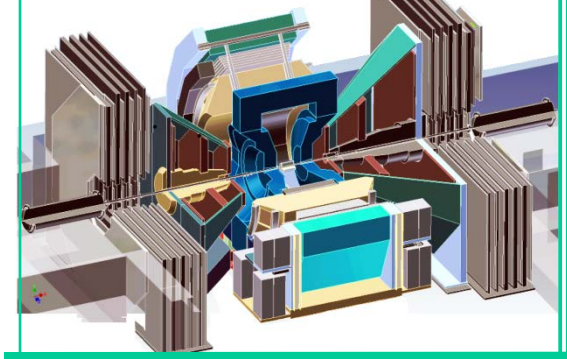
Fun4all

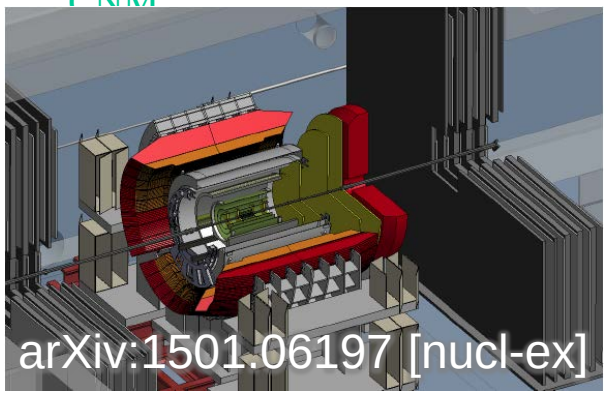
Where The Fun Always Shines!

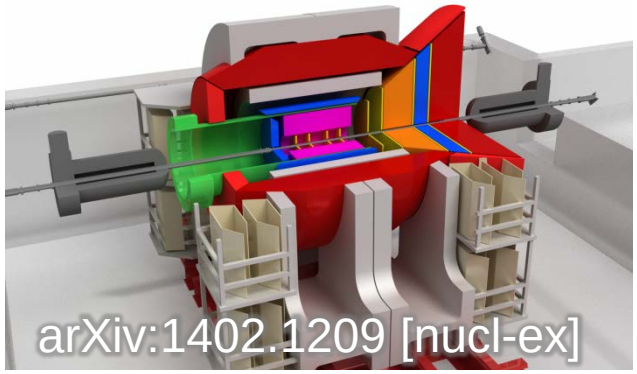
Evolution of the PHENIX experiment

Documented: http://www.phenix.bnl.gov/plans.html

Current PHENIX	*s*PHENIX (+fsPHENIX)	An EIC detector
• 14y+ operation 100+M$ investment • Broad spectrum of physics (QGP, Hadron Physics, DM) • 140+ published papers to date • Last run in this form 2016	‣ Comprehensive central upgrade base on BaBar magnet ‣ Rich jet and beauty quarkonia physics program → nature of QGP ‣ fsPHENIX : forward tracking, Hcal and muon ID → Spin, CNM	‣ Path of PHENIX upgrade leads to a capable EIC detector ‣ Large coverage of tracking, calorimetry and PID ‣ Open for new collaboration/new ideas







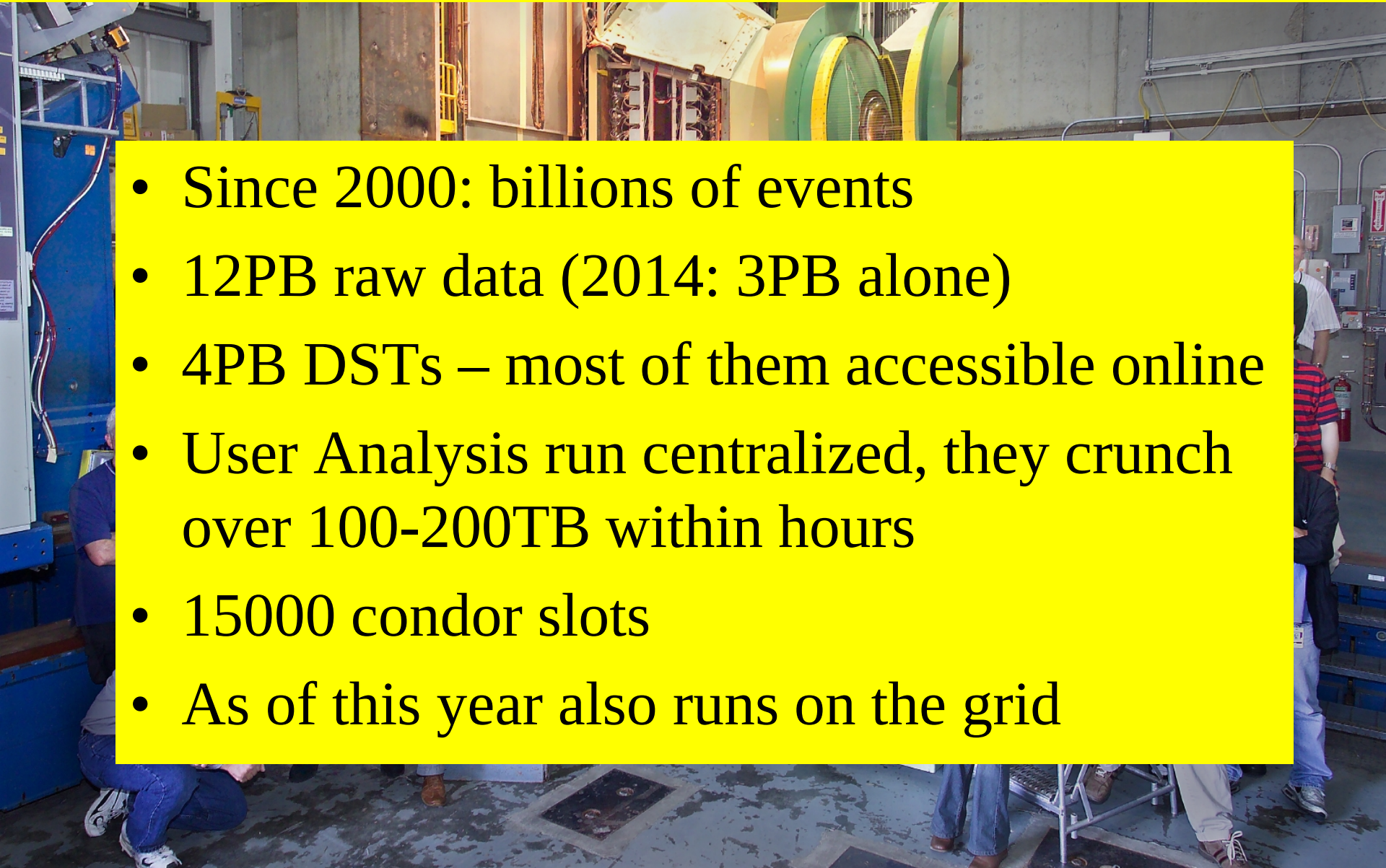
~2000 2017→2020 ~2025 Time

RHIC: A+A, spin-polarized p+p, spin-polarized p+A

EIC: e+p, e+A

PHENIX is big data (a'la Markus yesterday)

