Managing FLUKA Simulation Output Files using SHOE

G.B. S.M.

Introduction - 1

This short tutorial is meant to explain how to understand and use the MC data output produced for FOOT *(Electronic Spectrometer)* using SHOE.

The MC simulation is performed using the FLUKA code. This is described in a recent paper:

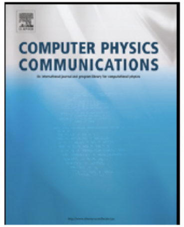
Computer Physics Communications 307 (2025) 109398

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

ELSEVIER



Computational Physics

The FLUKA Monte Carlo simulation of the magnetic spectrometer of the FOOT experiment ☆

Check for updates

Introduction - 2

The purpose is not to teach how to perform a correct FOOT Simulation using FLUKA, but just to use the simulation results.

The main topics are:

- **Give some basic info specific of FLUKA MC what everybody needs to know**
- **The structure of data produced by MC for FOOT**
- **Provide examples about the use and interpretation of these data, and the connection of detector hits and particle properties at MC-truth level**

The FLUKA MC code

https://www.fluka.org

All about the physics models of this code can be found in a very recent paper:

EPJ Nuclear Sci. Technol. **10**, 16 (2024)

https://doi.org/10.1051/epjn/2024015

EPJ Nuclear Sciences & Technologies

Available online at:
https://www.epj-n.org

Status and advances of Monte Carlo codes for particle transport simulation, edited by Andrea Zoia, Cheikh Diop and Cyrille de Saint Jean

REGULAR ARTICLE — OPEN ACCESS

The FLUKA code: Overview and new developments

Here we limit ourselves to summarize in a schematic way the use of nuclear interaction models

A few specific things of FLUKA MC that you need to know

Default units

the most important are:

time → s, length → cm, energy → GeV, momentum → GeV/c

masses → GeV/c^2 B → Tesla

Reference frame: (cartesian, right-handed)

z is primary beam direction
y is pointing upwards

It coincides with the global reference frame used in SHOE, with origin (0,0,0,) at the center of target

Particles:

each particle is identified by a number

A few specific things of FLUKA MC that you need to know

Fluka name	Fluka no.	Common name	Fluka name	Fluka no.	Common name
4-HELIUM	-6	Alpha	PION+	13	Positive Pion
3-HELIUM	-5	Helium-3	PION-	14	Negative Pion
TRITON	-4	Triton	KAON+	15	Positive Kaon
DEUTERON	-3	Deuteron	KAON-	16	Negative Kaon
HEAVYION	-2	Generic heavy ion with Z > 2	LAMBDA	17	Lambda
OPTIPHOT	-1	Optical Photon	ALAMBDA	18	Antilambda
RAY	0	Pseudoparticle	KAONLONG	12	Kaon-zero long
PROTON	1	Proton	KAONSHRT	19	Kaon zero short
APROTON	2	Antiproton	NEUTRIM	27	Muon neutrino
ELECTRON	3	Electron	ANEUTRIM	28	Muon antineutrino
POSITRON	4	Positron	TAU+	41	Positive Tau
NEUTRIE	5	Electron Neutrino	TAU-	42	Negative Tau
ANEUTRIE	6	Electron Antineutrino	NEUTRIT	43	Tau neutrino
PHOTON	7	Photon	ANEUTRIT	44	Tau antineutrino
NEUTRON	8	Neutron			
ANEUTRON	9	Antineutron			
MUON+	10	Positive Muon			
MUON-	11	Negative Muon			

Here only the most important

In FOOT you will find mostly: -6, -5, -4, -3, -2, 1, 3, 4, 7, 8

Less often (above ~290 MeV/u): 10, 11, 13, 14

Rare, only in case of high energy, also: 15, 16

Since we are mostly interested to nuclear fragments, notice:

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

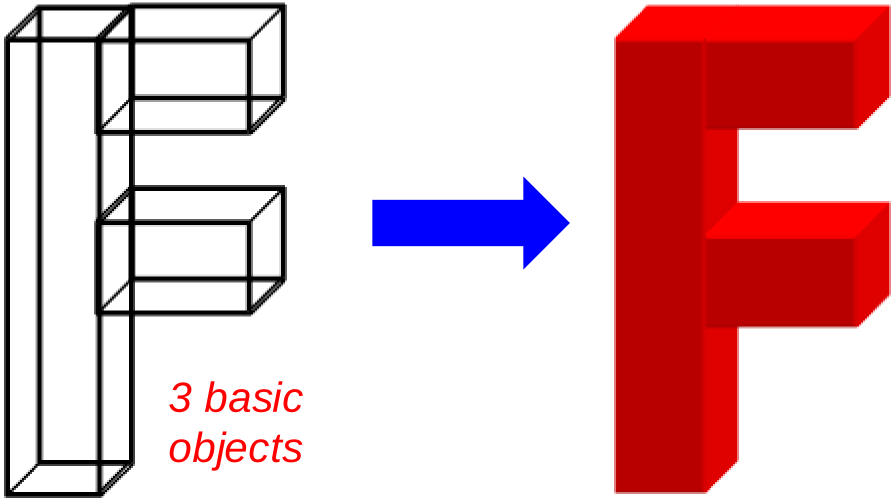
For A>4: FLUKA particle numbers is always -2, and nucleus is identified by Z and A

Very low energy fragments and nucleons originating in the "nuclear evaporation" phase are identified with a particle number in the range from -39 to -7. Again identified by Z and A.

In principle there could be also a way to identify isomers, but we do not include it in FOOT simulation

The concept of “Region” in FLUKA

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



1 complex object **= REGION**

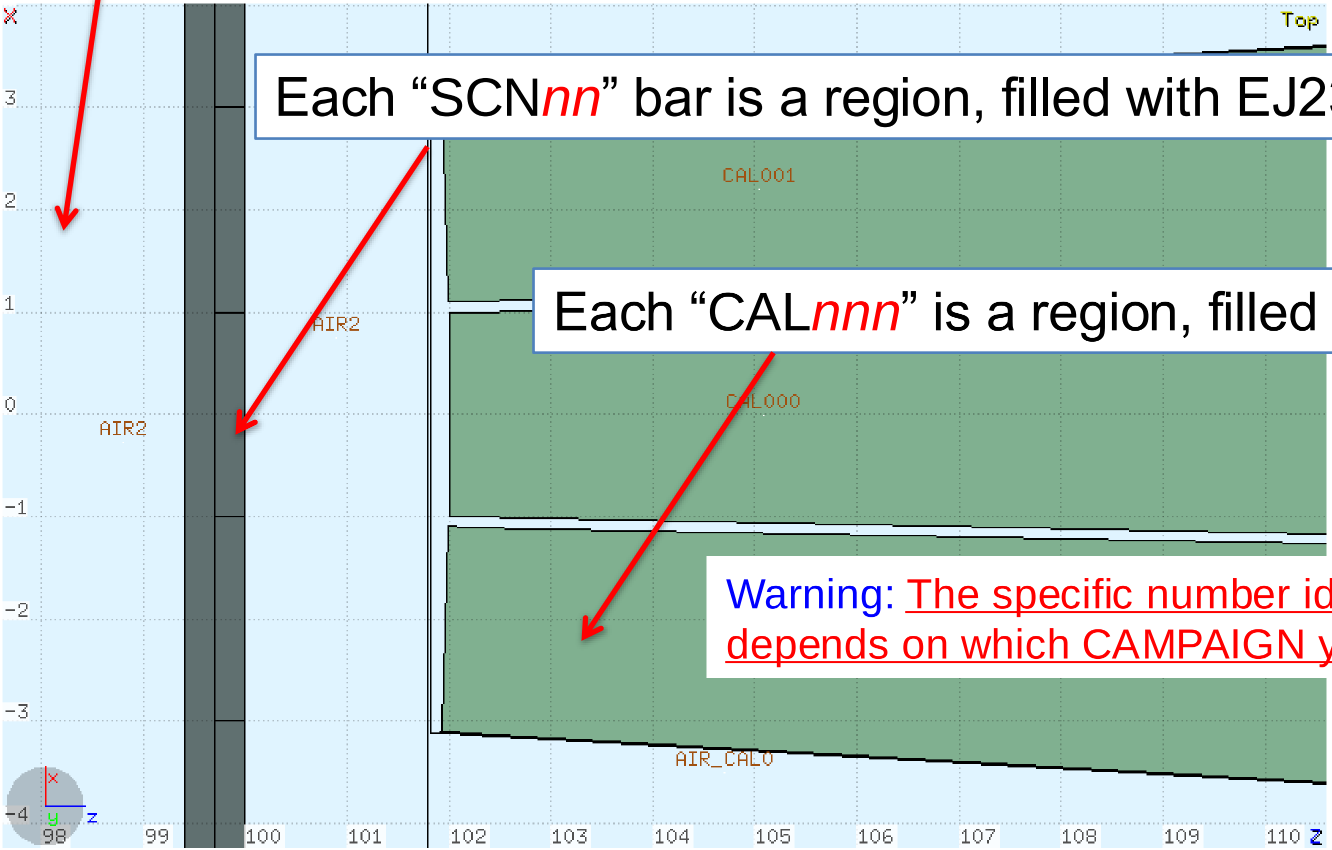
- The user knows the region usually by **name**, but internally (and in SHOE) it is identified by a **number**

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

“AIR2” is a region, filled with air (N, O, Ar @ STP)

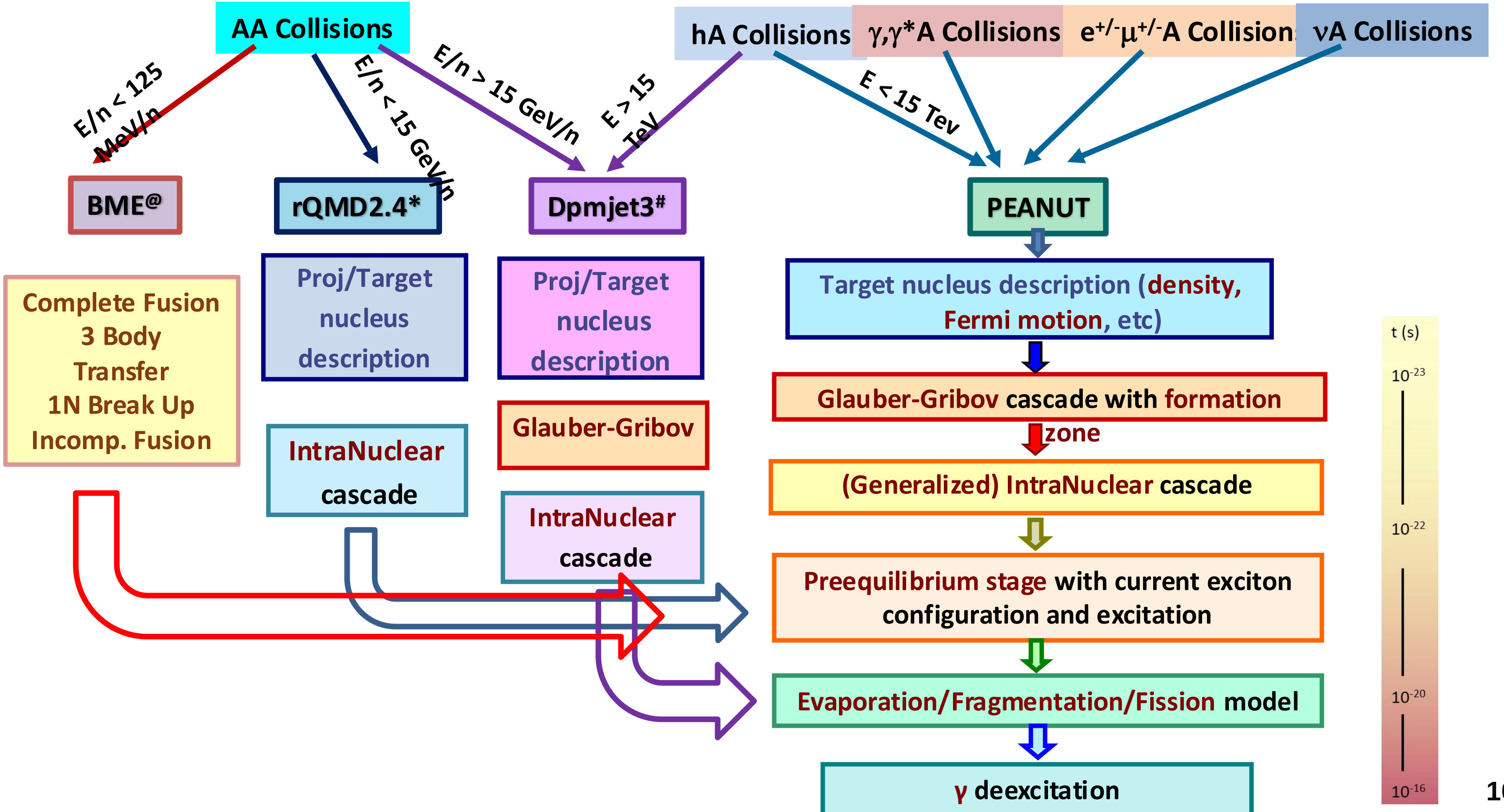
Each “SCN*nn*” bar is a region, filled with EJ232 scintillator

