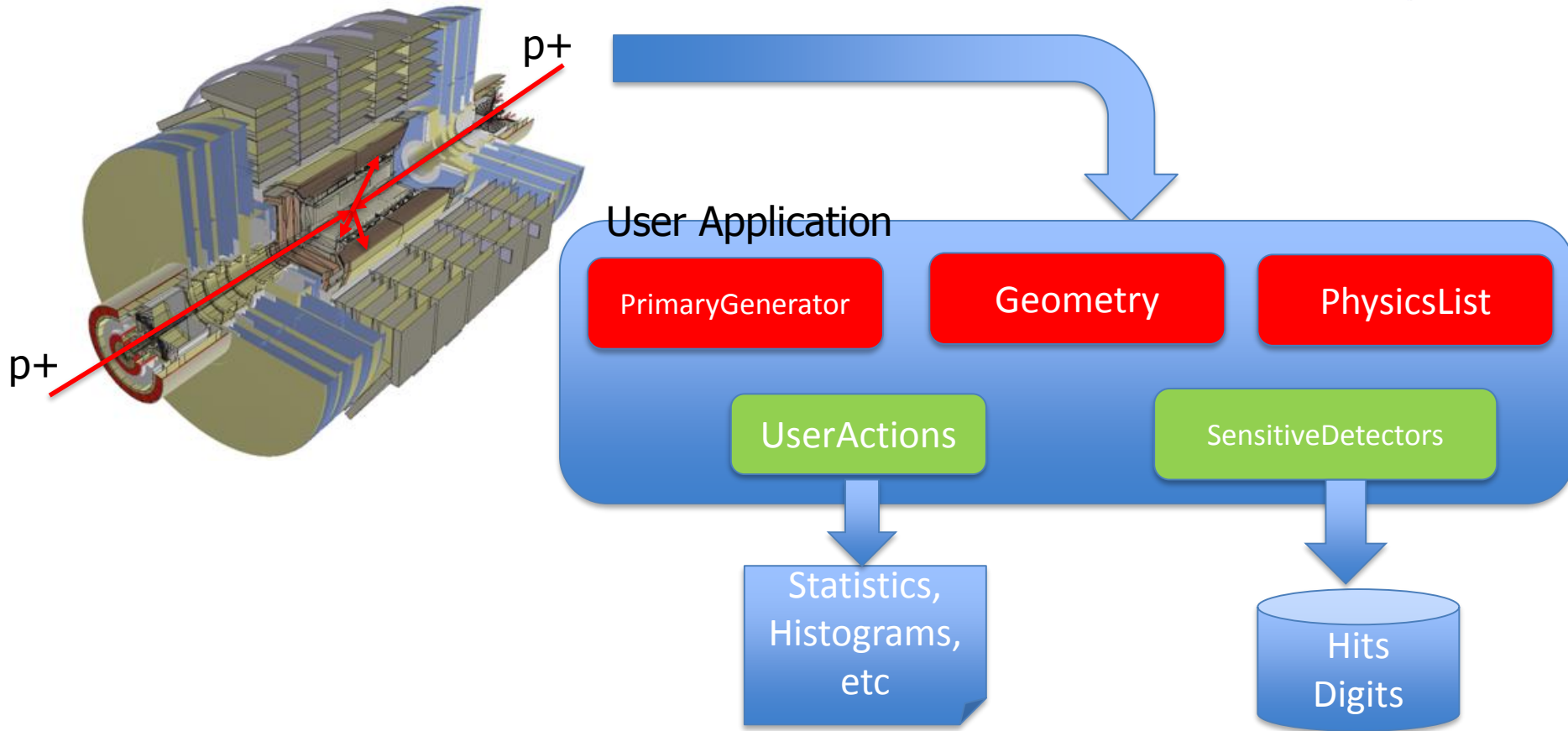


Primary Generators

Witek Pokorski (CERN)
Geant4 Beginners Course
23 January 2019
CERN

These slides include material presented before by M. Asai

What do we need to run simulation?

