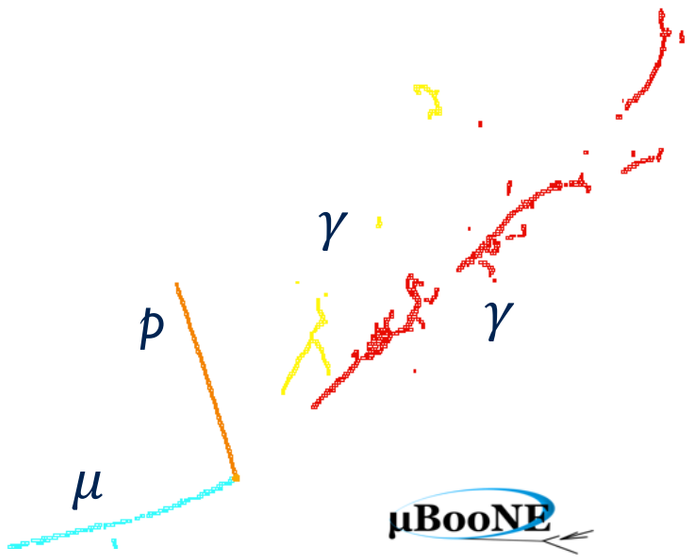


Lessons Learned from Collaborative Software Development using Pandora

J.S. Marshall for The Pandora Team

LArSoft Workshop

22 June 2016

Overview

Aim: Describe development practices that seem to have worked well for the Pandora multi-algorithm pattern recognition project

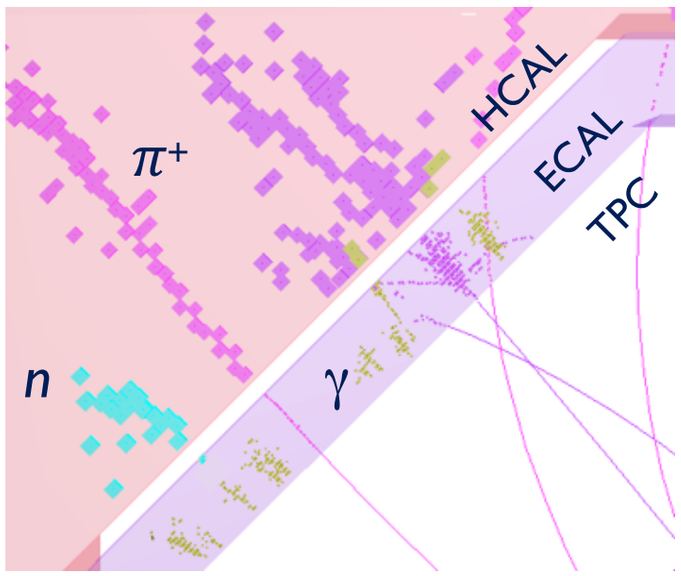
- **Pandora Goals and Architecture**
- **Pandora Developer Feedback**
- **Things that have worked well for:**
  1. **Designing and Implementing Algorithms**
  2. **Testing Algorithms**
- **Things that haven't worked so well**
- **Possible lessons for LArSoft**

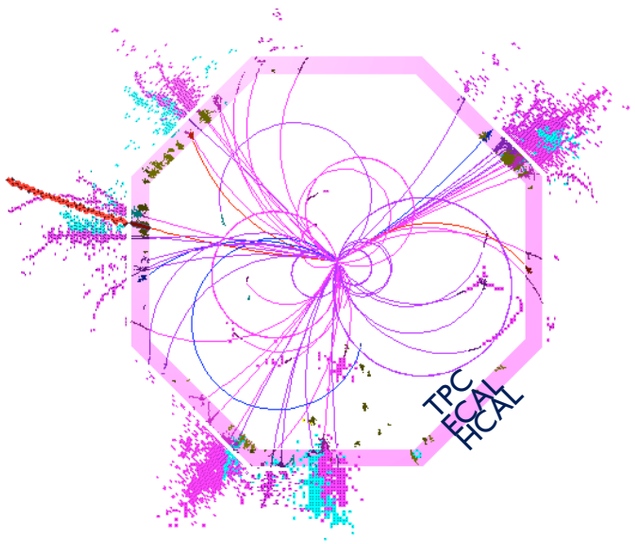
Absolutely not trying to claim any Pandora practices are perfect - we are still learning!

What Pandora is

- **Multi-algorithm approach to pattern recognition uses large numbers (80+) of algorithms, each designed to address specific topologies. Gradually build-up picture of events.**
- **This approach can be difficult to implement, so the Pandora Software Development Kit was designed to provide a robust, but user-friendly environment in which:**
  1. It is easy to provide the building-blocks that define a pattern recognition problem.
  2. Logic required to solve pattern recognition problems is cleanly implemented in algorithms.
  3. Operations to access or modify building-blocks requested by algs, performed by Pandora.

