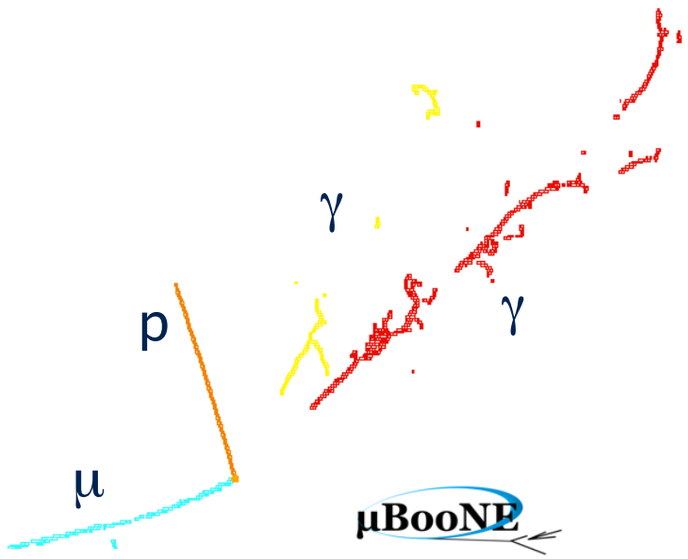


Pandora Exercise 6: Output to LArSoft

L. Escudero for the Pandora Team

MicroBooNE Pandora Workshop

July 11-14th 2016, Cambridge

Pandora Development in LArSoft

Pre-requisite: Exercise 1 - running Pandora in LArSoft.

Pre-requisite: Exercise 2 - setup Pandora environment and add a new algorithm.

To run algorithms developed during this workshop in LArSoft, need to build the algorithms and register their factories in the Pandora client application.

Can choose to add the algorithms to the LArPandoraContent library, the LArPandora client application or build as part of an all-new library, maybe with new dependencies.

Once built and registered, can then add algorithms to the relevant PandoraSettings file.

For Exercise 6: no visualisation needed, so feel free to user uboone machines at FNAL

Pandora Development in LArSoft

Brief instructions: Obtain a local development copy of the LArPandora client application, add your new source and build

```
source /cvmfs/uboone.opensciencegrid.org/products/setup_uboone.sh
setup uboonecode v05_13_00 -q e9:prof

mkdir LArSoft_v05_13_00
cd LArSoft_v05_13_00
mrb newDev
# Follow-prompt to setup local development area

cd $MRB_SOURCE
mrb g larpandora

cp /path/to/MyTestAlgorithm.cc $MRB_SOURCE/larpandora/larpandora/MicroBooNEPandora/
cp /path/to/MyTestAlgorithm.h $MRB_SOURCE/larpandora/larpandora/MicroBooNEPandora/

# Edit algorithm implementation to adjust include paths and namespaces as required, or to taste
(Don't forget to #include PandoraMonitoringApi.h if necessary)

# Register the algorithm factory in MicroBooNEPandora_module.cc
```

In MicroBooNEPandora::CreatePrimaryPandoraInstance, before the call to ReadSettings:

```
PANDORA_THROW_RESULT_IF(pandora::STATUS_CODE_SUCCESS, !=, PandoraApi::RegisterAlgorithmFactory(
    *m_pPrimaryPandora, "MyTest", new lar_pandora::MyTestAlgorithm::Factory));
```

Pandora Development in LArSoft

```
cd $MRB_BUILDDIR
mrbsetenv
mrb i -j4
mrbslp
```

See Exercise 1 for further details

* you might need to change version of larreco in larpandora/ups/product_deps

MyPandoraSettings_MicroBooNE_Neutrino.xml:

```
<algorithm type = "MyTest">
    <OutputClusterListName>MyTestClusters</OutputClusterListName>
    <NHitsPerCluster>10</NHitsPerCluster>
</algorithm>
```

myreco_uboone_mcc7_driver_stage2.fcl:

```
#include "reco_uboone_mcc7_driver_common.fcl"

process_name: PandoraWorkshop

services.DetectorClocksService.InheritClockConfig:  false
services.TFileService.fileName: "reco_stage_2_hist.root"

physics.producers.pandoraNu.ConfigFile:              "MyPandoraSettings_MicroBooNE_Neutrino.xml"

physics.reco: [ pandoraNu ]
physics.trigger_paths: [ reco ]
outputs.out1.fileName: "%ifb_%tc_reco2.root"
outputs.out1.dataTier: "reconstructed"
source.inputCommands: ["keep *_*_*_*", "drop *_*_*_McRecoStage2" ]
```

```
lar -c myreco_uboone_mcc7_driver_stage2.fcl -n 5 /path/to/reco2/file.root
```

