G4 TrackStacking and EventScoring as applied to LAr20 muons at DUSEL

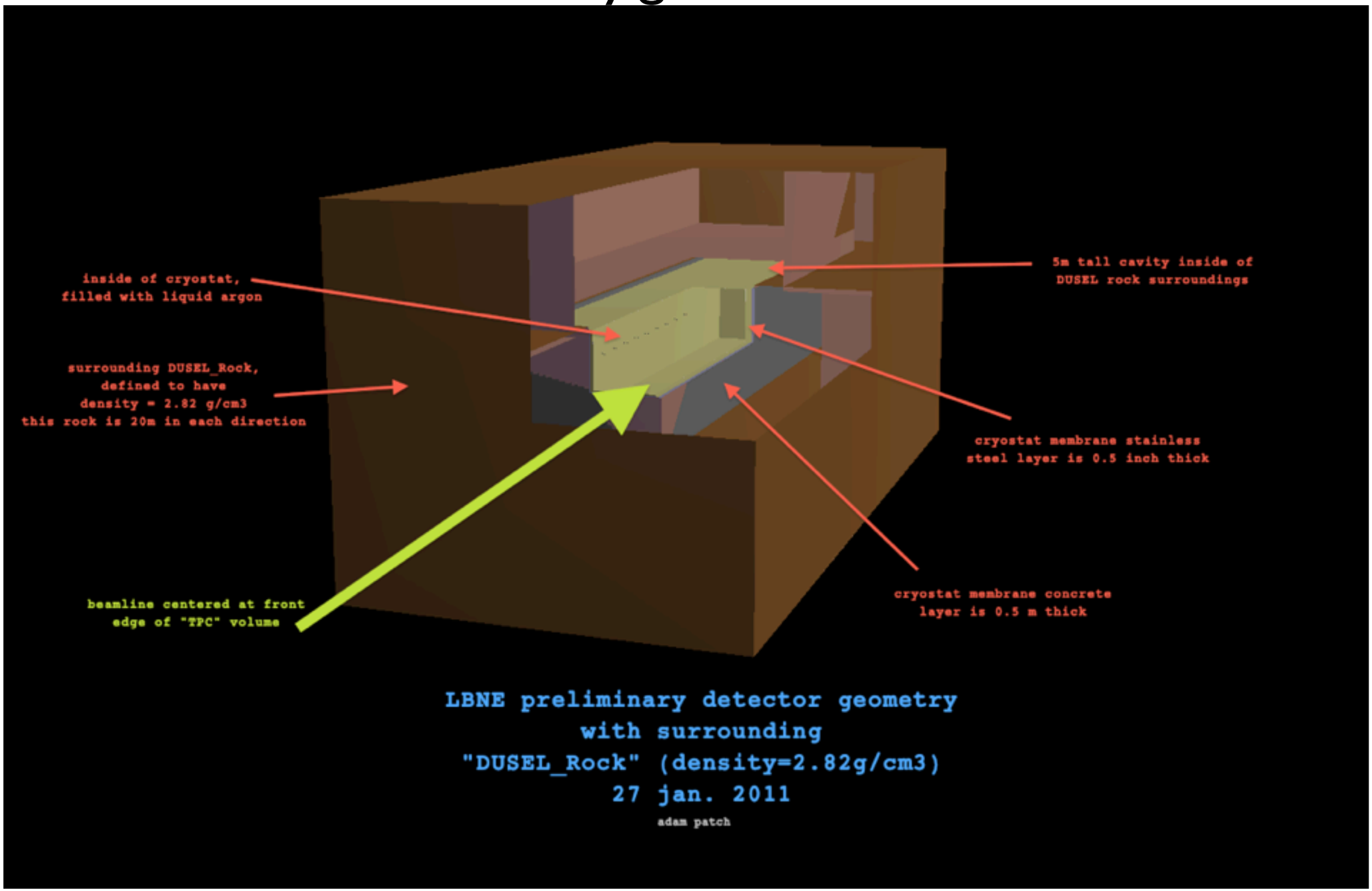
Eric Church, LArSoft Mtg, 18-May-2011

Reminder:

- Take input muons at 800' and run 'em through FileMuons/LArG4. (FileMuons is like SingleGen).
- The goal is to count how many neutrals get into LAr and how many then charge exchange to give Ndk bgds

The so-called lbne-bulky.gdml: lots of rock.

