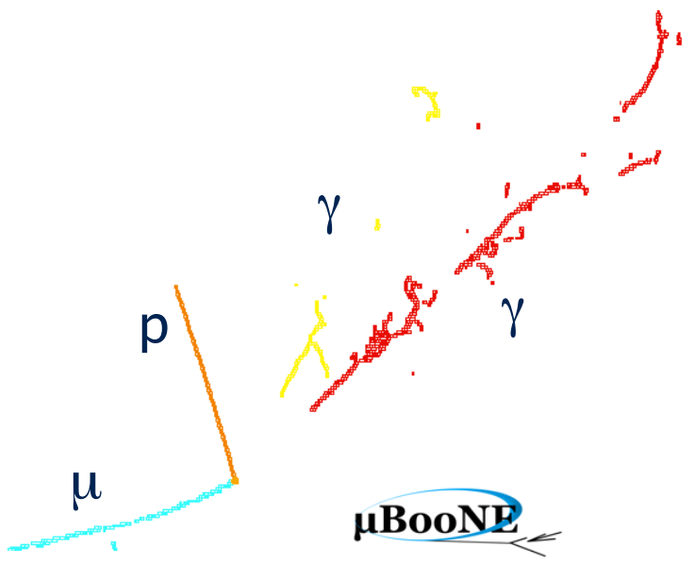


Pandora Talk 7: Shower and Event Reconstruction

J. S. Marshall for the Pandora Team

MicroBooNE Pandora Workshop

July 11-14th 2016, Cambridge

PandoraCosmic vs. PandoraNeutrino

- **Two reconstruction paths, as specified by relevant PandoraSetting XML files:**
  - PandoraSettings_MicroBooNE_Cosmic.xml
    PandoraSettings_MicroBooNE_Neutrino.xml
  - Cosmic pass: more strongly track-oriented; showers assumed to be delta rays, added as daughters of the muons; muon vertices at track high-y coordinate.
  - Neutrino pass: more careful to find interaction vertex and to protect particles emerging from vertex. Careful treatment to address track/shower tension.

- Neutrino pass begins by running fast/minimal version of the 2D reco, 3D track reco and 3D Hit creation. 3D Hits are then divided into slices, using proximity and direction-based metrics.
- Isolate neutrino interactions and/or cosmic-ray remnants in individual slices: original 2D Hits associated with a slice used as input to neutrino pass, each slice produces one neutrino Particle.

Event Slicing

NeutrinoParentAlgorithm

```
/**
 *  @brief  SlicingTool class
 */
class SlicingTool : public pandora::AlgorithmTool
{
public:
    /**
     *  @brief  Run the algorithm tool
     *
     *  @param  pAlgorithm address of the calling algorithm
     *  @param  caloHitListNames the hit type to calo hit list name map
     *  @param  clusterListNames the hit type to cluster list name map
     *  @param  sliceList to receive the populated slice list
     */
    virtual void Slice(const NeutrinoParentAlgorithm *const pAlgorithm, const NeutrinoParentAlgorithm::HitTypeToNameMap &caloHitListNames,
        const NeutrinoParentAlgorithm::HitTypeToNameMap &clusterListNames, NeutrinoParentAlgorithm::SliceList &sliceList) = 0;
};
```

Parent alg controls all operations:

Runs fast 3D reco algs

Calls slicing tool

Neutrino reco pass for each slice

NeutrinoParentAlgorithm

```
// ATTN Skipping-over some steps for brevity - view as pseudocode
SliceList sliceList;
m_pSlicingTool->Slice(this, m_caloHitListNames, m_clusterListNames, sliceList);

for (const Slice &slice : sliceList)
{
    for (const HitType hitType : m_hitTypeList)
    {
        // ATTN Also call 2D clustering alg here (removed for brevity)
        for (const std::string &algName : m_twoDAlgorithms)
            PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::RunDaughterAlgorithm(*this, algName));
    }

    // Subsequent reconstruction
    StringVector algorithms;
    algorithms.insert(algorithms.end(), m_vertexAlgorithms.begin(), m_vertexAlgorithms.end());
    algorithms.insert(algorithms.end(), m_threeDAlgorithms.begin(), m_threeDAlgorithms.end());
    algorithms.insert(algorithms.end(), m_mopUpAlgorithms.begin(), m_mopUpAlgorithms.end());
    algorithms.insert(algorithms.end(), m_threeDHitAlgorithms.begin(), m_threeDHitAlgorithms.end());
    algorithms.insert(algorithms.end(), m_neutrinoAlgorithms.begin(), m_neutrinoAlgorithms.end());

    for (const std::string &algName : algorithms)
        PANDORA_RETURN_RESULT_IF(STATUS_CODE_SUCCESS, !=, PandoraContentApi::RunDaughterAlgorithm(*this, algName));
}
```

Shower Reconstruction

- **Neutrino pass differs from cosmic pass by using 2D Clusters to identify interaction vertex, then, after 3D track algs, by providing more sophisticated shower reconstruction.**
- Try to add branches to any long Clusters that represent "shower spines". The spines may already exist in track Particles and any tracks that start to acquire multiple branches are deleted.
- The Particle is deleted and its 2D Clusters "released"; they will instead acquire the identified branches and will provide input to later 3D shower reconstruction algorithms.

Ultimately intend to perform shower growing in 3D (or 3x2D, simultaneously).

Current procedure rather clumsy: delete, grow then reform shower Particles.

