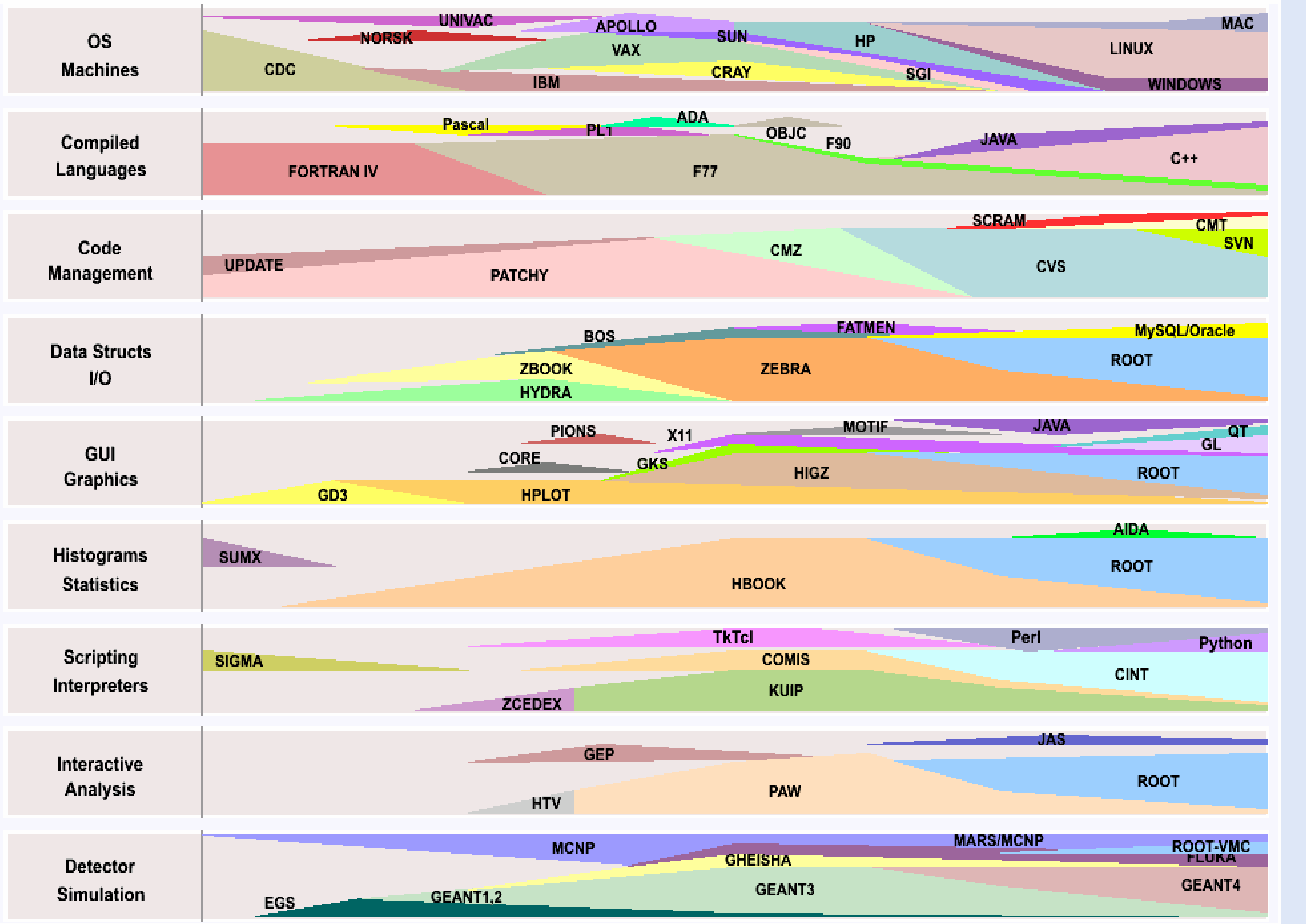
Software challenges for the next decade

DESY: October 29 2007

René Brun
CERN

My mini crystal ball

- Concentrate on HEP software only
- I am biased by the development of large software frameworks or libraries and their use in large experiments.
- Because of inertia or time to develop, there are things easy to predict.
- Technology can come with good or/and bad surprises. See weather Forecast


OS Machines
UNIVAC
NORSK
APOLLO
SUN
HP
MAC
VAX
LINUX
CDC
CRAY
SGI
IBM
WINDOWS
Compiled Languages
Pascal
PL1
ADA
OBJC
F90
JAVA
C++
FORTRAN IV
F77
Code Management
SCRAM
CMT
SVN
CMZ
UPDATE
CVS
PATCHY
Data Structs I/O
BOS
FATMEN
MySQL/Oracle
ROOT
ZBOOK
ZEBRA
HYDRA
GUI Graphics
PIONS
X11
MOTIF
JAVA
QT
GL
CORE
GKS
HIGZ
ROOT
GD3
HPLOT
Histograms Statistics
AIDA
SUMX
ROOT
HBOOK
Scripting Interpreters
TkTcl
Perl
Python
SIGMA
COMIS
CINT
KUIP
ZCEDEX
Interactive Analysis
JAS
GEP
ROOT
PAW
HTV
Detector Simulation
MARS/MCNP
MCNP
ROOT-VMC
GHEISHA
FLUKA
GEANT4
GEANT3
EGS
GEANT1,2
1970
1975
1980
1985
1990
1995
2000
2005
2010

Time to develop: Main message

- Big latency in the process to collect requirements, make a design, implementation, release and effective use.
- It takes between 5 and 10 years to develop a large system like ROOT or Geant4.
- It takes a few years for users to get familiar with the systems.
- It takes a few years for systems to become de facto standards.
- For example, LHC exps are discovering now the advantages of split-mode I/O designed in 1995 and promoted in 1997.
- This trend will likely continue will the large collaborations.
- This has of course positive and negative aspects. Users like stability, BUT competition is mandatory

The crystal ball in 1987

- Fortran 90X seems the obvious way to go
- OSI protocols to replace TCP/IP
- Processors: Vector or MPP machines
- PAW,Geant3,Bos,Zebra: Adapt them to F90X
- Methodoly trend: Entity Relationship Model
- Parallelism: vectorization or MPP (SIMD and MIMD)

- BUT hard to anticipate that
  – The WEB will come less than 3 years later
  – The 1993/1994 revolution for languages and projects
  – The rapid grow in CPU power starting in 1994 (Pentium)

Situation in 1997

- LHC projects moving to C++
- Several projects proposing to use Java
- Huge effort with OODBMS (ie Objectivity)
- Investigate Commercial tools for data analysis
- ROOT development not encouraged
- Vast majority of users very sceptic.

