PID Algorithms and Data Model in the ePIC Software Stack

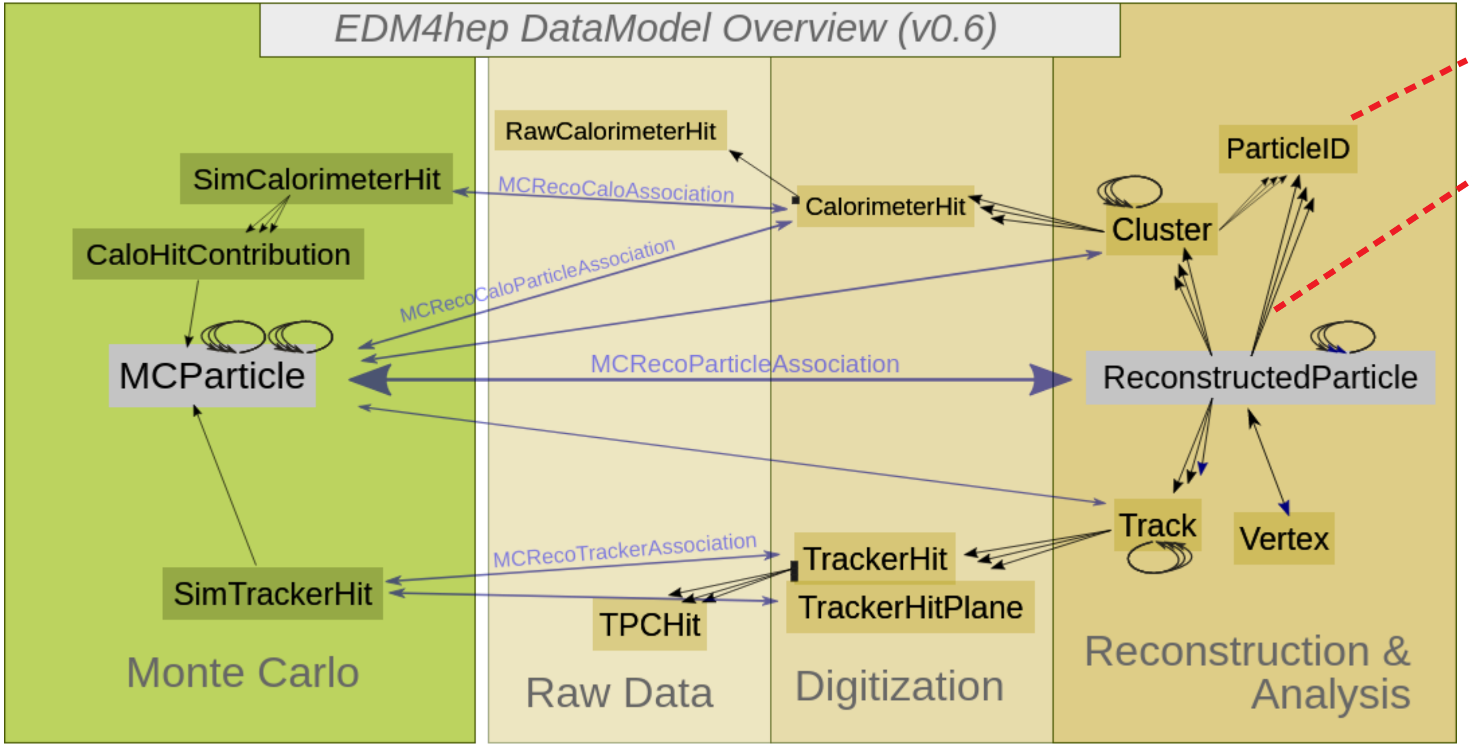
Christopher Dilks

PID Cross-Cutting Meeting

26 May 2023

Event Data Model (EDM) – at the General level

- **EDM4hep: https://github.com/key4hep/EDM4hep**
  - General data model shared by several HEP experiments



- "**ParticleID**" is the main datatype for PID
- One-to-many relation from "**ReconstructedParticle**" datatype to "**ParticleID**"

In output ROOT files, each **datatype** becomes a **TTree branch**

(can also use a PODIO frame reader)

EDM – at the EIC level

- **EDM4eic: https://github.com/eic/EDM4eic**
  - Experiment-specific data model; extends EDM4hep
  - Allows deviations from EDM4hep, where needed
    - e.g.: edm4hep::ReconstructedParticle vs. edm4eic::ReconstructedParticle

To view the data models, see the YAML files:

- EDM4hep: https://github.com/key4hep/EDM4hep/blob/master/edm4hep.yaml
- EDM4eic: https://github.com/eic/EDM4eic/blob/main/edm4eic.yaml

Reconstruction Framework Fundamentals

🔷 **Collection**

- A set of objects, such as “digitized hits”, or “PID hypotheses”
- Defined as “datatype” in the Event Data Model (EDM)

🔷 **Algorithm**

- An algorithm transforms collection(s) into collection(s)
- Examples:
  - Digitizer
    - Input: truth-level simulated hits
    - Output: digitized raw hits

- Algorithms should be:
  - Configurable – allow (external) configuration to tune for specific use cases or subsystems
  - Focused – don’t write a monolith
  - Shareable – some algorithms can be useful for multiple subsystems
  - Not dependent on EICrecon or JANA2 – Modularity
  - See Sylvester’s CHEP 2023 talk

Reconstruction Framework Fundamentals – Details

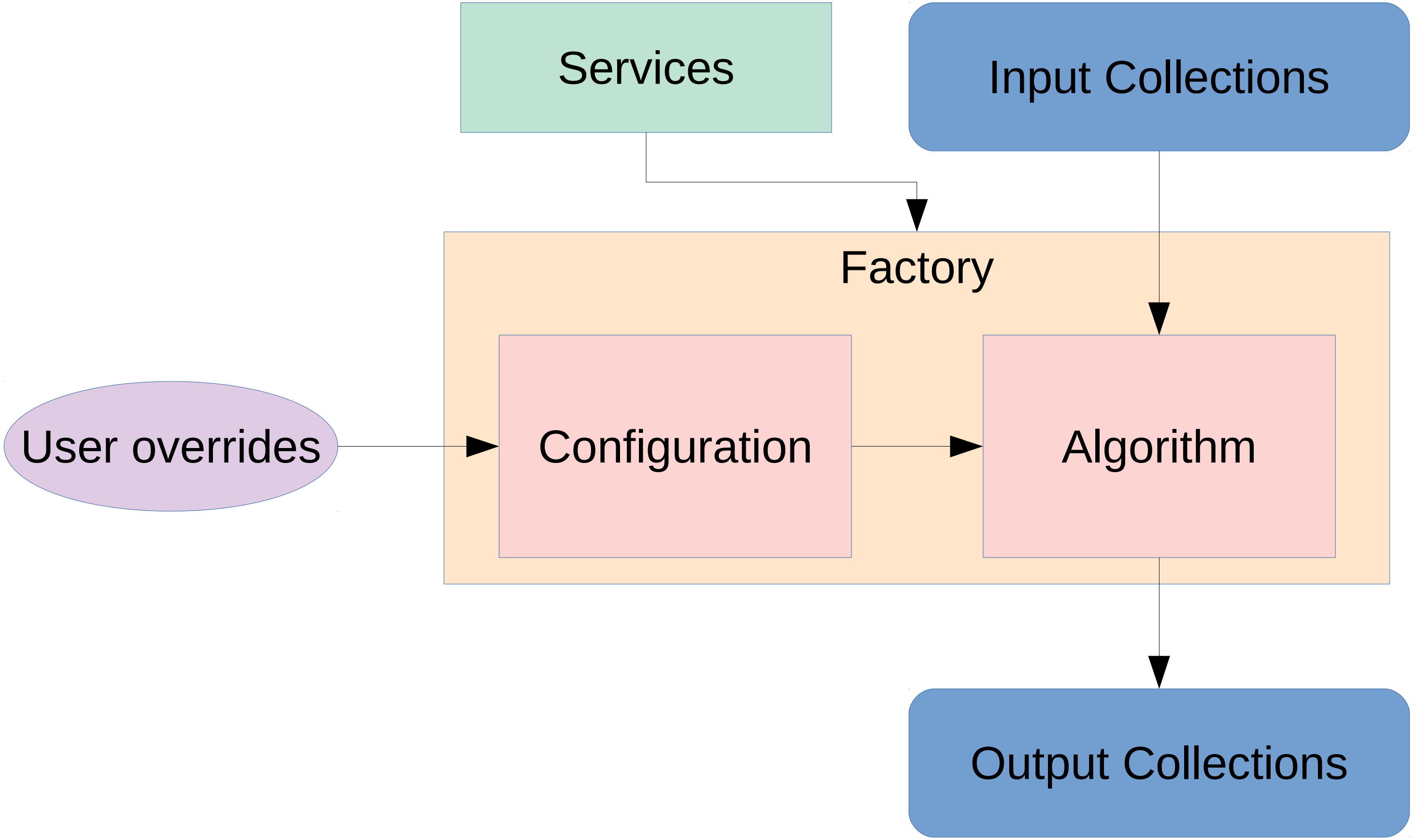
- **Algorithm Configuration**
  - Configuration parameters used to control and tune an algorithm
  - Can be changed externally from EICrecon
  - Example: two subsystems may use the same algorithm, but have different settings, e.g., a threshold

