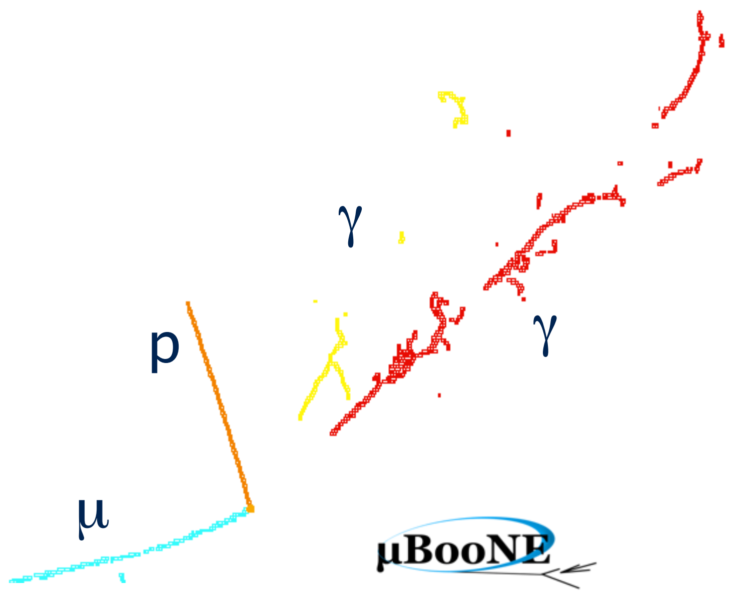


Pandora Talk 5: 3D Track Reconstruction

J. S. Marshall for the Pandora Team

MicroBooNE Pandora Workshop

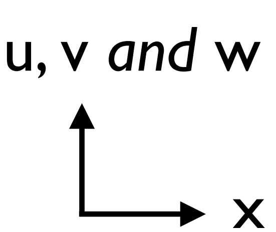
July 11-14th 2016, Cambridge

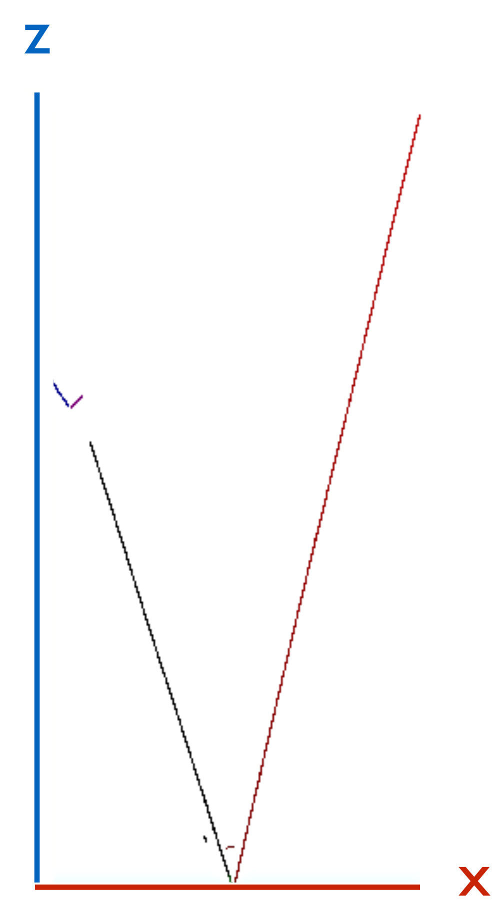
3D Track Reconstruction

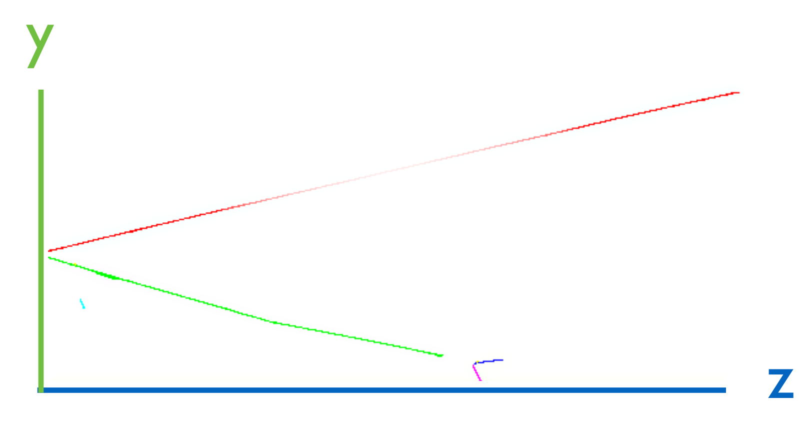
- **The main aim of the 3D track reconstruction is to identify three consistent, track-like Clusters (one from each readout plane) and group them together in a Particle.**
- If there are inconsistencies between the Clusters in the different views, algorithms can make iterative corrections to the 2D Clustering in order to allow unambiguous Particles to emerge.
- For each input 2D Hit in a Particle, a new 3D Hit (or "SpacePoint") can be created.

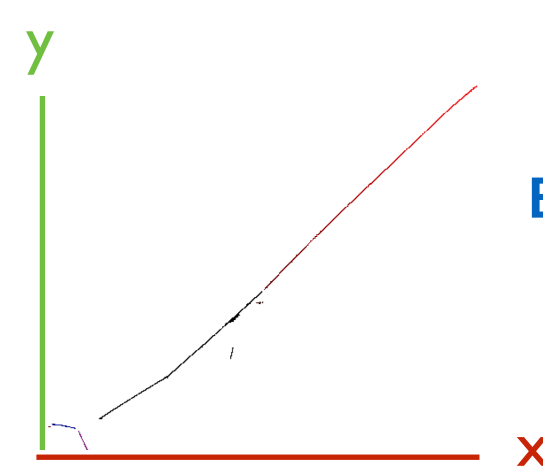
Overlay U,V,W Clusters:

Look for common features, exploiting common x coordinate









Build Particles and 3D Hits

Approach

- **Approach is for an algorithm to compare all permutations of 2D Clusters from the different readout planes and store the results in a rank-three tensor.**
- The three tensor indices are the Clusters in the U,V and W views and, for each combination, the value held in the tensor is a detailed record of the compatibility of the three Clusters.
- Tensor stores information for all the different Cluster combinations and provides a way for algorithms to understand the ambiguities/connections between different Cluster combinations.

Tensor | 3D Base Alg | Derived Algs | Tensor Tools

- A base class provides much of the functionality required to manage and query the tensor, whilst derived algorithms can provide different types of **OverlapResult** to store in the tensor.
- The tensor is examined by **AlgorithmTools** which identify ambiguities and request changes to the 2D Clusters until the tensor is diagonal and the correct combinations are unambiguous.

OverlapTensor

LArOverlapTensor

```
/**
 *  @brief  Set overlap result
 *
 *  @param  pClusterU address of cluster u
 *  @param  pClusterV address of cluster v
 *  @param  pClusterW address of cluster w
 *  @param  overlapResult the overlap result
 */
void SetOverlapResult(const pandora::Cluster *const pClusterU, const pandora::Cluster *const pClusterV,
    const pandora::Cluster *const pClusterW, const OverlapResult &overlapResult);

/**
 *  @brief  Replace an existing overlap result
 *
 *  @param  pClusterU address of cluster u
 *  @param  pClusterV address of cluster v
 *  @param  pClusterW address of cluster w
 *  @param  overlapResult the overlap result
 */
void ReplaceOverlapResult(const pandora::Cluster *const pClusterU, const pandora::Cluster *const pClusterV,
    const pandora::Cluster *const pClusterW, const OverlapResult &overlapResult);

/**
 *  @brief  Remove entries from tensor corresponding to specified cluster
 *
 *  @param  pCluster address of the cluster
 */
void RemoveCluster(const pandora::Cluster *const pCluster);
```

Interface for use by algs filling the tensor

Tensor data-store and navigation

```
typedef std::unordered_map<const pandora::Cluster*, pandora::ClusterList> ClusterNavigationMap;
typedef std::unordered_map<const pandora::Cluster*, OverlapResult> OverlapList;
typedef std::unordered_map<const pandora::Cluster*, OverlapList> OverlapMatrix;
typedef std::unordered_map<const pandora::Cluster*, OverlapMatrix> TheTensor;

TheTensor               m_overlapTensor;              ///< The overlap tensor
ClusterNavigationMap    m_clusterNavigationMapUV;     ///< The cluster navigation map U->V
ClusterNavigationMap    m_clusterNavigationMapVW;     ///< The cluster navigation map V->W
ClusterNavigationMap    m_clusterNavigationMapWU;     ///< The cluster navigation map W->U
```

Overlap Tensor

LArOverlapTensor

```
/**
 *  @brief  Get unambiguous elements
 *
 *  @param  ignoreUnavailable whether to ignore unavailable clusters
 *  @param  elementList to receive the unambiguous element list
 */
void GetUnambiguousElements(const bool ignoreUnavailable, ElementList &elementList) const;

/**
 *  @brief  Get the number of connections for a specified cluster
 *
 *  @param  pCluster address of a cluster
 *  @param  ignoreUnavailable whether to ignore unavailable clusters
 *  @param  nU to receive the number of u connections
 *  @param  nV to receive the number of v connections
 *  @param  nW to receive the number of w connections
 */
void GetNConnections(const pandora::Cluster *const pCluster, const bool ignoreUnavailable, unsigned int &nU, unsigned int &nV,
    unsigned int &nW) const;

/**
 *  @brief  Get a list of elements connected to a specified cluster
 *
 *  @param  pCluster address of a cluster
 *  @param  ignoreUnavailable whether to ignore unavailable clusters
 *  @param  elementList to receive the connected element list
 */
void GetConnectedElements(const pandora::Cluster *const pCluster, const bool ignoreUnavailable, ElementList &elementList) const;
```

Aim of tensor is to cleanly present algs/tools with key matching information they need

LArOverlapTensor::Element

```
const pandora::Cluster *m_pClusterU;           ///< The address of the u cluster
const pandora::Cluster *m_pClusterV;           ///< The address of the v cluster
const pandora::Cluster *m_pClusterW;           ///< The address of the w cluster
OverlapResult           m_overlapResult;       ///< The overlap result
```

Tensor stores OverlapResult for each combination of U, V and W Clusters. Crucially, it also helps algorithms to understand the connections/ambiguities between multiple Clusters.

