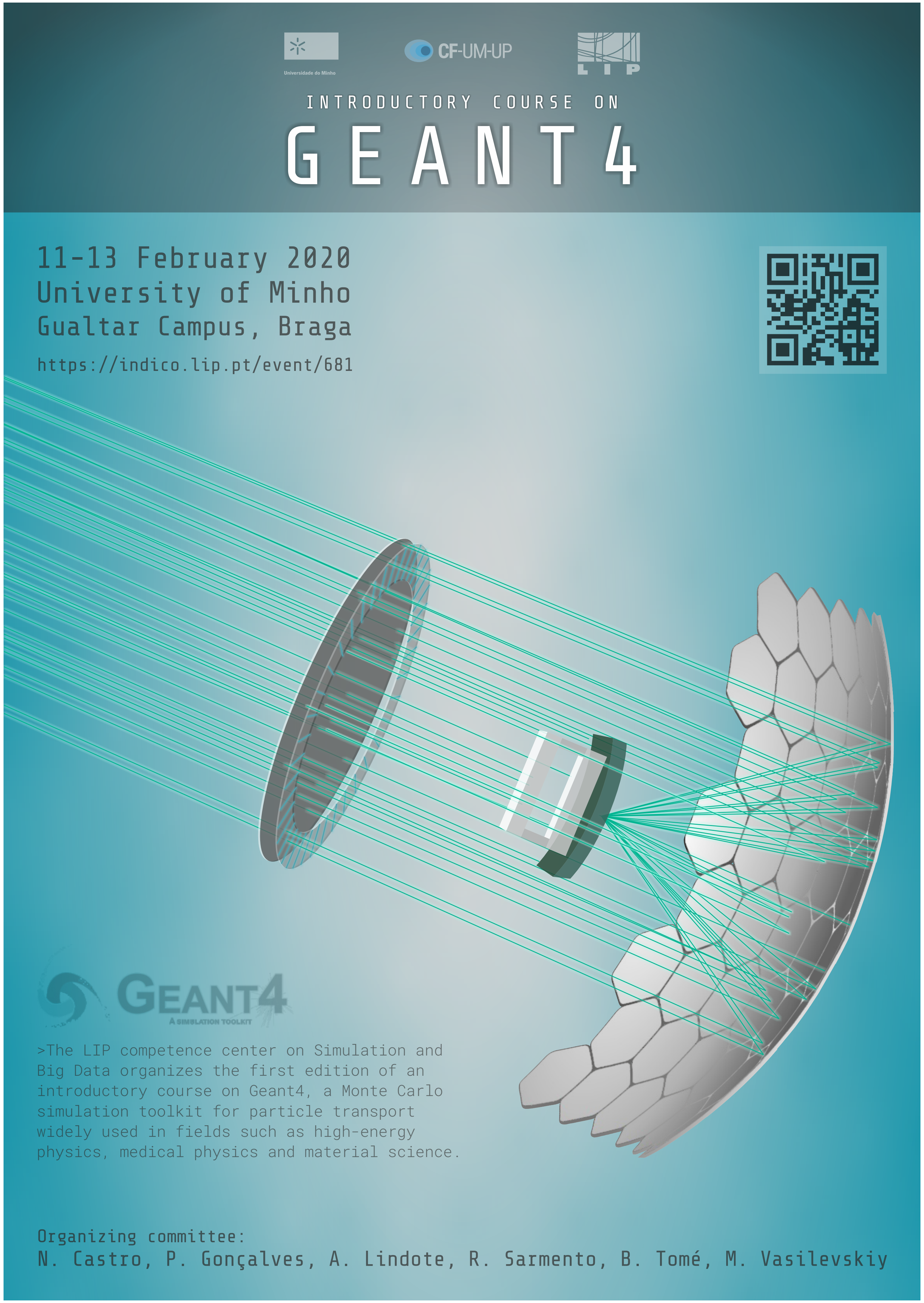
Primary generator



Primary Generator

•Particle transport in a Geant4 simulation starts with the generation of primary particles (or **primaries**)

•Some examples of primary sources typical of Geant4 applications are:

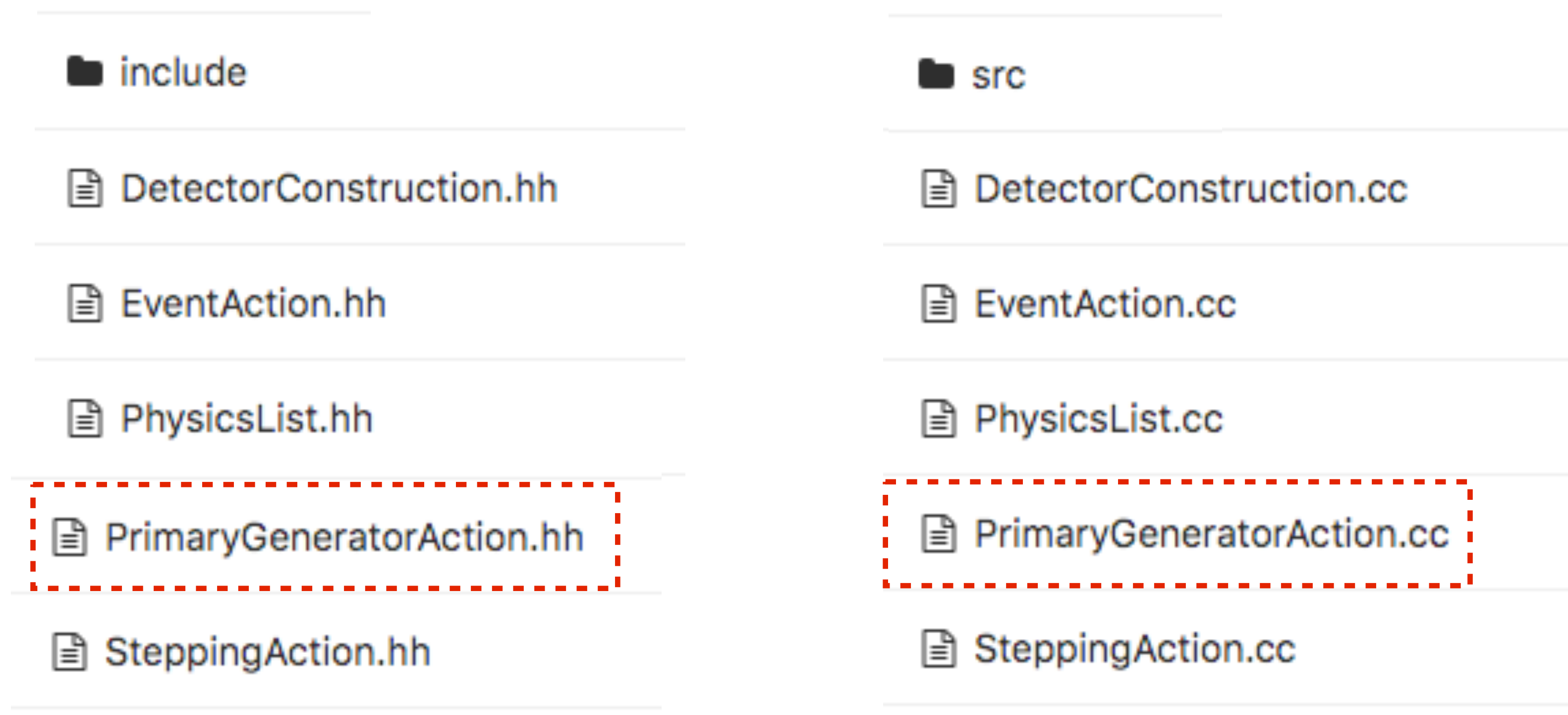
- a **proton beam** used in a hadron collider experiment
- the **atmospheric cosmic rays** at the sea level
- a **laser** used in an optics experiment
- a **radioactive source** used in a medical treatment

Primary Generator

- The application developer must introduce in the code the primaries properties, such as:
  - particle type
  - particle kinematics (energy, direction)
  - others (charge, polarization)

- In Geant4, the generation of primaries is controlled by a **mandatory** implementation of the *G4VUserPrimaryGeneratorAction* class

Primary Generator



- Header and implementation files for the primary generator are mandatory in any Geant4 application

Implementation

•What actions must the application developer undertake from scratch, code-wise?

