The ROOT file of Simulation Output: a short How-To

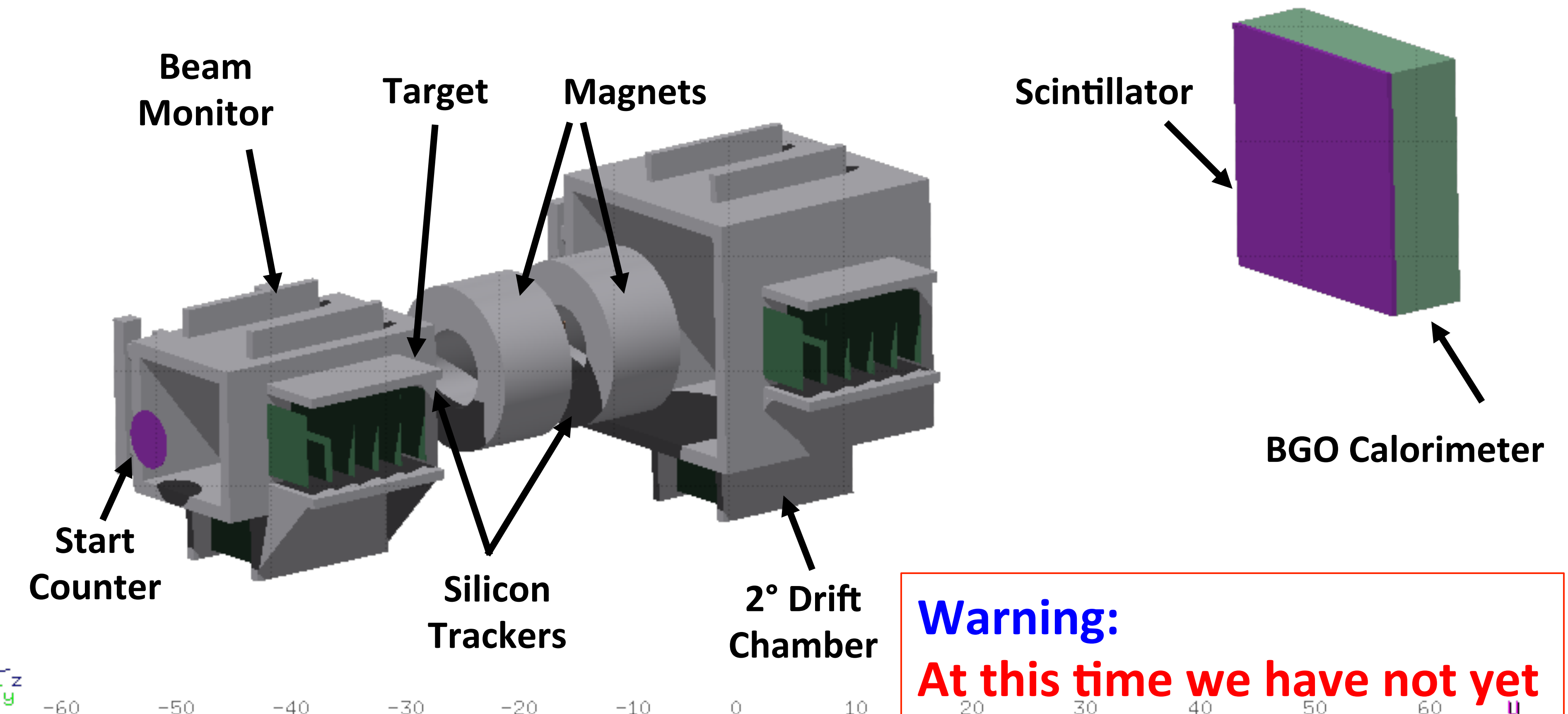
Milano & Roma 1

Introduction

This (very) short tutorial is meant to explain how to use the MC data output produced for FOOT. The main topics are:

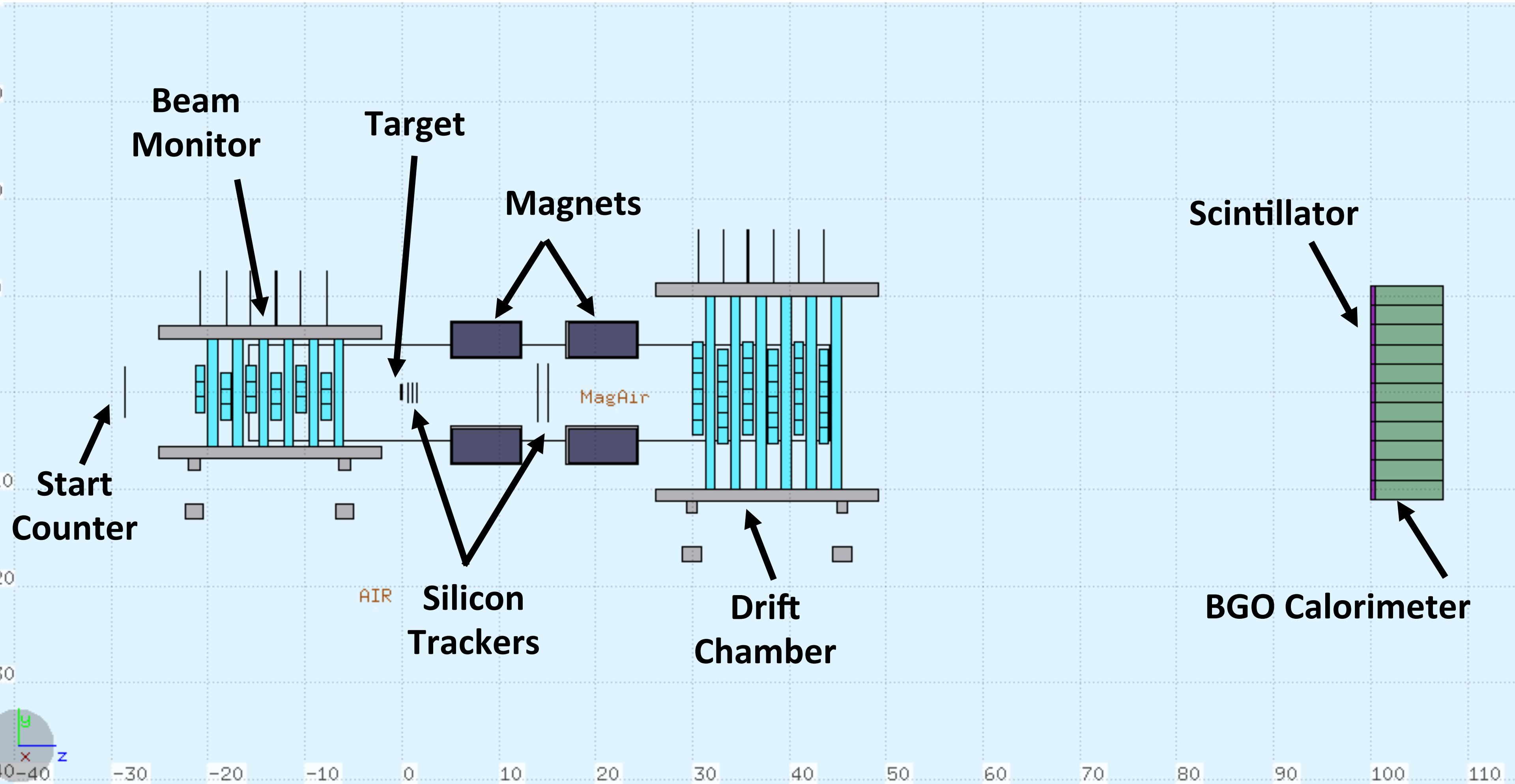
- The structure of the root data produced by MC
- A skeleton code/macro to perform a minimal analysis on MC data: AnaFOOT.h, AnaFOOT.cpp *(more professional and complete code to be developed within the general framework software)*
- Give some basic infos specific of MC simulation (FLUKA) that everybody needs to know

The setup in simulation (at this time)