PHENIX is big data (a'la Markus yesterday)

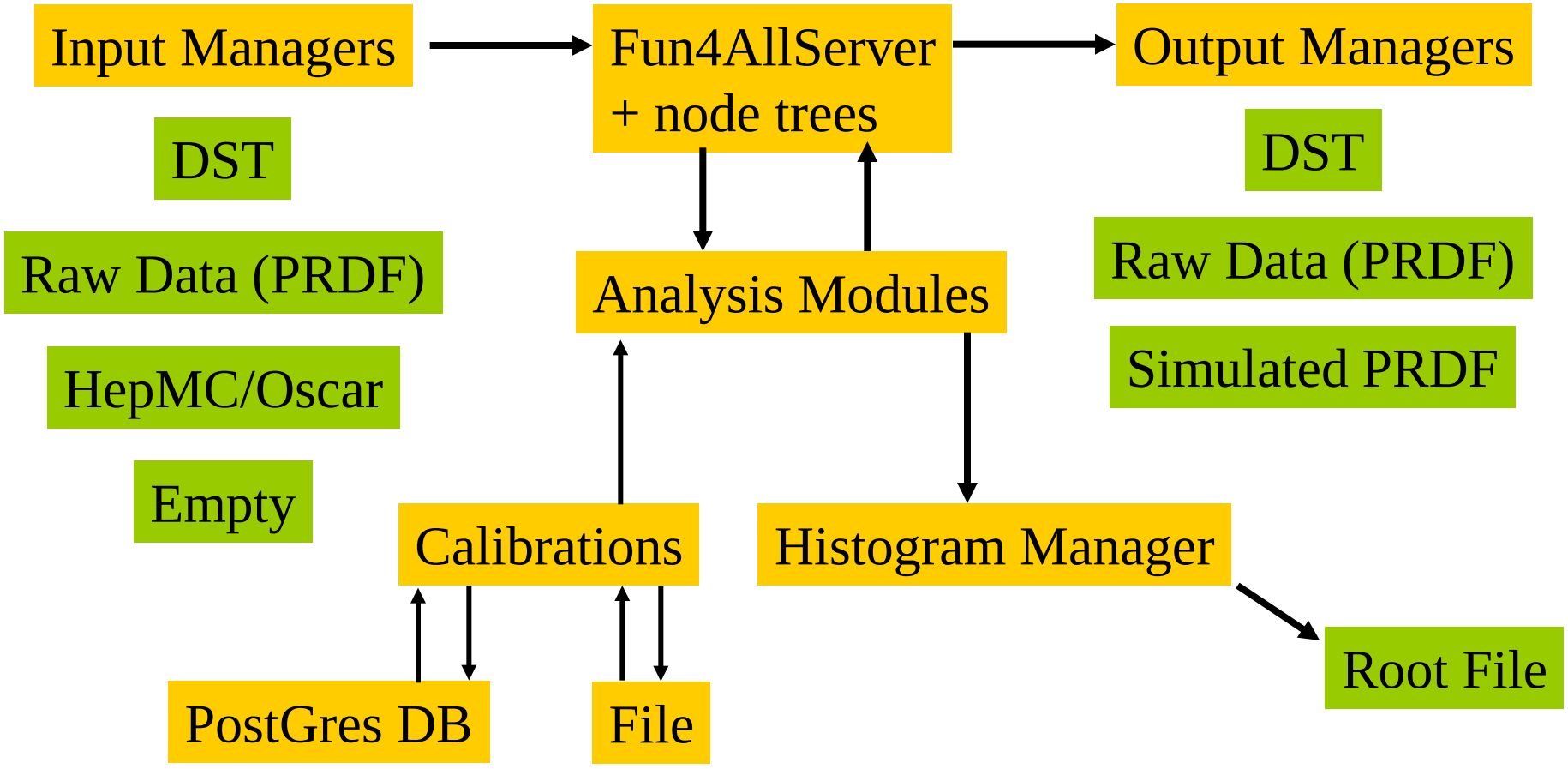


- Since 2000: billions of events
- 12PB raw data (2014: 3PB alone)
- 4PB DSTs – most of them accessible online
- User Analysis run centralized, they crunch over 100-200TB within hours
- 15000 condor slots
- As of this year also runs on the grid

History

- Development started in 2002, in use by PHENIX from 2003 on for reconstruction of real and simulated data, embedding and analysis
- Needed to get many subsystems who developed their code independently and without coordination under one umbrella
- Development driven by reconstruction and analysis needs (and the urgent need to process incoming data) – not by “beauty” or some abstract design considerations (or “rules”)
- KISS + Modularity key to be able to evolve and adapt
- GEANT3 based sims were always run inside their own framework, their output file was read in.

Structure of Fun4All



That’s all there is to it (8000 lines of code)

The Node Tree

- The Node Tree is at the center of the Phenix software universe (but it's more or less invisible to you). It's the way we organize our data.
- **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).
- 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

Node Tree for sPHENIX

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

```
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
  DST (PHCompositeNode)/
    G4HIT_HCALIN (PHIODataNode)
    G4HIT_ABSORBER_HCALIN (PHIODataNode)
    G4HIT_HCALOUT (PHIODataNode)
    G4HIT_ABSORBER_HCALOUT (PHIODataNo
    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)
```

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

Node Tree for sPHENIX

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

```
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 (PHIODataNo
  RUN (PHCompositeNode)/
    CYLINDERGEOM_SVTX (PHIODataNode)
    CYLINDERGEOM_SVTXSUPPORT (PHIODataNode)
    CYLINDERGEOM_EMCELECTRONICS_0 (PHIODataNode)
    CYLINDERGEOM_HCALIN_SPT (PHIODataNode)
PAR (PHCompositeNode)/
    SVTX (PHCompositeNode)/
      SvtxBeamSpot (PHIODataNode)
```

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

Node Tree for sPHENIX

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

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

Users can add their own branches. Resetting the objects is their responsibility.

