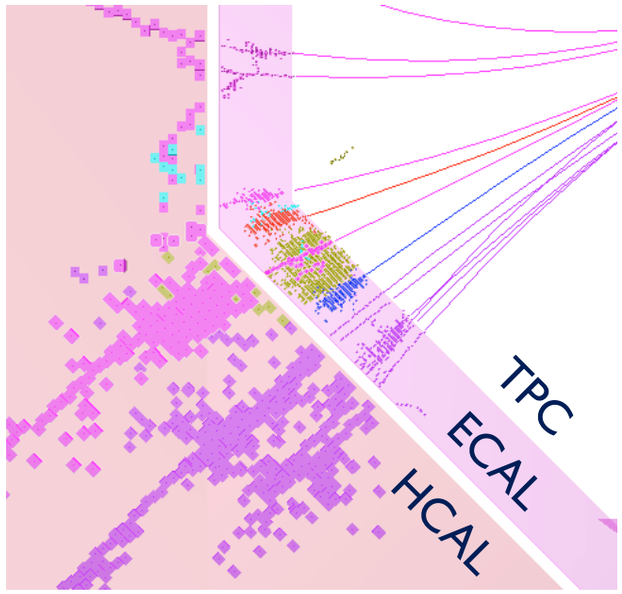
Pandora SDK Details

John Marshall for the Pandora Team

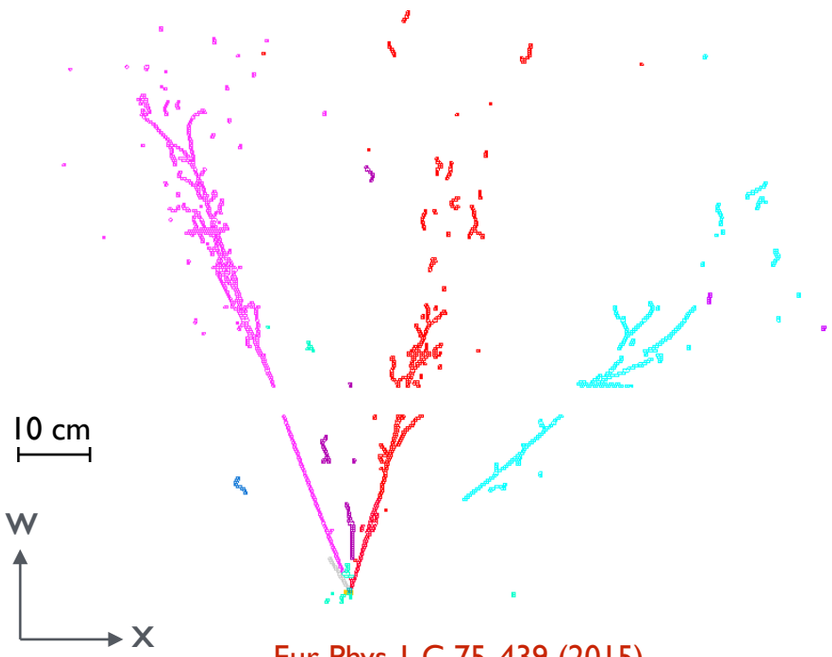
22nd June 2022

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.



Nucl. Instrum. Meth. A611, 25 (2009)
Nucl. Instrum. Meth. A700, 153 (2013)

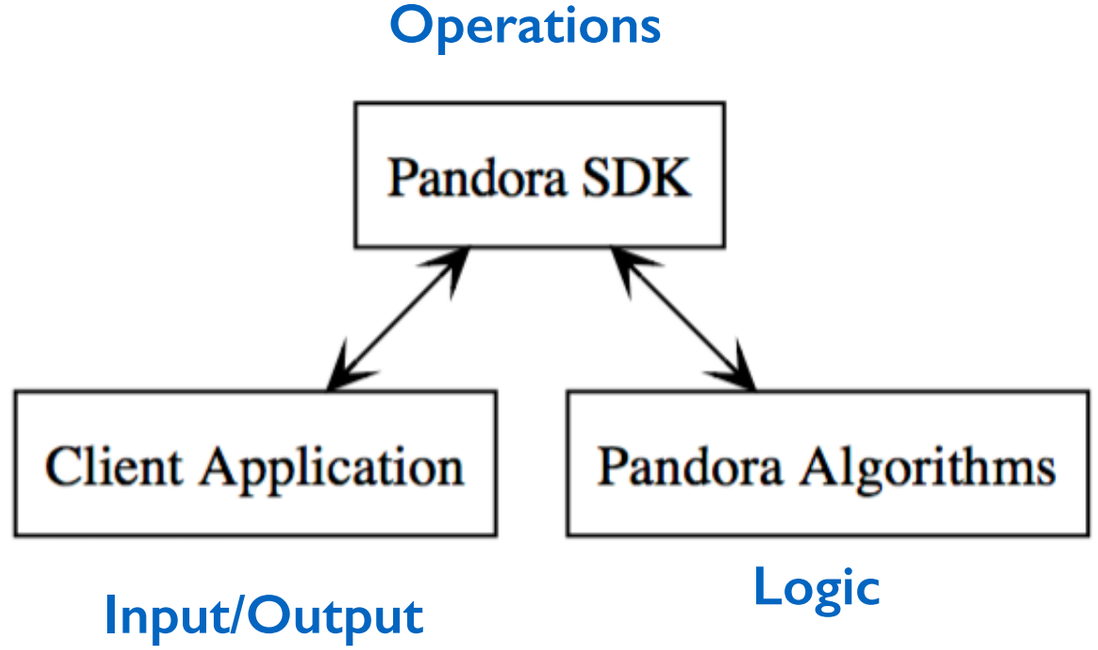


Eur. Phys. J. C 75, 439 (2015)
Eur. Phys. J. C 78, 82 (2018)

Design Principles

- **Created the Pandora SDK to develop and run 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.

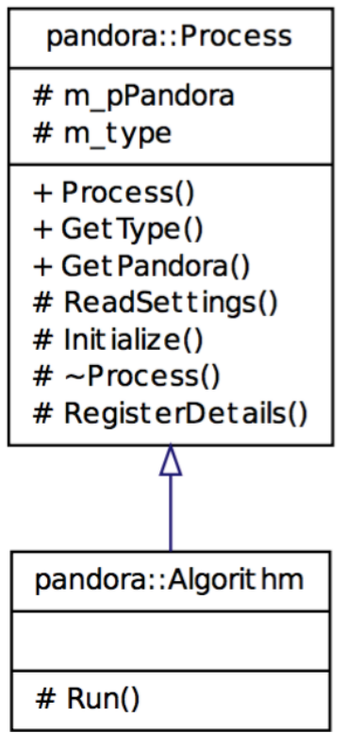


- This design is well-suited to the multi-algorithm approach: use a large number of decoupled algorithms, each targeting specific event topologies, typically merging or splitting Clusters.

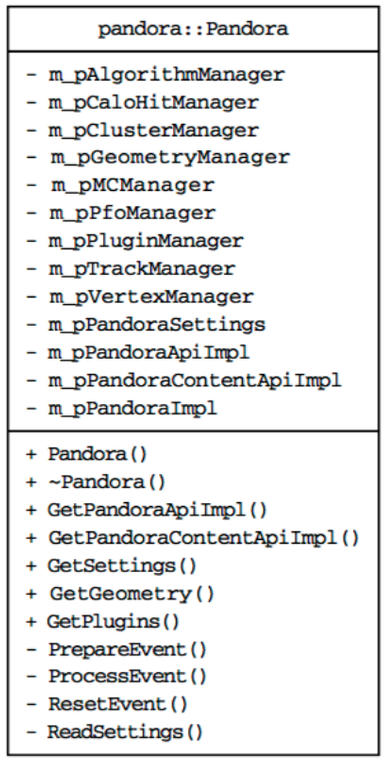
Implementation

- **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.

