ROOT Universal Struct File

BiWeekly Software & Simulation Meeting

2/17/2021

Zulkaida, Dustin, Kenichi

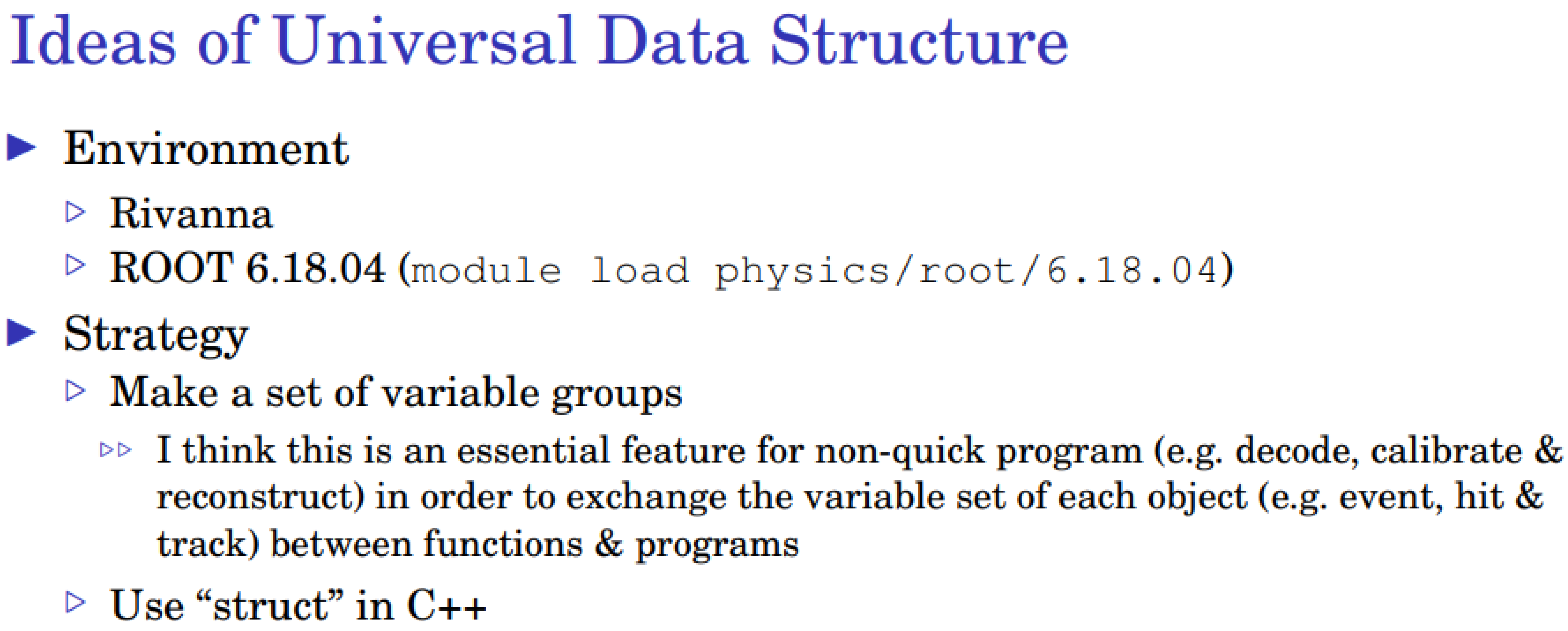
Outline

- ❑ Motivation
- ❑ Initial study
- ❑ Universal Struct format file
- ❑ Analysis example
- ❑ Summary & plan

Motivation

- Current file format (DST.root) requires external Fun4All library to be read
- We are experimenting with the new file structure in addition to the DST.root
- The key point is everyone can make quick analysis without external library (universal)
- One of the possibility is additional tree in DST.root contain variables which can be read without external library
- Additional requirement: compressible, easy to skim and store, not memory intensive, hit based info index, complete hit information, hold the truth information, …

