

Pandora



LAr TPC Reconstruction





Overview

- This talk covers LAr TPC **pattern recognition**: examine hits in order to identify particles.
- Will describe a novel, **multi-algorithm** approach to solving pattern recognition problems.
- This approach has been realised by using key functionality provided by the **Pandora SDK**.

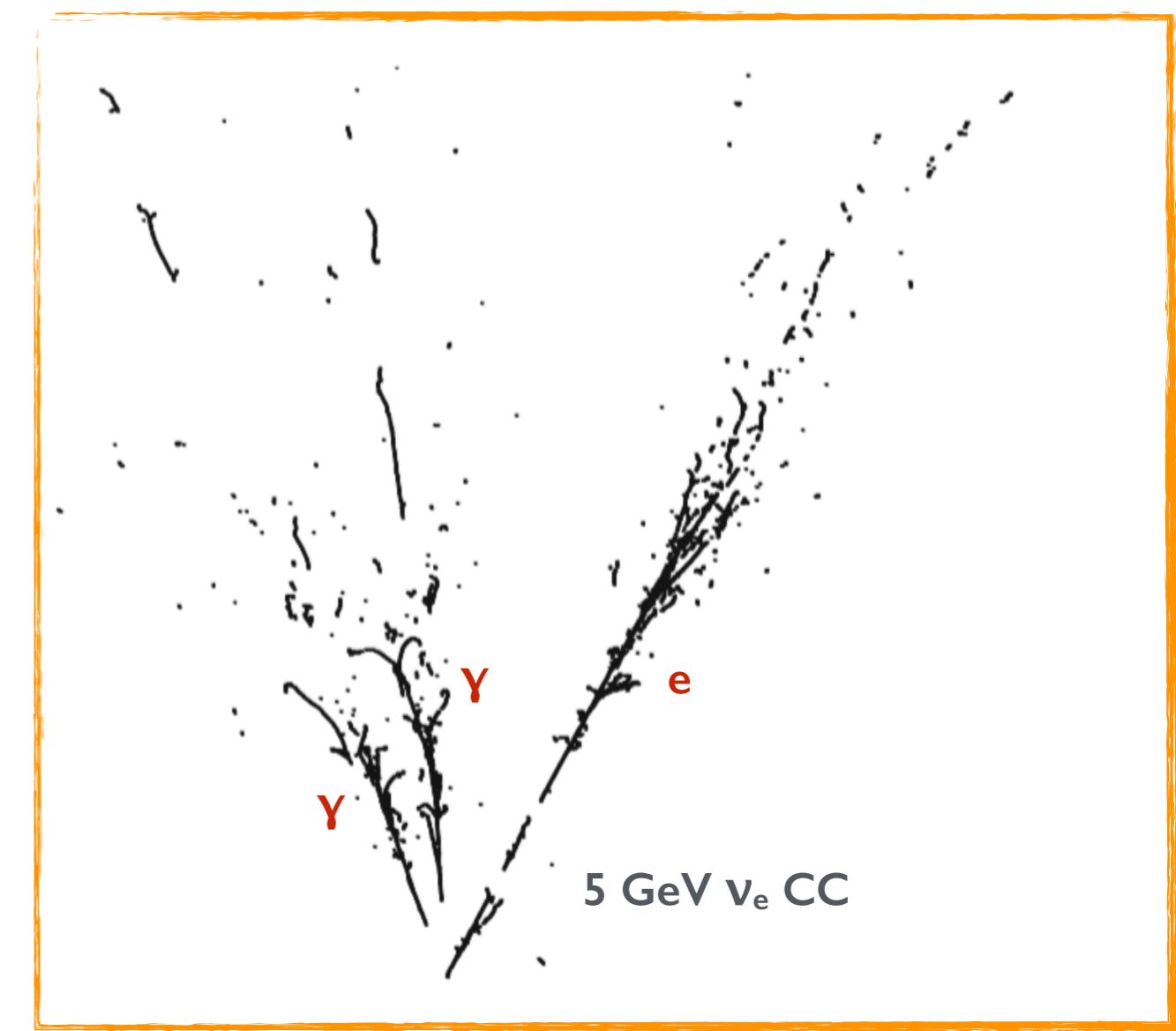
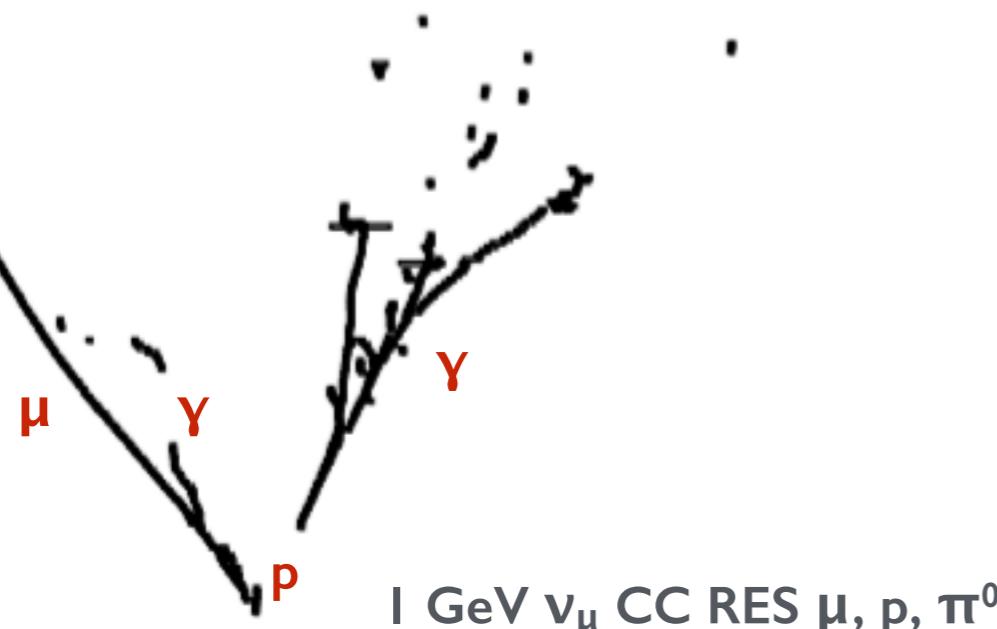
Milestones

1. Provide some brief details about Pandora, its functionality and other use-cases in HEP.
2. Give a high-level overview of the key ideas used by the Pandora LAr TPC algorithms.
3. Start a discussion about how best to characterise/understand reconstruction performance.
4. Explore interfaces with LArSoft: how to provide input hits and extract output particles.



LAr TPC Pattern Recognition

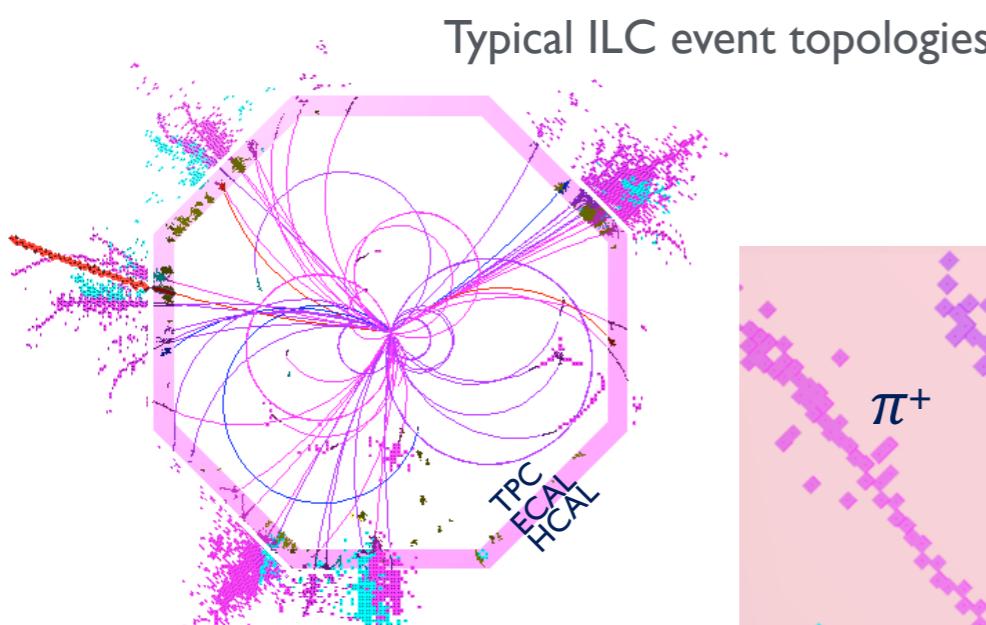
- LAr TPCs allow access to “photograph quality” images of the particles produced in neutrino interactions. The recorded events contain a wealth of information for use in physics analyses.
- The human brain is amazing at pattern recognition and can readily reconstruct most event topologies. **This guides the Pandora approach to automated computer pattern recognition.**



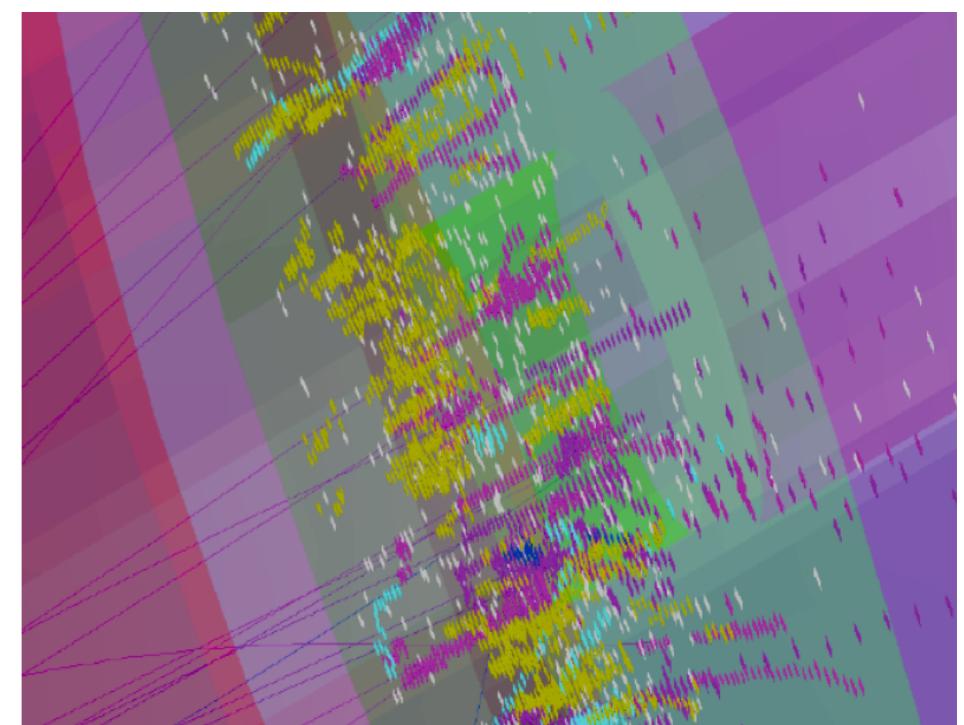


Multi-Algorithm Reconstruction

- This talk will describe an approach to automated reconstruction that uses large numbers (70+) of algorithms, each carefully developed to address specific event topologies, without mistakes.
- Significant change from traditional approach of single algorithms for e.g. track finding, track fitting, shower finding. Can bring together many ideas to provide a truly robust reconstruction.



Showers in CMS HGCAL



- Build upon previous successes with event reconstruction in fine granularity detectors at collider expts: ILC (NIMA.2009.09.009), CLIC (NIMA.2012.10.038) and CMS upgrade (LHCC-P-008).

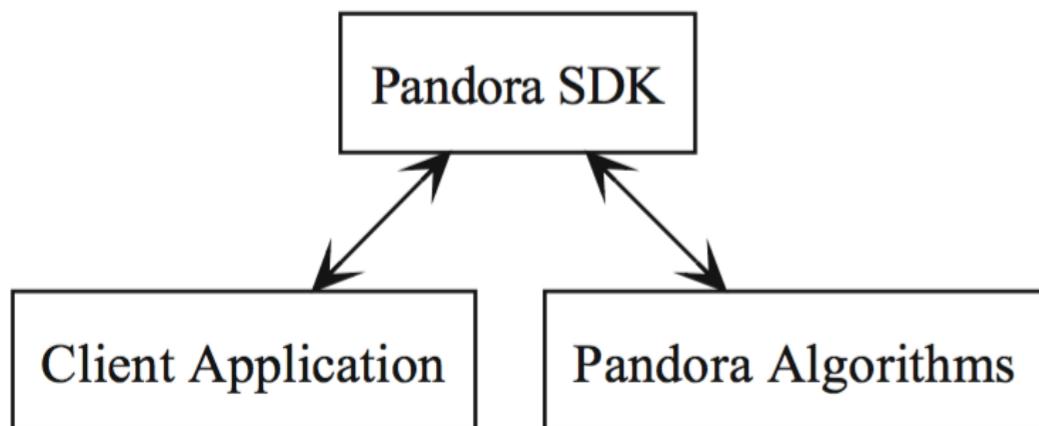


Pandora Introduction

A multi-algorithm event reconstruction can be difficult to implement. The **Pandora Software Development Kit** has been carefully engineered to provide a software environment in which:

1. It is easy for users to provide the building-blocks that define a pattern recognition problem.
2. Logic required to solve pattern recognition problems is cleanly implemented in algorithms.
3. Operations to access or modify building-blocks, or to create new structures, are requested by algorithms and performed by the Pandora framework.

It actively promotes use of large numbers of algorithms, each addressing specific event topologies.



<https://www.github.com/PandoraPFA>

EPJC.75.439

The Pandora Software Development Kit for Pattern Recognition

J. S. Marshall^{a,*}, M. A. Thomson^a

^aCavendish Laboratory, University of Cambridge, Cambridge, United Kingdom

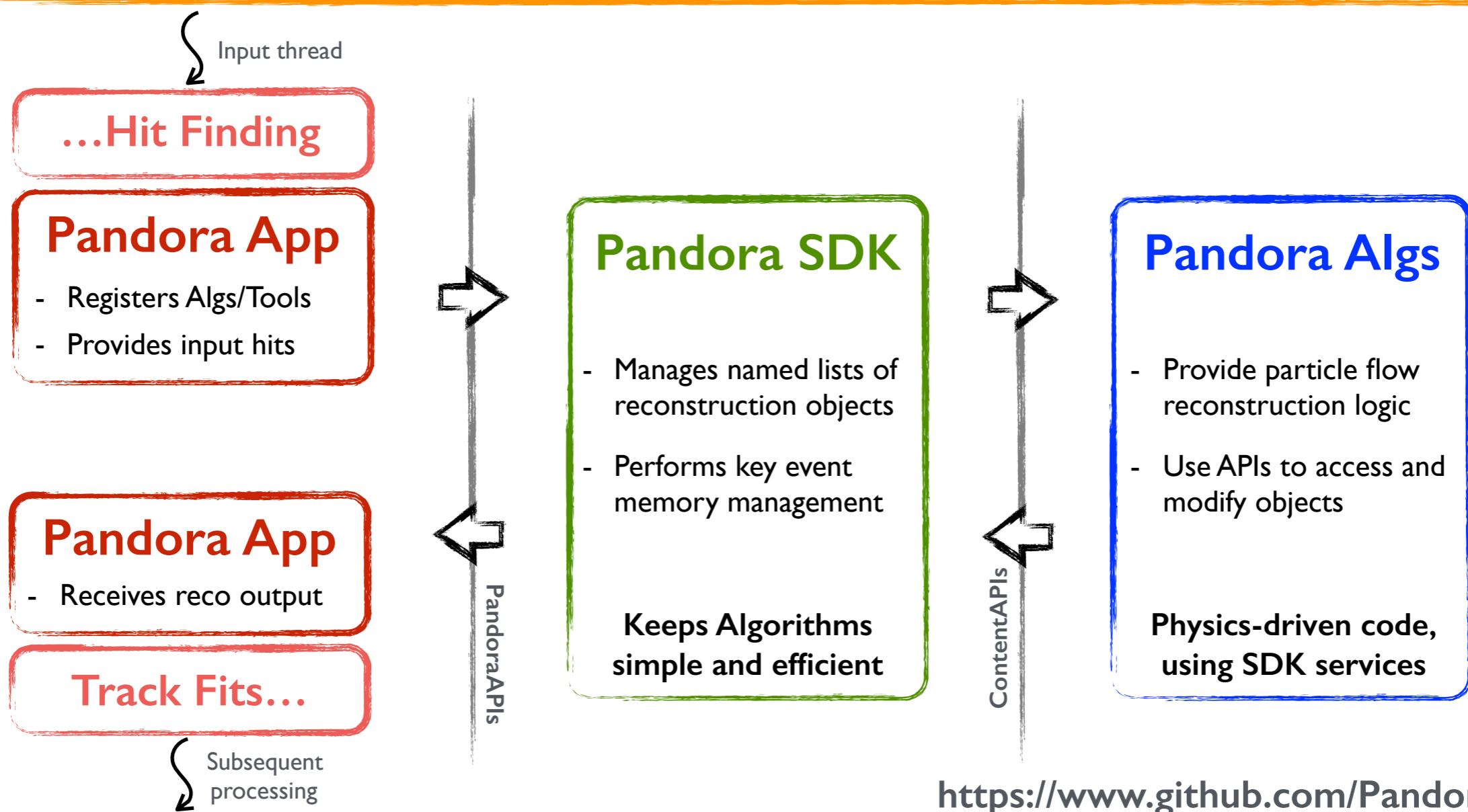
Abstract

The development of automated solutions to pattern recognition problems is important in many areas of scientific research and human endeavour. This paper describes the implementation of the Pandora Software Development Kit, which aids the process of designing, implementing and running pattern recognition algorithms. The Pandora Application Programming Interfaces ensure simple specification of the building-blocks defining a pattern recognition problem. The logic required to solve the problem is implemented in algorithms. The algorithms request operations to create or modify data structures and the operations are performed by the Pandora framework. This design promotes an approach using many decoupled algorithms, each addressing specific topologies. Details of algorithms addressing two pattern recognition problems in High Energy Physics are presented: reconstruction of events at a high-energy e^+e^- linear collider and reconstruction of cosmic ray or neutrino events in a liquid argon time projection chamber.

Keywords: Software Development Kit, Pattern recognition, High Energy Physics

Pandora in LArSoft

- Pandora is definitely **not** a replacement/alternative to LArSoft. Instead, it is an ideal framework for the TPC pattern recognition step. Carefully designed APIs enable multi-algorithm approach.
- Art producer module creates Pandora instance(s), registers algorithms and provides algorithm configuration. Each event, passes details about the Hits to Pandora and receives final output.



