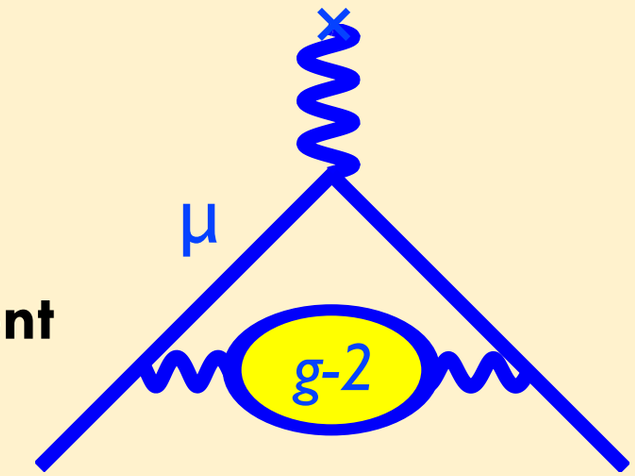
ART + GEANT4 = ARTG4

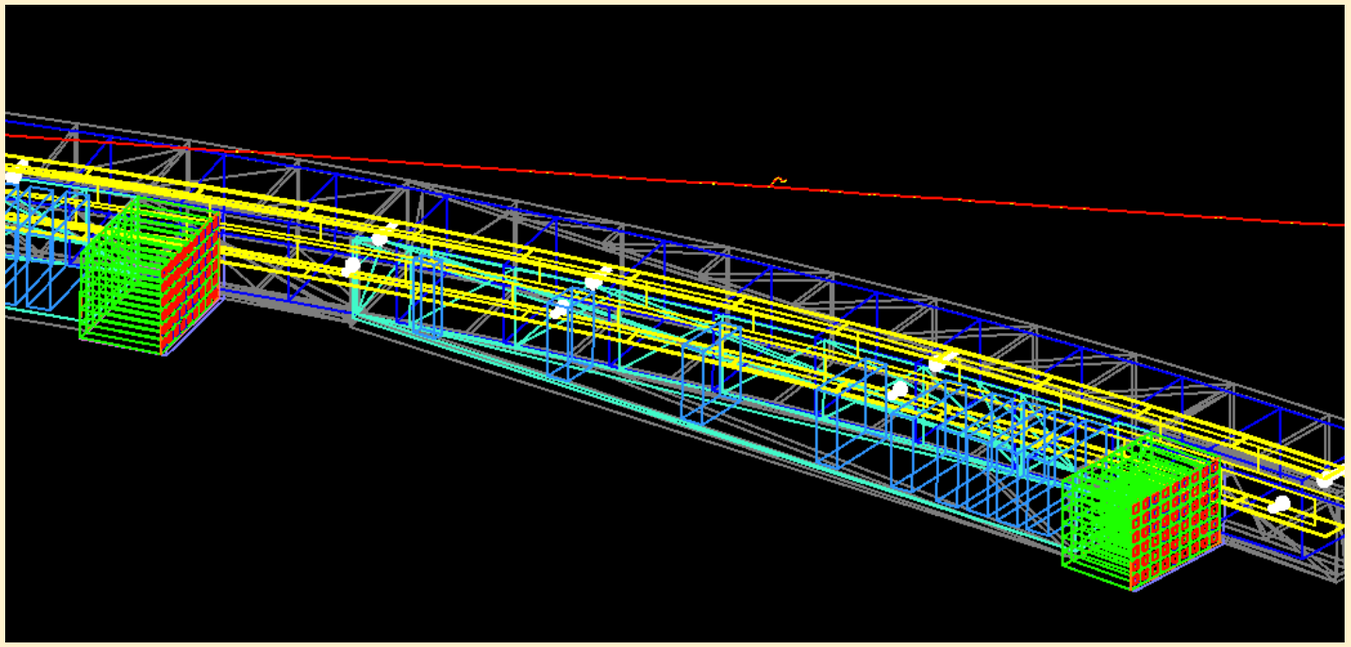
A Generic Framework for Geant4 Simulations

Adam Lyon & Tasha Arvanitis*

Fermilab Scientific Computing Division/Muon *g-2* experiment
CHEP October 2013



** Currently Harvey Mudd College Undergraduate Student*



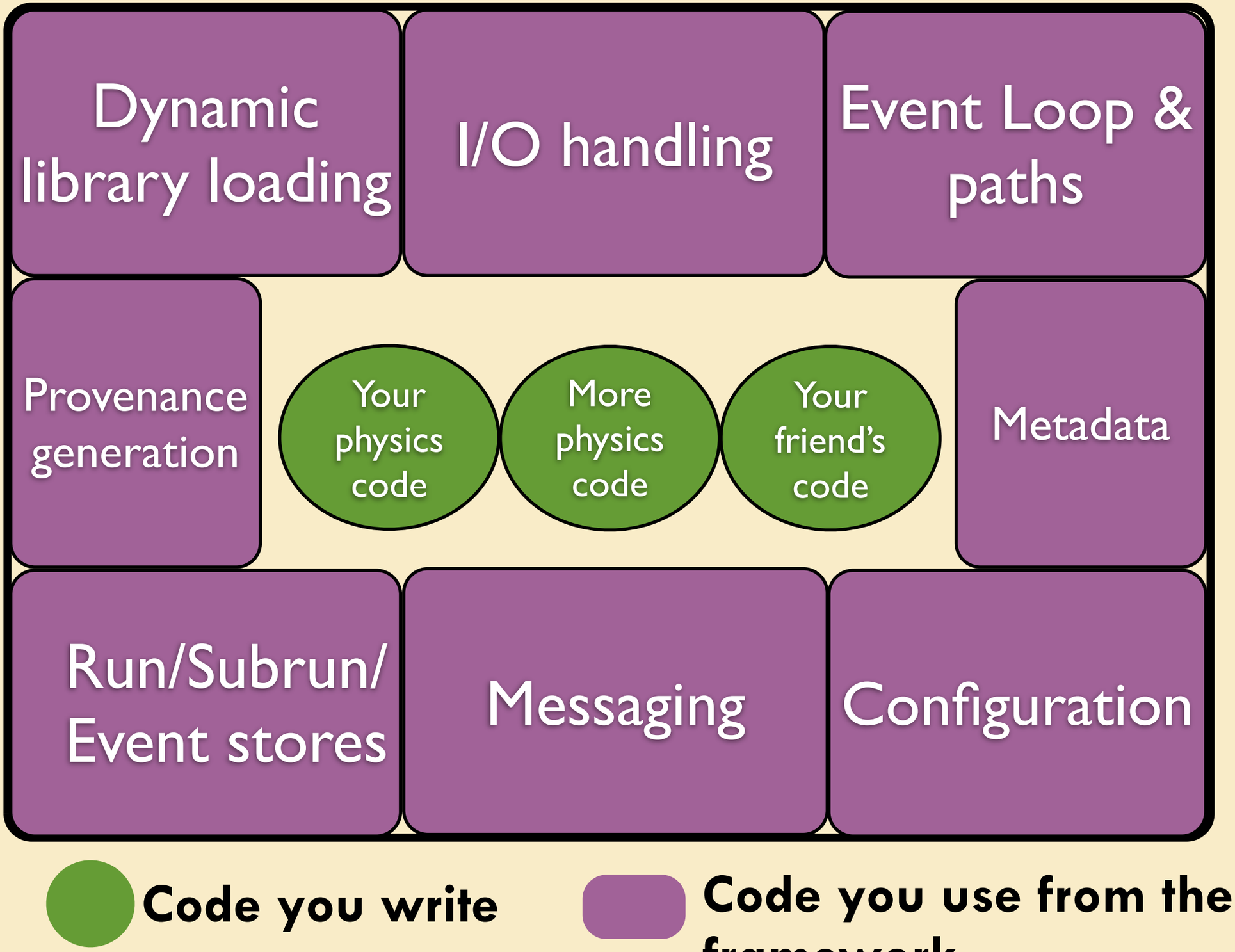
Overview

- **Science demands reproducibility.**
  We must have control over our software

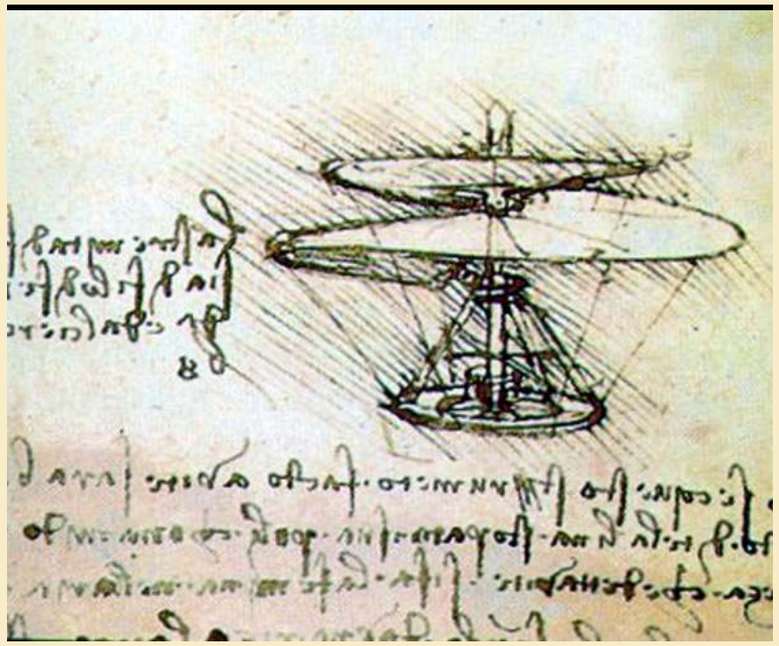
- **We want to work together.**
  Share ideas through code

- **We want to do physics, not computing.**
  We just wanna make plots! Somehow, that should be easy and sane

What does a framework do?



Fermilab's common framework from the Scientific Computing Division



ART

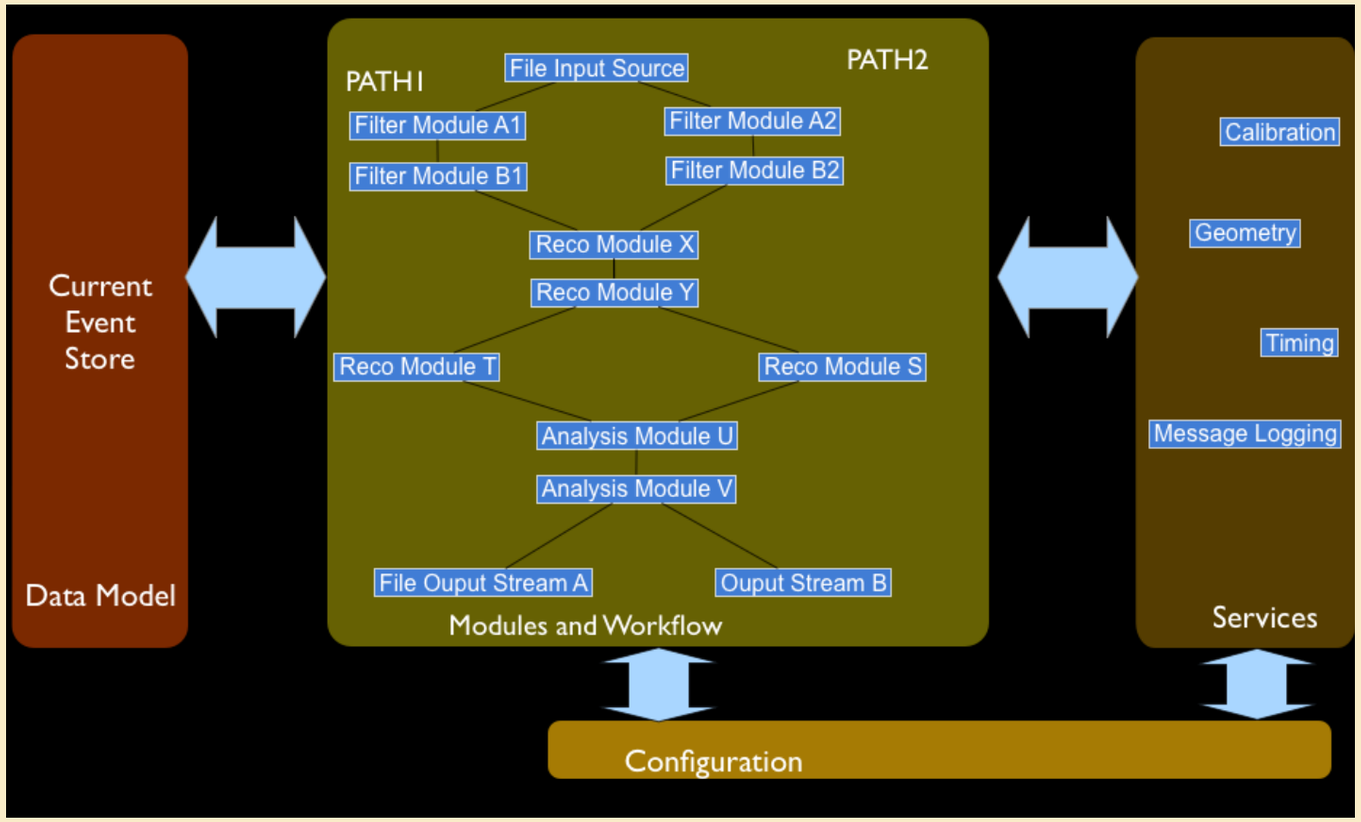
A "lite" forked version of the CMS framework