1) selection one generator (derived from G4VPrimaryGenerator):

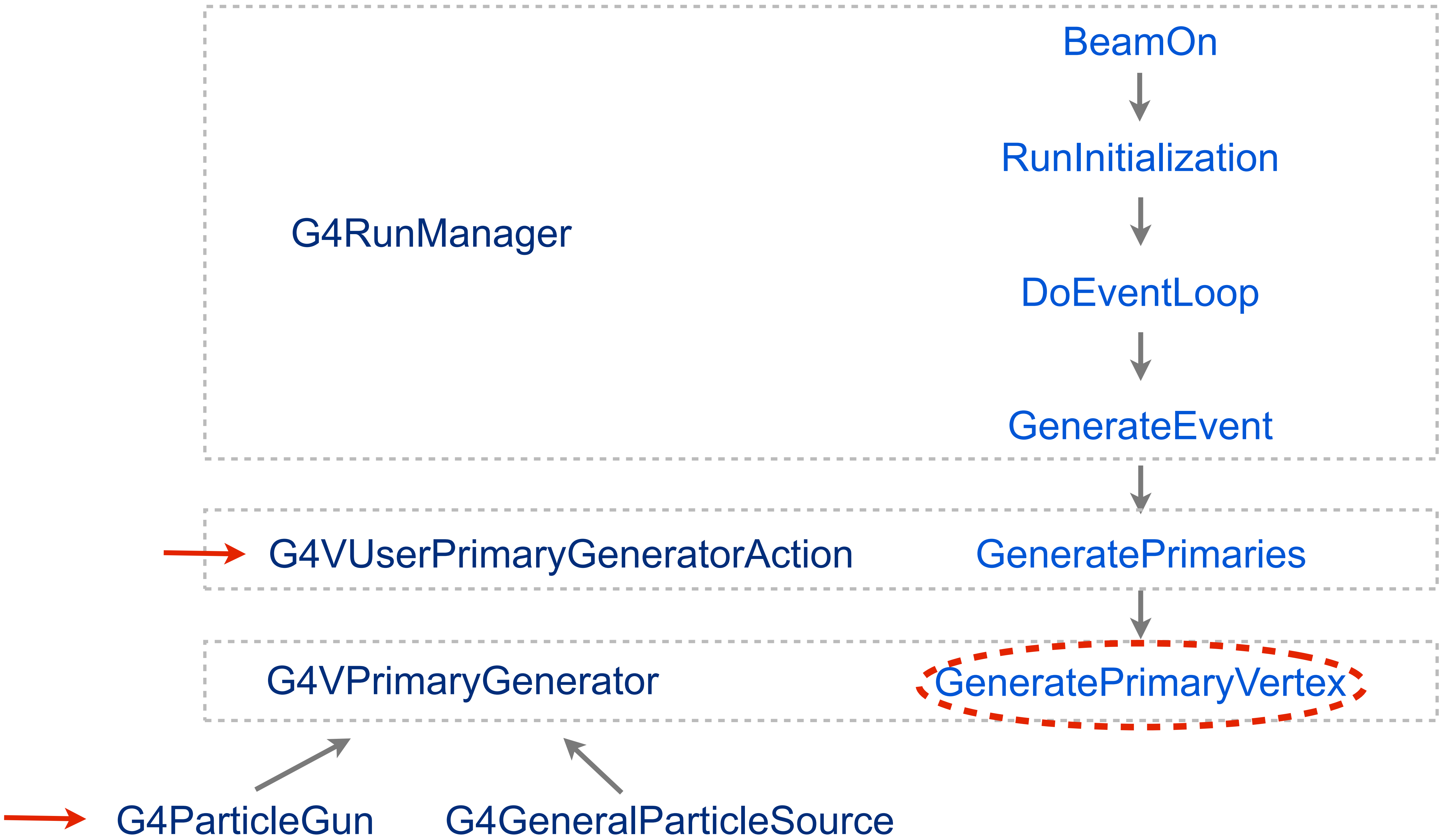
- G4ParticleGun
- G4GeneralParticleSource
- G4HEPEvtInterface

2) define the properties of the source you want to define

3) generate a primary: call GeneratePrimaryVertex at the end of GeneratePrimaries

Behind the scenes

- user code for the primary generator belongs here



G4ParticleGun

- Simplest approach for the definition of primaries
- Set methods to define the particle type, energy, direction, etc.
- It is possible to use randomization on the GeneratePrimaries method

G4ParticleGun: Set/Get methods

From the G4ParticleGun class reference:

•define the various source properties

void	SetParticleDefinition (G4ParticleDefinition *aParticleDefinition)
void	SetParticleEnergy (G4double aKineticEnergy)
void	SetParticleMomentum (G4double aMomentum)
void	SetParticleMomentum (G4ParticleMomentum aMomentum)
void	SetParticleMomentumDirection (G4ParticleMomentum aMomentumDirection)
void	SetParticleCharge (G4double aCharge)
void	SetParticlePolarization (G4ThreeVector aVal)
void	SetNumberOfParticles (G4int i)