- **Factory**
  - EICrecon code that:
    - Sends input collections to an algorithm
    - Runs that algorithm
    - Knows how to handle the output collections

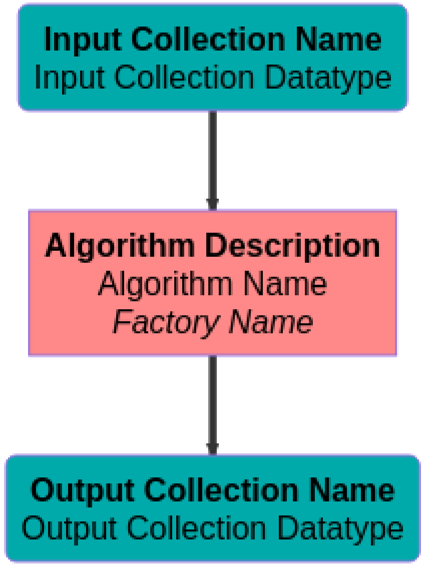
- **Services**
  - Common things
    - DD4hep geometry
    - Logger service
    - File I/O

- **Detector Plugin**: Runs it all!

Reconstruction Framework Fundamentals – Details



Algorithms & Collections → The Building Blocks

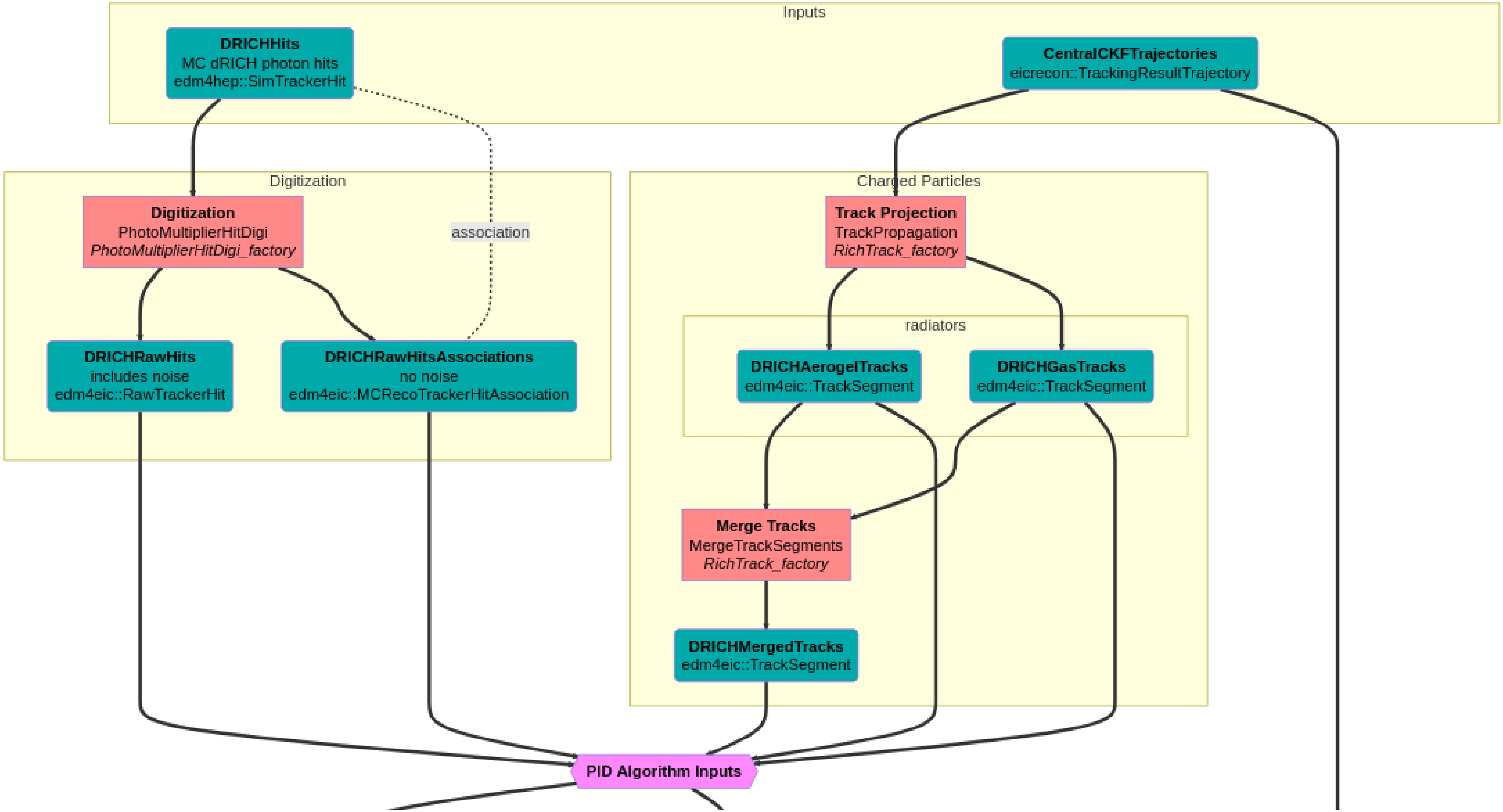


Input Collection Name
Input Collection Datatype

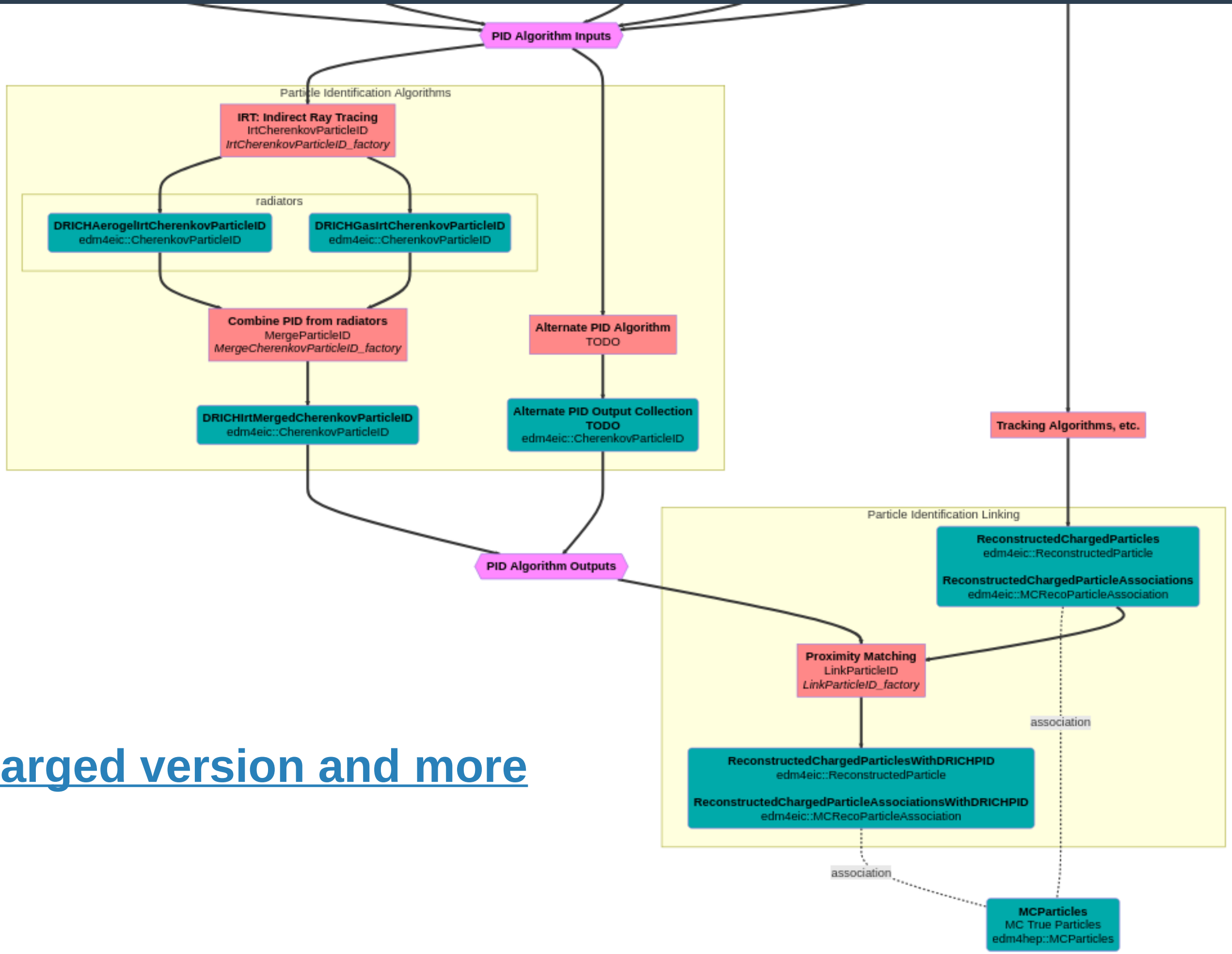
Algorithm Description
Algorithm Name
Factory Name