OverlapResult

- **The OverlapResult stored in the tensor is simply a cache of information that may be useful when deciding how best to match Clusters between views.**
- TransverseOverlapResult records details of Cluster x-overlap, the number of sampling points used to assess Cluster consistency, the number of matched sampling points and a χ^2 value.
- The tensor is examined by a series of algorithm tools, which can request Particle creation or request changes to the 2D pattern recognition in order to address matching ambiguities.

TransverseOverlapResult

```
/**
 *  @brief  Constructor
 *
 *  @param  nMatchedSamplingPoints the number of matched sampling points
 *  @param  nSamplingPoints the number of sampling points
 *  @param  chi2 the chi squared value
 *  @param  xOverlap the x (common-coordinate) overlap details
 */
TransverseOverlapResult(const unsigned int nMatchedSamplingPoints, const unsigned int nSamplingPoints, const float chi2,
    const XOverlap &xOverlap);
```

XOverlap

```
/**
 *  @brief  Constructor
 *
 *  @param  uMinX min x value in the u view
 *  @param  uMaxX max x value in the u view
 *  @param  vMinX min x value in the v view
 *  @param  vMaxX max x value in the v view
 *  @param  wMinX min x value in the w view
 *  @param  wMaxX max x value in the w view
 *  @param  xOverlapSpan the x overlap span
 */
XOverlap(const float uMinX, const float uMaxX, const float vMinX, const float vMaxX, const float wMinX, const float wMaxX,
    const float xOverlapSpan);
```

ThreeDBase Alg

```
/**
 *  @brief  Select a subset of input clusters for processing in this algorithm
 *
 *  @param  pInputClusterList address of an input cluster list
 *  @param  selectedClusterList to receive the selected cluster list
 */
virtual void SelectInputClusters(const pandora::ClusterList *const pInputClusterList, pandora::ClusterList &selectedClusterList) const = 0;

/**
 *  @brief  Calculate cluster overlap result and store in tensor
 *
 *  @param  pClusterU address of U view cluster
 *  @param  pClusterV address of V view cluster
 *  @param  pClusterW address of W view cluster
 */
virtual void CalculateOverlapResult(const pandora::Cluster *const pClusterU, const pandora::Cluster *const pClusterV,
    const pandora::Cluster *const pClusterW) = 0;

/**
 *  @brief  Examine contents of tensor, collect together best-matching 2D particles and modify clusters as required
 */
virtual void ExamineTensor() = 0;


/**
 *  @brief  Perform any preparatory steps required, e.g. caching expensive fit results for clusters
 */
virtual void PreparationStep();
```

ThreeDBaseAlgorithm

Owns OverlapTensor containing OverlapResults of a specific type.

A derived alg must calculate the OverlapResults and examine the tensor.

```
const pandora::ClusterList *m_pInputClusterListU;    ///< Address of the input cluster list U
const pandora::ClusterList *m_pInputClusterListV;    ///< Address of the input cluster list V
const pandora::ClusterList *m_pInputClusterListW;    ///< Address of the input cluster list W

pandora::ClusterList        m_clusterListU;          ///< The selected modified cluster list U
pandora::ClusterList        m_clusterListV;          ///< The selected modified cluster list V
pandora::ClusterList        m_clusterListW;          ///< The selected modified cluster list W

OverlapTensor<T>            m_overlapTensor;         ///< The overlap tensor
```

ThreeDBase Alg

```
/**
 *  @brief  Create particles using findings from recent algorithm processing
 *
 *  @param  protoParticleVector the proto particle vector
 *  @return whether particles were created
 */
virtual bool CreateThreeDParticles(const ProtoParticleVector &protoParticleVector);

/**
 *  @brief  Merge clusters together
 *
 *  @param  clusterMergeMap the cluster merge map
 *  @return whether changes to the tensor have been made
 */
virtual bool MakeClusterMerges(const ClusterMergeMap &clusterMergeMap);

/**
 *  @brief  Update to reflect a cluster merge
 *
 *  @param  pEnlargedCluster address of the enlarged cluster
 *  @param  pDeletedCluster address of the deleted cluster
 */
virtual void UpdateUponMerge(const pandora::Cluster *const pEnlargedCluster, const pandora::Cluster *const pDeletedCluster);

/**
 *  @brief  Update to reflect a cluster split
 *
 *  @param  pSplitCluster1 address of the first cluster fragment
 *  @param  pSplitCluster2 address of the second cluster fragment
 *  @param  pDeletedCluster address of the deleted cluster
 */
virtual void UpdateUponSplit(const pandora::Cluster *const pSplitCluster1, const pandora::Cluster *const pSplitCluster2,
    const pandora::Cluster *const pDeletedCluster);

/**
 *  @brief  Update to reflect addition of a new cluster to the problem space
 *
 *  @param  pNewCluster address of the new cluster
 */
virtual void UpdateForNewCluster(const pandora::Cluster *const pNewCluster);

/**
 *  @brief  Update to reflect cluster deletion
 *
 *  @param  pDeletedCluster address of the deleted cluster
 */
virtual void UpdateUponDeletion(const pandora::Cluster *const pDeletedCluster);
```

Controls common data-management operations:

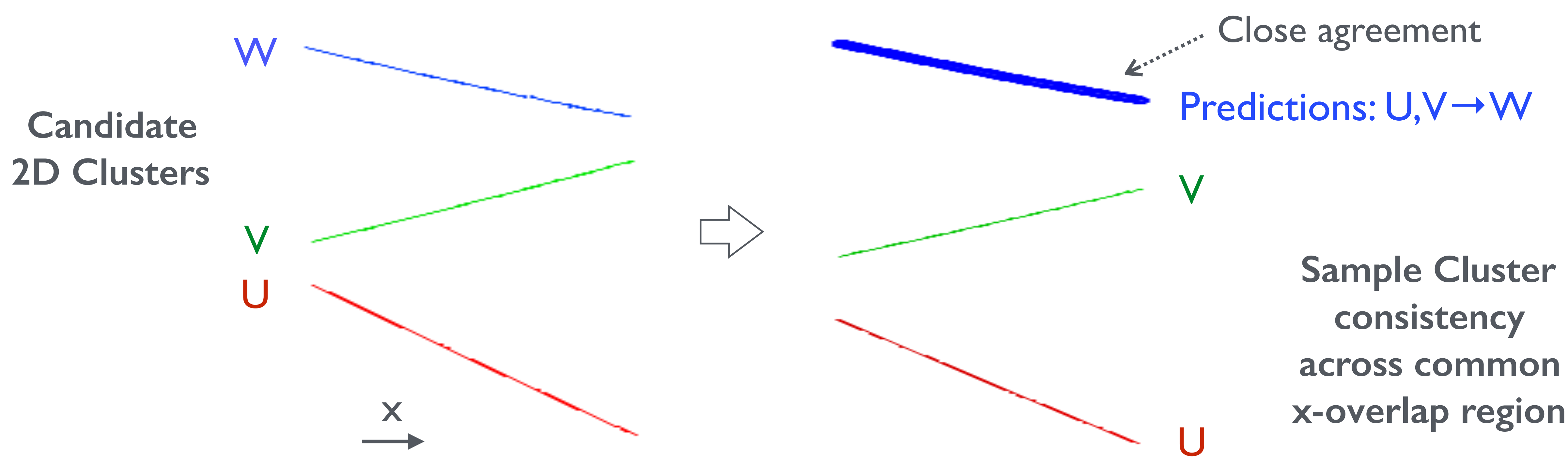
Can create Particles, split or merge Clusters and feed information back into tensor.

