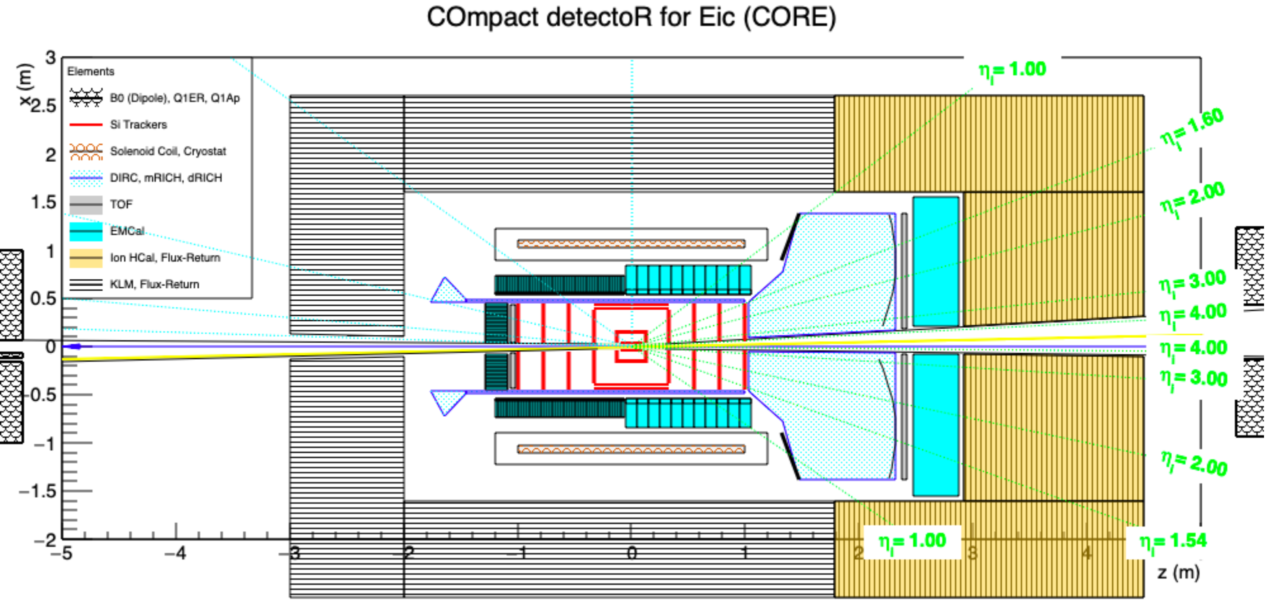
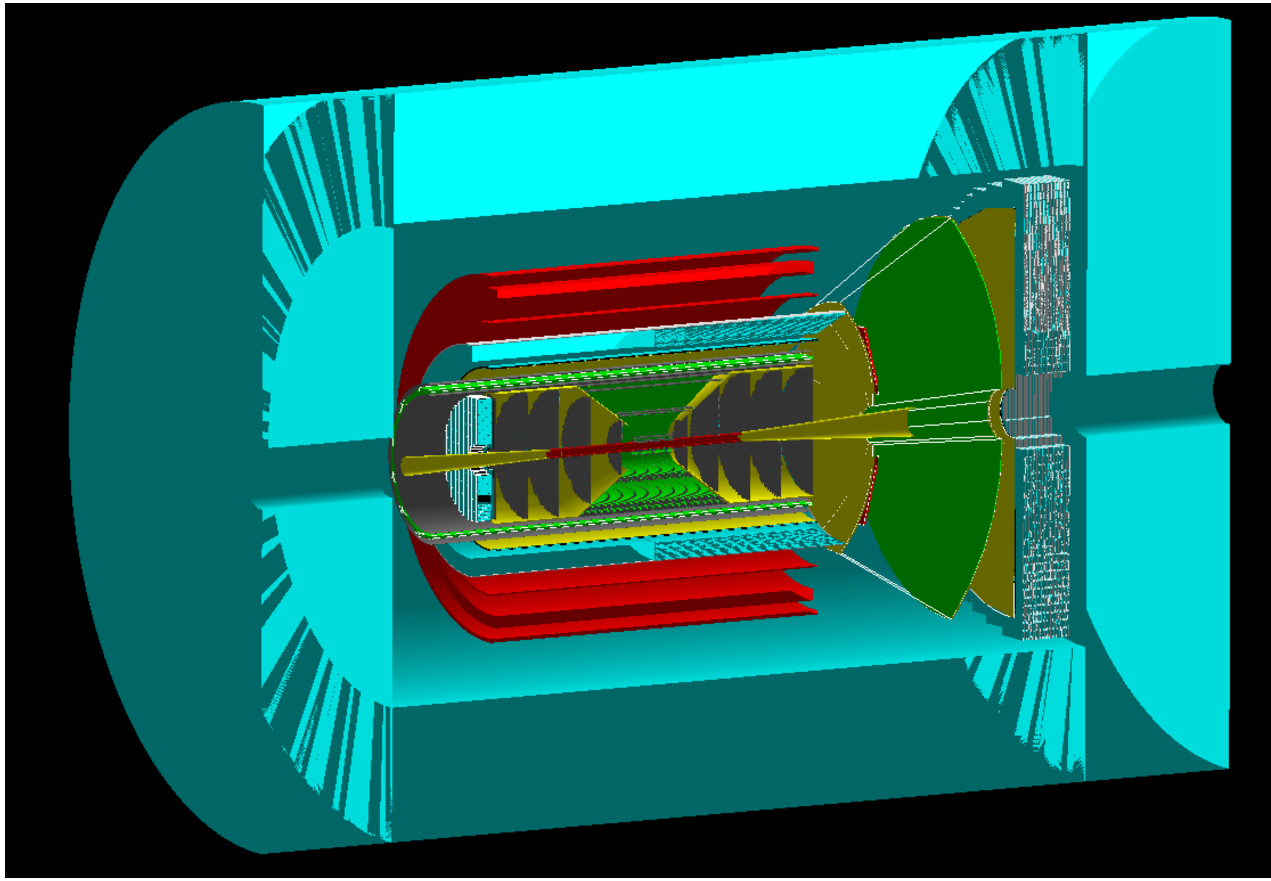
Fun4All Simulation – CORE central detector

