ROOT – A brief introduction

Survival with ROOT != Survival at ROOT != Survival despite ROOT

*

- Introduction to ROOT
  - What is it ?
  - Why is it good ?
- Using ROOT
  - Command line, batch-mode, root console or terminal
  - Scripting/Interpretation
    - Example script comparing two current-mode D/A converter architectures designed for COMPASS
  - Compilation
    - Compiling a script as a " *.so " shared object library
    - Compiling standalone
    - Application development
    - Example standalone application
- GUI of ROOT
  - Human interaction
  - Creating a GUI
- Survival [with/at/despite] ROOT
  - User's guide (refer once)
  - Referring to:
    - $ROOTSYS/tutorials (refer once per problem)
    - $ROOTSYS/test (refer once per problem)
  - HTML source code documentation (refer continuously)
- External library usage from within ROOT
  - DQM of ALICE experiment @ CERN
    - Simplified DAQ operation
  - Understanding the detector data
    - Accessing and decoding data

What is it ?

Well, it is:

- **A library:** *ROOT is a very **large class library** consisting of specialized smaller libraries.* ROOT libraries can be included in ***user programs*** and be used as external libraries:
  - ***GUI** development library*
  - *Library providing **signal-slot** mechanism*
  - *Libraries encapsulating functions for **data analysis**, etc.*

- **A C/C++ interpreter:** ROOT has CINT -an embedded C ***interpreter***- which allows easy algorithm development. You can execute C/C++ statements just like executing commands on a Linux shell or like writing shell scripts.
  - This way of working generates applications running slower, however:
  - Development is ***faster*** and in case of necessity the scripts can easily be compiled into *.so objects/libraries which allows faster execution

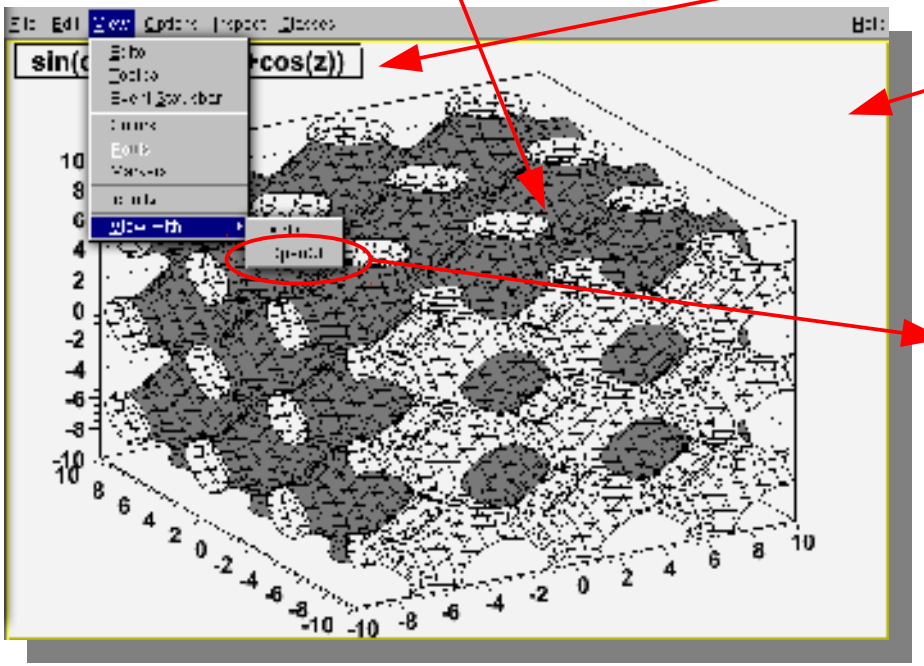
- **A framework:** ROOT provides the ***infrastructure*** needed by the physicists to work
  - Interpreter, histograming and analysis functions, GUI development capability, I/O functions, class libraries, parallel processing and threads, functions specific to sockets and network communication, etc. These are all ***ready to use***, minimizing the effort of development.

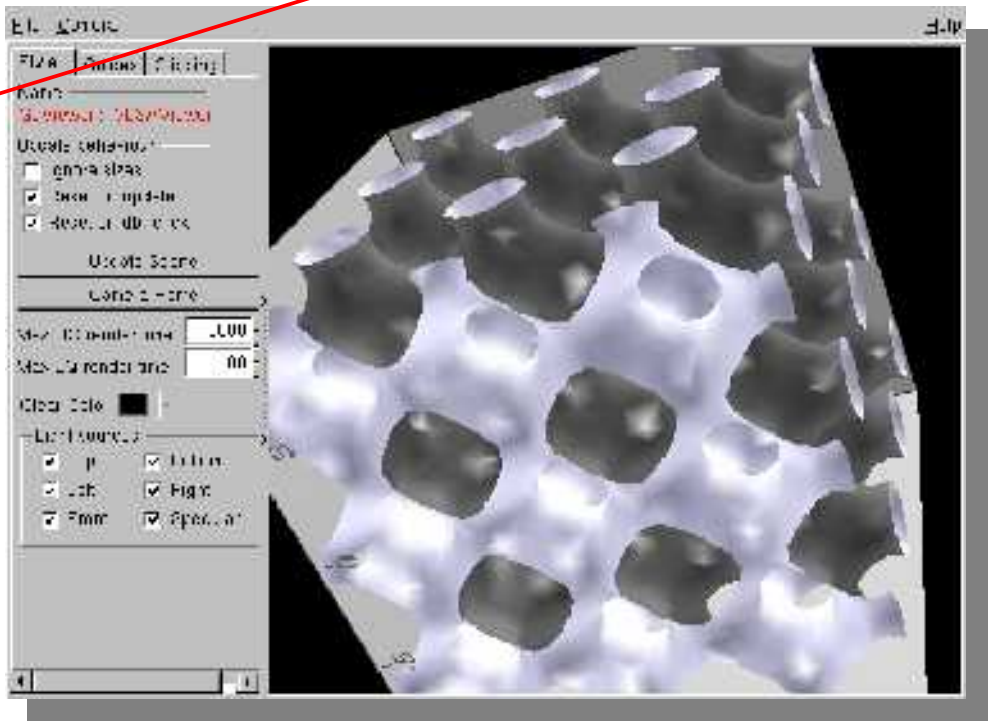
Why is it good ?

Because of the supposed fact that:

- There are advantages of working within a ***framework***, such as the following:
  - No need to write ***many commands*** to achieve a ***specific functionality***
  - High ***reliability*** of developed code due to extensive library usage
  - ***Consistent*** class hierarchy of the developed code
  - ***Flexibility*** of a modular architecture, thus “develop-and-reuse” is easier
  - Physicist can ***focus*** on his/her subject more

```
root [0] TF3 f1("HelloWorld","sin(cos(x)+sin(y)+cos(z))",-10,10,-10,10,-10,10)
root [1] f1->Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [2] HelloWorld->SetTitle("Something else !..")
```

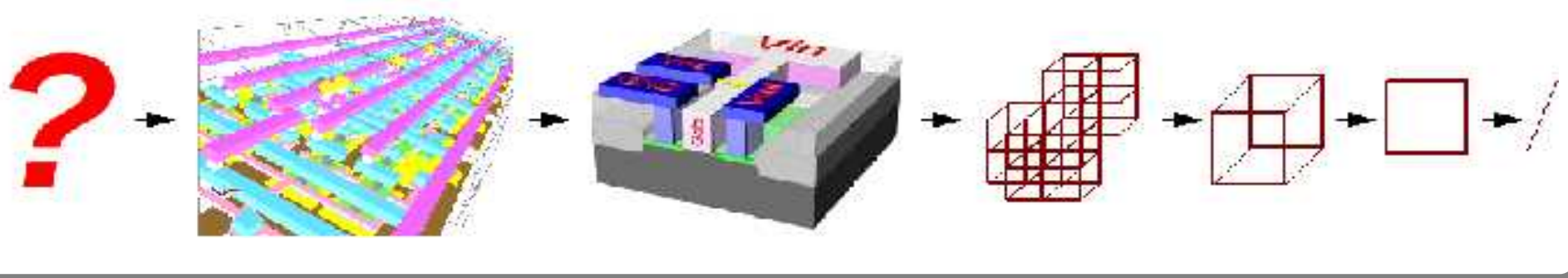




Why is it good ?

Because of the supposed fact that:

- There are advantages of working within an ***object-oriented*** framework:
  - Specific to object-oriented languages, the data are embedded within the class instances, allowing easier ***abstraction***, and leading to more develop-and-***reuse*** of library components
  - Generation of classes out of other classes allows ***further*** development and/or modification of already existing classes
  - ***Hierarchy of classes*** resembles conceptual hierarchy of real-world objects allowing easy abstraction/development



  - Developed code is far from being ***complex***, the data are localized within class instances and are not distributed over many global variables as it is the case for procedural approach (e.g. C or FORTRAN)
  - Removing and/or adding new classes into the developed code does not change the ***architecture*** therefore an architecture can be maintained during development

ROOT – A brief introduction

Survival with ROOT != Survival at ROOT != Survival despite ROOT

*

- Introduction to ROOT
  - What is it ?
  - Why is it good ?
- Using ROOT
  - Command line, batch-mode, root console or terminal
  - Scripting/Interpretation
    - Example script comparing two current-mode D/A converter architectures designed for COMPASS
  - Compilation
    - Compiling a script as a " *.so " shared object library
    - Compiling standalone
    - Application development
    - Example standalone application
- GUI of ROOT
  - Human interaction
  - Creating a GUI
- Survival [with/at/despite] ROOT
  - User's guide (refer once)
  - Referring to:
    - $ROOTSYS/tutorials (refer once per problem)
    - $ROOTSYS/test (refer once per problem)
  - HTML source code documentation (refer continuously)
- External library usage from within ROOT
  - DQM of ALICE experiment @ CERN
    - Simplified DAQ operation
  - Understanding the detector data
    - Accessing and decoding data

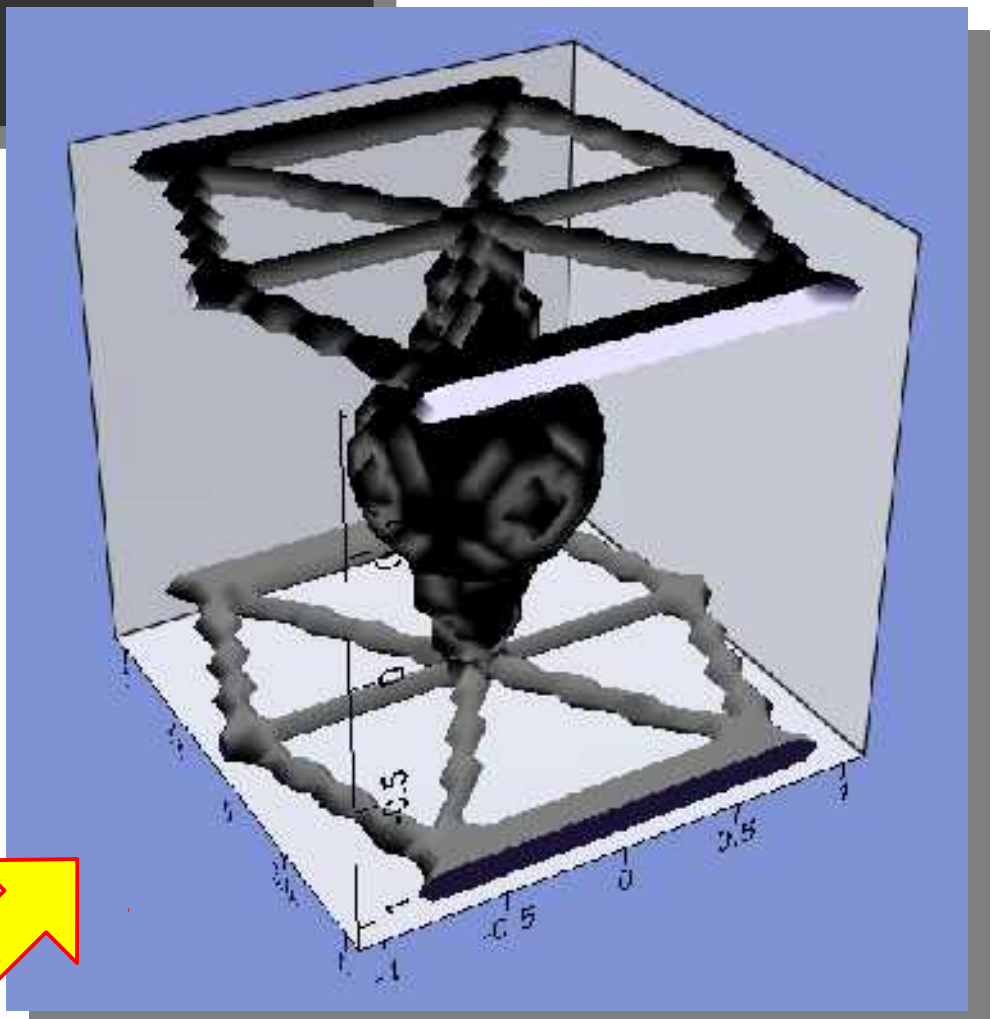
Installation & tie fighter math

Intended usage ?

```
> tar xvfz root_v5.22.00.source.tar.gz
> export ROOTSYS=$HOME/root
> export PATH=$PATH:$ROOTSYS/bin
> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ROOTSYS/lib
> cd root
> ./configure
> make
> sudo make install
> _
```

- Set environment, configure, compile and ROOT is ready to be used.

```
# root -l
> TF3 *tieFighter = new
  TF3("tieFighter","(x^2+y^2+z^2<0.2)+
  ((y^2+z^2<0.08)*(x<0.4)*(x>0))+
  (x^2+4*y^2<(1-TMath::Abs(z))*0.12)+
  ((TMath::Abs(z)<0.95)*(TMath::Abs(z)
  >0.9)*(TMath::Abs(x)
  +TMath::Abs(y)*0.3<1))+
  ((TMath::Abs(z)<1)*(TMath::Abs(z)>0.
  89))*((TMath::Abs(x)<0.7)*(TMath::Ab
  s(y)>0.9)+(TMath::Abs(y)<0.035)+
  (x>y*0.7-0.05)*(x<y*0.7+0.05)+(-
  x>y*0.7-0.05)*(-x<y*0.7+0.05)+
  ((TMath::Abs(x)
  +TMath::Abs(y)*0.3<1.05)*(TMath::Abs
  (x)+TMath::Abs(y)*0.3>0.95)))",-
  1.1,1.1,-1.1,1.1,-1.1,1.1);
> tieFighter->Draw()
```

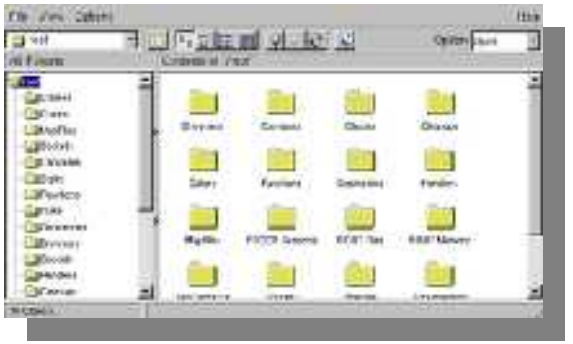


Usage – ROOT Command Line

Invoking commands

- *Invoking C/C++ statements just like invoking shell commands*
  - Almost all C/C++ phrases and all classes of ROOT can be used without the need for inclusion such as "**#include<stdio.h>**". You even do not have to start a function properly like: "**int main()**". Example: we create an instance of the ROOT class named TBrowser and call its instance as "myBelovedBrowser" below:

```
root [0] TBrowser myBelovedBrowser
root [1] _
```



