Introducción a Geant 4

E. Nácher, J. Balibrea

IFIC
INSTITUT DE FÍSICA CORPUSCULAR

Contenidos

- Características generales de Geant4
- Conceptos básicos en Geant4
- Estructura global de Geant4
- Ejemplo sencillo completo

Características generales de Geant4

¿Qué es Geant4?

- Geant4 es un conjunto de clases de C++ para la simulación Montecarlo de detectores en HEP. Es el sucesor de GEANT3, las librerías de simulación en Fortran77 incluidas *CERNLIB*.

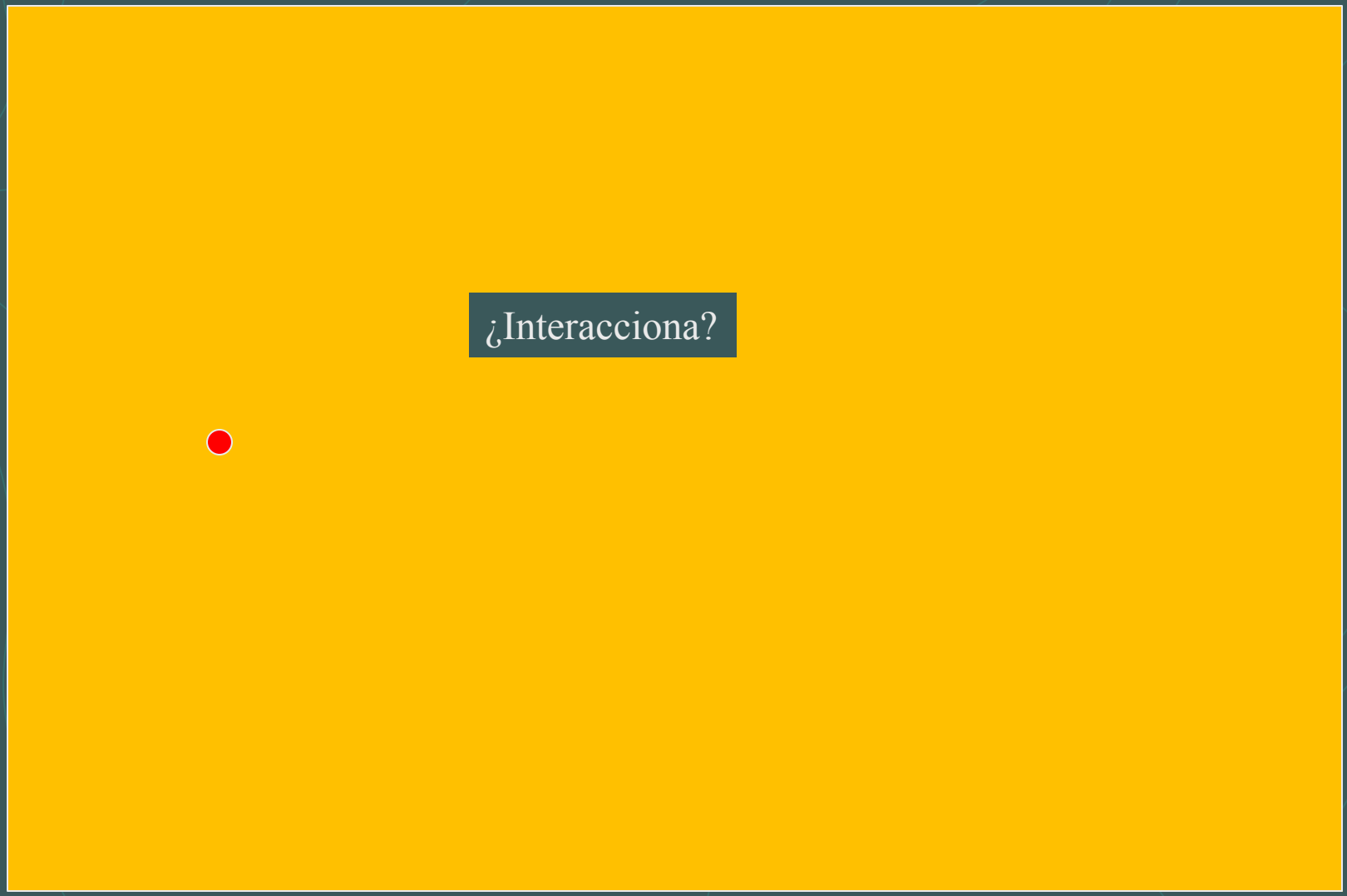
- Geant4 es el rediseño de un gran paquete de software de HEP para adaptarlo a las nuevas necesidades experimentales usando un entorno *Orientado a Objetos*: se ha pasado del Fortran77 al C++

- Se ha aumentado el grado de funcionalidad y flexibilidad para cumplir muchos requisitos impuestos por la física nuclear, física médica, astrofísica...