Output Collection Name
Output Collection Datatype

dRICH PID Plugin: Algorithm Flowchart – Part 1 of 2



dRICH PID Plugin: Algorithm Flowchart – Part 2 of 2



PID Algorithm Inputs

Particle Identification Algorithms

IRT: Indirect Ray Tracing
IrtCherenkovParticleID
IrtCherenkovParticleID_factory

radiators

DRICHAerogelIrtCherenkovParticleID
edm4eic::CherenkovParticleID

DRICHGasIrtCherenkovParticleID
edm4eic::CherenkovParticleID

Combine PID from radiators
MergeParticleID
MergeCherenkovParticleID_factory

DRICHIrtMergedCherenkovParticleID
edm4eic::CherenkovParticleID

Alternate PID Algorithm
TODO

Alternate PID Output Collection
TODO
edm4eic::CherenkovParticleID

Tracking Algorithms, etc.

PID Algorithm Outputs

Particle Identification Linking

ReconstructedChargedParticles
edm4eic::ReconstructedParticle

ReconstructedChargedParticleAssociations
edm4eic::MCRecoParticleAssociation

Proximity Matching
LinkParticleID
LinkParticleID_factory

association

ReconstructedChargedParticlesWithDRICHPID
edm4eic::ReconstructedParticle

ReconstructedChargedParticleAssociationsWithDRICHPID
edm4eic::MCRecoParticleAssociation

association

MCParticles
MC True Particles
edm4hep::MCParticles

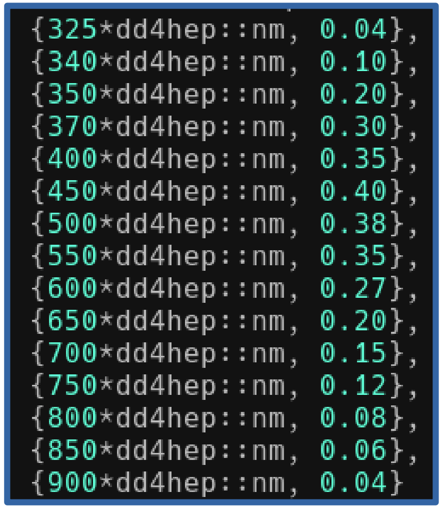
Click here for enlarged version and more

Algorithm: Digitization

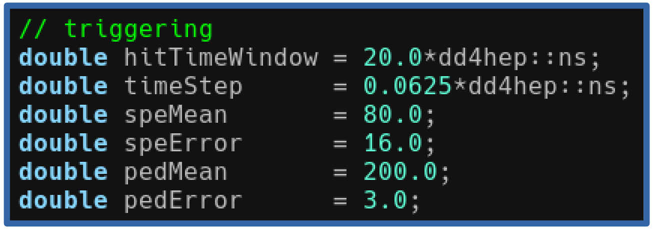
Common PMT Digitizer Algorithm

- Trigger parameters (gate, pedestal, etc.)
- Quantum Efficiency
- Empirical Safety Factor 70%
- Pixel Gap cuts (~88% survive)
- Noise injection (in progress, almost ready!)
- **TODO:** Time over Threshold (ToT)
- **TODO:** Refine configuration parameters

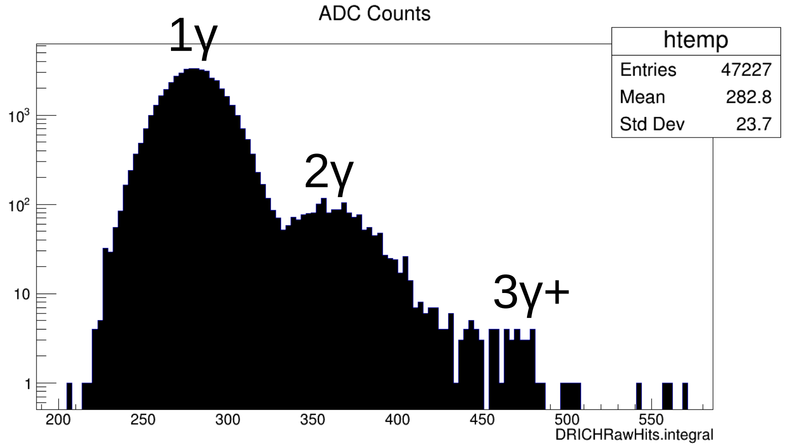
λ QE



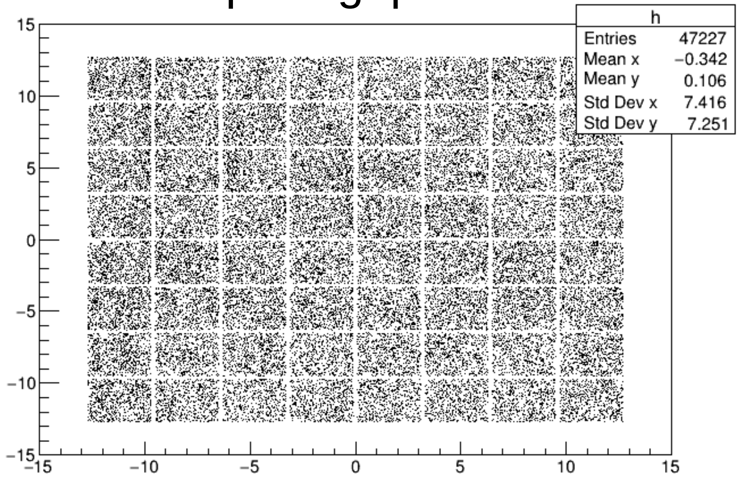
```
{325*dd4hep::nm, 0.04},
{340*dd4hep::nm, 0.10},
{350*dd4hep::nm, 0.20},
{370*dd4hep::nm, 0.30},
{400*dd4hep::nm, 0.35},
{450*dd4hep::nm, 0.40},
{500*dd4hep::nm, 0.38},
{550*dd4hep::nm, 0.35},
{600*dd4hep::nm, 0.27},
{650*dd4hep::nm, 0.20},
{700*dd4hep::nm, 0.15},
{750*dd4hep::nm, 0.12},
{800*dd4hep::nm, 0.08},
{850*dd4hep::nm, 0.06},
{900*dd4hep::nm, 0.04}
```



