Université de Strasbourg

Introduction to GEANT4

Alexandre Sécher

Hadrontherapy group

Office 123 - Building 25

alexandre.secher@iphc.cnrs.fr

Tips and docs

User's guide

https://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/ForApplicationDeveloper/html/

Source code:

https://www.apc.univ-paris7.fr/~franco/g4doxy/html/index.html

Course structure

- Marks based on the reproduction of results from some chosen publications

- Along the course you will have to approximate the acceptance of the experiment described in the paper:

  Evaluation of proton inelastic reaction models in Geant4 for prompt gamma production during proton radiotherapy,
  J. Jeyasugiththan and S. Peterson, 2015,
  Phys. Med. Biol. 60, 7617-7635

- The exercises will come in several level of difficulty, for you to choose

Monte-Carlo simulation

- Let's say you are running an experiment, and obtain some unexpected results.

Monte-Carlo simulation

- Let's say you are running an experiment, and obtain some unexpected results.
- In order to cross-check those results through a simulation, you will need to :
  - Define the geometry of the setup
  - Specify the beam configuration
  - Simulate real-world physics
  - Retrieve the generated data

Monte-Carlo simulation

- Let's say you are running an experiment, and obtain some unexpected results.

- In order to cross-check those results through a simulation, you will need to :
  - Define the geometry of the setup
  - Specify the beam configuration
  - Simulate real-world physics → Randomness : Monte-Carlo
  - Retrieve the generated data

Time to play

- Retrieve example_source.tar.gz from
  `/scratch/asecher/Geant4`
- Do not forget to setup your Geant4 and Root work environment
  `cernlib-use geant4 && cernlib-use root`
- Untar the file, and compile the sources inside an example_build directory you have created:

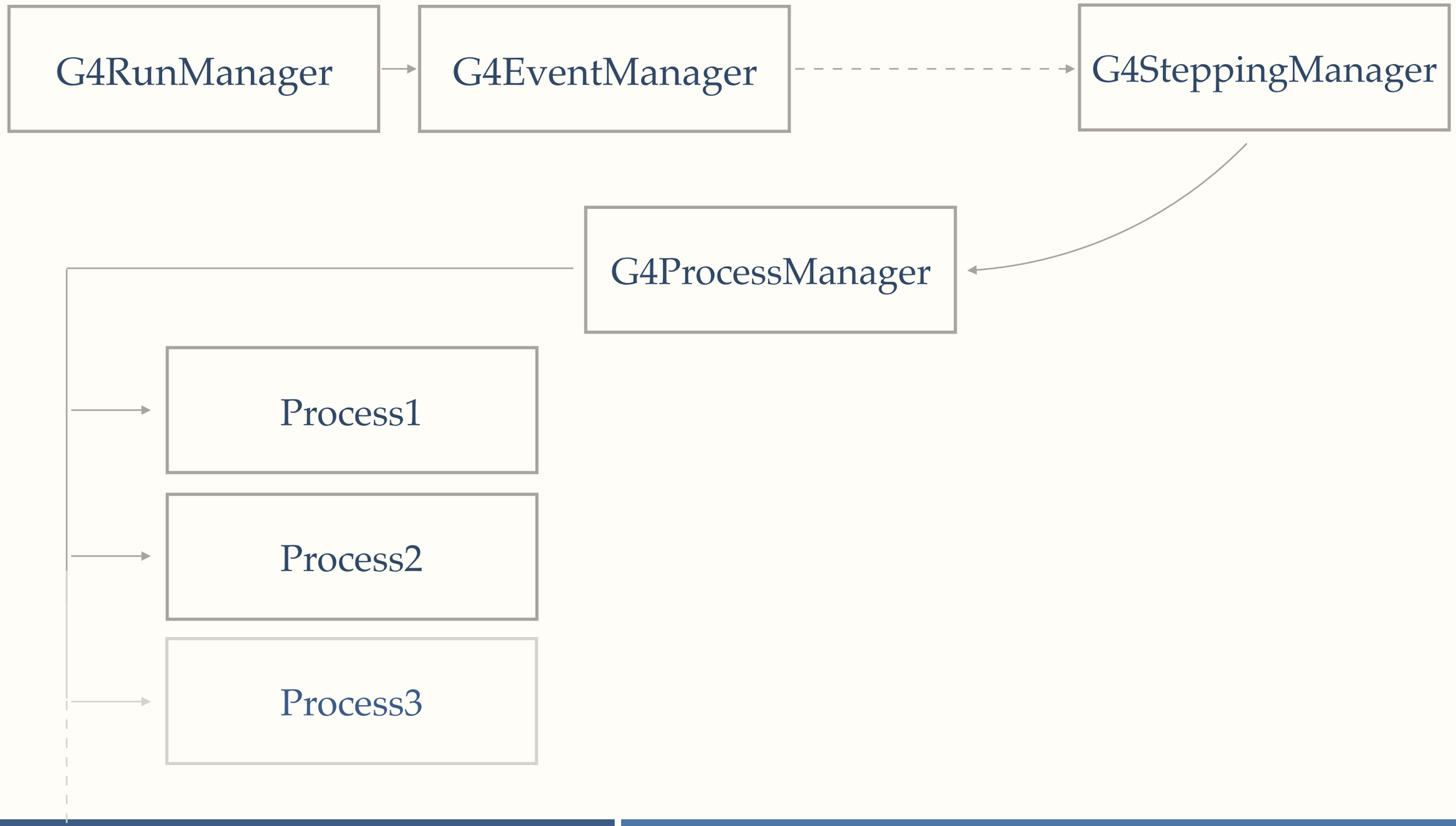
```
tar -xvf example_source.tar.gz
mkdir example_build && cd example_build
cmake ../example_source && make
```

- Run it, play with the interface, look at the source code and try to figure out what is happening

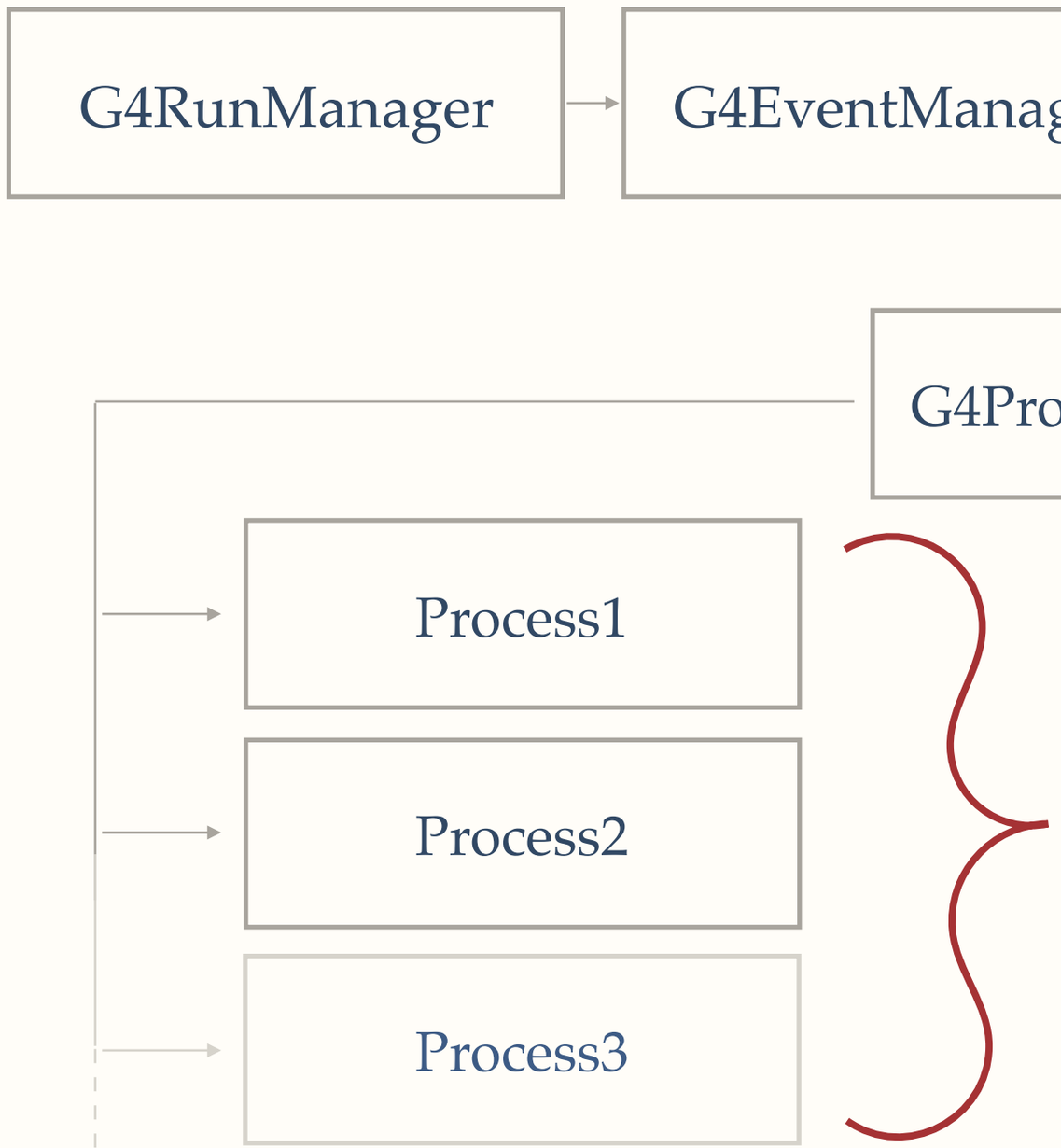
```
//with visualisation
./example
//inside the qt shell
/run/beamOn 10
```

```
//without visualisation
./example run.mac
```