Supplies all expected framework services as well as links between data objects (Ptr's and Assn's)

Used by many Fermilab Intensity Frontier Experiments (NOvA, Muon g-2, *Mu2e*, MicroBoone, LBNE) and some others (e.g. DS50)

Written by SCD/CET department

Currently being adapted for multi-processing and DAQ



Our first task

Convert our simulation code to Art

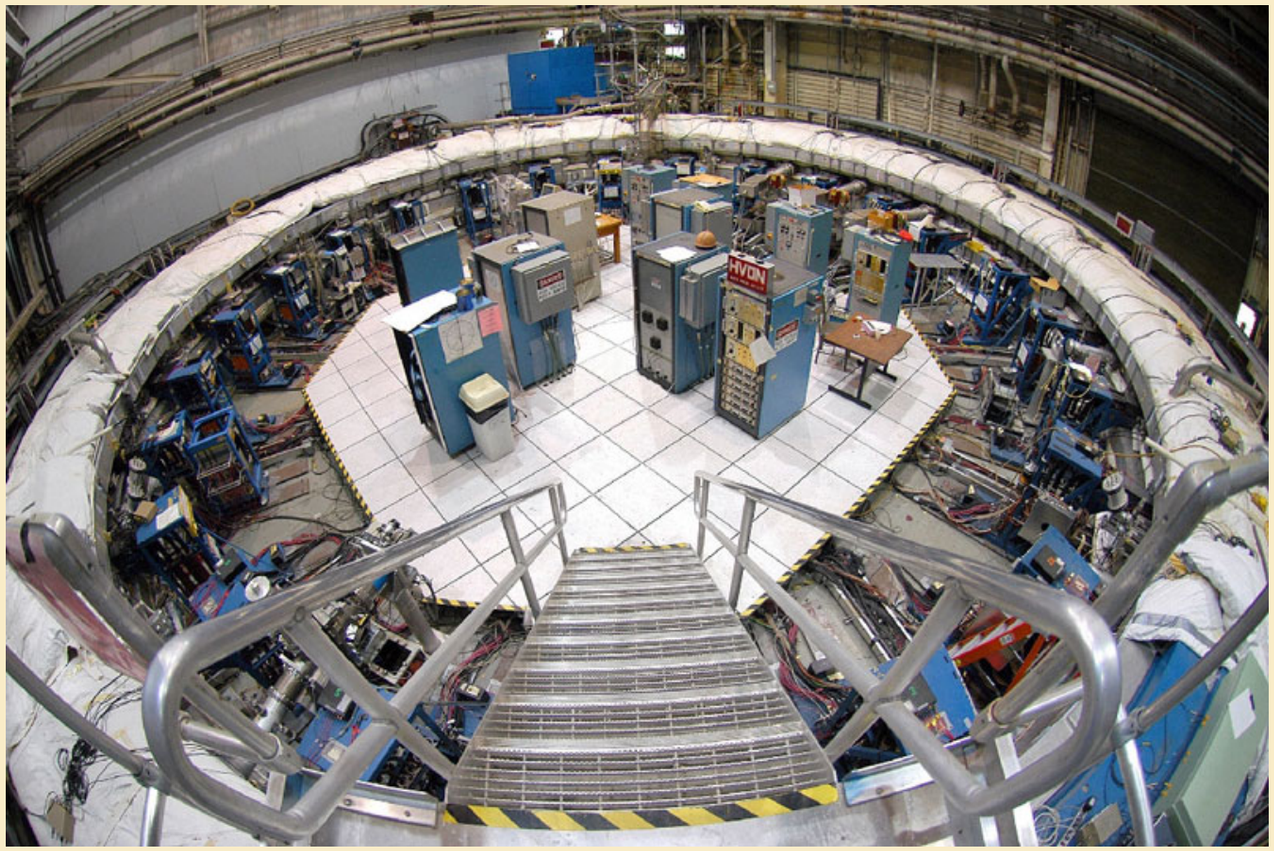
So first, let me tell you about the simulation code;

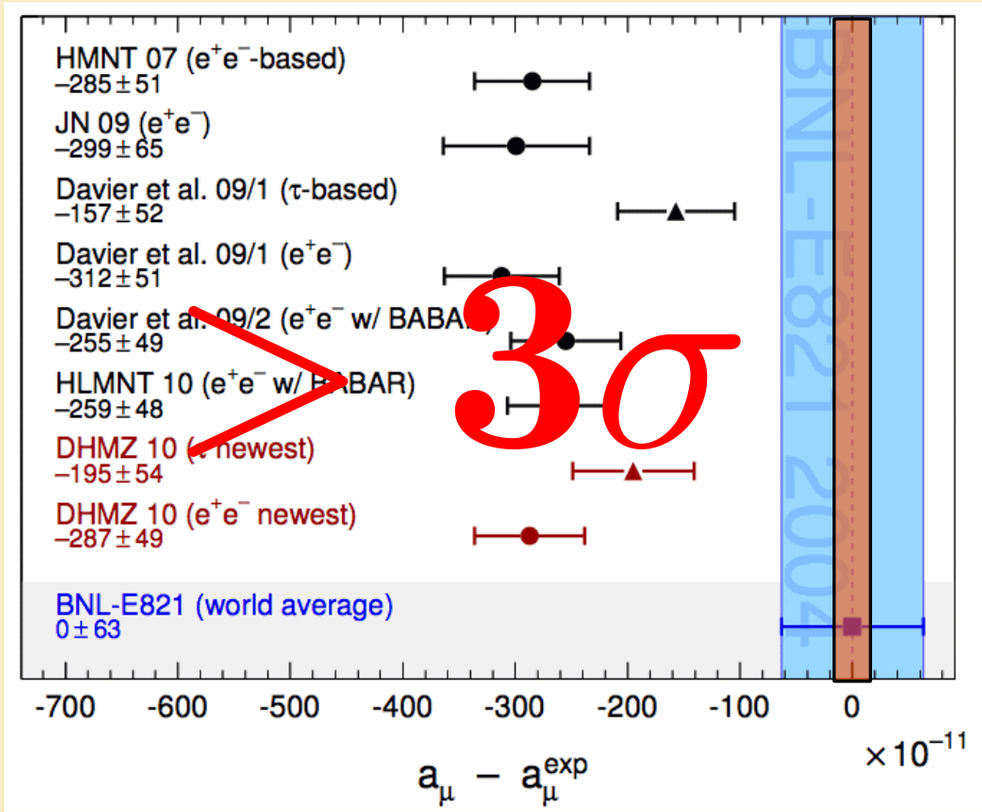
But more first, a little about Muon *g*-2 ...

Muon g-2 in one slide

- o Muon spin precesses in a magnetic field
- o *g-2* characterizes the precession rate
- o All known particles contribute to the Standard Model prediction of *g-2*
- o Measure a difference from the SM — indication of new particles!

E821 Muon *g-2* ring at Brookhaven National Laboratory, now at Fermilab





- o 0.54 ppm result at Brookhaven (1999-2001) Hint of new physics!
- o Redo at Fermilab with 0.14 ppm precision for definitive result

The *g2migtrace* simulation

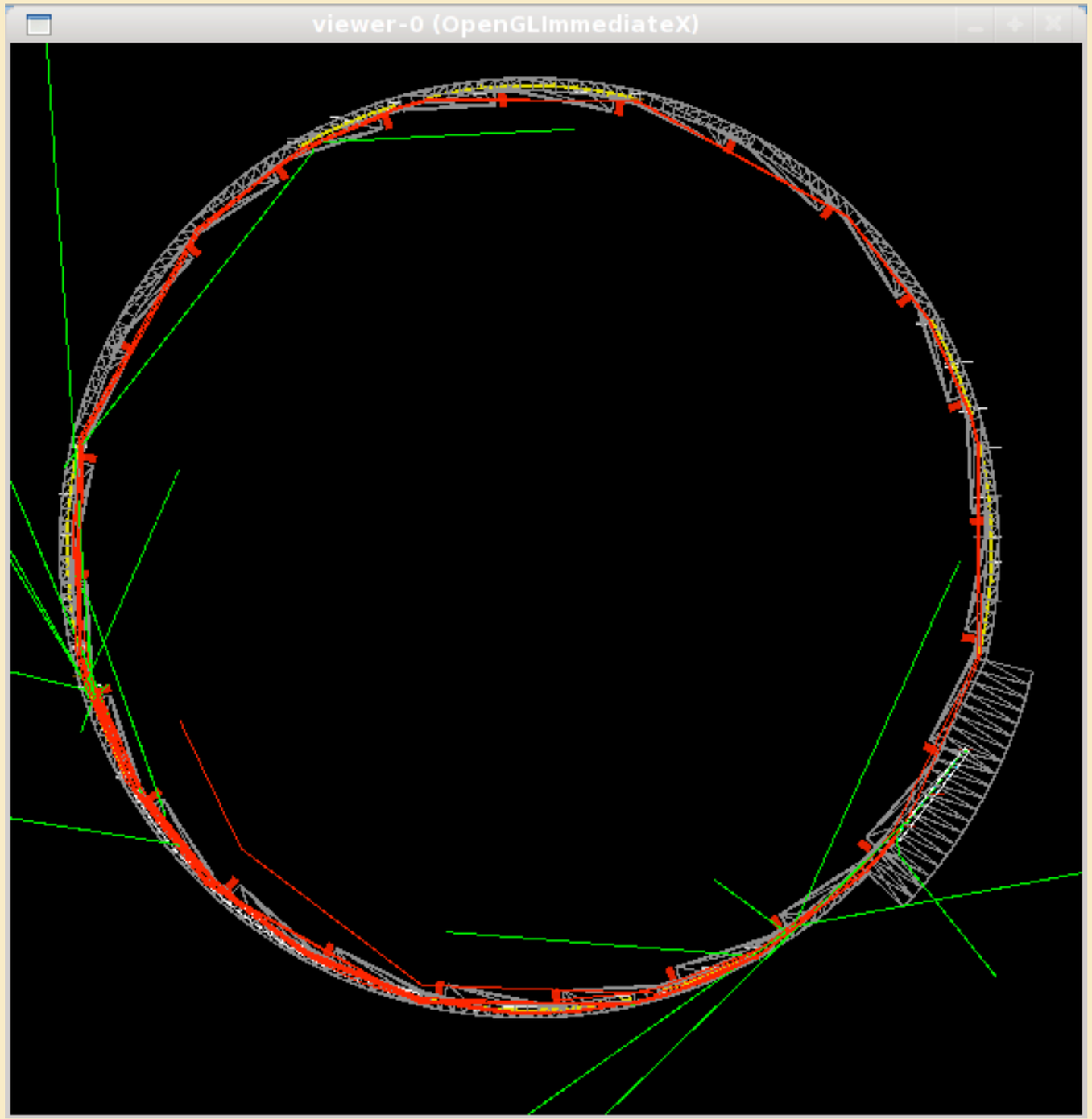
Muon Injection Geometry TRacking And Capture Efficiency

A very detailed Geant4 simulation of the entire *g-2* storage ring elements, magnetic fields, and detectors