Typical ILC event topologies:





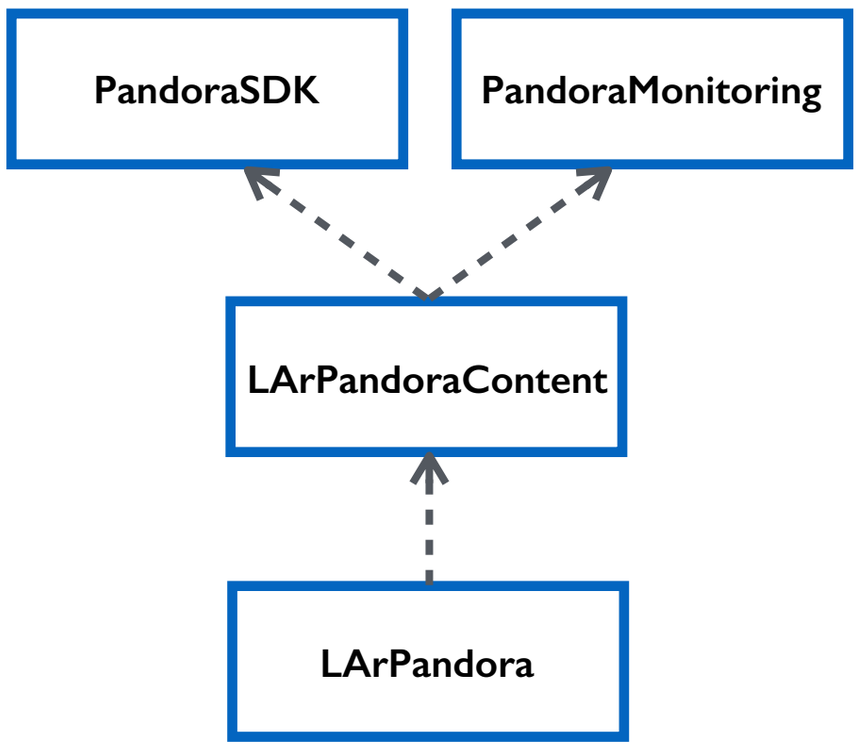
NIMA.2009.09.009, NIMA.2012.10.038, EPJC 75:439

- Great success with particle flow reconstruction in fine granularity detectors proposed for use at ILC.
- Quite a small project, attracting some success and interest by trying hard to do software "right".
- Significant effort to adopt proper software-engineering practices.

What Pandora is (not)

- **Pandora is *not* an alternative to (or replacement for) LArSoft. It is *not* a general purpose software framework to support diverse tasks through experiment life cycle.**
- Instead, it tries to make multi-algorithm event reconstruction easier to design, implement, test and maintain. It takes care of most memory-management, provides visual debugging, etc.
- **Example usage:** Try to implement pattern recognition in producer module. Process starts to involve multiple steps or maybe a recursive approach or reclustering would be helpful…
  - Implementation quickly becomes complex and likely reinvents wheel: **Pandora can help!**

Pandora integration with LArSoft

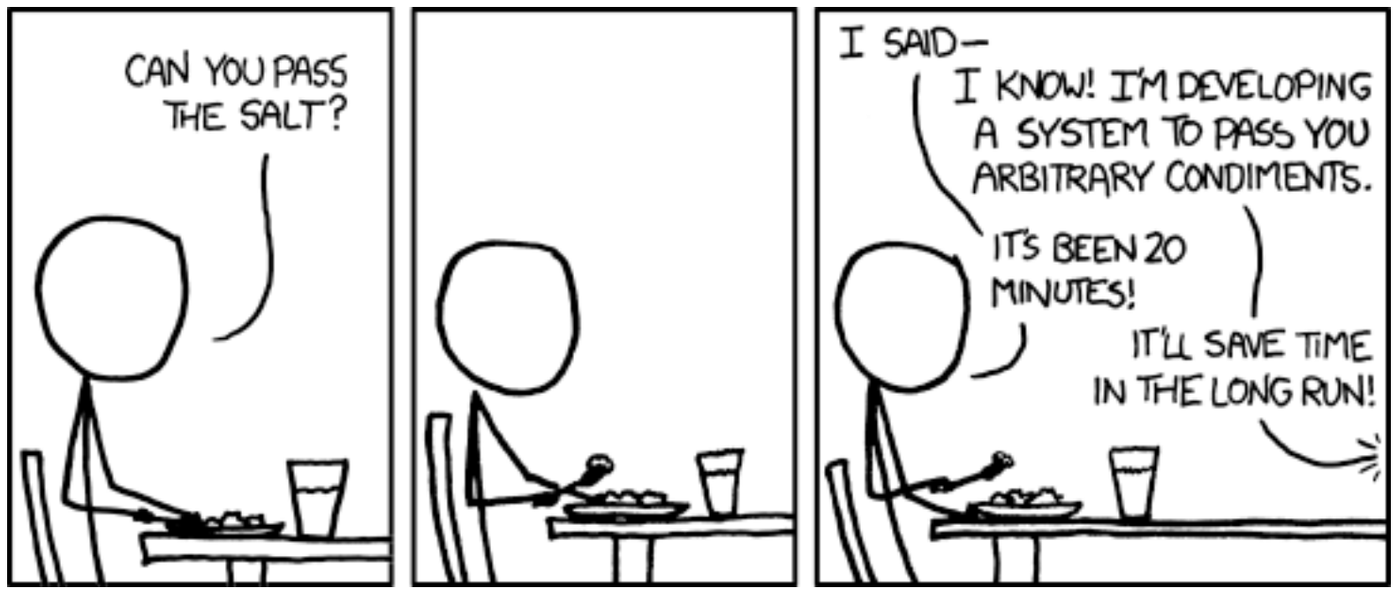


Re-usable libraries with APIs, support multi-alg approach

80+ algorithms and tools, specifically for LAr TPC usage

Producer module, provides translation LArSoft ⟷ Pandora

Pandora SDK Design



http://xkcd.com/974/

Pandora APIs

- **A client, or translation application is responsible for controlling pattern recognition: it creates the Pandora instance(s) and uses the Pandora APIs to request services.**
- It registers algorithm factories, giving Pandora instances the ability to instantiate specific algorithms, and it provides the algorithm configuration via an XML file.
- Each event, it asks Pandora to create the input 'building blocks' (e.g. Hits and, optionally, MCParticles) to describe an event and it receives the final output Particles.