Each “CAL*nnn*” is a region, filled with BGO

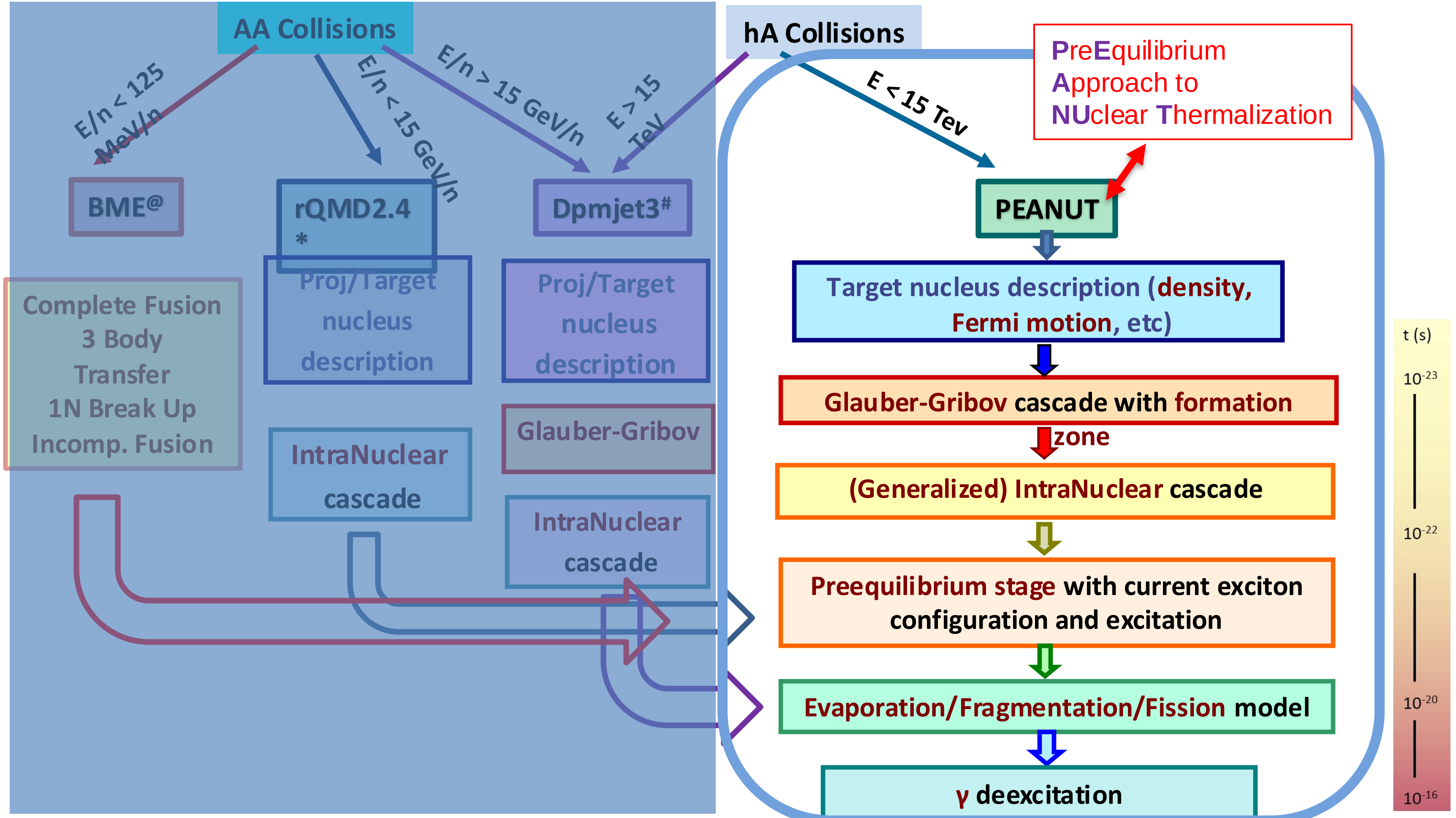


Warning: The specific number identifier of a region depends on which CAMPAIGN you are using

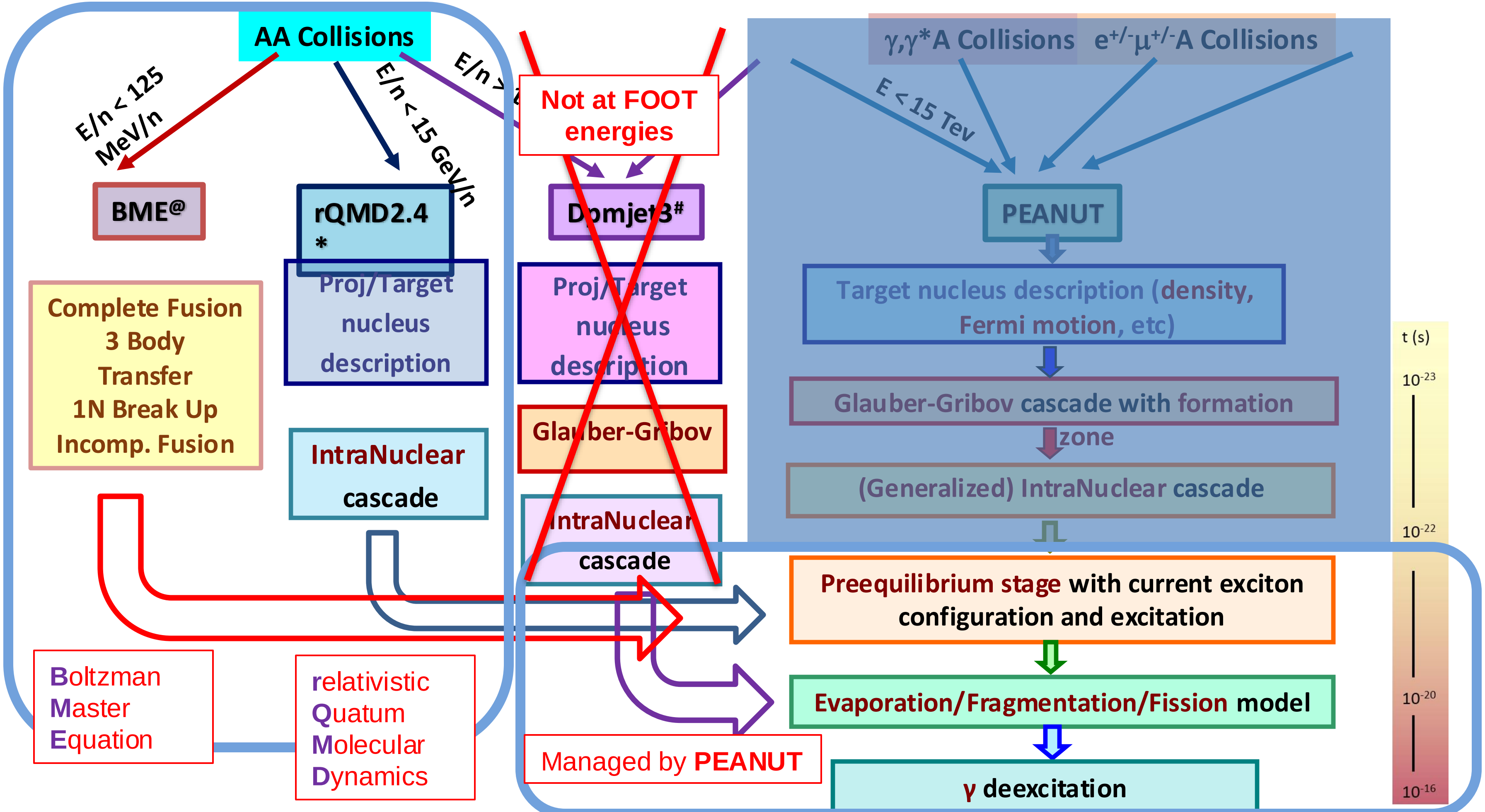
FLUKA nuclear interaction models:



hadron-Nucleus interactions: (p-N, n-N, …)



Nucleus-Nucleus interactions:



A few words on MC settings:

All models are automatically activated.

The energy threshold for charged particle and photon transport is set at 100 keV, while for neutrons is set at 10 μeV.
In order to limit CPU time and output file size, the transport of e^+e^- is switched off (a part few specific simulation studies), however, the carefully tuned models of FLUKA managing dE/dx and its fluctuations guarantee a result which is independent of the choice of the e^+e^- cut-off.

In order to prevent mistakes or mistypes, all the input directives and geometry setup for a given simulation campaign are created by SHOE

Conventions for MC campaign naming

We append _MC to the campaign name, to signify that this is a campaign of simulated data and distinguish it from the campaign of real experimental data.

Example:

CNAO2023 is the experimental campaign (data taken at CNAO in 2023)

CNAO2023_MC is the corresponding simulation campaign

Recently we have introduced in the simulation geometry some important passive regions (boxes, PCB), and we are substituting old campaigns. For example:

CNAO2023_MC ➔ CNAO23PS_MC

In simulation campaigns we have run numbers (in analogy to experimental data). We are adopting as convention a number corresponding to the beam energy/nucleon and the target material. For example, in campaign CNAO23PS_MC (^{12}C @ 200 MeV/u), we have the following runs:

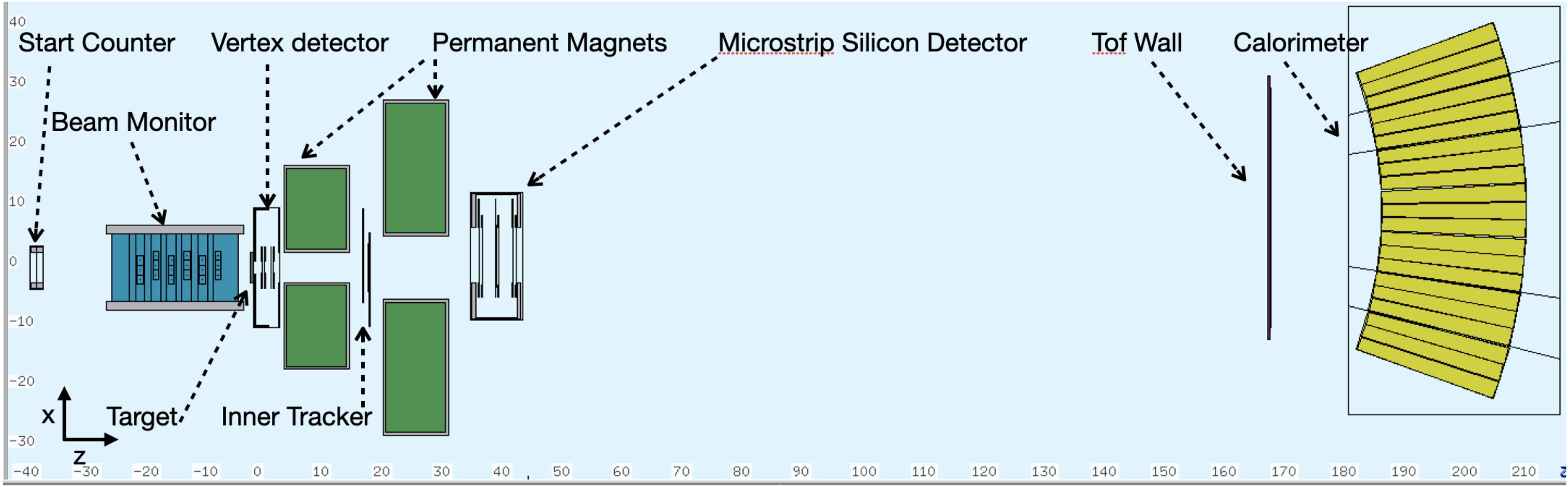
200 ➔ graphite target

201 ➔ polyethylene target

202 ➔ no target

203 ➔ Aluminum target

Detectors in CNAO23PS_MC Campaign



Where the FOOT user can retrieve relevant infos about geometry and materials used in simulation

For a given Campaign XXXX:

In shoe/build/Reconstruction/cammaps/XXXX.cam
you see the detectors included, and the possible run numbers. In FOOT.cam it is specified if the campaign is a simulated one

In shoe/build/Reconstruction/config/XXXX/FootGlobal.par
you see the detectors selected for reconstruction (*y* or *n* in a list) and specify other choices important also for simulated data *(see slide #18)*

In shoe/build/Reconstruction/geomaps/XXXX there are, among the others:

- FOOT(_nnn).geo which contains the positions (of the “center”), dimensions and rotation angles in global coordinates of all FOOT detectors and magnets. nnn is the run number: nnn is there only if there are more than 1 run.
- TA*detector(_nnn).map which contain, for each single detector (or magnet system), the relative coordinates and rotation angle of every element composing the detector itself, together with the material description.
- TAGdetector(_nnn).map contains info about target and primary beam

Example of cammap file (CNAO23PS_MC)

```
// Campaign file
CamName: "CNAO23PS_MC"
RunNumber: 200;201;202;203
NumberDevices: 10

DetectorName: "FOOT"
NumberFiles: 2
"./geomaps/CNAO23PS_MC/FOOT.geo": 200;201;202;203
"./geomaps/CNAO23PS_MC/FOOT.reg": -1

DetectorName: "DI"
NumberFiles: 1
"./geomaps/CNAO23PS_MC/TADIdetector.geo": -1

DetectorName: "ST"
NumberFiles: 1
"./geomaps/CNAO23PS_MC/TASTdetector.geo": -1

DetectorName: "BM"
NumberFiles: 3
"./geomaps/CNAO23PS_MC/TABMdetector.geo": -1
"./config/CNAO23PS_MC/TABMdetector.cfg": -1
"./calib/CNAO23PS_MC/TABM_T0_Calibration.cal": -1

DetectorName: "TG"
NumberFiles: 1
"./geomaps/CNAO23PS_MC/TAGdetector.geo": 200;201;202;203

DetectorName: "VT"
NumberFiles: 3
"./geomaps/CNAO23PS_MC/TAVTdetector.geo": -1
"./config/CNAO23PS_MC/TAVTdetector.cfg": -1
"./calib/CNAO23PS_MC/TAVTdetector.cal": -1
```

