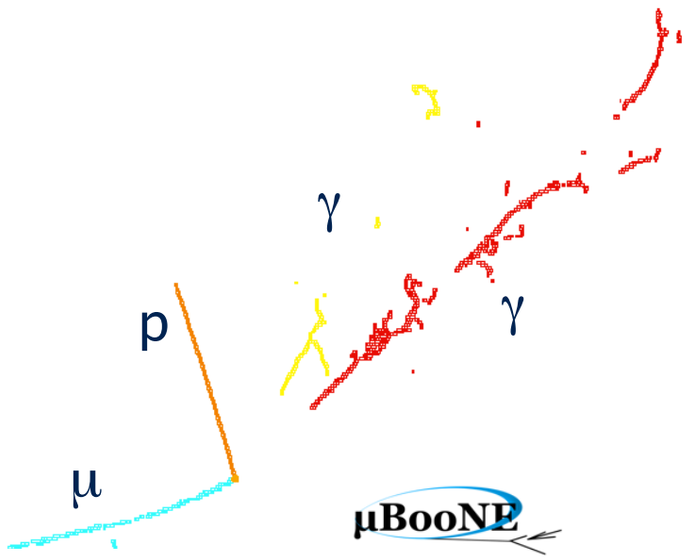


Pandora Talk 3: SDK Details

J. S. Marshall for the Pandora Team

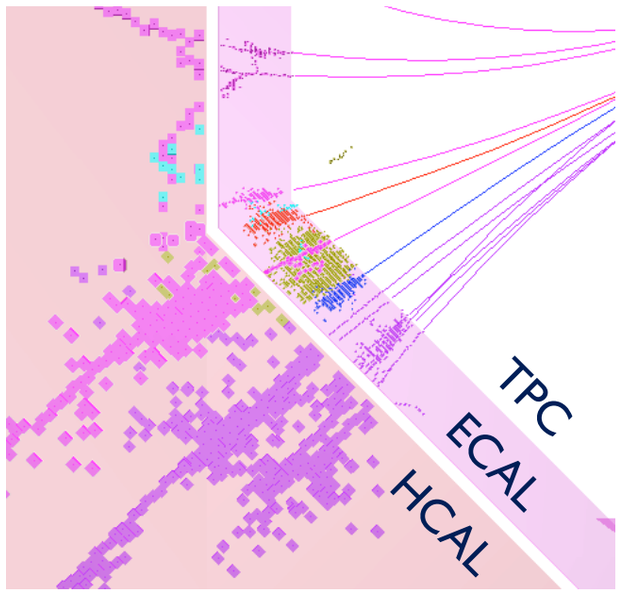
MicroBooNE Pandora Workshop

July 11-14th 2016, Cambridge

Introduction

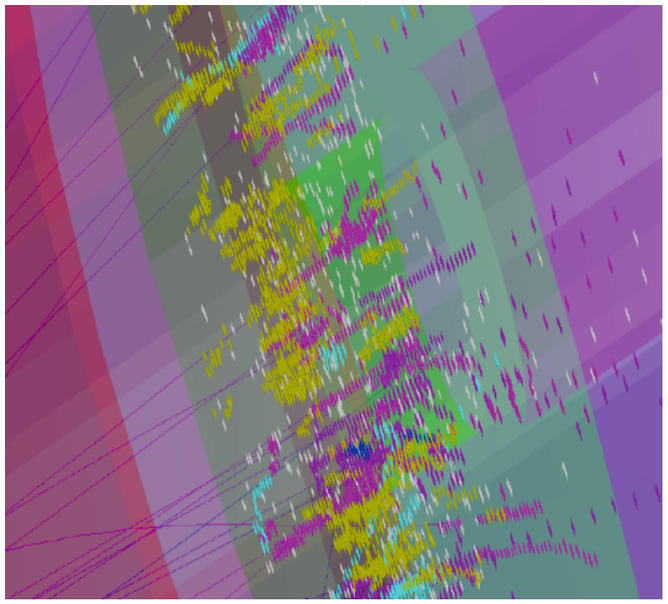
- The idea behind the Pandora Software Development Kit (SDK) is that the operations required to solve almost all pattern-recognition problems are well-defined:
  - Sort input points in time or space into higher-level structures e.g. Clusters,
  - Refine Clusters by merging and/or splitting operations,
  - Sort Clusters into groups and/or hierarchies, e.g. representing Particles.
- What differs between problems is the precise *logic* used to govern these operations.

NIMA.2009.09.009, NIMA.2012.10.038



arXiv:1307.7335, 1506.05348

LHCC-P-008



Design Principles

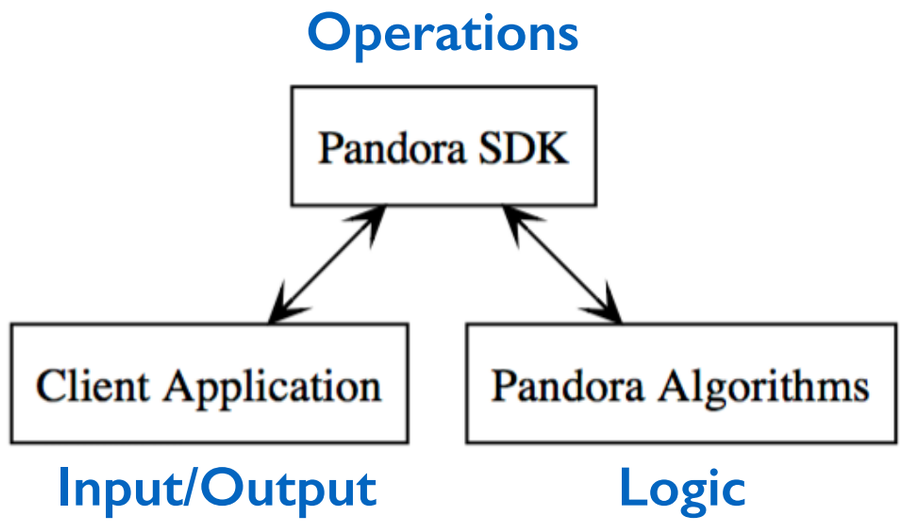
- **Created the Pandora SDK for developing and running pattern-recognition algorithms, with Application Programming Interfaces (APIs) designed to ensure that:**

  1. It is easy for users to provide the building-blocks defining 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 requested by algs, performed by Pandora.