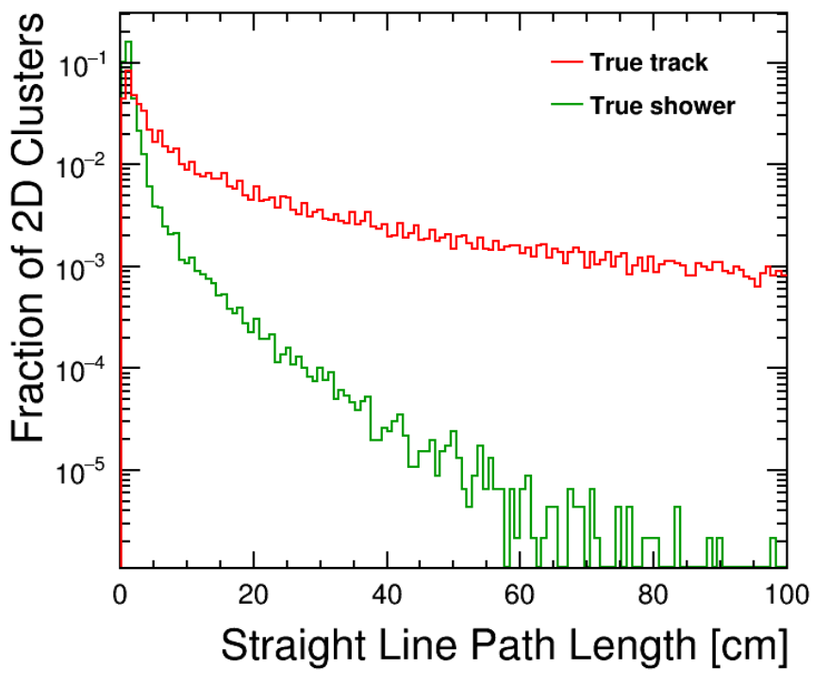
XML

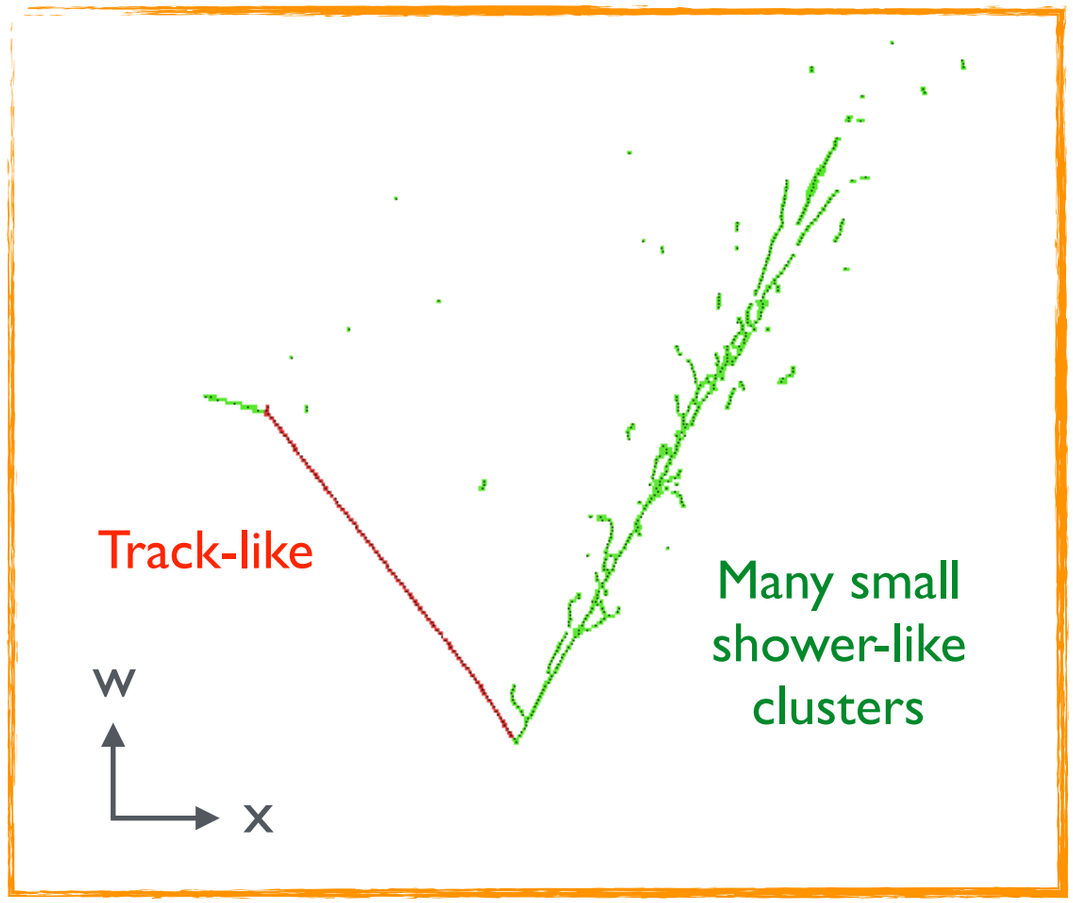
```
<!-- Shower reconstruction -->
<algorithm type = "LArClusterCharacterisation">
    <InputClusterListNames>ClustersU ClustersV ClustersW</InputClusterListNames>
</algorithm>
<algorithm type = "LArShowerGrowing">
    <InputClusterListNames>ClustersU ClustersV ClustersW</InputClusterListNames>
    <ShouldRemoveShowerPfos>true</ShouldRemoveShowerPfos>
    <InputPfoListNames>TrackParticles3D</InputPfoListNames>
</algorithm>
<algorithm type = "LArThreeDShowers">
    <InputClusterListNameU>ClustersU</InputClusterListNameU>
    <InputClusterListNameV>ClustersV</InputClusterListNameV>
    <InputClusterListNameW>ClustersW</InputClusterListNameW>
    <OutputPfoListName>ShowerParticles3D</OutputPfoListName>
    <ShowerTools>
        <tool type = "LArClearShowers"/>
        <tool type = "LArSplitShowers"><NCommonClusters>2</NCommonClusters></tool>
        <tool type = "LArSplitShowers"><NCommonClusters>1</NCommonClusters></tool>
        <tool type = "LArSimpleShowers"/>
    </ShowerTools>
</algorithm>
```

