Fun4all

Where The Fun Always Shines!

Tutorial Structure

- Short introduction to Fun4All
- Example 0: Implementing Jaroslav's Luminosity Monitor with least modifications
- Using Fun4All for real
- Example1: Implementing simple Detector in Fun4All
- Example 2: Momentum resolution of configurable Si+TPC
- Example 3: Adding your detector to a full detector implementation
- List of other things you can do with it
- HELP!!!!!
- Future

What is assumed

- Basic C++ knowledge (what is a base class and virtual methods)
- Basic familiarity with root (what is a root macro, drawing histograms from ntuples)
  - Currently ROOT5 and ROOT6 are supported with ROOT5 being phased out
- Can run our software, either via your rcf account or on your laptop/desktop:
  https://github.com/EIC-Detector/Singularity
- For running under JLab iFarm accounts:
  https://eic-detector.github.io/tutorials_example2_JLab.html
- Some familiarity with git (what is a repository, git clone)
- You definitely want a github account

Brief History of Fun4All

- Development started in 2002, in use by PHENIX from 2003 on (reconstruction and analysis of Run3 data)
- Needed to get many subsystems who developed their code independently and without coordination under one umbrella
- Development driven by reconstruction and analysis needs – not by “beauty” or some abstract design considerations (or “rules”)
- Modularity is key – components can be added/modified/removed without having to modify bits and pieces all over the place
- Plenty of functionality added over the years, some of them waiting to be rediscovered
- 2011 Adding Geant4 as subsystem
- Split from PHENIX in 2015, lots of cleanup and modernization

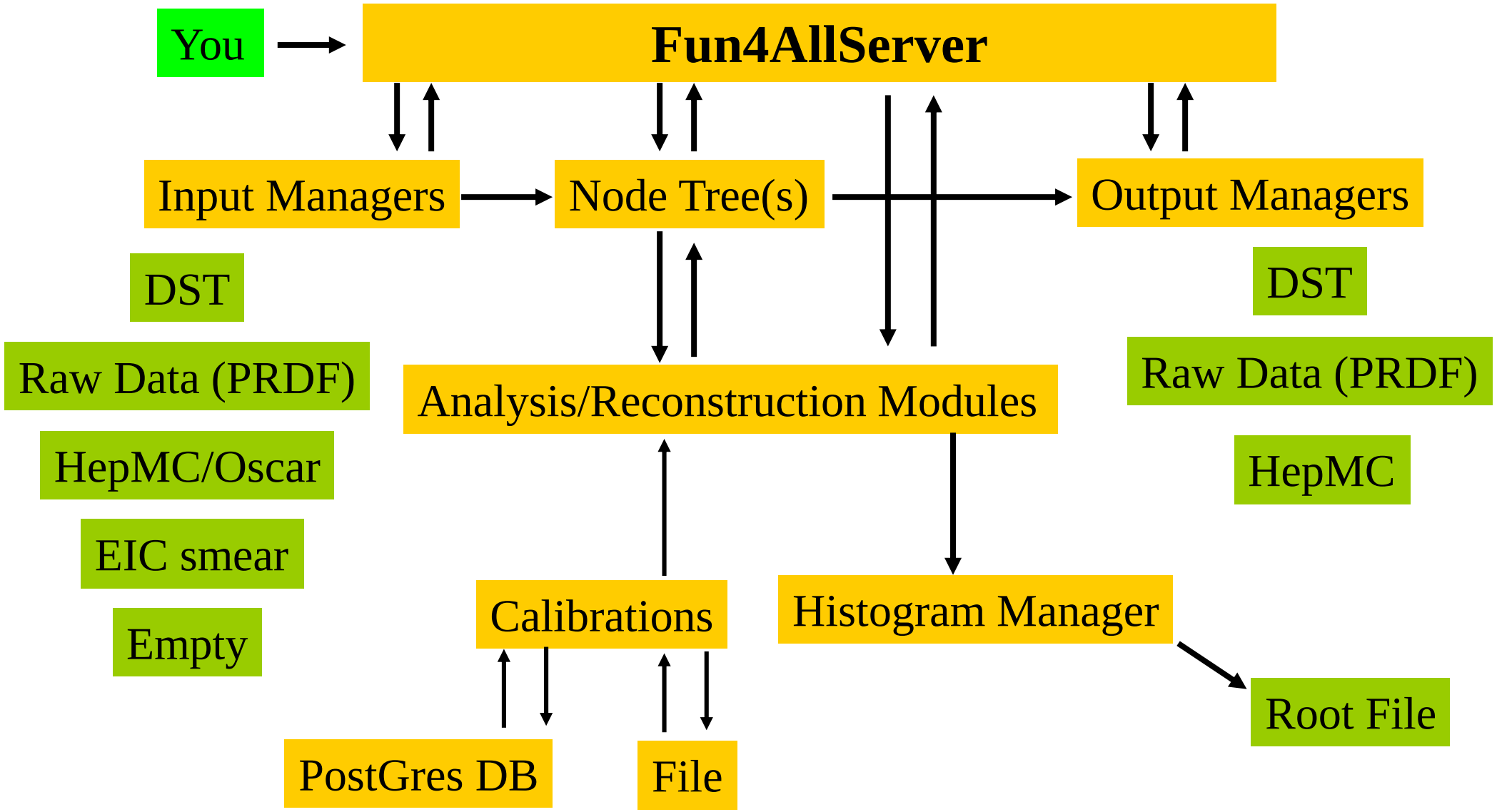
Code in
https://github.com/sPHENIX-Collaboration/coresoftware/tree/master/offline/framework

What does it do for us

- Read input files (many different types)
- Write output files (different types, automatic saving of selected data objects)
- Somewhat manages the Node Tree - our storage for data objects (more later)
- Call the analysis/reconstruction modules in the order in which they were registered (correct ordering is responsibility of the user)
- Make snapshots at any state of the reconstruction/analysis
- Access to calibrations

Fun4All has been our workhorse for 16 years, running raw data reconstruction, analysis, simulations and embedding

Structure of our framework Fun4All



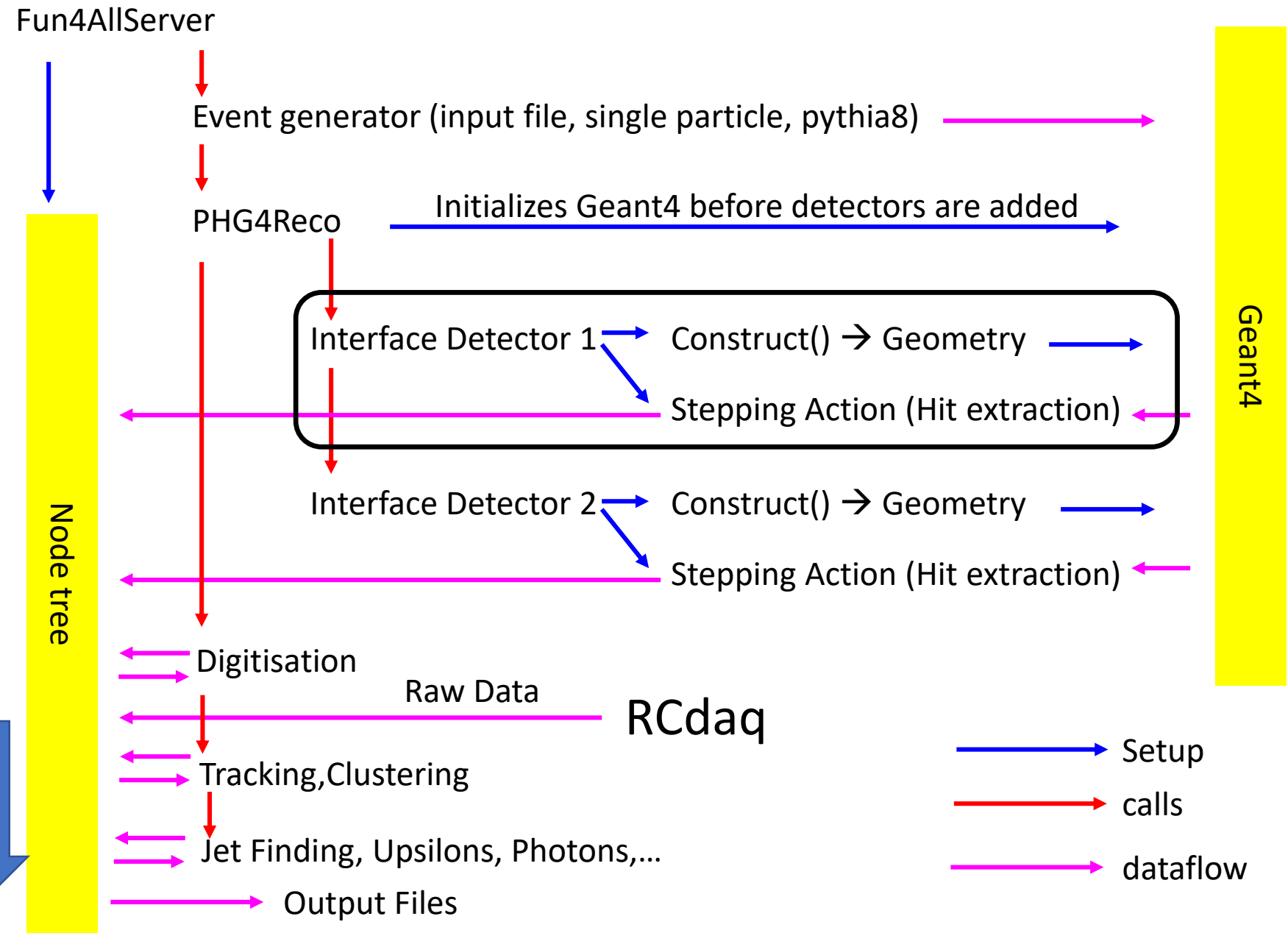
That's all there is to it, no backdoor communications – steered by ROOT macros

Simulations

- GEANT4 based (no GEANT3, no virtual Monte Carlo)
- Fully integrated into our analysis chain as Reconstruction Module
- Modular design – all detectors are self contained
- Configured with Root macros
- Generic detectors (boxes, cylinders, cones) exist
- EIC-sPHENIX subdetectors fully implemented
- 8 years of development
- Used for published EIC detector concepts:
  - https://arxiv.org/pdf/1402.1209.pdf (2104)
  - An EIC Detector Build Around The sPHENIX Solenoid (2018)

Tutorials:
https://github.com/sPHENIX-Collaboration/tutorials

G4 program flow within Fun4All



The processing is done by chaining up modules. At every step the state of the Node Tree can be saved and the analysis can pick up where it left

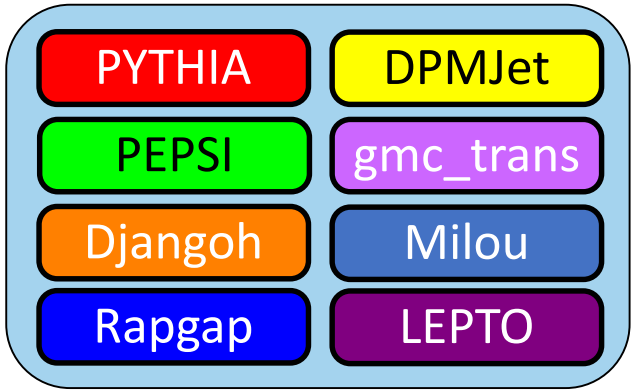
Common reco for raw and simulated data

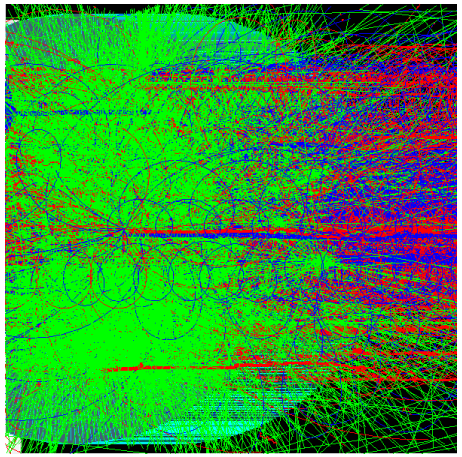
Modular: Each detector is its own entity providing the flexibility needed for complex setups