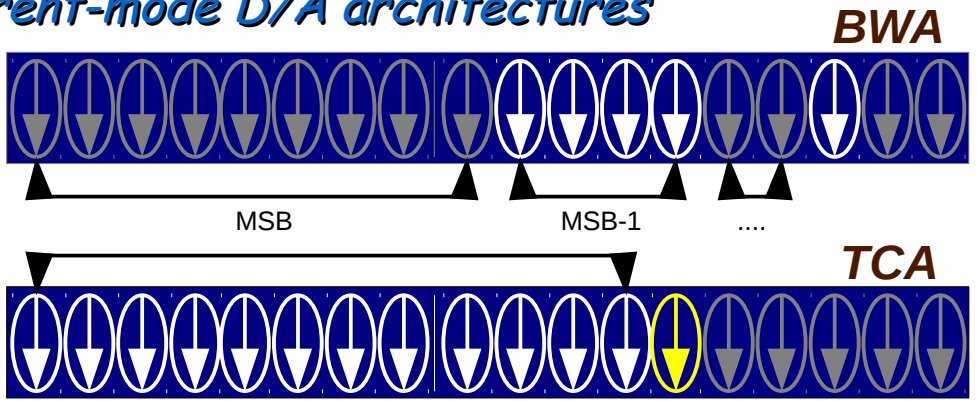
  - Or we write a loop on the fly:

```
root [0] for (int i=0 ; i<10 ; i++) {
end with '}', '@':abort > printf("Square root of %d is %d \n", i*i, i);
end with '}', '@':abort > }
Square root of 0 is 0
Square root of 1 is 1
Square root of 4 is 2
Square root of 9 is 3
Square root of 16 is 4
Square root of 25 is 5
Square root of 36 is 6
Square root of 49 is 7
Square root of 64 is 8
Square root of 81 is 9
root [1] _
```

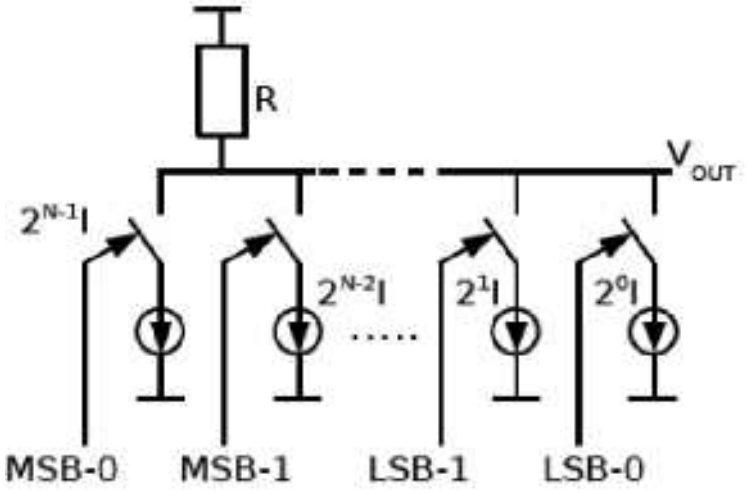
Usage - Scripting

An example script to compare two current-mode D/A architectures

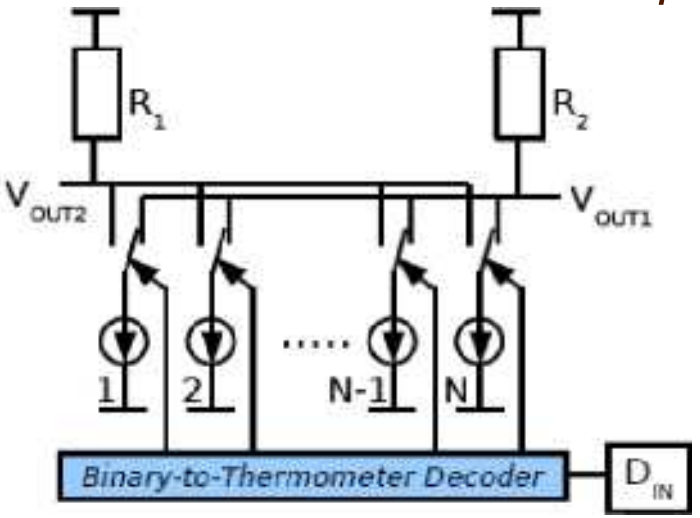
- A **10-Bit** current-mode D/A converter
- Two possible **architectures**
- Have to **choose one**
- Need for **qualitative comparison**
- Monte Carlo (MC) is a **must**



- Generate random numbers out of a Gaussian with a x_c **of 1** and a σ **of 0.02**
- Let these numbers to be the unit current sources forming the two D/As
  - ***TCA Case***: each step is represented by an addition of a single current source defining the output voltage
  - ***BWA Case***: let the sum of the first $2^{(N-1)}$ sources form MSB, sum of the next $2^{(N-2)}$ sources to form the next bit after MSB, and so on.
- Calculate INL and DNL for both of the architectures in RMS and compare.



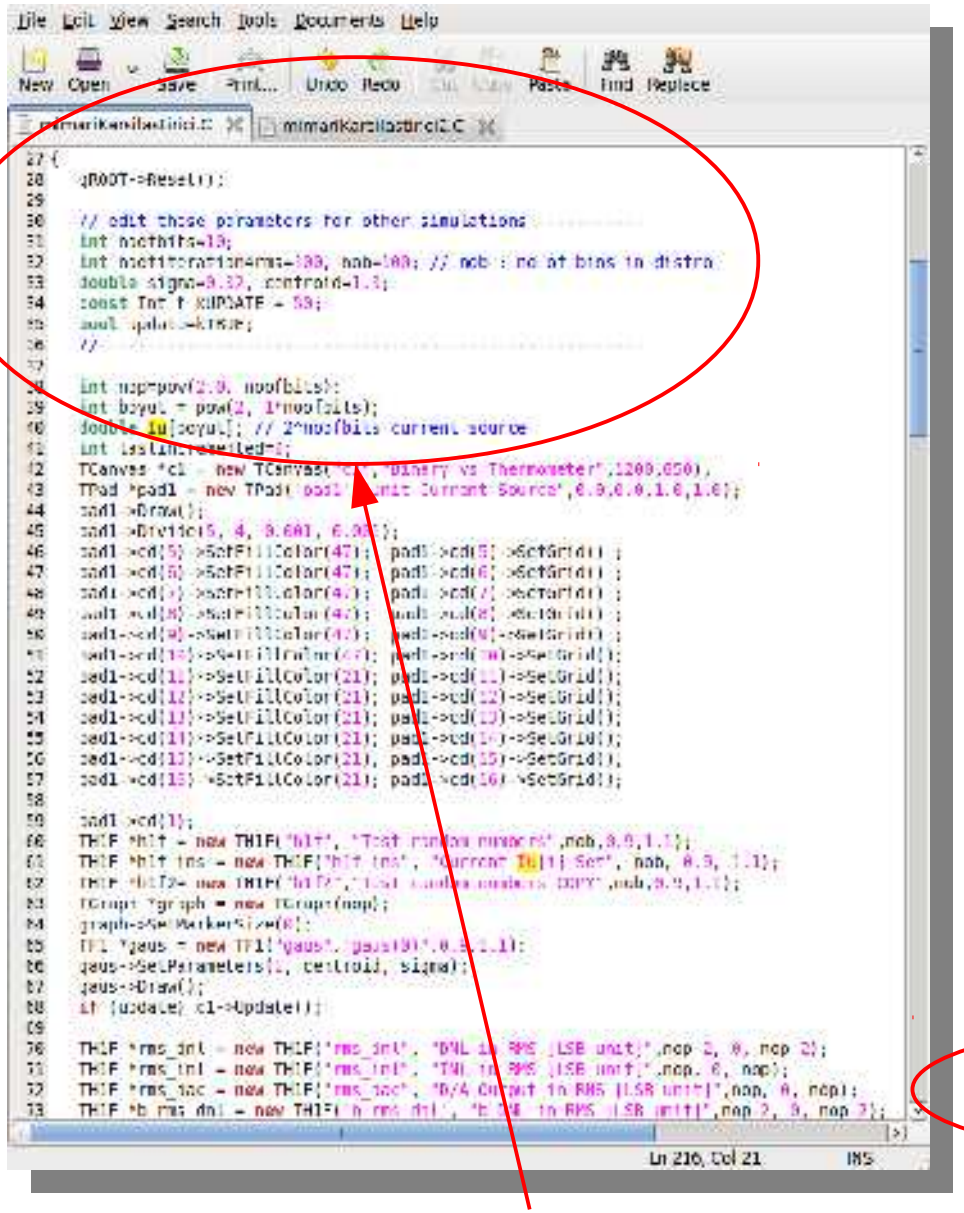
Binary weighted (BWA)

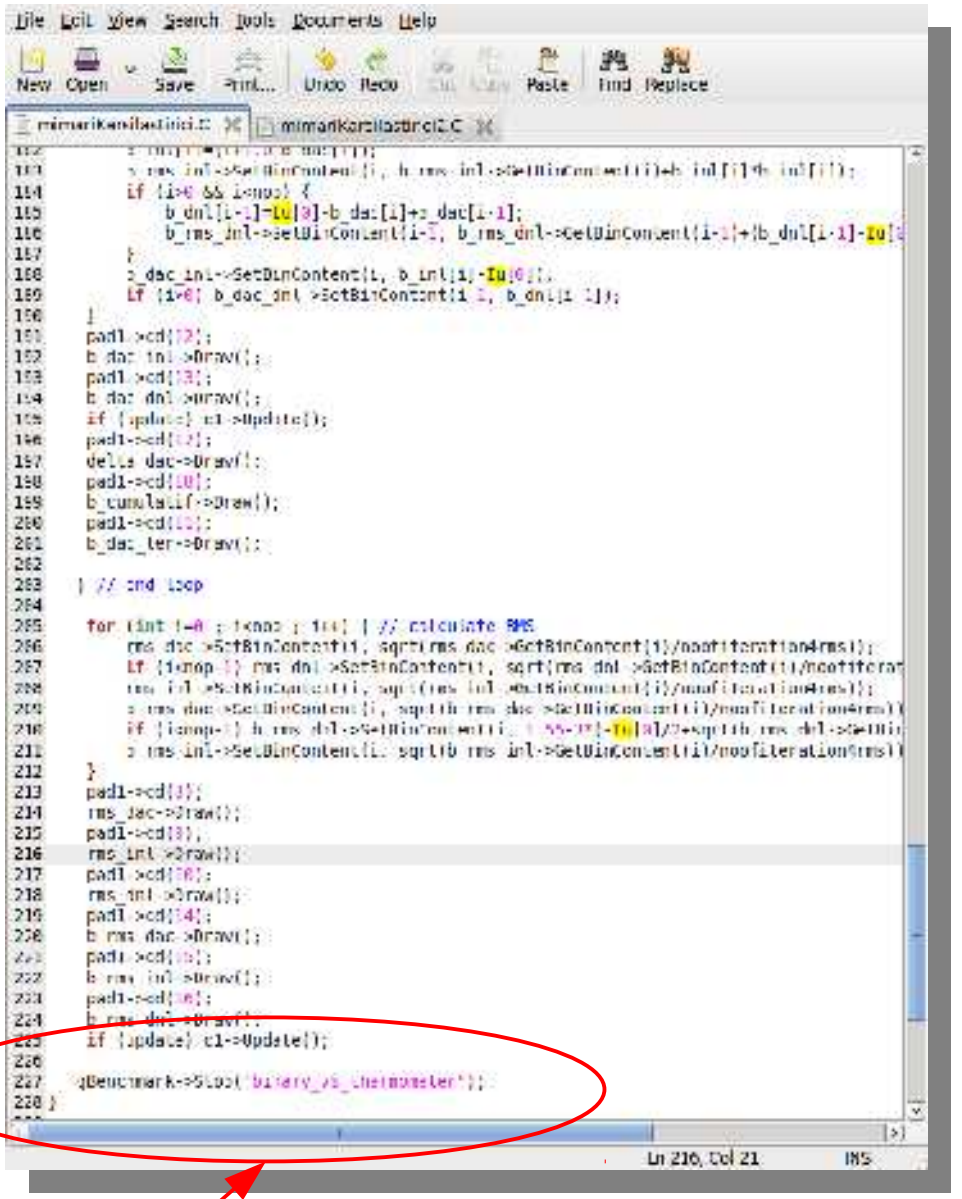


Thermometer coded (TCA)

Usage - Scripting

An example script to compare two current-mode D/A architectures



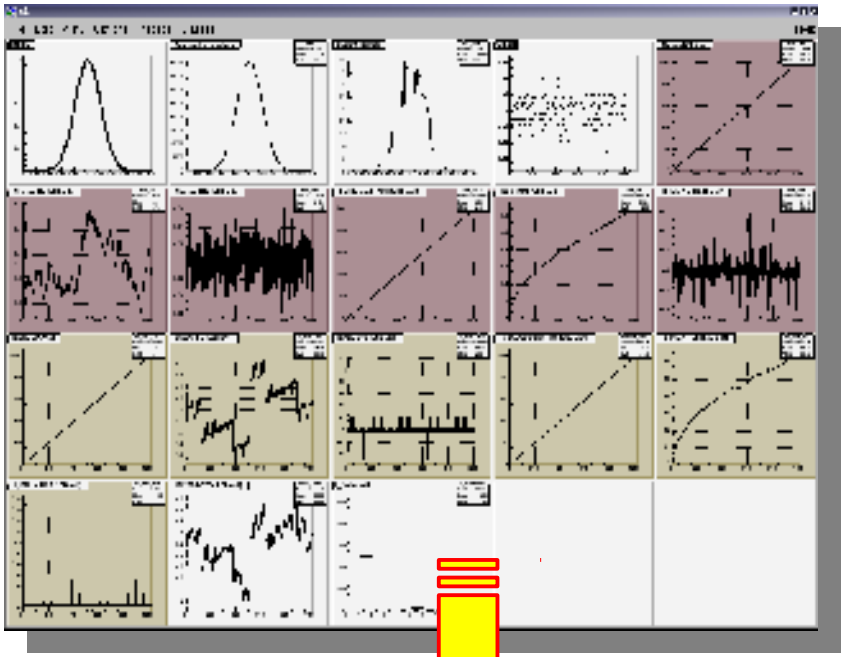


ArchitectureComparer.C

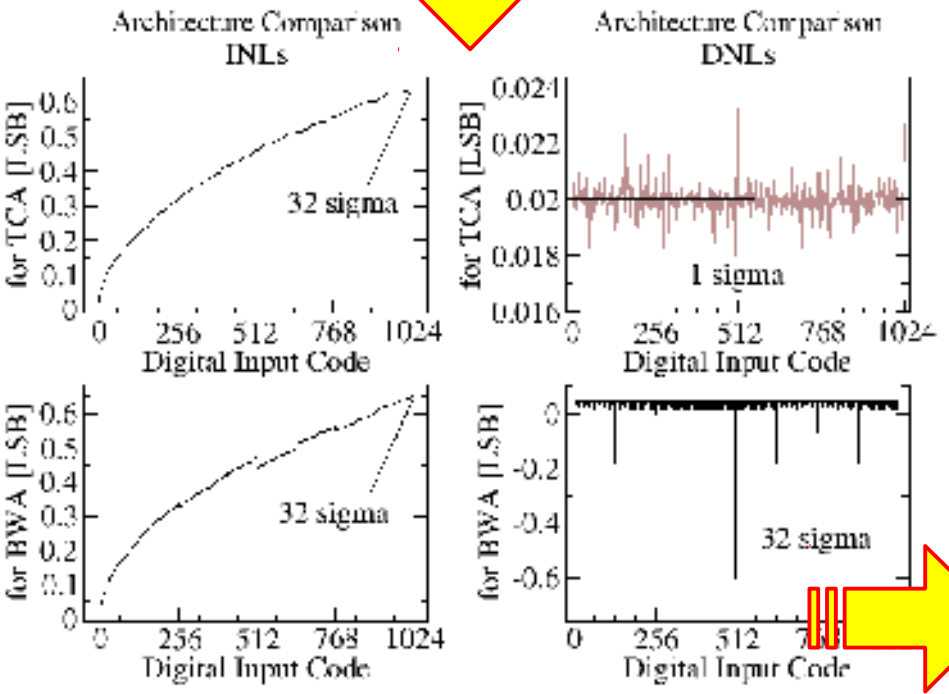
- *Beware that the script does **not** have a **name***
- *Note how the unnamed script starts (**no** header inclusion) and ends (**no** return value).*