- RAM <256 MB
- Program Size < 32 MB
- <500 KLOcs
- libs < 10
- static linking
- HSM: tape->Disk pool <1 TByte
- Network 2MB/s

The crystal ball in 1997

- C++ now, Java in 2000
- Future is OODBMS (ie Objectivity)
- Central Event store accessed through the net
- Commercial tools for data analysis

- But fortunately a few people did not believe in this direction :☺
- First signs of problems with Babar
- FNAL RUN2 votes for ROOT in 1998
- GRID: an unknown word in 1997 :☺

Situation in 2007

- It took far more time than expected to move people to C++ and the new frameworks.
- ROOT de facto standard for I/O and interactive analysis.
- The GRID:
- Experiment frameworks are monsters

Software Hierarchy

Layer	Size
End user Analysis software	0.1 MLOC
Experiment Software	2 MLOC
Frameworks like ROOT, Geant4	2 MLOC
OS & compilers	20 MLOC
Hardware	
Networking	

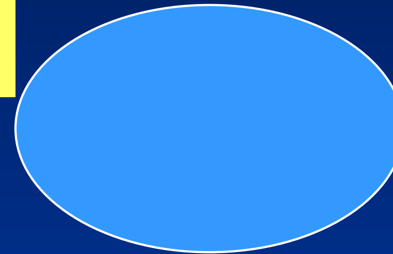
Challenge 0
Usability: Making things SIMPLER

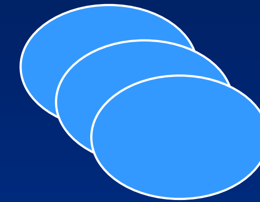
- Guru view vs user view
- A normal user has to learn too many things before being able to do something useful.
- LHC frameworks becoming monsters
- fighting to work on 64 bits with <2 GBytes
- take for ever to start because too much code linked (shared libs with too many dependencies)
- fat classes vs too many classes
- It takes time to restructure large systems to take advantage of plug-in managers.

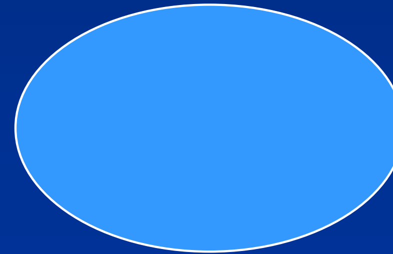
Challenge 1
Problem decomposition

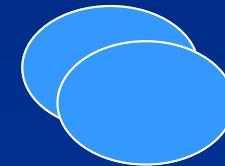
Will have to deal with many shared libs

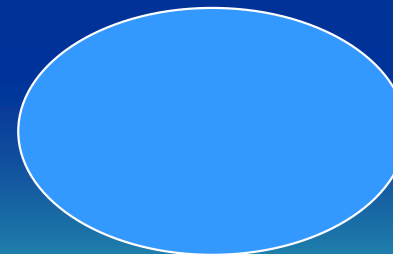
Only a small fraction of code used

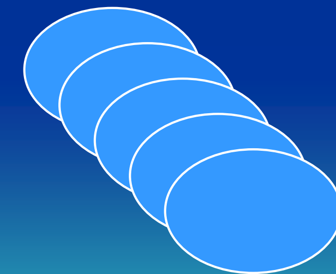












Some Facts

10 shared libs

200 classes

ROOT In 1995

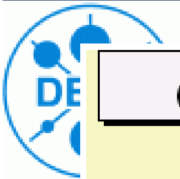
PAW model

100 shared libs

1800 classes

ROOT In 2007

Plug-in manager



code used in a batch use case

Shared lib size in bytes

10^7

10^6

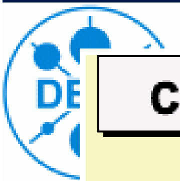
10^5

- shared lib size
- fraction of classes used
- fraction of methods used

core, cint, roofit, gui, mathmore, asimage, geom, xrootd, hist, matrix, graf, tree, gl, treeplayer, proof, ged, table, mathcore, gpad, g3d, treeviewer, qt, reflex, minuit2, histpainter, minuit, physics, eg, xml, postscript, x11, quadp, guibuilder, eompainter, pyroot, html, thread, vmc, mlp, venus, x3d, foam, rint, hbook, pythia6, xmlparser, fumili, cintex, ldap, netx, rfio, mysql, pgsql, dcache, clarens, splot, krb5auth, ruby, x11ttf, newdelete, gedold

```
Libs used                 : 4/86
Classes loaded            : 586/1459
Classes used (CU)         : 51
Methods all libs          : 46438
Methods loaded            : 19550
Methods in CU             : 2666
Methods used in CU        : 325
Code in all libs          : 76 Mb
Code loaded               : 7.1 Mb
Code Really Used (CRU)    : 0.7 Mb
Time to compile CRU       : 17 s
```

Fraction of ROOT code really used in a batch job



code used in a graphics use case

shared lib size
fraction of classes used
fraction of methods used

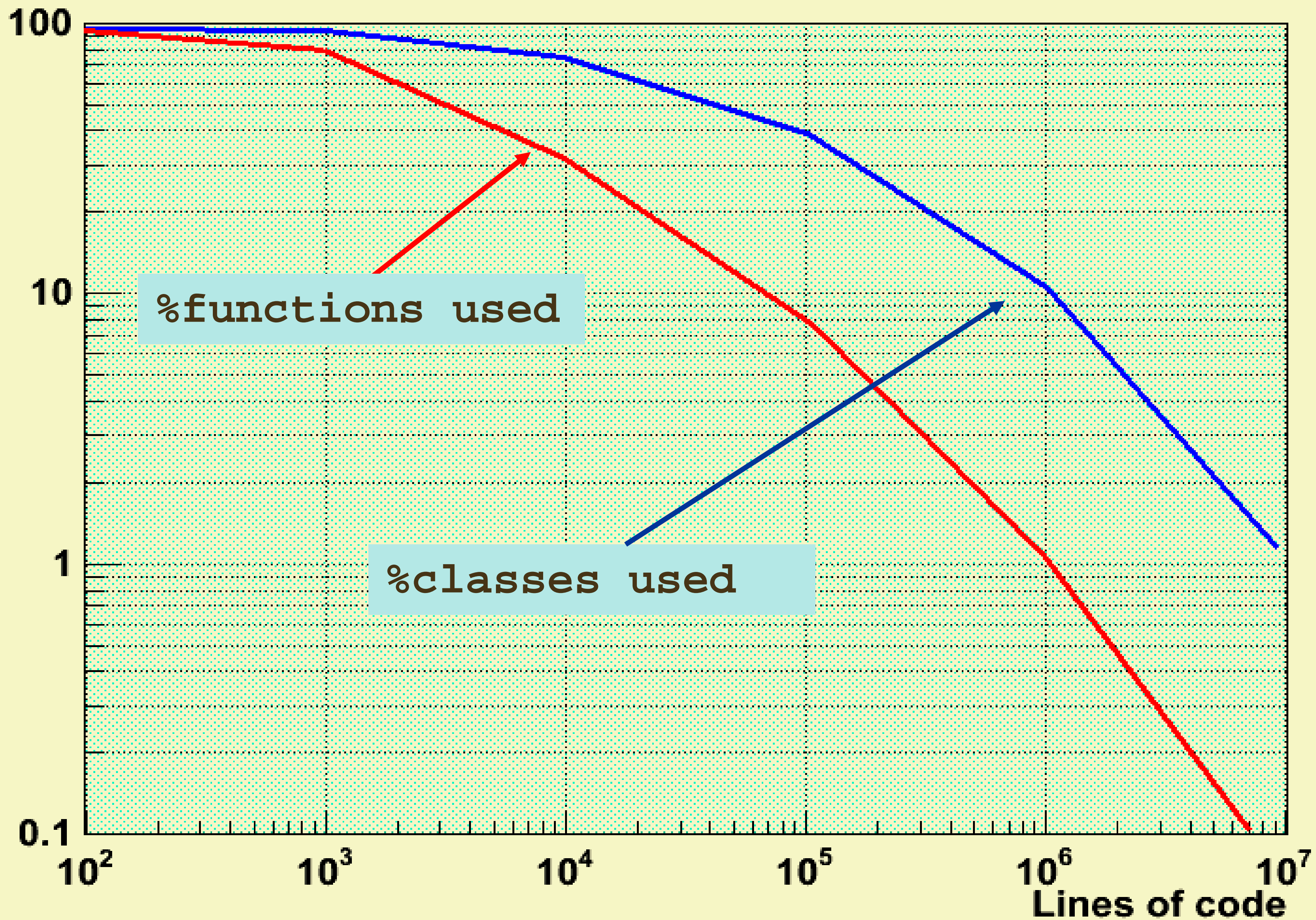
10^7
10^6
10^5

core cint roofit gui mathmore asimage geom xrootd hist matrix graf tree gl treeplayer proof ged table mathcore gpad g3d treeviewer qt reflex minuit2 histpainter minuit physics eg xml postscript x11 quadp guibuilder eompainter pyroot html thread vmc mlp venus x3d foam rint hbook pythia6 xmlparser fumili cintex ldap netx rfio mysql pgsql dcache clarens splot krb5auth ruby x11ttf newdelete gedold