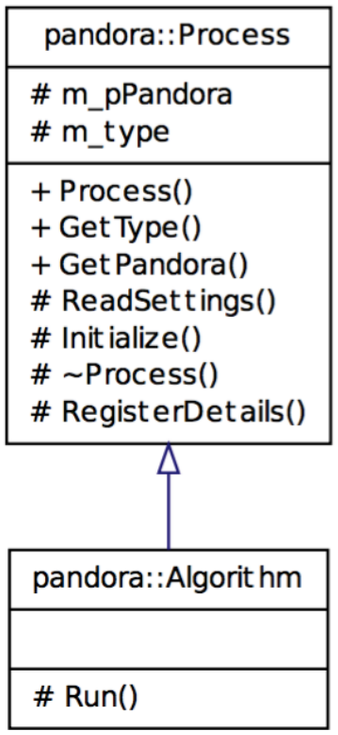
- This design is well-suited to the multi-algorithm approach: use a large number of decoupled algorithms, each targeting specific event topologies, typically merging or splitting Clusters.



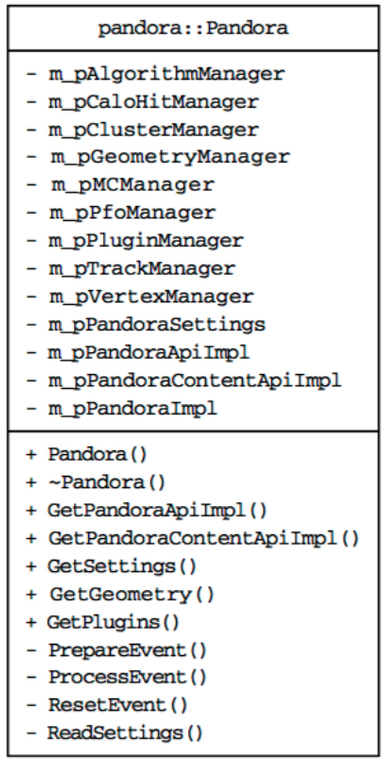
Implementation

- **The SDK consists of a dependency-free C++ library and its associated APIs. It provides an Event Data Model (EDM) for managing pattern-recognition problems.**
- Instances of objects in the EDM are owned by Pandora Managers and are stored in named lists. The Managers are able to create new objects, delete objects, create and save new lists, etc.
- The Managers provide a complete set of low-level operations that allow all the high-level operations likely to be needed by pattern-recognition algorithms to be satisfied.

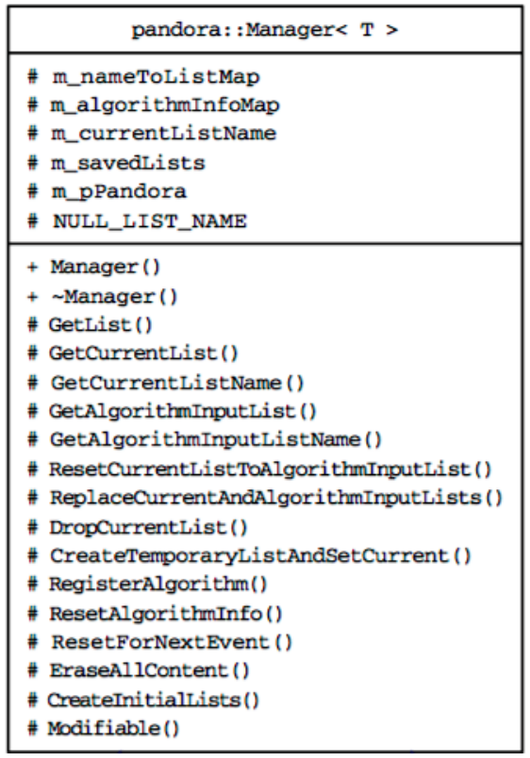
Discussed in this talk:



Algorithm

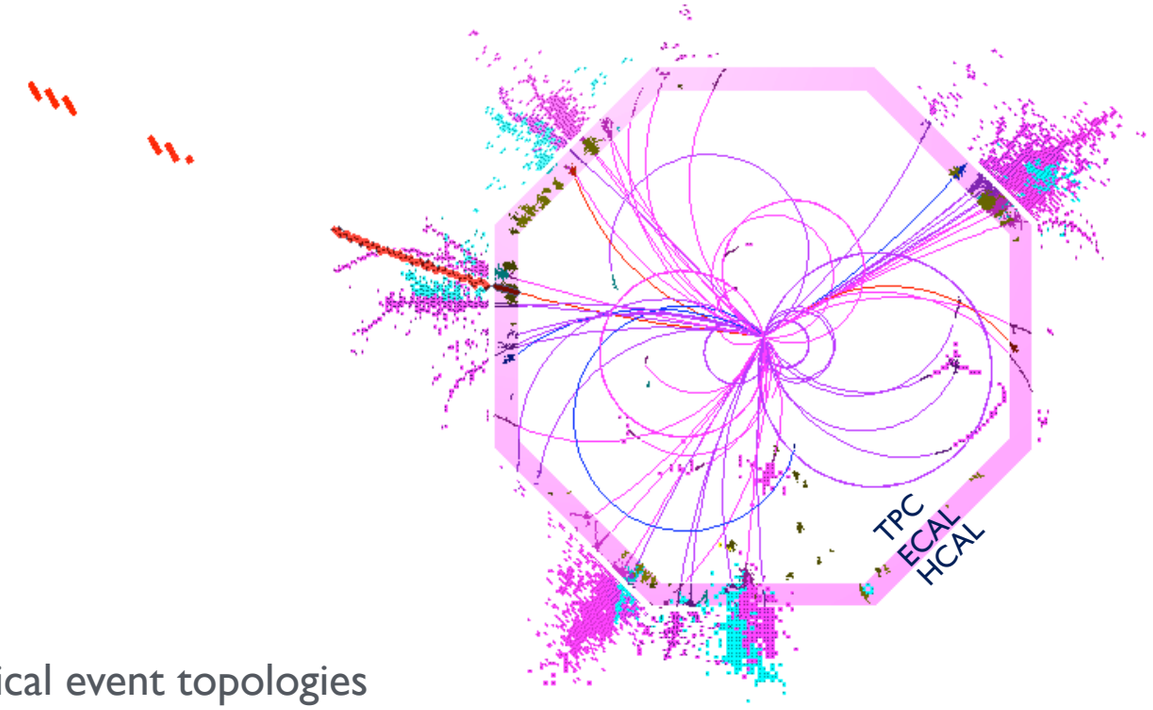


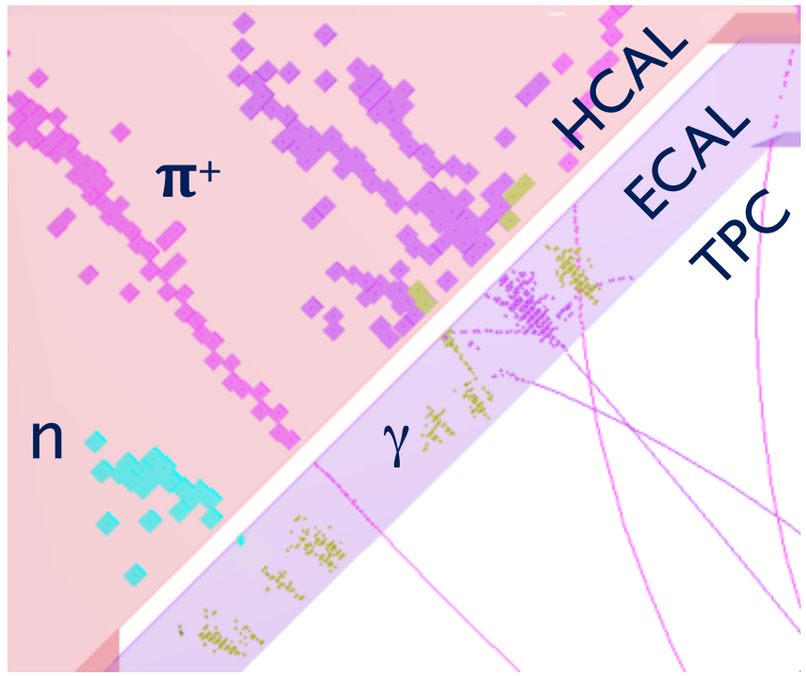
Pandora



Manager

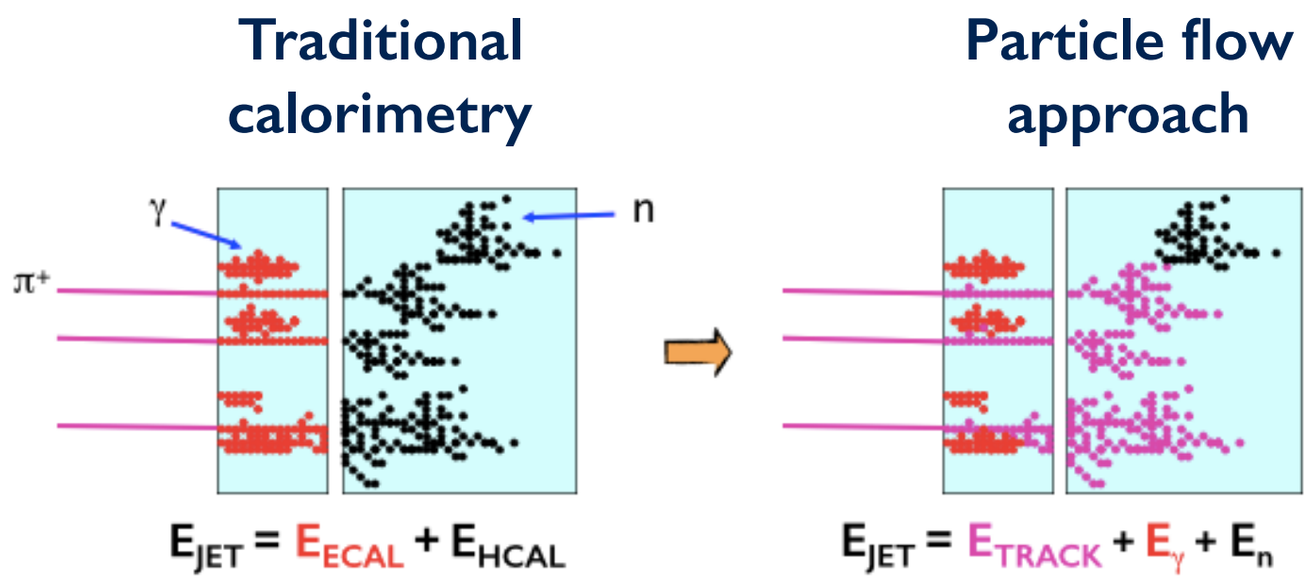
Historical Context





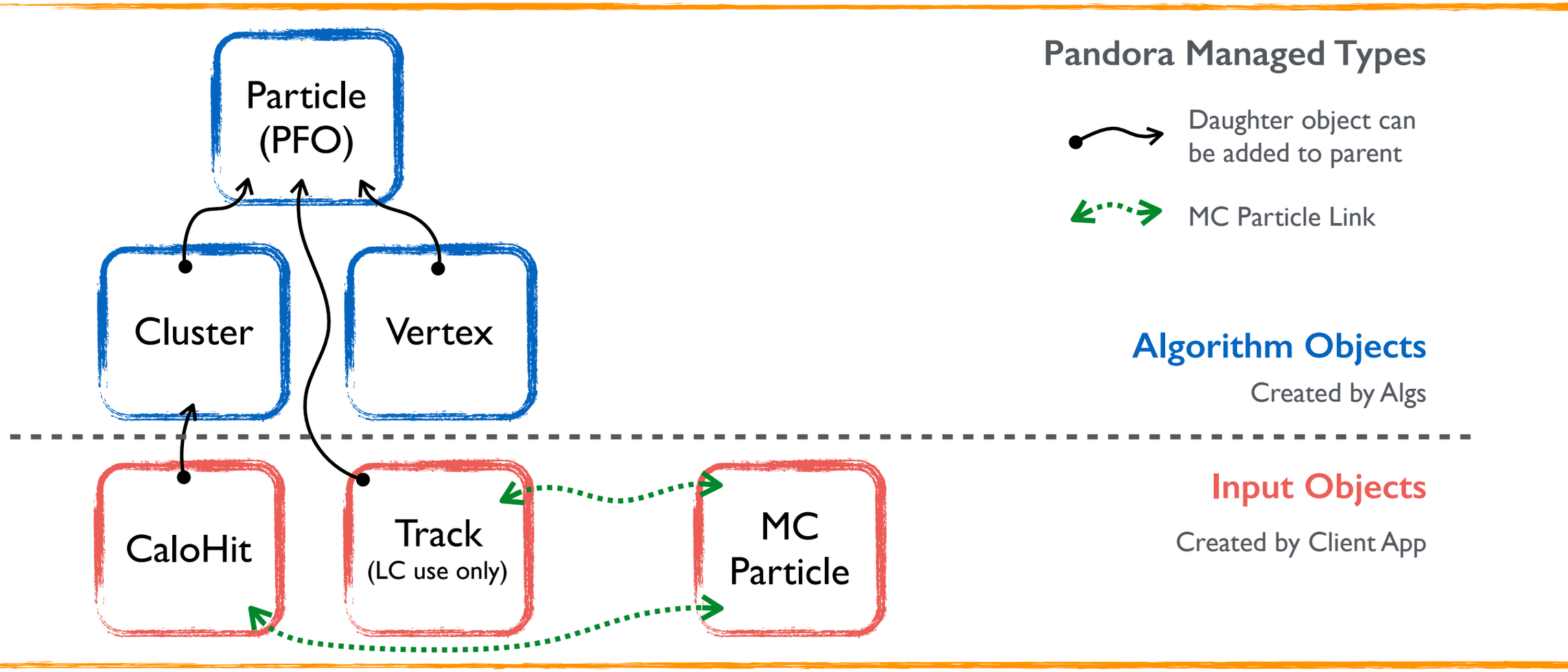
Typical event topologies

- SDK developed for *reimplementation* of particle flow reconstruction at future e^+e^- linear collider.
- Informed by lessons learned during original PandoraPFA implementation:
  - Support multi-algorithm approach
  - Support reclustering and recursion



Event Data Model

- **EDM consists of classes to represent the input building-blocks for pattern-recognition problems and the structures that can be created using these building-blocks.**
- Provides well-defined development environment for managing pattern-recognition problems and allows for independence of algorithms, which can only communicate via the EDM.
- EDM aims to be self-describing, with each object providing all the information required to allow investigation and processing by the pattern-recognition algorithms.



Input Objects

- **Input Objects are the building-blocks for pattern recognition, typically created by the client app before algorithm operations begin.**
- Their properties are defined at creation and cannot be changed. They are instead used to build new constructs, termed “Algorithm Objects".
- The usage of all Input Objects is monitored to ensure that no double-counting/usage occurs.

CaloHit	Primary building-block, defining a position and extent in space (or time), with an associated intensity or energy measurement and detector location details.
Track (LC use only)	Represents a continuous trajectory of well-defined space-points, with helix parameterisation. Track parent-daughter and sibling relationships supported.
MC Particle	For development purposes, provide details of true pattern-recognition solution. Support parent-daughter links and can be associated to CaloHits and Tracks.

Algorithm Objects

- **Algorithm Objects represent the higher-level structures created in order to solve pattern-recognition problems.**
- Pandora carefully manages the allocation and manipulation of these objects and all non-const operations can only be requested by algorithms via the Pandora Content APIs.
- Pandora is then able to perform the memory-management for these objects.

Cluster	Collection of CaloHits and main working-horse for algorithms (which create, merge, split Clusters). Provides some derived properties of CaloHit collection.
