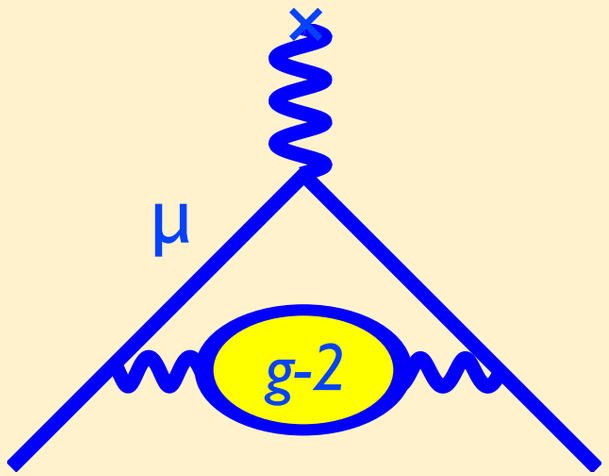
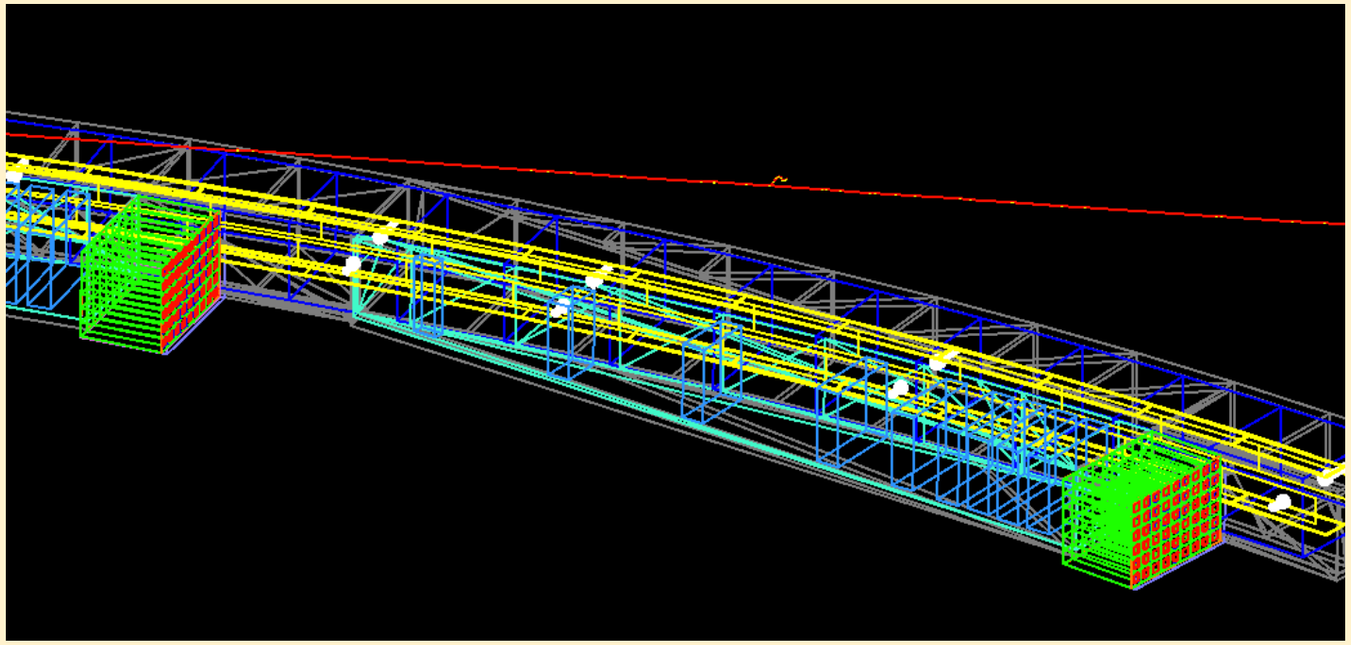
ART + GEANT4 = ARTG4

A Generic Geant4 Framework for Art

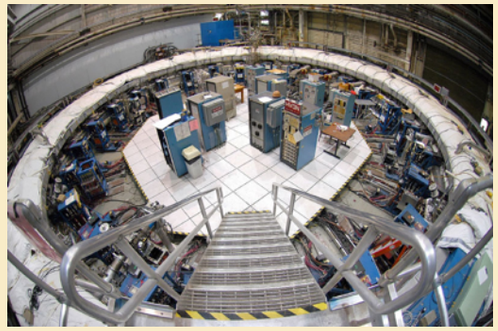
Adam Lyon
Fermilab SCD/REX + Muon *g-2* experiment
Computing Techniques Seminar
March 19, 2013



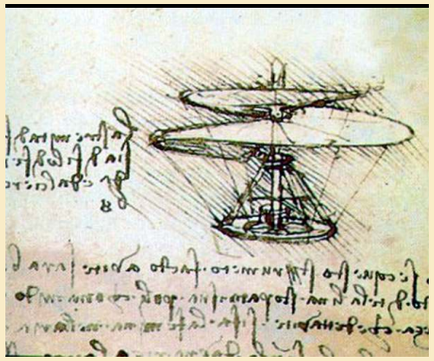


Outline of this seminar

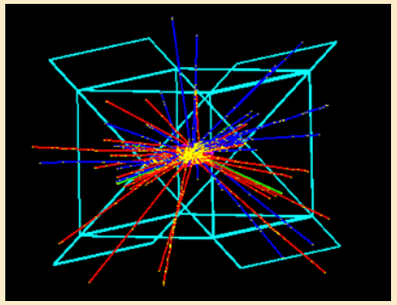
1) Motivation and Context
(+ a little bit of Muon g-2)



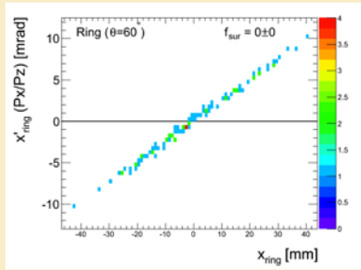
2) The ART Framework



3) ArtG4



4) gm2ringsim



Who am I?

Data Handling Group Leader in the SCD Running Experiments Department (e.g. SAM)

DØ Experiment – Dibosons & Data handling

Muon *g-2* – Computing and simulations

Our ultimate goal

We want to work together!

We need a software system that makes working together easy while maintaining or sanity

What does this mean?

- **Following best coding practices?**
- **Using standard libraries and APIs?**
- **Creating your own libraries for others to use?**
- **Share your code in a repository?**
- **Documenting your code?**
- **Find infrastructure code from somewhere?**

Yes to all the above

Requirements on physics software for physicists

- **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

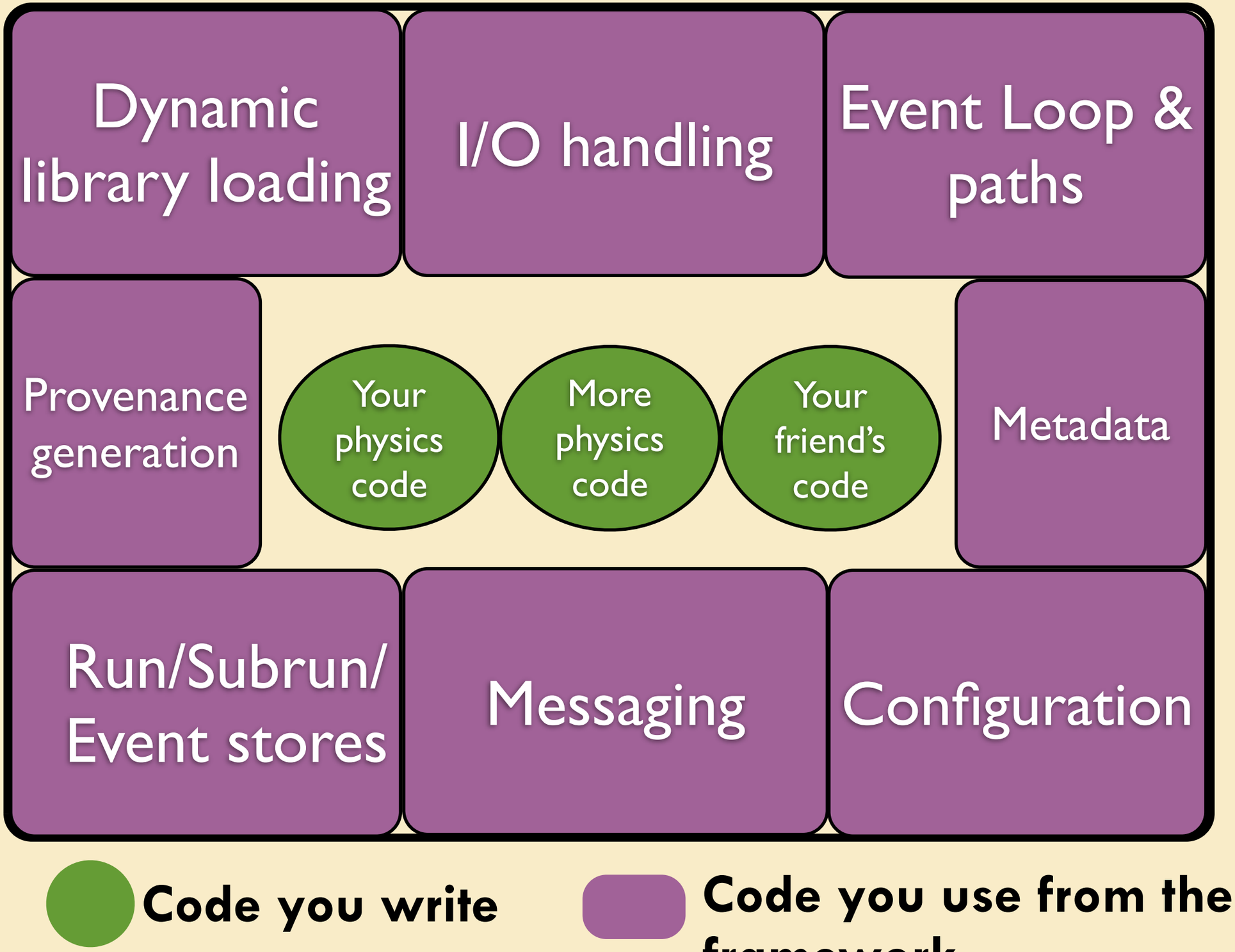
Requirements on physics software for physicists – solutions

- 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

What does a framework do?



What a framework gives you

Allows you to write your physics code without worrying about the infrastructure. Makes it easy to work with others.
<u>But not for free – you have to learn it!</u>

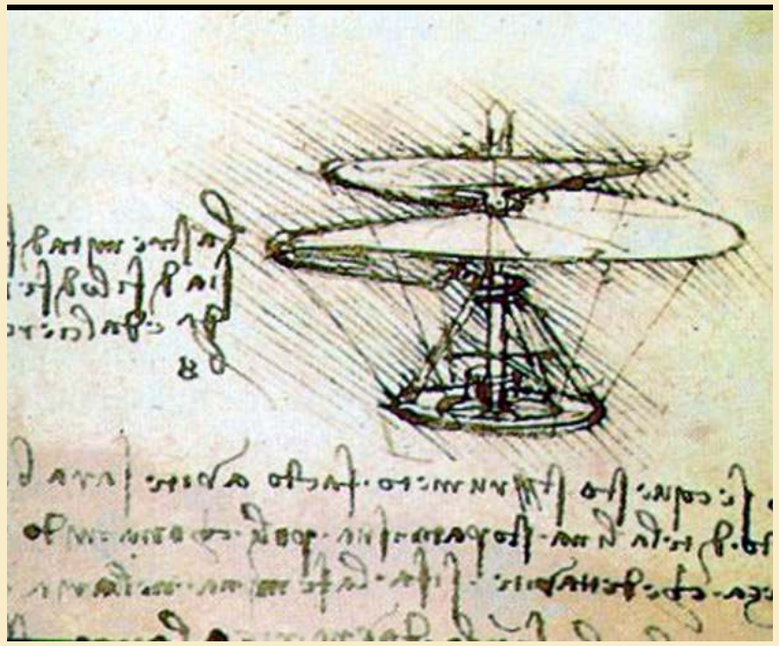
Some people find such a system <u>constraining</u>:

- **Infrastructure is hidden behind the scenes from you**
- **Your ideas may not be included**
- **You have to trust a system you didn't write**
- **You miss out on the fun of writing super-cool complicated C++ code**

Some people find such a system <u>liberating</u>:

- **You can concentrate on physics code**
- **Your C++ is pretty easy (you are *using* a complicated system, not *writing* it)**
- **You get to miss out having to maintain the complicated C++ code (yay!)**
- **You can use code from others and share yours with others**
- **You can get services for free (e.g. data handling)**

Fermilab's common framework from the Scientific Computing Division



ART

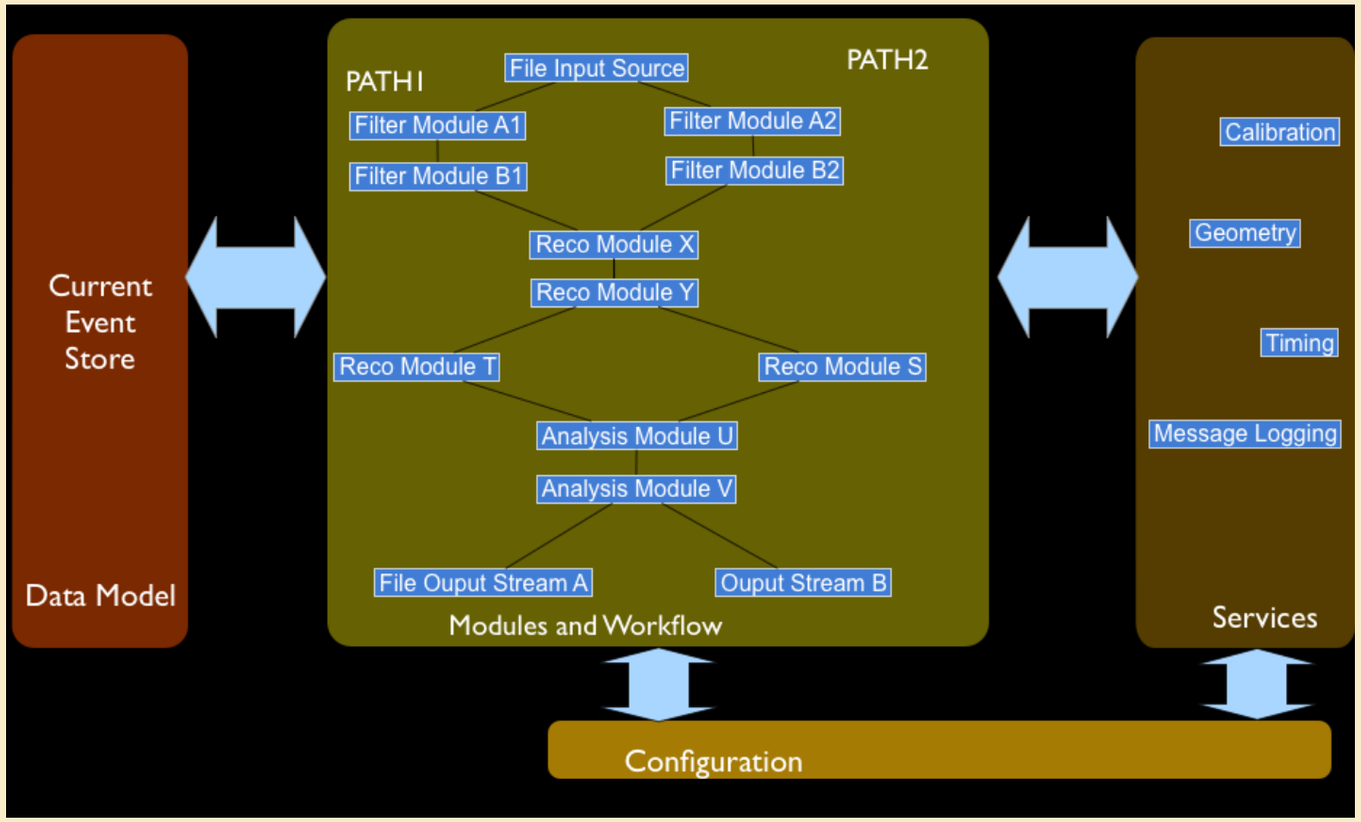
A "lite" forked version of the CMS framework

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



Why not write our own framework?