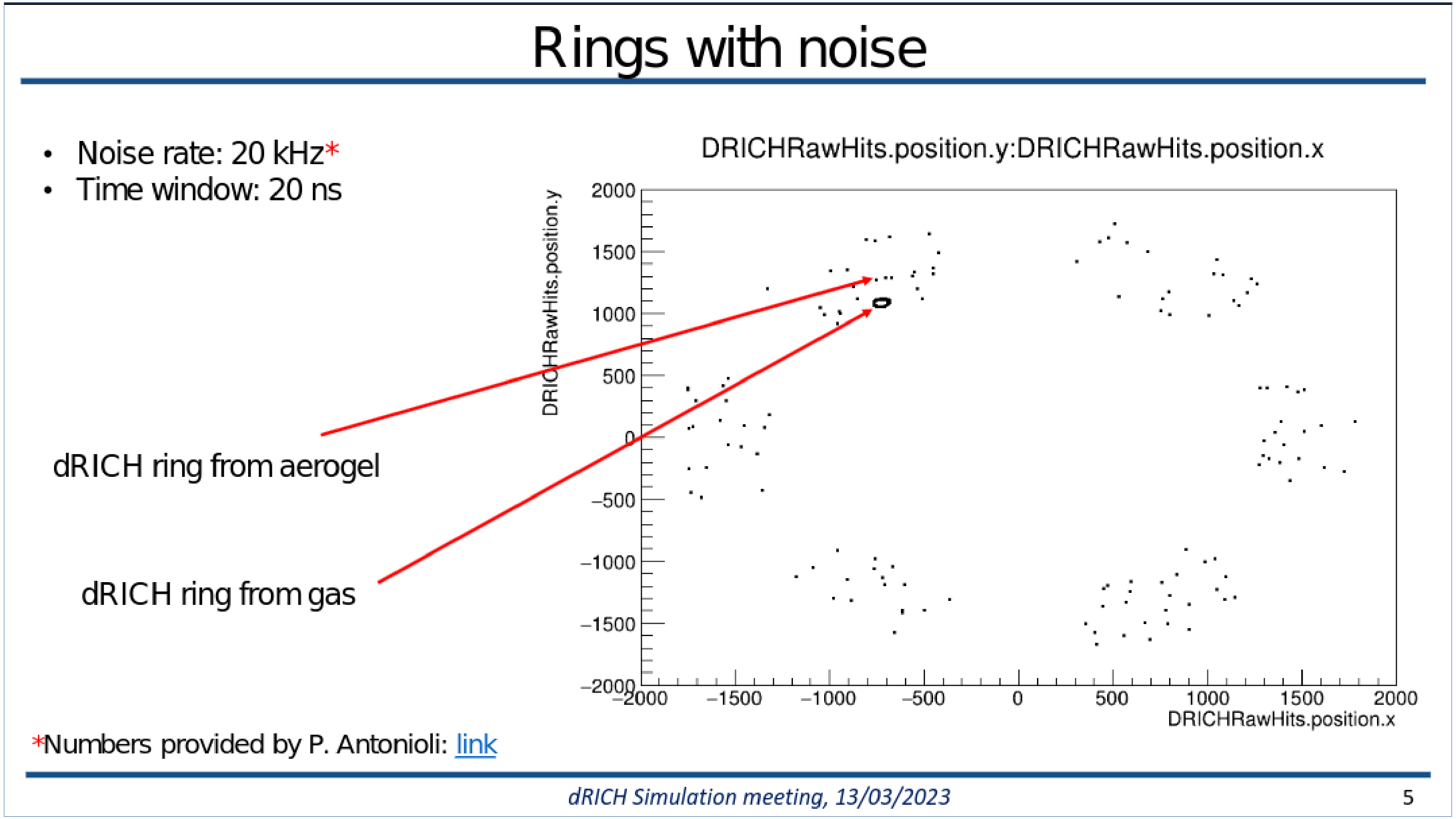
```
// triggering
double hitTimeWindow = 20.0*dd4hep::ns;
double timeStep      = 0.0625*dd4hep::ns;
double speMean       = 80.0;
double speError      = 16.0;
double pedMean       = 200.0;
double pedError      = 3.0;
```



SiPM pixel gaps



Algorithm: Digitization – Noise Injection



Rings with noise

- Noise rate: 20 kHz*
- Time window: 20 ns

dRICH ring from aerogel

dRICH ring from gas

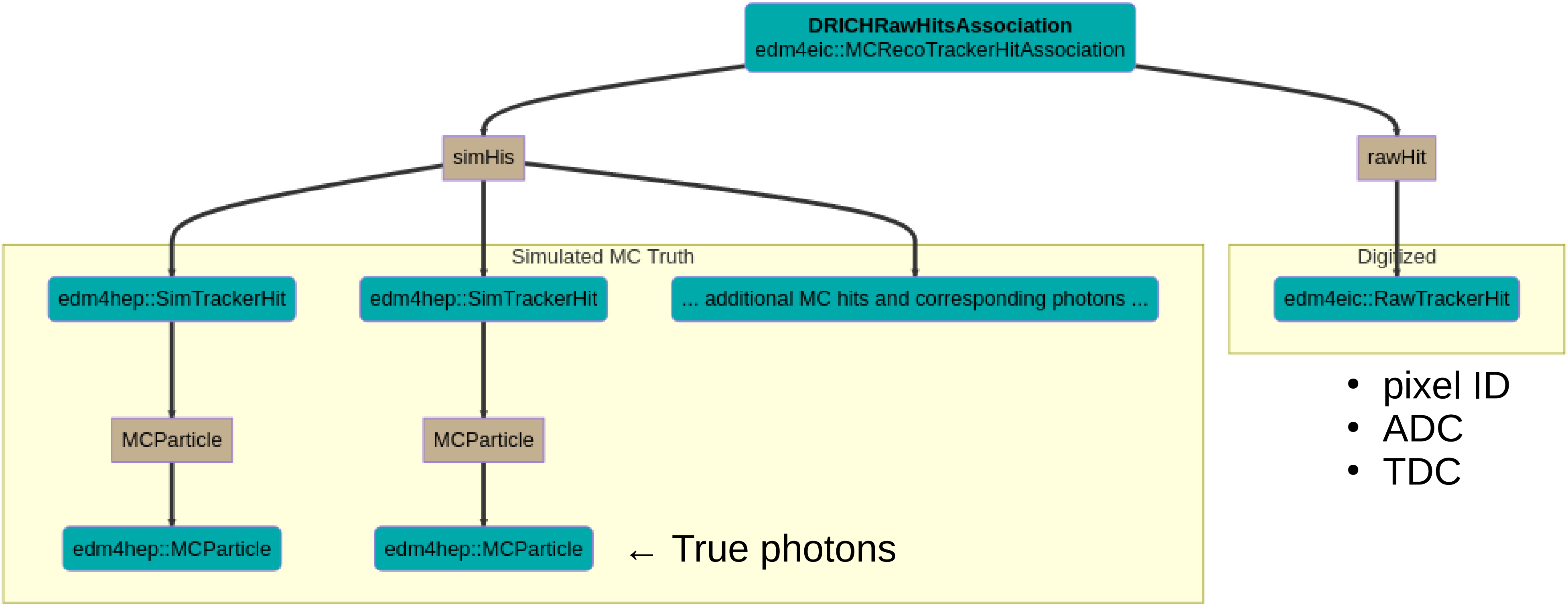
*Numbers provided by P. Antonioli: link

dRICH Simulation meeting, 13/03/2023

5

Slide from Luigi Dello Stritto

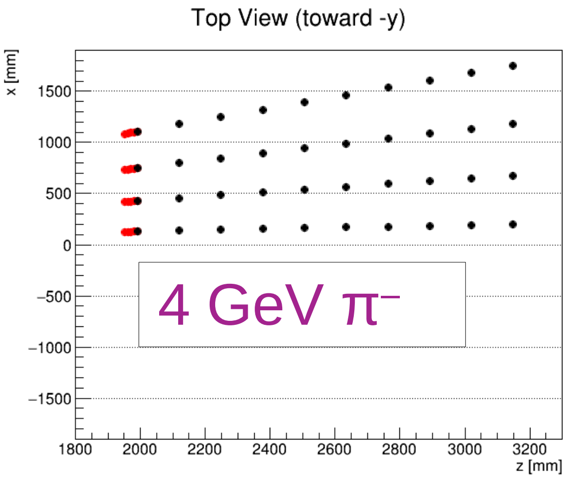
Data Model: Digitized Hits

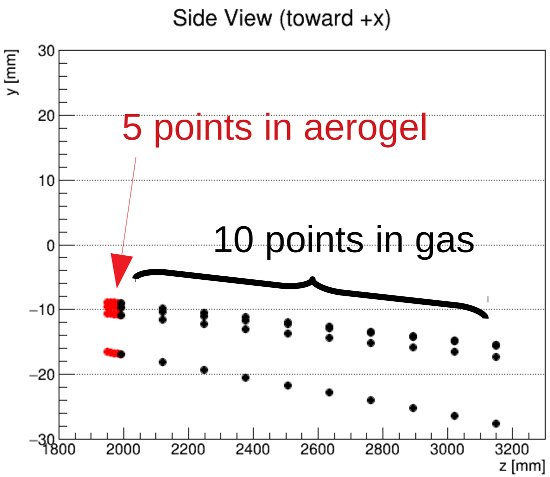


← True photons

- pixel ID
- ADC
- TDC

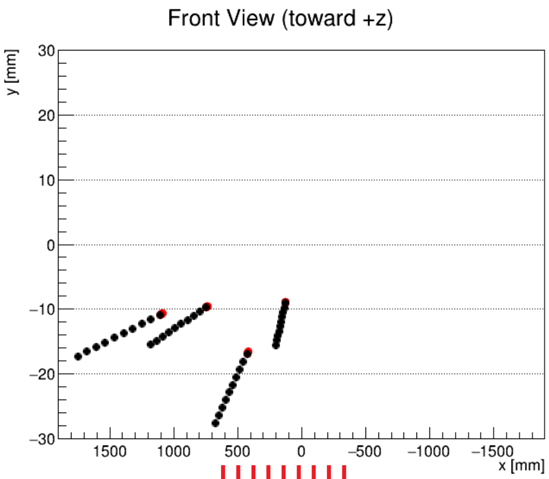
Algorithm: Charged Particle Track Projection