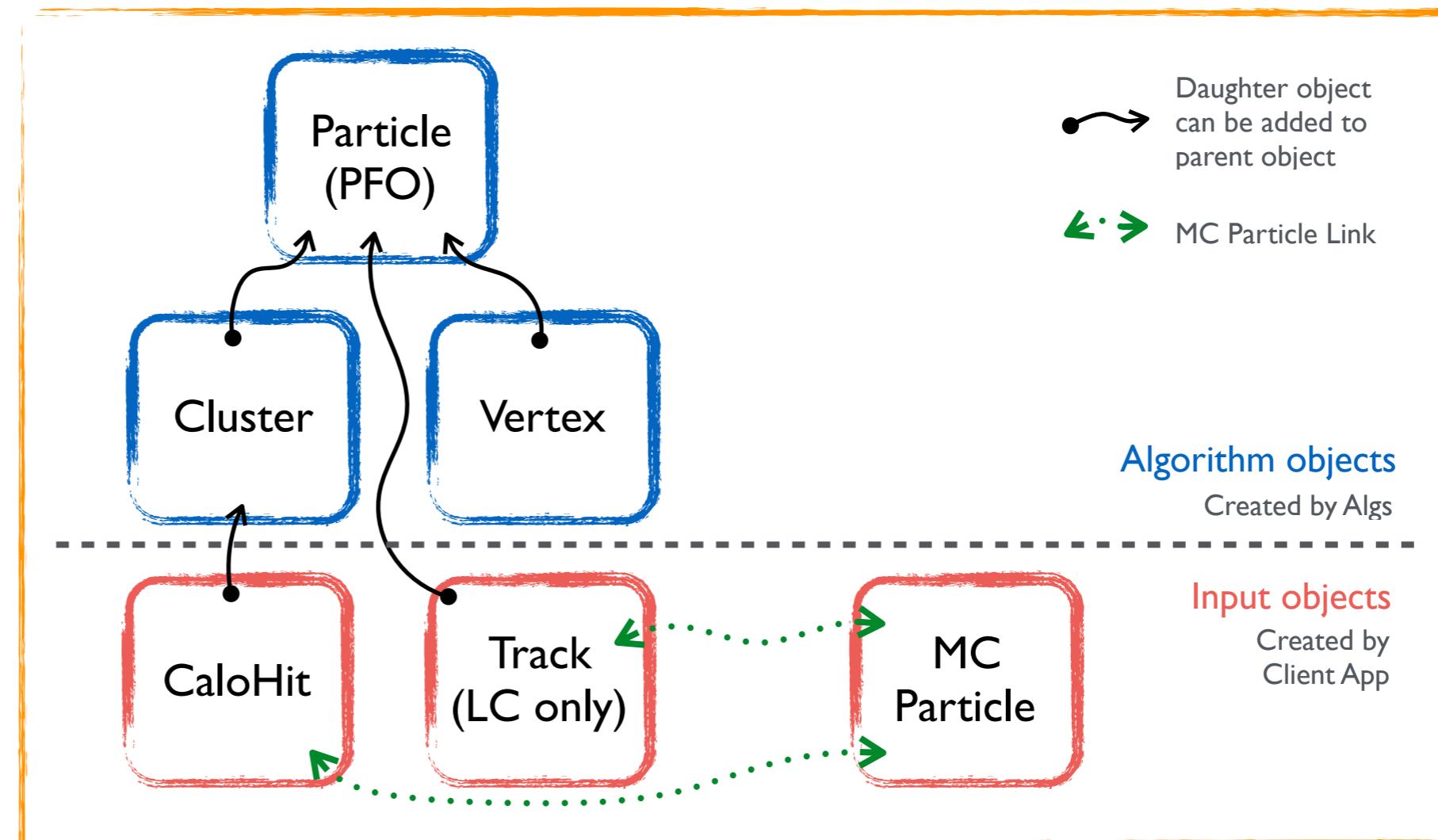


Pandora SDK

- The Pandora SDK provides a comprehensive **Event Data Model (EDM)** for managing pattern recognition problems. Instances of objects in the EDM are owned by **Pandora Managers**.
- The object instances are stored in named lists and the Managers are able to create new objects, delete objects, create and save new lists and move objects between lists.
- The Managers provide a complete set of low-level operations that allow the high-level operations requested by pattern recognition algorithms to be satisfied.

```
pandora::Pandora
{
    - m_pAlgorithmManager
    - m_pCaloHitManager
    - m_pClusterManager
    - m_pGeometryManager
    - m_pMCManager
    - m_pPfoManager
    - m_pPluginManager
    - m_pTrackManager
    - m_pVertexManager
    - m_pPandoraSettings
    - m_pPandoraApiImpl
    - m_pPandoraContentApiImpl
    - m_pPandoraImpl

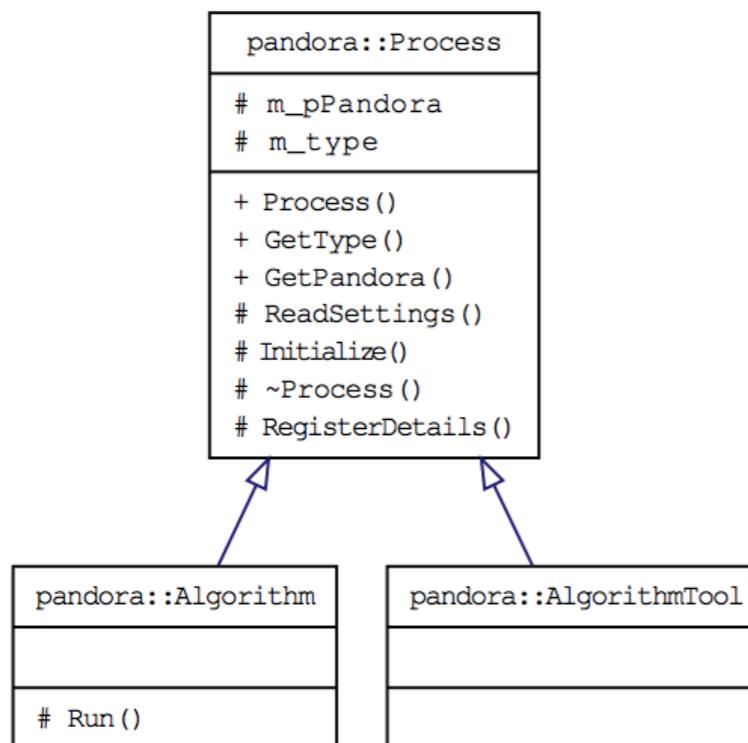
    + Pandora()
    + ~Pandora()
    + GetPandoraApiImpl()
    + GetPandoraContentApiImpl()
    + GetSettings()
    + GetGeometry()
    + GetPlugins()
    - PrepareEvent()
    - ProcessEvent()
    - ResetEvent()
    - ReadSettings()
}
```





Pandora Algorithms

- Pandora algorithms contain the step-by-step instructions for finding patterns in the provided data.
- They use the APIs to access objects and to request the Managers to make new objects or modify existing objects.
- They inherit from the **Process** class, which provides functionality for handshaking with Pandora, XML config and function callbacks.



Algorithm 1 Cluster creation pseudocode. The logic determining when to create new Clusters and when to extend existing Clusters will vary between algorithms.

```
1: procedure CLUSTER CREATION
2:   Create temporary Cluster list
3:   Get current CaloHit list
4:   for all CaloHits do
5:     if CaloHit available then
6:       for all newly-created Clusters do
7:         Find best host Cluster
8:       if Suitable host Cluster found then
9:         Add CaloHit to host Cluster
10:      else
11:        Add CaloHit to a new Cluster
12:   Save new Clusters in a named list
```

Algorithm 2 Cluster merging pseudocode. The logic governing the identification of suitable parent Clusters and daughter Clusters will vary between algorithms.

```
1: procedure CLUSTER MERGING
2:   Get current Cluster list
3:   for all Clusters do
4:     if Cluster is suitable parent then
5:       for all Clusters do
6:         Find best daughter Cluster
7:       if Suitable daughter Cluster found then
8:         Merge daughter Cluster into Parent
```



Pandora Algorithms

- Idea is that algorithms focus on physics, with Pandora performing memory-management, etc. Rapid development is possible in a **standalone environment**, with **visual debugging**.
- During the past three years, have implemented over 150 algorithms, tools, plugins, helper functions, etc. for LAr TPC reconstruction, grouped in the **LArPandoraContent** library.
- Some algorithms are complex and sophisticated. Others are improving, still under active development. The algorithms **gel together** to provide a coherent and robust reconstruction.

- Alg selection and config specified via XML file. Currently available for LAr TPC reco:
 - Dedicated reco for **cosmic ray** muons
 - Dedicated reco for **neutrino** events
 - **Cheated** reco (development use only!)
 - Reco with **3D event slicing**; then apply e.g. neutrino or CR reco to each slice.

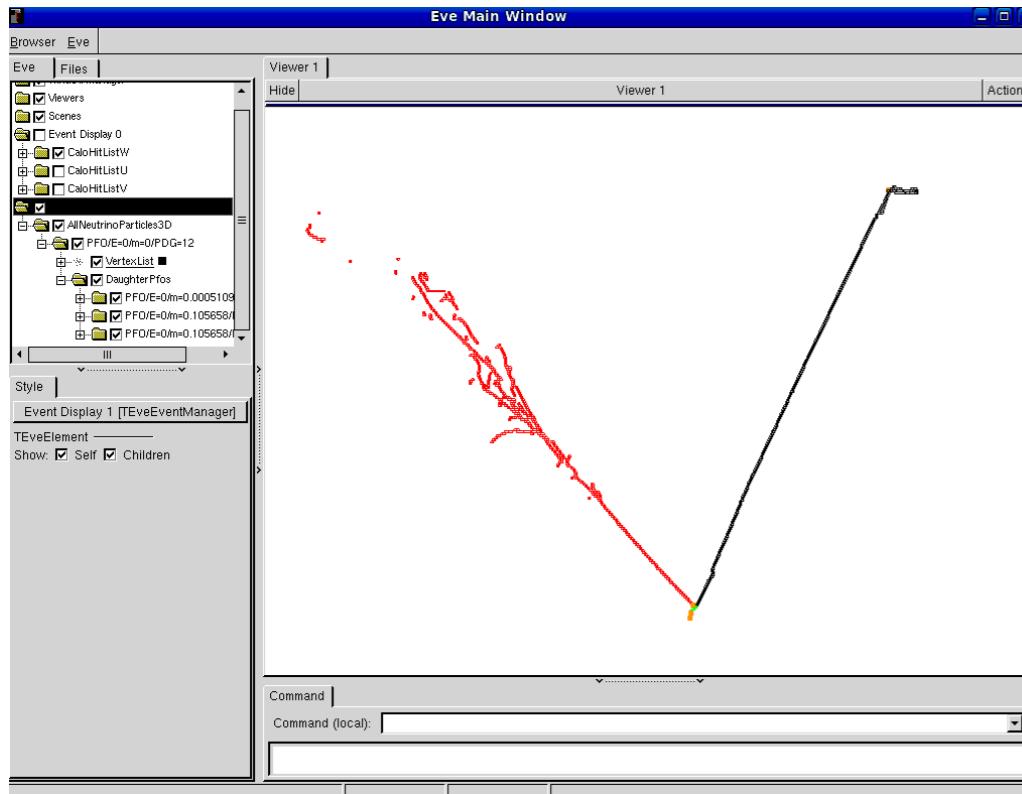
Example XML snippet - 3D track reco



```
<!-- 3D track reconstruction -->
<algorithm type = "LArThreeDTransverseTracks">
  <InputClusterListNameU>ClustersU</InputClusterListNameU>
  <InputClusterListNameV>ClustersV</InputClusterListNameV>
  <InputClusterListNameW>ClustersW</InputClusterListNameW>
  <OutputPfoListName>TrackParticles3D</OutputPfoListName>
  <TrackTools>
    <tool type = "LArClearTracks"/>
    <tool type = "LArLongTracks"/>
    <tool type = "LArOvershootTracks">
      <SplitMode>true</SplitMode>
    </tool>
    <tool type = "LArUndershootTracks">
      <SplitMode>true</SplitMode>
    </tool>
    <tool type = "LArOvershootTracks">
      <SplitMode>false</SplitMode>
    </tool>
    <tool type = "LArUndershootTracks">
      <SplitMode>false</SplitMode>
    </tool>
    <tool type = "LArMissingTrackSegment"/>
    <tool type = "LArTrackSplitting"/>
    <tool type = "LArLongTracks">
      <MinMatchedFraction>0.75</MinMatchedFraction>
      <MinXOverlapFraction>0.75</MinXOverlapFraction>
    </tool>
    <tool type = "LArMissingTrack"/>
  </TrackTools>
</algorithm>
```



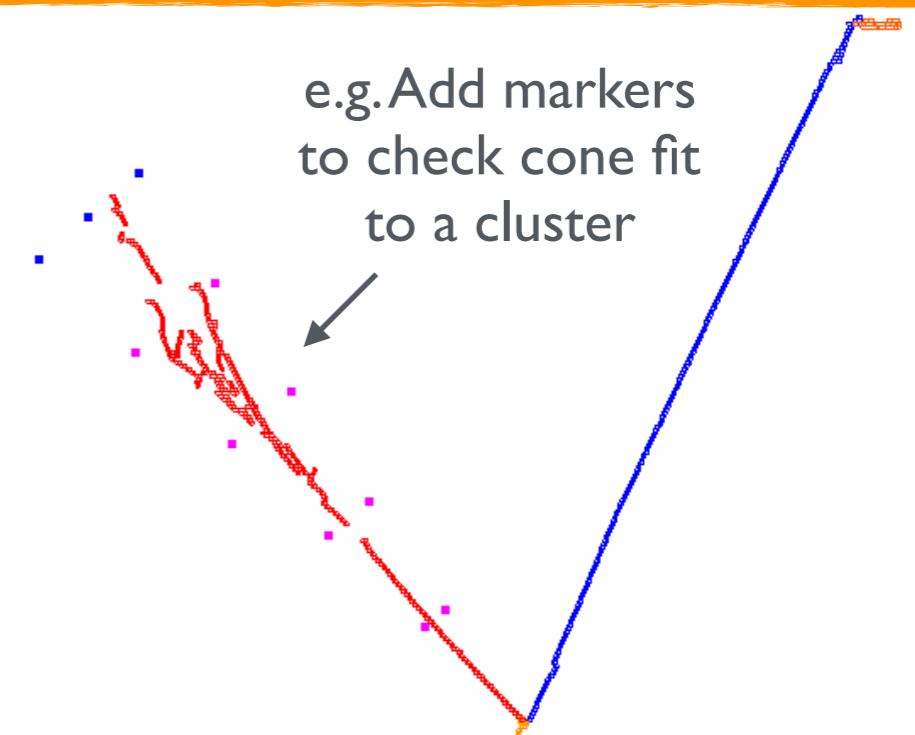
Pandora Visualisation



```
...
<algorithm type = "LArLayerSplitting"/>
<algorithm type = "LArLongitudinalAssociation"/>
<algorithm type = "LArVisualMonitoring">
    <ClusterListNames>ClustersU</ClusterListNames>
</algorithm>
<algorithm type = "LArTransverseAssociation"/>
<algorithm type = "LArVisualMonitoring">
    <ClusterListNames>ClustersU</ClusterListNames>
</algorithm>
<algorithm type = "LArLongitudinalExtension"/>
<algorithm type = "LArTransverseExtension"/>
<algorithm type = "LArOvershootSplitting"/>
<algorithm type = "LArBranchSplitting"/>
<algorithm type = "LArKinkSplitting"/>
...
```

e.g. Add two event display algs to examine changes as reconstruction progresses

- Pandora algorithms can choose to use the Pandora Monitoring library, which has a ROOT dependency.
- ROOT Event Visualisation Environment provides 2D or 3D displays of hits, clusters, particles or vertices.
- Reusable Pandora event display alg can be added to XML to view status of reconstruction at any point.
- Alternatively, algs can use visualisation APIs to provide custom visual debugging - rewarding way to work.





Pandora Key Points

- Pandora is not trying to be LArSoft. It is a framework for pattern recognition algorithms.
- It is generic and is used successfully across multiple (very different) projects.
- Its powerful functionality enables complex algorithms using e.g. recursion or reclustering.
- Design philosophy: support multi-algorithm approach, gradually build-up picture of event.

Will now discuss how to run Pandora in LArSoft...



Pandora Client App

- To use the Pandora SDK, must create a Pandora client application. This provides the input building-blocks to describe the pattern recognition problem and receives the final output.
- The client application is responsible for controlling the pattern recognition reconstruction. It creates the Pandora instance(s) and uses Pandora APIs to send requests to these instances.

- For LArSoft, the Pandora client application is **LArPandoraInterface**
- All Algs and Tools are built as part of the **LArPandoraContent** library
- Input **Hits** (and MCParticles) provided by named art producers.
- Output comprises: **PFParticles**, Clusters, Tracks, Showers, Vertices...

Algorithm 3 Pseudocode description of a client application for LAr TPC event reconstruction in a single drift volume

```
1: procedure MAIN
2:   Create a Pandora instance
3:   Register Algorithms and Plugins
4:   Ask Pandora to parse XML settings file
5:   for all Events do
6:     Create CaloHit instances
7:     Create MCParticle instances
8:     Specify MCParticle-Calohit relationships
9:     Ask Pandora to process the event
10:    Get output PFOs and write to file
11:    Reset Pandora before next event
```



Pandora Input Hits

I. Loop over art hits; collect required information for self-describing Pandora hits

```
void LArPandoraBase::CreatePandoraHits2D(const HitVector &hitVector, HitMap &hitMap) const
{
    mf::LogDebug("LArPandora") << " *** LArPandora::CreatePandoraHits2D(...) *** " << std::endl;

    // Set up ART services
    art::ServiceHandle<geo::Geometry> theGeometry;
    art::ServiceHandle<util::DetectorProperties> theDetector;

    // Loop over ART hits
    int hitCounter(0);

    for (HitVector::const_iterator iter = hitVector.begin(), iterEnd = hitVector.end(); iter != iterEnd; ++iter)
    {
        const art::Ptr<recob::Hit> hit = *iter;

        const geo::WireID hit_WireID(hit->WireID());
        const geo::View_t hit_View(hit->View());

        const double hit_Time(hit->PeakTime());
        const double hit_Charge(hit->Integral());
        const double hit_TimeStart(hit->PeakTimeMinusRMS());
        const double hit_TimeEnd(hit->PeakTimePlusRMS());

        double xyz[3];
        theGeometry->Cryostat(hit_WireID.Cryostat).TPC(hit_WireID.TPC).Plane(hit_WireID.Plane).Wire(hit_WireID.Wire).GetCenter(xyz);
        const double y0_cm(xyz[1]);
        const double z0_cm(xyz[2]);

        const double wire_pitch_cm(theGeometry->WirePitch(hit_View)); // cm

        const double xpos_cm(theDetector->ConvertTicksToX(hit_Time, hit_WireID.Plane, hit_WireID.TPC, hit_WireID.Cryostat));
        const double dxpos_cm(std::abs(theDetector->ConvertTicksToX(hit_TimeEnd, hit_WireID.Plane, hit_WireID.TPC, hit_WireID.Cryostat) -
            theDetector->ConvertTicksToX(hit_TimeStart, hit_WireID.Plane, hit_WireID.TPC, hit_WireID.Cryostat)));

        const double mips(this->GetMips(hit_Charge, hit_View));

        // Continued on next slide
        .
        .
    }
}
```

Snippet from [LArPandoraInterface](#)



Pandora Input Hits

```
PandoraApi::CaloHit::Parameters caloHitParameters;
caloHitParameters.m_expectedDirection = pandora::CartesianVector(0., 0., 1.);
caloHitParameters.m_cellNormalVector = pandora::CartesianVector(0., 0., 1.);
caloHitParameters.m_cellSize0 = m_dx_cm;
caloHitParameters.m_cellSize1 = (m_useHitWidths ? dxpos_cm : m_dx_cm);
caloHitParameters.m_cellThickness = wire_pitch_cm;
caloHitParameters.m_cellGeometry = pandora::RECTANGULAR;
caloHitParameters.m_time = 0.;
caloHitParameters.m_nCellRadiationLengths = m_dx_cm / m_rad_cm;
caloHitParameters.m_nCellInteractionLengths = m_dx_cm / m_int_cm;
caloHitParameters.m_isDigital = false;
caloHitParameters.m_hitRegion = pandora::SINGLE_REGION;
caloHitParameters.m_layer = 0;
caloHitParameters.m_isInOuterSamplingLayer = false;
caloHitParameters.m_inputEnergy = hit_Charge;
caloHitParameters.m_mipEquivalentEnergy = mips;
caloHitParameters.m_electromagneticEnergy = mips * m_mips_to_gev;
caloHitParameters.m_hadronicEnergy = mips * m_mips_to_gev;
caloHitParameters.m_pParentAddress = (void*)((intptr_t)(++hitCounter));

if (hit_View == geo::kW)
{
    caloHitParameters.m_hitType = pandora::TPC_VIEW_W;
    const double wpos_cm(z0_cm);
    caloHitParameters.m_positionVector = pandora::CartesianVector(xpos_cm, 0., wpos_cm);
}
else if(hit_View == geo::kU)
{
    caloHitParameters.m_hitType = pandora::TPC_VIEW_U;
    const double upos_cm(lar_content::LArGeometryHelper::GetLArTransformationPlugin(*pPandora)->YZtoU(y0_cm, z0_cm));
    caloHitParameters.m_positionVector = pandora::CartesianVector(xpos_cm, 0., upos_cm);
}
else if(hit_View == geo::kV)
{
    caloHitParameters.m_hitType = pandora::TPC_VIEW_V;
    const double vpos_cm(lar_content::LArGeometryHelper::GetLArTransformationPlugin(*pPandora)->YZtoV(y0_cm, z0_cm));
    caloHitParameters.m_positionVector = pandora::CartesianVector(xpos_cm, 0., vpos_cm);
}
else
{
    mf::LogError("LArPandora") << " --- WARNING: UNKNOWN VIEW !!! (View=" << hit_View << ")" << std::endl;
    throw pandora::StatusCodeException(pandora::STATUS_CODE_FAILURE);
}

// Create the Pandora hit
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::CaloHit::Create(*pPandora, caloHitParameters));
```

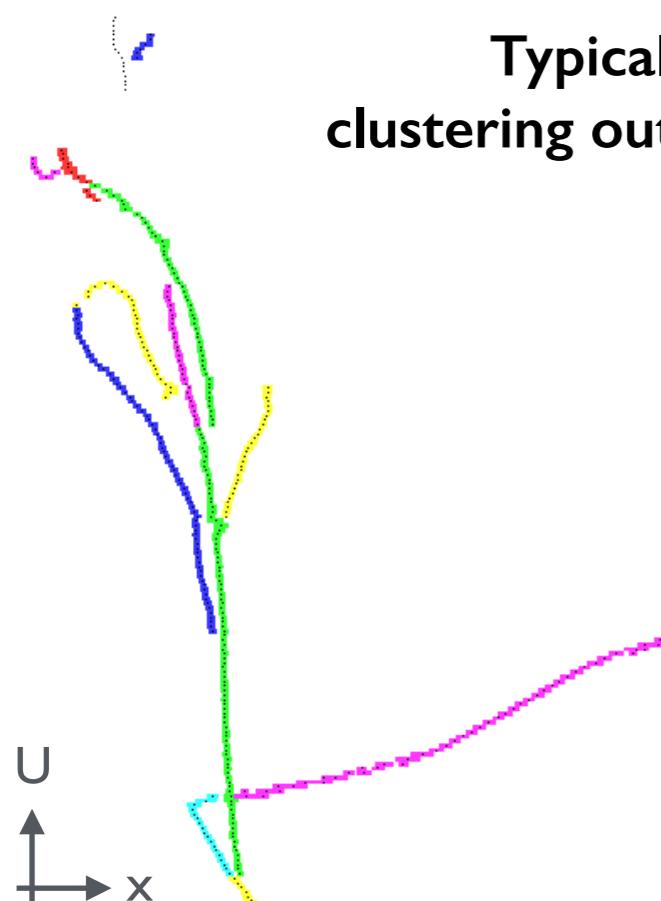
2. Assign information to members in a
PandoraApi::CaloHit::Parameters
instance, then call
PandoraApi::CaloHit::Create

I. Track Clustering in 2D

Overview of Pandora LAr TPC reconstruction algorithms:

- Start with cautious, track-oriented 2D clustering. Important to avoid mistakes at this early stage.
- Use series of topological-association algorithms to carefully merge/split the 2D proto-clusters.