Usage - Scripting

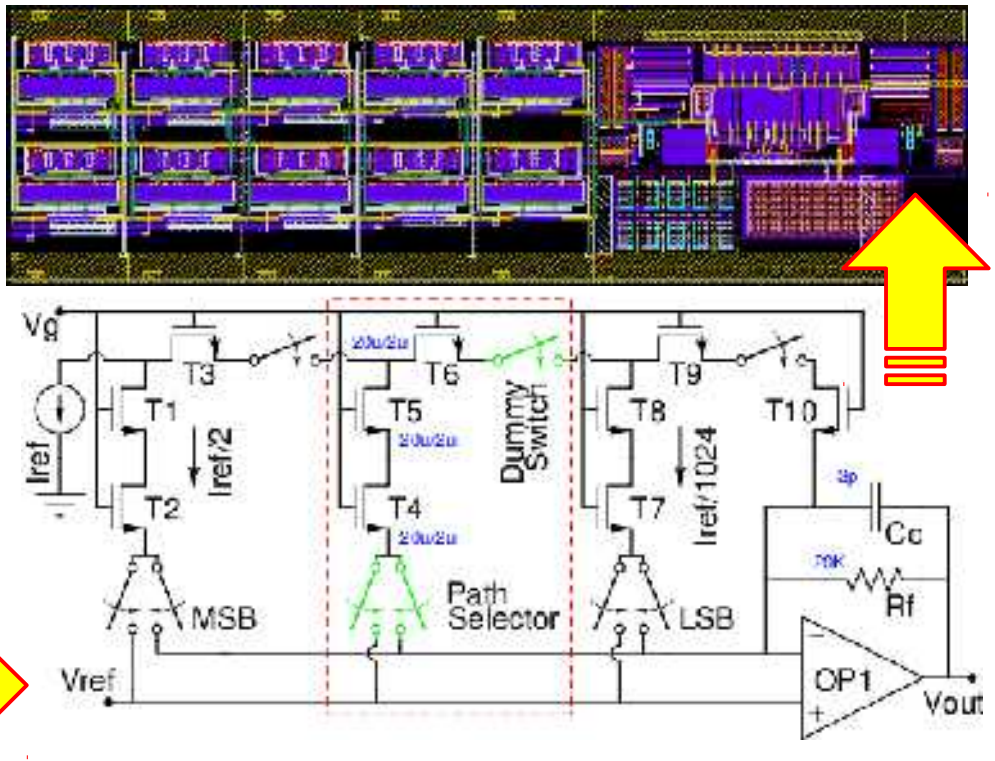
An example script to compare two current-mode D/A architectures



- *INLs are almost identical.*
- *DNLs are quite different, TCA behaves much better, however it will occupy a huge space on the chip layout*
- *Decided, designed the D/A accordingly, delivered it to the experiment people and published the paper*
- *Mission complete !!*



Used within the CMAD front-end ASIC designed for RICH-I detector of the COMPASS experiment at CERN



Usage - Compilation

Compiling a script into a "*.so" library

- ArchitectureComparer.C is ***interpreted*** by CINT "***slowly***"
- When ***compiled***, the script will be ***executed***, instead of being interpreted, this is "***fast***"
- With ***almost*** no modification, one can compile the script into a "*.so" library
- "..almost no modification.." actually means:
  - Header files of the classes used must be included (e.g. if *TCanvas* is used then I need the following statement to be inside my code: "`#include<TCanvas.h>`")
  - The function must be given a name, preferably same as the file name. (e.g. for "name*.C*" as the file name, I would write "`int name() {}`")

- Now the script is ready for being compiled into an "***.so***" lirary:

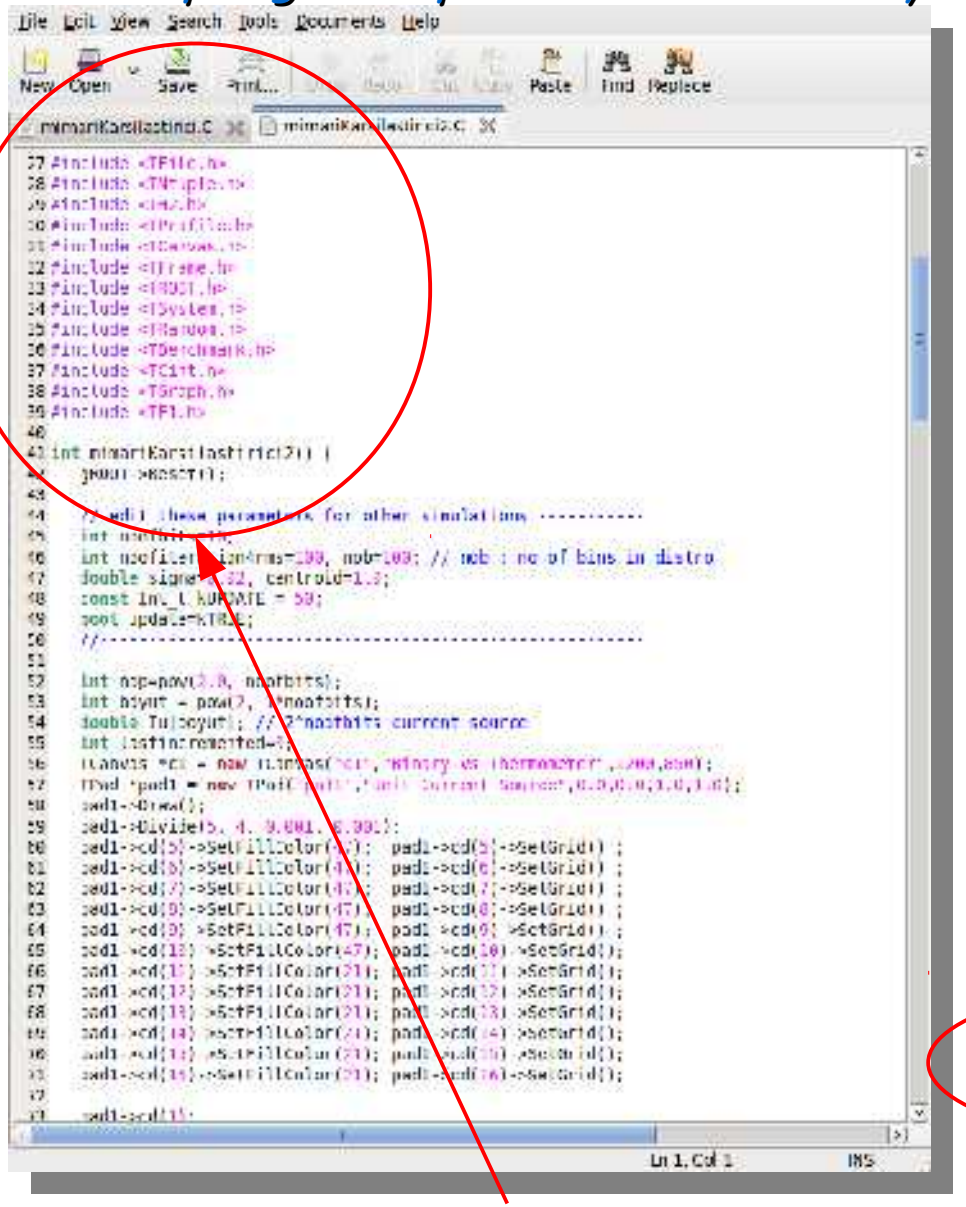
```
> root ArchitectureComparer.C++
root [0] Processing ArchitectureComparer.C++...
Info in <TUnixSystem::ACLiC>: creating shared library
  /home/oc/Documents/HEP_Okulu/workDir/root/./ArchitectureComparer_C.so
```

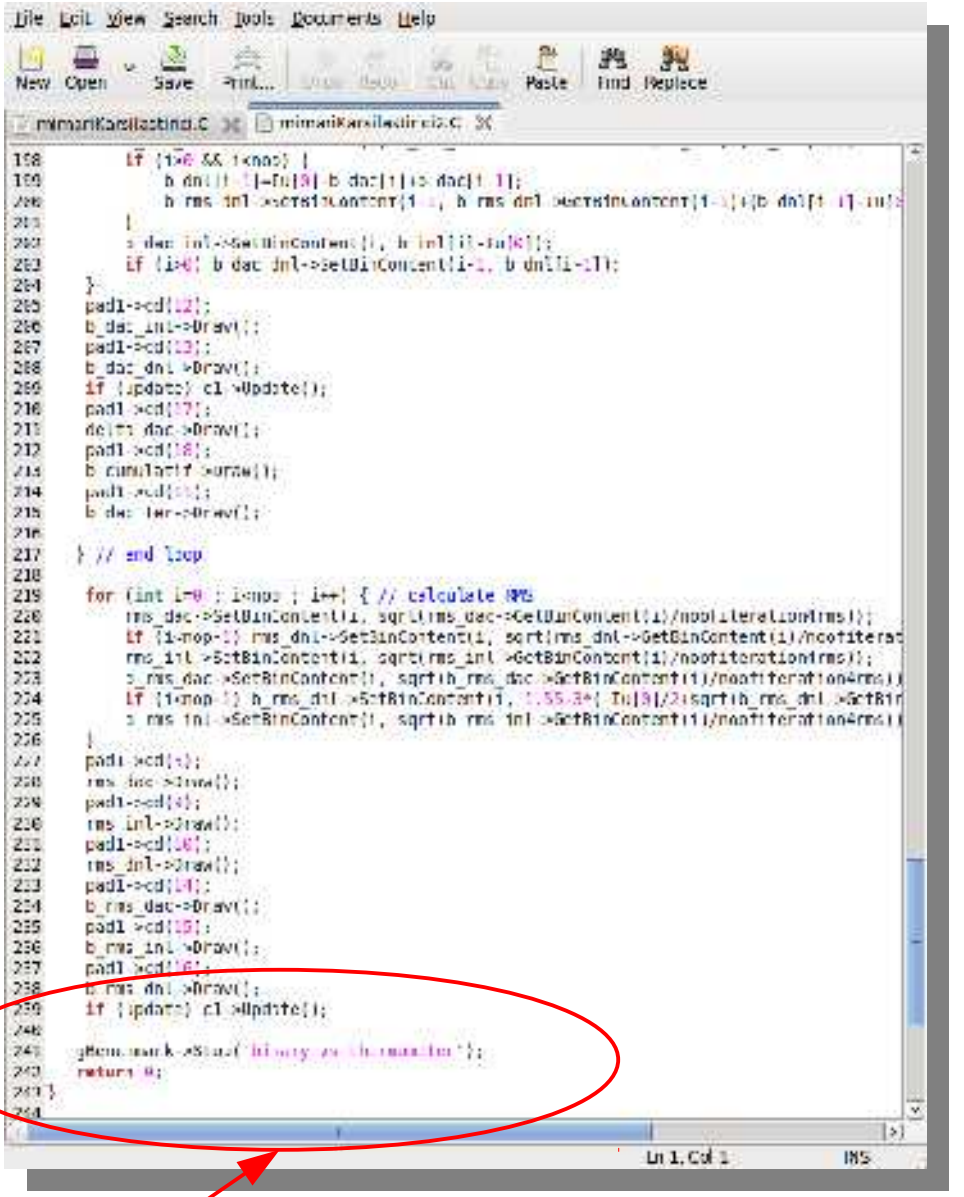
- I can use my library later at any time I wish so:

```
oc@olmak2:~/Documents/HEP_Okulu/workDir/root$ root -l
root [0] .L ArchitectureComparer_C.so
root [1] ArchitectureComparer()
```

Usage - Compilation

Compiling a script into a "*.so" library





ArchitectureComparer2.C

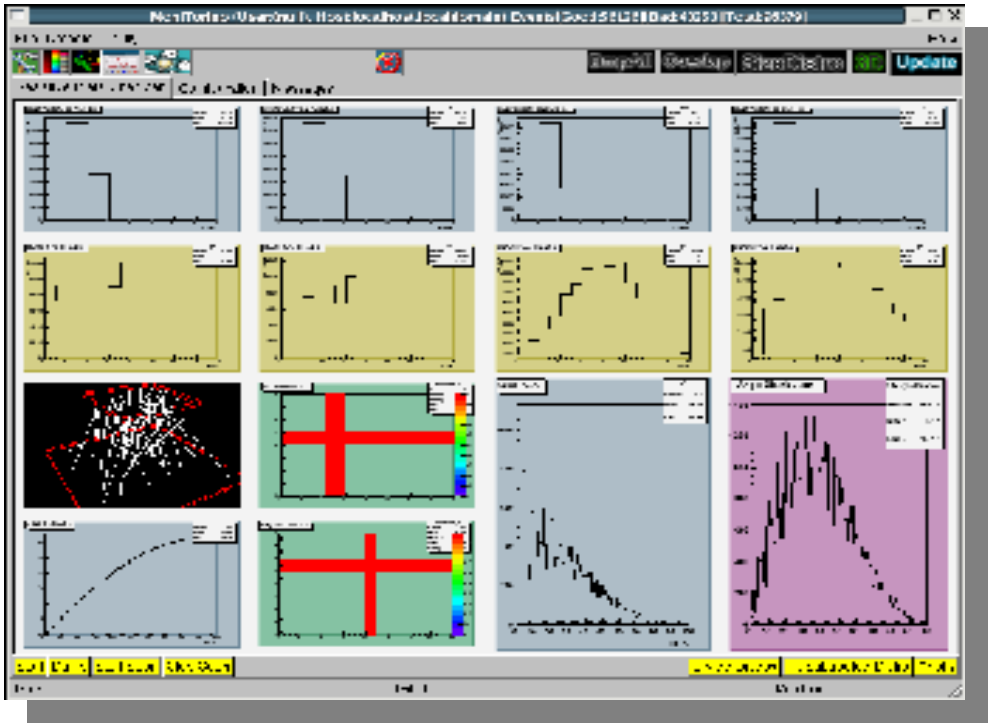
- *Beware that the script **does** have a name*
- *Note how the **named** script starts (**header inclusion**) and ends (**with** a return value).*

Usage – Compiling Standalone

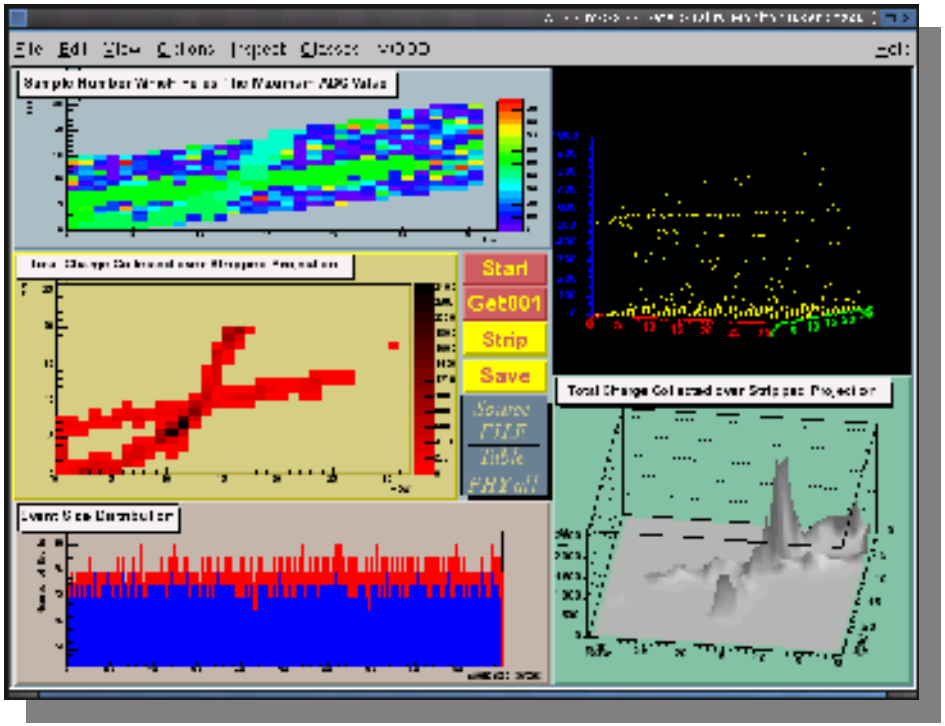
Application development

- ArchitectureComparer() can be loaded into ROOT environment by "***.L***" and can be invoked as if it was a *native ROOT command*
- It will be executed *much faster*
- It will need ROOT to have already been installed
- ***However***, it is also possible to have a standalone application which uses ROOT classes ***as external libraries*** without the need for ROOT environment for execution.