Barak Schmookler

CORE implementation in Fun4All





Code availability

- Code is available here:

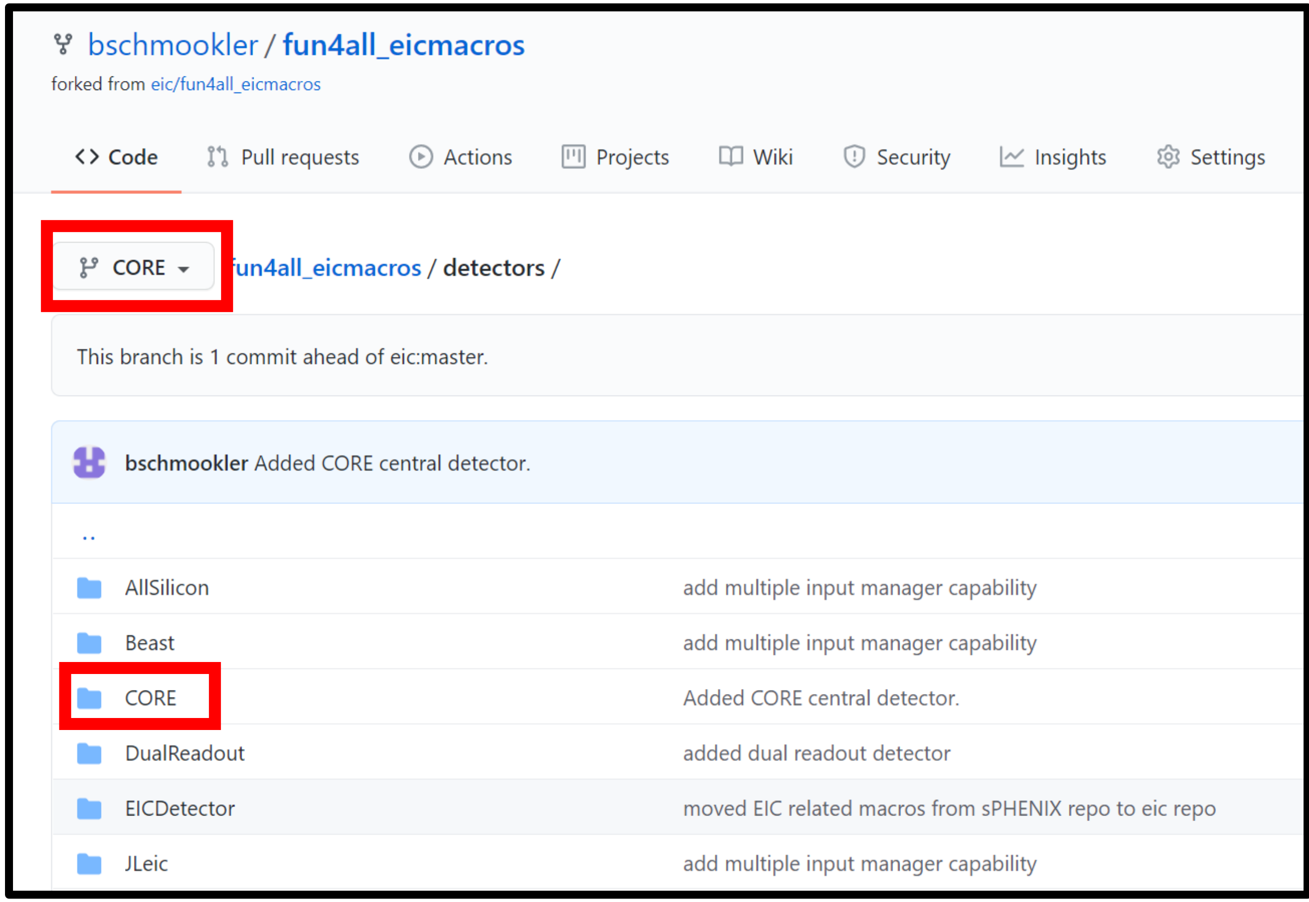
  https://github.com/bschmookler/fun4all_eicmacros/tree/CORE/detectors