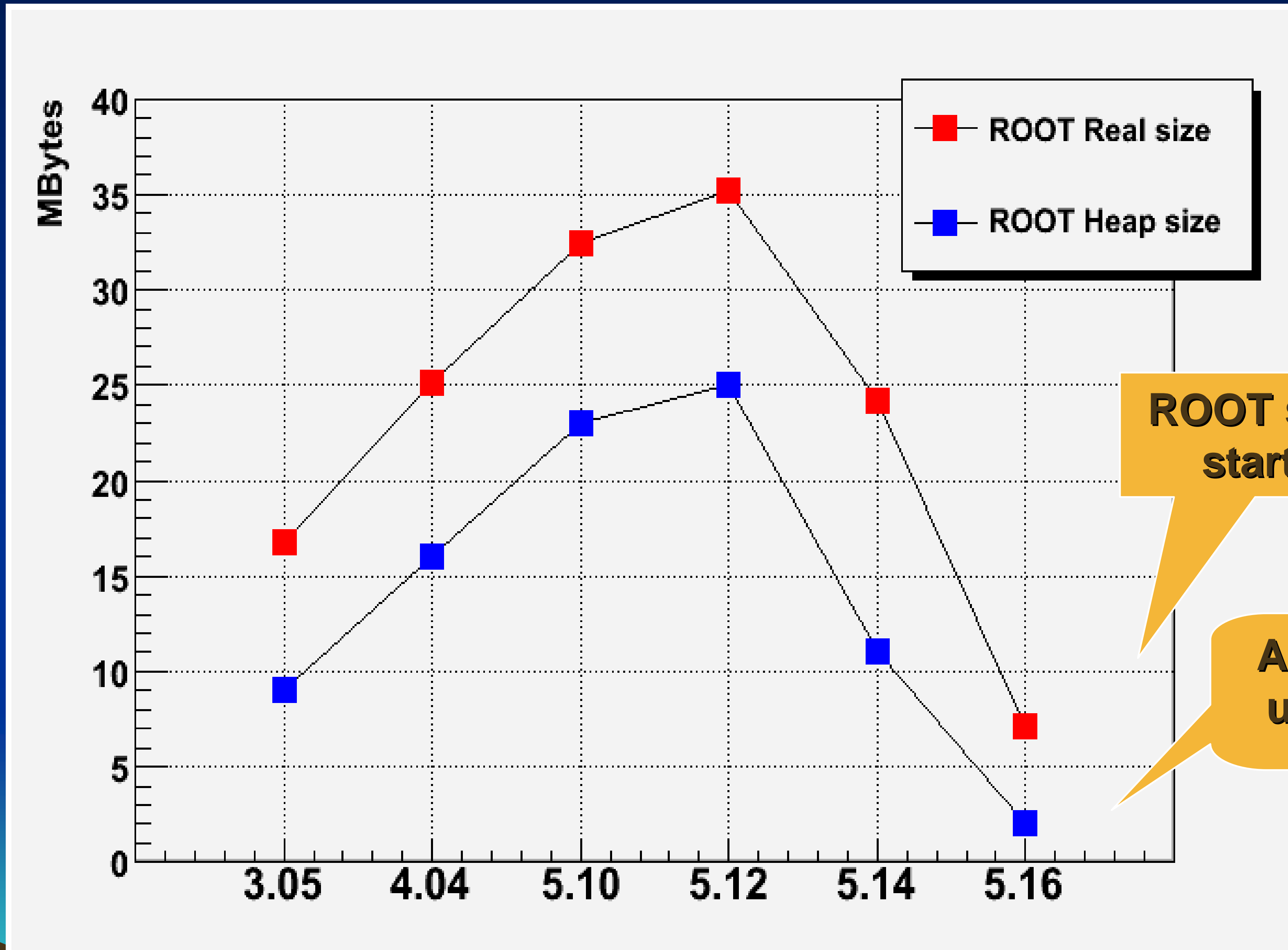
```
Libs used                 : 14/86
Classes loaded            : 865/1459
Classes used (CU)         : 142
Methods all libs          : 46438
Methods loaded            : 26996
Methods in CU             : 6676
Methods used in CU        : 740
Code in all libs          : 76 Mb
Code loaded               : 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

Fraction of code really used in one program



Large Heap Size Reduction



ROOT size at start-up

Also speed-up start-up time

Challenge 2
Hardware will force parallelism

- Multi-Core (2-8)
- Many-Core (32-256)
- Mixture CPU + GPU-like
- Virtualization
- May be a new technology?
- Parallelism: a must

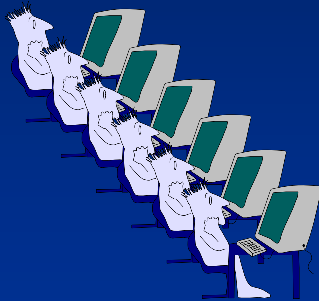
Challenge 3
Design for Parallelism

- The GRID is a parallel engine. However you will not use the GRID software on your 32-core laptop.
- Minimize globals and make tasks as independent as possible.
- Be thread-safe and better thread-capable
- Think Top->Down and Bottom->Up

Parallelism: Where ?

Multi-Core CPU laptop/desktop

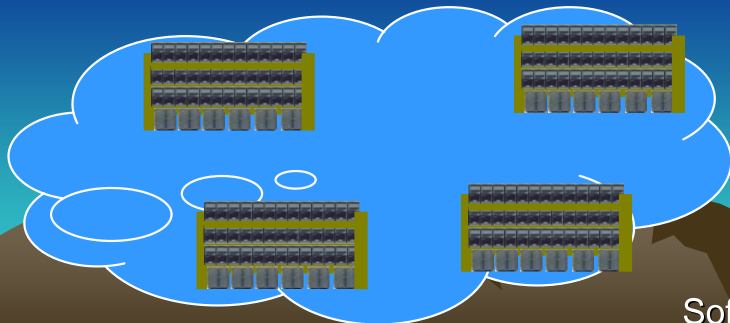
2(2007) 32(2012?)



Network of desktops

Local Cluster

with multi-core CPUs



GRID(s)

Challenge 4
Design for Client-Server

- The majority of today's applications are client-server (xrootd, Dcache, sql, etc).
- This trend will increase.
- Be able to stream objects or objects collections.
- Server logic robust against client changes.
- Server able to execute dynamic plug-ins.
- Must be robust against client or network crash

Challenge 5
Sophisticated Plug-in Managers

- When using a large software base distributed in hundred of shared libs, it is essential to discover automatically where to find a class.
- The interpreters must be able to autoload the corresponding libraries

Challenge 6
The Language Reflexion System

- Develop a robust dictionary system that can be migrated smoothly to the reflexion system to be introduced in C++ in a few years.
- Meanwhile reduce the size of dictionaries by doing more things at run time.
- Replace generated code by objects stored in ROOT files.
- Direct calls to compiled code from the interpreter instead of function stubs. This is compiler dependent (mangling/de-mangling symbols).

Problem with Dictionaries

Today cint/reflex dictionaries are machine dependent. They represent a very substantial fraction of the total code

We are now working to reduce this size by at least a factor 3!

	*.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%

Challenge 7
Opportunistic Use of Interpreters

- Use interpreted code only for:
  - External and thin layer (task organizer)
  - Slots execution in GUI signal/slots
  - Dynamic GUI builder in programs like event displays.
- Instead optimize the compiler/linker interface (eg TACLIC) to have
  - Very fast compilation/linking when performance is not an issue
  - Slower compilation but faster execution for the key algorithms
- ie use ONE single language for 99% of your code and the interpreter of your choice for the layer between shell programming and program orchestration.

Challenge 8
LAN and WAN I/O caches

- Must be able to work very efficiently across fat pipes but with high latencies.
- Must be able to cache portions or full files on a local cache.
- This requires changes in data servers (Castor, Dcache, xrootd). These tools will have to interoperate.
- The ROOT file info must be given to these systems for optimum performance. See TTreeCache improvements.

Disk cache improvements with high latency networks

- 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 **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

One query to a 280 MB Tree I/O = 16.6 MB

client	latency(ms)	cachesize=0	cachesize=64KB	cachesize=10MB
A	0.0	3.4	3.4	3.4
F	72.0	743.7	48.3	28.0
G	240.0	>1800s	125.4s	9.9s

We expect to reach 4.5 s

I/O: More

- -Efficient access via a LAN AND WAN
- -Caching
- -Better schema evolution
- -More support for complex event models
- -zip/unzip improvements (separate threads)
- -More work with SQL data bases

Challenge 9
Code Performance

- HEP code does not exploit hardware (see S.Jarp talk at CHEP)
- Large data structures spread over >100 Megabytes
- templated code pitfall
  - STL code duplication
  - good perf improvement when testing with a toy.
  - disaster when running real programs.
- std::string passed by value
- abuse of new/delete for small objects or stack objects
- linear searches vs hash tables or binary search
- abuse of inheritance hierarchy
- code with no vectors -> do not use the pipeline

Compilation Time

LHC software

	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

Challenge 10

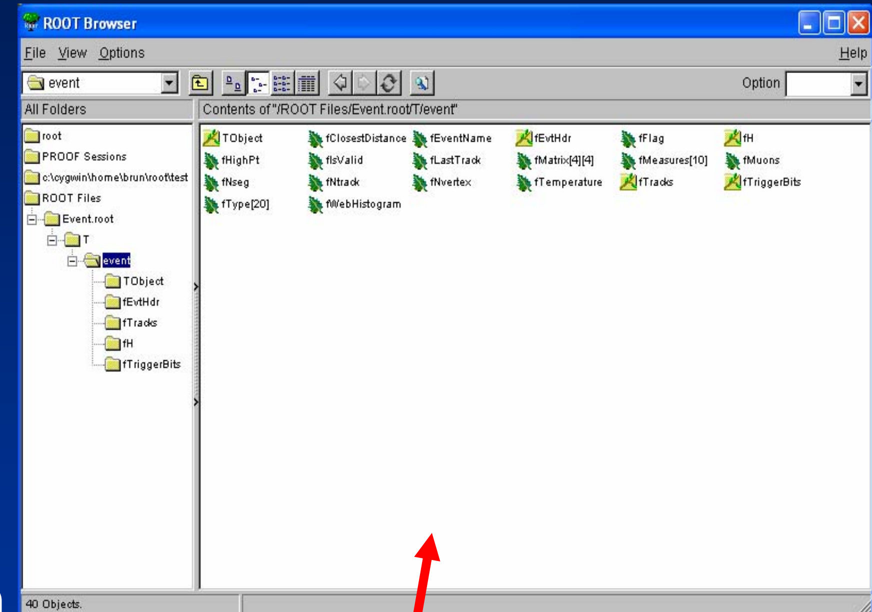
Towards Task-oriented programming

Challenge 11
Customizable and Dynamic GUIs

- From a standard browser (eg ROOT TBrowser) on must be able to include user-defined GUIs.
- The GUIs should not require any pre-processor.
- They can be executed/loaded/modified in the same session.

Browser Improvements

- The browser (TBrowser and derivatives) is an essential component (from beginners to advanced applications).
- It is currently restricted to the browsing of ROOT files or Trees.
- We are extending TBrowser such that it could be the central interface and the manager for any GUI application (editors, web browsers, event displays, etc).



Old/current browser

Hist Browser + stdin/stdout

ROOT Browser

Framework File Edit View Options Inspect Classes Help

Files | Classes

spectrum
splot
sql
thread
tree
unuran
xml
MyTasks.cxx
README
bb.txt
benchmarks.C
demos.C
demoshelp.C
fillrandom.root
form1;1
sqroot;1
h1f;1
gallery.root
geant3tasks.C
hsimple.C
hsimple.root
hsimple.svg
htmlex.C
regexp.C
rootalias.C
rootenv.C
rootlogoff.C
rootlogon.C
rootmarks.C
tasks.C
unix
unuran

All Files (*.*)

Editor | HTML | Canvas

Test random numbers

h1f	
Entries	10000
Mean	3.646
RMS	1.84

Command

Command (local):