Run structure



Run structure



The shortest step length will be selected, and the process associated with it

Geometry

Point of entry: main()

```
runManager->SetUserInitialization( new DetectorConstruction );
```

Point of entry: main()

```
runManager->SetUserInitialization( new DetectorConstruction );
```

```
void G4RunManager::SetUserInitialization
                        ( G4VUserDetectorConstruction* userInit )
{
    userDetector = userInit;
}
```

Base class

Register the geometry inside the run manager

Point of entry: main()

```
runManager->SetUserInitialization( new DetectorConstruction );
```

```
void G4RunManager::SetUserInitialization
                        ( G4VUserDetectorConstruction* userInit )
{
    userDetector = userInit;
}
```

Base class

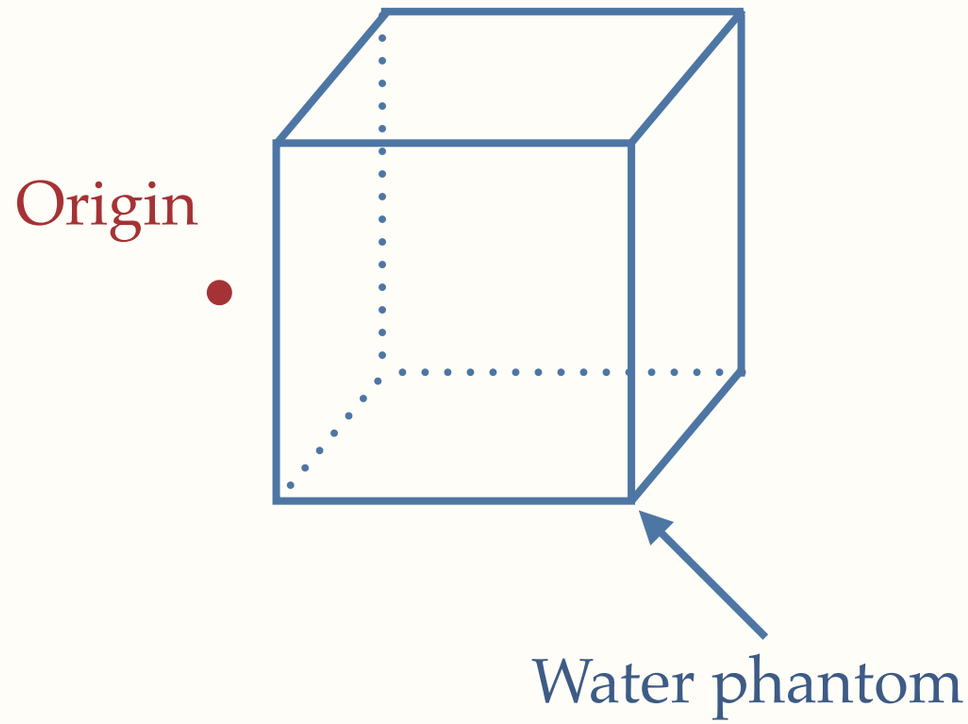
Register the geometry inside the run manager

```
void G4RunManager::InitializeGeometry()
{
    if(!userDetector){
        ... //throws exception
    }
    ...
    kernel->DefineWorldVolume(userDetector->Construct(), false);
    ...
}
```

Check the existence of a user-defined geometry

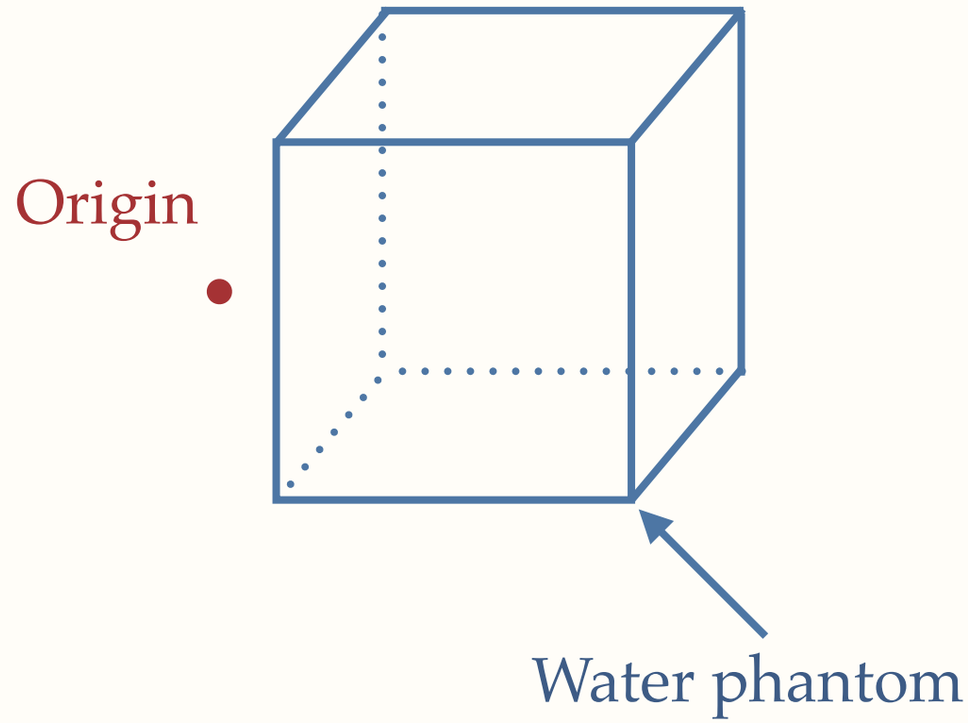
A bit of theory

- What information do you need in order to define a volume ?



