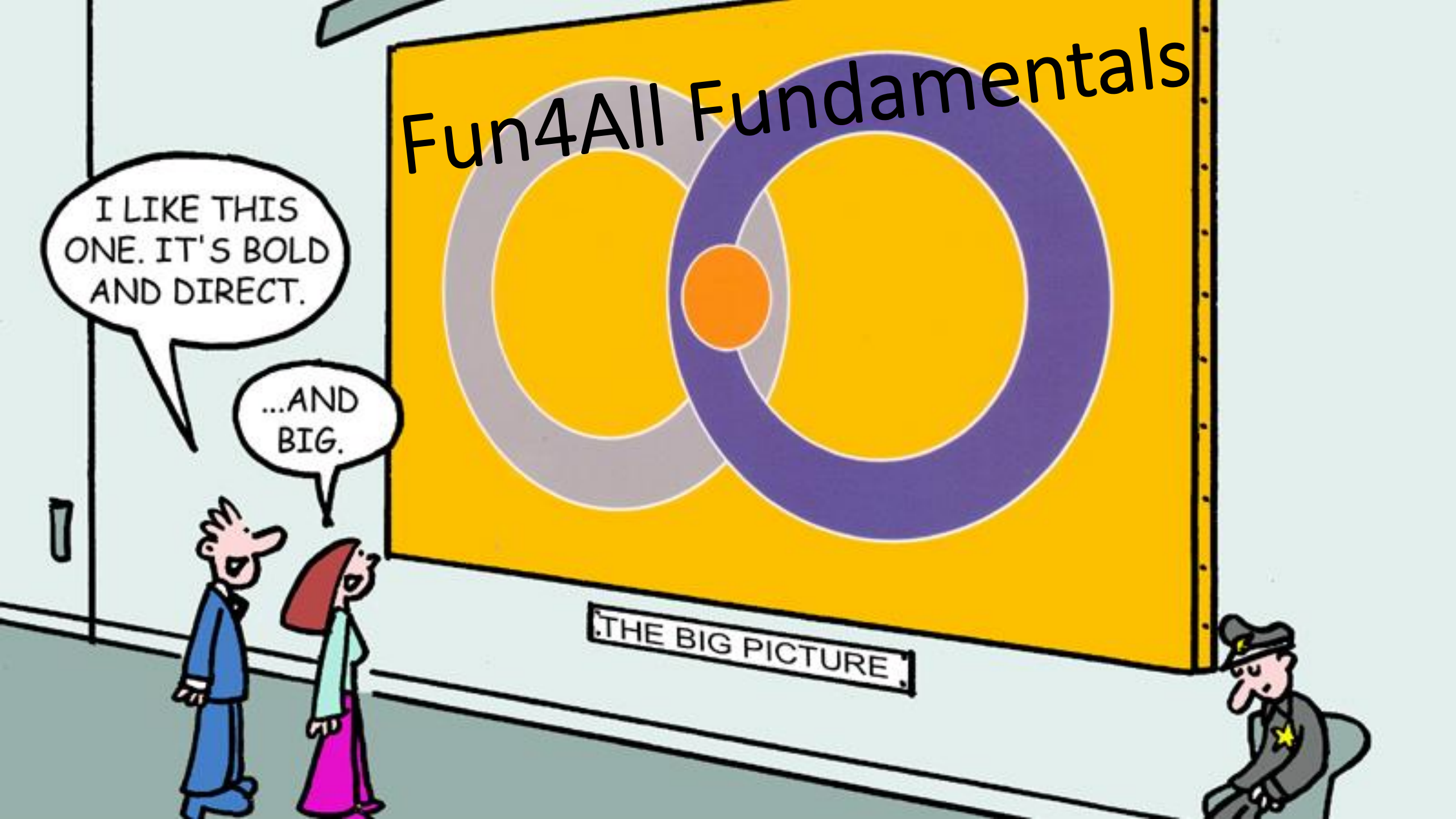
Fun4All Fundamentals

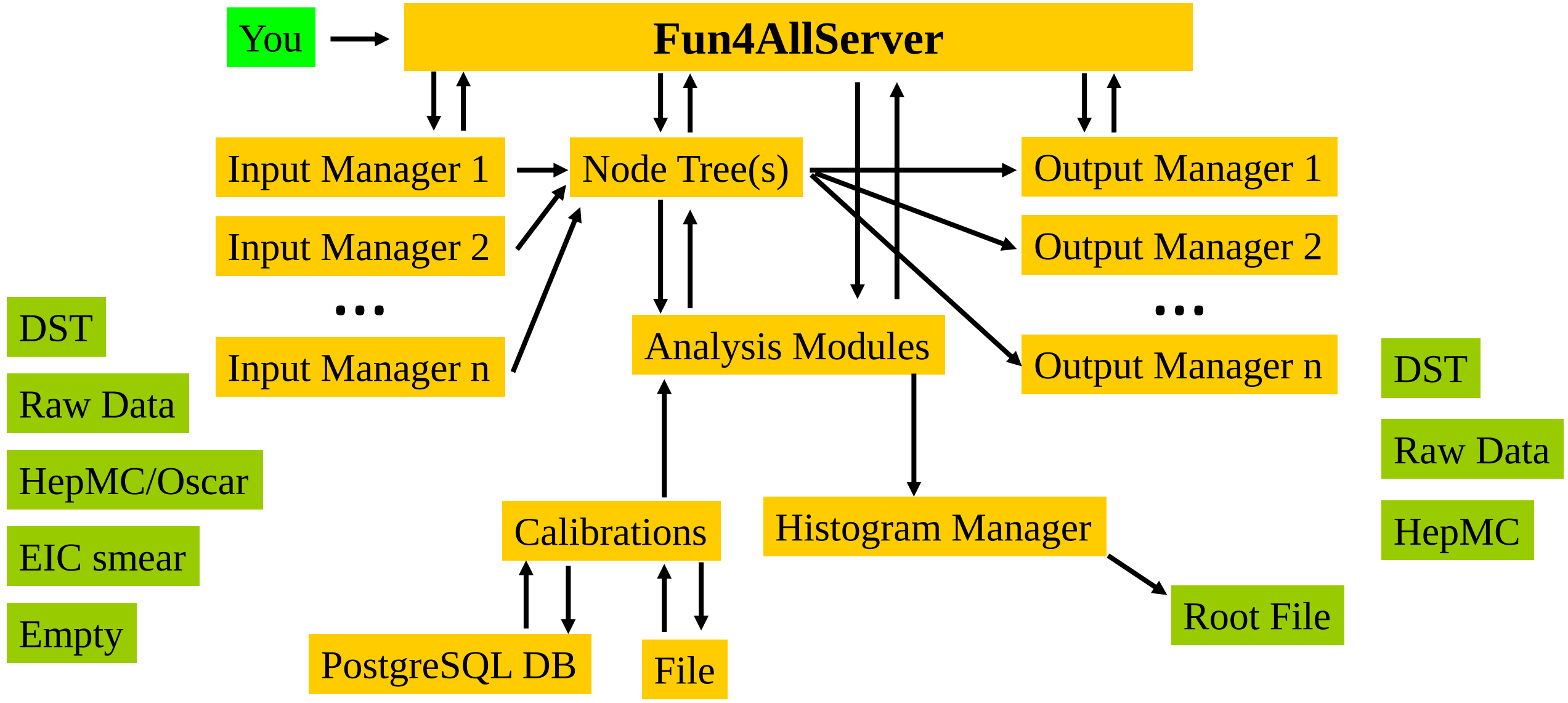


Brief History of Fun4All

- 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
- Designed for agile development in interactive and batch use
- 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
- March 2019 Fork for EIC simulations, addition of eic specific packages
  - Large fraction of full simulations for YR done with Fun4All (“find Fun4All” results in 14 hits)