Generic detectors like boxes, cylinders, cones exist and can be configured on a macro level

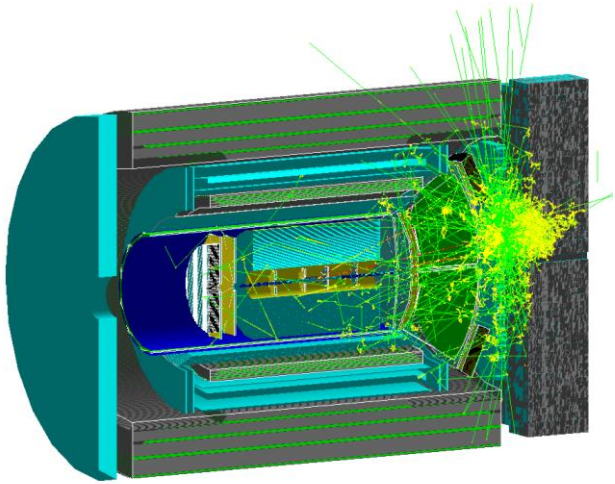
Currently implemented event generators

- Via Eic-Smear interface (e-p/e-A)



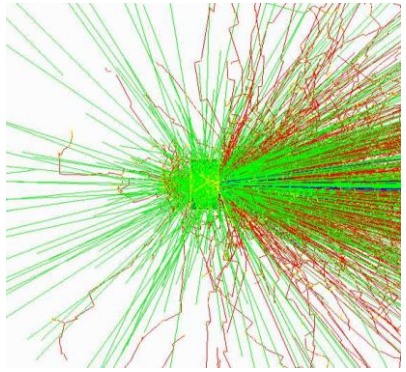


Very peripheral heavy ion collision from hijing

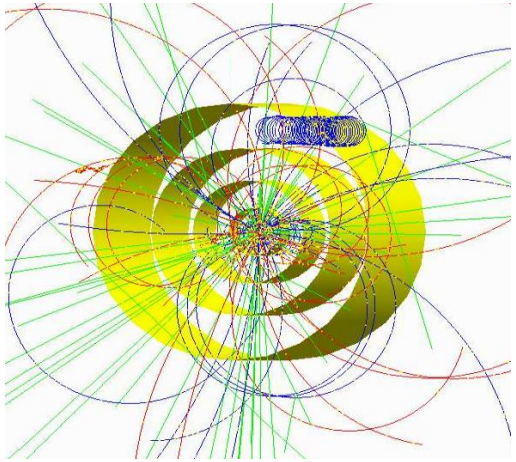


Sarte as seen by EIC detector

- Via Fun4All
  - Hijing
  - Pythia6
  - Pythia8
  - Sartre
  - Jetscape (needs work)
  - Generic HepMC 2/OSCAR
  - Single Particle Generators
  - Embedding in existing events



10 GeV Au on water phantom (NSRL)



Pythia8 in a 6 layer silicon detector mockup and 2T field

Full Truth information preserved: anything can be traced back to the input particle

Distribution

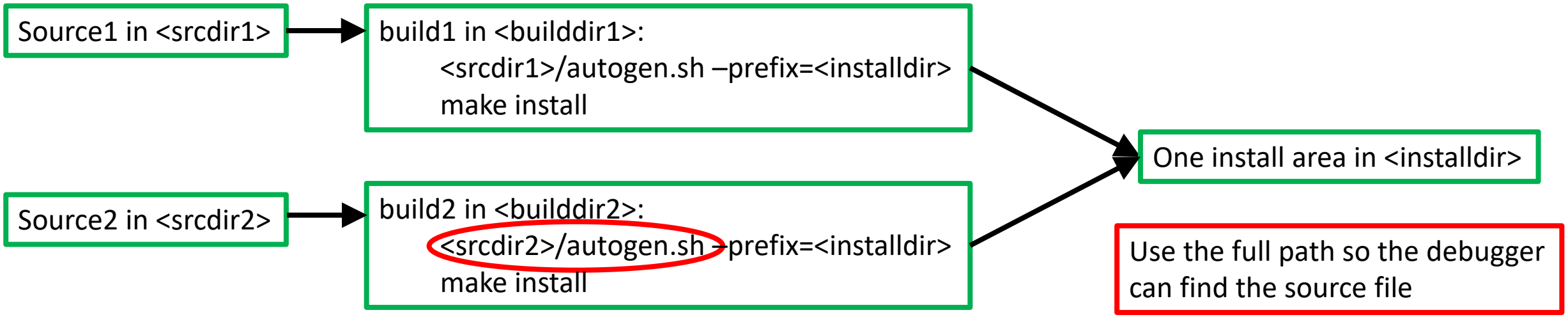
- OS image (rcf SL7 distribution) in singularity container
- Libraries and 3rd party packages installed in cvmfs – central maintenance of packages
- Dedicated EIC cvmfs volume: /cvmfs/eic.opensciencegrid.org

- Set up environment:
  source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/eic_setup.sh -n
- Set up local install area:
  source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/setup_local.sh <install area>

How to build a package

- We use autoconf/automake (configure) to build our code
  - This does put some files into your source area, be careful what you commit
- Each package (directory) is build by itself
  - You only have to build the package you are working on

Keep your source, build and install areas separately, but use common install area



If you use a private install area, you need to tweak your environment (.csh for cshell):
source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/setup_local.sh <install area>

Example 0: Jaroslav's Luminosity Monitor

Project: GEANT4 based simulation, uses hand made ROOT TTrees to save output
Goal: Keep code changes to an absolute minimum, not re-implement using Fun4All features
Approach: Using Fun4All-G4 Detector as simple wrapper

Changes needed:

- removal of detector and sensitive detector inheritance
- storing of physical volumes for lookup in stepping action
- propagating ROOT TTree downstream
- replace DetectorConstruction.cxx by G4LmonDetector interface
- Dropped G4 messenger
- Drop input reader (replace by eic-smear interface?)
- 20 lines changed in cxx files, 8 in include files (+some missing declarations)

Original code:
https://github.com/adamjaro/lmon

Fun4All based Code:
https://github.com/EIC-Detector/Fun4All-lmon

Example 0: Jaroslav's Luminosity Monitor

Cut and paste commands from https://www.phenix.bnl.gov/WWW/publish/phnxbld/EIC/tutorial/example0.html

Shell commands:

```
source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/eic_setup.sh -n
git clone https://github.com/EIC-Detector/Fun4All-lmon
cd Fun4All-lmon/source
mkdir build
cd build
../autogen.sh -prefix=$HOME/myinstall
make install
source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/setup_local.sh
$HOME/myinstall
cd ../../macros
root.exe
```

At the Root prompt:

```
.x Fun4All_Lmon.C(-1)
.L Display.C
PHG4Reco *g4 = QTGui();
```

And on the G4 cmd line:

```
/Fun4All/run 1
```

Alt F7 to move large G4 window in windows virtual box

Or at the Root prompt:

```
.x Fun4All_Lmon.C(-1)
.L DisplayOn.C
PHG4Reco *g4 = DisplayOn();
Fun4AllServer *se = Fun4AllServer::instance();
se->run(1);
```

Hint: Root understands tab expansion

What’s in it for Jaroslav?

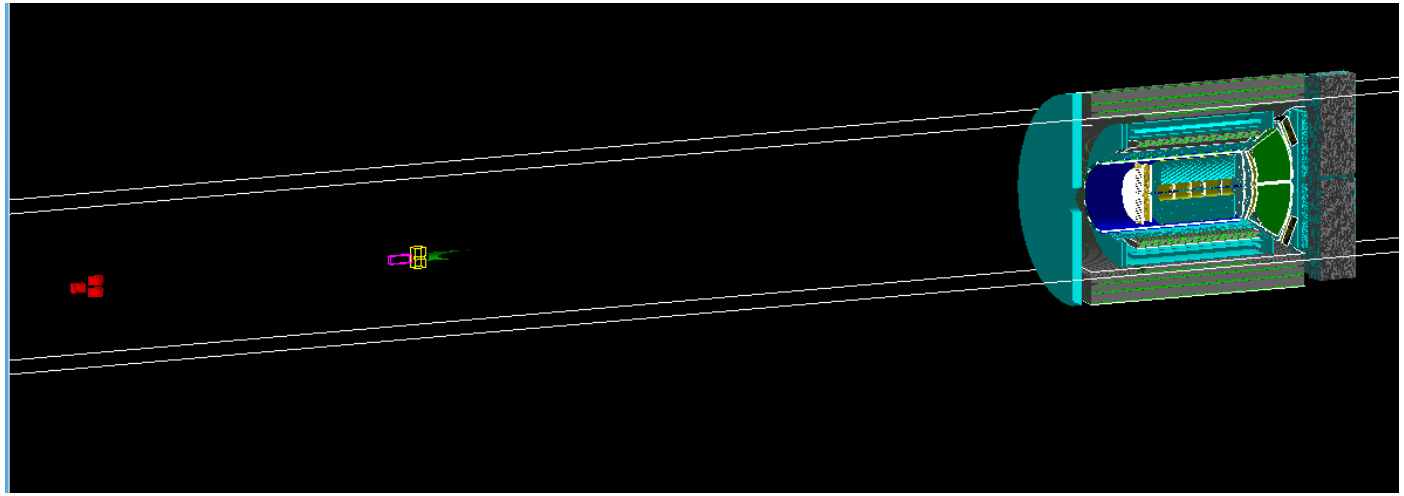
- Access to all implemented generators
- He can keep using everything he developed downstream, no changes to his output
- Less code to maintain

What’s in it for Jaroslav?

- Access to all implemented generators
- He can keep using everything he developed downstream, no changes to his output
- Less code to maintain

What's in it for Jaroslav?