Simulates the muon injection sequence (from inflector, to kickers, to scraping, to storing)

Converts Geant hit objects to objects in ROOT branches

Started in 2005 by
Kevin Lynch (York College, CUNY/g-2 and Mu2e)
Zach Hartwig (MIT/Fusion)



It contains incredibly valuable code

Geometry is mostly hard-coded with some JSON files

Interaction via Geant's messenger facility and command prompt

Extremely detailed simulation – would not want to rewrite

Valuable notes and comments

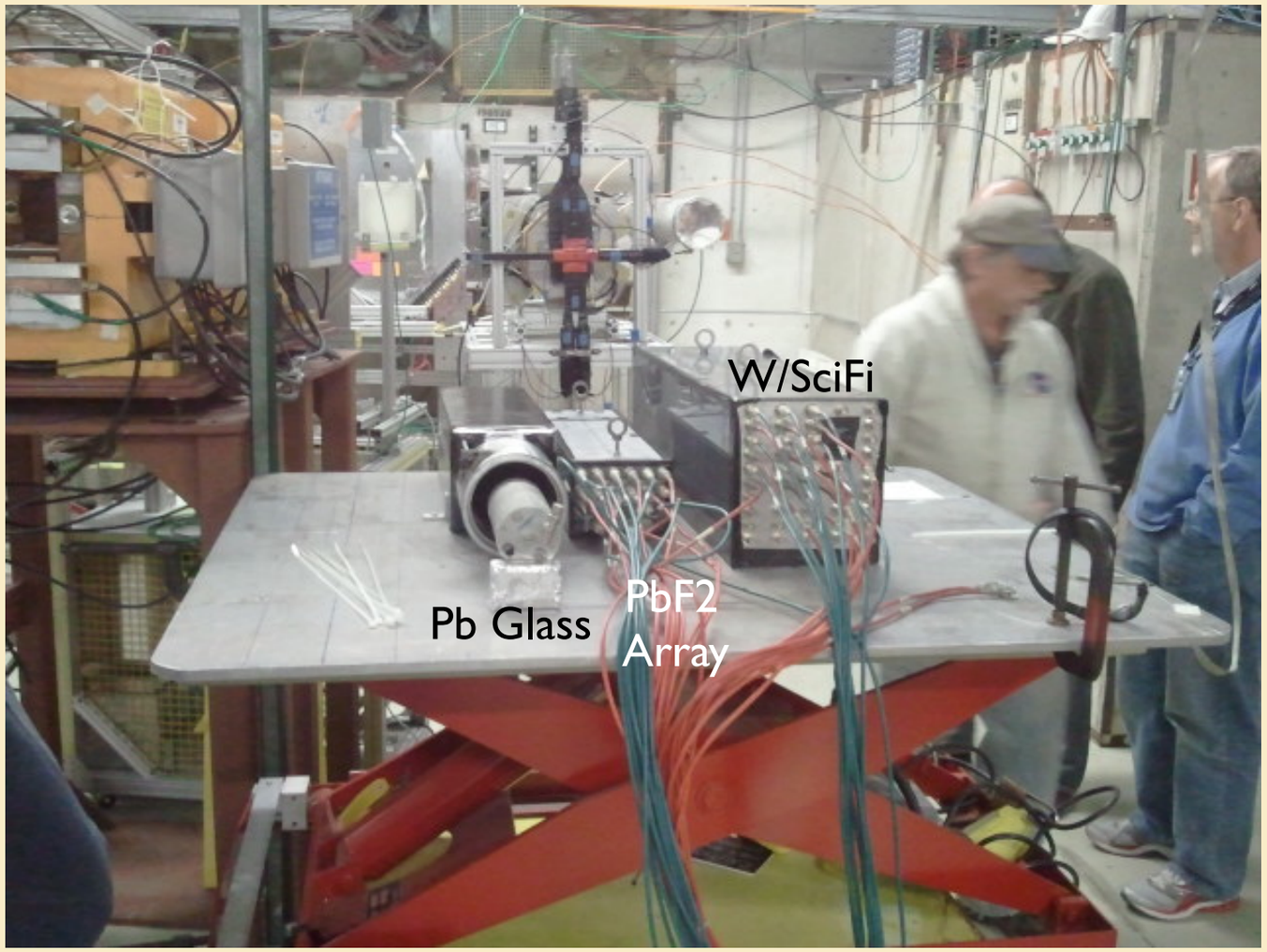
BUT - is a monolithic program.

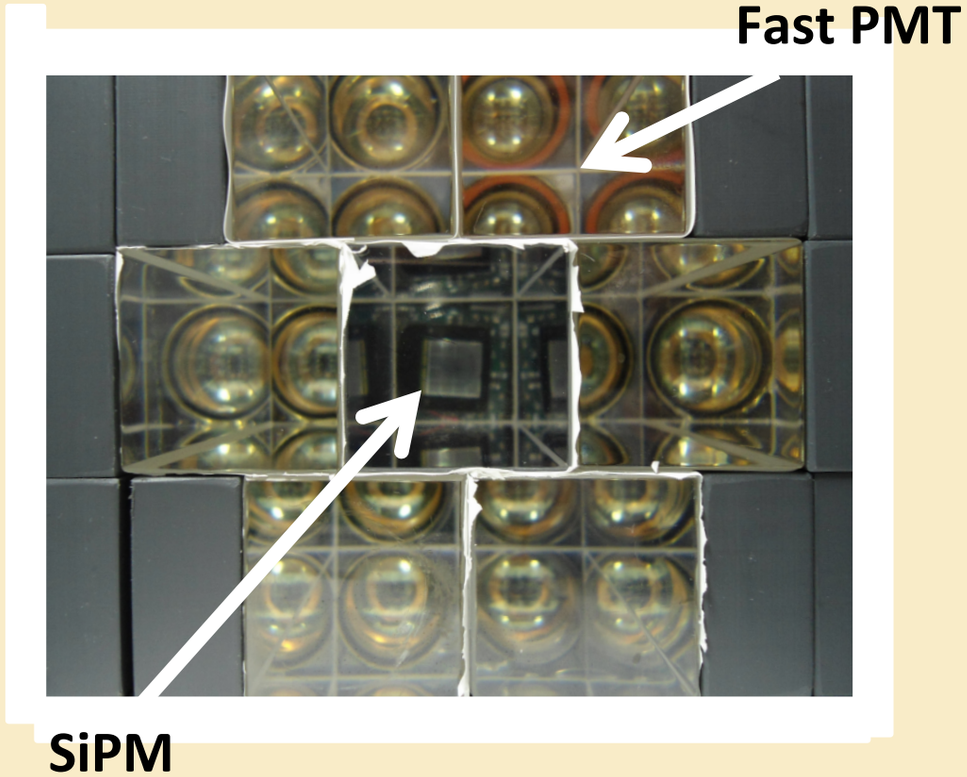
```
    To calculate the quadrupole E fields from the polar maps produced
    below, we find it necessary to convert (x,y,z)_world into
    (r_q,th_q)_quadrupole (See below coordates system).  Then do the
    interpolation on the polar grid, convert (E_rq,E_thq) into
    (E_xq,E_yq), and then finally convert (E_xq,E_yz) into
    (E_x,E_y,E_z) in world coordinates!

    To avoid confusion, some important definitions:
    a) subscript _q indicates value is in quadrupole coordinates defined below
    b) "r" is radial distance in storage planr from center of ring to particle
    c) "r_q" is distance from center of storage region to particle

    Quadrupole coordinates :

    <em>"Friggin' awesome ASCII art!" raves the New York Times</em>
    <pre>
              +r_q ^
                    \
          +th_q  ____>\
                /      \   ^ +y_q
               /       _\\__|____
              |       |  \ |    |
              |       |   \|    |
            +x_q <--------|----0    |    +z_q is into the emacs/downstream (ha! funny!)
                          |         |
                          |_________|
    </pre>
    Note: Coordinate systems should always be right handed!  Hence,
    +x_q points to the left so that +z_q points in the downstream
    direction.

    @author Zach Hartwig
    @author Kevin Lynch
    @date 2005-2009
*/


#include "G4UnitsTable.hh"
#include "G4RunManager.hh"
#include "G4Track.hh"
```

Hard to integrate new ideas without lots of switches and *if* statements

And wait till you see this...

Test beam in April 2012





Testing calorimeters & readout at the Fermilab Test Beam Facility

Needed a simulation. g2migtrace already has calorimeters, so...

In g2migtrace/src/primaryConstruction.cc

```
// constructionMaterials is essentially a "materials library" class.
// Passing to to construction functions allows access to all materials

  /**** BEGIN CONSTRUCTION PROCESS  ****/

  // Construct the world volume
  labPTR = lab -> ConstructLab();
  // Construct the "holders" of the actual physical objects
#ifdef TESTBEAM
  ArcH.push_back(labPTR);
#else
  ArcH = arc->ConstructArcs(labPTR);
#endif
  // Build the calorimeters
  //  cal -> ConstructCalorimeters(ArcH);
   station->ConstructStations(ArcH);
#ifndef TESTBEAM
  // Build the physical vacuum chambers and the vacuum itself
  VacH = vC -> ConstructVacChamber(ArcH);
```

In g2migtrace/src/primaryConstruction.cc

```
// constructionMaterials is essentially a "materials library" class.
// Passing to to construction functions allows access to all materials

  /**** BEGIN CONSTRUCTION PROCESS  ****/

  // Construct the world volume
  labPTR = lab -> ConstructLab();
  // Construct the "holders" of the actual physical objects
#ifdef TESTBEAM
  ArcH.push_back(labPTR);
#else
  ArcH = arc->ConstructArcs(labPTR);
#endif
  // Build the calorimeters
  //  cal -> ConstructCalorimeters(ArcH);
   station->ConstructStations(ArcH);
#ifndef TESTBEAM
  // Build the physical vacuum chambers and the vacuum itself
  VacH = vC -> ConstructVacChamber(ArcH);
```

I don't think we can't simultaneously maintain this code and our sanity

In g2migtrace/src/primaryConstruction.cc

```
// constructionMaterials is essentially a "materials library" class.
// Passing to to construction functions allows access to all materials

  /**** BEGIN CONSTRUCTION PROCESS  ****/

  // Construct the world volume
  labPTR = lab -> ConstructLab();
  // Construct the "holders" of the actual physical objects
#ifdef TESTBEAM
  ArcH.push_back(labPTR);
#else
  ArcH = arc->ConstructArcs(labPTR);
#endif
  // Build the calorimeters
  //  cal -> ConstructCalorimeters(ArcH);
   station->ConstructStations(ArcH);
#ifndef TESTBEAM
  // Build the physical vacuum chambers and the vacuum itself
  VacH = vC -> ConstructVacChamber(ArcH);
```

WHAT IF WE HAVE A DIFFERENT TEST BEAM?

WHAT IF I WANT A DIFFERENT DETECTOR CONFIGURATION?

THIS KIND OF CODE IS HARD TO EXCISE LATER

I don't think we can't simultaneously maintain this code and our sanity

Maintaining sanity is hard

It's hard to blame the person who did this

He just wanted results!