And ECCE has already a fork as well:
https://github.com/ECCE-EIC/coresoftware/tree/master/offline/framework

Fun4All Framework in a nutshell



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

What Fun4All does 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)
- The analysis is a continuous chain from the event generator/raw data up to jet reconstruction
- Make snapshots at any state of the reconstruction/analysis
- Access to calibrations

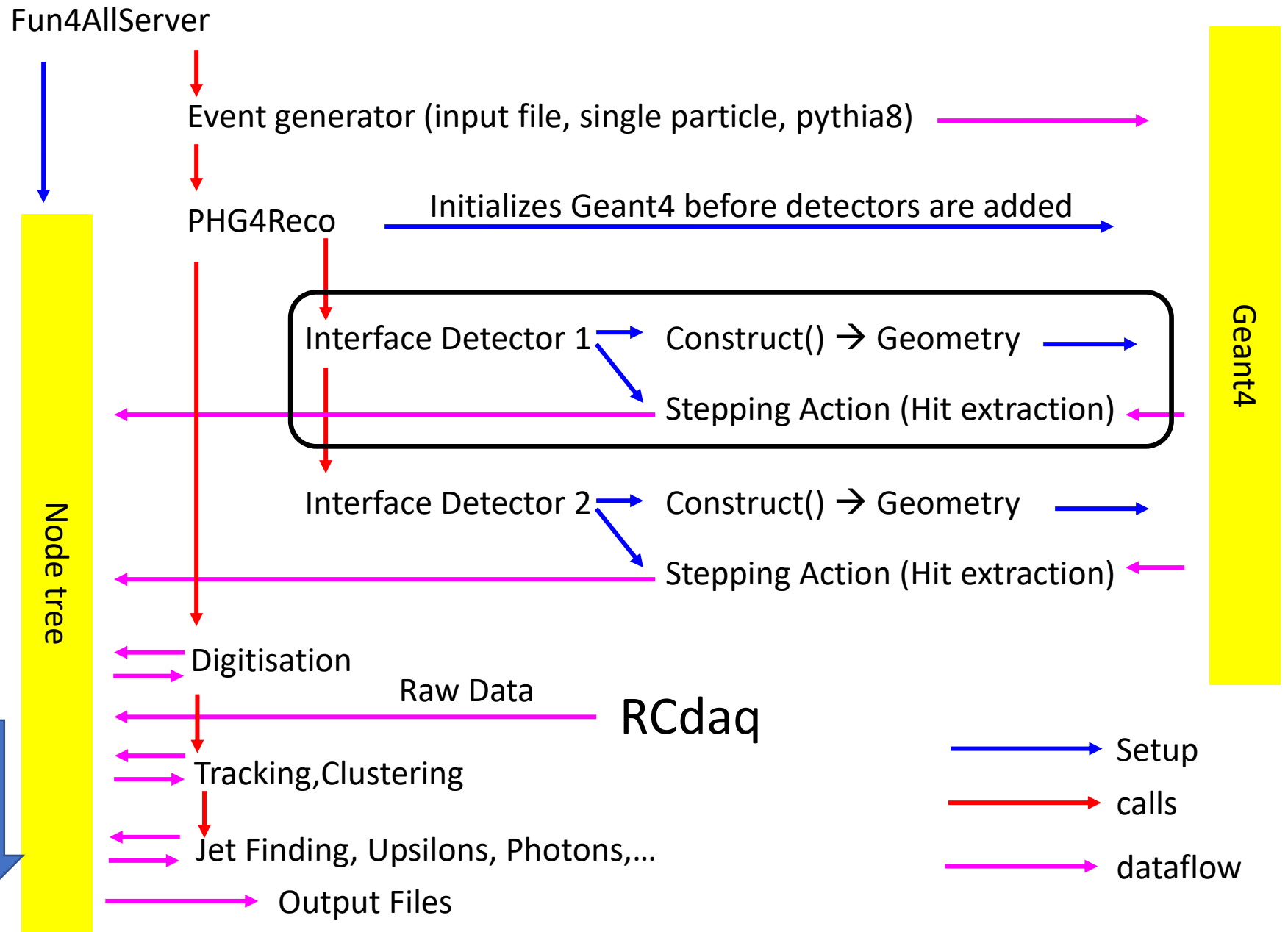
Fun4All has been our workhorse for 18 years, running raw data reconstruction, analysis, simulations and embedding

Simulations

- GEANT4 based (no GEANT3, no virtual Monte Carlo)
- Current version 10.06.p02, default physics list FTFP_BERT
- Fully integrated into our analysis chain as Reconstruction Module
- Modular design – all detectors are self contained
- Configured on the macro level
- Generic detectors (boxes, cylinders, cones) exist
- sPHENIX subdetectors (and a lot of ECCE) fully implemented
- 9 years of development

Tutorials:
https://github.com/ECCE-EIC/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

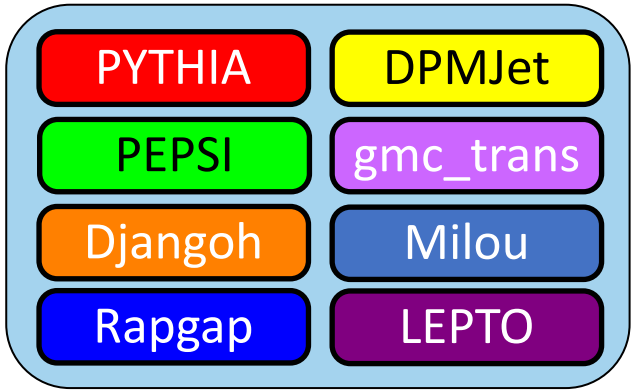
Modular: Each detector is its own entity providing the flexibility needed for complex setups

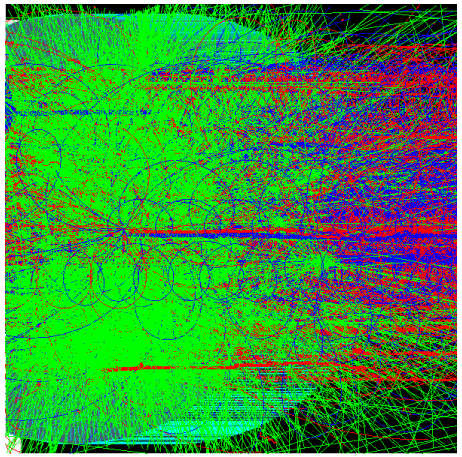
Generic detectors like boxes, cylinders, cones exist and can be configured on a macro level