LAr: 8mx12mx70m
(9.4 ktonne?)



I use 2-3 Geant4 tricks (define trick). All in LArG4/. (#1) TrackStacking

- Track Stacking: Only track primary mu+/-.
- Only secondaries that get tracked in rock are K0_{S,L}, Lambdas, neutrons. Track them next.
- Track all secondaries in LAr
- Go and track any other primaries

Set it up in LArG4.cxx under a user switch

```
// With an enormous detector with lots of rock ala LAr34 (nee LAr20)
// we need to be smarter about stacking.
if (fSmartStacking)
 {
        G4UserStackingAction* stacking_action = new LArStackingAction;
        m_runManager->SetUserAction(stacking_action);
 }
```

```
class LArStackingAction : public G4UserStackingAction
{
  public:
    LArStackingAction();
    virtual ~LArStackingAction();

  public:
    // These 3 methods must be implemented by us. EC, 16-Feb-2011.
    virtual G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track* aTrack);
    virtual void NewStage();
    virtual void PrepareNewEvent();

  private:
    //G4bool InsideRoI(const G4Track * aTrack,G4double ang);
    std::string InsideTPC(const G4Track * aTrack);
    //G4VHitsCollection* GetCollection(G4String colName);

    //ExN04TrackerHitsCollection* trkHits;
    //ExN04MuonHitsCollection* muonHits;
    //LArStackingActionMessenger* theMessenger;

    G4int fstage;
.
.
.
}
```

G4 Comes to ClassifyNewTrack each time it produces a new track.

```
LArStackingAction::LArStackingAction()
                : fstage(0)


G4ClassificationOfNewTrack
LArStackingAction::ClassifyNewTrack(const G4Track * aTrack)
{
  G4ClassificationOfNewTrack classification = fWaiting;
  art::ServiceHandle<geo::Geometry> geom;
  TString volName(InsideTPC(aTrack));
  Double_t buffer = 2000; // Keep muNucl neutrals within 20m (for now) of larTPC.

  // These 3 for now for investigation in gdb.
  int pdg = aTrack->GetDefinition()->GetPDGEncoding();
  int ppdg = aTrack->GetParentID();
  TString process("NA");
  if (ppdg) process = (TString)aTrack->GetCreatorProcess()->GetProcessName();

  switch(fstage){
  case 0: // fstage 0 : Primary muons only
    if(aTrack->GetParentID()==0){
      G4ParticleDefinition * particleType = aTrack->GetDefinition();
      if( (particleType==G4MuonPlus::MuonPlusDefinition())
          || (particleType==G4MuonMinus::MuonMinusDefinition())){
              classification = fUrgent;
      }
    }
    if (volName.Contains("unknown"))   classification = fKill;
    break;
  case 1: // Stage 1 : K0,Lambda,n's made urgent here.
          //          Suspended tracks will be sent to the waiting stack
    if ((aTrack->GetDefinition()->GetPDGEncoding()==2112 || aTrack->GetDefinition()->GetPDGEncoding()==130 || aTrack->GetDefinition()->GetPDGEncoding()
==310 || aTrack->GetDefinition()->GetPDGEncoding()==311 || aTrack->GetDefinition()->GetPDGEncoding()==3122 ) && (aTrack->GetParentID()==1) && !
volName.Contains("unknown"))
      {
          classification = fUrgent;
      }....
```