- Some information for setting up the environment to run the code – either on the BNL or Jefferson Lab machines or on the singularity container – can be found in the README file here:

  https://github.com/JeffersonLab/dis-reconstruction

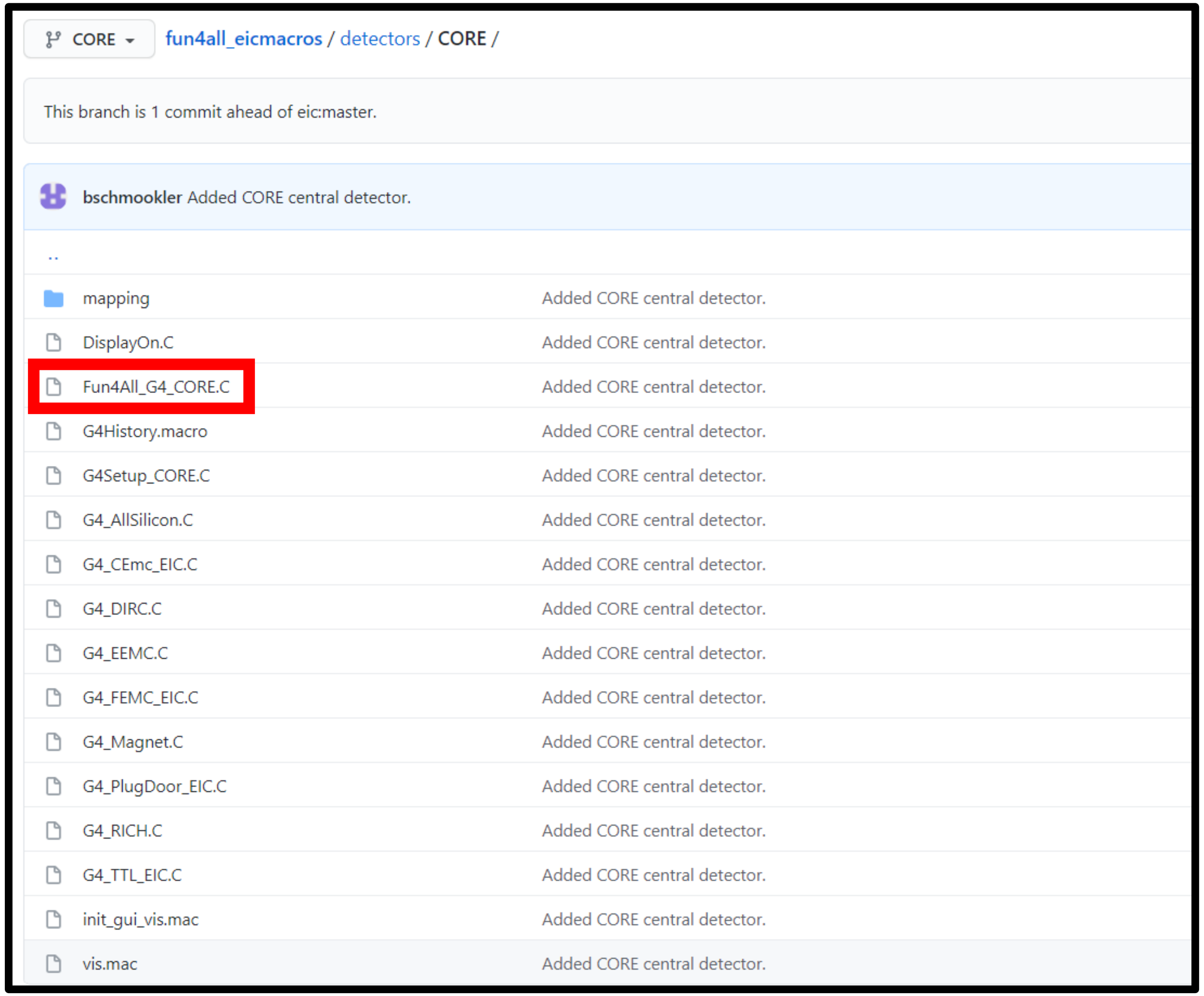
- More complete information can be found in this tutorial:

  https://indico.bnl.gov/event/7281/



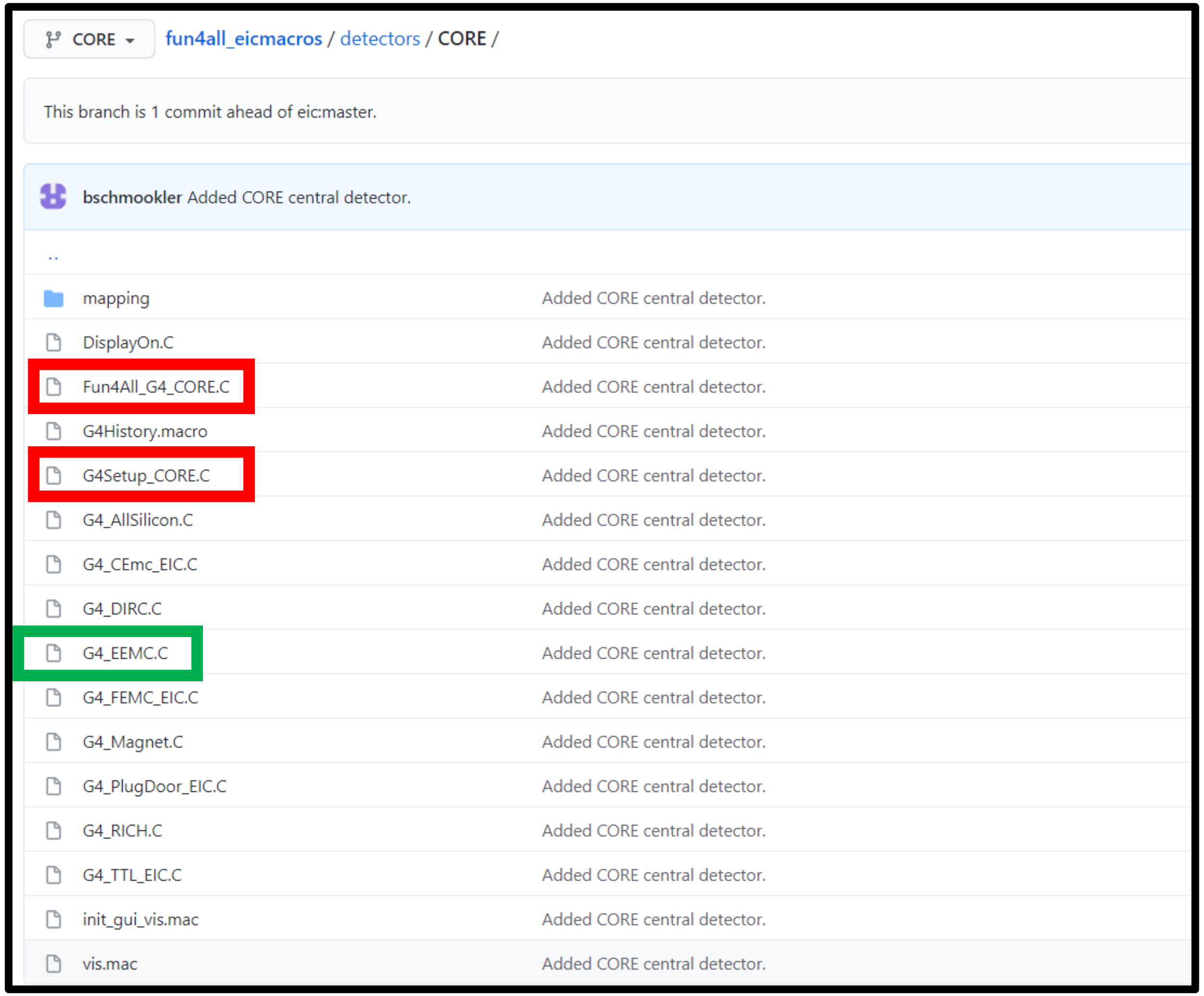
Running the simulation

- To run the simulation, simply do
  root -l 'Fun4All_G4_CORE.C(1000)'
- This will run 1000 events through the detector.
- The *Fun4All_G4_CORE.C* script is where we define the event generator to run (or read-in if the events were previously generated), the beam parameters (e.g., crossing angle) to use, the detectors and magnetic fields to use, and the output ROOT files we want to create.



Running the simulation

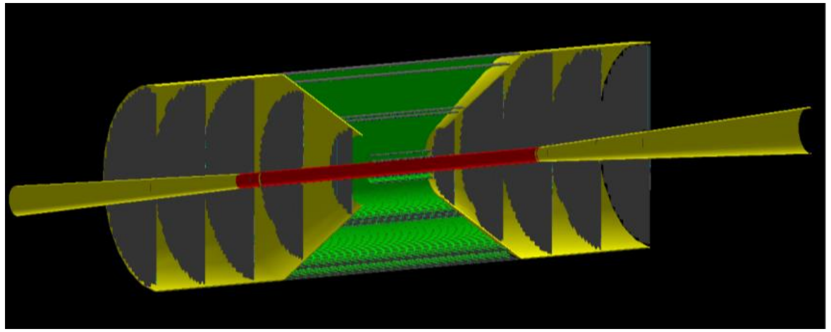
- ➢As an example, to include the electron-side electromagnetic calorimeter, we need to set **Enable::EEMC = true;** in *Fun4All_G4_CORE.C*
- ➢And we need to include *G4_EEMC.C* in *G4Setup_CORE.C*
- ➢In order to use the EEMC evaluator – to see tower hits and energy clusters in an output ROOT files– we also need to set **Enable::EEMC_EVAL = true;**



All-silicon tracking detector

- The all-silicon tracker is defined in *G4_AllSilicon.C*
- The tracker geometry is defined in a .gdml file that is read-in by the script.
- Track hits are saved for the active layers. The track reconstruction is done in the code *fun4all_eicmacros/common/G4_Tracking_LBL.C*
- The tracking performed is ‘fast tracking’ – the hits on the individual layers are smeared according to a set position resolution and then a Kalman filter is applied.

