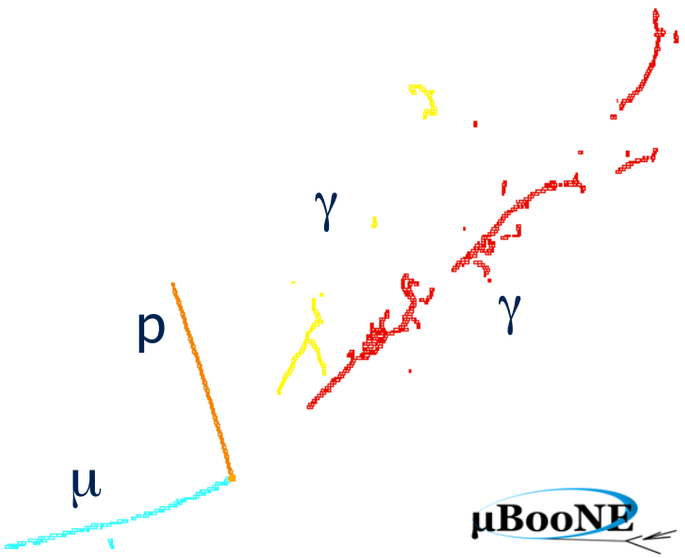


Pandora Talk 8: Output to LArSoft

L. Escudero for the Pandora Team

MicroBooNE Pandora Workshop

July 11-14th 2016, Cambridge

Pandora Output to LArSoft

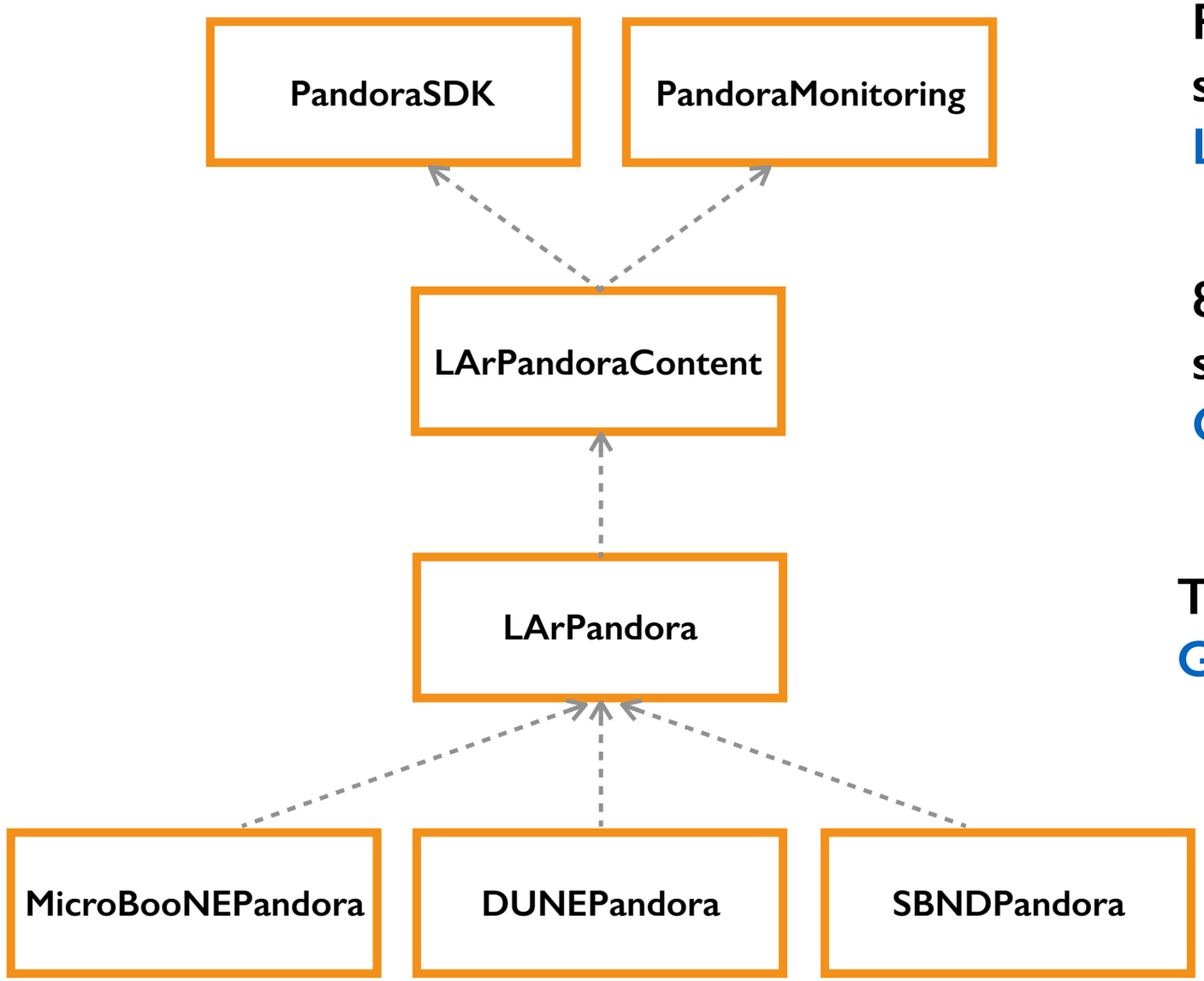
- Use the output of Pandora in LArSoft
  - Focus now on how to handle Pandora output in LArSoft
  - Discuss examples of static functions in larpandora/LArPandoraInterface/LArPandoraHelper
  - Preparation for the next exercise: write an analyser to analyse the output of your new algorithm

Disclaimer:

I am definitely not a LArSoft expert!