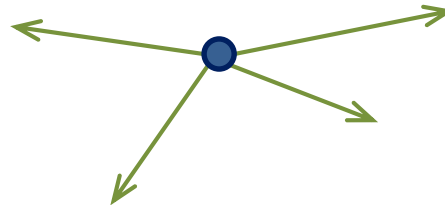


- User needs to provide 'source' of primary particles to Geant4
- Geant4 simulates the passages of those particles through the detector

Primary vertex and primary particle

- **Primary particle(s)** means particle(s) with which you **start an event**.
 - E.g. particles made by the primary p-p collision, an alpha particle emitted from radioactive material, a gamma-ray from treatment head, etc.
 - Then Geant4 tracks these primary particles in your geometry with physics interactions and generates secondaries, detector responses and/or scores.
- **Primary vertex** has position and time. **Primary particle** has a particle ID, momentum and optionally polarization. One or more primary particles may be associated with a primary vertex. One event may have one or more primary vertices.

G4PrimaryVertex objects
= {position, time}



G4PrimaryParticle objects
= {PDG, momentum,
polarization...}

- **Generation of primary vertex/particle** is one of the **user-mandatory tasks**.
G4VUserPrimaryGeneratorAction is the abstract base class to **control** the generation.
 - Actual generation should be delegated to **G4VPrimaryGenerator** class. Several concrete implementations, e.g. **G4ParticleGun**, **G4GeneralParticleSource**, are provided.

G4VUserPrimaryGeneratorAction

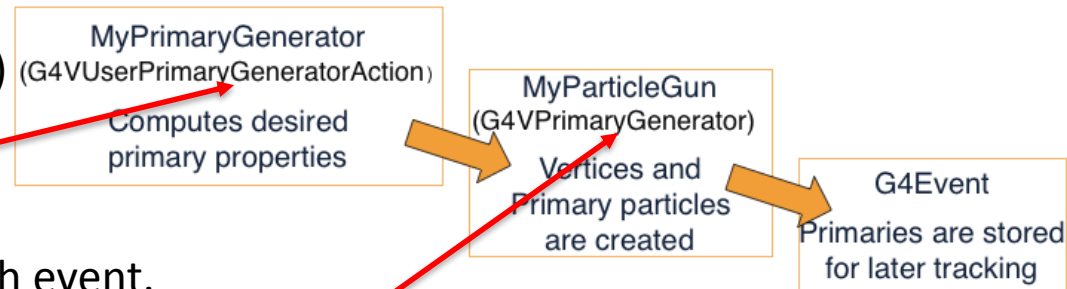
- This class is one of mandatory user classes to **control the generation** of primaries.
 - This class itself **should NOT** generate primaries but **invoke `GeneratePrimaryVertex()`** method of primary generator(s) to make primaries.

- **Constructor**

- Instantiate primary generator(s)
- Set default values to it(them)

