Basic analysis of EICrecon output in ROOT

Barak Schmookler & Tyler Kutz

ePIC software tutorial

March 7-8, 2023

Simple analysis of EICrecon

Approach: analyze EICrecon output files treating the branches as arrays of values (not as edm4eic structures)

Pro:

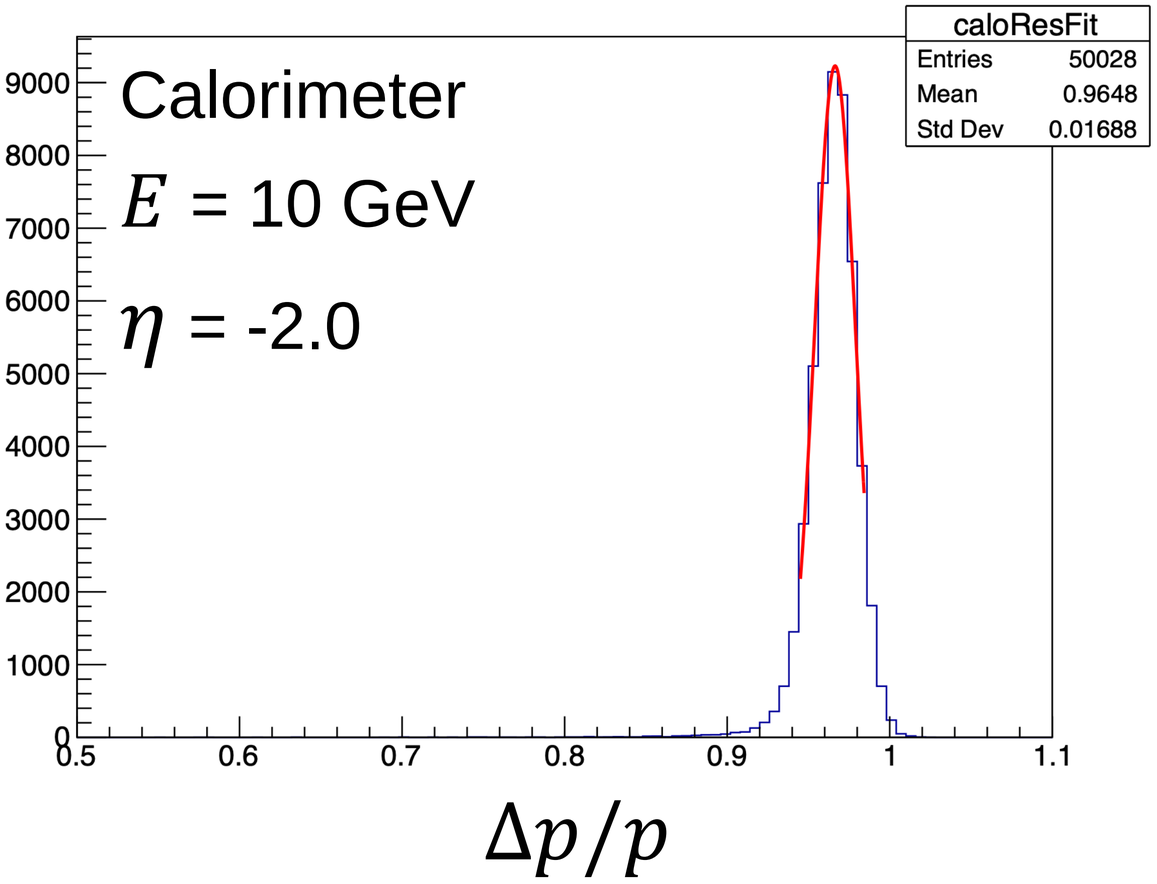
- No software overhead, no need for singularity, etc.
- Simply plot/read values from ROOT tree