On-line monitoring and off-line analysis tool (MoniTorino) developed for an educational double RPC experiment measuring angle distributions of cosmic particles at the university and INFN of Turin/Italy

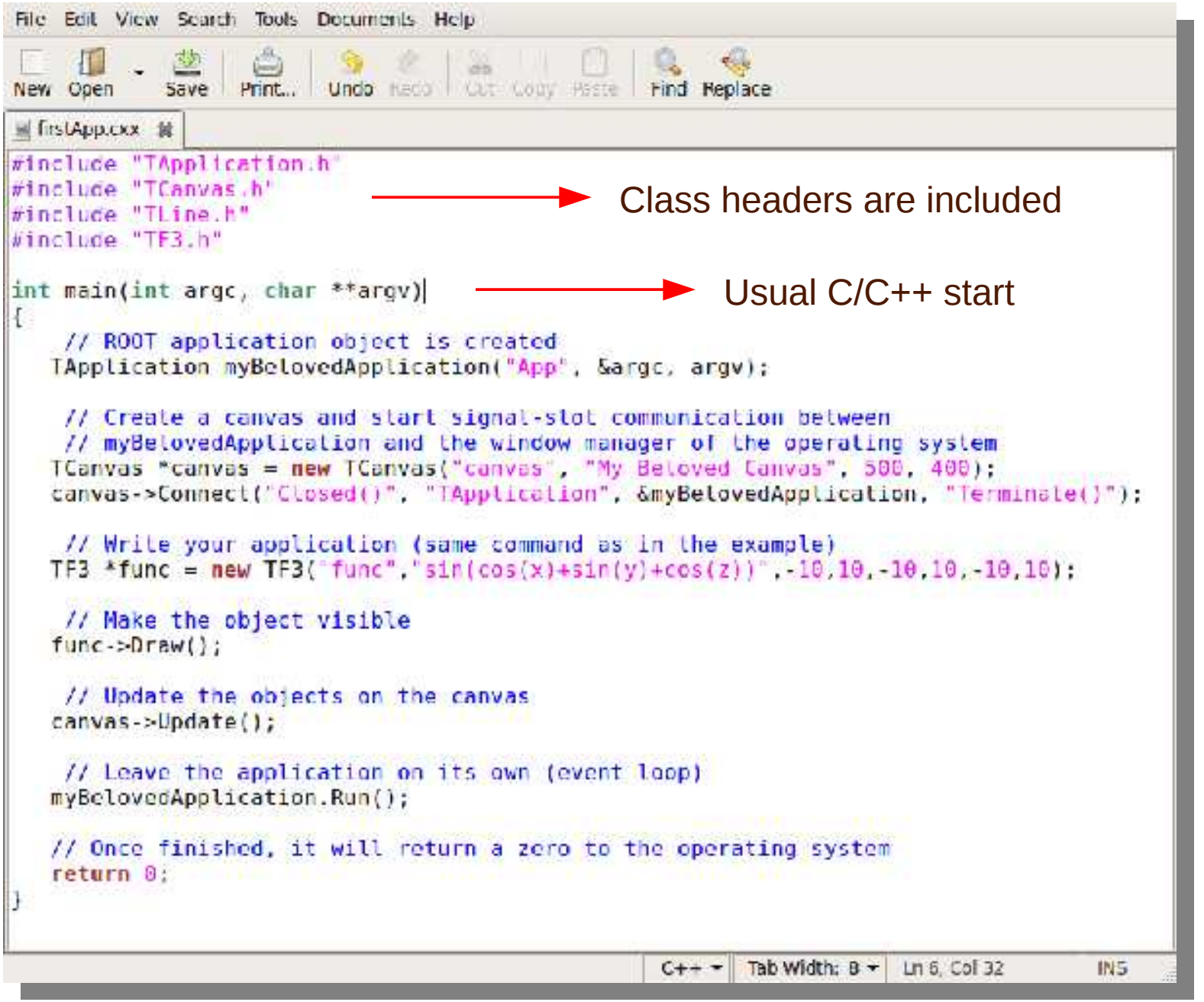


Very first version of the data quality monitoring tool, namely MOOD (Monitor Of On-line Data), developed for the ALICE experiment at CERN.



Usage – Compiling Standalone

Standalone application version of the example



Usage – Compiling Standalone

Standalone application version of the example

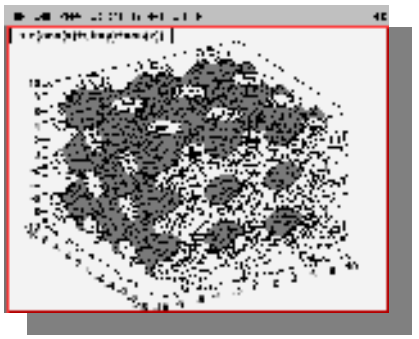
- *An elegant method to write applications is via **Makefile**, but we will not do it here*

- To compile the code in the previous page:
  - **g++ -L/usr/lib/root -lCore -lCint -lRIO -lNet -lHist -lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -lPhysics -lz -pthread -lm -ldl -rdynamic -pthread -m64 -I/usr/include/root -L/usr/lib/root -lCore -lCint -lRIO -lNet -lHist -lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -lPhysics -lz -lGui -pthread -lm -ldl -rdynamic firstApp.cxx -o firstApp**

- *Remembering all the above things is hard, therefore we will use:*
  - ***root-config**: a command-line tool to make lives of ROOT users easy (libraries usually have tools like this one)*
  - *It returns appropriate lines needed for compilation*
  - *Usually used in-between " ` ", aka escape symbol*

  - **g++ `root-config --glibs --cflags` firstApp.cxx -o firstApp**

```
oc@olmak2:~/ISOTDAQ/root$ ./firstApp &
[1] 16811
oc@olmak2:~/ISOTDAQ/root$ _
```



ROOT – A brief introduction

Survival with ROOT != Survival at ROOT != Survival despite ROOT

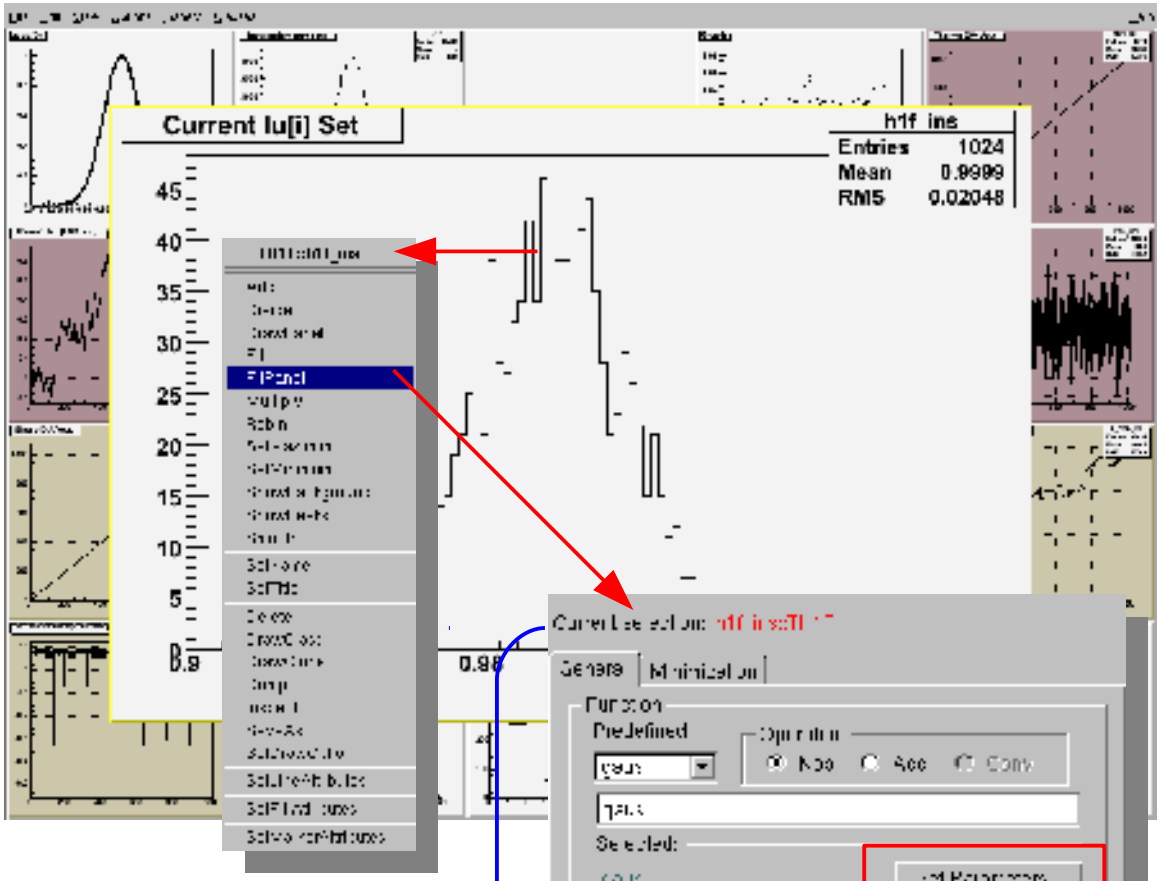
*

- Introduction to ROOT
  - What is it ?
  - Why is it good ?
- Using ROOT
  - Command line, batch-mode, root console or terminal
  - Scripting/Interpretation
    - Example script comparing two current-mode D/A converter architectures designed for COMPASS
  - Compilation
    - Compiling a script as a " *.so " shared object library
    - Compiling standalone
    - Application development
    - Example standalone application
- GUI of ROOT
  - Human interaction
  - Creating a GUI
- Survival [with/at/despite] ROOT
  - User's guide (refer once)
  - Referring to:
    - $ROOTSYS/tutorials (refer once per problem)
    - $ROOTSYS/test (refer once per problem)
  - HTML source code documentation (refer continuously)
- External library usage from within ROOT
  - DQM of ALICE experiment @ CERN
    - Simplified DAQ operation
  - Understanding the detector data
    - Accessing and decoding data

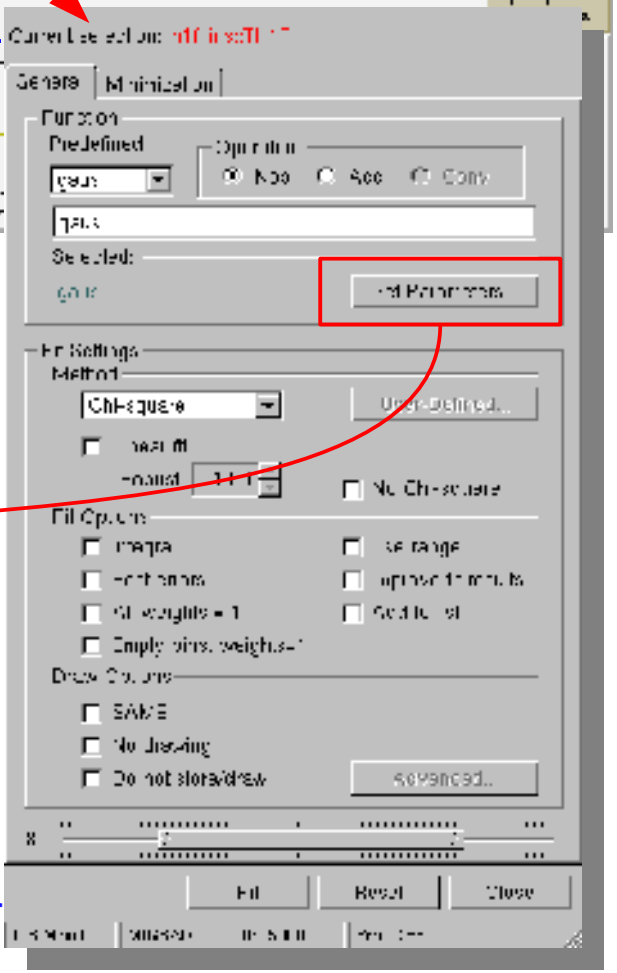
Usage

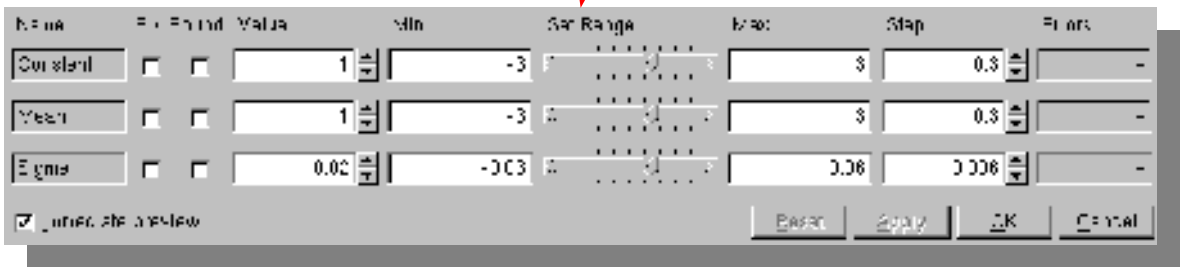
GUI – Human interaction

- All created windows and everything but everything on these windows (canvases, pads, histograms, titles, pave texts, sub-windows, axises, etc.) are either ROOT class instances or are instances of derived classes.
- Right click would bring a menu of some of the member functions of the instances created on the heap



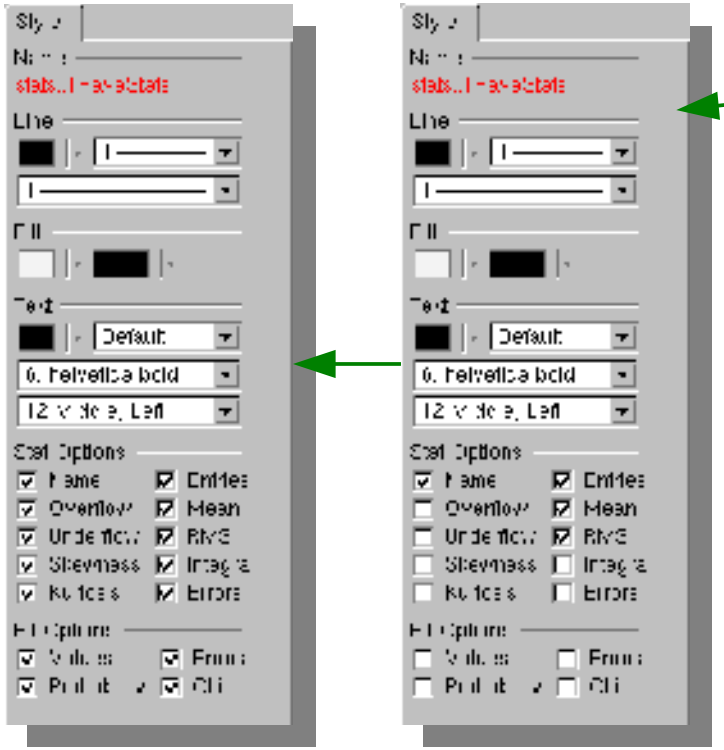
- *Pre-defined fit function*
- *Many options available*