Typical 2D clustering output



2D Reco Snippet from PandoraSettings XML file:

```
<algorithm type = "LArClusteringParent">
    <algorithm type = "LArTrackClusterCreation" description = "ClusterFormation"/>
    <InputCaloHitListName>CaloHitListW</InputCaloHitListName>
    <ClusterListName>ClustersW</ClusterListName>
    <ReplaceCurrentCaloHitList>false</ReplaceCurrentCaloHitList>
    <ReplaceCurrentClusterList>true</ReplaceCurrentClusterList>
</algorithm>

<algorithm type = "LArLayerSplitting"/>
<algorithm type = "LArLongitudinalAssociation"/>
<algorithm type = "LArTransverseAssociation"/>
<algorithm type = "LArLongitudinalExtension"/>
<algorithm type = "LArTransverseExtension"/>
<algorithm type = "LArOvershootSplitting"/>
<algorithm type = "LArBranchSplitting"/>
<algorithm type = "LArKinkSplitting"/>
```

2D Cluster Creation

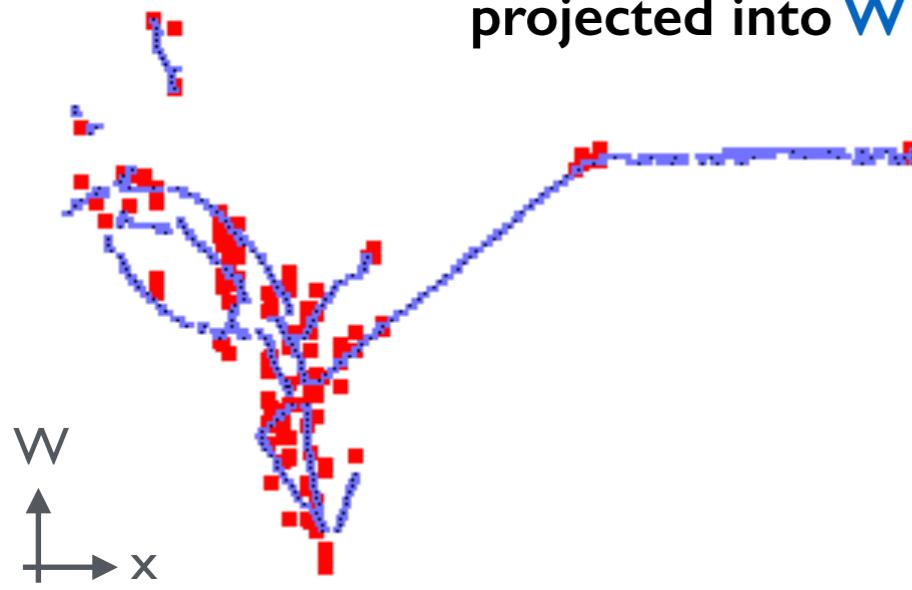
2D Cluster Merging/Splitting



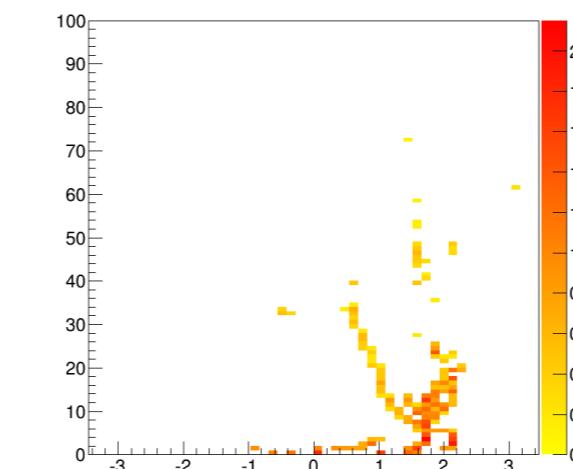
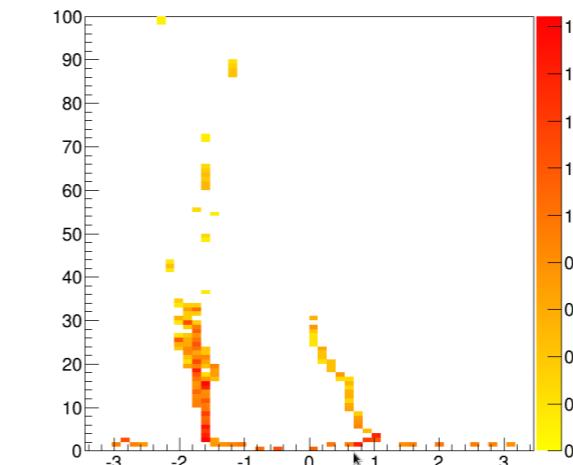
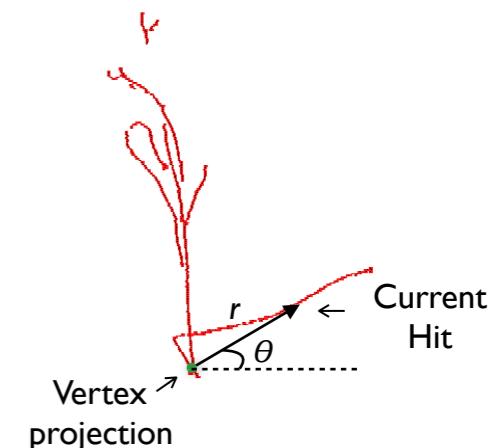
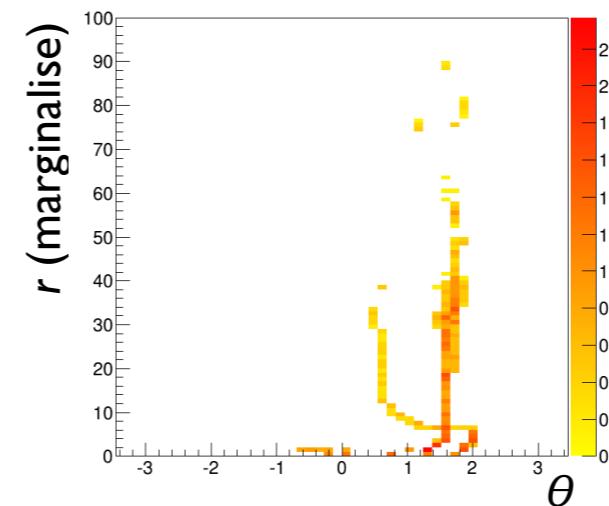
2. Vertex Reconstruction in 3D

- Use pairs of 2D clusters to produce list of possible 3D vertex positions.
- Examine candidate vertices, calculate a score for each and select the best.
- Can then e.g. split 2D clusters at projected vertex position, if required.
- Use vertex position to protect primary particles when growing showers.

3D candidate vertices projected into W view



Calculate a score for each vertex candidate

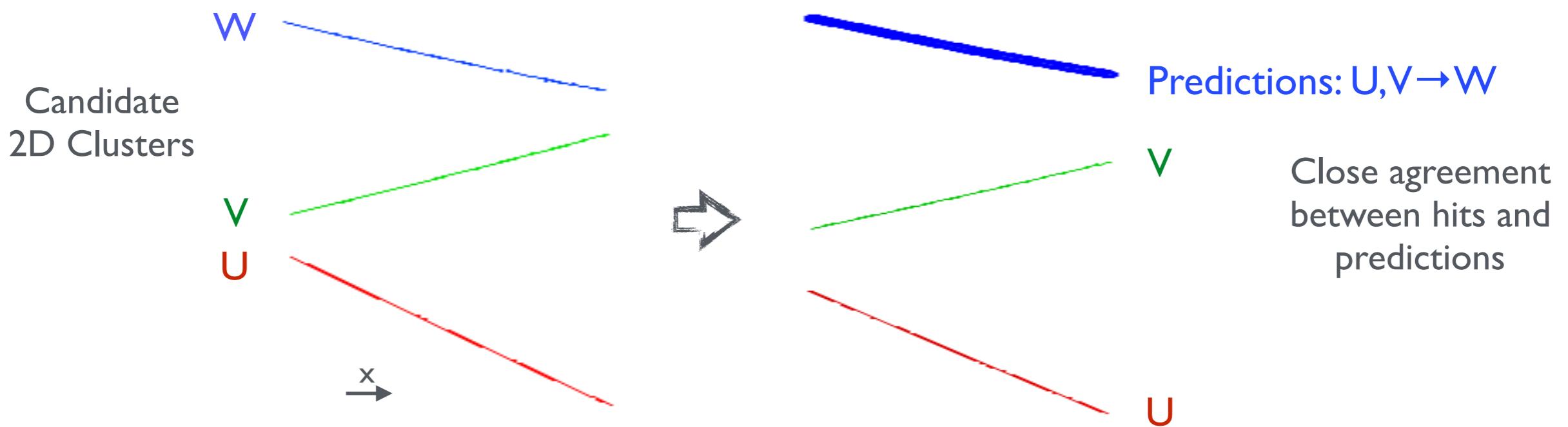


V

W

3. Track Reconstruction in 3D

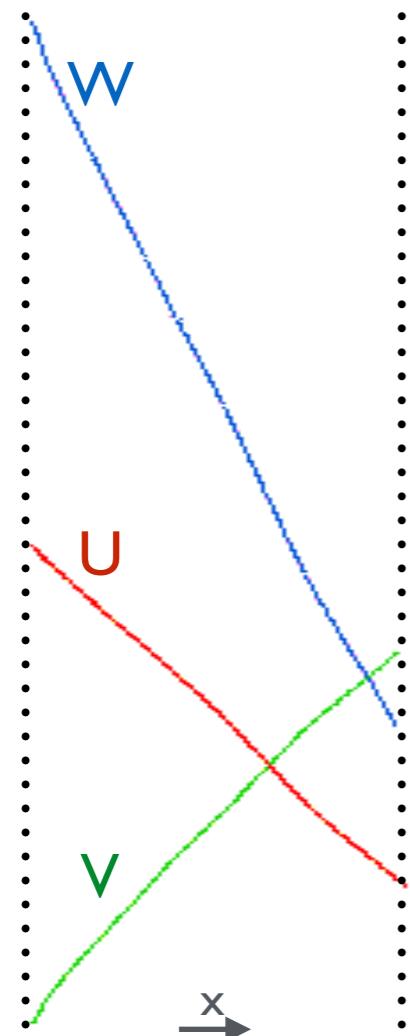
- The Pandora 3D track matching algorithms exploit x-overlap (time-overlap) between all possible groupings of candidate U, V and W clusters:
 - For a given x-coordinate, obtain **sliding linear fit** positions for a pair of clusters (e.g. U,V)
 - Use these values to predict the position of the third cluster (e.g. W)
 - Compare true sliding fit position with prediction, calculating a χ^2 value
 - Account for all possible predictions: U,V→W; VW→U; UW→V



All results are stored in a **tensor**, recording x-overlap span, number of sampling points, number of “matched” sampling points and χ^2 . Tensor neatly documents all matching ambiguities.

3. Track Reconstruction in 3D

Create a tensor, storing overlap details for trios of 2D clusters. Tools make 2D cluster changes in order to **diagonalise tensor**. If a tool makes a change (e.g. splits a cluster), all tools run again.



1:1:1

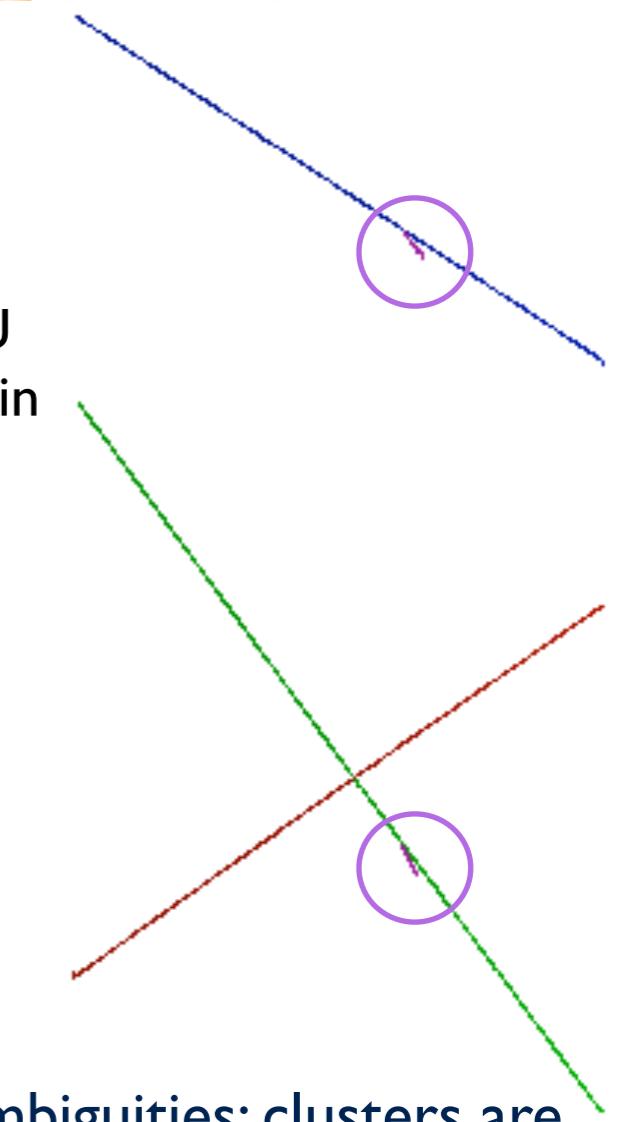
Aim: group together 3 x 2D clusters in a new track “particle”

Find unambiguous elements in the tensor, demanding that the common x-overlap is 90% of the x-span for all three clusters.

Clear Tracks Tool

e.g. 1:2:2

Ringed clusters in **V** and **W** views also match **U** cluster, so **U** cluster is ambiguous in tensor.



Resolve **obvious** ambiguities: clusters are matched in multiple configurations, but one tensor element is much better than others.

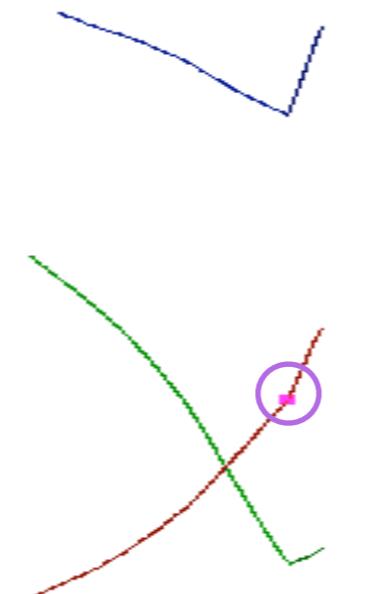
Long Tracks Tool



3. Track Reconstruction in 3D

1:2:2

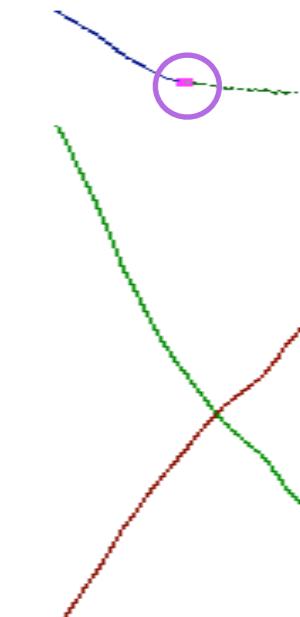
Two clusters in **W** and **V** views, matched to common cluster in **U**.
Split U cluster.



Overshoot Tracks Tool

1:1:2

Two clusters in **W** view, matched to common clusters in **U** and **V** views.
Merge W clusters.

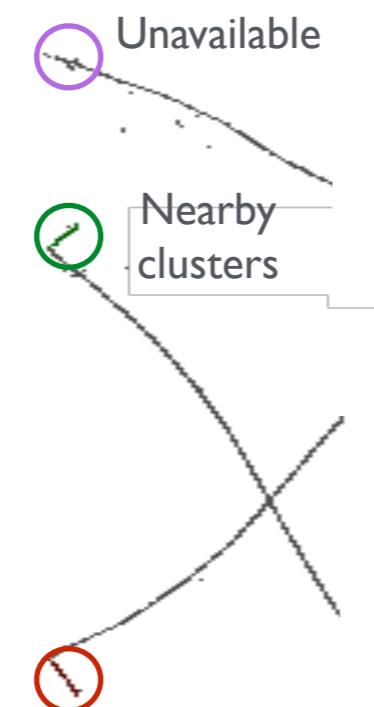


Undershoot Tracks Tool

2:2:1

Clear match between **U,V** and **W** clusters, but **W** cluster unavailable.

Track overlap in **W** view:
create two-cluster particle.



Missing Tracks Tool

1:1:Fragments

Clean track in **U** view
Clean track in **V** view
Many broken cluster fragments in **W** view

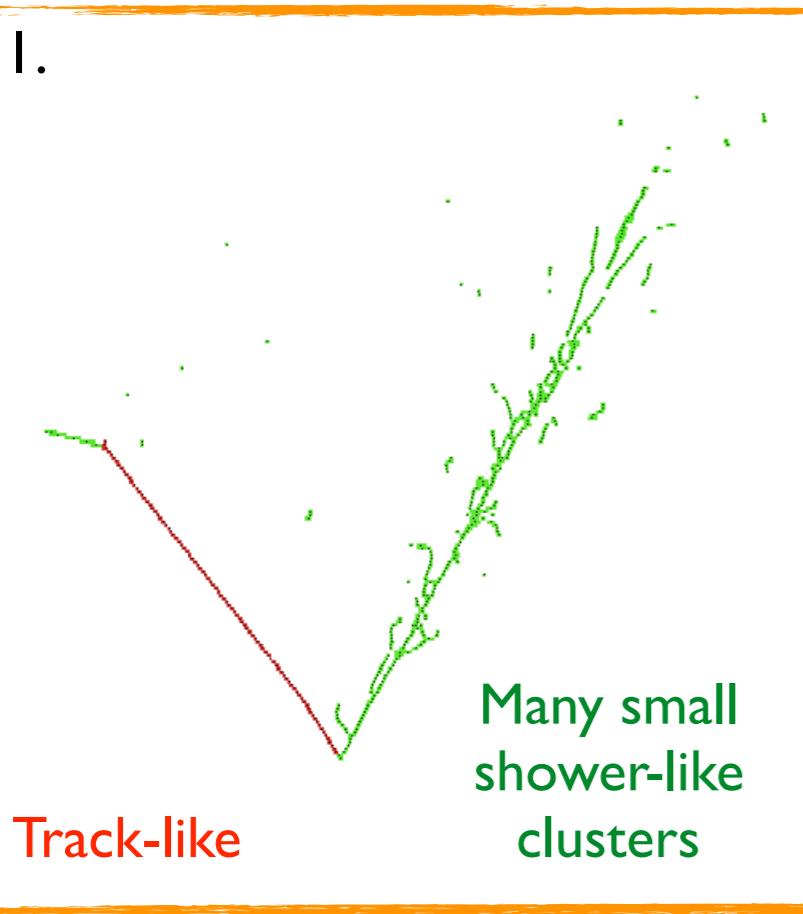
Clear Track Fragments

4. Shower Reconstruction in 2D

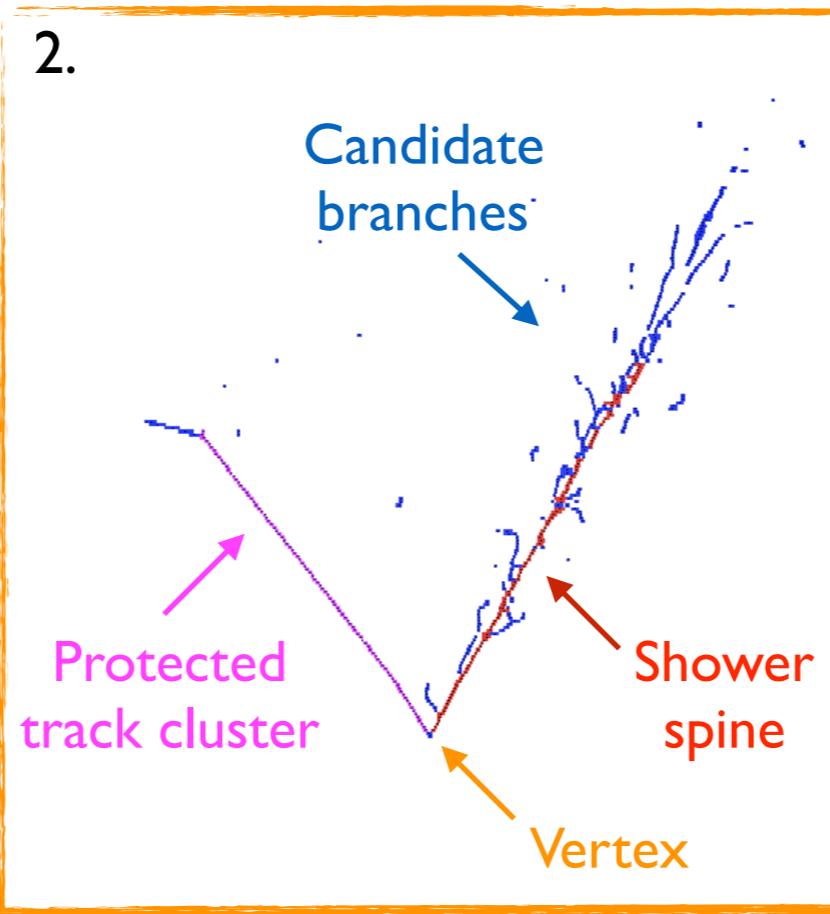
Attempt to add **branches** to long clusters representing shower **spines**. The long clusters may already exist in track particles. Grow 2D showers using three key ideas:

1. Characterise 2D clusters as track-like or shower-like.
2. Identify long (often vertex-associated), shower-like 2D clusters that represent shower spines.
3. Add branches to the list of spines: work recursively, finding branches for top-level spine, then branches on branches, etc. For every branch, record strength of association to each spine.