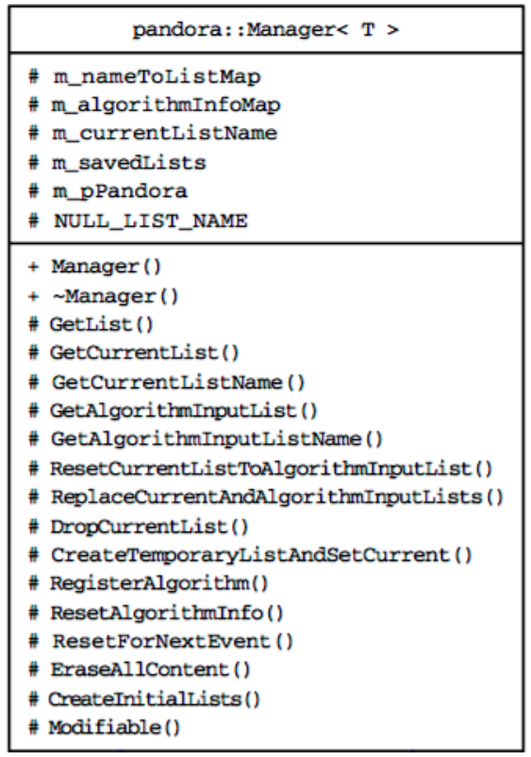
Discussed in this talk:



Algorithm

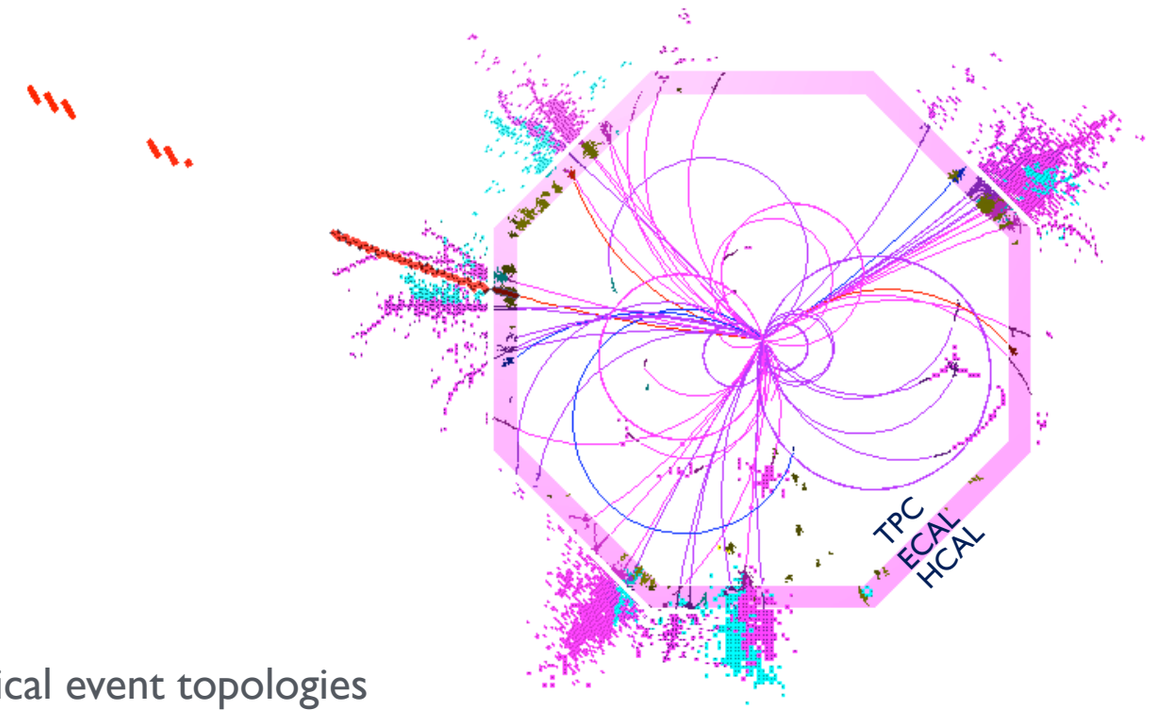


Pandora

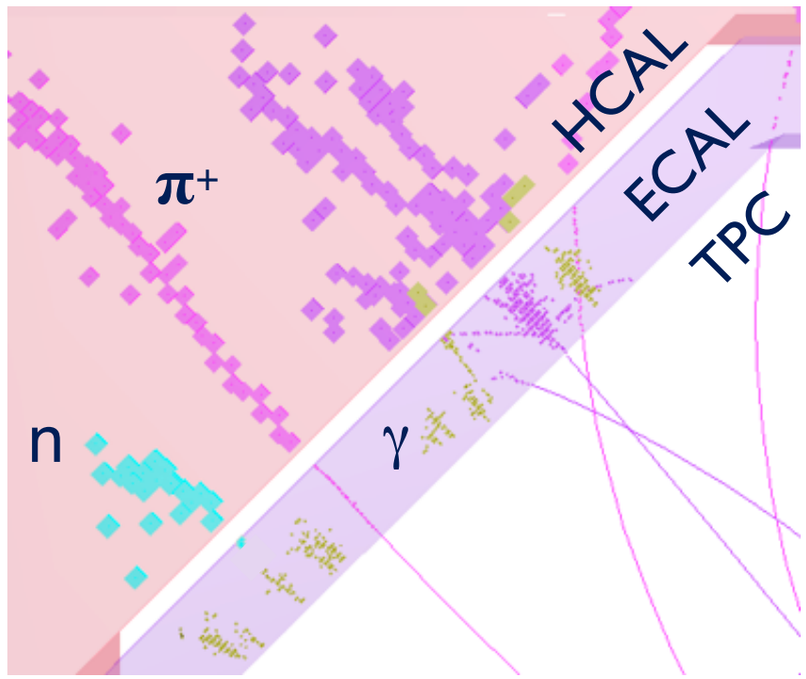


Manager

Historical Context

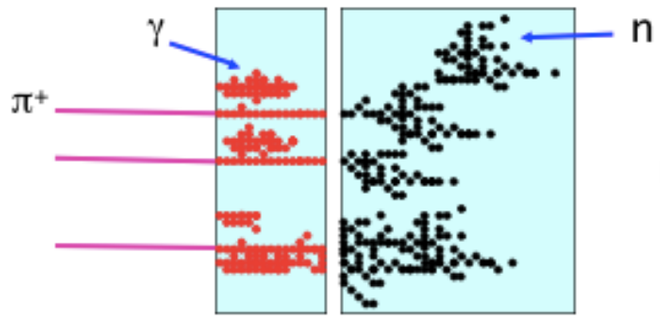


Typical event topologies



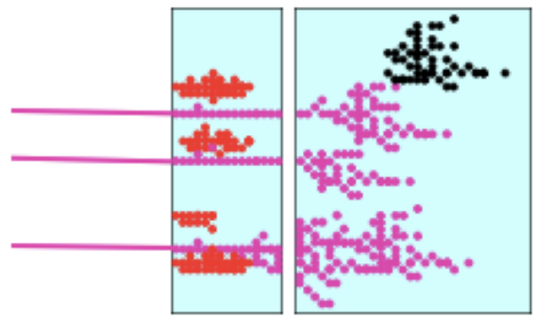
- SDK developed for *reimplementation* of PFA at future e^+e^- linear collider.
- Informed by lessons learned during original PandoraPFA implementation:
  - Support multi-algorithm approach
  - Support reclustering and recursion
- Then applied for LArTPC patrec.