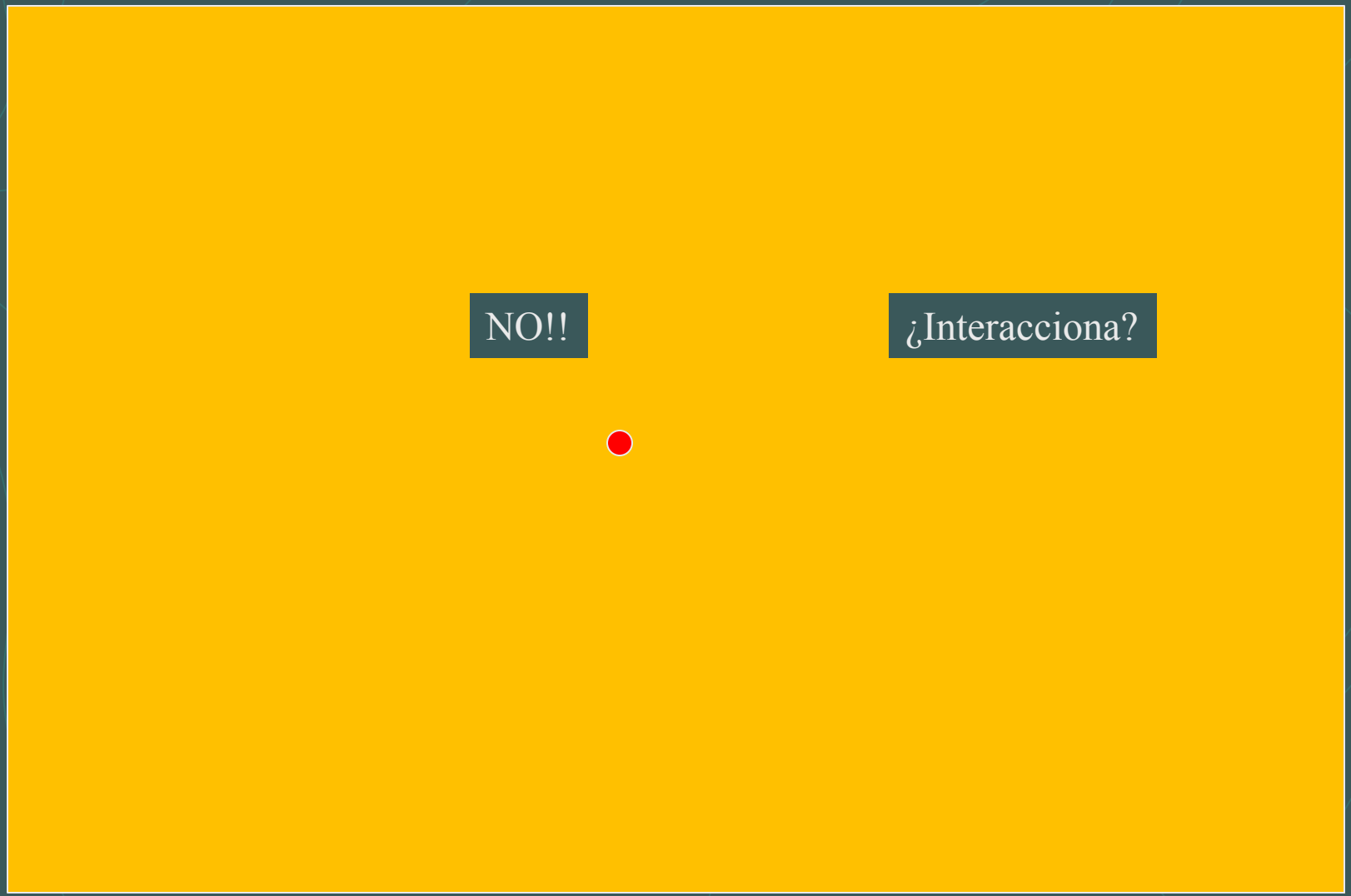
¿Qué hace Geant4?

- Geant4, a través de los procedimientos definidos en sus clases:
  - Hace un seguimiento (*tracking*) de cada partícula primaria a través de los distintos volúmenes definidos por el usuario en pasos de longitud dinámica
  - En cada paso calcula: si se produce o no interacción, tipo de interacción, energía depositada en caso de que la haya, nueva dirección de la partícula, partículas secundarias generadas
  - Hace un seguimiento completo de todas las partículas secundarias generadas
  - Termina este *tracking* cuando la partícula tiene un rango en el material menor de un cierto valor: Cut

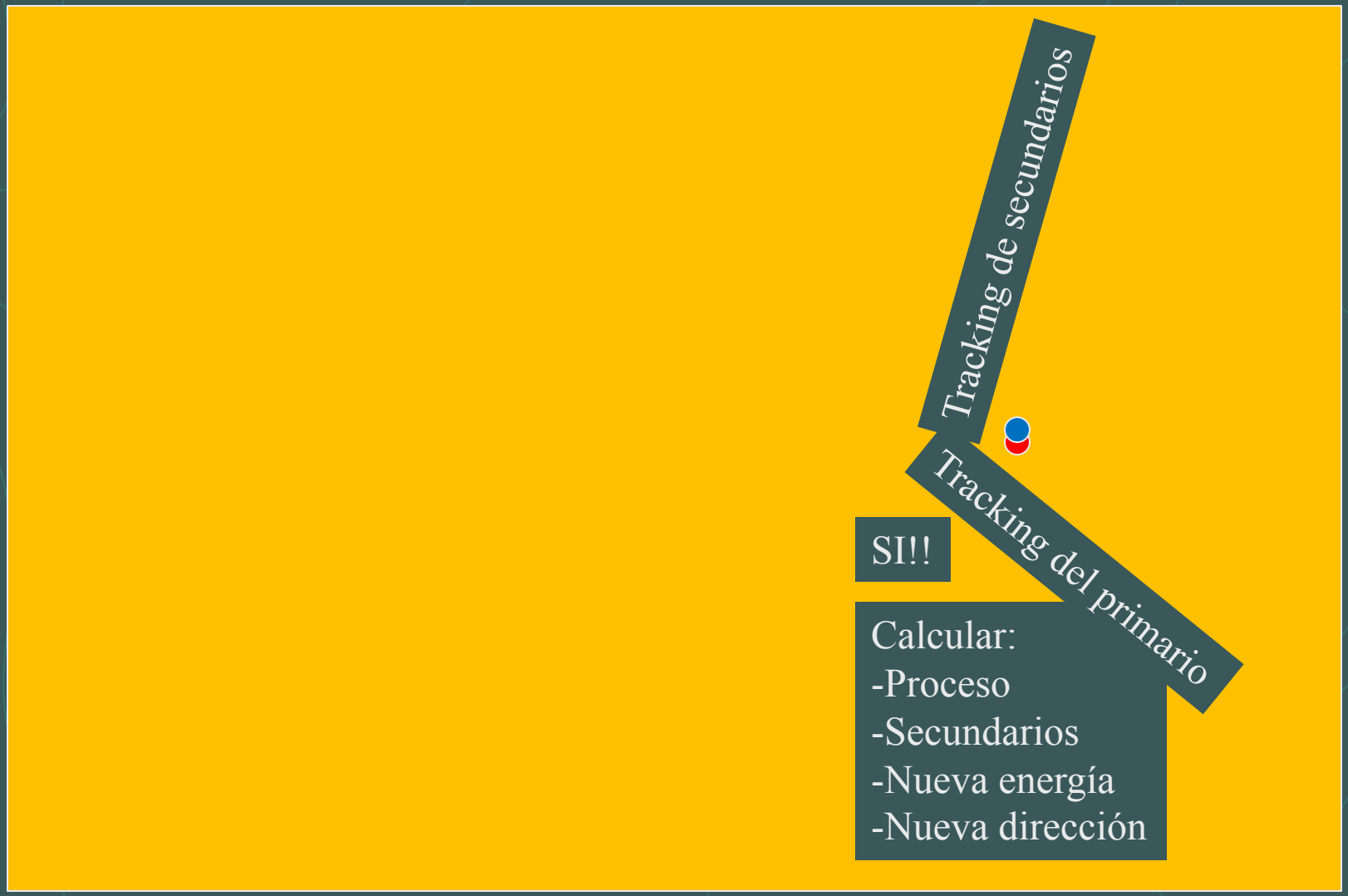
¿Qué hace Geant4?