Handling Pandora Output to LArSoft

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

First: Let's get a LArSoft example analyser to work on

1) Get mypandoraanalysis

```
cd $MRB_SOURCE
mrb g https://github.com/loressa/mypandoraanalysis.git
mrb uc          #to update CMakeLists

                NOTICE: Adding mypandoraanalysis to CMakeLists.txt file
                NOTICE: Adding larpandora to CMakeLists.txt file


cd $MRB_BUILDDIR
mrb z
mrbsetenv
mrb install
```

Disclaimer

mypandoraanalysis is a simple LArSoft analyser ready to follow the next pages

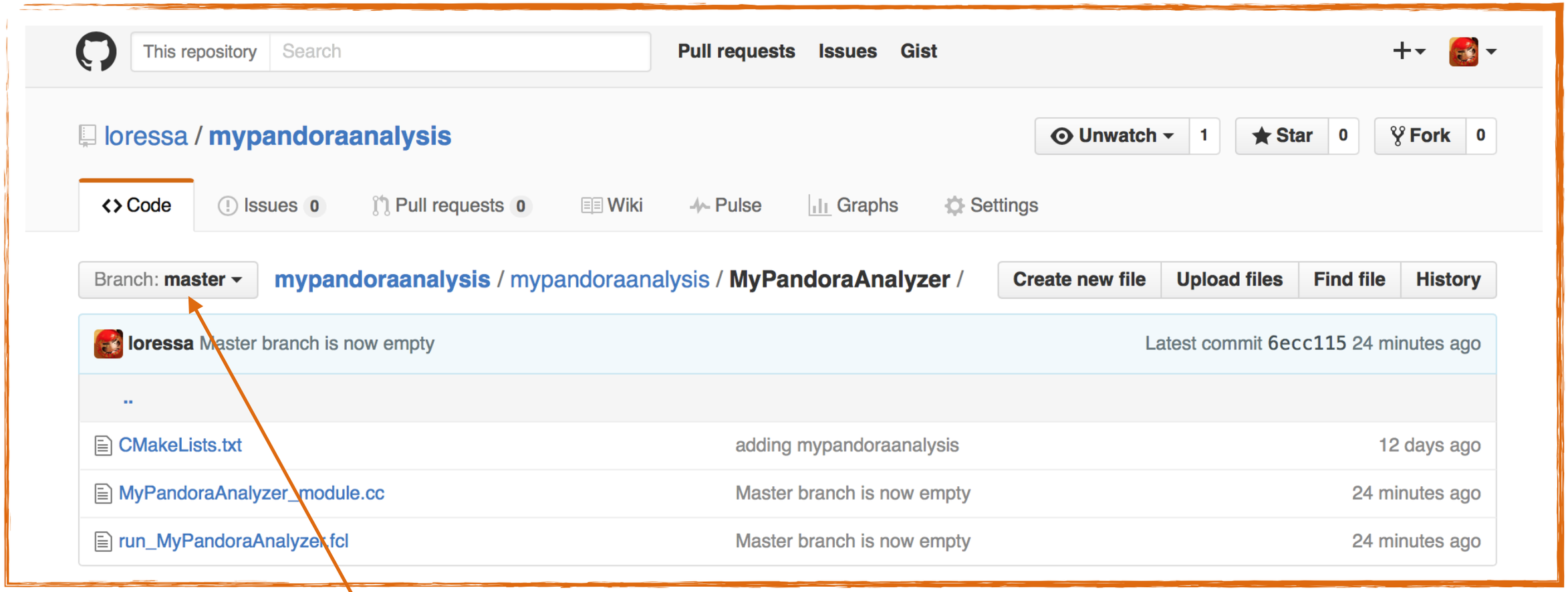
It has been created following larexample (LArSoftWiki: https://cdcvs.fnal.gov/redmine/projects/larsoft/wiki/_AnalysisExample_**) and the uboone analyser created by Wes Ketchum:** (https://github.com/wesketchum/ubuseranalysis/tree/master/UserAnalyzer)

If you are familiar with any of the above, or you have your preferred way to create an analyser already, go ahead and use it!