ThreeDBaseAlgorithm

ThreeDTransverseTracks Alg

```
class ThreeDTransverseTracksAlgorithm : public ThreeDTracksBaseAlgorithm<TransverseOverlapResult>
```

- **Select 2D Clusters (length cuts, etc.), compare all combinations between views and calculate OverlapResult tailored to 'transverse' tracks, i.e. those with notable x-extent:**
  - For given x-coordinate, obtain sliding linear fit positions for 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



ThreeDTransverseTracks Alg

Define x sampling point in overlap region

Use sliding linear fits to extract fit positions and directions at sampling point

```
float pseudoChi2Sum(0.f);
unsigned int nSamplingPoints(0), nMatchedSamplingPoints(0);

for (unsigned int n = 0; n <= nPoints; ++n)
{
    const float x(minX + (maxX - minX) * static_cast<float>(n) / static_cast<float>(nPoints));

    CartesianVector fitUVector(0.f, 0.f, 0.f), fitVVector(0.f, 0.f, 0.f), fitWVector(0.f, 0.f, 0.f);
    CartesianVector fitUDirection(0.f, 0.f, 0.f), fitVDirection(0.f, 0.f, 0.f), fitWDirection(0.f, 0.f, 0.f);

    if ((STATUS_CODE_SUCCESS != slidingFitResultU.GetTransverseProjection(x, fitSegmentU, fitUVector, fitUDirection)) ||
        (STATUS_CODE_SUCCESS != slidingFitResultV.GetTransverseProjection(x, fitSegmentV, fitVVector, fitVDirection)) ||
        (STATUS_CODE_SUCCESS != slidingFitResultW.GetTransverseProjection(x, fitSegmentW, fitWVector, fitWDirection)))
    {
        continue;
    }

    const float u(fitUVector.GetZ()), v(fitVVector.GetZ()), w(fitWVector.GetZ());
    const float uv2w(LArGeometryHelper::MergeTwoPositions(this->GetPandora(), TPC_VIEW_U, TPC_VIEW_V, u, v));
    const float uw2v(LArGeometryHelper::MergeTwoPositions(this->GetPandora(), TPC_VIEW_U, TPC_VIEW_W, u, w));
    const float vw2u(LArGeometryHelper::MergeTwoPositions(this->GetPandora(), TPC_VIEW_V, TPC_VIEW_W, v, w));

    const float deltaU((vw2u - u) * fitUDirection.GetX());
    const float deltaV((uw2v - v) * fitVDirection.GetX());
    const float deltaW((uv2w - w) * fitWDirection.GetX());

    const float pseudoChi2(deltaW * deltaW + deltaV * deltaV + deltaU * deltaU);
    pseudoChi2Sum += pseudoChi2;
    ++nSamplingPoints;

    if (pseudoChi2 < m_pseudoChi2Cut)
        ++nMatchedSamplingPoints;
}
```

Make predictions: U,V→W; VW→U; UW→V

Count matched sampling points and calculate χ^2

ThreeDTransverseTracksAlgorithm

TransverseTensor Tools

- **ThreeDTransverseTracksAlgorithm defines interface for its TransverseTensor tools:**
- Provides tools with Algorithm address to enable access to its cluster merging/splitting and tensor updating functionality. Also provides tools with direct access to the tensor.
- Algorithm owns an ordered list of TransverseTensorTools, which is populated according to XML configuration. These tools will be used to examine/process the tensor each event.

ThreeDTransverseTracksAlgorithm

```
/**
 *  @brief  TransverseTensorTool class
 */
class TransverseTensorTool : public pandora::AlgorithmTool
{
public:
    typedef ThreeDTransverseTracksAlgorithm::TensorType TensorType;
    typedef std::vector<TensorType::ElementList::const_iterator> IteratorList;

    /**
     *  @brief  Run the algorithm tool
     *
     *  @param  pAlgorithm address of the calling algorithm
     *  @param  overlapTensor the overlap tensor
     *
     *  @return whether changes have been made by the tool
     */
    virtual bool Run(ThreeDTransverseTracksAlgorithm *const pAlgorithm, TensorType &overlapTensor) = 0;
};



typedef std::vector<TransverseTensorTool*> TensorToolList;
TensorToolList              m_algorithmToolList;        ///< The algorithm tool list
```

TransverseTensor Tools

- **TransverseTensorTools have an XML-defined ordering:**
- If tool makes a change to the tensor, by creating a new Particle or modifying the 2D Clusters, the full list of tools runs again, repeating from the first tool. Run until no further changes.
- Promotes an approach where first tool makes Particles for unambiguous Cluster matches and later tools make 2D Cluster changes to remove ambiguities.

ThreeDTransverseTracksAlgorithm

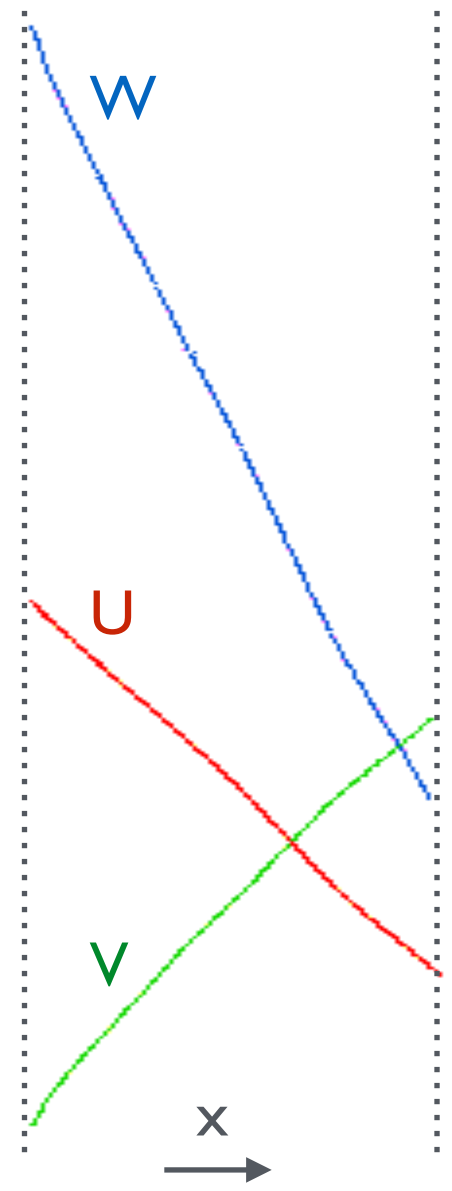
```
void ThreeDTransverseTracksAlgorithm::ExamineTensor()
{
    unsigned int repeatCounter(0);

    for (TensorToolList::const_iterator iter = m_algorithmToolList.begin(),
         iterEnd = m_algorithmToolList.end(); iter != iterEnd; )
    {
        if ((*iter)->Run(this, m_overlapTensor))
        {
            iter = m_algorithmToolList.begin();

            if (++repeatCounter > m_nMaxTensorToolRepeats)
                break;
        }
        else
        {
            ++iter;
        }
    }
}
```

```
<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>
```

XML

ClearTracks Tool

- **The first tool looks to directly build Particles from unambiguous groupings of three Clusters.**
- Examine tensor to find regions where only three Clusters are connected; one from each of U,V and W views.
- Quality cuts are applied to the TransverseOverlapResult and, if passed, a new Particle is created.
- The common x-overlap must be >90% of the x-span for all Clusters at this stage in the processing.



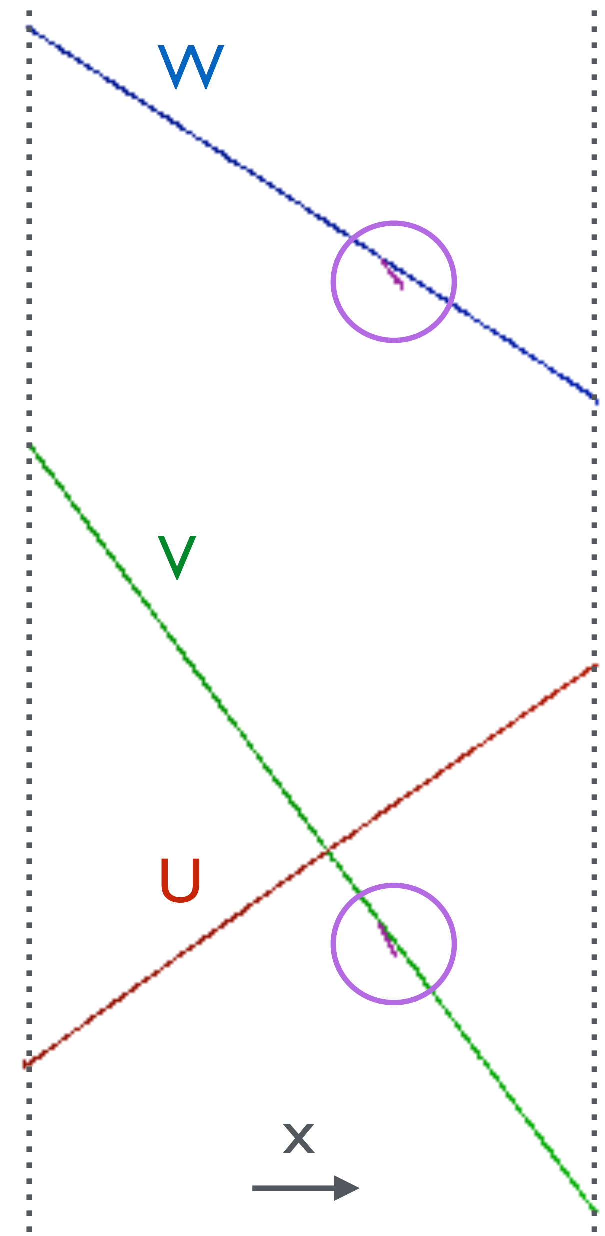
1:1:1

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.