The PAR node will probably turn into the node for calibrations which we want to keep on the DST

Keep it simple - Analysis Modules

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

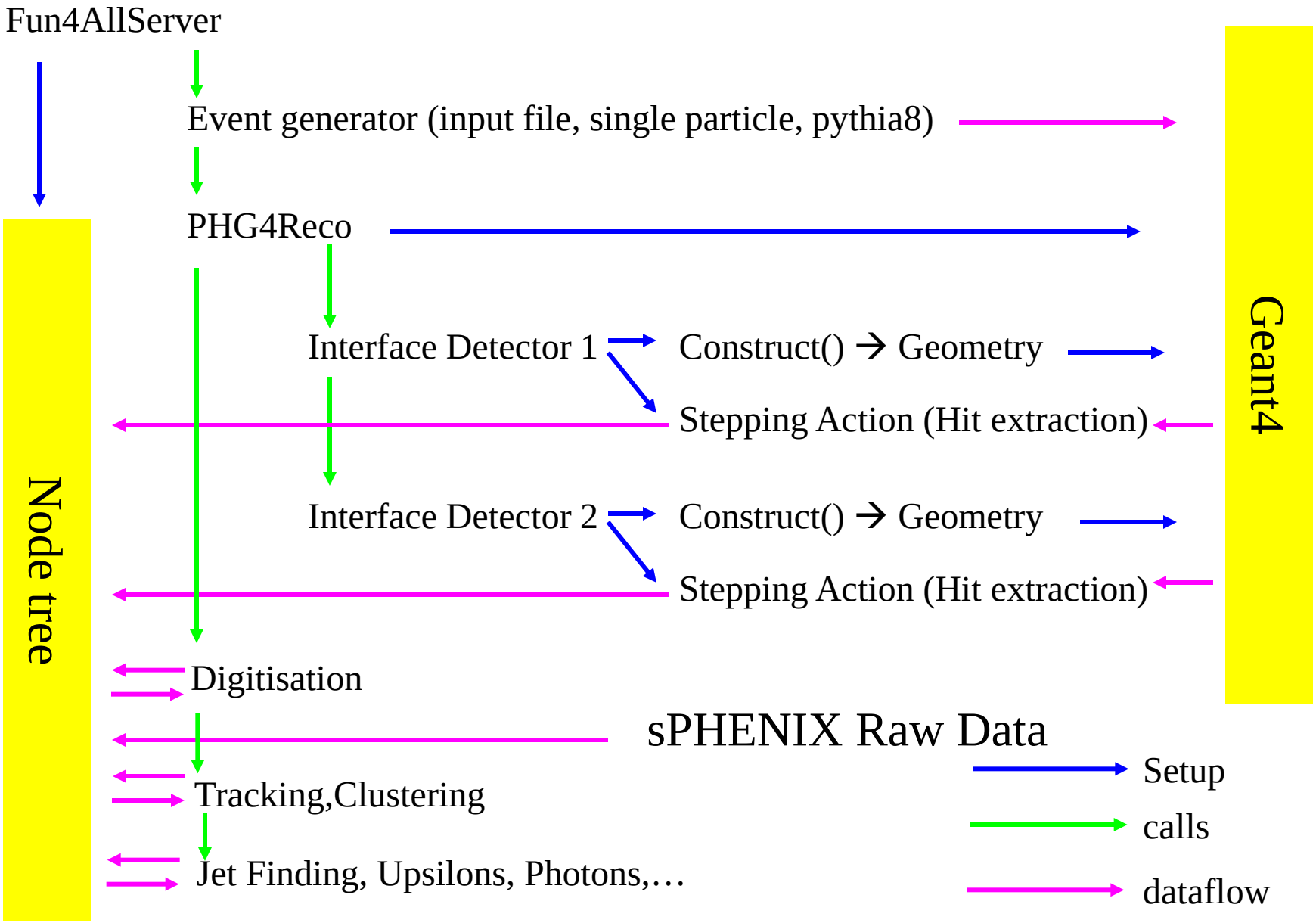
What to take from here

- Fun4All is a well developed mature framework but not overcomplicated, features driven by real processing and analysis needs
- Standard C++, Root with shared libraries, configured and run by CINT macros, Root only used if it is the best/only solution
- Writing multiple streams, event selection by modules
- Synchronized parallel reading of input files
- Calibration Database scheme
- Easy addition of external packages (fastjet, rave, genfit)
- Users have to write (simple) C++ analysis code

sPHENIX

- Effort started 4 years ago, the decision was to go with G4 (hadron calorimeters) and use Fun4All as framework so all development could concentrate on G4
- G4 simulations are implemented as an analysis module, the G4 command line interface is still intact and can be called from the root prompt
- Modular – each detector runs by itself
- Generic cylinders, boxes and cones available if you want to try something quick, a “black hole” provides leakage detections
- Higher level geometries: spacal (1d/2d projective), hcal with tilted slats, svtx ladders
- Truth information is propagated for evaluation
- Configurable on macro level
- If you want to scan configuration space – impractical to generate new geometry and recompile
- Code: https://github.com/sPHENIX-Collaboration/coresoftware
- Used to analyze upcoming Test Beam Data