Vertex	The identification and classification of a specific point in space, typically used to flag positions of particle creation or decay.
Particle	Container of Clusters, Tracks and Vertices, together with metadata describing e.g. particle type. Ultimate Pandora output and can represent a hierarchy.

Object Creation

- **As seen during discussion of client application, instantiation of objects in Pandora EDM follows a template design pattern with a clean and simple interface.**
- Object creation is typically requested by the client app (Input Objects) or by an algorithm (Algorithm Objects). Must create a parameters instance and provide all information up-front.
- Request to create object then made to Pandora, which will check that all required information has been provided and, if so, perform the allocation.
- The new object instance is owned and managed by Pandora (see upcoming discussion of object Managers), but can be accessed and manipulated by algorithms, via the APIs.

Provides clean, simple interface to create any/all Pandora objects:

i. Construct parameters, e.g. PandoraApi::CaloHit::Parameters
ii. Assign properties to parameters public member variables
iii. Request object creation, e.g. PandoraApi::CaloHit::Create(…)
iv. Failure to assign to all properties will raise an exception

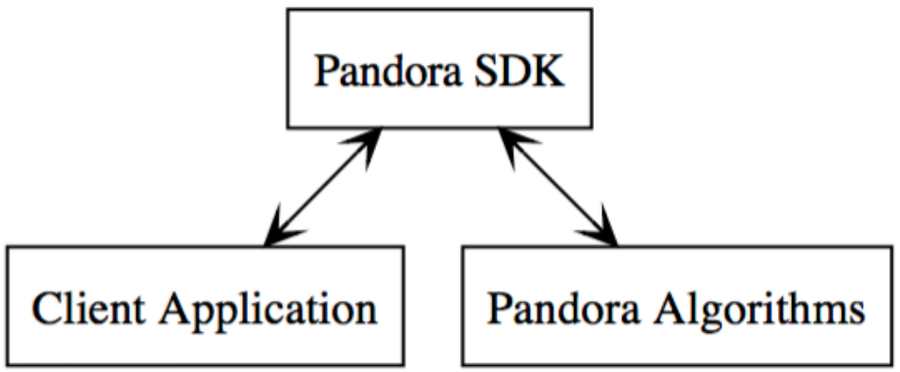
In client app:

```
PandoraApi::CaloHit::Parameters caloHitParameters;
caloHitParameters.m_positionVector = ...
caloHitParameters.m_expectedDirection = ...
...

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

Client Application

- **As previously discussed, client app is responsible for providing Input Objects that define the pattern-recognition problem and for persisting the output Particles.**
- Also responsible for creating Pandora instances, bringing-together (collections of) algorithm implementations and for configuring the reconstruction via the Pandora Settings XML file.
- Algorithms depend on Pandora SDK, but can also have as many external dependencies as required. The client app depends on Pandora and on all libraries providing algorithms.
- The actual algorithm instances used in the reconstruction are not created unless specified in the Pandora Settings; created when the XML file is parsed by Pandora.



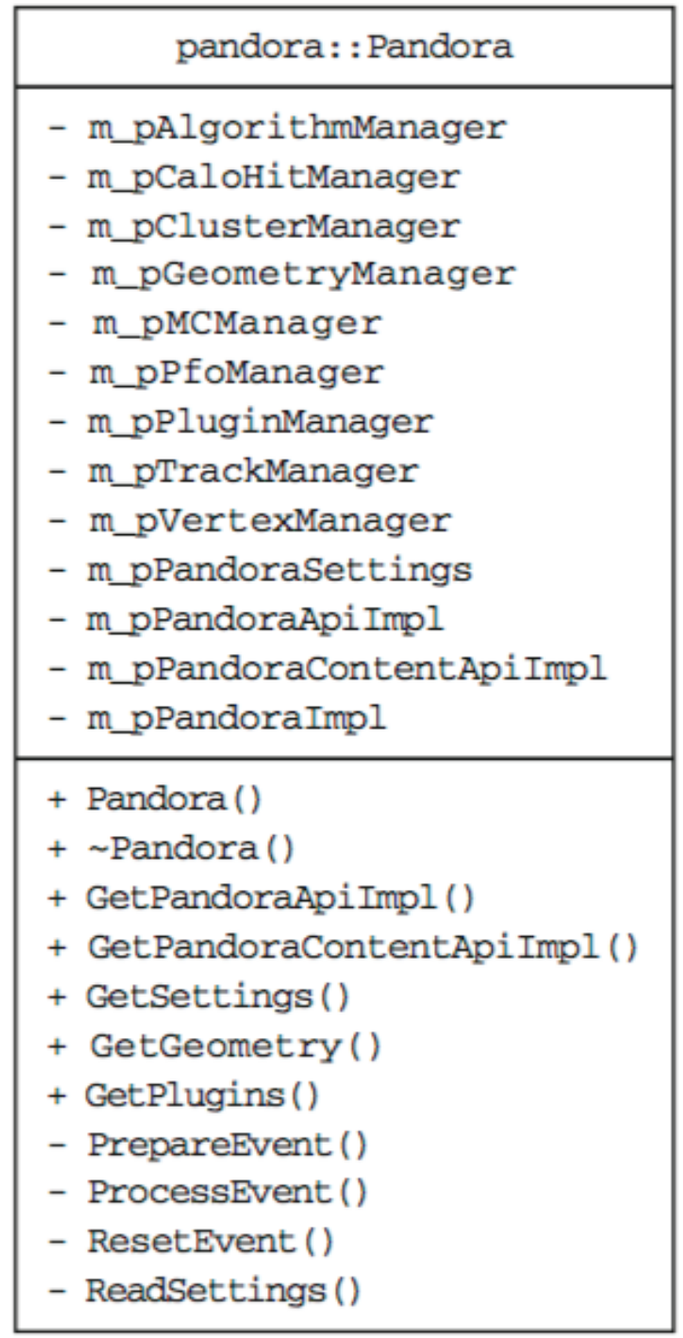
Built with client app, or in libraries of Pandora content

Algorithm 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
```