LongTracks Tool

- **The LongTracks tool aims to address any ambiguities in the tensor that have an obvious resolution.**
- Example shown has two small delta-rays near a long cosmic-ray track.
- Clusters are matched in multiple configurations; tensor is not diagonal.
- One of TransverseOverlapResults is, however, significantly better than others.
- Tensor element shows better x-overlap and more matched sampling points.
- Decision is to create a Particle representing long cosmic-muon track.
- Delta-rays can then be associated with cosmic-ray Particle at a later stage.



e.g. 1:2:2

Ringed clusters in V and W views also match U Cluster, so U Cluster ambiguous

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

Implementation: LongTracks Tool

```
ProtoParticleVector protoParticleVector;
ClusterList usedClusters;

ClusterVector sortedKeyClusters;
overlapTensor.GetSortedKeyClusters(sortedKeyClusters);

for (const Cluster *const pKeyCluster : sortedKeyClusters)
{
    if (!pKeyCluster->IsAvailable())
        continue;

    TensorType::ElementList elementList;
    overlapTensor.GetConnectedElements(pKeyCluster, true, elementList);

    IteratorList iteratorList;
    this->SelectLongElements(elementList, usedClusters, iteratorList);

    // Check that elements are significantly longer than any directly connected elements
    for (IteratorList::const_iterator iIter = iteratorList.begin(), iIterEnd = iteratorList.end(); iIter != iIterEnd; ++iIter)
    {
        if (LongTracksTool::HasLongDirectConnections(iIter, iteratorList))
            continue;

        if (!LongTracksTool::IsLongerThanDirectConnections(iIter, elementList, m_minMatchedSamplingPointRatio, usedClusters))
            continue;

        ProtoParticle protoParticle;
        protoParticle.m_clusterListU.insert((*iIter)->GetClusterU());
        protoParticle.m_clusterListV.insert((*iIter)->GetClusterV());
        protoParticle.m_clusterListW.insert((*iIter)->GetClusterW());
        protoParticleVector.push_back(protoParticle);

        usedClusters.insert((*iIter)->GetClusterU());
        usedClusters.insert((*iIter)->GetClusterV());
        usedClusters.insert((*iIter)->GetClusterW());
    }
}

return pAlgorithm->CreateThreeDParticles(protoParticleVector);
```

Get connected elements from tensor

Select subset with long Clusters and good overlap

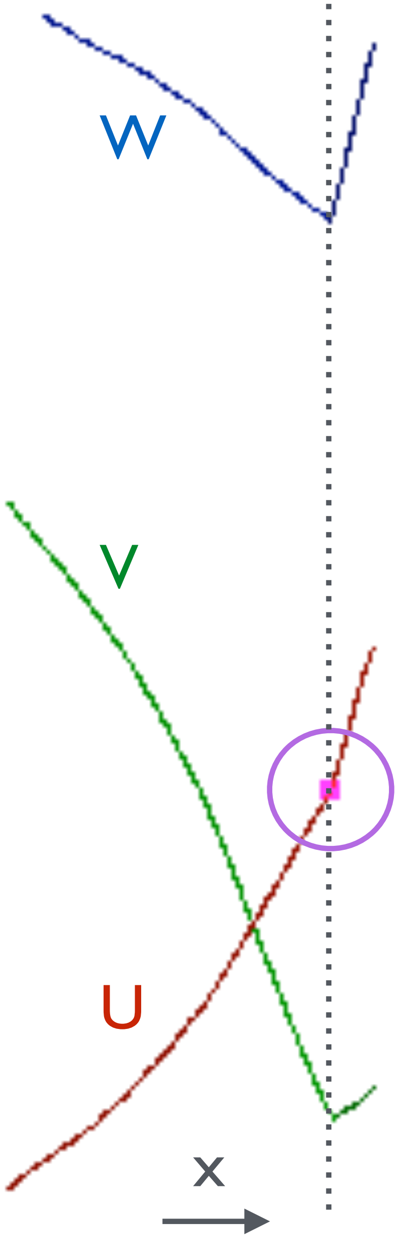
Ensure that selected element is better than alternatives

Specify Particle details and monitor Cluster usage in tool

Ask alg to make all Particles found

OvershootTracks Tool

- **The OvershootTracksTool examines the tensor to find Cluster matching ambiguities of the form e.g. 1:2:2**
- Two Clusters in V view and two Clusters in W view connect at common x.
- Single common Cluster in U view, which spans full x-extent of the Clusters.
- Use all connected Clusters to assess whether this is a true 3D kink topology.
- If kink is identified, split U Cluster at relevant x coordinate and feed two new U Clusters back into tensor.
- Initial ClearTracks tool then able to identify two unambiguous groupings of three Clusters and form two Particles.



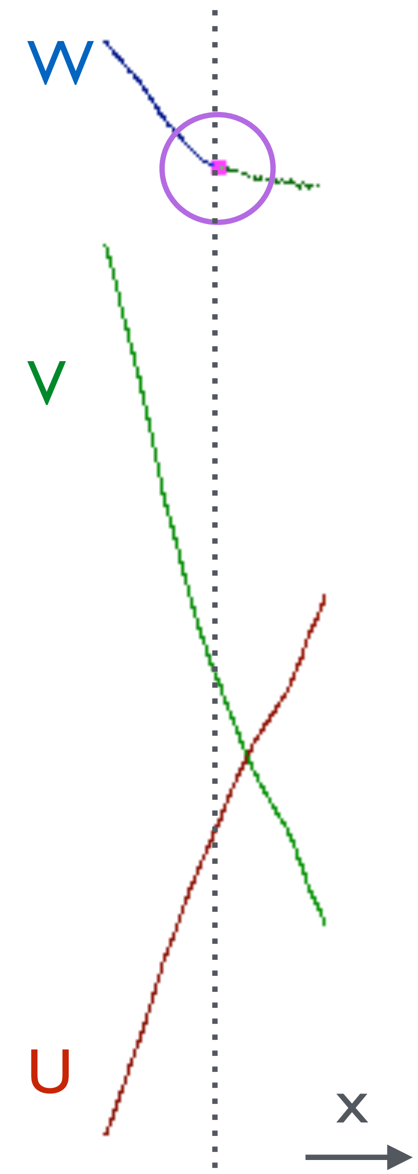
1:2:2

Two clusters in W and V views, matched to a common cluster in U view. Two tensor elements.

Identify whether this is a true 3D kink. If so, split U cluster at relevant position and feed back into tensor (diagonalise).

UndershootTracks Tool

- **The UndershootTracksTool examines the tensor to find Cluster matching ambiguities of the form e.g. 1:1:2**
- Two Clusters in W view matched to common Clusters in the U and V views, leading to conflicting tensor elements.
- Examine connected Clusters to assess whether this is a 3D kink topology (impl. shared with OvershootTracksTool).
- If a 3D kink is not found, the two W Clusters can be merged and a single W Cluster fed back into the tensor.
- Single new Particle can then be created by the ClearTracksTool.



1:1:2

Two clusters in W view, matched to common clusters in U and V views. Two tensor elements.

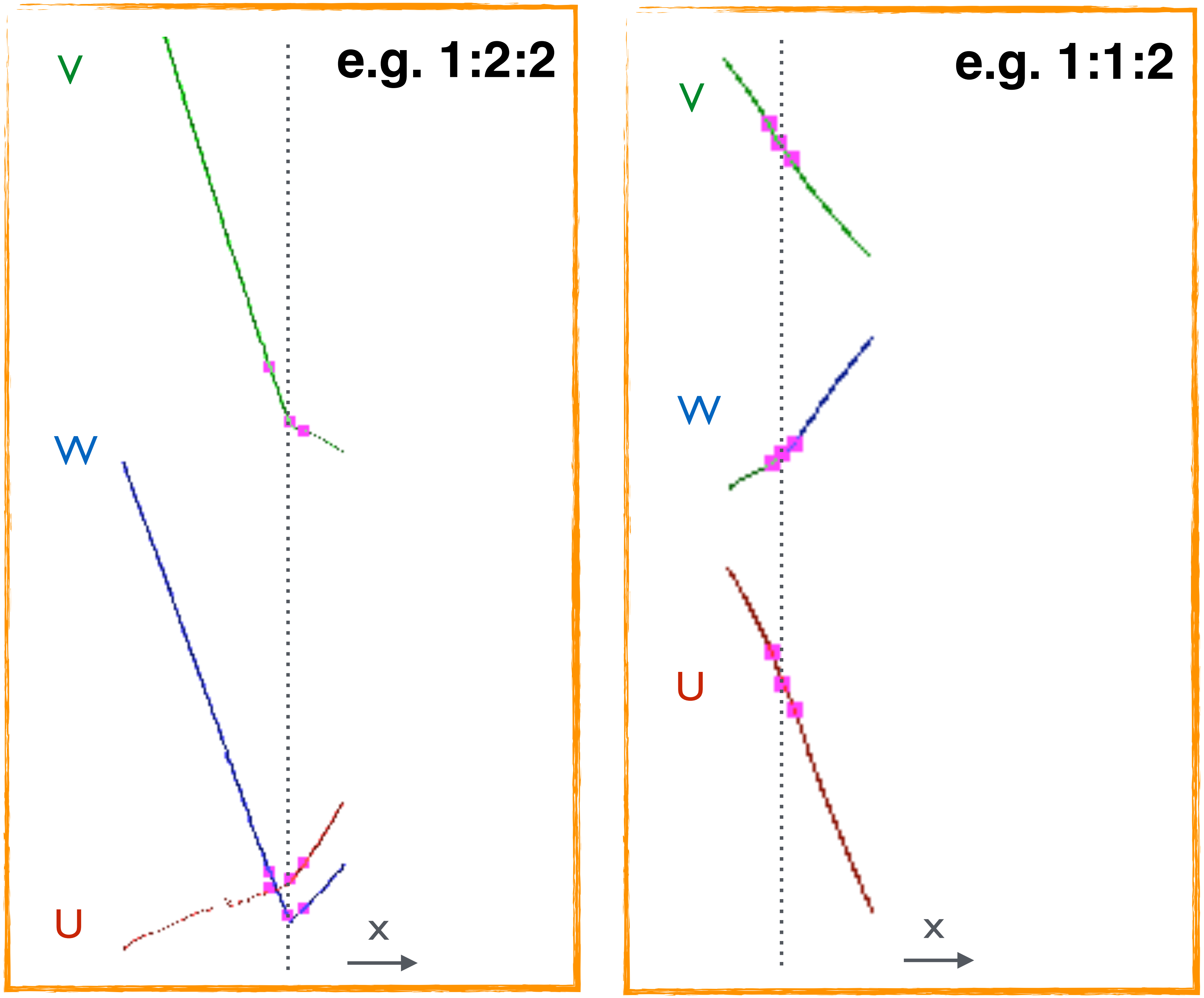
Find that this isn't truly a kink in 3D, so merge the clusters in the W view and feed back into tensor.