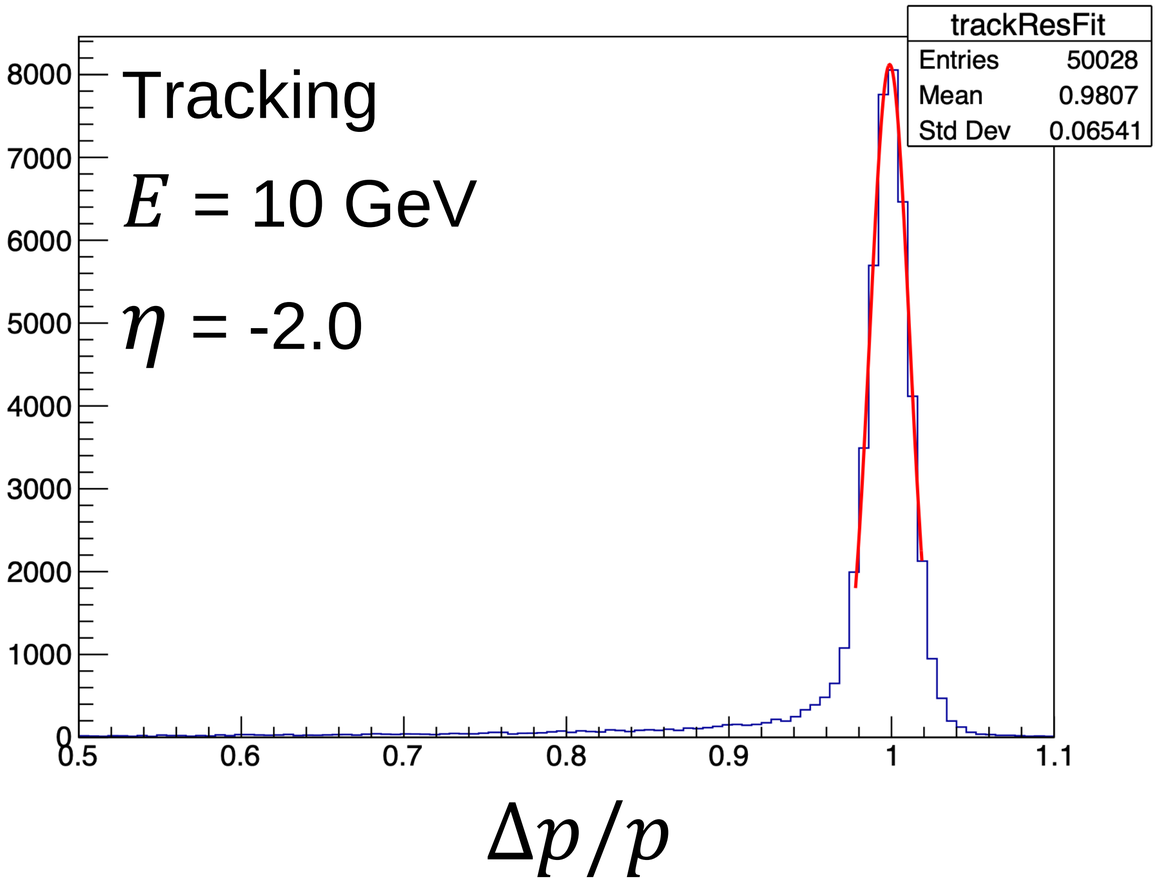
Con:

- Do not have access to edm4eic classes

Goal of this tutorial

Extract tracking and calorimeter resolutions from single-particle simulations





Opening EICrecon output file

```
TFile* eicreconFile =
            new TFile(“e-_10GeV_130to177deg.0001.eicrecon.tree.edm4eic.root");
TTree* eicreconTree = (TTree*)eicreconFile->Get("events");

int Nevents = eicreconTree->GetEntries();
```

Create a TTree Reader

```
//Create Array Reader
 TTreeReader tr(eicreconTree);

 TTreeReaderArray<float> px_mc(tr,"GeneratedParticles.momentum.x");
 TTreeReaderArray<float> py_mc(tr,"GeneratedParticles.momentum.y");
 TTreeReaderArray<float> pz_mc(tr,"GeneratedParticles.momentum.z");
 TTreeReaderArray<float> E_mc(tr,"GeneratedParticles.energy");
 TTreeReaderArray<float> M_mc(tr,"GeneratedParticles.mass");
 TTreeReaderArray<int>   PDG_mc(tr,"GeneratedParticles.PDG");
 TTreeReaderArray<int>   type_mc(tr,"GeneratedParticles.type");

 TTreeReaderArray<float> px_track(tr,"ReconstructedChargedParticles.momentum.x");
 TTreeReaderArray<float> py_track(tr,"ReconstructedChargedParticles.momentum.y");
 TTreeReaderArray<float> pz_track(tr,"ReconstructedChargedParticles.momentum.z");
 TTreeReaderArray<float> M_track(tr,"ReconstructedChargedParticles.mass");
 TTreeReaderArray<int> PDG_track(tr,"ReconstructedChargedParticles.PDG");

 TTreeReaderArray<float> h_endcap_E(tr,"EcalEndcapPMergedClusters.energy");
 TTreeReaderArray<float> h_endcap_theta(tr,"EcalEndcapPMergedClusters.intrinsicTheta");
 TTreeReaderArray<float> h_endcap_phi(tr,"EcalEndcapPMergedClusters.intrinsicPhi");
```

