation
use ROOT=http://root.cern.ch/root5.14/source.root
use YYYY=http://cms.cern.ch/packages/yyyy
h = new TH1F("h","example",100,0,1);
v = new LorentzVector(….);
gener = new myClass(v.x());
h.Fill(gener.Something());
h.Draw();
```

Use Case 3

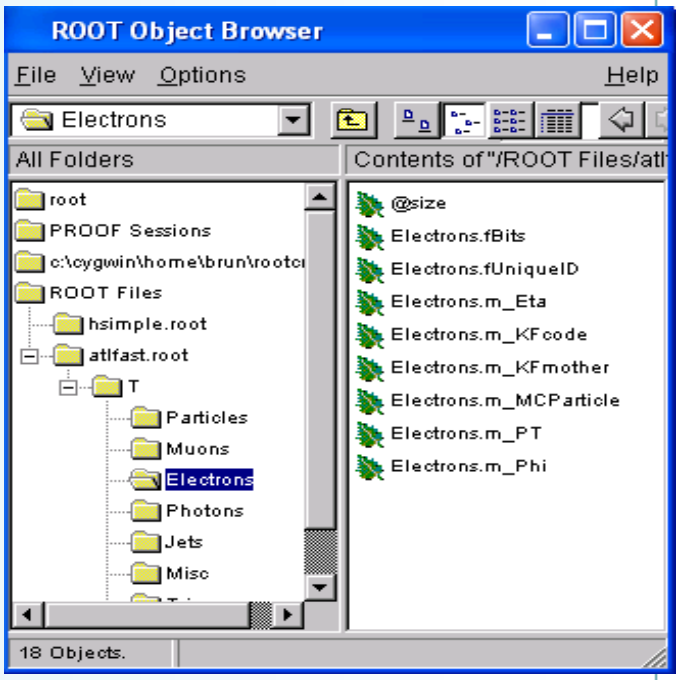
- A variant of Use Case 2
- A bug has been found in class LorentzVector of ROOT and fixed in new version ROOT6

```
use ROOT, YYYY=http://cms.cern.ch/packages/yyyy
use ROOT6=http://root.cern.ch/root6.00/code.root
use ROOT6::LorentzVector
h = new TH1F("h","example",100,0,1);
v = new LorentzVector(….);
gener = new myClass(v.x());
h.Fill(gener.Something());
```

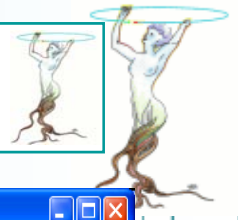
Use Case 4: Specialized Code Generators

```
use ATLFAST=http://atlas.cern.ch/atlfast/atlfastcode.root
