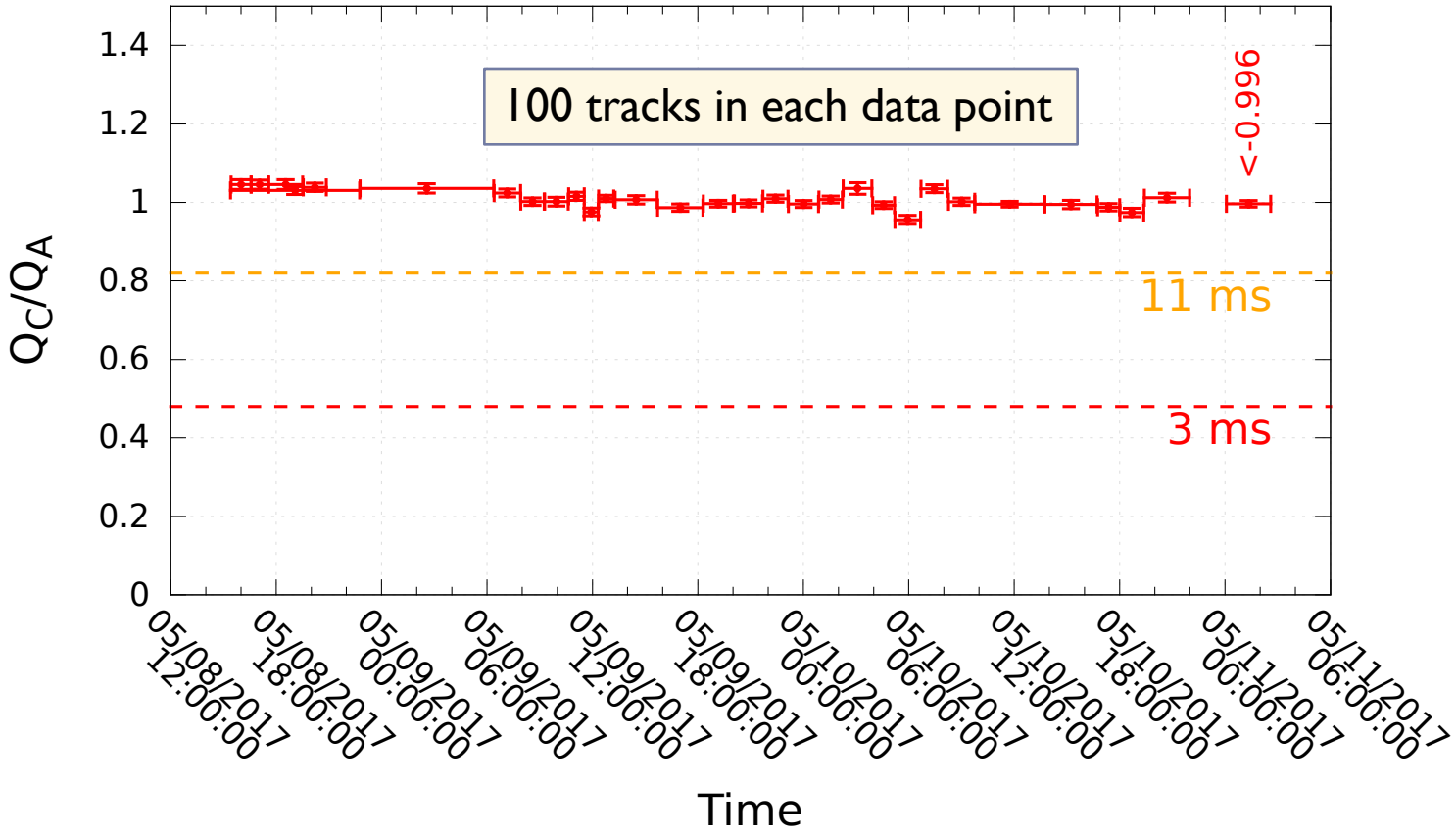
Nearline Purity Monitor

Bruce Baller
June 8, 2017

Goal – Monitor Purity vs Time

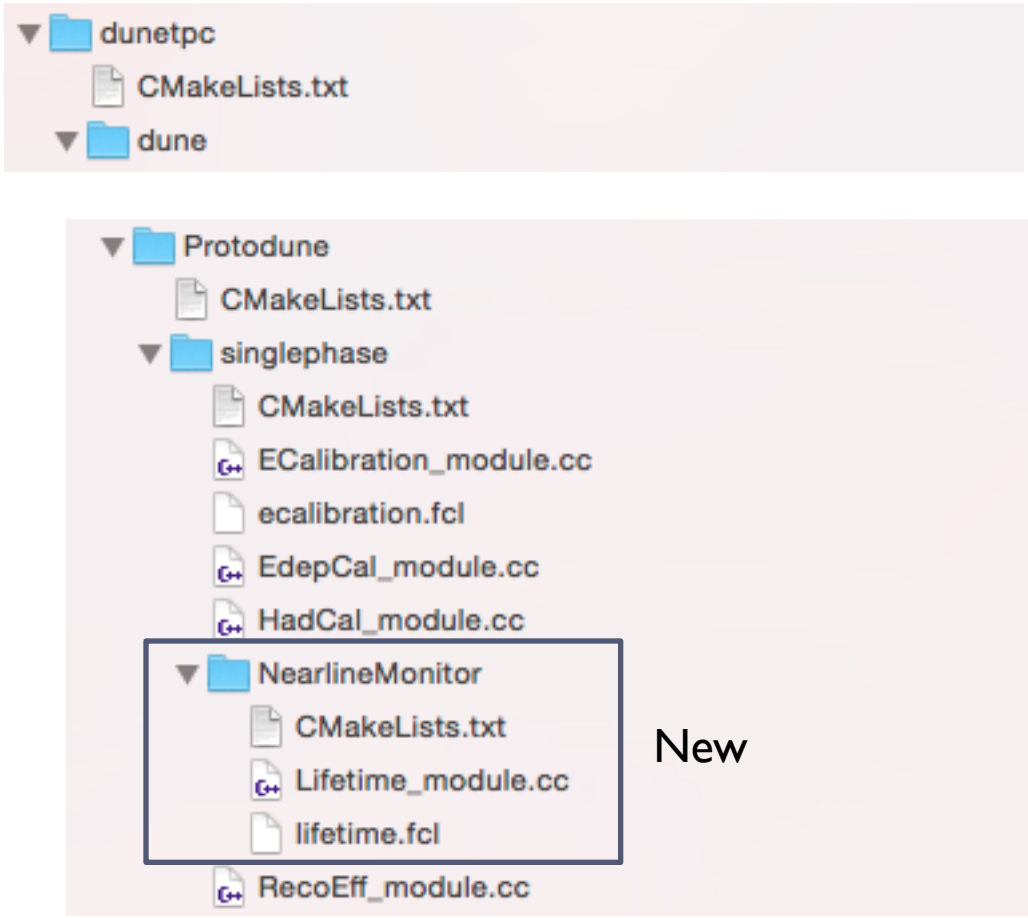
Using Cathode-Anode piercing Cosmic Rays