Create a TTree Reader

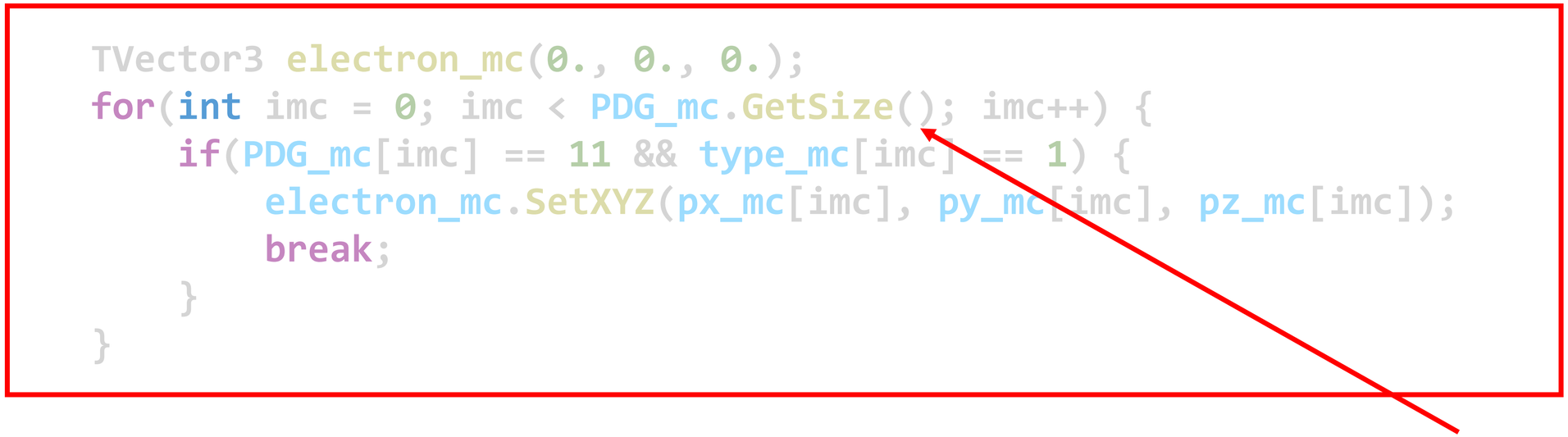
```
TTreeReaderArray<float> e_endcap_E(tr,"EcalEndcapNMergedClusters.energy");
TTreeReaderArray<float> e_endcap_theta(tr,"EcalEndcapNMergedClusters.intrinsicTheta");
TTreeReaderArray<float> e_endcap_phi(tr,"EcalEndcapNMergedClusters.intrinsicPhi");

TTreeReaderArray<float> barrel_E(tr,"EcalBarrelSciGlassClusters.energy");
TTreeReaderArray<float> barrel_theta(tr,"EcalBarrelSciGlassClusters.intrinsicTheta");
TTreeReaderArray<float> barrel_phi(tr,"EcalBarrelSciGlassClusters.intrinsicPhi");
```

Loop over reader and get MC truth information

```
while (tr.Next()) {

        if(ev%10000 == 0) cout << ev/1000 << "k / " << Nevents/1000 << "k events
processed" << endl;
        ResetVariables();

        TVector3 electron_mc(0., 0., 0.);
        for(int imc = 0; imc < PDG_mc.GetSize(); imc++) {
            if(PDG_mc[imc] == 11 && type_mc[imc] == 1) {
                electron_mc.SetXYZ(px_mc[imc], py_mc[imc], pz_mc[imc]);
                break;
            }
        }
```

Loop over reader and get MC truth information

```
while (tr.Next()) {

        if(ev%10000 == 0) cout << ev/1000 << "k / " << Nevents/1000 << "k events
processed" << endl;
        ResetVariables();

    TVector3 electron_mc(0., 0., 0.);
    for(int imc = 0; imc < PDG_mc.GetSize(); imc++) {
        if(PDG_mc[imc] == 11 && type_mc[imc] == 1) {
            electron_mc.SetXYZ(px_mc[imc], py_mc[imc], pz_mc[imc]);
            break;
        }
    }
```



All the ‘GeneratedParticles’ TTreeReaderArrays have the same length. So, we can pick one array to define the size and then find the index which corresponds the electron.