Managers

- **At very heart of Pandora design are the Managers, which own all instances of objects in Pandora EDM.**
- The Managers are designed to provide a complete set of low-level object manipulation functions.
- Algs request high-level services (e.g. merge two Clusters), which are then satisfied when the hidden implementation calls the low-level Manager functions in the correct order.
- Approach helps ensure that implementation is extensible, easy to maintain and rather human-readable.
- Key part of design is that algorithms can *only* access or modify managed objects via the APIs, so Managers are able to perform memory-management.



A Pandora instance is simply a container of Manager instances and API implementation instances

Managers

- **Pandora objects are heap-allocated and their addresses are stored in named object lists, owned by the relevant object Manager instance.**
- Object lists are unordered_sets, a storage strategy that ensures efficient retrieval of specific object instances, although unordered nature means care is often required in algorithms.
- Each Manager holds a mapping from the list name (string) to address of the object list. It also stores the set of saved list names, plus the name of the algorithm-designated "current" list.

- **Algorithms can use the Pandora APIs to receive const references to the object lists from the Managers. Algorithms can access lists by name or ask for the current list.**

PandoraContentApi.h

```
/**
 *  @brief  Get the current list
 *
 *  @param  algorithm the algorithm calling this function
 *  @param  pT to receive the address of the current list
 *  @param  listName to receive the current list name
 */
template <typename T>
static pandora::StatusCode GetCurrentList(const pandora::Algorithm &algorithm, const T *&pT, std::string &listName);
```

- Managers hold address of associated Pandora instance and record details of all algs running: e.g. current list name when alg began, names of any temporary lists created.

Managers

Manager template base class provides functionality for supervising and accessing named lists of objects.

Derived classes provide functionality reflecting different rules governing creation and usage of Algorithm and Input Objects.

Input Object Managers

- **Input Objects can be created, via APIs, by any function with access to the Pandora instance. Most common point of creation is the client application.**
- Newly-requested objects are created on heap by relevant Manager, and address is stored in a specific named list: the "Input" list.
- Idea is that Input Objects cannot be modified or deleted by algorithms, although new, refined objects could be created. Input list keeps full record of all instances created.
- Algorithms can choose to work with Input list or, more typically, save new lists (under new names) containing only a subset of the Input list (Input Objects can appear in multiple lists).
- Memory-management is simple, as all Input Objects are deleted, and all lists erased/reset, only when the client application asks to reset Pandora between events.

CaloHit

Track
(LC use only)

MC Particle

Algorithm Object Managers

- **Memory-management is considerably more complex for Algorithm Objects, which will be created, modified and deleted as the pattern recognition progresses.**
- Pandora enforces a specific approach which maintains flexibility, but is ultimately built around its flagship reclustering functionality.
- To create a new Algorithm Object, must first instruct relevant Manager to have a new, temporary object list as the current list, waiting to receive newly-created instances.
- The temporary list is associated with the alg that requested it. When this alg finishes processing the event, all its temporary lists are erased and the list contents deleted.
- In order to persist the Algorithm Objects, the algorithm must first ask to save some/all the objects in a new or existing named list.
- Unlike Input Objects, it is enforced that Algorithm Objects can exist in only one list.

Cluster **Vertex** **Particle**

Monitoring Object Usage

- **Algorithm Objects are typically containers of other objects. Clusters, are containers of CaloHits, whilst Particle are containers of Clusters, Tracks and Vertices.**
- Important role played by the Managers is to monitor object usage and ensure that no double-counting can occur.
- Monitoring generally simple, but significantly more complex when reclustering allows algorithms to simultaneously explore multiple alternative Cluster configurations!
- Enforce that objects cannot appear in multiple objects (e.g. must remove from first before allowed to add to second). In reclustering, rules applied for each *set* of Cluster candidates.