Initial Study by Kenichi



Ideas of Universal Data Structure

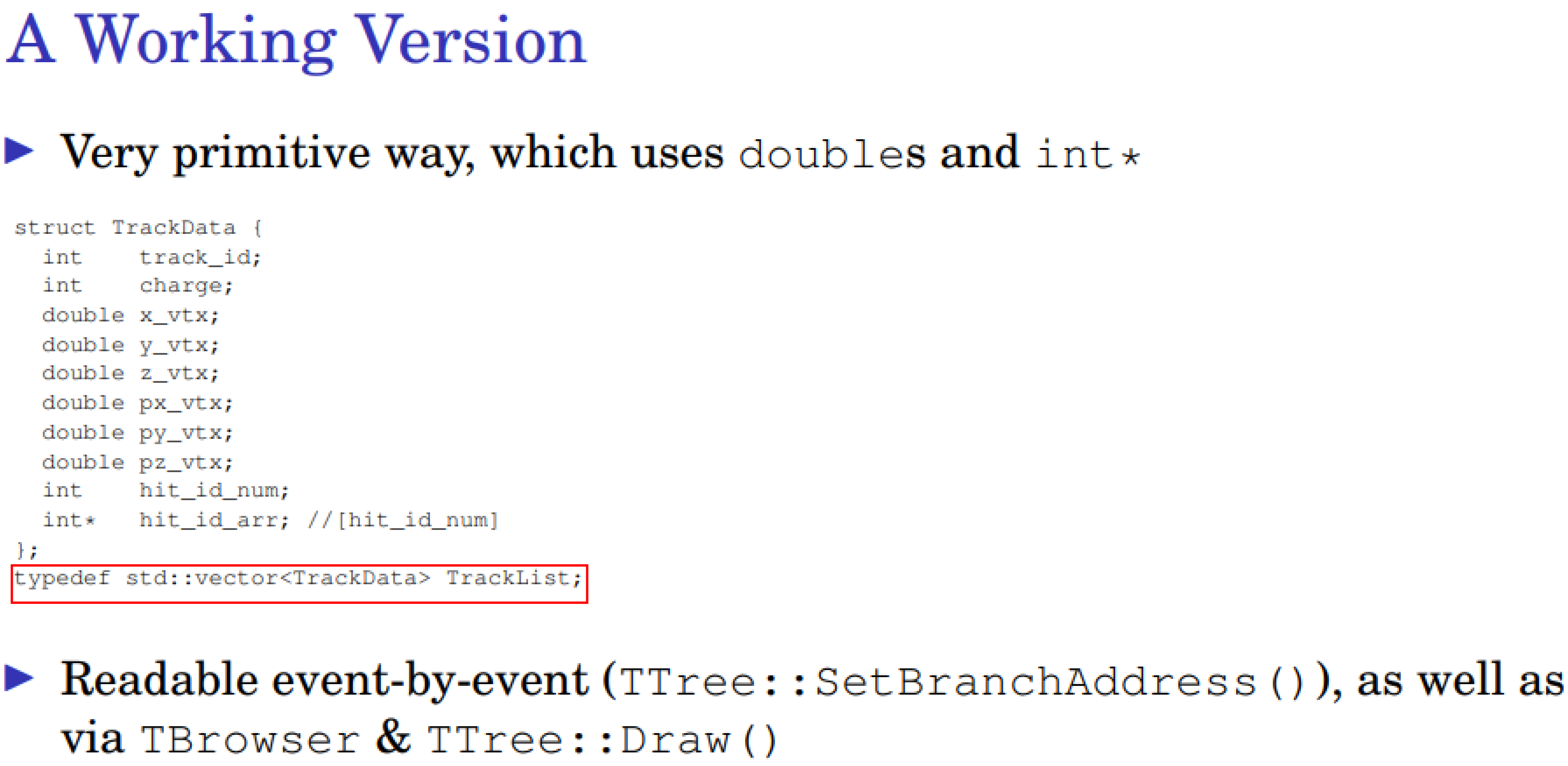
- Environment
  - Rivanna
  - ROOT 6.18.04 (`module load physics/root/6.18.04`)
- Strategy
  - Make a set of variable groups
    - I think this is an essential feature for non-quick program (e.g. decode, calibrate & reconstruct) in order to exchange the variable set of each object (e.g. event, hit & track) between functions & programs
  - Use "struct" in C++