At Muon *g-2* (and I suspect most other small experiments),
<u>We don't have...</u>

- **The expertise**
  Writing large C++ systems is hard (need low dependences, efficient generic programming, follow software engineering best practices)

- **The time**
  With lots of milestones and reviews, there's no time to devote to correctly writing such a large system

- **The energy**
  We just wanna make plots! Not write infrastructure code

Muon g-2's solution

- ART for the framework
- CMake for the build system (same as used by ART developers)
- git & Redmine for repository and software management
- Relocatable UPS for release management
- Custom script (gm2d) for development environment tasks

```
[lyon@gm2gpvm03 workshop1]$ gm2d -h
Usage gm2d (listTags | newDev | getRedmineGit | newProduct | setup_for_development | build | zapBuild | updateDepsCM | updateDepsPD) [-h for help]"

  Tools ( for help on tool, do "gm2d <tool> -h" )

   newDev (n)                 Start a new development area
   getRedmineGit (g)          Clone a Redmine git repository
   newProduct (p)             Create a new product from scratch
   setup_for_development (s)  Setup a development enviornment
   build (b)                  Run buildtool

   zapBuild (z)               Delete everything in your build area
   listTags (l)               List the git tags for a product
   updateDepsCM (uc)           Update CMakeLists.txt file for latest dependencies
   updateDepsPD (up)           Update product_deps file for latest dependencies
```

Our first task

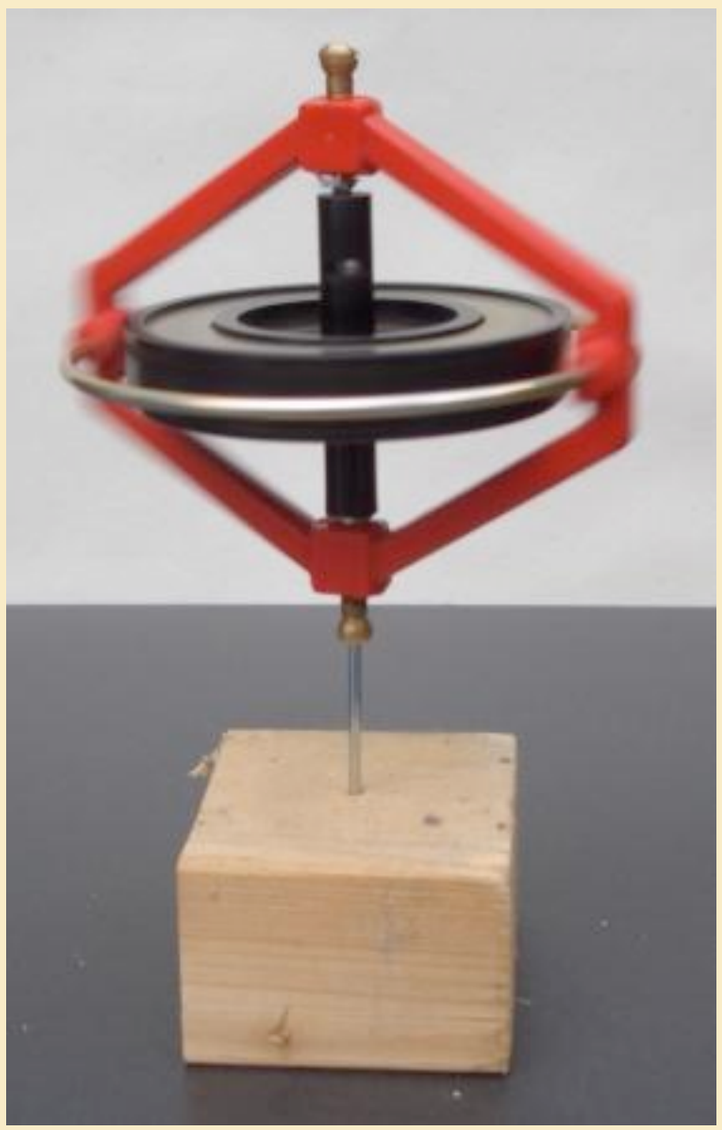
Convert our simulation code to Art

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

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

Spin and magnetic moments

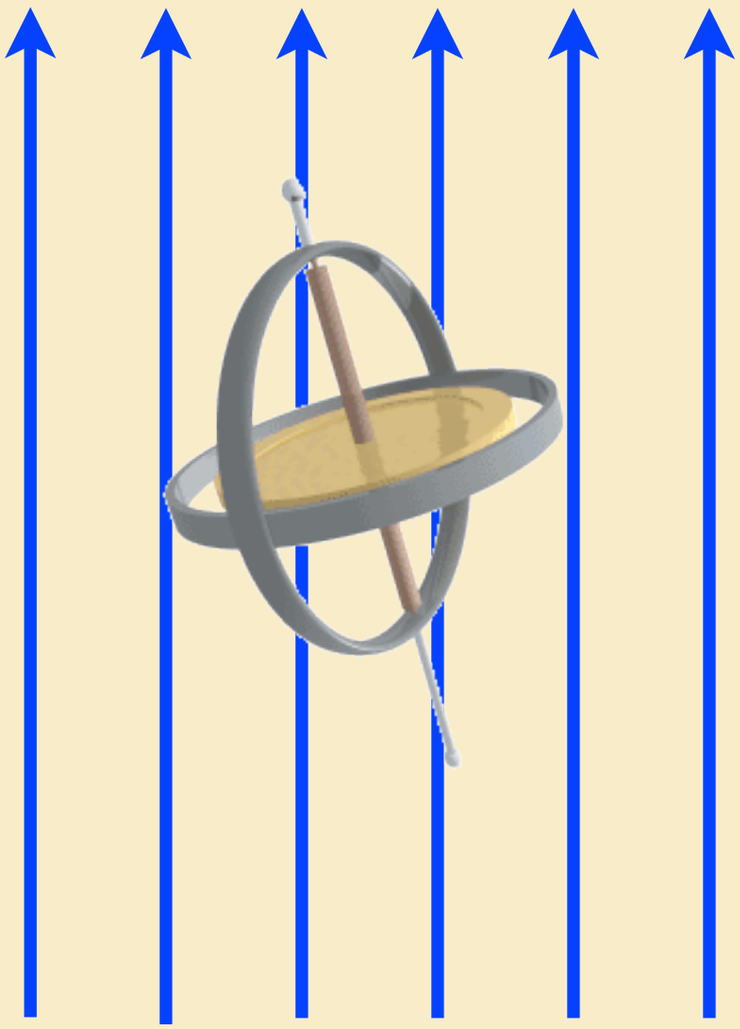
Elementary particles have an intrinsic angular momentum – called SPIN



Put a particle in a magnet, and the spin PRECESSES about the magnetic field

Precession frequency:

$$\omega_s = g\frac{eB}{2mc}$$



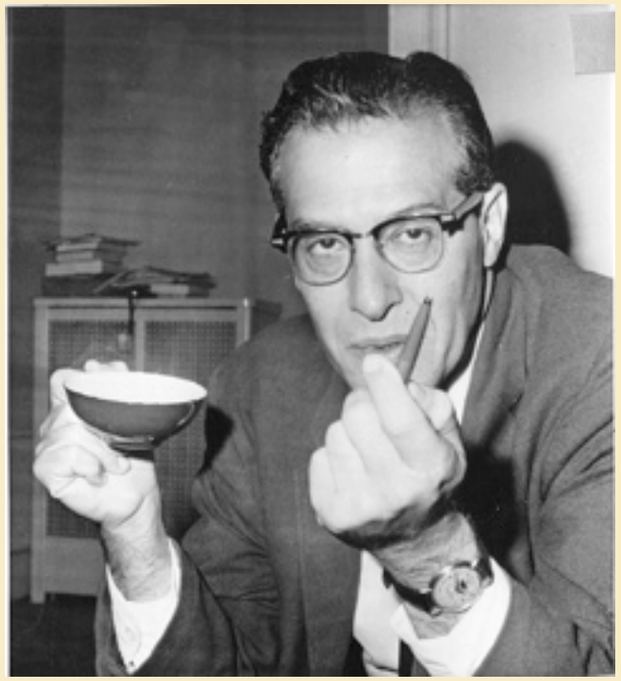
In classical systems, $g = 1$

For elementary particles, $g = 2$ + a little more ("anomalous" part)

The "+ a little more" is the really cool part

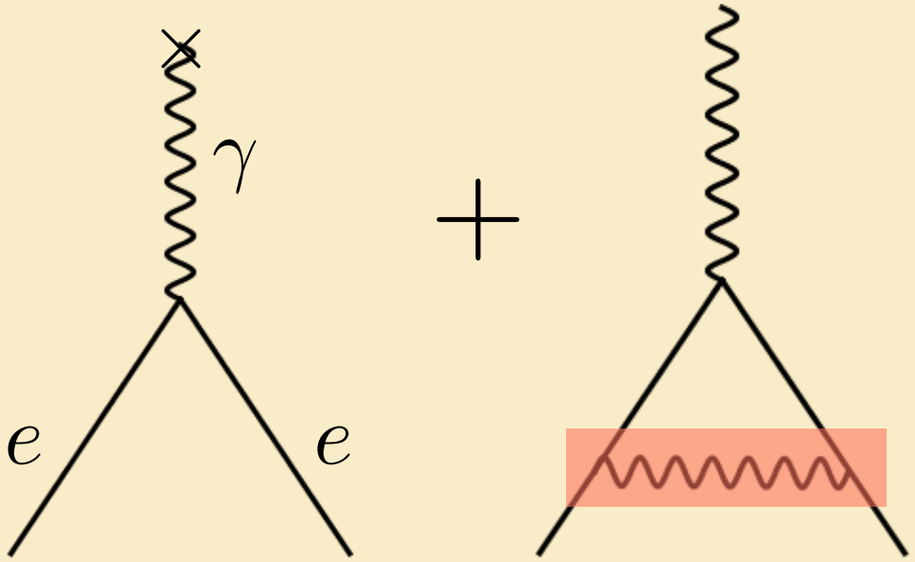
Empty space (the vacuum) is not empty.

There are "radiative corrections" that affect particles.



Julian Schwinger (Nobel Prize 1965)

$g_e =$

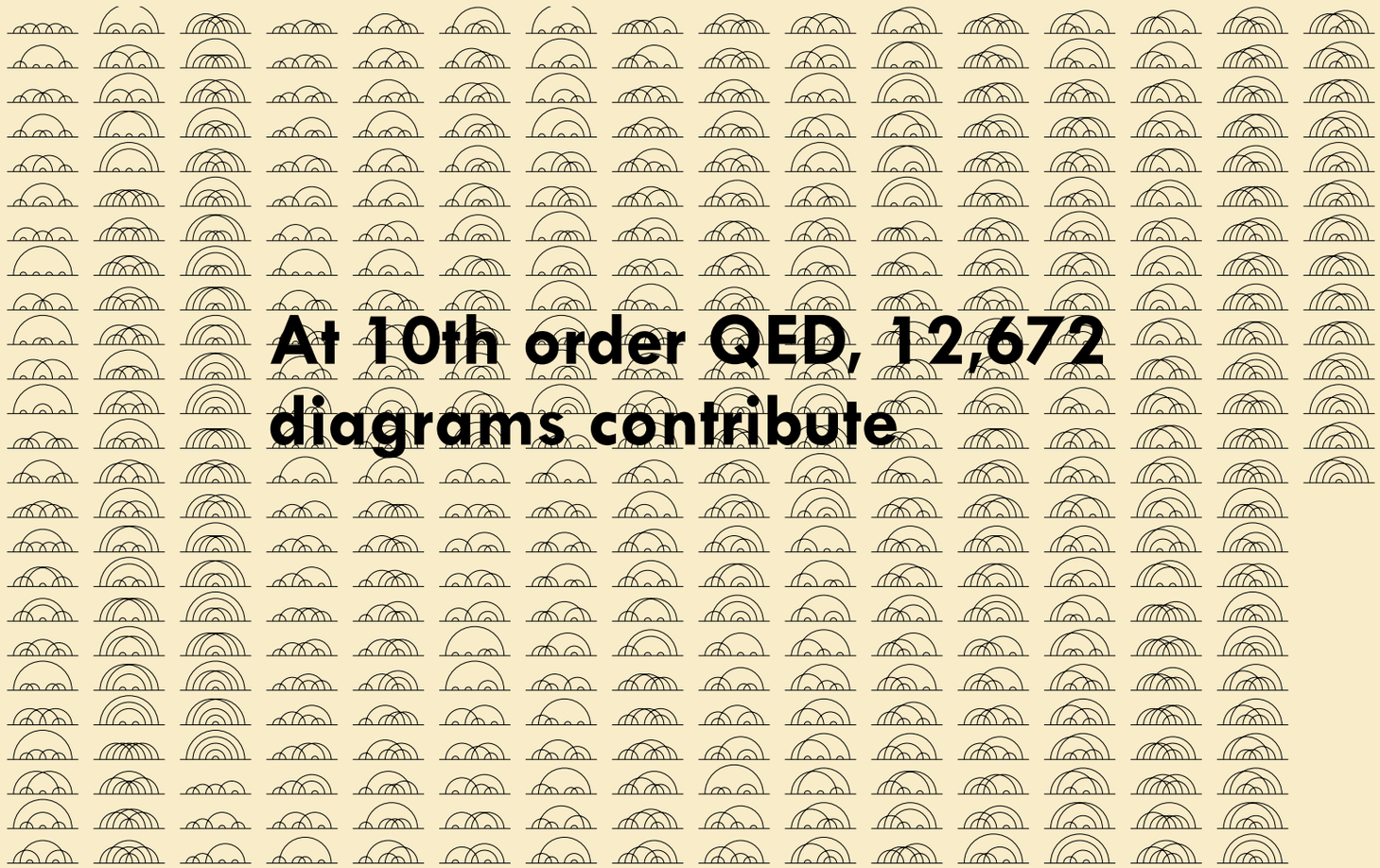


$+$

$+ \quad \cdots$

$2 \quad + \quad 0.00236 \quad + \quad \cdots$

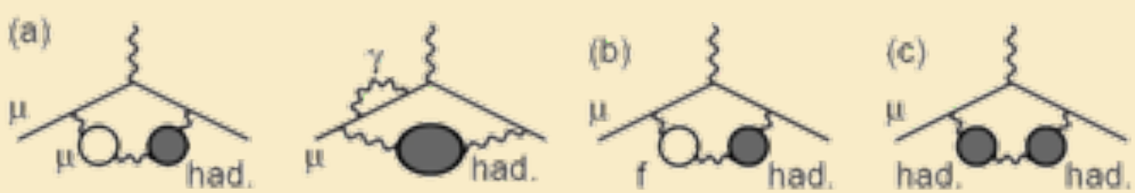
The "+ a little more" can become very complicated

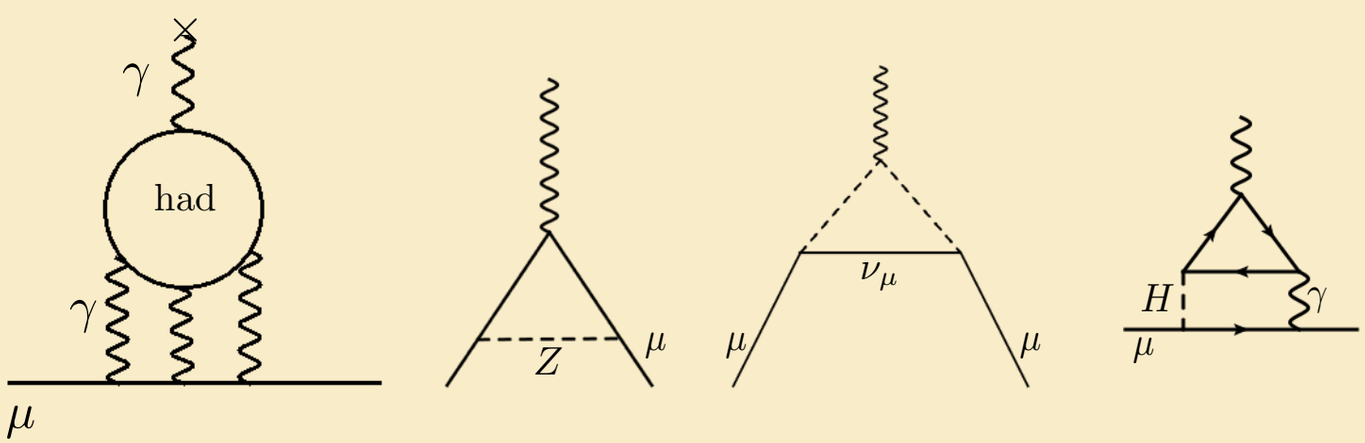


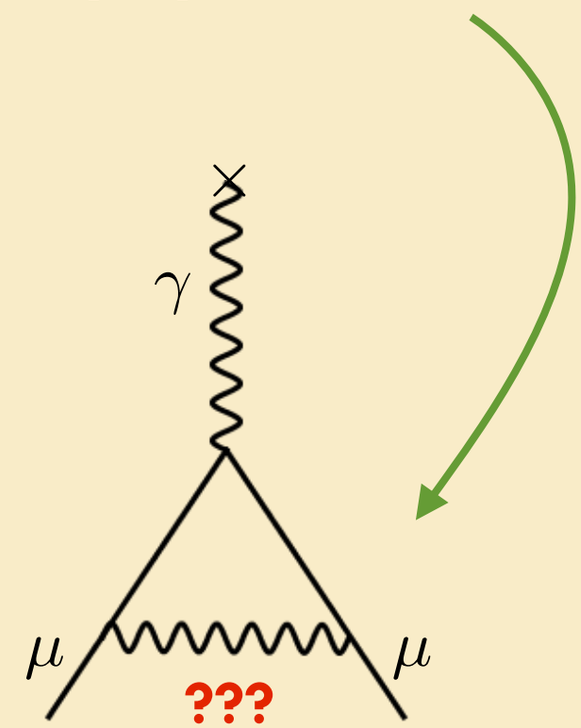
At 10th order QED, 12,672 diagrams contribute

Comparing the theoretical prediction to a measurement is a great test of the Standard Model

A disagreement would indicate presence of new physics!