Loop over reader and get MC truth information

```
while (tr.Next()) {

        if(ev%10000 == 0) cout << ev/1000 << "k / " << Nevents/1000 << "k events
processed" << endl;
        ResetVariables();

        TVector3 electron_mc(0., 0., 0.);
        for(int imc = 0; imc < PDG_mc.GetSize(); imc++) {
            if(PDG_mc[imc] == 11 && type_mc[imc] == 1) {
                electron_mc.SetXYZ(px_mc[imc], py_mc[imc], pz_mc[imc]);
                break;
            }
        }
mc_p = electron_mc.Mag();
mc_eta = electron_mc.Eta();
mc_phi = electron_mc.Phi();
```

Calculate momentum, pseudorapidity, and azimuthal angle of track

Still in loop over reader – Get reconstructed track…

```
//Assume a single reconstructed track
rec_track = 0;
TVector3 electron_track(0., 0., 0.);
if(PDG_track.GetSize()==1) {
    electron_track.SetXYZ(px_track[0], py_track[0], pz_track[0]);
    rec_track = 1;
}

track_p = electron_track.Mag();
track_eta = electron_track.Eta();
track_phi = electron_track.Phi();
```

Still in loop over reader – …and calorimeter clusters

```
if(h_endcap_E.GetSize()>0)
    h_endcap_clust_E = h_endcap_E[0];
if(e_endcap_E.GetSize()>0)
    e_endcap_clust_E = e_endcap_E[0];

for(int icl = 0; icl < barrel_E.GetSize(); icl++) {
    barrel_clust_E += barrel_E[icl];
}
```

Still in loop over reader – …and calorimeter clusters

```
if(h_endcap_E.GetSize()>0)
    h_endcap_clust_E = h_endcap_E[0];
if(e_endcap_E.GetSize()>0)
    e_endcap_clust_E = e_endcap_E[0];

for(int icl = 0; icl < barrel_E.GetSize(); icl++) {
    barrel_clust_E += barrel_E[icl];
}
```

Already using merged endcap clusters

Manually sum clusters in barrel

Putting it all together

- See ROOT macro getSingleElectronRecon.C
  - Takes two TString arguments, input and output file names
- Gets MC truth info, reconstructed track, and calorimeter clusters
- Saves to output ROOT tree
- Resolutions easily obtained by plotting ratio of reconstructed/MC values