We don't have a system that tries to make this easy

It's not the system's fault - it wasn't written for that

Writing such a system is hard (need experts)

Learning such a system is non-trivial too

Use a system that makes this easy

Want a system that makes it easy to work together

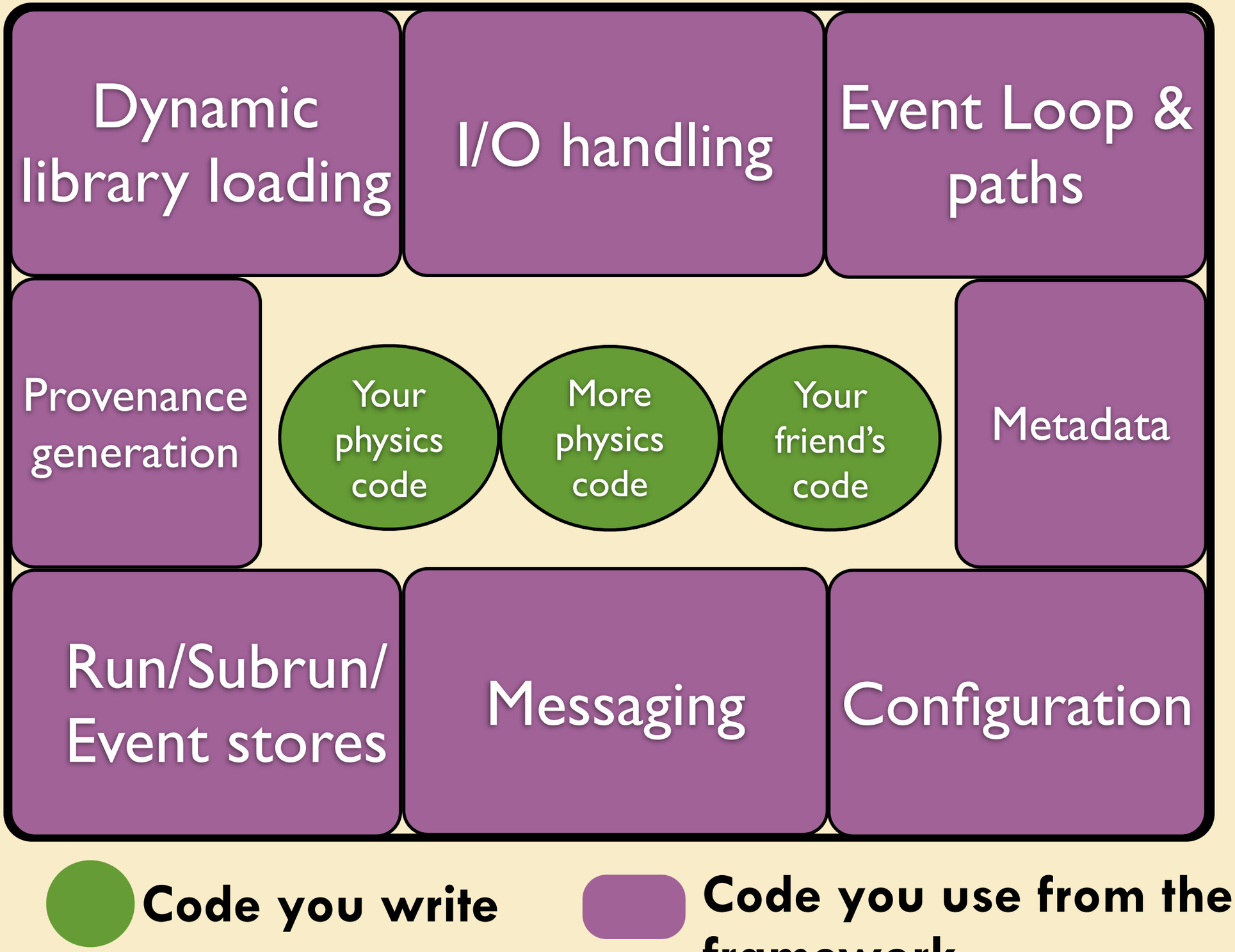
ART

Modular (you write modules that piece together)
Built in Root i/o
Built in Configuration System
C++11

The idea:
Using ART, build a modular Geant4 system where the configuration file defines the simulation

Here's a little bit about ART (<u>not</u> a full tutorial)...

What does a framework do?



What do you write?

You write modules that can access data and do things at certain times

<u>Types of MODULES:</u>
(All modules can read data from the event)

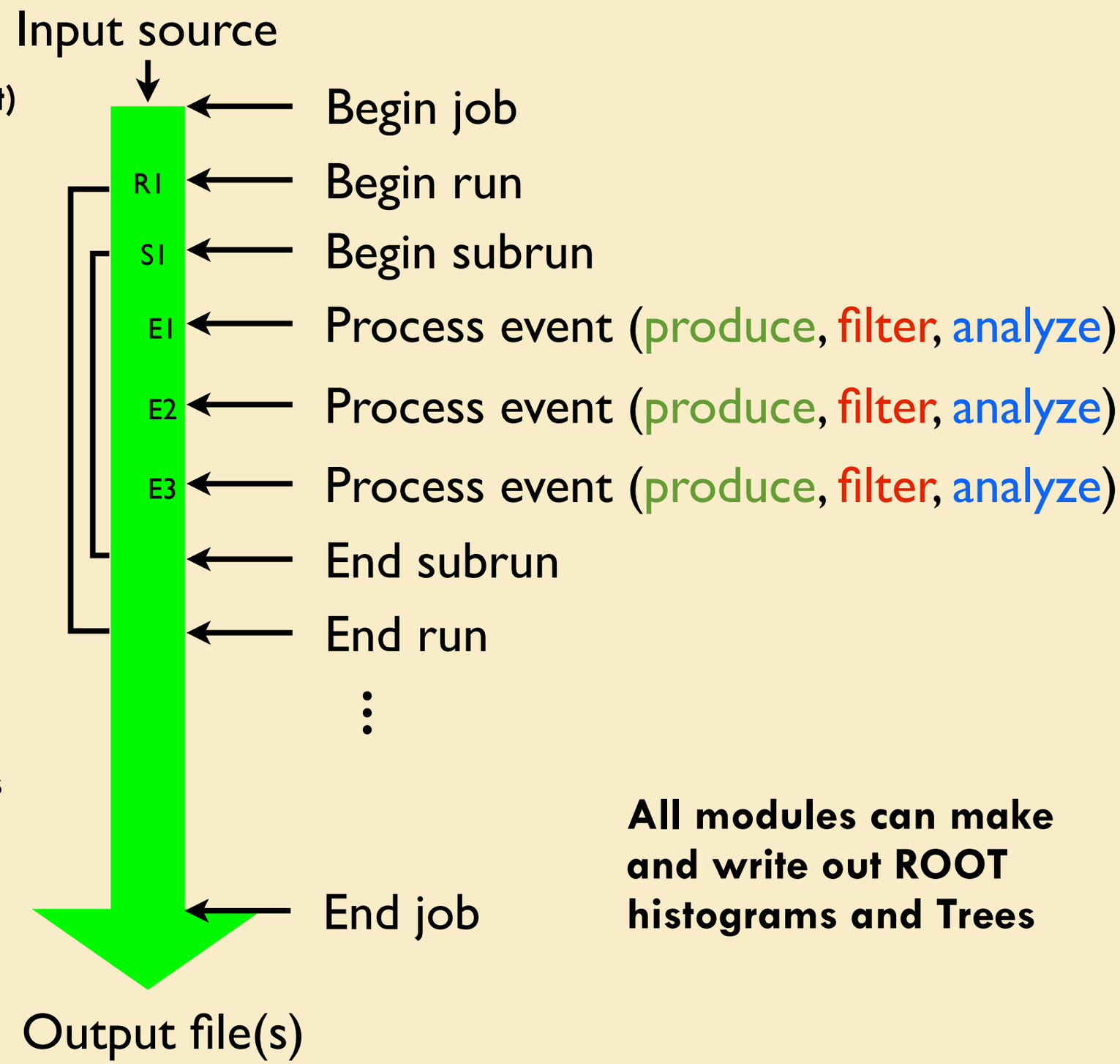
o Input source:
A source for data. E.g. a ROOT file or Empty for start of simulated data

o Producers:
Create new event data from scratch or by running algorithms on existing data

o Filters:
Like producers, but can stop running of downstream modules

o Analyzers:
Cannot save to event. For, e.g. diagnostics plots

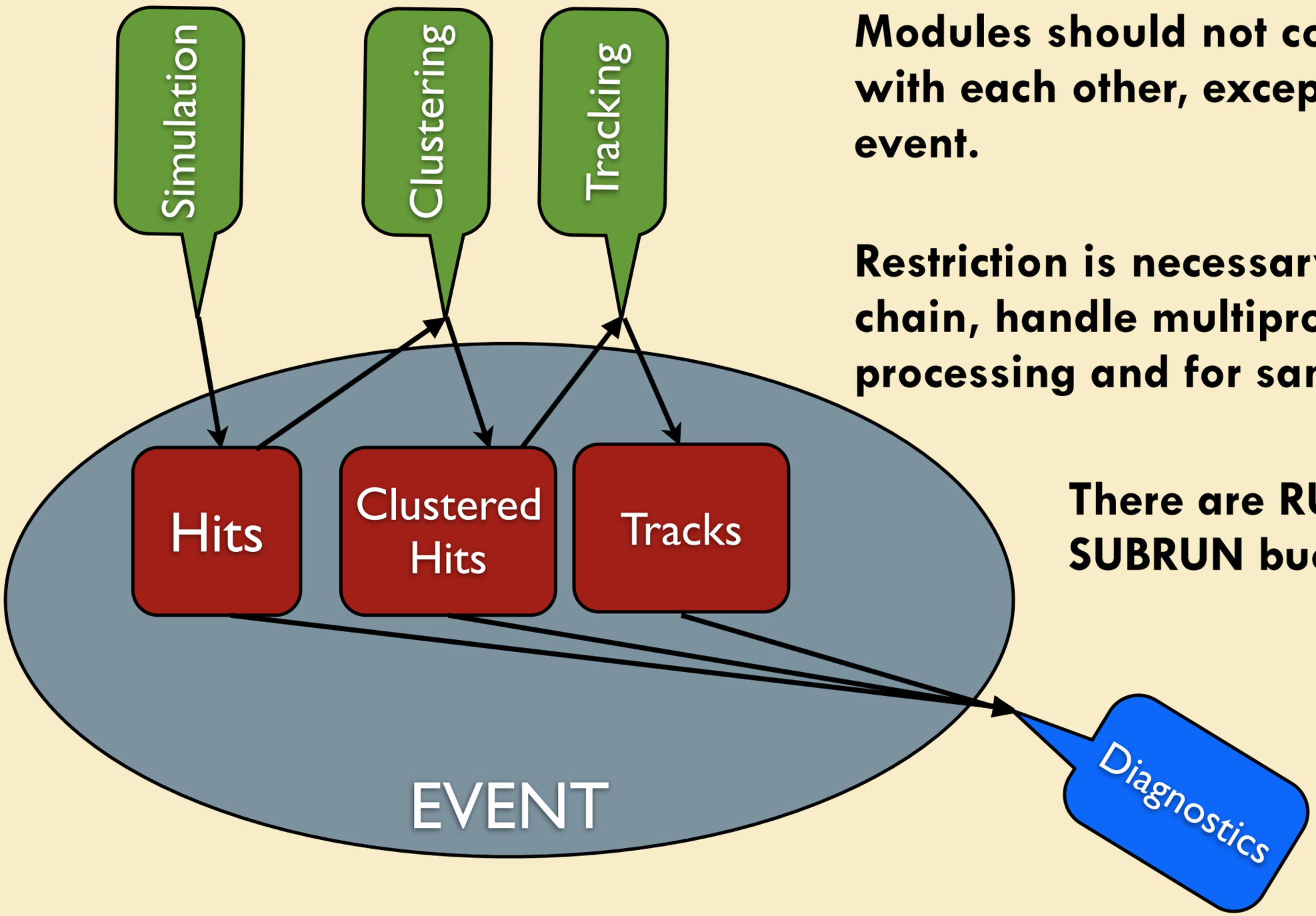
o Output module:
Writes data to output file (ROOT). Can specify conditions and have many files



All modules can make and write out ROOT histograms and Trees

Chain modules - but an important golden gule

Modules must only pass data to each other via the EVENT



Modules should not communicate with each other, except through the event.

Restriction is necessary to break chain, handle multiprocessor processing and for sanity.