¿Qué hace Geant4?



¿Qué hace Geant4?



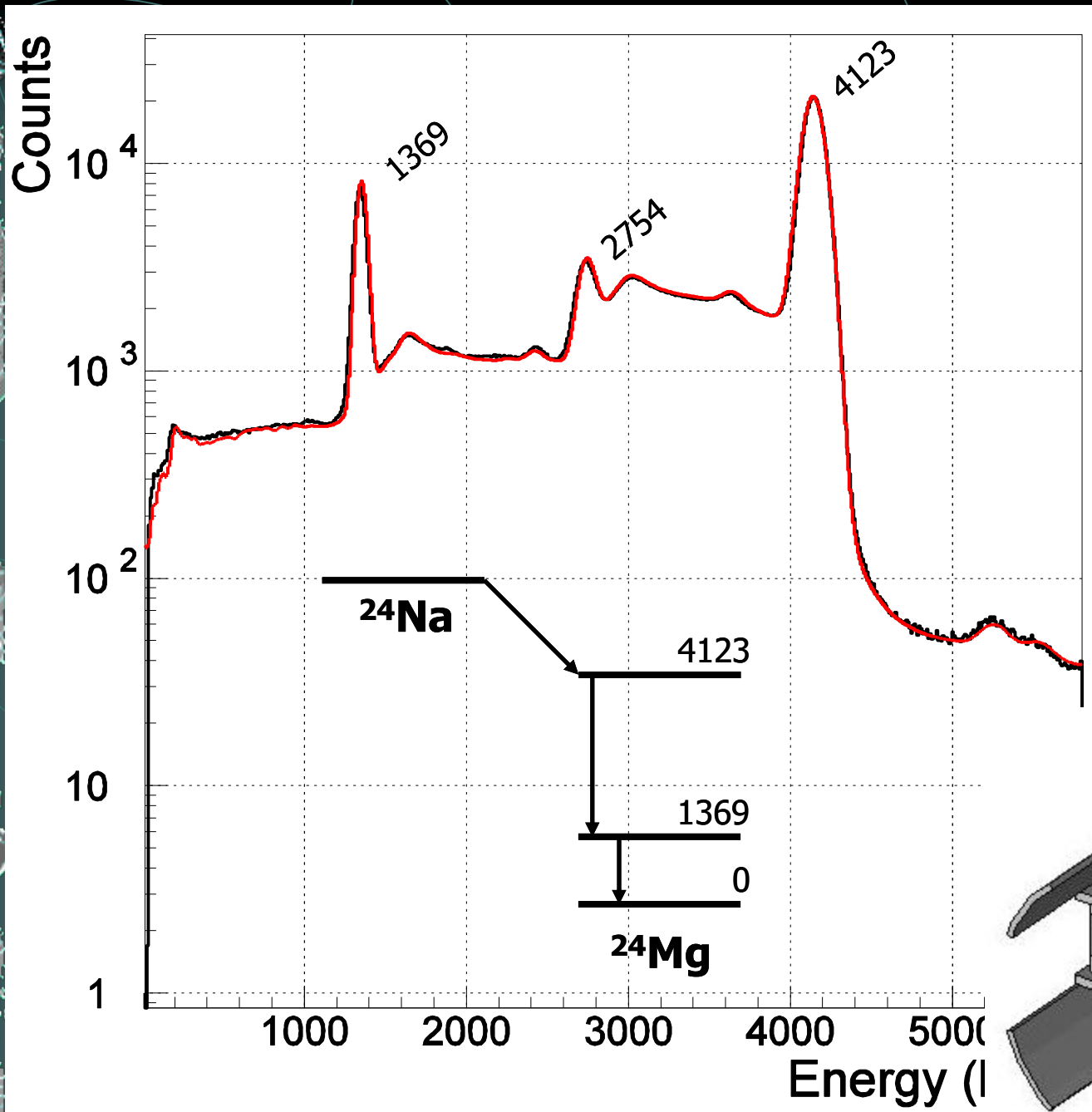
Flexibilidad de Geant4

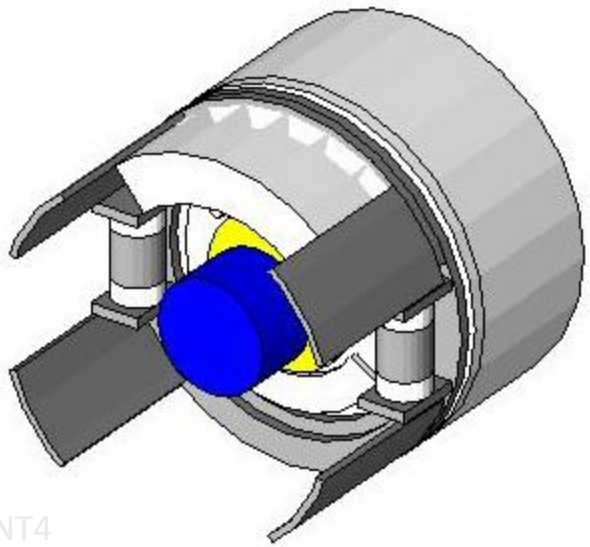
- G4 proporciona un alto grado de flexibilidad (a veces demasiado!!) a distintos niveles:
  - Geometría: se pueden definir formas y materiales tan complicados como se quiera. Se pueden utilizar operaciones booleanas, parametrizaciones…
  - Física: Se puede (debe!!) elegir el modelo más adecuado a la simulación particular (rango energético), así como 'conectar' o 'desconectar' distintos procesos físicos.
  - Se pueden usar distintos tipos de (G)UI, drivers de Visualización, Herramientas de análisis, almacenado de datos…