Pandora LArSoft Integration

Simple cartoon showing current packages and an indicative hierarchy:



Re-usable libraries and APIs, support multi-alg approach
LArSoft external product

80+ pattern recognition algs, specifically for LAr TPC usage
Git repo with Redmine remote

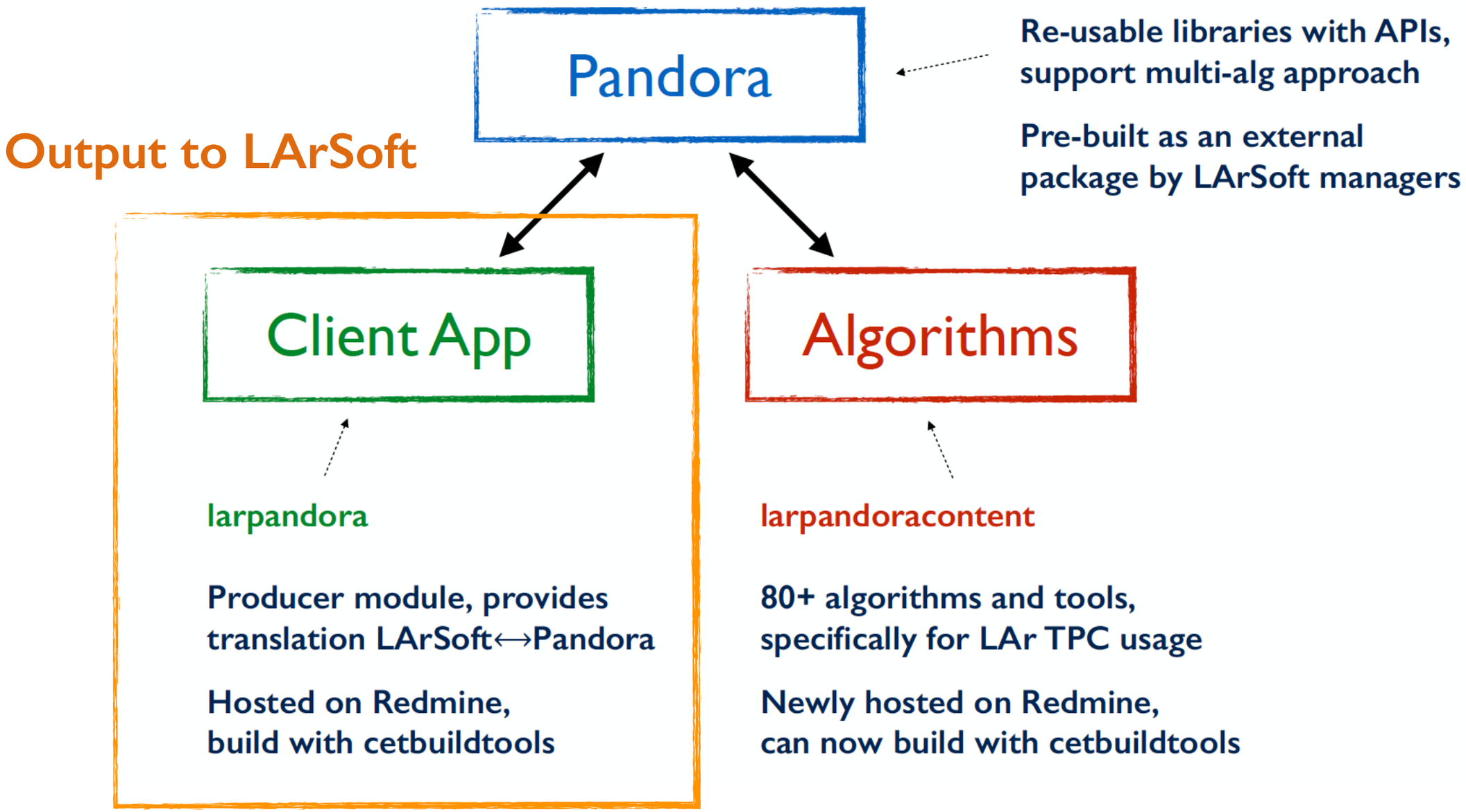
Translation LArSoft ⟷ Pandora
Git repo with Redmine remote

Any detector-specific aspects. Create Pandora Instances, define drift volumes
Git repos with Redmine remote

Ongoing: Replace detector-specific producer modules with a single, abstracted module.