To first order (in 2D reco mistakes), should always:

- Split single Cluster for e.g. 1:2:2 configs.
- Merge pair of Clusters for e.g. 1:1:2 configs.

3D kink finding helps to cover second order cases.

Examine 3D directions either side of feature point.



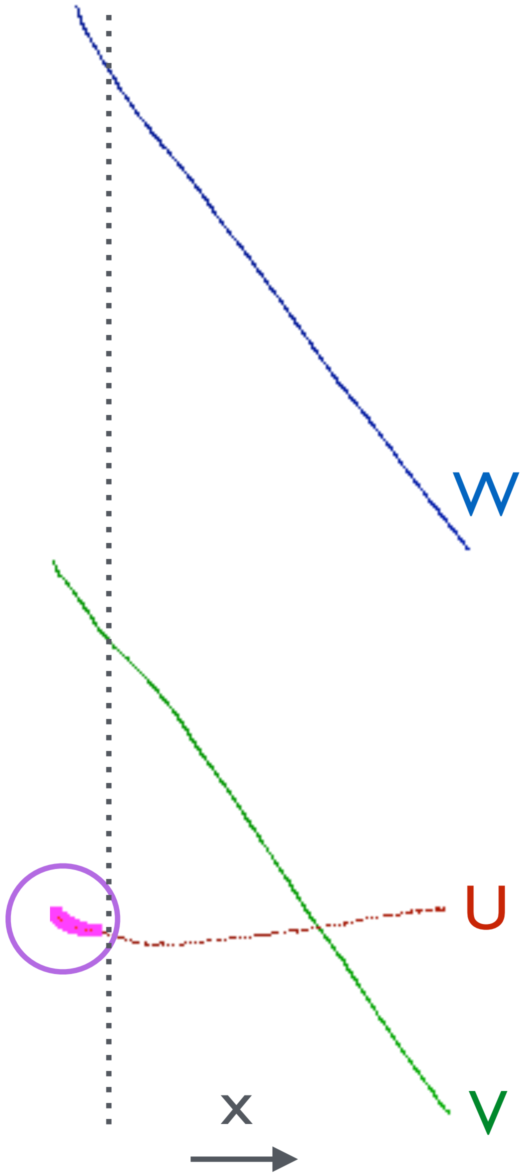
Truly a kink: split merged clusters

Not a kink: merge split clusters

MissingTrackSegment Tool

- **The MissingTrackSegmentTool tries to address discrepancies between Cluster x-overlap.**
- Uses sliding fit results from two long Clusters to predict the continued track position in the short Cluster view.
- Can add available small Clusters to the end of the short Cluster to address the discrepancy.
- Cluster combinations may then satisfy selection requirements of ClearTracks tool, which can create a Particle.

1:1:1



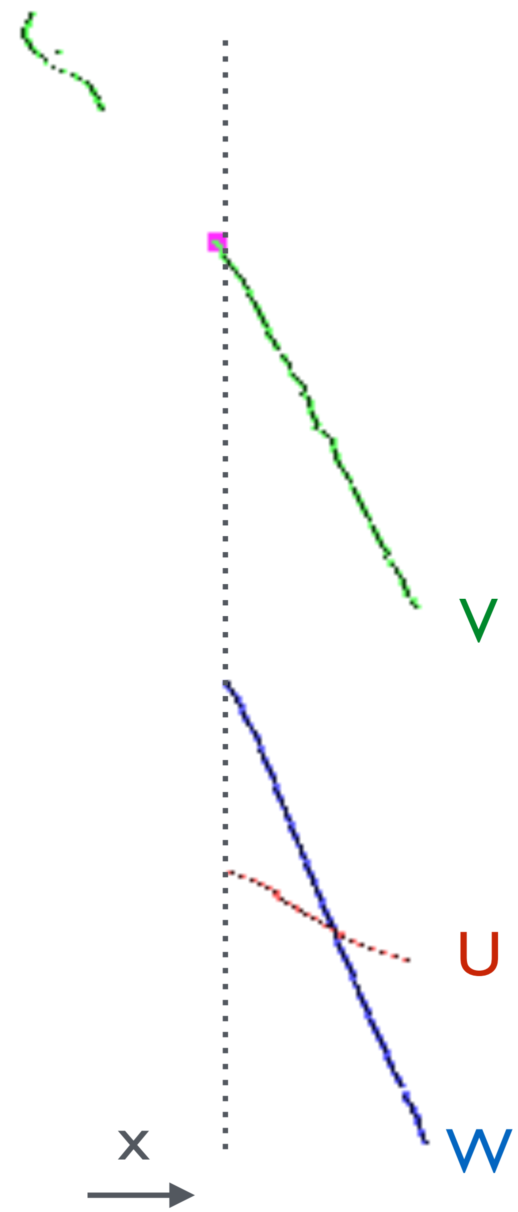
Unambiguous connections, but U cluster has reduced x-span in comparison to V and W clusters

Use V and W clusters to predict continued track position in U view. Add clusters omitted by 2D pattern-recognition failures.

TrackSplitting Tool

- **The TrackSplittingTool performs the reverse operation to address Cluster x-overlap discrepancies.**
- Look for cases where Cluster in a single view appears to be anomalously long.
- Some evidence of a gap in the Cluster, so split to ensure Cluster consistency.
- MissingTrackSegment and TrackSplitting tools - logic careful to avoid repeatedly applying/undoing same operations.

1:1:1



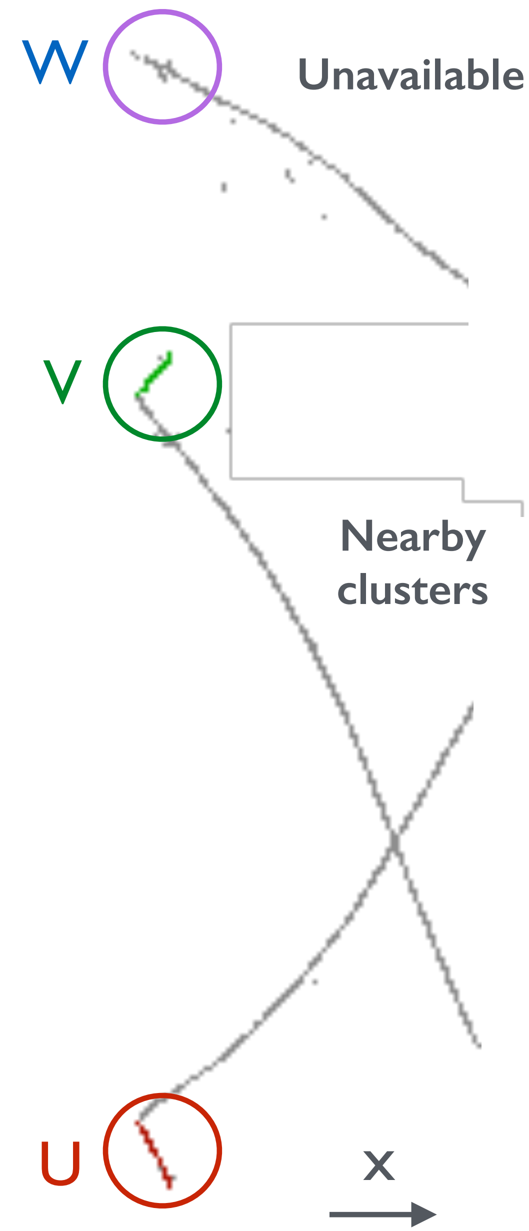
Unambiguous connections, but V Cluster has much larger x-span than U and W clusters

U and W cluster minimum x-positions match closely, plus there is evidence of a gap in the V cluster: split the cluster.

MissingTrack Tool

- **The MissingTracksTool looks for cases where particle features may be obscured in one view.**
- Single Cluster may represent multiple overlapping particles in one view.
- Tool looks for appropriate Cluster overlap using the relationship information available from tensor.
- If selection satisfied, can create a Particle consisting of just two Clusters.

