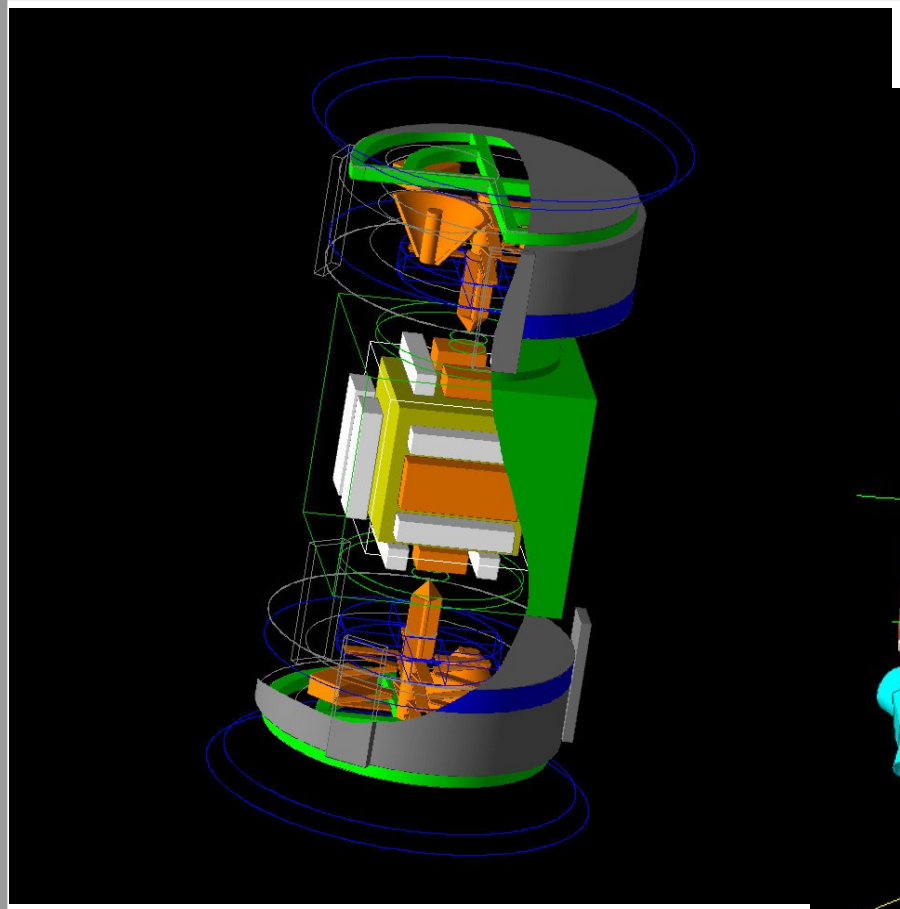
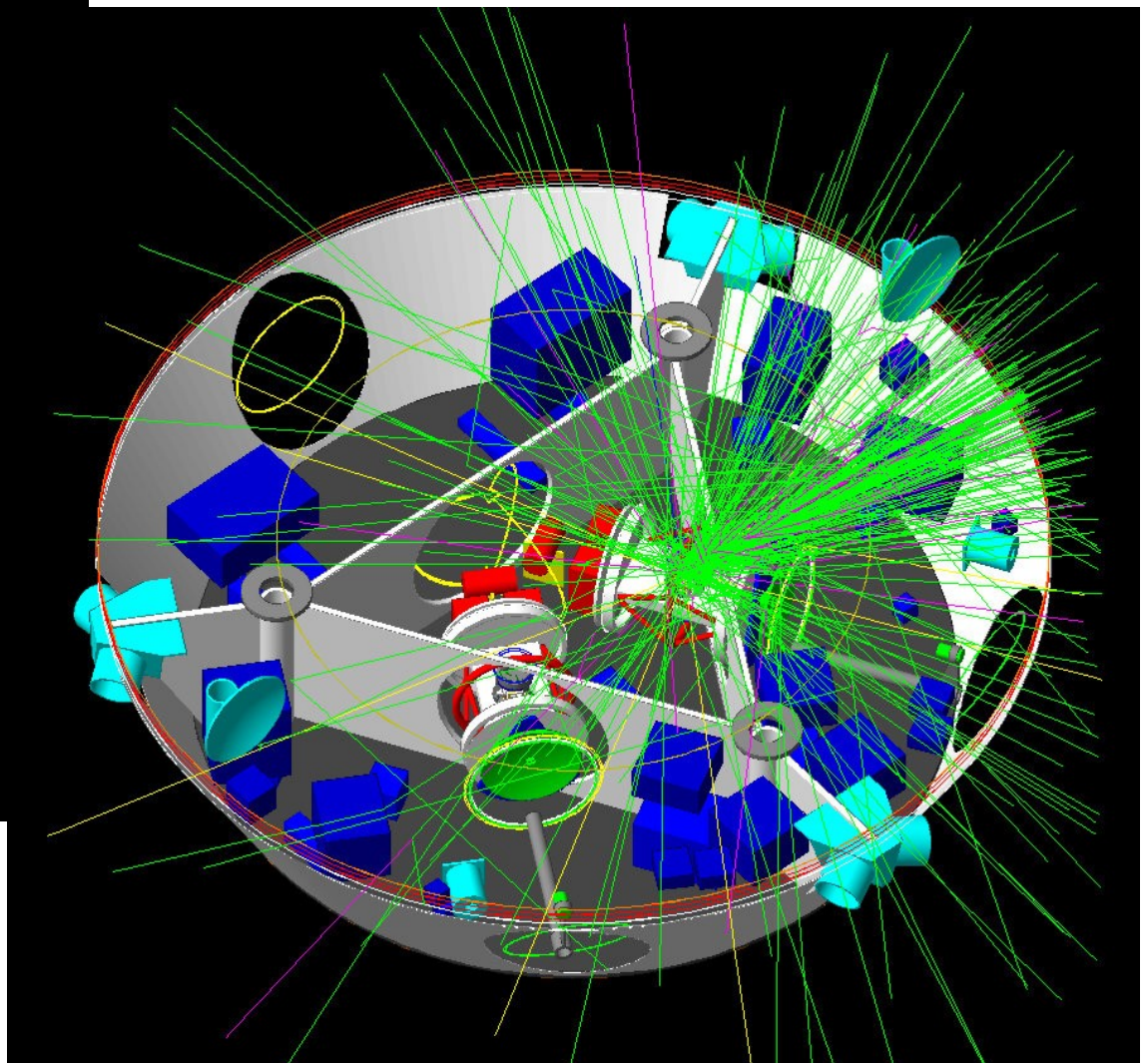
GEANT 4 Introductory Course



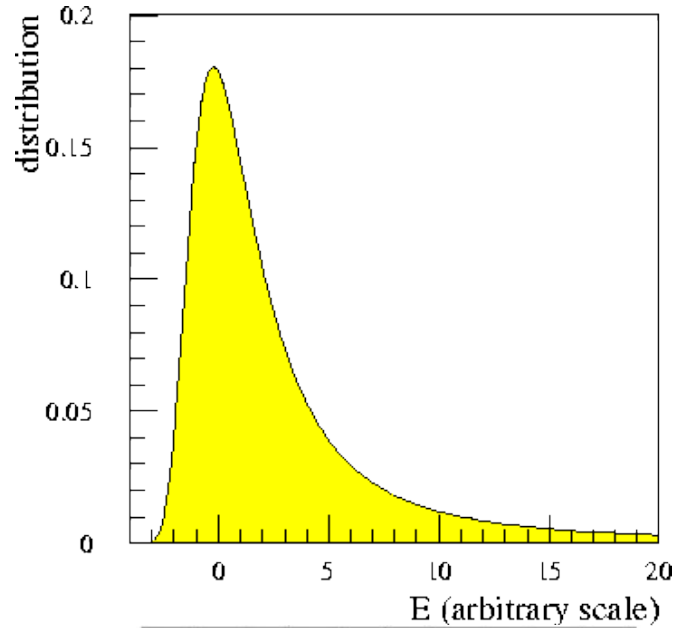


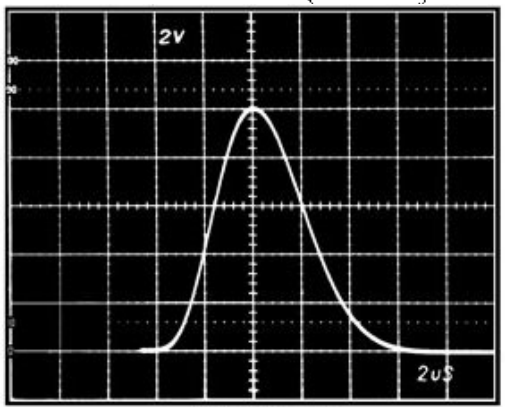
Outline

- Why the Monte-Carlo approach
- Basic GEANT4 concepts
- A quick reminder of advanced C++
- Defining materials and volumes
- Defining detectors
- Example: A trigger counter
- Example: Raytracing in GEANT4

Why the Monte-Carlo approach

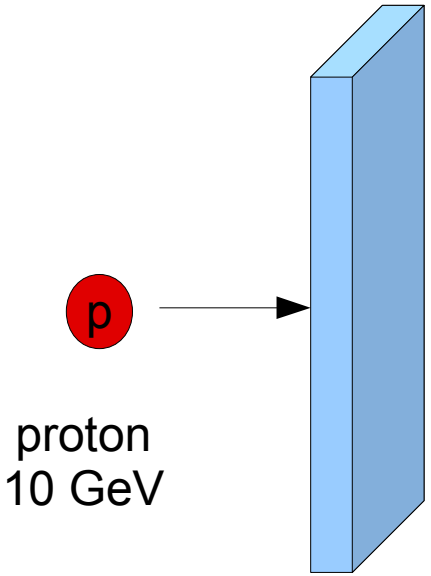
- High-Energy Physics Detectors
  - a particle interacts with the detector material, depositing energy (e.g. ionization)
    - e.g. Landau distributed
  - energy deposit → analog electrical signal
    - noise, non-linearities of amplifiers, other statistical effects
  - analog signal → digital signal
    - noise, limited dynamic range