Pandora LArSoft Integration

Or you prefer, another way to look at the LArSoft integration, from Talk 1

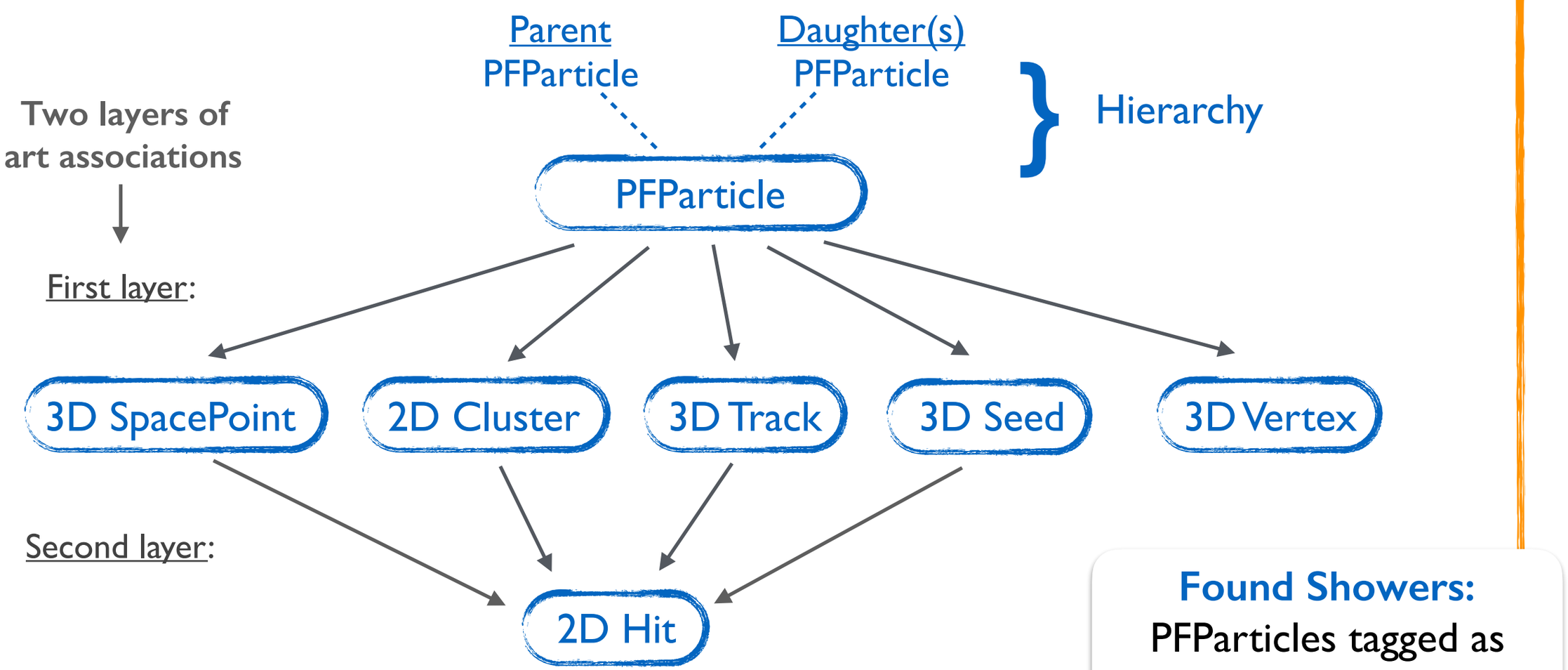


Pandora LArSoft Integration

Where are we?

	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	← Improved gap treatment
Production release	v05_08_00	↓	↓	v02_07_00	
	…			↓	← Algorithm reproducibility
	v05_09_01			v02_07_01	
	…			↓	← Improved LArSoft integration
	v05_11_01	↓	↓	v02_07_03	

Pandora Output to LArSoft



Found Showers:
PFParticles tagged as showers (not LArSoft shower objects yet)

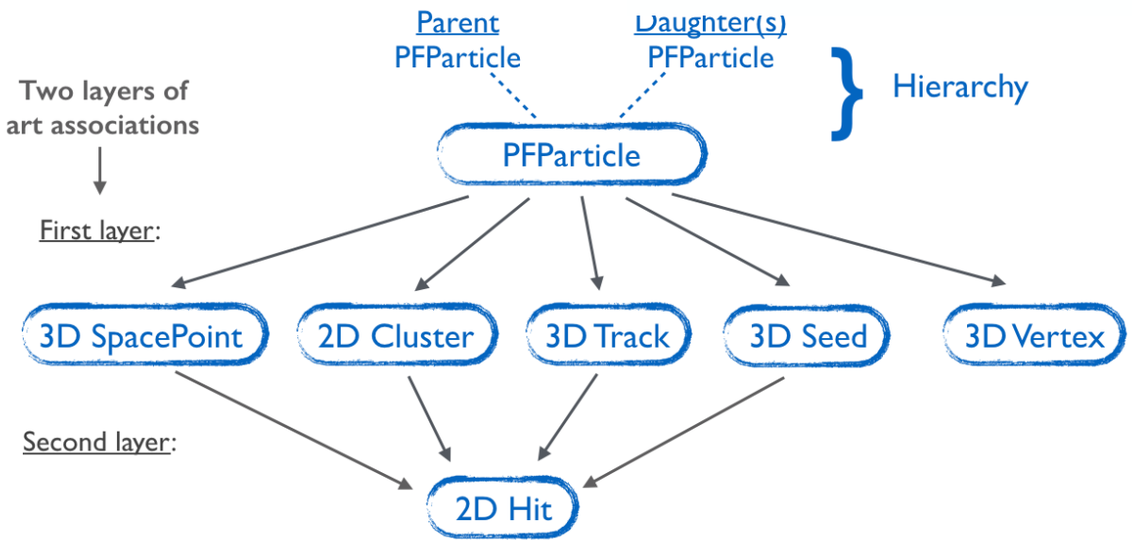
Note distinction between **Found Tracks** and **Found Showers** provided by Pandora and any downstream **Fitted Tracks** or **"Value-added" Showers** (with calorimetry information).