There are RUN and SUBRUN buckets too

Services – an extremely useful feature

Globally accessible objects can be managed by ART as Services

Provide functionality to many modules (same object is accessible to all modules)

Examples:

Message facility, timers, memory checkers, Random numbers, Geometry information

Since a service is an ordinary C++ object, it can hold data and state

BUT - Remember the golden rule! Event information goes into the EVENT, not a service

<u>Easy to create:</u>

Your class .cc file simply needs

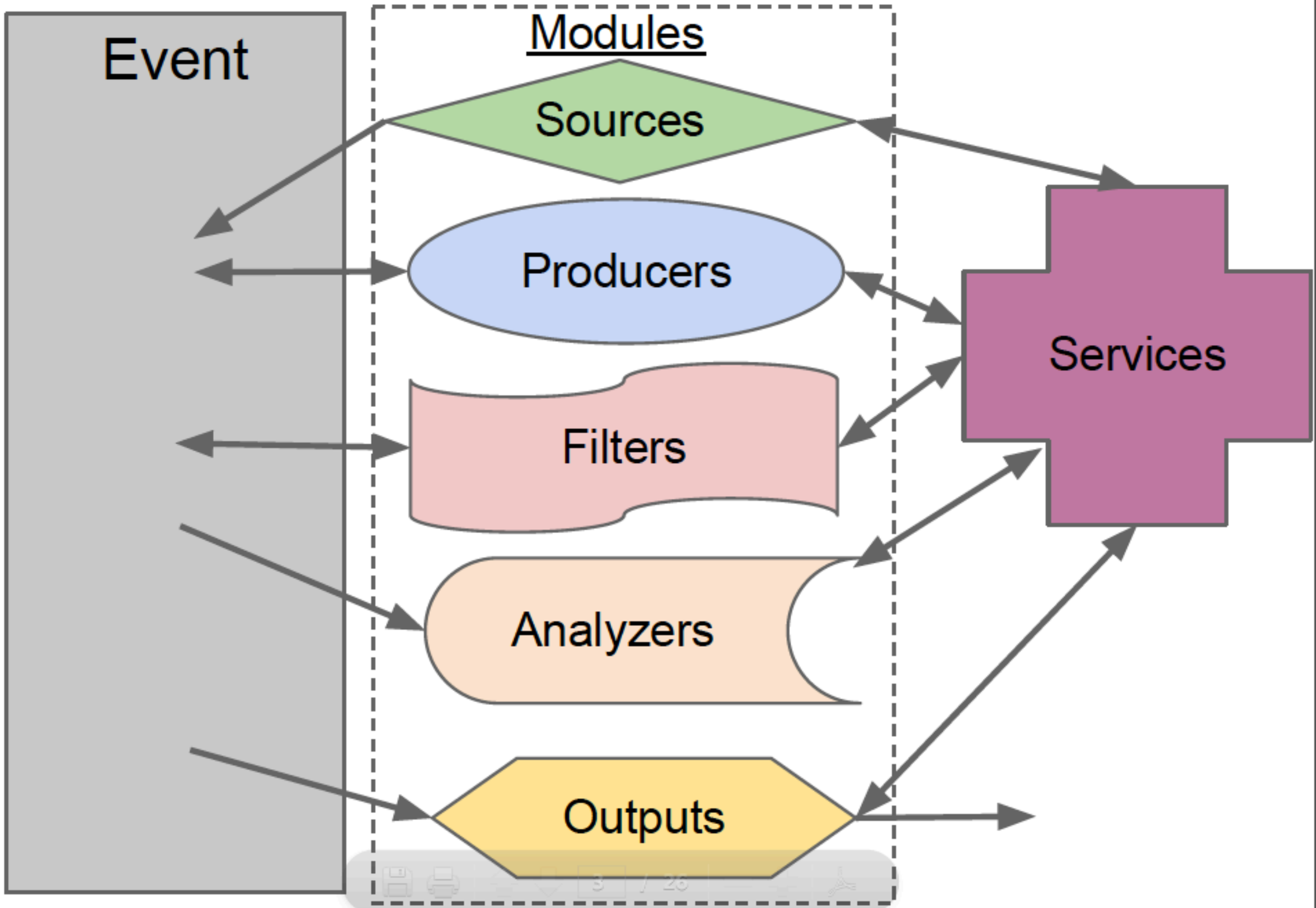
```
#include "art/Framework/Services/Registry/ServiceMacros.h"

// Ordinary class implementation goes here

using artg4example::PhysicsListService;
DEFINE_ART_SERVICE(PhysicsListService)
```

<u>Easy to use:</u>

The handle acts just like a pointer to the object

```
#include "artg4/services/PhysicsListHolder_service.hh"

// ...

void artg4::artg4Main::beginRun(art::Run & r)
{
  // Get the physics list and pass it to Geant and initialize the list if necessary
  art::ServiceHandle<PhysicsListHolderService> physicsListHolder;
  runManager_->SetUserInitialization( physicsListHolder->makePhysicsList() );
  physicsListHolder->initializePhysicsList();

// ...
```

Art Glossary



How to marry ART and Geant4?

GEANT4 is a huge library for detailed simulations of particles traversing materials

GEANT4 Basic pieces:

- **Detectors:**
  - **Geometry, materials, hierarchy**
    Shapes, G4LogicalVolume, G4VPhysicalVolume
  - **Sensitive detectors make hits**

- **Actions (Code hooks to run my code at certain points in the simulation):**
  - **Begin/end run and event**
  - **Generating first particles**
  - **Upon a new trajectory**
  - **On each simulation step**

- **Other stuff:**
  - **Physics lists (specify allowable particles and how they behave)**

Adapting g2MIGTRACE to ART

Preserve **the valuable parts**
- **detector and magnetic fields construction**
- **coordinate system**
- **algorithms for simulation (Sensitive detectors)**
- **Want to cut and paste as much Geant code as possible**

Reorganize the code to fit with ART

<u>Requirements:</u>
- **Modularity: Detectors and Actions are "plug and play"**
- **Configuration: Simulation is defined by config file**
  - **Can make changes without recompiling**
- **Store Geant "products" to ART event**
- **Of course old & new output must be identical**

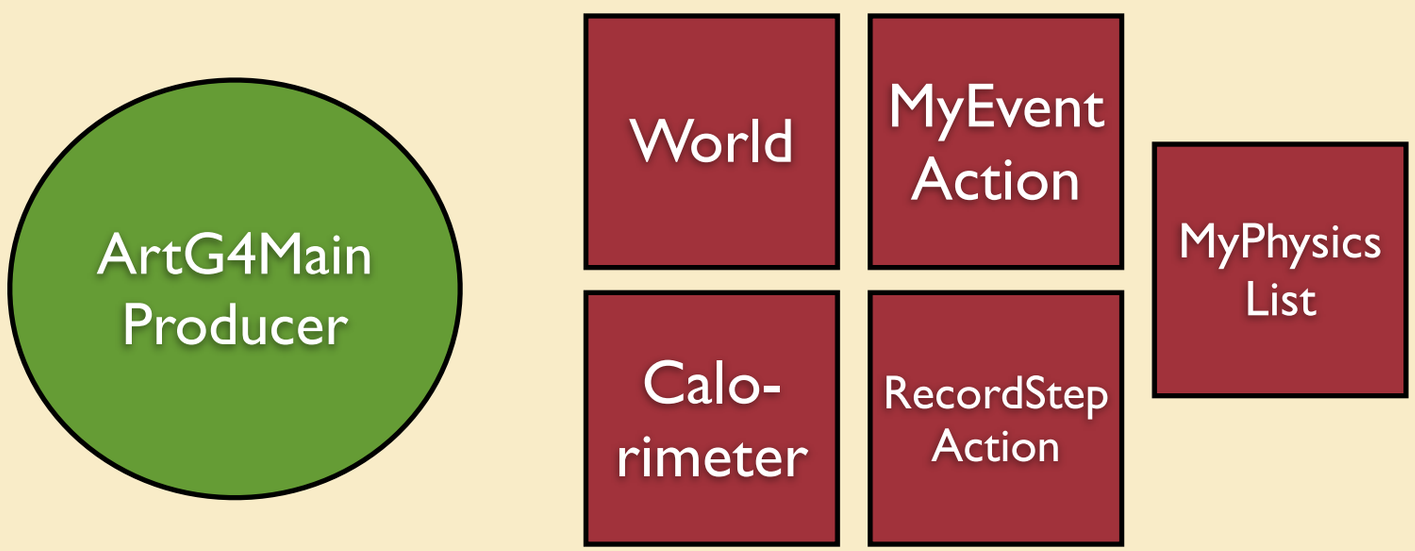
Allow us to easily work together using the ART framework

A model using SERVICES works!

One producer that handles Geant: ArtG4Main

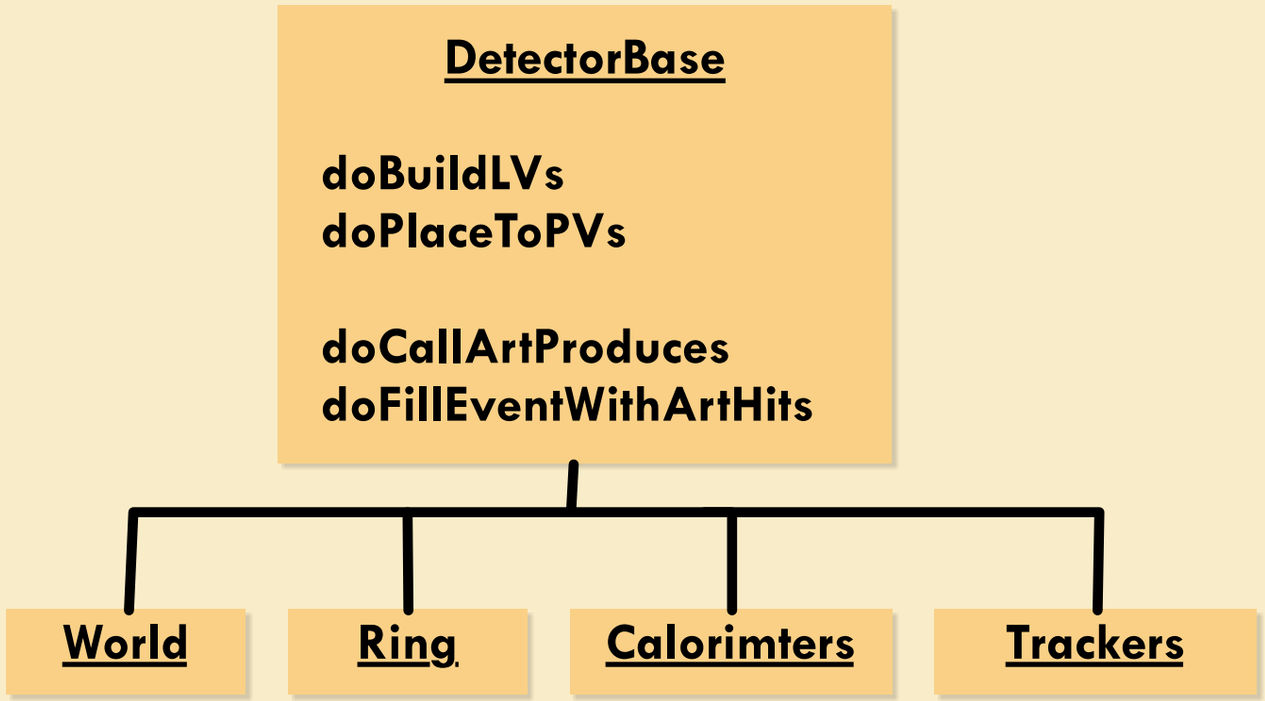
To make it generic, ArtG4Main <u>delegates</u> lots of responsibilities to SERVICES that are <u>ONLY</u> used by ArtG4Main. Since only one producer for Geant, we satisfy the golden rule.