There are 4 runs
Same energy (200 MeV/u),
different targets

(possible) different geometries

CNAO23PS_MC.cam

```
DetectorName: "IT"
NumberFiles: 2
"./geomaps/CNAO23PS_MC/TAITdetector.geo": -1
"./config/CNAO23PS_MC/TAITdetector.cfg": -1

DetectorName: "MSD"
NumberFiles: 3
"./geomaps/CNAO23PS_MC/TAMSDdetector.geo": -1
"./config/CNAO23PS_MC/TAMSDdetector.cfg": -1
"./config/CNAO23PS_MC/TAMSDdetector.map": -1

DetectorName: "TW"
NumberFiles: 6
"./geomaps/CNAO23PS_MC/TATWdetector.geo": -1
"./config/CNAO23PS_MC/TATWdetector.cfg": -1
"./config/CNAO23PS_MC/TATW_BBparameters.cfg": -1
"./config/CNAO23PS_MC/TATWbarsMapStatus.map": -1
"./calib/CNAO23PS_MC/TATW_Energy_Calibration.cal": -1
"./calib/CNAO23PS_MC/TATW_Tof_Calibration.cal": -1

DetectorName: "CA"
NumberFiles: 4
"./geomaps/CNAO23PS_MC/TACAdetector.geo": -1
"./config/CNAO23PS_MC/TACAdetector.cfg": -1
"./config/CNAO23PS_MC/TACAcrystalMapStatus.map": -1
"./calib/CNAO23PS_MC/TACA_Energy_Calibration.cal": -1
```

Examples from geomaps

FOOT_200.geo

```
StartBaseName: "ST"
StartPosX: 0. StartPosY: 0. StartPosZ: -45.925
StartAngX: 0. StartAngY: 0. StartAngZ: 0.

TargetBaseName: "TG"
TargetPosX: 0. TargetPosY: 0. TargetPosZ: 0.
TargetAngX: 0. TargetAngY: 0. TargetAngZ: 0.

BmBaseName: "BM"
BmPosX: 0. BmPosY: 0. BmPosZ: -12.85
BmAngX: 0. BmAngY: 0. BmAngZ:   0.

VertexBaseName: "VT"
VertexPosX: 0. VertexPosY:   0. VertexPosZ: 2.61
VertexAngX: 0. VertexAngY:   0. VertexAngZ: 0.

MagnetsBaseName: "DI"
MagnetsPosX: 0. MagnetsPosY:   0. MagnetsPosZ: 19.00
MagnetsAngX: 0. MagnetsAngY:   0. MagnetsAngZ: 0.

InnerTrackerBaseName: "IT"
InnerTrackerPosX: 0. InnerTrackerPosY: 0. InnerTrackerPosZ: 19.00
InnerTrackerAngX: 0. InnerTrackerAngY: 0. InnerTrackerAngZ: 0.

MicroStripBaseName: "MSD"
MicroStripPosX: 1.9 MicroStripPosY: 0. MicroStripPosZ: 40.9
MicroStripAngX: 0. MicroStripAngY: 0. MicroStripAngZ: 0.

TofWallBaseName: "TW"
TofWallPosX: 9.1 TofWallPosY: -1.6 TofWallPosZ: 169.75
TofWallAngX: 0. TofWallAngY:  0. TofWallAngZ: 0.

CaloBaseName: "CA"
CaloPosX: 8.56 CaloPosY: -1.7 CaloPosZ: 200.5
CaloAngX: 0. CaloAngY:  0. CaloAngZ: 0.
```

TAGdetector_200.map

```
// -+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+-
// Beam info
// -+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+-
BeamSize:          0.48
BeamShape:         "Gaussian"
BeamEnergy:        0.2   //! GeV/u
BeamAtomicMass:    12   //! A Beam
BeamAtomicNumber:   6   //! Z Beam
BeamMaterial:      "C"   //! Beam Material
BeamPartNumber:     1   // particles in Beam
BeamPosX:          -0.4   BeamPosY:    0.1    BeamPosZ:    -63.0
BeamSpreadX:       0.26668 BeamSpreadY:  0.57112  BeamSpread:    0.0
BeamDiv:           0.0000

// -+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+-
// Target info (cm)
// -+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+--+-+-+-+-+-
TargetShape:       "cubic"
TargetSizeX:       5.0   TargetSizeY:    5.0    TargetSizeZ:  0.5
TargetMaterial:    "C"
TargetAtomicMass:  12.0107
TargetDensity:     1.83
TargetExc:         78.0e-6
```

It contains Beam and Target parameters

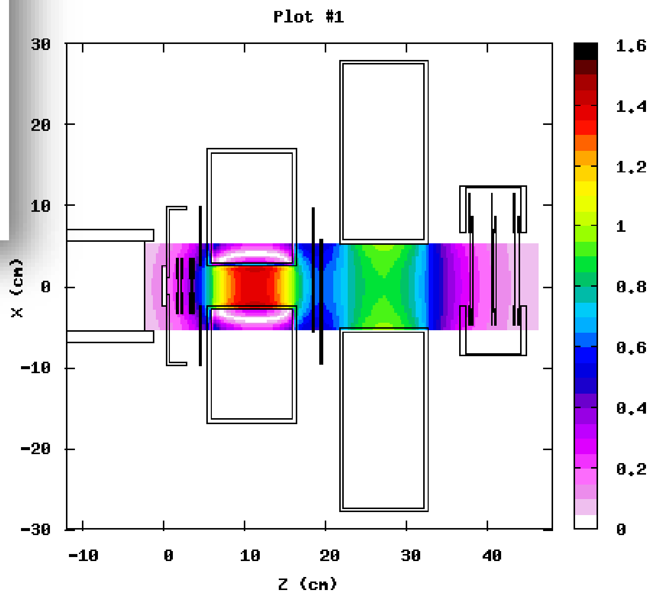
All these data are used to create in the same way, both the simulation geometry and parameters (such as materials composition), and an identical geometry and set of parameters for the reconstruction used in SHOE

Configuration of Magnets

The name of the magnetic map filename and geometry parameters for the magnets are in geomaps/XXXX/TADIdetector.map.

88641 21 21 201

x	y	z	Bx	By	Bz
-5.00000000000	-5.00000000000	-50.0000000000	1.36243669343	-30.6770814637	0.459840156935E-16
-5.00000000000	-5.00000000000	-49.5000000000	1.35464451635	-30.8022112988	0.528218691543
-5.00000000000	-5.00000000000	-49.0000000000	1.34685233926	-30.9273411338	1.05643738309
-5.00000000000	-5.00000000000	-48.5000000000	1.35906061221	-31.0238462683	1.64156333646
-5.00000000000	-5.00000000000	-48.0000000000	1.40067845000	-31.2369115992	2.25326833731
-5.00000000000	-5.00000000000	-47.5000000000	1.43021238978	-31.4820544068	2.78929569702
-5.00000000000	-5.00000000000	-47.0000000000	1.45974632956	-31.7271972143	3.32532305672
-5.00000000000	-5.00000000000	-46.5000000000	1.48928026933	-31.9723400218	3.86135041643
-5.00000000000	-5.00000000000	-46.0000000000	1.56924075328	-32.2800339546	4.50091052135
-5.00000000000	-5.00000000000	-45.5000000000	1.65712860643	-32.6735541981	5.15804511664
-5.00000000000	-5.00000000000	-45.0000000000	1.71563615145	-33.1755302923	5.73249807025
-5.00000000000	-5.00000000000	-44.5000000000	1.80905040239	-33.6194597825	6.32749263121
-5.00000000000	-5.00000000000	-44.0000000000	1.93494986313	-34.1492021605	7.15672152936
-5.00000000000	-5.00000000000	-43.5000000000	2.04046917779	-34.7722498674	7.87790619204
-5.00000000000	-5.00000000000	-43.0000000000	2.20446996652	-35.4680040321	8.63503397742
-5.00000000000	-5.00000000000	-42.5000000000	2.38506450881	-36.3909573966	9.57226578086
-5.00000000000	-5.00000000000	-42.0000000000	2.47932864781	-37.4124786747	10.4730338056



The map of magnetic field is contained in shoe/build/Reconstruction/data

at present we make use of the map in the file “MagneticMap_2023.table”:

It is the map measured during 2023 beam test

Accessing Regions in Simulated data

For some specific analysis of simulated data, it may be useful to exploit infos about "Regions" of simulated geometry, trying to answer questions like:

- In which region this particle was generated?
- Which are the coordinates and the values of kinematics variables of a given particle while passing from on region to another? (for example: exiting from target into air, entering in a given sensor of VTX, etc. etc.)
- …

In the case of simulated campaigns, Shoe gives access to such information.

Let us remind that the different regions are identified by a number (*see slide #6*), therefore interested users must know the correspondence between those numbers and the name of the regions.

Region numbering is summarized in geomaps/XXXX/FOOT.reg

Warning: there is no explanation on the meaning of region names

Example: Region Numbering for CNAO23PS_MC

Interpreted from geomaps/CNAO23PS_MC/FOOT.reg

	Number.	Name:	Piece of detector
Region n.	1	BLACK	"Black Hole"
Region n.	2	AIR1	Air
Region n.	3	AIR2	Air
Region n.	4-18	AIRCAL0 – AIRCAL14	Pieces of Air around Calo
Region n.	19	STC	Start Counter
Region n.	29	STCMYL1	Mylar foil in front of Start Counter
Region n.	21	STCMYL2	Mylar foil on the back of SC
Region n.	22	BMN_SHI	BM Al Shield
Region n.	23	BMN_MYL0	BM Mylar foil at the entrance
Region n.	24	BMN_MYL1	BM Mylar foil at the exit
Region n.	25-60	BMN_C000 – BMN_C117	BM Cells
Region n.	61	BMN_FWI	BM Field wires
Region n.	62	BMN_SWI	BM Sense wires
Region n.	63	BMN_GAS	BM gas (non – sensitive)
Region n.	64	TARGET	Target
Region n.	65-76	VTXE0 – VTXP3	All different parts of VTX sensors
Region n.	65-76	VTXE0 – VTXP3	All different parts of VTX sensors
Region n.	77-224	ITRE00 – ITRY112	All different parts of IT sensirs
Region n.	225-242	MSDS0 – MSDM5	All different parts of MSD sensors
Region n.	243-246	MAG0 – MAG_SH1	All different parts of Magnets
Region n.	247-286	SCN000 – SCN119	TW bars
Region n.	287-619	CAL000 – CAL332	BGO crystals
Region n.	620-656	ACAL_00 – ACAL_36	AIR gaps around the BGO crystals

etc. etc.

In order to better understand in detail the different simulation regions, interested people should contact
giuseppe.battistoni@mi.infn.it
silvia.muraro@mi.infn.it

Numbers may vary from campaign to campaign: *They depend on the geometry of a given setup*

Actually, Shoe allows to retrieve the region number (to be used in coding) from the region name (see later)

Simulated data files and their processing

Simulated data are distributed as Root files containing the structure of the raw simulated data organized in Shoe trees. The structure of simulated events will be explained in a next section

Simulated data are stored in a shared area in the INFN computing resources. For example:
/storage/gpfs_data/foot/shared/SimulatedData/CNAO23PS_MC/**12C_C_200_1.root**

Projectile Target Energy

Therefore, this is a run 200 of campaign CNAO23PS_MC

At present, our default is to write on file **all events** (1 primary = 1 event)

These are not yet reconstructed data (→ no track reconstruction!).
However, simulated data can be:

1) Used just as raw data to perform analyses at the "MC truth" level
2) Processed and reconstructed using the Shoe global tracking by applying the same approach used for experimental data

Global Reconstruction of simulated data

Users have to take care of reconstruction. The same Shoe code used for real data has to be invoked.