Pandora Output to LArSoft

LArPandoraOutput

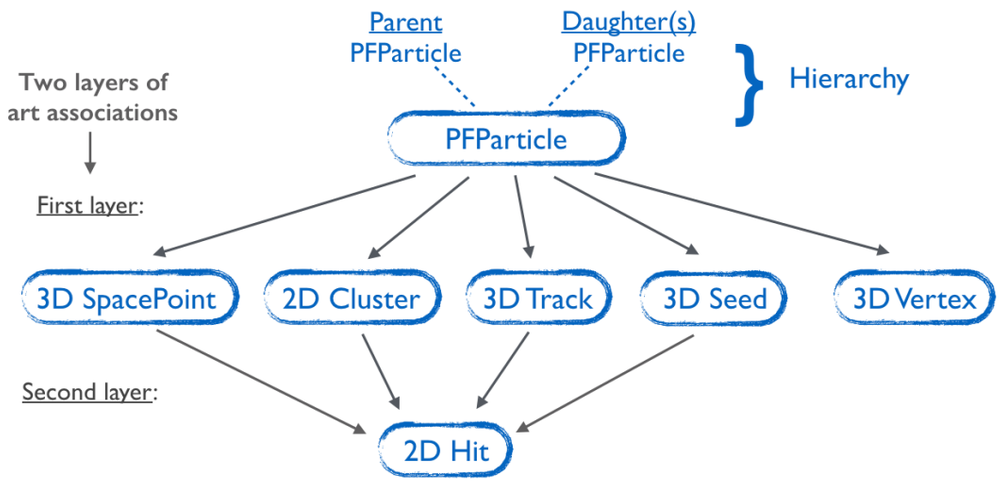
Produce

```
class LArPandoraOutput
{
public:
    /**
     *  @brief  Convert the Pandora PFOs into ART clusters and write into ART event
     *
     *  @param  settings the settings
     *  @param  idToHitMap the mapping from Pandora hit ID to ART hit
     *  @param  evt the ART event
     */
    static void ProduceArtOutput(const Settings &settings, const IdToHitMap &idToHitMap, art::Event &evt);

    /**
     *  @brief Build a recob::Cluster object from an input vector of recob::Hit objects
     *
     *  @param id the id code for the cluster
     *  @param hitVector the input vector of hits
     *  @param isolatedHits the input list of isolated hits, not to be fed to the cluster parameter algorithms
     *  @param algo Algorithm set to fill cluster members, if unsure StandardClusterParamsAlg is a good default
     */
    static recob::Cluster BuildCluster(const int id, const HitVector &hitVector, const HitList &isolatedHits,
       cluster::ClusterParamsAlgBase &algo);

    /**
     *  @brief Lookup ART hit from an input Pandora hit
     *
     *  @param idToHitMap the mapping between Pandora and ART hits
     *  @param pCaloHit the input Pandora hit (2D)
     */
    static art::Ptr<recob::Hit> GetHit(const IdToHitMap &idToHitMap, const pandora::CaloHit *const pCaloHit);

    ...
```



larpandoracontent/LArPandoraInterface/LArPandoraOutput.h

Pandora Output to LArSoft



We will focus on LArPandoraHelper in this talk and exercise 6

larpandoracontent/LArPandoraInterface/LArPandoraHelper.h

Navigate

LArPandoraHelper

```
class LArPandoraHelper
{
public:
    /**
     *  @brief  DaughterMode enumeration
     */
    enum DaughterMode
    {
        kIgnoreDaughters = 0,    // Only use parent particles
        kUseDaughters = 1,       // Use both parent and daughter partcles
        kAddDaughters = 2        // Absorb daughter particles into parent particles
    };

    /**
     *  @brief Collect the reconstructed wires from the ART event record
     *
     *  @param evt the ART event record
     *  @param label the label for the Wire list in the event
     *  @param wireVector the ouput vector of Wire objects
     */
    static void CollectWires(const art::Event &evt, const std::string label, WireVector &wireVector);

    /**
     *  @brief Collect the reconstructed Hits from the ART event record
     *
     *  @param evt the ART event record
     *  @param label the label for the Hit list in the event
     *  @param hitVector the ouput vector of Hit objects
     */
    static void CollectHits(const art::Event &evt, const std::string label, HitVector &hitVector);

    /**
     *  @brief Collect the reconstructed PFParticles from the ART event record
     *
     *  @param evt the ART event record
     *  @param label the label for the PFParticle list in the event
     *  @param particleVector the output vector of PFParticle objects
     */
    static void CollectPFParticles(const art::Event &evt, const std::string label, PFParticleVector &particleVector);
```

Warning to mariners

- LArSoft output must be handled **carefully:** use PFParticle functionality to navigate along particle hierarchies, then must always use art associations to navigate to related objects.
  - Be consistent with hit finders and producer modules
  - Be careful when using output from cosmic and neutrino passes
  - Example usage: **`larpandora/LArPandoraInterface/LArPandoraHelper`**

Navigate carefully

Warning to mariners

•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.

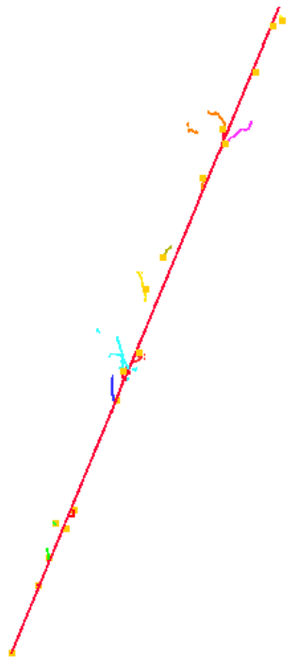
•Instead, it tries to make multi-algorithm event reconstruction easier to design, implement, test and maintain. It takes care of most memory-management, provides visual debugging, etc.