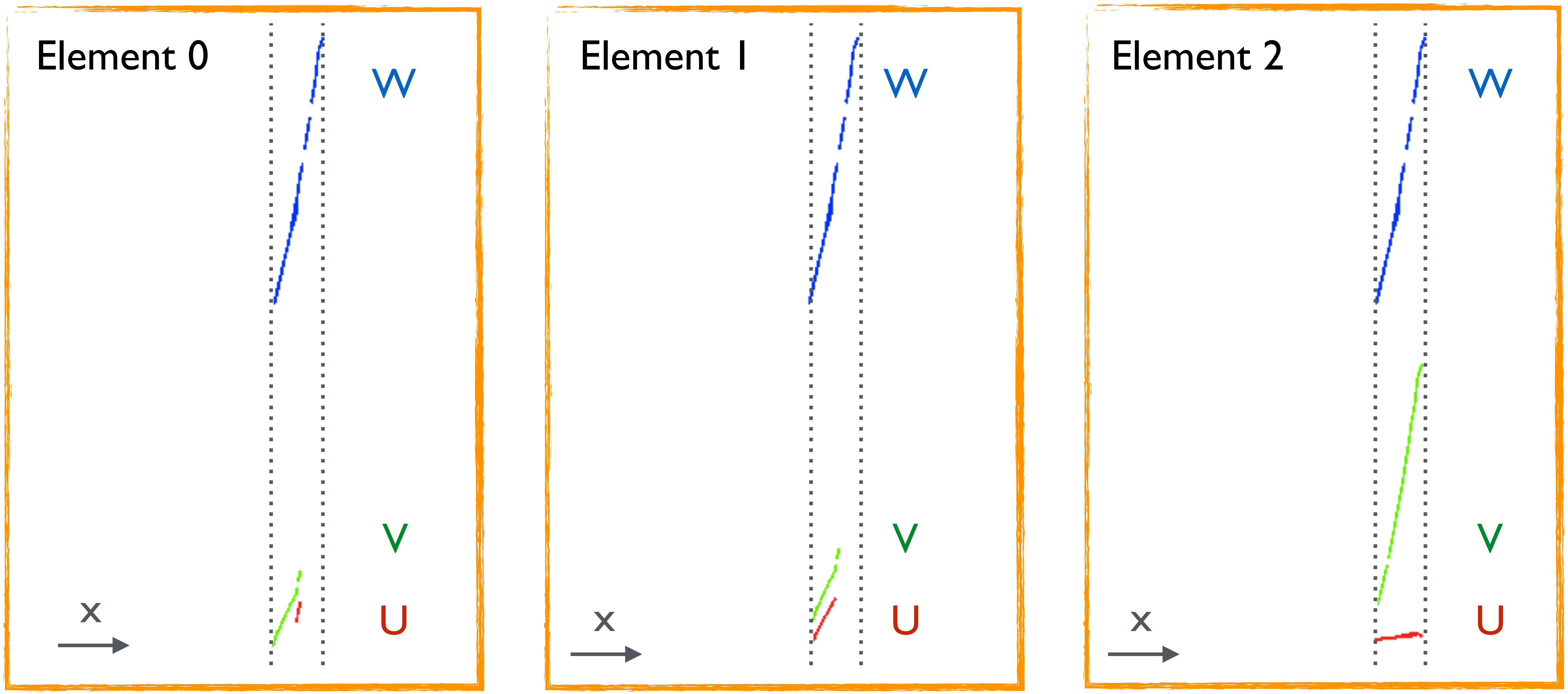
2:2:1



Small U and V Clusters match W Cluster, but this is unavailable: already in a long Particle

If the matching is very good, and it seems that there must simply be two overlapping tracks, create a two-cluster particle.

TensorVisualisation Tool



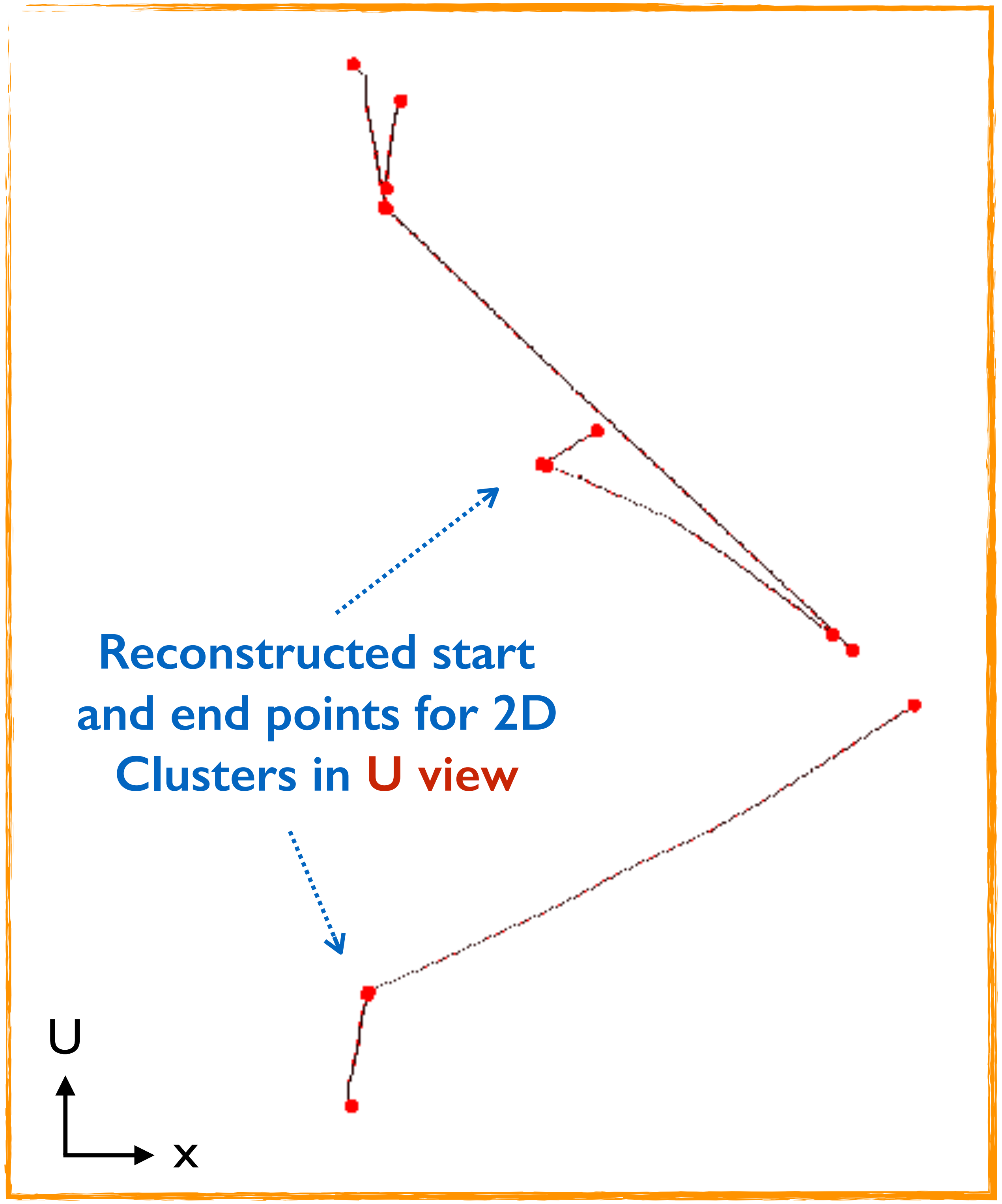

```
> Running Algorithm: 0x7feef6db4c80, LArThreeDTransverseTracks
----> Running Algorithm Tool: 0x7feef6db4ee0, LArTransverseTensorVisualization
Connections: nU 3, nV 2, nW 1, nElements 3
Element 0: MatchedFraction 1, MatchedSamplingPoints 18, xSpanU 1.18993, xSpanV 8.50827, xSpanW 14.9815, xOverlapSpan 1.18861
Press return to continue ...

Element 1: MatchedFraction 1, MatchedSamplingPoints 81, xSpanU 6.87953, xSpanV 8.50827, xSpanW 14.9815, xOverlapSpan 6.80493
Press return to continue ...

Element 2: MatchedFraction 1, MatchedSamplingPoints 187, xSpanU 13.9872, xSpanV 14.1038, xSpanW 14.9815, xOverlapSpan 13.6472
Press return to continue ...
```

Result here: picks Element 2 and also makes a separate, two-Cluster Particle

ThreeDLongitudinalTracks Alg

```
class ThreeDLongitudinalTracksAlgorithm : public ThreeDTracksBaseAlgorithm<LongitudinalOverlapResult>
```