Handling Pandora Output to LArSoft

First: Let's get a LArSoft example analyser to work on

2) Inside mypandoraanalysis



There are two branches:

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

```
cd $MRB_SOURCE/mypandoraanalysis
git checkout final
```

Please don't push anything!

Handling Pandora Output to LArSoft

First: Let's get a LArSoft example analyser to work on

3) .fcl file to run our analyser `run_MyPandoraAnalyzer.fcl`

```
#include "services_microboone.fcl"

process_name : MyPandoraAnalyzer      #The process name must NOT contain any underscores

services : {

  scheduler:               { defaultExceptions: false }    # Make all uncaught exceptions fatal.
  TFileService: { fileName: "MyPandoraAnalyzer.root" }
  Timing:       {}
  SimpleMemoryCheck:     { ignoreTotal: 1 }     #default is one
  RandomNumberGenerator: {}                     #ART native random number generator
  user:
  {
        #BELOW you see a list of the common services! Uncomment the ones you need.
        DetectorClocksService:            @local::microboone_detectorclocks
  }
```

…usual .fcl file for an analyzer…

```
services.user.DetectorClocksService.InheritClockConfig:                    false
services.user.DetectorClocksService.TriggerOffsetTPC:                      -0.400e3
```

BUT we need to add these bits to work with MC truth information because in LArPandoraInterface/LArPandoraHelper we use the DetectorClocksService (some LArSoft magic trick…)

```
void LArPandoraHelper::BuildMCParticleHitMaps(const HitVector &hitVector,
const SimChannelVector &simChannelVector,
    HitsToTrackIDEs &hitsToTrackIDEs)
{
    auto const* ts = lar::providerFrom<detinfo::DetectorClocksService>();
```

Handling Pandora Output to LArSoft

And from now on…

4) Edit MyPandoraAnalyzer

We will edit in the following pages the analyser here:
srcs/mypandoraanalysis/mypandoraanalysis/MyPandoraAnalyzer/MyPandoraAnalyzer_module.cc

Remember to do after editing and before testing it the following:

```
cd $MRB_BUILDDIR
make install
mrbslp (x2)
```

5) Provide input file(s) and run

```
cd ..        (or $HOME)
echo path/to/file > myfilepath.txt     # e.g. output file with the result of your new algorithm or
                                       # LArSoft 'reco2' files in http://www.hep.phy.cam.ac.uk/~marshall/private/
                                       # Workshop/Events/ (you can add more than one file if you want)

lar -c run_MyPandoraAnalyzer.fcl -S myfilepath.txt
```

Handling Pandora Output to LArSoft

Let's start editing!

0) Include larpandora to use the LArPandoraHelper functions

- Add header file in the analyser:

srcs/mypandoraanalysis/mypandoraanalysis/MyPandoraAnalyzer/MyPandoraAnalyzer_module.cc

```
#include "larpandora/LArPandoraInterface/LArPandoraHelper.h"
```

- Add LArPandoraInterface in the CMakeLists.txt

srcs/mypandoraanalysis/mypandoraanalysis/MyPandoraAnalyzer/CMakeLists.txt

```
art_make(
         MODULE_LIBRARIES
                  larpandora_LArPandoraInterface
                  ${ART_FRAMEWORK_CORE}
                  ${ART_FRAMEWORK_IO_SOURCES}
                  ${ART_FRAMEWORK_PRINCIPAL}
                  ${ART_FRAMEWORK_SERVICES_REGISTRY}
                  ${ART_FRAMEWORK_SERVICES_OPTIONAL}
                  ${ART_FRAMEWORK_SERVICES_OPTIONAL_TFILESERVICE_SERVICE}
                  ${ART_PERSISTENCY_COMMON}
                  ${ART_PERSISTENCY_PROVENANCE}
                  ${ART_UTILITIES}
                  ${MF_MESSAGELOGGER}
                  ${MF_UTILITIES}
                  ${FHICLCPP}
                  ${CETLIB}
                  ${ROOT_GEOM}
                  ${ROOT_XMLIO}
                  ${ROOT_GDML}
                  ${ROOT_BASIC_LIB_LIST}
                  ${SIMULATIONBASE}
)

install_headers()
install_source()
install_fhicl()
```

Handling Pandora Output to LArSoft

Let's start editing!