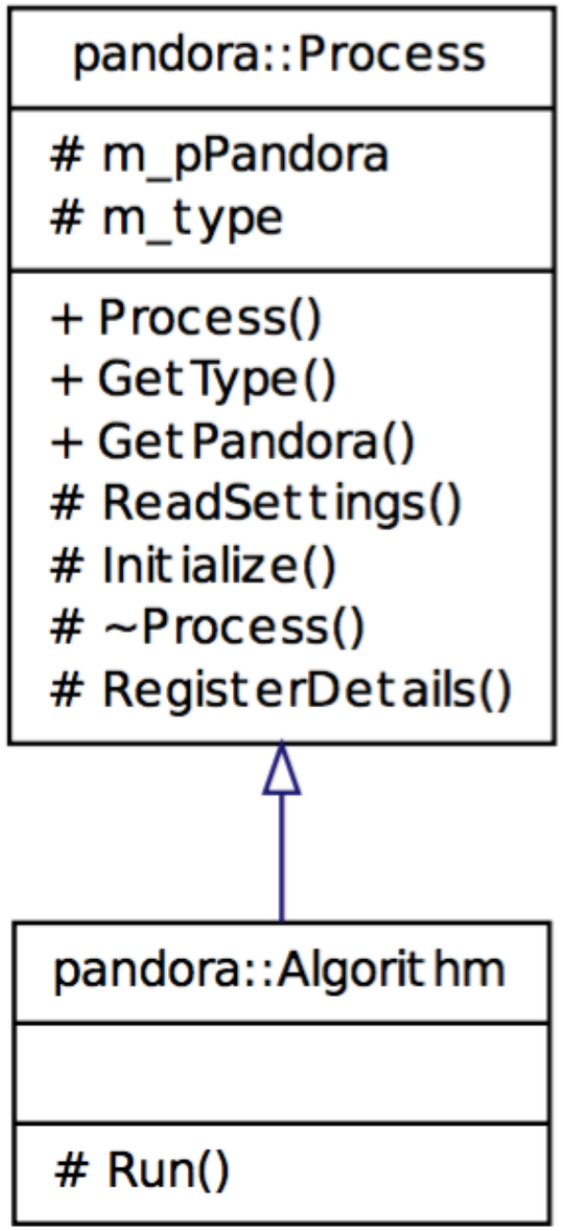
PandoraContentApi.h

Algs can use APIs to ask whether objects are available or have already been used

```
/**
 *  @brief  Is object, or a list of objects, available as a building block
 *
 *  @param  algorithm the algorithm calling this function
 *  @param  pT address of the object
 *
 *  @return boolean
 */
template <typename T>
static bool IsAvailable(const pandora::Algorithm &algorithm, const T *const pT);
```

Algorithms

- **Algs contain step-by-step instructions, using Pandora APIs to request object creation/modification services.**
- Algs inherit from the Pandora Process abstract base class. Inherited functionality controls handshaking between Pandora instance and algorithm instance.
- Process provides ability to receive a ReadSettings callback with an XML handle (tinyxml) from which configurable parameters can be extracted. Also an Initialize callback.
- The Algorithm purely abstract base class provides the interface for the Run callback, which is called each event and is the entry point for all event processing.



- **Algorithm Factories registered (under a specific name), by the client app are extremely simple:**
- Must allocate instance of derived algorithm type and return pointer to Algorithm base class.

Algorithm Configuration

- **Algs configured by XML file provided by client application. Algorithm Manager parses file and looks for algorithm tags within the top-level <Pandora></Pandora> tags.**
- Extracts algorithm type, which must match name of a registered Alg Factory. If match found, Factory creates new instance of desired type and Manager stores pointer to base class.
- After creation, Manager will call ReadSettings member function of new algorithm, providing a handle to the XML element describing the algorithm.
- ReadSettings can demand presence of specific daughter XML tags, or can search for optional tags to override default parameter values, if present. Examples in exercises.

When client app calls `ProcessEvents`, Pandora calls `Run` for each top-level algorithm, in order, then returns thread

```
<algorithm type = "LArCandidateVertexCreation">
    <InputClusterListNameU>ClustersU</InputClusterListNameU>
    <InputClusterListNameV>ClustersV</InputClusterListNameV>
    <InputClusterListNameW>ClustersW</InputClusterListNameW>
    <OutputVertexListName>CandidateVertices</OutputVertexListName>
    <ReplaceCurrentVertexList>true</ReplaceCurrentVertexList>
</algorithm>
<algorithm type = "LArVertexSelection">
    <InputCaloHitListNameU>CaloHitListU</InputCaloHitListNameU>
    <InputCaloHitListNameV>CaloHitListV</InputCaloHitListNameV>
    <InputCaloHitListNameW>CaloHitListW</InputCaloHitListNameW>
    <OutputVertexListName>SelectedVertices</OutputVertexListName>
    <ReplaceCurrentVertexList>true</ReplaceCurrentVertexList>
    <BeamMode>true</BeamMode>
</algorithm>
```

Nested Algorithms

- **The Algorithm Manager only searches for algorithm XML tags within the top-level Pandora tags. These are the algorithms to be called, in order, each event.**
- In its ReadSettings callback, however, each algorithm is given full control of parsing details contained within its XML tag.
- The algorithm can search for nested daughter algorithms, which could be specified in a named list, or may be identified via an XML description attribute.
- The parent alg can use an API to instruct the Alg Manager to construct/configure a new daughter algorithm instance and return the unique name of the daughter algorithm.
- During event processing the parent algorithm can use an API to ask to run the daughter algorithm with the stored unique name.