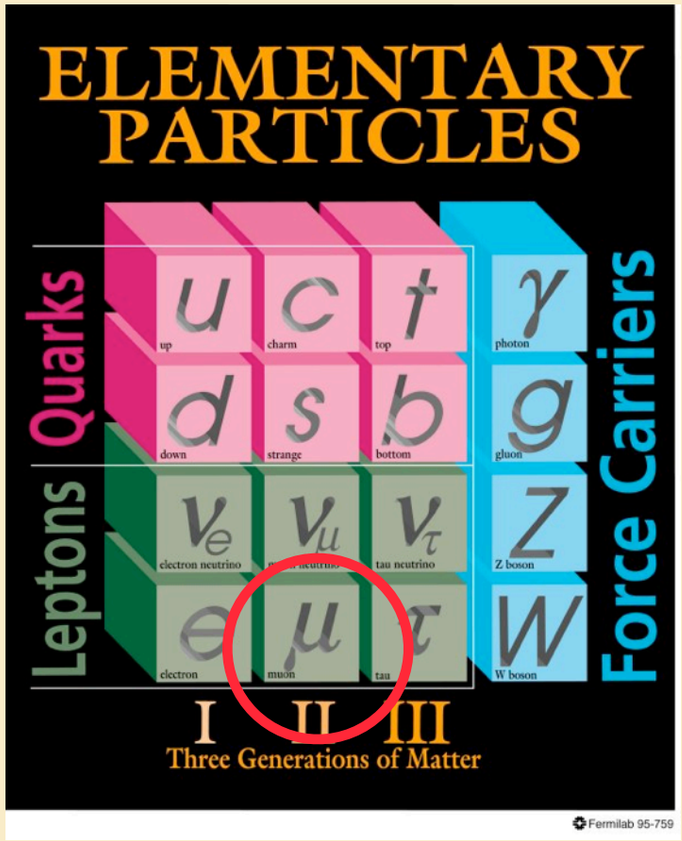






We use muons for discovery

- Muons are like electrons, but 200 times heavier
- They decay to electrons and neutrinos
- They are made in decays of pions



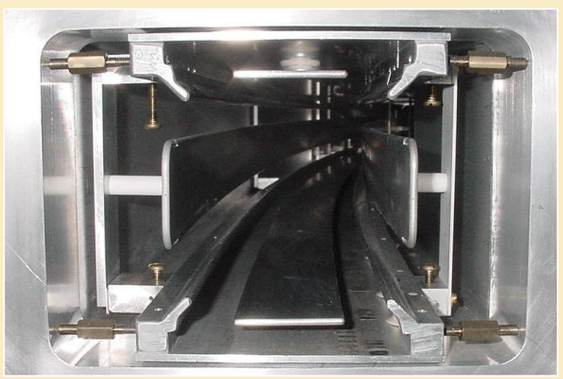
Muons are more sensitive than electrons to very subtle radiative corrections

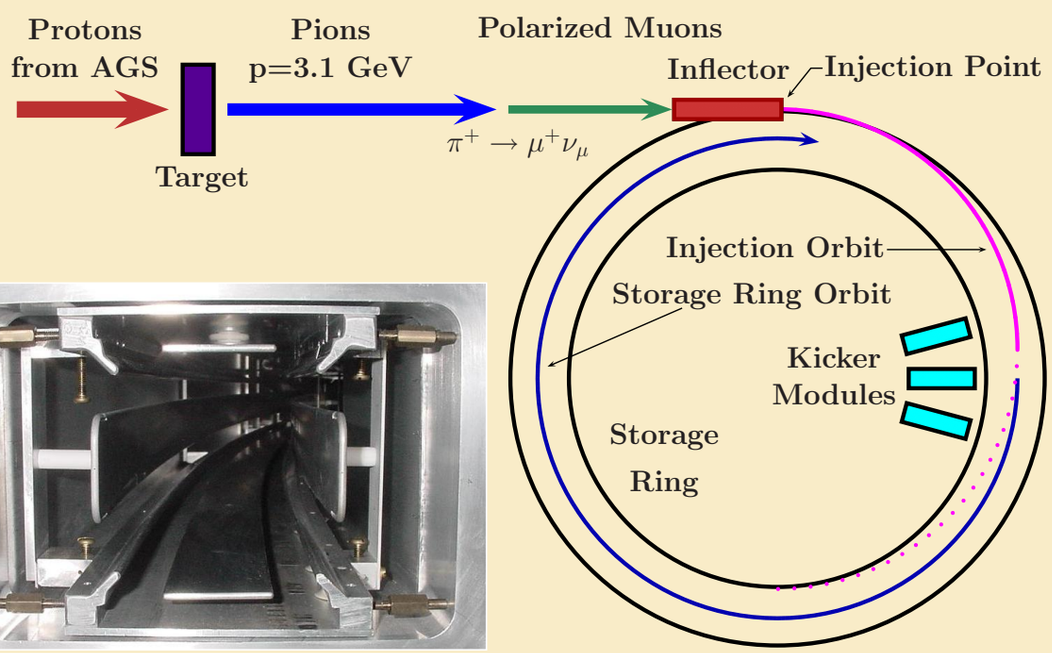
Measure $a_\mu = \frac{g_\mu - 2}{2}$

This measurement has over 60 years of history!
Nevis (Columbia), CERN, Brookhaven, and now FNAL!

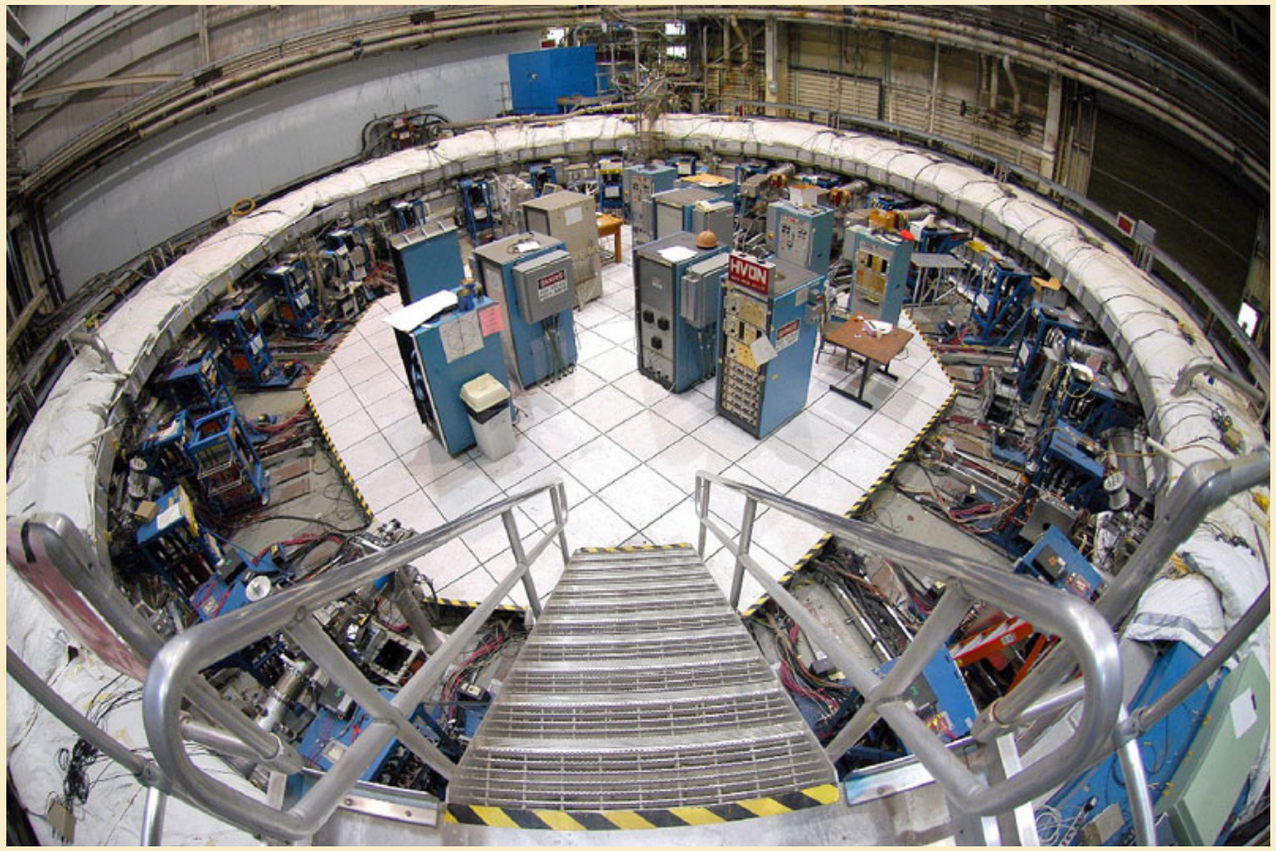
Put the muons in a magnetic field

To make the muon spins precess, put them in a magnetic storage ring – Brookhaven already has one!



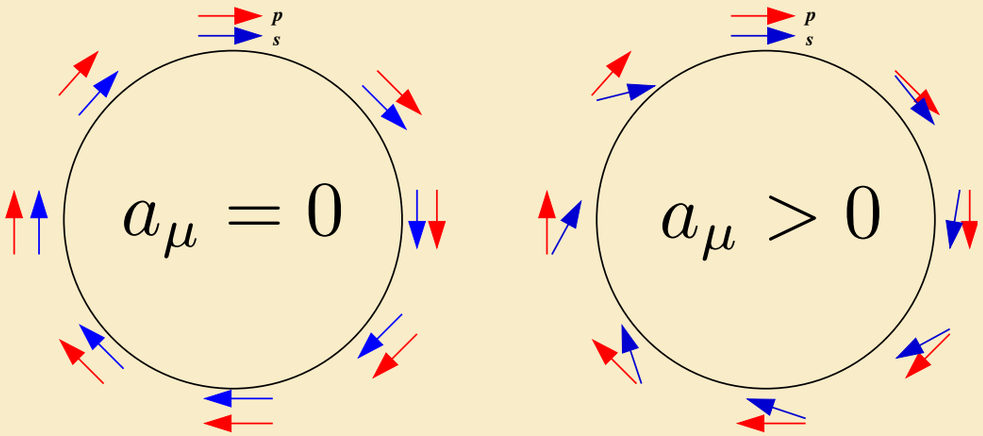


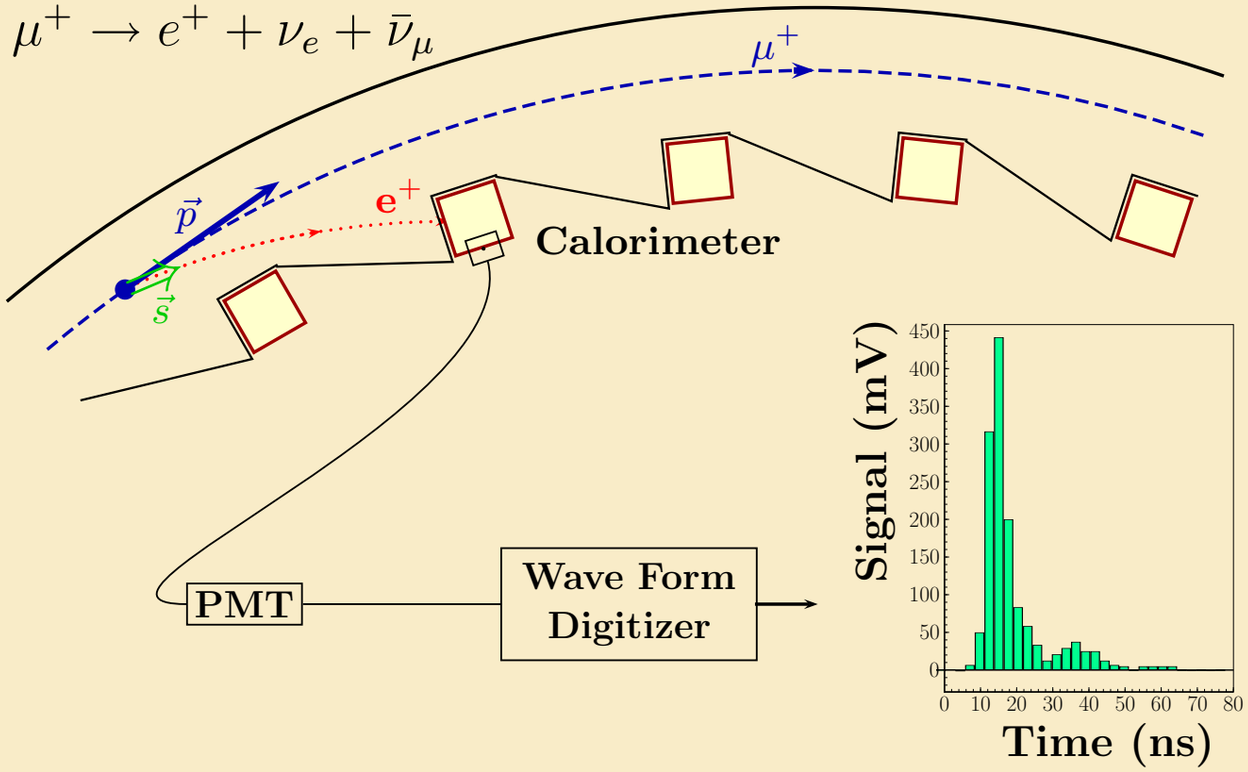
E821 Muon *g-2* ring at Brookhaven National Laboratory

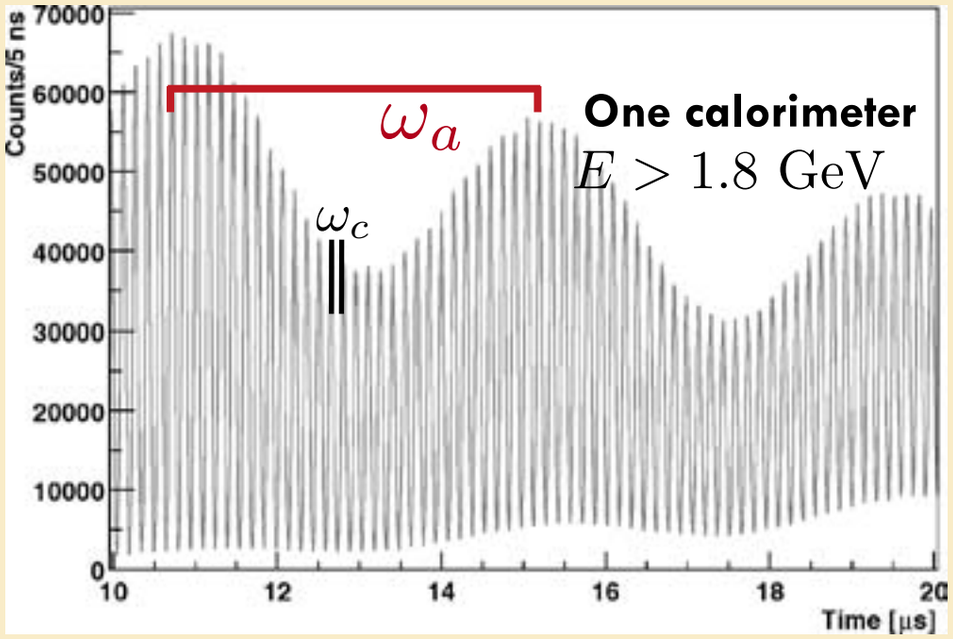


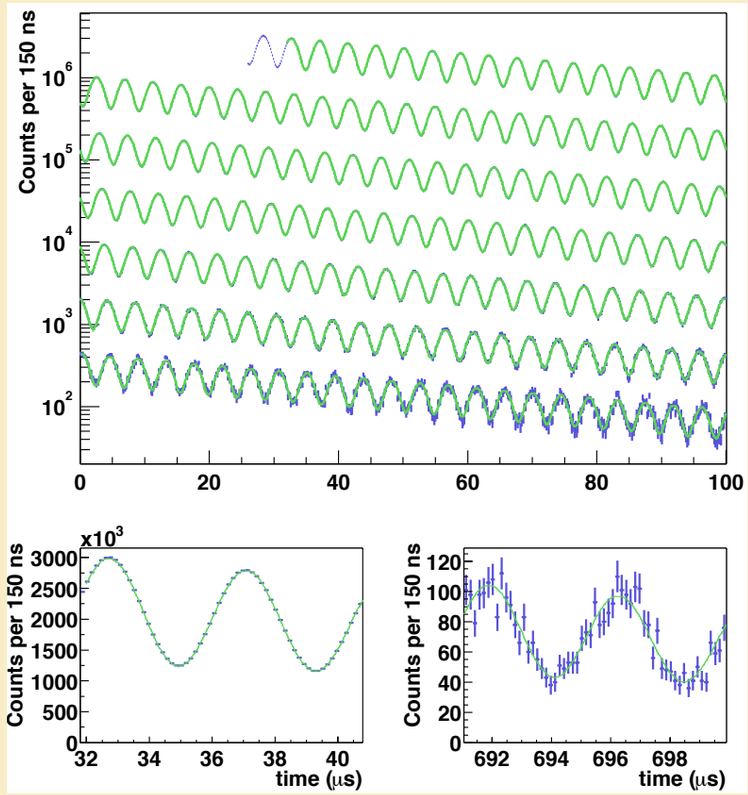
Muons travel at the "magic momentum" $(p_\mu = 3.09\ \mathrm{GeV}/c)$ **(0.9995c and have lifetime of 64 microseconds)**

How do we measure the anomalous moment?