Traditional calorimetry



$E_{JET} = E_{ECAL} + E_{HCAL}$

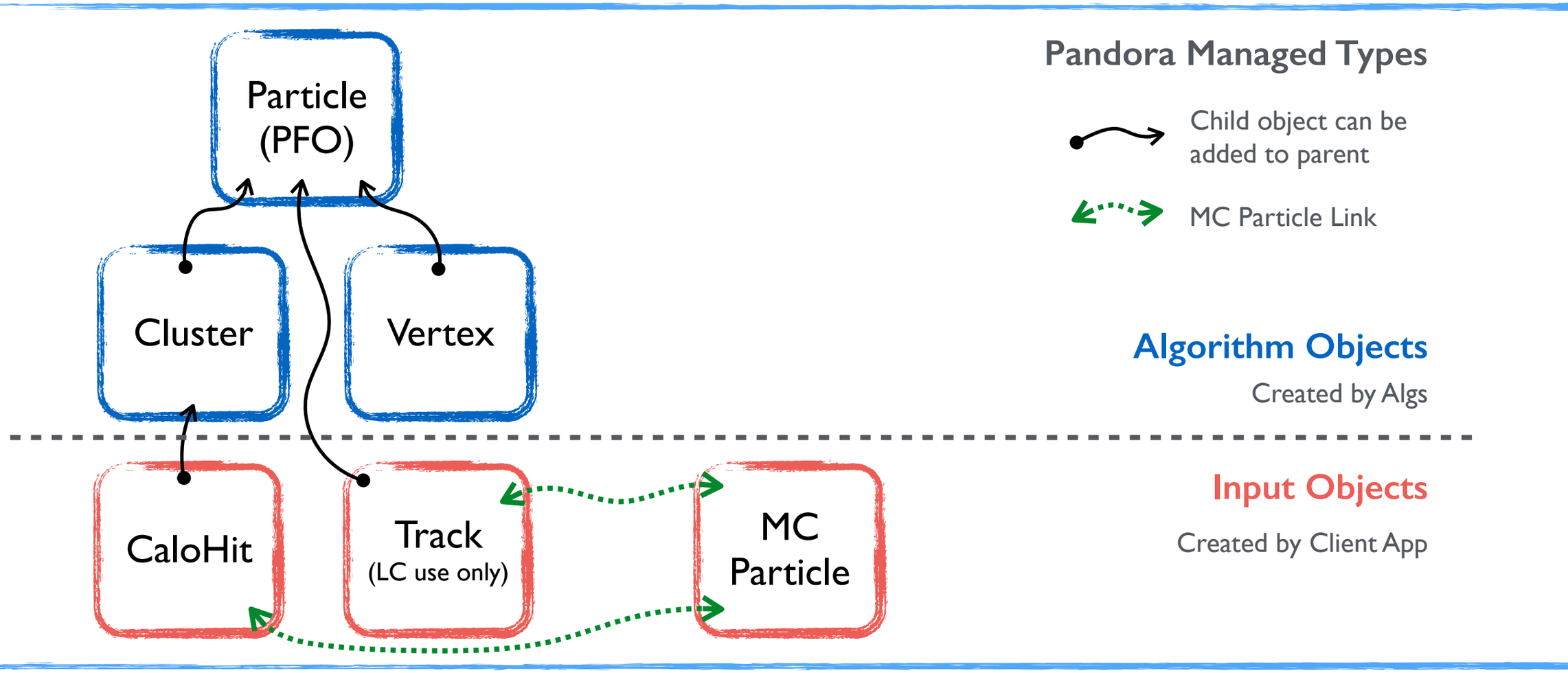
Particle flow approach



$E_{JET} = E_{TRACK} + E_{\gamma} + E_{n}$

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-child and sibling relationships supported.
MC Particle	For development purposes, provide details of true pattern-recognition solution. Support parent-child 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

- **Instantiation of objects in Pandora follows a 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:

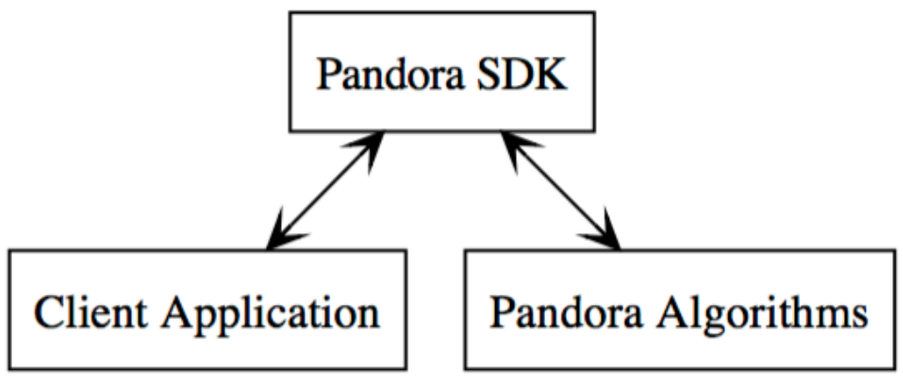
i. Construct parameters, e.g. PandoraApi::CaloHit::Parameters
ii. Assign properties to parameters public member variables
iii. Request object creation, e.g. PandoraApi::CaloHit::Create(…)
iv. Failure to assign to all properties will raise an exception

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

- **A client (or translation) 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.



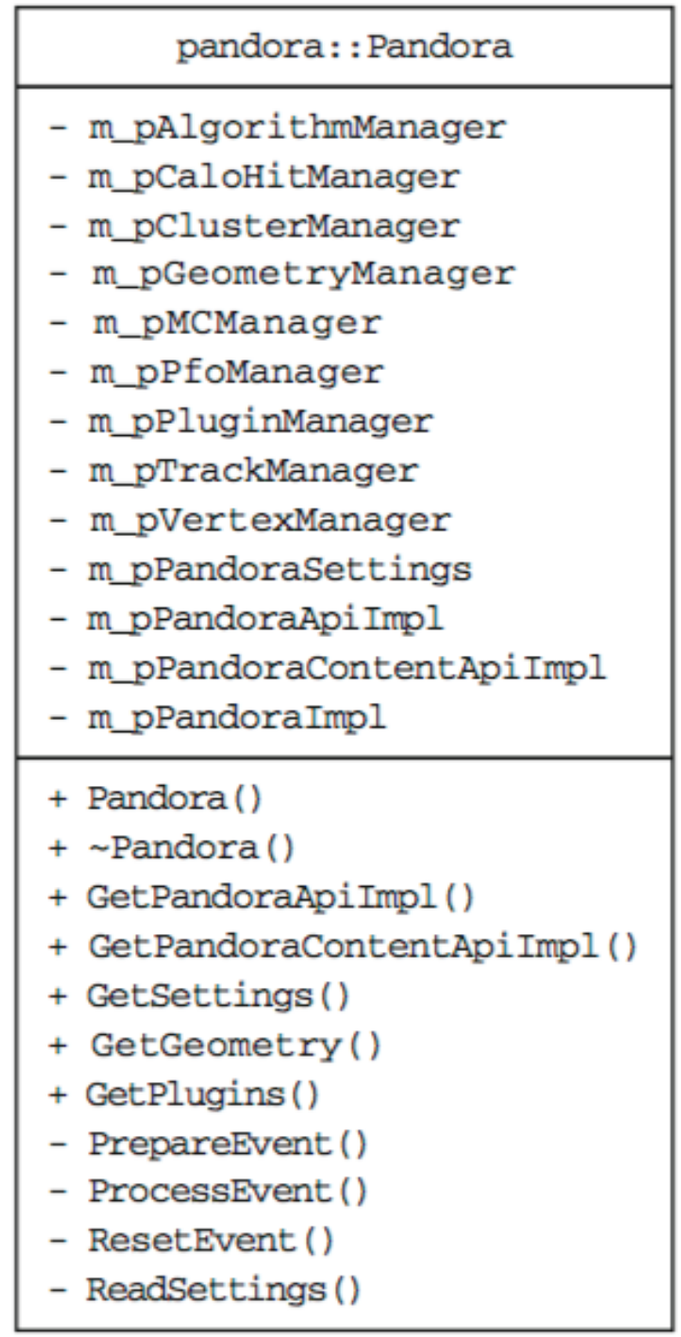
Built with client app, or in libraries of Pandora content

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 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 std::lists and so hold content orderings reflecting orders in which they were populated, via a client app or algorithm.
- 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