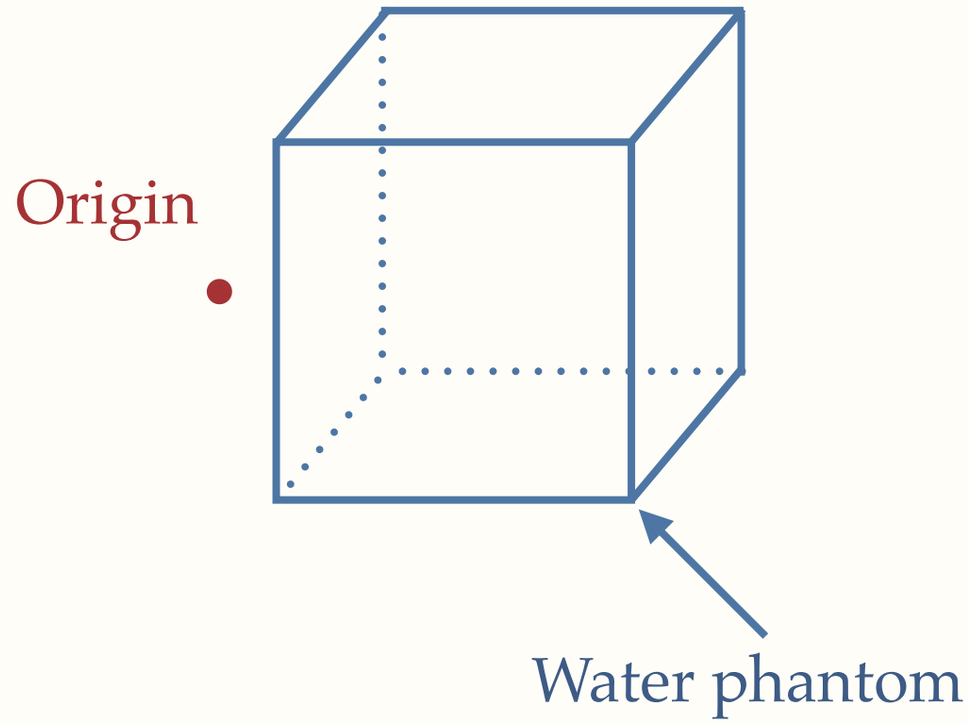
A bit of theory

- What information do you need in order to define a volume ?
  - form
  - dimensions
  - material
  - position
  - sensibility
  - …



A bit of theory

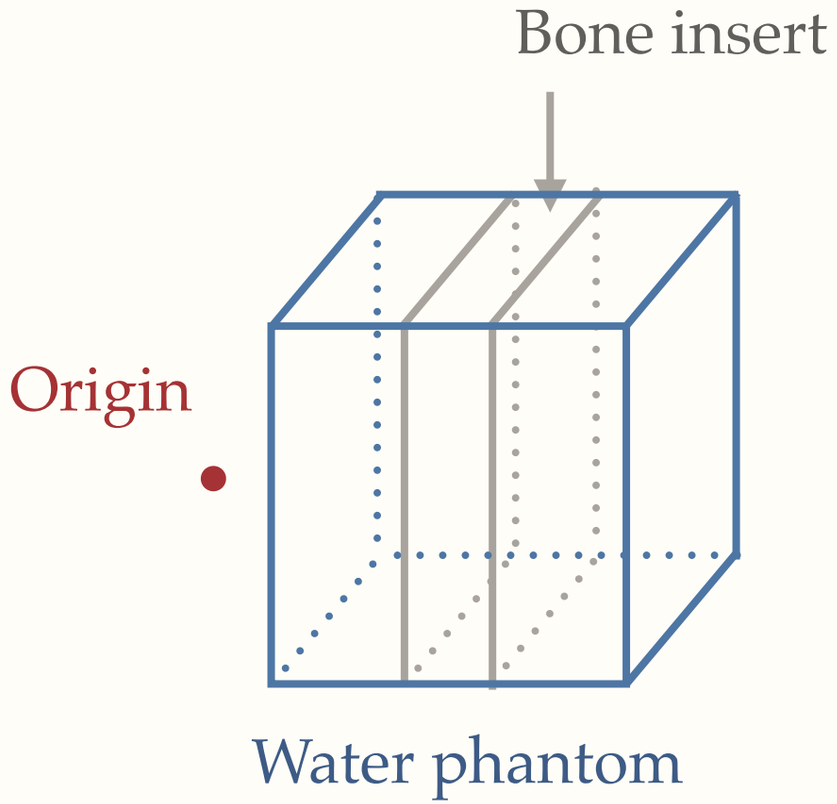
- What information do you need in order to define a volume ?
  - form
  - dimensions
  - material
  - position
  - sensibility
  - …



- The geometry definition is organised through **two layers**: logical and physical

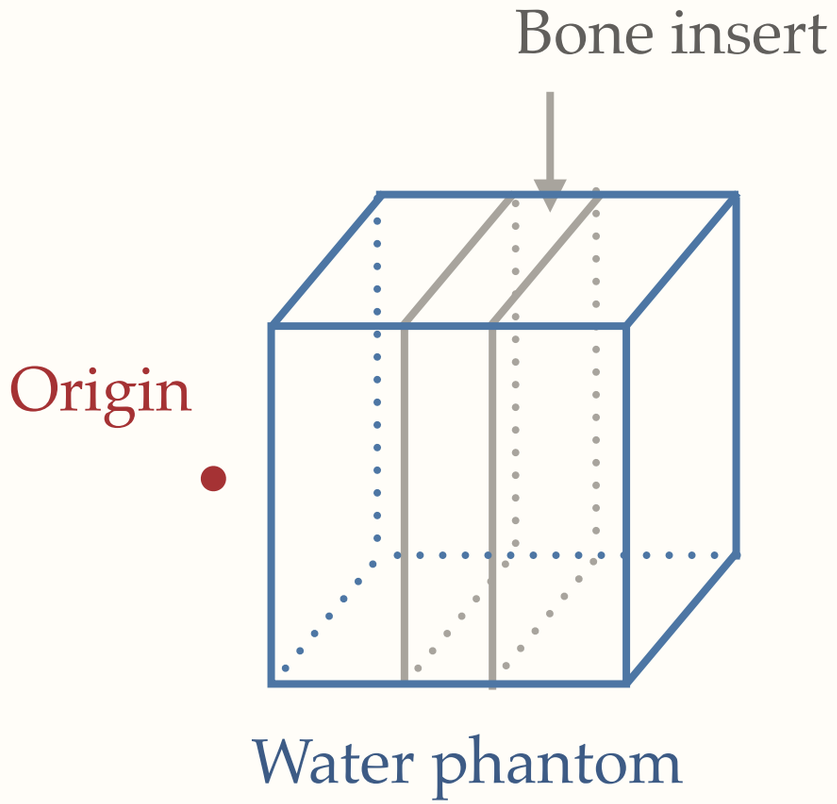
A bit of theory

- How would you handle **overlapping** volumes?