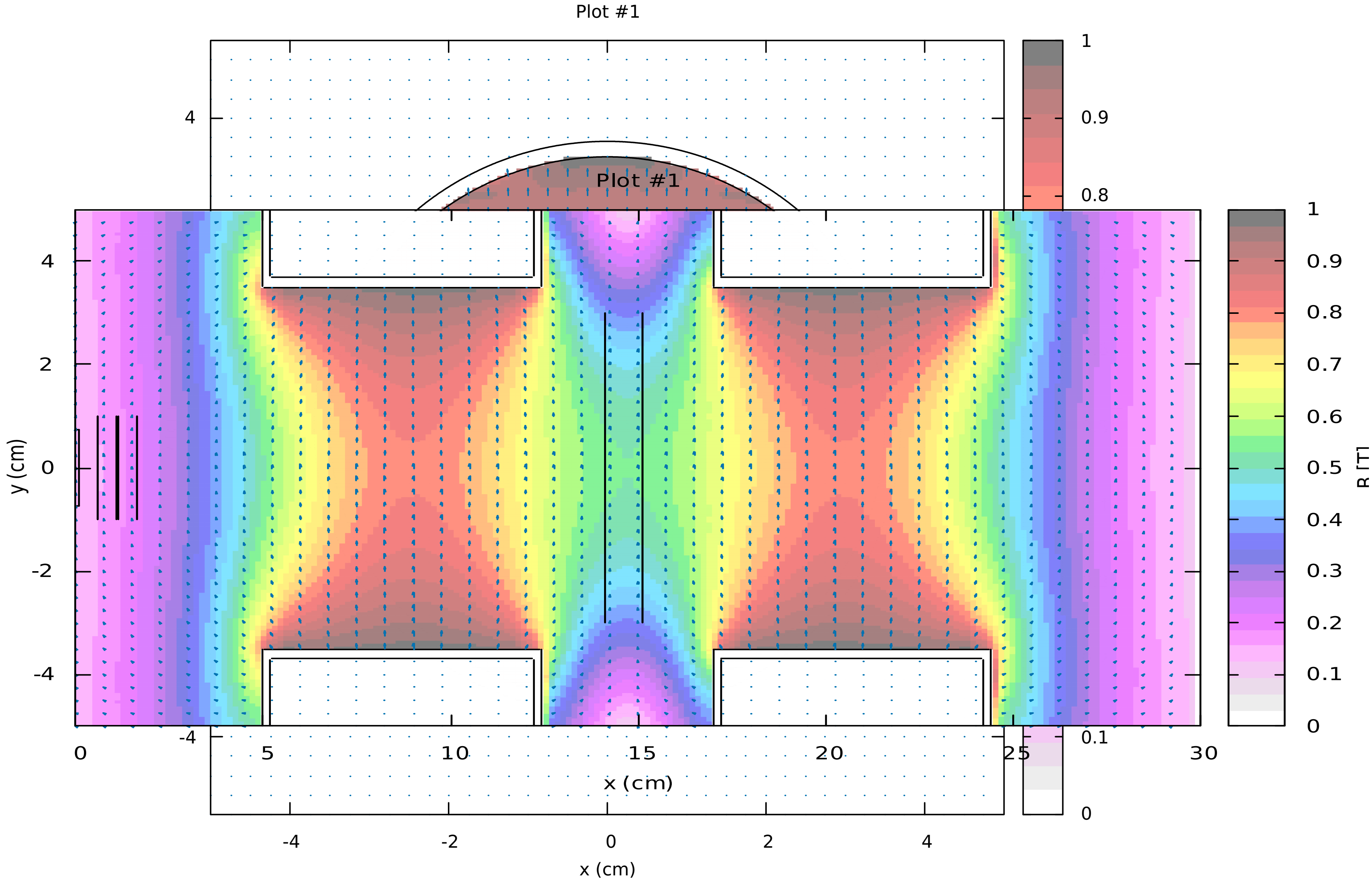


Warning:
At this time we have not yet included the ECC setup which however is ready

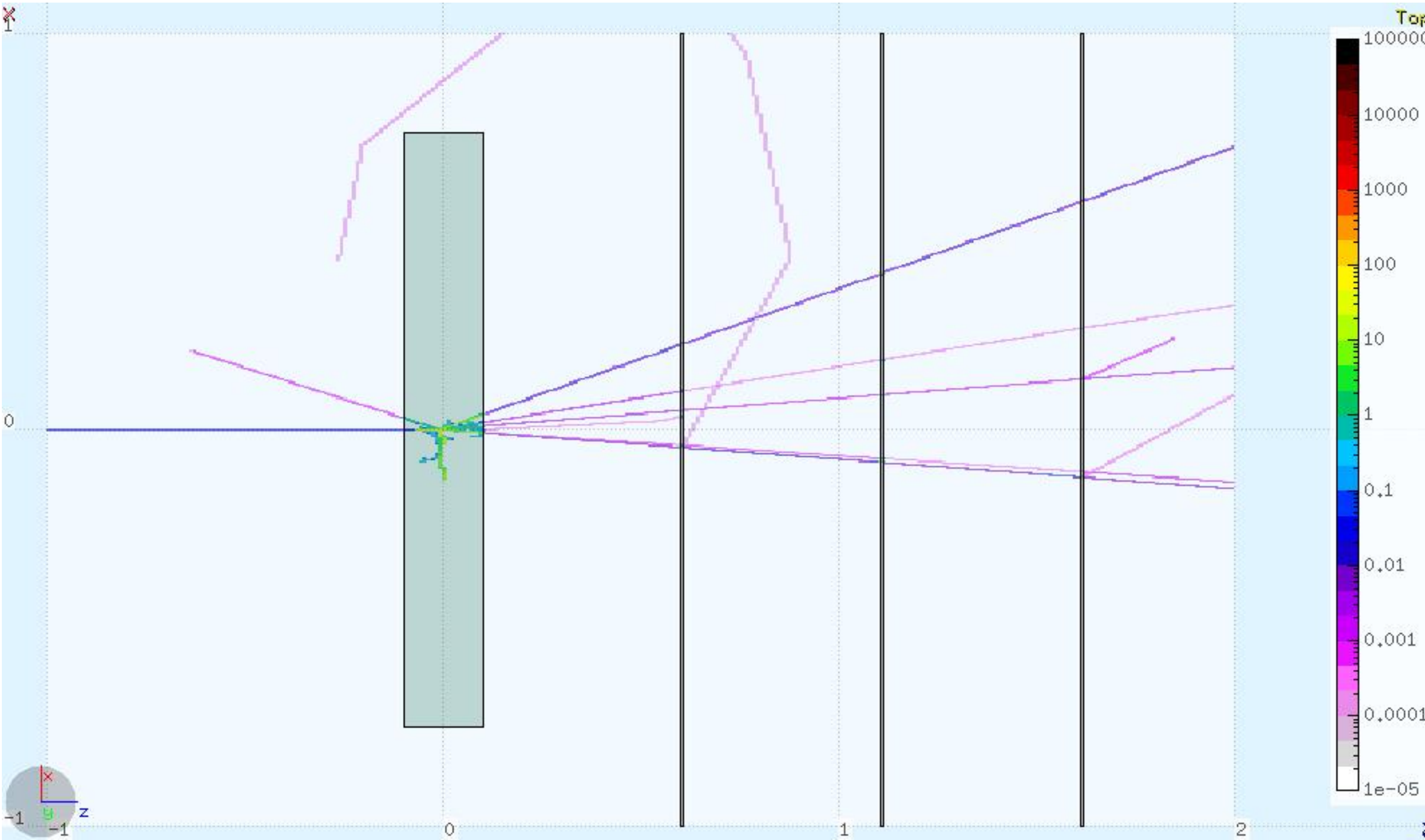
The setup in simulation (at this time)