1. PFParticles in our event which are track-like and daughters of the neutrino

Collect Pandora information I: PFParticles

- Now in the "analyze" part of our Analyzer (main), we will start including the vectors, maps and thrill call the static functions we have seen in the workshop talk #8 (output to LArSoft)

- The first thing we need is to collect the list of PFParticles. We will collect them using the option constructing at the same time the map PFParticles to Clusters, as we will need that later

```
#include "lardata/RecoBase/PFParticle.h"
#include "lardata/RecoBase/Vertex.h"
#include "lardata/RecoBase/Track.h"
#include "lardata/RecoBase/Cluster.h"
#include "lardata/RecoBase/Hit.h"
```

- We need to include the header files corresponding to the objects we will use

```
void nmspc::MyPandoraAnalyzer::analyze(art::Event const & evt)
{

  //Vectors and maps we will use to store Pandora information
  lar_pandora::PFParticleVector pfParticleList; //vector of PFParticles
  lar_pandora::PFParticlesToClusters pfParticleToClusterMap; //PFParticle-to-cluster map

  //Use LArPandoraHelper functions to collect Pandora information
  lar_pandora::LArPandoraHelper::CollectPFParticles(evt, m_particleLabel, pfParticleList, pfParticleToClusterMap);
                                 //collect PFParticles and build map PFParticles to Clusters
```

Handling Pandora Output to LArSoft

1. PFParticles in our event which are track-like and daughters of the neutrino

Choosing module labels

- **In this exercise, we are interested in the output of "pandoraNu" (using BNB only)**
- We will simply use the usual reconfigure function as follows:

```
private:
  std::string  m_hitfinderLabel;          ///<
  std::string  m_spacepointLabel;         ///<
  std::string  m_particleLabel;           ///<
  std::string  m_geantModuleLabel;        ///<
public:
  void reconfigure(fhicl::ParameterSet const &pset) override;
```

```
nmspc::MyPandoraAnalyzer::MyPandoraAnalyzer(fhicl::ParameterSet const & pset):  EDAnalyzer(pset)
{
  this->reconfigure(pset);
}
```

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

Handling Pandora Output to LArSoft

1. PFParticles in our event which are track-like and daughters of the neutrino

Investigate Pandora information I: PFParticles

PFParticles in the event

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

```
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;

  }
```

PFParticle ID

Helper functions to identify neutrinos, tracks and showers

Example output:

```
PFParticle: 0 IsPrimary? 0 IsNeutrino? 0 IsTrack? 0 IsShower? 1
PFParticle: 1 IsPrimary? 0 IsNeutrino? 0 IsTrack? 1 IsShower? 0
PFParticle: 2 IsPrimary? 0 IsNeutrino? 0 IsTrack? 1 IsShower? 0
PFParticle: 3 IsPrimary? 0 IsNeutrino? 0 IsTrack? 1 IsShower? 0
PFParticle: 4 IsPrimary? 1 IsNeutrino? 1 IsTrack? 0 IsShower? 0
```

In this event with 5 PFParticles: 1 neutrino, 1 shower and 3 tracks

Handling Pandora Output to LArSoft

1. PFParticles in our event which are track-like and daughters of the neutrino

Investigate Pandora information II: Neutrino daughters

```
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();
      std::cout << "Neutrino has ID: " << pfParticleList.at(n)->Self()
                << " and " << daughterIDs.size() << " daughters" << std::endl;

      for (unsigned int m = 0; m < daughterIDs.size(); ++m)
        {
          const art::Ptr<recob::PFParticle> daughter = pfParticleList.at(daughterIDs.at(m));

          std::cout << "Neutrino daughter ID: " << daughterIDs[m]
        << " IsTrack? " << lar_pandora::LArPandoraHelper::IsTrack(daughter)
                    << " IsShower? " << lar_pandora::LArPandoraHelper::IsShower(daughter)
        << std::endl;
        }
    }
}
```

Get the vector of IDs of the daughters of the neutrino

Notice that: daughterID[j] is equivalent to pfParticleList.at(daughterIDs[j])->Self()

```
Neutrino has ID: 4 and 4 daughters
Neutrino daughter ID: 0 IsTrack? 0 IsShower? 1
Neutrino daughter ID: 1 IsTrack? 1 IsShower? 0
Neutrino daughter ID: 2 IsTrack? 1 IsShower? 0
Neutrino daughter ID: 3 IsTrack? 1 IsShower? 0
```

Same event as in previous slide: the 3 tracks and the shower are daughters of the neutrino (ID 4)

Handling Pandora Output to LArSoft

2.Select those track-like daughters of the neutrino with 3 clusters

Using maps of associations I: PFParticles-to-Cluster

- First we are going to find the neutrino with more daughters in the event (in case there is more than one neutrino reconstructed)

```
size_t nuID = -std::numeric_limits<std::size_t>::max();
bool found_neutrino = false;
unsigned int n_daughters = 0;

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();
        if(daughterIDs.size() > n_daughters)
          {
            n_daughters = daughterIDs.size();
            found_neutrino = true;
            nuID = n;
          }
      }
  }//at the end, nuID is the index of the PFParticle
//corresponding to the neutrino with more daughters
```

We will save the ID of the neutrino with more daughters to use it later

Be careful: we might not even have one neutrino reconstructed!