2D Cluster Characterisation

- **Characterise 2D Clusters as track-like or shower-like using topological measures (some use of calorimetric information would also be desirable in the future).**
- Track selection cuts placed on length of the Clusters, measurement of their transverse width and how sparse the Hit distribution is along the Clusters.
- Provide indication of likely Cluster particle id for use in downstream shower reco algorithms.

```
if (this->IsClearTrack(pCluster))
{
    PandoraContentApi::Cluster::Metadata metadata;
    metadata.m_particleId = MU_MINUS;
    PandoraContentApi::AlterMetadata(*this, pCluster, metadata);
}
```



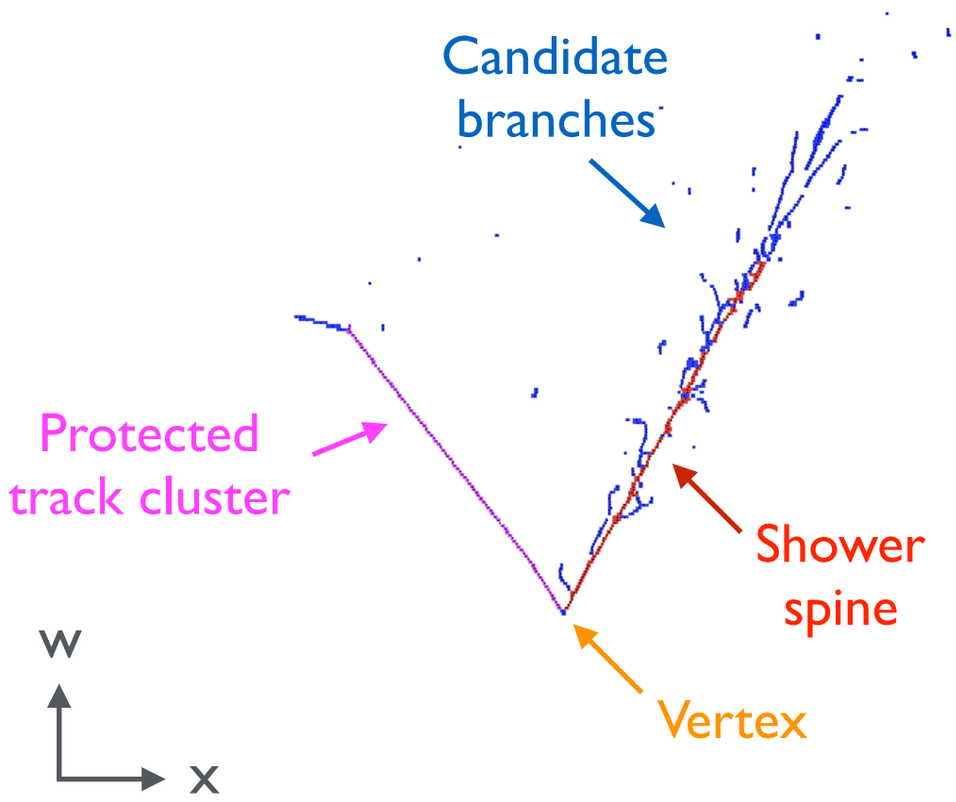


2D Shower Spine Selection

- **Identify 2D Clusters that could represent shower spines. These are typically long, flagged as shower-like and vertex-associated (either near vertex position or point to it).**
- Shower spines could already be part of existing track Particles (which may even group spines for same shower in all views). New alg needed to add branches to existing shower *Particles*.
- Existence in Particle not guaranteed: ability of 2D reco to track along shower length not well-defined. Likely poor common x-overlap; maybe miss in one view; problems with sparse showers.