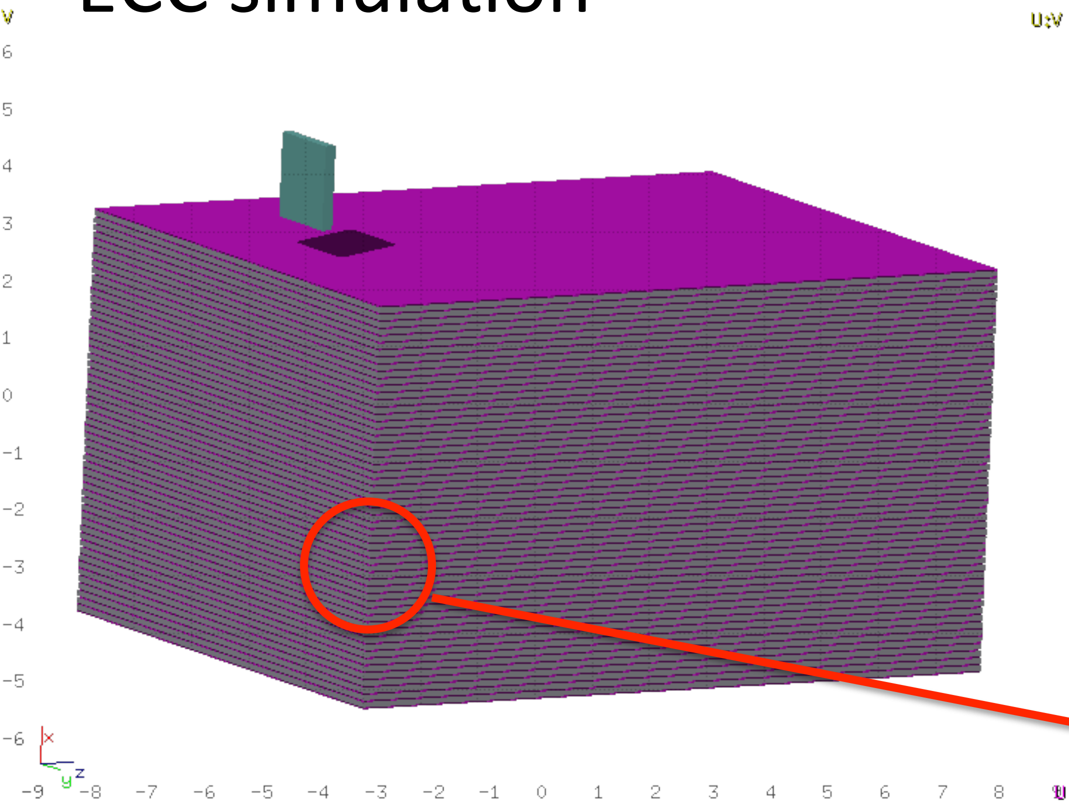
The B-field map for tracking

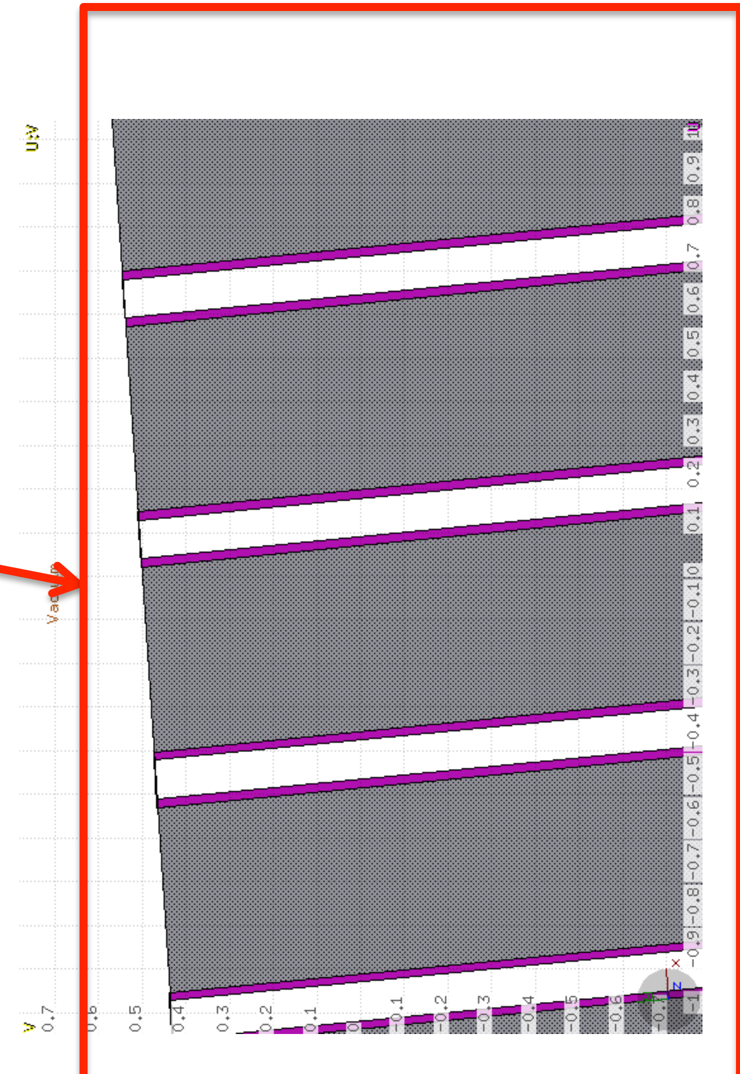


Example of one interesting fragmentation event in the target



ECC simulation



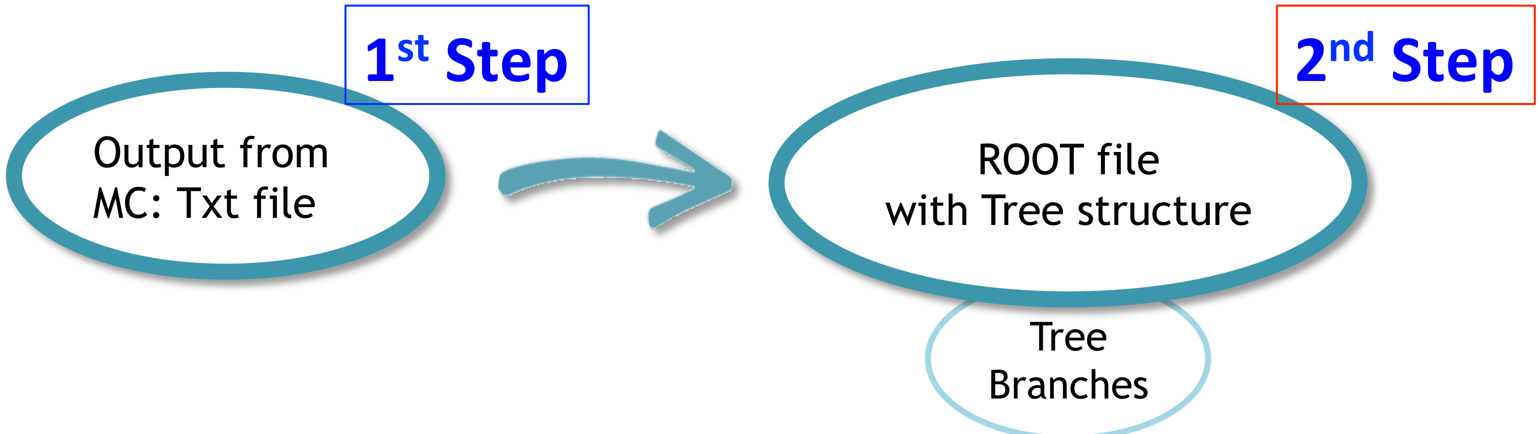


Separate setup
Output to be defined...
Not considered today

Building our taylored MC Output

We have configured some user routines of FLUKA to produce an "ad hoc" event-by-event output written as an ASCII file (*TXT.dat)

Those ASCII files contain information about all the particles and interaction simulated. A simple and portable code reads these txt's and outputs ROOT files