•one may also retrieve the source information

G4ParticleDefinition *	GetParticleDefinition () const
G4ParticleMomentum	GetParticleMomentumDirection () const
G4double	GetParticleEnergy () const
G4double	GetParticleMomentum () const
G4double	GetParticleCharge () const
G4ThreeVector	GetParticlePolarization () const
G4int	GetNumberOfParticles () const

G4ParticleGun example

PrimaryGeneratorAction.hh

```
#ifndef PrimaryGeneratorAction_h
#define PrimaryGeneratorAction_h 1

#include "G4VUserPrimaryGeneratorAction.hh"

class DetectorConstruction;
class G4ParticleGun;
class G4Event;

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

class PrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
  public:
    PrimaryGeneratorAction(DetectorConstruction*);
   ~PrimaryGeneratorAction();

  public:
    void GeneratePrimaries(G4Event*);

  private:
    G4ParticleGun* particleGun;
    DetectorConstruction* myDetector;
};

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

#endif
```

A class that inherits from *G4VUserPrimaryGeneratorAction*, which:

The method *GeneratePrimaries* sends primaries to Event object

Requires an instance of the *G4ParticleGun* type

G4ParticleGun example

PrimaryGeneratorAction.cc

```
PrimaryGeneratorAction::PrimaryGeneratorAction(DetectorConstruction* myDC)
:myDetector(myDC)
{
    G4int n_particle = 1;
    particleGun = new G4ParticleGun(n_particle);

    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
    G4ParticleDefinition* particle = particleTable->FindParticle("geantino");
    particleGun->SetParticleDefinition(particle);

    particleGun->SetParticleEnergy(100*keV);
    particleGun->SetParticlePosition(G4ThreeVector(0.,0.,0.));
    particleGun->SetParticleMomentumDirection(G4ThreeVector(1.,0.,0.));

}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

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

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    //create vertex
    particleGun->GeneratePrimaryVertex(anEvent);
}
```

Fixes the properties of the primary particles via the Set methods

At the end, a call to the *GeneratePrimaryVertex* method of your generator

G4ParticleGun example

- You may include randomization on the GeneratePrimaries - information passed before the generation of the primary vertex

- Example for setting a random position of the primary particle:

```
G4double x0  = 4*cm, y0  = 4*cm, z0  = 4*cm;
G4double dx0 = 1*cm, dy0 = 1*cm, dz0 = 1*cm;
x0 += dx0*(G4UniformRand()-0.5);
y0 += dy0*(G4UniformRand()-0.5);
z0 += dz0*(G4UniformRand()-0.5);
fParticleGun->SetParticlePosition(G4ThreeVector(x0,y0,z0));

//create vertex
//
fParticleGun->GeneratePrimaryVertex(anEvent);
```

Particles table

- Define the source particle type with G4ParticleDefinition from the G4ParticleTable

- Types: gluon/quarks/diquarks, leptons, mesons, baryons, ions, others

```
G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
particleTable->DumpTable("ALL");
```

```
--- G4ParticleDefinition ---
 Particle Name : e-
 PDG particle code : 11 [PDG anti-particle code: -11]
 Mass [GeV/c2] : 0.000510999     Width : 0
 Lifetime [nsec] : -1
 Charge [e]: -1
 Spin : 1/2
 Parity : 0
 Charge conjugation : 0
 Isospin : (I,Iz): (0/2 , 0/2 )
 GParity : 0
 MagneticMoment [MeV/T] : -5.79509e-11
 Quark contents     (d,u,s,c,b,t) : 0, 0, 0, 0, 0, 0
 AntiQuark contents               : 0, 0, 0, 0, 0, 0
 Lepton number : 1 Baryon number : 0
 Particle type : lepton [e]
 Stable : stable
```

Radioactive isotopes

•It is possible to define radioactive isotopes from the G4IonTable

•Stores the properties of the decay chains (daughter nuclei, mean life, decay modes, branching ratios, emission spectra) from the selected radioactive isotope

in GeneratePrimaries

```
//fluorine
G4int Z = 9, A = 18;
G4double ionCharge   = 0.*eplus;
G4double excitEnergy = 0.*keV;

G4ParticleDefinition* ion
   = G4IonTable::GetIonTable()->GetIon(Z,A,excitEnergy);
fParticleGun->SetParticleDefinition(ion);
fParticleGun->SetParticleCharge(ionCharge);
```