Assuming to be in a directory shoe/build/Reconstruction, the essential parameters to drive track reconstruction are contained in the file shoe/build/Reconstruction/config/XXXX/FootGlobal.par

The line command to start global tracking is:

Mandatory for simulated data

../bin/DecodeGlb –in 12C_C_200_1.root –exp CNAO23PS_MC –run 200 –mc –nev *nnn* (–nsk *mmm)* –out *yourfilename*

No. of events to be processed

No. of events to be skipped

But this <u>only after checking the content of FootGlobal.par</u>

Inside the FootGlobal.par file

IncludeKalman: y
IncludeTOE: n
IncludeStraight: n
FromLocalReco: n
…
…
N measure in global tracking: 9
…

To select tracking method

Minimum no. of points required to define a global track

IncludeDI: y
IncludeST: y
IncludeBM: y
IncludeTG: y
IncludeVT: y
IncludeIT: y
IncludeMSD: y
IncludeTW: y
IncludeCA: y

To select which detectors have to be included in track reconstruction

…
EnableTree: y
EnableFlatTree: n
EnableHisto: y
EnableTracking: y

Optional

EnableSaveHits: y
~~EnableRootObject~~: y
EnableRegionMc: y
EnableElecNoiseMc: y
…

Specific for simulated data

Working at the level of MC truth:

What you can take out from the root file with raw simulated data

(no global tracking yet)

Most relevant SHOE classes for MC

TAMCevent

TAMCntuEvent

TAMCntuPart

TAMCntuHit

TAMCntuRegion

See their implementation in /shoe/Libraries/TAMCbase

In the following, some examples of coding, to be used in SHOE macros to readout simulated data, will be given.
These examples can be implemented (or, in part, are already implemented) in a template macro that you can find in /shoe/Reconstruction/macros

ReadShoeTreeMain.C
ReadShoeTreeFunc.h

You are invited to start using such a template

Working with MC with a SHOE macro - 1

When processing a simulated root file, you can use in your macro the methods defined in shoe/Libraries/TAMCbase (TAMCntuEve.hxx, TAMCntuEve.cxx)

```
//opens the file and access the tree
TTree *tree = 0;
TFile *f = new TFile(nameFile.Data());
tree = (TTree*)f->Get("tree");
if(tree==nullptr){
  tree = (TTree*)f->Get("EventTree");
}....
```

All this can be the same for both real and simulated data

```
//Accessing basic infos: campaign name and run number
....
TAGroot gTAGroot;
static TAGrunInfo* runinfo;
TString expName;
runinfo=(TAGrunInfo*)(f->Get("runinfo"));
const TAGrunInfo construninfo(*runinfo);
gTAGroot.SetRunInfo(construninfo);
expName=runinfo->CampaignName();
if(expName.EndsWith("/"))
  expName.Remove(expName.Length()-1);
Int_t runNumber=runinfo->RunNumber();
TAGrecoManager::Instance(expName);
TAGcampaignManager* campManager = new TAGcampaignManager(expName);
campManager->FromFile();
cout << "Campaign is: " << expName << endl;
cout << "Run Number is: " << runNumber << endl;
```

TAGrunInfo

Getting Campaign name and run number

Both for real and simulated data

Working with MC with a SHOE macro - 2

This variable checks if you are reading simulated data.
All these instruction can be used in the same way for both real and simulated data

```
//Checking the existence of detector elements
   IncludeMC=campManager->GetCampaignPar(campManager->GetCurrentCamNumber()).McFlag;
IncludeREG=runinfo->GetGlobalPar().EnableRegionMc;
IncludeIT = runinfo->GetGlobalPar().IncludeIT;
IncludeDI = runinfo->GetGlobalPar().IncludeDI;
IncludeSC = runinfo->GetGlobalPar().IncludeST;
IncludeBM = runinfo->GetGlobalPar().IncludeBM;
IncludeVT = runinfo->GetGlobalPar().IncludeVT;
IncludeTG = runinfo->GetGlobalPar().IncludeTG;
IncludeMSD = runinfo->GetGlobalPar().IncludeMSD;
IncludeTW = runinfo->GetGlobalPar().IncludeTW;
IncludeCA = runinfo->GetGlobalPar().IncludeCA;
```

Other important classes, common to both real and simulated data, are those which give access to geometry parameters:

TAGparGeo, TASTparGeo, TABMparGeo, TAVTparGeo, *etc. (one for each detector)*

```
//Accessing geometry infos
   static TAGgeoTrafo*  geoTrafo;
geoTrafo = new TAGgeoTrafo();
TString parFileName = campManager>GetCurGeoFile(TAGgeoTrafo::GetBaseName(), runNumber);
geoTrafo->FromFile(parFileName);

if (IncludeSC) {
  TAGparaDsc* parGeoSt = new TAGparaDsc(new TASTparGeo());
  TASTparGeo* stparGeo = (TASTparGeo*)parGeoSt->Object();
  parFileName = campManager->GetCurGeoFile(TASTparGeo::GetBaseName(), runNumber);
  stparGeo->FromFile(parFileName);
}
```

Example for SC (Start Counter), but it's similar for the other detectors

Working with MC with a SHOE macro - 3

How to retrieve region numbers(*). This is meaningful, of course, only on Simulated Data

```
//Retrieves region numbers fom region names
 TString regnameTg="TARGET";
Int_t RegTarg = runinfo->GetRegion(regnameTg);
TString regnameSTC="STC";
Int_t RegSTC = runinfo->GetRegion(regnameSTC)
...
if (IncludeCA) {
  TString regnameCALmin="CAL000";
  RegCALmin = runinfo->GetRegion(regnameCALmin);
  TString maxCryReg;
  if (nCry<10) {
    maxCryReg = Form("CAL00%d",nCry-1);
  } else if (nCry>9 && nCry<100) {
    maxCryReg = Form("CAL0%d",nCry-1);
  } else {
    maxCryReg = Form("CAL%d",nCry-1);
  }
  cout << "Last Crystal Reg: " << maxCryReg << endl;
  TString regnameCALmax = maxCryReg;
  RegCALmax = runinfo->GetRegion(regnameCALmax);
 }
```

Example for Calorimeter

This is the same for both real and simulated data

```
//Retrieving Beam and Target properties
   Aweight = parGeo->GetTargetPar().AtomicMass;
 density = parGeo->GetTargetPar().Density;
 thickness= parGeo->GetTargetPar().Size[2];
 material = parGeo->GetTargetPar().Material;
 tgtcent = geoTrafo->GetTGCenter().Z();

 Abeam = parGeo->GetBeamPar().AtomicMass;
 Zbeam = parGeo->GetBeamPar().AtomicNumber;
 Ebeam = parGeo->GetBeamPar().Energy;
 Xbeam = parGeo->GetBeamPar().Position.X();
 Ybeam = parGeo->GetBeamPar().Position.Y();
 zbeam = parGeo->GetBeamPar().Position.Z();
 FWHMXbeam = parGeo->GetBeamPar().AngSpread.X();
 FWHMYbeam = parGeo->GetBeamPar().AngSpread.Y();
```

(*) This modality of retrieving region numbers from region names assumes that you know the meaning of the names. There is also another way: next slides

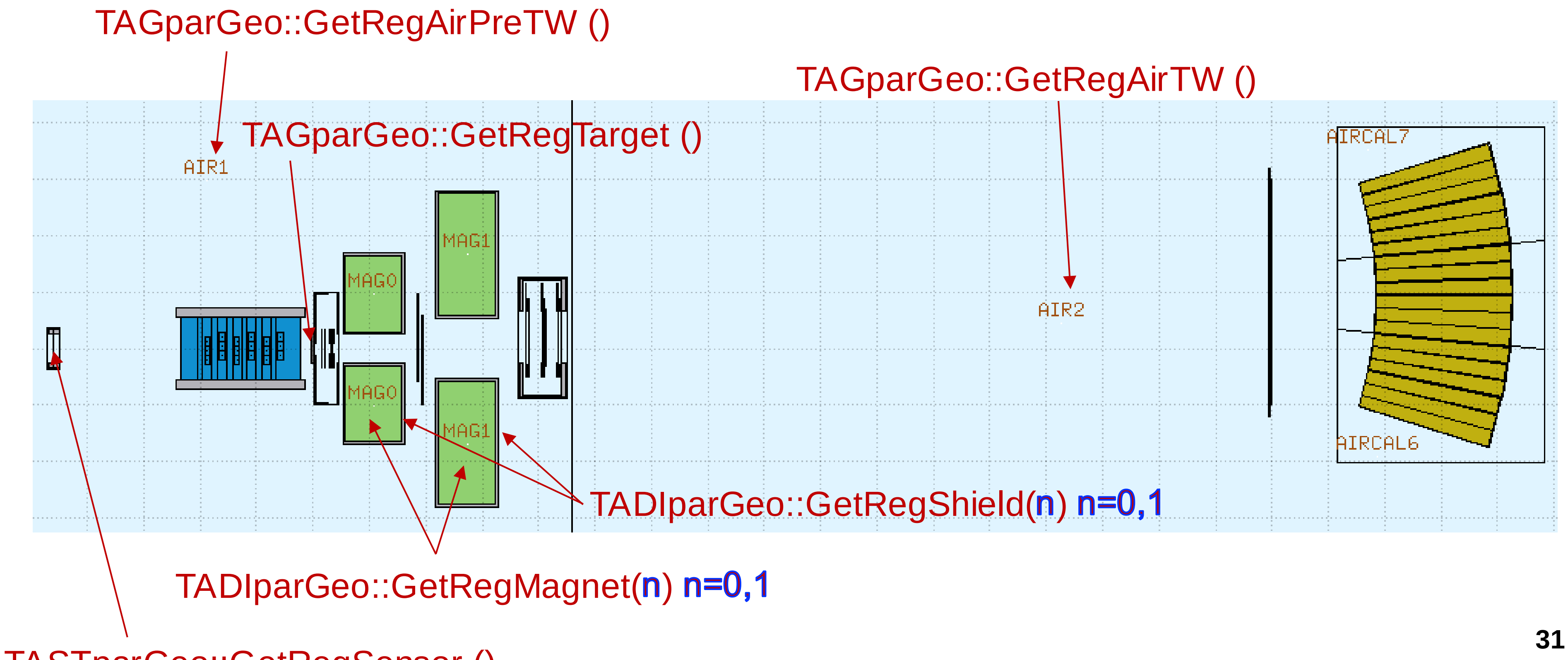
Retrieving Region Numbers by meaning (detector specific)

In the various TA*parGeo classes of the different detectors, including TAGparGeo, there are methods called GetReg***:
they allow to recover the number of specific, and fundamental, regions on the basis of their purpose (not all the regions)

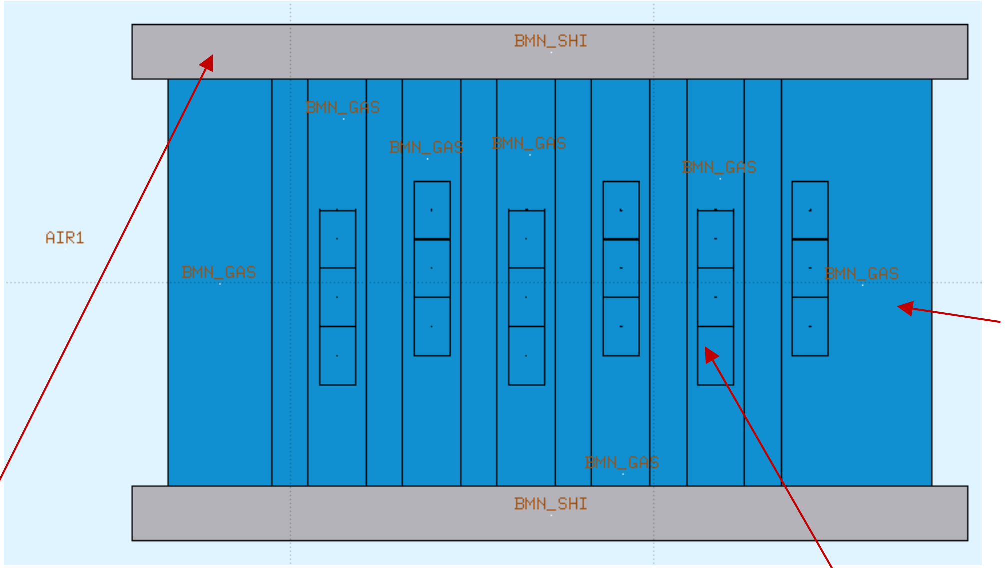
The exact name and modality is detector dependent.
Search inside the various TA*parGeo.hxx under /shoe/Libraries/TA**

For example: TAGparGeo::GetRegTarget() returns the region number of the target

Retrieving Region Numbers by meaning (detector specific)



Retrieving Region Numbers by meaning (detector specific): Beam Monitor



TABMparGeo:: GetRegGas()

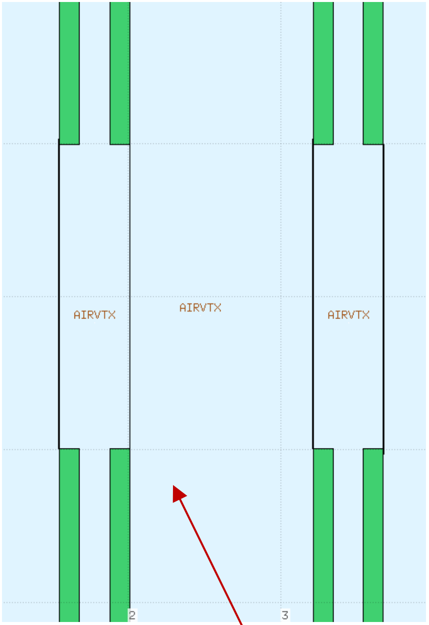
TABMparGeo::GetRegShield()

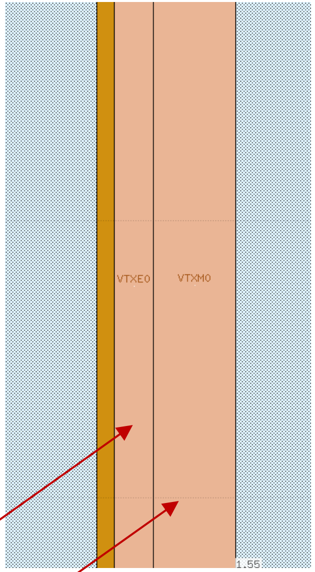
TABMparGeo:: GetRegCell (nlay, nview, ncell)

nlay=0-5, nview=0,1 ncell=0,2

TABMparGeo::GetRegFieldWires()
TABMparGeo::GetRegSenseWires()