LArPointingClusterHelper

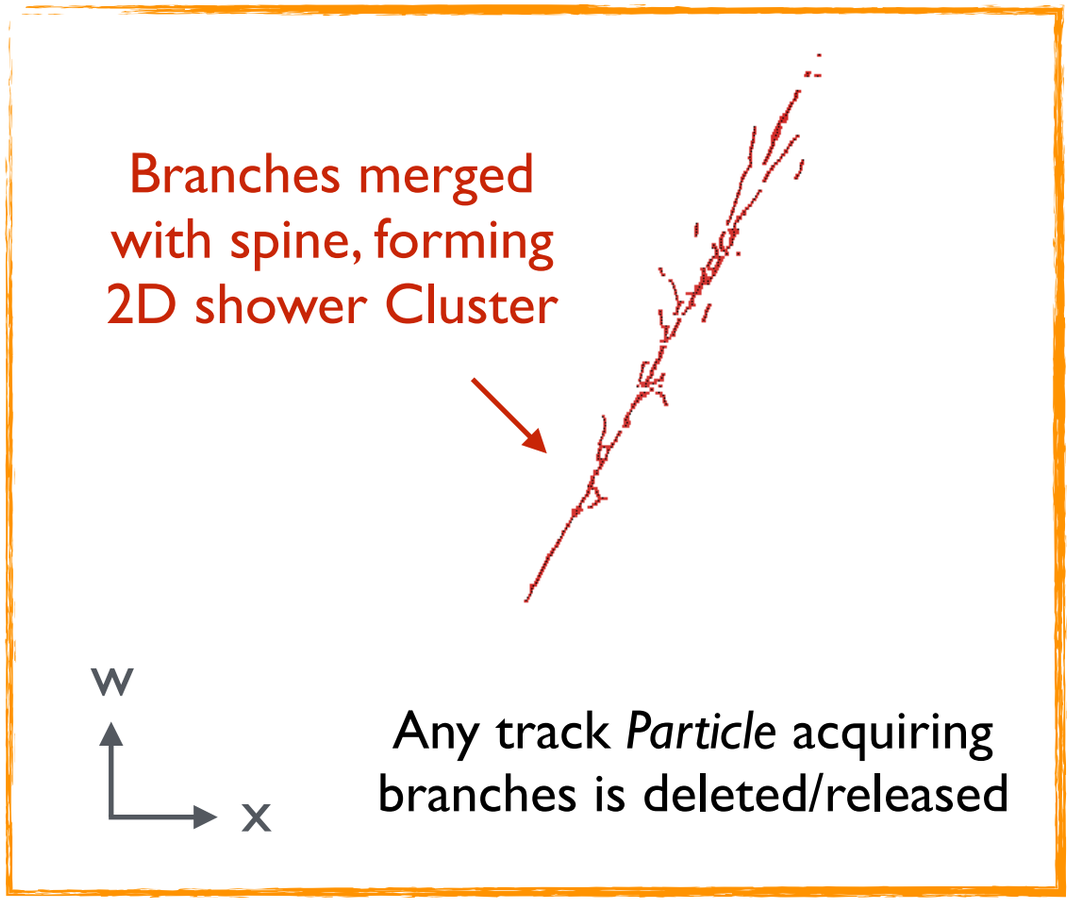
```
/**
 *  @brief  Whether pointing vertex is emitted from a given position
 *
 *  @param  parentVertex the parent vertex position
 *  @param  daughterVertex the daughter pointing vertex
 *  @param  minLongitudinalDistance the min longitudinal distance cut
 *  @param  maxLongitudinalDistance the max longitudinal distance cut
 *  @param  maxTransverseDistance the max transverse distance cut
 *  @param  angularAllowance the pointing angular allowance in degrees
 *
 *  @return boolean
 */
static bool IsEmission(const pandora::CartesianVector &parentVertex,
    const LArPointingCluster::Vertex &daughterVertex,
    const float minLongitudinalDistance,
    const float maxLongitudinalDistance,
    const float maxTransverseDistance,
    const float angularAllowance);
```



2D Shower Branch Growing

- **Add 2D branch Clusters to most appropriate 2D shower spine. Recursive approach, finding branches on spine candidate, then branches on branches, etc.**
- For every branch, record strength of association to each shower spine candidate. Keep track of strengths of "paths of association" between branches and spines.
- Approach allows informed decisions about which branches to add to which shower spines, providing information about the context of the overall event topology.

- **Design pattern: a derived algorithm just needs to provide following implementation:**
  1. **Selection of interesting shower spine and shower branch candidates.**
  2. **Logic dictating "strength" of association between any provided pair of Clusters.**



3D Shower Matching

- **Following 2D shower reconstruction, 2D shower Clusters are hopefully rather complete and are matched between readout planes to form 3D shower Particles.**
- Ideas and base classes developed for 3D track reconstruction are re-used. A ThreeDShowers algorithm builds a tensor to store ShowerOverlapResult objects for Cluster combinations.
- A series of algorithm tools query the tensor, making changes to the 2D Clusters in order to ensure that unambiguous shower Particles can be created.

LArShowerOverlapResult

```
/**
 *  @brief  Constructor
 *
 *  @param  nMatchedSamplingPoints the number of matched sampling points
 *  @param  nSamplingPoints the number of sampling points
 *  @param  xOverlap the x overlap details
 */
ShowerOverlapResult(const unsigned int nMatchedSamplingPoints, const unsigned int nSamplingPoints, const XOverlap &xOverlap);
```

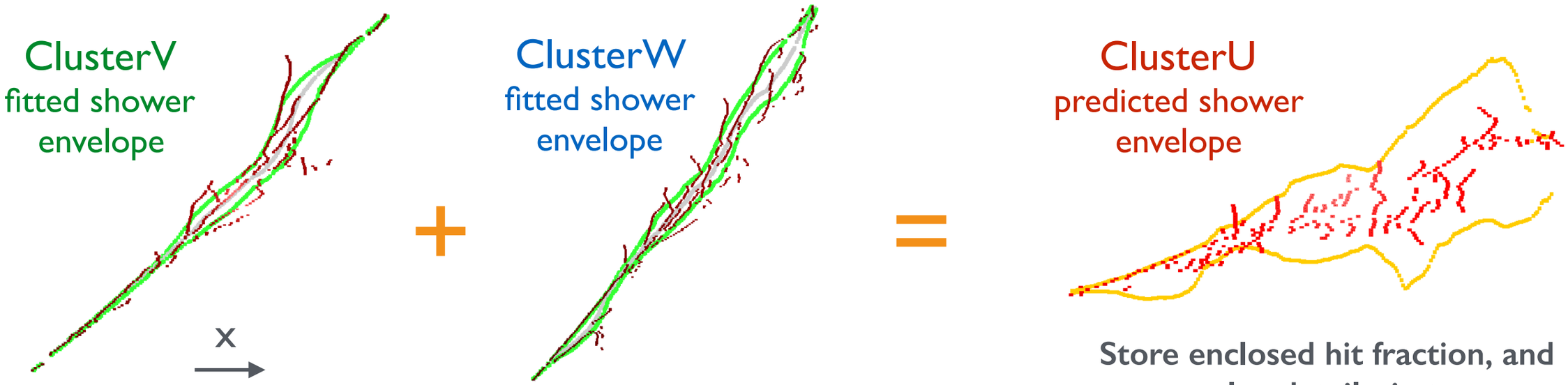
Collect details of 2D shower envelopes

ThreeDShowersAlgorithm

```
ShowerPositionMapPair positionMapsU, positionMapsV, positionMapsW;
this->GetShowerPositionMaps(fitResultU, fitResultV, fitResultW, xSampling, positionMapsU, positionMapsV, positionMapsW);

unsigned int nSampledHitsU(0), nMatchedHitsU(0);
this->GetBestHitOverlapFraction(pClusterU, xSampling, positionMapsU, nSampledHitsU, nMatchedHitsU);

unsigned int nSampledHitsV(0), nMatchedHitsV(0);
this->GetBestHitOverlapFraction(pClusterV, xSampling, positionMapsV, nSampledHitsV, nMatchedHitsV);

unsigned int nSampledHitsW(0), nMatchedHitsW(0);
this->GetBestHitOverlapFraction(pClusterW, xSampling, positionMapsW, nSampledHitsW, nMatchedHitsW);

const unsigned int nMatchedHits(nMatchedHitsU + nMatchedHitsV + nMatchedHitsW);
const unsigned int nSampledHits(nSampledHitsU + nSampledHitsV + nSampledHitsW);
```