So basically we want to classify as fUrgent things we want tracked first. fKill tracks we never want to track. fWait tracks we want to come back to.

```
void LArStackingAction::NewStage()
{

  // Here when Urgent stack is empty. Waiting stack about to be made Urgent,
  // upon saying ReClassify().
  fstage++;

    // I yanked the ExN04's use here of stackManager->clear(), which clears stack
    // and prepares to end the event. Think I may wanna do something like this if
    // muon has been tracked, its doca is large, there are no hit voxels in the TPC,
    // and I'm in further need of optimization.

  if(fstage==1){
  // Stage 0->1 : check for interesting rock neutrals

    stackManager->ReClassify();
    return;
  }

  else if(fstage==2){
  // Stage 1->2 : check for other primaries.
    stackManager->ReClassify();
    return;
  }

  else{
  // Other stage change : just re-classify
    stackManager->ReClassify();
  }
}

void LArStackingAction::PrepareNewEvent()
{
  fstage = 0;
  //trkHits = 0;
  //muonHits = 0;
}
```

G4 trick #2: Event Biasing/Scoring

- re-toss N secondaries of specified processes and weigh those tracks as 1/N.
- This is really my only hope to get even a few K0s/Lambdas into the LAr.

```
class MuNuclearSplittingProcess : public G4WrapperProcess {
// Override PostStepDoIt method
  public:
    MuNuclearSplittingProcess() {};
    ~MuNuclearSplittingProcess() {};

    void SetNSplit(G4int nTrx) {fNSplit = nTrx;};
    void SetIsActive(G4bool doIt) {fActive = doIt;};

  private:
// Data members
    G4int fNSplit;
    G4bool fActive;
    G4VParticleChange* PostStepDoIt(const G4Track& track, const G4Step& step);

};
```

```
G4VParticleChange* MuNuclearSplittingProcess::PostStepDoIt(const G4Track& track, const G4Step& step)
{

  G4VParticleChange* particleChange = new G4VParticleChange();
  G4double weight = track.GetWeight()/fNSplit;
  std::vector<G4Track*> secondaries; // Secondary store

  // Loop over PostStepDoIt method to generate multiple secondaries.
  // The point is for each value of ii up to Nsplit we want to re-toss
  // all secondaries for the muNucl process. Each toss gives back numSec
  // secondaries, which will be a different sample of species/momenta each time.
  for (unsigned int ii=0; ii<fNSplit; ii++) {
    particleChange = pRegProcess->PostStepDoIt(track, step);
    assert (0 != particleChange);
    G4int j(0);
    G4int numSec(particleChange->GetNumberOfSecondaries());
    for (j=0; j<numSec; j++)
      {
            G4Track* newSec = new G4Track(*(particleChange->GetSecondary(j)));

              if/else ..SLIDE 14 ....
               {
                      secondaries.push_back(newSec);
               }


            }
      }
  }

  particleChange->SetNumberOfSecondaries(secondaries.size());
  particleChange->SetSecondaryWeightByProcess(true);
  int numSecAdd = secondaries.size();
  std::vector<G4Track*>::iterator iter = secondaries.begin(); // Add all secondaries
  while (iter != secondaries.end()) {
    G4Track* myTrack = *iter;
    G4String pdgstr = myTrack->GetParticleDefinition()->GetParticleName();
    G4double ke = myTrack->GetKineticEnergy()/GeV;
    G4int pdg = myTrack->GetParticleDefinition()->GetPDGEncoding();

    myTrack->SetWeight(weight);
    particleChange->AddSecondary(myTrack);
    iter++;
  }
  return particleChange;
}
```

The key step

Create (push_back) N secondaries, all re-tossed from the relevant process. Then Add them, appropriately weighted.