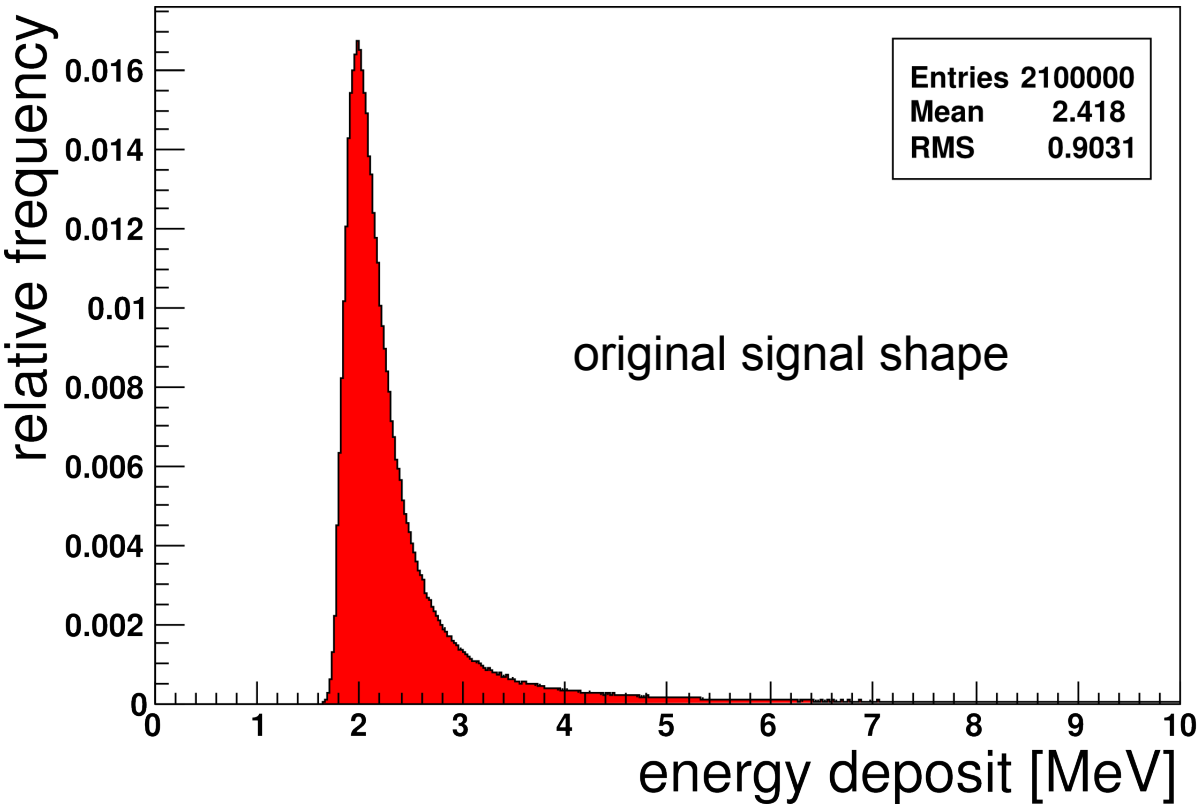




semi-gaussian amplifier pulse

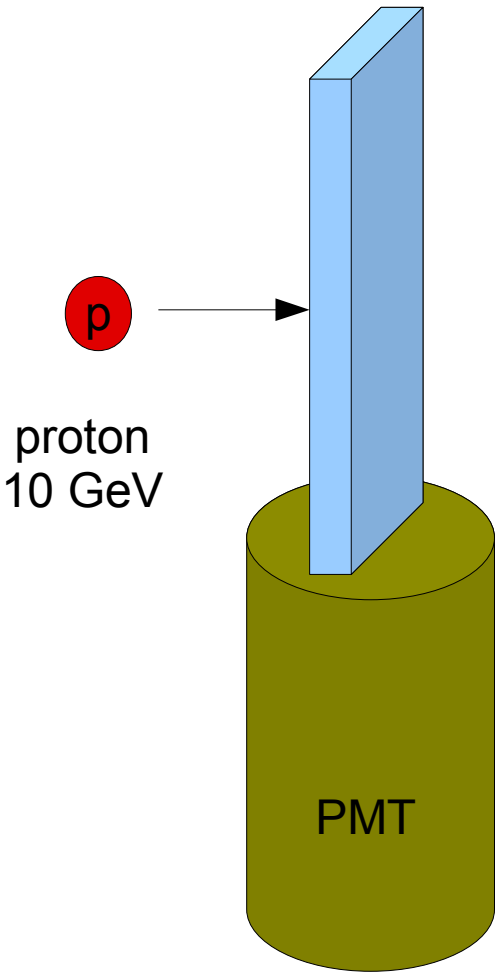
Simple Particle Detector: Scintillator Counter

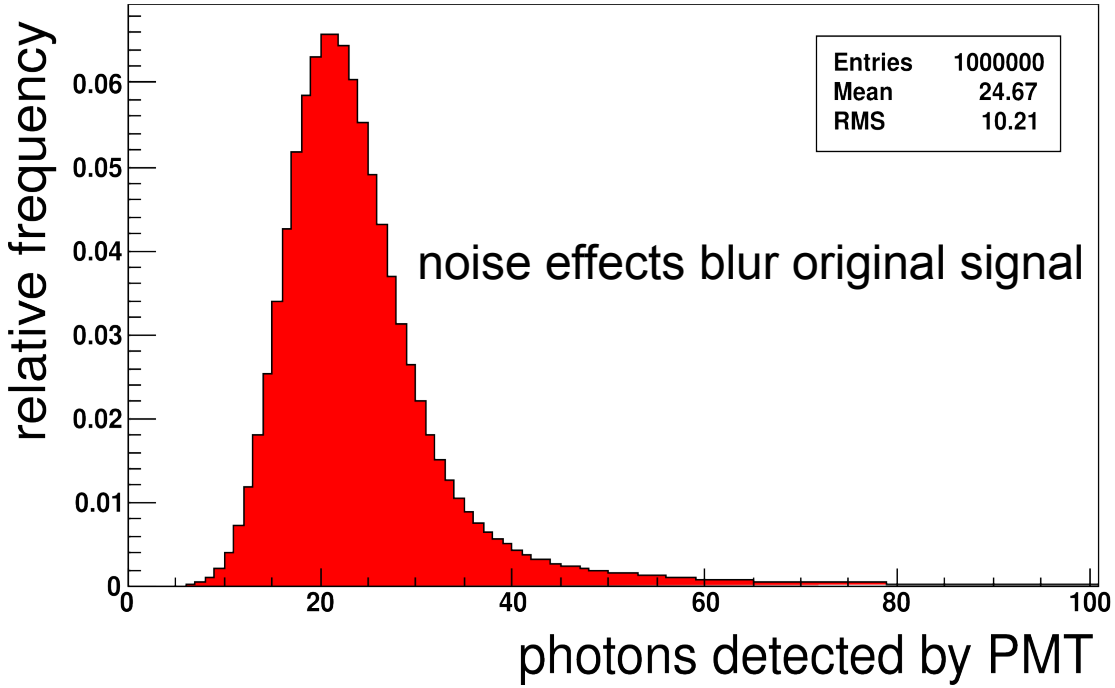




- Consider a proton passing through a 10mm thick piece of scintillator
- Energy deposit is Landau distributed

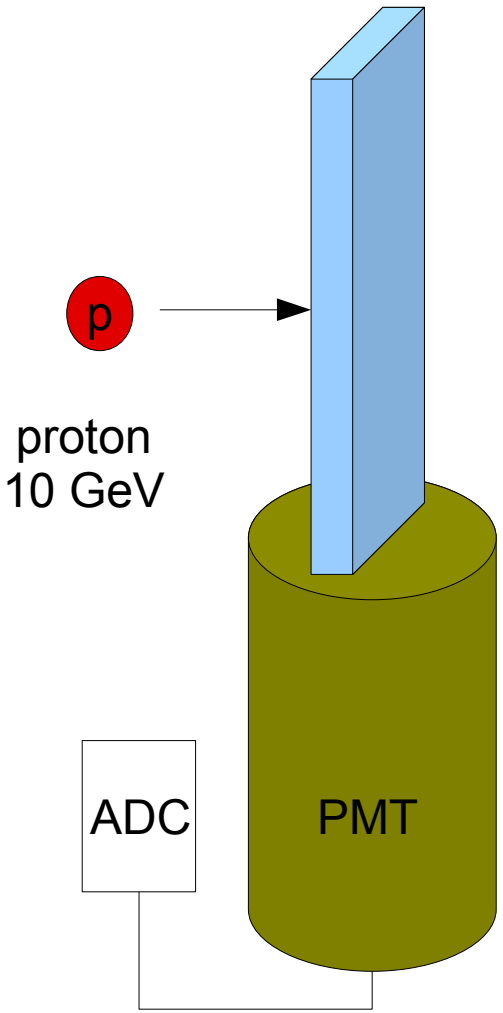
Simple Particle Detector: Scintillator Counter

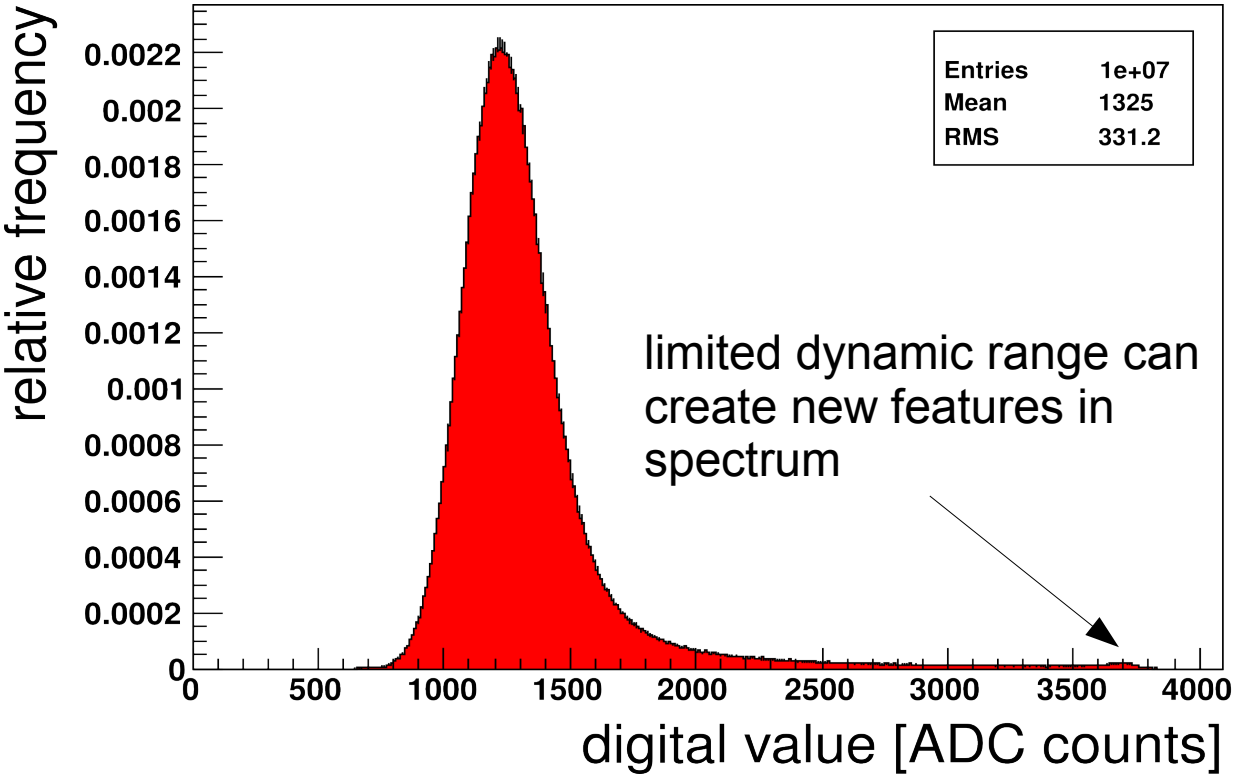




- ~25 photons detected by photomultiplier tube
- Landau shape now convoluted with Poisson distribution

Simple Physics Detector: Scintillator Counter





- signal shape can be calculated convoluting half a dozen functions
- or using a Monte-Carlo method

Let's add some Complexity

- What if we vary the angle of incidence?
- Instead of one channel we have to consider between 10,000 channels (e.g. miniPEBS) and 10,000,000 channels (e.g. ATLAS, CMS)
- We no longer have on physical process to worry about but several
  - ionization energy loss
  - cherekov radiation
  - bending of track in magnetic field
  - multiple scattering
  - ...

