Please take a quick look at the slides as follow I prepared to get a basic understanding of the EMCal-related code and G4Hit in sPHENIX.

CaloBase and CaloReco

github:
sPHENIX-Collaboration/coresoftware/offline/packages/CaloBase
sPHENIX-Collaboration/coresoftware/offline/packages/CaloReco

The classes in **CaloBase** store information related to the calorimeter (calo), which can be saved and retrieved.
At the **tower level**:

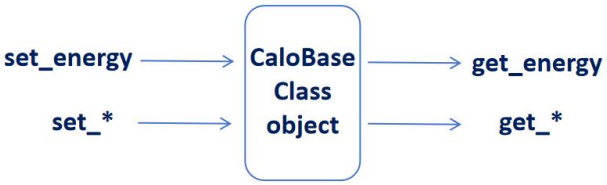
- `RawTower` stores deposited energy and other information within a tower,
- `RawTowerGeom` stores the geometry information of the tower, and
- `TowerInfo` contains internal information about the tower.

At the **cluster level**:

- `RawCluster` holds information related to clusters.

*class_Container is map of <id, *class>

...



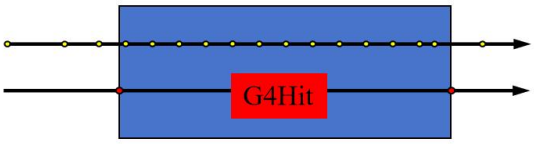
The classes in **CaloReco** are used to analyze and reconstruct calo data, and then write the processed results into the classes in **CaloBase**. For instance, `RawClusterBuilderTemplate` reads in `RawTower` and `RawTowerGeom`, and uses them to reconstruct `RawCluster`.

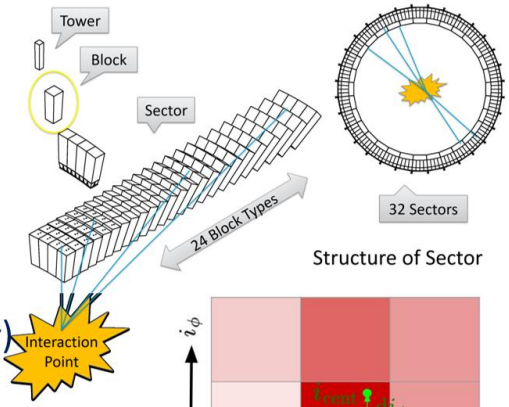
In simple terms, the classes in **CaloBase act like data-recording objects (can be set and get), while **CaloReco** is like a data-analysis macro. CaloBase serves as both the input and output for CaloReco.**

2

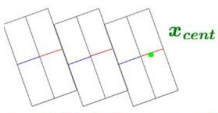
Commonly used data

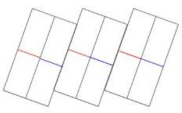
simu:
Primary Particle -- information such as the primary particle's ID, momentum, and p_xyz ...
G4hit_CEMC -- energy deposition process in the EMCal

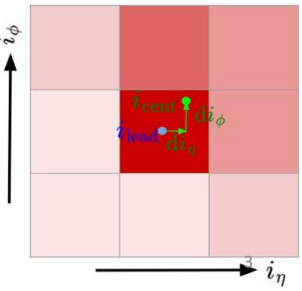




reco: raw data (wave, cell ...)
-> tower -- (deposited energy, eta phi(bin), xyz ... of tower)
-> cluster -- (deposited energy, eta phi, xyz ... of cluster)