G4 program flow in sPHENIX



Here a more scary geometry

```
G4VSolid*
PHG4OuterHcalDetector::ConstructSteelPlate(G4LogicalVolume* hcalenvelope)
{
  // calculate steel plate on top of the scinti box. Lower edge is the upper edge of
  // the scintibox + 1/2 the airgap
  double mid_radius = params->inner_radius + (params->outer_radius - params->inner_radius) / 2.;
  // first the lower edge, just like the scinti box, just add the air gap
  // and calculate intersection of edge with inner and outer radius.
  Point_2 p_in_1(mid_radius, 0); // center of lower scintillator
  double angle_mid_scinti = M_PI / 2. + params->tilt_angle / rad;
  double xcoord = params->scinti_gap / 2. * cos(angle_mid_scinti / rad) + mid_radius;
  double ycoord =   params->scinti_gap / 2. * sin(angle_mid_scinti / rad) + 0;
  Point_2 p_loweredge(xcoord, ycoord);
  Line_2 s2(p_in_1, p_loweredge); // center vertical
  Line_2 perp =  s2.perpendicular(p_loweredge); // that is the lower edge of the steel plate
  Point_2 sc1(params->inner_radius, 0), sc2(0, params->inner_radius), sc3(-params->inner_radius, 0);
  Circle_2 inner_circle(sc1, sc2, sc3);
  vector< CGAL::Object > res;
  CGAL::intersection(inner_circle, perp, std::back_inserter(res));
  Point_2 lowerleft;
  vector< CGAL::Object >::const_iterator iter;
  for (iter = res.begin(); iter != res.end(); ++iter)
    {
      CGAL::Object obj = *iter;
      if (const std::pair<CGAL::Circular_arc_point_2<Circular_k>, unsigned> *point = CGAL::object_cast<std::pair<CGAL::Circular_arc_point_2<
                    {
                      if (CGAL::to_double(point->first.x()) > 0)
                        {
                          Point_2 pntmp(CGAL::to_double(point->first.x()), CGAL::to_double(point->first.y()));
                          lowerleft = pntmp;
                        }
```

Here a more scary geometry

```
                }
    else
                {
                  cout << "CGAL::Object type not pair..." << endl;
                }
  }
Point_2 so1(params->outer_radius, 0), so2(0, params->outer_radius), so3(-params->outer_radius, 0);
Circle_2 outer_circle(so1, so2, so3);
res.clear(); // just clear the content from the last intersection search
CGAL::intersection(outer_circle, perp, std::back_inserter(res));
Point_2 lowerright;
for (iter = res.begin(); iter != res.end(); ++iter)
  {
    CGAL::Object obj = *iter;
    if (const std::pair<CGAL::Circular_arc_point_2<Circular_k>, unsigned> *point = CGAL::object_cast<std::pair<CGAL::Circular_arc_point_2<
                {
                  if (CGAL::to_double(point->first.x()) >  CGAL::to_double(p_loweredge.x()))
                    {
                      Point_2 pntmp(CGAL::to_double(point->first.x()), CGAL::to_double(point->first.y()));
                      lowerright = pntmp;
                    }
                }
    else
                {
                  cout << "CGAL::Object type not pair..." << endl;
                }
  }
// now we have the lower left and rigth corner, now find the upper edge
// find the center of the upper scintilator
```

Here a more scary geometry

```
double phi_midpoint = 2 * M_PI / params->n_scinti_plates;
  double xmidpoint = cos(phi_midpoint) * mid_radius;
  double ymidpoint = sin(phi_midpoint) * mid_radius;
  // angle of perp line at center of scintillator
  angle_mid_scinti = (M_PI / 2. - phi_midpoint) - (M_PI / 2. + params->tilt_angle / rad);
  double xcoordup = xmidpoint - params->scinti_gap / 2. * sin(angle_mid_scinti / rad);
  double ycoordup = ymidpoint - params->scinti_gap / 2. * cos(angle_mid_scinti / rad);
  Point_2 upperleft;
  Point_2 upperright;
  Point_2 mid_upperscint(xmidpoint, ymidpoint);
  Point_2 p_upperedge(xcoordup, ycoordup);
  {
    Line_2 sup(mid_upperscint, p_upperedge); // center vertical
    Line_2 perp =  sup.perpendicular(p_upperedge); // that is the upper edge of the steel plate
    Point_2 sc1(params->inner_radius, 0), sc2(0, params->inner_radius), sc3(-params->inner_radius, 0);
    Circle_2 inner_circle(sc1, sc2, sc3);
    vector< CGAL::Object > res;
    CGAL::intersection(inner_circle, perp, std::back_inserter(res));
    vector< CGAL::Object >::const_iterator iter;
    double pxmax = 0.;
    for (iter = res.begin(); iter != res.end(); ++iter)
      {
                CGAL::Object obj = *iter;
                if (const std::pair<CGAL::Circular_arc_point_2<Circular_k>, unsigned> *point = CGAL::object_cast<std::pair<CGAL::Circular_arc
                  {
                    if (CGAL::to_double(point->first.x()) > pxmax)
                      {
                                pxmax = CGAL::to_double(point->first.x());
                                Point_2 pntmp(CGAL::to_double(point->first.x()), CGAL::to_double(point->first.y()));
                                upperleft = pntmp;
                      }
```

Here a more scary geometry

```
                else
                  {
                    cout << "CGAL::Object type not pair..." << endl;
                  }
    }
  Point_2 so1(params->outer_radius, 0), so2(0, params->outer_radius), so3(-params->outer_radius, 0);
  Circle_2 outer_circle(so1, so2, so3);
  res.clear(); // just clear the content from the last intersection search
  CGAL::intersection(outer_circle, perp, std::back_inserter(res));
  for (iter = res.begin(); iter != res.end(); ++iter)
    {
                CGAL::Object obj = *iter;
                if (const std::pair<CGAL::Circular_arc_point_2<Circular_k>, unsigned> *point = CGAL::object_cast<std::pair<CGAL::Circular_arc
                  {
                    if (CGAL::to_double(point->first.x()) >  CGAL::to_double(p_loweredge.x()))
                      {
                                Point_2 pntmp(CGAL::to_double(point->first.x()), CGAL::to_double(point->first.y()));
                                upperright = pntmp;
                      }
                  }
                else
                  {
                    cout << "CGAL::Object type not pair..." << endl;
                  }
    }
}
// the left corners are on a secant with the inner boundary, they need to be shifted
// to be a tangent at the center
ShiftSecantToTangent(lowerleft, upperleft, upperright, lowerright);
```

