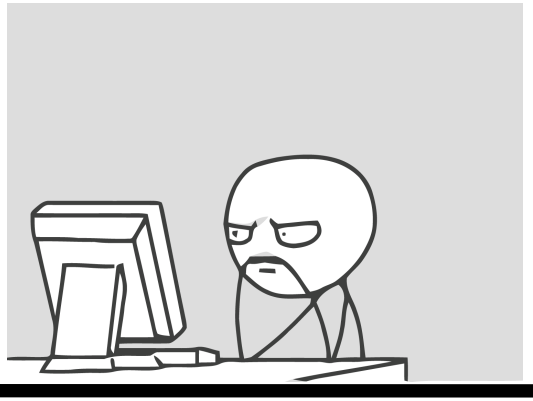
FOOT software framework

A framework for FOOT

- What is and why we need a framework
  - A framework is a library that provides templates and tools for handling some basic software actions, regardless of the actual content and data structure that is being handled.
    - Data I/O. E.g. I want to read either data (decoding the VME output) or MC (taking as input the root files) in a transparent way.
    - "per event" data processing arranged in sequences that are easy customizable by the user. E.g. I want to be able to decide which algorithms are executed in each event and in what sequence
    - Calibration constants and geometry informations (not "event" related), containing the info on the actual experiment and data under study, should be handled/used in an easy way with dedicated templates
- Proposal: ROOT based framework developed @ GSI and used by the FIRST experiment.

What is not covered here

- The basic input production!
  - The framework will handle from an input file onwards. We are not discussing here how to encode the data that are the output of the DAQ nor the MC simulation softwares.
- The framework is completely transparent to the adopted solutions for providing the input data (either exp. or simulation)..The only needed info is the coding of the input information, to be used in the decoding classes that will access the info at run time.
  - Examples / templates of those classes are available in the current version of the framework.



TAG* framework

- Prerequisites:
  - MacOsx or Linux with root.
  - 3D graphics should be enabled (for Event display): if you have installed root by hand, check that the Eve lib is available.
- Basic classes (TAGbase lib):
  - TAGaction (everything that has to be executed for each event)
  - TAGdata* (handles the data types, used for I/O, accessed event by event)
  - TAGpara* (handles the calibration info, loaded during the initialization, available event by event)
  - TAGgeo* (handles the geometry info and transformations of reference systems)
- There are a lot more classes that are used to do "ROOT" related stuff: reading/writing, create 3D event display, etc etc… The framework has a lot of tools developed for fragmentations studies that could be re-used / adapted for our needs….

The reconstruction organization

- The proposed scheme (fully adaptable, changeable according to other ideas, proposals) foresee 2 levels:
  - L0: sub-detector level. Each detector of FOOT ("trigger" and "run info" included) decodes its own data, providing a set of "finite" objects:
    - E.g. beam monitor: read the info from the input files, provides a set of "fired cells, total number of hits reconstructed, track candidates with attached χ^2"
    - **This step has to be properly configured to deal with data and MC input files in different ways!**
  - HL: global event reconstruction level. The global reconstruction algorithms start from the building blocks assembled @ L0 and provide a final full reconstruction of the event.
    - E.g. reco chain: tracks from BM, tracks from VTX and trackers, hits from scintillator and calo are combined using the mag field info to build a list of "global track candidates".
    - **This step is completely transparent to the event types (data or MC).**

L0 detector templates

- There are few templates for the L0 step that have been "imported" from the FIRST experiment, **and hence have to be properly adapted**.
- Ex. the beam monitor
  - Actions [and the related TAGdata classes] to decode and tuple the raw data (DatRaw), the MC info (NtuMC), the candidate tracks (NtuTrack)
  - The output structure (Data) of the decoding is the SAME for data and MC (it is the input for the HL reco!)
  - The par* classes providing the necessary calibration and geometrical info needed for the decoding of the raw information (e.g. ST relations, positioning of the wires in absolute space, map of active cells,……)

TABMbase/TABMactDatRaw.cxx
TABMbase/TABMactNtuMC.cxx
TABMbase/TABMactNtuRaw.cxx
TABMbase/TABMactNtuTrack.cxx