The Advantages of Monte-Carlo-Methods

- We can break down a complex problem into many simple problems that are treated separately for a random initial state
  - break down a long particle track into many small steps
  - look at each detector component and each particle seperately
  - treat physics processes consecutively instead of having to worry about everything at once
- The output of our simulated detector is qualitatively the same as of the real detector

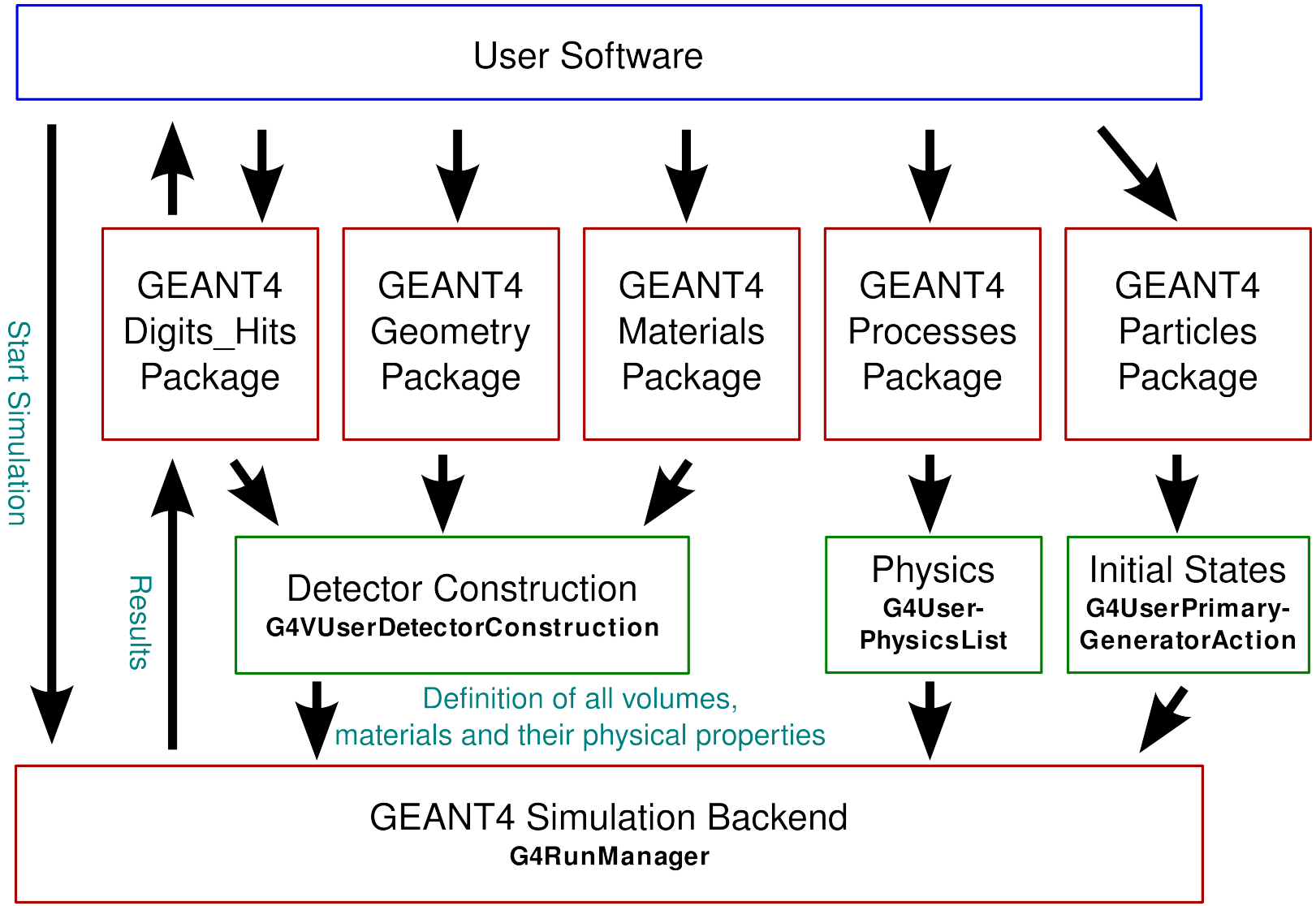
BUT BEWARE

- A simulation will only produce what you put it
  - A simulation will help you understand your detector but it's a tool without scientific value by itself
- Simulated detectors are based to a large part on „Educated Guesses“
  - Use set of parameters to be able to adjust your detector to match measurements.
- Actually simulating all physical processes accurately is not possible
  - You have to make a lot of approximations: This WILL distort your results.

Basic GEANT4 Concepts

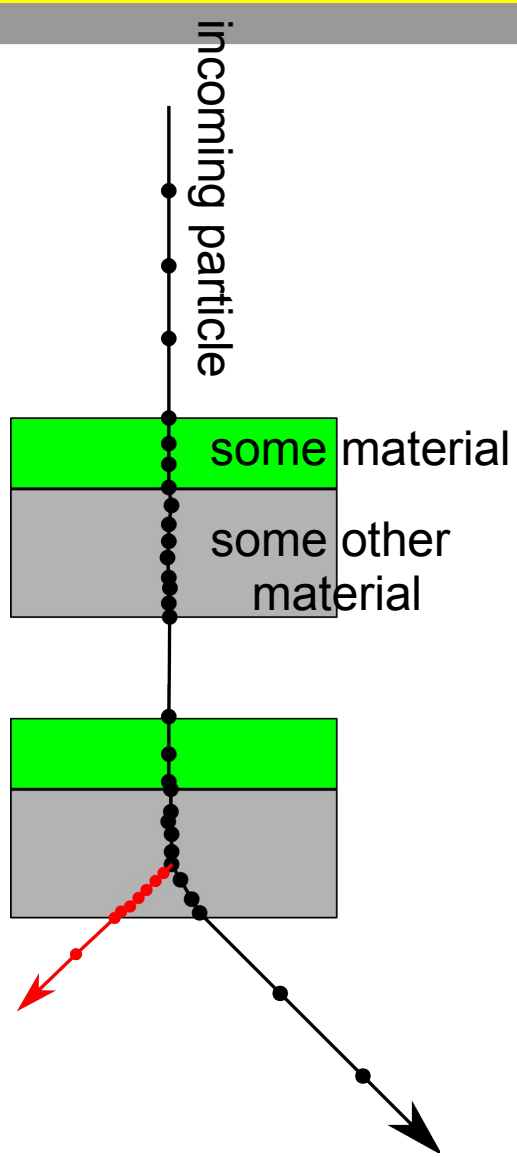
- GEANT4 is a framework with an interface for integration in C++ programs
- It does not offer a command-line interface or a GUI to define the geometry
- Every simulation is a compilable C++ program
- The user supplies the geometry, the list of physics processes, the initial states
- GEANT4 supplies the information about particles passing through user-defined inspection points called detectors

Basic GEANT4 Concepts



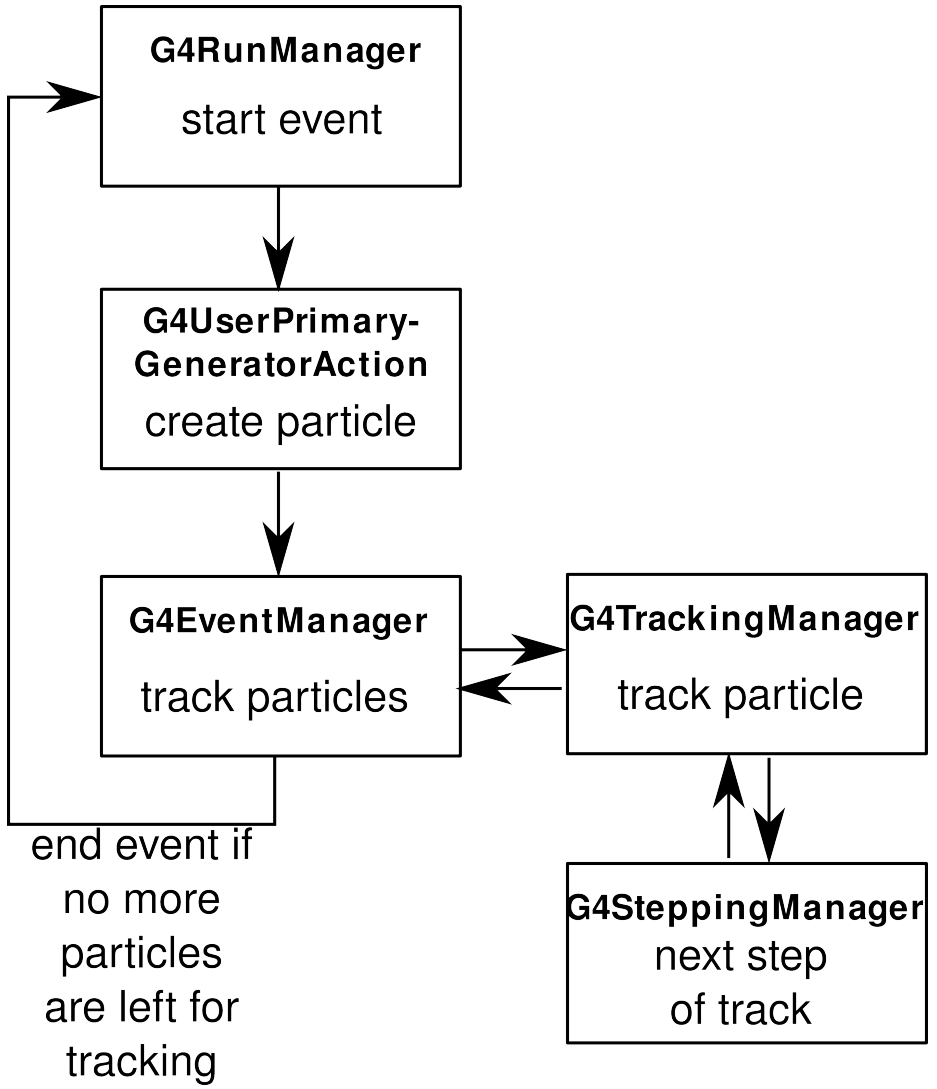
Under the hood

- break down particle track into small steps
  - store particle momentum, positions before and after last step, polarization...
- apply processes between steps
  - processes may modify particle momentum, direction, polarization and/or create new particles
  - processes may also modify current volume (e.g. store an energy deposit)
- newly created particles are also tracked