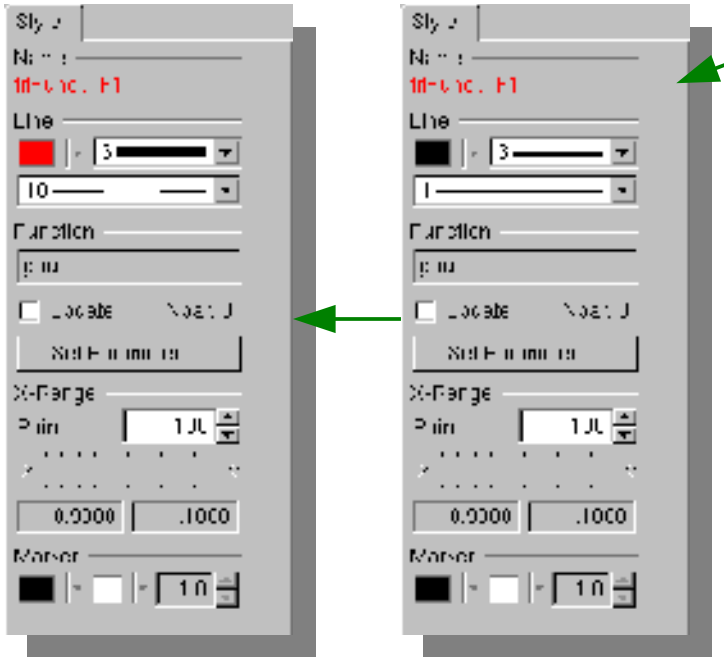


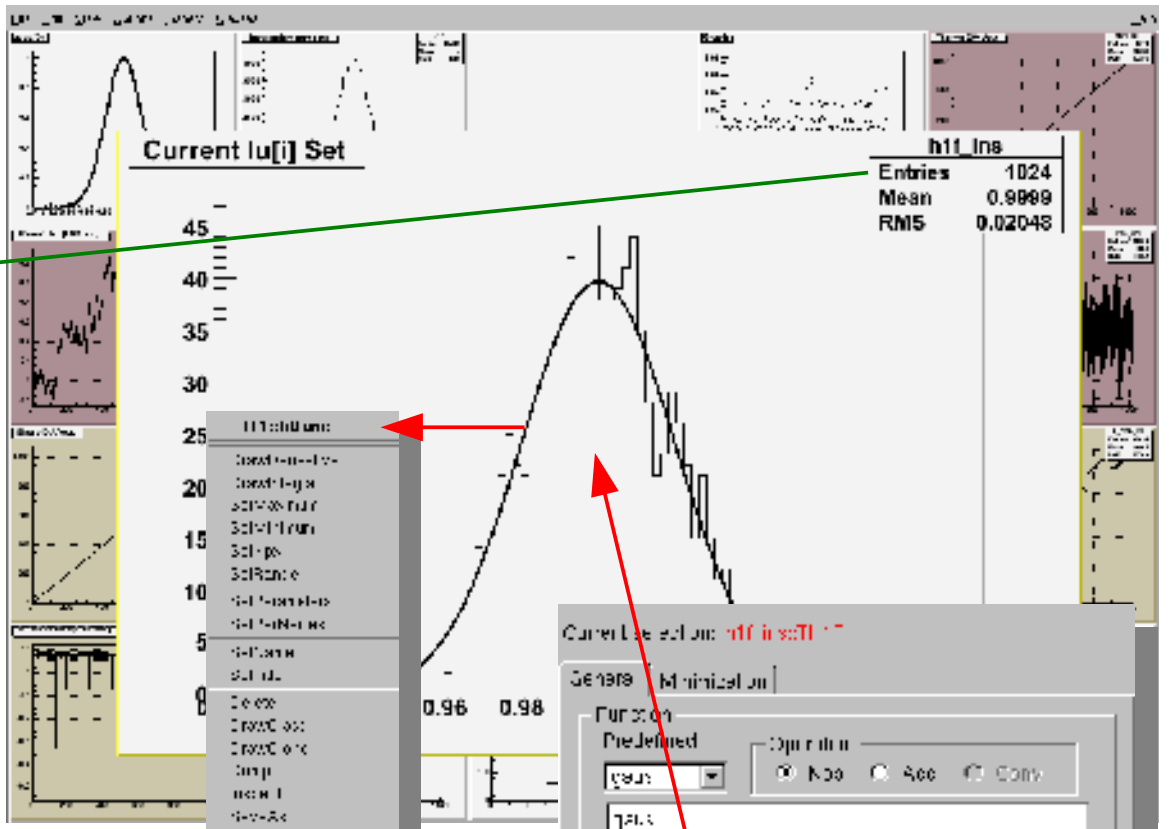


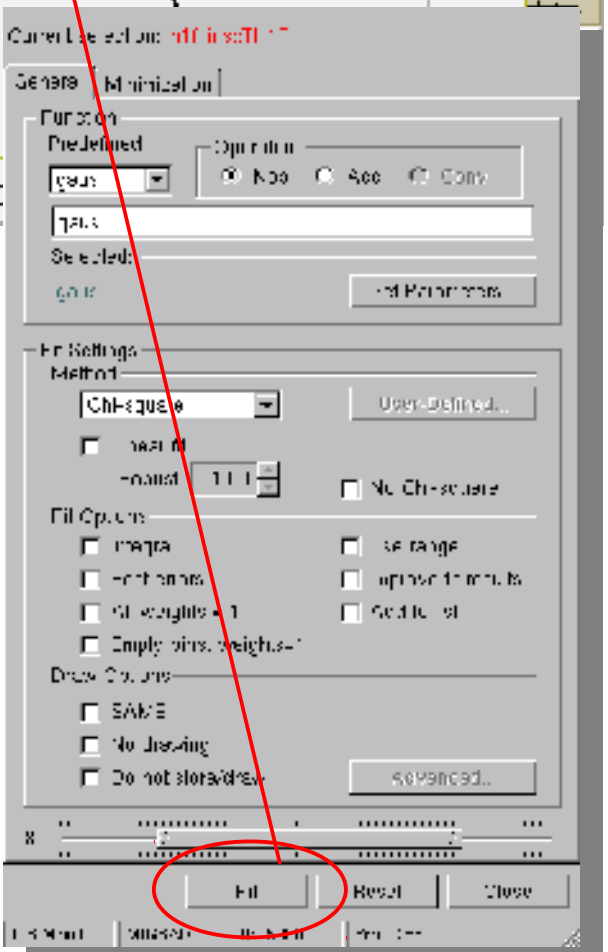
Usage

GUI - Human interaction







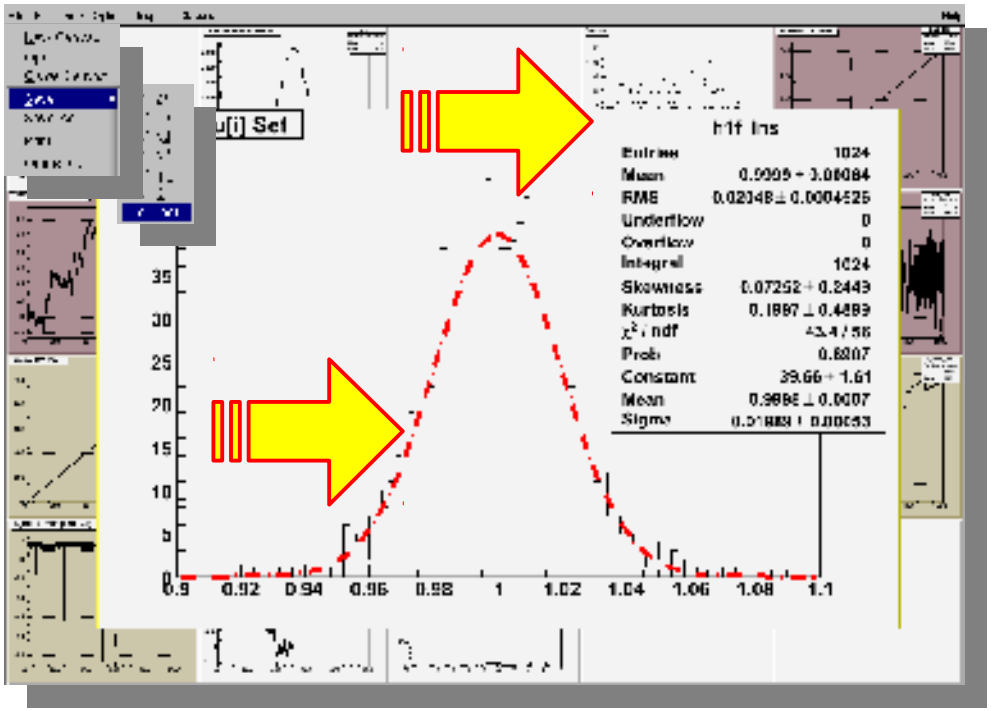


- *After fitting ends, adding make-up is also possible*
- *I want my fit curve to be **thick**, **dashed** and **red***
- *And I want more statistical information on the histogram*

Usage

GUI – Human interaction

- We have developed the code needed, performed the analysis, made the results more visible by adding make-up, ***BUT*** the mission is not yet complete !
- The ***work*** should be ***saved*** !
- There are many available methods:
  - It can be saved as a *.C source code
    - To get the results again it must be reinterpreted by ROOT (e.g. "`root -l code.C`")
  - It can be saved as a *.root file
    - Content can be directly browsed by a TBrowser object (e.g. "`TBrowser a`")
  - It can be saved as a picture:
    - ps, eps, gif v.b.



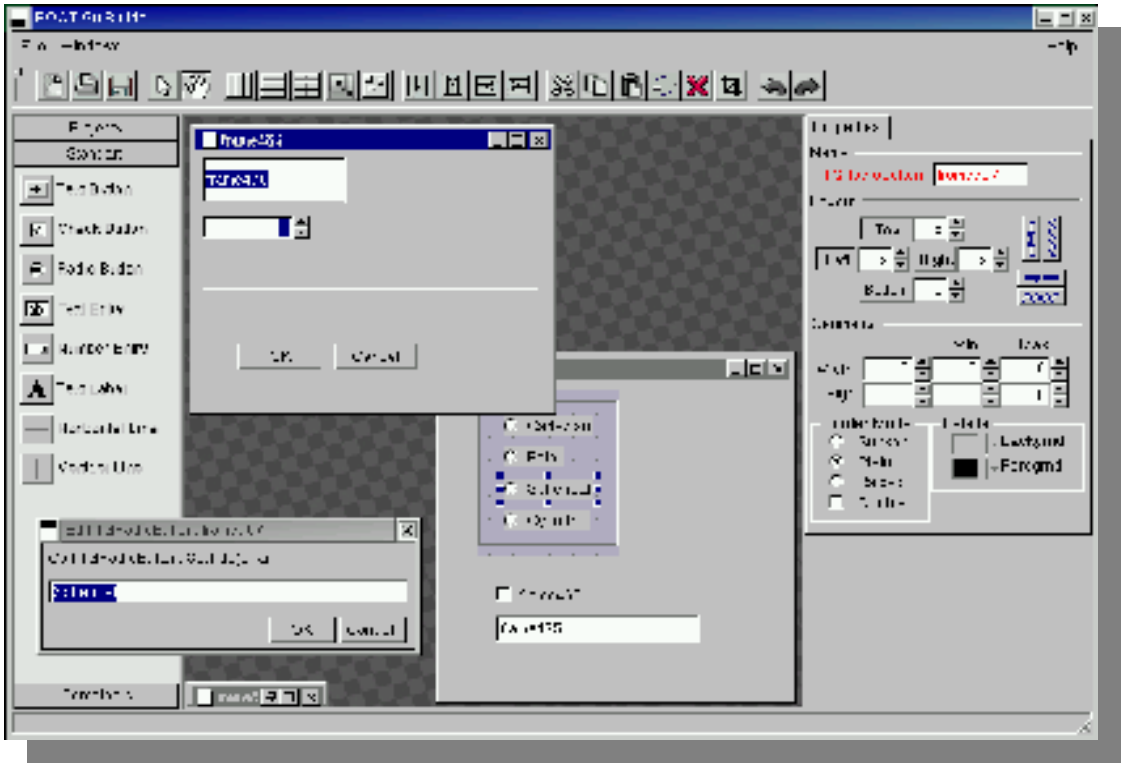
- ***Thicker***, ***dashed*** *and* ***red***
- *More statistical information*

Example

Creating a GUI

- Users can design GUIs for their applications developed with ROOT libraries
- There are two main ways of doing so, usually iteratively used:
  - Writing source code
  - Using an instance of "***TRootGuiBuilder***" class

```
oc@olmak2:~/ISOTDAQ/workDir/root$ root -l
root [0] TRootGuiBuilder a
root [1] _
```



- Graphically prepared application is saved as *.C script and development of actual code is continued over this "template".
- ***Educative: design*** the GUI, ***save*** it, ***read*** *it !!*
- ***Saves time***: you do not have to memorize ROOT GUI classes
- ***Save frequently against frequent crashes not to lose work !!***

ROOT – A brief introduction

Survival with ROOT != Survival at ROOT != Survival despite ROOT

*

- Introduction to ROOT
  - What is it ?
  - Why is it good ?
- Using ROOT
  - Command line, batch-mode, root console or terminal
  - Scripting/Interpretation
    - Example script comparing two current-mode D/A converter architectures designed for COMPASS
  - Compilation
    - Compiling a script as a " *.so " shared object library
    - Compiling standalone
    - Application development
    - Example standalone application
- GUI of ROOT
  - Human interaction
  - Creating a GUI
- Survival [with/at/despite] ROOT
  - User's guide (refer once)
  - Referring to:
    - $ROOTSYS/tutorials (refer once per problem)
    - $ROOTSYS/test (refer once per problem)
  - HTML source code documentation (refer continuously)
- External library usage from within ROOT
  - DQM of ALICE experiment @ CERN
    - Simplified DAQ operation
  - Understanding the detector data
    - Accessing and decoding data

Survival with/at/despite ROOT

User's Guide

- Meeting with a new library stands upon 4 legs: ***(subjective)***
  i. User's manual
  ii. Source code documentation
  iii. Set of examples showing library usage (tutorial, test, example, etc.)
  iv. You, reading these resources and trying them out

- ***We are lucky:*** ***ROOT has all these things !!*** Meeting with ROOT ***is trivial !!***

- ***User's guide***, tells us on which ideals the library has been developed, its architecture and "***hello world***" examples with a lot of relevant explanations.
- It ***does not change*** fast, it is rather ***static***.
- It must be studied/digested ***once*** at the beginning
- For many of us, only having a ***skin-deep look*** renders ***enough*** for every new ***major release*** (i.e. change of major version number)

- ***All*** the things mentioned within this lecture ***and more***, written ***fluently*** with a lot of useful tricks can be found in ROOT User's Guide.

Survival with/at/despite ROOT

User's Guide

- Meeting with a new library stands upon 4 legs: *(subjective)*
  i. User's manual
  ii. Source code documentation
  iii. Set of examples showing library usage (tutorial, test, example, etc.)
  iv. You, reading these resources and trying them out

- ***We are lucky:*** ***ROOT has all these things !!*** Meeting with ROOT ***is trivial !!***

- ***User's guide***, tells us on which ideals the library has been developed, its architecture and "***hello world***" examples with a lot of relevant explanations.
- It ***does not change*** fast, it is rather ***static***.
- It must be studied/digested ***once*** at the beginning
- For many of us, only having a ***skin-deep look*** renders ***enough*** for every new ***major release*** (i.e. change of major version number)

- ***All*** the things mentioned within this lecture ***and more***, written ***fluently*** with a lot of useful tricks can be found in ROOT User's Guide.

*

Survival with/at/despite ROOT

User's Guide

ROOT User's Guide is more than ~500 pages ! Moreover ...

... it just teaches a language but not what to say in that language !

*

Survival with/at/despite ROOT

Effective function of $ROOTSYS/tutorials directory

- The line separating ***death*** and ***life*** while ***scripting***
- Not to be memorized; to be ***referred continuously*** while ROOTing; the lowest level of scripting examples, a teacher

- Learning to use a new library, especially in languages where almost everything is hard without library support, is ***equivalent*** to learning a new programing language
- A language is best learned by ***practicing***
- This directory is where you practice ROOT ***scripting*** language

- To get acquainted with ROOT, open (with your favorite text editor) ***all but all*** the scripts, have a look at each code, run with ***root -l xxx.C*** and understand what they do; not deeply, just to have an idea

- This is vital because:
  - Understanding ***what percentage*** of the tutorial codes can actually run without problems is important. Is the library you will use ***perfect*** ?
  - The scripts within this directory are the ***starting points*** for your future developments.
  - You do not have to remember which one does what.
  - However, you ***must be able to say*** "***there was something in the tutorials directory that does a similar thing...***" when you need assistance

Survival with/at/despite ROOT

Effective function of $ROOTSYS/test directory

- The line separating ***death*** and ***life*** while ***compiling standalone applications***
- Not to be memorized; to be ***referred continuously*** while ROOTing; the lowest level of application examples, a teacher

- Learning to use a new library, especially in languages where almost everything is hard without library support, is ***equivalent*** to learning a new programing language
- A language is best learned by ***practicing***
- This directory is where you practice ***application development*** with ROOT

- To get acquainted with ROOT, open (with your favorite text editor) ***all but all*** the source codes, have a look at each code, compile with ***make xxx*** and understand what they do; not deeply, just to have an idea