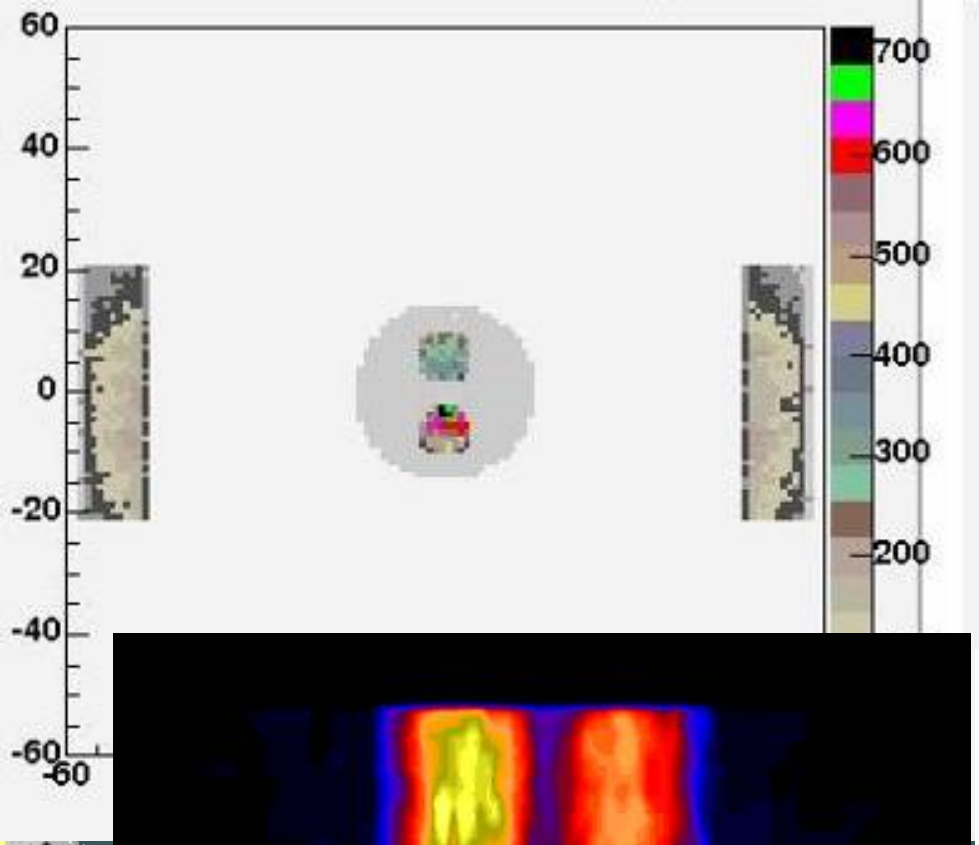
Física en Geant4

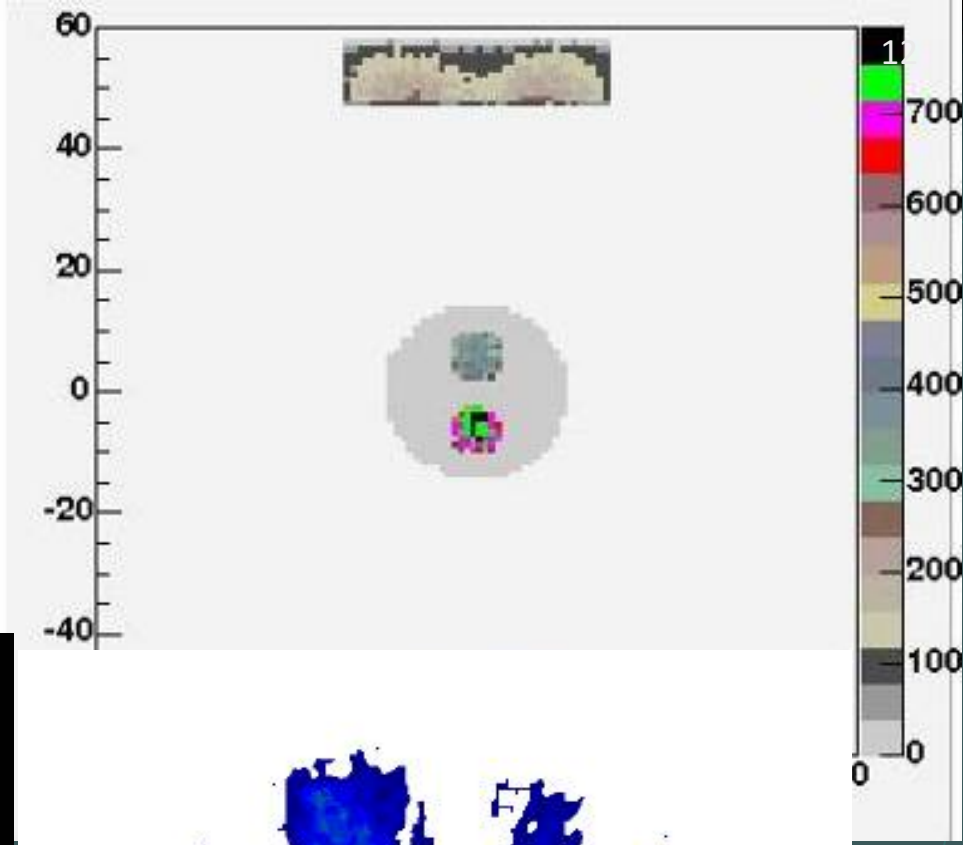
- No se puede desarrollar un modelo válido para todo tipo de partículas y rango de energía
- Parte de la física implementada viene de teoría, otra parte de tablas de datos experimentales (secciones eficaces…), y otra de modelos fenomenológicos
- Cada tabla de datos o modelo físico tiene su propio rango energético de aplicación. Combinando varios se puede dar cobertura a un amplio rango de energías y partículas para distintas simulaciones
- El usuario elige el modelo más adecuado a la simulación particular (rango energético), así como los distintos procesos físicos relevantes (¿demasiada flexibilidad?)