•Set:
 - Z, A, ion charge, excitation energy

•Energy and momentum direction (=0 if at rest) and position must also be set

General Particle Source

- G4GeneralParticleSource class

- allows for more sophisticated sources

- provides direct implementation for the specification of:

  - spatial sampling
  - angular distribution
  - energy spectrum
  - multiple sources

of the primary particles, also using user-defined histograms

General Particle Source

- implementation via command line, macro file or hard-coded

- hard-coded info: check the class reference; some methods are similar to particle gun

- line commands are those used in a macro

- macro is the best option, we will follow that approach

A GPS macro example

- plane source, gamma, linear energy spectrum, cosine-law angular distribution

particle type

shape and position

angular distribution

energy spectrum

```
# Macro test2.g4mac
/control/verbose 0
/tracking/verbose 0
/event/verbose 0
/gps/verbose 2
/gps/particle gamma
/gps/pos/type Plane
/gps/pos/shape Square
/gps/pos/centre 1 2 1 cm
/gps/pos/halfx 2 cm
/gps/pos/halfy 2 cm
/gps/ang/type cos
/gps/ene/type Lin
/gps/ene/min 2 MeV
/gps/ene/max 10 MeV
/gps/ene/gradient 1
/gps/ene/intercept 1
/run/beamOn 10000
```

From the Application Developers Guide:

•gps commands equivalent to particle gun

Table 2.2: `G4ParticleGun` equivalent commands.

Command	Arguments	Description and restrictions
/gps/List		List available incident particles
/gps/particle	name	Defines the particle type [default *geantino*], using GEANT4 naming convention.
/gps/direction	Px Py Pz	Set the momentum direction [default (1,0,0)] of generated particles using (2.1)
/gps/energy	E unit	Sets the energy [default 1 MeV] for mono-energetic sources. The units can be eV, keV, MeV, GeV, TeV or PeV. (NB: it is recommended to use /gps/ene/mono instead.)
/gps/position	X Y Z unit	Sets the centre co-ordinates (X,Y,Z) of the source [default (0,0,0) cm]. The units can be micron, mm, cm, m or km. (NB: it is recommended to use /gps/pos/centre instead.)
/gps/ion	Z A Q E	After `/gps/particle ion`, sets the properties (atomic number Z, atomic mass A, ionic charge Q, excitation energy E in keV) of the ion.
/gps/ionLvl	Z A Q lvl	After `/gps/particle ion`, sets the properties (atomic number Z, atomic mass A, ionic charge Q, Number of metastable state excitation level (0-9) of the ion.
/gps/time	t0 unit	Sets the primary particle (event) time [default 0 ns]. The units can be ps, ns, us, ms, or s.
/gps/polarization	Px Py Pz	Sets the polarization vector of the source, which does not need to be a unit vector.
/gps/number	N	Sets the number of particles [default 1] to simulate on each event.
/gps/verbose	level	Control the amount of information printed out by the GPS code. Larger values produce more detailed output.

From the Application Developers Guide:

- some of the gps commands for setting the source position

Table 2.4: Source position and structure.

Command	Arguments	Description and restrictions
/gps/pos/type	dist	Sets the source positional distribution type: *Point* [default], *Plane*, *Beam*, *Surface*, *Volume*.
/gps/pos/shape	shape	Sets the source shape type, after `/gps/pos/type` has been used. For a Plane this can be *Circle*, *Annulus*, *Ellipse*, *Square*, *Rectangle*. For both Surface or Volume sources this can be *Sphere*, *Ellipsoid*, *Cylinder*, *Para* (parallelepiped).
/gps/pos/centre	X Y Z unit	Sets the centre co-ordinates (X,Y,Z) of the source [default (0,0,0) cm]. The units can only be micron, mm, cm, m or km.
/gps/pos/rot1	$R1_x$ $R1_y$ $R1_z$	Defines the first (x' direction) vector R1 [default (1,0,0)], which does not need to be a unit vector, and is used together with `/gps/pos/rot2` to create the rotation matrix of the shape defined with `/gps/shape`.
/gps/pos/rot2	$R2_x$ $R2_y$ $R2_z$	Defines the second vector R2 in the xy plane [default (0,1,0)], which does not need to be a unit vector, and is used together with `/gps/pos/rot1` to create the rotation matrix of the shape defined with `/gps/shape`.
/gps/pos/halfx	len unit	Sets the half-length in x [default 0 cm] of the source. The units can only be micron, mm, cm, m or km.
/gps/pos/halfy	len unit	Sets the half-length in y [default 0 cm] of the source. The units can only be micron, mm, cm, m or km.
/gps/pos/halfz	len unit	Sets the half-length in z [default 0 cm] of the source. The units can only be micron, mm, cm, m or km.
/gps/pos/radius	len unit	Sets the radius [default 0 cm] of the source or the outer radius for annuli. The units can only be micron, mm, cm, m or km.