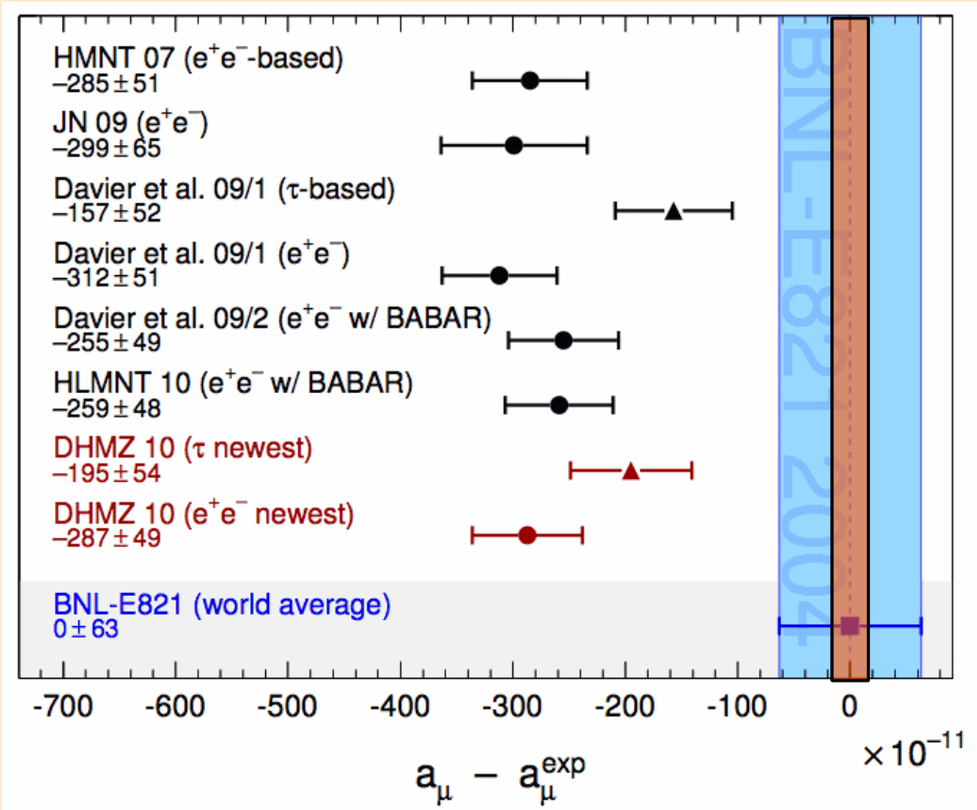








Previous experiment (BNL E821)



$$a_\mu^{\rm QED} = 0.00\,116\,584\,718\,09(15)$$

$$a_\mu^{\rm had} = 0.00\,000\,006\,930(49)$$

$$a_\mu^{\rm EW} = 0.00\,000\,000\,154(2)$$

$$a_\mu^{\rm SM} = 0.00\,116\,591\,802(49)$$

$$a_\mu^{\rm exp} = 0.00\,116\,592\,089(63)$$

0.54 ppm measurement

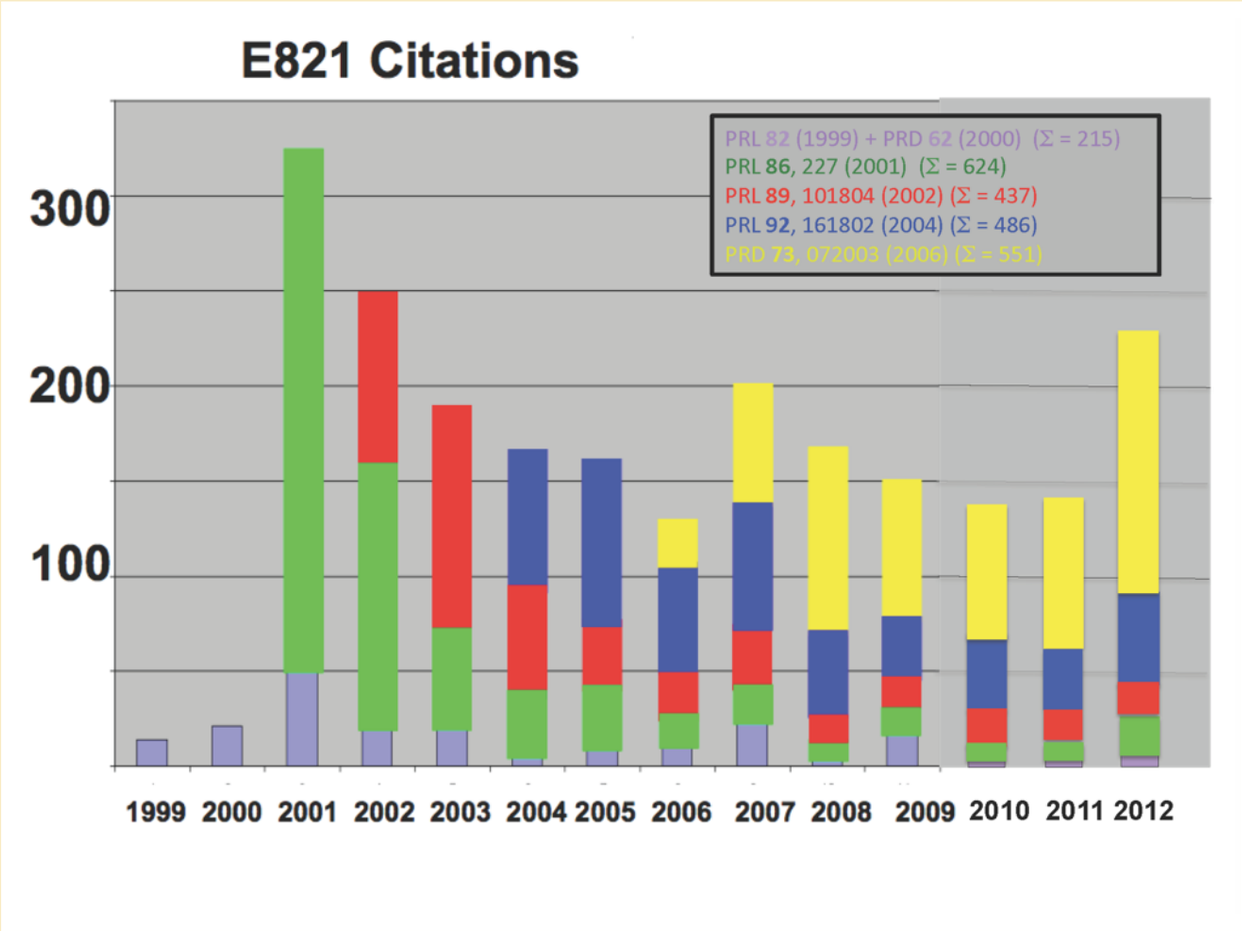
$$\mathbf{a}_\mu^{\rm exp} - \mathbf{a}_\mu^{\rm SM} = \mathbf{287(80)} \times \mathbf{10^{-11}}$$

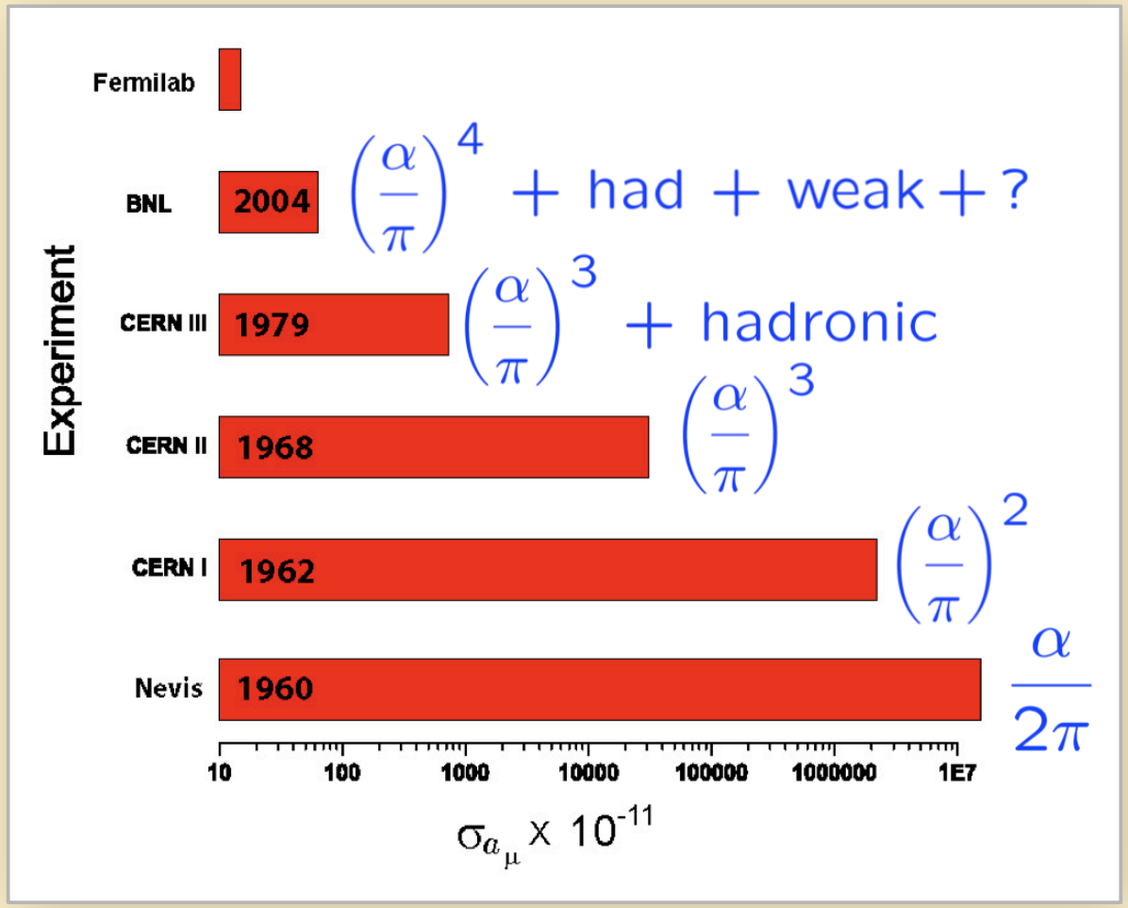
$$> 3\sigma$$

Large enough difference to be interesting, but not conclusive

Repeat at FNAL for 0.14 ppm

Enormous interest in the result



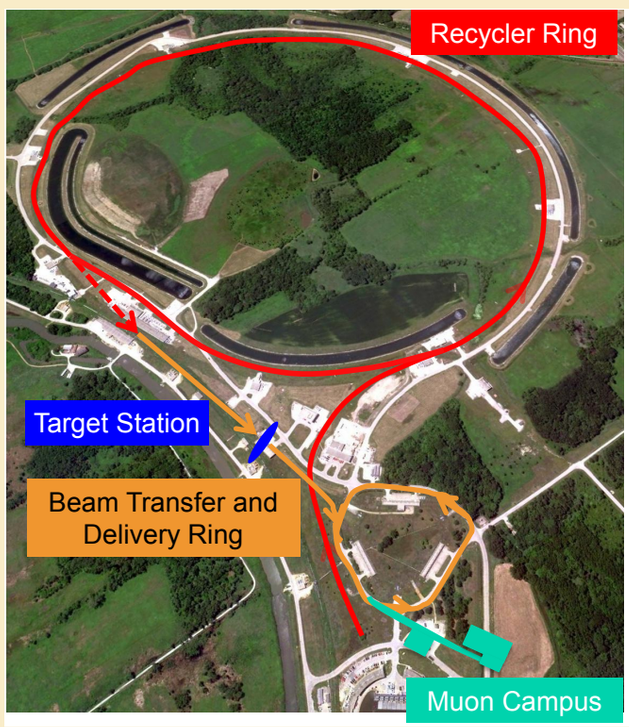


With a 0.14 ppm measurement at Fermilab, current difference becomes 5.6σ (7.5σ if theory drops to 0.3 ppm)

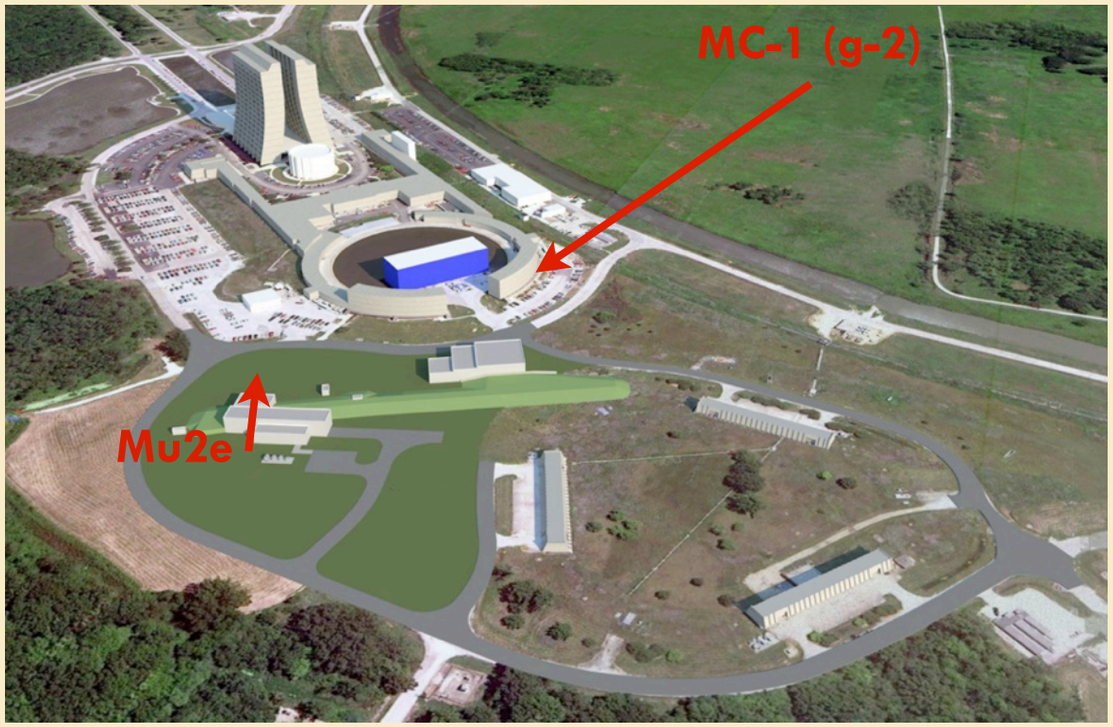
If difference persists, we're talking major discovery!

What we'll do better at Fermilab

Better statistics (21x more muon decays than E821)
Smaller backgrounds (E821 suffered from "hadronic flash")
More precise magnetic field measurement, better detectors - almost everything will be better



- Recycler
  - 8 GeV protons from Booster
  - Re-bunched in Recycler
  - New connection from Recycler to P1 line (existing connection is from Main Injector)
- Target station
  - Target
  - Focusing (lens)
  - Selection of magic momentum
- Beamlines / Delivery Ring
  - P1 to P2 to M1 line to target
  - Target to M2 to M3 to Delivery Ring
  - Proton removal
  - Extraction line (M4) to g-2 stub to ring in MC1 building



The Fermilab Muon Campus

Moving the ring from Brookhaven to Fermilab

The hard part is moving the three superconducting coils

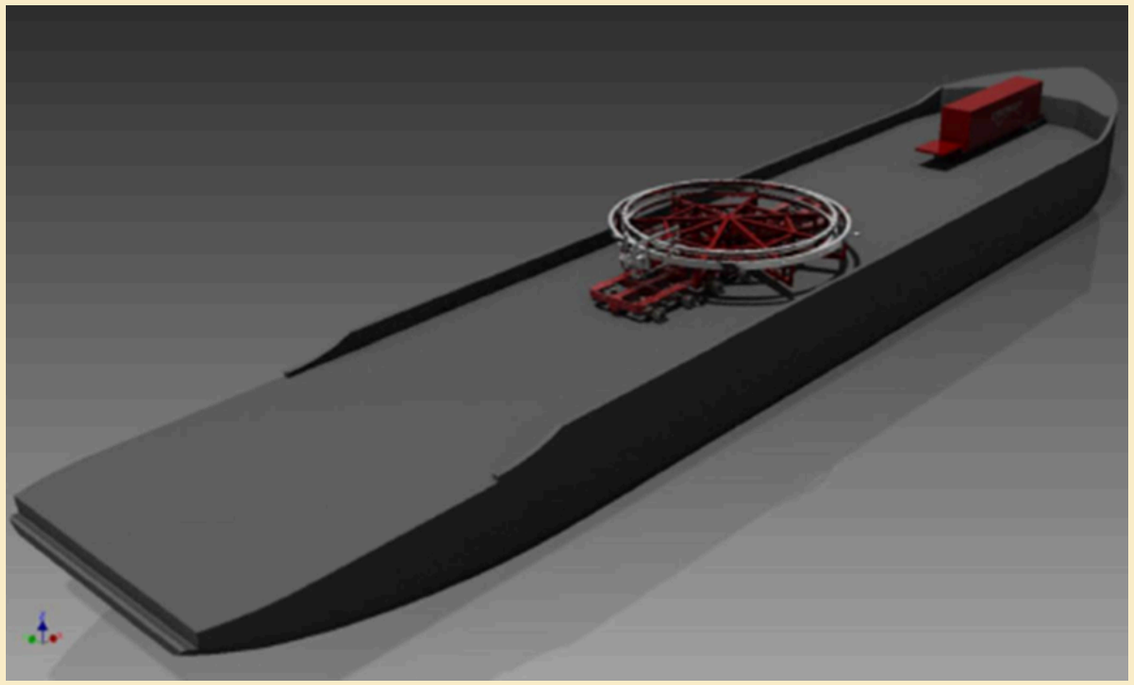
Continuously wound coils, can't break into pieces

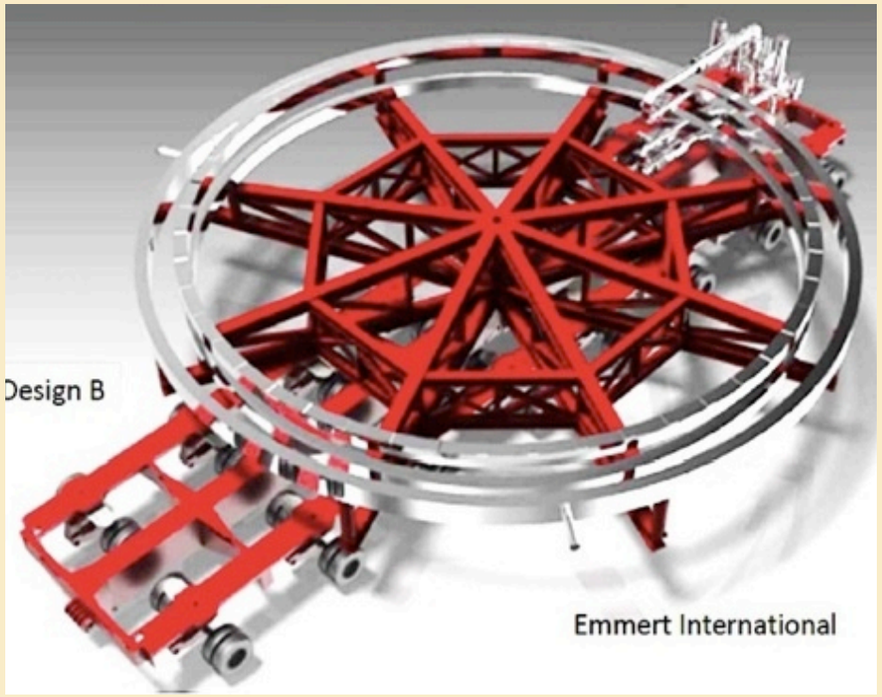
They're big!
50 ft diameter
(takes up ~ 4 lanes on the highway). Total load 17 tons (<< 40t max)