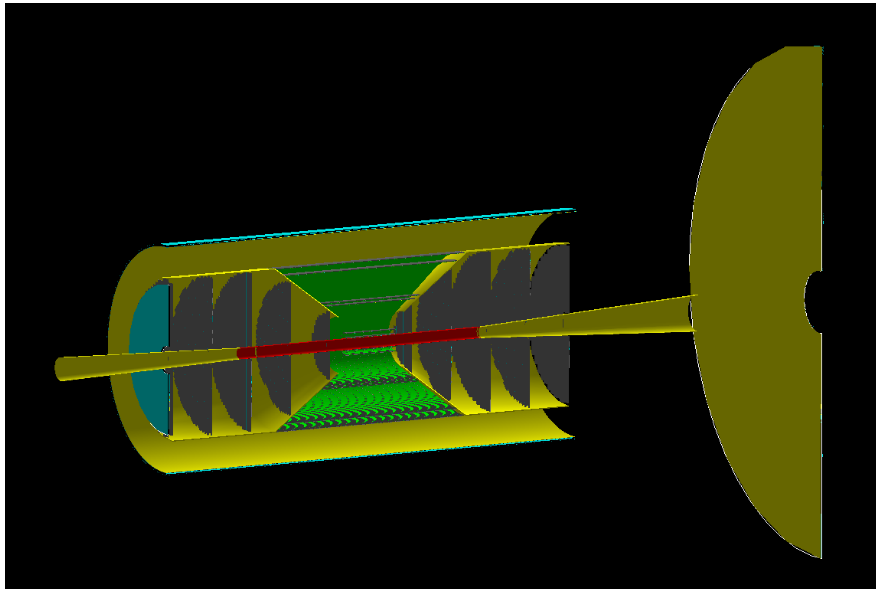
```
void AllSiliconSetup(PHG4Reco *g4Reco)
{
  bool AbsorberActive = Enable::ABSORBER || Enable::ALLSILICON_ABSORBER;
  bool OverlapCheck = Enable::OVERLAPCHECK || Enable::ALLSILICON_OVERLAPCHECK;
  AllSiliconTrackerSubsystem *allsili = new AllSiliconTrackerSubsystem();
  allsili->set_string_param("GDMPath", string(getenv("CALIBRATIONROOT")) + "/AllSiliconTracker/genfitGeom_AllSi_v2.gdml");

  allsili->AddAssemblyVolume("VST");       // Barrel
  allsili->AddAssemblyVolume("FST");       // Forward disks
  allsili->AddAssemblyVolume("BST");       // Backward disks
  allsili->AddAssemblyVolume("BEAMPIPE");  // Beampipe
  allsili->SuperDetector("LBLVTX");
  allsili->OverlapCheck(OverlapCheck);
  allsili->SetActive();                          // this saves hits in the MimosaCore volumes
  if (AbsorberActive) allsili->SetAbsorberActive();  // this saves hits in all volumes (in the absorber node)
  g4Reco->registerSubsystem(allsili);
}
```



Additional Silicon (or GEM) detectors

- Additional silicon sensors (can be easily replaced by GEMs) are implemented in *G4_TTL_EIC.C*
- The sensors are called *FTTL* (forward), *ETTL* (electron-side), and *CTTL* (central).
- If we can specify position resolutions, the hits can be added to the fast-track reconstruction module.

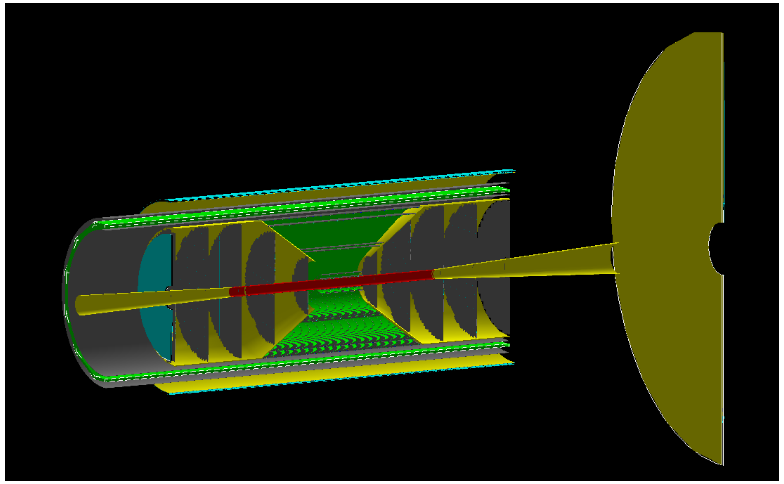
```
ttl->get_geometry().AddLayer("SiliconSensor", "G4_Si", tSilicon, true, 100);
ttl->get_geometry().AddLayer("Metalconnection", "G4_Al", 100 * um, false, 100);
ttl->get_geometry().AddLayer("HDI", "G4_KAPTON", 20 * um, false, 100);
ttl->get_geometry().AddLayer("Cooling", "G4_WATER", 100 * um, false, 100);
ttl->get_geometry().AddLayer("Support", "G4_GRAPHITE", 50 * um, false, 100);
ttl->get_geometry().AddLayer("Support_Gap", "G4_AIR", 1 * cm, false, 100);
ttl->get_geometry().AddLayer("Support2", "G4_GRAPHITE", 50 * um, false, 100);
```



DIRC geometry

- The geometry of the DIRC is implemented in *G4_DIRC.C*
- The geometry is implemented in 3 parts: the 12-sector quartz radiator, the aluminum inner skin, and the aluminum outer skin.
- No read-out or evaluator modules have been implemented yet.