Example of Data Structure

- Most natural way, which uses ROOT classes & `std::vector`

```
struct EventData {
  int  run_id;
  int  spill_id;
  int  event_id;
  bool fpga_bits[5];
  bool  nim_bits[5];
  int  reco_status;
};

struct HitData {
  int    hit_id;
  int    detector_id;
  int    element_id;
  double tdc_time;
  double drift_time;
  double drift_distance;
};
typedef std::vector<HitData> HitList;

struct TrackData {
  int            track_id;
  int            charge;
  TVector3       pos_vtx;
  TLorentzVector mom_vtx;
  std::vector<int> hit_id_list;
};
typedef std::vector<TrackData> TrackList;
```

- `TBrowser` & `TTree::Draw()` work fine
- Key problems:
  Event-by-event analysis (using `TTree::SetBranchAddress()`) cannot read
  - ROOT class (e.g. `TVector3`) in `vector` and
  - `vector` in `vector`
  - (When the data-structure library is not loaded)



A Working Version

- Very primitive way, which uses doubles and int*

```
struct TrackData {
  int    track_id;
  int    charge;
  double x_vtx;
  double y_vtx;
  double z_vtx;
  double px_vtx;
  double py_vtx;
  double pz_vtx;
  int    hit_id_num;
  int*   hit_id_arr; //[hit_id_num]
};
typedef std::vector<TrackData> TrackList;
```

- Readable event-by-event (TTree::SetBranchAddress()), as well as via TBrowser & TTree::Draw()

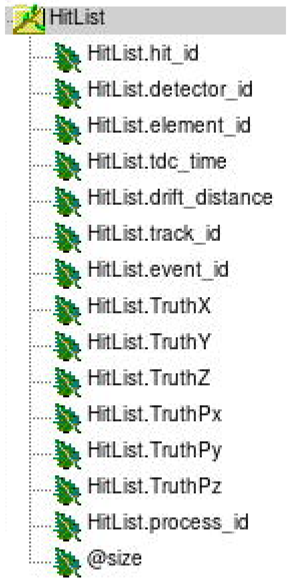
- Vector of struct which the struct members are simple variables can be read without external library
- The following slides show the example of this file format created by a “DST2Root module”

Universal Struct Format

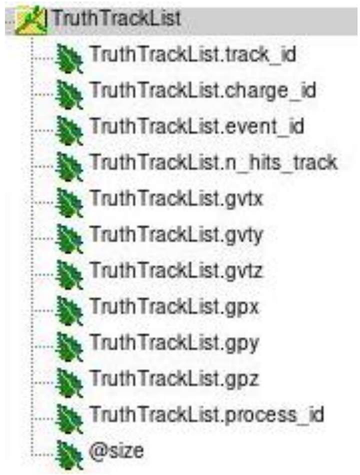
Branch:

- **HitList**: contains the basic hit information such as hit id, detector id, element id, track id, truth position, …
- **TruthTrackList**: contains the truth information of the track such as vertex and momentum
- **RecTrackList**: contains the reconstructed track information based on Kalman filter
- **TruthDimuon**: contains the dimuon truth information such as energy, momentum, xF, phi, ..
- **RecDimuon**: contains the reconstructed dimuon information based on Kalman filter
- **EventData**: contains general event information such as event id, number of truth tracks, number of reconstructed tracks, trigger, …
- **DetectorInfo**: currently contains detector position

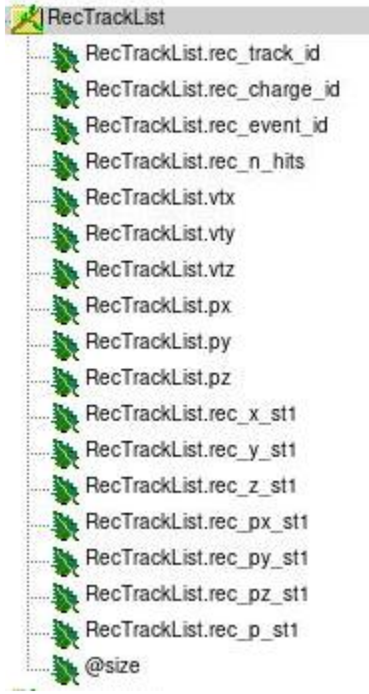
Notes: This version is far from final. More and more information will be added