```
root [1] .ls
TFile**         fillrandom.root
 TFile*         fillrandom.root
  KEY: TFormula form1;1 abs(sin(x)/x)
  KEY: TF1      sqroot;1        x*gaus(0)+[3]*form1
  KEY: TH1F     h1f;1   Test random numbers
```

TGhtml web browser plug-in

URL

You can browse a root file

You can execute a script

ROOT Browser

Framework File Favorites Tools Help

Files Classes | Editor HTML Canvas

http://root.cern.ch/files/

Index of /files

Name	Last modified	Size	Description
Parent Directory		-	
H1display.root	14-May-2007 22:05	8.5K	
aleph.root			
aleph_all.C	20-Nov-2004 18:20	609	
alice.root	03-Apr-2006 18:31	1.2M	
alice_all.C	26-Nov-2004 12:11	1.4K	
alice_itsd.C	09-Nov-2004 16:16	461	
alice_itsv.C	21-Nov-2004 11:47	931	
ams.C			
ams_all.C	17-Nov-2004 09:51	173	
archive.zip	08-Jun-2006 15:22	563K	
atlas.root	05-Apr-2005 12:04	4.0M	

Command

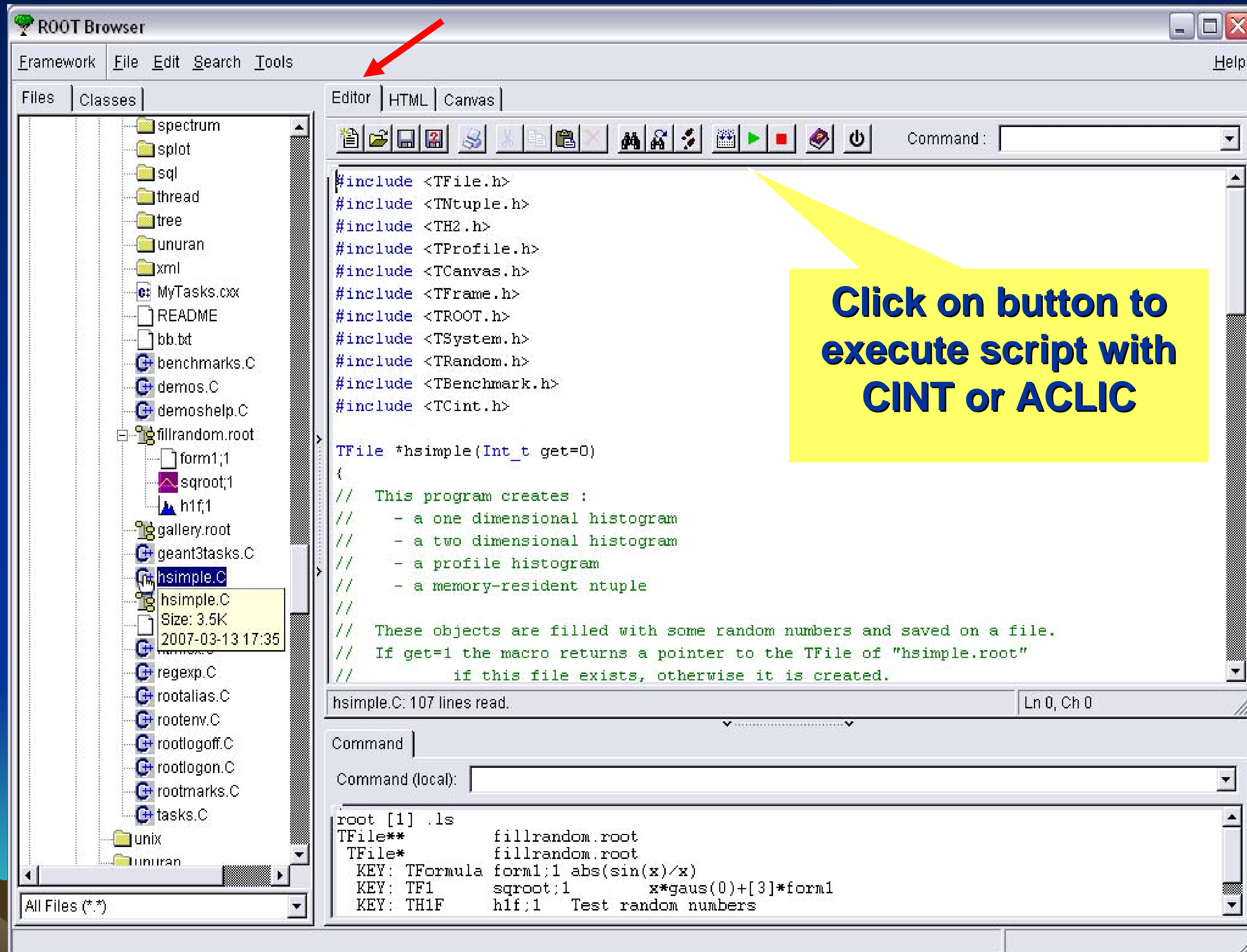
Command (local):