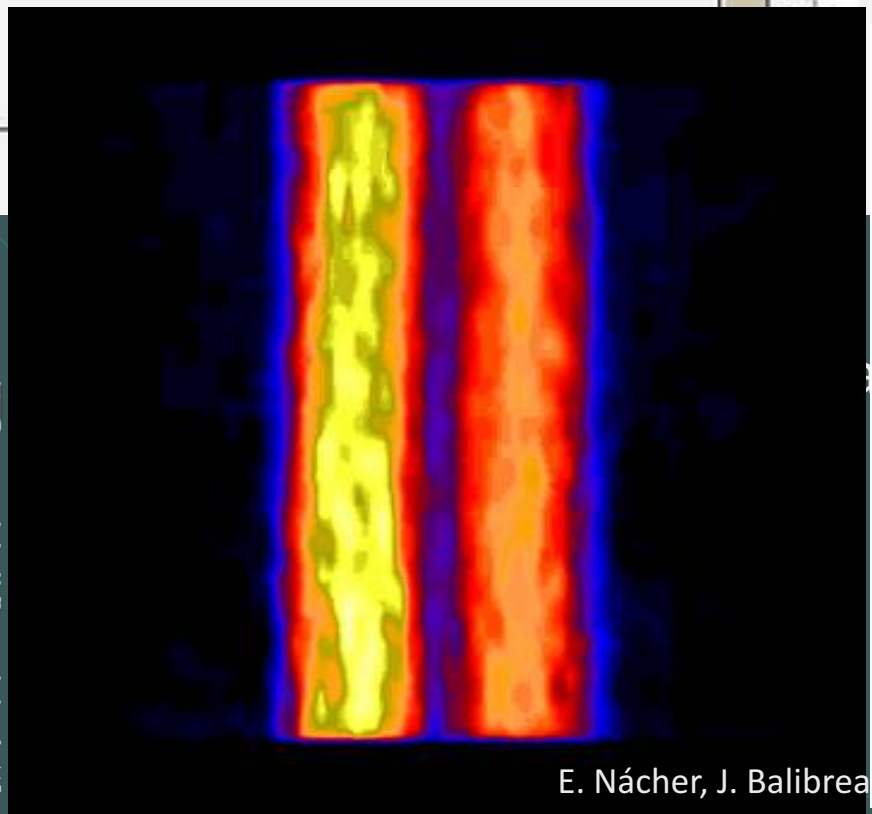
de datos
respuesta

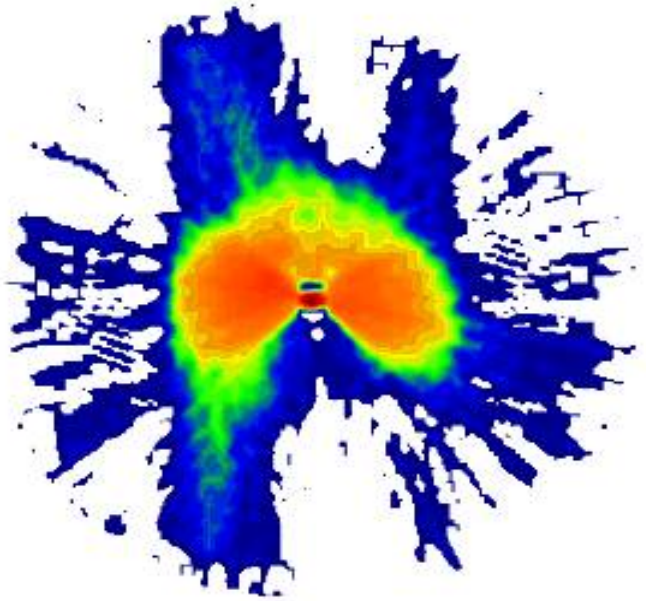





60
40
20
0
-20
-40
-60
700
600
500
400
300
200


60
40
20
0
-20
-40
700
600
500
400
300
200
100
0





Conceptos básicos en Geant4

Run

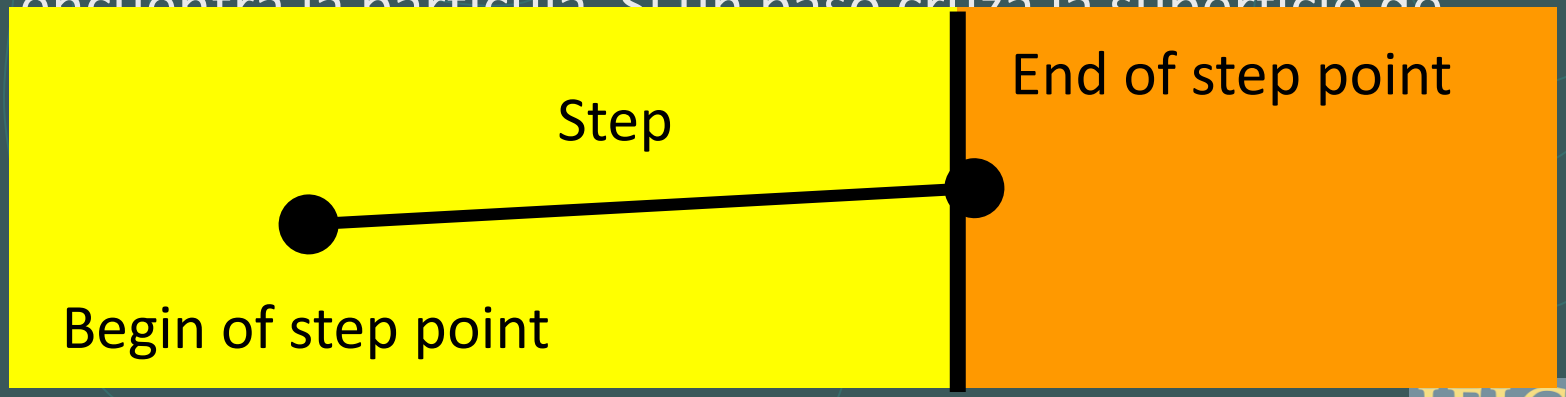
- Como analogía con los experimentos en HEP, un *run* de Geant4 comienza con: “Beam On”.
- Conceptualmente, el *run* es una colección de *eventos* que comparten el mismo detector y condiciones físicas.
- Durante el *run*, el usuario no puede cambiar:
  - La geometría del sistema de detección
  - La física de la simulación