Volumes can be put into world or mother volumes

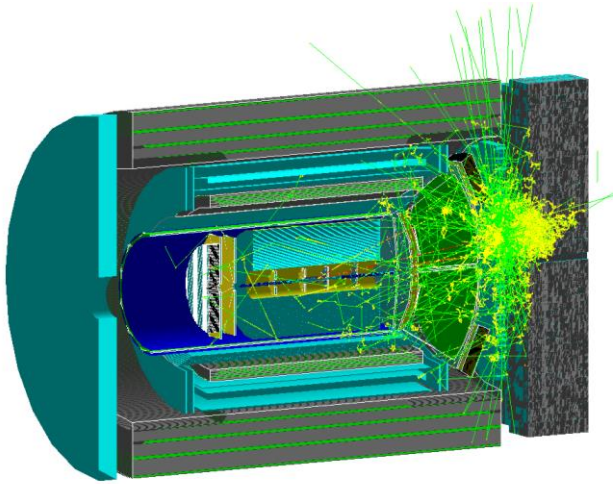
Currently implemented event generators

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



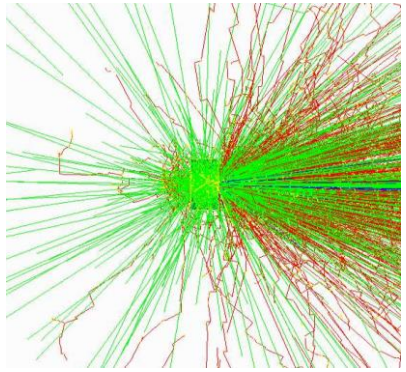


Very peripheral heavy ion collision from hijing

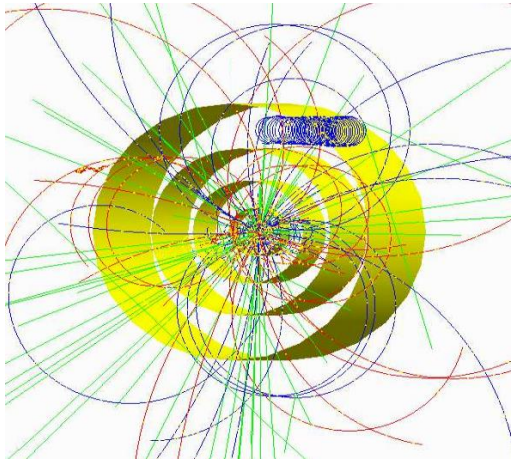


Sarte as seen by EIC detector

- 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)

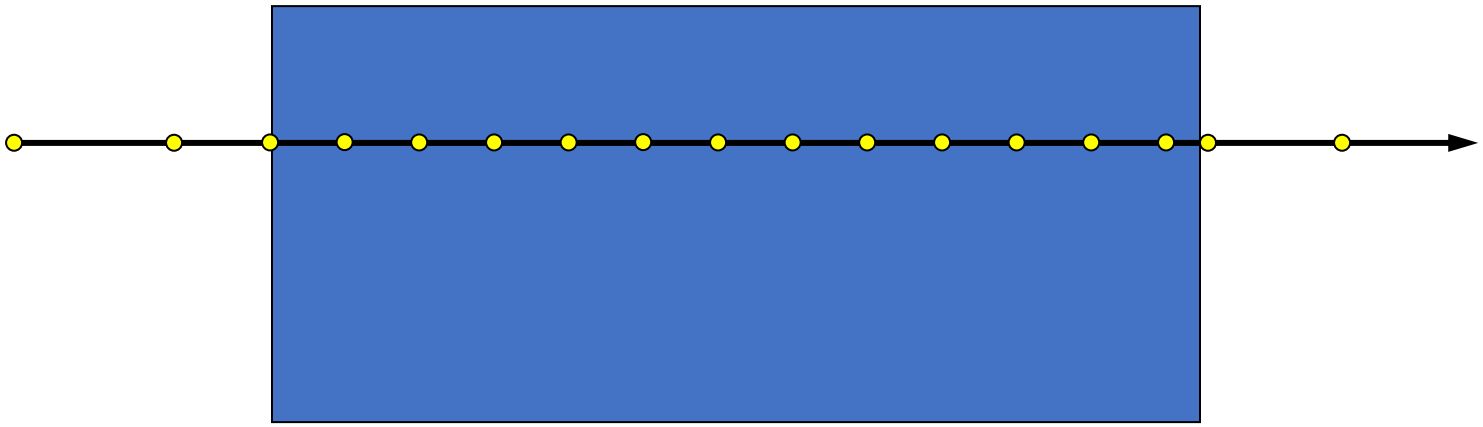


Pythia8 in a 6 layer silicon detector mockup and 2T field

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

Recap: GEANT steps

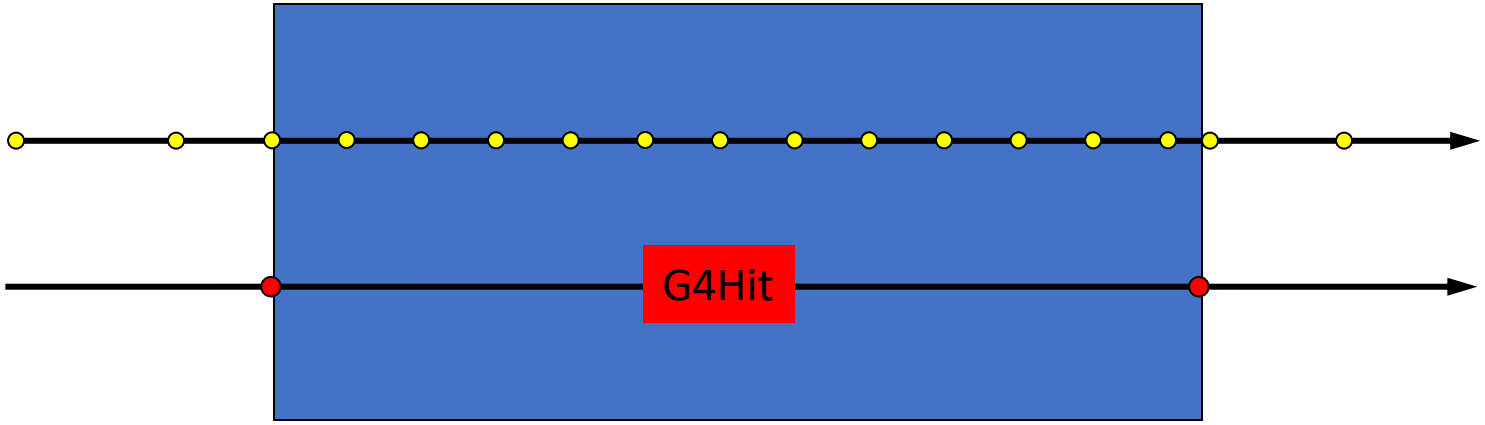
GEANT propagates particles one step at a time. The step size is determined by the physics processes associated with the current particle or when a boundary between volumes is crossed



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,…)

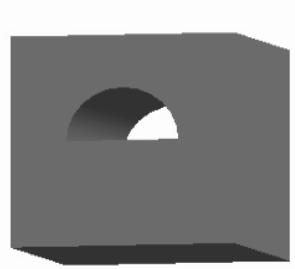


We also keep the ancestry for G4Hits so any hit can be traced back to a primary particle. To reduce size we do not store particles which do not leave G4Hits and are not in the ancestry of a particle which created a G4Hit