Event Flow

- Each event starts with a single created particle
- trakcing initial particle and all secondaries
- After each event data for the particle trajectories can be accessed
- GEANT4 allows intervening at any given point of the simulation

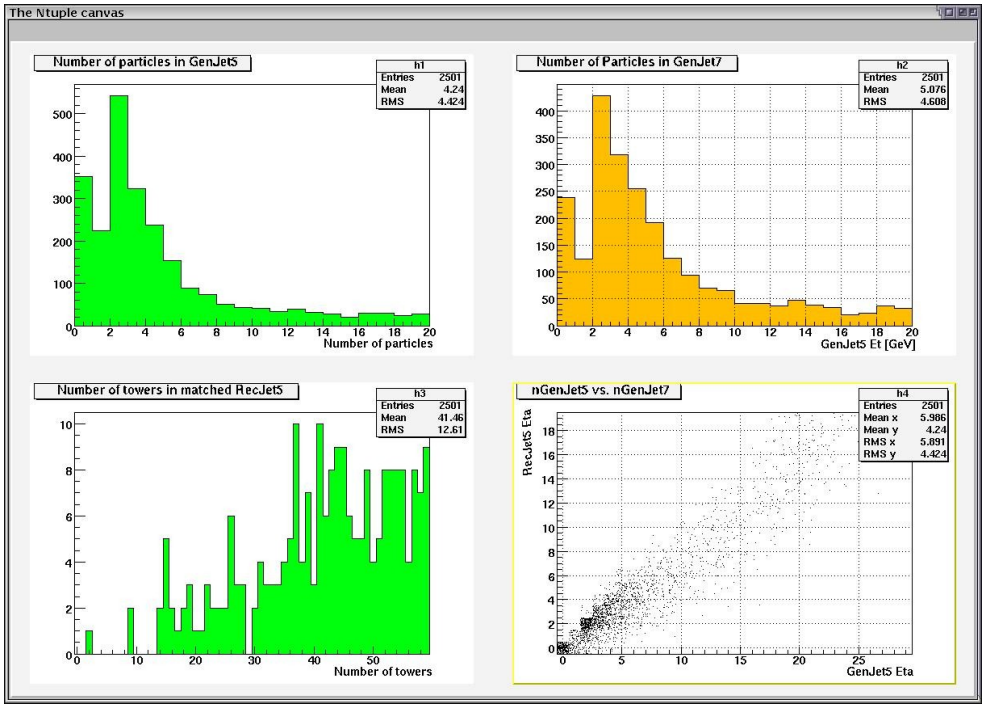


A quick Reminder of advanced C++

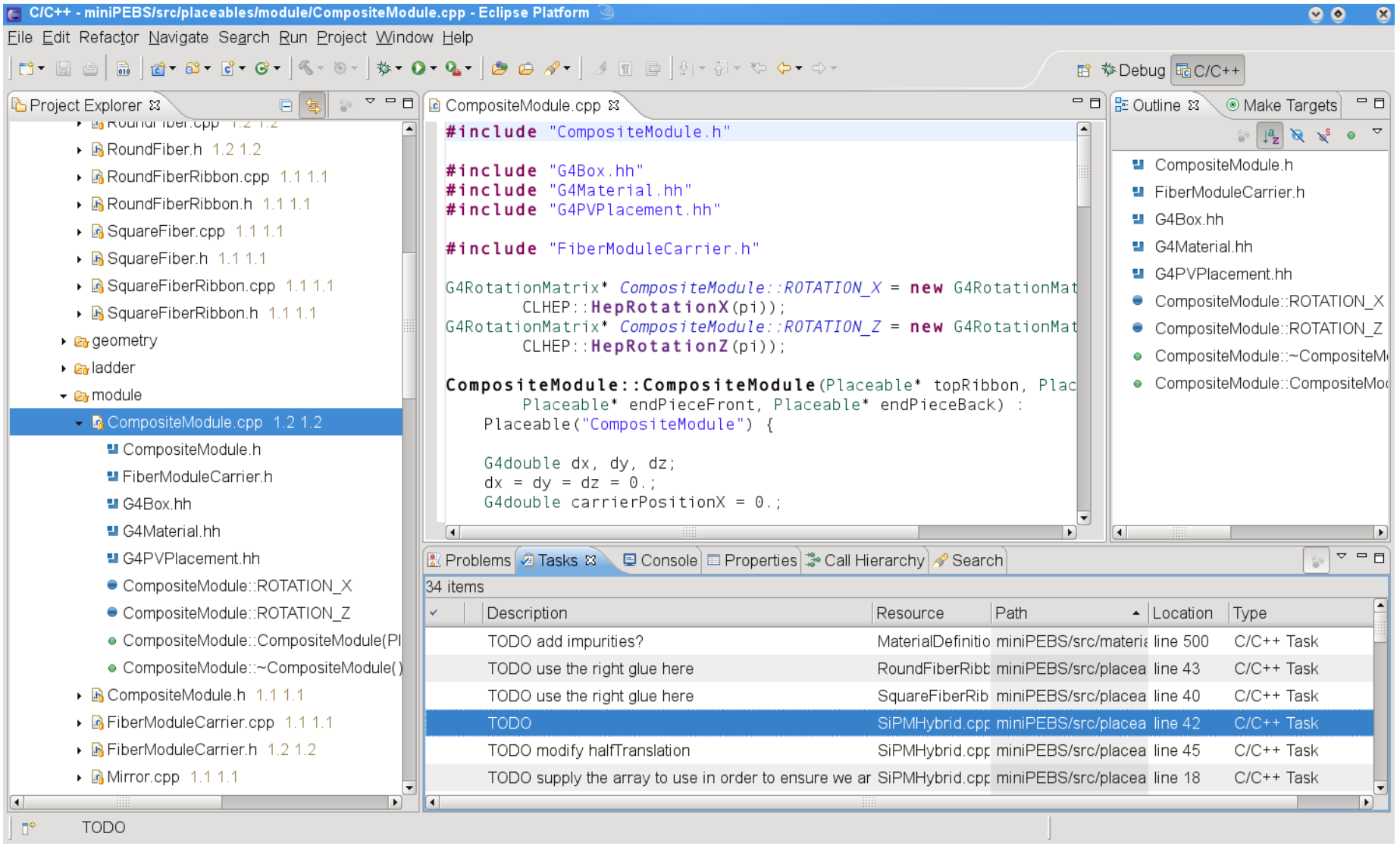
- C++ allows for object-oriented software development and that is what we SHOULD do!
  - this helps breaking down a software into convenient units called classes
  - each component in your detector SHOULD be a class!
- C++ provides a number of useful libraries
  - STL (map, vector, ...) for storing objects + useful algorithms for sorting, searching and copying objects
  - IOStream (fstream, sstream, ...) for reading/writing data
  - String (string) for character strings

Then there is ROOT

- Data produced by GEANT4 SHOULD be analyzed using ROOT
- The data format of your GEANT4 output SHOULD match the one of the real detector
  - you can write your readout/analysis software before your actual detector is working!

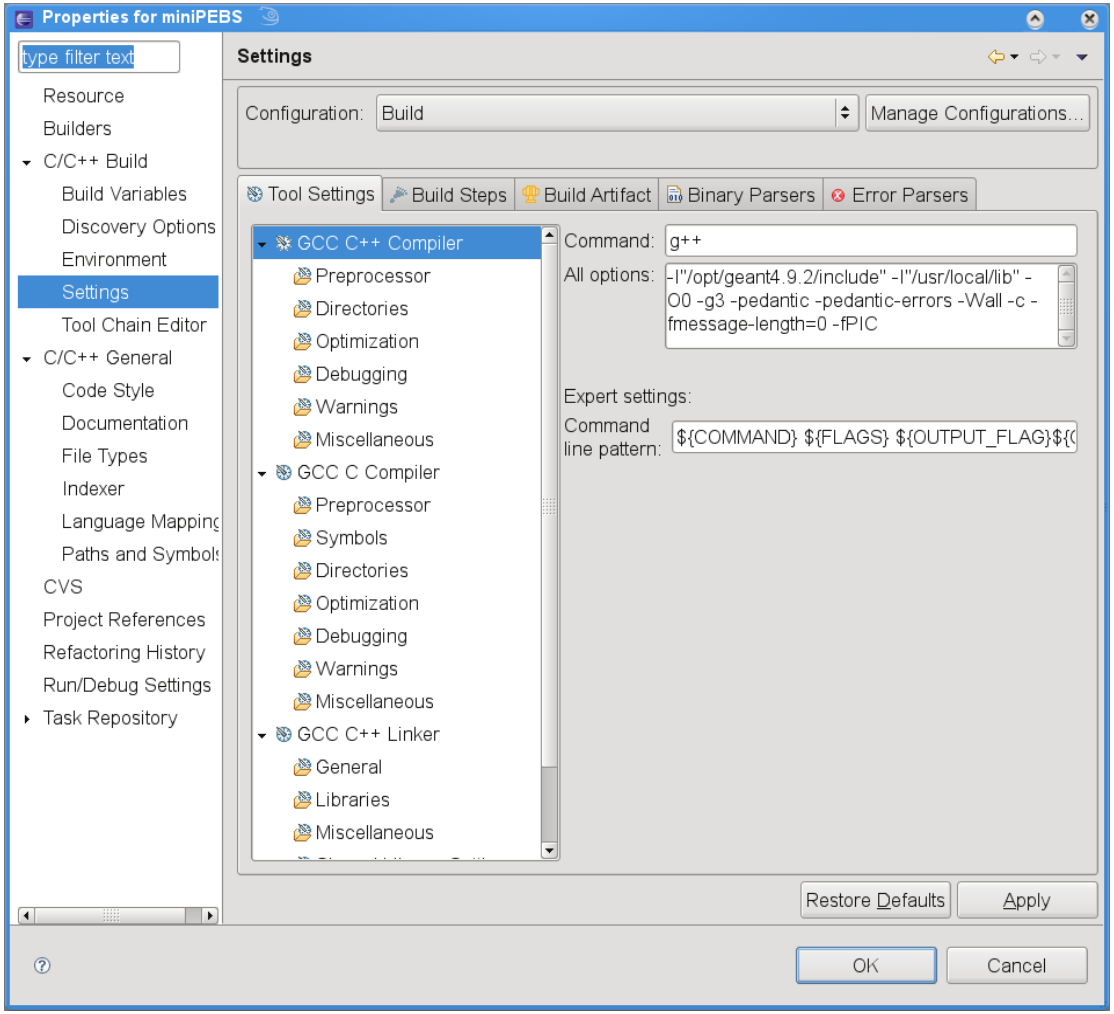


Use an Integrated Development Environment



Using eclipse with GEANT4

- eclipse is an IDE
  - editor with syntax highlighting and syntax checker
  - advanced code browsing
- CVS integration
- build process is entirely handled by eclipse
  - no more Makefiles