~ $3M to move. Would be ~10x more if we had to construct them anew

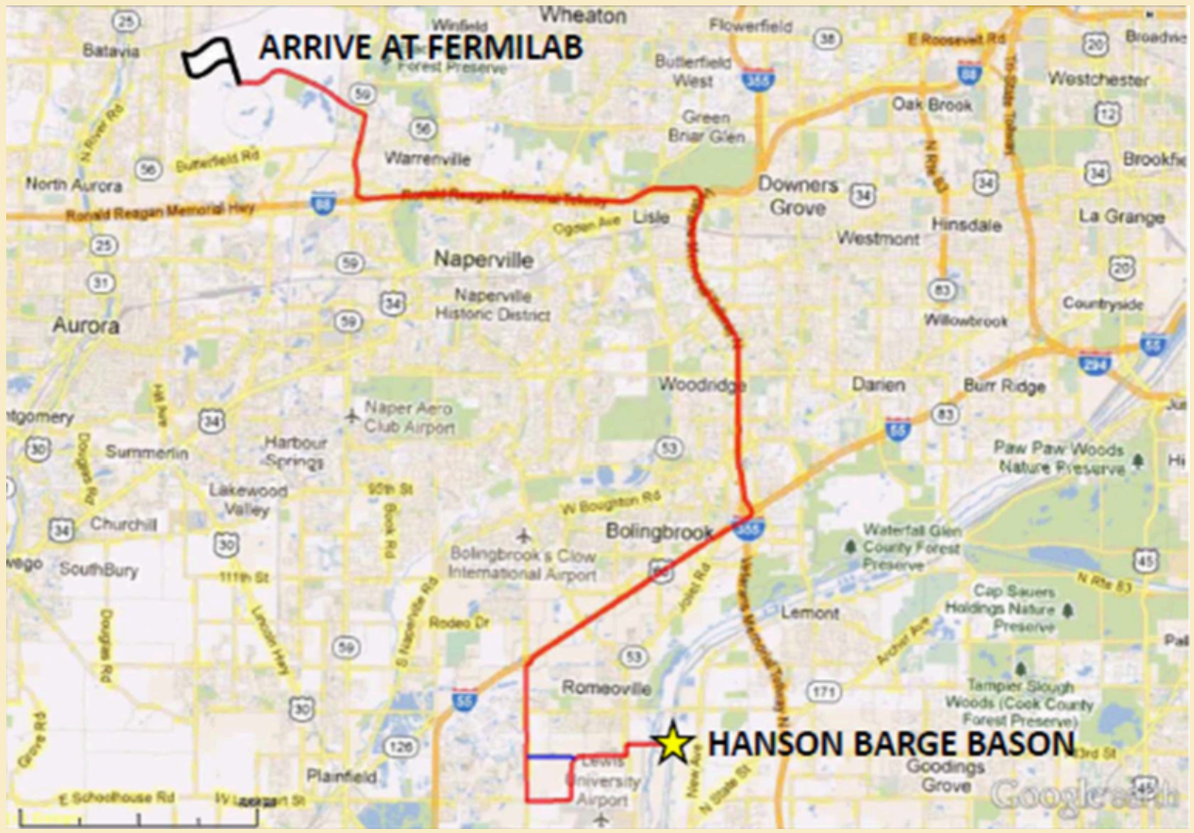
The ultimate road/water/road trip

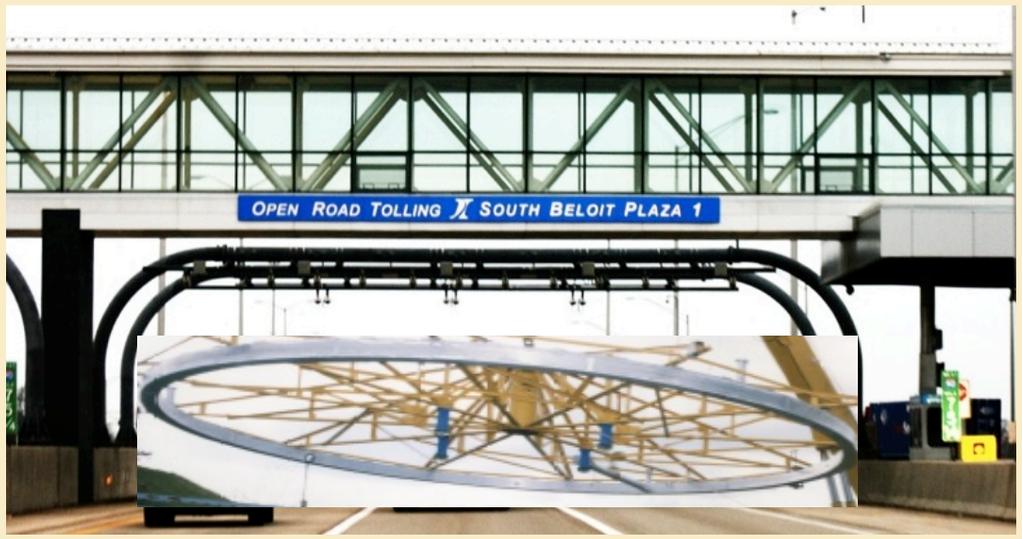
Long part by barge (two possible routes; ~ 1 month travel time)





Hardest part – from barge to FNAL



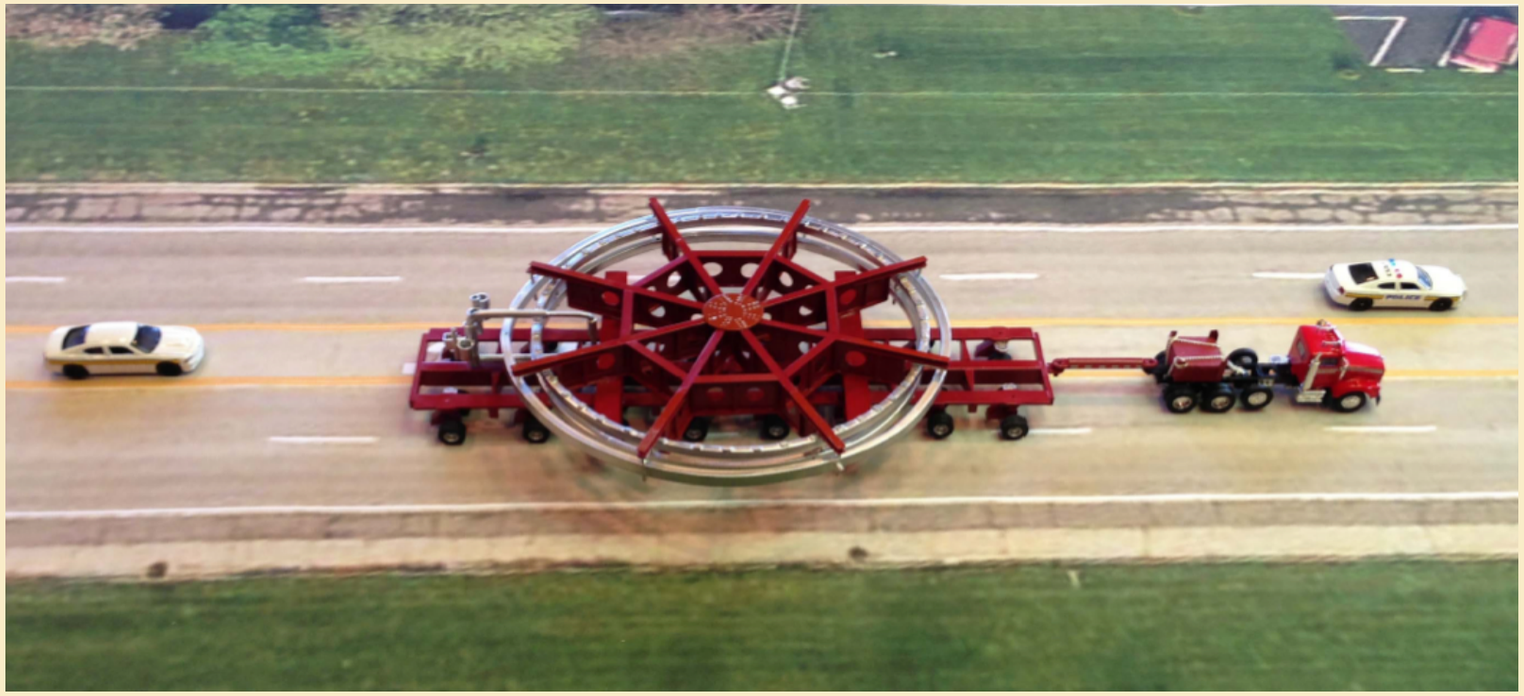


~ 4 lanes across
Fits through express toll plaza with 1 foot to spare!

Travel speed < 5 mph

Trailer has its own steering, leveling and height controls

(Picture at right is a model!)



… back to code

Simulations did not play a big role in BNL E821.

There was a Geant3 based simulation for beam dynamics studies

After E821, started asking questions about how to improve the apparatus

- **Add more kicker magnets?**
- **Alter the inflector magnet?**

A more sophisticated simulation was required

The *g2migtrace* simulation

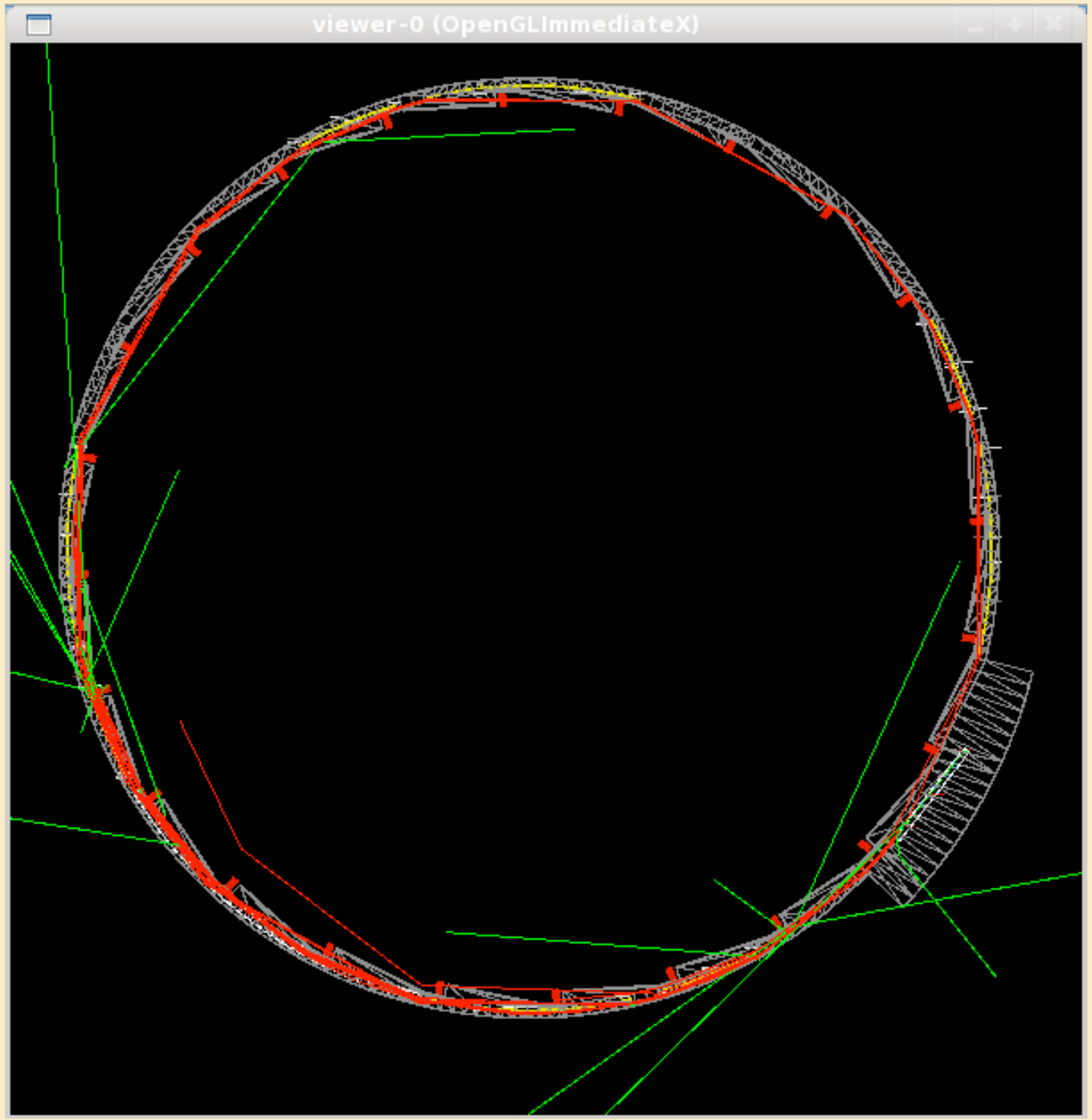
Muon Injection Geometry TRacking And Capture Efficiency

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

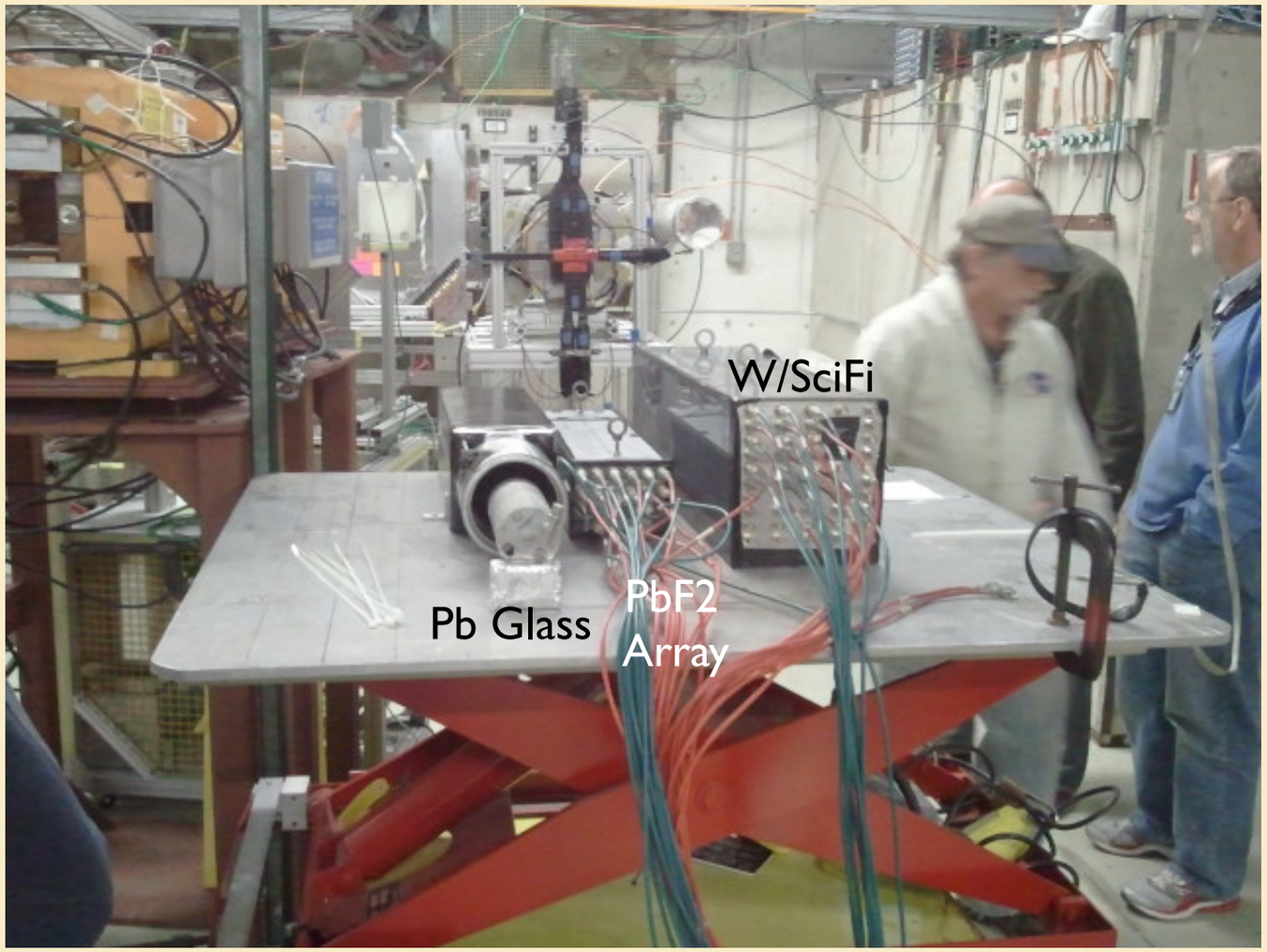
BUT - is a monolithic program.