Retrieving Region Numbers by meaning (detector specific): VTX



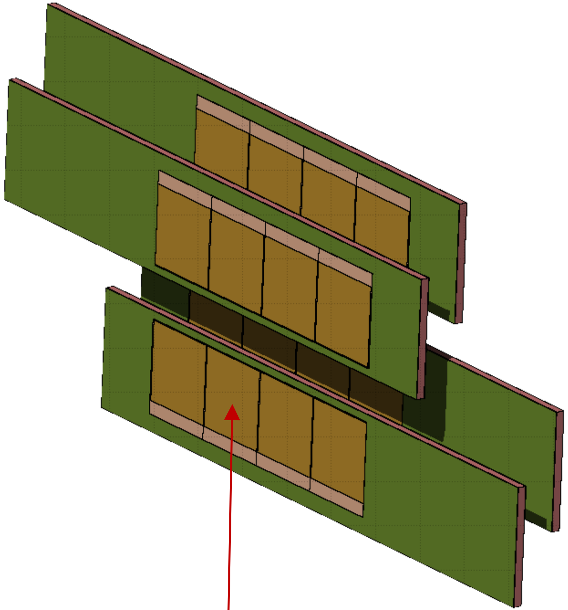


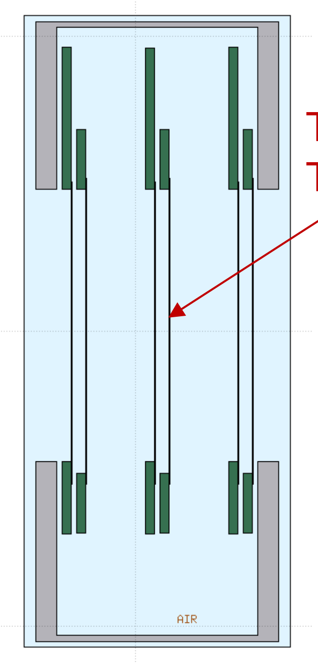
That’s the sensitive region

TAVTparGeo::**GetRegEpitaxial**(n);
TAVTparGeo::GetRegModule(n); n=0,1,2,3 (sensor #)

There are other GetReg methods Search inside the source files*

Retrieving Region Numbers by meaning (detector specific): IT and MSD





TAMSDparGeo::**GetRegStrip**(n)
TAMSDparGeo::GetRegModule(n)

n=0-5 (sensor #)

That's the sensitive region

TAITparGeo::**GetRegEpitaxial**(n)
TAITparGeo::GetRegModule(n)

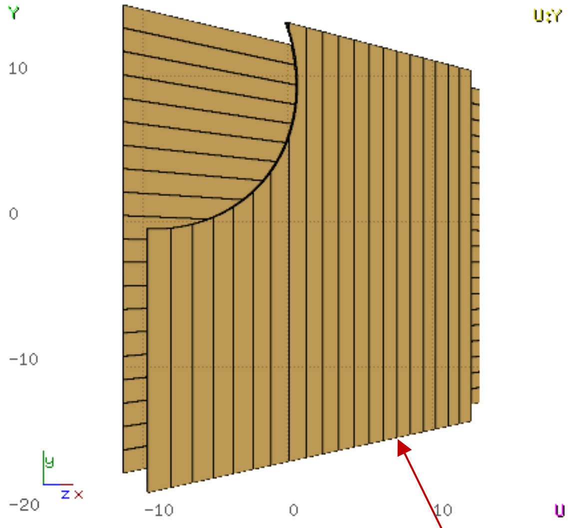
n=0-31 (sensor #)

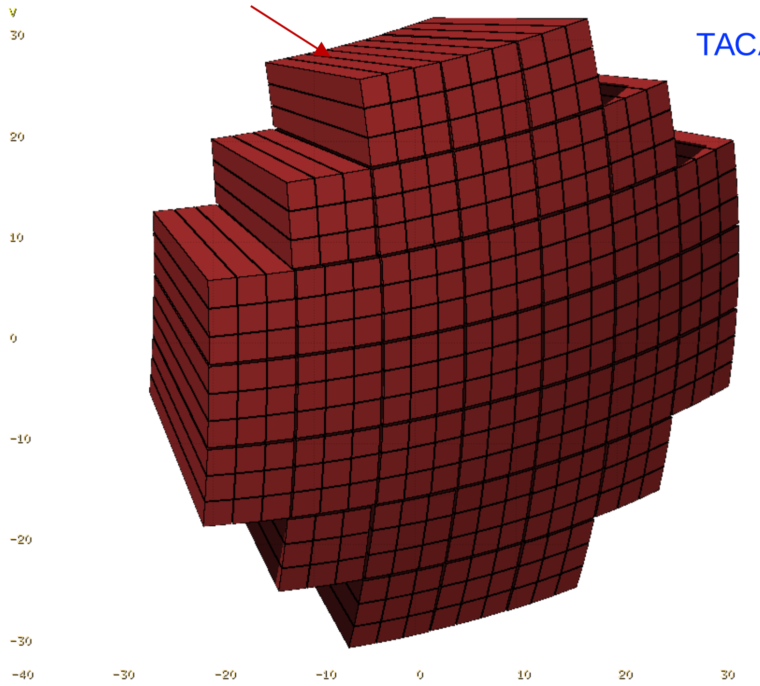
There are other GetReg methods Search inside the source files*

Retrieving Region Numbers by meaning (detector specific): TW and Calo

TACAparGeo::GetRegCrystal(n) n=0,max no. of crystals

TACAparGeo::GetCrystalsN()

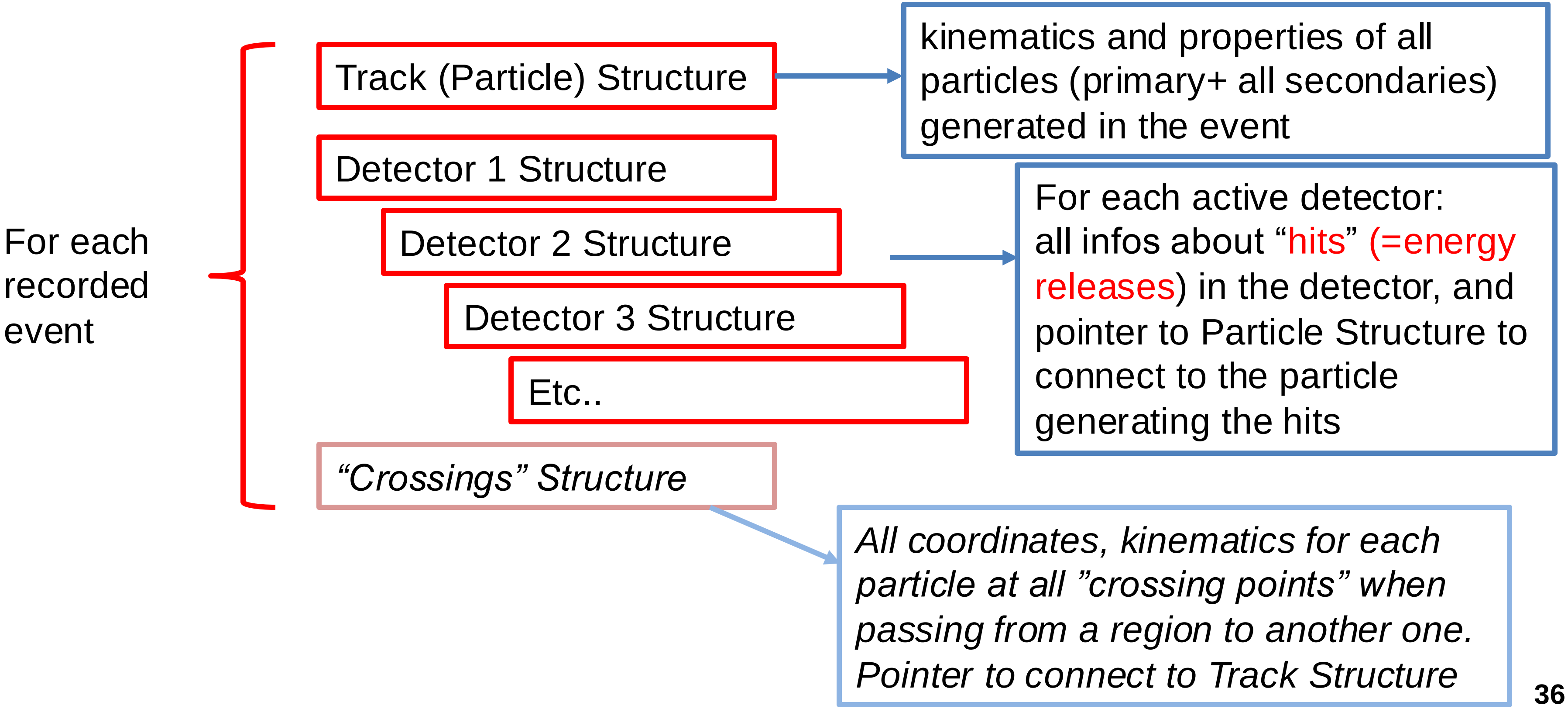




There are other GetReg methods Search inside the source files*

TATWparGeo::GetRegStrip(nlayer, nstrip) nlayer=0,1 nstrip=0,19

The native structure of raw simulated data



Variables available in the particle structure

Event by Event:

- number of particles produced in the event

Event by Event, for each particle contained in the event:

- Pointer to index the particle (*see later*)
- generation number
- charge (charge number Z)
- barionic number (mass number A)
- particle mass (GeV/c^2)
- Time of production the particle (s)
- Time between death and birth of the particle (s)
- Total track length of the particle from birth to death (cm)
- FLUKA code for the particle (*for example: proton=1, neutron=8, photon=7, ...*)
- number of the region where the particle has been produced
- Coordinates of the birth point of the particle (cm)
- Coordinates of the death point of the particle (cm)
- Components of the momentum of the particle (GeV/c) at birth point
- Components of the momentum of the particle (GeV/c) at death point

TAMCevent

```
//! Get Event container
TAMCntuEvent*  GetNtuEvent() const { return fEvent;  }
//! Get particle container
TAMCntuPart*   GetNtuTrack() const { return fTrack;  }
//! Get region container
TAMCntuRegion* GetNtuReg()   const { return fRegion; }
//! Get STC hits container
TAMCntuHit*    GetHitSTC()   const { return fHitSTC; }
//! Get BM hits container
TAMCntuHit*    GetHitBMN()   const { return fHitBMN; }
//! Get VTX hits container
TAMCntuHit*    GetHitVTX()   const { return fHitVTX; }
//! Get ITR hits container
TAMCntuHit*    GetHitITR()   const { return fHitITR; }
//! Get MSD hits container
TAMCntuHit*    GetHitMSD()   const { return fHitMSD; }
//! Get TW hits container
TAMCntuHit*    GetHitTW()    const { return fHitTW;  }
//! Get CAL hits container
TAMCntuHit*    GetHitCAL()   const { return fHitCAL; }
```

➔ gets the event

➔ particle structure in the event

➔ “crossing” structure in the event

➔ hits in the Start Counter

➔ hits in the Beam Monitor

➔ hits in the Vertex

➔ hits in the Inner Tracker

➔ hits in the MSD

➔ hits in the Tof-Wall

➔ hits in the Calorimeter

TAMCntuEvent

```
//! Get event number
Int_t GetEventNumber() const { return fEventNumber;   }
```

➔ Sequential event number in the simulation file

TAMCntuPart

```
// Get number of tracks
Int_t        GetTracksN() const;
// Get particle
TAMCpart*    GetTrack(Int_t i);
// Get Fluka Id
Int_t        GetFlukaID()     const { return fFlukaId;     }
// Get mother Id
Int_t        GetMotherID()    const { return fMotherId;    }
//! Get atomic charge
Int_t        GetCharge()      const { return fCharge;      }
// Get baryon number
Int_t        GetBaryon()      const { return fBaryon;      }
// Get initial position
TVector3     GetInitPos()     const { return fInitPos;     }
// Get final position
TVector3     GetFinalPos()    const { return fFinalPos;    }
// Get initial momentum
TVector3     GetInitP()       const { return fInitMom;     }
//! Get final momentum
TVector3     GetFinalP()      const { return fFinalMom;    }
// Get mass
Double_t     GetMass()        const { return fMass;        }
// Get region
Int_t        GetRegion()      const { return fRegion;      }
// Get particle time
Double32_t   GetTime()        const { return fTime;        }
// Get track length
Double32_t   GetTrkLength()   const { return fTrkLength;   }
// Get time of flight
Double32_t   GetTof()         const { return fTof;         }
```

➔ no. of particles in the event

➔ gets the particle

➔ FLUKA particle code

➔ Index of the mother

➔ Z

➔ A

➔ initial x,y,z

➔ final x,y,z

➔ Intial P_x,P_y,P_z

➔ final P_x,P_y,P_z

➔ Mass

➔ Number of region of birth

➔ Particle Time

➔ Total track length

➔ Particle Tof

About the meaning of "Birth" (Initial) and "Death" (Final)

Birth coordinates: the coordinates of the point in the global reference frame where a particle is injected, or generated by interaction or decay

Birth momentum: the 3-vector P components at the point of injection or generation

Death coordinates: the coordinates of the point in the global reference frame where a particle "dies". A particle dies when: 1) has an inelastic interaction; 2) decays; 3) exits from the geometry; 4) its energy goes below the transport threshold which has been set in simulation: it is then propagated to the end of the remaining CSDA range.

Death momentum: the 3-vector P components at the point of death. In case 4) P_{final} components are 0.