If you are wondering about larlite… not just yet!

larpandora isn't integrated in larlite yet, at the moment is not possible to use the static functions we will see there

PandoraNu and PandoraCosmic

Cosmic pass



Optimised for cosmic rays reconstruction

- Strongly track-oriented.
- Showers assumed to be delta-rays, added as muon daughters
- Muon vertices at track high-y position
- Producer

  pandoraCosmic

- Algorithms defined by

  PandoraSettings_MicroBooNE_Cosmic.xml

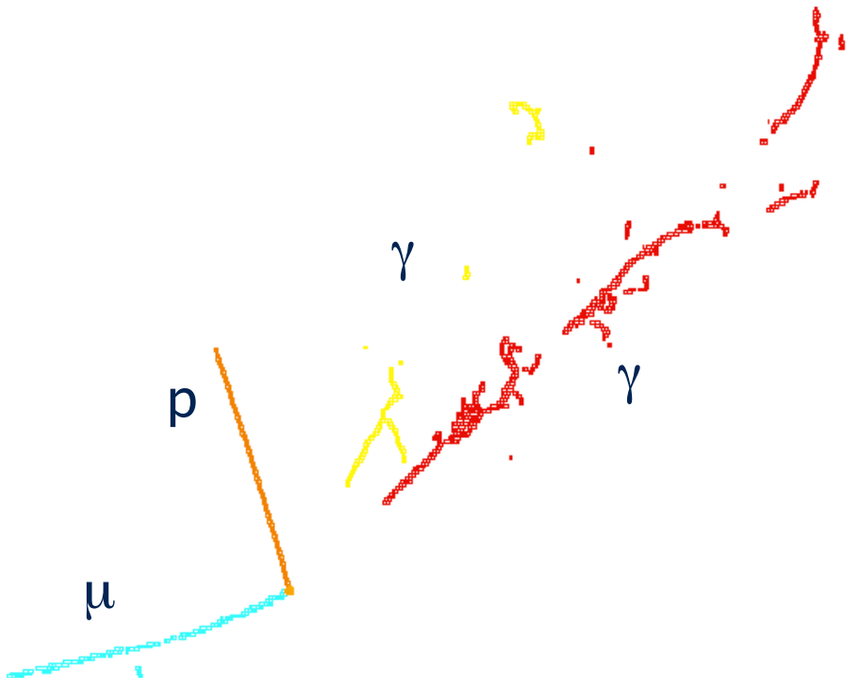
- Input hits

  gaushit

```
void nmspc::MyPandoraAnalyzer::reconfigure(fhicl::ParameterSet const & pset)
{
  m_particleLabelCosmic  = pset.get<std::string>("PFParticleModuleCosmic","pandoraCosmic");
  m_hitfinderLabelCosmic = pset.get<std::string>("HitFinderModuleCosmic","gaushit");
}
```

PandoraNu and PandoraCosmic

Neutrino pass



Optimised for neutrino reconstruction

- Carefully finding neutrino interaction vertex
- Daughters emerging from neutrino vertex
- Producer

  pandoraNu

- Algorithms defined by

  PandoraSettings_MicroBooNE_Neutrino.xml

- Input hits

  pandoraCosmicKHitRemoval (gaushit if only BNB)

```
void nmspc::MyPandoraAnalyzer::reconfigure(fhicl::ParameterSet const & pset)
{
  m_particleLabel = pset.get<std::string>("PFParticleModule","pandoraNu");
  m_hitfinderLabel = pset.get<std::string>("HitFinderModule","pandoraCosmicKHitRemoval");
}
```

PandoraNu and PandoraCosmic

- In between both producers, cosmic tagging and cosmic hit removal happens, completely external to Pandora. This means that pandoraNu will run the neutrino-interaction oriented pass to all remaining hits after cosmic hit removal.

- So the output from pandoraCosmic will be the output obtained using all available hits (gaushit) and the output from pandoraNu will be the output obtained using available hits after cosmic hit removal (pandoraCosmicKHitRemoval)

- Same of the hits may remain after the cosmic hit removal, and then be in both outputs. So, if using both producers at the same time, check for those not repeating

As an example, you can see in the PiZeroROIFilter a function created to:

- loop over the hits in cosmic clusters (clustersToHitsCosmic map)
- compare them with hits in pandoraNu clusters (clustersToHits map)
- decide whether to usethe cosmic cluster (if hits were not re-used)

https://github.com/jzennamo/PiZeroROI/tree/feature/pfparticle_filter/PiZeroFilter

Pandora LArPandoraHelper

In LArPandoraHelper, we have static functions to:

1. Collect Wires and Hits
2. Collect PFParticles, Tracks and Showers
3. Collect SpacePoints, Vertices and Clusters
4. Build maps PFParticles to Hits
5. Get MC particles and maps to hits
6. Get information about PFParticles (IsTrack?…)
7. Navigate PFParticles hierarchy (GetParent…)

Pandora LArPandoraHelper

0) Typedef examples

Inside LArPandoraHelper we find definitions of objects vectors or maps of other art objects to handle the output information:

```
namespace lar_pandora
{
typedef std::vector< art::Ptr<recob::Hit> >          HitVector;
typedef std::vector< art::Ptr<recob::SpacePoint> >   SpacePointVector;
typedef std::vector< art::Ptr<recob::Cluster> >      ClusterVector;
typedef std::vector< art::Ptr<recob::Vertex> >       VertexVector;
typedef std::vector< art::Ptr<recob::Track> >        TrackVector;
typedef std::vector< art::Ptr<recob::Shower> >       ShowerVector;
typedef std::vector< art::Ptr<recob::PFParticle> >   PFParticleVector;
typedef std::vector< art::Ptr<simb::MCTruth> >       MCTruthVector;
typedef std::vector< art::Ptr<simb::MCParticle> >    MCParticleVector;
```

Vectors of art objects

```
typedef std::map< art::Ptr<recob::PFParticle>, TrackVector >          PFParticlesToTracks;
typedef std::map< art::Ptr<recob::PFParticle>, ShowerVector >         PFParticlesToShowers;
typedef std::map< art::Ptr<recob::PFParticle>, ClusterVector >        PFParticlesToClusters;
typedef std::map< art::Ptr<recob::PFParticle>, SeedVector >           PFParticlesToSeeds;
typedef std::map< art::Ptr<recob::PFParticle>, VertexVector >         PFParticlesToVertices;
typedef std::map< art::Ptr<recob::PFParticle>, SpacePointVector >     PFParticlesToSpacePoints;
typedef std::map< art::Ptr<recob::PFParticle>, HitVector >            PFParticlesToHits;
```