Analysis with Uproot

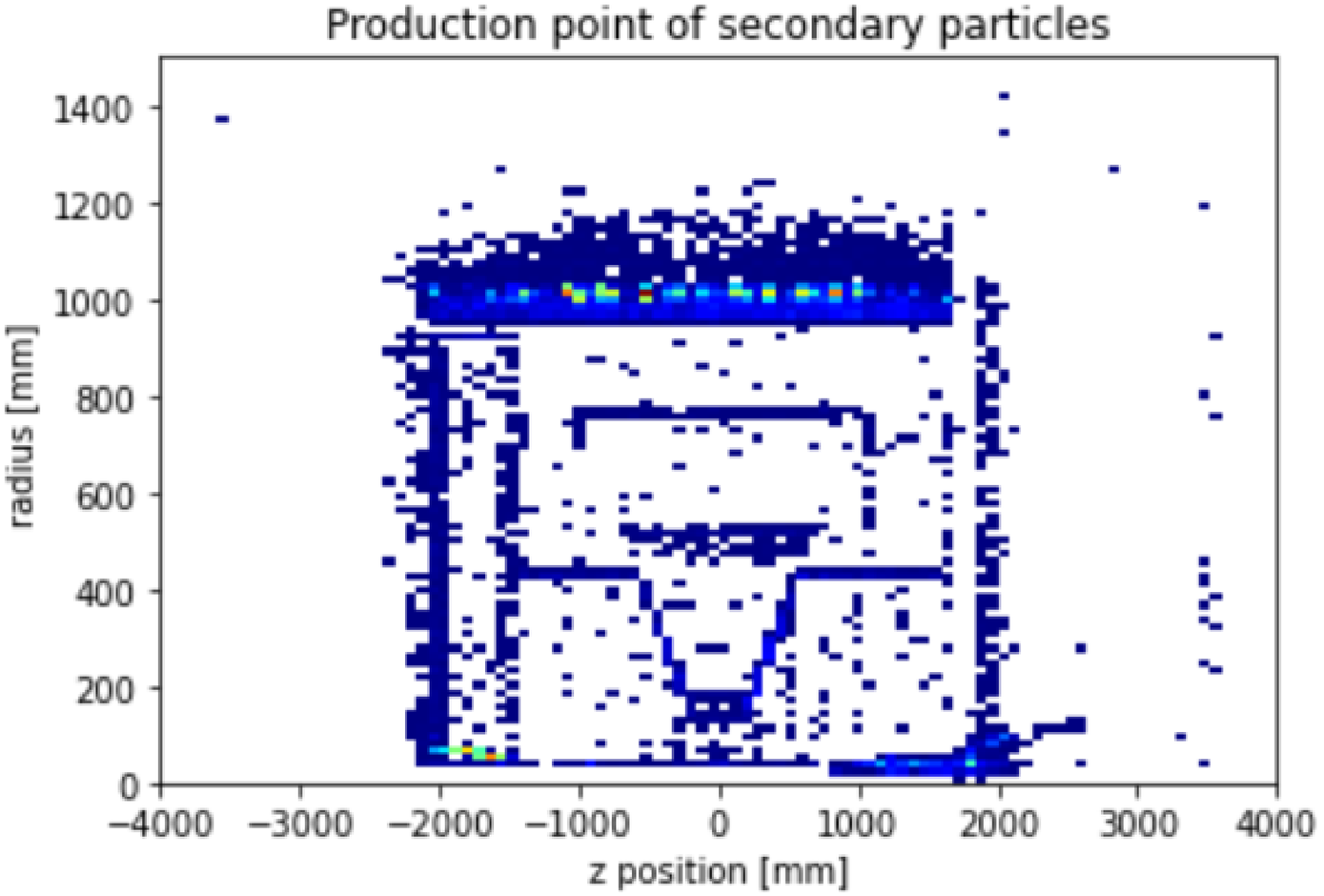
```python
import uproot
import awkward as ak
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

#load ROOT file and get MCParticles branches
events = uproot.open('input/output.edm4hep.root:events')

arrays = events.arrays(filter_name="MCParticles/*")
print("")
print(arrays.type)
print("")

#cuts to apply
cut_primary = arrays["MCParticles.generatorStatus"]==1
cut_secondary = arrays["MCParticles.generatorStatus"]==0

#primary particle analysis (we simulated single electron here)
print("Primary particle analysis...")

px = arrays["MCParticles.momentum.x"][cut_primary]
py = arrays["MCParticles.momentum.y"][cut_primary]
pz = arrays["MCParticles.momentum.z"][cut_primary]
mass = arrays["MCParticles.mass"][cut_primary]

pt = np.sqrt(px**2+py**2)
mom = np.sqrt(px**2+py**2+pz**2)
energy = np.sqrt(mom**2+mass**2)
theta = np.arccos(pz/mom)
eta = -1.*np.log( np.tan(theta/2.) )
```

```python
#secondary particles analysis
print("Secondary particles analysis...")

vx = arrays["MCParticles.vertex.x"][cut_secondary]
vy = arrays["MCParticles.vertex.y"][cut_secondary]
vz = arrays["MCParticles.vertex.z"][cut_secondary]
px = arrays["MCParticles.momentum.x"][cut_secondary]
py = arrays["MCParticles.momentum.y"][cut_secondary]
pz = arrays["MCParticles.momentum.z"][cut_secondary]
mass = arrays["MCParticles.mass"][cut_secondary]

pt = np.sqrt(px**2+py**2)
mom = np.sqrt(px**2+py**2+pz**2)
energy = np.sqrt(mom**2+mass**2)
theta = np.arccos(pz/mom)
eta = -1.*np.log( np.tan(theta/2.) )
radius = np.sqrt(vx**2+vy**2)
```