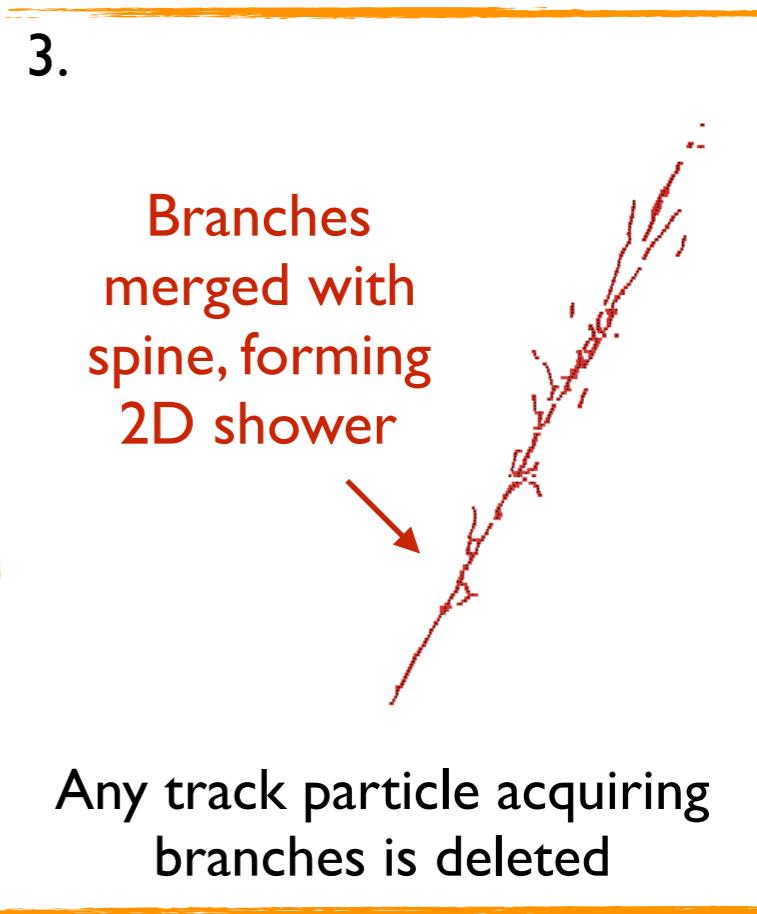
1.



2.



3.



5. Shower Reconstruction in 3D

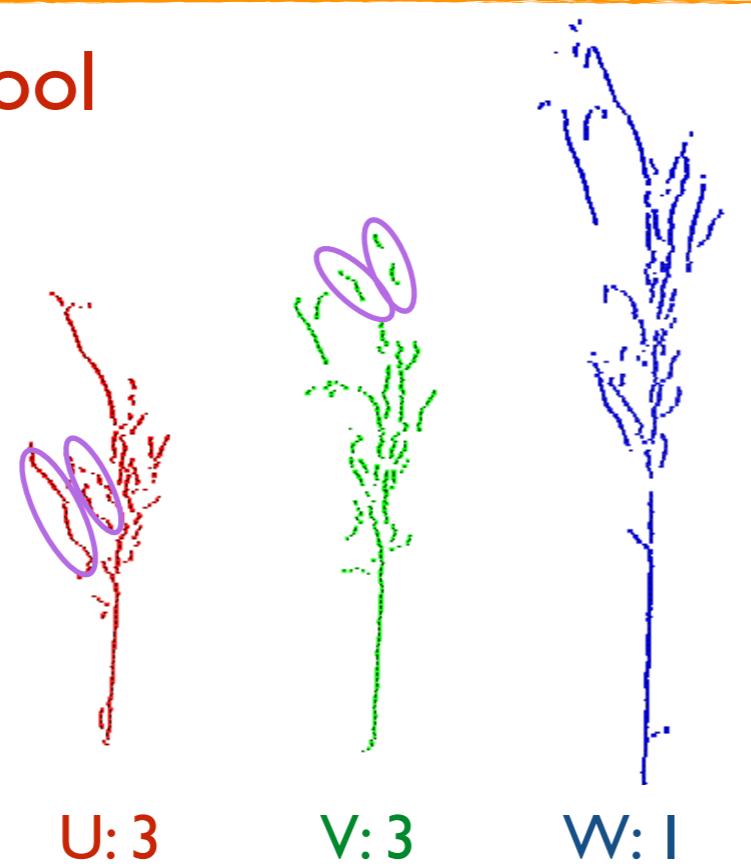
Re-use ideas and base classes from 3D track reconstruction: build and diagonalise a tensor



Clear Showers Tool

e.g. 3:3:1

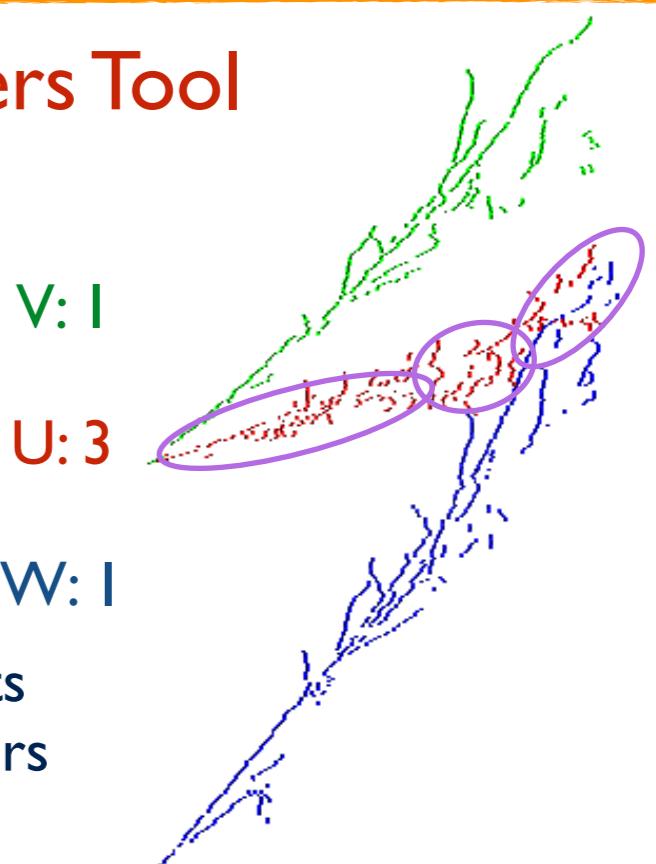
Small problems cause ambiguities, but best combination obvious



Split Showers Tool

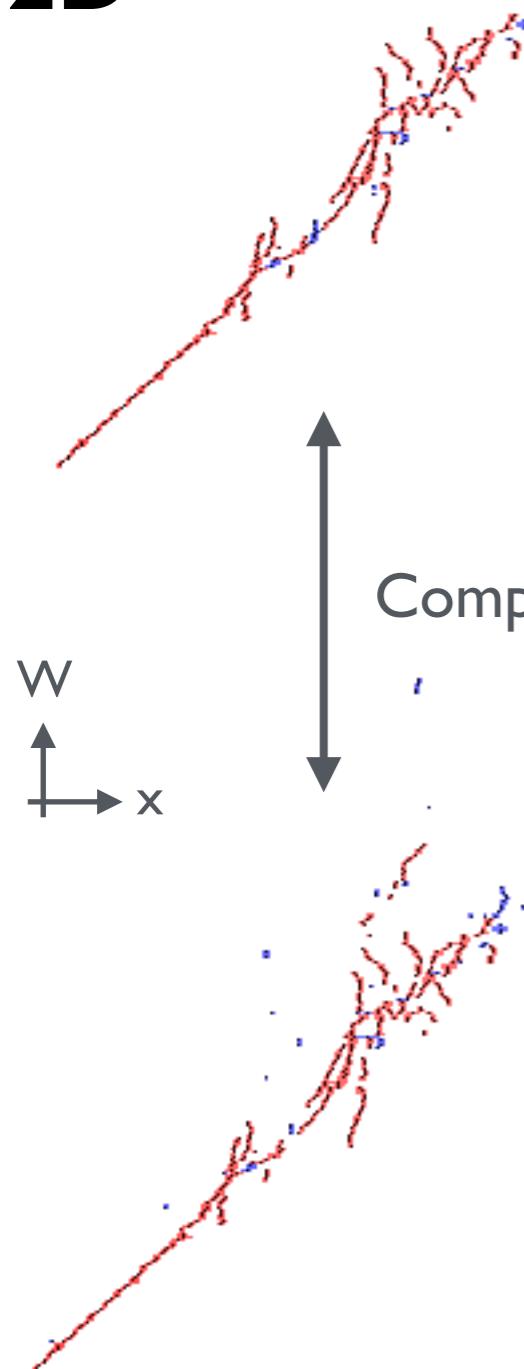
e.g. 3:1:1

Tensor elements have two clusters in common



6. Mop-Up in 2D and 3D

2D



BoundedCluster

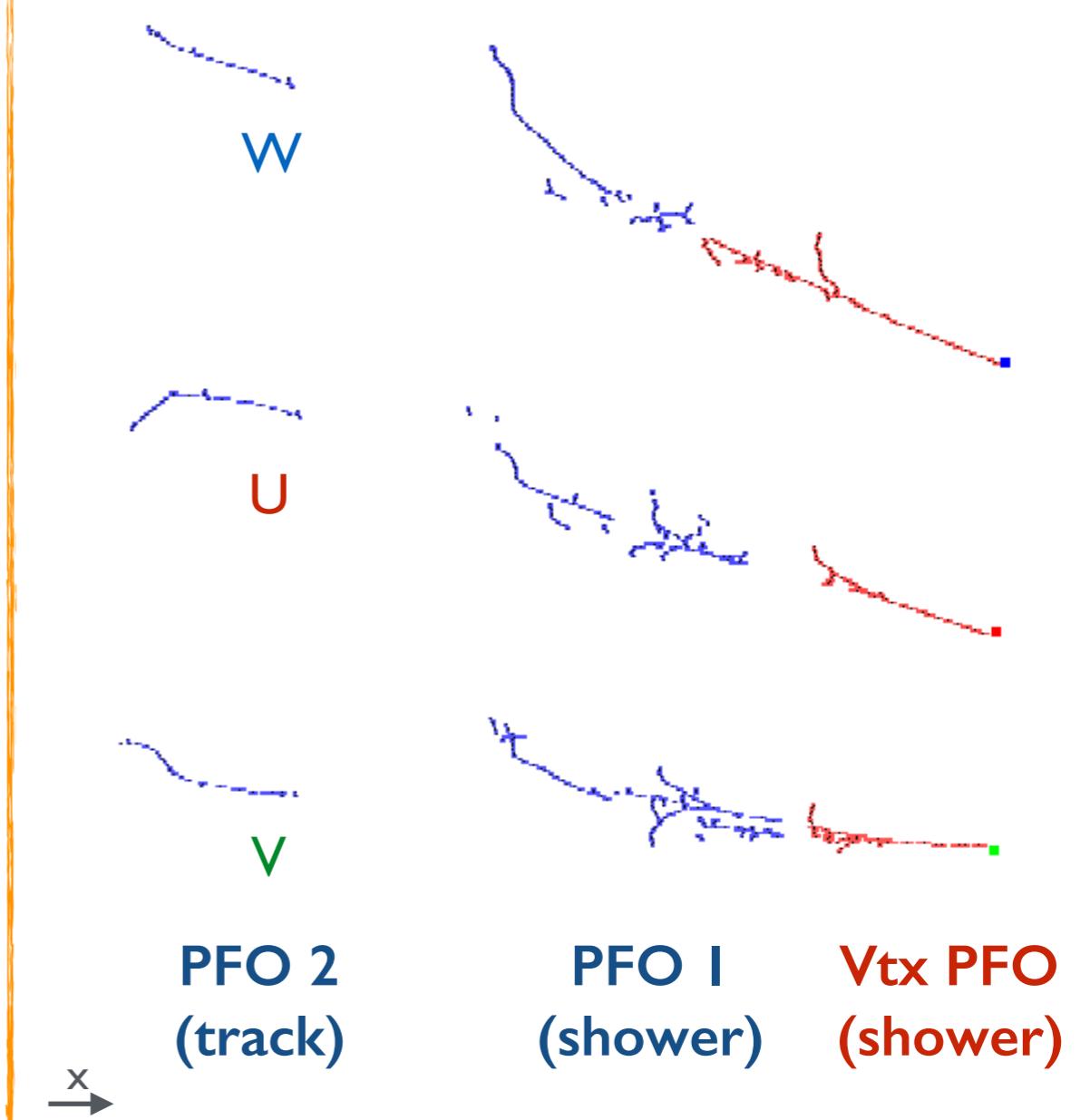
mop-up, picks-up lone clusters enclosed in shower envelopes.

Compare

ConeBased mop-up,
picks-up clusters
enclosed in a cone
about shower.

3D

VertexBasedPFO mop-up,
important for sparse showers.



PFO \leftrightarrow Particle Flow Object



7. Event Building in 3D

The final algorithms and tools create **3D SpacePoints** for each reconstructed particle. The full **particle hierarchy** is also reconstructed, so a typical event output is as shown below:

5 GeV ν_e CC: Display 1/4

The reconstructed neutrino particle contains:

- Metadata: PDG code, 4-momentum, etc
- A 3D interaction vertex
- A list of daughter particles

3D neutrino
interaction vertex





7. Event Building in 3D

The final algorithms and tools create **3D SpacePoints** for each reconstructed particle. The full **particle hierarchy** is also reconstructed, so a typical event output is as shown below:

5 GeV ν_e CC: Display 2/4

- + Primary daughter particles of the neutrino, each of which has:
 - Particle metadata
 - A list of 2D clusters and a 3D cluster
 - A 3D interaction vertex
 - A list of any further daughter particles



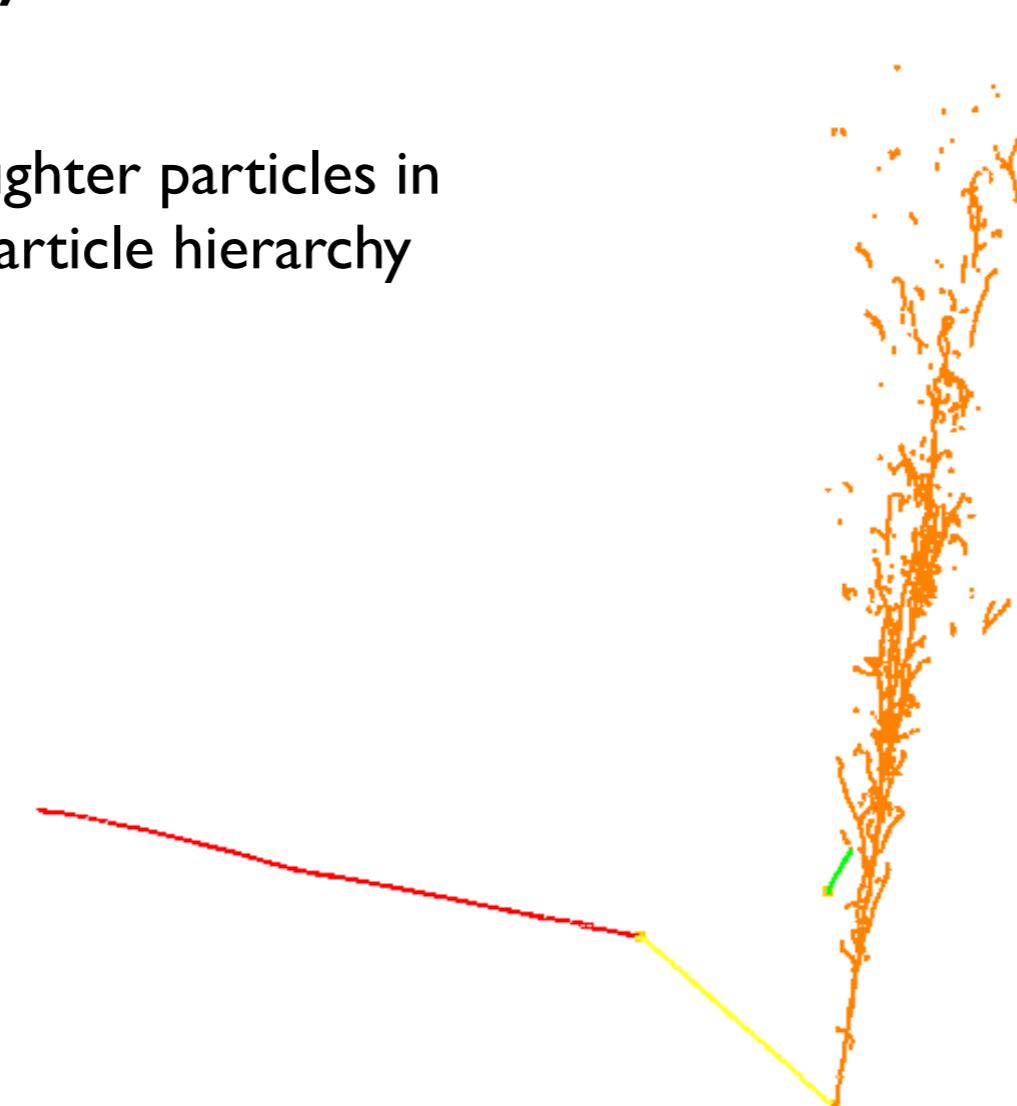


7. Event Building in 3D

The final algorithms and tools create **3D SpacePoints** for each reconstructed particle. The full **particle hierarchy** is also reconstructed, so a typical event output is as shown below:

5 GeV ν_e CC: Display 3/4

+ Complete list of daughter particles in the reconstructed particle hierarchy



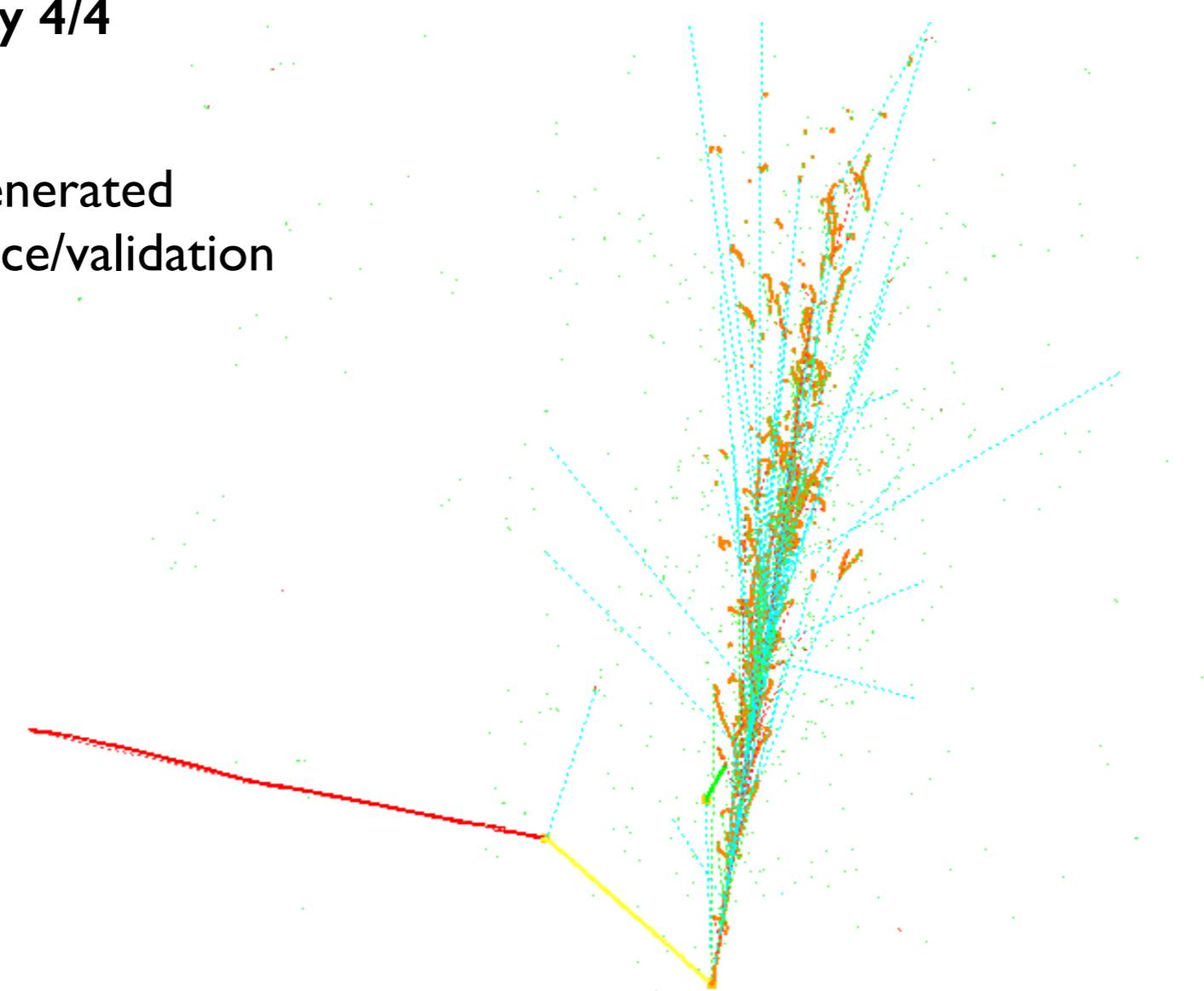


7. Event Building in 3D

The final algorithms and tools create **3D SpacePoints** for each reconstructed particle. The full **particle hierarchy** is also reconstructed, so a typical event output is as shown below:

5 GeV ν_e CC: Display 4/4

+ Overlay details of generated particles, for reference/validation

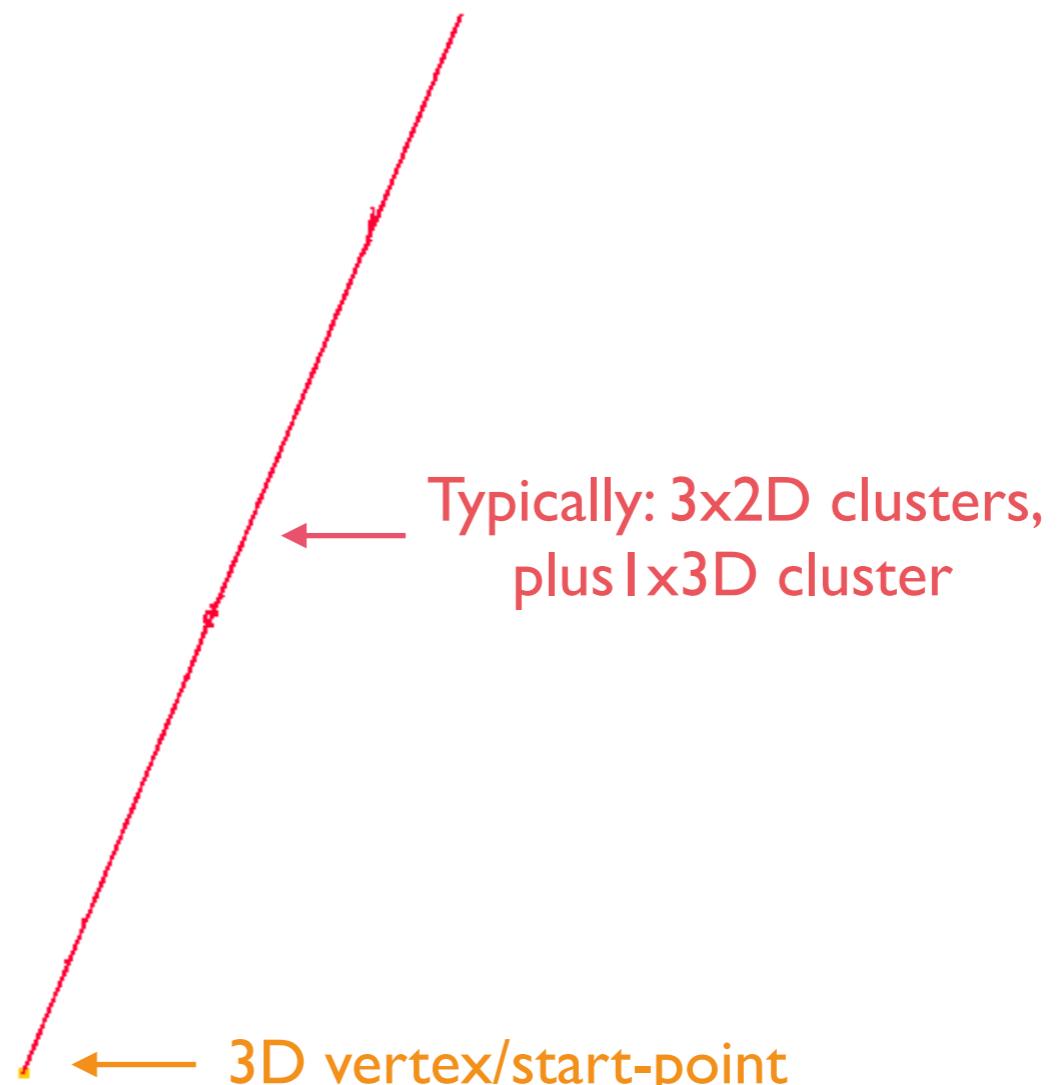


Pandora Example Events

Cosmic Ray Muon: Display 1/2

The reconstructed cosmic ray contains:

- Particle metadata
- A 3D vertex/start-point
- A list of 2D and 3D muon clusters
- A list of daughter particles

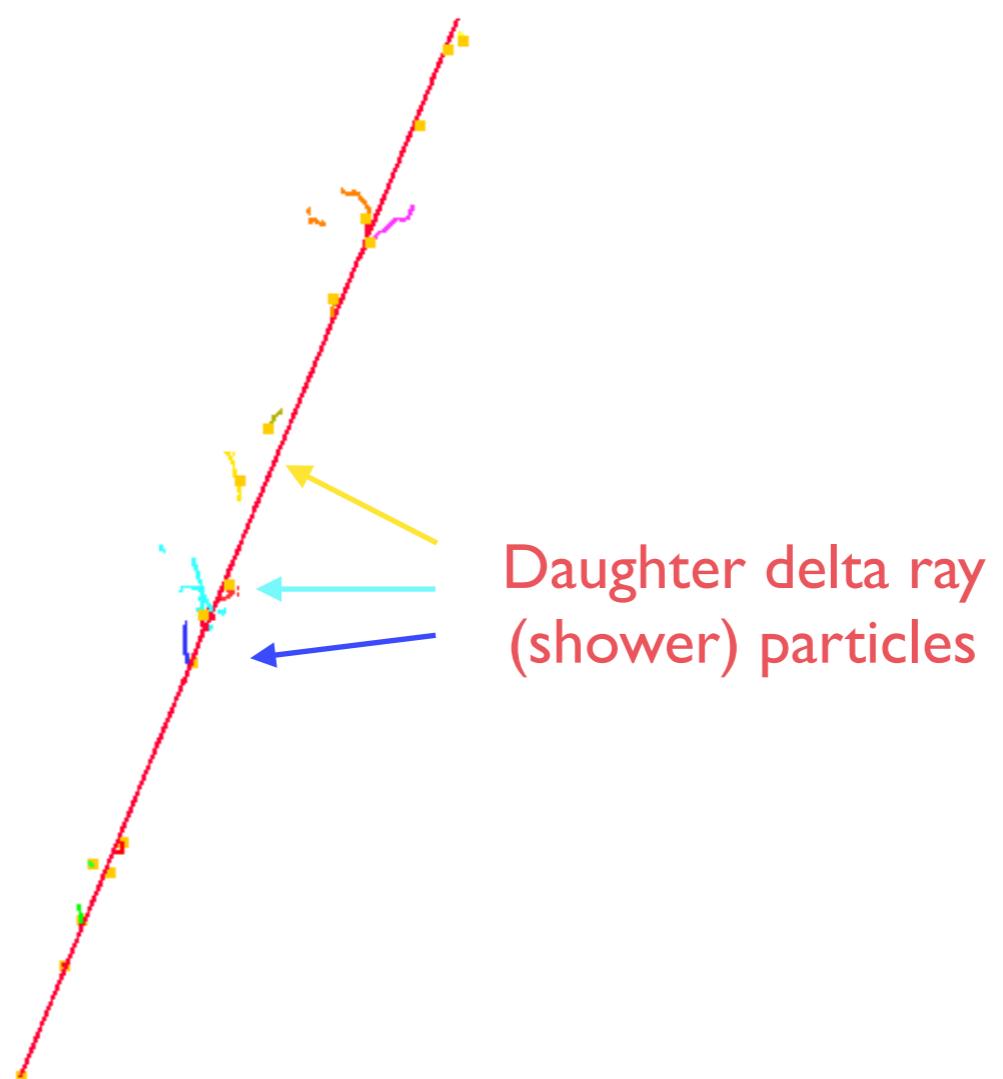


Pandora Example Events

Cosmic Ray Muon: Display 2/2

+ Daughter delta-rays, each of which has:

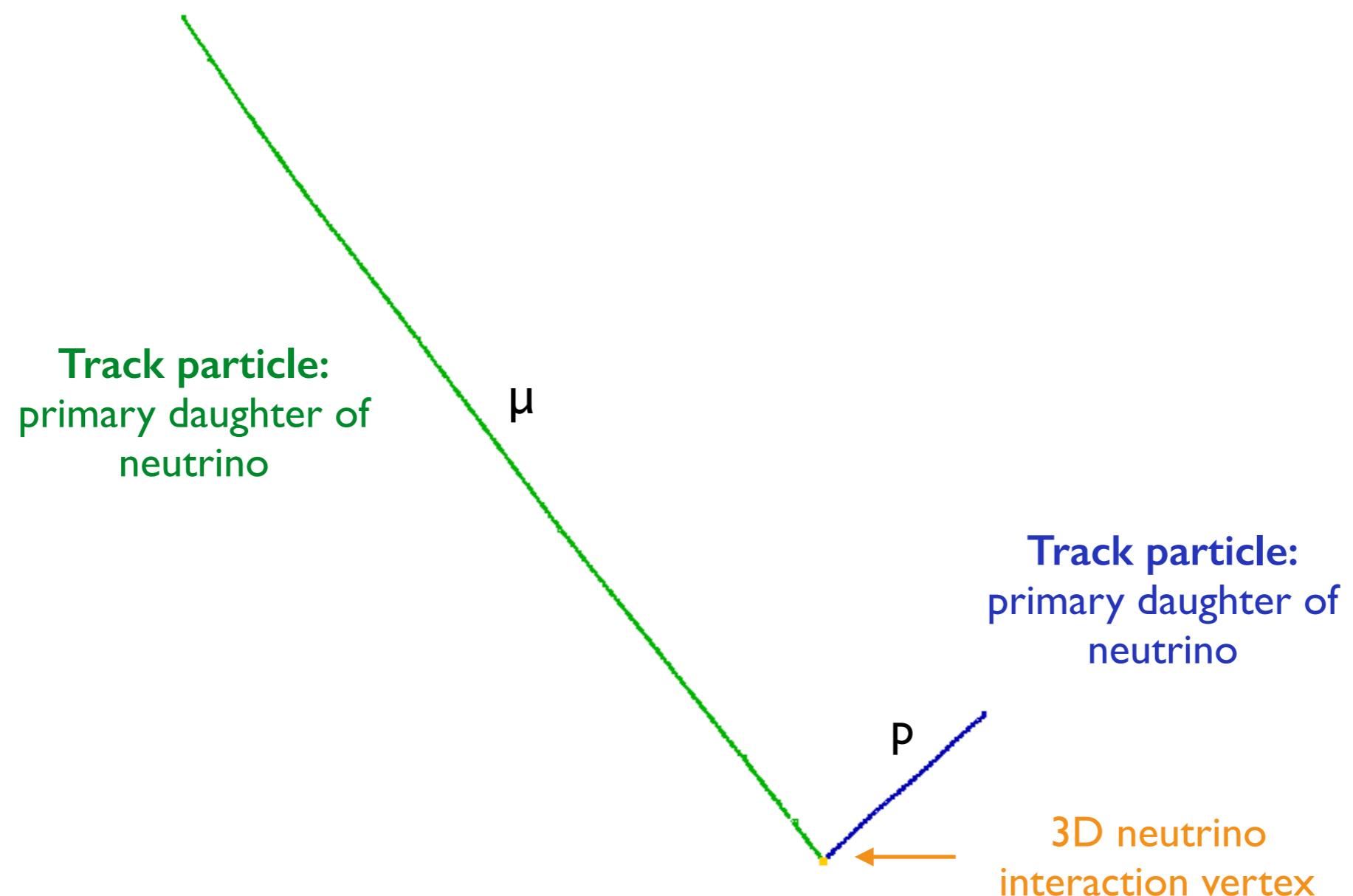
- Particle metadata
- A list of 2D clusters and a 3D cluster
- A 3D vertex position





Pandora Example Events

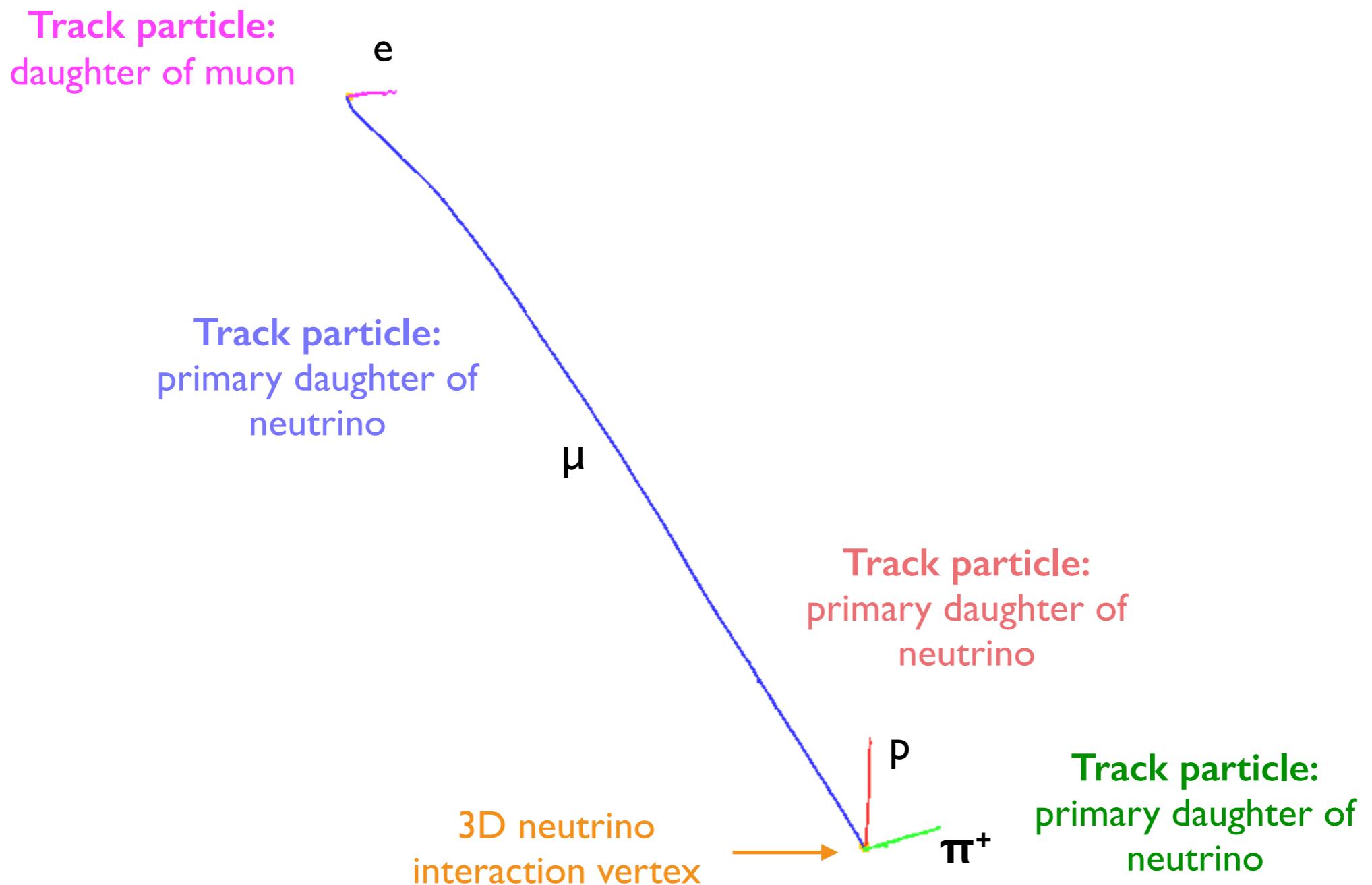
BNB ν_μ CC QEL: Combined display





Pandora Example Events

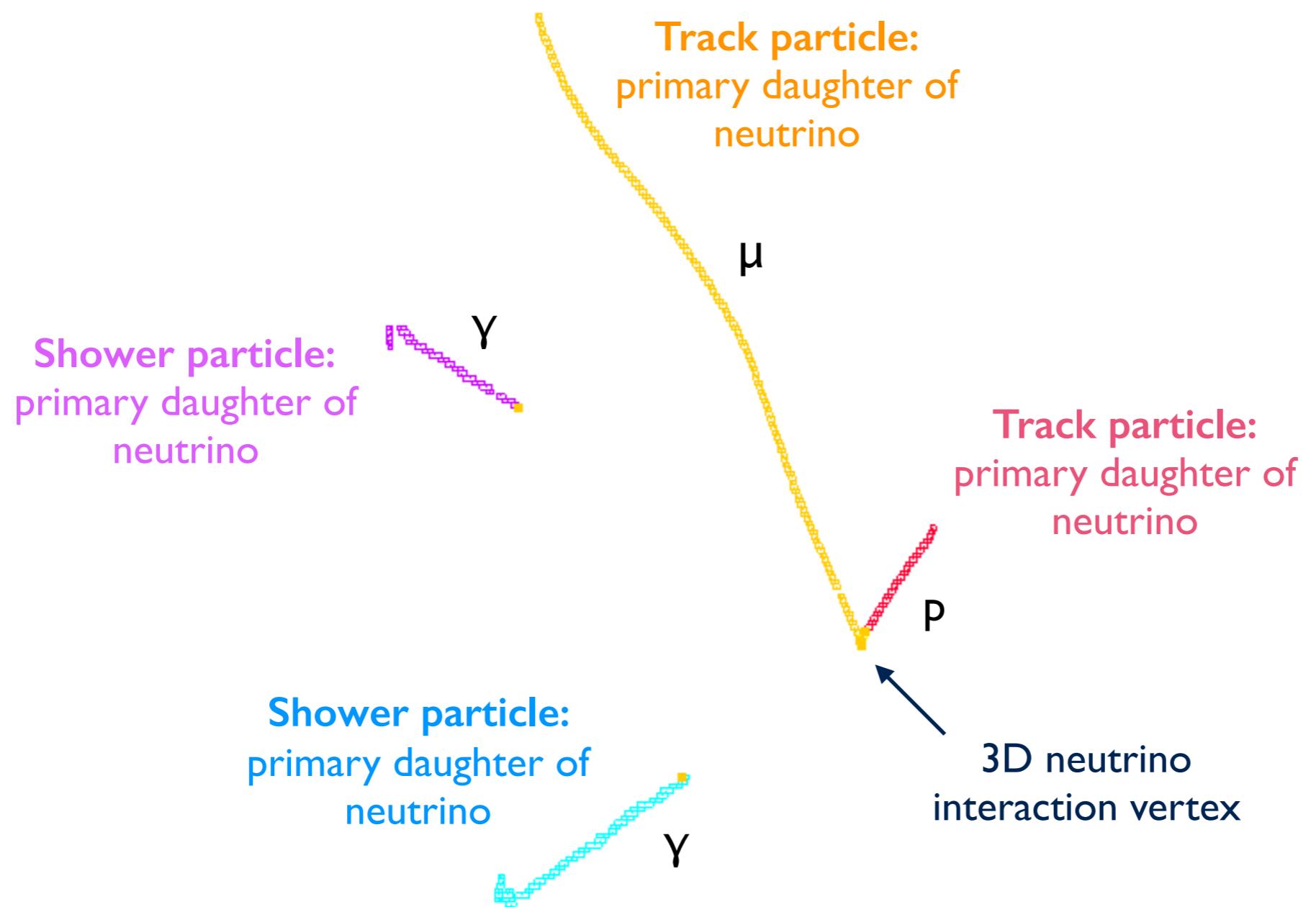
BNB ν_μ CC RES μ, p, π^+ : Combined display





Pandora Example Events

BNB ν_μ CC RES μ, p, π^0 : Combined display





Pandora LAr TPC Algorithms Key Points



- Algs mostly developed in context of MicroBooNE, but 35-t (DUNE) applications exist.
- Pandora delivers particles, e.g. for MicroBooNE get list of “Cosmics” + “Neutrinos”
- Pandora particles contain rich information, much more than just tracks or clusters.
- The algs perform rather well in tests using MicroBooNE simulation. See next slides...



Pandora Performance Metrics

- Use performance metrics to assess reconstruction output and drive development:
 - Look at specific types of neutrino interactions in MicroBooNE simulation.
 - Carefully match reconstructed particles to each true (primary) particle.
 - Count reconstructed particles for each true particle and assess quality of matches.
- Reconstructing particles in a LAr TPC is a new challenge, but so too is understanding how well the reconstruction works! Does e.g. adding a new algorithm make things better or worse?
- Have developed a well-defined approach, but it is **not very forgiving**. Events with minor errors, dismissed by eye (i.e. output agrees with visual inspection), are often classed as failures.
- **Striving for perfection - look to accurately match one reco particle to each true particle.**



Pandora Performance Metrics

I. Determine the primary true particle in each 2D hit.

- Take the particle with the largest energy contribution.

True particles must have
 ≥ 15 true hits

2. Match reconstructed particles to true particles:

- For each reco/true combination, find the number of ‘matched’ 2D hits (common to both the reco and true particles). Fold all daughter reco and true particles back into parent primaries.
- Matching algorithm, find all “strong” matches, then pick-up remaining “weak” matches:
 - i. Find the strongest (most shared hits) match between any reco and true particle
 - ii. Repeat step i, using reco and true particles at most once, until no further matches possible
 - iii. Assign any remaining reco particles to the true particle with which they share most hits

3. ‘Efficiency’ = fraction of true particles with at least one matched reco particle.

‘Completeness’ = fraction of 2D hits in true particle shared with the reco particle.

‘Purity’ = fraction of 2D hits in reco particle shared with the true particle.