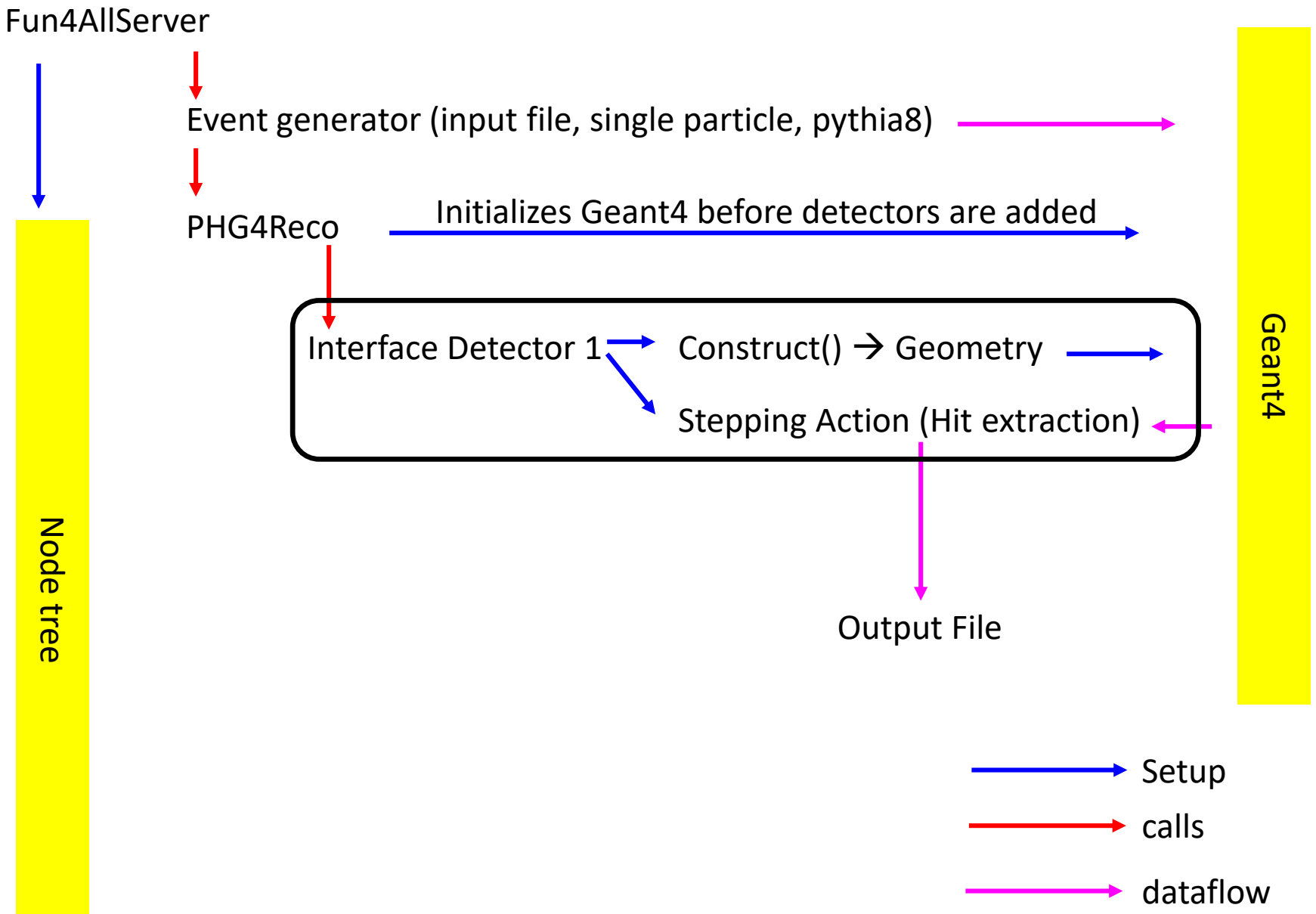
He can start playing with the big kids, detectors are added on the macro level, **no change in any other code needed**



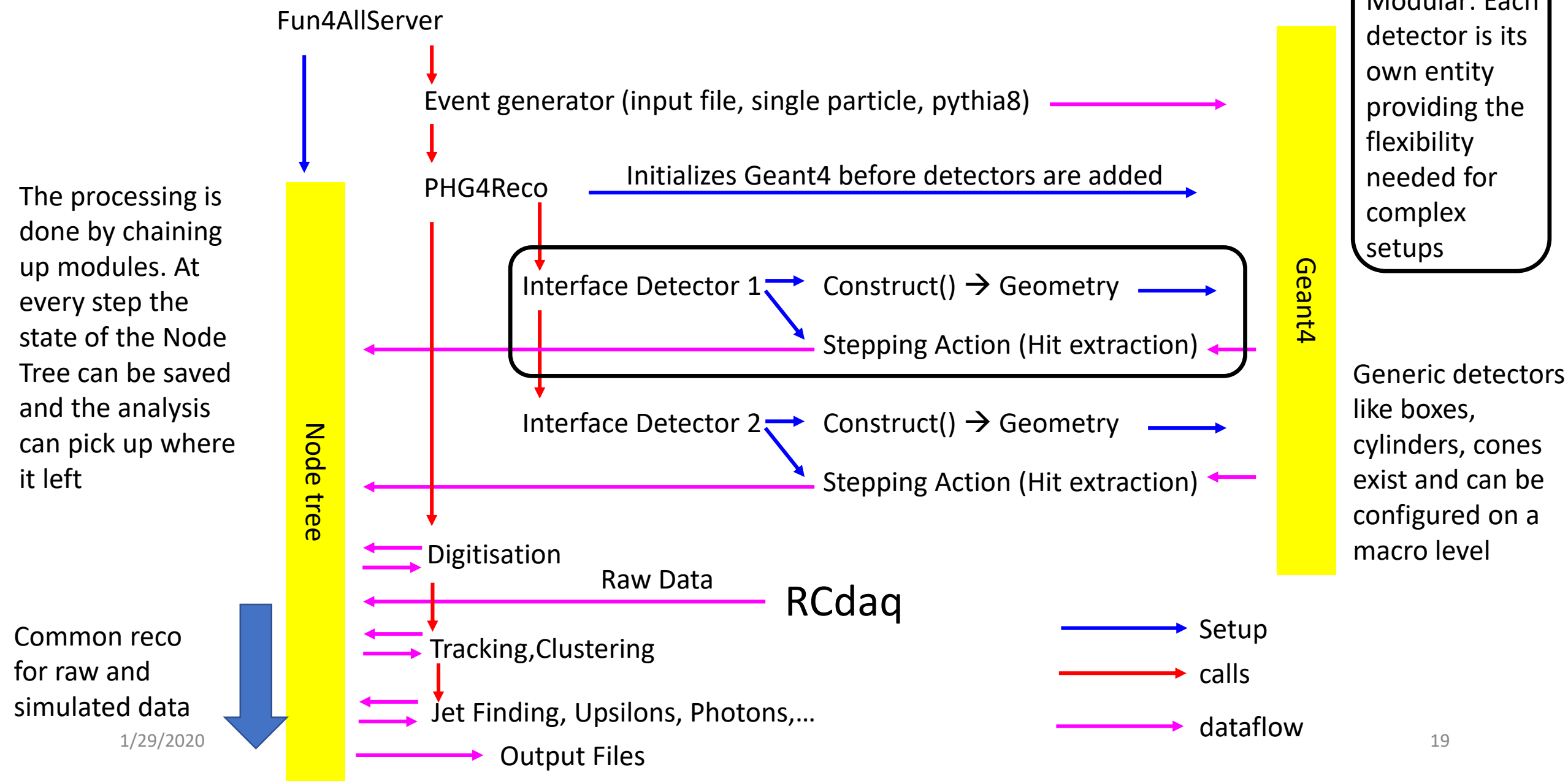
The fine print:
The stepping action does not put any objects on our node tree
This does not include the detector into any analysis chain
There are overlaps (the vacuum box) which need to be dealt with
Our world is filled with G4_Air, need to add a vacuum system

Using Fun4All for Real

G4 program flow Example 0



G4 program flow within Fun4All



Modular: Each detector is its own entity providing the flexibility needed for complex setups

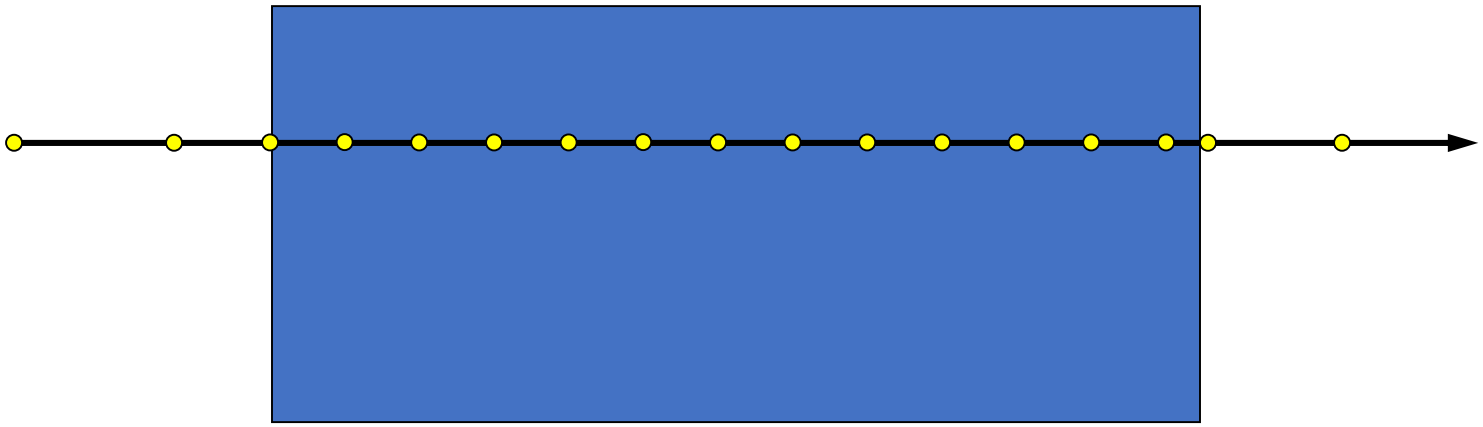
The processing is done by chaining up modules. At every step the state of the Node Tree can be saved and the analysis can pick up where it left

Generic detectors like boxes, cylinders, cones exist and can be configured on a macro level

Common reco for raw and simulated data

GEANT steps

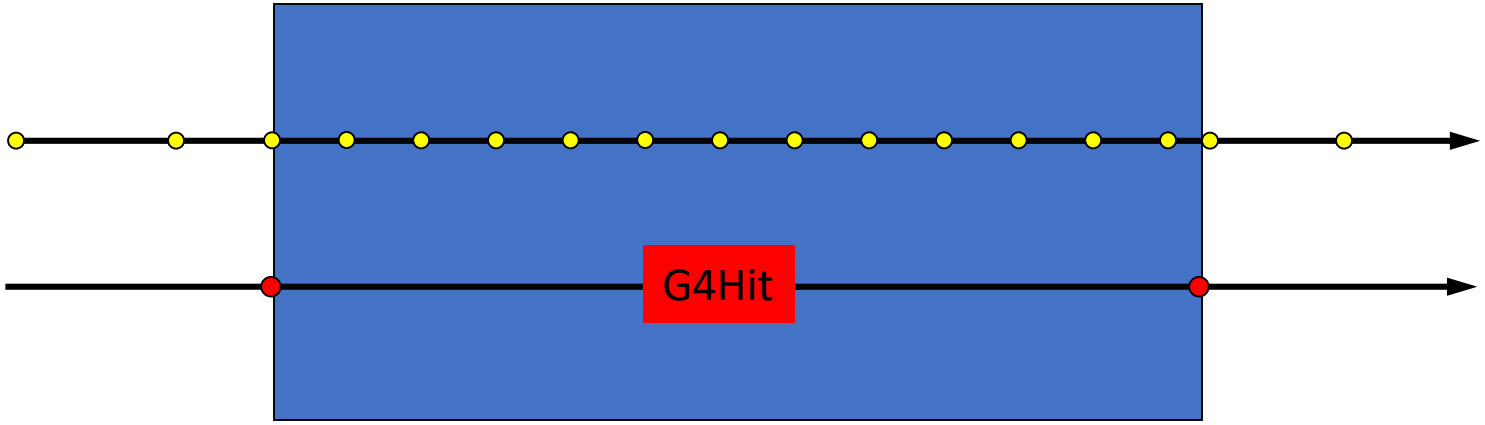
GEANT propagates particles one step at a time. The step size is determined by the physics processes associated with the current particle or when a boundary between volumes is crossed