TFile f("mcrun.root");
for each entry in f.T
   for each electron in Electrons
      if(electron.m_Eta > 1) h.Fill(electron.m_Pt);
h.Draw
```

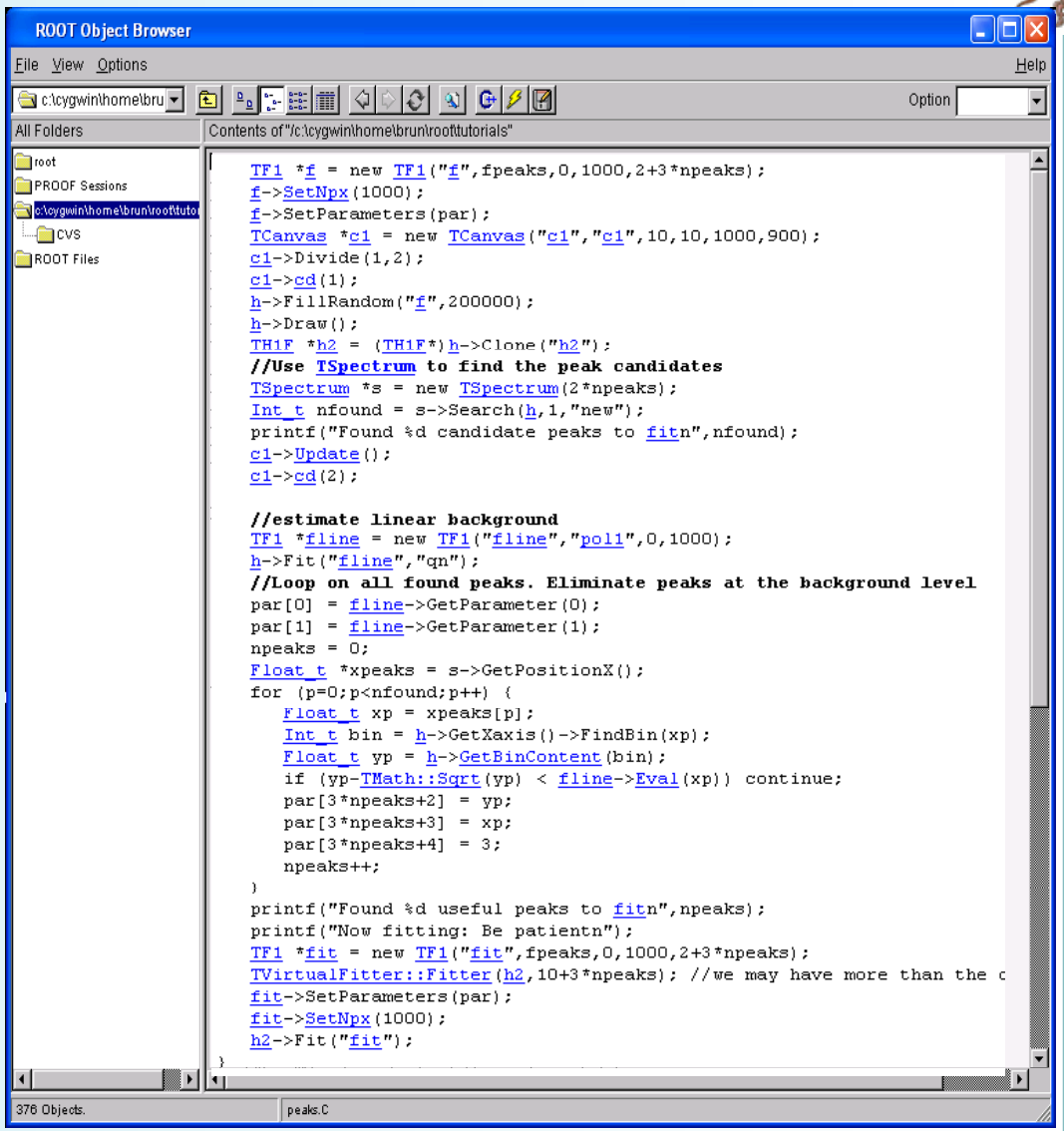
- High Level ROOT Selector understanding named collections in memory (ROOT,STL) or collections in ROOT files.
- PROOF compliant
- Extension of TTree::MakeProxy code generator.
- Do not read referenced but unused branches.



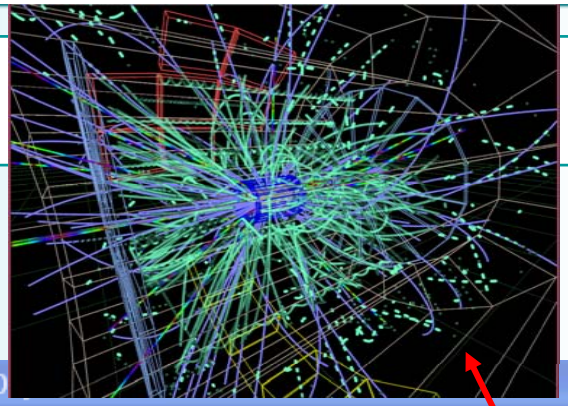
Use Case 5: Dynamic HELP, Dynamic html



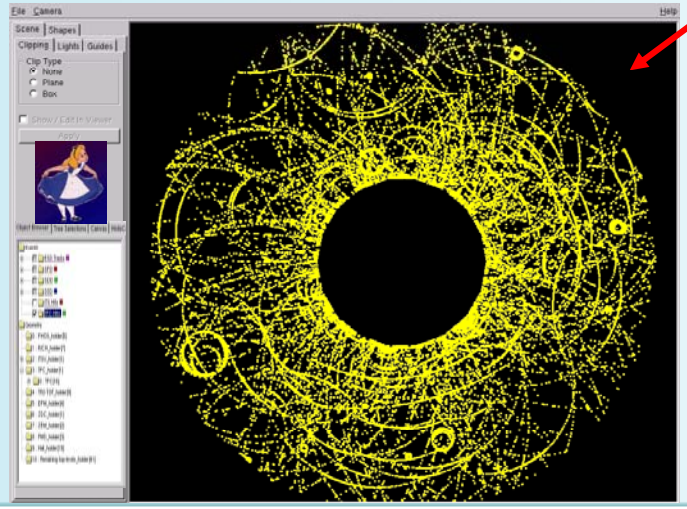
- Source files and scripts are browsable in html format generated dynamically.
- Combination of new version of THtml and the new GUI widget TGHtml.
- Both classes use extensively the Reflex dictionary and the pre-digested documentation.