Maps to associate different art objects

1) Collect Information

```
/**
 *  @brief Collect the reconstructed Hits from the ART event record
 *
 *  @param evt the ART event record
 *  @param label the label for the Hit list in the event
 *  @param hitVector the ouput vector of Hit objects
 */
static void CollectHits(const art::Event &evt, const std::string label, HitVector &hitVector);
```

Label of producer module: gaushit, pandoraCosmicKHitRemoval…

Vectors of art objects where we will collect the output

Example usage

```
// Collect Hits
// ============
lar_pandora::HitVector hitVector;
lar_pandora::LArPandoraHelper::CollectHits(evt, m_hitfinderLabel, hitVector);
```

Pandora LArPandoraHelper

1) Collect Information

```
    /**
     *  @brief Collect the reconstructed PFParticles and associated SpacePoints from the ART event record
     *
     *  @param evt the ART event record
     *  @param label the label for the PFParticle list in the event
     *  @param particleVector the output vector of PFParticle objects
     *  @param particlesToSpacePoints the output map from PFParticle to SpacePoint objects
     */
    static void CollectPFParticles(const art::Event &evt, const std::string label,
PFParticleVector &particleVector, PFParticlesToSpacePoints &particlesToSpacePoints);

    /**
     *  @brief Collect the reconstructed PFParticles and associated Clusters from the ART event record
     *
     *  @param evt the ART event record
     *  @param label the label for the PFParticle list in the event
     *  @param particleVector the output vector of PFParticle objects
     *  @param particlesToClusters the output map from PFParticle to Cluster objects
     */
    static void CollectPFParticles(const art::Event &evt, const std::string label,
PFParticleVector &particleVector, PFParticlesToClusters &particlesToClusters);
```

Label of producer module: pandoraNu, pandoraCosmic

Output vector + output map

Example usage

```
    // Collect PFParticles and match to clusters
    // =========================================
    lar_pandora::PFParticleVector pfParticleList; //vector of PFParticles
    lar_pandora::PFParticlesToClusters pfParticleToClusterMap; //PFParticle-to-cluster map
    lar_pandora::LArPandoraHelper::CollectPFParticles(evt, m_particleLabel, pfParticleList,
pfParticleToClusterMap); //collect PFParticles and build map PFParticles to Clusters
```

Pandora LArPandoraHelper

1) Collect Information

```
  // Collect PFParticles and match to clusters
  // =========================================
  lar_pandora::PFParticleVector pfParticleList; //vector of PFParticles
  lar_pandora::PFParticlesToClusters pfParticleToClusterMap; //PFParticle-to-cluster map
  lar_pandora::LArPandoraHelper::CollectPFParticles(evt, m_particleLabel, pfParticleList,
pfParticleToClusterMap); //collect PFParticles and build map PFParticles to Clusters
```

What is this function doing behind the scenes?

```
void LArPandoraHelper::CollectPFParticles(const art::Event &evt, const std::string label,
PFParticleVector &particleVector,PFParticlesToClusters &particlesToClusters)
{
  art::Handle< std::vector<recob::PFParticle> > theParticles;
  evt.getByLabel(label, theParticles);

  -- CHECK is valid and --

  art::FindManyP<recob::Cluster> theClusterAssns(theParticles, evt, label);
  for (unsigned int i = 0; i < theParticles->size(); ++i)
  {
      const art::Ptr<recob::PFParticle> particle(theParticles, i);
      particleVector.push_back(particle);

      const std::vector< art::Ptr<recob::Cluster> > clusters = theClusterAssns.at(i);
      for (unsigned int j=0; j<clusters.size(); ++j)
      {
          const art::Ptr<recob::Cluster> cluster = clusters.at(j);
          particlesToClusters[particle].push_back(cluster);
      }
  }
}
```

art::Handle to get the PFParticles

art::FindManyP cluster associations

Build output vector + map

2) Build maps navigating first to second layer of associations

```
    /**
     *  @brief Build mapping between PFParticles and Hits using PFParticle/Cluster/Hit maps
     *
     *  @param particleVector the input vector of PFParticle objects
     *  @param particlesToClusters the input map from PFParticle to Cluster objects
     *  @param clustersToHits the input map from Cluster to Hit objects
     *  @param particlesToHits the output map from PFParticle to Hit objects
     *  @param hitsToParticles the output map from Hit to PFParticle objects
     *  @param daughterMode treatment of daughter particles in construction of maps
     */
    static void BuildPFParticleHitMaps(const PFParticleVector &particleVector, const
PFParticlesToClusters &particlesToClusters,const ClustersToHits &clustersToHits, PFParticlesToHits
&particlesToHits, HitsToPFParticles &hitsToParticles, const DaughterMode daughterMode =
kUseDaughters);

    /**
     *  @brief Build mapping between PFParticles and Hits starting from ART event record
     *
     *  @param evt the ART event record
     *  @param label_pfpart the label for the PFParticle list in the event
     *  @param label_space the label for the Intermediate list in the event
     *  @param particlesToHits output map from PFParticle to Hit objects
     *  @param hitsToParticles output map from Hit to PFParticle objects
     *  @param daughterMode treatment of daughter particles in construction of maps
     *  @param useClusters choice of intermediate object (true for Clusters, false for SpacePoints)
     */
    static void BuildPFParticleHitMaps(const art::Event &evt, const std::string label_pfpart, const
std::string label_mid, PFParticlesToHits &particlesToHits, HitsToPFParticles &hitsToParticles, const
DaughterMode daughterMode = kUseDaughters, const bool useClusters = true);
```