- Then we will loop over the daughters of this neutrino, call a function to find the number of clusters, and select those with 3 clusters.

Handling Pandora Output to LArSoft

2.Select those track-like daughters of the neutrino with 3 clusters

Using maps of associations I: PFParticles-to-Cluster (cont'd)

```
if(found_neutrino)
   {
     //Now we can start directly with the neutrino PFParticle
     const art::Ptr<recob::PFParticle> neutrino = pfParticleList.at(nuID);
     const std::vector<size_t> &daughterIDs = neutrino->Daughters();
     for (unsigned int m = 0; m < daughterIDs.size(); ++m)
       {
         const art::Ptr<recob::PFParticle> daughter = pfParticleList.at(daughterIDs[m]);
         unsigned int n_clusters = this->NumberClusters(daughter,pfParticleToClusterMap);
         if(n_clusters==3)
           {
                ★ (To continue here)
           }

       }//daughters
   }//found neutrino
```

Call to the function returning number of clusters

```
private:

  // Declare member data here.
  unsigned int NumberClusters(const art::Ptr<recob::PFParticle> particle,
                              const lar_pandora::PFParticlesToClusters pfParticleToClusterMap) const;
```

Handling Pandora Output to LArSoft

2.Select those track-like daughters of the neutrino with 3 clusters

Using maps of associations I: PFParticles-to-Cluster (cont'd)

```
unsigned int nmspc::MyPandoraAnalyzer::NumberClusters(const art::Ptr<recob::PFParticle> particle, const
lar_pandora::PFParticlesToClusters pfParticleToClusterMap) const)
{
  unsigned int n_clusters = 0;
  lar_pandora::PFParticlesToClusters::const_iterator clusterMapIter = pfParticleToClusterMap.find(particle);//find
clusters
  if (clusterMapIter != pfParticleToClusterMap.end())
    {
      lar_pandora::ClusterVector clusters = clusterMapIter->second;
      if(clusters.size()>3)
        std::cerr << "Daughter has more than three clusters!" << std::endl;
      //check there are no more than one cluster in each plane
      for(unsigned int k = 0; k<3; k++)
        {
          int n_clusters_view = 0;
          for(unsigned int l = 0; l < clusters.size(); ++l)
            {
              if(clusters[l]->Plane().Plane==k)
                n_clusters_view++;
            }
          if(n_clusters_view>1)
            std::cerr << "Daughter has more than one cluster in one plane!" << std::endl;
        }
      n_clusters = clusters.size();
    }

  return n_clusters;
}
```

Check no PFParticle has more than 3 clusters, or more than one in one plane (this should not happen!)

Handling Pandora Output to LArSoft

3.Selecting those with a minimum of 30 hits per cluster

Using maps of associations II: Clusters-to-Hits

- We are going to build a function giving us for each PFParticle selected a vector with the number of hits in its clusters.

- First we collect the hits and store the vector of clusters and the map clusters to hits and pass it to the function

```
private:

  // Declare member data here.
   bool NumberRecoHits(const art::Ptr<recob::PFParticle> particle, const
 lar_pandora::PFParticlesToClusters pfParticleToClusterMap, const lar_pandora::ClustersToHits
 clustersToHits) const;
```

```
void nmspc::MyPandoraAnalyzer::analyze(art::Event const & evt)
{
  // Collect Clusters
  // ================
  lar_pandora::ClusterVector clusterVector; //vector of clusters
  lar_pandora::ClustersToHits clustersToHits;//Clusters-to-Hits map
  lar_pandora::LArPandoraHelper::CollectClusters(evt, m_spacepointLabel, clusterVector, clustersToHits);
                                 //map Clusters-to-hits
```

★ `if(this->NumberRecoHits(daughter,pfParticleToClusterMap,clustersToHits))`

Handling Pandora Output to LArSoft

3.Selecting those with a minimum of 30 hits per cluster

Using maps of associations II: Clusters-to-Hits (cont'd)