Top View (toward -y)

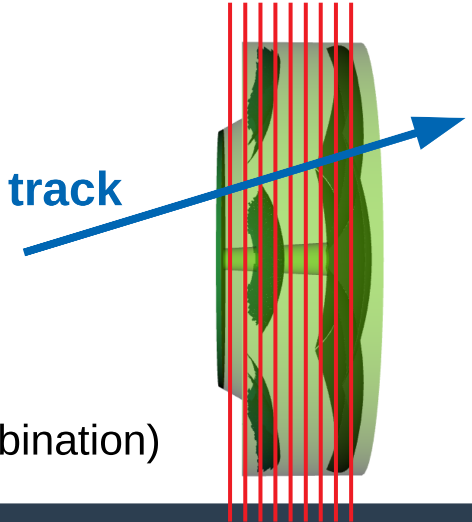

Side View (toward +x)

5 points in aerogel

10 points in gas

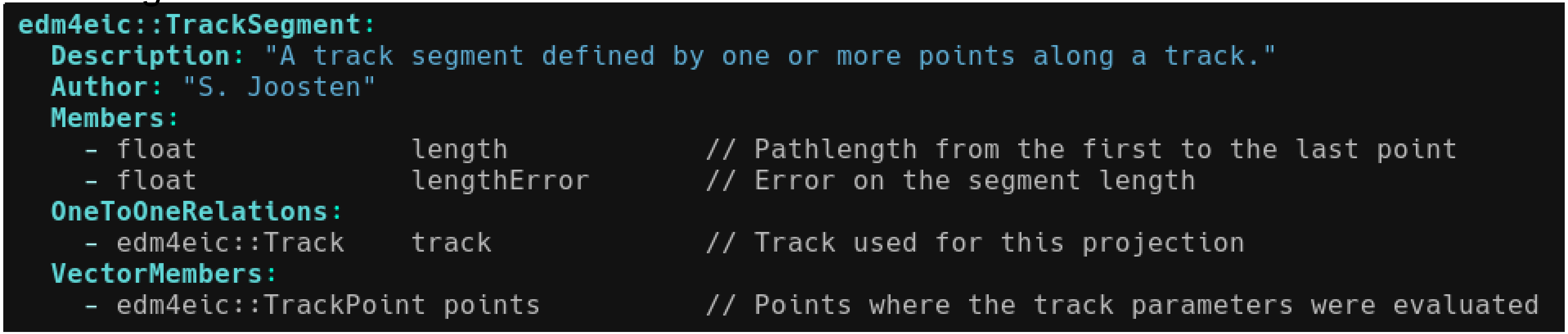

Front View (toward +z)

- Example: 4 GeV pions in horizontal y=0 plane
- Propagate to xy-planes in the dRICH radiators
  - 5 planes in aerogel
  - 10 planes in gas
- **<u>Reconstructed</u>** track points in **Aerogel** and **Gas** (and the merged combination)



Data Model: Charged Particle Track Points

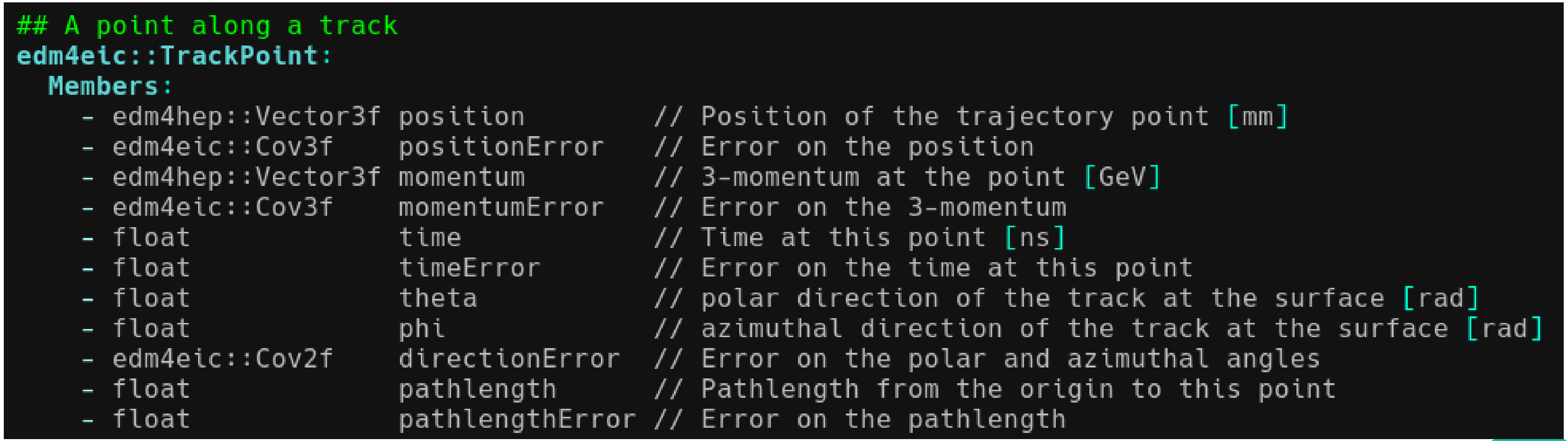
TrackSegment: a set of TrackPoints



```
edm4eic::TrackSegment:
  Description: "A track segment defined by one or more points along a track."
  Author: "S. Joosten"
  Members:
    - float             length            // Pathlength from the first to the last point
    - float             lengthError       // Error on the segment length
  OneToOneRelations:
    - edm4eic::Track    track             // Track used for this projection
  VectorMembers:
    - edm4eic::TrackPoint points          // Points where the track parameters were evaluated
```

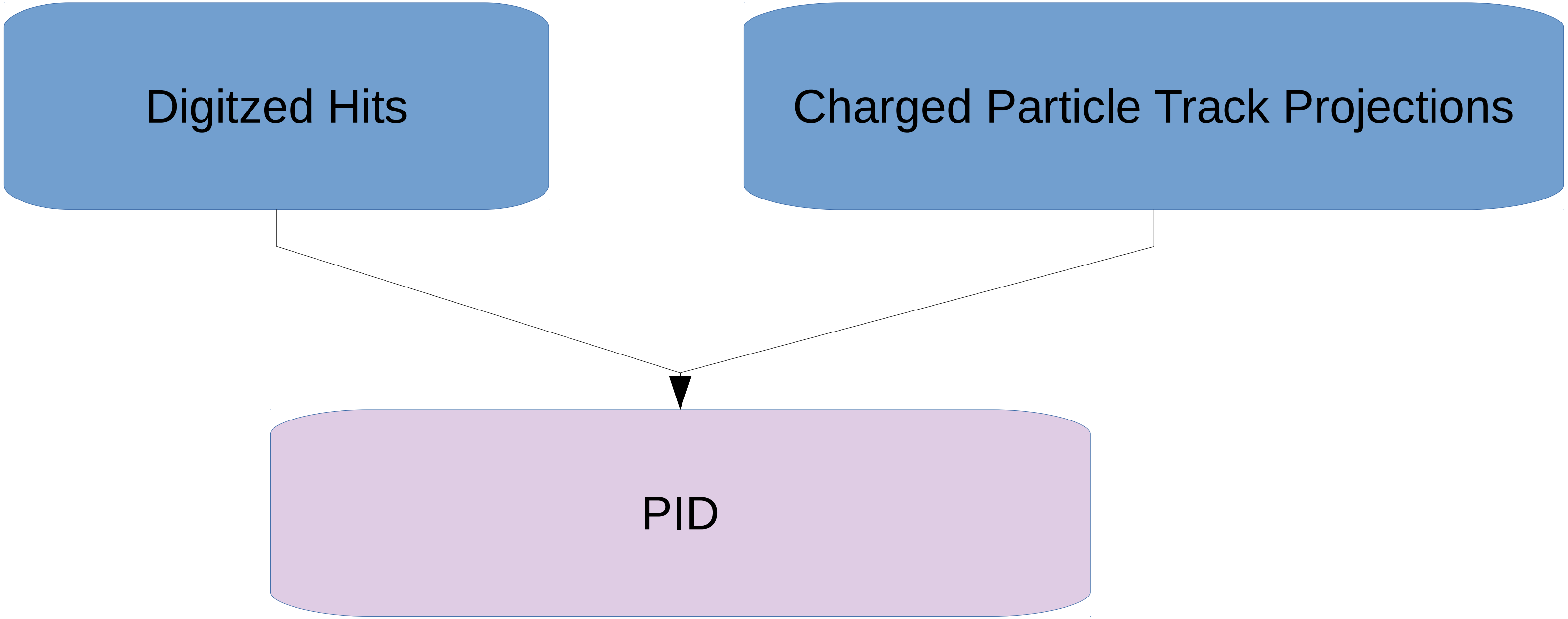
TrackPoints: the projected points:

- position
- momentum
- time
- and more



```
## A point along a track
edm4eic::TrackPoint:
  Members:
    - edm4hep::Vector3f position        // Position of the trajectory point [mm]
    - edm4eic::Cov3f    positionError   // Error on the position
    - edm4hep::Vector3f momentum        // 3-momentum at the point [GeV]
    - edm4eic::Cov3f    momentumError   // Error on the 3-momentum
    - float             time            // Time at this point [ns]
    - float             timeError       // Error on the time at this point
    - float             theta           // polar direction of the track at the surface [rad]
    - float             phi             // azimuthal direction of the track at the surface [rad]
    - edm4eic::Cov2f    directionError  // Error on the polar and azimuthal angles
    - float             pathlength      // Pathlength from the origin to this point
    - float             pathlengthError // Error on the pathlength
```

PID Algorithm: Inputs

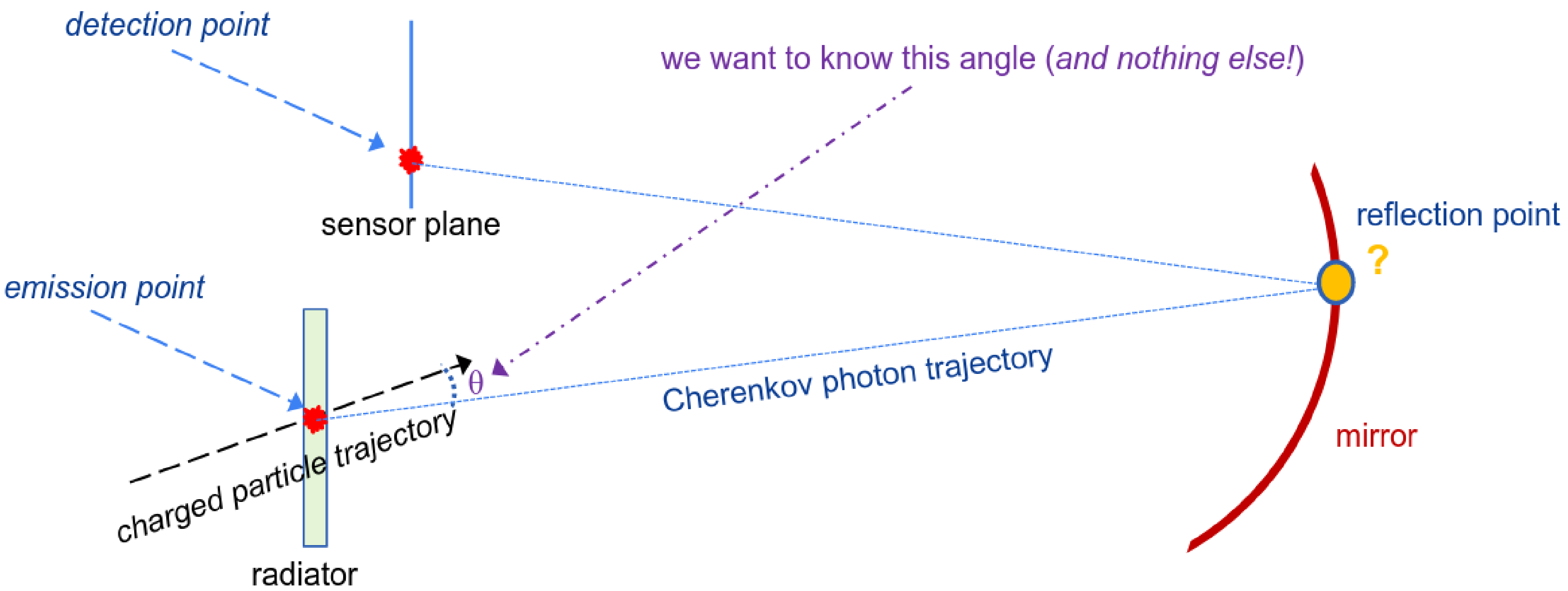


Digitzed Hits

Charged Particle Track Projections

PID

PID Algorithm: Indirect Ray Tracing (IRT)


detection point
sensor plane
we want to know this angle (*and nothing else!*)
reflection point
?
emission point
θ
Cherenkov photon trajectory
mirror
charged particle trajectory
radiator

- Given sensor hits and optics, determine the photon emission angle, sampled along a charged particle trajectory
- Newton-Gauss iterative solver for optical path
- Compact, standalone library used for Geant4 and ATHENA
- Interfaced with EICrecon (and Juggler) for ePIC

https://github.com/eic/irt

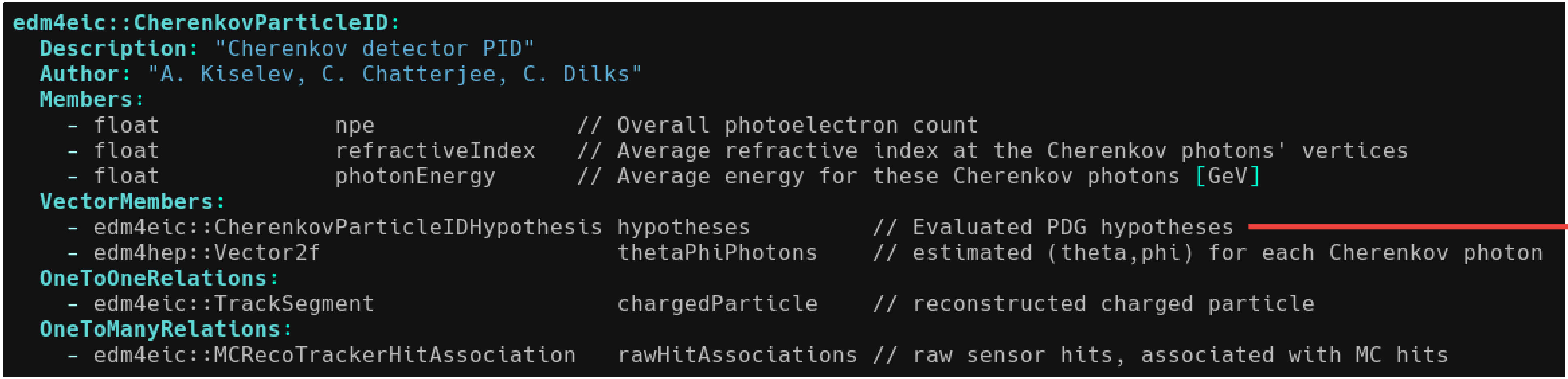
Figures from Alexander Kiselev, From meeting on RICH Pattern Recognition Challenges
https://agenda.infn.it/event/30966/

PID Algorithm: Alternatives

- To be integrated with EICrecon
- **The doors are open for development & integration!**
  - Inputs are available
  - Handling of outputs (mostly) implemented