- **`GeneratePrimaries()` method**

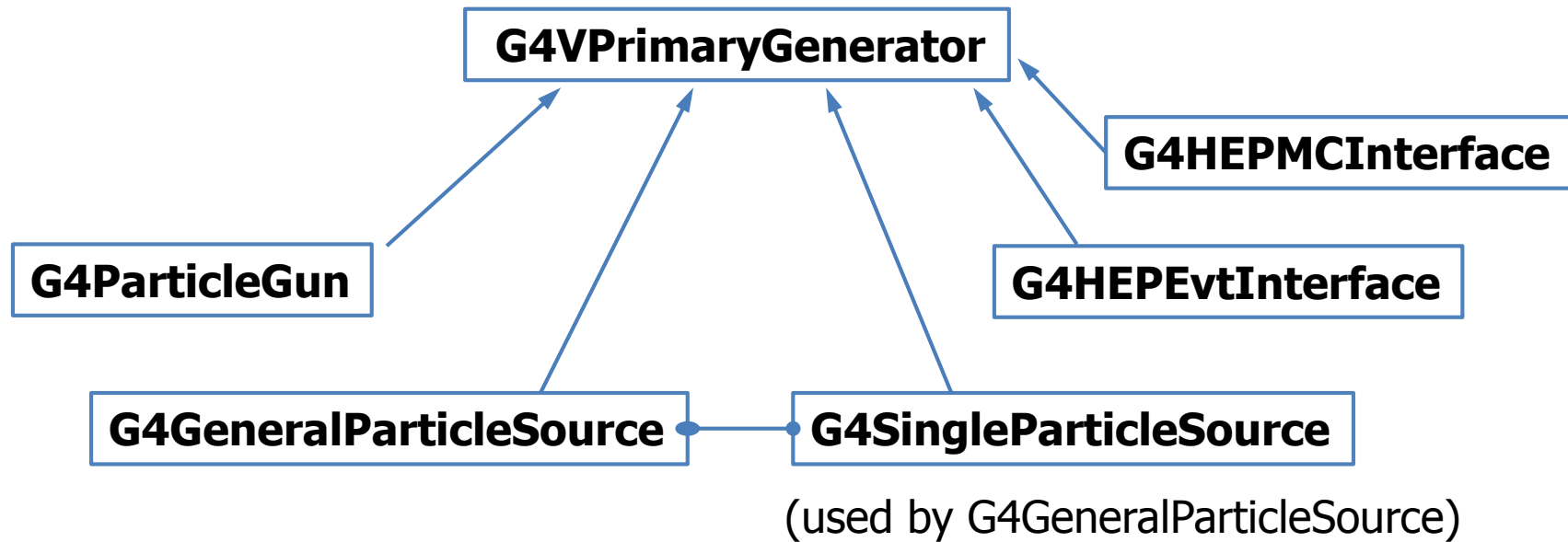
- Invoked at the beginning of each event.
- Randomize particle-by-particle value(s)
- Set these values to primary generator(s)
 - Never use hard-coded UI commands
- Invoke **`GeneratePrimaryVertex()`** method of primary generator(s)



- Your concrete class of **`G4VUserPrimaryGeneratorAction`** must be instantiated in the **`Build()`** method of your **`G4VUserActionInitialization`**

Constructor : Invoked only once	{	MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
		{
		G4int n_particle = 1;
		fparticleGun = new G4ParticleGun(n_particle);
		 // default particle kinematic
		G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
		G4ParticleDefinition* particle = particleTable->FindParticle("gamma");
		fparticleGun->SetParticleDefinition(particle);
		fparticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
		fparticleGun->SetParticleEnergy(100.*MeV);
		fparticleGun->SetParticlePosition(G4ThreeVector(0.,0.,-50*cm));
		}
Invoked once per each event	{	void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
		{
		fparticleGun->SetParticleMomentum(G4RandomDirection());
		fparticleGun->GeneratePrimaryVertex(anEvent);
		}

Built-in primary particle generators



- Concrete implementations of **G4VPrimaryGenerator**
 - A good example for experiment-specific primary generator implementation
- It shoots one primary particle of a certain energy from a certain point at a certain time to a certain direction.
 - Various set methods are available
 - Intercoms commands are also available for setting initial values
- One of most frequently asked questions is :
I want “particle shotgun”, “particle machinegun”, etc.
- Instead of implementing such a fancy weapon, in your implementation of **UserPrimaryGeneratorAction**, you can
 - Shoot random numbers in arbitrary distribution
 - Use set methods of **G4ParticleGun**
 - Use **G4ParticleGun** as many times as you want
 - Use any other primary generators as many times as you want to make overlapping events