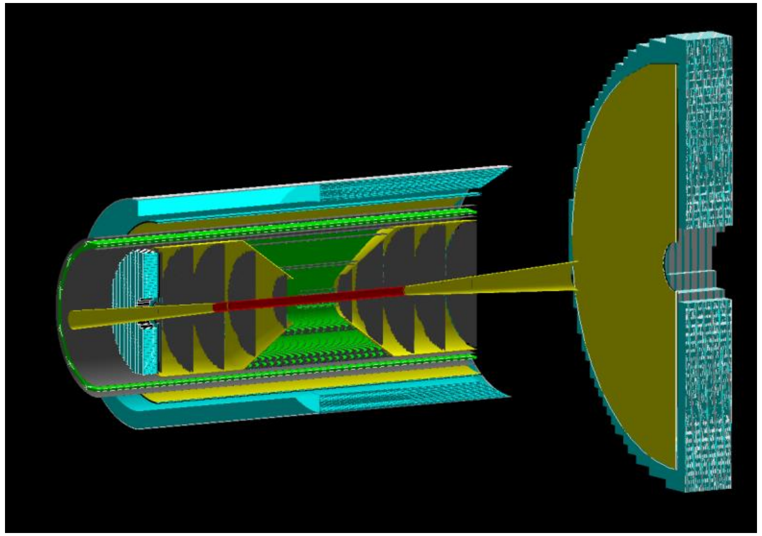
```
PHG4SectorSubsystem *dirc;
dirc = new PHG4SectorSubsystem("DIRC");
dirc->get_geometry().set_normal_polar_angle(M_PI / 2);
dirc->get_geometry().set_normal_start(G4DIRC::radiator_R * PHG4Sector::Sector_Geometry::Unit_cm());
dirc->get_geometry().set_min_polar_angle(atan2(G4DIRC::radiator_R, G4DIRC::z_start));
dirc->get_geometry().set_max_polar_angle(atan2(G4DIRC::radiator_R, G4DIRC::z_end));
dirc->get_geometry().set_min_polar_edge(PHG4Sector::Sector_Geometry::FlatEdge());
dirc->get_geometry().set_max_polar_edge(PHG4Sector::Sector_Geometry::FlatEdge());
dirc->get_geometry().set_material("Quartz");
dirc->get_geometry().set_N_Sector(12);
dirc->OverlapCheck(OverlapCheck);
dirc->get_geometry().AddLayer("Radiator", "Quartz", 1.7 * PHG4Sector::Sector_Geometry::Unit_cm(), true);
g4Reco->registerSubsystem(dirc);
```



Electromagnetic calorimeters

- The forward and electron-side electromagnetic calorimeters are implemented in *G4_FEMC_EIC.C* and *G4_EEMC.C,* respectively.
- These scripts read-in the calorimeter geometry from a text file in the *mapping* subdirectory. Those text files are created by a simple ROOT macro which defines the calorimeter position, inner and outer radius, and tower geometry.
- The barrel calorimeter is defined in *G4_CEmc_EIC.C.* The geometry is defined directly in the script. The inner radius is 65 cm and the thickness is 11.8 cm. The negative η side is PbWO4 and extends down to z = 140 cm. The positive η side consists of 40 layers: PMMA as the active part, and W powder absorber.
- For all three of these calorimeters, tower digitization and clustering algorithms are defined in the above scripts.

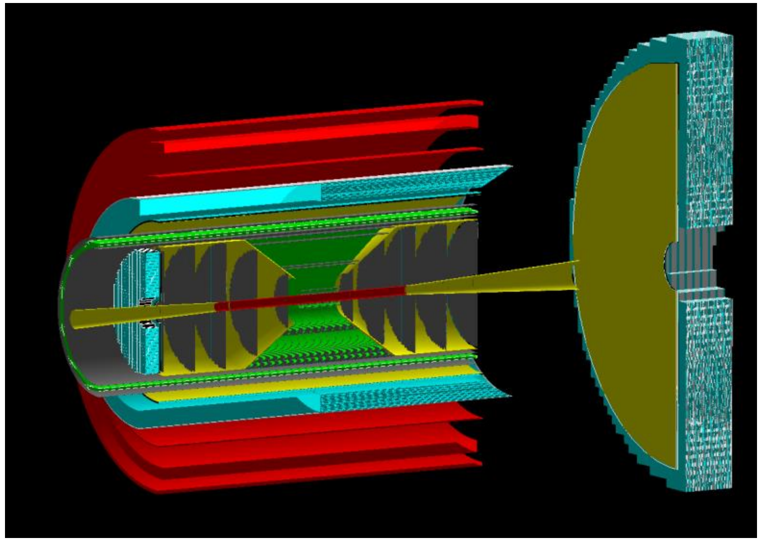
```
RawTowerBuilderByHitIndex *tower_EEMC = new RawTowerBuilderByHitIndex("TowerBuilder_EEMC");
tower_EEMC->Detector("EEMC");
tower_EEMC->set_sim_tower_node_prefix("SIM");
tower_EEMC->GeometryTableFile(mapping_eemc.str());

se->registerSubsystem(tower_EEMC);

/* Calorimeter digitization */

// CMS lead tungstate barrel ECAL at 18 degree centrigrade: 4.5 photoelectrons per MeV
const double EEMC_photoelectron_per_GeV = 4500;

RawTowerDigitizer *TowerDigitizer_EEMC = new RawTowerDigitizer("EEMCRawTowerDigitizer");
TowerDigitizer_EEMC->Detector("EEMC");
TowerDigitizer_EEMC->Verbosity(verbosity);
TowerDigitizer_EEMC->set_raw_tower_node_prefix("RAW");
TowerDigitizer_EEMC->set_digi_algorithm(RawTowerDigitizer::kSimple_photon_digitization);
TowerDigitizer_EEMC->set_pedstal_central_ADC(0);
TowerDigitizer_EEMC->set_pedstal_width_ADC(8);  // eRD1 test beam setting
TowerDigitizer_EEMC->set_photonelec_ADC(1);     //not simulating ADC discretization error
TowerDigitizer_EEMC->set_photonelec_yield_visible_GeV(EEMC_photoelectron_per_GeV);
TowerDigitizer_EEMC->set_zero_suppression_ADC(16);  // eRD1 test beam setting
```



Magnet geometry and field

- The magnet geometry is implemented in *G4_Magnet.C*
- The magnet extends from a radius of 90 cm out to 122.5 cm. The half-length of the magnet is 125 cm. The geometry consists of an inner and outer cryostat, and magnetic coil. The material is aluminum-5083. This is (roughly) consistent with the *ZEUS* magnet.
- The magnetic field is currently implemented as a uniform solenoidal field (in *Fun4All_G4_CORE.C*) , which can be set to any strength. In the simulation, the field is implemented separately from the magnet – this is, one can remove the magnet material and keep the field.
- In *Fun4All_G4_CORE.C,* there is the option to read-in a magnetic field map.