Other gps commands also for defining:

- source energy spectra
- source direction and angular distribution
- multiple sources

- or for loading information from user defined histograms

Questions?

Hands-On Session #3

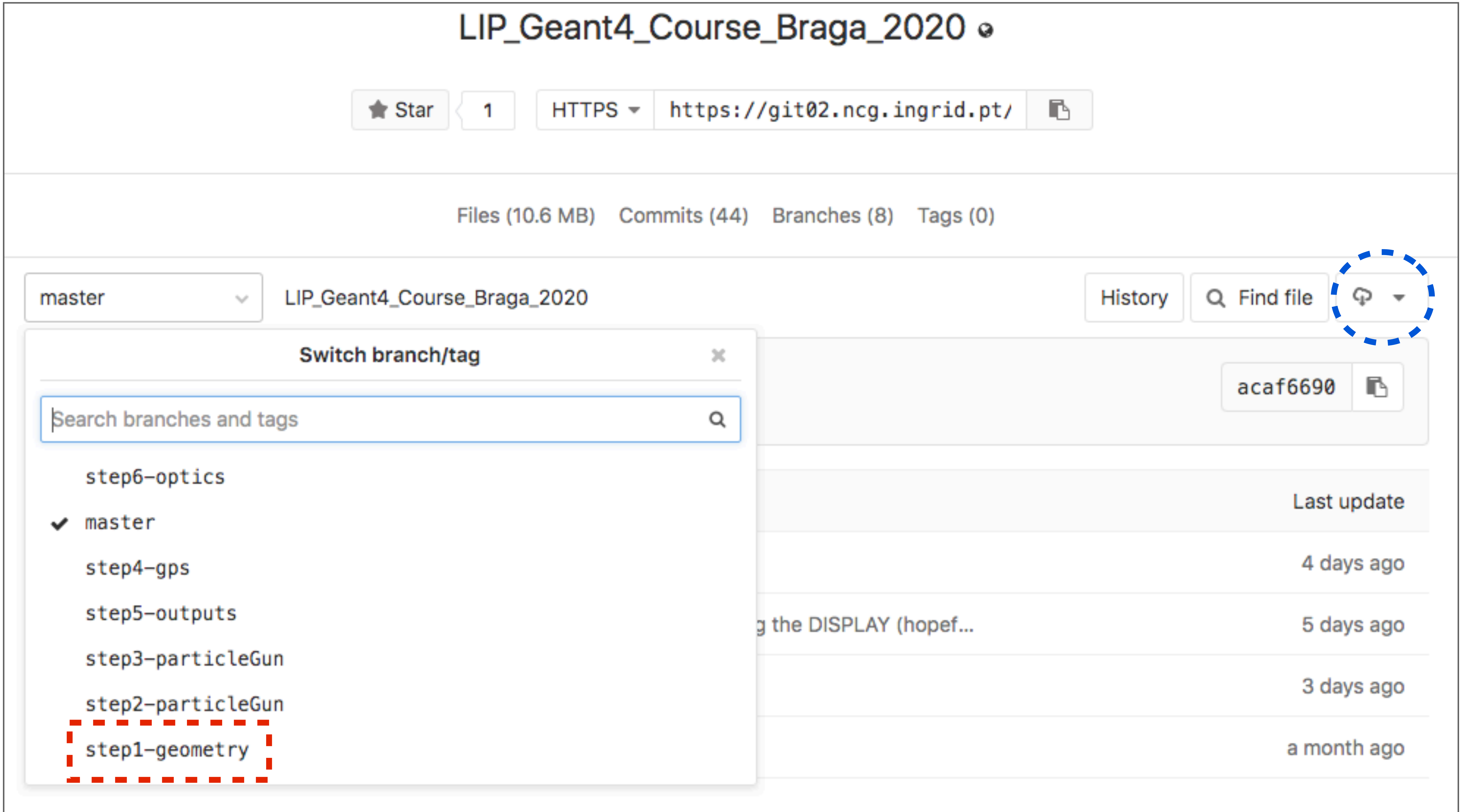
Particle Gun
General Particle Source

- If, by the end of the previous hands-on session, you ran into problems in finalizing the definition of the geometry, you may:

```
git commit -am “My geometry”
git checkout step1-geometry
```