Here a more scary geometry

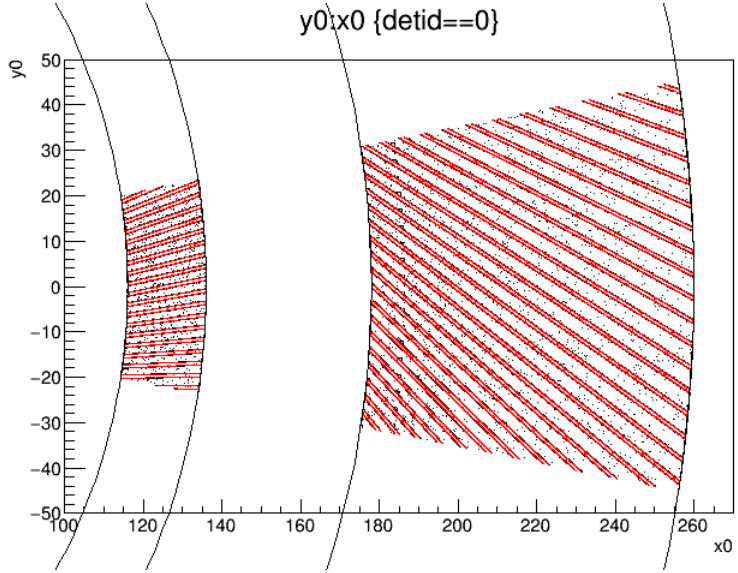
```
G4TwoVector v1(CGAL::to_double(upperleft.x()), CGAL::to_double(upperleft.y()));
 G4TwoVector v2(CGAL::to_double(upperright.x()), CGAL::to_double(upperright.y()));
 G4TwoVector v3(CGAL::to_double(lowerright.x()), CGAL::to_double(lowerright.y()));
 G4TwoVector v4(CGAL::to_double(lowerleft.x()), CGAL::to_double(lowerleft.y()));
 std::vector<G4TwoVector> vertexes;
 vertexes.push_back(v1);
 vertexes.push_back(v2);
 vertexes.push_back(v3);
 vertexes.push_back(v4);
 G4TwoVector zero(0, 0);
 G4VSolid* steel_plate_uncut =  new G4ExtrudedSolid("SteelPlateUnCut",
                                                                                                vertexes,
                                                                                                params->size_z  / 2.0,
                                                                                                zero, 1.0,
                                                                                                zero, 1.0);


    G4RotationMatrix *rotm = new G4RotationMatrix();
            rotm->rotateX(-90 * deg);

 // now cut out space for magnet at the ends
     G4VSolid* steel_firstcut_solid = new G4SubtractionSolid("SteelPlateFirstCut",steel_plate_uncut,steel_cutout_for_magnet,rotm,G4ThreeVector
     //   DisplayVolume(steel_plate_uncut, hcalenvelope);
   //    DisplayVolume(steel_cutout_for_magnet, hcalenvelope);
   //    DisplayVolume(steel_cutout_for_magnet, hcalenvelope,rotm);
      //    DisplayVolume(steel_firstcut_solid, hcalenvelope);
     rotm = new G4RotationMatrix();
            rotm->rotateX(90 * deg);
     G4VSolid* steel_cut_solid = new G4SubtractionSolid("SteelPlateCut",steel_firstcut_solid,steel_cutout_for_magnet,rotm,G4ThreeVector(0,0,0)
     //          DisplayVolume(steel_cut_solid, hcalenvelope);

 return steel_cut_solid;
```

And this is the hcal setup macro

```
PHG4InnerHcalSubsystem *hcal;
 hcal = new PHG4InnerHcalSubsystem("HCALIN");
 hcal->SetMaterial("SS310"); // SS310 stainless steel
 // these are all the defaults
 // hcal->SetGapWidth(0.85);
 // hcal->SetScintiThickness(0.7);
 // hcal->SetNumScintiPlates(5*64);
 // hcal->SetTiltViaNcross(4);
 hcal->SetTiltAngle(10);
 hcal->SetActive();
 hcal->SuperDetector("HCALIN");
 if (absorberactive)  hcal->SetAbsorberActive();
 hcal->OverlapCheck(overlapcheck);
 //hcal->SetLightCorrection(116.0,0.85,135.0,1.0);
g4Reco->registerSubsystem( hcal );
```

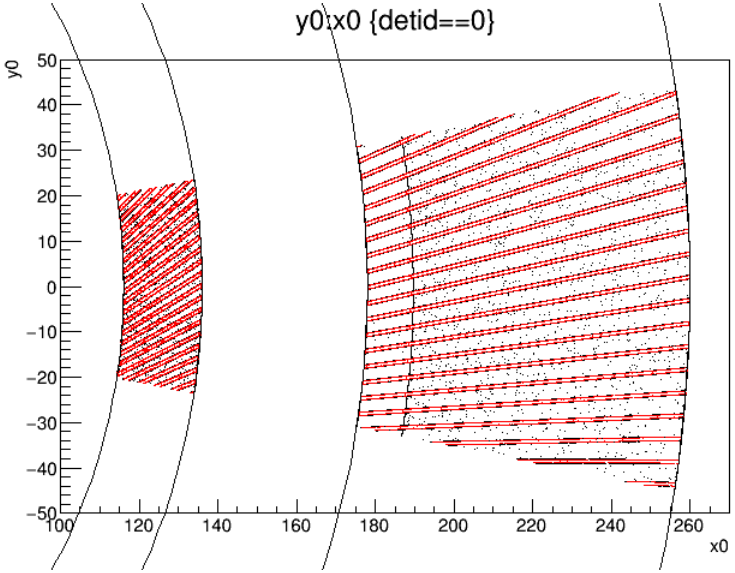
Yes we definitely hide complexity from our standard users who perform parameters scans like tilt angles