These data are stored as CaloBase class,
And we can change the class reco by change resposed CaloReco code

where to access the data

```
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
  DST (PHCompositeNode)/
    G4HIT_HCALIN (PHIODataNode)
    G4HIT_ABSORBER_HCALIN (PHIODataNode)
    G4HIT_HCALOUT (PHIODataNode)
    G4HIT_ABSORBER_HCALOUT (PHIODataNode)
    SVTX (PHCompositeNode)/
      SvtxHitMap (PHIODataNode)
      SvtxClusterMap (PHIODataNode)
    SVTX_EVAL (PHCompositeNode)/
      SvtxClusterMap_G4HIT_SVTX_Links (PHIODataNode)
  RUN (PHCompositeNode)/
    CYLINDERGEOM_SVTX (PHIODataNode)
    CYLINDERGEOM_SVTXSUPPORT (PHIODataNode)
    CYLINDERGEOM_EMCELECTRONICS_0 (PHIODataNode)
    CYLINDERGEOM_HCALIN_SPT (PHIODataNode)
  PAR (PHCompositeNode)/
    SVTX (PHCompositeNode)/
      SvtxBeamSpot (PHIODataNode)
```

Print it from the cmd line with
Fun4AllServer *se = Fun4AllServer::instance();
se->Print("NODETREE");

DST and RUN Node: default for I/O
- •DST – eventwise
- •RUN - runwise

Objects under the DST node are reset after every event to prevent event mixing. You can select the objects to be saved in the output file. Subnodes like SVTX are saved and restored as well. DST/RUN nodes can be restored from file under other TopNodes
ROOT restrictions apply:
Objects cannot be added while running to avoid event mixing

Commonly used data in EMCal analysis mentioned on the previous page, are usually classes from CaloBase.

4

access the data we need

Please change the nodename ("nodename") at the end to the one where your data is stored,
or we can `root -l your_dstfile`, then `*tree->Print()` to display the node tree and see how you should write

usually:

get primary particle:

```
auto m_truth_info = findNode::getClass<PHG4TruthInfoContainer>(topNode, "G4TruthInfo");
```

get g4hit:

```
auto hits_CEMC = findNode::getClass<PHG4HitContainer>(topNode, "G4HIT_CEMC");
```

get calo data:

```
auto Geom_container = findNode::getClass <RawTowerGeomContainer> (topNode, "TOWERGEOM_CEMC");
auto Tower_container = findNode::getClass <RawTowerContainer> (topNode, "TOWER_SIM_CEMC");
auto EMCal_tower_calib = findNode::getClass <TowerInfoContainerv2> (topNode, "TOWERINFO_SIM_CEMC");
auto Cluster_container = findNode::getClass <RawClusterContainer> (topNode, "CLUSTER_CEMC");
```

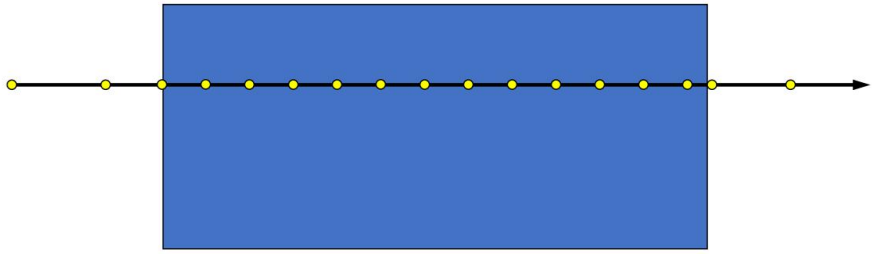
```
for (int i = 0; i < *class_container -> size(); ++i) //loop over container { *class = *class_container->get_* }
```

5

G4Hit:

GEANT steps

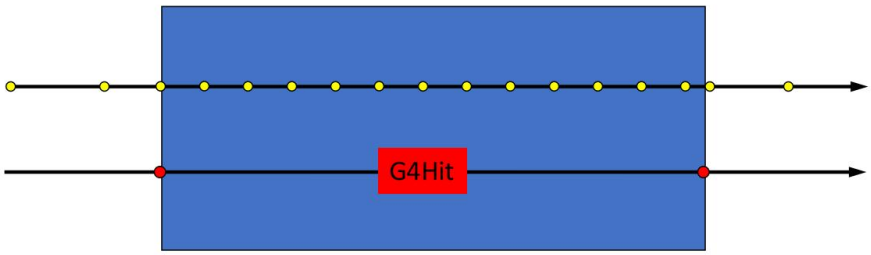
GEANT propagates particles one step at a time. The step size is determined by the physics processes associated with the current particle or when a boundary between volumes is crossed