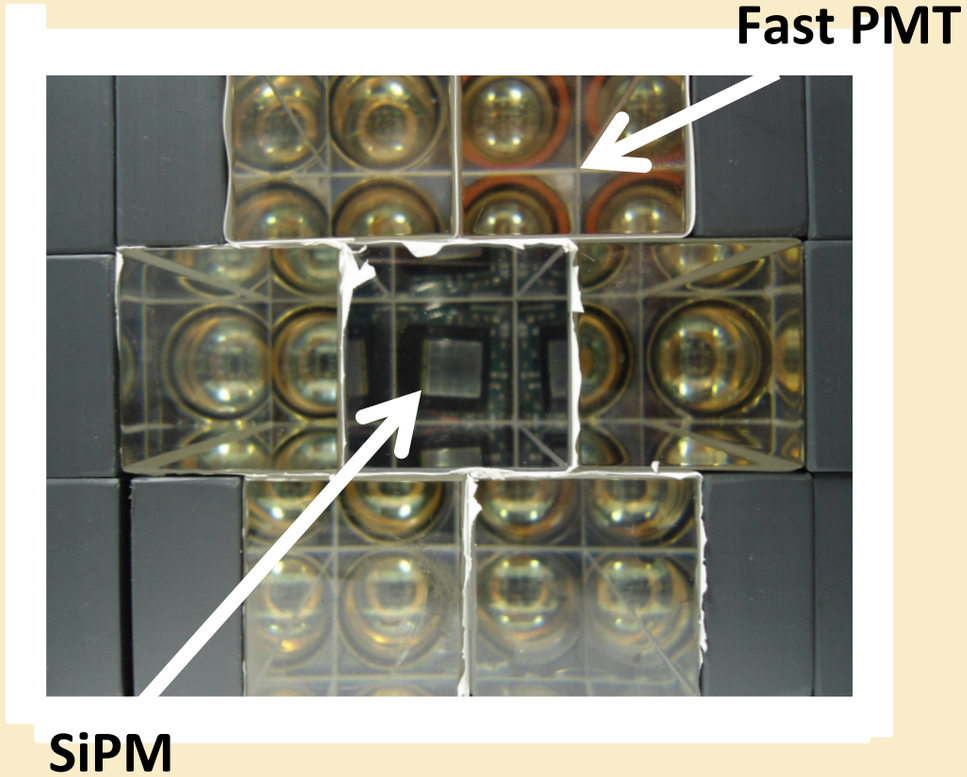
```
    To calculate the quadrupole E fields from the polar maps produced
    below, we find it necessary to convert (x,y,z)_world into
    (r_q,th_q)_quadrupole (See below coordates system).  Then do the
    interpolation on the polar grid, convert (E_rq,E_thq) into
    (E_xq,E_yq), and then finally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

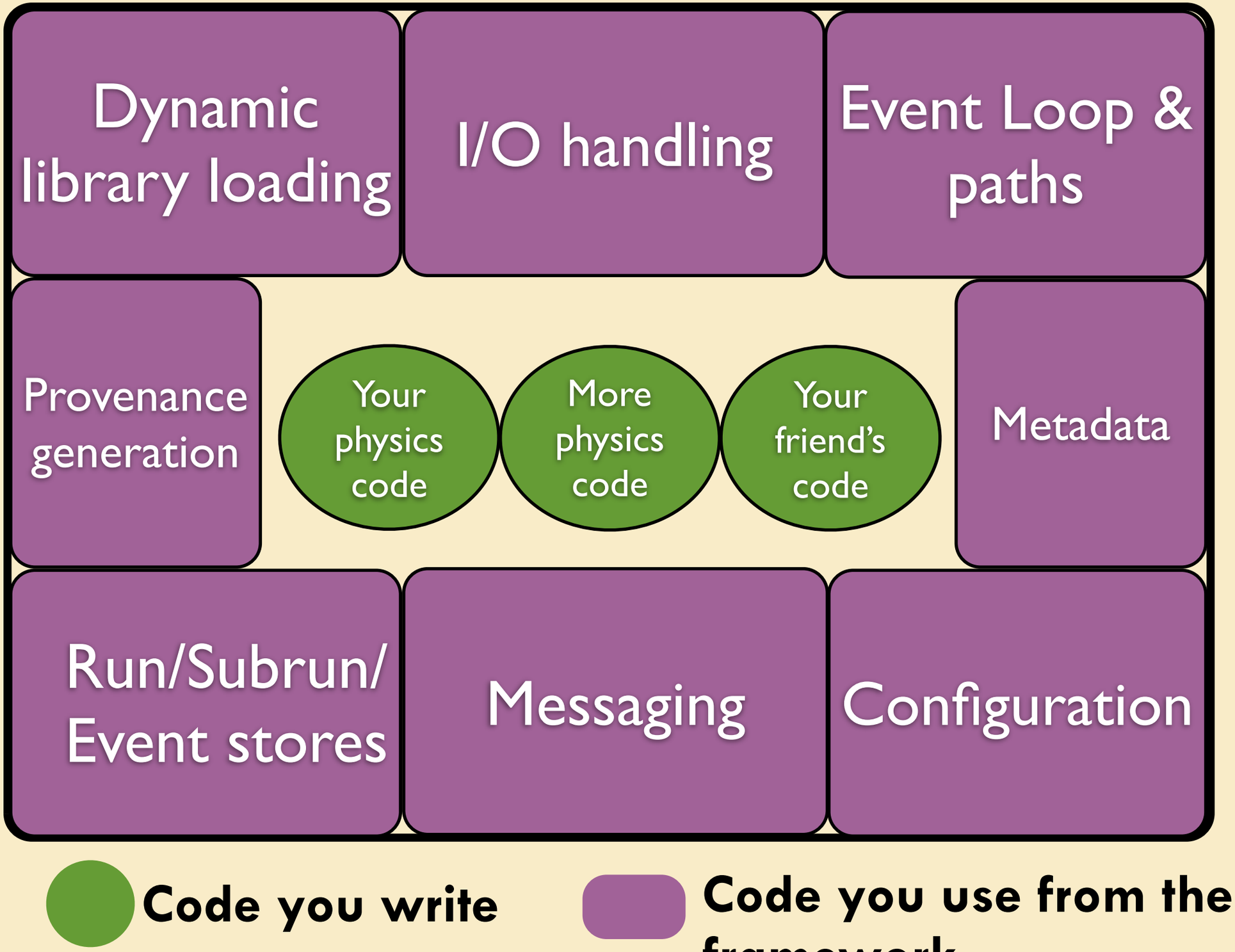
ART

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

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)

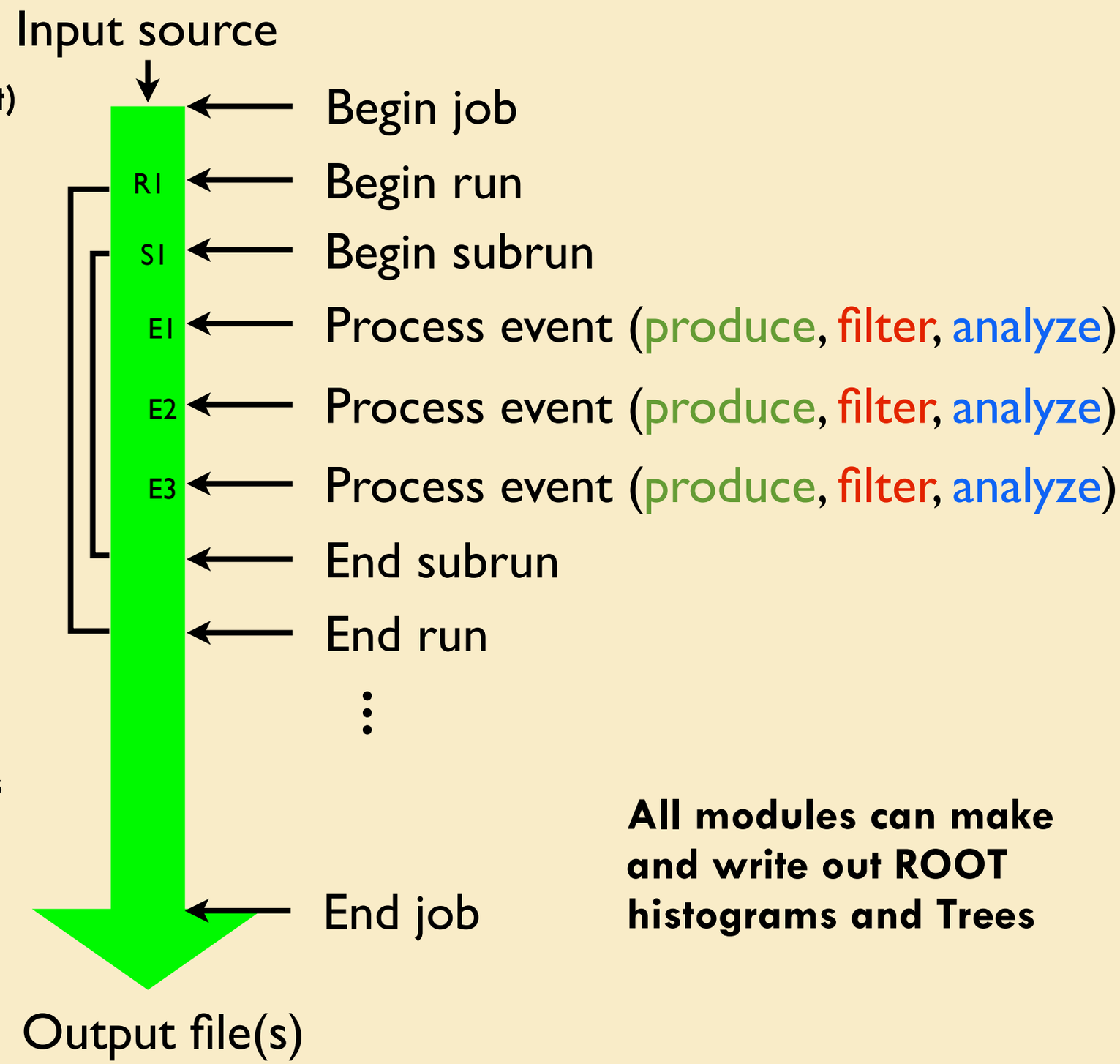
o Input source:
A source for data. E.g. a ROOT file or Empty for start of simulated data

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

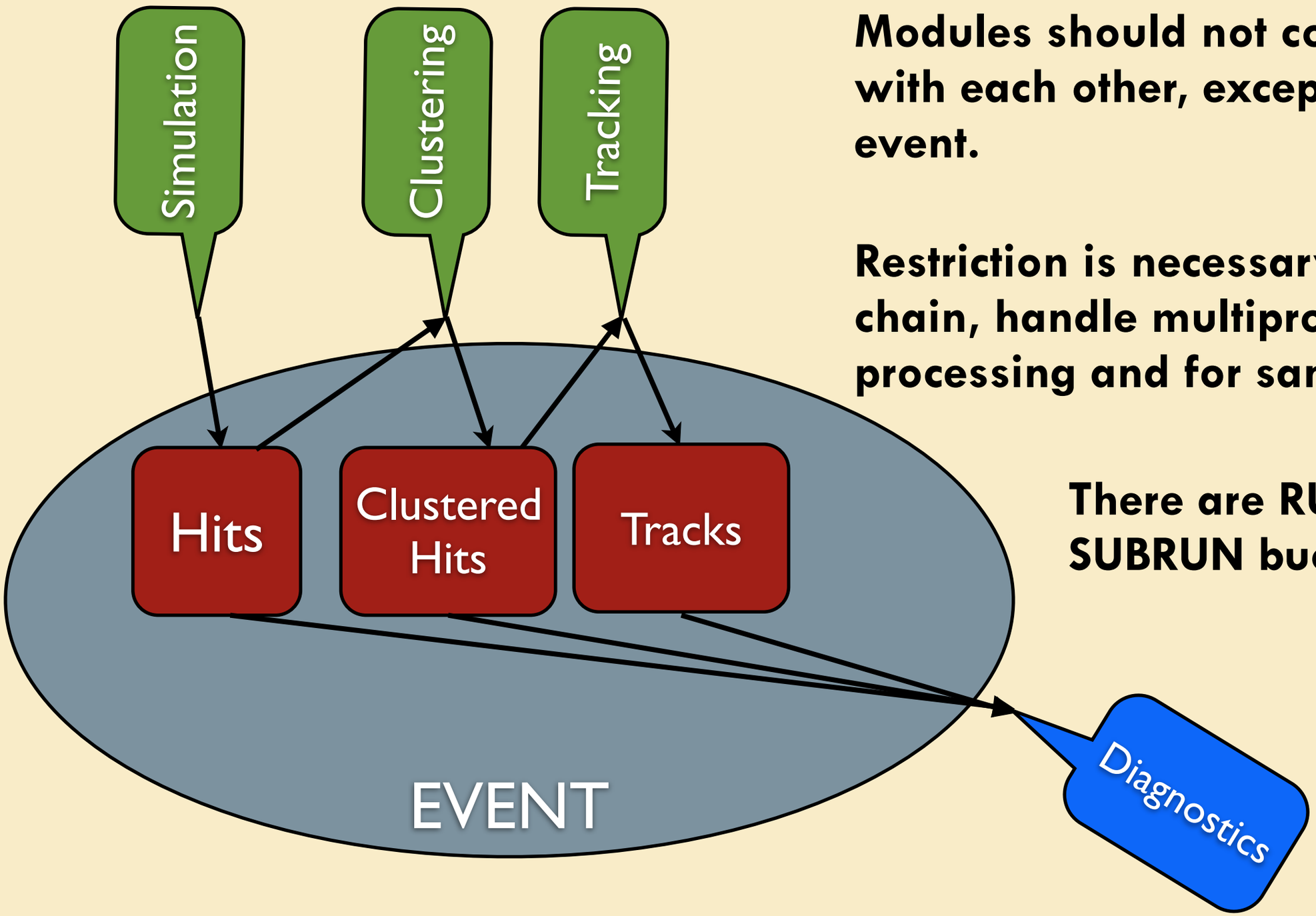
o Output module:
Writes data to output file (ROOT). Can specify conditions and have many files



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

An example “Hello world” module

HelloWorld1_module.cc

Note no header

override is helpful

Must have DEFINE at bottom

The “artmod” scripts writes this skeleton for you

This gets built into its own shared object

```
#include "art/Framework/Core/EDAnalyzer.h"
#include "art/Framework/Core/ModuleMacros.h"
#include "art/Framework/Principal/Event.h"

namespace artex {


  class HelloWorld1 : public art::EDAnalyzer {

  public:

    explicit HelloWorld1(fhicl::ParameterSet const& pset);

    void analyze(const art::Event& event ) override;

    virtual ~HelloWorld1();

  };

  HelloWorld1::HelloWorld1(fhicl::ParameterSet const& ){}

  HelloWorld1::~HelloWorld1() {}

  void HelloWorld1::analyze(const art::Event& event){

    mf::LogVerbatim("test") << "Hello, world. From analyze. " << event.id();

  }

}

using artex::HelloWorld1;
DEFINE_ART_MODULE(HelloWorld1)
```

An example config (FHICL) file

Note empty source

analyzers

module label and module_type

Run this with

```
[lyon@gm2gpvm01 ~]$ gm2 -c hello1.fcl
```

```
#include "minimalMessageService.fcl"
services.message: @local::default_message

process_name: helloWorld1

source: {
  module_type: EmptyEvent
  maxEvents:  2
}

physics: {

  analyzers: {

    hello: {
      module_type: HelloWorld1
    }

  }

  path1: [ hello ]
  end_paths: [ path1 ]

}
```

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();

// ...
```

Services must be in your FHICL

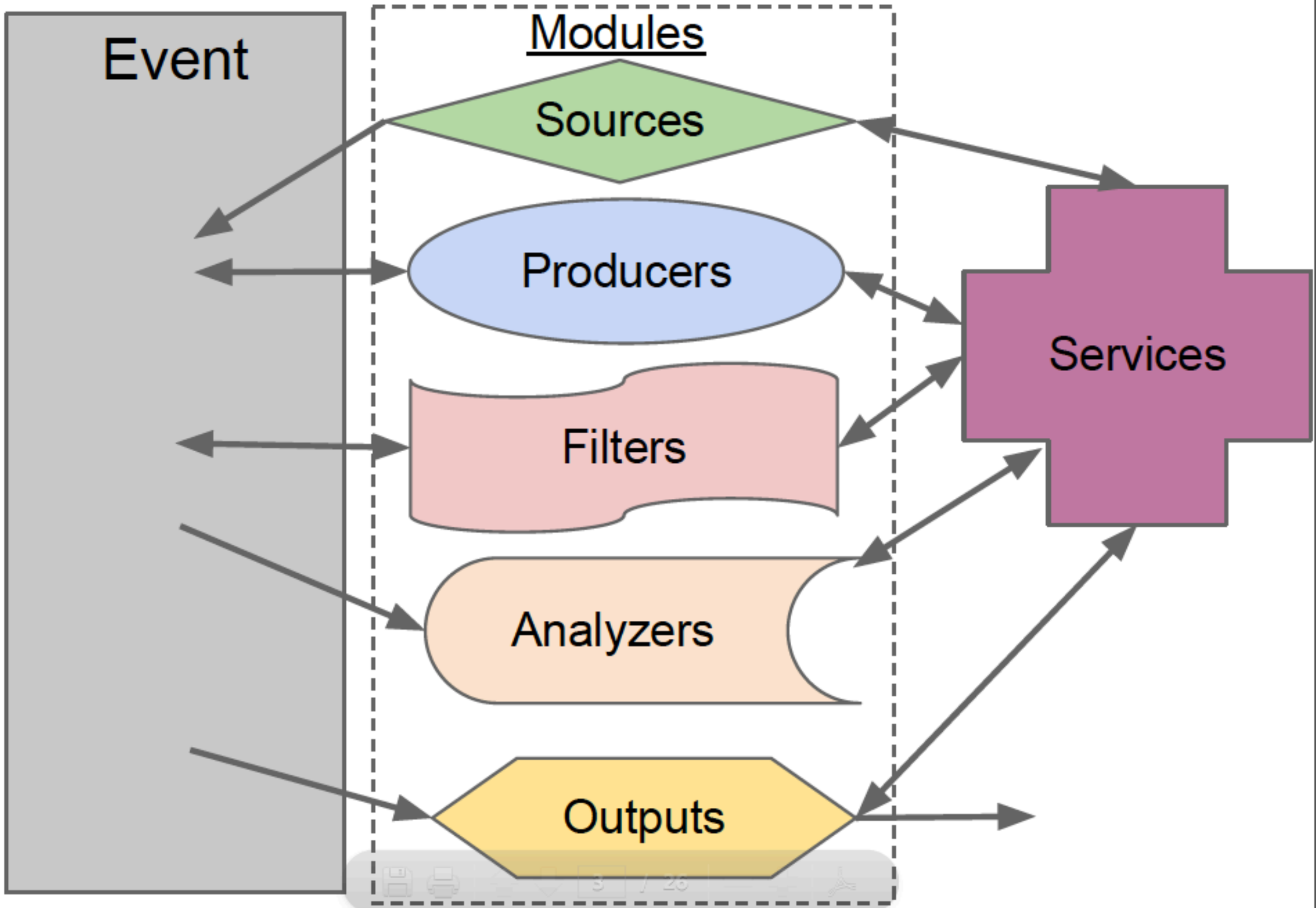
e.g. Gm2PhysicsList_service.cc

Build system creates
artg4example_Gm2PhysicsList_service.so

Specifying Gm2PhysicsList in FCL 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: {}