- To create an input building block, must provide a list of simple parameters: energy, position, etc.
- Algorithms access information stored in building blocks but do not need to know how information was obtained.
- **Translation app isolates Pandora algorithms from host framework.**

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

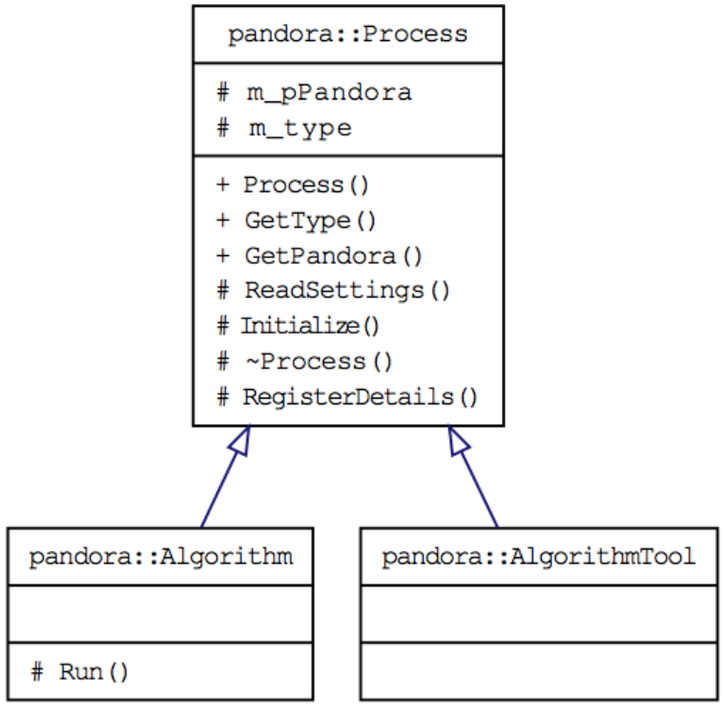
```
1:  procedure MAIN
2:      Create a Pandora instance
3:      Register Algorithms and Plugins
4:      Ask Pandora to parse XML settings file
5:      for all Events do
6:          Create CaloHit instances
7:          Create MCParticle instances
8:          Specify MCParticle-CaloHit relationships
9: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

Pandora APIs - Comments

- **Getting interface right for Pandora input/output vital to allow re-use of functionality:**
  - Translation app for each use-case tightly focused on, i), collecting information to fully define pattern recognition input (operations may be detector specific) and, ii), processing output.
  - Allowed LC algorithms to be re-used successfully for CMS HGCAL studies; just needed a new Pandora translation app to provide self-describing Hits, Tracks, etc.
  - Ensures Pandora SDK (and Monitoring) can be re-used to solve very different problems in HEP. Current major use cases: ILC / CLIC (inc. CMS) and LAr TPC reconstruction.

Pandora Content APIs

- Pandora algorithms contain the step-by-step instructions for finding patterns in the provided data: provide the "brain power".
- Algorithms can use APIs to access Pandora "content" and to ask Pandora to make new objects or modify existing objects.
- **Each API implementation fully tested** so that it can be used with total confidence. Exception handling ensures robustness.



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

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

Pandora Content APIs - Comments

- **Getting interface right for accessing/manipulating instances of objects in Pandora Event Data Model was vital to make multi-algorithm approach a reality:**