Q_C (Q_A) = Average charge of tracks near the Cathode (Anode)



NearlineMonitor Repository

Feature branch bb_Purity



Tentative Work Flow
Minimum Processing

- Process wire signals and reconstruct hits on the collection plane
  - Alternative if noise is low: Reconstruct hits from raw wire signals?
- Reconstruct cosmic rays with TrajCluster
  - One pass configured for long trajectories
- Lifetime analyzer module
  - Define fiducial X cut
  - Select clusters having X length ~ drift distance
  - Calculate Q_A and Q_C for each TPC and pass results to nearline process (somehow)
    - Incorporate space charge correction?

```
void nlana::Lifetime::analyze(art::Event const & evt)
{

  int event  = evt.id().event();
  int run    = evt.run();
  int subrun = evt.subRun();
  std::cout<<"Inside analyze "<<run<<" "<<" subrun "<<subrun<<" event "<<event<<"\n";



  static bool first = true;
  if(first) {
    first = false;
    // Get the low and high tick range for plane 2 in each TPC
    const geo::GeometryCore* geom = lar::providerFrom<geo::Geometry>();
    const detinfo::DetectorProperties* detprop = lar::providerFrom<detinfo::DetectorPropertiesService>();
    double local[3] = {0.,0.,0.};
    double world[3] = {0.,0.,0.};

    for (const geo::TPCID& tpcid: geom->IterateTPCIDs()) {
      geo::TPCGeo const& tpc = geom->TPC(tpcid);
      unsigned short not03 = (tpcid.TPC % 4);
      if(not03 == 0 || not03 == 3) continue;
      tpc.LocalToWorld(local,world);
      double xx = world[0]-geom->DetHalfWidth(tpcid.TPC, tpcid.Cryostat) + fFiducialCut;
      fTickLo[tpcid.TPC] = detprop->ConvertXToTicks(xx, 2, tpcid.TPC, tpcid.Cryostat);
      xx = world[0]+geom->DetHalfWidth(tpcid.TPC, tpcid.Cryostat) - fFiducialCut;
      fTickHi[tpcid.TPC] = detprop->ConvertXToTicks(xx, 2, tpcid.TPC, tpcid.Cryostat);
      if(fTickLo[tpcid.TPC] > fTickHi[tpcid.TPC]) std::swap(fTickLo[tpcid.TPC], fTickHi[tpcid.TPC]);
      std::cout<<"TPC "<<tpcid<<" Lo "<<fTickLo[tpcid.TPC]<<" Hi "<<fTickHi[tpcid.TPC]<<"\n";
    } // tpcid
  }

  const geo::GeometryCore* geom = lar::providerFrom<geo::Geometry>();
  for (const geo::TPCID& tpcid: geom->IterateTPCIDs()) {
    unsigned short not03 = (tpcid.TPC % 4);
    if(not03 == 0 || not03 == 3) continue;
    std::cout<<"tpc "<<tpcid<<" Lo "<<fTickLo[tpcid.TPC]<<" Hi "<<fTickHi[tpcid.TPC]<<"\n";
  }

  art::ValidHandle<std::vector<recob::Cluster>> clsVecHandle = evt.getValidHandle<std::vector<recob::Cluster>>(fClusterModuleLabel);

  for(unsigned int icl = 0; icl < clsVecHandle->size(); ++icl) {
    art::Ptr<recob::Cluster> cls = art::Ptr<recob::Cluster>(clsVecHandle, icl);
    // only consider the collection plane
    if(cls->Plane().Plane != 2) continue;
    float sTick = cls->StartTick();
    float eTick = cls->EndTick();
    if(sTick > eTick) std::swap(sTick, eTick);
    if(cls->StartTick() > fTickLo[cls->Plane().TPC] || cls->EndTick() < fTickHi[cls->Plane().TPC]) continue;
    std::cout<<"cls "<<icl<<" "<<cls->Plane().TPC<<" "<<(int)cls->StartTick()<<" "<<(int)cls->EndTick()<<"\n";
  } // icl

} // analyze
```

A skeleton exists

Next step: Find the rate