Event

- Al principio de su procesado, un *event* contiene partículas primarias (y su posición, momento inicial y energía). Estos primarios se 'lanzan', y se crea un 'stack' que contiene toda la información del evento.
- Cuando termina el procesado del evento se vacía el 'stack'.
- La clase G4Event representa un evento. Contiene los siguientes objetos al final de su procesado:
  - Lista de vértices y partículas primarias
  - Colección de Trayectorias (opcional)
  - Colección de Hits y partículas secundarias …
  - La información que el usuario quiera añadir (p.ej. acumulada en cada *step*)

Step

- Cada *step* tiene dos puntos (inicio y final), además de información tipo: “delta” de la partícula (energía depositada en el paso, tiempo de vuelo del paso, etc.).
- La longitud del paso la calcula el programa de manera dinámica en función del rango de la partícula en el material (el usuario puede limitar este cálculo).
- En cada punto se conoce el volumen en el que se encuentra la partícula. Si un paso cruza la superficie de



Sistema de Unidades en Geant4

- Geant4 no tiene una unidad por defecto. Para introducir el valor de una magnitud física, una unidad debe estar multiplicando a un número .
  - Ejemplos:

    double width = 12.5*m;

    double density = 2.7*g/cm3;
  - Todas las unidades más comunes están incluidas.
  - El usuario puede definir unidades nuevas.

- Para obtener un valor en una cierta unidad debe dividirse la variable por esa unidad:

  G4cout << deltaE / MeV << “ MeV” << G4endl;

Resumen

Run → Geometría y Física fija!

Sistema de unidades de Geant4

Beam on!

Event

Event 0
Event 1
....
Event N

→Lista de vértices y partículas primarias
→Trayectorias (opcional)
→Hits en detectores
→Partículas secundarias
→Información adicional

Step

Step 0
Step 1
....
Step N'

→ Bloque básico para la simulación de eventos.
→Interacciones incluidas en la física definida con la geometría definida!

Estructura global de Geant4

Funcionamiento de la aplicación G4

- Inicialización
  - Construcción de los materiales y geometría
  - Construcción de partículas, procesos físicos y cálculo de tablas de secciones eficaces

- "Beam-On" → Inicia el *run*
  - Event Loop

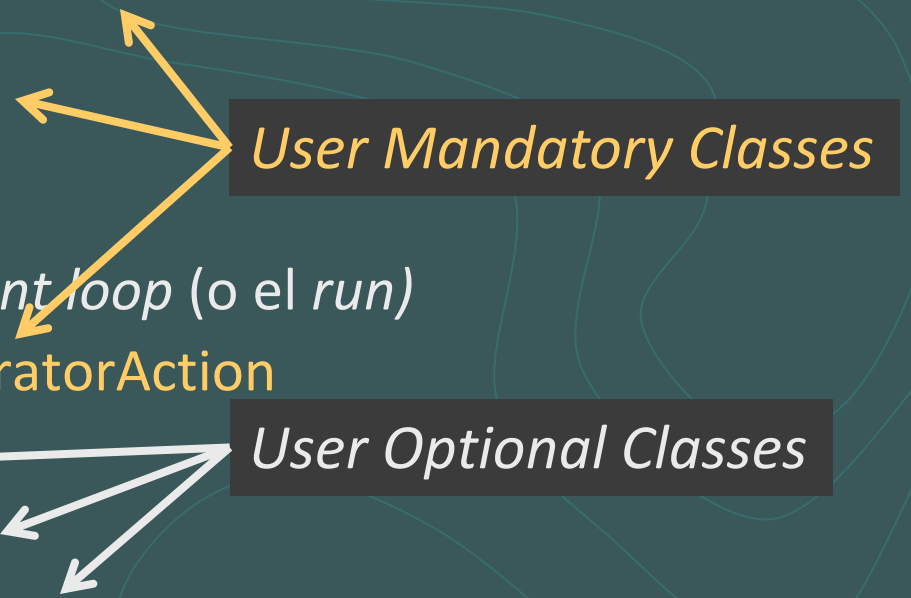
  → Se pueden lanzar más de un *run* con diferentes geometrías o tipo de partículas primarias…

User classes

- Clases que debe definir el usuario

- Initialization classes
  - Se invocan en la inicialización
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList

- Action classes
  - Se invocan durante el *event loop* (o el *run)*
    - G4VUserPrimaryGeneratorAction
    - G4UserRunAction
    - G4UserEventAction
    - G4UserSteppingAction



El programa principal

- Geant4 no proporciona un *main*().

- El usuario debe definir el *main()*:
  - Construir el *Run Manager (*puntero tipo *G4RunManager)*
  - Invocar las *user mandatory classes* desde el *Run Manager*
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
    - G4VUserPrimaryGeneratorAction

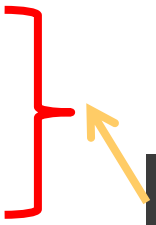
- En el *main()* se puede definir el VisManager, el tipo de (G)UI e invocar las *user optional classes*.