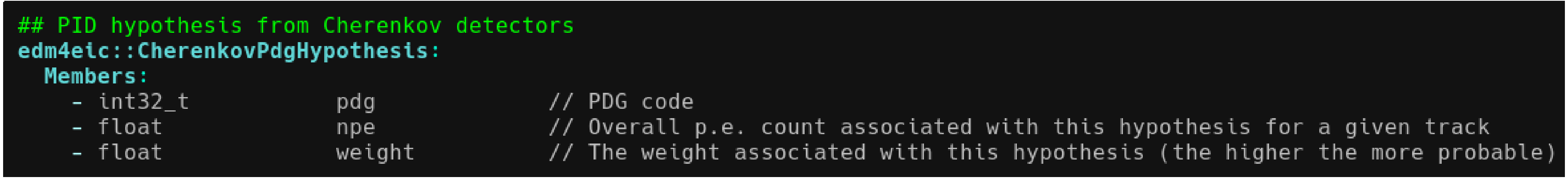
Data Model: Cherenkov PID

CherenkovParticleID datatype



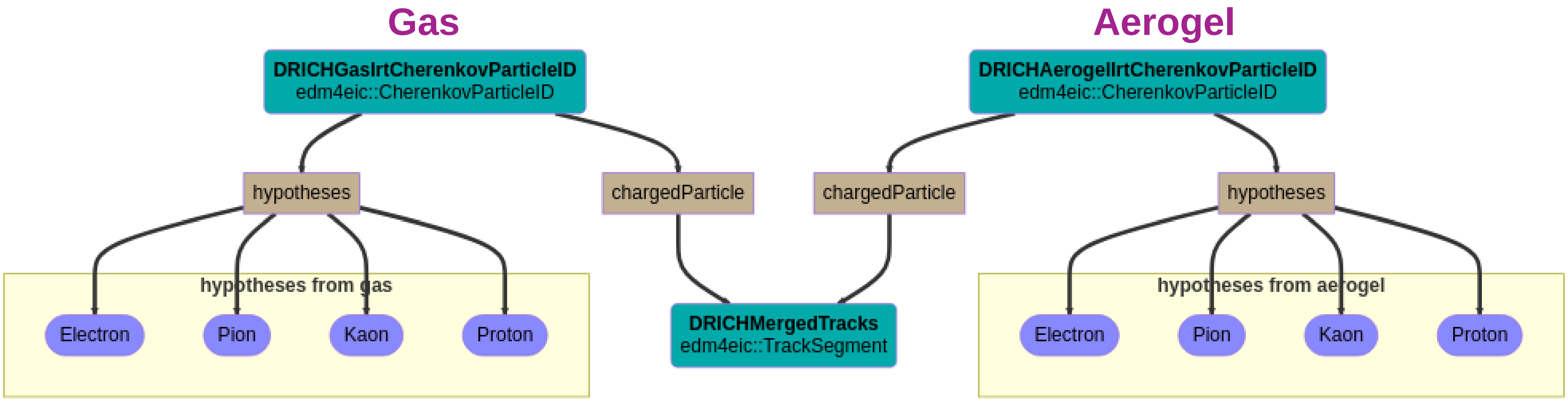
```
edm4eic::CherenkovParticleID:
  Description: "Cherenkov detector PID"
  Author: "A. Kiselev, C. Chatterjee, C. Dilks"
  Members:
    - float             npe                // Overall photoelectron count
    - float             refractiveIndex    // Average refractive index at the Cherenkov photons' vertices
    - float             photonEnergy       // Average energy for these Cherenkov photons [GeV]
  VectorMembers:
    - edm4eic::CherenkovParticleIDHypothesis hypotheses         // Evaluated PDG hypotheses
    - edm4hep::Vector2f                      thetaPhiPhotons    // estimated (theta,phi) for each Cherenkov photon
  OneToOneRelations:
    - edm4eic::TrackSegment                  chargedParticle    // reconstructed charged particle
  OneToManyRelations:
    - edm4eic::MCRecoTrackerHitAssociation   rawHitAssociations // raw sensor hits, associated with MC hits
```

CherenkovPdgHypothesis component: one for each PDG (mass) hypothesis:



```
## PID hypothesis from Cherenkov detectors
edm4eic::CherenkovPdgHypothesis:
  Members:
    - int32_t           pdg                // PDG code
    - float             npe                // Overall p.e. count associated with this hypothesis for a given track
    - float             weight             // The weight associated with this hypothesis (the higher the more probable)
```

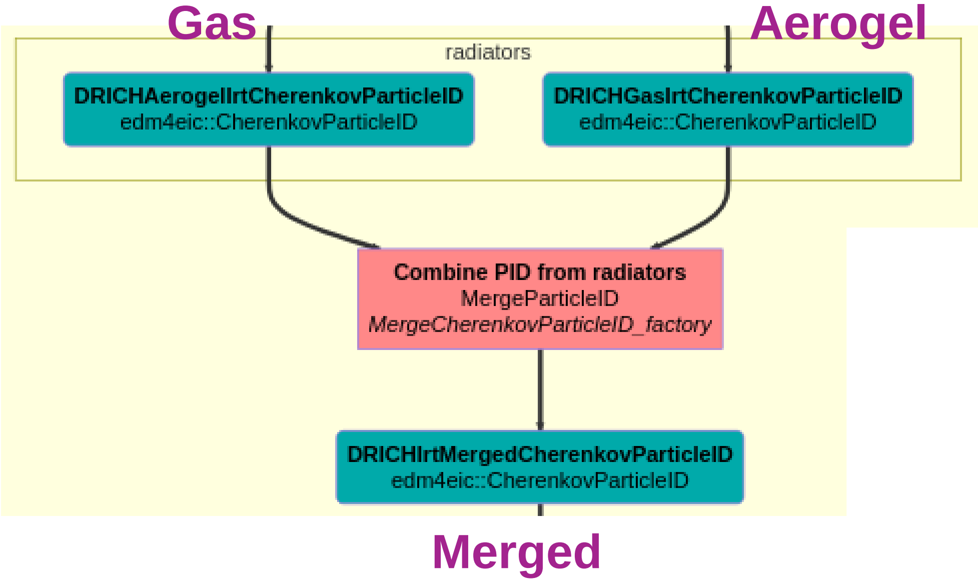
Data Model: Cherenkov PID



- One for each radiator (and one for the merged combination)
- All point to the same TrackSegment (as a unique ID)
- This is the “expert-level” PID object, specific for CherenkovPID

Algorithm: Merging Cherenkov PID Objects

- Simple Particle ID object merging implemented
- Currently handles merging dRICH gas + aerogel
- Could be generalized to merge PID objects from various subsystems
  - See Oskar’s and Markus’s talks (next) for combination strategies



Gas

Aerogel

radiators

DRICHAerogelIrtCherenkovParticleID
edm4eic::CherenkovParticleID

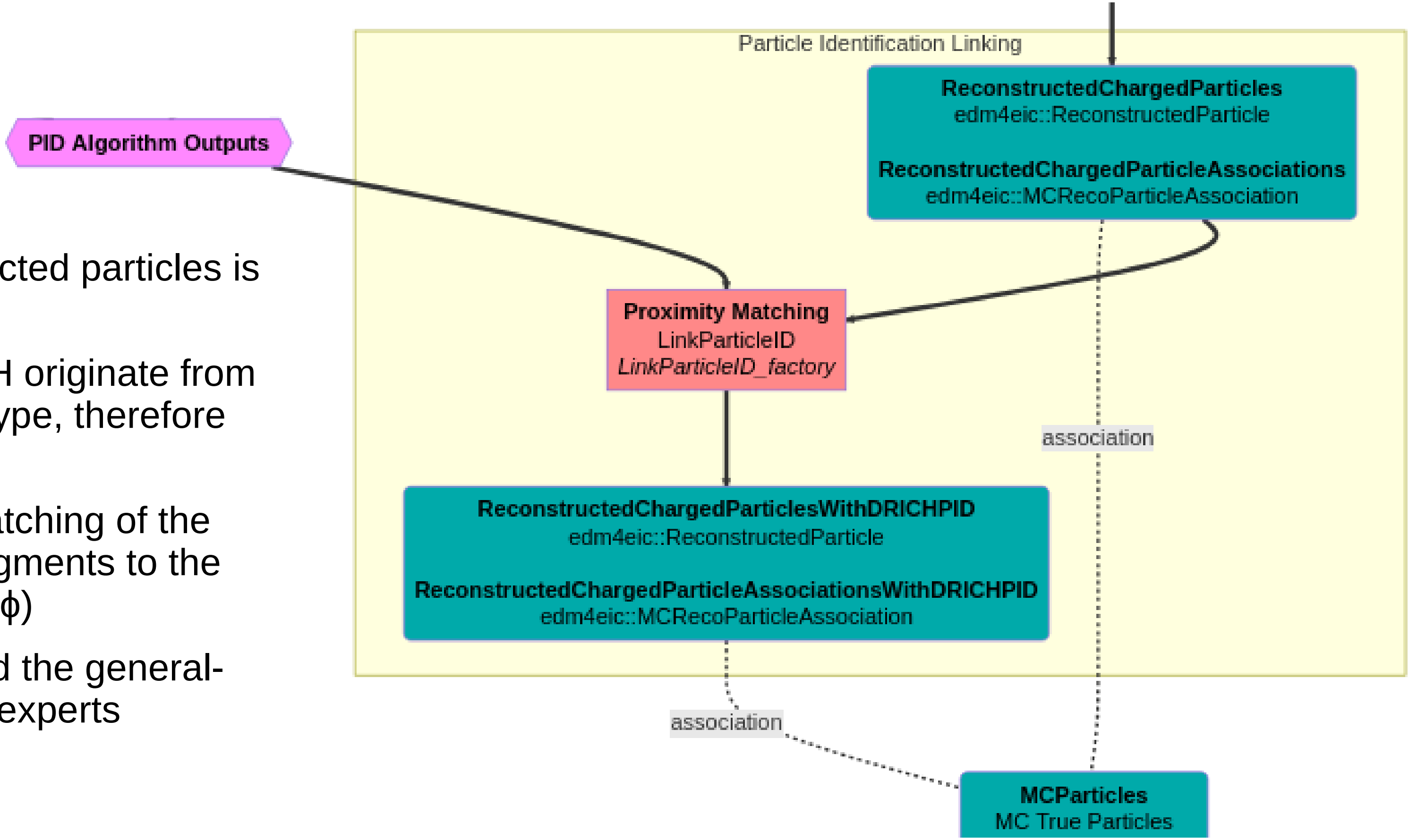
DRICHGasIrtCherenkovParticleID
edm4eic::CherenkovParticleID