Setting up GEANT4

- Download & Install CLHEP (version 2.0.4.2)
  - http://proj-clhep.web.cern.ch/proj-clhep/DISTRIBUTION/
- Download & Install ROOT
  - http://root.cern.ch/drupal/content/development-version-523
- Download & Install GEANT4 (+ data files)
  - http://geant4.web.cern.ch/geant4/support/download.shtml
  - GEANT4 requires a lot of options when installing – see next slide
- Download & Install eclipse Ganymede for C/C++
  - http://www.eclipse.org/downloads/
  - (or install eclipse CDT extension in your version of eclipse)

GEANT4 Install options

- Run „Configure -build“ and set the following options (for all others you can use the default)
  - Do you want to copy all Geant4 headers in one directory? [Y]
  - Do you want to build 'shared' (.so) libraries? [Y]
  - Do you want to build 'global' compound libraries? [Y]
  - Do you want to build 'granular' libraries too? [N]
  - Do you want to compile libraries in DEBUG mode (-g)? [Y] (you may switch this to NO later if your software runs to slowly)
  - Set G4UI_NONE/G4VIS_NONE environment variable? [N]
  - G4VIS_BUILD_OPENGLX_DRIVER / G4VIS_USE_OPENGLX [Y]
- Run „Configure -install“

Setting your system with GEANT4

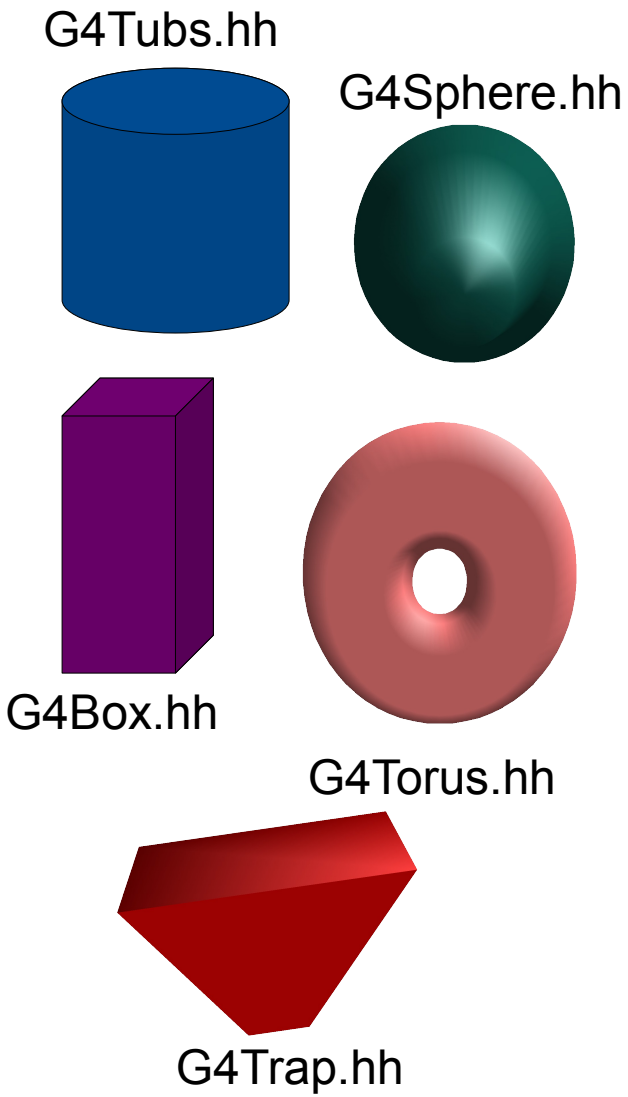
- Set up your environment variables (e.g. adding the following lines to your ~/.bashrc file)
  - source /path/to/geant4/.config/bin/Linux-g++/env.sh
  - export ROOTSYS=/usr/local
  - export LD_LIBRARY_PATH=$LD_LIBRARY_PATH: $ROOTSYS/lib

Setting up eclipse for GEANT4

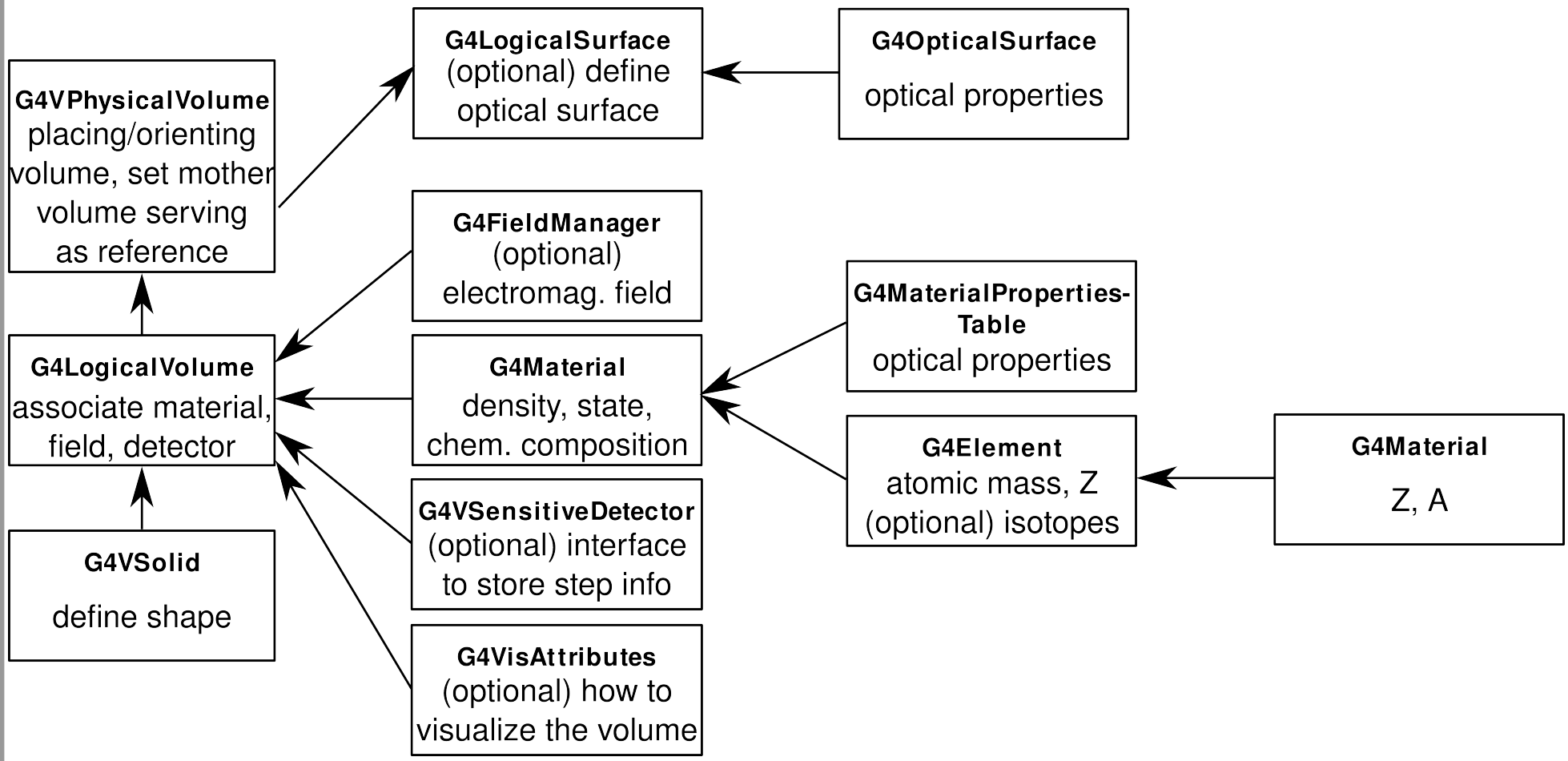
- eclipse requires list of GEANT4 libraries
  - this is very tedious if you are using granular libraries (GEANT4 then consists of 20+ libraries)
  - all eclipse options for GEANT4 (and ROOT) are set in the „geant4-sample“ project in the CVS
    - in eclipse go to File→New→Project...
    - choose „CVS→Projects From CVS“, Next...
    - Choose „Create a new repository location“
    - Host: portal.physik.rwth-aachen.de
    - Repository Path: /.automount/home/home__home4/institut_1b/amsana/Offline/CVS
    - User / Pass (your cluster login); Connection Type „ext“, Next...
    - Use an existing module (will list available modules)
    - choose „geant4-sample“, Next..., Finish!

Defining Materials and Volumes

- The simulated detector geometry and materials are defined by a class extending G4VUserDetectorConstruction
- Use predefined volumes from GEANT4
- Each volume can have any number of sub-volumes of different materials
- You can create unions & intersections of volumes



GEANT4 Geometry Management



GEANT4: Defining Elements

```
void DetectorConstruction::defineElements() {
    // define Elements. They will be stored by GEANT4.
    new G4Element("Hydrogen", "H", 1, 1.008 * CLHEP::g / CLHEP::mole);
    new G4Element("Boron", "B", 5, 10.811 * CLHEP::g / CLHEP::mole);
    new G4Element("Carbon", "C", 6, 12.011 * CLHEP::g / CLHEP::mole);
    new G4Element("Oxygen", "O", 8, 15.9994 * CLHEP::g / CLHEP::mole);
    new G4Element("Sodium", "Na", 11, 22.98977 * CLHEP::g / CLHEP::mole);
    new G4Element("Aluminum", "Al", 13, 26.981538 * CLHEP::g / CLHEP::mole);
    new G4Element("Silicon", "Si", 14, 28.0855 * CLHEP::g / CLHEP::mole);
    new G4Element("Potassium", "K", 19, 39.0983 * CLHEP::g / CLHEP::mole);
}
```

name under which element is stored

Z

atomic weight

GEANT4: Defining Materials

number of elements

state of matter

name of material

density

```
void DetectorConstruction::defineMaterials() {
    // define Materials. They will be stored by GEANT4.
    G4Material* polystyrene = new G4Material("Polystyrene", 1.0320 * CLHEP::g
            / CLHEP::cm3, 2, kStateSolid);
    polystyrene->AddElement(G4Element::GetElement("Hydrogen", true), 8);
    polystyrene->AddElement(G4Element::GetElement("Carbon", true), 8);

    G4Material* borosilicateGlass = new G4Material("Borosilicate Glass", 2.23
            * CLHEP::g / CLHEP::cm3, 6, kStateSolid);
    borosilicateGlass->AddElement(G4Element::GetElement("Boron", true),
            0.040064);
    borosilicateGlass->AddElement(G4Element::GetElement("Oxygen", true),
            0.539562);
    borosilicateGlass->AddElement(G4Element::GetElement("Sodium", true),
            0.028191);
    borosilicateGlass->AddElement(G4Element::GetElement("Aluminum", true),
            0.011644);
    borosilicateGlass->AddElement(G4Element::GetElement("Silicon", true),
            0.377220);
    borosilicateGlass->AddElement(G4Element::GetElement("Potassium", true),
            0.003321);
}
```