Pandora Performance

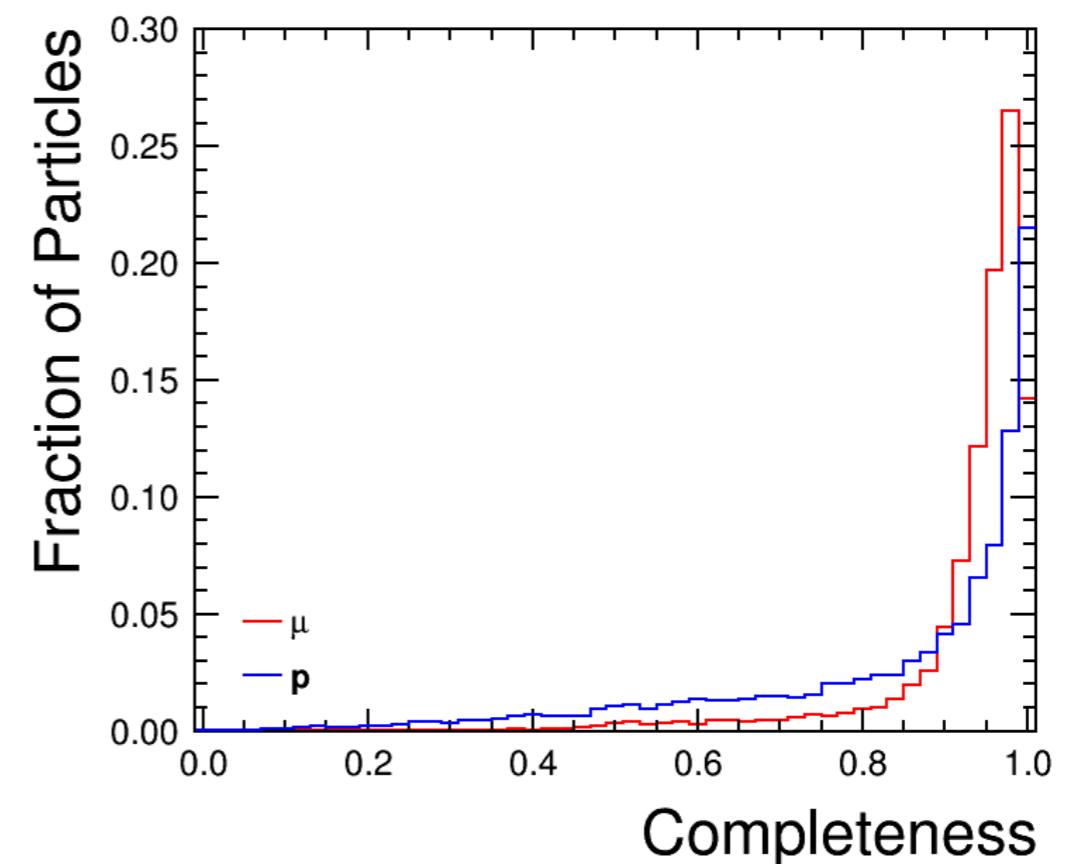
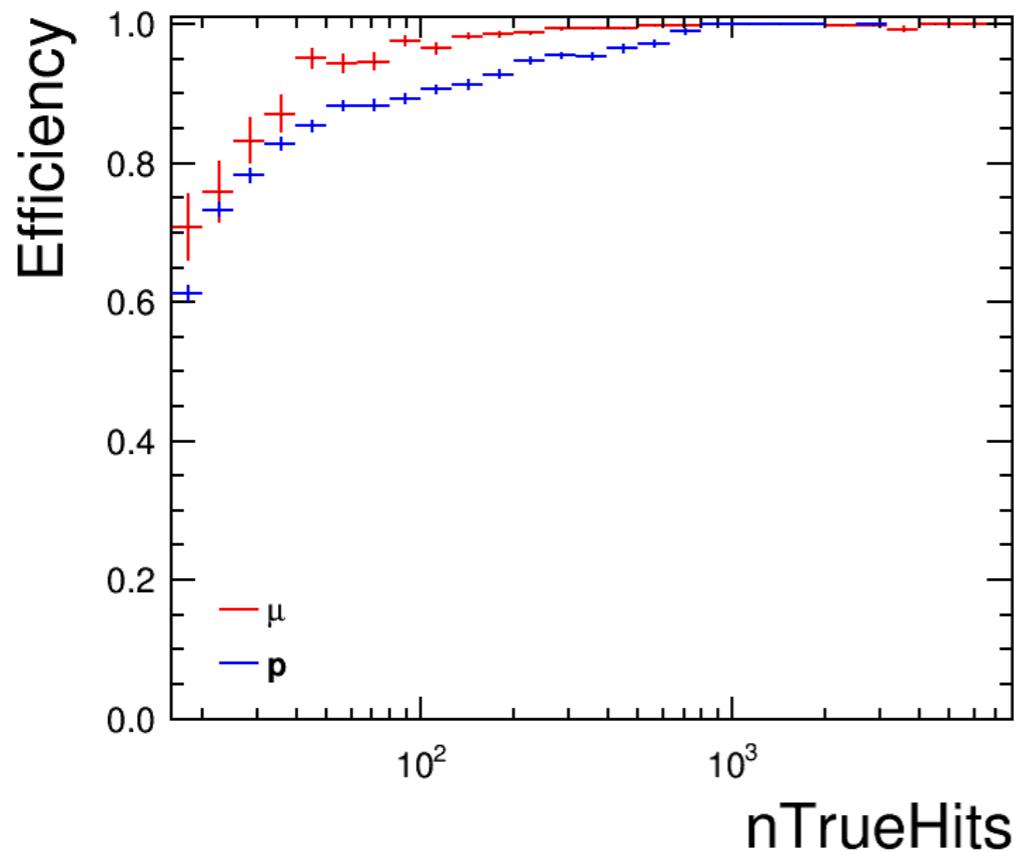
MicroBooNE simulation: BNB ν_μ CC QEL μ, p

| #MatchedPFOs | 0 | 1 | 2 | 3+ | |
|--------------|-------|-------|--------|-------|-----|
| Particles | μ | 231 | 20,346 | 1,430 | 139 |
| | p | 3,079 | 18,212 | 740 | 115 |

#Events: 22,146

#Perfect: 78%

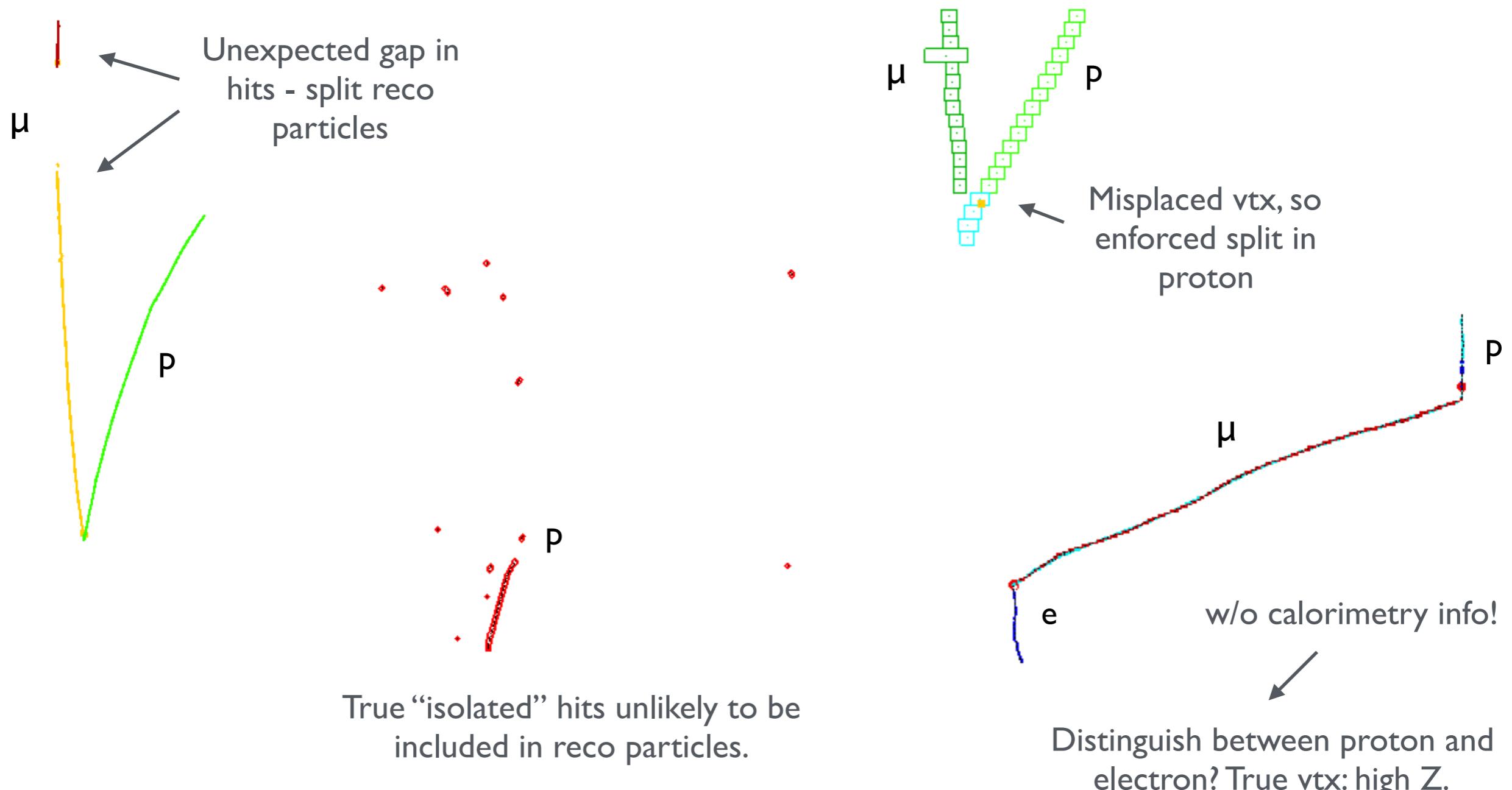
If muon and proton merged into a single reco particle, this particle is more likely to be matched to muon





Pandora Performance

What kind of things go wrong? e.g. BNB ν_μ CC QEL μ, p





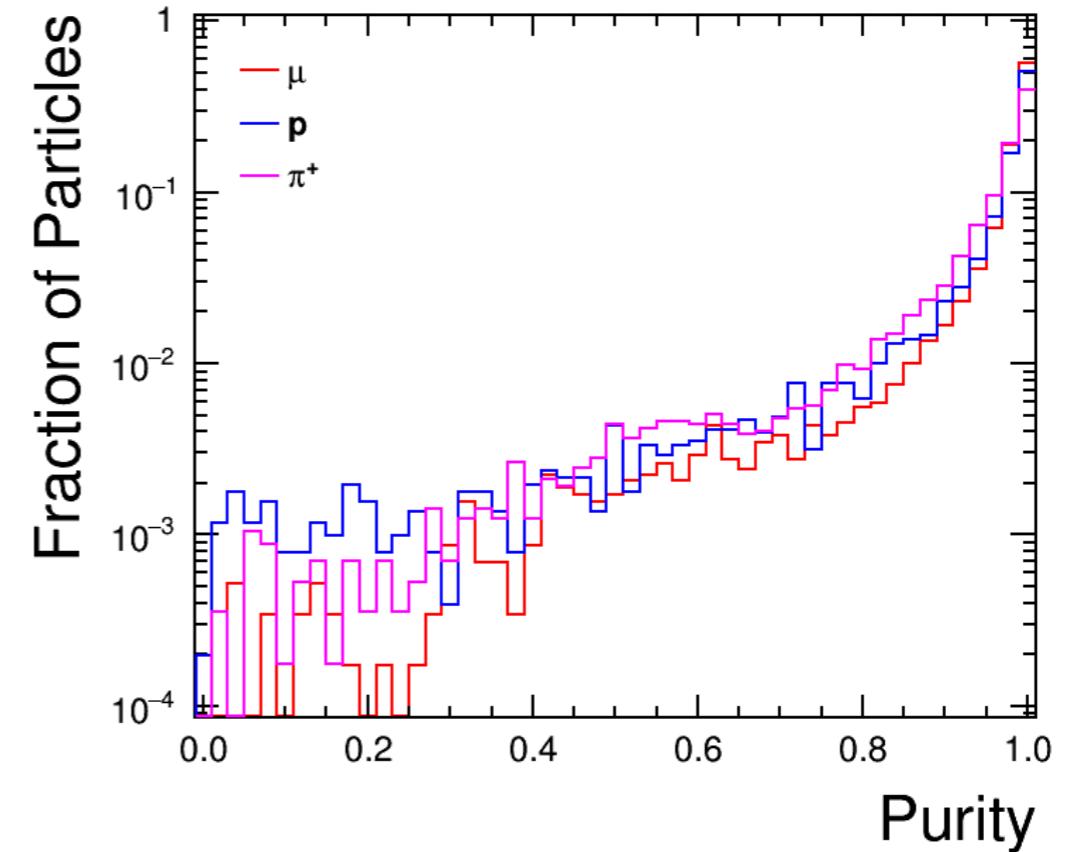
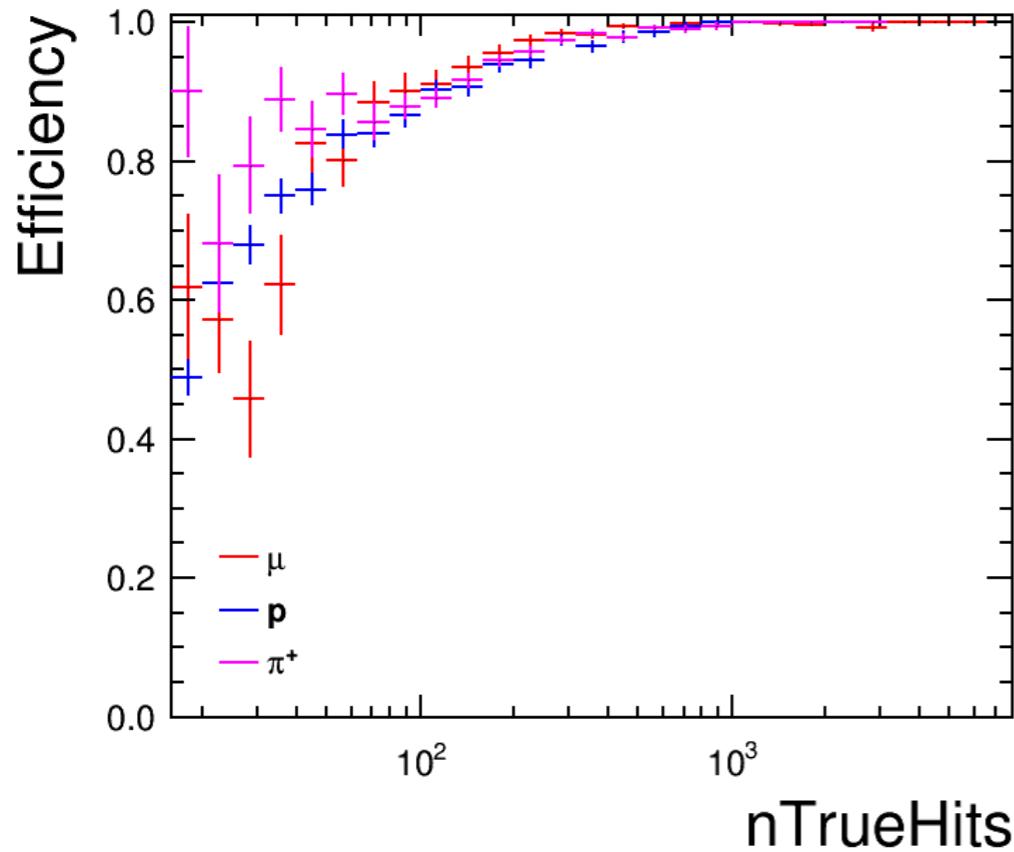
Pandora Performance

MicroBooNE simulation: BNB ν_μ CC RES μ, p, π^+

| #MatchedPFOs | 0 | 1 | 2 | 3+ | |
|--------------|---------|-----|-------|-----|-----|
| Particles | μ | 202 | 5,474 | 321 | 20 |
| | p | 910 | 4,772 | 296 | 40 |
| | π^+ | 322 | 4,619 | 798 | 279 |

#Events: 6,018

#Perfect: 58%





Pandora Performance

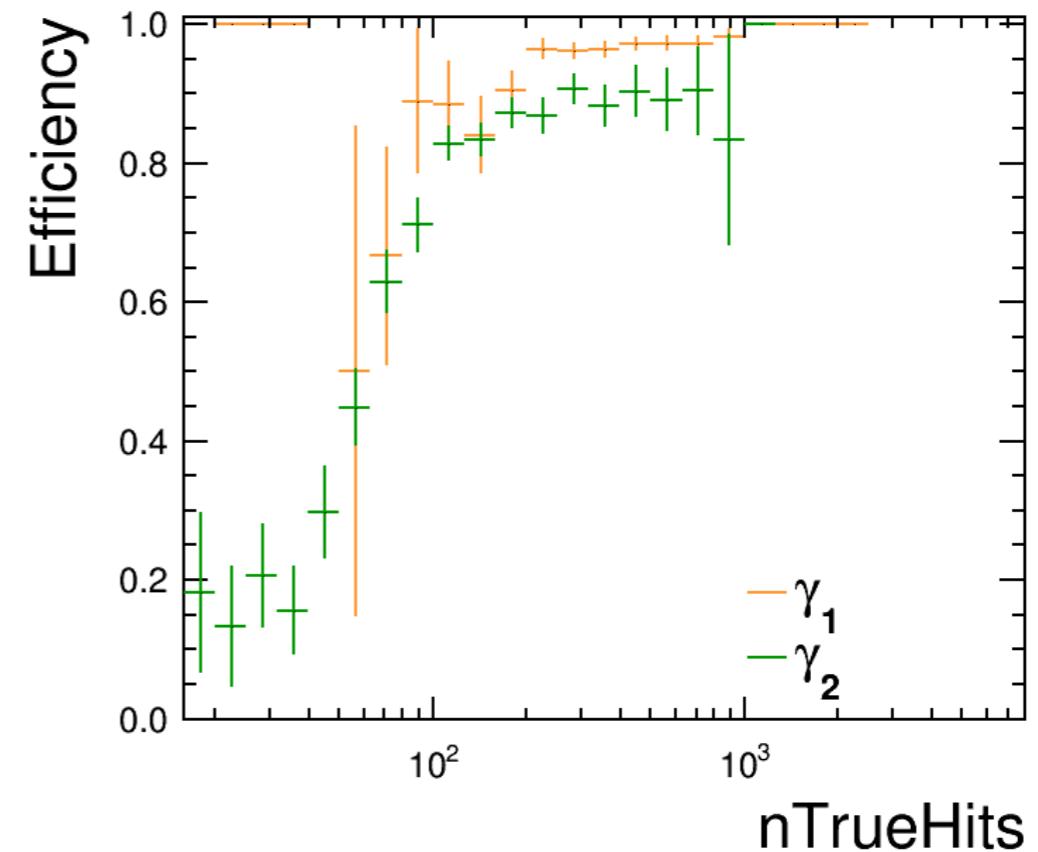
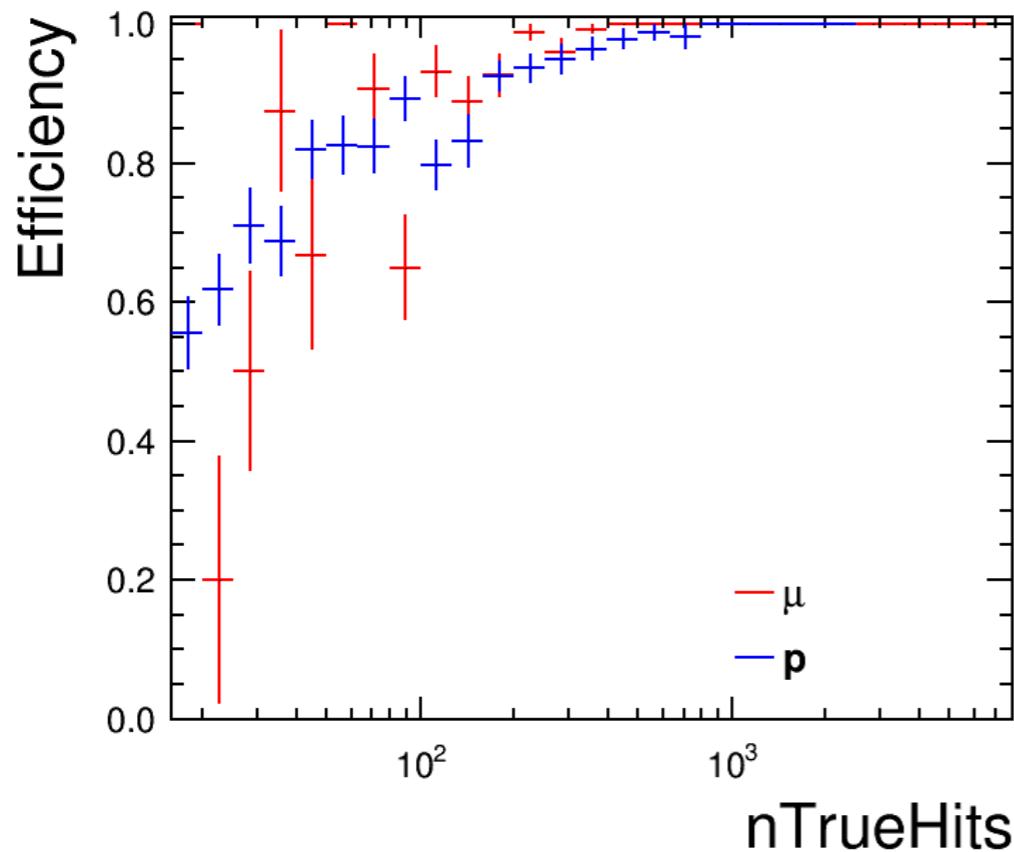
MicroBooNE simulation: BNB ν_μ CC RES μ, p, π^0

| #MatchedPFOs | 0 | 1 | 2 | 3+ | |
|--------------|------------|-----|-------|-----|-----|
| Particles | μ | 54 | 1,544 | 87 | 10 |
| | p | 261 | 1,294 | 123 | 17 |
| | γ_1 | 71 | 942 | 433 | 249 |
| | γ_2 | 394 | 1,004 | 211 | 86 |

#hits(γ_1) > #hits(γ_2)