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, …)

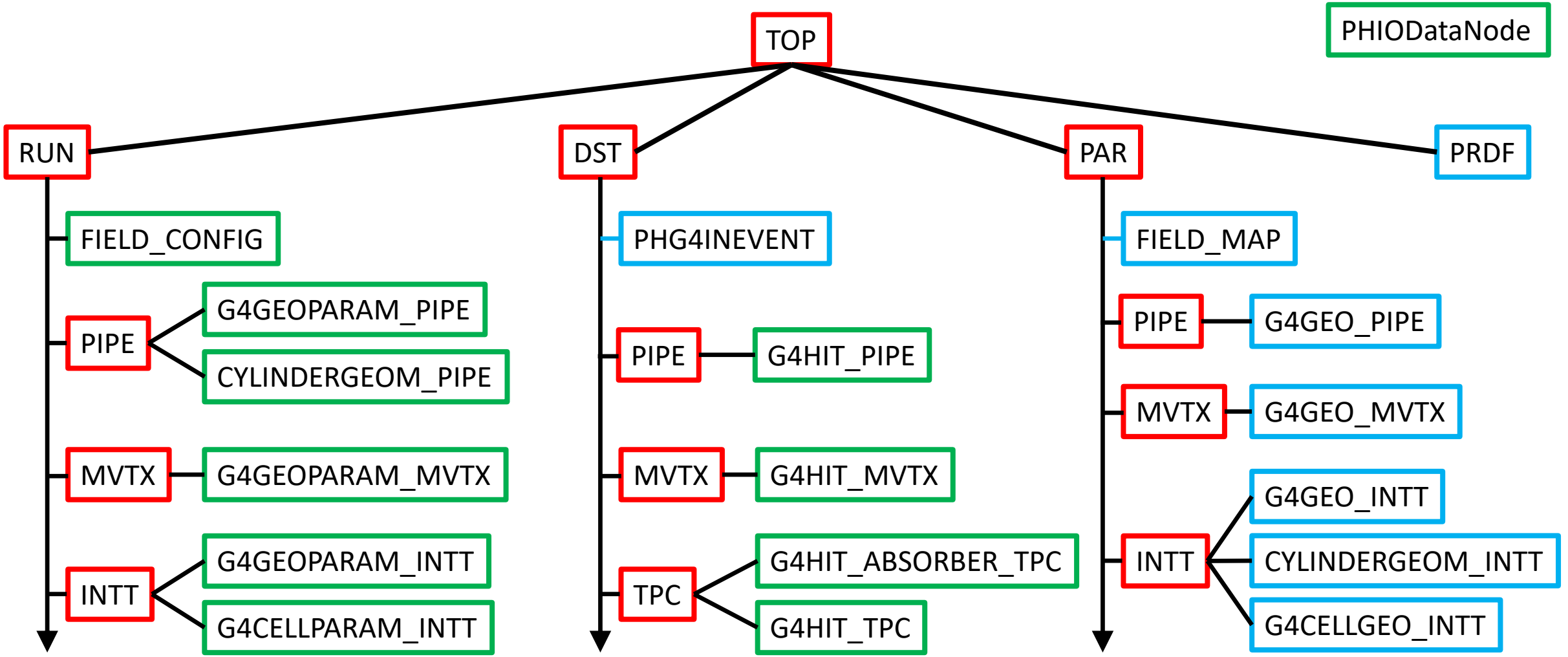
Our G4Hits (you'll hear us talking about them a lot)

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

Access to Data Objects, understanding our Node Tree



The Node Tree

- The Node Tree is at the center of the Fun4All software universe (but it's more or less invisible to you). It's the way we organize our data and make them accessible to modules
- **It is NOT a Root TTree**
- We have 3 different Types of Nodes:
  - PHCompositeNode: contains other Nodes
  - PHDataNode: contains any object
  - PHIODataNode: contains objects which can be written out to DST
- PHCompositeNodes and PHIODataNodes can be saved to a DST and read back
- This DST contains root TTrees, the node structure is saved in the branch names. Due to Roots limitations not all objects can become PHIODataNodes (e.g. anything containing BOOST). This needs to be revisited with Root 6.
- We currently save 2 root trees in each output file, one which contains the eventwise information, one which contains the runwise information
- Input Managers put objects as PHIODataNodes on the node tree, output managers save selected PHIODataNodes to a file.
- Fun4All can manage multiple independent node trees

The Node Tree

```
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
  DST (PHCompositeNode)/
    PHG4INEVENT (PHDataNode)
    PIPE (PHCompositeNode)/
      G4HIT_PIPE (IO,PHG4HitContainer)
    MVTX (PHCompositeNode)/
      G4HIT_MVTX (IO,PHG4HitContainer)
    INTT (PHCompositeNode)/
      G4HIT_INTT (IO,PHG4HitContainer)
    TPC (PHCompositeNode)/
      G4HIT_ABSORBER_TPC (IO,PHG4HitContainer)
      G4HIT_TPC (IO,PHG4HitContainer)
    CEMC_ELECTRONICS (PHCompositeNode)/
      G4HIT_CEMC_ELECTRONICS (IO,PHG4HitContainer)
    CEMC_SPT (PHCompositeNode)/
      G4HIT_CEMC_SPT (IO,PHG4HitContainer)
    G4HIT_CEMC (IO,PHG4HitContainer)
    G4HIT_ABSORBER_CEMC (IO,PHG4HitContainer)
    HCALIN (PHCompositeNode)/
      G4HIT_ABSORBER_HCALIN (IO,PHG4HitContainer)
    ...
```

TOP: Top of Default Node Tree
Creation and populating of other node trees is possible (used for embedding)

You will see this printout of the node tree whenever the processing starts

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

The Node Tree

```
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
  DST (PHCompositeNode)/
    PHG4INEVENT (PHDataNode)
    PIPE (PHCompositeNode)/
      G4HIT_PIPE (IO,PHG4HitContainer)
    MVTX (PHCompositeNode)/
      G4HIT_MVTX (IO,PHG4HitContainer)
    INTT (PHCompositeNode)/
      G4HIT_INTT (IO,PHG4HitContainer)
    TPC (PHCompositeNode)/
      G4HIT_ABSORBER_TPC (IO,PHG4HitContainer)
  RUN (PHCompositeNode)/
    FIELD_CONFIG (IO,PHFieldConfigv1)
    PIPE (PHCompositeNode)/
      G4GEOPARAM_PIPE (IO,PdbParameterMapContainer)
      CYLINDERGEOM_PIPE (IO,PHG4CylinderGeomContainer)
    MVTX (PHCompositeNode)/
      G4GEOPARAM_MVTX (IO,PdbParameterMapContainer)
    INTT (PHCompositeNode)/
      G4GEOPARAM_INTT (IO,PdbParameterMapContainer)
...
```

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

You will see this printout of the node tree whenever the processing starts

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

The Node Tree

```
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
  DST (PHCompositeNode)/
    PHG4INEVENT (PHDataNode)
    PIPE (PHCompositeNode)/
      G4HIT_PIPE (IO,PHG4HitContainer)
    MVTX (PHCompositeNode)/
      G4HIT_MVTX (IO,PHG4HitContainer)
RUN (PHCompositeNode)/
    FIELD_CONFIG (IO,PHFieldConfigv1)
    PIPE (PHCompositeNode)/
      G4GEOPARAM_PIPE (IO,PdbParameterMapContainer)
      CYLINDERGEOM_PIPE (IO,PHG4CylinderGeomContainer)
    MVTX (PHCompositeNode)/
      G4GEOPARAM_MVTX (IO,PdbParameterMapContainer)
    INTT (PHCompositeNode)/
      G4GEOPARAM_INTT (IO,PdbParameterMapContainer)
  PAR (PHCompositeNode)/
    FIELD_MAP (PHDataNode)
    PIPE (PHCompositeNode)/
      G4GEO_PIPE (PHDataNode)

...
```

Users can add their own PHCompositeNodes Under the TOP Node. But then resetting the objects is their responsibility.

The PAR node hold more complicated geometry Objects which we do not want to save on DST

You will see this printout of the node tree whenever the processing starts

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

The Node Tree

```
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
  DST (PHCompositeNode)/
    PHG4INEVENT (PHDataNode)
    PIPE (PHCompositeNode)/
      G4HIT_PIPE (IO,PHG4HitContainer)
    HCALIN (PHCompositeNode)/
      G4HIT_ABSORBER_HCALIN (IO,PHG4HitContainer)
      G4HIT_HCALIN (IO,PHG4HitContainer)
      G4CELL_HCALIN (IO,PHG4CellContainer)
      TOWER_SIM_HCALIN (IO,RawTowerContainer)
      TOWER_RAW_HCALIN (IO,RawTowerContainer)
      TOWER_CALIB_HCALIN (IO,RawTowerContainer)
      CLUSTER_HCALIN (IO,RawClusterContainer)
    BBC (PHCompositeNode)/
      BbcVertexMap (IO,BbcVertexMapv1)
    TRKR (PHCompositeNode)/
      TRKR_HITSET (IO,TrkrHitSetContainer)
      TRKR_HITTRUTHASSOC (IO,TrkrHitTruthAssoc)
      TRKR_CLUSTER (IO,TrkrClusterContainer)
      TRKR_CLUSTERHITASSOC (IO,TrkrClusterHitAssoc)
...
```

Type of Node is given (IO is PHIODataNode)

Class of Data IO Object is given (you will need to know this when accessing the data)

You will see this printout of the node tree whenever the processing starts

Print it from the command line with

```
Fun4AllServer *se = Fun4AllServer::instance();
se->Print("NODETREE");
```

The Node Tree

```
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
  DST (PHCompositeNode)/
    PHG4INEVENT (PHDataNode)
    PIPE (PHCompositeNode)/
      G4HIT_PIPE (IO,PHG4HitContainer)
    MVTX (PHCompositeNode)/
      G4HIT_MVTX (IO,PHG4HitContainer)
    INTT (PHCompositeNode)/
      G4HIT_INTT (IO,PHG4HitContainer)
    TPC (PHCompositeNode)/
      G4HIT_ABSORBER_TPC (IO,PHG4HitContainer)
      G4HIT_TPC (IO,PHG4HitContainer)
    CEMC_ELECTRONICS (PHCompositeNode)/
      G4HIT_CEMC_ELECTRONICS (IO,PHG4HitContainer)
    CEMC_SPT (PHCompositeNode)/
      G4HIT_CEMC_SPT (IO,PHG4HitContainer)
    G4HIT_CEMC (IO,PHG4HitContainer)
    G4HIT_ABSORBER_CEMC (IO,PHG4HitContainer)
    HCALIN (PHCompositeNode)/
      G4HIT_ABSORBER_HCALIN (IO,PHG4HitContainer)
      G4HIT_HCALIN (IO,PHG4HitContainer)
...
```

Caveat: You loose ownership once an object is put on the node tree. Fun4All deletes the node tree when cleaning up. Deleting nodes is not supported (if you give me a good reason I'll work on that)

You will see this printout of the node tree whenever the processing starts

Print it from the command line with

```
Fun4AllServer *se = Fun4AllServer::instance();
se->Print("NODETREE");
```

Your Analysis Module

You need to inherit from the SubsysReco Baseclass (offline/framework/fun4all/SubsysReco.h) which gives the methods which are called by Fun4All. If you don't implement all of them it's perfectly fine (the beauty of base classes)

- Init(PHCompositeNode *topNode): called once when you register the module with the Fun4AllServer
- InitRun(PHCompositeNode *topNode): called whenever data from a new run is encountered
- Process_event (PHCompositeNode *topNode): called for every event
- ResetEvent(PHCompositeNode *topNode): called after each event is processed so you can clean up leftovers of this event in your code
- EndRun(const int runnumber): called before the InitRun is called (caveat the Node tree already contains the data from the first event of the new run)
- End(PHCompositeNode *topNode): Last call before we quit

If you create another node tree you can tell Fun4All to call your module with the respective topNode when you register your modue

Okay, How do I navigate the Node Tree which is in every argument????