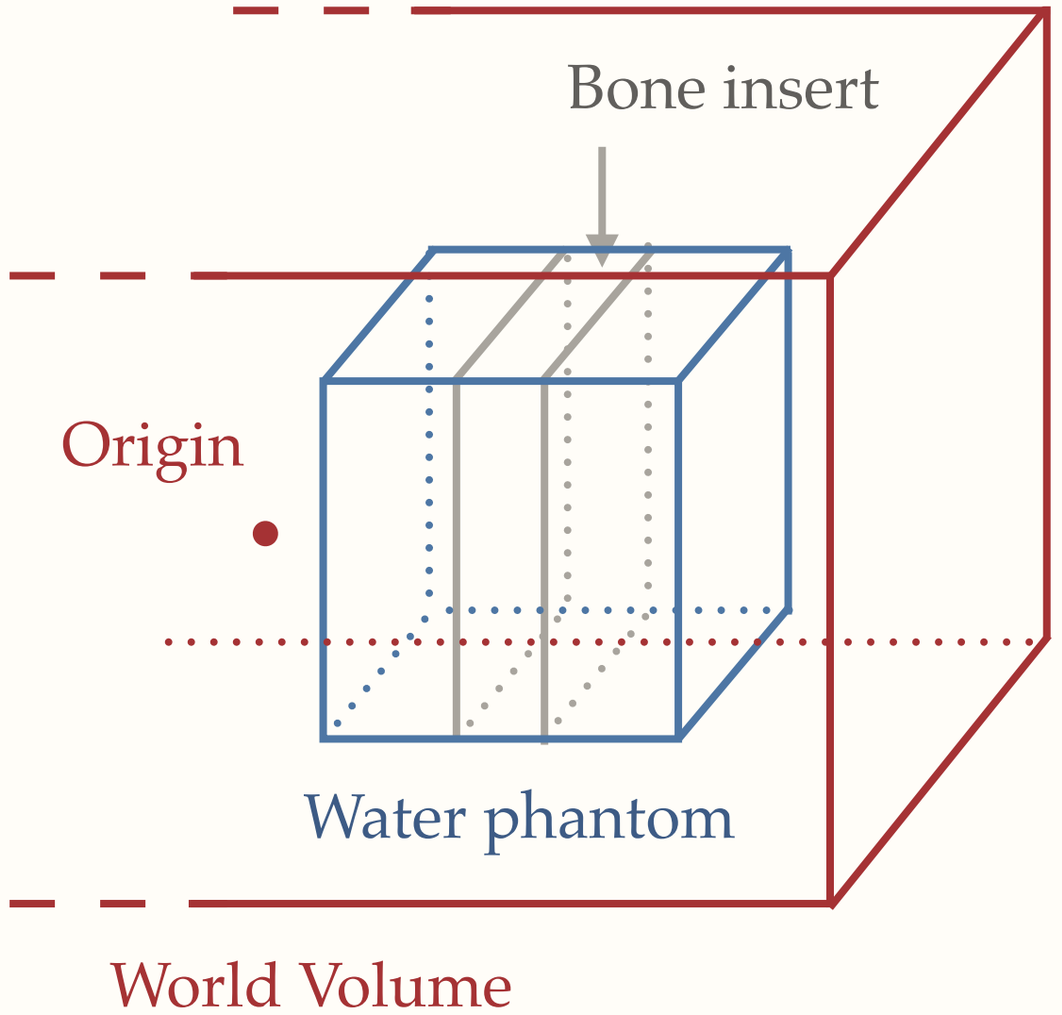
A bit of theory

- How would you handle **overlapping** volumes?
- GEANT4 geometry is based around a **hierarchical structure**



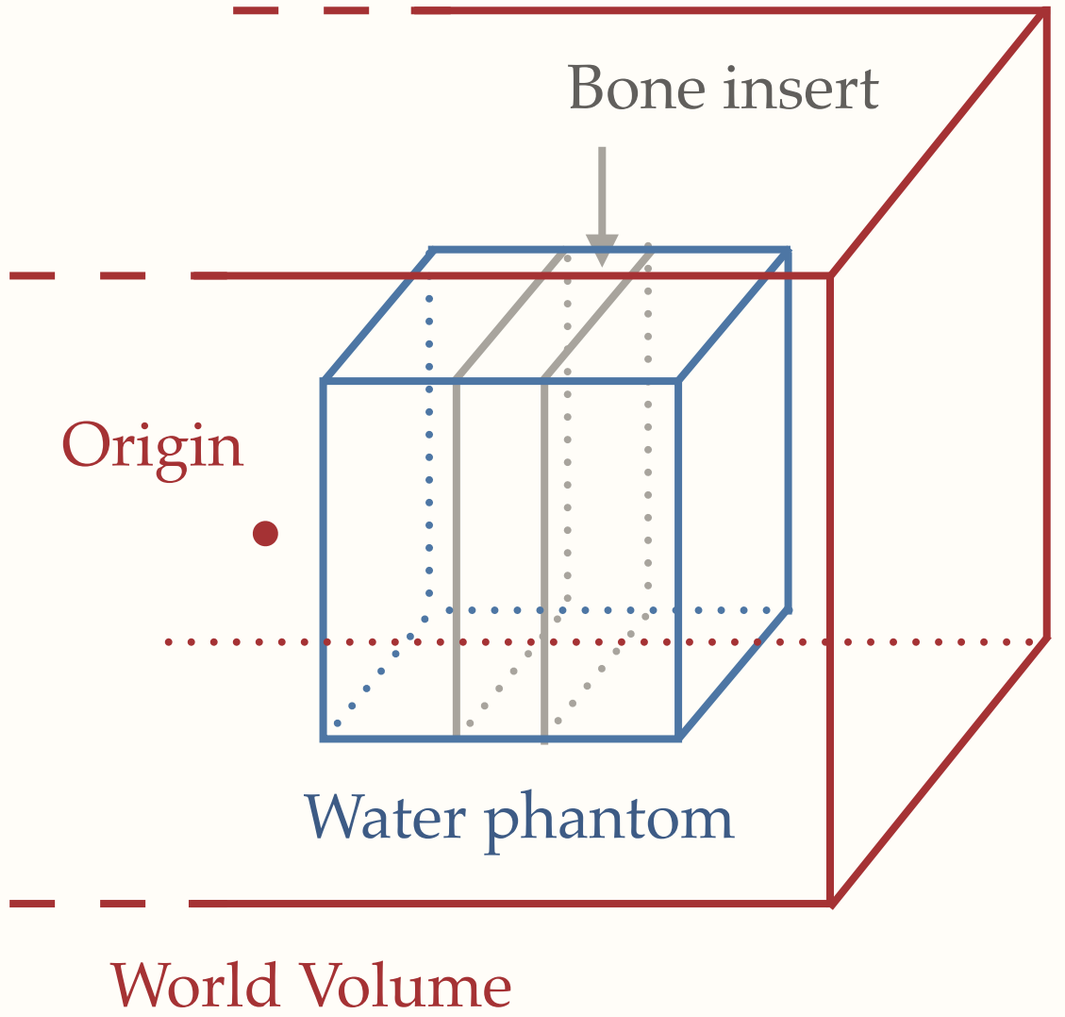
A bit of theory

- How would you handle **overlapping** volumes?
- GEANT4 geometry is based around a **hierarchical structure**
- The frame of reference for the experiment needs to be defined: **world volume**



A bit of theory

- How would you handle **overlapping** volumes?
- GEANT4 geometry is based around a **hierarchical structure**
- The frame of reference for the experiment needs to be defined: **world volume**



- Both the position inside of the hierarchy and in the world are defined in the physical volume layer

Two different layers: logical volume

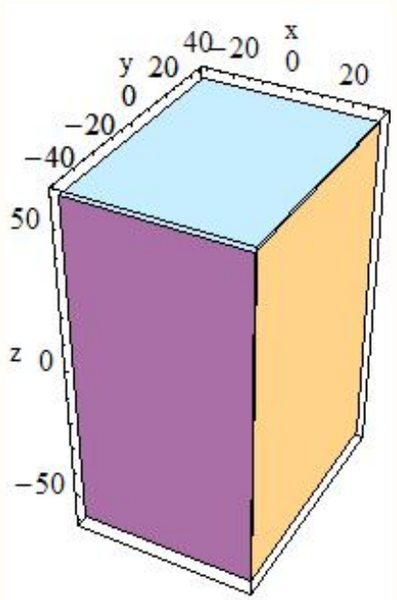
Defines form and dimensions

```
G4LogicalVolume( G4VSolid* pSolid,
                 G4Material* pMaterial,
                 const G4String& name,
                 G4FieldManager* pFieldMgr=0,
                 G4VSensitiveDetector* pSDetector=0,
                 G4UserLimits* pULimits=0,
                 G4bool optimise=true );
```

Defines material contained inside of the volume

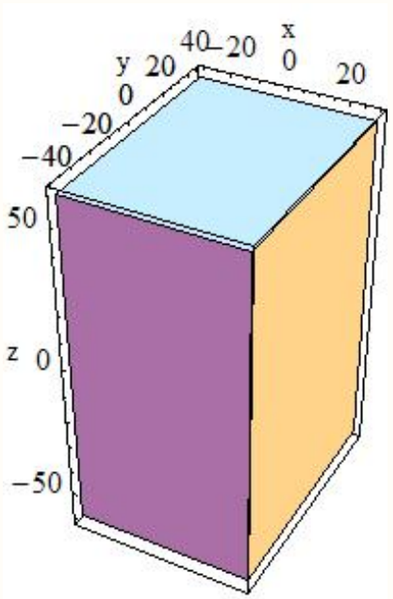
How to define a form: CSG

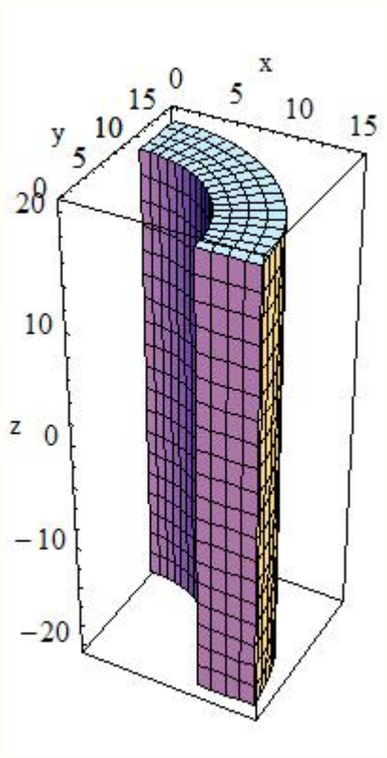
```
G4Box( const G4String& pName,
       G4double pX,
       G4double pY,
       G4double pZ );
```