About the meaning of time:

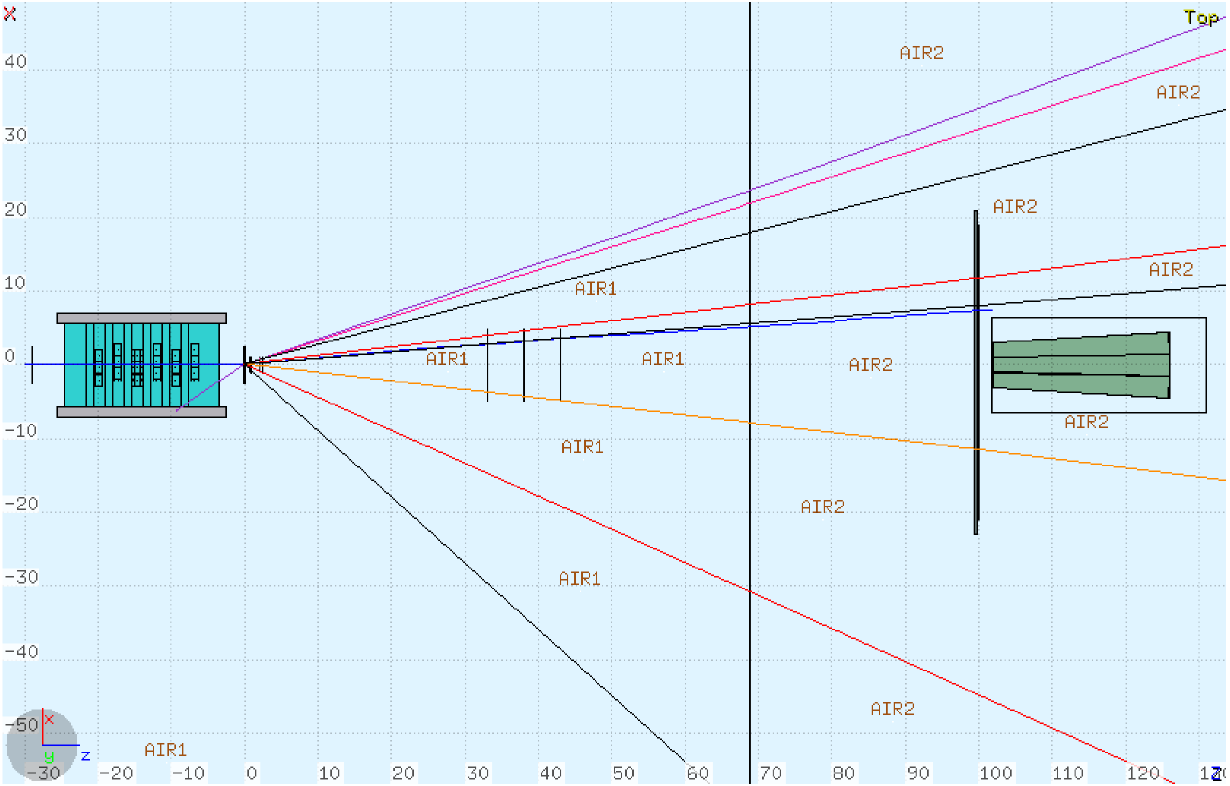
In a single event, Time starts from 0 in the point where the primary particle is injected.

Particle time: it has to be **0** for the primary. If the primary travels with velocity β, and interacts after a length **L**, the secondaries will be generated at $t = L/(\beta c)$ and that will be the value inside their time

Tof: it is the time difference between the ”death” and “birth” of a particle

An example to illustrate the potentiality of particle structure and the meaning of particle index

From an old simplified FOOT simulation of 2020



About the Index of the particles in the events

Index of the particle: index = 0 is the primary. The first track in the structure

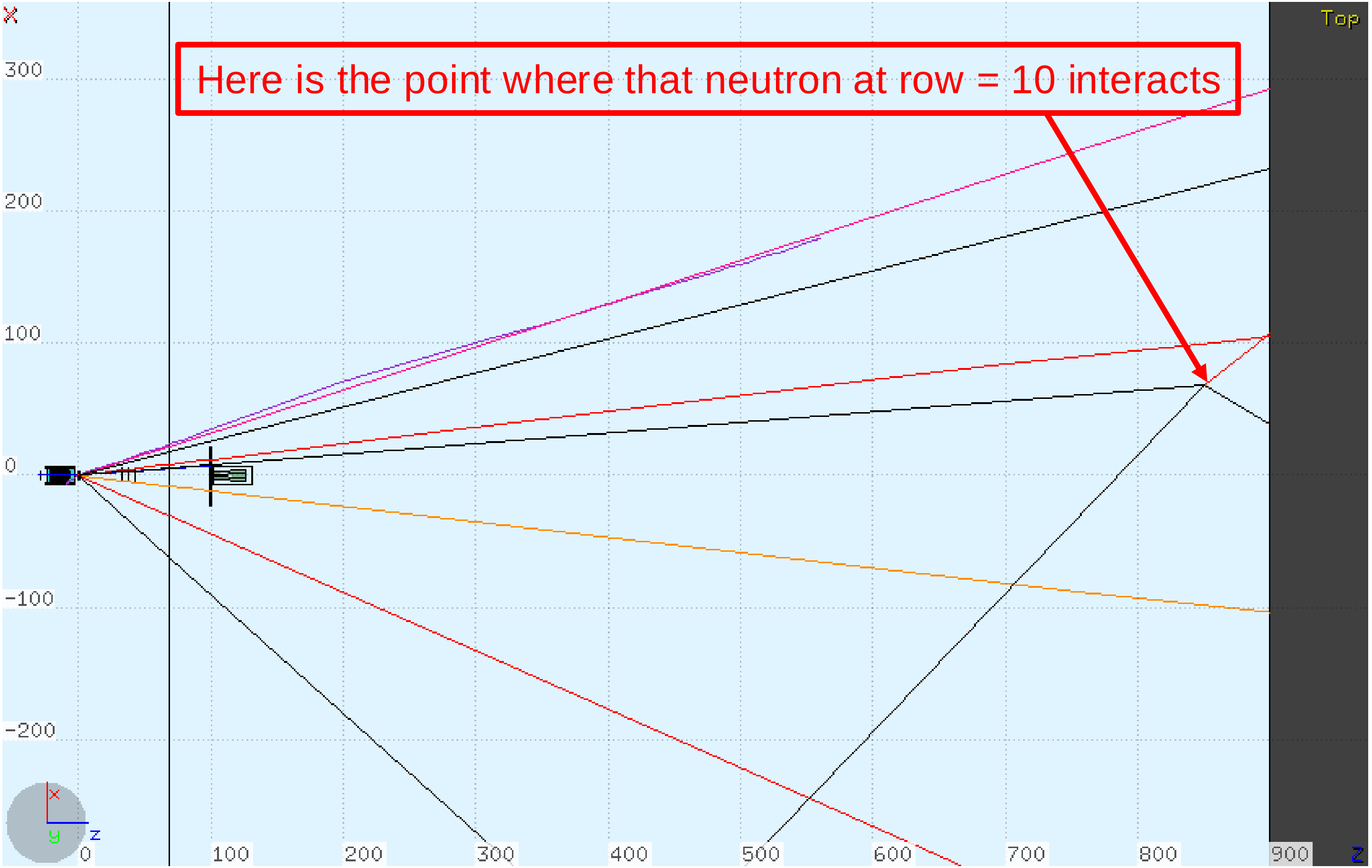
```
root [2] EventTree->Scan("TRn:TRpaid:TRfid:TRcha:TRbar:TRiz:TRfz","EventNumber==158")
***********************************************************************************************************
*    Row   * Instance *       TRn *    TRpaid *     TRfid *     TRcha *     TRbar *      TRiz *      TRfz *
***********************************************************************************************************
*        0 *        0 *        18 *         0 *        -2 *         6 *        12 *       -30 * -0.082383 *
*        0 *        1 *        18 *         1 *        -3 *         1 *         2 * -0.082383 * -9.383039 *
*        0 *        2 *        18 *         1 *        -3 *         1 *         2 * -0.082383 * 560.54650 *
*        0 *        3 *        18 *         1 *        -2 *         3 *         7 * -0.082383 * 101.68155 *
*        0 *        4 *        18 *         1 *        -5 *         2 *         3 * -0.082383 *       900 *
*        0 *        5 *        18 *         1 *        -6 *         2 *         4 * -0.082383 *       900 *
*        0 *        6 *        18 *         1 *         1 *         1 *         1 * -0.082383 *       900 *
*        0 *        7 *        18 *         1 *         1 *         1 *         1 * -0.082383 *       900 *
*        0 *        8 *        18 *         1 *         8 *         0 *         1 * -0.082383 *       900 *
*        0 *        9 *        18 *         1 *         8 *         0 *         1 * -0.082383 *       900 *
*        0 *       10 *        18 *         1 *         8 *         0 *         1 * -0.082383 * 850.46374 *
*        0 *       11 *        18 *        11 *        -6 *         2 *         4 * 850.46374 * 851.90222 *
*        0 *       12 *        18 *        11 *        -6 *         2 *         4 * 850.46374 * 850.58166 *
*        0 *       13 *        18 *        11 *        -6 *         2 *         4 * 850.46374 * 850.41534 *
*        0 *       14 *        18 *        11 *         8 *         0 *         1 * 850.46374 * -71.90350 *
*        0 *       15 *        18 *        11 *         1 *         1 *         1 * 850.46374 *       900 *
*        0 *       16 *        18 *        11 *         8 *         0 *         1 * 850.46374 *       900 *
*        0 *       17 *        18 *         1 *         1 *         1 *         1 * -0.082383 * -0.092312 *
***********************************************************************************************************
```

For index>0: **index-1 points to the parent particle**

All particles with index = 1 have been generated by the primary (index=0)

```
root [2] EventTree->Scan("TRn:TRpaid:TRfid:TRcha:TRbar:TRiz:TRfz","EventNumber==158")
************************************************************************************************************
*    Row   * Instance *       TRn *    TRpaid *     TRfid *     TRcha *     TRbar *      TRiz *      TRfz *
************************************************************************************************************
*        0 *        0 *        18 *         0 *        -2 *         6 *        12 *       -30 * -0.082383 *
*        0 *        1 *        18 *         1 *        -3 *         1 *         2 * -0.082383 * -9.383039 *
*        0 *        2 *        18 *         1 *        -3 *         1 *         2 * -0.082383 * 560.54650 *
*        0 *        3 *        18 *         1 *        -2 *         3 *         7 * -0.082383 * 101.68155 *
*        0 *        4 *        18 *         1 *        -5 *         2 *         3 * -0.082383 *       900 *
*        0 *        5 *        18 *         1 *        -6 *         2 *         4 * -0.082383 *       900 *
*        0 *        6 *        18 *         1 *         1 *         1 *         1 * -0.082383 *       900 *
*        0 *        7 *        18 *         1 *         1 *         1 *         1 * -0.082383 *       900 *
*        0 *        8 *        18 *         1 *         8 *         0 *         1 * -0.082383 *       900 *
*        0 *        9 *        18 *         1 *         8 *         0 *         1 * -0.082383 *       900 *
*        0 *       10 *        18 *         1 *         8 *         0 *         1 * -0.082383 * 850.46374 *
*        0 *       11 *        18 *        11 *        -6 *         2 *         4 * 850.46374 * 851.90222 *
*        0 *       12 *        18 *        11 *        -6 *         2 *         4 * 850.46374 * 850.58166 *
*        0 *       13 *        18 *        11 *        -6 *         2 *         4 * 850.46374 * 850.41534 *
*        0 *       14 *        18 *        11 *         8 *         0 *         1 * 850.46374 * -71.90350 *
*        0 *       15 *        18 *        11 *         1 *         1 *         1 * 850.46374 *       900 *
*        0 *       16 *        18 *        11 *         8 *         0 *         1 * 850.46374 *       900 *
*        0 *       17 *        18 *         1 *         1 *         1 *         1 * -0.082383 * -0.092312 *
************************************************************************************************************
```

These particles with **index = 11** have been generated by the particle at row index-1 = 10 *(a neutron which interacts in air far away)*



Notice:

<u>At present simulation is generic and does not include a realistic room size: this means, for example, that no possible back-splash from walls is considered</u>

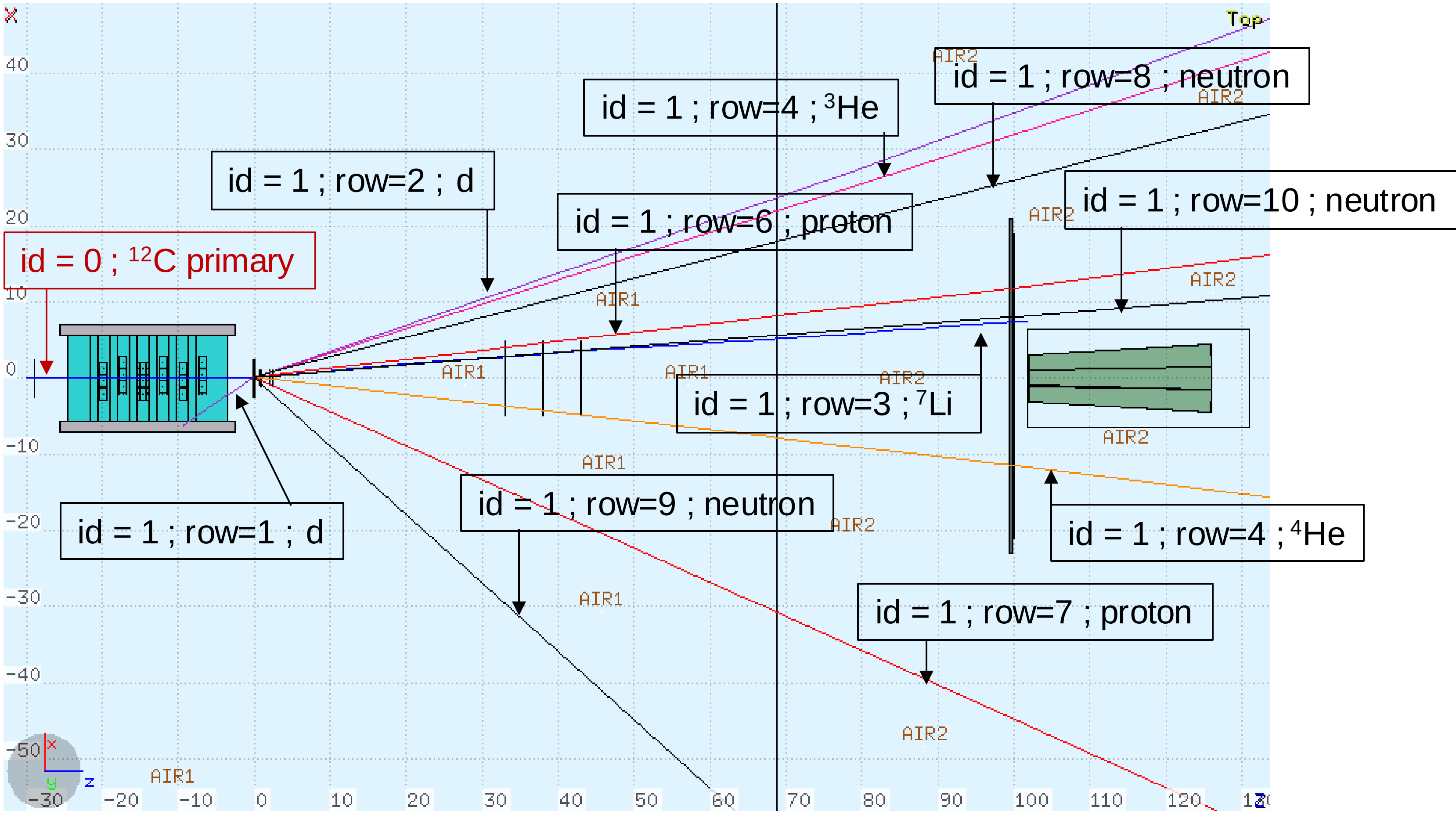
These particles exit from the geometry far away (z=900 cm)

```
root [2] EventTree->Scan("TRn:TRpaid:TRfid:TRcha:TRbar:TRiz:TRfz","EventNumber==158")
```