The first goal of this meeting is to start to get familiar with the Tree structure of our output. A simple macro will be used to perform some exercises

The second goal is to allow people to start working seriously on event reconstruction by means of the proposed framework software for both real and simulated data

Example prepared for this meeting

Root file: FOOT_EMFon*.root

500K events of ^{16}O on C_2H_4 target

(*only events with inelastic interaction in the target where written on output, for compactness)*

Simple macro for root: **AnaFOOT** (compiled)

A few things specific of FLUKA MC that are useful to know

Default units

the most important are:

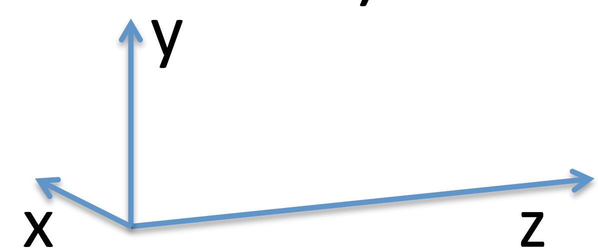
time → s, length → cm, energy → GeV,

masses → GeV/c^2

Reference frame: (cartesian, right-handed)

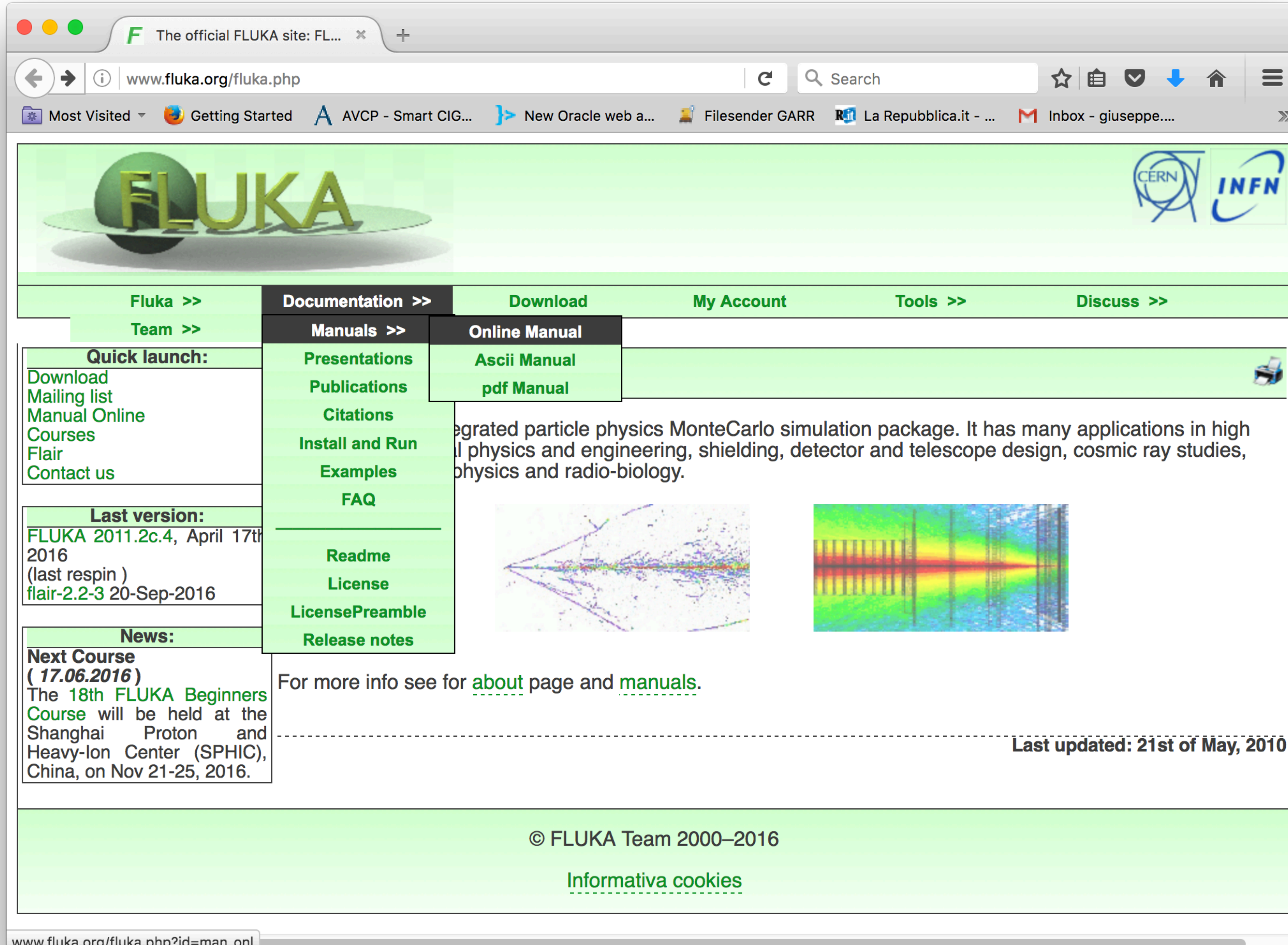
z is primary beam direction

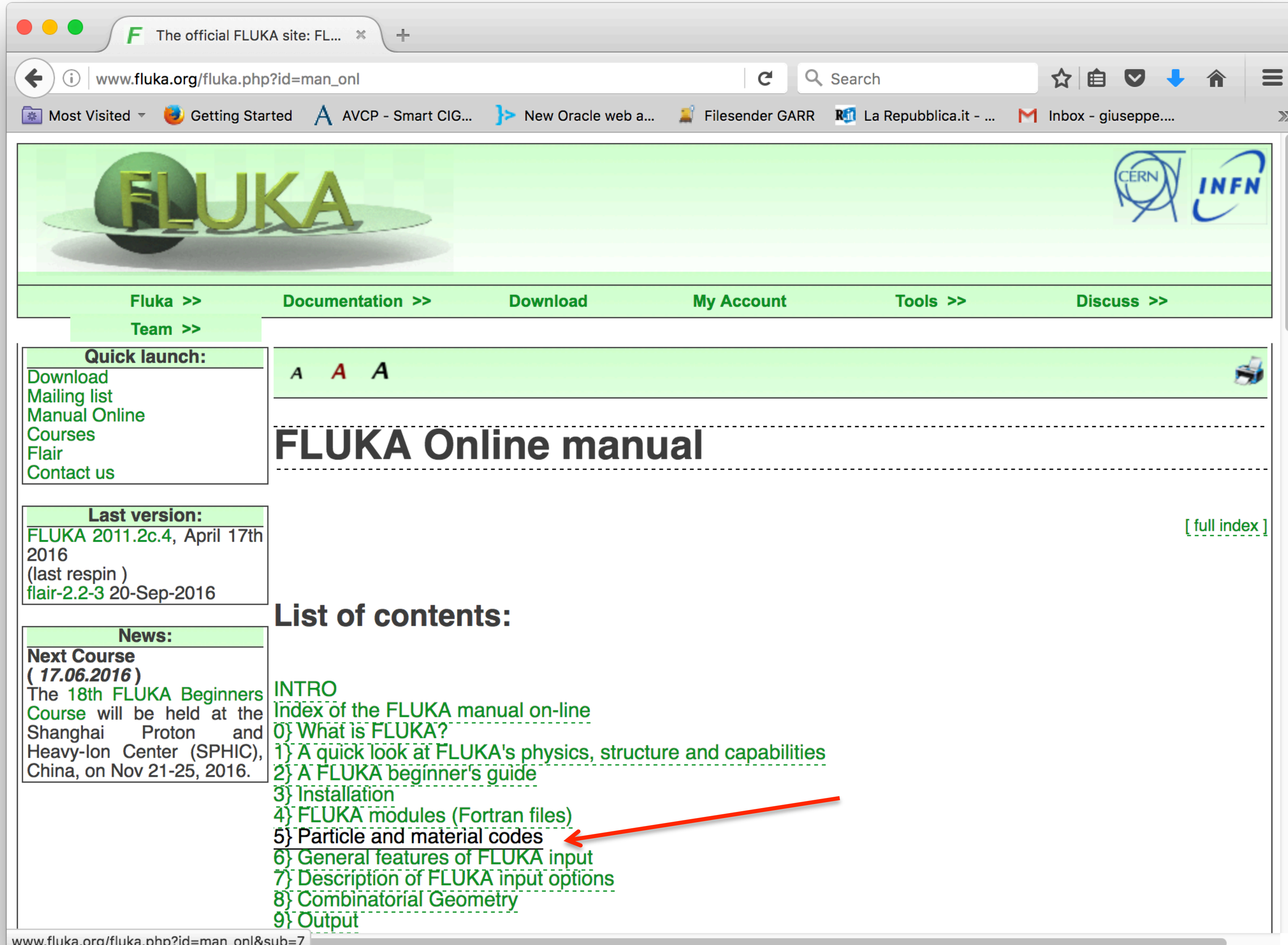
y is pointing upwards



Particles:

each particle is identified by a number


The official FLUKA site: FL...
www.fluka.org/fluka.php
Search
Most Visited
Getting Started
AVCP - Smart CIG...
New Oracle web a...
Filesender GARR
La Repubblica.it - ...
Inbox - giuseppe....
FLUKA
CERN
INFN
Fluka >>
Documentation >>
Download
My Account
Tools >>
Discuss >>
Team >>
Manuals >>
Online Manual
Ascii Manual
pdf Manual
Presentations
Publications
Citations
Install and Run
Examples
FAQ
Readme
License
LicensePreamble
Release notes
Quick launch:
Download
Mailing list
Manual Online
Courses
Flair
Contact us
Last version:
FLUKA 2011.2c.4, April 17th 2016
(last respin)
flair-2.2-3 20-Sep-2016
News:
Next Course
(17.06.2016)
The 18th FLUKA Beginners Course will be held at the Shanghai Proton and Heavy-Ion Center (SPHIC), China, on Nov 21-25, 2016.
egrated particle physics MonteCarlo simulation package. It has many applications in high
l physics and engineering, shielding, detector and telescope design, cosmic ray studies,
physics and radio-biology.
For more info see for about page and manuals.
Last updated: 21st of May, 2010
© FLUKA Team 2000–2016
Informativa cookies
www.fluka.org/fluka.php?id=man_onl


The official FLUKA site: FL...
www.fluka.org/fluka.php?id=man_onl
Search
Most Visited
Getting Started
AVCP - Smart CIG...
New Oracle web a...
Filesender GARR
La Repubblica.it - ...
Inbox - giuseppe....
FLUKA
CERN
INFN
Fluka >>
Documentation >>
Download
My Account
Tools >>
Discuss >>
Team >>
Quick launch:
Download
Mailing list
Manual Online
Courses
Flair
Contact us
Last version:
FLUKA 2011.2c.4, April 17th 2016
(last respin)
flair-2.2-3 20-Sep-2016
News:
Next Course
(17.06.2016)
The 18th FLUKA Beginners Course will be held at the Shanghai Proton and Heavy-Ion Center (SPHIC), China, on Nov 21-25, 2016.
A A A
FLUKA Online manual
[full index]
List of contents:
INTRO
Index of the FLUKA manual on-line
0} What is FLUKA?
1} A quick look at FLUKA's physics, structure and capabilities
2} A FLUKA beginner's guide
3} Installation
4} FLUKA modules (Fortran files)
5} Particle and material codes
6} General features of FLUKA input
7} Description of FLUKA input options
8} Combinatorial Geometry
9} Output
www.fluka.org/fluka.php?id=man_onl&sub=7

FLUKA name	FLUKA number	PDG correspondence	
4-HELIUM (1)	-6	Alpha	---
3-HELIUM (1)	-5	Helium-3	
TRITON (1)	-4	Triton	---
DEUTERON (1)	-3	Deuteron	---
HEAVYION (1)	-2	Generic heavy ion (see command HI-PROPE)	
OPTIPHOT	-1	Optical Photon	---
RAY (2)	0	Pseudoparticle	---
PROTON	1	Proton	2212
APROTON	2	Antiproton	-2212
ELECTRON	3	Electron	11
POSITRON	4	Positron	-11
NEUTRIE	5	Electron Neutrino	12
ANEUTRIE	6	Electron Antineutrino	-12
PHOTON	7	Photon	22
NEUTRON	8	Neutron	2112
ANEUTRON	9	Antineutron	-2112
MUON+	10	Positive Muon	-13
MUON-	11	Negative Muon	13
KAONLONG	12	Kaon-zero long	130
PION+	13	Positive Pion	211
PION-	14	Negative Pion	-211
KAON+	15	Positive Kaon	321
KAON-	16	Negative Kaon	-321
LAMBDA	17	Lambda	3122
ALAMBDA	18	Antilambda	-3122
KAONSHRT	19	Kaon zero short	310
SIGMA-	20	Negative Sigma	3112
SIGMA+	21	Positive Sigma	3222
SIGMAZER	22	Sigma-zero	3212
PIZERO	23	Pion-zero	111
KAONZERO	24	Kaon-zero	311
AKAONZER	25	Antikaon-zero	-311

Manual can also be accessed within Flair: press F1

flair manual

5.1} Particles codes

Each particle which can be transported by FLUKA is identified by an alphanumeric name and by an integer number. Negative values of such numerical identifiers are reserved to light and heavy ions, and to optical photons. The value 0 indicates a pseudoparticle RAY, which can be used to scan the geometry. Numbers > 200 designate "families" of particles, grouped according to some common characteristics (all hadrons, or all charged particles, etc.). In FLUKA, they are called Generalised Particles and can be used only for scoring. Various forms of scored energy, transferred momentum, induced activity etc. are also treated as Generalised Particles.

The identifier values are reported in the following Table, together with the corresponding particle numbering scheme of the Particle Data Group [PDG].

Fluka name	Fluka number	Common name	Standard PDG number (Particle Data Group)
4-HELIUM (1)	-6	Alpha	---
3-HELIUM (1)	-5	Helium-3	
TRITON (1)	-4	Triton	---
DEUTERON (1)	-3	Deuteron	---
HEAVYION (1)	-2	Generic heavy ion with Z > 2 (see command HI-PROPE)	
OPTIPHOT	-1	Optical Photon	---
RAY (2)	0	Pseudoparticle	---
PROTON	1	Proton	2212
APROTON	2	Antiproton	-2212
ELECTRON	3	Electron	11
POSITRON	4	Positron	-11
NEUTRIE	5	Electron Neutrino	12
ANEUTRIE	6	Electron Antineutrino	-12
PHOTON	7	Photon	22
NEUTRON	8	Neutron	2112
ANEUTRON	9	Antineutron	-2112
MUON+	10	Positive Muon	-13
MUON-	11	Negative Muon	13
KAONLONG	12	Kaon-zero long	130
PION+	13	Positive Pion	211
PION-	14	Negative Pion	-211
KAON+	15	Positive Kaon	321
KAON-	16	Negative Kaon	-321
LAMBDA	17	Lambda	3122
ALAMBDA	18	Antilambda	-3122
KAONSHRT	19	Kaon zero short	310
SIGMA-	20	Negative Sigma	3112
SIGMA+	21	Positive Sigma	3222

Since we are mostly interested to nuclear fragments, notice that:

for p, n, d, t, ^{3}He, ^{4}He there is a specific FLUKA particle number

For A>4: FLUKA particle numbers is always -2, and nucleus is identified by Z and A