- We store the number of hits per cluster (plane) and ask the function if all have >30 hits

```
bool nmspc::MyPandoraAnalyzer::NumberRecoHits(const art::Ptr<recob::PFParticle> particle,const
lar_pandora::PFParticlesToClusters pfParticleToClusterMap, const lar_pandora::ClustersToHits
clustersToHits) const
{
  double hits[3] = {0.0,0.0,0.0};
  lar_pandora::PFParticlesToClusters::const_iterator clusterMapIter =
pfParticleToClusterMap.find(particle);//find clusters
  if (clusterMapIter != pfParticleToClusterMap.end())
    {
      const lar_pandora::ClusterVector & clusters = clusterMapIter->second;
      for(unsigned int l = 0; l < clusters.size(); ++l)
        {
          int index = clusters[l]->Plane().Plane;
          lar_pandora::ClustersToHits::const_iterator clustersHitsMapIter =
clustersToHits.find(clusters[l]);
          if(clustersHitsMapIter != clustersToHits.end())
            {
              const lar_pandora::HitVector hitvec = clustersHitsMapIter->second;
              hits[index]=hitvec.size();
            }
        }
    }

  return ((hits[0]>30)&&(hits[1]>30)&&(hits[2]>30));
}
```

Association I: particle to cluster map

Use the plane of the cluster as index

Association II: cluster to hits

Handling Pandora Output to LArSoft

4. Compare with MC truth to find matching hits

Using truth information

- In this exercise, we will use the MC truth information and compare with reconstructed information to find matching hits
- We need to add the following to handle the MC truth information now:

```
 // Collect MCParticles and match True Particles to Hits
 // ----------------------------------------------------
   lar_pandora::MCParticleVector trueParticleVector;
   lar_pandora::MCTruthToMCParticles truthToParticles;
   lar_pandora::MCParticlesToMCTruth particlesToTruth;
   lar_pandora::MCParticlesToHits trueParticlesToHits;
   lar_pandora::HitsToMCParticles trueHitsToParticles;

   lar_pandora::LArPandoraHelper::CollectMCParticles(evt, m_geantModuleLabel, trueParticleVector);
   lar_pandora::LArPandoraHelper::CollectMCParticles(evt, m_geantModuleLabel, truthToParticles,
particlesToTruth);
   lar_pandora::LArPandoraHelper::BuildMCParticleHitMaps(evt, m_geantModuleLabel, hitVector,
trueParticlesToHits, trueHitsToParticles, (m_useDaughterMCParticles ? (m_addDaughterMCParticles ?
lar_pandora::LArPandoraHelper::kAddDaughters : lar_pandora::LArPandoraHelper::kUseDaughters) :
lar_pandora::LArPandoraHelper::kIgnoreDaughters));
```

Vectors and maps we need to store the information

To add as well in reconfigure:

```
m_addDaughterMCParticles = pset.get<bool>("AddDaughterMCParticles",true);
```

Handling Pandora Output to LArSoft

4. Compare with MC truth to find matching hits

Using truth information

- Same maps for reconstructed PFParticles:

```
// 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,
(m_useDaughterPFParticles ? (m_addDaughterPFParticles ? lar_pandora::LArPandoraHelper::kAddDaughters :
lar_pandora::LArPandoraHelper::kUseDaughters) : lar_pandora::LArPandoraHelper::kIgnoreDaughters));

  if (m_printDebug)
    std::cout << "  RecoParticles: " << recoParticleVector.size() << std::endl;
```

Vectors and maps we need to store the information

To add as well in reconfigure:

```
m_addDaughterPFParticles = pset.get<bool>("AddDaughterPFParticles",true);
```

Handling Pandora Output to LArSoft

4. Compare with MC truth to find matching hits

Using truth information (cont'd)

4.1. We will use the following function (adapted from PFParticleMonitoring_module.cc in larpandora/LArPandoraAnalysis) to build a map of matches between MC particles and reconstructed particles

```
private:

  void GetRecoToTrueMatches(const lar_pandora::PFParticlesToHits &recoParticlesToHits,
                            const lar_pandora::HitsToMCParticles &trueHitsToParticles,
                            lar_pandora::MCParticlesToPFParticles &matchedParticles,
                            lar_pandora::MCParticlesToHits &matchedHits) const;
```

```
  // Match Reco Particles to True Particles
  // ======================================
  lar_pandora::MCParticlesToPFParticles matchedParticles;
  lar_pandora::MCParticlesToHits matchedParticleHits;
  this->GetRecoToTrueMatches(recoParticlesToHits, trueHitsToParticles, matchedParticles,
matchedParticleHits);
```