Row	Instance	TRn	TRpaid	TRfid	TRcha	TRbar	TRiz	TRfz
0	0	18	0	-2	6	12	-30	-0.082383
0	1	18	1	-3	1	2	-0.082383	-9.383039
0	2	18	1	-3	1	2	-0.082383	560.54650
0	3	18	1	-2	3	7	-0.082383	101.68155
0	4	18	1	-5	2	3	-0.082383	900
0	5	18	1	-6	2	4	-0.082383	900
0	6	18	1	1	1	1	-0.082383	900
0	7	18	1	1	1	1	-0.082383	900
0	8	18	1	8	0	1	-0.082383	900
0	9	18	1	8	0	1	-0.082383	900
0	10	18	1	8	0	1	-0.082383	850.46374
0	11	18	11	-6	2	4	850.46374	851.90222
0	12	18	11	-6	2	4	850.46374	850.58166
0	13	18	11	-6	2	4	850.46374	850.41534
0	14	18	11	8	0	1	850.46374	-71.90350
0	15	18	11	1	1	1	850.46374	900
0	16	18	11	8	0	1	850.46374	900
0	17	18	1	1	1	1	-0.082383	-0.092312

This proton has been generated by primary in the target, but dies in the target

Our example



id = 0 ; 12C primary
id = 1 ; row=1 ; d
id = 1 ; row=2 ; d
id = 1 ; row=3 ; 7Li
id = 1 ; row=4 ; 3He
id = 1 ; row=4 ; 4He
id = 1 ; row=6 ; proton
id = 1 ; row=7 ; proton
id = 1 ; row=8 ; neutron
id = 1 ; row=9 ; neutron
id = 1 ; row=10 ; neutron
AIR1
AIR2
Top

Data omitted in the event recording

1. Unfortunately, we never included (so far) Z, A of the target nucleus where interaction occur. At present Target Nucleus can be often reconstructed by checking Z and A conservation: $\sum Z_i$ of secondary particles having id=1 has to be equal to the sum of Z of primary and Z of target. The same for baryonic number conservation.

2. We have not marked in any way elastic scattering. Be careful when interaction occurs in materials where Hydrogen is present: the recoiling proton (H) from elastic scattering of the primary (or of a secondary fragment) may appear from coordinates where no inelastic interaction occurred…

The individual MC detector (hit) structures

For each detector *DET* with n energy releases (hits) we store some variables:

- number of hits (energy releases) in the detector
- index to the particle responsible of the hit
- initial position of hit
- final position of hit
- initial momentum of hit
- final momentum
- energy release in the hit ← *In FLUKA it is in GeV, in Shoe is in general converted to MeV*
- initial time of the energy release

specific variables depending on the type of *DET:* Layer, View, ….

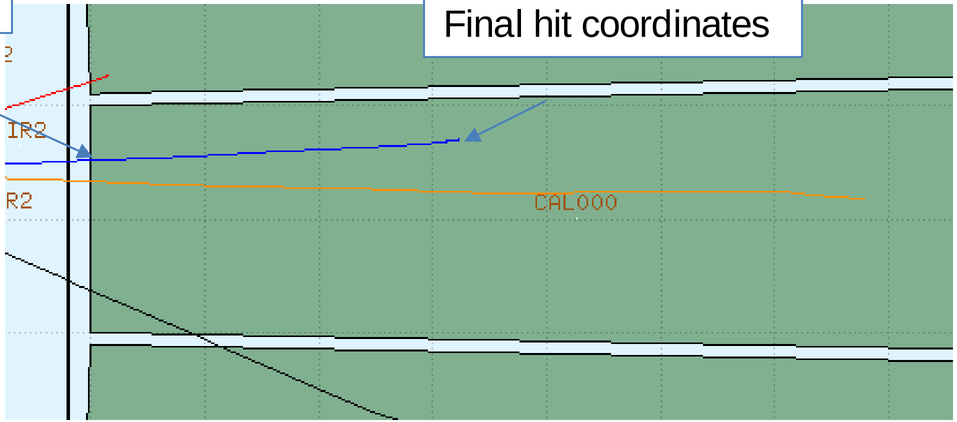
About the energy release in simulation - 1

<u>Charged particles</u>

A "hit" will be the energy lost during a "step" (with fluctuations of dE/dx properly considered in a continuous way). In a region where there is no tracking in magnetic field, each hit is a single step

Example for CAL

Initial hit coordinates



Final hit coordinates

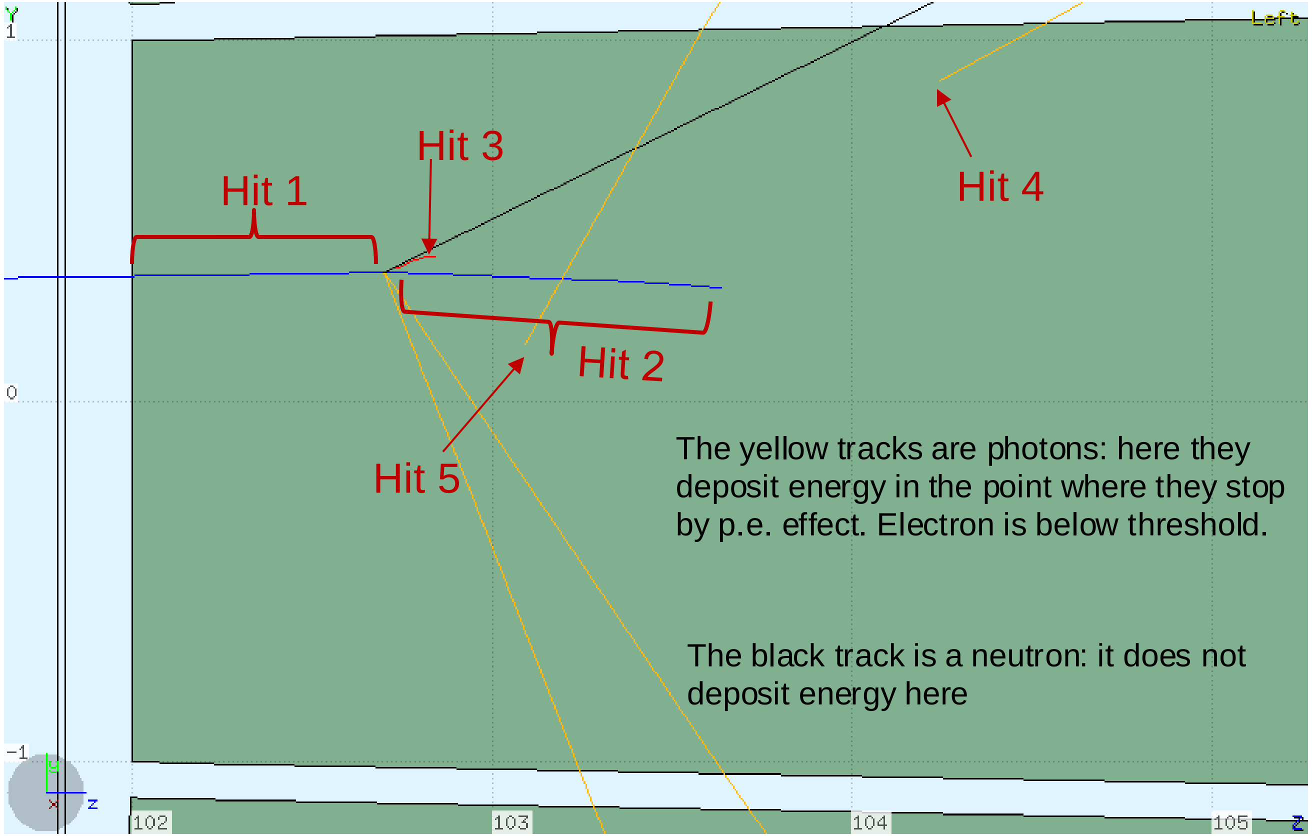
At the same points you can get the momentum components of the particle releasing energy, both at beginning and end of each hit

No electronics/detector effects → no experimental resolution
No quenching factors introduced so far
<u>Only physics intrisic fluctuations</u> (i.e. "*Landau*" fluct.)

These have to be introduced in your post-processing macros

About the energy release in simulation - 2

More complex cases



Of course:

1) the energy released per event in the same detector is the **sum of all ΔE**
2) The energy released per event in a single element of the detector is obtained by restricting the sum to a selected element. In this case you could usea specific variable to select a given crystal

About the energy release in simulation - 3

There are cases in which the Hits (Energy depositions) have point-like space dimension.

In Fluka this may occurr in some cases. For example:

a) for e^+/e^-/photons which go below transport energy threshold

b) “Low Energy” neutrons (E<20 MeV) which deposit energy by kerma factors

TAMCntuHit

```
Int_t        GetHitsN() const;        ➔ no. of hits for a given detector
TAMChit*     GetHit(Int_t i);         ➔ gets the hit
```

General

```
//! Get Track index
Int_t      GetTrackIdx()      const { return fID;       }   ➔ pointer to the particle generating the hit
//! Get position in
TVector3   GetInPosition()    const { return fInPosition;  }   ➔ initial coordinates of the hit
//! Get position out
TVector3   GetOutPosition()   const { return fOutPosition; }   ➔ final coordinates of the hit
//! Get momentum in
TVector3   GetInMomentum()    const { return fInMomentum;  }   ➔ Px, Py, Pz at the begin of the hit
//! Get momentum out
TVector3   GetOutMomentum()   const { return fOutMomentum; }   ➔ Px, Py, Pz at the end of the hit
//! Get energy loss
Double_t   GetDeltaE()        const { return fDeltaE;   }   ➔ enegy release in the hit (MeV)
//! Get time of flight
Double_t   GetTof()           const { return fTof;      }   ➔ time of the hit
```

Detector Specifc

```
//! Get Sensor id (VTX, IT...)
Int_t      GetSensorId()      const { return fLayer;    }
//! Get TW bar id
Int_t      GetBarId()         const { return fLayer;    }
//! Get CAL crystal id
Int_t      GetCrystalId()     const { return fLayer;    }
//! Get layer (meaning changes with detector)
Int_t      GetLayer()         const { return fLayer;    }
//! Get BM view or TW layer
Int_t      GetView()          const { return fView;     }
//! Get BM cell
Int_t      GetCell()          const { return fCell;     }
```

Retrieving MC HITS from Detector Structures in SHOE

This is possible, of course, only on Simulated Data

```
// MC hits of SC
TAMCntuHit *scMChits = 0x0;
// MC hits and tracks of Beam Monitor
TAMCntuHit *bmMCeve = 0x0;
// MC hits of VTX
TAMCntuHit *vtMChits = 0x0;
// MC hits of MSD
TAMCntuHit *msMChits = 0x0;
// MC hits of ITR
 TAMCntuHit *itMChits =  0x0;
// MC hits of SCN
 TAMCntuHit *twMChits =  0x0;
// MC hits of CAL
TAMCntuHit *caMChits = 0x0;
....
```

```
if(IncludeMC>0){
  if(IncludeSC>0) {
    scMChits = new TAMCntuHit();   // Get SC Hits
    tree->SetBranchAddress(FootBranchMcName(kST), &scMChits);
  }
  if(IncludeBM>0) {
    bmMCeve = new TAMCntuHit();   // Get BM Hits
    tree->SetBranchAddress(FootBranchMcName(kBM), &bmMCeve);
  }
  if(IncludeVT>0) {
    vtMChits = new TAMCntuHit();   // Get VT Hits
    tree->SetBranchAddress(FootBranchMcName(kVTX), &vtMChits);
  }
  if(IncludeIT>0) {
    itMChits = new TAMCntuHit();   // Get ITR Hits
    tree->SetBranchAddress(FootBranchMcName(kITR), &itMChits);
  }
  if(IncludeMSD>0) {
    msMChits = new TAMCntuHit();   // Get MSD Hits
    tree->SetBranchAddress(FootBranchMcName(kMSD), &msMChits);
  }
  if(IncludeTW>0) {
    twMChits = new TAMCntuHit();   // Get SCN Hits
    tree->SetBranchAddress(FootBranchMcName(kTW), &twMChits);
  }
 ...
```

Retrieving MC HITS from Detector Structures in SHOE, an example: the Beam Monitor

….