  - Algorithms structured around key operations and can be written in simple pseudo-code form. Common algorithms, with associated design patterns, include:
    - `Creating Clusters (containers of Hits)`
    - `Refining Clusters (Cluster merging or Cluster splitting)`
    - `Creating Particles (containers of Clusters, Tracks and Vertices)`
    - `Refining Particles (Particle merging or Particle splitting)`
  - Algorithms can then focus on providing key Boolean logic to drive operations; logic is typically determined by investigating event topology.
  - As "non-const" operations can only be requested via APIs, Pandora is able to perform memory management and book-keeping for changes to underlying objects (non trivial).
  - **Proves effective at concentrating developer thoughts/ideas. Algorithm implementation can remain rather clean, aiding readability and maintenance.**

Feedback from Developers

- **Some typical tasks for new developers:**
  - **Add a new algorithm -** Can quickly identify key algorithm operations (e.g. Cluster merging or splitting) and build algorithm structure around API calls. Free to focus on guiding logic.
  - **Optimise an existing algorithm -** Need only focus on algorithm guiding logic. Visualisation APIs allow for sophisticated visual debugging to understand treatment of individual events.

- **Asked developers to identify best and worst features of working with Pandora SDK:**

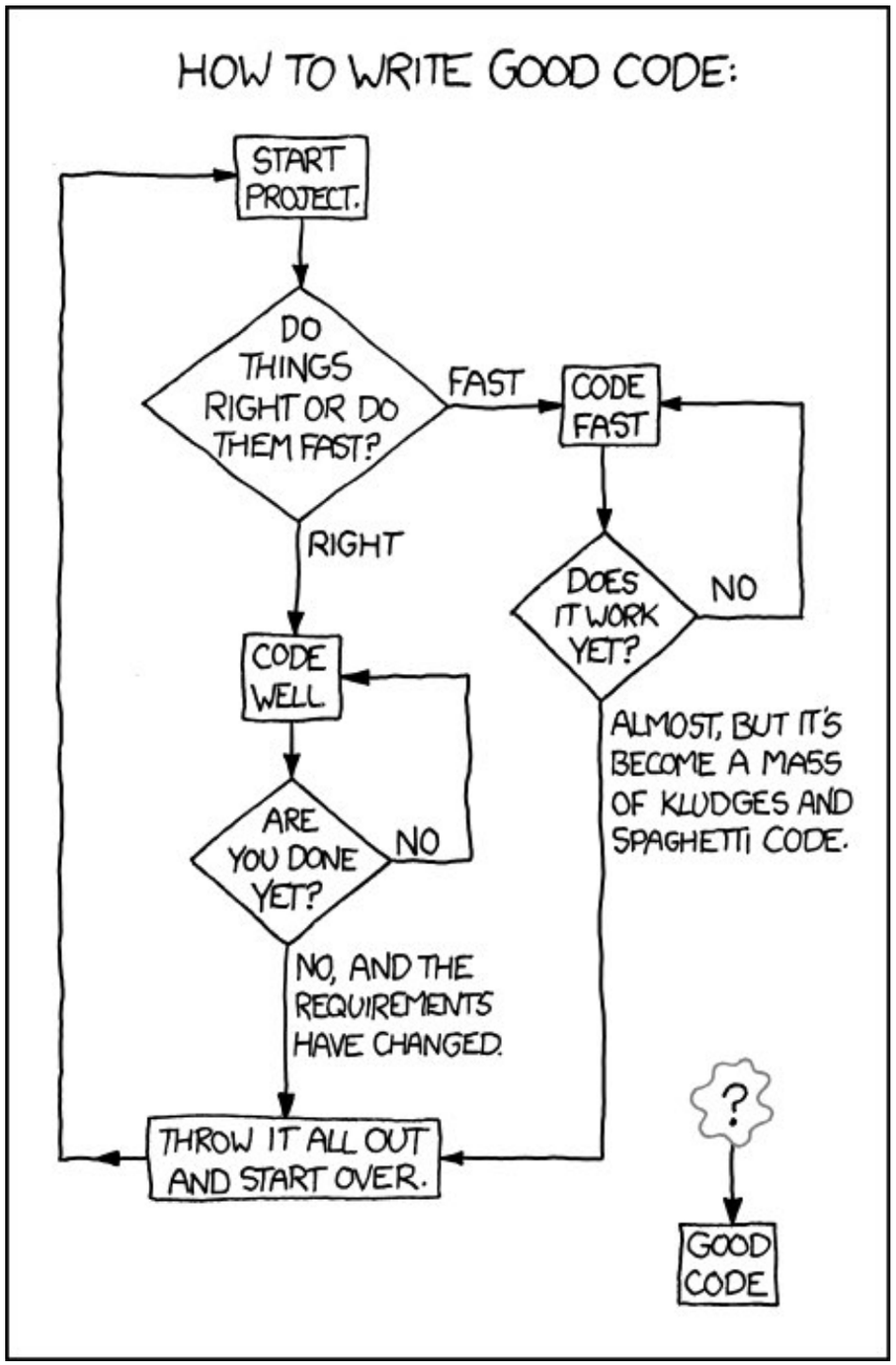
Best features

- Quick to try out new ideas.
- Visual debugging works really well.
- Standalone development environment.
- Simple XML-based configuration.
- Can re-use algorithms very easily.
- Can trust SDK services completely.

Worst features

- Sometimes non-obvious how to use available APIs to achieve goals.
- Adding new properties to objects in EDM is possible, but difficult.
- ROOT event display idiosyncrasies.
- Build system non-intuitive.

Algorithm Design and Implementation

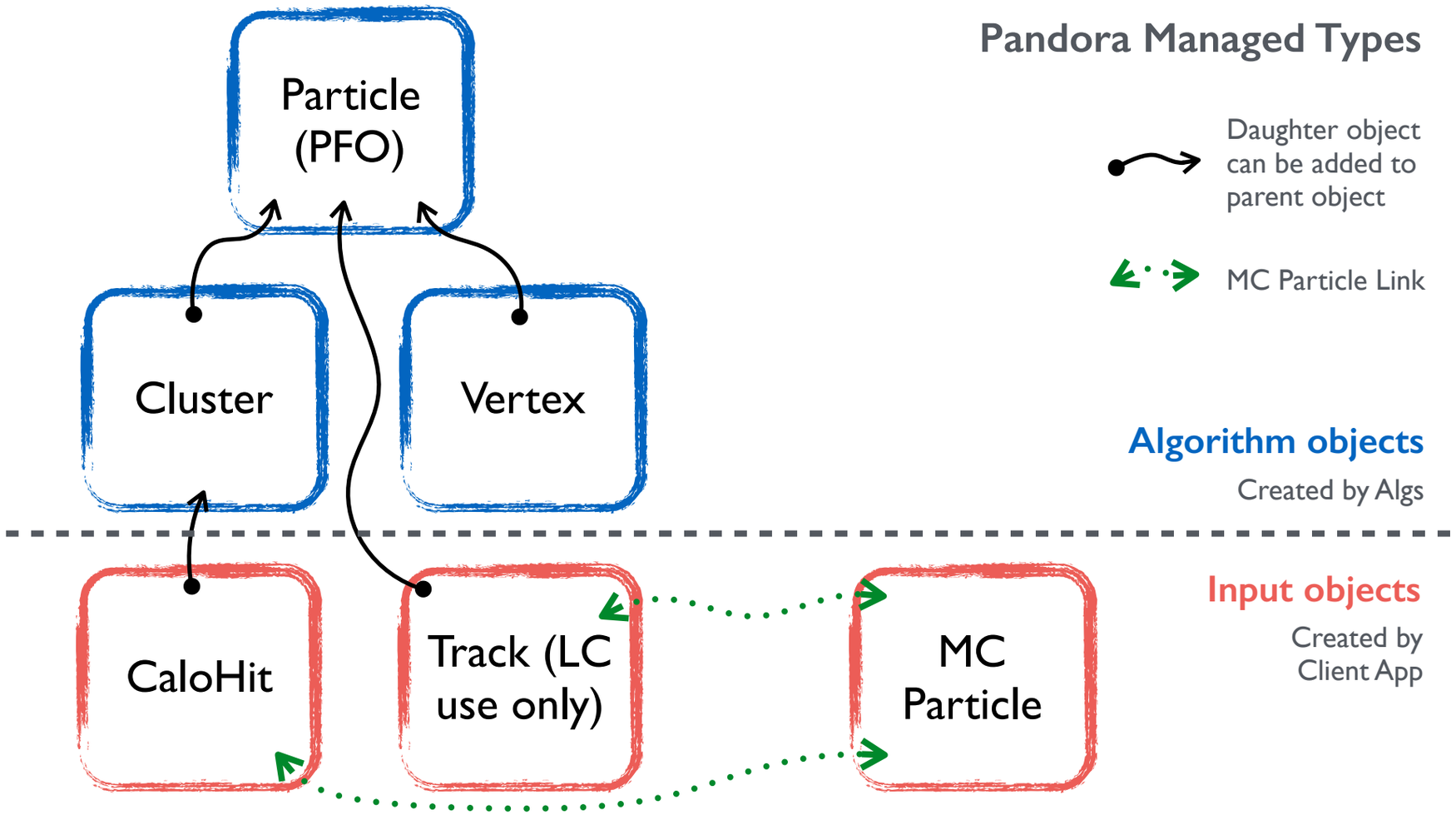


http://xkcd.com/844/