Register Wrapped Process In LArG4/PhysicsList.cxx:

If we want K0Bias (Re-tossing of secondaries) Register WrappedProcess with this G4MuonNuclear.

```
        if (!PhysicsName.compare("SynchrotronAndGN") && lgp->K0Bias())
         {
                genSecondaries = true;
         }


    }
    if (genSecondaries AND muon)
     {
        G4int nSecondaries(lgp->K0Bias());
        mf::LogInfo("PhysicsList: ") << "Turning on WrappedMuNuclear for " << particle->GetParticleName() << "s with " << nSecondaries << " appropriately weighted secondaries.";
        G4MuNuclearInteraction* g4MNI = new G4MuNuclearInteraction();
        MuNuclearSplittingProcess* munuclSplitting = new MuNuclearSplittingProcess();
        G4bool active(true);

        munuclSplitting->SetNSplit(nSecondaries);
        munuclSplitting->SetIsActive(active);
        munuclSplitting->RegisterProcess(g4MNI);
        pmanager->AddProcess(munuclSplitting,-1,-1,3);
     }
```

3rd G4 "Trick": turn on particular physics processes

- I turned on G4MuNuclearInteraction (a subprocess of G4ExtraEM). This is what produces muon photo-dissociation (aka spallation) K0s,Lambdas
- I turned on ChargeExchange for K0s,Lambdas. Already on for neutrons. This is how we'll get pdk bgd.

In LArG4/ConfigurablePhysicsList.icc:

If our PhysicsList includes SynchrotronAndGN (G4ExtraEM) turn On MuNuclear

```
        if (!PhysicsName.compare("SynchrotronAndGN"))
{
  std::cout<<"ConfigurablePhysicsList : " << PhysicsName << ": Turning on MuNuclear, GammaNuclear, Synchrotron." << std::endl;
        G4String on("on");
        ((G4EmExtraPhysics*)g4v)->MuonNuclear(on);
        ((G4EmExtraPhysics*)g4v)->GammaNuclear(on);
        ((G4EmExtraPhysics*)g4v)->Synch(on);

}
```

In LArG4/PhysicsList.cxx:

If our PhysicsList includes ChargeExchange, actually turn it on (for other than protons/neutrons).

```
if (!PhysicsName.compare("ChargeExchange"))
 {
        mf::LogInfo("PhysicsList: ") << "Turning on ChargeExchange for " << particle->GetParticleName() << "s.";
        G4ChargeExchange* model = new G4ChargeExchange();
        G4ChargeExchangeProcess* p = new G4ChargeExchangeProcess();
        //p->AddDataSet(new G4UElasticCrossSection(particle));
        p->RegisterMe(model);
        pmanager->AddDiscreteProcess(p);

 }
```

Swap in K0_Ls

- Turn 50% of K^0_S into K^0_L

```
G4String pdgstr = newSec->GetParticleDefinition()->GetParticleName();
G4double ke = newSec->GetKineticEnergy()/GeV;
G4int pdg = newSec->GetParticleDefinition()->GetPDGEncoding();
if (abs(pdg)==310 ||abs(pdg)==311 || abs(pdg)==3122 || abs(pdg)==2112)
 {
  //            std::cout << "MuNuclSplProc: Creating " << pdgstr << " of Kinetic Energy " << ke << " GeV. numSec is " << j << std::endl;

  if (pdg==G4KaonZeroShort::KaonZeroShort()->GetPDGEncoding()&&G4UniformRand()<0.50)
   {
        pdg = G4KaonZeroLong::KaonZeroLong()->GetPDGEncoding();
        pdgstr = G4KaonZeroLong::KaonZeroLong()->GetParticleName();
        G4LorentzVector pK0L(newSec->GetMomentum(),TMath::Sqrt(TMath::Power(G4KaonZeroLong::KaonZeroLong()->GetPDGMass(),2)+TMath::Power(newSec->GetMomentum().mag(),2)));
        G4DynamicParticle *newK0L = new G4DynamicParticle(G4KaonZeroLong::KaonZeroLong(),pK0L);

        G4Track* newSecK0L = new G4Track(newK0L,track.GetGlobalTime(),track.GetPosition());
        secondaries.push_back(newSecK0L);
   }
```