#Events: 1,695

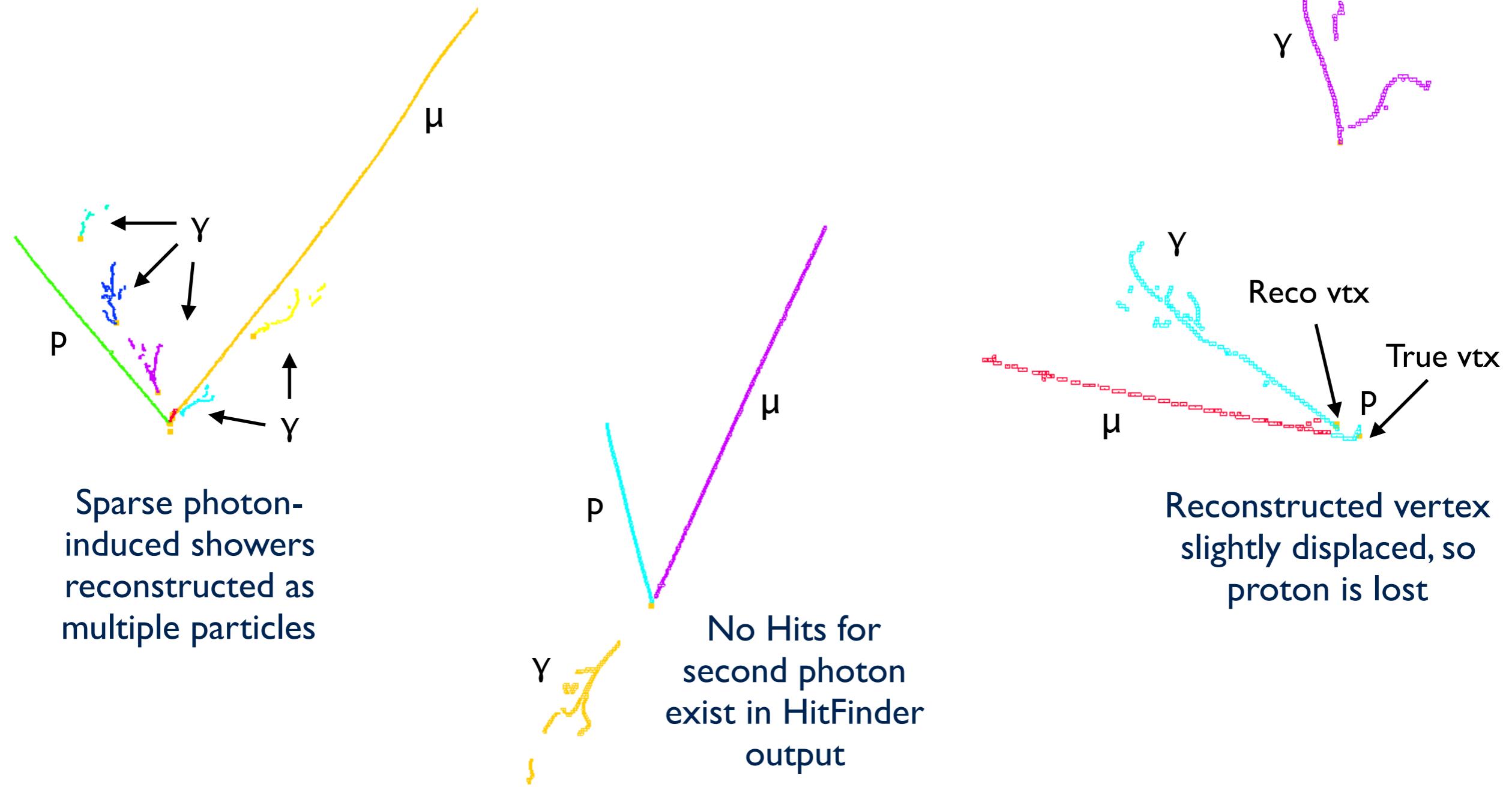
#Perfect: 24%





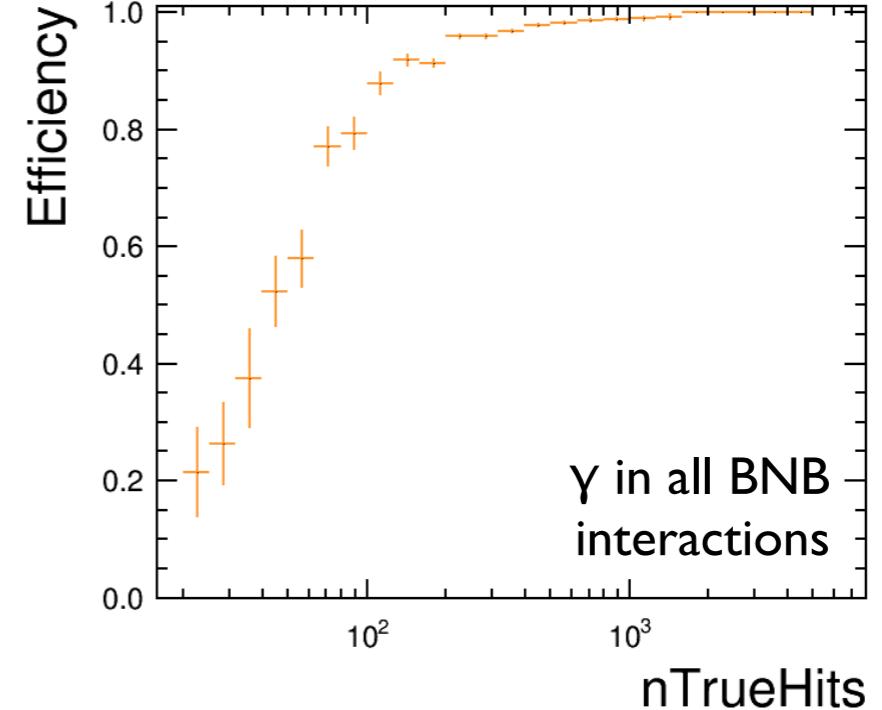
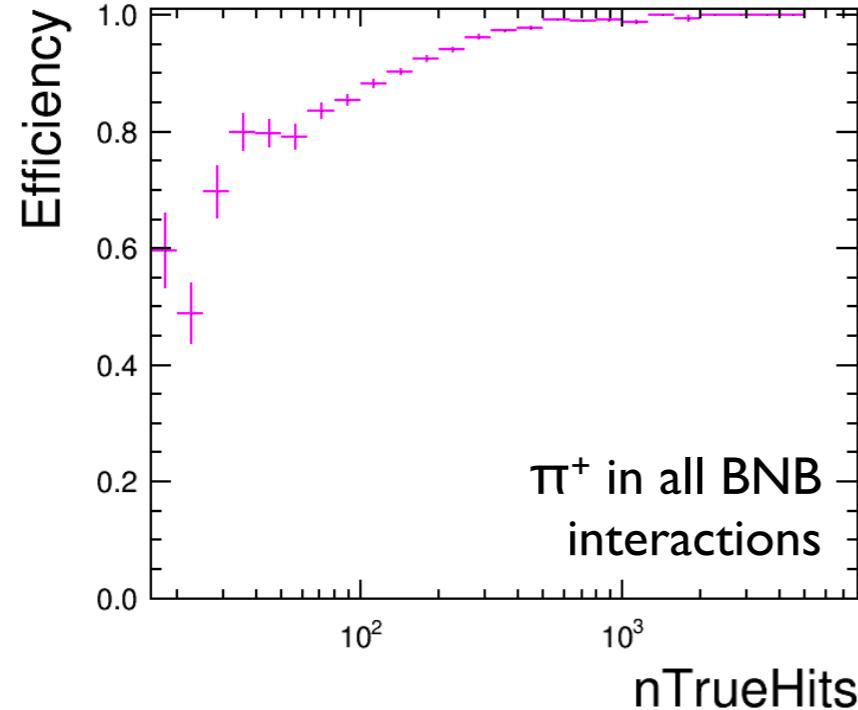
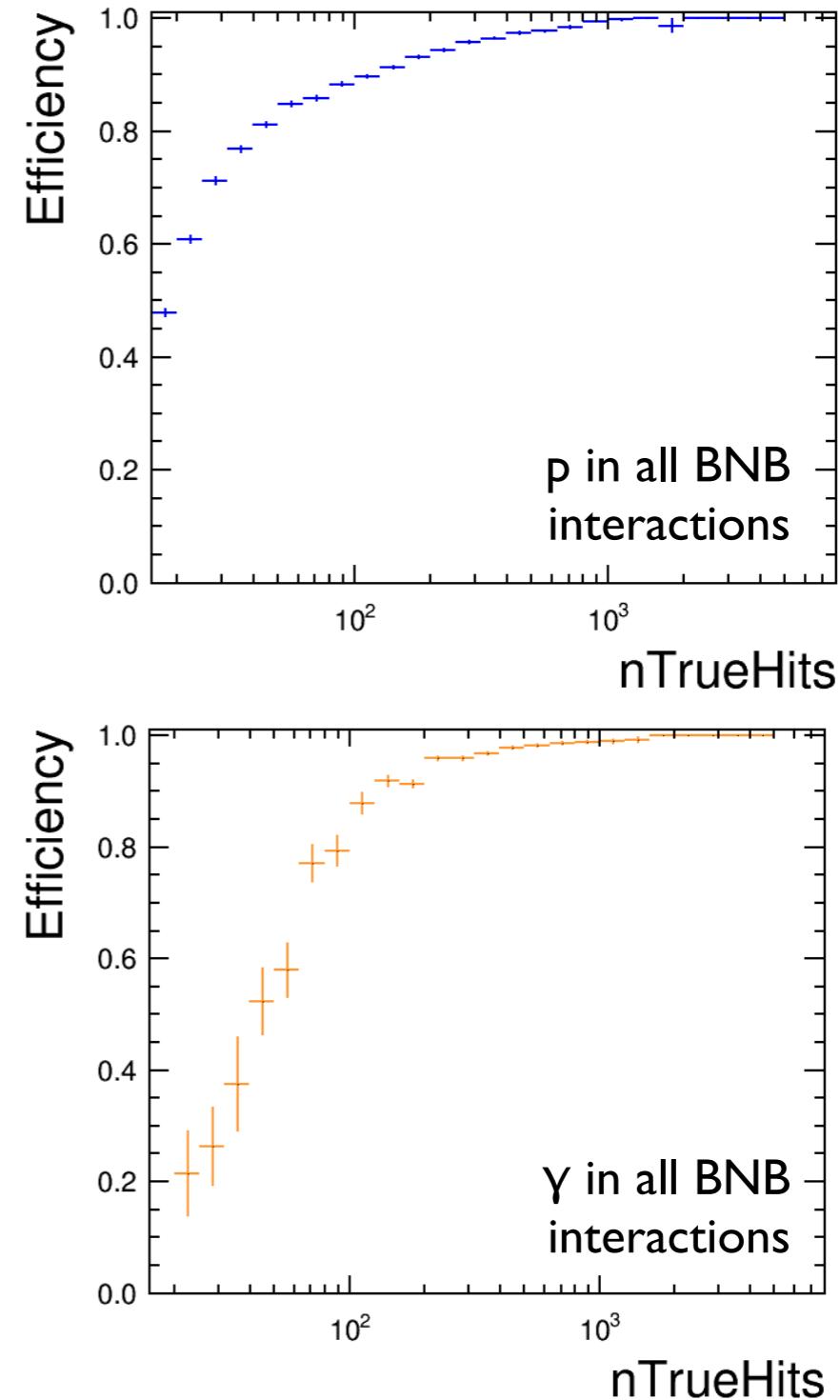
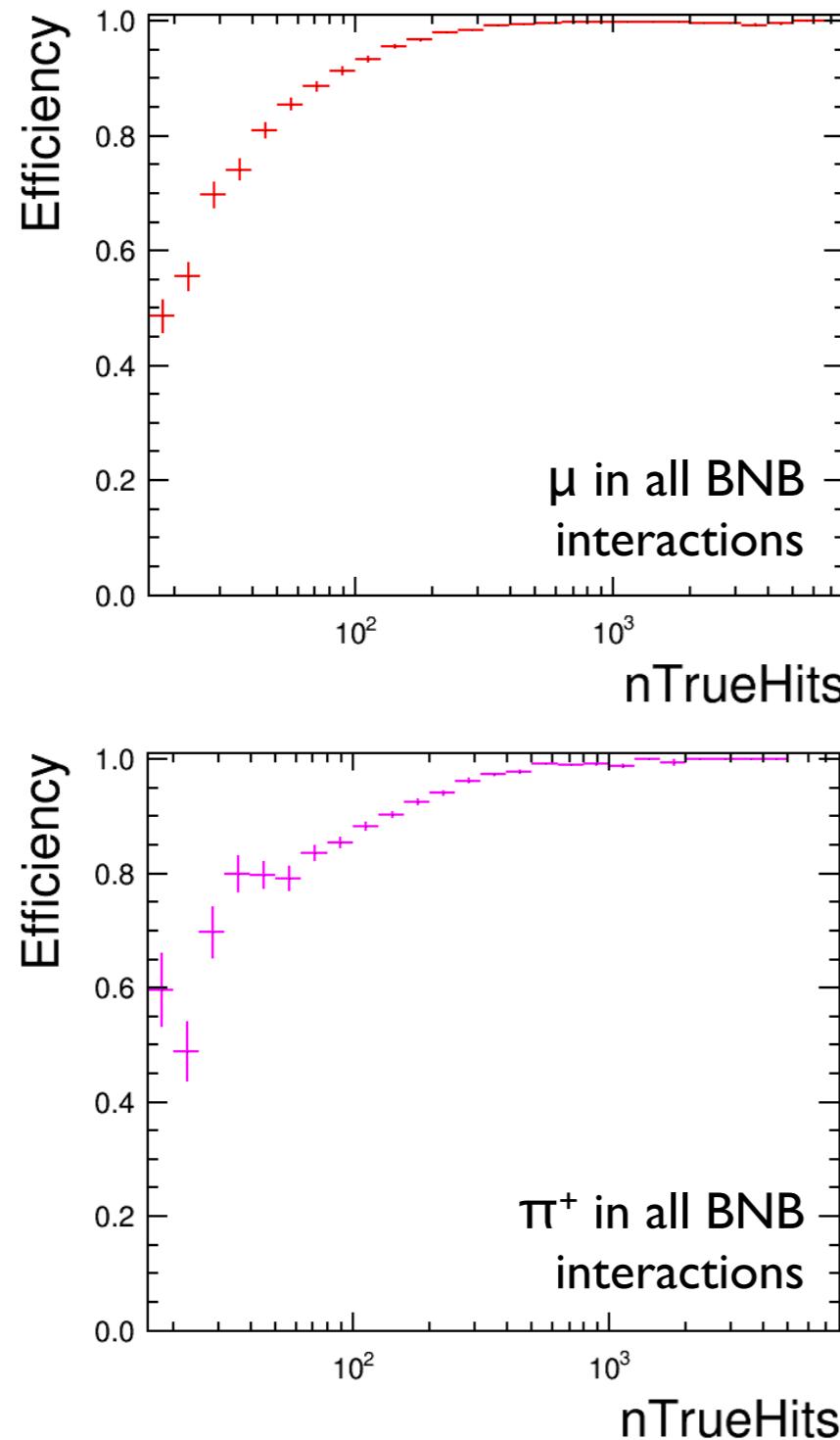
Pandora Performance

What kind of things go wrong? e.g. BNB ν_μ CC RES μ, p, π^0





Pandora Performance

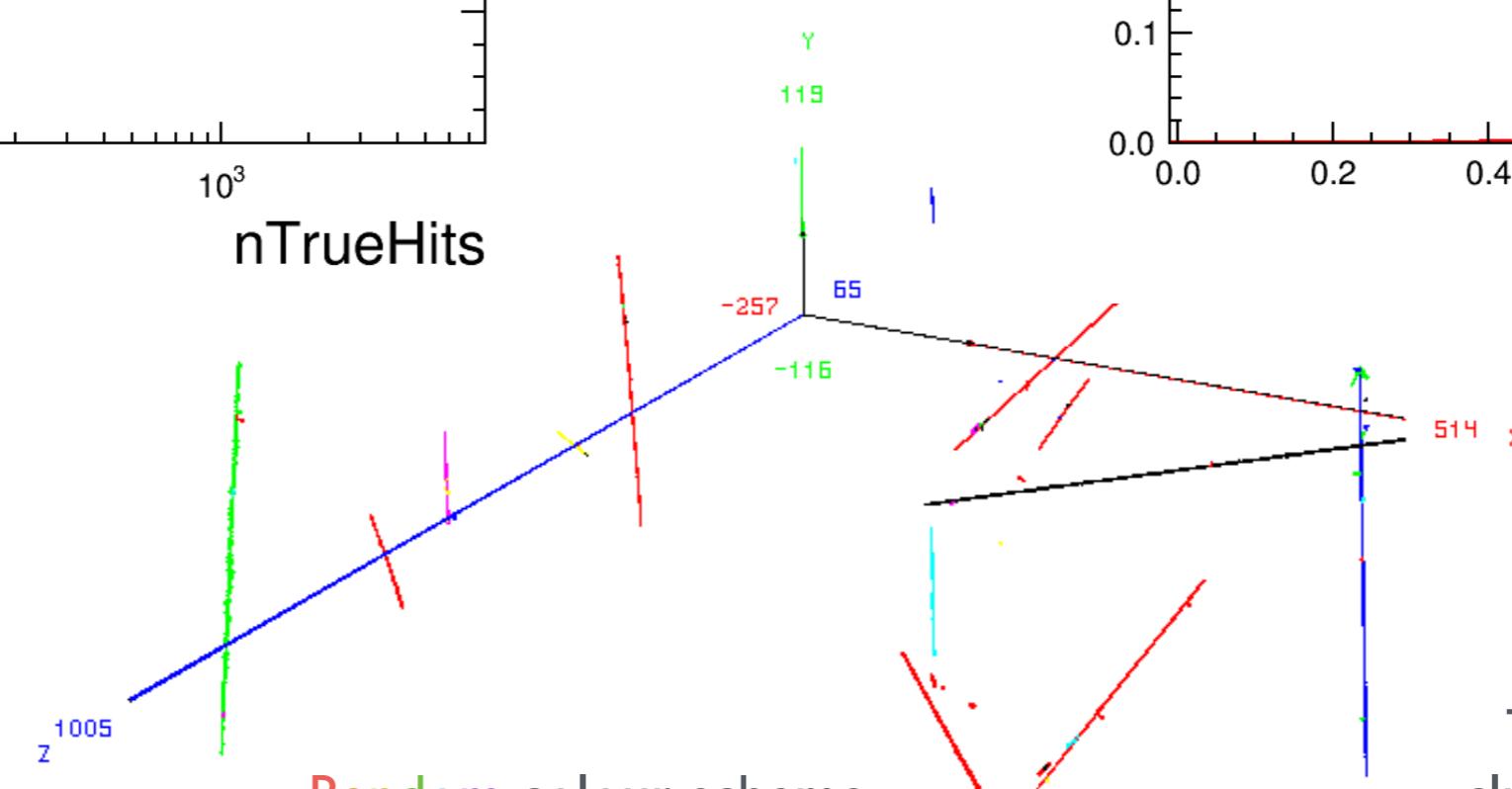
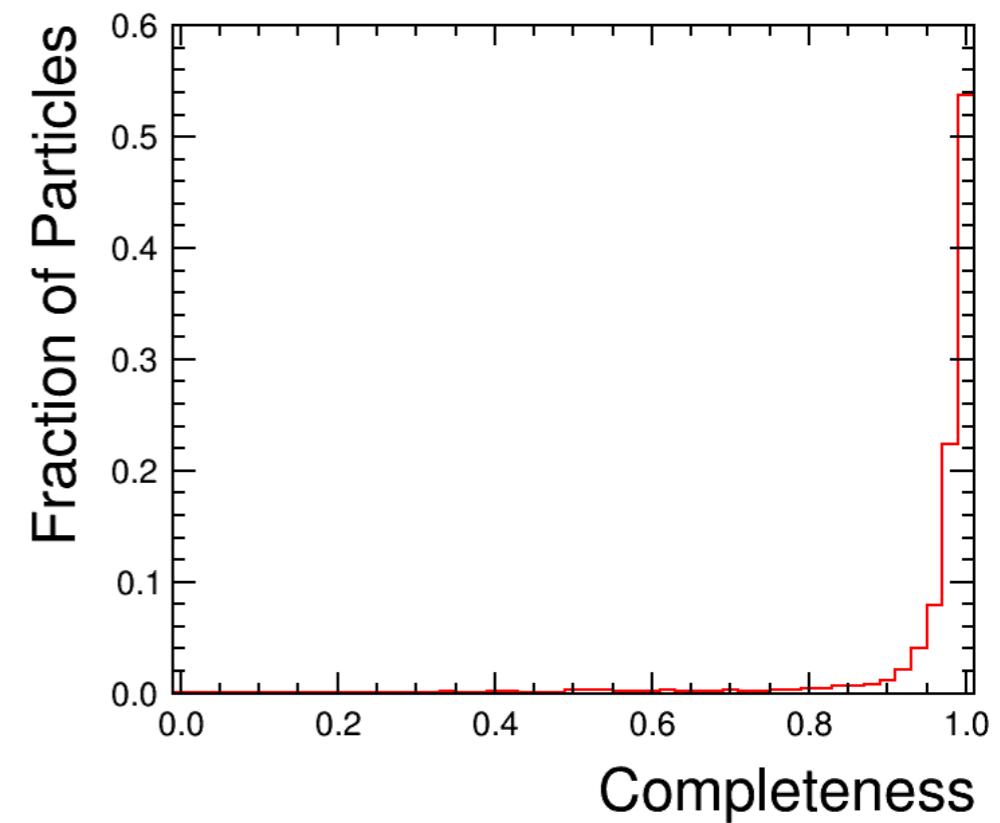
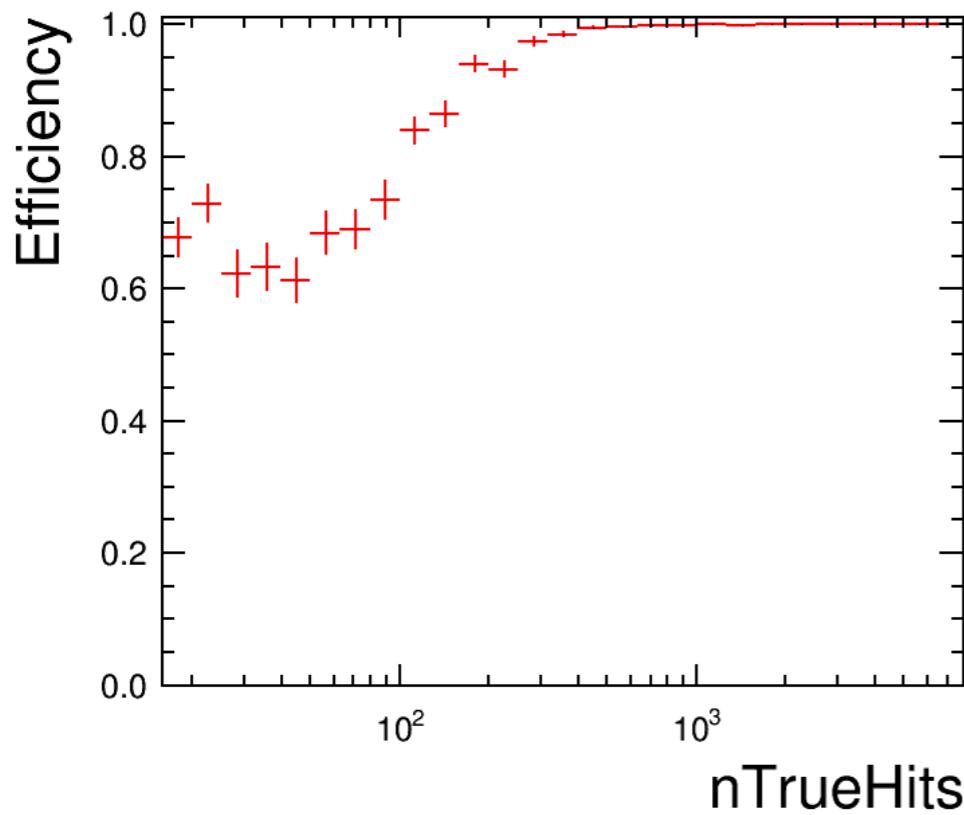


“Lost” long particles e.g. muons? Typically occur when multiple long, true particles are merged into a single reco particle. This is then matched to only one of the true particles (based on #hits shared).