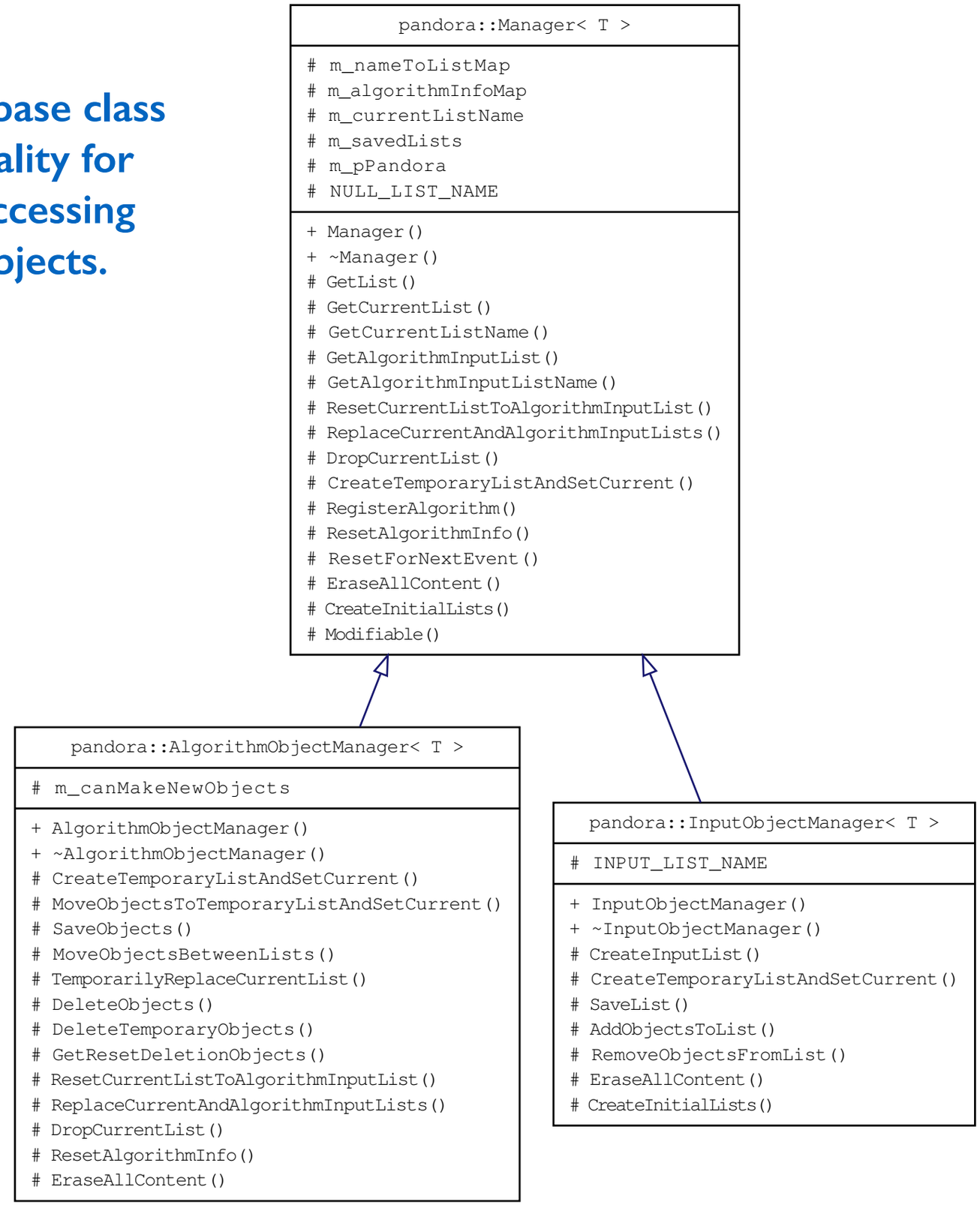
```
/**
 *  @brief  Get the current list
 *
 *  @param  algorithm the algorithm calling this function
 *  @param  pT to receive the address of the current list
 *  @param  listName to receive the current list name
 */
template <typename T>
static pandora::StatusCode GetCurrentList(const pandora::Algorithm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

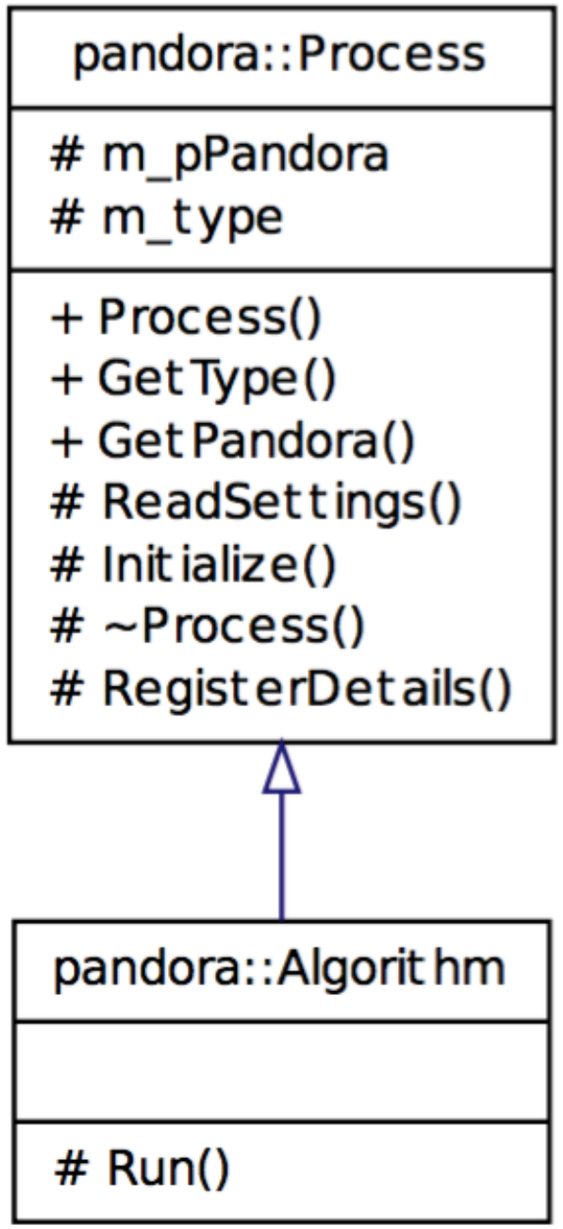
Algs can use APIs to ask whether objects are available or have already been used

PandoraContentApi.h

```
/**
 *  @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 child XML tags, or can search for optional tags to override default parameter values, if present.

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 child 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 child algorithm instance and return the unique name of the child algorithm.
- During event processing the parent algorithm can use an API to ask to run the child algorithm with the stored unique name.

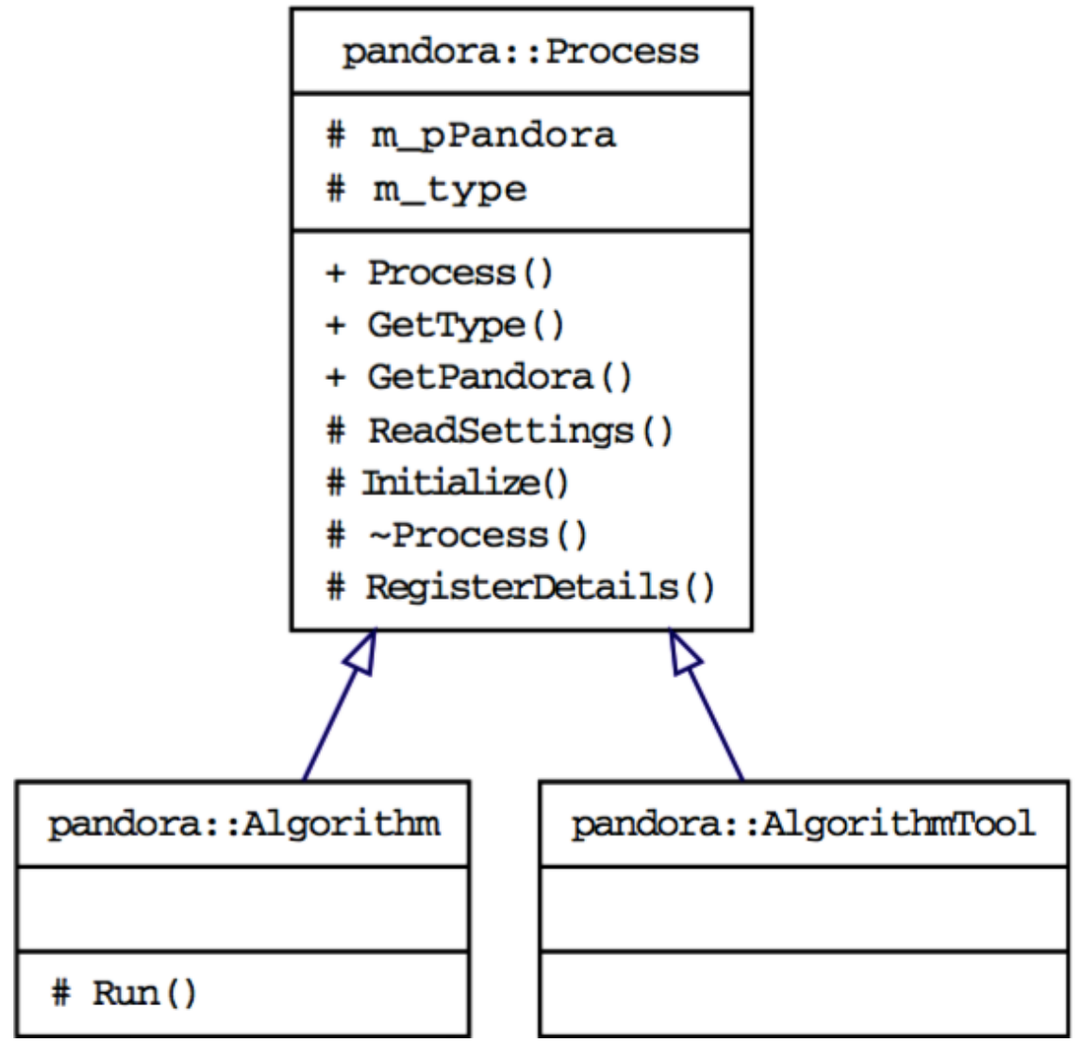
Nesting allows parent alg to e.g. manipulate current object lists, then call reusable child algs to process list contents