Event Data Model

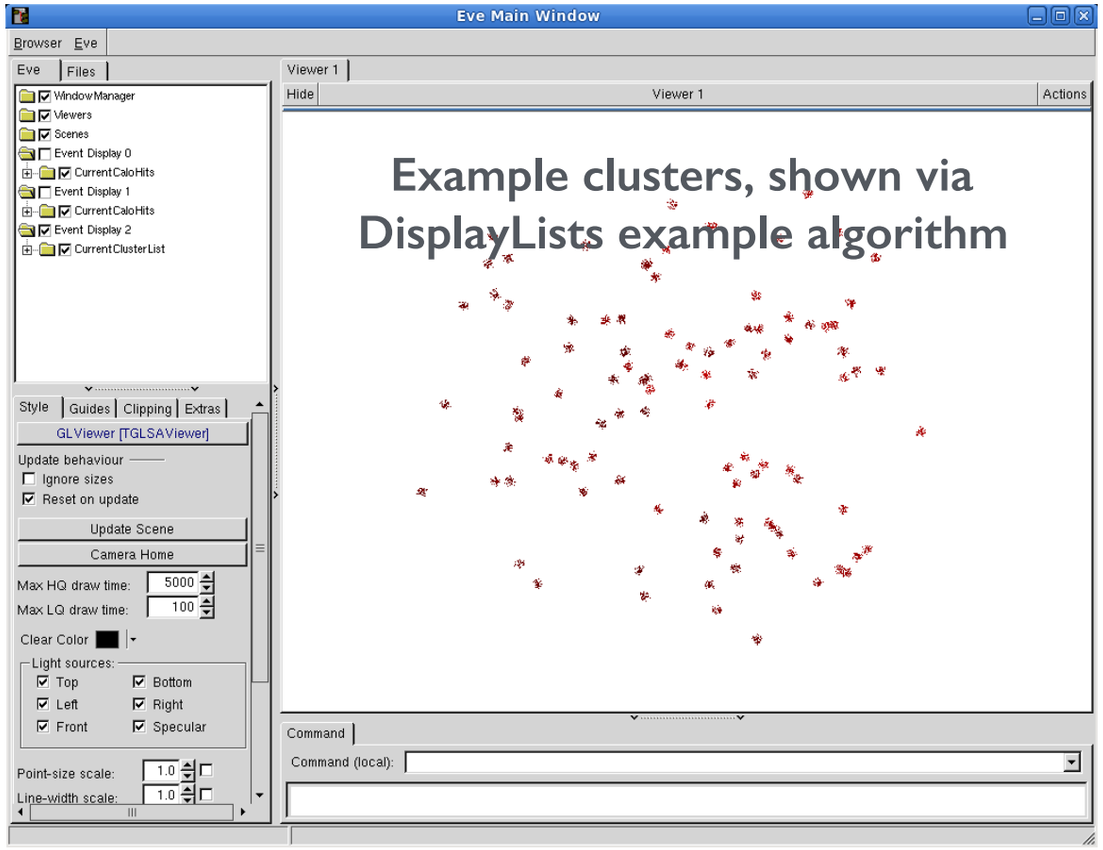
- **EDM is critically important: a set of classes representing the input building-blocks for a problem and the structures that can be created using these building blocks.**
  - **Successful EDM provides a well-defined development environment and allows for independence of the algorithms, which can only communicate via the EDM.**
  - **Constructs to which developers will be exposed on a daily basis: should be a "joy to use" and must facilitate simple and "clean" assessment of event topologies.**

- Aided by simple constructs to avoid re-inventing wheel:
  - Three-vectors, linear fits, etc.
- Re-usable Helper fns keep algs clean, avoid code repetition.



Developer Training

- Pandora algorithms create and/or modify Clusters, Vertices and Particles. Decisions whether to proceed with these operations can be complex and use-case specific.
  - Learning library aims to demonstrate/test key functionality in a simple use-case. The logic is deliberately trivial, but all API usage is present and explained.
  - Developers can be exposed to major operations and style-guide, with notes of explanation, before starting to write real algorithms: seems to work well.



- Library consists of example Algorithms, Tools, Plugins and Helper functions:
  - Example list access and display
  - Cluster, Vertex and Particle creation
  - Cluster merging, splitting, deletion, reclustering
  - Creating and saving new lists of objects
  - Using Algorithm Tools and Plugins.

https://github.com/PandoraPFA/Documentation

Communication

- **Pandora project is sufficiently small that it is possible to have one manager/librarian who can maintain an overview of the entire project and (try to) keep things on track.**
  - Widely recognised as a good idea for each package (library and interfaces) to designate a package manager with a sense of ownership/pride and detailed understanding.
  - Need to understand how their package appears to others who do not have intimate knowledge of inner workings. Police interface changes and dependencies.