What to do and where to do

- In the constructor of your UserPrimaryGeneratorAction
 - Instantiate **G4ParticleGun**
 - Set default values by set methods of **G4ParticleGun**
 - Particle type, kinetic energy, position and direction
- In your macro file or from your interactive terminal session
 - Set values for a run
 - Particle type, kinetic energy, position and direction
- In the **GeneratePrimaries()** method of your UserPrimaryGeneratorAction
 - Shoot random number(s) and prepare track-by-track or event-by-event values
 - Kinetic energy, position and direction
 - Use set methods of **G4ParticleGun** to set such values
 - Then invoke **GeneratePrimaryVertex()** method of G4ParticleGun
 - If you need more than one primary tracks per event, loop over randomization and GeneratePrimaryVertex().
- examples/basic/B5/src/B5PrimaryGeneratorAction.cc is a good example to start with.

```
void T01PrimaryGeneratorAction::
    GeneratePrimaries(G4Event* anEvent)
{ G4ParticleDefinition* particle;
  G4int i = (int)(5.*G4UniformRand());
  switch(i)
  { case 0: particle = positron; break; ... }
  particleGun->SetParticleDefinition(particle);
  G4double pp =
    momentum+(G4UniformRand()-0.5)*sigmaMomentum;
  G4double mass = particle->GetPDGMass();
  G4double Ekin = sqrt(pp*pp+mass*mass)-mass;
  particleGun->SetParticleEnergy(Ekin);
  G4double angle = (G4UniformRand()-0.5)*sigmaAngle;
  particleGun->SetParticleMomentumDirection
    (G4ThreeVector(sin(angle),0.,cos(angle)));
  particleGun->GeneratePrimaryVertex(anEvent);
}
```

- You can repeat this for generating more than one primary particles.

- Other concrete implementations of **G4VPrimaryGenerator** ready to use
 - A good example for experiment-specific primary generator implementation
- **G4HEPEvtInterface**
 - Suitable to /HEPEVT/ common block, which many of (FORTRAN) HEP physics generators are compliant to.
 - ASCII file input
- **G4HepMCInterface**
 - An interface to HepMC class, which a few new (C++) HEP physics generators are compliant to.
 - ASCII file input or direct linking to a generator through HepMC.

- Yet another concrete implementation of **G4VPrimaryGenerator**
 - Suitable especially to space applications

```
MyPrimaryGeneratorAction::
```

```
    MyPrimaryGeneratorAction()
```

```
{ generator = new G4GeneralParticleSource; }
```

```
void MyPrimaryGeneratorAction::
```

```
    GeneratePrimaries(G4Event* anEvent)
```

```
{ generator->GeneratePrimaryVertex(anEvent); }
```

- Detailed description

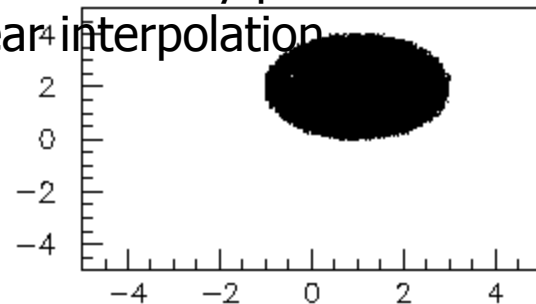
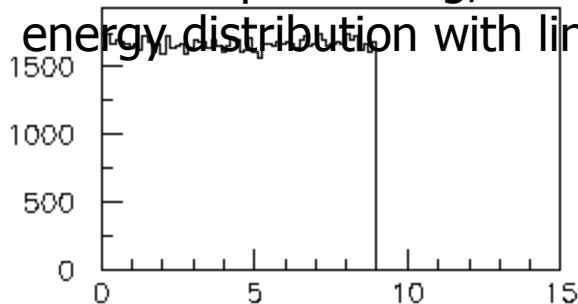
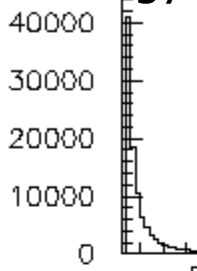
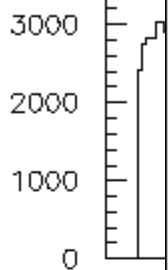
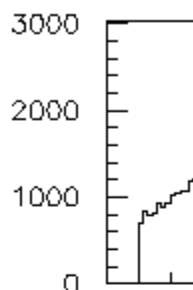
[Section 2.7 of Application Developer's Guide](#)