mcparticles_check.ipynb

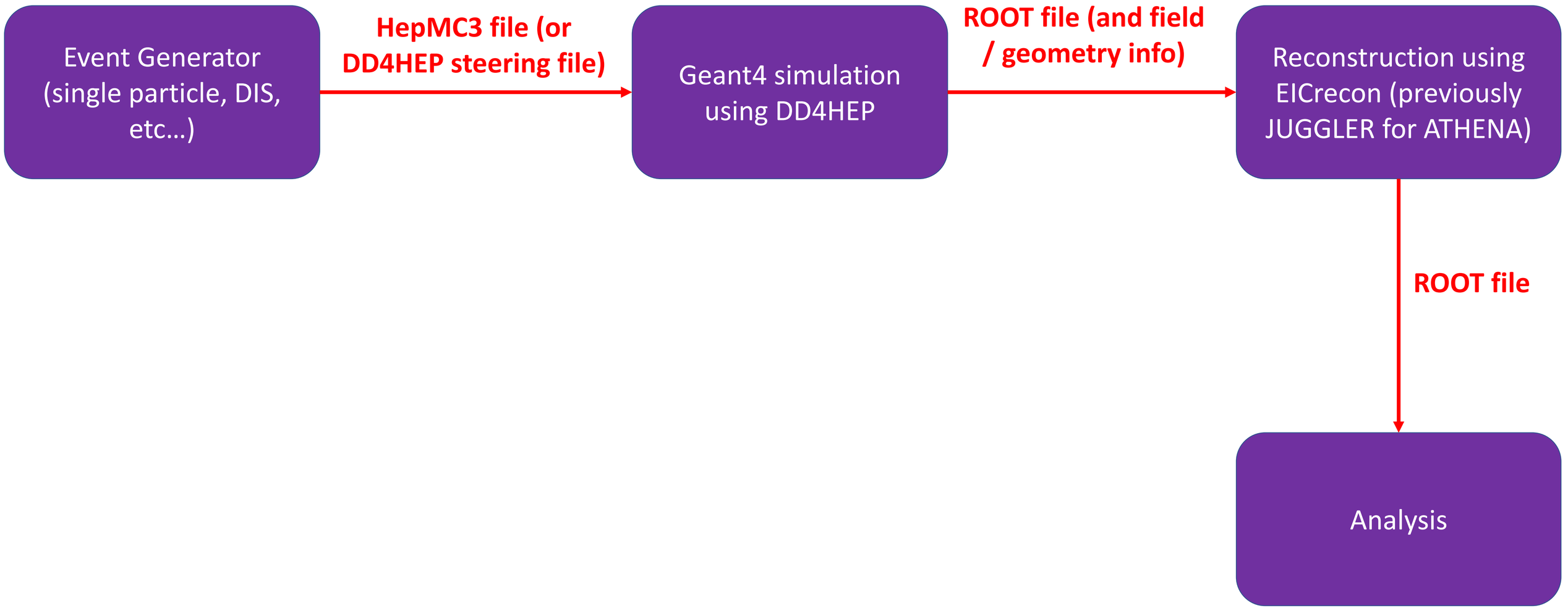
Analysis with Uproot

```
plt.hist2d(np.array(ak.flatten(vz)),np.array(ak.flatten(radius)),bins=(100,100),range=([-4e3,4e3],[0,1.5e3]),
           cmap=plt.cm.jet,cmin = 1)
plt.title("Production point of secondary particles")
plt.xlabel("z position [mm]")
plt.ylabel("radius [mm]")
plt.show()
```