```
root [1] .ls
TFile**         fillrandom.root
 TFile*         fillrandom.root
  KEY: TFormula form1;1 abs(sin(x)/x)
  KEY: TF1      sqroot;1        x*gaus(0)+[3]*form1
  KEY: TH1F     h1f;1   Test random numbers
```

spectrum, splot, sql, thread, tree, unuran, xml, MyTasks.cxx, README, bb.txt, benchmarks.C, demos.C, demoshelp.C, fillrandom.root, form1;1, sqroot;1, h1f;1, gallery.root, geant3tasks.C, hsimple.C, hsimple.root, hsimple.svg, htmlex.C, regexp.C, rootalias.C, rootenv.C, rootlogoff.C, rootlogon.C, rootmarks.C, tasks.C, unix, unuran

All Files (*.*)

Macro Manager/Editor plug-in



GL Viewer plug-in

Alice event display prototype using the new browser

Reve Main Window

Reve File Camera Help

Reve | Files | Macros

GL Viewer | Editor | Viewer Pepe

- Viewers
  - GL-One
    - SI - Geometry scene
    - SI - Event scene
    - SI - Projected Event
  - Viewer Pepe
- Scenes
  - Geometry scene
    - Gentle Geometry
  - Event scene
  - Projected Event
- Event 0
  - ITS Clusters
  - TPC Clusters
  - ESD Tracks
  - TRD Hits "
- RhoZ 0.010000
  - NLT Event 0
    - NLT ITS Clusters
    - NLT TPC Clusters
    - NLT ESD Tracks

Style

Name

RhoZ 0.010000::Reve::NLTProjector

RenderElement

RnrSelf RnrChildren

Axis

StepMode Position

SplitLevel 1

NLTProjection

Type RhoZ

Distortion: 10.00

FixedR: 300.0

Command

Command (local):

Challenge 12
Executing Anywhere from Anywhere

- One should be able to start an application from any web browser.
- The local UI and GUI can execute transparently on a remote process.
- The resulting objects are streamed to the local session for fast visualization.
- Prototype in latest ROOT using ssh technology.

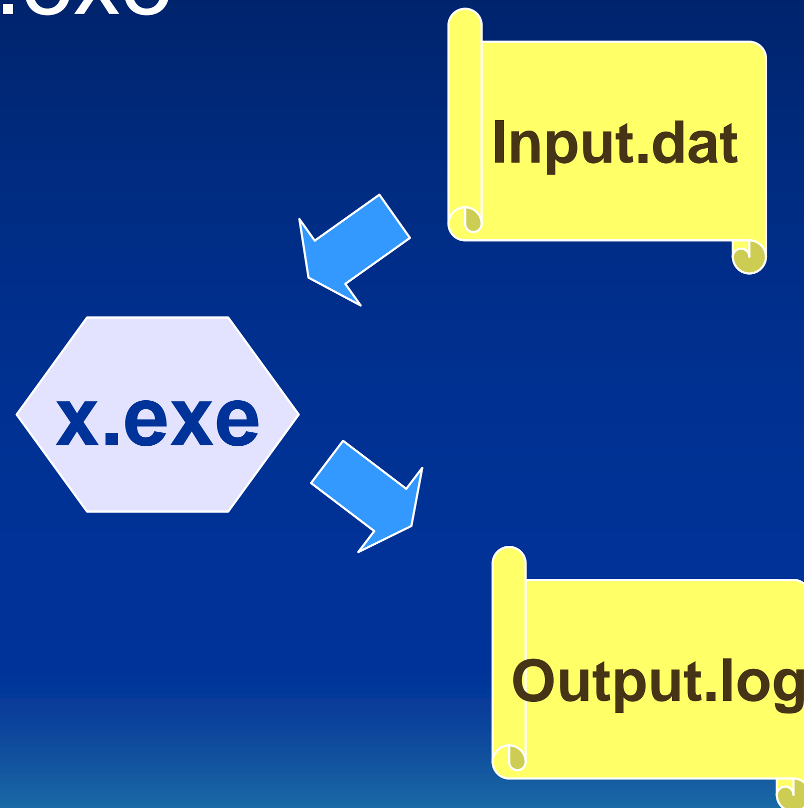
```
root > .R lxplus.cern.ch
lxplus > .x doSomething.C
lxplus > .R
root > //edit the local canvas
```

Challenge 13
Evolution of the Execution Model

- From stand alone modules
- To shared libs
- To plug-in managers
- To distributed computing
- To distributed and parallel computing

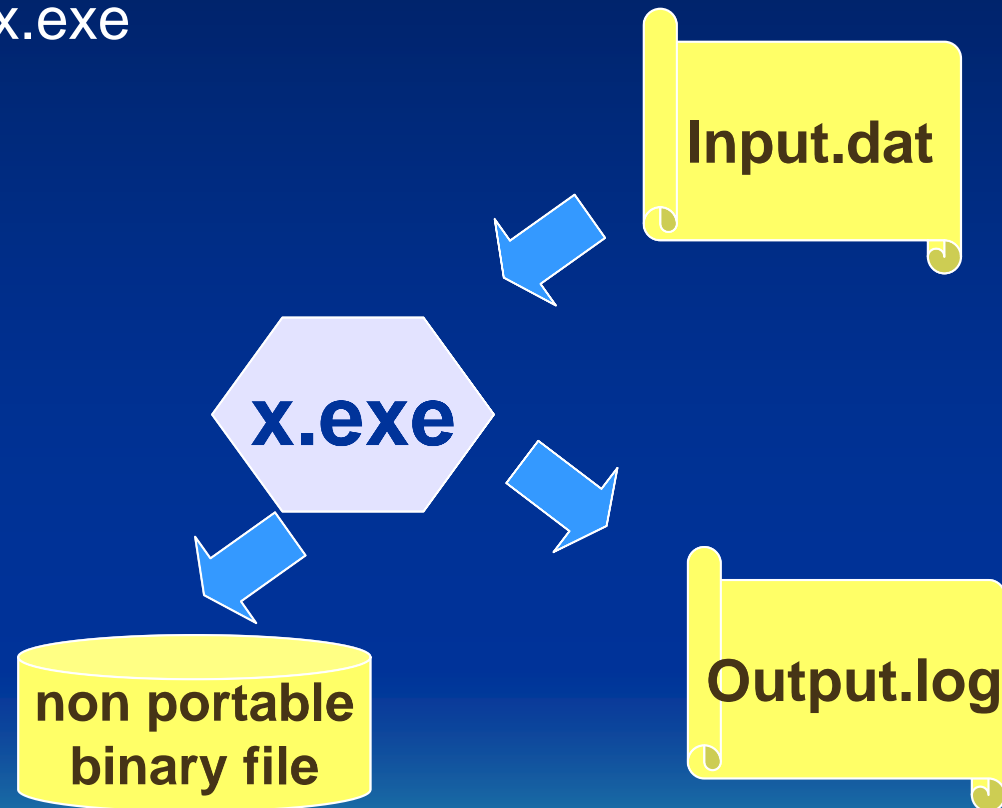
Executable module in 1967

- x.f -> x.o -> x.exe



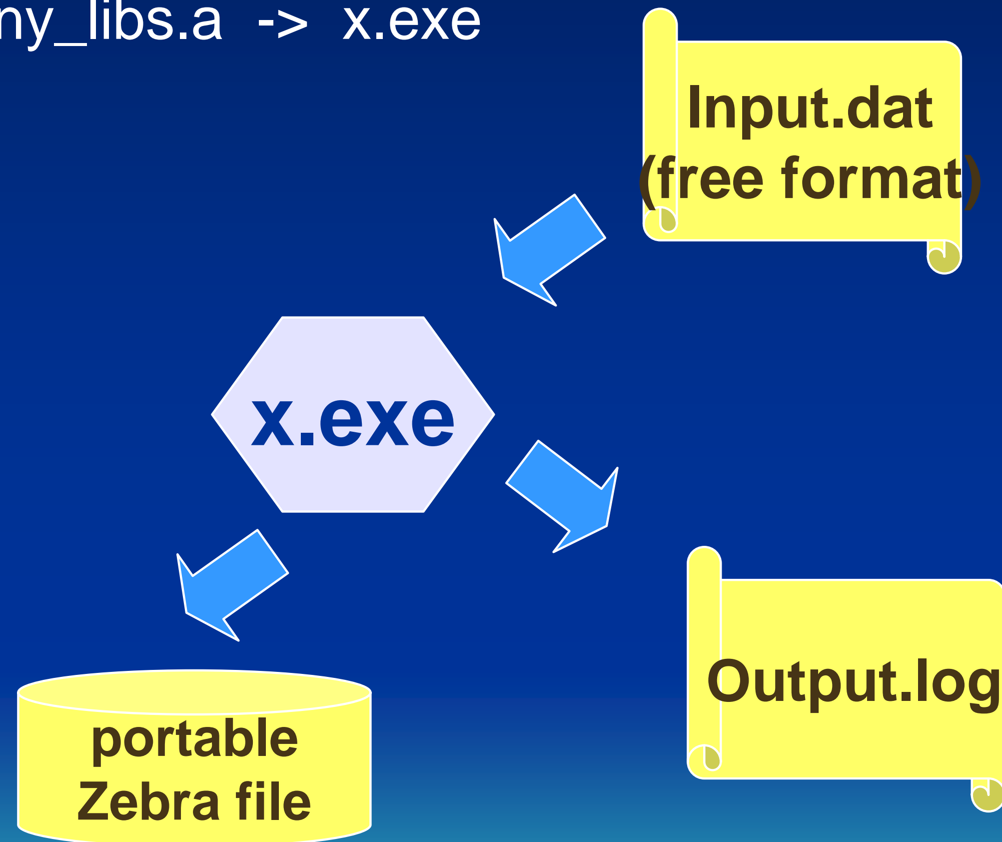
Executable module in 1977

- x.f -> x.o
- x.o + libs.a -> x.exe



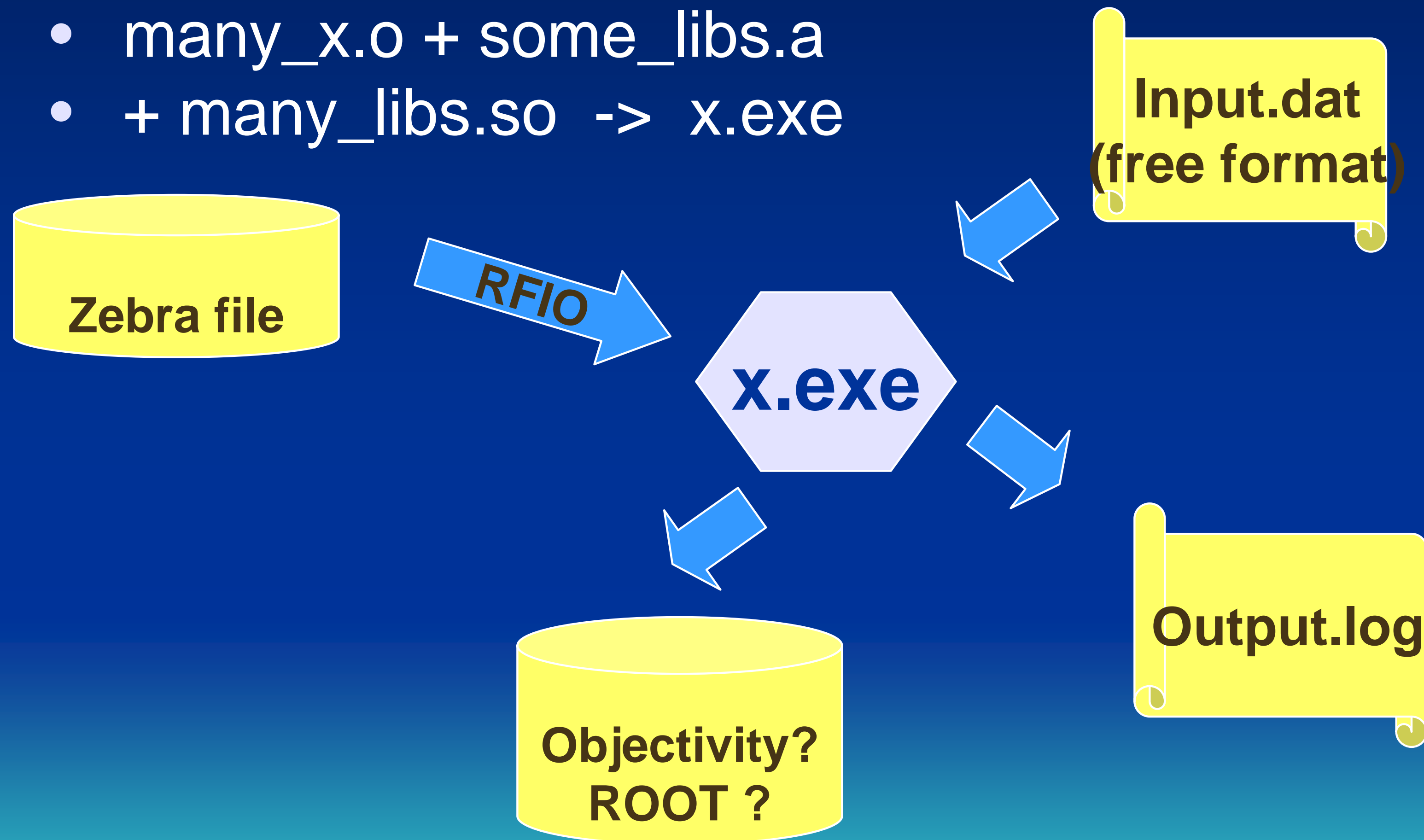
Executable module in 1987

- many_x.f -> many_x.o
- many_x.o + many_libs.a -> x.exe



Executable module in 1997

- many_x.f -> many_x.o
- many_x.o + some_libs.a
- + many_libs.so -> x.exe



Executable module in 2007

Executable module in 2017 ?

Local shared libs dynamically