The configuration files says what Services to load



Detector Services

- Must load DetectorHolder_service – manages detectors and does registration behind the scenes
- Every detector must have name, category, mother category



- This organization seems simple, but is quite powerful
- DetectorHolder service makes this truly modular
- Framework knows nothing about detectors (don't need to bake them in)
- This is a re-organization – most code (the hard stuff) remains untouched

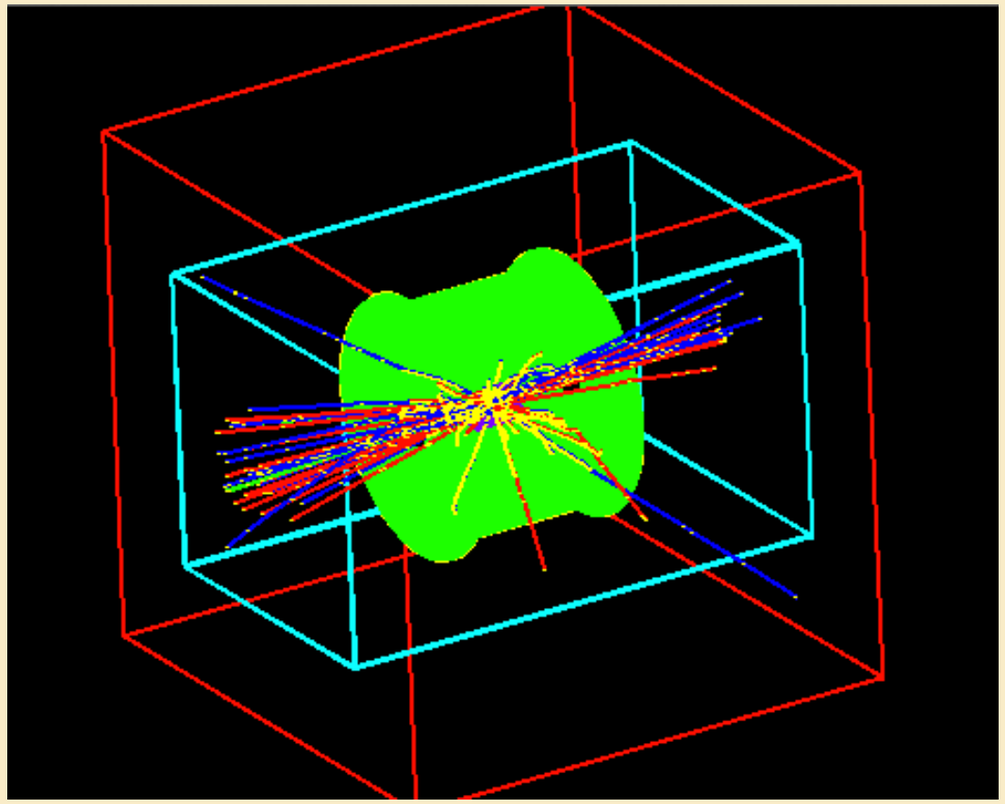
Example Configuration File

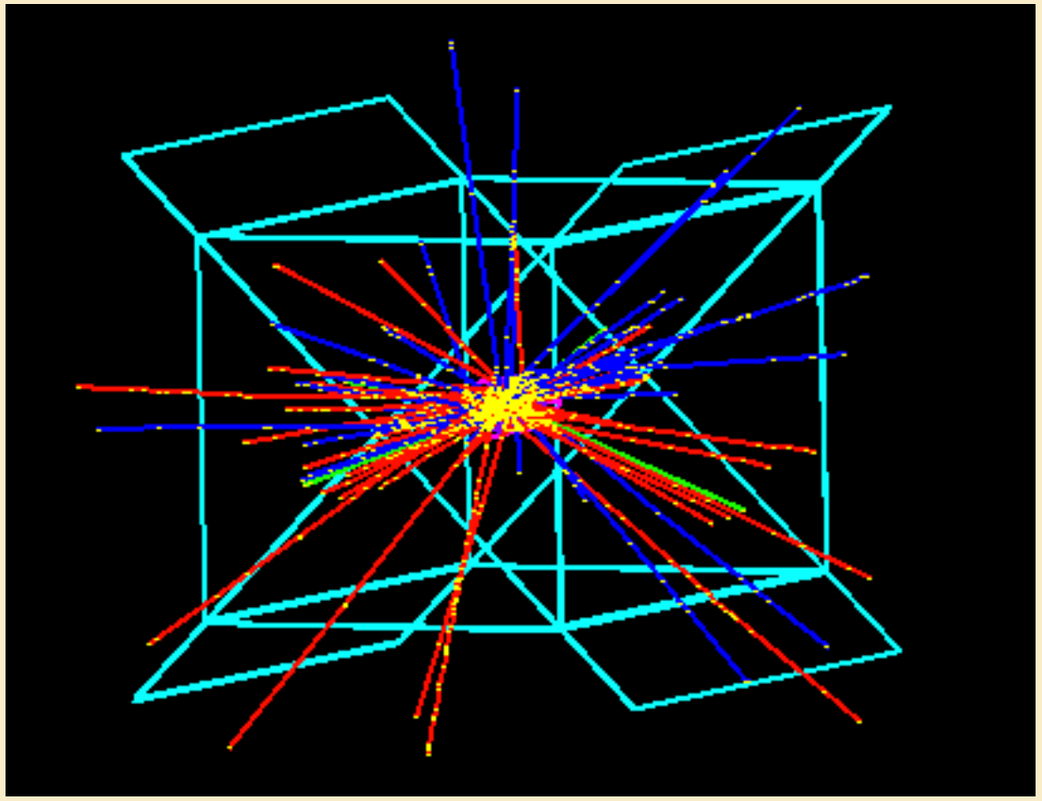
```
#include "detectorDefaults.fcl"

process_name:processA

source: {
  module_type: EmptyEvent
  maxEvents:  3
}

services: {

  user: {
    DetectorHolder: {}
    ActionHolder: {}
    RandomNumberGenerator: {}
    PhysicsListHolder: {}

    PhysicsList: {}

    // Detector(s) for the simulation
    ExampleWorldDetector:   @local::ExampleWorldDetectorDefaults

    ExampleTrackerDetector: @local::ExampleTrackerDetectorDefaults

    ExampleMuonDetector: {
      name: "exampleMuon"
      category: "exampleMuon"
      mother_category: "world"
      box_radius: 350
      width: 345
      thick: 1
      length: 590
      n_muon_counters: 4
    }
```

...

```
outputs: {
out1: {
  module_type: RootOutput
  fileName: "out.root"
  }
}

physics: {
  producers: {
    artg4Main: {
      module_type: artg4Main
      enableVisualization: true
      macroPath: ".:./macros"
      visMacro: "vis.mac"
      afterEvent: pause
    }

  }

  path1: [ artg4Main ]
  stream1:  [ out1 ]

  trigger_paths: [ path1 ]
  end_paths: [ stream1 ]
}
```

N04 Example





From config, don't include calorimeter and make 8 muon planes
[No rebuild necessary]

Action Services

Must load ActionHolder_service – manages actions

There are 6 action base classes

- **EventActionBase: beginOfEventAction, endOfEventAction***
- **RunActionBase: beginOfRunAction, endOfRunAction**
- **PrimaryGeneratorActionBase: generatePrimaries (mandatory)**
- **TrackingActionBase: preUserTrackingAction, postUserTrackingAction**
- **SteppingActionBase: userSteppingAction**
- **StackingActionBase: killNewTrack**

*** There's an internal endOfEventAction that tells the detectors to write out their data to ART**

Actions are useful for diagnostics and truth information. Every action can write out information (callArtProduces, fillEventWithArtStuff, fillRunAtBeginWithArtStuff, fillRunAtEndWithArtStuff)

Can combine actions into one object with multiple inheritance

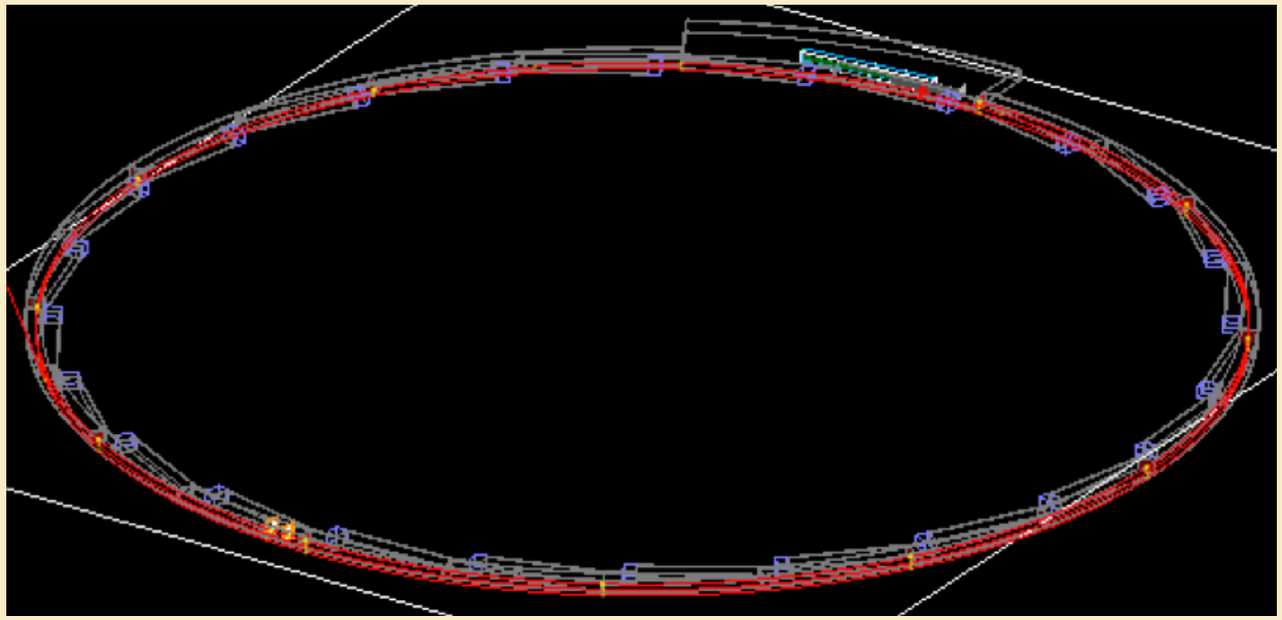
Examples: TrackingTruth, GDMLGenerator, KillCrystalTracks, MuonStorageStatus