atoms per unit

relative atomic abundance in material

element to add

GEANT4: Defining Volumes

```
G4VPhysicalVolume* DetectorConstruction::Construct() {
    // first create the world
    G4Box* world = new G4Box("World", 0.5 * CLHEP::m, 0.5 * CLHEP::m, 0.5
            * CLHEP::m);
    // logical volume defines the volume but not its placement in the world
    // Air is a predefined material, I was too lazy to define it
    _logicalWorld = new G4LogicalVolume(world, G4Material::GetMaterial("Air",
            true), "World");
    // physical volume places logical volume somewhere in the world, you can
    // place a logical volume multiple times
    _physicalWorld = new G4PVPlacement(0, G4ThreeVector, _logicalWorld,
            "World", 0, false, 0, true);

    // define scintillator 4cm X 40cm * 1 cm in size
    G4Box* scintillator = new G4Box("Scintillator", 2 * CLHEP::cm, 20
            * CLHEP::cm, 0.5 * CLHEP::cm);
    G4LogicalVolume* logicalScintillator = new G4LogicalVolume(scintillator,
            G4Material::GetMaterial("Polystyrene", true), "Scintillator");
    new G4PVPlacement(0, G4ThreeVector, logicalScintillator, "Scintillator",
            _logicalWorld, false, 0, true);

    return _physicalWorld;
}
```

define shape of volume
by sub-class of G4VSolid

return pointer to parent volume of whole detector

define physical volume, setting position and orientation

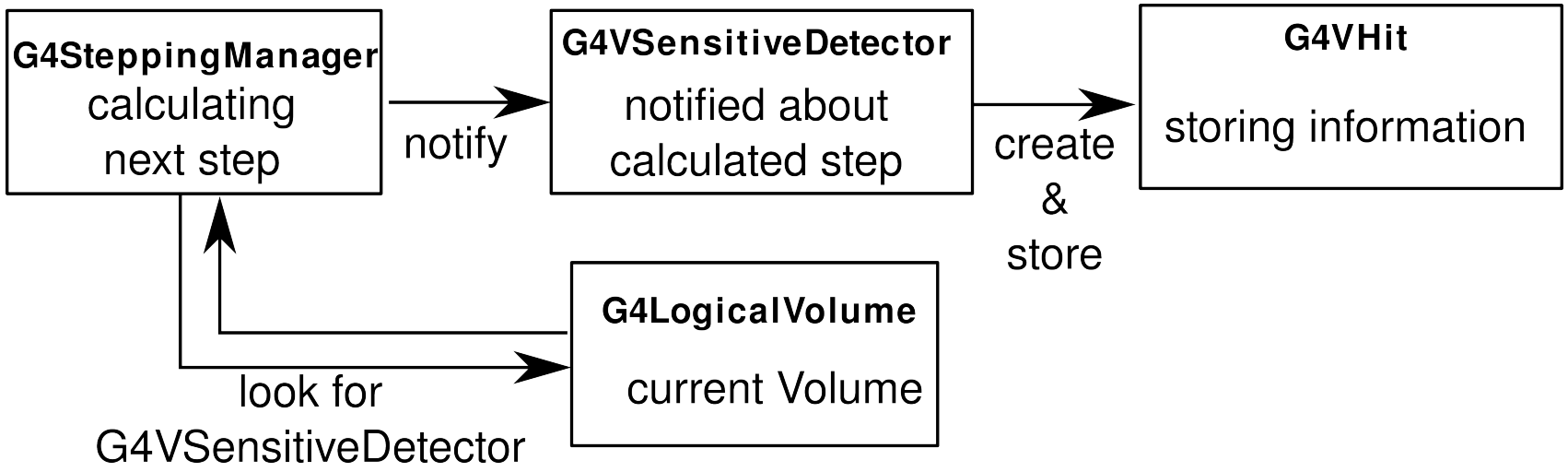
define logical volume, setting the material of the volume

Some words on Memory Management

- GEANT4 does not have any consistent memory management
- For every class that you instantiate you will ultimately have to check whether GEANT4 cleans it up or whether you have to do it
  - and no, it's not in the documentation
- handled by GEANT4
  - G4VSolids
  - G4VPhysicalVolumes
  - G4LogicalVolumes
  - ...
- handled by user
  - G4Materials
  - G4MaterialPropertiesTables
  - G4Elements
  - ...

Defining Detectors

- GEANT4 provides a G4VSensitiveDetector class that should be extended to build a detector
- G4VSensitiveDetector classes can be associated with (one or many) G4LogicalVolumes
- GEANT4 will notify a G4VSensitiveDetector when a particle passes through associated Volume



GEANT4: Defining the G4VSensitiveDetector

```
 * @brief Detector class extending G4VSensitiveDetector
class Detector: public G4VSensitiveDetector {
public:
     * Constructor
    Detector(const char* name);

     * Constructor setting the name of the detector to "Detector"
    Detector();

     * Destructor
    virtual ~Detector();

     * This function will be called automatically at the end of each event
    virtual void EndOfEvent(G4HCofThisEvent* aHC);

     * This function will be called automatically at the beginning of each event
    virtual void Initialize(G4HCofThisEvent* aHC);

     * This accessor can be used to access the collection of hits stored in
    virtual G4THitsCollection<DetectorHit>* GetHitsCollection();

     * This function will be called whenever a particle passes through the
    virtual G4bool ProcessHits(G4Step*aStep, G4TouchableHistory*ROhist);
```

initialize detector and the containers for the stored particle hits before each event

interface to handle stored hits after each event

called by G4SteppingManager to inform on particles

GEANT4: Creating the G4VHit

```
G4bool Detector::ProcessHits(G4Step* aStep, G4TouchableHistory* ROhist) {
    G4THitsCollection<DetectorHit>* hc = GetHitsCollection();
    if (hc == NULL)
        return false;

    // create hit and store it.
    DetectorHit* hit = new DetectorHit(aStep);
    hc->insert(hit);
    return true;
}
```

store hit

class holding infos about the particle

offers access to current volume

You can obtain all kinds of information via the G4Step* interface

```
DetectorHit::DetectorHit(G4Step* aStep) {
    // construct from aStep
    _preStepPoint = aStep->GetPreStepPoint()->GetPosition();
    _postStepPoint = aStep->GetPostStepPoint()->GetPosition();
    _particle = aStep->GetTrack()->GetDefinition();
    _energyDeposit = aStep->GetTotalEnergyDeposit();

    if (_particle == G4OpticalPhoton::Definition()) {
        aStep->GetTrack()->SetTrackStatus(fKillTrackAndSecondaries);
        _energyDeposit += aStep->GetTrack()->GetMomentum().mag() * CLHEP::c_light;
    }
}
```

You can also do arbitrary changes to particle

GEANT4: Accessing stored hits at the end of event

- Extend G4UserEventAction to access the event data

```
 * @brief EventAction class extending G4UserEventAction that can be submitted
class EventAction: public G4UserEventAction {
public:
     * Constructor
    EventAction(TH1* dataHist);

     * Destructor
    virtual ~EventAction();

     * @override G4UserEventAction::BeginOfEventAction
    virtual void BeginOfEventAction(const G4Event* anEvent);

     * @override G4UserEventAction::EndOfEventAction
    virtual void EndOfEventAction(const G4Event* anEvent);
```

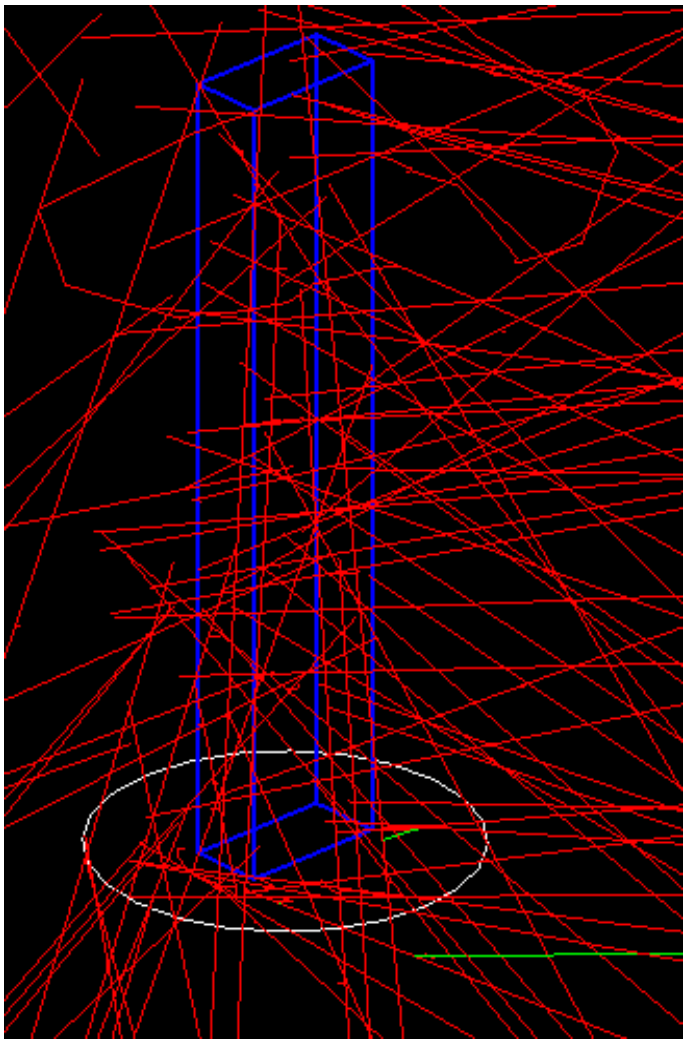
interface of G4UserEventAction

```
void EventAction::EndOfEventAction(const G4Event* anEvent) {
    // access hit collections and handle the stored hits
    for (int i = 0; i < anEvent->GetHCofThisEvent()->GetNumberOfCollections(); i++) {
        G4VHitsCollection* hits = anEvent->GetHCofThisEvent()->GetHC(0);
        double totalEnergy = 0;
        for (unsigned int j = 0; j < hits->GetSize(); j++) {
            totalEnergy += ((DetectorHit*)hits->GetHit(j))->GetEnergyDeposit();
        }
        _dataHistogram->Fill(hits->GetSize());
    }
```

access stored hits via G4Event*

Example: A Trigger Counter

- Let's look at our first GEANT4 program
- Simulate a Trigger counter made up of Polystyrene
  - 40mm x 400mm x 10mm in size
- Question: How much energy does a muon deposit in the Scintillator?
  - isotropic incidence
  - 1 GeV momentum