Square

Spherical

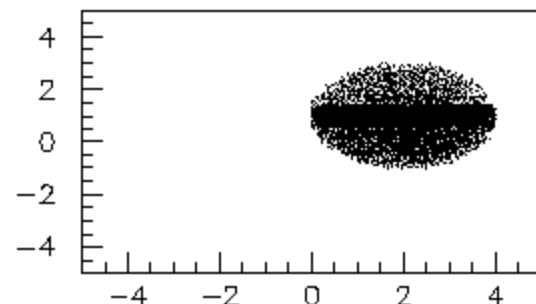
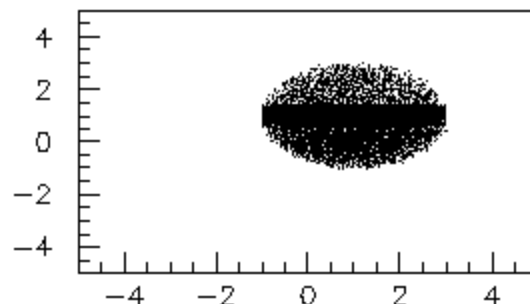
Cylindrical

Spherical volume with z biasing, isotropic radiation with theta and phi biasing, integral arbitrary point-wise energy distribution with linear interpolation



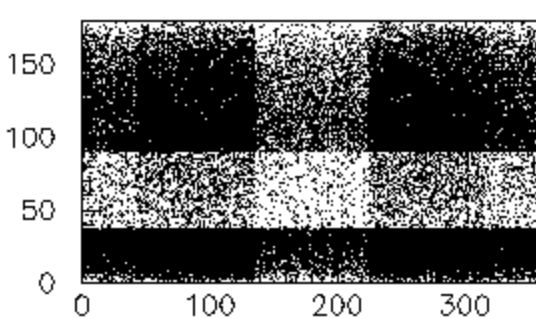
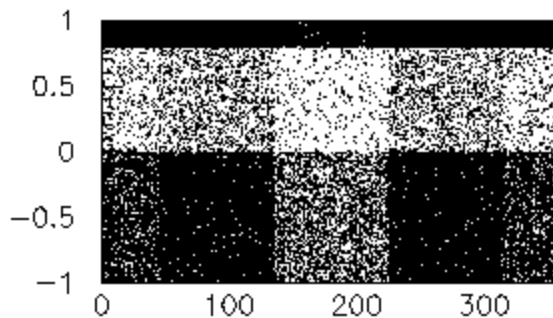
Source Energy Spectrum

Source X-Y distribution



Source X-Z distribution

Source Y-Z distribution



Source $\cos(\theta)$ - ϕ distribution

Source θ/ϕ distribution

Example commands of General Particle Source

two beams in a generator

#

beam #1

default intensity is 1 now change to 5.

/gps/source/intensity 5.

#

/gps/particle proton

/gps/pos/type Beam

#

the incident surface is in the y-z plane

/gps/pos/rot1 0 1 0

/gps/pos/rot2 0 0 1

#

the beam spot is centered at the origin and is of

1d gaussian shape with a 1 mm central plateau

/gps/pos/shape Circle

/gps/pos/centre 0. 0. 0. mm

/gps/pos/radius 1. mm

/gps/pos/sigma_r .2 mm

#

the beam is travelling along the X_axis with

5 degrees dispersion

/gps/ang/rot1 0 0 1

/gps/ang/rot2 0 1 0

/gps/ang/type beam1d

/gps/ang/sigma_r 5. deg

#

the beam energy is in gaussian profile

centered at 400 MeV

/gps/ene/type Gauss

/gps/ene/mono 400 MeV

/gps/ene/sigma 50. MeV

(macro continuation...)

beam #2

2x the intensity of beam #1

/gps/source/add 10.