Combine PID from radiators
MergeParticleID
MergeCherenkovParticleID_factory

DRICHIrtMergedCherenkovParticleID
edm4eic::CherenkovParticleID

Merged

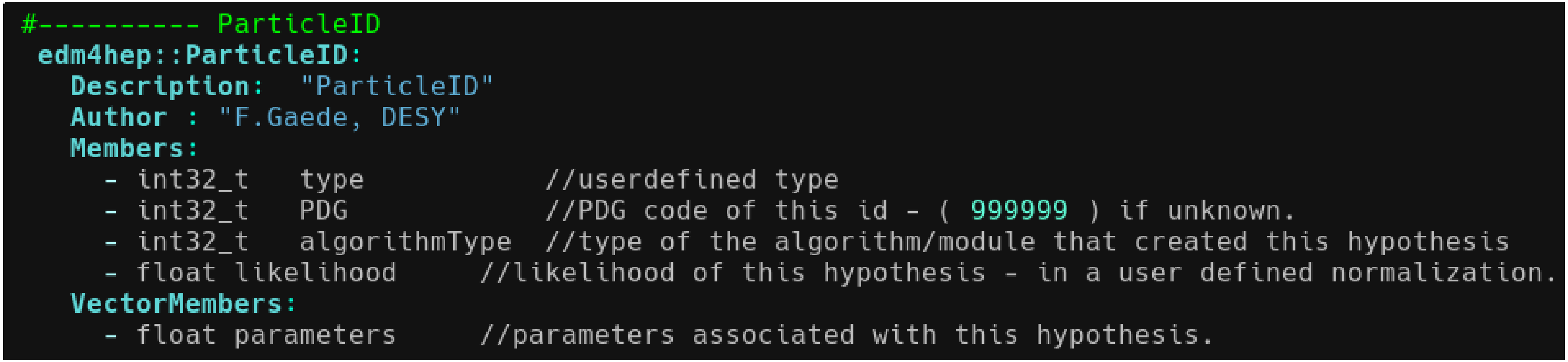
Algorithm: Linking to Reconstructed Particles



Particle Identification Linking
ReconstructedChargedParticles
edm4eic::ReconstructedParticle
ReconstructedChargedParticleAssociations
edm4eic::MCRecoParticleAssociation
PID Algorithm Outputs
Proximity Matching
LinkParticleID
LinkParticleID_factory
association
ReconstructedChargedParticlesWithDRICHPID
edm4eic::ReconstructedParticle
ReconstructedChargedParticleAssociationsWithDRICHPID
edm4eic::MCRecoParticleAssociation
association
MCParticles
MC True Particles

- Linking PID and reconstructed particles is (slightly) non-trivial….
- Track projections in dRICH originate from a non-EDM4hep/eic datatype, therefore cannot link back to it
- Workaround: proximity matching of the projected dRICH TrackSegments to the reconstructed particles (η,ϕ)
- At this stage, we also build the general-level PID objects, for non-experts

Data Model: General PID

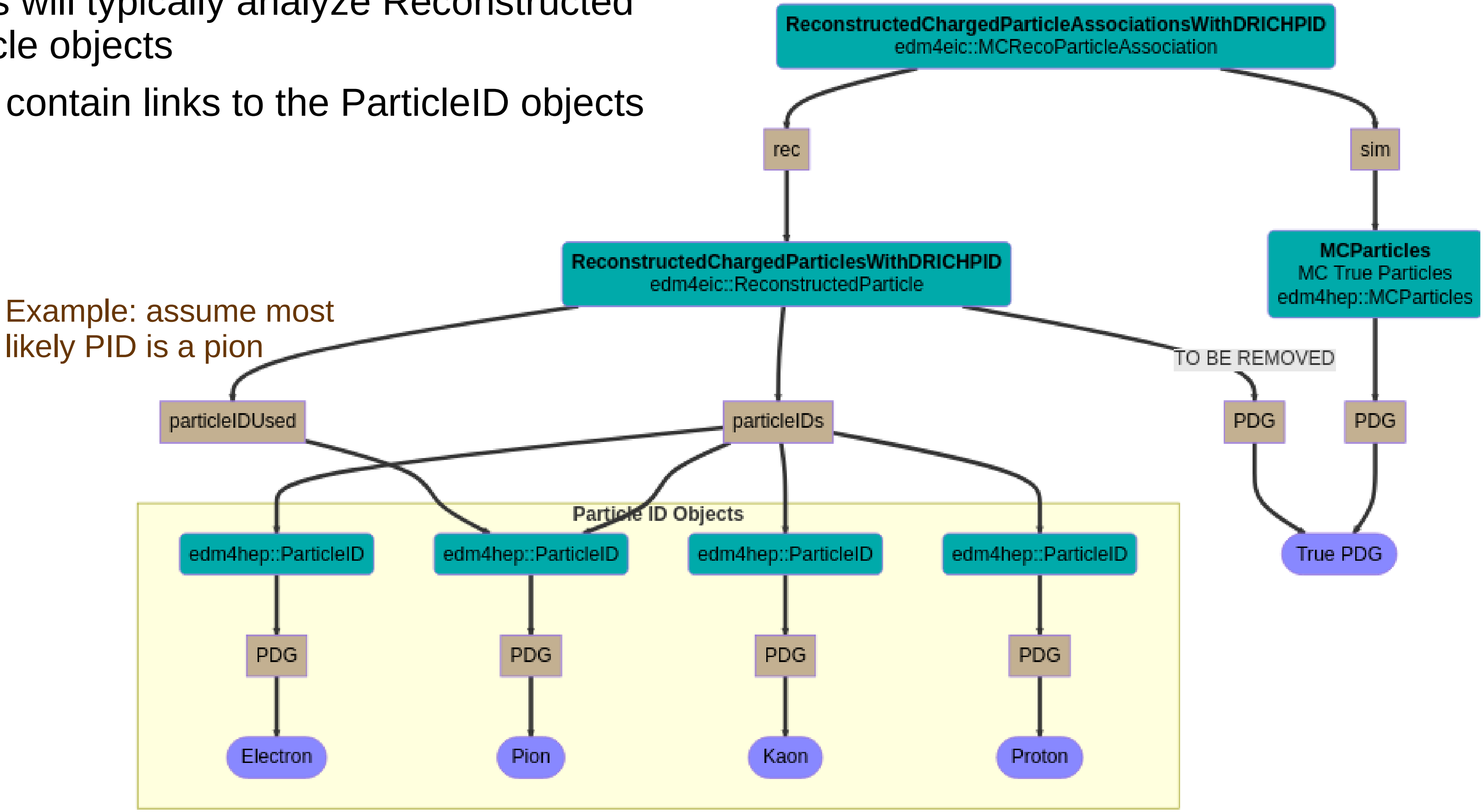


```
#---------- ParticleID
 edm4hep::ParticleID:
   Description:  "ParticleID"
   Author : "F.Gaede, DESY"
   Members:
     - int32_t   type             //userdefined type
     - int32_t   PDG              //PDG code of this id - ( 999999 ) if unknown.
     - int32_t   algorithmType  //type of the algorithm/module that created this hypothesis
     - float likelihood      //likelihood of this hypothesis - in a user defined normalization.
   VectorMembers:
     - float parameters      //parameters associated with this hypothesis.
```

- “**ParticleID**” is the main datatype for PID
- Used by many experiments, *including* ePIC
- This is the “user-level” PID object

Data Model: General PID

- Users will typically analyze Reconstructed Particle objects
- They contain links to the ParticleID objects



Summary

Practically all of these algorithms and datatypes can be shared with other PID subystems

- Is edm4eic::CherenkovParticleID sufficient for pfRICH and DIRC? Or do you need your own datatype?
- What algorithms can your subsystem use (with or without modifications)?
- What additional algorithms do you need to write?
- Can you draw a similar algorithm+collections flowchart for your subsystem?

N.B.: not all of these algorithms have been merged into EICrecon `main`; the latest updated version is found on the `irt-algo` branch, or `irt-algo-stable` for more stability