You need to know the name of the node and the class of the object you want (e.g. some PHG4HitContainer version in the node called G4HIT_HCALIN (that's where the Fun4All printout of the Node Tree comes in handy)

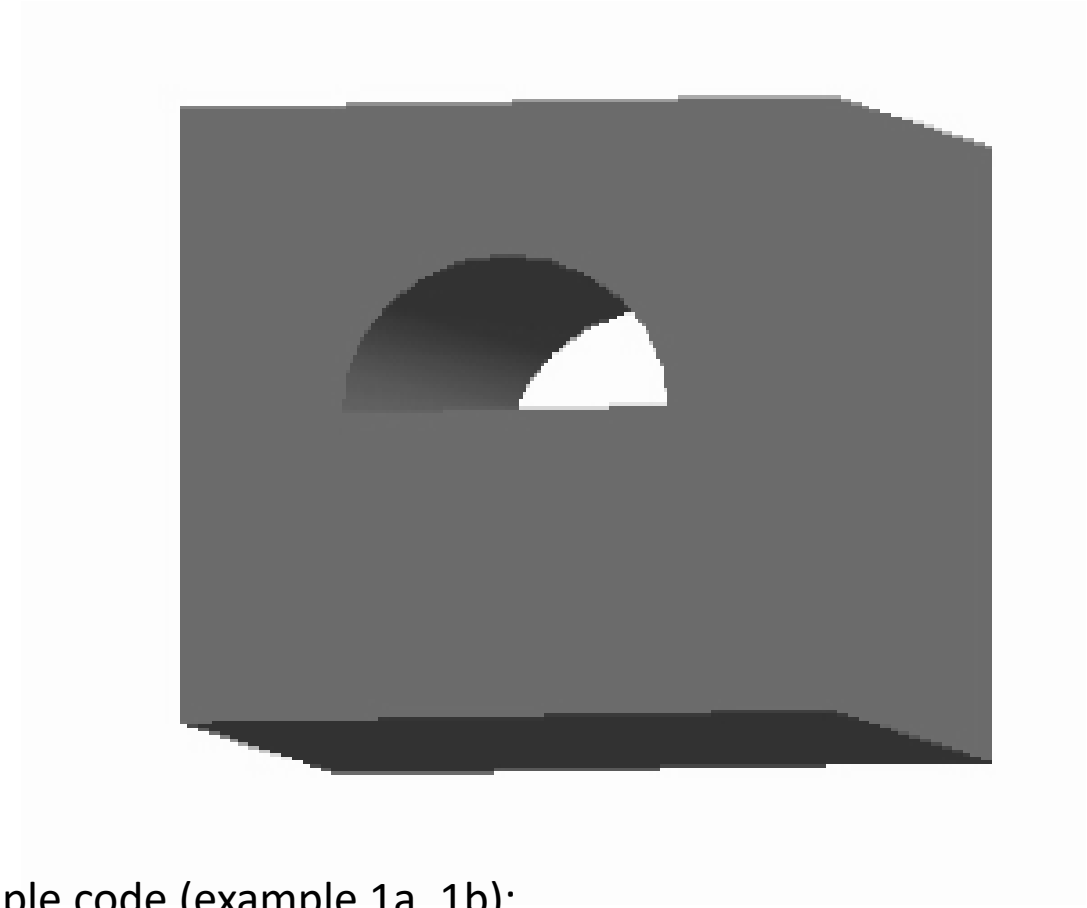
topNode of the node tree your module is registered with

```
#include <g4hit/PHG4HitContainer.h>
#include <fun4all/getClass.h>

Myanalysis::process_event(PHCompositeNode *topNode)
{
   PHG4HitContainer *g4hits =
findNode::getClass<PHG4HitContainer>(topNode,"G4HIT_HCALIN");
 if (g4hits)
   …
}
```

Caveat: getClass will return the first node of a given name, if you have have multiple identically named nodes you need to search differently

Example 1: A simple Detector with Hardcoded Geometry



Yes – it is a box with a half pipe hole

Github location:
https://github.com/EIC-Detector/g4exampledetector
g4exampledetector/simple/source
g4exampledetector/simple/macros

Ntuple code (example 1a, 1b):
https://github.com/sPHENIX-Collaboration/coresoftware/blob/master/simulation/g4simulation/g4histos/G4HitNtuple.cc

Example 1: A simple Detector

Cut and paste commands from https://www.phenix.bnl.gov/WWW/publish/phnxbld/EIC/tutorial/example1.html

Shell commands:

```
source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/eic_setup.sh -n
git clone https://github.com/EIC-Detector/g4exampledetector
cd g4exampledetector/simple/source
mkdir build
cd build
../autogen.sh -prefix=$HOME/myinstall
make install
source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/setup_local.sh
$HOME/myinstall
cd ../../macros
root.exe
```

At the Root prompt:

```
.x Fun4All_Example01.C(-1)
.L Display.C
PHG4Reco *g4 = QTGui();
```

And on the G4 cmd line:

```
/Fun4All/run 1
```

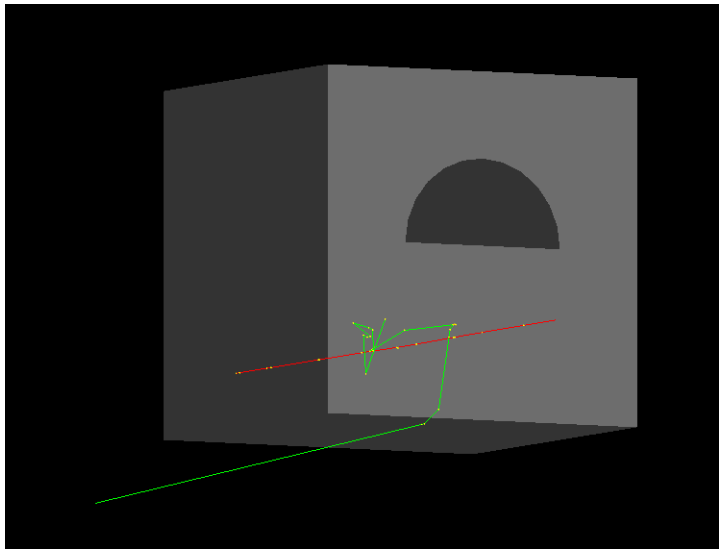
Alt F7 to move large G4 window in windows virtual box

Example 1: A simple Detector

Cut and paste commands from https://www.phenix.bnl.gov/WWW/publish/phnxbld/EIC/tutorial/example1.html

Shell commands:

```
source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/eic_setup.sh -n
git clone https://github.com/EIC-Detector/g4exampledetector
cd g4exampledetector/simple/source
mkdir build
cd build
../autogen.sh -prefix=$HOME/myinstall
make install
source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/setup_local.sh
$HOME/myinstall
cd ../../macros
root.exe
```



π^- in center of simple detector

At the Root prompt:

```
.x Fun4All_Example01.C(-1)
.L Display.C
PHG4Reco *g4 = QTGui();
```

And on the G4 cmd line:

```
/Fun4All/run 1
```

Alt F7 to move large G4 window in windows virtual box

Example 1a: Geometry Verification with a Geantino Scan

Cut and paste commands from https://www.phenix.bnl.gov/WWW/publish/phnxbld/EIC/tutorial/example1a.html

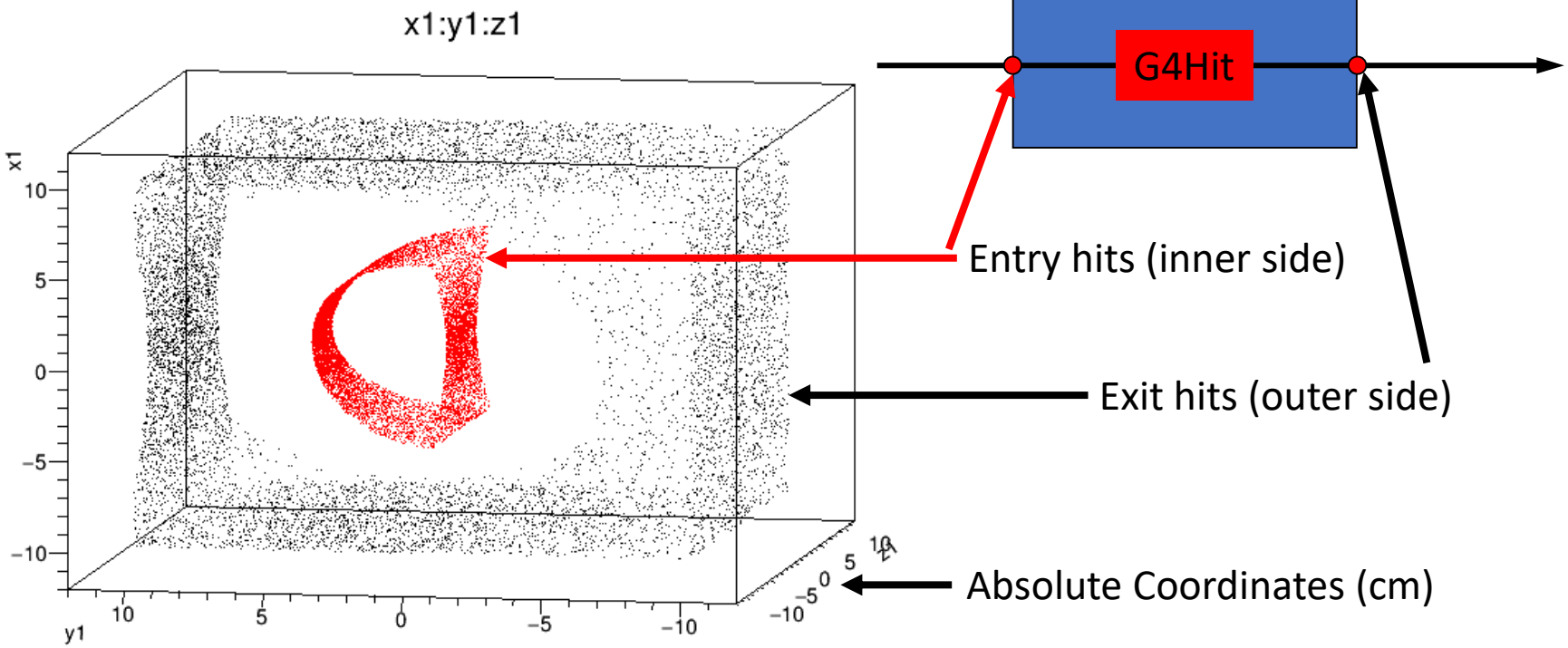
After all those rotations and translations, how do you verify that the detector position is actually where you think it should be???