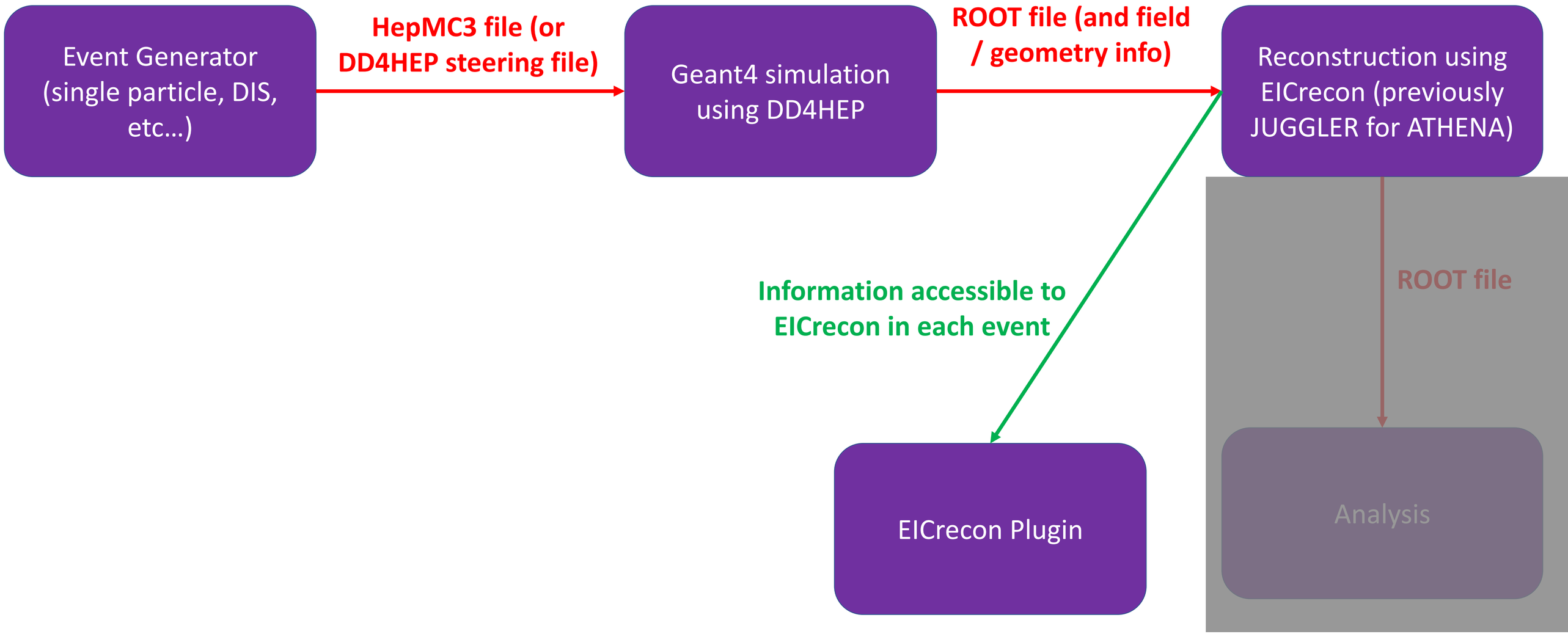


mcparticles_check.ipynb

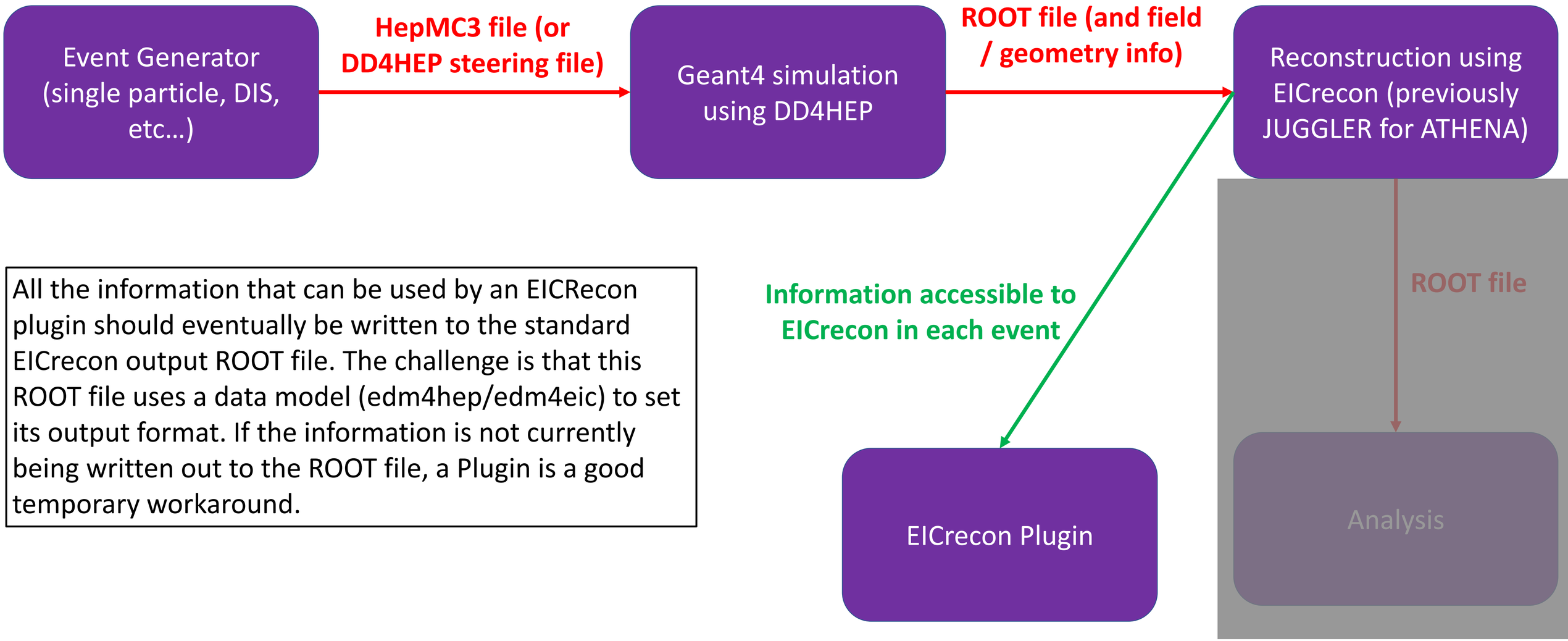
Simulation workflow for ePIC



Simulation workflow for ePIC



Simulation workflow for ePIC



All the information that can be used by an EICRecon plugin should eventually be written to the standard EICrecon output ROOT file. The challenge is that this ROOT file uses a data model (edm4hep/edm4eic) to set its output format. If the information is not currently being written out to the ROOT file, a Plugin is a good temporary workaround.

Plugin example

.gitignore	Updated Plugin
CMakeLists.txt	Added Plugin for track projection to EEMC
EemcTrkPropagation_processor.cc	Updated processor
EemcTrkPropagation_processor.h	Updated processor
Eemc_TrkPropagation.cc	Added Plugin for track projection to EEMC
mysteer.py	Updated processor
plot_hists.C	Updated processor
plot_hists.pdf	Updated processor
run_sim.sh	Updated processor

https://github.com/eic/EICrecon/tree/EEMCProj_Proc/src/tests/EEMCProjPlugin

Plugin example

```
//----------------------------
// Init
//
// This is called once before the first call to the Process method
// below. You may, for example, want to open an output file here.
// Only one thread will call this.
void Init() override;

//----------------------------
// Process
//
// This is called for every event. Multiple threads may call this
// simultaneously. If you write something to an output file here
// then make sure to protect it with a mutex or similar mechanism.
// Minimize what is done while locked since that directly affects
// the multi-threaded performance.
void Process(const std::shared_ptr<const JEvent>& event) override;

//----------------------------
// Finish
//