Compiled/loaded/unloaded
from a source URL

Challenge 14

Data Analysis on the GRID

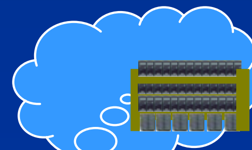
5,000 physicists

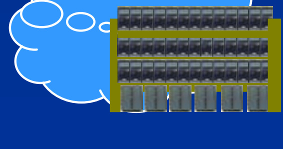
in 1000 locations

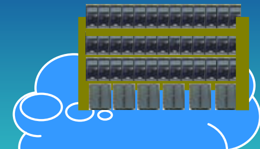
100,000 computers

in 1000 locations

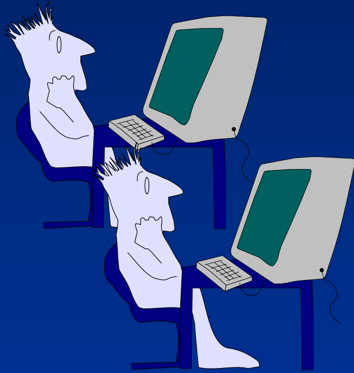
LAN



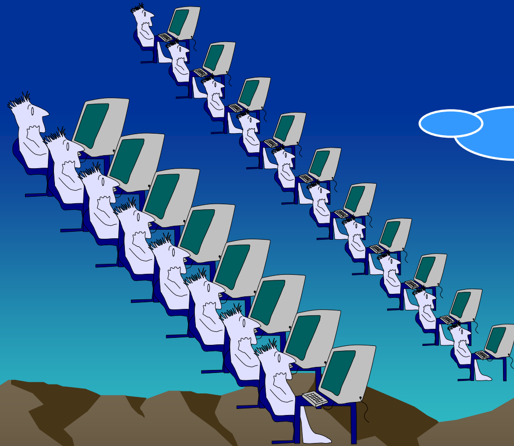




GRID: Users profile



Few big users submitting many long jobs (Monte Carlo, reconstruction)
They want to run many jobs in one month



Many users submitting many short jobs (physics analysis)
They want to run many jobs in one hour or less

Big but few Users

- Monte Carlo jobs (one hour → one day)
  - Each job generates one file (1 GigaByte)
- Reconstruction job (10 minutes -> one hour)
  - Input from the MC job or copied from a storage centre
  - Output (< input) is staged back to a storage centre
- Success rate (90%). If the job fails you resubmit it.
- For several years, GRID projects focused effort on big users only.

Small but many Users

- Scenario 1: submit one batch job to the GRID. It runs somewhere with varying response times.
- Scenario 2: Use a splitter to submit many batch jobs to process many data sets (eg CRAB, Ganga, Alien). Output data sets are merged automatically. Success rate < 90%. You see the final results only when the last job has been received and all results merged.
- Scenario 3: Use PROOF (automatic splitter and merger). Success rate close to 100%. You can see intermediate feedback objects like histograms. You run from an interactive ROOT session.

GRID & Parallelism: 1

- The user application splits the problem in N subtasks. Each task is submitted to a GRID node (minimal input, minimal output).
- The GRID task can run synchronously or asynchronously. If the task fails or time-out, it can be resubmitted to another node.
- One of the first and simplest use of the GRID, but not many applications in HEP.
- Examples are SETI, BOINC

GRID & Parallelism: 2

- The typical case of Monte Carlo or reconstruction in HEP.
- It requires massive data transfers between the main computing centres.
- This activity has concentrated so far a very large fraction of the GRID projects and budgets.
- It has been an essential step to foster coordination between hundreds of sites, improve international network bandwidths and robustness.

GRID & Parallelism: 3

- Distributed data analysis will be a major challenge for the coming experiments.