Shell commands:
root.exe

At the Root prompt, run 10000 events
.x Fun4All_G4_Geantino.C(10000)

Shell commands:
root.exe HitNtuple.root

At the Root prompt:
hitntup->Draw("x1:y1:z1")
hitntup->SetMarkerColor(2);
hitntup->Draw("x0:y0:z0","","same");



Geantino: Non interacting Geant particle, Geant4 introduced charged Geantinos. Discovery announcements typically on April 1st

Example 1c: Saving and Reading Chain Snapshot

Cut and paste commands from https://www.phenix.bnl.gov/WWW/publish/phnxbld/EIC/tutorial/example1b.html

Shell commands:
root.exe

At the Root prompt, run 10000 events:
.x Fun4All_G4_Write_Dst.C(10000)

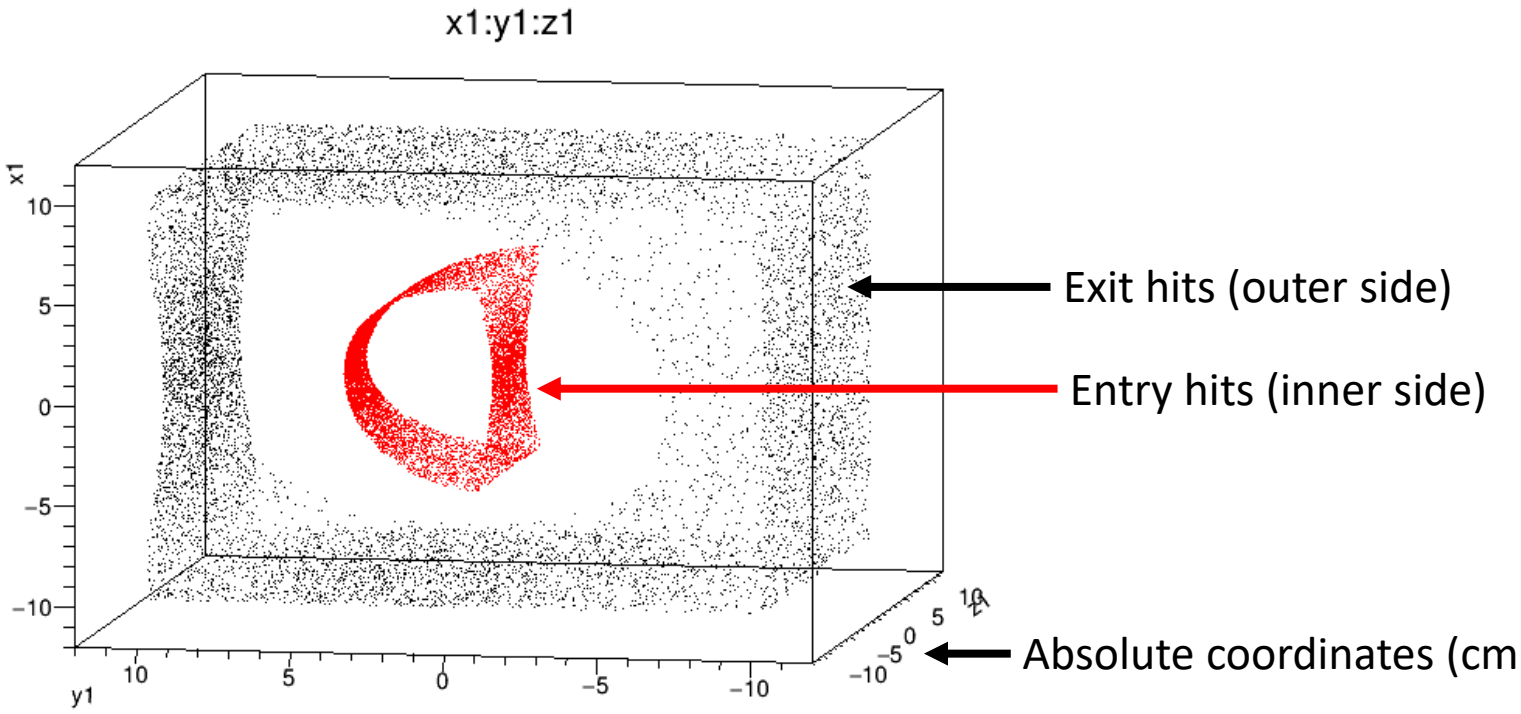
Shell commands:
root.exe

At the Root prompt, run all events:
.x Fun4All_Read_Dst.C(0)

Shell commands:
root.exe HitsFromDst.root

At the Root prompt:
hitntup->Draw("x1:y1:z1")
hitntup->SetMarkerColor(2);
hitntup->Draw("x0:y0:z0","","same");

Note:
no analysis code dependence in Fun4All_G4_Dst.C
No Geant4 dependence in Fun4All_Read_Dst.C
→ Dst's can be created and read independently without having all code



Ntuple code: https://github.com/sPHENIX-Collaboration/coresoftware/blob/master/simulation/g4simulation/g4histos/G4HitNtuple.cc

Example 2:
Fast Momentum Resolution Estimate during Design Stage

What are the first questions any tracking design has to answer?

- What is the momentum resolution?
- What is the distance of closest approach?
- Can you do better?
- Do you have any idea how much this costs??? How about fewer layers?

Welcome to our full GEANT4 simulation with Kalman Filter reconstruction and vertexing.

- Put a detector together from simple shapes in a ROOT macro (cylinders, boxes)
- Pick your particle and a couple of energies
- tell the tracking the error on the hits and run
- Wash, rinse, repeat until you(r boss) like(s) the result

Example 2:
Fast Momentum Resolution Estimate during Design Stage

Cut and paste commands from https://www.phenix.bnl.gov/WWW/publish/phnxbld/EIC/tutorial/example2.html

Shell commands:

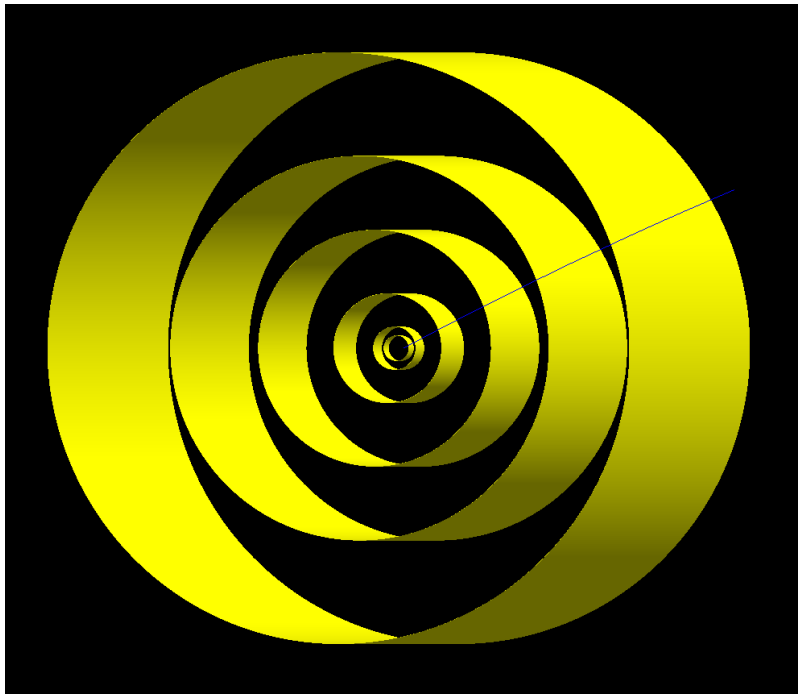
source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/eic_setup.sh -n
git clone https://github.com/sPHENIX-Collaboration/tutorials
cd tutorials/Momentum
root.exe

At the Root prompt:

.x Fun4All_G4_Momentum.C(-1)
.L DisplayOn.C
PHG4Reco *g4 = QTGui();

And on the G4 cmd line:

/Fun4All/run 1



Let's have a look at a 6 layer silicon detector

Radii/Thickness (in cm):

- 2.71, 0.02
- 4.63, 0.02
- 11.765, 0.0625
- 25.46, 0.032
- 41.38, 0.032
- 63.66, 0.032

Using 4GeV/c positrons

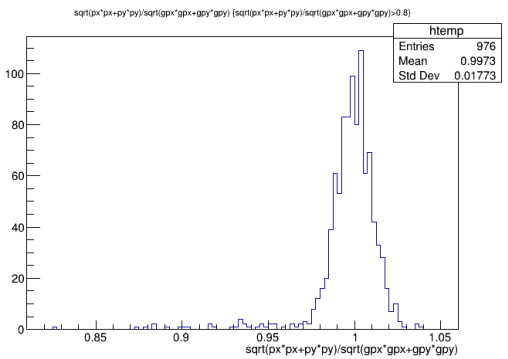
Root macro in github:
https://github.com/sPHENIX-Collaboration/tutorials/Momentum/Fun4All_G4_Momentum.C

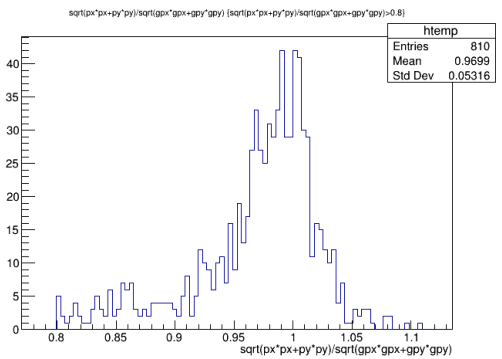
Example 2:
Fast Momentum Resolution Estimate during Design Stage

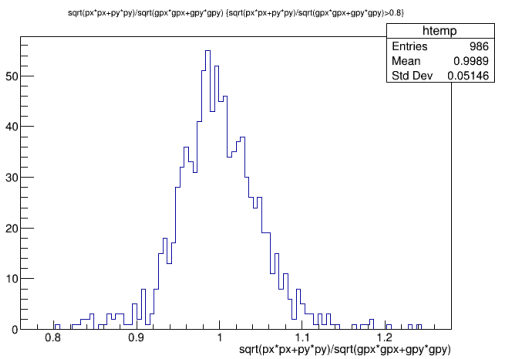
Positrons @ 4GeV/c at midrapidity

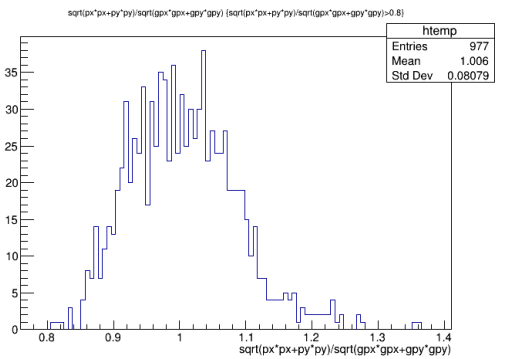
Nominal | X10 Si thickness | ½ layer radius | 1/10 resolution

Reco(p_t)/Input(p_t)