How to define a form: CSG

```
G4Box( const G4String& pName,
       G4double pX,
       G4double pY,
       G4double pZ );
```





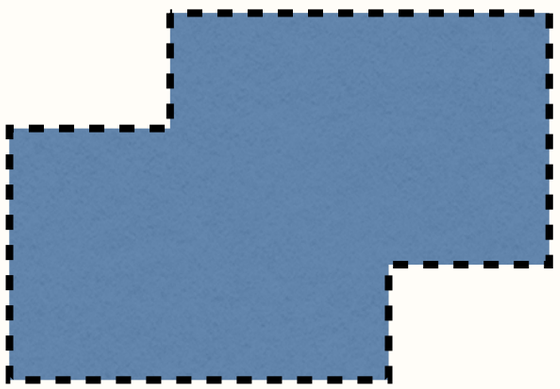

```
G4Tubs( const G4String& pName,
        G4double pRMin,
        G4double pRMax,
        G4double pDz,
        G4double pSPhi,
        G4double pDPhi );
```

How to define a form

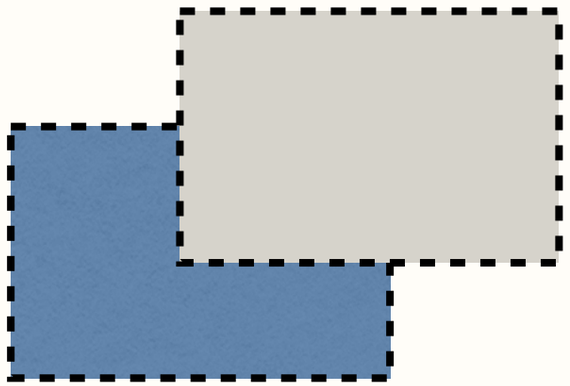
- Those volumes can be **combined** to form more complicated shapes

```
G4BooleanSolid( const G4String& pName,
                G4VSolid* pSolidA ,
                G4VSolid* pSolidB,
                G4RotationMatrix* rotMatrix,
                const G4ThreeVector& transVector    );
```

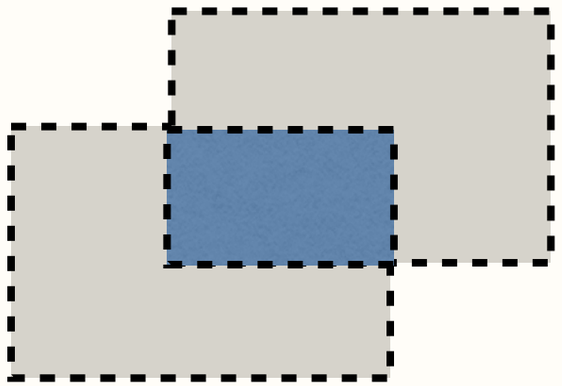
G4UnionSolid



G4SubtractionSolid



G4IntersectionSolid



How to define a material

- What do you need to specify in order to define a material ?

How to define a material

- What do you need to specify in order to define a material ?
  - atoms
  - density
  - temperature
  - state
  - pressure
  - …

How to define a material

- What do you need to specify in order to define a material ?
  - atoms
  - density
  - temperature
  - state
  - pressure
  - …

```
G4Element( const G4String& name,
           const G4String& symbol,
           G4double  Zeff,
           G4double  Aeff );
```

How to define a material

```
G4Material( const G4String& name,
            G4double  density,
            G4int     nComponents,
            G4State   state     = kStateUndefined,
            G4double  temp      = NTP_Temperature,
            G4double  pressure = CLHEP::STP_Pressure );

void G4Material::AddElement( G4Element* element,
                             G4int      nAtoms   );
void G4Material::AddElement( G4Element* element,
                             G4double   fraction );
```

Number of different elements

Fraction of the total mass

Number of atoms of this element in the molecule/ compound

An example: logical volume

```
double phantom_size = 25 * CLHEP::cm;
G4VSolid* solidPhantom = new G4Box( "Phantom",
                                    0.5 * phantom_size,
                                    0.5 * phantom_size,
                                    0.5 * phantom_size );

auto* H = new G4Element("Hydrogen", "H", 1, 1.008 * CLHEP::g/
CLHEP::mole);
auto* O = new G4Element("Oxygen", "O", 8, 16.00 * CLHEP::g/
CLHEP::mole);
auto* H2O = new G4Material("Water", 1.0 * CLHEP::g/CLHEP::cm3, 2);
H2O->AddElement(H, 2);
H2O->AddElement(O, 1);

auto* logicPhantom = new G4LogicalVolume( solidPhantom,
                                          H2O,
                                          "Phantom"  );
```

Two different layers: physical volume

Defines rotation and translation with respect to mother volume

```
G4PVPlacement( G4RotationMatrix *pRot,
               const G4ThreeVector &tlate,
               G4LogicalVolume *pCurrentLogical,
               const G4String& pName,
               G4LogicalVolume *pMotherLogical,
               G4bool pMany,
               G4int  pCopyNo,
               G4bool pSurfChk=false );
```

The associated logical volume

The mother's logical volume

Logical volumes can be repeated several times, but the copy number need to be incremented each time

An example: physical volume

```
auto * rotation = new G4RotationMatrix;
rotation->rotateY(45*CLHEP::deg);

new G4PVPlacement( rotation,
                   G4ThreeVector(0., 0., 10 * CLHEP::cm),
                   logicPhantom,
                   "Phantom",
                   logicWorld,
                   false,
                   0,
                   false);
```

Point of use: G4RunManager

```
void G4RunManager::InitializeGeometry()
{
    if(!userDetector){
        ... //throws exception
    }
    ...
    kernel->DefineWorldVolume(userDetector->Construct(), false);
    ...
}
```

Point of use: G4RunManager

```
void G4RunManager::InitializeGeometry()
{
    if(!userDetector){
        ... //throws exception
    }
    ...
    kernel->DefineWorldVolume(userDetector->Construct(), false);
    ...
}
```

Point of use: G4RunManager

```
void G4RunManager::InitializeGeometry()
{
    if(!userDetector){
        ... //throws exception
    }
    ...
    kernel->DefineWorldVolume(userDetector->Construct(), false);
    ...
}
```

```
void DefineWorldVolume( G4VPhysicalVolume * worldVol,
                        G4bool topologyIsChanged = true );
```

Point of use: G4RunManager

```
void G4RunManager::InitializeGeometry()
{
    if(!userDetector){
        ... //throws exception
    }
    ...
    kernel->DefineWorldVolume(userDetector->Construct(), false);
    ...
}
```

```
void DefineWorldVolume( G4VPhysicalVolume * worldVol,
                        G4bool topologyIsChanged = true );
```

```
virtual G4VPhysicalVolume*
        G4VUserDetectorConstruction::Construct() = 0;
```

The Construct() method must be overridden and must return the physical volume corresponding to the world volume

Advices and warnings

- The world volume is the one defining the global coordinates of the simulation
- The volumes are positioned through their centre
- Specifying the units you are using is **always** a good idea
- In order to define basic materials, it is a good idea to look into the NIST Database (and to use the G4NistManager) http://www.apc.univ-paris7.fr/%7Efranco/g4doxy/html/G4NistMaterialBuilder_8cc-source.html

Exercise

- Goal : reproduction of the geometry given in the Jeyasugiththan publication
- Three difficulty levels:
  - (A) Fill-in the blanks
  - (B) Structural code
  - (C) Empty class
- Retrieve the chosen files in:
  `/scratch/asecher/Geant4/geometry_exercise`
- And the rest of the files needed to run the simulation in:
  `/scratch/asecher/Geant4/core_exercise`