  }

}

# ...
```

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 FHICL 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

Write ARTG4

Summer student Tasha Arvanitis and I set out to work work on this project

Tasha: Sophomore at Harvey Mudd College [was also here as an IMSA High School student]

With some C++ experience, Tasha quickly learned ART and appreciated its capabilities

Took a "break" to compete in the Ultimate Frisbee World Championship in Dublin!!

Hinsdale girl competes in Ultimate in Dublin

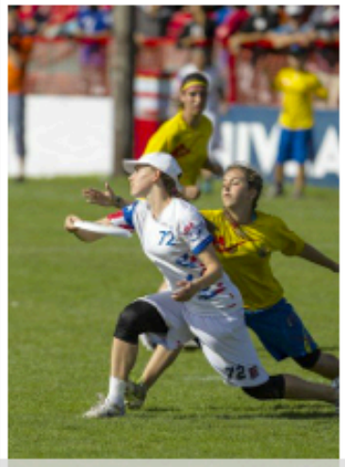
Natasha Arvanitis and 20-member US Junior Women's team took home silver

September 20, 2012 | By Graydon Megan, Special to the Tribune

Recommend 0 Tweet 0 Share 28 +1 0

As the 2012 Olympic Games were wrapping up in London, 18-year old Hinsdale resident Natasha Arvanitis was representing the United State in another international sports competition.

Arvanitis was part of the 20-member US Junior Women's team competing in Dublin, Ireland, in a sport called simply Ultimate. Arvanitis, who goes by Tasha, and her teammates won silver in the weeklong event, finishing second behind Colombia in the finals after beating Germany in the semi-finals.



Natasha Arvanitis of Hinsdale was p...

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"
```

The difficulty

Geant has it's own event loop and data store

Art wants to own the event loop and wants you to use its data store

We use code from Mu2e (thanks!) to break up the "Beam On" Geant4 process, allowing Art to control the Geant event loop (now part of Geant4)

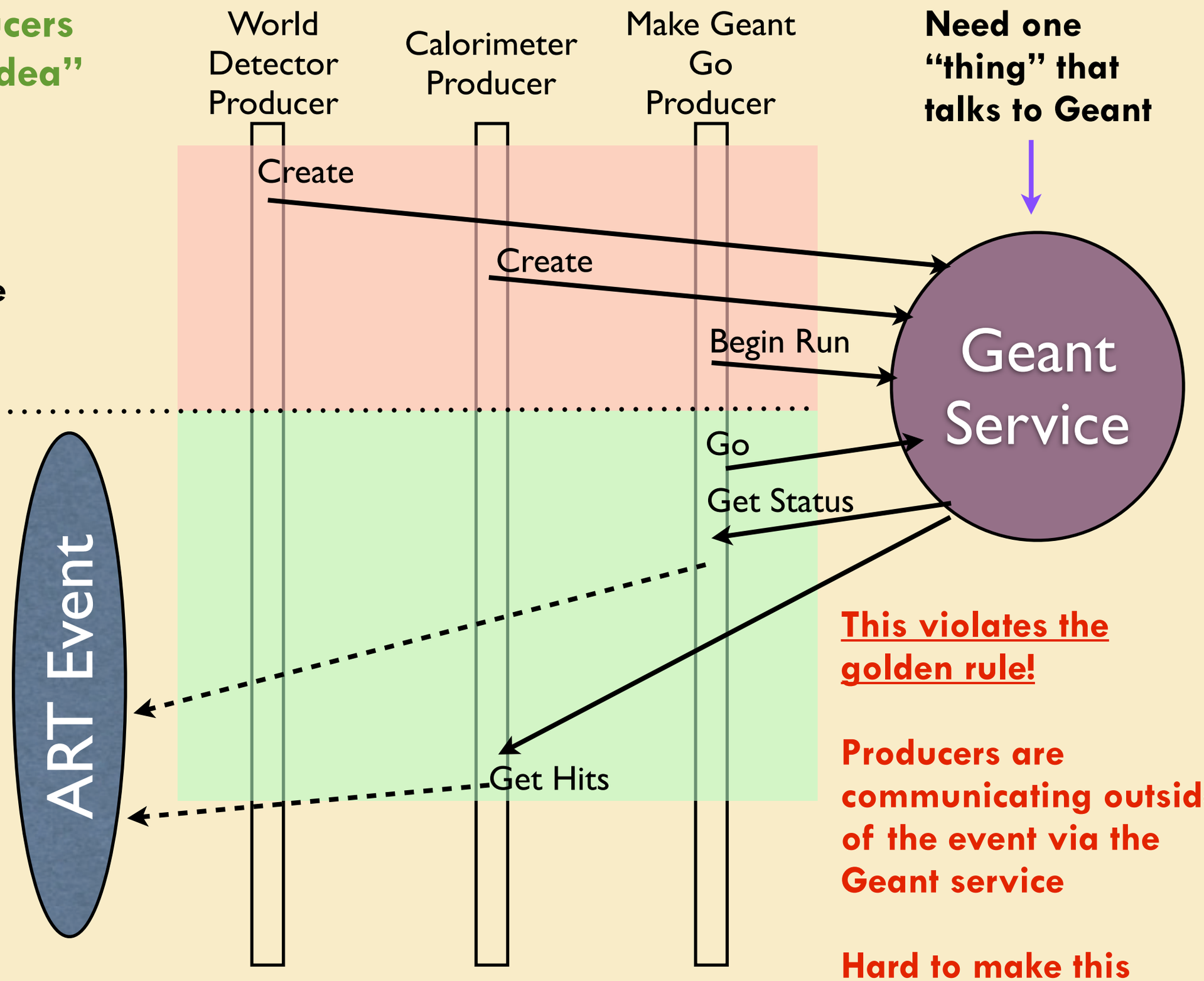
Several schemes were imagined

The "Detector Producers and Geant Service idea"

Begin Run:
Create detectors,
Tell Geant to initialize

Produce (event):
Tell Geant to go
Wait to complete

Get Geant hits
Put into Art event



This violates the golden rule!

Producers are communicating outside of the event via the Geant service

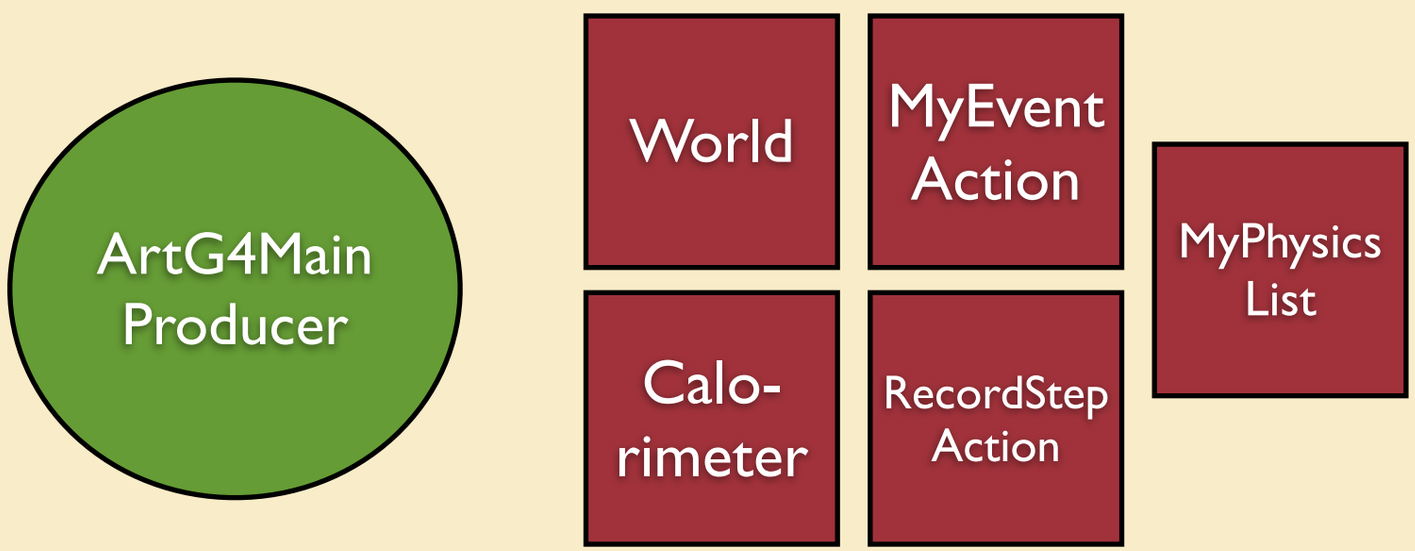
Hard to make this multiprocess-able

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

Every detector must have name, category, mother category

Individual detector services must inherit from DetectorBase and must override

```
// Build the detector logical volumes and return them
virtual std::vector<G4LogicalVolume*> doBuildLVs() = 0;

// Place the detectors within the passed in mother logical volumes and return
// the resulting physical volumes
virtual std::vector<G4VPhysicalVolume*> doPlaceToPVs(
                                std::vector<G4LogicalVolume*> motherLV ) = 0;
```

Why return vectors? Because we typically make many (nearly) identical copies of the same detector (e.g. 12 arcs, 24 calorimeters).

If a detector makes data (e.g. stores hits), then override

```
// Tell Art what you will put into the event.
virtual void doCallArtProduces(art::EDProducer *) {}

// Convert G4 hits into Art hits. Put them in the event (which you can get
// from the DetectorHolder service).
virtual void doFillEventWithArtHits(G4HCofThisEvent*) {}
```

Example MuonDetector_service.cc

From the Novice 04 GEANT Example

```
artg4example::ExampleMuonDetectorService::
ExampleMuonDetectorService(fhicl::ParameterSet const & p,
                           art::ActivityRegistry &)
: DetectorBase(p,
               p.get<string>("name","exampleMuon"),
               p.get<string>("category","exampleMuon"),
               p.get<string>("mother_category","world")),
box_radius_(p.get<double>("box_radius")*cm),
width_(p.get<double>("width")*cm),
thick_(p.get<double>("thick")*cm),
length_(p.get<double>("length")*cm),
noMuonCounter_(p.get<int>("n_muon_counters")),
logInfo_("ExampleMuonDetector")
{}
```

Registration with DetectorHolder is done behind the scenes

```
void artg4example::ExampleMuonDetectorService::doCallArtProduces(
                                                art::EDProducer * producer)
{
   producer -> produces<MuonArtHitDataCollection>(myName());
}
```

Example MuonDetector_service.cc

```
vector<G4LogicalVolume *> artg4example::ExampleMuonDetectorService::doBuildLVs()
{
  G4VSolid * muoncounter_box  = new G4Box("muoncounter_box", width_, thick_, length_);

  G4LogicalVolume * muoncounter_log  = new G4LogicalVolume(muoncounter_box,
                              artg4Materials::BC404Scintillator(),"mucounter_L",0,0,0);

  G4VisAttributes* muoncounter_logVisAtt = new G4VisAttributes(G4Colour(0.0,1.0,1.0));
  muoncounter_logVisAtt->SetForceWireframe(true);
  muoncounter_log->SetVisAttributes(muoncounter_logVisAtt);

  G4SDManager* SDman = G4SDManager::GetSDMpointer();
  ExN04MuonSD * muonSD = new ExN04MuonSD("/mydet/muon");
  SDman->AddNewDetector(muonSD);
  muoncounter_log->SetSensitiveDetector(muonSD);

  vector<G4LogicalVolume *> myLVvec;
  myLVvec.push_back(muoncounter_log);

  return myLVvec;
}
```

```
vector<G4VPhysicalVolume *> artg4example::ExampleMuonDetectorService::
    doPlaceToPVs(vector<G4LogicalVolume *> motherLVs)
{

  vector<G4VPhysicalVolume *> myPVvec;

  for(int i=0; i<noMuonCounter_ ; i++)
  {
    G4double phi, x, y, z;
    phi = 360.*deg/noMuonCounter_*i;
    x = box_radius_*std::sin(phi);
    y = box_radius_*std::cos(phi);
    z = 0.*cm;
    G4RotationMatrix rm;
    rm.rotateZ(phi);
    myPVvec.push_back(new G4PVPlacement(G4Transform3D(rm,
                                        G4ThreeVector(x,y,z)),
                                        lvs()[0],
                                        "muoncounter_P",
                                        motherLVs[0],
                                        false,
                                        i));
  }

  return myPVvec;
}
```

Example MuonDetector_service.cc

```
void artg4example::ExampleMuonDetectorService::doFillEventWithArtHits(
                                                    G4HCofThisEvent * myHC)
{

  std::unique_ptr<MuonArtHitDataCollection> myArtHits(new MuonArtHitDataCollection);

  G4SDManager* fSDM = G4SDManager::GetSDMpointer();
    G4int collectionID = fSDM->GetCollectionID("muonCollection");

  ExN04MuonHitsCollection* myCollection =
      (ExN04MuonHitsCollection*)(myHC->GetHC(collectionID));

  if (NULL != myCollection) {
    vector<ExN04MuonHit*> geantHits = *(myCollection->GetVector());

    for ( auto entry : geantHits ) {
            myArtHits->emplace_back(entry->GetPos(), entry->GetEdep());
    }
  }
  else {
    throw cet::exception("MUON") << "Null collection of Geant muon hits"
                                 << ", aborting!" << std::endl;
  }

  art::ServiceHandle<artg4::DetectorHolderService> detectorHolder;
  art::Event & e = detectorHolder -> getCurrArtEvent();

  e.put(std::move(myArtHits), myName());
}
```