Hcal parameter space scan



Geantino Scan in $\pm 10^0$

Complex geometries needed for complex detectors

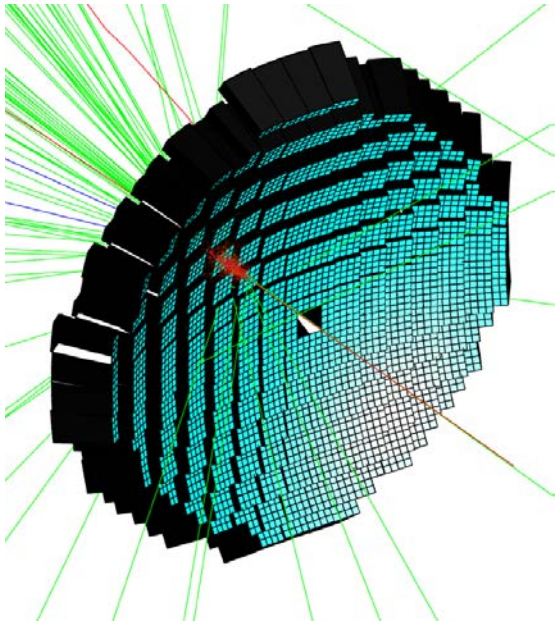


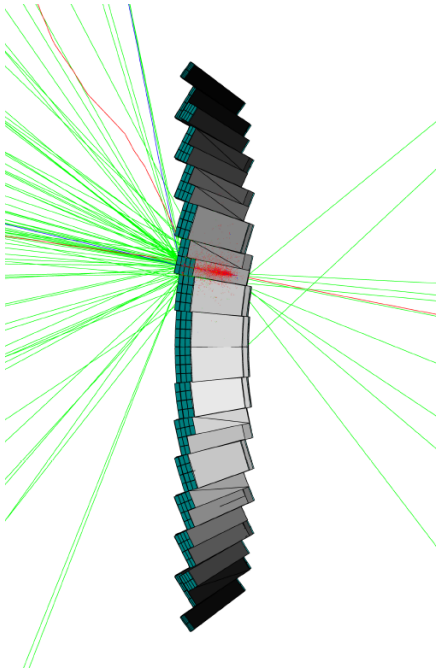
EIC detector based on Babar magnet

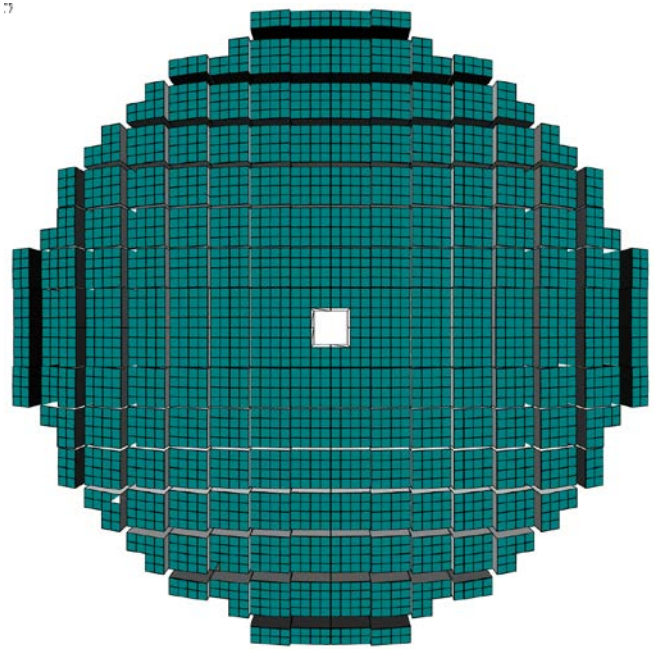
- Based on sPHENIX infra structure
- Uses sPHENIX barrel (without the silicon tracker)

Calorimeter/e-going EMCal

- Based on PANDA forward PWO4 design. This design requires high resolution calorimetry to determine energy for the scattered electron.
- Implemented in detail with tower and carbon fiber support structure
- Read more: Joshua LaBounty's talk on EIC-detector/"ePHENIX" meeting: https://indico.bnl.gov/conferenceDisplay.py?confId=1259



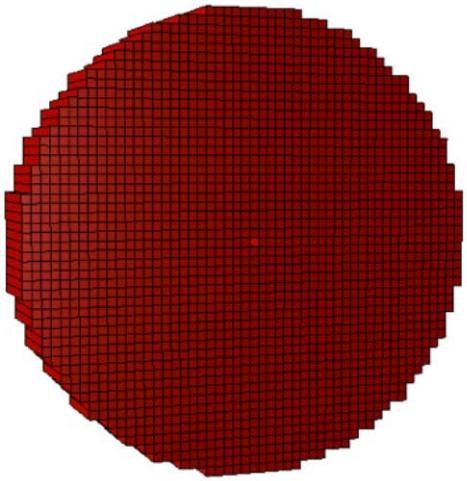




Calorimeter/h-going EMCal/HCal

- Forward ECal :
  - PHENIX-like lead/scintillator sampling calorimeter
- Forward HCal :
  - Iron scintillator sampling calorimeter
  - Hermetic connection with the central Hcal
  - Also intended to use in fsPHENIX for forward jet measurements
  - Read more: Nils Feege talk on EIC-detector/"ePHENIX" meeting: https://indico.bnl.gov/conferenceDisplay.py?confId=1311