  - Perform code reviews to help enforce consistency in design/implementation. Using git version control strongly promotes this work flow: Pull request ⇒ communicate!

Idealistic, but achievable: hope is for managers to develop a sense of pride and ownership. Can act as spokesperson for the package and must be included in all relevant discussions.

Algorithm Design

- **Must have basic knowledge of development environment and open-up communication:**
  - Work through core algorithm operations and rules using learning library.
  - Discuss algorithm aims with package manager to answer a number of key questions:
    - How/where the algorithm best-fits in to the existing multi-algorithm approach?
    - What are the inputs, outputs and key services to request from Pandora?
    - Does any re-usable infrastructure already exist for this type of algorithm?
  - As intended, should allow construction of skeleton algorithm where it remains to "just" provide the key Boolean logic to guide non-const operations on event objects.

- **Embrace object oriented approach:** existing algs typically compare e.g. Clusters and store all information in custom Association instances, whose `operator<` can identify best matches.
- **Embrace visual development:** allows design of topological selection cuts at point of usage, or can use tree-writing functionality to understand selection in context of full distributions.
- **Expect iterative improvement:** many algs start off trying to address too wide a range of topologies. Incrementally become a series of connected algs, or one alg using multiple tools.

Style Guide

- **Pandora has rigorous style "guide". Guides criticised for stifling creativity, but we don't agree: algorithms should be creative, but implementation should be clean and readable.**
  - Following discussion of interfaces and logic, in design phase, aim is that any/all developers should provide a near identical implementation.