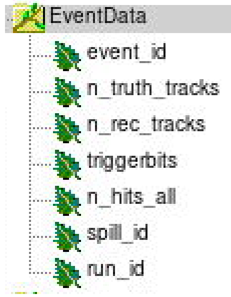
HitList
- HitList.hit_id
- HitList.detector_id
- HitList.element_id
- HitList.tdc_time
- HitList.drift_distance
- HitList.track_id
- HitList.event_id
- HitList.TruthX
- HitList.TruthY
- HitList.TruthZ
- HitList.TruthPx
- HitList.TruthPy
- HitList.TruthPz
- HitList.process_id
- @size



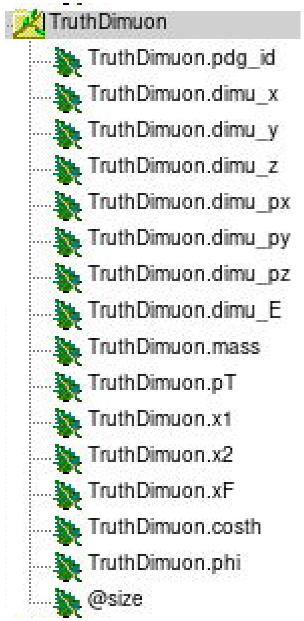
TruthTrackList
- TruthTrackList.track_id
- TruthTrackList.charge_id
- TruthTrackList.event_id
- TruthTrackList.n_hits_track
- TruthTrackList.gvtx
- TruthTrackList.gvty
- TruthTrackList.gvtz
- TruthTrackList.gpx
- TruthTrackList.gpy
- TruthTrackList.gpz
- TruthTrackList.process_id
- @size



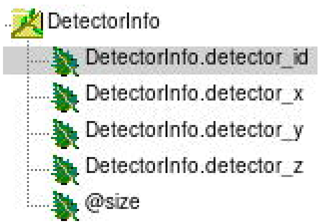
RecTrackList
- RecTrackList.rec_track_id
- RecTrackList.rec_charge_id
- RecTrackList.rec_event_id
- RecTrackList.rec_n_hits
- RecTrackList.vtx
- RecTrackList.vty
- RecTrackList.vtz
- RecTrackList.px
- RecTrackList.py
- RecTrackList.pz
- RecTrackList.rec_x_st1
- RecTrackList.rec_y_st1
- RecTrackList.rec_z_st1
- RecTrackList.rec_px_st1
- RecTrackList.rec_py_st1
- RecTrackList.rec_pz_st1
- RecTrackList.rec_p_st1
- @size



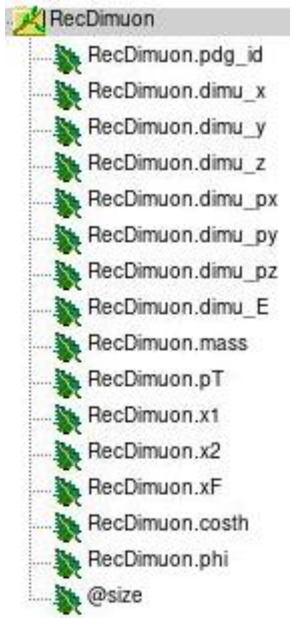
EventData
- event_id
- n_truth_tracks
- n_rec_tracks
- triggerbits
- n_hits_all
- spill_id
- run_id



TruthDimuon
- TruthDimuon.pdg_id
- TruthDimuon.dimu_x
- TruthDimuon.dimu_y
- TruthDimuon.dimu_z
- TruthDimuon.dimu_px
- TruthDimuon.dimu_py
- TruthDimuon.dimu_pz
- TruthDimuon.dimu_E
- TruthDimuon.mass
- TruthDimuon.pT
- TruthDimuon.x1
- TruthDimuon.x2
- TruthDimuon.xF
- TruthDimuon.costh
- TruthDimuon.phi
- @size



DetectorInfo
- DetectorInfo.detector_id
- DetectorInfo.detector_x
- DetectorInfo.detector_y
- DetectorInfo.detector_z
- @size



RecDimuon
- RecDimuon.pdg_id
- RecDimuon.dimu_x
- RecDimuon.dimu_y
- RecDimuon.dimu_z
- RecDimuon.dimu_px
- RecDimuon.dimu_py
- RecDimuon.dimu_pz
- RecDimuon.dimu_E
- RecDimuon.mass
- RecDimuon.pT
- RecDimuon.x1
- RecDimuon.x2
- RecDimuon.xF
- RecDimuon.costh
- RecDimuon.phi
- @size