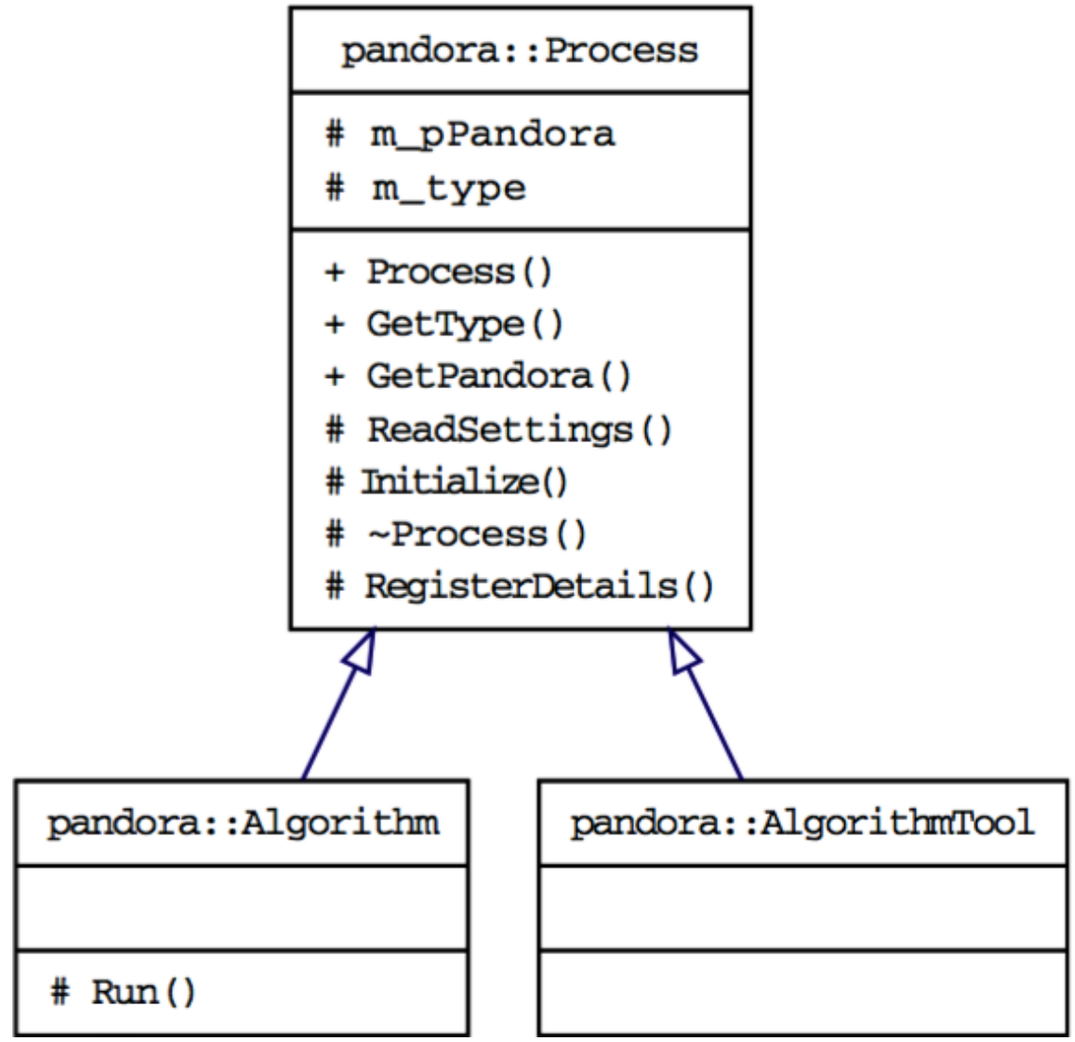
Nesting allows parent alg to e.g. manipulate current object lists, then call reusable daughter algs to process list contents

daughter alg

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

Algorithm Tools

- **Daughter alg functionality promotes the development of small, reusable algs to perform specific operations.**
- Parent and daughter algs are decoupled and can only communicate by manipulating objects in EDM or the object lists.
- AlgorithmTools inherit from the Process class, so have all the handshaking and configuration functionality of an algorithm.
- Don't receive Run callback. Instead, parent alg defines interface for its tools and is given access to pointers to tool instances.
- Parent alg can create complex object, then give it to its tools for processing. Tool selection/configuration specified via XML.



Algorithms must provide Run implementation; AlgorithmTools have user-defined interface to provide services to Algorithms

APIs

- **APIs are static functions, typically templated to allow operations on each of the different types in Pandora EDM.**
- Content APIs only usable by algs and take alg reference as argument, allowing static functions to resolve to a Pandora instance.
- Careful friending of classes ensure the API implementation instance can call Manager functionality inaccessible to other classes.
- APIs used by client app take a reference to a Pandora instance as an argument, but otherwise work in identical manner.
- The final algorithms can be structured around their key API calls and can be written in simple pseudo-code form.