GEANT4: PhysicsList

```
class PhysicsList: public G4VModularPhysicsList {
public:
    * Constructor
    PhysicsList();

    * Destructor
    ~PhysicsList();

    * Sets the cuts for particle production.
    void SetCuts();
};
```

```
PhysicsList::PhysicsList() {
    // a number of physics constructors to handle particle creation and process
    // definition
    RegisterPhysics(new G4EmStandardPhysics());
    // suppress messages
    SetVerboseLevel(0);

    // this value defines the minimal range of a particle that should be
    // considered for tracking in GEANT4. It makes sense reducing the cut
    // value if you have small detector components
    defaultCutValue = 1 * mm;
}
void PhysicsList::SetCuts() {
    SetCutsWithDefault();
}
```

all the physics we need for muons

GEANT4: PrimaryGeneratorAction

```
PrimaryGeneratorAction::PrimaryGeneratorAction() {
    _gun = new G4ParticleGun(G4MuonMinus::Definition());
}

PrimaryGeneratorAction::~PrimaryGeneratorAction() {
    delete _gun;
}

void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent) {
    G4double x = (G4UniformRand() - .5) * 10 * CLHEP::cm;
    G4double y = (G4UniformRand() - .5) * 50 * CLHEP::cm;
    G4double z = 1 * CLHEP::cm;
    _gun->SetParticlePosition(G4ThreeVector(x, y, z));
    double phi = G4UniformRand() * CLHEP::twopi;
    double cosTheta = - G4UniformRand();
    double sinTheta = sqrt(1 - cosTheta * cosTheta);
    _gun->SetParticleMomentumDirection(G4ThreeVector(sinTheta * sin(phi),
            sinTheta * cos(phi), cosTheta));

    double energy = 1 * CLHEP::GeV;

    _gun->SetParticleEnergy(energy);
    _gun->GeneratePrimaryVertex(anEvent);
}
```

G4ParticleGun produces primary particles. We load it with a μ-

Set initial position of particle

Set direction of particle

Set energy and fire away

int main(int argc, char** argv)

```
// Construct the default run manager
G4RunManager* runManager = new G4RunManager;

// set mandatory initialization classes
G4VUserDetectorConstruction* detector = new DetectorConstruction();
runManager->SetUserInitialization(detector);
G4VUserPhysicsList* physics = new PhysicsList();
runManager->SetUserInitialization(physics);

// set mandatory user action class
runManager->SetUserAction(new PrimaryGeneratorAction());
// event action is optional
runManager->SetUserAction(new EventAction(dataHistogram));

// Initialize G4 kernel
runManager->Initialize();

// Initialize vis manager
G4VisManager* visManager = new G4VisExecutive;
visManager->Initialize();

// Get the pointer to the UI manager and set verbosities
G4UImanager* UI = G4UImanager::GetUIpointer();
```

G4RunManager is the backend of the simulation framework

Definition of Geomety

Physics Definition

Set initial state

Our own EventAction Class used for readout

some initalization

int main(int argc, char** argv)

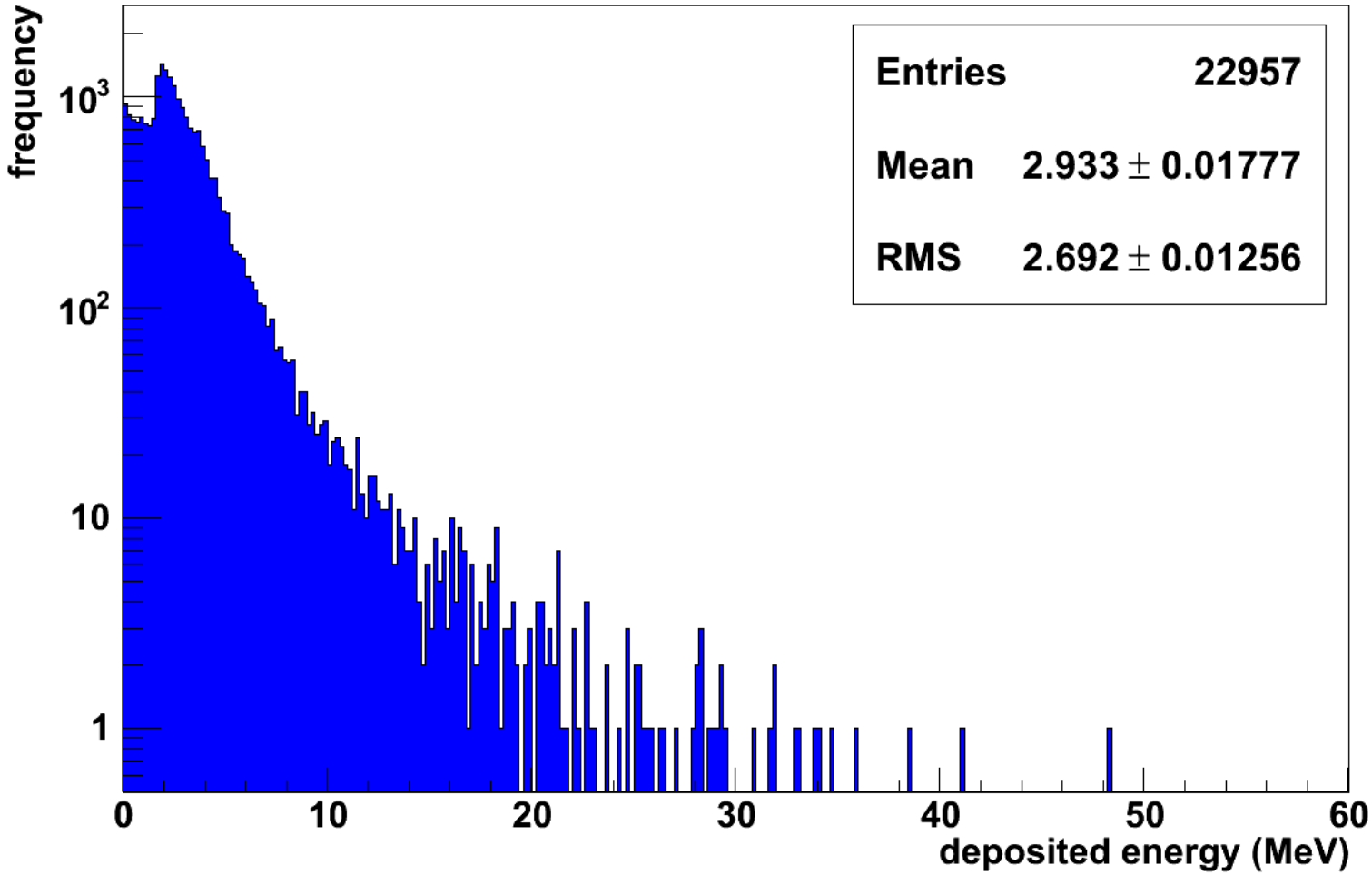
```
if (macro == "") {
    // Start a run
    G4int numberOfEvent = 1000000;
    if (display)

    runManager->BeamOn(numberOfEvent);
} else {

// Job termination
//
// Free the store: user actions, physics_list and detector_description are
//                 owned and deleted by the run manager, so they should not
//                 be deleted in the main() program !

if (display) {
    G4UIsession* session = new G4UIterminal(new G4UItcsh);
    session->SessionStart();

    delete session;
}

delete visManager;

delete runManager;
```

Run Simulation

Start GEANT4 command shell

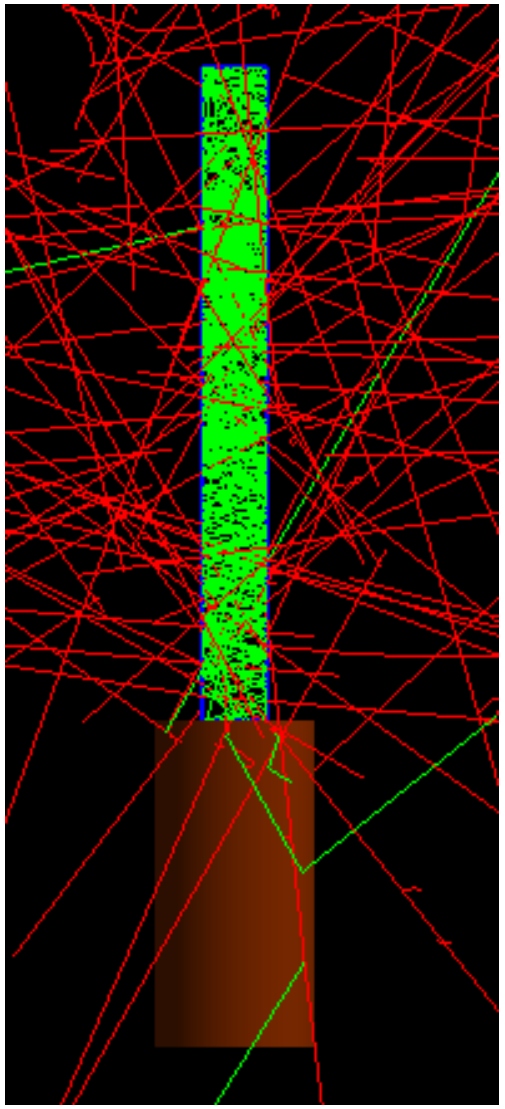
cleanup

The Result: Energy left by a μ in 1cm scintillator



Example: Raytracing in GEANT4

- Next up: Add Photons
- GEANT4 knows about scintillation and optical processes
- Use this to determine how many photons can be detected reading out scintillator with PMT
  - quantum efficiency 20%
  - borosilicate glass cathode of PMT (5cm diameter)
  - reflective coating of scintillator (reflectivity 92%)



GEANT4: New PhysicsList - OpticalPhysics

```
class OpticalPhysics : public G4VPhysicsConstructor {
public:
     * Constructor
    OpticalPhysics();

     * Destructor
    virtual ~OpticalPhysics();

     * @override G4VPhysicsConstructor::ConstructProcess
    void ConstructProcess();

     * @override G4VPhysicsConstructor::ConstructParticle
    void ConstructParticle();
};
```

this is all it takes to define a new particle

```
void OpticalPhysics::ConstructParticle() {
    // call G4OpticalPhoton::Definition to define particle once and for all
    G4OpticalPhoton::Definition();
}
```