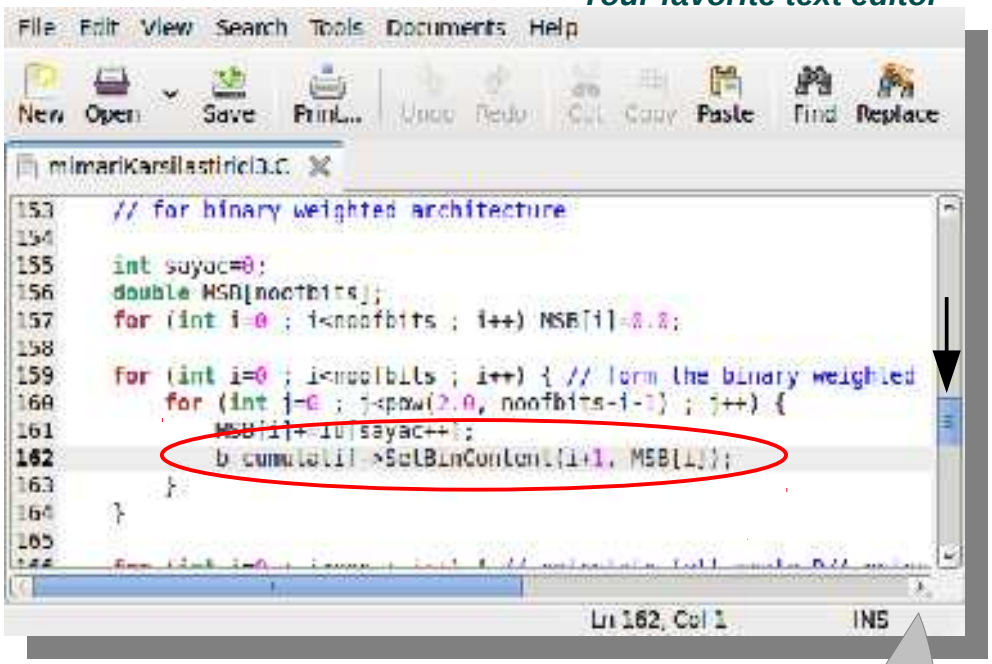
- This is vital because:
  - Understanding ***what percentage*** of the test codes can actually run without problems is important. Is the library you will use ***perfect*** ?
  - The source codes within this directory are the ***starting points*** for your future developments.
  - You do not have to remember which one does what.
  - However, you ***must be able to say*** "***there was something in the test directory that does a similar thing...***" when you need assistance

Survival with/at/despite ROOT

Usage of HTML documentation

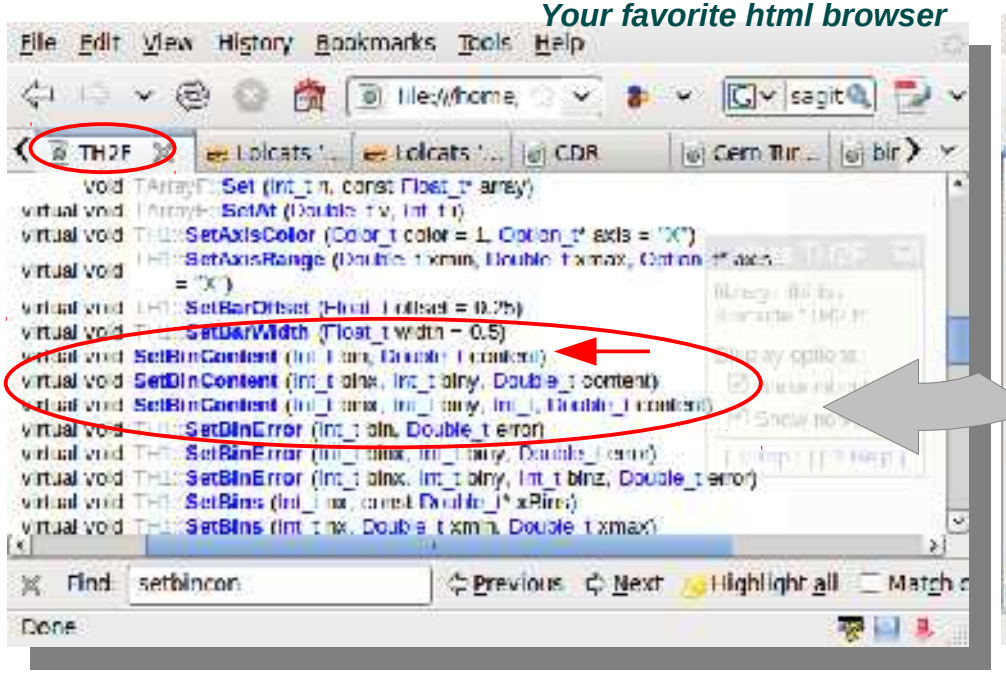
- ***The line*** in-between death and life
- It is not something to learn, it is to refer in a cyclic manner while ROOTing (documentation which is the closest to what you actually use)
- ***On-line/live:*** it is generated out of the version you use at the time, therefore:
  - *Un-like user's guides, it is **not static**; it is not full of "**old**" knowledge; it is **valid** and **up-to-date**, truly useful*
  - Equivalent to reading the source code of the ROOT library, ***least error-prone*** technique to learn what the command you use in your code actually does

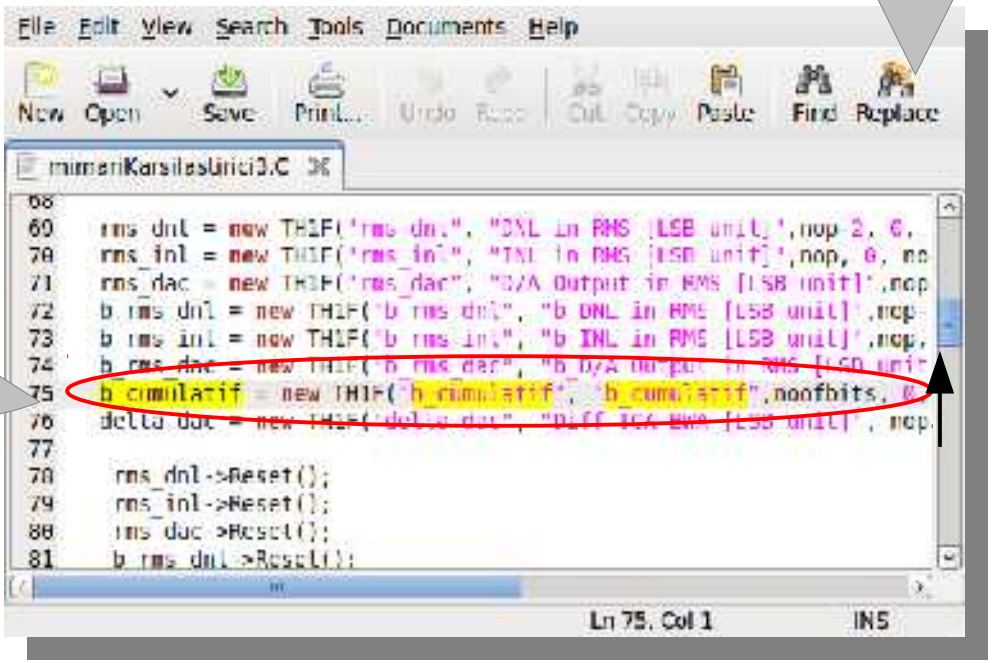
Your favorite text editor



Search for b_cumulatif via Ctrl-F to see what objects instance it is: an instance of TH1F class

Your favorite html browser





ROOT – A brief introduction

Survival with ROOT != Survival at ROOT != Survival despite ROOT

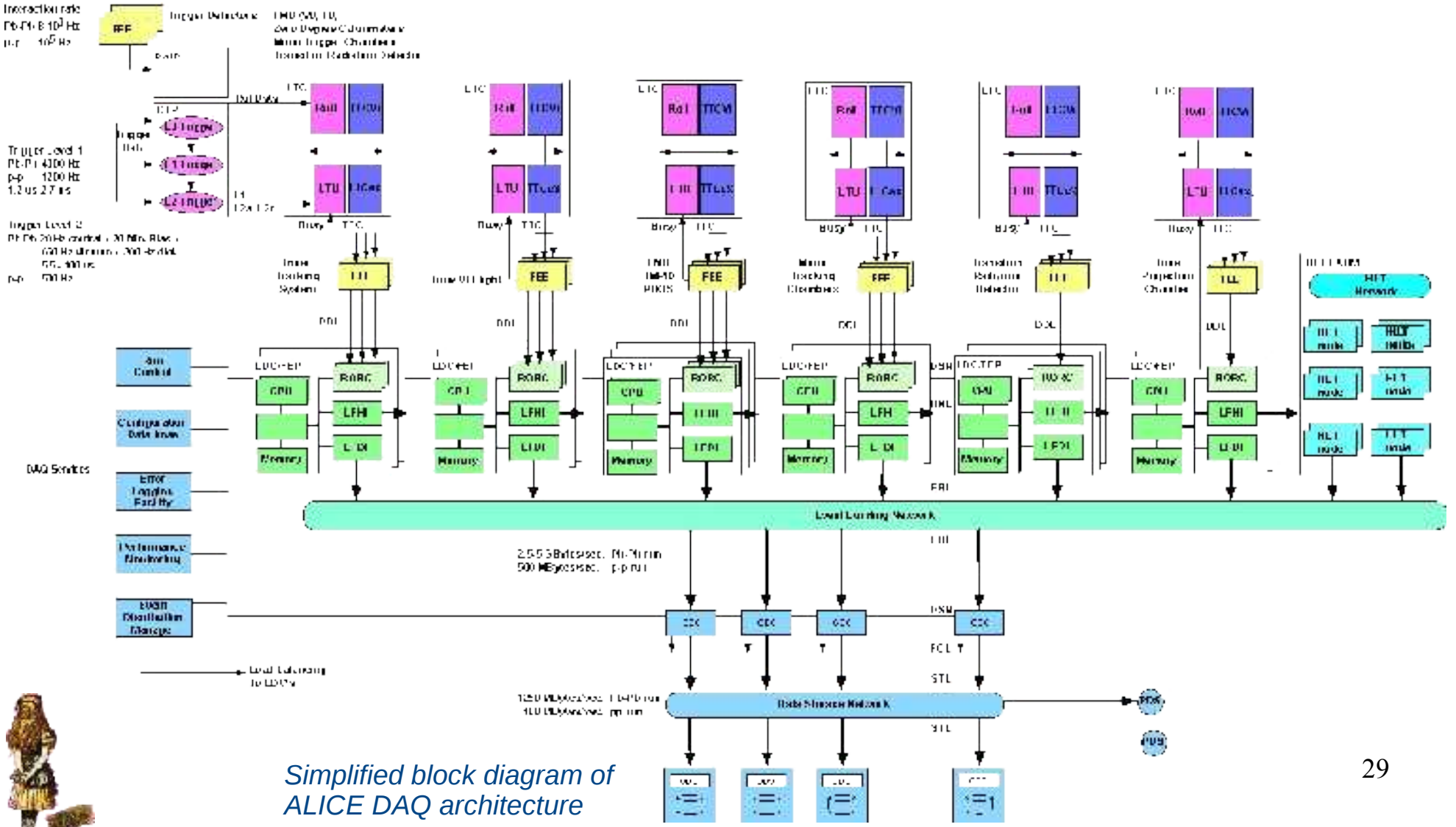
*

- Introduction to ROOT
  - What is it ?
  - Why is it good ?
- Using ROOT
  - Command line, batch-mode, root console or terminal
  - Scripting/Interpretation
    - Example script comparing two current-mode D/A converter architectures designed for COMPASS
  - Compilation
    - Compiling a script as a " *.so " shared object library
    - Compiling standalone
    - Application development
    - Example standalone application
- GUI of ROOT
  - Human interaction
  - Creating a GUI
- Survival [with/at/despite] ROOT
  - User's guide (refer once)
  - Referring to:
    - $ROOTSYS/tutorials (refer once per problem)
    - $ROOTSYS/test (refer once per problem)
  - HTML source code documentation (refer continuously)
- External library usage from within ROOT
  - DQM of ALICE experiment @ CERN
    - Simplified DAQ operation
  - Understanding the detector data
    - Accessing and decoding data

External library usage from within ROOT

Example: Data quality monitoring application for ALICE experiment at CERN

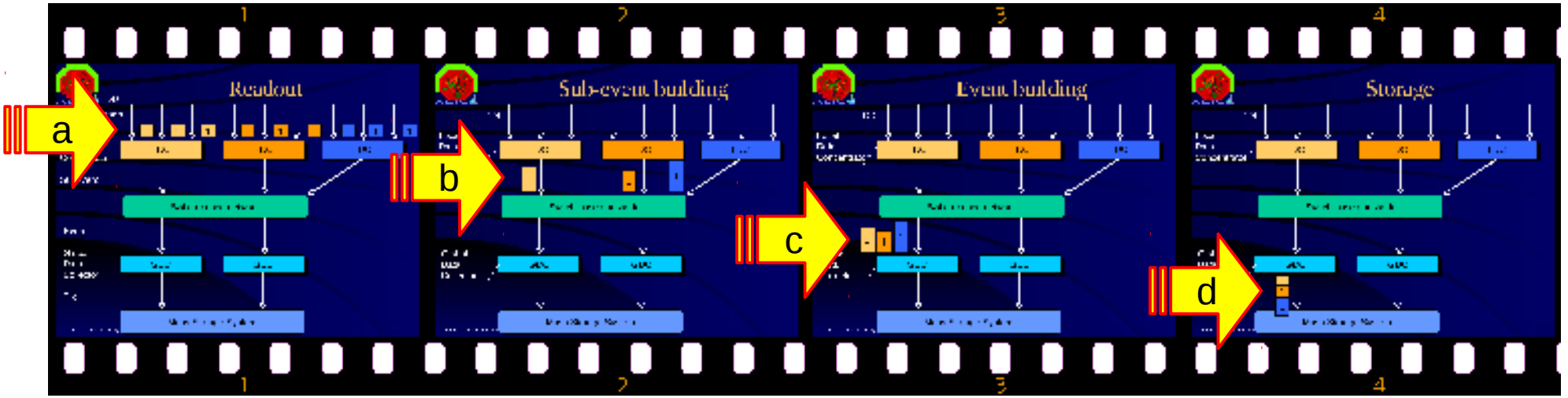
- **Large-scale** experiments having a **deep hierarchy** of sub-systems
- These systems need to be **monitored at different levels** (e.g. LDC, GDC, etc). This is because the quality of the data depends on many parameters (human, accelerator, detector slow control, etc.)
- Access to data is established by a **C/C++ library** provided by the **DAQ team**



Simplified block diagram of ALICE DAQ architecture

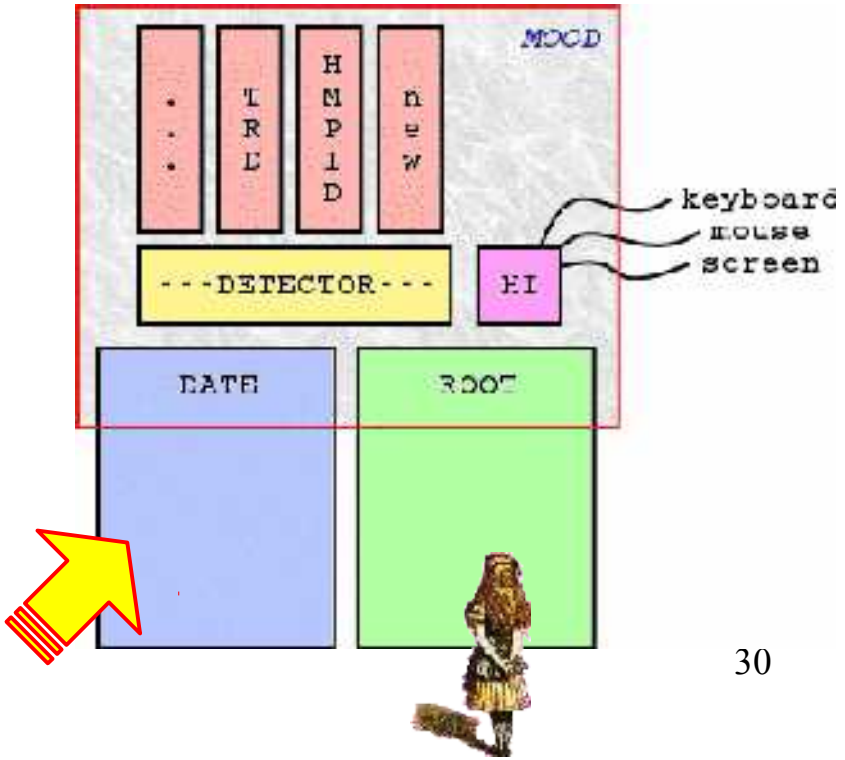
External library usage from within ROOT

Simplified DAQ operation



* Further simplified architecture (from Pierre Vande Vyvre's presentation)

- FE ASICs reading out the detectors generate ***payloads*** *(a)*
- Payloads travel further down within the DAQ network into LDCs (Local Data Collector) to get together, forming the ***event fragments*** *(b)*
- Event fragments move further down to GDCs (Global Data Concentrator) and are put together, forming ***sub-events*** *(c)*
- Sub-events get together forming the ***super-event*** and are sent to ***permanent data storage*** for ***off-line analysis*** *(d)*



- ***Our task*** is to develop a ***monitoring application*** using ***DATE*** library for data access and ***ROOT*** class library for GUI, event display, and analysis.