child alg

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

Algorithm Tools

- **Child alg functionality promotes the development of small, reusable algs to perform specific operations.**
- Parent and child 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.

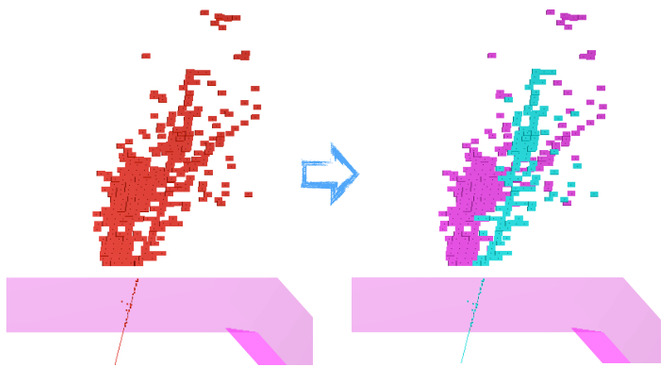
```
1:  procedure CLUSTER CREATION
2:      Create temporary Cluster list
3:      Get current CaloHit list
4:      for all CaloHits do
5:          if CaloHit available then
6:              for all newly-created Clusters do
7:                  Find best host Cluster
8:              if Suitable host Cluster 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.

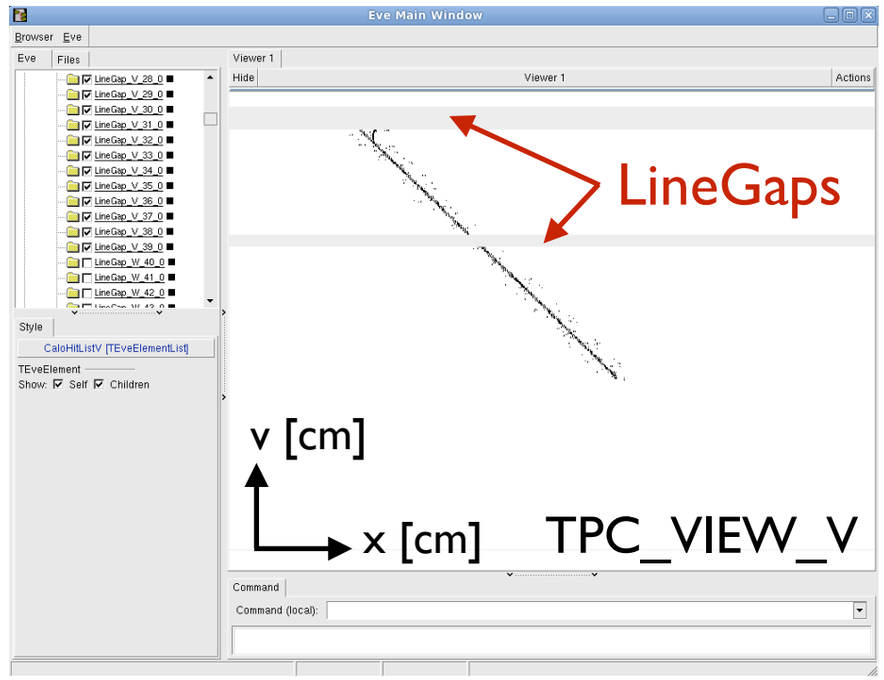


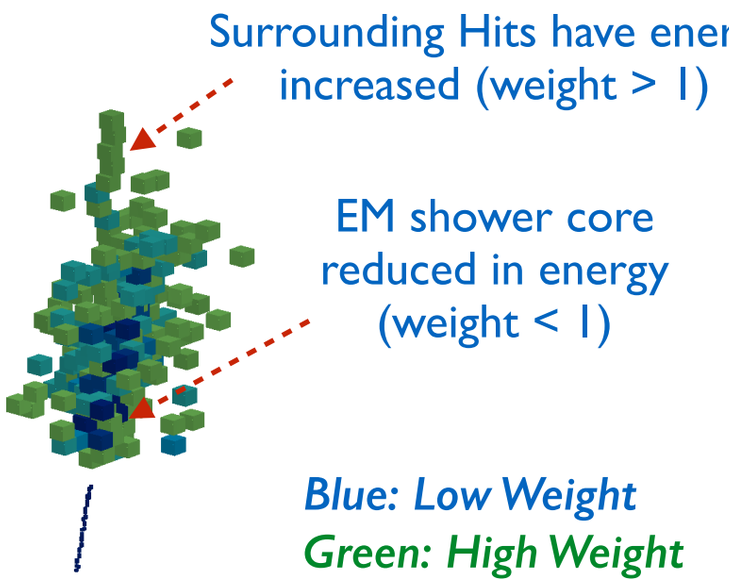
+ *Local* reclustering allows direct comparison of two Cluster configurations within single alg

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 child 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

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.



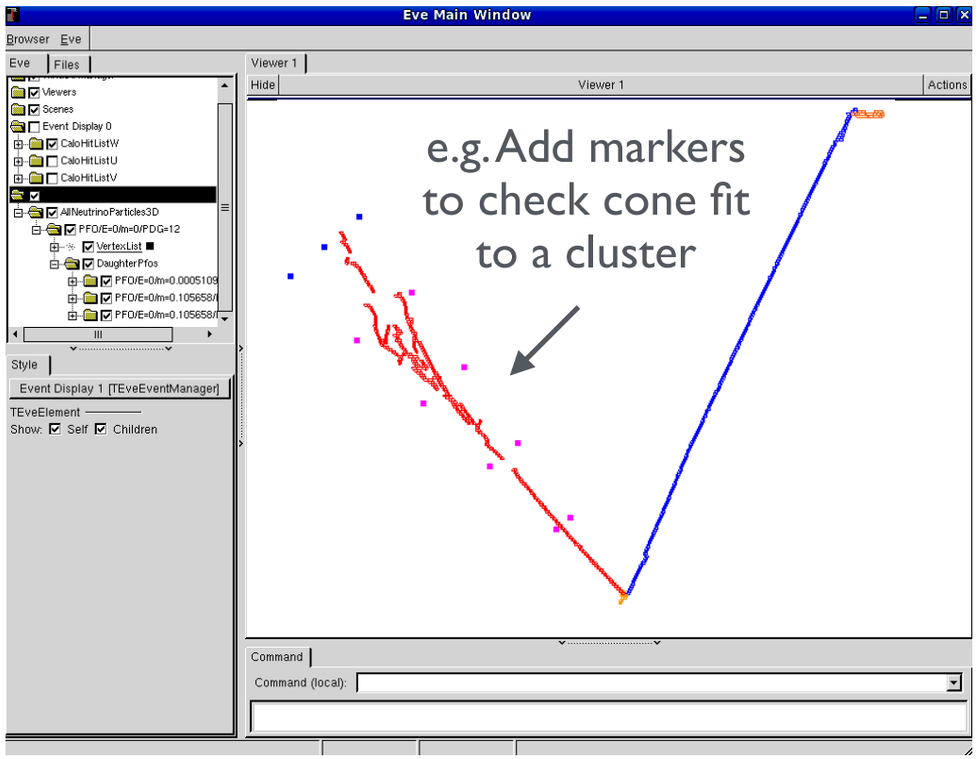


E.g. Novel hadronic energy estimator at ILC

- **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 controlling usage of ROOT.