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
```

Reclustering

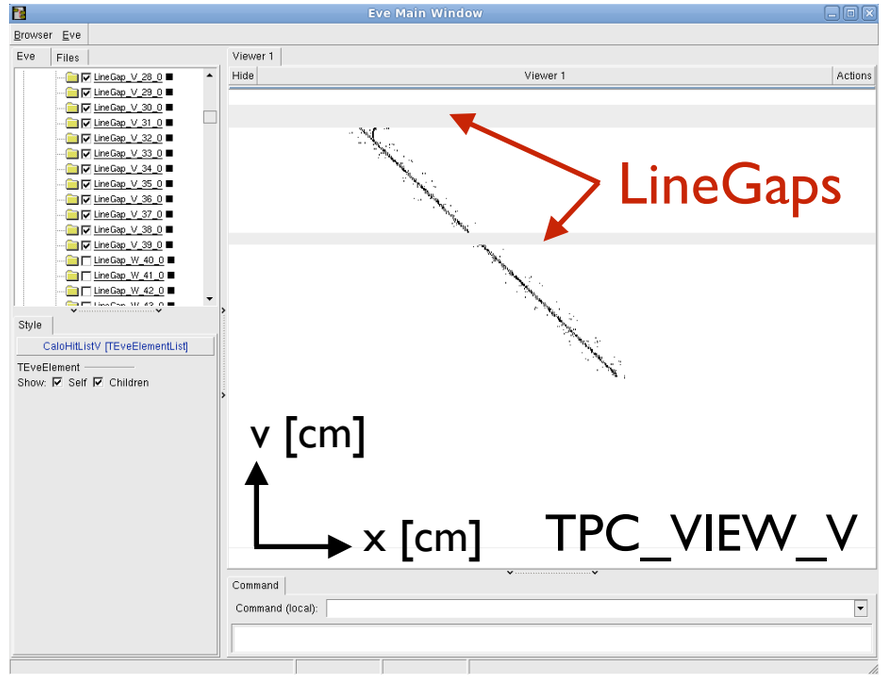
- **Reclustering allows algorithms to simultaneously explore multiple different Cluster configurations. Clustering results can be compared side-by-side and the best selected.**
- Pandora will automatically tidy-up any discarded Cluster options and the selected Clusters will seamlessly replace the originals, which entered the reclustering process.
- Instead of selecting the best algorithmic approach to solve a problem, the user is able to control a process whereby the approach that best solved the problem is identified.
- Yet to exploit this for LAr TPC reco: just need a good figure of merit to assess Clusters.

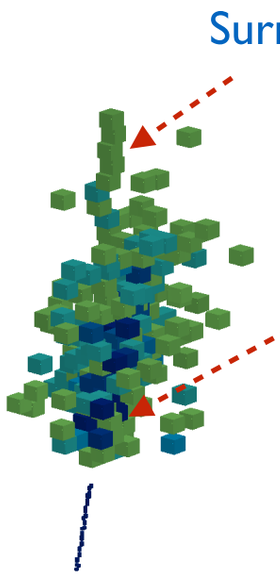
1. **Ask for current Cluster list, spot issues and ask to recluster**
   Original Clusters moved to a new temporary list; current CaloHit list changed
2. **Ask to run a clustering algorithm**
   New temporary list formed and filled by daughter clustering algorithm
3. **Calculate figure of merit for new Cluster candidates**
4. **Repeat stages 2 and 3 as required**
   Can re-use original clustering alg, with different parameters, or try a new alg
5. **Choose most appropriate Cluster candidates**
   Cluster lists will be tidied as required; original Clusters are seamlessly replaced

+ *Local* reclustering allows direct comparison of two Cluster configurations within single alg

Geometry and Plugins

- **Client app can provide basic detector geometry, which can then be accessed by algorithms.**
- Can specify named sub detectors, with assumed polygonal structure.
- Can also provide information about Line, Concentric and Box Gaps in detector active volume.
- In general, existing Pandora algs try to avoid use of geometry info and work with Hits/Clusters alone.



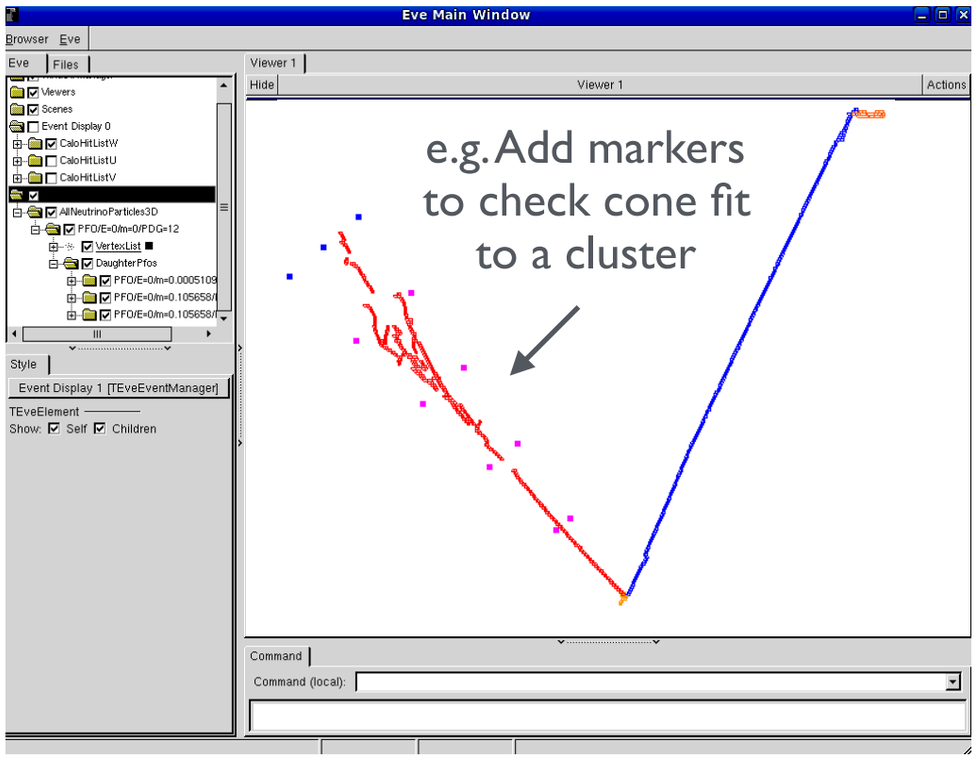


E.g. Novel hadronic energy estimator at ILC

- **Plugins inherit from the Process base class and have interfaces tailored to their specific usage:**
  - Particle id,
  - Cluster energy estimators,
  - EM shower profile characterisation,
  - Magnetic field maps access,
  - Division of detector volume into layers.

Visualisation

- **PandoraMonitoring package depends on the Pandora SDK and ROOT. It understands how to translate Pandora objects into ROOT TEVE for visualisation.**
  - PandoraMonitoring APIs allow algs to perform customised, visual debugging. Algs can choose which objects to display, when and in which colours. Can add guiding markers, etc.
  - Reusable visualisation algs can be added to PandoraSettings XML config files at different points in multi-algorithm reconstruction without rebuilding.
  - Also offers TTree-writing and histogram functionality, whilst keeping ROOT at arm's length.




```
. . .
<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

Persistency

- **Pandora persistency allows Input Objects to be serialised in .pndr files (small, portability not guaranteed) or .xml files (large, but compressible).**
  - No longer need full client/translation app to develop or test algs: can move to lightweight environment where Entry Point constructs Pandora instance and runs reconstruction.
  - Enables development without delays or complications introduced by parent software framework and build system: rebuild and run in seconds, making for healthy development.

```
// ATTN: Edited for slide display; inc. removal of API return value checks
int main(int argc, char *argv[])
{
    Parameters parameters;

    if (!parameters.ParseCommandLine(argc, argv))
        return 1;

    const pandora::Pandora *const pPandora(new pandora::Pandora());
    LArContent::RegisterAlgorithms(*pPandora);
    PandoraApi::ReadSettings(*pPandora, parameters.m_pandoraSettingsFile);

    unsigned int nEvents(0);
    while (nEvents++ < parameters.m_nEventsToProcess)
    {
        PandoraApi::ProcessEvent(*pPandora);
        PandoraApi::Reset(*pPandora);
    }

    delete pPandora;
    return 0;
}
```

- Self-describing Input Objects: algs don't need to worry how/where object properties were calculated.
- Objects serialised/deserialised by Pandora, following requests from EventReading, EventWriting algs.

```
<!-- ALGORITHM SETTINGS -->
<algorithm type = "LArEventReading">
    <EventFileName>/PATH/TO/Events.pndr</EventFileName>
    <ShouldReadEvents>true</ShouldReadEvents>
    <SkipToEvent>0</SkipToEvent>
</algorithm>
```

Building a LAr TPC Reconstruction

- Save input CaloHits in separate U, V and W lists
- For each of U, V and W CaloHit lists:
  - 2D Clustering alg
  - 2D Cluster merging/splitting refinement algs

  ⟩ Talk 4: 2D Reconstruction
- Match Clusters between views and create Particles

  ⟩ Talk 5: 3D Reconstruction
- Particle merging/splitting refinement algs
- Vertex creation and Particle hierarchy construction

  ⟩ Talks 6 and 7: Vertex, Shower and Event Reco.
- Process output

  ⟩ Talk 8: Output and Performance

Questions?