TABMbase/TABMdatRaw.cxx
TABMbase/TABMntuRaw.cxx
TABMbase/TABMntuTrack.cxx

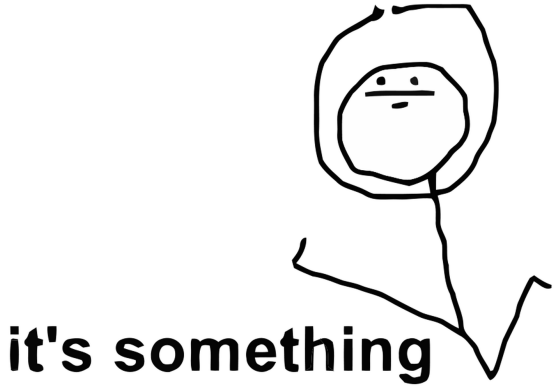
TABMbase/TABMparCon.cxx
TABMbase/TABMparGeo.cxx
TABMbase/TABMparMap.cxx

TABMbase/TABMvieTrackFIRST.cxx

Viewer class to be adapted from FIRST

L0 detectors II

- Other already implemented classes, to be adapted are handling:
  - The SC
  - The vex detector
  - The magnet
  - The trigger info
- Templates for scintillator and calo detectors could be easily inferred from the already existing classes.



HL methods and classes

- The high level classes templates are already available in the TAGfoot folder but are a simple porting of the FIRST code… Of course any attempt to combine at high level the basic building blocks should proceed first trough a complete revision and implementation of the actual detectors and relative positioning in the FOOT reference frame.
- However, just to give an overview of what is available:

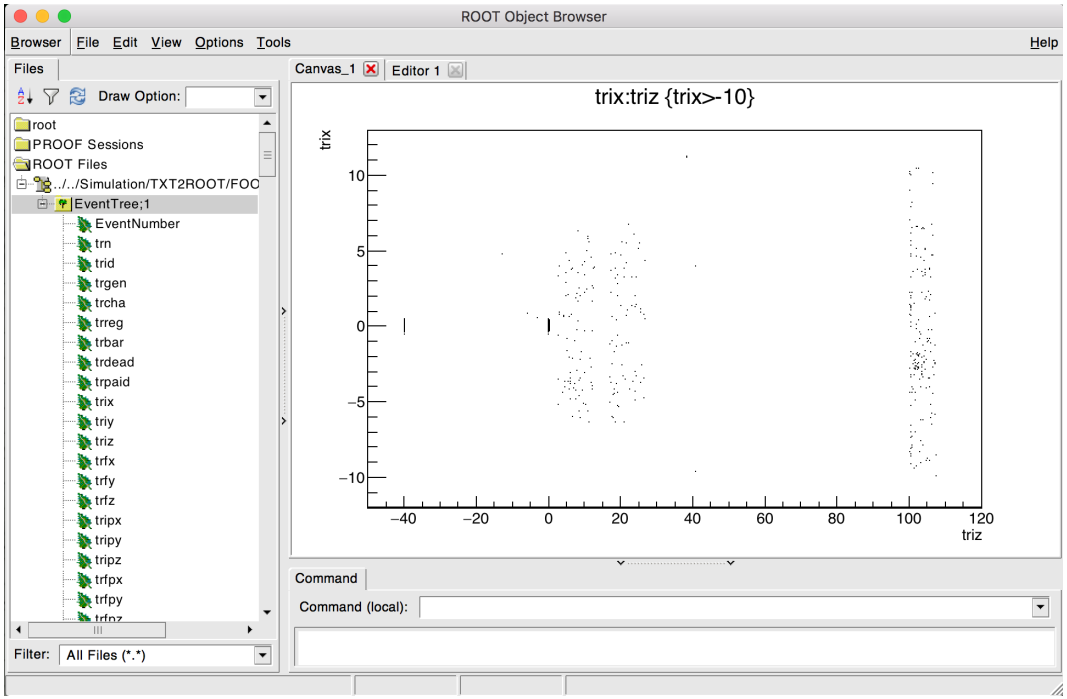
Classes for Fragment ID (ChargBetheBlock), Event display (EventDispl*; TAGVie*), Tracking algorithms (GlobalTrack* ; TA*GlbTrack*; Tracking, Trackable*)

There's also a class for handling the Mag field: MagneticField… Will handle the FOOT magnetic field for the tracking…

```
ChargeBetheBlock.cxx
EventDisplay.cxx
GlbEventDisplay.cxx
GlobalTrack.cxx
GlobalTrackCandidate.cxx
MagneticField.cxx
Makefile
TADIparGeo.cxx
TAGactGlbTracking.cxx
TAGntuGlbTracks.LinkDef.h
TAGntuGlbTracks.cxx
TAGvieHorzFIRST.cxx
TAGvieHorzIR.cxx
TAGvieHorzMCIR.cxx
TAGvieHorzTracks.cxx
TrackableParticle.cxx
Tracking.cxx
TrkStructures.h
```

Reconstruction example

- A real example (already available in git, for details see later)
  1. Define the input and output files.