// This is called once after all events have been processed. You may,
// for example, want to close an output file here.
// Only one thread will call this.
void Finish() override;
```

```
//Histograms
TH2 *h1a; //Rec track momentum vs. true momentum
TH2 *h1b; //EEMC cluster E vs. true energy
TH1 *h2a; //Track projection z at EEMC
TH1 *h2b; //EEMC Cluster z position
TH2 *h3a; //EEMC Cluster x position vs. Track projection x
TH2 *h3b; //EEMC Cluster y position vs. Track projection y
TH1 *h4a; //EEMC Cluster x position minus Track projection x
TH1 *h4b; //EEMC Cluster y position minus Track projection y
TH1 *h5a; //EEMC cluster E divided by rec track momentum
```

Plugin example

```
// Get trajectories from tracking
auto trajectories = event->Get<eicrecon::TrackingResultTrajectory>("CentralCKFTrajectories");

// Iterate over trajectories
m_log->trace("Propagating through {} trajectories", trajectories.size());
for (size_t traj_index = 0; traj_index < trajectories.size(); traj_index++) {
    auto &trajectory = trajectories[traj_index];
    m_log->trace(" -- trajectory {} --", traj_index);

    edm4eic::TrackPoint* projection_point;
    try {
        // >>> try to propagate to surface <<<
        projection_point = m_propagation_algo.propagate(trajectory, m_ecal_surface);
    }
    catch(std::exception &e) {
        m_log->warn("Exception in underlying algorithm: {}. Trajectory is skipped", e.what());
    }
```

We call an algorithm (which uses the ACTS propagate class) to project the track fitted track to an arbitrary surface. This track projection is saved as an edm4eic:Trackpoint datatype – which is a data component and cannot currently be saved to the standard EICrecon TTree.

Plugin example