Beam

Point of entry: main()

```
runManager->SetUserAction(new PrimaryGeneratorAction);
```

Point of entry: main()

```
runManager->SetUserAction(new PrimaryGeneratorAction);
```

```
void G4RunManager::SetUserAction(G4VUserPrimaryGeneratorAction*
userAction)
{
  userPrimaryGeneratorAction = userAction;
}
```

Base class

Registers the primary generator

Point of entry: main()

```
runManager->SetUserAction(new PrimaryGeneratorAction);
```

```
void G4RunManager::SetUserAction(G4VUserPrimaryGeneratorAction*
userAction)
{
  userPrimaryGeneratorAction = userAction;
}

G4Event* G4RunManager::GenerateEvent(G4int i_event)
{
  if(!userPrimaryGeneratorAction)
  {
   ...//throws exception
  }
  G4Event* anEvent = new G4Event(i_event);
  userPrimaryGeneratorAction->GeneratePrimaries(anEvent);
  ...
}
```

Base class

Registers the primary generator

Checks the existence of a user primary generator

Point of use: G4RunManager

```
G4Event* G4RunManager::GenerateEvent(G4int i_event)
{
  if(!userPrimaryGeneratorAction)
  {
   ...//throws exception
  }
  G4Event* anEvent = new G4Event(i_event);
  userPrimaryGeneratorAction->GeneratePrimaries(anEvent);
  ...
}
```

```
virtual void G4VUserPrimaryGeneratorAction::GeneratePrimaries
                                (G4Event* anEvent) = 0;
```

The GeneratePrimaries() method must be overridden in your implementation

G4VUserPrimaryGeneratorAction::Purpose

- This class is **not** the generator of primaries, it is used to configure the generator

- GEANT4 delivers two generators that can be used in order to create the vertex of primaries: **G4ParticleGun** and **G4GeneralParticleSource**

- Both of them inherits from **G4VPrimaryGenerator**, and override the following method:

```
virtual void
   G4VPrimaryGenerator::GeneratePrimaryVertex(G4Event* evt) = 0;
```

G4VUserPrimaryGeneratorAction::Usage

- The PrimaryGeneratorAction class should contain a **G4VPrimaryGenerator**

G4VUserPrimaryGeneratorAction::Usage

- The PrimaryGeneratorAction class should contain a **G4VPrimaryGenerator**

G4VUserPrimaryGeneratorAction::Usage

- The PrimaryGeneratorAction class should contain a **G4VPrimaryGenerator**

- The method **G4VPrimaryGenerator ::**GeneratePrimaryVertex() should be called inside of GeneratePrimaries()

G4VUserPrimaryGeneratorAction::Usage

- The PrimaryGeneratorAction class should contain a **G4VPrimaryGenerator**

- The method **G4VPrimaryGenerator ::**GeneratePrimaryVertex() should be called inside of GeneratePrimaries()

G4VUserPrimaryGeneratorAction::Usage

- The PrimaryGeneratorAction class should contain a **G4VPrimaryGenerator**

- The method **G4VPrimaryGenerator ::**GeneratePrimaryVertex() should be called inside of GeneratePrimaries()

- **G4ParticleGun** can be used to configure the beam at compile time

G4VUserPrimaryGeneratorAction::Usage

- The PrimaryGeneratorAction class should contain a **G4VPrimaryGenerator**

- The method **G4VPrimaryGenerator ::**GeneratePrimaryVertex() should be called inside of GeneratePrimaries()

- **G4ParticleGun** can be used to configure the beam at compile time

G4VUserPrimaryGeneratorAction::Usage

- The PrimaryGeneratorAction class should contain a **G4VPrimaryGenerator**
- The method **G4VPrimaryGenerator ::**GeneratePrimaryVertex() should be called inside of GeneratePrimaries()
- **G4ParticleGun** can be used to configure the beam at compile time
- **G4GeneralParticleSource** should be use to configure the beam at run time, through a configuration file

Example::G4ParticuleGun

```
PrimaryGeneratorAction::PrimaryGeneratorAction() :
    fParticleGun{ new G4ParticleGun(1) }
{
  G4ParticleTable* particleTable =
                  G4ParticleTable::GetParticleTable();
  G4ParticleDefinition* particle =
                 particleTable->FindParticle("proton");
  fParticleGun->SetParticleDefinition( particle );

  fParticleGun ->
        SetParticleMomentumDirection( G4ThreeVector(0.,0.,1.) );
  fParticleGun->SetParticleEnergy( 100.*MeV );
}
```

Example::G4ParticuleGun

```
void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
  G4double x0 = 0, y0 = 0, z0 = -20 * CLHEP::cm;

  fParticleGun->SetParticlePosition(G4ThreeVector(x0,y0,z0));

  fParticleGun->GeneratePrimaryVertex(anEvent);
}
```

Example::G4GeneralParticleSource

```
PrimaryGeneratorAction::PrimaryGeneratorAction () :
    fGeneralParticleSource{ new G4GeneralParticleSource }
{ }


void PrimaryGeneratorAction::GeneratePrimaries (G4Event* anEvent)
{
   fGeneralParticleSource->GeneratePrimaryVertex (anEvent) ;
}
```

In the corresponding .mac configuration file:

```
/gps/particle proton
/gps/ene/mono 100.0 MeV
/gps/pos/centre 0 0 -20 cm
/gps/direction 0 0 1
```

Physics

Point of entry: main()

```
runManager->SetUserInitialization(new PhysicsList);
```

Point of entry: main()

```
runManager->SetUserInitialization(new PhysicsList);
```

```
void G4RunManager::SetUserInitialization
                                   (G4VUserPhysicsList* userInit)
{
  physicsList = userInit;
  ...
  physicsList->ConstructParticle(); //in kernel
  ...
}
```

Base class

PhysicsList is registered

The ConstructParticle method must be overridden

Point of entry: main()

```
runManager->SetUserInitialization(new PhysicsList);
```

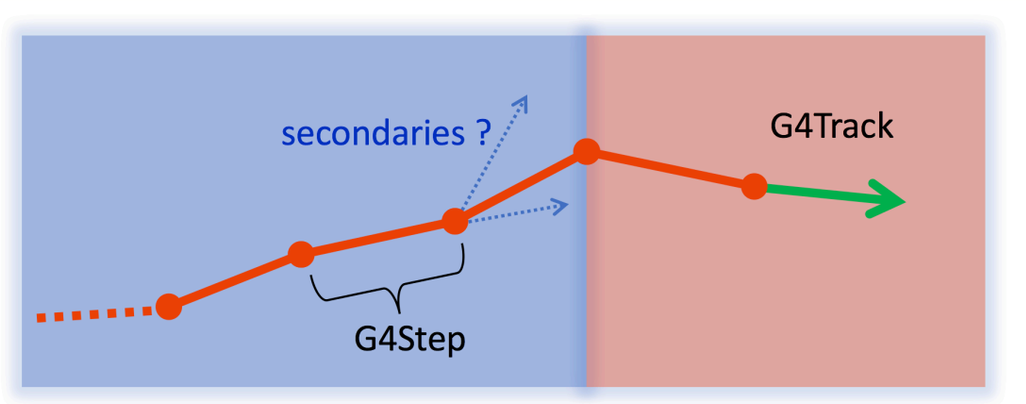
```
void G4RunManager::InitializePhysics()
{
  ...
  if(physicsList)
  {
    ...
    physicsList->ConstructProcess();
    ...
  }
  ...
}
```