  - Intention is to write self-describing code where every type and instance is well-named to help ensure readability. This is genuinely achievable!
  - Place aesthetics/readability and proof of correctness first, then use tools such as Coverity and Intel VTune Amplifier to check/optimise. Can later accept changes for e.g. speed reasons.

E.g. Type names start in upper case ◆ Variable names start in lower case ◆ Typedef template specialisations for readability, e.g. stl containers ◆ Prefix pointer variables with "p" ◆ Prefix member variables with "m_" ◆ Function names should indicate action(s) performed ◆ Refactor functions to aim for max. 7 ± 2 lines implementation ◆ Interfaces only in header files ◆ All implementation in source files ◆ Consider const correctness ◆ Provide Doxygen information ◆ Comments in implementation only for non-standard features, prefixed with ATTN, TODO, etc.

Algorithm Testing

http://xkcd.com/292/

Simple Configuration

- **XML-based configuration is ultimately limited, but is a standard/recognised tool and is extremely easy to read and understand. It is much liked by Pandora developers.**
  - By presenting each algorithm with control of its own XML parsing in its ReadSettings callback, can actually introduce some quite sophisticated behaviour.
  - XML structure suits nested Parent/Daughter algorithm use-cases, recursive approaches or use of algorithm tools to perform specific tasks.

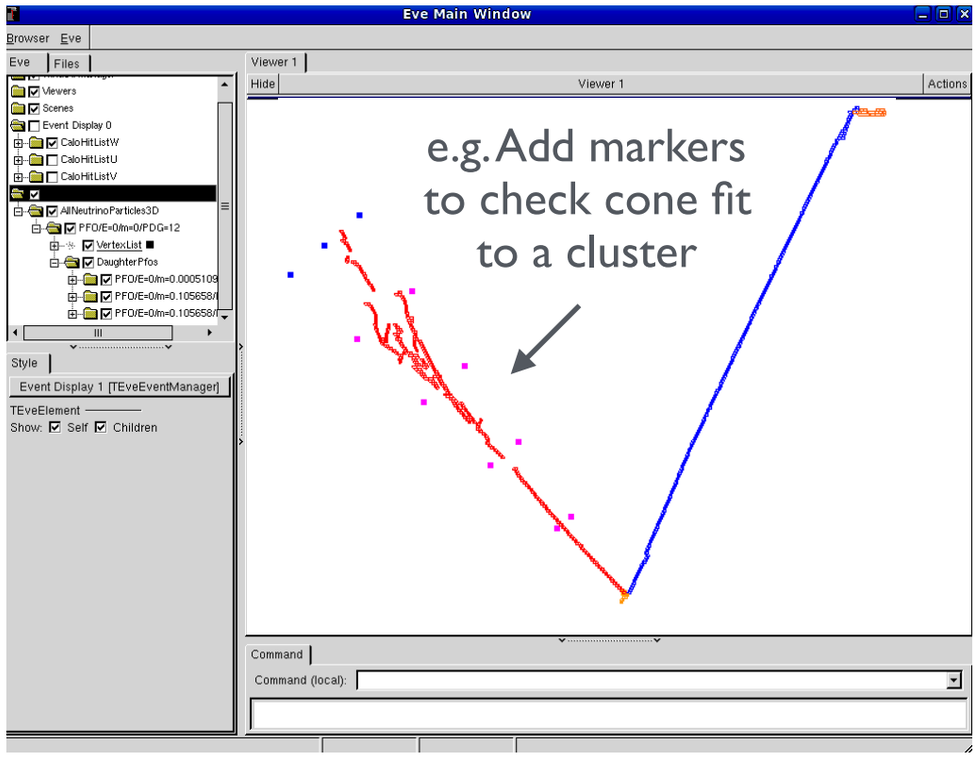
- XML-configured multi-algorithm chains currently available for LAr TPC reco:
  - Dedicated reco for **cosmic ray** muons
  - Dedicated reco for **neutrino** events
  - **Cheated** reco (development use only!)
  - Reco with 3D **event slicing;** then apply e.g. neutrino or CR reco to each slice.

Example XML snippet - 3D track reco →

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

Visual Debugging

- **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.
  - Extremely rewarding way to work: seeing a problem presented visually can often lead to proper understanding much more rapidly than print statements or debugger.