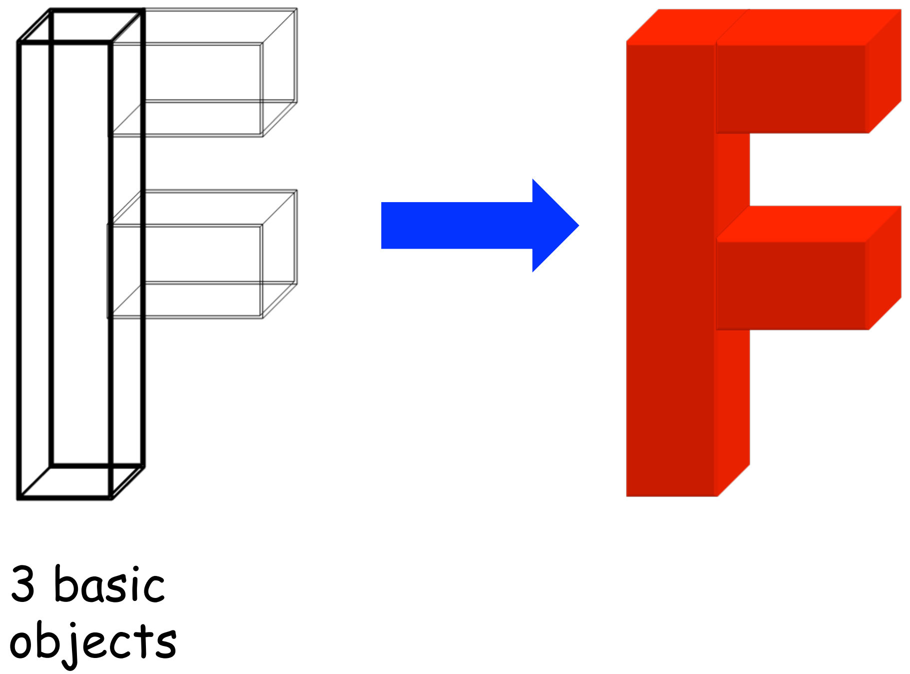
Fragments and nucleons originating in the “nuclear evaporation” phase are identified with particle number in the range from -30 to -7. Again identified by Z and A.

there would be also a way to identify isomers, but we can omit this now

The concept of Region

FLUKA makes use of "Combinatorial Geometry" (originally from Oak Ridge and later extended/modified)

Basic objects called bodies (such as cylinders, spheres, parallelepipeds, etc.) are combined to form more complex objects called **Regions**



1 complex object **= REGION**

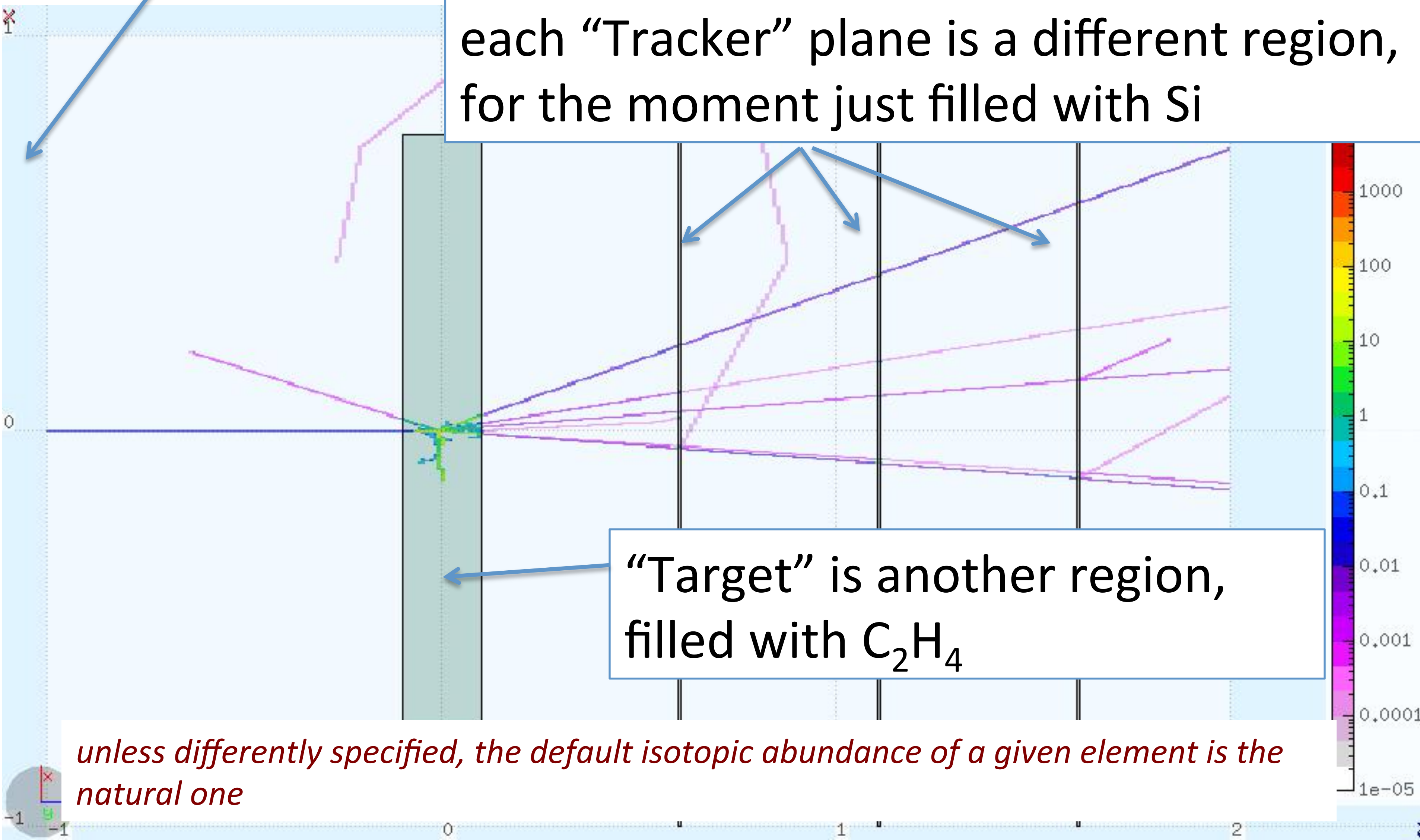
- internally identified by a number
- to each region is assigned a single **Material** (chemical element or compound or mixture)

Example

“Air” is one region, filled with air (N, O, Ar @ STP)

each “Tracker” plane is a different region, for the moment just filled with Si

“Target” is another region, filled with C_2H_4



unless differently specified, the default isotopic abundance of a given element is the natural one

Region numbering in (present) FOOT simulation

Region n.	**2** : **AIR**	**Air all around**
Region n.	**3** : **TARGET**	**Target**
Region n.	**4** : **STARTC**	**Start Counter**
Region n.	**5** : **VTX1**	**The 3 planes of vertex silicon tracker**
Region n.	**6** : **VTX2**	
Region n.	**7** : **VTX3**	
Region n.	**8** : **VTX4**	**The 2 planes of intermediate silicon tracker**
Region n.	**9** : **VTX5**	
Region n	**10-13**:	*Parts of magnet bodies*
Region n.	**14** : **MagAir**	***A special part of air: where there is B-field***
Region n.	**15-32** : **vp*n*c*m*a**	**Beam Monitor Y Cells**
Region n.	**33-50** : **up*n*c*m*a**	**Beam Monitor X Cells**
Region n.	**51-55** :	*External mech. of Beam Monitor*
Region n.	**56-91** : **vp*n*c*m*b**	**2^{nd} Drift Ch. Y Cells**
Region n.	**92-127** : **up*n*c*m*b**	**2^{nd} Drift Ch. X Cells**
Region n.	**128-132** :	*External mech. of 2^{nd} Drift Ch.*
Region n.	**133-253** : **SCIN*nnmm***	**Scintillator tiles (11x11)**
Region n.	**254-374** : **CALO*nnmm***	**Calorimeter crystals (11x11)**

The root data by FLUKA

The data are stored in a root file with several block in the structure EVENTO_STRUCTPIX *(defined in the file EventStructPix.h)*:

- The particle block: kinematics information of the produced particles
- The detector block: information about the detector outputs of the event and namely about energy releases and hits + links to “MC truth”.
- The crossing block: informatioon about the particle that cross different regions of the setup (both inactive and active)