Creating your own Detector

3 classes mandatory
- Subsystem
- Detector
- SteppingAction

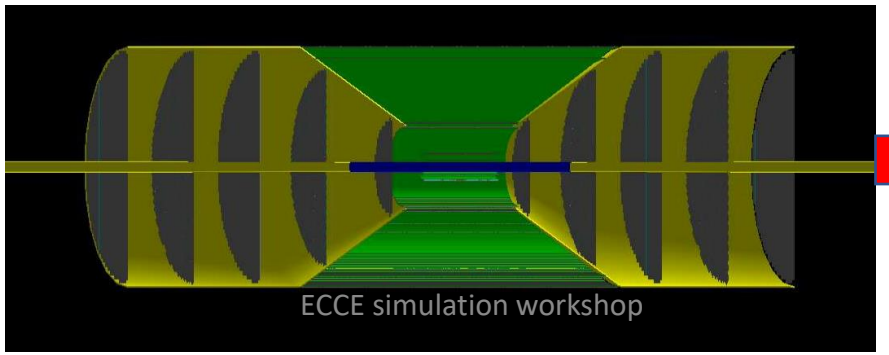
Optional
- Use of parameters
- DisplayAction

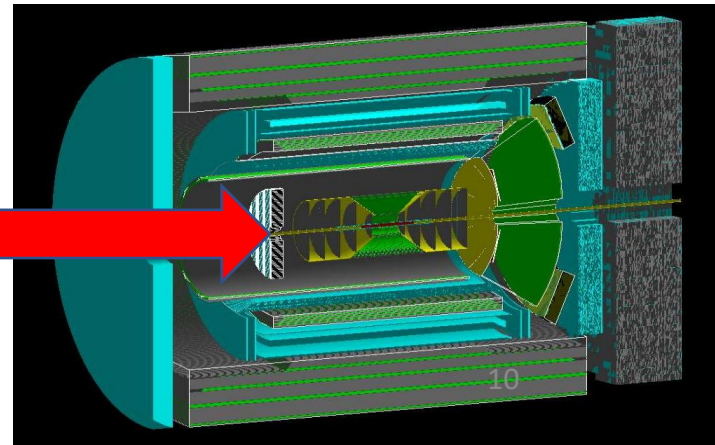


Examples implementing this beautiful box with a half pipe hole including macros:
https://github.com/sPHENIX-Collaboration/g4exampledetector

Template Example introduces script to add detector:
CreateG4Subsystem.pl <Detector Name>
options are:
--all : Also create autogen.sh, configure.ac and Makefile.am
--overwrite : overwrite existing files (handle with care, we only have snapshots of $HOME)
https://github.com/sPHENIX-Collaboration/g4exampledetector/tree/master/template

You can then add your detector to our existing setups on the macro level



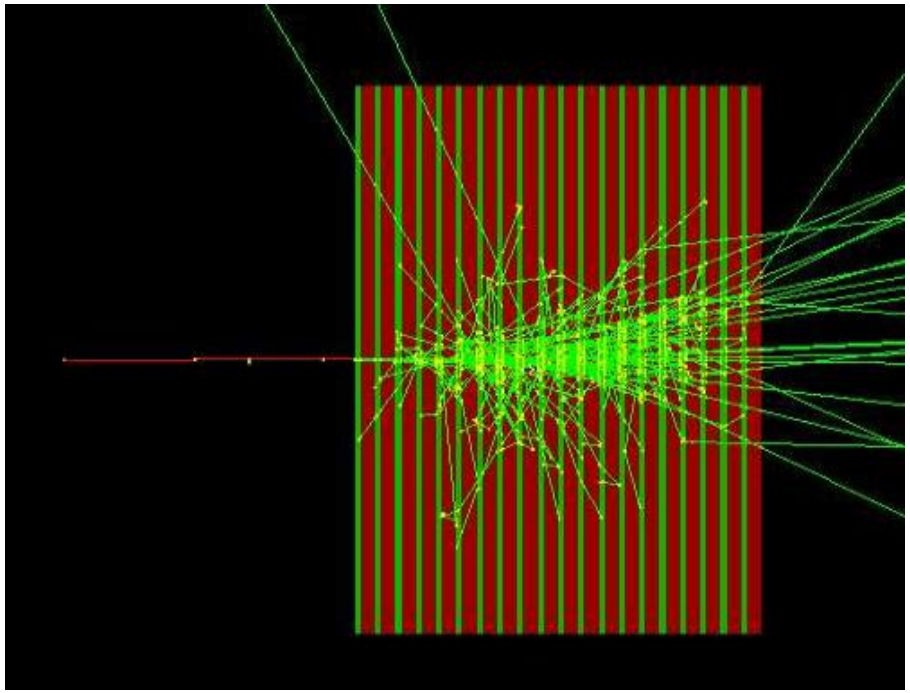


Let’s simulate a calorimeter

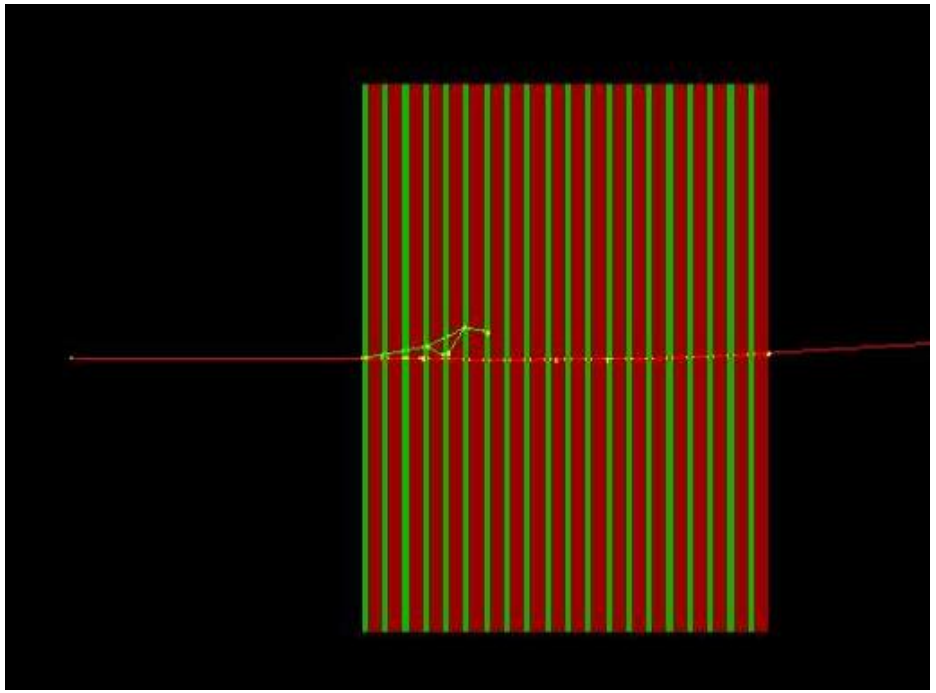
- Purely macro based, using generic blocks
- 20 layer Shashlik
  - 0.2cm Tungsten
  - 0.5 cm Scintillator
- Work effort 30 minutes
  - Mostly fixing typos

```
double xsize = 20.;
double ysize = 20.;
double zsizeW = 0.2;
double zsizeSc = 0.5;
for (int i=0; i<20; i++)
 {
   PHG4BlockSubsystem *boxW = new PHG4BlockSubsystem("boxW",i);
   boxW->set_double_param("size_x",xsize);
   boxW->set_double_param("size_y",ysize);
   boxW->set_double_param("size_z",zsizeW);
   boxW->set_color(0,1,0,0.8);
   boxW->set_double_param("place_z",zsizeW/2.+zstart);
   boxW->set_string_param("material","G4_W"); // material of box
   boxW->SetActive();
   boxW->OverlapCheck(1);
   boxW->SuperDetector("boxW");
   g4Reco->registerSubsystem(boxW);
   zstart += zsizeW;
   PHG4BlockSubsystem *boxSc = new PHG4BlockSubsystem("boxSc",i);
   boxSc->set_double_param("size_x",xsize);
   boxSc->set_double_param("size_y",ysize);
   boxSc->set_double_param("size_z",zsizeSc);
   boxSc->set_color(1,0,0,0.6);
   boxSc->set_double_param("place_z",zsizeSc/2.+zstart);
   boxSc->set_string_param("material","G4_POLYSTYRENE"); // material of box
   boxSc->SetActive();
   boxSc->OverlapCheck(1);
   boxSc->SuperDetector("boxSc");
   g4Reco->registerSubsystem(boxSc);
   zstart += zsizeSc;
}
```