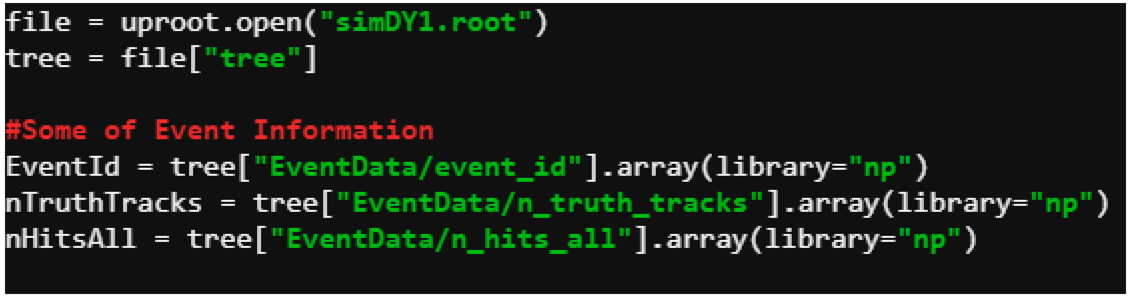
Example Analysis

- This file format could be read by ROOT C++ macro or Python
- In this example, I use Uproot which is a Python interface to read ROOT file

Python Uproot advantage:

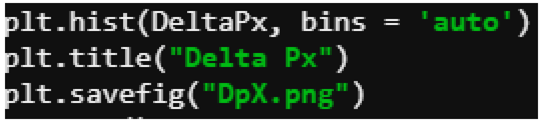
- Simple:

Reading



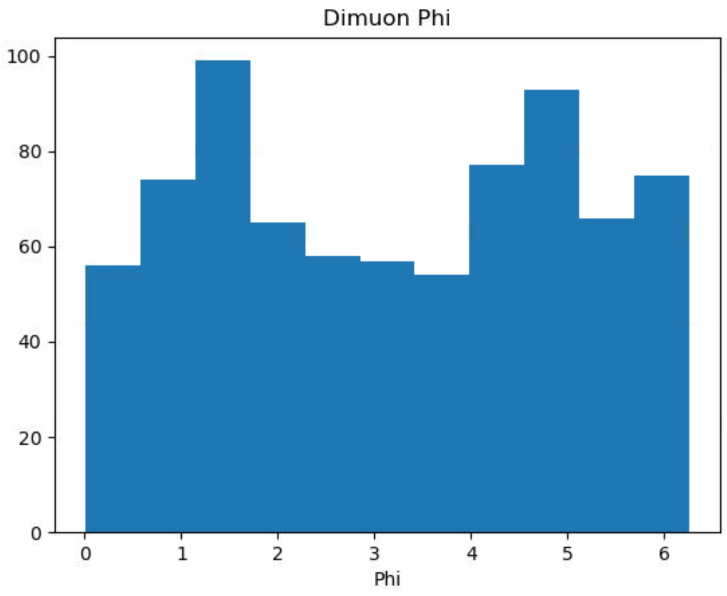
```
file = uproot.open("simDY1.root")
tree = file["tree"]

#Some of Event Information
EventId = tree["EventData/event_id"].array(library="np")
nTruthTracks = tree["EventData/n_truth_tracks"].array(library="np")
nHitsAll = tree["EventData/n_hits_all"].array(library="np")
```