#

this is a electron beam

/gps/particle e-

/gps/pos/type Beam

it beam spot is of 2d gaussian profile

with a 1x2 mm2 central plateau

it is in the x-y plane centred at the origin

/gps/pos/centre 0. 0. 0. mm

/gps/pos/halfx 0.5 mm

/gps/pos/halfy 1. mm

/gps/pos/sigma_x 0.1 mm

the spread in y direction is stronger

/gps/pos/sigma_y 0.2 mm

#

#the beam is travelling along -Z_axis

/gps/ang/type beam2d

/gps/ang/sigma_x 2. deg

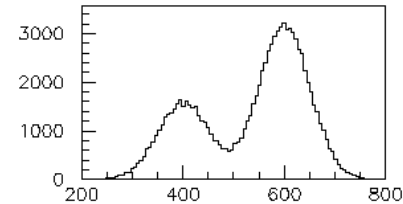
/gps/ang/sigma_y 1. deg

gaussian energy profile

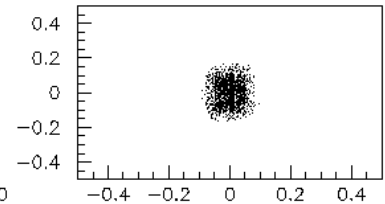
/gps/ene/type Gauss

/gps/ene/mono 600 MeV

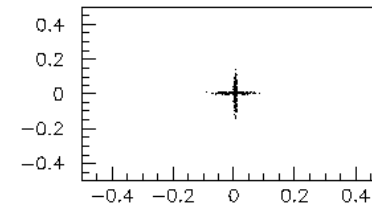
/gps/ene/sigma 50. MeV



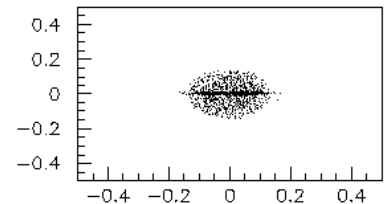
Source Energy Spectrum



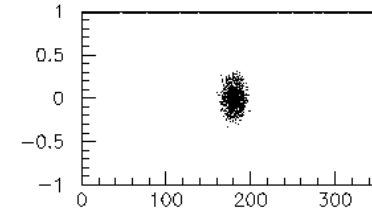
Source X-Y distribution



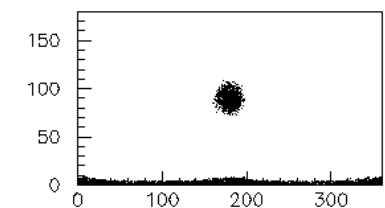
Source X-Z distribution



Source Y-Z distribution



Source cos(theta)-phi distribution



Source theta/phi distribution

- **Particle Gun**

- Simple and naïve
- Shoot one track at a time
- Easy to handle.
 - Use set methods to alternate track-by-track or event-by-event values.

- **General Particle Source**

- Powerful
- Controlled by UI commands.
 - Almost impossible to control through set methods
- Capability of shooting particles from a surface of a volume.
- Capability of randomizing kinetic energy, position and/or direction following a user-specified distribution (histogram).

- If you need to shoot primary particles from a surface of a volume, either outward or inward, GPS is the choice.
- If you need a complicated distribution, not flat or simple Gaussian, GPS is the choice.
- Otherwise, use Particle Gun.