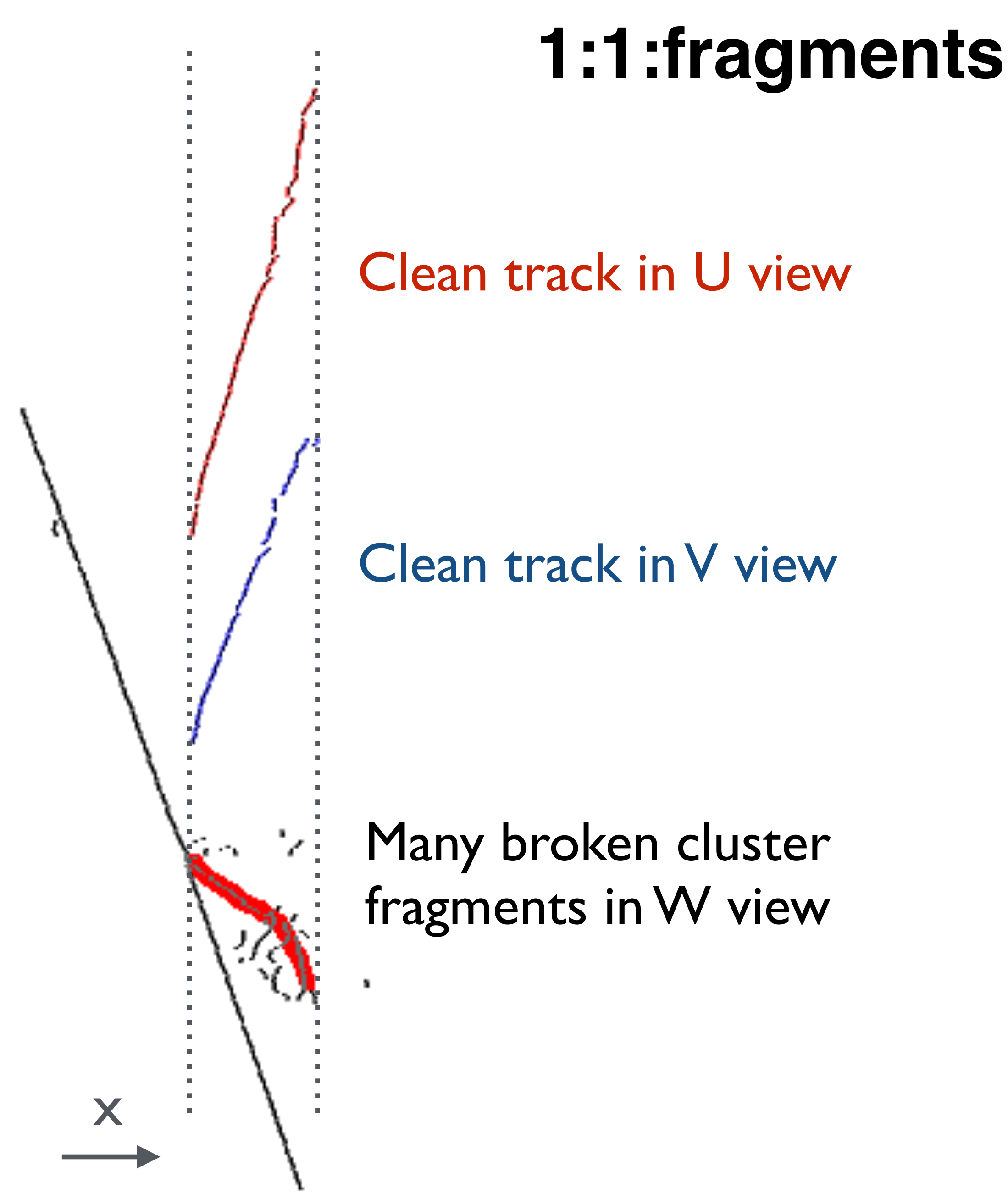
- **ThreeDLongitudinalTracks Algorithm stores a different OverlapResult type in its tensor and uses different tools.**
- Examine case where x-extent of a Cluster grouping is small.
- There are too many ambiguities when trying to sample Clusters at fixed x.
- Such longitudinal Clusters typically left untouched by TransverseTracks alg.
- New alg postulates that Cluster start and end positions match in U,V and W views.
- Allows creation of 3D end-points, so defining a 3D trajectory to assess the Cluster compatibility.
- Simple tools to create Particles for clear matches and address obvious ambiguities.



ThreeDTrackFragments Alg

```
class ThreeDTrackFragmentsAlgorithm : public ThreeDTracksBaseAlgorithm<FragmentOverlapResult>
```

- **Look for situations with single clean Clusters in two views, associated to multiple fragments in third view.**
- A different type of algorithm with a different type of OverlapResult stored in its tensor.
- OverlapResult stores list of matched Hits and their parent Clusters, plus fraction of projected positions resulting in a match.
- Fragment Clusters can be merged, enabling the Particle to be recovered.



ParticleRecovery Alg

Aggressively match any remaining, unassociated track-like Clusters.

Simplified approach and drop requirement for matches in all three views.

SimpleOverlapTensor

```
/**
 *  @brief  Add an association between two clusters to the simple overlap tensor
 *
 *  @param  pCluster1 address of cluster 1
 *  @param  pCluster2 address of cluster 2
 */
void AddAssociation(const pandora::Cluster *const pCluster1, const pandora::Cluster *const pCluster2);


pandora::ClusterList    m_keyClusters;              ///< The list of key clusters
ClusterNavigationMap    m_clusterNavigationMapUV;   ///< The cluster navigation map U->V
ClusterNavigationMap    m_clusterNavigationMapVW;   ///< The cluster navigation map V->W
ClusterNavigationMap    m_clusterNavigationMapWU;   ///< The cluster navigation map W->U
```

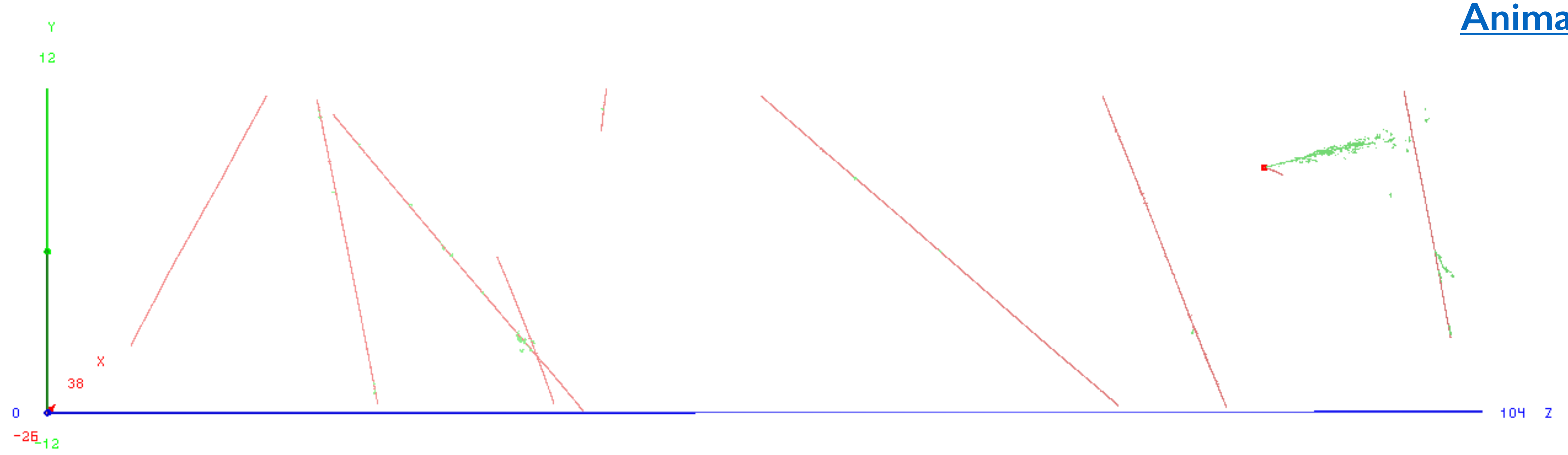
```
void ParticleRecoveryAlgorithm::ExamineTensor(const SimpleOverlapTensor &overlapTensor) const
{
    for (const Cluster *const pKeyCluster : overlapTensor.GetKeyClusters())
    {
        ClusterList clusterListU, clusterListV, clusterListW;

        overlapTensor.GetConnectedElements(pKeyCluster, true, clusterListU, clusterListV, clusterListW);
        const unsigned int nU(clusterListU.size()), nV(clusterListV.size()), nW(clusterListW.size());

        if ((0 == nU * nV) && (0 == nV * nW) && (0 == nW * nU))
            continue;

        if ((1 == nU * nV * nW) && this->CheckConsistency(clusterListU, clusterListV, clusterListW))
        {
            this->CreateTrackParticle(clusterListU, clusterListV, clusterListW);
        }
        else if ((0 == nU * nV * nW) && ((1 == nU && 1 == nV) || (1 == nV && 1 == nW) || (1 == nW && 1 == nU)))
        {
            this->CreateTrackParticle(clusterListU, clusterListV, clusterListW);
        }
        else
        {
            // TODO May later choose to resolve simple ambiguities, e.g. of form nU:nV:nW == 1:2:0
        }
    }
}
```

ParticleRecoveryAlgorithm

3D Hit Creation

- **Particles contain 2D Clusters from (typically) multiple readout planes. For each input 2D Hit in a Particle, attempt to create a new 3D Hit or "SpacePoint".**
- Mechanics differ depending upon Cluster topologies. Series of Algorithm tools used for:
  - Hits on transverse tracks with Clusters in all views,
  - Hits on longitudinal tracks with Cluster in all views,
  - Hits on tracks that are multivalued at specific x coordinates,
  - Hits on tracks with Clusters in only two views,
  - Hits in showers, etc.