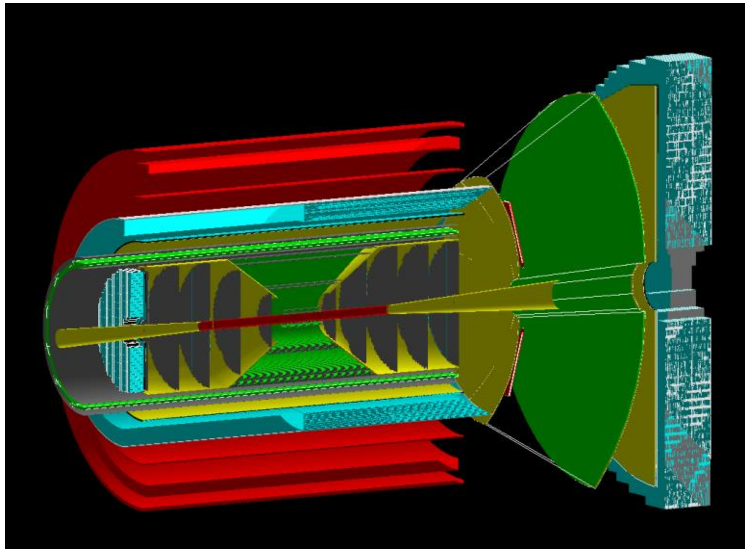
```
double Magnet(PHG4Reco* g4Reco, double radius)
{
  bool AbsorberActive = Enable::ABSORBER || Enable::MAGNET_ABSORBER;
  bool OverlapCheck = Enable::OVERLAPCHECK || Enable::MAGNET_OVERLAPCHECK;
  int verbosity = std::max(Enable::VERBOSITY, Enable::MAGNET_VERBOSITY);

  double magnet_inner_cryostat_wall_radius = 90;
  double magnet_inner_cryostat_wall_thickness = 1;
  double magnet_coil_radius = 105;
  double magnet_coil_thickness = 8.;
  double magnet_length = 250.;
  double coil_length = 242.;
  if (radius > magnet_inner_cryostat_wall_radius)
  {
    cout << "inconsistency: radius: " << radius
         << " larger than Magnet inner radius: " << magnet_inner_cryostat_wall_radius << endl;
    gSystem->Exit(-1);
  }
```



(Gas) RICH geometry

- Dual-radiator RICH with outward-reflecting mirrors should go in the forward endcap (eRD14 consortium).
- Currently, the geometry of the gas RICH from the *ePHENIX* detector is used as a placeholder. This geometry is implemented in *G4_RICH.C*

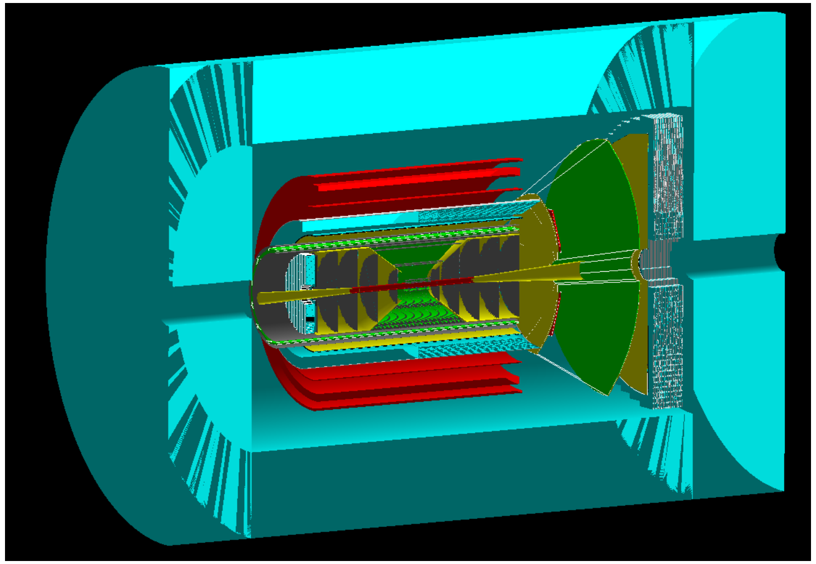
```
//! ePHENIX Gas RICH. Ref to Geometry parameter defined in ePHENIXRICH::RICH_Geometry
//! \param[in] N_RICH_Sector number of RICH sectors
//! \param[in] min_eta minimal eta coverage
//! \param[in] R_mirror_ref Radius for the reflection layer of the mirror
void RICHSetup(PHG4Reco* g4Reco,                    //
               const int N_RICH_Sector = 8,         //
               const double min_eta = 1.3,          //
               const double R_mirror_ref = 190,     //cm // Reduced from 195 (2014 LOI) ->
               const double z_shift = 75,           // cm
               const double R_shift = 18.5,         // cm
               const double R_beampipe_front = 8,   // clearance for EIC beam pipe flange
               const double R_beampipe_back = 27    // clearance for EIC beam pipe flange
)
```