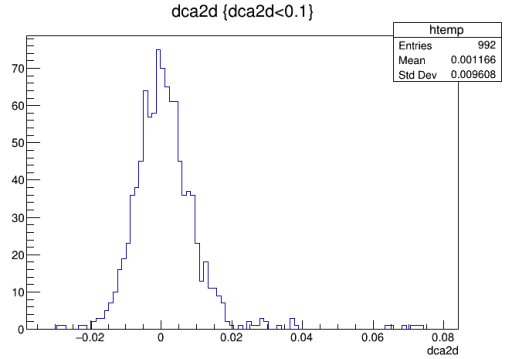


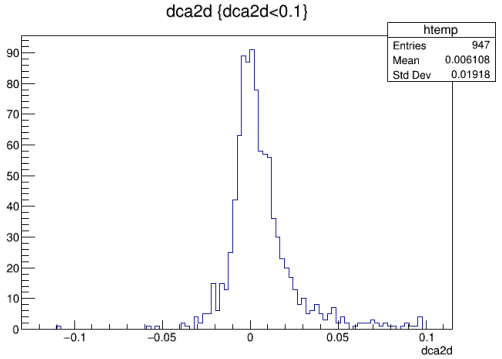


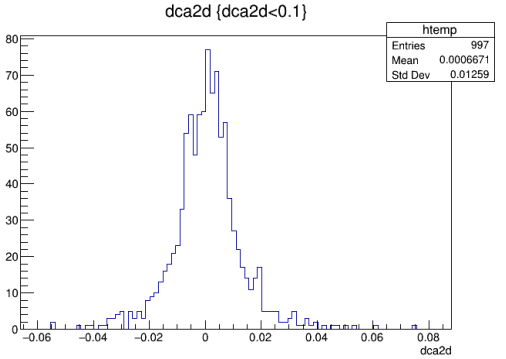


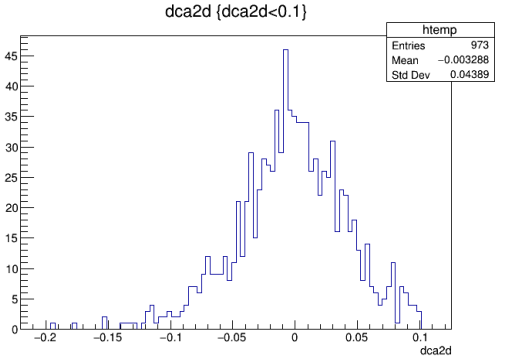


dca2d

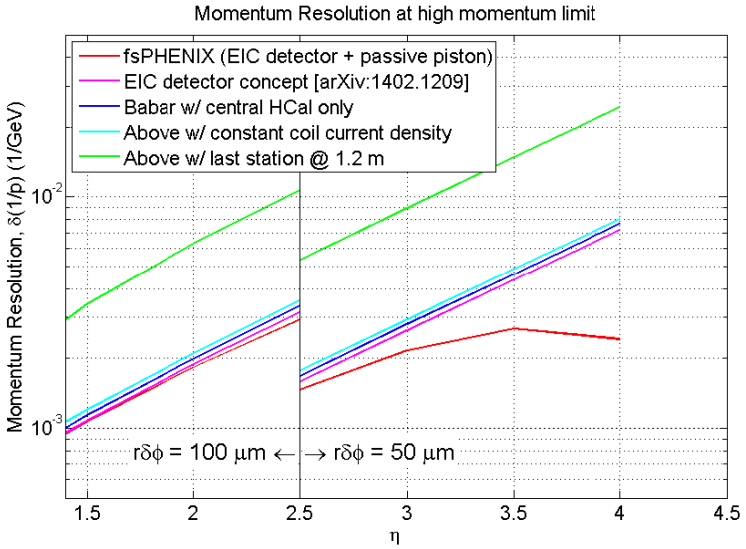


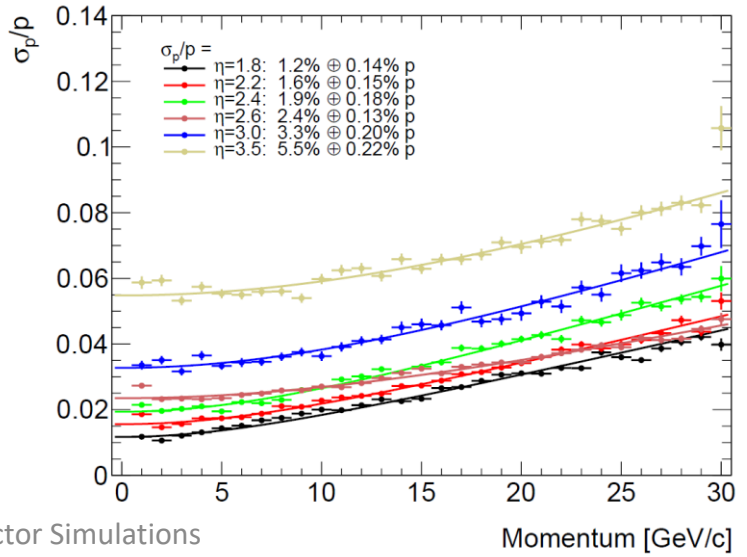




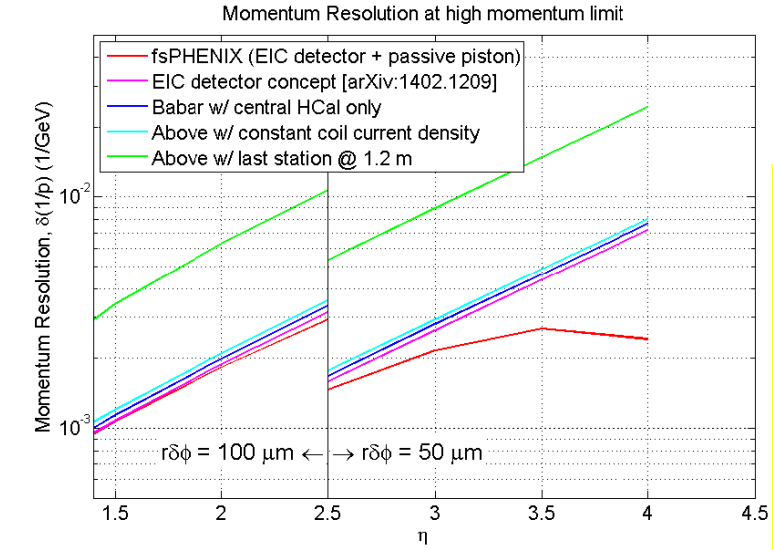


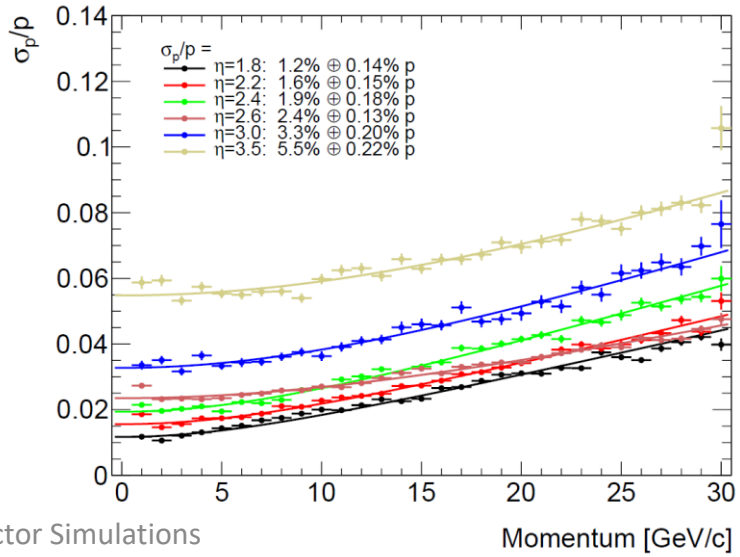
Example 2:
Fast Momentum Resolution Estimate during Design Stage





Example 2:
Fast Momentum Resolution Estimate during Design Stage





This prospect is meant to motivate you to learn about programming loops and running batch jobs

Example 3: Adding detectors to existing setups

Cut and paste commands from https://www.phenix.bnl.gov/WWW/publish/phnxbld/EIC/tutorial/example3.html

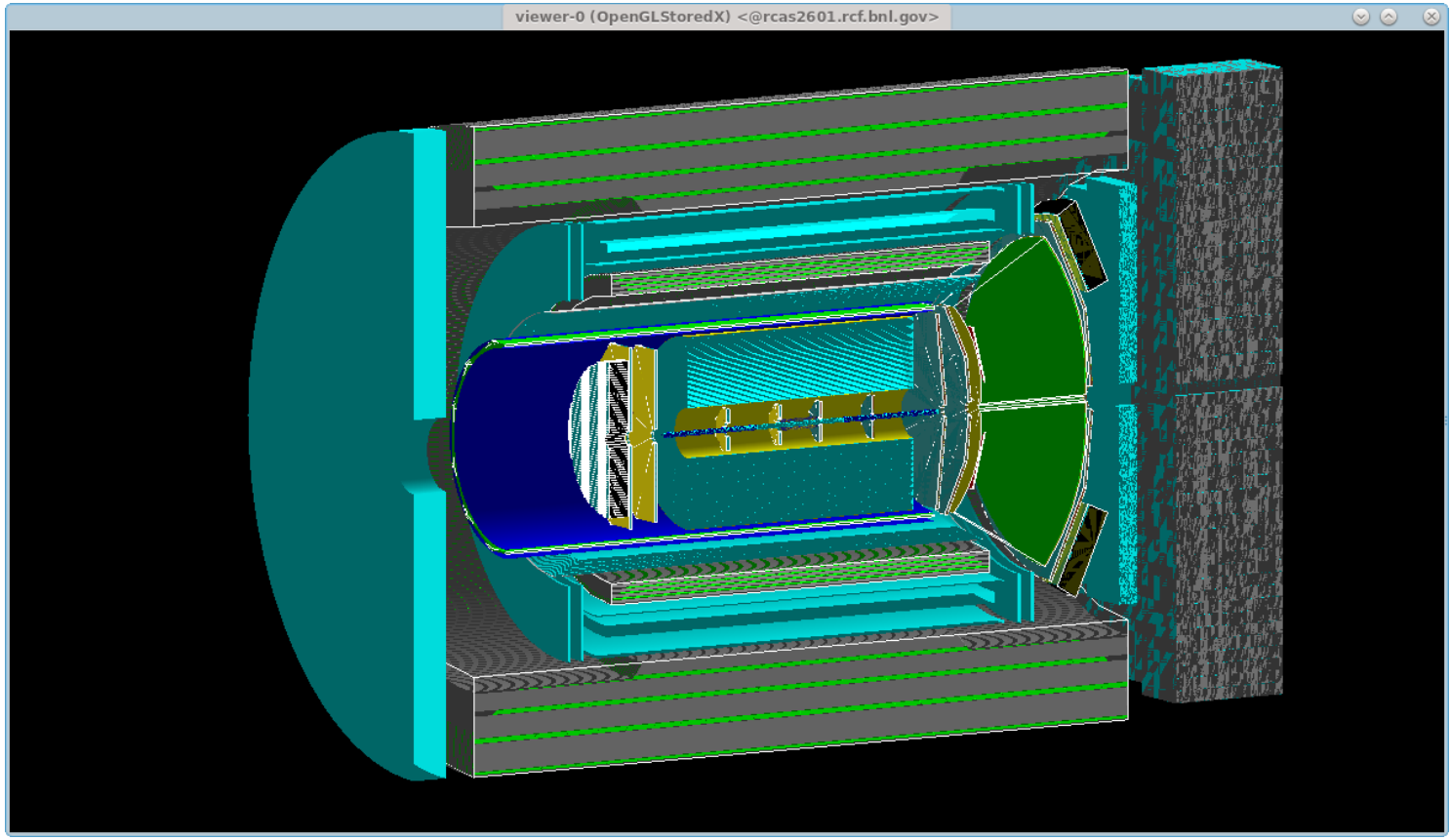
Shell commands:

source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/eic_setup.sh -n
source /cvmfs/eic.opensciencegrid.org/x8664_sl7/opt/sphenix/core/bin/setup_local.sh $HOME/myinstall
git clone https://github.com/sPHENIX-Collaboration/macros
cd macros/macros/g4simulations
root.exe