Animated gif



3D Hit Creation Tools

```
/**
 *  @brief  Create a new three dimensional hit from a two dimensional hit
 *
 *  @param  pCaloHit2D the address of the two dimensional calo hit, for which a new three dimensional hit is to be created
 *  @param  position3D the position vector for the new three dimensional calo hit
 *  @param  pCaloHit3D to receive the address of the new three dimensional calo hit
 */
void CreateThreeDHit(const pandora::CaloHit *const pCaloHit2D, const pandora::CartesianVector &position3D,
    const pandora::CaloHit *&pCaloHit3D) const;

/**
 *  @brief  Get the list of 2D calo hits in a pfo for which 3D hits have and have not been created
 *
 *  @param  pPfo the address of the pfo
 *  @param  usedHits to receive the list of two dimensional calo hits for which three dimensional hits have been created
 *  @param  remainingHits to receive the list of two dimensional calo hits for which three dimensional hits have not been created
 */
void SeparateTwoDHits(const pandora::ParticleFlowObject *const pPfo, pandora::CaloHitList &usedHits,
    pandora::CaloHitList &remainingHits) const;


typedef std::vector<HitCreationBaseTool*> HitCreationToolList;
HitCreationToolList     m_algorithmToolList;        ///< The algorithm tool list
```

ThreeDHitCreationAlgorithm

Algorithm passes Particle and all unused 2D Hits to an XML-configured, ordered list of tools.

HitCreationBaseTool

```
/**
 *  @brief  Run the algorithm tool
 *
 *  @param  pAlgorithm address of the calling algorithm
 *  @param  pPfo the address of the pfo
 *  @param  inputTwoDHits the list of input two dimensional hits
 *  @param  newThreeDHits to receive the new three dimensional hits
 */
virtual void Run(ThreeDHitCreationAlgorithm *const pAlgorithm, const pandora::ParticleFlowObject *const pPfo,
    const pandora::CaloHitList &inputTwoDHits, pandora::CaloHitList &newThreeDHits) = 0;
```

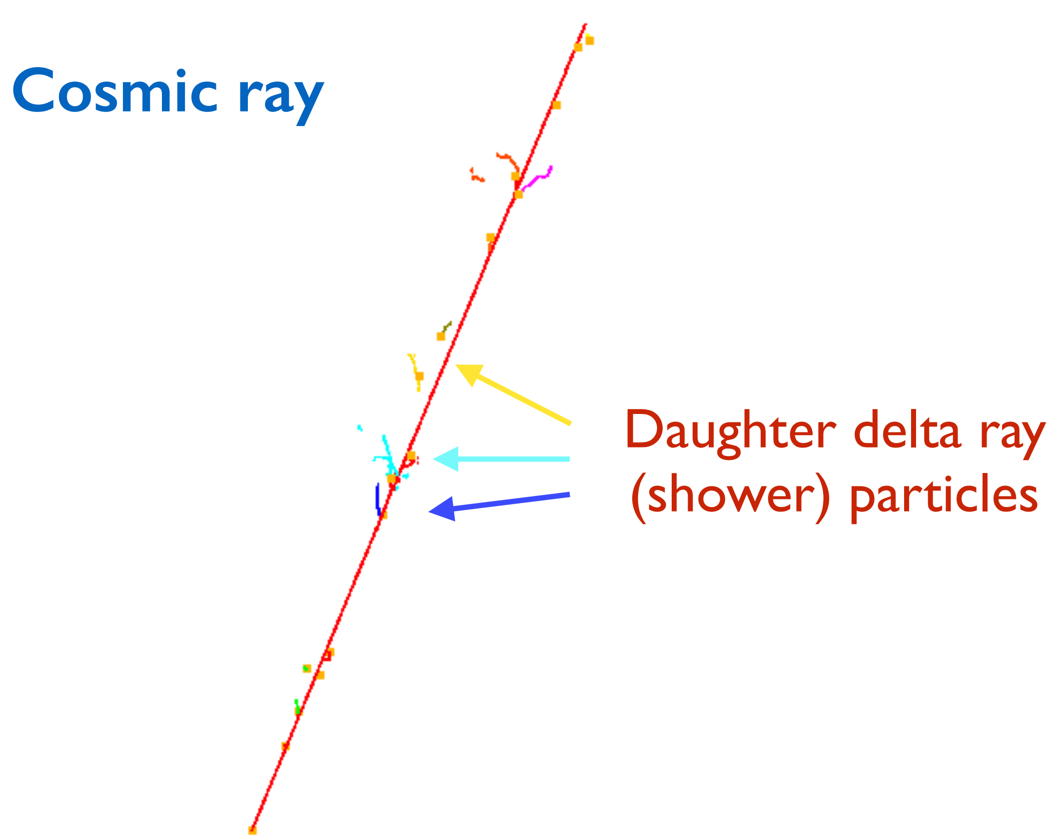
Approaches to 3D Hit Creation

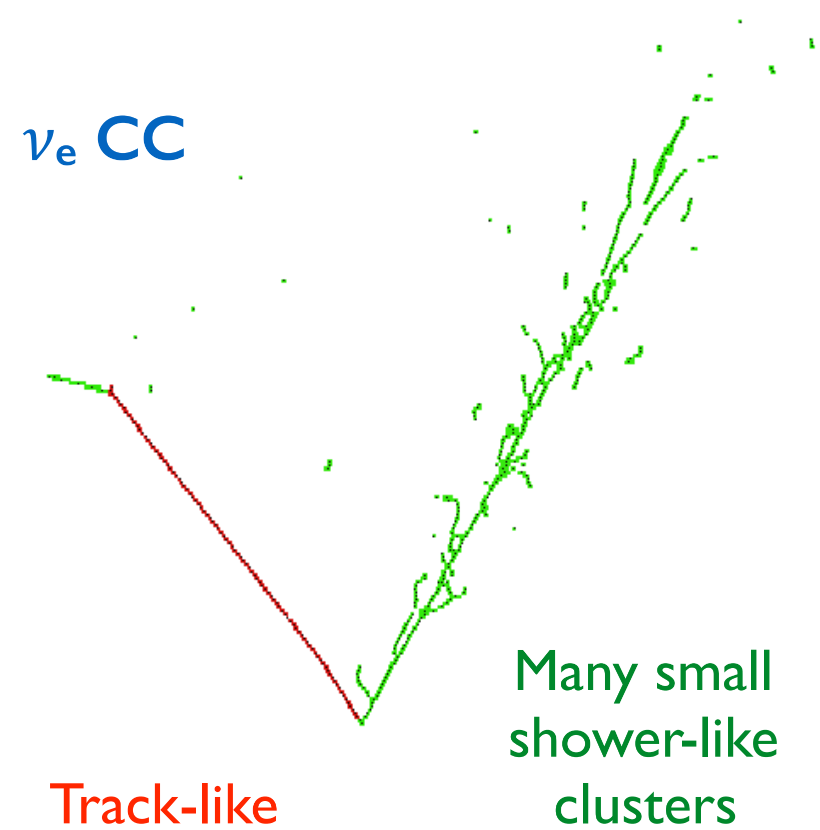
- **For simple transverse tracks, with Clusters in all views, approach is to take 2D Hit in one view e.g. U and sliding fit positions for e.g. V and W Clusters at same x coordinate.**
- Function provided as part of Coordinate Transformation Plugin (registered by client app) provides analytic χ^2 minimisation to provide optimal y and z coordinates at specified x.
- Can also run in mode whereby chosen y and z coordinates are such that they represent a projection of the two fit positions onto the specific wire associated with the 2D Hit.

```
/**
 *  @brief  Get the y, z position that yields the minimum chi squared value with respect to specified u, v and w coordinates
 *
 *  @param  u the u coordinate
 *  @param  v the v coordinate
 *  @param  w the w coordinate
 *  @param  sigmaU the uncertainty in the u coordinate
 *  @param  sigmaV the uncertainty in the v coordinate
 *  @param  sigmaW the uncertainty in the w coordinate
 *  @param  y to receive the y coordinate
 *  @param  z to receive the z coordinate
 *  @param  chiSquared to receive the chi squared value
 */
virtual void GetMinChiSquaredYZ(const double u, const double v, const double w, const double sigmaU, const double sigmaV, const double sigmaW,
    double &y, double &z, double &chiSquared) const = 0;

typedef std::pair<double, pandora::HitType> PositionAndType;

/**
 *  @brief  Get the y, z position that corresponds to a projection of two fit positions onto the specific wire associated with a hit
 *
 *  @param  hitPositionAndType the hit position and hit type
 *  @param  fitPositionAndType1 the first fit position and hit type
 *  @param  fitPositionAndType2 the second fit position and hit type
 *  @param  sigmaHit the uncertainty in the hit coordinate
 *  @param  sigmaFit the uncertainty in the fit coordinates
 *  ...
 */
virtual void GetProjectedYZ(const PositionAndType &hitPositionAndType, const PositionAndType &fitPositionAndType1,
    const PositionAndType &fitPositionAndType2, const double sigmaHit, const double sigmaFit, double &y, double &z, double &chiSquared) const = 0;
```

Remaining Reconstruction Steps

- **For cosmic-ray reconstruction pass, any remaining Hits (not in a track Particle) are reclustered using a simple, proximity-based algorithm to find delta-rays:**
  - Use a few topological association algs to improve delta-ray completeness before matching delta-ray Clusters between views and identifying appropriate cosmic-ray parent Particle.
- **For neutrino pass, still need to find interaction Vertex, perform 2D shower reco (adding branches to long Clusters representing shower spines) and build 3D shower Particles.**
  - Discussed in Talks 6 and 7.





Questions?