INPUT

```
//  TTree *tree = 0;
TChain *tree = new TChain("EventTree");
for(int ifi=0; ifi<my_files.size(); ifi++) {
  tree->Add(my_files.at(ifi).data());
  cout<<"Adding :: "<<my_files.at(ifi).data() << " file"<<endl;
}

tagr->AddRequiredItem("my_out");
tagr->Print();
if (my_out->Open(m_oustr, "RECREATE")) return;
```

OUTPUT [code provided by TAGbase]

Reconstruction example II

2. Define the data that has to be read and write by each action and any geometry or calibration information that is needed in the processing
3. Define the sequence of actions that has to be performed

Data and action for MC truth block decoding

```
/*Ntupling the general MC event information*/
myn_mceve    = new TAGdataDsc("myn_mceve", new TAGntuMCeve());

new TAGactNtuMCeve("an_mceve", myn_mceve, myStr);

my_out->SetupElementBranch(myn_mceve,     "mceve.");
```

```
/*Ntupling the MC Beam Monitor information*/
myn_bmraw    = new TAGdataDsc("myn_bmraw", new TABMntuRaw());
myp_bmcon  = new TAGparaDsc("myp_bmcon", new TABMparCon());

initBMCon(myp_bmcon);

new TABMactNtuMC("an_bmraw", myn_bmraw, myp_bmcon, myStr);

my_out->SetupElementBranch(myn_bmraw,     "bmrh.");

myp_bmgeo  = new TAGparaDsc("p_bmgeo", new TABMparGeo());

initBMGeo(myp_bmgeo);

myn_bmtrk    = new TAGdataDsc("myn_bmtrk", new TABMntuTrack());

new TABMactNtuTrack("an_bmtrk", myn_bmtrk, myn_bmraw, myp_bmgeo, myp_bmcon);

my_out->SetupElementBranch(myn_bmtrk,     "bmtrk.");
```

Data and action for Beam Monitor decoding: provides hits, cells, tracks….