The particle structure

for each of the produced particles we register the info in arrays: i.e. trmass[2] is the mass of the 3rd produced particle

EventNumber = FLUKA event number:

trn= number of particles produced: max equal to

MAXNUMP = 1000

trpaid = index in the part common of the particle parent

trcha = charge

trbar = barionic number

trfid = FLUKA code for the particle (es: photon, jpa=7)

trgen = generation number

trix, triy, triz = production position of the particle

trfx, trfy, trfz = final position of the particle

tripx,tripy,tripz = production momentum of the particle

trifx,trfpy,trfpz = final momentum of the particle

trmass = particle mass

trtime = production time of the particle

trlen = track lenght of the particle

```
Int_t EventNumber;
Int_t trm;
Int_t trpaid[MAXNUMP];
Int_t trgen[MAXNUMP];
Int_t trcha[MAXNUMP];
Int_t trreg[MAXNUMP];
Int_t trbar[MAXNUMP];
Int_t idead[MAXNUMP];
Int_t trfid[MAXNUMP];
Double_t trix[MAXNUMP];
Double_t triyi[MAXNUMP];
Double_t triz[MAXNUMP];
Double_t trfx[MAXNUMP];
Double_t trfy[MAXNUMP];
Double_t trfz[MAXNUMP];
Double_t tripx[MAXNUMP];
Double_t tripy[MAXNUMP];
Double_t tripz[MAXNUMP];
Double_t trfpx[MAXNUMP];
Double_t trfpy[MAXNUMP];
Double_t trfpz[MAXNUMP];
Double_t trmass[MAXNUMP];
Double_t trtime
[MAXNUMP];
Double_t tof[MAXNUMP];
Double_t trlen[MAXNUMP];
```

The individual detectors structures

For each detector with n energy releases the info are stored in arrays (x, p, De, time, etc...) with the i-th component related to the i-th release . Same syntax for all scint detector: "info""NAMEDETECTOR"[index of the release]

DETn = number of energy release in the detector DET

DETid = position of the particle responsible of the release in the particle block

DETinx, DETiny, DETinz = inizial position of energy release

DEToutx, DETouty, DEToutz = final position " "

DETnpx, DETinpy, DETinpz = inizial momentum " "

DEToutpx, DEToutpy, DEToutpz = final momentum " "

DETde = energy release

DET = quenched energy release

DETtime = initial time of the energy release

Start Counter: st

```
Int_t stn;
Int_t stid[MAXSC];
Double_t stinx[MAXSC];
Double_t stiny[MAXSC];
Double_t stinz[MAXSC];
Double_t stoutx[MAXSC];
Double_t stouty[MAXSC];
Double_t stoutz[MAXSC];
Double_t stinpx[MAXSC];
Double_t stinpy[MAXSC];
Double_t stinpz[MAXSC];
Double_t stoutpx[MAXSC];
Double_t stoutpy[MAXSC];
Double_t stoutpz[MAXSC];
Double_t stde[MAXSC];
Double_t stal[MAXSC];
Double_t sttim[MAXSC];


MAXSC = 500
```

Simple case of non-segmented detector

Vertex tracker: vtx

This is instead a segmented (=pixelated) detector
Additional variables are needed

```
Int_t nvtx;
Int_t idvtx[MAXVTX];
Int_t iplavtx[MAXVTX];
Int_t irowvtx[MAXVTX];
Int_t icolvtx[MAXVTX];
Double_t xinvtx[MAXVTX];
Double_t yinvtx[MAXVTX];
Double_t zinvtx[MAXVTX];
Double_t xoutvtx[MAXVTX];
Double_t youtvtx[MAXVTX];
Double_t zoutvtx[MAXVTX];
Double_t pxinvtx[MAXVTX];
Double_t pyinvtx[MAXVTX];
Double_t pzinvtx[MAXVTX];
Double_t pxoutvtx[MAXVTX];
Double_t pyoutvtx[MAXVTX];
Double_t pzoutvtx[MAXVTX];
Double_t devtx[MAXVTX];
Double_t alvtx[MAXVTX];
Double_t timvtx[MAXVTX]; MAXVTX = 500
```

Plane

Row (in a given plane)

Column (in a given plane)

Identify the pixel

Inner tracker: IT

```
Int_t nIT; ... MAXIT = 500
Int_t iplaIT[MAXIT];
Int_t irowIT[MAXIT];
Int_t icolIT[MAXIT];
```

beam monitor (1st drift ch.): mon

```
Int_t nmon; ... MAXBM = 500
Int_t ilayer[MAXBM];   → layer #
Int_t icell[MAXBM]; → cell #
Int_t iview[MAXBM]; → view (0:x 1:y)
```

2nd drift ch.: 2dc

```
Int_t n2dc; ... MAX2DC = 500
Int_t ipla2dc[MAX2DC];
Int_t icell2dc[MAX2DC];
Int_t iview2dc[MAX2DC];
```

scintillator: scint

```
Int_t nscint; ... MAXSCINT = 1000
Int_t  Int_t irowscint[MAXSCINT];
Int_t  icolscint[MAXSCINT];
```

crystal calorimeter: cry

```
Int_t ncry; ...  MAXCRY = 2000
Int_t irowcry[MAXCRY];
Int_t icolcry[MAXCRY];
```

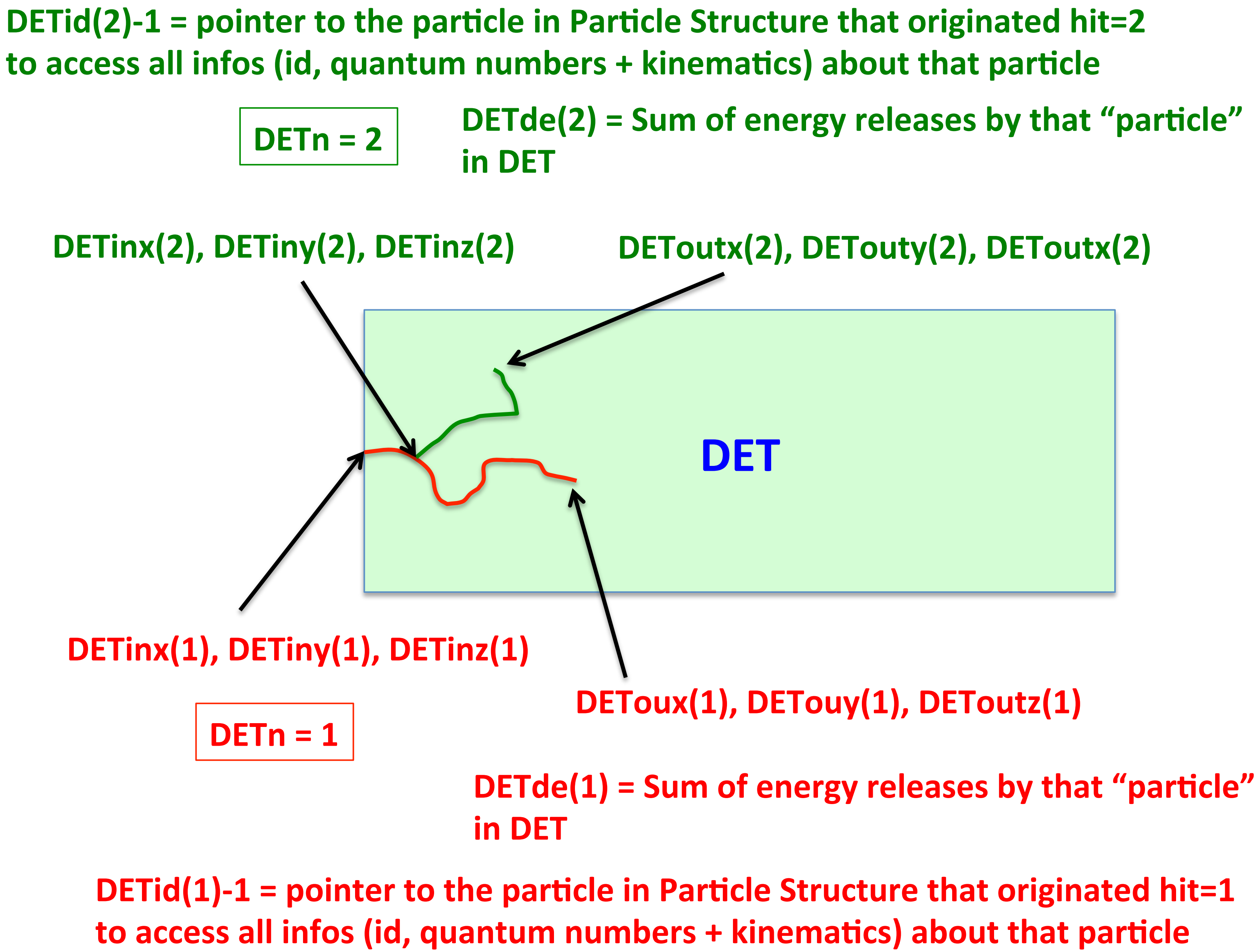
Energy releases and hits connection to particles