Mandatory class

Must be overridden

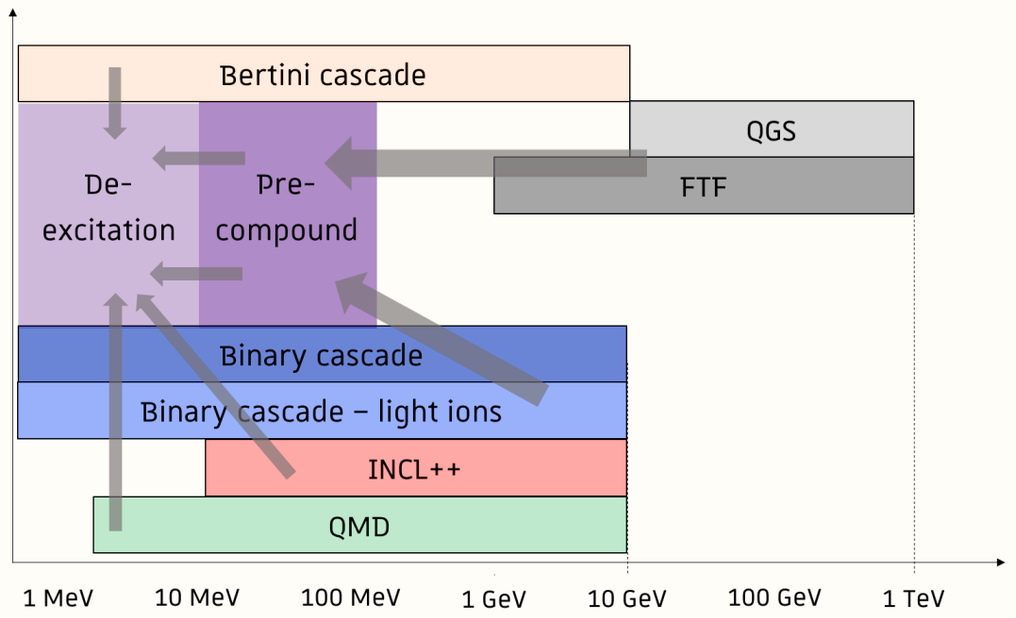
PhysicsList::Purpose

- Implementation of real-world physics numerically
- Discretisation of continuous processes implies the use of “steps”, forming a "track" when put together
- The physics list is responsible for the registration of the particles, and of the processes they can undergo inside of the simulation



PhysicsList::MainUsage

- Several ready-to-use physics lists are provided by the GEANT4 developers
- They are subdivided into modules handling some kind of physics: i.e., hardonic physics, electromagnetic physics, decays …
- Different models can be used on a same energy range, you have to choose one, and eventually compare



PhysicsList::MainUsage

- Non-exhaustive list of ready-to-use physics list:
  - QBBC
  - FTF_BERT_HP
  - FTFP_INCLXX_HP
  - QGSP_BERT
  - QGSP_BIC
  - QGSP_INCLXX
  - …

G4VModularPhysicsList

- Base class for the ready-to-use physics lists
- Register physics modules in its constructor:

```
void G4VModularPhysicsList::RegisterPhysics
(G4VPhysicsConstructor* fPhysics)
```

→ Base class for a physics module

G4VModularPhysicsList

- Base class for the ready-to-use physics lists
- Register physics modules in its constructor:

```
void G4VModularPhysicsList::RegisterPhysics
(G4VPhysicsConstructor* fPhysics)
```

Base class for a physics module

```
PhysicsList::PhysicsList()
{
  RegisterPhysics(new G4DecayPhysics);
  RegisterPhysics(new G4EmStandardPhysics);
  RegisterPhysics(new G4RadioactiveDecayPhysics);
}
```

Necessary modules to describe radioactive decay

PhysicsListCustomization::Example

```
class PhysicsList final : public G4VModularPhysicsList
{
public:
    PhysicsList();
    void ConstructProcess() override;
};
```

Registration of all the modules

```
PhysicsList::PhysicsList()
{
    RegisterPhysics( new G4EmStandardPhysics );
    RegisterPhysics( new G4EmExtraPhysics );
    RegisterPhysics( new G4DecayPhysics );
    RegisterPhysics( new G4HadronElasticPhysics );
    RegisterPhysics( new G4HadronPhysicsQGSP_BIC );
    RegisterPhysics( new G4StoppingPhysics );
    RegisterPhysics( new G4IonPhysics );
    RegisterPhysics( new G4NeutronTrackingCut );
}
```

PhysicsListCustomization::Example

```
class PhysicsList final : public G4VModularPhysicsList
{
public:
    PhysicsList();
    void ConstructProcess() override;
};
```

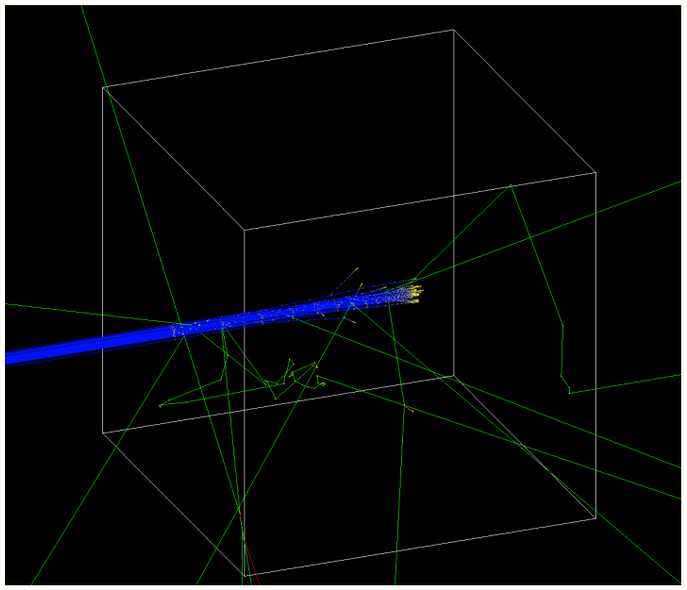
```
void PhysicsList::ConstructProcess()
{
    G4VModularPhysicsList::ConstructProcess();

    auto* proton = G4Proton::Definition();
    auto* manager = proton->GetProcessManager();
    auto* msc = manager->GetProcess("msc");
    manager->RemoveProcess(msc);
}
```

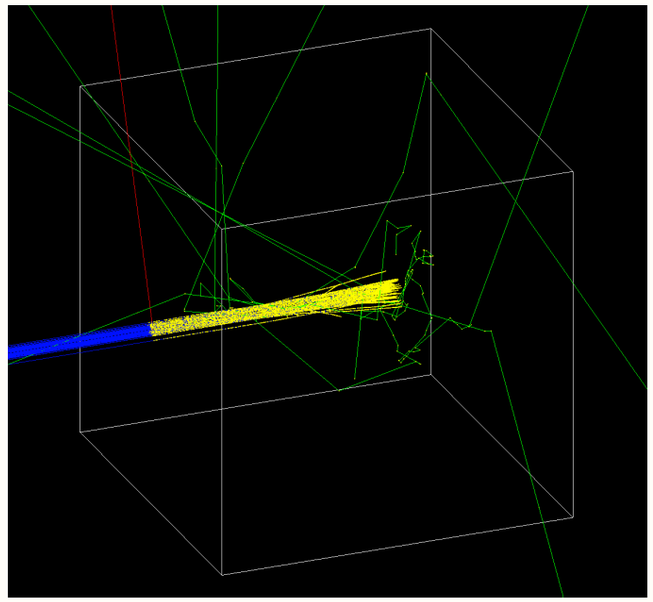
Construction of the processes

Removing the multiple coulomb scattering for the protons

Production cuts

- In order to reduce the computational load, production cuts are used
- Some secondaries (**γ**, e-, e+) are only produced and transported if their energy is sufficient to propagate farther than a pre-defined distance : it is the cut value (1 mm by default)





```
/run/setCuts 10 um
```

Exercise

- Goal : preparation of several macros reproducing the gamma lines of interest (isotropic and cylindric source)

- Three difficulty levels:
  - (A) Fill-in the blanks
  - (B) Guided construction
  - (C) Empty class

- Retrieve the chosen files in:
  `/scratch/asecher/Geant4/beam&physics_exercise`

- Do not forget to uncomment the corresponding lines in your main

Data retrieval

Point of entry: main()

```
auto* runAction = new RunAction;
runManager->SetUserAction( runAction ) ;

auto* eventAction = new EventAction{ runAction };
runManager->SetUserAction( eventAction ) ;

runManager->SetUserAction( new SteppingAction{ eventAction } );
```

Base class

```
void G4RunManager::SetUserAction(G4UserRunAction* userAction)
{
    userRunAction = userAction;
}
```