External library usage from within ROOT

Accessing data via monitoring functions provided by an external library

```
oc@olmak-x200:~$ root -l
root [0] gSystem->Load("${DATE_MONITOR_DIR}/
{DATE_SYS}/libmonitor.so");
root [1] monitorSetDataSource(":"); /* Enable local on-line monitoring*/
root [2] int *event;
root [3] monitorGetEventDynamic( &event );
root [4] printf( "%08x %08x \n", event[0], event[1] );
0000878c da1e5afe
Root [5] .q
oc@olmak-x200:~$ _
```

DATE Event Format

0 - Total size of the event (0000878c)
1 - Unique DATE event signature (da1e5afe)
2 - Size of the header (base & extension)
3 - Base event header structure version
4 - Type of event
5 - Number of the run associated to the event
6 - Unique event identification
7 - Level 2 trigger associated to the event
8 - Detector pattern associated to the event
9 - Attributes associated to the event
10 - ID of the LDC
11 - ID of the GDC
12 - Time stamp at the creation of the event

Super Event header

<Equipment Header "n">
<Data associated to the equipment "n">

Data generated by a specific part of a detector (payload)

<Equipment Header "n-1">
<Data associated to the equipment "n-1">

Payload

...

Summary

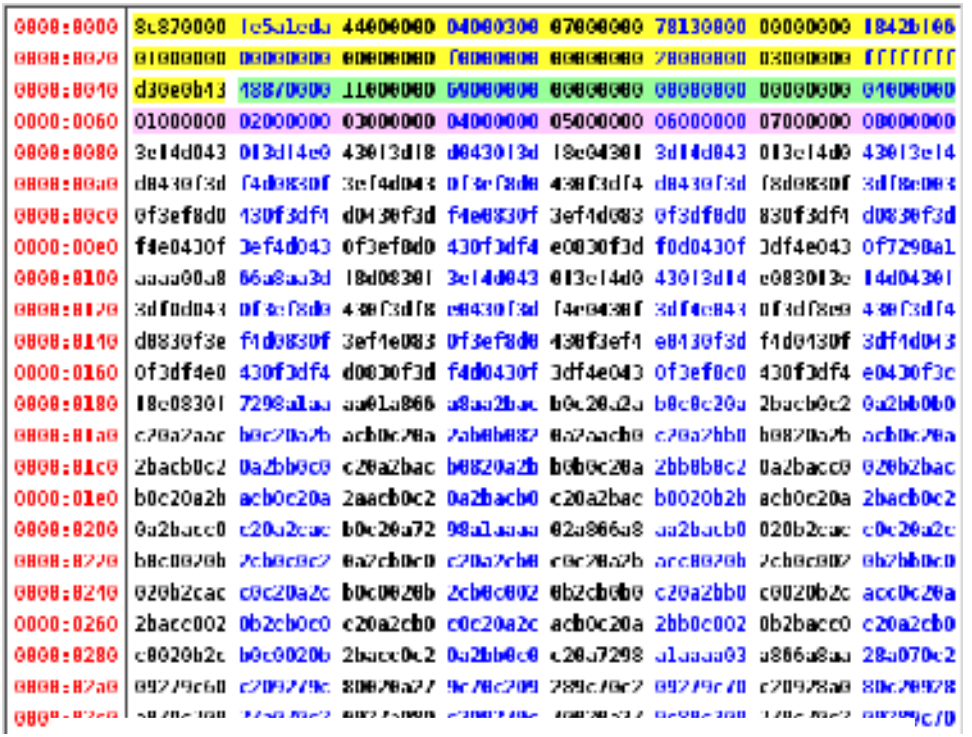
```
Root [0] → Load the monitoring library
Root [1] → Set data source
Root [2] → Variable to cast data onto
Root [3] → Get the data into the variable
Root [4] → Show the first two "words"
Root [4] → End of session
```

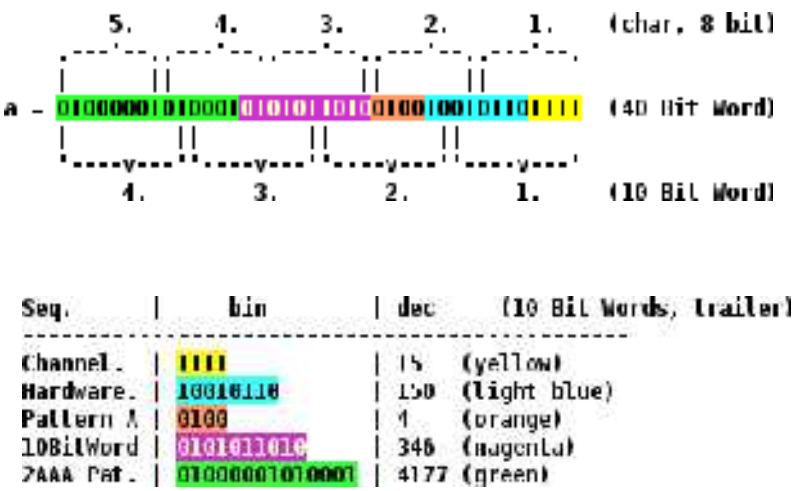
This is ~literally more information than your government has about you :)

External library usage from within ROOT

Parsing and/or casting the detector data – Decoding payload

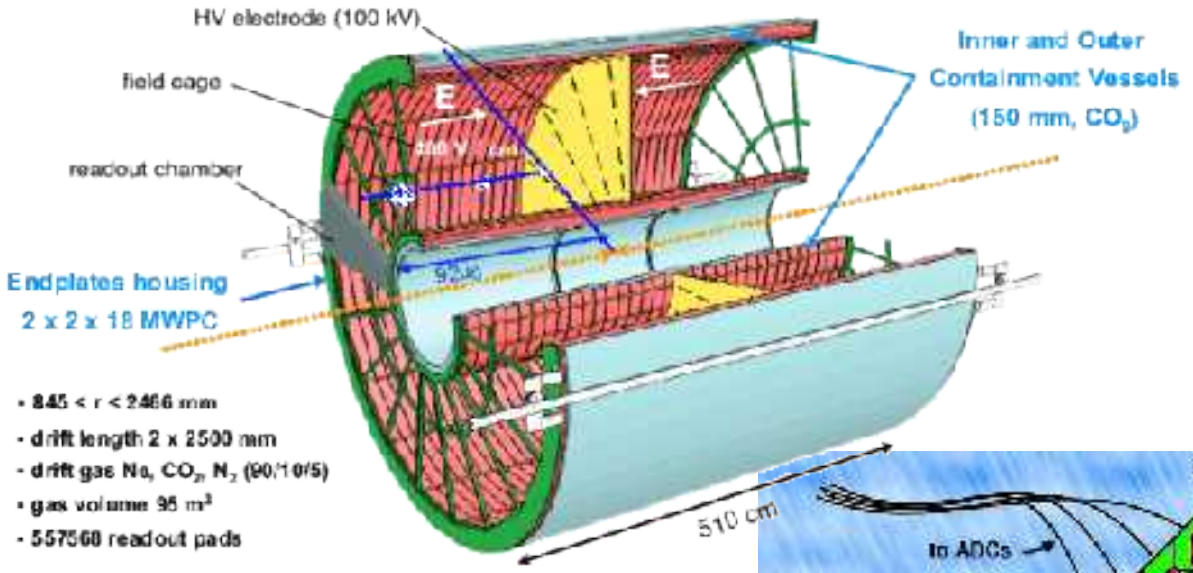
- Beginning of an ALICE ***TPC*** (Time Projection Chamber) event (total size is roughly 60 Mbyte)
- ***Yellow***, ***green*** and ***magenta*** highlighted parts represent data headers generated by different levels of hardware within the DAQ chain (***event*** / ***equipment*** / ***hardware*** headers)
- The rest is ***raw data*** generated by the TPC
- A TPC "***word***" is also analyzed below (a decoding example):
  - A phrase composed of 4 words of ***10 bits***
  - A phrase composed of 5 words of ***8 bits***
  - A single word of ***40 bits***
  - A phrase composed of 5 words of, from left to right ***4, 8, again 4, 10, and 14 bits*** (we are interested in this level)





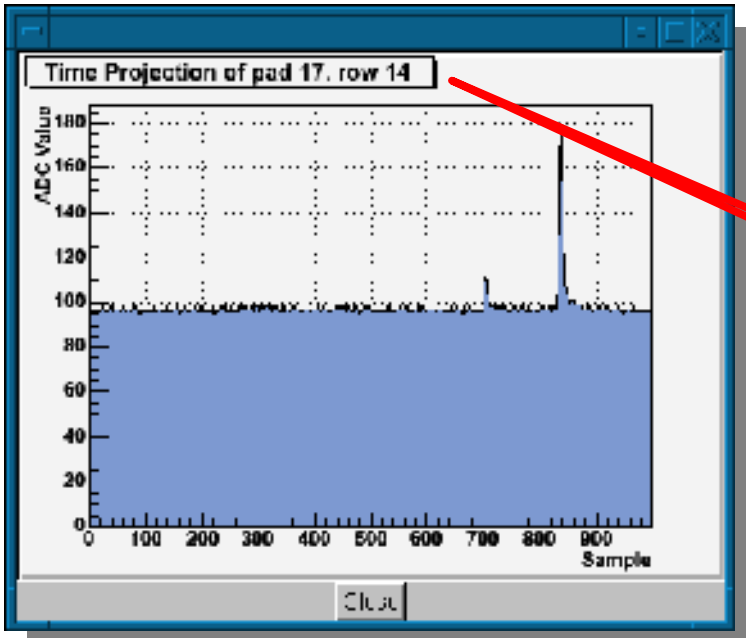
External library usage from within ROOT

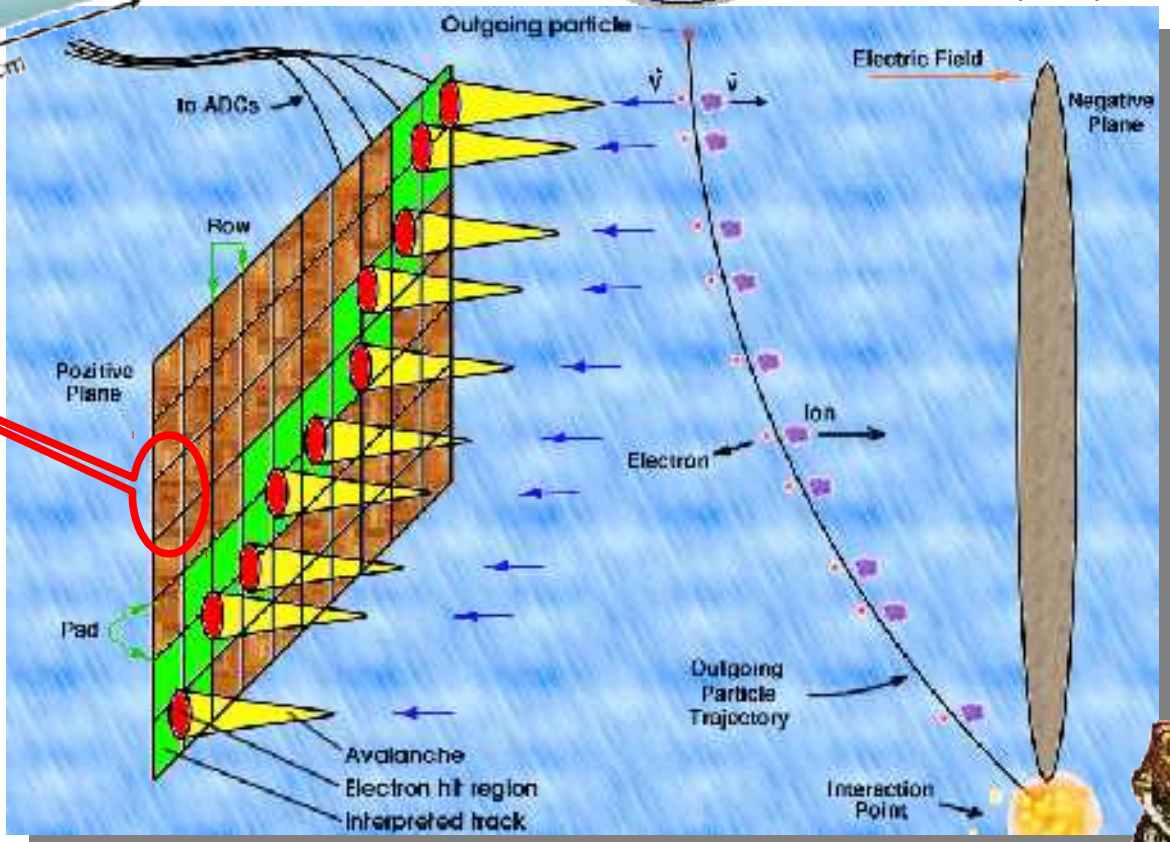
Understanding the detector data



- **For our example, let us choose the Time Projection Chamber detector of ALICE experiment.**

TPC detection principle

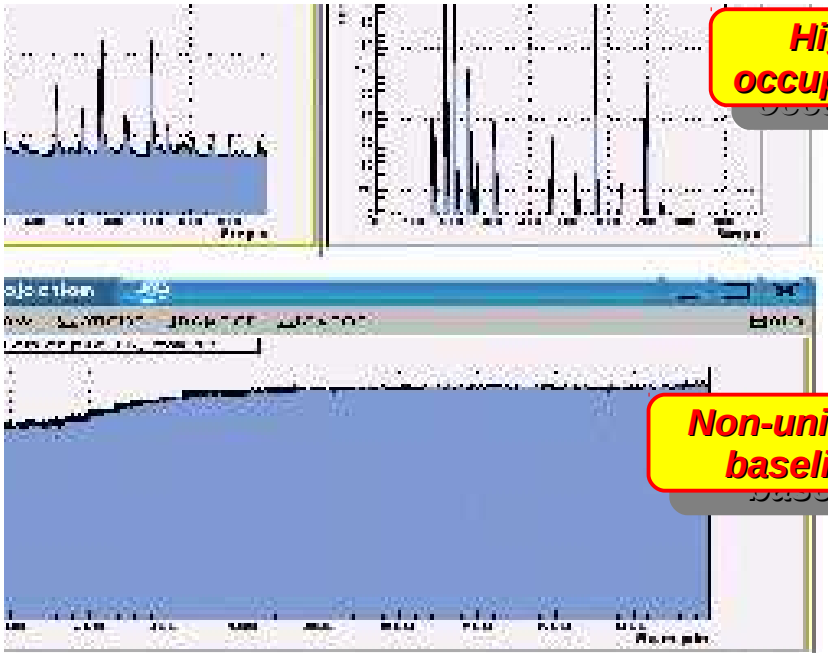




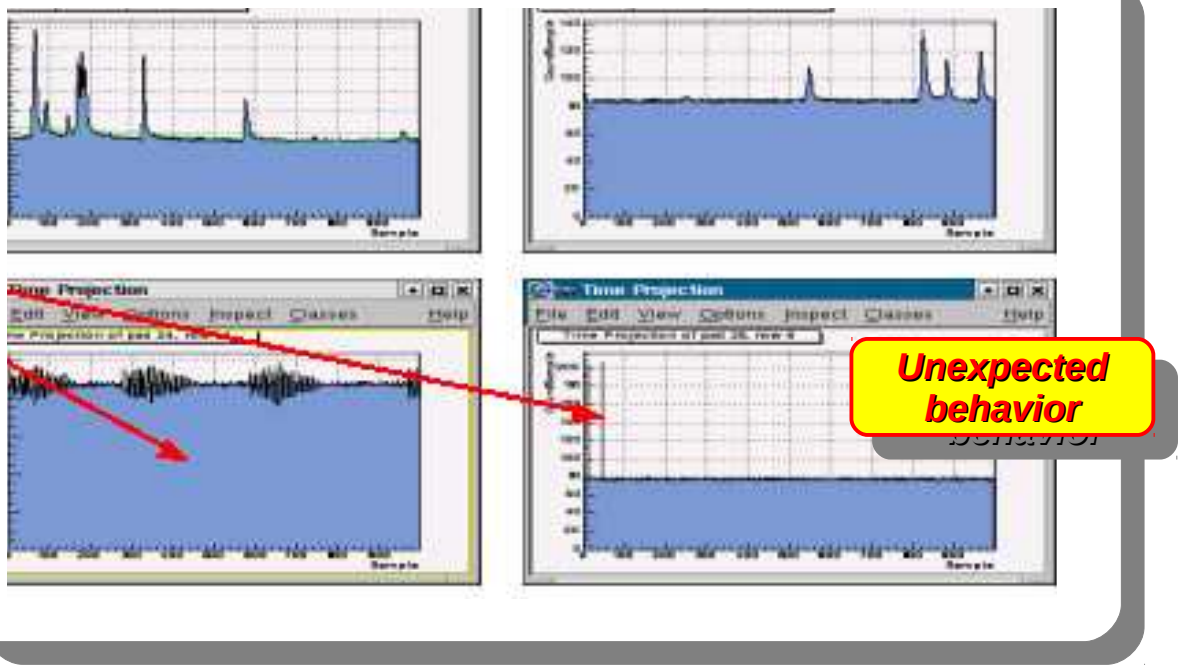
What do we expect to see ?