To find which particle released the energy of a detector energy release we need to build a pointer to the particle block. Given the j–th energy release in the detector DET, then we build:

```
pointer= DETid[j]–1;
```

Then the features of the particles responsible of the release (for example the mass and the x coord of production) can be retrieved from the particle block as:

```
Massa = trmass[pointer];
Xprod = trix[pointer];
```


DETid(2)-1 = pointer to the particle in Particle Structure that originated hit=2
to access all infos (id, quantum numbers + kinematics) about that particle
DETn = 2
DETde(2) = Sum of energy releases by that “particle”
in DET
DETinx(2), DETiny(2), DETinz(2)
DEToutx(2), DETouty(2), DEToutx(2)
DET
DETinx(1), DETiny(1), DETinz(1)
DETn = 1
DEToux(1), DETouy(1), DEToutz(1)
DETde(1) = Sum of energy releases by that “particle”
in DET
DETid(1)-1 = pointer to the particle in Particle Structure that originated hit=1
to access all infos (id, quantum numbers + kinematics) about that particle

The crossing data structure

This structure registers the info on the particles that cross the boundaries between the different regions of the setup (detector elements, air, target). At each crossing the info are stored in arrays

ncross = number of boundary crossing

idcross = position of the crossing particle in the particle block

nregcross = no. of region in which the particle is entering

nregoldcross = np. of region the particle is leaving

pxcross, pycross, pzcross = momentum at the boundary crossing

xcross, ycross, zcross = position of the boundary crossing

tcross = time of the boundary crossing

chcross = charge of crossing particle

macross = mass of the crossing particle

```
Int_t ncross;
Int_t idcross[MAXCROSS];
Int_t nregcross[MAXCROSS];
Int_t nregoldcross[MAXCROSS];
Double_t xcross[MAXCROSS];
Double_t ycross[MAXCROSS];
Double_t zcross[MAXCROSS];
Double_t pxcross[MAXCROSS];
Double_t pycross[MAXCROSS];
Double_t pzcross[MAXCROSS];
Double_t mcross[MAXCROSS];
Double_t chcross[MAXCROSS];
Double_t tcross[MAXCROSS];
```

MAXCROSS = 10000

The simple code AnaFOOT

The code reads the root data from MC and produces some example histos. It's thought as an example/skeleton for more complex code specific to different analysis.

Compiling & Linking: make –f Makefile_AnaFlukaHIT (clean)

Usage: can be seen typing "AnaFOOT –help" :

> AnaFOOT [opKons] with possible opKons

- nev value : [def=all] Numbers of events to process
- in file : [def=In.root] Root input file generated by FLUKA
- out file : [def=Out.root] Root output file with analysis histo
- iL : [def=none] input file is a list of files
- deb value : [def=0] Enables debugging

N.B To process multiple files make a list of files to be process in a list file (ex: lista.txt) and give the command:

./AnaFOOT –iL –in lista.txt –out TotAnaFile.root

Let's look a little bit inside AnaFOOT.cpp

Just to be ready to make some simple exercise today...

Pointer to structure

```
EVENTOPIX_STRUCT *pevstr = new EVENTOPIX_STRUCT;
TFile *f_out = new TFile(Outname,"RECREATE");
```

Opens output root file

```
status = Booking(f_out);
```

Books histograms.
Booking is the routine where you have to book your custom histos

```
TFile *f_input = new TFile(Inname);

f_input->cd();
f_input->GetObject("EventTree",ptree);
EventoPix *Ev = new EventoPix();
status = Ev->FindBranches(ptree,pevstr);
```

Loads root tree with MC output

```
int currentev = 0;

for (int kk=EvStart; kk<EvStart+Nproc; kk++){
  if(kk%1000==0){ cout<<"Processed event = "<<kk<<endl;}

  status = Ev->Clean();

  currentev = ptree->GetEvent(kk);

  status = Vertex(pevstr);
  status = Calo(pevstr);
  status = Crossings(pevstr);
}
// End of event loop
```

Start event loop

Gets event in tree

invokes some routines which have been prepared to show examples about manipulating variables in Vertex, Calorimeter and Crossing structures

writes and closes output files containing custom histos

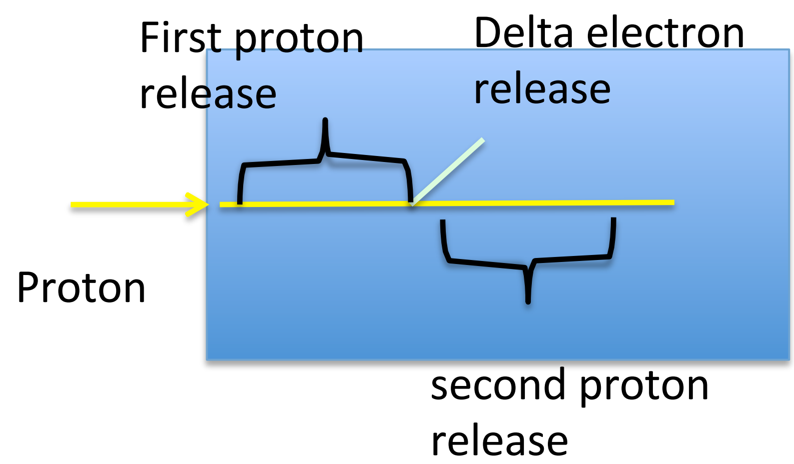
```
f_out->Write();
f_out->Close(); // Delete also histos and trees

return 0;
```

customized code: Example A

I want to retrieve the code and the mass of the particle that release energy in the start counter.
A particle may generate more than a single release in the detector (see the drawing): it’s a good idea to consider the first energy release to retrieve the features of the impinging particle.

```
double  massa;
int ipart_pointer, kpart_type;
If ((pevstr->stn)>0){
      ipart_pointer = pevstr->stid[0]-1;
      massa = pevstr->trmass[ipart_pointer];
      kpart_type = pevstr->trfid[ipart_pointer];
}
```



Example B

I want to know how many neutrons arriving at calorimeter have been generated inside the target

```
int neutron_from_target=0;
int neutron_code = 8;
int target_region = 3;
int first_cry_region = 254;
int last_cry_region = 354;
Int ipart_pointer;
for( int ii=0; ii<pevstr->ncross); ii++){
    if(pevstr->nregcross[ii]>=first_cry_region &&
        pevstr->nregcross[ii]<=last_cry_region ) {
            ipart_pointer = pevstr->idcross[ii]-1;
            if ( (pevstr->trreg[ipart_pointer]==target_region) &&
             (pevstr->trfid[ipart_pointer]==neutron_code) {
                neutron_from_target++;
            }
    }
}
```

loops on all the crossings entering a crystal and cheks if pointer in particle block is coming from target and if is a neutron