Note that sensitive detector code was copied verbatim

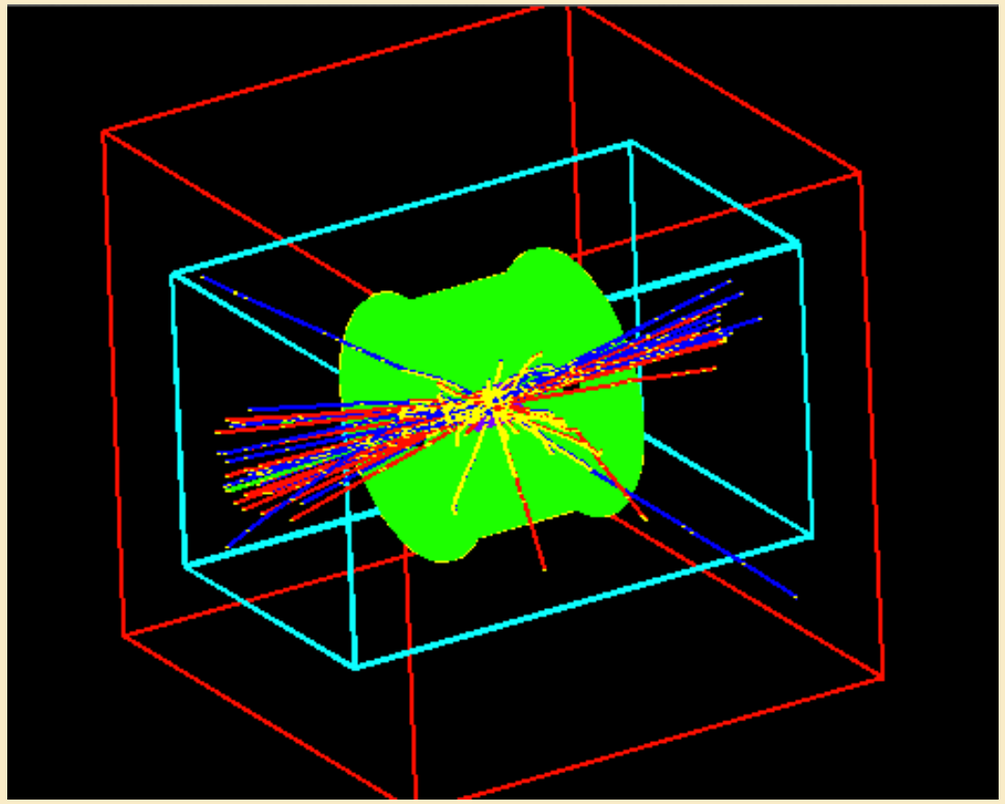
Example FHICL file

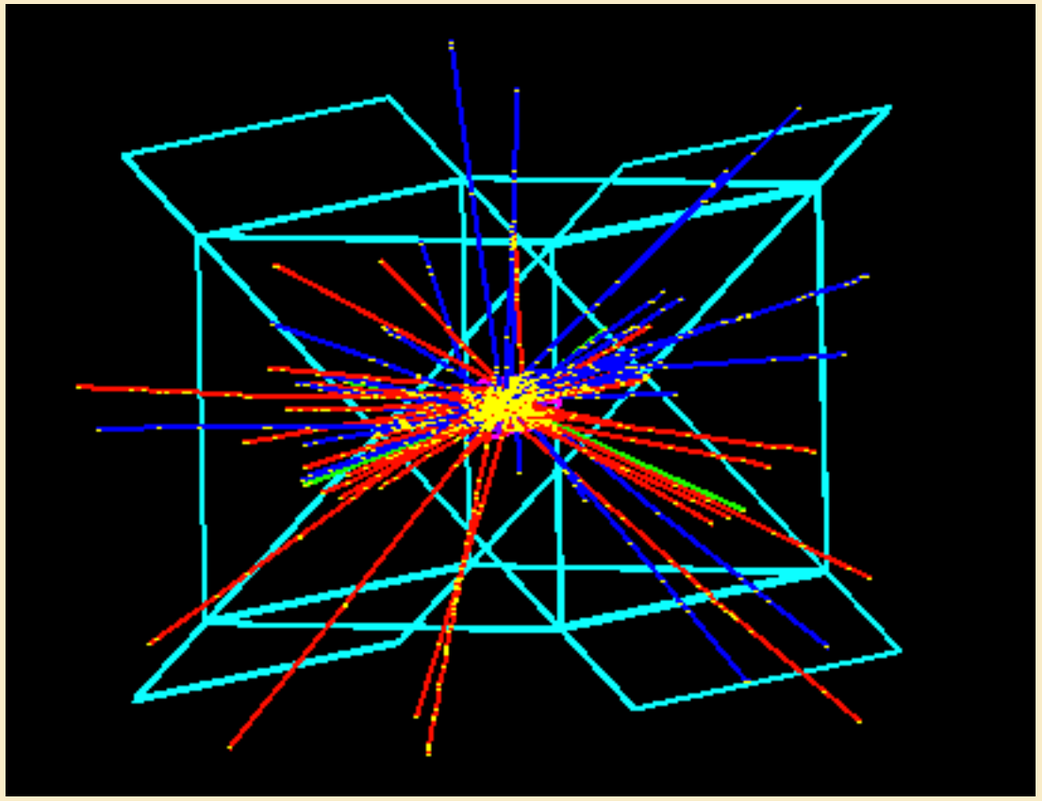
```
#include "detectorDefaults.fcl"

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 FHICL, 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

Special services

PhysicsListHolder_service/PhysicsList_service – manages physics lists

Geometry_service – manages geometry configuration for detectors (right now uses FHICL 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();
```

Utilities

```
// Set visual attributes
void setVisAtts(G4LogicalVolume* lv, bool display, const std::vector<double> & rgba);
void setVisAtts(G4LogicalVolume* lv, bool display, const std::vector<double> & rgba,
                std::function<void (G4VisAttributes*)> func );

// Put a number in a name
std::string addNumberToName(const std::string& name, int number);
```

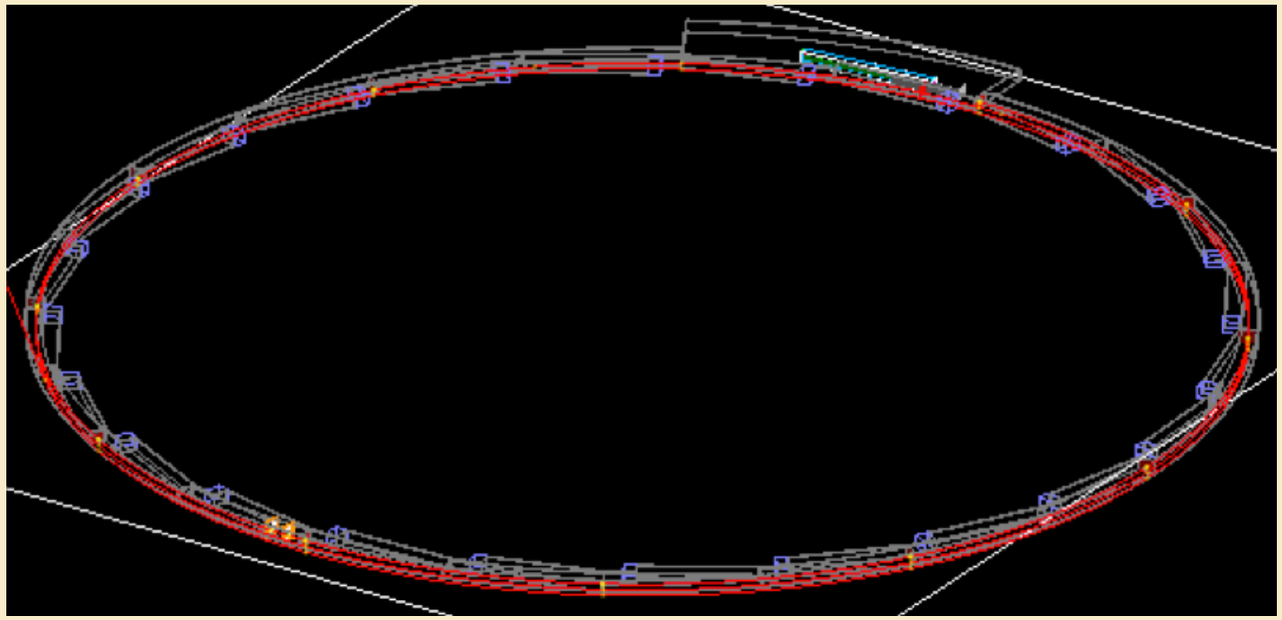
```
void gm2ringsim::VacuumChamber::makeWallLVs(const VacGeometry& g) {

  for ( unsigned int arcNum=0; arcNum != 12; ++arcNum) {

    G4UnionSolid* us = buildUnionSolid(g, g.wallRegion, arcNum);

    std::string wallName = artg4::addNumberToName("VacuumChamberWallLV", arcNum);

    G4LogicalVolume* wallLV = new G4LogicalVolume(
                                                  us,
                                                  artg4Materials::Al(),
                                                  wallName.c_str(),
                                                  0,
                                                  0);

    // Set the attributes (using a C++11 lambda function)
    artg4::setVisAtts( wallLV, g.displayWall, g.wallColor,
                       [] (G4VisAttributes* att) {
                               att->SetForceWireframe(1);
                               att->SetVisibility(1);
                       }
    );

    wallLVs_.push_back(wallLV);

  }
}
```

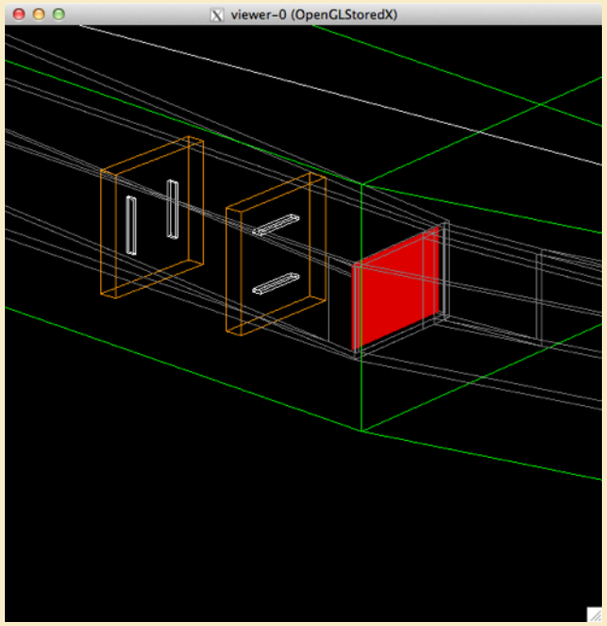
gm2ringsim

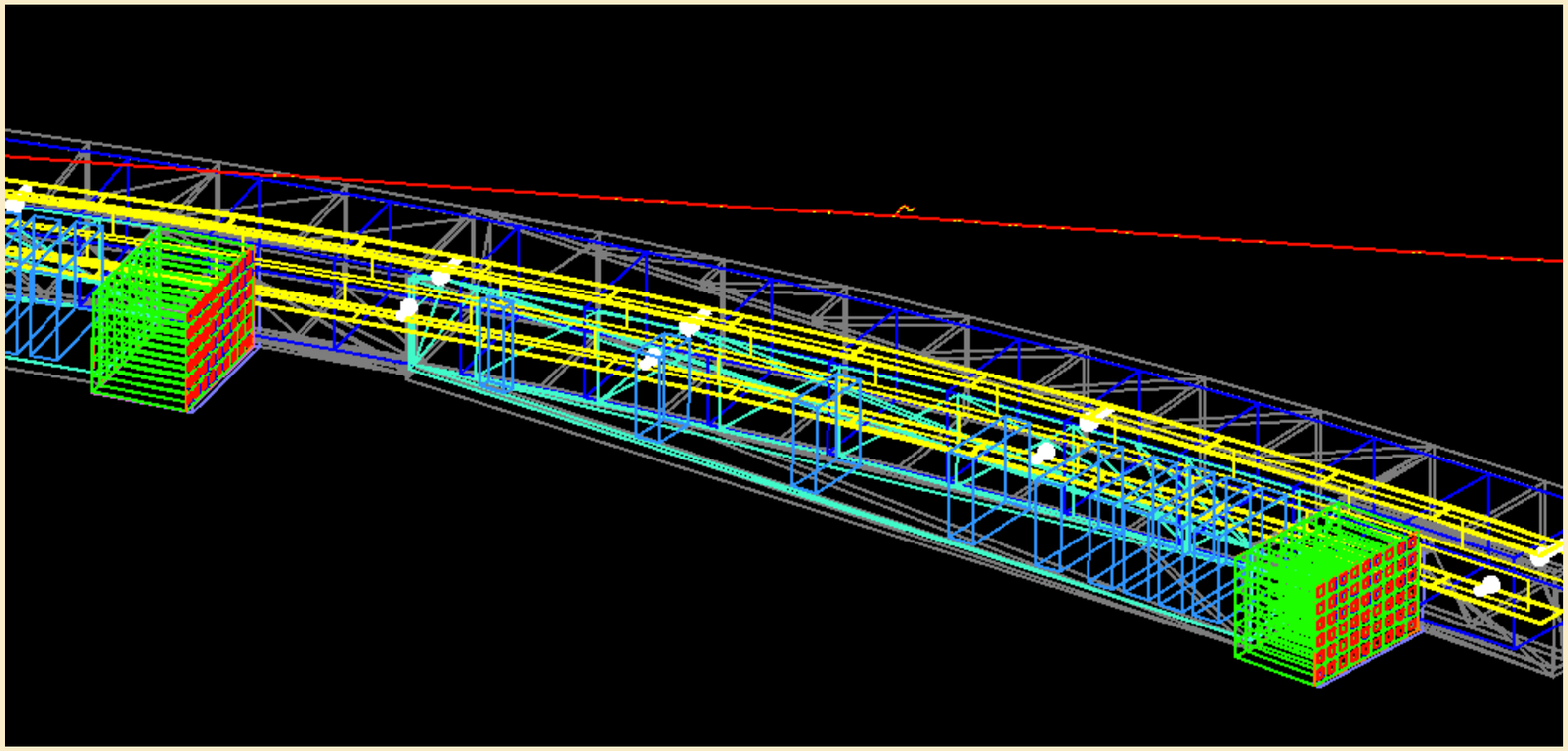
Started 10/12

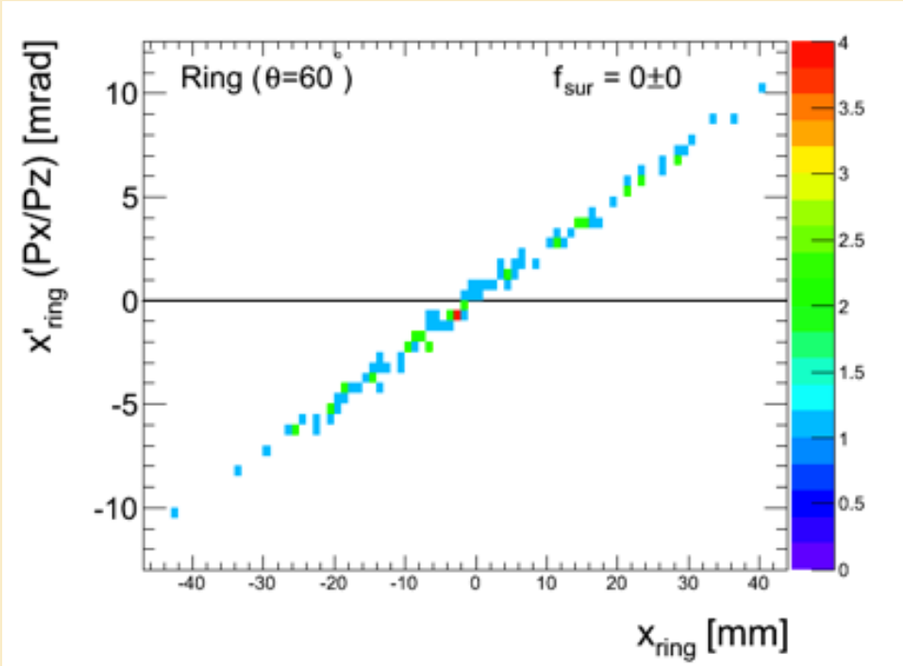
2.5 months with 5 active people

Now have many more analyzing



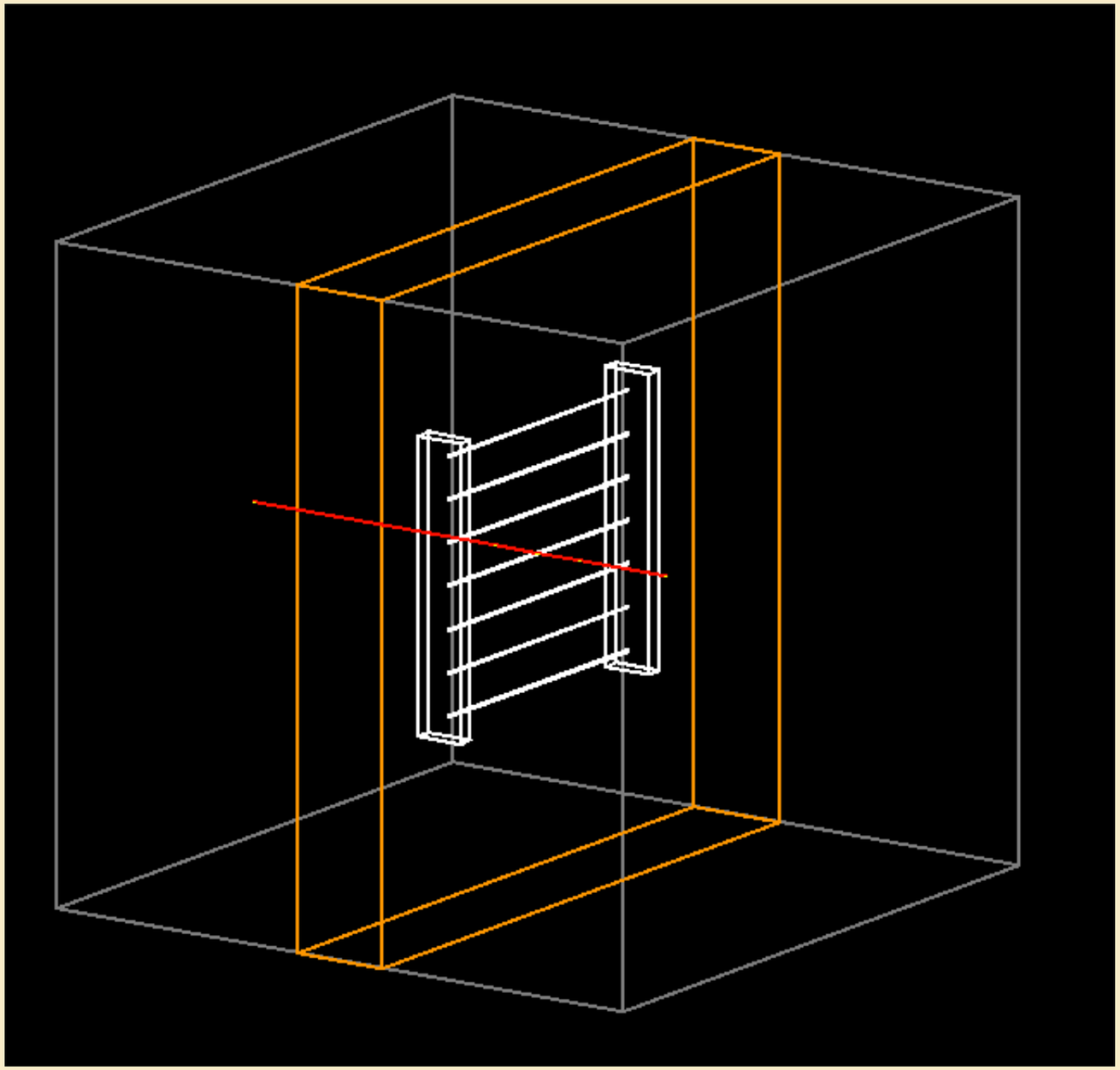






A “Test Beam” Simulation

```
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)

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)

Current Muon g-2 status

Achieved CD0 2012

MC-1 Site preparation underway

Ring move to commence in next few months

CD1 Review this spring

Start taking data in FY16

Some g-2 numbers

Cyclotron freq is 6.7 MHz or 149 ns per rotation

~29 cyclotron rotations per one omega_a rotation
g-2 freq is 229 KHz or 4.3 microsec

Field is 1.4513 T