Maps to store matching information

Defined and filled in previous slides

Handling Pandora Output to LArSoft

4. Compare with MC truth to find matching hits

```
void nmspc::MyPandoraAnalyzer::GetRecoToTrueMatches(const lar_pandora::PFParticlesToHits
&recoParticlesToHits, const lar_pandora::HitsToMCParticles &\
trueHitsToParticles, lar_pandora::MCParticlesToPFParticles &matchedParticles, lar_pandora::MCParticlesToHits
&matchedHits) const
{
  bool foundMatches(false);

  for (lar_pandora::PFParticlesToHits::const_iterator iter1 = recoParticlesToHits.begin(), iterEnd1 =
recoParticlesToHits.end();
       iter1 != iterEnd1; ++iter1)
    {
      const art::Ptr<recob::PFParticle> recoParticle = iter1->first;
      const lar_pandora::HitVector &hitVector = iter1->second;
      lar_pandora::MCParticlesToHits truthContributionMap;

      for (lar_pandora::HitVector::const_iterator iter2 = hitVector.begin(), iterEnd2 = hitVector.end();
iter2 != iterEnd2; ++iter2)
        {
          const art::Ptr<recob::Hit> hit = *iter2;
          const double recoHitTime(hit->PeakTime());
          const geo::WireID recoHitWire(hit->WireID());

          lar_pandora::HitsToMCParticles::const_iterator iter3 = trueHitsToParticles.end();
          for (lar_pandora::HitsToMCParticles::const_iterator iter4 = trueHitsToParticles.begin(), iterEnd4
= trueHitsToParticles.end(); iter4 != iterEnd4; ++iter4)
            {
             const art::Ptr<recob::Hit> trueHit = iter4->first;
              const double trueHitTime(trueHit->PeakTime());
              const geo::WireID trueHitWire(trueHit->WireID());
              if( (trueHitTime==recoHitTime) &&
                  (trueHitWire==recoHitWire) )
                iter3 = trueHitsToParticles.find(trueHit);
            }
```

1) loop reco particle to hits

Storage map

2) loop hits to true particle

CAVEAT: This should not be necessary, instead it should simply replaced by trueHitsToParticles.find(hit) as it is done in PFParticleMonitoring. However, in this example we are using reconstructed files from LArSoft v5_08 in v5_13, mismatching in hits

continues in next slide…

Handling Pandora Output to LArSoft

4. Compare with MC truth to find matching hits

```
          if (trueHitsToParticles.end() == iter3)
            continue;

          const art::Ptr<simb::MCParticle> trueParticle = iter3->second;
          truthContributionMap[trueParticle].push_back(hit);
        }

      lar_pandora::MCParticlesToHits::const_iterator mIter = truthContributionMap.end();
    for (lar_pandora::MCParticlesToHits::const_iterator iter5 = truthContributionMap.begin(), iterEnd5 =
truthContributionMap.end(); iter5 != iterEnd5; ++iter5)
        {
          if ((truthContributionMap.end() == mIter) || (iter5->second.size() > mIter->second.size()))
            {
              mIter = iter5;
            }
        }

      if (truthContributionMap.end() != mIter)
        {
          const art::Ptr<simb::MCParticle> trueParticle = mIter->first;

          lar_pandora::MCParticlesToHits::const_iterator iter6 = matchedHits.find(trueParticle);

          if ((matchedHits.end() == iter6) || (mIter->second.size() > iter6->second.size()))
            {
              matchedParticles[trueParticle] = recoParticle;
              matchedHits[trueParticle] = mIter->second;
              foundMatches = true;
            }
        }
    }

  if (!foundMatches)
  return;

}
```

Final matching reco-true particles is for the matching with more matched hits

Handling Pandora Output to LArSoft

5.Print some interesting output

```
void PrintMatchingHits(const art::Ptr<recob::PFParticle> particle,
                       const lar_pandora::PFParticlesToHits &recoParticlesToHits,
                       const lar_pandora::MCParticlesToHits trueParticlesToHits,
                       lar_pandora::MCParticlesToPFParticles &matchedParticles,
                       lar_pandora::MCParticlesToHits &matchedHits) const;
```