After each step the user stepping method is called with a pointer to the current volume which has access to the full information (energy loss, particle momentum at beginning and end of step, …)

Our G4Hits (you'll hear us talking about them a lot)

In our stepping method we add the energy loss in each volume and store the entry and exit coordinates and time (and subdetector specific info like ionization energy, light output,…)



We also keep the ancestry for G4Hits so any hit can be traced back to a primary particle. To reduce size we do not store particles which do not leave G4Hits and are not in the ancestry of a particle which created a G4Hit

Goal:
Obtain the position measurement information of electrons and positrons (charged hadrons) by the EMCal.
When a charged particle strikes the EMCal, it initiates an electromagnetic shower, depositing energy within the calorimeter. The EMCal readout units are towers, and we can reconstruct the EMCal's measurement of charged particles by utilizing the information from the towers.
Moreover, in simulations, we have access to all the underlying truth information. We treat the truth-level information obtained from Geant4 as the "true" reference and compare the reconstructed results to this truth to evaluate performance and apply corrections to make the reconstruction closer to the truth.

Regarding position reconstruction for charged particles, I am using four types of position information:

Truth level: The reference code is in the CodeExplain.C file.

1. Primary particle first hit on CEMC:
This represents the position where the primary charged particle first hits the EMCal.
We first identify all primary particles and collect the G4Hits produced by these primaries (excluding those from secondary particles).
We then record the entrance position of the first G4Hit in the EMCal for each primary particle.
This entrance position is taken as the location where the primary particle hits the CEMC.

2. Shower center with energy weight:
This represents the energy-weighted center of the electromagnetic shower.
We read all G4Hits produced in the CEMC (including those from secondary particles) and compute the weighted average of their positions, using the energy deposition of each G4Hit as the weight.
This gives us the energy-weighted center of the shower.

Reco level: 3. Cluster reco with geometry center(4. with inner face center)

In our macro `Fun4All_physiTuto.C`, the `G4Setup_sPHENIX.C` file is included, and within it, the `G4_CEmc_Spacal.C` file is also included.
In `Fun4All_physiTuto`, the cluster reconstruction code is triggered by:

```
if (Enable::CEMC_CLUSTER) CEMC_Clusters();
```

This `CEMC_Clusters()` function is defined in `G4_CEmc_Spacal.C`.
The reconstruction uses the `RawClusterBuilderTemplate` module, which in turn calls the `BEmcRec` module to perform cluster reconstruction.

The process involves: Using a threshold to eliminate noisy towers, keeping only towers with significant energy deposition. Grouping adjacent towers that pass the threshold into clusters based on continuity. Using the energy of each tower as a weight to calculate the energy-weighted average position (x, y, z) of the cluster.

Relevant code references are in the `RawClusterBuilderTemplate` and `BEmcRec` modules.

Initially, the tower positions were defined by projecting the tower onto a fixed radius of 93.5 cm. However, this approach has known issues in the reconstruction.

In my study, I use the updated tower geometry description provided by Virgile:

```
// Load the modified geometry
CaloGeomMappingv2 *cgm = new CaloGeomMappingv2();
cgm->set_detector_name("CEMC");
cgm->setTowerGeomNodeName("TOWERGEOM_CEMCv3");
se->registerSubsystem(cgm);
```

With this code, we load the updated tower geometry.
The node `TOWERGEOM_CEMCv3` stores the new `RawTowerGeomv5` objects, each of which contains the eight vertices of the tower.
You can obtain the center of the tower using `get_center_x/y/z()`, or the center of the tower's **inner face** using `get_center_int_x/y/z()`.

In the tutorial, the function:

```
void setUseRawTowtTowerGeomNodeName("TOWERGEOM_CEMCv3");
```

specifies from which node the tower geometry is read — in this case, the newly registered `TOWERGEOM_CEMCv3`.