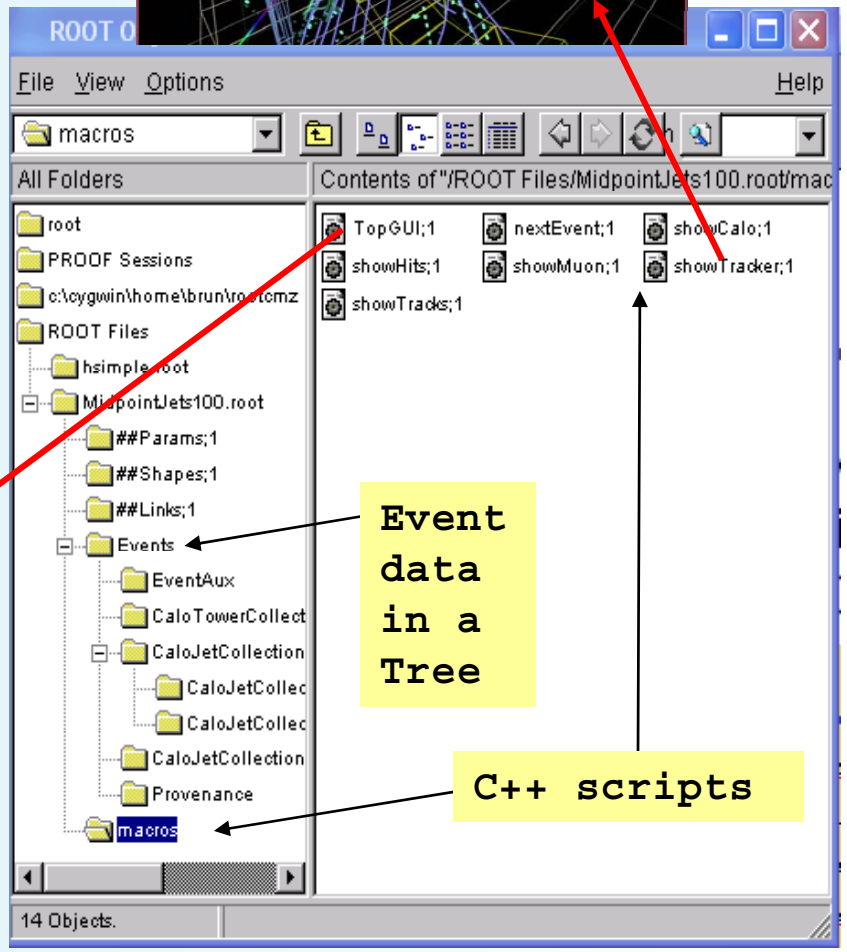


Use Case 6: Event Displays



- In general, Event Displays require the full experiment infrastructure (Pacific, Obelix, WonderLand, Crocodile).
- This is complex and not good for users and OUTREACH.
- A data file with the visualization scripts is far more powerful
- <u>This implies that the GUI must be fully scriptable</u>. This is the case for ROOT GUI.





BOOT: Dream or Reality ?

What did ve achieve so far ?

What next?

The challenges

- 1-Efficient access to remote source or binary files.
- 2-Local caches and proxies .
- 3-Improve compiler & linker interfaces
- 4-Minimize size of executable, (ie strict minimum to download).
- 5-Plug-in for an existing browser or develop a new browser? Must run with native ROOT GUI.
- 6-Authentication issues when running on a remote machine (need for ssh?) in client/server mode.
- 7-Develop extensions to C++ that can easily be converted to standard C++ via CINT/ACLIC.

Efficient access to remote files

- This was one of the first problems to solve.
- And we solved it (or very close).
- The solution (TFileCacheRead/TFileCacheWrite) is also a general improvement for ROOT I/O.