1st-2nd associations

1st-2nd associations

2) Build maps navigating first to second layer of associations

Example usage

```
// Collect PFParticles and match Reco Particles to Hits
// ====================================================
lar_pandora::PFParticleVector recoParticleVector;
lar_pandora::PFParticlesToHits recoParticlesToHits;
lar_pandora::HitsToPFParticles recoHitsToParticles;

lar_pandora::LArPandoraHelper::CollectPFParticles(evt, m_particleLabel, recoParticleVector);
lar_pandora::LArPandoraHelper::BuildPFParticleHitMaps(evt, m_particleLabel, m_spacepointLabel,
recoParticlesToHits, recoHitsToParticles,lar_pandora::LArPandoraHelper::kAddDaughters)
```

Output maps

```
enum DaughterMode
{
    kIgnoreDaughters = 0,    // Only use parent particles
    kUseDaughters = 1,       // Use both parent and daughter particles
    kAddDaughters = 2        // Absorb daughter particles into parent particles
};
```

Basically, when building the map you can choose to use only parent particles, or parents and daughters, or absorb daughters into parents

2) Build maps navigating first to second layer of associations

What are these functions doing behind the scenes?

Input maps

Output maps

```
void LArPandoraHelper::BuildPFParticleHitMaps(const PFParticleVector &particleVector, const
PFParticlesToClusters &particlesToClusters, const ClustersToHits &clustersToHits,
PFParticlesToHits &particlesToHits, HitsToPFParticles &hitsToParticles, const DaughterMode daughterMode)
{
    // Build mapping from particle to particle ID for parent/daughter navigation
    PFParticleMap particleMap;
    (build map…)
    // Loop over hits and build mapping between reconstructed final-state particles and reconstructed hits
    for (PFParticlesToClusters::const_iterator iter1 = particlesToClusters.begin(), iterEnd1 =
particlesToClusters.end();
         iter1 != iterEnd1; ++iter1)
    {
        const art::Ptr<recob::PFParticle> thisParticle = iter1->first;
        const art::Ptr<recob::PFParticle> particle((kAddDaughters == daughterMode) ?
            LArPandoraHelper::GetFinalStatePFParticle(particleMap, thisParticle) : thisParticle);
        if ((kIgnoreDaughters == daughterMode) && !LArPandoraHelper::IsFinalState(particleMap, particle))
           continue;
        const ClusterVector &clusterVector = iter1->second;
        for (ClusterVector::const_iterator iter2 = clusterVector.begin(), iterEnd2 = clusterVector.end();
iter2 != iterEnd2; ++iter2)
        {
            const art::Ptr<recob::Cluster> cluster = *iter2;
            ClustersToHits::const_iterator iter3 = clustersToHits.find(cluster);

            const HitVector &hitVector = iter3->second;
            for (HitVector::const_iterator iter4 = hitVector.begin(), iterEnd4 = hitVector.end(); iter4 !=
iterEnd4; ++iter4)
            {
                const art::Ptr<recob::Hit> hit = *iter4;

                particlesToHits[particle].push_back(hit);
                hitsToParticles[hit] = particle;
            }
        }
    }
}
```

Input: 1st layer associations particlesToClusters, ClustersToHits

Output: 1st-2nd layer associations particlesToHits

3) Get MC truth maps

```
    /**
     *  @brief Build mapping between Hits and MCParticles, starting from Hit/TrackIDE/MCParticle
information
     *
     *  @param hitsToTrackIDEs the input map from hits to true energy deposites
     *  @param truthToParticles the input map of truth information
     *  @param particlesToHits the mapping between true particles and reconstructed hits
     *  @param hitsToParticles the mapping between reconstructed hits and true particles
     *  @param daughterMode treatment of daughter particles in construction of maps
     */
    static void BuildMCParticleHitMaps(const HitsToTrackIDEs &hitsToTrackIDEs, const
MCTruthToMCParticles &truthToParticles,MCParticlesToHits &particlesToHits, HitsToMCParticles
&hitsToParticles, const DaughterMode daughterMode = kUseDaughters);

    /**
     *  @brief Build mapping between Hits and MCParticles, starting from ART event record
     *
     *  @param evt the ART event record
     *  @param label the label for the truth information in the event
     *  @param hitVector the input vector of reconstructed hits
     *  @param particlesToHits the output mapping between true particles and reconstructed hits
     *  @param hitsToParticles the output mapping between reconstructed hits and true particles
     *  @param daughterMode treatment of daughter particles in construction of maps
     */
    static void BuildMCParticleHitMaps(const art::Event &evt, const std::string label, const
HitVector &hitVector,MCParticlesToHits &particlesToHits, HitsToMCParticles &hitsToParticles,
const DaughterMode daughterMode = kUseDaughters);
```

4) Get PFParticle info and hierarchy

PFParticles in the event

Useful to handle each PFParticle, and avoid writing pfParticleList.at(n) in each function later

PFParticle ID

```
for (unsigned int n = 0; n < pfParticleList.size(); ++n)
  {
    const art::Ptr<recob::PFParticle> particle = pfParticleList.at(n);

    std::cout <<"PFParticle: " << particle->Self()
              << " IsPrimary? " << particle->IsPrimary()
              << " IsNeutrino? " << lar_pandora::LArPandoraHelper::IsNeutrino(particle)
              << " IsTrack? " << lar_pandora::LArPandoraHelper::IsTrack(particle)
              << " IsShower? " << lar_pandora::LArPandoraHelper::IsShower(particle)
    << std::endl;

  }
```