```
. . .
<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

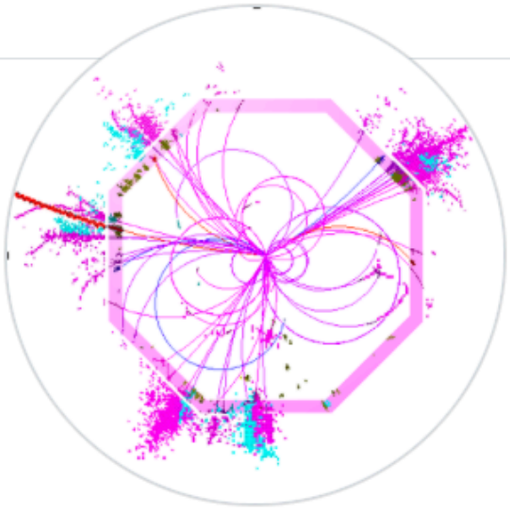
- **Pandora persistency allows Input Objects to be serialised in .pndr files (small, portability not guaranteed) or .xml files (large, but compressible).**
  - No longer need full client/translation app to develop or test algs: can move to lightweight environment where Entry Point constructs Pandora 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>
```

https://github.com/PandoraPFA



Multi-algorithm pattern recognition

PandoraPFA

Follow

53 followers · 1 following

Overview | Repositories 17 | Projects | Packages | Stars

Pinned

Documentation Public
Useful documents describing the Pandora project
Python ☆ 2 ⑂ 8

PandoraSDK Public — Eur. Phys. J. C 75, 439 (2015)
Powerful Software Development Kit for pattern recognition algorithms
C++ ☆ 4 ⑂ 29

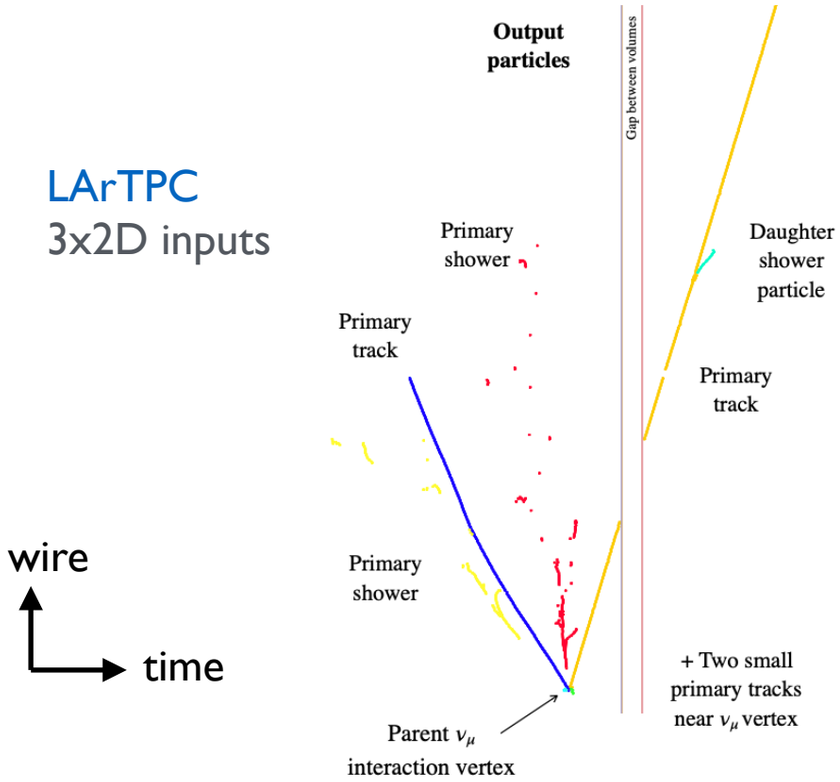
PandoraMonitoring Public
ROOT-based Event Visualisation Environment and tree-writing functionality
C++ ☆ 3 ⑂ 14

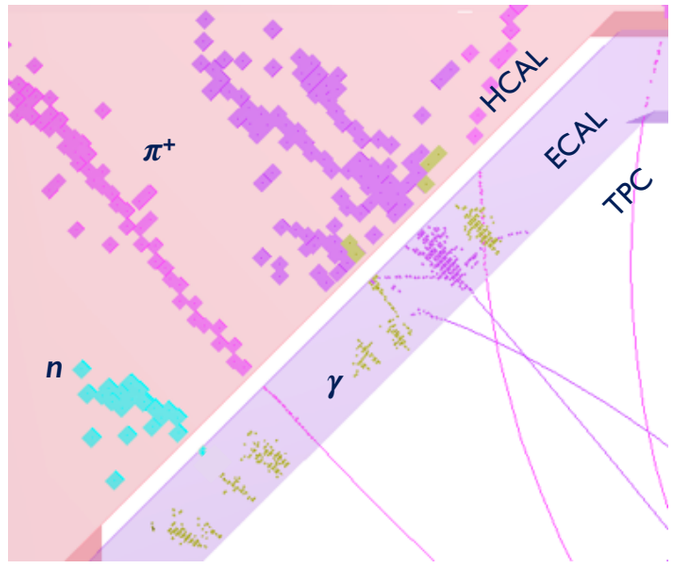
ExampleContent Public
Algorithms and tools for reconstruction in a simple learning / test environment
C++ ⑂ 4

LArContent Public — Eur. Phys. J. C 78, 82 (2018); ProtoDUNE-SP paper soon
Algorithms and tools for LAr TPC event reconstruction
C++ ☆ 4 ⑂ 41

LCContent Public — Nucl. Instrum. Meth. A611, 25 (2009); Nucl. Instrum. Meth. A700, 153 (2013)
Algorithms and tools for LC event reconstruction
C++ ⑂ 9

LArTPC
3x2D inputs





ILC/CLIC
3D inputs

Questions or comments?

WARWICK
THE UNIVERSITY OF WARWICK

Pandora Client App

John Marshall for the Pandora Team

22nd June 2022

Create Pandora Instance

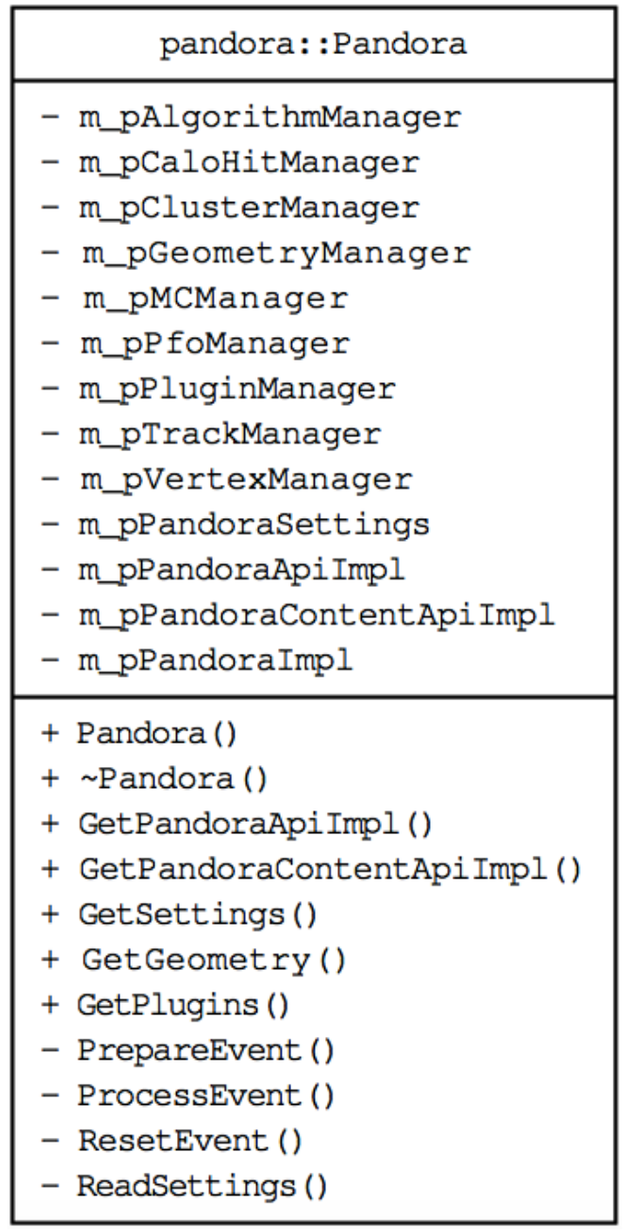
```
/**
 *    @brief Pandora class
 */
class Pandora
{
public:
    /**
     *  @brief  Default constructor
     */
    Pandora();