Pandora Performance

MicroBooNE simulation: Cosmic Ray Muons

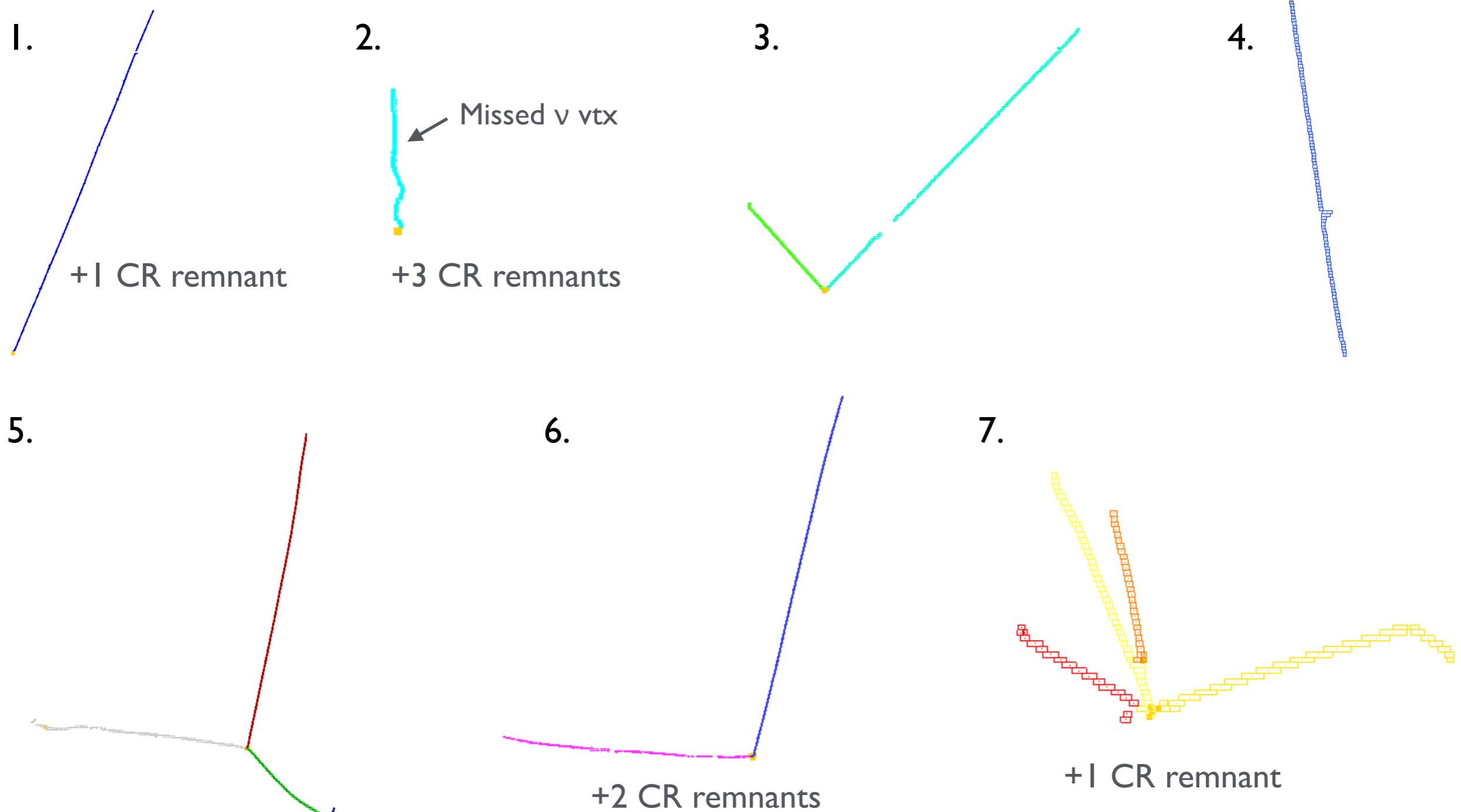


Typical 3D
cluster display



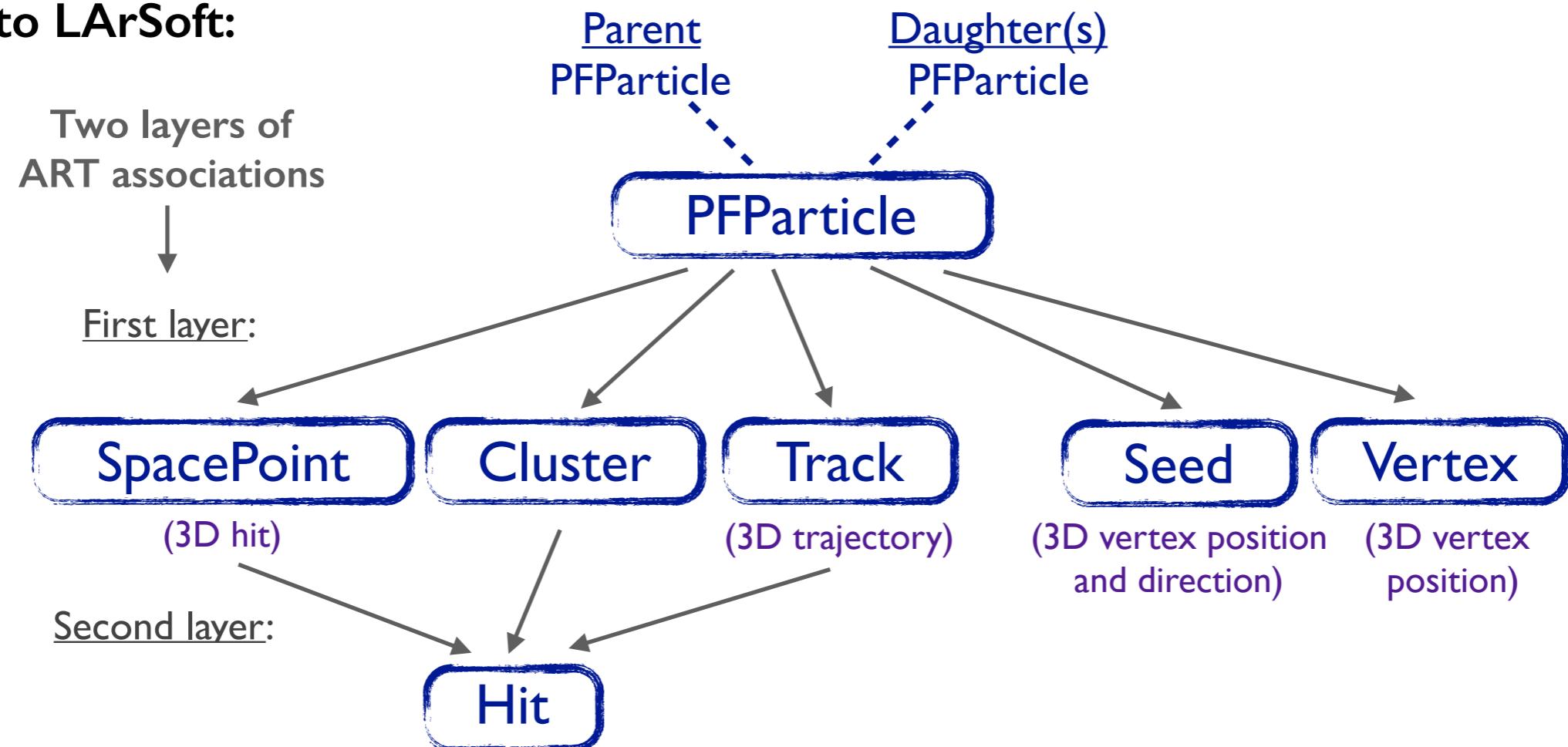
Pandora Performance

Example BNB nu events, after CR removal and Pandora 3D event slicing (first 7 in file):



Pandora Output

Output to LArSoft:



- LArSoft output initially looks rather complex, but, once Showers have been added, should provide a comprehensive translation of results obtained in the Pandora Event Data Model.
- Extremely important that we now get people to try/test this new PFPARTICLE output, checking that all the information from Pandora is persisted correctly.
- Help would be greatly appreciated to complete the LArSoft output by, i), producing Shower objects and, ii), discovering why the Kalman Filter vetoes many clean-looking Pandora Tracks.



Pandora Output

- Example investigation of Pandora output in LArSoft: Use prodgenie_bnb_nu_uboone.fcl, standard_g4_uboone.fcl, standard_detsim_uboone.fcl, then reco_uboone_stage_1.fcl
- reco_uboone_stage_2.fcl then runs neutrino reco pass (stage 1 is CR reco). Use a new art analyzer to examine performance, w/ some art functionality provided by LArPandora app.

```
void PFParticleMonitoring::analyze(const art::Event &evt)
{
    // Collect Hits
    HitVector hitVector;
    LArPandoraCollector::CollectHits(evt, m_hitfinderLabel, hitVector);

    // Collect PFParticles and match Reco Particles to Hits
    PFParticlesToHits recoParticlesToHits;
    HitsToPFParticles hitsToRecoParticles;
    LArPandoraCollector::BuildPFParticleHitMaps(evt, m_particleLabel, m_clusterLabel, recoParticlesToHits, hitsToRecoParticles,
                                                 LArPandoraCollector::kAddDaughters);

    // Collect MCParticles and match True Particles to Hits
    MCParticlesToHits trueParticlesToHits;
    HitsToMCParticles hitsToTrueParticles;
    LArPandoraCollector::BuildMCParticleHitMaps(evt, m_geantModuleLabel, hitVector, trueParticlesToHits, hitsToTrueParticles,
                                                 LArPandoraCollector::kAddDaughters);

    // Match Reco Particles to True Particles
    ParticleMatchingMap particleMatchingMap;
    this->GetRecoToTrueMatches(recoParticlesToHits, trueParticlesToHits, hitsToTrueParticles, particleMatchingMap);

    for (const ParticleMatchingMap::value_type &trueToRecoEntry : particleMatchingMap)
    {
        std::cout << "MCPDG " << trueToRecoEntry.first->PdgCode()
            << ", #MCHits " << trueParticlesToHits.at(trueToRecoEntry.first).size() << std::endl;

        for (const RecoParticleToNMatchedHits::value_type &recoToNMatchedHitsEntry : trueToRecoEntry.second)
        {
            std::cout << "--RecoPDG " << recoToNMatchedHitsEntry.first->PdgCode()
                << ", #PfoHits " << recoParticlesToHits.at(recoToNMatchedHitsEntry.first).size()
                << ", #MatchedHits " << recoToNMatchedHitsEntry.second << std::endl;
        }
    }
}
```

Use LArSoft know-how to build maps:

reco particles → associated hits
true particles → associated hits
hits → associated true particles

E.g. Print
matching details



Pandora Output

```
void PFParticleMonitoring::GetRecoToTrueMatches(const PFParticlesToHits &recoParticlesToHits, const MCParticlesToHits &trueParticlesToHits,
                                                const HitsToMCParticles &hitsToTrueParticles, ParticleMatchingMap &particleMatchingMap) const
{
    // Create a placeholder entry for all mc particles with >0 hits
    for (const MCParticlesToHits::value_type &trueParticleToHitsEntry : trueParticlesToHits)
    {
        if (!trueParticleToHitsEntry.second.empty())
            (void) particleMatchingMap.insert(ParticleMatchingMap::value_type(trueParticleToHitsEntry.first, RecoParticleToNMatchedHits()));

    }

    // Store true to reco matching details
    for (const PFParticlesToHits::value_type &recoParticleToHitsEntry : recoParticlesToHits)
    {
        const art::Ptr<recob::PFParticle> pRecoParticle(recoParticleToHitsEntry.first);
        const HitVector &hitVector(recoParticleToHitsEntry.second);

        for (const art::Ptr<recob::Hit> pHit : hitVector)
        {
            HitsToMCParticles::const_iterator trueParticleIter = hitsToTrueParticles.find(pHit);

            if (hitsToTrueParticles.end() == trueParticleIter)
                continue;

            const art::Ptr<simb::MCParticle> pTrueParticle = trueParticleIter->second;
            particleMatchingMap[pTrueParticle][pRecoParticle]++;
        }
    }
}
```

Populate mapping:
True particle → Reco particle → nMatchedHits

MCPDG 13, #MCHits 430
--RecoPDG 13, #PfoHits 420, #MatchedHits 416

MCPDG 2212, #MCHits 109
--RecoPDG 13, #PfoHits 105, #MatchedHits 105
--RecoPDG 13, #PfoHits 420, #MatchedHits 4

Typical output for a BNB
CC QEL μ, p event



Pandora Development Key Points

- Pandora provides an easy and fast development platform. It has no external dependencies beyond ROOT, which it uses purely for event visualisation and monitoring purposes.
- Visualisation APIs provide simple access to user-customised event displays in algorithms, enabling a rapid and rewarding visual approach to debugging/development.
- Pandora can provide a complete standalone environment for rapid development. Need only run the LArSoft app once to persist input Hits in Pandora binary or XML formats.
- Pandora is ideally suited for distributed development i.e. people can work on standalone algorithms, which can then be slotted into the reconstruction via simple XML config.

<https://www.github.com/PandoraPFA>

EPJC.75.439



Pandora Summary

- Pandora brings together a large number of exciting new pattern recognition ideas for LAr TPC reconstruction in a novel multi-algorithm approach.
- The multi-algorithm approach is realised using the advanced functionality provided by the well-documented Pandora SDK, building on its successful use at collider experiments.
- The Pandora LAr TPC algorithms continue to develop and improve. There are already a number of key algorithms available that are extremely sophisticated and well-tested.
- We strongly encourage people to start examining the Pandora LArSoft outputs, alongside the “other/default” reconstruction, and provide feedback to drive future development.
- We very strongly welcome new developers!



Pandora Documentation



Pandora Documentation



<http://arxiv.org/abs/1506.05348>

or EPJC.75.439 or PandoraPFA github page

The Pandora Software Development Kit for Pattern Recognition

J. S. Marshall^{a,*}, M. A. Thomson^a

^a*Cavendish Laboratory, University of Cambridge, Cambridge, United Kingdom*

Abstract

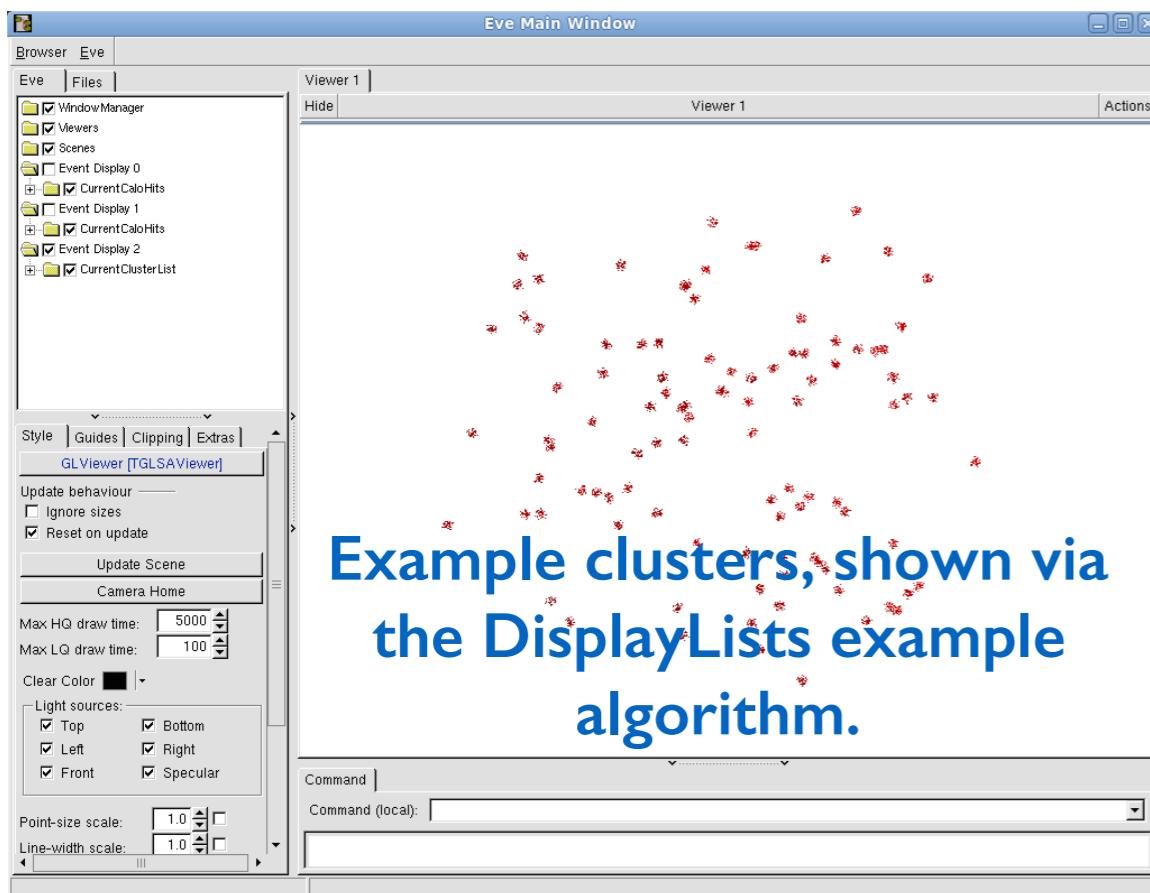
The development of automated solutions to pattern recognition problems is important in many areas of scientific research and human endeavour. This paper describes the implementation of the Pandora Software Development Kit, which aids the process of designing, implementing and running pattern recognition algorithms. The Pandora Application Programming Interfaces ensure simple specification of the building-blocks defining a pattern recognition problem. The logic required to solve the problem is implemented in algorithms. The algorithms request operations to create or modify data structures and the operations are performed by the Pandora framework. This design promotes an approach using many decoupled algorithms, each addressing specific topologies. Details of algorithms addressing two pattern recognition problems in High Energy Physics are presented: reconstruction of events at a high-energy e^+e^- linear collider and reconstruction of cosmic ray or neutrino events in a liquid argon time projection chamber.

Keywords: Software Development Kit, Pattern recognition, High Energy Physics



Pandora Learning Library

- Pandora algorithms create and/or modify clusters, vertices and PFOs. Their decisions (the algorithm logic) whether to proceed with operations can be complex and use-case specific.
- The aim of the Pandora ExampleContent library and test application is to demonstrate the key Pandora functionality in a very **simple testing and learning environment**.
- The ExampleContent library is structured in exactly the same manner as the LCContent and LArContent libraries, currently in use for Linear Collider and LAr TPC reconstruction.



- The library consists of example Algorithms, AlgorithmTools, Plugins and Helper functions:
 - Example list access and display
 - Example Cluster, Vertex and PFO creation
 - Cluster manipulation, including merging, deletion, fragmentation and reclustering
 - Creating and saving new lists of objects
 - Using Algorithm Tools and Plugins
 - Writing a tree using PandoraMonitoring.

<https://github.com/PandoraPFA/Documentation>