You may check that your source code (e.g. the file DetectorConstruction.cc) has changed

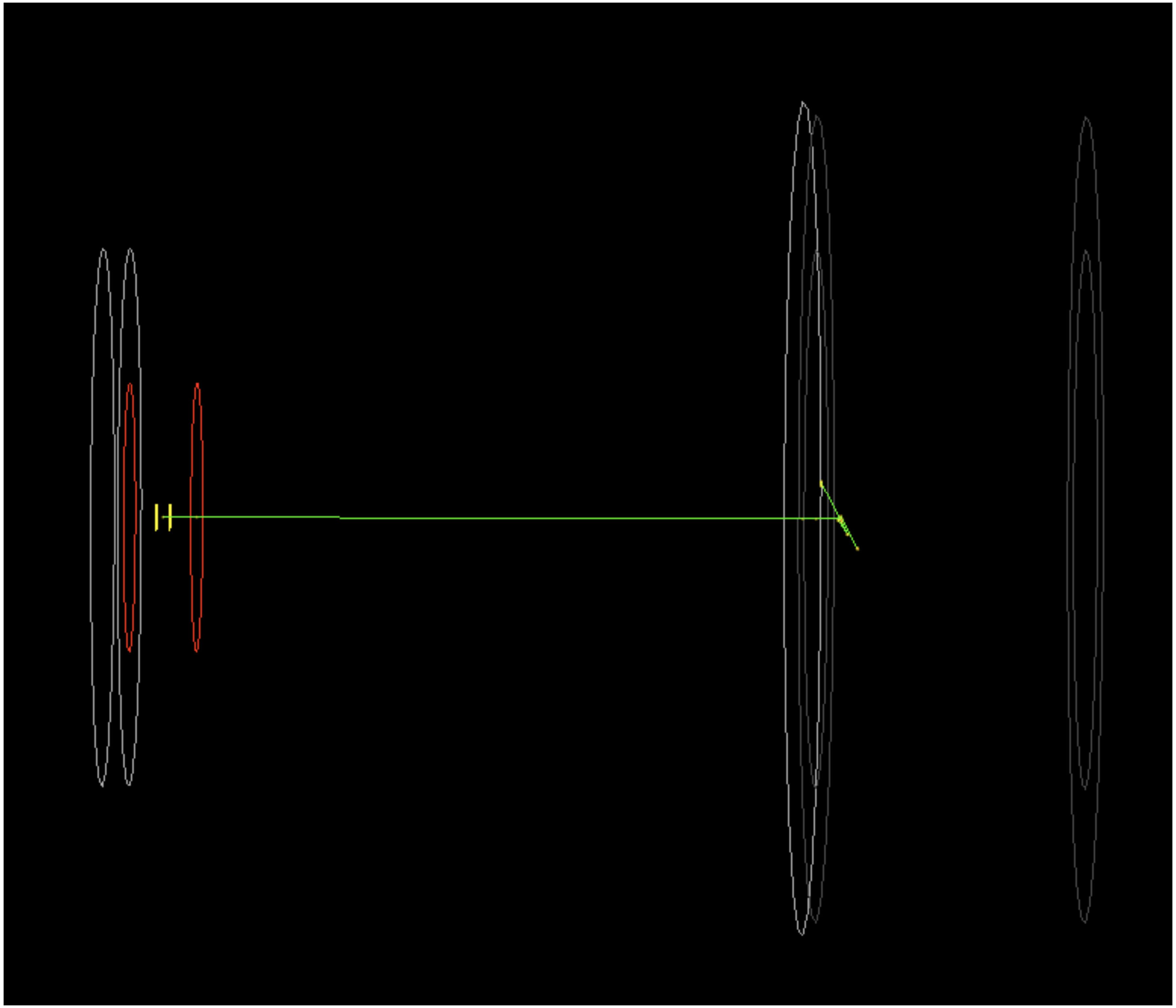
- Alternatively, in the git repository select the branch step1-geometry and download its contents:



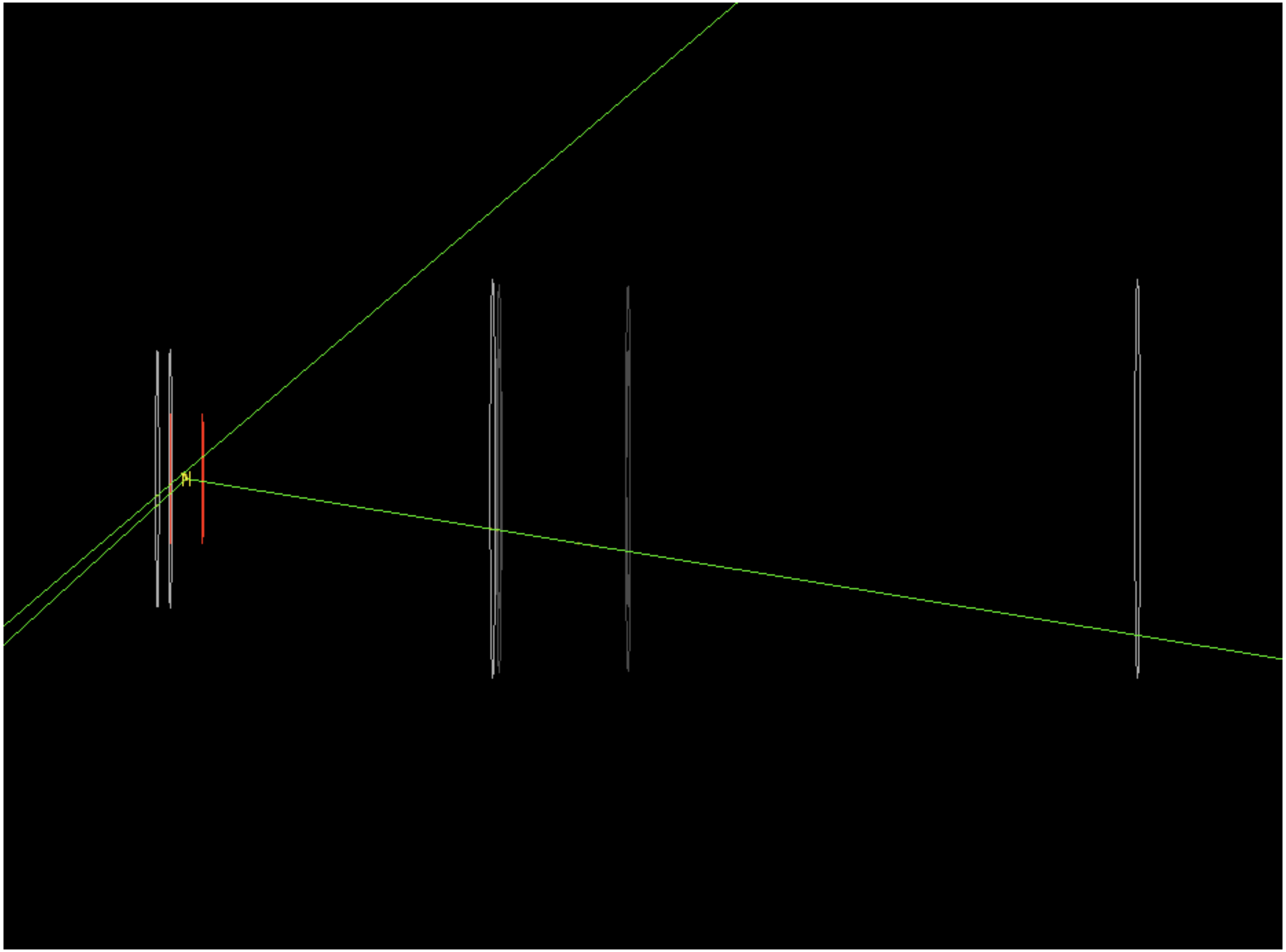
- Use ParticleGun to:
  - generate mono-energetic gammas (2.6 MeV) emitted in the +z direction from the center of the source plastic case
  - now, instead of the gammas, let the radioactive isotope Na-22 decay from the same position

- Use GeneralParticleSource with a macro to let the radioactive isotope Na-22 decay from the active source volume

Mono-energetic gammas from the center of the source plastic case



Na-22 decay from the center of the source plastic case



Na-22 decay from the active source volume (zoomed)
How do you interpret the event?