    ...
};
```

Pandora.h

In client app:

```
const pandora::Pandora *const pPandora = new pandora::Pandora();
```

- **Simple to create a Pandora instance (on stack or heap) via public default constructor.**
- **Will then find that its functionality is only available via its APIs, which are divided into:**
  i. **PandoraAPIs for use by a client app.**
  ii. **PandoraContentAPIs for use by algorithms.**



Member variables are addresses of Manager instances, API implementation instances and a Settings instance. Services are typically accessed via APIs.

Register Content

PandoraApi.h

```
/**
 *  @brief  Register an algorithm factory with pandora
 *
 *  @param  pandora the pandora instance to register the algorithm factory with
 *  @param  algorithmType the type of algorithm that the factory will create
 *  @param  pAlgorithmFactory the address of an algorithm factory instance
 */
static pandora::StatusCode RegisterAlgorithmFactory(const pandora::Pandora &pandora, const std::string &algorithmType,
    pandora::AlgorithmFactory *const pAlgorithmFactory);
```

API to register an algorithm factory, giving Pandora instance ability to instantiate a specific algorithm type

LArContent.h

```
/**
 *  @brief  Register all the lar content algorithms and tools with pandora
 *
 *  @param  pandora the pandora instance with which to register content
 */
static pandora::StatusCode RegisterAlgorithms(const pandora::Pandora &pandora);

/**
 *  @brief  Register lar coordinate transformation plugin with pandora
 *
 *  @param  pandora the pandora instance with which to register content
 *  @param  pLArTransformationPlugin the address of the lar transformation plugin
 */
static pandora::StatusCode SetLArTransformationPlugin(const pandora::Pandora &pandora,
    lar_content::LArTransformationPlugin *const pLArTransformationPlugin);
```

E.g. Quickly register all 80+ algorithm factories in the LAr TPC 'content' library

LArTransformationPlugin interface

```
/**
 *  @brief  Transform from (U,V) to W position
 *
 *  @param  U the U position
 *  @param  V the V position
 */
 virtual double UVtoW(const double u, const double v) const = 0;


/**
 *  @brief  Transform from (U,V) to world volume Y coordinate
 *
 *  @param  U the U position
 *  @param  V the V position
 */
virtual double UVtoY(const double u, const double v)  const = 0;

...
```

Note preprocessor macro checking API return values

In client app:

```
const pandora::Pandora *const pPandora = new pandora::Pandora();
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LArContent::RegisterAlgorithms(*pPandora));
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, LArContent::SetLArTransformationPlugin(*pPandora, new MicroBooNETransformationPlugin));
```

Read Pandora Settings

PandoraApi.h

```
/**
 *  @brief  Read pandora settings
 *
 *  @param  pandora the pandora instance to run the algorithms initialize
 *  @param  xmlFileName the name of the xml file containing the settings
 */
static pandora::StatusCode ReadSettings(const pandora::Pandora &pandora, const std::string &xmlFileName);
```

path to file describing Pandora reconstruction config.

In client app:

```
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::ReadSettings(*m_pPandora, configFileName));
```

- Specify which algorithms to instantiate and the order of algorithm execution.
- Parent algorithms can run (lists of) child algorithms, or (lists of) algorithm tools.

E.g. LArTPC 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>
```

Create Pandora Input

PandoraApi.h

```
/**
 *  @brief  Object creation helper class
 *
 *  @param  PARAMETERS the type of object parameters
 *  @param  OBJECT the type of object
 */
template <typename PARAMETERS, typename OBJECT>
class ObjectCreationHelper
{
public:
    typedef PARAMETERS Parameters;
    typedef OBJECT Object;

    /**
     *  @brief  Create a new object from a user factory
     *
     *  @param  pandora the pandora instance to create the new object
     *  @param  parameters the object parameters
     *  @param  factory the factory that performs the object allocation
     */
    static pandora::StatusCode Create(const pandora::Pandora &pandora, const Parameters &parameters,
        const pandora::ObjectFactory<Parameters, Object> &factory = pandora::PandoraObjectFactory<Parameters, Object>());
};

typedef ObjectCreationHelper<CaloHitParameters, pandora::CaloHit> CaloHit;
typedef ObjectCreationHelper<MCParticleParameters, pandora::MCParticle> MCParticle;
```

Advanced functionality: Can provide custom object instantiation factory to ‘decorate’ base objects in Pandora Event Data Model

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));
```

Create Pandora CaloHits

PandoraApi.h

```
/**
 *  @brief  CaloHitParameters class
 */
class CaloHitParameters : public pandora::ObjectParameters
{
public:
    pandora::InputCartesianVector   m_positionVector;           ///< Position vector of center of calorimeter cell, units mm
    pandora::InputCartesianVector   m_expectedDirection;        ///< Unit vector in direction of expected hit propagation
    pandora::InputCartesianVector   m_cellNormalVector;         ///< Unit normal to sampling layer, pointing outwards from the origin
    pandora::InputCellGeometry      m_cellGeometry;             ///< The cell geometry type, pointing or rectangular
    pandora::InputFloat             m_cellSize0;                ///< Cell size 0 [pointing: eta, rect: up in ENDCAP, along beam in BARREL, units mm]
    pandora::InputFloat             m_cellSize1;                ///< Cell size 1 [pointing: phi, rect: perp. to size 0 and thickness, units mm]
    pandora::InputFloat             m_cellThickness;            ///< Cell thickness, units mm
    pandora::InputFloat             m_nCellRadiationLengths;    ///< Absorber material in front of cell, units radiation lengths
    pandora::InputFloat             m_nCellInteractionLengths;  ///< Absorber material in front of cell, units interaction lengths
    pandora::InputFloat             m_time;                     ///< Time of (earliest) energy deposition in this cell, units ns
    pandora::InputFloat             m_inputEnergy;              ///< Corrected energy of calorimeter cell in user framework, units GeV
    pandora::InputFloat             m_mipEquivalentEnergy;      ///< The calibrated mip equivalent energy, units mip
    pandora::InputFloat             m_electromagneticEnergy;    ///< The calibrated electromagnetic energy measure, units GeV
    pandora::InputFloat             m_hadronicEnergy;           ///< The calibrated hadronic energy measure, units GeV
    pandora::InputBool              m_isDigital;                ///< Whether cell should be treated as digital
    pandora::InputHitType           m_hitType;                  ///< The type of calorimeter hit
    pandora::InputHitRegion         m_hitRegion;                ///< Region of the detector in which the calo hit is located
    pandora::InputUInt              m_layer;                    ///< The subdetector readout layer number
    pandora::InputBool              m_isInOuterSamplingLayer;   ///< Whether cell is in one of the outermost detector sampling layers
    pandora::InputAddress           m_pParentAddress;           ///< Address of the parent calo hit in the user framework
};
```

InputTypes template checks assignment operator is used, plus vetoes NaN and INF assignments

- **List of variables to which client app must assign before requesting CaloHit creation.**
- Still oriented towards collider experiments: have a derived class to ‘decorate’ with LAr-specific properties. Information available to algs, but doesn’t mean any/all properties *need* to be used.
- Algorithms can access information stored in Hits, but do not need to know how properties were obtained: client application isolates algorithms from input software framework.

Create Pandora MCParticles

PandoraApi.h