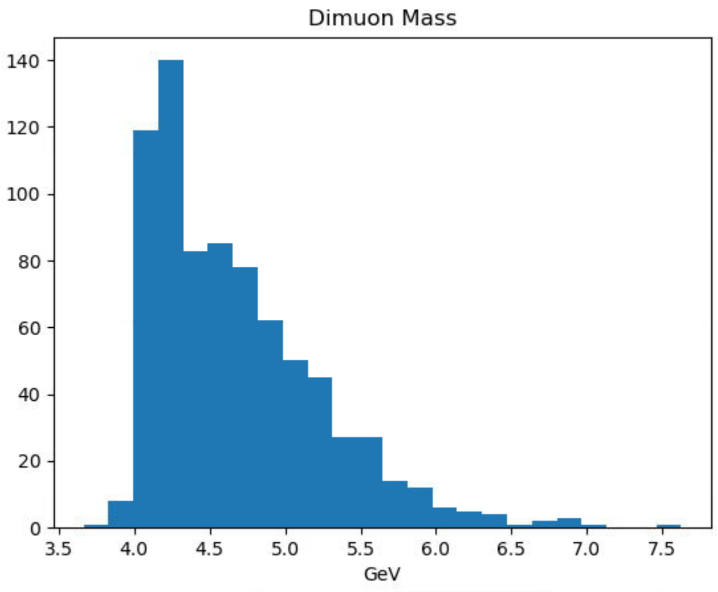
Plotting

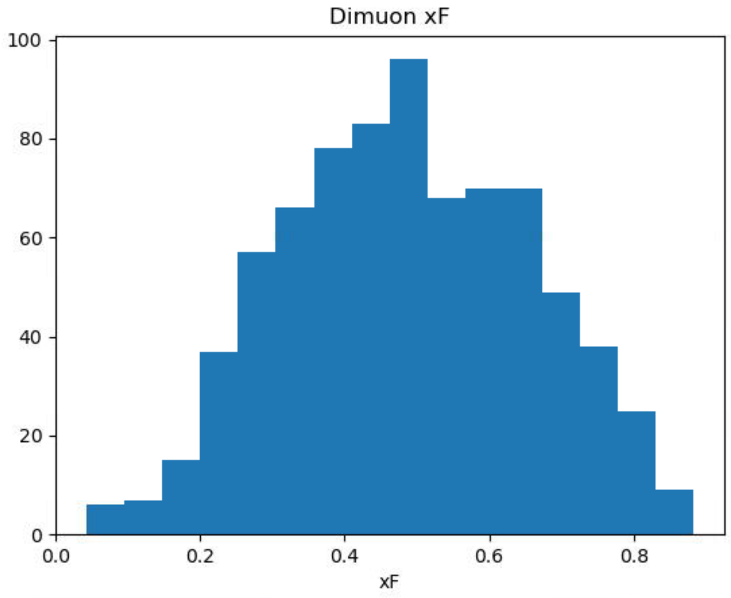


```
plt.hist(DeltaPx, bins = 'auto')
plt.title("Delta Px")
plt.savefig("DpX.png")
```

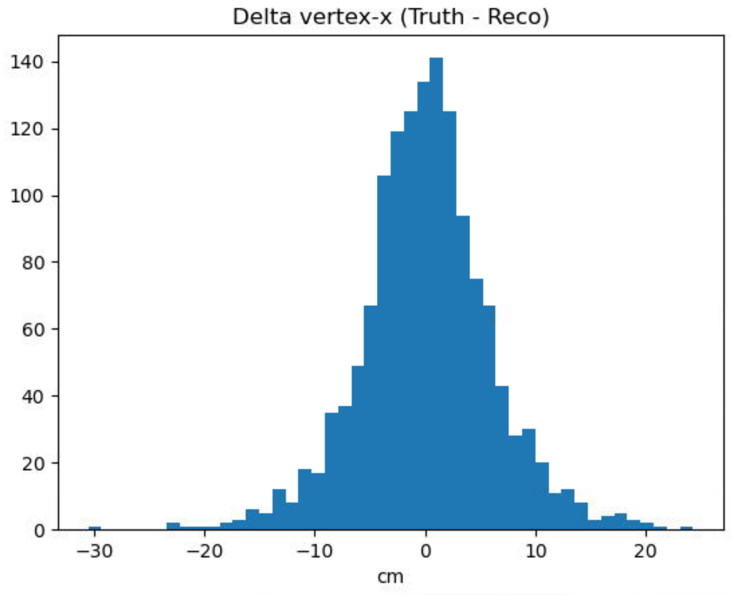
- Flexible: uproot provide interface to many data-processing library. The data from ROOT file can be retrieved as multi-dimensional array (Numpy) or Data Frame (Panda). Numpy provide access to various fitting routine while using Data Frame, we can apply simple queries to cut, select or filter (relational database) in comparison to C++ style which require multiple loop and if statements. The general advantage for using Python library is access to various data analysis and data visualization tools.