After loading the new tower geometry, we have two options for cluster position reconstruction:

3. Cluster reco with geometry center: using the tower's geometric center.

4. Cluster reco with inner face center: using the center of the tower's inner surface.

I added two functions in `RawClusterBuilderTemplate`:

```
void setUseRawTowerGeomv5(bool flag = true) { m_use_RawTowerGeomv5 =
flag; }
```

This decides whether to use the new tower geometry.

```
void setProjectToInnerSurface(bool flag = true)
{ m_project_tower_innersurface = flag; }
```

This decides whether to reconstruct cluster positions using the tower's inner face center.

In the implementation of `CEMC_Clusters()` in `G4_CEmc_Spacal.C`, I use:

```
RawClusterBuilderTemplate *ClusterBuilder = new
RawClusterBuilderTemplate("EmcRawClusterBuilderTemplate");
ClusterBuilder->setUseRawTowerGeomv5(true);
ClusterBuilder->setProjectToInnerSurface(true);
```

To use the new EMCal tower geometry setup and others I modified, you need to replace the analysis modules in **CaloBase** and **CaloReco** mentioned below with my updated code. You can access the code under **/sphenix/user/jzhang1/Virgikguide**

Then, compile the **tutorial class** under **/sphenix/user/jzhang1/INTT-EMCAL/InttSeedingTrackDev/ParticleGen/physiTuto** , and after that, you can run **Fun4All_physiTuto.C** from within the **/sphenix/user/jzhang1/INTT-EMCAL/InttSeedingTrackDev/ParticleGen/macro .**

Since all of these depend on RawTowerGeom on CaloBase and Reconstruction on CaloReco, please compile CaloBase directory first, then compile CaloReco directory, and finally compile your analysis module class.

RawTowerGeomv5.h & RawTowerGeomv5.cc

These files are used to store the descriptions of the eight vertices of each tower, along with some simple functions to obtain the centers of certain surfaces.

CaloGeomMappingv2.h & CaloGeomMappingv2.cc

CaloGeomMappingv2 is used to load the new, accurate EMCal geometry and store it within RawTowerGeomv5.

RawClusterBuilderTemplate.h & RawClusterBuilderTemplate.cc

```
void setUseRawTowerGeomv5( bool flag = true) { m_use_RawTowerGeomv5 = flag; }
void setProjectToInnerSurface( bool flag = true ) { m_project_tower_innersurface = flag; }
```

Modifications have been made so that the cluster reconstruction uses the settings from RawTowerGeomv5. Additionally, two small interfaces are provided to choose which point of the tower is used for reconstruction (either the inner surface or the center of the tower).

CaloGeomTest.h & CaloGeomTest.cc

These files simply output the geometry settings for verification purposes and are not particularly useful beyond that.

Fun4All_physiTuto.C

```
// Load the modified geometry
    CaloGeomMappingv2 *cgm = new CaloGeomMappingv2();
    cgm->set_detector_name("CEMC");
    cgm->setTowerGeomNodeName("TOWERGEOM_CEMCv3");
    se->registerSubsystem(cgm);

    // tutorial info get
    tutorial* analysis_module = new tutorial( "name", output_directory, output_filename);
    analysis_module->setTowerGeomNodeName("TOWERGEOM_CEMCv3");
```

This code loads the new geometry settings. In Fun4All, the included header `<G4Setup_sPHENIX.C>` (which in turn includes `"G4_CEmc_Spacal.C"`) provides the source for `CEMC_Clusters()`. Therefore, modifications were made in `G4_CEmc_Spacal.C` within the `CEMC_Clusters()` function to enable the use of the new geometry for cluster reconstruction. The `tutorial` module is used to read the information, and it has also been modified accordingly.

tutorial.h & tutorial.cc

```
prepareEMCal(topNode);
prepareEMCalClus(topNode);

prepareG4Turth(topNode);
prepareG4HIT(topNode);
```

These functions handle the preparation of EMCal reconstruction and now include G4-level information such as primary particles and G4 hits.

G4_CEmc_Spacal.C

```
// JingyuM
void CEMC_Clusters()
```

Within the CEMC_Clusters() function mentioned above, modifications have been made for cluster reconstruction. Specifically, the reconstruction now makes use of both the inner surface center and the overall tower center of each tower.