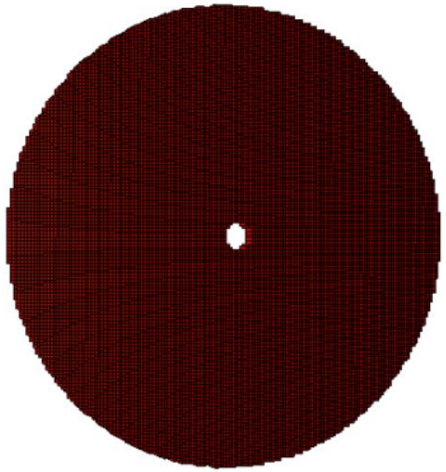
Forward HCal early concept



2046 Tower

each tower:
10x10 cm^2 sampling
100 cm long
30 layers
4/5 iron
1/5 scintillator

Forward ECal early concept

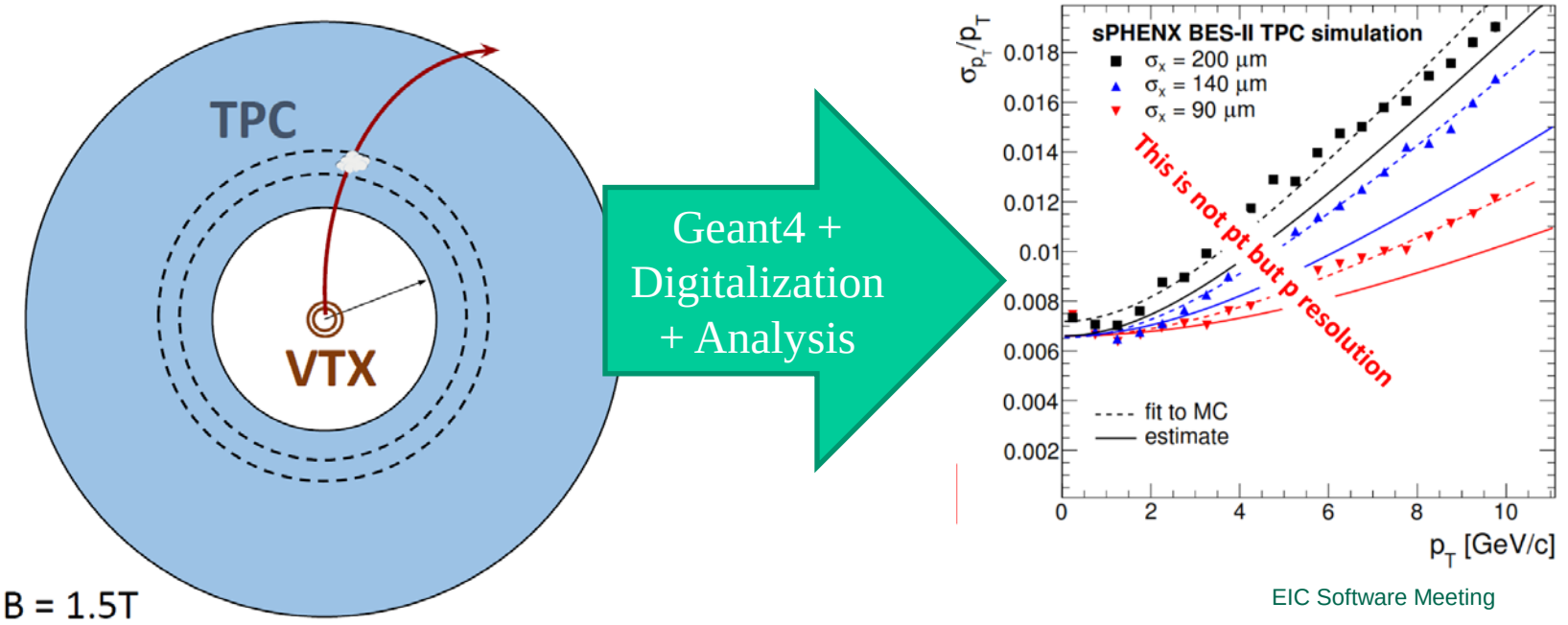


17350 Tower

each tower:
3x3 cm^2 sampling
17 cm long
60 layers
2/3 lead
1/3 scintillator

Tracking/central TPC tracking

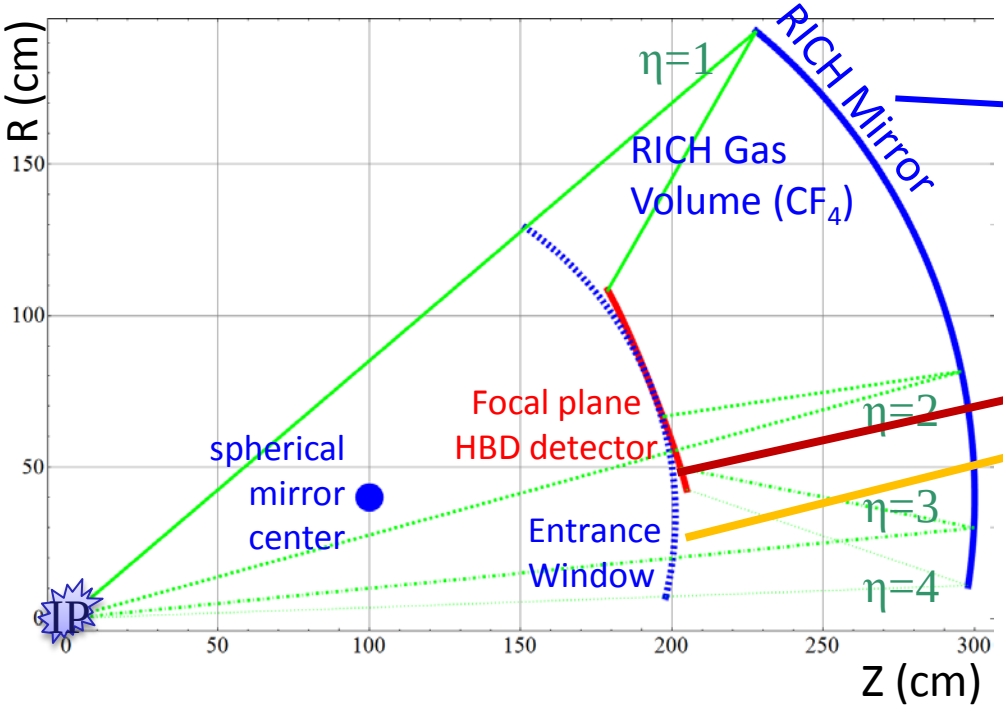
- Developed by sPHENIX TPC team
- Comprehensive analysis chain, active developments
  - Geant4 ionization -> initial charge -> Diffusion -> Avalance -> Raw data ADC -> clustering -> track finding -> Kalman filter fitting
- Read more: Alan Dion's talk on sPHENIX meeting: https://indico.bnl.gov/conferenceDisplay.py?confId=1337



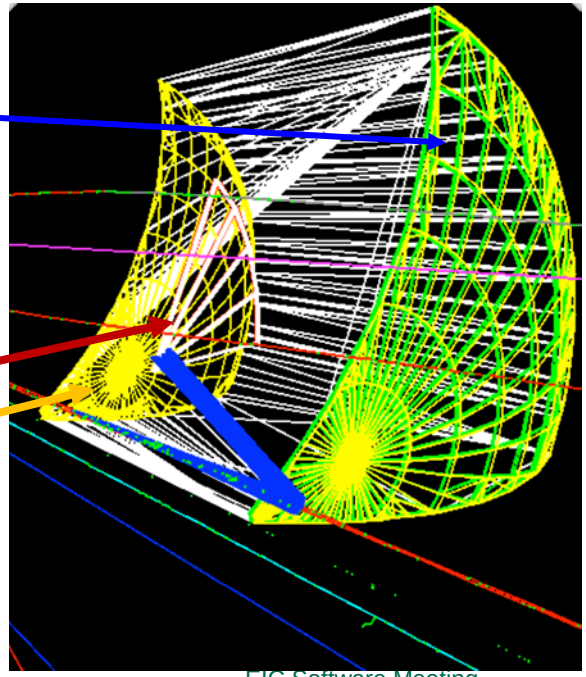
PID/gas Cheronkov

- Based on development of EIC R&D eRD6
  - 1m CF4 radiator/off-axis spherical mirror/HBD GEM photon sensor
  - Specially tuned yoke to minimize track bending inside RICH radiator volume
- Material/optics implemented in Geant4.
- Under tuning/developing likelihood based analysis code