Let’s simulate a calorimeter



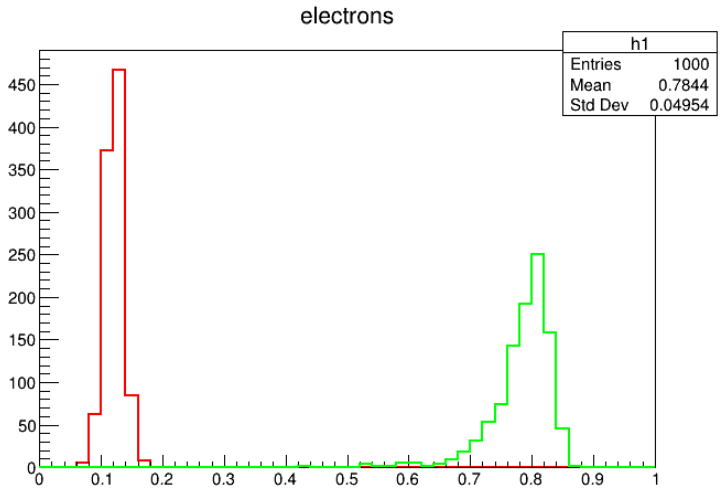
1GeV/c Electron



1GeV/c π^-

You want to design your EMCal that electrons/photons have a high probability to shower (many radiation lengths)
But hadrons should not (few interaction lengths) → don’t use too many absorbers

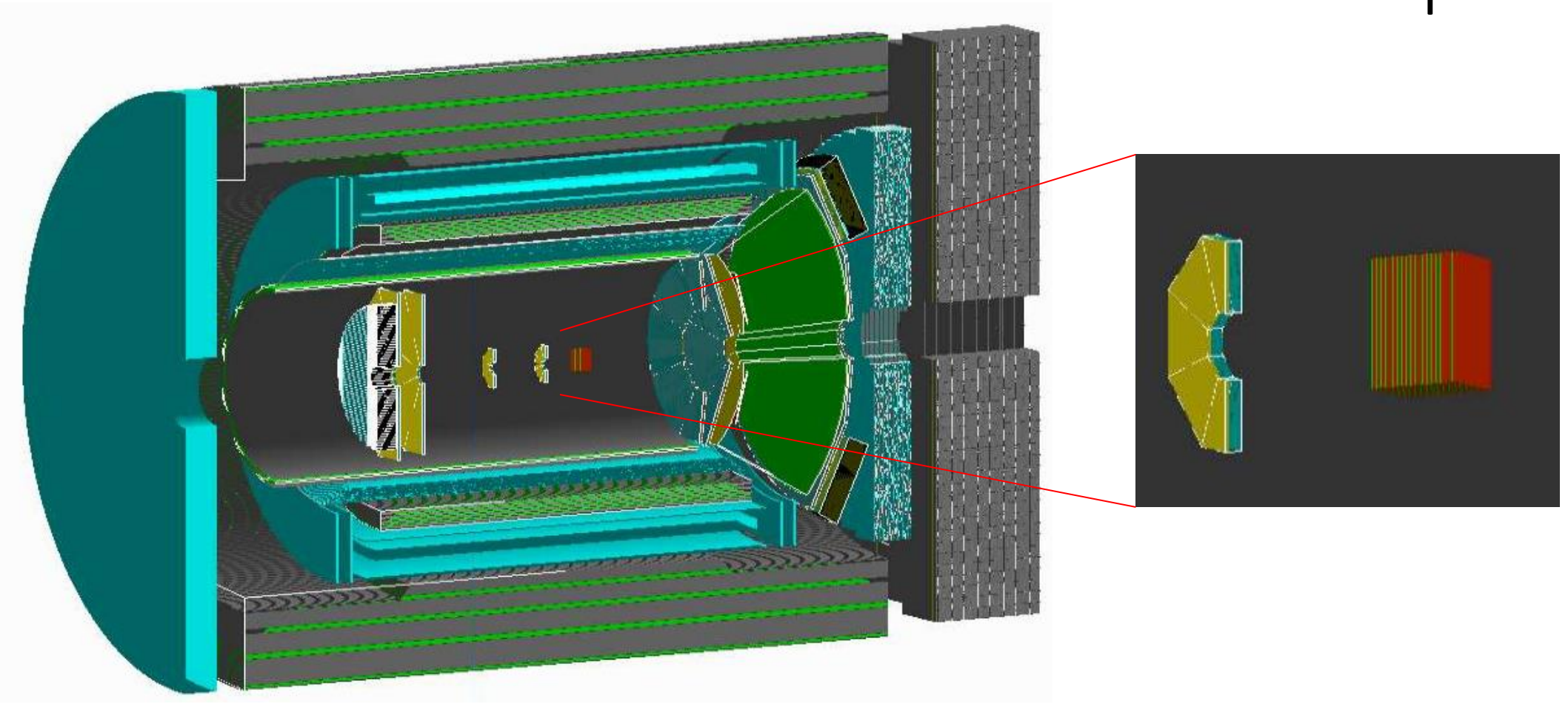
How is our calorimeter doing for 1GeV/c particles?



- Red: energy deposited in Scintillator
- Green: Energy deposited in absorber
- Size of our calorimeter is roughly right
  - We catch all electrons
  - We contain the electron energy
  - Most Pions basically leave a MIP signal
  - Some do leave an electron like signal in the scintillator