Registering

Point of entry: main()

```
auto* runAction = new RunAction;
runManager->SetUserAction( runAction ) ;

auto* eventAction = new EventAction{ runAction };
runManager->SetUserAction( eventAction ) ;

runManager->SetUserAction( new SteppingAction{ eventAction } );
```

```
void G4RunManager::SetUserAction(G4UserEventAction* userAction)
{
    eventManager->SetUserAction(userAction);
    userEventAction = userAction;
}
```

Base class

Registering

Point of entry: main()

```
auto* runAction = new RunAction;
runManager->SetUserAction( runAction ) ;

auto* eventAction = new EventAction{ runAction };
runManager->SetUserAction( eventAction ) ;

runManager->SetUserAction( new SteppingAction{ eventAction } );
```

Base class

```
void G4RunManager::SetUserAction(G4UserSteppingAction* userAction)
{
    eventManager->SetUserAction(userAction);
    userSteppingAction = userAction;
}
```

Registering

Outline of a run

```
void G4RunManager::BeamOn( G4int n_event,
                           const char* macroFile,
                           G4int n_select           )
{
    ...
    RunInitialization();
    DoEventLoop(n_event,macroFile,n_select);
    RunTermination();
    ...
}

void G4RunManager::RunInitialization()
{
    ...
    if(userRunAction) userRunAction->BeginOfRunAction(currentRun);
    ...
}
```

Should be overridden

Outline of a run

```
void G4RunManager::BeamOn( G4int n_event,
                           const char* macroFile,
                           G4int n_select           )
{
    ...
    RunInitialization();
    DoEventLoop(n_event,macroFile,n_select);
    RunTermination();
    ...
}

void G4RunManager::DoEventLoop(…)
{
    ...
    currentEvent = GenerateEvent(i_event);
    eventManager->ProcessOneEvent(currentEvent);
    ...
}
```

Outline of a run

```
void G4RunManager::BeamOn( G4int n_event,
                           const char* macroFile,
                           G4int n_select           )
{
    ...
    RunInitialization();
    DoEventLoop(n_event,macroFile,n_select);
    RunTermination();
    ...
}

void G4RunManager::RunTermination()
{
    ....
    if(userRunAction) userRunAction->EndOfRunAction(currentRun);
    ...
}
```

Should be overridden

Outline of a run

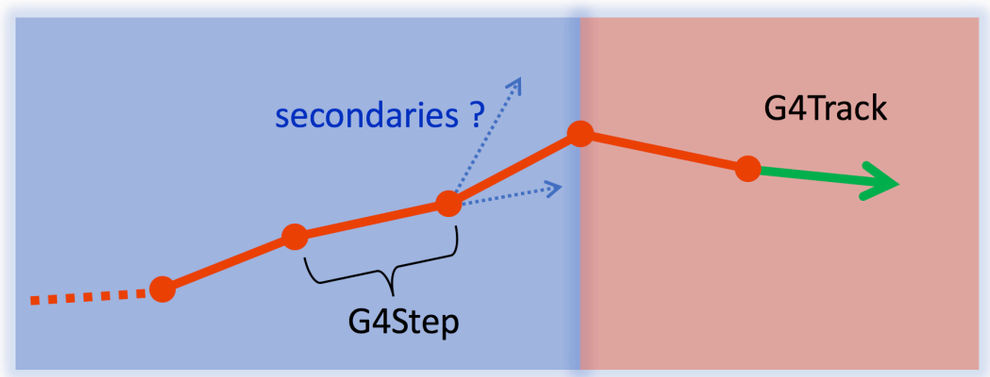
```
void G4RunManager::BeamOn( G4int n_event,
                           const char* macroFile,
                           G4int n_select           )
{
    ...
    RunInitialization();
    DoEventLoop(n_event,macroFile,n_select);
    RunTermination();
    ...
}

void G4RunManager::RunTermination()
{
    ....
    if(userRunAction) userRunAction->EndOfRunAction(currentRun);
    ...
}
```

Should be overridden

Outline of a run

```
void G4EventManager::ProcessOneEvent(G4Event* anEvent)
{
     ...
     trackContainer->PrepareNewEvent();
     ...
     trackManager->ProcessOneTrack( track  );
     ...
}

void G4TrackingManager::ProcessOneTrack(G4Track* apValueG4Track)
{
     ...
     fpSteppingManager->Stepping();
     ...
}
```

Additional tracks will eventually be added to the track container



Outline of a run

```
G4StepStatus G4SteppingManager::Stepping()
{
      ...
      if( fUserSteppingAction != 0 ) {
          fUserSteppingAction->UserSteppingAction(fStep);
      }
      ...
}
```

Should be overridden

RunAction::example

```
RunAction::RunAction() :
     fEvents{-1},
     fEdep{-1.},
     fDepth{0.},
     fPartName{"unknown"},
     fTrackID{0},
     fParentID{-1},
     fTree{"tree", "tree"}
{
     fTree.Branch("Edep", &fEdep, "Edep /D");
     fTree.Branch("Depth", &fDepth, "Depth /D");
     fTree.Branch("ptclName" , &fPartName);
     fTree.Branch("trackID"  , &fTrackID  ,"trackID  /I") ;
     fTree.Branch("parentID"  , &fParentID  ,"parentID  /I") ;
}
```

RunAction::example

```
RunAction::RunAction() :
     fEvents{-1},
     fEdep{-1.},
     fDepth{0.},
     fPartName{"unknown"},
     fTrackID{0},
     fParentID{-1},
     fTree{"tree", "tree"}
{
     fTree.Branch("Edep", &fEdep, "Edep /D");
     fTree.Branch("Depth", &fDepth, "Depth /D");
     fTree.Branch("ptclName" , &fPartName);
     fTree.Branch("trackID"  , &fTrackID  ,"trackID  /I") ;
     fTree.Branch("parentID"  , &fParentID  ,"parentID  /I") ;
}
```

```
TTree( const char *name,
       const char *title, … )
```

RunAction::example

```
RunAction::RunAction() :
     fEvents{-1},
     fEdep{-1.},
     fDepth{0.},
     fPartName{"unknown"},
     fTrackID{0},
     fParentID{-1},
     fTree{"tree", "tree"}
{
     fTree.Branch("Edep", &fEdep, "Edep /D");
     fTree.Branch("Depth", &fDepth, "Depth /D");
     fTree.Branch("ptclName" , &fPartName);
     fTree.Branch("trackID"  , &fTrackID  ,"trackID  /I") ;
     fTree.Branch("parentID"  , &fParentID  ,"parentID  /I") ;
}
```

```
TTree( const char *name,
       const char *title, … )
```

```
TBranch* TTree::Branch( const char * name,
                        void * address,
                        const char * leaflist, … )
```

RunAction::example

```
void RunAction::BeginOfRunAction(const G4Run* aRun)
{
    fEvents = aRun->GetNumberOfEventToBeProcessed();
}
```

```
void RunAction::EndOfRunAction ( const G4Run* )
{
    TFile outputFile{"Output.root", "recreate"};
    fTree.Write() ;
    outputFile.Close() ;
}
```

Creation and filling of the root file

SteppingAction::Example

```
void SteppingAction::UserSteppingAction(const G4Step* step)
{
    auto* track_h = step->GetTrack() ;
    if( track_h->GetVolume()->GetName() == "Phantom" ){

        auto* runAction_h = fEventAction->GetRunAction() ;
        runAction_h->SetEdep( step->GetTotalEnergyDeposit() *
                              CLHEP::MeV );
        ...
        runAction_h->SetTrackID( track_h->GetTrackID() );
        ...

        runAction_h->GetTree().Fill();
    }
}
```

The tree is filled here, at each step inside of the "Phantom" volume

Exercise

- Goal : return the number of primary photons entering the detector, and deduce the corresponding acceptance

- Three difficulty levels:
  - (A) Fill-in the blanks
  - (B) Guided construction
  - (C) Empty class
  - (C+) Add a way to retrieve the energy spectrum

- Retrieve the chosen files in:
  `/scratch/asecher/Geant4/data_exercise`

- Do not forget to uncomment the corresponding lines in your main