Reconstruction example III

2. Loop on events starts, and "NextEvent()" is executed to trigger the list of actions for each event!

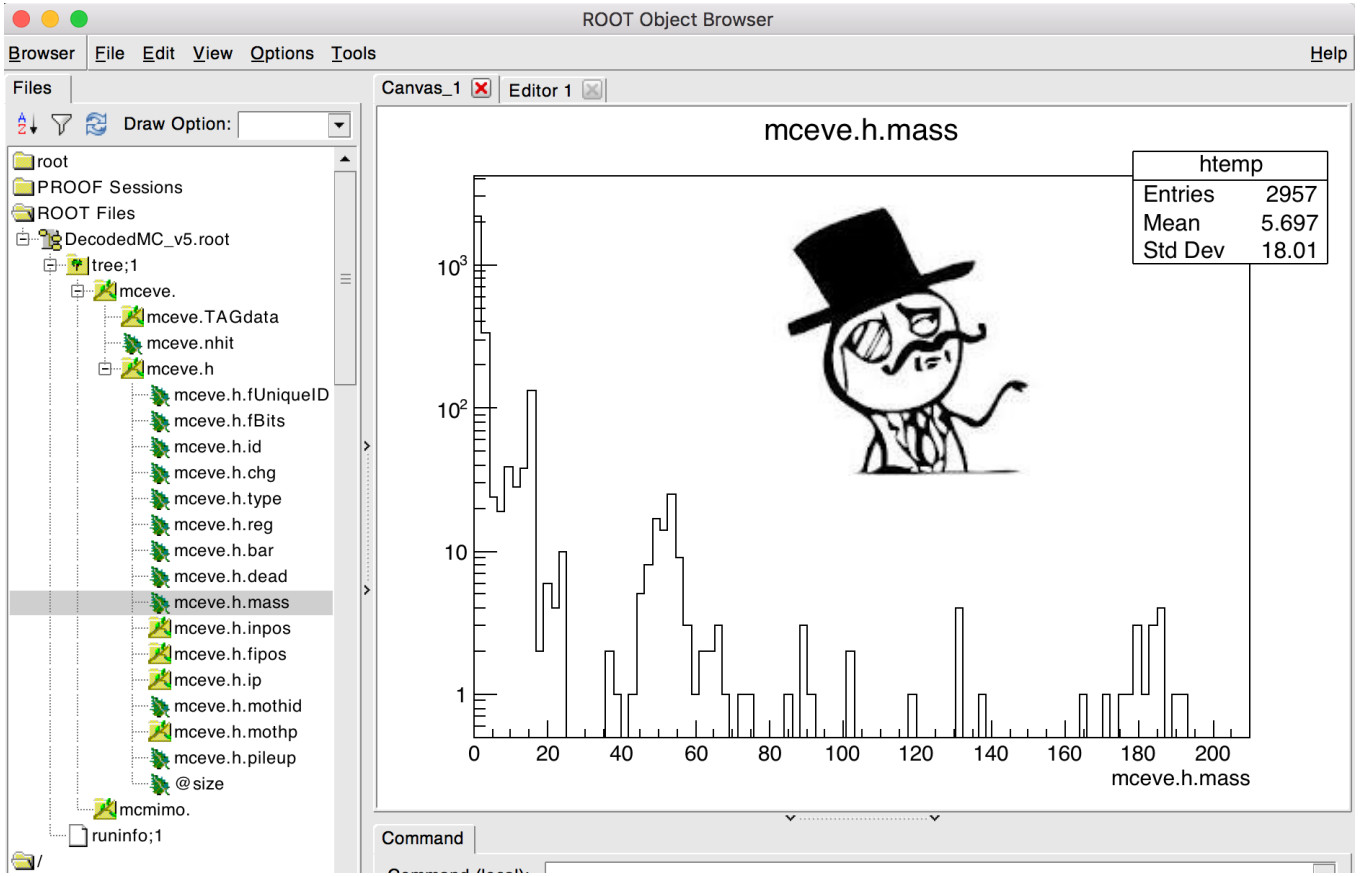
E.g. of an executable in which the info from the MC eve block is decoded and saved…

```
TAGroot:
Known Actions:
  name                type
  actGeoTrafo         TAGgeoTrafo
  my_out              TAGactTreeWriter
  an_mceve            TAGactNtuMCeve
  an_mcmimo           TAGactNtuMCmimo
Known ParaDsc's:
  name                type
Known DataDsc's:
  name                type                    produced by
  myn_mceve           TAGntuMCeve             an_mceve
  myn_mcmimo          TAGntuMCmimo            an_mcmimo
Required Actions:
  my_out              TAGactTreeWriter
```

```
tagr->BeginEventLoop();

Long64_t nentries = tree->GetEntries();

Long64_t nbytes = 0, nb = 0;
char flag[200]; bool tobedrawn = kFALSE;
if(m_nev != 0) nentries = m_nev;

if(m_debug) cout<<" Starting Event Loop "<<endl;

for (Long64_t jentry=0; jentry<nentries;jentry++) {
  if(m_debug) cout<<" New Eve "<<endl;
  nb = tree->GetEntry(jentry);   nbytes += nb;
  // if (Cut(ientry) < 0) continue;

  tagr->NextEvent();
```