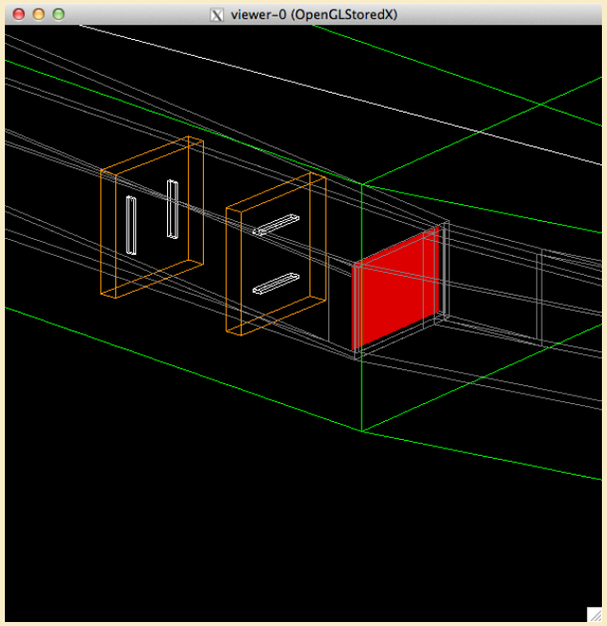
gm2ringsim

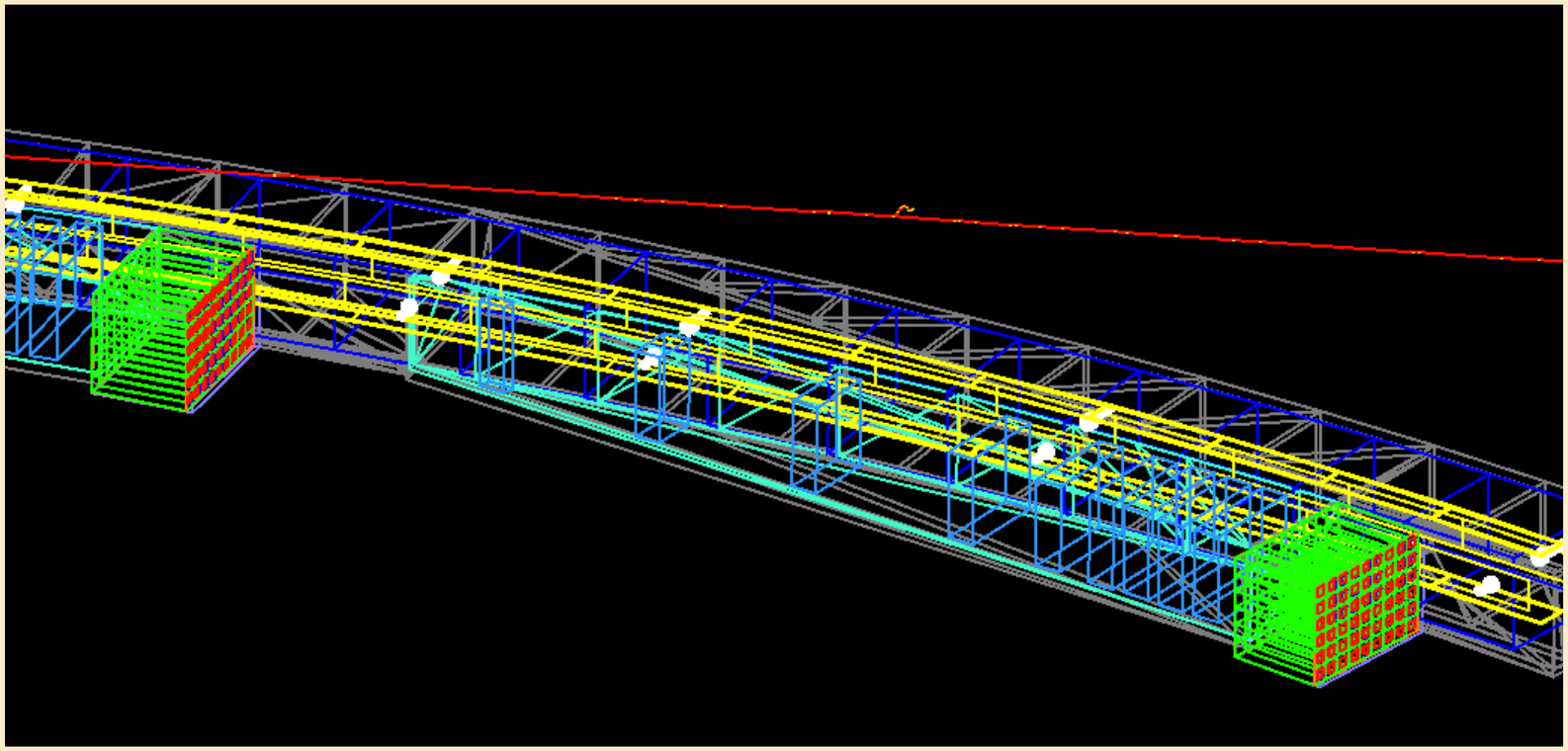
Started 10/12

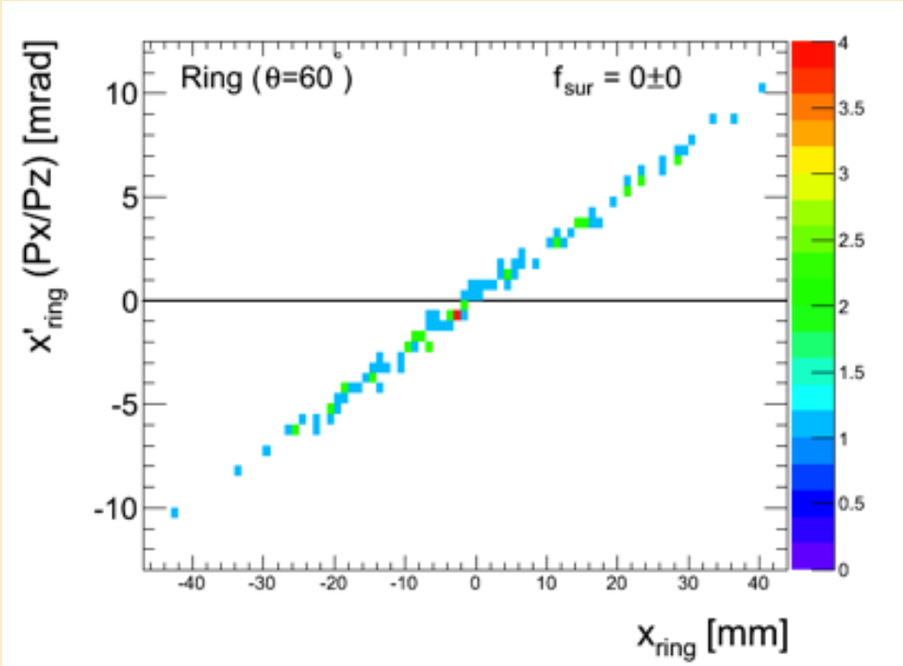
2.5 months with 5 active people

Now have many more analyzing



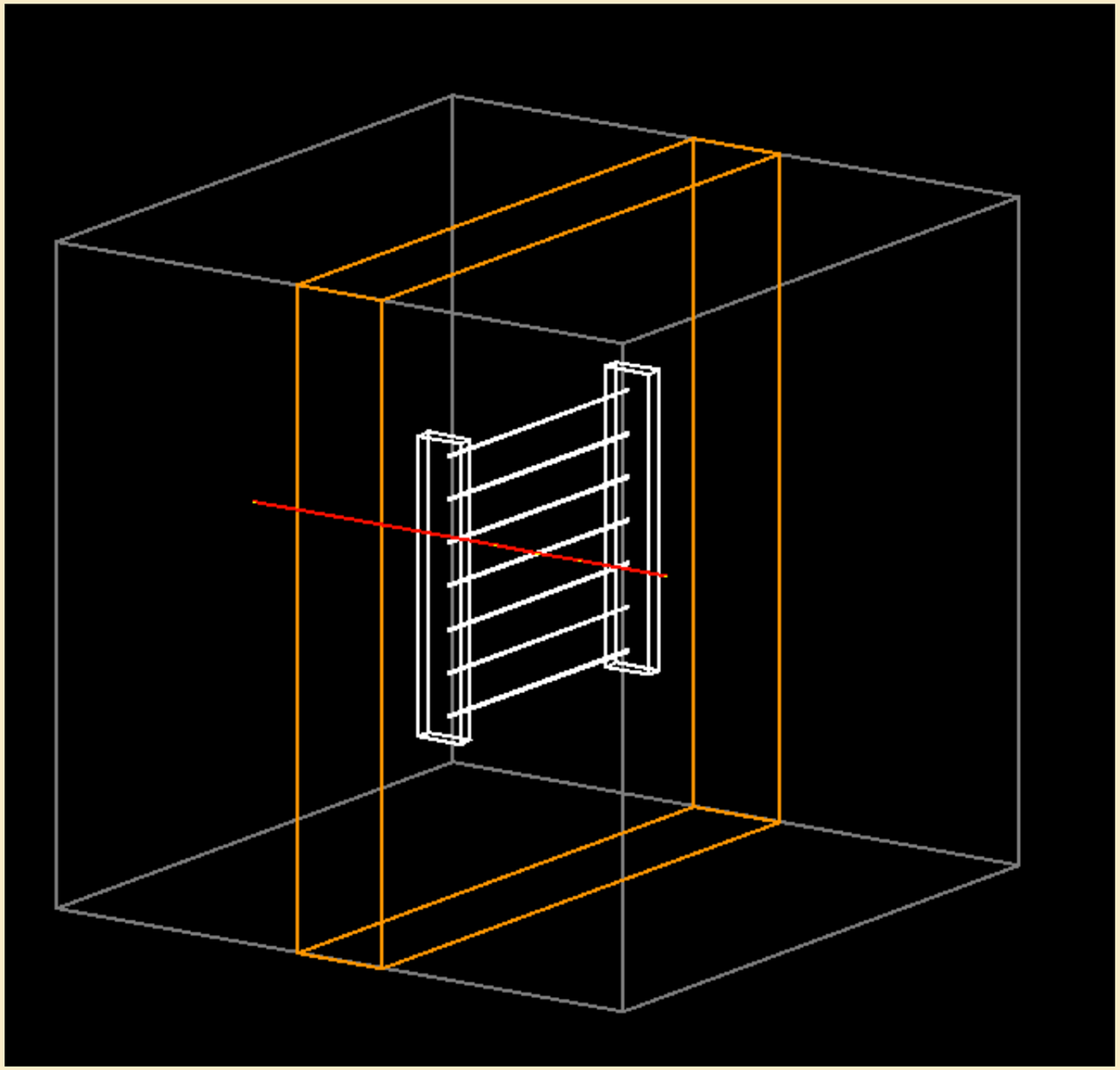






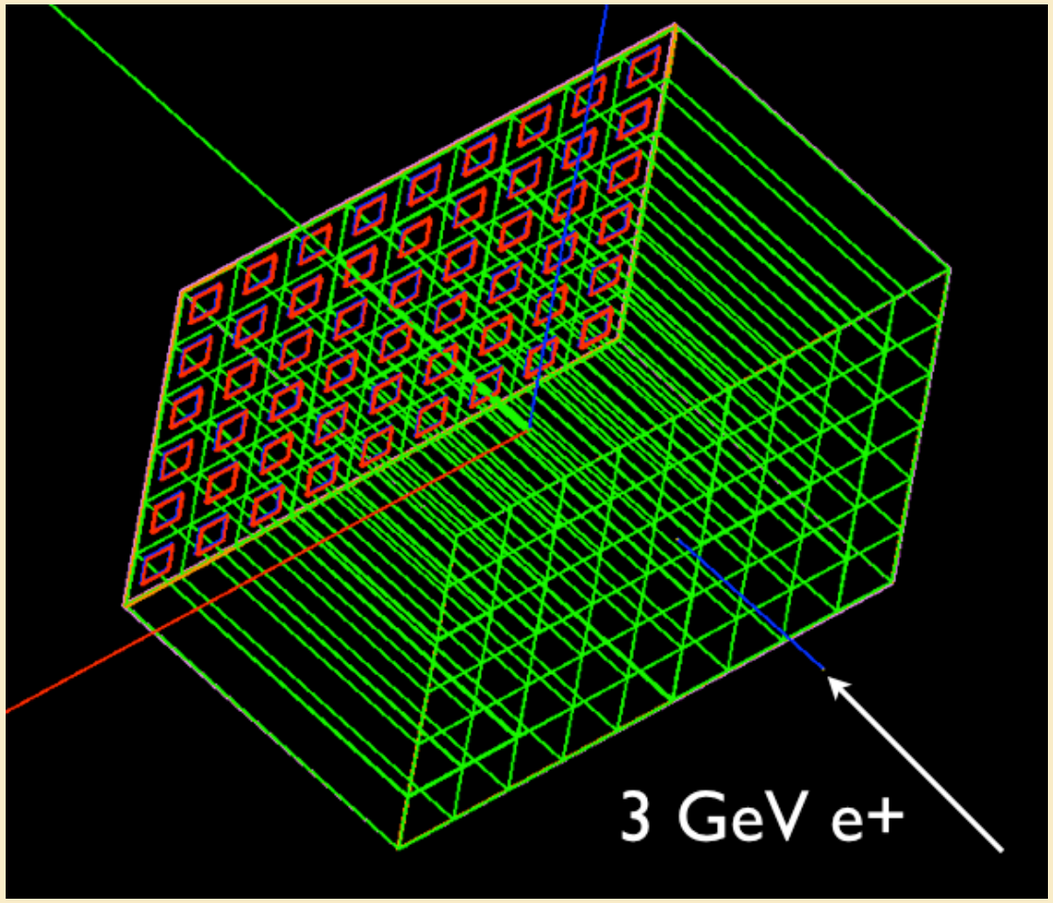
A “Test Beam” Simulation

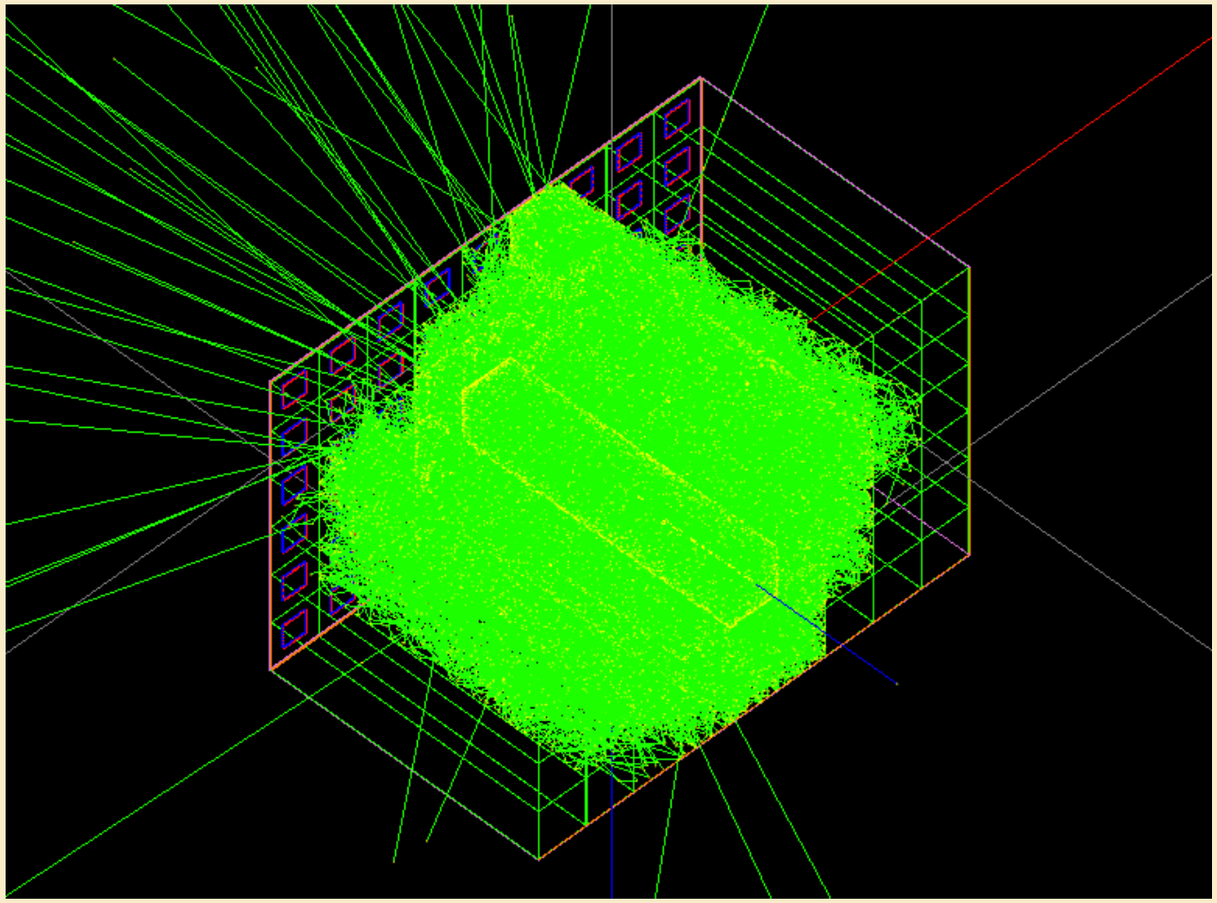
```
user : {

  // Mandatory ArtG4 services
  DetectorHolder: {}
  ActionHolder: {}
  PhysicsListHolder: {}
  RandomNumberGenerator: {}

  // Geometry
  Geometry: {
    world: @local::world_geom
    fiberHarp: @local::fiberHarp_geom
  }

  // Actions
  SimpleParticleSource: {}
  Gm2PhysicsList: {}
  ClockAction: {}

  // Detectors
  World: {}
  FiberHarp: {}

  } //user
} //services

// Override to make a test beam
services.user.Geometry.world.world_x: 100
services.user.Geometry.world.world_y: 100
services.user.Geometry.world.world_z: 100

services.user.FiberHarp.mother_category : world
services.user.Geometry.fiberHarp.nHarps : 1
services.user.Geometry.fiberHarp.RMagicScale : 0
services.user.Geometry.fiberHarp.harpType : [1]
services.user.Geometry.fiberHarp.vacWallPos: [0]
services.user.Geometry.fiberHarp.azimuthalPos : [0]
```



A fiber harp test WITH NO CODE CHANGES (no #ifdefs)

Calorimeter Test Beam





(no #ifdefs)

Summary

ArtG4 is a generic simulation infrastructure for Geant4 within the ART Framework

All detectors and actions are plug-and-play and the configuration file defines the simulation

Though written with Muon *g-2* in mind, it should be useful for many experiments

We are now in the process of validating gm2ringsim and using it for studies

Where you can learn more (see Repository and Wiki):
https://cdcvs.fnal.gov/redmine/projects/artg4 and
https://cdcvs.fnal.gov/redmine/projects/artg4example and
https://cdcvs.fnal.gov/redmine/projects/artg4geantn02 (see various branches)

Requirements on physics software for physicists – solutions

- Science demands reproducibility.
  Official results come from version controlled software

- We want to work together.
  Code repositories; modular frameworks

- We want to do physics, not computing.
  Infrastructure in a framework + an easy build system

Services must be in your config file

e.g. Gm2PhysicsList_service.cc

Build system creates
artg4example_Gm2PhysicsList_service.so

Specifying Gm2PhysicsList in config will find it in your LD_LIBRARY_PATH

```
services: {

  message : {
    debugModules : ["*"]
    suppressInfo : []

    destinations : {
      LogToConsole : {
        type : "cout"
        threshold : "DEBUG"
      }
      LogToFile : {
        type : "file"
        filename : "gm2ringsim.log"
        append : false
        threshold : "DEBUG"
      }
    }
  }

  user : {

    // Mandatory ArtG4 services
    DetectorHolder: {}
    ActionHolder: {}
    PhysicsListHolder: {}
    RandomNumberGenerator: {}

    Gm2PhysicsList: {}

  }

}

# ...
```

Steal from others?

<u>What did NOvA do?</u> They have a GDML based simulation; incompatible with g2MIGTRACE

<u>What did Mu2e do?</u>
They ported their simulation to ART some time ago. Some very useful routines, but they have "uber" code [classes that know about EVERY aspect of the simulation]. e.g. A zillion #includes

```
/ Mu2e include files
#include "GeometryService/inc/GeometryService.hh"
#include "GeometryService/inc/DetectorSystem.hh"
#include "GeometryService/src/DetectorSystemMaker.hh"
#include "GeometryService/inc/WorldG4.hh"
#include "GeometryService/inc/WorldG4Maker.hh"
#include "Mu2eBuildingGeom/inc/BuildingBasics.hh"
#include "Mu2eBuildingGeom/inc/BuildingBasicsMaker.hh"
#include "Mu2eBuildingGeom/inc/Mu2eBuilding.hh"
#include "Mu2eBuildingGeom/inc/Mu2eBuildingMaker.hh"
#include "ProductionTargetGeom/inc/ProductionTarget.hh"
#include "ProductionTargetGeom/inc/ProductionTargetMaker.hh"
#include "ProductionSolenoidGeom/inc/ProductionSolenoid.hh"
#include "ProductionSolenoidGeom/inc/ProductionSolenoidMaker.hh"
#include "ProductionSolenoidGeom/inc/PSEnclosure.hh"
#include "ProductionSolenoidGeom/inc/PSEnclosureMaker.hh"
#include "ProductionSolenoidGeom/inc/PSVacuum.hh"
#include "ProductionSolenoidGeom/inc/PSVacuumMaker.hh"
#include "ProductionSolenoidGeom/inc/PSShield.hh"
#include "ProductionSolenoidGeom/inc/PSShieldMaker.hh"