ShowerOverlapResult

- **2D sliding shower fit performed to 2D shower Clusters. Consists of three sliding linear fit results: one to all Hits in 2D Cluster and one for fits to each of two "shower edges".**
  - Shower edge fits consider only Hits with extremal transverse coordinates, wrt the shower axis. Provides mechanism for parameterising envelope of the 2D shower Cluster.
  - To calculate ShowerOverlapResult for combination of three Clusters, shower edges from two views (e.g. V, W) are combined to produce shower envelope for third Cluster (e.g. U).
  - Fraction of Hits in the third Cluster contained within the envelope is recorded. All three combinations of Clusters used to evaluate ShowerOverlapResult (also records x-overlap).



ClearShowers Tool

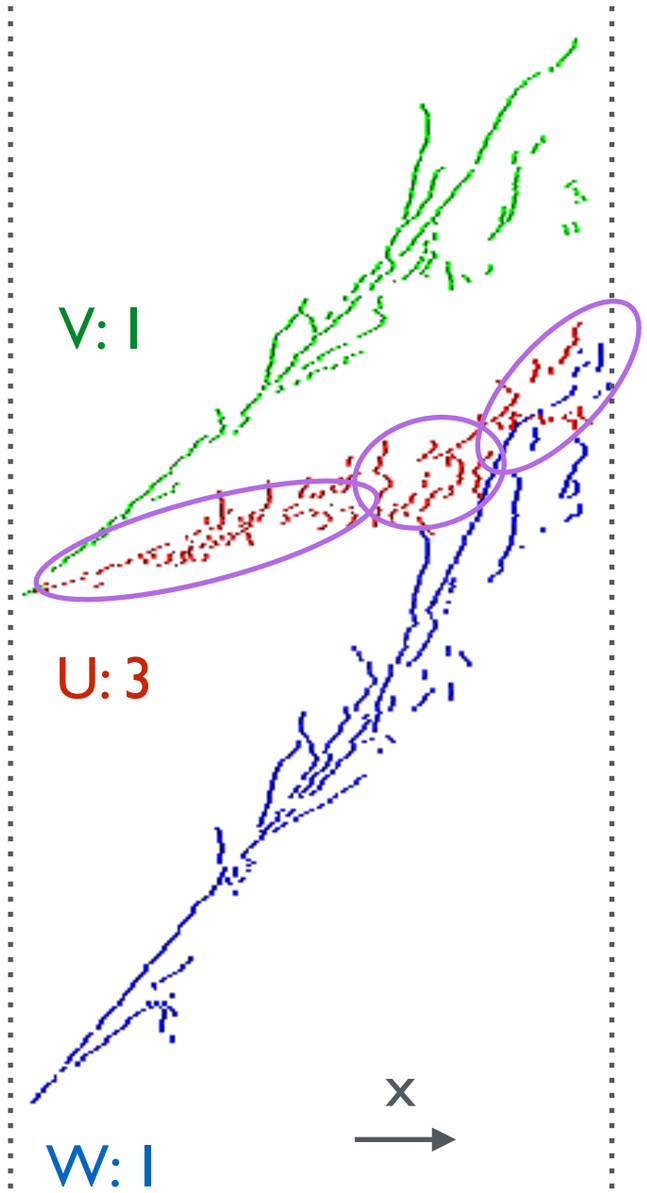
- **First is the ClearShowersTool, which looks to form shower Particles from any unambiguous associations.**
- Rather like the LongTracksTool, it also looks to resolve "obvious" ambiguities, where the best combination is clear.
- Association between Clusters must satisfy quality cuts on common x-overlap and fraction of Hits enclosed in envelopes.
- As with track reco, full list of tools runs again if any tool makes a change to the 2D Clusters or forms a Particle.

e.g. 3:3:1

Small problems cause ambiguities, but the best shower Cluster combination is obvious.

SplitShowers Tool

- **SplitShowerTool looks to address kind of topology shown in example.**
- Three tensor elements share common Clusters in V and W views.
- Flags the three U Clusters as worthy of further investigation.
- In two passes of tool (each merging one pair of Clusters), association between U Clusters examined and Clusters merged.
- Tool can operate in modes where it examines ShowerOverlapResults with either one or two Clusters in common.
- Looks to see if plausible Cluster merges can remove Particle creation ambiguities.
- Further tensor tools foreseen to improve this area, e.g. introducing shower splitting.