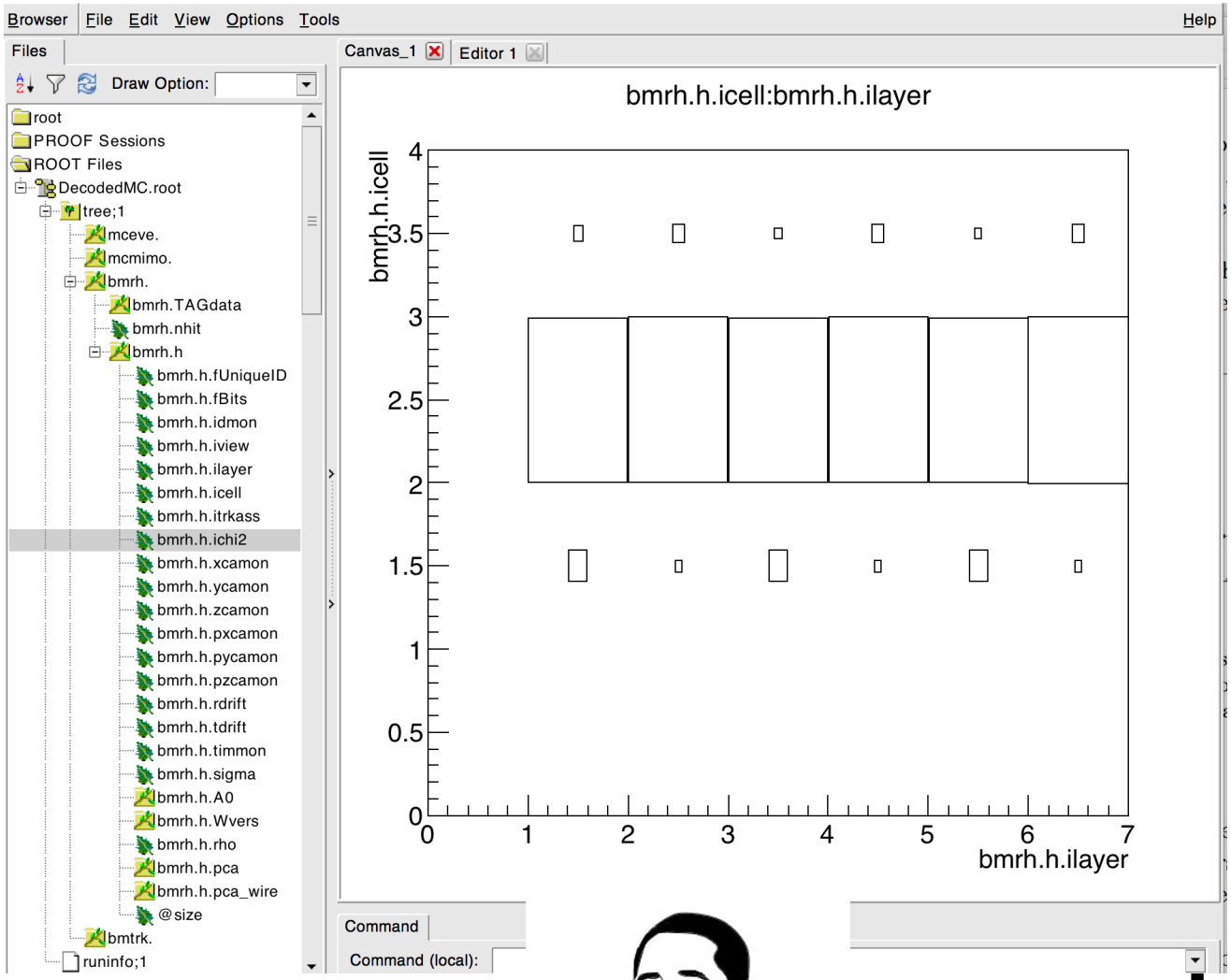
Example: reading MC info

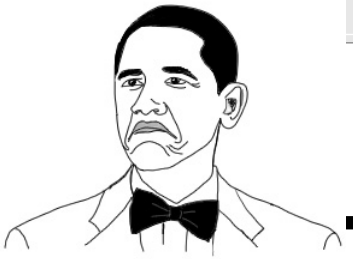
- Code is available under the Reconstruction/level0 folder
- The executable (RecoL0.cc) makes use of RecoTools.{cc,h} libraries
- Takes in input the MC block and produces an output class for use in hlreco or analysis studies.
- First/quick porting from FIRST code: still plenty of "FIRST" relics have to be cleaned up!!! It is just an example of how to play with MC events using the framework!



The BM example

- For now it is a mere tupling of the MC truth info… no "ST" relation calibration is implemented nor some more elaborated post processing of the info….
  - This will go into the TABMpar* and TABMact* actions ASAP….