What errors do we expect to catch via the monitoring tool ?

- Electronic issues
  - Stack-at errors
  - *Non-configurable FE, etc.*
  - *Unexpected behavior*
- *Detector issues*
  - *Gas mixture and dynamics*
  - *Occupancy, etc.*
  - *Unexpected behavior*
- *General*
  - *Temperature & supply variations*
  - *Peak formation*
  - *Baseline fluctuations, etc.*
  - *Unexpected behavior*



- A channel generating the same value all the time
- Un-familiar condition within the detector



Compiling Standalone

Standalone application

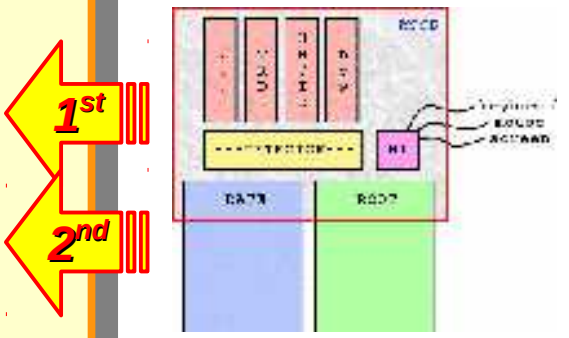
- *An elegant method to write applications is via **Makefile**, but we will not do it here*

- To compile a code similar to the one on the previous page:
  - **g++ -L/date_home -lbase -laz -lgui -thread -lm -lmst -ldynamic -lnonStand -lshift -lcustomDet -lmntrno {...} secondApp.cxx -o secondApp**

- *Remembering all the above things is hard, therefore we will use:*
  - ***date-config**: a command-line tool to make lives of DATE users easy (libraries usually have tools like this one, remember ?)*
  - *It returns appropriate lines needed for compilation*
  - *Usually used in-between " ` ", aka escape symbol*

  - **g++ `date-config --glibs --cflags` secondApp.cxx -o secondApp**

g++
`root-config *--glibs --cflags*`
`date-config *--glibs --cflags*`
thirdApp.cxx -o thirdApp



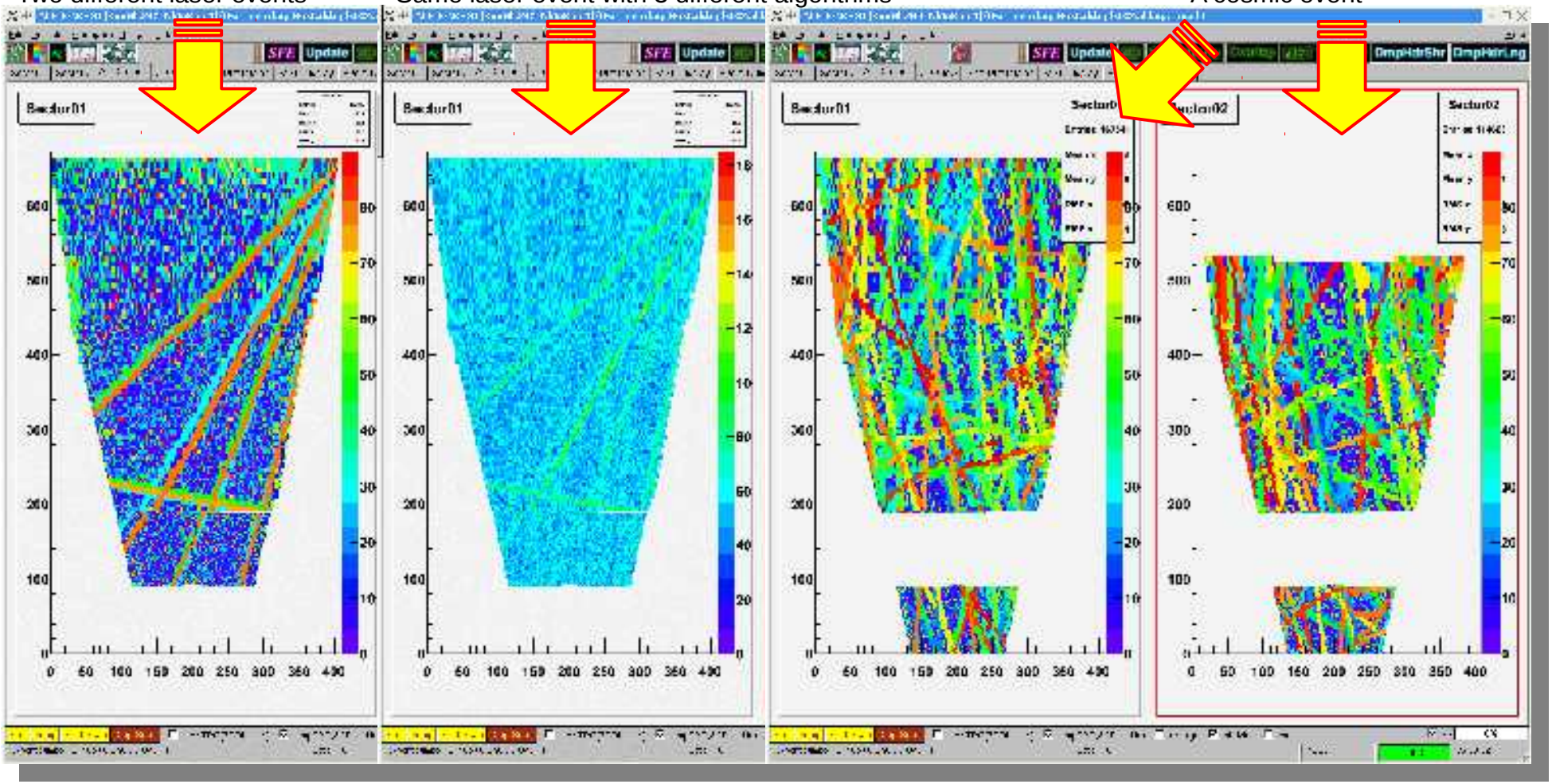
External library usage from within ROOT

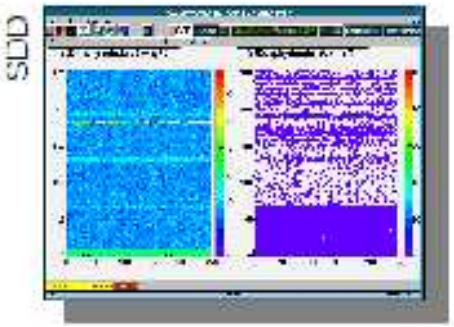
Full featured application being delivered to detector people

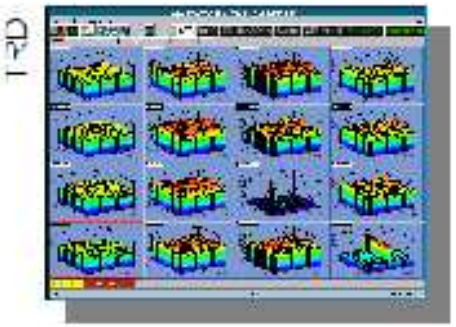
Two different laser events

Same laser event with 3 different algorithms

A cosmic event



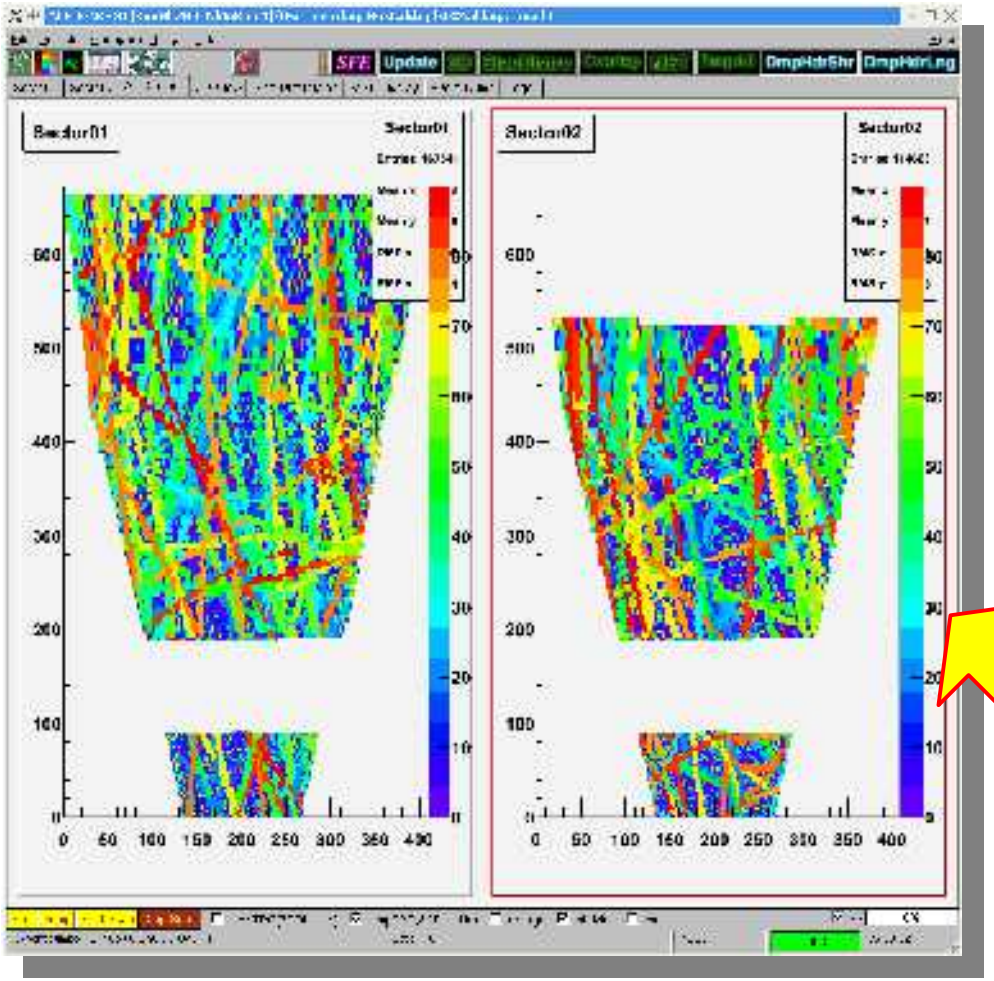


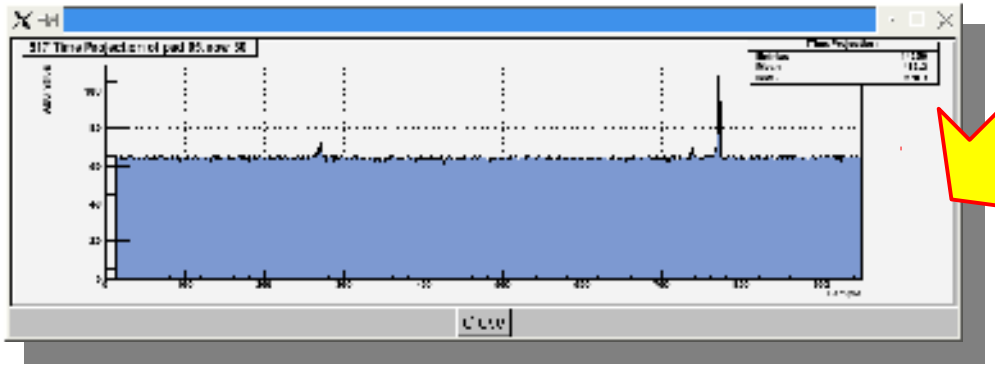


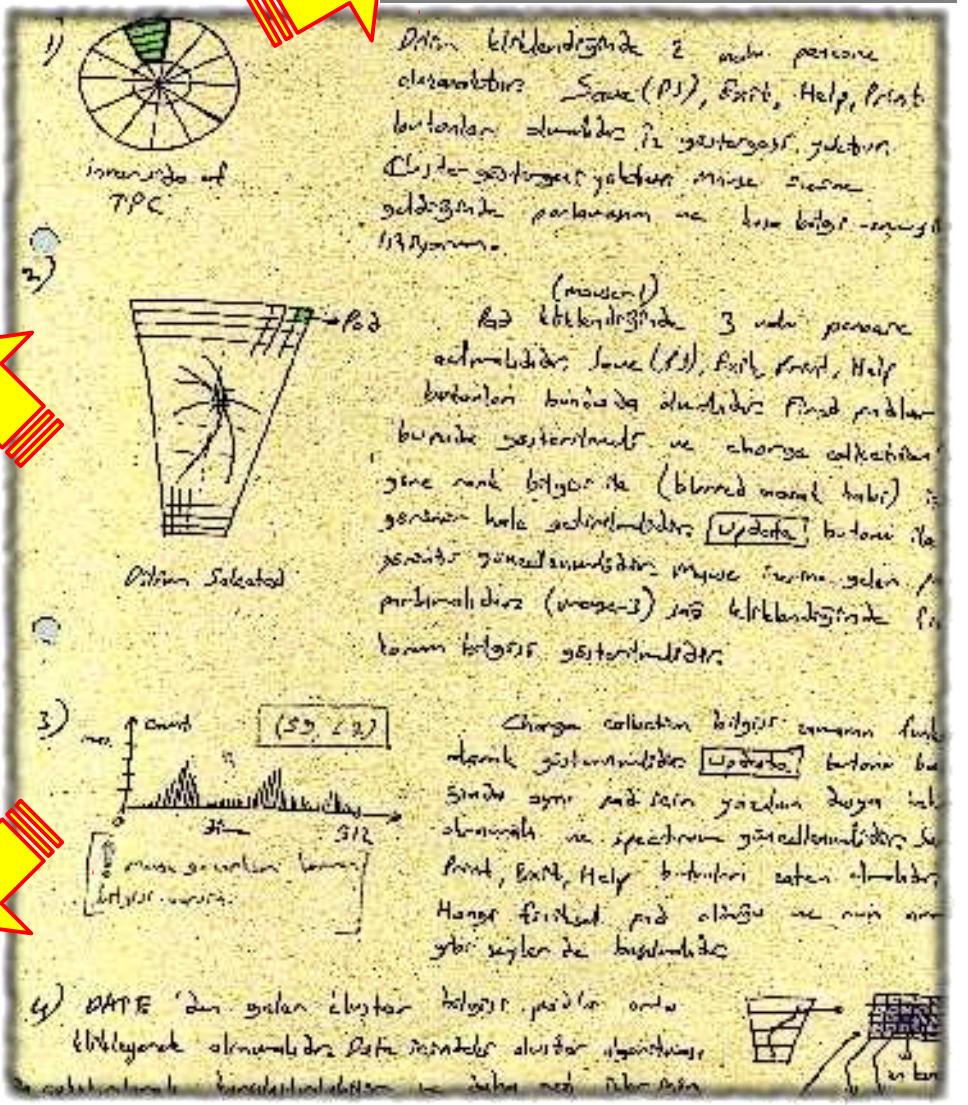
Coding or design ?

Think, design, write, implement and only then start coding

From 2002 To today







ROOT – A brief introduction

Survival with ROOT != Survival at ROOT != Survival despite ROOT

- Introduction to ROOT
  - What is it ?
  - Why is it good ?
- Using ROOT
  - Command line, batch-mode, root console or terminal
  - Scripting/Interpretation
    - Example script comparing two current-mode D/A converter architectures designed for COMPASS
  - Compilation
    - Compiling a script as a " *.so " shared object library
    - Compiling standalone
    - Application development
    - Example standalone application
- GUI of ROOT
  - Human interaction
  - Creating a GUI
- Survival [with/at/despite] ROOT
  - User's guide (refer once)
  - Referring to:
    - $ROOTSYS/tutorials (refer once per problem)
    - $ROOTSYS/test (refer once per problem)
  - HTML source code documentation (refer continuously)
- External library usage from within ROOT
  - DQM of ALICE experiment @ CERN
    - Simplified DAQ operation
  - Understanding the detector data
    - Accessing and decoding data

*