We are done – not bad for 1 hour of work in the morning of the workshop

Add our calorimeter to a detector setup

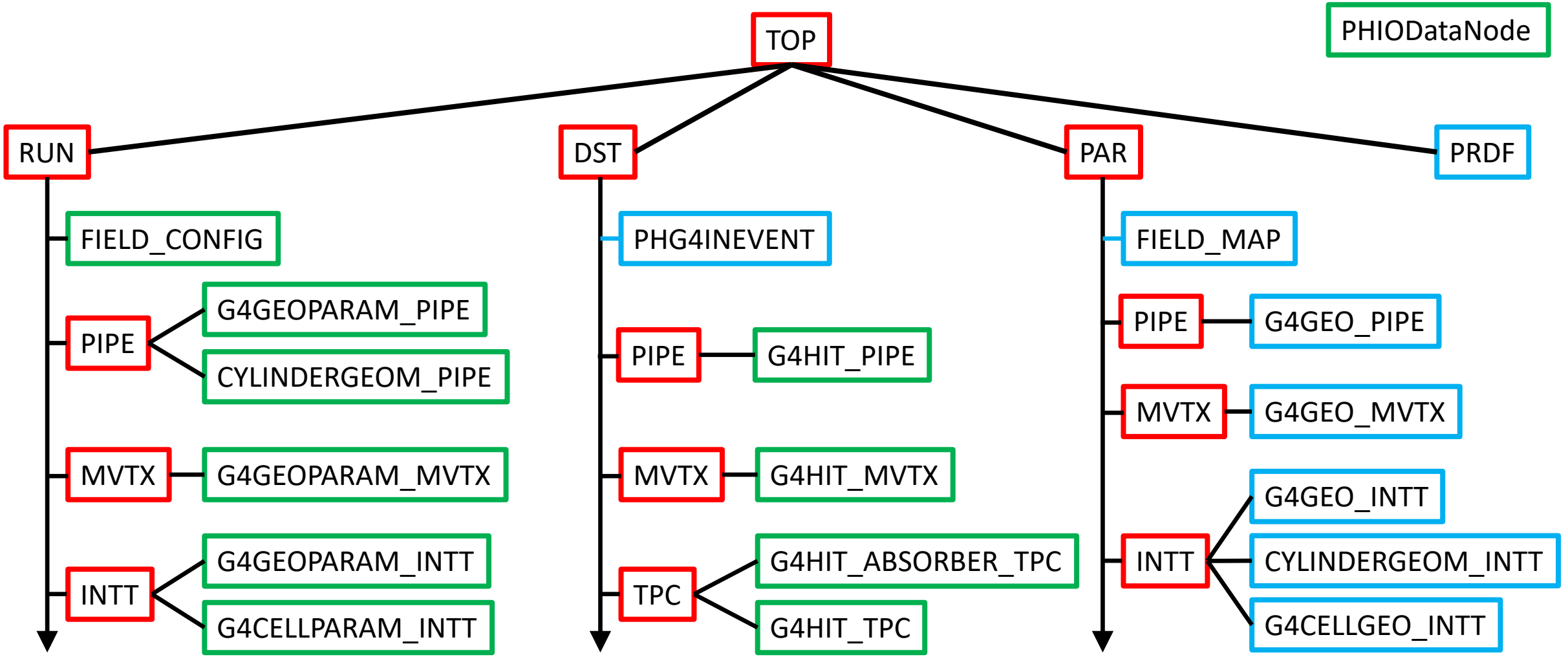


New detectors can be added on the macro level – the G4_User.C provides hooks for that
Not a single compilation was necessary for all that

The Node Tree

- The Node Tree is at the center of the Fun4All software universe (but it is 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 (smart pointers seem to be problematic – looking at ACTS track storage).
- We currently save 2 root trees in each output file, one which contains the eventwise information, one which contains the runwise information
- Input Managers recreate PHCompositeNode layout and put objects as PHIODataNodes on the node tree, output managers save layout and selected PHIODataNodes to a file.
- Fun4All can manage multiple independent node trees

Access to Data Objects, understanding our Node Tree



Phool Node Tree in Fun4All

```
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");

Phool Node Tree in Fun4All

```
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");

Phool Node Tree in Fun4All

```
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)

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");

Phool Node Tree in Fun4All

```
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");

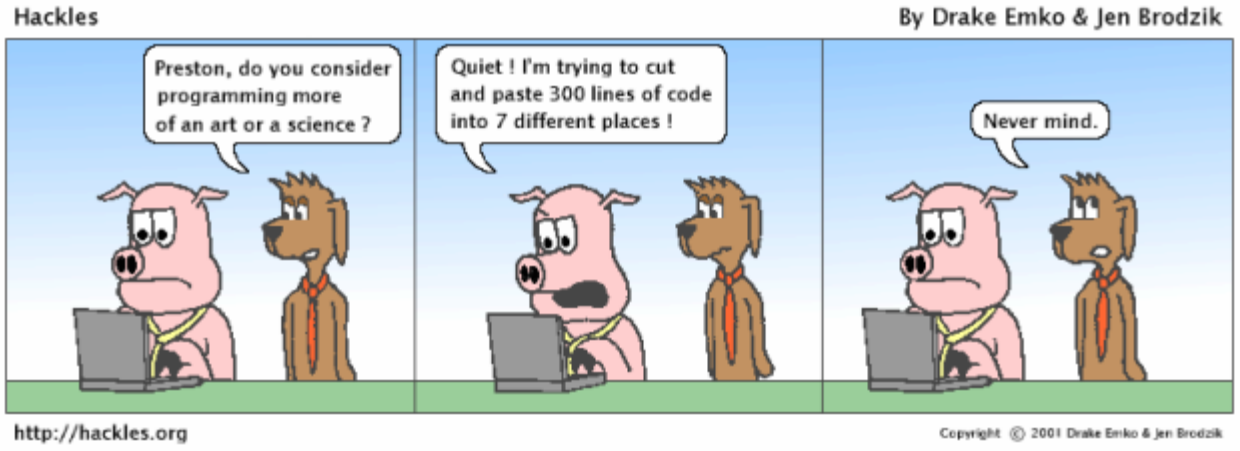
Your Analysis Module

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

Writing your own Analysis Module



To get help, type: CreateSubsysRecoModule.pl

Usage:
CreateSubsysRecoModule.pl <Module Name>
options:
--all : create also autogen.sh, configure.ac and Makefile.am
--overwrite : overwrite existing files

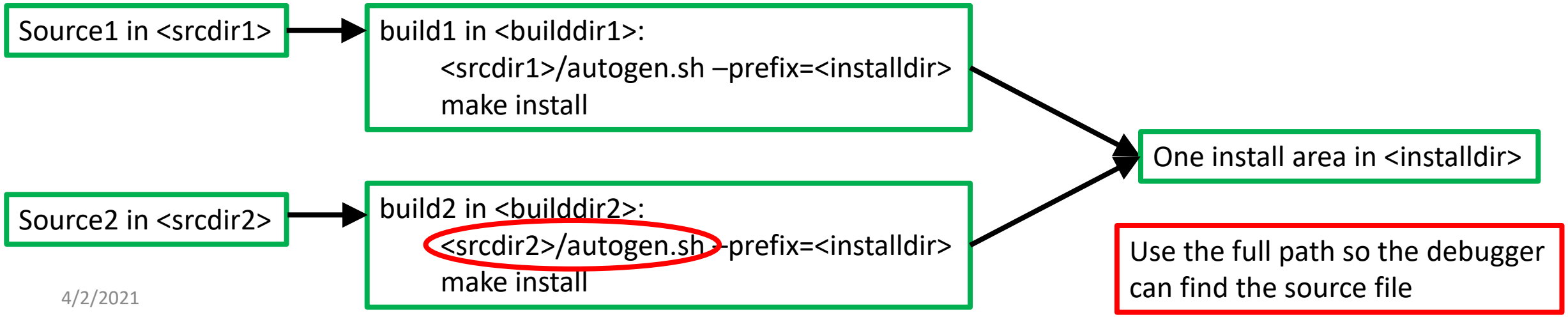
Creates <Module Name>.h and <Module Name>.cc with a cout in all available methods. Much better starting point than cutting and pasting from an existing module

The Makefile.am and configure.ac do not contain the instructions needed for i/o classes
Sorry - you will still need to cut and paste this from existing sources