Helper functions to identify neutrinos, tracks and showers (as explained in Talk #8)

```
for (unsigned int n = 0; n < pfParticleList.size(); ++n)
    {
      const art::Ptr<recob::PFParticle> particle = pfParticleList.at(n);

      if(lar_pandora::LArPandoraHelper::IsNeutrino(particle))
        {
          const std::vector<size_t> &daughterIDs = particle->Daughters();
        }
      }
```

Get the vector of IDs of the daughters of the neutrino

Pandora Examples in AnaTree

uboonecode/uboone/AnalysisTree/AnalysisTree_module.cc

```
   // * PFParticles
   lar_pandora::PFParticleVector pfparticlelist;
   lar_pandora::PFParticlesToClusters pfParticleToClusterMap;
   lar_pandora::LArPandoraHelper::CollectPFParticles(evt, fPFParticleModuleLabel,
 pfparticlelist, pfParticleToClusterMap);

         ...

   if(fSavePandoraNuVertexInfo) {
     lar_pandora::PFParticleVector particleVector;
     lar_pandora::LArPandoraHelper::CollectPFParticles(evt, fPandoraNuVertexModuleLabel,
particleVector);
     lar_pandora::VertexVector vertexVector;
     lar_pandora::PFParticlesToVertices particlesToVertices;
     lar_pandora::LArPandoraHelper::CollectVertices(evt, fPandoraNuVertexModuleLabel,
vertexVector, particlesToVertices);

         ...

   lar_pandora::PFParticleVector neutrinoPFParticles;
   lar_pandora::LArPandoraHelper::SelectNeutrinoPFParticles(pfparticlelist, neutrinoPFParticles);
   PFParticleData.pfp_numNeutrinos = neutrinoPFParticles.size();

         ...
```

Thanks to Jack Weston (Cambridge)

Handling Pandora Output to LArSoft

Ready for exercise 6 !

We are going to build an analyser using Pandora information to test our new cluster merging algorithm*

PLAN!

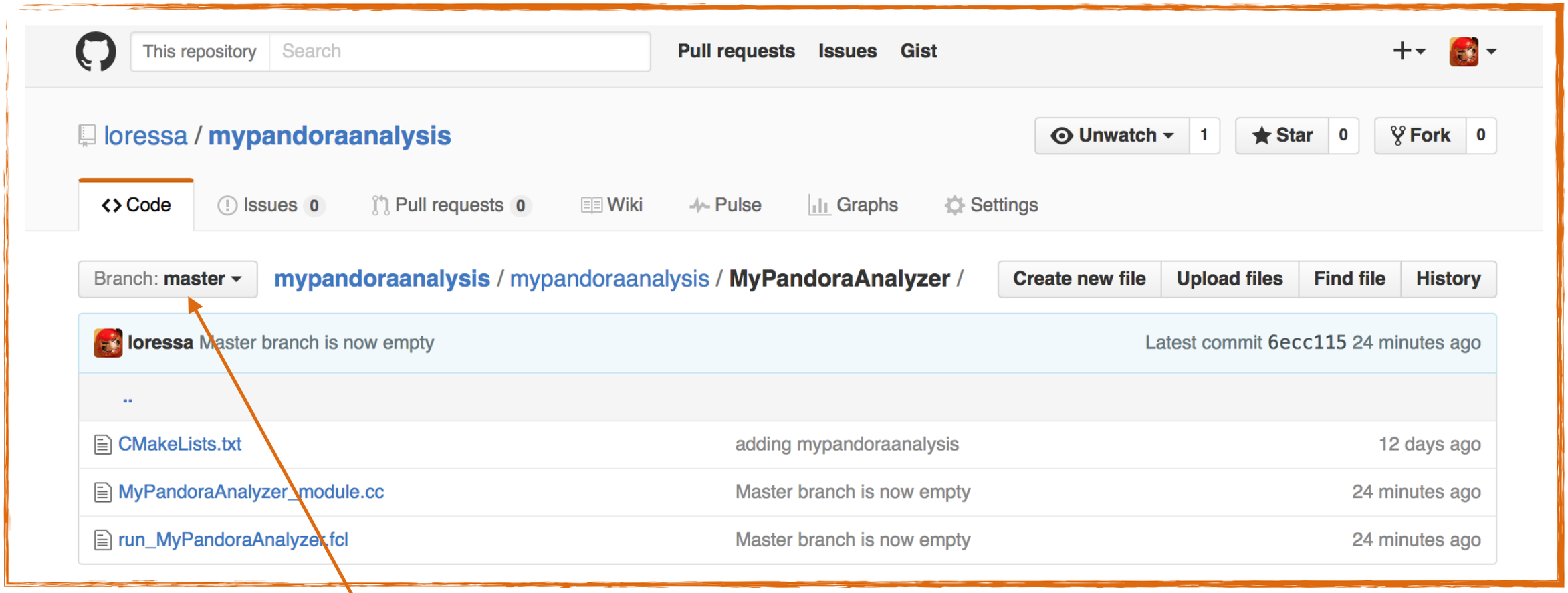
In this example exercise, we will be interested in finding:

1. PFParticles in our event which are track-like and daughters of the neutrino (with more daughters in case more than one neutrino)
2. Select from the above those with 3 clusters
3. Select from the above those with a minimum number of hits in each cluster
4. Build a map of matches comparing reconstructed and MC true particles
5. Print matching hits reco-true in each view (cluster)

*Don't worry if you didn't finish your new algorithm, just use a reco2 input file as in exercise 1

Handling Pandora Output to LArSoft

Ready for exercise 6 !



There are two branches:

- **master**: with an empty analyser to start writing it from scratch
- **final**: with the complete analyser after following these pages

Follow instructions in Exercise 6 slides

Questions?