- This is the area with thousands of people running many different styles of queries, most of the time in a chaotic way.
- The main challenges:
  - Access to millions of data sets (eg 500 TeraBytes)
  - Best match between execution and data location
  - Distributing/compiling/linking users code(a few thousand lines) with experiment large libraries (a few million lines of code).
  - Simplicity of use
  - Real time response
  - Robustness.

GRID & Parallelism: 3a

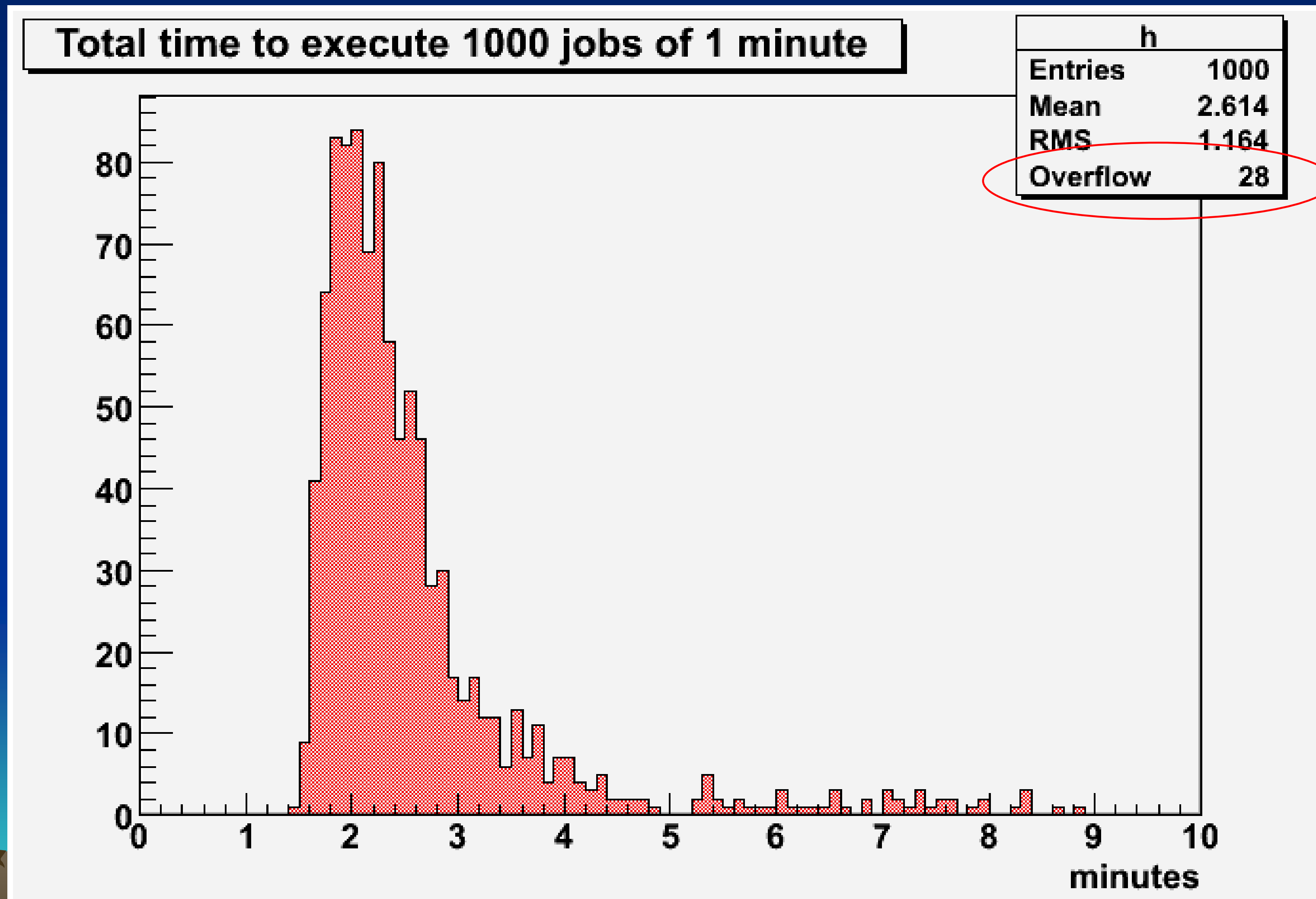
- Currently 2 different & competing directions for distributed data analysis.
  - Batch solution using the existing GRID infrastructure for Monte Carlo and reconstruction programs. A front-end program partitions the problem to analyze ND data sets on NP processors.
  - Interactive solution PROOF. Each query is parallelized with an optimum match of execution and data location.

Scenario 1 & 2: PROS

- Job level parallelism. Conventional model. Nothing to change in user program.
  - Initialisation phase
  - Loop on events
  - Termination
- Same job can run on laptop or on a GRID job.

Scenario 1 & 2: CONS(1)

- Long tail in the jobs wall clock time distribution.



Scenario 1 & 2: CONS(2)

- Can only merge output after a time cut.
- More data movement (input & output)
- Cannot have interactive feedback
- Two consecutive queries will produce different results (problem with rare events)
- Will use only one core on a multi core laptop or GRID node.
- Hard to control priorities and quotas.

Scenario 3: PROS

- Predictive response. Event level parallelism. Workers terminate at the same time
- Process moved to data as much as possible, otherwise use network.
- Interactive feedback
- Can view running queries in different ROOT sessions.
- Can take advantage of multi core cpus

Scenario 3: CONS

- Event level parallelism. User code must follow a template: the TSelector API.
- Good for a local area cluster. More difficult to put in place in a GRID collection of local area clusters.
- Interactive schedulers, priority managers must be put in place.
- Debugging a problem slightly more difficult.

Challenge 15
Languages

- C++ clear winner in our field and also other fields
- see, eg a recent compilation at http://www.lextrait.com/vincent/implementations.html
- From simple C++ to complex templated code
- Unlike Java, no reflexion system. This is essential for I/O and interpreters.
- C++2009: better thread support, Aspect-oriented
- C++2014: first reflexion system?

Challenge 16
Software Development Tools

- better integration with Xcode, VisualStudio or like