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

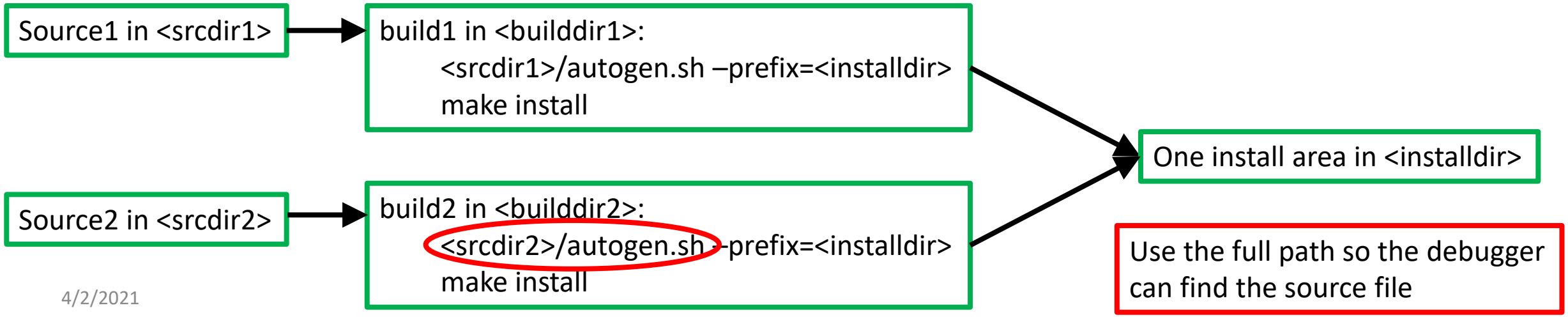
Keep your source and build areas separately, use common install area



How to use your compiled package

- The default setup from the sphenix setup scripts does not add your installation area
- ROOT6 is peculiar with finding includes
- So we have a script to set up everything for your installation area:

source $OPT_FUN4ALL/bin/setup_local.csh (or sh) <installdir>



Debugging/Profiling – use professional tools

~~cout~~

- gdb: first line of defense, compile with CXXFLAGS='-g –std=c++17'
- Runtime checks
  - valgrind: finds most memory corruptions
  - insure: rcf only - finds array boundary violations
- static code analysers
  - Cppcheck
  - Coverity
  - Scan-build

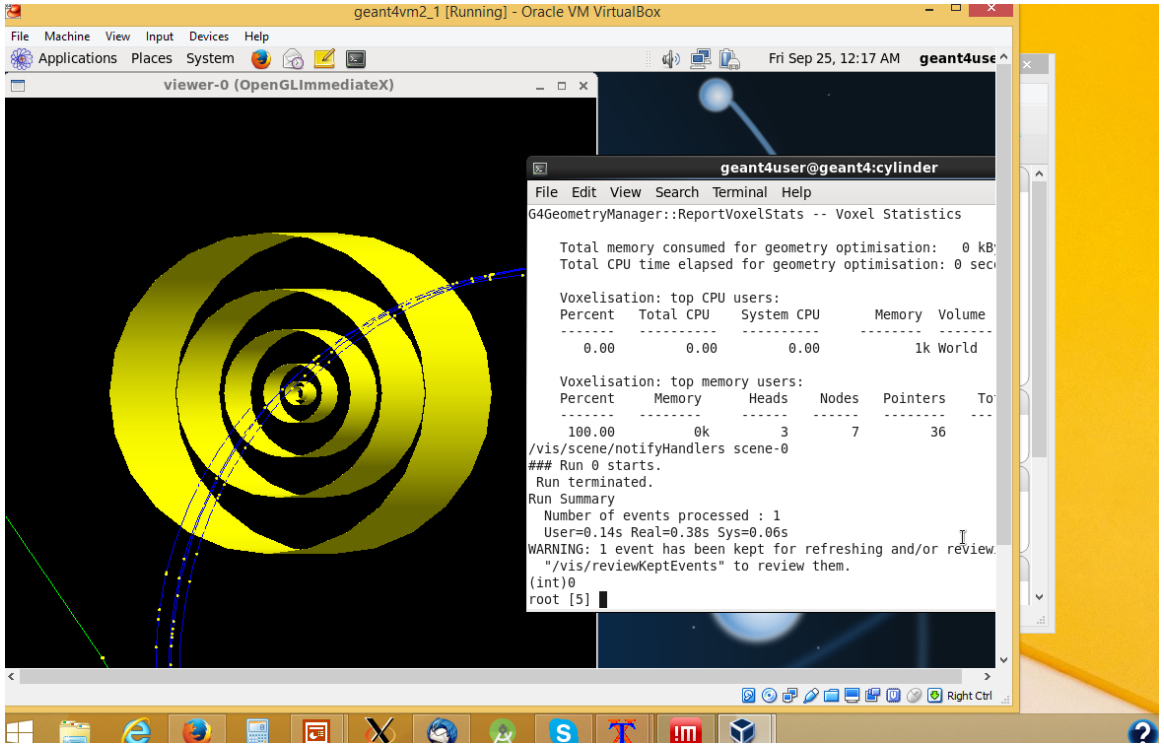
Profiling:

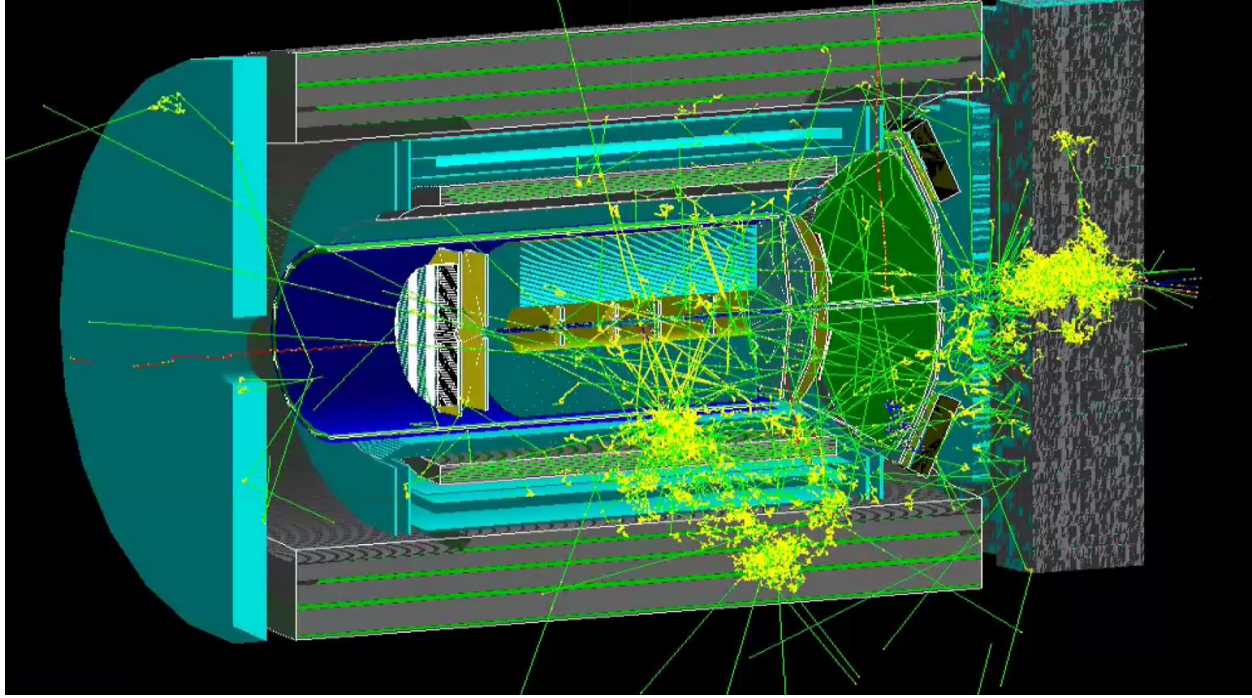
- Callgrind: where does your code spend all its time?
- Massif: where did all this memory go?