At the Root prompt:

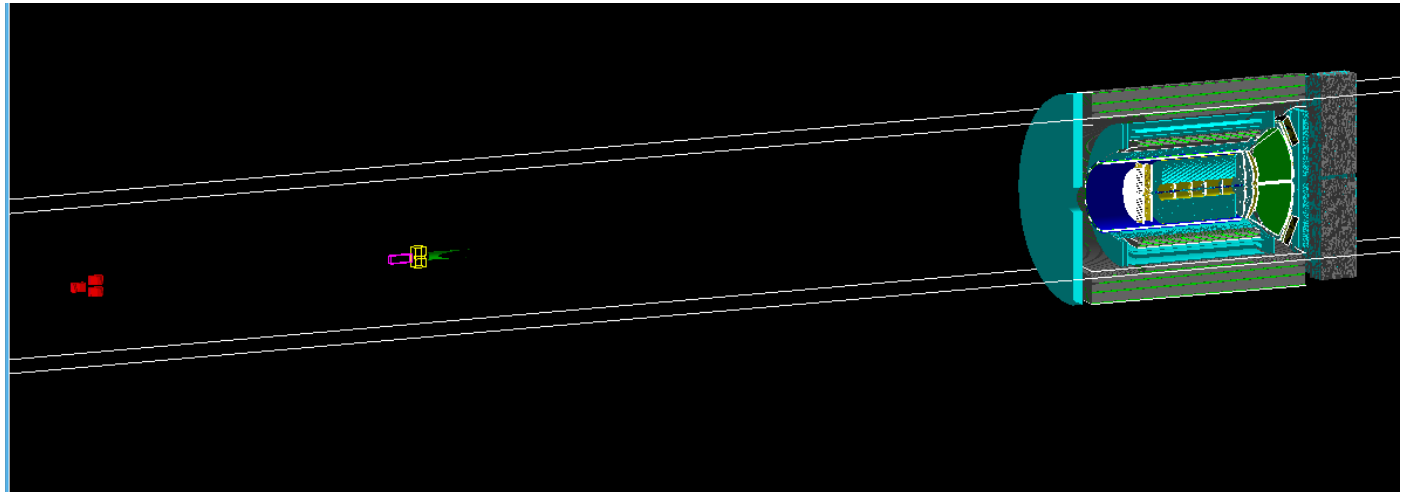
.x Fun4All_G4_EICDetector.C(-1)
.L DisplayOn.C
PHG4Reco *g4 = DisplayOn();
g4->ApplyCommand("/vis/viewer/panTo 0 100 cm")

QT is way to slow for more complex detectors but you are welcome to try
PHG4Reco *g4 = QTGui();



Example 3: Adding detectors to existing setups

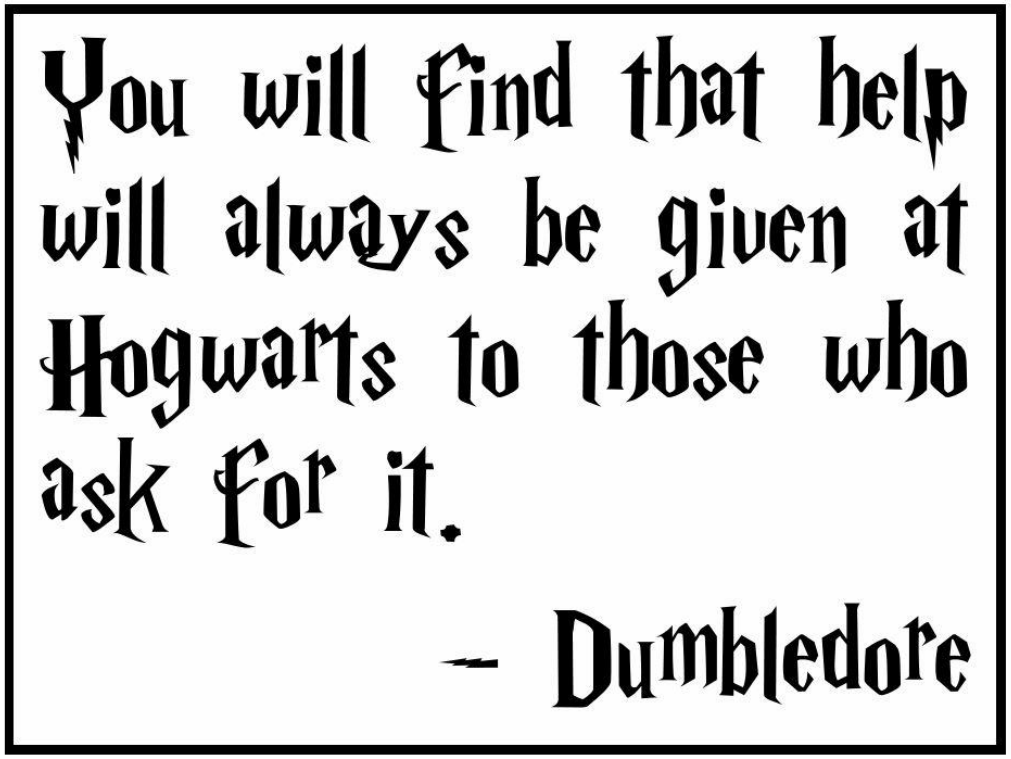
Now open G4Setup_EICDetector.C



Now open G4Setup_EICDetector.C and add 4 lines:
After line 27:
#include <g4lmondetector/G4LmonSubsystem.h>
R__LOAD_LIBRARY(libg4lmondetector.so)
After line 310 (before the truth subsystem):
G4LmonSubsystem *lmon = new G4LmonSubsystem("LumiMon");
g4Reco->registerSubsystem(lmon);

The fine print:
The stepping action does not put any objects on our node tree
This does not include the detector into any analysis chain
There are overlaps (the vacuum box) which need to be dealt with
Our world is filled with G4_Air, need to add a vacuum system

HELP!!!!!!!



- Documentation (more to come):
  https://eic-detector.github.io
- Mattermost chat (BNL hosted open source slack). It is a public channel. Rcf accounts can subscribe, non BNL accounts need invite:
  https://chat.sdcc.bnl.gov/eic/channels/fun4all-software-support
- Last *resort:*
  Send mail to or Chris Pinkenburg (pinkenburg at bnl.gov) and/or Jin Huang (jinhuang at bnl.gov)

Final Remarks

- This was just a small selection of our capabilities
- Fun4All was already used to simulate and design a new collider detector
- Fun4All was used for published EIC detector concepts
- Synergy: Collaboration with an existing experiment ensures long term support and warm bodies to work on it
  - Reuse existing code (tracking, clustering, calibrations,…)
  - Read and reconstruct raw data from a commonly used daq (rcdaq)
- EIC specific detectors can go into their own repository (https://github.com/EIC-Detector)
- EIC cvmfs volume: /cvmfs/eic.opensciencegrid.org
- EIC specific 3rd party libraries can be installed there
- Users can use their own repository and just link against Fun4All in cvmfs
- Distribution in cvmfs and singularity container run on most platforms (including Jlab and OSG)

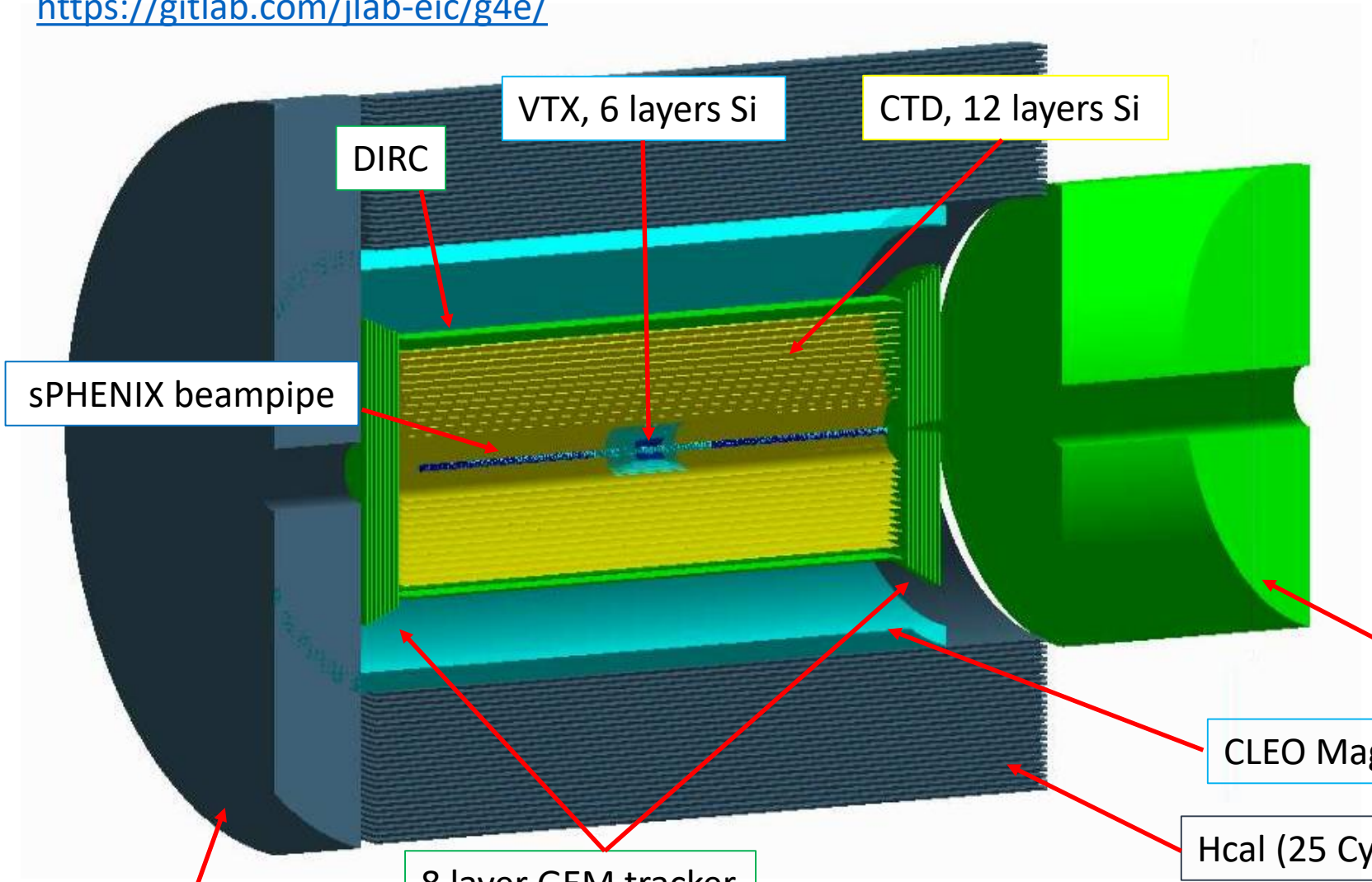
Congratulations – you made it!

YOU HAVE GONE TOO FAR

BACKUP ZONE

Partial JLEIC reverse engineered from g4e

https://gitlab.com/jlab-eic/g4e/



- Two days of work (mostly spend on re-familiarizing how we do this)
- Only one Box detector (vtx) needed actual cut and paste coding
  - Just too lazy to look into our generic boxes
- Forward/backward only container volumes
- Forward/Backward pieces missing