Three tensor elements with two Clusters in common (V and W)

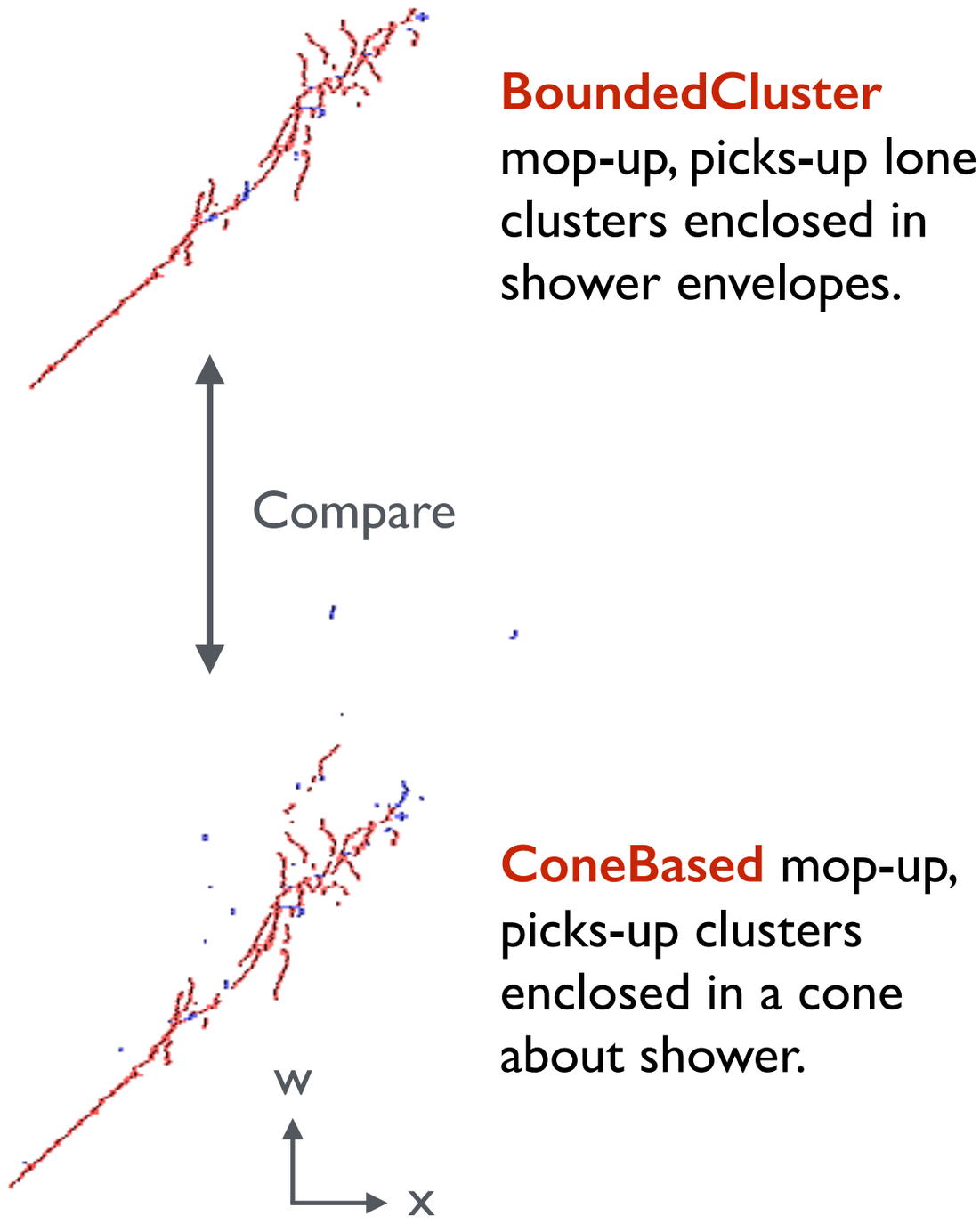
Particle Id

- **Particle identification labels (PDG codes) attached to the final Particles identify:**
  - Particles created by 3D track reconstruction: μ, PDG 13.
  - Particles created by 3D shower reconstruction: e, PDG 11.

- Reassessing particle identification for the final Particles is on the TODO list. One simple approach would be to re-use 2D ClusterCharacterisation-type of approach: 3x2D.
- Current approach relies, in part, on information calculated before any branches were added to shower spines! Number of branches added: important piece of information!

Particle Refinement - 2D

- **Began by forming 2D Clusters, then using topological association algs to improve purity and completeness.**
- Now perform similar operations, to refine initial track and shower Particles.
- For shower Particles, use simple 2D algorithms to pick-up missing objects: small, unassociated 2D Clusters or Hits.
- Use shower envelopes and cone fits to 2D shower Clusters to control merging of lone Clusters with *Clusters in Particles*.



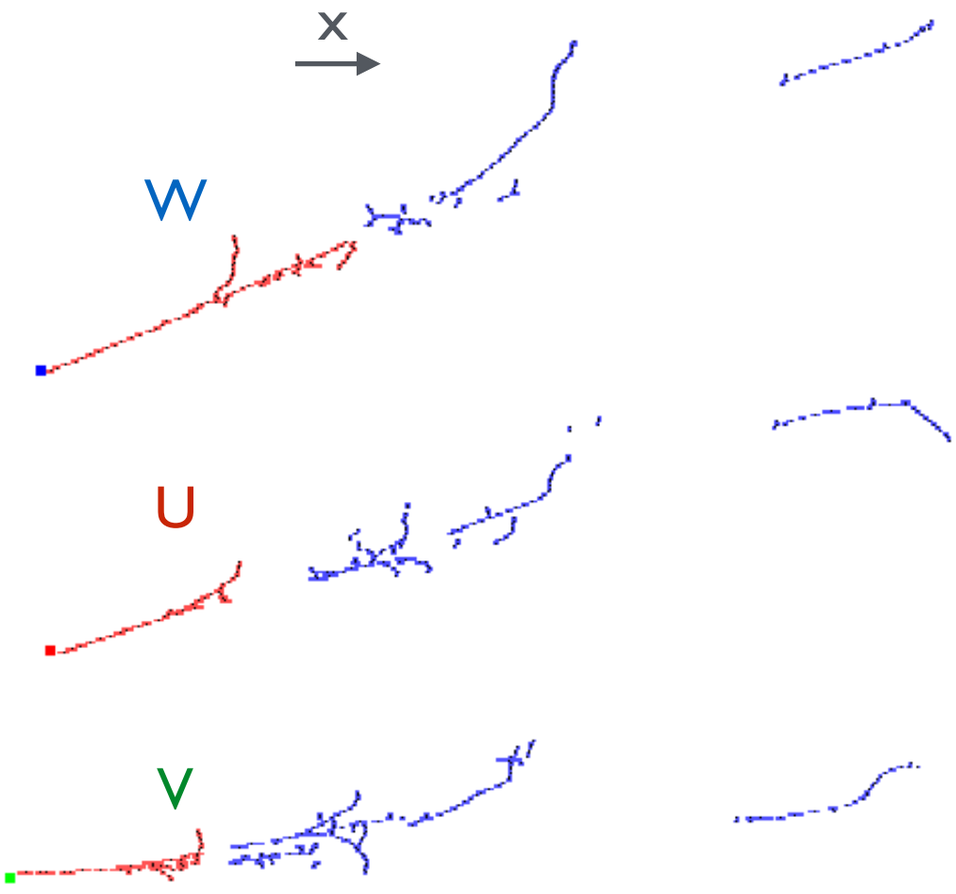
BoundedCluster mop-up, picks-up lone clusters enclosed in shower envelopes.