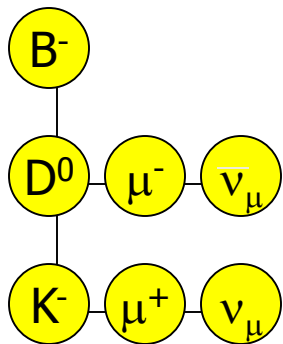
Pre-assigned decay

- By default, when an unstable particle comes to its decay point, **G4DecayProcess** looks up the decay table defined in the **G4ParticleDefinition** of this particle type and randomly selects a decay channel.
- Alternatively, you may define a particular decay channel to **G4PrimaryParticle**.
 - Then, **G4DecayProcess** takes that channel without looking up the decay table and Lorentz-boost.
- Two major use cases.
 - Shooting exotic primary particle, e.g. Higgs. Geant4 does not know how to decay Higgs, thus you have to define the decay daughters.
 - Forcing decay channel for each particle, e.g. forcing a rare channel

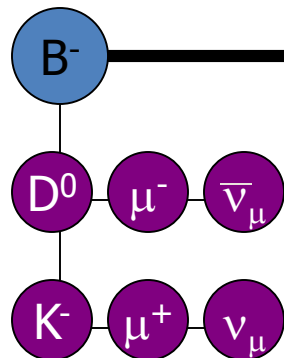
Pre-assigned decay products

- Physics generator can assign a decay channel for **each individual particle separately**.
 - Decay chain can be “pre-assigned”.
- A parent particle in the form of G4Track object travels in the detector, bringing “pre-assigned” decay daughters as objects of G4DynamicParticle.
 - When the parent track comes to the decay point, pre-assigned daughters become to secondary tracks, instead of randomly selecting a decay channel defined to the particle type. Decay time of the parent can be pre-assigned as well.

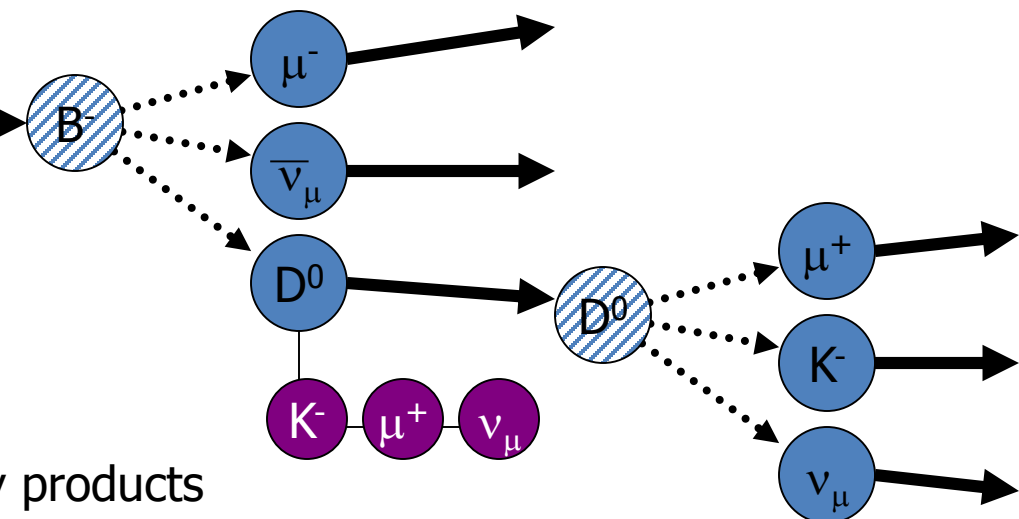
G4PrimaryParticle



G4Track



pre-assigned decay products



- User primary generator action is a **mandatory class** that user must implement
 - this class can re-use existing primary generators
 - it plays the role of providing ‘primary particles’ that Geant4 transports through the detector
- ‘**particle guns**’ used for test-beam or fixed target simulations
- **General Particle Source** capable of shooting particles from a surface of a volume
 - useful for space applications, medical applications, etc
- **interface to HepMC** event record used for MC event generators