```
/**
 *  @brief  MCParticleParameters class
 */
class MCParticleParameters : public pandora::ObjectParameters
{
public:
    pandora::InputFloat             m_energy;               ///< The energy of the MC particle, units GeV
    pandora::InputCartesianVector   m_momentum;             ///< The momentum of the MC particle, units GeV
    pandora::InputCartesianVector   m_vertex;               ///< The production vertex of the MC particle, units mm
    pandora::InputCartesianVector   m_endpoint;             ///< The endpoint of the MC particle, units mm
    pandora::InputInt               m_particleId;           ///< The MC particle's ID (PDG code)
    pandora::InputMCParticleType    m_mcParticleType;       ///< The type of mc particle, e.g. vertex, 2D-projection, etc.
    pandora::InputAddress           m_pParentAddress;       ///< Address of the parent MC particle in the user framework
};
```

Properties that must be provided before MCParticle creation can be requested

```
/**
 *  @brief  Set parent-daughter mc particle relationship
 *
 *  @param  pandora the pandora instance to register the relationship with
 *  @param  pParentAddress address of parent mc particle in the user framework
 *  @param  pDaughterAddress address of daughter mc particle in the user framework
 */
static pandora::StatusCode SetMCParentDaughterRelationship(const pandora::Pandora &pandora, const void *const pParentAddress,
    const void *const pDaughterAddress);
```

Set parent-child relationships to full describe MCParticle hierarchy in Pandora

```
/**
 *  @brief  Set calo hit to mc particle relationship
 *
 *  @param  pandora the pandora instance to register the relationship with
 *  @param  pCaloHitParentAddress address of calo hit in the user framework
 *  @param  pMCParticleParentAddress address of mc particle in the user framework
 *  @param  mcParticleWeight weighting to assign to the mc particle
 */
static pandora::StatusCode SetCaloHitToMCParticleRelationship(const pandora::Pandora &pandora, const void *const pCaloHitParentAddress,
    const void *const pMCParticleParentAddress, const float mcParticleWeight = 1);
```

Set (custom/energy-weighted) relationships between Hits and MCParticles in Pandora

Run Pandora Algorithms

PandoraApi.h

```
/**
 *  @brief  Process an event
 *
 *  @param  pandora the pandora instance to process event
 */
static pandora::StatusCode ProcessEvent(const pandora::Pandora &pandora);
```

In client app:

```
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::ProcessEvent(*pPandora));
```

- **Pass thread to a Pandora instance, which will process the event by running the algorithms as specified in the Pandora Settings XML file.**
- **Algorithms will form Clusters, Vertices and Particles to represent the pattern-recognition solution. The thread will then be returned for output to be persisted.**

Extract Pandora Output

PandoraApi.h

```
/**
 *  @brief  Get the current pfo list
 *
 *  @param  pandora the pandora instance to get the objects from
 *  @param  pPfoList to receive the address of the particle flow objects
 */
static pandora::StatusCode GetCurrentPfoList(const pandora::Pandora &pandora, const pandora::PfoList *&pPfoList);
```

In client app:

```
const pandora::PfoList *pPfoList(nullptr);
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::GetCurrentPfoList(*pPandora, pPfoList));
```

- **Access list of Particles as specified/selected by final algorithm, and designated to be the 'current' list. Particles may be organised into a hierarchy.**
- **From Particles, can navigate to constituent Clusters, (Tracks,) Vertices and CaloHits. Can use ParentAddresses in Pandora objects to identify associated inputs.**

Reset Pandora

PandoraApi.h

```
/**
 *  @brief  Reset pandora to process another event
 *
 *  @param  pandora the pandora instance to reset
 */
static pandora::StatusCode Reset(const pandora::Pandora &pandora);
```

In client app:

```
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::Reset(*pPandora));
```

- **Ask to reset a Pandora instance, deleting all objects and lists made by algorithms and all input building-blocks provided by the client application.**
- **Pandora instance is then ready to begin receiving new Hits, (Tracks) and MCParticles to describe the next input event.**

Questions or comments?

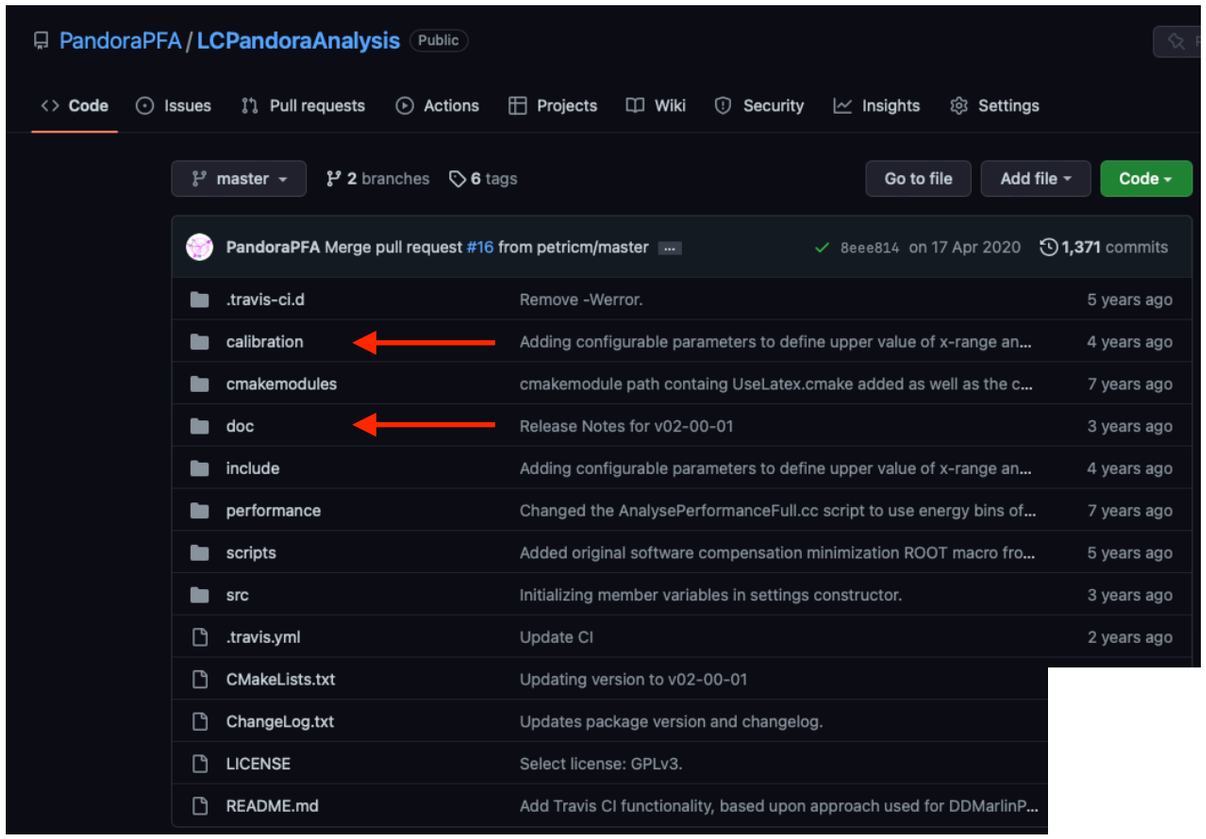
WARWICK
THE UNIVERSITY OF WARWICK

Pandora ILD Calibration

John Marshall for the Pandora Team

22nd June 2022

https://github.com/PandoraPFA/LCPandoraAnalysis



Pandora Analysis Calibration Executables

Steve Green

Abstract

The calibration procedure required for simulation and reconstruction of high energy physics events at a future electron positron collider using the particle flow algorithm PandoraPFA is described in full. The calibration procedure addresses how to set the constants used in the simulation for both digitisation, the estimation of energy deposits in the absorber material of the sampling calorimeters based on the energy deposits in the active material, and those used within PandoraPFA to set the hadronic and electromagnetic energy scales for the various particle showers being reconstructed. To be assured of accurate reconstruction, recalibration must be performed for all changes to detector design and/or reconstruction logic in PandoraPFA.

These studies were developed in the context of simulation of ILD (ILD_o1_v06) using Mokka for the detector simulation. This procedure can be adapted to other contexts as fundamentally the logic applied is universal to simulation of sampling calorimeters.