Ejemplo de programa principal

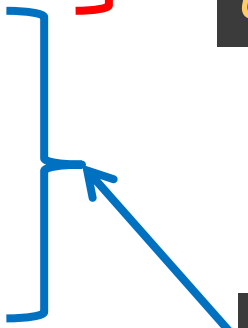
```
int main(int argc, char** argv){
// Construct the default run manager
G4RunManager* runManager = new G4RunManager;
// set mandatory initialization classes
 runManager->SetUserInitialization(new DetectorConstruction);
 runManager->SetUserInitialization(new PhysicsList);
 runManager->SetUserAction(new PrimaryGeneratorAction);

// set aditional user action classes
 RunAction* run = new RunAction;
 runManager->SetUserAction(run);
 EventAction* event = new EventAction(run);
 runManager->SetUserAction(event);
 runManager->SetUserAction(new SteppingAction(event));


// Initialize G4 kernel
 runManager->Initialize();
//get the pointer to the User Interface manager
 G4UImanager* UImanager = G4UImanager::GetUIpointer();
 delete runManager;
//Alternativa a la interfaz de usuario:
 runManager->BeamOn(10000000)//Lanza 10000000 eventos
 return 0;
}
```



User Mandatory Classes



User Optional Classes

Interfaz con el usuario (Esta práctica)

Descripción del detector

- Se deriva una clase propia que deriva de la clase base abstracta *G4VUserDetectorConstruction*

- En el método virtual *Construct*(),
  - Se construyen los materiales necesarios
  - Los volúmenes necesarios para describir la geometría del detector
  - Se define la geometría situando estos volúmenes en el lugar adecuado y con la orientación adecuada

- Opcionalmente se pueden definir los *visualization attributes* de los distintos elementos del detector.

Ejemplo Detector construction (I)

```
G4VPhysicalVolume* DetectorConstruction::Construct()
 // Na
G4Element* Na  = new G4Element(name="Sodium"  ,symbol="Na" , z=11., a=22.990*g/mole);
 // I
   G4Element* I = new G4Element(name="Iodine",symbol="I" , z=53., a=126.904*g/mole);

 pressure    = 3.e-18*pascal;
 temperature = 2.73*kelvin;
 density     = 1.e-25*g/cm3;

G4Material* Vacuum = new G4Material(name="Vacuum", z=1., a=1.01*g/mole,
                        density,kStateGas,temperature,pressure);

 // Sodium Iodide
density = 3.67*g/cm3;
G4Material* NaI = new G4Material(name="NaI", density, ncomponents=2);
NaI->AddElement(Na, natoms=1);
NaI->AddElement(I, natoms=1);
```

```
 //Otra opción→ NIST MANAGER ( https://apc.u-paris.fr/~franco/g4doxy/html/classG4NistManager.html)
// G4NistManager* manager = G4NistManager::Instance();
// G4Element* elm = manager->FindOrBuildElement("symb", G4bool iso);
// G4Element* elm = manager->FindOrBuildElement(G4int Z, G4bool iso);
// G4Material* mat = manager->FindOrBuildMaterial("name", G4bool iso);
....
```

Ejemplo Detector construction (II)

```
//El universo es oibligatorio de definir!!!
 G4double WorldSize= 200.*cm;

G4Box*    solidWorld = new G4Box("World",                                    //its name
                                 WorldSize/2,WorldSize/2,WorldSize/2);  //its size

G4LogicalVolume*    logicWorld = new G4LogicalVolume(solidWorld,        //its solid
                                                     Vacuum,            //its material
                                                     "World");          //its name

G4VPhysicalVolume *physWorld = new G4PVPlacement(0,                     //no rotation
                                                 G4ThreeVector(),       //at (0,0,0)
                                                 "World",               //its name
                                                 logicWorld,            //its logical volume
                                                 NULL,                  //its mother  volume ← World es 1 caso especial!
                                                 false,                 //no boolean operation
                                                 0);                    //copy number

 G4double diameter = 10.*cm, thickness = 20.*cm;
 G4Tubs *solidNaI = new G4Tubs("solidNaI", 0., diameter/2., thickness/2, 0., 360.*deg);
 G4LogicalVolume *logicNaI = new G4LogicalVolume(solidNaI, NaI, "logicNaI");
 new G4PVPlacement(0,G4ThreeVector(0.*cm,0.*cm,20.*cm),"physNaI",logicNaI,physWorld,false,0);
....

//Invisible vacuum volumes
logicWorld->SetVisAttributes (G4VisAttributes::Invisible);
//Orange color for NaI
G4VisAttributes* Att1= new G4VisAttributes(G4Colour(0.42,0.34,0.239));
Att1->SetVisibility(true);
logicNaI->SetVisAttributes(Att1);
}
```

Selección de los procesos físicos

- Geant4 no proporciona partículas ni procesos físicos por defecto
  - Incluso para el proceso *transportation* hace falta una llamada explícita en el programa
- Se deriva una clase propia que deriva de la clase base abstracta *G4VUserPhysicsList*
  - Se definen todas las partículas necesarias
  - Se definen todos los procesos físicos necesarios y se asignan a cada partícula previamente definida
  - Se definen los '*cuts*'

Ejemplo de lista de física

```
PhysicsList::PhysicsList():  G4VUserPhysicsList(){
                defaultCutValue = 1.*um;
                SetVerboseLevel(0);
//electromagnetic physics
                emPhysicsList = new G4EmStandardPhysics_option3();
}
void PhysicsList::ConstructParticle(){
// gamma, e+, e- ...
                emPhysicsList -> ConstructParticle();
}
void PhysicsList::ConstructProcess(){
                AddTransportation();
                emPhysicsList -> ConstructProcess();
}
```

Alternativa → G4VModularPhysicsList (Ready to use) https://geant4.web.cern.ch/node/155

```
void PhysicsList::PhysicsList():G4VUserPhysicsList()
{
// Hadronic physics
RegisterPhysics(new G4HadronElasticPhysics ());
RegisterPhysics(new G4HadronPhysicsFTFP_BERT_TRV());
// EM physics
RegisterPhysics(new G4EmStandardPhysics());
}
```

Generación de los primarios

- Se deriva una clase propia que deriva de la clase base abstracta *G4VUserPrimaryGeneratorAction*
- Se pasa un objeto *G4Event* a un generador de primarios particular que generará los vértices y partículas primarias
- Geant4 proporciona varios generadores diferentes
  - *G4ParticleGun*
  - *G4HEPEvtInterface*, *G4HepMCInterface*
  - *G4GeneralParticleSource*