```
. . .
<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

Standalone Environment

- **Pandora persistency allows input objects (Hits, MCParticles, Gaps etc.) 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.
  - Simple Makefile option and command line app: in realm of standard, well documented C++

```
// 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>
```

Exception Handling

- **Pandora services aim to offer robust exception handling and reporting: idea is that operations should not proceed if input is invalid or unexpected situation arises.**
  - Better to refuse to return a value, *and say why*, than to return a nonsense (or supposedly safe) value that computer cannot intrinsically distinguish as incorrect.
  - SDK services try to check/pre-empt all possible ways operations could be going awry. Running a new alg for first time will likely bail out of operations and say why.
  - In debug mode, StatusCodeExceptions unwind stack at point of construction and store backtrace for user reference. Correct alg logic until all runs cleanly in soak testing.

- **Extremely effective because Pandora services play a key role in core alg operations:**
  - E.g. Prevent/flag re-use of Hit in a second Cluster,
  - Prevent split/delete of Cluster already in Particle.
- **Thought req'd:** return value checks vs. exceptions.
- **Speed implications:** avoid "routine exceptions".

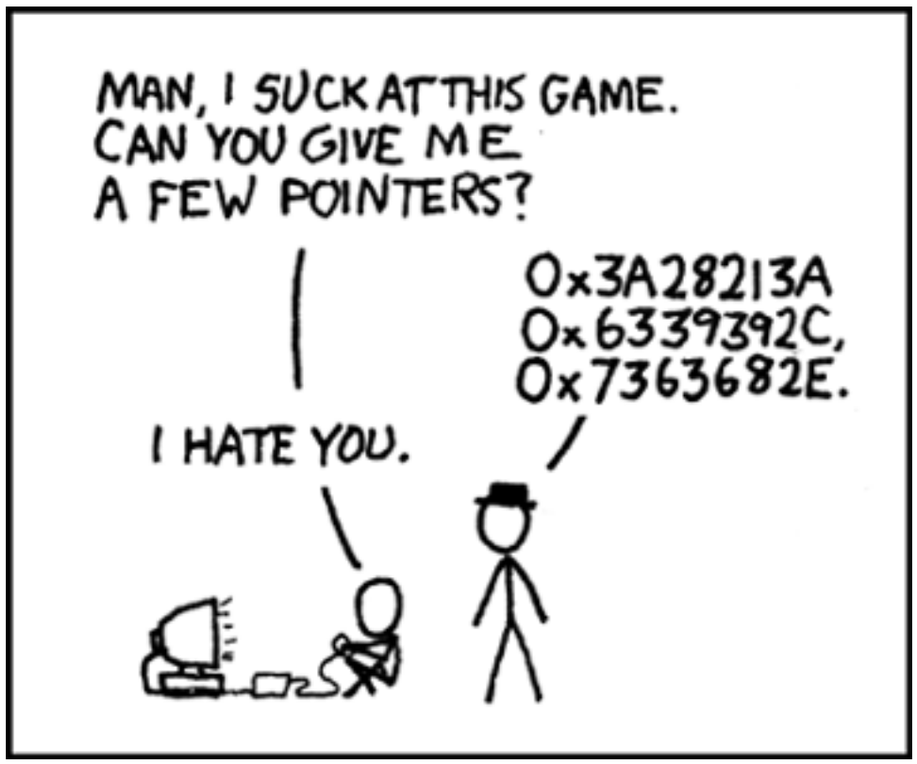
Example screen output:

```
Pandora::ReadSettings - Invalid xml file.
Failure in reading pandora settings, STATUS_CODE_FAILURE
PandoraApi::ReadSettings(*pPandora,
    parameters.m_pandoraSettingsFile)
 throw STATUS_CODE_FAILURE
 in function: main
 in file: /PATH/LArReco/test/PandoraInterface.cxx line#: 134
Pandora Exception caught: STATUS_CODE_FAILURE
```

Things that haven't worked so well

- **Ancillary components require significant attention too:**
  - Build mechanics originally inherited from ilcsoft. Still not found an elegant and robust system that works for everyone (difficulty of trying to exist in multiple software projects).
  - ROOT TEVE is a vital tool, but also source of nearly all difficulties with building or running Pandora. Ability to able to turn PandoraMonitoring/ROOT on/off at build-time helpful.
- **Attempts to provide globally reusable features:**
  - Truly generic geometry services extremely difficult. Current geometry still leans towards collider detector, with LAr TPC-specific geometry plugins built into LAr algorithm library.
  - Some properties of objects in the EDM arguably relevant only to collider use-case. Ultimately must prune generic objects, and decorate with specific LC and LAr properties.
  - Not managed to offer any truly generic algorithms, genuinely reusable by any client app.
- **Remember "second order" effects:**
  - Promotion of unordered_containers is fine, but means that developers must think hard about algorithm reproducibility. Problems arose (and were addressed) for LC and LAr reco.

Lessons Learned



http://xkcd.com/138/

Lessons Learned

- **Extremely simple message: The Pandora project has tried hard to adopt some widely recommended software engineering practices and believes they make a big difference!**
- **More difficult with wide user-base spread across the globe, but communication is vital:**
  - Package managers with "sense of ownership" maintain oversight and review pull requests.
  - Proper code review dialogue, with multiple iterations before merges to master branch.
- **Desirable to build applications where main thread uses clear services to achieve goals:**
  - Service interfaces are crucial; get these right and instils user confidence, improves user implementation and goes a long way to solving documentation and testing issues.
  - EDM should help to make development easier, and should have a simple/clean interface.
  - Well-maintained learning libraries can help to demonstrate usage of key services.

Thanks for your attention!

Contact details overleaf…

Pandora LAr TPC Reconstruction

Pandora is an open project and new contributors would be extremely welcome.
We'd love to hear from you and we will always try to answer your questions!

	Contact details:
Framework development	John Marshall (marshall@hep.phy.cam.ac.uk) Mark Thomson (thomson@hep.phy.cam.ac.uk)
LAr TPC algorithm development	John Marshall Andy Blake (a.blake@lancaster.ac.uk)
Performance metrics and validation	John Marshall Andy Blake Lorena Escudero (escudero@hep.phy.cam.ac.uk) Joris Jan de Vries (jjd49@hep.phy.cam.ac.uk) Jack Weston (weston@hep.phy.cam.ac.uk)

Please visit https://github.com/PandoraPFA

UNIVERSITY OF CAMBRIDGE

Lancaster University

Pandora in LArSoft

Place some of the more recent Pandora developments in a LArSoft context:

	LArSoft	Pandora SDK	Pandora Monitoring	Pandora Algorithms	
Production release	v04_25_00	v02_01_00	v02_01_00	v02_03_00	
	…	↓	↓	↓	← Vertex improvements
	v04_36_00	← Sync. with ILC versions →			
	v04_36_01			v02_04_00	
	…	↓	↓	↓	← Shower improvements
	v05_04_00	v02_02_00	v02_02_00	v02_05_00	
	…	↓ ← Add Gap Support →	↓	↓	← Vertex gap treatment
	v05_07_00	v02_03_00	v02_03_00	v02_06_00	
Production release	v05_08_00	↓	↓	v02_07_00	← Improved gap treatment
	…			↓	← Algorithm reproducibility
	v05_09_01			v02_07_01	
	…			↓	← Improved LArSoft integration
	v05_11_01	↓	↓	v02_07_03	