Dimuon Phi

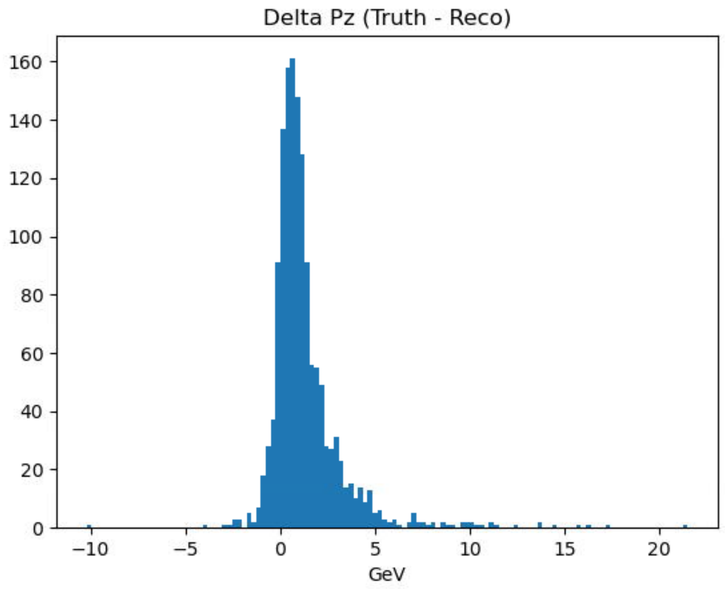

Dimuon Mass


Dimuon xF

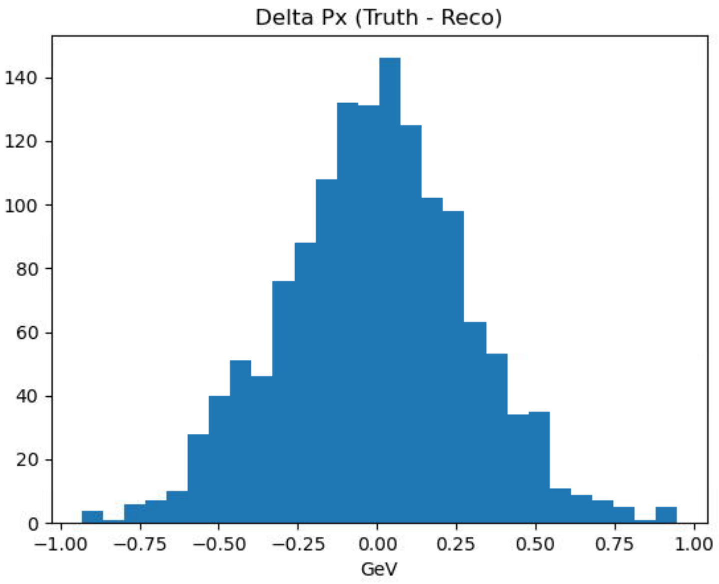
Example-1: Basic Dimuon Plot



Delta vertex-x (Truth - Reco)