Details about these tools in our wiki:
https://wiki.bnl.gov/sPHENIX/index.php/Tools

Distribution

Containers are a solution for the problem to support many compilers and OS's guaranteeing identical outcomes. Can be run on the OSG as is as well as HPC. Full development possible in container





VM's on laptops do work for simple setups or fast eic-smear type simulations, but as soon as you do "real GEANT4 simulation runs" for detector designs running over 1000's (mio's) of events and have to deal with calorimeter showers that laptop approach reaches its limits quickly

Software deployment

- One script to rule them all

  csh: source /cvmfs/eic.opensciencegrid.org/ecce/gcc-8.3/opt/fun4all/core/bin/ecce_setup.csh –n <version>

  bash: source /cvmfs/eic.opensciencegrid.org/ecce/gcc-8.3/opt/fun4all/core/bin/ecce_setup.sh –n <version>

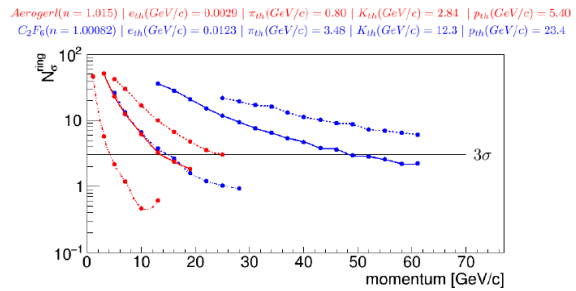
- Builds
  - New: daily build of current software, 4 versions (new.1, new.2,…) with roll over
  - Ana: weekly archival build (stable, kept forever) ∞ versions (ana.1,…)
  - Likely more to come (play, test, debug,…)

  Builds are self contained with copy of 3rd party libs, distributed via cvmfs

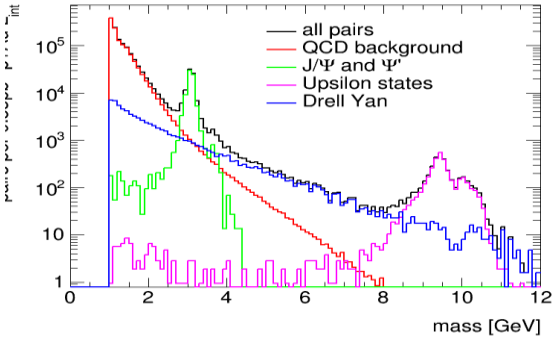
Since we all use the same OS and base libraries we can reproduce problems and help debugging them, no “what OS?”, “what compiler?”, “what…?”

Summary

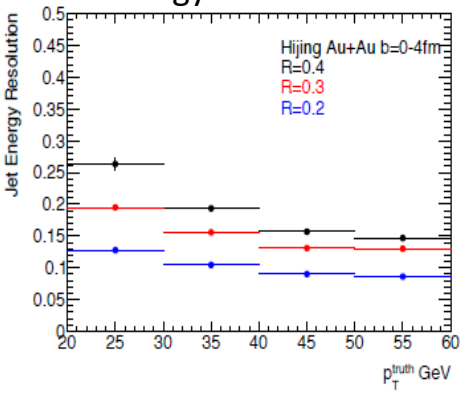
Rich PID

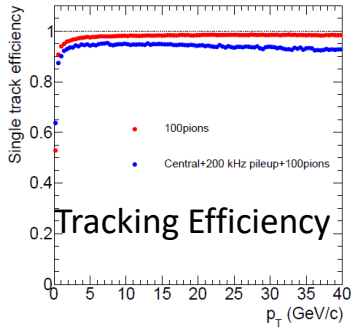


Forward Quarkonia

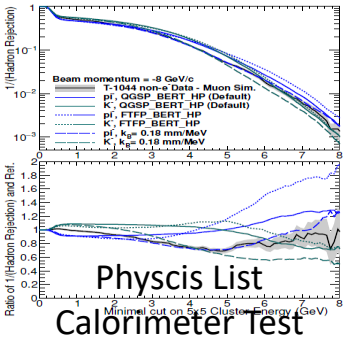


Central Barrel Jet Energy Resolution





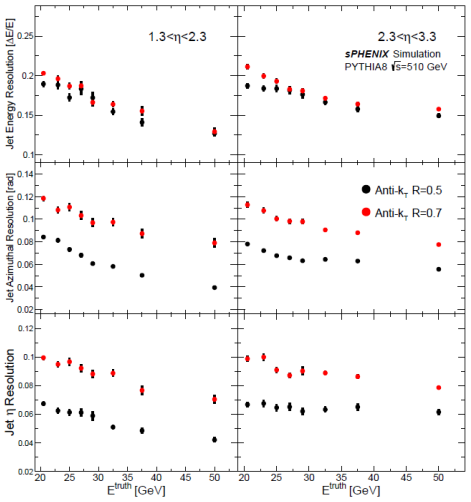
Tracking Efficiency

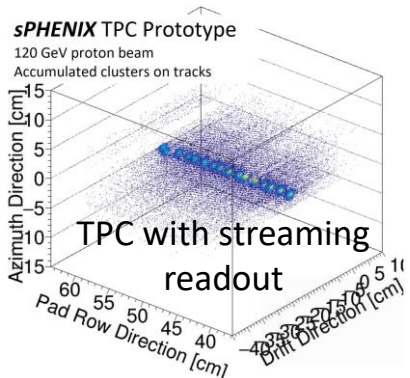


Physcis List Calorimeter Test

- Fun4All is a battle tested framework which has seen many billions of events – real and simulated
- Fun4All was designed to allow rapid and agile development, just what you need for detector development
- A lot of the heavy lifting has been done already – we used it to shepherd one large detector from the first drawings on a napkin through the CD process
- You can concentrate on the actual running and analysis of simulations
- You have your own repositories to develop anything you want

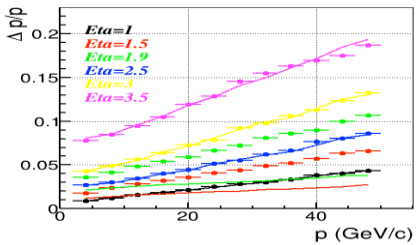
Forward Jet Energy Resolution

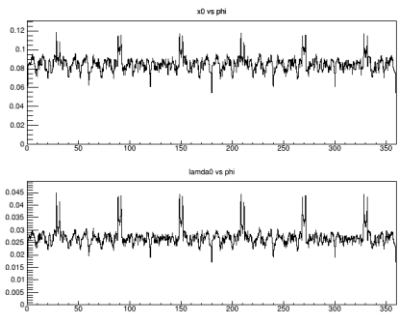




TPC with streaming readout

Forward Momentum Resolution





Material budget of sPHENIX inner tracker