In paper / test beam

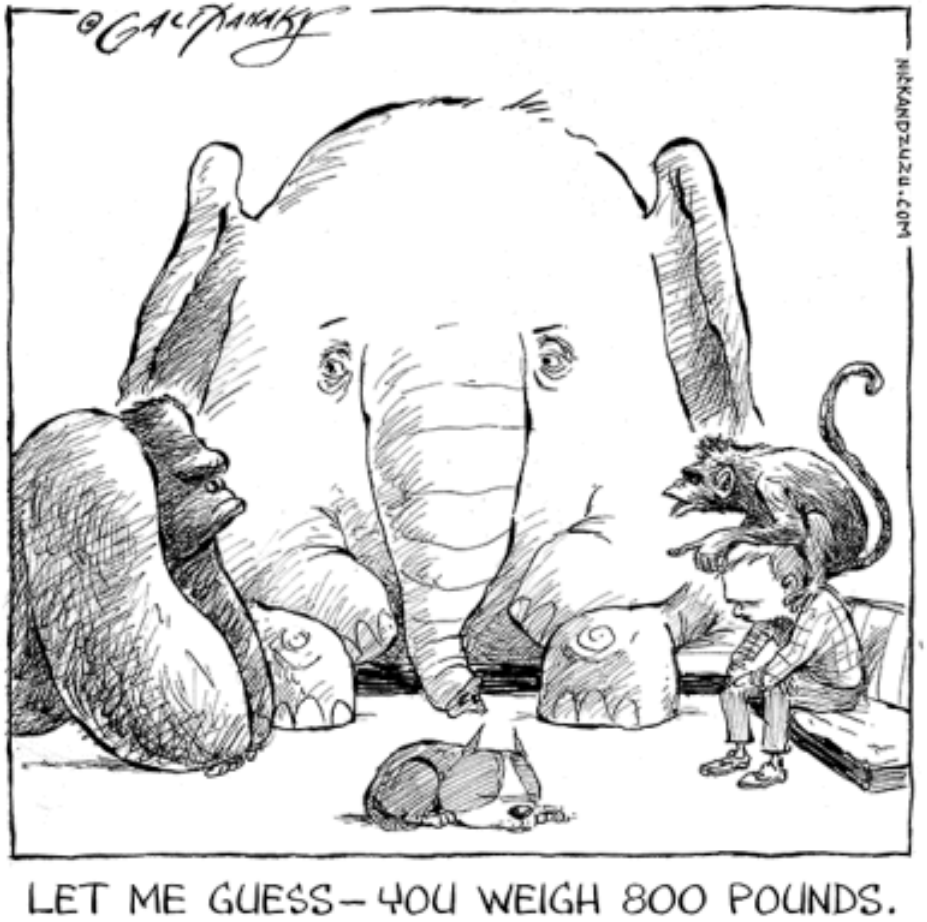


In Fun4All/Geant4



Open Issues

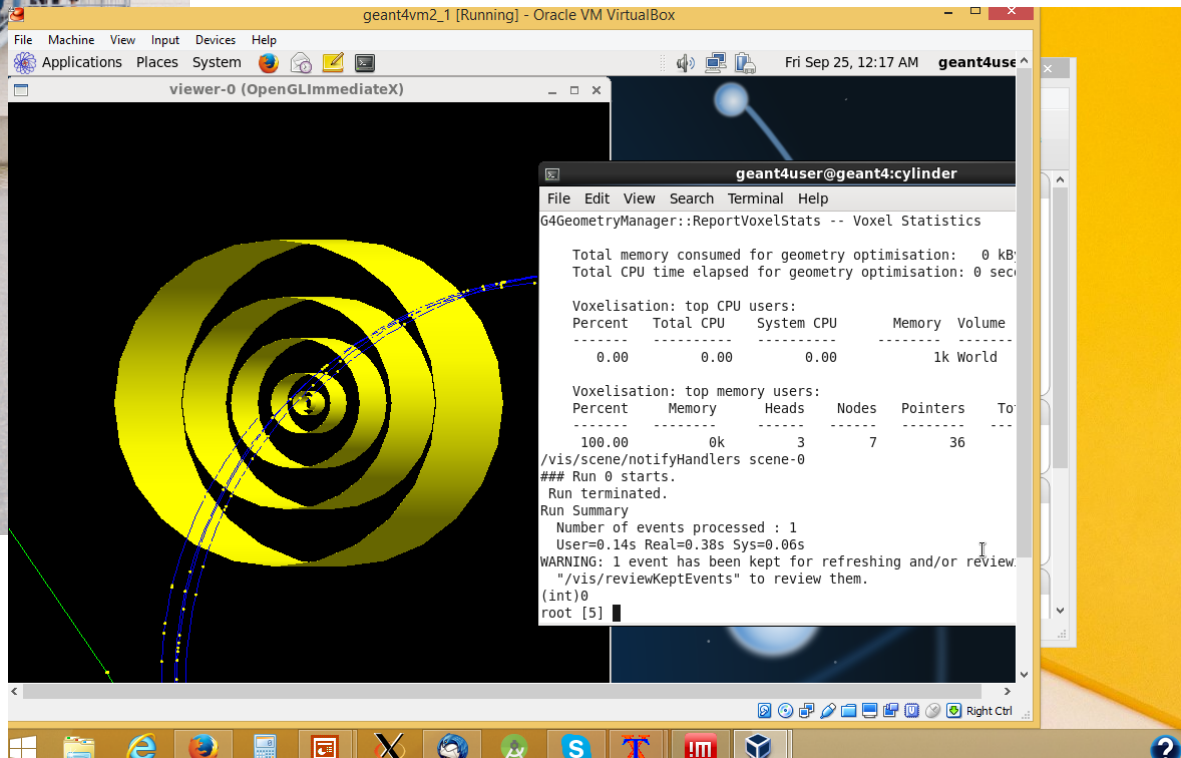
Geometry sharing between simulations and real data is our biggest challenge, current plan to use Tgeo



Virtual machine

Before everyone queues up for an rcf account

Try our virtual machine



Parting words

- PHENIX users in general like the simplicity of Fun4All, it is comparably easy to add your own analysis and run it
- Requiring students to code is not a bad thing
- Fun4All provides a common interface to our data – user analysis is run centrally, enabling code checking and freeing the user from dealing with all the problems of running thousands of jobs
- In PHENIX we still use simulation code which was written 10 years before the experiment started taking data, be careful what you drag along
- Customized frameworks don’t have to be large but you have a permanent position behind it
- Interactive capabilities extremely valuable for development/testing (run 1000 events, look, wash rinse repeat)