NOT BAD

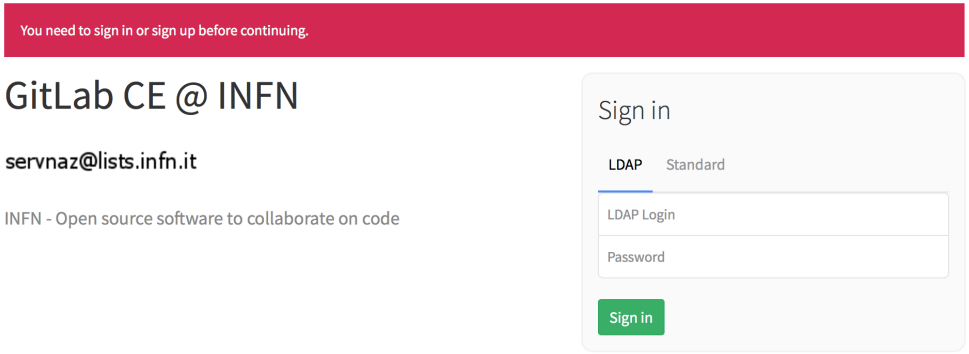
The proposed choice: pros

- What are the advantages of the proposed framework choice
  - ROOT framework is easily portable to different platforms, uses c++ (widely diffused), highly customizable
  - Several parts of the code can be reused (SC, BM, part of vtx detector, trigger class) + geometry , calibration classes… This provides a lot of templates for people that have to start from scratch (calo, scintillator, new trackers…)
  - HL reconstruction implementing forward tracking in mag field is available! Has to be adapted to FOOT, of course.. Has to be retuned, redesigned and needs to include Kalman filtering, but some basic starting structure is in place, as well as the VTX tracking trough Hough transforms and the particle ID of the fragments trough the DeltaE/ToF ID tools.
- **Of course ANY other options is welcomed and there's no prejudice against any other proposal.**

(A proposal for)

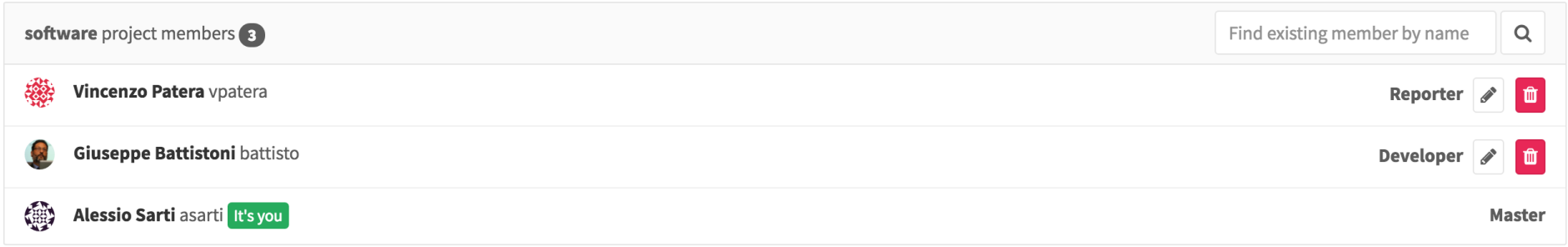
Sharing the code

- While a first set of templates and tools is available, the work to implement the framework in the FOOT landscape **has to be done from scratch** for most of the code/algorithms and foresees the contribution from a lot of different peoples/experts.
- How to organize the parallel in the most efficient way?
  - Solution proposed: share the code using git @ INFN.
- GIT @ INFN guarantees:
  - Fully backupped and online service for code access anytime.
  - Easy handling of the rights to commit/read/contribute to the code trough the INFN interface: all the infn users can access the baltig portal [https://baltig.infn.it/] to register with their credentials.



Handling the users…

- Baltig provides
  - Web interface or command line interface..
- Started the baltig "software" project with 3 users:
  - Reporter (can download and compile/run the code)
  - Developer (Reporter + can commit to non protected branches)
  - Master (Developer + can commit to protected branches)



An agreement on how to proceed with code releases / branches creation will have to be taken in order to allow the collectivity to have always a stable, running version to be used for code testing and developments…

Documenting the code

- Proposed solution: twiki pages.
  - Started a FOOT software project under twiki server @ roma1. http://arpg-serv.ing2.uniroma1.it/twiki/bin/view/Main/FOOTSoftware
    - This page documents how to get and install the code.
    - Few tips on how to deal with "git" are given.
  - Anyone can register and help maintaining the code!
  - For now two pages have been setup:
    - Reconstruction: holds the documentation on how to process data and MC events http://arpg-serv.ing2.uniroma1.it/twiki/bin/view/Main/FOOTReconstruction
    - Simulation: holds the documentation on how to decode the MC events. http://arpg-serv.ing2.uniroma1.it/twiki/bin/view/Main/FOOTSimulation
  - Additional pages could be easily implemented for specific needs of subdetectors….

Disclaimer….

- When switching from ROOT 5 to ROOT 6 some problems were encountered..
  - https://root.cern.ch/phpBB3/viewtopic.php?f=3&t=21981
  - Not everything has been re-checked… Some classes still implements the "buggy" ROOT v5 streamer definition… : will go trough all the instances as long as people start using the code and validate it….