- A spin-off of this exercise was TTreeCache
- See talk by Leo

Example of TTreeCache improvement

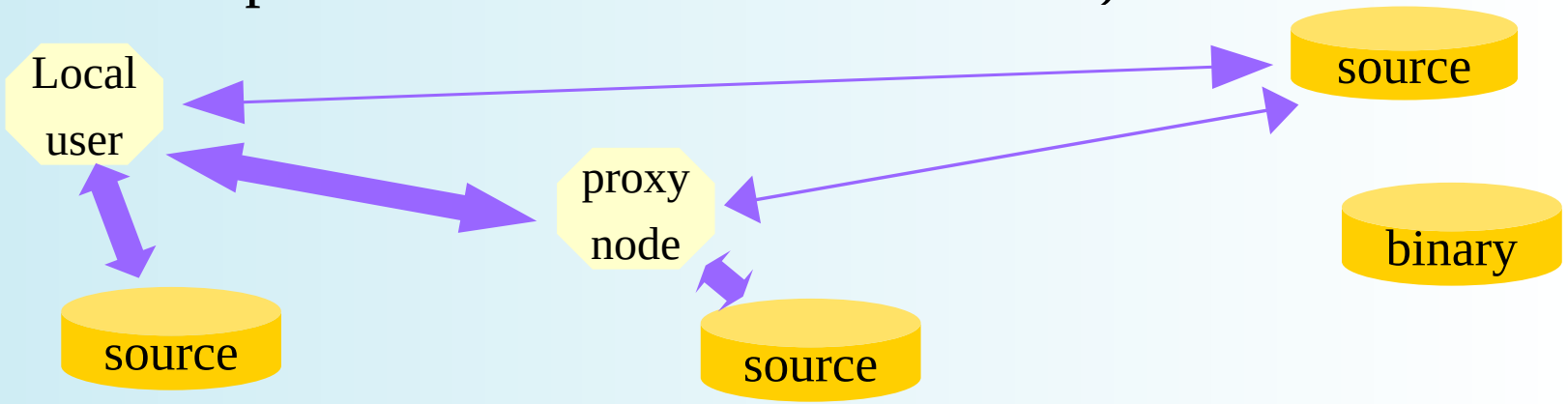
- The file is on a CERN machine connected to the CERN LAN at at 100MB/s.
- The client **A** is on the same machine as the file (local read)
- The client **B** is on a CERN LAN connected at 100 Mbits/s with a network latency of 0.3 milliseconds (P IV 3 Ghz).
- The client **C** is on a CERN Wireless network connected at 10 Mbits/s with a network latency of 2 milliseconds (Mac Intel Coreduo 2Ghz).
- The client **D** is in Orsay (LAN 100 Mbits/s) connected to CERN via a WAN with a bandwith of 1 Gbits/s and a network latency of 11 milliseconds (P IV 3 Ghz).
- The client **E** is in Amsterdam (LAN 100 Mbits/s) connected to CERN via a WAN with a bandwith of 10 Gbits/s and a network latency of 22 milliseconds (AMD64 280).
- The client **F** is connected via ADSL with a bandwith of 8Mbits/s and a latency of 70 milliseconds (Mac Intel Coreduo 2Ghz).
- The client G is connected via a 10Gbits/s to a CERN machine via Caltech latency 240 ms.
- The times reported in the table are realtime seconds

client	latency(ms)	cachesize=0	cachesize=64KB	cachesize=10MB
A	0.0	3.4	3.4	3.4
B	0.3	22.0	6.0	4.0
C	2.0	11.6	5.6	4.9
D	11.0	124.7	12.3	9.0
E	22.0	230.9	11.7	8.4
F	72.0	743.7	48.3	28.0
G	240.0	>1800	125.4	9.9

One query to a 280 MB Tree I/O = 6.6 MB

Local caches and proxies

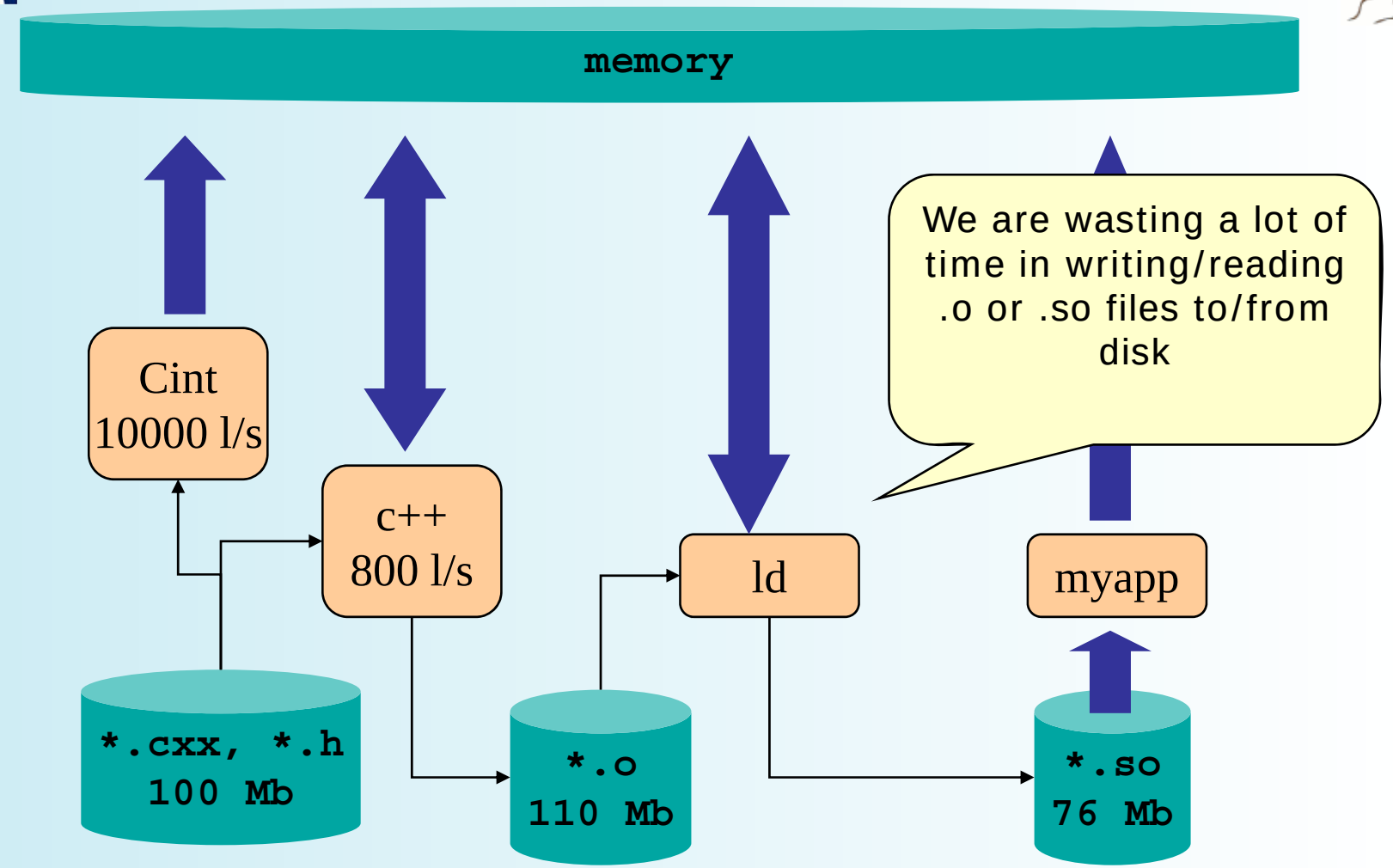
- A local cache must be provided to:
- Cache parts of the remote files (sources or libs)
- Cache compilers/linkers output
- Cache must be able to support multiple versions of the sources.
- The system must be able to discover the info on the nearest and cheapest proxy (source or may be the result of a compilation for a same architecture)



Improve compiler & linker interfaces

- A large fraction of the compiler and linker real time is spent in pure I/O operations.
- We are currently looking at the internals of gcc/ld to understand how one could bypass useless memory->disk->memory operations.