★ The last piece in our analyser, so this is how its main body would look like:

```
  if(found_neutrino)
    {
      const art::Ptr<recob::PFParticle> neutrino = pfParticleList.at(nuID);
      const std::vector<size_t> &daughterIDs = neutrino->Daughters();
      for (unsigned int m = 0; m < daughterIDs.size(); ++m)
        {
          const art::Ptr<recob::PFParticle> daughter = pfParticleList.at(daughterIDs[m]);
          unsigned int n_clusters = this->NumberClusters(daughter,pfParticleToClusterMap);
          if(n_clusters==3)
            {
              if(this->NumberRecoHits(daughter,pfParticleToClusterMap,clustersToHits))
                {
                  this->PrintMatchingHits(daughter, recoParticlesToHits, trueParticlesToHits,
matchedParticles, matchedParticleHits);
                }
            }

        }//daughters
    }//found neutrino
```

Handling Pandora Output to LArSoft

5.Print some interesting output

```
void nmspc::MyPandoraAnalyzer::PrintMatchingHits(const art::Ptr<recob::PFParticle> particle, const
lar_pandora::PFParticlesToHits &recoParticlesToHits, const lar_pandora::MCParticlesToHits trueParticlesToHits,
lar_pandora::MCParticlesToPFParticles &matchedParticles, lar_pandora::MCParticlesToHits &matchedParticleHits) const
{
  lar_pandora::PFParticlesToHits::const_iterator pIter = recoParticlesToHits.find(particle);
  const lar_pandora::HitVector &recoHitVector = pIter->second;
  std::cout << "Daughter particle ID : " << particle->Self() << std::endl;
  std::cout << "     - Hits: " << recoHitVector.size()
            << " (" << this->CountHitsByType(geo::kU,recoHitVector)
            << " , " << this->CountHitsByType(geo::kV,recoHitVector)
            << " , " << this->CountHitsByType(geo::kW,recoHitVector)
            << ")" << std::endl;
  for (lar_pandora::MCParticlesToHits::const_iterator iter = trueParticlesToHits.begin(), iterEnd =
trueParticlesToHits.end(); iter != iterEnd; ++ite\
r)
    {
      const art::Ptr<simb::MCParticle> trueParticle = iter->first;
      const lar_pandora::HitVector &trueHitVector = iter->second;
      lar_pandora::MCParticlesToPFParticles::const_iterator pIter1 = matchedParticles.find(trueParticle);
      if (matchedParticles.end() != pIter1)
        {
          const art::Ptr<recob::PFParticle> recoParticle = pIter1->second;
          if(recoParticle==particle)
            {
              lar_pandora::MCParticlesToHits::const_iterator pIter2 = matchedParticleHits.find(trueParticle);
              const lar_pandora::HitVector &matchedHitVector = pIter2->second;
               std::cout << "     - Matches MC particle with PDG code : " << trueParticle->PdgCode() << " (" <<
trueHitVector.size() << " hits)" << std::endl;
              std::cout << "                 matched hits = " << matchedHitVector.size()
                        << " (" << this->CountHitsByType(geo::kU,matchedHitVector)
                        << " , " << this->CountHitsByType(geo::kV,matchedHitVector)
                        << " , " << this->CountHitsByType(geo::kW,matchedHitVector)
                        << ")" << std::endl;
            }
        }
    }
}
```

Using this function to separate in different views (next slide)

Handling Pandora Output to LArSoft

5.Print some interesting output

```
int CountHitsByType(const int view, const lar_pandora::HitVector &hitVector) const;
```

```
int nmspc::MyPandoraAnalyzer::CountHitsByType(const int view, const lar_pandora::HitVector
&hitVector) const
{
  int nHits(0);

  for (lar_pandora::HitVector::const_iterator iter = hitVector.begin(), iterEnd =
hitVector.end(); iter != iterEnd; ++iter)
    {
      const art::Ptr<recob::Hit> hit = *iter;
      if (hit->View() == view)
   ++nHits;
    }

  return nHits;
}
```

you can continue and e.g. define purity per clusters

End of the exercise!

Handling Pandora Output to LArSoft

Still time? Want to try adding cosmics to the equation?

- Remember to:
  - Use different vectors and maps to store the information for the neutrino pass and the cosmic pass
  - Use two different labels to get the output from **pandoraCosmic** and **pandoraNu**
  - Build a function to check which cosmic hits are kept as so and not passed again to the pandoraNu, to avoid duplication

Next Exercise: Write a more complex Algorithm - Particle Merging