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

Particle Refinement - 3D

- **Now merge reconstructed *Particles* that look like they represent elements of the same true particle.**
- Important for very sparse showers, where there can be sizeable distances between groupings of Hits.
- Example shows single true shower, split into multiple reconstructed Particles: one track and two showers.
- `VertexBasedMerging` alg works outwards from interaction vertex and iteratively picks-up downstream Particles.
- Uses cone fits to 2D Clusters, but looks for evidence of association in all views. Continues until all possible merges made.

VertexBasedPFO mop-up, important for sparse showers.



Vertex Particle (shower) | **Daughter Particle 1 (shower)** | **Daughter Particle 2 (track)**

Particle Hierarchy

Final Particles are organised into a hierarchy, which assumes the input 2D Hits in slice represent a neutrino interaction. Use the newly-created 3D Hits/Clusters:

1. Create a neutrino Particle and attach the interaction vertex.
2. Add primary daughters: look for evidence of association between 3D Clusters in track and shower Particles and the interaction vertex (nearby, pointing?).
3. Add subsequent daughter Particles to existing primary daughters of neutrino e.g. add decay electron to parent primary muon.
4. Set particle id for neutrino, based on whether daughter Particle with most 2D Hits is a track or a shower: label as ν_μ or ν_e respectively.
5. Provide 3D vertex positions for each Particle in hierarchy: points of closest approach to parent Particles, or to interaction vertex (if primary).

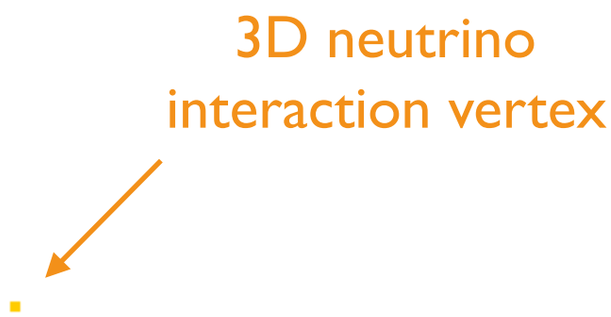
Particle Hierarchy

An example event output (arbitrarily, a rather high-energy ν_e) 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



Particle Hierarchy

An example event output (arbitrarily, a rather high-energy ν_e) is as shown below:

5 GeV ν_e CC: Display 2/4

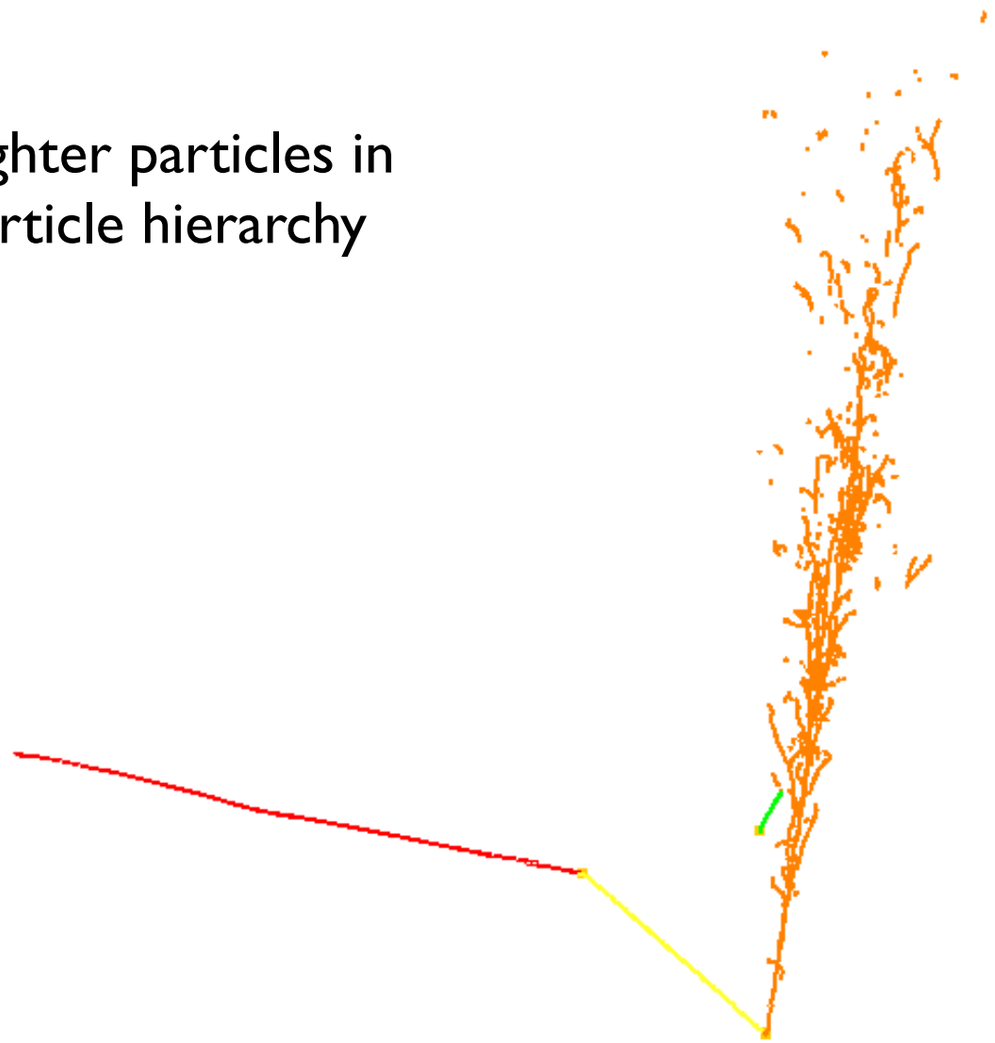
+ Primary daughter particles of the neutrino, each of which has:
  - Particle metadata
  - A list of 2D clusters and a 3D cluster
  - A 3D interaction vertex
  - A list of any further daughter particles