GEANT4: New PhysicsList - OpticalPhysics

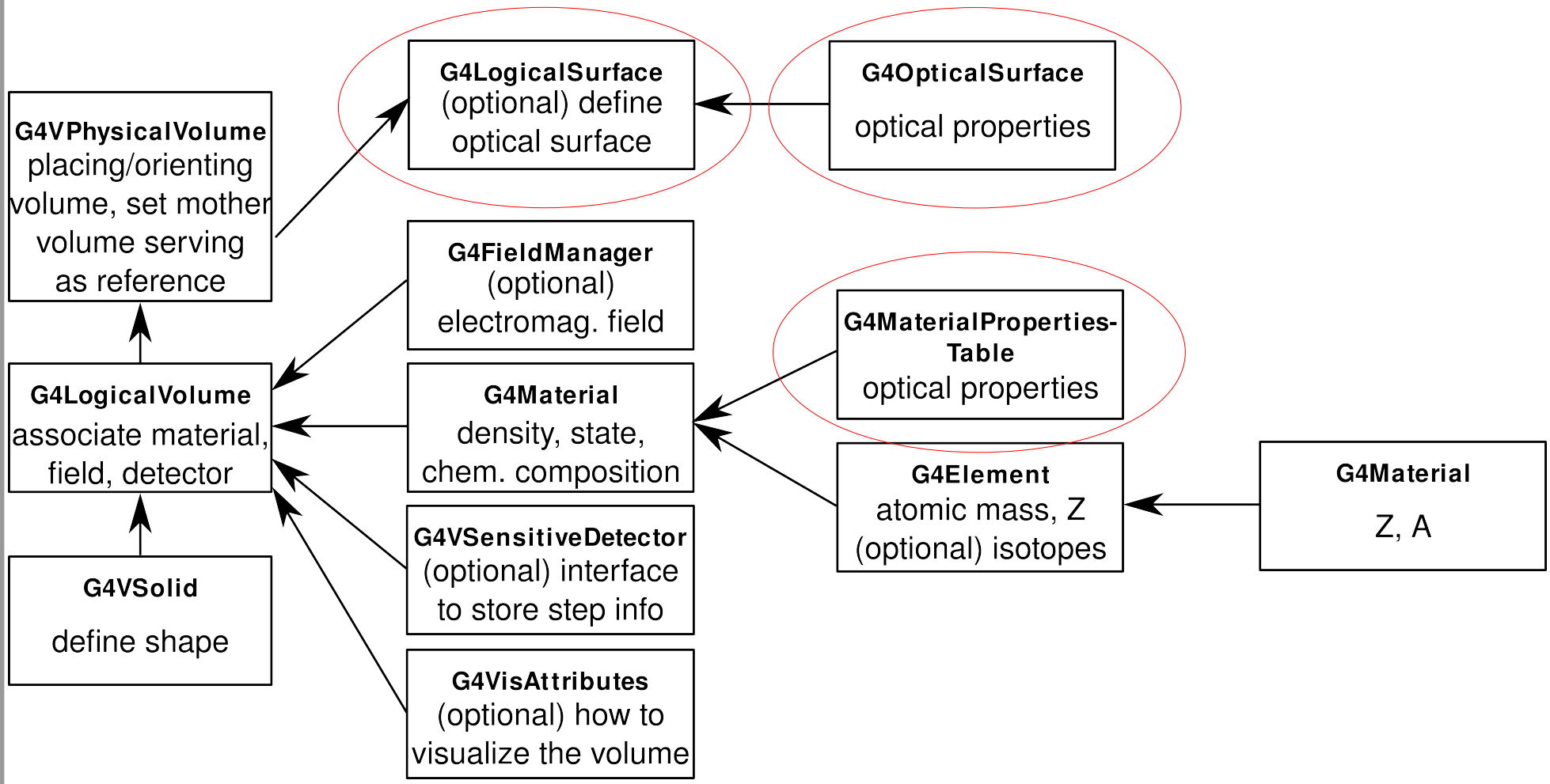
```
void OpticalPhysics::ConstructProcess() {
    // create one instance of scintillation process. If we wanted scintillation
    // for different particles to behave in a different manner we would not do that
    G4Scintillation *theScintillation = new G4Scintillation();

    // track photons first will reduce the memory requirements slightly
    // otherwise its completely unnecessary
    theScintillation->SetTrackSecondariesFirst(true);

    // an iterator to iterate over all defined particles
    theParticleIterator->reset();

    // (*theParticleIterator)() increments the particleIterator and returns
    // true or returns false
    while ((*theParticleIterator)()) {
        G4ParticleDefinition* particle = theParticleIterator->value();
        G4ProcessManager* processManager = particle->GetProcessManager();

        if (theScintillation->IsApplicable(*particle)) {
            processManager->AddProcess(theScintillation);
            // theScintillation has to be done last because it needs to know
            // how much energy the other processes have deposited.
            processManager->SetProcessOrderingToLast(theScintillation,
                    idxAtRest);
            processManager->SetProcessOrderingToLast(theScintillation,
                    idxPostStep);
        }

        // add scattering, absorption, refraction/reflection for photons
        // the basic transportation is added automagically by G4VUserPhysicsList
        // however you may still modify it at this point
        if (particle == G4OpticalPhoton::OpticalPhoton()) {
            processManager->AddDiscreteProcess(new G4OpAbsorption());
            processManager->AddDiscreteProcess(new G4OpRayleigh());
            processManager->AddDiscreteProcess(new G4OpBoundaryProcess());
        }
    }
}
```

construct the scintillation process

add scintillation process to all particles that it is applicable to

add optical physics processes to photons (absorption/reflection /refraction/scattering)

GEANT4 Geometry Management



GEANT4: Optical Properties of Materials

```
void DetectorConstruction::defineOpticalAir() {
    // set optical properties of Air
    G4MaterialPropertiesTable* propertiesTable =
            new G4MaterialPropertiesTable();

    // set absorption length to a low value to make display simpler
    G4MaterialPropertyVector* absorption = new G4MaterialPropertyVector();
    // note: GEANT4 may crash when encountering photons with energies outside
    // the boundary energies defined for MaterialPropertiesVectors. Fortunately
    // we control which optical photons are produced!
    absorption->AddElement(0.1 * CLHEP::eV, 1 * CLHEP::mm);
    absorption->AddElement(10. * CLHEP::eV, 1 * CLHEP::mm);
    propertiesTable->AddProperty("ABSLENGTH", absorption);

    // set refractive index
    G4MaterialPropertyVector* rindex = new G4MaterialPropertyVector();
    rindex->AddElement(0.1 * CLHEP::eV, 1.59);
    rindex->AddElement(10. * CLHEP::eV, 1.59);
    propertiesTable->AddProperty("RINDEX", rindex);

    G4Material::GetMaterial("Air", true)->SetMaterialPropertiesTable(
            propertiesTable);
}
```

G4MaterialPropertiesTable holds all optical properties (and so far ONLY optical properties)

define abs. length to enable absorption process in material

optical photons cannot enter any materials without valid refractive index

GEANT4: Optical Surfaces

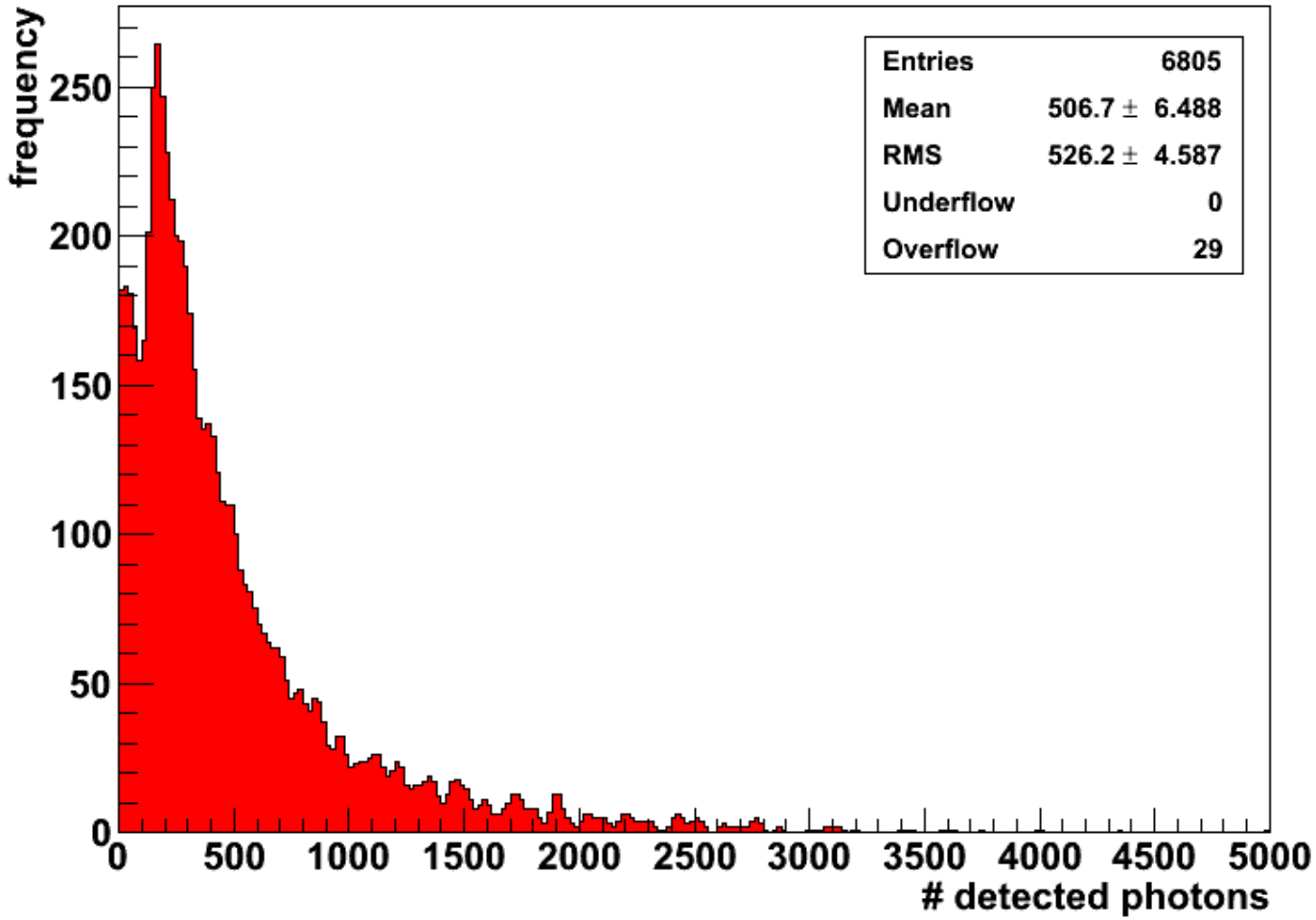
```
void DetectorConstruction::defineOpticalSurfaces() {
    _reflectiveSurface = new G4OpticalSurface("Scintillator-Air", glisur,
            polished, dielectric_metal);

    // here reflectivity of the surface is set
    G4MaterialPropertiesTable* propertiesTable =
            new G4MaterialPropertiesTable();
    G4MaterialPropertyVector* reflectivity = new G4MaterialPropertyVector();
    // reflectivity 0.97
    reflectivity->AddElement(0.1 * CLHEP::eV, 0.97);
    reflectivity->AddElement(10. * CLHEP::eV, 0.97);

    propertiesTable->AddProperty("REFLECTIVITY", reflectivity);

    _reflectiveSurface->SetMaterialPropertiesTable(propertiesTable);
}
```

G4OpticalSurface holds surface properties

```
    ////////////////////////////// OPTICAL SURFACES //////////////////////////////
    // this will add a reflecting surface to the scintillator where it is not
    // covered by the PMT. We claim a dielectric_metal interface between
    // scintillator and air in order to force reflection
    new G4LogicalBorderSurface("Scintillator-Air", physicalScintillator,
            physicalWorld, _reflectiveSurface);
```

G4LogicalBorderSurface associates interface between two physical volumes with G4OpticalSurface

Result: # Photons from Scintillator



Excercise:

- Let's assume we did not have a large PMT for readout but a 3mm x 3mm silicon photomultiplier with a quantum efficiency of 50%
  - How many photons would you detect with such a SiPM?
  - What time resolution would you achieve with that number of photons assuming that you can measure the time the first photon hits the SiPM?
    - GEANT4 keeps track of the time for each G4Step it calculates
    - To determine the time resolution, you have to keep track of the point where the primary particle passes through your scintillator
    - The simulation is very slow. You should parallelize your efforts and have GEANT4 continuously write 'out the latest event