Somewhere inside a Loop on the events:

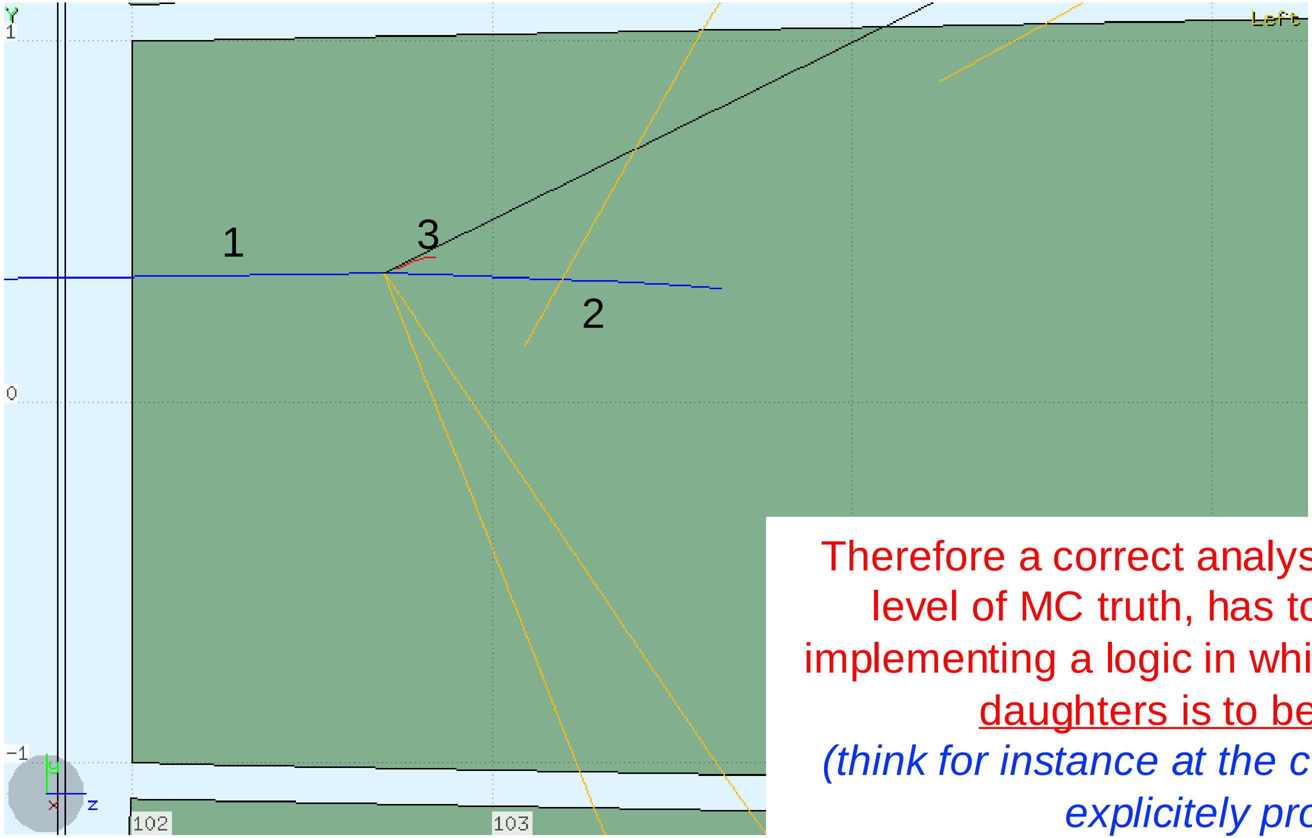
….

```
Int_t nbmHits  = bmNtuHit->GetHitsN();  gets the number of Hits in the event

for (Int_t i = 0; i < nbmHits; i++) {.         loop on the number of Hits
  TABMhit* hit = bmNtuHit->GetHit(i);    gets the Hit
  Int_t plane = hit->GetPlane();
  Int_t view  = hit->GetView();
  Int_t cell  = hit->GetCell();
 …
      etc. etc.
}
```

On the question of associating Hits with Particles

The issus is not so simple.
In this example the incoming particle releases energy with 3 Hits



Only one of them is directly associated to this particle (no. 1)

The other 2 Hits are associated to daughters of the incoming particle (products of an interaction)

Therefore a correct analysis of this kind, at the level of MC truth, has to be performed by implementing a logic in which the whole chain of daughters is to be considered

(think for instance at the case when δ-rays are explicitely produced)

Connecting Hits in Detectors to Track Structure: example in a SHOE macro

```
….
Somewhere inside a Loop on the events:
….
 Int_t nscMCHits  = scMChits->GetHitsN();  // Counts the hits in the Start Counter

  for (int i=0; i<nscMCHits; i++) {// Loop on the hits in the Start Counter
    TAMChit* schit=scMChits->GetHit(i);// Gets the i hit
    TAMCpart* mcpart=mcNtuPart->GetTrack(schit->GetTrackIdx());

   if (mcpart->GetCharge()==6 && mcpart->GetBaryon()==12) {
 …
   }
      etc. etc.
 }
```

pointer to the particle that generated that hit

Checks if that particle was a ^{12}C

The "region 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).

- number of boundary crossing
- index of the crossing particle in the particle block
- no. of region in which the particle is entering
- no. of region the particle is leaving
- Components of the momentum at the boundary crossing
- Coordinates of the point of the boundary crossing
- time of the boundary crossing
- charge of crossing particle
- mass of the crossing particle

} Redundant with respect to the variables from particle structure (charge, mass)

Very useful for many analyses about MC truth

TAMCntuRegion

```
// Get number of regions
Int_t          GetRegionsN() const;
// Get region
TAMCregion*      GetRegion(Int_t i);
//! Get track index
Int_t       GetTrackIdx()   const  { return fID;    }
//! Get number of crossing region
Int_t       GetCrossN()     const  { return fCrossN;  }
//! Get number of old crossing region
Int_t       GetOldCrossN()  const  { return fOldCrossN; }
//! Get poistion
TVector3     GetPosition()   const  { return fPosition;  }
//! Get momentum
TVector3     GetMomentum()   const  { return fMomentum;   }
//! Get mass
Double_t     GetMass()       const  { return fMass;      }
//! Get atomic charge
Double_t     GetCharge()     const  { return fCharge;    }
//! Get time
Double_t     GetTime()       const  { return fTime;      }
```

➔ number of crossings in the event

➔ gets a crossing

➔ pointer to the particle generating the crossing

➔ no. of region in which the particle is entering

➔ no. of the region from which the particle exits

➔ coordinats of the crossing point

➔ components of momentum at crossing point

➔ mass of the crossing particle

➔ charge number of the crossing particle

➔ time of the particle at the crossing point

Example: How to exploit Region Crossings in a SHOE macro

```
TAMCntuRegion* mcNtuReg;
if(IncludeMC>0){
   if(IncludeREG>0) {
     mcNtuReg = new TAMCntuRegion(); // Get MC Crossings
     tree->SetBranchAddress(TAGnameManager::GetBranchName(mcNtuReg->ClassName()), &mcNtuReg);
   }
....
```

Somewhere inside a Loop on the events:

```
Int_t nCross  = mcNtuReg->GetRegionsN(); // Counts the number of region crossings in the event
 for (int i=0; i<nCross; i++) { // Loop on the region crossings
  TAMCregion* cross=mcNtuReg->GetRegion(i); // Gets the i-crossing
    TVector3 crosspos = cross->GetPosition();        // Gets x, y, z global coordinates at crossing
  Int_t OldReg = cross->GetOldCrossN();           // Gets the number of the region from which the particle is exiting
    Int_t NewReg = cross->GetCrossN();              // Gets the number of the region in which the particle is entering
    Double_t time_cross = cross->GetTime();       // Gets the time at the moment of crossing
    TVector3 mom_cross = cross->GetMomentum(); // retrieves P at crossing
//now retrieves TrackID: which particle was making that region crossing?
    TAMCpart* mcpart=mcNtuPart->GetTrack(schit->GetTrackIdx());
  fid = mcpart->GetFlukaID();  // Gets the FLUKA particle-id
  cha = mcpart->GetCharge();  // Gets its charge
  bar = mcpart->GetBaryon();  // Gets its mass number
  reg = mcpart->GetRegion();  // Gets the number of the region where the particle was originated
....
}
```

Possible Basic Exercises using SHOE – MC truth

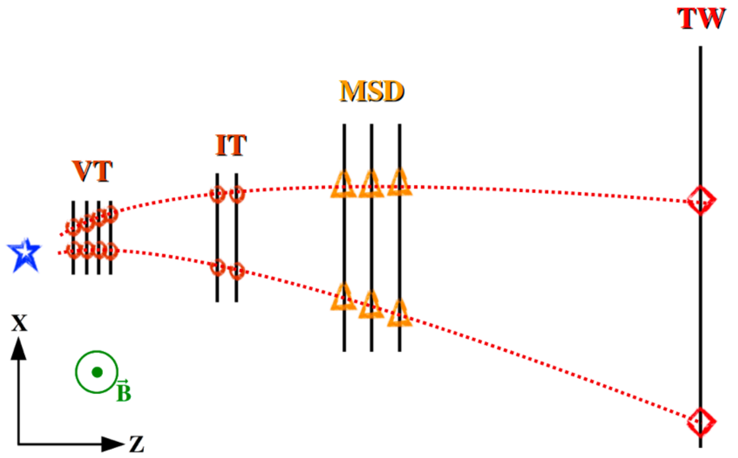
1. Make a plot of the multiplicity per event of particles produced anywhere in the detector
2. Make a plot of the multiplicity per event of particles produced by the primary in the target
3. Make the previous plot only for those particle which exit the target going in the forward region and are produced with E>50 MeV/u
4. Make a plot of the energy distribution of fragments produced in target for a few different Z and/or A
5. Make a plot of the energy released per event in the TW
6. Make a plot of the energy released per event in the CA and for a selected crystal of your choice

Slightly Increasing Difficulty:

7. Compare the distribution of energy released by p and ^{4}He in the 1st layer of MSD *(in the approximation that they do not produce daughters there)*

8. Select particles produced in the target which arrive at TW and make a plot of the energy that they have lost in the path from target to TW

Global tracks reconstructed in Simulated Data: How do we connect them to MC truth infos?



Tracks are reconstructed as in experimental data, just using detector hits & clusters, without exploiting data which would not be available in the real experiment.

All infos about actual particles in the simulated event are of course “forgotten” in reconstruction

It might also occur that points from different particles in the same event are accidentally used to define the same reconstructed track! (as in experimental data)

However, for simulated events, for each point in the track, it is possible to access the information about the actual particle which generated the hit

Global tracks reconstructed in Simulated Data: How do we connect them to MC truth infos?

Some possible operations:

```
static TAGntuGlbTrack *glbntutrk;
glbntutrk = new TAGntuGlbTrack();
tree->SetBranchAddress(TAGnameManager::GetBranchName(glbntutrk->ClassName()), &glbntutrk);
…
for(int i=0;i<glbntutrk->GetTracksN();i++){ // Loop on all trconstructed tracks
        TAGtrack* glbtrack=glbntutrk->GetTrack(i); // Gets the i-th track
        npoints = glbtrack->GetPointsN();       //No. of points in the  i-th reconstructed track
    if (IncludeMC) {
     Int_t mainPartId = glbtrack->GetMcMainTrackId(); // Id of the most prob. MC particles associated to the rec. track
           TAMCpart* mainPart = mcNtuPart->GetTrack(glbtrack->GetMcMainTrackId()); // Id of the most prob. MC particle
     for (int ic=0; ic<glbtrack->GetPointsN(); ic++) { // loop  on the points of the i-th reconstructed track
               TAGpoint *tmp_poi = glbtrack->GetPoint(ic);  // getting the ic-th point

               for(int t=0;t<tmp_poi->GetMcTracksN();t++) { // loop on all MC part. which can be associated to the ic-th track point
                  TAMCpart* tmpPart = mcNtuPart->GetTrack(tmp_poi->GetMcTrackIdx(t)); // gets the t-th MC particle
               }
            }
         }
}
```

A possible tweak for Global Track reconstruction for Simulated Data

The charge Z of a reconstructed track is obtained by combining ToF and Energy Loss in the TW. Z is available as a property of the object that we call “TW point”

A calibration is necessary, depending on energy and distance from target to TW

For Simulated Events, it is possible to ask to attribute to TW points, as charge reconstruction, the actual Z of the MC particle. This is achieved by means of a parameter in:
shoe/Reconstruction/config/XXXX/TATWdetector.cfg

```
EnableZmc:          0
EnableNoPileUp:     0
EnableZmatching:    1
EnableCalibBar:     0
EnableRateSmearMc:  0
BarsN:              40
GainWD:             1
EnableEnergyThr:    1
```

default (no MC charge)

Exercises using SHOE for MC rec. tracks

1. Make a scatter plot of the reconstructed charge Z for each point in the TW vs the charge of the actual MC particle associated to that point
2. For all reconstructed tracks, search for the MC particles contributing to the points of the track and:

- what is the fraction of ”pure” tracks (i.e. with all points belonging to the same particle)?
- for “pure” tracks, compare the reconstructed momentum with their MC momentum
- check if those particles were really produced by the primary in the target

mcpart->GetMotherID() == 0 (this means that the mother was a primary)

mcpart->GetRegion() == region number of target (campaign dependent)