Hadronic Calorimeter / Flux Return

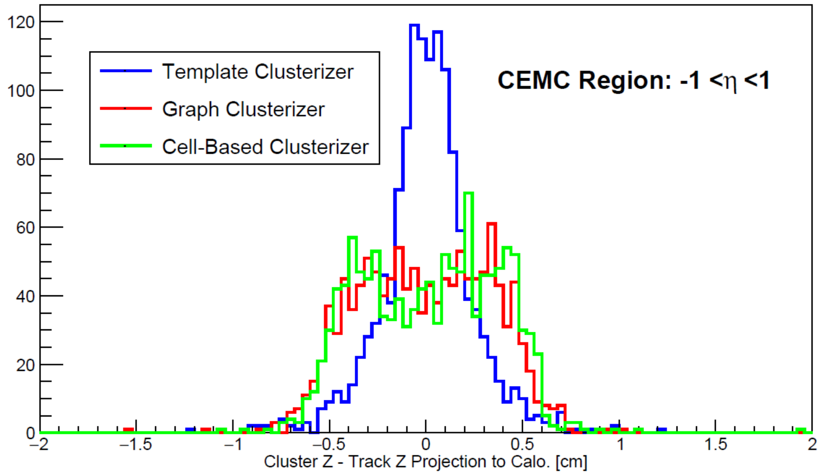
- The geometry of the hadronic calorimeter / flux return has been implemented in *G4_PlugDoor_EIC.C* with passive steel-1006.
- The material extends from z = -300 cm to z = + 440 cm. The outer-radius is 250 cm.

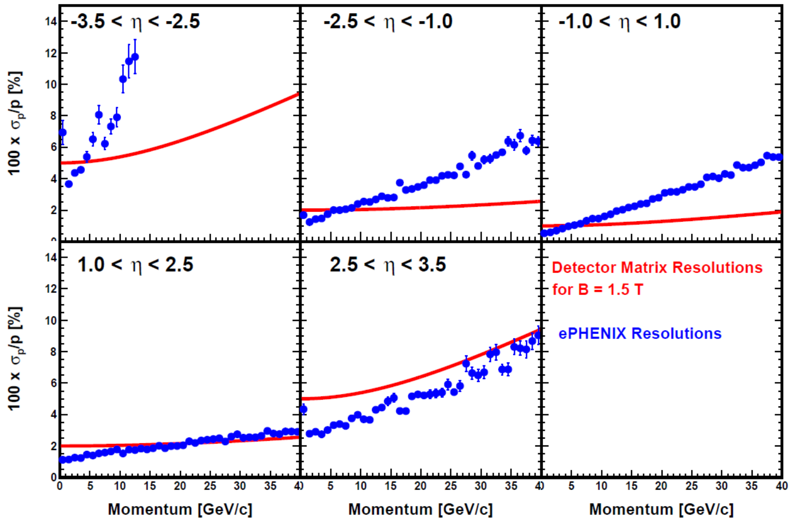
```
PHG4CylinderSubsystem *flux_return_minus = new PHG4CylinderSubsystem("FLUXRET_ETA_MINUS", 0);
flux_return_minus->set_double_param("length", G4PLUGDOOR::length_1);
flux_return_minus->set_double_param("radius", G4PLUGDOOR::r_1);
flux_return_minus->set_double_param("place_z", G4PLUGDOOR::place_z1);
flux_return_minus->set_double_param("thickness", G4PLUGDOOR::r_2 - G4PLUGDOOR::r_1);
flux_return_minus->set_string_param("material", material);
flux_return_minus->SetActive(flux_door_active);
flux_return_minus->SuperDetector("FLUXRET");
flux_return_minus->OverlapCheck(OverlapCheck);
g4Reco->registerSubsystem(flux_return_minus);
```



Example (using *ePHENIX* detector) batch farm and analysis code

```
#Go into scratch directory
chmod g+rx ${_CONDOR_SCRATCH_DIR}
cd ${_CONDOR_SCRATCH_DIR}

#Make subdirectory and move there
INPUT=$(( 0 + $1 ))
echo $INPUT
DIR=`printf "%04d" $INPUT`
mkdir $DIR
cd $DIR

#Soft links to necessary scripts
ln -s /sphenix/user/baschmoo/standard/macros/detectors/EICDetector/Fun4All_G4_EICDetector.C
ln -s /sphenix/user/baschmoo/standard/macros/detectors/EICDetector/G4Setup_EICDetector.C
ln -s /sphenix/user/baschmoo/standard/analysis/EICAnalysis/macros/diskinematics_fun4all/Fun4All_EICAnalysis_DIS.C

#Run simulation
echo "start running in directory $PWD"
echo "Running Job Number $1"
root -l -b -q 'Fun4All_G4_EICDetector.C(1000)'
root -l -b -q 'Fun4All_EICAnalysis_DIS.C(1000)'

#Move output files and cleanup
echo "Cleaning Up..."
mv -v G4EICDetector.root /sphenix/user/baschmoo/standard/macros/detectors/EICDetector/output/electron_single/ephenix_full/G4EICDetector
_${INPUT}.root
mv -v G4EICDetector_g4tracking_eval.root /sphenix/user/baschmoo/standard/macros/detectors/EICDetector/output/electron_single/ephenix_fu
ll/G4EICDetector_g4tracking_eval_${INPUT}.root
mv -v output.root /sphenix/user/baschmoo/standard/macros/detectors/EICDetector/output/electron_single/ephenix_full/analyzer_${INPUT}.ro
ot
```





*Post this and the analysis code online and add a link

Thank you!

Questions?