- fast memory checkers
- faster valgrind
- faster profilers
- Better tools to debug parallel applications
- Code checkers and smell detection
- Better html page generators

Challenge 17
Distributed Code Management

- patchy, cmz -> cvs
- cvs -> svn
- cmt? scram? (managing dependencies)
- automatic project creation from cvs/svn to VisualStudio or Xcode and vice-versa

Challenge 18
Simplification of Software Distribution

- tar files
- source + make
- install from http://source
- install from http://binary proxy
- install on demand via plugin manager, autoloader
- automatic updates
- time to install
- fraction of code used

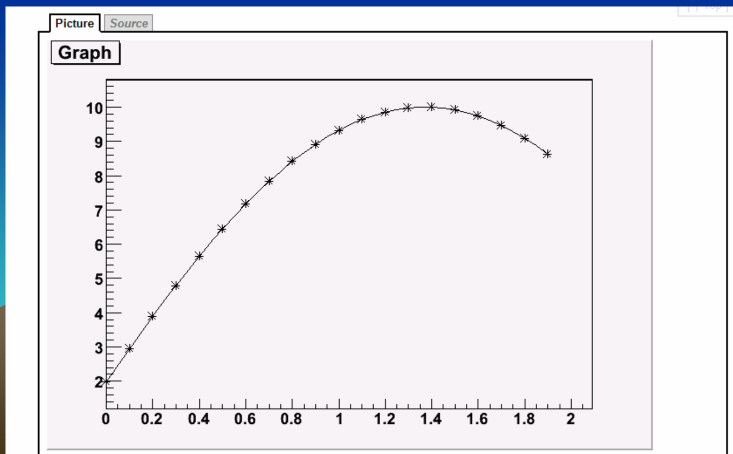
See BOOT
Project
First release
In June 08

Challenge 19
Software Correctness

- -big concern with multi million lines of code
- -validation suite
- -unit test
- -combinatorial test
- -nightly builds (code + validation suite)

Challenge 20
Scalable Software Documentation

- Legacy Doxygen
- Need for something more dynamic, understanding Math, Latex, 2-D and 3-D graphics,interactive tutorials.
- See results of new THtml at:
  - http://root.cern.ch/root/html/TGraph.html



Picture Source

```
{
   TCanvas *c1 = new TCanvas("c1","A Simple Graph Example",200,10,700,500);
   Double_t x[100], y[100];
   Int_t n = 20;
   for (Int_t i=0;i<n;i++) {
     x[i] = i*0.1;
     y[i] = 10*sin(x[i]+0.2);
   }
   gr = new TGraph(n,x,y);
   gr->Draw("AC*");
   return c1;
}
```

Challenge 21
Education

- Training must be a continuous effort
- Core Software guys often desperate with newcomers.
- Software Engineering and discipline required to participate to large international projects is absent in University programs.

Summary

- A large fraction of the software for the next decade already in place or shaping up.
- Long time between design and effective use.
- Core Software requires Open Source and international cooperation to guarantee stability and smooth evolution.
- Parallelism will become a key parameter
- More effort must be invested in software quality, training and education.

Summary-2

- But the MAIN challenge will be to deliver scalable systems:
- Simple to use for beginners with some very basic rules and tools.
- Browsing (understanding) an ever growing dynamic code and data will be a must.

Summary-3

- Building large software systems is like building a large city. One needs to standardize on the communication pipes for input and output and setup a basic set of rules to extend the system or navigate inside.