- This could become important on multi-core cpus where I/O and networking could be one of the bottlenecks.

Faster ACLIC



Minimize size of executable module

- One of the main efforts in the past few months.
- We will continue this work.
- It would be nice if we could reach a maximum virtual size below 50 Mbytes and 10 Mbytes real size with a web browser including
  - The browser itself
  - A subset of the ROOT GUI
  - The ROOT Core
- For comparison, on the MAC, Safari uses 400 Mbytes of virtual memory and 30 Mbytes real memory

Extensions to C++

- One could take advantage of the work on BOOT to implement some extensions to the language (eg the USE statement, FOREACH in a collection,..).
- We have experience in code generators (rootcint, tree proxies, makeclass, etc). Of course this generator must be able to generate valid and portable C++ code behind the scene.

Develop our web browser ?

- At a first glance, the question looks obscene.
- Many man years have been spent in existing browsers.
- However we see emerging a few open source projects like xclass (we started from this project for the GUI) or tkhtml (looks very promising).
- A browser has the following main components
  - Html parser (this is complex)
  - Graphics renderer (fonts, menus, images,..)
  - Network layer
- Valeriy Onuchin has a prototype based on xclass and the ROOT graphics layer (TVirtualX and TASImage). A move to tkhtml will give a CSS2 compliant browser!

Valeriy Browser prototype

able to parse very complex html

Rendering by standard ROOT graphics

TVirtualX, TASImage

Installing BOOT itself

- Like the ROOT binary modules today, BOOT will be downloadable from the ROOT web site (or proxies).
- BOOT should support the automatic update feature found in all OS today.

Status

- The BOOT project launched one year ago is a driven force with a gradual implementation in the coming releases.
- Many of the necessary requirements have already been implemented or will come shortly in the coming releases.
- We are very interested by your feedback at this point