#include "ProtonBeamDumpGeom/inc/ProtonBeamDump.hh"
#include "ProtonBeamDumpGeom/inc/ProtonBeamDumpMaker.hh"
#include "TargetGeom/inc/Target.hh"
#include "TargetGeom/inc/TargetMaker.hh"
#include "LTrackerGeom/inc/LTracker.hh"
#include "LTrackerGeom/inc/LTrackerMaker.hh"
#include "TTrackerGeom/inc/TTracker.hh"
#include "TTrackerGeom/inc/TTrackerMaker.hh"
#include "ITrackerGeom/inc/ITracker.hh"
#include "ITrackerGeom/inc/ITrackerMaker.hh"
#include "CalorimeterGeom/inc/Calorimeter.hh"
#include "CalorimeterGeom/inc/DiskCalorimeterMaker.hh"
#include "CalorimeterGeom/inc/DiskCalorimeter.hh"
#include "CalorimeterGeom/inc/VaneCalorimeterMaker.hh"
#include "CalorimeterGeom/inc/VaneCalorimeter.hh"
#include "BFieldGeom/inc/BFieldConfig.hh"
#include "BFieldGeom/inc/BFieldConfigMaker.hh"
#include "BFieldGeom/inc/BFieldManager.hh"
#include "BFieldGeom/inc/BFieldManagerMaker.hh"
#include "BeamlineGeom/inc/Beamline.hh"
#include "BeamlineGeom/inc/BeamlineMaker.hh"
#include "GeometryService/inc/VirtualDetector.hh"
#include "GeometryService/inc/VirtualDetectorMaker.hh"
#include "CosmicRayShieldGeom/inc/CosmicRayShield.hh"
#include "CosmicRayShieldGeom/inc/CosmicRayShieldMaker.hh"
#include "ExtinctionMonitorFNAL/Geometry/inc/ExtMonFNALBuilding.hh"
#include "ExtinctionMonitorFNAL/Geometry/inc/ExtMonFNALBuildingMaker.hh"
#include "ExtinctionMonitorFNAL/Geometry/inc/ExtMonFNAL.hh"
#include "ExtinctionMonitorFNAL/Geometry/inc/ExtMonFNAL_Maker.hh"
#include "ExtinctionMonitorUCIGeom/inc/ExtMonUCI.hh"
#include "ExtinctionMonitorUCIGeom/inc/ExtMonUCIMaker.hh"
#include "MECOStyleProtonAbsorberGeom/inc/MECOStyleProtonAbsorber.hh"
#include "MECOStyleProtonAbsorberGeom/inc/MECOStyleProtonAbsorberMaker.hh"
#include "MBSGeom/inc/MBS.hh"
#include "MBSGeom/inc/MBSMaker.hh"
#include "GeometryService/inc/Mu2eEnvelope.hh"
```

Special services

PhysicsListHolder_service/PhysicsList_service – manages physics lists

Geometry_service – manages geometry configuration for detectors (right now uses configuration file, future database)

```
vac_geom : {

  // General values ---
  inflectorExtensionValue : 750
  topBottomWall: 78.65
  outerWallThickness: 9.65
  torus_rmin: 277 // in
  torus_rmax: [284.75, 284.37] // in
  torus_sphi: 0   // deg
  torus_dphi: 30  // deg

  // Outer scallop ---
  phi_a: 12.83  // deg
  phi_b: 2.68   // deg
  wallR: [275.542, 268.261, 276.624, 284.750] // in
  wallPhi: [0., 11.84, 11.96, 0]  // deg
  vacR: [275.916, 268.645, 276.707, 284.370] // in
  vacPhi: [0, 11.80, 11.92, 0.0] // deg

  // Inner scallop ---
  phi_q_subtractor: 15 // deg

  // If true then use phi_q; otherwise phi_q is a calculation
  set_phi_q: false
  phi_q:  0 // deg

  zz: [2.0, 2.0]  // in
  ext: [1.0, 1.0] // in

  // Tracker ---
  tracker_sphi:  0.1 // deg
  tracker_dphi: 0.01 // deg

  // Turn counter ---
  turn_sphi: 24   // deg
  turn_dphi: 0.01 // deg

  // Visibility
  // For colors, [red, green, blue, opacity]
  displayWall : true
  wallColor : [ 0.5, 0.5, 0.5, 1.0 ]
```

python script

```
gm2ringsim::VacGeometry::VacGeometry(std::string const & detName) :
  GeometryBase(detName),
  inflectorExtensionValue( p.get<double>("inflectorExtensionValue") * mm),
  topBottomWall( p.get<double>("topBottomWall") * mm),
  outerWallThickness( p.get<double>("outerWallThickness") * mm),
  torus_rmin( p.get<double>("torus_rmin") * in), //from conversions.hh
  torus_rmax( p.get<std::vector<double>>("torus_rmax") ),
  torus_sphi( p.get<double>("torus_sphi") * deg),
  torus_dphi( p.get<double>("torus_dphi") * deg),
  phi_a( p.get<double>("phi_a") * deg),
  phi_b( p.get<double>("phi_b") * deg),
  wallR( p.get<std::vector<double>>("wallR") ),
  wallPhi( p.get<std::vector<double>>("wallPhi") ),
  vacR( p.get<std::vector<double>>("vacR") ),
  vacPhi( p.get<std::vector<double>>("vacPhi") ),
  phi_q_subtractor( p.get<double>("phi_q_subtractor") * deg),
  set_phi_q( p.get<bool>("set_phi_q") ),
  phi_q( p.get<double>("phi_q") * deg),
  zz( p.get<std::vector<double>>("zz") ),
  ext( p.get<std::vector<double>>("ext") ),
  tracker_sphi( p.get<double>("tracker_sphi") * deg),
  tracker_dphi( p.get<double>("tracker_dphi") * deg),
  turn_sphi( p.get<double>("turn_sphi") * deg),
  turn_dphi( p.get<double>("turn_dphi") * deg),
```

```
VacGeometry g(myName());
g.print();
```