Particle Hierarchy

An example event output (arbitrarily, a rather high-energy ν_e) is as shown below:

5 GeV ν_e CC: Display 3/4

+ Complete list of daughter particles in the reconstructed particle hierarchy

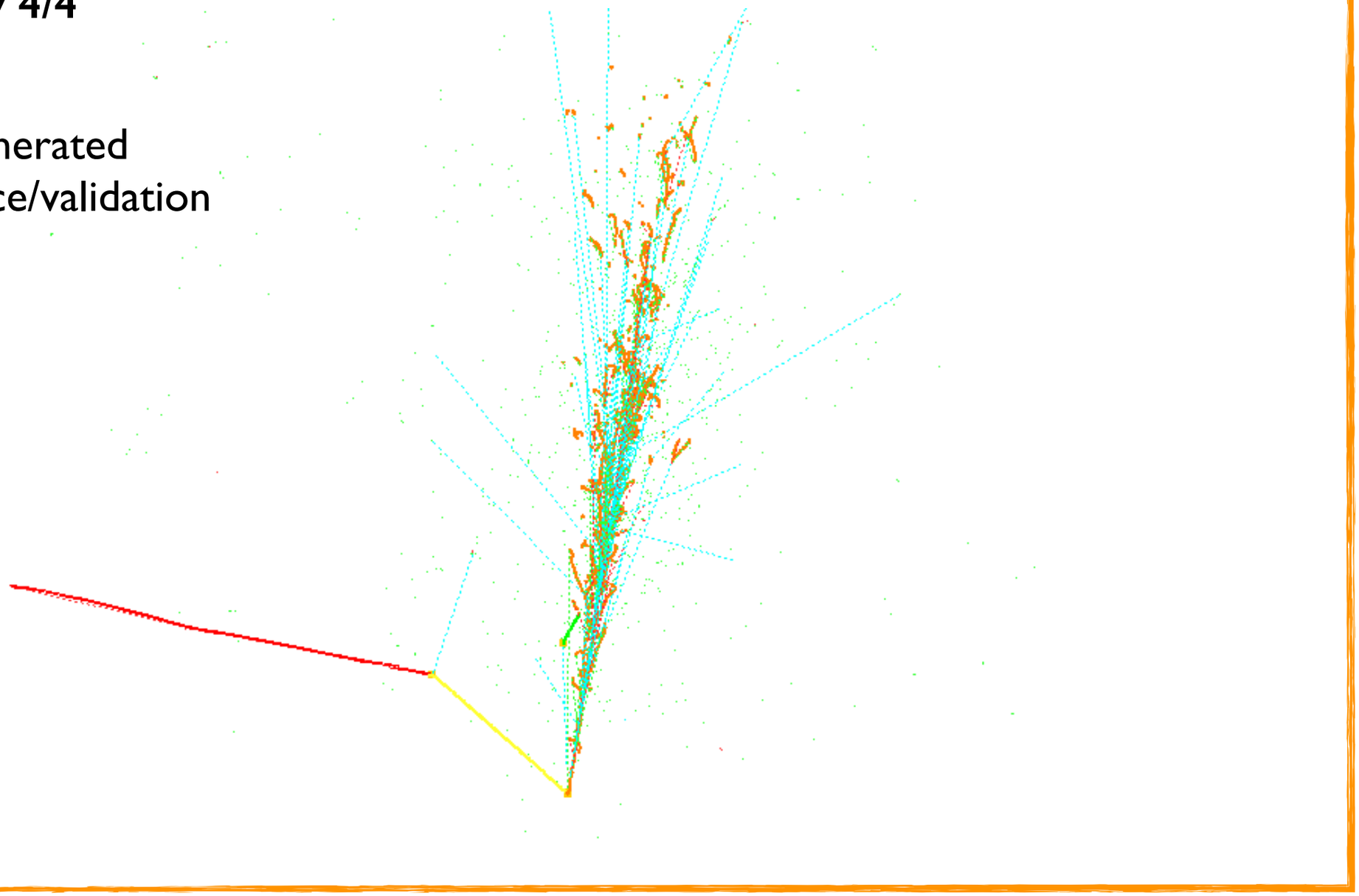


Particle Hierarchy

Each input slice results in one neutrino Particle, which is extracted by the client app and translated to the LArSoft EDM for downstream processing - Talk 8, Handling Outputs.

5 GeV ν_e CC: Display 4/4

+ Overlay details of generated particles, for reference/validation



Questions?