Ejemplo de PrimaryEventGenerator (I)

```
PrimaryGeneratorAction::PrimaryGeneratorAction(){
// Default values
particleGun = new G4GeneralParticleSource();  ← Controlado por un macro externo! (Esta práctica)
particleGun->SetCurrentSourceIntensity (1);
particleGun->SetParticlePosition(G4ThreeVector());
particleGun->SetParticleCharge(0.);
}
PrimaryGeneratorAction::~PrimaryGeneratorAction(){  delete particleGun;}

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

```
// Ejemplo que vamos a usar en la práctica (source.mac)
/gps/particle gamma
/gps/particle e-
/gps/number 1
/gps/pos/type Plane
/gps/pos/shape Circle
/gps/pos/radius 0.5 mm
/gps/pos/centre  0 0 0  mm
/gps/energy 800 keV
/gps/ang/type iso
```

(http://hurel.hanyang.ac.kr/Geant4/Geant4_GPS/reat.space.qinetiq.com/gps/examples/examples.html)

Ejemplo de PrimaryEventGenerator (II)

```
Example of user PrimaryGeneratorAction using G4ParticleGun !
void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent){
G4ParticleDefinition* particle;
G4int i = (int)(5.*G4UniformRand());
switch(i){
case 0: particle = positron; break;
case 1:
...
}
void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent){
                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);
}
```

//GeneratePrimaryVertex(anEvent) Tan flexible como se quiera, se pueden incluir procesos muy complejos como por ejemplo eventos de fisión:

→ Fragmentos de fisión (2-3)

→ Neutrones (2-3)

→ Gammas de fisión (6-8)

→Correlaciones entre todos los observables (Correlaciones angulares, boost cinemático, conservación energía…)

Corre a cuenta del usuario que lo crea!

User action classes 'opcionales'

- Todas las *user action classes*, deben construirse en el *main*() a través del Run Manager.

- G4UserRunAction
  - BeginOfRunAction(const G4Run*)
    - Define histogramas, trees, ficheros de salida…
  - EndOfRunAction(const G4Run*)
    - Escribe histogramas, trees… en disco

- G4UserEventAction
  - BeginOfEventAction(const G4Event*)
    - Inicializa el event (p. ej. pone a cero algunas variables)
  - EndOfEventAction(const G4Event*)
    - Analiza el evento (p.ej. rellena un histograma)

Ejemplo de run y event actions (I)

```
void RunAction::BeginOfRunAction(const G4Run*){
   outputFile  = new TFile("./data/output.root", "recreate");
   h_1 = new TH1F("h_1", "Crystal energy", 4096, 0., 10000.);
   h_2 = new TH1F("h_2", "Plastic energy", 4096, 0., 10000.);
   h_3 = new TH1F("h_3", "Crystal energy beta gated", 4096, 0., 10000.);}

void RunAction::EndOfRunAction(const G4Run*){
h_1 -> Write();
delete h_1;
h_2 -> Write();
delete h_2;
h_3 -> Write();
delete h_3;
outputFile->Close();
delete outputFile;
}

void RunAction::fillHistograms(G4double ener, G4double ener2){
   h_1 -> Fill(ener);
   h_2 -> Fill(ener2);
   if(ener2 > 5.) h_3->Fill(ener);
}
```

Ejemplo de run y event actions (I)

```
EventAction::EventAction(RunAction* RunAct):runAction(RunAct){}

EventAction::~EventAction(){}

void EventAction::BeginOfEventAction(const G4Event*){
   energy = 0.;
   energy2 = 0.;
}
void EventAction::EndOfEventAction(const G4Event* evt){
                    // fill histograms          runAction-> fillHistograms(energy, energy2);
}
```

User action classes 'opcionales'

- G4UserSteppingAction
  - UserSteppingAction(const G4Step*)
    - Guarda información de cada paso (p. ej. energía depositada, proceso físico…)

Ejemplo de Stepping action

```
SteppingAction::SteppingAction(EventAction* EvAct):eventAction(EvAct){ }
SteppingAction::~SteppingAction(){ }

void SteppingAction::UserSteppingAction(const G4Step* aStep){
   // collect the energy deposited in the sensitive volumen
   const G4String currentPhysicalName    = aStep->GetPreStepPoint()->GetPhysicalVolume()->GetName();

   const G4int currentCopyNumber= aStep->GetPreStepPoint()->GetTouchableHandle()->GetCopyNumber();

   G4String name=    aStep->GetTrack()->GetDefinition()->GetParticleName();
      G4double EdepStep = aStep->GetTotalEnergyDeposit() / keV;
      if(currentPhysicalName == "physNaI"){
                  eventAction->energy = eventAction->energy + EdepStep;
   }
}
```

//Alternativa para Stepping Action → Sensitive detectors

https://geant4.web.cern.ch/sites/default/files/geant4/collaboration/working_groups/geometry/training/D3-Sensitivity_Field.pdf
https://indico.cern.ch/event/294651/sessions/55918/attachments/552022/760640/UserActions.pdf

Visualización

- Se deriva una clase propia a partir de la clase base abstracta *G4VVisManager*

- Geant4 proporciona interfaces a distintos drivers
  - DAWN -- Fukui renderer
  - WIRED
  - RayTracer -- Ray tracing by Geant4 tracking
  - OPACS
  - OpenGL
  - OpenInventor
  - VRML
  - QT -- Interactivo <-- (Esta práctica)

Más información:

- Geant4 está continuamente desarrollandose (v11.0)
  - Nuevas funcionalidades, modelos, física… ☺
  - Versiones anteriores dejan de ser compatibles ☹
    - Nuevos estandares para C++: C++14/C++17/C++21
    - El código evoluciona!→ Mejoras en rendimiento
- https://geant4.web.cern.ch/
  - https://geant4-userdoc.web.cern.ch/UsersGuides/InstallationGuide/html/index.html
  - https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/html/index.html
- Consejo → Si vais a usar GEANT4, usad linux:
  - https://ubuntu.com/download