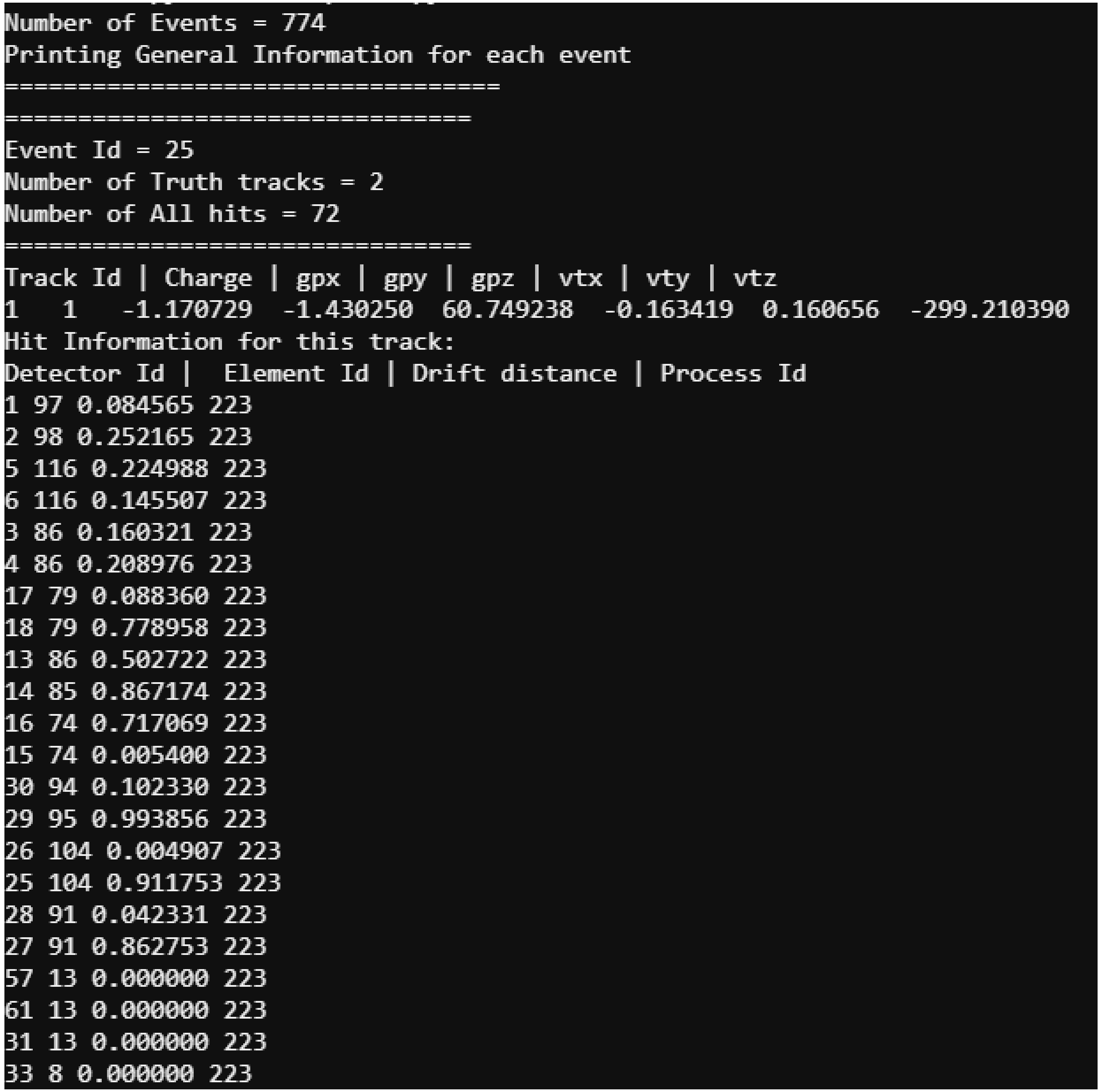
Delta Pz (Truth - Reco)



Delta Px (Truth - Reco)

Example-2: AnaTrackQA analysis

Example-3: Provide hit information for Neural-network training



```
Number of Events = 774
Printing General Information for each event
==================================
================================
Event Id = 25
Number of Truth tracks = 2
Number of All hits = 72
================================
Track Id | Charge | gpx | gpy | gpz | vtx | vty | vtz
1   1   -1.170729  -1.430250  60.749238  -0.163419  0.160656  -299.210390
Hit Information for this track:
Detector Id |  Element Id | Drift distance | Process Id
1 97 0.084565 223
2 98 0.252165 223
5 116 0.224988 223
6 116 0.145507 223
3 86 0.160321 223
4 86 0.208976 223
17 79 0.088360 223
18 79 0.778958 223
13 86 0.502722 223
14 85 0.867174 223
16 74 0.717069 223
15 74 0.005400 223
30 94 0.102330 223
29 95 0.993856 223
26 104 0.004907 223
25 104 0.911753 223
28 91 0.042331 223
27 91 0.862753 223
57 13 0.000000 223
61 13 0.000000 223
31 13 0.000000 223
33 8 0.000000 223
```

Conclusion & Plan

- We need a universal ROOT file that can be accessed without external library, easy to skim, complete hit information and store and not memory intensive.
- Vector of struct which the struct members are simple variables can be read without external library
- The file can be read & analyzed using ROOT C++ or Python interface
- Next step: completing the file with more struct and information